Language Models

Philipp Koehn

23 April 2024
Language models

- **Language models** answer the question:
  
  *How likely is a string of English words good English?*

- Help with word order

  \[ p_{LM}(\text{the house is small}) > p_{LM}(\text{small the is house}) \]

- Help with word choice

  \[ p_{LM}(\text{I am going home}) > p_{LM}(\text{I am going house}) \]
N-Gram Language Models

• Given: a string of English words $W = w_1, w_2, w_3, ..., w_n$

• Question: what is $p(W)$?

• Sparse data: Many good English sentences will not have been seen before

→ Decomposing $p(W)$ using the chain rule:

$$p(w_1, w_2, w_3, ..., w_n) = p(w_1) p(w_2|w_1) p(w_3|w_1, w_2) ... p(w_n|w_1, w_2, ... w_{n-1})$$

(not much gained yet, $p(w_n|w_1, w_2, ... w_{n-1})$ is equally sparse)
• **Markov assumption:**
  - only previous history matters
  - limited memory: only last $k$ words are included in history
    (older words less relevant)
  $\rightarrow$ *$k$th order Markov model*

• For instance 2-gram language model:

\[
p(w_1, w_2, w_3, \ldots, w_n) \sim p(w_1) \ p(w_2|w_1) \ p(w_3|w_2) \ldots p(w_n|w_{n-1})
\]

• What is conditioned on, here $w_{i-1}$ is called the **history**
Estimating N-Gram Probabilities

- Maximum likelihood estimation

\[ p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \]

- Collect counts over a large text corpus

- Millions to billions of words are easy to get
  (trillions of English words available on the web)
Example: 3-Gram

- Counts for trigrams and estimated word probabilities

<table>
<thead>
<tr>
<th>word</th>
<th>c.</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>paper</td>
<td>801</td>
<td>0.458</td>
</tr>
<tr>
<td>group</td>
<td>640</td>
<td>0.367</td>
</tr>
<tr>
<td>light</td>
<td>110</td>
<td>0.063</td>
</tr>
<tr>
<td>party</td>
<td>27</td>
<td>0.015</td>
</tr>
<tr>
<td>ecu</td>
<td>21</td>
<td>0.012</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>word</th>
<th>c.</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross</td>
<td>123</td>
<td>0.547</td>
</tr>
<tr>
<td>tape</td>
<td>31</td>
<td>0.138</td>
</tr>
<tr>
<td>army</td>
<td>9</td>
<td>0.040</td>
</tr>
<tr>
<td>card</td>
<td>7</td>
<td>0.031</td>
</tr>
<tr>
<td>,</td>
<td>5</td>
<td>0.022</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>word</th>
<th>c.</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>box</td>
<td>16</td>
<td>0.296</td>
</tr>
<tr>
<td>.</td>
<td>6</td>
<td>0.111</td>
</tr>
<tr>
<td>flag</td>
<td>6</td>
<td>0.111</td>
</tr>
<tr>
<td>,</td>
<td>3</td>
<td>0.056</td>
</tr>
<tr>
<td>angel</td>
<td>3</td>
<td>0.056</td>
</tr>
</tbody>
</table>

- 225 trigrams in the Europarl corpus start with the red
- 123 of them end with cross
→ maximum likelihood probability is \( \frac{123}{225} = 0.547 \).
How good is the LM?

- A good model assigns a text of real English $W$ a high probability
- This can be also measured with cross entropy:

$$H(W) = -\frac{1}{n} \log_2 p(W_1^n)$$

- Or, perplexity

$$\text{perplexity}(W) = 2^{H(W)}$$
## Example: 3-Gram

<table>
<thead>
<tr>
<th>prediction</th>
<th>$p_{LM}$</th>
<th>$-\log_2 p_{LM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{LM}(i</td>
<td>&lt;/s&gt;&lt;s&gt;)$</td>
<td>0.109</td>
</tr>
<tr>
<td>$p_{LM}(\text{would}</td>
<td>&lt;s&gt;i)$</td>
<td>0.144</td>
</tr>
<tr>
<td>$p_{LM}(\text{like}</td>
<td>i \text{ would})$</td>
<td>0.489</td>
</tr>
<tr>
<td>$p_{LM}(\text{to}</td>
<td>\text{would like})$</td>
<td>0.905</td>
</tr>
<tr>
<td>$p_{LM}(\text{commend}</td>
<td>\text{like to})$</td>
<td>0.002</td>
</tr>
<tr>
<td>$p_{LM}(\text{the}</td>
<td>\text{to commend})$</td>
<td>0.472</td>
</tr>
<tr>
<td>$p_{LM}(\text{rapporteur}</td>
<td>\text{commend the})$</td>
<td>0.147</td>
</tr>
<tr>
<td>$p_{LM}(\text{on}</td>
<td>\text{the rapporteur})$</td>
<td>0.056</td>
</tr>
<tr>
<td>$p_{LM}(\text{his}</td>
<td>\text{rapporteur on})$</td>
<td>0.194</td>
</tr>
<tr>
<td>$p_{LM}(\text{work}</td>
<td>\text{on his})$</td>
<td>0.089</td>
</tr>
<tr>
<td>$p_{LM}(.</td>
<td>\text{his work})$</td>
<td>0.290</td>
</tr>
<tr>
<td>$p_{LM}(&lt;/s&gt;</td>
<td>\text{work}.)$</td>
<td>0.99999</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td>2.634</td>
</tr>
</tbody>
</table>
## Comparison 1–4-Gram

<table>
<thead>
<tr>
<th>word</th>
<th>unigram</th>
<th>bigram</th>
<th>trigram</th>
<th>4-gram</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>6.684</td>
<td>3.197</td>
<td>3.197</td>
<td>3.197</td>
</tr>
<tr>
<td>would</td>
<td>8.342</td>
<td>2.884</td>
<td>2.791</td>
<td>2.791</td>
</tr>
<tr>
<td>like</td>
<td>9.129</td>
<td>2.026</td>
<td>1.031</td>
<td>1.290</td>
</tr>
<tr>
<td>to</td>
<td>5.081</td>
<td>0.402</td>
<td>0.144</td>
<td>0.113</td>
</tr>
<tr>
<td>commend</td>
<td>15.487</td>
<td>12.335</td>
<td>8.794</td>
<td>8.633</td>
</tr>
<tr>
<td>the</td>
<td>3.885</td>
<td>1.402</td>
<td>1.084</td>
<td>0.880</td>
</tr>
<tr>
<td>rapporteur</td>
<td>10.840</td>
<td>7.319</td>
<td>2.763</td>
<td>2.350</td>
</tr>
<tr>
<td>on</td>
<td>6.765</td>
<td>4.140</td>
<td>4.150</td>
<td>1.862</td>
</tr>
<tr>
<td>his</td>
<td>10.678</td>
<td>7.316</td>
<td>2.367</td>
<td>1.978</td>
</tr>
<tr>
<td>work</td>
<td>9.993</td>
<td>4.816</td>
<td>3.498</td>
<td>2.394</td>
</tr>
<tr>
<td></td>
<td>4.896</td>
<td>3.020</td>
<td>1.785</td>
<td>1.510</td>
</tr>
<tr>
<td>&lt;/s&gt;</td>
<td>4.828</td>
<td>0.005</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>average</td>
<td>8.051</td>
<td>4.072</td>
<td>2.634</td>
<td>2.251</td>
</tr>
<tr>
<td>perplexity</td>
<td>265.136</td>
<td>16.817</td>
<td>6.206</td>
<td>4.758</td>
</tr>
</tbody>
</table>
N-Gram Backoff Language Model

• Previously, we approximated

\[ p(W) = p(w_1, w_2, ..., w_n) \]

• ... by applying the chain rule

\[ p(W) = \sum_i p(w_i|w_1, ..., w_{i-1}) \]

• ... and limiting the history (Markov order)

\[ p(w_i|w_1, ..., w_{i-1}) \approx p(w_i|w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1}) \]

• Each \( p(w_i|w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1}) \) may not have enough statistics to estimate
  → we back off to \( p(w_i|w_{i-3}, w_{i-2}, w_{i-1}) \), \( p(w_i|w_{i-2}, w_{i-1}) \), etc., all the way to \( p(w_i) \)
  — exact details of backing off get complicated — “interpolated Kneser-Ney”
Refinements

- A whole family of back-off schemes
- Skip-n gram models that may back off to $p(w_i|w_{i-2})$
- Class-based models $p(C(w_i)|C(w_{i-4}), C(w_{i-3}), C(w_{i-2}), C(w_{i-1}))$

⇒ We are wrestling here with
  - using as much relevant evidence as possible
  - pooling evidence between words
First Sketch

Output Word

Hidden Layer

History

Softmax

FF

\( h \)

\( w_i \)

\( w_{i-4} \) \( w_{i-3} \) \( w_{i-2} \) \( w_{i-1} \)
Representing Words

- Words are represented with a one-hot vector, e.g.,
  - \textit{dog} = (0,0,0,0,1,0,0,0,0,0,....)
  - \textit{cat} = (0,0,0,0,0,0,0,1,0,....)
  - \textit{eat} = (0,1,0,0,0,0,0,0,0,0,....)

- That’s a large vector!

- Remedies
  - limit to, say, 20,000 most frequent words, rest are OTHER
  - splitting rare words into subwords
  - character-based models
word embeddings
Add an Embedding Layer

- Map each word first into a lower-dimensional real-valued space
- Shared weight matrix $E$
Details (Bengio et al., 2003)

- Add direct connections from embedding layer to output layer

- Activation functions
  - input → embedding: none
  - embedding → hidden: tanh
  - hidden → output: softmax

- Training
  - loop through the entire corpus
  - update between predicted probabilities and 1-hot vector for output word
Word Embeddings

- By-product: embedding of word into continuous space
- Similar contexts $\rightarrow$ similar embedding
- Recall: distributional semantics
Word Embeddings
Word Embeddings
Are Word Embeddings Magic?

- Morphosyntactic regularities (Mikolov et al., 2013)
  - adjectives base form vs. comparative, e.g., good, better
  - nouns singular vs. plural, e.g., year, years
  - verbs present tense vs. past tense, e.g., see, saw

- Semantic regularities
  - clothing is to shirt as dish is to bowl
  - evaluated on human judgment data of semantic similarities
recurrent neural networks
Recurrent Neural Networks

- Start: predict second word from first
- Mystery layer with nodes all with value 1
Recurrent Neural Networks

Softmax

$\tanh$

Embed

$w_1$

Copy

Output Word

Hidden Layer

Embedding

History

W1

$0$

W2
Recurrent Neural Networks
Training

- Process first training example
- Update weights with back-propagation
• Process second training example
• Update weights with back-propagation
• And so on...
• But: no feedback to previous history
• After processing a few training examples, update through the unfolded recurrent neural network
Visualizing Individual Cells

Cell sensitive to position in line:
The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action—the one Kutuzov and the general mass of the army demanded—namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all—carried on by vis inertiae—pressed forward into boats and into the ice-covered water and did not surrender.

Cell that turns on inside quotes:
"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Karpathy et al. (2015): "Visualizing and Understanding Recurrent Networks"
Visualizing Individual Cells

Cell that robustly activates inside if statements:

```c
static int _dequeue_signal(struct sigpending *pending, sigset_t *mask, siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!((current->notifier)(current->notifier_data))) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```c
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX defines the longest valid length. */
```
encoder-decoder models
Recurrent Neural Translation Model

- We predicted the words of a sentence

- Why not also predict their translations?
- Obviously madness

- Proposed by Google (Sutskever et al. 2014)
What is Missing?

- Alignment of input words to output words

⇒ Solution: attention mechanism
neural translation model
with attention
• Inspiration: recurrent neural network language model on the input side
Hidden Language Model States

- This gives us the hidden states

- These encode left context for each word

- Same process in reverse: right context for each word
• Input encoder: concatenate bidirectional RNN states

• Each word representation includes full left and right sentence context
• Input is sequence of words $x_j$, mapped into embedding space $\tilde{E}x_j$

• Bidirectional recurrent neural networks

\[
\begin{align*}
\tilde{h}_j &= f(\tilde{h}_{j+1}, \tilde{E}x_j) \\
\hat{h}_j &= f(\hat{h}_{j-1}, \tilde{E}x_j)
\end{align*}
\]

• Various choices for the function $f()$: feed-forward layer, GRU, LSTM, ...
We want to have a recurrent neural network predicting output words.
• We want to have a recurrent neural network predicting output words

• We feed decisions on output words back into the decoder state
Decoder

- We want to have a recurrent neural network predicting output words

- We feed decisions on output words back into the decoder state
- Decoder state is also informed by the input context
• Given what we have generated so far (decoder hidden state)
• ... which words in the input should we pay attention to (encoder states)?
• Given: – the previous hidden state of the decoder $s_{i-1}$
  – the representation of input words $h_j = (\overrightarrow{h_j}, \overleftarrow{h_j})$

• Predict an alignment probability $a(s_{i-1}, h_j)$ to each input word $j$
  (modeled with with a feed-forward neural network layer)
**Attention**

- Normalize attention (softmax)

\[
\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}
\]
• Relevant input context: weigh input words according to attention: $c_i = \sum_j \alpha_{ij} h_j$
• Use context to predict next hidden state and output word
attention
• Machine translation is a structured prediction task
  – output is not a single label
  – output structure needs to be built, word by word

• Relevant information for each word prediction varies

• Human translators pay attention to different parts of the input sentence when translating

⇒ Attention mechanism
Computing Attention

- Attention mechanism in neural translation model (Bahdanau et al., 2015)
  - previous hidden state $s_{i-1}$
  - input word embedding $h_j$
  - trainable parameters $b, W_a, U_a, v_a$

\[
a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j + b)
\]

- Other ways to compute attention
  - Dot product: $a(s_{i-1}, h_j) = s_{i-1}^T h_j$
  - Scaled dot product: $a(s_{i-1}, h_j) = \frac{1}{\sqrt{|h_j|}} s_{i-1}^T h_j$
  - General: $a(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$
  - Local: $a(s_{i-1}) = W_a s_{i-1}$
Attention of Luong et al. (2015)

- Luong et al. (2015) demonstrate good results with the dot product
  \[ a(s_{i-1}, h_j) = s_{i-1}^T h_j \]

- No trainable parameters

- Additional changes

- Currently more popular
General View of Dot-Product Attention

- Three element
  - **Query**: decoder state
  - **Key**: encoder state
  - **Value**: encoder state

- Intuition
  - given a query (the decoder state)
  - we check how well it matches keys in the database (the encoder states)
  - and then use the matching score to scale the retrieved value (also the encoder state)

- Computation
  \[
  \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V
  \]
Scaled Dot-Product Attention

- Refinement of query, key, and value
- Scale it down to lower-dimensional vectors (e.g., 512 from 4096)
- Using a weight matrix for each: $Q W^Q, K W^K, V W^V$
Multi-Head Attention

• Add redundancy
  – say, 16 attention weights
  – each based on its own parameters $W$
  – matrix $W$ also reduces the dimensionality

• Formally:

$$\text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)$$

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

• Multi-head attention is a form of ensembling
Self Attention

• Finally, a very different take at attention

• Motivation so far: need for alignment between input words and output words

• Now: refine representation of input words in the encoder
  – representation of an input word mostly depends on itself
  – but also informed by the surrounding context
  – previously: recurrent neural networks (considers left or right context)
  – now: attention mechanism

• Self attention:
  Which of the surrounding words is most relevant to refine representation?
Self Attention

• Formal definition (based on sequence of vectors $h_j$, packed into matrix $H$

$$\text{self-attention}(H) = \text{Attention}(HW_i^Q, HW_i^K, HW_i^V)$$

• Association between every word representation $h_j$ any other context word $h_k$

• Resulting vector of normalized association values used to weigh context words
transformer
Self Attention: Transformer

- Self-attention in encoder
  - refine word representation based on relevant context words
  - relevance determined by self attention

- Self-attention in decoder
  - refine output word predictions based on relevant previous output words
  - relevance determined by self attention

- Also regular attention to encoder states in decoder

- Currently most successful model
  (maybe only with self attention in decoder, but regular recurrent decoder)
Self Attention Layer

- Given: input word representations $h_j$, packed into a matrix $H = (h_1, ..., h_j)$

- Self attention
  \[
  \text{self-attention}(H) = \text{MultiHead}(H, H, H)
  \]

- Shortcut connection
  \[
  \text{self-attention}(h_j) + h_j
  \]

- Layer normalization
  \[
  \hat{h}_j = \text{layer-normalization}(\text{self-attention}(h_j) + h_j)
  \]

- Feed-forward step with ReLU activation function
  \[
  \text{relu}(W\hat{h}_j + b)
  \]

- Again, shortcut connection and layer normalization
  \[
  \text{layer-normalization}(\text{relu}(W\hat{h}_j + b) + \hat{h}_j)
  \]
Stacked Self Attention Layers

- Stack several such layers (say, $D = 6$)

- Start with input word embedding
  \[ h_{0,j} = E x_j \]

- Stacked layers
  \[ h_{d,j} = \text{self-attention-layer}(h_{d-1,j}) \]
Decoder computes attention-based representations of the output in several layers, initialized with the embeddings of the previous output words.
Self-Attention in the Decoder

- Same idea as in the encoder

- Output words are initially encoded by word embeddings $s_i = Ey_i$.

- Self attention is computed over previous output words
  - association of a word $s_i$ is limited to words $s_k$ ($k \leq i$)
  - resulting representation $\tilde{s}_i$

$$\text{self-attention}(\tilde{S}) = \text{MultiHead}(\tilde{S}, \tilde{S}, \tilde{S})$$
Attention in the Decoder

• Original intuition of attention mechanism: focus on relevant input words

• Compute attention between the decoder states \( \tilde{S} \) and the final encoder states \( H \)

\[
\text{attention}(\tilde{S}, H) = \text{MultiHead}(\tilde{S}, H, H)
\]

• Note: attention mechanism formally mirrors self-attention
Full Decoder

- Self-attention
  \[
  \text{self-attention}(\tilde{S}) = \text{MultiHead}(\tilde{S}, \tilde{S}, \tilde{S})
  \]
  - shortcut connections
  - layer normalization
  - feed-forward layer

- Attention
  \[
  \text{attention}(\tilde{S}, H) = \text{softmaxMultiHead}(\tilde{S}, H, H)
  \]
  - shortcut connections
  - layer normalization
  - feed-forward layer

- Multiple stacked layers
Multi-Head Attention

It is in this spirit that a majority of American governments have passed new laws since 2009, making the registration or voting process more difficult.
Multi-Head Attention

“Many of the attention heads exhibit behaviour that seems related to the structure of the sentence.”
large language models
Masked Language Model Training

• Transformer expect an input and an output sequence

• Masked training
  – output sequence: one sentence of text
  – input sequence: same sentence, with some words masked out
    \textit{This [MASK] an [MASK] \rightarrow This is an example}

• Next sentence prediction
  – input sequence: one sentence
  – output sequence: next sentence in document

• Masked training with multiple shuffled sentences
  – same idea as masked training
  – multiple sentences (say, 3)
  – also reorder order of sentence in input sequence
Train language models on relatively clean text data (GPT-2)

- Convert any NLP problem into a text continuation problem
  - goes into some detail of how each task is converted
  - impressive performance on many tasks
Decoder-Only Models

- Alternative architecture: Just decoder of Transformer model

⇒ no input, only self-attention

- Trained with next-word prediction
Three Stages of Training Large Language Model

• Stage 1: Train on massive amounts of text (up to a trillion words)

• Stage 2: Instruction training
  – Examples of requests / responses constructed by human annotators
  – "Summarize the following: ..."
  – "Give me ten examples of ..."
  – "Translate from French into English: ..."

• Stage 3: Reinforcement learning from human feedback
  – Machine generates multiple responses to a prompt
  – Human annotators rank them
  – Train a reward model from
  – Fine-tune model with reward model
Instruction Training

- Examples of requests and responses constructed by human annotators
- May be collected from actual user requests and edited by experts
- May be generated from existing data sets

<table>
<thead>
<tr>
<th>Question Answering</th>
<th>Summarization</th>
<th>Translation</th>
</tr>
</thead>
</table>
| What is the highest mountain in the world? | *Summarize the following paragraph into one sentence.*
The Federal Reserve paused its campaign of interest rate increases for the first time in more than a year. But officials suggested that rates would rise more in 2023, as inflation remains “well above” the central bank’s target. Summary: No interest rate rise for now but maybe later in the year. | *Translate from English to German.*
English: My name is Ozymandias, King of Kings; Look on my Works, ye Mighty, and despair!
German: Mein Name ist Ozymandias, König der Könige; Schau auf meine Werke, du Mächtiger, und verzweifle! |
Reinforcement Learning from Human Feedback

- Machine generates multiple responses to a prompt
- Human annotators rank them
- Train a reward model
  - predict human annotation
  - can be applied to any text generated by model
  - normalized as reward function
- Fine-tune model with reward model
  - model generates response to prompt
  - reward function assesses response
  - if low reward, model needs to change
Prompt Engineering

• How a task is presented to the language model matters

• Black art: often unclear what is in the training data

• “Think step by step.”
  – language model has very limited working memory
  – complex reasoning may require several inference steps
  – by allowing the model to produce intermediate steps, working memory is created

• “Imagine you are...”
  – creates a style or sets a context for generation
In-Context Learning

• Provide examples of the task in the prompt

I want you to assess the sentiment of statements as either positive or negative.

Here are some examples:
*It’s a beautiful day.* Assessment: positive
*We did not play well in the game.* Assessment: negative
*That’s not bad.* Assessment: positive

Now assess the following statement:
*Yeah, right.* Assessment:
Retrieval Augmented Generation

- Question answering task
  - may be aided by specific documents
  - these documents may not even be part of LLM training

- Step 1: Retrieve relevant document

- Step 2: Add document to the prompt

- Step 3: Call LLM
Decomposition of Tasks

Write an outline for an argument that LLMs are great.

Outline:
1. LLMs have emerged as general problem solvers
2. LLMs have become publicly available
3. Many people find LLMs useful
4. ...

Write a paragraph about “LLMs have emerged as general problem solvers”.

Language model (LLMs) have rapidly emerged as versatile general problem solvers across various domains. Leveraging vast datasets and sophisticated algorithms, LLMs exhibit remarkable capabilities in understanding, generating,...
Tool Use by LLMs

- LLM generates call to APIs
- Executes call
- Receives result back
- Uses result to continue text generation

(Toolformer, Schick et al., 2023)
Programmatic Calls to LLMs

- If LLMs are universal problem solvers, use them for any task

```python
a = 1.6
b_text = "What is the distance to the moon in miles?"
c_text = LLM(b_text)
c = process_response(c_text)
print("Distance to moon in km:", a * c)
```
• Create training data ... with a language model

• Solve problem ... with a language model

• Evaluate results ... with a language model
Large Language Models for Mere Mortals

• Considering the size of language models
  – parameters are typically stored as 16-bit floats
  – during training also gradients and optimizer states need to be stored
    ⇒ 6 bytes per parameter
  – Also need to store the state of training examples
    (depends on sequence length and batch size)

• Size of GPUs
  – A100: 40-80GB RAM ($15,000)
  – RTX2080ti: 11GB RAM ($800)

• Only a few billion parameters models fit on single GPU
Adapting Compressed Models

QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers\textsuperscript{*}  
Artidoro Pagnoni\textsuperscript{*}  
Ari Holtzman

Luke Zettlemoyer  
University of Washington  
\{dettmers,artidoro,ahai,lsz\}@cs.washington.edu

- Low-rank adaptation parameter matrices
- Quantize all values into 4-bit floats
- Double quantization: also quantize the quantization factors
- Paged optimizers: NVIDIA unified memory that pages into CPU RAM
- Integrated into Huggingface, may adapt any model
Grand Challenges

- Hallucinations
- Interpretability
- HHH: Helpful, Honest, Harmless
- Preserving privacy of training data
questions?