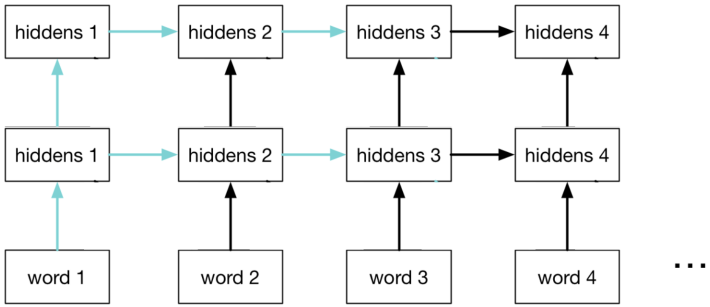
COMS 4995 Lecture 10: Transformers

Richard Zemel

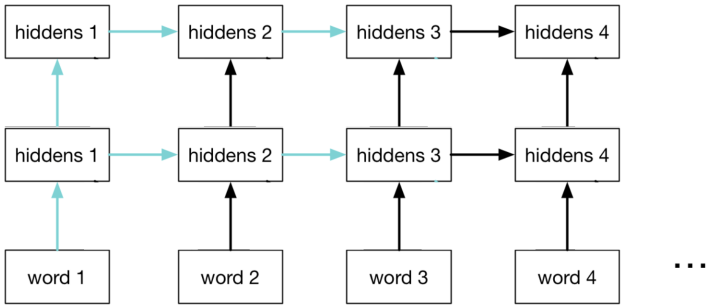
Attention is All You Need (Transformers)

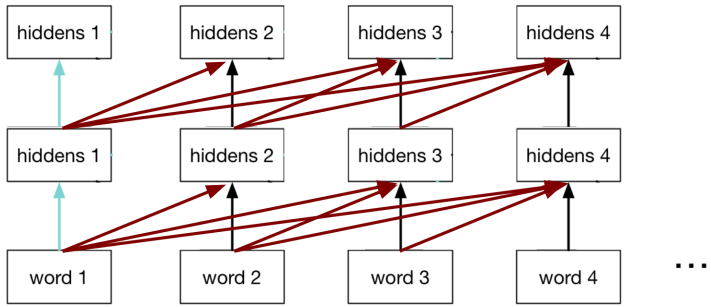
- We would like our model to have access to the entire history at the hidden layers.
- Previously we achieved this by having the recurrent connections.



Attention is All You Need (Transformers)

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





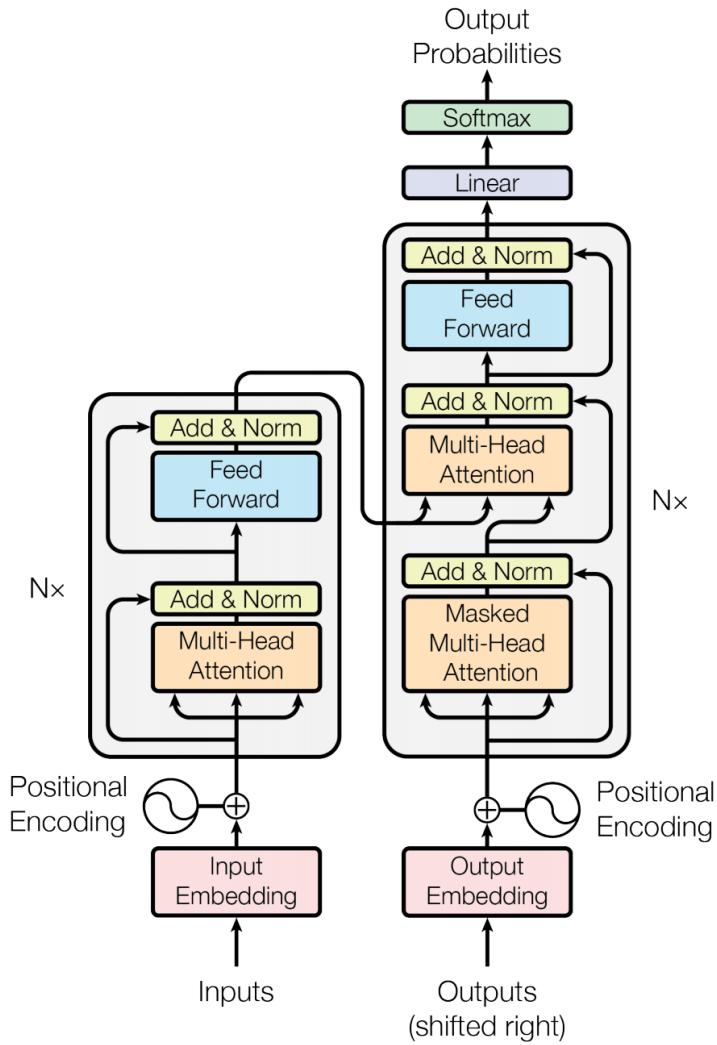
- **Core idea**: use attention to aggregate the context information by attending to one or a few important inputs from the past history.

Attention is All You Need

- We will now study a very successful neural network architecture for machine translation:

  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 attention modules.



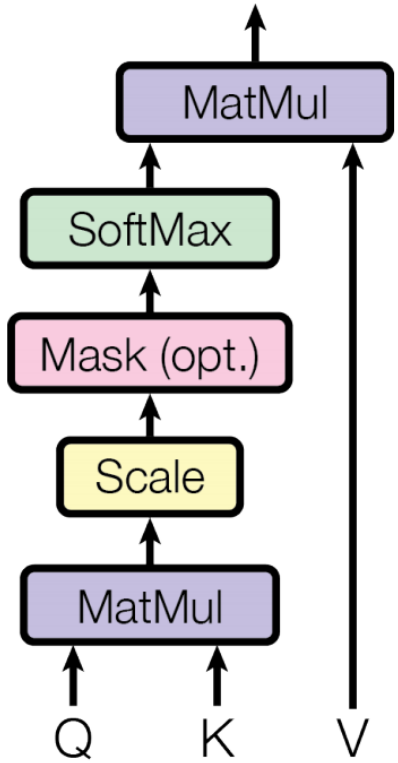
Attention is All You Need

- 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.



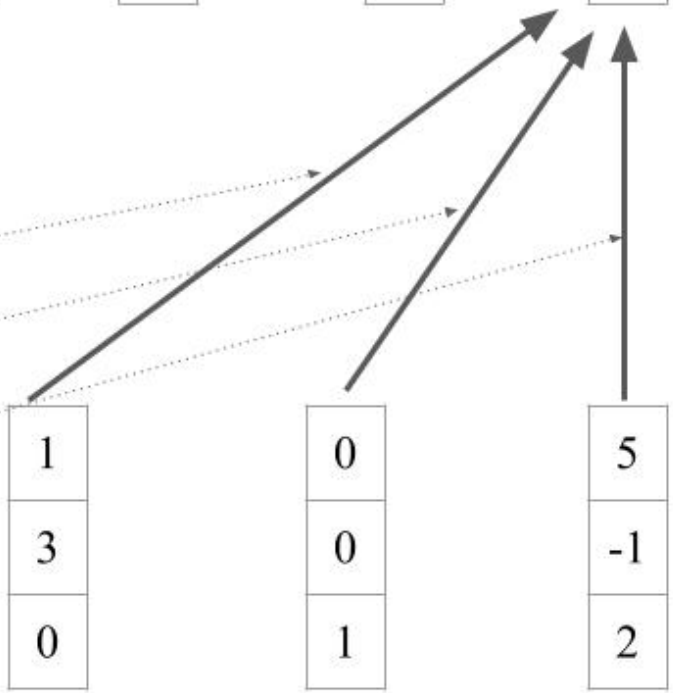
Example2: Dot-Product Attention

Assume the keys and the values are the same vectors:

$$\text{context} = \text{attention}(\underset{\text{query}}{\begin{bmatrix}1\\1\\0\end{bmatrix}}, \underset{\text{key / value}}{\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}}) \approx 0.5 \times \begin{bmatrix}1\\3\\0\end{bmatrix} + 0 \times \begin{bmatrix}0\\0\\1\end{bmatrix} + 0.5 \times \begin{bmatrix}5\\-1\\2\end{bmatrix} = \begin{bmatrix}3\\1\\0\end{bmatrix}$$

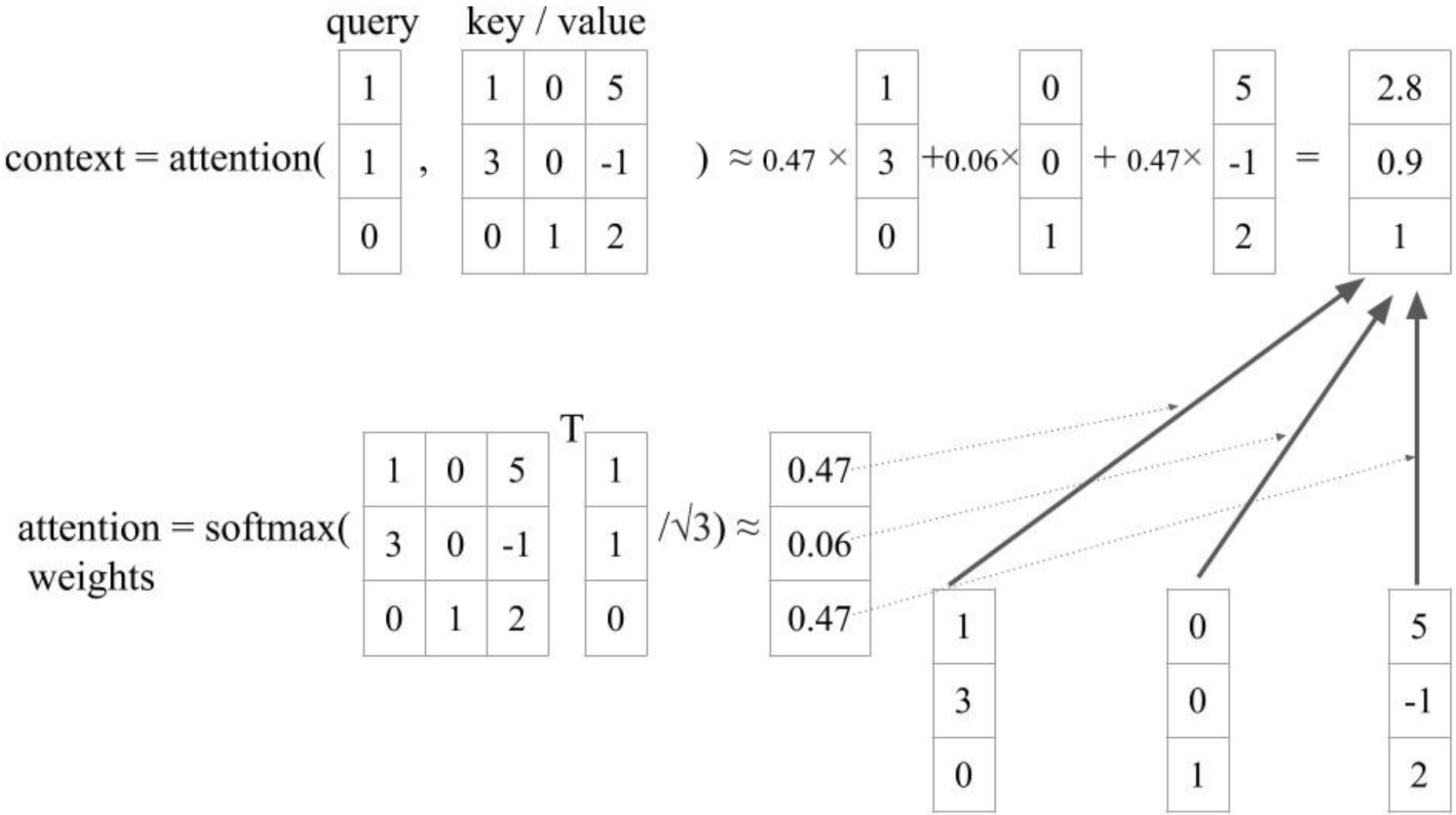
$$f(\text{qeury}, \text{key}) = \text{dot}(\text{key}, \text{query})$$

$$\text{attention} = \text{softmax}(\underset{\text{weights}}{\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}} \begin{bmatrix}1\\1\\0\end{bmatrix}^T) \approx \begin{bmatrix}0.5\\0\\0.5\end{bmatrix}$$



Example3: Scaled Dot-Product Attention

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

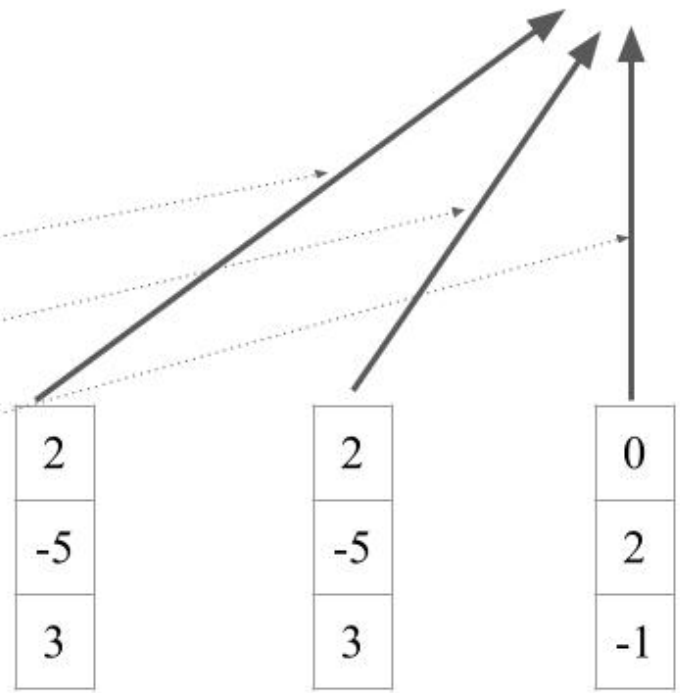


Example4: Different Keys and Values

When the key and the value vectors are different:

