Introduction to Deep Learning

Attention Mechanisms and Transformers

Andres Mendez-Vazquez

July 2, 2025

Outline

Outline

The Attention Mechanism

An inspiration, Natural Language Processing

The Attention Mechanism

An inspiration, Natural Language Processing

Basically attention could be seen as vector of importance

- For example, when you are putting attention in eating, we have

$$\begin{bmatrix} 0 & 0.1 & 0.8 & 0.1 & 0.2 & 0.5 \\ He & is & eating & a & toasted & ham \end{bmatrix}$$

The Attention Mechanism

An inspiration, Natural Language Processing

Basically attention could be seen as vector of importance

- For example, when you are putting attention in eating, we have

$$\begin{bmatrix} 0 & 0.1 & 0.8 & 0.1 & 0.2 & 0.5 \\ He & is & eating & a & toasted & ham \end{bmatrix}$$

Thus, how do we predict ham to be important in **eating**

- Therefore, we have that $\frac{0.8+0.5}{2} = 0.65$

Outline

And Back of NLP

There was a model trying to use attention before

- The seq2seq model was born in the field of language modeling [1]
  - One of the Co-founders of **Open AI** - Ilya Sutskever

And Back of NLP

There was a model trying to use attention before

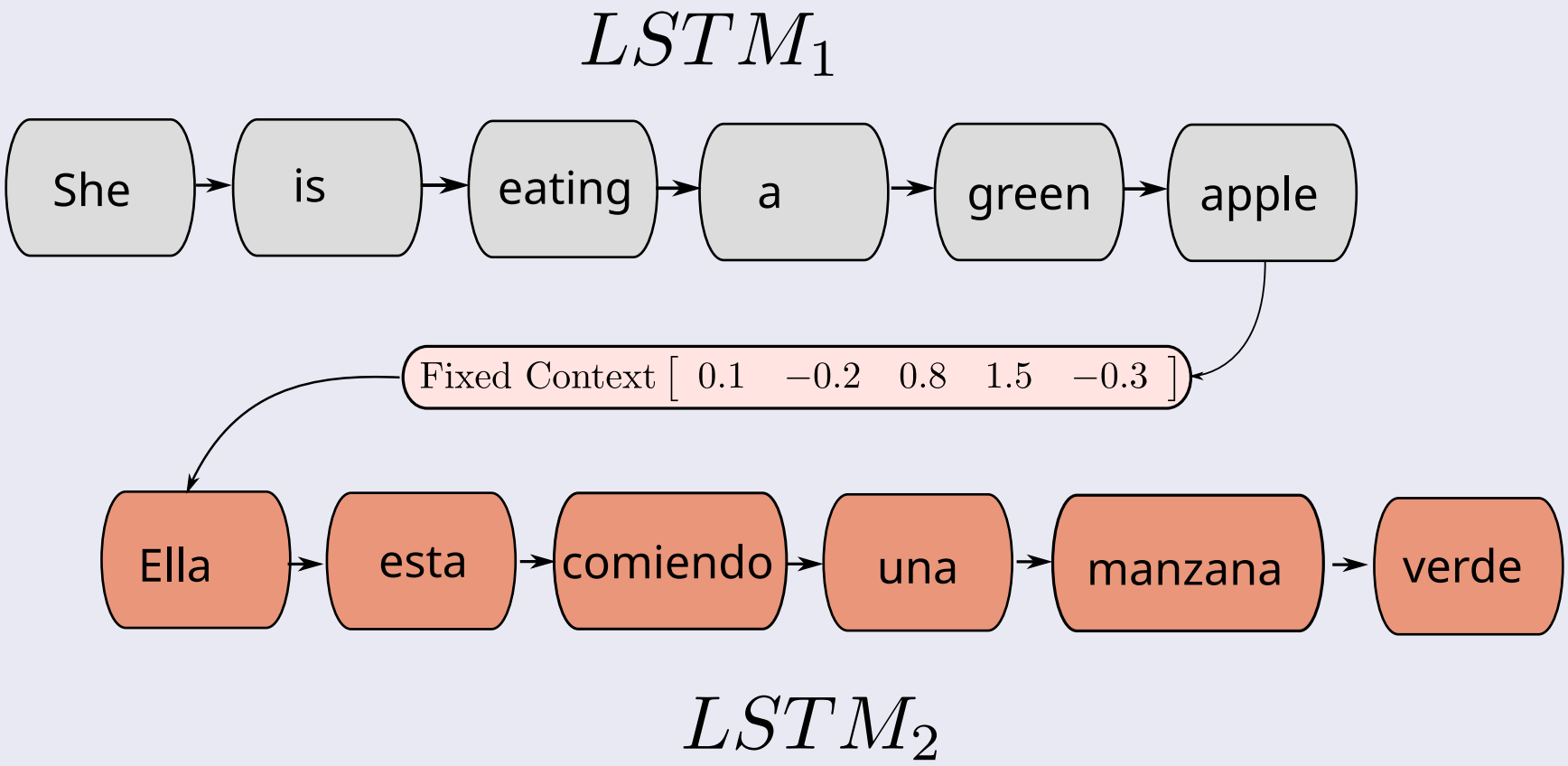
- The seq2seq model was born in the field of language modeling [1]
  - One of the Co-founders of **Open AI** - Ilya Sutskever

He observed the following

- "Despite their flexibility and power, DNNs can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality."

Thus, the LSTM proposal

Two LSTM and a kind of sequential optimization



A Problem with this model

A Problem with long sequences

- Long sentences are not remembered by the model.

A Problem with this model

A Problem with long sequences

- Long sentences are not remembered by the model.

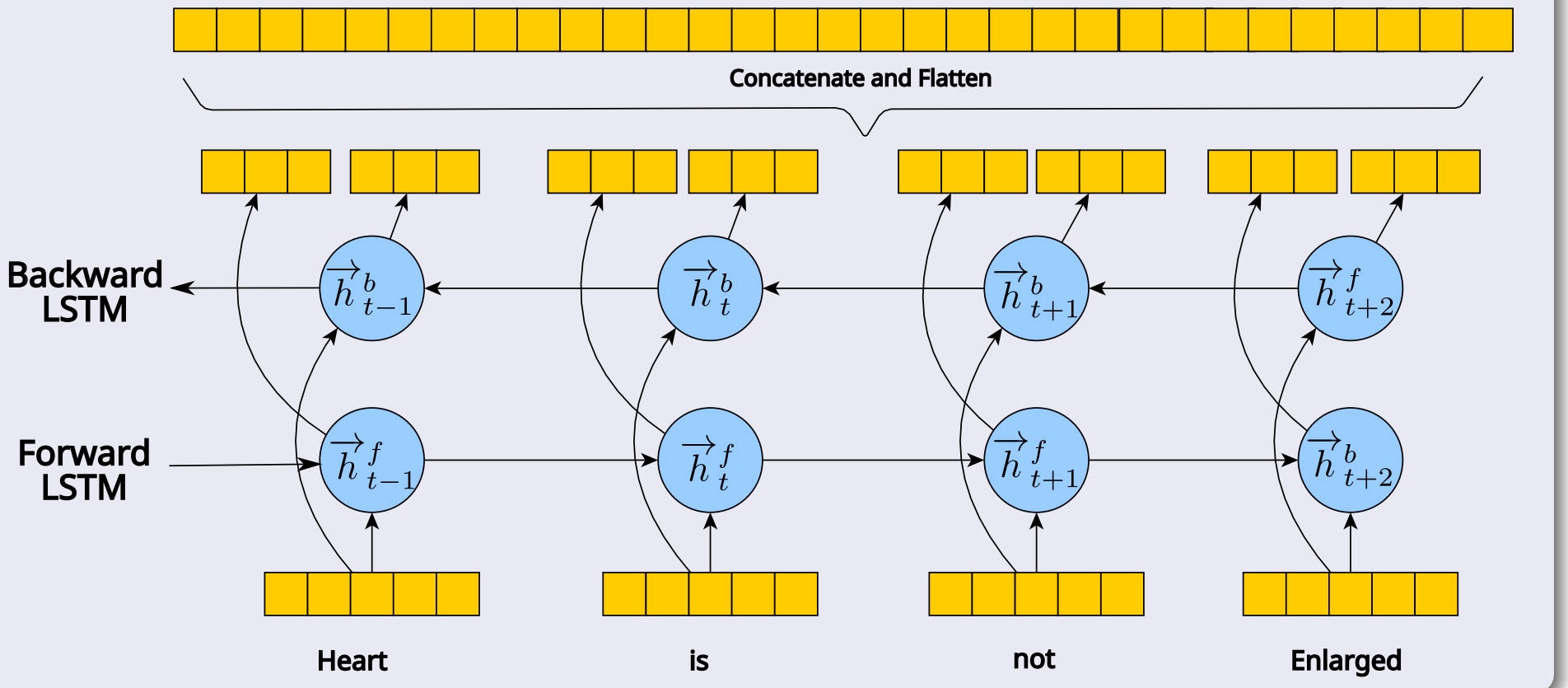
Then, Bahdanau et al., 2014 [2]

- Introduced the idea of attention to solve this problems in LSTM

Outline

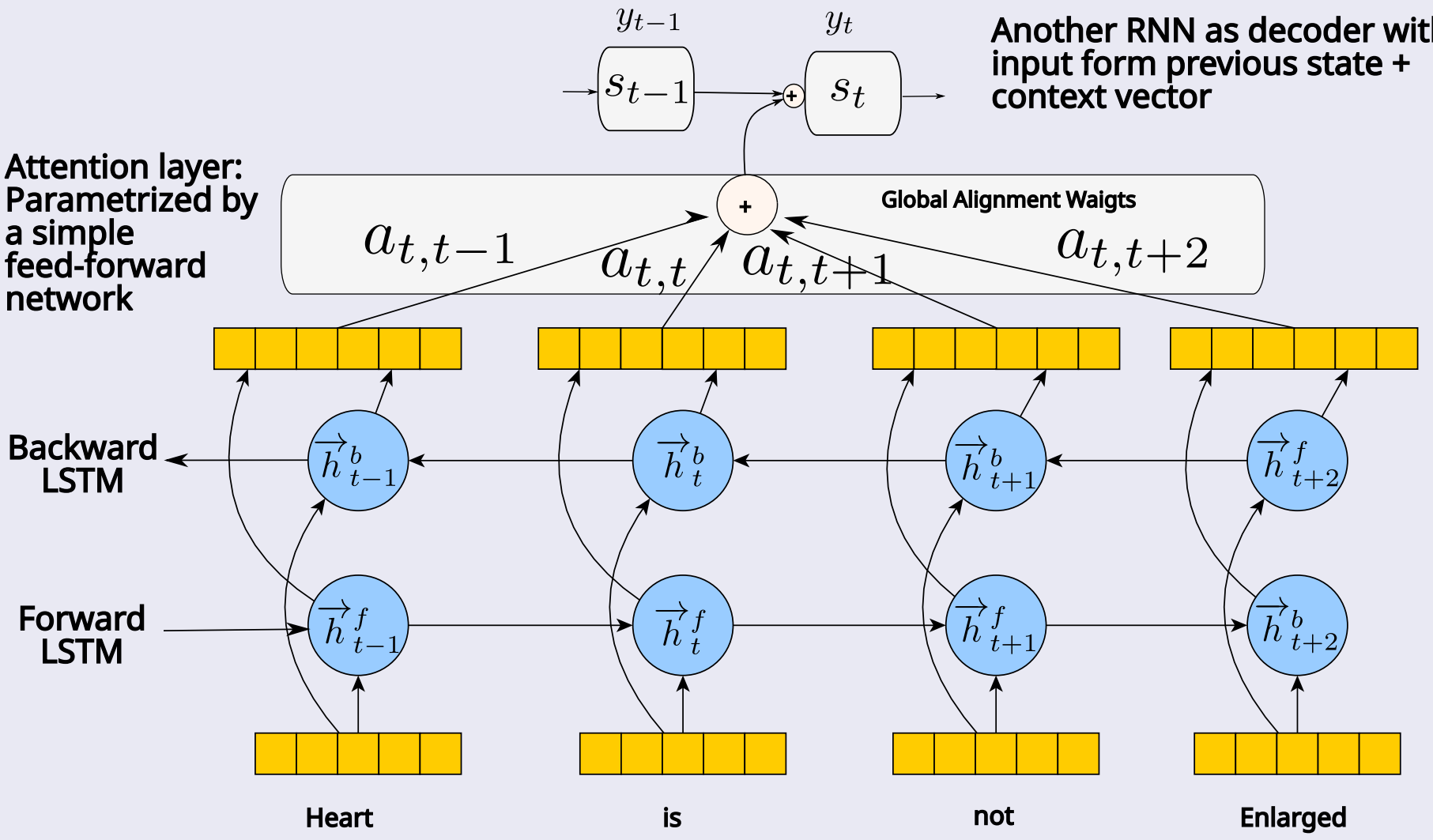
The proposal uses as basis

The Bidirectional LSTM



Then, they proposed an attention layer

How this works?



Outline

First Steps

We have the following idea

- Say we have two sequences, source $\boldsymbol{x}$ of lenght n and target $\boldsymbol{y}$ of lenght m:

$$\boldsymbol{x} = [x_1, x_2, ..., x_n]$$
$$\boldsymbol{y} = [y_1, y_2, ..., x_m]$$

First Steps

We have the following idea

- Say we have two sequences, source $\boldsymbol{x}$ of lenght n and target $\boldsymbol{y}$ of lenght m:

$$\boldsymbol{x} = [x_1, x_2, ..., x_n]$$
$$\boldsymbol{y} = [y_1, y_2, ..., x_m]$$

Thus, we have the classic encoder$\longleftrightarrow$decoder architecture

- The encoder is a bidirectional LSTM
  - Here we have a forward hidden state vector $\overrightarrow{\boldsymbol{h}}_i$
  - Also a backward hidden state vector $\overleftarrow{\boldsymbol{h}}_i$

First Steps

We have the following idea

- Say we have two sequences, source $\boldsymbol{x}$ of lenght n and target $\boldsymbol{y}$ of lenght m:

$$\boldsymbol{x} = [x_1, x_2, ..., x_n]$$
$$\boldsymbol{y} = [y_1, y_2, ..., x_m]$$

Thus, we have the classic encoder$\longleftrightarrow$decoder architecture

- The encoder is a bidirectional LSTM
  - Here we have a forward hidden state vector $\overrightarrow{\boldsymbol{h}}_i$
  - Also a backward hidden state vector $\overleftarrow{\boldsymbol{h}}_i$

So we can generate a vector

$$\boldsymbol{h}_i = \left[\overrightarrow{\boldsymbol{h}}_i^T; \overleftarrow{\boldsymbol{h}}_i^T\right]^T, \ i = 1, ..., n$$

At the Decoder

The decoder network has hidden state

- $\boldsymbol{s}_t = f\left(\boldsymbol{s}_{t-1}, y_{t-1}, \underbrace{\boldsymbol{c}_t}_{context\ vector}\right)$ for the output word at position $t = 1, ..., m$

At the Decoder

The decoder network has hidden state

- $\boldsymbol{s}_t = f\left(\boldsymbol{s}_{t-1}, y_{t-1}, \underbrace{\boldsymbol{c}_t}_{context\ vector}\right)$ for the output word at position $t = 1, ..., m$

The context vector $\boldsymbol{c}_t$

- The context vector $\boldsymbol{c}_t$ is a sum of hidden states of the input sequence, weighted by alignment scores

$$\boldsymbol{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \text{ (Context vector for output } y_t)$$

At the Decoder

The decoder network has hidden state

- $\boldsymbol{s}_t = f\left(\boldsymbol{s}_{t-1}, y_{t-1}, \underbrace{\boldsymbol{c}_t}_{context\ vector}\right)$ for the output word at position $t = 1, ..., m$

The context vector $\boldsymbol{c}_t$

- The context vector $\boldsymbol{c}_t$ is a sum of hidden states of the input sequence, weighted by alignment scores

$$\boldsymbol{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \left(\text{Context vector for output } y_t\right)$$

Here, we use $\alpha_{t,i}$ to see how well two words y_t and x_i are aligned

$$\alpha_{t,i} = align\left(y_t, x_i\right) = \frac{\exp\left\{score\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right)\right\}}{\sum_{j=1}^{n} \exp\left\{score\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j\right)\right\}}$$

The $score$ function

The alignment model assigns a score $\alpha_{t,i}$

- to to the pair of input at position i and output at position t

The $score$ function

The alignment model assigns a score $\alpha_{t,i}$

- to to the pair of input at position i and output at position t

This score is encoded in a feed-forward network

- with a single hidden layer and this network is jointly trained with other parts of the model.

The *score* function

The alignment model assigns a score $\alpha_{t,i}$

- to to the pair of input at position i and output at position t

This score is encoded in a feed-forward network

- with a single hidden layer and this network is jointly trained with other parts of the model.

Thus, the score function looked as

$$score\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right) = \boldsymbol{v_a^T} \tanh\left(W_a\left[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i\right]\right)$$

- where v_a^T and W_a are weight matrices to be learned in the alignment model.

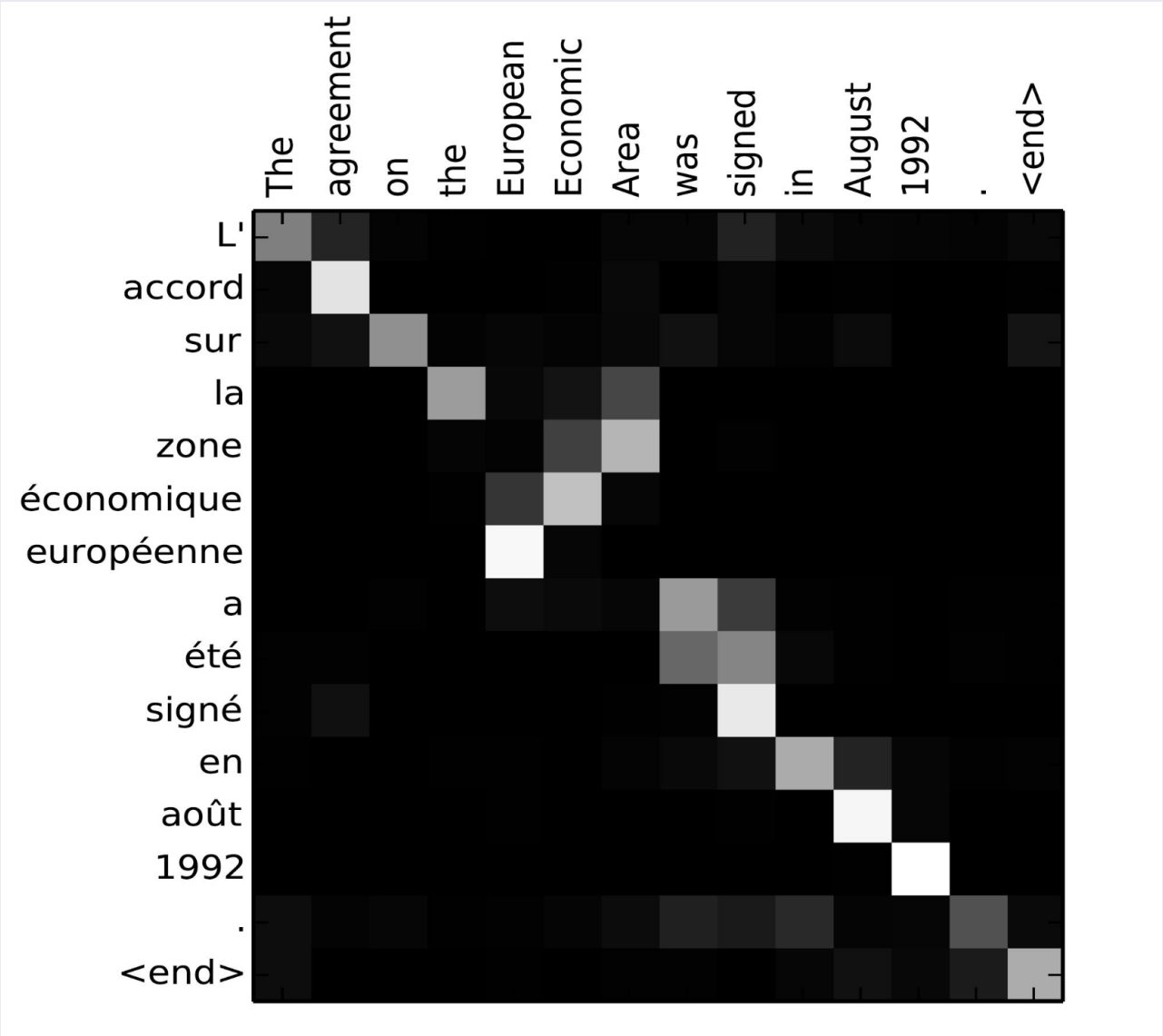
Yes, Using Backpropagation

So something important

- We stress this again, we allow the architecture to guide us...

The Score Matrix example

At the paper after training, we have the following example

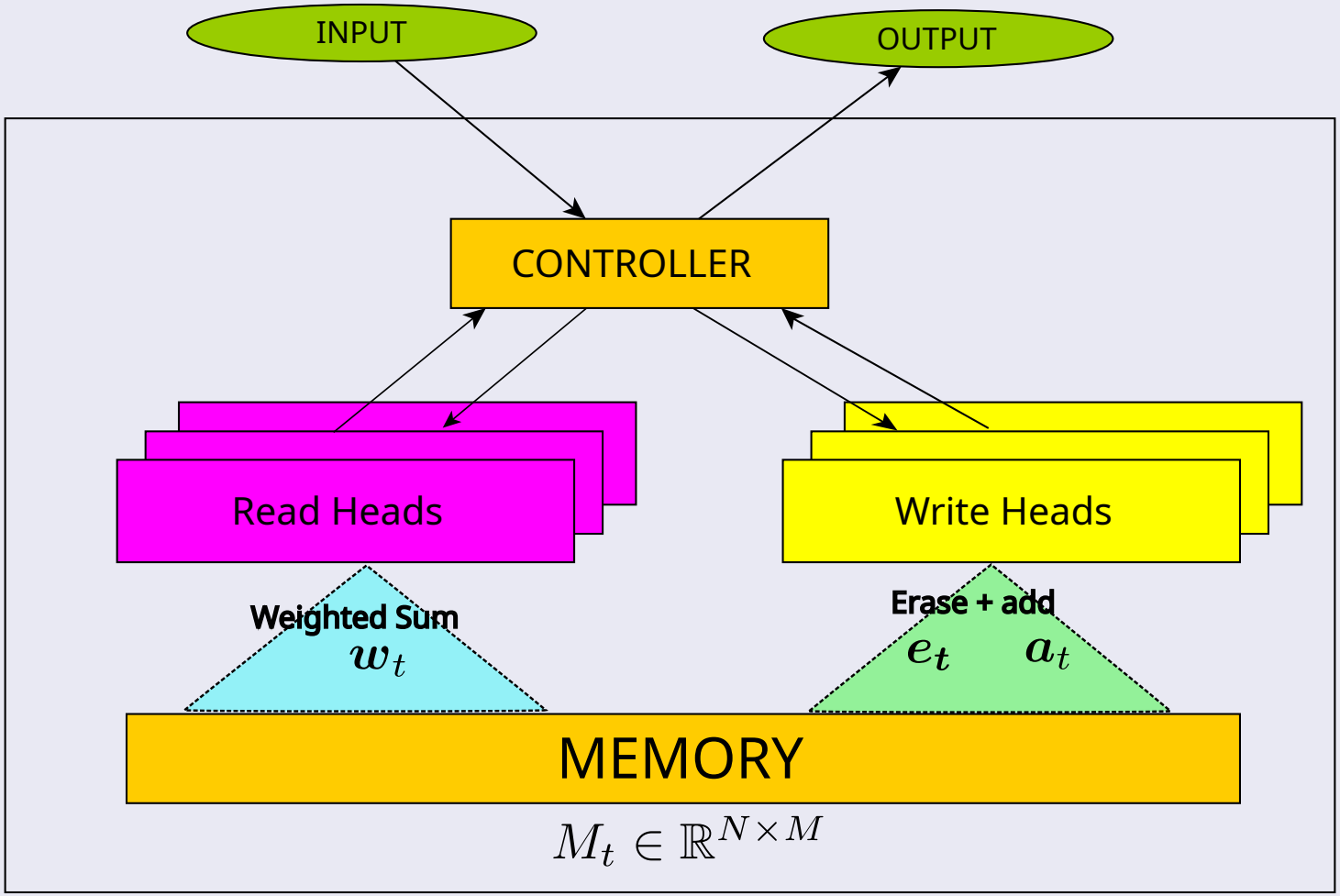


Outline

Outline

Content-Base Attention [3]

The Neural Turing Machine Architecture



Reading and Writing

Something Notable

- When reading from the memory at time t, an attention vector of size N, $\boldsymbol{w}_t$ controls how much attention to assign to different memory locations (matrix rows).

Reading and Writing

Something Notable

- When reading from the memory at time t, an attention vector of size N, $\boldsymbol{w}_t$ controls how much attention to assign to different memory locations (matrix rows).

The read vector is a sum weighted by attention intensity

$$\boldsymbol{r}_t = \sum_{i=1}^{N} \boldsymbol{w}_t\left(i\right) M_t\left(i\right)$$

$$\text{where } \sum_{i=1}^{N} \boldsymbol{w}_t\left(i\right) = 1, \forall i : 0 \leq \boldsymbol{w}_t\left(i\right) \leq 1$$

Then at reading

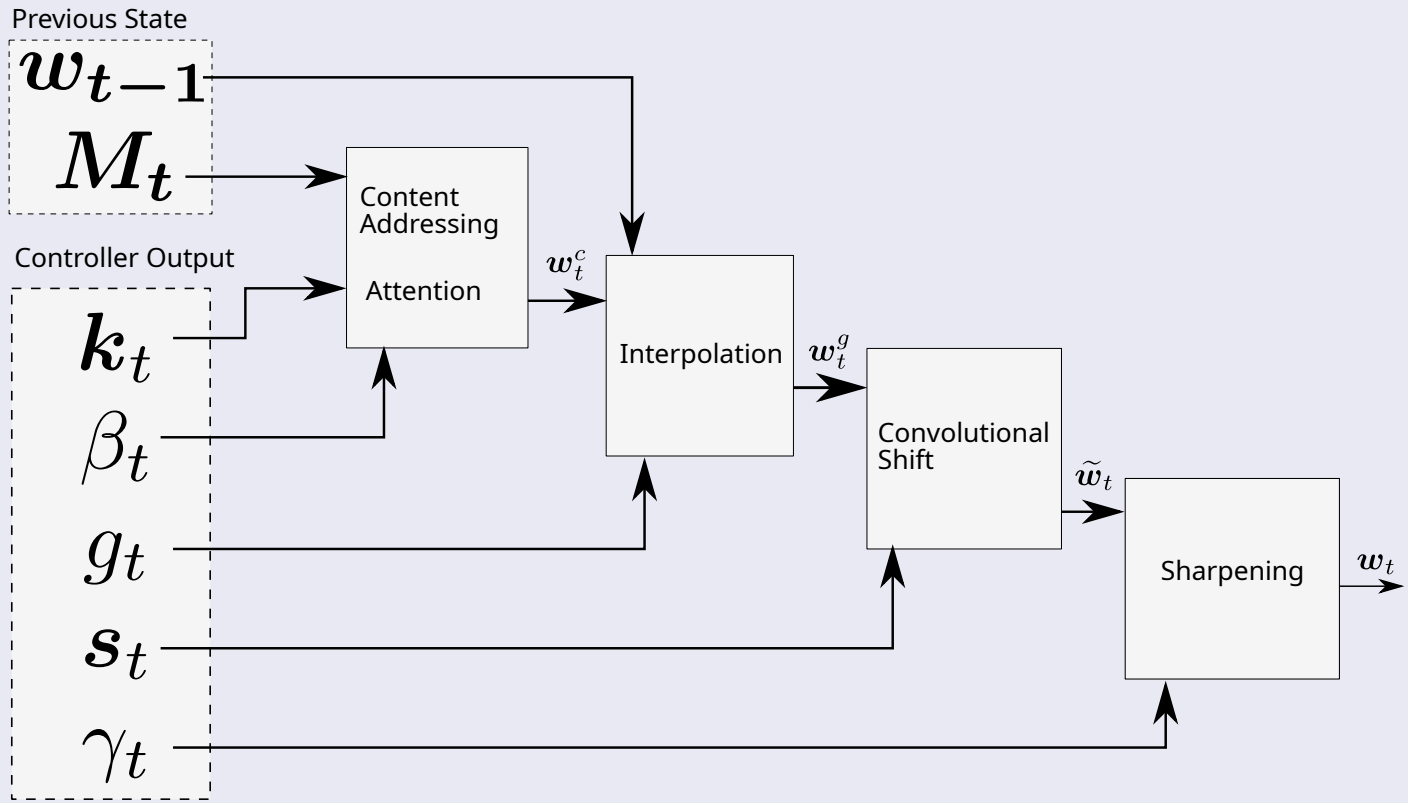
When writing into the memory at time t

- A write head first wipes off some old content according to an erase vector $\boldsymbol{e}_t$.
- Then adds new information by an add vector $\boldsymbol{a}_t$.

$$\widetilde{\boldsymbol{M}}_t(i) = \boldsymbol{M}_{t-1}(i)\left[1 - \boldsymbol{w}_t(i)\,\boldsymbol{e}_t\right]$$
$$\boldsymbol{M}_t(i) = \widetilde{\boldsymbol{M}}_t(i) + \boldsymbol{w}_t(i)\,\boldsymbol{a}_t$$

The interesting part

Flow Diagram of the Addressing Mechanism.



The Content-Base Attention

Something Notable

$$w_t^c(i) = \frac{\exp\{\beta_t K[\boldsymbol{k}_t, M_t(i)]\}}{\sum_j \exp\{\boldsymbol{k}_t, M_t(j)\}}$$

The Content-Base Attention

Something Notable

$$w_t^c(i) = \frac{\exp\{\beta_t K[\boldsymbol{k}_t, M_t(i)]\}}{\sum_j \exp\{\boldsymbol{k}_t, M_t(j)\}}$$

Where, we have the cosine similarity

$$K[\boldsymbol{u}, v] = \frac{\langle \boldsymbol{u}, v \rangle}{\|\boldsymbol{u}\| \|v\|}$$

Interpolation

We have that

- Then an interpolation gate scalar g_t is used to blend the newly generated content-based attention vector with the attention weights in the last time step:

$$\boldsymbol{w}_t^g = g_t \boldsymbol{w}_t^c + (1 - g_t)\, \boldsymbol{w}_{t-1}$$

Convolutional Shift/Location-based addressing

Location-based addressing

- The location-based addressing sums up the values at different positions in the attention vector, weighted by a weighting distribution over allowable integer shifts

$$AKA \approx 1 - d \text{ Convolution}$$

Convolutional Shift/Location-based addressing

Location-based addressing

- The location-based addressing sums up the values at different positions in the attention vector, weighted by a weighting distribution over allowable integer shifts

$$AKA \approx 1-d \text{ Convolution}$$

Thus, we have

$$\tilde{\boldsymbol{w}}_t(i) = \sum_{j=1}^{N} \boldsymbol{w}_t^g(j)\, \boldsymbol{s}_t(i-j) \quad \text{(Circular Convolution)}$$

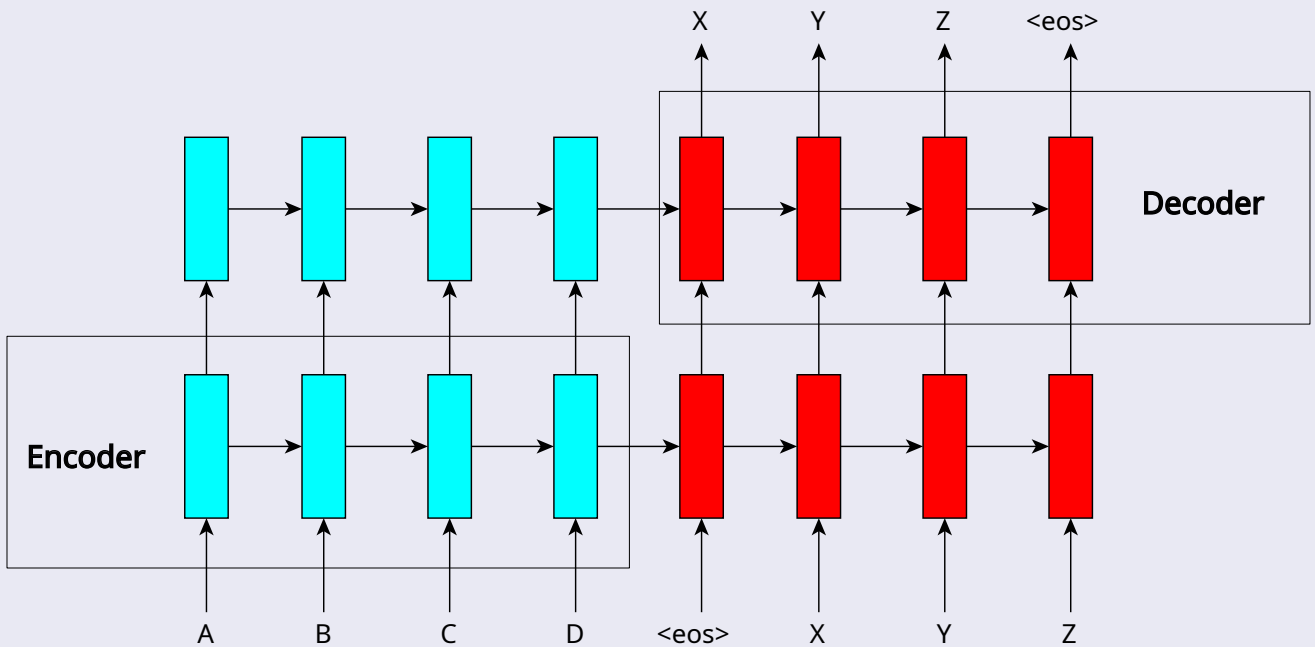
$$\boldsymbol{w}_t(i) = \frac{\left[\tilde{\boldsymbol{w}}_t(i)\right]^{\gamma_t}}{\sum_{j=1}^{N}\left[\tilde{\boldsymbol{w}}_t(j)\right]} \quad \text{(Sharpen)}$$

Outline

Location-Base Attention

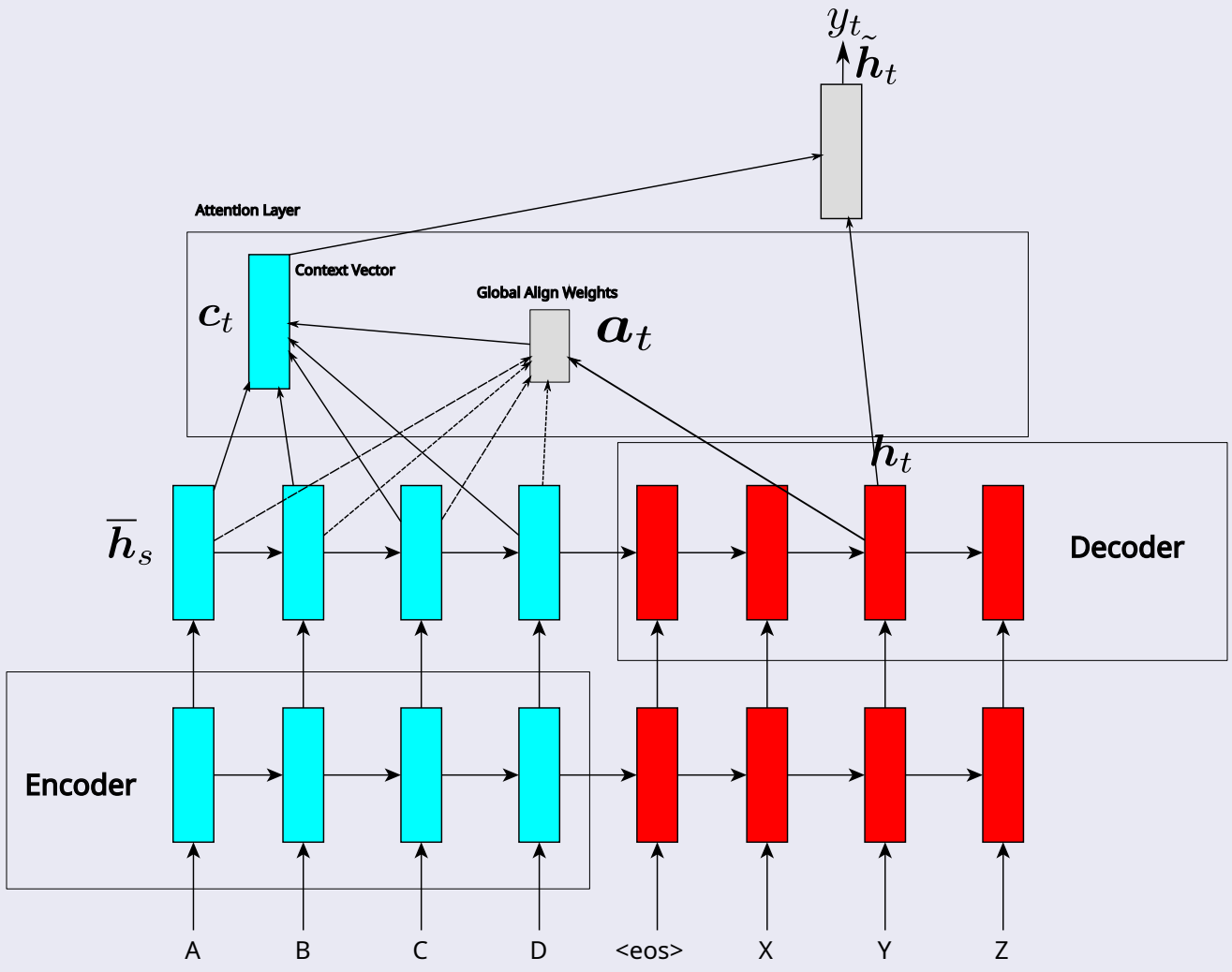
This was introduced by Luong et al. around 2015 [4]

- It modifies the classic stack translation architecture.



Luong et al. proposed

A Global Attention Model



Global Attention

Main Concept

- The idea of a global attention model is to consider all the hidden states of the encoder when deriving the context vector $\boldsymbol{c}_t$.

Global Attention

Main Concept

- The idea of a global attention model is to consider all the hidden states of the encoder when deriving the context vector $\boldsymbol{c}_t$.

It defines a vector $\boldsymbol{a}_t$ with size equal the number of steps on the source

- Each element in the vector is derived by comparing the current state $\boldsymbol{h}_t$ with each source state $\overline{\boldsymbol{h}}_s$

$$\boldsymbol{a}_t(s) = align\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \frac{\exp\left\{score\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right)\right\}}{\sum_{s'} \exp\left\{\boldsymbol{h}_t, \overline{\boldsymbol{h}}_{s'}\right\}}$$

Where Score is defined

We have three alternatives for content-based functions

- Dot

$$score\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \boldsymbol{h}_t^T \overline{\boldsymbol{h}}_s$$

- General

$$score\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \boldsymbol{h}_t^T W_a \overline{\boldsymbol{h}}_s$$

- Concat

$$score\left(\boldsymbol{h}_t, \overline{\boldsymbol{h}}_s\right) = \boldsymbol{v}_a^T \tanh\left(W_a \left[\boldsymbol{h}_t^T; \overline{\boldsymbol{h}}_s\right]\right)$$

In addition

A location-based function is defined

$$\boldsymbol{a}_t = \text{softmax}\left(W_a \boldsymbol{h}_t\right)$$

In addition

A location-based function is defined

$$\boldsymbol{a}_t = \text{softmax}\left(W_a \boldsymbol{h}_t\right)$$

Here, they define a computation

$$\boldsymbol{h}_t \rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \tilde{\boldsymbol{h}}_t$$

- There are other parts but take a look at the paper.

Outline

Finally in the "Attention all is you need" [5]

The source of the transformers

- They proposed a scaled dot product for the score

$$score\left(\boldsymbol{s}_t, \boldsymbol{h}_i\right) = \frac{\boldsymbol{s}_t^T \boldsymbol{h}_i}{\sqrt{n}}$$

Outline

Self Attention

Self-attention [6], also known as intra-attention

- It is is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence.

Self Attention

Self-attention [6], also known as intra-attention

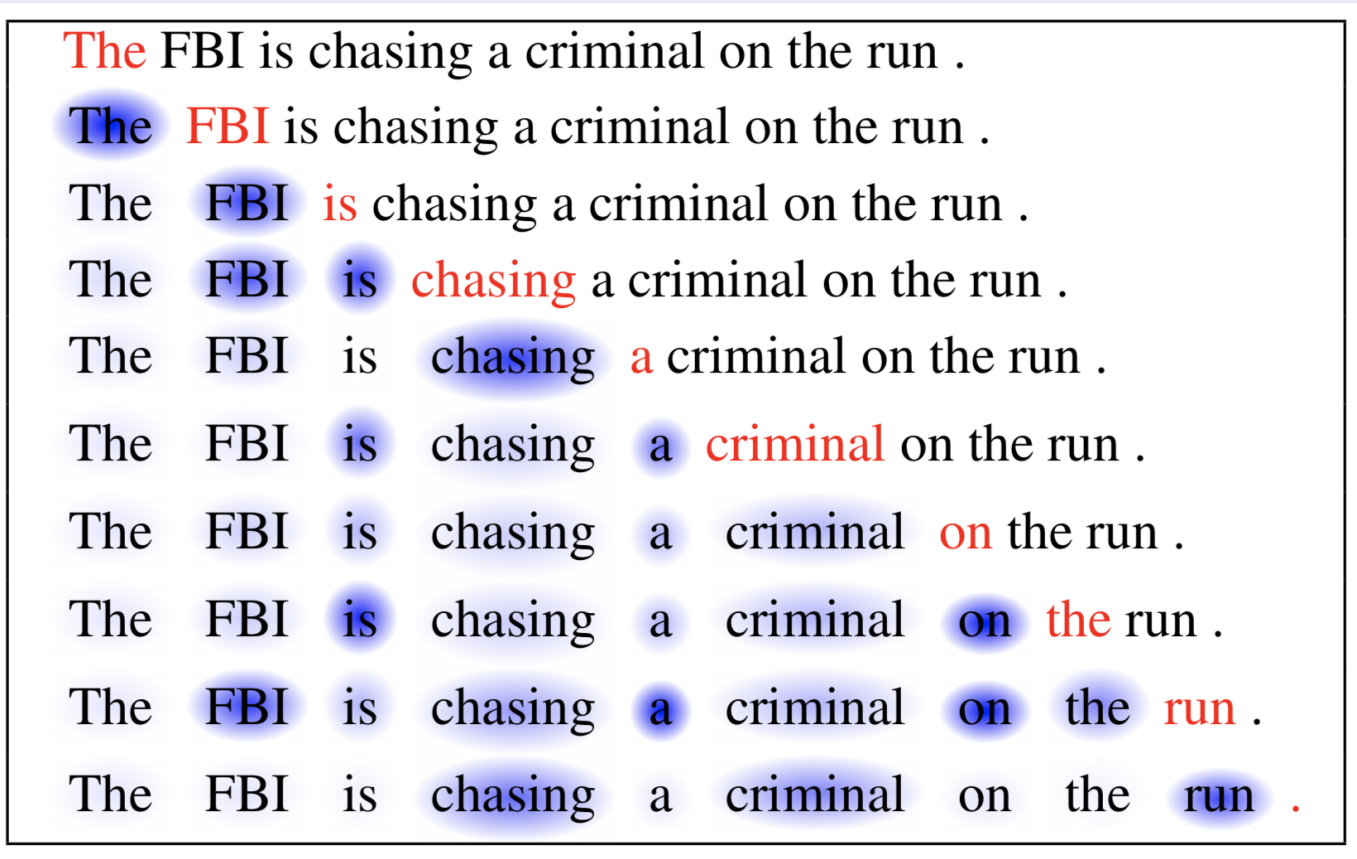
- It is is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence.

Quite useful in

- machine reading, abstractive summarization, or image description generation
  - Actually redefined for the Transformer!!!

For example

Using LSTM - Image source: Cheng et al., 2016



As you can see

We mention early

- LSTM are not that great for long sentences

As you can see

We mention early

- LSTM are not that great for long sentences

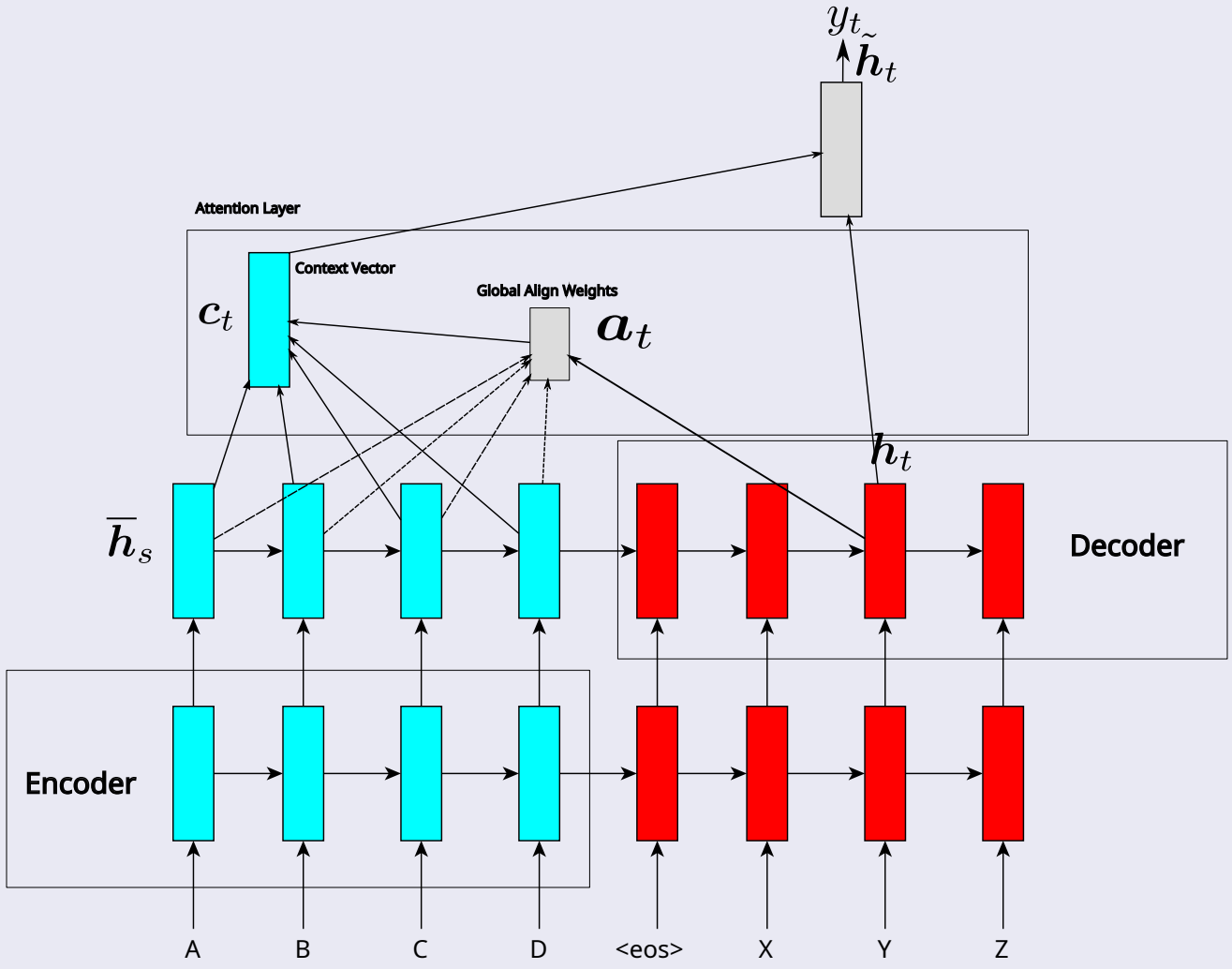
But It can be used for short phrases

- As in the previous example.

Outline

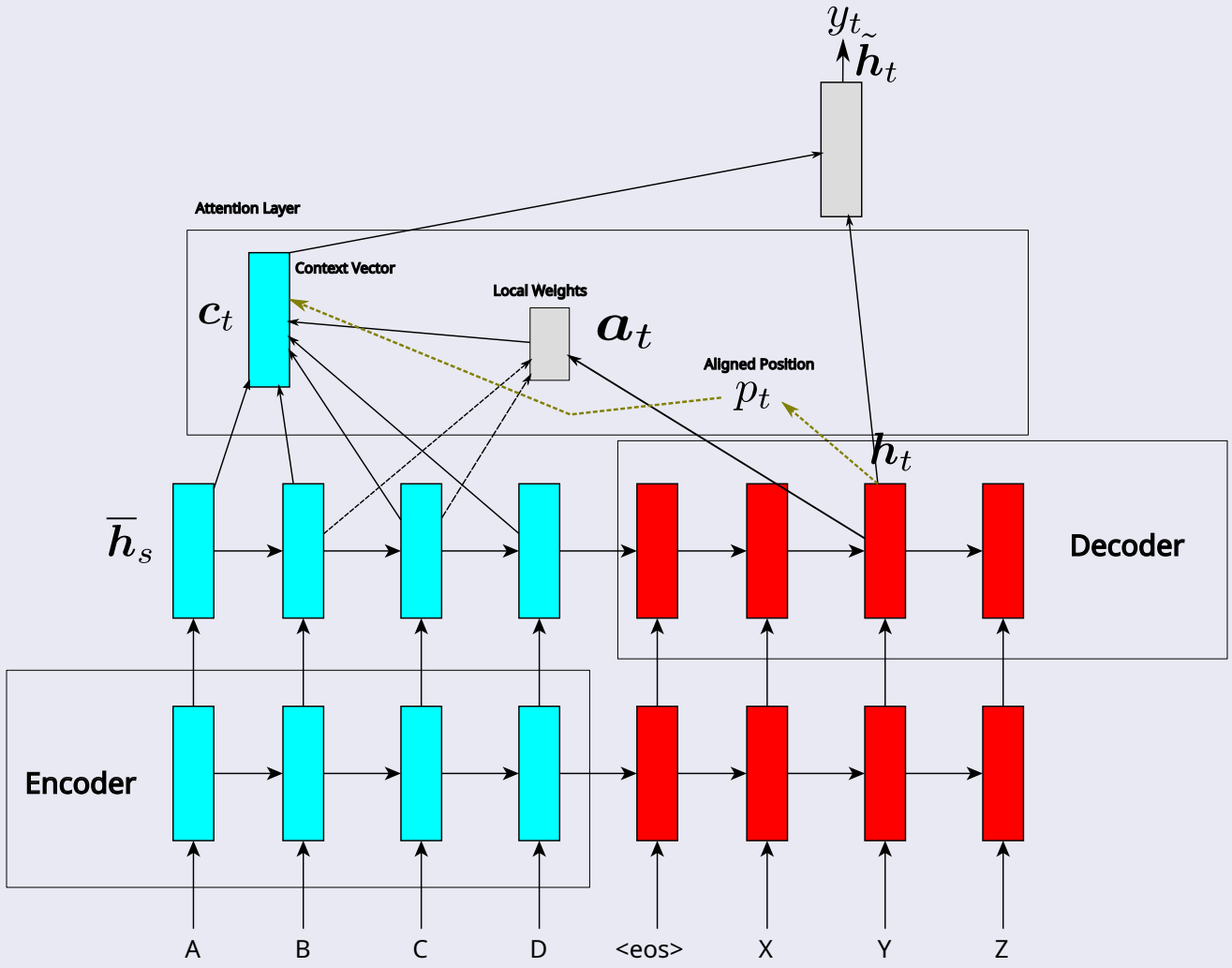
We already

We already seen this



We already

We already seen this

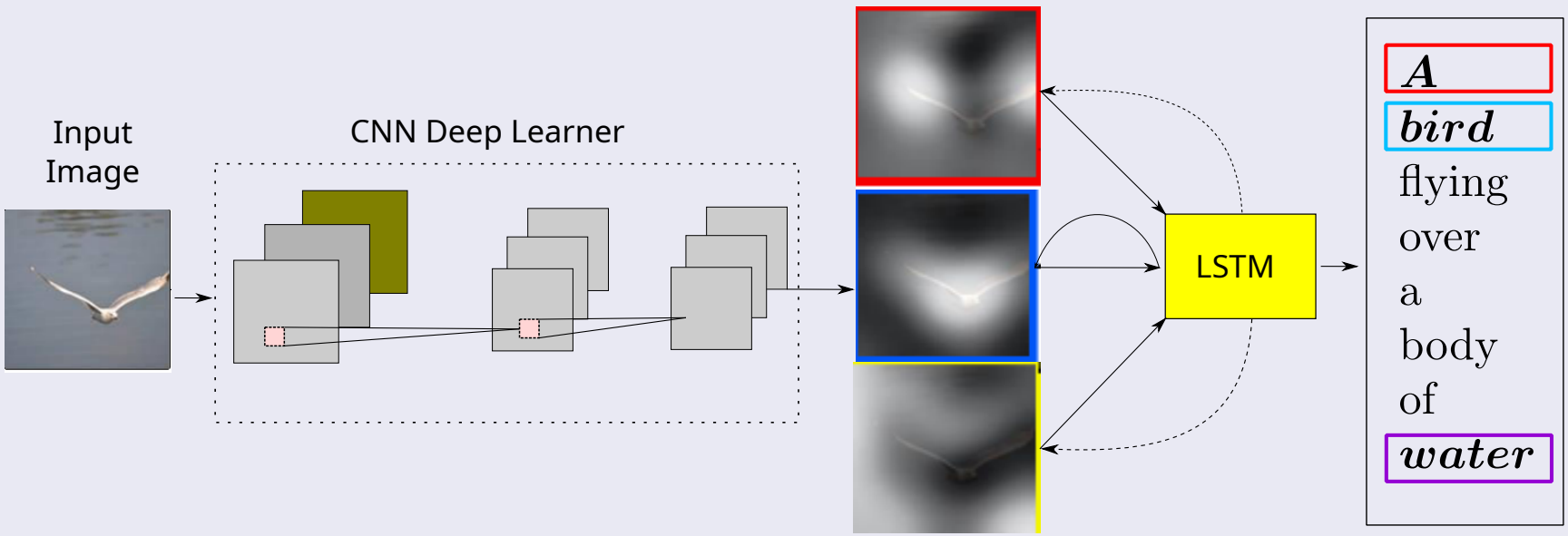


Outline

Soft vs Hard Attention [7]

Xu et al. (Circa 2016) proposed a LSTM together with a CNN

- To generate one of the first automated captioning system based on Deeplearning.



Basically CNN/Encoder

The model does the following

- A single raw image and generates a caption y encoded as a sequence of 1-of-K encoded words.

$$y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_C\}, \boldsymbol{y}_i \in \mathbb{R}^K$$

Basically CNN/Encoder

The model does the following

- A single raw image and generates a caption y encoded as a sequence of 1-of-K encoded words.

$$y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_C\}, \boldsymbol{y}_i \in \mathbb{R}^K$$

Thus the Convolutional Network extract a series of features

- L vectors, each of which is a D-dimensional representation corresponding to a part of the image.

$$a = \{\boldsymbol{a}_1, ..., \boldsymbol{a}_L\}, \boldsymbol{a}_i \in \mathbb{R}^D$$

Basically CNN/Encoder

The model does the following

- A single raw image and generates a caption y encoded as a sequence of 1-of-K encoded words.

$$y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_C\}, \boldsymbol{y}_i \in \mathbb{R}^K$$

Thus the Convolutional Network extract a series of features

- L vectors, each of which is a D-dimensional representation corresponding to a part of the image.

$$a = \{\boldsymbol{a}_1, ..., \boldsymbol{a}_L\}, \boldsymbol{a}_i \in \mathbb{R}^D$$

This is based in Deep Taylor Decomposition ideas

- Take a look at "Explaining nonlinear classification decisions with deep Taylor decomposition"[8]

The things change a little bit at LSTM/Decoder

Using $T_{s,t} : \mathbb{R}^s \to \mathbb{R}^t$ to denote a simple affine transformation with parameters that are learned

$$\begin{pmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \boldsymbol{g}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} T_{D+m+n,n} \begin{pmatrix} \boldsymbol{E}\boldsymbol{y}_{t-1} \\ \boldsymbol{h}_{t-1} \\ \hat{\boldsymbol{z}}_t \end{pmatrix}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i} \odot \boldsymbol{g}_t$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

- $\boldsymbol{i}_t$ input
- $\boldsymbol{f}_t$ forget
- $\boldsymbol{c}_t$ memory
- $\boldsymbol{o}_t$ output
- $\boldsymbol{h}_t$ hidden state

Not only that

The vector $\hat{z}_t \in \mathbb{R}^D$ is the context vector

- capturing the visual information associated with a particular input location

Not only that

The vector $\hat{z}_t \in \mathbb{R}^D$ is the context vector

- capturing the visual information associated with a particular input location

And always Embeddings

- Or projection matrix (Linear Algebra always risign his head to wink at us)

$$E \in \mathbb{R}^{m \times K}$$

  - Where m and n denote the embedding and LSTM dimensionality respectively

Outline

Then, we have for attention

We have a MLP as an attention mechanism using $\boldsymbol{a}_i$ and $\boldsymbol{h}_{t-1}$

$$e_{ti} = MLP\left(\boldsymbol{a}_i, \boldsymbol{h}_{t-1}\right)$$

Then, we have for attention

We have a MLP as an attention mechanism using $\boldsymbol{a}_i$ and $\boldsymbol{h}_{t-1}$

$$e_{ti} = MLP\left(\boldsymbol{a}_i, \boldsymbol{h}_{t-1}\right)$$

Then the output is passed through a softmax

$$\alpha_{ti} = \frac{\exp\left\{e_{ti}\right\}}{\sum_1^L \exp\left\{e_{ti}\right\}}$$

Then, we have for attention

We have a MLP as an attention mechanism using $\boldsymbol{a}_i$ and $\boldsymbol{h}_{t-1}$

$$e_{ti} = MLP\left(\boldsymbol{a}_i, \boldsymbol{h}_{t-1}\right)$$

Then the output is passed through a softmax

$$\alpha_{ti} = \frac{\exp\left\{e_{ti}\right\}}{\sum_1^L \exp\left\{e_{ti}\right\}}$$

And finally $\hat{\boldsymbol{z}}_t$

- It is the result of using

$$\hat{\boldsymbol{z}}_t = \phi\left(\left\{\boldsymbol{a}_i\right\}, \left\{\alpha_i\right\}\right)$$

This function can be defined by "Hard" Attention

First and foremost an indicator function s_t

- We represent the location variable s_t as where the model decides to focus attention when generating the t^{th} word

This function can be defined by "Hard" Attention

First and foremost an indicator function s_t

- We represent the location variable s_t as where the model decides to focus attention when generating the t^{th} word

Which is a one hot encoding

- $s_{t,i}$ is an indicator one-hot variable which is set to 1 if the i^{th} location (out of L) is the one used to extract visual features.

This function can be defined by "Hard" Attention

First and foremost an indicator function s_t

- We represent the location variable s_t as where the model decides to focus attention when generating the t^{th} word

Which is a one hot encoding

- $s_{t,i}$ is an indicator one-hot variable which is set to 1 if the i^{th} location (out of L) is the one used to extract visual features.

Thus, we can define a Multinoulli distribution

- Which is defined as follow...

Multinoulli Distribution

Definition

- Let X be a $K \times 1$ discrete random vector. Let the support of X be the set of $K \times 1$ vectors having one entry equal to 1 and all other entries equal to 0:

$$R_X = \left\{ x \in \{0,1\}^K \mid \sum_{j=1}^{K} x_j = 1 \right\}$$

Multinoulli Distribution

Definition

- Let X be a $K \times 1$ discrete random vector. Let the support of X be the set of $K \times 1$ vectors having one entry equal to 1 and all other entries equal to 0:

$$R_X = \left\{ x \in \{0,1\}^K \mid \sum_{j=1}^{K} x_j = 1 \right\}$$

Let $p_1, ..., p_K$ be K strictly positive numbers such that

$$\sum_{j=1}^{K} p_j = 1$$

Multinoulli Distribution

Definition

- Let X be a $K \times 1$ discrete random vector. Let the support of X be the set of $K \times 1$ vectors having one entry equal to 1 and all other entries equal to 0:

$$R_X = \left\{ x \in \{0,1\}^K \mid \sum_{j=1}^{K} x_j = 1 \right\}$$

Let $p_1, ..., p_K$ be K strictly positive numbers such that

$$\sum_{j=1}^{K} p_j = 1$$

We say that X has a Multinoulli distribution with probabilities $p_1, ..., p_K$ if its joint probability mass function is

$$p_X(x_1, ..., x_K) = \begin{cases} \prod_{j=1}^{K} p_j^{x_j} & \text{if } (x_1, ..., x_K) \in R_X \\ 0 & otherwise \end{cases}$$

Therefore

We have that

$$\alpha_{t,i} = p\left(s_{t,i} = 1 | s_{j<t}, \boldsymbol{a}\right)$$

$$\hat{\boldsymbol{z}}_t = \sum_i s_{t,i} a_i$$

Therefore

We have that

$$\alpha_{t,i} = p\left(s_{t,i} = 1 | s_{j<t}, \boldsymbol{a}\right)$$

$$\hat{\boldsymbol{z}}_t = \sum_i s_{t,i} a_i$$

We define a new objective function L_s that is a variational lower bound

- on the marginal log-likelihood $\log p\left(\boldsymbol{y}|\boldsymbol{a}\right)$ by observing the sequence of words $\boldsymbol{y}$ given image features $\boldsymbol{a}$.

Therefore

We have that

$$\alpha_{t,i} = p\left(s_{t,i} = 1 | s_{j<t}, \boldsymbol{a}\right)$$

$$\hat{\boldsymbol{z}}_t = \sum_i s_{t,i} a_i$$

We define a new objective function L_s that is a variational lower bound

- on the marginal log-likelihood $\log p\left(\boldsymbol{y}|\boldsymbol{a}\right)$ by observing the sequence of words $\boldsymbol{y}$ given image features $\boldsymbol{a}$.

We have then

$$\begin{aligned}
\log p\left(\boldsymbol{y}|\boldsymbol{a}\right) &= \log \sum_s p\left(s|\boldsymbol{a}\right) p\left(\boldsymbol{y}|s, \boldsymbol{a}\right) \\
&\geq \sum_s p\left(s|\boldsymbol{a}\right) \log p\left(\boldsymbol{y}|s, \boldsymbol{a}\right) \\
&= L_s
\end{aligned}$$

Then, we derive by W, weights in general

We have

$$\frac{\partial L_s}{\partial W} \approx \sum_s p\left(s|\boldsymbol{a}\right)\left[\frac{\partial \log p\left(\boldsymbol{y}|s,\boldsymbol{a}\right)}{\partial W} + \log p\left(\boldsymbol{y}|s,\boldsymbol{a}\right)\frac{\partial \log p\left(s|\boldsymbol{a}\right)}{\partial W}\right]$$

Then, we derive by W, weights in general

We have

$$\frac{\partial L_s}{\partial W} \approx \sum_s p\left(s|\boldsymbol{a}\right)\left[\frac{\partial \log p\left(\boldsymbol{y}|s,\boldsymbol{a}\right)}{\partial W} + \log p\left(\boldsymbol{y}|s,\boldsymbol{a}\right)\frac{\partial \log p\left(s|\boldsymbol{a}\right)}{\partial W}\right]$$

Here, they assume that

$$\frac{\partial \log p\left(s|\boldsymbol{a}\right)}{\partial W} = \frac{1}{p\left(s|a\right)}$$

Then, we derive by W, weights in general

We have

$$\frac{\partial L_s}{\partial W} \approx \sum_s p\left(s|\boldsymbol{a}\right)\left[\frac{\partial \log p\left(\boldsymbol{y}|s,\boldsymbol{a}\right)}{\partial W} + \log p\left(\boldsymbol{y}|s,\boldsymbol{a}\right)\frac{\partial \log p\left(s|\boldsymbol{a}\right)}{\partial W}\right]$$

Here, they assume that

$$\frac{\partial \log p\left(s|\boldsymbol{a}\right)}{\partial W} = \frac{1}{p\left(s|a\right)}$$

Therefore, we can assume that $p\left(s|\boldsymbol{a}\right)$ is a constant

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial \log p\left(\boldsymbol{y}|s^i,\boldsymbol{a}\right)}{\partial W} + \log p\left(\boldsymbol{y}|s^i,\boldsymbol{a}\right)\frac{\partial \log p\left(s^i|\boldsymbol{a}\right)}{\partial W}\right]$$

- where $s^i \sim Multinoulli_L\left(\{\alpha_i\}\right)$

The final equation is based on Entropy

To further reduce the estimator variance

- An entropy term on the Multinouilli distribution H[s] is added

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\partial \log p\left(\boldsymbol{y}|s^i, \boldsymbol{a}\right)}{\partial W} + \cdots \right.$$
$$\left. \lambda_r \left[\log p\left(\boldsymbol{y}|s, \boldsymbol{a}\right) - b\right] \frac{\partial \log p\left(s^i|\boldsymbol{a}\right)}{\partial W} + \lambda_e \frac{\partial H\left[s^i\right]}{\partial W} \right]$$

The final equation is based on Entropy

To further reduce the estimator variance

- An entropy term on the Multinouilli distribution H[s] is added

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\partial \log p\left(\boldsymbol{y}|s^i, \boldsymbol{a}\right)}{\partial W} + \cdots \right.$$
$$\left. \lambda_r \left[\log p\left(\boldsymbol{y}|s, \boldsymbol{a}\right) - b\right] \frac{\partial \log p\left(s^i|\boldsymbol{a}\right)}{\partial W} + \lambda_e \frac{\partial H\left[s^i\right]}{\partial W} \right]$$

We have then

- λ_r and λ_e are two hyper-parameters set by cross-validation.

Outline

Here

We take the expected value over the context vector $\hat{\boldsymbol{z}}_t$

$$\mathbb{E}_{p(s_t|a)}\left[\hat{\boldsymbol{z}}_t\right] = \sum_{i=1}^{L} \alpha_{t,i}\boldsymbol{a}_i$$

Here

We take the expected value over the context vector $\hat{\boldsymbol{z}}_t$

$$\mathbb{E}_{p(s_t|a)}\left[\hat{\boldsymbol{z}}_t\right] = \sum_{i=1}^{L} \alpha_{t,i}\boldsymbol{a}_i$$

Formulate a deterministic attention model by computing a soft attention weighted annotation vector

$$\phi\left(\{\boldsymbol{a}_i\}, \{\alpha_i\}\right) = \sum_{i=1}^{L} \alpha_{t,i}\boldsymbol{a}_i$$

- It is for you to look at the rest of the development...
  - They used and soft derivable function.

Last Observations

Hard Attention

- Only selects one patch of the image to attend to at a time
  - Pro: less calculation at the inference time.
  - Con: the model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train. (Luong, et al., 2015)

Last Observations

Hard Attention

- Only selects one patch of the image to attend to at a time
  - Pro: less calculation at the inference time.
  - Con: the model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train. (Luong, et al., 2015)

Soft Attention

- The alignment weights are learned and placed "softly" over all patches in the source image; essentially the same type of attention as Bahdanau et al., 2015.
  - Pro: the model is smooth and differentiable.
  - Con: expensive when the source input is large.

Outline

“Attention is All you Need” (Vaswani, et al., 2017) [5]

One of the most impactful and interesting paper in 2017

- It presented a lot of improvements to the soft attention and make it possible to do seq2seq modeling without recurrent network units (Get away from recurrence).

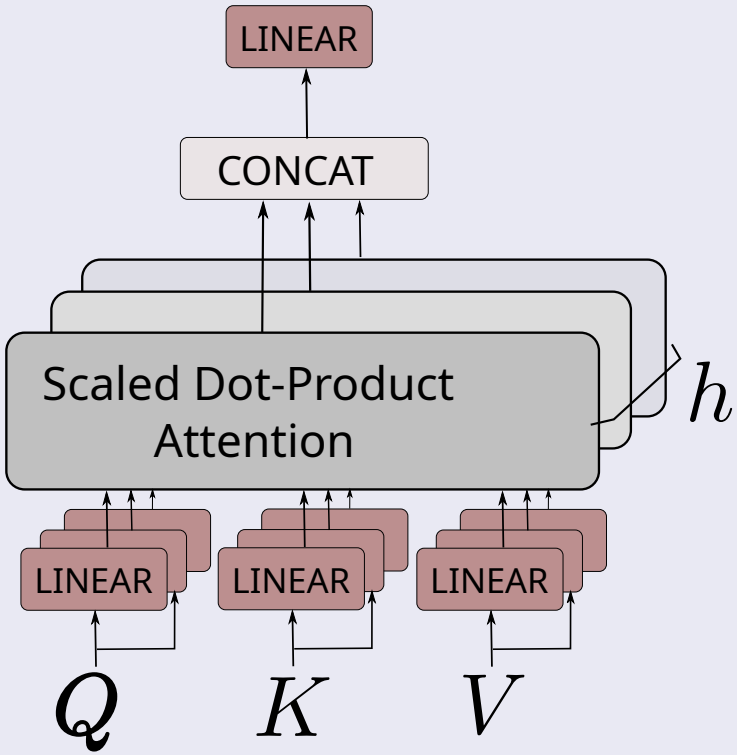
"Attention is All you Need" (Vaswani, et al., 2017) [5]

One of the most impactful and interesting paper in 2017

- It presented a lot of improvements to the soft attention and make it possible to do seq2seq modeling without recurrent network units (Get away from recurrence).

Basically, they proposed

- The Multihead Attention as basis of the Transformer



Outline

What is Self-Attention?

At Google Brain (Circa 2018)

- "Self Attention calculates a weighted average of feature representation with the weight proportional to a similarity score between pairs of representation"

What is Self-Attention?

At Google Brain (Circa 2018)

- "Self Attention calculates a weighted average of feature representation with the weight proportional to a similarity score between pairs of representation"

Formally, assume that $X \in \mathbb{R}^{batch \times n \times d_{model}}$

- And trainable weight matrices $W^Q, W^K, W^V \in \mathbb{R}^{d_{model} \times d_k}$
  - Where
    - ★ d_{model} is the size of the embedding vector of each input element from our sequence.
    - ★ d_k is the inner dimension of that is specific to each self-attention layer.
    - ★ $batch$ is the batch size
    - ★ n is the number of elements/tokens that our sequence has.

Outline

Embedding, Always Embedding!!!

It is clear, assuming NLP task, that you need to encode text into vectorial subspaces

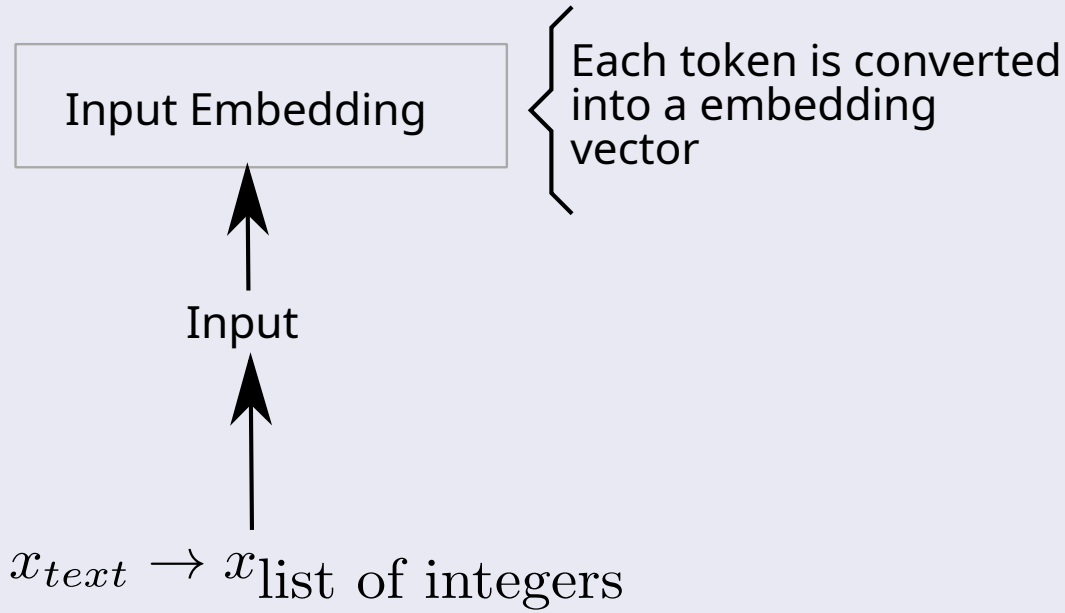
- Input text is encoded with tokenizers to sequence of integers called input tokens

Embedding, Always Embedding!!!

It is clear, assuming NLP task, that you need to encode text into vectorial subspaces

- Input text is encoded with tokenizers to sequence of integers called input tokens

Input tokens are mapped to sequence of vectors (word embeddings) via embeddings layer



This idea of using integers for representation

It is quite important

- For example, input text is split into frequent words e.g. transformer tokenization.
- Sometimes we append special tokens to the sequence e.g. class token ([CLS])

This idea of using integers for representation

It is quite important

- For example, input text is split into frequent words e.g. transformer tokenization.
- Sometimes we append special tokens to the sequence e.g. class token ([CLS])

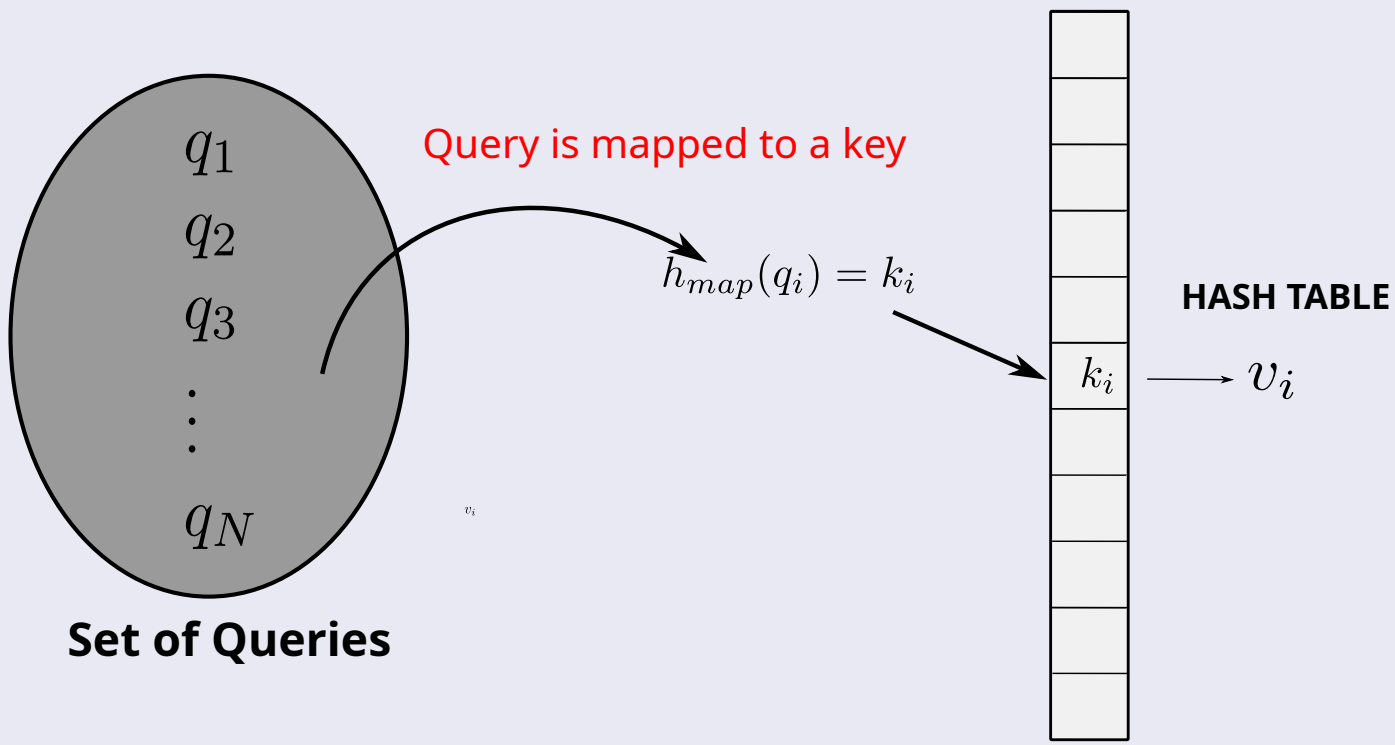
Latter on we will see the last secret source

- The Positional Encoding!!!

Ok, we have the tokens as vectors (Remember this is learned!!!)

We have the classic model, the hash map, where

- Queries are converted into keys then used to find a model



Ok, we have the tokens as vectors (Remember this is learned!!!)

We have three representations/subspaces

- The Query Space $W^Q \in \mathbb{R}^{d_{model} \times d_k}$

$$Q = XW^Q$$

- The Key Space $W^K \in \mathbb{R}^{d_{model} \times d_k}$

$$K = XW^K$$

- The Value $W^V \in \mathbb{R}^{d_{model} \times d_k}$

$$V = XW^V$$

Ok, we have the tokens as vectors (Remember this is learned!!!)

We have three representations/subspaces

- The Query Space $W^Q \in \mathbb{R}^{d_{model} \times d_k}$

$$Q = XW^Q$$

- The Key Space $W^K \in \mathbb{R}^{d_{model} \times d_k}$

$$K = XW^K$$

- The Value $W^V \in \mathbb{R}^{d_{model} \times d_k}$

$$V = XW^V$$

Therefore, we have that

- All the subspaces are in $\mathbb{R}^{batch \times n \times d_k}$

And the final definition of attention by Transformers

The attention layer Y

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

And the final definition of attention by Transformers

The attention layer Y

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

What is the quantity inside?

$$\text{dot-scores} = \frac{QK^T}{\sqrt{d_k}}$$

And the final definition of attention by Transformers

The attention layer Y

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

What is the quantity inside?

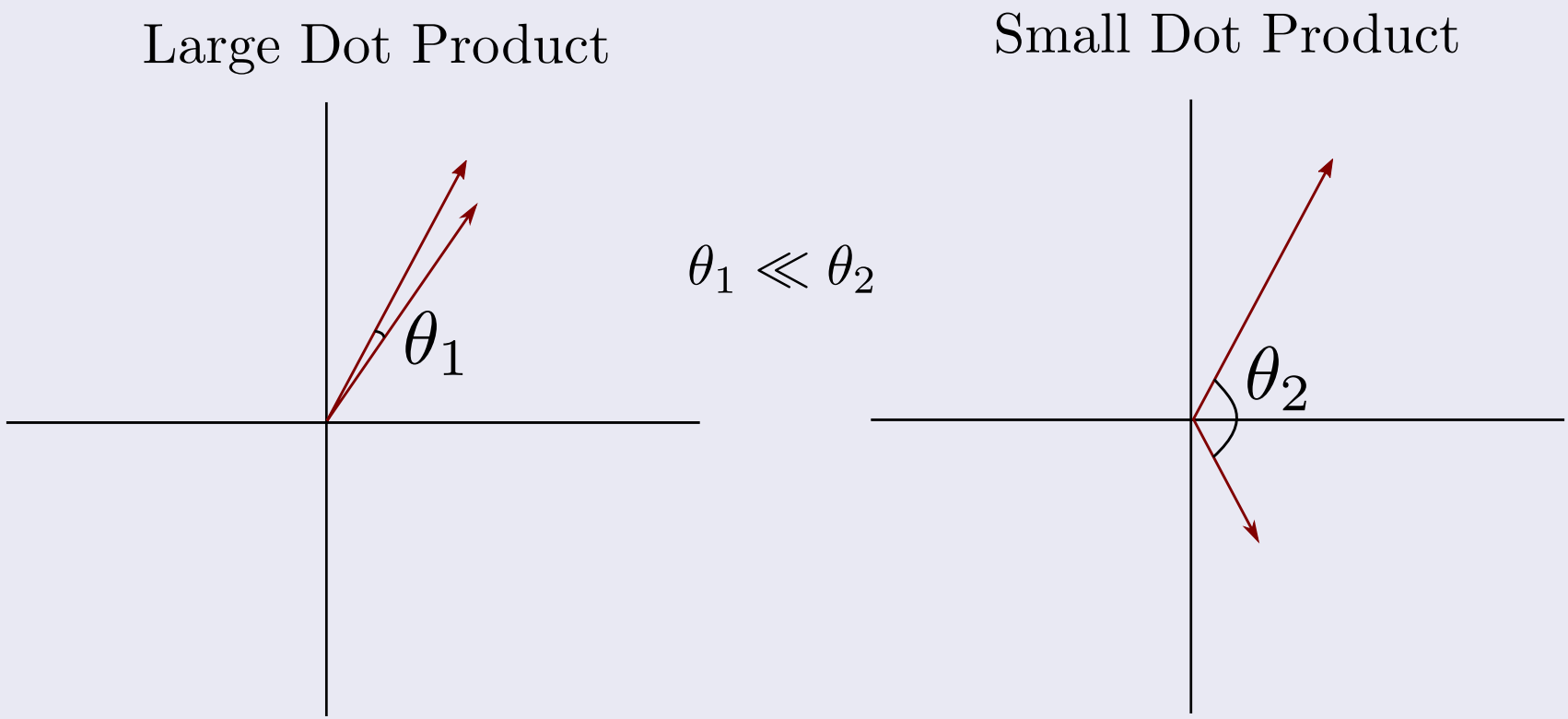
$$\text{dot-scores} = \frac{QK^T}{\sqrt{d_k}}$$

Something interesting

- The Higher they are the higher are the attention "weights"
  - This is why it is considered a similarity measure

What is the meaning of this "high" dot product

We use the angle to see if two vectors are similar



Outline

Here, we will consider something different

Queries do not come from the same sequences as keys and vectors

- The query is a sequence of 4 tokens and the sequence that we would like to associate with, contains 5 tokens.

Here, we will consider something different

Queries do not come from the same sequences as keys and vectors

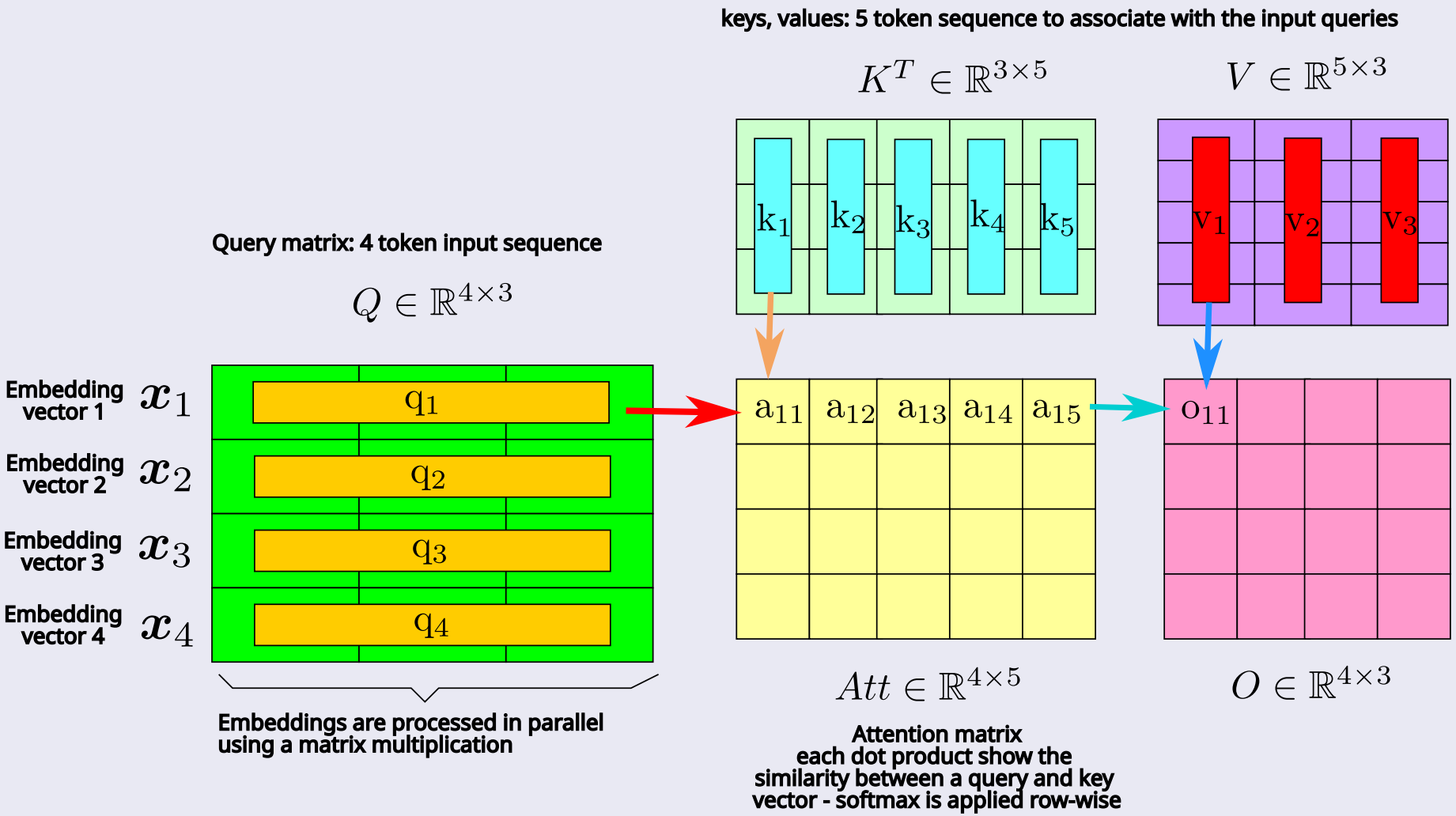
- The query is a sequence of 4 tokens and the sequence that we would like to associate with, contains 5 tokens.

Both sequences contain vectors of the same embedding dimension

$$d_{model} = 3$$

Therefore, we have

Think that you have row and column vectors!!!



The Query-Key matrix multiplication

Something Notable

- The query matrix in the attention layer is conceptually the “search” in the database.

The Query-Key matrix multiplication

Something Notable

- The query matrix in the attention layer is conceptually the "search" in the database.

Actually

- Intuitively, the keys are the bridge between the queries (what we are looking for) and the values (what we will actually get).

Keys are generated to search based on the Query

But Imagine the Backpropagation

- Based on the data generating W^Q, W^K, W^V

Keys are generated to search based on the Query

But Imagine the Backpropagation

- Based on the data generating W^Q, W^K, W^V

Here, we need to mention the fact that as softmax exist

$$\alpha_{ij} = \frac{\exp\{e_{ij}\}}{\sum_{k=1}^{T_x} \exp\{e_{ik}\}}$$

Keys are generated to search based on the Query

But Imagine the Backpropagation

- Based on the data generating W^Q, W^K, W^V

Here, we need to mention the fact that as softmax exist

$$\alpha_{ij} = \frac{\exp\{e_{ij}\}}{\sum_{k=1}^{T_x} \exp\{e_{ik}\}}$$

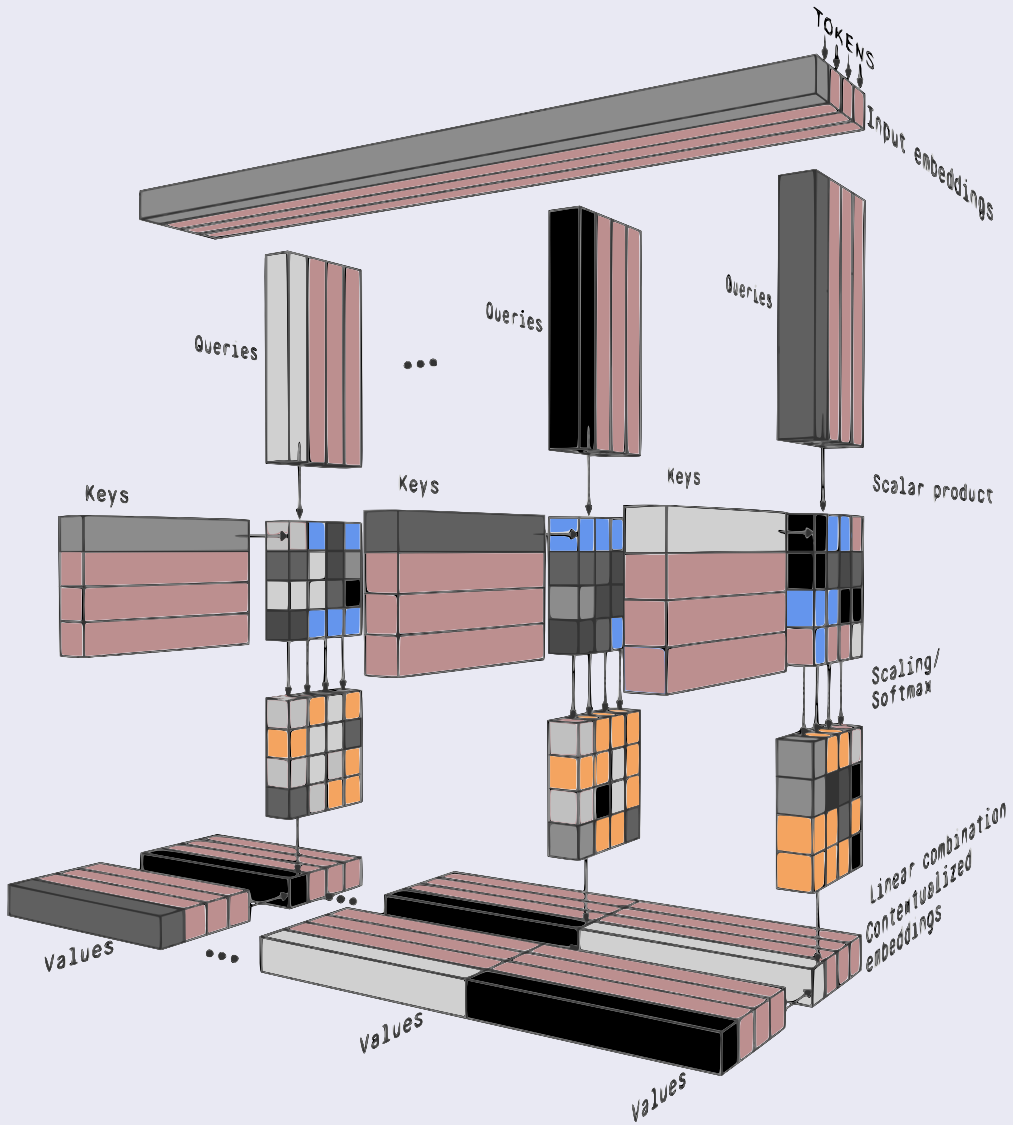
And the scale down factor

$$\sqrt{d_k}$$

Outline

Here is the other secret sauce

Instead of having a single attention head - they went for multiple



Actually

The idea is quite clever

- Intuitively, multiple heads enable us to attend independently to (parts of) the sequence.

Actually

The idea is quite clever

- Intuitively, multiple heads enable us to attend independently to (parts of) the sequence.

Formally, we have another transformation W^o

$$MultiHead\left(Q,K,V\right) = Concat\left(head_1,...,head_h\right)W^o$$

$$\text{Where } head_i = \texttt{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)$$

Outline

Self-Attention is Not Symmetric

We have the following

$$\frac{QK^T}{\sqrt{d_k}} = \frac{XW_Q\left(XW_K\right)^T}{\sqrt{d_k}} = \frac{XW_QW_k^TX^T}{\sqrt{d_k}}$$

Self-Attention is Not Symmetric

We have the following

$$\frac{QK^T}{\sqrt{d_k}} = \frac{XW_Q\left(XW_K\right)^T}{\sqrt{d_k}} = \frac{XW_QW_k^TX^T}{\sqrt{d_k}}$$

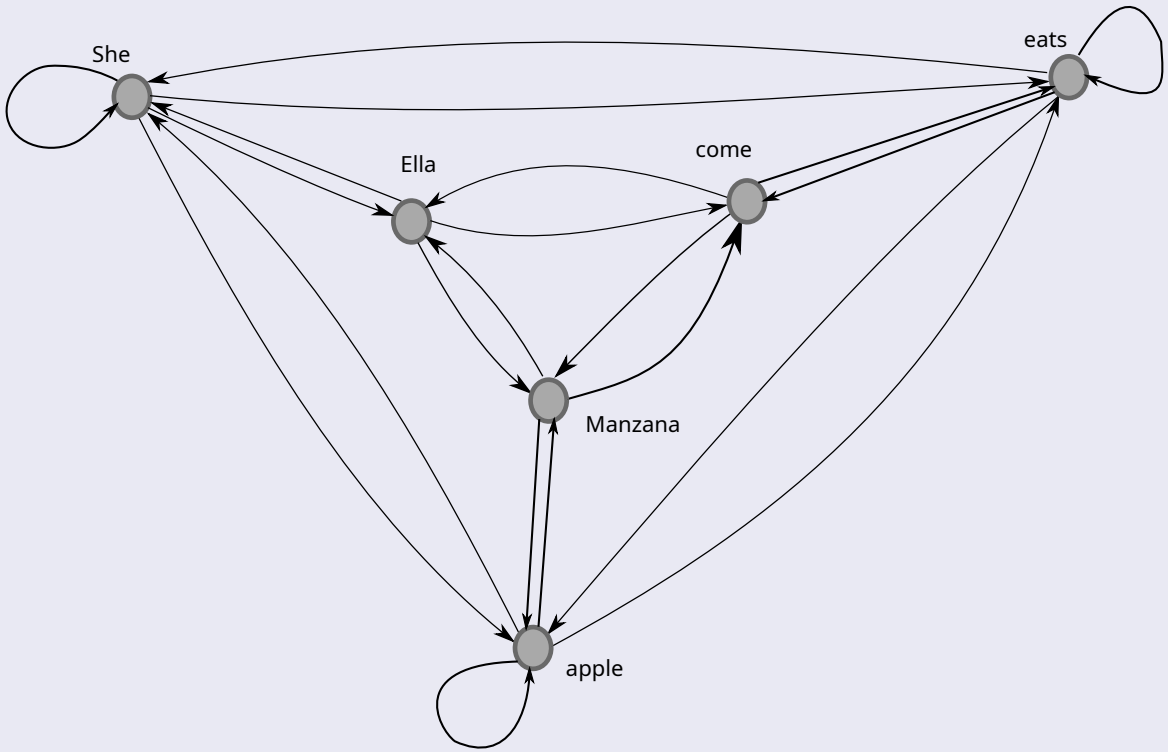
Easily seen because there are two different matrices

- Actually, the attention can be seen as a directed graph.

What?

If the Keys and Queries have the same amount of N tokens

- Not all the relations are shown



Outline

All this requieres memory and GPU time

There are attempts to reduce the number of heads

- "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned" [9]

All this requieres memory and GPU time

There are attempts to reduce the number of heads

- "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned" [9]

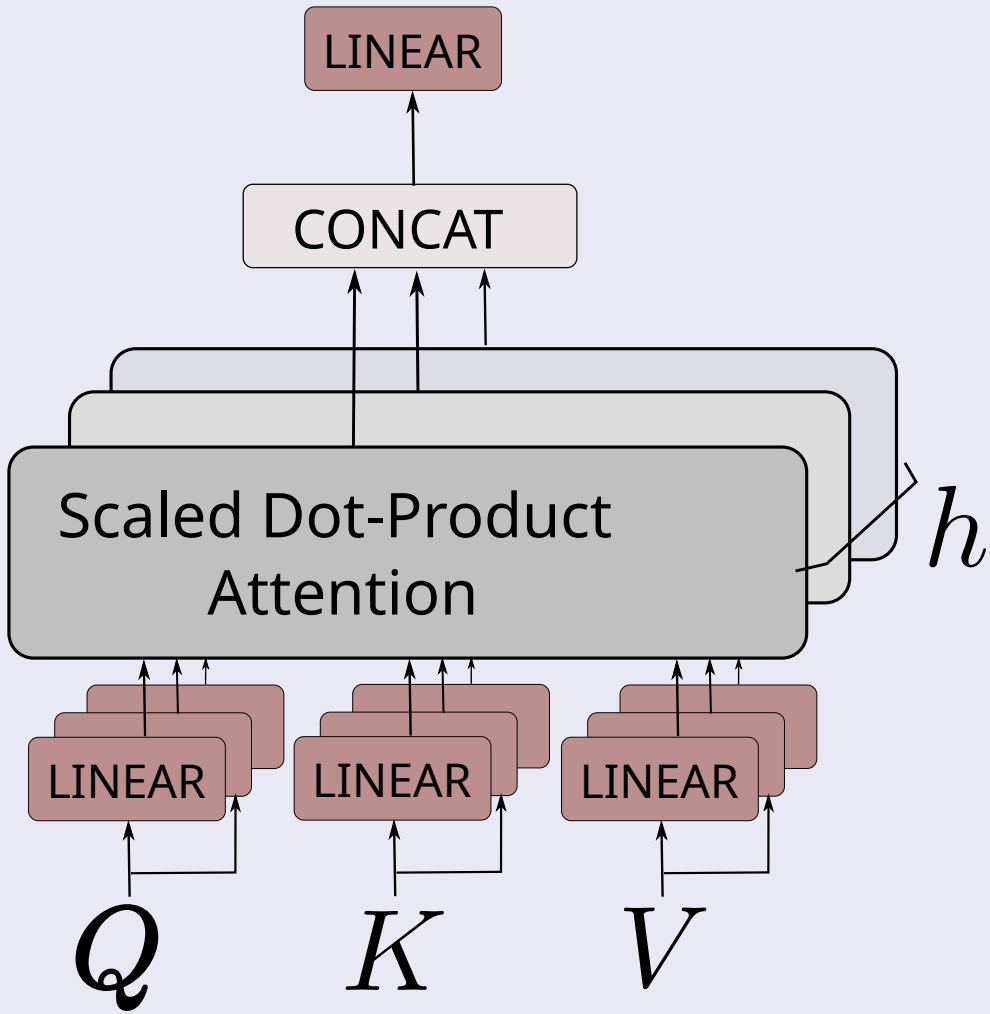
They used Layer Wise Relevance Propagation (LRP) [10]

- Which is based in Taylor Decomposition which will generate the ideas of Deep Taylor Decomposition [8]

Remember

Basically, they proposed

- The Multihead Attention as basis of the Transformer



Different Heads, Different Purposes

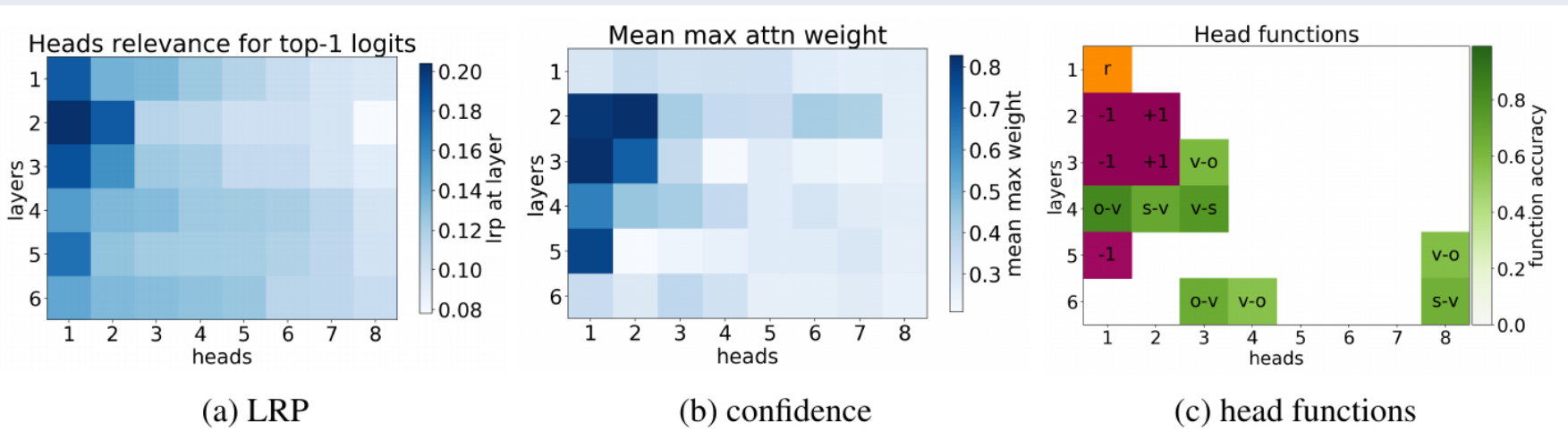
We have the following hierarchy

- Positional: the head points to an adjacent token
- Syntactic: the head points to tokens in a specific syntactic relation
- Rare words: the head points to the least frequent tokens in a sentence

We have the following maps in Multihead Attention

Importance (according to LRP), confidence, and function of self-attention heads

- In each layer, heads are sorted by their relevance according to LRP.
- Model trained on 6 million OpenSubtitles EN-RU data.
- +1 -1 near tokens, o = object, s = subject, v = verb, r = rare words



(a) LRP (b) confidence (c) head functions

Thus

The Multihead Attention

- It can be pruned to have only the essential mechanisms.

Thus

The Multihead Attention

- It can be pruned to have only the essential mechanisms.

