CSC413/2516 Lecture 8: Attention and Transformers

Jimmy Ba

Overview

- We have seen a few RNN-based sequence prediction models.
- It is still challenging to generate long sequences, when the decoders only has access to the final hidden states from the encoder.
  - Machine translation: it's hard to summarize long sentences in a single vector, so let's allow the decoder peek at the input.
  - Vision: have a network glance at one part of an image at a time, so that we can understand what information it's using
- This lecture will introduce attention that drastically improves the performance on the long sequences.
- We can also use attention to build differentiable computers (e.g. Neural Turing Machines)

Overview

- Attention-based models scale very well with the amount of training data. After 40GB text from reddit, the model generates:

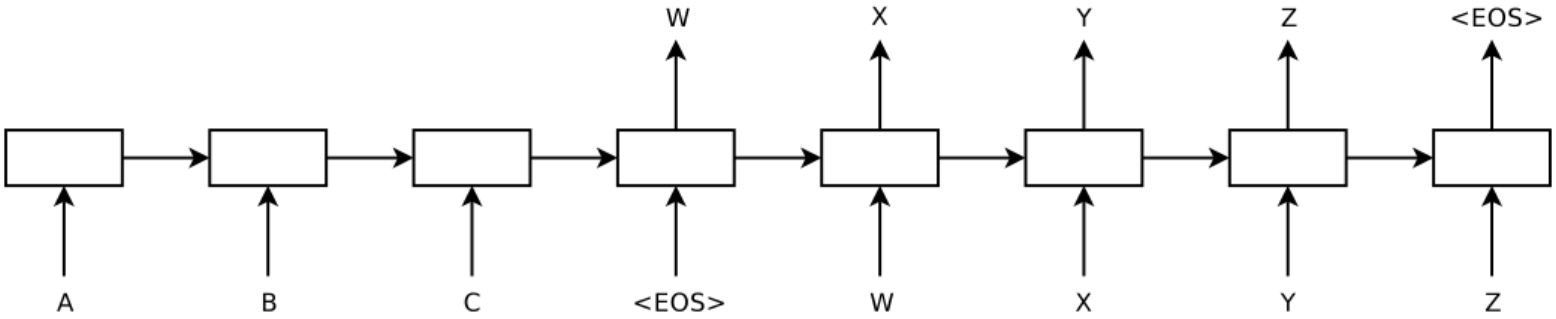
Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow. Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez. Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

For the full text samples see Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners." 2019.

https://talktotransformer.com/

Attention-Based Machine Translation

- Remember the encoder/decoder architecture for machine translation:



- The network reads a sentence and stores all the information in its hidden units.
- Some sentences can be really long. Can we really store all the information in a vector of hidden units?
  - Let's make things easier by letting the decoder refer to the input sentence.

Attention-Based Machine Translation

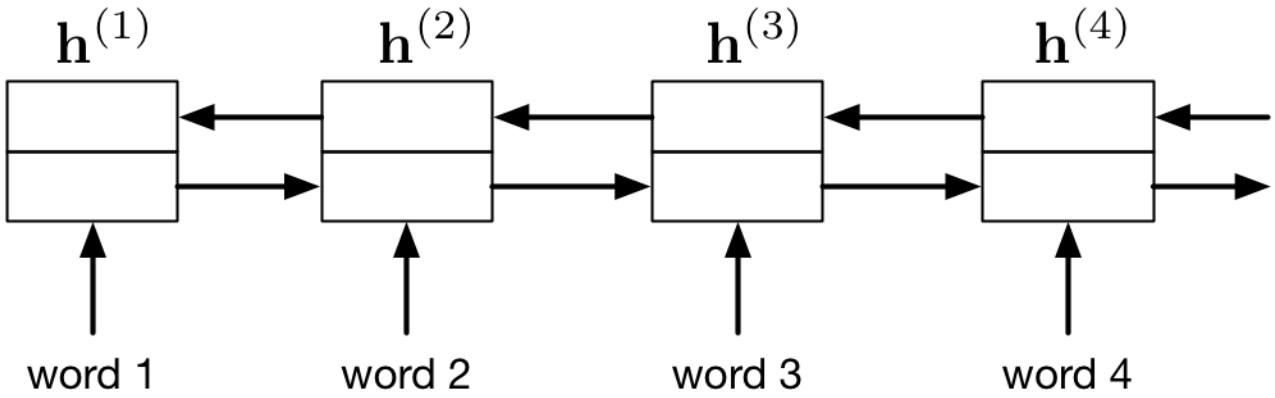
- We'll look at the translation model from the classic paper:

  Bahdanau et al., Neural machine translation by jointly learning to align and translate. ICLR, 2015.

- Basic idea: each output word comes from one word, or a handful of words, from the input. Maybe we can learn to attend to only the relevant ones as we produce the output.

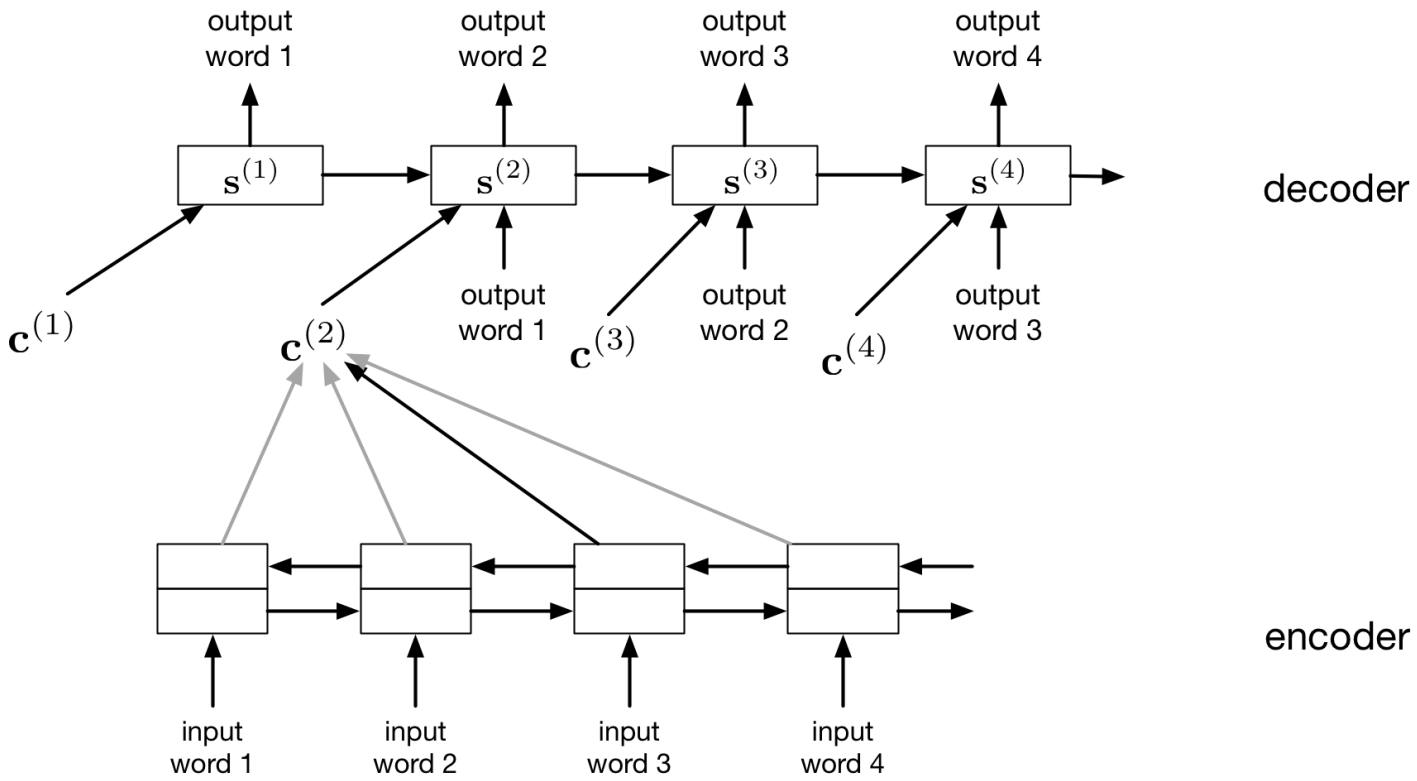
Attention-Based Machine Translation

- The model has both an encoder and a decoder. The encoder computes an annotation of each word in the input.
- It takes the form of a bidirectional RNN. This just means we have an RNN that runs forwards and an RNN that runs backwards, and we concantenate their hidden vectors.
  - The idea: information earlier or later in the sentence can help disambiguate a word, so we need both directions.
  - The RNN uses an LSTM-like architecture called gated recurrent units.



Attention-Based Machine Translation

- The decoder network is also an RNN. Like the encoder/decoder translation model, it makes predictions one word at a time, and its predictions are fed back in as inputs.
- The difference is that it also receives a context vector $\mathbf{c}^{(t)}$ at each time step, which is computed by attending to the inputs.



Attention-Based Machine Translation

- The context vector is computed as a weighted average of the encoder's annotations.

$$\mathbf{c}^{(i)} = \sum_j \alpha_{ij} h^{(j)}$$

- The attention weights are computed as a softmax, where the inputs depend on the annotation and the decoder's state:

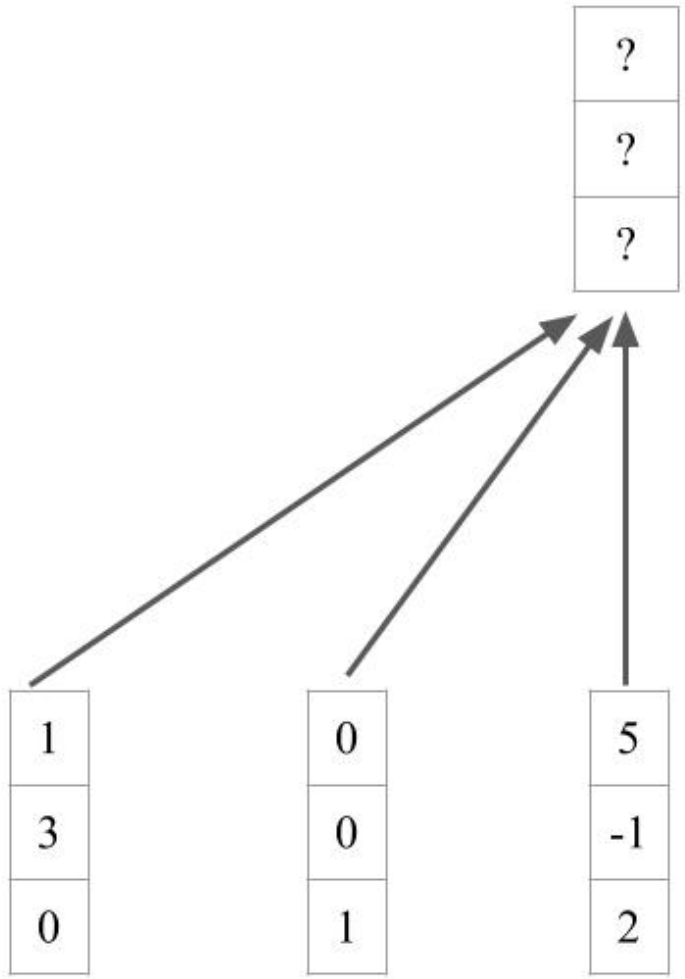
$$\alpha_{ij} = \frac{\exp(\tilde{\alpha}_{ij})}{\sum_{j'} \exp(\tilde{\alpha}_{ij'})}$$

$$\tilde{\alpha}_{ij} = f(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

- Note that the attention function, f depends on the annotation vector, rather than the position in the sentence. This means it's a form of content-based addressing.
  - My language model tells me the next word should be an adjective. Find me an adjective in the input.

Example: Pooling

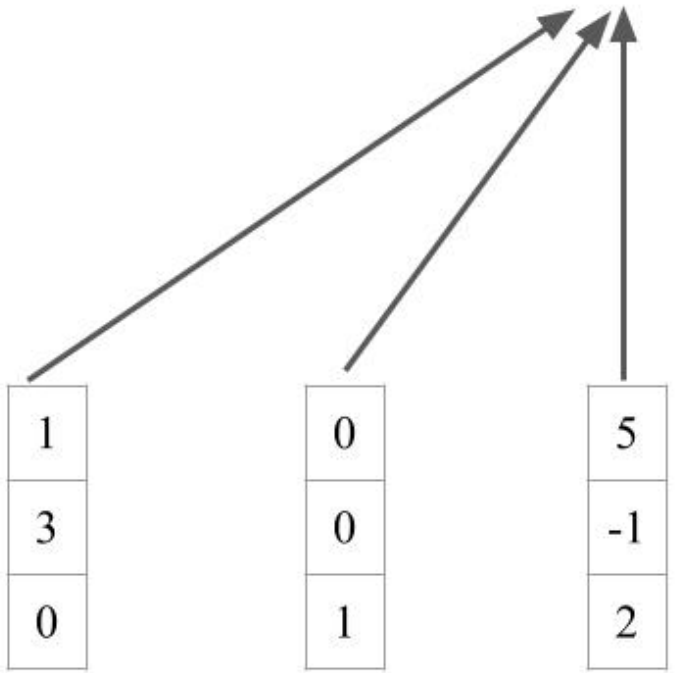
Consider obtain a context vector from a set of annotations.



