Natural Language Processing
Sequence to Sequence Models, Attention and the Transformer

Felipe Bravo-Marquez

November 7, 2019

Language Models and Language Generation

- Language modeling is the task of assigning a probability to sentences in a language.
- Example: what is the probability of seeing the sentence “the lazy dog barked loudly”?
- The task can be formulated as the task of predicting the probability of seing a word conditioned on previous words:

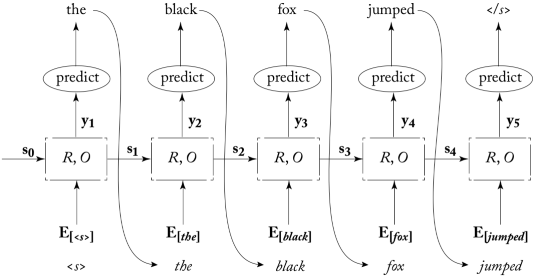
$$P(w_i|w_1, w_2, \cdots, w_{i-1}) = \frac{P(w_1, w_2, \cdots, w_{i-1}, w_i)}{P(w_1, w_2, \cdots, w_{i-1})}$$

Language Models and Language Generation

- RNNs can be used to train language models by tying the output at time i with its input at time $i + 1$.
- This network can be used to generate sequences of words or random sentences.
- Generation process: predict a probability distribution over the first word conditioned on the start symbol, and draw a random word according to the predicted distribution.
- Then predict a probability distribution over the second word conditioned on the first, and so on, until predicting the end-of-sequence $< /\mathrm{s}>$ symbol.

Language Models and Language Generation

- After predicting a distribution over the next output symbols $P(t_i = k | t_{1:i-1})$, a token t_i is chosen and its corresponding embedding vector is fed as the input to the next step.



- Teacher-forcing: during **training** the generator is fed with the ground-truth previous word even if its own prediction put a small probability mass on it.
- It is likely that the generator would have generated a different word at this state in **test time**.

Sequence to Sequence Problems

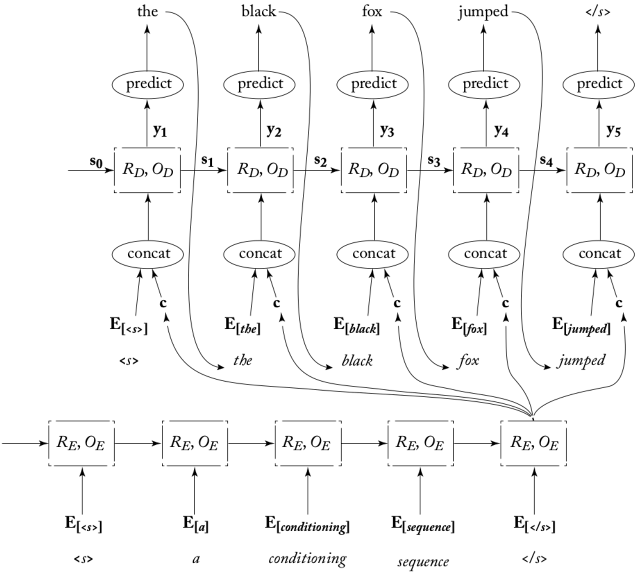
Nearly any task in NLP can be formulated as a sequence to sequence (or condionated generation) task i.e., generate output sequences from input ones. Input and output sequences can have different lengths.

- Machine Translation: source language to target language.
- Summarization: long text to short text.
- Dialogue (chatbots): previous utterances to next utterance.

Conditioned Generation

- While using the RNN as a generator is a cute exercise for demonstrating its strength, the power of RNN gnerator is really revealed when moving to a conditioned generation or enconder-decoder framework.
- Core idea: using two RNNs.
- Encoder: One RNN is used to encode the source input into a vector $\overrightarrow{c}$.
- Decoder: Another RNN is used to decode the encoder's output and generate the target output.
- At each stage of the generation process the context vector $\overrightarrow{c}$ is concatenated to the input $\hat{t}_j$ and the concatenation is fed into the RNN.

