Administrative

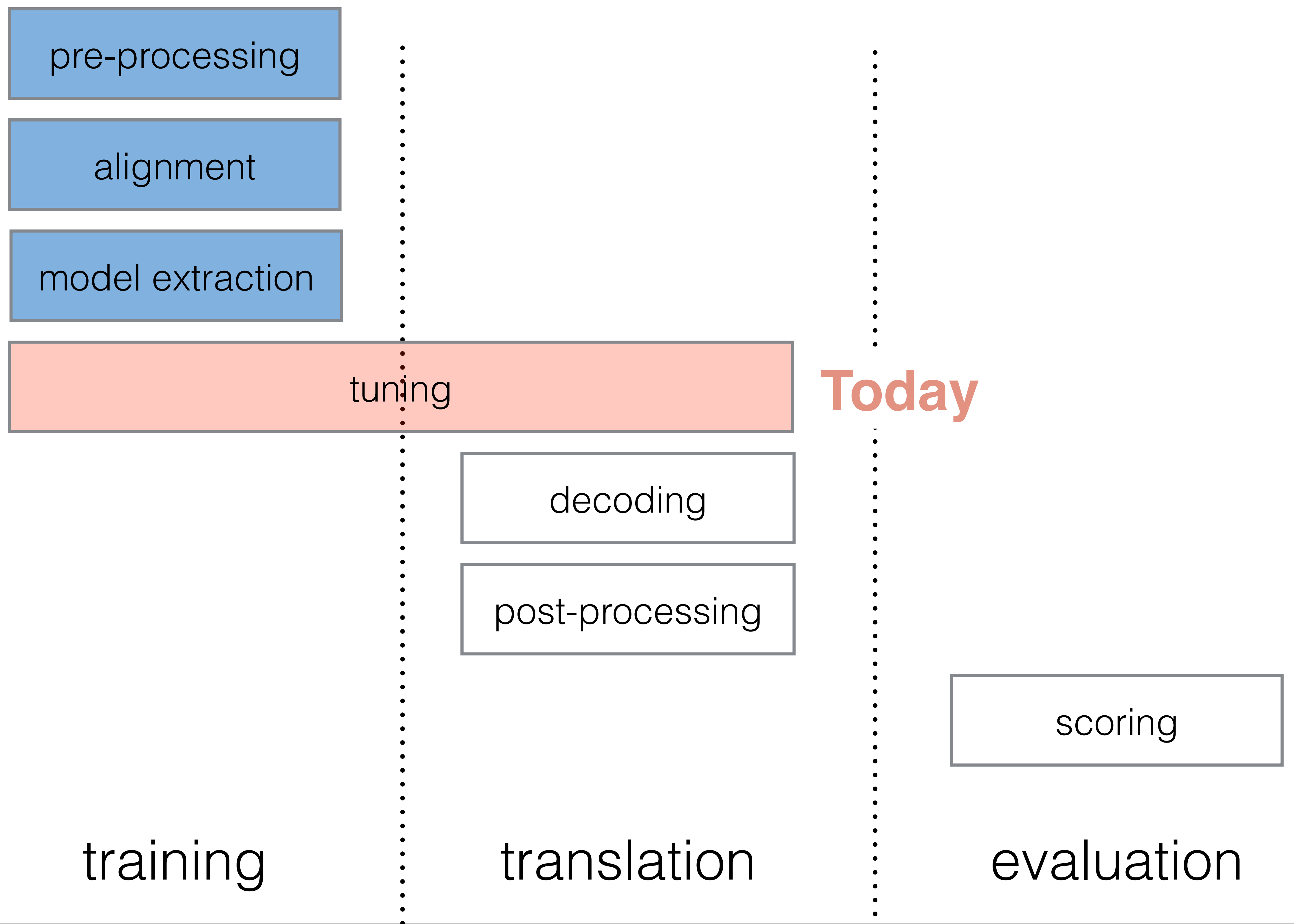
- Homework 3: due March 24 @ 6 PM, writeup next day in class

- Task: Write a program to determine which of a pair of translations is better, e.g.,

  A) This type of zápisníku was very ceněn writers and cestovateli.

  B) This type of notebook was very prized by writers and travellers.

Big Picture


pre-processing
alignment
model extraction
tuning
Today
decoding
post-processing
scoring
training
translation
evaluation

Weighted linear models

- We have defined the decoder search from this

$$(e^*, a^*) = \underset{e,a}{\textsf{argmax}}\ p(e)p(f, a \mid e)$$

- into an exponential model containing weighted component sub models

$$(e^*, a^*) = \underset{e,a}{\textsf{argmax}}\ \frac{\textsf{exp}\left\{\sum_i \lambda_i h_i(f, a, e)\right\}}{\sum_{e'} \textsf{exp}\left\{\sum_i \lambda_i h_i(f, a, e')\right\}}$$

- The *h*s are functions, λs are weights (parameters)

- For example, the following feature functions give us the standard noisy channel model

$$h_1 = p(f \mid e) \qquad \lambda_1 = 0.5$$
$$h_2 = p(e) \qquad \lambda_2 = 0.5$$

- The weights are important

- But why do we need this probabilistic interpretation?

