Deep learning

Attention models

Hamid Beigy

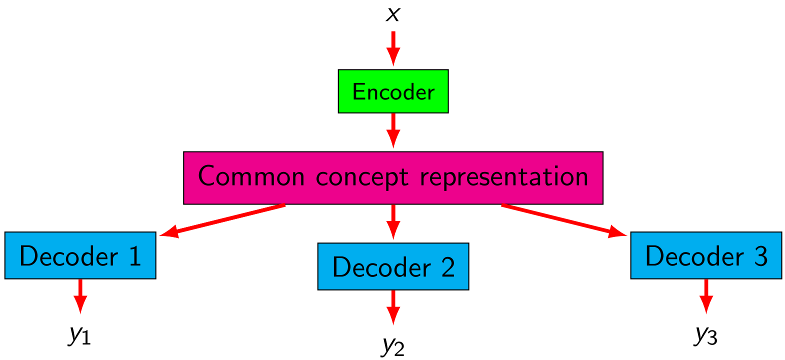
Sharif University of Technology

December 17, 2021

Table of contents

Introduction

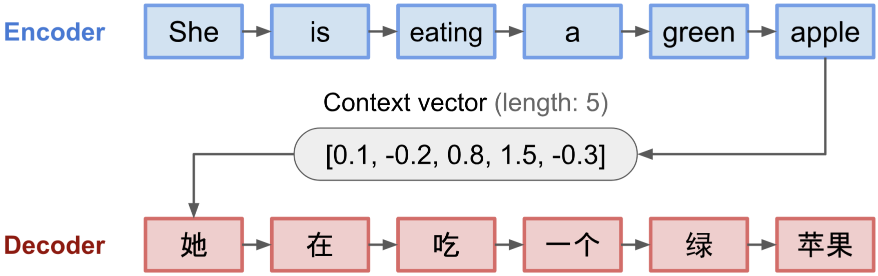
1. Consider the task of transferring a concept from a source domain to different target domains.



2. For example, consider the following tasks
   - A translation from Persian language to English language
   - A translation from Persian language to German language
   - A translation from Persian language to French language

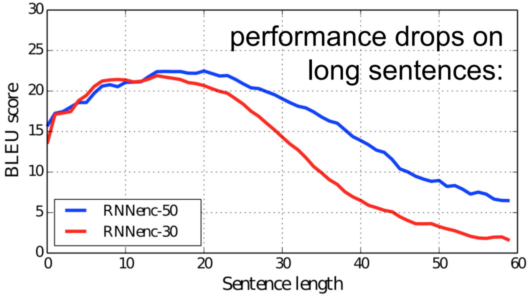
Sequence to sequence models

1. In seq2seq, the idea is to have two recurrent neural networks (RNNs) with an encoder-decoder architecture:
   - read the input words one by one to obtain a vector representation of a fixed dimensionality (encoder), and
   - conditioned on these inputs, extract the output words one by one using another RNN (decoder).
2. Both the encoder and decoder are recurrent neural networks such as LSTM or GRU units.



3. A critical disadvantage of this fixed-length context vector design is incapability of remembering long sentences.

1. RNNs cannot remember longer sentences and sequences due to the vanishing/exploding gradient problem.
2. The performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases.



3. In psychology, attention is the cognitive process of selectively concentrating on one or a few things while ignoring others.

Example (Counting the number of people in a photo)

Counting the number of heads and ignoring the rest.

1. Consider two different tasks : neural machine translation and image captioning.

neural machine translation (heatmap)

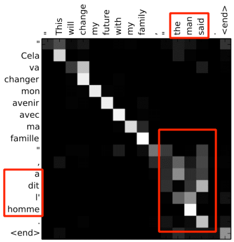
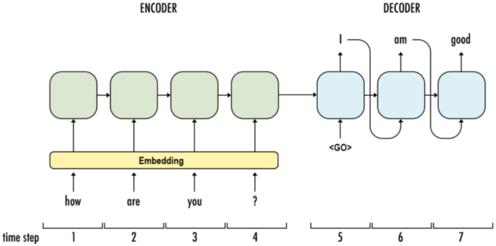


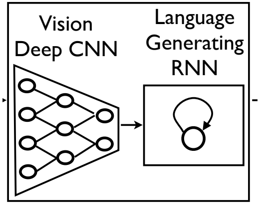
Image captioning

a red double decker bus driving down a street.

Neural network model



Neural network model



Attention models

1. The attention mechanism was born to help memorize long source sentences in neural machine translation (NMT) (Bahdanau, Cho, and Bengio 2015).
2. Instead of building a single context vector out of the encoder's last hidden state, the goal of attention is to create shortcuts between the context vector and the entire source input.
3. The weights of these shortcut connections are customizable for each output element.
4. The alignment between the source and target is learned and controlled by the context vector.
5. Essentially the context vector consumes three pieces of information:
    - Encoder hidden states
    - Decoder hidden states
    - Alignment between source and target

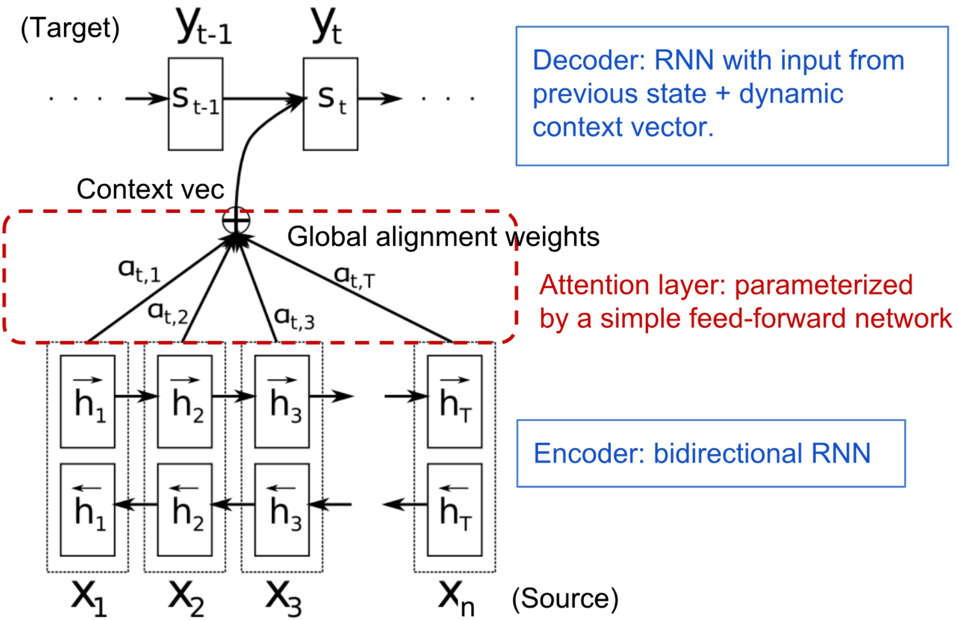
1. Assume that we have a source sequence x of length n and try to output a target sequence y of length m

$$x = [x_1, x_2, \ldots, x_n]$$
$$y = [y_1, y_2, \ldots, y_m]$$

2. The encoder is a bidirectional RNN with a forward hidden state $\overrightarrow{h}_i$ and a backward one $\overleftarrow{h}_i$.
3. A simple concatenation of these two hidden states represents the encoder state.
4. The motivation is to include both the preceding and following words in the annotation of one word.

$$\boldsymbol{h}_i = \left[\overrightarrow{\boldsymbol{h}}_i^\top ; \overleftarrow{\boldsymbol{h}}_i^\top\right]^\top \quad i = 1, 2, \ldots, n$$

1. Model of attention



1. The decoder network has hidden state $\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, y_{t-1}, \mathbf{c}_t)$ at position $t = 1, 2, \ldots, m$.
2. The context vector $\mathbf{c}_t$ is a sum of hidden states of the input sequence, weighted by alignment scores:

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

$$\alpha_{t,i} = \text{align}(y_t, x_i) \qquad \text{How well two words } y_t \text{ and } x_i \text{ are aligned.}$$

$$= \frac{exp\left(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i)\right)}{\sum_{j=1}^{n} \exp\left(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j)\right)} \qquad \text{Softmax of predefined alignment score.}$$

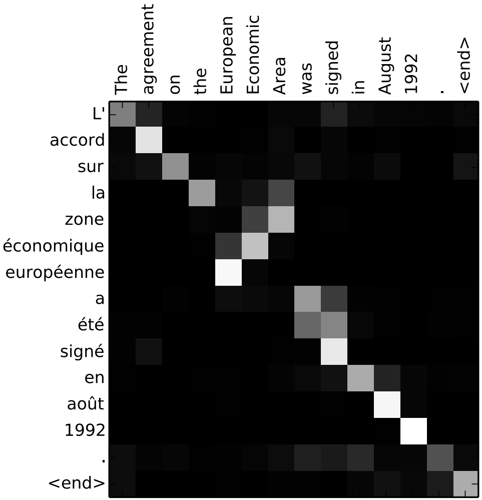
3. The alignment model assigns a score $\alpha_{t,i}$ to the pair of (y_t, x_i) based on how well they match.
4. The set of $\{\alpha_{t,i}\}$ are weights defining how much of each source hidden state should be considered for each output.