Example: Pooling

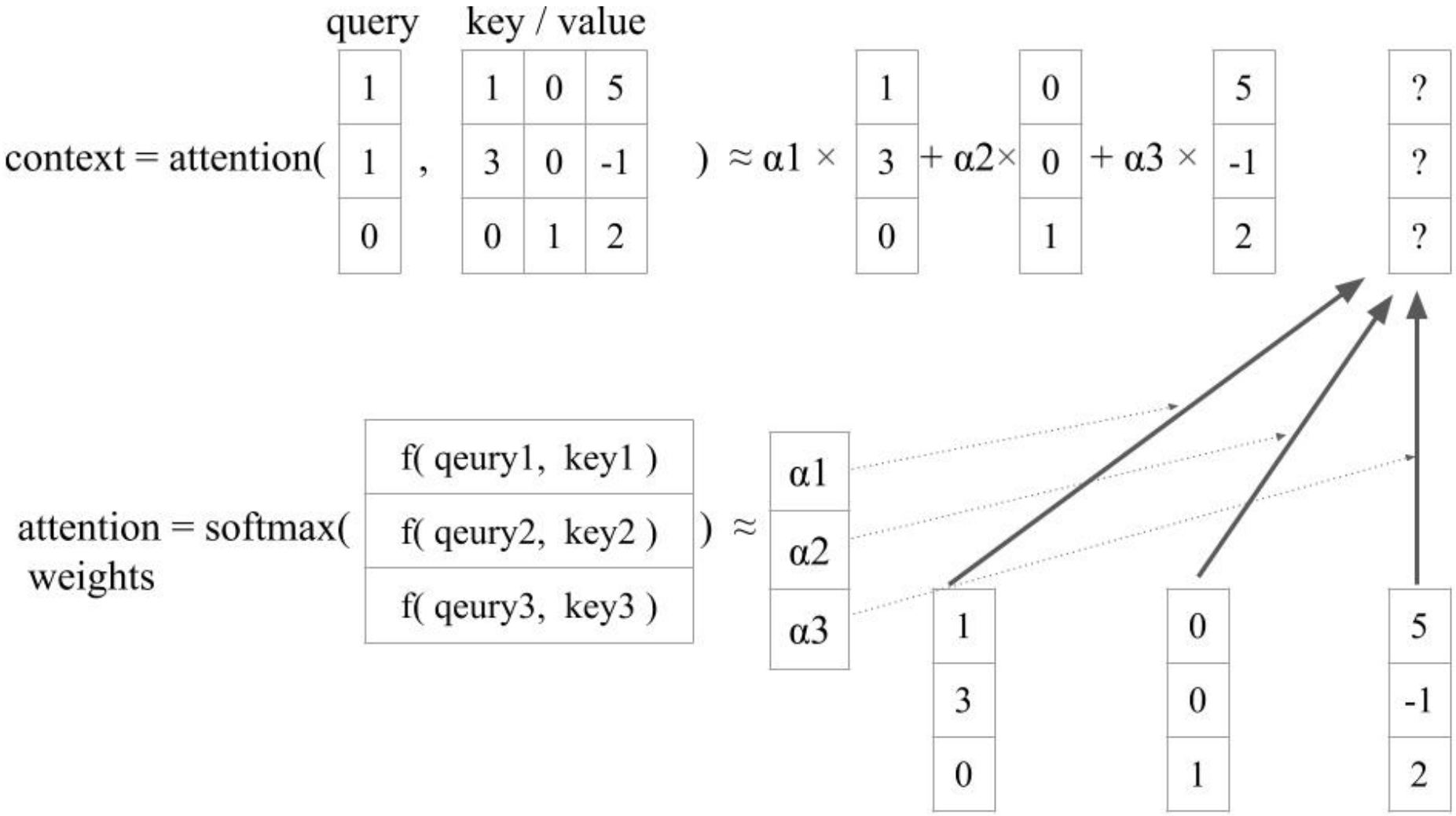
We can use average pooling but it is content independent.

$$\text{context} = \text{avg-pooling}\left(\begin{bmatrix}1 & 0 & 5\\ 3 & 0 & -1\\ 0 & 1 & 2\end{bmatrix}\right) = 0.33\times\begin{bmatrix}1\\3\\0\end{bmatrix} + 0.33\times\begin{bmatrix}0\\0\\1\end{bmatrix} + 0.33\times\begin{bmatrix}5\\-1\\2\end{bmatrix} = \begin{bmatrix}2\\0.6\\1\end{bmatrix}$$



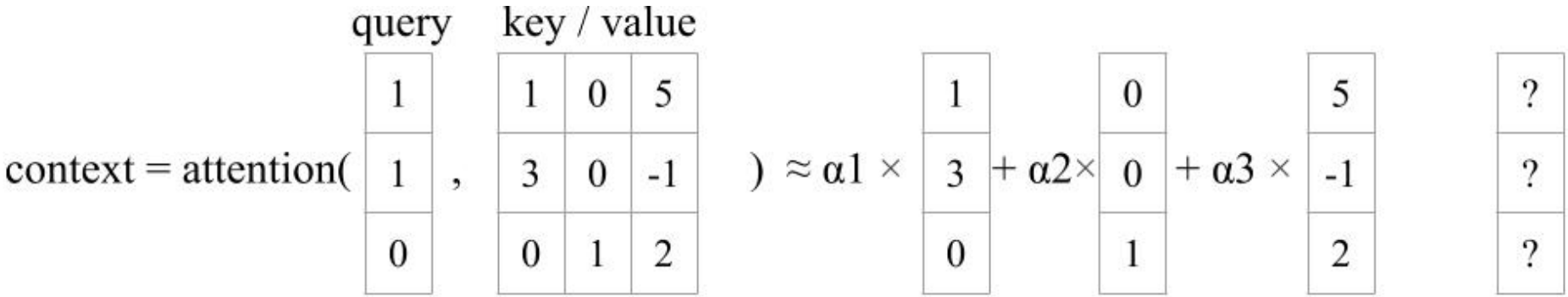
Example1: Bahdanau's Attention

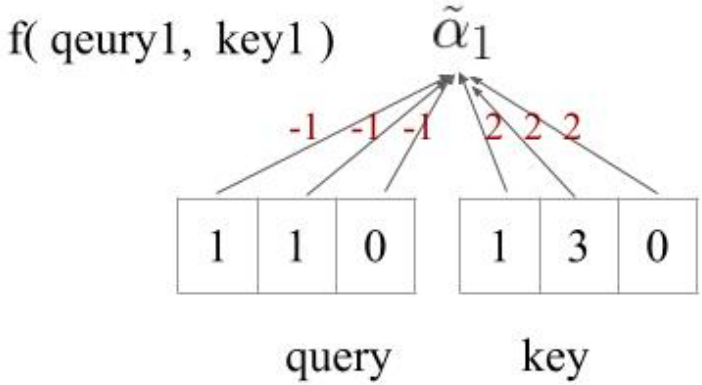
Content-based addressing/lookup using attention.

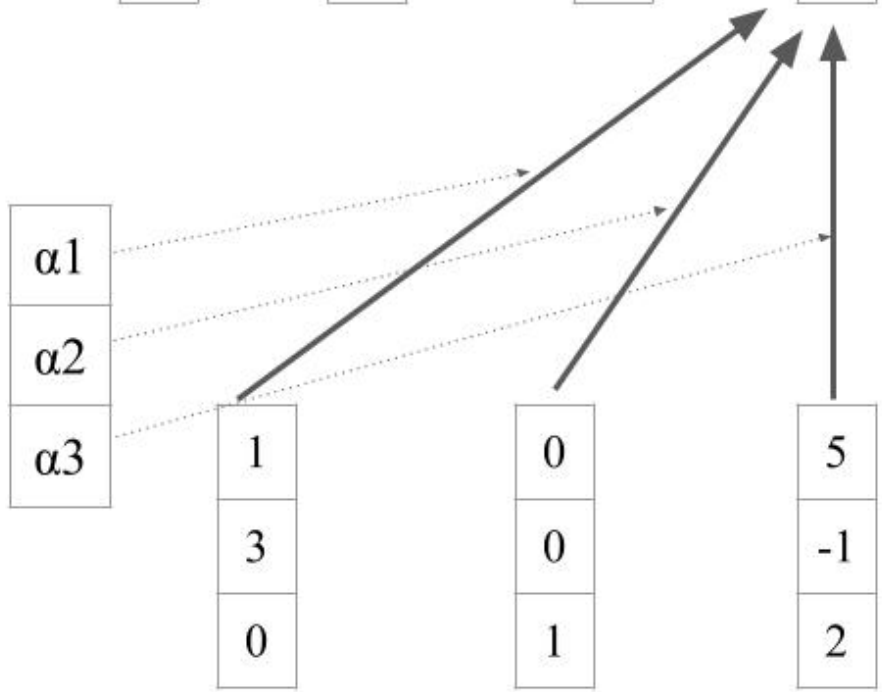


Example1: Bahdanau's Attention

Consider a linear attention function, f.

