Lecture 23: "Attention is All You Need"

Mark Hasegawa-Johnson

ECE 537, Fall 2022

Outline

Attention

$$\text{Context Vector:} \quad c(q^i) = \sum_{j=1}^{n} \alpha_{i,j} v^j$$

$$\text{Attention:} \quad \alpha_{i,j} = \frac{\exp\left(\text{Similarity}(q^i, k^j)\right)}{\sum_{j=1}^{n} \exp\left(\text{Similarity}(q^i, k^j)\right)}$$

- The query, q (sometimes q^i), is the vector whose context we want
- The key, k (sometimes k^j), tells us whether or not v^j is useful context
- The value, v (sometimes v^j), provides the actual context

Inter-Attention

- The query, q (sometimes q^i), is the vector whose context we want
  - For example, the previous layer or $(i-1)^{\mathrm{st}}$ time-step of the decoder
- The key, k (sometimes k^j), tells us whether or not v^j is useful context
  - For example, something computed from the j^{th} timestep of the encoder
- The value, v (sometimes v^j), provides the actual context
  - For example, something computed from the j^{th} timestep of the encoder

Self-Attention

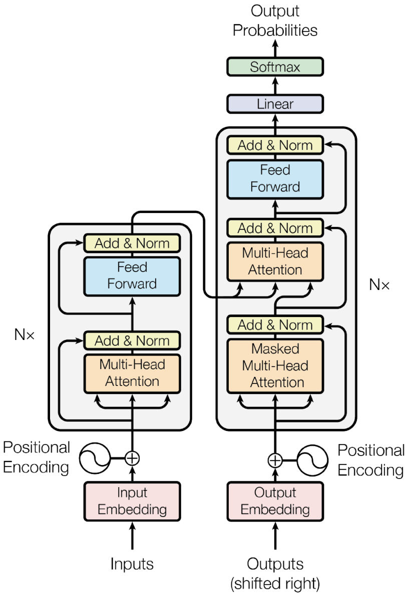
- The query, q (sometimes q^i), is the vector whose context we want
  - For example, something computed from the i^{th} timestep of the encoder
- The key, k (sometimes k^j), tells us whether or not v^j is useful context
  - For example, something computed from the j^{th} timestep of the encoder
- The value, v (sometimes v^j), provides the actual context
  - For example, something computed from the j^{th} timestep of the encoder

Difference Between Attention and Self-Attention

- Attention computes the context vector $c(q)$ by summarizing information from a different data source (e.g., encoder vectors providing context for a decoder vector).
- Self-attention computes the context vector $c(q)$ by summarizing temporally-distant information from the same data source (e.g., encoder vectors providing context for an encoder vector).

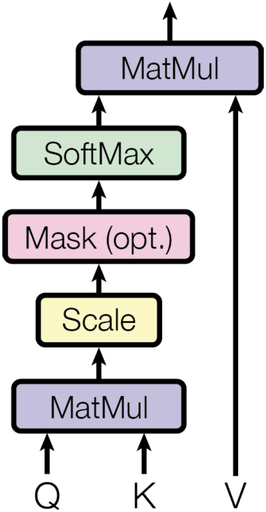
Outline

The Transformer



Outline

Scaled Dot-Product Attention



Vaswani et al., 2017, Figure 2(a)

The Data Matrices

$$Q = \begin{bmatrix} q^1 \\ \vdots \\ q^n \end{bmatrix}, \quad K = \begin{bmatrix} k^1 \\ \vdots \\ k^n \end{bmatrix}, \quad V = \begin{bmatrix} v^1 \\ \vdots \\ v^n \end{bmatrix}$$

- $q = q^i \in \Re^{d_k}$ is a query vector
- $k = k^j \in \Re^{d_k}$ is a key vector
- $v = v^j \in \Re^{d_v}$ is a value vector

The Dot-Product

$$QK^T = \begin{bmatrix} q^1 k^{1,T} & \cdots & q^1 k^{n,T} \\ \vdots & \ddots & \vdots \\ q^n k^{1,T} & \cdots & q^n k^{n,T} \end{bmatrix},$$

is the matrix whose $(i,j)^{\mathrm{th}}$ element is the dot product between q^i and k^j.

The Scaled Dot-Product

Suppose that q^i and k^j are each normalized so that they are independent Gaussian random variables with zero mean and unit variance. Then

$$q^i k^{j,T} = \sum_{t=1}^{d_k} q_t^i k_t^j$$

is a difference of chi-squared random variables, with zero mean and variance d_k. We can re-normalize it (to zero mean and unit variance) by computing

$$\frac{q^i k^{j,T}}{\sqrt{d_k}} = \frac{1}{\sqrt{d_k}} \sum_{t=1}^{d_k} q_t^i k_t^j$$

Scaled Dot-Product Attention

We assume that q and k have been transformed by some preceding neural net, so qk^T is large if and only if they should be considered similar. Therefore the similarity score is

$$e_{i,j} = \frac{1}{\sqrt{d_k}} q^i k^{j,T},$$

and the corresponding attention weight is

$$\alpha_{i,j} = \text{softmax}(e_{i,j}) = \frac{\exp(e_{i,j})}{\sum_{j=1}^{n} \exp(e_{i,j})}$$

$$\begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,n} \end{bmatrix} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Scaled Dot-Product Attention

The context summary vector is then

$$c(q^i) = \sum_{j=1}^{n} \alpha_{i,j} v^j$$

If we stack these up into a matrix, we get

$$\begin{bmatrix} c(q^1) \\ \vdots \\ c(q^n) \end{bmatrix} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \begin{bmatrix} v^1 \\ \vdots \\ v^n \end{bmatrix}$$

Masking

If q, k and v are decoder vectors being produced autoregressively (e.g., decoder self-attention), then $c(q^i)$ can only depend on values of v^j for $j < i$:

$$c(q^i) = \sum_{j=1}^{i-1} \alpha_{i,j} v^j$$

This can be done by setting $\alpha_{i,j} = 0$ for $j \geq i$. In turn, this can be done by masking the similarity scores as follows:

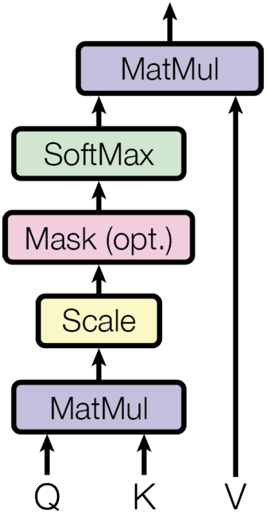
$$e_{i,j} = \frac{1}{\sqrt{d_k}} q^i k^{j,T} + m_i^j,$$

where

$$m_i^j = \begin{cases} 0 & j < i \\ -\infty & j \geq i \end{cases}$$

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$



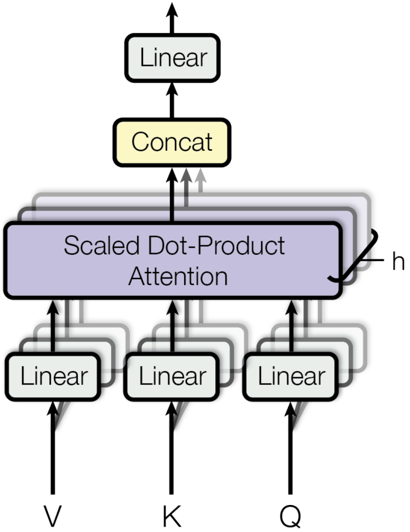
Vaswani et al., 2017, Figure 2(a)

Outline

Multi-Head Attention: Why

- Dot-product attention assumes that q^i and k^j have already been transformed by some neural network so that $q^i k^{j,T}$ is large if and only if v^j is an important part of the context.
- What if you need several types of context? One type tells you about speaker ID, one type tells you about dialect, one type tells you the topic of conversation, etc.
- Multi-Head Attention computes many different types of q vectors, and many different types of k vectors, so that different types of context may be accumulated in parallel.

Multi-Head Attention



Vaswani et al., 2017, Figure 2(b)

Multi-Head Attention

$$\begin{aligned}\text{head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \\ &= \text{softmax}\left(\frac{QW_i^Q W_i^{K,T} K^T}{\sqrt{d_k}}\right) VW_i^V,\end{aligned}$$

where the weight matrices W_i^Q, W_i^K, and W_i^V, for $1 \leq i \leq h$, are learned matrices summarizing the type of context accumulated in each head. Then

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O,$$

where W^O is a final transformation that can, e.g., combine information from different heads in a learned manner.

Outline

Why Self-Attention?

- Encoder-decoder attention is well-established, but the transformations that compute q and k can be (1) convolutional, (2) recurrent, or (3) self-attention. When is self-attention the best approach?
- Recurrent networks have to propagate information from the start of the sequence to the end (path length$=n$). Information can get forgotten.
- Convolutional networks are much quicker, but need to learn weights covering the entire width of the kernel (k). For reasons of data-efficient learning, most systems therefore use small k.
- Self-attention is as fast as convolution, without pre-trained kernel weights. Instead, the attention weights are based on similarity, which is computed using a more efficient network. Therefore, the "kernel width" for self-attention is usually $k = n$.

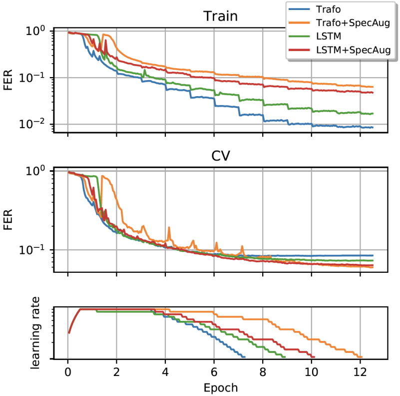
Why Self-Attention?

Layer Type	Complexity/Layer	Path Length
Recurrent	$O\{nd^2\}$	$O\{n\}$
Convolutional	$O\{knd^2\}$	$O\{\log_k(n)\}$
Self-Attention	$O\{n^2d\}$	$O\{1\}$

- n = sequence length
- d = representation dimension
- k = kernel dimension

Outline

- Because of the shorter pathlength, Transformer trains faster than LSTM.
- Transformer sometimes overtrains (time alignment is too flexible).
- Overtraining can be compensated by data augmentation, giving it exactly the same accuracy as LSTM.



Zeyer et al., "A Comparison of Transformer and LSTM Encoder Decoder Models for ASR," (c) IEEE, 2019

Summary

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O,$$