Institute of Neural Information Processing | Ulm University

LEARNING WITH ATTENTION

Dr. Sebastian Gottwald

1. RECURRENT NEURAL NETWORKS (RNNS)

RECAP: SUPERVISED LEARNING IN FEED-FORWARD ANNS

- **Feed-forward Artificial neural networks** (ANNs) are mappings $f_\theta : \mathcal{X} \to \mathcal{Y}$, from a space of inputs $\mathcal{X} \subset \mathbb{R}^n$ to a space of outputs $\mathcal{Y} \subset \mathbb{R}^m$, that are explicitly parametrized by a set of so-called **weights** (and biases).

- **Network architecture:** the details of this parametrization, e.g. for a *multi-layer perceptron* (MLP), $\theta = (W_1, b_1, \dots, W_d, b_d)$, and

$$f_\theta(x) = f_1(f_2(\cdots f_d(x))), \qquad f_k(x) = \sigma_k(W_k x + b_k)$$

  where d is the number of layers, $(W_k x)_i = \sum_j (W_k)_{ij} x_j$, and $\sigma_k : \mathbb{R} \to \mathbb{R}$ a non-linear (so-called) **activation** function (sigmoid, tanh, ReLU, etc.) that is applied *componentwise*.

- **Supervised learning:** Given training data $(x_i, y_i)_{i=1}^N$ of inputs x_i and targets y_i, find θ such that $f_\theta(x_i) \approx y_i$ for all $i = 1, \dots, N$, by minimizing a differentiable **loss function** that determines the discrepancy between $f_\theta(x_i)$ and y_i

Note: We can absorb biases in the weight matrices by assuming that one dimension of x, usually the last, is set to 1.

RESTRICTING FEED-FORWARD NETWORKS

By various so-called **universal approximation theorems**, fully connected feed-forward ANNs can approximate well-behaved functions between Euclidean spaces arbitrarily well. These theorems **do not specify mechanisms** by which such mappings can be learned efficiently.

Instead of simply using the most universal fully connected model, in practice it is very valuable (faster and less noisy!) to **choose a more restrictive network architecture that fits the given problem**.

Example: convolutional neural networks (CNNs) are designed to learn from 2-dimensional data with neighbourhood relationships among the entries, in particular for image recognition.

Here, we are interested in learning from **sequential data** to perform tasks such as

- **Prediction**: Stock market, weather forecast, ...
- **Sequence transformation**: Language translation, video-to-text, text-to-speech, ...

SEQUENTIAL DATA

Consider data, where a **single datapoint consists of a sequence** of items, e.g. a sequence of images (a video), or a sequence of words (a text).

One could represent such datapoints as a single flat vector, but then the network would have to **relearn the original sequential structure** of the data from scratch, e.g. the relationship between the pixels of two subsequent images, or the grouping of letters as words or sentences.

In general, there are two types of internal dependencies in a sequential datapoint:

- **Short-range** dependencies, i.e. relationships between subsequent items, e.g., two words that have a high chance of following each other (this is similar to local relationships between pixels in images, and thus could be learned using convolutional networks)
- **Long-range** dependencies, for example, consider the two sentences "The **animal** didn't cross the street because **it** was too **tired**" and "The animal didn't cross the **street** because **it** was too **wide**." (important for translation into languages like German or French)

n-GRAM MODELS

In theory, if we had a large dataset $\{\mathbf{x}^1, \mathbf{x}^2, \dots\}$ of sequences $\mathbf{x} = (x_1, \dots, x_N)$ we could find a probability distribution

$$p(\mathbf{X}) = \prod_{k=1}^{N} p(X_k | X_1, \dots, X_{k-1}).$$

However, for reasonably large N, the amount of datapoints required is unreasonable (the number of terms grows exponentially with N). Obvious approximations:

- **Markov assumption**: $p(X_k|X_1, \dots, X_{k-1}) = p(X_k|X_{k-1})$
- **2nd order Markov assumption**: $p(X_k|X_1, \dots, X_{k-1}) = p(X_k|X_{k-1}, X_{k-2})$
- **n-gram models**: $p(X_k|X_1, \dots, X_{k-1}) = p(X_k|X_{k-1}, \dots, X_{k-(n-1)})$

Note: n-gram models are often used for **autocompletion** in searchboxes, e.g. on the web, where the "grams" are usually letters or parts of words.

BASIC STRUCTURE OF RNNS

In Hidden Markov Models (HMMs) we have seen another possible solution: track the sequence history using **hidden states**, i.e. in spirit,

$$p(X_k|X_1, \dots, X_{k-1}) \approx p(X_k|\text{hidden state}(k))$$

The same idea applied to neural networks results in RNNs:

- The **task** is to map an input sequence $\mathbf{x}_1, \dots, \mathbf{x}_N$ to an output sequence $\mathbf{y}_1, \dots, \mathbf{y}_N$
- Introduce **hidden states** $\mathbf{h}_1, \dots, \mathbf{h}_N$, where $\mathbf{h}_k$ is a function of the previous state $\mathbf{h}_{k-1}$ and the current input $\mathbf{x}_k$, i.e.

$$\mathbf{h}_k = f(\mathbf{h}_{k-1}, \mathbf{x}_k)$$

- Each **output** is a function of the current hidden state, i.e. $\mathbf{y}_k = g(\mathbf{h}_k)$

In particular, f and g do **not** depend on the timestep, but are **shared among all** k.

SINGLE-LAYER RNNS

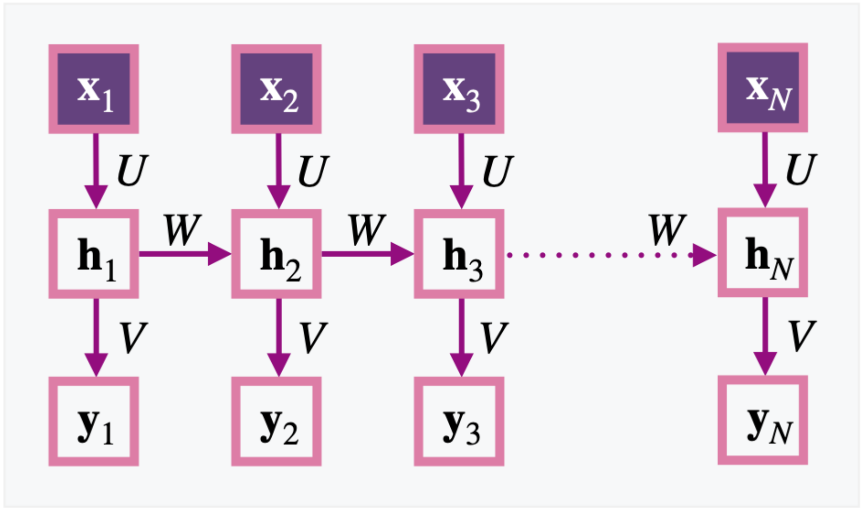
A single layer RNN is defined by three weight matrices U, V, W, s.th.

$$\mathbf{h}_k = \sigma(U\mathbf{x}_k + W\mathbf{h}_{k-1})$$

and

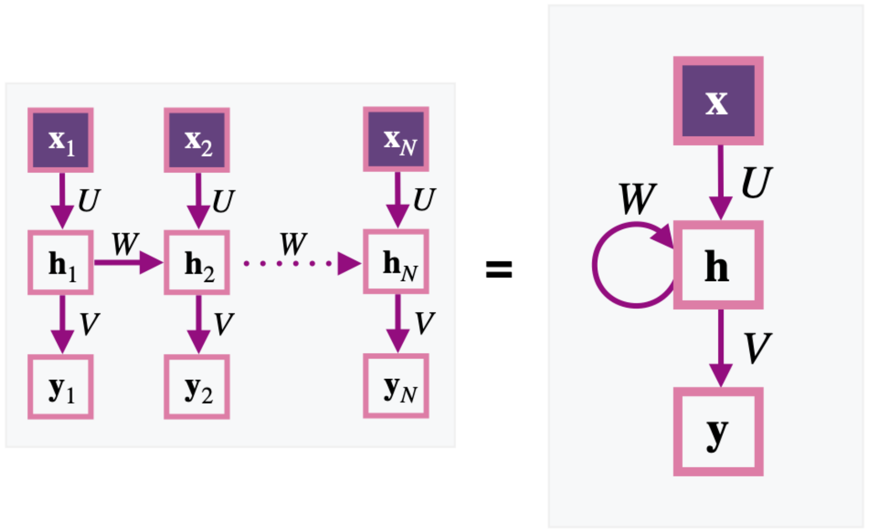
$$\mathbf{y}_k = \sigma(V\mathbf{h}_k)$$

where σ denotes a non-linear activation function.