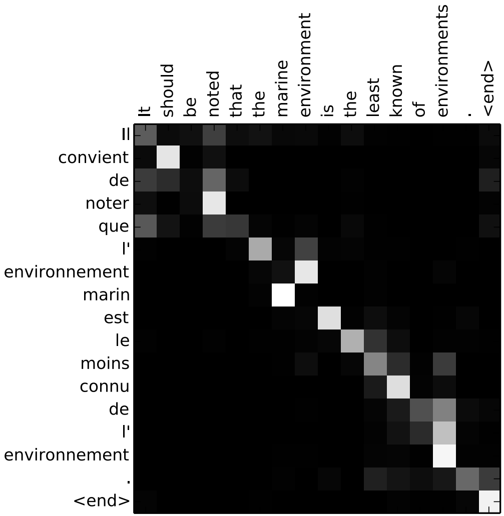
1. The alignment score α is parametrized by a feed-forward network with a single hidden layer (Bahdanau, Cho, and Bengio 2015).
2. This network is jointly trained with other parts of the model.
3. The score function is in the following form.

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

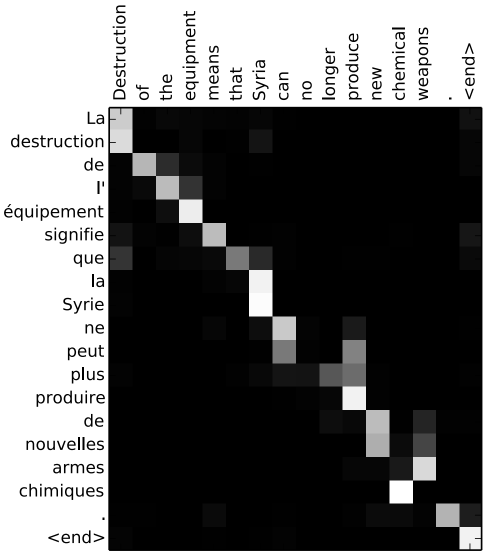
where both $\mathbf{V}_a$ and $\mathbf{W}_a$ are weight matrices to be learned in the alignment model.

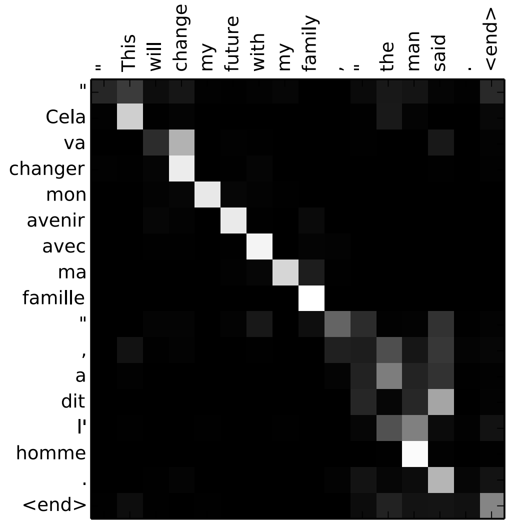
1. The matrix of alignment scores explicitly show the correlation between source and target words.





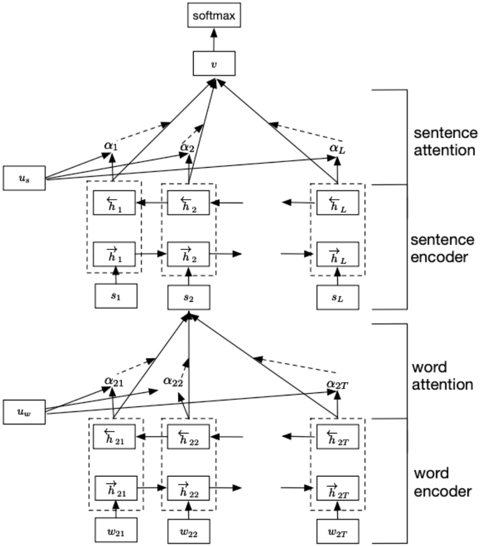
1. The matrix of alignment scores explicitly show the correlation between source and target words.





hierarchical attention network (HAN)

1. Attention can be effectively used on various levels (Yang et al. 2016).
2. HAN applicable to classification problem, not sequence generation.
3. HAN has two encoders: word and sentence.
   - Word encoder processes each word and aligns them a sentence of interest.
   - Then, sentence encoder aligns each sentence with final output.
4. HAN enables hierarchical interpretation of
   - which sentence is crucial in classifying document,
   - which part of a sentence (which words) are salient in that sentence.



1. Consider the following example

Example (Self-Attention)

- Consider the following sentence

  The animal didn't cross the street because it was too tired.

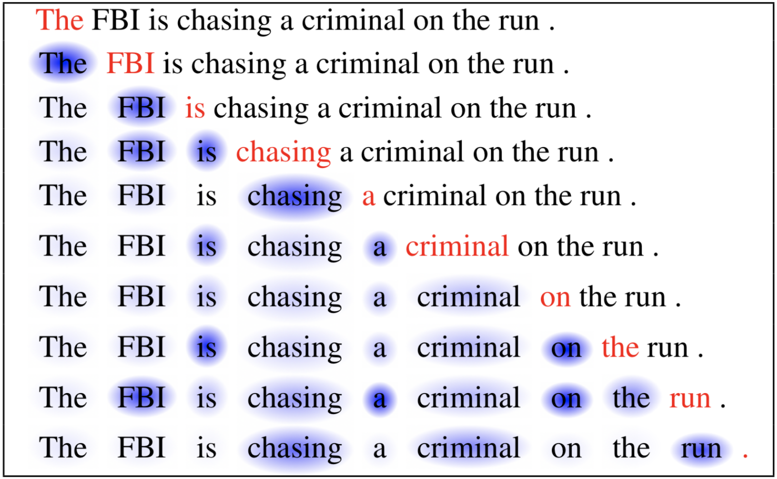
- What does **it** in this sentence refer to?

- Is it referring to **the street** or to **the animal**?

2. Self-attention (intra-attention) is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence (Cheng, Dong, and Lapata 2016).
3. It is very useful in
   - Machine reading (the automatic, unsupervised understanding of text)
   - Abstractive summarization
   - Image description generation

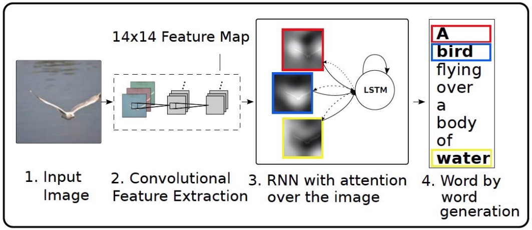
Self Attention

1. The self-attention mechanism enables us to learn the correlation between the current words and the previous part of the sentence.



2. The current word is in red and the size of the blue shade indicates the activation level.

1. Self-attention is applied to the image to generate descriptions (Xu et al. 2015).



2. Image is encoded by a CNN and a RNN with self-attention consumes the CNN feature maps to generate the descriptive words one by one.
3. The visualization of the attention weights clearly demonstrates which regions of the image, the model pays attention to output a certain word.

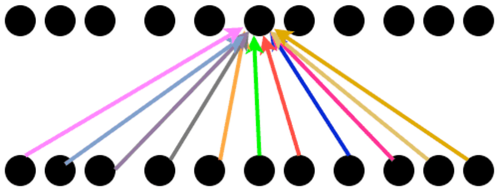
Self Attention

Soft vs Hard Attention

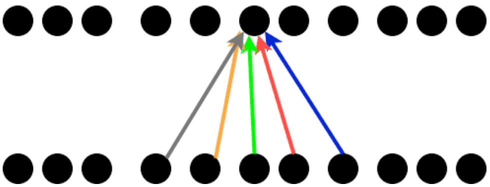
1. The soft vs hard attention is another way to categorize how attention is defined based on whether the attention has access to the entire image or only a patch.
   - Soft Attention: the alignment weights are learned and placed "softly" over all patches in the source image (same idea as in (Bahdanau, Cho, and Bengio 2015)).
     - Soft attention, in its simplest variant, is no different for images than for vector-valued features and is implemented exactly.
     - Pro: the model is smooth and differentiable.
     - Con: expensive when the source input is large.
   - Hard Attention: only selects one patch of the image to attend to at a time.
     - Hard attention for images has been known for a very long time: **image cropping**.
     - 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.

1. Global and local attention are proposed in (Luong, Pham, and Manning 2015).
2. The idea of a global attentional model is to consider all the hidden states of the encoder when deriving the context vector.