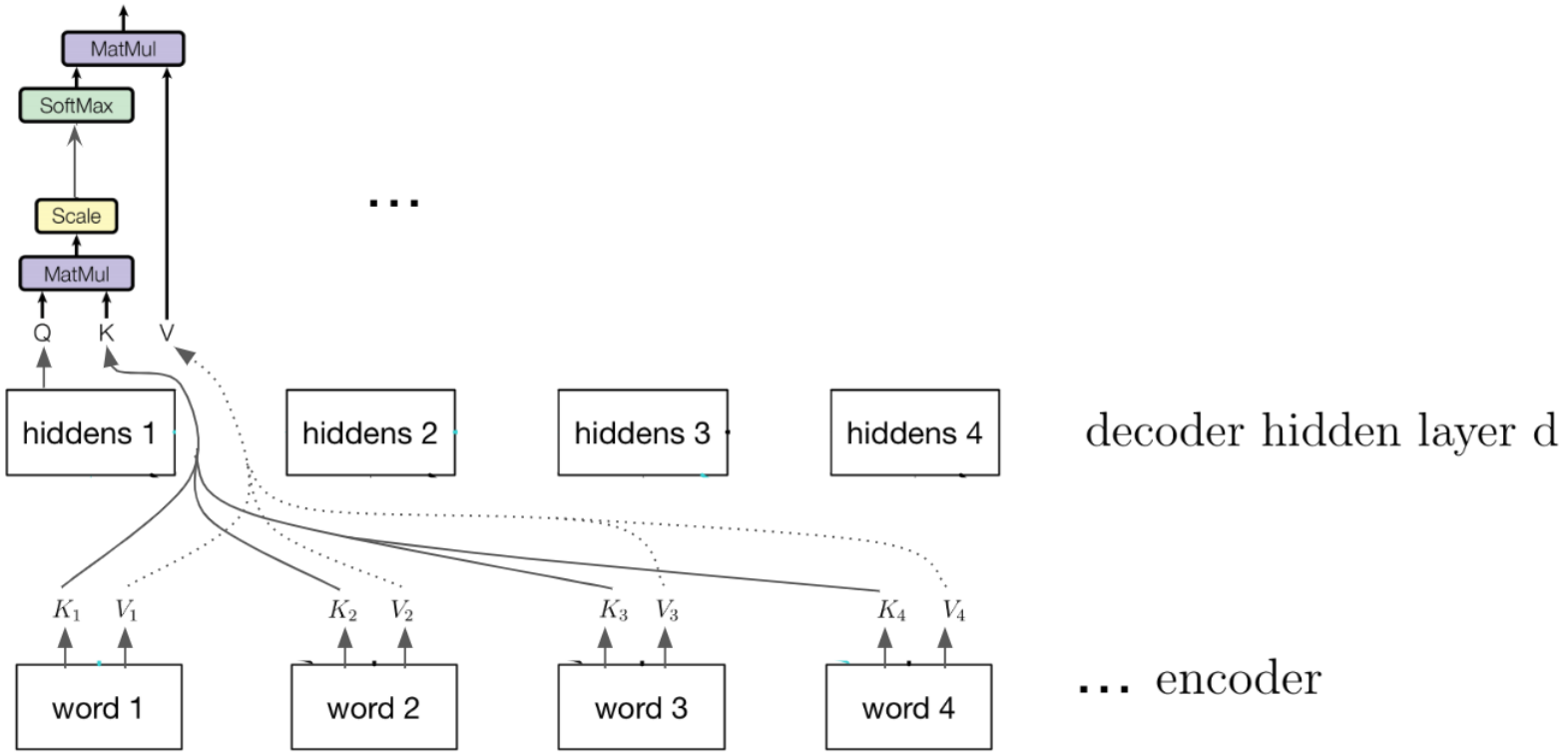
$$\text{context} = \text{attention}\left(\underset{\text{query}}{\begin{bmatrix}1\\1\\0\end{bmatrix}}, \underset{\text{key}}{\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}}, \underset{\text{value}}{\begin{bmatrix}2&2&0\\-5&-5&2\\3&3&-1\end{bmatrix}}\right) \approx 0.47\times\begin{bmatrix}2\\-5\\3\end{bmatrix}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

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



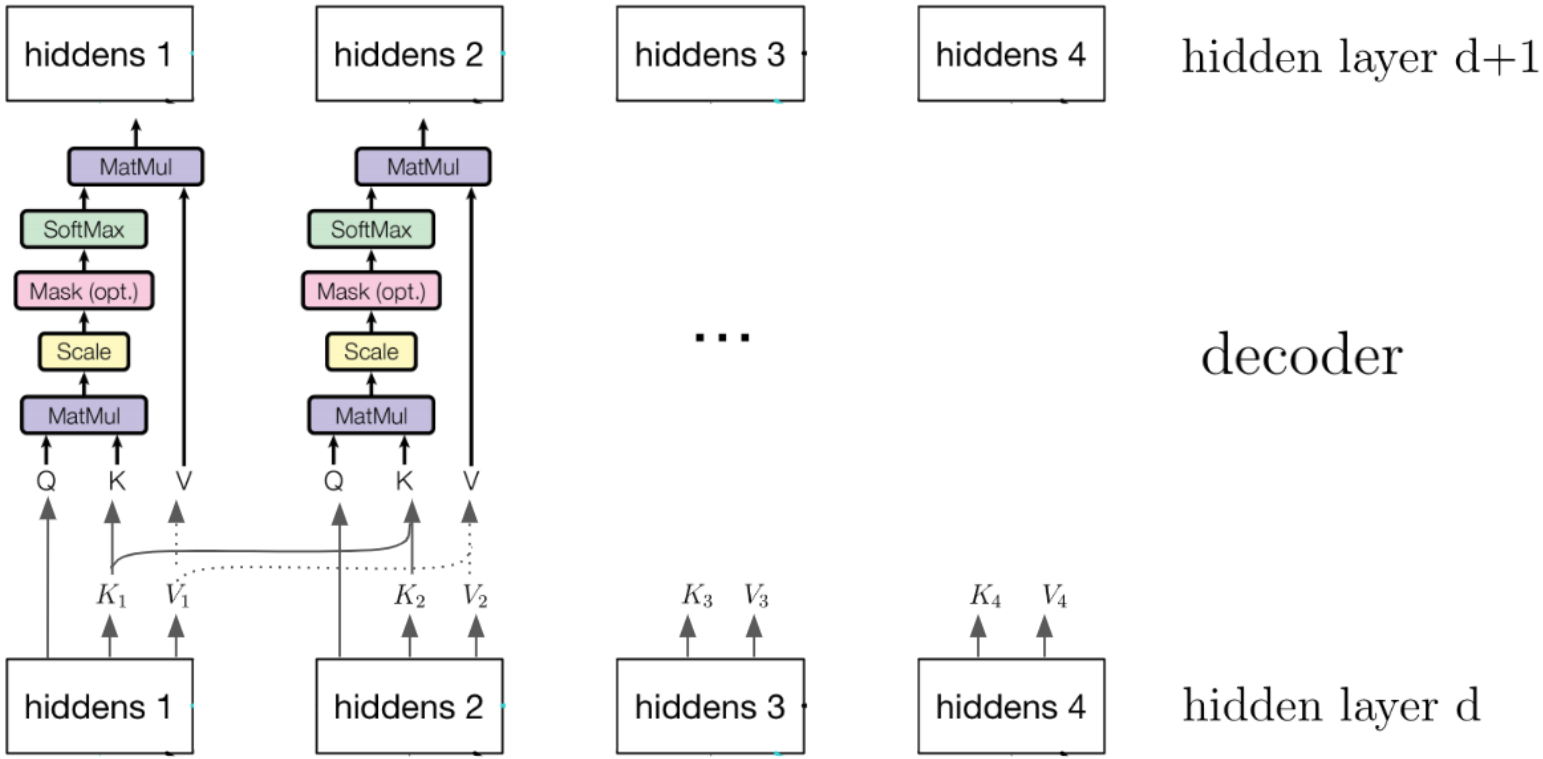
Attention is All You Need

- 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 encoder outputs.

