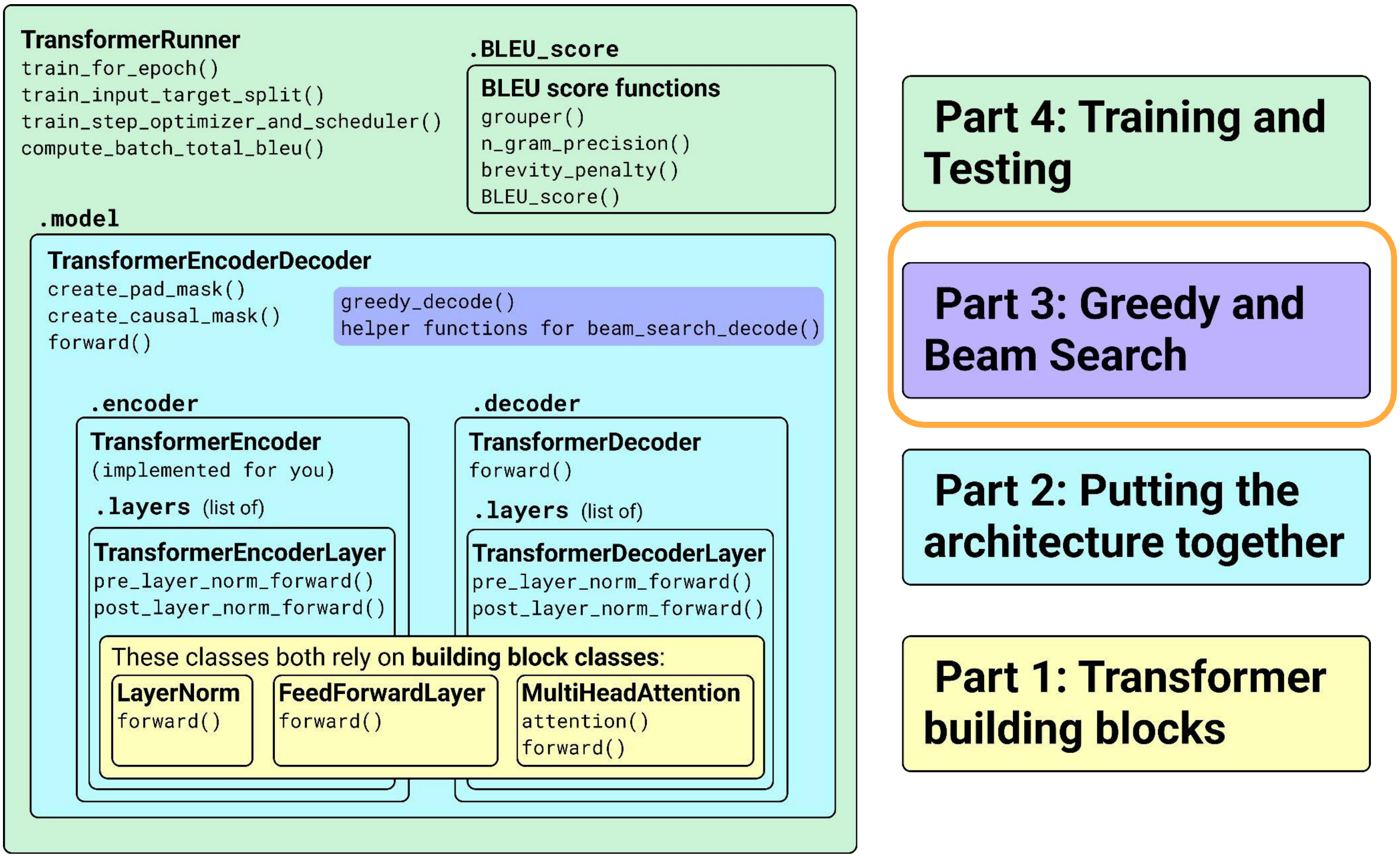
Neural Machine Translation using Transformers

CSC401/2511 A2 Tutorial 2

Winter 2024

Overview


TransformerRunner
train_for_epoch()
train_input_target_split()
train_step_optimizer_and_scheduler()
compute_batch_total_bleu()
.BLEU_score
BLEU score functions
grouper()
n_gram_precision()
brevity_penalty()
BLEU_score()
.model
TransformerEncoderDecoder
create_pad_mask()
create_causal_mask()
forward()
greedy_decode()
helper functions for beam_search_decode()
.encoder
TransformerEncoder
(implemented for you)
.layers (list of)
TransformerEncoderLayer
pre_layer_norm_forward()
post_layer_norm_forward()
.decoder
TransformerDecoder
forward()
.layers (list of)
TransformerDecoderLayer
pre_layer_norm_forward()
post_layer_norm_forward()
These classes both rely on building block classes:
LayerNorm
forward()
FeedForwardLayer
forward()
MultiHeadAttention
attention()
forward()
Part 4: Training and Testing
Part 3: Greedy and Beam Search
Part 2: Putting the architecture together
Part 1: Transformer building blocks

Decoding: what is it all about?

- At each time step *t*, our model computes a vector of scores for each token in our vocabulary for given all previous token $y_{<t}$, $S \in \mathbb{R}^V$:

$$S = f(\{y_{<t}\})$$

$f(\,.\,)$ is your model

- Then, we compute a probability distribution P over these scores with a softmax function:

$$P(y_t = w|\{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

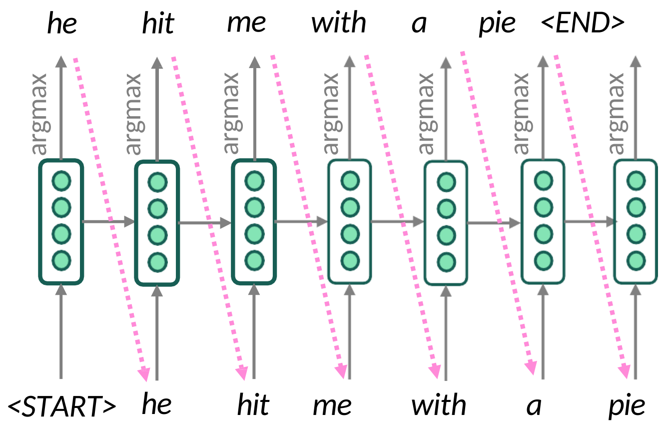
- Our decoding algorithm defines a function to select a token from this distribution:

$$\hat{y}_t = g(P(y_t|\{y_{<t}\}))$$

$g(\,.\,)$ is your decoding algorithm

Decoding: Greedy decoding

- Generate (or “decode”) the target sentence by taking argmax on each step of the decoder