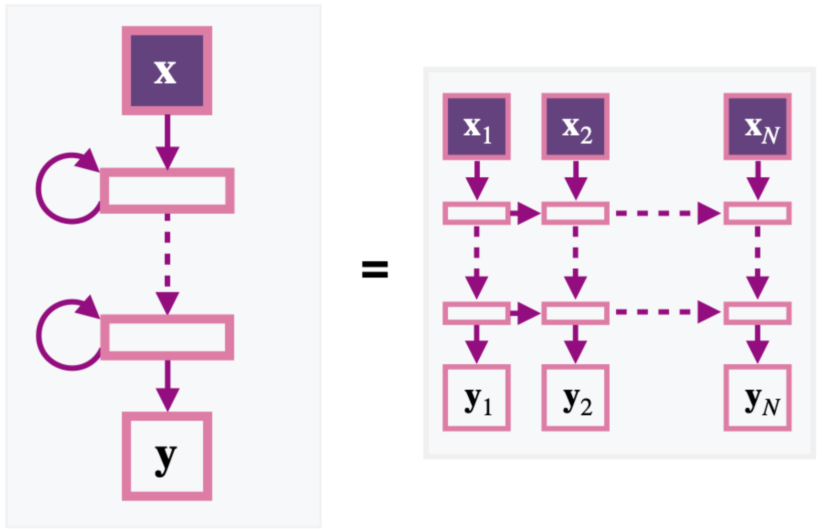


Note: The main difference to a feed-forward network is the **parameter sharing** between the time steps.

GRAPHICAL REPRESENTATIONS



a **single-layer** RNN



a **multilayer** RNN

EXAMPLE APPLICATIONS OF RNNS

- **Element-wise classification** (symbol-to-symbol mappings), e.g. parts of speech tagging (noun, verb, article, etc..), video frame classification, etc.
- **Sequence generation** (inputs are previous outputs), e.g. generation of music, speech, text, etc.
- **Conditional sequence generation** (start running the RNN on a given input and then continue with generation), e.g. handwriting generation in a style of a given person, image captioning (e.g. use output of a cnn as initial state)
- **Sequence classification** (sequence as input, only one output), e.g. sentiment analysis, movie classification into genres, etc....
- **Sequence translation** (sequence-to-sequence mappings): need more advanced architectures (later)

VARIANTS OF RNNS

Depending on how the layers of a multilayer RNN are connected and how the functions f and g are defined, we obtain

- Gated Recurrent Units networks (GRUs)
- Long-Short-Term-Memory networks (LSTMs)
- Deep RNNs

and many more. See e.g. the Dive into Deep Learning open source book for an overview.

The listed variants address an important problem of ordinary RNNs that appears during training with gradient descent: **exploding and vanishing gradients**.

GRADIENTS

Irrespective of the choice of the loss function, the derivative

$$\frac{\partial(\mathbf{h}_k)_m}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\sigma\Big((U\mathbf{x}_k)_m + \sum_n w_{mn}(\mathbf{h}_{k-1})_n\Big) \propto \delta_{im}(\mathbf{h}_{k-1})_j + \sum_n w_{mn}\frac{\partial(\mathbf{h}_{k-1})_n}{\partial w_{ij}}$$

appears in the update rule for the weight matrix $W = (w_{ij})_{i,j}$.

Hence, for the derivative of $\mathbf{h}_k$ we need the derivative of $\mathbf{h}_{k-1}$, for which we need the derivative of $\mathbf{h}_{k-2}$, and so on. There are two issues with this recurrence relation for large k:

- In theory, we would have to calculate the gradients of all previous time steps for every update (in practice the weights are only updated in chunks).
- More troubling is the factor w_{mn} linking the derivative of $\mathbf{h}_k$ and the derivative of $\mathbf{h}_{k-1}$, which for large k will blow up if it is larger than 1 or go to 0 if it is smaller than one.

2. ENCODER-DECODER ARCHITECTURES

SEQUENCE-TO-SEQUENCE MODELS

While exploding and vanishing gradients can be dealt with by more advanced modules (e.g. LSTMs, GRUs) there are still fundamental **problems with the underlying structure** of the RNNs discussed so far:

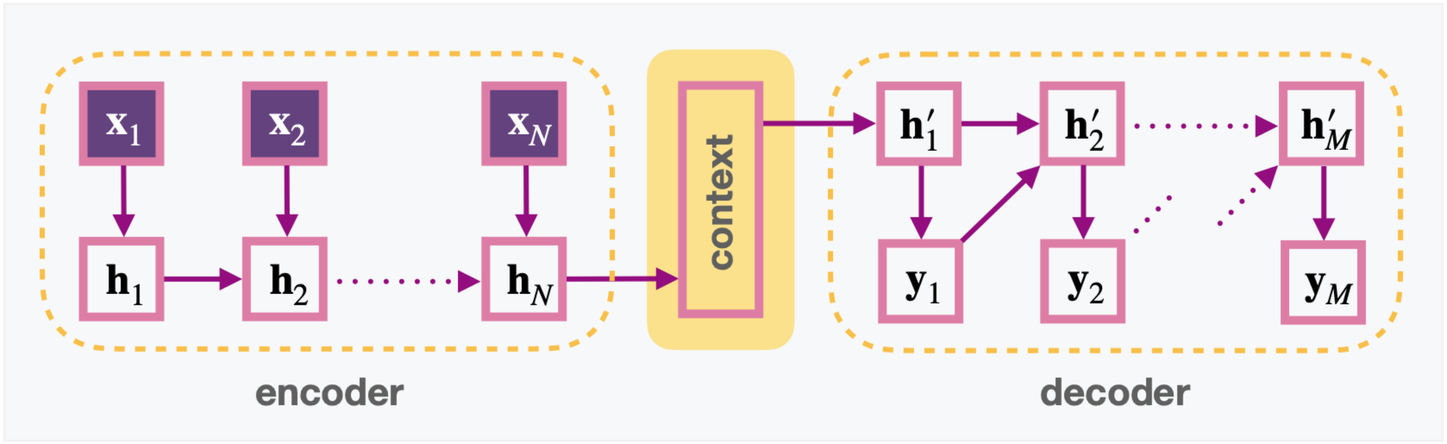
- What if the input and output sequences **do not have the same length**?
- What if the **order** of input and output sequences are different, e.g. the **start** of the output sequence **depends on the end** of the input sequence?

Possible Solution: Wait until the full sequence has been processed before producing the output sequence. This is known as **encoder-decoder architecture**, originally introduced for language translation, known as *Neural Machine Translation* (NMT).

Note: Another attempt to solve the issue with processing order are so-called *Bidirectional RNNs*, which contain a forward and a backward pass.

BASIC ENCODER-DECODER ARCHITECTURE

Connect two RNNs serially, **one for reading the input**, and **one for producing the output**:

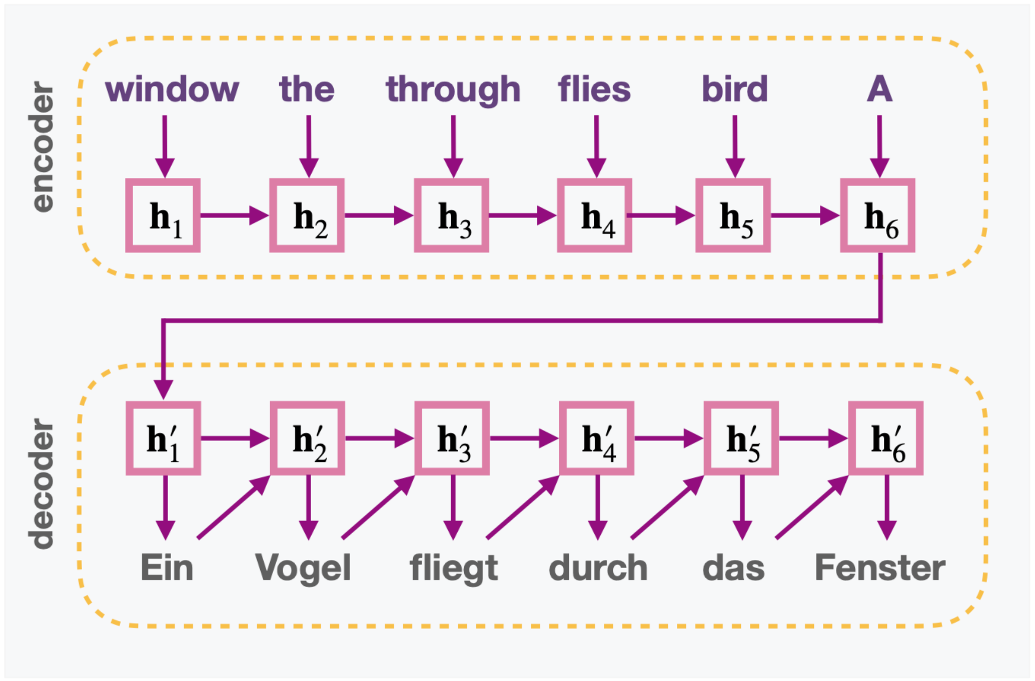


- **Encoder:** Calculate hidden states as before, i.e. $\mathbf{h}_k = f_{\text{enc}}(\mathbf{h}_{k-1}, \mathbf{x}_k)$,
- **Context:** In the basic architecture, the context is simply the last encoder state: $\mathbf{c} = \mathbf{h}_N$
- **Decoder:** Initialize $\mathbf{h}'_1 = \mathbf{c}$, for $k > 1$ let the previous output be the input to $\mathbf{h}'_k$, i.e. $\mathbf{h}'_k = f_{\text{dec}}(\mathbf{h}'_{k-1}, \mathbf{y}_{k-1})$, and hidden states produce outputs as usual, i.e. $\mathbf{y}_k = g_{\text{dec}}(\mathbf{h}'_k)$.

ENCODER-DECODER RNN
EXAMPLE

The encoder-decoder architecture allows to process full sentences, so that, e.g., when translating "A bird flies through the window" to German, "the" can be correctly translated to "das" (instead of "die" or "der").

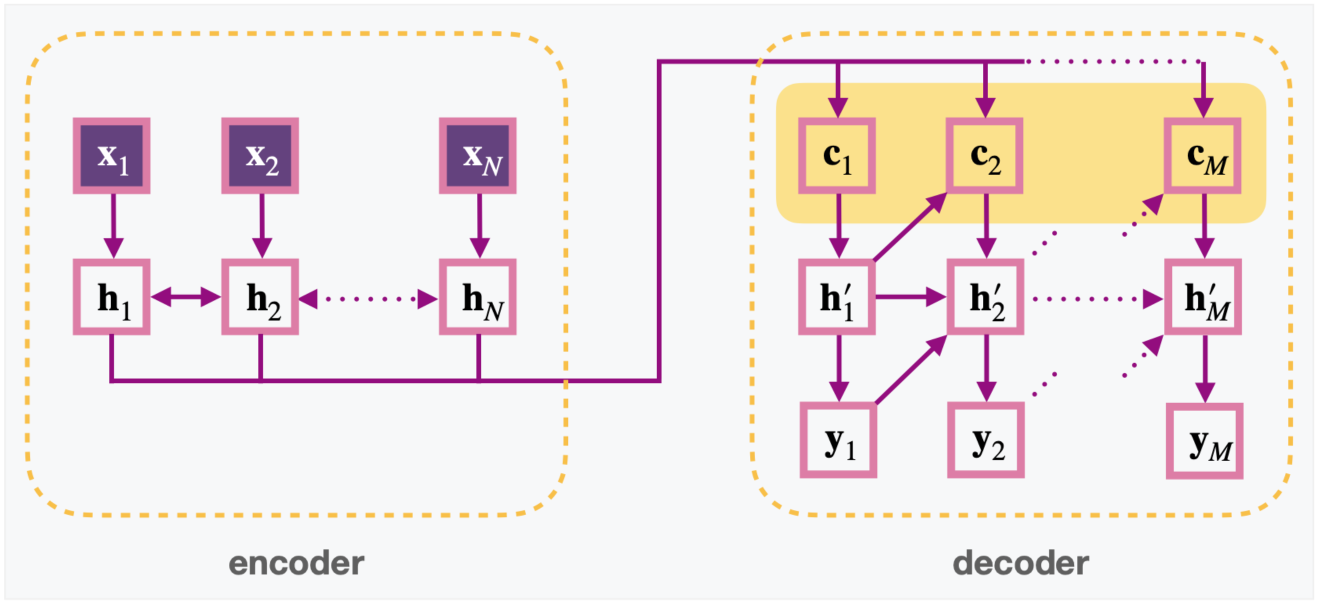
Problem: The information of each word has to travel through 7 states.



Solution attempt: Reverse the order of the input sequence, so that the output sequence can be "unpacked".

This is an improvement, but still has **long travel distances** for words at the **end of a sentence** and when the **order of the output is different** from the input.

REFINEMENT OF ENCODER-DECODER ARCHITECTURES



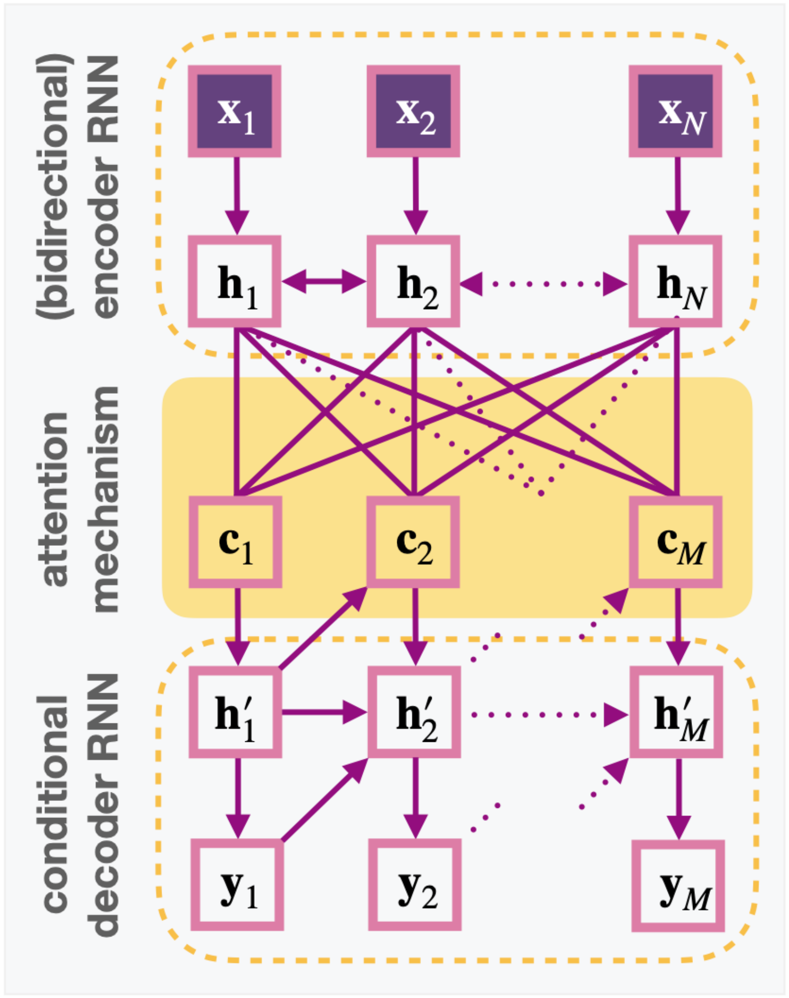
Sutskever et al. (2014): Basic encoder-decoder architecture

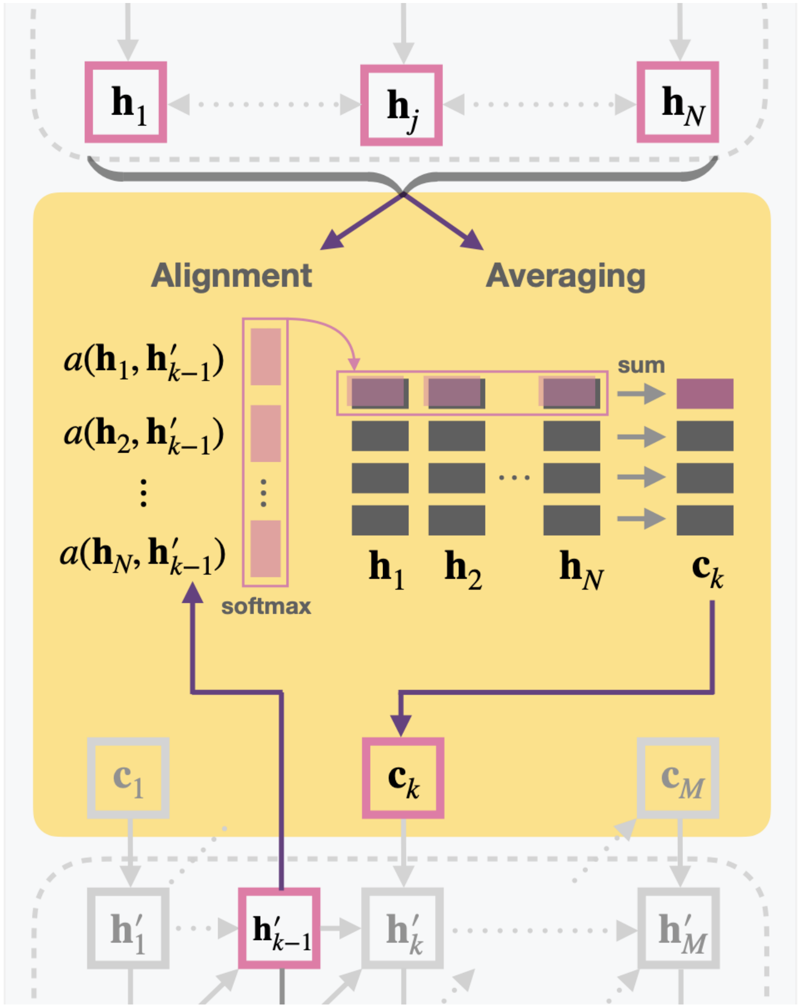
Cho et al. (2014): $\mathbf{h}'_k = f_{\text{dec}}(\mathbf{h}'_{k-1}, \mathbf{y}_{k-1}, \mathbf{c})$

Bahdanau et al. (2015): $\mathbf{h}'_k = f_{\text{dec}}(\mathbf{h}'_{k-1}, \mathbf{y}_{k-1}, \mathbf{c}_k)$ where $\mathbf{c}_k$ is a function of $\mathbf{h}'_{k-1}$ and all encoder states $\mathbf{h}_1, \ldots, \mathbf{h}_N$ (and the encoder is bidirectional).