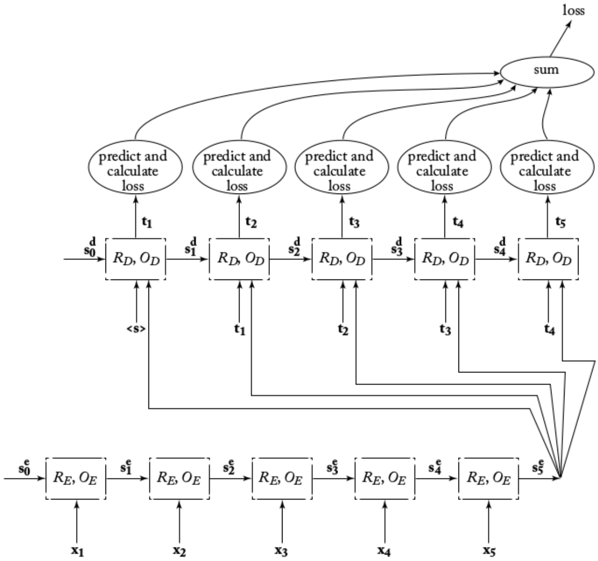
Encoder Decoder Framework



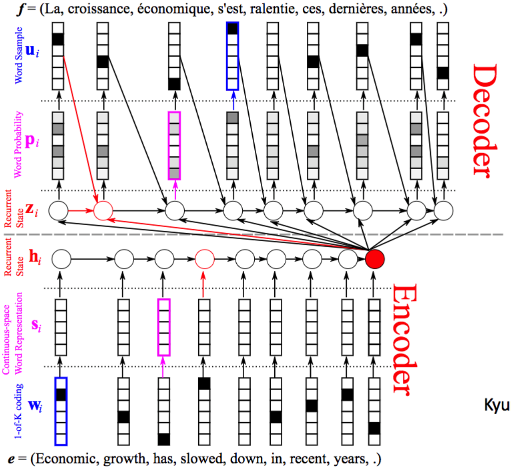
Conditioned Generation

- This setup is useful for mapping sequences of length n to sequences of length m.
- The encoder summarizes the source sentence as a vector $\vec{c}$.
- The decoder RNN is then used to predict (using a language modeling objective) the target sequence words conditioned on the previously predicted words as well as the encoded sentence $\vec{c}$.
- The encoder and decoder RNNs are trained jointly.
- The supervision happens only for the decoder RNN, but the gradients are propagated all the way back to the encoder RNN.

Sequence to Sequence Training Graph



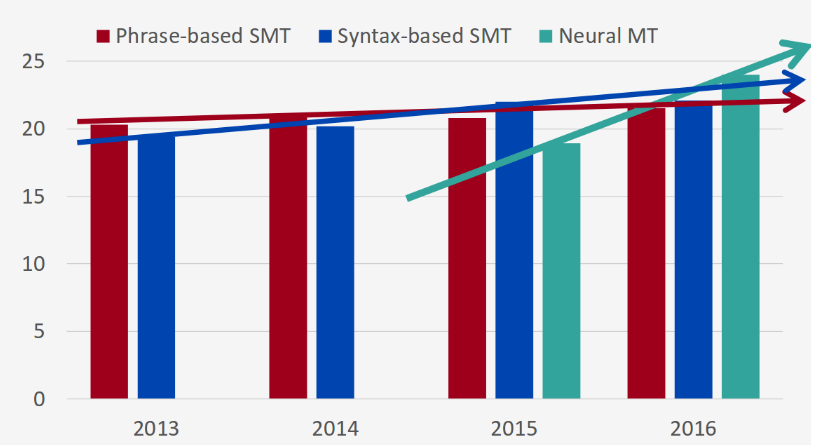
Neural Machine Translation



f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
Recurrent State
$\mathbf{z}_i$
Decoder
Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Kyunghyun Cho et al. 2014

Machine Translation BLEU progress over time



[Edinburgh En-De WMT]

[0] source: `http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf`

Decoding Approaches

- The decoder aims to generate the output sequence with maximal score (or maximal probability), i.e., such that $\sum_{i=1}^{n} P(\hat{t}_i|\hat{t}_{1:i-1})$ is maximized.
- The non-markovian nature of the RNN means that the probability function cannot be decomposed into factors that allow for exact search using standard dynamic programming.
- Exact search: finding the optimum sequence requires evaluating every possible sequence (computationally prohibitive).
- Thus, it only makes sense to solving the optimization problem above approximately.
- Greedy search: choose the highest scoring prediction (word) at each step.
- This may result in sub-optimal overall probability leading to prefixes that are followed by low-probability events.

