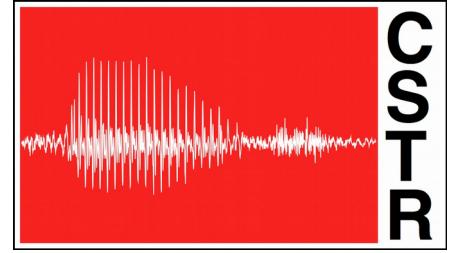




THE UNIVERSITY *of* EDINBURGH
informatics



Attention is All You Need

Erfan Loweimi

Centre for Speech Technology Research (CSTR)
The University of Edinburgh

Attention Is All You Need

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez* †

University of Toronto

aidan@cs.toronto.edu

Łukasz Kaiser*

Google Brain

lukaszkaiser@google.com

Illia Polosukhin* ‡

illia.polosukhin@gmail.com

NIPS 2017

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.



Equal contribution



Attention Is All You Need

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez* †

University of Toronto

aidan@cs.toronto.edu

Łukasz Kaiser*

Google Brain

lukaszkaiser@google.com

Ilia Polosukhin* ‡

illia.polosukhin@gmail.com



NIPS 2017

Attention is all you need



Videos

Images

News

Books

More

Settings

Tools

About 2,010,000,000 results (0.52 seconds)

Attention Is All You Need

<https://arxiv.org/> > cs ▾

9/11/2019

by A Vaswani - 2017 - Cited by 4209 - Related articles

12 Jun 2017 - Attention Is All You Need. The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism.

Equal contribution



Outline

- Seq2Seq modelling via RNN Encoder-Decoder
- Attention Mechanism
- Self-Attention
- Transformer



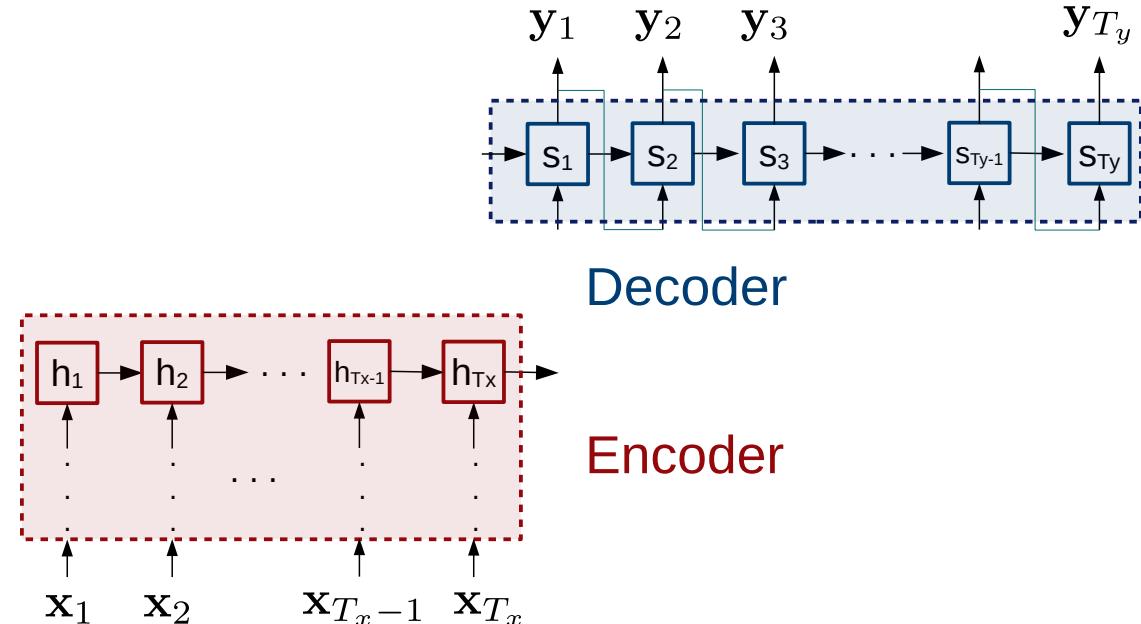
Sequence-to-Sequence Modelling

- Many-to-Many mapping

$$p(Y_1, Y_2, \dots, Y_{T_y} | X_1, X_2, \dots, X_{T_x}) = p(Y_1^{T_y} | X_1^{T_x})$$

Sequence-to-Sequence Modelling

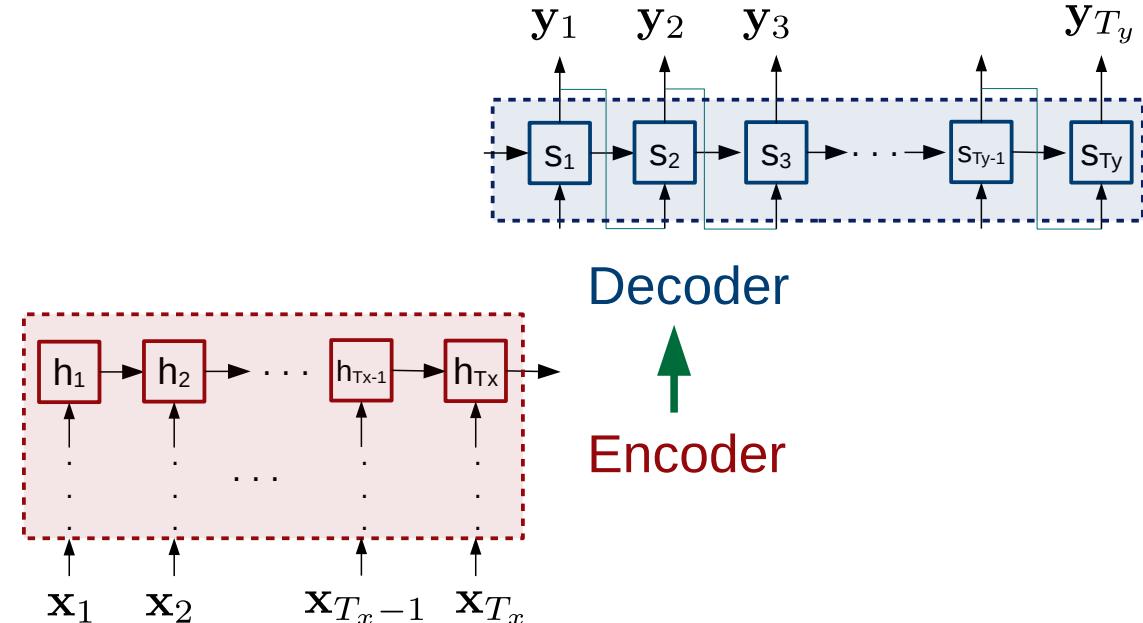
- Many-to-Many mapping



$$p(Y_1, Y_2, \dots, Y_{T_y} | X_1, X_2, \dots, X_{T_x}) = p(Y_1^{T_y} | X_1^{T_x})$$

Sequence-to-Sequence Modelling

- Many-to-Many mapping
- Some approximation & conditioning required!



$$p(Y_1, Y_2, \dots, Y_{T_y} | X_1, X_2, \dots, X_{T_x}) = p(Y_1^{T_y} | X_1^{T_x})$$

RNN Encoder-Decoder

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho

Bart van Merriënboer Caglar Gulcehre

Université de Montréal

firstname.lastname@umontreal.ca

Dzmitry Bahdanau

Jacobs University, Germany

d.bahdanau@jacobs-university.de

Fethi Bougares Holger Schwenk

Université du Maine, France

firstname.lastname@lium.univ-lemans.fr

Yoshua Bengio

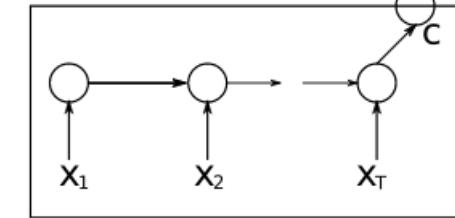
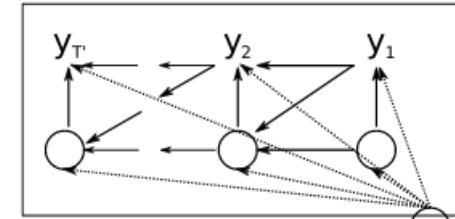
Université de Montréal, CIFAR Senior Fellow

find.me@on.the.web

$$p(Y_i|Y_1, \dots Y_{i-1}, X_1^{T_x}) \approx g(Y_{i-1}, s_i, c)$$

$$s_i = f(y_{i-1}, s_{i-1}, c)$$

Decoder



Encoder

[Learning Phrase Representations using RNN Encoder ...](#)

<https://arxiv.org/> > cs ▾

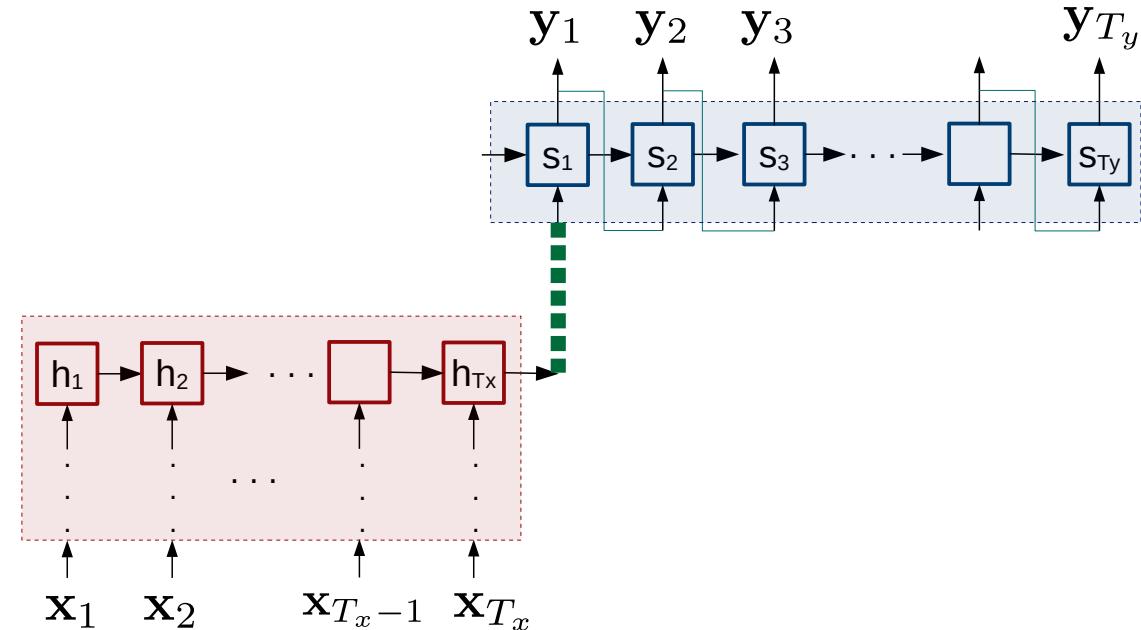
by K Cho - 2014 - Cited by 6907 - Related articles

9/10/2019

3 Jun 2014 - One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of the proposed model are jointly trained to maximize the conditional probability of a target sequence given a source sequence.

RNN Encoder-Decoder

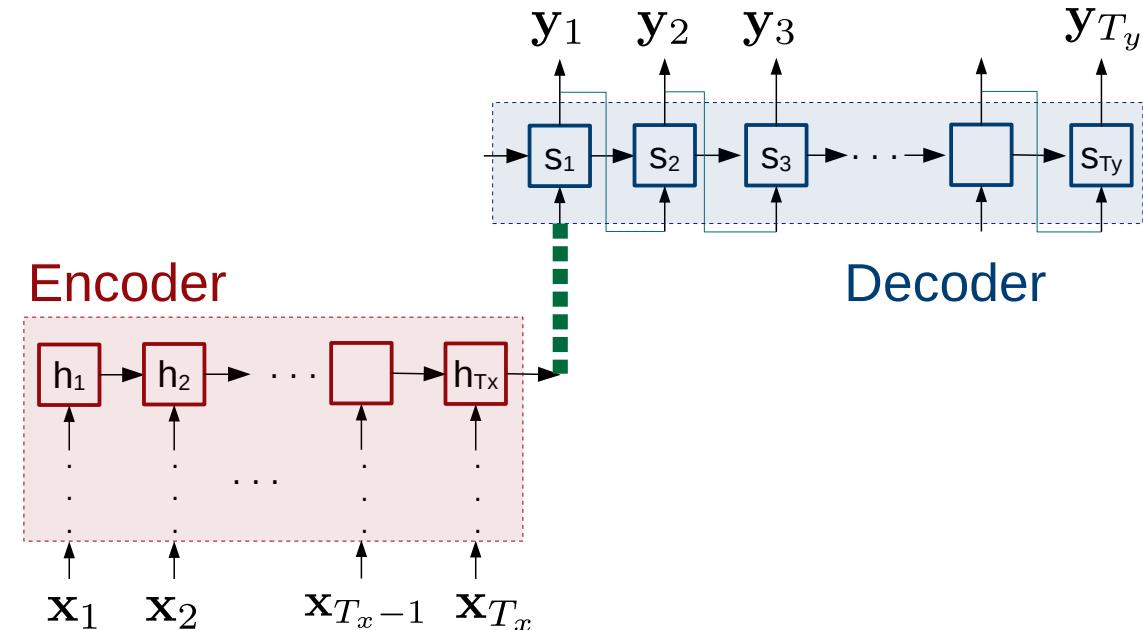
- Break the Many-to-Many into
 - **Many-to-One**
 - **One-to-Many**



$$p(Y_i | Y_1^{i-1}, X_1^{T_x}) \approx g(Y_{i-1}, s_i, c)$$

RNN Encoder-Decoder

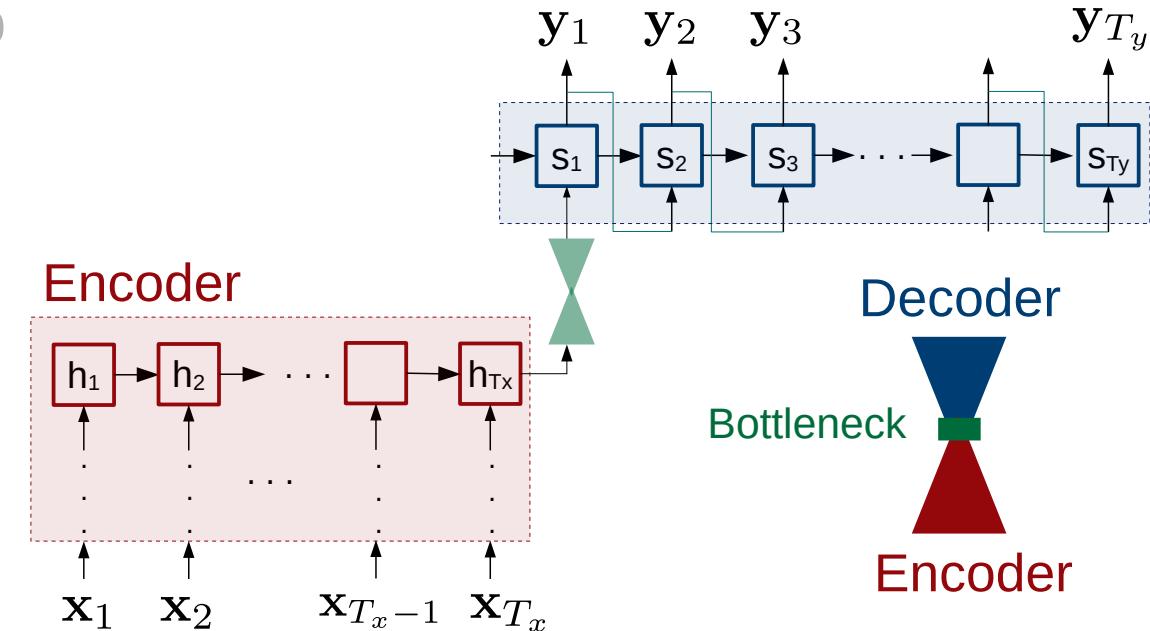
- Break the Many-to-Many into
 - Many-to-**One** \leftrightarrow Encoder
 - **One**-to-Many \leftrightarrow Decoder



$$p(Y_i | Y_1^{i-1}, X_1^{T_x}) \approx g(Y_{i-1}, s_i, c)$$

RNN Encoder-Decoder

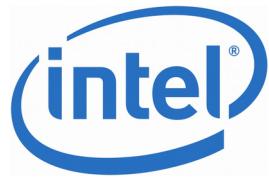
- Break the Many-to-Many into
 - Many-to-One \leftrightarrow Encoder
 - One-to-Many \leftrightarrow Decoder
- “One” \rightarrow **Bottleneck**
 - Context/thought vector
 - Fixed-length representation
 - Combines all info
 - Local/Global/Dependencies



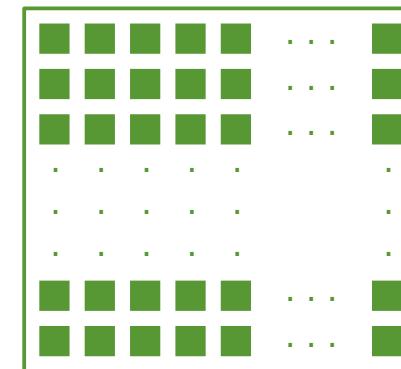
$$p(Y_i | Y_1^{i-1}, X_1^{T_x}) \approx g(Y_{i-1}, s_i, c)$$

RNN Encoder-Decoder Problems (1)

- Sequential computation is hard to parallelise
 - Not what modern HPCs excels at!



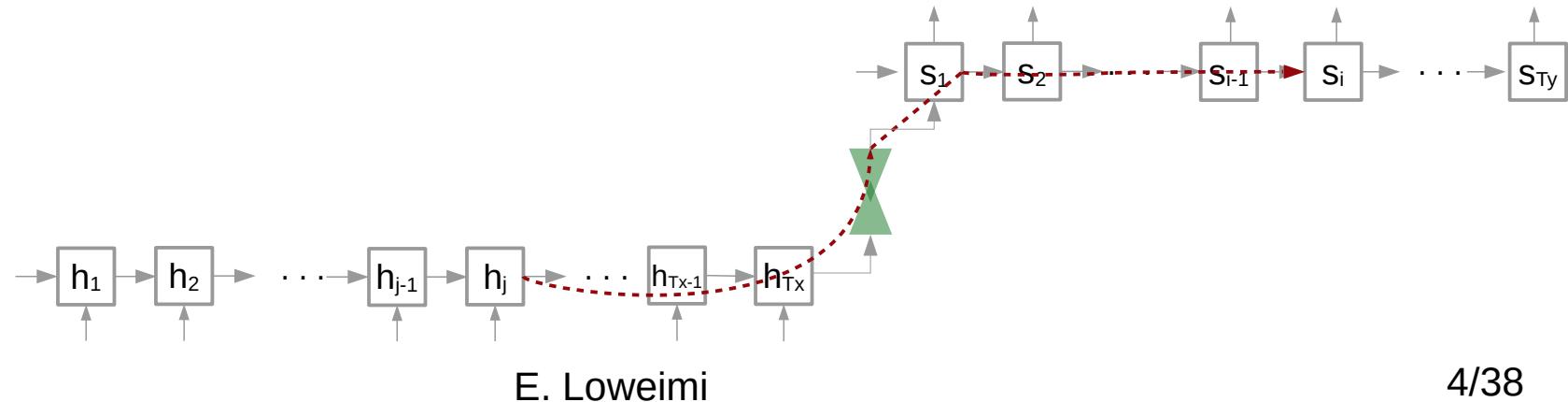
CPUs → Multiple Cores



GPUs → Hundreds to
Thousands Cores

RNN Encoder-Decoder Problems (2)

- Information flow
 - Combination of all info in a single **embedding**
 - Info path between En and De states is long
 - Capturing Long-term dependencies is tricky





Possible Solutions/Alternatives

- Attention mechanism
- Transformer
- CNNs for sequence modelling
 - Appendix 1





Attention Mechanism

Published as a conference paper at ICLR 2015

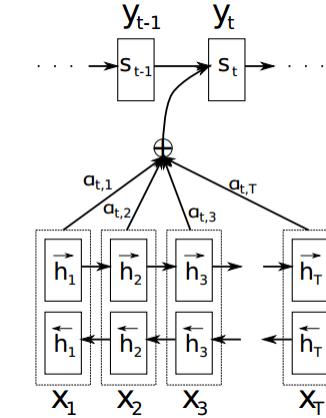
NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio*
Université de Montréal

$$p(Y_i|Y_1, \dots Y_{i-1}, X_1^{T_x}) \approx g(Y_{i-1}, s_i, c_i)$$

$$s_i = f(y_{i-1}, s_{i-1}, c_i)$$



[Neural Machine Translation by Jointly Learning to Align and ...](#)

[https://arxiv.org > cs](https://arxiv.org/cs)

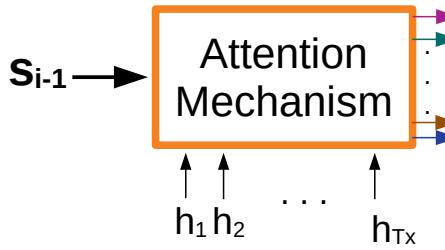
by D Bahdanau - 2014 - Cited by 9235 - Related articles

9/11/2019

Neural Machine Translation by Jointly Learning to Align and Translate. ... Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance.

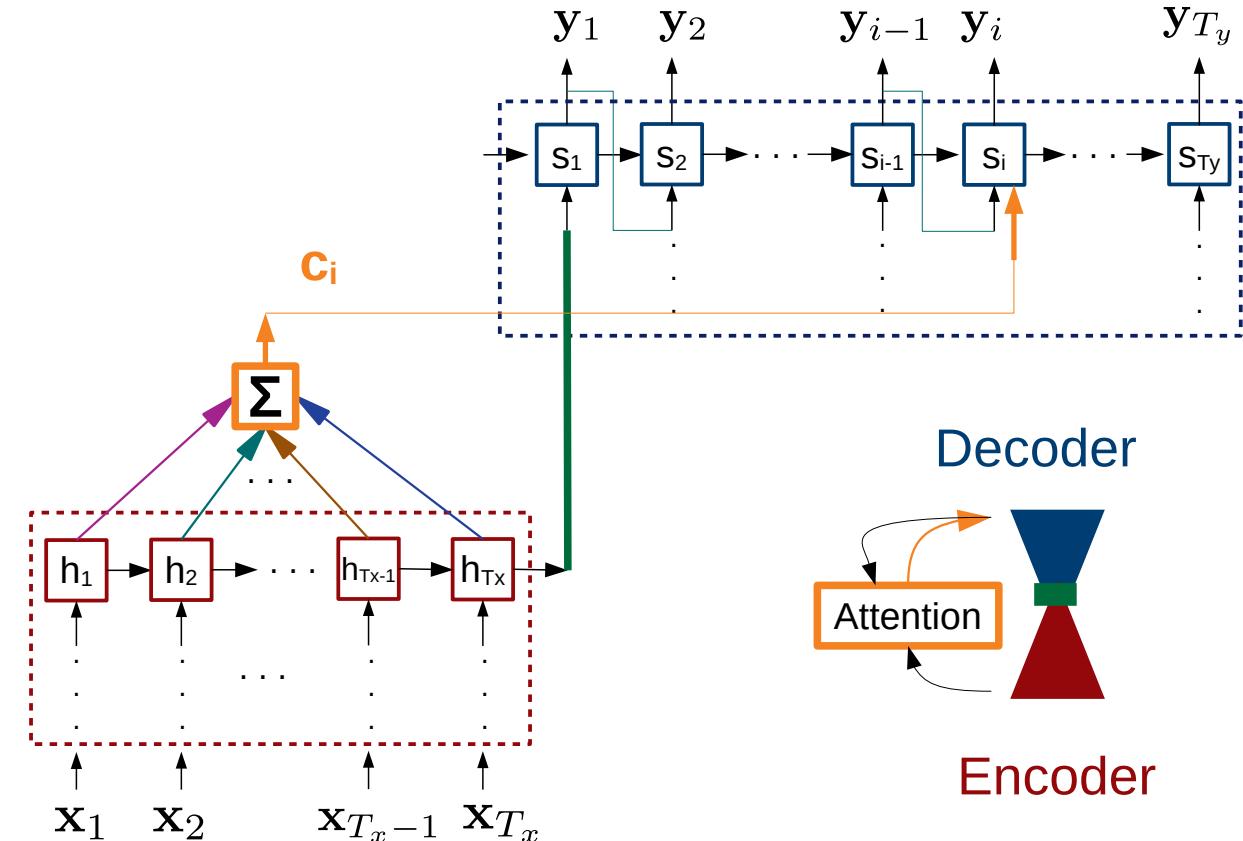


Attention Mechanism



$$p(Y_i|Y_1^{i-1}, X_1^{T_x}) \approx g(Y_{i-1}, s_i, c_i)$$

$$s_i = f(y_{i-1}, s_{i-1}, c_i)$$

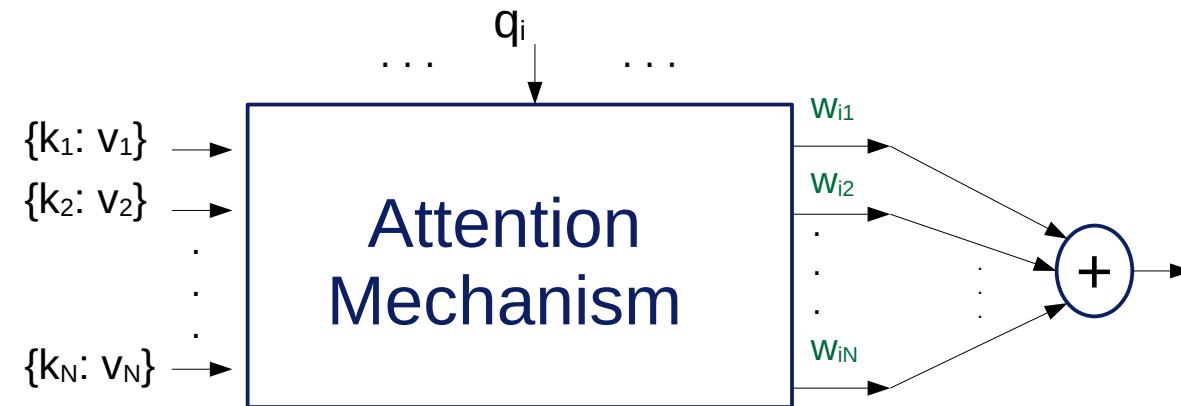


Attention Mechanism

- Attention is a focus mechanism on task-important parts of input
- Given \leftarrow A set of **key-value pairs**, and queries
- Output \rightarrow A **weighted mean** of the values

Attention Mechanism

- Attention is a focus mechanism on task-important parts of input



{key: value}

E. Loweimi

Attention Mechanism

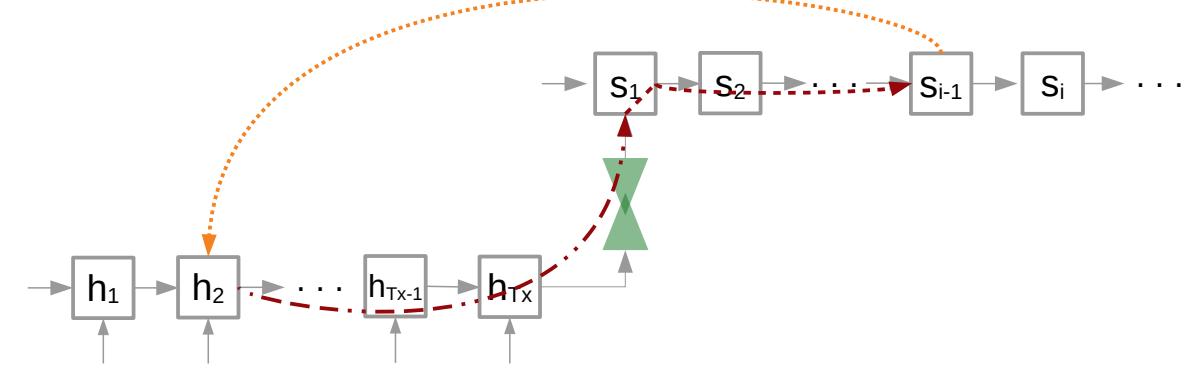
- Attention is a focus mechanism on task-important parts of input
- Given \leftarrow A set of key-value pairs, and queries
- Output \rightarrow A **weighted mean** of the values
- **Weights** \rightarrow prop. to similarity of the query & keys

Attention Mechanism

- Query (Q)
 - Determines where focus should be steered
- Keys (K) and Values (V) pairs, {k:v}
 - Some prototypes
- In RNN En-De models
 - $Q \rightarrow$ Decoder states (\mathbf{s}_{i-1})
 - K and $V \rightarrow$ Encoder states ($\mathbf{h}_{1:T_x}$) \rightarrow Identical HERE!

Attention Advantages

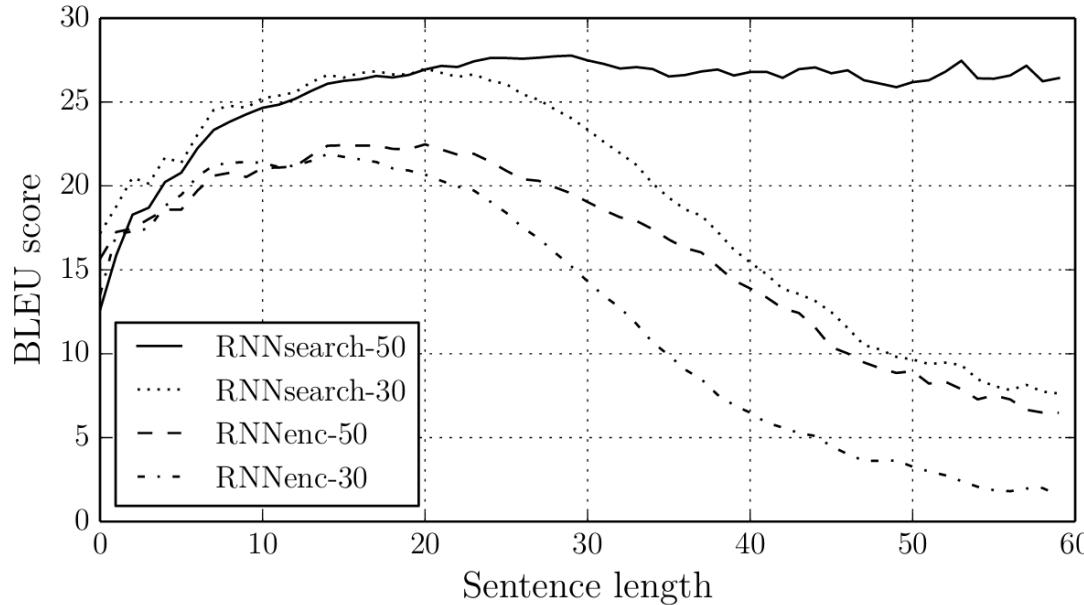
- Solves the info **bottleneck** issue of RNN En-De
- Shorten info path between $h_{1:T_x}$ and each s_i
 - Long-range dependencies better captured/modelled



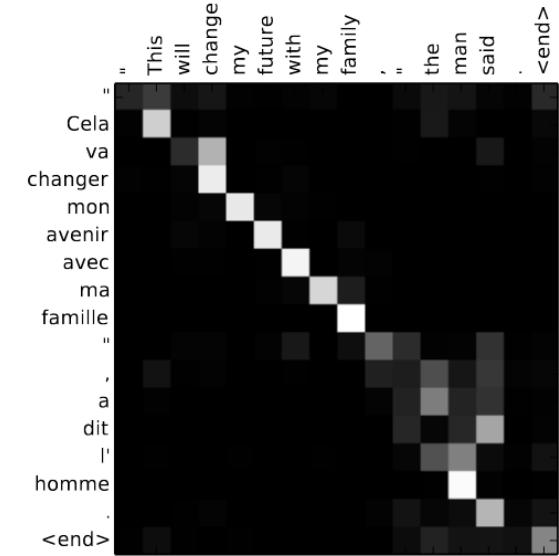
Attention Advantages

- Solves the info bottleneck issue of RNN En-De
- Shorten info path between $\mathbf{h}_{1:T_x}$ and each \mathbf{s}_i
- Helps with gradient vanishing
- Jointly learns alignment & classification
 - Visualisation and understanding

Attention Advantages in NMT



Dealing with long-range dependencies



Visualisation of alignment

Attention Model

- Score (e_{ij})
- Alignment (α_{ij})
- Context (c_{ij})
 - Aka *glimpse* (g_i)
- RNN Decoder

$$\begin{aligned}\alpha_{ij} &= \text{Attention}(\mathbf{s}_{i-1}, \mathbf{h}_j) \\ &= \text{softmax}(e_{ij})\end{aligned}$$

$$\mathbf{c}_i = \alpha_i^T \mathbf{h}_j = \sum_j \alpha_{ij} \mathbf{h}_j$$

$$\mathbf{y}_i \sim \text{Label-Distribution}(\mathbf{y}_{i-1}, \mathbf{s}_i, \mathbf{c}_i)$$

Alignment model → Compute Score

- Dot-product
 - Basic
 - Linear projection

$$e_{ij} = \mathbf{s}_i^T \mathbf{h}_j$$

$$e_{ij} = \mathbf{s}_i^T W \mathbf{h}_j$$

Alignment model → Compute Score

- Dot-product
 - Basic
 - Linear projection
- Additive
 - Content
 - Location

$$e_{ij} = \mathbf{s}_i^T \mathbf{h}_j$$

$$e_{ij} = \mathbf{s}_i^T W \mathbf{h}_j$$

$$e_{ij} = v^T \tanh(W\mathbf{s}_{i-1} + V\mathbf{h}_j + \mathbf{b})$$

$$e_{ij} = v^T \tanh(W\mathbf{s}_{i-1} + V\mathbf{h}_j + U f_{ij} + \mathbf{b})$$

Sharpening the Focus / Attention

- Use *inverse temperature*, β ,

- $\beta > 1 \Rightarrow$ sharpening the pdf
 - $\beta < 1 \Rightarrow$ smoothing the pdf

$$a_{ij} = \frac{\exp(\beta e_{ij})}{\sum_{j'} \exp(\beta e_{ij'})}$$

- Top-k

- Keep top k values of $e_i \rightarrow$ Set the rest to zero \rightarrow Normalise

- Caveat: requires computing all e_{ij} s

- Computational complexity $\rightarrow O(T_x T_y)$
 - **Solution:** Windowed attention

Sharpening the Focus / Attention

- **Windowed Attention**
 - Window length: $2w \rightarrow 2w \ll L$
 - Window centre: $p_i \rightarrow$ median of α_{i-1}
 - α_{i-1} shortlists the encoder states (h_j)
 - Compute attention only on $\tilde{h} = (h_{p_i-w}, \dots, h_{p_i+w-1})$
- **Caveats:**
 - Not useful for short utterances, too sharp
 - **Solution** → *Smoothing* → Replace \exp with *sigmoid* in *softmax*
 - Window length and location are suboptimal
 - **Solution** → *Fully-trainable Windowed Attention*

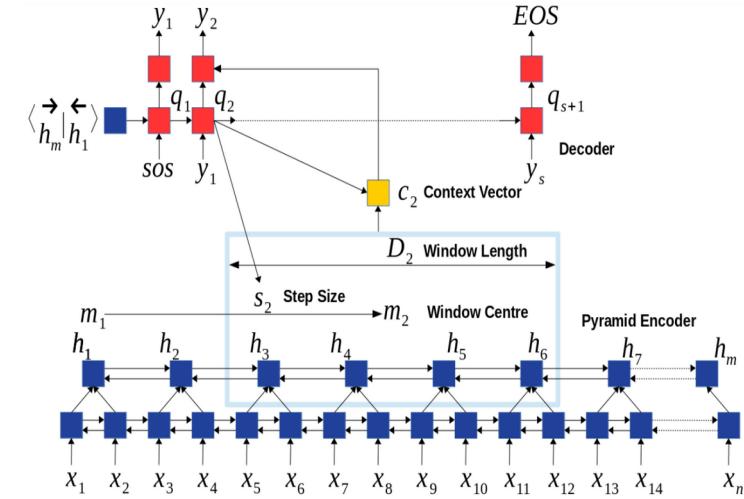


(Fully-Trainable) Windowed Attention

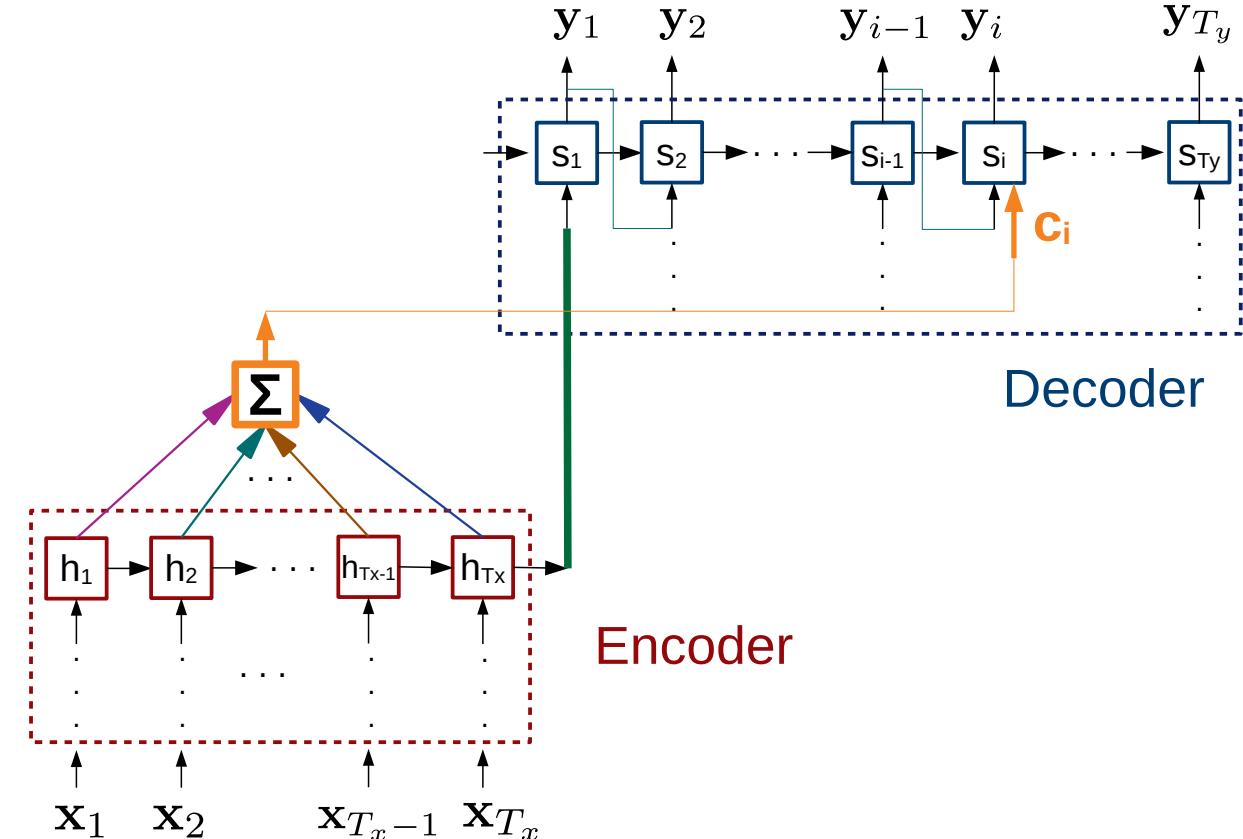
WINDOWED ATTENTION MECHANISMS FOR SPEECH RECOGNITION

Shucong Zhang, Erfan Loweimi, Peter Bell, Steve Renals

Centre for Speech Technology Research, University of Edinburgh, Edinburgh, UK

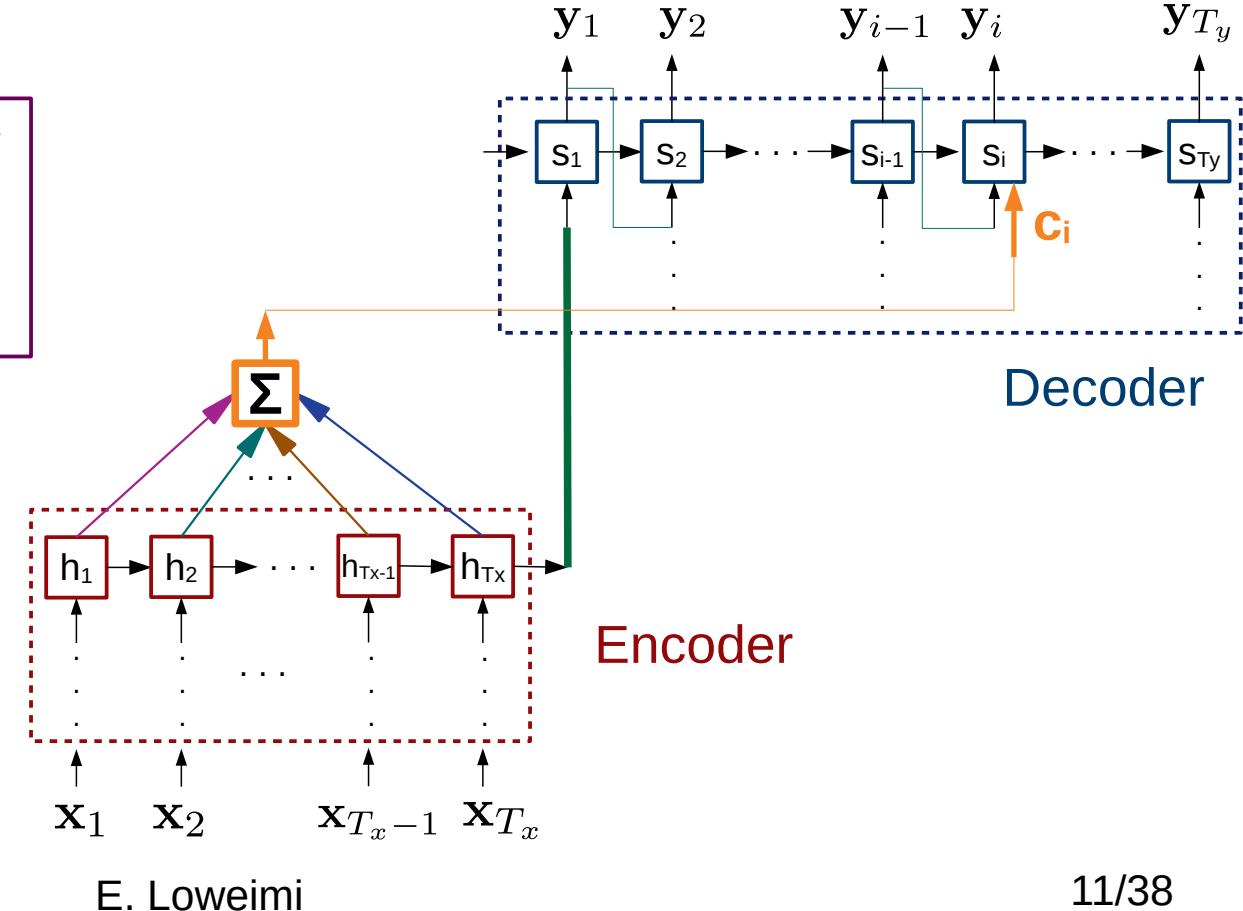


(Fully-Trainable) Windowed Attention



(Fully-Trainable) Windowed Attention

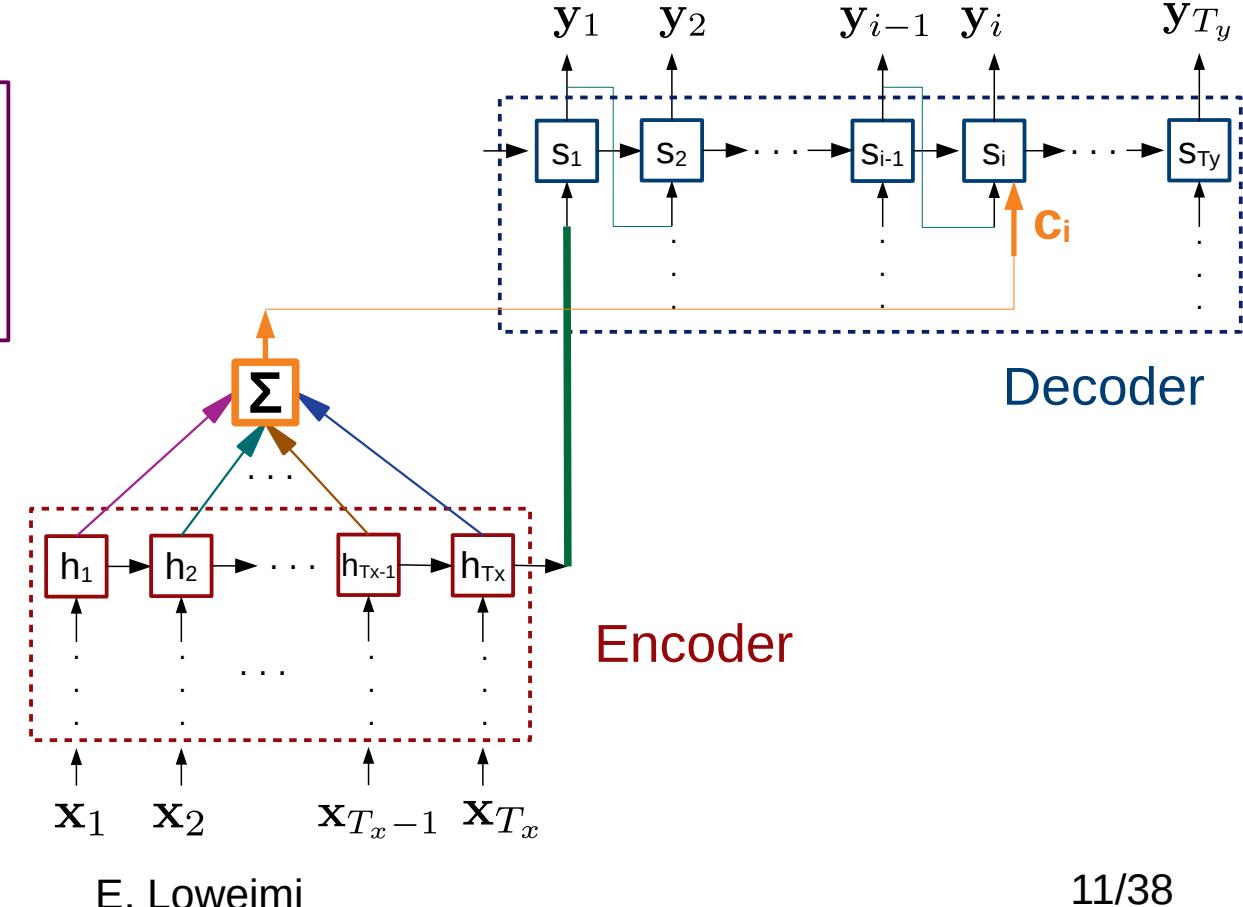
What if \mathbf{X} sequence is very long & y_i is only correlated with a small part of \mathbf{X} ...



(Fully-Trainable) Windowed Attention

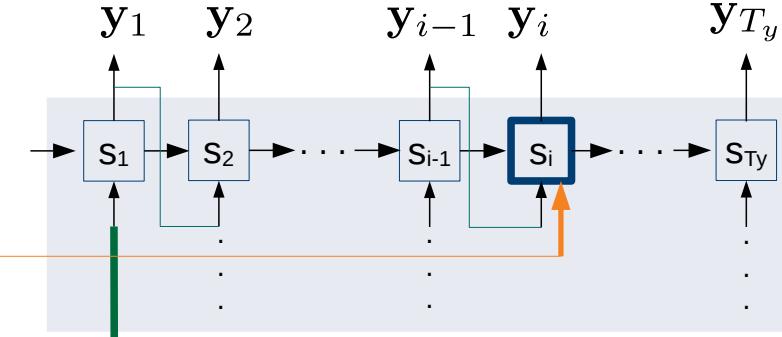
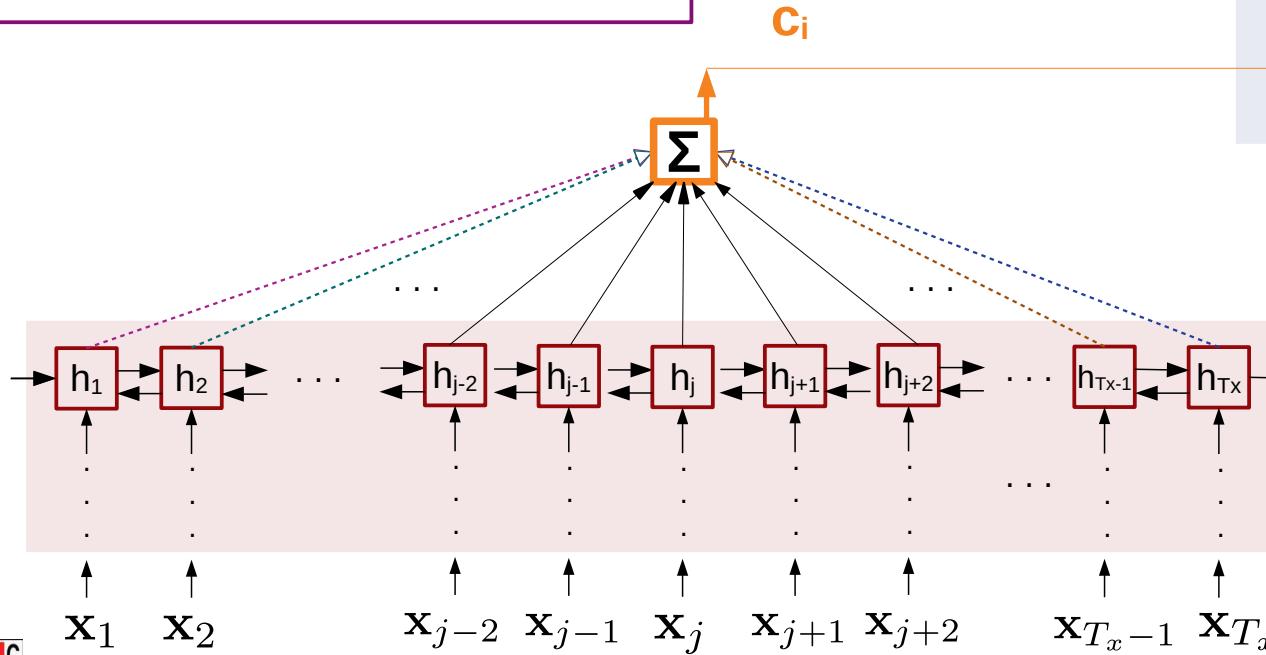
What if \mathbf{X} sequence is very long & \mathbf{y}_i is only correlated with a small part of \mathbf{X} ...

c_i contains noisy info from irrelevant $h_j \rightarrow$ Suboptimal attention



(Fully-Trainable) Windowed Attention

Many α_{ij} are zero. Why/Can
should we learn them?!

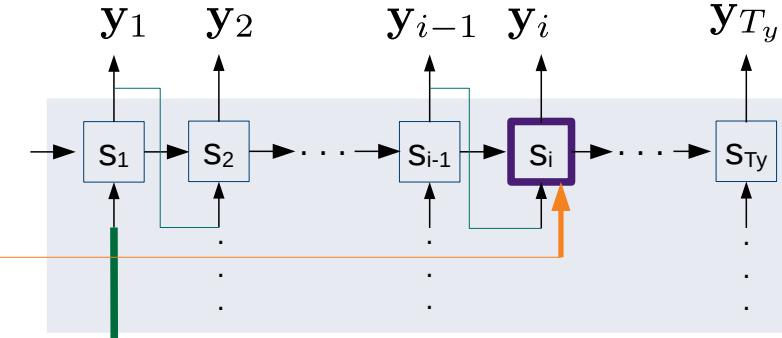
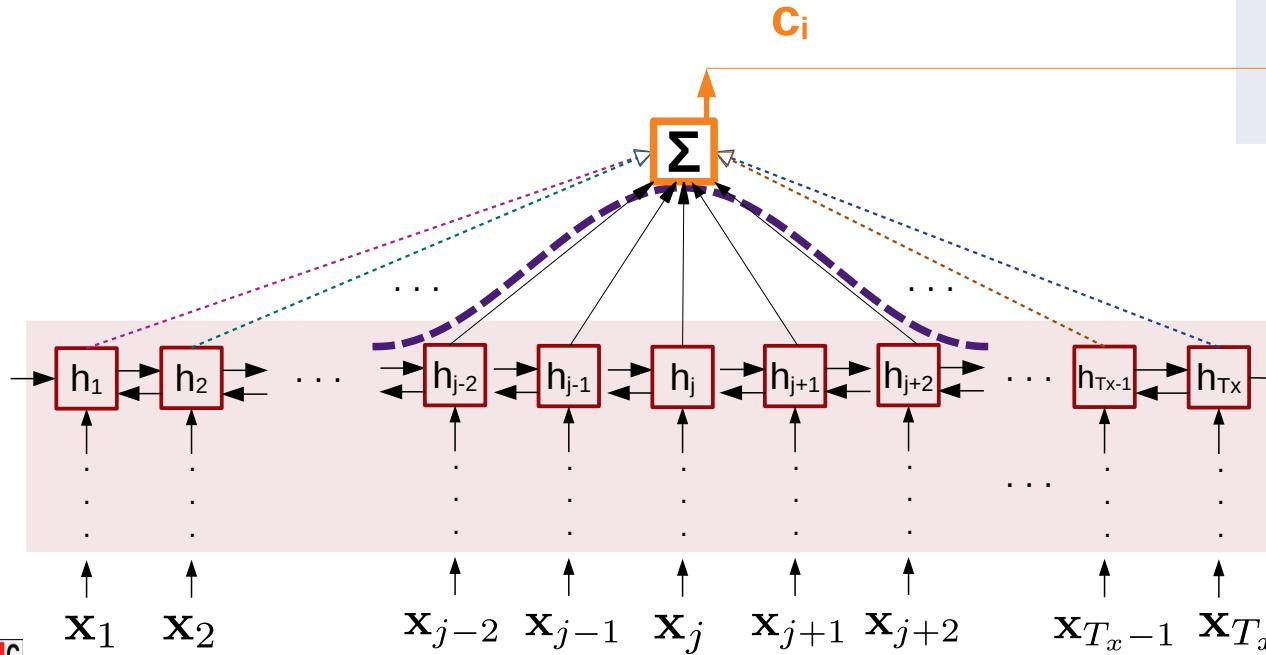


Decoder

Encoder

(Fully-Trainable) Windowed Attention

Impose sparsity by windowing ...



Decoder

Encoder



(Fully-Trainable) Windowed Attention

$$L_i = L_{max} \sigma(MLP(\mathbf{s}_i))$$

$$sh_i = SH_{max} \sigma(MLP(\mathbf{s}_i))$$

$$m_i = m_{i-1} + sh_i$$

Fully-trainable → BOTH window length and window shift are learned.

$$l_{ij} = \begin{cases} \exp(-\frac{(j-m_i)^2}{2(D_{iL}/2)^2}), & j \in (m_i - D_{iL}, m_i) \\ \exp(-\frac{(j-m_i)^2}{2(D_{iR}/2)^2}), & j \in (m_i, m_i - D_{iR}) \end{cases}$$

$$\alpha_{ij} = \frac{\exp(e_{ij}) l_{ij}}{\sum_{m_i-D_{iL}}^{m_i+D_{iR}} \exp(e_{ik}) l_{ik}}$$

Windowed Attention Mechanism for Speech Recognition, Zhang et al, ICASSP 2019



(Fully-Trainable) Windowed Attention

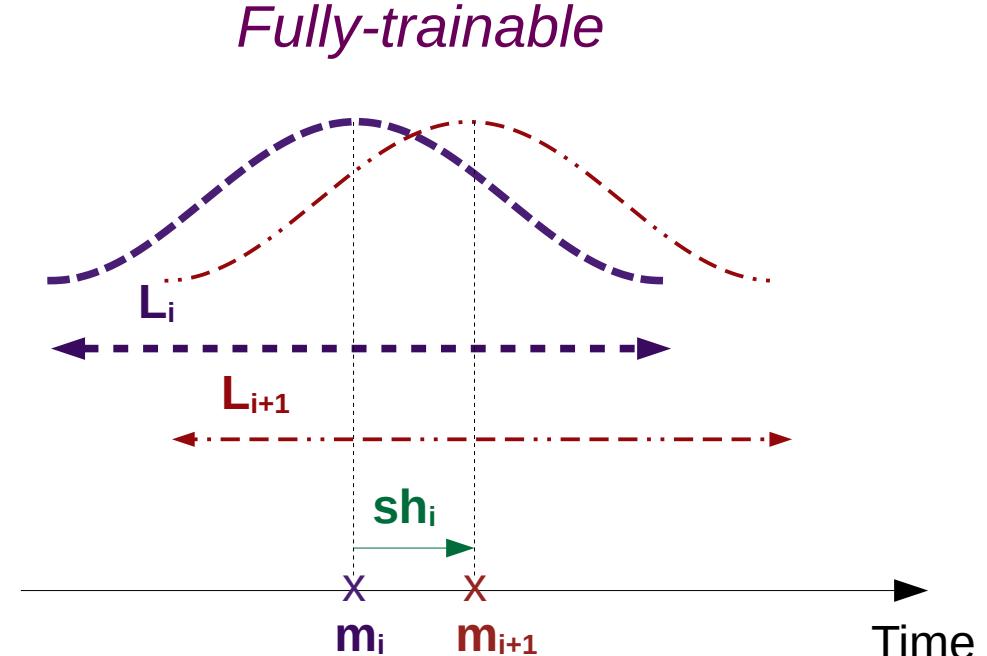
$$L_i = L_{max} \sigma(MLP(\mathbf{s}_i))$$

$$sh_i = SH_{max} \sigma(MLP(\mathbf{s}_i))$$

$$m_i = m_{i-1} + sh_i$$

$$l_{ij} = \begin{cases} \exp(-\frac{(j-m_i)^2}{2(D_{iL}/2)^2}), & j \in (m_i - D_{iL}, m_i) \\ \exp(-\frac{(j-m_i)^2}{2(D_{iR}/2)^2}), & j \in (m_i, m_i - D_{iR}) \end{cases}$$

$$\alpha_{ij} = \frac{\exp(e_{ij}) l_{ij}}{\sum_{m_i-D_{iL}}^{m_i+D_{iR}} \exp(e_{ik}) l_{ik}}$$



Windowed Attention Mechanism for Speech Recognition, Zhang et al, ICASSP 2019

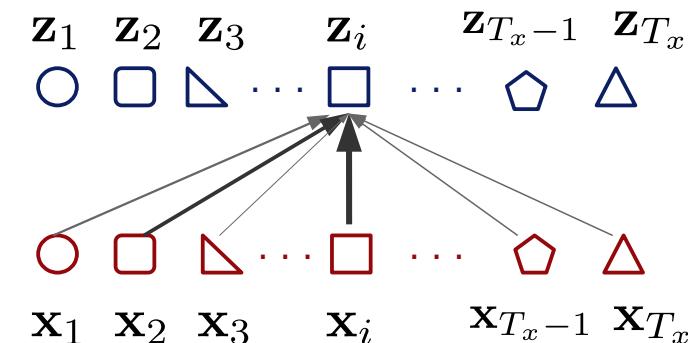
Self-Attention

- An attention within a layer (representation)
 - Encoder \leftrightarrow Classic attention \leftrightarrow Decoder



Self-Attention

- An attention within a layer (representation)
- Each weight is prop. to similarity of two vertices

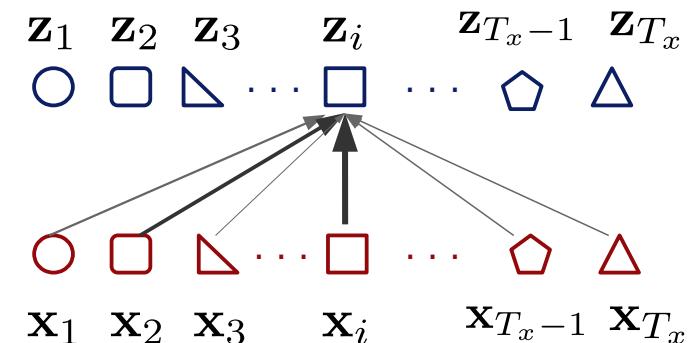


Self-Attention

- An attention within a layer (representation)
- Each weight is prop. to similarity of two vertices

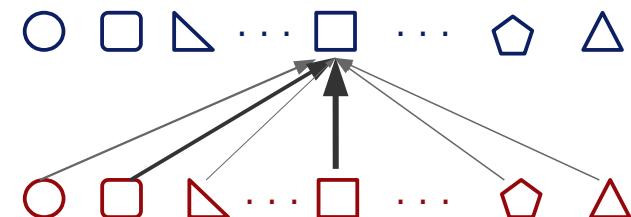
$$\mathbf{z}_i = \sum_j w_{ij} \mathbf{x}_j$$

$$\begin{cases} w_{ij} = \text{similarity}(\mathbf{x}_i, \mathbf{x}_j) \\ w_{ij} \geq 0, \quad \sum_j w_{ij} = 1 \end{cases}$$



Self-Attention Advantages

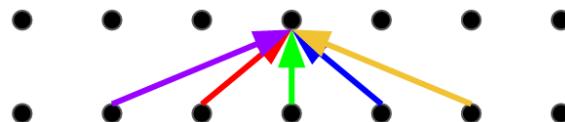
- Constant path length between positions, $O(1)$
 - Direct interaction, no locality bias
- Long-range dependencies are captured well
- Multiplicative interaction → some kind of gating
- Permutation invariant
- Trivial to parallelise



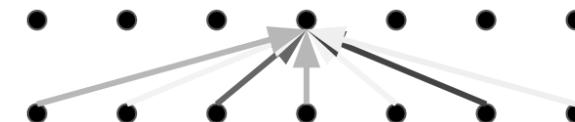
Convolution vs Self-Attention

- CNN
 - Linear Time Invariant
 - Suboptimal filter replication
 - Seq. modelling requires depth
- Self-Attention
 - Linear(?) Time Variant
 - One filter per node
 - Direct interaction for all

Convolution



Self-Attention



Self-Attention Disadvantageous

- Globally, sequentiality is lost
 - has no notion of temporal order!
 - Permutation invariant!
- Locally, temporal resolution is lost
 - Owing to attention-weighted averaging

Self-Attention Disadvantageous

- Globally, **sequentiality is lost**
 - has no notion of temporal order!
 - Permutation invariant!
- Locally, temporal resolution is lost
 - Owing to attention-weighted averaging
- **Solution:** Positional Encoding



Computational Complexity

- Self-attention $\rightarrow O(n^2d)$
 - Quadratic in sequence length (n)
 - Linear in representation dimension (d)
- RNN $\rightarrow O(nd^2)$
 - Linear in seq. length; Quadratic in repr. dim



Computational Complexity

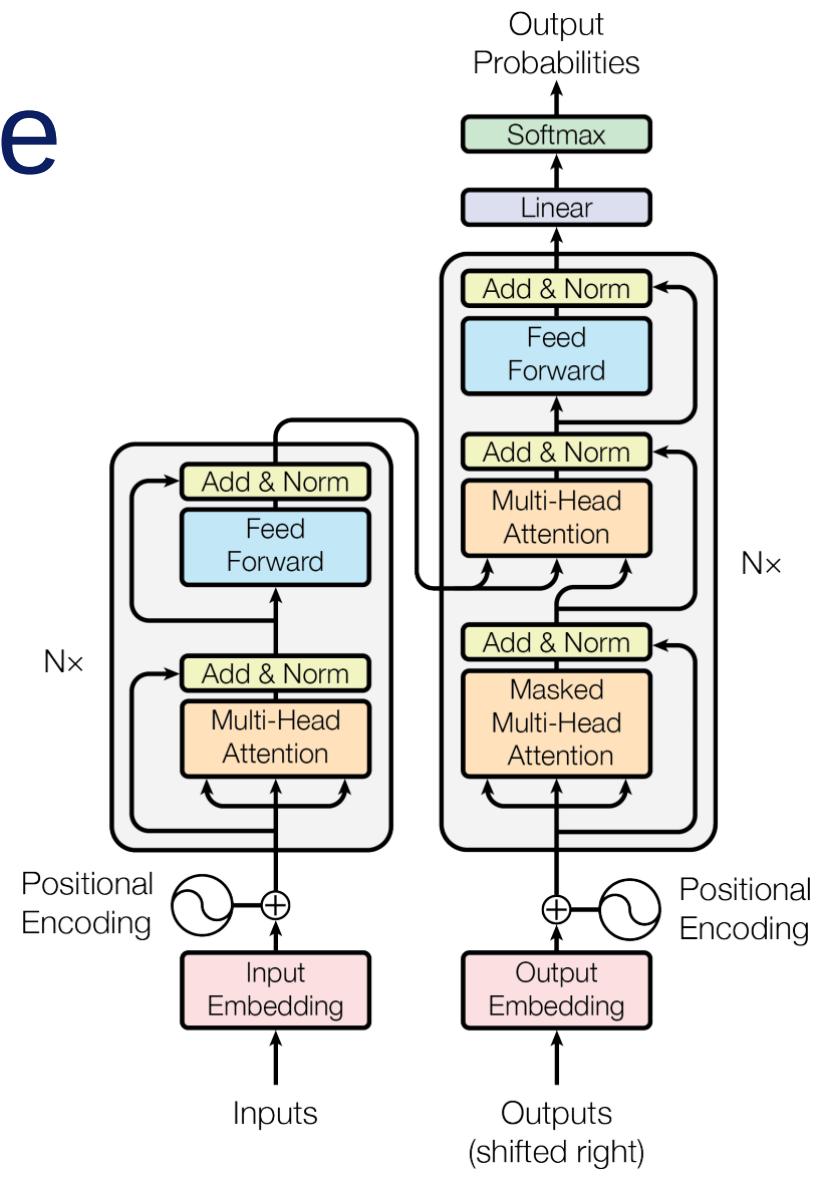
- Self-attention → $O(n^2d)$
 - Quadratic in sequence length (n)
 - Linear in representation dimension (d)
- RNN → $O(nd^2)$
 - Linear in seq. length; Quadratic in repr. dim
- If $n < d$ → Self-attention is more economic, e.g. NMT
- If $n > d$ → Self-attention is parallelisable, e.g. ASR



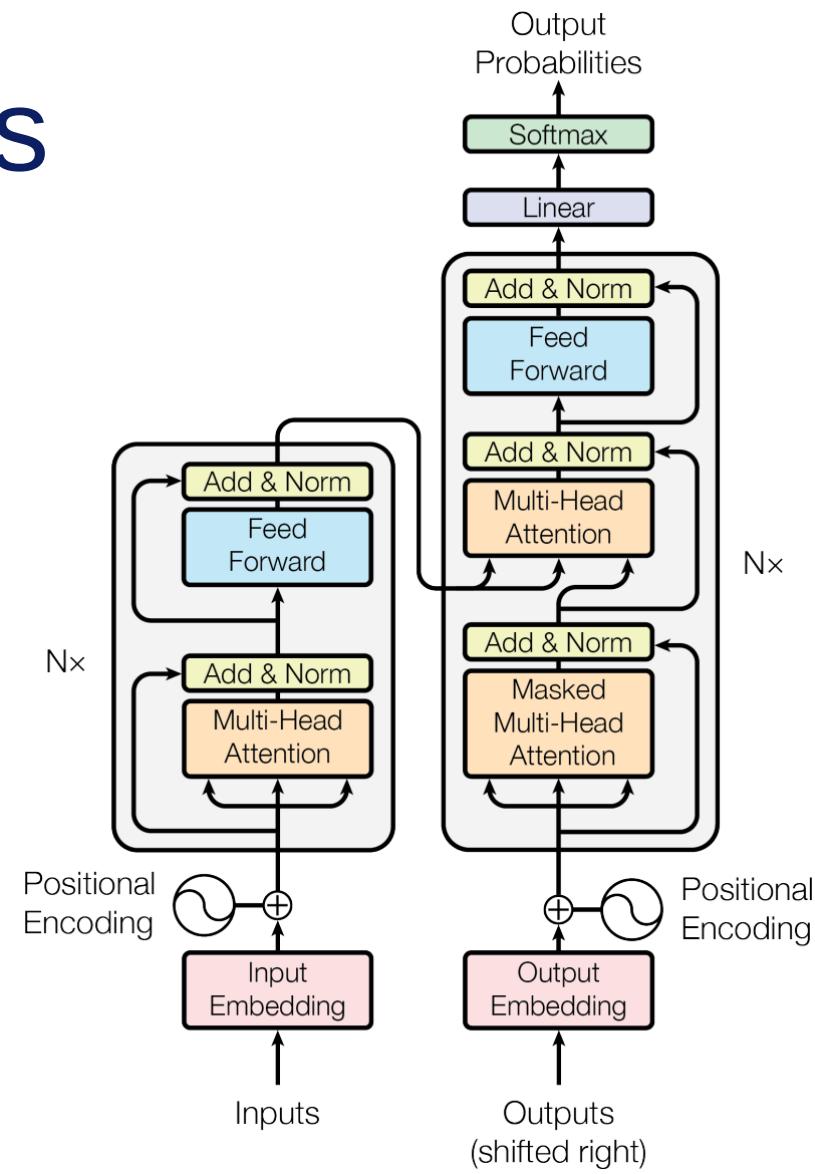
Transformers



Architecture

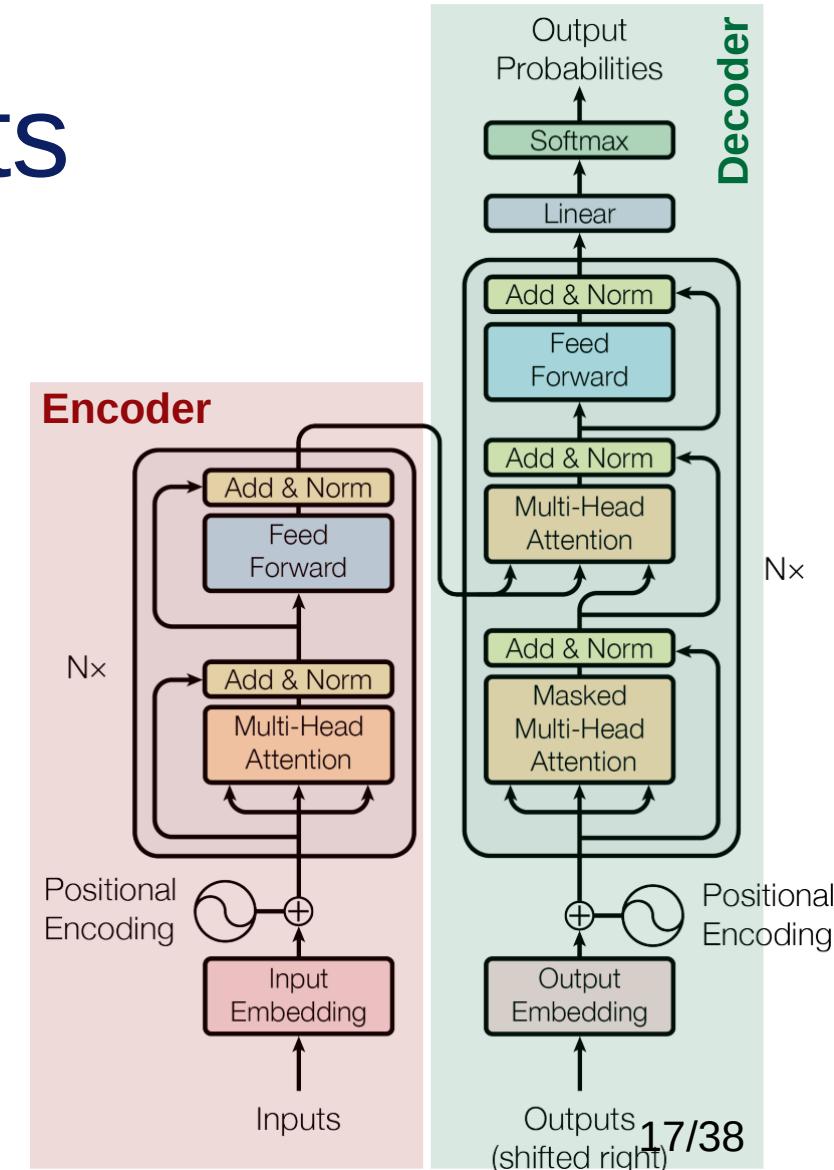


Ingredients



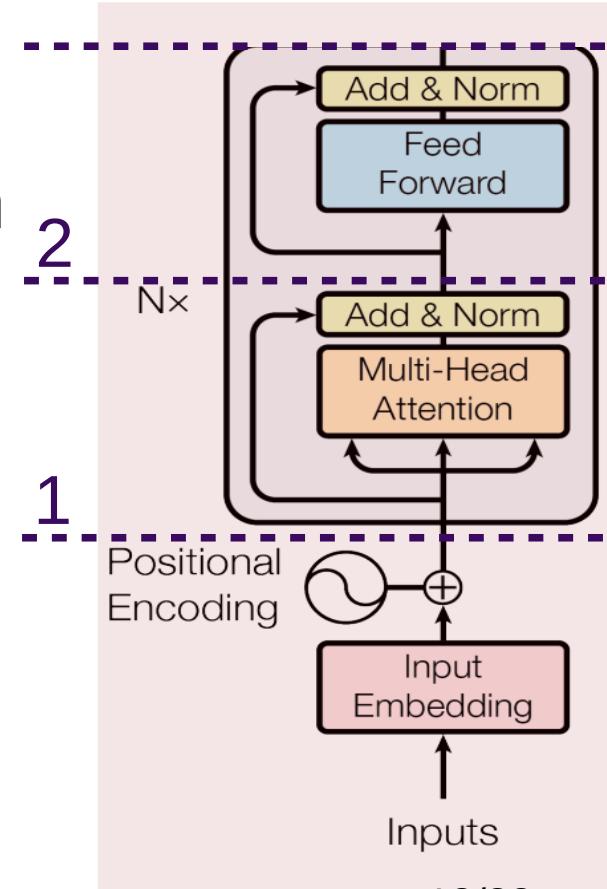
Ingredients

- **Encoder-Decoder** structure
- Positional Embedding
- Multi-Head self-Attention
- Feed Forward NN (FFNN)
- Add & Norm



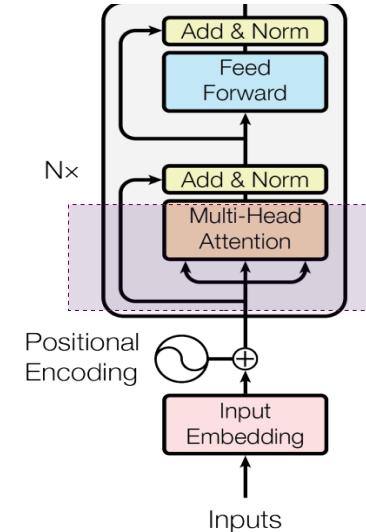
Encoder

- 6 Layers, each one has ...
 - Sublayer 1: Multi-head Self-attention
 - Sublayer 2: (Point-wise) FFNN
- Add & Norm after each sublayer
 - Sublayer = Norm($x + \text{sublayer}(x)$)



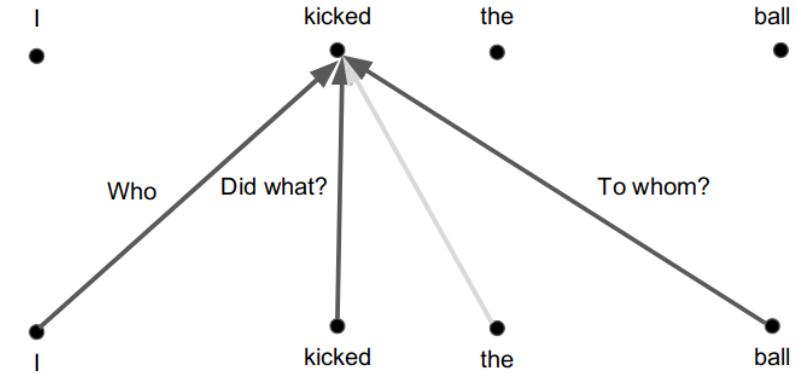
Multi-head Self-attention – Intuition

- Process multiple typesstreams of info or subtasks *independently*



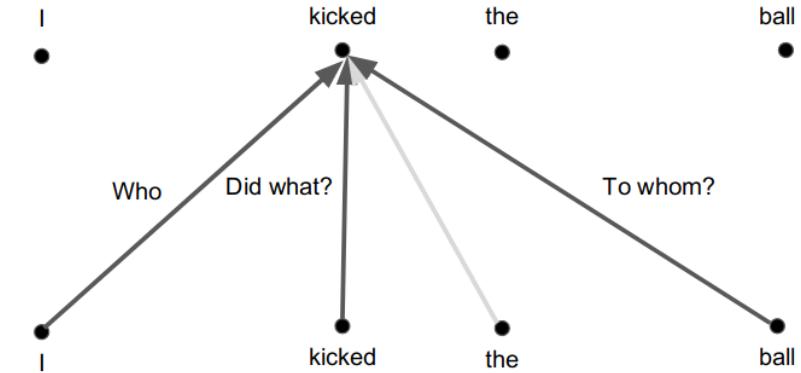
Multi-head Self-attention – Intuition

- Process multiple typesstreams of info or subtasks *independently*, e.g.
 - Who?
 - Did what?
 - To whom?



Multi-head Self-attention – Intuition

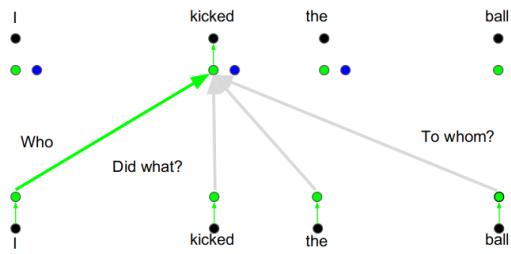
- Process multiple typesstreams of info or subtasks *independently*, e.g.
 - Who?
 - Did what?
 - To whom?



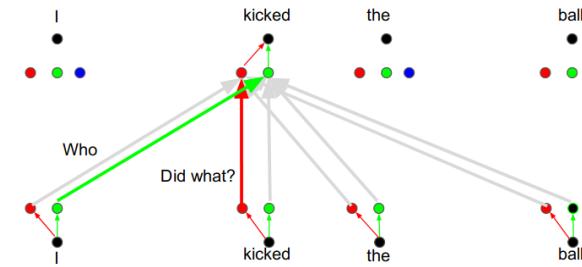
Each subtask and/or piece of info requires a different solution and attention.

Multi-head Self-attention – Intuition

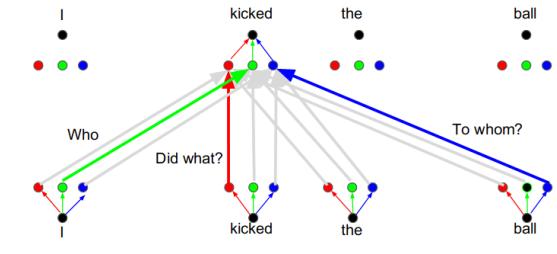
- Process multiple types of info



Head 1: Who?



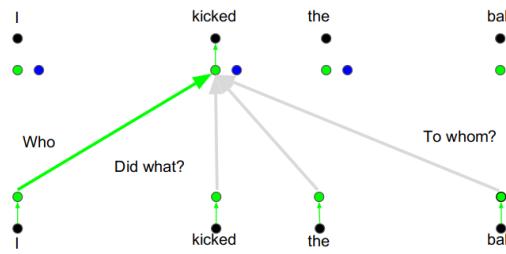
Head 2: Did what?



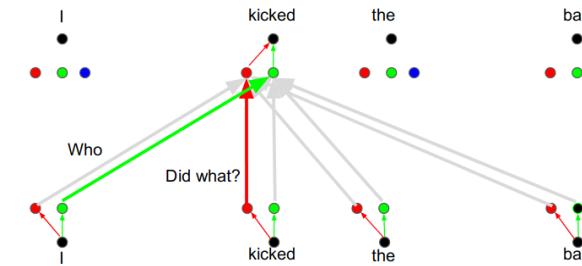
Head 3: To whom?

Multi-head Self-attention – Intuition

- Process multiple types of info

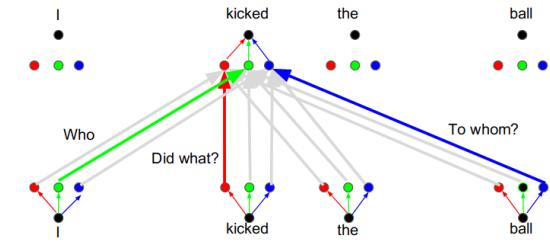


Head 1: Who?



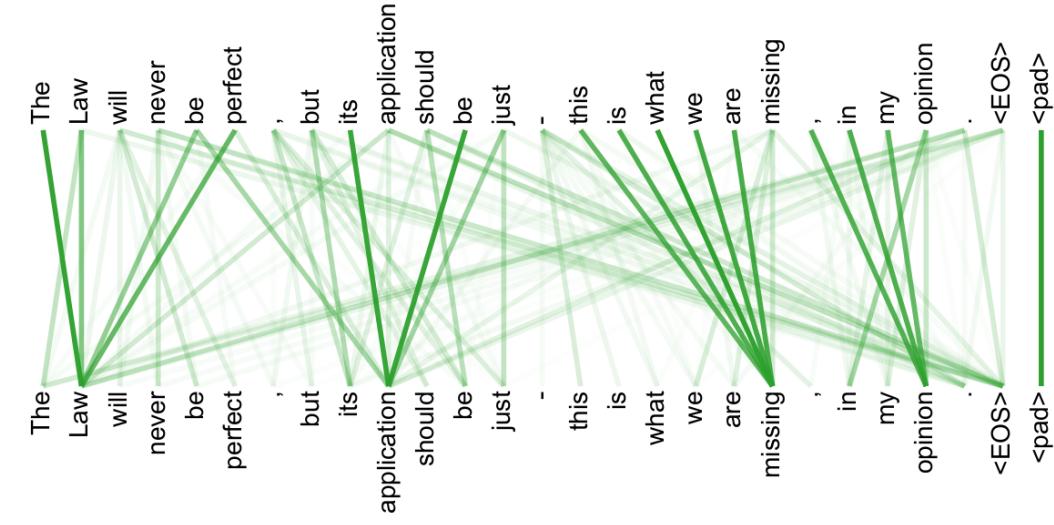
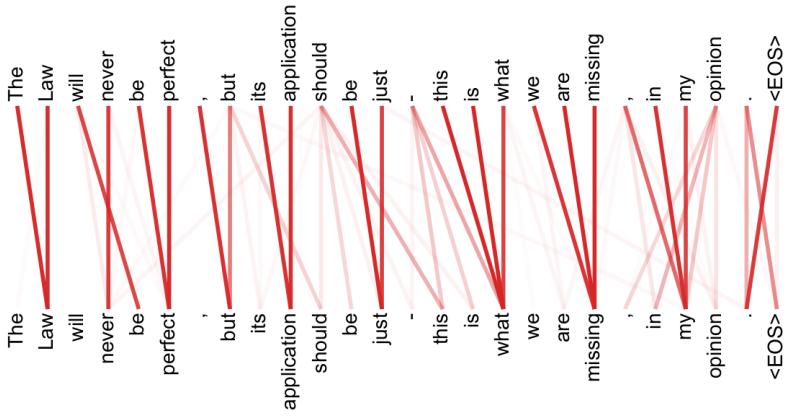
Head 2: Did what?

Parallelisable



Head 3: To whom?

Multi-head Self-attention – Intuition



- Two heads form encoder self-attention at layer 5 (out of 6).
- Heads learn to perform different tasks.

Multi-head self-Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

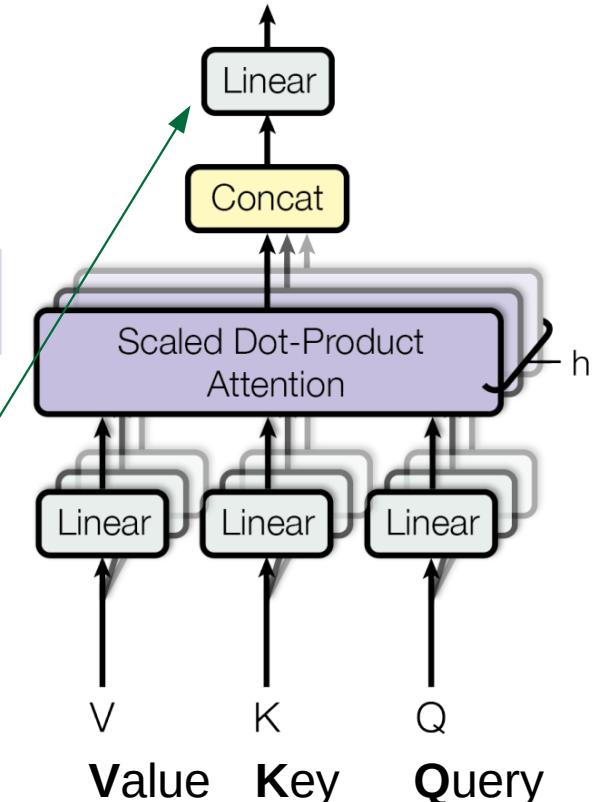
$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

$$W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

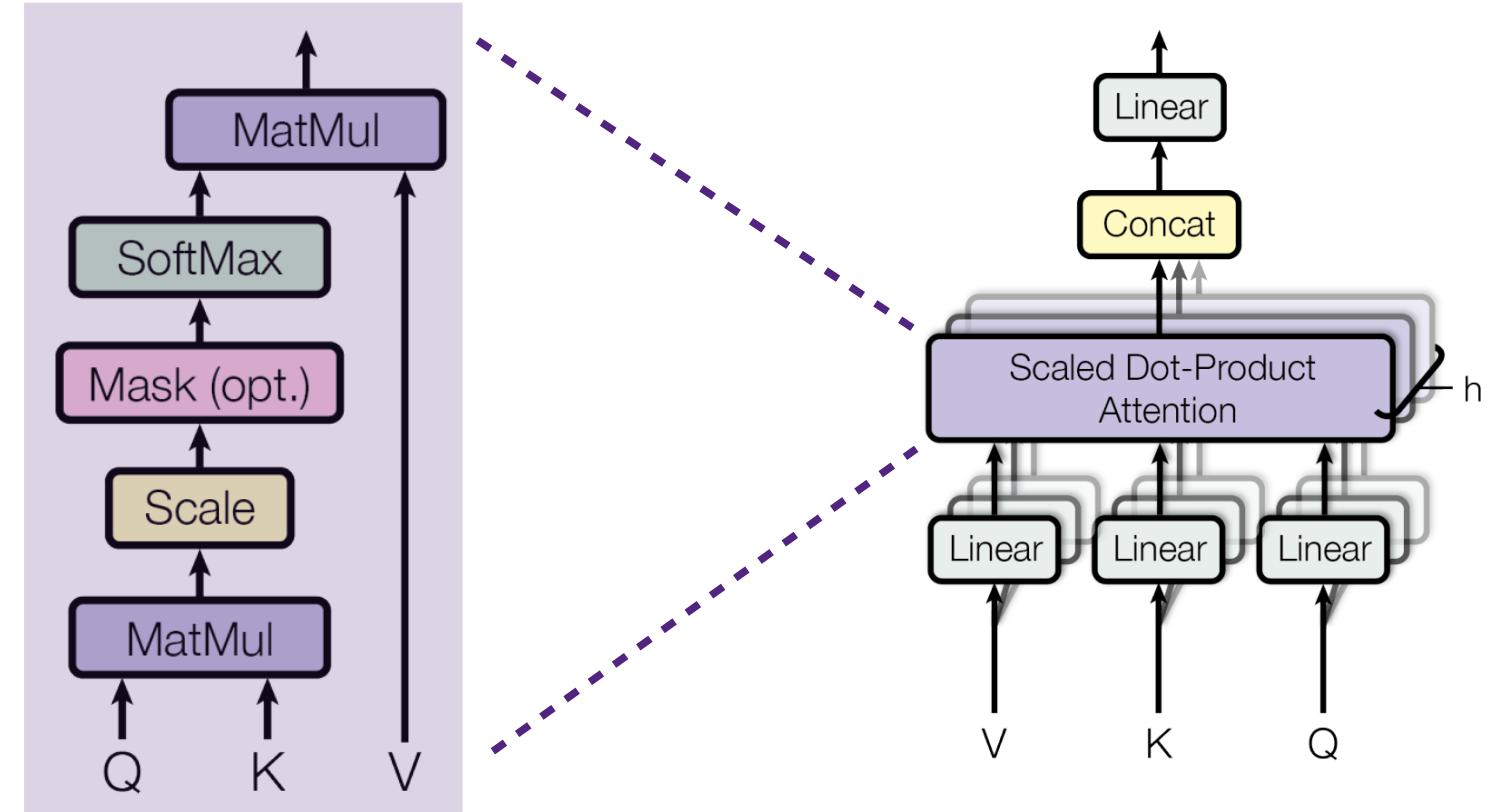
$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

Linearly combines heads' outputs.



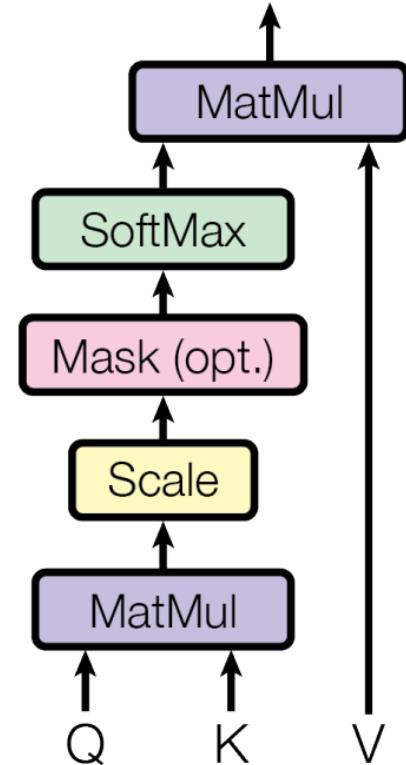
Single-head self-Attention



Single-head self-Attention

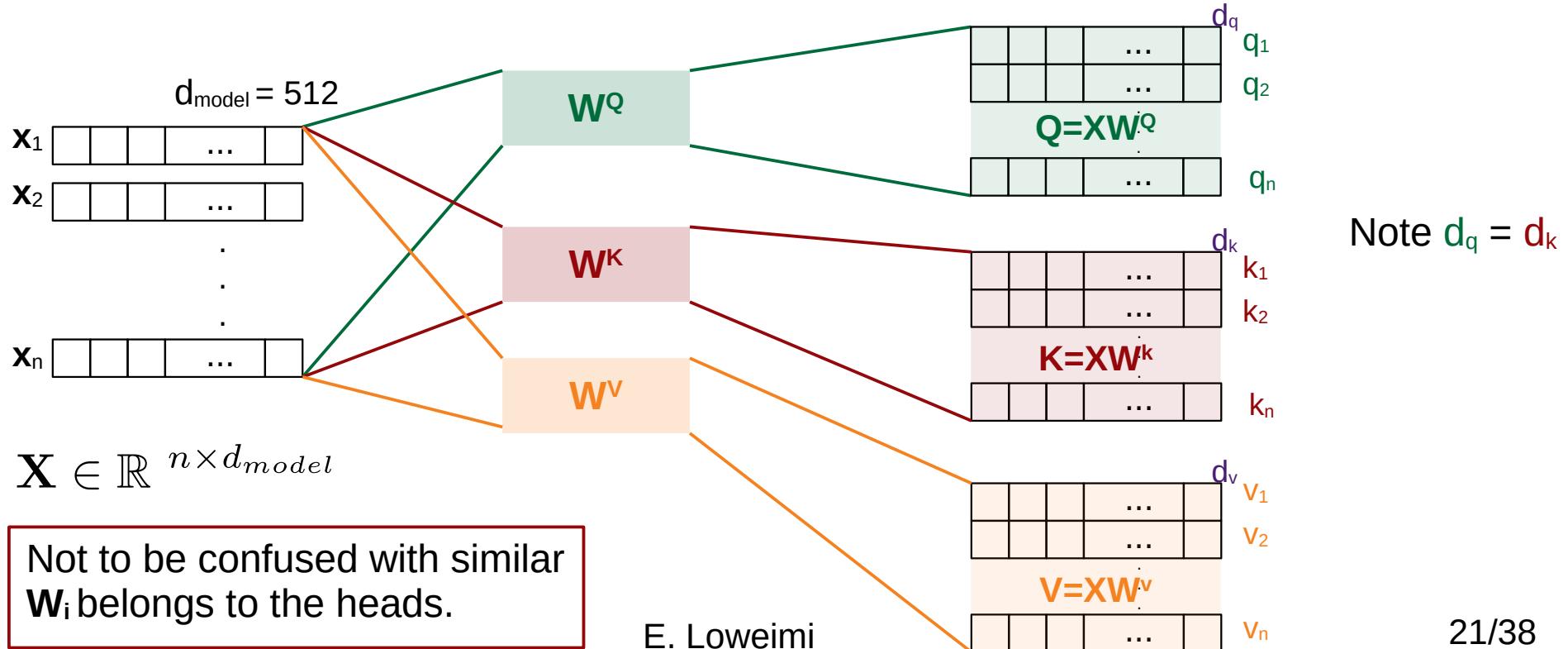
- Given: **Query**, **Key** and **Value** ({k:v})
- Output: attention-weighted mean of Values
- Weights prop. to similarity of **K** & **Q**
- Similarity: scaled-dot product
 - Scaled → to control magnitude@high d

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



Generate Q, K, V via Linear transformation

- Embedding → Linear transformation



Multi-head self-Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

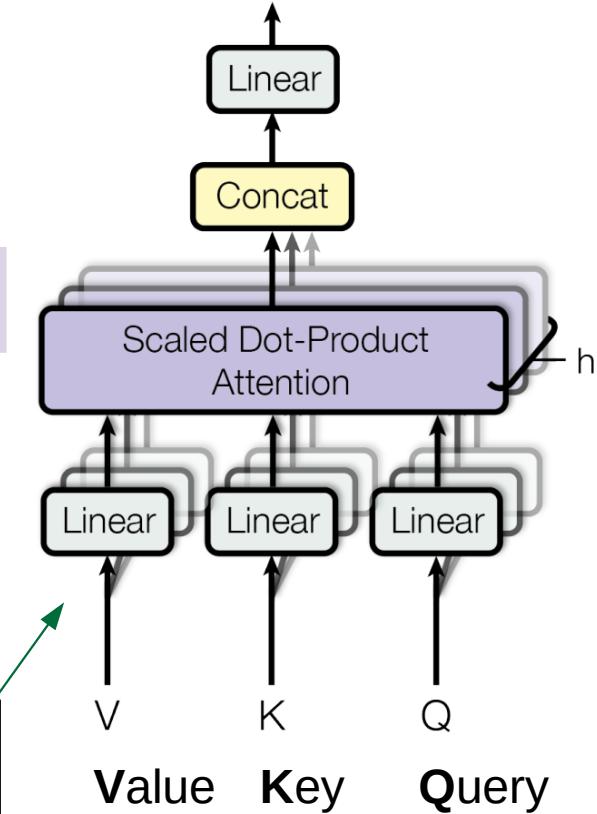
$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

$$W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

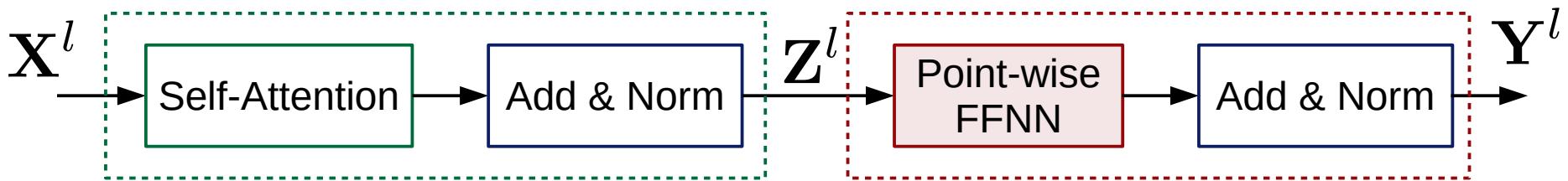
$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

- $d_q = d_k = d_v = d_{\text{model}} / h$
- $d_{\text{model}} = 512, h = 8$

Linear projection to space where dot product is a better proxy for similarity.

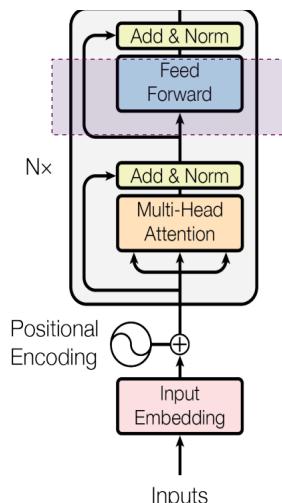
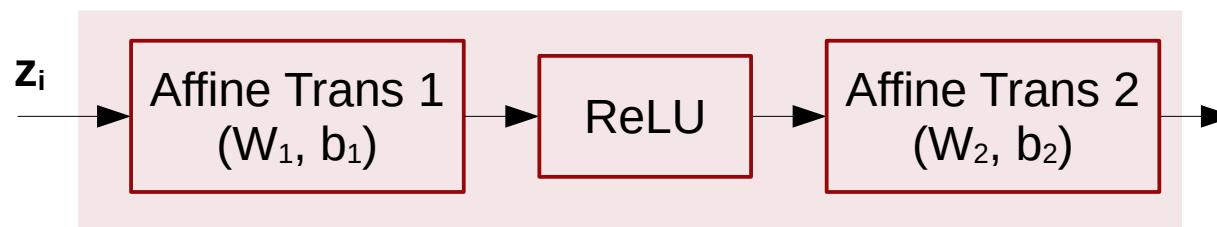


Point-wise FFNN

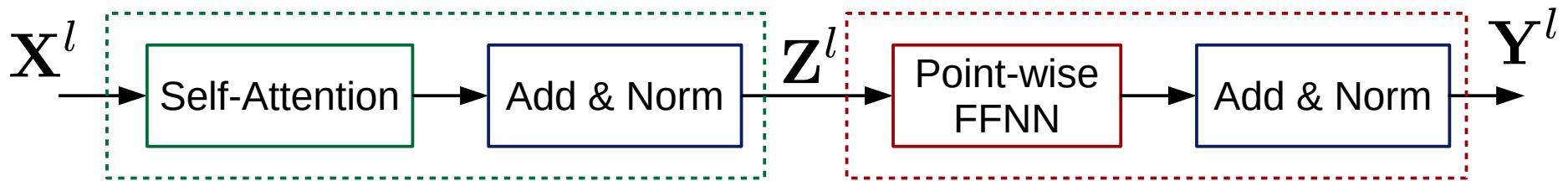


$$\mathbf{y}_i^l = FFNN(\mathbf{z}_i^l) = \text{ReLU}(\mathbf{z}_i^l \mathbf{W}_1^l + b_1^l) \mathbf{W}_2^l + b_2^l$$

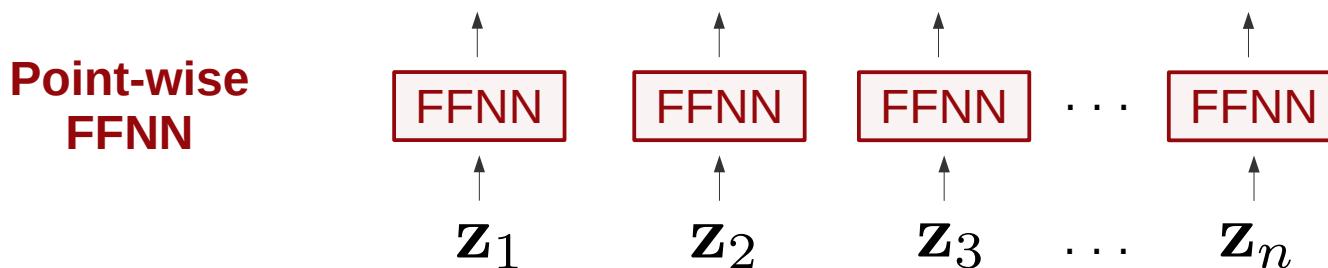
Point-wise FFNN



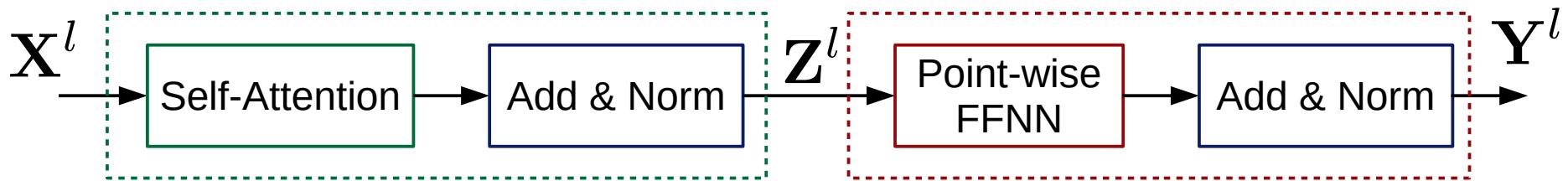
Point-wise FFNN



$$\mathbf{y}_i^l = FFNN(\mathbf{z}_i^l) = \text{ReLU}(\mathbf{z}_i^l W_1^l + b_1^l) W_2^l + b_2^l$$



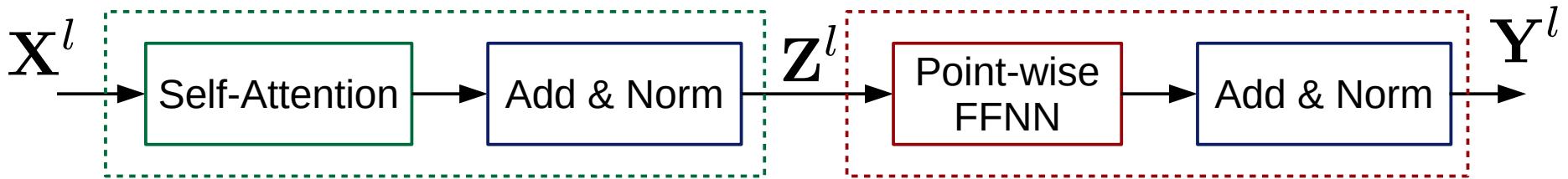
Point-wise FFNN



$$\mathbf{y}_i^l = FFNN(\mathbf{z}_i^l) = \text{ReLU}(\mathbf{z}_i^l W_1^l + b_1^l) W_2^l + b_2^l$$

- **Point-wise**: applied to each position (\mathbf{z}_i) independently & identically.
- Each layer has its own FFNN, shared inside layer.
- Dimensions: $W_1^l \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2^l \in \mathbb{R}^{d_{ff} \times d_{model}}$ ($d_{ff} = 2048$)

Point-wise FFNN



$$\mathbf{y}_i^l = FFNN(\mathbf{z}_i^l) = \text{ReLU}(\mathbf{z}_i^l W_1^l + b_1^l) W_2^l + b_2^l$$

The representation dimension does not change across layers and sublayers.

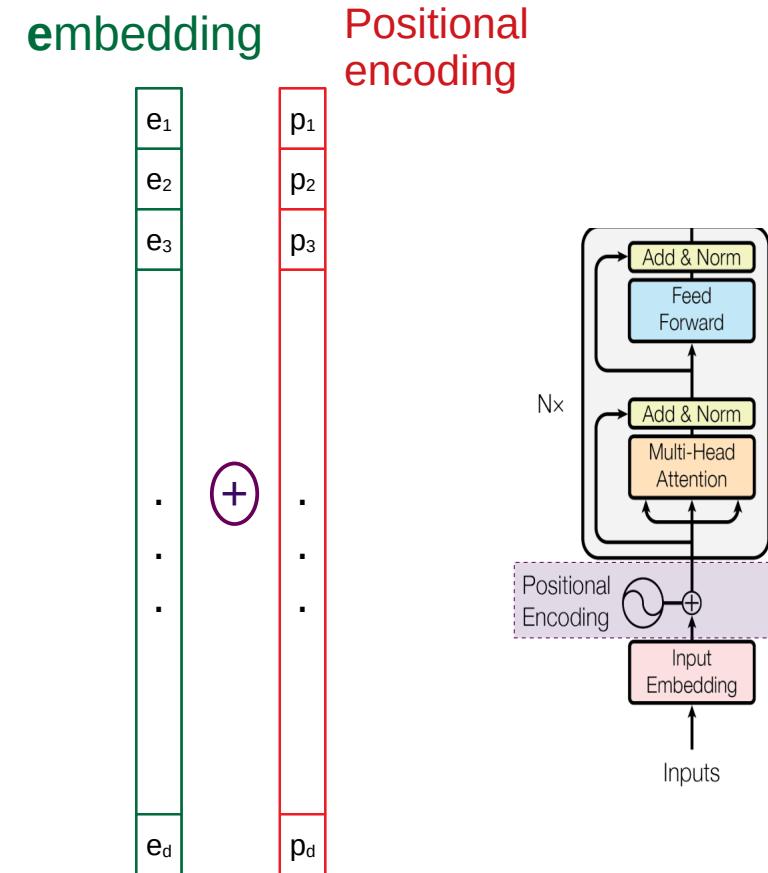
$$\mathbf{X} \in \mathbb{R}^{n \times d_{model}}$$

$$\mathbf{Z} \in \mathbb{R}^{n \times d_{model}}$$

$$\mathbf{Y} \in \mathbb{R}^{n \times d_{model}}$$

Positional Coding

- **Problem:**
 - Self-attention has no notion of temporal order
- **Solution:**
 - **Positional encoding**
 - Add it to embeddings
 - element-wise



Positional Coding

- **Problem:**
 - Self-attention has no notion of temporal order
- **Solution:**
 - Positional encoding
- Sinusoidal positional encoding
 - Limited/stable range → [-1,1]
 - Deals with any (unseen) length

$$PE_{(pos,2d)} = \sin\left(\frac{pos}{10^{\frac{8d}{D}}}\right)$$

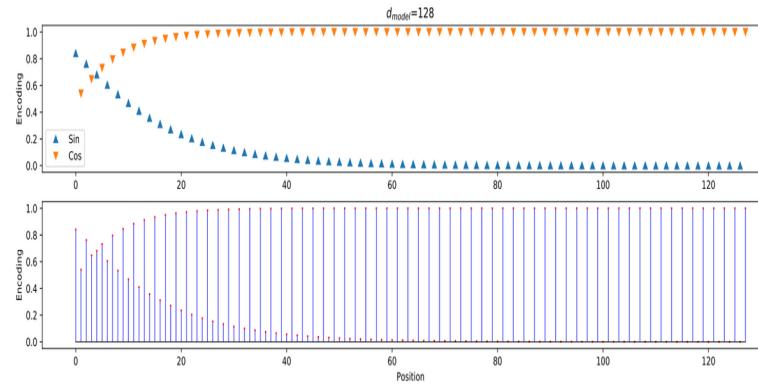
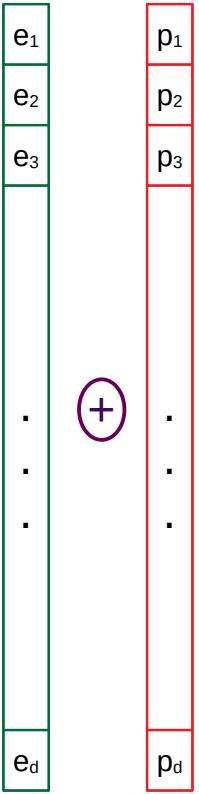
$$PE_{(pos,2d+1)} = \cos\left(\frac{pos}{10^{\frac{8d}{D}}}\right)$$

$$0 \leq pos < n$$

$$d = 0, 1, \dots, D/2$$

Positional Coding

embedding



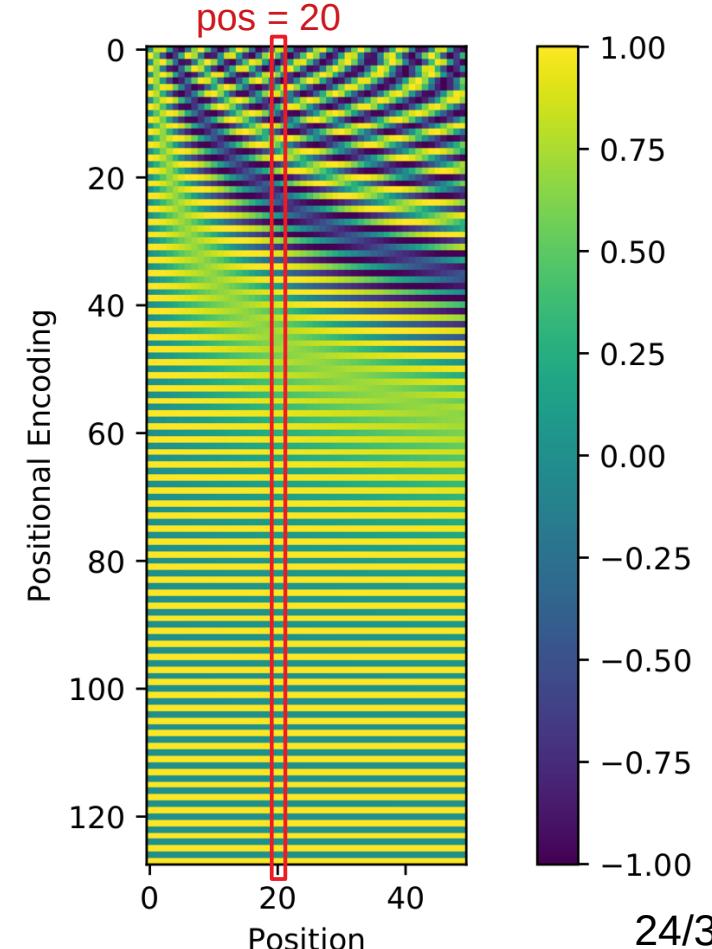
$$PE_{(pos,2d)} = \sin\left(\frac{pos}{10^{\frac{8d}{D}}}\right)$$

$$PE_{(pos,2d+1)} = \cos\left(\frac{pos}{10^{\frac{8d}{D}}}\right)$$

$$0 \leq pos < n$$

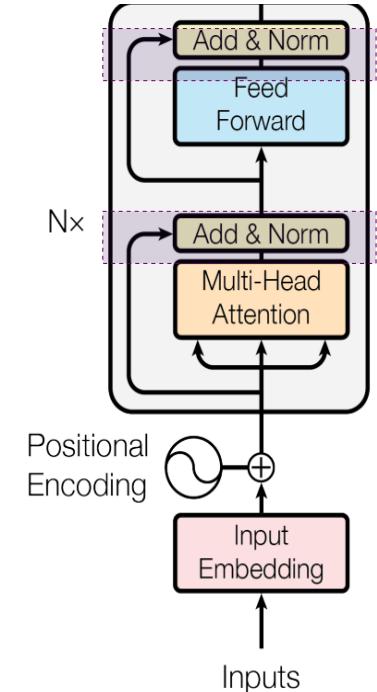
$$d = 0, 1, \dots, D/2$$

E. Loweimi



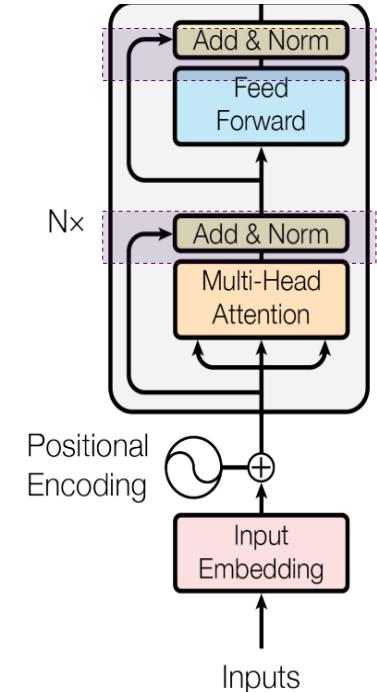
Add and Norm

- Applied after each sublayer
 - **Add** → residual connection
 - **Norm** → Layer Normalisation
 - Sublayer = **Norm**($x + \text{DropOut}\{\text{sublayer}(x)\}$)



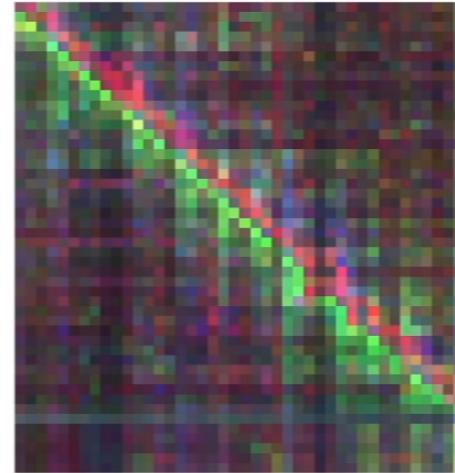
Add and Norm

- Applied after each sublayer
 - Add → residual connection
 - Norm → Layer Normalisation
 - Sublayer = Norm($x + \text{DropOut}\{\text{sublayer}(x)\}$)
- Residual connection
 - Stabilises the training
 - Injects positional info into the model

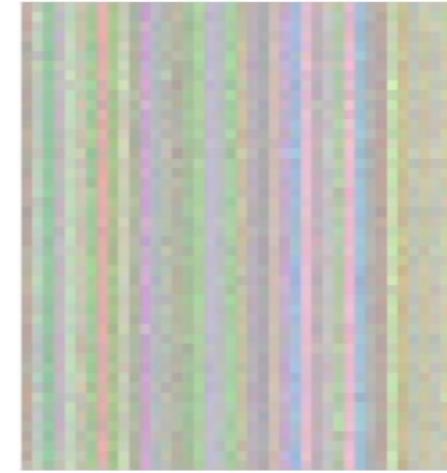


Residual Connection Role

- Residual connection injects positional info into model
 - *Diagonal alignment* in Attention Encoder-Decoder



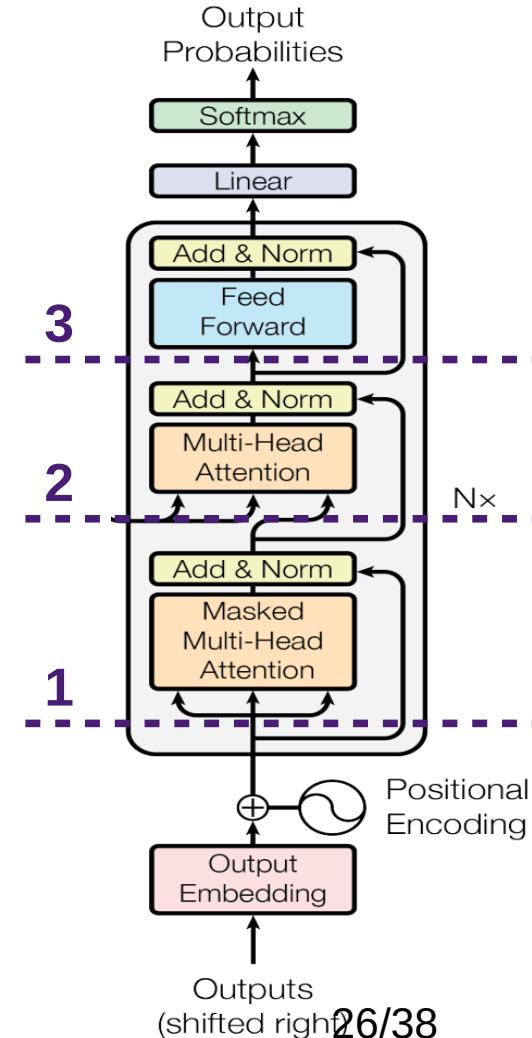
With residual
connections



Without residual
connections

Decoder

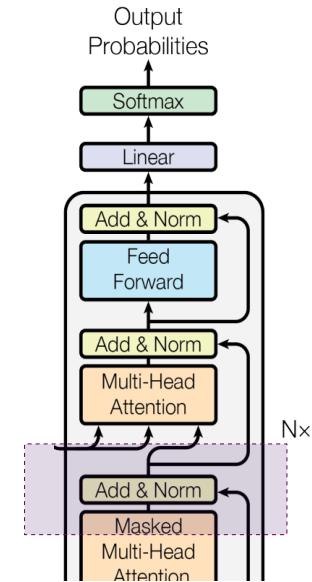
- 6 layers, each one has ...
 - Sublayer 1: Masked MHSL*
 - Sublayer 2: Attention Encoder-Decoder
 - Sublayer 3: Point-wise FFNN
- Each sublayer has **Add & Norm**



 MHSA*: Multi-head Self-attention

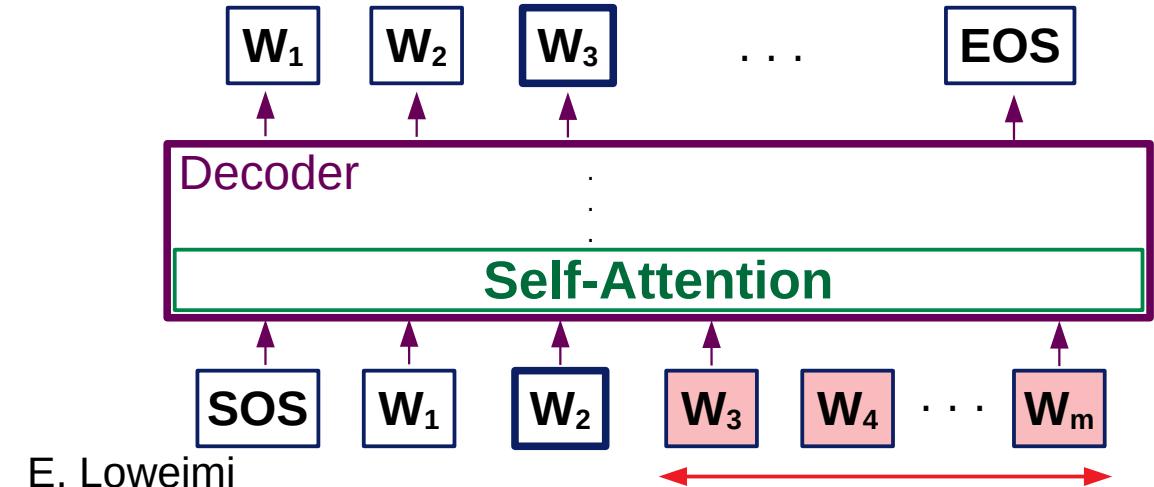
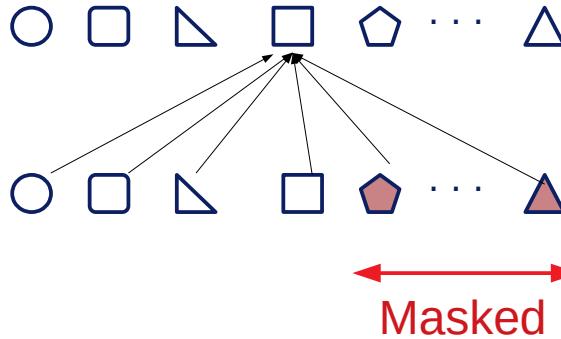
Masked Multi-head Self-Attention

- Decoder generates one word at a time, left-to-right
- Masks preserve causality and autoregressive property of decoder, e.g. at $t=3$, w_i for $i > 3$ should be masked



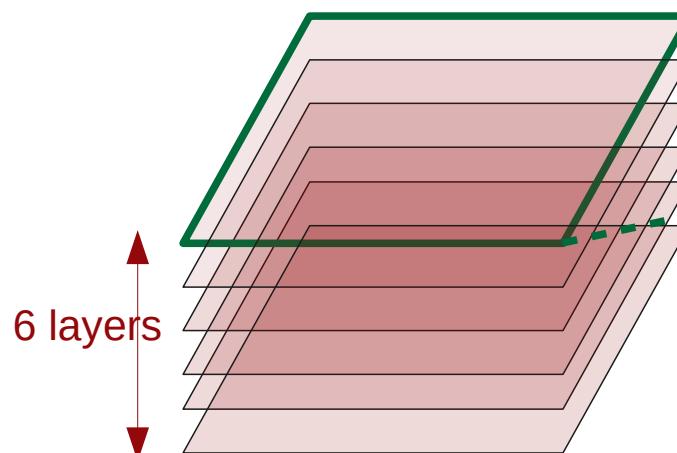
Masked Multi-head Self-Attention

- Decoder generates one word at a time, left-to-right
- Masks preserve causality and autoregressive property of decoder, e.g. at $t=3$, w_i for $i > 3$ should be masked

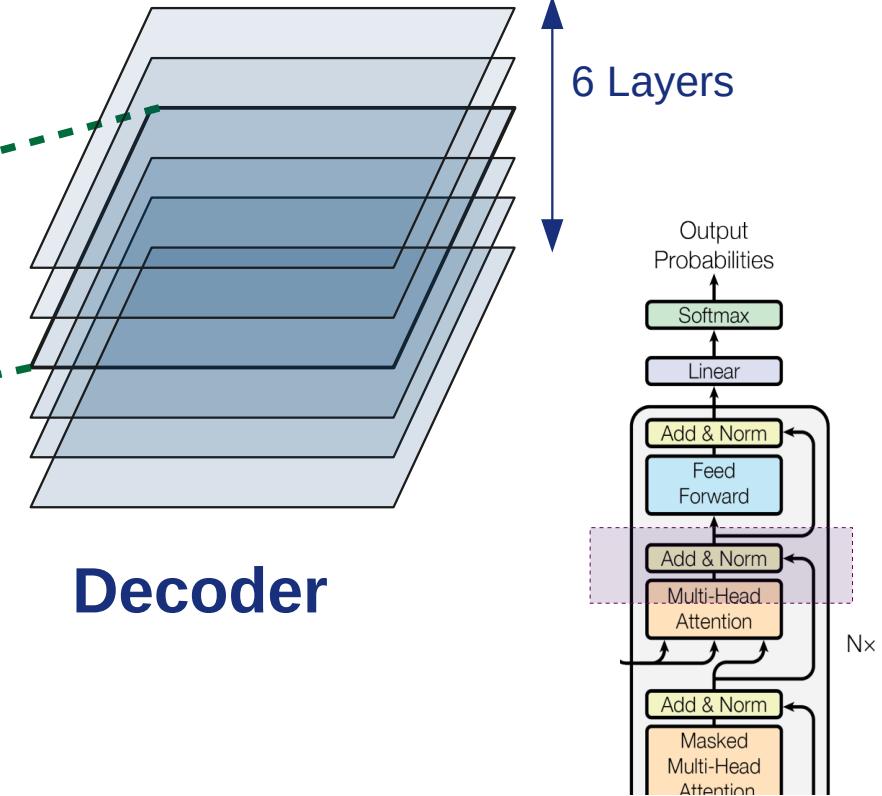


Attention Encoder-Decoder

Encoder-Decoder attention between each decoder layer and the last layer of encoder

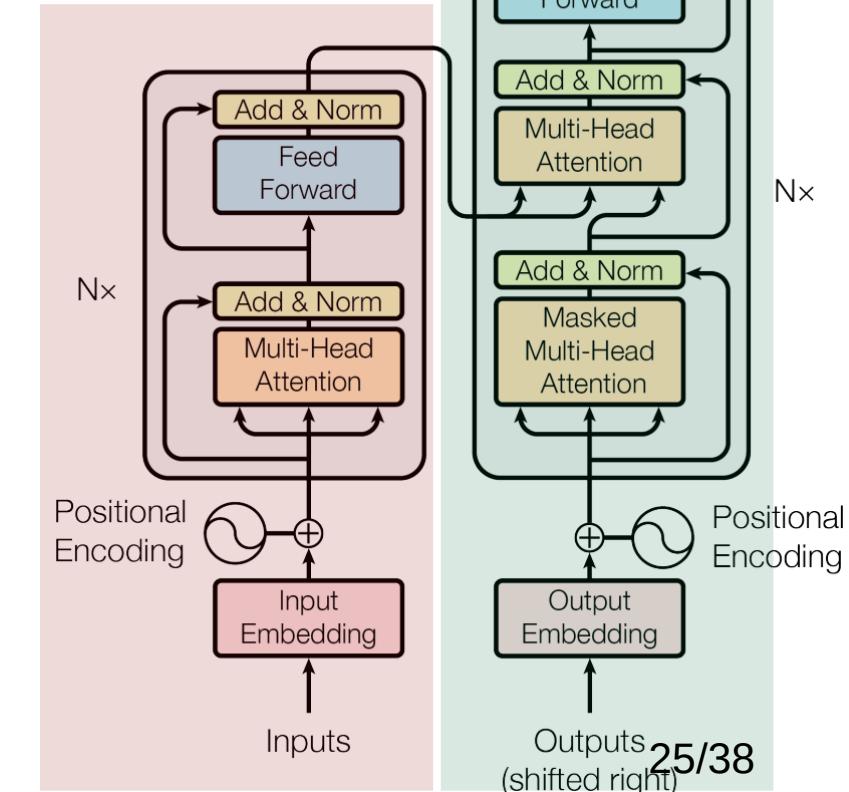


Encoder



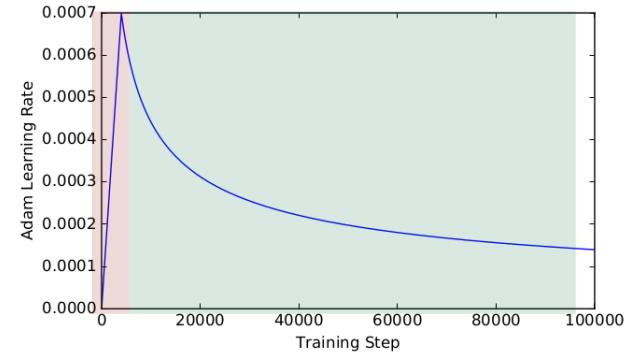
Ingredients – Recap

- **Encoder-Decoder** structure
- Positional Embedding
- Multi-Head self-Attention
- Feed Forward NN (FFNN)
- Add & Norm



Training Setup

- TensorFlow → [Tensor2Tensor library](#) → [github](#)
- Optimisation
 - Adam w/ learning rate **warmup** and **exponential decay**
- Regularisation
 - Dropout → rate: 0.1
 - Label smoothing → $\epsilon_{ls} = 0.1$
 - Relax confidence on labels (C : #classes)



$$y_c \leftarrow y_c(1 - \epsilon_{ls}) + \epsilon_{lk}/C$$

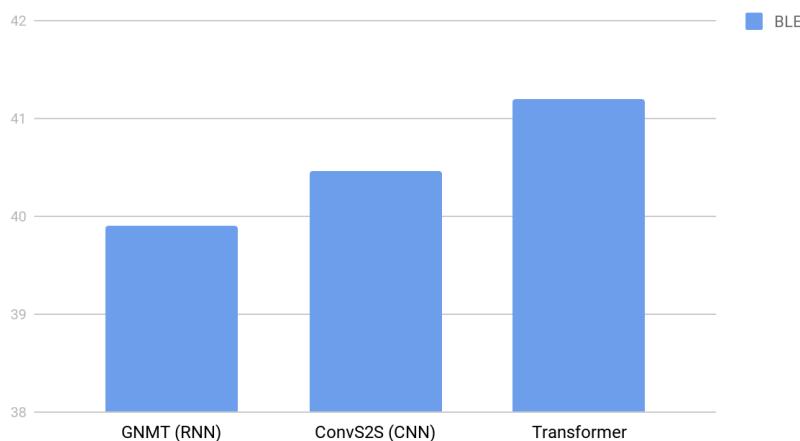
State-of-the-art on WMT 2014

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	$2.3 \cdot 10^{19}$	

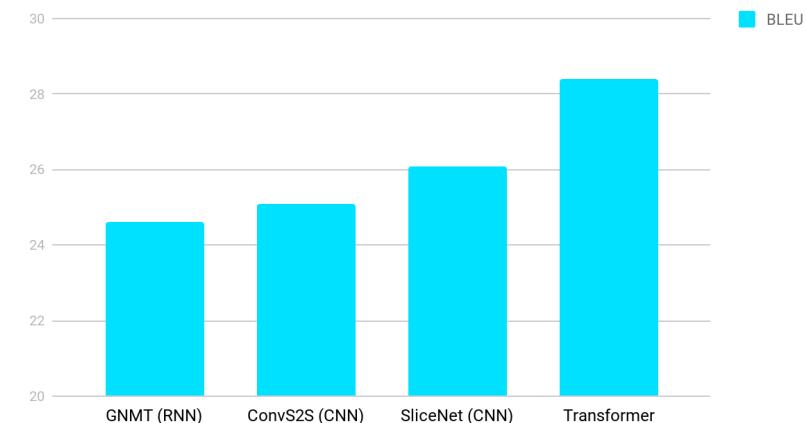
- BLEU score: * EN-DE: 28.4 * EN-FR: 41.8
- Data amount: * 4.5M pairs * 36M pairs

NMT → WMT 2014

English French Translation Quality



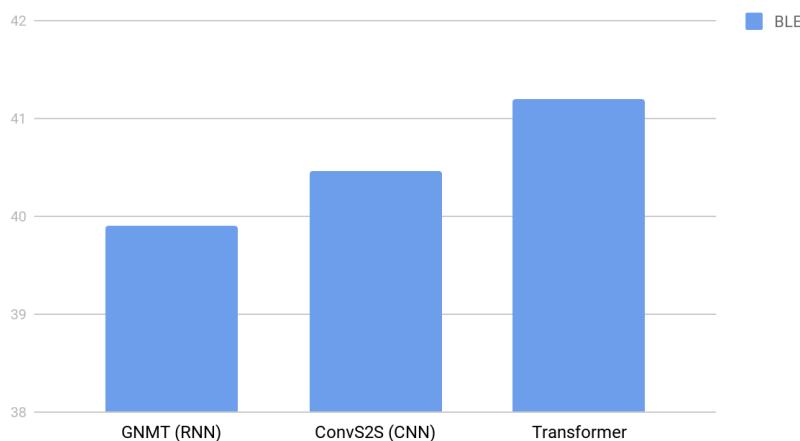
English German Translation quality



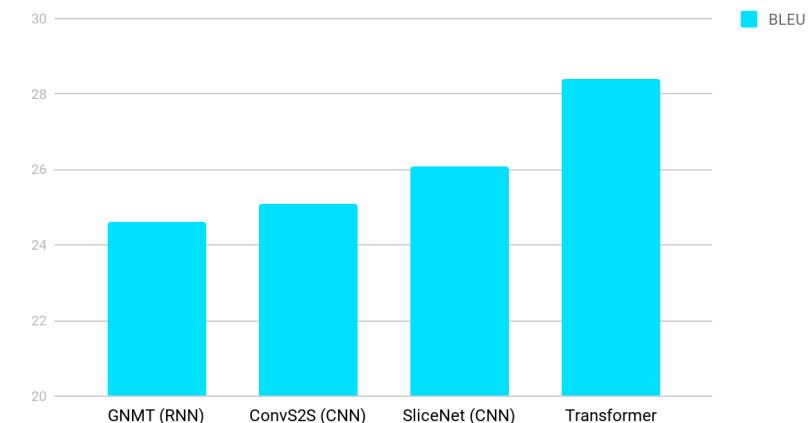
- Measure: BLEU scores (higher is better)
- Task/Data: Standard WMT newstest2014

WMT 2014

English French Translation Quality



English German Translation quality



In WMT 2016 summary report, "RNN" appeared 44 times.

In WMT 2018 report "RNN" appeared 9 and "Transformer" 63 times.

<https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture07-fancy-rnn.pdf>

Transformer Hyperparameters

- Data: devset EN-DE
 - testnews2013

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$			
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65			
(A)					1	512	512				5.29	24.9			
					4	128	128				5.00	25.5			
					16	32	32				4.91	25.8			
					32	16	16				5.01	25.4			
(B)					16				5.16		25.1	58			
					32				5.01		25.4	60			
(C)					2					6.11	23.7	36			
					4					5.19	25.3	50			
					8					4.88	25.5	80			
					256	32	32				5.75	24.5			
					1024	128	128				4.66	26.0			
					1024						5.12	25.4			
					4096						4.75	26.2			
									0.0		5.77	24.6			
(D)									0.2		4.95	25.5			
									0.0		4.67	25.3			
									0.2		5.47	25.7			
					positional embedding instead of sinusoids							4.92	25.7		
big	6	1024	4096	16					0.3	300K	4.33	26.4	213		

Transformer Hyperparameters

- Base vs big models
 - $d_{model} \rightarrow 512$ vs 1024
 - $d_{ff} \rightarrow 2048$ vs 4096
 - $h \rightarrow 8$ vs 16
 - $P_{drop} \rightarrow 0.1$ vs 0.3
 - #param $\rightarrow 65$ vs 213 M
 - Bigger model is better

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$					
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65					
(A)				1	512	512				5.29	24.9						
				4	128	128				5.00	25.5						
				16	32	32				4.91	25.8						
				32	16	16				5.01	25.4						
(B)					16					5.16	25.1	58					
										5.01	25.4	60					
(C)				2						6.11	23.7	36					
				4						5.19	25.3	50					
				8						4.88	25.5	80					
				256	32	32				5.75	24.5	28					
				1024		128				4.66	26.0	168					
				1024		128				5.12	25.4	53					
										4.75	26.2	90					
				4096						0.0	5.77	24.6					
										0.2	4.95	25.5					
										0.0	4.67	25.3					
										0.2	5.47	25.7					
(E)					positional embedding instead of sinusoids					4.92	25.7	213					
					big	6	1024	4096	16	0.3	300K	4.33					

Transformer Hyperparameters

- **(A) → #heads (h)**
 - $h=1$ → BLEU 0.9 worse
 - $h=16$ → BLEU 0.4 worse
 - h should not be too large

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$			
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65			
(A)				1	512	512				5.29	24.9				
				4	128	128				5.00	25.5				
				16	32	32				4.91	25.8				
				32	16	16				5.01	25.4				
(B)							16			5.16	25.1	58			
							32			5.01	25.4	60			
(C)				2						6.11	23.7	36			
				4						5.19	25.3	50			
				8						4.88	25.5	80			
				256			32	32		5.75	24.5	28			
				1024			128	128		4.66	26.0	168			
				1024			128	128		5.12	25.4	53			
										4.75	26.2	90			
(D)							0.0	0.0	0.0	5.77	24.6				
							0.2			4.95	25.5				
							0.2	0.2		4.67	25.3				
										5.47	25.7				
(E)	positional embedding instead of sinusoids								0.0	4.92	25.7				
big	6	1024	4096	16				0.3	300K	4.33	26.4	213			

Transformer Hyperparameters

- **(B) → key size (d_k)**
 - Reducing key size hurts
 - More sophisticated compatibility function may be beneficial

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$			
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65			
(A)					1	512	512				5.29	24.9			
					4	128	128				5.00	25.5			
					16	32	32				4.91	25.8			
					32	16	16				5.01	25.4			
(B)					16				5.16		25.1	58			
					32				5.01		25.4	60			
(C)					2					6.11	23.7	36			
					4					5.19	25.3	50			
					8					4.88	25.5	80			
					256	32	32				5.75	24.5			
					1024	128	128				4.66	26.0			
					1024						5.12	25.4			
					4096						4.75	26.2			
									0.0	5.77	24.6				
(D)									0.2	4.95	25.5				
									0.0	4.67	25.3				
									0.2	5.47	25.7				
					positional embedding instead of sinusoids							4.92			
big	6	1024	4096	16					0.3	300K	4.33	26.4			
												213			

Transformer Hyperparameters

- **(C) → Model size**
 - Larger N helps
 - Larger d_{model} helps
 - Larger d_{ff} helps
 - Larger model is better

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$					
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65					
(A)				1	512	512				5.29	24.9						
				4	128	128				5.00	25.5						
				16	32	32				4.91	25.8						
				32	16	16				5.01	25.4						
(B)					16					5.16	25.1	58					
										5.01	25.4	60					
(C)				2						6.11	23.7	36					
				4						5.19	25.3	50					
				8						4.88	25.5	80					
				256						5.75	24.5	28					
										4.66	26.0	168					
										5.12	25.4	53					
										4.75	26.2	90					
(D)				1024		32	0.0	0.2		5.77	24.6						
										4.95	25.5						
										4.67	25.3						
										5.47	25.7						
(E)				4096	positional embedding instead of sinusoids												
					4.92					25.7							
big	6	1024	4096	16		0.3	300K		4.33	26.4	213						

Transformer Hyperparameters

- **(D) → Regularisation**
 - Dropout helps
 - Label smoothing helps
 - Rate should be adjusted
 - 0.1 better than 0 or 0.2

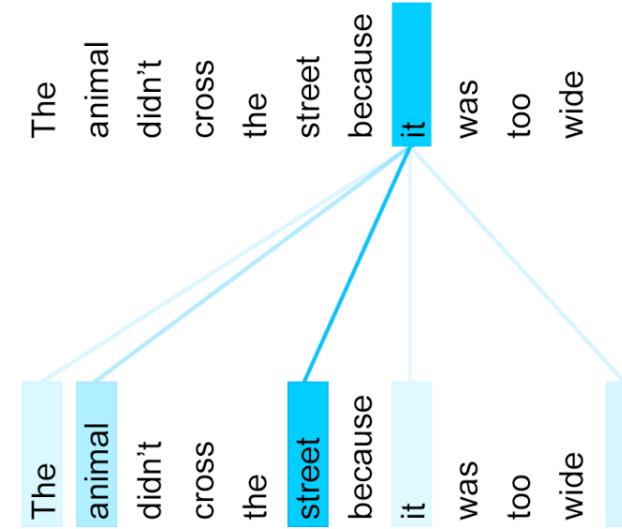
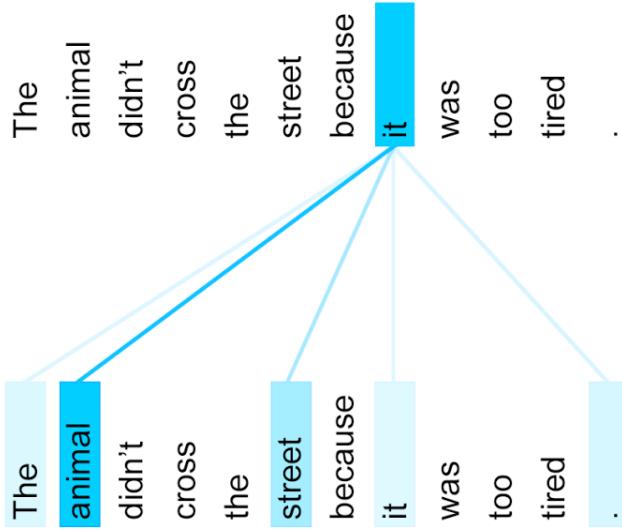
	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)					1	512	512					5.29
					4	128	128					5.00
					16	32	32					4.91
					32	16	16					5.01
(B)					16							
					32							
(C)					2					6.11	23.7	36
					4					5.19	25.3	50
					8					4.88	25.5	80
					256	32	32					5.75
					1024	128	128					4.66
					1024							
					4096							
									0.0	5.77	24.6	
(D)									0.2	4.95	25.5	
									0.0	4.67	25.3	
									0.2	5.47	25.7	
					positional embedding instead of sinusoids							
(E)									0.3	300K	4.92	25.7
big	6	1024	4096	16					0.3	300K	4.33	26.4
									300K			213

Transformer Hyperparameters

- **(E) → Positional Coding**
 - Learning embedding slightly worsen results
 - Sinusoidal encoding is good enough

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)					1	512	512					5.29
					4	128	128					5.00
					16	32	32					4.91
					32	16	16					5.01
(B)					16							5.16
					32							5.01
(C)					2					6.11	23.7	36
					4					5.19	25.3	50
					8					4.88	25.5	80
					256	32	32					5.75
					1024	128	128					4.66
					1024							5.12
					4096							4.75
									0.0	5.77	24.6	
									0.2	4.95	25.5	
									0.0	4.67	25.3	
									0.2	5.47	25.7	
(E)	positional embedding instead of sinusoids										4.92	25.7
	big	6	1024	4096	16					0.3	300K	4.33
												213

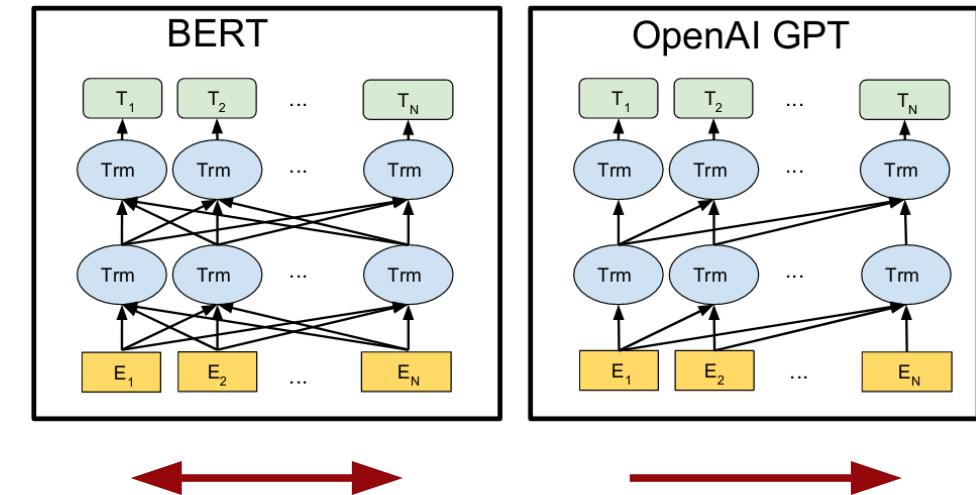
Coreference Resolution (Winograd Schemas)



- Encoder self-attention visualisation at layer 5 (out of 6) ...
 - * The **animal** didn't cross the **street** because **it** was too **tired**.
 - * The **animal** didn't cross the **street** because **it** was too **wide**.

Ongoing Work ...

- BERT and OpenAI GPT
- Self-supervision and classification
- Multitask learning
- And many more ...





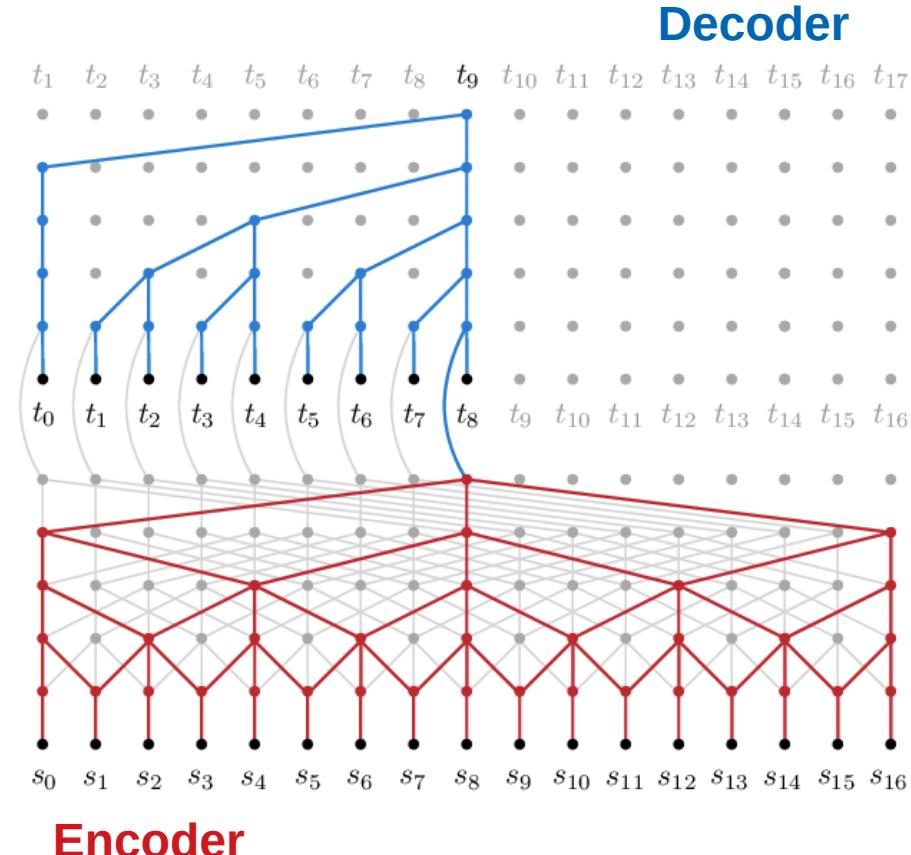
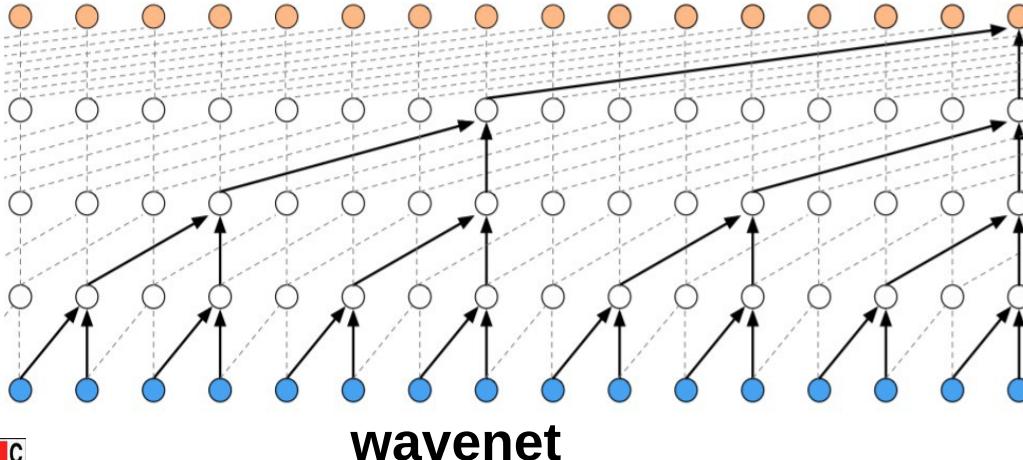
That's it!

- Thanks for your ATTENTION!
 - That's all I needed ;-)
- Q/A
- Appendix
 - CNN Encoder-Decoder



CNN Encoder-Decoder

- Exp: ByteNet and ConvS2S



CNN Encoder-Decoder

- CNN advantages
 - Sparsity of connections → weight sharing
 - Exploiting local dependencies → kernel size
 - Translational invariance → pooling
 - Easy to parallelise within layer

CNN Encoder-Decoder

- Modelling long-range dependencies requires
 - Many layers → makes training harder
 - Large kernel → computational cost, overfitting
- Path length between positions (in a sequence)
 - Linear ↔ no dilation
 - Log ↔ with dilation