BAHDANAU ET AL. (2015)

FIRST ENCODER-DECODER RNN WITH ATTENTION





BAHDANAU ET AL. (2015)
FIRST ENCODER-DECODER RNN WITH ATTENTION

At step k of decoding:

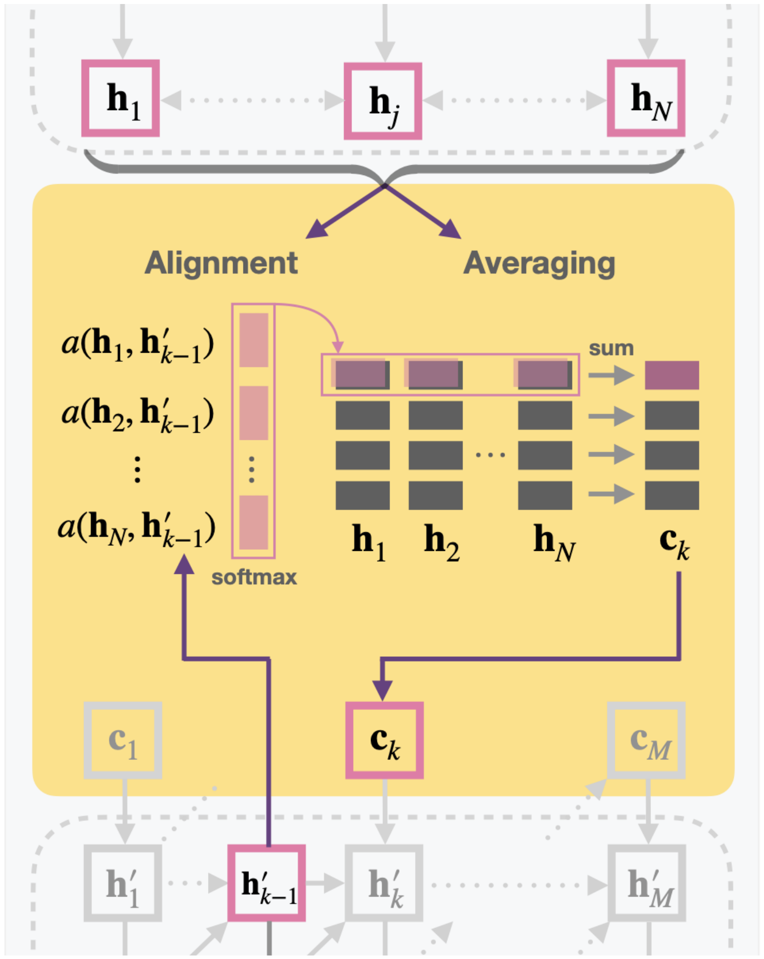
Input: previous decoder state $\mathbf{h}' := \mathbf{h}'_{k-1}$ and all encoder states $\mathbf{h}_1, \ldots, \mathbf{h}_N$

Output: context vector $\mathbf{c}_k$

There are two steps:

Alignment: For each i, calculate a similarity score between $\mathbf{h}_i$ and $\mathbf{h}'$ using an *alignment model* a, represented by a separate feed-forward neural network, and define $\alpha_i = \text{softmax}_i(a(\mathbf{h}_i, \mathbf{h}'))$

Averaging: $\mathbf{c}_k = \sum_i \alpha_i \mathbf{h}_i$



ATTENTION IN RNNS

To summarize the attention mechanism discussed so far:

- attention is a **soft selection mechanism**
- it uses a **similarity measure** to compare vectors of two sets, a set of **query vectors** (e.g. the decoder states $\mathbf{h}'$) and a set of **value vectors** (e.g. the encoder states $\mathbf{h}$)
- a given **query vector is compared to all value vectors** using the similarity measure
- the softmax of the result of this comparison is then used to take the **weighted average of the value vectors**

Extreme cases:

- If the two sets are identical, consisting of vectors $\mathbf{h}_1, \ldots, \mathbf{h}_N$, then for any reasonable similarity measure a and any fixed k, $\alpha_j = \mathrm{softmax}_j(a(\mathbf{h}_j, \mathbf{h}_k))$ will be peaked around $j = k$, so that the weighted average wrt. α will more or less retrieve $\mathbf{h}_k$.
- If the similarity measure a cannot detect differences between the two sets of vectors so that all $a(\mathbf{h}, \mathbf{h}')$ are the same, then the weighted average is the arithmetic mean.

REMAINING ISSUES

- **RNNs are slow**, since they need to process the input sequence **vector by vector**, in particular they cannot be parallelized well.
- In case of longer sequences, the discussed **attention mechanism is slow**: the calculation of the alignment scores

$$s_{ij} = a(\mathbf{h}_i, \mathbf{h}'_j) \qquad \forall i \in \{1, \ldots N\},\ \forall j \in \{1, \ldots, M\}$$

  requires $N \cdot M$ evaluations of the corresponding neural network.

$\Longrightarrow$ both of these issues are solved (partly) by the networks in the next section.

3. TRANSFORMERS

DOT-PRODUCT ATTENTION

Instead of evaluating the attention network $N \cdot M$ times to calculate the scores

$$s_{ij} = a(\mathbf{h}_i, \mathbf{h}'_j) \qquad \forall i \in \{1, \dots N\},\ \forall j \in \{1, \dots, M\},$$

in the attention mechanism popularized by Vaswani et al. (2017) this is simplified to

$$s_{ij} = f(\mathbf{h}_i) \cdot g(\mathbf{h}'_j) \qquad \forall i \in \{1, \dots N\},\ \forall j \in \{1, \dots, M\}$$

so that the number of evaluations reduces to $N + M$, followed by an inner product.

The functions f and g could, e.g., be two neural networks with the same output dimensions, or in the simplest case, taken to be identities $f = g = id$ (see e.g. Luong et al. (2015)).

In **Query-Key-Value attention** (Vaswani et al. (2017)), f and g are learnable linear maps.

QUERY-KEY-VALUE MECHANISM

Inputs: $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and $\mathbf{z}_1, \ldots, \mathbf{z}_M$

Outputs: $\mathbf{y}_1, \ldots, \mathbf{y}_M$

Learnable weight matrices: $\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v$

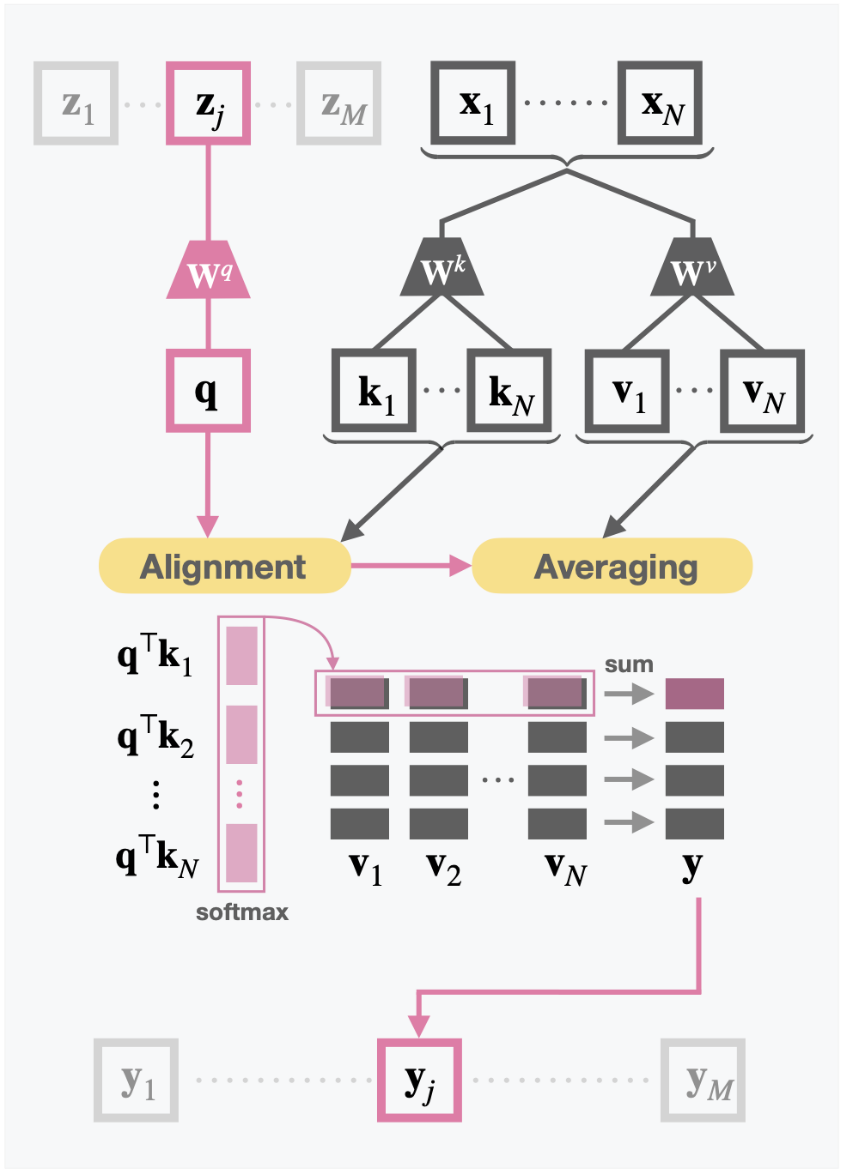
The vectors $\mathbf{x}_i$ are transformed to **keys** $\mathbf{k}_i := \mathbf{W}^k\mathbf{x}_i$ and **values** $\mathbf{v}_i := \mathbf{W}^v\mathbf{x}_i$, while $\mathbf{z}_j$ are transformed to **queries** $\mathbf{q}_j := \mathbf{W}^q\mathbf{z}_j$.