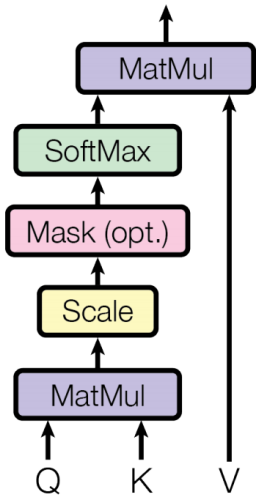


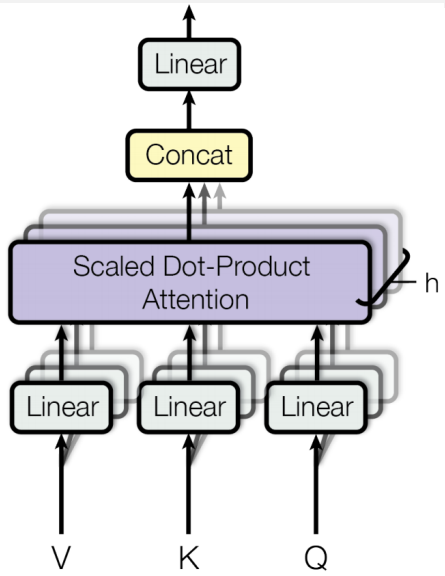
Attention is All You Need

- Transformer models also use "self-attention" on its previous hidden layers.
- When applying attention to the previous hidden layers, the causal 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

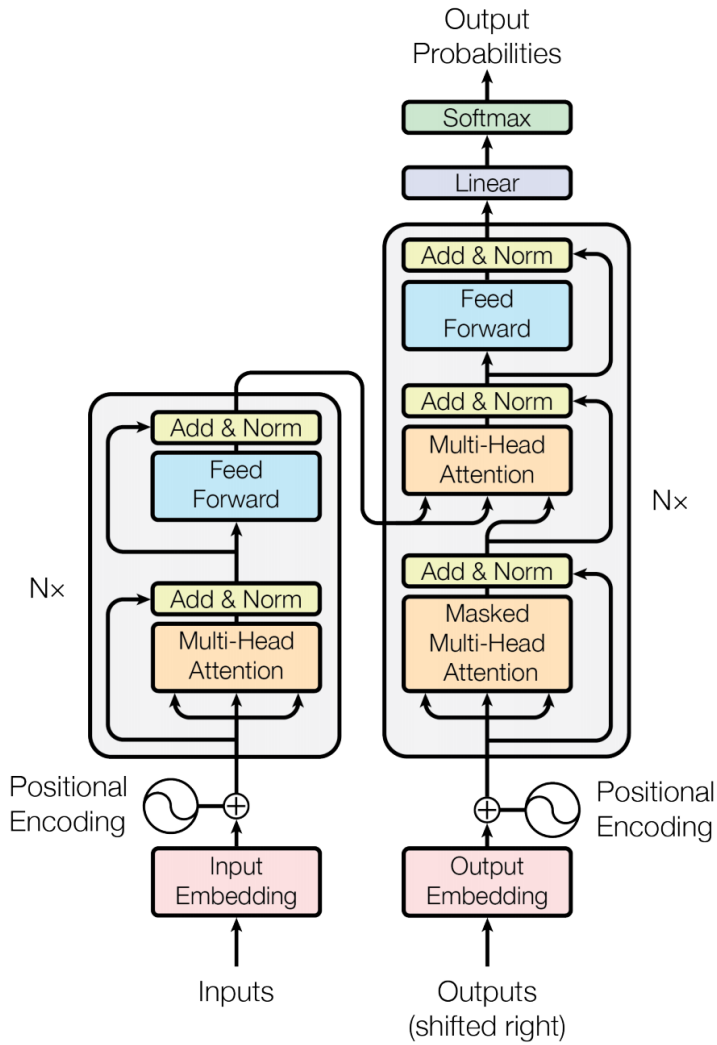
- Unlike RNNs and CNNs encoders, the attention encoder outputs do not depend on the order of the inputs. (Why?)
- The order of the sequence conveys important information for the machine translation tasks and language modeling.
- The idea: add positional information of a input token in the sequence into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

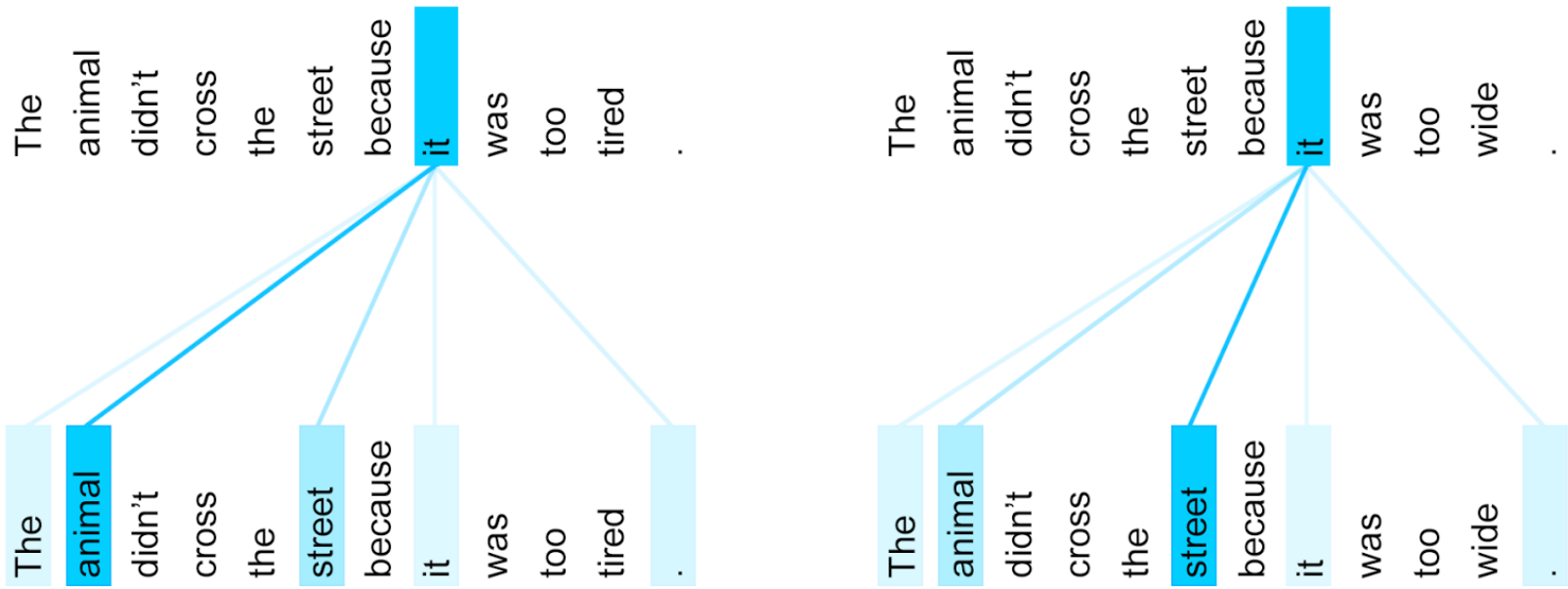
Transformer Machine Translation

- 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)

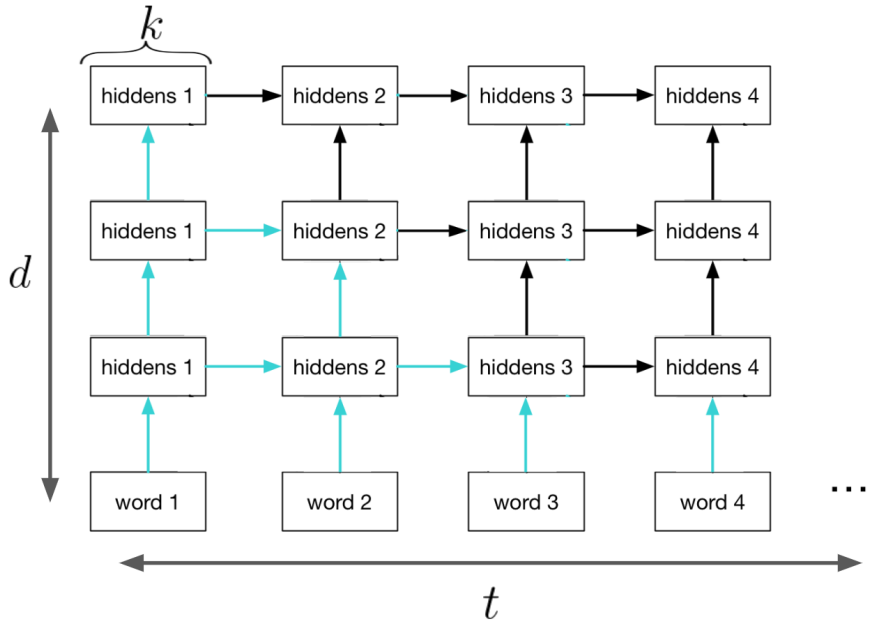
Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

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

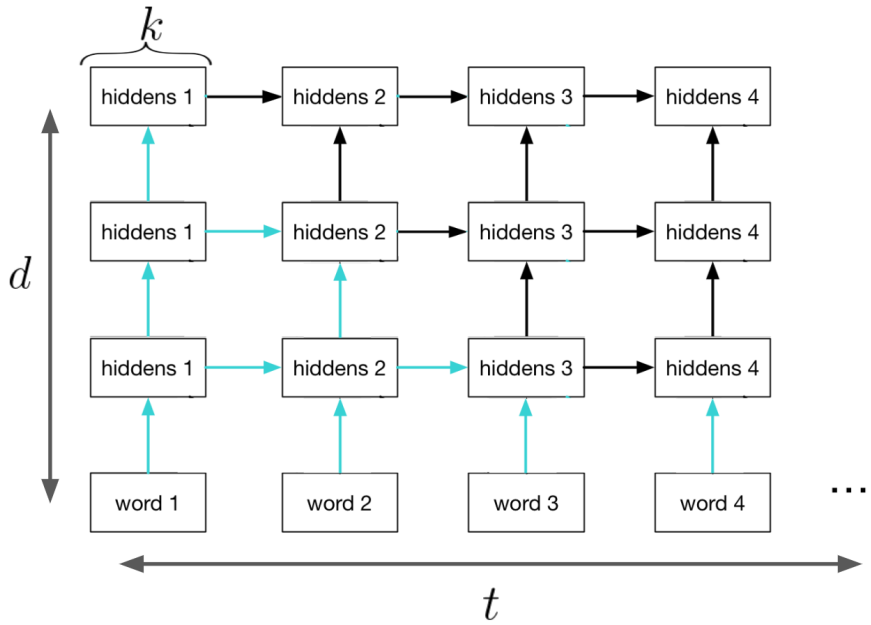
- Consider a standard d layer RNN with k hidden units, training on a sequence of length t.



- 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

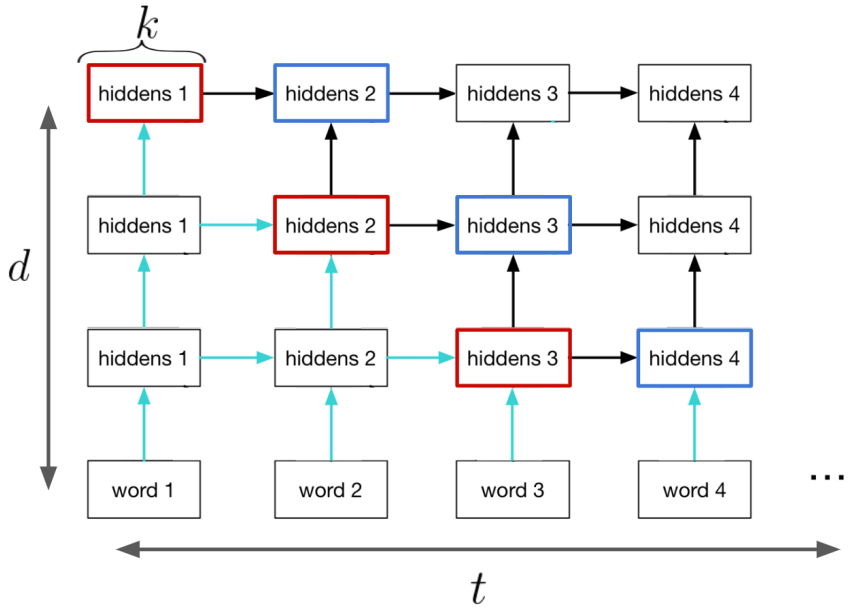
- Consider a standard d layer RNN from Lecture 7 with k hidden units, training on a sequence of length t.



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

Computational Cost and Parallelism

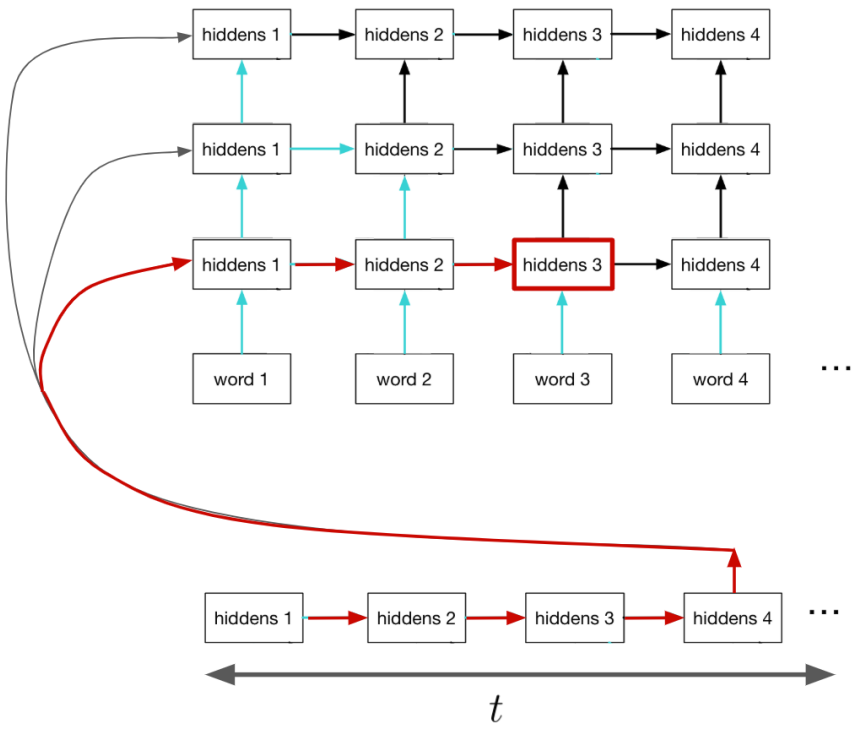
- Consider a standard d layer RNN from Lecture 7 with k hidden units, training on a sequence of length t.

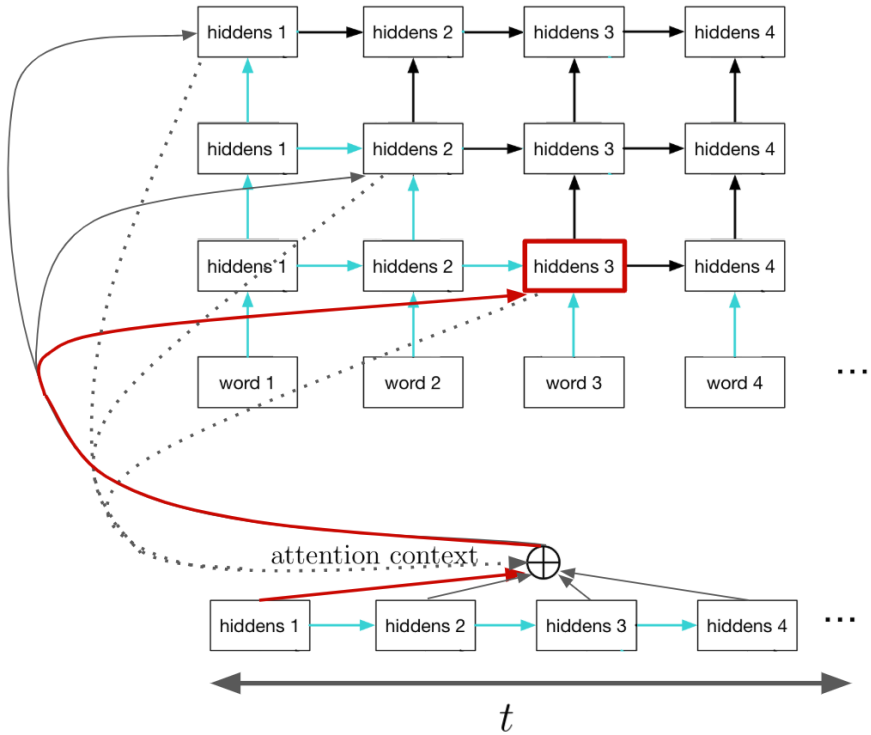


- Both the input embeddings and the outputs of an RNN can be computed in parallel.
- The blue hidden units are independent given the red.
- The number of sequential operations is still proportional to t.

