# Attentions II

Jian Li

Tsinghua University

Seq2seq without RNN??

ConvS2S: convolutional seq2seq

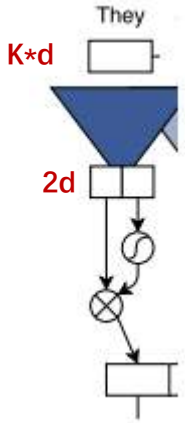
convolution+attention

**Convolutional Sequence to Sequence Learning**

- a 100% convolutional architecture to represent hierarchical representation of input sequence.
- The crux is that close input elements interact at lower layers while distant interacts at higher layers.

# Convolution filter

Each convolution kernel is parameterized as $W \in \mathbb{R}^{2d \times kd}$, $b_w \in \mathbb{R}^{2d}$ and takes as input $X \in \mathbb{R}^{k \times d}$ which is a concatenation of $k$ input elements embedded in $d$ dimensions and maps them to a single output element $Y \in \mathbb{R}^{2d}$ that has twice the dimensionality of the input elements;




Nonlinearity: Gated linear unit

$$Y = [A\ B] \in \mathbb{R}^{2d}:$$

$$v([A\ B]) = A \otimes \sigma(B)$$

Pointwise multiplication

$\sigma(B)$ control which part of A is relevant

Overall encoder-decoder structure (see later)

$\mathbf{h}^l = (h_1^l, \ldots, h_n^l)$ Output of the decoder

$\mathbf{z}^l = (z_1^l, \ldots, z_m^l)$ Output of the encoder

To enable deep convolutional networks, we add residual connections from the input of each convolution to the output of the block (He et al., 2015a).

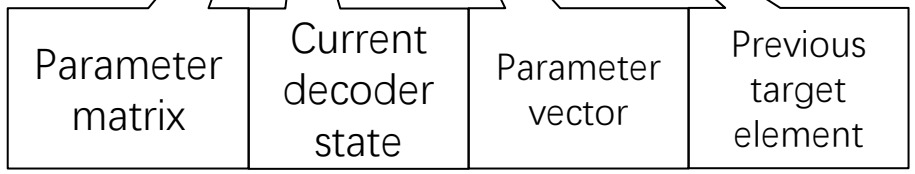
$$h_i^l = v(W^l[h_{i-k/2}^{l-1}, \ldots, h_{i+k/2}^{l-1}] + b_w^l) + h_i^{l-1}$$

# Multi-step attention

- attention mechanism for each decoder layer.

Decoder state i summary:

$$d_i^l = W_d^l h_i^l + b_d^l + g_i$$

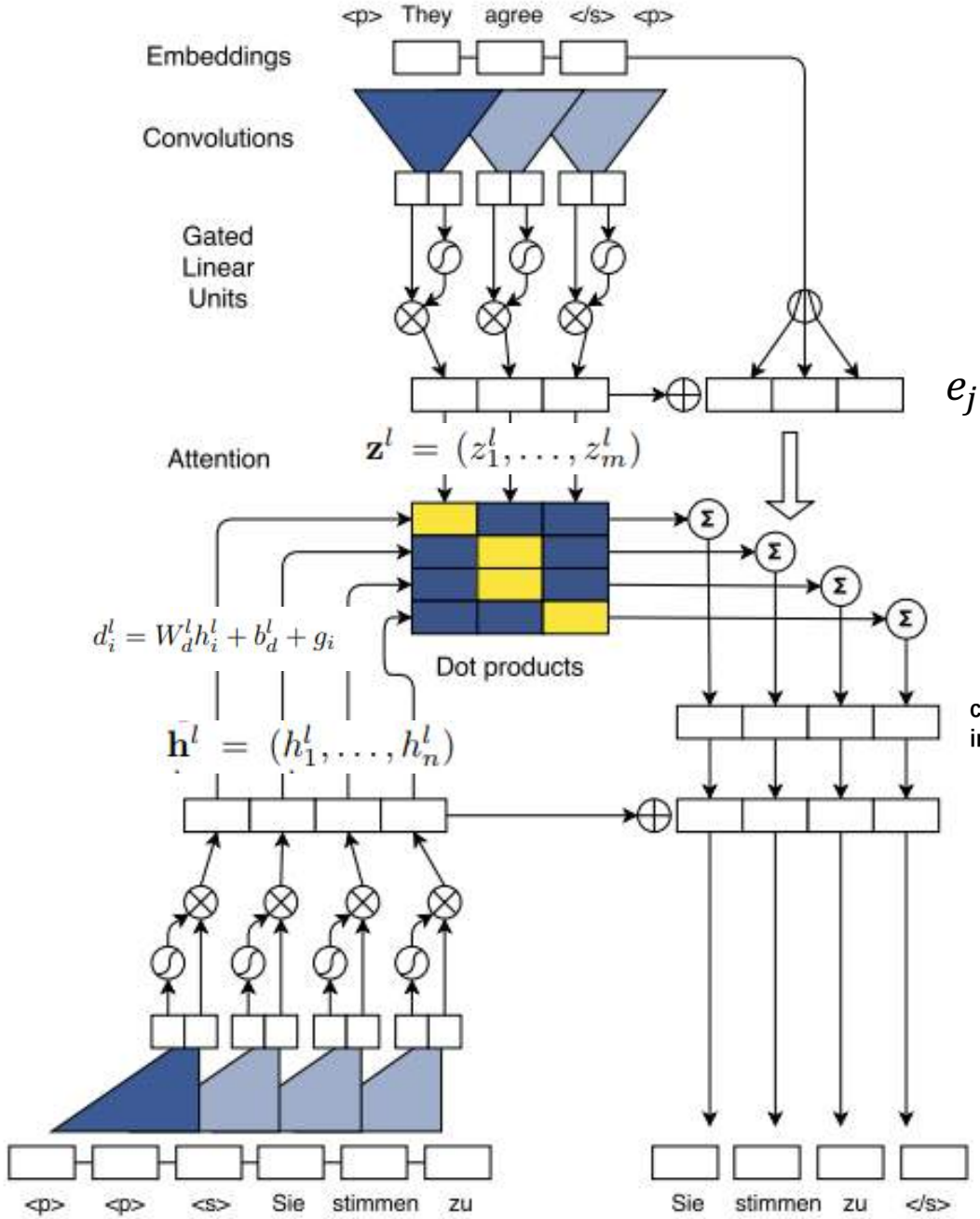



For decoder layer $l$ the attention $a_{ij}^l$ of state $i$ and source element $j$ is computed as a dot-product between the decoder state summary $d_i^l$ and each output $z_j^u$ of the last encoder block $u$:

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^{m} \exp\left(d_i^l \cdot z_t^u\right)}$$

$\mathbf{h}^l = (h_1^l, \ldots, h_n^l)$ Output of the decoder

$\mathbf{z}^l = (z_1^l, \ldots, z_m^l)$ Output of the encoder




$e_j$

*Figure 1.* Illustration of batching during training. The English source sentence is encoded (top) and we compute all attention values for the four German target words (center) simultaneously. Our attentions are just dot products between decoder context representations (bottom left) and encoder representations. We add the conditional inputs computed by the attention (center right) to the decoder states which then predict the target words (bottom right). The sigmoid and multiplicative boxes illustrate Gated Linear Units.

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^m \exp\left(d_i^l \cdot z_t^u\right)}$$

conditional input $c_i^l = \sum_{j=1}^m a_{ij}^l (z_j^u + e_j)$ $e_j$:element input embedding

$c_i^l$: *added to* $h_i^l$

padding

Training phase

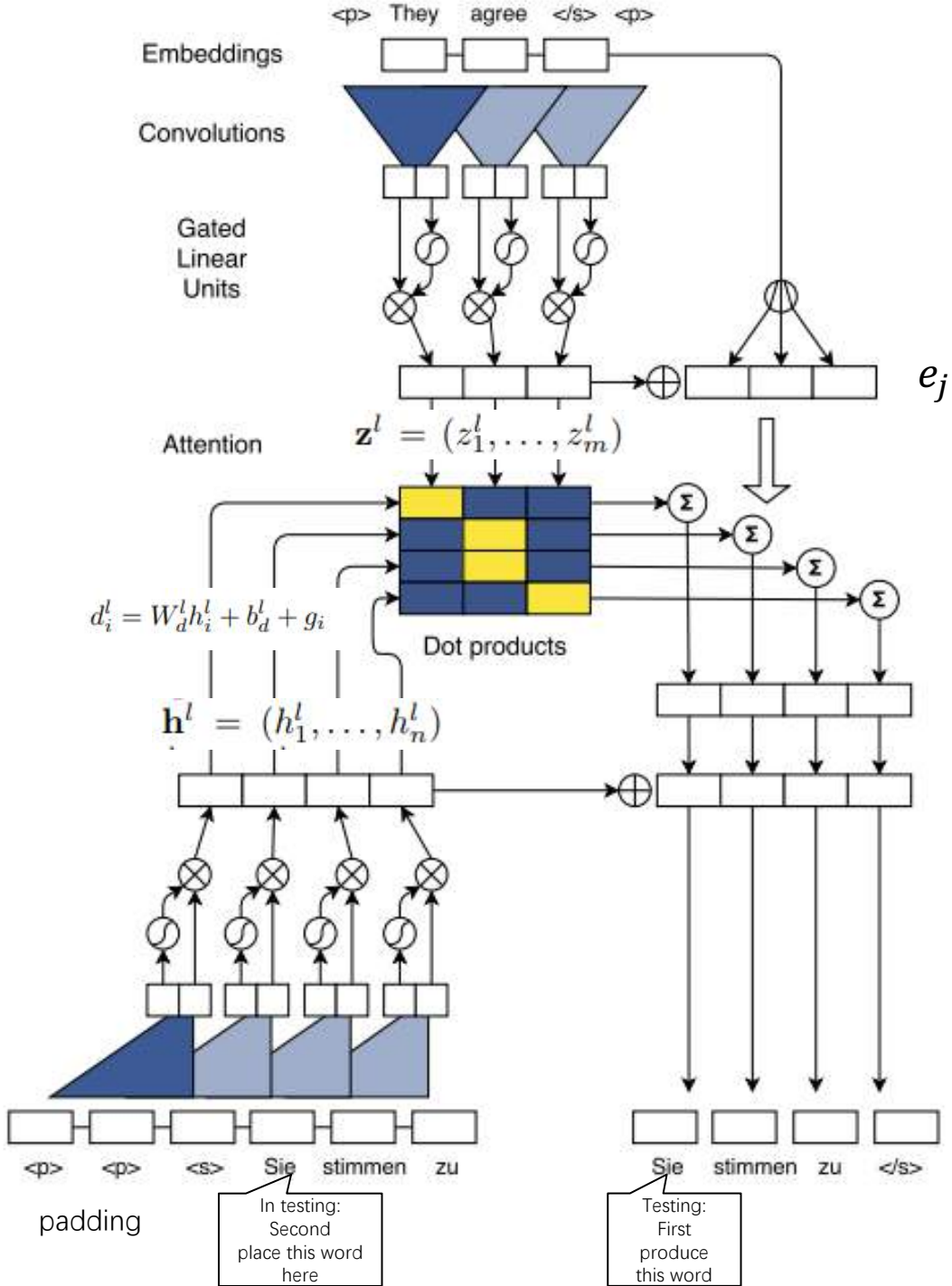



*Figure 1.* Illustration of batching during training. The English source sentence is encoded (top) and we compute all attention values for the four German target words (center) simultaneously. Our attentions are just dot products between decoder context representations (bottom left) and encoder representations. We add the conditional inputs computed by the attention (center right) to the decoder states which then predict the target words (bottom right). The sigmoid and multiplicative boxes illustrate Gated Linear Units.

$$a_{ij}^l = \frac{\exp\left(d_i^l \cdot z_j^u\right)}{\sum_{t=1}^m \exp\left(d_i^l \cdot z_t^u\right)}$$

$$c_i^l = \sum_{j=1}^m a_{ij}^l (z_j^u + e_j)$$ $e_j$:element input embedding

$c_i^l$: *added to* $h_i^l$

$\mathbf{h^L}$ final layer of the decoder just before the softmax
Thera are L convolutional layer

Output: $p(y_{i+1}|y_1, \ldots, y_i, \mathbf{x}) = \text{softmax}(W_o h_i^L + b_o) \in \mathbb{R}^T$

# Multi-step attention (in testing)


In particular, the attention of the first layer determines a useful source context which is then fed to the second layer that takes this information into account when computing attention etc.
The decoder also has immediate access to the attention history of the k − 1 previous time steps because the conditional inputs

# A trick: positional embedding

First, we embed input elements $\mathbf{x} = (x_1, \ldots, x_m)$ in distributional space as $\mathbf{w} = (w_1, \ldots, w_m)$, where $w_j \in \mathbb{R}^f$ is a column in an embedding matrix $\mathcal{D} \in \mathbb{R}^{V \times f}$. We also equip our model with a sense of order by embedding the absolute position of input elements $\mathbf{p} = (p_1, \ldots, p_m)$ where $p_j \in \mathbb{R}^f$. Both are combined to obtain input element representations $\mathbf{e} = (w_1 + p_1, \ldots, w_m + p_m)$.

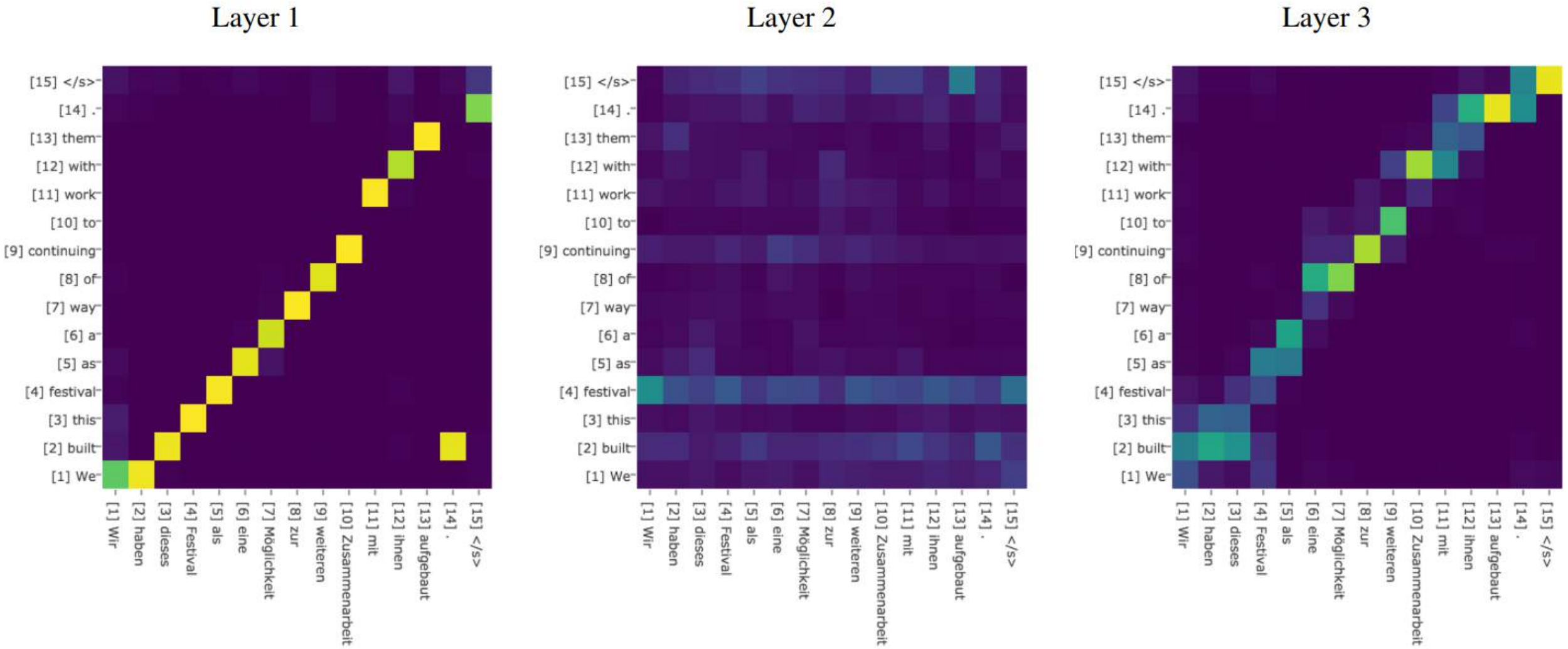
- capturing a sense of order in a sequence
- This encoding gives the model a sense of which portion of the sequence of the input (or output) it is currently dealing with. The positional encoding can be learned, or fixed. Authors made tests (PPL, BLEU) showing that both: learned and fixed positional encodings perform similarly.

| WMT'16 English-Romanian | BLEU |
|---|---|
| Sennrich et al. (2016b) GRU (BPE 90K) | 28.1 |
| ConvS2S (Word 80K) | 29.45 |
| ConvS2S (BPE 40K) | 30.02 |

| WMT'14 English-German | BLEU |
|---|---|
| Luong et al. (2015) LSTM (Word 50K) | 20.9 |
| Kalchbrenner et al. (2016) ByteNet (Char) | 23.75 |
| Wu et al. (2016) GNMT (Word 80K) | 23.12 |
| Wu et al. (2016) GNMT (Word pieces) | 24.61 |
| ConvS2S (BPE 40K) | 25.16 |

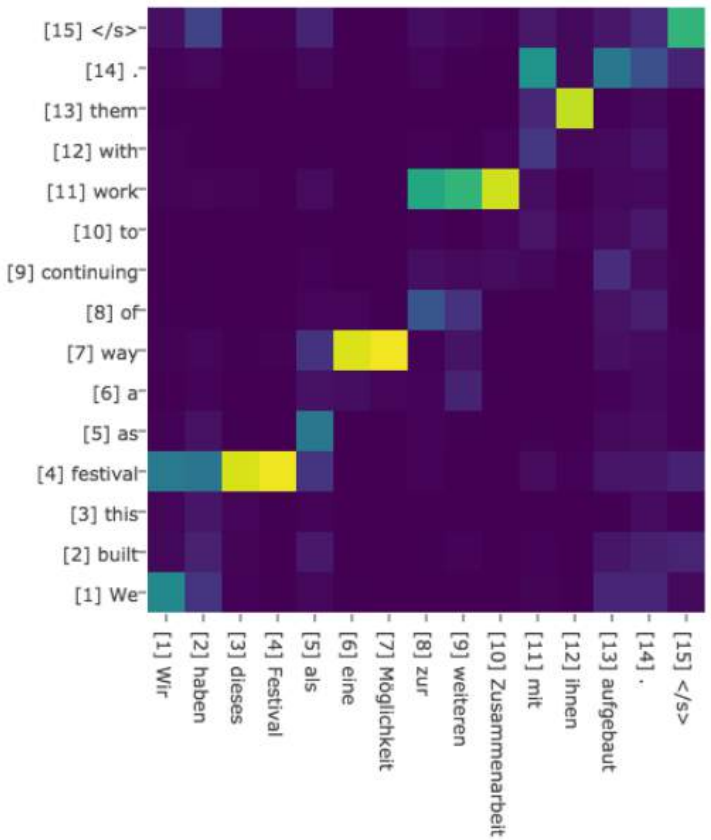
| WMT'14 English-French | BLEU |
|---|---|
| Wu et al. (2016) GNMT (Word 80K) | 37.90 |
| Wu et al. (2016) GNMT (Word pieces) | 38.95 |
| Wu et al. (2016) GNMT (Word pieces) + RL | 39.92 |
| ConvS2S (BPE 40K) | 40.51 |

*Table 1.* Accuracy on WMT tasks comapred to previous work. ConvS2S and GNMT results are averaged over several runs.

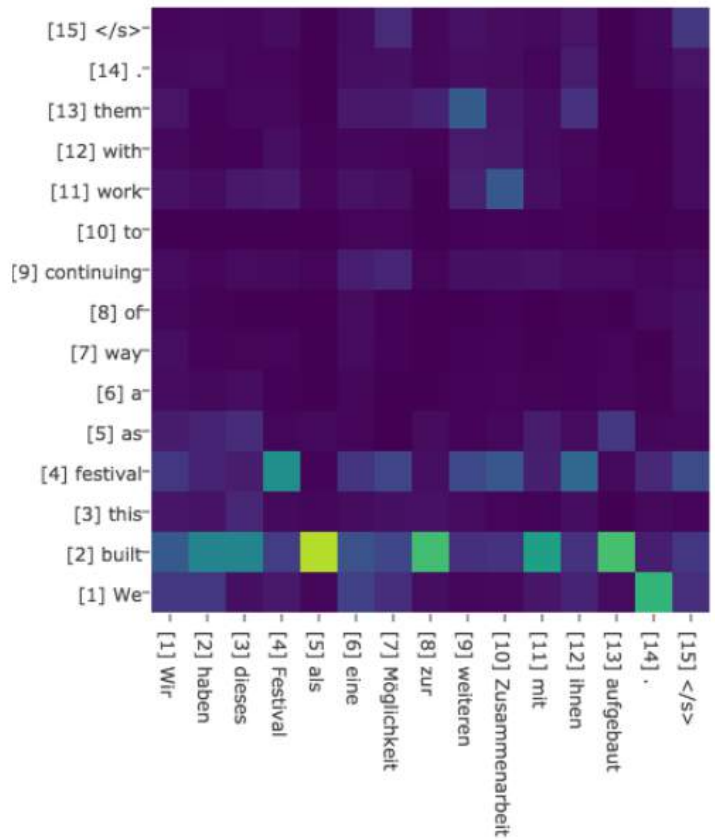



Attention scores for different decoder layers for a sentence translated from English (y-axis) to German (x-axis). This model uses 8 decoder layers and a 80k BPE vocabulary

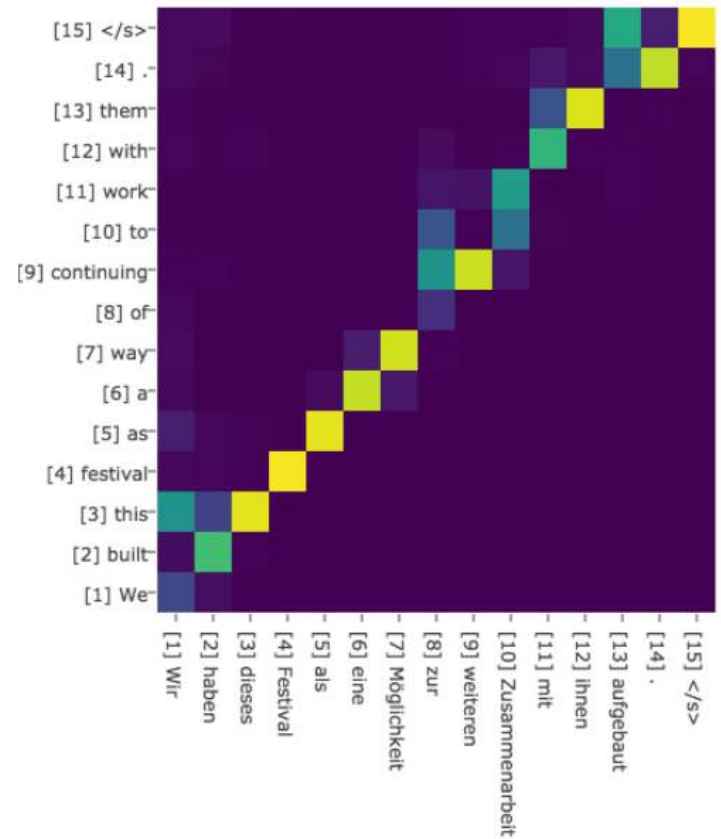
Layer 4




Layer 5




Layer 6



# Tranformer:
# Attention is all you need

# Seq2Seq

**RNN:**

- **Advantages:** are popular and successful for variable-length representations such as sequences (e.g. languages), images, etc. RNN are considered core of seq2seq (with attention). The gating models such as LSTM or GRU are for long-range error propagation.
- **Problems:** The sequentiality prohibits parallelization within instances. Long-range dependencies still tricky, despite gating. Sequence-aligned states in RNN are wasteful. Hard to model hierarchical-alike domains such as languages.

**CNN:**

- **Advantages:** Trivial to parallelize (per layer) and fit intuition that most dependencies are local.
- **Problems:** Path length between positions can be logarithmic when using dilated convolutions, left-padding for text. (autoregressive CNNs WaveNet, ByteNET )

Some material from https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/#.WtstO3puaUl

- Tranformer:

**Solution:** Multi-head self-attention mechanism.

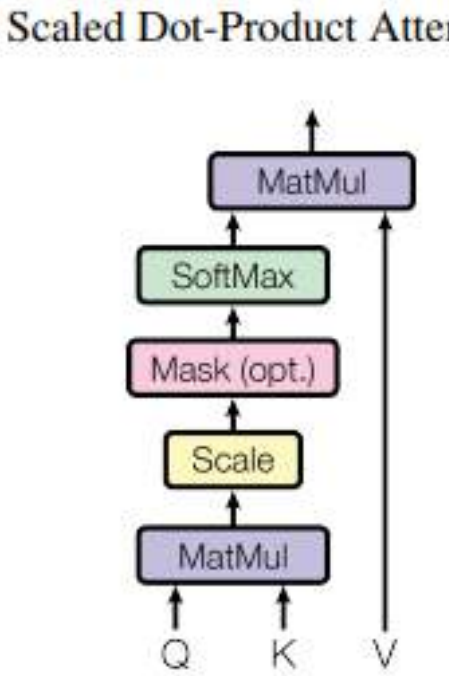
Why attention? Table 2 of the paper (later) shows that such attention networks can save 2–3 orders of magnitude of operations!

## Objective

**Parallelization of Seq2Seq:** RNN/CNN handle sequences word-by-word sequentially which is an obstacle to parallelize. Transformer achieve parallelization by replacing recurrence with attention and encoding the symbol position in sequence. This in turn leads to significantly shorter training time.

•**Reduce sequential computation:** Constant O(1) number of operations to learn dependency between two symbols independently of their position distance in sequence.

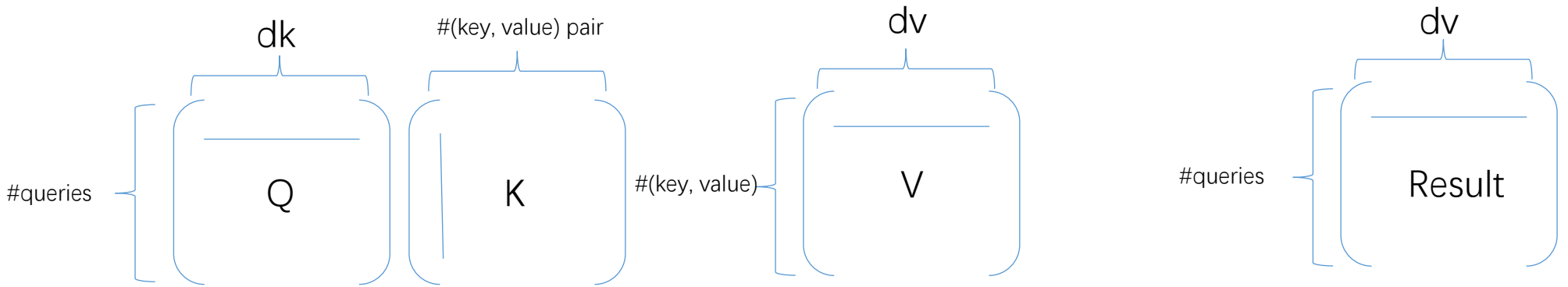
# Key Value Attention




The input consists of queries and keys of dimension dk, and values of dimension dv

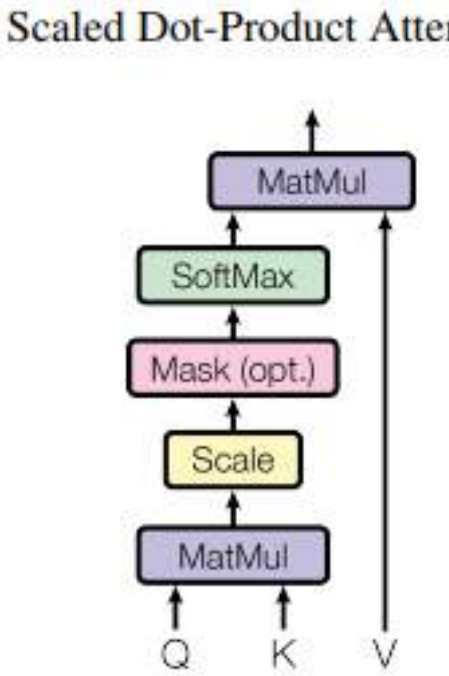
Q: queries
K: keys
V: values

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




For each query, return the most relevant (linear combination) value.
Relevance determined by <query, key>

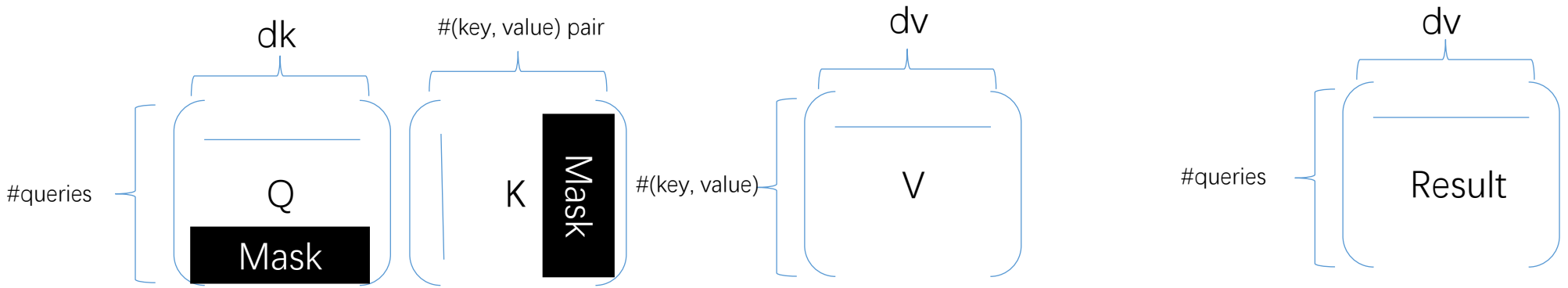
# Key Value Attention




The input consists of queries and keys of dimension dk, and values of dimension dv

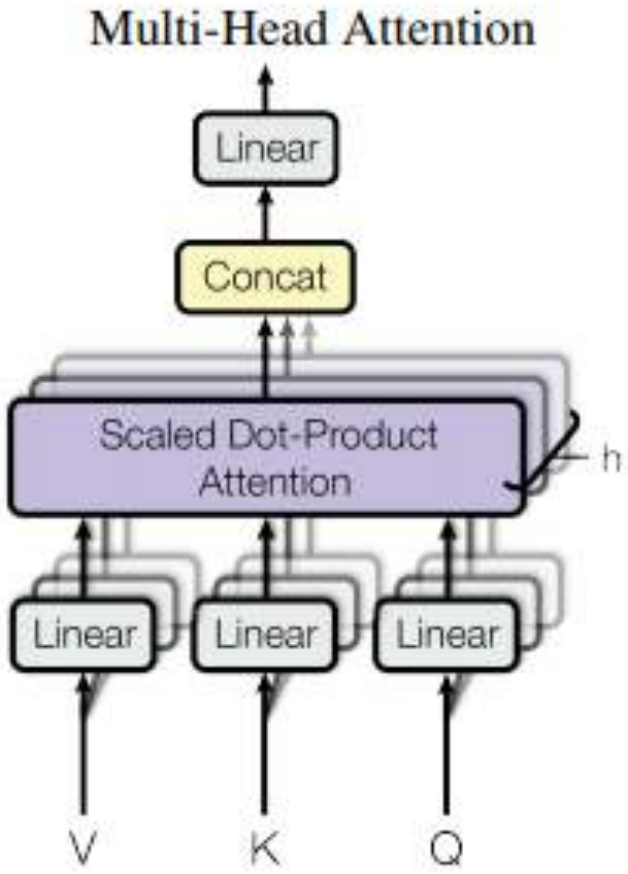
Q: queries
K: keys
V: values

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




masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections

# Multi-head attention




$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

h = 8 parallel attention layers, or heads
dmodel = 512.
dk = dv = dmodel/h = 64.

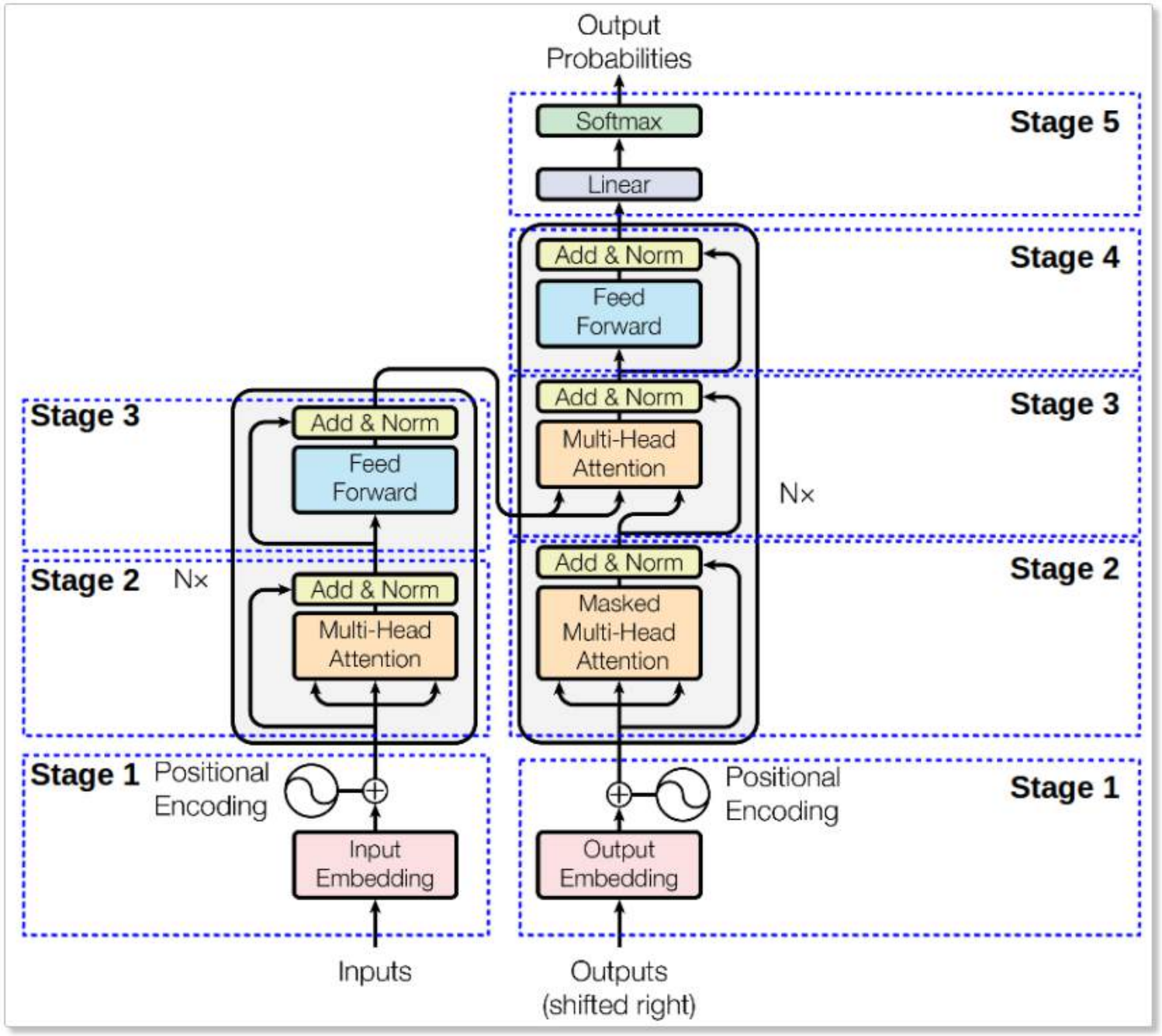



Figure 2. Single layer of Encoder (left) and Decoder (right) that is build out of $N = 6$ identical layers.

Encoder：

- Stage 1 – Encoder Input
- Stage 2 – Multi-head attention
- Stage 3 – position-wise FFN
  - Stages 2 and 3 use the residual connection

Decoder：

- Stage 1 – Decoder input
- Stage 2 Masked Multi-head attention
  - Modified to prevent positions to attend to subsequent positions.
- Stage 3 – Multi-head attention
- Stage 4 – position-wise FFN
  - Stages 2, 3 and 4 also use the residual connection followed by normalization layer at its output.

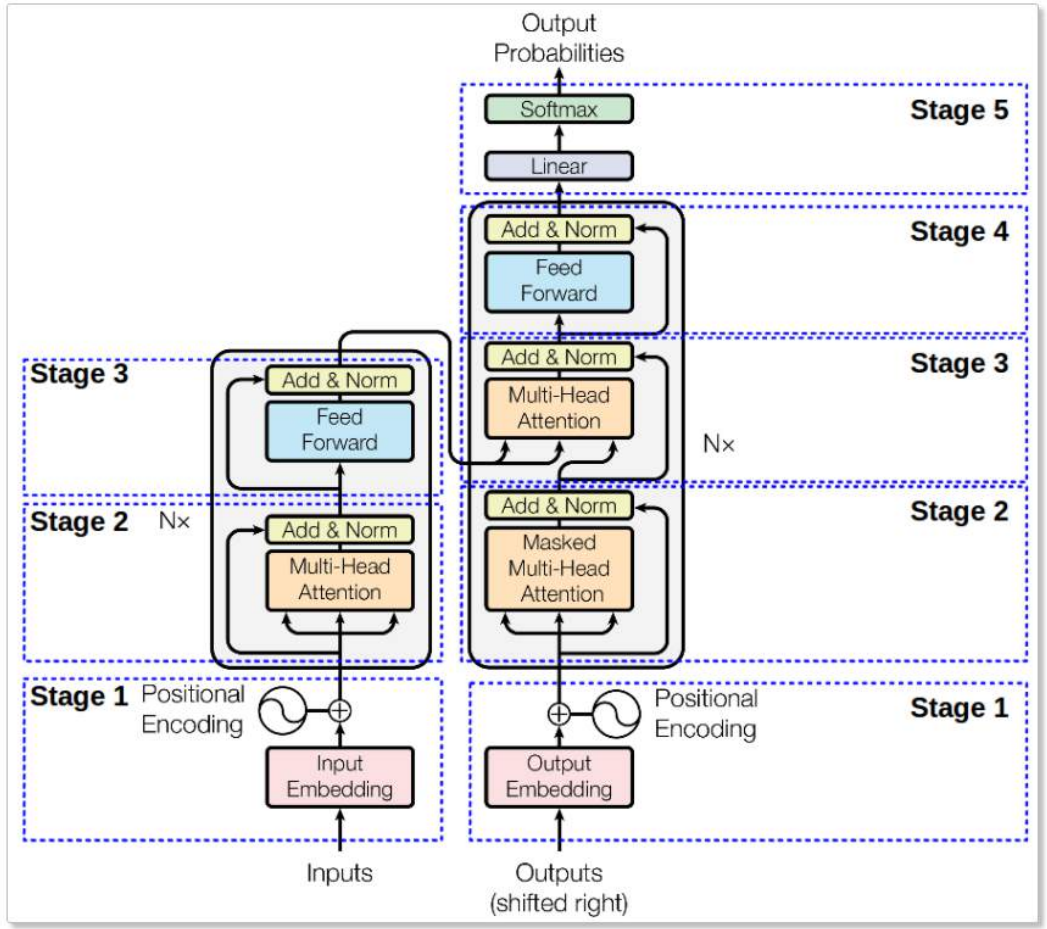



Figure 2. Single layer of Encoder (left) and Decoder (right) that is build out of $N = 6$ identical layers.

Thus, encoder works like this:

```
Stage1_out = Embedding512 + TokenPositionEncoding512
Stage2_out = layer_normalization(multihead_attention(Stage1_out) + Stage1_out)
Stage3_out = layer_normalization(FFN(Stage2_out) + Stage2_out)

out_enc = Stage3_out
```

Put together decoder works as follows:

```
Stage1_out = OutputEmbedding512 + TokenPositionEncoding512

Stage2_Mask = masked_multihead_attention(Stage1_out)
Stage2_Norm1 = layer_normalization(Stage2_Mask) + Stage1_out
Stage2_Multi = multihead_attention(Stage2_Norm1 + out_enc) +  Stage2_Norm1
Stage2_Norm2 = layer_normalization(Stage2_Multi) + Stage2_Multi

Stage3_FNN = FNN(Stage2_Norm2)
Stage3_Norm = layer_normalization(Stage3_FNN) + Stage2_Norm2

out_dec = Stage3_Norm
```

# Masked Multi-head attention

- We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.
- masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections

# Positional embedding

- Fixed positional embedding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

i: indicates the frequency
The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$

- Feed Forward Network

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

# Why use self-attention

- self-attention layer enable each position to attend to all previous positions in the decoder, including the current position.
- use of self-attention layers instead of recurrent or convolutional layers
  - self-attention layers connects all positions with O(1) number of sequentially executed operations (eg. vs O(n) in RNN)
  - Minimize maximum path length between any two input and output positions in network composed of the different layer types . The shorter the path between any combination of positions in the input and output sequences, the easier to learn long-range dependencies.

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) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

# Flowchart

# Coreference resolution

- It is worth noting that this self-attention strategy allows to face the issue of **coreference resolution** where e.g. word "*it*" in a sentence can refer to different noun of the sentence depending on context.

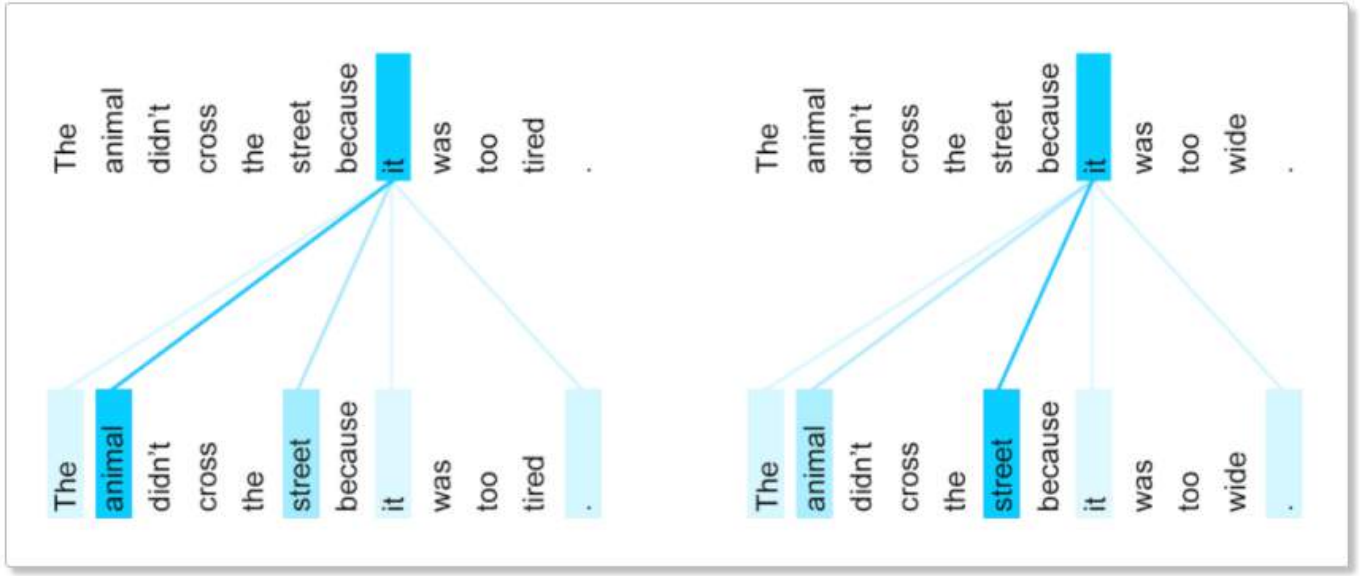



Figure 5. Co-reference resolution. The *it* in both cases relates to different token. Adopted from *Google Blog*

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 | **$3.3 \cdot 10^{18}$** | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |

## Attention Visualizations

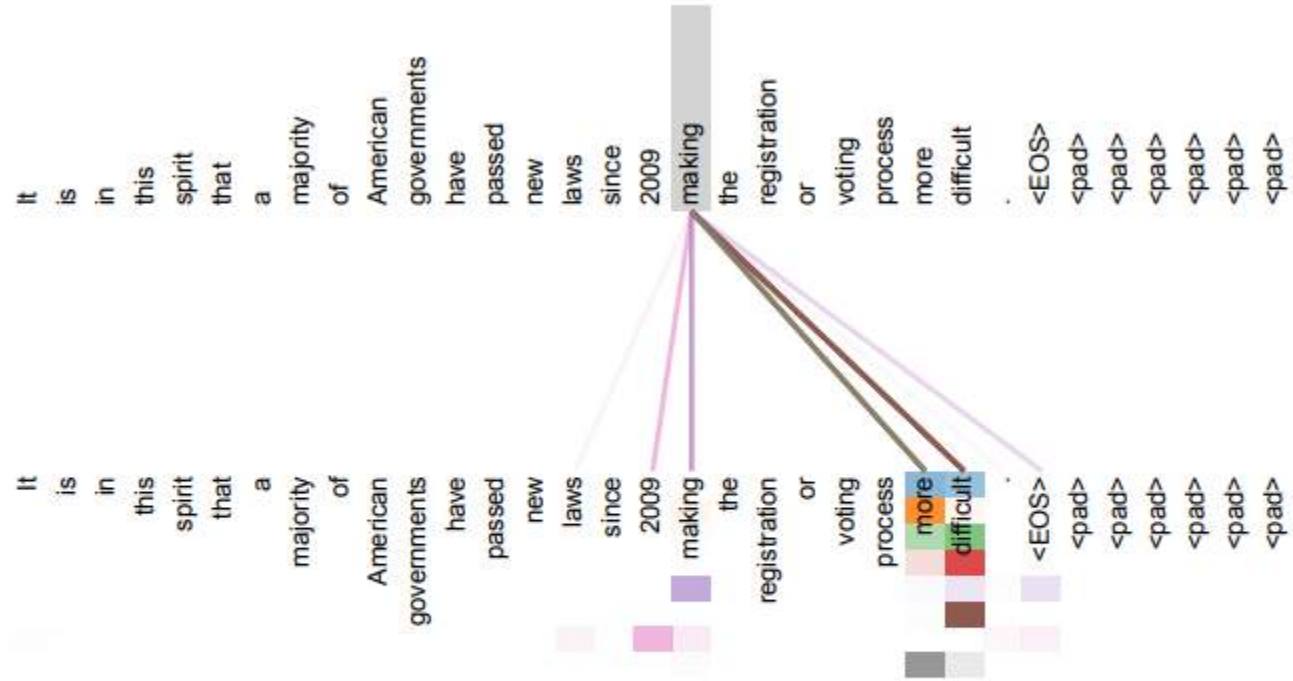



Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

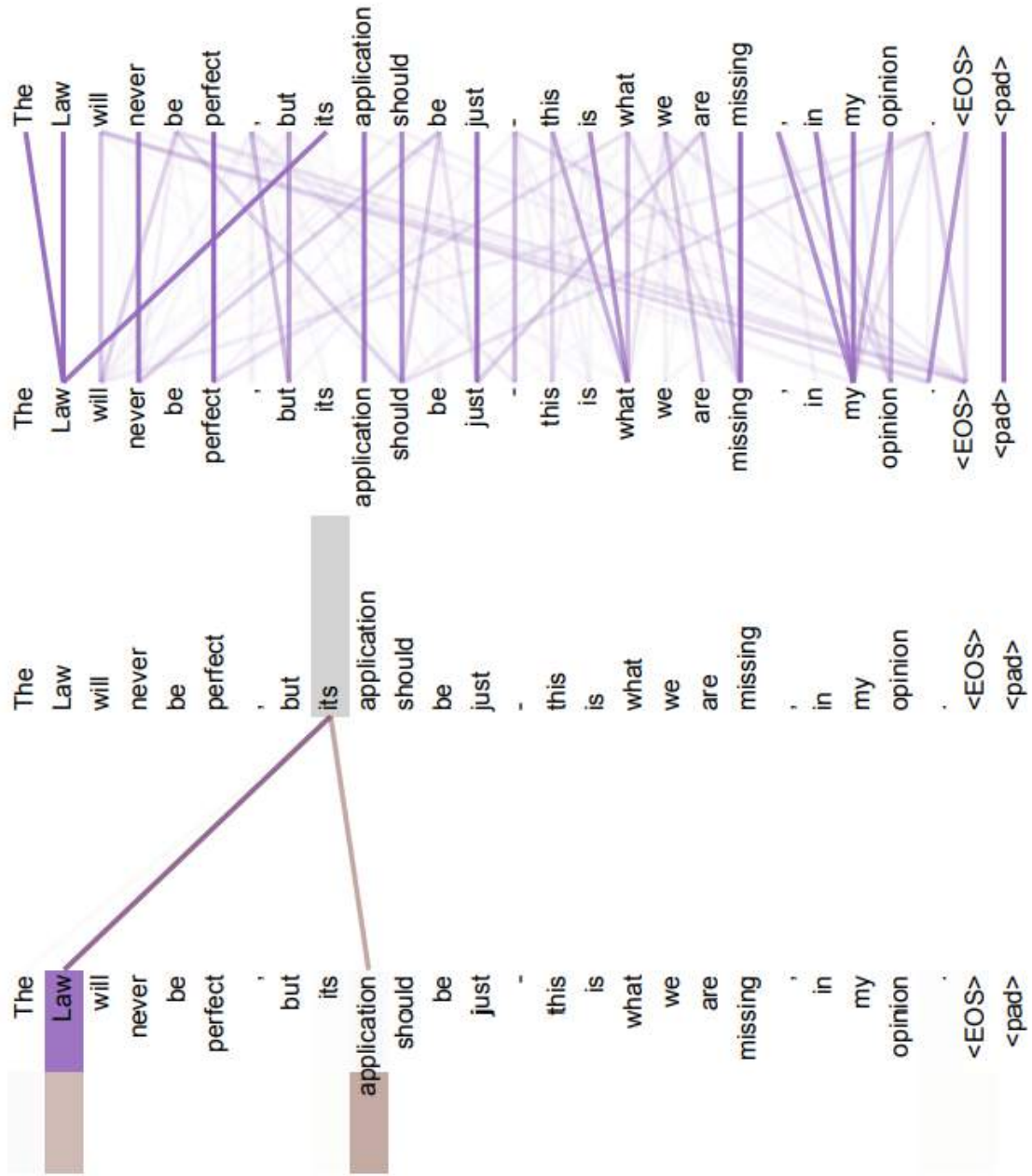
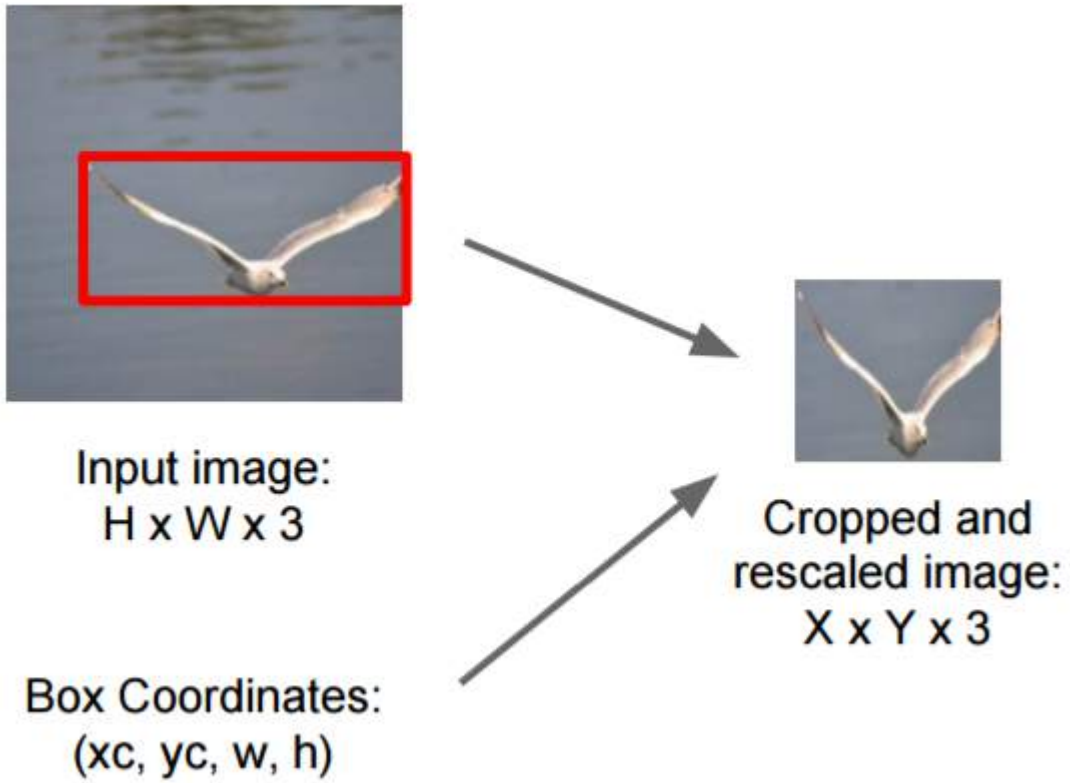


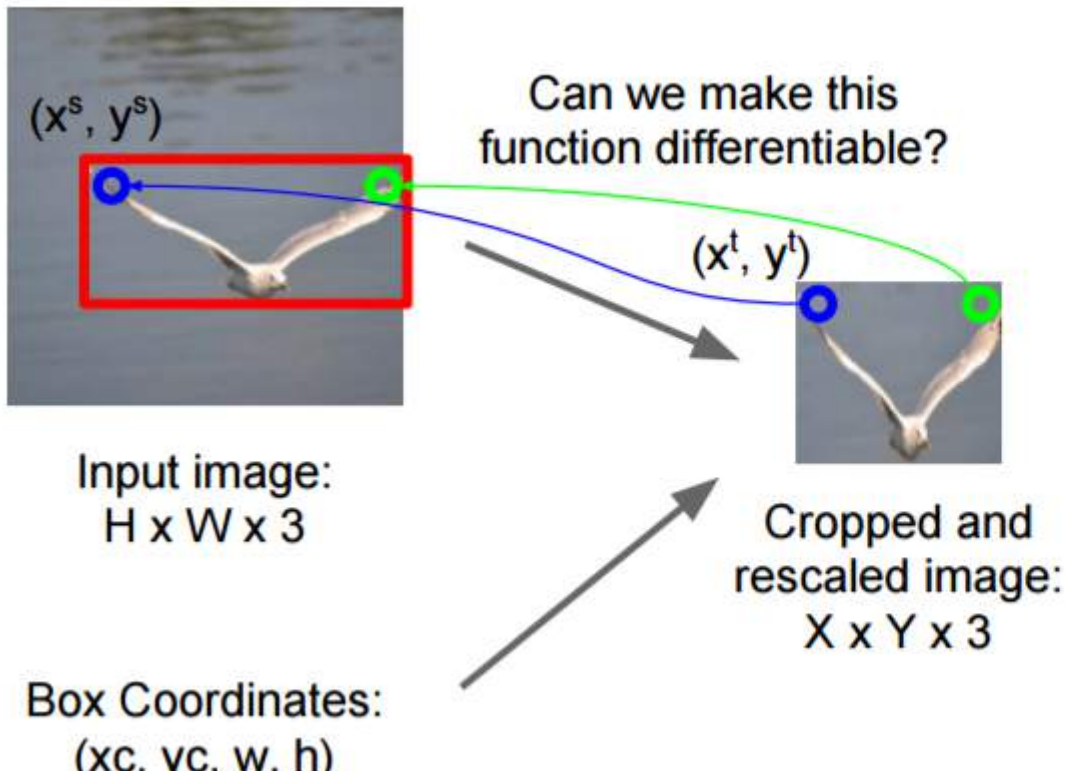

Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

# Spatial Transformer Networks
## -a different attention mechanism

Jaderberg et al, “Spatial Transformer Networks”, NIPS 2015

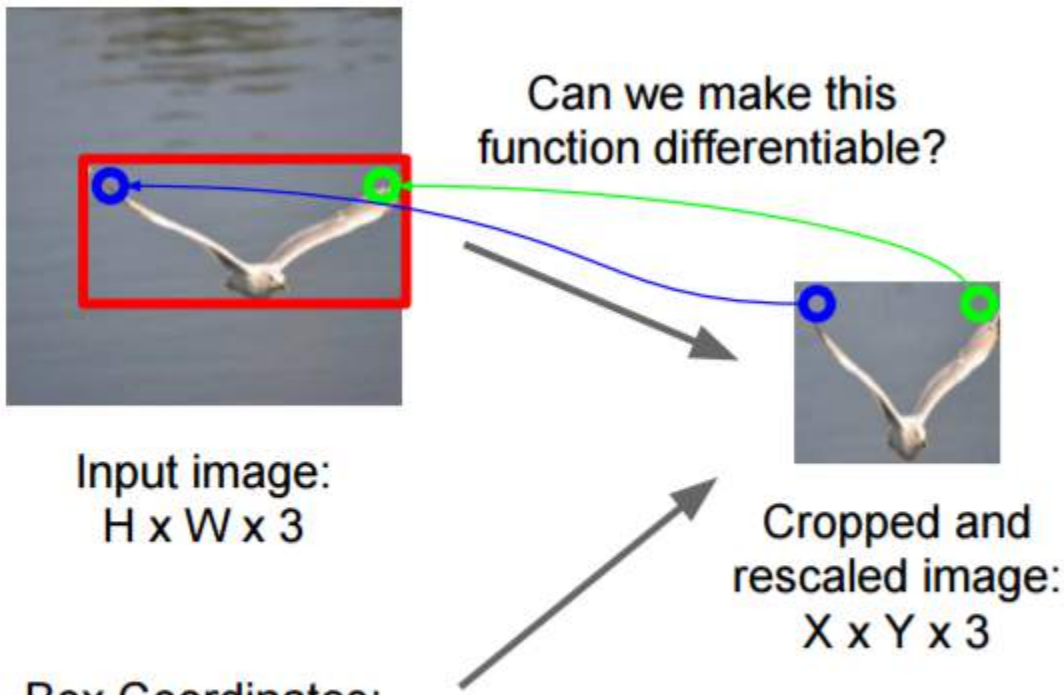
- Would like to pay **attention** to certain areas of an image






**Idea**: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

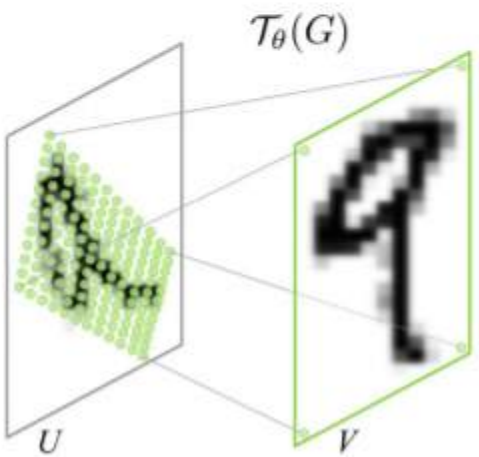
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$




**Idea**: Function mapping *pixel coordinates* (xt, yt) of output to *pixel coordinates* (xs, ys) of input

$\theta$: parameters we need to learn

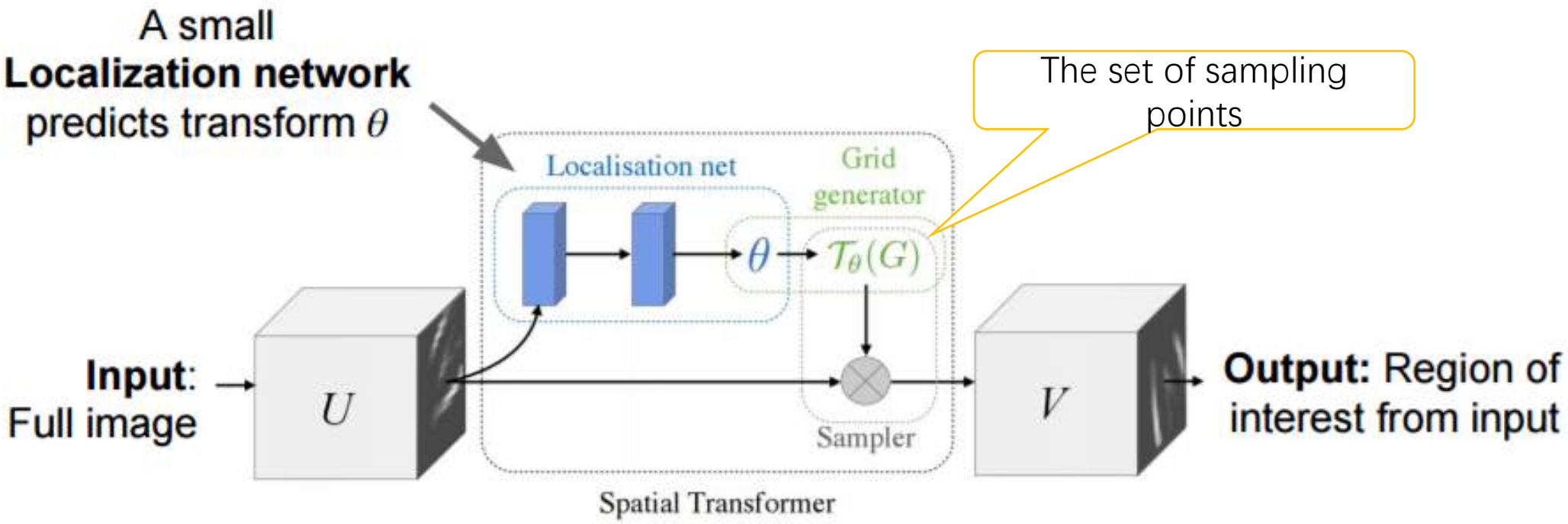
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

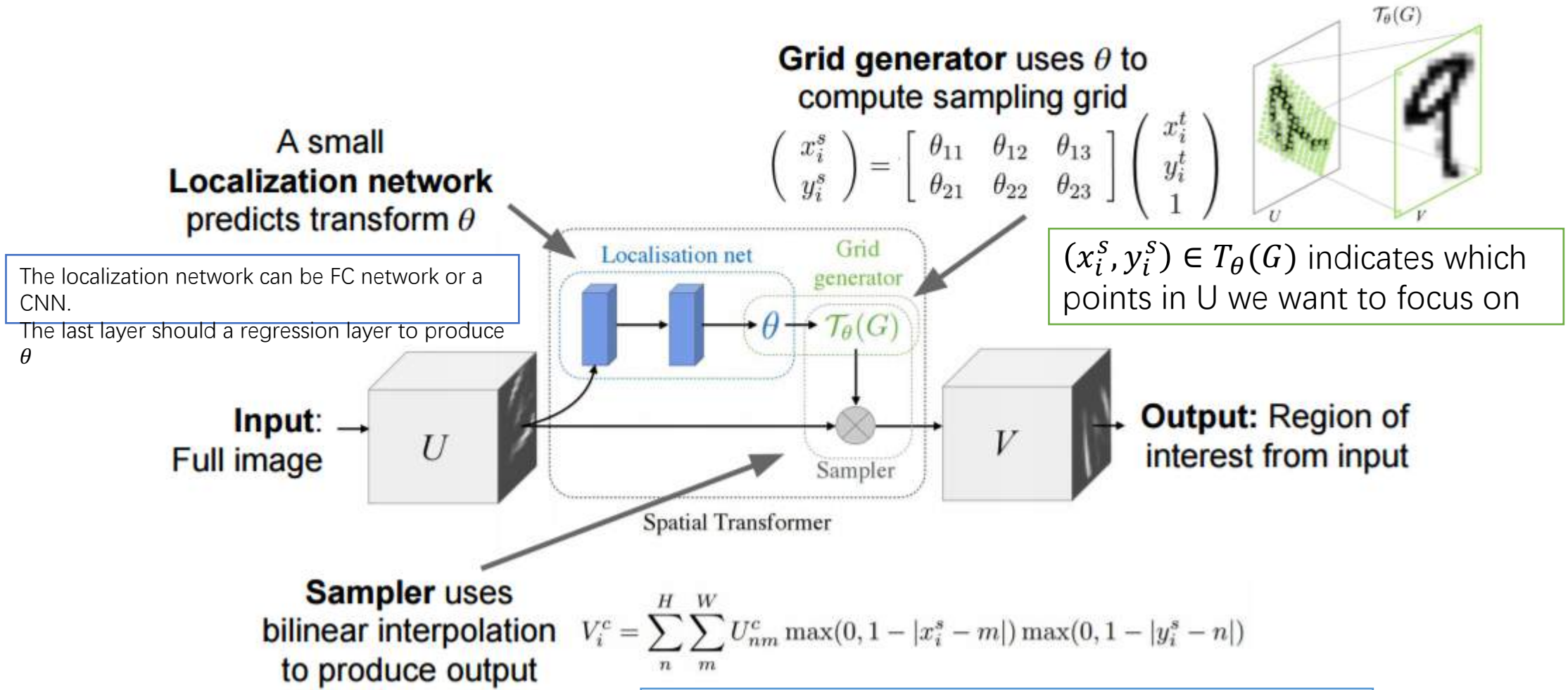



Repeat for all pixels in *output* to get a **sampling grid**

Affine transformation.
But it can be a more general transform

- A module can be inserted to any place of a network
  - Used several times in later deepmind papers






The localization network can be FC network or a CNN.
The last layer should a regression layer to produce $\theta$

$(x_i^s, y_i^s) \in T_\theta(G)$ indicates which points in U we want to focus on

Output V is determined by input U and sampling points $(x_i^s, y_i^s) \in T_\theta(G)$

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

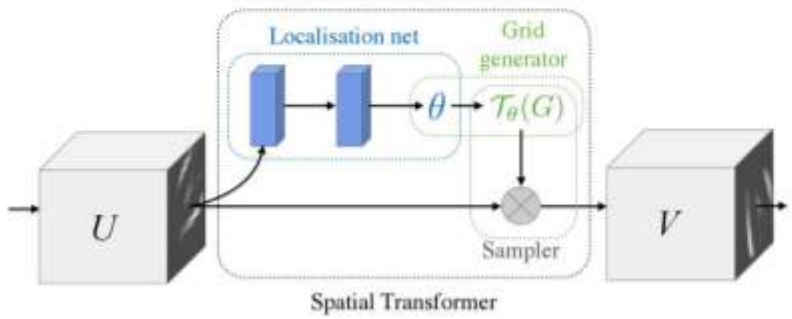
$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

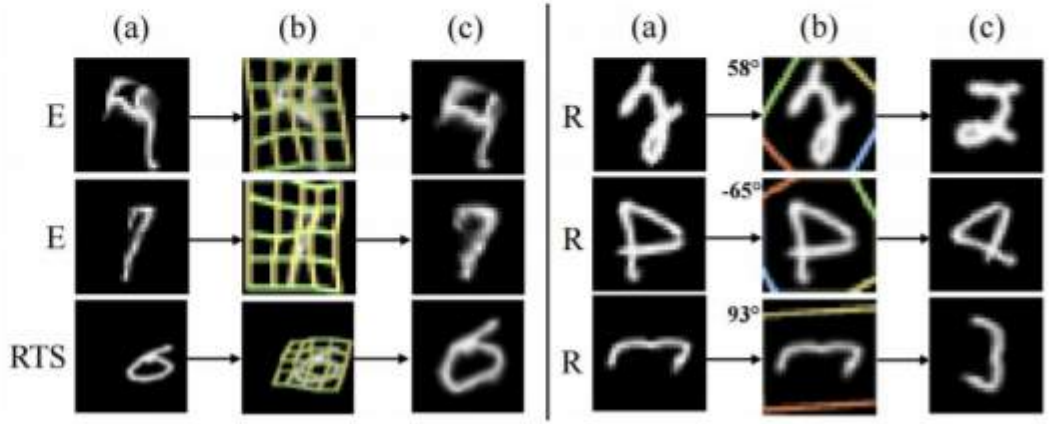
$\mathcal{T}_\theta = M_\theta B$, where $B$ is a target grid representation

- We can even learn the target grid B (using "thin plate spline") (again through backprop)

Differentiable "attention / transformation" module




Insert spatial transformers into a classification network and it learns to attend and transform the input



# Resources

Resources for attentions and NTMs

- https://distill.pub/2016/augmented-rnns/

- A good read : http://ruder.io/deep-learning-nlp-best-practices/

(lot of notes and tricks for deep learning in NLP)

- Google cache: http://webcache.googleusercontent.com/search?q=cache:t_VgmDEvBo8J:ruder.io/deep-learning-nlp-best-practices/+&cd=3&hl=en&ct=clnk&gl=us
- https://distill.pub/2016/augmented-rnns/#attentional-interfaces

(with very nice animation)

https://distill.pub/ (strongly recommend!)

# Thanks

Some materials are taken from https://blog.heuritech.com/2016/01/20/attention-mechanism/
https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/#.WtstO3puaUl