Global attention



Local attention



1. The global attention has a drawback that it has to attend to all words on the source side for each target word, which is expensive and can potentially render it impractical to translate longer sequences,
2. The local attentional mechanism chooses to focus only on a small subset of the source positions per target word.
3. Local one is an interesting blend between hard and soft, an improvement over the hard attention to make it differentiable:
4. The model first predicts a single aligned position for the current target word and a window centered around the source position is then used to compute a context vector.

$$\boldsymbol{p}_t = n \times \mathit{sigmoid}\left(\mathbf{v}_p^\top \tanh(\mathbf{W}_p \boldsymbol{h}_t)\right)$$

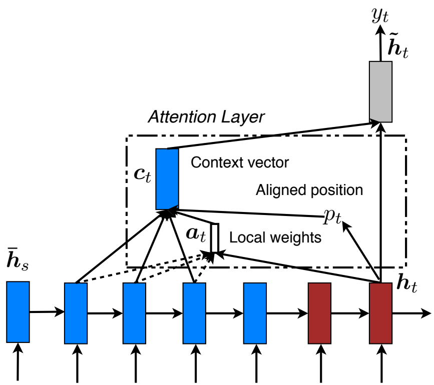
n is length of source sequence. Hence, $\boldsymbol{p}_t \in [0, n]$.

1. To favor alignment points near $\boldsymbol{p}_t$, they placed a Gaussian distribution centered around $\boldsymbol{p}_t$. Specifically, the alignment weights are defined as

$$a_{st} = align(h_t, \bar{h}_s) \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right)$$

and

$$\boldsymbol{p}_t = n \times sigmoid\left(\mathbf{v}_p^\top \tanh(\mathbf{W}_p \boldsymbol{h}_t)\right)$$



Generalized model of attention

1. Consider the following table, called PERSONS, in a relational database.

ID	Name	Family
005123174812	Ali	Ahmadi
015843268901	Mohammad Reza	Ali Mohammadi
005123174823	Ashkan	Mohammadi

2. Now consider the following queries.
   - SELECT ID, Name, Family FROM PERSONS WHERE ID='015843268901'
   - SELECT ID, Name, Family FROM PERSONS WHERE ID like '00512317%'
3. Here, concepts of query,key, and value become and the result is retrieved using the following similarity function.

$$Similarity(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \sum_i Similarity(\mathbf{q}, \mathbf{k}_i) \times \mathbf{v}_i$$

Retrieving a value from neural Turing machine memory

1. Consider the following memory in the neural Turing machine.

Key	Value
key 1	Value 1
key 2	Value 2
key 3	Value 3

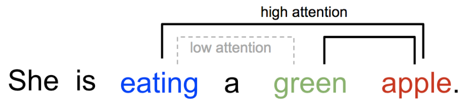
2. When reading from the memory at time t, an attention vector of size p, $\mathbf{w}_t$ controls how much attention to assign to different memory locations.
3. The read vector $\mathbf{r}_t$ is a sum weighted by attention intensity:

$$\mathbf{r}_t = \sum_{i=1}^{p} w_t(i)\mathbf{M}_t(i)$$

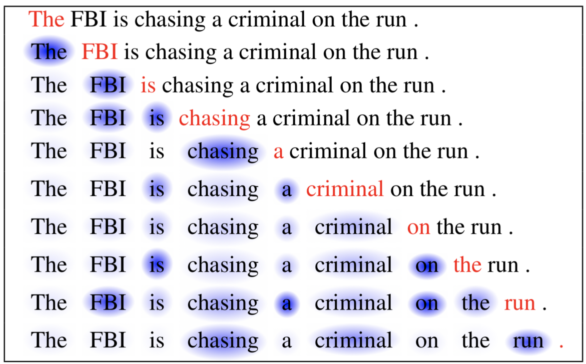
$$\sum_{i=1}^{p} w_t(i) = 1, \forall i : 0 \leq w_t(i) \leq 1$$

Generalized model of attention

1. Consider the following sentence.



2. For calculating the attention of a target word with respect to the input word,
   - we first use the query of the target word and the key of the input word,
   - next calculate a matching score, and
   - finally calculate the weighted sum of value vectors using the matching scores.

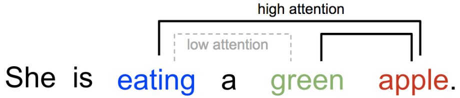


Generalized model of attention

1. Each word is key, query and value.
2. Each word w is represented by a vector $\mathbf{x} \in \mathbb{R}^d$ by using an embedding method.
3. Calculate query ($\mathbf{q} \in \mathbb{R}^p$) for $\mathbf{x} \in \mathbb{R}^d$, which is projection of $\mathbf{x}$ to a new space.
$$\mathbf{q} = \mathbf{w}_q^\top \mathbf{x}.$$
4. Calculate key ($\mathbf{k} \in \mathbb{R}^p$) for $\mathbf{x} \in \mathbb{R}^d$, which is projection of $\mathbf{x}$ to a new space.
$$\mathbf{k} = \mathbf{w}_k^\top \mathbf{x}.$$
5. Calculate value ($\mathbf{w} \in \mathbb{R}^p$) for $\mathbf{x} \in \mathbb{R}^d$, which is projection of $\mathbf{x}$ to a new space.
$$\mathbf{v} = \mathbf{w}_v^\top \mathbf{x}.$$
6. A single word $\mathbf{x}$ has three different representations. Sometimes, we look at this word as query, sometimes as key, and sometimes as value.
7. The self-attention means that looking a word as query and compute the similarity of the query with all of the words seen as key.
8. Then use the softmax for computing the weights and compute the weighted average all of the words seen as value.
9. This computes the attention vector.

Generalized model of attention

1. Consider the following sentence.



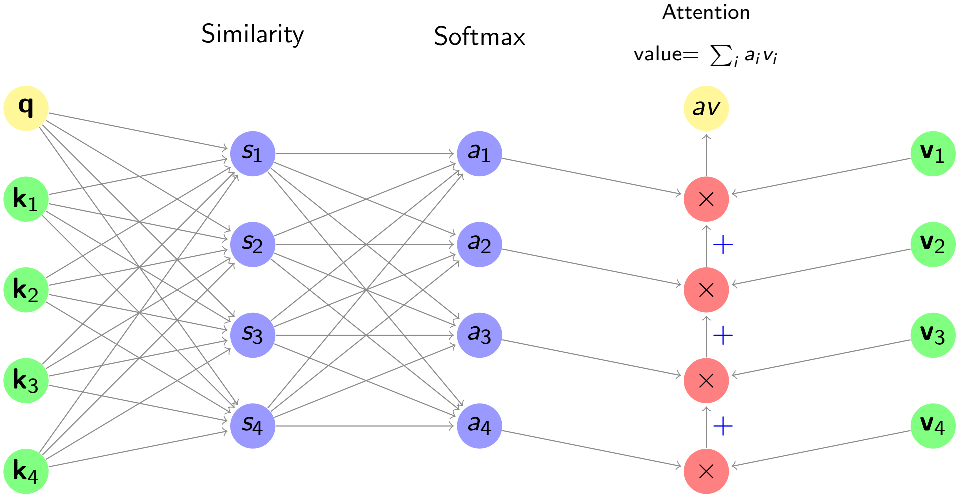
2. Calculating the attention for word apple.
3. Taking the inner product of the query vector of apple to the key vector of the previous words.

$$\mathbf{a} = softmax\left(\mathbf{q}_{apple}^{\top}\mathbf{k}_{she}, \mathbf{q}_{apple}^{\top}\mathbf{k}_{is}, \mathbf{q}_{apple}^{\top}\mathbf{k}_{eating}, \mathbf{q}_{apple}^{\top}\mathbf{k}_{a}, \mathbf{q}_{apple}^{\top}\mathbf{k}_{green}\right)$$

4. Suppose that we obtain $\mathbf{a} = (0.1, 0.1, 0.5, 0.1, 0.2)$. Then we obtain

$$\mathbf{v}_{apple} = 0.1\mathbf{v}_{she} + 0.1\mathbf{v}_{is} + 0.5\mathbf{v}_{eating} + 0.1\mathbf{v}_{a} + 0.2\mathbf{v}_{green}$$

