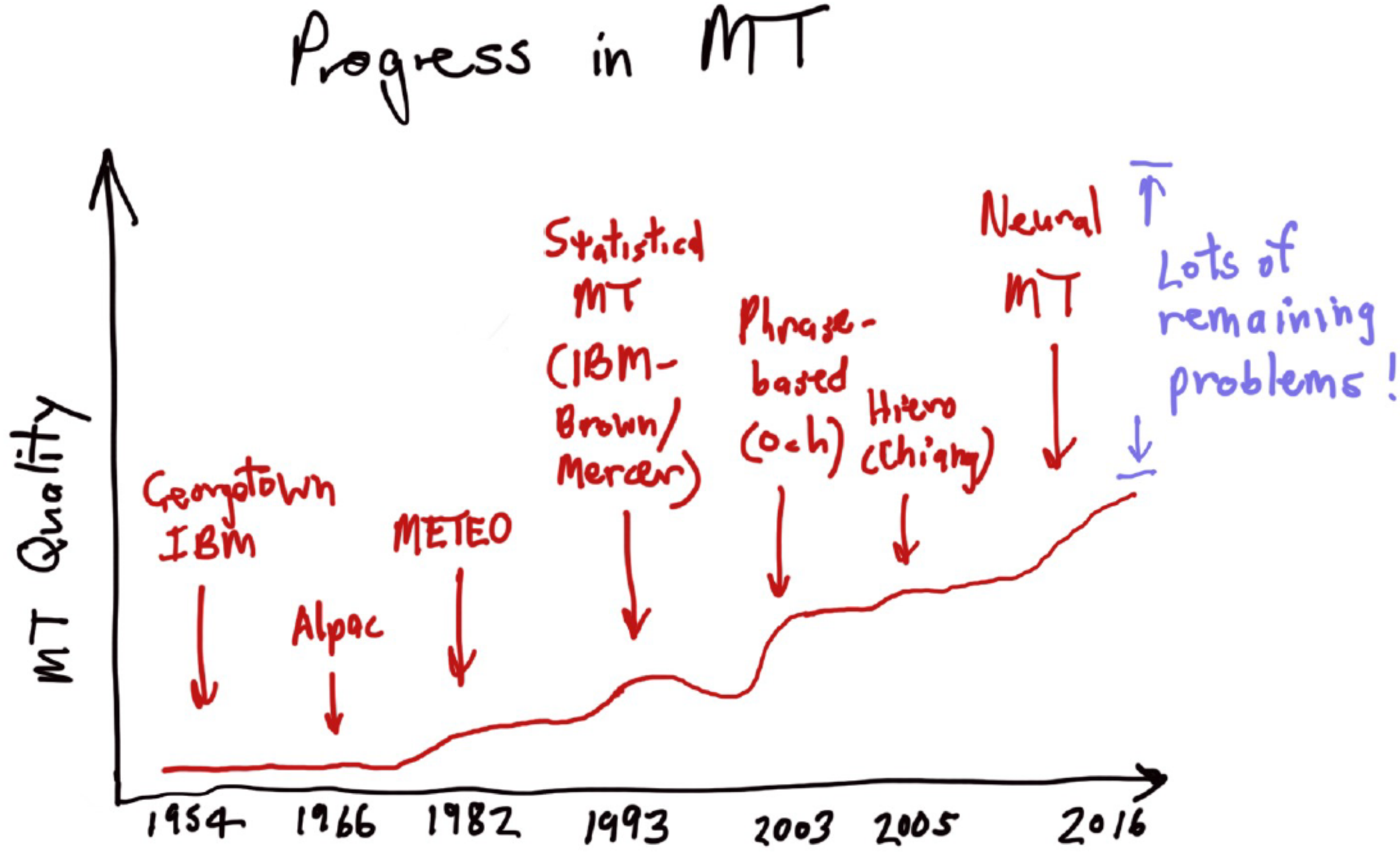
# ”Attention is all you need”

## Vaswani et al. 2017

ML Paper Club @Google Campus with nPlan

6th June 2019

*François Steiner - francois@b3metrix.com*




*in: NMT Tutorial ACL 2016 – Luong et al.*

# When NMT Could Help…


*in: "Sign of the times - Beijing puts an end to Chinglish" – J. Fullerton, The Australian 5 June 2019*

# NN architectures

<table>
<tr><th></th><th>Feed Forward</th><th>Convolutional</th><th>Recurrent</th><th>Self-Attention</th></tr>
<tr><td>Main unit</td><td>Node</td><td colspan="4">Cell</td></tr>
<tr><td>Input</td><td>Scalar</td><td colspan="4">Sequence</td></tr>
<tr><td>Tied weights</td><td>No</td><td colspan="2">Yes</td><td>Sort of…</td></tr>
<tr><td>Process</td><td>-</td><td>Parallel</td><td>Sequential</td><td>Parallel</td></tr>
<tr><td>Properties</td><td>-</td><td>Translation-<br>invariant</td><td>Variable length<br>Position-aware</td><td>Captures LT<br>dependencies</td></tr>
</table>

# Comparison of complexity, path lengths and number of sequential operations

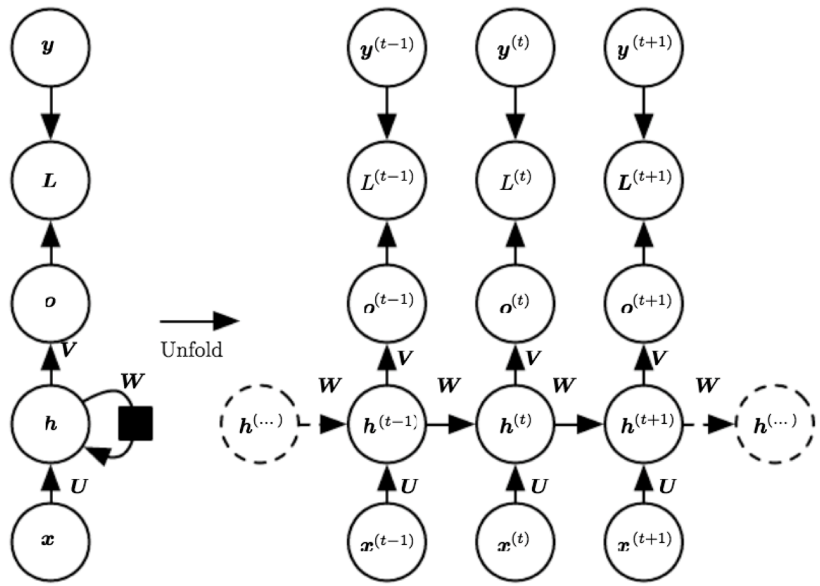
Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. $n$ is the sequence length, $d$ is the representation dimension, $k$ is the kernel size of convolutions and $r$ the size of the neighborhood in restricted self-attention.

| Layer Type | Complexity per Layer | Sequential Operations | Maximum Path Length |
|---|---|---|---|
| Self-Attention | $O(n^2 \cdot d)$ | $O(1)$ | $O(1)$ |
| Recurrent | $O(n \cdot d^2)$ | $O(n)$ | $O(n)$ |
| Convolutional | $O(k \cdot n \cdot d^2)$ | $O(1)$ | $O(log_k(n))$ |
| Self-Attention (restricted) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

# RECURRENT NETWORKS

# Recurrent networks

- Each ***cell*** processes a ***sequence***
- Suited to sequences of ***variable length***
- Use of ”internal” or “cell” ***state***
- On this example, input and output sequences of identical length

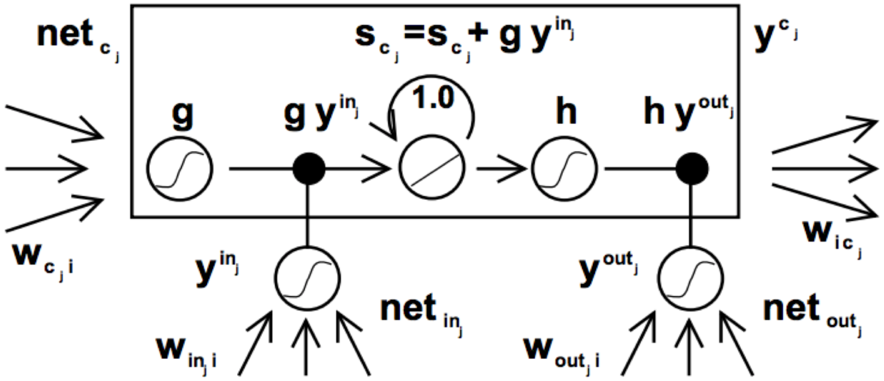



*in: Deep Learning – Goodfellow et al. MIT Press 2016*

# LSTM

A specific case of RNNs

- Several ***gates*** control the flow of information between (time) steps
- Objective: address ***long-term dependencies***

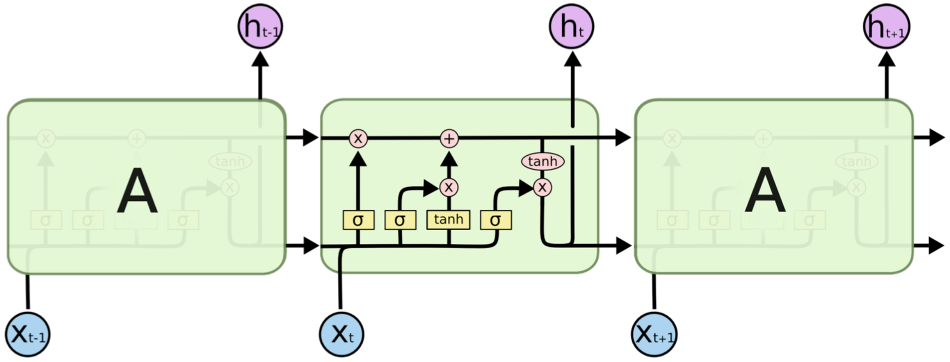



*in: Long Short-Term Memory – Hochreiter et al. 1997*

# LSTM

A specific case of RNNs

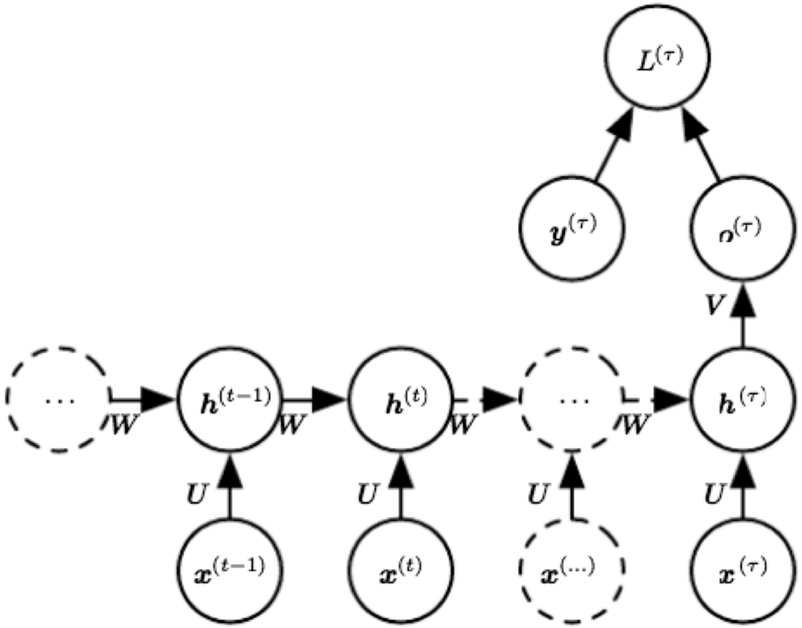
- Several ***gates*** control the flow of information between (time) steps
- Objective: address ***long-term dependencies***




*in: colah's blog – Christopher Olah (colah.github.io)*

# RNNs architecture variants

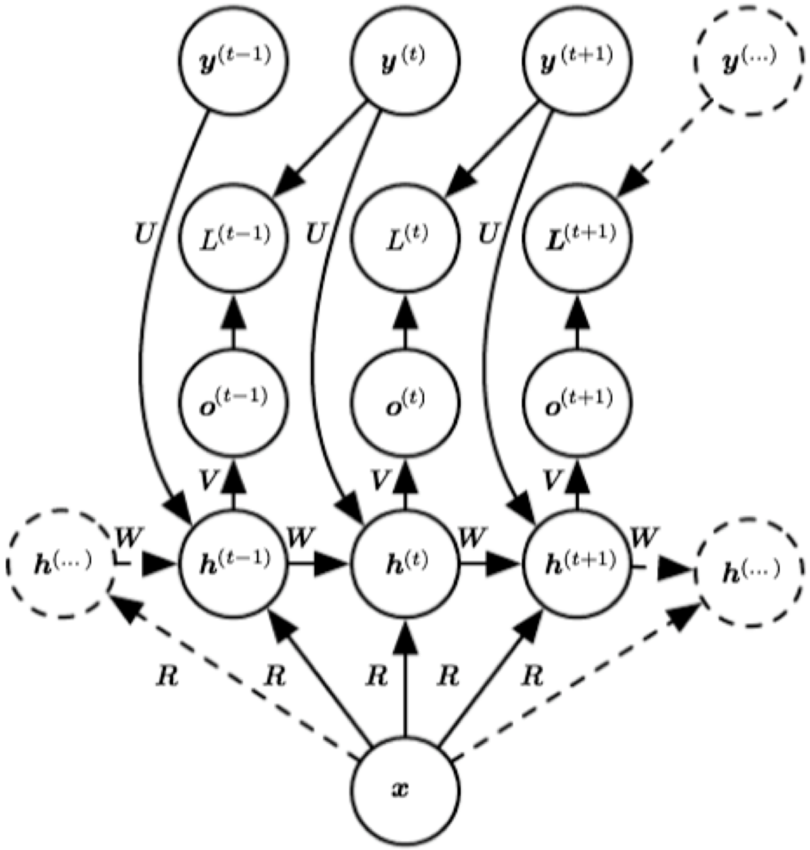
- RNN with ***single output***
- Provides a ***fixed-size representation*** of a sequence




*in: Deep Learning – Goodfellow et al. MIT Press 2016*

# RNNs architecture variants

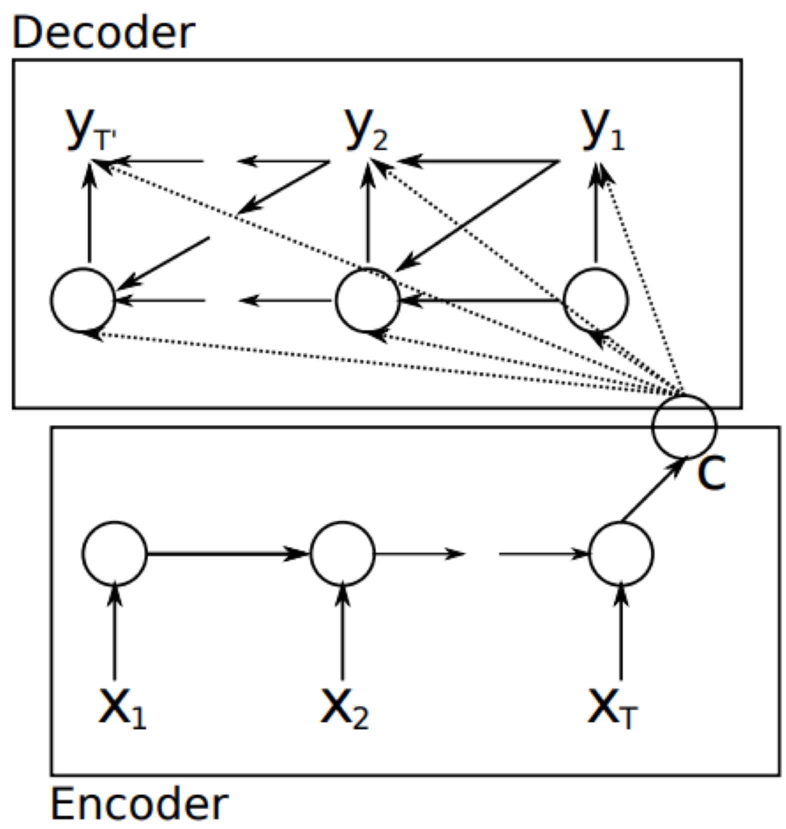
- RNN that maps a ***fixed-length vector into a sequence***
- Example of use: image captioning




*in: Deep Learning – Goodfellow et al. MIT Press 2016*

# RNN Encoder-Decoder

Combination of 2 RNNs

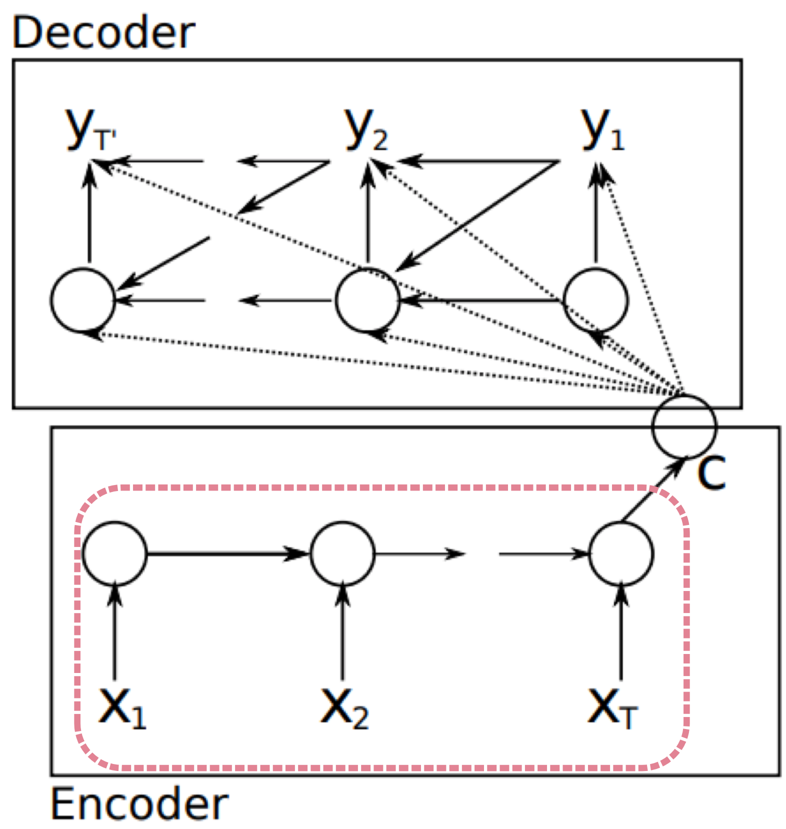



*in: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation – Cho et al. 2014*

# RNN Encoder-Decoder

Combination of 2 RNNs

- ***Encoding*** a sequence into a fixed-length representation

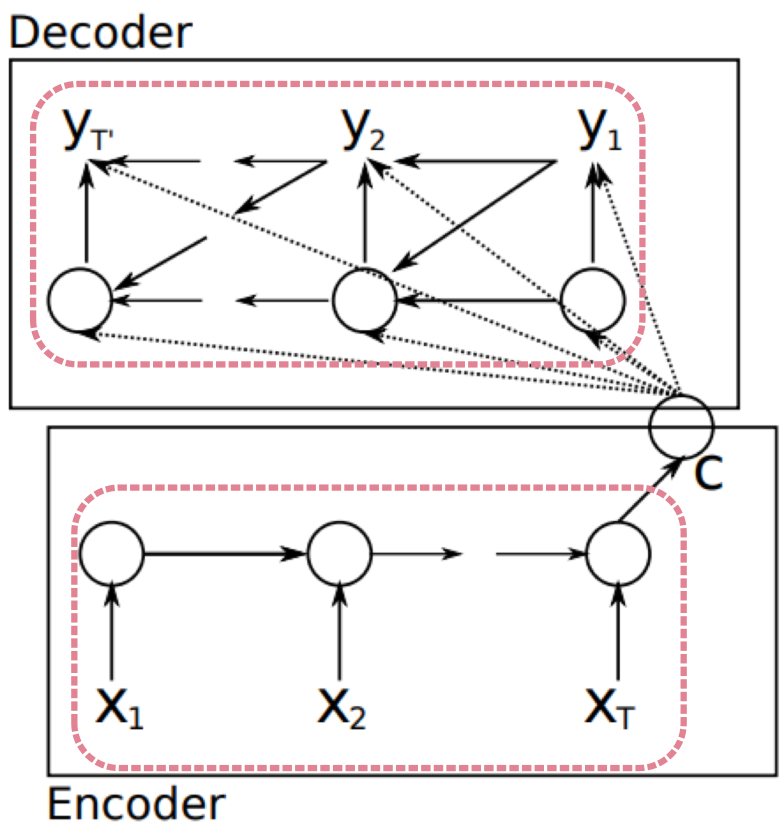



*in: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation – Cho et al. 2014*

# RNN Encoder-Decoder

Combination of 2 RNNs

- *Encoding* a sequence into a fixed-length representation
- *Decoding* a single context vector into a variable length sequence

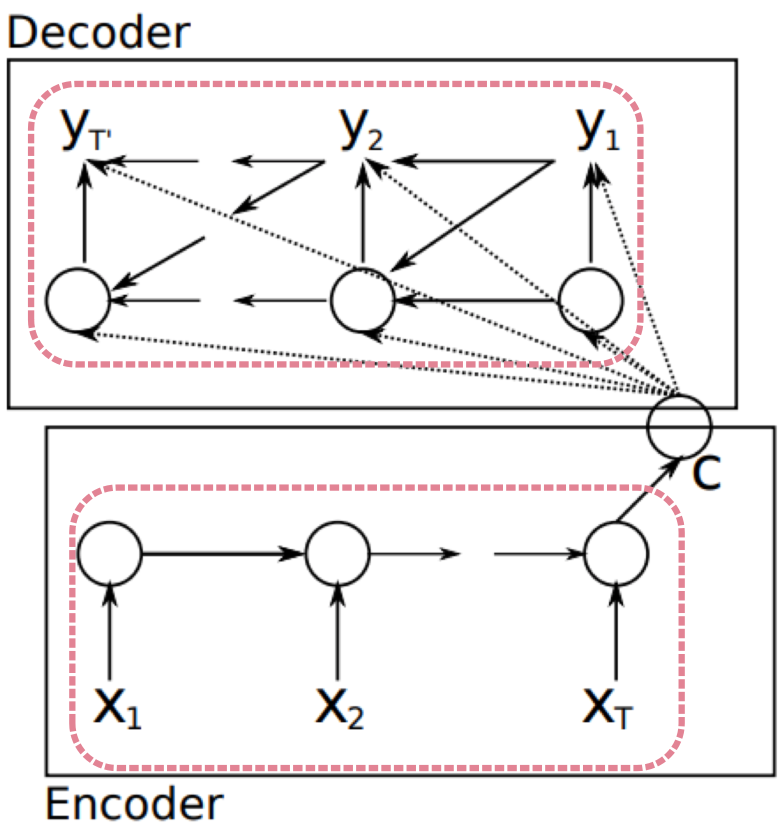



*in: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation – Cho et al. 2014*

# RNN Encoder-Decoder

Combination of 2 RNNs

- *Encoding* a sequence into a fixed-length representation
- *Decoding* a single context vector into a variable length sequence
- Input and output sequences of *different lengths*: suitable for machine translation

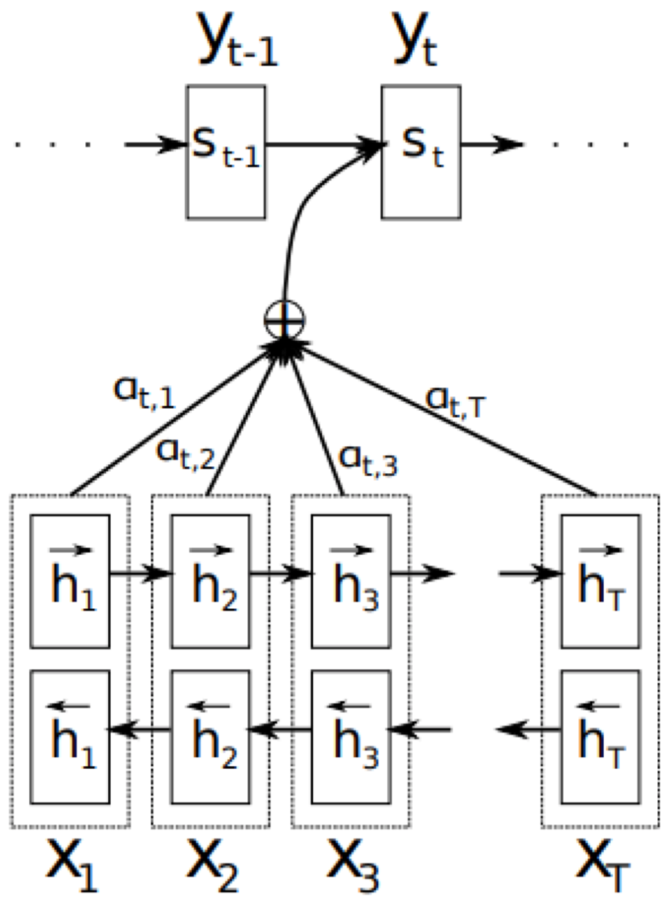



*in: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation – Cho et al. 2014*

# ATTENTION MECHANISM

# Attention mechanism

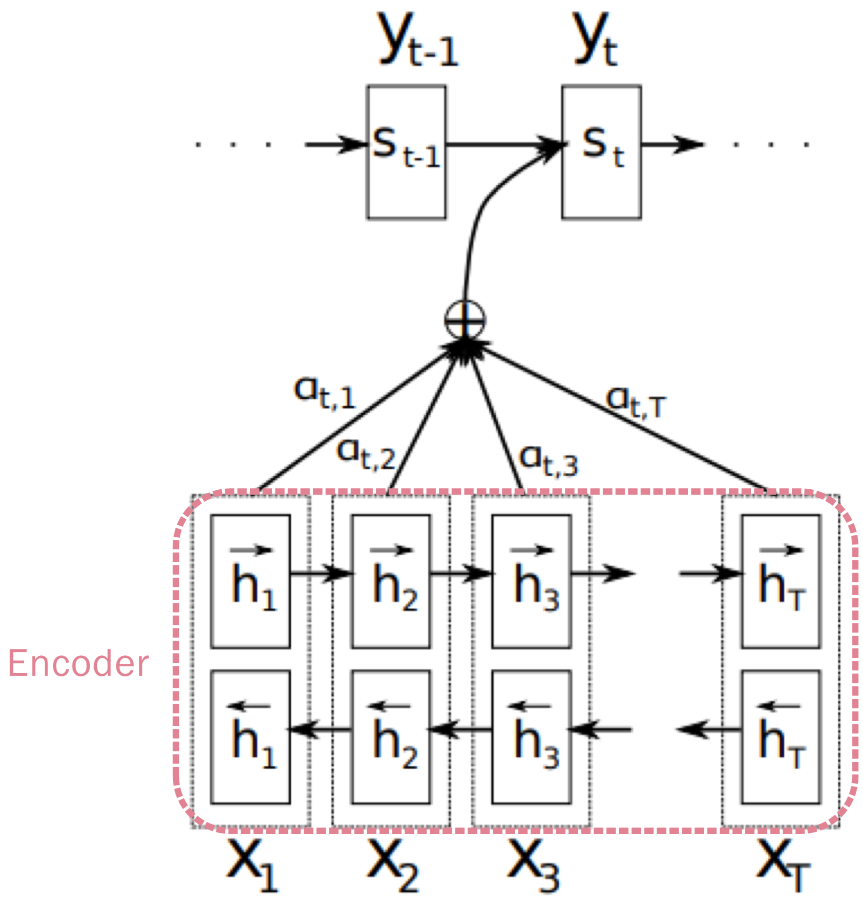
- Addresses bottleneck of ***fixed-length intermediate representation***




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention mechanism

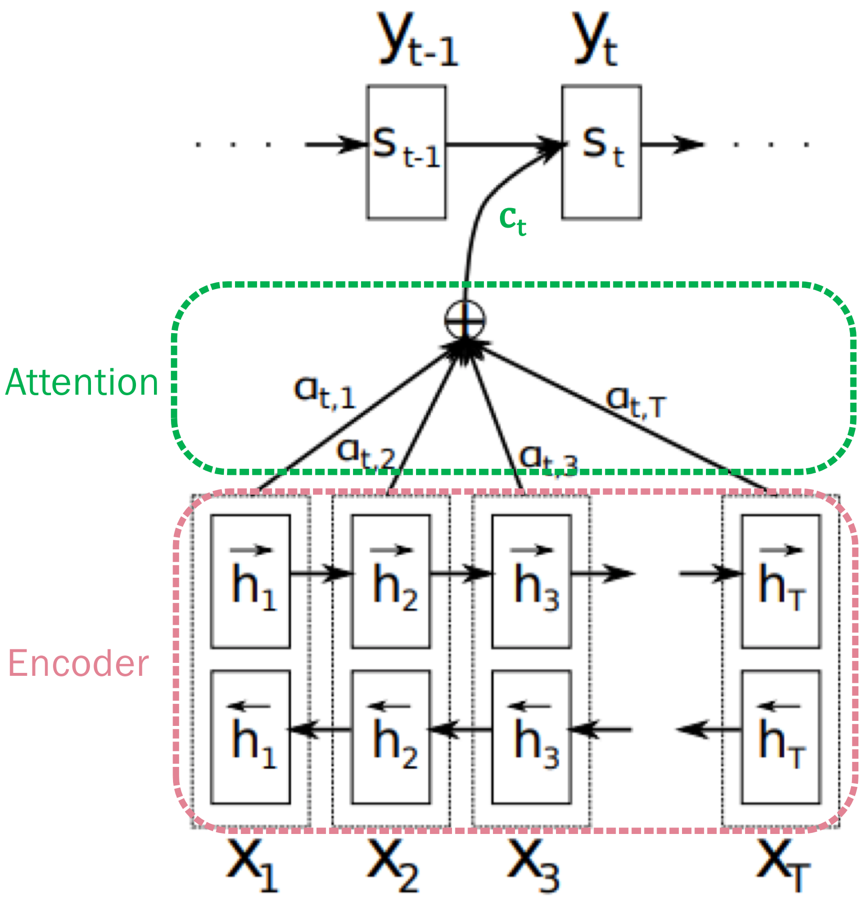
- Addresses bottleneck of ***fixed-length intermediate representation***
- Creates ***annotations*** $h_t$ for each input $X_t$ using a bi-directional LSTM (Encoder)




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention mechanism

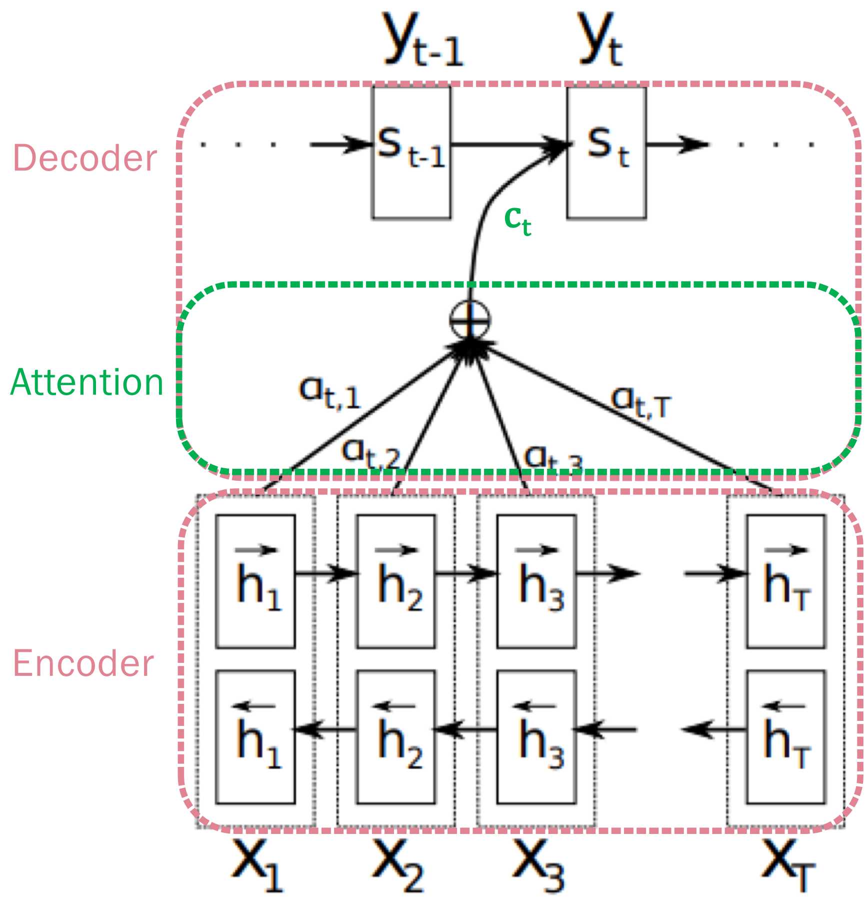
- Addresses bottleneck of ***fixed-length intermediate representation***
- Creates ***annotations*** $h_t$ for each input $X_t$ using a bi-directional LSTM (Encoder)
- Iteratively produces ***context vectors*** applying a set of ***weights*** $\alpha_{t't}$ to the annotations




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention mechanism

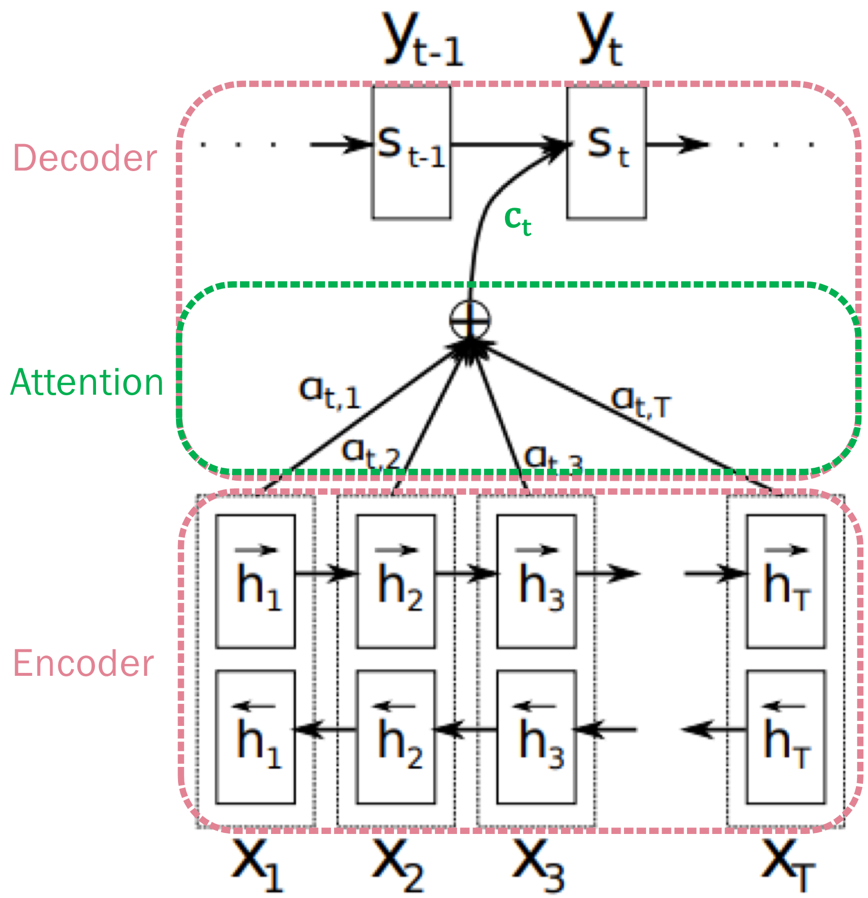
- Addresses bottleneck of ***fixed-length intermediate representation***
- Creates ***annotations*** $h_t$ for each input $X_t$ using a bi-directional LSTM (Encoder)
- Iteratively produces ***context vectors*** applying a set of ***weights*** $\alpha_{t't}$ to the annotations
- Context vectors used as an input by a second LSTM (Decoder)




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

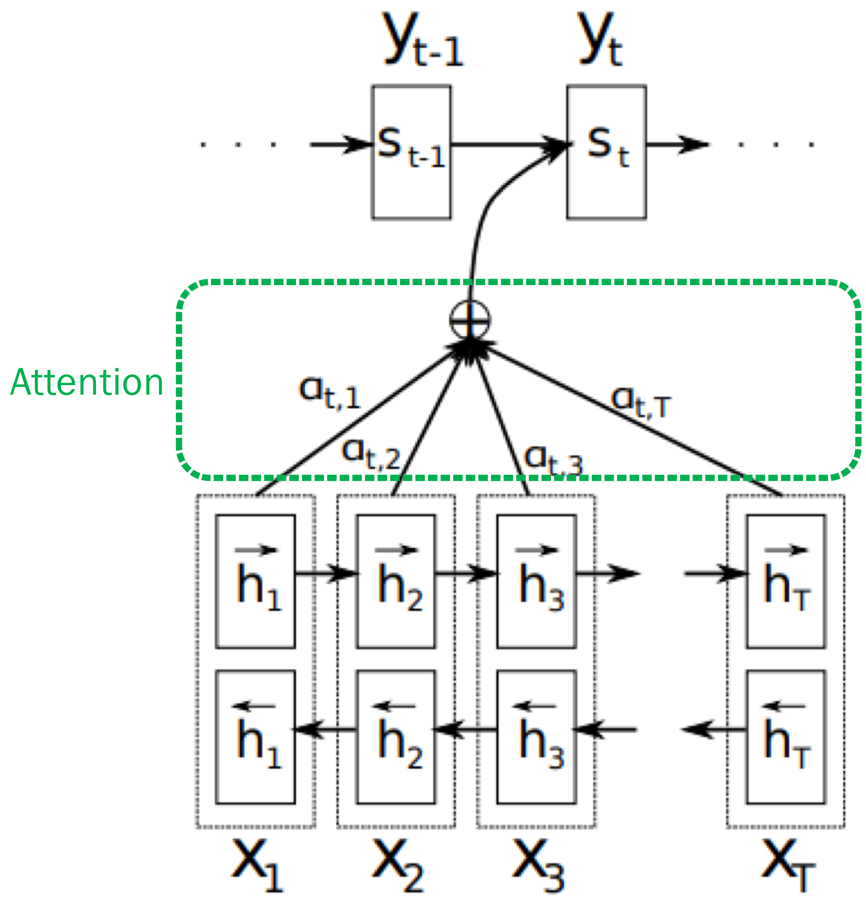
# Attention mechanism

- Addresses bottleneck of ***fixed-length intermediate representation***
- Creates ***annotations*** $h_t$ for each input $X_t$ using a bi-directional LSTM (Encoder)
- Iteratively produces ***context vectors*** applying a set of ***weights*** $\alpha_{t't}$ to the annotations
- Context vectors used as an input by a second LSTM (Decoder)
- Weights produced by an ***attention model*** (feed-forward network) – each context vector is different




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

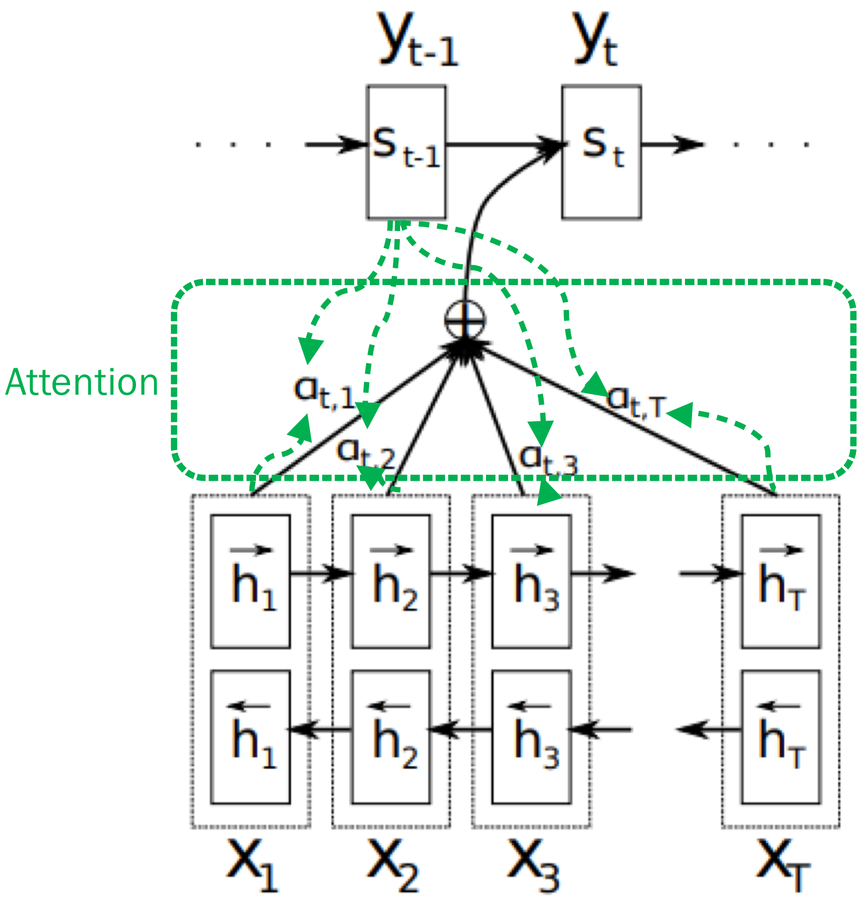
# Attention model




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention model

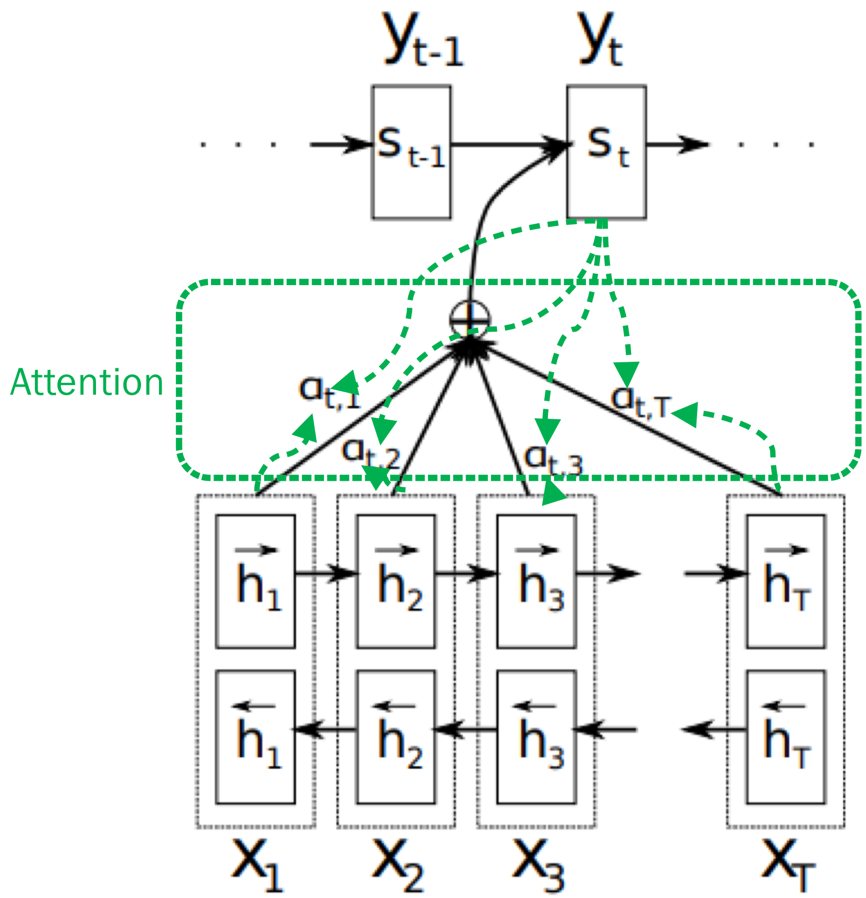
$$e_{ij} = a(s_{i-1}, h_j)$$




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention model

$$e_{ij} = a(s_{i-1}, h_j)$$

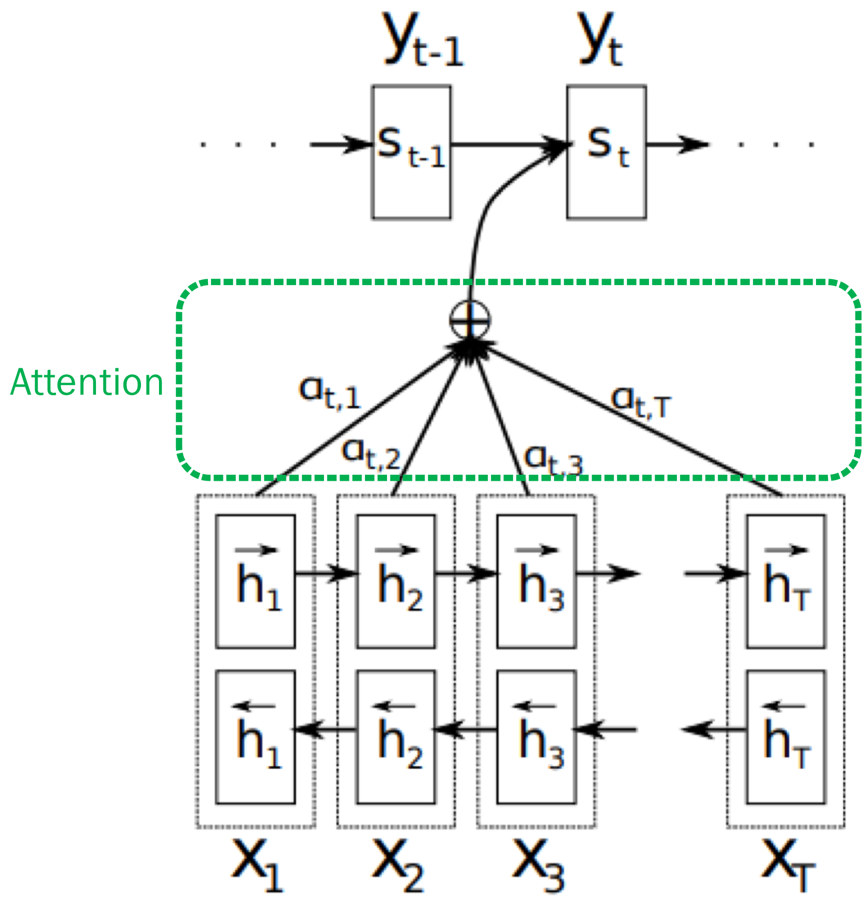



*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention model

$$e_{ij} = a(s_{i-1}, h_j)$$

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



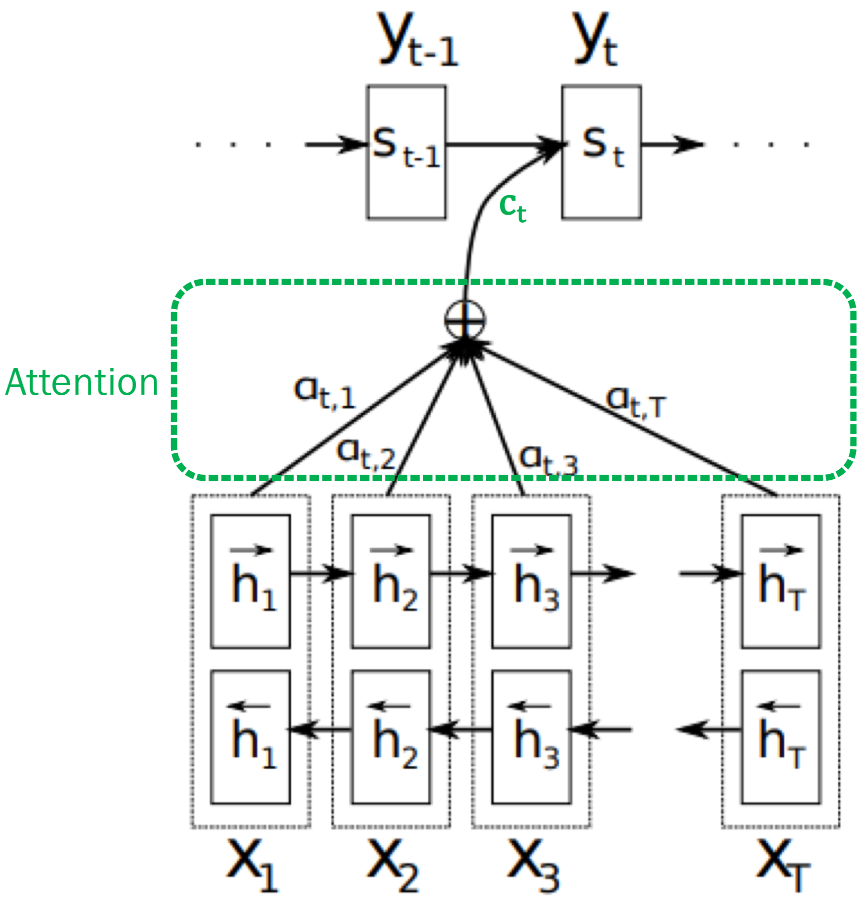

*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Attention model

$$e_{ij} = a(s_{i-1}, h_j)$$

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

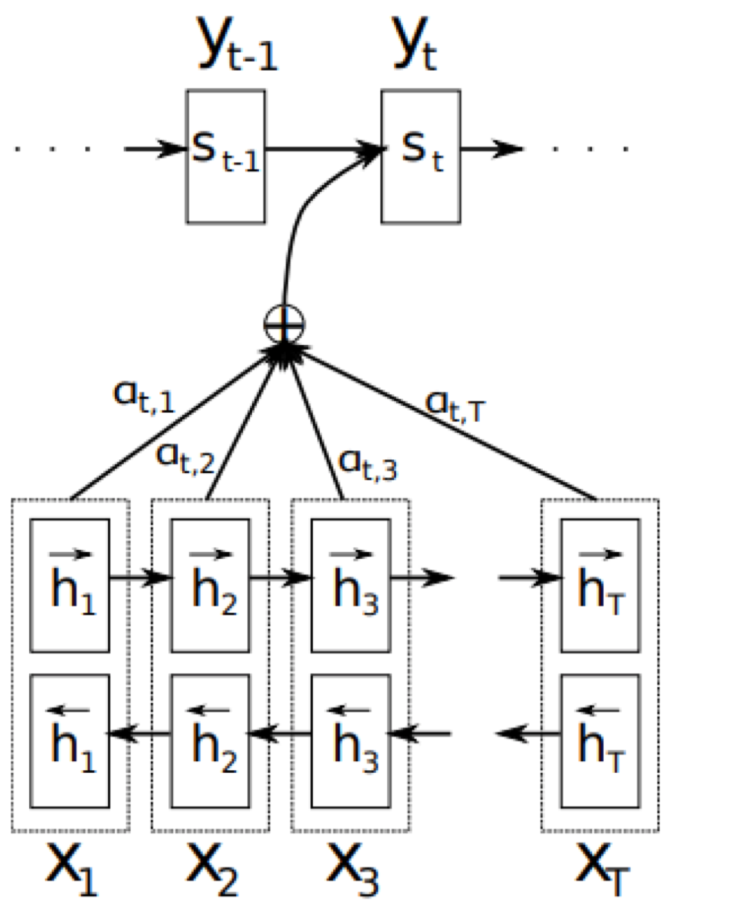
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$




*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# Formalisation

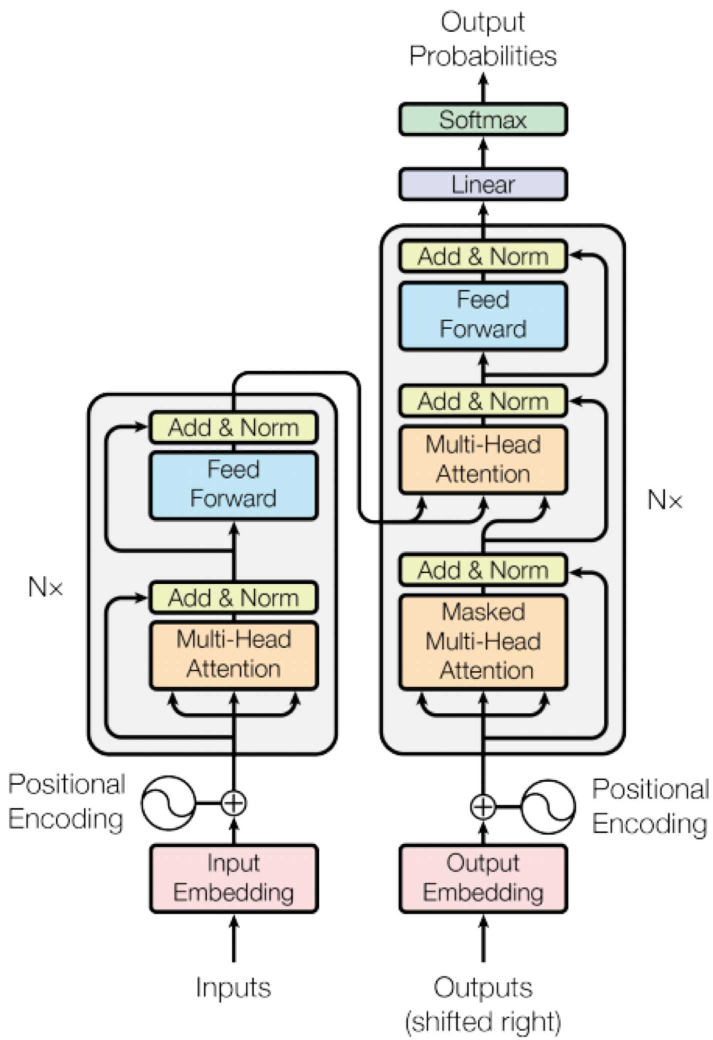
- $s_{t-1}$: queries, matrix Q
- $h_t$ (as attention parameters): keys, matrix K
- $h_t$ (as values): values, matrix V
- Attention = a(Q, K).V



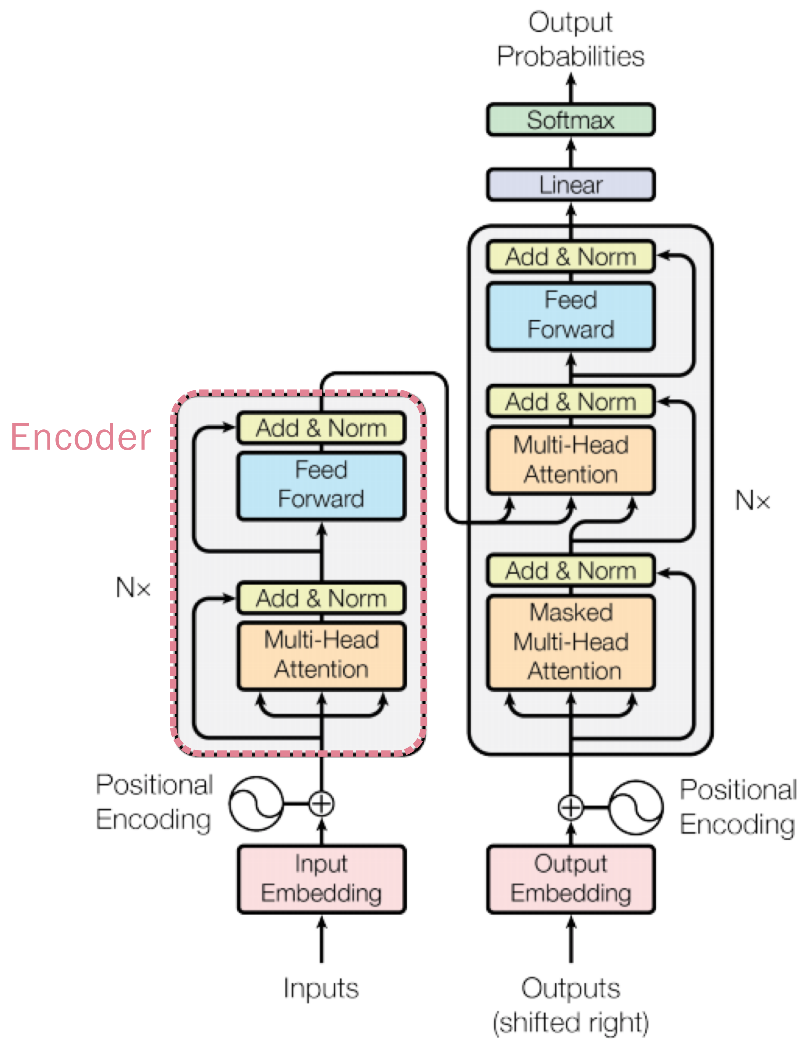

*in: Neural Machine Translation by Jointly Learning to Align and Translate - Bahdanau et al. 2015*

# TRANSFORMER

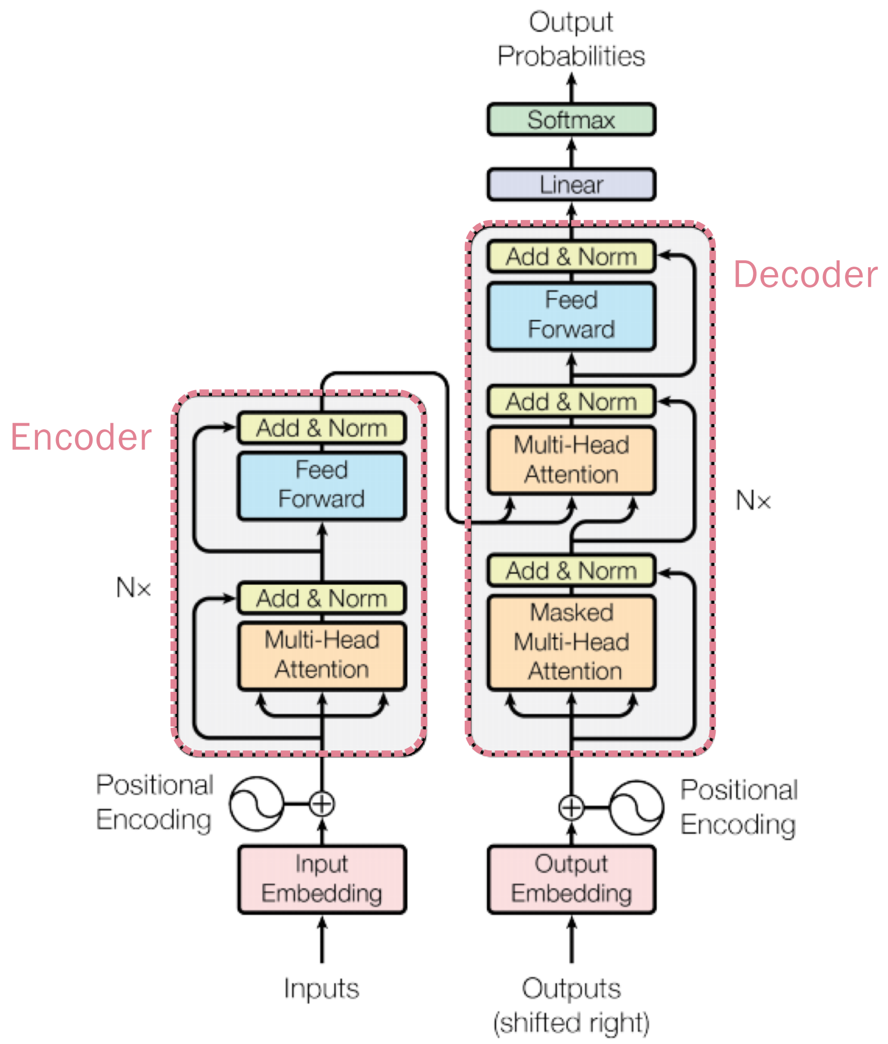
# Attention-only architecture



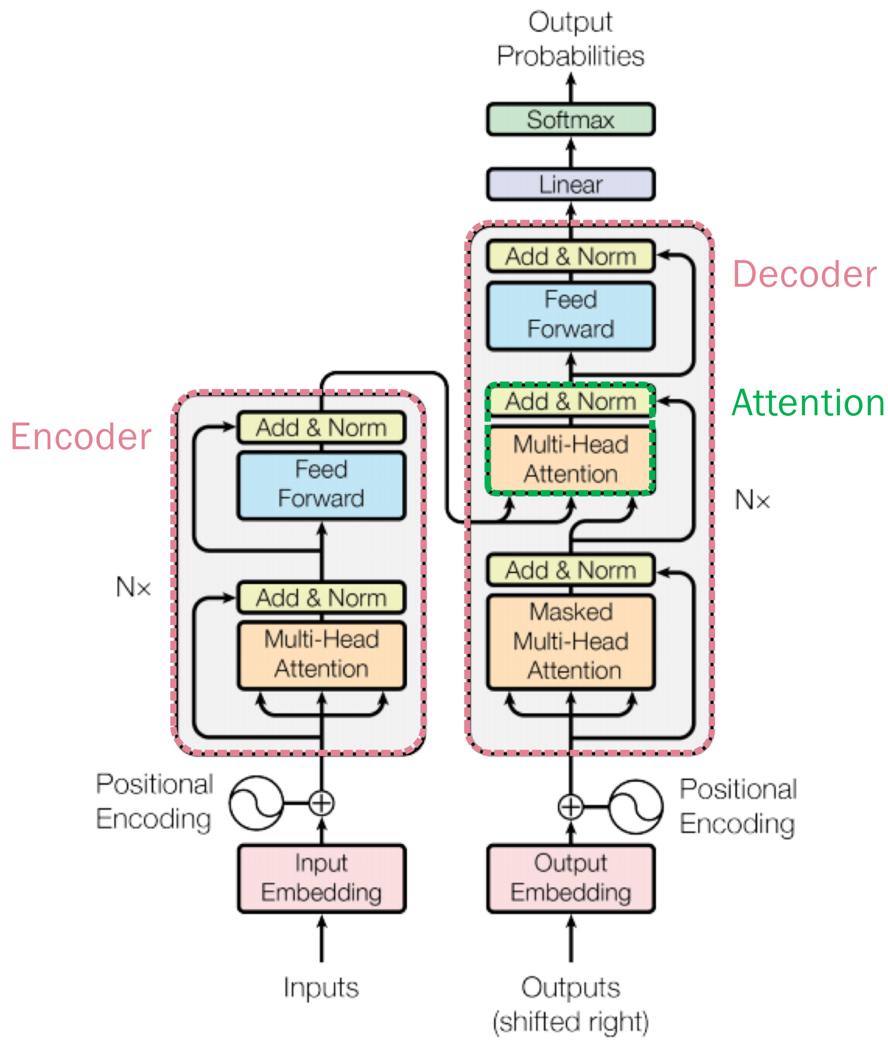
# Attention-only architecture



# Attention-only architecture

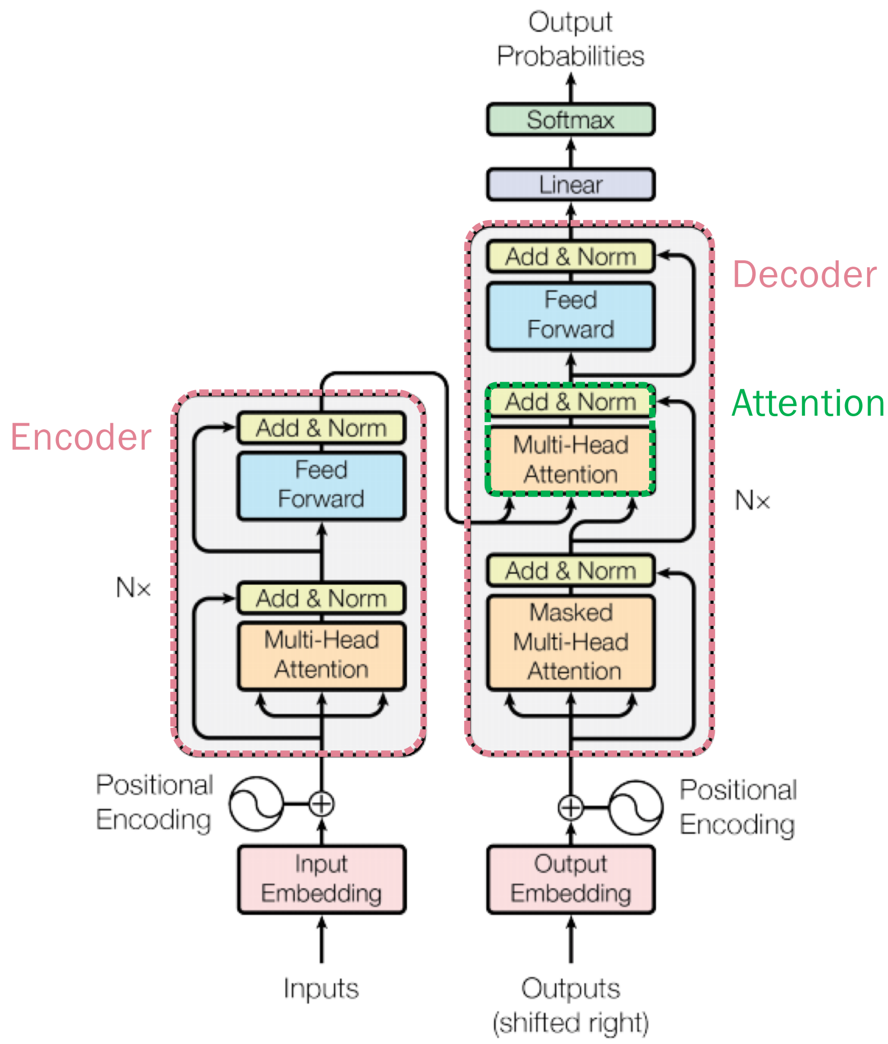


# Attention-only architecture



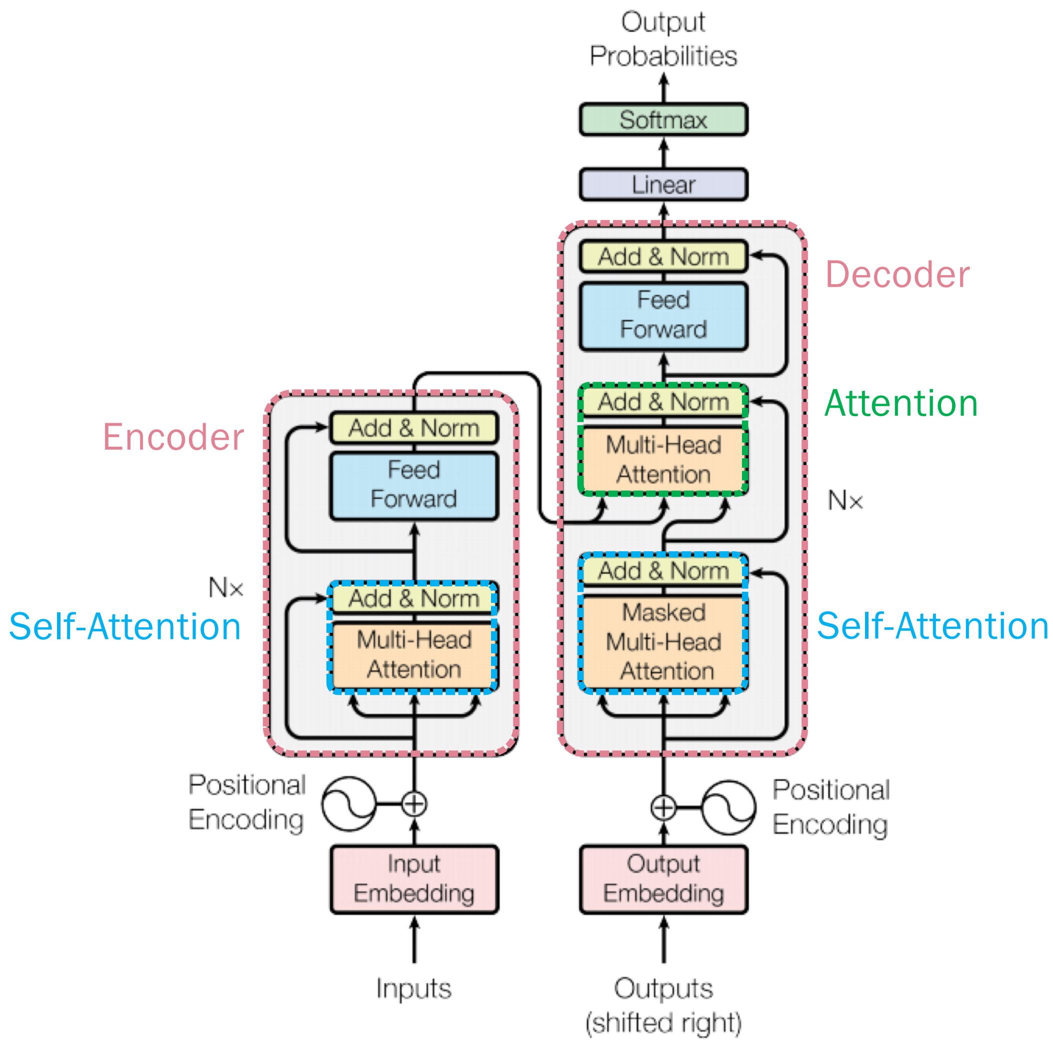
# Attention-only architecture

- Addresses limitation of RNNs due to their sequential nature (complexity, time, maximum path length)



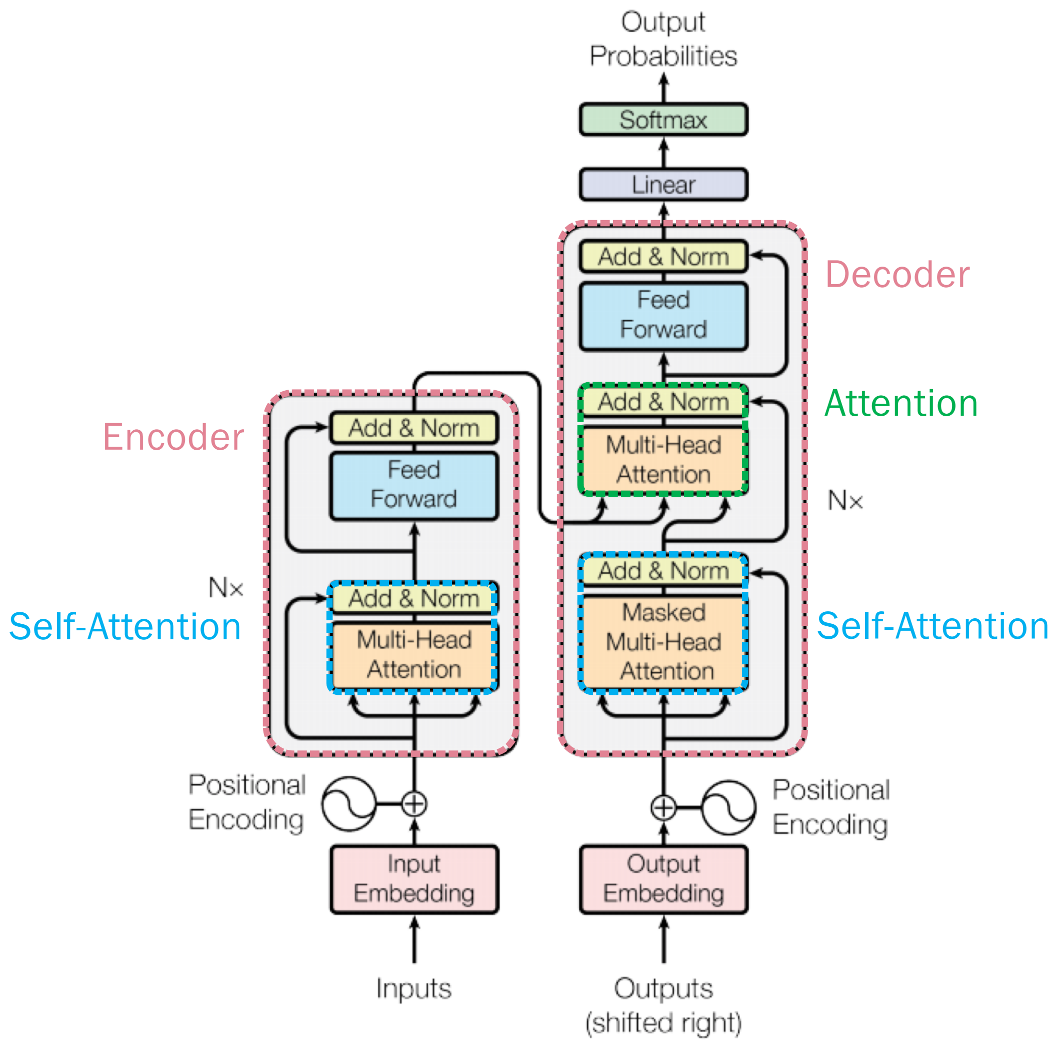
# Attention-only architecture

- Addresses limitation of RNNs due to their sequential nature (complexity, time, maximum path length)
- One central idea: substitute LSTMs with self-attention mechanisms



# Attention-only architecture

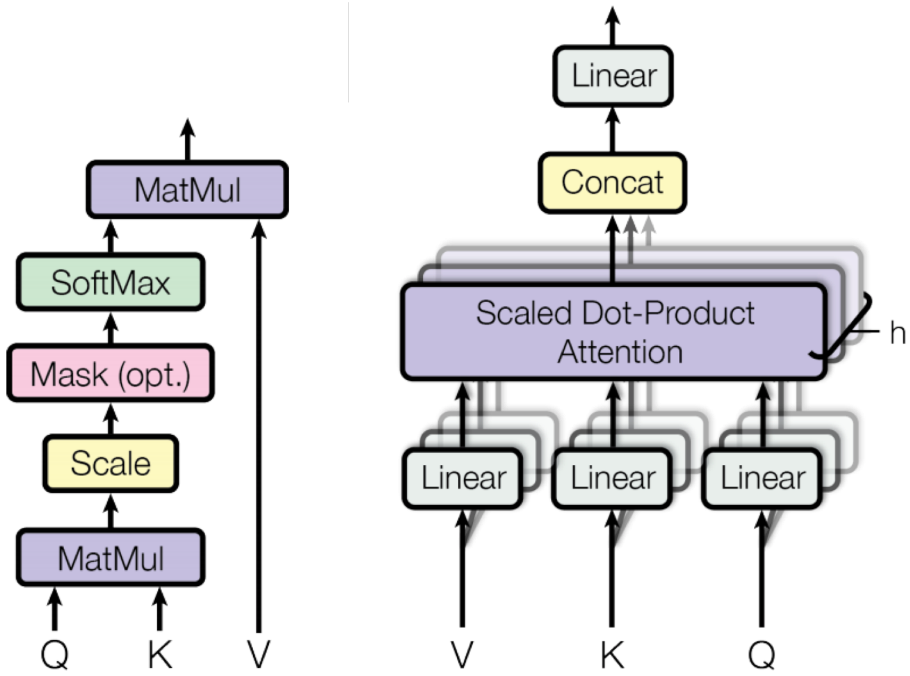
- Addresses limitation of RNNs due to their sequential nature (complexity, time, maximum path length)
- One central idea: substitute LSTMs with self-attention mechanisms
- Numerous details in implementation



# Multi-Head Attention

Changes vs. previous mechanism:

- ***Dot product (scaled)*** in place of feedforward network
- Multiple attention models performed in parallel across several (learned) ***linear projections***

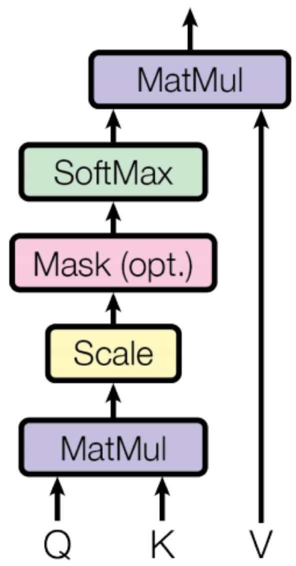



*from: Attention is All You Need – Vaswani et al. 2017*

# Attention model

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

With $d_k$ dimension of queries and keys




*from: Attention is All You Need – Vaswani et al. 2017*

# Multi-Head (1)

Original input: sequence S encompassing T words

$$S = (w_t) \text{ with } t \in [1, T]$$

After embedding ($d_{model}$ dimensions) and PE:

$$X = [X_t] \text{ with } t \in [1, T], \text{ dimension } T \times d_{model}$$

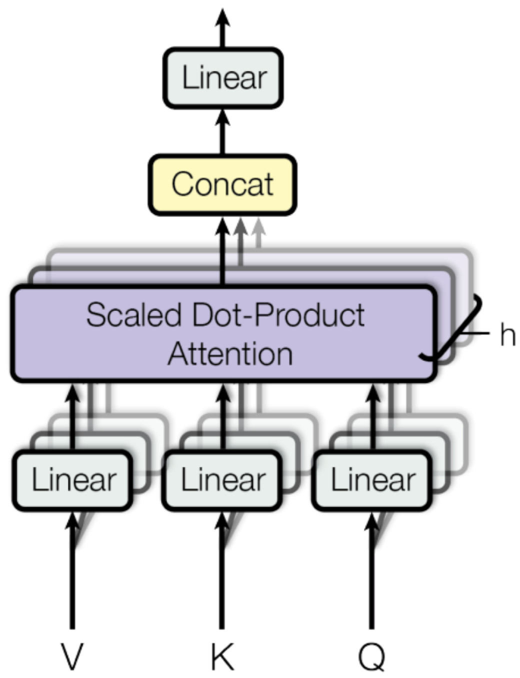
Defining the number of heads h and dimension of q, k and v

$$d_k = d_{model}/h$$

Finally defining 3 (learned) linear projections per head $i$

$$W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{model} \times d_k}$$

And a linear transformation $W^O \in \mathbb{R}^{d_{model} \times d_{model}}$




*from: Attention is All You Need – Vaswani et al. 2017*

# Multi-Head (2)

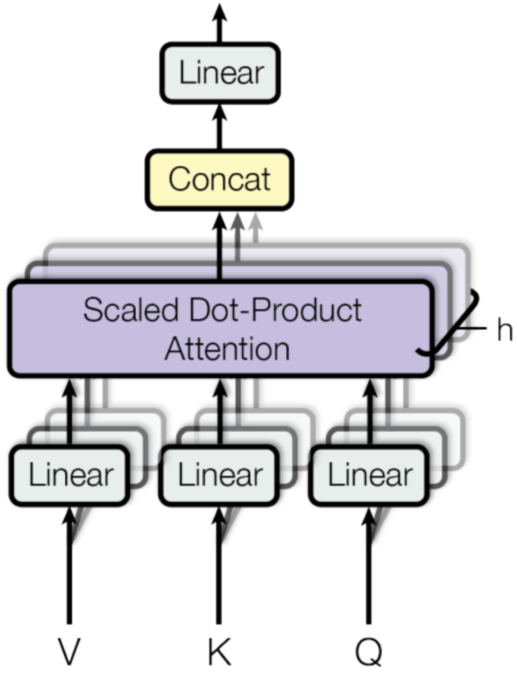
In the case of the encoder, ***all keys, values and queries*** come from the output of the previous layer of the encoder $X^{(l-1)}$, with $X^{(0)} = X$

For each head $i$:

$$Q_i \stackrel{\text{def}}{=} X^{(l-1)} W_i^Q$$

$$K_i \stackrel{\text{def}}{=} X^{(l-1)} W_i^K$$

$$V_i \stackrel{\text{def}}{=} X^{(l-1)} W_i^V$$




*from: Attention is All You Need – Vaswani et al. 2017*

# Multi-Head (3)

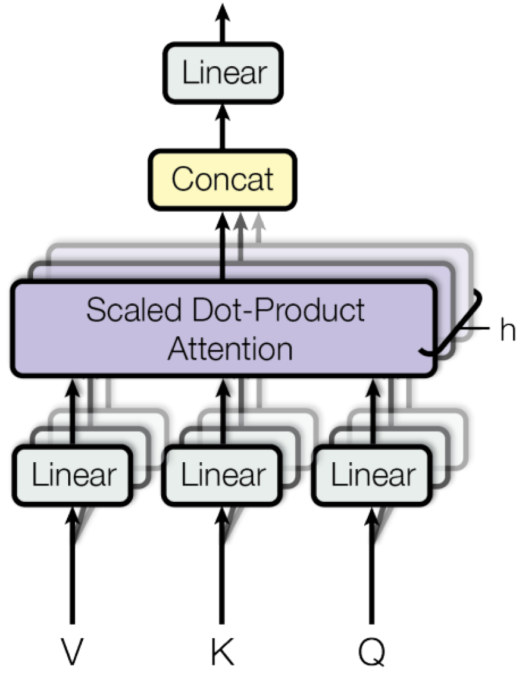
Every head has therefore the following form (dimension $T \times d_k$):

$$head_i \stackrel{\text{def}}{=} \text{Attention}(Q_i, K_i, V_i)$$

The multi-head is a (learned) linear transform of the concatenation of these different representation subspaces:

$$\text{MultiHead}(\text{X}^{(\text{l-1})}) \stackrel{\text{def}}{=} \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O$$

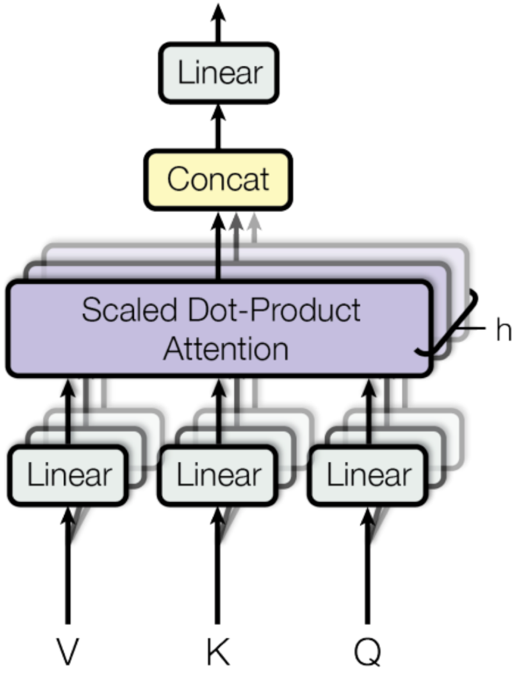
Producing T "annotations" or "hidden states" of dimension $d_{model}$




*from: Attention is All You Need – Vaswani et al. 2017*

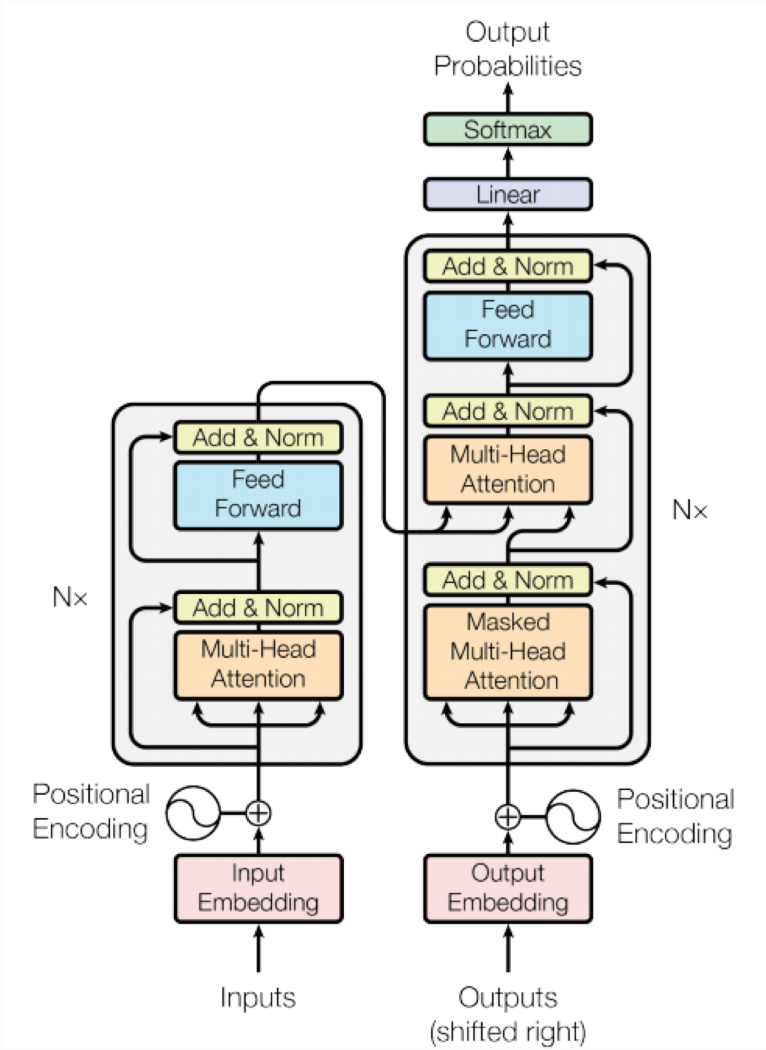
# Multi-Head (4)

- The decoder self-attention mechanism is similar to the encoder's, except that during training step ***future output values are masked.***
- For the "encoder-decoder" attention mechanism, ***queries*** come from the ***previous decoder layer*** (representation of the translated sentence at this step), while ***keys and values*** both come from the final ***output of the encoder*** (representation of the original sentence to translate).



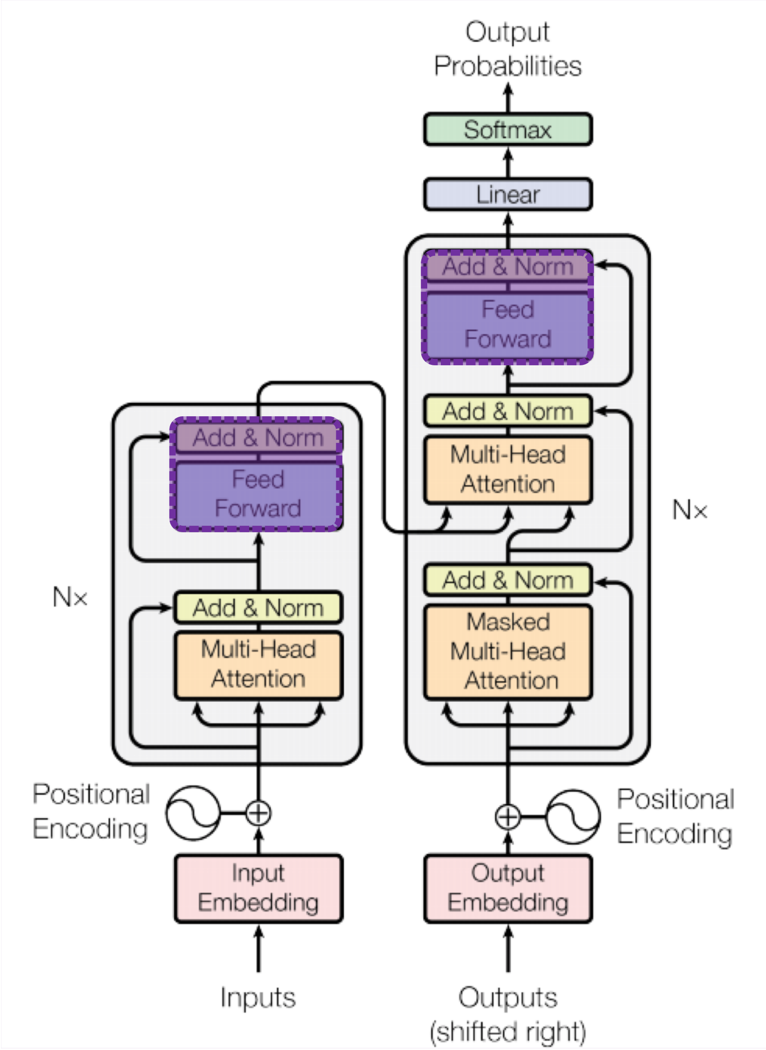

*from: Attention is All You Need – Vaswani et al. 2017*

# Other architecture components



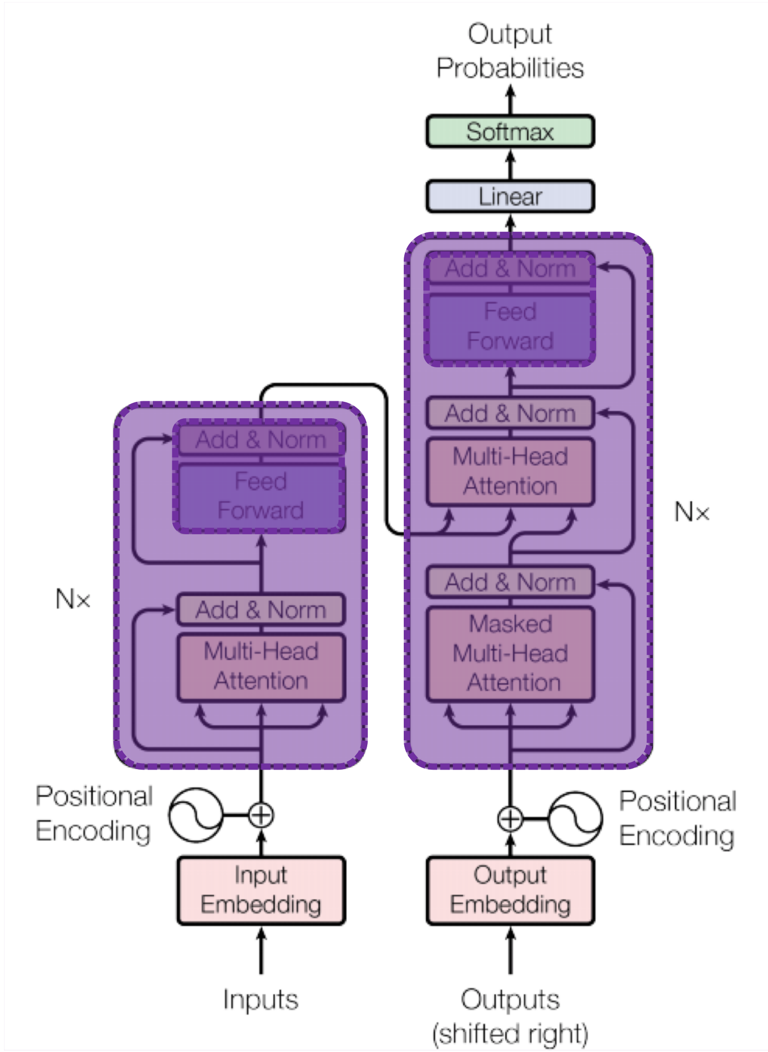
# Other architecture components

- Output of attention mechanism passed through a feedforward network



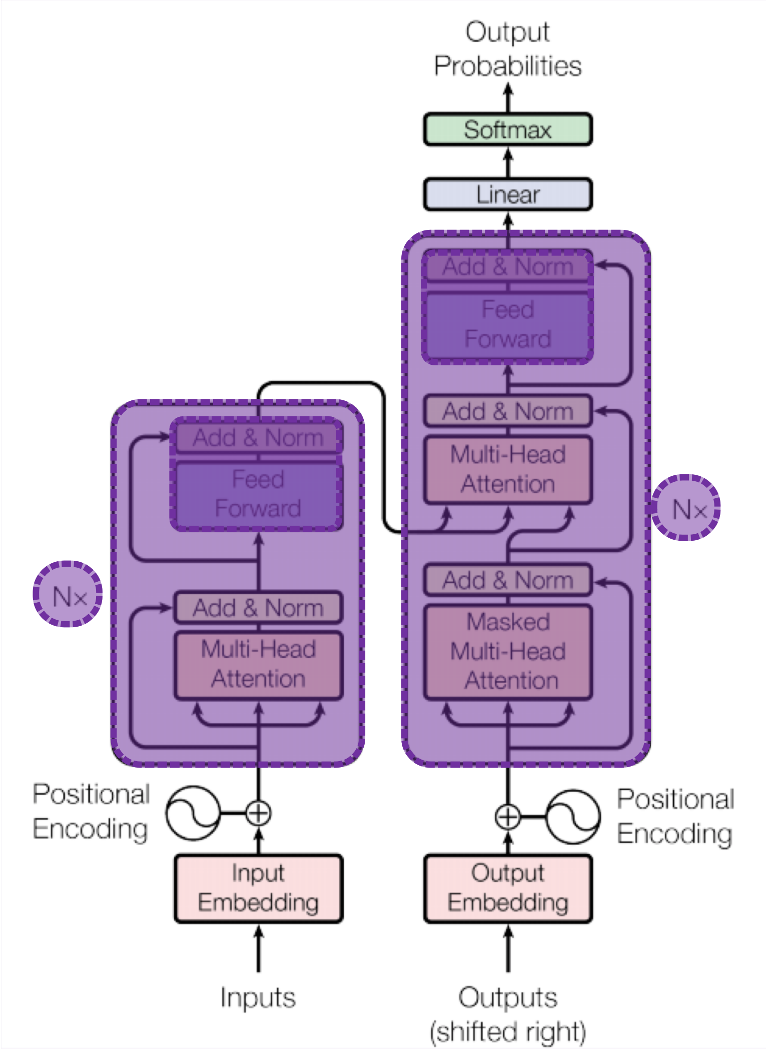
# Other architecture components

- Output of attention mechanism passed through a feedforward network
- The combination of the attention mechanism(s) and the feedforward network constitutes a layer



# Other architecture components

- Output of attention mechanism passed through a feedforward network
- The combination of the attention mechanism(s) and the feedforward network constitutes a layer
- Both encoder and decoder have 6 such stacked layers

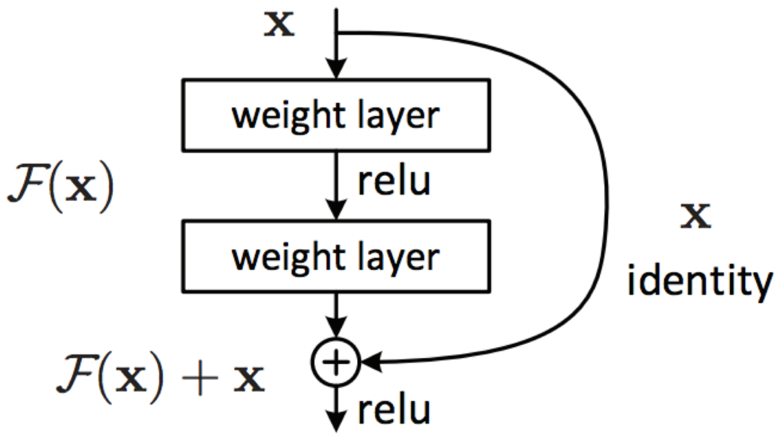


# Numerous implementation details

- **Residual Connection**
- Layer Normalization
- Scaled Dot Product
- Multi-Head
- **Linked embeddings**
- **Positional encoding**
- Residual Dropout
- **Label Smoothing**
- **Beam Search**

# Residual connection

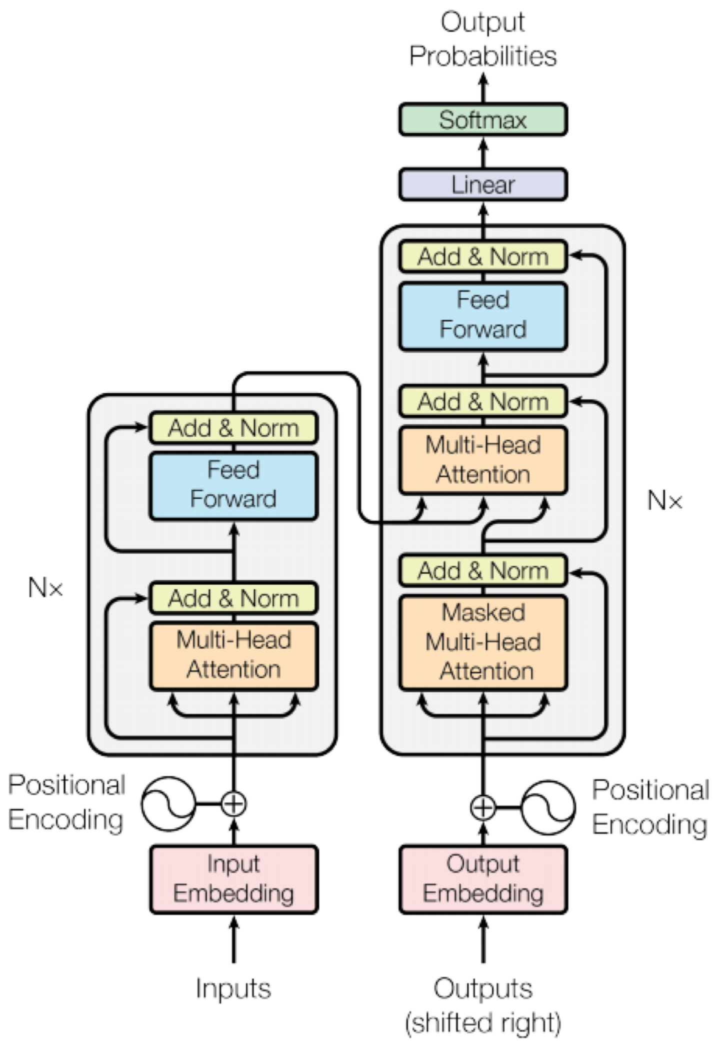
- Additional layers do not always improve performance
- Intuition: solvers struggle to fit an identity mapping
- Central idea: perform identity mapping through shortcut connection




*in: Deep Residual Learning for Image Recognition – Kaiming et al. 2015*

# Linked embeddings

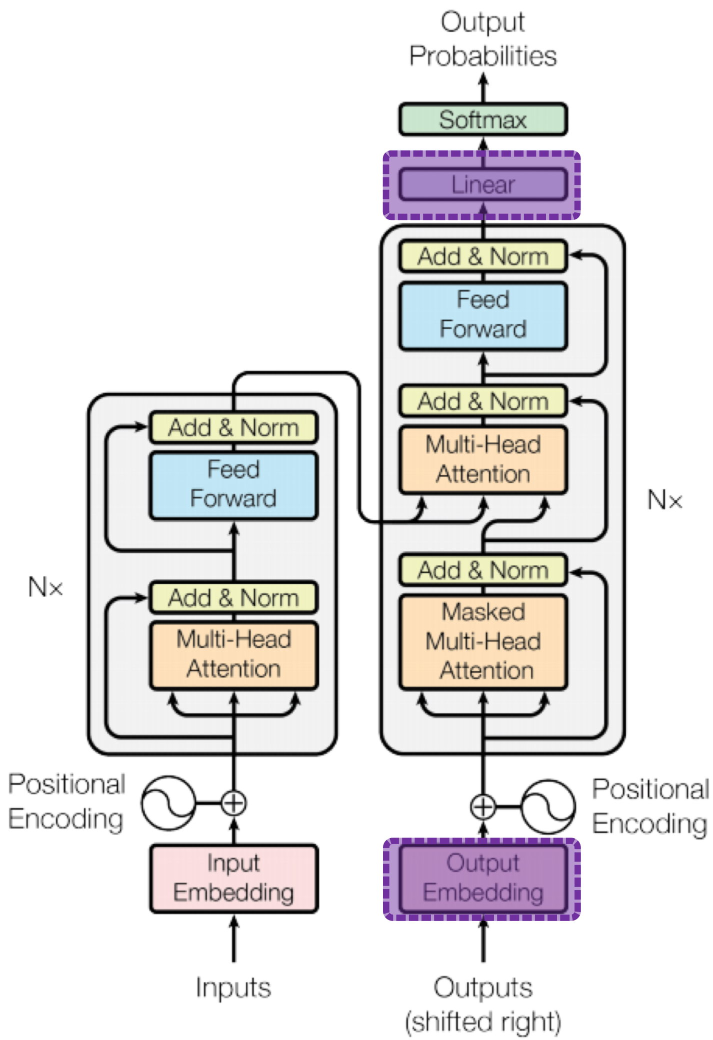
*in: Using the Output Embedding to Improve Language Models - Press et al. 2017*



# Linked embeddings

- Output layer (from continuous representation to score) has the same structure as the input embedding

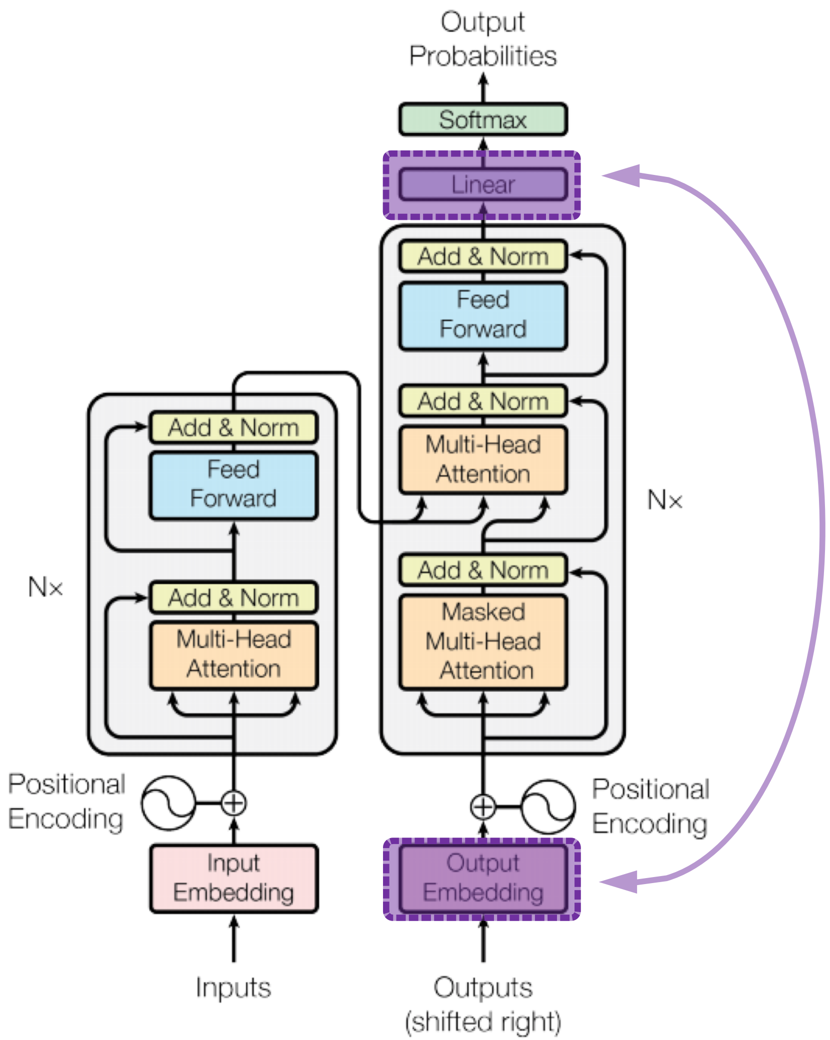
*in: Using the Output Embedding to Improve Language Models - Press et al. 2017*



# Linked embeddings

- Output layer (from continuous representation to score) has the same structure as the input embedding
- Tying them ($IE = \sqrt{d_{model}} \times OE$) improves performance

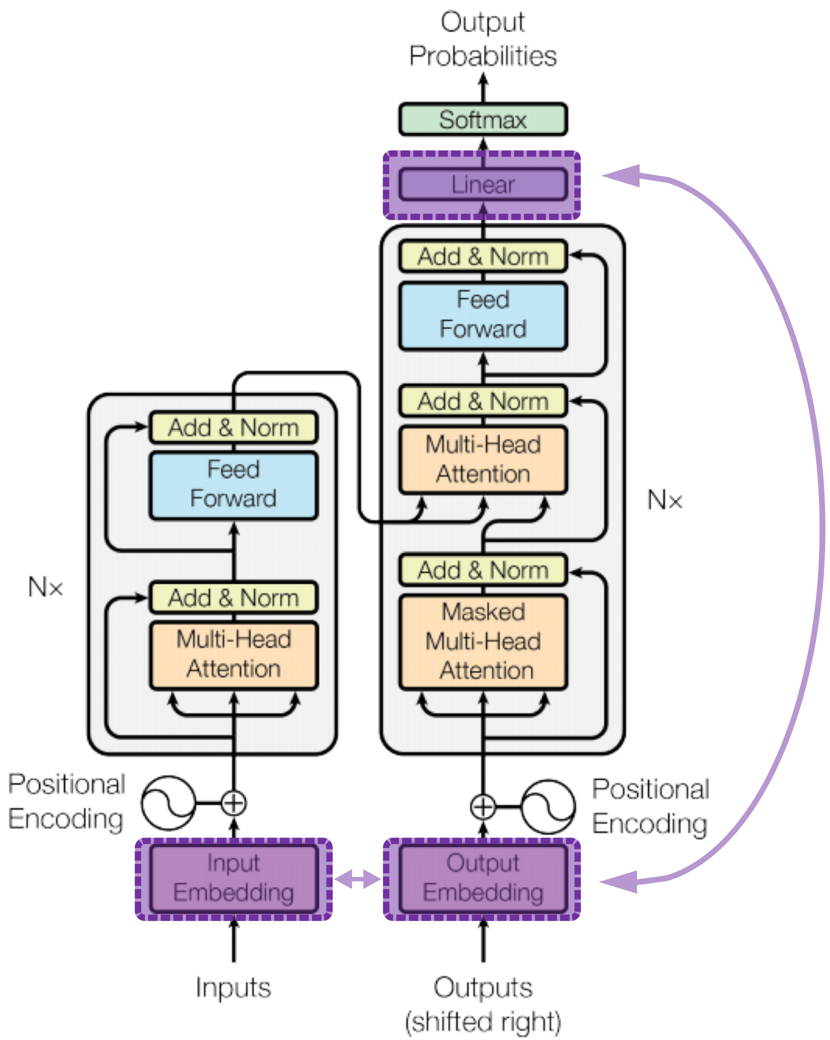
*in: Using the Output Embedding to Improve Language Models - Press et al. 2017*



# Linked embeddings

- Output layer (from continuous representation to score) has the same structure as the input embedding
- Tying them ($IE = \sqrt{d_{model}} \times OE$) improves performance
- In case of tokenisation with sub-words, can be tied 3-ways

*in: Using the Output Embedding to Improve Language Models - Press et al. 2017*



# Positional encoding

Let us define:

$$\boldsymbol{R}_{\theta} \stackrel{\text{def}}{=} \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\theta \stackrel{\text{def}}{=} 10.000^{-2/d_{model}}$$

$$P_0 \stackrel{\text{def}}{=} \left.\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}\right\} d_{model}$$

$$\boldsymbol{R} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{R}_{\theta} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \boldsymbol{R}_{\theta^{d_{model}/2}} \end{bmatrix}$$

Then if $P_n$ is the positional encoding added to the embedding vector in n[th] position,

$$P_n = \boldsymbol{R}^n \, P_0$$

# Label smoothing

During training, replace each ground truth one-hot encoded vector :

$$[0,\ 0, \dots, 0\ , \mathbf{1},\ 0,\ 0,\ \dots, 0]\ \textit{(dimension } V = \textit{size of vocabulary)}$$

With :

$$\left[\frac{\varepsilon}{V},\ \frac{\varepsilon}{V}, \dots ,\ \frac{\varepsilon}{V}, \left(\mathbf{1} - \ \frac{\boldsymbol{V-1}}{\boldsymbol{V}}\boldsymbol{\varepsilon}\right),\ \frac{\varepsilon}{V},\ \frac{\varepsilon}{V}, \dots ,\ \frac{\varepsilon}{V}\right]$$

Which acts as ***regularization*** (increases perplexity, improves accuracy and BLEU) by ***preventing the model of being "too confident"*** (i.e. overfitting) over the training data.

In the paper  the value retained for $\varepsilon$ is 0.1

*in: Rethinking the Inception Architecture for Computer Vision – Szegedy et al. 2015*

# Greedy search

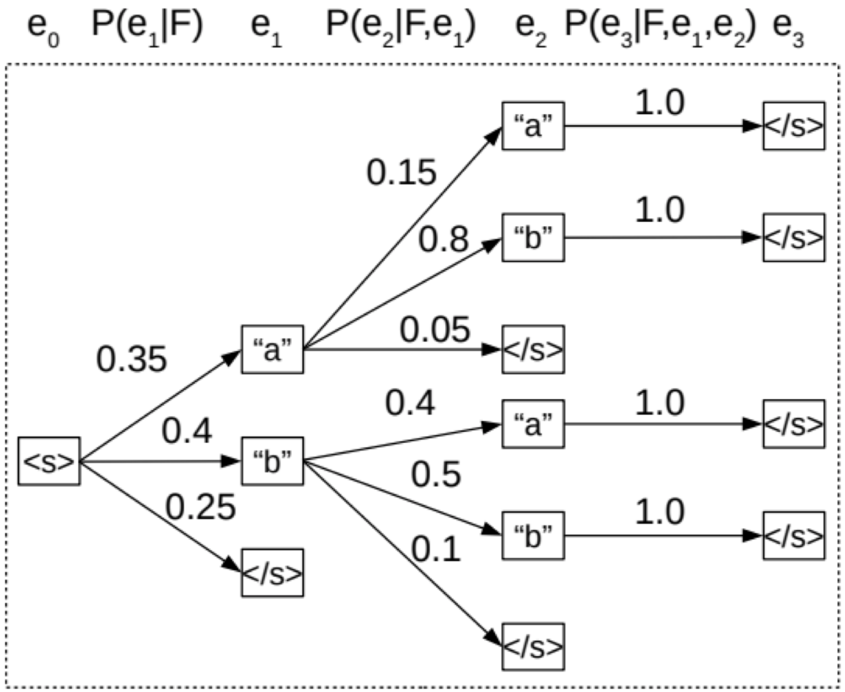



Figure 22: A search graph where greedy search fails.

*in: Neural Machine Translation and Sequence-to-sequence Models: A Tutorial – Neubig 2017*

# Beam search

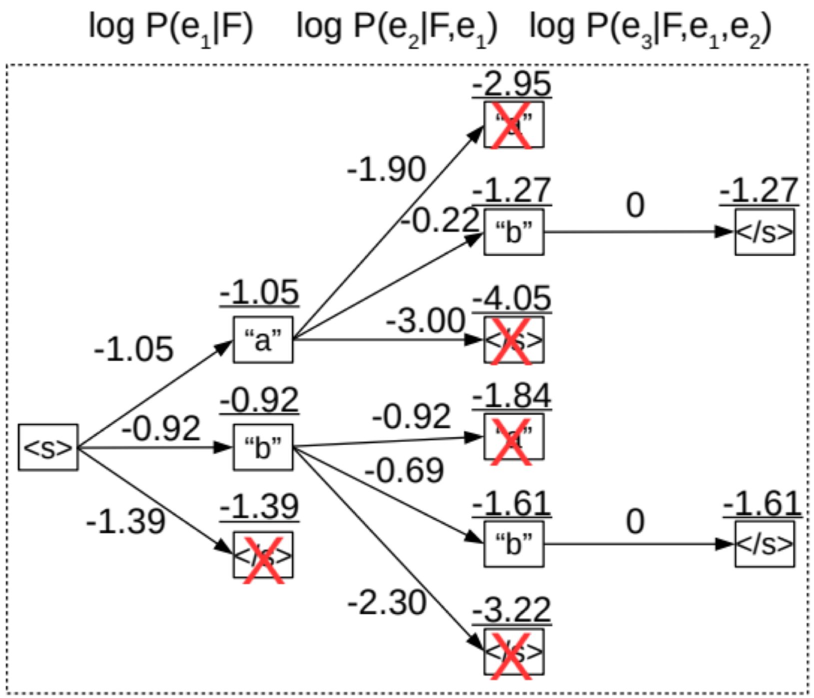



Figure 23: An example of beam search with $b = 2$. Numbers next to arrows are log probabilities for a single word $\log P(e_t \mid F, e_1^{t-1})$, while numbers above nodes are log probabilities for the entire hypothesis up until this point.

*in: Neural Machine Translation and Sequence-to-sequence Models: A Tutorial – Neubig 2017*

# Results

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 [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 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 [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 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** | **41.8** | $2.3 \cdot 10^{19}$ | |

# Moving forward…

*"**BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) is conceptually simple and empirically powerful. It obtains **new state-of-the-art** results on **eleven** natural language processing tasks (…)"*

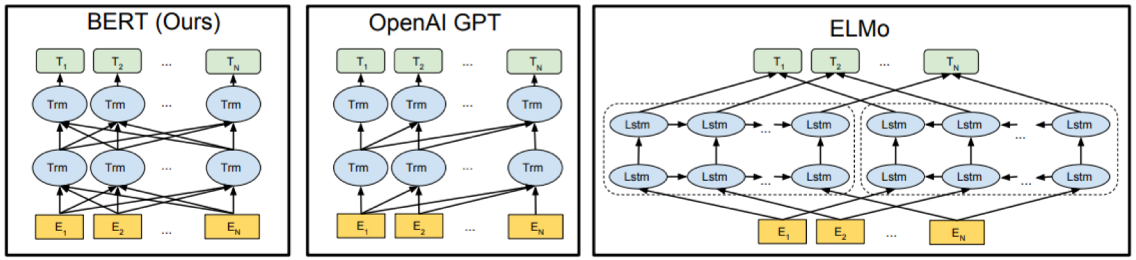



Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

*in: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding – Devlin et al. 2018*

# THAT'S ALL FOR NOW


*in: Les Shadoks, Rouxel et al. 1968*