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

Discussion 8

This discussion covers unsupervised pretraining methods with transformers, and beam search.

1 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).

1.1 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.

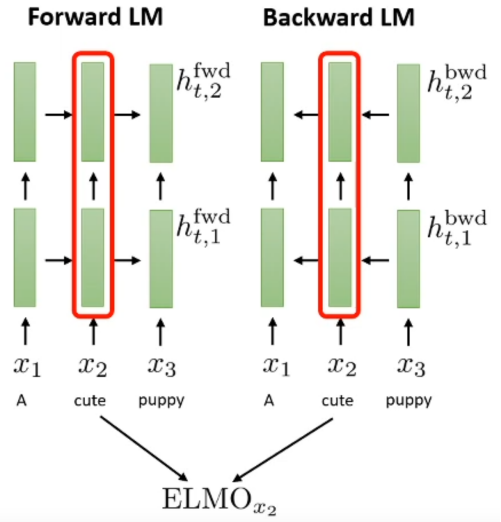

Figure 1: ELMo takes the hidden states in a bi-directional LSTM to generate word embeddings. The LSTMs are both trained via sequence prediction.

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.

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.

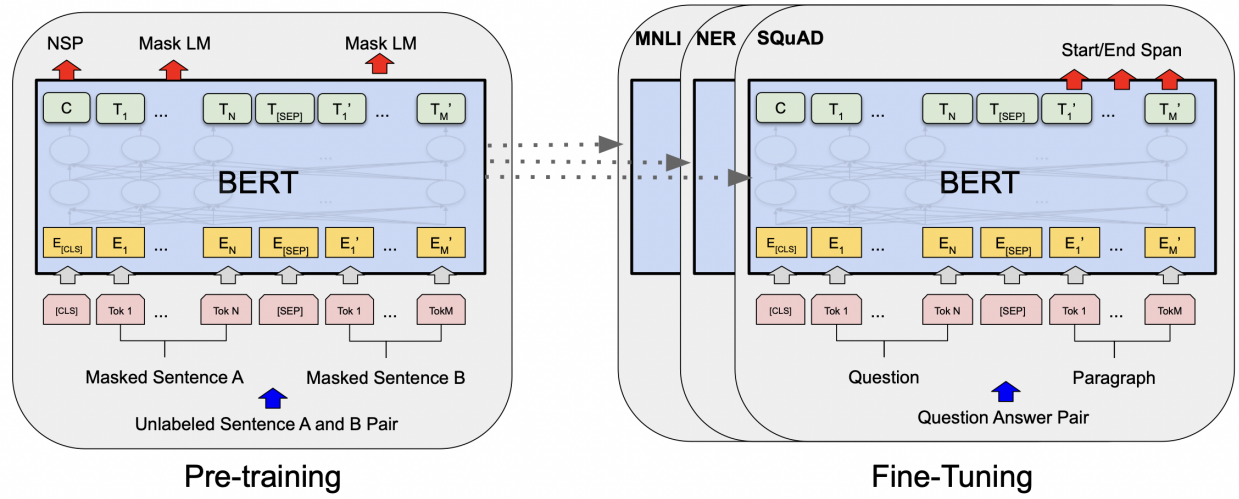

Figure 2: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architec- tures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters 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, randomly permutes their order, and trains a binary classifier to predict which of the two sentences came first originally.

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.

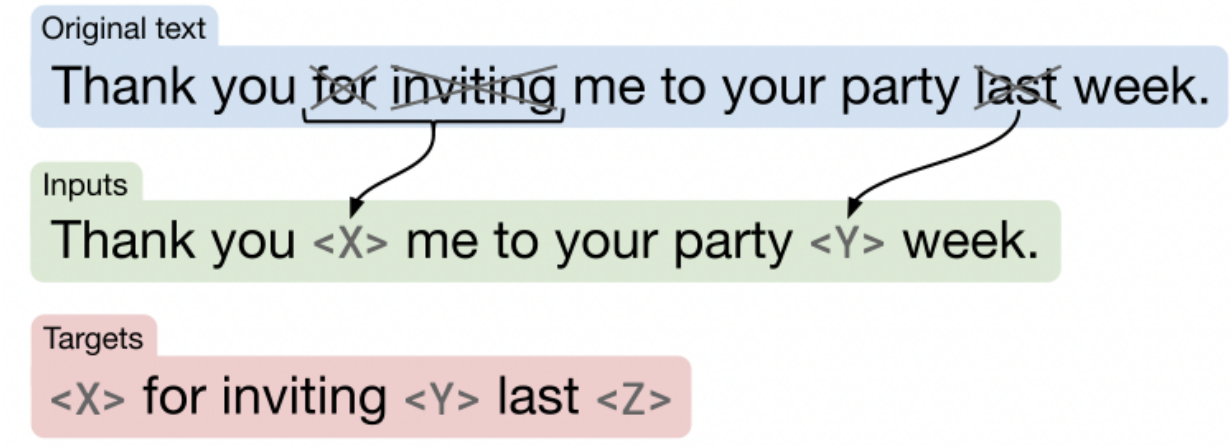

Figure 3: Example of how the BERT-style masked language modeling pretraining task is adapted to T5.

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 similar to as in 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?

2 Beam Search

When making predictions with an autoregressive sequence model, it can be intractable to decode the true most likely sequence of the model, as doing so would require exhaustively searching the tree of all $O(M^T)$ possible sequences, where M is the size of our vocabulary, and T is the max length of a sequence. We could decode our sequence by greedily decoding the most likely token each timestep, and this can work to some extent, but there are no guarantees that this sequence is the actual most likely sequence of our model.

Instead, we can use beam search to limit our search to only candidate sequences that are the most likely so far. In beam search, we keep of the k most likely predictions of our model so far. At each timestep, we expand our predictions to all of the possible expansions of these sequences after one token, and then we keep only the top k of the most likely sequences out of these. In the end, we return the most likely sequence out of our final candidate sequences.

The beam search procedure can be written as the following pseudocode:

Algorithm 1 Beam Search

for each time step t **do**
  for each hypothesis $y_{1:t-1,i}$ that we are tracking **do**
    find the top k tokens $y_{t,i,1},...,y_{t,i,k}$
  end for
  sort the resulting k^2 length t sequences by their total log-probability
  keep the top k
  advance each hypothesis to time $t+1$
end for

Problem: Beam Search

We are running the beam search to decode a sequence of length 3. Consider the following probability predictions of the decoder, where each node in the tree represents the next token log probability prediction of one step of the decoder conditioned on previous tokens. The vocabulary consists of two word: "neural", and "network". What sequence/sequences could beam search with $k = 2$ output?
Decoder log probability prediction of next token given previous tokens: