Neural Machine Translation, Transformers, and Assignment 2

Arvid Frydenlund

October 11, 2024

Neural Machine Translation

Transformers

Assignment 2

Neural Machine Translation

- Task is to automatically translate a sentence in a **source** language to a sentence in a **target** language.
  - Terminology: 'source-side', 'src-side' 'target-side', 'tgt-side'.
- Methodology is to set up the task so that we can solve it via a neural network.
- Uses the **sequence-to-sequence** aka 'seq-2-seq' aka 'encoder-decoder' paradigm.
  - Let $X = x_1, x_2, \ldots, x_S$ be a src sequence of S tokens
  - Let $Y = y_1, y_2, \ldots, y_T$ be a tgt sequence of T tokens
  - Learn mapping of $X \rightarrow Y$ with a neural net.

Probabilistic Model

- We wish to model the joint probability of Y conditioned on X, i.e. $P(y_1, y_2, \ldots, y_T \,|\, y_0, X)$
- Use chain rule to model this as a per-token conditional distribution, $\prod_{t=1}^{T} P(y_t, \,|\, y_{<t}, X)$
  - Y_0 is a special **Start-of-Sentence** token (SOS).
  - We also need a special **End-of-Sentence** token (EOS) for inference/generation.
- This is a supervised learning problem where we are given (X, Y) pairs during training and Y acts as our targets.
  - However, there is no per-token alignment supervision.
- Train the model by minimizing the log probability. $L = -\sum_{t=1}^{T} \log P(y_t, \,|\, y_{<t}, X)$

Neural Model Skeleton (Encoder)

Encode X via an **encoder**

- First convert discrete tokens into **embeddings** via a look-up table.
- Since X is fully observed (provided as input), we can create a holistic representation of X by considering all tokens in the sequence.
- Let H^X be a set of S hidden states representing X.
  - These will be created by a transformer applied over X.
  - *Think BERT but without any masking.*
  - Each $h_1^X, \ldots, h_S^X$ is contextualized by all tokens in X
- Then our model becomes $\prod_{t=1}^{T} P(y_t, \mid y_{<t}, H^X)$

Neural Model Skeleton (Decoder)

Decode or generate the predicted sequence $\hat{Y}$ via a **decoder**.

- Where Y is the ground-truth sequence given during training and $\hat{Y}$ is the predicted sequence.
- The decoder implements the conditional distribution and is used to generate $\hat{Y}$ token-by-token **autoregressively**.
  - Let the model output a token $\hat{y}_t$ at step t given X and $\hat{y}_{<t}$
  - This then concatenates to form the input $\hat{y}_{<t+1}$ at step $t+1$.
  - Thus each step corresponds to one term in the chain rule decomposition.
- *This is not done during training*, but rather the ground-truth tokens, Y, are used as input instead of the model's predictions, $\hat{Y}$.
- This is called **teacher-forcing** aka **maximum likelihood training**.

Teacher-forcing

- The full ground-truth sequence Y is input during training.
- For a transformer model, the condition that tokens in Y can only condition on prior tokens is achieved via a **causal attention mask**.
- Note the training is defined as predicting the **next-token** given prior tokens.
  - This means we shift the inputted Y and the target Y by one token.
  - This is done via an **input target split**.
  - So inputs $= y_0, y_1, \ldots, y_{T-1}$ and targets $= y_1, y_2, \ldots, y_T$.
  - You can also think about this as inputs have SOS at the start and targets have EOS at the end.
  - Debugging tip: temporarily making the inputs and the targets the same values should allow your model get a near-zero loss.

Neural Model Skeleton (Encoder-Decoder)

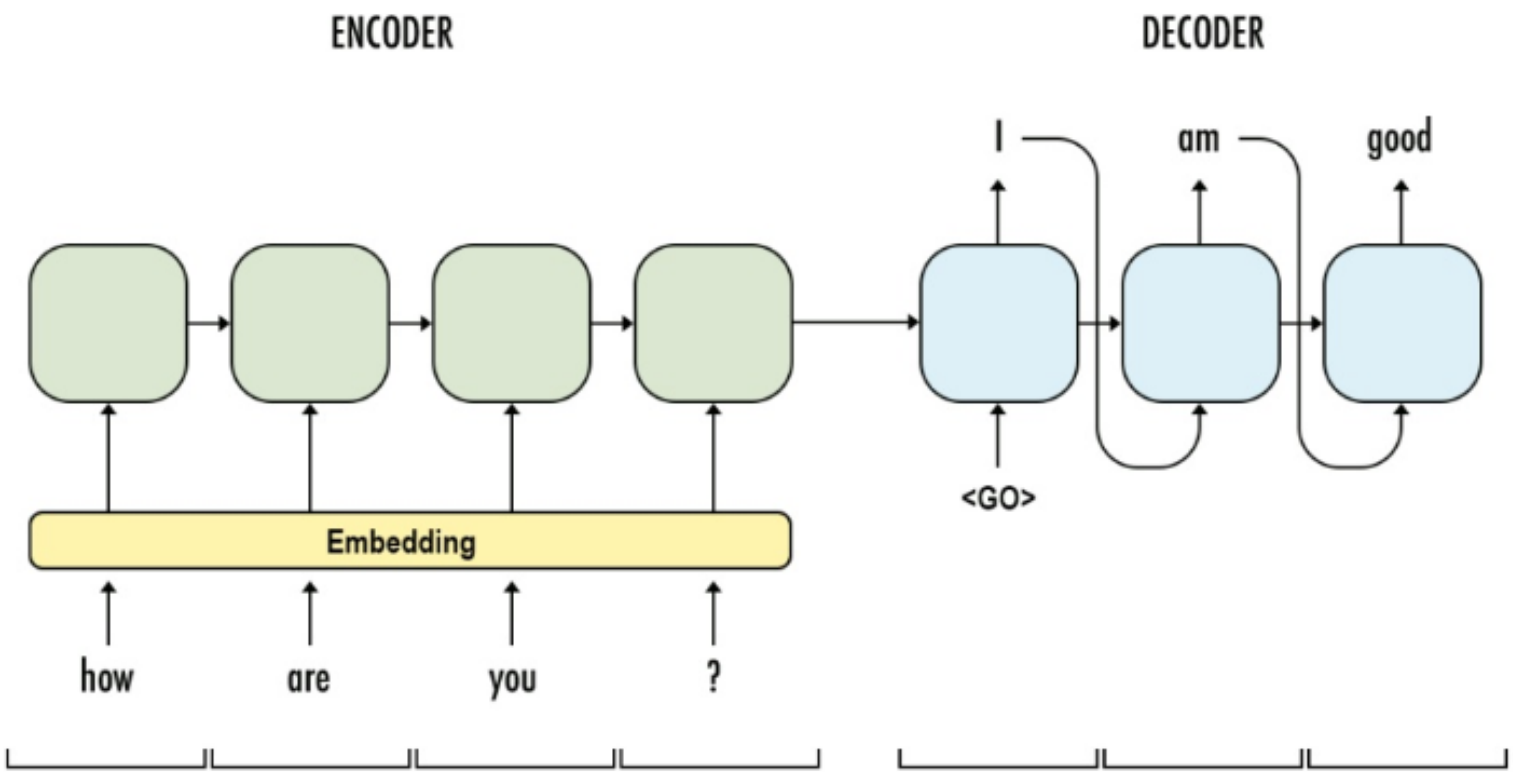

Figure: Autoregressive encoder-decoder model (fig. from here).

- Side note: language models are decoder-only models.
- *Neural* machine translation and other seq-2-seq models are just conditional language models.

Encoder-Decoder Transformer

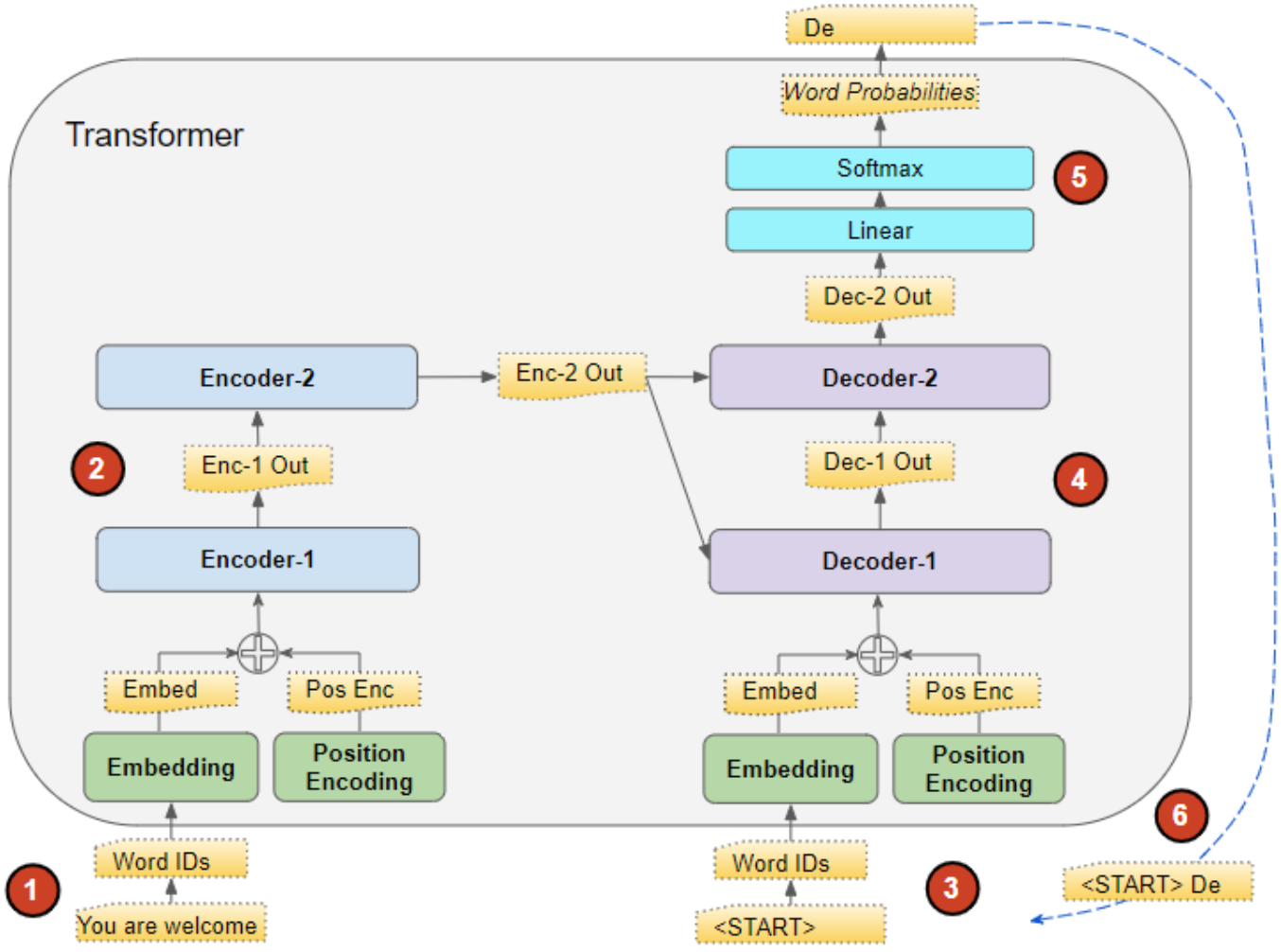

Figure: Autoregressive encoder-decoder model (fig. from here).

Transformers

The core building blocks:

- **Embedding layer**:
  - Converts token IDs into embeddings (int to vector)
- **Positional embedding layer**:
  - Adds a numerical ordinal bias to each embedding
  - This is required because transformers only consider pair-wise comparisons between tokens, and thus have no intrinsic understanding of sequence order.
  - Sequence order is, of course, critical for understanding language.

These are given to you and you only have to know when to apply them.

Transformers

The core building blocks:

- **Feed-forward layer**:
  - Actually two linear projections, two dropouts, and an activation function.
  - Idea is to project to a higher dimension, apply the activation function and dropout, project back down, and then apply another dropout.
  - This applies the same non-linear transformation to each hidden-state.
- **Layer norm**:
  - This normalizes each hidden-state *across the features* i.e. not across the batch and not across the sequence.
  - This normalizes the inputs (or outputs) of each layer.
- **Multi-headed attention**:
  - The workhorse of transformers.
  - Allows for pairwise considerations between tokens.