- This is greedy decoding (take most probable word on each step)

Greedy Decode

TransformerEncoderDecoder.greedy_decode()

- Greedy approach to generating the translated sentence: Until each sentence in the batch has a finished translation, generate a new token.
- Methods to use: ***all_finished, torch.argmax(), concatenate_generation_sequence, pad_generation_sequence***

Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ____
  - → *he hit* ____
  - → *he hit a* ____ *(whoops! no going back now…)*

- How to fix this?

Exhaustive search decoding

- Ideally, we want to find a (length *T*) translation *y* that maximizes

$$\begin{aligned} P(y|x) &= P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x) \\ &= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x) \end{aligned}$$

- We could try computing all possible sequences *y*
  - This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
  - This $O(V^T)$ complexity is far too expensive!

Beam search decoding

- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10, in NMT)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam search decoding: example

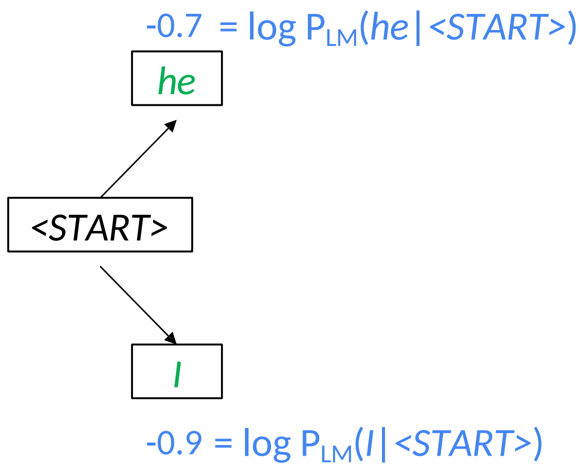
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob
dist of next word

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\mathrm{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\mathrm{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

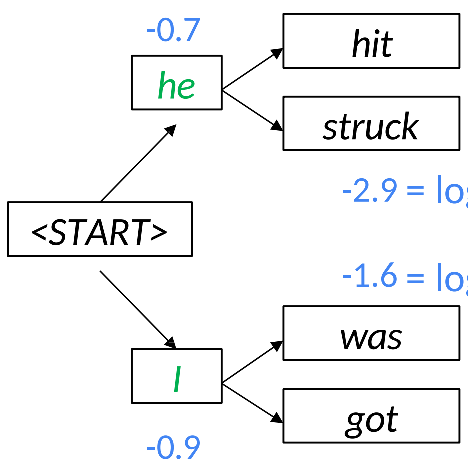


Take top *k* words and compute scores

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

-1.7 = log P_{LM}(*hit* | *<START> he*) + -0.7



-2.9 = log P_{LM}(*struck* | *<START> he*) + -0.7

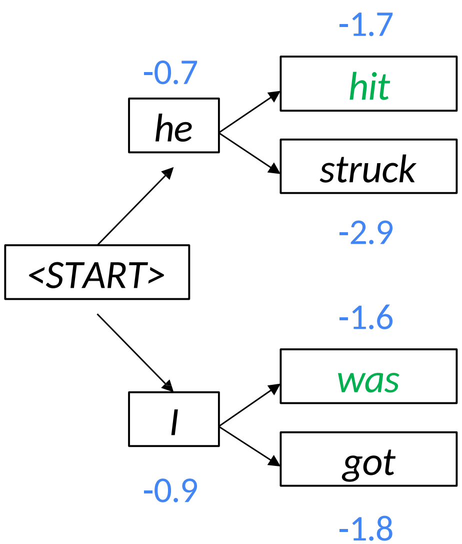
-1.6 = log P_{LM}(*was* | *<START> I*) + -0.9

-1.8 = log P_{LM}(*got* | *<START> I*) + -0.9

For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

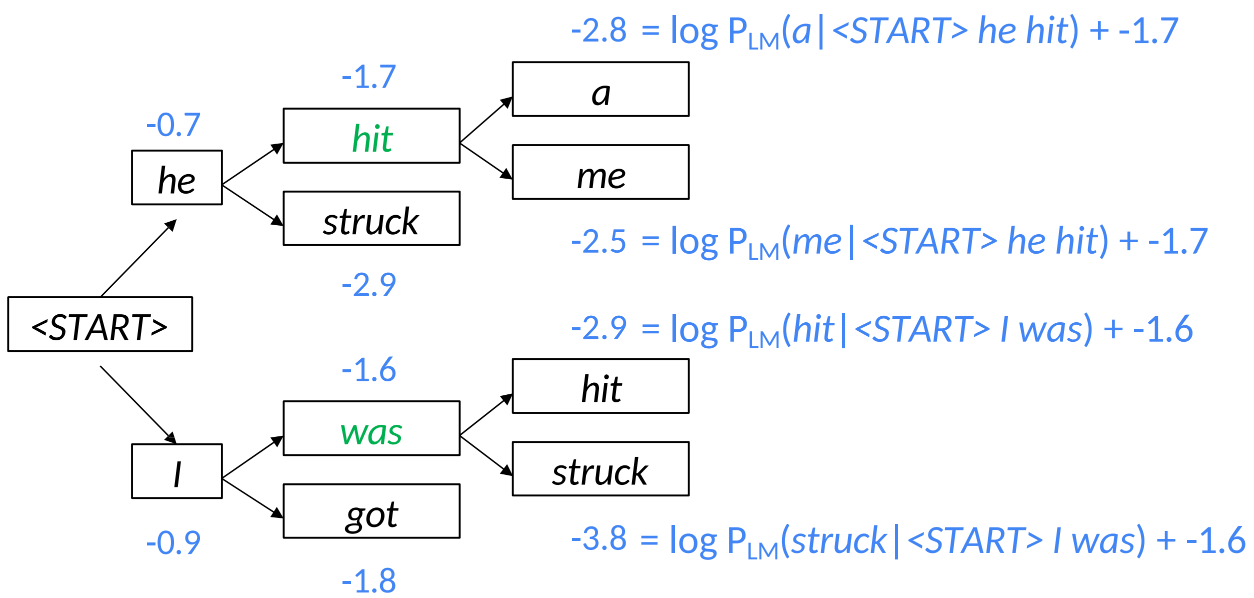
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

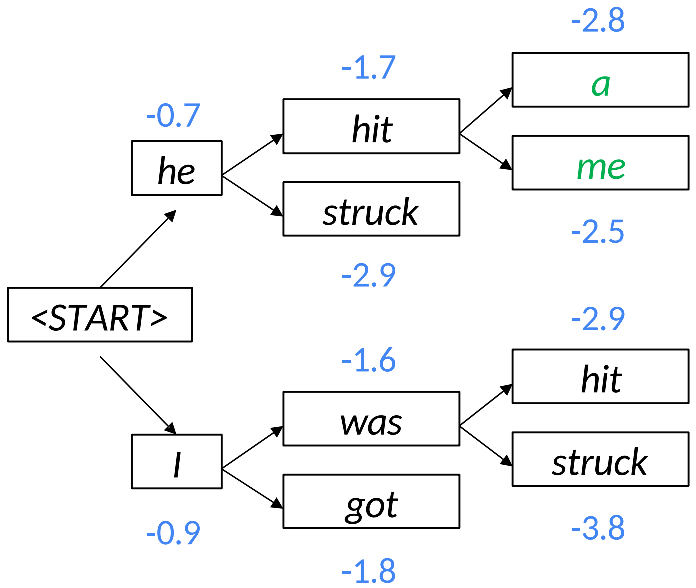
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

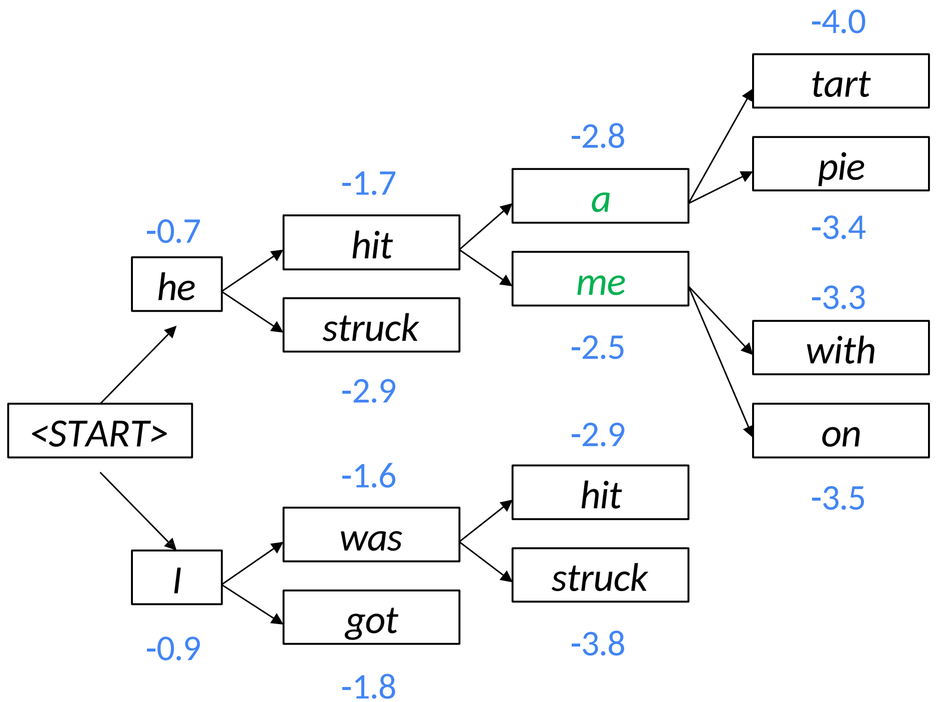
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

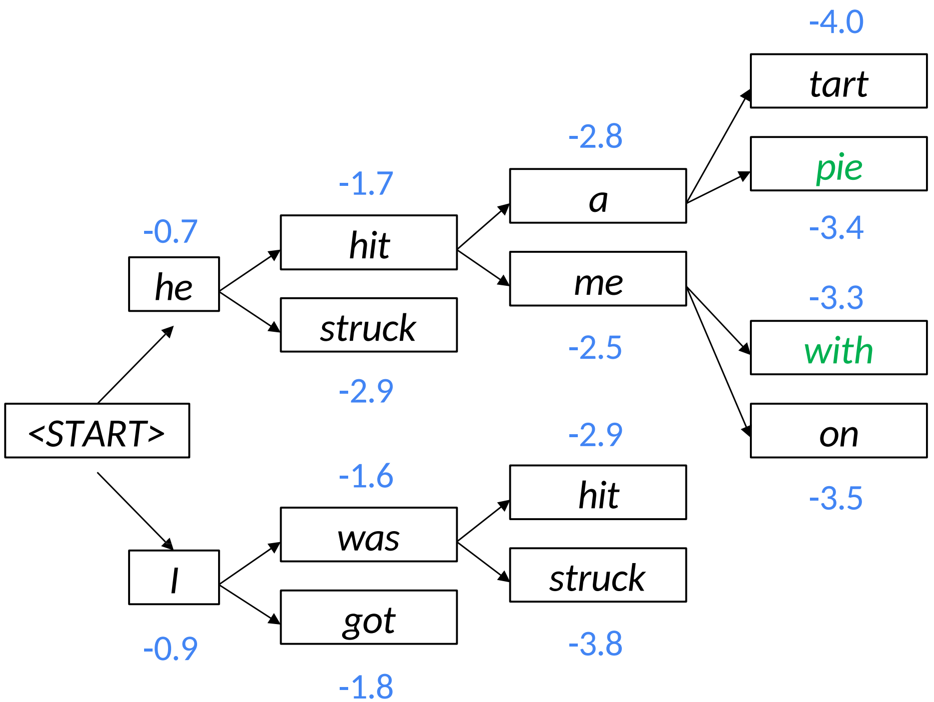
Beam size = k = 2. Blue numbers = $\text{score}(y_1,\ldots,y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