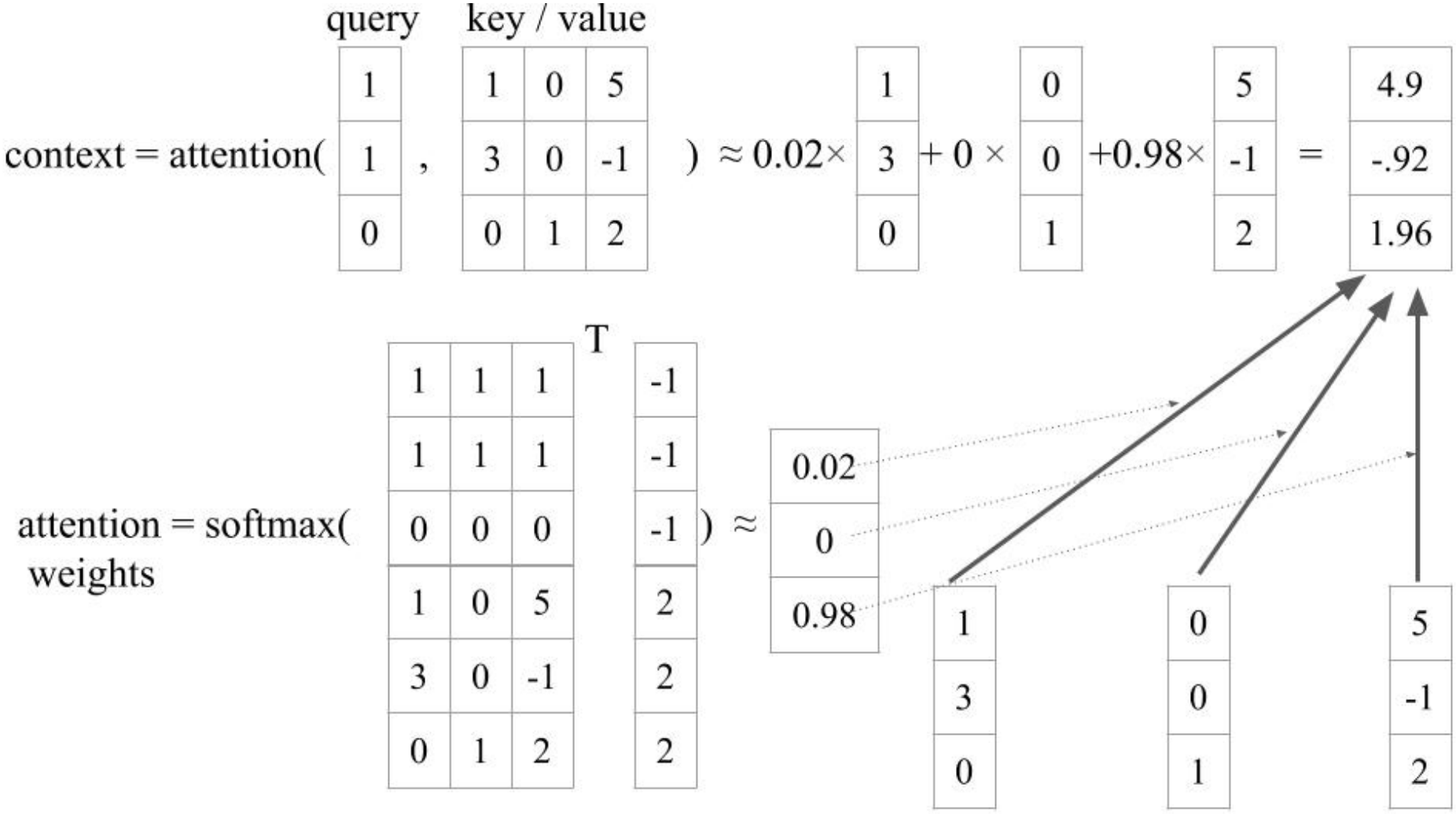






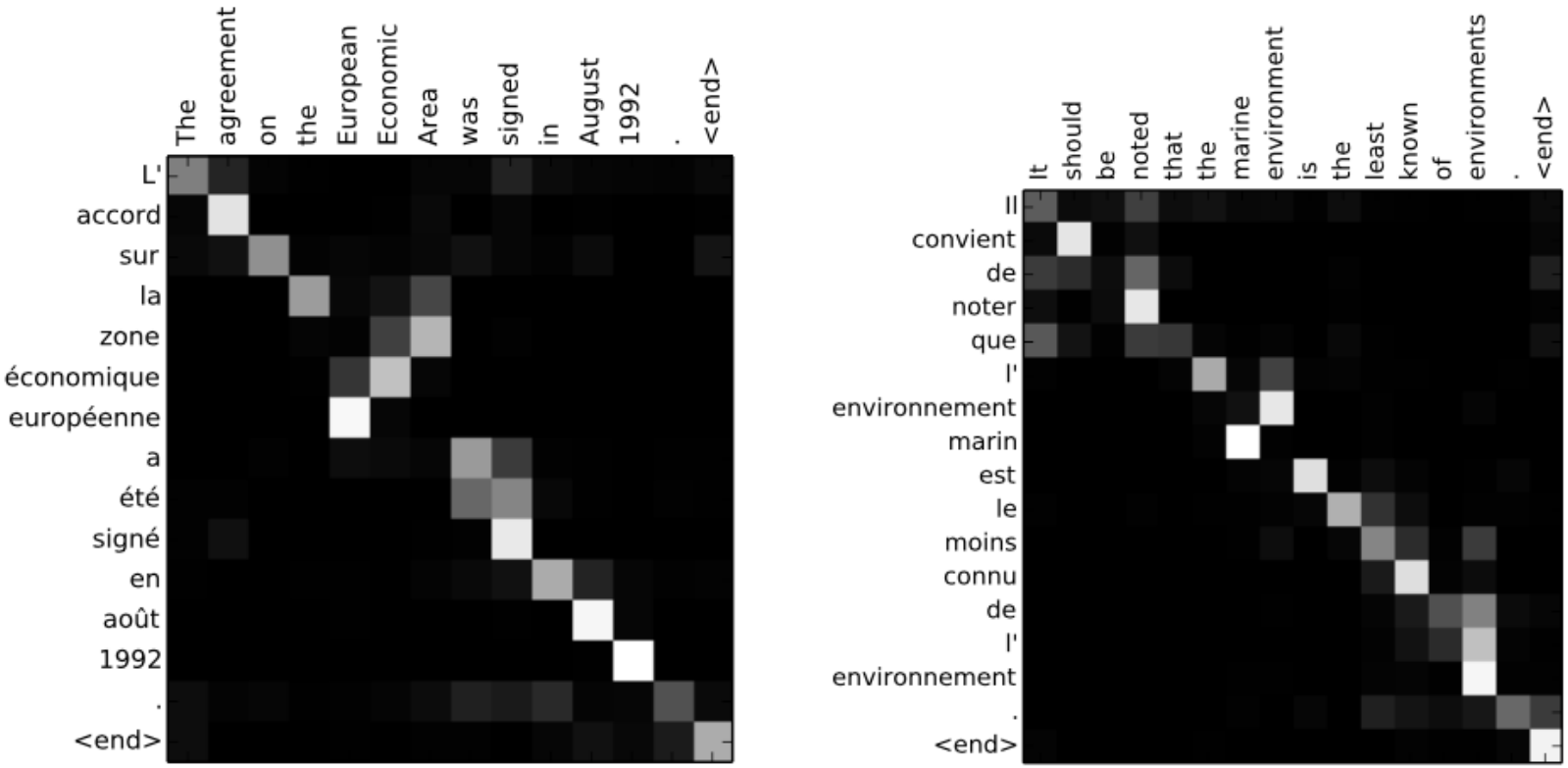
Example1: Bahdanau's attention

Vectorized linear attention function.



Attention-Based Machine Translation

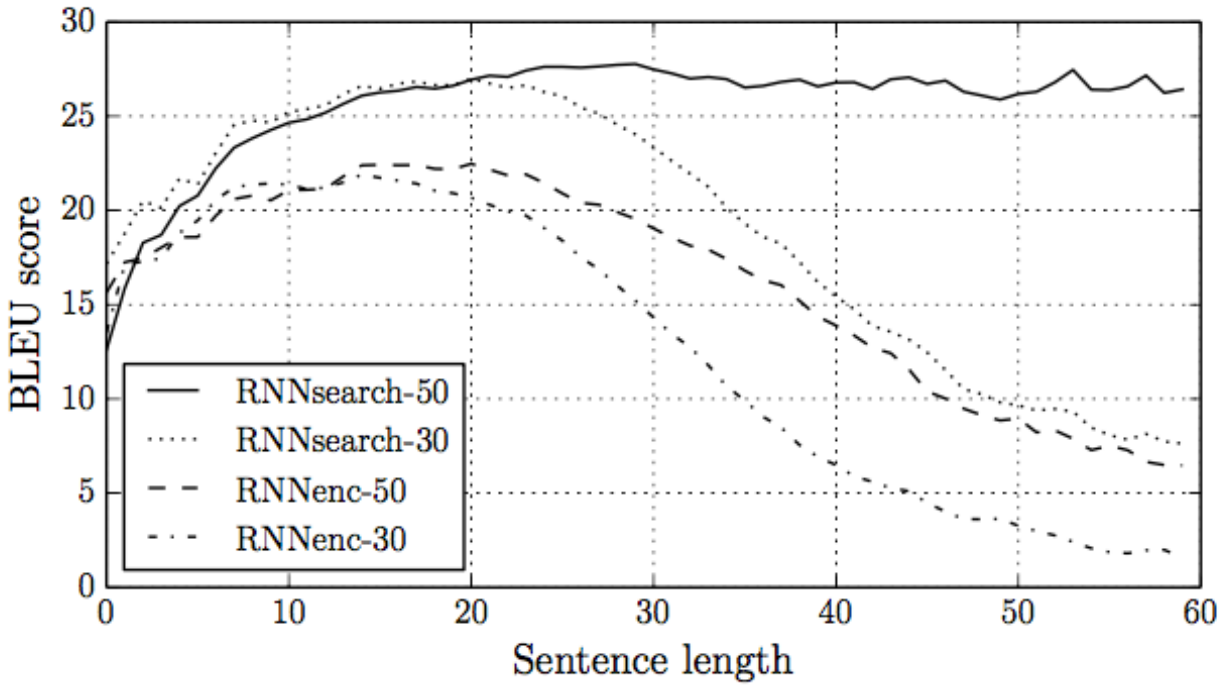
- Here's a visualization of the attention maps at each time step.



- Nothing forces the model to go linearly through the input sentence, but somehow it learns to do it.
  - It's not perfectly linear — e.g., French adjectives can come after the nouns.

Attention-Based Machine Translation

- The attention-based translation model does much better than the encoder/decoder model on long sentences.



Attention-Based Caption Generation

- Attention can also be used to understand images.
- We humans can't process a whole visual scene at once.
  - The fovea of the eye gives us high-acuity vision in only a tiny region of our field of view.
  - Instead, we must integrate information from a series of glimpses.
- The next few slides are based on this paper from the UofT machine learning group:

  Xu et al. Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

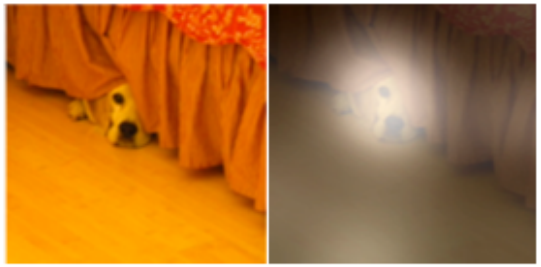
Attention-Based Caption Generation

- The caption generation task: take an image as input, and produce a sentence describing the image.
- **Encoder:** a classification conv net (VGGNet, similar to AlexNet). This computes a bunch of feature maps over the image.
- **Decoder:** an attention-based RNN, analogous to the decoder in the translation model
  - In each time step, the decoder computes an attention map over the entire image, effectively deciding which regions to focus on.
  - It receives a context vector, which is the weighted average of the conv net features.

Attention-Based Caption Generation

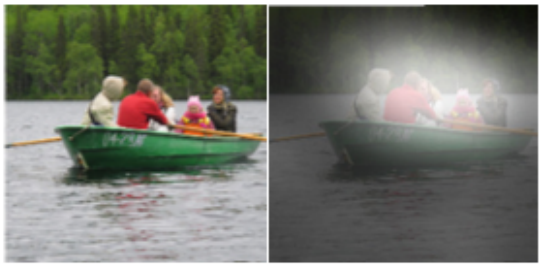
- This lets us understand where the network is looking as it generates a sentence.


A woman is throwing a frisbee in a park.


A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.


A little girl sitting on a bed with a teddy bear.


A group of people sitting on a boat in the water.


A giraffe standing in a forest with trees in the background.

Attention-Based Caption Generation

- This can also help us understand the network's mistakes.

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

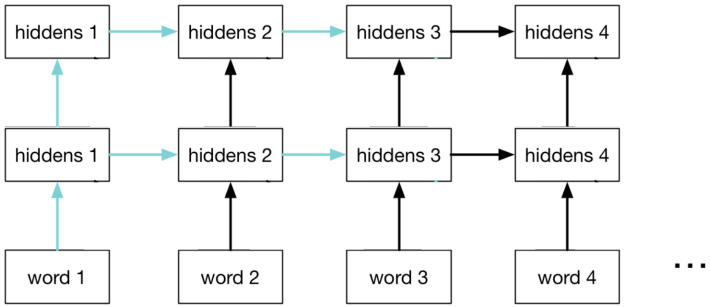
A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

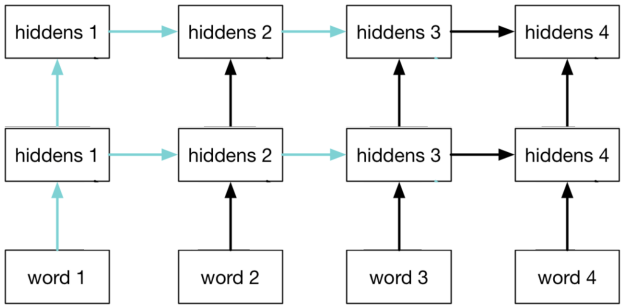
Attention is All You Need (Transformers)

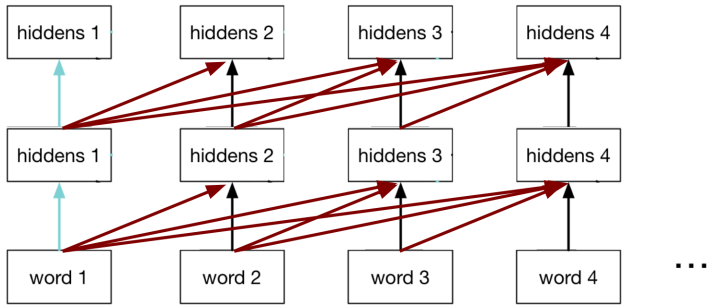
- We would like our model to have access to the entire history at the hidden layers.
- Previously we achieve this by having the recurrent connections.



Attention is All You Need (Transformers)

- We would like our model to have access to the entire history at the hidden layers.
- Previously we achieve this by having the recurrent connections.





