HYPERPARAMETER OPTIMIZATION FOR NEURAL MACHINE TRANSLATION SYSTEMS

by

Xuan Zhang

A dissertation submitted to The Johns Hopkins University
in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland
August 2024

Abstract

Machine translation, a sequence-to-sequence task, involves translating text from one language to another. Currently, transformer-based systems dominate the field of neural machine translation (NMT). To optimize these systems, various critical decisions regarding architecture design and training processes must be made—these decisions are the hyperparameters of the system. Typically, these hyperparameters are set before training begins and remain unchanged until convergence. Traditionally, they are tuned manually based on intuition, heuristics, previous studies, or default settings provided in open-source frameworks. This approach often leads to suboptimal exploration of the hyperparameter space, which can cause exaggerated performance differences and potentially misleading conclusions. Despite the proliferation of hyperparameter optimization (HPO) methods under the umbrella of Automated Machine Learning (AutoML), their effectiveness in NMT has not been thoroughly evaluated, primarily due to the significant computational demands of NMT models and their vast hyperparameter search spaces. This challenge is further complicated by the need to optimize multiple objectives simultaneously, such as translation accuracy and decoding speed.

This thesis addresses these challenges by conducting a comprehensive study of HPO specifically within the context of NMT. First, we introduce a benchmark dataset employing a "table-lookup" based benchmarking procedure, designed to promote reproducible research in HPO for NMT. Second, we propose a novel HPO algorithm using graph-based optimization, which flexibly incorporates prior knowledge about hyperparameters. Third, we develop a post-hoc interpretation framework to better understand the significance and interrelationships of individual hyperparameters. Fourth, we evaluate the efficacy of a multi-fidelity HPO method, successive halving, and propose best practices for its application in NMT and large language models. Finally, this work includes the creation of an HPO toolkit tailored for NMT research, designed to streamline the experimental process and allow researchers to concentrate on innovation instead of the mundane.

Primary Reader and Advisor: Kevin Duh

Secondary Readers: Philipp Koehn & Kenton Murray

Acknowledgments

Committee

I cannot express enough gratitude to my advisor, Kevin Duh. I find it hard for me to put into words how exceptional both an advisor and a person Kevin is, to whom I owe an immense debt. It was Kevin who, during a particularly frustrating job search towards the end of my Master's program, offered me a research assistant position despite my limited research experience, and a year later, extended an offer to a Ph.D. program. Without Kevin, neither this dissertation nor the person I have become would exist. Kevin is an excellent advisor and researcher, and a joy to collaborate with, someone I can always rely on. He sets a strong example and fosters growth in all aspects of my academic abilities. He continuously encouraged me, trusted in my abilities, and appreciated my work, which bolstered my confidence and convinced me to believe in myself. If not for Kevin's full support, I would not have had the opportunity to work on the sign language projects that have opened so many doors for me. I feel that Kevin is a wise person, as I learn a great deal from him, not only from his explicit advice but also from the unspoken lessons in his actions. I feel that Kevin sees the success of his students of his own–he consistently goes beyond what I expect, takes more care of my needs than I do myself, actively promotes me and facilitates connections with others whenever opportunities arise, and rejoices in my smallest achievements. I feel deeply thankful, fortunate, and proud to have Kevin as my advisor.

I would also like to extend my gratitude to other members of my dissertation committee, Philipp Koehn and Kenton Murray. I had the privilege of taking the machine translation class from Philipp and am one of the many beneficiaries of his two books: Statistical Machine Translation and Neural Machine Translation. Additionally, I have been a participant in the machine translation reading group organized by Philipp, from whom I have gained substantial knowledge in the field. I am grateful for the opportunity to collaborate with Philipp, whose insights have been invaluable. I first met Kenton at the SCALE workshop in 2018, before I began my Ph.D. His dissertation closely aligns with mine and has been a significant influence. Kenton strikes me as exceptionally cool and smart. He is also very approachable and has been a crucial contributor to the machine translation reading group, consistently offering profound insights. I am also grateful to Philipp, Kenton, Amitabh Basu, and Najim Dehak for serving on my GBO committee. Their valuable suggestions on my dissertation proposal shaped the development of this dissertation.

CLSP Faculty

The professors at CLSP set exemplary standards, create a supportive environment, and offer immense support for students to achieve their goals.

Sanjeev Khudanpur, who takes care of CLSP, is the intersection and center of its social network. He connects people by organizing and encouraging participation in social events, remembers every student's name, and is aware of their research. He promotes, praises, and includes me in important events. I thank him for encouraging my sign language research and including me in the sign language project at JSALT 2024.

I served as the teaching assistant for the machine learning class in the Fall 2022, taught jointly by Mark Dredze and Anqi Liu. Both are exceptional and dedicated teachers who deliver excellent lectures with up-to-date, well-illustrated, and structured materials. They value student feedback and stayed up late to work together with me on preparing assignments. I am grateful for their recognition of my contributions, which made it a memorable and enjoyable experience. I also thank Mark for his unwavering recommendation for the AI2AI fellowship.

Throughout my Ph.D., I have been part of the CLSP student seminar committee, with Najim Dehak and Laureano Moro-Velazquez serving as faculty coordinators consecutively. I appreciate their help in facilitating the process. Special thanks to Laureano for his exceptional responsibility, responsiveness, and charismatic leadership. I also thank Kevin and Mahsa Yarmohammadi, who served as coordinators of external seminars and collaborated with me to organize seminars with timely communication and coordination.

I am deeply thankful to Jason Eisner, who was my advisor during my Master's program. He has been a role model for NLP researchers, exemplifying the conscientiousness and standards of NLP research. If not for Jason, who shared my resume with the CLSP faculty, I would never have had the opportunity to work with Kevin.

I would also like to thank Leibny Paola Garcia for organizing JSALT 2024 and taking care of the students at CLSP. She is the guardian angel of CLSP and has always been kind to me. She often asks critical research questions as well. Additionally, I thank Jesús Villalba for his kindness and consistent support alongside Paola.

Supervisors

I am fortunate to have worked under the supervision of several esteemed senior researchers. I extend my gratitude to Marine Carpuat at University of Maryland for her guidance on my first two publications. I also thank Xiaodong Liu, Hoifung Poon, and Hao Zhu for their mentorship during my internship at Microsoft Research in the summer of 2020. Additionally, I appreciate Fangyun Wei and Yu Wu for their inclusion and guidance in the sign language research at Microsoft Research Asia in 2022. I am thankful to Christos Christodoulopoulos and Venkatesh Ravichandran at Amazon for serving as liaisons of the AI2AI fellowship, providing valuable advice, and facilitating connections within the industry. I thank Murat Saraclar from Bogazici University for our exciting discussions about sign languages.

Lastly, I want to give my special thanks to Marek Hrúz at the University of West

Bohemia. Marek leads the sign language project team at JSALT 2024, of which I am a proud member. His enthusiasm for research and openness to discussion make collaboration stimulating. Marek is frank, consistently getting straight to the point. Beyond his professional prowess, he is a humorous friend and a charismatic leader who fosters a joyful team dynamic. I thoroughly enjoy working with him and am grateful for his tolerance of my bluntness during our heated discussions. I defended my dissertation during the JSALT workshop, and I want to thank Marek for treating it as a special and important event and taking all the Sign Language LLM team members to come to my defense to support me. They made it an even more memorable day.

Sign Language

During my Ph.D., I have been conducting research on sign language understanding with the support of Kevin and the AI2AI fellowship. I am grateful to Matthew Sampson, from whom I have been learning American Sign Language (ASL) for two years. I appreciate our discussions about my research and his insights into ASL and deaf culture. His patience with my slow and struggling signing during office hours, and his insistence on signing slowly with me instead of typing, have helped improve my ASL fluency. I also want to thank Annemarie Kocab for inviting me to a memorable hour-long discussion about sign language research.

Collaborators

ACKNOWLEDGMENTS

One of the most rewarding experiences throughout my Ph.D. has been the opportunity to collaborate with individuals from diverse backgrounds, from whom I have learned immensely. These collaborators include Kevin Duh, Philipp Koehn, Kenton Murray, Marine Carpuat, Marek Hruz, Paul McNamee, Hirofumi Inaguma, Gaurav Kumar, Huda Khayrallah, Jeremy Gwinnup, Marianna Martindale, Pamela Shapiro, Brian Thompson, Rebecca Knowles, Jason Naradowsky, Kiron Deb, Navid Rajabi, Murat Saraclar, Ivan Gruber, Jiri Mayer, Dominik Machacek, Tomas Zelezny, Jakub Straka, Vaclav Javorek, Ondrej Valach, Karahan Sahin, Shester Gueuwou, Florian Metze, Lale Akarun, Alessa Carbo, and Richard Fung.

JHU Staff

I must express my gratitude to the outstanding administrative staff at JHU. Special thanks go to Ruth Scally, Lauren Meek, and Kim Franklin for their exceptional work and support. I am also thankful to Dan Meade and Joe McKnight for their excellent management of the computing cluster and their prompt and helpful responses to grid-related inquiries.

Colleagues

I am grateful to have shared this journey with my wonderful colleagues at CLSP.

I owe special thanks to my senior colleagues—Hongyuan Mei, Huda Khayrallah, Shuoyang Ding, Rebecca Knowles, Brian Thompson, and Gaurav Kumar—who exemplify what it means to be outstanding Ph.D. students. Hongyuan is always deeply engaged in his research and has generously offered his support. Huda, the dependable and inclusive ”big sister” of our machine translation group, has been exceptionally supportive. Shuoyang, never disturbed, always available for help or a chat. I will never forget the extensive email Rebecca sent me, detailing a long list of every work of sign language research she knew. Both Brian and Gaurav are not only excellent researchers but also gracious and supportive to younger scholars like I once was.

Among my colleagues, some have become close friends. Jinyi Yang and Suzanna Sia, my Asian sisters at CLSP, have been confidantes with whom I can safely share secrets. Desh Raj, a cool guy, manages his work gracefully while keeping his girlfriend and friends happy. I only wish I had met Hanjie Chen sooner, as we share so much in common. Zili Huang, my longest-known friend from the Master’s program, is one of the most honest and sincere people I’ve ever met. I’m grateful to Ruizhe Huang for driving me and Zili to supermarkets and the engaging conversations we have shared.

Special thanks to my lab mates in Hackerman 323 for keeping my spirits high and making our workspace a delight. Our office conversations, always filled with laughter and an endless supply of snacks, are a highlight of my day. Samik Sadhu is wonderfully hospitable; I’m thankful for the invitations to his delightful parties. Henry Li, a knowledgeable and jovial gentleman, who is passionate for extracurriculars like soccer, chess, and languages, which validates my own interests in Kendo and language learning. Zexin Cai is considerate and a great listener, encouraging me to share many random thoughts. Yahan Li, though much younger, is incredibly hardworking and a reliable confidante for life advice.

I also appreciate Elias Stengel-Eskin, Ke Li, Desh, Shuoyang, Huda, Shijie Wu, Zhuowan Li, Elizabeth Salesky, and others for their invaluable help during my job search.

My heartfelt thanks to Elina Baral for her trust and the enriching conversations we have shared. I also treasure the interactions with Saurabhchand Bhati, Dongji Gao, Arya McCarthy, Rachel Wicks, Neha Verma, Sabrina Mielke, Tianjian Cao, Jack Zhang, Guanghui Qin, Brian Lu, Marc Marone, Jiefu Ou, Kaiser Sun, Cihan Xiao, Qinwan Rabbani, Carlos Aguirre, Alexandra DeLucia, Niyati Bafna, Shuo Sun, and Mitchell Gordon.

Lastly, I am grateful to Ruizhe Huang, Elina Baral, Haoran Xu, and Helin Wang for their collaboration with me on the CLSP student seminar committee. I also thank all the CLSP Ph.D. students who presented at the CLSP seminar during my Ph.D. for their support to our committee's efforts.

JHU Kendo Club

I began practicing Kendo during my second year of PhD studies, and it quickly became an integral and uplifting part of my life. I am deeply thankful to everyone at the JHU Kendo Club for enriching this experience.

Special thanks to Sean Kim sensei for his professional expertise and exceptional

dedication to his students. I am equally grateful to my Kendo "senpai," Yuzo Ishikawa, Qinyang Du, and Justin Chan, who have shared their deep knowledge and provided patient, meticulous instruction. I also appreciate my club mates—Kayla Lin, Fernando José Torres, Emily Nakayama, Chloe Liang, Mufasa Cruz, Brian Zhou, Nico Guerra, Kangxin Wang, Evan Li, and Calista Huang—for the numerous practice sessions, and for joining me at tournaments and tests. It is through the camaraderie and challenges shared with the members of the JHU Kendo Club that I have come to understand the meaning of 交剣知愛 (crossing swords, knowing love).

Friends

I am immensely grateful to my friends for their support and kindness. I thank Zheyun Shen for inviting me to stay at her place in New Jersey, where we shared some wonderful moments together. I am also grateful to Morgan Flotteron for welcoming me into her home in Long Beach, where I spent one of the most memorable July 4th holidays with her family, and introducing me to Grace. My thanks extend to Yujia Xiao, who accompanied me from the Beijing train station to the airport, just to ensure we had enough time for a goodbye. I appreciate Shirish Singh and Meng Xie for their generous and genuine support. Special thanks to Brian and Emalie for their continuous and heartwarming support as well. Last but not least, I feel grateful to become friends with Tomas Zelezny at the very end of my Ph.D. program. Always with a bright smile on his face, Tomas brought me so much joy. He also introduced

me to metal music, which has become my new energy boost.

Family

Last but certainly not least, I want to thank my family for their endless support and love. My grandmother, along with my wonderful aunts, Jixian Wang and Ruixian Wang, and uncles, Fuxi Ji and Chaojie Duan, have always believed in me unconditionally. My little cousin, Wenxi Duan, looks up to me as her role model and her kindness always warms my heart.

My parents, Fengxian Wang, and Kaidong Zhang, are the very reason I am here today. I am grateful to them for respecting my choices and giving me the freedom to venture far from home. Their unconditional support and love have been my anchor, and they have always encouraged me to find my own path without worrying about them.

Dedication

This dissertation is dedicated to my mother, Fengxian Wang, and my father, Kaidong Zhang, who have given me the gift of this wonderful life and their unconditional love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Machine translation is the task of automatically translating from one language to another by machines. It enhances global communication and broadens access to information. It aims toward a future where real-time conversations can occur between speakers of different languages without the need for human translators and where individuals can read books in any language, gaining insights into various cultures. This ability could potentially reduce biases and conflicts by deepening our understanding of people from different countries and backgrounds.

The history of translation demonstrates its significant benefits to humanity. For instance, translating the Bible made its teachings accessible to a wider audience. Similarly, during the late 19th century in China, the Self-Strengthening Movement led to the systematic translation of Western scientific and technical literature, playing an essential role in modernizing China's military, industry, and education systems.

However, translation errors can have severe consequences. A notable example occurred during World War II. Japanese Premier Kantaro Suzuki used the term "mokusatsu" to describe his government's response to the Potsdam Declaration, which could mean "to ignore" or "treat with silent contempt." However, it was translated into English as "not worthy of comment," suggesting a dismissive attitude. This misinterpretation was taken by the Allies as a rejection of the surrender terms, influencing the decision to use atomic bombs on Hiroshima and Nagasaki.

In contemporary research, the focus is on developing machine translation systems that are capable of accurate and fast translations across any language and domain. These systems, primarily based on deep neural networks, feature complex architectures with millions or billions of parameters. Effective training of these systems requires careful configuration of the model and training procedures, a process known as hyperparameter optimization.

Currently, hyperparameter tuning is often conducted manually or through basic heuristics. Given that modern NMT systems demand extensive computational resources and involve numerous hyperparameters, this manual approach is inefficient and unreliable. This dissertation emphasizes a systematic approach to HPO, addressing the gap in research specifically for NMT systems.

The dissertation is structured to cover several key aspects of HPO for NMT:

- **Benchmark:** Chapter 3 proposes the first HPO benchmark specifically for NMT tasks to enable reproducible and efficient evaluation of different HPO methods.
- **Algorithm:** Chapter 4 introduces a new HPO method that incorporates prior knowledge of the search space, providing performance comparable to existing methods.
- **Interpretation:** Chapter 5 develops a framework to understand the importance and interaction of hyperparameters in NMT systems.
- **Best practices:** Chapter 6 evaluates the robustness of an existing multi-fidelity HPO method and proposes the best practices.

- **Toolkit:** Chapter 7 creates an HPO toolkit tailored for NMT, designed to streamline the search and training process and reduce the burden of manual tuning.

HPO is key to the effectiveness of NMT systems. The field has recognized its importance and has included a term for documenting the HPO process under the "Responsible NLP Checklist"[1] required for all paper submissions.

1.2 Publications

This dissertation includes work that has been featured in the following publications.

- Chapter 3 and Chapter 4: Zhang and Duh (2020).
- Chapter 5: Deb et al. (2022).
- Chapter 6: Zhang and Duh (2024).
- Chapter 7: Zhang et al. (2023b).

The author also has additional publications not included in this dissertation.

- **Machine translation:** Inaguma et al. (2018), Zhang et al. (2018), Zhang et al. (2019b), Thompson et al. (2019), Naradowksy et al. (2020), Zhang et al. (2023c).
- **Sign language translation:** Zhang and Duh (2021), Zhang and Duh (2023a), Zhang and Duh (2023b), Zhang et al. (2023e).

[1] `https://aclrollingreview.org/responsibleNLPresearch/`

Chapter 2

Background

2.1 Neural Machine Translation

Machine translation serves a multitude of applications, significantly enhancing communication across different languages and reducing the reliance on human translators in various contexts such as travel, education, real-time communication (both online and offline), social media, e-commerce, business, healthcare, and video subtitling. It also facilitates cross-lingual information access through search engines and the broader internet. In scenarios where precise translations are crucial, machine translation can also support human translators by accelerating their workflow.

The history of machine translation dates back to 1947, during the early stages of the Cold War, a time when the first digital electronic computers were employed primarily for code-breaking. Subsequently, the field explored numerous rule-based approaches, such as direct translation, transfer-based translation, and interlingual translation. The concept of example-based machine translation, which leverages existing translated examples to craft new translations, was introduced in 1984. Since the late 1980s, the advent of parallel corpora and open-source software marked a significant shift towards data-driven methods. Statistical machine translation—including word-based, phrase-based (Koehn et al., 2003), and syntax-based models—originated from IBM Research labs and dominated the field until the 2010s.

Neural methods were first integrated into machine translation by incorporating neural language models into existing statistical frameworks (Schwenk et al., 2006). The evolution to purely neural approaches began with convolutional methods

(Kalchbrenner and Blunsom, 2013). The RNN encoder-decoder framework, which further advanced the field, was introduced by Cho et al. (2014) and Sutskever et al. (2014). This framework was subsequently enhanced by Bahdanau et al. (2015) through the integration of the attention mechanism. The introduction of the Transformer architecture by Vaswani et al. (2017), which relies solely on attention mechanisms and dispenses with both recurrence and convolutions, marked a significant milestone. Since its introduction, the Transformer has set new benchmarks on numerous sequence transduction tasks, including machine translation, and continues to dominate the field of natural language processing as of the writing of this thesis.

2.1.1 Evaluation

The primary goals of machine translation are to achieve adequacy—preserving the original text's meaning—and fluency—ensuring it reads naturally in the target language (Koehn, 2020). Translation quality can be assessed either through human evaluation or automatic metrics.

Human assessment Human evaluations typically involve asking evaluators to score translations on a graded scale or to rank outputs from several systems. The major drawbacks of human assessment include its time-consuming nature and the high cost of human effort. Variability in annotators' quality and grading criteria can also lead to unreliable results. Furthermore, for each minor modification of the system's configuration, a complete re-evaluation is required, which can be inefficient and costly.

Automatic evaluation Automatic evaluation generally involves comparing the system-generated translation to a human-generated reference translation. This approach is preferred in scenarios requiring frequent evaluations due to its efficiency. Lee et al. (2023) and Chauhan and Daniel (2023) have provided comprehensive surveys on machine translation evaluation metrics and their taxonomies, as illustrated in Figure 2.1. In the following, we introduce some representative automatic machine translation evaluation metrics.

BLEU BLEU (Papineni et al., 2002) measures the overlap of n-grams between the reference and the system output, applying a penalty for excessively short translations. It is defined as:

$$\text{BLEU} = \text{brevity-penalty} \times \exp \sum_{i=1}^{4} \log \frac{\text{matching } i\text{-grams}}{\text{total } i\text{-grams in machine translation}} \tag{2.1}$$

where

$$\text{brevity-penalty} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r, \end{cases}$$

c is the output length and r is the reference length.

The SacreBLEU (Post, 2018) implementation of BLEU standardizes tokenization, enhancing reproducibility.

METEOR METEOR (Banerjee and Lavie, 2005) addresses BLEU's limitation by considering synonyms and stemming. It matches words first by their surface forms,

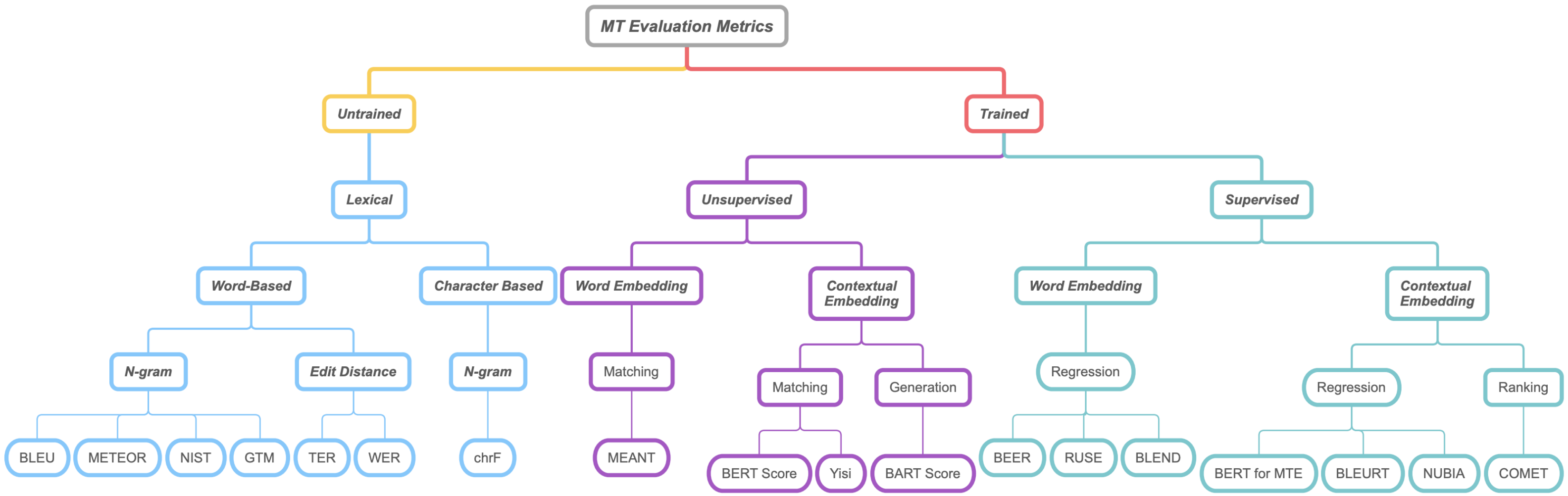


Figure 2.1: Taxonomy of automatic machine translation evaluation metrics proposed by Lee et al. (2023). This figure is from Lee et al. (2023)

then by stems, and finally by semantic classes, though it is computationally intensive.

TER Translation error rate (TER, Snover et al., 2006) is calculated by considering the operations of addition, deletion, substitution, and shifting of words, where moving any sequence counts as a single error. It is computed similarly to the word error rate and can be interpreted more intuitively than BLEU and METEOR.

chrF Character F score (chrF, Popović, 2015) evaluates translation quality based on character-level precision and recall, offering a different perspective that can be especially relevant for languages where character-level nuances are crucial.

BERTscore Utilizing the contextual embeddings from BERT, Zhang et al. (2019a) developed BERTscore, which assesses translation quality by measuring the cosine similarity between embeddings of words in the machine-generated and reference texts. This approach focuses on semantic preservation, moving beyond mere form matching to assess the meaningfulness of translations.

COMET COMET (Rei et al., 2020) utilizes pretrained neural models to evaluate translations by comparing not just the translated text and the reference but also considering the original source text. This method involves embedding sentences to capture their semantic content and training the model to align closely with human judgment, making it a robust tool for measuring semantic accuracy in translations.

2.1.2 Transformer

The Transformer model, introduced by Vaswani et al. (2017), significantly advanced the field of sequence-to-sequence learning, especially in machine translation. This architecture departs from traditional recurrent models by utilizing an encoder-decoder structure with an extensive use of attention mechanisms, enhancing performance and training efficiency. The model architecture is depicted in Figure 2.2.

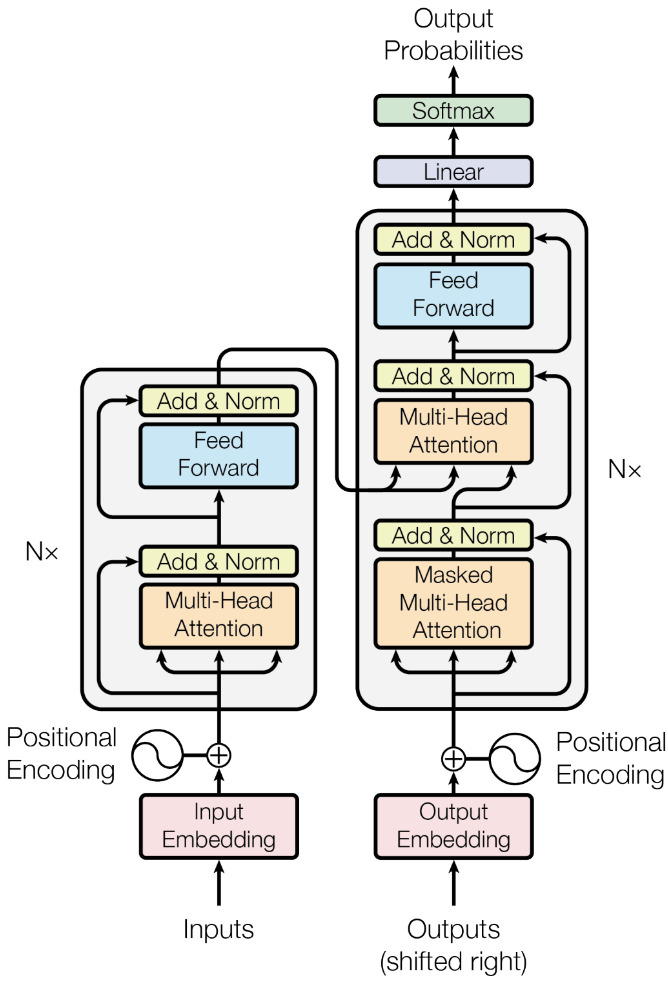


Figure 2.2: The Transformer model architecture. This figure is from Vaswani et al. (2017).

Encoder-decoder architecture In machine translation systems, the encoder processes the input tokens to encode the semantic and syntactic information into a set of sentence embeddings or hidden states. These embeddings are then utilized by the decoder, which generates the target sequence by conditioning on these hidden states. The encoder must effectively capture all pertinent information from the source sentence without losing details, regardless of sentence length. The decoder, tasked with output generation, needs to determine which parts of the input to focus on at each step of the translation.

Encoder The Transformer encoder consists of N identical layers, each featuring a multi-head self-attention mechanism and a fully connected feed-forward network. Residual connections and layer normalization are applied after each sub-layer to facilitate training deep networks. The input word embeddings are augmented with positional embeddings to provide the model with information about the order of the words.

Decoder The decoder also comprises N identical layers but includes an additional sub-layer for encoder-decoder attention, allowing each position in the decoder to attend to all positions in the encoder output. Like the encoder, the decoder employs multi-head attention and feed-forward networks, supplemented with residual connections and layer normalization. The decoder is inherently auto-regressive, using the output from previous time steps as additional input to generate subsequent elements in the sequence.

Attention mechanisms At the heart of the Transformer model is the attention mechanism, which allows the model to dynamically focus on different parts of the input sequence as it generates each word in the output sequence. Specifically:

- *Scaled dot-product attention:* This form of attention computes the attention scores by scaling the dot products of the queries with the keys. The attention scores determine how much each element of the input sequence contributes to each element in the output sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.2}$$

  where Q, K, and V are matrices packed with the query, key, and value vectors, respectively, and d_k is the dimensionality of the queries and keys.

- *Multi-head attention:* This extends the model' s capacity to focus on different positions by running parallel attention processes. This setup allows the Transformer to capture information from different representation subspaces at different positions in the sequence. Each head performs its own attention operation with distinct learned weights, and the outputs are concatenated and linearly transformed.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W^O, \tag{2.3}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \tag{2.4}$$

and W are the parameter matrices for each head.

There are three types of attention in the Transformer model:

1. *Encoder self-attention*: Each position in the encoder can attend to all positions in the previous layer of the encoder.

2. *Decoder self-attention*: Positions in the decoder can only attend to earlier positions in the decoder, preventing future in formation leakage.

3. *Encoder-decoder attention*: Each position in the decoder attends to all positions in the encoder, integrating the encoded information with the developing output.

Theses attention mechanisms enable the Transformer to model dependencies regardless of distance in the input sequence, providing both computational efficiency and interpretability.

2.1.3 Training

For NMT models, the process of translating a source sentence $\mathbf{x}$ into a target sentence $\boldsymbol{y}$ involves modeling the conditional probability of generating each word in the target sequence given the source sentence and all previously generated target words. This conditional probability is mathematically expressed as:

$$p(\boldsymbol{y} \mid \mathbf{x}; \theta) = \prod_{j=1}^{J} p(y_j \mid y_{<j}, x; \theta), \tag{2.5}$$

where θ represents the model parameters, J is the length of the target sentence, y_j is the j-th word in the target sentence, and $y_{<j}$ is the sequence of words before y_j.

The training objective for an NMT system is to maximize the likelihood of the correct translation of a given source sentence, which translates into minimizing the negative log-likelihood (NLL) of the target sentence conditioned on the source sentence. Given a parallel corpus consisting of N pairs of aligned source and target sentences $(\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^{N}$, the training objective can be formulated as:

$$\mathcal{L}_{NLL}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log p(\boldsymbol{y}^i \mid \mathbf{x}^i; \theta). \tag{2.6}$$

The objective function is equivalent to the cross-entropy loss between the predicted probability distributions of the target words and the actual distributions in the training data.

NMT models are typically trained using stochastic gradient descent (SGD) or one of its variants, where training data are grouped into mini-batches. This approach not only utilizes computational resources more efficiently but also helps in stabilizing the gradients during training. The models are typically trained on Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs), and the training duration can vary from several hours to weeks, depending on factors such as the size of the dataset, the complexity of the model, and the computational power available.

To prevent overfitting and to ensure that the model generalizes well on unseen data, training often incorporates an early stopping mechanism. This involves continuously monitoring the model's performance on a validation set during training and stopping the training process when there is no noticeable improvement in performance, such as a decrease in validation loss, over a number of epochs. In addition to early stopping, other regularization techniques like dropout, label smoothing, or L2 regularization might be employed to improve the robustness and generalization of the model.

2.1.4 Inference

In machine translation, the process of generating a translated sentence from a source sentence is known as inference or decoding. For models like the Transformer, decoding is executed auto-regressively, meaning the model generates one token at a time, with each token's generation depending on the tokens that were previously generated.

Greedy search During decoding, the model predicts a probability distribution over the vocabulary for the next word in sequence. In greedy search, the word with the highest probability is selected at each time step and used to predict the next word. This approach is computationally efficient and straightforward but tends to yield repetitive and less varied text. Because it always chooses the most likely next word, the greedy search can lack diversity in the output, leading to deterministic and potentially uncreative translations.

Beam search As an enhancement over greedy search, beam search (Algorithm 1) maintains not just one but the top-N most probable sequences at each decoding step. This process involves expanding each of the top-N sequences by considering all possible next words for each sequence and calculating the cumulative probabilities. Only the sequences with the highest overall probabilities are retained for further expansion. This iterative process continues until each sequence reaches a predefined maximum length or an end-of-sequence token is generated. Finally, the sequence with the highest cumulative probability is selected as the output. Beam search is noted for its effectiveness in producing more coherent and contextually appropriate translations than greedy search.

At each step in beam search, a “beam” of width N is maintained, meaning the model explores multiple high-potential sequences simultaneously. This approach significantly increases the likelihood of generating high-quality text, as it balances between breadth and depth in the search strategy. By maintaining a limited set of potential sequences rather than exploring all possible sequences, beam search manages to be more computationally efficient than exhaustive search methods. However, the choice of beam width has a profound impact on the performance and speed of inference: a narrower beam makes the process faster but more resembles the greedy search, risking poorer quality outputs; a wider beam, explores a broader range of possible translations and can result in higher quality outputs but at the cost of increased computational demand. This balance between search width and computational efficiency makes beam search a preferred strategy in modern NMT systems, facilitating the production of translations that are both accurate and contextually rich.

Algorithm 1 Beam Search.

```
Require: decoder, start_token_id, end_token_id, vocab_size, beam_width,
    max_length
Initialize beams with start token
for each step in max_length do
    Initialize candidates list
    for each beam (score, sequence) do
        if sequence ends with end token then
            Add beam to candidates
            continue
        end if
        Get next token probabilities and scores from decoder
        Select top tokens and their scores
        for each top token do
            Create new sequence and update score
            Add new candidate to candidates list
        end for
    end for
    Sort candidates by score in descending order
    Keep top scoring candidates as new beams
end for
return sequence with the highest score from beams
```

2.1.5 Hyperparameters

Hyperparameters are predefined settings of a model that are fixed before training and do not update during the training process. They play a crucial role in the development of NMT systems, particularly for the Transformer model.

No free lunch theorem According to the no free lunch theorem (Wolpert, 1996), there is no universally best model for all types of problems. In the context of machine translation, this implies that optimal hyperparameters can vary significantly across different tasks—such as language pairs, domains, and sizes of training data.

Hyperparameters for developing an NMT system are diverse and can be categorized into several groups, including architecture, training, inference, and data preprocessing. Table 2.1 outlines some of the critical hyperparameters within these categories.

Category	**Hyperparameter**	**Type**
Architecture	Number of layers for encoder and decoder.	Integer
	Number of units in Transformer layers (model size).	Integer
	Number of hidden units in feed-forward layers.	Integer
	Number of heads for all self-attention layers.	Integer
	Type of the activation to use for each feed-forward layer.	Categorical
Training	Mini-batch size per process.	Integer
	Initial learning rate.	Real number
	Learning rate scheduler type.	Categorical
	Type of the optimizer.	Categorial
	Weight decay constant.	Real number
	Smoothing constant for label smoothing.	Real number
	Dropout probability for source and target embeddings.	Real number
	Dropout probability for multi-head attention.	Real number
	Maximum number of updates.	Integer
	Checkpoint and evaluate every x updates.	Integer
	Random seed.	Integer
Inference	Size of the beam for beam search.	Integer
Data	Number of subword units.	Integer

Table 2.1: Hyperparameters for neural machine translation system development.

2.1.5.1 Architecture Hyperparameters

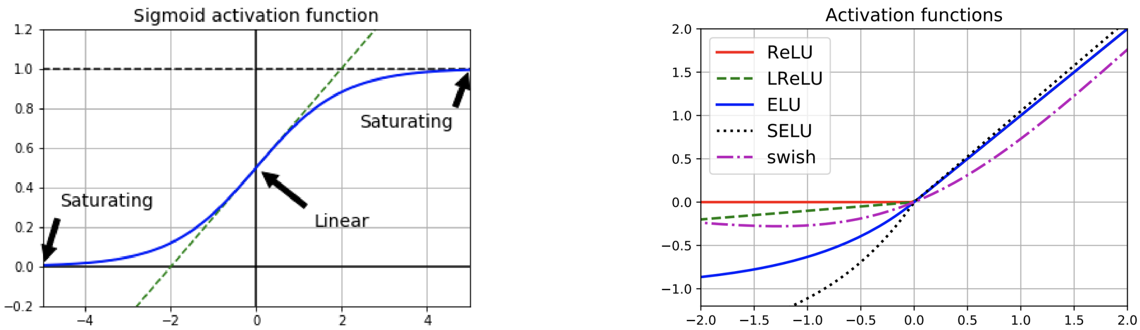
Architecture hyperparameters define the structural aspects of the model.

Number of layers and units The number of layers, or the depth of the model, and the number of units in each layer, or the model's width, collectively determine the model's capacity. The appropriate depth and width depend on the complexity of the task and the volume of training data. Deeper and wider networks can model more complex functions but are prone to overfitting, especially with limited training data. Conversely, simpler models may underfit, failing to capture sufficient complexity. Larger models are also more challenging to train effectively, often requiring longer training times and more computational resources for both training and inference.

Number of heads In the Transformer architecture, the multi-head attention mechanism allows the model to attend to information from different representations at different positions simultaneously. Each 'head' processes (Q, K, V) independently, which enhances the model's ability to capture various aspects of the input data. Although multiple heads generally improve performance by enabling the model to focus on different types of information, having too many heads can lead to redundancy, as some heads may learn to perform similar computations, thus plateauing performance improvements relative to the increased computational cost (Vaswani et al., 2017; Voita et al., 2019).

Type of the activation Activation functions introduce non-linear properties to neural networks, enabling them to learn complex patterns. The choice of activation function significantly influences the network's training dynamics and its ability to model complex functions. The Transformer typically uses the ReLU activation function (Nair and Hinton, 2010). Alternative activation functions, such as the sigmoid and hyperbolic tangent (tanh), are also commonly used, as illustrated in Figure 2.3.

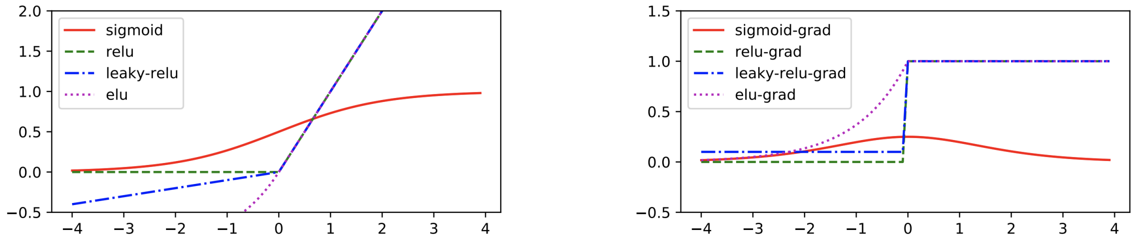
The sigmoid function outputs values between 0 and 1, and its gradient saturates at 1 for large positive inputs and at 0 for large negative inputs. Similarly, the tanh function outputs values between -1 and 1, with gradients that also saturate, causing a vanishing gradient problem in deep network layers. These characteristics can hinder training in very deep models. In contrast, the ReLU function activates only positive inputs, effectively "turning off" negative inputs. This behavior helps mitigate the vanishing gradient problem because it ensures that the gradient does not vanish as long as the input is positive. However, a potential drawback of ReLU is the "dead ReLU" phenomenon, where neurons can become inactive if weights are initialized such that they only produce negative values for all inputs in the forward pass.



(a) *(b)*

Figure 2.3: (a) Illustration of how sigmoid function is linear for inputs near 0, but saturates for large positive and negative inputs. (b) Plots of gradients of some activation functions. This figure is from Murphy (2022).

To address the issue of dead ReLU, alternatives (Figure 2.4) such as Leaky ReLU (Maas et al., 2013) have been proposed. Leaky ReLU introduces a small, non-zero gradient for negative inputs (represented by a constant α), which helps maintain the flow of gradients during training, even when the input is negative.



(a) (b)

Figure 2.4: Some popular activation functions (a) and the plots of their gradients (b). This figure is from Murphy (2022).

2.1.5.2 Training Hyperparameters

Training hyperparameters are critical as they define the dynamics and efficiency of the training process for NMT models.

Batch size The Transformer model typically employs SGD for optimization. In SGD, model weights are updated iteratively to minimize the loss function according to the equation:

$$\theta_{t+1} = \theta_t - \rho_t \nabla \theta_t, \tag{2.7}$$

where ρ is the learning rate, adjusting the step size during the gradient descent. In practice, training data are grouped into mini-batches rather than processed individually. This modification to the standard SGD approach, often referred to as mini-batch Gradient Descent, allows the gradients of all samples within a mini-batch to be computed simultaneously, leveraging computational parallelism for efficiency.

The batch size is a pivotal hyperparameter that significantly influences the training process. Using larger batches can stabilize the training process by reducing the variance in gradient estimates, as each update is informed by a broader set of examples. This can lead to smoother convergence behaviors. However, larger batches require more memory and computational resources. Additionally, overly stable gradients might hinder the model's ability to escape local minima, potentially affecting the ability to find better solutions on the loss landscape. Smaller batches, conversely, increase the stochastic nature of the training process. This can be beneficial as the increased noise in the gradient estimates might help the model escape suboptimal local minima. However, smaller batches often lead to more volatile training paths and may require more iterations or epochs to converge, which can extend the overall training time. The reduced memory requirement is a practical advantage for smaller batch sizes, making them suitable for environments with limited computational resources. Choosing the appropriate batch size is thus a balance between computational efficiency, memory constraints, and the desired stability in the training updates. This decision can have a profound impact on the training speed, the quality of the trained model, and its ultimate performance.

Learning rate and scheduler The choice of learning rate is critical when using SGD as it fundamentally influences the convergence of the training process. An excessively small learning rate can slow down training progress significantly, leading to underfitting, while an overly large learning rate may cause the training to become unstable and oscillate without settling to a minimum.

To address the challenges of selecting an appropriate learning rate, NMT training often adopts a learning rate scheduler, which adjusts the learning rate according to a predefined schedule as training progresses. Figure 2.5 illustrates some commonly used learning rate schedulers that dynamically adjust the rate during the training cycle.

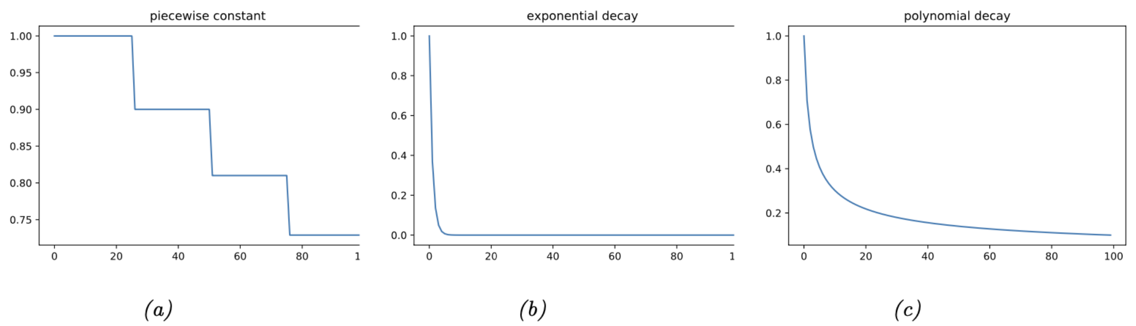
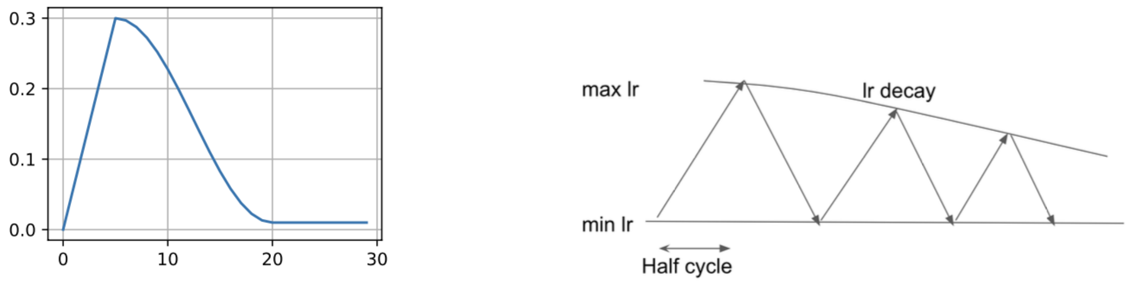


Figure 2.5: Illustrations of some common learning rate schedulers. This figure is from Murphy (2022).

An effective scheduling technique is the learning rate warmup, where the learning rate initially starts low and then quickly ramps up before gradually decreasing. This strategy is illustrated in Figure 2.6 (a). The rationale behind this strategy is to mitigate early training instability when model parameters are far from optimal regions of the loss landscape. For warmup scheduling, it is necessary to specify the number of warmup updates and the maximum learning rate to which the learning rate should initially ramp up.

Another strategy is the cyclical learning rate, shown in Figure 2.6 (b), where the learning rate oscillates between a lower and an upper bound. This method aims to escape local minima by allowing temporary increases in the learning rate to “jump out” of shallow solution basins, potentially leading to better overall solutions. For cyclical learning rate scheduling, the minimal and maximal learning rates must be defined, along with the number of updates for each phase of the cycle—both increasing and decreasing.



(a) *(b)*

Figure 2.6: Illustrations of different learning rate scheduling strategies. (a) Linear warmup followed by cosine cool-down. (b) Cyclical learning rate schedule. This figure is from Murphy (2022).

Effective use of learning rate schedulers can significantly enhance model training by adapting the step size of updates to the changing requirements of the training process. This adaptability is crucial for training deep neural networks efficiently and achieving robust convergence behavior.

Optimizer The type of optimizer dictates how model parameters are updated based on computed gradients. Popular optimizers include AdaGrad (Duchi et al., 2011), RMSProp, AdaDelta (Zeiler, 2012), and Adam (Kingma and Ba, 2014), with Adam demonstrating robust performance in numerous deep learning tasks, including NMT.

Adam combines the advantages of AdaGrad and RMSProp, addressing their respective limitations by adjusting the learning rate for each parameter based on estimates of the first and second moments of the gradients. The parameter update rule in Adam is given by:

$$\theta_{t+1} = \theta_t - \rho_t \frac{1}{\sqrt{\mathbf{s}_t} + \epsilon} \mathbf{m}_t, \tag{2.8}$$

where $\mathbf{m}_t$ and $\mathbf{s}_t$ represent exponentially weighted moving averages (EWMAs) of the squared gradients and gradients:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t, \tag{2.9}$$

$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2, \tag{2.10}$$

where β_1 and β_2 are typically set to 0.9 and 0.999, respectively, $\epsilon > 0$ is a small term to avoid dividing by zero, and $\mathbf{g} = \nabla \mathcal{L}(\theta)$. The hyperparameters of β_1 and β_2 help control the decay rates of these moving averages, influencing the optimizer's sensitivity to recent versus accumulated gradient information.

The term $\mathbf{m}_t$, often referred to as momentum, accelerates the gradient descent by integrating gradients from previous updates. This mechanism allows the optimizer to build up velocity in directions with consistent gradients, promoting faster convergence. The adaptive component $\sqrt{\mathbf{s}_t}$ helps in scaling the updates, especially for handling sparse data which is common in NMT. For frequently occurring features (like common words), $\mathbf{s}_t$ increases quickly, leading to smaller parameter updates as the model becomes more confident about those features. Conversely, for infrequent features (like rare words), $\mathbf{s}_t$ remains small, allowing larger updates that help in learning from rare but potentially informative instances.

This dynamic adjustment of learning rates for each parameter enables Adam to be highly effective across a variety of data distributions, making it a favored choice for training NMT models where different features may exhibit vastly different frequencies and importance.

Regularization Regularization is crucial for preventing overfitting in large neural models such as the Transformer. Some common regularization techniques include weight decay, label smoothing, dropout, and early stopping, each serving to constrain the model's complexity and improve its generalization abilities.

Weight decay Weight decay is a form of regularization that adds a penalty to the loss function to encourage smaller weight values, thus simplifying the model. This is typically implemented as L2 regularization:

$$\mathcal{L}'(\theta) = \mathcal{L}(\theta) + \frac{\lambda}{2}||\theta||^2, \tag{2.11}$$

where λ is the weight decay coefficient, determining the extent to which larger weights are penalized. This technique helps in reducing overfitting by discouraging the learning of overly complex models.

Label smoothing Label smoothing, introduced by (Szegedy et al., 2016), modifies the target distribution by reducing the target probability for the correct class and distributing a small amount of probability across all other classes. This approach prevents the model from becoming overly confident in its predictions, a common issue that can lead to poor generalization:

$$\boldsymbol{y}_{smoothed} = (1 - \eta) \cdot \boldsymbol{y} + \eta \cdot \frac{1}{V}, \tag{2.12}$$

where η is the label smoothing hyperparameter that controls how much probability is spread to other words, and V is the vocabulary size. One common choice for η in the Transformer models used for NMT is 0.1.

Dropout Dropout is a widely used regularization technique that randomly deactivates a subset of neurons during training with a probability p, effectively thinning the network temporarily. This technique helps to mitigate overfitting by forcing each neuron to function independently, reducing the risk of co-adaptation where neurons rely too heavily on the presence of particular other neurons.

Early stopping Early stopping is a simple regularization strategy that stops training when the validation error ceases to improve for a consecutive number of evaluation checkpoints, known as "patience". This approach helps ensure that the model does not overfit the training data by limiting the training time. The frequency of checkpointing—how often the model's performance is evaluated—also plays a crucial role. If set too high, important improvements might be missed; if too low, it can unnecessarily extend the training duration, especially in scenarios where model evaluation is computationally expensive.

Random seed The random seed, often an overlooked hyperparameter, plays a critical role in the reproducibility and consistency of training neural network models, including the Transformer. It governs aspects of the training process such as weight initialization and the ordering of training data. Research by Dodge et al. (2020) highlights the significant impact that variations in random seeds can have on the performance of Transformer models. Their findings indicate that some combinations of weight initializations and data orders, determined by different random seeds, can lead to markedly different model performances. This variability underscores the importance of carefully selecting and reporting random seeds in experiments to ensure that results are reliable and reproducible.

2.1.5.3 Inference Hyperparameters

Beam size Beam size used in beam search affects both the quality of the output and the inference speed. Selecting an appropriate beam size involves balancing between computational efficiency and the breadth of search. A smaller beam size leads to faster inference times by reducing the number of candidate translations considered at each step, but this can compromise the quality of the final output due to a more limited exploration of the potential solution space. Conversely, a larger beam size allows for a more extensive search, increasing the likelihood of finding higher-quality translations at the expense of greater computational demands and slower processing speeds. This trade-off between speed and quality must be carefully managed to optimize performance for specific translation tasks and computational environments.

2.1.5.4 Data Preprocessing Hyperparameters

Number of subword units Words are not uniformly distributed; many languages feature a long tail of rare words. To manage rare words and out-of-vocabulary (OOV) issues, one effective strategy is to segment words into smaller units known as subwords. Common methods for generating subword units include Byte Pair Encoding (BPE, Sennrich et al., 2016) and SentencePiece (Kudo and Richardson, 2018), which help improve handling of vocabulary diversity and reduce the model's complexity.

BPE starts by dividing words in the corpus into individual characters. It then repeatedly merges the most frequent adjacent pairs of characters or character sequences, adding these to the vocabulary as new units. This process continues until a predefined number of merging operations is reached, which effectively controls the granularity of the vocabulary. BPE can be applied separately to the source and target texts or jointly on a concatenated set of both, depending on whether a shared vocabulary is desirable.

SentencePiece is a toolkit that implements a variation of BPE. Unlike BPE, SentencePiece treats the input text as a raw sequence of Unicode characters, including whitespace, which it handles as a distinct symbol. This approach is particularly advantageous for languages without clear whitespace delimiters, such as Chinese or Japanese, as it eliminates the need for pre-tokenization. SentencePiece also allows for the direct specification of the desired vocabulary size instead of the number of merge operations as in BPE.

For example:

- **Raw text:** Hello world.
- **BPE tokenization:** [Hello][wor][@@ld][@@.]
- **SentencePiece tokenization:** [Hello][_wor][ld][.]

The choice between BPE and SentecePiece and the specific hyperparameters used, such as the number of merge operations or vocabulary size, should be tailored to the characteristics of the training data.

2.1.6 Machine Translation with Large Language Models

The emergence of Large Language Models (LLMs) such as GPT-4 (Achiam et al., 2023) and ChatGPT (OpenAI, 2023) have significantly reshaped the landscape of machine translation. These models, trained on vast amounts of data with extensive parameters, are increasingly capable of delivering translation performance that rivals traditional fully supervised machine translation systems, even in zero-shot scenarios where they have not been explicitly trained on translation tasks (Wei et al., 2021; Jiao et al., 2023). Moreover, LLMs have demonstrated impressive capabilities in more challenging translation scenarios, such as translating long documents, which typically pose substantial challenges for conventional models (Sia and Duh, 2023; Wang et al., 2023; Zhang et al., 2023d).

LLMs can be adapted for machine translation through two primary methods: in-context learning and fine-tuning.

In-context learning This technique involves adapting the LLM to machine translation tasks by using carefully crafted prompts that include a few relevant translation examples. These prompts guide the LLM to generate appropriate translations based on the provided context, without any updates to the model' s parameters (no back-propagation involved). Important hyperparameters in this method include the choice of prompt, the selection and number of example translations, and the temperature setting, which controls the model's output variability—higher for more creative outputs, lower for more deterministic responses.

Fine-tuning Fine-tuning, unlike in-context learning, involves additional training of a pre-trained LLM on targeted translation tasks to refine its abilities. This method updates the model's parameters and necessitates careful tuning of several training hyperparameters such as batch size, learning rate, schedulers, optimizers, dropout rates, and evaluation frequency, as detailed in Section 2.1.5.2. This process tailors the LLM more closely to specific translation needs.

Variability and selection of LLMs The field of LLMs is rapidly evolving, with a plethora of models emerging that vary not only in size but also in their training regimes—ranging from pre-training and instruction tuning to more complex enhancements like reinforcement learning with human feedback (RLHF), retrieval augmented generation (RAG), or distillation. These models are often specialized, being pre-tuned for specific tasks like coding or conversational applications.

The effectiveness of LLMs in machine translation can vary based on their training data, language coverage, and the specific methodologies applied during their training. Therefore, selecting an appropriate LLM model is crucial and should be considered a hyperparameter in the development of machine translation systems using these models. The choice of the right LLM is essential to maximize performance across various languages and translation tasks (Zhang et al., 2023d).

2.2 Hyperparameter Optimization

With the emergence of more advanced and complex machine learning models, especially deep learning models, developers face numerous critical decisions to ensure the success of these systems. These decisions include:

- How to preprocess the data and what features to extract?
- How much data and what data to use for training?
- What training process to follow?
- Can previous training experiences be transferred and utilized?
- How to evaluate the model and which evaluation metrics to use to ensure its generalizability?
- Which model family to choose and what model architecture to build?

- What hyperparameter configurations to use?
- How to allocate computational resources?

Automated machine learning (AutoML) aims to automate these decisions. The goal is for users to simply provide data and specify the task target, while the AutoML system makes all necessary decisions, conducts the training and evaluation processes intelligently, and delivers a high-performing machine learning system. This approach democratizes machine learning by enabling users who are not machine learning experts to build state-of-the-art systems.

AutoML encompasses various topics:

1. *Neural Architecture Search (NAS)*: Focuses on automating the design of model architectures.
2. *Meta-Learning*: Studies how to learn from prior experiences to accelerate learning new tasks.
3. *Hyperparameter Optimization (HPO)*: The focus of this thesis, deals with optimizing the hyperparameters of machine learning systems. Hyperparameters are configurations for model training and architecture, which can not be updated during training.

HPO has several use cases:

- It reduces human effort and minimizes the need for extensive domain knowledge to build machine learning systems.

- It enhances the performance of machine learning systems, leading to state-of-the-art benchmarks across various fields.

- It improves the reproducibility and fairness of scientific studies. The importance of documenting the HPO process in research has been widely recognized and is included as an item under the “Responsible NLP Checklist”,[1] which emphasizes that presenting extensive tables of hyperparameters and the best-found values is essential in research publications.

HPO offers significant advantages but also faces several challenges that have led some developers to prefer manual tuning. These challenges include:

- *High cost*: HPO can be expensive, particularly when function evaluations are computationally intensive, as is often the case with large deep learning models.

- *Configuration space complexity*: Defining the configuration space can be challenging, especially in high-dimensional spaces where domain knowledge may be lacking.

- *Algorithm selection*: There are numerous HPO algorithms to choose from, requiring developers to have a solid understanding of each to make informed decisions.

- *Limited transferability*: Hyperparameter configurations often do not transfer well across different datasets, necessitating separate HPO runs for different tasks.

[1] https://aclrollingreview.org/responsibleNLPresearch/

This can become prohibitively expensive if a developer is building systems for a large number of tasks.

These challenges highlight the need for continued research and development in HPO methods to make them more accessible, efficient, and adaptable to various machine learning tasks.

In this section, we will formally define the HPO problem in Section 2.2.1, introduce some typical HPO algorithms in Section 2.2.2, and discuss publicly available HPO toolkits in Section 2.2.3.

2.2.1 Problem Definition

Given a machine learning algorithm with H hyperparameters, we denote the domain of the h-th hyperparameter by Λ_h and the overall hyperparameter configuration space as $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \dots \Lambda_H$. When trained with a hyperparameter setting $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ on data $\mathcal{D}_{train}$, the algorithm's performance metric on some validation data $\mathcal{D}_{valid}$ is $f(\boldsymbol{\lambda}) := \boldsymbol{\mathcal{V}}(\boldsymbol{\lambda}, \mathcal{D}_{train}, \mathcal{D}_{valid})$, where $f(\cdot)$ or $\boldsymbol{\mathcal{V}}(\cdot)$ could be accuracy or decoding time on $\mathcal{D}_{valid}$. In general, $f(\cdot)$ is computationally expensive to obtain; it requires training a model to completion, and then evaluating some performance metric on a validation set. For purposes of exposition, we assume that lower $f(\cdot)$ is better.

The goal of HPO is then to find a $\boldsymbol{\lambda}_\star = \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} f(\boldsymbol{\lambda})$, with as few evaluations of $f(\cdot)$ as possible. An HPO problem can be challenging for three reasons: (a) $\boldsymbol{\Lambda}$ may be a combinatorially large space, prohibiting grid search over hyperparameters. (b)

$f(\cdot)$ may be expensive to compute, so there is a tight budget on how many evaluations of $f(\cdot)$ are allowed. (c) f is not a continuous function and no gradient information can be exploited, forcing us to view the $\arg\min$ as a blackbox discrete search problem.

2.2.2 Algorithms

In this section, we introduce representative HPO algorithms. We begin with a broad overview of HPO strategies (Section 2.2.2.1). Subsequently, we introduce foundational techniques such as grid and random search (Section 2.2.2.2). Following this, we explore more sophisticated approaches, including Bayesian Optimization (Section 2.2.2.3) and population-based optimization (Section 2.2.2.4), which offer advancements over simpler methods. We then examine strategies for utilizing noisy but less computationally demanding function evaluations through multi-fidelity optimization (Section 2.2.2.5). The section concludes by extending the discussion to multi-objective optimization (Section 2.2.2.6), addressing scenarios with multiple criteria.

2.2.2.1 Generalization

There are many HPO methods, which can be categorized along various aspects as shown in Figure 2.7. We will first discuss these HPO algorithms in a broad context and then delve into each of them in detail in the following sections.

Sequential vs. Parallel: This categorization depends on whether the HPO algorithm is inherently designed for parallelism. Bayesian optimization is sequential

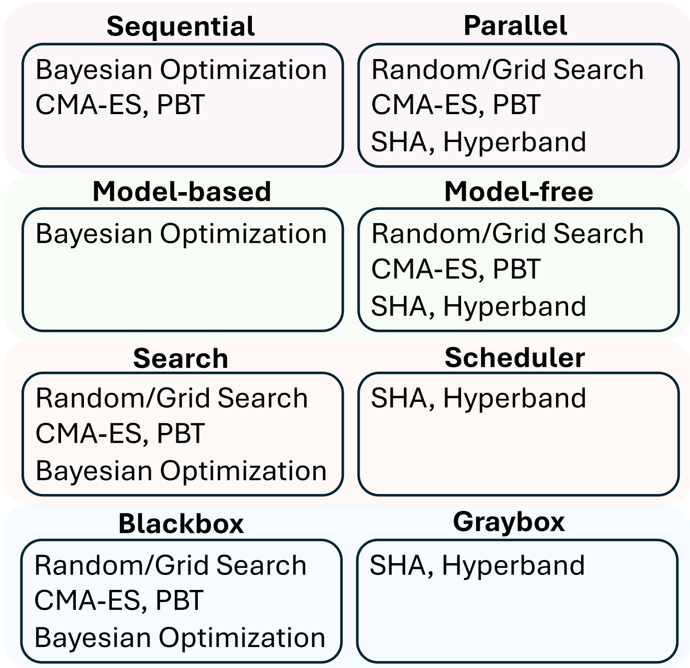


Figure 2.7: Generalizations of HPO algorithms.

because it requires evaluating previous configurations to propose new ones, making decisions based on past evaluations. In contrast, random search and grid search are parallel in nature since each configuration is sampled independently and can be evaluated simultaneously. Covariance matrix adaptation evolutionary strategy (CMA-ES) and population-based training (PBT) start with a set of configurations and sample new ones from a multivariate Gaussian distribution. Successive halving (SHA) and Hyperband are designed to efficiently evaluate multiple configurations concurrently. However, Bayesian optimization can be extended to a parallel approach by proposing multiple configurations at each step (Ginsbourger et al., 2011). Similarly, CMA-ES and PBT can be viewed as sequential since CMA-ES samples new configurations based on previous distributions, and PBT retains model parameters from previous configurations for evaluation of new ones.

Model-Based vs. Model-Free: Bayesian optimization uses a prediction model to fit all target function evaluations. In contrast, methods such as random search, grid search, CMA-ES, PBT, SHA, and Hyperband do not employ such a model.

Search vs. Scheduler: Some algorithms focus on searching—how to sample and what to sample next, such as random search, grid search, CMA-ES, PBT, and Bayesian optimization. Others, like SHA and Hyperband, focus on scheduling—deciding when to train a model and when to stop training. However, search algorithms and schedulers can be mixed and matched. For example, Hyperband adopts random search by default but can also be combined with Bayesian optimization (BOHB).

Blackbox vs. Graybox: Blackbox algorithms treat the function evaluation process as a black box, while graybox algorithms consider intermediate responses during the training process. Random search, grid search, CMA-ES, PBT, and Bayesian optimization visit the function evaluation only after the model is trained to convergence. In contrast, SHA and Hyperband terminate the training of poorly performing configurations midway. They can also be described as multi-fidelity optimization algorithms where training time is limited and noisy measurements are considered.

2.2.2.2 Grid Search and Random Search

Grid Search discretizes the range of each hyperparameter and performs an exhaustive search over the Cartesian product of these values. A significant drawback of grid search is that the number of function evaluations increases exponentially with the dimension of the search space (the curse of dimensionality) and the resolution of discretization.

Random Search samples hyperparameters randomly across the search space. It has several advantages over grid search. Random search is more flexible and evaluates a wider range of values for each hyperparameter. If the budget is B evaluations and there are N hyperparameters, grid search can evaluate only $B^{\frac{1}{N}}$ values for each hyperparameter, while the random search can evaluate up to B different values. As shown in Figure 2.8, this is particularly beneficial when some hyperparameters are more important than others, a common scenario (Bergstra and Bengio, 2012).

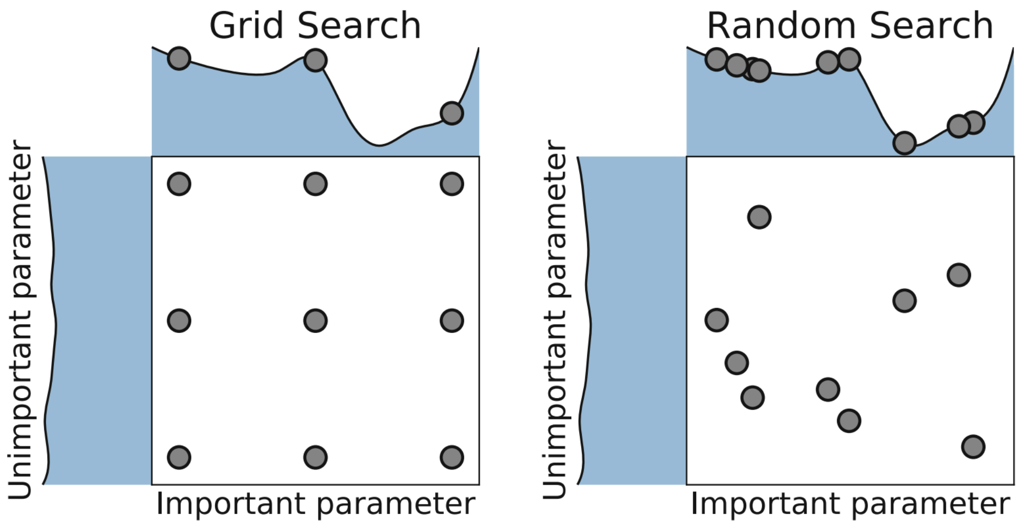


Figure 2.8: Comparison of grid search and random search for minimizing a function with one important and one unimportant parameter. This figure is from Hutter et al. (2019b) and Feurer and Hutter (2019a), which is based on the illustration in Fig. 1 of Bergstra and Bengio (2012).

Random search, despite its simplicity, has various use cases:

- It serves as a useful baseline that makes no assumptions about the machine learning algorithm being optimized. It often proves to be a strong baseline as well (Bergstra and Bengio, 2012).
- It can be used to initialize the search process for more advanced HPO algorithms.
- It can be combined with and interleaved with other HPO algorithms to add exploration and improve model-based search.

2.2.2.3 Bayesian Optimization

Bayesian optimization (Mockus, 1974; Jones et al., 1998) is a sequential model-based optimization method applied to global optimization problems. It aims to improve predictions with more data by balancing exploration (gathering information from under-explored areas) and exploitation (maximizing gains). This approach helps avoid getting trapped in local minima.

Bayesian optimization relies on two main components: a surrogate model and an acquisition function, and it operates iteratively. During each iteration, the surrogate model, which is less expensive to evaluate than the target function, is fitted to all observations of the target function collected so far. The acquisition function, using the predictive distribution of the probabilistic model, determines the utility of different candidate points. Choices for the surrogate model include Gaussian processes, neural networks, and random forests, while options for the acquisition function include probability of improvement (Kushner, 1964), expected improvement (Jones et al., 1998), lower confidence bound (Jones, 2001), and predictive entropy search (Hernández-Lobato et al., 2014). Gaussian processes and expected improvement are often the default choices for Bayesian optimization.

Gaussian Processes A Gaussian process (Rasmussen, 2003) is a nonparametric model where the number of parameters is determined by the dataset size. It assumes a multivariate Gaussian distribution of data points, updated based on Bayes' rule. Formally, a Gaussian process $\mathcal{G}(m(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}'))$ is defined by a mean function $m(\boldsymbol{\lambda})$ and a covariance function $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$. The mean function controls the smoothness and amplitude of samples, while the covariance function determines the quality of the surrogate model. Posterior predictions $\mu(\cdot)$ (typically with a prior $\mu_0(\boldsymbol{\lambda}) = 0$) and $\sigma^2(\cdot)$ are calculated as follows:

$$\mu(\boldsymbol{\lambda}) = \boldsymbol{k}_\star^T \boldsymbol{K}^{-1} \boldsymbol{y}, \tag{2.13}$$

$$\sigma^2(\boldsymbol{\lambda}) = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - \boldsymbol{k}_\star^T \boldsymbol{K}^{-1} \boldsymbol{k}_\star, \tag{2.14}$$

where $\boldsymbol{k}_\star$ represents the vector of covariances between $\boldsymbol{\lambda}$ and all previous observations, $\boldsymbol{K}$ is the covariance matrix of all previously evaluated configurations, and y is the observed function value. Common choices for the kernel function $k(\cdot, \cdot')$ include the

Matérn 5/2 kernel, squared exponential kernel, and Gaussian kernel.

Once the surrogate model is selected, the posterior distribution at any point can be determined by the mean and kernel function. The mean indicates expected results, where a smaller mean value (for minimization problems) implies a higher possibility of finding the optimum. The kernel indicates uncertainty, where a larger covariance value suggests that exploration might be beneficial. It measures the similarity between samples—if a sample is closer to an evaluated point, it is likely to have a similar value with less uncertainty.

Expected Improvement (EI) The acquisition function uses the predictive distribution of the Gaussian process to identify the point that balances exploitation (lower mean value) and exploration (higher covariance value) for maximum utility. The *expected improvement* is defined as follows:

$$u_{EI}(\boldsymbol{\lambda}) = \mathbb{E}[\max(f_{min} - y, 0)], \tag{2.15}$$

where f_{min} is the best observed value so far, and y is the prediction at configuration $\boldsymbol{\lambda}$. Specifically, it can be computed as follows:

$$u_{EI}(\boldsymbol{\lambda}) = (f_{min} - \mu(\boldsymbol{\lambda}))\Phi(\frac{f_{min} - \mu(\boldsymbol{\lambda})}{\sigma}) + \sigma\phi(\frac{f_{min} - \mu(\boldsymbol{\lambda})}{\sigma}), \tag{2.16}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the density and the cumulative distribution function of the standard normal distribution, respectively.

Figure 2.9 illustrates Bayesian optimization with Gaussian processes as the surrogate model and expected improvement as the acquisition function for minimizing a 1-dimensional function. The acquisition function value is low around evaluated points, with the highest value at points where the predicted function is low and predictive uncertainty is high. At iteration 4, despite low uncertainty around the true minimum, the next evaluation is performed there due to its expected improvement over the best point so far.

Despite its advantages, the Gaussian process has some drawbacks: (1) It only supports real-valued samples, lacking support for categorical or conditional hyperparameter configurations. (2) It scales poorly with high dimensions or a large number of data points, requiring $O(DN^3)$ to compute the kernel matrix, where D is the sample dimension and N is the number of data points.

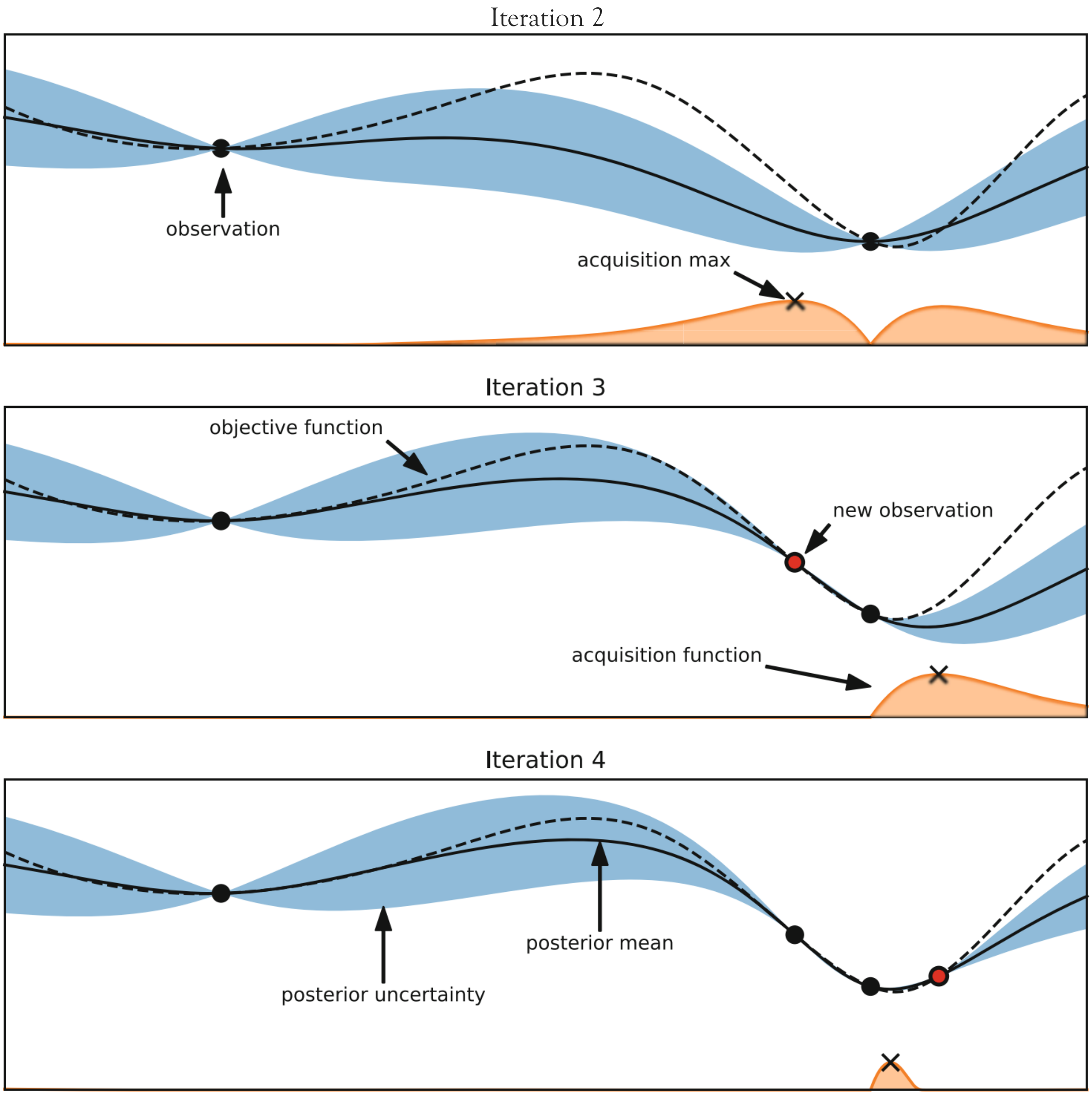


Figure 2.9: Illustration of Bayesian optimization on a 1-d function. The goal is to minimize the dashed line using a Gaussian process surrogate (predictions shown as the black line, with the blue tube representing the uncertainty) by maximizing the acquisition function represented by the orange curve. The figure is from Feurer and Hutter (2019a).

Surrogate Models Another option for the surrogate model is *neural networks*. Neural networks are highly flexible and can handle various types of inputs, such as categorical inputs via one-hot encoding. They also scale better than Gaussian processes in terms of sample dimensions and data sizes. After approximately 250 function evaluations, neural networks tend to be faster than Gaussian processes, enabling large-scale parallelism (Feurer and Hutter, 2019a). Neural networks can be adapted to a probabilistic model for Bayesian optimization primarily through two approaches: adding a Bayesian linear regression layer to the final layer of the network (Snoek et al., 2015), or using a Bayesian neural network treatment (Springenberg et al., 2016).

Random forests are another alternative surrogate model for Bayesian optimization (Hutter et al., 2011). They offer two main advantages over Gaussian processes: (1) They naturally handle larger, categorical, and conditional configuration spaces, and (2) They scale better with many data points, requiring only $O(n \log n)$ to fit the data points. Due to these advantages, the SMAC framework (Hutter et al., 2011) employs random forests for Bayesian optimization.

Acquisition functions In addition to expected improvement, another common acquisition function is the *lower confidence bound* (LCB, Jones, 2001):

$$u_{LCB}(\boldsymbol{\lambda}) = \kappa \cdot \sigma(\boldsymbol{\lambda}) - \mu(\boldsymbol{\lambda}), \tag{2.17}$$

where κ is a hyperparameter that controls the balance between exploration and exploitation, a smaller κ value leads to more exploitation, focusing on areas with lower predicted values, while a larger κ value encourages exploration by targeting high-variance points. Essentially, LCB treats local uncertainty as an additive bonus at each $\boldsymbol{\lambda}$ to promote exploration.

Tree Parzen Estimator (TPE) Tree Parzen Estimator (Bergstra et al., 2011; Bergstra et al., 2013) is another variant of Bayesian optimization. Unlike Gaussian processes, which model the probability $p(y \mid \boldsymbol{\lambda})$ of the observation y given the configuration $\boldsymbol{\lambda}$, TPE models the probability $p(\boldsymbol{\lambda} \mid y)$. TPE transforms $p(\boldsymbol{\lambda} \mid y)$ into a tree-structured representation:

$$p(\boldsymbol{\lambda} \mid y) = \begin{cases} l(\boldsymbol{\lambda}), & y < \alpha \\ g(\boldsymbol{\lambda}), & y \geq \alpha, \end{cases} \tag{2.18}$$

where α is a threshold that divides observations into good and bad, typically set to 15%. For the acquisition function, TPE maximizes the ratio $\frac{l(\boldsymbol{\lambda})}{g(\boldsymbol{\lambda})}$. The TPE workflow involves drawing a sample from $l(\boldsymbol{\lambda})$, evaluating $\frac{l(\boldsymbol{\lambda})}{g(\boldsymbol{\lambda})}$, and selecting the configuration $\boldsymbol{\lambda}$ that maximizes this ratio. TPE is conceptually simple, easy to parallelize, and is implemented in both Hyperopt-sklearn (Komer et al., 2014) and BOHB (Falkner et al., 2018)

Compared to grid search and random search, Bayesian optimization, including TPE, is more computationally efficient, requiring fewer attempts to find the optimal hyperparameter configuration. Additionally, users do not need prior knowledge of the hyperparameters' distribution. The posterior distribution in Bayesian optimization is updated after each trial, becoming more informative with more trials.

2.2.2.4 Population-based Optimization

Population-based methods are essentially a series of random searches based on genetic algorithms, such as evolutionary algorithms (Simon, 2013; Orive et al., 2014), particle swarm optimization (Eberhart and Shi, 1998; Lorenzo et al., 2017), and covariance matrix adaption evolutionary strategy (CMA-ES, Hansen, 2016). These algorithms maintain a population of configurations and iteratively enhance this population by applying local perturbations (*mutations*) and combinations of different members (*inheritance* and *crossover*) to generate a new, improved generation of configurations. This approach effectively merges parallel search capabilities with the benefits of sequential optimization. Here, we discuss two representative population-based methods: CMA-ES and population-based training (PBT Jaderberg et al., 2017).

CMA-ES CMA-ES optimizes by sampling configurations from a multivariate Gaussian distribution, focusing each iteration on regions of the search space that have shown high values of the objective function $f(\boldsymbol{\lambda})$. This is depicted in Figure 2.10, where the parameters $\boldsymbol{\theta}$ of the multivariate Gaussian distribution, defined by its mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, are adaptively updated at every iteration based on the performance outcomes $f(\boldsymbol{x})$ on the sampled configurations.

Formally, configuration sampling is performed as follows:

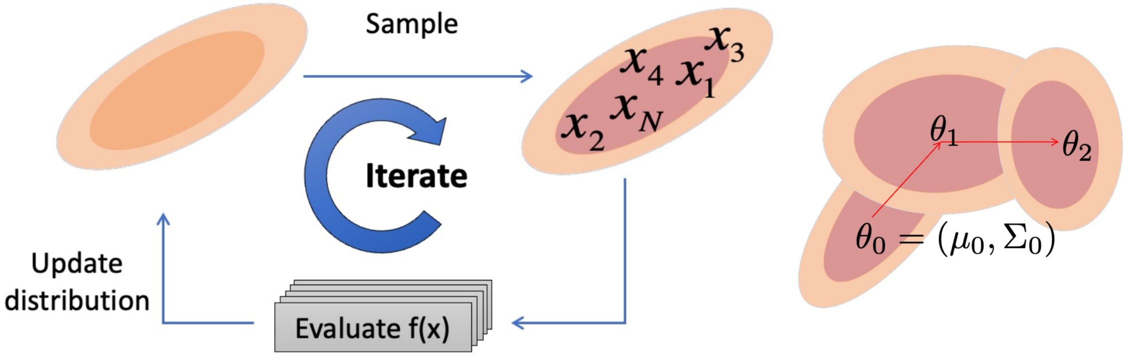


Figure 2.10: The illustration of covariance matrix adaption evolutionary strategy (CMA-ES). CMA-ES samples configurations from a multivariate Gaussian distribution, which is specified by a mean $\boldsymbol{\mu}$ and a variance $\boldsymbol{\Sigma}$. They are updated at each iteration based on the success of the population's individuals.

$$\hat{\boldsymbol{\lambda}} \sim \mathcal{N}(\boldsymbol{\lambda} \mid \hat{\boldsymbol{\theta}}) \quad s.t. \quad \hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \int f(\boldsymbol{\lambda})\mathcal{N}(\boldsymbol{\lambda} \mid \boldsymbol{\theta})\, d\boldsymbol{\lambda}, \tag{2.19}$$

where the integral

$$\int f(\boldsymbol{\lambda})\mathcal{N}(\boldsymbol{\lambda} \mid \boldsymbol{\theta})\, d\boldsymbol{\lambda} \triangleq \mathbb{E}[f(\boldsymbol{\lambda}) \mid \boldsymbol{\theta}]. \tag{2.20}$$

To update the distribution, the following updates are applied (Tanaka et al., 2016):

$$\hat{\boldsymbol{\mu}}_n = \hat{\boldsymbol{\mu}}_{n-1} + \epsilon_{\boldsymbol{\mu}} \sum_k w(y_k)(\boldsymbol{\lambda}_k - \hat{\boldsymbol{\mu}}_{n-1}), \tag{2.21}$$

$$\hat{\boldsymbol{\Sigma}}_n = \hat{\boldsymbol{\Sigma}}_{n-1} + \epsilon_{\boldsymbol{\Sigma}} \sum_k w(y_k) \cdot ((\boldsymbol{\lambda}_k - \hat{\boldsymbol{\mu}}_{n-1})(\boldsymbol{\lambda}_k - \hat{\boldsymbol{\mu}}_{n-1})^T - \hat{\boldsymbol{\Sigma}}_{n-1}), \tag{2.22}$$

where $w(y_k)$ is a weight function that assigns higher weights to configurations with more desirable outcomes y_k, facilitating a focus on more promising regions of the search space is successive generations. This method, through its adaptive nature, enables an efficient exploration and exploitation of the search space.

PBT Unlike methods such as CMA-ES, which assume models are trained to convergence, PBT periodically evaluates models during training. PBT dynamically adapts hyperparameters by replacing underperforming models with better-performing ones, inheriting their weights, and possibly exploring new hyperparameter configurations. This blend of adaptive hyperparameters and the integration of parallel and sequential optimization makes PBT a computationally effective method.

Figure 2.11 illustrates how PBT compares with sequential optimization, random search, and grid search. PBT begins with a random search, where models are evaluated at intervals. If a model in the population underperforms, it will exploit the better-performing models by adopting their hyperparameters and model weights. Concurrently, it explores new hyperparameter configurations by tweaking these inherited parameters before continuing training. This process allows PBT to effectively balance exploitation and exploration, optimizing model performance in a dynamic and ongoing manner.

2.2.2.5 Multi-fidelity Optimization

Hyperparameter optimization (HPO) is often a resource-intensive process, particularly when tuning neural networks due to their lengthy training times. This

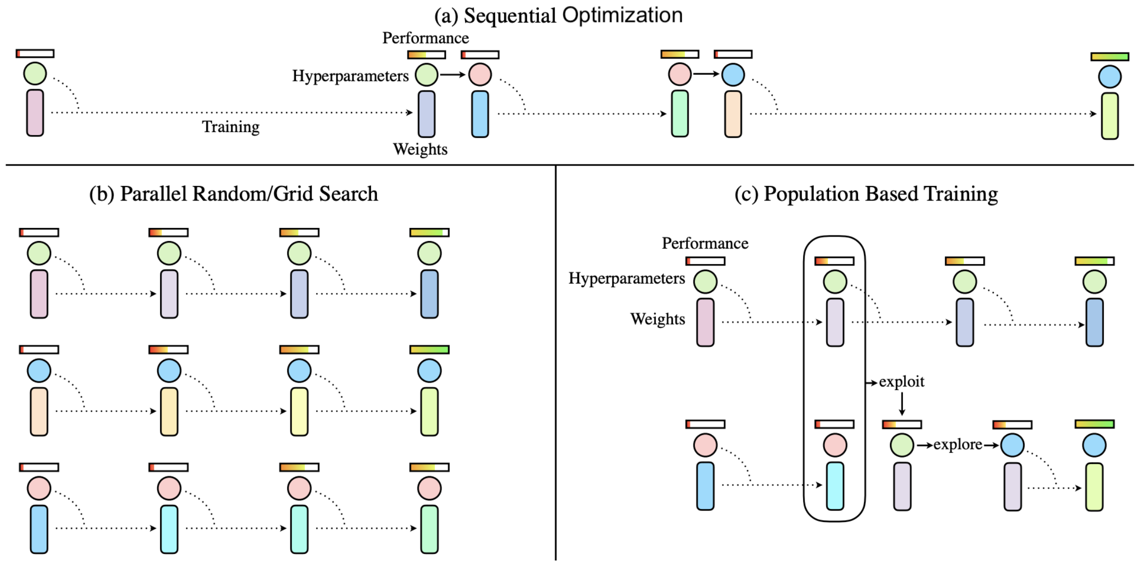


Figure 2.11: Comparison of different HPO paradigms: sequential optimization, parallel methods, and Population-Based Training (PBT). (a) Sequential optimization requires a training run to be completed before a new hyperparameter configuration can be evaluated. (b) Parallel methods, such as random search and grid search, train multiple models simultaneously. (c) PBT begins with a random search, with each model periodically evaluating its performance and asynchronously updating its hyperparameters based on a strategic balance of exploration and exploitation. This figure is from Jaderberg et al. (2017).

section explores HPO methods that leverage low-fidelity performance approximations to identify the best configurations within the search space. Such approximations can be achieved by training models on a reduced subset of the training (Petrak, 2000; Bosch, 2004; Krueger et al., 2015; Sparks et al., 2015; Sabharwal et al., 2016) or by implementing early stopping techniques, the latter being the primary focus of our discussion.