Greedy Search

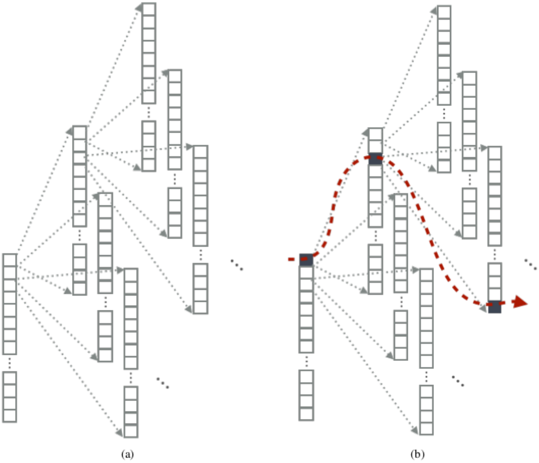


Figure 6.4: (a) Search space depicted as a tree. (b) Greedy search.

[0] Source: [Cho, 2015]

Beam Search

- Beam search interpolates between the exact search and the greedy search by changing the size K of hypotheses maintained throughout the search procedure [Cho, 2015].
- The Beam search algorithm works in stages.
- We first pick the K starting words with the highest probability
- At each step, each candidate sequence is expanded with all possible next steps.
- Each candidate step is scored.
- The K sequences with the most likely probabilities are retained and all other candidates are pruned.
- The search process can halt for each candidate separately either by reaching a maximum length, by reaching an end-of-sequence token, or by reaching a threshold likelihood.
- The sentence with the highest overall probability is selected.

[0] More info at: `https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/`

Conditioned Generation with Attention

- In the encoder-decoder networks the input sentence is encoded into a single vector, which is then used as a conditioning context for an RNN-generator.
- This architectures forces the encoded vector $\vec{c}$ to contain all the information required for generation.
- It doesn't work well for long sentences!
- It also requires the generator to be able to extract this information from the fixed-length vector.
- "You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!" -Raymond Mooney
- This architecture can be can be substantially improved (in many cases it) by the addition of an attention mechanism.
- The attention mechanism attempts to solve this problem by allowing the decoder to "look back" at the encoder's hidden states based on its current state.

Conditioned Generation with Attention

- The input sentence (a length n input sequence $\vec{x}_{1:n}$) is encoded using a biRNN as a sequence of vectors $\vec{c}_{1:n}$.
- The decoder uses a soft attention mechanism in order to decide on which parts of the encoding input it should focus.
- At each stage j the decoder sees a weighted average of the vectors $\vec{c}_{1:n}$, where the attention weights ($\vec{\alpha}^j$) are chosen by the attention mechanism.

$$\vec{c}^j = \sum_{i=1}^{n} \vec{\alpha}^j_{[i]} \cdot \vec{c}_i$$

- The elemements of $\vec{\alpha}^j$ are all positive and sum to one.