Encoder

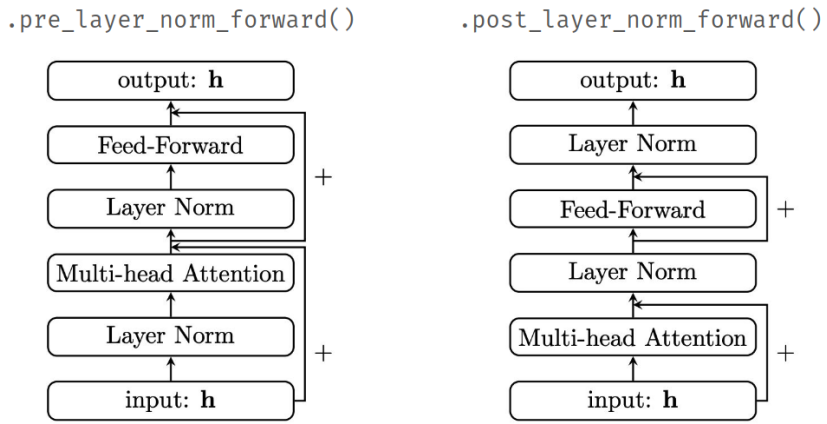

Figure: Pre- and post-norm encoder layer.

- TransformerEncoderLayer
- Multiple layers get stacked to form TransformerEncoder
- You will be asked to implement both versions
- The performance between the two will be very similar, as the differences only matter after a dozen-plus layers.
- Note the **residual connections** '+'. These are $h = h + f(h)$.

Decoder

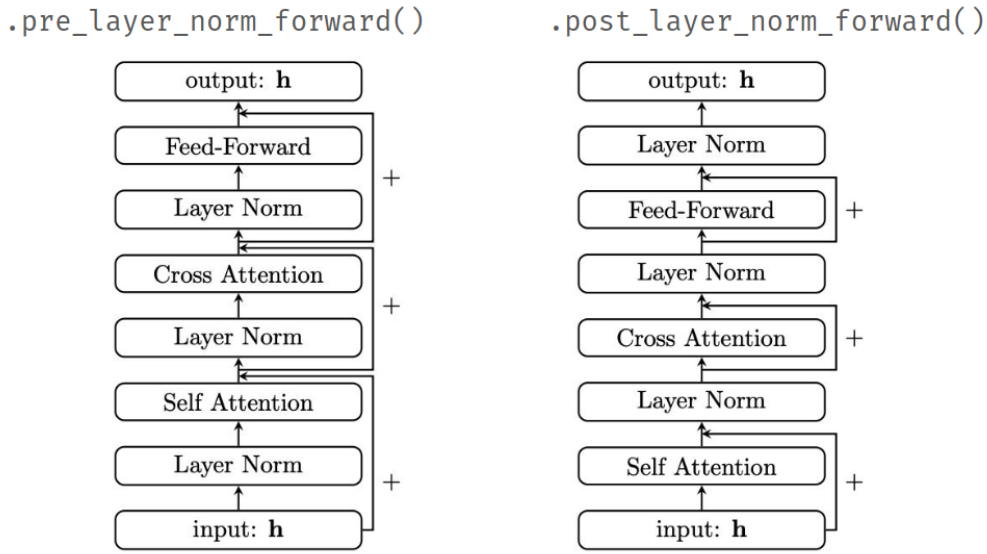

Figure: Pre- and post-norm decoder layer.

- TransformerDecoderLayer
- Multiple layers get stacked to form TransformerDecoder
- The same as the encoder but with **cross attention** as well.

Attention Mechanism

- Attention takes in a query (vector) sequence, Q, a key sequence K, and a matching value sequence V.
- Two different kinds of attention: **self-attention** and **cross-attention**.
- Self-attention assumes Q, K, V are all the same sequence.
  - This is used in the encoder where every token in X attends to all other tokens in X.
  - A **causally masked** version of this is used in the decoder where every token attends to all **prior** tokens in Y.
- Cross-attention is used when Q is a different sequence from K and V.
  - This is used in the decoder where each token in Y attends to all tokens in X in order to condition on the source-side information.

Attention Calculation

- Each of the Q, K, V has a linear projection applied to them.
- These are each then partitioned into h **heads** s.t. a d-dimensional vector becomes h smaller d/h-denominational vectors.
  - This is done by reshaping the tensor.
- Attention scores are produced via the dot product between Q and K (which is scaled by a constant $\sqrt{d/h}$)
- Attention 'probabilities' or normalized scores are produced via the softmax function.
- These then are used to create a weighted average from V.

$$\text{dropout}(\text{softmax}(\frac{\mathbf{QK}^\top}{\sqrt{d/h}}))\ \mathbf{V} \tag{1}$$

Masked Attention

Some tokens need to be excluded from attending to each other.

- Our data is **batched** sequences which means they will not all be the same length.
- They are padded to length in the batch with a special pad token (sometimes called the 'pad_idx' in the code).
- Call this the length mask or **pad mask**.
- These are masked from each other by setting their attention probabilities to be zero.
- What is the proper way to do this for softmax? (It's -inf for those who missed the tutorial)

Causal Mask and Teacher-Forcing

Tokens in Y can only attend to prior tokens.

- During training we use teacher-forcing. This inputs the ground-truth sequence Y
- However, we need to enforce the causal constraint that tokens in Y can not 'see into the future'.
- This is done via a triangular attention mask such that y_1 can only attend to y_1 and y_2 can only attend to y_1 and y_2 etc.

Note: the target sequence is also padded to length for batching, however, this can be accounted for by masking the cost.

Causal Mask

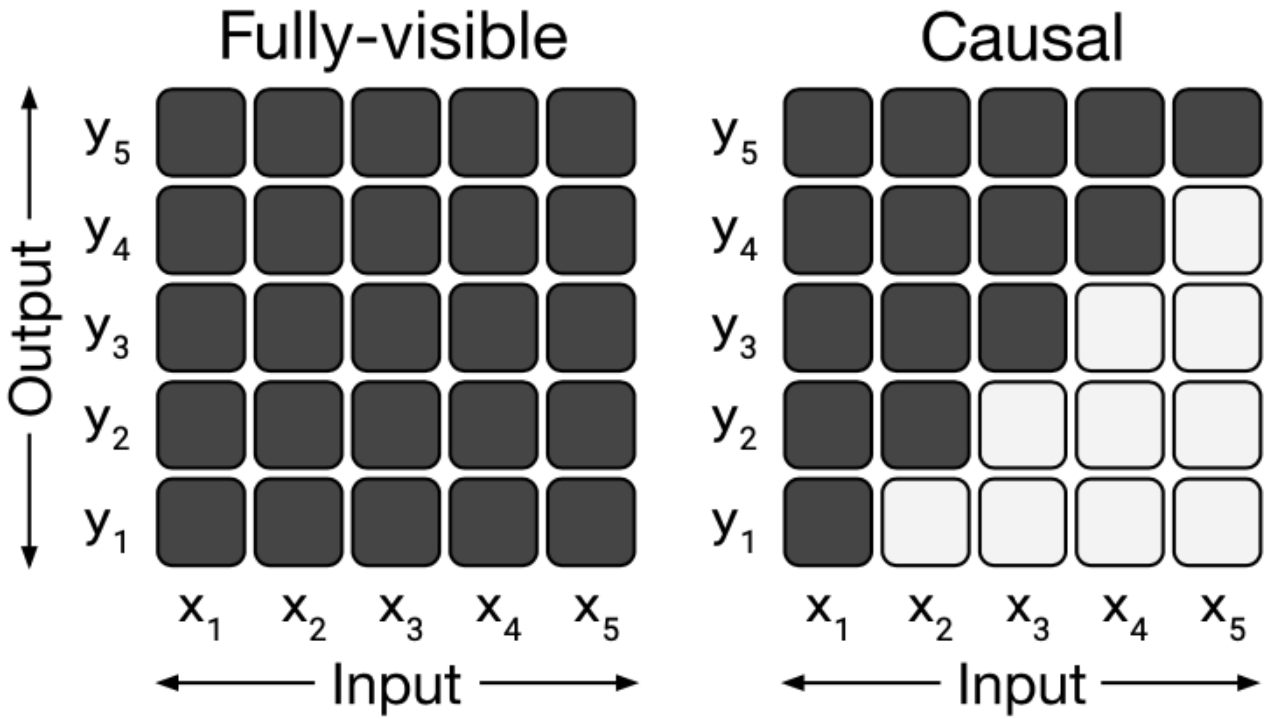

Figure: Full attention and causal attention (fig. from here).

Assignment 2

Five parts:

1. Implementing transformer building blocks
2. Implementing transformer model
3. Implementing greedy and beam search (inference/decoding)
4. Implementing the training and inference loops (training/testing)
5. Experiments or analysis

Assignment 2

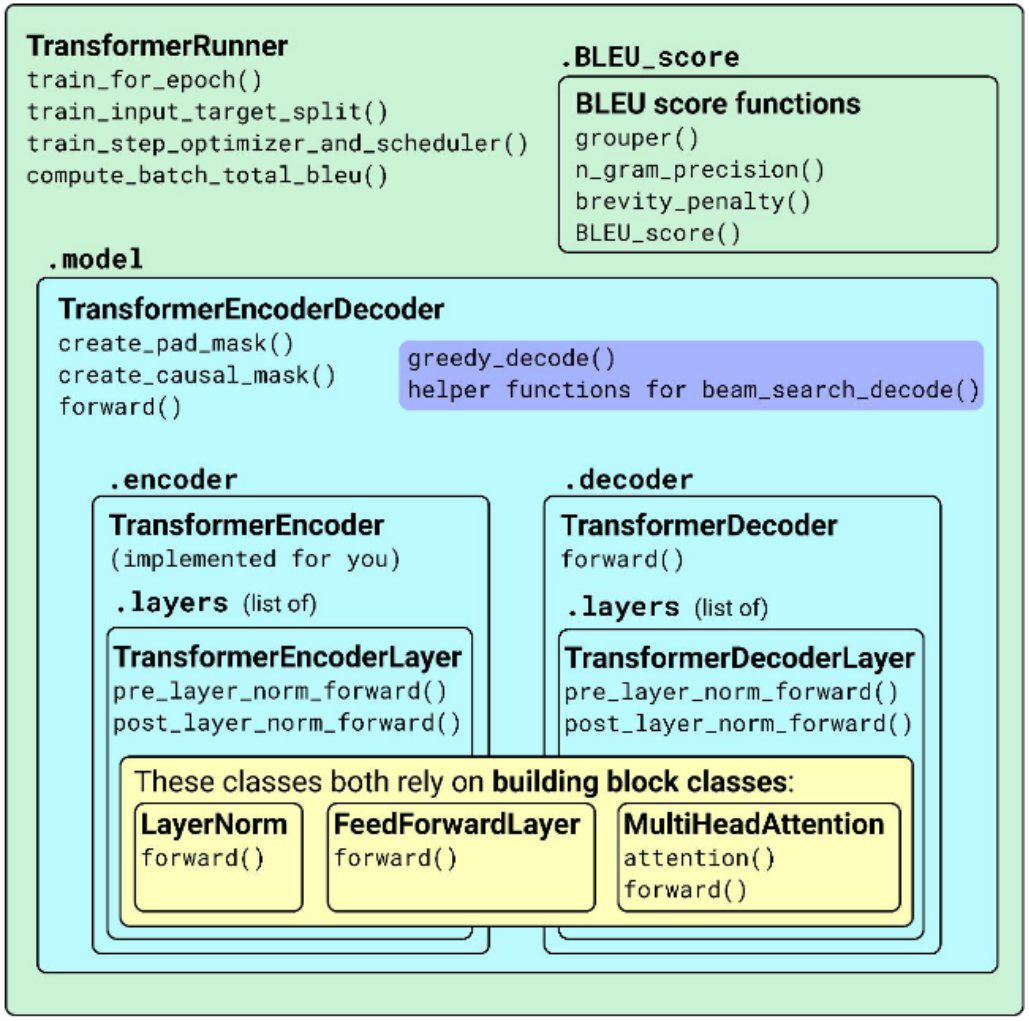

Figure: Assignment 2 code structure.

Assignment 2 Tips

- A lot of work:
  - Don't be overwhelmed, it's not difficult but there are a lot of things you need to consider at once which can be challenging.
  - Start early. Minor errors in neural nets can be catastrophic and hard to debug. Also, you don't want to compete for GPUs.
- Star with a) training loop b) TransformerEncoderDecoder forward c) building blocks, d) BLEU score
- Leave inference (greedy and beamseach decoding) until you have the majority of the model and training loop done.
- Try only partially completing the function in order to get a single training forward step working. That is, bypass parts of the function when necessary or helpful.

Assignment 2 Tips

- Match tensor shapes like Lego and always comment and print out the shapes of any given tensor.
- First thing you should do is look at how the dataloader gives data. Print out tensors, tensor shapes and try to convert sentences back to strings. This is not required but will help you understand data.

Next Tutorial

- Inference
- Back-propagation and optimization in Torch
  - Please see this image classification tutorial in Torch if you have never done these before.
- Gradient accumulation