1. Self-attention uses the following neural network architecture.

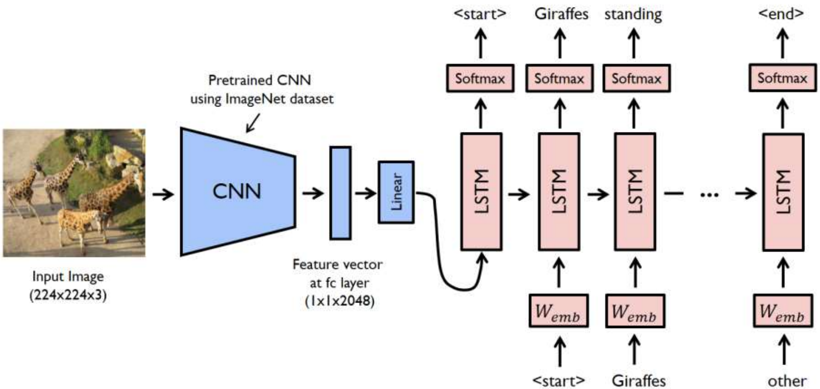


1. By defining three different vectors corresponding to each word.
   - Key $\mathbf{k} \in \mathbb{R}^p$ and $\mathbf{k} = \mathbf{W}_k^\top \mathbf{x}$, where $\mathbf{W}_k \in \mathbb{R}^{d \times p}$ and $\mathbf{x} \in \mathbb{R}^d$.
   - Query $\mathbf{q} \in \mathbb{R}^p$ and $\mathbf{q} = \mathbf{W}_q^\top \mathbf{x}$, where $\mathbf{W}_q \in \mathbb{R}^{d \times p}$ and $\mathbf{x} \in \mathbb{R}^d$.
   - Value $\mathbf{v} \in \mathbb{R}^p$ and $\mathbf{v} = \mathbf{W}_v^\top \mathbf{x}$, where $\mathbf{W}_v \in \mathbb{R}^{d \times p}$ and $\mathbf{x} \in \mathbb{R}^d$.
2. By defining the following matrices
   - $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$, where $\mathbf{X} \in \mathbb{R}^{d \times n}$.
   - $\mathbf{K} = [\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_n]$, where $\mathbf{K} \in \mathbb{R}^{p \times n}$.
   - $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n]$, where $\mathbf{Q} \in \mathbb{R}^{p \times n}$.
   - $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n]$, where $\mathbf{V} \in \mathbb{R}^{p \times n}$.
3. Then, the new value $\mathbf{Z} \in \mathbb{R}^{p \times n}$ equals to $\mathbf{Z} = \mathbf{V}\ \mathit{Softmax}\ \left(\frac{\mathbf{Q}^\top \mathbf{K}}{\sqrt{p}}\right)$.

Name	Alignment score function	Paper (https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html)
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	A. Graves, et al. "Neural Turing machines", arXiv, 2014.
Additive	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	D. Bahdanau, et al."Neural machine translation by jointly learning to align and translate", ICLR 2015.
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$	T. Luong, , et al. "Effective Approaches to Attention-based Neural Machine Translation", EMNLP 2015.
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$	Same as the above
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Same as the above
Scaled Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$	A. Vaswani, et al. "Attention is all you need", NIPS 2017.

Attention in computer vision

1. The natural image caption generator was proposed in (Xu et al. 2015).



2. This network is a combination of CNN and LSTM networks.

1. The outputs of lower layers of CNN are used as representation of values.

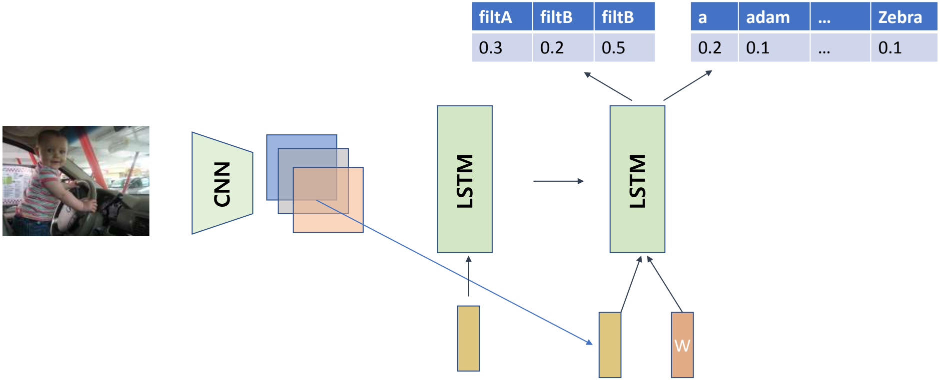


Image captioning results

1. Examples of attending to the correct object

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

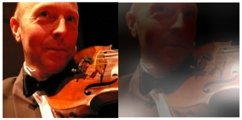
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

2. Examples of mistakes

A large white bird standing in a forest.

A woman holding a clock in her hand.



A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

Image captioning results

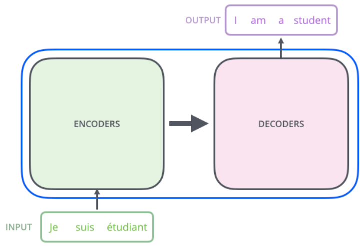
1. There is also a method given in (Vinyals et al. 2015).

Transformers family

Transformers family

Transformers model

1. The soft attention and make it possible to do sequence to sequence modeling without recurrent network units (Vaswani et al. 2017).
2. The transformer model is entirely built on the self-attention mechanisms without using sequence-aligned recurrent architecture.



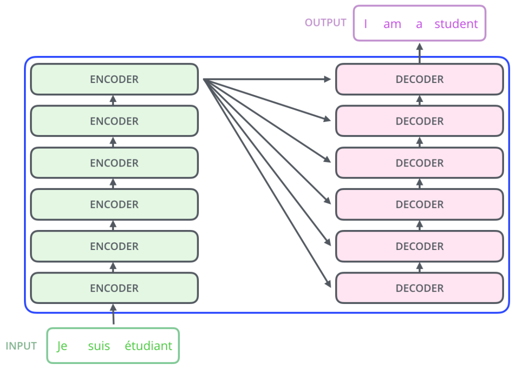


Figure: Jay Alammar

3. The encoding component is a stack of six encoders.
4. The decoding component is a stack of decoders of the same number.

Transformers training

1. The Transformers works slightly differently during training and inference.
2. Input sequence: You are welcome in English, target sequence: De nada in Spanish

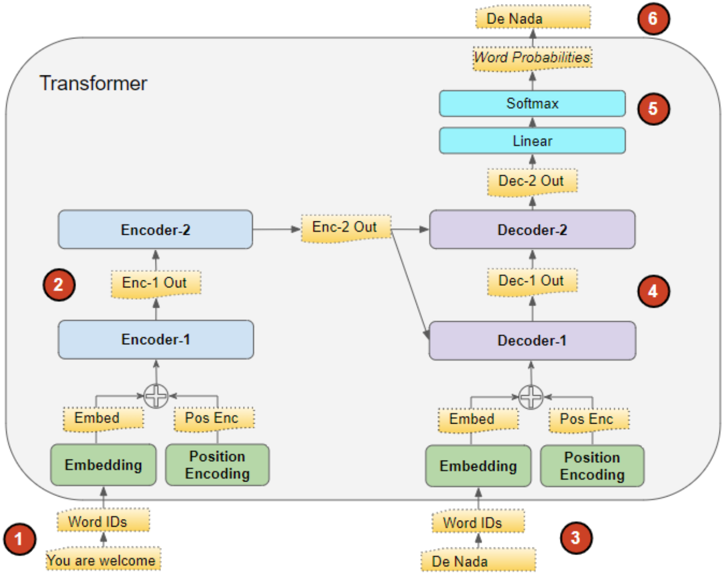


Figure:Ketan Doshi

1. During Inference, we have only the input sequence and don't have the target sequence to pass as input to the Decoder.
2. The goal is to produce the target sequence from the input sequence alone.

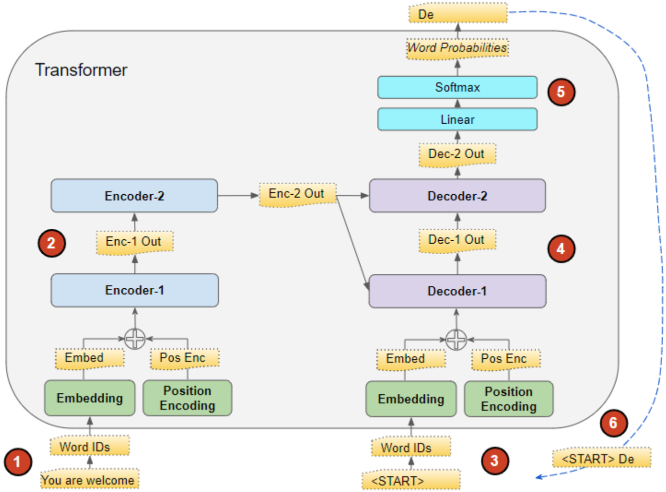
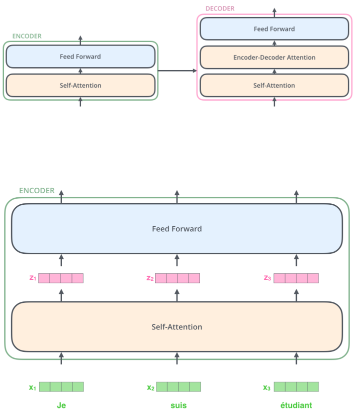


Figure:Ketan Doshi

1. Each encoder has two sub-layers and each decoder has three sub-layers
2. Each sublayer has residual connection.
3. All encoders receive a list of vectors each of the size 512.
4. The size of this list is hyper-parameter we can set (it would be the length of the longest sentence in our training dataset).



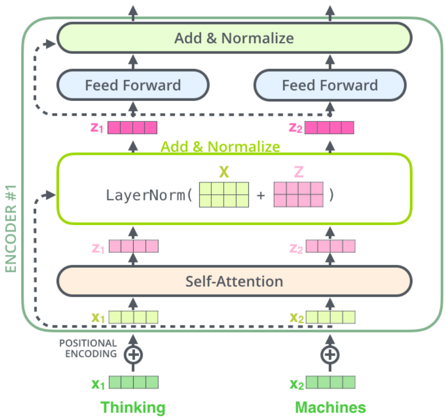


Figure: Jay Alammar

1. A transformer of two stacked encoders and decoders

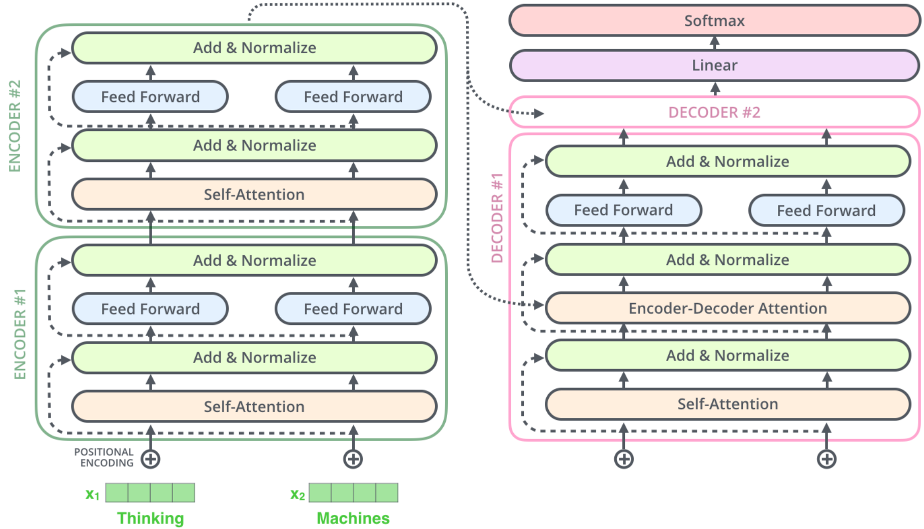


Figure: Jay Alammar

Transformers embedding and position encoding

1. Transformers needs two things for a word: its meaning and its position in sequence.
2. The Transformers has two Embedding layers.
   - Input sequence is fed to the first embedding layer (Input Embedding).
   - Target sequence is fed to the second embedding layer after shifting the targets right by one position and inserting a Start token in the first position.

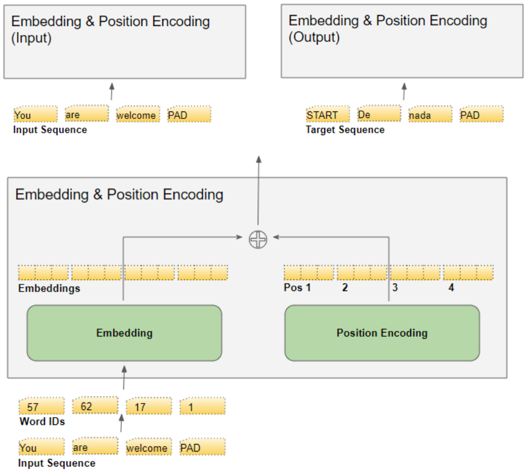


Figure:Ketan Doshi

1. There are two position encoding layers for: input sequence and output sequence.
2. Let d be size of embedding for each word and L be length of input sequence.
3. Transformers considers an array of $d \times L$ to encode positions of the input sequence.

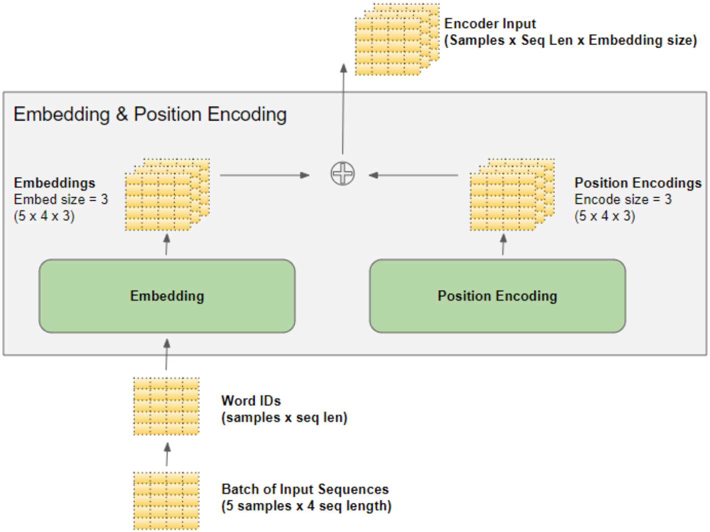


Figure:Ketan Doshi

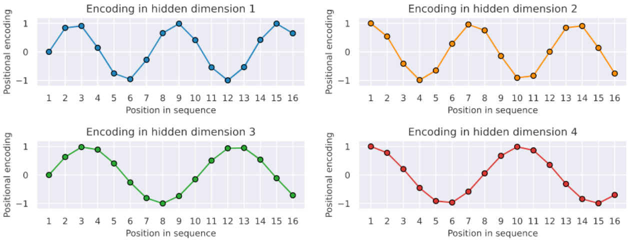
Transformers position encoding

1. Let pos be the position of word in sequence and i be the index value in positional encoding. Then, PE is computed using

$$PE(pos, i) = \begin{cases} \sin\left(\frac{pos}{10000^{i/d}}\right) & \text{if } i = 2k \\ \cos\left(\frac{pos}{10000^{i/d}}\right) & \text{if } i = 2k+1 \end{cases}$$

2. For word w at position $pos \in [0, L-1]$ in the input sequence $w = (w_0, \ldots, w_{L-1})$, with 4-dimensional embeddinge_w, and $d = d_{model} = 4$, the operation would be

$$\begin{aligned} e'_w &= e_w + \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right)\right] \\ &= e_w + \left[sin\left(pos\right), cos\left(pos\right), sin\left(\frac{pos}{100}\right), cos\left(\frac{pos}{100}\right)\right] \end{aligned}$$



1. Position encoding interleaves a *sine* curve and a *cos* curve, with sine values for all even indexes and cos values for all odd indexes.
2. This results the following position encoding and the corresponding curves.

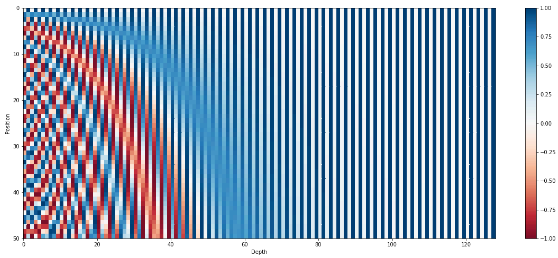


Figure: Amirhossein Kazemnejad

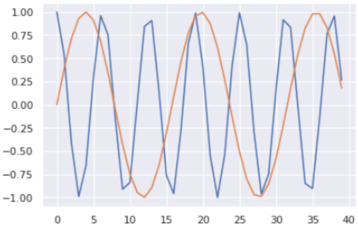
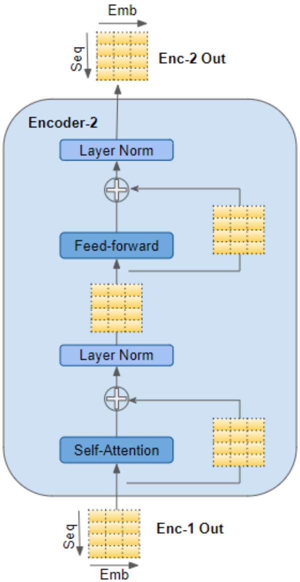


Figure: Ketan Doshi

1. The Encoder passes its input into a Multi-head Self-attention layer.
2. The Self-attention output is passed into a Feed-forward layer, which then sends its output upwards to the next Encoder.



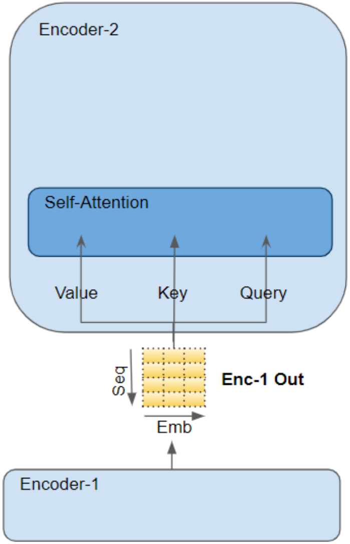
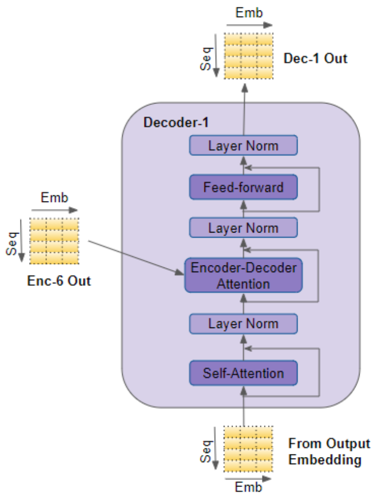


Figure: Ketan Doshi

1. The Decoder passes its input into a Multi-head Self-attention layer.
2. This operates in a slightly different way than the one in the Encoder.
3. It is only allowed to attend to earlier positions in the sequence. This is done by masking future positions.



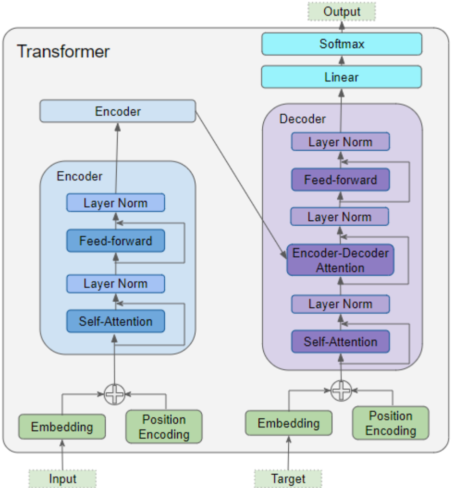
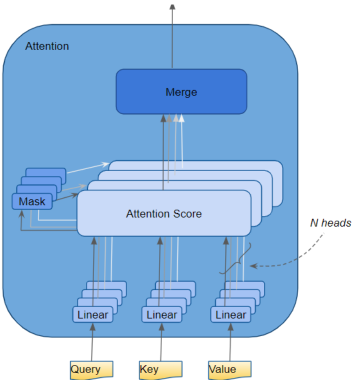


Figure: Ketan Doshi

1. The Transformers calls each Attention processor an Attention Head and repeats it several times in parallel.
2. This is known as Multi-head attention.
3. It gives its Attention greater power of discrimination, by combining several similar Attention calculations.



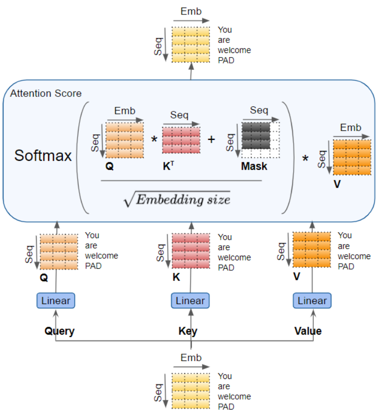


Figure: Ketan Doshi

Transformers multi-head attention

1. There are three separate Linear layers for the Query, Key, and Value.
2. Each Linear layer has its own weights.
3. The input is passed through these Linear layers to produce the **Q**, **K**, and **V** matrices.

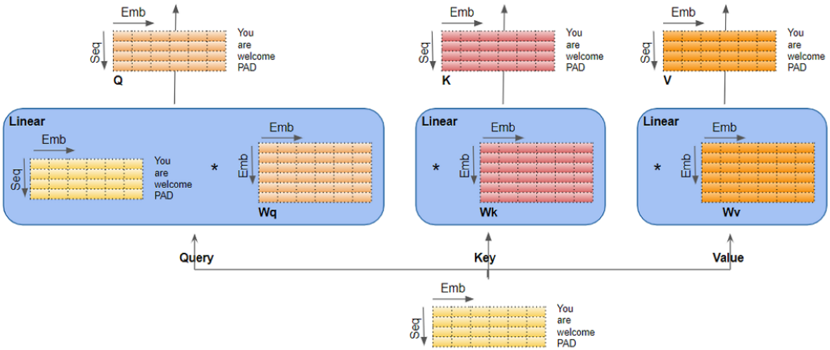


Figure: Ketan Doshi

1. The data are split across the multiple Attention heads so that each can process it independently.
2. This is a logical split only. The Query, Key, and Value are not physically split into separate matrices, one for each Attention head.
3. A single data matrix is used for the Query, Key, and Value, respectively, with logically separate sections of the matrix for each Attention head.

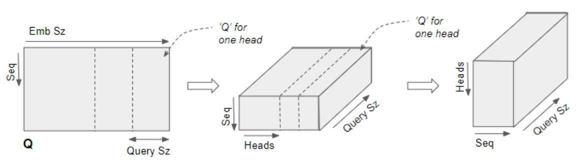
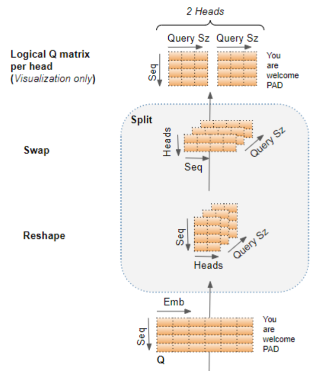


Figure: Ketan Doshi



1. We now have separate Attention Scores for each head.
2. They need to be combined together into a single score.
3. This Merge operation is essentially the reverse of the Split operation.
4. It is done by simply reshaping the result matrix to eliminate the Head dimension.
   - Reshape the Attention Score matrix by swapping the Head and Sequence dimensions.
   - Collapse the Head dimension by reshaping .

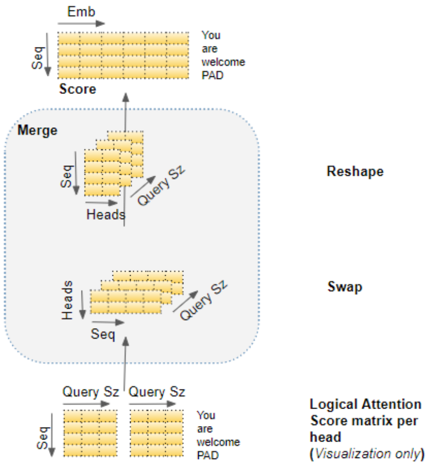


Figure: Ketan Doshi

1. The end-to-end flow of the Multi-head Attention is

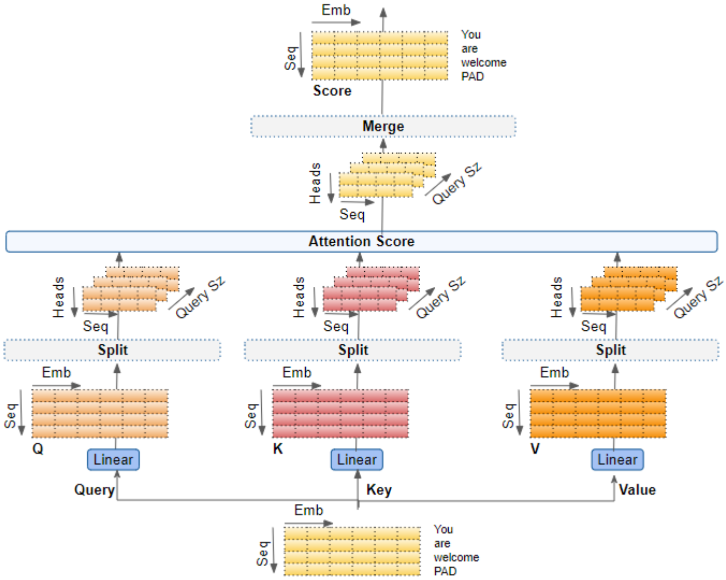


Figure: Ketan Doshi

1. The different attention heads are focusing on different words as we encode the word it.

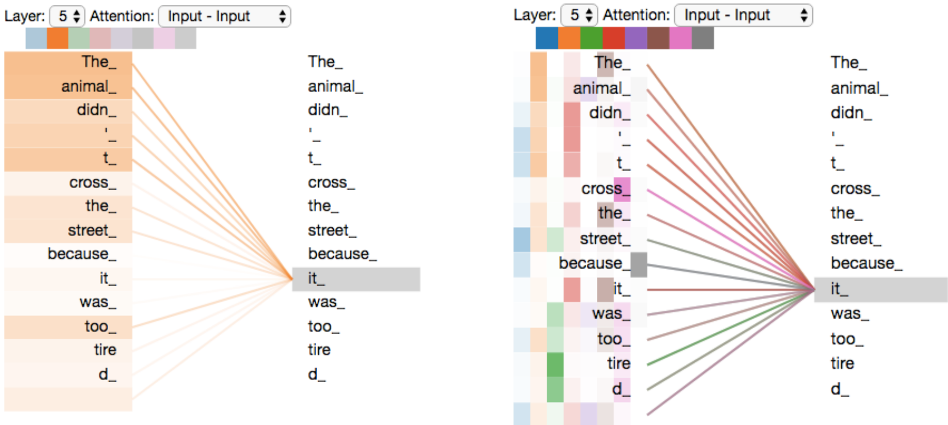


Figure: Jay Alammar

1. The attention layers of Transformers decoder are

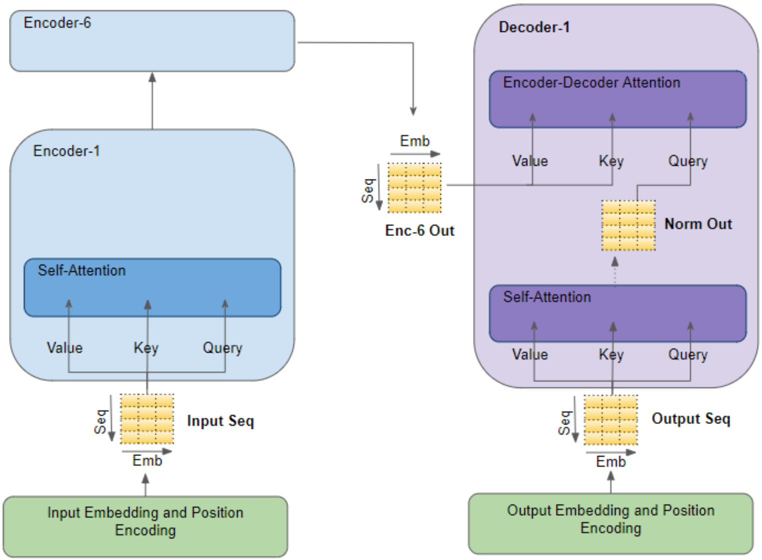


Figure: Ketan Doshi

Transformers decoder self-attention and masking

1. The Decoder Self-Attention works just like the Encoder Self-Attention, except that it operates on each word of the target sequence.