Computational Cost and Parallelism

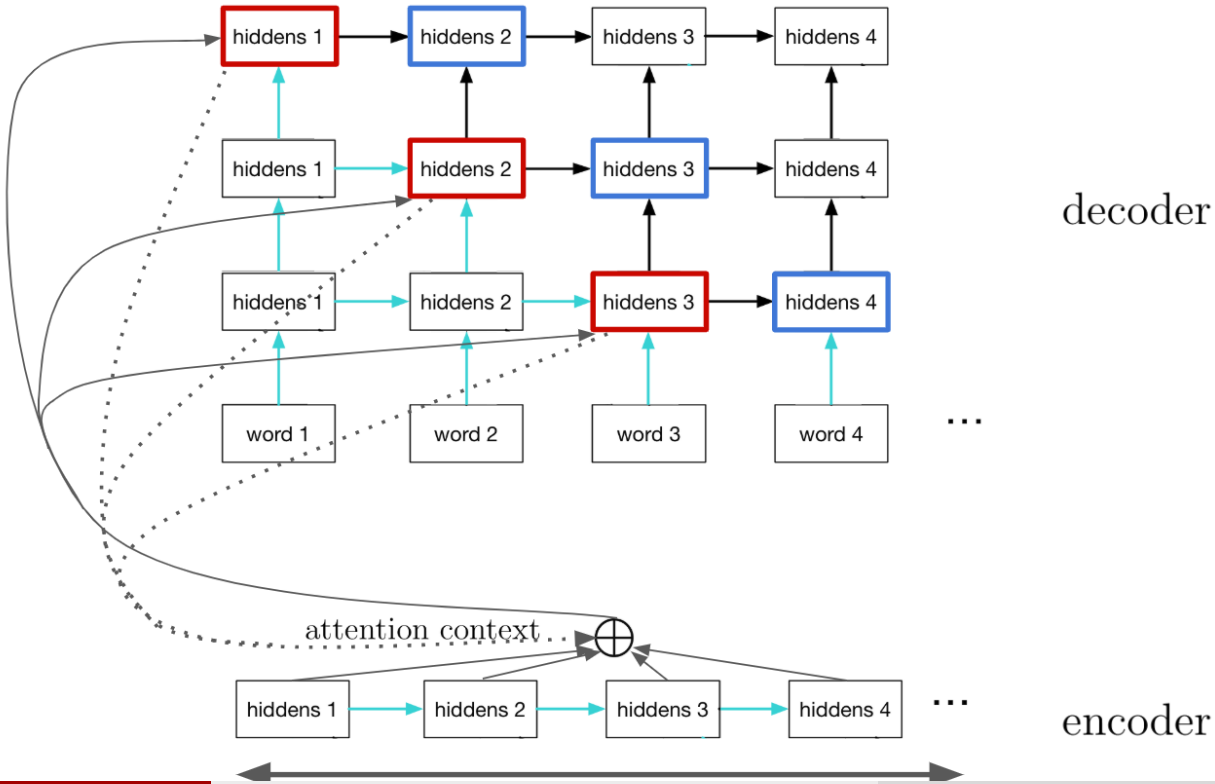
- During backprop, in the standard encoder-decoder RNN, the maximum path length across time is the number of time steps.
- Attention-based RNNs have a constant path length between the encoder inputs and the decoder hidden states.
  - Learning becomes easier. Why?





Computational Cost and Parallelism

- During a 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 the 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.

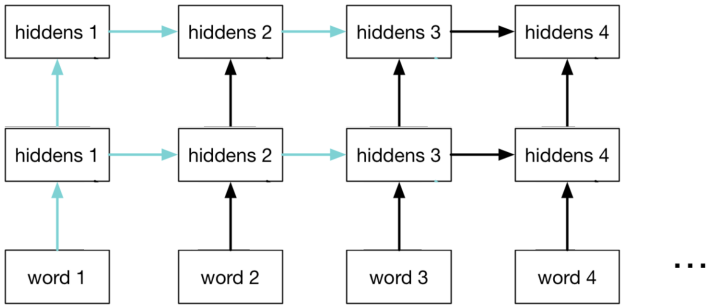
Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$

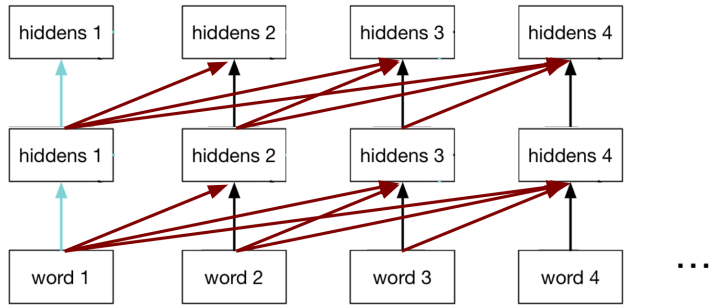
- Attention needs to re-compute context vectors at every time step.
- Attention has the benefit of reducing the maximum path length between long range dependencies of the input and the target sentences.

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.

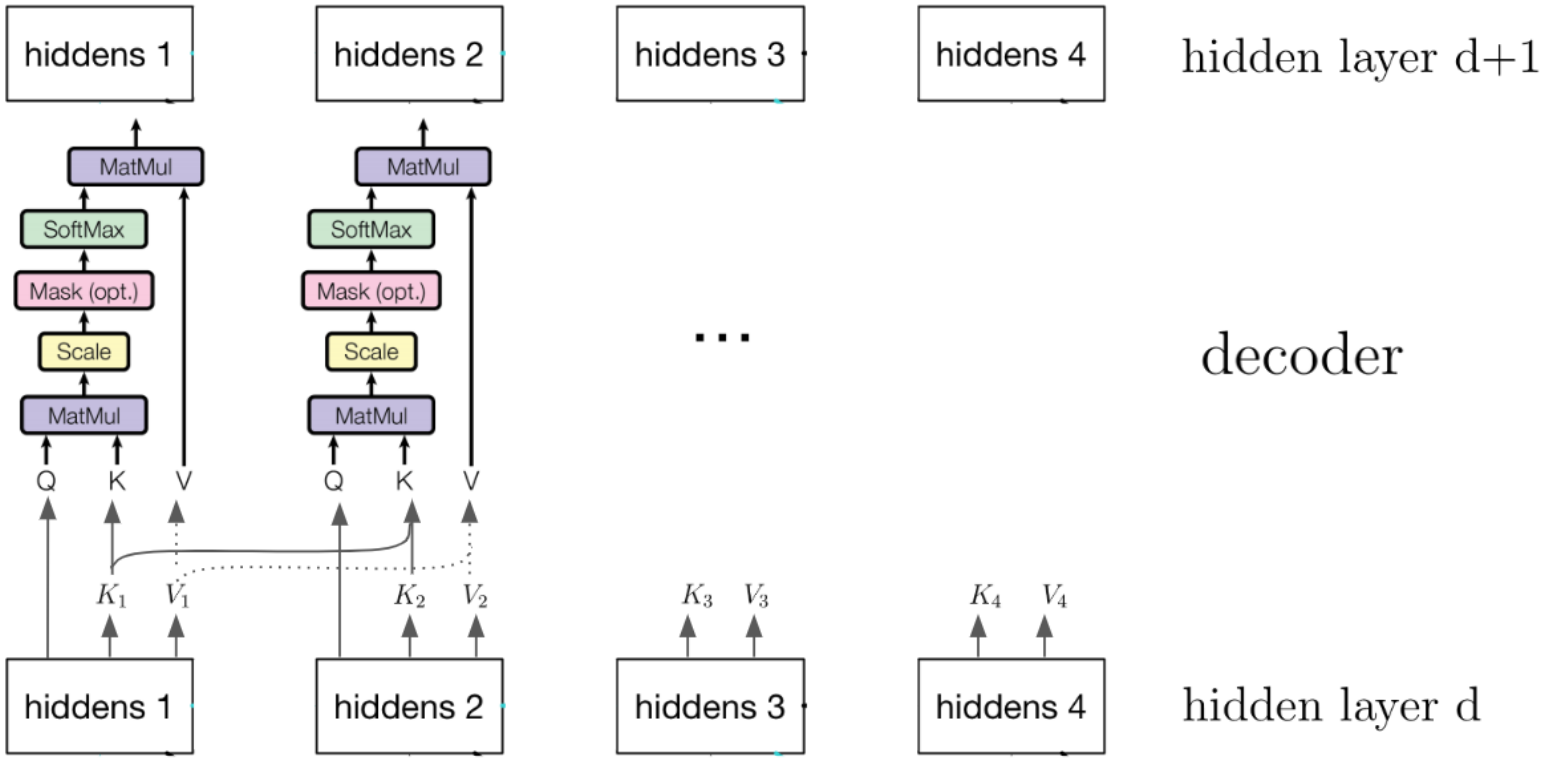




  - Benefit: the number of sequential operations is now linear in the depth d, but is independent of the sequence length t. (usually $d << t$.)

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.

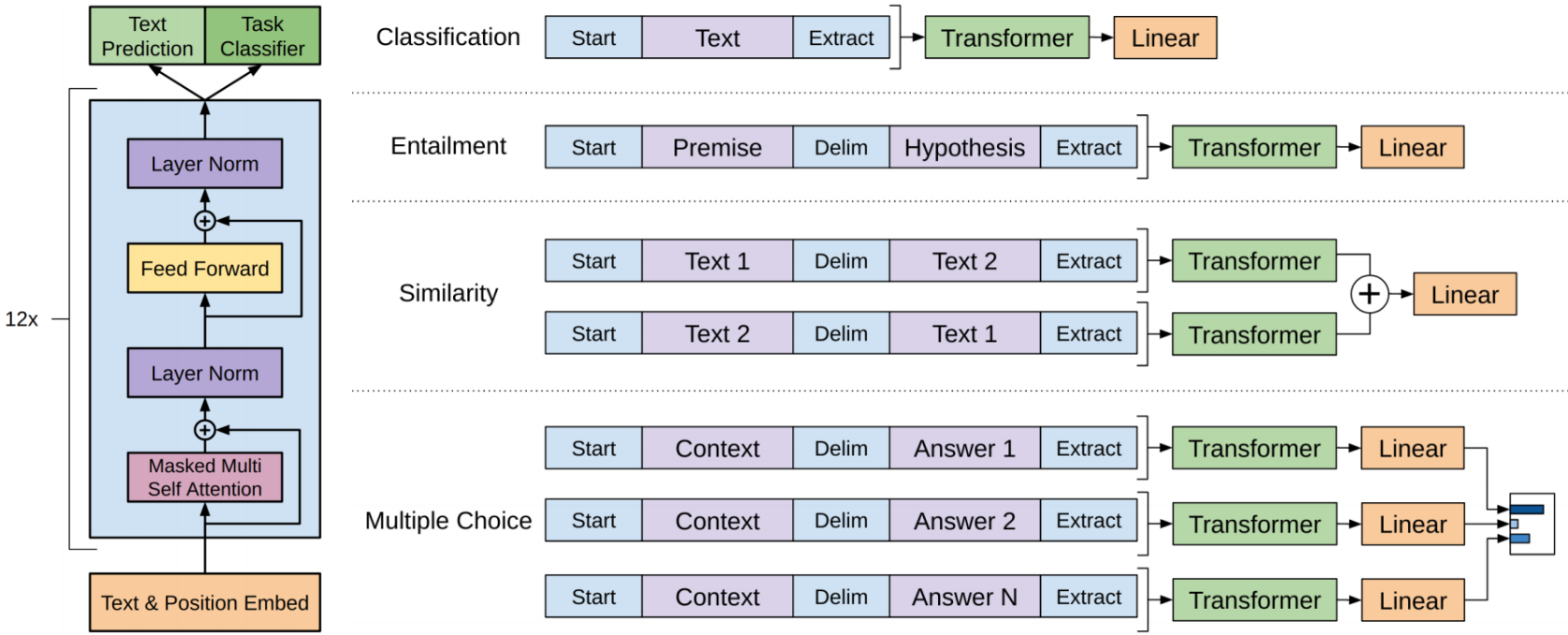
Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$
transformer	$t^2 \times k \times d$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

- **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.

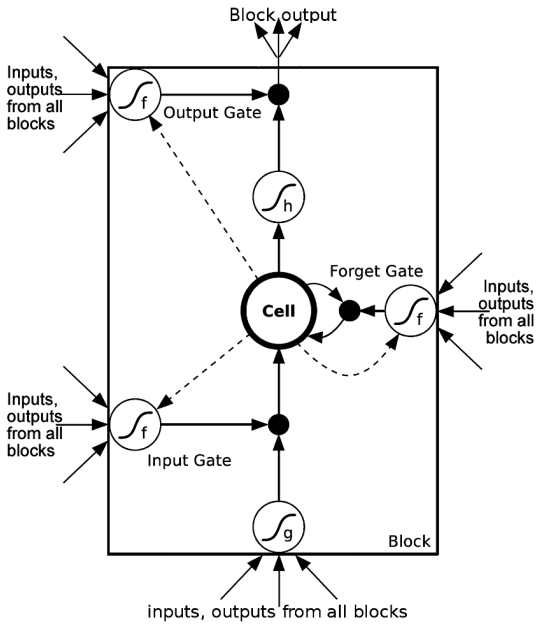
Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

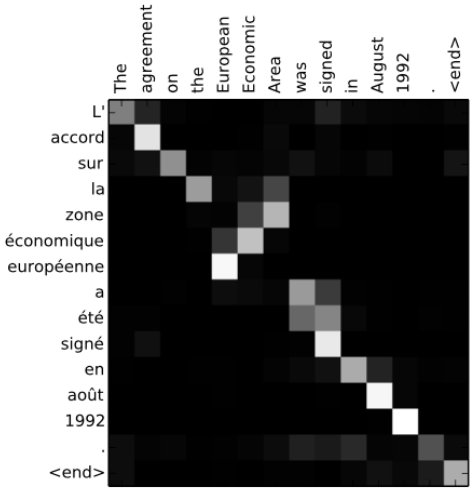
Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

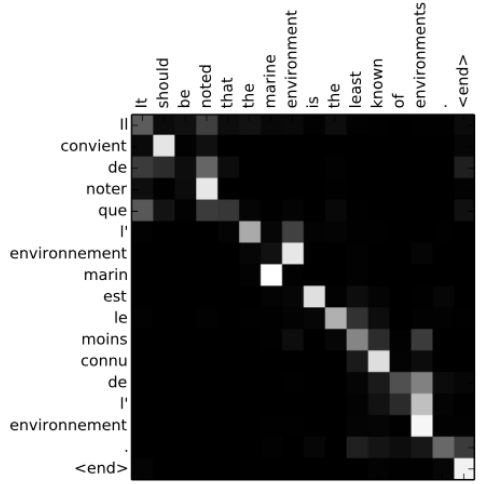
For the full text samples see Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners." 2019.