Early stopping in HPO fundamentally differs from its use in neural network training, where it typically serves to prevent overfitting. In the context of HPO, early stopping is employed to terminate trials prematurely if they do not show promise relative to other configurations. This approach aims to optimize the use of computational resources by focusing efforts on training models with the most promising hyperparameter configurations. We will delve into several key methodologies in this area, including successive halving (SHA), its asynchronous counterpart (ASHA), the more robust HyperBand, and a hybrid approach combining Bayesian optimization with HyperBand (BOHB). These methods exemplify strategies for efficiently navigating the hyperparameter search space by reducing the computational burden of full training cycles.

Successive halving (SHA) As introduced by Jamieson and Talwalkar (2016), SHA is an efficient method for HPO that operates under a finite computational budget. SHA requires several inputs from the user: (1) the total budget B, which can be defined as either the number of iterations or the total training time, (2) the initial number of trials n, (3) the fraction $\frac{1}{p}$ of configurations to retain at each iteration, and (4) the minimal budget r allocated to each configuration. The process, illustrated in Figure 2.12, begins by training n hyperparameter configurations for r iterations. It then evaluates the performance of each configuration, retains the top $\frac{1}{p}$ performing configurations, discards the rest, and repeats this process until only one configuration remains.

The primary challenge with SHA is the trade-off between the number of configurations n and the budget per configuration $\frac{B}{n}$. This trade-off can be summarized as follows:

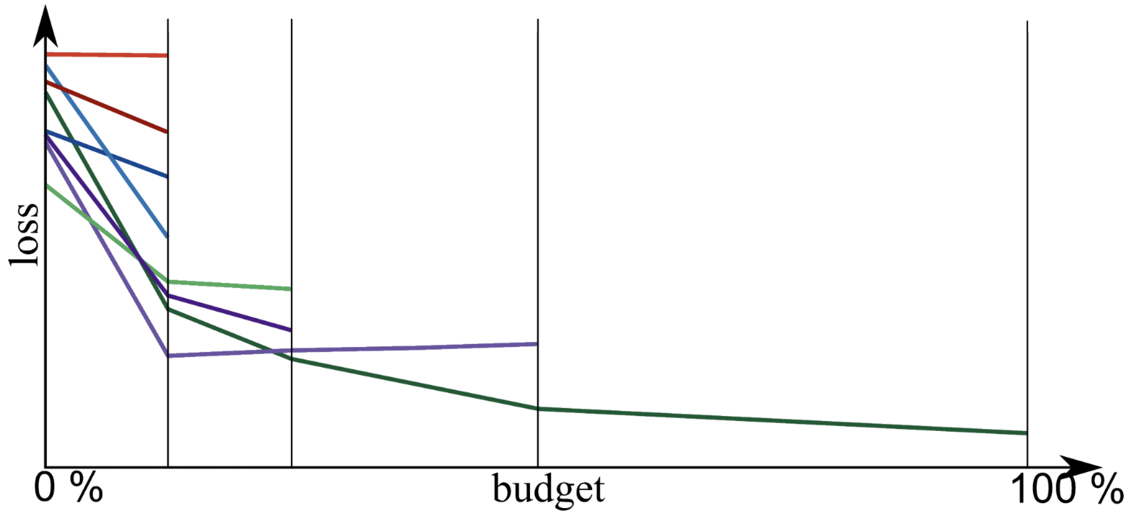


Figure 2.12: Illustration of successive halving. At each iteration, the worst-performing configurations are discarded until only one remains. For the remaining configurations, the budget is progressively increased until it reaches the maximum allocation. The figure is from Bissuel (2020).

1. **A large n:** If n is large, the resources allocated to each configuration $\frac{B}{n}$ will be small. This limited budget may not allow configurations sufficient training to showcase their potential, leading to potentially premature termination of promising configurations.

2. **A small n:** If n is small, the resources allocated to each configuration $\frac{B}{n}$ will be larger, allowing each to be trained more comprehensively. However, this comes at the expense of exploring fewer configurations, which might limit the discovery of the optimal configuration.

Asynchronous successive halving (ASHA) In the standard SHA approach, the algorithm waits for all configurations within a rung to complete before promoting any configurations to the next tier. However, ASHA (Li et al., 2020a), removes this bottleneck by allowing asynchronous promotions. A configuration is promoted to the next rung when (1) there is an idle worker available, and (2) the configuration has secured a position in the top $\frac{1}{p}$ of its current rung. This approach is depicted in Figure 2.13, which illustrates a run of ASHA for tuning hyperparameters for NMT systems. ASHA allows for enhanced parallelization and maximal GPU utilization.

HyperBand To address the "n vs. $\frac{B}{n}$ problem" in SHA and ASHA, HyperBand (Li et al., 2018) extends the principle of SHA by dynamically adjusting n and $\frac{B}{n}$ across different runs to make the process more robust and comprehensive. HyperBand effectively manages the exploration-exploitation trade-off by executing several SHA instances with varying configurations, thus improving the likelihood of identifying optimal hyperparameters.

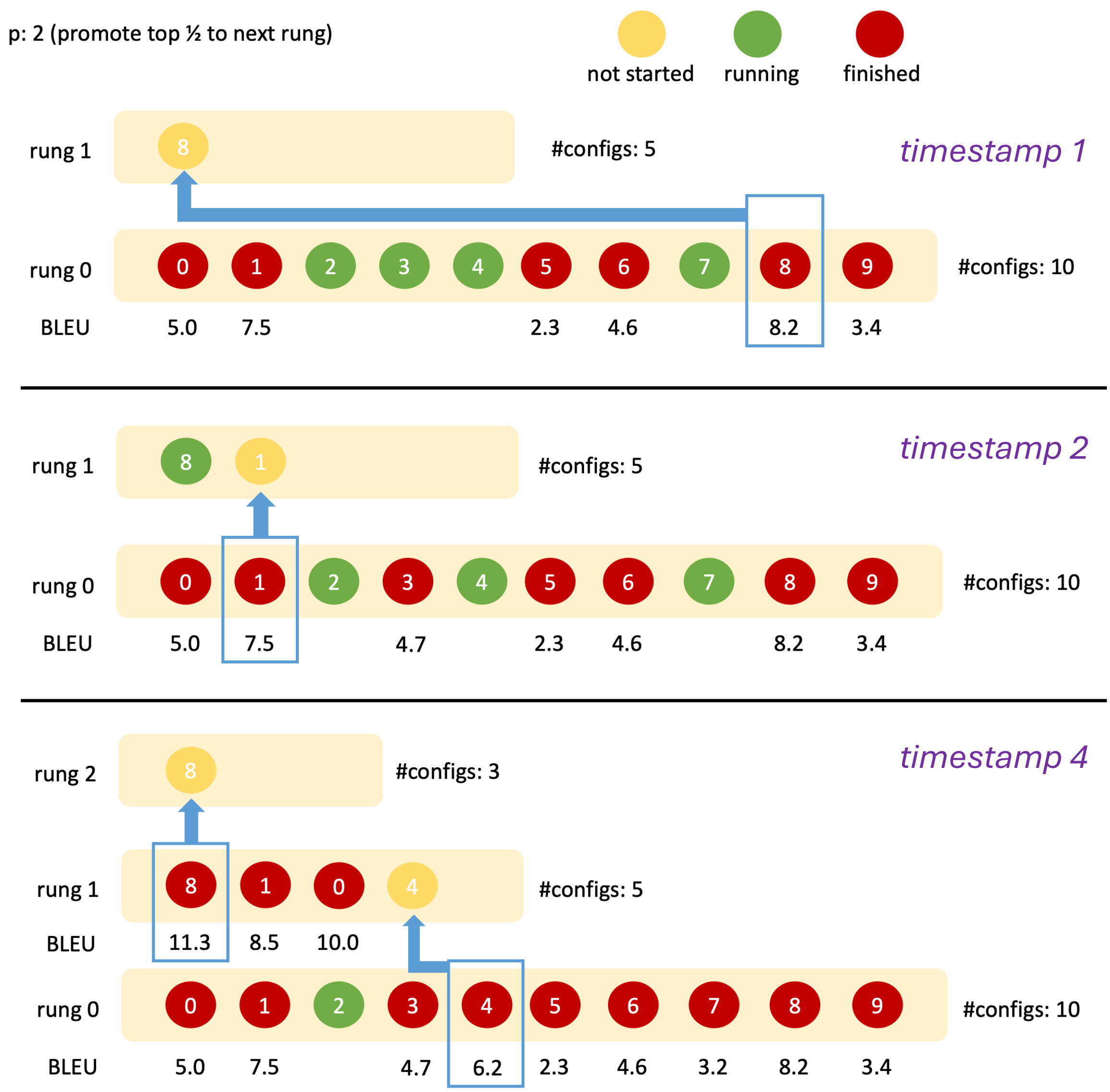


Figure 2.13: An example run of asynchronous successive halving applied to hyperparameter tuning for neural machine translation systems, demonstrating the asynchronous promotions. There are 10 configurations under evaluation. At each rung, configurations are scored on the BLEU score, with only the top $\frac{1}{2}$ advancing to the next rung. For instance, at timestamp 1, configuration 8 is promoted despite others still training, as it has secured a top position for the next tier.

BOHB BOHB enhances the traditional HyperBand approach by integrating Bayesian optimization into the search strategy, replacing the default random sampling. This hybrid method, introduced by Falkner et al. (2018), leverages the robust scheduling and budgeting framework of HyperBand while employing TPE for the Bayesian optimization component.

In BOHB, the systematic selection of budgets and the scheduling of trials are directly inherited from HyperBand, ensuring that the method retains its capacity for parallel execution. The Bayesian optimization part, specifically TPE, focuses on efficiently navigating the hyperparameter space by modeling the probability of achieving improvements over the best-observed performances. This combination allows BOHB to achieve rapid convergence towards the optimal hyperparameter configuration, benefiting from the strengths of both Bayesian optimization for precision and HyperBand for scalable parallelization.

2.2.2.6 Multi-objective Optimization

The HPO algorithms we have discussed so far focus on single-objective optimization. However, in practical applications, it is often necessary to balance multiple conflicting objectives, such as optimizing model performance while minimizing computational resources. A common approach to handle this complexity is to search for the Pareto front or a set of Pareto optimal points (Igel, 2005; Shah and Ghahramani, 2016; Horn and Bischl, 2016; Hernández-Lobato et al., 2016), which represents a set of configurations where no configuration can outperform another across all objectives without compromising at least one other objective. The challenge in multi-objective optimization is to efficiently identify this set of Pareto optimal configurations through as few evaluations of the objective functions as possible.

Pareto front Formally, consider the case where we aim to maximize $J \geq 2$ objectives, and the function evaluation of a configuration $\boldsymbol{\lambda}$ results in a vector of outcomes $F(\boldsymbol{\lambda}) = [\boldsymbol{y}^1, \boldsymbol{y}^2, \cdots, \boldsymbol{y}^J]$. The goal is to identify a set of Pareto optimal points. For distinct configurations $\boldsymbol{\lambda}_i$, and corresponding outcomes $\boldsymbol{y}_i \in \mathbb{R}^J$, for $i = 1, \cdots, n$, we denote $\boldsymbol{y}_i \succeq \boldsymbol{y}_k$ if $\boldsymbol{y}_i^j \geq \boldsymbol{y}_k^j$ for each $j = 1, \cdots, J$, indicating "$\boldsymbol{y}_i$ dominates $\boldsymbol{y}_k$". Within the set of all configurations $\mathcal{Y} = \{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n\}$, the subset of Pareto optimal points, $\mathcal{P}(\mathcal{Y}) \subseteq \mathcal{Y}$, is defined as

$$\mathcal{P}(\mathcal{Y}) = \{\boldsymbol{y}_i \in \mathcal{Y} : \boldsymbol{y}_k \not\succeq \boldsymbol{y}_i, \forall \boldsymbol{y}_k \in \mathcal{Y} \backslash \{\boldsymbol{y}_i\}\}. \tag{2.23}$$

In other words, the Pareto front is the set of non-dominated points. A point is considered dominated if there exists another point that achieves equal or higher values across all J objectives, rendering the dominated point suboptimal. Identifying this Pareto front allows decision-makers to choose from a set of equally optimal solutions, each representing a different trade-off among the objectives.

Pareto hypervolume While improving the Pareto front in multi-objective optimization offers a qualitative goal, unlike the quantitative improvements measured in

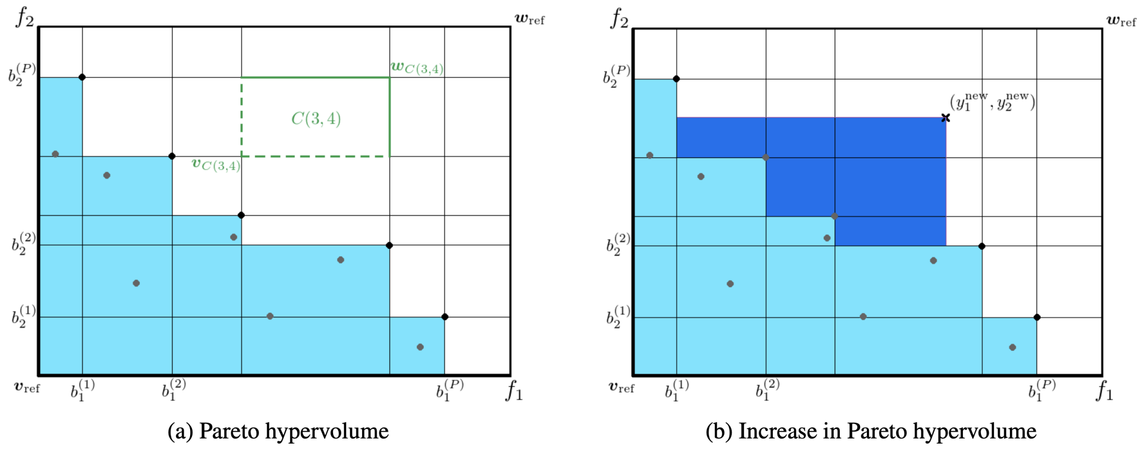


Figure 2.14: An example of optimizing (maximizing) two objectives simultaneously. This figure is from Shah and Ghahramani (2016).

single-objective optimization where enhancement is simply quantified by $f(\boldsymbol{\lambda}) - f(\boldsymbol{\lambda}')$, a metric is still required to assess the quality of a set of Pareto efficient points. The Pareto hypervolume, introduced by Zitzler and Thiele (1998), serves as a suitable measure for this purpose.

Given a set of distinct points $\mathcal{Y} = \{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n\}$, with its defined Pareto optimal subset, $\mathcal{P}(\mathcal{Y})$, we select a reference point $\boldsymbol{v}_{ref} \in \mathbb{R}^J$ that is dominated by each element of $\mathcal{P}(\mathcal{Y})$, such that $\boldsymbol{u} \succeq \boldsymbol{v}_{ref}$ for each $\boldsymbol{u} \in \mathcal{P}(\mathcal{Y})$. The Pareto hypervolume of $\mathcal{P}(\mathcal{Y})$ with respect to $\boldsymbol{v}_{ref}$ is defined as:

$$\mathrm{Vol}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y})) = \int_{\mathbb{R}^J} \mathbb{I}[\boldsymbol{y} \succeq \boldsymbol{v}_{ref}] \left[1 - \prod_{\boldsymbol{u} \in \mathcal{P}(\mathcal{Y})} \mathbb{I}[\boldsymbol{u} \not\succeq \boldsymbol{y}]\right] d\boldsymbol{y}, \tag{2.24}$$

where $\mathbb{I}(\cdot)$ is the indicator function. $\mathrm{Vol}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y}))$ measures the volume of points in $\mathbb{R}^J$ which dominate $\boldsymbol{v}_{ref}$ but are dominated by at least one element of the Pareto optimal set, $\mathcal{P}(\mathcal{Y})$. Illustrated in Figure 2.14, the shaded area represents this volume.

The Pareto hypervolume is a monotonic function; as more dominant points are added to $\mathcal{Y}$, the hypervolume $\mathrm{Vol}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y}))$ either increases or remains constant. Consequently, a larger Pareto hypervolume indicates a more dominant set of Pareto optimal points, making it a robust measure of the efficacy of the proposed solutions. Conversely, a marginal increase in the set of Pareto points will result in a smaller increment in the hypervolume, signifying lesser improvement in the dominance. Thus, the Pareto hypervolume effectively quantifies the "goodness" or efficacy of a set of Pareto optimal points in multi-objective optimization scenarios.

Expected improvement in Pareto hypervolume (EIPV) Similar to the concept of expected improvement used in Bayesian optimization, EIPV extends this idea to multi-objective contexts. Proposed by Emmerich et al. (2011), EIPV can be mathematically defined as follows:

$$\mathrm{EIPV}(\boldsymbol{\lambda}_{t+1} \mid \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{y}_{t+1}|\mathcal{D})}\big[\mathrm{Vol}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y} \cup \{\boldsymbol{y}_{t+1}\})) - \mathrm{Vol}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y}))\big], \tag{2.25}$$

where $\mathcal{D} = \{\boldsymbol{\lambda}_s, \boldsymbol{y}_s\}_{s=1}^{t}$ represents the set of evaluated configurations up to timestamp t and their corresponding outcomes. The probability distribution $p(\boldsymbol{y} \mid \mathcal{D})$ can be modeled using Gaussian processes, as in Bayesian optimization. Originally, Emmerich et al. (2011) considered modeling each objective as an independent Gaussian process. Shah and Ghahramani (2016) later extended this approach to accommodate correlated Gaussian process objectives, allowing for a more nuanced representation of dependencies between different objectives.

2.2.3 Toolkits

HPO toolkits can generally be categorized into two types: open-source tools and proprietary cloud-based services.

Open-source HPO toolkits: The Python ecosystem offers a variety of open-source HPO toolkits. `Spearmint` (Snoek et al., 2012), `BoTorch` (Balandat et al., 2020), and `Dragonfly` (Kandasamy et al., 2020) implement Bayesian optimization using Gaussian processes, each adding unique features to enhance this approach. `SMAC` (Hutter et al., 2011) supports both random forests and Gaussian processes as the surrogate model and is integral to several AutoML platforms, such as `auto-sklearn` (Feurer et al., 2015a). `HyperOpt` (Bergstra et al., 2013) focuses on using the TPE for Bayesian optimization. `Optuna` (Akiba et al., 2019) allows for dynamic construction of the search space, and `Orion`[2] specializes in asynchronous optimization. More advanced toolkits like `Neural Network Intelligence`[3] and `Ray.Tune`[4] provide comprehensive support for the latest HPO algorithms, catering to a wide range of optimization needs.

Cloud-based HPO services: For those requiring more substantial computational

[2] https://github.com/Epistimio/orion.
[3] https://github.com/microsoft/nni.
[4] https://docs.ray.io/en/latest/tune/index.html.

resources, cloud-based HPO services offer scalable solutions that support parallel training processes. These services are typically closed-source and need the users to pay for the service. Examples include `Google Vizier` (Golovin et al., 2017) and `Amazon SageMaker`,[5] which require minimal user configurations and support classical search algorithms, some early stopping methods, transfer learning capabilities, and user-friendly GUIs for task management and result visualization.

For a detailed introduction and comparison of different HPO toolkits, readers are encouraged to refer to the comprehensive reviews provided by the likes of Bischl et al. (2023) and Yu and Zhu (2020), which offer in-depth analyses and insights into the capabilities and applications of these tools.

2.2.4 Hyperparameter Optimization for Neural Machine Translation

Research on HPO for NMT is relatively sparse. Murray (2020) provides a comprehensive study on optimizing hyperparameters during the training of an NMT system, differing from the more common setup where hyperparameter configurations are set prior to training. They propose various methods for dynamically adjusting hyperparameters specific to NMT systems.

Beck et al. (2016) examines a constrained Bayesian optimization problem in

[5] `https://aws.amazon.com/sagemaker/`.

statistical machine translation, focusing on maximizing BLEU scores while ensuring that decoding speed exceeds a predetermined threshold. There is comparatively more research involving Neural Architecture Search (NAS) for Transformer-based NMT systems. Wang et al. (2020) propose designing Hardware-Aware Transformers (HAT) using NAS to facilitate low-latency inference on resource-constrained hardware. Hu et al. (2021a) introduces RankNAS, a performance ranking method that uses pairwise ranking to enable efficient NAS with significantly fewer training examples, thus reducing the time cost compared to HAT. Zhao et al. (2021) adapts the differentiable architecture search (DARTS), primarily used in vision tasks, to Transformers using a memory-efficient method.

For a broader overview of the application of AutoML techniques to NLP, including NMT model-building processes, Duh and Zhang (2023a) and Duh and Zhang (2023b) provide tutorials that are highly informative. Readers are encouraged to refer to these tutorials for a comprehensive understanding of this topic.

Chapter 3

Reproducible and Efficient Benchmarks for Hyperparameter Optimization of Machine Translation

As discussed in Chapters 1 and 2, while previous literature has proposed methods for automatic HPO, there has been limited work on applying these methods to NMT due to the high costs associated with training numerous model variants. To address this gap, this chapter introduces a lookup-based approach utilizing a library of pre-trained models to enable fast and cost-effective HPO experimentation.

We begin by motivating and introducing the ”table-lookup” approach for reproducible and efficient benchmarks in Section 3.1 and Section 3.2. We then describe the process of creating the benchmark dataset and the subsequent analyses in Section 3.3. In Section 3.4, we outline the evaluation protocols for assessing different HPO methods on the benchmark datasets. Related work is discussed in Section 3.5, and we conclude the chapter in Section 3.6.

3.1 Introduction

Choosing effective hyperparameters is crucial for building strong NMT systems. While some choices present obvious trade-offs (e.g., more and larger layers tend to increase quality at the cost of speed), others are more subtle (e.g., effects of batch size, learning rate, and normalization techniques on different layer types). Optimal versus suboptimal hyperparameters can lead to dramatic swings in system performance; consider the wide range of BLEU scores for variants of the same base system in Figure 3.1 (left). In practice, these hyperparameters are often tuned manually

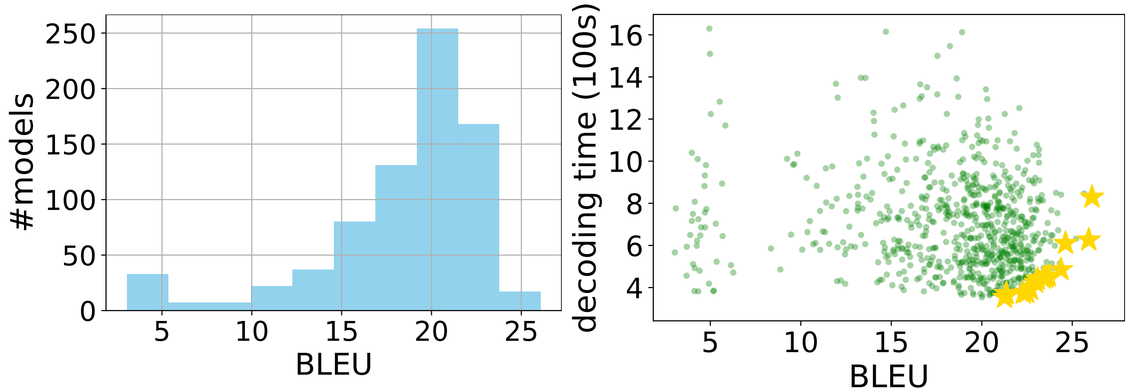


Figure 3.1: Left: Histogram of BLEU scores that show wide variance in performance for a base NMT system (Transformer) with different hyperparameters, e.g. BPE operations, # of layers, initial learning rate. Right: Scatter plot of BLEU & decoding time with different hyperparameters. Gold stars represent the Pareto-optimal systems.

based on intuition and heuristics, a tedious and error-prone process that can lead to unreliable experimental results and underperforming shared tasks or production systems. The difficulty is compounded when system builders must jointly optimize *multiple objectives*, such as translation accuracy (BLEU) and decoding speed, which are largely uncorrelated as shown in Figure 3.1 (right).

In the past decade, various HPO methods have emerged in the machine learning literature under the labels of "AutoML" (Bergstra et al., 2011; Hutter et al., 2011; Bardenet et al., 2013; Snoek et al., 2015) and "Neural Architecture Search" (Zoph and Le, 2016; Liu et al., 2018a; Liu et al., 2018b; Cai et al., 2018; Real et al., 2019). However, it is unclear how they perform on NMT; we are not aware of any prior work with comprehensive evaluation. One challenge is that the state-of-the-art NMT models (Sutskever et al., 2014; Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017) require significant computational resources for training. Secondly, they usually have large hyperparameter search spaces. Thus, it is prohibitively expensive in practice to apply HPO methods to NMT tasks.

Recently, with the rise of LLMs, NMT built upon LLMs has shown promising results (Hendy et al., 2023; Zhu et al., 2023; Sia and Duh, 2023; Zhang et al., 2023c). Adapting LLMs to NMT tasks typically involves in-context learning and supervised fine-tuning. Given the abundance of parallel data, fine-tuning has proven to be more effective than in-context learning (Zhang et al., 2023c). Parameter efficient fine-tuning (PEFT), such as Low-Rank Adapter (LoRA, Hu et al. (2021b)), is often favored over full fine-tuning due to its efficiency—fewer parameters are trained while achieving comparable or superior performance. Despite the fixed architecture of LLMs during PEFT, new hyperparameters are introduced, including the LoRA rank and the specific parameters to tune, alongside traditional hyperparameters like batch size and learning rate.

In order to *enable reproducible HPO research on NMT tasks*, we adopt a benchmark procedure based on "table-lookup." This approach was introduced to neural architecture search by Ying et al. (2019), and to hyperparameter optimization by Klein and Hutter (2019). First, we train an extremely large number of NMT models with diverse hyperparameter settings and record their performance metrics (e.g. BLEU, decoding time) in a table. Then, we constrain our HPO methods to sample from this finite set of models. This allows us to simply "look-up" their pre-computed performance metrics, and amortizes the burden of computation: as long as we ensure that we have trained and pre-computed a large number of representative NMT models beforehand, HPO algorithm developers no longer need to deal with the cost of training NMT. Importantly, this kind of benchmark significantly speeds up the HPO experiment turnover time, enabling fast repeated trials for rigorous tests and facilitating detailed error analysis.

The main focuses of this chapter are:

1. **Dataset**: We introduce a benchmark dataset, **NMTLC**,[1] designed for comparing hyperparameter optimization methods on NMT models. This "table-lookup" HPO dataset supports both single-objective and multi-objective optimization of translation accuracy and decoding time. It includes models trained from scratch as well as those fine-tuned from LLMs, with recorded learning curves and performance metrics for various hyperparameter settings. The dataset comprises 2469 models trained on 9 different corpora, costing approximately 2519 GPU[2] days. It is the first HPO benchmark dataset focused on NMT models.

2. **Evaluation protocols**: We provide three kinds of metrics for evaluating HPO methods, based on different computational budgets.

[1] https://github.com/Este1le/hpo_nmt

[2] We used GeForce RTX 2080 Ti with 13.45 teraflops for single-precision (32-bit) and NVIDIA TITAN RTX with 16.3 teraflops.

3.2 Methodology

3.2.1 Table-lookup Framework

To evaluate a newly devised HPO algorithm, one needs to run it on a hyperparameter search space, which involves querying the NMT model with a hyperparameter configuration, training the model, and getting evaluation results. This is computationally expensive: we need to train a new NMT system each time we sample a new hyperparameter.

The idea of table lookup is to simply pre-train a large set of NMT systems and record the configuration-evaluation pairs in a table. Thus, when testing an HPO algorithm, the developer can look up the table whenever necessary, without having to train an NMT model from scratch. This significantly speeds up the experimental process. The advantages are:

1. One can perform multiple random trials of the same algorithm, to test robustness.

2. One can perform comparisons with more baseline algorithms, to make stronger claims.

3. One can perform the same experiment under different budget constraints, to simulate different real-world use cases.

4. One can track the progress of an experiment with respect to Oracle results, allowing for a more detailed error analysis of HPO.

To be effective, table lookup depends on two important assumptions: First, the table has to be sufficiently large to cover the space of hyperparameters. Second, the HPO algorithm needs to be modified to sample from the finite set of hyperparameters in the table; this is usually easy to implement but the assumption is that finite-sample results will generalize.

3.2.2 HPO Algorithm Selection/Development

There exist many choices of HPO algorithm, which can be evaluated or further developed on our lookup tables. Figure 3.2 illustrates this process. The performance of HPO algorithm candidates on various MT datasets serves as the basis for HPO selection. The selected HPO algorithm can then be applied to new MT datasets.

There are two kinds of generalization effects at play: (1) generalization of an HPO algorithm across MT datasets, and (2) generalization of MT models and their associated hyperparameters across MT datasets. We mainly care about (1) in the algorithm development process, which is why we opt to provide 9 distinct datasets described in Section 3.3 (as opposed to e.g. 1 dataset trained on large MT data). If an HPO algorithm performs efficiently in finding good hyperparameter configurations on many MT datasets, then we can more reasonably believe that it will run quickly on a new dataset, regardless of the underlying MT data characteristics. Even if the best configuration on one MT dataset does not transfer to another, a robust HPO algorithm should still be capable of finding good hyperparameters because the algorithm learns

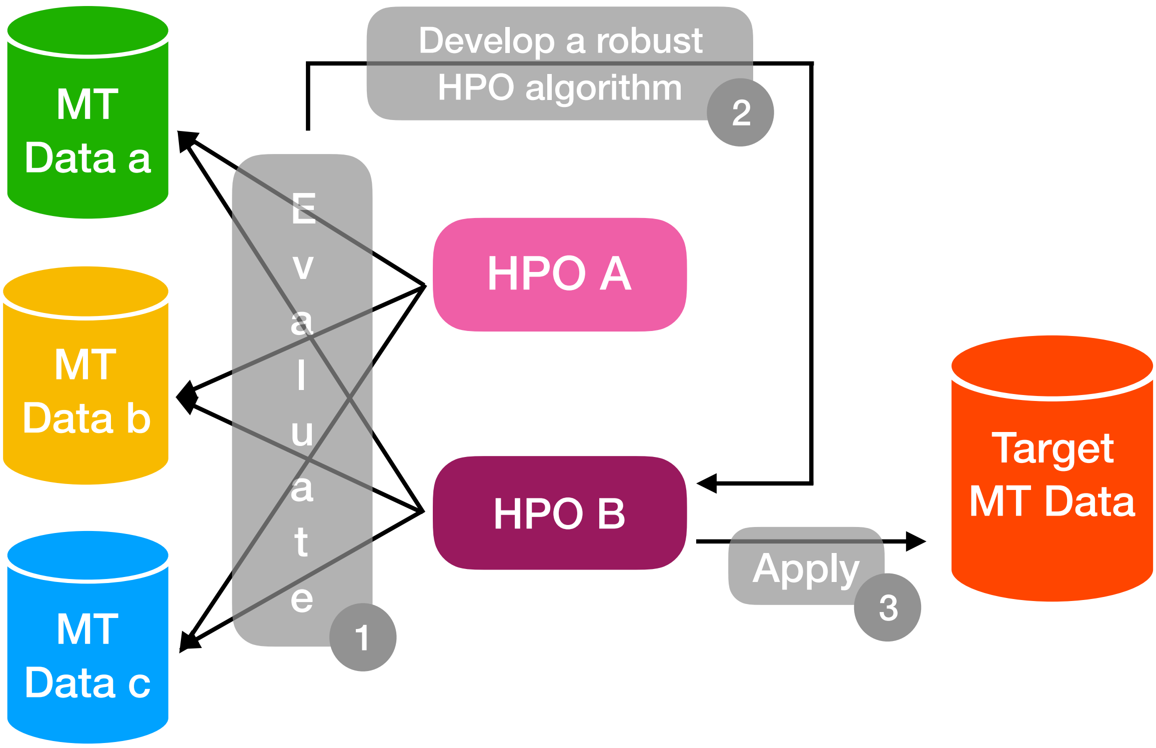


Figure 3.2: The workflow of HPO algorithm selection/development. HPO algorithm candidates are first evaluated on lookup tables built from multiple MT datasets. Promising candidates may be further developed and evaluated. The most robust one will be selected to apply to the target MT data.

from scratch on each dataset independently.

3.2.3 Reproducible and Efficient Benchmarks

Our table-lookup dataset enables reproducible and efficient benchmarks for the HPO of NMT systems. Li and Talwalkar (2019) introduce two notions of reproducibility: exact reproducibility — the reproducibility of reported experimental results; and broad reproducibility — the generalization of the experimental results.[3]

[3] They comment: "Of the 12 papers published since 2018 at NeurIPS, ICML, and ICLR that introduce novel Neural Architecture Search methods, none are exactly reproducible."

Our benchmarks are exactly reproducible in the sense that we provide the tables that record all model results. However, they are not guaranteed to be broadly reproducible, since the generalizability of the results might be restricted due to fixed collections of hyperparameter configurations, the variance associated with multiple runs, and the unknown best representative set of MT data. As a result, we should be careful not to make general conclusions from the observations, but to show how the dataset can be potentially used in facilitating HPO research.

3.3 Benchmark Datasets

We pre-train a large set of NMT systems and record their configurations, learning curves, and evaluation results in a table. This allows for efficient evaluation of HPO by looking up the table as needed, without training each model from scratch, significantly speeding up the experimental process. Our dataset includes 2469 models trained on 9 different corpora, encompassing both models trained from scratch and those fine-tuned from LLMs, with a total computational cost of approximately 2519 GPU days. This is the first HPO benchmark dataset on NMT tasks. It is also the first to include models fine-tuned from LLMs, facilitating HPO research on this emerging task.

3.3.1 Data and Setup

3.3.1.1 Trained-from-Scratch NMT models

To create a robust HPO benchmark, we trained NMT models from scratch on six different parallel corpora, which exhibit a variety of characteristics:

- **TED Talks**: We trained Chinese-English (**zh-en**) and Russian-English (**ru-en**) models on the datasplit of Duh (2018). This is a mid-resource setup, where $\mathcal{D}_{train}$ consists of 170k lines for zh-en and 180k lines for ru-zh. $\mathcal{D}_{valid}$ has 1958 sentences and is multi-way parallel for both language pairs.

- **WMT2019 Robustness Task** (Li et al., 2019): We trained models on the Japanese-English data, in both directions (**ja-en**, **en-ja**). $\mathcal{D}_{train}$ has 4M lines from a mix of domains. $\mathcal{D}_{valid}$ is a concatenation of 4k mixed-domain sentences and 1k Reddit sentences, for a total of 5405 lines. The goal of the Robustness task is to test how NMT systems perform on non-standard and noisy text.

- **Low Resource tasks**: We trained models using the IARPA MATERIAL datasets for Swahili-English (**sw-en**) and Somali-English (**so-en**). $\mathcal{D}_{train}$ consists of only 24k lines for both language pairs (BUILD set), and $\mathcal{D}_{valid}$ consists of 2675 lines (ANALYSIS2 set).

While there are many potential MT datasets we could choose from, we believe these 6 datasets form a good representative set. It ranges from high to low resources;

dataset	**bpe** (1k)	**#layers**	**#embed**	**#hidden**	**#att**	**lr** (10^{-4})
zh, ru, ja, en	10, 30, 50	2, 4	256, 512, 1024	1024, 2048	8, 16	3, 6, 10
sw	1, 2, 4, 8, 16, 32	1, 2, 4, 6	256, 512, 1024	1024, 2048	8, 16	3, 6, 10
so	1, 2, 4, 8, 16, 32	1, 2, 4	256, 512, 1024	1024, 2048	8, 16	3, 6, 10

Table 3.1: Hyperparameter search space for the trained-from-scratch NMT systems.

it contains both noisy and clean settings. They also have different levels of similarity: e.g., zh-en and ru-en TED tasks use the same multi-way parallel $\mathcal{D}_{valid}$, so one could ask whether the optimal hyperparameters transfer.

The text is tokenized by Jieba for Chinese, by Kytea for Japanese, and by the Moses tokenizer for the rest. Byte pair encoding (BPE) segmentation (Sennrich et al., 2016) is learned and applied separately for each side of bitext. We train Transformer NMT models with Sockeye[4] (Hieber et al., 2017), focusing on the these hyperparameters:

- **preprocessing configurations**: number of BPE symbols[5] (bpe)
- **training settings**: initial learning rate (lr) for the Adam optimizer
- **architecture designs**[6]: number of layers (#layers), embedding size (#embed), number of hidden units in each layer (#hidden), number of heads in self-attention (#att).

These hyperparameters are chosen because they significantly affect both the accuracy and speed of the resulting NMT. Other hyperparameters are kept at their

[4]`https://github.com/awslabs/sockeye`
[5]Same number of BPE operations is used for both sides.
[6]Same values are used for encoder and decoder.

Dataset	**#models**	**Best BLEU**	**bpe**	**#layers**	**#embed**	**#hidden**	**#att**	**lr**
zh-en	118	14.66	30k	4	512	1024	16	3e-4
ru-en	176	20.23	10k	4	256	2048	8	3e-4
ja-en	150	16.41	30k	4	512	2048	8	3e-4
en-ja	168	20.74	10k	4	1024	2048	8	3e-4
sw-en	767	26.09	1k	2	256	1024	8	6e-4
so-en	604	11.23	8k	2	512	1024	8	3e-4

Table 3.2: For each language pair, we report the number of NMT systems trained on it, the oracle best BLEU we obtained and its corresponding hyperparameter configuration.

Sockeye defaults.[7] Table 3.1 shows our overall hyperparameter space, in total among all 6 datasets, we have 1983 models; Table 3.2 shows the exact number of models per dataset, along with the best models and their hyperparameter settings.[8]

BLEU scores from existing literature: Among the submissions to the WMT19 Machine Translation robustness task (Li et al., 2019), Bérard et al. (2019) achieved the highest ranking, with BLEU scores of 16.41 and 17.73 for ja-en and en-ja, respectively. Their approach incorporated various enhancements to Transformer models, including the use of back-translation with additional data. In contrast, using only a thorough hyperparameter search, our system reached BLEU scores of 16.41 and 20.74 for the same language pairs, as detailed in Table 3.2.

[7] In this work, we only focused on integer and real-valued hyperparameters. Categorical hyperparameters need special treatment for most HPO algorithms and thus are not considered.

[8] Note that not all possible hyperparameter configurations are included in the dataset: we excluded ones where training failed or did not learn (e.g. achieved ≈ 0 BLEU).

3.3.1.2 NMT Models Fine-tuned from LLMs

LLMs excel in most NLP tasks (Yang et al., 2024). Recently, fine-tuning LLMs for machine translation has shown promising results (Zhang et al., 2023c; Moslem et al., 2023; Zhu et al., 2024). HPO on fine-tuned models is rarely studied, particularly for fine-tuned LLMs in machine translation. To address this gap, we include fine-tuned LLMs in our **NMTLC** benchmark datasets.

method	**domain**	**lang**	**train**	**dev**	**#cfg**
scratch	IARPA	sw-en	24k	2675	767
	IARPA	so-en	24k	2675	604
	TED Talks	zh-en	170k	1,958	118
	TED Talks	ru-en	170k	1,958	176
	WMT19	ja-en	4M	5,405	150
	WMT19	en-ja	4M	5,405	168
FT	WMT23	fr-en	404k	289	162
	WMT23	zh-en	421k	2139	162
	WMT23	de-en	435k	2342	162

Table 3.3: Data used for training NMT systems to build the **NMTLC** benchmark dataset, where NMT systems are trained either from scratch (*scratch*) or fine-tuned from LLMs (*ft*).

We explore 3 language pairs as shown in Table 3.3. For fr-en, the input format is as follows:

Translate French to English: French: [fr sent] English: [en sent] <eos>

A special <eos> token is added for post-processing.

We experiment with two types of LLMs:

1. **XGLM** - a multilingual language model trained on a balanced corpus covering 30 diverse languages with 500B tokens. The XGLM 7.5B outperforms GPT-3

on the FLORES-101 (Goyal et al., 2022b) machine translation benchmark in few-shot learning scenarios.

2. **BLOOMZ** - a multilingual BLOOM model (Scao et al., 2022) fine-tuned with the xP3 dataset (Muennighoff et al., 2022), which consists of multilingual datasets with English prompts, totaling 95 GiB of text.

We consider four hyperparameters to define the search space:

- **LLM (6):** BLOOMZ 560m, 1b7, and 3b, XGLM 564M, 1.7B, and 2.9B. Various versions affect model size, feed-forward size, number of layers, and vocabulary size.
- **LoRA rank (3):** 2, 16, and 64.
- **Batch size (3):** 16, 32, and 64.
- **Learning rate (3):** $2e-5$, $1e-4$, and $2e-4$.

We utilize QLoRA (Dettmers et al., 2023), a fine-tuning approach designed to minimize memory usage while maintaining the high performance typical of 16-bit precision. This approach works by first quantizing a pretrained model to 4-bit precision. After quantization, a small, trainable set of LoRA weights is integrated, which are then fine-tuned using backpropagation. We set the LoRA scaling factor to 32, limit trainable parameters to the self-attention layers, and apply a dropout rate of 0.05 in the LoRA layer. The model weights are quantized to 4-bit precision, and mixed-precision training (using float16 and float32) is enabled to accelerate the process. We use the Adam optimizer, evaluating performance every 1000 steps, and consider the model converged when performance does not improve for 12 checkpoints. Models are trained on a single NVIDIA RTX GPU with 24GB of memory.