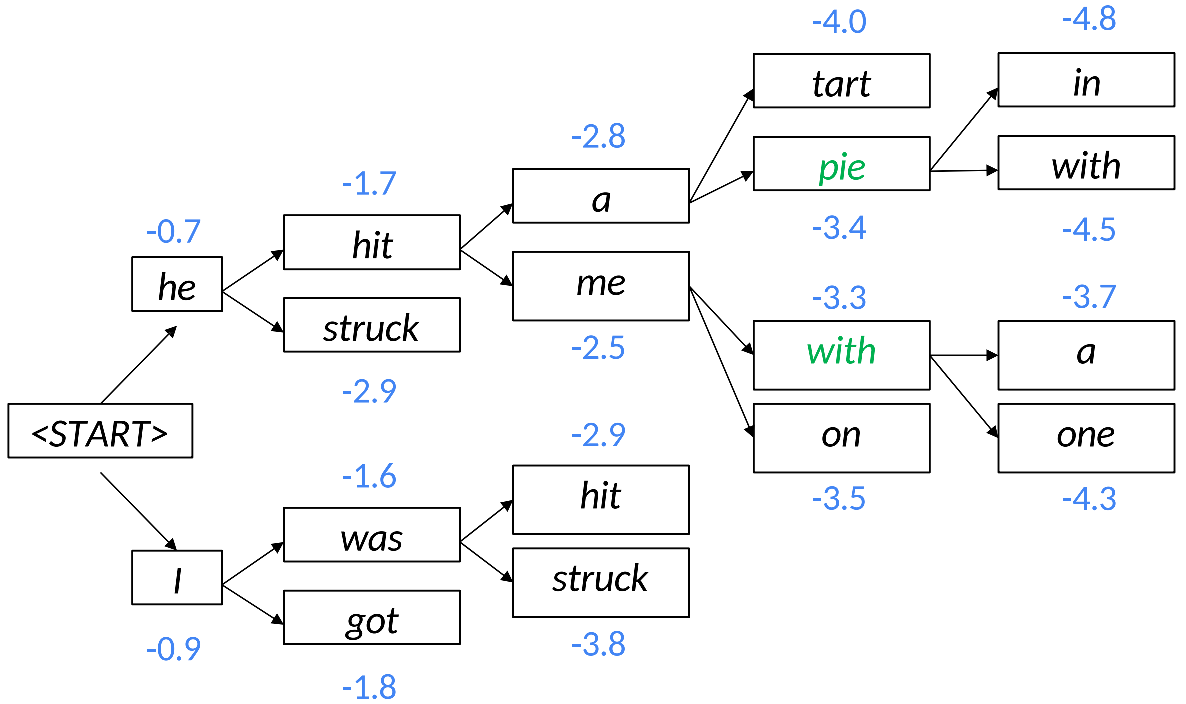
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

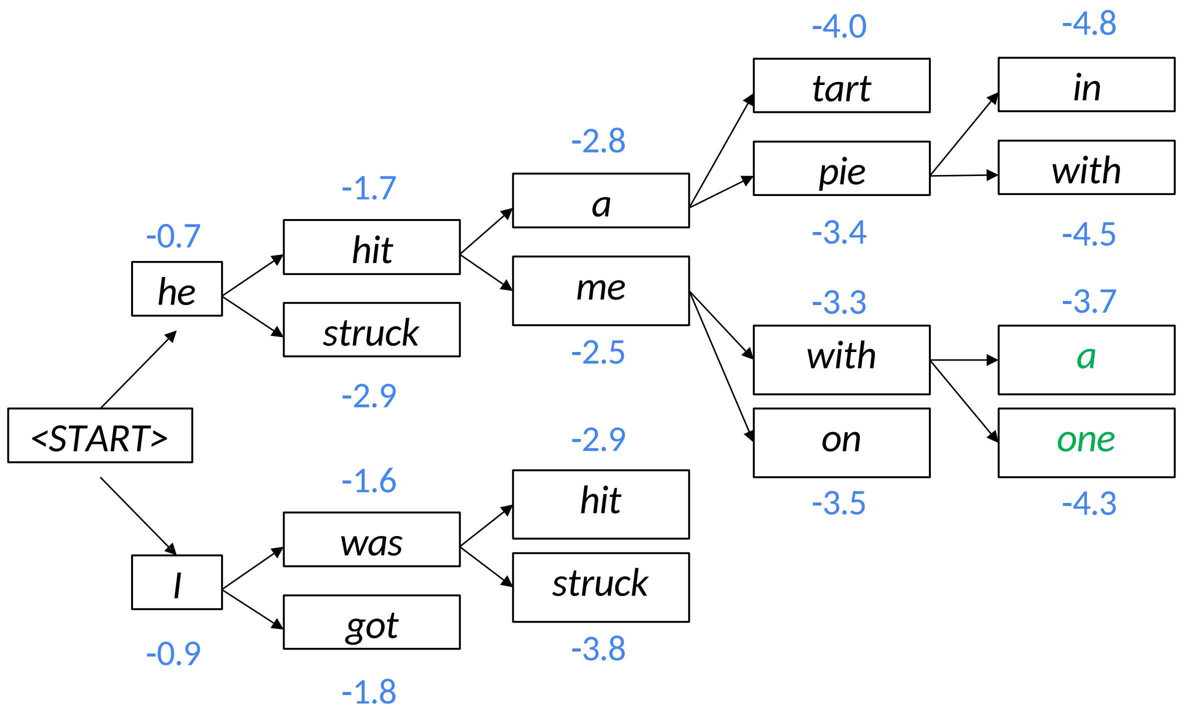
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