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:



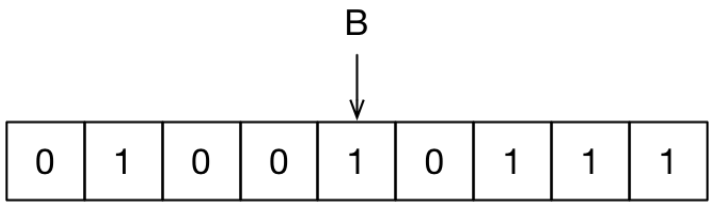




- Computers have a huge memory, but they only access a handful of locations at a time. Can we make neural nets more computer-like?

Neural Turing Machines (optional)

- Recall Turing machines:

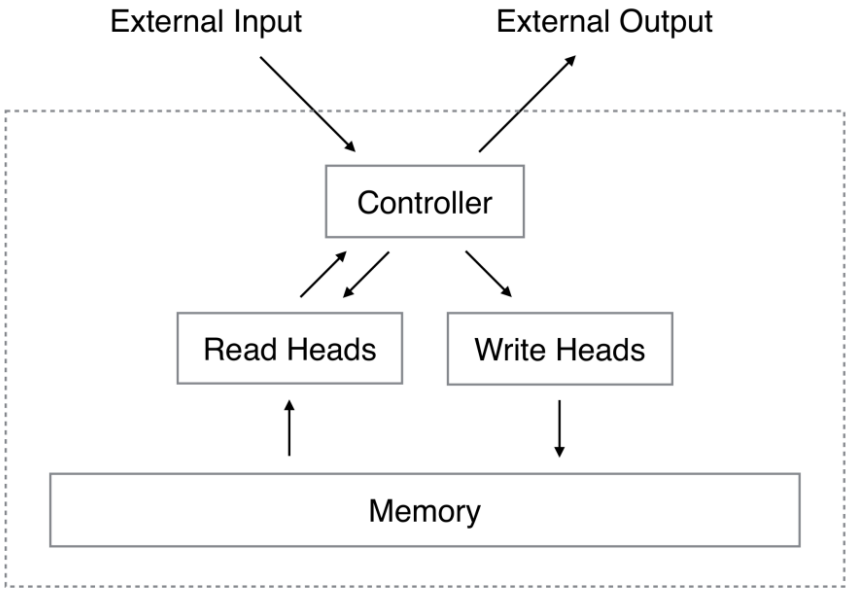


$$\langle A, 0, 0, B, \rightarrow \rangle$$
$$\langle A, 1, 0, A, \leftarrow \rangle$$
$$\ldots$$

- 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.

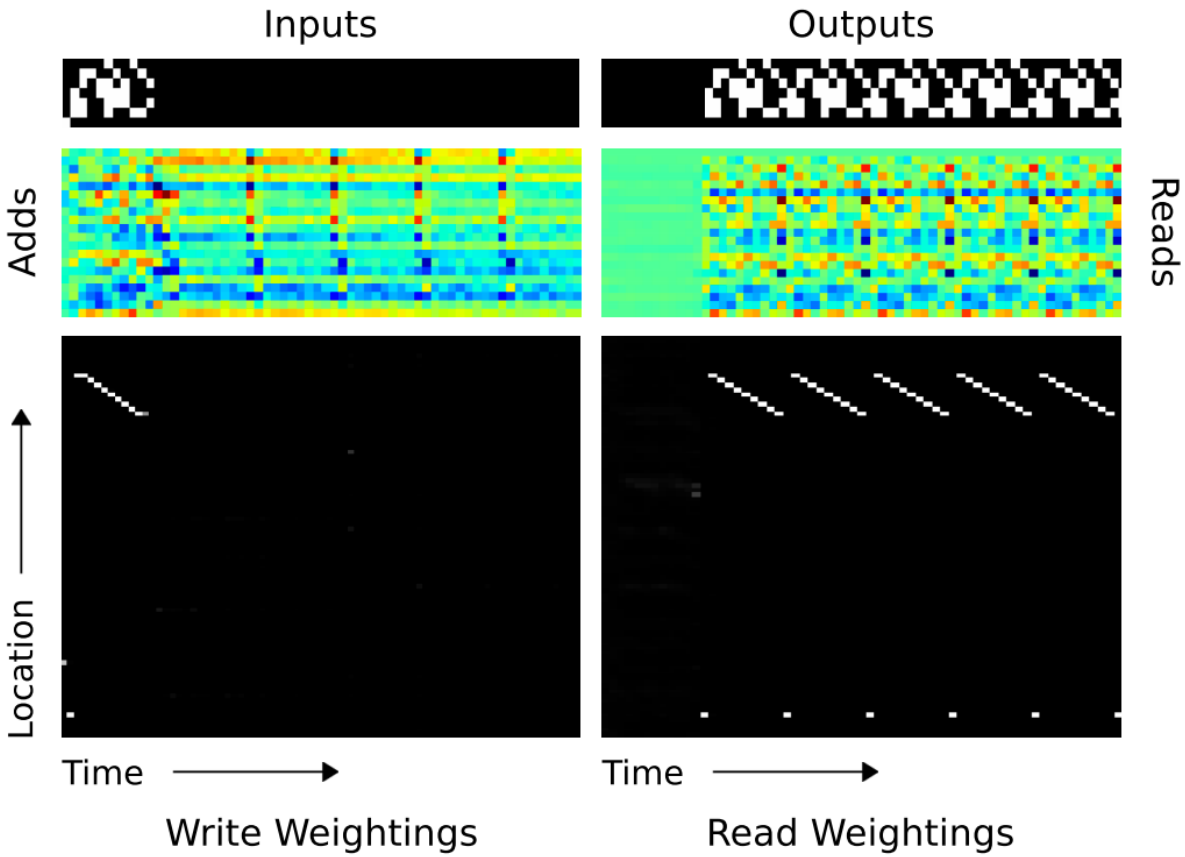
Neural Turing Machines (optional)

- 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.



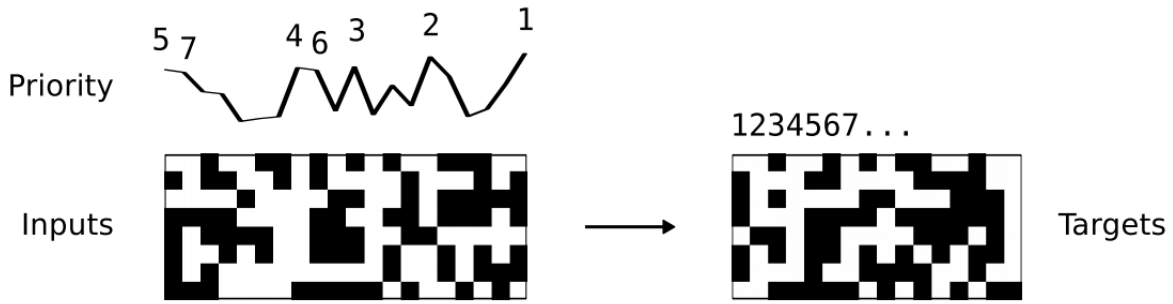
Neural Turing Machines (optional)

- 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: