Lecture 21: Transformer

Mark Hasegawa-Johnson
These slides are in the public domain

University of Illinois

ECE 417: Multimedia Signal Processing

Outline

The Cauchy-Schwartz Inequality

The Cauchy-Schwart inequality says that, for any two vectors $\boldsymbol{x} = [x_1, \ldots, x_N]^T$ and $\boldsymbol{y} = [y_1, \ldots, y_N]^T$,

$$|\boldsymbol{x}^T \boldsymbol{y}| \le \|\boldsymbol{x}\| \|\boldsymbol{y}\|$$

If we define the unit vectors as follows,

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}, \quad \hat{\boldsymbol{y}} = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|},$$

then the Cauchy-Schwartz inequality says that

$$-1 \le \hat{\boldsymbol{x}}^T \hat{\boldsymbol{y}} \le 1$$

The Cauchy-Schwartz Inequality: Proof

Suppose we have a particular $\boldsymbol{x}$, and we want to:

- choose $y_1, \ldots, y_N$ in order to maximize the dot product,

$$\boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{N} x_i y_i$$

- subject to the constraint that the length of $\boldsymbol{y}$ is fixed, say,

$$1 = \sum_{i=1}^{N} y_i^2$$

This can be done by choosing $y_1, \ldots, y_N$ to maximize the following Lagrangian:

$$\mathcal{L}(\boldsymbol{y}) = \sum_{i=1}^{N} x_i y_i + \lambda \left(\sum_{i=1}^{N} y_i^2 - 1 \right)$$

The Cauchy-Schwartz Inequality: Proof

$$\mathcal{L}(\boldsymbol{y}) = \sum_{i=1}^{N} x_i y_i + \lambda \left(\sum_{i=1}^{N} y_i^2 - 1 \right)$$

Setting $\frac{\partial \mathcal{L}}{\partial y_i} = 0$ yields:

$$y_i = -\frac{x_i}{2\lambda},$$

There are two values of λ that satisfy the constraint $\|\boldsymbol{y}\| = 1$. They are:

$$y_i = \pm \frac{x_i}{\|\boldsymbol{x}\|}$$

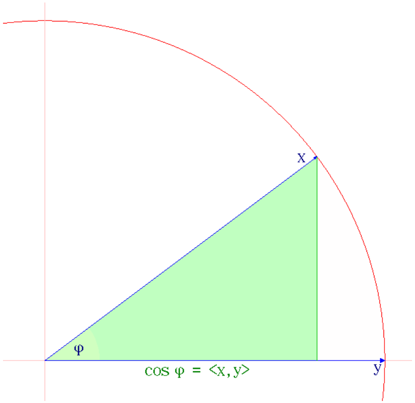
Cosine Distance

The Cauchy-Schwartz inequality can be written as:

$$-1 \leq \hat{\boldsymbol{x}}^T \hat{\boldsymbol{y}} \leq 1,$$

where $\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$ and $\hat{\boldsymbol{y}} = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|}$. This is an N-dimensional generalization of the 2D geometric interpretation of the dot product:

$$\hat{\boldsymbol{x}}^T \hat{\boldsymbol{y}} = \cos \phi$$



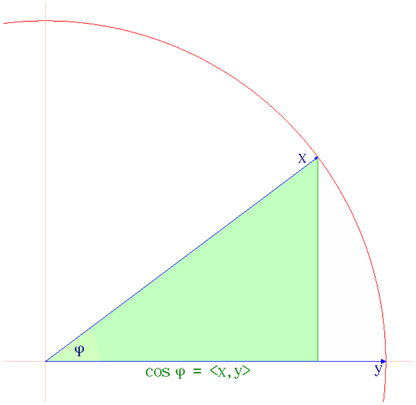
CC-SA 4.0, https://commons.wikimedia.org/wiki/File:Cauchy-Schwarz_inequation_in_Euclidean_plane.gif

Cosine Distance

Large-magnitude vectors have a tendency to swamp the training criterion for a neural net. It's often useful to explicitly ignore the magnitude of the vector, and to only minimize the angle between two vectors on the $(N-1)$-dimensional hypersphere. This is done by minimizing the **cosine distance**,

$$\begin{aligned}\text{cosd}(\boldsymbol{x},\boldsymbol{y}) &= 1-\cos(\boldsymbol{x},\boldsymbol{y})\\ &= 1-\hat{\boldsymbol{x}}^T\hat{\boldsymbol{y}}\end{aligned}$$

Which is equivalent to maximizing the **cosine similarity**, $\hat{\boldsymbol{x}}^T\hat{\boldsymbol{y}}$.



CC-SA 4.0,

https://commons.wikimedia.org/wiki/File:

Cauchy-Schwarz_inequation_in_Euclidean_plane.gif

Outline

PCA and Smart PCA

Consider trying to find a set of vectors, $\mathbf{w}_k$ $(1 \leq k \leq K)$, in order to minimize

$$\text{MSE} = E\left[\|\mathbf{x} - \sum_{k=1}^{K} h_k \mathbf{w}_k\|^2\right],$$

where expectation is over the training dataset. The minimum-MSE solution has orthogonal vectors, and therefore $h_k = \frac{\mathbf{w}_k^T \mathbf{x}}{\|\mathbf{w}_k\|^2}$ is the minimum-MSE weight.

- The MMSE solution can be computed using principal components analysis (PCA).
- The MMSE solution can also be computed using an autoencoder neural network. If K is much less than the vector dimension, the autoencoder computation is faster, so this is called "smart PCA" (SPCA).

Consider a training database, $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. An autoencoder computes

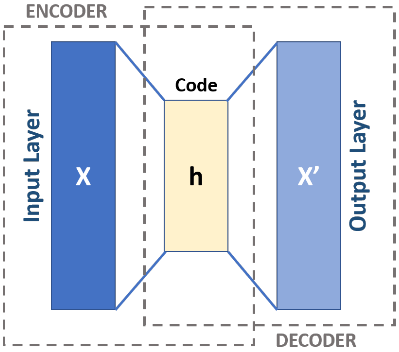
$$h_{i,k} = \boldsymbol{x}_i^T \boldsymbol{w}_k,$$

and

$$\boldsymbol{x}_i' = \sum_{k=1}^{K} h_{i,k} \boldsymbol{w}_k = \boldsymbol{W}\boldsymbol{h}$$

then trains $\boldsymbol{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K]$ to minimize

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{x}_i'\|^2$$



CC-SA 4.0,

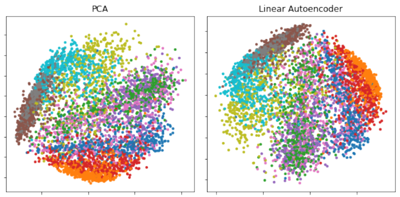
`https://commons.wikimedia.org/wiki/File:`

`Autoencoder_schema.png`

Both PCA and SPCA choose unit-length vectors, $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K]$ in order to minimize

$$\text{MSE} = E\left[\|\mathbf{x} - \mathbf{W}\mathbf{h}\|^2\right],$$

therefore both SPCA and PCA choose vectors that span the same vector subspace. SPCA does not decide the order of the vectors, or their sign, so SPCA can produce a vector subspace that's a rotated version of the one chosen by PCA.



Plot of the first two Principal Components (left) and the two hidden units' values of a Linear Autoencoder (right) applied to the Fashion MNIST dataset. The two models being both linear learn to span the same subspace.

CC-SA 4.0,

`https://upload.wikimedia.org/wikipedia/commons/0/0b/PCA_vs_Linear_Autoencoder.png`

PCA and Smart PCA

The training criterion for both PCA and SPCA is

$$\begin{aligned}\text{MSE} &= E\left[\|\boldsymbol{x}-\boldsymbol{W}\boldsymbol{h}\|^2\right]\\ &= E\left[\|\boldsymbol{x}\|^2+\|\boldsymbol{W}\boldsymbol{h}\|^2-2\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{h}\right]\end{aligned}$$

So minimizing the MSE is the same as maximizing the dot product between the hidden vector, $\boldsymbol{h}$, and the transformed input vector, $\boldsymbol{W}^T\boldsymbol{x}$. This is a kind of dot-product similarity measure.

Outline

ASR and Machine Translation (MT): Similarities and Differences

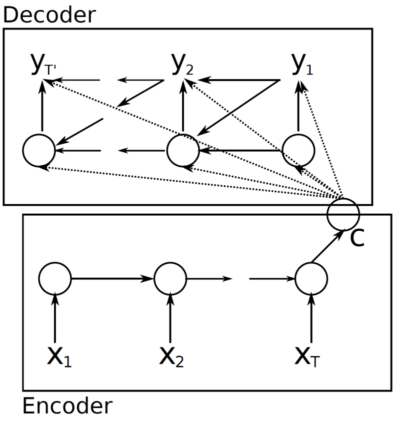
The idea of an “attention-weighted summary” was proposed and refined from 2012 to 2017 in a series of papers, some of which were automatic speech recognition (ASR) papers, and some of which were neural machine translation (MT) papers.

	Input, Output Lengths	Input, Output Sequence Order
ASR	Different	Same
MT	Different	Different

Encoder-Decoder Neural Nets

Cho et al., "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation" (Sep. 2014) proposed a solution:

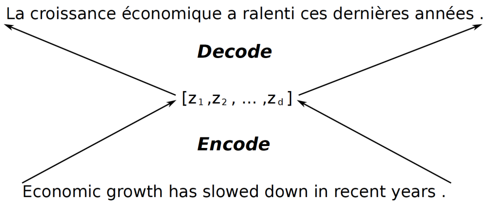
- RNN "encodes" all of the input to a summary vector, $\boldsymbol{c}$, then
- RNN "decodes" $\boldsymbol{c}$ in order to produce the output.



Cho, Merriënboer, Gulcehre, Bahdenau, Bougares, Schwenk & Bengio, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, Fig. 1.

Encoder-Decoder Neural Nets

Cho et al., “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches” (Oct. 2014) proposed that the encoder summary didn’t need to be a single vector, it could be a sequence of vectors.


La croissance économique a ralenti ces dernières années .
Decode
[z₁ ,z₂ , ... ,z_d]
Encode
Economic growth has slowed down in recent years .

Cho, Merriënboer, Bahdenau & Bengio, On the Properties of Neural Machine Translation: Encoder–Decoder Approaches, Fig. 3.

Encoder-Decoder: Too Much Information?

- Encoder-decoder models beat the state of the art in some tasks (usually tasks with a lot of data), but had a fatal flaw.
- If the encoder creates a small summary, then it accidentally throws away important information, because it doesn't always know which information is important.
- If the encoder creates a large summary, then the decoder doesn't know which data to use for any given computation, so training takes longer, and sometimes fails.

Attention

In December 2014, Chorowski et al. proposed a new type of encoder-decoder algorithm, based on "attention."

- The i^{th} time step in the **Decoder** is computed based on an attention-weighted summary of the inputs:

$$\boldsymbol{s}_i = \text{Recurrency}\left(\boldsymbol{s}_{i-1}, \sum_{j=1}^{L} \alpha_{i,j}\boldsymbol{h}_j\right)$$

- The **Attention** is a kind of probability mass (sums to one) over the different input time steps, $1 \le j \le L$:

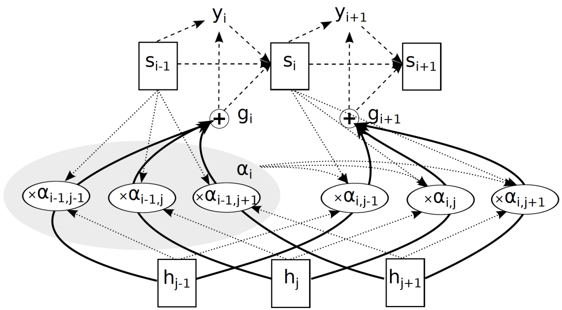
$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{j=1}^{L}\exp(e_{i,j})}, \quad \sum_{j=1}^{L}\alpha_{i,j} = 1$$

- The **Attention Score** is a special neural net that measures the similarity between input vector h_j and output vector s_{i-1}:

$$e_{i,j} = \text{Score}(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)$$

Attention

- $\boldsymbol{s}_{i-1}$ and $\boldsymbol{h}_j$ determine $\alpha_{i,j}$
- $\boldsymbol{s}_i$ is determined by $\sum_{j=1}^{L} \alpha_{i,j} \boldsymbol{h}_j$



Chorowski, Bahdanau, Serdyk, Cho & Bengio, Attention-Based Models for Speech Recognition, Fig. 1

When should you pay attention?

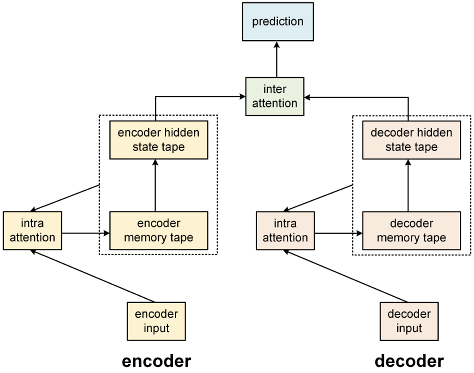
- The key idea of attention is that there is some sort of similarity score, $e_{i,j} = \text{Similarity}(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)$, so that you can compute attention weights according to

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

- Raw inputs (speech) and raw outputs (text) are not inherently similar.
- There needs to be a network that converts the inputs into some hidden vectors, $\boldsymbol{h}_j$ and $\boldsymbol{s}_{i-1}$, that are similar if and only if $\alpha_{i,j}$ should be large.

Cheng, Dong and Lapata, "Long Short-Term Memory-Networks for Machine Reading," proposed using a new kind of attention called "intra-attention" or "self-attention" to compute

- $\boldsymbol{h}_j$ from the inputs, and
- $\boldsymbol{s}_{i-1}$ from the preceding outputs, so that
- inter-attention will correctly measure Similarity$(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)$.



Cheng, Dong & Lapata, "Long Short-Term Memory-Networks for Machine Reading," 2016, Fig. 3(a)

Intra-Attention

An RNN with intra-attention computes the RNN state vector $\tilde{h}_t$ as the attention-weighted summary of past RNN state vectors:

$$\tilde{\boldsymbol{h}}_t = \sum_{i=1}^{t-1} \alpha_{t,i} \boldsymbol{h}_i,$$

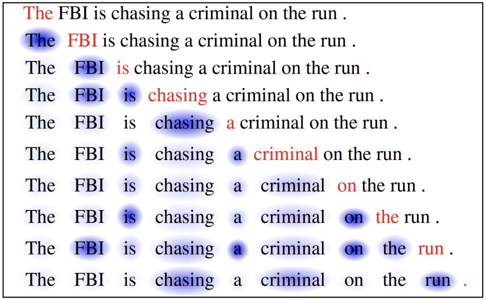
where the weights, $\alpha_{t,i}$, are softmax normalized:

$$\alpha_{t,i} = \mathrm{softmax}(e_{t,i}),$$

based on similarities computed between $\boldsymbol{h}_i$, $\tilde{\boldsymbol{h}}_{t-1}$, and the input vector $\boldsymbol{x}_t$:

$$e_{t,i} = \boldsymbol{w}_e^T \tanh\left(\boldsymbol{W}_h \boldsymbol{h}_i + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_{\tilde{h}} \tilde{\boldsymbol{h}}_{t-1}\right)$$

- The representation of each word (in red) is computed based on. . .
- an attention-weighted summary of all previous words (attention weights in blue).
- Thus, the meaning of a word depends on its context.

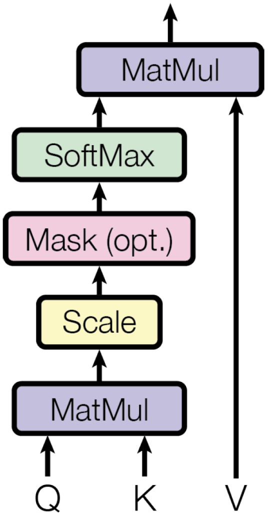


The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .

Cheng, Dong & Lapata, "Long Short-Term Memory-Networks for Machine Reading," 2016, Fig. 1

Outline

Transformer: Scaled Transformed Dot-Product Attention



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

The Data Matrices

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_1^T \\ \vdots \\ \boldsymbol{q}_n^T \end{bmatrix}, \quad \boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1^T \\ \vdots \\ \boldsymbol{k}_n^T \end{bmatrix}, \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_n^T \end{bmatrix}$$

- $\boldsymbol{q}_i \in \Re^{d_k}$ is a query vector
- $\boldsymbol{k}_j \in \Re^{d_k}$ is a key vector
- $\boldsymbol{v}_j \in \Re^{d_v}$ is a value vector

The Dot-Product

$$\boldsymbol{QK}^T = \begin{bmatrix} \boldsymbol{q}_1^T\boldsymbol{k}_1 & \cdots & \boldsymbol{q}_1^T\boldsymbol{k}_n \\ \vdots & \ddots & \vdots \\ \boldsymbol{q}_n^T\boldsymbol{k}_1 & \cdots & \boldsymbol{q}_n^T\boldsymbol{k}_n \end{bmatrix},$$

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

The Scaled Dot-Product

Suppose that $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$ are each normalized so that they are independent Gaussian random variables with zero mean and unit variance. Then

$$\boldsymbol{q}_i^T \boldsymbol{k}_j = \sum_{t=1}^{d_k} q_{i,t} k_{j,t}$$

is a zero-mean random variable with variance d_k. We can re-normalize it (to zero mean and unit variance) by computing

$$\frac{\boldsymbol{q}_i^T \boldsymbol{k}_j}{\sqrt{d_k}} = \frac{1}{\sqrt{d_k}} \sum_{t=1}^{d_k} q_{i,t} k_{j,t}$$

Scaled Dot-Product Attention

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

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

and the corresponding attention weight is

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

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

Scaled Dot-Product Attention

The context summary vector is then

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

If we stack these up into a matrix, we get

$$\begin{bmatrix} \boldsymbol{c}(\boldsymbol{q}_1)^T \\ \vdots \\ \boldsymbol{c}(\boldsymbol{q}_n)^T \end{bmatrix} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right) \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_n^T \end{bmatrix}$$

Masking

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

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

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:

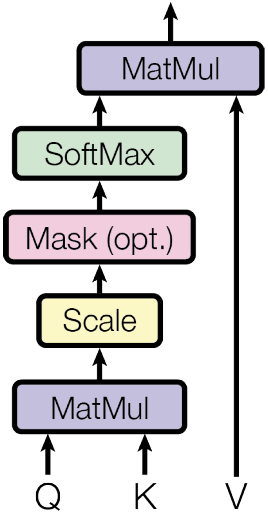
$$e_{i,j} = \frac{1}{\sqrt{d_k}} \boldsymbol{q}_i^T \boldsymbol{k}_j + m_{i,j},$$

where

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

Scaled Dot-Product Attention

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



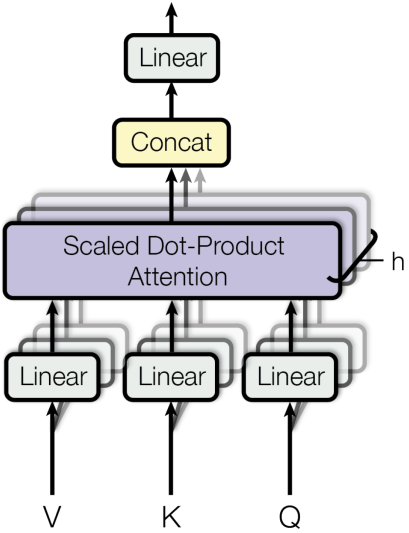
Vaswani et al., 2017, Figure 2(a)

Outline

Multi-Head Attention: Why

- Dot-product attention assumes that $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$ have already been transformed by some neural network so that $\boldsymbol{q}_i^T\boldsymbol{k}_j$ is large if and only if $\boldsymbol{v}_j$ 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 $\boldsymbol{q}_i$ vectors, and many different types of $\boldsymbol{k}_j$ 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

$$\begin{aligned} \text{head}_i &= \text{Attention}\left(\boldsymbol{QW}_i^Q, \boldsymbol{KW}_i^K, \boldsymbol{VW}_i^V\right) \\ &= \text{softmax}\left(\frac{\boldsymbol{QW}_i^Q(\boldsymbol{KW}_i^K)^T}{\sqrt{d_k}}\right)\boldsymbol{VW}_i^V, \end{aligned}$$

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

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

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

Outline

Why Self-Attention?

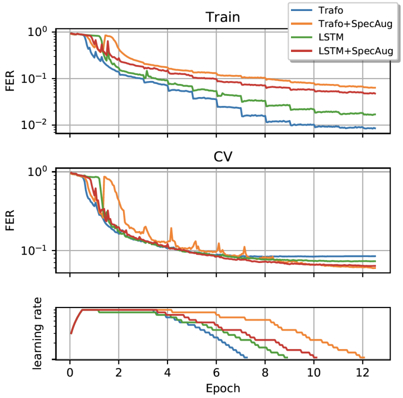
- Encoder-decoder attention is well-established, but the transformations that compute $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$ can be (1) convolutional, (2) recurrent, or (3) self-attention. When is self-attention the best approach?
- Recurrent networks have to propagate information from the start of 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	$O\{n^2 d\}$	$O\{1\}$

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

- 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

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

$$\text{head}_i = \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right)$$

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