3.3.1.3 Rationale for Hyperparameter Values

There are various design trade-offs in deciding the range and granularity of hyperparameter values. First, we might expand on a wider range of values (e.g. change #hidden = {1024, 2048} to {512, 1024, 2048, 4096}). The effect of this is that we test the HPO algorithm on a wider range of inputs, with potentially more variability in metrics like BLEU and inference speed. Second, we might expand on a more fine-grain range of values (e.g. change #hidden = {1024, 2048} to {1024, 1536, 2048}). This might result in smoother metrics, making it easier for HPO algorithms to learn. While wider range and finer granularity are desirable properties for an HPO dataset, each additional value causes an exponential increase in the number of models due to the cross-product of all values. In general, we think Table 3.1 represents a reasonable set of values used in the literature. Nevertheless, it should be clarified that empirical findings from table-lookup datasets should be interpreted in light of the limits of hyperparameter range and granularity.

3.3.1.4 Samples

We train all models on $\mathcal{D}_{train}$ until they converge in terms of perplexity on $\mathcal{D}_{valid}$. We then record various performance measurements. Each sample in the benchmark dataset includes:

1. Hyperparameter configuration.
2. Meta-information about the MT dataset (size, language pairs, domain).
3. Learning curve: a list of evaluation results (perplexity, BLEU) on $\mathcal{D}_{valid}$ throughout training until convergence.
4. Performance measurements:
   - **translation accuracy**: Optimal BLEU and perplexity on $\mathcal{D}_{valid}$.
   - **computational cost**: GPU wall clock time for decoding $\mathcal{D}_{valid}$, number of updates for the model to converge, GPU memory used for training, total number of model parameters.

In terms of models fine-tuned from LLMs, for de-en, we provide perplexity learning curves. For fr-en and zh-en, we include both perplexity and BLEU learning curves to study the correlation between these metrics.

3.3.2 Analysis

In this section, we present statistics and analysis of the **NMTLC** samples. We examine the distribution of BLEU scores (Section 3.3.2.1) and training times (Section 3.3.2.2) across different MT datasets. We also investigate the transferability of effective hyperparameter configurations between MT tasks (Section 3.3.2.3) and assess the importance of various hyperparameters (Section 3.3.2.4) based on their impact on NMT system performance. Additionally, given that NMT training can be non-deterministic due to random initialization, we explore the effect of random seeds (Section 3.3.2.5) on NMT performance and HPO outcomes.

3.3.2.1 BLEU Distribution

Figure 3.3 illustrates the performance variance of NMT models trained with different hyperparameter configurations in the **NMTLC** dataset, measured by the BLEU score. Models trained from scratch (*scratch*) and those fine-tuned from LLMs (*ft*) exhibit distinct BLEU score distributions. The BLEU scores of the *scratch* models generally follow a left-skewed distribution, indicating that most configurations result in good performance. In contrast, the BLEU scores of the *ft* models display a multimodal distribution, suggesting a wide variation in performance, with many configurations yielding either very good or poor results. For instance, in *ft_fr-en*, the BLEU scores range widely, with differences up to 30 points between the best and worst models. Additionally, some configurations in almost all tasks (except *scratch_ja-en*) produce

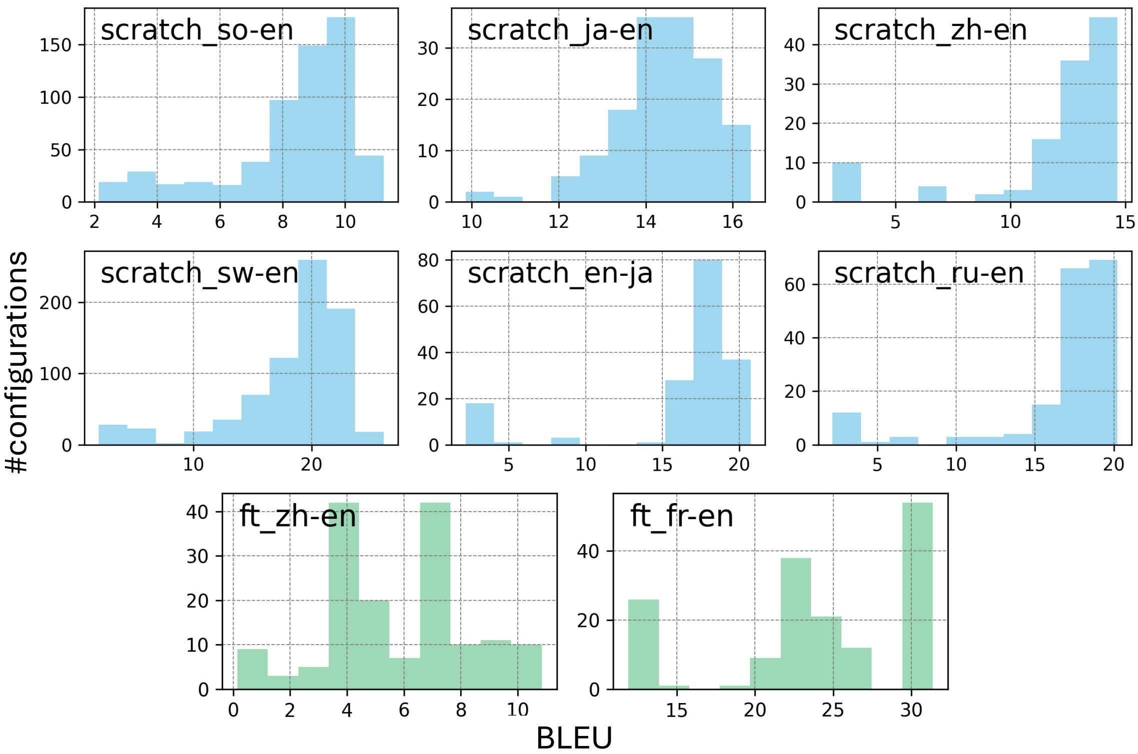


Figure 3.3: BLEU distribution on the hyperparameter search space of the **NMTLC** benchmark dataset, which includes both NMT systems trained from scratch (*scratch* with blue histograms) and fine-tuned from LLMs (*ft* with green histograms). This shows that the performance of NMT systems is very sensitive to hyperparameter configurations.

nearly zero BLEU scores, underscoring the importance of extensive hyperparameter search. This highlights the necessity of HPO in efficiently exploring a large search space to find optimal hyperparameter configurations.

3.3.2.2 Length Distribution

Figure 3.4 shows the distribution of the lengths of the learning curves in the **NMTLC** dataset, where longer curves indicate models that take more time to converge. The length distribution reveals that in most tasks, a small number of models have

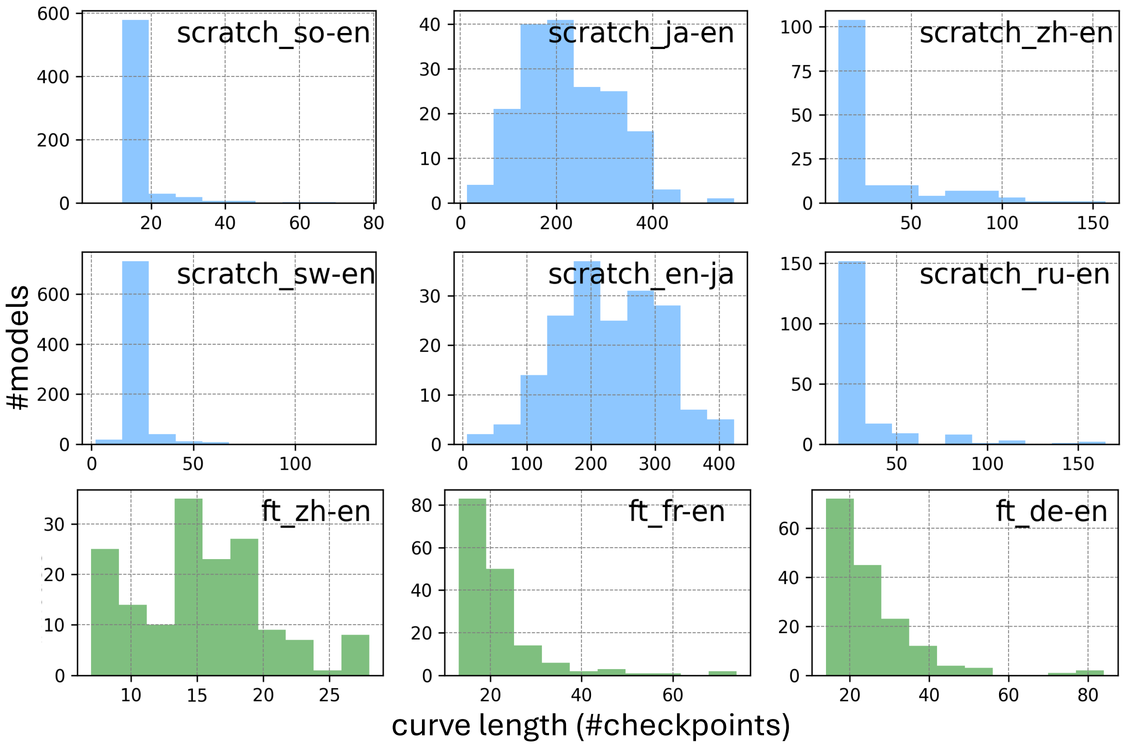


Figure 3.4: Learning curve length distribution on the hyperparameter search space of **NMTLC**. Colors are assigned to be consistent with Figure 3.3. This shows that the training time differs across and within MT datasets with different hyperparameter configurations.

extended training times, resulting in a long right tail in the distribution. In these cases, terminating unpromising models early in the training process can be beneficial in HPO, which saves substantial computational resources. Additionally, the length distributions vary across different tasks. While *scratch_ja-en*, *scratch_en-ja*, and *ft_zh-en* exhibit distributions similar to a normal distribution, other tasks display more left-skewed distributions. This variability further motivates the usage of multi-fidelity HPO methods, such as successive halving, to efficiently navigate the diverse convergence behaviors and optimize hyperparameter configurations.

3.3.2.3 Hyperparameter Correlation

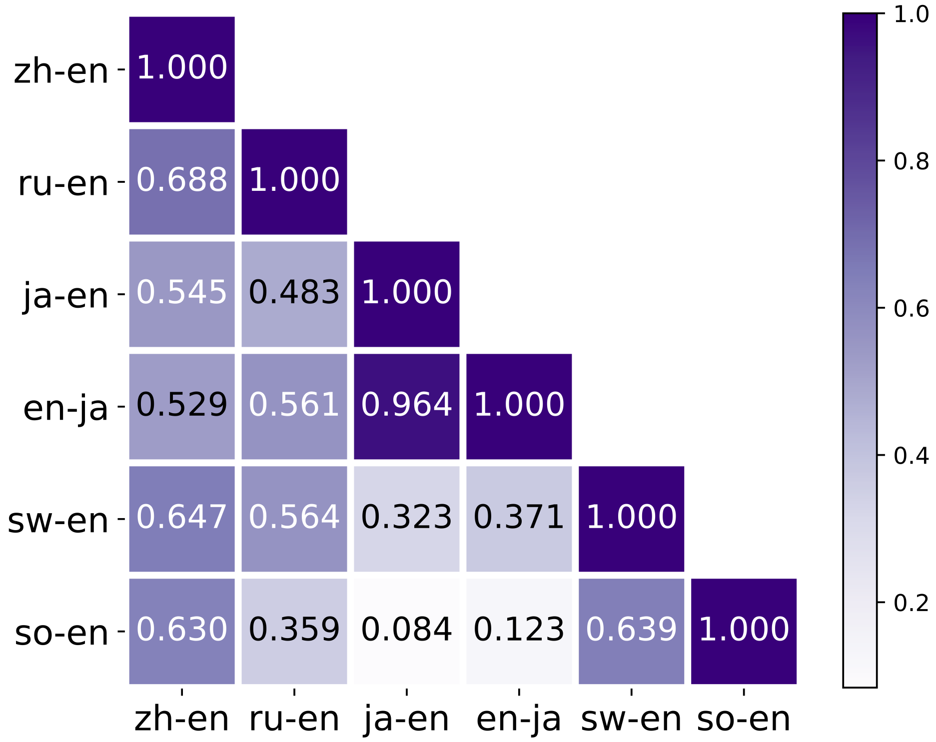


Figure 3.5: Correlation of hyperparameter rankings across MT datasets. The ranking is computed only on the subset of MT systems common in all datasets. For this, we consider 30k bpe (for zh, ru, ja, en) to be equivalent to 32k bpe (for sw, so).

We might be interested in seeing whether good configurations are always good across datasets. This can be done by ranking configurations by BLEU for each dataset and then measuring the correlation between rankings. We show the Spearman's correlation coefficient in Figure 3.5. NMT systems with the same language pairs (ja-en[9] vs. en-ja) are highly correlated. On the contrary, other pairs show low correlation (0.084 for ja-en vs. so-en), implying the need to run HPO on new datasets separately.

We also find that datasets within the same domain and of similar sizes can result

[9]In the following, when language pairs are mentioned without the prefix 'scratch' or 'ft', they refer to the trained-from-scratch datasets.

in low transferability of hyperparameter configuration rankings, as seen in zh-en vs. ru-en and sw-en vs. so-en. Notably, Chinese and Russian, as well as Swahili and Somali, do not belong to the same language family. This raises the intriguing question of whether languages within the same family might exhibit higher transferability. Additionally, the strong correlation observed for ja-en vs. en-ja may indicate that reversed translation directions within the same language pair often transfer well. Further experiments are needed to explore how factors like dataset size, domain, and language pairs influence hyperparameter transferability. Transfer learning for HPO could be a promising avenue, assuming that some correlations exist that, while not immediately evident, could potentially be learned with sufficient data.

Figure 3.6 illustrates the overlap coefficients between the top 20% configurations for each dataset pair. The overlap coefficient for two sets A and B is defined as:

$$\text{Overlap Coefficient} = \frac{|A \cap B|}{\min(|A|, |B|)}, \tag{3.1}$$

where $|A| = |B|$ in our case. This metric quantifies the similarity between two sets and how much they overlap. The figure reveals that, for most dataset pairs, the top-performing systems overlap by less than 50%. Notably, there is no overlap at all between so-en and ja-en. Comparing this with Figure 3.5, we see that a high correlation in the overall search space does not necessarily translate to a high overlap among the top-performing hyperparameter configurations. For instance, while so-en

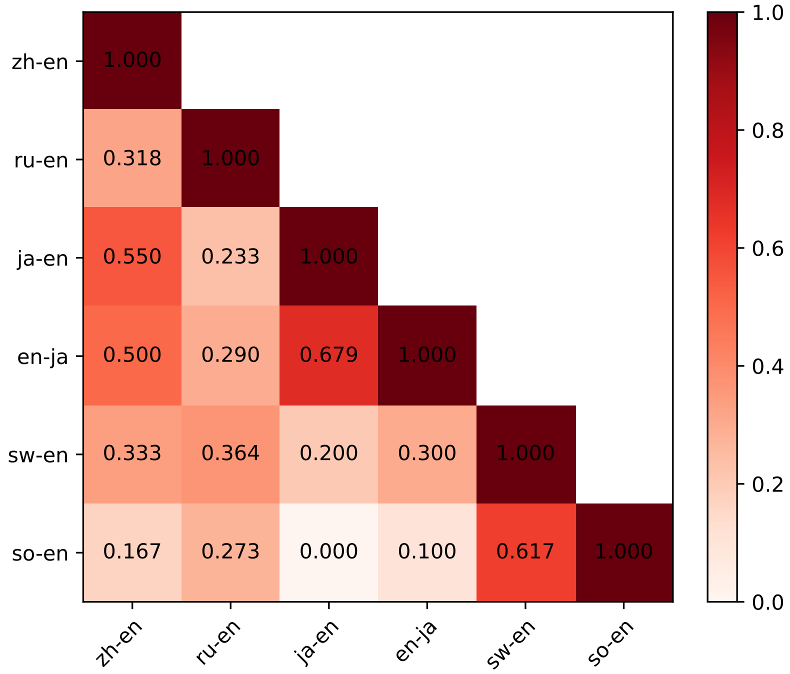


Figure 3.6: Overlap coefficients (computed by the size of the intersection of the two sets divided by the total size) of the 20% top-performing systems across MT datasets. The ranking is computed only on the subset of MT systems common in all datasets. For this, we consider 30k bpe (for zh, ru, ja, en) to be equivalent to 32k bpe (for sw, so).

and zh-en exhibit a ranking correlation of 0.630 across the entire search space (Figure 3.5), their overlap coefficient for the top 20% configurations is only 0.167 (Figure 3.6), indicating low transferability.

3.3.2.4 Hyperparameter Importance

The table-lookup approach also enables in-depth analyses of how hyperparameters generally affect system performance. Following Klein and Hutter (2019), we assess the importance of hyperparameters with fANOVA, which computes the variation in BLEU when changing a specific hyperparameter with values of all the other hyperparameters

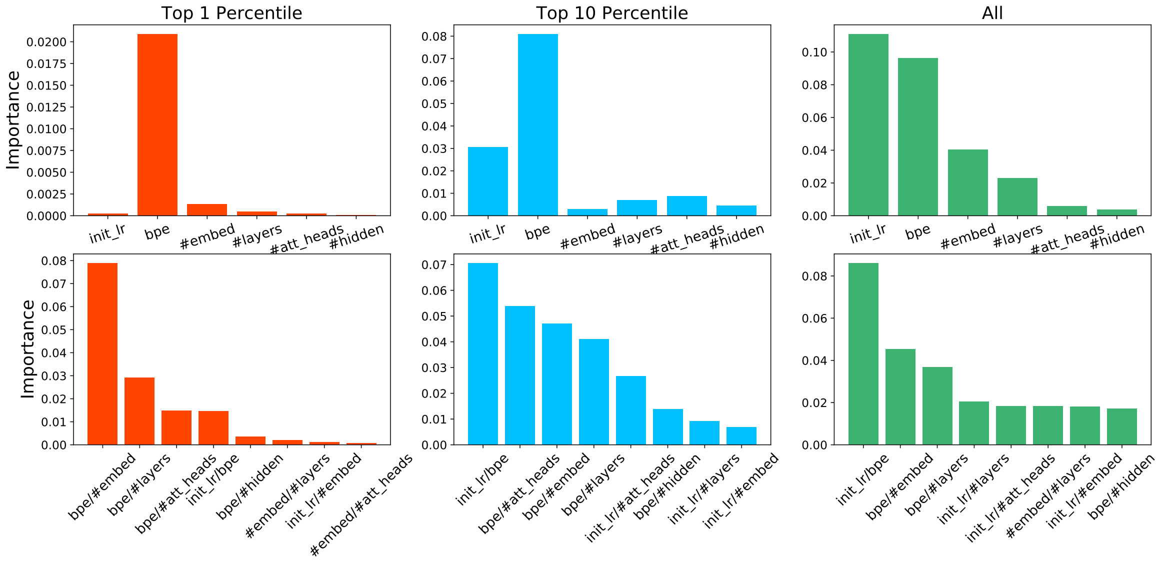


Figure 3.7: The importance of each hyperparameter (top) and the 8 most important hyperparameter pairs (bottom) for top 1% (left), top 10% (middle), and all NMT models (right) ranked by BLEU on en-ja.

fixed. In Figure 3.7, on en-ja, when considering only the top performing NMT models (top left), #att_heads, init_lr and #embed impact BLEU the most. While over the entire configuration space (top middle), #embed is the distinguishing factor. The analysis can be extended to pairs of hyperparameters, where we observe the interaction of init_lr and #embed being important (Figure 3.7 bottom left).

Questions may arise over whether the results on en-ja can be taken as general conclusions. We find that it is dataset-dependent — hyperparameter importance ranking differs across language pairs, and is dependent on the range and granularity of hyperparameters considered. As shown in the right column of Figure 5, bpe is the most important hyperparameter for sw-en, instead of #embed. This shows the diversity of our selected MT datasets and the hyperparameter importance analysis is a good tool for probing the search space characteristics of these datasets.

3.3.2.5 Effect of Random Seeds

NMT training might not be deterministic due to the random initialization of model parameters. All the experimental results so far are obtained by a single run using one random seed. In order to explore the variance of the model performance induced by initialization effects, we fix the hyperparameter configurations and train models initialized with various random seeds. Specifically, we select five hyperparameter configurations,[10] and re-trained them for additional five times each with different random initializations. We did this for two datasets: the low-resource sw-en task and the larger WMT2019 ja-en task.

The results on ja-en and sw-en are shown in Figure 3.8. The variance of performance is kept in a small range in most cases and the ranking of configurations remains about the same when different random seeds are applied. Based on this observation, we think that it is a reasonable strategy to use a single run to build table-lookup datasets; but at the same time it should be understood that the BLEU scores in the lookup table are only approximations. We note that there can be a few cases where variance is large, and this might be best addressed by inventing HPO methods that explicitly accounts for such uncertainty.

[10] Four of these are randomly selected. We also include the configuration that achieved the best BLEU in Table 3.2.

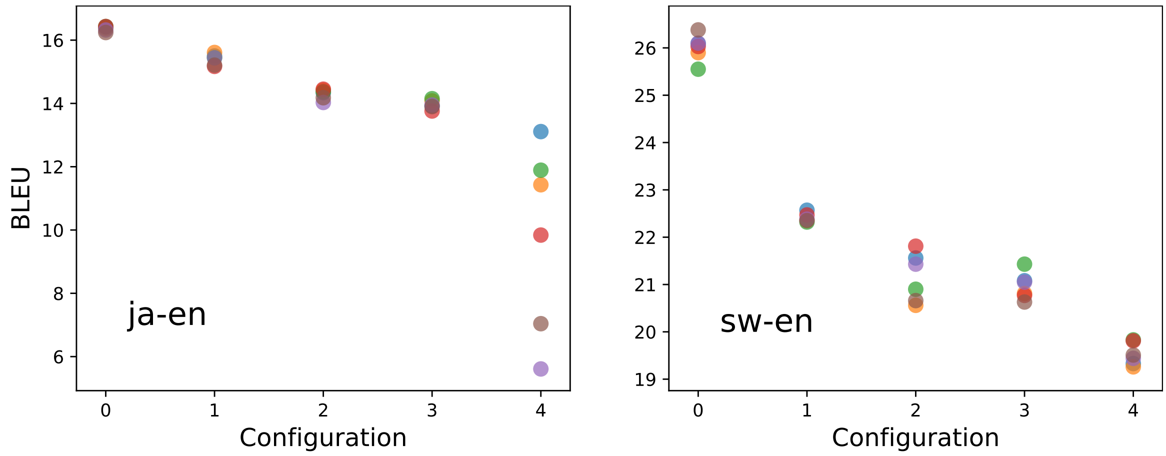


Figure 3.8: BLEU of ja-en and sw-en models trained with 6 random seeds. Circles with different colors stand for different random seeds.

3.4 Evaluation Protocols

To assess HPO method performance, we measure the **runtime** to reach a *quality indicator* (e.g. BLEU) target value. The **runtime** is defined as the number of NMT models trained, or equivalently the number of function evaluations. We consider two ways to measure the HPO performance: **fixed-target** and **fixed-budget**.

3.4.1 Single-objective Evaluation Metrics

For single-objective optimization, we have:

- **fixed-target best (ftb)**: We fix the quality indicator value to the best value in the dataset and measure runtime to reach this target.
- **fixed-target close (ftc)**: We measure the runtime to reach a target that is slightly less than the oracle best. This is useful when one can tolerate some

performance loss.

- **fixed-budget (fb)**: We fix the budget of function evaluations and measure the difference between the oracle best quality indicator value (e.g. oracle best BLEU) in the dataset vs. the best value achieved by systems queried by the HPO method.

The fixed-budget metric asks what is the best possible system assuming a hard constraint on training resources. The fixed-target metrics ask how much training resources is needed to find the best (or approximate best) system in the dataset.

3.4.2 Multi-objective Evaluation Metrics

In practice, one might desire to optimize multiple objectives, such as translation accuracy and speed. Suppose we have J objectives and they can be jointly represented as $F(\boldsymbol{\lambda}) = [f^1(\boldsymbol{\lambda}), f^2(\boldsymbol{\lambda}), \cdots, f^J(\boldsymbol{\lambda})]$, where λ is a hyperparameter configuration. As it is unlikely that any one $\boldsymbol{\lambda}$ will optimize all objectives simultaneously, we adopt the concept of *Pareto optimality* (Godfrey et al., 2007). In the context of minimization, $\boldsymbol{\lambda}$ is said to dominate $\boldsymbol{\lambda}'$, i.e. $\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$, if $f^j(\boldsymbol{\lambda}) \leq f^j(\boldsymbol{\lambda}')\ \forall j$ and $f^j(\boldsymbol{\lambda}) < f^j(\boldsymbol{\lambda}')$ for at least one j. If nothing dominates $\boldsymbol{\lambda}$, we call it *Pareto optimal* solution. The set of all Pareto solutions is referred to as *Pareto front*, i.e. $\{\boldsymbol{\lambda} \mid \nexists \boldsymbol{\lambda}' \in\ : \boldsymbol{\lambda}' \prec \boldsymbol{\lambda}\}$. Intuitively, these are solutions satisfying all possible tradeoffs in the multi-objective space. Figure 3.1 shows an example of Pareto solutions that maximize BLEU and minimize decoding time.

For multi-objective optimization, the quality indicator becomes the Pareto front, thus we have:

- **fixed-target all (fta)**: We measure the runtime to find all points on the Pareto front.
- **fixed-target one (fto)**: We measure the runtime to get at least one Pareto point.
- **fixed-budget (fbp)**: We fix the budget of function evaluations and measure the number of Pareto-optimal points obtained.

In the literature, a common way to compare HPO methods is to plot quality indicator value as a function of runtime on a graph. The proposed metrics can be viewed as summary statistics drawn as line thresholds on such graphs (Hansen et al., 2016), where the budget/target is set to a value appropriate for the use case.

3.4.3 Repeated Trials

Some HPO methods may be sensitive to randomness in initial seeds (Feurer et al., 2015b). We suggest that repeated randomized trials are important for a rigorous evaluation, and this is only feasible with a table-lookup dataset. To be specific, in the evaluation and comparison of various HPO methods, we suggest average results of HPO runs across 100 trials, where each trial is seeded with a different set of 3 random initial hyperparameter settings.

3.5 Related Work

To alleviate the computational burden for benchmarking HPO methods and to improve research reproducibility, several works have explored the table-lookup framework. Klein and Hutter (2019) published a mix of datasets focusing on feed-forward neural networks. Ying et al. (2019) released a dataset of convolutional architectures for image classification problems. Zöller and Huber (2021) evaluated selected HPO algorithms and AutoML frameworks on 137 data sets. To the best of our knowledge, this work is the first that focuses on NMT with the Transformer models and LLMs.

One challenge with table lookup is that sufficient coverage of the hyperparameter grid is assumed. Eggensperger et al. (2015) and Klein et al. (2019) propose using a predictive meta-model trained on a table-lookup benchmark. This approach can approximate hyperparameters not present in the table, and it could also be applied to our benchmark dataset to extrapolate missing values. Similarly, Zela et al. (2020) create cheap surrogate benchmarks for NAS. This is an interesting avenue for future work.

3.6 Conclusions

In this chapter, we presented a benchmark dataset for HPO of NMT systems, **NMTLC**. It is built by utilizing a table-lookup approach, where hyperparameter configurations are pre-evaluated and recorded in a table. When running HPO, developers can refer to this table instead of running evaluations on an NMT model, enabling fast and efficient HPO evaluation. **NMTLC** also allows for extensive analysis of NMT models across a large hyperparameter configuration space. We demonstrated that the BLEU score distribution and the need to optimize multiple objectives make manual tuning challenging, highlighting the necessity of HPO. Additionally, we showed that the ranking and importance of hyperparameter configurations do not transfer across different MT tasks. Therefore, when evaluating or comparing HPO methods, it is crucial to ensure their robustness across various MT datasets. We also provided multiple evaluation protocols for comparing HPO methods in both single-objective and multi-objective optimization, emphasizing the importance of repeated trials. We hope that this dataset will facilitate reproducible research and rigorous evaluation of HPO for complex and expensive models.

Chapter 4

Graph-based Hyperparameter Optimization

The NMTLC benchmark datasets introduced in Chapter 3 facilitate the development of new hyperparameter optimization (HPO) methods and enable their evaluation against existing HPO approaches. In this chapter, we introduce a novel HPO algorithm based on graph-based semi-supervised learning. We benchmark this new algorithm on NMTLC against established HPO algorithms, including random search and Bayesian optimization.

We begin by discussing the background and motivation for introducing the graph-based HPO algorithm in Section 4.1. In Section 4.2, we present a general HPO framework—sequential model-based optimization. Both Bayesian optimization and the newly proposed graph-based HPO follow this framework. Our methods are evaluated on NMTLC in Section 4.3 and Section 4.4. Finally, we conclude the chapter in Section 4.5.

4.1 Introduction

Well-established HPO algorithms iteratively propose and evaluate hyperparameter configurations to optimize machine learning models (Bischl et al., 2023). These algorithms can generally be classified into two categories: model-free and model-based. Model-free methods, such as grid and random search, do not make assumptions about the models being optimized. However, they suffer from the curse of dimensionality: as the dimensionality of the configuration space increases, grid search requires exponentially more evaluations, and random search may not adequately cover the space.

In contrast, model-based HPO methods, like Bayesian optimization, use a surrogate model to make informed decisions, thus reducing the number of required evaluations. These surrogate models, such as Gaussian processes in Bayesian optimization, are fitted on hyperparameter configurations and their corresponding performance metrics. They predict the performance of new configurations and are typically lighter than the original machine learning model. The setup of HPO is akin to that in semi-supervised learning where labeled data is scarce and expensive but unlabeled data is abundant.

Semi-supervised learning aims to utilize a small labeled dataset alongside a large unlabeled dataset to improve prediction accuracy with minimal annotation effort. Graph-based semi-supervised learning is a common approach within this domain. It constructs a graph where nodes represent labeled and unlabeled data points, and edges reflect their similarities. The assumption is that nodes connected by high-weight edges are likely to share the same label, allowing labels to propagate through the graph, benefiting from the properties of spectral graph theory.

In graph-based semi-supervised learning, label smoothness is enforced over the graph, ensuring that neighboring nodes tend to have the same label. Applied to HPO, this approach assumes that similar hyperparameters yield comparable evaluation metrics. A graph can thus be constructed on the configuration space, with configurations as nodes and evaluation metrics as labels.

Additionally, graph-based semi-supervised learning can be combined with active learning, which iteratively selects unlabeled instances to be labeled. This combination incrementally increases the labeled dataset, enhancing the learning process.

In this chapter, we introduce a novel HPO method that combines graph-based semi-supervised learning with active learning. We formulate this method within the framework of sequential model-based optimization and compare it to Bayesian optimization. Furthermore, we extend our approach to multi-objective optimization. Our method is benchmarked on the NMTLC dataset, demonstrating comparable performance to Bayesian optimization and significantly outperforming random search.

4.2 Methodology

In this section, we first formulate the HPO problem (Section 4.2.1). We then introduce the general framework of sequential model-based optimization (Section 4.2.1), and discuss a typical HPO method that follows this framework, Bayesian optimization (Section 4.2.2). Finally, we present the graph-based HPO method (Section 4.2.3) and outline its connections and key differences with Bayesian optimization (Section 4.2.4).

4.2.1 Sequential Model-Based Optimization

Given a machine learning algorithm with H hyperparameters, we denote the domain of the h-th hyperparameter by Λ_h and the overall hyperparameter configuration space as $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \dots \Lambda_H$. When trained with a hyperparameter setting $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ on data $\mathcal{D}_{train}$, the algorithm's performance metric on some validation data $\mathcal{D}_{valid}$

is $f(\boldsymbol{\lambda}) := \boldsymbol{\mathcal{V}}(\boldsymbol{\lambda}, \mathcal{D}_{train}, \mathcal{D}_{valid})$. In the context of NMT, $f(\cdot)$ or $\boldsymbol{\mathcal{V}}(\cdot)$ could be the perplexity, translation accuracy (e.g. BLEU score), or decoding time on $\mathcal{D}_{valid}$. In general, $f(\cdot)$ is computationally expensive to obtain; it requires training a model to completion, then evaluating some performance metric on a validation set. For purposes of exposition, we assume that lower $f(\cdot)$ is better, so we might define $f(\cdot)$ as 1 - BLEU.

The goal of hyperparameter optimization is then to find a $\boldsymbol{\lambda}^{\star} = \arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} f(\boldsymbol{\lambda})$, with as few evaluations of $f(\cdot)$ as possible. An HPO problem can be challenging for three reasons: (a) $\boldsymbol{\Lambda}$ may be a combinatorially large space, prohibiting grid search over hyperparameters . (b) $f(\cdot)$ may be expensive to compute, so there is a tight budget on how many evaluations of $f(\cdot)$ are allowed. (c) f is not a continuous function and no gradient information can be exploited, forcing us to view the arg min as a blackbox discrete search problem. NMT HPO search exhibits all these conditions.

One class of algorithms that tackle the HPO problem is sequential model-based optimization (SMBO), illustrated in Figure 4.1. SMBO approximates f with a cheap-to-evaluate surrogate model $\hat{f}$ (Feurer and Hutter, 2019b; Luo, 2016; Jones et al., 1998). SMBO starts by querying f with initial hyperparameters $\{\boldsymbol{\lambda}_{init}\}$ and recording the resulting $(\boldsymbol{\lambda}_{init}, f(\boldsymbol{\lambda}_{init}))$ pairs. Then, it iteratively 1) fits the surrogate $\hat{f}$ on pairs observed so far; 2) gets the predictions $\hat{f}(\boldsymbol{\lambda}_i)$ for unlabeled/unobserved hyperparameters; 3) selects a promising $\boldsymbol{\lambda}_p$ to query next based on these predictions and an acquisition function, whose role is to trade off exploration in $\boldsymbol{\Lambda}$ with high model uncertainty and exploitation in $\boldsymbol{\Lambda}$ with low $\hat{f}(\cdot)$.

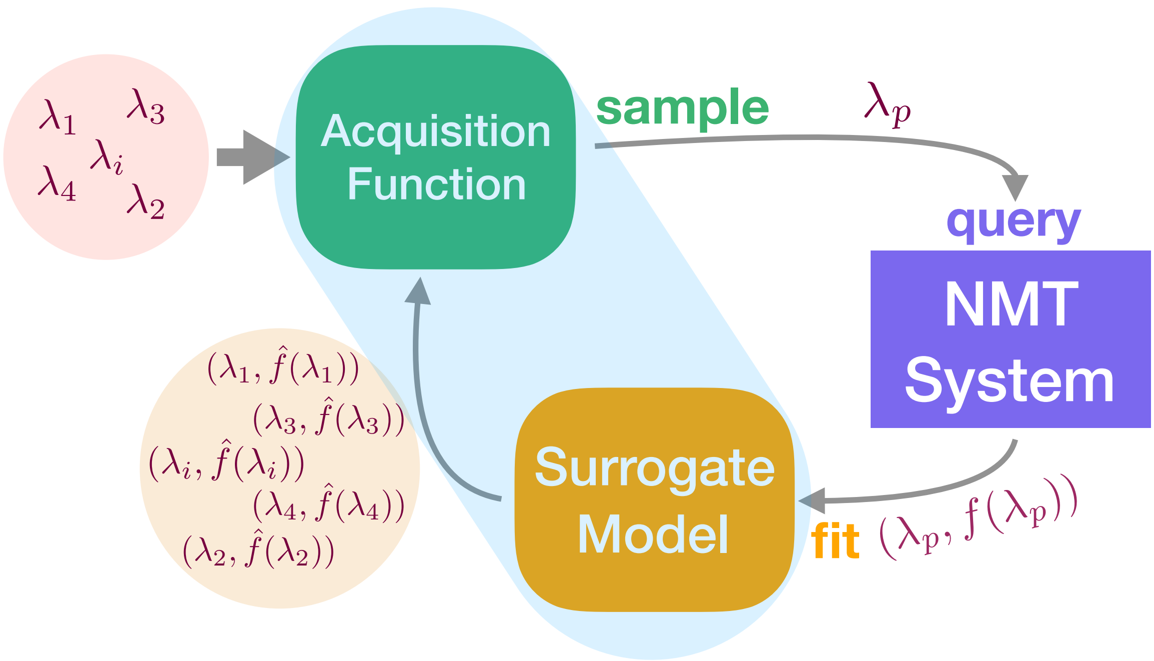


Figure 4.1: Sequential Model-Based Optimization (SMBO) framework for HPO. The part shaded in light blue contains two ingredients required for implementing an SMBO method: the surrogate model and the acquisition function. SMBO works iteratively: it starts by querying the function f with a hyperparameter configuration $\boldsymbol{\lambda}_p$ to get $f(\boldsymbol{\lambda}_p)$. The surrogate model then fits on all the evaluated configurations and generates predictions accordingly for all the unseen data points $(\boldsymbol{\lambda}_i, \hat{f}(\boldsymbol{\lambda}_i))$. Based on the predictions, the acquisition function then selects the most promising configuration to query the NMT system in the next iteration.

Evolutionary algorithms (Eberhart and Shi, 1998; Simon, 2013) are also used to solve HPO problems. Unlike SMBO, they do not approximate f with a surrogate $\hat{f}$; rather, they directly sample hyperparameters with high $f(\cdot)$ from a population and recombine them to form the next query.

4.2.2 Bayesian Optimization

Given a target function $f : \boldsymbol{\Lambda} \to \mathbb{R}$, Bayesian optimization (Brochu et al., 2010; Shahriari et al., 2015; Frazier, 2018) aims to find an input $\boldsymbol{\lambda}^\star = \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} f(\boldsymbol{\lambda})$. It models f with a posterior probability distribution $p(f \mid \boldsymbol{\mathcal{L}})$, where $\boldsymbol{\mathcal{L}}$ is a set of observed points. This posterior distribution is updated each time we observe f at a new point $\boldsymbol{\lambda}_p$. The *utility* of each candidate point is quantified by an acquisition function $a : \boldsymbol{\Lambda} \to \mathbb{R}$, and $\boldsymbol{\lambda}_p \in \arg\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} a(\boldsymbol{\lambda})$. In practice, a prominent choice for $p(f \mid \boldsymbol{\mathcal{L}})$ is Gaussian process regression, and a common acquisition function is Expected Improvement (EI).

4.2.2.1 Gaussian Process Regression

Gaussian Process (GP) regression (Rasmussen, 2003) is a Bayesian statistical approach for modeling functions. We describe GP by focusing on a collection of points $[\cdots, \boldsymbol{\lambda}, \cdots]$ with function values $[\cdots, f(\boldsymbol{\lambda}), \cdots]$. These points are supposed to be drawn from a prior probability distribution. GP assumes this prior distribution to be a multivariate Gaussian distribution. A GP $\mathcal{G}(\mu(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}'))$ is fully specified by a mean function $\mu(\boldsymbol{\lambda})$ and a covariance function or a *kernel* $\Sigma(\boldsymbol{\lambda}, \boldsymbol{\lambda}') = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, which measures the similarity between two points $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$. The kernel is chosen so that points $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$ that are closer in the input space have a large positive correlation, encoding the belief that they should have more similar function values $f(\boldsymbol{\lambda})$ and $f(\boldsymbol{\lambda}')$ than points that are far apart.

Suppose we have a collection of input points $\boldsymbol{\lambda}_{1:k}$ indicating the sequence $\boldsymbol{\lambda}_1, \cdots, \boldsymbol{\lambda}_k$, and we have observed their corresponding function values $f(\boldsymbol{\lambda}_{1:k}) = [f(\boldsymbol{\lambda}_1), \cdots, f(\boldsymbol{\lambda}_k)]$. The resulting prior distribution is

$$f(\boldsymbol{\lambda}_{1:k}) \sim \mathcal{N}(\mu_0(\boldsymbol{\lambda}_{1:k}), \Sigma_0(\boldsymbol{\lambda}_{1:k}, \boldsymbol{\lambda}_{1:k})), \tag{4.1}$$

where

$$\begin{aligned} \mu_0(\boldsymbol{\lambda}_{1:k}) &= [\mu_0(\boldsymbol{\lambda}_1), \cdots, \mu_0(\boldsymbol{\lambda}_k)], && (4.2) \\ \Sigma_0(\boldsymbol{\lambda}_{1:k}, \boldsymbol{\lambda}_{1:k}) &= \begin{bmatrix} \Sigma_0(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_1) & \cdots & \Sigma_0(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_k) \\ \vdots & \ddots & \vdots \\ \Sigma_0(\boldsymbol{\lambda}_k, \boldsymbol{\lambda}_1) & \cdots & \Sigma_0(\boldsymbol{\lambda}_k, \boldsymbol{\lambda}_k) \end{bmatrix}. && (4.3) \end{aligned}$$

Given the observed training data, one can make predictions for unseen test data by computing the posterior following Bayes' rule. Let $\boldsymbol{f}_0$ be the known function values of the training cases, and let $\boldsymbol{f}_*$ be a set of function values corresponding to the test set inputs. The joint distribution would be expressed as:

$$\begin{bmatrix} \boldsymbol{f}_0 \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu}_0 \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \Sigma_0 & \Sigma_* \\ \Sigma_*^T & \Sigma_{**} \end{bmatrix}), \tag{4.4}$$

where $\boldsymbol{\mu}_0$ stands for the training means and $\boldsymbol{\mu}_*$ for the test means. For the covariance, Σ_0 is used for the training set covariances, Σ_* for training-test set covariances, and Σ_{**} for test set covariances. And we are interested in the conditional distribution of $\boldsymbol{f}_*$ given $\boldsymbol{f}$: $\boldsymbol{f}_* \mid \boldsymbol{f}$.

Given a joint Gaussian distribution:

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \sim \left(\begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \end{bmatrix}, \begin{bmatrix} \boldsymbol{A} & \boldsymbol{C} \\ \boldsymbol{C}^T & \boldsymbol{B} \end{bmatrix} \right), \tag{4.5}$$

the conditional distribution can be computed using this formula:

$$\boldsymbol{x} \mid \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{a} + \boldsymbol{C}\boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{b}), \boldsymbol{A} - \boldsymbol{C}\boldsymbol{B}^{-1}\boldsymbol{C}^T). \tag{4.6}$$

Thus, in our case,

$$\boldsymbol{f}_* \mid \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}_* + \Sigma_*^T \Sigma_0^{-1}(\boldsymbol{f} - \boldsymbol{\mu}_0), \Sigma_{**} - \Sigma_*^T \Sigma_0^{-1} \Sigma_*). \tag{4.7}$$

This means that the posterior mean is a weighted average between the prior $\boldsymbol{\mu}_*$ and an estimate based on the training data with a weight that depends on the kernel. The posterior variance is equal to the prior covariance Σ_{**} less a term that corresponds to the variance removed by observing the training data.

4.2.2.2 Expected Improvement

The EI score (Schonlau et al., 1998) is defined as:

$$a_{EI}(\boldsymbol{\lambda}) = \mathbb{E}[max(\hat{f}(\boldsymbol{\lambda}) - f_{min}), 0)], \tag{4.8}$$

where f_{min} is the best observed value thus far, and $\hat{f}(\boldsymbol{\lambda}) = \mu(\boldsymbol{\lambda})$. When the prediction $\hat{f}(\boldsymbol{\lambda})$ follows a normal distribution as in the GP, EI can be computed in a closed form. Our acquisition function computes EI for each point in the grid of hyperparameters and queries the one with the largest value.

4.2.3 Graph-Based Optimization

Semi-supervised learning addresses the question how to utilize a handful of labeled data and a large amount of unlabeled data to improve prediction accuracy. Graph-based semi-supervised learning (GBSSL, Zhu et al., 2003; Zhu, 2005) describes the structure of data with a graph, where each vertex is a data point and each weighted edge reflects the similarity between vertices. It makes a *smoothness* assumption that neighbors connected by edges tend to have similar labels, and labels can propagate throughout the graph.

In SMBO surrogate modeling, we hope to make predictions for the unlabeled or not-evaluated points in the hyperparameter space based on the information of labeled or evaluated points. If we pre-define the set of all potential points, then this becomes highly related to semi-supervised learning. From this point of view, we propose GBSSL equipped with suitable acquisition functions as a new SMBO method for searching over a grid of representative hyperparameter configurations.

4.2.3.1 Graph-Based Regression

Suppose we have a graph $G = (V, E)$ (Figure 4.2) with nodes V corresponding to n points, of which $\boldsymbol{\mathcal{L}}$ denotes the set of labeled points $\{(\boldsymbol{\lambda}_1, f(1)), \cdots, (\boldsymbol{\lambda}_l, f(l))\}$, where $f(i)$ is short for $f(\boldsymbol{\lambda}_i)$, and $\boldsymbol{\mathcal{U}}$ denotes the set of unlabeled points $\{\boldsymbol{\lambda}_{l+1}, \cdots, \boldsymbol{\lambda}_{l+u}\}$, where $n = l + u$. The edges E are represented by a $n \times n$ weight matrix W. For instance, W can be given as the radial basis function (RBF):

$$w_{ij} = \exp\left(-\frac{1}{2\sigma^2}\sum_d (\boldsymbol{\lambda}_{id} - \boldsymbol{\lambda}_{jd})^2\right). \tag{4.9}$$

Note G is not necessarily fully connected, in practice, kNN graphs with a small k turn out to perform well, where nodes i, j are connected if i is in j's k-nearest-neighborhood or vice versa.[1]

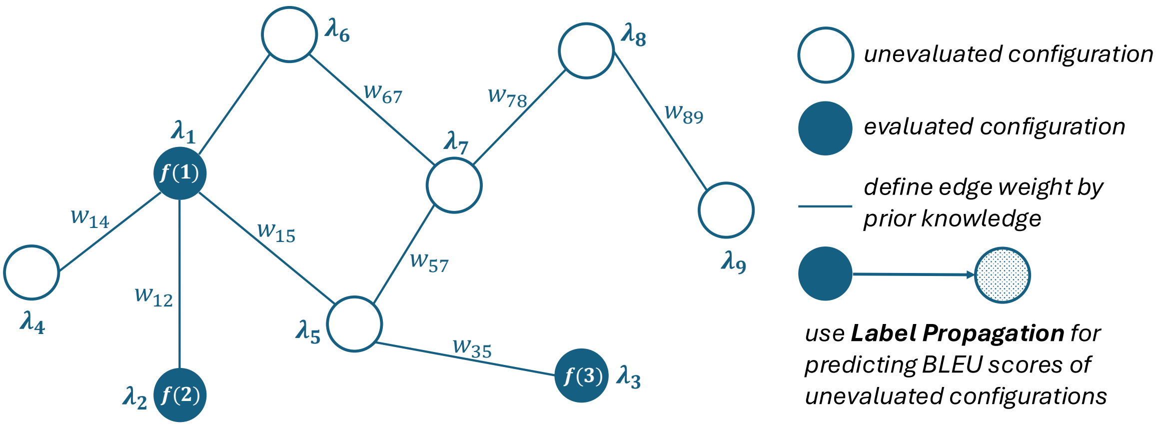


Figure 4.2: A graph on the hyperparameter search space. Nodes denote hyperparameter configurations, which are connected by weighted edges, where the weights are defined based on the similarity between neighbor nodes.

[1] In experiments, based on initial tuning, we set kNN so that each point has on average $\frac{n}{7}$ neighbors.

Since closer points are assumed to have similar labels, we define the *energy* function as:

$$E(f) = \frac{1}{2}\sum_{i,j} w_{ij}(f(i) - f(j))^2, \tag{4.10}$$

and we constrain $f(i)$, $i \in L$ or f_L to be true labels and aim to find $f(i)$, $i \in U$ or f_U that minimizes the energy. We define a diagonal matrix D, where $D_{ii} = \sum_j W_{ij}$ and the *combinatorial Laplacian* $\boldsymbol{\Delta} = D - W$, Equation 4.10 can then be rewritten to $E(f) = \boldsymbol{f}^T\boldsymbol{\Delta}\boldsymbol{f}$. The minimum energy function $f = \arg\min_{f_L=Y_L} E(f)$ satisfies $f_L = Y_L$ for labeled points L, and $\Delta f = 0$ on unlabeled data points U. This leads to

$$f(i) = \frac{1}{D_{ii}}\sum_{j\sim i} w_{ij} f(j), \text{ for } i \in U, \tag{4.11}$$

i.e.the function value of $f(i)$ at each unlabeled point i is the average of its neighbors in the graph. If we partition the Laplacian matrix into blocks:

$$\boldsymbol{\Delta} = \begin{bmatrix} \boldsymbol{\Delta}_{LL} & \boldsymbol{\Delta}_{LU} \\ \boldsymbol{\Delta}_{UL} & \boldsymbol{\Delta}_{UU} \end{bmatrix}, \tag{4.12}$$

the solution $\Delta f = 0$ subject to $f_L = Y_L$ is given by:

$$f_U = -\boldsymbol{\Delta}_{UU}^{-1}\boldsymbol{\Delta}_{UL} f_L. \tag{4.13}$$

We can then predict the function values for unlabeled points with the label propagation rule defined by Equation 4.13.

4.2.3.2 Expected Influence

We propose a novel acquisition function called *expected influence* that exploits the graph structure. The idea is to query the point such that, if its $f()$ is observed, has the highest potential to change the $f()$ of all other points as we re-run label propagation through the graph.

We first scale the labels on the graph $f(i) \in \mathbb{R}$ to be between 0 or 1. The best-labeled point is set to 1; for the other labeled points, we first compute the probability that a random walk starting at 1 reaches it, then set the label to be 1 if the probability is larger than 0.5 and 0 otherwise.

If we were to query an unlabeled point k, there are two scenarios: its label is either 1 with probability $f(k)$ or 0 with probability $1 - f(k)$. For each scenario, we then consider including k as a newly added "labeled" point and re-running label propagation. $f^{+(\boldsymbol{\lambda}_k,1)}(i)$ are the new predictions for points i in the scenario where k is added with label 1. If k is an influencer in the positive direction, this means that many points i will now have large $f^{+(\boldsymbol{\lambda}_k,1)}(i)$; otherwise, $f^{+(\boldsymbol{\lambda}_k,1)}(i)$ might be small on average in magnitude. On the other hand, suppose we add k with label 0 and run label propagation again to obtain new predictions $f^{+(\boldsymbol{\lambda}_k,0)}(i)$. If k is an influencer in the negative direction, this means that $f^{+(\boldsymbol{\lambda}_k,0)}(i)$ will be small (or conversely $1 - f^{+(\boldsymbol{\lambda}_k,0)}(i)$ will be large).

We can now define an influence score and have the acquisition function seeking point p that maximizes the following:

$$\begin{aligned} a_{EIF}(\boldsymbol{\lambda}_k) &= (1 - f(k)) \sum_{i=1}^{n} (1 - f^{+(\boldsymbol{\lambda}_k, 0)}(i)) \\ &\quad + f(k) \sum_{i=1}^{n} f^{+(\boldsymbol{\lambda}_k, 1)}(i) \end{aligned} \tag{4.14}$$

Intuitively, we try adding each unlabeled point as either a desirable point (label 1) or an undesirable point (0). We measure whether this addition changes the result of GB regression, and finally query the hyperparameter that is expected to cause the most significant change.

4.2.4 Bayesian Optimization vs. Graph-Based Optimization

There is a connection between the BO and GB due to the link between GPs and graphs. The GB method defines a Gaussian random field on the graph, which is a multivariate Gaussian distribution on the nodes. This is equivalent to "finite set" GPs. Zhu (2005) showed that the kernel matrix K of the finite set GP is equivalent to the inverse of a function of the graph Laplacian $\boldsymbol{\Delta}$, i.e. $K = (2\beta(\boldsymbol{\Delta} + \frac{I}{\sigma^2}))^{-1}$.[2] The difference between the finite set GP and GP is that the kernel matrix of the former is defined on $\boldsymbol{\mathcal{L}} \cup \boldsymbol{\mathcal{U}}$, while the latter is defined on $\boldsymbol{\Lambda}$. As a semi-supervised method, the

[2] β and σ are adjustable parameters.

label propagation rule of GB (Equation 4.13) shows that all the nodes on the graph contribute to the prediction of a single unlabeled node. While for GP, the posterior predictive distribution of a new point does not depend on other unlabeled points as shown by Equation 4.7.

The main advantage of GB is that it offers flexibility to build graphs over the search space. For instance, one can build a graph with configurations from different model architectures, e.g. RNN, CNN and Transformers. Nodes of the same architecture might gather into a cluster, and clusters can be connected with each other. One can also manipulate the edge weights by manually defined heuristics. One example of such rules could be Euclidean distance scaled by hyperparameter importance. We leave this as future work.

The theoretical caveat of GB method is that it is restricted to a discrete search space defined by a graph. If a dense grid is desired to mimic a continuous search space, increasing time and space complexity would make it a less efficient method.

4.2.5 Multi-Objective Optimization

For multi-objective optimization, we can use the same surrogate models to estimate each $\hat{f}^j$ independently; but we need a new acquisition function that considers the Pareto front. Various methods have been proposed (Zitzler and Thiele, 1998; Ponweiser et al., 2008; Picheny, 2015; Shah and Ghahramani, 2016; Svenson and Santner, 2016). Here, we adopt the expected hypervolume improvement (EHVI) method (Emmerich et al.,

Algorithm 2 Multi-objective SMBO

Input: Initial seeds $\{\boldsymbol{\lambda}_{init}\}$, Budget B

Output: Pareto-front approximation $\boldsymbol{\mathcal{P}}$, $\boldsymbol{\mathcal{L}} \leftarrow \{\cdots(\boldsymbol{\lambda}_{init}, F(\boldsymbol{\lambda}_{init}))\cdots\}$

While $b <= B$ **do**
  $\boldsymbol{\mathcal{P}} \leftarrow$ Compute the Pareto front of $\boldsymbol{\mathcal{L}}$
  Fit surrogate models $\hat{f}^1, \cdots, \hat{f}^J$ on $\boldsymbol{\mathcal{L}}$
  Select a new point $\boldsymbol{\lambda}_p$ based on an infill criterion and surrogate model predictions
  $\boldsymbol{\mathcal{L}} \leftarrow \boldsymbol{\mathcal{L}} \cup \{(\boldsymbol{\lambda}_p, F(\lambda_p))\}$
end

return $\boldsymbol{\mathcal{P}}$

2011), which is a generalization of EI. EHVI as an *infill criterion* and can be combined with different surrogate models. Algorithm 2 provides pseudo-code for the framework.

4.3 Experiments

We evaluate HPO methods on six NMT tasks (trained-from-scratch NMT models) with the NMTLC benchmark dataset and report their performance measured by three runtime-based assessment metrics mentioned in Section 3.4. The code base is provided to ensure reproducibility: `https://github.com/Este1le/gbopt`.

4.3.1 Single-Objective Optimization

For single-objective optimization, our goal is to find a hyperparameter configuration giving the highest BLEU score over a pre-defined grid. We run the comparison with two surrogate models, two kernels, and two acquisition functions. We choose Matérn52 and RBF kernel because they exhibit different properties and are both frequently used in literature. As shown in Rasmussen (2003), a parameter ν of the Matérn class of covariance functions can affect the smoothness of the functions drawn from GP. For $\nu = \frac{1}{2}$, the process becomes very rough, and for $\nu \to \infty$, the covariance function converges to RBF kernel. This leads to the following HPO systems, where all the GB systems are introduced by this work:

- **RS**: random search (Bergstra and Bengio, 2012) which uniformly samples hyperparameter configurations at random over the grid.
- **BO_EI_M**: GP-based BO with Matérn52 covariance function and expected improvement as acquisition function.
- **BO_EI_R**: GP-based BO with RBF kernel and EI as acquisition function.
- **GB_EI_M**: GB with Matérn52 kernel and EI as acquisition function.[3]
- **GB_EI_R**: GB with RBF kernel and EI.
- **GB_EIF_M**: GB with Matérn52 kernel and expected influence as acquisition function.
- **GB_EIF_R**: GB with RBF and EIF.

We use the George library (Ambikasaran et al., 2014) for GP implementation. For all the methods, configurations are sampled without replacement.

[3] We can make an equivalence between the covariance matrix in multivariate Gaussian distribution and the inverse of a function of the graph Laplacian $\boldsymbol{\Delta}$ (see Section 4.2.4 for details), so EI can also be applied to GB models.

Results for single-objective optimization are summarized in Table 4.1:

- RS always needs to explore roughly half of all the NMT models to get the best one (ftb).
- The effectiveness of BO is confirmed: on sw-en, BO_EI_M only takes 10% of the runtime used by RS to achieve the optima.
- For ftb, the best GB outperforms the best BO on 4 of the 6 datasets: on en-ja, GB_EI_M reduces the ftb runtime of BO_EI_M by 38. GB_EIF often works better than GB_EI.
- Matérn kernel and RBF kernel are almost equally good for both BO and GB.
- Adjusting initialization can result in a noticeable variance on performance. We suggest that researchers experiment with enough random trials when evaluating HPO systems.

4.3.2 Multi-Objective Optimization

We now show multi-objective optimization on the NMTLC benchmark dataset. Our goal is to search for configurations achieving higher BLEU and less decoding time. We run the comparison on the following systems, where GB systems are introduced by this work:

- **RS**: random search, uniformly samples the configurations at random.

	zh-en			ru-en		
	ftb	**ftc**	**fb**	**ftb**	**ftc**	**fb**
RS	61±34	14±11	0.26±0.25	79±47	20±17	0.42±0.29
BO_EI_M	29±19	13±9	0.24±0.24	41±19	26±17	0.51±0.36
BO_EI_R	24±15	11±8	0.22±0.26	40±26	20±13	0.44±0.37
GB_EI_M	84±15	13±8	0.35±0.21	50±34	18±17	0.35±0.25
GB_EI_R	86±15	12±7	0.33±0.20	51±32	18±17	0.35±0.28
GB_EIF_M	19±21	8±5	0.11±0.17	32±18	22±13	0.46±0.31
GB_EIF_R	**13**±20	**6**±4	**0.06**±0.15	**28**±17	**17**±12	**0.33**±0.30

	en-ja			ja-en		
	ftb	**ftc**	**fb**	**ftb**	**ftc**	**fb**
RS	71±46	12±10	0.71±0.37	71±43	16±15	0.40±0.24
BO_EI_M	60±29	15±17	0.86±0.60	27±17	16±15	0.39±0.45
BO_EI_R	62±36	13±12	0.79±0.58	20±11	13±9	0.33±0.44
GB_EI_M	**22**±20	11±11	**0.42**±0.57	23±7	**6**±3	0.14±0.11
GB_EI_R	24±21	13±12	0.47±0.59	21±6	**6**±3	0.10±0.12
GB_EIF_M	47±22	**9**±7	0.63±0.32	**13**±4	**6**±2	**0.01**±0.04
GB_EIF_R	45±22	10±7	0.69±0.39	**13**±3	**6**±2	**0.01**±0.05

	sw-en			so-en		
	ftb	**ftc**	**fb**	**ftb**	**ftc**	**fb**
RS	334±201	186±152	2.45±0.97	301±161	39±39	0.63±0.32
BO_EI_M	**33**±17	**29**±17	1.60±1.41	65±62	19±21	0.41±0.36
BO_EI_R	55±47	33±24	**1.42**±1.33	52±70	**13**±11	**0.24**±0.30
GB_EI_M	63±37	62±36	3.56±0.95	187±99	61±28	1.17±0.44
GB_EI_R	56±26	55±26	3.39±0.95	201±104	62±29	1.16±0.44
GB_EIF_M	58±24	57±24	3.13±0.51	**42**±30	26±8	0.48±0.13
GB_EIF_R	59±25	58±25	3.15±0.52	**42**±30	28±7	0.49±0.12

Table 4.1: Evaluation on NMT models trained with different language pairs for single-objective (BLEU) optimization. Results are averaged over 100 trials and standard deviations are also reported. Fixed-target best (ftb) and fixed-target close (ftc) are measured by a number of model evaluations, and fixed-budget (fb) is measured by BLEU difference. For ftc, the tolerance of performance degradation is set to 0.5 BLEU. (Except for en-ja where tolerance is set to 1 BLEU because the BLEU difference between the top 2 models is > 0.5.) For fb, the runtime budget is set to 20. (Including 3 initial evaluations.)

- **BO_M**: GP-based BO equipped with Matérn kernel and expected hypervolume improvement (EHVI) as the infill criterion.
- **BO_R**: GP-based BO with RBF kernel and EHVI.
- **GB_M**: GB equipped with Matérn kernel and EHVI as the infill criterion.
- **GB_R**: GB with RBF kernel and EHVI.

The multi-objective optimization evaluation results are summarized in Table 4.2:

- RS is a bad choice for multi-objective optimization, if one aims to quickly collect as many Pareto-optimal configurations as possible: to get all the true optima, RS usually needs to go through the whole search space (fta), and with fixed budget it obtains much fewer Pareto points than other methods (fbp).
- BO is generally superior across datasets. On sw-en, it only spends less than half of the time that RS takes to get the Pareto set (344 vs. 719), and can find 8.6 more Pareto points than RS with 200 NMT models evaluated.
- GB provides comparable performance as BO on 4 datasets. While on sw-en and so-en, BO noticeably outperforms GB, which might not be a perfect solution for multi-objective task.

	zh-en			**ru-en**		
	fto	**fta** (J=3)	**fbp** (B=50)	**fto**	**fta** (J=4)	**fbp** (B=50)
RS	30±24	88±22	1.3±0.8	33±26	139±28	1.3±0.9
BO_M	24±16	81±16	1.7±0.7	**16**±14	**80**±26	**2.4**±0.9
BO_R	**20**±13	**75**±15	**1.8**±0.5	17±15	84±32	**2.4**±1.0
GB_M	24±16	85±16	**1.8**±0.6	17±14	102±30	1.9±0.9
GB_R	24±15	90±12	1.7±0.6	17±12	103±30	2.0±0.9

	en-ja			**ja-en**		
	fto	**fta** (J=8)	**fbp** (B=50)	**fto**	**fta** (J=5)	**fbp** (B=50)
RS	17±16	150±17	2.5±1.4	21±18	129±20	1.7±1.0
BO_M	**15**±10	100±34	**4.6**±1.7	17±13	**77**±28	**3.3**±1.3
BO_R	17±13	**93**±30	4.3±2.0	18±14	94±32	2.8±1.2
GB_M	17±13	121±28	4.0±1.5	**16**±12	103±21	2.4±1.1
GB_R	17±14	119±24	3.6±1.5	19±12	107±20	2.2±1.0

	sw-en			**so-en**		
	fto	**fta** (J=14)	**fbp** (B=200)	**fto**	**fta** (J=7)	**fbp** (B=200)
RS	54±51	719±47	3.4±1.7	88±73	534±55	2.1±1.3
BO_M	**26**±20	**344**±201	**12.0**±2.8	**30**±21	**321**±113	**5.1**±1.2
BO_R	28±27	454±153	10.0±2.2	31±25	399±129	4.7±1.4
GB_M	59±75	469±198	7.8±4.3	61±63	447±99	2.9±1.4
GB_R	58±75	509±193	7.4±4.1	66±58	426±102	2.9±1.4

Table 4.2: Evaluation on NMT models trained with different language pairs for multi-objective (BLEU & decoding time) optimization on NMTLC. Fixed-target one (fto) and fixed-target all (fta) are measured by the number of model evaluations, and fixed-budget (fbp) is measured by the number of Pareto-optimal points. J is the size of the true Pareto set and B is the runtime budget, which is adjusted based on the size of the search space.

4.4 Analysis

Section 4.3 shows how to rigorously compare HPO methods based on various performance metrics. Here we illustrate examples of how to obtain deeper insights into HPO algorithm behavior using the table-lookup framework.

For single-objective optimization, we compare the best BLEU and mean squared error (MSE), which is the averaged squared difference between ground-truth BLEU and predictions, achieved by different HPO methods across time. We can see from Figure 4.3 (left) that BO and GB converge much faster than RS, and GB is superior over time. This could be partly explained by Figure 4.3 (right), GB can already fit the data well in the beginning, while BO starts from a much larger MSE and decreases gradually.

For multi-objective optimization, we show the evolution of Pareto-optimal fronts in Figure 4.4. There is a trend that Pareto fronts are moving towards the lower right corner at each iteration, verifying the effectiveness of our HPO methods.

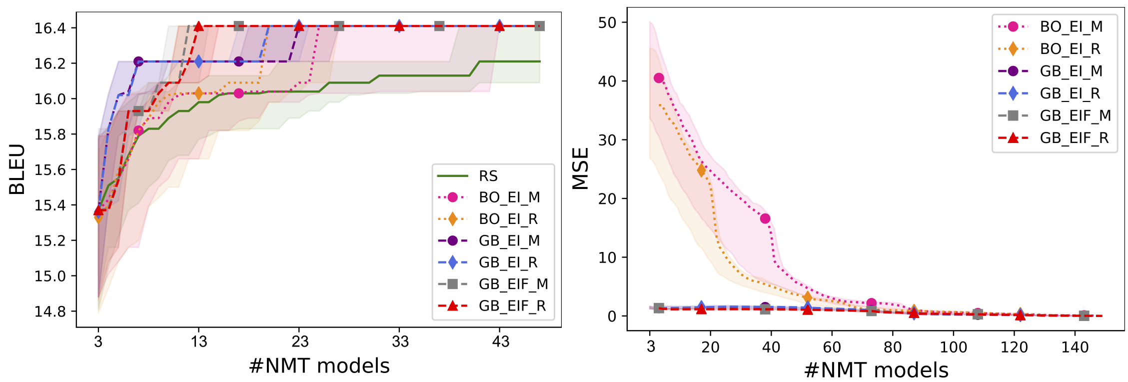


Figure 4.3: Left: Best BLEU found by different HPO methods over time on ja-en NMT models. Right: Mean squared error achieved by different HPO methods over time on ja-en NMT models. We plot the median and the 25th and 75th quantile across 100 independent runs.

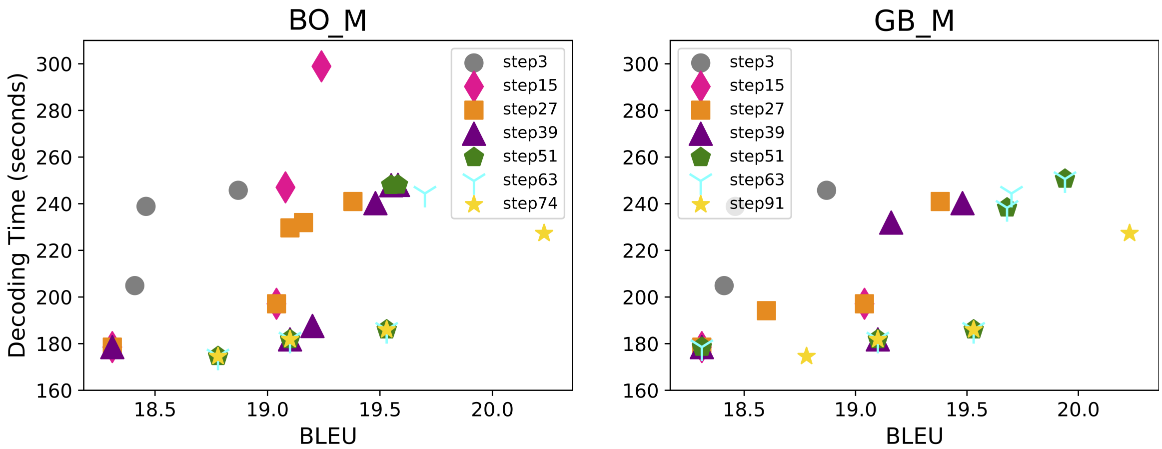


Figure 4.4: Pareto-front approximation during multi-objective optimization using BO_M and GB_M on ru-en. "Step" is the number of evaluated MT models. Gray circles form the Pareto set of initial seeds. In this example, all three initial seeds happen to be Pareto points. Gold stars are the Pareto solutions of the dataset. Lower-right corner is better.

4.5 Conclusions

In this chapter, we introduce a typical HPO framework, the sequential model-based optimization method, along with a well-known implementation, Bayesian optimization. We establish connections between graph-based semi-supervised learning and the HPO problem setup, proposing a novel HPO method that employs graph-based regression as the surrogate model and expected influence as the acquisition function.

We benchmark this new method against random search and Bayesian optimization using the NMTLC dataset, demonstrating that the graph-based method achieves superior performance across various evaluation protocols for both single and multi-objective optimization. This chapter also provides a concrete example of how to develop and evaluate a new HPO method using the NMTLC benchmark dataset.

Importantly, our graph-based HPO method is not limited to NMT models; it is applicable to any machine learning model. It is left to future work to evaluate its performance on other models.

Chapter 5

Post-Hoc Interpretation of Neural Machine Translation Hyperparameters with Explainable Boosting Machines

CHAPTER 5. POST-HOC INTERPRETATION OF HYPERPARAMETERS WITH EXPLAINABLE BOOSTING MACHINES

In previous chapters, we show that hyperparameter tuning is important for achieving high accuracy in deep learning models, yet little interpretability work has focused on hyperparameters. In this chapter, we propose to use the Explainable Boosting Machine (EBM), a glassbox method, as a post-hoc analysis tool for understanding how hyperparameters influence model accuracy. We present a case study on the benchmark dataset **NMTLC** to illustrate the kinds of insights that may be gleaned and perform extensive analysis to test the robustness of EBM under different data conditions.

This chapter is organized as follows: We motivate the adoption of a post-hoc interpretation framework in Section 5.1. In Section 5.2, we first briefly describe the EBM, which is the glassbox model used in our interpretability framework. Then we develop further the idea of post-hoc interpretation of hyperparameters and contrast it with other types of interpretability research. Section 5.3 presents a case study on **NMTLC**, to illustrate how our framework can be used to understand which hyperparameters are important, how their influence changes according to different hyperparameter values, and whether pairwise interactions are present. Section 5.4 analyzes the robustness of EBM: it helps characterize under what conditions are the interpretability results valid. Finally, we conclude this chapter in Section 5.6.

5.1 Introduction

Deep neural networks have revolutionized the field of AI, bringing about impressive improvements in accuracy at various tasks. There is now a growing interest in interpreting what the model is doing that leads to these high accuracies (Bastings et al., 2021). A better understanding is useful in many ways: it can provide researchers with a more in-depth view of the problem, assist developers in debugging the model, or give users a way to act on the model result.

Our goal in this chapter is to improve our understanding of neural network *hyperparameters*. While there are many research efforts on explaining a model's prediction or interpreting a model's parameters, there has been little work on hyperparameters. Hyperparameters like the number of layers and learning rate are important factors that impact model performance. In practice, many engineering hours are spent on tuning hyperparameters. We believe methods and tools for interpreting hyperparameters are needed to help practitioners tune more effectively; there are also applications in the growing field of AutoML (Hutter et al., 2019a), where our understanding of hyperparameters can help guide researchers in designing more effective search spaces.

In this chapter, we advocate a *post-hoc interpretation framework* for hyperparameters. This framework requires that a set of neural network models with different hyperparameters are trained and that their resulting accuracy metrics are recorded. Then, a glassbox model is fit on this data to reveal trends in hyperparameters.

We use Explainable Boosting Machines (EBM, Lou et al. (2013)) as the glassbox model; it is a Generalized Additive Model similar to Boosted Trees, except that its additive feature function is visualizable in 1-D or 2-D plots, making it well-suited for understanding hyperparameters.

The focus of this chapter is two-fold: First, we advocate a framework for understanding hyperparameters with EBMs and present a case study on machine translation Transformers to illustrate its usefulness. Second, we perform extensive experiments on EBMs to characterize the conditions where interpretability results are robust.

5.2 Methodology

5.2.1 Explainable Boosting Machine

In consistency with Chapter 4, we represent a hyperparameter configuration as $\boldsymbol{\lambda}$, which is the input feature vector to EBM. We denote the performance of the machine learning algorithm in interest as $f(\boldsymbol{\lambda})$. We use EBM as introduced in Lou et al. (2012) and Lou et al. (2013) and implemented in Nori et al. (2019). EBM is a generalized additive model with the form:

$$y = \beta_0 + \sum_j f_j(\boldsymbol{\lambda}_j) + \sum_{ij} f_{ij}(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j), \tag{5.1}$$

where f_j is a feature function for hyperparameter $\boldsymbol{\lambda}_j$, and y is the prediction of $f(\boldsymbol{\lambda})$. Since EBM is an additive model, by examining f_j, the contribution of a single hyperparameter to the final model performance can be easily interpreted. Additionally, EBM also includes pairwise terms f_{ij} to increase accuracy and enable analysis of pairwise interactions between hyperparameters. EBM is trained with bagging and gradient boosting, where the feature functions f_j are built as trees.

In our experiments, we focus on 6 Transformer hyperparameters (Table 3.1), so $\boldsymbol{\lambda}$ is a vector of dimension 6 and the EBM model is a sum of 6 single-hyperparameter functions f_j, up to $(6 \times 5)/2 = 15$ pairwise functions f_{ij}, and a bias term β_0.

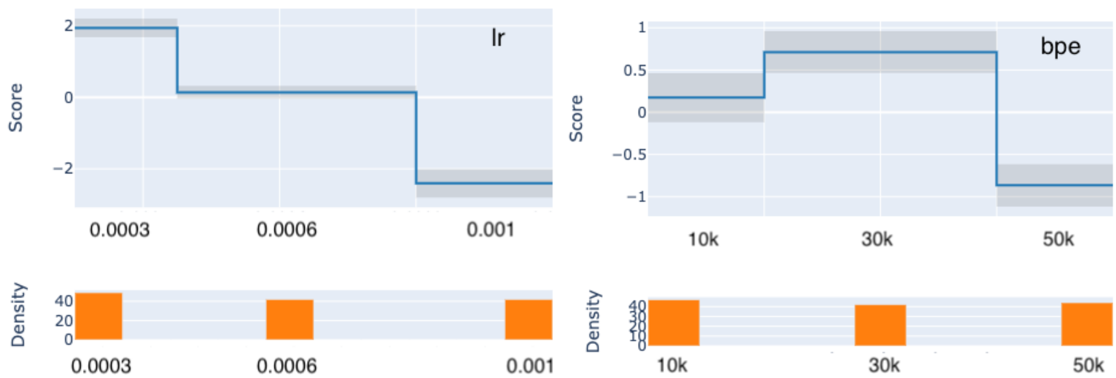


Figure 5.1: Single hyperparameter feature function on en-ja. Left: initial learning rate. Right: bpe symbols. Higher *score* indicates a higher chance of getting a high BLEU score. *Density* refers to the number of samples in the dataset.

An attractive aspect of EBM is that $f_j(\boldsymbol{\lambda}_j)$ is based on a single feature, and can be of arbitrary shape. See examples of f_j in Figure 5.1: on the left, we see that $f_{j=1}(\boldsymbol{\lambda}_1)$ decrease in score as the learning rate hyperparameter increases; on the right, we see a different $f_{j=2}(\boldsymbol{\lambda}_2)$ increase in score slightly as BPE hyperparameter from 10k to 30k, then drop sharply when BPE increases to 50k. Since the $f_j(\cdot)$ are summed *linearly* to predict $f(\boldsymbol{\lambda})$ (accuracy or BLEU score), we can obtain an intuitive understanding of how each hyperparameter impacts the final accuracy.

5.2.2 Interpreting Hyperparameters

Proposed framework: Our goal is to gain insights about hyperparameters for a class of deep neural networks. We require the existence of a set of models with different hyperparameter settings trained on the same dataset. For example, assume a set of Transformer (Vaswani et al., 2017) models $\{M_\lambda\}, \lambda \in \Lambda$ where Λ represents the hyperparameter space, M_λ represents a model with a specific hyperparameter setting (e.g. 6-layer encoder, 2-layer decoder, 8 heads, 256-word embedding size); each model has an accuracy metric $s(M_\lambda)$,[1] and a glassbox model is fit on pairs $P \triangleq \{(M_\lambda, s(M_\lambda))\}$. Assume there is a person building the models (model builder) and a person analyzing the models after the fact (model analyzer); they may or may not be the same person. Our framework consists of three steps:

1. On a dataset D, the model builder trains N models $\{M_\lambda\}$ and record their accuracy metric $s(M_\lambda)$. The metric can be any scalar in $\mathbb{R}$; for this chapter, we focus on machine translation and use the development set BLEU score.

2. The model analyzer fits an EBM on $P \triangleq \{(M_\lambda, s(M_\lambda))\}$. The EBM is a function $F(\cdot)$ that maps from hyperparameter space to BLEU score, $F : \Lambda \to \mathbb{R}$. In

[1] Same as $f(\boldsymbol{\lambda})$ in Section 5.2.1.

practice, a small subset of P is held out to measure EBM's generalization, and we would proceed only if we trust that the EBM has not over-fit or under-fit.

3. The model analyzer visualizes the internal features of EBM to glean insights about hyperparameters.

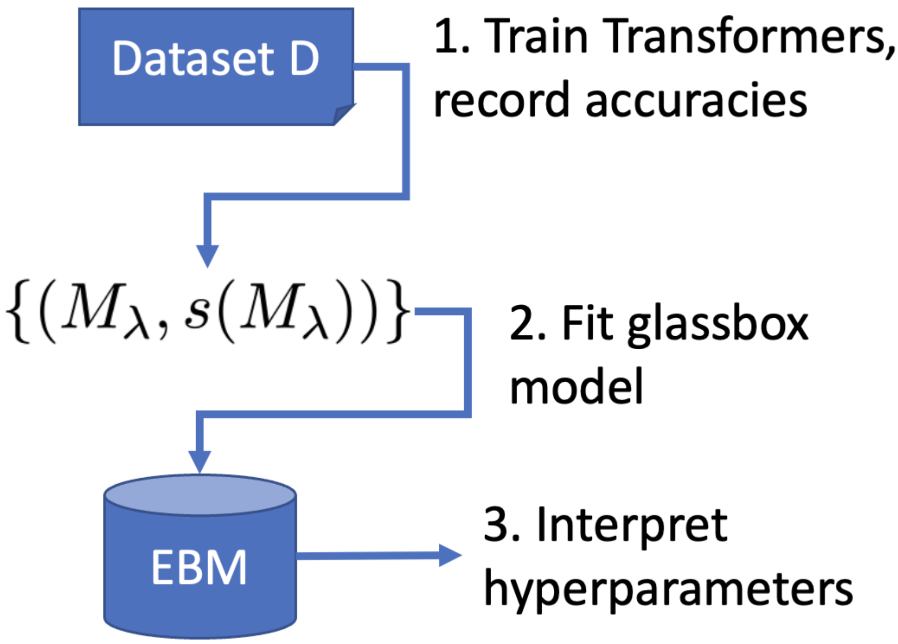


Figure 5.2: Proposed framework for the post-hoc interpretation of hyperparameters with Explainable Boosting Machine (EBM). 1. Transformers with different hyperparameter configurations are trained and their performance is recorded $\{(M_\lambda, s(M_\lambda))\}$. 2. EBM fits on those data points. 3. The internal features of EBM are visualized for the interpretation of hyperparameters.

The overall framework is shown in Figure 5.2. Step 1 is critical because it provides the data for EBM fitting. How large must N be, and are there requirements for the samples from Λ to be independent, identically distributed (i.i.d.)? Neural models can be expensive to train, so we assume that Step 1 is the result of whatever hyperparameter search was performed by the model builder. Thus, the model analyzer may not have full control over the models available for analysis. Section 5.4 characterizes under what conditions EBM is robust over different sizes and distributions of P.

Step 2 is the core component of our framework. Different glassbox regression models are possible, but we choose EBM due to its excellent visualization ability. Note that while there is a considerable amount of work on interpreting a Transformer' s parameters such as attention weights (Kobayashi et al., 2020; Abnar and Zuidema, 2020; Tay et al., 2021; Lim et al., 2018), these methods are not readily applicable due to the non-differentiability and heterogeneity of hyperparameters. Thus, an external model $F : \Lambda \rightarrow \mathbb{R}$ that treats hyperparameters as input features is more amenable. This external model is essentially finding hyperparameter "features" that are predictive of accuracy. As long as this model is glassbox in the sense that its internals are viewable, then we are able to interpret the results in Step 3.

Broader context: We would like to provide context on what our framework does and does not do. In the Explainable AI literature, one way to characterize explainability/interpretability research is to ask where the method sits on the local vs. global and self-explaining vs post-hoc continuum (Danilevsky et al., 2020). Local methods explain the model's behavior on a specific input, whereas global methods inspect the model generally. Our framework is global in the sense that it identifies hyperparameter trends based on the accuracy of a batch of inputs. Self-explaining methods generate explanations as part of the model's prediction process, whereas the post-hoc method builds an external model after the predictions have been made. Our framework sits squarely in the post-hoc camp because we work on top of trained Transformers, but it is interesting to note that the glassbox EBM employed can be called a self-explaining method.

In terms of research on hyperparameters, there is a branch of work (Bahar et al., 2017; Britz et al., 2017; Araabi and Monz, 2020) aiming at finding the optimal choices of hyperparameter values. In those works, hyperparameters are usually manually tuned based on experience, and massive experiments are conducted to gather results. Those works would make recommendations on which hyperparameter combinations to use in general. We call this approach *prescriptive*; they are useful to inform the building of specific models.

In contrast, our framework is *descriptive*: models have already been trained, and we are interested in understanding the relationship between hyperparameters and accuracy. In other words, rather than predicting whether to set the embedding size to 256 or 512, we are more interested in seeing how accuracy changes according to various embedding sizes and understanding whether other hyperparameters like the number of layers would interact. This is an example of post-hoc analysis, which is also used in medicine (TDI, 2022; Srinivas et al., 2015) – after the effectiveness of a new treatment is tested, post-hoc analysis on both the failed and successful trials are conducted. It is not the intent of the original study, but it is the support for further trials. The distinctions between the two kinds of interpretation work are summarized in Table 5.1. Post-hoc interpretation on hyperparameters is well suited to the

Type	**Goal**	**Example Result**
Prescriptive	Model building	Given past experience, we recommend setting the embedding size to 256 and attention head to 8 on dataset D.
Descriptive (this work)	Post-hoc understanding	Given N models that are trained on dataset D, we find that embedding size influences BLEU more than attention heads.

Table 5.1: Two kinds of goals for Interpretability Research on hyperparameters.

following two scenarios: (a) Suppose a practitioner has already performed extensive hyperparameter tuning and has deployed the best model. It would be a waste to throw away all the data pairs P. Running post-hoc interpretation allows us to extract more knowledge from the data. Knowledge about which hyperparameters are important, for example, may inform future hyperparameter tuning experiments; it may also assist AutoML researchers in designing more efficient search spaces for hyperparameter optimization and neural architecture search. (b) Suppose a researcher proposes a new neural network model. Providing a post-hoc analysis of hyperparameters is akin to showing feature ablation experiments. In sum, our work can be considered as an effort to unpack "blackbox" deep learning models at the level of hyperparameters.

5.3 Experiments

We now provide a case study on **NMTLC** to illustrate the kinds of insight we can learn from the proposed post-hoc interpretation framework. We show examples that through EBM, we can explore the importance (Section 5.3.2) and the correlation (Section 5.3.3) of hyperparameters to the performance of the NMT systems. Besides the effect of single hyperparameters, the pairwise interaction (Section 5.3.4) can also be detected by EBM.

5.3.1 Setup

We adopt the implementation[2] of EBM from Nori et al. (2019). To be specific, we train an EBM regressor on each of the language pairs (for trained-from-scratch models in **NMTLC**), which results in 6 models.

5.3.2 Hyperparamter Importance

EBM learns an importance score for each feature, which indicates how much the model performance would change with varying feature values. It is computed as the absolute expected value of f_j over the dataset. Figure 5.3 plots the hyperparameter importance ranking on six language pairs. As shown in the figure, hyperparameters are not equally important and there is a large discrepancy between features. On ru-en, *#embed* and *lr* are the most critical hyperparameters in determining Transformer's performance followed by *bpe*; while adjusting *#layers*, *attn* and *#hidden* (not shown in the figure) would only slightly affect the results. On zh-en, *lr* and *#embed* are also at the top of the listing, but the overall ranking is different from ru-en. Some

[2]https://github.com/interpretml/interpret

important hyperparameters for ru-en, e.g. *bpe*, rank low on zh-en. Some insignificant hyperparameters for ru-en, e.g. *#hidden*, are elevated to higher positions on zh-en.

In summary, there are only a limited number of critical hyperparameters for Transformers, and it would be more efficient to focus more on tuning them when developing a model. Across 6 language pairs, *#attn* is always ranked low and can be probably dropped from future hyperparameter searches.

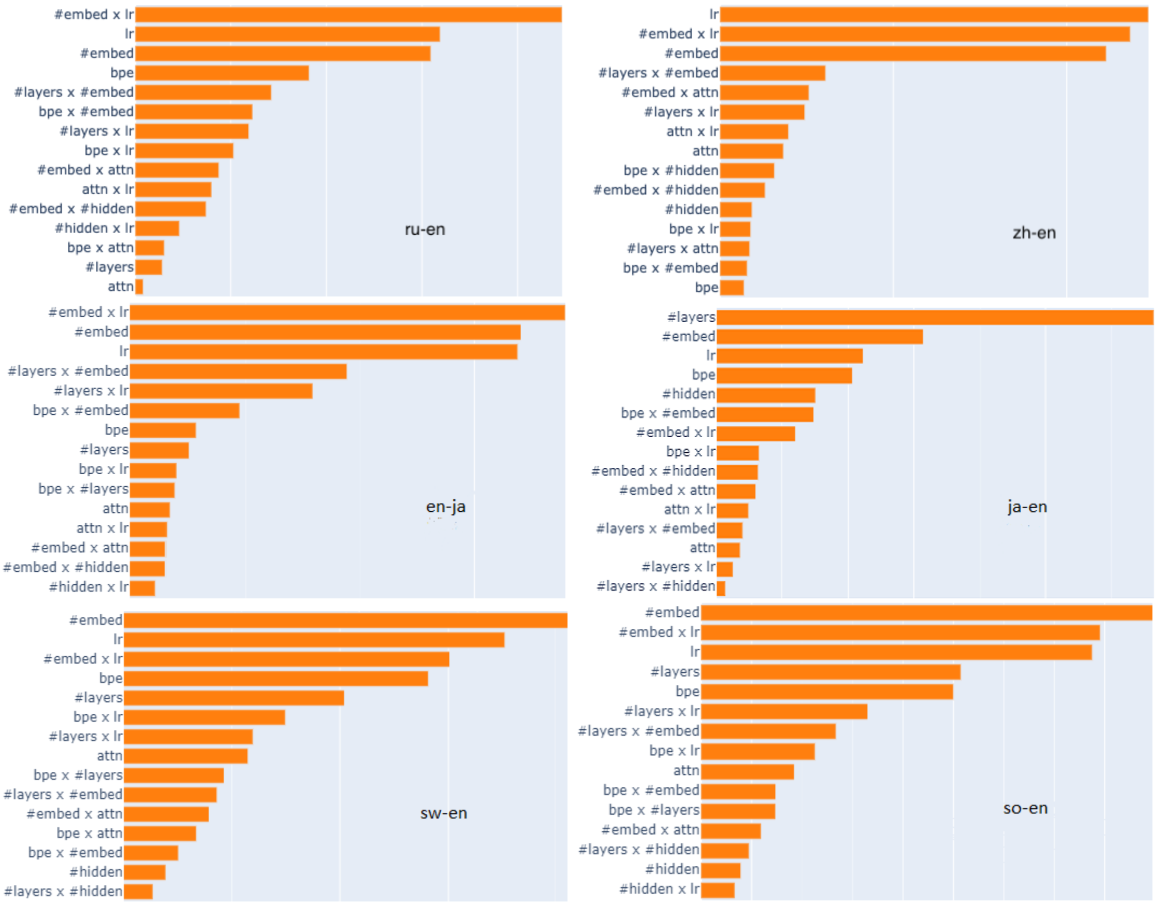


Figure 5.3: Hyperparameter contribution rank on trained-from-scratch models in **NMTLC**. Hyperparameters are ordered by importance score–for ru-en, $\#embed \times lr$ is the most important, while *attn* is the least important. Hyperparameters that are not included in the plots are in lower ranks than the shown ones.

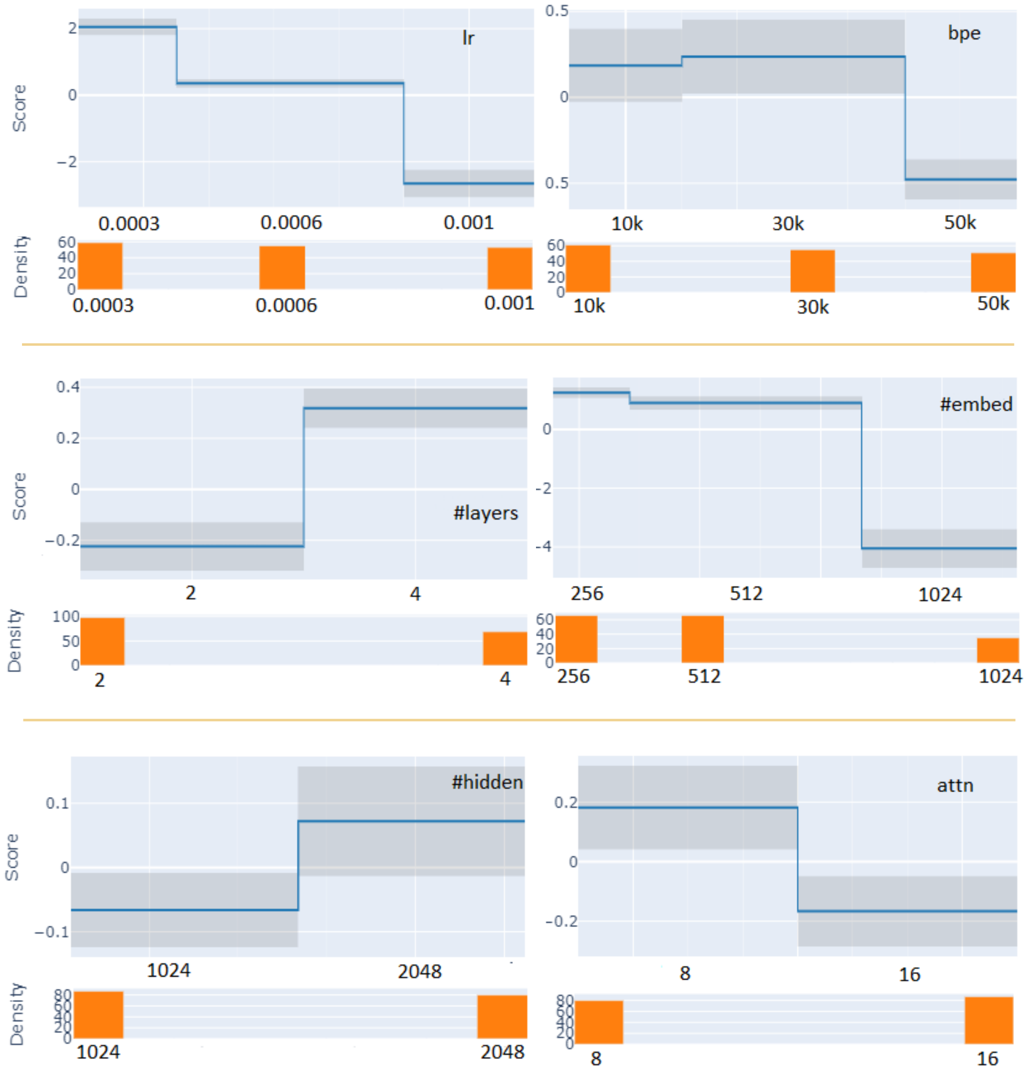


Figure 5.4: Single hyperparameter feature function on en-ja. Higher *score* indicates a higher chance of getting a high BLEU score. *Density* refers to the number of samples in the dataset. It shows that a slight change in a single hyperparameter can result in a big impact. Besides, BLEU scores and hyperparameter values are not always monotonically correlated, as seen in *bpe*.

5.3.3 Single Hyperparameter Analysis

Besides the macro view of the contributions of all the hyperparameters, EBM also provides a micro view of studying how the segments within each hyperparameter relate. Figure 5.4 depicts the single feature function extracted from the trained EBM model on en-ja. As *lr* increases from 0.0003 to 0.001, the BLEU score decreases significantly. While it is not the case for *bpe*, where the BLEU score does not change monotonically – it rises a little when *bpe* increases from 10k to 30k, then drops notably when *bpe* becomes 50k. This finding tells us both 10k and 30k are positively correlated with BLEU and the difference is not so distinct, but 50k is not desirable.

5.3.4 Pairwise Interactions

EBM can automatically detect and include pairwise interaction terms in its modeling. Figure 5.5 shows an example of how two hyperparameters interact to determine the NMT performance. On en-ja, *#embed* with the size of 1024 and *lr* with a size of 0.0003 produce the highest BLEU score among all the combinations. On the contrary, *#embed* 1024 and *lr* 0.001 output the worst Transformer. This is consistent with Figure 5.4 Left – larger *lr* worsens the performance.

However, this does not hold for *#embed* 256 and 512: given these values, there is not so strong of a (negative) correlation between *lr* and BLEU score. This seems to imply that while *lr* is sensitive for a large *#embed* 1024, it is less sensitive when

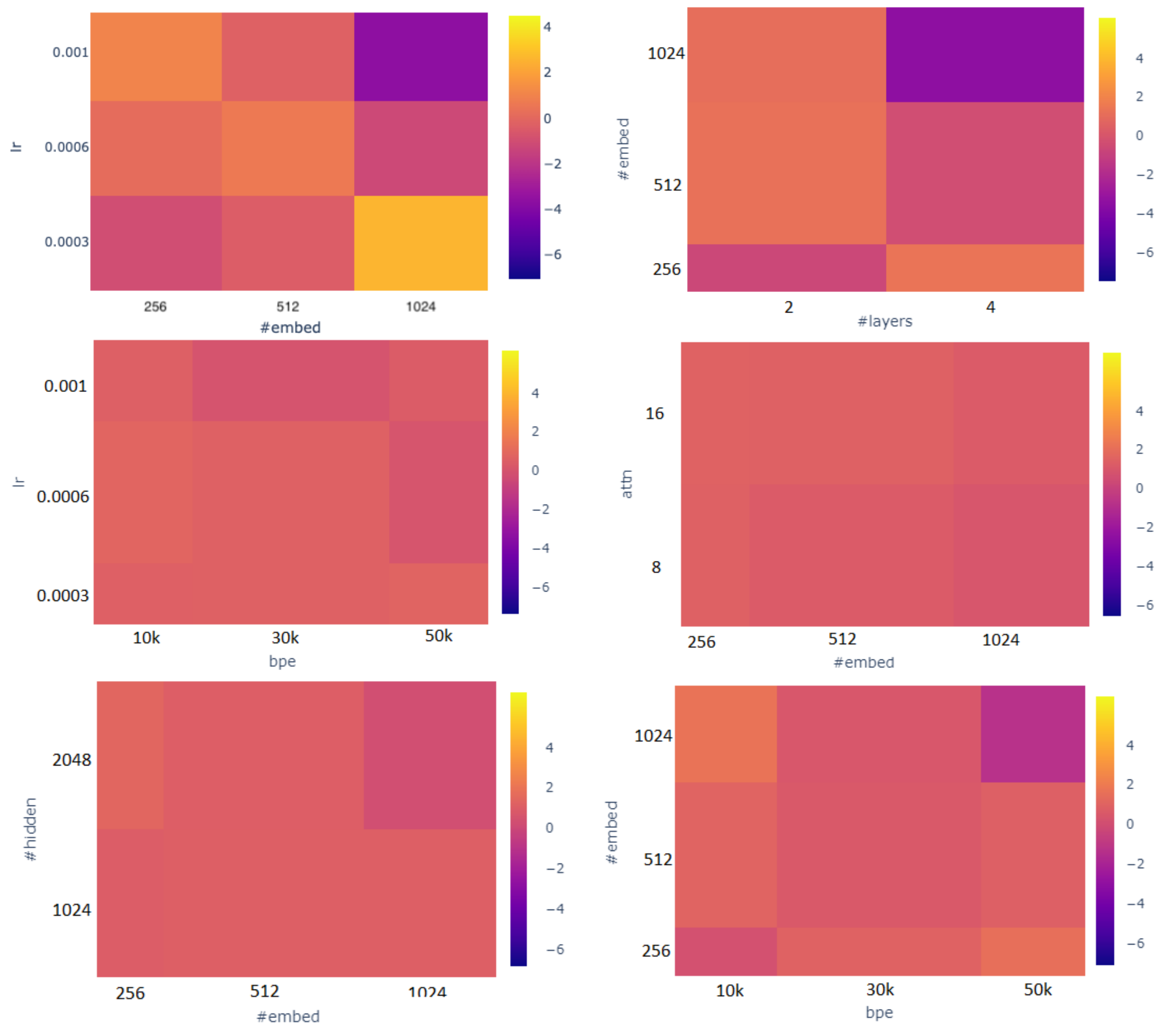


Figure 5.5: Pairwise interaction between two hyperparameters on en-ja. Higher *score* (yellow) indicates higher odds of getting higher BLEU scores. With different values of one hyperparameter, the sensitivity of BLEU scores to values of another hyperparameter varies. For *lr*-*#embed* (top left), when *#embed* is 256, the model performance does not vary much with different *lr* values, which is not the case when *#embed* is 1024.

#embed is small. We do have to interpret this result carefully because there may be confounding factors from the individual feature functions f_j that are added, but this is illustrative of the potential insights we may gain from this case study. Theoretically, the EBM formulation can allow for higher-order interactions (e.g. three-way). This may be a promising direction for future work.

5.4 Analysis

To ensure the validity of our post-hoc interpretation framework, we need to analyze the robustness of EBM to different kinds of data sizes and distributions. Specifically, one important requirement for our framework is the availability of $P \triangleq \{(M_\lambda, s(M_\lambda))\}$; one might not be able to fully control how this data is acquired. It may be a by-product of an extensive grid search, a manual and focused hyperparameter tuning guided by an engineer's intuition, or an AutoML experiment. This implies that hyperparameters may not be sampled uniformly from the space Λ, and the number of samples for EBM fitting may not be very large.

In order to gain a better understanding of EBM's robustness under different conditions, we conduct four experiments. We first study how EBM's fitting ability would be affected if the size or the distribution of training data[3] changes. We then make connections to HPO and examine EBM's performance on data generated by

[3]Data here refers to the (hyperparameter configuration, BLEU score) pairs, instead of the sentence pairs that are used to train an NMT model.

sampling from two different HPO methods. Finally, we investigate the generalization ability of EBM. To be more specific, we test whether an EBM model trained on one dataset can perform well on another dataset.

The experiments following are all conducted on sw-en except for the one in Section 5.4.4. We split the sw-en dataset, the largest dataset among the six trained-from-scratch tasks in **NMTLC** which contains 767 (configuration, BLEU score) pairs, into a train set with 614 samples and a test set with 153 samples. An EBM regressor is trained on a subset of the train set and its performance on the test set is reported. We repeat the process 5 times with different random seeds to generate 5 different train-test splits. Thus, the results reported below are all averaged over 5 runs.

5.4.1 Varying Data Sizes

In practice, it is often infeasible to get a tabular dataset as large as the one in Zhang and Duh (2020), where around 2,000 Transformers are trained. This raises the question of how EBM would perform with insufficient training data. In other words, it is in doubt if its interpretations on hyperparameters (e.g. observations shown in Section 5.3) are trustworthy when it is trained with less data.

In order to answer the questions above, we create datasets with different sizes by randomly sampling from the train set of sw-en. We experimented with subsets ranging from containing only 5% of the training samples, that is 31 samples, to the whole train set, i.e. 614 samples.

We use the following metrics to measure EBM's performance:

- **Mean Squared Error (MSE)** We calculate the average of the squared difference between the actual BLEU scores and EBM regressor's predictions given hyperparameter configurations. As a widely used measure of an estimator's quality, MSE is useful when compared between estimators. To be more specific, when there are multiple MSE scores, a lower one indicates a stronger estimator. When there is only a single MSE score, it is hard to judge whether it is low enough to testify to a good EBM model. Thus, we propose the following metrics as complements to MSE.

- **Spearman's Rank Correlation Coefficient (SRC)** We measure SRC between the ranking of real BLEU scores and EBM's predictions. For the purpose of interpreting hyperparameters, it is not necessary that EBM would predict the exact BLEU scores. Instead, it is more important that it recovers the ranking. For SRC, higher is better.

- **Mean Reciprocal Rank (MRR)** In some cases, for example, in hyperparameter search, one might be more interested in getting the best configuration and would be less concerned with the ranking of all the configurations. Reciprocal rank is defined as $\frac{1}{rank}$, where *rank* is the position of the best configuration predicted by EBM in the real ranking. MRR, in our case, is the average over 5 runs. It is better if MRR is closer to 1.

We plot EBM's performance with varying data sizes in Figure 5.6. It can be observed that although MSE rises drastically when the data size shrinks from 30% to 5%, it remains roughly at the same level when the size is larger than 30%. This means that a relatively accurate EBM model can be obtained with only 185 samples, and data sizes smaller than that would worsen the model significantly.

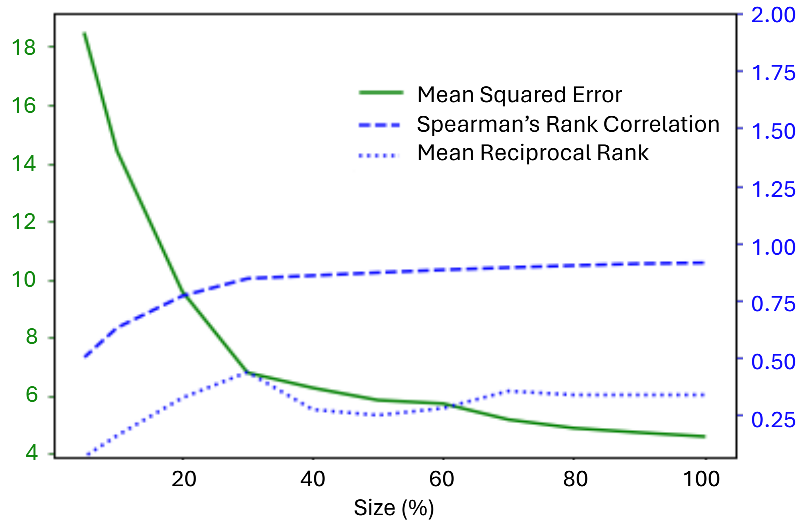


Figure 5.6: EBM's fitting ability with varying data sizes of sw-en. Subsets are generated by randomly sampling from the train set. Results are averaged over 5 runs with different random seeds.

The same trend is also shown in other metrics and 30% is the turning point for all the lines. SRC ends up getting close to 1 when the data size increases, suggesting EBM's great ability to recover the ranking. MRR stops at $\frac{1}{3}$, which means EBM mistakes the third best configuration as the best one. However, the difference between the BLEU score of the top three and the top one is small, which is only 0.41.

Size(%)	5	10	20	30	40	50	60	70	80	90	100[4]
Mean	18.43	14.40	9.58	6.79	6.26	5.84	5.72	5.17	4.88	4.73	4.59
Std	3.51	1.56	0.84	0.54	0.41	0.57	0.32	0.16	0.31	0.13	0.10

Table 5.2: The mean and standard deviation of MSE on sw-en test set. EBM is trained on subsets of train sets with various sizes and data compositions. Each subset is sampled 5 times with different random seeds. 100% refers to using all the samples in the train set, which takes up 80% of the original sw-en dataset. MSE here is not determined because we also randomly sampled the train set from the whole dataset multiple times.

5.4.2 Varying Data Distributions

Section 5.4.1 shows that a comparably good EBM model can be obtained by training on as few as 185 samples. Would this stay true if those 185 samples were replaced with other 185 samples? In other words, would EBM be robust to varying data distributions?

We evaluate EBM models trained with different data compositions and data sizes. Results are summarized in Table 5.2. As the amount of training data increases, the standard deviation of MSE decreases gradually, i.e. the EBM model becomes more robust. When given limited data, EBM is more prone to underfitting and generalize poorly to the test set. It can be inferred that hyperparameter interpretations produced by EBM models trained with more samples are more trustworthy and accurate than those trained with limited data.

5.4.3 Connections to HPO

In this section, we focus on investigating how EBM would fit the sampling by HPO methods, where sampling refers to the candidates proposed by acquisition function along one complete run of HPO – this is related to Section 5.4.2 since HPO sampling generates another unique data distribution for EBM to train on.

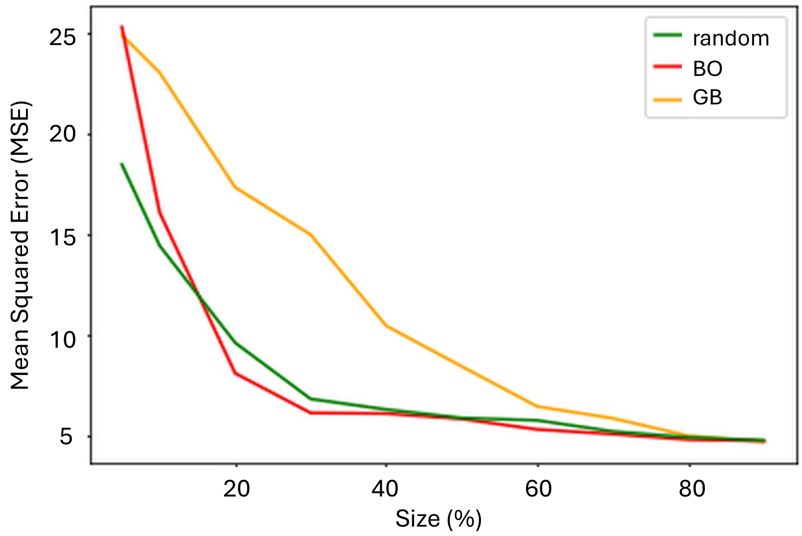


Figure 5.7: The performance of EBM trained on sw-en data sampled by Bayesian Optimization (BO), Graph-Based HPO method (GB), and random sampling (random). EBM is evaluated on a held-out test set, which takes up 20% of the sw-en data.

We experiment with two HPO methods, Bayesian Optimization (BO) and a Graph-Based HPO method (GB, Zhang and Duh (2020)). For BO, we use Gaussian Processes as a surrogate model and expected improvement as an acquisition function. For GB, we use the Matérn52 kernel and expected influence. We run BO and GB separately on sw-en and record the sampling order of hyperparameter configurations. We then compare the performance of EBM models trained on the first n% data points in the sampling with those trained on randomly sampled data. Results are plotted in Figure 5.7.

BO and *random* show similar trends with MSE falling sharply when the training data increases from 5% to 30%. While GB drops at a slower pace with MSE always staying the highest among the three. The discrepancy between the curves testifies to the discrepancy between the sampling of BO and GB. Compared to *random* and BO, the distribution of GB sampling is more skewed. At size 15%, BO surpasses *random* and maintains the lowest MSE till size 100%. This suggests that BO sampling makes EBM a better model than random sampling.

EBM can be used in combination with HPO in two ways: 1) During the run of HPO, EBM can be adopted as an analysis tool. By fitting the HPO sampling, it can provide insights on hyperparameter importance (Section 5.3.2) and make suggestions on hyperparameter values (Section 5.3.3). The HPO algorithm can then adjust its search space accordingly for later runs. But one should be cautious when the HPO algorithm in employment generates poor sampling distribution like GB does. 2) EBM can also be adopted as an alternative surrogate model considering its good fitting ability.

5.4.4 Transferability

So far, we have examined EBM's behaviors on specific language pairs. We have trained isolated EBM models on 6 MT tasks. Next, we explore whether EBM can leverage knowledge learned from one task and transfer it to another. Specifically, we evaluate each trained model on the test set of each of the language pairs. Figure 5.8 summarizes the results.

EBM faces difficulty on some of the transfers, for example, from sw-en (y-axis)

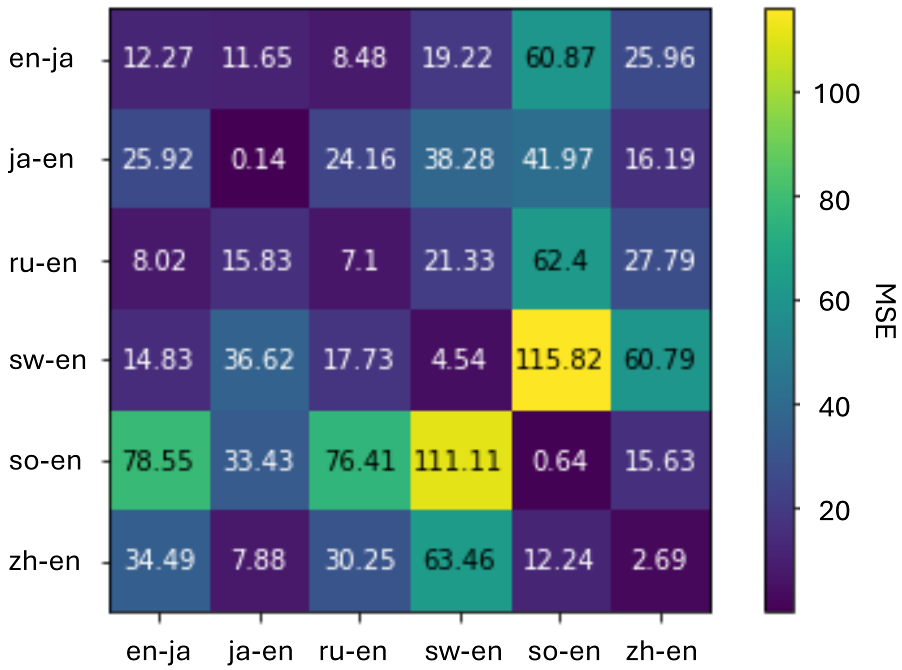


Figure 5.8: Mean Squared Error (MSE) of EBMs trained on one dataset (x-axis) and tested on another (y-axis). A smaller number suggests a better transferability between datasets. Some datasets show good transferability (e.g. ja-en to zh-en: 7.88, ru-en to en-ja: 8:48), while there are also pairs showing poor transferability as well (e.g. sw-en to so-en: 111.11, so-en to sw-en: 115.82).

to so-en (x-axis) and from so-en to sw-en. Meanwhile, there are also some successful transfers, for example, from en-ja to ru-en and from ru-en to en-ja. Surprisingly, EBM trained on en-ja generalizes so well on ja-en and ru-en that MSE obtained on those two test sets is even lower than that obtained on en-ja's test set. However, overall, there does not exist a single dataset that can produce a good EBM that can generalize well on all the other datasets. An interesting future work is to implement interpretability tools to analyze when transfer works and when it does not.

5.5 Related Work

Previous work that explores the effect of choices of hyperparameters can be mainly divided into two categories: the prescriptive approach aims to offer advice on the configurations by large-scale experimental runs and those developing tools to improve the understanding of the hyperparameters. Our work follows the descriptive approach, which seeks to interpret trends from a set of already-trained models. Related are some studies that measure hyperparameter importance: Hutter et al. (2014) and Sharma et al. (2019) applied a functional ANOVA framework to assess the importance, while Probst et al. (2019) adopted a variant term, hyperparameter tunability, conditioned on the difference on the performance of default and optimal settings of hyperparameters.

Exploration of the hyperparameter space also appears in research on HPO interpretability (Pfisterer et al., 2019; Freitas, 2019; Xanthopoulos et al., 2020). Moosbauer et al. (2021) attempted to interpret the HPO process with a variant of the partial dependence plot and showed what the surrogate model learned about the search space and how the final model is found.

5.6 Conclusions

In this chapter, we propose a framework for interpreting the hyperparameters of NMT models. Our framework work uses EBM as a post-hoc analysis tool, and we show that as a glassbox model, EBM is effective at interpreting hyperparameters.

While the computational needs of generating training data for EBM may seem large at first glance, we emphasize that we advocate for *post-hoc* analysis. In other words, the analysis is performed on the results of whatever hyperparameter search the model builder needs to perform to deploy a model.

Our MT case study demonstrates the kinds of insights one can glean regarding the relationship between hyperparameter configurations and Transformer performance; for example, we discover that not all hyperparameters are equally important, and some hyperparameters exhibit non-monotonic correlation with BLEU scores. Further, we conducted a series of analyses to test the robustness of EBM's fitting ability under varying data sizes and distributions. We show that EBM fits well under limited data, yet struggles with transfer across different MT datasets. It should also be noted that the conclusions drawn from MT tasks might not apply to other Transformer-based tasks.

Hyperparameter tuning is often viewed as a critical yet unintuitive part of the model-building process. We hope that our proposal provides a first step in unveiling the mysterious masks of hard-to-interpret hyperparameters in deep learning models.

Chapter 6

Best Practices of Successive Halving on Neural Machine Translation and Large Language Models

The HPO methods we introduced in previous chapters, e.g. random search, grid search, Bayesian optimization, and graph-based HPO, enhance NMT models but demand substantial computational resources. Successive halving, on the contrary, a multi-fidelity HPO method, mitigates this by early stopping unpromising models and allocating more resources to promising ones. This method is particularly relevant for NMT and large language models, which are computationally intensive. However, successive halving relies on a noisy estimation of model performance and assumes that early performance is highly correlated with final performance. In this chapter, we study the reliability of successive halving and propose best practices for its application in NMT and large language models.

We will begin the chapter by discussing the advantages and risks of successive halving in Section 6.1. In Section 6.2, we detail the implementation of successive halving. Next, we evaluate the performance of successive halving on the NMTLC dataset in Section 6.3. In Section 6.4, we explore learning curve extrapolation and its usability for successive halving, followed by a review of related work in Section 6.5. Finally, we conclude the chapter in Section 6.6.

6.1 Introduction

NMT models, whether trained from scratch or fine-tuned from LLMs, require extensive computational time, often taking days or weeks to converge. This makes hyperparameter searches over a reasonable space challenging. For instance, if an NMT model takes 2 GPU days to train, tuning 5 hyperparameters with 3 different values each would result in a total of $3^5 * 2 = 486$ GPU days! Practitioners with limited computational resources are thus often forced to resort to manual tuning or random search instead of more systematic methods like grid search or advanced HPO algorithms, increasing the risk of unfair comparisons between systems.

Successive halving (Karnin et al., 2013; Jamieson and Talwalkar, 2016) accelerates HPO by terminating unpromising models early in a set of models trained in parallel, saving more resources with more aggressive early stopping strategies. It has shown effectiveness in computer vision (Li et al., 2018) and NLP tasks (Dodge et al., 2020). However, its effectiveness for training NMT models or adapting LLMs for NMT tasks remains unclear.

The termination decision in successive halving is heuristic, based on the ranking of model performance up to the current timestamp. It assumes that early performance is highly correlated with late performance, which may not always be true. This raises the question: Does this assumption hold for NMT? If not, can we make it more reliable without relying solely on this assumption?

This chapter focuses on the effectiveness of successive halving for HPO in NMT models, whether trained from scratch or fine-tuned from an LLM. The main focuses are summarized as follows:

- **Evaluation:** We evaluate the effectiveness of successive halving for NMT HPO under different experimental setups.

- **Model:** We introduce a novel model for learning curve extrapolation, built upon the LCRankNet introduced in Wistuba and Pedapati (2020), and name it ***LCRankNet-v2***. We aim to determine whether "looking into the predicted future" enhances the reliability of successive halving compared to "looking back to the completed past."

Our findings indicate that the initial assumption of successive halving-that early performance predicts late performance—generally holds for NMT HPO with appropriate setups.

6.2 Successive Halving

The goal of successive halving (Karnin et al., 2013; Jamieson and Talwalkar, 2016) is to efficiently find the optimal hyperparameter configuration within a given search space. Suppose we have N configurations to explore. We begin by training all N models, and at every c checkpoints, we continue training only the top $\frac{1}{p}$ configurations based on their performance up to that point, discarding the rest. This process is repeated until only one configuration remains, which is then trained to convergence.

As shown in Figure 6.1, we start with $N = 10$ configurations and halve ($p = 2$) the number of configurations every $c = 5$ checkpoint. Each cut is based on the best performance of the configurations up to the current checkpoint. For example, at

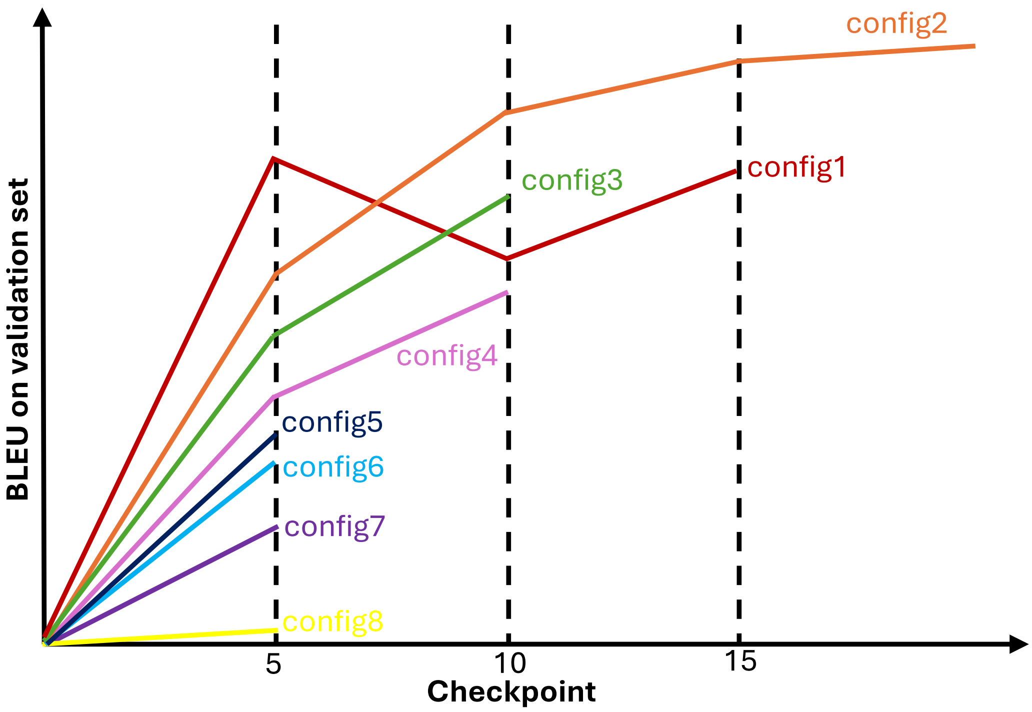


Figure 6.1: An example of successive halving, where $N = 10$, $c = 5$, $p = 2$.

checkpoint 10, when comparing config1 and config3, we compare config1's performance at checkpoint 5 with config3's performance at checkpoint 10.

In this example, assuming it takes one GPU day (20 checkpoints) for each model to converge, successive halving can reduce the total time for hyperparameter search from 10 days to 3.75 days. The aggressiveness of successive halving can be adjusted by changing the values of p and c. For instance, if $p = 3$ and $c = 2$, the total time could be further reduced to 1.3 days. However, a more aggressive strategy increases the risk of discarding good configurations too early. In the case of $p = 3$ and $c = 2$, config1 might be chosen over config2, even if config2 could have performed better in the long run.

6.3 Experiments

To evaluate the reliability of successive halving in NMT, we begin by identifying an appropriate evaluation metric (perplexity vs. BLEU) for termination decisions (Section 6.3.1). We then investigate whether halving consistently retains the best-performing half of configurations at different learning curve lengths (Section 6.3.2.1). Finally, we conduct extensive successive halving runs on random subsets of the configuration search space to assess its ability to consistently select the best configuration (Section 6.3.2.2).

6.3.1 BLEU vs. Perplexity

During training, models can be evaluated on the development set using either BLEU or perplexity. BLEU is more aligned with the ultimate goal of NMT, as BLEU scores are commonly reported for system comparison on development and test sets. However, perplexity is more closely aligned with the training objective and is significantly more efficient to compute. In our experiments, calculating perplexity is approximately 1000 times faster than BLEU on a single sentence, which means obtaining a BLEU score for an evaluation set can take hours. For HPO, we aim to select a configuration quickly while ensuring it achieves the best BLEU score. This raises the question: can we use perplexity instead of BLEU for selection and termination decisions in successive halving to accelerate HPO?

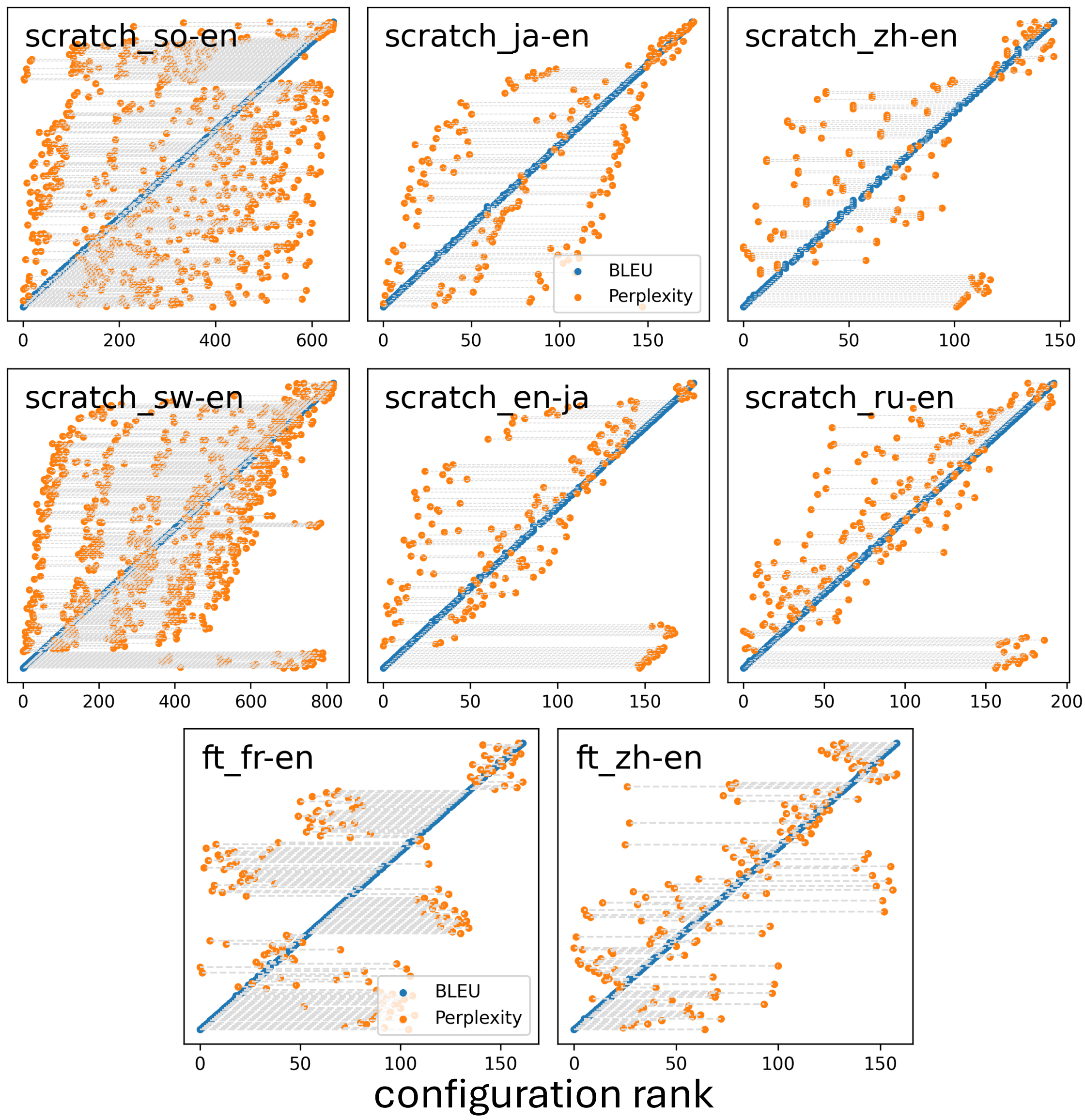


Figure 6.2: Configurations ranked by perplexity and BLEU. Configurations are ranked by their lowest perplexity on the development set and highest BLEU score, respectively. Perplexity does not correlate well with BLEU for all the datasets.

Figure 6.2 shows the ranking of configurations by their best BLEU and perplexity scores on the development set. The results indicate that perplexity does not consistently align with BLEU across all datasets. For example, in *scratch_sw-en*, *scratch_en-ja*, and *scratch_ja-en*, configurations with the best BLEU scores (lower left) often have the worst perplexity. This suggests that perplexity may not be a suitable alternative to BLEU for model selection and early stopping in HPO for NMT tasks.

Moreover, other machine translation metrics, such as METEOR, COMET, and others, might exhibit similar behavior to BLEU in their misalignment with perplexity. It may therefore be prudent to conduct successive halving using a metric that more closely aligns with the target evaluation metric.

6.3.2 Successive Halving on NMT

In this section, we evaluate the reliability of successive halving on NMT tasks.

6.3.2.1 Binary Rank

In successive halving, at each checkpoint, the bottom half of the configurations are discarded based on their performance up to that point. To understand how the ranking of partial learning curves correlates with the full curves, we calculate Spearman's rank correlation coefficient (ρ) on the binary ranks of configurations at each checkpoint (Figure 6.3). Generally, as the number of checkpoints increases, the correlation between the rankings of partial and full learning curves improves. This trend holds true for both perplexity and BLEU. Some datasets, such as *scratch_so-en*, *scratch_zh-en*, and *scratch_ru-en* for perplexity, and *ft_fr-en* for BLEU, achieve high

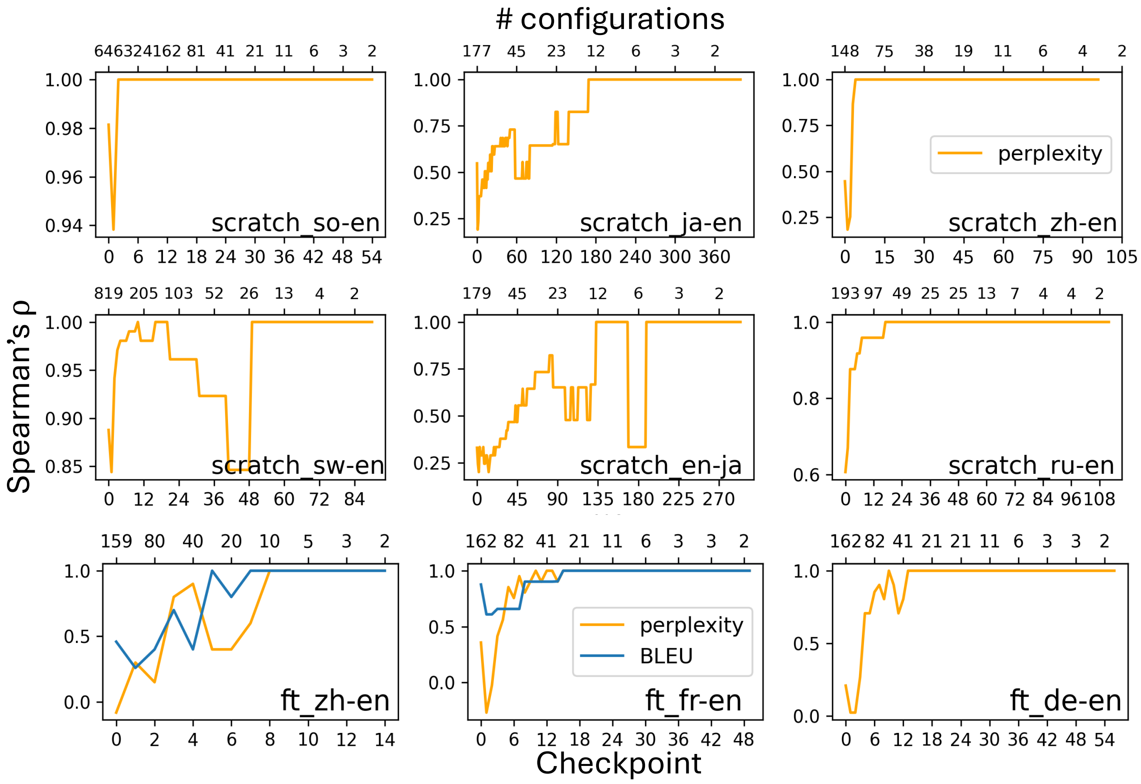


Figure 6.3: Spearman's rank correlation coefficient ρ on binary ranks of learning curves at each checkpoint. At each checkpoint, learning curves are ranked based on their best performance (perplexity or BLEU on the development set) up to that point. Curves are assigned a rank of 0 if they are in the top half and 1 if they are in the bottom half. There are fewer longer learning curves, as shown in the figure, as the checkpoint number increases, the number of models (upper x-axis) decreases.

correlation early in training.