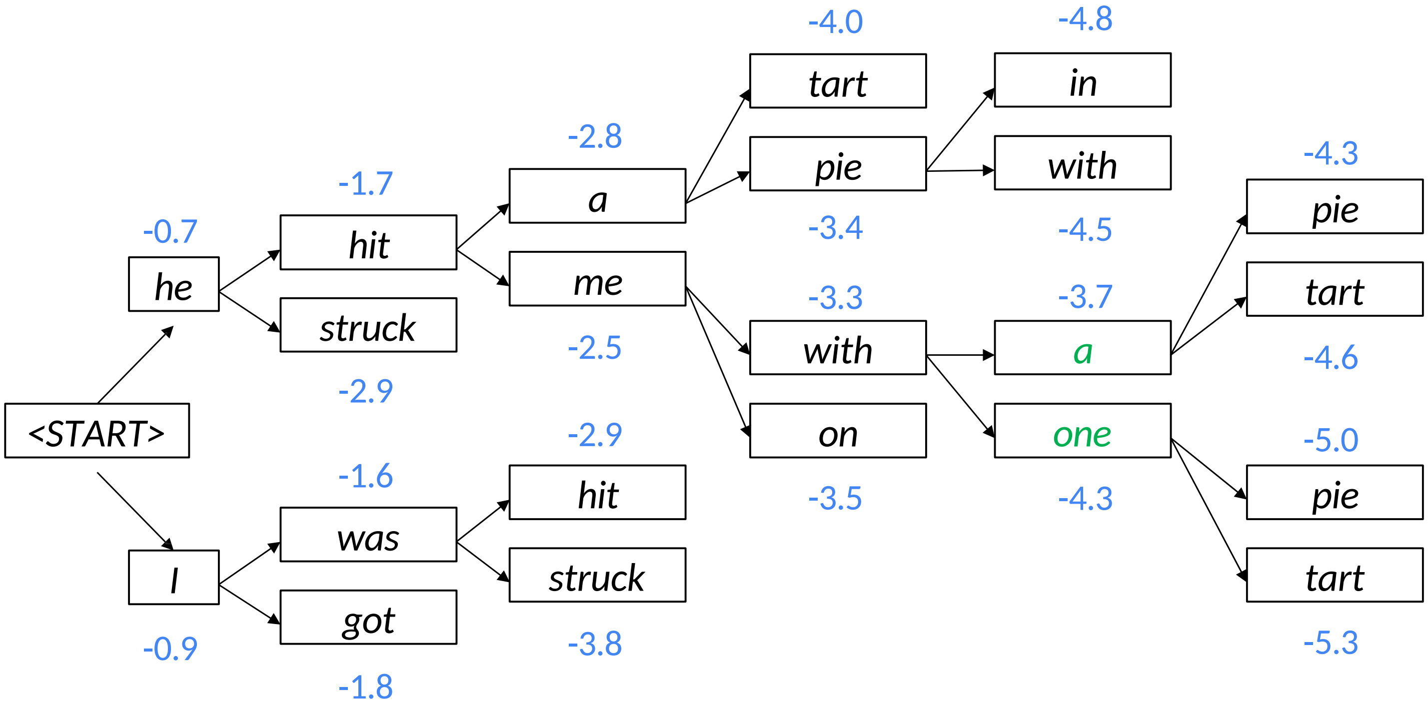
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

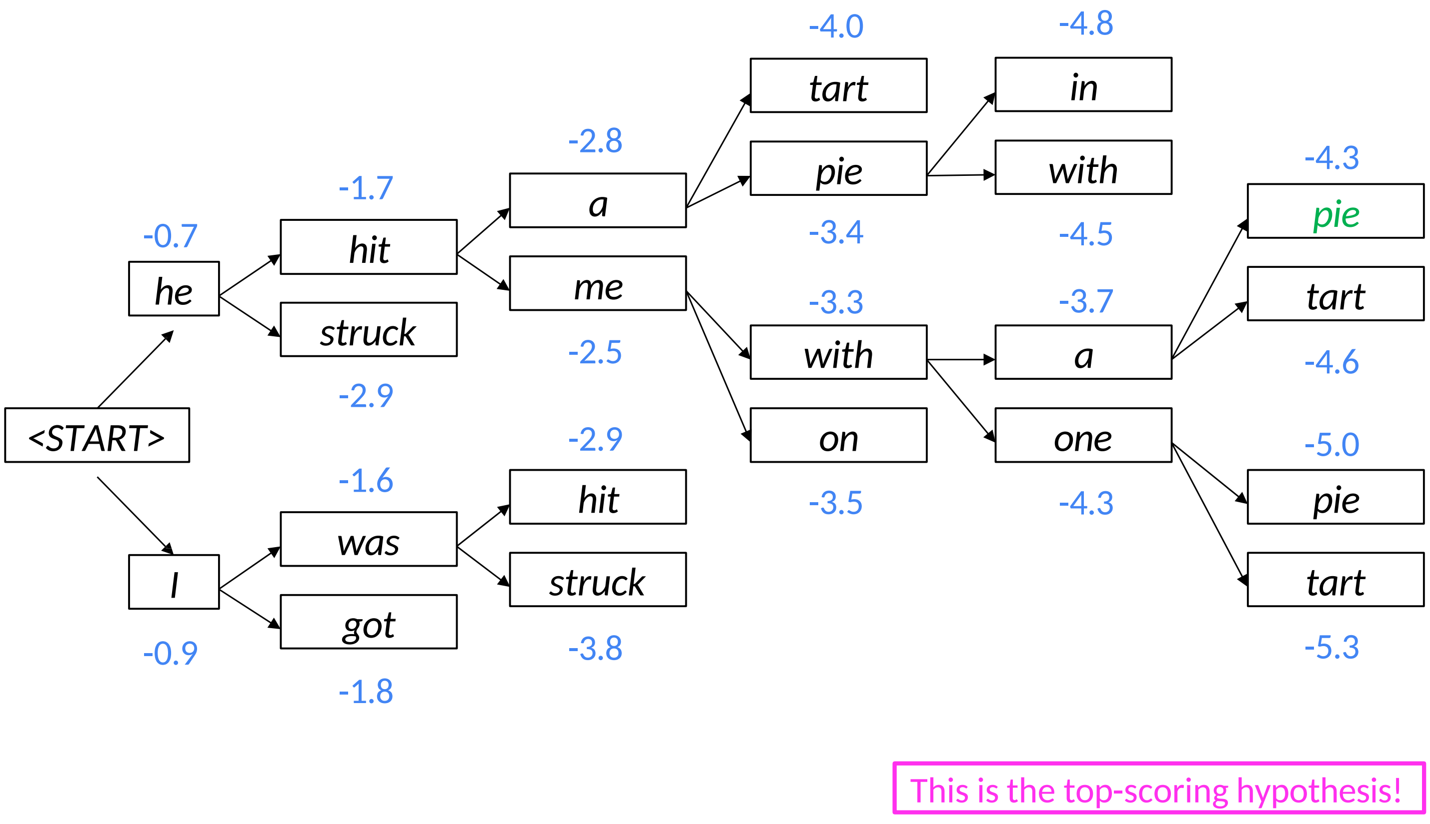
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

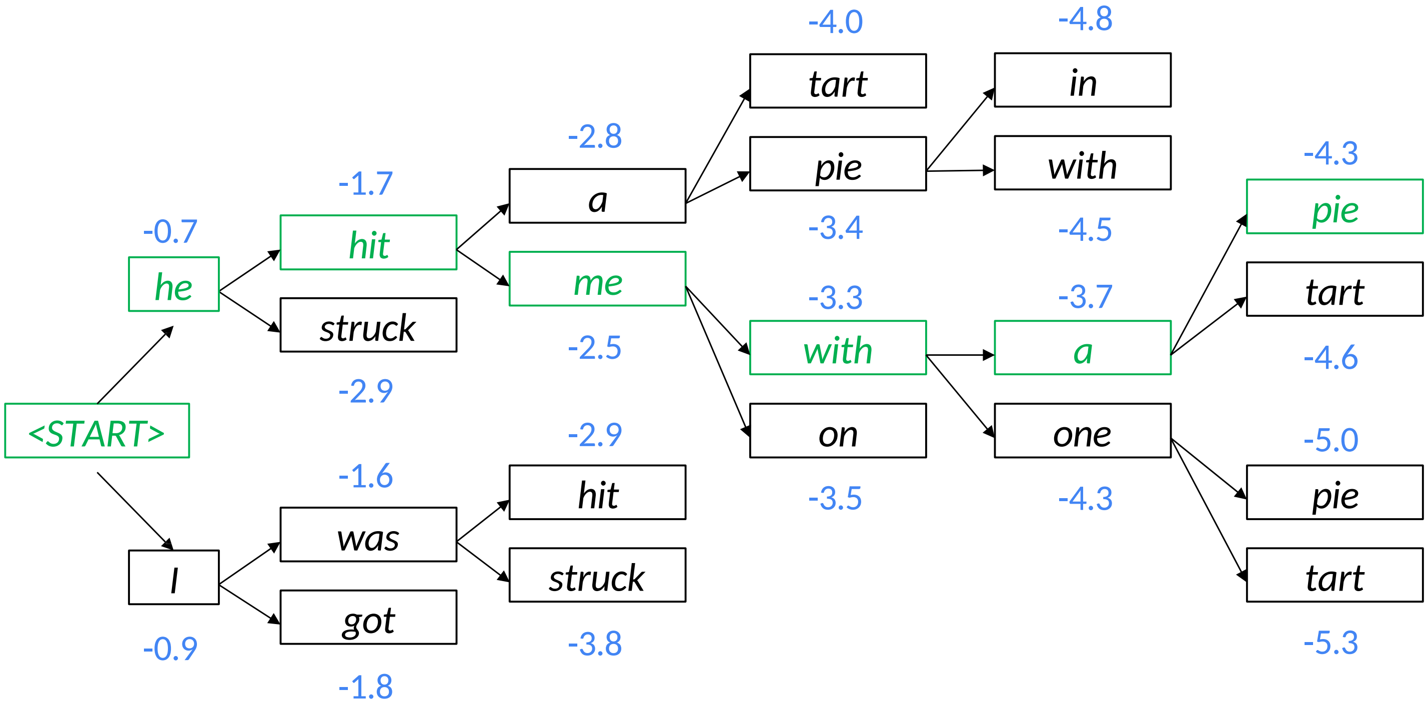
Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1,\ldots,y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$



Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces an <END> token
  - **For example:** *<START> he hit me with a pie <END>*

- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one?

- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

- **Problem with this:** longer hypotheses have lower scores

- **Fix:** Normalize by length. Use this to select top one instead:

$$\frac{1}{t}\sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

Summarize

- **Greedy Decoding**
  - Selects the highest probability token in $P(y_t | y_{<t})$

$$\hat{y}_t = \underset{w \in V}{\textbf{argmax}} \, P(y_t = w | y_{<t})$$

- **Beam Search**
  - Also aims to find strings that maximize the log-prob, but with wider exploration of candidates

Beam search helper methods

You need to complete 5 ***`TransformerEncoderDecoder`*** methods:

`initialize_beams_for_beam_search()`

- Takes first decoder step and uses the top-k outputs to initialize beams
- There are several steps listed in the docstring -- follow them carefully
- Tip: you need to call the encoder first (look at how this is done in `decode_greedy()`)

`expand_encoder_for_beam_search()`

- This is a helper method called at the end of the previous method.
- Goal: Expands source embeddings and mask to have shape [batch_size * k, ...] instead of [batch_size, ...]
- This gives the src embeddings (encoder output) a similar shape to the decoder beams, letting us process things in parallel
- Relevant pytorch method: `expand()`