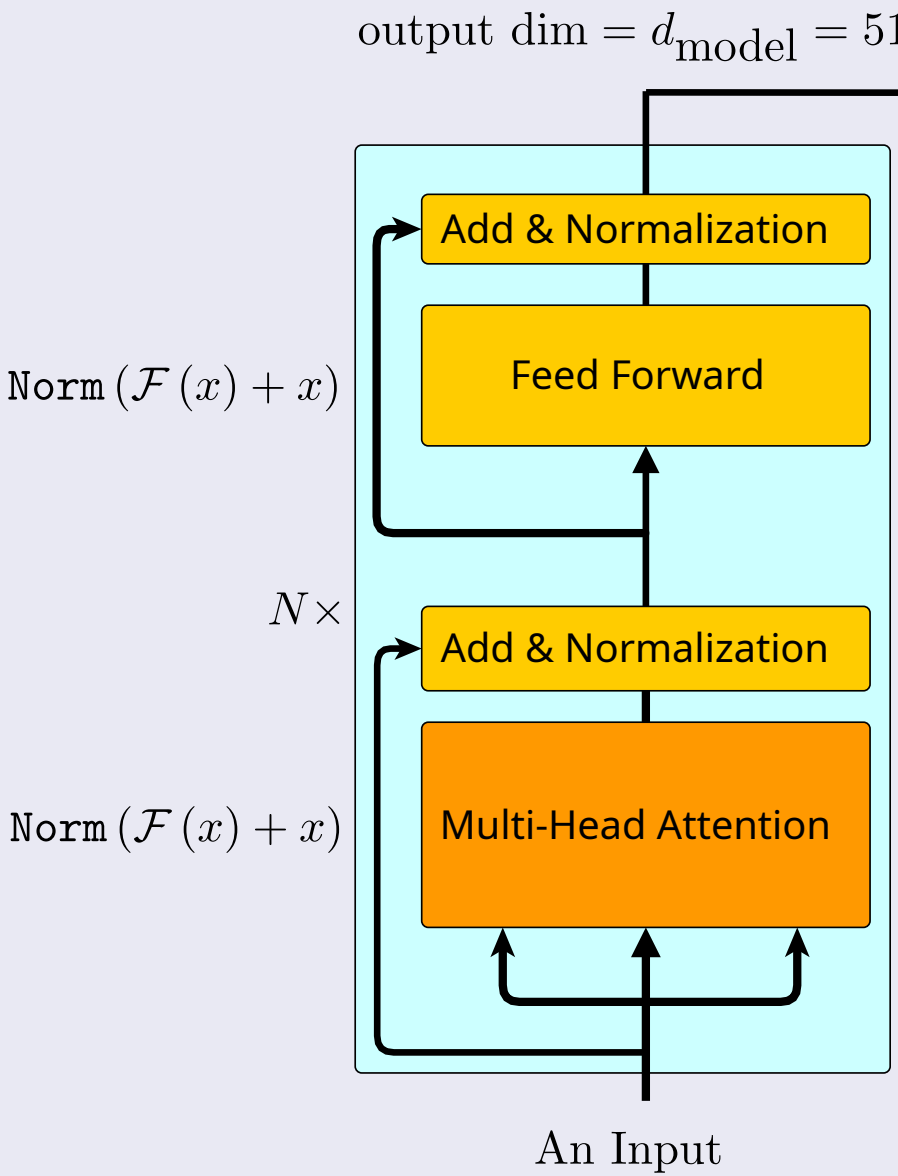
This is quite necessary

- Once we take a look to the monster, the Encoder-Decoder Transformer

Outline

We need to explain the structure of the Transformer

The Encoder

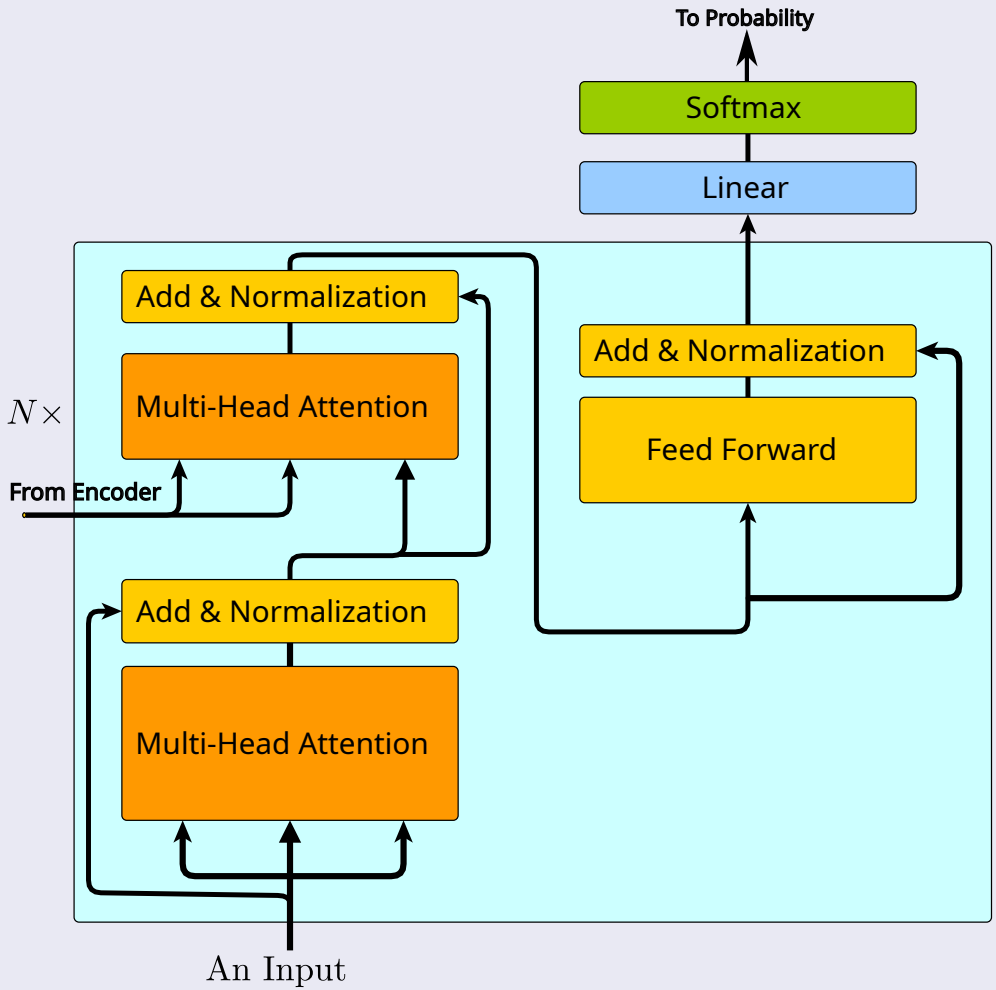


- A stack of N = 6 identical layers
- Each layer has a multi-head self-attention layer and a simple position-wise fully connected feed-forward network.
- Each sub-layer adopts a residual connection and a layer normalization.

Outline

At The Decoder

We have the following Architecture

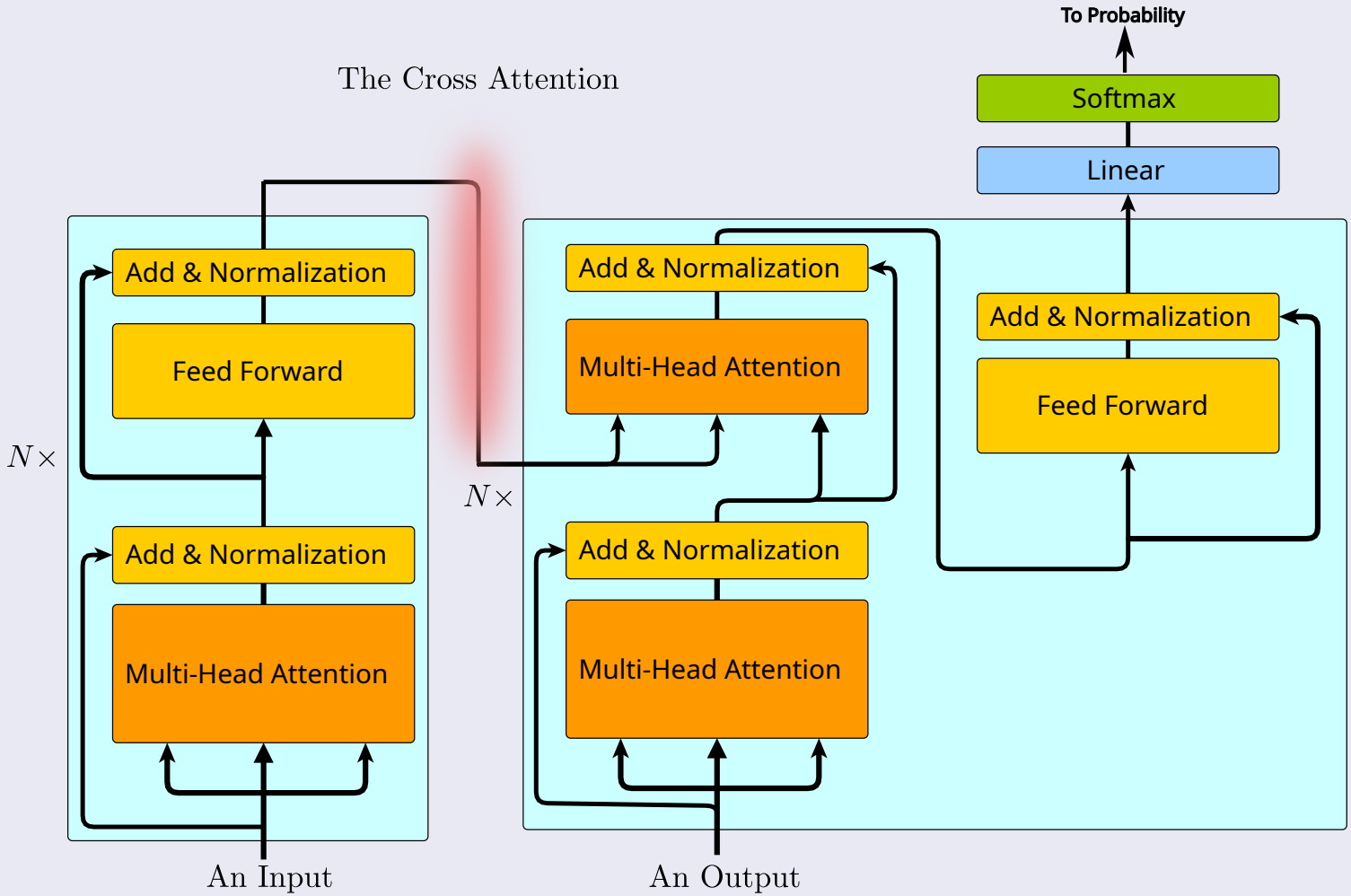


- A stack of N = 6 identical layers
- Each layer has two sub-layers of multi-head attention mechanisms and one sub-layer of fully-connected feed-forward network.
- Similar to the encoder, each sub-layer adopts a residual connection and a layer normalization.
- The first multi-head attention sub-layer is modified to prevent positions from attending to subsequent positions, as we do not want to look into the future of the target sequence when predicting the current position.

Outline

Putting All Together

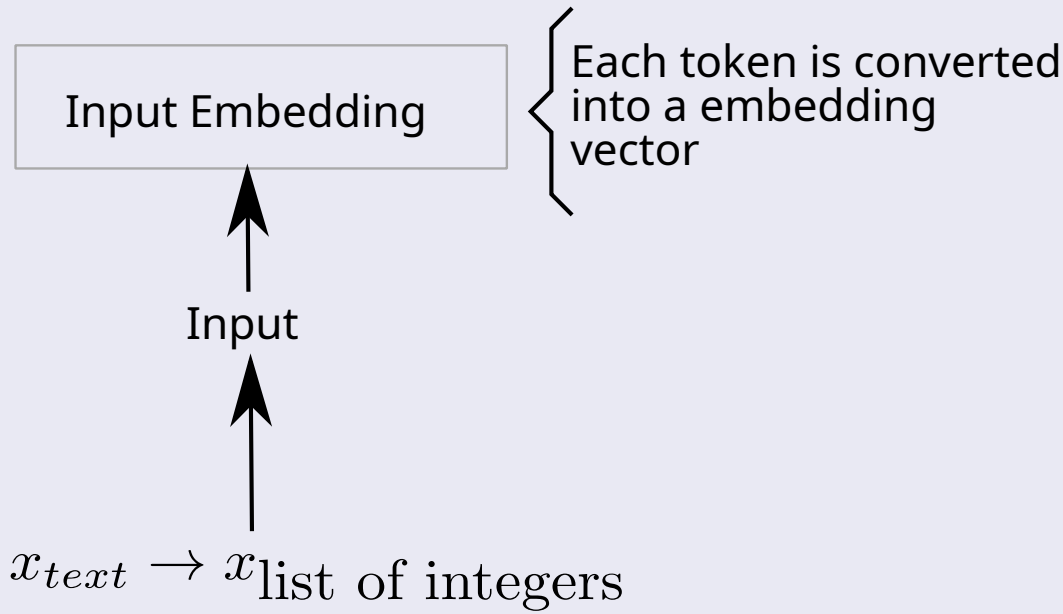
We notice how the input of the Decoder is different than the output of the Encoder



Outline

The Input/Output

Input tokens are mapped to sequence of vectors (word embeddings) via embeddings layer



Ok, but no positional infromation exists!!!

We ned to add such information to each vectos-token

Sequence of vectors representing		Index of Token		Postional Matrix			
I	$\Rightarrow$	0	$\Rightarrow$	P_{00}	P_{01}	$\cdots$	P_{0d}
am	$\Rightarrow$	1	$\Rightarrow$	P_{10}	P_{11}	$\cdots$	P_{1d}
a	$\Rightarrow$	2	$\Rightarrow$	P_{20}	P_{21}	$\cdots$	P_{1d}
Robot	$\Rightarrow$	3	$\Rightarrow$	P_{30}	P_{31}	$\cdots$	P_{1d}

Why not simply use the indexes?

Drawbacks

- For long sequences, the indices can grow large in magnitude.
- If you normalize the index value to lie between 0 and 1, it can create problems for variable length sequences as they would be normalized differently.

Sine Function

$sin\,(t)$

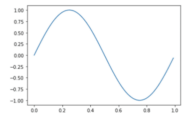
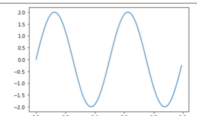
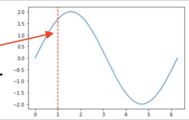
- The function's range is [-1,+1].
- The frequency of this waveform is the number of cycles completed in one second.
- The wavelength is the distance over which the waveform repeats itself.

Sine Function

$sin\,(t)$

- The function's range is [-1,+1].
- The frequency of this waveform is the number of cycles completed in one second.
- The wavelength is the distance over which the waveform repeats itself.

Thus we have

Equation	Graph	Frequency	Wavelength
$\sin(2\pi t)$		1	1
$\sin(2 * 2\pi t)$		2	1/2
$\sin(t)$		$1/2\pi$	2π
$\sin(ct)$	Depends on c	$c/2\pi$	$2\pi/c$

We can use it for the positional encoding

Thus, we have that

- An input sequence of length L and require the position of the k^{th} object within this sequence.

We can use it for the positional encoding

Thus, we have that

- An input sequence of length L and require the position of the k^{th} object within this sequence.

The positional encoding is given by sine and cosine functions of varying frequencies

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$
$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

- k: Position of an object in the input sequence $0 \leq k < L$,
- d: dimension of the output space
- n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.
- i: Used for mapping to column indices $0 \leq i < \frac{d}{2}$, with a single value of maps to both sine and cosine functions

Example

Look at this when $d = 4$ and $n = 100$

Sequence of Vectors		Index		Positional Matrix			
I	⇒	0	⇒	sin(0) = 0	cos(0) = 1	sin(0) = 0	cos(0) = 1
am	⇒	1	⇒	sin(1/1) = 0.84	cos(1/1) = 0.54	sin(1/10) = 0.10	cos(1/10) = 0
a	⇒	2	⇒	sin(2/1) = 0.91	cos(2/1) = −0.42	sin(2/10) = 0.20	cos(2/10) = 0.98
Robot	⇒	3	⇒	sin(3/1) = 0	cos(3/1) = −0.99	sin(3/10) = 0.30	cos(0) = 0.96

This has an explanation on the Fourier Transform

Fourier sine transform

- $f^s(\xi) = \int_{-\infty}^{\infty} f(t) \sin(2\pi\xi t)\, dt$

This has an explanation on the Fourier Transform

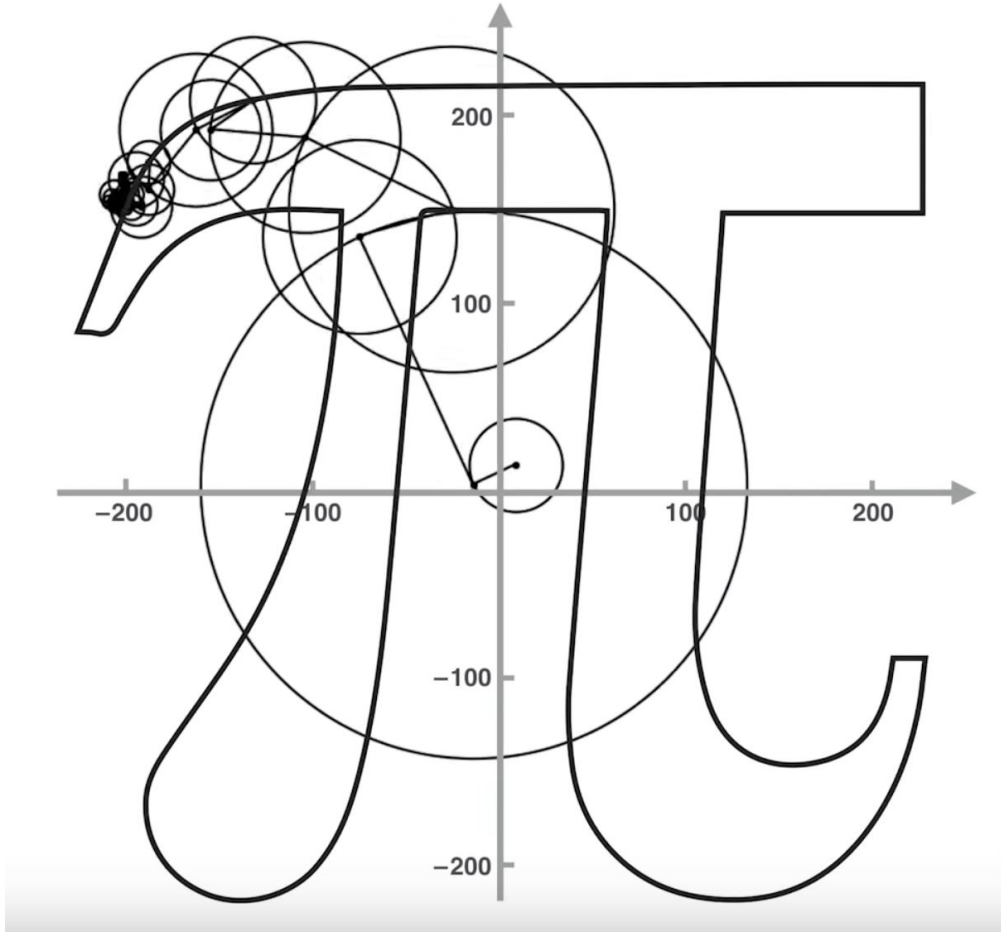
Fourier sine transform

- $f^s(\xi) = \int_{-\infty}^{\infty} f(t) \sin(2\pi\xi t)\, dt$

Fourier cosine transform

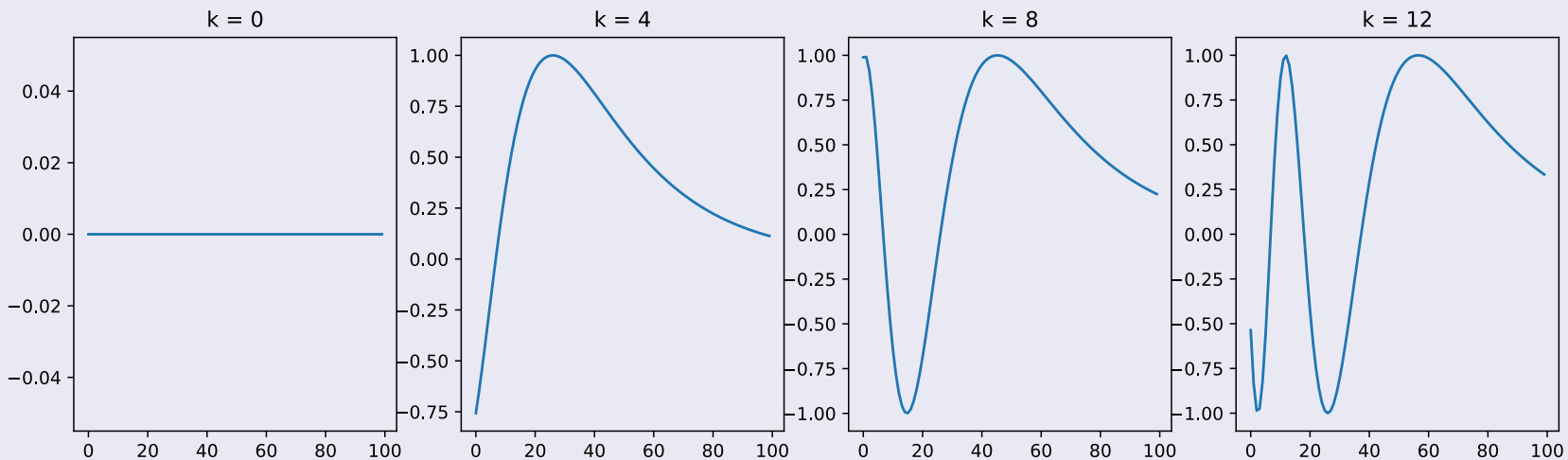
- $f^s(\xi) = \int_{-\infty}^{\infty} f(t) \cos(2\pi\xi t)\, dt$

After all, we can generate complex stuff



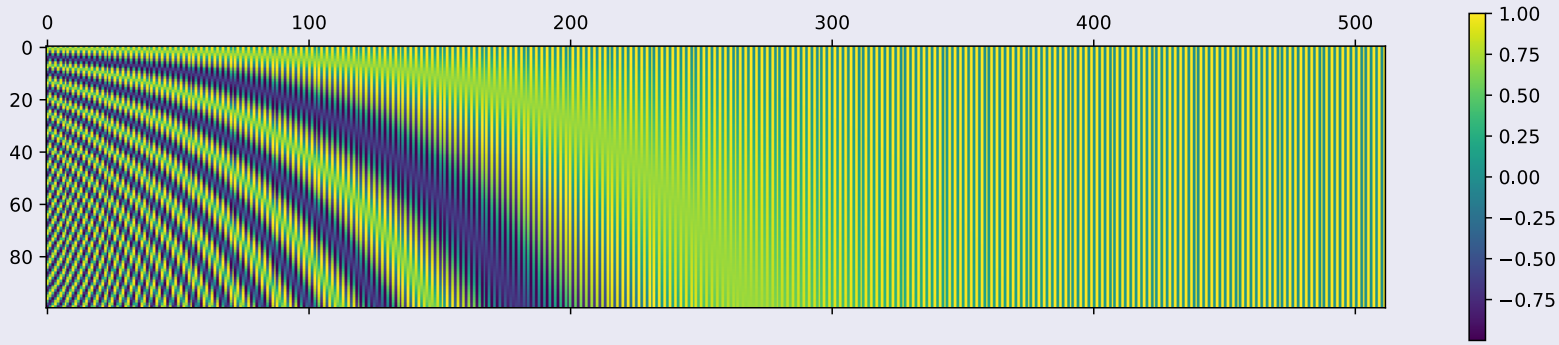
For example

We have the following Sine functions



An the Strange Vector Space

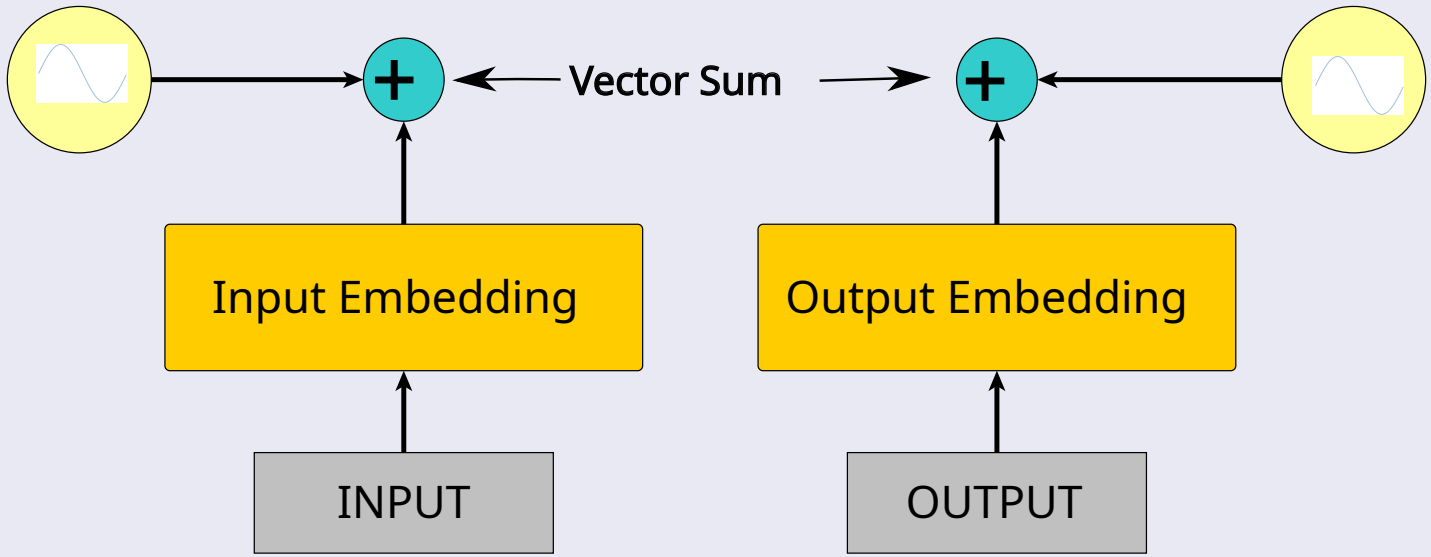
Something Notable



Outline

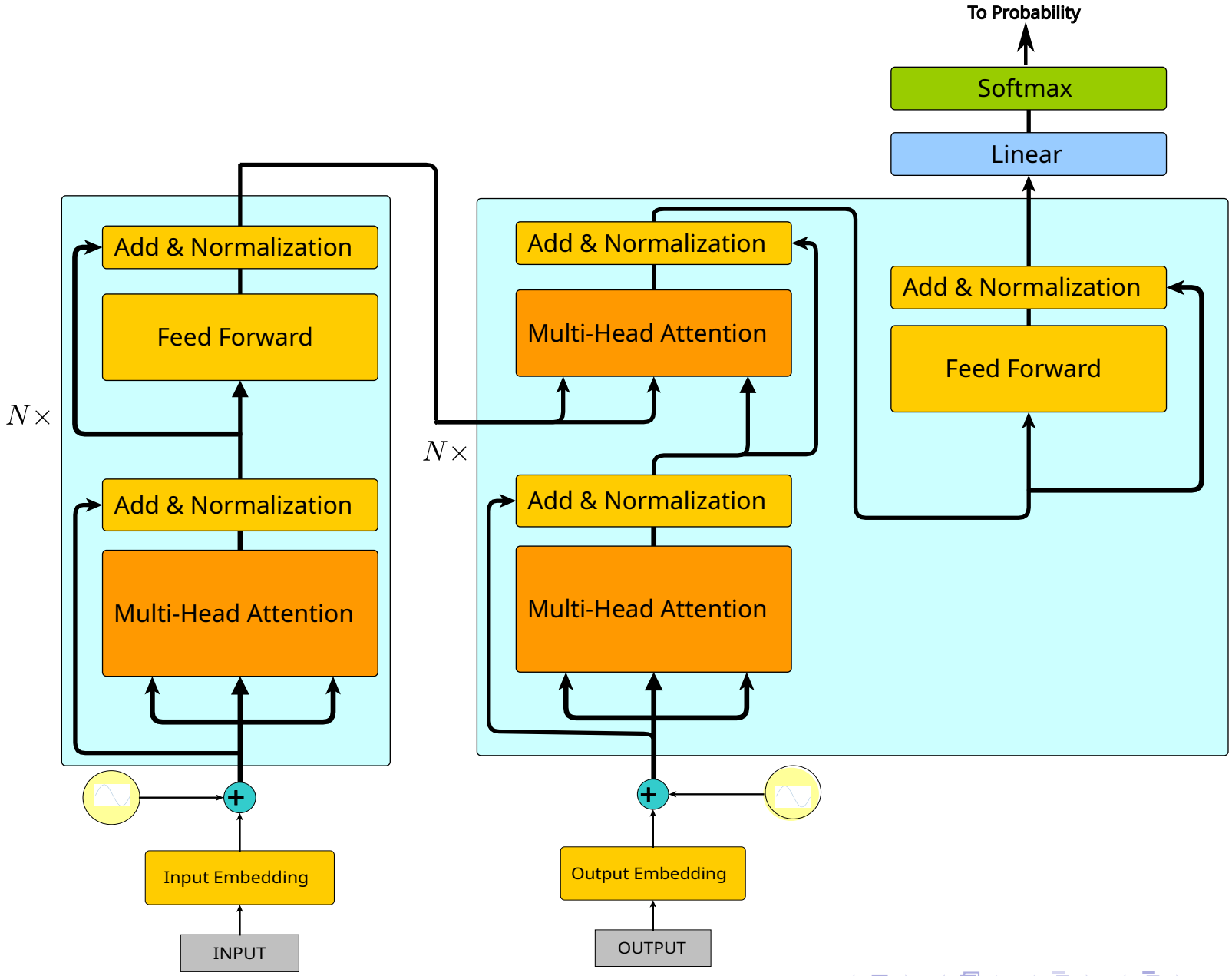
We have the following final input

Basically we take the Word embedding which has the same size that the Positional Encoding



- We simply sum it!!! Apart of some using of
  - BOS = begin of sentence marker
  - EOF = end of sentence marker
    - ★ The shifted right outputs

Finally, we have this!!! With cross-entropy at the top!!!



Outline

As Always, we need GPU processing

Comparison: NVIDIA H100 vs. NVIDIA A100 on the MosaicML Platform 7B-parameter MosaicGPT on 134B Tokens		
GPU	GPU Hours to Train	Approx. Cost
8 x H100 (bf16)	5,220	$24,850
8 x H100 (fp8)	4,100	$19,500
8 x A100	11,462	$25,300

However, under Quantization

Something Notable

- A new method named QLoRA enables the fine-tuning of large language models on a single GPU
  - QLoRA (Quantized Low Rank Adapters)

However, under Quantization

Something Notable

- A new method named QLoRA enables the fine-tuning of large language models on a single GPU
  - QLoRA (Quantized Low Rank Adapters)

The people at the University of Washington created Guanaco

- The largest Guanaco variant, with 65 billion parameters, achieves above 99 percent of the performance of ChatGPT (GPT-3.5-turbo) in a benchmark run with GPT-4.

We have

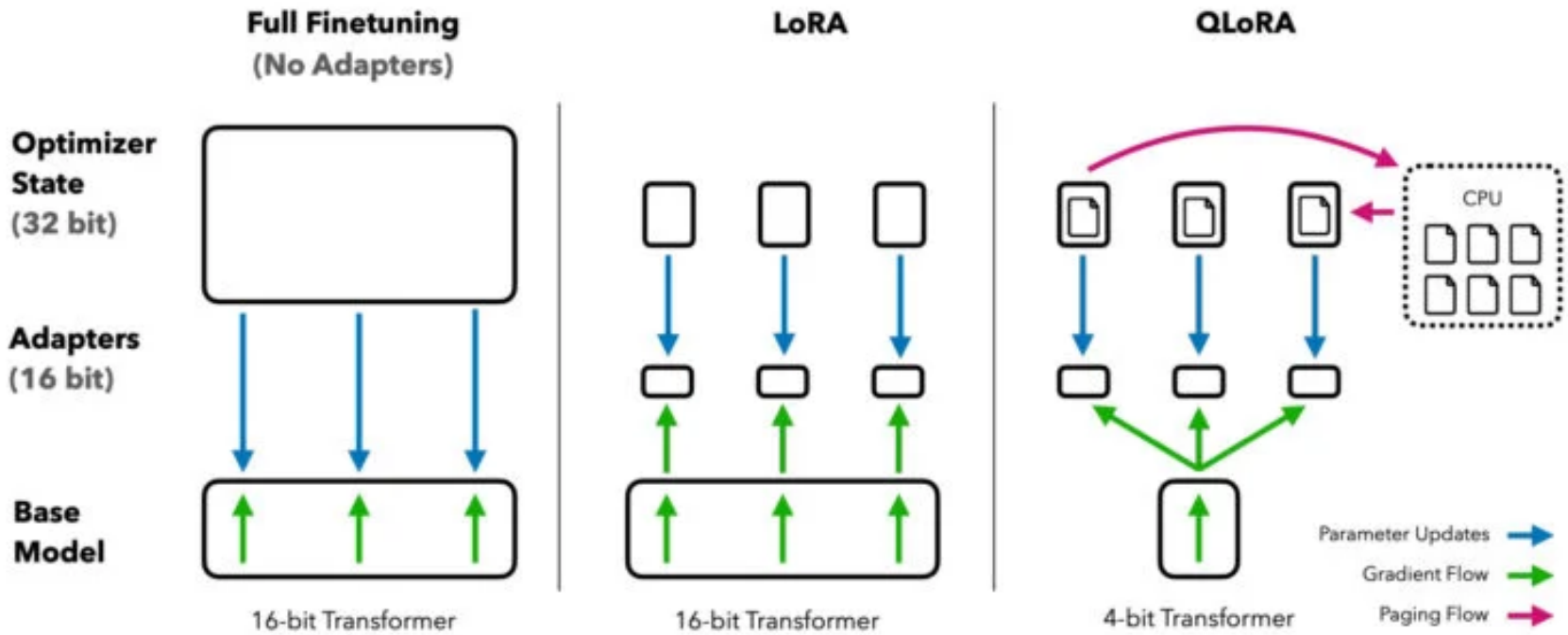


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

New Databases

Vector Databases

Conclusions

Transformers are the begining of a new Era

- With more heterogeneous architectures
- More strange Loss functions
- New quantization's
- An a new Era of better NLP models!!!


I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.


D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.


A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," 2014.


M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015.


A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.


J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," 2016.


K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *CoRR*, vol. abs/1502.03044, 2015.


G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern recognition*, vol. 65, pp. 211–222, 2017.


I. Schlag, P. Smolensky, R. Fernandez, N. Jojic, J. Schmidhuber, and J. Gao, "Enhancing the transformer with explicit relational encoding for math problem solving," *CoRR*, vol. abs/1910.06611, 2019.


S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.