Announcements

- A2 is due on Friday at 11:59pm
  - Careful with the number of late days you have left! If you used three late days on the last assignment, you have two late days left for the quarter.
  - Make sure to commit/push your .preds files by the deadline, in addition to your code and writeup
- Extra office hours this week – the course google calendar lists the OH schedule
- Quiz 5 goes out on Canvas today at the end of lecture
  - Available until Friday at 2:20pm; you’ll have 15 minutes to complete it once you start.
  - Remember that you can use your notes during the quiz
  - Will cover material from last Wednesday’s lecture through the end of Monday’s lecture (so, Viterbi, CRFs, and neural sequence labeling)
Transformers: outline

LSTMs: their pros and cons

The transformer architecture at a high level and how it addresses LSTMs’ cons

Attention mechanisms

The transformer architecture at a slightly lower level

The additional idea behind BERT and co. (pretraining and finetuning!)
Things we like about LSTMs

Can deal with arbitrary-length sequences (like text!) while taking the order of the sequence into account (like text does!)

Were the dominant model architecture in NLP for years for a wide range of tasks
Things we don't like as much about LSTMs

Recency bias (references at end)

LSTMs were designed to mitigate this issue compared to Elman RNNs, but still suffer from it

Time required to train an LSTM
A brief aside about some visual shorthand I'll be using
A 3-layer LSTM's calculations for an input of 10 tokens
One layer of the transformer architecture (Vaswani et al. 2017)
One layer of the transformer architecture (Vaswani et al. 2017)

ok but how does this mess help anything sofia
Recency bias is not as much of a problem as in LSTMs
Comparing training times: how many functions do we need to backpropagate through?
Comparing training times: how many functions do we need to backpropagate through?

**Transformers parallelize a lot of the computations that LSTMs make us do in sequence**
Comparing training times: how many functions do we need to backpropagate through?

**Transformers parallelize a lot of the computations that LSTMs make us do in sequence**

And (a very specific, but nonempty, subset of) you can therefore train a transformer on a ridiculously large amount of data in a way that you cannot for an LSTM.
What kind of function can take in a variable number of inputs like that without recursively applying an operation a bunch of times?
Attention mechanisms
Building up to the attention mechanism

What about an average?

But we probably don’t want to weight all input vectors equally...

How about a weighted average?

Great idea! **How can we automatically decide the weights for a weighted average of the input vectors?**

What kind of function can take in a variable number of inputs like that without recursively applying an operation a bunch of times?
Bahdanau attention (Bahdanau et al. 2014)

- Parameter vector
- (Variable number of) input vectors

**Computed how?**
1. Dot product between param vector and each input vector
2. Softmax the set of resulting scalars.
Pros and cons

Pros:

● We have a function that can compute a weighted average (largely) in parallel of an arbitrary number of vectors!
● The parameters determining what makes it into our output representation are learned

Cons:

● We’re also hoping to produce n different output token representations... and this just produces one...
“What if instead of comparing each vector of the sequence to a single learned vector, we compared the sequence to itself?”
(num attention heads) * (dimension of each attention head)

learned Q params

learned K params

learned V params

technically could be any constant divisible by the number of att heads, but huggingface always sets it to the dimension of the input vectors, for good reason

Q

K

V

number of att heads

dimension of each attention head

input to function

one attention head likely move these columns

input to function (possibly from an encoder)

input to function (possibly from an encoder)

number of att heads

dimension of each attention head

Add learned bias vector to each row, then reshape each row

Add learned bias vector to each row, then reshape each row

Add learned bias vector to each row, then reshape each row

individual
tokenvectors

queries

keys

values
number of attt heads

dimension of each attention head

Q

dimension of each attention head

K

dimension of each attention head

V

K turned on its side

big mxn matrices, one for each attention head (huggingface calls these "attention scores")

divide all elements by \sqrt{\text{dim of each attt head)}},

then softmax over each row vector (separate softmax for each (row, attt head) pair)

new big mxn matrices

numbers \geq 0 that add up to 1

(a different set for each row/attt head pair)

individual
token
vector
pretext representation

new text representation

individual
token
vectors

concatenate all attt heads' contributions
Hooray for self attention!

Our function is still made up almost entirely of matrix multiplications! *Which are very parallelizable* (→ efficient!)

We still learn fixed-size blocks of parameters that can be used for a sequence with an arbitrary length.

We’re now capable of producing $n$ different new token representations!
Self attention is the key component of the transformer
Filling in some last transformer details
The full figure of a layer from the transformer

From Vaswani et al. 2017
Position embeddings

From Vaswani et al. 2017
Position embeddings

Probably the least intuitive part of a transformer.

A transformer’s only sense of the order of words is a set of position embeddings, one per token index, that are added to the corresponding tokens of an input.

In practice, this also means that unlike for LSTMs, the maximum length of a sequence for a transformer is capped [at the number of position embeddings it’s got].

The original transformer paper experimented with both learned positional embeddings and sine/cosine-based positional embeddings (sec 3.5).
Masked attention

From Vaswani et al. 2017
Masked attention

If you’re learning a model that’s supposed to be able to generate text token by token (an “autoregressive model”)… then looking ahead to previous tokens during training would be cheating.

In practice, we mask and renormalize the attention distributions to include only the tokens that that time step has seen so far. (In other words, for the token representation at position $t$, only take an attention distribution over the first $t$ tokens.)
Encoder AND decoder??

From Vaswani et al. 2017
Encoder AND decoder??

Only for some tasks! (For example, machine translation, where having the input space and output space have different sets of trained word embeddings makes more sense)

Notably, BERT:

GPT___:
So we like this architecture, but what will we train it to do?
Issues with just training the model to do your task of interest

Your task of interest might not have that much labeled data available.

Even if that weren’t an issue, these models are quite large, and take a lot of resources to properly train.

Feels like a waste to have each separate project consume all those resources.

(Never mind that a lot of people who’d like to use these models don’t have access to those kinds of resources.)
Pretraining and finetuning

Based on idea of transfer learning (not a new idea in machine learning-- Pan and Yang 2010 cite a NeurIPS ‘95 workshop as already discussing this idea)

Pretraining and finetuning is basically transfer learning, BUT with the understanding that the vast majority of training is accomplished in the pretraining stage.

What kinds of tasks make good, generalizable pretraining tasks?
ELMo  (*Peters et al. 2018*)

The big takeaway:

if you train a **language model**,  
then just replace the model’s output layer and use the parameters from the original language model to adjust your word embeddings in the model’s first few layers, 

those new word embeddings can help you perform **really** well on a whole range of NLP tasks, especially if you **finetune** the parameters (i.e., train them to perform your actual task of interest).
The BERT training objective (Devlin et al. 2018)

Very similar idea to ELMo, but

- used the transformer architecture (unlike ELMo)
- used masked language modeling as its pretraining objective instead

The quick brown _______
The quick brown fox _______

The ______ brown fox jumps ______ the...
The quick _______ ______ jumps over the...
Side note: the reason for this little red box

Want a good representation of a sentence?

It’s common to use BERT (or RoBERTa or something) to encode the sentence, and then just take the first representation (corresponding to the special [CLS] token) from the final layer in the transformer as a representation of the sentence as a whole.
Sampling from a trained autoregressive LM transformer
Lots of different sampling strategies proposed

For example, Nucleus Sampling (which we won’t talk about today due to time).

The way in which you sample from a language model’s output probability distribution can have a big effect on the kind of text you get!

We’ll briefly go over the top-K sampling algorithm as an example.
The top-K sampling algorithm

We will represent $P(\cdot | W_{1..i})$ by $p = (p_1, p_2, ..., p_{|V|})$, where the elements is sorted that $p_1 \geq p_2 \geq p_3 \ldots \geq p_{|V|}$.

Top-K sampling transforms $p$ to $\hat{p}$ by:

$$\hat{p}_i = \frac{p_i \cdot 1\{i \leq K\}}{Z}$$

And we sample $W_{i+1}$ from $\hat{p}$.

$$\sum_{w \in V_{\text{top-K}}} P(w| \text{“The”, “car”}) = 0.99$$

← from
https://huggingface.co/blog/how-to-generate
(Optional reading)
Examples from the GPT2 model

- Prompt: MIT is a private research university in Cambridge, Massachusetts. It is one of the best universities in the U.S.,

- GPT2 with naive sampling: but the teaching of traditional African-American studies and African-American literacy continued. Soon thereafter, MIT was renamed The International Comparative University by Lord (then), ...

- GPT2 with topk40 sampling: and the home of most of the top international universities in the world. Our alumni are internationally renown, but our mission is unique. We are the only university in the world where there is a chance to take on the challenge of making an impact, ...

- topk40 another sample: with a reputation for innovation and open and flexible public systems. Its principal research area deals with autonomous vehicles, robotics and artificial intelligence. To date, MIT has published 40 peer-reviewed papers on this topic, ...

- Message: sampling algorithms provide a sweet quality-diversity trade-off.
- (which is the key difference to decoding e.g., beam-search)
- Tianxing did not cherry-pick these examples!

Slide by Tianxing He
On recency bias in LSTMs, and comparing LSTMs to the transformer:


One Layer of Self-Attention for One Instance in a Batch

Technically, the number of self-attention heads is equal to the dimension of the individual Q, K, and V vectors. However, for reasons of efficiency and to make the model more flexible, we typically choose a smaller number of heads. In our example, we have chosen to use 8 heads.

The Q, K, and V vectors are learned parameters. They are initialized randomly and then fine-tuned during training to learn the relationships between the input tokens.

The Q, K, and V vectors are then used to compute the attention scores, which are used to weight the input tokens. The attention scores are computed by taking the dot product of the Q and K vectors, and then taking the softmax of the result.

The attention scores are then used to compute the weighted average of the V vectors, which is the output of the self-attention layer.

The output of the self-attention layer is then fed into the next layer of the neural network.