For each query $\mathbf{q} := \mathbf{q}_j \in \{\mathbf{q}_1, \ldots, \mathbf{q}_M\}$,

- calculate the **similarity scores** $s_i = \mathbf{q}^\top\mathbf{k}_i$
- and take the **weighted average** of the values $\mathbf{v}_i$ with respect to

$$\alpha_i := \mathrm{softmax}_i(s_i)$$

resulting in $\mathbf{y}_j := \sum_i \alpha_i \mathbf{v}_i$.

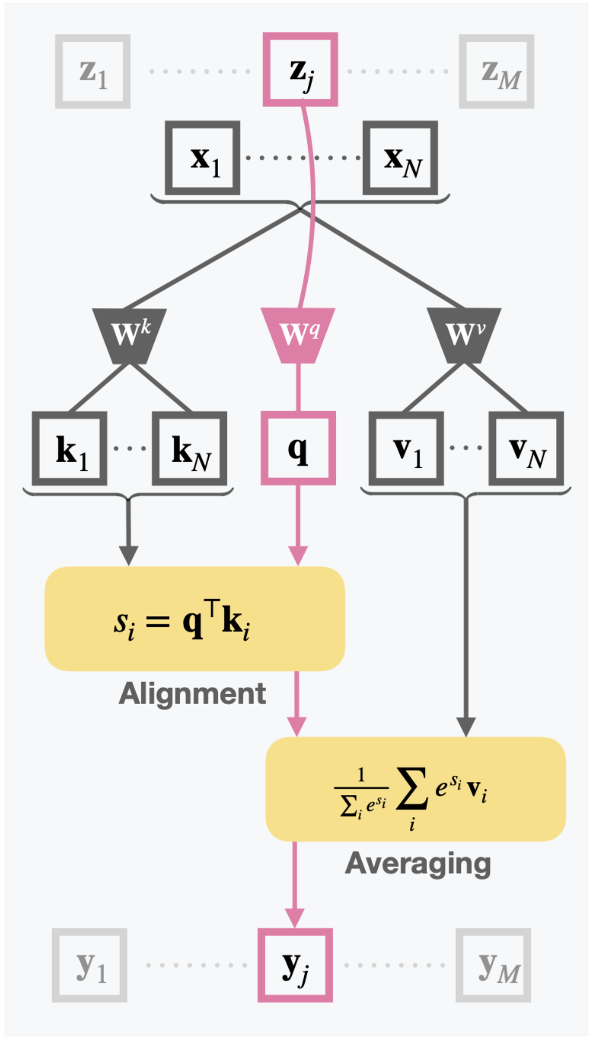


QUERY-KEY-VALUE MECHANISM

COMMENTS

- Often, we assume that $\mathbf{W}^k = \mathbf{W}^v$, so that the keys and values are identical sets.
- For NLP, the input vectors are **learned word embeddings** (using available algorithms such as word2vec), usually high-dimensional, e.g., $d_{\text{embedding}} = 512$.
- The target dimension of $\mathbf{W}^q$, $\mathbf{W}^k$ and $\mathbf{W}^v$ is usually much smaller, so that the corresponding mappings can be thought of "projections" to a lower dimensional feature space.
- There are usually **multiple attention heads**, copies of the attention mechanism with different weights, focussing on **different features** (total output = concat(head outputs)).

For example, one head for semantic features (e.g. words that determine the topic) and one for grammatical features (e.g. subject).



VASWANI ET AL. (2017)

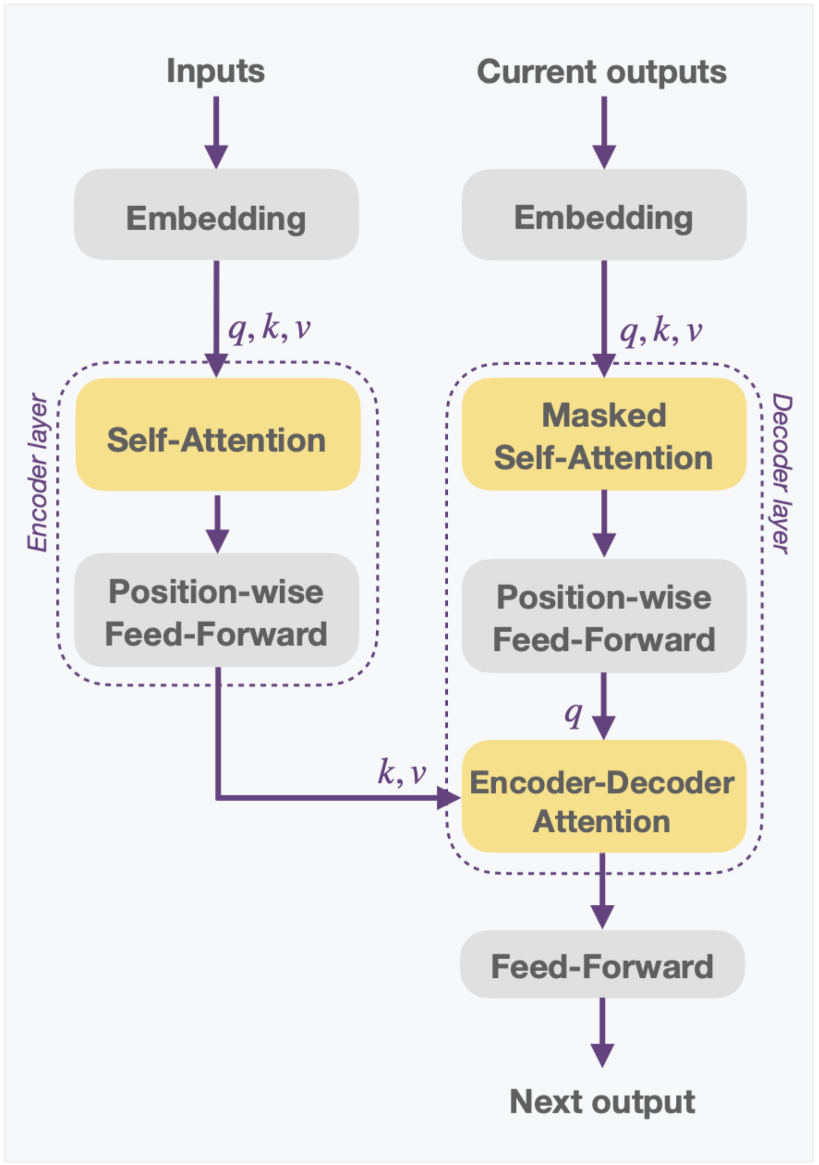
ATTENTION IS ALL YOU NEED

The basic transformer architecture contains two different types of query-key-value attention:

- **Self-Attention** and **Masked Self-Attention**
- **Encoder-Decoder (Cross-)Attention**

which differ in how the two input sequences $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and $(\mathbf{z}_1, \ldots, \mathbf{z}_M)$ are chosen.

Note: There are often **multiple encoder and decoder layers**, as well as **multiple attention heads** per layer.