6.3.2.2 Evaluation Results

We run successive halving 100 times on randomly sampled subsets of hyperparameter configurations, varying p and c as shown in Table 6.1. The reliability of successive halving is measured by whether the best configuration is selected at the end (**acc**) and when the best configuration is discarded (**dif**).

Most runs achieve either perfect **acc** or a **dif** of around 1, indicating that the best configuration is usually selected, and if not, it is discarded near the final stage. Table 6.2 further demonstrates that even when successive halving inadvertently eliminates the best-performing system, the performance of the second-best system remains nearly equivalent.

Increasing the discarding aggressiveness by increasing p and decreasing c reduces reliability (lower **acc** and higher **dif**) unevenly across datasets—*fr-en(ft)* is significantly affected, while *so-en(scratch)* and *zh-en(scratch)* remain stable.

		p=2, c=10		**p=2, c=5**		**p=4, c=10**	
		avg acc	**avg dif**	**avg acc**	**avg dif**	**avg acc**	**avg dif**
scratch	sw-en	99	0	97	0	95	0
	so-en	100	0	100	0	100	0
	zh-en	100	0	100	0	100	0
	ru-en	100	0	96	0	100	0
	ja-en	69	0.2	67	0.1	68	0.1
	en-ja	77	0.1	69	0.2	70	0.1
ft	fr-en	69	1.2	11	3.6	54	0.9
	zh-en	100	0	83	0.7	100	0
	de-en	100	0	61	1.6	57	0.8

Table 6.1: Successive halving evaluation results. Each dataset runs successive halving 100 times on randomly selected 40 configurations. The discarding ratio $\frac{p-1}{p}$ and frequency c checkpoints are varied. **Acc** indicates the percentage of runs where the best configuration is selected, and **dif** represents the average difference between total stages and the stage that discards the best configuration. A **dif** of 1 means the best configuration was discarded at the last stage.

	scratch						**ft**		
	sw-en	so-en	zh-en	ru-en	ja-en	en-ja	fr-en	zh-en	de-en
perplexity 1st	5.635	13.720	24.282	13.286	8.461	5.998	2.212	2.761	2.442
perplexity 2nd	5.694	13.756	24.282	13.355	8.585	6.141	2.216	2.762	2.443
BLEU 1st	26.09	11.23	14.66	20.23	16.41	20.74	31.36	10.84	N.A.
BLEU 2nd	25.91	11.09	14.66	20.08	16.21	20.21	31.26	10.82	N.A.

Table 6.2: Top two performing systems for each language pair. The difference in performance between these two systems is marginal, indicating that even if successive halving accidentally eliminates the top system, the performance of the second-best system remains nearly equivalent.

6.4 Learning Curve Extrapolation

Successive halving uses the best performance observed so far (*BSF*) to rank configurations at each checkpoint, assuming early performance correlates with final performance. However, as shown in Figure 6.3, this correlation can be low when learning curves are short. To improve on the heuristic BSF, we explore ”looking forward into the predicted future” by extrapolating the optimal performance of a configuration based on partial learning curves. This predicted optimal accuracy can then be used to rank configurations more effectively in successive halving.

6.4.1 LCRankNet-v2

Our learning curve extrapolation model, LCRankNet-v2, is a variation of LCRankNet (Wistuba and Pedapati, 2020). It takes three inputs: partial learning curves, hyperparameter configurations, and task meta-information (including dataset

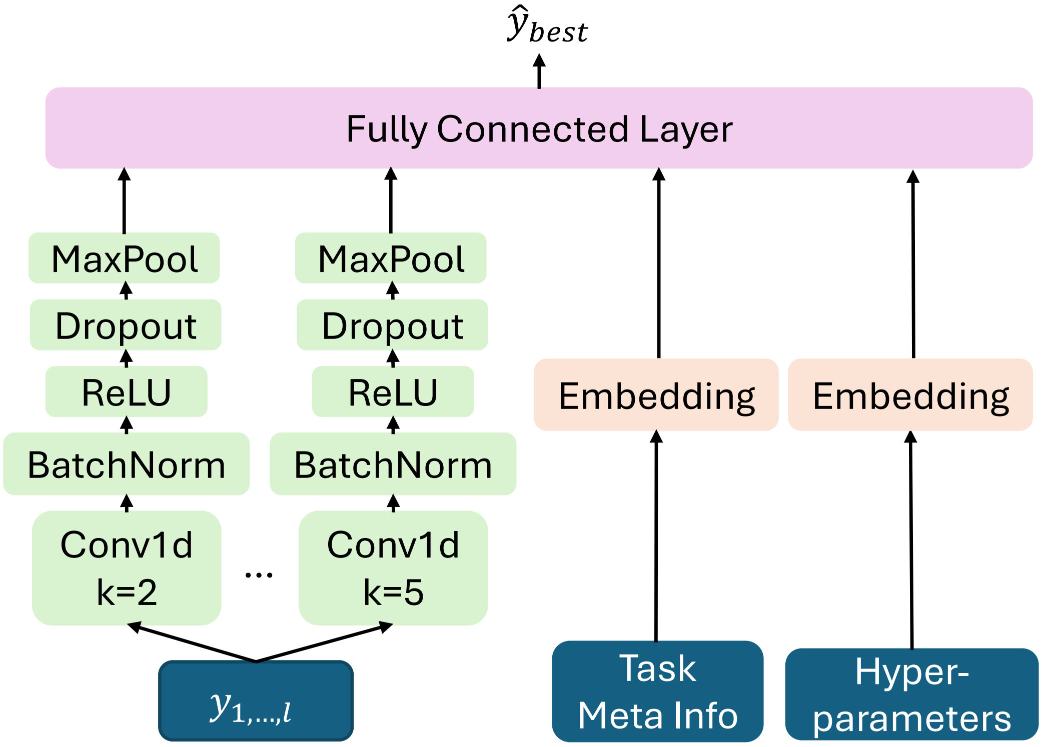


Figure 6.4: Architecture of LCRankNet-v2. Partial learning curves ($y_{1,\cdots,l}$) are processed through convolutional layers with kernel sizes ranging from 2 to 5. Task meta-information and hyperparameter configurations are embedded and then combined with the curve features. The concatenated features are fed into fully connected layers to predict the best performance of the configuration ($\hat{y}_{best}$).

ID, task type, source and target language, and base model). The architecture is shown in Figure 6.4. We removed the architecture embedding component from LCRankNet since it is defined in the hyperparameter configuration in our settings.

We pad partial learning curves to a length of 450. The convolutional layers have an output channel size of 128. Each hyperparameter and task meta-information is embedded with a size of 2. The feed-forward layer size is set to 128. For regularization, we use a dropout rate of 0.1 and a weight decay of 10^{-3}. The initial learning rate is set to 10^{-4}, with Adam as the optimizer and cosine annealing as the learning rate scheduler. The minimum learning rate (η_{min}) is set to 10^{-7}, and $T_{\max}$ is set to 10,000. Validation occurs every 1000 steps, and the batch size is 64. Training runs for 5 epochs.

6.4.2 Training Objectives

LCRankNet-v2 is trained using two loss functions: reconstruction loss $\mathcal{L}_{rec}$ and rank loss $\mathcal{L}_{rank}$. Given the true best performance y^i_{best} and the prediction $\hat{y}^i_{best}$ for learning curve i, the reconstruction loss is:

$$\mathcal{L}_{rec} = \sum_i (y^i_{best} - \hat{y}^i_{best})^2. \tag{6.1}$$

The probability that configuration i outperforms configuration j is defined as:

$$p_{i>j} = \begin{cases} 1 & \text{if } y^i_{best} > y^j_{best} \\ 0.5 & \text{if } y^i_{best} = y^j_{best} \\ 0 & \text{if } y^i_{best} < y^j_{best} \end{cases} \tag{6.2}$$

The corresponding prediction is:

$$\hat{p}_{i>j} = \frac{e^{\hat{y}^i_{best} - \hat{y}^j_{best}}}{1 + e^{\hat{y}^i_{best} - \hat{y}^j_{best}}}. \tag{6.3}$$

The rank loss is a binary cross-entropy loss:

$$\mathcal{L}_{rank} = \sum_{i,j} -p_{i>j} \log \hat{p}_{i>j} - (1 - p_{i>j}) \log(1 - \hat{p}_{i>j}) \tag{6.4}$$

To ensure fair comparisons, we always compare partial learning curves of the same length when computing $\mathcal{L}_{rank}$. To handle curves of different lengths, we include multiple truncated versions of each full learning curve in the training set. The total loss is:

$$\mathcal{L}_0 = w_{rec}\mathcal{L}_{rec} + w_{rank}\mathcal{L}_{rank}. \tag{6.5}$$

Additionally, we consider the *BSF* when ranking configurations. If the model predicts that performance will not improve beyond *BSF*, we set $\hat{y}_{best}$ to BSF. The probability of improvement p^{imp} over BSF is defined similarly to $pi > j$, and the improvement loss $\mathcal{L}_{imp}$ is:

$$\mathcal{L}_{imp} = \sum_{i} -p_i^{imp} \log \hat{p}_i^{imp} - (1 - p_i^{imp}) \log(1 - \hat{p}_i^{imp}). \tag{6.6}$$

The updated total loss is:

$$\mathcal{L}_1 = \mathcal{L}_0 + w_{imp}\mathcal{L}_{imp}. \tag{6.7}$$

During training, we set w_{rec} to 1, w_{rank} to 1000, and w_{imp} to 100. At inference, if $\hat{p}_i^{imp} > 0.5$, we set $\hat{y}_{best}$ to *BSF*.

6.4.3 Experiments Results

We conduct experiments to evaluate whether learning curve extrapolation improves the reliability of successive halving. Specifically, we compare the accuracy of ranking configurations using LCRankNet-v2's predictions versus the heuristic *BSF*. LCRankNet-v2 was trained using a leave-one-out strategy, excluding the target dataset from the training data and warming up the network with 20 examples from the target dataset, as suggested by Wistuba and Pedapati (2020).

In Table 6.3, we compare the performance of the heuristic *BSF* and LCRankNet-v2 trained to minimize $\mathcal{L}_0$ in predicting the rank between two configurations given partial learning curves, where we consider all the possible pairs with the same length. There are four cases: both methods rank correctly (**B**⭕**P**⭕), both methods rank incorrectly (**B**❌**P**❌), or one is correct and the other is incorrect (**B**❌**P**⭕ or **B**⭕**P**❌). On 7 out of 9 datasets, **B**❌**P**⭕ is less than **B**⭕**P**❌, indicating that while LCRankNet-v2 can sometimes correct *BSF*'s mistakes, overall, *BSF* performs better.

When trained to minimize $\mathcal{L}_1$, LCRankNet-v2 converges to *BSF* on all datasets, resulting in **B**❌**P**⭕ = **B**⭕**P**❌ = 0, and **B**⭕ = **P**⭕. Therefore, LCRankNet-v2 does not outperform the heuristic *BSF* in most of our settings.

	acc (B⭕)	**B⭕P⭕**	**B❌P⭕**	**B⭕P❌**	**B❌P❌**
sw-en	99.78%	95.30%	0.04%	4.48%	0.18%
so-en	99.76%	93.79%	0.19%	5.97%	0.05%
zh-en	75.73%	63.07%	17.61%	12.66%	6.63%
ru-en	99.63%	83.35%	0.12%	16.28%	0.24%
ja-en	95.86%	73.19%	2.35%	22.67%	2.08%
en-ja	94.73%	64.64%	4.73%	30.09%	0.56%
fr-en	84.43%	57.69%	6.94%	26.64%	8.73%
zh-en	75.73%	63.07%	17.61%	12.66%	6.63%
de-en	85.94%	35.20%	5.32%	50.74%	8.74%

Table 6.3: Performance of LCRankNet-v2 trained with $\mathcal{L}_0$. **Acc** (or **B⭕**) indicates the accuracy of ranking configuration pairs based on *BSF*. *B* represents ranking by *BSF* (vanilla successive halving), while *P* represents ranking by LCRankNet-v2's prediction. If **B❌P⭕** > **B⭕P❌**, successive halving is more reliable with LCRankNet-v2's prediction.

Is learning curve extrapolation necessary for successive halving on NMT? Not really. In Table 6.3, **P** generally underperforms compared to **B** in ranking configurations. This suggests that incorporating learning curve extrapolation is unlikely to significantly alter the results of successive halving.

6.5 Related Work

Learning curve extrapolation aims to predict model performance later in training based on early checkpoints. Kolachina et al. (2012) model learning curves for statistical machine translation systems by fitting them to various power-law family functions. Domhan et al. (2015) use a weighted combination of parametric model families to model learning curves. Klein et al. (2022) build a Bayesian neural network, while Chandrashekaran and Lane (2017) propose an ensemble method, and Baker et al. (2017) use frequentist regression models for learning curve extrapolation. Adriaensen et al. (2024) propose a Transformer pretrained on data generated from a prior, performing approximate Bayesian inference. Wistuba and Pedapati (2020) introduce LCRankNet, which encodes hyperparameters, dataset IDs, model architectures, and partial learning curves for performance prediction.

6.6 Conclusions

Successive halving is both efficient and effective for hyperparameter search in NMT tasks, significantly reducing computational resources and reliably selecting the best model with appropriate setups. However, its reliability depends on the target task and the choices of the cutting ratio (p) and cutting frequency (c). Based on the studies conducted in this chapter, we propose the following **best practices for successive halving in NMT and LLMs**:

1. Rank configurations at each checkpoint based on BLEU rather than perplexity.

2. Before running an extensive hyperparameter search with successive halving, train several configurations to convergence to estimate training time and learning curve trends, which helps in determining appropriate values for p and c.

3. Instead of keeping only one configuration at the end, increase the number of configurations that are trained to convergence (two might be sufficient, as our experiments suggest) to reduce the risk of discarding the best one at the last stage.

Successive halving can be integrated with other model-based HPO methods, such as Bayesian optimization and graph-based HPO. Although the evaluations and analyses in this chapter focus on NMT tasks, we believe that the best practice recommendations, particularly the second and third principles mentioned earlier, are also applicable to other machine learning tasks.

Chapter 7

A Hyperparameter Optimization Toolkit for Neural Machine Translation Research

In previous chapters, we demonstrate the benefits and necessity of using HPO methods over manual tuning for the development of NMT systems. There are two approaches to incorporating HPO into the development loop: (1) utilizing an existing HPO toolkit, which may require reimplementing the training pipeline; or (2) maintaining the existing training pipeline and implementing an HPO wrapper over it from scratch. In this chapter, we present a use case where the second approach is adopted. Specifically, we introduce an HPO toolkit for NMT designed to help researchers focus on creative tasks rather than mundane ones. This toolkit is implemented as a wrapper on top of the open-source Sockeye NMT software. Using the Asynchronous Successive Halving Algorithm (ASHA), we demonstrate that it is possible to discover near-optimal models within a computational budget with minimal effort.[1]

In the following, we first give an overview of the toolkit (Section 7.1 and Section 7.2) and hyperparameter optimization algorithm (Section 7.3). Then, the case study in Section 7.4 illustrates how the toolkit can help a researcher search over thousands of hyperparameter configurations with ease. Section 7.5 discusses our design choices, hopefully serving as a reference for those who want to implement similar toolkits for different NLP software. We discuss the related work, limitations, and ethical concerns in Section **??**, Section 7.6, and Section 7.6, respectively. Finally, we conclude this chapter in Section 7.8.

[1] `https://github.com/kevinduh/sockeye-recipes3` (code), `https://cs.jhu.edu/~kevinduh/j/demo.mp4` (video demo)

7.1 Introduction

The rapid development of new neural network architectures implies that the HPO process will only become more expensive. Currently, HPO tends to be performed manually by researchers in an ad hoc fashion, using scripts put together independently. The lack of open-source support tools means that the level of rigor in hyperparameter optimization may vary widely. This poses two risks:

1. **Insufficient exploration** of the hyperparameter space may lead to poor results, killing an otherwise promising research idea.

2. **Inequitable allocation** of compute resources for hyperparameter optimization of one model over another may lead to exaggerated results differences and misleading conclusions.

To support these efforts, we believe it will be beneficial to develop open-source tools to improve the HPO process itself.

This chapter presents a *hyperparameter optimization toolkit* for NMT research. It enables researchers to easily explore the hyperparameter space of various NMT models based on the PyTorch codebase of AWS Sockeye framework (Hieber et al., 2022). One simply specifies (1) the desired set of hyperparameter options to search, (2) the compute resource constraints, and (3) the training data paths, then the toolkit will plan and execute an automatic HPO and return the best model discovered. The toolkit implements the Asynchronous Successive Halving Algorithm (ASHA) (Li et al., 2020b), which is well-suited for commodity off-the-shelf distributed grids.

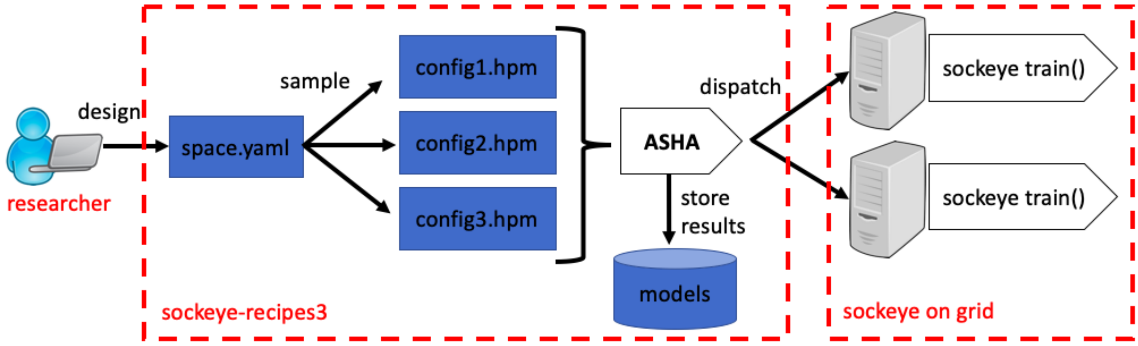


Figure 7.1: An overview of the `sockeye-recipes3` hyperparameter optimization toolkit. The researcher designs a hyperparameter search space (*space.yaml*), where some hyperparameter configurations are sampled randomly (*config.yaml*). The toolkit automatically calls the Asynchronous Successive Halving (ASHA) algorithm and dispatch training of NMT systems specified with different hyperparameter configurations on devices on the computational grid.

7.2 Usage Overview

Our HPO toolkit is named `sockeye-recipes3`, since it cooks up different models by training models with the AWS Sockeye NMT framework, version 3. An overview is shown in Figure 7.1. For concreteness, let us suppose the researcher in Figure 7.1 wants to run a rigorous HPO to obtain a strong Transformer baseline for a new dataset.

Step 1: The researcher designs a hyperparameter search space for his/her model. Table 7.1 shows some common hyperparameters for Transformers, but the toolkit is flexible to incorporate any user-defined hyperparameter. This hyperparameter space is expressed as a YAML file, e.g. `space.yaml`:

```
transformer_model_size: [256, 512, 1024]
```

```
transformer_attention_heads: 8
transformer_feed_forward_num_hidden: [1024, 2048]
...
```

The snippet above indicates that the researcher wishes to explore three choices for model size, one choice for attention head, and two choices for a feed-forward number of hidden units. The Cartesian product of all these choices forms the full hyperparameter space.

Step 2: `sockeye-recipes3` samples from the full hyperparameter space to generate a set of bash files called `hpm` files. Each `hpm` file represents a *specific* hyperparameter configuration and encapsulates all the information needed to train a model. This includes not only hyperparameter settings but also paths to training and validation data. For example, `config1.hpm` might train a model with:

```
transformer_model_size=256
transformer_attention_heads=8
transformer_feed_forward_num_hidden=1024
train_data=~/data/wmt.train.de-en.bitext
validation_data=~/data/wmt.dev.de-en.bitext
```

The set of `hpm` files represents all the hyperparameter configurations to be explored by the HPO algorithm. Rather than randomly sampling a subspace, one can also generate the full Cartesian product or manually edit some `hpm` files based on prior knowledge. Depending on the researcher's usage scenario, this set typically numbers from tens to thousands.

Step 3: Once the researcher is ready, he/she starts the ASHA program with resource specifications such as the number of concurrent GPUs to use and the number of checkpoints per training run. This Python code dispatches the training processes as standard Sockeye jobs to a distributed grid.[2] ASHA will attempt to efficiently train as many models as possible given the computational constraints. It is a bandit learning method that automatically learns when to stop a not-so-promising training run in order to allocate resources to other hyperparameter configurations. Details are in Section **??**.

Step 4: The results of all Sockeye training runs dispatched by ASHA are stored on disk. Each `hpm` file will have a corresponding subdirectory with the output log of a Sockeye training process. This makes it easy to replicate or continue any training runs in the future, with or without the `sockeye-recipes3` toolkit. Ultimately, the researcher can pick out the best model from the set for further experimentation.

Additional features:

(a) Metric: The toolkit's default is to find models with high BLEU on the validation set. This can be changed to any user-specified metric. Also, we have devised a multi-objective version of ASHA to enable joint optimization of accuracy and inference speed based on Pareto optimality (Marler and Arora, 2004).

(b) Analysis: After an ASHA run, one may wish to see if there are certain trends in hyperparameters, e.g. are some more important than others. This introspection can be helpful in understanding the model or designing future hyperparameter spaces. We have included a tool for posthoc analysis using Explainable Boosting Machines as introduced in Chapter 5.

[2]The dispatch in `sockeye-recipes3` is currently implemented for the Univa Grid Engine (UGE) but is easily extendable to other similar grid management software like SLURM.

Name & Description	Settings
Architecture Hyperparameters	
transformer_model_size - size of model/embeddings	{256, 512, 1024}
transformer_attention_heads - # of heads	8
transformer_feed_forward_num_hidden - # units in feedforward layer	{1024, 2048}
num_layers - for "encoder:decoder"	{6:6, 8:4, 4:4, 6:2}
Data Pre-processing Hyperparameters	
bpe_symbols_src - # of BPE symbols on source side	{5k, 10k, 30k}
bpe_symbols_trg - # of BPE symbols on target side	{5k, 10k, 30k}
Training Hyperparameters	
optimized_metric	perplexity
initial_learning_rate: initial rate for ADAM optimizer	{0.0002, 0.001, 0.002}
embed_dropout - dropout rate for source:target embeddings	.0:.0
label_smoothing	0.1
seed - random initialization seed	{1, 2}
Hardware-related Hyperparameters	
batch_size - # of words in batch	4096
checkpoint_interval - #batches before saving checkpoint to disk	4000

Table 7.1: Hyperparameter space used in the case study. The settings in red font are searched over, while others are held fixed. In total, we will explore $3 \times 2 \times 4 \times 3 \times 3 \times 3 \times 2 = 1296$ configurations.

7.3 HPO with ASHA

Problem: Suppose we have N hyperparameter configurations (`hpm` files) and a max compute budget of B, measured in terms of the total number of training checkpoints available. Let us select n configurations for actual training, where $n \leq N$. If each configuration is allocated the same budget, then each would be trained up to B/n checkpoints. When N is large, we have an untenable problem:

- If we choose n to be large (close to N), then B/n will be small, indicating that each configuration is only trained for a few checkpoints. Most models likely will

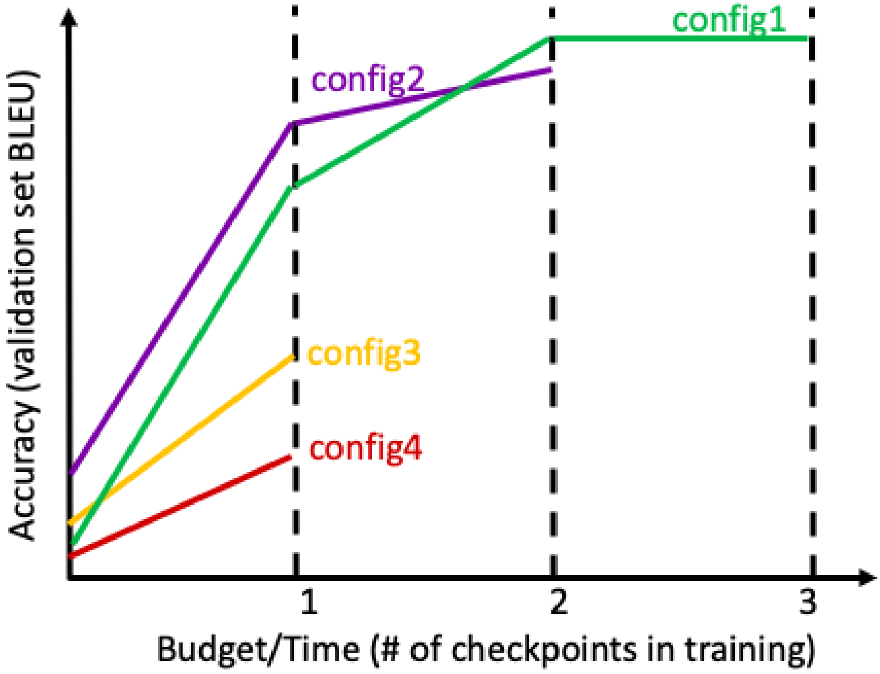


Figure 7.2: Illustration of Successive Halving. At each checkpoint, training of the bottom half of poor-performing systems is terminated.

not have converged.

- If we choose n to be small (despite N being large), then configurations that are chosen are trained well (large B/n) but the majority of configurations are not even trained at all.

The only solution is to allocate each configuration with a variable budget: i.e. train promising configurations for more checkpoints and terminate the not-so-promising ones prior to convergence. This is an intuitive idea that has probably been performed countless times by researchers by tracking learning curves in a manual fashion.

Successive Halving: The Successive Halving Algorithm (Jamieson and Talwalkar,

2016) introduced in Chapter 6, implements this intuition algorithmically, and is illustrated in Figure 7.2. Suppose we choose $n = 4$ hyperparameter configurations to explore and the total budget is $B = 7$ checkpoints. We begin by first training each configuration up to checkpoint 1 and measuring their validation accuracy. The configurations with lower accuracies at this point (config3, config4) are deemed not-so-promising and are terminated. The remaining half (config1, config2) are trained longer, and validation accuracy is measured again at checkpoint 2. Again, half of the configurations are terminated and the other half is "promoted" to be trained longer; this is done successively until the total budget is reached.

The main assumption of Successive Halving is that learning curves of different configurations are comparable and that the relative ranking of validation accuracy at intermediate checkpoints correlates to that at convergence. This is an assumption that cannot be proved but is likely reasonable in most cases with the proper setting of checkpoint intervals.

ASHA: In practice, the Successive Halving Algorithm (Chapter 6) has a bottleneck at each checkpoint: we need to wait for all configurations to return their validation score before deciding the best half to promote. The actual time that a configuration needs to reach a checkpoint depends on many factors such as GPU device type and model size. So we may end up waiting for the slowest training run, causing poor grid utilization.

To address this, an Asynchronous Successive Halving Algorithm (ASHA) is introduced (Li et al., 2020b). The idea is to promote a configuration as soon as it is guaranteed to be in the top half, without waiting for all configurations to return with their checkpoints' validation accuracy. For example in Figure 7.2, suppose three configurations (e.g. config2, config3, config4) have already returned an accuracy for checkpoint 1. We are then safe to promote the best one out of the group (config2) without waiting for config1 to return since config2 will be among the top half regardless of config1's accuracy.

Please refer to the original papers on ASHA, Successive Halving, and a variant called Hyperband (Li et al., 2016) for more detailed analyses. We focus on ASHA in `sockeye-recipes3`.

7.4 Case Study

Goal: To illustrate how `sockeye-recipes3` works in practice, we show a case study on building a strong Transformer baseline for a new Telugu-to-English dataset. Our initial training set consists of 900k lines of bitext obtained from public sources via the OPUS portal (Tiedemann, 2012). This is augmented with 7 million lines of back-translated data obtained by running a reverse system (English-to-Telugu NMT trained on 900k) on web-scraped news from the Leipzig corpus (Goldhahn et al., 2012). 3000 lines are held out from the initial training set to serve as the validation set.

Given this setup, our goal is to run HPO on a standard Transformer architecture to obtain the best possible model according to validation BLEU. This model can serve as a strong baseline for any future NMT experiment based on the same dataset. Since this is a low-resource language pair that is relatively unexplored in the research community, we opt to search a large hyperparameter space.

Hyperparameter space: Our `space.yaml` file is defined according to the options listed in Table 7.1. While any user-defined hyperparameter is possible, `sockeye-recipes3` exposes the most common options. We explore a total of 1296 configurations.

ASHA run: We run ASHA using the resource settings in Table 7.2. The reduction rate decides the fraction of configurations that are promoted each time: a factor p=2 reduction rate corresponds to "halving", but in practice, one can choose to be more or less aggressive. We also specify the number of GPUs that can be used concurrently by ASHA: here, it will dispatch jobs asynchronously up to that limit of G=40.

Finally, the settings for min, max, and per-rung checkpoints are NMT-specific modifications we found useful for ASHA. In Figure 7.2, halving is performed at each checkpoint, or at each "rung" in ASHA terminology. It is convenient to give NMT researchers the flexibility to choose the exact schedule: here, we decide that each configuration is trained for at least r=5 checkpoints (corresponding to 5×4000 batches due to the `checkpoint_interval` in Table 7.1) before we perform successive halving at the first rung. Thereafter, each configuration is trained for u=2 checkpoints before successive halving is performed. Finally, no configurations will be trained with more than R=25 checkpoints regardless of other ASHA settings; this small number of

Reduction rate. Top 1/p promoted	p=2
# of GPUs available	G=40
min checkpoints per model	r=5
#checkpoints per config per rung	u=2
max checkpoints per model	R=25

Table 7.2: ASHA settings for case study.

maximum checkpoints will probably not obtain state-of-the-art results but is suitable for the purpose of discovering several good configurations. The researcher may first inspect the ASHA results to identify several promising configurations, then manually train them for longer.[3]

Figure 7.3 samples a few learning curves (out of the 1296 configurations in total) to demonstrate how ASHA works in practice. The top figure is analogous to Successive Halving in Figure 7.2, while the bottom figure shows how the asynchronous dispatch occurs over time.

Comparison with grid search: To confirm whether ASHA finds good models, we also run a grid search on the same 1296 configurations, training each with up to 25 checkpoints. This corresponds to a total cost of $25 \times 1296 = 32,400$. In comparison, the ASHA run in our case study costs 60% less at 9066 checkpoints in total.

Table 7.3 confirms that ASHA can find good models that are found by an exhaustive grid search. For example, the maximum BLEU score by grid search is 20.3, and while this model is terminated at rung 4, the final model discovered by ASHA has a

[3]The best model discovered has 8 encoder layers, 4 decoder layers, 1024 model size, 2048 feedforward size, 10k source subwords, 30k target subwords, and achieves 35.6 spBLEU on the FLORES101 devtest (Goyal et al., 2022a).

Figure 7.3: Learning curves for a random sample of configurations in ASHA. The y-axis is the validation BLEU score. The top figure, where the x-axis represents # of checkpoints, is analogous to Figure 7.2 and shows which configurations are promoted. The bottom figure represents the same configurations plotted against wallclock time on the x-axis; this illustrates the asynchronous nature of ASHA. Observe that configurations are not started in sync, and long plateaus indicate when ASHA decided to pause the configuration at a checkpoint to allocate GPUs for other ones.

competitive BLEU score of 20.1. In our experience, ASHA is effective at finding a set of reasonable models at a fraction of the computational cost; if we desire the best possible model, nothing can replace the manual effort of an experienced researcher.

7.5 Design

`sockeye-recipes3` is designed with two principles: (1) All NMT codes, such as a researcher's proposed extension of the Sockeye framework, are encapsulated in

rung	ckpt	config	budget	med	max
0	5	1296	6480	0.3	20.3
1	7	648	7776	17.2	20.3
2	9	324	8424	18.9	20.3
3	11	162	8748	19.4	20.3
4	13	81	8910	19.7	20.3
5	15	40	8990	19.7	20.1
6	17	20	9030	19.7	20.1
7	19	10	9050	19.8	20.1
8	21	5	9060	19.8	20.1
9	23	2	9064	20.0	20.1
10	25	1	9066	20.1	20.1

Table 7.3: ASHA vs. Grid search: Each row lists the # of configurations explored in each rung, # of checkpoints (ckpt) trained so far per configuration, and accumulated budget (total checkpoints). The med/max columns are median/max BLEU scores among the configurations explored if they were trained to completion in a grid search. For example, in rung 2, 324 configurations were explored by ASHA and trained up to 9 checkpoints. If they were trained up to the full 25 checkpoints and their BLEU scores were collected, the median would be 18.9 and the max would be 20.3. ASHA preserves many of the top configurations that would be found by grid search.

separate conda environments. (2) All hyperparameters and data paths (for baseline and proposed methods) are explicitly specified in `hpm` files, and stored together with each sockeye training run. This means that it is easy to replicate or continue any training run by referring to (1) and (2). ASHA dispatches will run Sockeye training for u checkpoints at a time, so a job will automatically return the GPU resource at the end of each rung.

The ASHA implementation is a Python script that sits on a single server and regularly checks the status of Sockeye training runs on the distributed grid setup. The pseudocode is shown in Algorithm 3. The script keeps track of configurations that are training or paused at a checkpoint. When there is an idle GPU, it will decide whether to explore a new `hpm` or promote an existing one. The dispatch is a job submission command that starts a Sockeye train process on a GPU node. It depends only on the conda-environment provided, so it is easy to optimize different NMT implementations by exchanging the environment while keeping similar `space.yaml`, leading to equitable tuning.

Algorithm 3 ASHA pseudocode

```
while budget remains do
    for all c ∈ configs do
        s = check_state(c)                    ▷ Still training or at checkpoint?
    end for
    for all g ∈ idle GPU do
        h = get_hpm(configs)                  ▷ Explore new or promote?
        dispatch(h, g, conda-env)             ▷ Sockeye train()
    end for
    pause for m minutes
end while
```

7.6 Limitations

Scope of support: The `sockeye-recipes3` toolkit only supports the AWS Sockeye NMT framework. It is suitable for researchers who plan to implement and test out different NMT models in PyTorch using Sockeye's codebase. It is not meant to be extensible to HPO methods for other frameworks in NLP. The reason is that each toolkit has its own nuanced error messages and hyperparameter definitions, so it is easier to do design a focused toolkit.

No guarantees: In general, HPO methods give no theoretical guarantees; there is always an aspect of uncertainty. For example, there is no guarantee that ASHA will keep the top configurations if the learning curves do not follow our assumptions. One may be more conservative by setting more checkpoints per rung in ASHA, but this decreases the potential for efficiency.

Manual design: `sockeye-recipes3` does not fully automate the entire model-building process. The researcher still needs to design the hyperparameter space for each task. This search space is critical for the success of ASHA that follows. One may imagine a transfer learning (or meta-learning) approach where hyperparameter spaces from similar tasks are borrowed, but this is currently an open problem.

7.7 Environmental Impact

Automated HPO can lead to efficiencies in model building, but we need to be cognizant that there is also a risk of excessive optimization. The user needs to design what is a reasonable search space: for example, would it be worthwhile to optimize over many different random initialization seeds or over small differences between model sizes?

Excessive optimization poses three risks: First, one may select models that "overfit", though this can be ameliorated by proper choices of validation sets. Second, HPO gives an advantage to research teams with large compute resources; ASHA and similar methods are not useful on grids with less than e.g. 10 GPUs.

Third and perhaps more important, the computation may be wasteful. “Green AI” is an important call-to-arms for the research community: HPO is a double-edged sword in that proper usage leads to efficiency while excessive usage leads to wastefulness.

For example, to quantify the CO_2e emissions in our case study, we estimate that ASHA and grid search spent a total of 3050 hours on the GPU compute node. Our grid contains a mix of NVIDIA TITAN RTX, GeForce RTX 2080 Ti, and Tesla V100. In future versions of `sockeye-recipes3`, we plan to track power use individually for all jobs but let us assume an average power consumption of 250 watts, for a total of 0.762MWh. If we assume carbon efficiency[4] is at 432 kg CO_2e per MWh, data center power usage effectiveness (PUE) is 1.5, and there are no additional offsets for renewable energy, we end up with:

$$\frac{0.762 \text{ MWh}}{1} \times \frac{432 \text{ kg}}{\text{MWh}} \times \frac{1.5}{1} = 494 \text{ kg CO}_2\text{e}$$

This corresponds to the CO_2e of driving a car for 2000km or burning 247kg of coal. Ideally, we will eventually reach an understanding as a community of what amount of use is appropriate or excessive.

[4] https://mlco2.github.io/impact/

7.8 Conclusions

There is a progression of toolkit development that enables researchers to do better work. Deep learning toolkits like PyTorch and Tensorflow made it easy to exploit GPU hardware. Application-specific toolkits like Sockeye and Fairseq were built on top of that and enabled researchers to quickly prototype new ideas. Furthermore, we believe that HPO toolkits and experiment management toolkits in general will further help advance the speed and rigor of research.

We presented `sockeye-recipes3`, an open-source HPO toolkit for NMT research. Our hope is this will relieve some of the mundane aspects of manual hyperparameter tuning so that researchers can focus on more creative activities. A rigorous and automated HPO process will also lead to more trustworthy experiment results.

Chapter 8

Conclusions and Future Work

8.1 Conclusion

This dissertation addresses the significant research gap in HPO for NMT systems. Through an examination of the challenges and requirements in this field, the following contributions have been made:

- In Chapter 3, we address the notable absence of comprehensive studies on HPO for NMT models by constructing a benchmark dataset specifically tailored for evaluating and developing HPO algorithms for NMT systems. This dataset employs a table-lookup approach, featuring a large collection of pre-trained NMT models covering a predefined hyperparameter search space. These NMT models are trained towards various task settings, varying language pairs, training set sizes, and domains. We also include models either trained from scratch or adapted from LLMs.

  We introduced an evaluation framework that assesses the robustness of HPO methods across multiple tasks. Furthermore, recognizing the dependency of HPO performance on initialization and its variability across runs, we propose statistical-based evaluation metrics derived from extensive repeated trials, which are made feasible by this benchmark dataset.

- In Chapter 4, we propose a novel HPO method, which is based on graph-based semi-supervised learning. In this method, the prior knowledge of the hyperparameter search space can be encoded through the graph construction, by setting up parametric edge weights. Following the SMBO framework, this method performs comparably to Bayesian optimization and other existing HPO methods for both single-objective and multi-objective optimization. It is applicable not only to NMT but across all machine learning models.

- Chapter 5 delves into the analysis of hyperparameters in NMT systems, employing the EBM algorithm for post-hoc interpretation. Our findings reveal that not all hyperparameters exert equal influence and that interactions between them can be observed. We hope this study enhances the understanding of hyperparameter dynamics and aids in the design of hyperparameter search spaces for future HPO efforts.
- In Chapter 6, we explore the application of a multi-fidelity HPO method, successive halving, for NMT tasks. This method is particularly useful in the context of NMT due to the substantial computational demands typical of NMT training. Our studies conclude the general reliability of successive halving in preserving the optimal configuration in the end with proper setups.
- Chapter 7 presents an open-source HPO toolkit for NMT research. It is developed aiming at reducing the labor-intensive aspects of manual hyperparameter tuning. This toolkit facilitates a more focused and creative research environment and enhances the reliability and trustworthiness of experimental outcomes in NMT studies.

8.2 Future Work

This dissertation presents an extensive study, yet it is not exhaustive. Many avenues remain open for future research. In this section, we outline three possible directions for forthcoming studies: addressing the issue of overfitting and enhancing the generalization capabilities of HPO algorithms (Section 8.2.1); expanding the scale of HPO applications (Section 8.2.2); and exploring the potential of LLMs on HPO (Section 8.2.3).

8.2.1 Overfitting and Generalization

In the context of this dissertation, there are two potential types of overfitting: overfitting a task, which is a consideration when selecting an HPO method robust across various tasks; and overfitting a validation set, which pertains to choosing the optimal hyperparameter configuration for a single task using an HPO method.

Overfitting a task: When the task is to choose one HPO method across multiple alternatives, it is important for the evaluation of each HPO method to be conducted across a variety of tasks. An HPO method that excels in one task but performs poorly in others may be overfitting that particular task and thus lacks generalizability. In Chapter 3, we introduce an evaluation benchmark and framework that necessitate the assessment of HPO methods across different tasks, thereby addressing the issue of task-specific overfitting.

Overfitting a validation set: This form of overfitting concerns the application of an HPO method to select the best hyperparameters for a particular task. It is different from the overfitting in NMT training. For NMT systems, overfitting can often be identified and mitigated by evaluating model performance on a separate validation set. However, the HPO algorithm itself typically trains on data that pairs hyperparameter configurations with their corresponding performance metrics on an MT validation set, often without an additional set to validate the HPO algorithm's generalization to new data. This limitation can make the HPO process particularly susceptible to overfitting, especially given the computational costs that restrict exploration of the hyperparameter space. As noted by Feurer and Hutter (2019b) and Yang and Shami (2020), this form of overfitting remains an open problem in HPO and has not been specifically addressed in this dissertation.

A straightforward approach to reduce this type of overfitting involves the use of multiple HPO-validation sets. During the HPO process, the performance of each sampled hyperparameter configuration should be evaluated across various HPO-validation sets to determine the optimal configuration, rather than relying solely on the MT-validation set. Additionally, varying the training-validation split for each function evaluation can help prevent overfitting, as well.

Further analysis is required to assess the extent of overfitting within HPO processes. It is also interesting to investigate whether certain HPO methods are more prone to overfitting than others and whether HPO methods are more likely to overfit in NMT tasks compared to other machine learning tasks.

8.2.2 Scalability

HPO faces significant challenges when applied to large-scale machine learning models, such as Large Language Models (LLMs) with trillions of parameters and extensive multilingual machine translation systems like NLLB-200 (Costa-jussà et al., 2022), which features around 250 million parameters. The extreme costs associated with function evaluations in these contexts necessitate innovative approaches. Multi-fidelity methods, detailed in Section 2.2.2.5, offer one such strategy by allowing estimations of final model performance through partial training, as explored in Chapter 6. Another method involves conducting hyperparameter searches on a reduced subset of the training data (Visalpara et al., 2021), raising questions about how to effectively select these subsets, the transferability of results from smaller to full datasets, and task-specific variability in these strategies. To address these issues, establishing a benchmark that utilizes subsets of the same dataset—rather than different ones—allows for the exploration of how hyperparameter optimization varies with dataset size and whether optimal settings on smaller datasets can be effectively transferred to larger ones.

An additional direction for future research is to enhance efficiency by continuing training from a previous checkpoint rather than starting anew with each hyperparameter configuration. This approach could leverage prior training, utilizing it as a form of advanced initialization that potentially leads to faster convergence and conserves computational resources. PBT (Jaderberg et al., 2017), as described in Section 2.2.2.4, adjusts hyperparameters during ongoing training based on heuristic rules, such as optimization duration or performance benchmarks. However, systematic methods for deciding the optimal moment to switch hyperparameters remain underexplored. For instance, determining whether it's more advantageous to continue training from the most recent checkpoint or to revert to an earlier one to circumvent potential local minima is an open question.

Moreover, PBT's current limitation is its focus solely on training hyperparameters without accommodating changes in model architecture. An area for future exploration is how to inherit model parameters effectively across different architectures, which could potentially unlock new efficiencies in model training and adaptation. These inquiries could significantly advance the field of HPO by reducing the substantial computational demands associated with training large-scale machine learning models and facilitating more dynamic and adaptable optimization strategies.

8.2.3 Large Language Models for Hyperparameter Optimization

LLMs are making significant strides across various domains in machine learning, but their potential in HPO remains a novel area of exploration. A study by Zhang et al.

(2023a) demonstrates an innovative approach where LLMs are prompted with an initial set of instructions—outlining the specific dataset, model, and hyperparameters—to recommend hyperparameters for evaluation. Following the evaluation, the performance metrics from the validation set are fed back into the LLM to suggest the next set of hyperparameters. This iterative prompting strategy has shown results that are comparable to or even surpass traditional HPO methods such as random search and Bayesian optimization on the HPOBench (Eggensperger et al., 2021), which includes diverse machine learning algorithms like support vector machines, logistic regression, XGBoost, random forests, and MLPs. However, the efficacy of LLMs in HPO for NMT systems remains untested.

Exploring how LLMs can be adapted to HPO is another promising research direction. In-context learning and fine-tuning are potential methods to achieve that goal. Moreover, the capability of LLMs in transfer learning for HPO can also be examined—specifically, whether an LLM can leverage the knowledge acquired from HPO processes in one task to improve hyperparameter tuning in another task based on task descriptions. An even more ambitious question is whether LLMs could suggest optimal hyperparameter configurations for training themselves, thus closing the loop on their own optimization process.

These questions underscore the potential for LLMs to revolutionize the field of HPO, extending their utility beyond traditional applications and into the realm of meta-learning where models contribute to their own training efficacy. Exploring these possibilities could lead to significant advancements in how machine learning models are optimized and applied across various domains including NMT.

8.3 Closing Remarks

In this dissertation, we have explored HPO for NMT with a particular focus on the computational resources available at the time of writing. The methods and insights presented here are tailored to the limitations and possibilities of current technology. However, as advancements in computational power continue to accelerate, the systematic HPO strategies discussed in this work will likely become even more applicable and indispensable to the development of machine learning systems. The increasing availability of advanced computing resources will also enable more researchers to innovate in this field.

Moreover, this progression invites us to imagine a future where HPO is no longer hindered by resource limitations, potentially transforming how we approach machine learning problems altogether. It opens up exciting possibilities for what HPO could achieve in an era where the only limits are our imagination and ingenuity. The work presented here serves as a stepping stone towards that future, offering both a foundation for further exploration and a glimpse into the vast potential that lies ahead.

Bibliography

Samira Abnar and Willem Zuidema (2020). "Quantifying Attention Flow in Transformers". *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pages 4190–4197. DOI: 10.18653/v1/2020.acl-main.385. URL: https://aclanthology.org/2020.acl-main.385 (cited on page 123).

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. (2023). "Gpt-4 technical report". *arXiv preprint arXiv:2303.08774*. URL: https://arxiv.org/abs/2303.08774 (cited on page 31).

Steven Adriaensen, Herilalaina Rakotoarison, Samuel Müller, and Frank Hutter (2024). "Efficient bayesian learning curve extrapolation using prior-data fitted networks". *Advances in Neural Information Processing Systems* 36. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/3f1a5e8bfcc3005724d246abe454c1e5-Paper-Conference.pdf (cited on page 157).

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama (2019). "Optuna: A next-generation hyperparameter optimization framework". *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631 (cited on page 60).

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W Hogg, and Michael O'Neil (2014). "Fast direct methods for gaussian processes". *arXiv preprint arXiv:1403.6015* (cited on page 108).

Ali Araabi and Christof Monz (2020). "Optimizing transformer for low-resource neural machine translation". *arXiv preprint arXiv:2011.02266* (cited on page 124).

Parnia Bahar, Tamer Alkhouli, Jan-Thorsten Peter, Christopher Jan-Steffen Brix, and Hermann Ney (2017). "Empirical investigation of optimization algorithms in neural machine translation". *The Prague Bulletin of Mathematical Linguistics* 108.1, pages 13–25 (cited on page 124).

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". *Proceedings of the 3rd International Conference on Learning Representations* (cited on pages 7, 65).

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik (2017). "Accelerating neural architecture search using performance prediction". *arXiv preprint arXiv:1705.10823*. URL: https://arxiv.org/pdf/1705.10823.pdf (cited on page 157).

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy (2020). "BoTorch: A framework for efficient Monte-Carlo Bayesian optimization". *Advances in neural information processing systems* 33, pages 21524–21538 (cited on page 60).

Satanjeev Banerjee and Alon Lavie (2005). "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments". *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72 (cited on page 8).

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag (2013). "Collaborative hyperparameter tuning". *Proceedings of the 30th International conference on machine learning* (cited on page 65).

Jasmijn Bastings, Yonatan Belinkov, Emmanuel Dupoux, Mario Giulianelli, Dieuwke Hupkes, Yuval Pinter, and Hassan Sajjad, editors (2021). *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Punta Cana, Dominican Republic: Association for Computational Linguistics. URL: https://aclanthology.org/2021.blackboxnlp-1.0 (cited on page 118).

Daniel Beck, Adrià de Gispert, Gonzalo Iglesias, Aurelien Waite, and Bill Byrne (2016). "Speed-constrained tuning for statistical machine translation using bayesian optimization". *arXiv preprint arXiv:1604.05073* (cited on page 61).

Alexandre Bérard, Ioan Calapodescu, and Claude Roux (2019). "Naver Labs Europe' s Systems for the WMT19 Machine Translation Robustness Task". *Proceedings of the*

Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), pages 526–532. URL: https://aclanthology.org/W19-5361.pdf (cited on page 74).

James Bergstra and Yoshua Bengio (2012). "Random search for hyper-parameter optimization". *Journal of Machine Learning Research* 13.Feb, pages 281–305 (cited on pages xxiv, 40, 41, 108).

James Bergstra, Daniel Yamins, and David Cox (2013). "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". *International conference on machine learning*. PMLR, pages 115–123 (cited on pages 47, 60).

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (2011). "Algorithms for hyper-parameter optimization". *Proceedings of the 25th Advances in Neural Information Processing Systems* (cited on pages 47, 65).

Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. (2023). "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges". *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 13.2, e1484 (cited on pages 61, 93).

Aloïs Bissuel (2020). "Hyper-parameter optimization algorithms: a short review". *Medium* (cited on pages xxv, 53).

Antal Van den Bosch (2004). "Wrapped progressive sampling search for optimizing learning algorithm parameters". *Proceedings of the Belgium-Netherlands Conference* *on Artificial Intelligence, BNAIC'04, 21-22 oktober, 2004*. Unknown Publisher, pages 219–226 (cited on page 51).

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le (2017). "Massive exploration of neural machine translation architectures". *arXiv preprint arXiv:1703.03906* (cited on page 124).

Eric Brochu, Vlad M Cora, and Nando De Freitas (2010). "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". *arXiv preprint arXiv:1012.2599* (cited on page 98).

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang (2018). "Efficient architecture search by network transformation". *Thirty-Second AAAI Conference on Artificial Intelligence* (cited on page 65).

Akshay Chandrashekaran and Ian R Lane (2017). "Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds". *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10*. Springer, pages 477–492. URL: http://akshayc.com/publications/ecml2017_gelc_submitted.pdf (cited on page 157).

Shweta Chauhan and Philemon Daniel (2023). "A comprehensive survey on various fully automatic machine translation evaluation metrics". *Neural Processing Letters* 55.9, pages 12663–12717 (cited on page 8).

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). "On the properties of neural machine translation: Encoder-decoder approaches". *arXiv preprint arXiv:1409.1259* (cited on page 7).

Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, et al. (2022). "No language left behind: Scaling human-centered machine translation". *arXiv preprint arXiv:2207.04672* (cited on page 182).

Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen (2020). "A Survey of the State of Explainable AI for Natural Language Processing". *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. Suzhou, China: Association for Computational Linguistics, pages 447–459. URL: https://aclanthology.org/2020.aacl-main.46 (cited on page 123).

Kiron Deb, Xuan Zhang, and Kevin Duh (2022). "Post-Hoc Interpretation of Transformer Hyperparameters with Explainable Boosting Machines". *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, pages 51–61. URL: https://aclanthology.org/2022.blackboxnlp-1.5 (cited on page 4).

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer (2023). "Qlora: Efficient finetuning of quantized llms". *arXiv preprint arXiv:2305.14314* (cited on page 76).

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith (2020). "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping". *arXiv preprint arXiv:2002.06305*. URL: https://arxiv.org/pdf/2002.06305 (cited on pages 29, 143).

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter (2015). "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves". *Twenty-fourth international joint conference on artificial intelligence*. URL: https://ml.informatik.uni-freiburg.de/wp-content/uploads/papers/15-IJCAI-Extrapolation_of_Learning_Curves.pdf (cited on page 157).

John Duchi, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research* 12.7 (cited on page 25).

Kevin Duh (2018). *The Multitarget TED Talks Task*. http://www.cs.jhu.edu/~kevinduh/a/multitarget-tedtalks/ (cited on page 72).

Kevin Duh and Xuan Zhang (2023a). "AutoML for NLP". *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, pages 25–26 (cited on page 62).

Kevin Duh and Xuan Zhang (2023b). "Tutorial: AutoML for NLP". *The 17th Conference of the European Chapter of the Association for Computational Linguistics*, page 25 (cited on page 62).

Russell C Eberhart and Yuhui Shi (1998). "Comparison between genetic algorithms and particle swarm optimization". *International Conference on Evolutionary Programming* (cited on pages 48, 97).

Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2015). "Efficient benchmarking of hyperparameter optimizers via surrogates". *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (cited on page 90).

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter (2021). "HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO". *arXiv preprint arXiv:2109.06716*. URL: https://arxiv.org/abs/2109.06716 (cited on page 184).

Michael TM Emmerich, André H Deutz, and Jan Willem Klinkenberg (2011). "Hypervolume-based expected improvement: Monotonicity properties and exact computation". *2011 IEEE Congress of Evolutionary Computation (CEC)* (cited on pages 59, 106).

Stefan Falkner, Aaron Klein, and Frank Hutter (2018). "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". *International Conference on Machine Learning* (cited on pages 47, 56).

Matthias Feurer and Frank Hutter (2019a). “Hyperparameter Optimization”. Edited by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer. Chapter 1, pages 3–38 (cited on pages xxiv, 40, 45, 46).

Matthias Feurer and Frank Hutter (2019b). “Hyperparameter optimization”. *Automated Machine Learning*. Springer, pages 3–33 (cited on pages 96, 181).

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter (2015a). “Efficient and robust automated machine learning”. *Advances in neural information processing systems* 28 (cited on page 60).

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter (2015b). “Initializing bayesian hyperparameter optimization via meta-learning”. *Twenty-Ninth AAAI Conference on Artificial Intelligence* (cited on page 89).

Peter I Frazier (2018). “A tutorial on bayesian optimization”. *arXiv preprint arXiv:1807.02811* (cited on page 98).

Alex A Freitas (2019). “Automated machine learning for studying the trade-off between predictive accuracy and interpretability”. *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, pages 48–66 (cited on page 139).

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin (2017). “Convolutional sequence to sequence learning”. *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (cited on page 65).

David Ginsbourger, Janis Janusevskis, and Rodolphe Le Riche (2011). "Dealing with asynchronicity in parallel Gaussian process based global optimization". PhD thesis. Mines Saint-Etienne (cited on page 38).

Parke Godfrey, Ryan Shipley, and Jarek Gryz (2007). "Algorithms and analyses for maximal vector computation". *The VLDB Journal—The International Journal on Very Large Data Bases* 16.1, pages 5–28 (cited on page 88).

Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff (2012). "Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages". *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), pages 759–765. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/327_Paper.pdf (cited on page 168).

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley (2017). "Google vizier: A service for black-box optimization". *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495 (cited on page 61).

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan (2022a). "The Flores-101 Evaluation Benchmark for Low-Resource and Multilingual Machine Translation". *Transactions of the Association for*

Computational Linguistics 10, pages 522–538. DOI: 10.1162/tacl_a_00474. URL: https://aclanthology.org/2022.tacl-1.30 (cited on page 170).

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc' Aurelio Ranzato, Francisco Guzmán, and Angela Fan (2022b). "The flores-101 evaluation benchmark for low-resource and multilingual machine translation". *Transactions of the Association for Computational Linguistics* 10, pages 522–538 (cited on page 76).

Nikolaus Hansen (2016). "The CMA evolution strategy: A tutorial". *arXiv preprint arXiv:1604.00772* (cited on page 48).

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar (2016). "COCO: Performance assessment". *arXiv preprint arXiv:1605.03560* (cited on page 89).

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla (2023). "How good are gpt models at machine translation? a comprehensive evaluation". *arXiv preprint arXiv:2302.09210*. URL: https://arxiv.org/pdf/2302.09210 (cited on page 66).

Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams (2016). "Predictive entropy search for multi-objective bayesian optimization". *International conference on machine learning*. PMLR, pages 1492–1501 (cited on page 56).

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani (2014). "Predictive entropy search for efficient global optimization of black-box functions". *Advances in neural information processing systems* 27 (cited on page 42).

Felix Hieber, Michael Denkowski, Tobias Domhan, Barbara Darques Barros, Celina Dong Ye, Xing Niu, Cuong Hoang, Ke Tran, Benjamin Hsu, Maria Nadejde, Surafel Lakew, Prashant Mathur, Anna Currey, and Marcello Federico (2022). *Sockeye 3: Fast Neural Machine Translation with PyTorch*. DOI: 10.48550/ARXIV.2207.05851. URL: https://arxiv.org/abs/2207.05851 (cited on page 161).

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post (2017). "Sockeye: A toolkit for neural machine translation". *arXiv preprint arXiv:1712.05690* (cited on page 73).

Daniel Horn and Bernd Bischl (2016). "Multi-objective parameter configuration of machine learning algorithms using model-based optimization". *2016 IEEE symposium series on computational intelligence (SSCI)*. IEEE, pages 1–8 (cited on page 56).

Chi Hu, Chenglong Wang, Xiangnan Ma, Xia Meng, Yinqiao Li, Tong Xiao, Jingbo Zhu, and Changliang Li (2021a). "RankNAS: Efficient Neural Architecture Search by Pairwise Ranking". *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2469–2480 (cited on page 62).

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2021b). “Lora: Low-rank adaptation of large language models”. *arXiv preprint arXiv:2106.09685* (cited on page 66).

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2014). “An efficient approach for assessing hyperparameter importance”. *International conference on machine learning*. PMLR, pages 754–762 (cited on page 139).

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown (2011). “Sequential model-based optimization for general algorithm configuration”. *Proceedings of the 5th International Conference on Learning and Intelligent Optimization* (cited on pages 46, 60, 65).

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors (2019a). *Automated Machine Learning - Methods, Systems, Challenges*. Springer (cited on page 118).

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors (2019b). *Automatic Machine Learning: Methods, Systems, Challenges*. Springer (cited on pages xxiv, 40).

Christian Igel (2005). “Multi-objective model selection for support vector machines”. *International conference on evolutionary multi-criterion optimization*. Springer, pages 534–546 (cited on page 56).

Hirofumi Inaguma, Xuan Zhang, Zhiqi Wang, Adithya Renduchintala, Shinji Watanabe, and Kevin Duh (2018). “The JHU/KyotoU Speech Translation System for IWSLT 2018”. *Proceedings of the International Workshop on Spoken Language*

Translation. Bruges, Belgium. URL: https://workshop2018.iwslt.org/downloads/Proceedings_IWSLT_2018.pdf (cited on page 4).

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. (2017). "Population based training of neural networks". *arXiv preprint arXiv:1711.09846*. URL: https://arxiv.org/pdf/1711.09846 (cited on pages xxv, 48, 51, 183).

Kevin Jamieson and Ameet Talwalkar (2016). "Non-stochastic best arm identification and hyperparameter optimization". *Artificial intelligence and statistics*. PMLR, pages 240–248 (cited on pages 52, 143, 144, 166).

Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu (2023). "Is ChatGPT a good translator? A preliminary study". *arXiv preprint arXiv:2301.08745* 1.10. URL: https://arxiv.org/abs/2301.08745 (cited on page 31).

Donald R Jones (2001). "A taxonomy of global optimization methods based on response surfaces". *Journal of global optimization* 21, pages 345–383 (cited on pages 42, 46).

Donald R Jones, Matthias Schonlau, and William J Welch (1998). "Efficient global optimization of expensive black-box functions". *Journal of Global Optimization* 13.4, pages 455–492 (cited on pages 41, 42, 96).

Nal Kalchbrenner and Phil Blunsom (2013). “Recurrent continuous translation models”. *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1700–1709 (cited on page 7).

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing (2020). “Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly”. *Journal of Machine Learning Research* 21.81, pages 1–27 (cited on page 60).

Zohar Karnin, Tomer Koren, and Oren Somekh (2013). “Almost optimal exploration in multi-armed bandits”. *International conference on machine learning*. PMLR, pages 1238–1246. URL: https://proceedings.mlr.press/v28/karnin13.pdf (cited on pages 143, 144).

Diederik P Kingma and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. *Proceedings of the 3rd International Conference on Learning Representations* (cited on page 25).

Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez (2019). “Meta-Surrogate Benchmarking for Hyperparameter Optimization”. *arXiv preprint arXiv:1905.12982* (cited on page 90).

Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter (2022). “Learning curve prediction with Bayesian neural networks”. *International conference* *on learning representations*. URL: https://openreview.net/pdf?id=S11KBYclx (cited on page 157).

Aaron Klein and Frank Hutter (2019). “Tabular Benchmarks for Joint Architecture and Hyperparameter Optimization”. *arXiv preprint arXiv:1905.04970* (cited on pages 66, 84, 90).

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui (2020). “Attention is Not Only a Weight: Analyzing Transformers with Vector Norms”. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, pages 7057–7075. DOI: 10.18653/v1/2020.emnlp-main.574. URL: https://aclanthology.org/2020.emnlp-main.574 (cited on page 123).

Philipp Koehn (2020). *Neural machine translation*. Cambridge University Press (cited on page 7).

Philipp Koehn, Franz Josef Och, and Daniel Marcu (2003). “Statistical phrase-based translation”. *2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Langauge Technology (HLT-NAACL 2003)*. Association for Computational Linguistics, pages 48–54 (cited on page 6).

Prasanth Kolachina, Nicola Cancedda, Marc Dymetman, and Sriram Venkatapathy (2012). “Prediction of learning curves in machine translation”. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1:*

Long Papers), pages 22–30. URL: https://aclanthology.org/P12-1003.pdf (cited on page 157).

Brent Komer, James Bergstra, and Chris Eliasmith (2014). "Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn." *Scipy*, pages 32–37 (cited on page 47).

Tammo Krueger, Danny Panknin, and Mikio L Braun (2015). "Fast cross-validation via sequential testing." *J. Mach. Learn. Res.* 16.1, pages 1103–1155 (cited on page 51).

Taku Kudo and John Richardson (2018). "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics (cited on page 30).

Harold J Kushner (1964). "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise" (cited on page 42).

Seungjun Lee, Jungseob Lee, Hyeonseok Moon, Chanjun Park, Jaehyung Seo, Sugyeong Eo, Seonmin Koo, and Heuiseok Lim (2023). "A survey on evaluation metrics for machine translation". *Mathematics* 11.4, page 1006 (cited on pages xxiv, 8, 9).

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar (2020a). "A system for massively parallel hyperparameter tuning". *Proceedings of Machine Learning and Systems* 2, pages 230–246. URL: https://proceedings.mlsys.org/paper_ files/paper/2020/file/a06f20b349c6cf09a6b171c71b88bbfc-Paper.pdf (cited on page 53).

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar (2020b). “A System for Massively Parallel Hyperparameter Tuning”. *Proceedings of Machine Learning and Systems*. Edited by I. Dhillon, D. Papailiopoulos, and V. Sze. Volume 2, pages 230–246. URL: https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf (cited on pages 161, 167).

Liam Li and Ameet Talwalkar (2019). “Random Search and Reproducibility for Neural Architecture Search”. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)* (cited on page 70).

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar (2018). “Hyperband: A novel bandit-based approach to hyperparameter optimization”. *Journal of Machine Learning Research* 18.185, pages 1–52. URL: https://arxiv.org/pdf/1603.06560 (cited on pages 54, 143).

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet S. Talwalkar (2016). “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. *J. Mach. Learn. Res.* 18, 185:1–185:52 (cited on page 168).

Xian Li, Paul Michel, Antonios Anastasopoulos, Yonatan Belinkov, Nadir Durrani, Orhan Firat, Philipp Koehn, Graham Neubig, Juan Pino, and Hassan Sajjad (2019).

"Findings of the First Shared Task on Machine Translation Robustness". *Proceedings of the Fourth Conference on Machine Translation* (cited on pages 72, 74).

Robert Lim, Kenneth Heafield, Hieu Hoang, Mark Briers, and Allen Malony (2018). "Exploring Hyper-Parameter Optimization for Neural Machine Translation on GPU Architectures". *arXiv preprint arXiv:1805.02094* (cited on page 123).

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy (2018a). "Progressive neural architecture search". *Proceedings of the European Conference on Computer Vision (ECCV)* (cited on page 65).

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu (2018b). "Hierarchical Representations for Efficient Architecture Search". *International Conference on Learning Representations* (cited on page 65).

Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor (2017). "Particle swarm optimization for hyper-parameter selection in deep neural networks". *Proceedings of the genetic and evolutionary computation conference*, pages 481–488 (cited on page 48).

Yin Lou, Rich Caruana, and Johannes Gehrke (2012). "Intelligible Models for Classification and Regression". *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China: Association for Computing Machinery, pages 150 – 158. ISBN: 9781450314626. DOI:

10.1145/2339530.2339556. URL: https://doi.org/10.1145/2339530.2339556 (cited on page 119).

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker (2013). "Accurate Intelligible Models with Pairwise Interactions". *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, pages 623 – 631. ISBN: 9781450321747. DOI: 10.1145/2487575.2487579. URL: https://doi.org/10.1145/2487575.2487579 (cited on page 119).

Gang Luo (2016). "A review of automatic selection methods for machine learning algorithms and hyper-parameter values". *Network Modeling Analysis in Health Informatics and Bioinformatics* 5.1, page 18 (cited on page 96).

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. (2013). "Rectifier nonlinearities improve neural network acoustic models". *Proc. icml*. Volume 30. 1. Atlanta, GA, page 3 (cited on page 21).

R. Marler and Jasbir Arora (2004). "Survey of multi-objective optimization methods for engineering". *Structural and Multidisciplinary Optimization* 26, pages 369–395 (cited on page 164).

Jonas Mockus (1974). "On Bayesian methods for seeking the extremum". *Proceedings of the IFIP Technical Conference*, pages 400–404 (cited on page 41).

Julia Moosbauer, Julia Herbinger, Giuseppe Casalicchio, Marius Lindauer, and Bernd Bischl (2021). "Explaining Hyperparameter Optimization via Partial Dependence Plots". *Advances in Neural Information Processing Systems* 34 (cited on page 139).

Yasmin Moslem, Rejwanul Haque, and Andy Way (2023). "Fine-tuning large language models for adaptive machine translation". *arXiv preprint arXiv:2312.12740*. URL: https://arxiv.org/pdf/2312.12740 (cited on page 75).

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. (2022). "Crosslingual generalization through multitask finetuning". *arXiv preprint arXiv:2211.01786* (cited on page 76).

Kevin P Murphy (2022). *Probabilistic machine learning: an introduction*. MIT press (cited on pages xxiv, 21, 22, 24, 25).

Kenton Murray (2020). *Learning Hyperparameters for Neural Machine Translation*. University of Notre Dame (cited on page 61).

Vinod Nair and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814 (cited on page 20).

Jason Naradowksy, Xuan Zhang, and Kevin Duh (2020). "Machine Translation System Selection from Bandit Feedback". *Proceedings of the 15th Conference of the Association for Machine Translation in the Americas*. URL: https://arxiv.org/pdf/2002.09646 (cited on page 4).

Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana (2019). "Interpretml: A unified framework for machine learning interpretability". *arXiv preprint arXiv:1909.09223* (cited on pages 119, 126).

OpenAI (2023). *ChatGPT: Optimizing Language Models for Dialogue*. https://www.openai.com/chatgpt. Accessed: 2023-08-14 (cited on page 31).

David Orive, G Sorrosal, Cruz Enrique Borges, Cristina Martin, and A Alonso-Vicario (2014). "Evolutionary algorithms for hyperparameter tuning on neural networks models". *Proceedings of the 26th european modeling & simulation symposium. Burdeos, France*, pages 402–409 (cited on page 48).

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu (2002). "BLEU: a method for automatic evaluation of machine translation". *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (cited on page 8).

Johann Petrak (2000). "Fast subsampling performance estimates for classification algorithm selection". *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pages 3–14 (cited on page 51).

Florian Pfisterer, Janek Thomas, and Bernd Bischl (2019). "Towards human centered AutoML". *arXiv preprint arXiv:1911.02391* (cited on page 139).

Victor Picheny (2015). "Multiobjective optimization using Gaussian process emulators via stepwise uncertainty reduction". *Statistics and Computing* 25.6, pages 1265–1280 (cited on page 106).

Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze (2008). “Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted $\mathcal{S}$-Metric Selection”. *International Conference on Parallel Problem Solving from Nature*. Springer, pages 784–794 (cited on page 106).

Maja Popović (2015). “chrF: character n-gram F-score for automatic MT evaluation”. *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Lisbon, Portugal: Association for Computational Linguistics, pages 392–395. DOI: 10.18653/v1/W15-3049. URL: https://aclanthology.org/W15-3049 (cited on page 10).

Matt Post (2018). “A Call for Clarity in Reporting BLEU Scores”. *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, pages 186–191. DOI: 10.18653/v1/W18-6319. URL: https://aclanthology.org/W18-6319 (cited on page 8).

Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl (2019). “Tunability: importance of hyperparameters of machine learning algorithms”. *The Journal of Machine Learning Research* 20.1, pages 1934–1965 (cited on page 139).

Carl Edward Rasmussen (2003). “Gaussian processes in machine learning”. *Summer School on Machine Learning*. Springer, pages 63–71 (cited on pages 42, 98, 108).

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le (2019). “Regularized evolution for image classifier architecture search”. *Proceedings of the AAAI Conference on Artificial Intelligence* (cited on page 65).

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie (2020). "COMET: A Neural Framework for MT Evaluation". *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, pages 2685–2702. DOI: 10.18653/v1/2020.emnlp-main.213. URL: https://aclanthology.org/2020.emnlp-main.213 (cited on page 10).

Ashish Sabharwal, Horst Samulowitz, and Gerald Tesauro (2016). "Selecting near-optimal learners via incremental data allocation". *Proceedings of the AAAI Conference on Artificial Intelligence*. Volume 30. 1 (cited on page 51).

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. (2022). "Bloom: A 176b-parameter open-access multilingual language model". *arXiv preprint arXiv:2211.05100* (cited on page 76).

Matthias Schonlau, William J Welch, and Donald R Jones (1998). "Global versus local search in constrained optimization of computer models". *Lecture Notes-Monograph Series*, pages 11–25 (cited on page 100).

Holger Schwenk, Daniel Déchelotte, and Jean-Luc Gauvain (2006). "Continuous space language models for statistical machine translation". *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 723–730 (cited on page 6).

Rico Sennrich, Barry Haddow, and Alexandra Birch (2016). "Neural machine translation of rare words with subword units". *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (cited on pages 30, 73).

Amar Shah and Zoubin Ghahramani (2016). "Pareto frontier learning with expensive correlated objectives". *International Conference on Machine Learning*, pages 1919–1927 (cited on pages xxv, 56, 58, 60, 106).

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas (2015). "Taking the human out of the loop: A review of Bayesian optimization". *Proceedings of the IEEE* 104.1, pages 148–175 (cited on page 98).

Abhinav Sharma, Jan N van Rijn, Frank Hutter, and Andreas Müller (2019). "Hyperparameter importance for image classification by residual neural networks". *International Conference on Discovery Science*. Springer, pages 112–126 (cited on page 139).

Suzanna Sia and Kevin Duh (2023). "In-context Learning as Maintaining Coherency: A Study of On-the-fly Machine Translation Using Large Language Models". *Proceedings of Machine Translation Summit XIV (Volume 1: Research Track)*. URL: https://arxiv.org/pdf/2305.03573 (cited on pages 31, 66).

Dan Simon (2013). *Evolutionary optimization algorithms*. John Wiley & Sons (cited on pages 48, 97).

Jasper Snoek, Hugo Larochelle, and Ryan P Adams (2012). "Practical bayesian optimization of machine learning algorithms". *Proceedings of the 26th Advances in Neural Information Processing Systems* (cited on page 60).

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams (2015). "Scalable bayesian optimization using deep neural networks". *Proceedings of the 32nd International Conference on Machine Learning* (cited on pages 46, 65).

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul (2006). "A study of translation edit rate with targeted human annotation". *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231 (cited on page 10).

Evan R Sparks, Ameet Talwalkar, Daniel Haas, Michael J Franklin, Michael I Jordan, and Tim Kraska (2015). "Automating model search for large scale machine learning". *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380 (cited on page 51).

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter (2016). "Bayesian optimization with robust Bayesian neural networks". *Advances in neural information processing systems* 29 (cited on page 46).

Titte R Srinivas, Bing Ho, Joseph Kang, and Bruce Kaplan (2015). "Post Hoc Analyses: After the Facts". *Transplantation* 99 (cited on page 124).

Ilya Sutskever, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". *Proceedings of the 28th Advances in Neural Information Processing Systems* (cited on pages 7, 65).

Joshua Svenson and Thomas Santner (2016). "Multiobjective optimization of expensive-to-evaluate deterministic computer simulator models". *Computational Statistics & Data Analysis* 94, pages 250–264 (cited on page 106).

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). "Rethinking the inception architecture for computer vision". *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826 (cited on page 27).

Tomohiro Tanaka, Takafumi Moriya, Takahiro Shinozaki, Shinji Watanabe, Takaaki Hori, and Kevin Duh (2016). "Automated structure discovery and parameter tuning of neural network language model based on evolution strategy". *2016 ieee spoken language technology workshop (slt)*. IEEE, pages 665–671 (cited on page 49).

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng (2021). "Synthesizer: Rethinking self-attention for transformer models". *International conference on machine learning*. PMLR, pages 10183–10192 (cited on page 123).

ALS TDI (2022). *What is Post-Hoc Analysis in a Clinical Trial*. URL: https://www.als.net/news/what-is-post-hoc-analysis-in-a-clinical-trial/ (visited on 2022) (cited on page 124).

Brian Thompson, Rebecca Knowles, Xuan Zhang, Huda Khayrallah, Kevin Duh, and Philipp Koehn (2019). "HABLex: Human Annotated Bilingual Lexicons for Experiments in Machine Translation". *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Hong Kong, China. URL: https://aclanthology.org/D19-1142/ (cited on page 4).

Jörg Tiedemann (2012). "Parallel Data, Tools and Interfaces in OPUS". *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), pages 2214–2218. URL: http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf (cited on page 168).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". *Proceedings of the 31st Advances in Neural Information Processing Systems* (cited on pages xxiv, 7, 11, 20, 65, 121).

Savan Visalpara, Krishnateja Killamsetty, and Rishabh Iyer (2021). "A data subset selection framework for efficient hyper-parameter tuning and automatic machine learning". *ICML Workshops*. URL: https://krishnatejakillamsetty.me/files/Hyperparam_SubsetML.pdf (cited on page 182).

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov (2019). "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned". *Proceedings of the 57th Annual Meeting of the Association* *for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pages 5797–5808. DOI: 10.18653/v1/P19-1580. URL: https://aclanthology.org/P19-1580 (cited on page 20).

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han (2020). "HAT: Hardware-Aware Transformers for Efficient Natural Language Processing". *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688 (cited on page 62).

Longyue Wang, Chenyang Lyu, Tianbo Ji, Zhirui Zhang, Dian Yu, Shuming Shi, and Zhaopeng Tu (2023). "Document-level machine translation with large language models". *arXiv preprint arXiv:2304.02210*. URL: https://arxiv.org/abs/2304.02210 (cited on page 31).

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le (2021). "Finetuned language models are zero-shot learners". *arXiv preprint arXiv:2109.01652*. URL: https://arxiv.org/abs/2109.01652 (cited on page 31).

Martin Wistuba and Tejaswini Pedapati (2020). "Learning to rank learning curves". *International Conference on Machine Learning*. PMLR, pages 10303–10312. URL: https://proceedings.mlr.press/v119/wistuba20a/wistuba20a.pdf (cited on pages 144, 151, 155, 157).

David H Wolpert (1996). "The lack of a priori distinctions between learning algorithms". *Neural computation* 8.7, pages 1341–1390 (cited on page 18).

Iordanis Xanthopoulos, Ioannis Tsamardinos, Vassilis Christophides, Eric Simon, and Alejandro Salinger (2020). “Putting the Human Back in the AutoML Loop.” *EDBT/ICDT Workshops* (cited on page 139).

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu (2024). “Harnessing the power of llms in practice: A survey on chatgpt and beyond”. *ACM Transactions on Knowledge Discovery from Data* 18.6, pages 1–32. URL: https://dl.acm.org/doi/pdf/10.1145/3649506 (cited on page 75).

Li Yang and Abdallah Shami (2020). “On hyperparameter optimization of machine learning algorithms: Theory and practice”. *Neurocomputing* 415, pages 295–316. URL: https://arxiv.org/pdf/2007.15745 (cited on page 181).

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter (2019). “Nas-bench-101: Towards reproducible neural architecture search”. *arXiv preprint arXiv:1902.09635* (cited on pages 66, 90).

Tong Yu and Hong Zhu (2020). “Hyper-parameter optimization: A review of algorithms and applications”. *arXiv preprint arXiv:2003.05689* (cited on page 61).

Matthew D Zeiler (2012). “Adadelta: an adaptive learning rate method”. *arXiv preprint arXiv:1212.5701* (cited on page 25).

Arber Zela, Julien Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter (2020). “Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks". *arXiv preprint arXiv:2008.09777*. URL: https://arxiv.org/abs/2008.09777 (cited on page 90).

Michael R Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba (2023a). "Using large language models for hyperparameter optimization". *NeurIPS 2023 Foundation Models for Decision Making Workshop*. URL: https://arxiv.org/pdf/2312.04528 (cited on page 183).

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi (2019a). "Bertscore: Evaluating text generation with bert". *arXiv preprint arXiv:1904.09675* (cited on page 10).

Xuan Zhang and Kevin Duh (2020). "Reproducible and Efficient Benchmarks for Hyperparameter Optimization of Neural Machine Translation Systems". *Transactions of the Association for Computational Linguistics* 8, pages 393–408. DOI: 10.1162/tacl_a_00322. URL: https://aclanthology.org/2020.tacl-1.26 (cited on pages 4, 132, 136).

Xuan Zhang and Kevin Duh (2021). "Approaching Sign Language Gloss Translation as a Low-Resource Machine Translation Task". *Proceedings of the 1st International Workshop on Automatic Translation for Signed and Spoken Languages (AT4SSL)*. Virtual: Association for Machine Translation in the Americas, pages 60–70. URL: https://aclanthology.org/2021.mtsummit-at4ssl.7 (cited on page 4).

Xuan Zhang and Kevin Duh (2023a). "Handshape-Aware Sign Language Recognition: Extended Datasets and Exploration of Handshape-Inclusive Methods". *The 2023*

Conference on Empirical Methods in Natural Language Processing. URL: https://openreview.net/pdf?id=qJqJXpysnh (cited on page 4).

Xuan Zhang and Kevin Duh (2023b). "Sign Language Gloss Translation". *Sign Language Machine Translation*. Edited by Andy Way, Loraine Leeson, Dimitar Shterionov, and Christian Rathmann. Springer. URL: https://link.springer.com/book/9783031473616 (cited on page 4).

Xuan Zhang and Kevin Duh (2024). "Best Practices of Successive Halving on Neural Machine Translation and Large Language Models". Proceedings of the 16th biennial conference of the Association for Machine Translation. URL: https://www.cs.jhu.edu/~xzhan138/papers/AMTA2024_LC.pdf (cited on page 4).

Xuan Zhang, Kevin Duh, and Paul McNamee (2023b). "A Hyperparameter Optimization Toolkit for Neural Machine Translation Research". *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Toronto, Canada: Association for Computational Linguistics, pages 161–168. URL: https://aclanthology.org/2023.acl-demo.15 (cited on page 4).

Xuan Zhang, Gaurav Kumar, Huda Khayrallah, Kenton Murray, Jeremy Gwinnup, Marianna J Martindale, Paul McNamee, Kevin Duh, and Marine Carpuat (2018). "An Empirical Exploration of Curriculum Learning for Neural Machine Translation". *arXiv preprint arXiv:1811.00739*. URL: https://arxiv.org/abs/1811.00739 (cited on page 4).

Xuan Zhang, Navid Rajabi, Kevin Duh, and Philipp Koehn (2023c). "Machine Translation with Large Language Models: Prompting, Few-shot Learning, and Fine-tuning with QLoRA". *Proceedings of the Eighth Conference on Machine Translation*. Singapore: Association for Computational Linguistics, pages 468–481. URL: https://aclanthology.org/2023.wmt-1.43 (cited on pages 4, 66, 75).

Xuan Zhang, Navid Rajabi, Kevin Duh, and Philipp Koehn (2023d). "Machine Translation with Large Language Models: Prompting, Few-shot Learning, and Fine-tuning with QLoRA". *Proceedings of the Eighth Conference on Machine Translation*. Singapore: Association for Computational Linguistics, pages 468–481. URL: https://aclanthology.org/2023.wmt-1.43 (cited on pages 31, 33).

Xuan Zhang, Pamela Shapiro, Gaurav Kumar, Paul McNamee, Marine Carpuat, and Kevin Duh (2019b). "Curriculum Learning for Domain Adaptation in Neural Machine Translation". *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Minneapolis, Minnesota. URL: https://arxiv.org/abs/1905.05816 (cited on page 4).

Xuan Zhang, Yu Wu, Fangyun Wei, and Shujie Liu (2023e). "Exploration of Pseudo-gloss in Sign Language Translation". In submission (cited on page 4).

Yuekai Zhao, Li Dong, Yelong Shen, Zhihua Zhang, Furu Wei, and Weizhu Chen (2021). "Memory-Efficient Differentiable Transformer Architecture Search". *Findings of the*

Association for Computational Linguistics: ACL-IJCNLP 2021, pages 4254–4264 (cited on page 62).

Dawei Zhu, Pinzhen Chen, Miaoran Zhang, Barry Haddow, Xiaoyu Shen, and Dietrich Klakow (2024). “Fine-Tuning Large Language Models to Translate: Will a Touch of Noisy Data in Misaligned Languages Suffice?” *arXiv preprint arXiv:2404.14122*. URL: https://arxiv.org/pdf/2404.14122 (cited on page 75).

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Lingpeng Kong, Jiajun Chen, Lei Li, and Shujian Huang (2023). “Multilingual machine translation with large language models: Empirical results and analysis”. *arXiv preprint arXiv:2304.04675*. URL: https://arxiv.org/pdf/2304.04675 (cited on page 66).

Xiaojin Zhu (2005). “Semi-supervised learning with graphs”. PhD thesis (cited on pages 101, 105).

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty (2003). “Semi-supervised learning using gaussian fields and harmonic functions”. *Proceedings of the 20th International Conference on Machine Learning (ICML-03)* (cited on page 101).

Eckart Zitzler and Lothar Thiele (1998). “Multiobjective optimization using evolutionary algorithms—a comparative case study”. *International Conference on Parallel Problem Solving from Nature* (cited on pages 58, 106).

Marc-André Zöller and Marco F Huber (2021). “Benchmark and survey of automated machine learning frameworks”. *Journal of artificial intelligence research* 70, pages 409–472. URL: https://arxiv.org/pdf/1904.12054 (cited on page 90).

Barret Zoph and Quoc V Le (2016). “Neural architecture search with reinforcement learning”. *arXiv preprint arXiv:1611.01578* (cited on page 65).