Beam search helper methods

`repeat_and_reshape_for_beam_search()`

- We expand [batch_size * k, cur_len] -> [batch_size * k, expan, cur_len] so we can get n=expan completions for each of the current k translations per beam.
- We reshape [batch_size * k, expan, cur_len] -> [batch_size, k * expan, cur_len], so that (later) we can select the best k per sentence in the batch.
- Relevant pytorch method: `expand()`

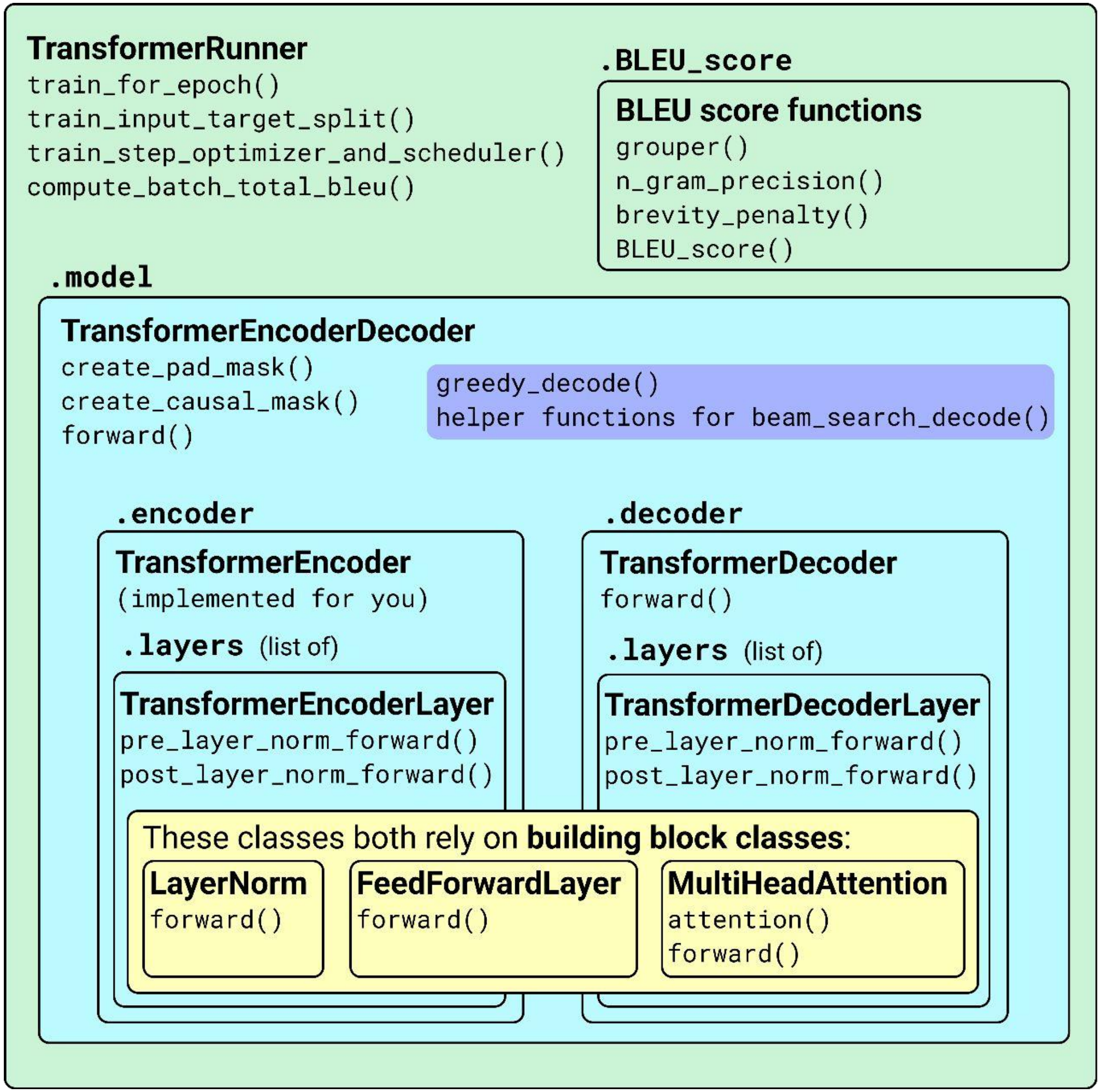
`score_sequence_for_beam_search()`

- You only need to do the second step (scoring) the sentences by summing log probabilities.

`finalize_beams_for_beam_search()`

- This pads the generated sequences so they are all the same length.
- We need to do this because beam search removes finished beams at each step (so the generated sequences can have different lengths)

Overview



Part 4: Training and Testing

Part 3: Greedy and Beam Search

Part 2: Putting the architecture together

Part 1: Transformer building blocks

BLEU evaluation

- **BLEU (BiLingual Evaluation Understudy)** is an automatic and popular method for evaluating MT.
  - It uses **multiple** human **reference** translations, and looks for local matches, allowing for phrase movement.

  - **Candidate:** *n.* a translation produced by a machine.

- There are a few parts to a **BLEU score**…

[1]Papineni, Kishore, et al. "Bleu: a method for automatic evaluation of machine translation." *Proceedings of the 40th ACL*. 2002. [link]

Example of BLEU evaluation

- **Reference 1**: *It is a guide to action that ensures that the military will forever heed Party commands*
- **Reference 2**: *It is the guiding principle which guarantees the military forces always being under command of the Party*
- **Reference 3**: *It is the practical guide for the army always to heed the directions of the party*

- **Candidate 1**: *It is a guide to action which ensures that the military always obeys the commands of the party*
- **Candidate 2**: *It is to insure the troops forever hearing the activity guidebook that party direct*

BLEU: Unigram precision

- The **unigram precision** of a candidate is

$$\frac{C}{N}$$

where N is the number of words in the **candidate**
and C is the number of words in the **candidate** which are in **at least one reference.**

- e.g., **Candidate 1**: *It is a guide to action which ensures that the military always obeys the commands of the party*
  - **Unigram precision =** $\frac{17}{18}$
    (*obeys* appears in none of the three references).

BLEU: Modified unigram precision

- **Reference 1**: *The lunatic is on the grass*
- **Reference 2**: *There is a lunatic upon the grass*
- **Candidate**: *The the the the the the the*
  - Unigram precision $= \frac{7}{7} = 1$ 

- **Capped unigram precision**:

  A candidate word type w can only be correct a **maximum** of $cap(\mathrm{w})$ times.
  - e.g., with $cap\ (the) = 2$, the above gives $p_1 = \frac{2}{7}$

BLEU: Generalizing to *N*-grams

- Generalizes to higher-order *N*-grams.
  - **Reference 1**: ***It is*** *a guide to action that ensures that the military will forever heed Party commands*
  - **Reference 2**: ***It is*** *the guiding principle which guarantees the military forces always being under command of the Party*
  - **Reference 3**: ***It is*** *the practical guide for the army always to heed the directions of the party*

  - **Candidate 1**: ***It is*** *a guide to action which ensures that the military always obeys the commands of the party*
  - **Candidate 2**: ***It is*** *to insure the troops forever hearing the activity guidebook that party direct*

Bigram precision, p_2

$p_2 = 10/17$

$p_2 = 1/13$

BLEU: Precision is not enough

- **Reference 1**: It is a guide to action that ensures that the *military will forever heed Party commands*
- **Reference 2**: It is the guiding principle which guarantees the *military forces always being under command* **of the** Party
- **Reference 3**: It is the practical guide for the army always to *heed the directions* **of the** party

- **Candidate 1**: **of the**

$$\text{Unigram precision, } p_1 = \frac{2}{2} 1 = \quad \text{Bigram precision, } p_2 = \frac{1}{1} 1 =$$

BLEU: Brevity

- Solution: Penalize brevity.
- **Step 1**: for each candidate,
  find the reference **most similar in length**.
- **Step 2**: c_i is the length of the i^{th} candidate, and
  r_i is the nearest length among the references,

$$brevity_i = \frac{r_i}{c_i}$$

Bigger = too brief

- **Step 3**: multiply precision by the **brevity penalty**:

$$BP_i = \begin{cases} 1 & \text{if } brevity_i < 1 \\ e^{1-brevity_i} & \text{if } brevity_i \geq 1 \end{cases}$$

$(r_i < c_i)$

$(r_i \geq c_i)$

BLEU: Final score

- On slide 87, $r_1 = 16, r_2 = 17, r_3 = 16$, and $c_1 = 18$ and $c_2 = 14$,

$$brevity_1 = \frac{17}{18} \qquad BP_1 = 1$$

$$brevity_2 = \frac{16}{14} \qquad BP_2 = e^{1-\left(\frac{8}{7}\right)} = 0.8669$$

- I

$$\boxed{BLEU_C = BP_C \times (p_1 p_2 \ldots p_n)^{1/n}}$$

where p_n is the n-gram precision. (You can set n empirically)

Example: Final BLEU score

- **Reference 1:** *I am afraid Dave*
  Reference 2: *I am scared Dave*
  Reference 3: *I have fear David*
  Candidate: *I fear David*

Assume $cap(\,) = 2$ for all N-grams

Also assume BLEU order $n = 2$

- $brevity = \frac{4}{3} \geq 1$ so $BP = e^{1-\left(\frac{4}{3}\right)}$
- $p_1 = \frac{1+1+1}{3} = 1$
- $p_2 = \frac{1}{2}$
- $BLEU = BP(p_1 p_2)^{\frac{1}{2}} = e^{1-\left(\frac{4}{3}\right)} \left(\frac{1}{2}\right)^{\frac{1}{2}} \approx 0.5067$

BLEU: summary

- BLEU is a **geometric mean** over n-gram precisions.
  - These precisions are **capped** to avoid strange cases.
    - E.g., the translation "*the the the the*" is not favoured.

  - This geometric mean is **weighted** (*brevity penalty*) so as not to favour unrealistically short translations, e.g., "*the*"

- Initially, evaluations showed that BLEU predicted human judgements very well, but:
  - People started **optimizing** MT systems to **maximize** BLEU. Correlations between BLEU and humans **decreased**.

When an evaluation metric becomes the target of optimization, it ceases to be an evaluation metric.

BLEU Score

`grouper()`

- Extract all n-grams from a sequence
- Use a sliding window approach to generate n-grams

`n_gram_precision()`

- Calculates the precision for a given order of n-gram
- First generate n-grams for both reference and candidate sequences
- Then count how many candidate n-grams in the reference n-grams and divide by the total

`brevity_penality()`

- Calculates the brevity penalty between a reference and candidate

`BLEU_score()`

- Compute the n-gram precisions for all orders from 1 to n
- Apply the formula

Training loop

train_for_epoch()

- Follow the instructions in the docstring
- Don’t forget to normalize loss!
- tqdm: easy progress bar

train_input_target_split()

- Split target tokens into input and target for maximum likelihood training (teacher forcing)
- model inputs exclude the last token in each sequence, and outputs exclude the first token in each sentence

train_step_optimizer_and_scheduler()

- Step the optimizer, zero out the gradient, and step scheduler

compute_batch_total_bleu()

- Computes bleu score for a batch of sentences
- tip: don’t pass sos, eos, and pad tokens to bleu_score_func

teach.cs with GPU: srun

- First make sure your code works in cpu mode! Debugging in CUDA mode is much more difficult

- Basic usage:

  `srun -p csc401 --gres gpu your_regular_command`

  - `srun -p csc2511 --gres gpu` if you enrolled in CSC 2511
- Check current queue: `squeue -p csc401`
- Keep training after disconnecting: Use `screen`

Analysis

Let’s translate some sentences!

Here, you translate 8 sentences from French to English, using the following models:

- The model you trained
- A fine-tuned pre-trained transformer model (T5 MT model or Bart MT model)
- A large, established model (Google Translate or ChatGPT)

Then, you answer four questions comparing them.

Q&A

Slides is from:
CSC401 Fall 2024 Lecture slides
CSC401 Fall 2024 Tutorial slides
Stanford CS 224N Winter 2023