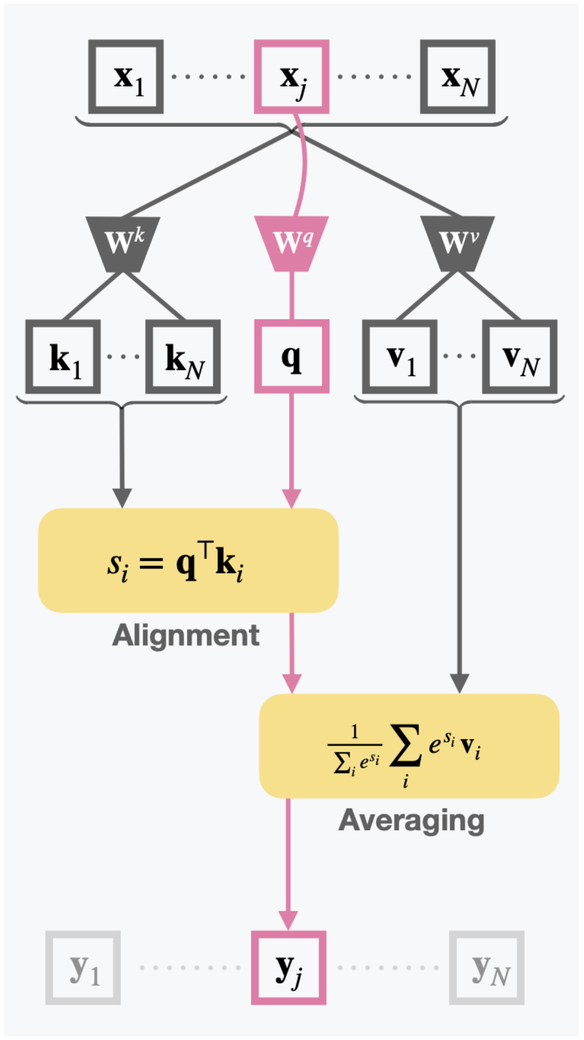
SELF-ATTENTION

Query-Key-Value attention with $\mathbf{z}_j = \mathbf{x}_j \; \forall j$.

That is, all three components, queries, keys, and values, result from the same input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ through the transformations $\mathbf{W}^q$, $\mathbf{W}^k$, and $\mathbf{W}^v$.

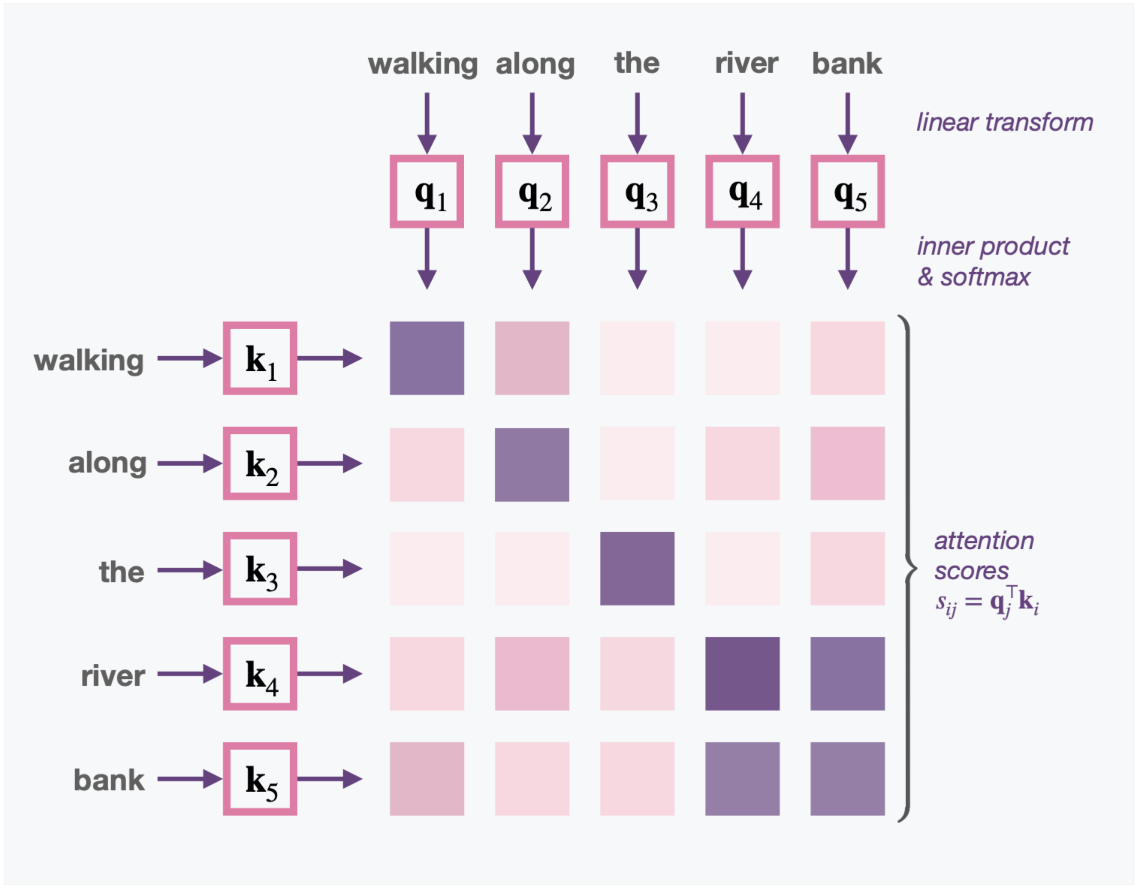
- Each (transformed) input vector can attend to any of the other input vectors to build a **combined representation** or an **abstraction**.
- In a multihead self-attention block, each head can focus on forming abstractions of **different features**.
- Using **multiple layers** of self-attention blocks, abstractions can attend to other abstractions.

Note: This is similar to convolutional neural networks but without being restricted to the local neighbourhood!



SELF-ATTENTION

EXAMPLE



An exemplary attention head where "river bank" forms a **combined representation**

ENCODER-DECODER ATTENTION

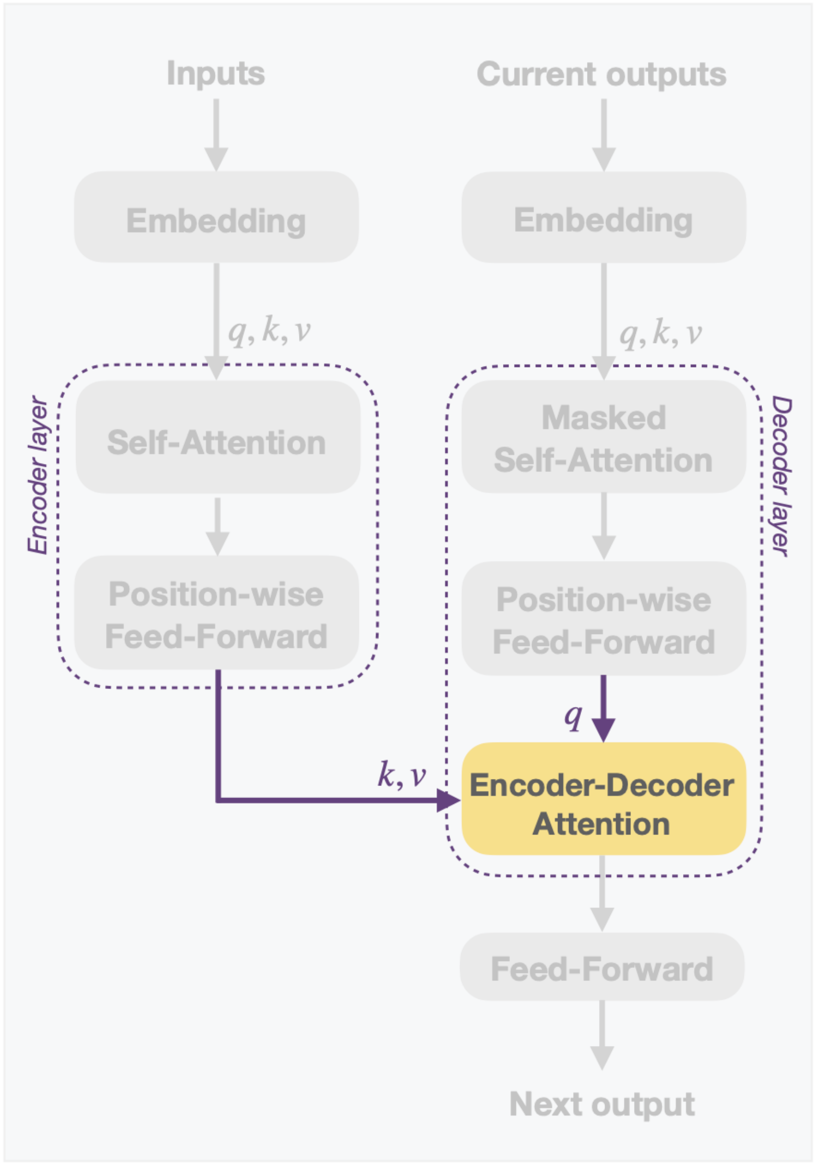
Query-Key-Value attention where

- **keys and values** come from the final output of the encoder
- **queries** originate from the self-attention output of the decoder.

This way, the decoder can attend to the outputs of the encoder.

Again, multiple attention heads allow to consider different feature spaces.

Note: This is the only connection between encoder and decoder ("Attention is all you need!").