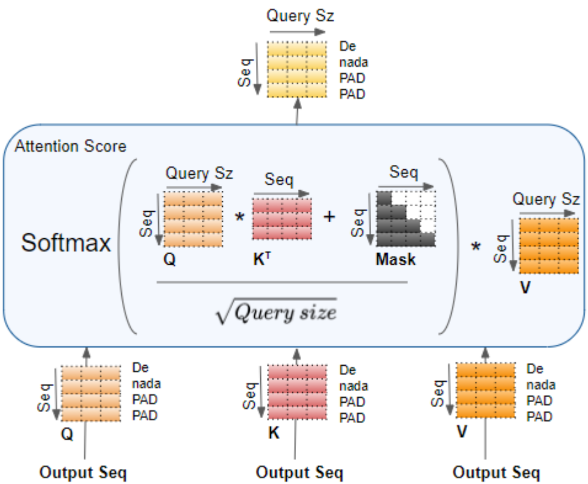


Figure: Ketan Doshi

1. The Encoder-Decoder Attention takes its input from two sources.
2. The Encoder-Decoder Attention computes the interaction between each target word with each input word.
3. The Masking masks out the Padding words in the target sequence.

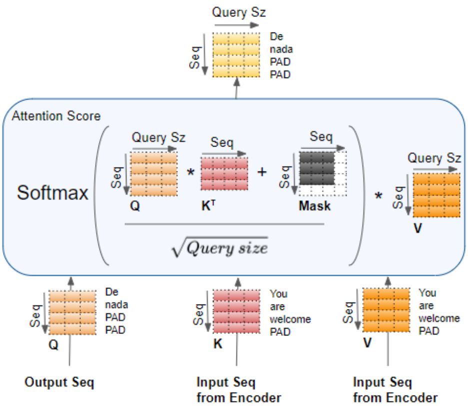


Figure: Ketan Doshi

Simple Neural Attention Meta-Learner (SNAIL)

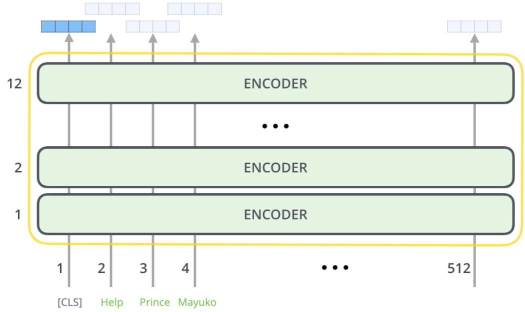
1. The SNAIL was developed partially to resolve the problem with positioning in the transformer model by combining the self-attention mechanism in transformer with convolutions (Mishra et al. 2018).
2. It has been demonstrated to be good at both supervised learning and reinforcement learning tasks.

Transformers family

BERT model

BERT model

1. BERT (Pre-training of Deep Bidirectional Transformers for Language Understanding) is basically a trained Transformers Encoder stack (Devlin et al. 2019).
2. Each position outputs a vector. For the sentence classification, we focus on the output of only the first position ([CLS]).
3. That vector can now be used as the input for a classifier. The paper achieves great results by just using a single-layer neural network as the classifier.



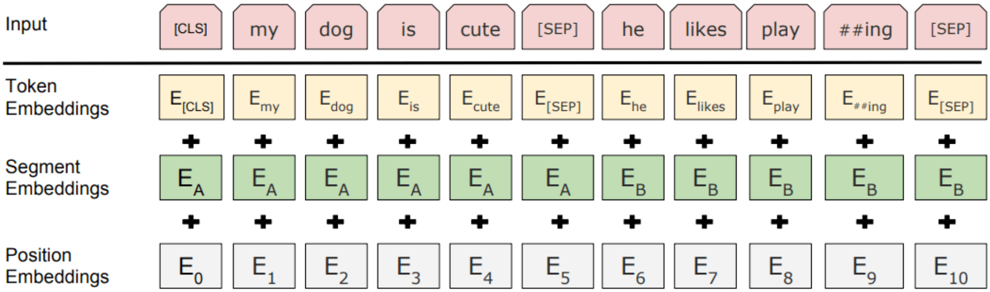
4. BERT makes use of a novel technique called Masked LM (MLM): it randomly masks words in the sentence and then it tries to predict them.

1. BERT needs the input to be massaged and decorated with some extra meta data:

   Token embeddings A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

   Segment embeddings A marker indicating Sentence A or Sentence B is added to each token. This allows the encoder to distinguish between sentences.

   Positional embeddings A positional embedding is added to each token to indicate its position in the sentence.



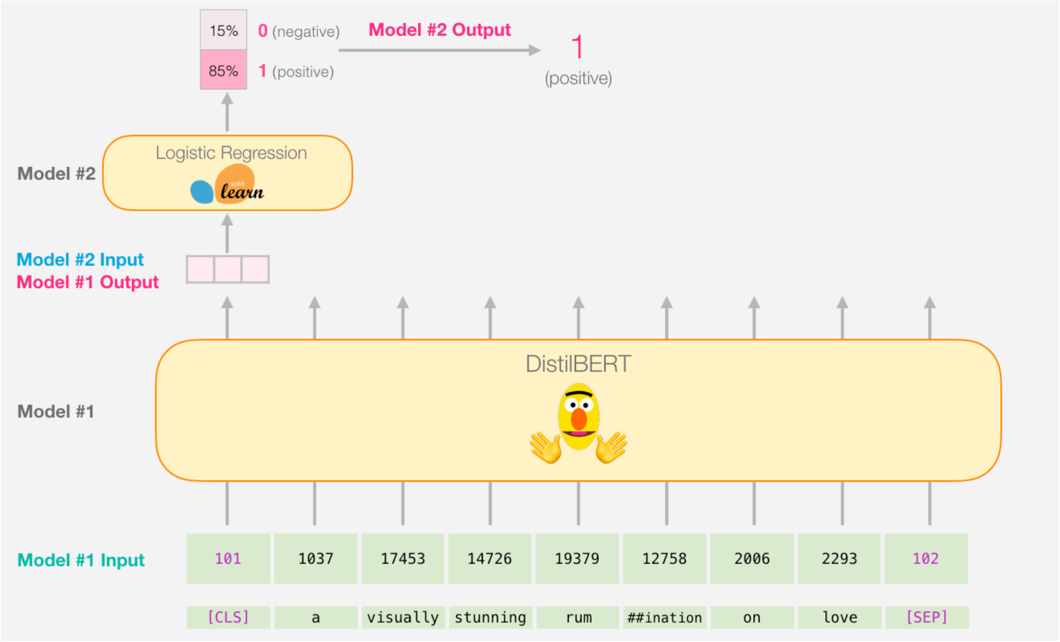


Figure: Jay Alammar

Training BERT using masked language model

1. Randomly mask out 15% of the words in the input (replacing them with a [MASK] token) .
2. Then run the entire sequence through the BERT attention based encoder and predict only the masked words, based on the context provided by the other non-masked words in the sequence.
3. The problem here is : the model only tries to predict when the [MASK] token is present in the input, while we want the model to try to predict the correct tokens regardless of what token is present in the input.
4. To deal with this issue, out of the 15% of the tokens selected for masking:
   - 80% of the tokens are actually replaced with the token [MASK].
   - 10% of the time tokens are replaced with a random token.
   - 10% of the time tokens are left unchanged.

Training BERT using next Sentence prediction model

1. To understand relationship between two sentences, BERT training process also uses next sentence prediction.
2. A pre-trained model with this kind of understanding is relevant for tasks like question answering.
3. During training the model gets as input pairs of sentences and it learns to predict if the second sentence is the next sentence in the original text as well.
4. BERT separates sentences with a special [SEP] token.
5. During training the model is fed with two input sentences at a time such that
   - 50% of the time the second sentence comes after the first one.
   - 50% of the time it is a a random sentence from the full corpus.
6. BERT is then required to predict whether the second sentence is random or not.
7. To predict if the second sentence is connected to the first one or not, the output of the [CLS] token is given to a classifier.

1. There are two types of pre-trained versions of BERT depending on the scale of the model architecture

   BERT-Base 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters.

   BERT-Large 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters.

Transformers family

GPT-2 model

GPT-2 model

1. The GPT-2 is built using transformer decoder blocks (Radford et al. 2019).
2. BERT uses transformer encoder blocks.
3. A key difference between the two is that GPT2 outputs one token at a time.

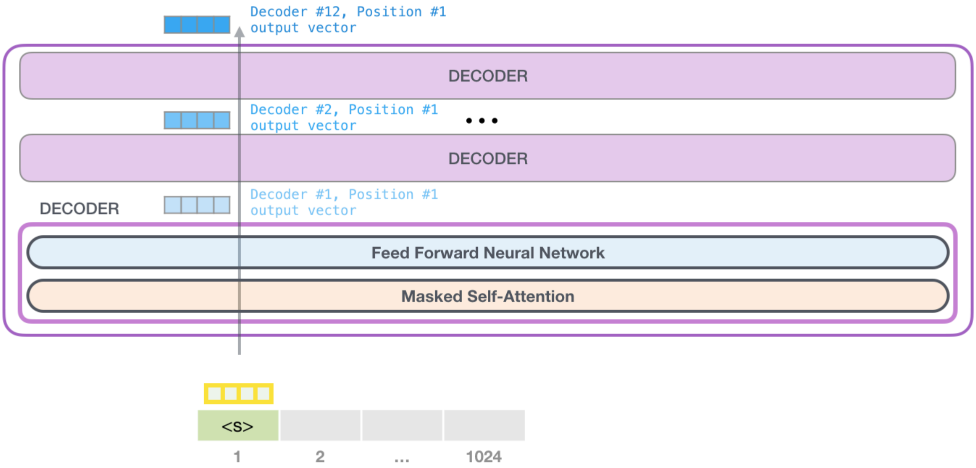


Figure: Jay Alammar

GPT-2 pre-trained architecture

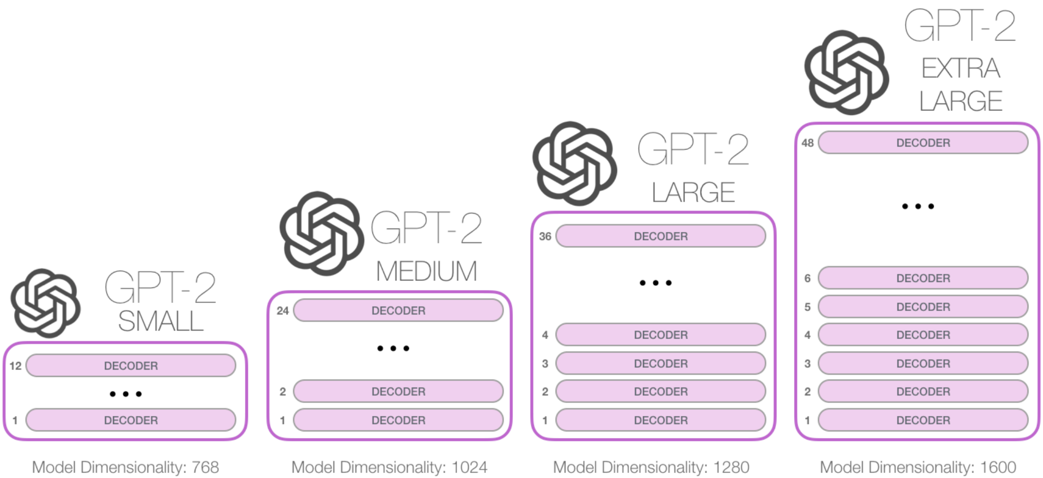
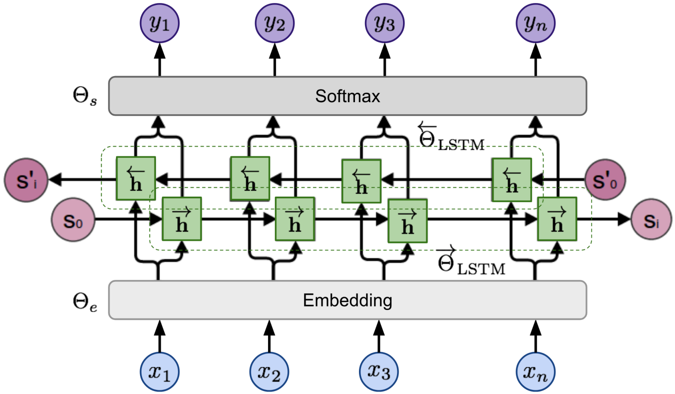


Figure: Jay Alammar

ELMo model

1. ELMo learns contextualized word representation by pre-training a language model in an unsupervised way (Peters et al. 2018).



Embeddings from Language Model (ELMo)

1. The bidirectional Language Model (biLM) is the foundation for ELMo.
2. While the input is a sequence of n tokens, $(x_1, \ldots, x_n)$, the language model learns to predict the probability of next token given the history.
3. In the forward pass, the history contains words before the target token,

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})$$

4. In the backward pass, the history contains words after the target token,

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i+1}, \ldots, x_n)$$

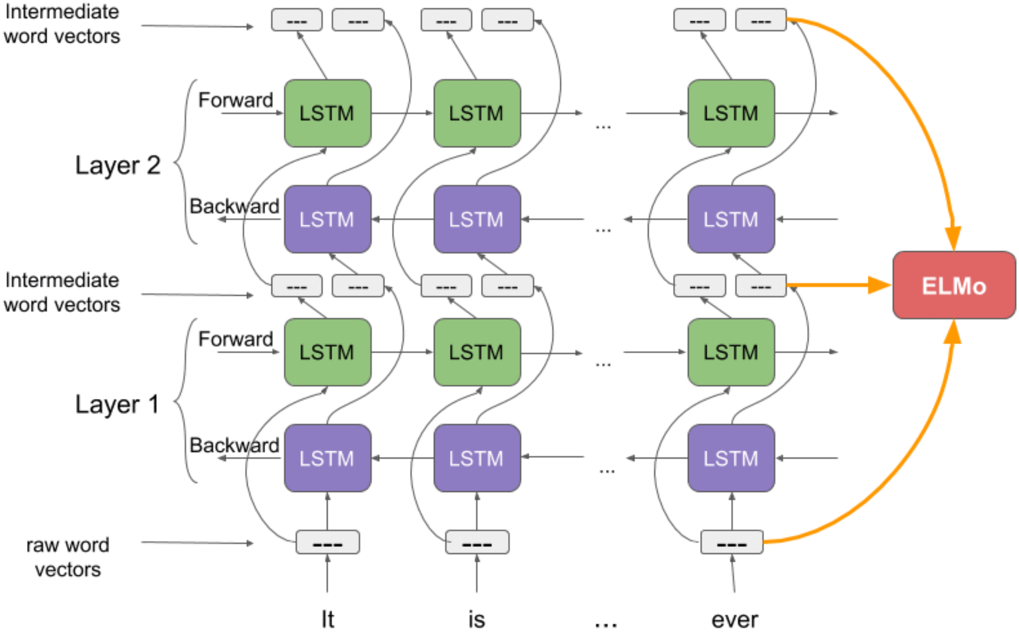
5. The predictions in both directions are modeled by multi-layer LSTMs with hidden states.

1. The model is trained to minimize the negative log likelihood (= maximize the log likelihood for true words) in both directions:

$$\mathcal{L} = -\sum_{i=1}^{n} \Big(\log p(x_i \mid x_1, \ldots, x_{i-1}; \Theta_e, \overrightarrow{\Theta}_{\text{LSTM}}, \Theta_s) + \\ \log p(x_i \mid x_{i+1}, \ldots, x_n; \Theta_e, \overleftarrow{\Theta}_{\text{LSTM}}, \Theta_s) \Big)$$

2. ELMo word representations are functions of the entire input sentence.
3. A linear combination of the vectors stacked above each input word is learned as the representation of each token.

Embeddings from Language Model (ELMo)



Reading

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations.*

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). "Long Short-Term Memory-Networks for Machine Reading". In: *Proceedings of Conference on Empirical Methods in Natural Language Processing, EMNLP.* Ed. by Jian Su, Xavier Carreras, and Kevin Duh, pp. 551–561.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proc. of Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186.

Luong, Thang, Hieu Pham, and Christopher D. Manning (2015). "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421.

Mishra, Nikhil et al. (2018). "A Simple Neural Attentive Meta-Learner". In: *International Conference on Learning Representations.*

- Peters, Matthew E. et al. (2018). "Deep Contextualized Word Representations". In: *Proc. of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2227–2237.
- Radford, Alec et al. (2019). *Language Models are Unsupervised Multitask Learners*. Technical report, OpenAi.
- Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.
- Vinyals, Oriol et al. (2015). "Show and tell: A neural image caption generator". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3156–3164.
- Xu, Kelvin et al. (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37, pp. 2048–2057.
- Yang, Zichao et al. (2016). "Hierarchical Attention Networks for Document Classification". In: *The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489.

Questions?