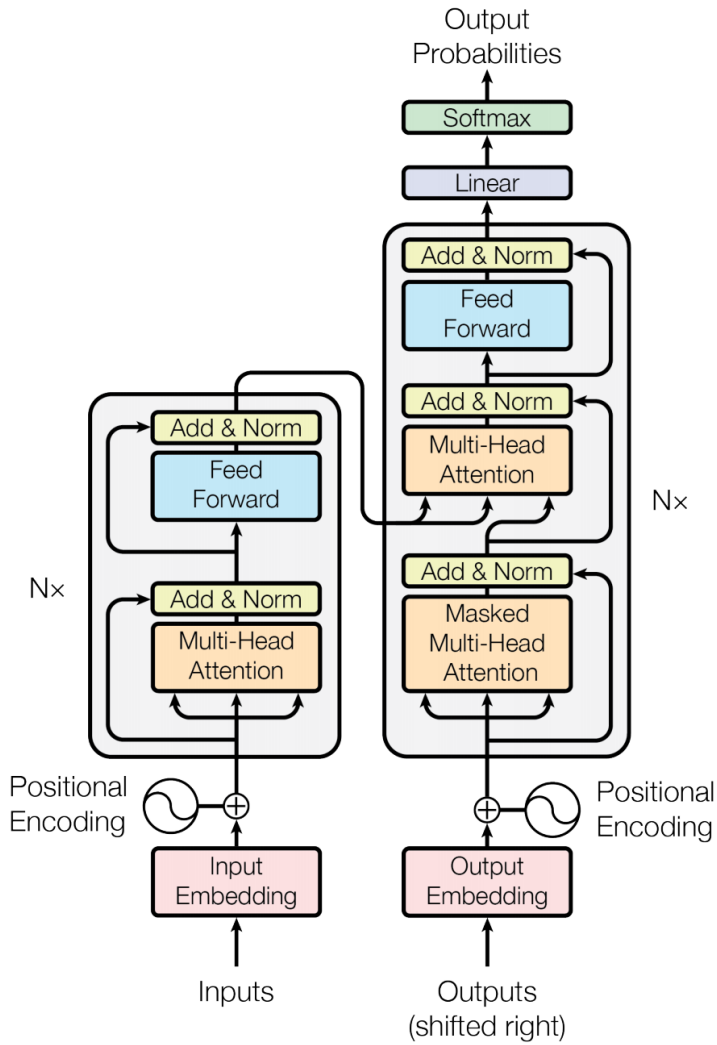
- **Core idea**: use attention to aggregate the context information by attending to one or a few important inputs from the past history.

Attention is All You Need

- We will now study a very successful neural network architecture for machine translation in the last few years:

  Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

- "Transformer" has a encoder-decoder architecture similar to the previous sequence-to-sequence RNN models.
  - except all the recurrent connections are replaced by the attention modules.



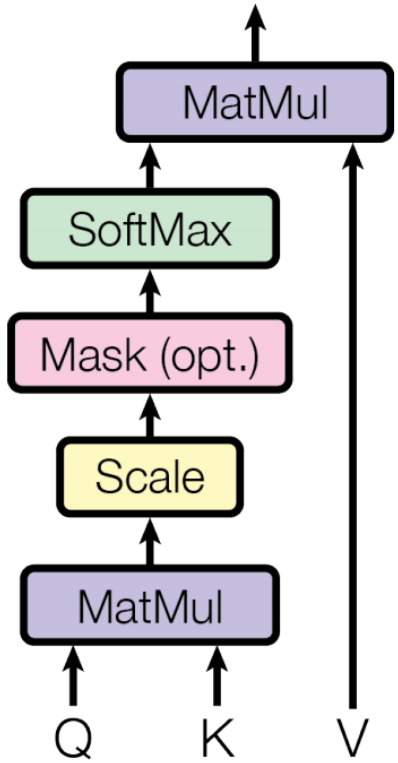
Attention is All You Need

- In general, Attention mappings can be described as a function of a query and a set of key-value pairs.
- Transformers use a "Scaled Dot-Product Attention" to obtain the context vector:

$$\mathbf{c}^{(t)} = \text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V,$$

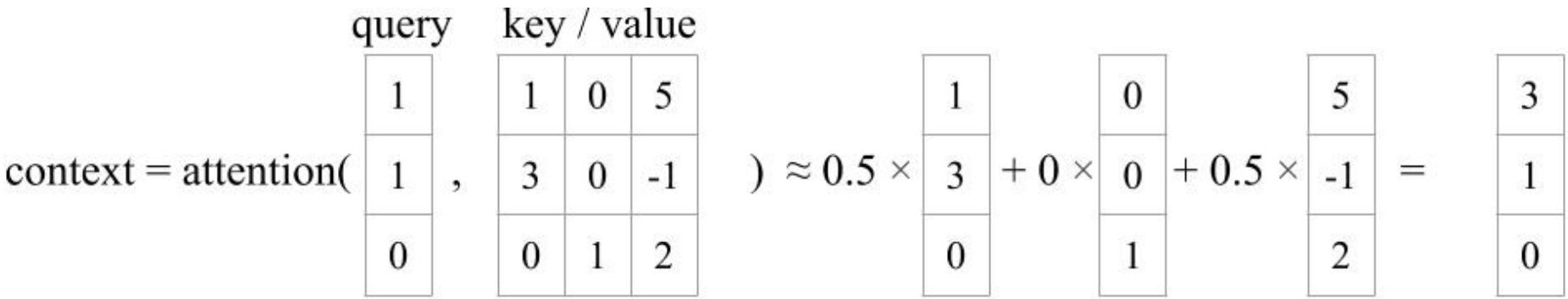
scaled by square root of the key dimension d_K.

- Invalid connections to the future inputs are masked out to preserve the autoregressive property.

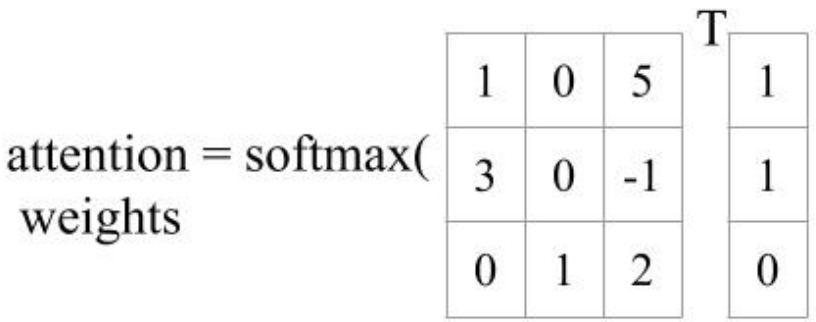


Example2: Dot-Product Attention

Assume the keys and the values are the same vectors:

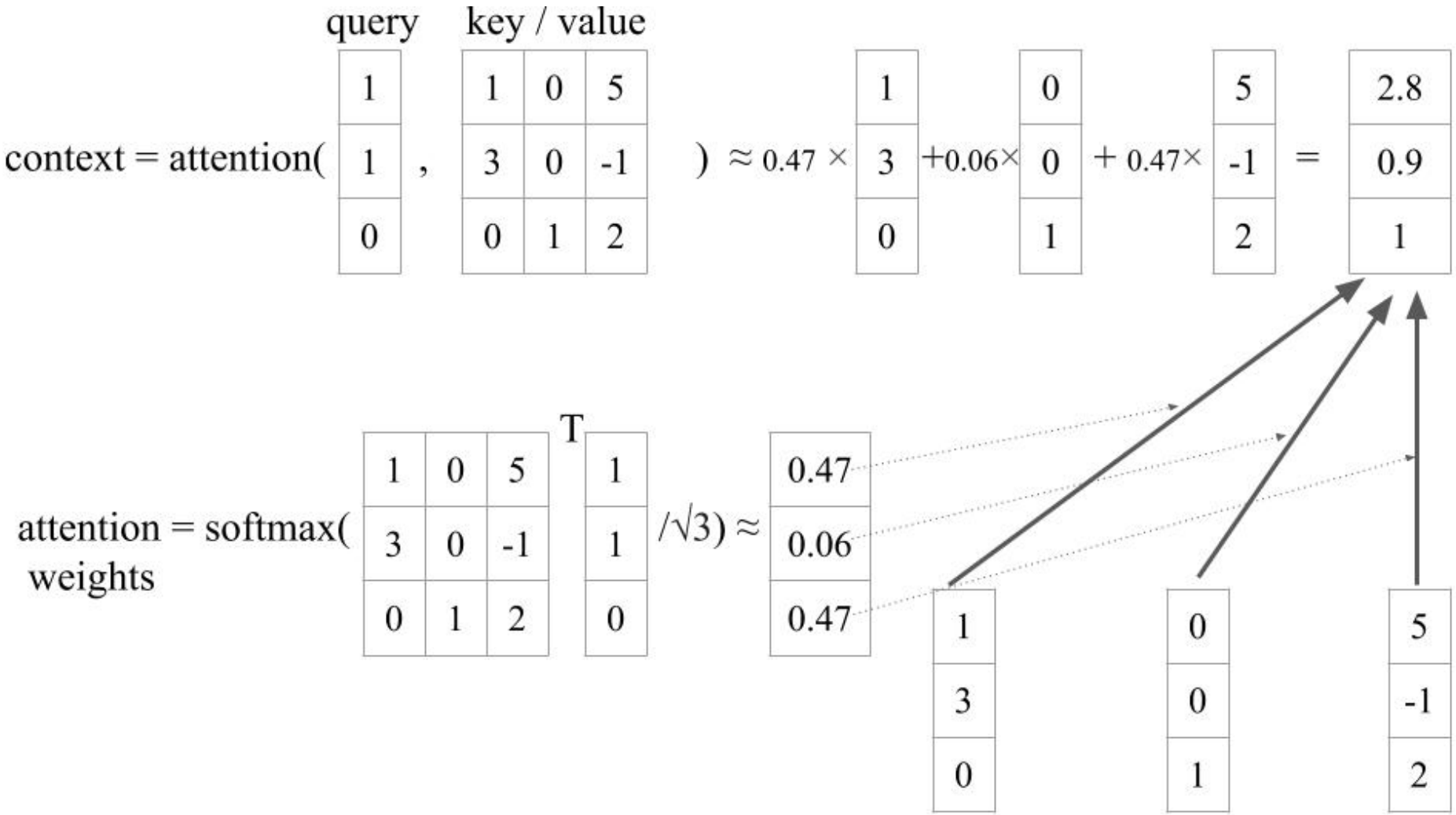


f(qeury, key) = dot(key, query)



Example3: Scaled Dot-Product Attention

Scale the un-normalized attention weights by the square root of the vector length:

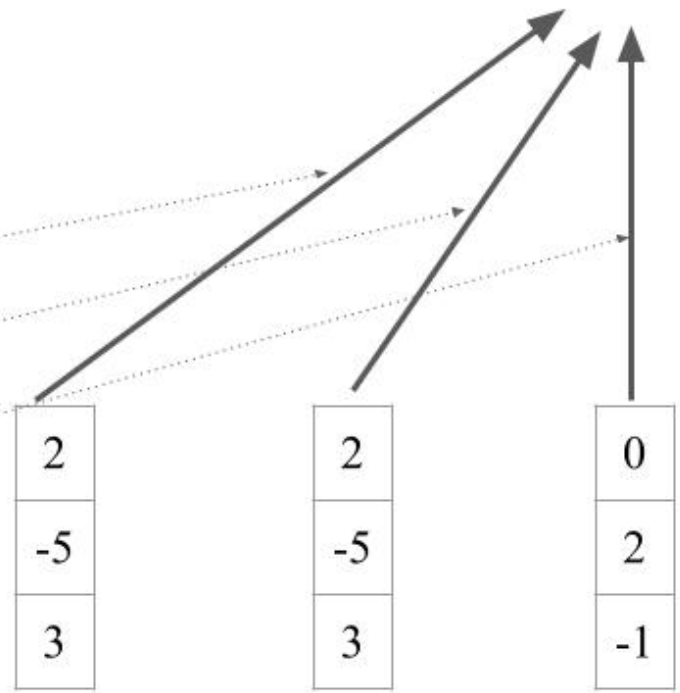


Example4: Different Keys and Values

When the key and the value vectors are different:

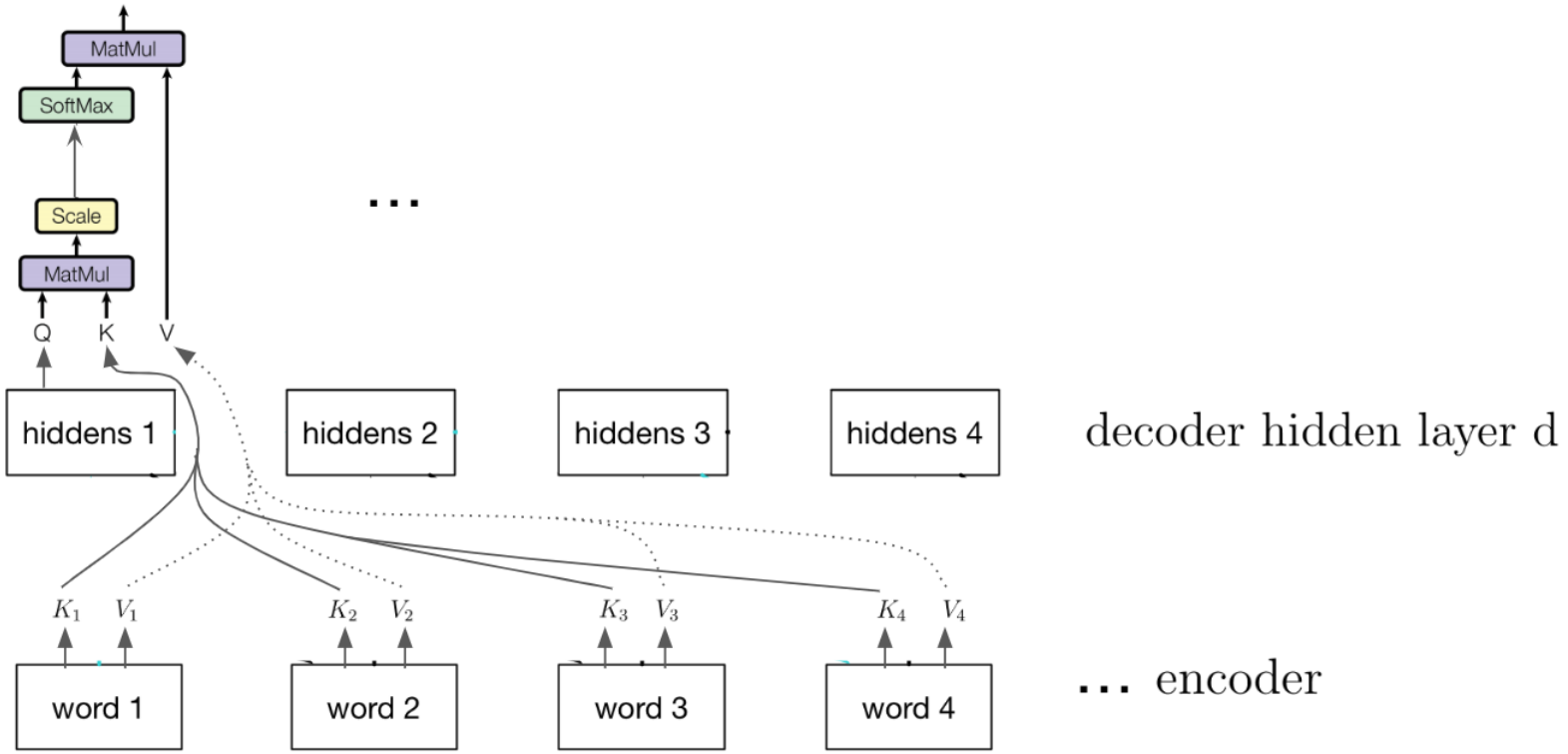
$$\text{context} = \text{attention}\Big(\underset{\text{query}}{\begin{bmatrix}1\\1\\0\end{bmatrix}}, \underset{\text{key}}{\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}}, \underset{\text{value}}{\begin{bmatrix}2&2&0\\-5&-5&2\\3&3&-1\end{bmatrix}}\Big) \approx 0.47\times\begin{bmatrix}2\\-5\\3\end{bmatrix} + 0.06\times\begin{bmatrix}2\\-5\\3\end{bmatrix} + 0.47\times\begin{bmatrix}0\\2\\-1\end{bmatrix} = \begin{bmatrix}1.06\\-1.71\\1.12\end{bmatrix}$$

$$\underset{\text{weights}}{\text{attention}} = \text{softmax}\Big(\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}^T/\sqrt{3}\Big) \approx \begin{bmatrix}0.47\\0.06\\0.47\end{bmatrix}$$



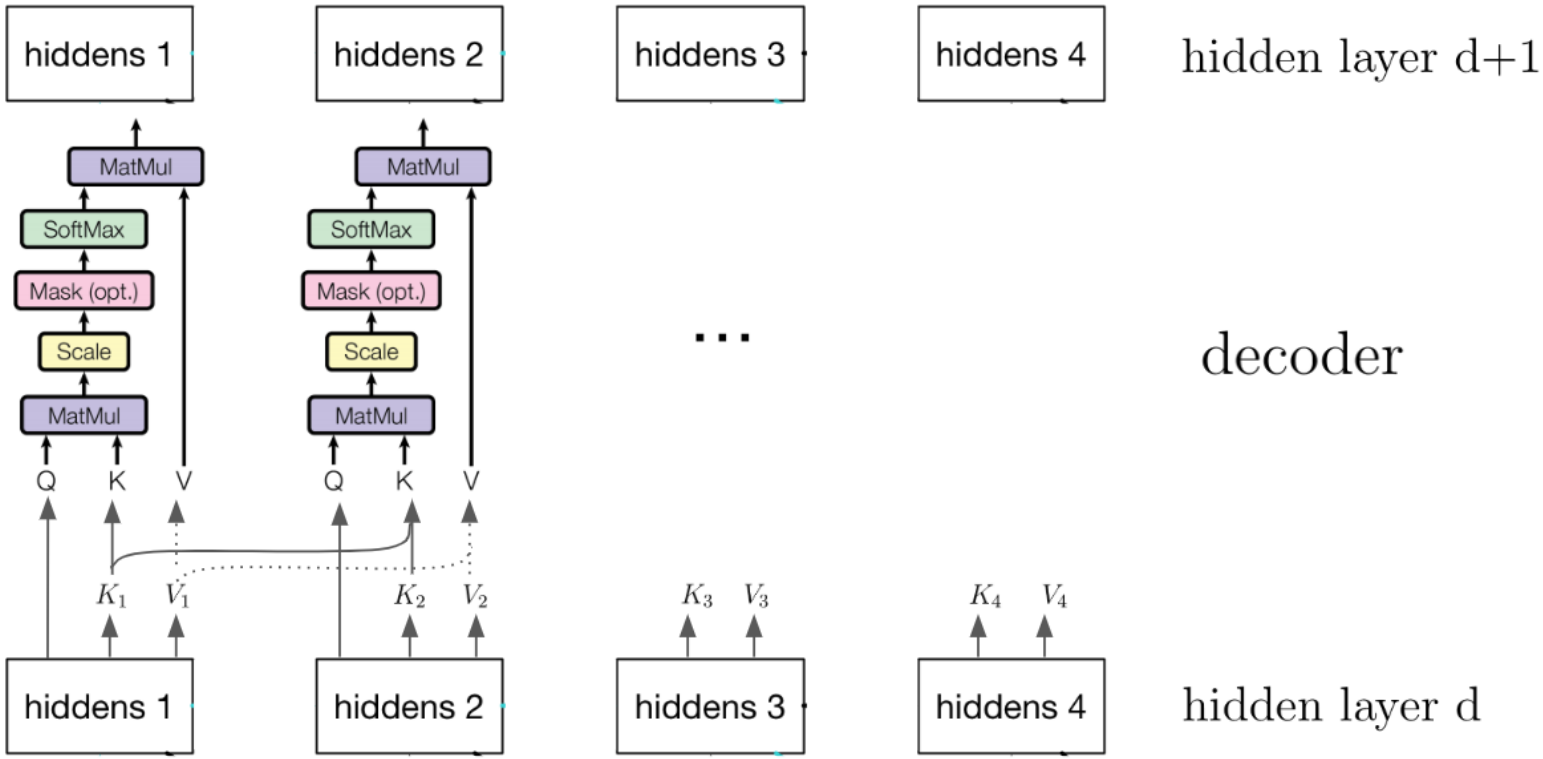
Attention is All You Need

- Transformer models attend to both the encoder annotations and its previous hidden layers.
- When attending to the encoder annotations, the model computes the key-value pairs using linearly transformed the encoder outputs.

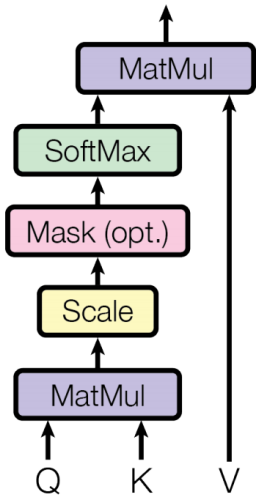


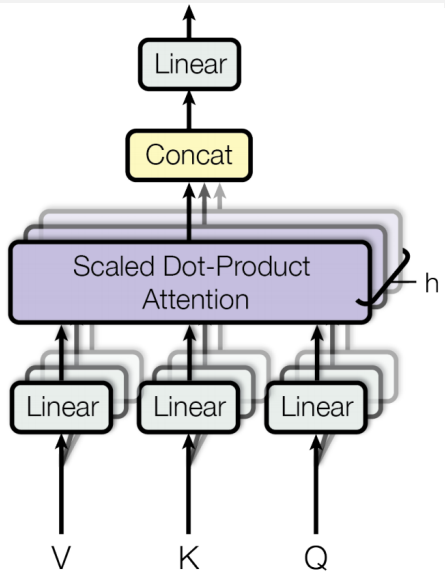
Attention is All You Need

- Transformer models also use "self-attention" on its previous hidden layers.
- When applying attention to the previous hidden layers, the casual structure is preserved.



Attention is All You Need





- The Scaled Dot-Product Attention attends to one or few entries in the input key-value pairs.
  - Humans can attend to many things simultaneously.
- The idea: apply Scaled Dot-Product Attention multiple times on the linearly transformed inputs.

$$\text{MultiHead}(Q, K, V) = \text{concat}\left(\mathbf{c}_1, \cdots, \mathbf{c}_h\right) W^O,$$

$$\mathbf{c}_i = \text{attention}(QW_i^Q, KW_i^K, VW_i^V).$$

Positional Encoding

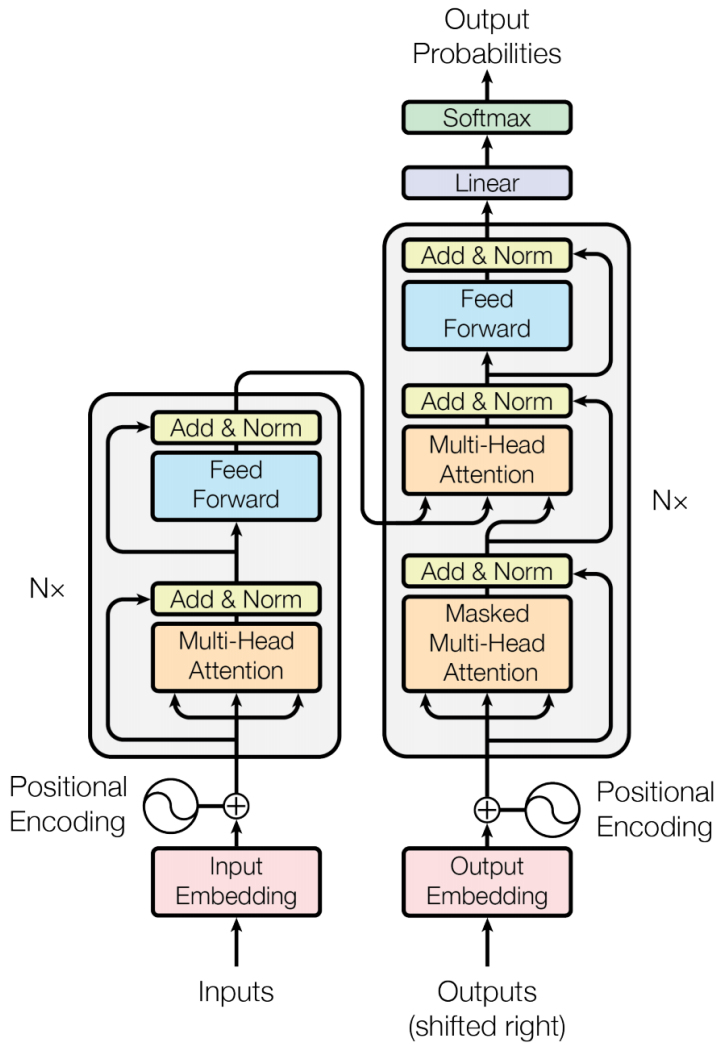
- Unlike RNNs and CNNs encoders, the attention encoder outputs do not depend on the order of the inputs. (Why?)
- The order of the sequence conveys important information for the machine translation tasks and language modeling.
- The idea: add positional information of a input token in the sequence into the input embedding vectors.

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

- The final input embeddings are the concatenation of the learnable embedding and the postional encoding.

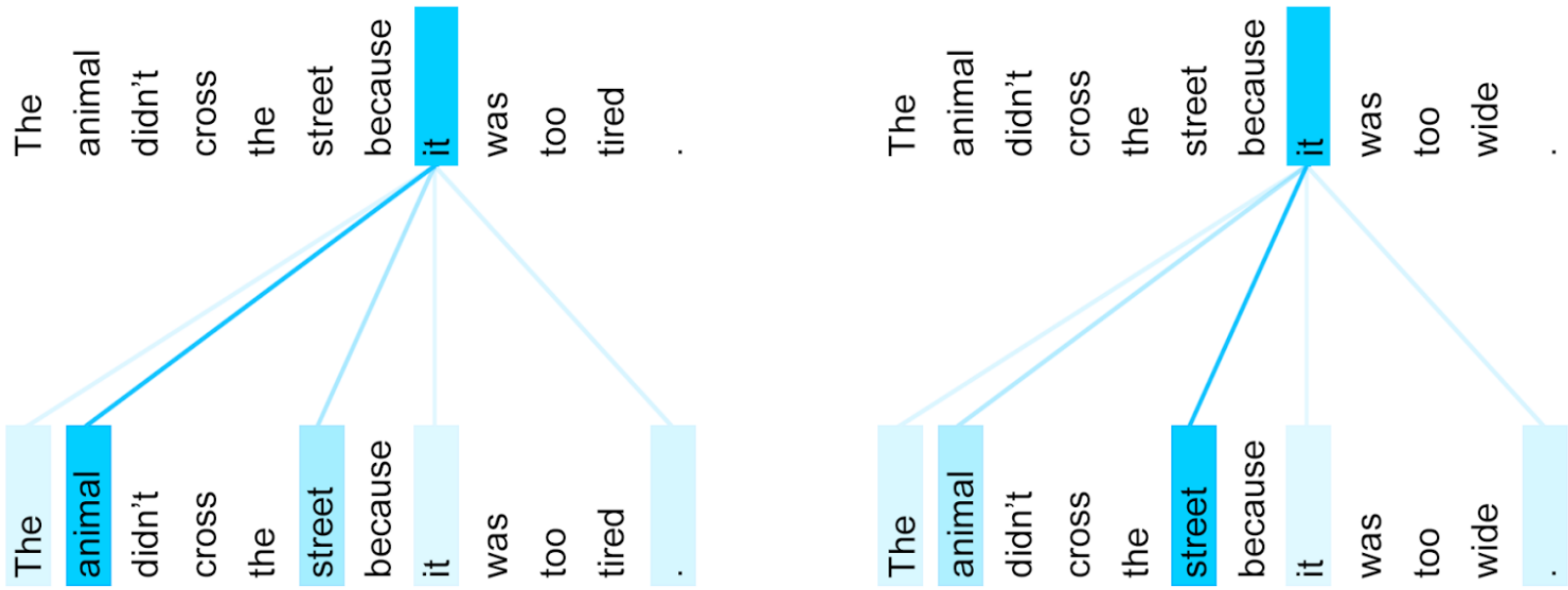
Transformer Machine Translation

- Transformer has a encoder-decoder architecture similar to the previous RNN models.
  - except all the recurrent connections are replaced by the attention modules.
- The transfomer model uses N stacked self-attention layers.
- Skip-connections help preserve the positional and identity information from the input sequences.



Transformer Machine Translation

- Self-attention layers learnt "it" could refer to different entities in the different contexts.



- Visualization of the 5th to 6th self-attention layer in the encoder.

`https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html`

Transformer Machine Translation

- BLEU scores of state-of-the-art models on the WMT14 English-to-German translation task

Translation Model	Training time	BLEU (diff. from MOSES)
Transformer (large)	3 days on 8 GPU	**28.4 (+7.8)**
Transformer (small)	1 day on 1 GPU	24.9 (+4.3)
GNMT + Mixture of Experts	1 day on 64 GPUs	26.0 (+5.4)
ConvS2S (FB)	18 days on 1 GPU	25.1 (+4.5)
GNMT	1 day on 96 GPUs	24.6 (+4.0)

Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

After the break

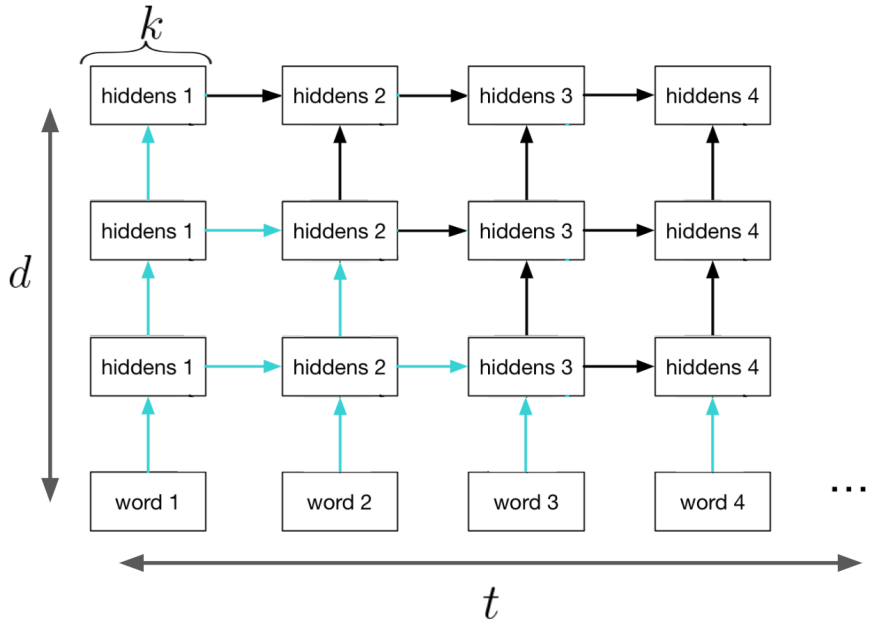
After the break: **Computational Cost**

Computational Cost and Parallelism

- There are a few things we should consider when designing an RNN.
- Computational cost:
  - **Number of connections.** How many add-multiply operations for the forward and backward pass.
  - **Number of time steps.** How many copies of hidden units to store for Backpropgation Through Time.
  - **Number of sequential operations.** The computations cannot be parallelized. (The part of the model that requires a for loop).
- **Maximum path length across time**: the shortest path length between the first encoder input and the last decoder output.
  - It tells us how easy it is for the RNN to remember / retreive information from the input sequence.

Computational Cost and Parallelism

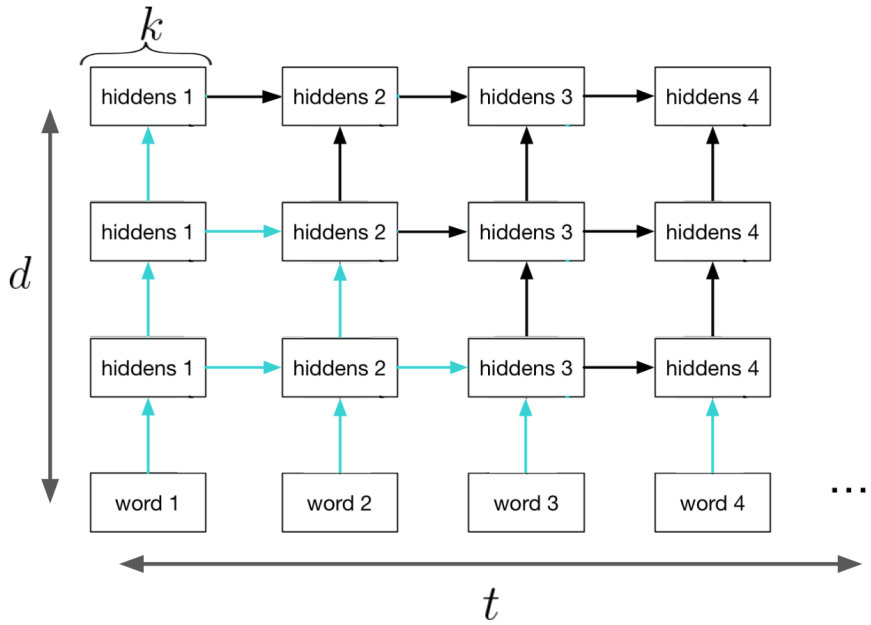
- Consider a standard d layer RNN from Lecture 7 with k hidden units, training on a sequence of length t.



- There are k^2 connections for each hidden-to-hidden connection. A total of $t \times k^2 \times d$ connections.
- We need to store all $t \times k \times d$ hidden units during training.
- Only $k \times d$ hidden units need to be stored at test time.

Computational Cost and Parallelism

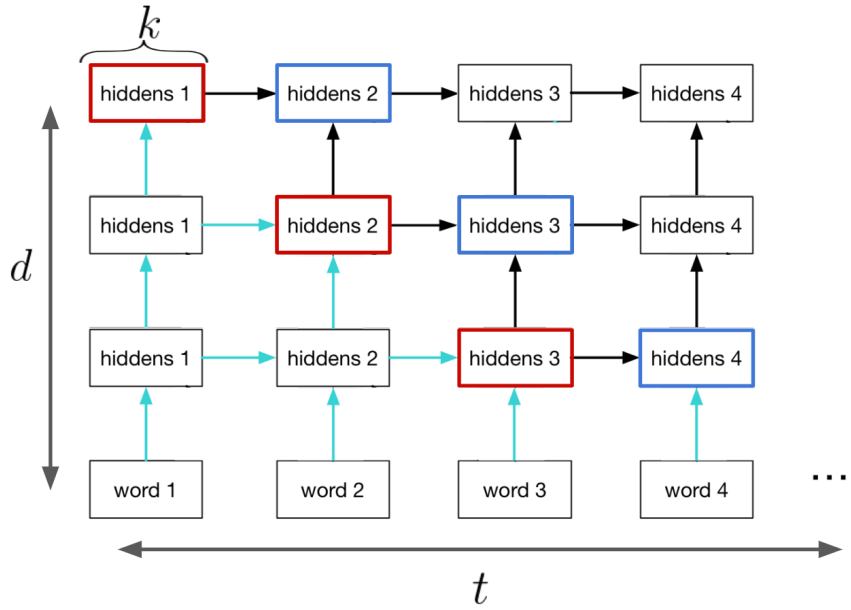
- Consider a standard d layer RNN from Lecture 7 with k hidden units, training on a sequence of length t.



- Which hidden layers can be computed in parallel in this RNN?

Computational Cost and Parallelism

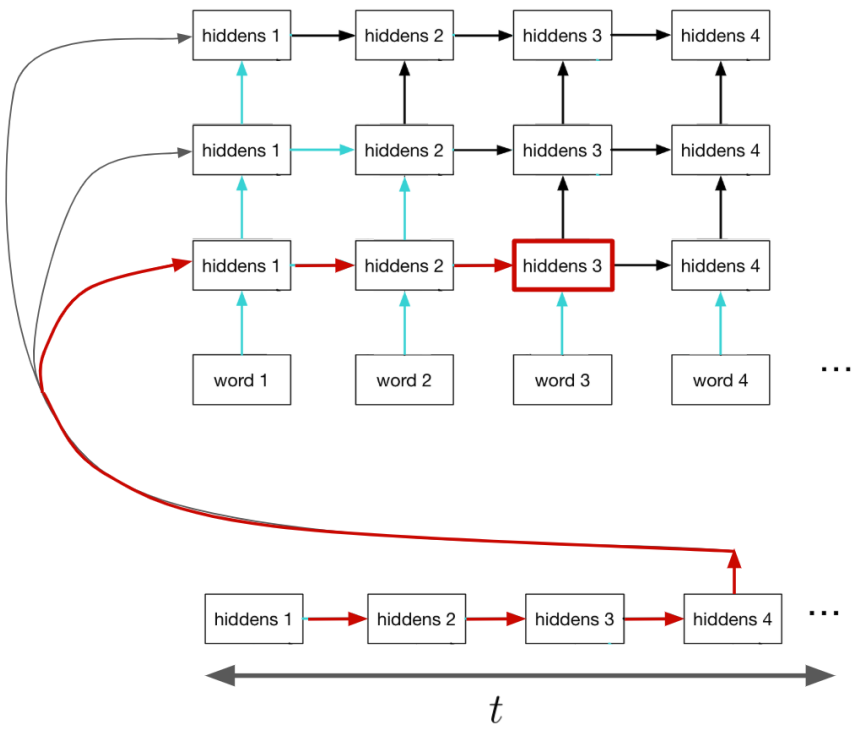
- Consider a standard d layer RNN from Lecture 7 with k hidden units, training on a sequence of length t.

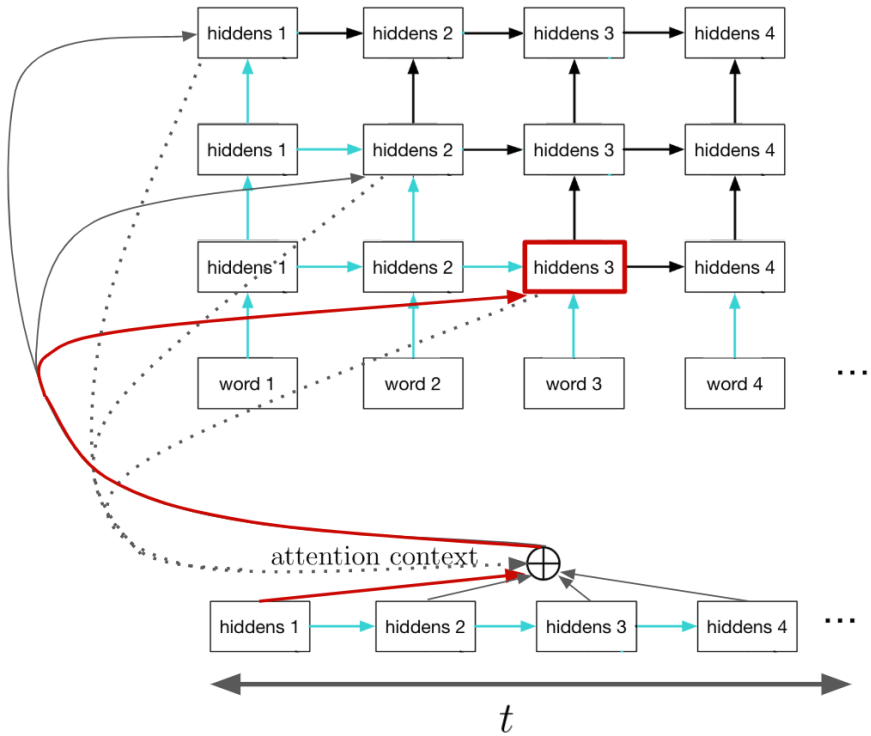


- Both the input embeddings and the outputs of an RNN can be computed in parallel.
- The blue hidden units are independent given the red.
- The numer of sequential operation is still propotional to t.

Computational Cost and Parallelism

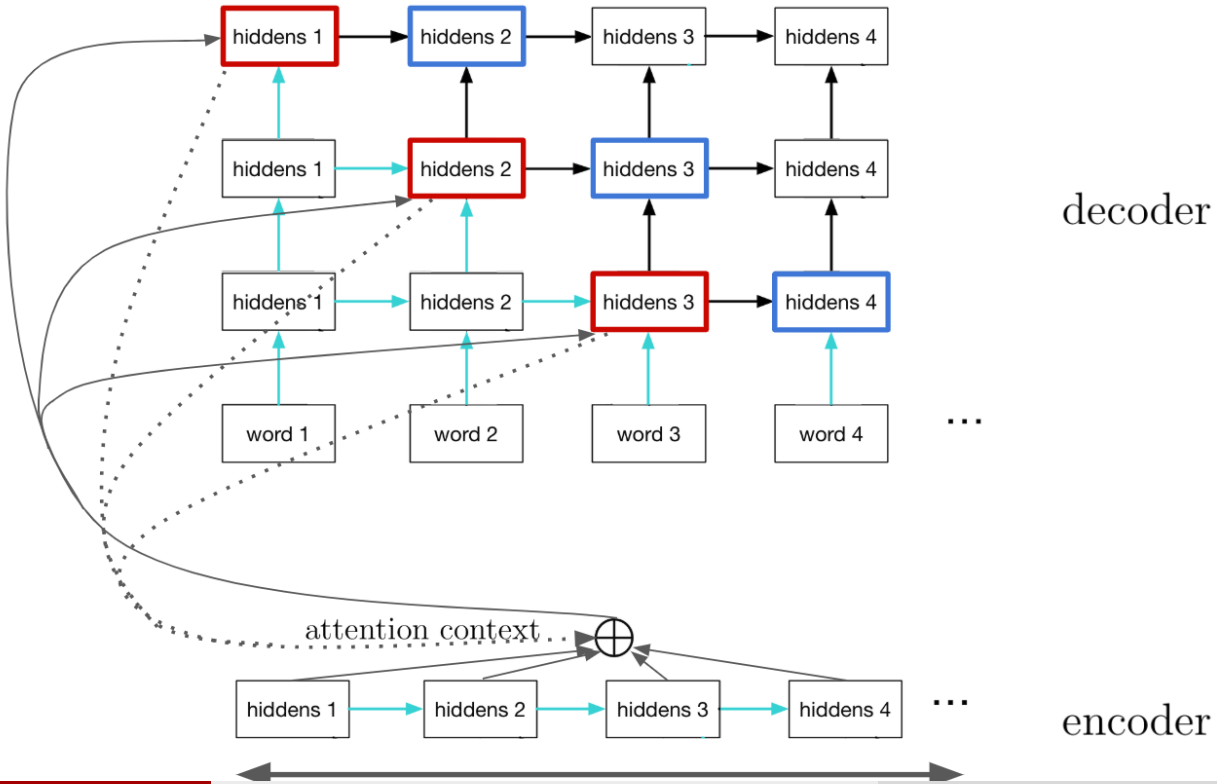
- During backprop, in the standard encoder-decoder RNN, the maximum path length across time is the number of time steps.
- Attention-based RNNs have a constant path length between the encoder inputs and the decoder hidden states.
  - Learning becomes easier. Why?





Computational Cost and Parallelism

- During forward pass, attention-based RNNs achieves efficient content-based addressing at the cost of re-computing context vectors at each time step.
  - *Bahdanau et. al.* computes context vector over the entire input sequence of length t using a neural network of k^2 connections.
  - Computing the context vectors adds a $t \times k^2$ cost at each time step.



Computational Cost and Parallelism

- In summary:
  - t: sequence length, d: # layers and k: # neurons at each layer.

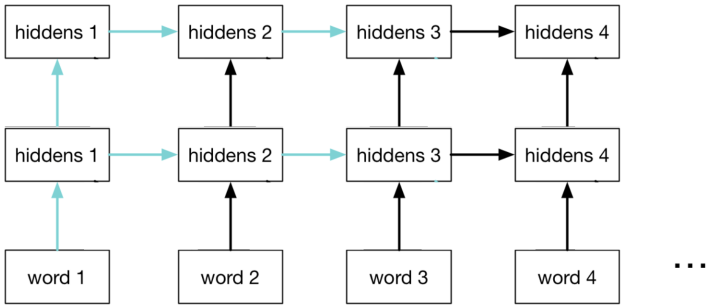
Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$

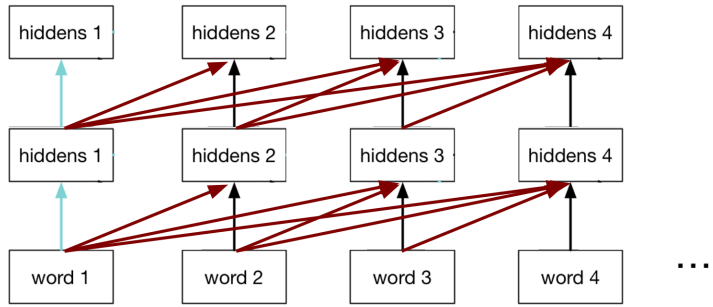
- Attention needs to re-compute context vectors at every time step.
- Attention has the benefit of reducing the maximum path length between long range dependencies of the input and the target sentences.

Model	sequential operations	maximum path length across time
RNN	t	t
RNN+attn.	t	1

Improve Parallelism

- RNNs are sequential in the sequence length t due to the number hidden-to-hidden lateral connections.
  - RNN architecture limits the parallelism potential for longer sequences.
- Improve parallelism: remove the lateral connections. We will have a deep autoregressive model, where the hidden units depends on all the previous time steps.

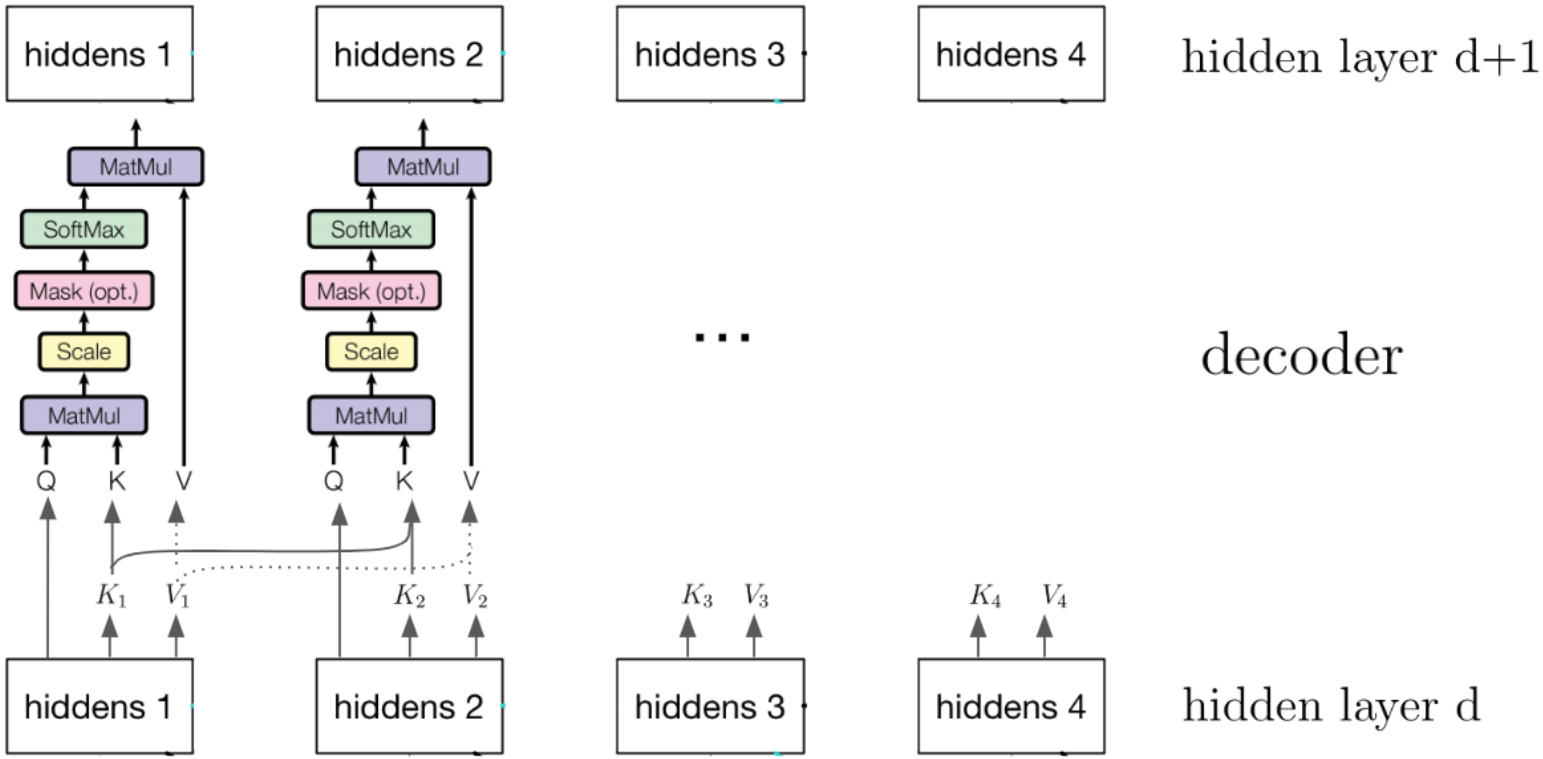




  - Benefit: the number of sequential operations is now linear in the depth d, but is independent of the sequence length t. (usually $d << t$.)

Computational Cost and Parallelism

- Self-attention allows the model to learn to access information from the past hidden layer, but decoding is very expensive.
- When generating sentences, the computation in the self-attention decoder grows as the sequence gets longer.



Computational Cost and Parallelism

- t: sequence length, d: # layers and k: # neurons at each layer.

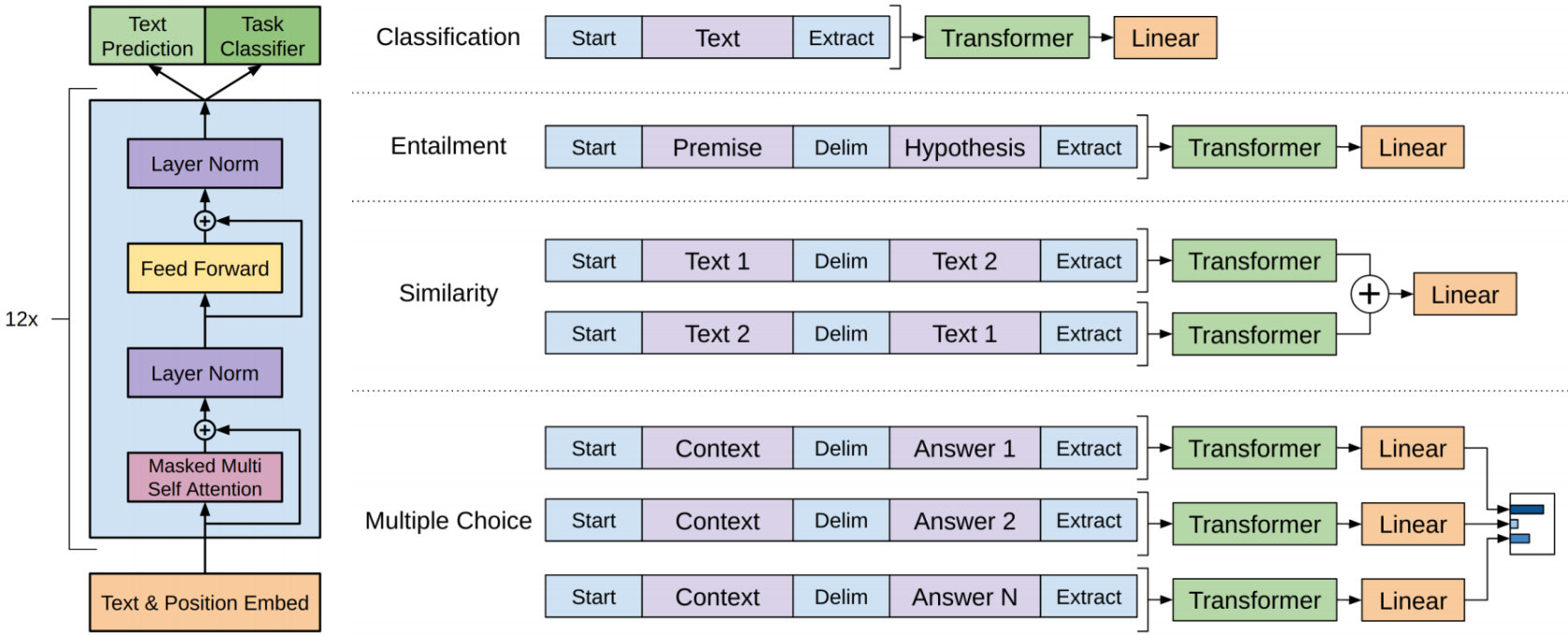
Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$
transformer	$t^2 \times k \times d$	$t \times k \times d$	$t^2 \times k \times d$	$t \times k \times d$

- **Transformer vs RNN**: There is a trade-off between the sequencial operations and decoding complexity.
  - The sequential operations in transformers are independent of sequence length, but they are very expensive to decode.
  - Transformers can learn faster than RNNs on parallel processing hardwards for longer sequences.

Model	sequential operations	maximum path length across time
RNN	t	t
RNN+attn.	t	1
transformer	d	1

Transformer Language Pre-training

- Similar to pre-training computer vision models on ImageNet, we can pre-train a language model for NLP tasks.
  - The pre-trained model is then fine-tuned on textual entailment, question answering, semantic similarity assessment, and document classification.



Radford, Alec, et al. "Improving Language Understanding by Generative Pre-Training." 2018.

Transformer Language Pre-training

- Increasing the training data set and the model size has a noticible improvement on the transformer language model. Cherry picked generated samples from *Radford, et al., 2019*:

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

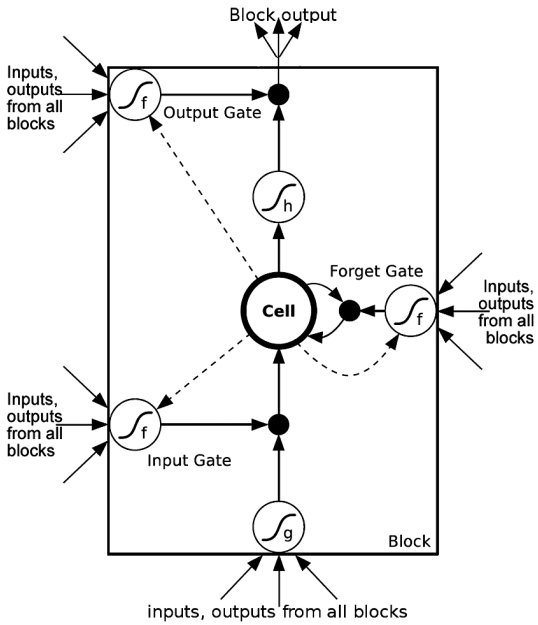
Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

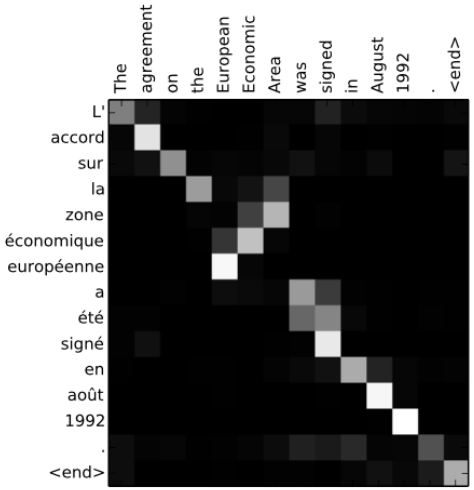
Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

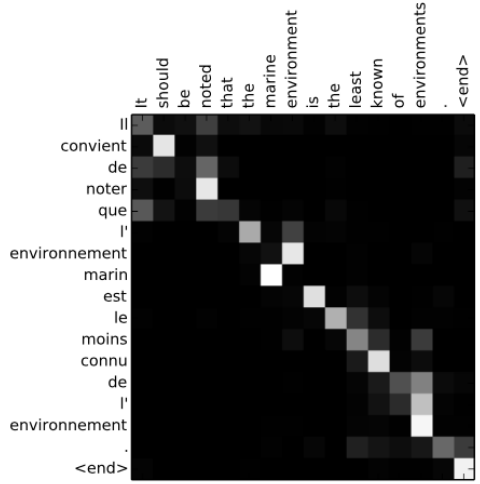
For the full text samples see Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners." 2019.

Neural Turing Machines (optional)

- We said earlier that multilayer perceptrons are like differentiable circuits.
- Using an attention model, we can build differentiable computers.
- We've seen hints that sparsity of memory accesses can be useful:



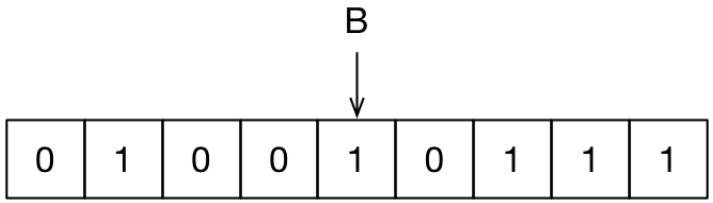




- Computers have a huge memory, but they only access a handful of locations at a time. Can we make neural nets more computer-like?

Neural Turing Machines (optional)

- Recall Turing machines:

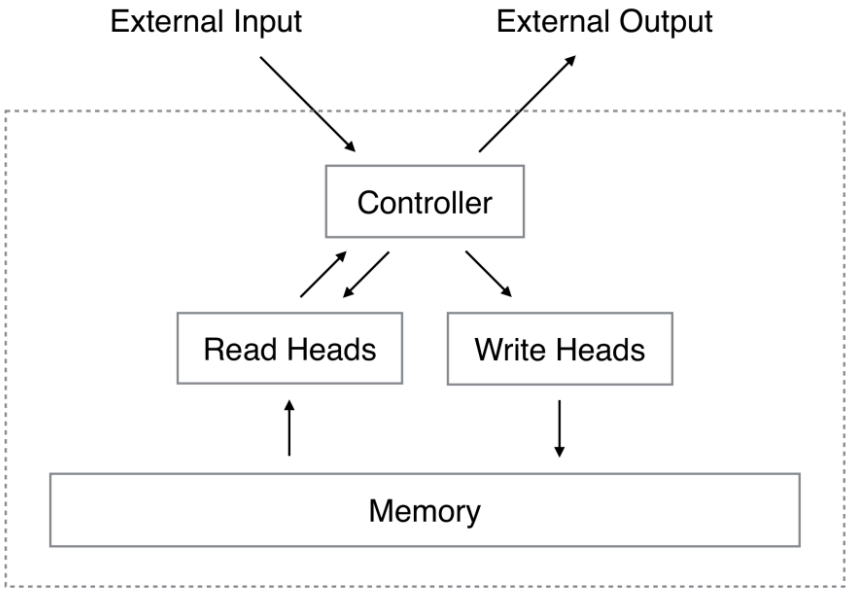


$$\langle A, 0, 0, B, \rightarrow \rangle$$
$$\langle A, 1, 0, A, \leftarrow \rangle$$
$$\ldots$$

- You have an infinite tape, and a head, which transitions between various states, and reads and writes to the tape.
- "If in state A and the current symbol is 0, write a 0, transition to state B, and move right."
- These simple machines are universal — they're capable of doing any computation that ordinary computers can.

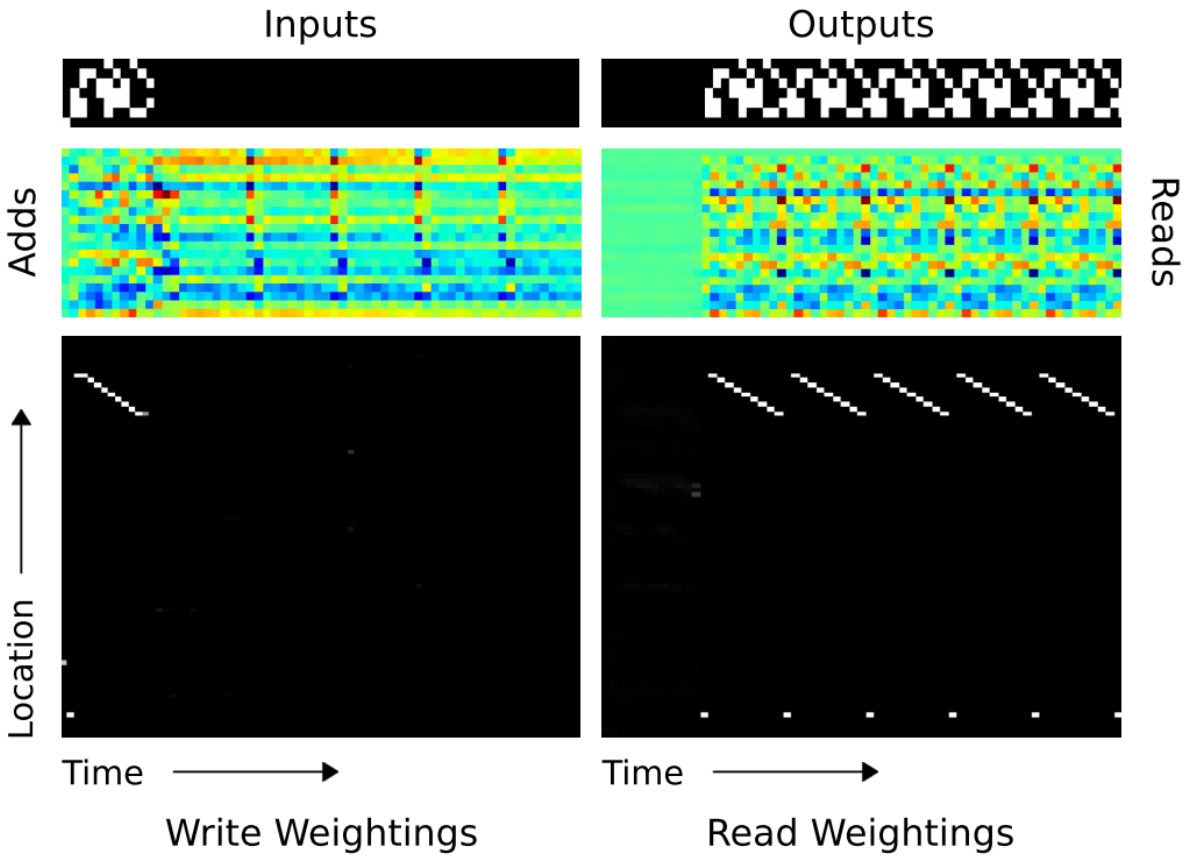
Neural Turing Machines (optional)

- Neural Turing Machines are an analogue of Turing machines where all of the computations are differentiable.
  - This means we can train the parameters by doing backprop through the entire computation.
- Each memory location stores a vector.
- The read and write heads interact with a weighted average of memory locations, just as in the attention models.
- The controller is an RNN (in particular, an LSTM) which can issue commands to the read/write heads.



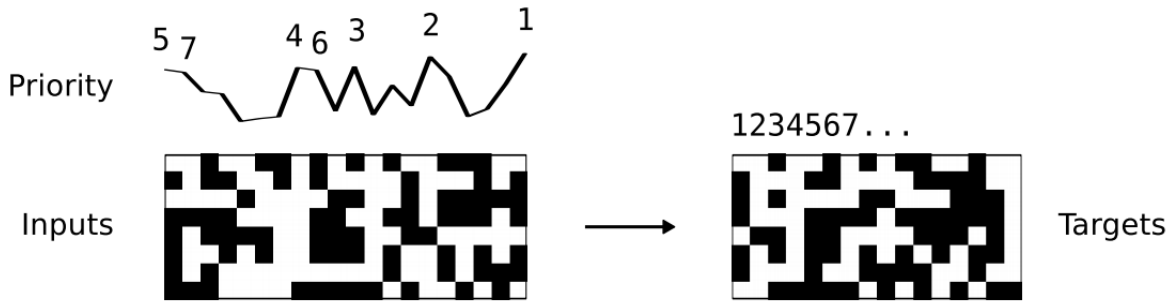
Neural Turing Machines (optional)

- Repeat copy task: receives a sequence of binary vectors, and has to output several repetitions of the sequence.
- Pattern of memory accesses for the read and write heads:



Neural Turing Machines (optional)

- Priority sort: receives a sequence of (key, value) pairs, and has to output the values in sorted order by key.



- Sequence of memory accesses: