CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2022 Marvin Zhang **Discussion 7**

This discussion covers Transformers and Pretraining.

1 Transformers

At a high-level, transformers consist of the Transformer Encoder and Transformer Decoders.

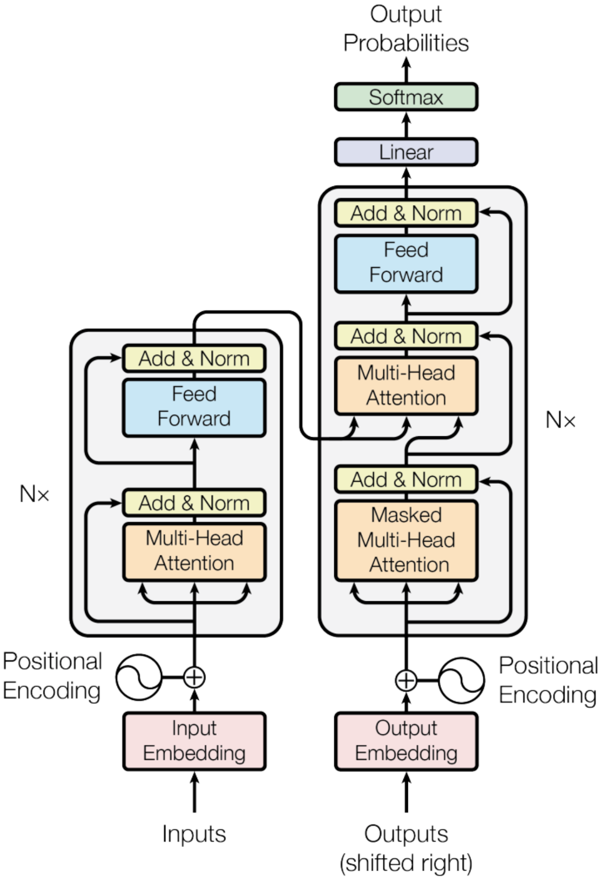

Figure 1: Overview of Transformer architecture

Both operate similarly, except the Transformer Decoder takes x_{target} as input, but Transformer Encoder takes in x_{source} as input. In addition, there are several differences in cross-attention and self-attention operations. In particular, transformers are novel in that they add,

- Positional Encoding: Addresses lack of sequence information
- Multi-headed Attention: Allows querying multiple positions at each layer
- Non-linearities

- Masked Decoding: Prevent attention lookups into the future

1.1 Notations

To ensure a level of clarity, we will let B be the batch size, L_{source} represent the source sequence length, L_{target} be the target sequence length, D represent the model hidden dimension and H represent the number of attention heads.

In particular, transformers receive two sequences as input. The first is $x_{source} \in \mathbb{Z}^{B\times L_{source}}$ and the second is $x_{target} \in \mathbb{Z}^{B\times L_{target}}$. These are integer tensors, and each integer represents a word or token.

1.2 Transformer Encoders

Input & Positional Embedding The source tensor is embedded into the model hidden dimension, and produces a tensor $X_{source} \in \mathbb{R}^{B\times L_{source}\times D}$. We then add a positional encoding that differs for each sequence position in order to enable the model to differentiate the positions in the sequence. In general, we need this information since position of words in a sentence carries information.

Encoder Attention The Encoder Attention is self-attention. Specifically, in Transformer networks, we use the Scaled QKV Attention (not covered explicitly in lecture). In other words, we would like to build a representation of a single sequence such that every position in the sequence has information about every other position in the sequence. In particular, to enable this, we will use the Query-Key-Values (QKV) Attention. Our queries, keys and values will be tensors in $X_{source} \in \mathbb{R}^{B\times L_{source}\times D}$ and weight matrices will be $W_Q, W_K, W_V \in \mathbb{R}^{D\times D}$. Ultimately, we will retrieve,

$$
\begin{aligned}
Q &= X_{source}W_Q \\
K &= X_{source}W_K \\
V &= X_{source}W_V
\end{aligned}
$$

Using Q, K, V, we will compute the attention scores (tensor in $\mathbb{R}^{B\times L_{source}\times L_{source}}$). For each element in the batch, each entry i, j in the matrix would be $\frac{q_i^\top k_j}{\sqrt{D}}$ for scaled dot product attention. Alternatively, we can compute, $\frac{QK^\top}{\sqrt{D}}$. To produce weights over each position in the sequence, we want each score to sum to one over the keys K. To accomplish this, we take a softmax update over the last dimension of the attention scores. Then, to produce the attention update, we multiply these attention weights by our values V,

$$C_{update} = \mathsf{softmax}\left(\frac{QK^\top}{\sqrt{D}}\right)V$$

where $C_{update} \in \mathbb{R}^{B\times L_{source}\times D}$

One of the key changes in Transformers is the *multi-headed attention* mechanism. To turn it into multi-headed attention, we can take any such update matrices and reshape and permute the matrix from shape $B \times L_{source} \times D$ to $B \times H \times L_{source} \times \frac{D}{H}$.

We finally consider padding. In general, we operate on a batch of B sequences, but these sequences may not be the same length. We pad each sequence to L_{source}. To prevent our model from paying attention to padded positions, we add $-\infty$ to attention scores prior to the Softmax of any position that should be ignored.

Feedforward Layer The feedforward layer applies linear transformation to each position, apply a nonlinear activation, then applies a second linear transformation.

1.3 Transformer Decoder

Masked Decoder Self-Attention Masked decoder self-attention is the same as encoder self-attention, but with different masking. In particular, we would like every position to pay attention to all previous positions, but not future positions. To achieve this, we set attention score to $\frac{q_i^\top k_j}{\sqrt{D}}$ if $i \leq j$ and $-\infty$ otherwise.

Encoder-Decoder Attention Encoder-Decoder attention operated similarly as well, except that we have two sequences: (1) generate queries and (2) generate keys-values. Hence, we let $Q = X_{target}W_Q, K = X_{source}W_K, V = X_{source}W_V$, where X_{source} is the output of the transformer encoder on the source sequences.

Problem: Machine Translation

1. What is the reason for positional encoding? How is it typically implemented?
2. What is the advantage of multi-head attention? Give some examples of structures that can be found using multi-head attention
3. For input sequences of length M and output sequences of length N, what are the complexities of (1) Encoder Self-Attention (2) Decoder-Encoder Attention (3) Decoder Self-Attention. Further let k be the hidden dimension of the network
4. Do activation of the encoder depend on decoder activation? How much additional computation is needed to translate a source sequence into a different target language, in terms of M and N?

Solution: Machine Translation

1. Position encoding is used to ensure that word position is known. Because attention is applied symmetrically to all input vectors from the layer below, there is no way for the network to know which positions were filtered through to the output of the attention block. Position encoding also allows the network to compare words (nearby position encodings have high inner product) and find nearby words. We can either use learned position encodings or precomputed sinusoids such that each dimension of the position encoding corresponds to a different sinusoidal frequency.
2. Multi-Head attention allows for a single attention module to attend to multiple parts of an input sequence. This is useful when the output is dependent on multiple inputs (such as in the case of the tense of a verb in translation). Attention heads find features like start of sentence and paragraph, subject/object relations, pronouns, etc.
3. (1) $\mathcal{O}(M^2k)$ (2) $\mathcal{O}(MNk)$ (3) $\mathcal{O}(N^2k)$
4. No. The encoder activations do not depend on the decoder activations. Thus, you only need $\mathcal{O}(MN + N^2)$ additional computation to decode into a new sequence.

1.4 Why Transformers

In general, transformers are good for long-range connections, are easy to parallelize and transformers can be made much deeper than RNNs. On the other hand, attention computations are complex to implement and computations take $\mathcal{O}(n^2)$ time.

However, in practice, it turns out the benefits vastly outweigh the downsides, and transformers work better than RNNs and LSTMs in many cases.

2 Unsupervised Pretraining

We will review several techniques for unsupervised pretraining with transformers, particularly in natural language processing (NLP). The general idea is to use unlabelled data, which is often easily accessible (for example text data on the internet, in books, other publications, etc...) in order to learn representations that can be useful for downstream tasks, such that not as much task-specific data is needed for good performance on that task.

To illustrate why we might expect this to be helpful, we can imagine we want to translate English sentences to French, and are given a labelled dataset of English/French sentence pairs. You can imagine this task would be really difficult if you had no prior knowledge of English, while being much more manageable if you came in with a general understanding of the English language already, which can be learned using unsupervised data (for example, all the English text we see on the internet).

2.1 Pretrained Language Models

At a high level, one simple way we can embed words in a context-dependent manner is to take a language model (for example an LSTM) trained on some task, and to run a sentence through it, taking the hidden state of the model as the embedding for each word. Since these language models presumably had to use the context in order to solve the task they were trained on, using the hidden state as an embedding should provide context-dependent representations of words.

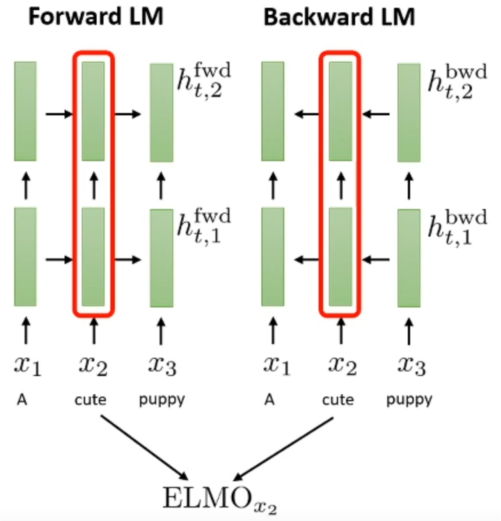

Figure 2: ELMo takes the hidden states in a bi-directional LSTM to generate word embeddings. The LSTMs are both trained via sequence prediction.

ELMo: We note that if we simply ran an LSTM forward through a sentence to generate the embeddings of words, the embedding of each word would only depend on those that came before it, rather than the full context of the word. ELMo addresses this issue by simply training a bidirectional LSTM (both trained to predict the next/previous word), and concatenating hidden states of both directions together to form an embedding. ELMo has been largely replaced by other models in current NLP research, and the following models are more represenative of what is currently used in NLP today.

GPT: GPT (and its successors GPT-2 and GPT-3) are high-capacity transformer-based language models trained on very large amounts of unlabeled text (e.g. text from the internet). Because they are forward generative language models, they model architectures consists only of a transformer decoder. While conceptually simple, these models can be incredibly powerful for generating text data, with the most recent version GPT-3 being able to generate text that is almost indistinguishable from text written by a human. The representations learned by GPT can also be effectively used for downstream tasks, but they may be a suboptimal from some tasks because GPT is a forward language model, so its representations only incorporate context from past context, not the entire sequence of text.

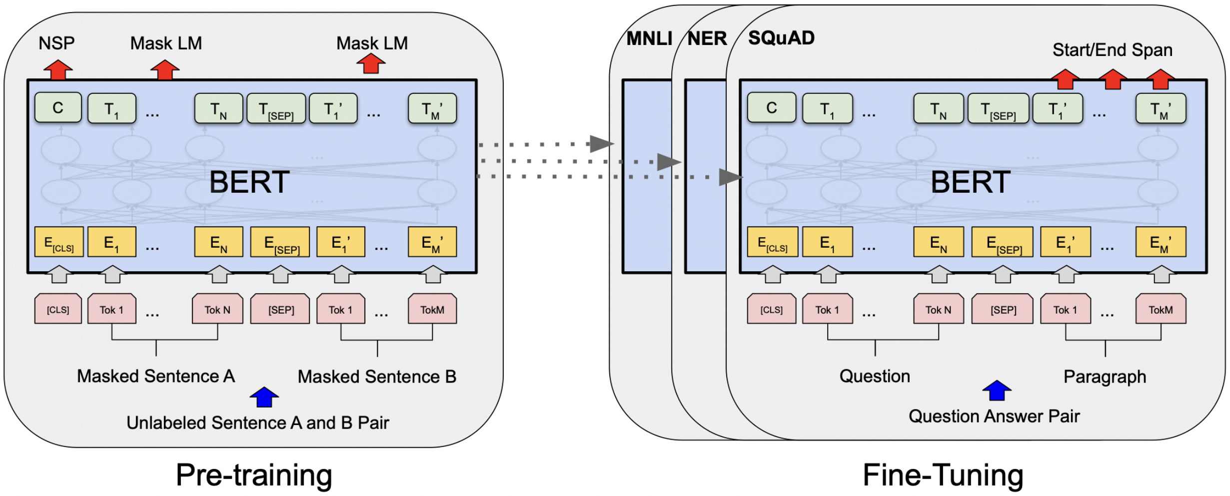

Figure 3: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating ques- tions/answers).

BERT: One can imagine incorporating bidirectional context with a transformer-based language model in similar manner as ELMo, where we can learn both a forward and backward language model and concatenate their embeddings. However, while such an embedding would capture bidirectional context, the individual tasks of forward and backward language modeling are inherently unidirectional, so simply concatenating their embeddings may not learn representations that capture bidirectional relationships well. Instead, BERT relies on a *single* transformer encoder to generate embeddings that incorporate bidirectional context, using an inherently bidirectional pretraining task.

While the previous transformers we saw for sequence modeling relied on masked self-attention to avoid peeking into the future, our goal here is to digest the entire context of a word to produce an embedding, which eliminates the need for the mask. However, this presents a complication if we were to try train embeddings to predict the next word like ELMo or GPT. The issue here is that if we did unmasked self-attention, we can already directly see the next word in the input, making prediction completely trivial and preventing useful representations from being learned.

The solution is to simply change the unsupervised task. Instead of predicting the next word in sentence, we instead randomly mask out certain words in the input, and then train the embedding to predict the masked out words. In this way, our model is forced to learn context dependent word-level representations to predict the missing words.

In addition to learning word-level representations by predicting masked out words, BERT also tries to learn *sentence-level* representations. To train this, BERT takes in pairs of sentences, half of which are consecutive and half of which are paired randomly. It trains a binary classifier to predict whether the two sentences are consecutive or not.

This pretraining procedure gives BERT the ability to produce powerful represenations for downstream tasks that require language understanding. Such tasks include sentiment analysis, textual entailment, and question answering. Depending on the downstream task, we can either use the sentence level representation outputted by BERT or the word-level representations in the downstream task. We can use BERT for downstream tasks both by simply finetuning the entire model on the downstream tasks, or taking combinations of the hidden states as fixed representations.

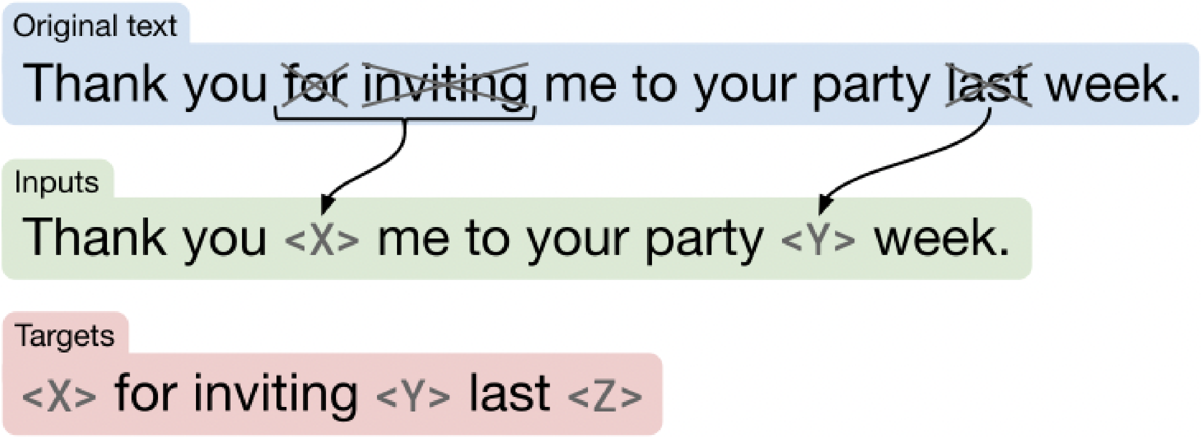

Figure 4: Example of how the BERT-style masked language modeling pretraining task is adapted to T5.

T5: T5 was the result of an extensive empirical analysis on the best practices for pre-training a large transformer model for transfer learning on downstream tasks. They investigated various design decisions including model architectures, pre-training objectives, and pre-training datasets. In the end, they concluded the best performance was offered by the BERT-style masked language modeling pre-training objective, but changing the architecture to be a standard encoder-decoder transformer, instead of using only a transformer encoder like BERT. They do this by proposing to reframe all NLP tasks (including pre-training and downstream tasks) into a unified text-to-text (sequence-to-sequence) format. For example, for the masked language modeling task, the input to the encoder is the same as it would be for BERT, but now the decoder is trained to autoregressively predict a sequence that contains the predictions for the missing text.

Through this architecture choice, T5 is more flexible and easily adapted for sequence-to-sequence downstream tasks, such as machine translation. Also, the more general and flexible architecture of T5 allows it to be more readily used for multi-task learning, where a single model can be fine-tuned on multiple downstream tasks, which can potentially lead to better performance than training on any single task alone. At the time of its development, T5 achieved state-of-the-art performance on many popular NLP benchmarks.

Problem: Pretrained Language Models

What are the pros and cons of each of the discussed pretrained language models? In which situations is each type of model most useful for?

Solution: Pretrained Language Models

ELMo, BERT, and T5 are most useful for downstream tasks that require bidirectional context for understanding the content of some text. BERT and T5 are typically even better suited for these tasks because its pre-training task is inherently bidirectional, unlike ELMo. BERT and T5 also use transformers instead of LSTMs, which can be helpful for modeling long-term dependencies and parallelization of training on large datasets. T5 can be more broadly applicable than BERT due to its more flexible text-to-text framework. GPT models are less suitable for downstream tasks that require text understanding because its representations only incorporate unidirectional context, but they are better suited for text generation because they were explicitly trained for this purpose, and text generation requires unidirectional contexts that models like BERT and T5 were not pre-trained on.