Transformers
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley
Is Attention All We Need?
If we have **attention**, do we even need recurrent connections?

Can we transform our RNN into a **purely attention-based** model?

Attention can access **every** time step

Can in principle do **everything** that recurrence can, and more!

**This has a few issues we must overcome:**

- Problem 1: now step $l = 2$ can’t access $s_1$ or $s_0$
  - The encoder has no temporal dependencies at all!
  - We **must** fix this first
Self-Attention

\[ a_t = \sum_{t} \alpha_{l,t} v_t \]
\[ \alpha_{l,t} = \frac{\exp(e_{l,t})}{\sum_{t'} \exp(e_{l,t'})} \]
\[ e_{l,t} = q_l \cdot k_t \]
\[ v_t = v(h_t) \quad \text{before just had } v(h_t) = h_t, \text{ now e.g. } v(h_t) = W_v h_t \]
\[ k_t = k(h_t) \quad \text{(just like before)} \quad \text{e.g., } k_t = W_k h_t \]
\[ q_t = q(h_t) \quad \text{e.g., } q_t = W_q h_t \]

this is not a recurrent model!

but still weight sharing:
\[ h_t = \sigma(W x_t + b) \]

(or any other nonlinear function)
Self-Attention

keep repeating until we’ve processed this enough
at the end, somehow decode it into an answer (more on this later)
From Self-Attention to Transformers

The basic concept of self-attention can be used to develop a very powerful type of sequence model, called a transformer.

But to make this actually work, we need to develop a few additional components to address some fundamental limitations:

1. Positional encoding addresses lack of sequence information
2. Multi-headed attention allows querying multiple positions at each layer
3. Adding nonlinearities so far, each successive layer is linear in the previous one
4. Masked decoding how to prevent attention lookups into the future?

$$a_t = \sum_t \alpha_{t,t} v_t$$

$$v_t = W_v h_t$$
Sequence Models with Self-Attention
From Self-Attention to Transformers

The basic concept of self-attention can be used to develop a very powerful type of sequence model, called a transformer.

But to make this actually work, we need to develop a few additional components to address some fundamental limitations:

1. Positional encoding addresses lack of sequence information
2. Multi-headed attention allows querying multiple positions at each layer
3. Adding nonlinearities so far, each successive layer is linear in the previous one
4. Masked decoding how to prevent attention lookups into the future?

\[ a_t = \sum_{t} \alpha_{l,t} v_t \]
\[ v_t = W_v h_t \]
Positional encoding: what is the order?

what we see:
he hit me with a pie

what naïve self-attention sees:
a pie hit me with he
a hit with me he pie
he pie me with a hit

most alternative orderings are nonsense, but some change the meaning

in general the position of words in a sentence carries information!

Idea: add some information to the representation at the beginning that indicates where it is in the sequence!

\[ h_t = f(x_t, t) \]

some function
Positional encoding: sin/cos

**Naïve positional encoding:** just append $t$ to the input $x_t = \begin{bmatrix} x_t \\ t \end{bmatrix}$

This is not a great idea, because **absolute** position is less important than **relative** position.

I walk my dog every day

every single day I walk my dog

The fact that “my dog” is right after “I walk” is the important part, not its absolute position

we want to represent **position** in a way that tokens with similar **relative** position have similar **positional encoding**

**Idea:** what if we use **frequency-based** representations?

$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \vdots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$

**Dimensionality of positional encoding**

“even-odd” indicator

“first-half vs. second-half” indicator
Positional encoding: learned

Another idea: just learn a positional encoding

- Different for every input sequence
- The same learned values for every sequence
- but different for different time steps

How many values do we need to learn?

\[ P = [p_1, p_2, \ldots, p_T] \in \mathbb{R}^{d \times T} \]

- + more flexible (and perhaps more optimal) than sin/cos encoding
- + a bit more complex, need to pick a max sequence length (and can’t generalize beyond it)
How to incorporate positional encoding?

At each step, we have $x_t$ and $p_t$

**Simple choice:** just concatenate them  

$$\bar{x}_t = \begin{bmatrix} x_t \\ p_t \end{bmatrix}$$

**More often:** just add after embedding the input

input to self-attention is $\text{emb}(x_t) + p_t$

some learned function (e.g., some fully connected layers with linear layers + nonlinearities)
From Self-Attention to Transformers

The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**.

But to make this actually work, we need to develop a few additional components to address some fundamental limitations:

1. Positional encoding addresses lack of sequence information
2. **Multi-headed attention** allows querying multiple positions at each layer
3. Adding nonlinearities so far, each successive layer is *linear* in the previous one
4. Masked decoding how to prevent attention lookups into the future?

$$a_t = \sum_t \alpha_{l,t} v_t$$

$$v_t = W_v h_t$$
Since we are relying entirely on attention now, we might want to incorporate more than one time step because of softmax, this will be dominated by one value. Hard to specify that you want two different things (e.g., the subject and the object in a sentence).
Multi-head attention

Idea: have multiple keys, queries, and values for every time step!

full attention vector formed by concatenation:

\[
a_2 = \begin{bmatrix}
a_{2,1} \\
a_{2,2} \\
a_{2,3}
\end{bmatrix}
\]

compute weights **independently** for each head

\[
e_{l,t,i} = q_{l,i} \cdot k_{l,i}
\]

\[
\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})
\]

\[
a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}
\]

around 8 heads seems to work pretty well for big models
From Self-Attention to Transformers

The basic concept of self-attention can be used to develop a very powerful type of sequence model, called a transformer. But to make this actually work, we need to develop a few additional components to address some fundamental limitations:

1. Positional encoding addresses lack of sequence information
2. Multi-headed attention allows querying multiple positions at each layer
3. Adding nonlinearities so far, each successive layer is linear in the previous one
4. Masked decoding how to prevent attention lookups into the future?

\[
a_t = \sum_{t} \alpha_{l,t} v_t
\]
\[
v_t = W_v h_t
\]
Self-Attention is **Linear**

Every self-attention "layer" is a linear transformation of the previous layer (with non-linear weights)

This is not very expressive
Alternating self-attention & nonlinearity

Some non-linear (learned) function e.g., $h_t^\ell = \sigma(W^\ell a_t^\ell + b^\ell)$

Just a neural net applied at every position after every self-attention layer!

Sometimes referred to as “position-wise feedforward network”

We’ll describe some specific commonly used choices shortly
The basic concept of **self-attention** can be used to develop a very powerful type of sequence model, called a **transformer**.

But to make this actually work, we need to develop a few additional components to address some fundamental limitations:

1. **Positional encoding** addresses lack of sequence information.
2. **Multi-headed attention** allows querying multiple positions at each layer.
3. **Adding nonlinearities** so far, each successive layer is *linear* in the previous one.
4. **Masked decoding** how to prevent attention lookups into the future?

\[
a_t = \sum_t \alpha_{l,t} v_t \\
v_t = W_v h_t
\]
Self-attention can see the future!

A **crude** self-attention “language model”:

(in reality, we would have many alternating self-attention layers and position-wise feedforward networks, not just one)

**Big problem**: self-attention at step 1 can look at the value at steps 2 & 3, which is based on the inputs at steps 2 & 3

At **test time** (when decoding), the inputs at steps 2 & 3 will be based on the output at step 1...

...which requires knowing the input at steps 2 & 3
Masked attention

A **crude** self-attention “language model”:

At **test time** (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1…

…which requires knowing the **input** at steps 2 & 3

Must allow self-attention into the **past**…

…but not into the **future**

Easy solution:

\[ e_{l,t} = q_l \cdot k_t \]

\[ e_{l,t} = \begin{cases} 
    q_l \cdot k_t & \text{if } l \geq t \\
    -\infty & \text{otherwise}
\end{cases} \]

in practice:

just replace \( \exp(e_{l,t}) \) with 0 if \( l < t \) inside the softmax
We can implement a **practical** sequence model based **entirely** on self-attention.

- Alternate self-attention “layers” with nonlinear position-wise feedforward networks (to get nonlinear transformations).
- Use positional encoding (on the input or input embedding) to make the model aware of relative positions of tokens.
- Use multi-head attention.
- Use masked attention if you want to use the model for decoding.
The Transformer
There are a number of model designs that use successive self-attention and position-wise nonlinear layers to process sequences.

These are generally called “Transformers” because they transform one sequence into another at each layer.

- See Vaswani et al. Attention Is All You Need. 2017

The “classic” transformer (Vaswani et al. 2017) is a sequence to sequence model.

A number of well-known follow works also use transformers for language modeling (BERT, GPT, etc.)
The “classic” transformer

As compared to a sequence to sequence RNN model

self-attention "layer"

position-wise nonlinear network

cross attention

position-wise nonlinear network

masked self-attention

position-wise encoder

position-wise encoder

we’ll discuss how this bit works soon

repeated N times

repeated N times
Combining encoder and decoder values

“Cross-attention”

Much more like the **standard** attention from the previous lecture

**query:** $q^\ell_t = W^\ell_q s^\ell_t$

output of position-wise nonlinear network at (decoder) layer $\ell$, step $t$

**key:** $k^\ell_t = W^\ell_k h^\ell_t$

output of position-wise nonlinear network at (encoder) layer $\ell$, step $t$

**value:** $v^\ell_t = W^\ell_k h^\ell_t$


cross attention output

$$e^\ell_{l,t} = q^\ell_t \cdot k^\ell_t$$

$$\alpha^\ell_{l,t} = \frac{\exp(e^\ell_{l,t})}{\sum_{t'} \exp(e^\ell_{l,t'})}$$

$$c^\ell_l = \sum_t \alpha^\ell_{l,t} v^\ell_t$$

in reality, cross-attention is **also** multi-headed!
One last detail: layer normalization

**Main idea:** batch normalization is very helpful, but hard to use with sequence models

Sequences are different lengths, makes normalizing across the batch hard

Sequences can be very long, so we sometimes have small batches

**Simple solution:** “layer normalization” – like batch norm, but not across the batch

**Batch norm**

```
\mu = \frac{1}{B} \sum_{i=1}^{B} a_i \\
\sigma = \sqrt{\frac{1}{B} \sum_{i=1}^{B} (a_i - \mu)^2} \\
\tilde{a}_i = \frac{a_i - \mu}{\sigma} \gamma + \beta
```

**Layer norm**

```
\mu = \frac{1}{d} \sum_{i=1}^{d} a_j \\
\sigma = \sqrt{\frac{1}{d} \sum_{i=1}^{d} (a_j - \mu)^2} \\
\tilde{a} = \frac{a - \mu}{\sigma} \gamma + \beta
```
Putting it all together

The Transformer

6 layers, each with $d = 512$

$$\tilde{h}^\ell_t = \text{LayerNorm}(\tilde{a}^\ell_t + h^\ell_t)$$ passed to next layer $\ell + 1$

$$h^\ell_t = W^\ell_2 \text{ReLU}(W^\ell_1 \tilde{a}^\ell_t + b^\ell_1) + b^\ell_2$$ 2-layer neural net at each position

$$\tilde{a}^\ell_t = \text{LayerNorm}(\tilde{h}^{\ell-1}_t + a^\ell_t)$$ essentially a residual connection with LN

input: $\tilde{h}^{\ell-1}_t$
output: $a^\ell_t$

concatenates attention from all heads

Decoder decodes one position at a time with masked attention

$$h^\ell_t = W^\ell_2 \text{ReLU}(W^\ell_1 a^\ell_t + b^\ell_1) + b^\ell_2$$ residual connection with LN

multi-head cross attention

same as encoder only masked

Why transformers?

Downsides:
- Attention computations are technically $O(n^2)$
- Somewhat more complex to implement (positional encodings, etc.)

Benefits:
+ Much better long-range connections
+ Much easier to parallelize
+ In practice, can make it much deeper (more layers) than RNN

The benefits seem to **vastly** outweigh the downsides, and transformers work **much** better than RNNs (and LSTMs) in many cases.

Arguably one of the most important sequence modeling improvements of the past decade.
Why transformers?

In practice, this means we can use larger models for the same cost

larger model = better performance

much faster training

great translation results

previous state of the art seq2seq model

text summarization

lower is better (this metric is similar to 1/likelihood)

We’ll learn more about the power of transformers as language models next time!