- Instead, we can formulate decoding as a **weighted linear model**

$$
\begin{aligned}
(e^*, a^*) &= \underset{e,a}{\text{argmax}} \; \frac{\exp\left\{\sum_i \lambda_i h_i(f, a, e)\right\}}{\sum_{e'} \exp\left\{\sum_i \lambda_i h_i(f, a, e')\right\}} \\
&= \underset{e,a}{\text{argmax}} \; \exp\left\{\sum_i \lambda_i h_i(f, a, e)\right\} \\
&= \underset{e,a}{\text{argmax}} \; \sum_i \lambda_i h_i(f, a, e)
\end{aligned}
$$

Features

- Typical translation systems use 10 or so features
  - language model: log p(e)
  - phrasal translation probabilities: log p(f | e) and log p(e | f)
  - lexical translation probabilities
  - phrase count
  - word count
  - OOV penalty

- There are lots of possible features
- Constraint: features must factor over the search graph
  - Basic operation of decoding: is extending a hypothesis with a translation
  - Features can access the source sentence, the n-gram state (or other DP state), and the translated phrase pair

old hypothesis

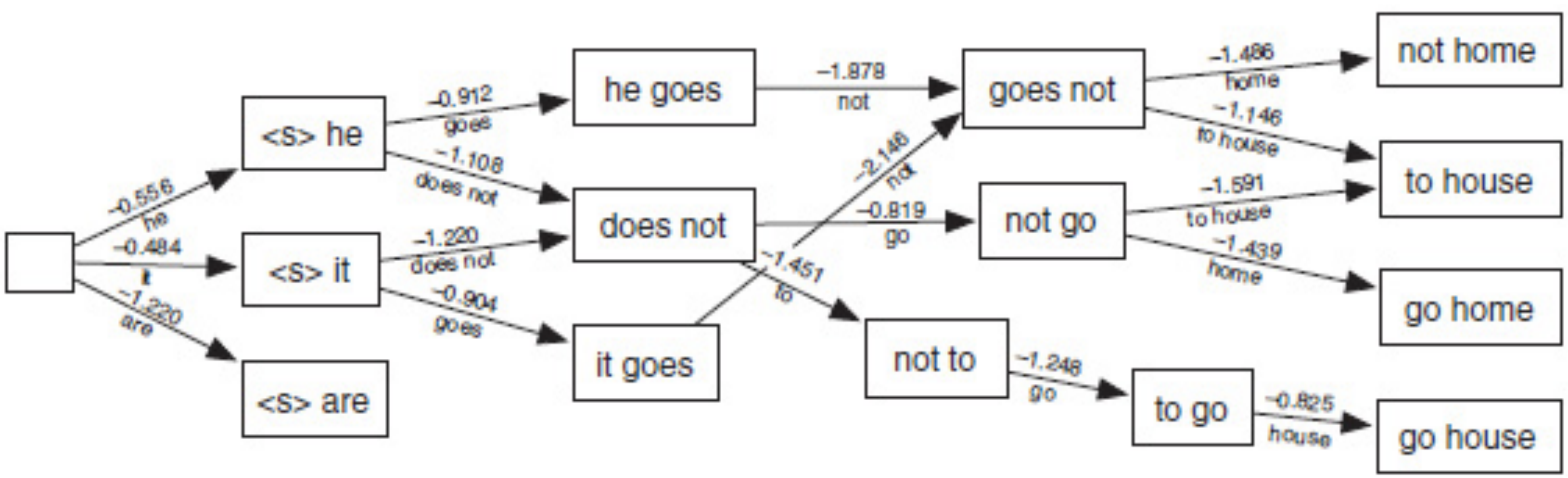
add word

new hypothesis

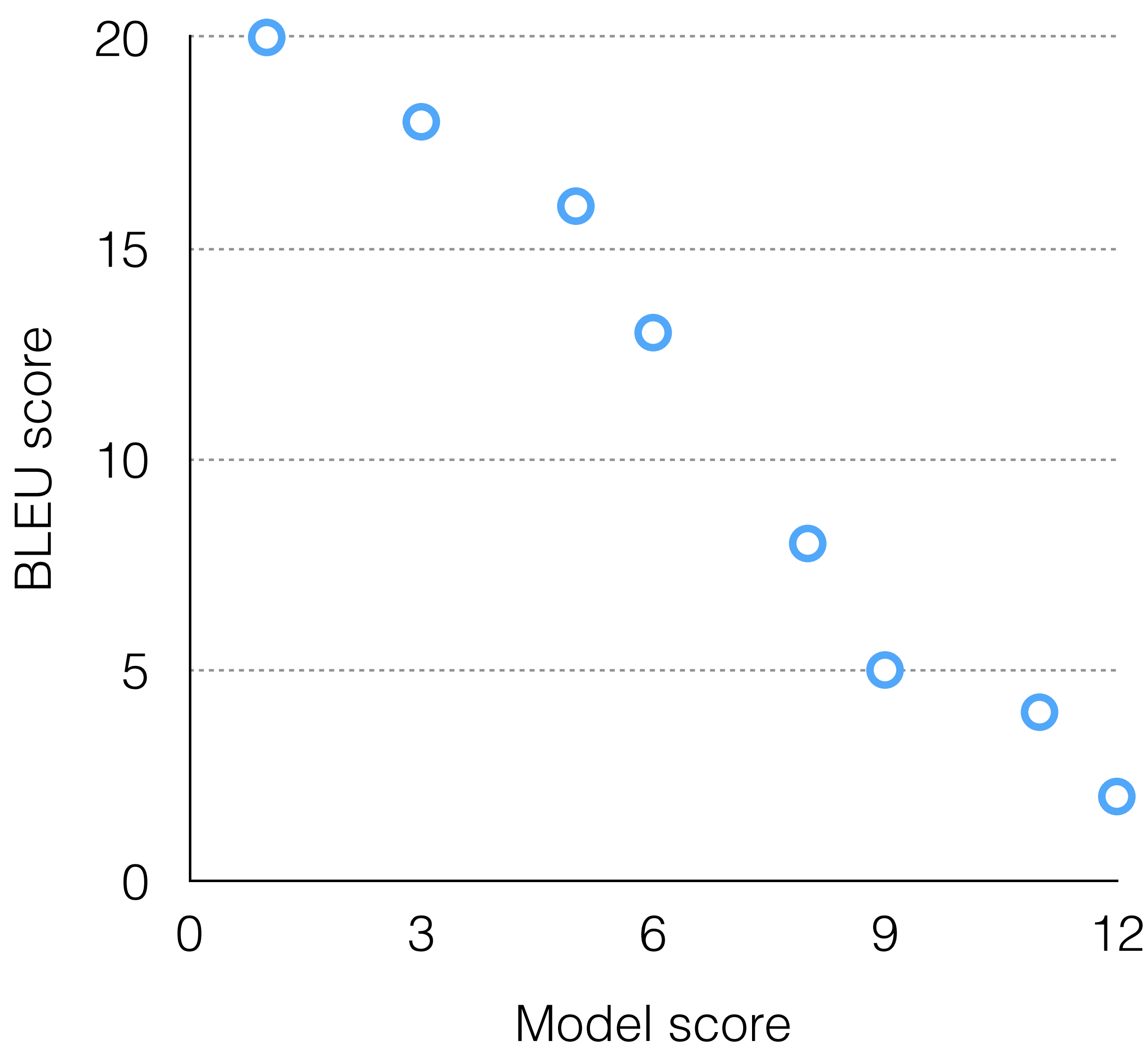
score += $P_{TM}(\text{am} \mid \text{tengo}) + P_{LM}(\text{am} \mid \text{I})$

Search space

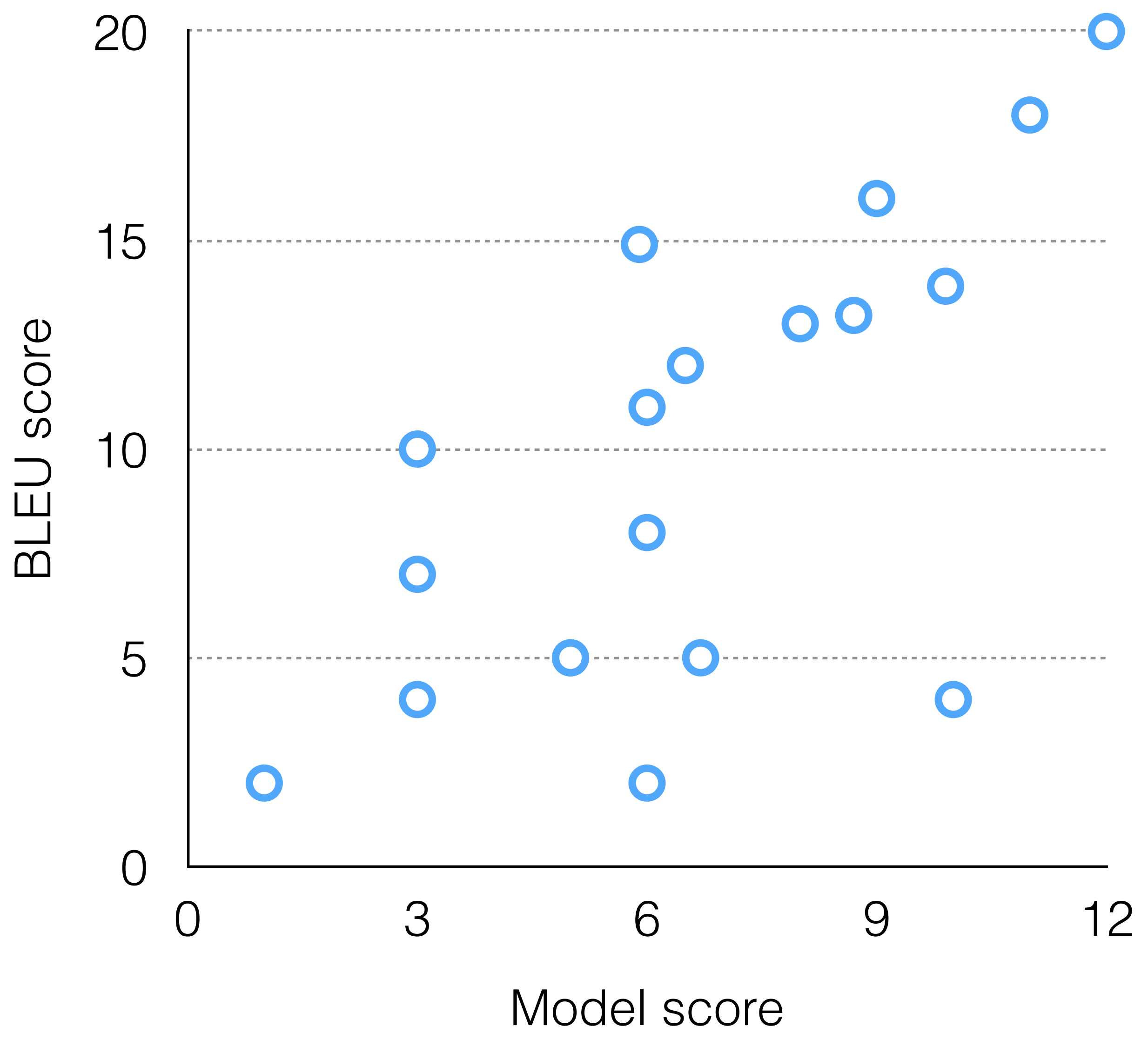
- Consider a standard search lattice representing the hypothesis space for the translation of a single sentence



- Enumerate the translations and plot their scores against a metric (like BLEU)
- This plot will take different shapes depending on the parameters (λs)
- This is a bad model



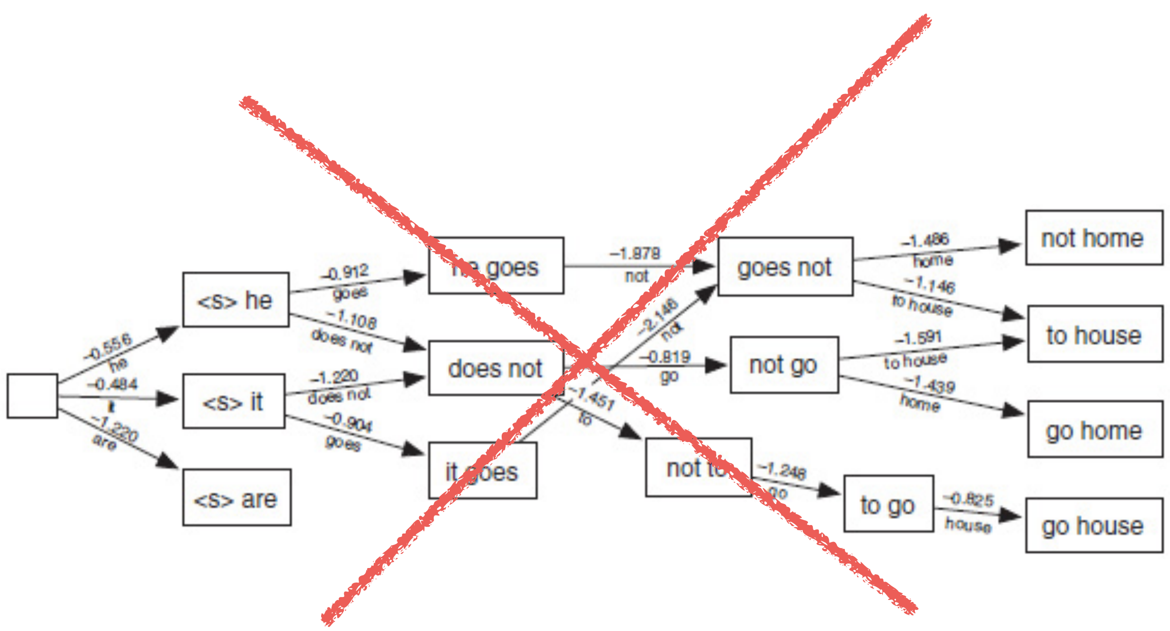
- This picture captures better what we'd like the model to do



- We need to learn how to set the parameters so that high model scores correlate with our metric (e.g., BLEU)
- However, lattices are complex to deal with

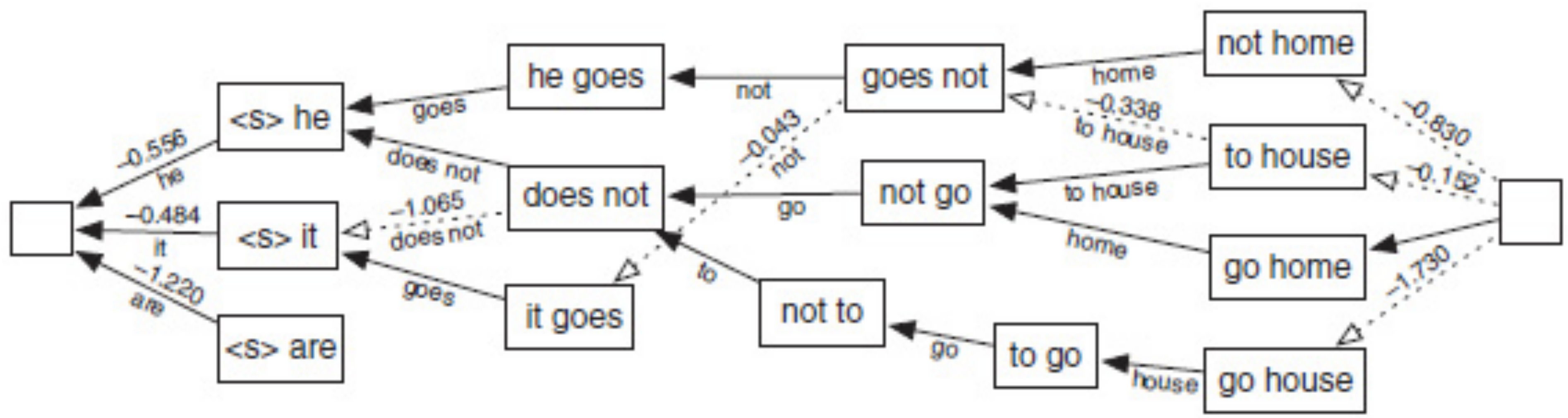
K-best extraction

- Instead of working directly with the *implicit* space represented by a lattice, we use an *explicit* list of hypotheses, e.g.,



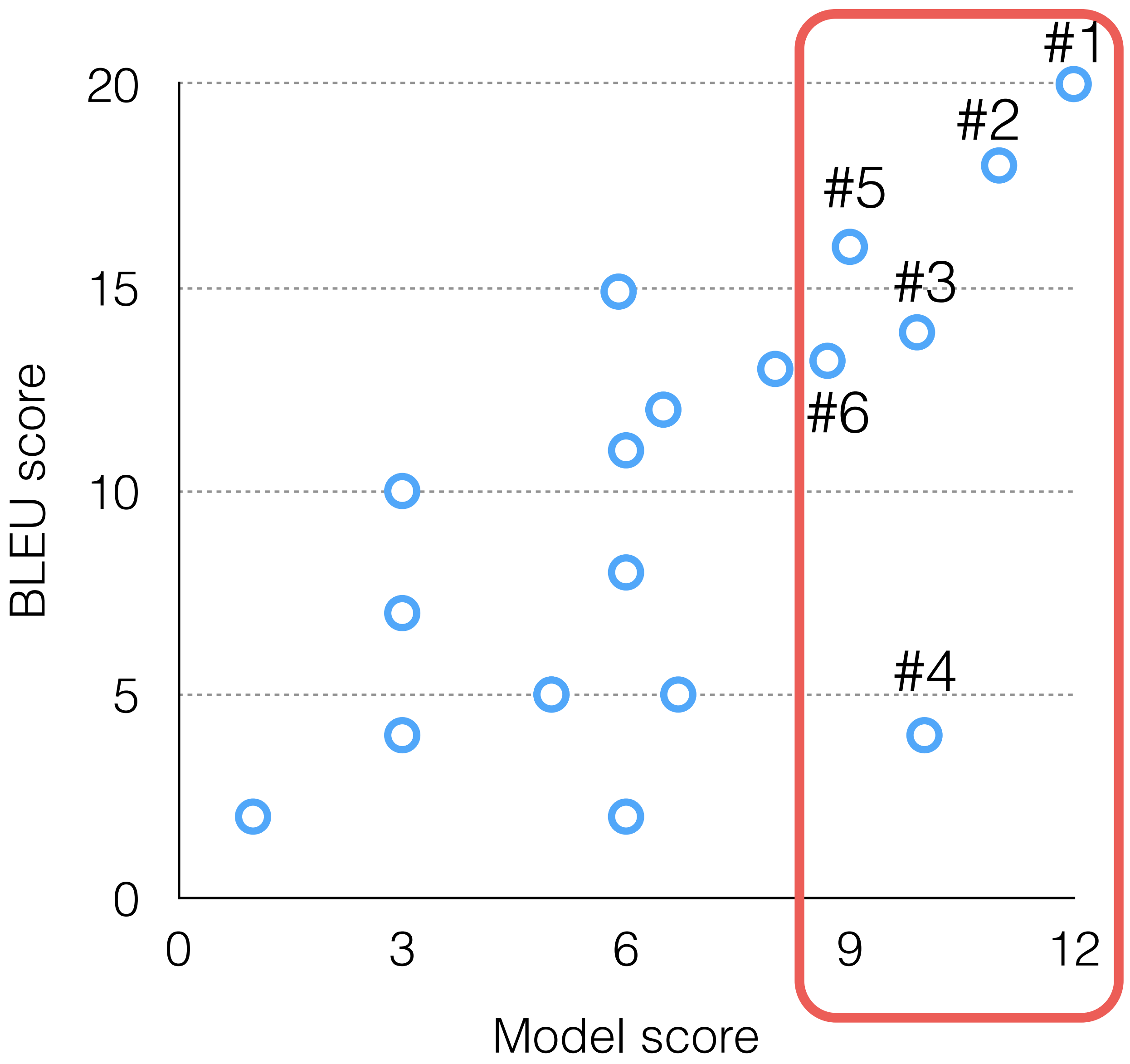
Rank	Score	Sentence
1	–4.182	he does not go home
2	–4.334	he does not go to house
3	–4.672	he goes not to house
4	–4.715	it goes not to house
5	–5.012	he goes not home
6	–5.055	it goes not home
7	–5.247	it does not go home
8	–5.399	it does not go to house
9	–5.912	he does not to go house
10	–6.977	it does not to go house

- How do we produce this?
- You already know how to extract the one-best translation. How do you get the second-best one?
- Note that every translation hypothesis corresponds to a single path through the lattice
- And recall that *recombined nodes* contain multiple tail pointers denoting every path to that node



- How do you get the second-best path?
  - Make a single sub-optimal choice
- How do you get the third-best one?
  - A suboptimal choice from the first-best or second-best path
- How do you get the nth-best one?

- After k-best extraction, we have an explicit representation of a portion of the search space



- Each of these has a sentence, a value for each feature h_i in the model, a total model score given by λ•h
- To this, we can add an *error* or *loss* term (the y-axis from before)

Translation	h_1	h_2	h_3	h_4	h_5	h_6	Error
	Feature values						
it is not under house	−32.22	−9.93	−19.00	−5.08	−8.22	−5	0.8
he is not under house	−34.50	−7.40	−16.33	−5.01	−8.15	−5	0.6
it is not a home	−28.49	−12.74	−19.29	−3.74	−8.42	−5	0.6
it is not to go home	−32.53	−10.34	−20.87	−4.38	−13.11	−6	0.8
it is not for house	−31.75	−17.25	−20.43	−4.90	−6.90	−5	0.8
he is not to go home	−35.79	−10.95	−18.20	−4.85	−13.04	−6	0.6
he does not home	−32.64	−11.84	−16.98	−3.67	−8.76	−4	**0.2**
it is not packing	−32.26	−10.63	−17.65	−5.08	−9.89	−4	0.8
he is not packing	−34.55	−8.10	−14.98	−5.01	−9.82	−4	0.6
he is not for home	−36.70	−13.52	−17.09	−6.22	−7.82	−5	0.4

- We will use these k-best lists to learn the best values of the model parameters (the λs)
- The fundamental insight is as follows:
  - Model score for each hypothesis is given as

$$\text{score}(e_k) = \lambda_1 \cdot h_1 + \lambda_2 \cdot h_2 + \cdots + \lambda_6 \cdot h_6$$

  - The chosen translation is the highest-scoring $\{e_k\}$
  - We can vary one λ_i at a time and change the highest-scoring hypothesis to one that has low loss

Minimum error-rate training

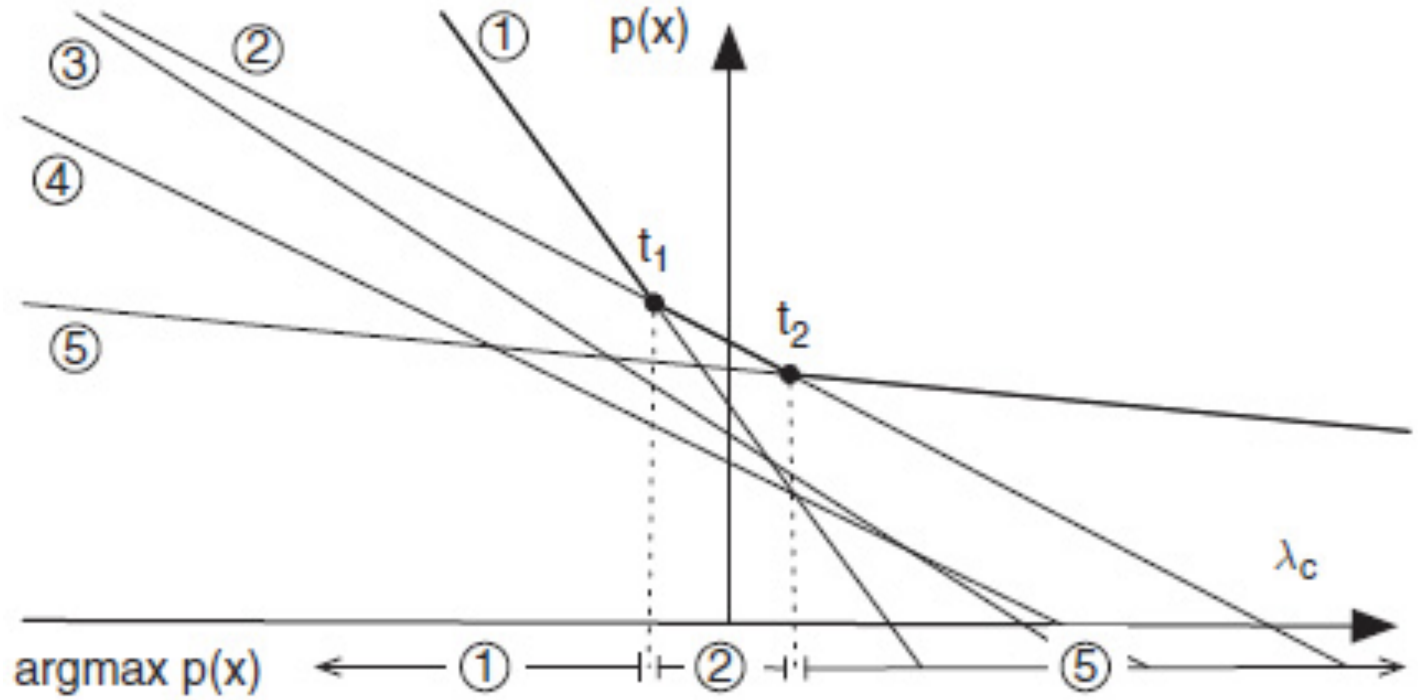
- Directly optimizes the model parameters to increase BLEU score
- Typically done on a *dev set*
  - A few thousand sentences
  - Should be representative of the test data
- Only scales to a few tens of parameters

(at least, before Galley et al. (EMNLP 2013))

- Input
    - k-best lists for all sentences in the dev set
    - loss function
- We modify one of the λ_is at a time
    - This gives us an equation of a single variable for the model score of each hypothesis

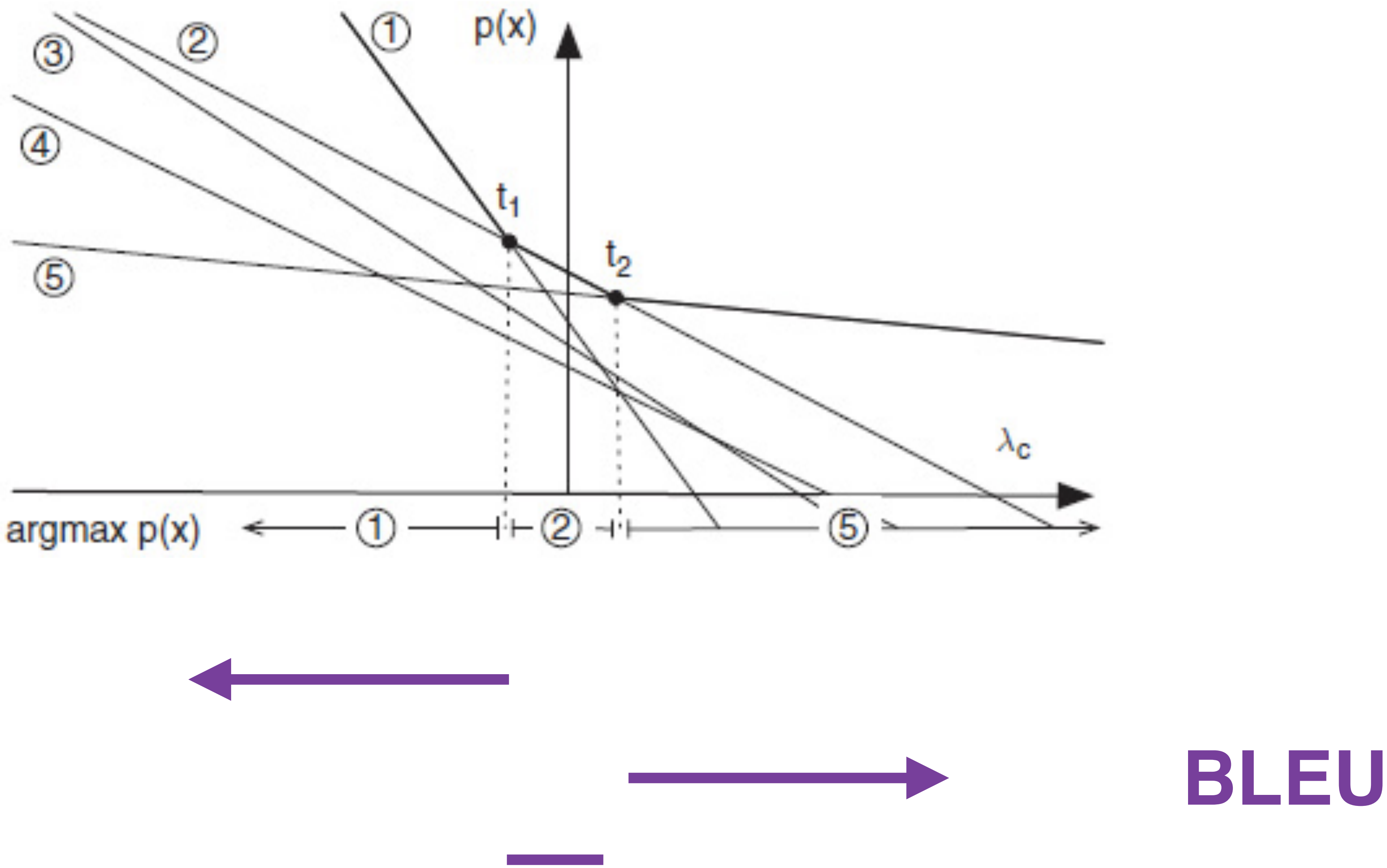
$$\begin{aligned} \text{score}(e_k) &= \lambda_1 \cdot h_1 + \boxed{\lambda_2 \cdot h_2 + \cdots + \lambda_6 \cdot h_6} \quad \mathbf{= b} \\ &= h_1 \cdot \lambda_1 + b \\ y &= mx + b \end{aligned}$$

- We can then plot all hypotheses for a sentence



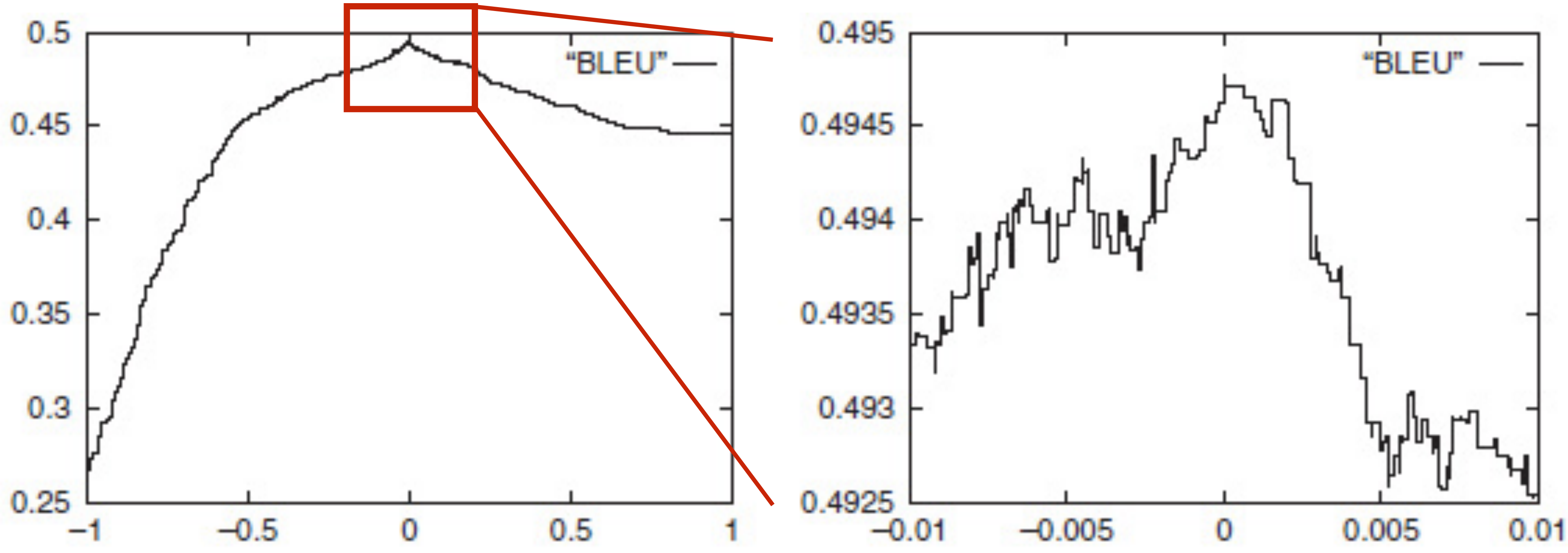
- Only lines in the *upper envelope* can ever be the best
- This can be efficiently computed (sort by slope)
- We accumulate a plot like this for every input sentence

- Important point: the BLEU score changes only at the plot's *threshold points* (values of λ where a new candidate reaches the top)



- We combine all the threshold points, and then sum BLEU along all the intervals, to find the best setting for the current λ

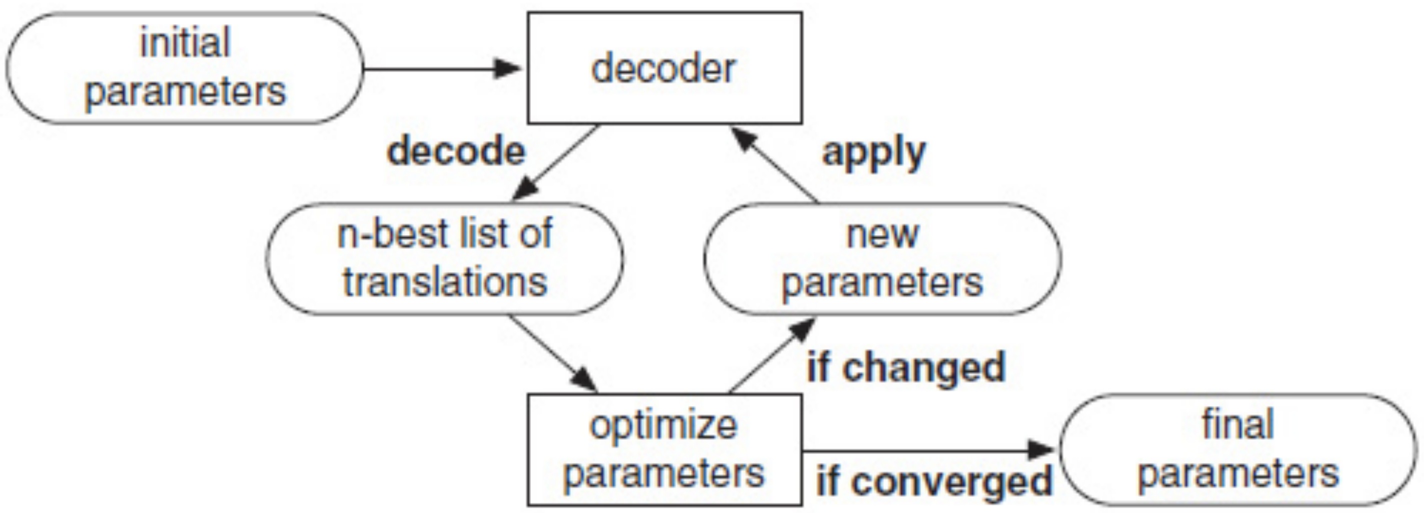
- Here is the BLEU error surface (Figure 9.10)



(detail of peak)

More details

- MERT actually iterates, building up k-best lists across iterations



- Quit when the λs don't change enough, or the k-best lists don't change

- How to initialize? Usually uniformly
- Iterating helps produce k-best lists that are more representative of the whole candidate space
- There are extensions of MERT to lattices, hypergraphs, and to thousands of parameters
- Open-source implementations: Moses mert, cmert, Zmert (Joshua)