MASKED SELF-ATTENTION

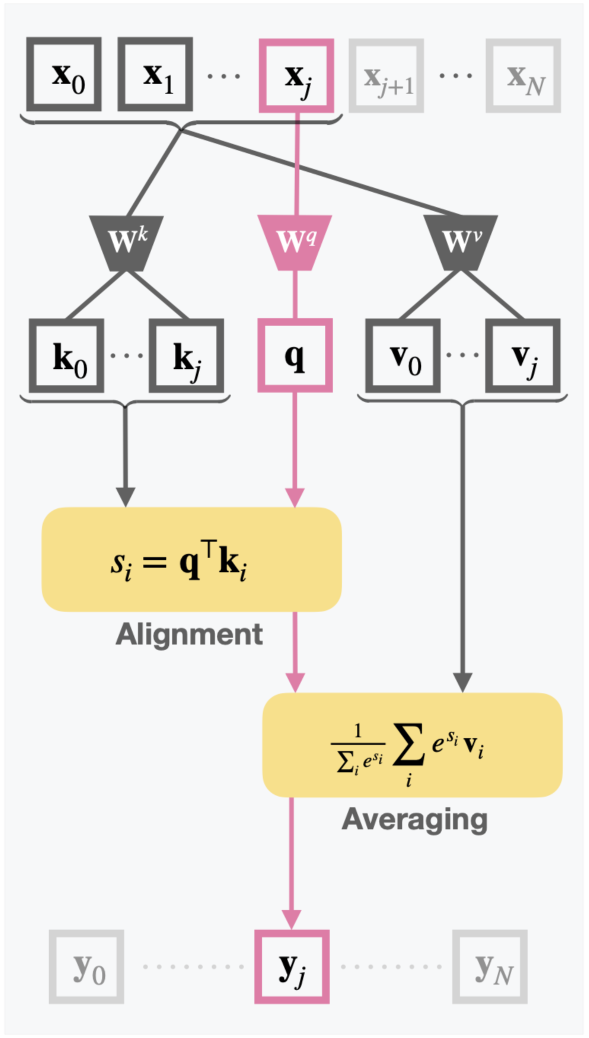
Self-Attention where **each word can only attend to words in the past**.

In particular,

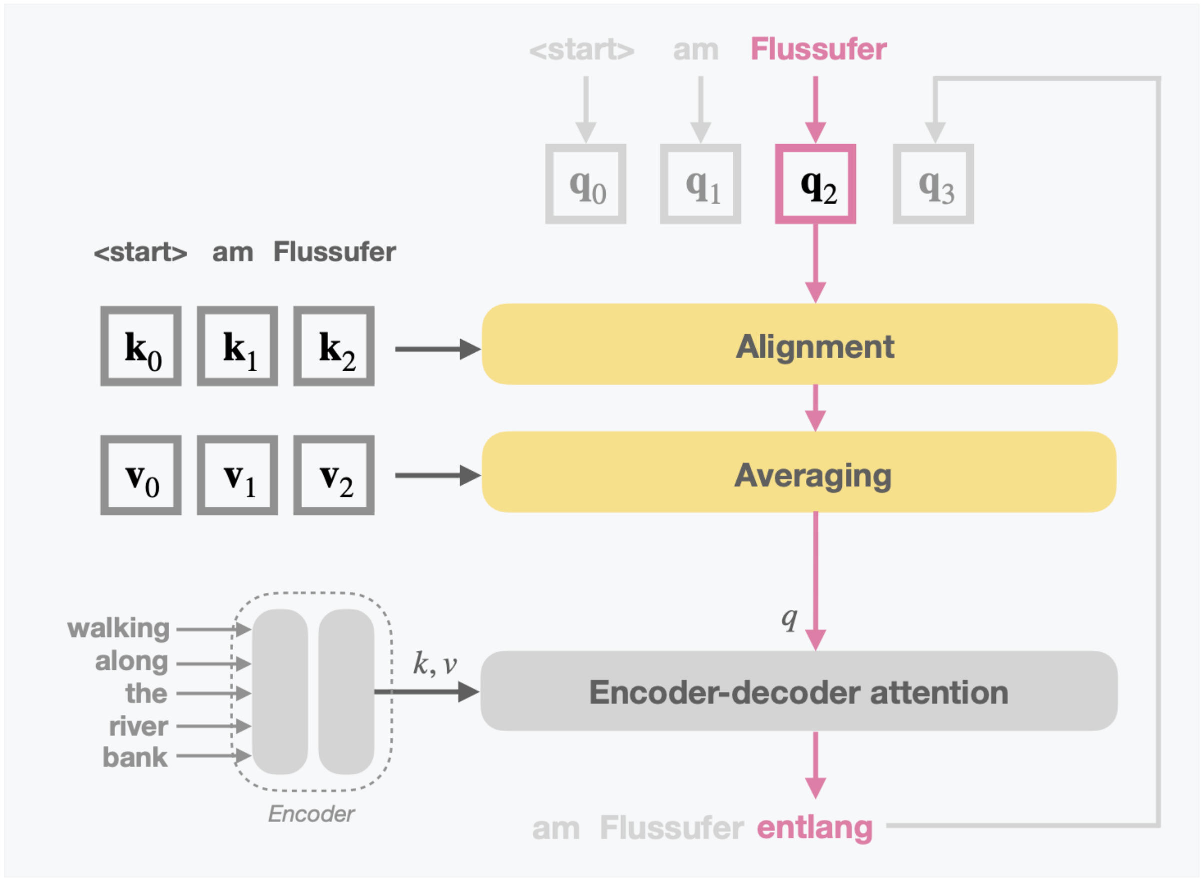
- positions are offset by one entry
- the attention scores s_{ij} are masked so that positions are not allowed to attend to subsequent positions

Combined, this means that predictions for position k can only depend on known outputs at positions less than k

Note: Without the masking, the decoder self-attention would simply look ahead to the next symbol during training and be useless for prediction.



MASKED SELF-ATTENTION
PREDICTION



Third step of translating "walking along the river bank" to German. The **encoder is only executed once**, reading the input sequence in parallel, while the **decoder outputs word by word**.

WILLIAMS, ZIPSER (1989)

TEACHER FORCING IN RNNS

In short: **Use ground truth as input** instead of the model output of the previous time step.

A strategy already used to train RNNs as an alternative to backpropagation through time.

Instead of using the previous-output-as-input scheme of RNNs during training, one uses the actual training data as input to the next state (and not only to define the error at the end).

$\Rightarrow$ Learning is more stable, since the model **is not punished for consequential errors** (only once for each error).

$\Rightarrow$ Training can be **parallelized over the sequence**, since we do not have to wait for the output of the previous step before computing the current step.

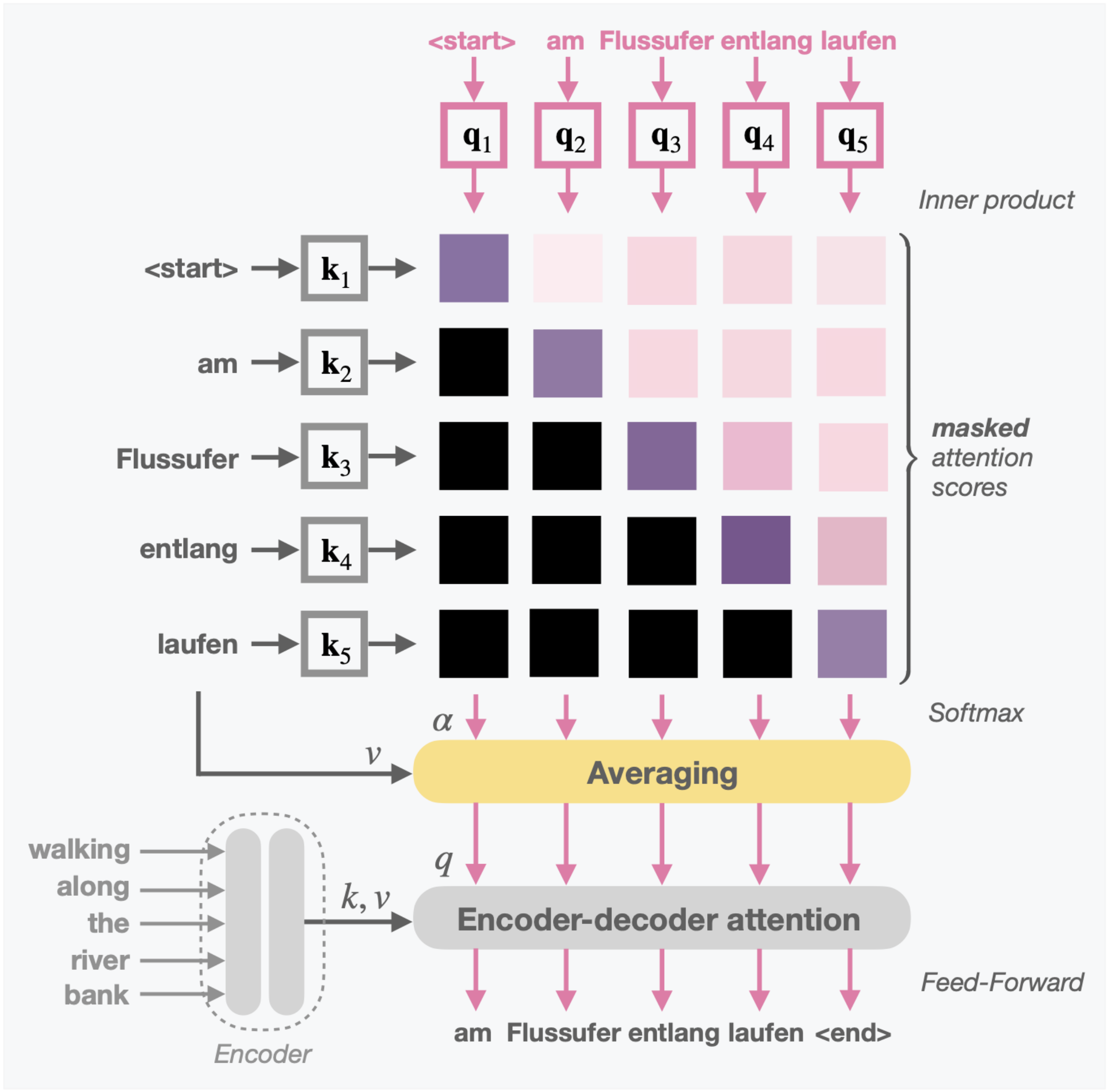
Analogy: Learning a song by singing while listening to it (instead of singing it from memory).

MASKED SELF-ATTENTION

TRAINING

Using **teacher forcing**, the decoder can be **parallelized during learning** like the encoder:

- Regular self-attention on shifted output sequence.
- Masking the upper diagonal of the attention scores with -10^9 before the softmax is applied.



SOME ADDITIONAL DETAILS

There are some components of the Transformer architecture that we did not discuss so far:

Scaled dot-product: Divide $\mathbf{q}^\top \mathbf{k}$ by $\sqrt{d}$ where $d =$ dimension of keys and queries.

Positional encoding: There is no learnable positional information in the attention mechanism discussed so far. Thus one has to add some type of positional encoding to the embeddings, e.g. sine and cosine with frequencies that depend on the position.

Residual connections: There are residual connections between inputs and outputs of each attention layer.

Layer Normalization (Lei Ba, Kiros, Hinton (2016)): Normalize over the units in a layer by shifting and rescaling.

Final softmax layer: The final layer of the Transformer architecture consists of a fully connected layer followed by a **softmax over all words** in the respective vocabulary.

MACHINE TRANSLATION RESULTS

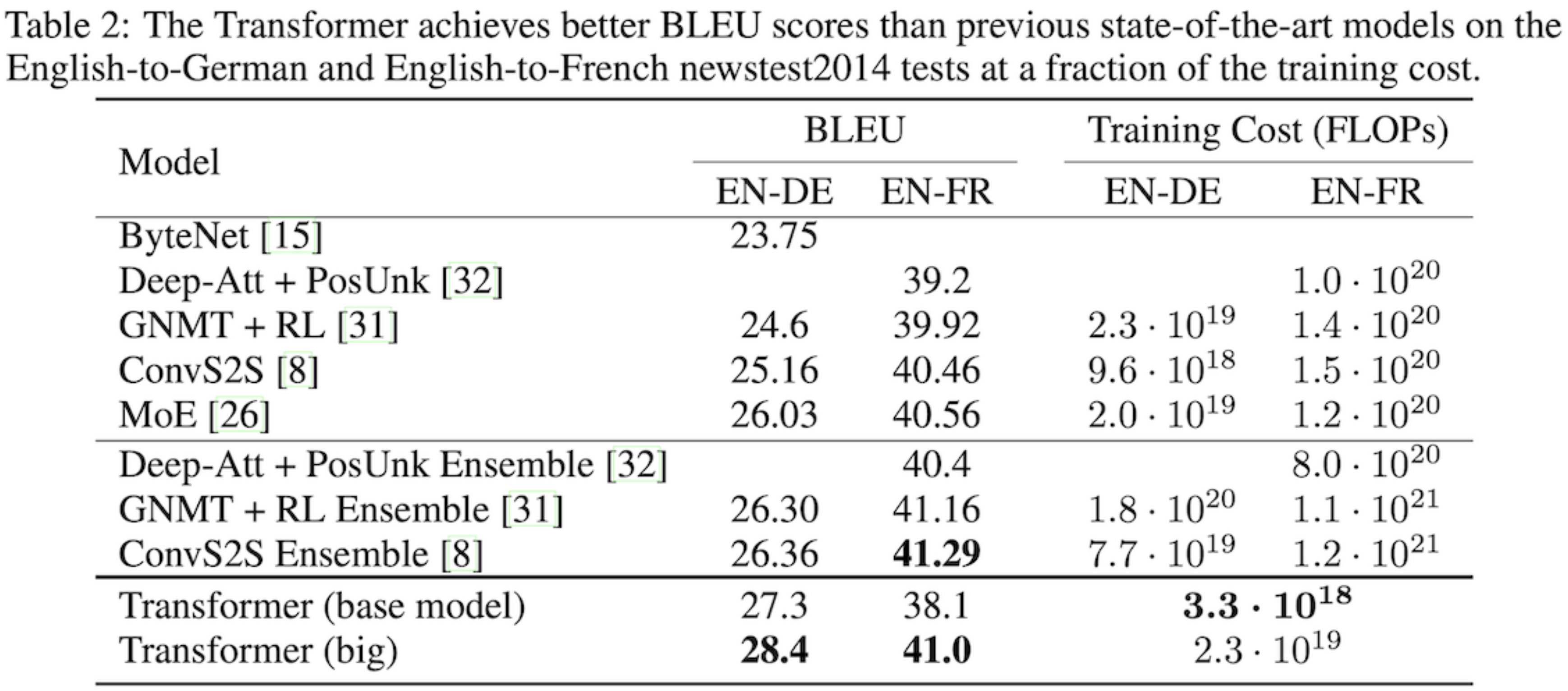


Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Image taken from Vaswani et al. (2017) (human translators reach BLEU scores of ~30 for EN-DE)

APPLICATIONS TO OTHER TASKS

VISION

- Mixing CNNs with self-attention: Wang et al. (2018), Carion et al. (2020)
- **Vision Transformer** (ViT): Dosovitskiy et al. (2020)
- **Perceiver**: Jaegle et al. (2021)

...and many more

FULL LANGUAGE MODELS

- **BERT**: Devlin et al. (2019) (110 million parameters)
- **GPT-3**: Brown et al. (2020) (31 authors, 175 billion parameters)

REINFORCEMENT LEARNING

- **Decision transformer**: Chen et al. (2021)

PERFORMANCE ISSUE OF TRANSFORMERS

Transformers are a large improvement over traditional RNNs, in particular due to the parallelization of learning and the ability to efficiently attend to arbitrary elements of a sequence.

However, in the standard transformer, a **very large sequence length** N is still a problem (e.g. training a network to summarize articles, or when applied to vision). The culprit is the attention mechanism:

- Original attention in RNNs: calculate $a(\mathbf{h}_i, \mathbf{h}_j)\ \forall i, j = 1, \dots, N$, where a is an ANN
- Query-Key-Value attention: calculate $\mathbf{q}_j^\top \mathbf{k}_i\ \forall i, j = 1, \dots, N$,

In most cases, Query-Key-Value attention is much faster than the original approach since the evaluation of a neural network is replaced by the linear operations $\mathbf{W}^k\mathbf{x}$, $\mathbf{W}^q\mathbf{x}$ and the inner product.

But the issue is that every element is still allowed to attend to any other element, which is an **everyone-with-everyone** operation that scales **quadratically with sequence length**.

RECENT PROPOSALS TO DEAL WITH QUADRATIC COSTS

- **Reformer (2020)**: More efficient transformers using several tricks to achieve $\mathcal{O}(N \log N)$.
- **LINformer (2020)**: Linear complexity by projecting keys and values to lower dimensions.
- **Performer (2020)**: Linearize and approximate the softmax of the attention scores using kernels. Beats the LINformer and Reformer while still scaling linearly.
- **Perceiver (2021)**: Linear complexity by using cross-attention where queries come from a latent array of a fixed size, resulting in a fixed bottleneck independent of input size.
- **Shared Workspace (2021)**: Use two attention mechanisms to communicate through a shared workspace, also creating a bottleneck.

Even more recently, models got rid of the attention mechanism as well, only keeping the mixing of tokens and were shown to be similarly successful:

- **MLP-Mixer (2021)**: There are two type of MLP-layers, one applied to image patches (mixing per-location features), and one applied accross patches (mixing spatial information).
- **FNET (2021)**: Replace self-attention by a linear mixing algorithm using Fourier transforms. Achieves 92-97% accuracy, but is 70-80% faster.

THANK YOU FOR YOUR *ATTENTION*!