

Notes

29 March 2024 08:00

Outline

Lecture Plan

- From recurrence (RNN) to attention-based NLP models
- The Transformer model
- Great results with Transformers
- Drawbacks and variants of Transformers

Reminders:

Extra details are in the [brand new lecture notes](#), wooooo!

Assignment 4 due a week from today! Use Colab for the final training if you don't have a GPU.

Final project proposal out tonight

Please try to hand in the project proposal on time; we want to get you feedback quickly!

Stanford

What's wrong with RNNs?

- After discovery of RNNs, the go to way for NLP problems was uni/bi-directional LSTM equipped with convolutional attention
- However RNNs suffer from:
 - Modelling Long-distance dependencies
 - Parallelizability: future hidden state calculations cannot be done before the past hidden states are calculated

Basic Idea of attention

If not recurrence, then what? **How about attention?**

- Attention treats each word's representation as a **query** to access and incorporate information from a **set of values**.
 - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.

8

Attention as a soft, averaging lookup table

We can think of **attention** as performing fuzzy lookup in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.

9

Self-Attention: keys, queries, values from the same sequence

Let $w_{1:n}$ be a sequence of words in vocabulary V , like Zuko made his uncle tea.

For each w_i , let $x_i = Ew_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

- Transform each word embedding with weight matrices Q, K, V , each in $\mathbb{R}^{d \times d}$
$$q_i = Qx_i \quad (queries) \quad k_i = Kx_i \quad (keys) \quad v_i = Vx_i \quad (values)$$
- Compute pairwise similarities between keys and queries; normalize with softmax
$$e_{ij} = q_i^T k_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij})}$$
- Compute output for each word as weighted sum of values

3. Compute output for each word as weighted sum of values



How attention solves problems of RNNs?

- Attention scores can be calculated for all the words in parallel
- (embeddings of) All the words have access to (embeddings of) all other words – so no linear dependency distance

Potential problems with attention

- Sequence of words is not taken into consideration: Then let's encode the positional information

Fixing the first self-attention problem: sequence order

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$p_i \in \mathbb{R}^d$, for $i \in \{1, 2, \dots, n\}$ are position vectors

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Recall that x_i is the embedding of the word at index i . The positioned embedding is:

$$\tilde{x}_i = x_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

13

How to encode positional information?

Option 1: Sinusoidal encoding

Position representation vectors through sinusoids

- Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2+1/d}) \\ \cos(i/10000^{2+1/d}) \\ \vdots \\ \sin(i/10000^{2+\frac{d}{2}/d}) \\ \cos(i/10000^{2+\frac{d}{2}/d}) \end{pmatrix}$$

Dimension
Index in the sequence

- Pros:
 - Periodicity indicates that maybe "absolute position" isn't as important
 - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
 - Not learnable; also the extrapolation doesn't really work!

Stanford

Option 2: Learned Positional encoding

Position representation vectors learned from scratch

- Learned absolute position representations:** Let all p_i be learnable parameters!
Learn a matrix $p \in \mathbb{R}^{d \times n}$, and let each p_i be a column of that matrix!
- Pros:
 - Flexibility: each position gets to be learned to fit the data
- Cons:
 - Definitely can't extrapolate to indices outside $1, \dots, n$.
- Most systems use this!
- Sometimes people try more flexible representations of position:
 - Relative linear position attention [Shaw et al., 2018]

- no non-linearity -- easy to fix by adding a feed forward network containing non-linearity

- Not looking into future -- I.e. not looking at words that are yet to come (especially in decoding)

- Restricting q, k to only past words is possible but in-efficient
- Simply mask out words in future

- To use self-attention in decoders, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys** and **queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

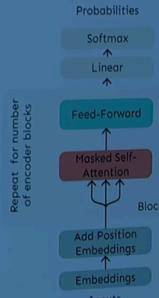
$$e_{ij} = \begin{cases} q_i^T k_j, & j \leq i \\ -\infty, & j > i \end{cases}$$

Complete minimal design of self-attention block

Necessities for a self-attention building block:

- Self-attention:**
 - the basis of the method.
- Position representations:**
 - Specify the sequence order, since self-attention is an unordered function of its inputs.
- Nonlinearities:**
 - At the output of the self-attention block
 - Frequently implemented as a simple feed-forward network.
- Masking:**
 - In order to parallelize operations while not looking at the future.
 - Keeps information about the future from "leaking" to the past.

21



Transformers

This section (from 40:00 – 56:00) discusses basics of attention/multi-head attention. I would like to refer to the **Illustrated Transformer** article written by Jay Alammar - <http://jalammar.github.io/illustrated-transformer/>

Below are the notes of key modifications introduced in attention mechanism to make transformer architecture work better.

Scaled-dot product attention:

Q: Why to scale (down) the dot product of Q and K?

- Remember gradient of softmax function is $\text{softmax}(\text{input}) \times (1 - \text{softmax}(\text{input}))$

Scaled Dot Product [Vaswani et al., 2017]

- "Scaled Dot Product" attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
 - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we've seen:

$$\text{output}_\ell = \text{softmax}(X Q_\ell K_\ell^T X^T) * X V_\ell$$
- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

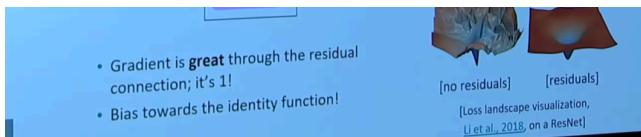
$$\text{output}_\ell = \text{softmax}\left(\frac{X Q_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * X V_\ell$$

Q: Why to use residual connections in transformer ?

The Transformer Encoder: Residual connections [He et al., 2016]

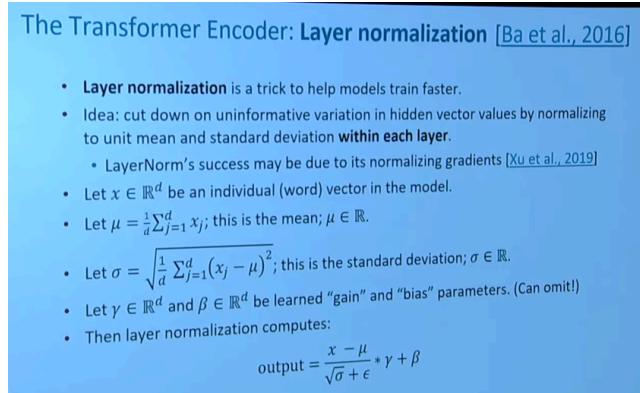
- Residual connections** are a trick to help models train better.
 - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

- We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)



Q: Why to normalization in attention mechanism?

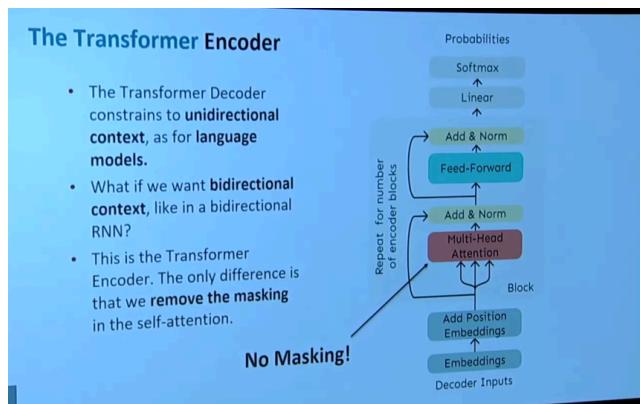
- All the matrix multiplications, non-linearities vastly impact the magnitude of values inside the NN.
- High-variance in magnitude of these values (either input values and/or resulting gradients) introduces numerical instability in training.
- That's why normalize the inputs so that we can maintain unit mean and variance



Transformer Encoder:

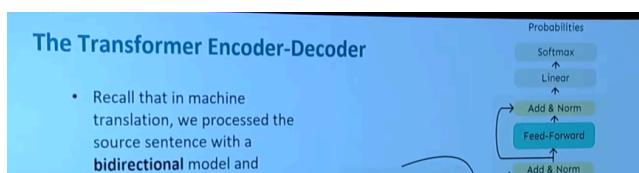
What changes from decoder?

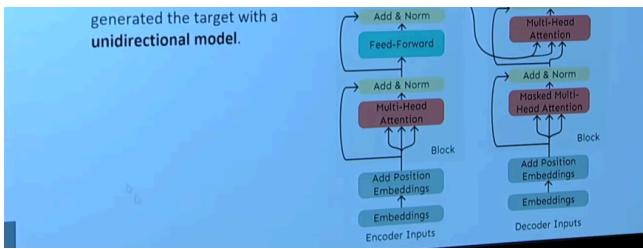
- Only the attention is non-masked (so future tokens/words) are visible to give a sense of bidirectional model



Complete Transformer

- Encoder + decoder put together.
- Still a uni-directional model, to introduce bi-directionality Cross attention is used. Cross attention also introduces a sense of memory (As decoder goes back to encoder again and again to see what encoder has produced and then using attention calculations decoder computes encoder features with high similarity for current word being decoded)
- Cross attention:
 - Using Key and Queries from encoder's output and values from decoder's features.
I.e. Decoder using all the information available in encoder for decoding each word.





Drawbacks of transformer / self-attention:

- Quadratic complexity of attention calculations wrt. The sequence length
 - The RNNs/LSTMs only had linear complexity wrt. Sequence length