Conditioned Generation with Attention

- Unnormalized attention weights ($\bar{\alpha}_{[i]}^{j}$) are produced taking into account the decoder state at time j ($\vec{s}_j$) and each of the vectors $\vec{c}_i$.
- They can be obtained in various ways, basically any differentiable function returning a scalar out of two vectors $\vec{s}_j$ and $\vec{c}_i$ could be employed.
- The simplest approach is a dot product: $\bar{\alpha}_{[i]}^{j} = \vec{s}_j \cdot \vec{c}_i$.
- Another one is Additive attention which uses a Multilayer Perceptron: $\bar{\alpha}_{[i]}^{j} = MLP^{att}([\vec{s}_j; \vec{c}_i]) = \vec{v} \cdot \mathtt{tanh}([\vec{s}_j; \vec{c}_i]U + \vec{b})$

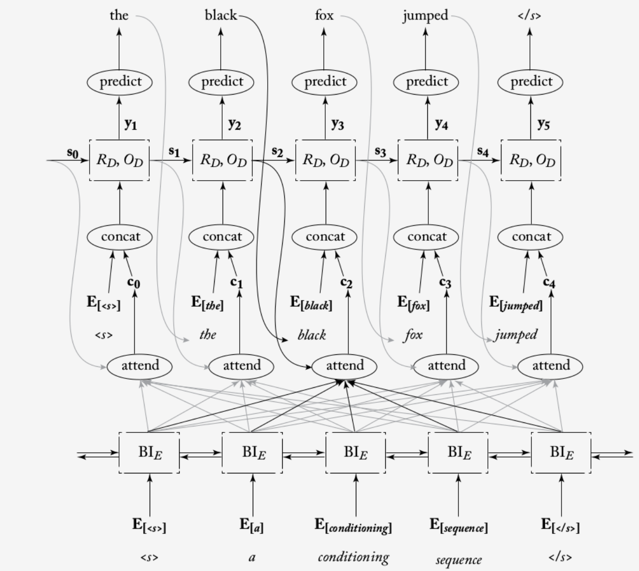
Conditioned Generation with Attention

- These unnormalized weights are then normalized into a probability distribution using the softmax function.

$$
\begin{aligned}
\mathrm{attend}(\boldsymbol{c}_{\mathbf{1}:\boldsymbol{n}}, \hat{t}_{1:j}) &= \boldsymbol{c}^{\boldsymbol{j}} \\
\boldsymbol{c}^{\boldsymbol{j}} &= \sum_{i=1}^{n} \boldsymbol{\alpha}^{\boldsymbol{j}}_{[\boldsymbol{i}]} \cdot \boldsymbol{c}_{\boldsymbol{i}} \\
\boldsymbol{\alpha}^{\boldsymbol{j}} &= \mathrm{softmax}(\bar{\boldsymbol{\alpha}}^{\boldsymbol{j}}_{[\mathbf{1}]}, \ldots, \bar{\boldsymbol{\alpha}}^{\boldsymbol{j}}_{[\boldsymbol{n}]}) \\
\bar{\boldsymbol{\alpha}}^{\boldsymbol{j}}_{[\boldsymbol{i}]} &= \mathrm{MLP}^{\mathrm{att}}([\boldsymbol{s}_{\boldsymbol{j}}; \boldsymbol{c}_{\boldsymbol{i}}]),
\end{aligned}
$$

- The encoder, decoder, and attention mechanism are all trained jointly in order to play well with each other.

Attention


the
black
fox
jumped
</s>
predict
predict
predict
predict
predict
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\mathbf{y}_4$
$\mathbf{y}_5$
$\mathbf{s}_0$
R_D, O_D
$\mathbf{s}_1$
R_D, O_D
$\mathbf{s}_2$
R_D, O_D
$\mathbf{s}_3$
R_D, O_D
$\mathbf{s}_4$
R_D, O_D
concat
concat
concat
concat
concat
$\mathbf{E}_{[<s>]}$
$\mathbf{c}_0$
$\mathbf{E}_{[the]}$
$\mathbf{c}_1$
$\mathbf{E}_{[black]}$
$\mathbf{c}_2$
$\mathbf{E}_{[fox]}$
$\mathbf{c}_3$
$\mathbf{E}_{[jumped]}$
$\mathbf{c}_4$
<s>
the
black
fox
jumped
attend
attend
attend
attend
attend
BI_E
BI_E
BI_E
BI_E
BI_E
$\mathbf{E}_{[<s>]}$
$\mathbf{E}_{[a]}$
$\mathbf{E}_{[conditioning]}$
$\mathbf{E}_{[sequence]}$
$\mathbf{E}_{[</s>]}$
<s>
a
conditioning
sequence
</s>

Attention and Word Alignments

- In the context of machine translation, one can think of MLP att as computing a soft alignment between the current decoder state $\vec{s}_j$ (capturing the recently produced foreign words) and each of the source sentence components $\vec{c}_i$.

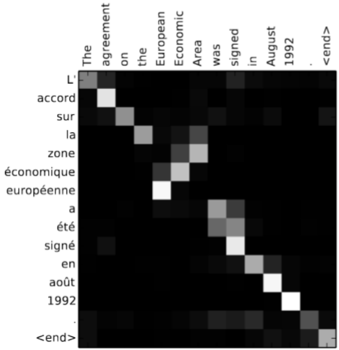
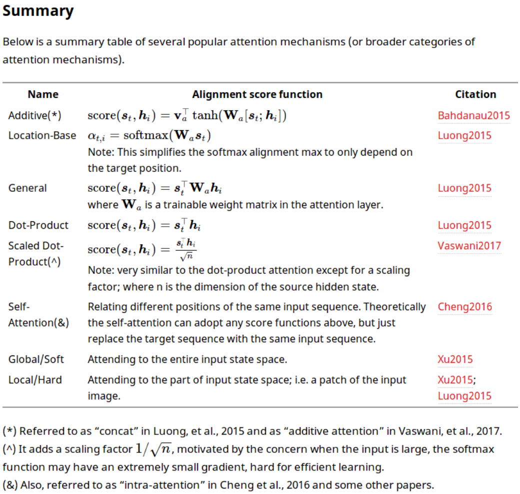


Fig. 2. Visualization of the attention weights α_j^t of the attention-based neural machine translation model [32]. Each row corresponds to the output symbol, and each column the input symbol. Brighter the higher α_j^t.

Figure: Source: [Cho et al., 2015]

Other types of Attention



Summary

Below is a summary table of several popular attention mechanisms (or broader categories of attention mechanisms).

Name	Alignment score function	Citation
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment max to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.
(&) Also, referred to as "intra-attention" in Cheng et al., 2016 and some other papers.

Figure: Source: `https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`

What’s Wrong with RNNs?

- Given what we just learned above, it would seem like attention solves all the problems with RNNs and encoder-decoder architectures. [1]
- There are a few shortcomings of RNNs that another architecture called the **Transformer** tries to address.
- The Transformer discards the recursive component of the Encoder-Decoder architecture and purely relies on attention mechanisms [Vaswani et al., 2017].
- When we process a sequence using RNNs, each hidden state depends on the previous hidden state.
- This becomes a major pain point on GPUs: GPUs have a lot of computational capability and they hate having to wait for data to become available.
- Even with technologies like CuDNN, RNNs are painfully inefficient and slow on the GPU.

[1] The following material is based on: `http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/`

Dependencies in neural machine translations

In essence, there are three kinds of dependencies in neural machine translations:

1. Dependencies between the input and output tokens.
2. Dependencies between the input tokens themselves.
3. Dependencies between the output tokens themselves.

The traditional attention mechanism largely solved the first dependency by giving the decoder access to the entire input sequence.

The Transformer

- The novel idea of the Transformer is to extend this mechanism to the processing input and output sentences as well.
- The RNN processes input sequences sequentialy.
- The Transformer, on the other hand, allows the encoder and decoder to see the entire input sequence all at once.
- This is done using attention.

The Key Component: Multi-Head Attention

- The attention mechanism in the Transformer is interpreted as a way of computing the relevance of a set of **values** (information) based on some **keys** and **queries**.
- The attention mechanism is used as a way for the model to **focus on relevant information** based on what it is currently processing.
- In the RNN encoder-decoder architecture with attention:
    1. Attention weights were the relevance of the encoder hidden states (values) in processing the decoder state (query).
    2. These values were calculated based on the encoder hidden states (keys) and the decoder hidden state (query).

The Key Component: Multi-Head Attention

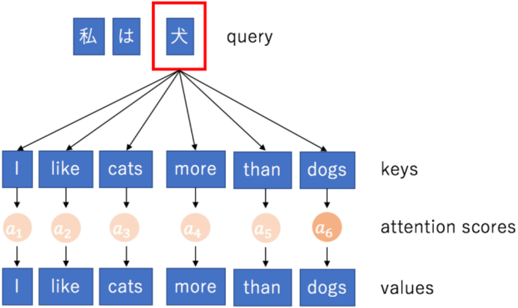


Figure: In this example, the query is the word being decoded (which means dog) and both the keys and values are the source sentence. The attention score represents the relevance, and in this case is large for the word “dog” and small for others.

The Key Component: Multi-Head Attention

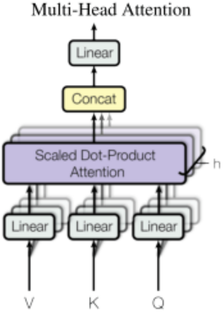
- When we think of attention this way, we can see that the keys, values, and queries could be anything.
- They could even be the same.
- For instance, both values and queries could be input embeddings (self attention).
- If we only computed a single attention weighted sum of the values, it would be difficult to capture various different aspects of the input.

The Key Component: Multi-Head Attention

- For instance, in the sentence “I like cats more than dogs”, you might want to capture the fact that the sentence compares two entities, while also retaining the actual entities being compared.
- To solve this problem the Transformer uses the Multi-Head Attention block.
- This block computes multiple attention weighted sums instead of a single attention pass over the values.
- Hence the name “Multi-Head” Attention.

The Key Component: Multi-Head Attention

- To learn diverse representations, the Multi-Head Attention applies different linear transformations to the values, keys, and queries for each “head” of attention.



The Key Component: Multi-Head Attention

- A single attention applies a unique linear transformation to its input queries, keys, and values.
- Then computes the attention score between each query and key.
- The attention score is used to weight the values and sum them up.
- The Multi-Head Attention block just applies multiple blocks in parallel, concatenates their outputs, then applies one single linear transformation.

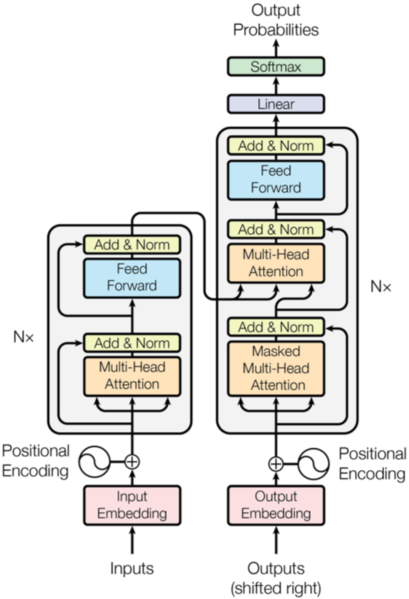
Scaled Dot Product Attention

- Transformer uses a particular form of attention called the “Scaled Dot-Product Attention”:

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

- where Q is the matrix of queries packed together and K and V are the matrices of keys and values packed together.
- d_k represents the dimensionality of the queries and keys.
- The normalization over $\sqrt{d_k}$ is used to rescale the dot products between queries and keys (dot products tend to grow with the dimensionality).

The Transformer



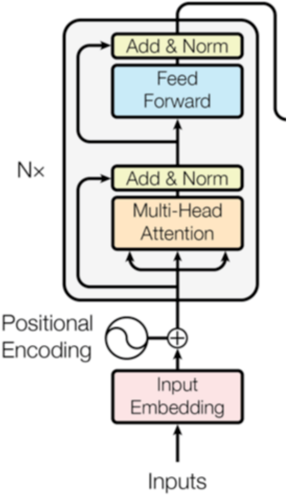
The Transformer

- The Transformer still uses the basic encoder-decoder design of RNN neural machine translation systems.
- The left-hand side is the encoder, and the right-hand side is the decoder.
- The initial inputs to the encoder are the embeddings of the input sequence.
- The initial inputs to the decoder are the embeddings of the outputs up to that point.
- The encoder and decoder are composed of N blocks (where $N = 6$ for both networks).
- These blocks are composed of smaller blocks as well.
- Let's look at each block in further detail.

The Encoder

- The encoder is composed of two blocks (which we will call sub-layers to distinguish from the *N* blocks composing the encoder and decoder).
- One is the Multi-Head Attention sub-layer over the inputs, mentioned above.
- The other is a simple feed-forward network.
- Between each sub-layer, there is a residual connection followed by a layer normalization.
- A residual connection is basically just taking the input and adding it to the output of the sub-network, and is a way of making training deep networks easier.
- Layer normalization is a normalization method in deep learning that is similar to batch normalization.

The Encoder


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

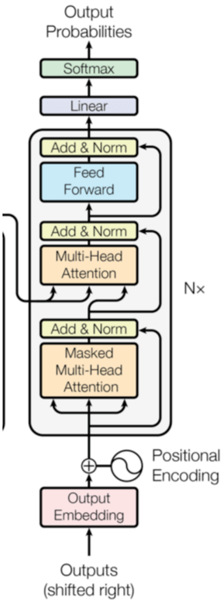
The Encoder

- What each encoder block is doing is actually just a bunch of matrix multiplications followed by a couple of element-wise transformations.
- This is why the Transformer is so fast: everything is just parallelizable matrix multiplications.
- The point is that by stacking these transformations on top of each other, we can create a very powerful network.
- The core of this is the attention mechanism which modifies and attends over a wide range of information.

The Decoder

- The decoder is very similar to the encoder but has one Multi-Head Attention layer labeled the “masked multi-head attention” network.
- This network attends over the previous decoder states, so plays a similar role to the decoder hidden state in RNN machine translation architectures.
- The reason this is called the masked multi-head attention block is that we need to mask the inputs to the decoder from future time-steps.
- When we train the Transformer, we want to process all the sentences at the same time.
- When we do this, if we give the decoder access to the entire target sentence, the model can just repeat the target sentence (in other words, it doesn’t need to learn anything).
- To prevent this from happening, the decoder masks the “future” tokens when decoding a certain word.
- The masking is done by setting to $-\infty$ all values in the inputof the softmax which correspond to illegal connections

The Decoder


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

Positional Encodings

- Unlike recurrent networks, the multi-head attention network cannot naturally make use of the position of the words in the input sequence.
- Without positional encodings, the output of the multi-head attention network would be the same for the sentences “I like cats more than dogs” and “I like dogs more than cats”.
- Positional encodings explicitly encode the relative/absolute positions of the inputs as vectors and are then added to the input embeddings.

Positional Encodings

- The paper uses the following equation to compute the positional encodings:
  $PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}})$
  $PE(pos, 2i+1) = \cos(pos/10000^{2i/d_{model}})$
- Where *pos* represents the position, and *i* is the dimension.
- Basically, each dimension of the positional encoding is a wave with a different frequency.

Conclusions

- The Transformer achieves better BLUE scores than previous state-of-the-art models for English-to-German translation and English-to-French translation at a fraction of the training cost.

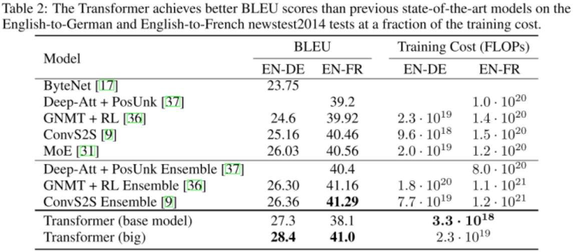


Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

- The Transformer is a powerful and efficient way to replace recurrent networks as a method of modeling dependencies based completey on attention.

Questions?

Thanks for your Attention!

References I

Cho, K. (2015).
Natural language understanding with distributed representation.
arXiv preprint arXiv:1511.07916.

Cho, K., Courville, A., and Bengio, Y. (2015).
Describing multimedia content using attention-based encoder-decoder networks.
IEEE Transactions on Multimedia, 17(11):1875–1886.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is all you need.
In *Advances in neural information processing systems*, pages 5998–6008.