Deep learning for sequence processing: Section 9.4, Speech and Language Processing, 3rd edition draft, Jurafsky & Martin (2020).

9.1: Self-Attention & Transformers

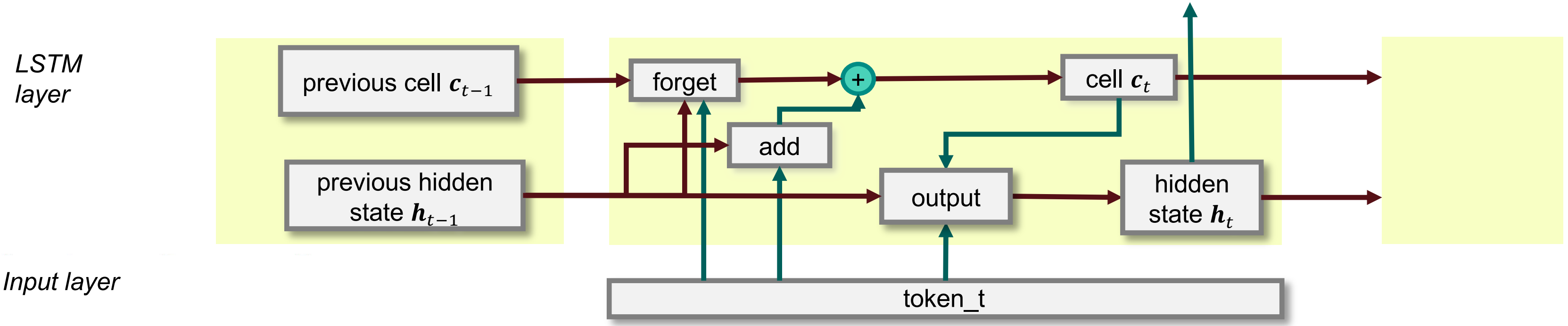
Edwin Simpson

Department of Computer Science,

University of Bristol, UK.

Recap: LSTMs

- Sequential text processing allows a model to interpret each token within the context of a sentence and compose the meaning of the sequence from the individual words.
- LSTMs are a type of RNN for sequential text processing.
- The LSTM memory cell stores context information that passes along the sequence to help make the decision for each token.



Drawbacks of LSTMs

- **Scalability**: processing each token in turn means we cannot parallelise the processing of a sequence.
- **Long-range dependencies:**
  - LSTMs retain information over a longer distance than standard RNNs;
  - But context information still has to pass through each time-step until it’s needed.
  - Small changes to the cell mean information is still lost and gradients go to zero over long sequences.
- **Limited capacity of the memory cell:**
  - Complex sentences may require the memory to retain multiple pieces of information;
  - But a single memory cell may forget earlier context that becomes important later on.

Connections in Neural Networks

- The drawbacks of LSTMs stem from the recurrent connections that pass information along the sequence.
- We can improve this with a new type of connection!
- So far we have encountered two types of neural network layers that use different kinds of connection:

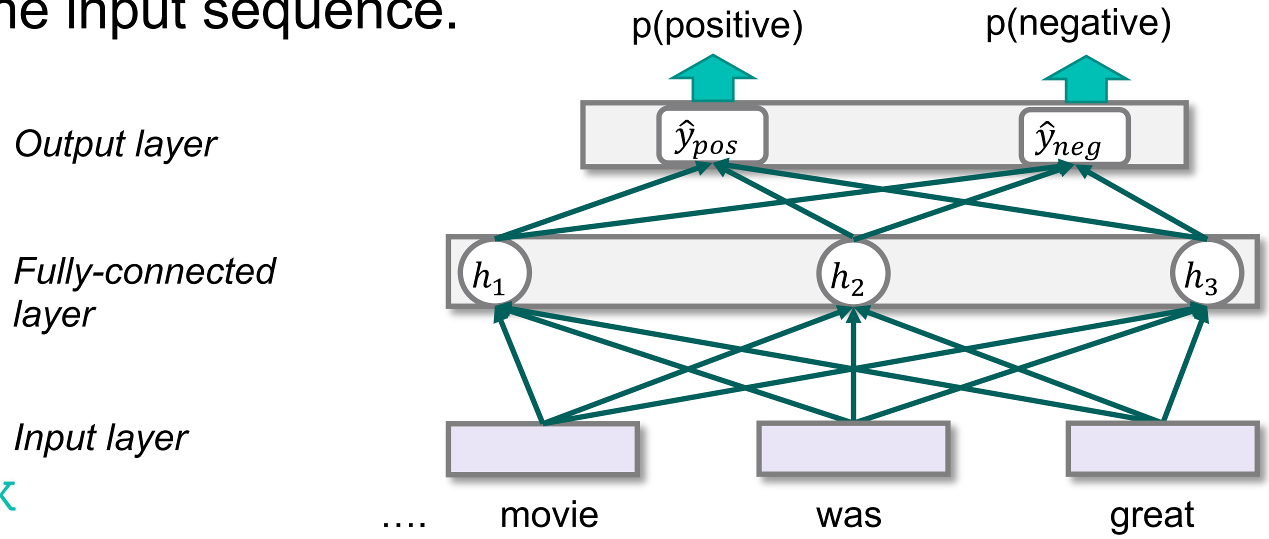
Which two types of layers have we looked at so far?

Connections in Neural Networks

- The drawbacks of LSTMs stem from the recurrent connections that pass information along the sequence.
- We can improve this with a new type of connection!
- So far we have encountered two types of neural network layers that use different kinds of connection:
  - Fully-connected layers with feed-forward connections.
  - Recurrent layers with recurrent connections.
- The connections in both types of layer are fixed when we design the model.

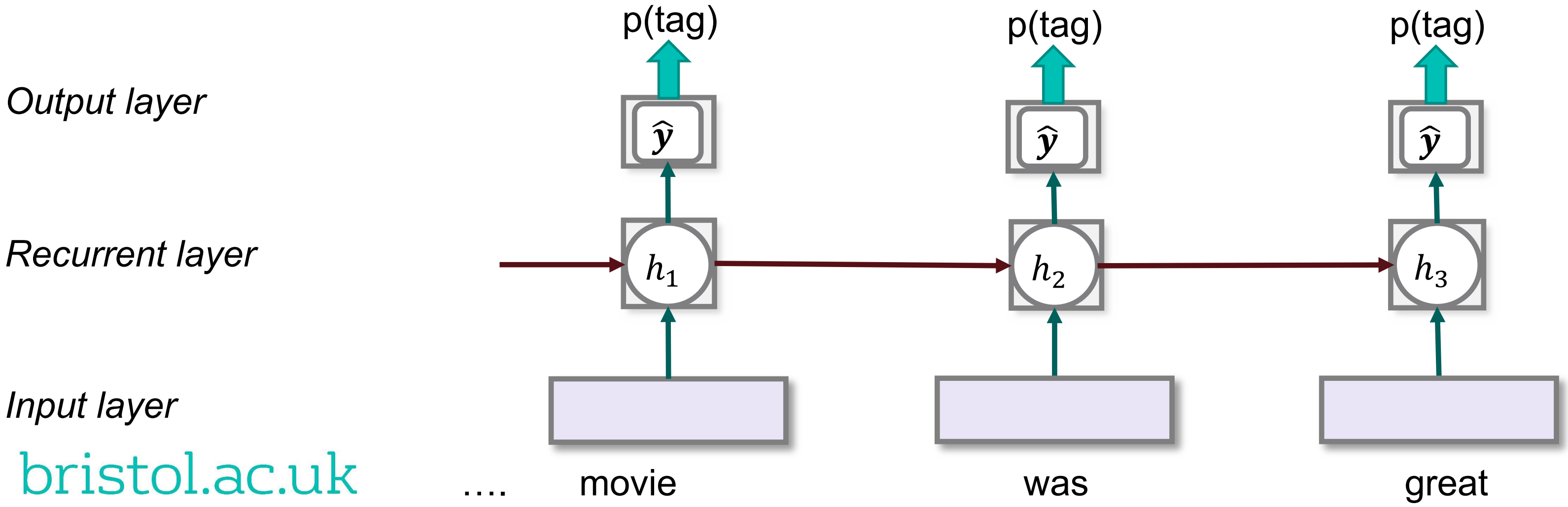
Fully-Connected (Dense) Layers

- Nodes in a fully-connected layer are connected to every node in the previous layer (here, this is the input sequence)
- The weights that are applied to each token depend solely on its position in the input sequence.



Recurrent Layers

- Nodes in a recurrent layer are connected to the corresponding input and the output of the previous node in the sequence.
- All context information must pass along the sequence.



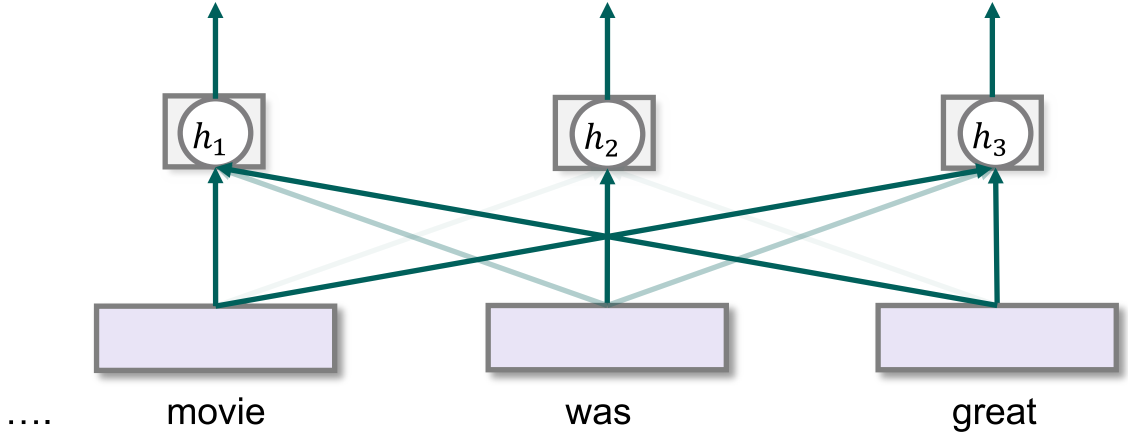
Connections To Anywhere in the Sequence?

- Ideally, the network would take **relevant context information from anywhere** in the sequence to process the current token.
- Connections would be created depending on the inputs.
- The weights applied to each input would then adapt to the inputs, rather than remaining fixed like a fully-connected layer.
- Direct connections mean shorter paths for context information to follow compared to RNNs.
- This can be achieved using **self-attention** layers.

Self-Attention

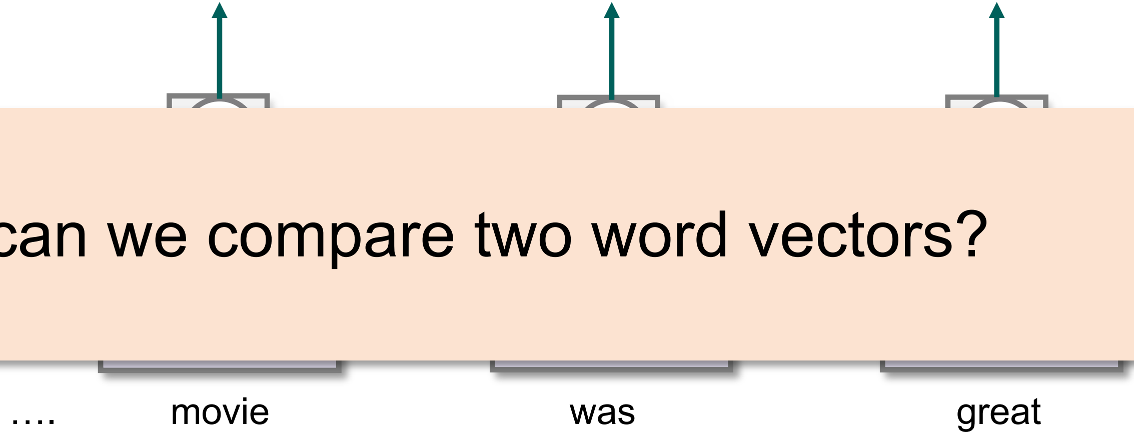
Layer n: Self attention layer

Layer n-1: Input layer



- **Attention** weights the connections between a sequence of values in layer *n-1* and a sequence of nodes in layer *n.*
- The strengths of the connections are determined by comparing the values at each position in layer *n-1* (the input sequence) with each other.

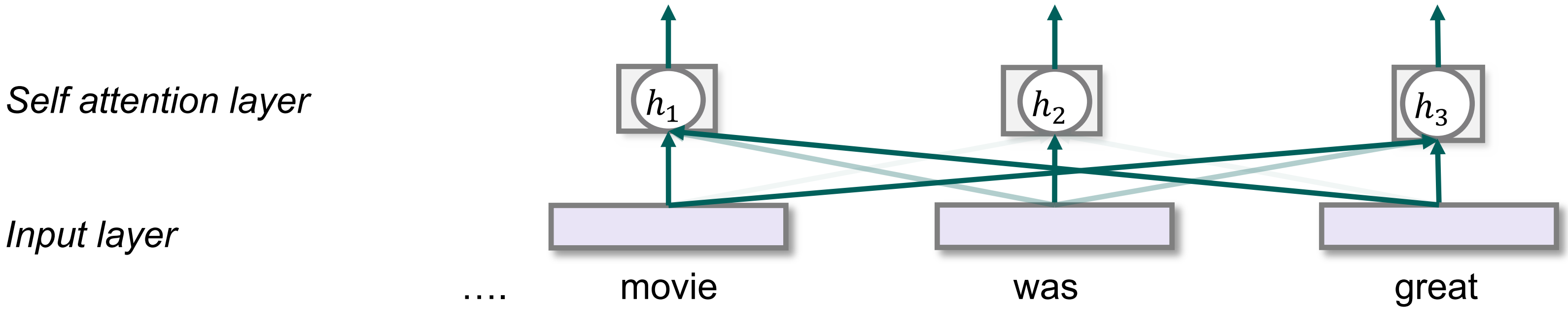
Self-Attention



How can we compare two word vectors?

- **Attention** weights the connections between a sequence of values in layer *n-1* and a sequence of nodes in layer *n.*
- The strengths of the connections are determined by comparing the values at each position in layer *n-1* (the input sequence) with each other.

Self-Attention

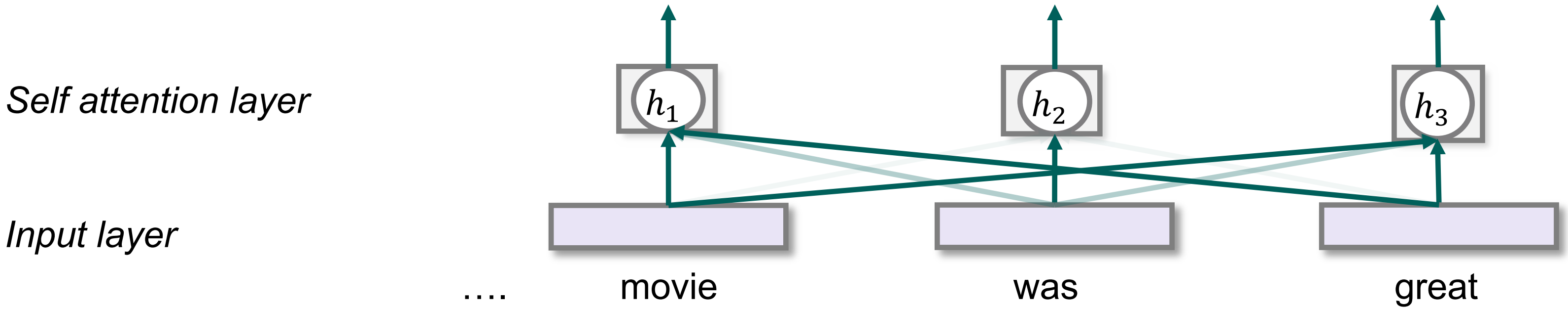


- Use dot product to compare the vectors for of two nodes in *n-1:*

$$score(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$

- We also use dot product to compare vectors using cosine similarity.
- But this score is based only on the input vectors!
- It provides no way to learn how to make the connections between words in a sequence.

Self-Attention



- Use dot product to compare the vectors for of two nodes in *n-1:*

$$score(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{W}^q \boldsymbol{x}_i \cdot \boldsymbol{W}^k \boldsymbol{x}_j$$

- $\boldsymbol{W}^q$ and $\boldsymbol{W}^k$ are matrices of weights that can be learned.
- Now we can learn how to words relate to each other in a sequence.

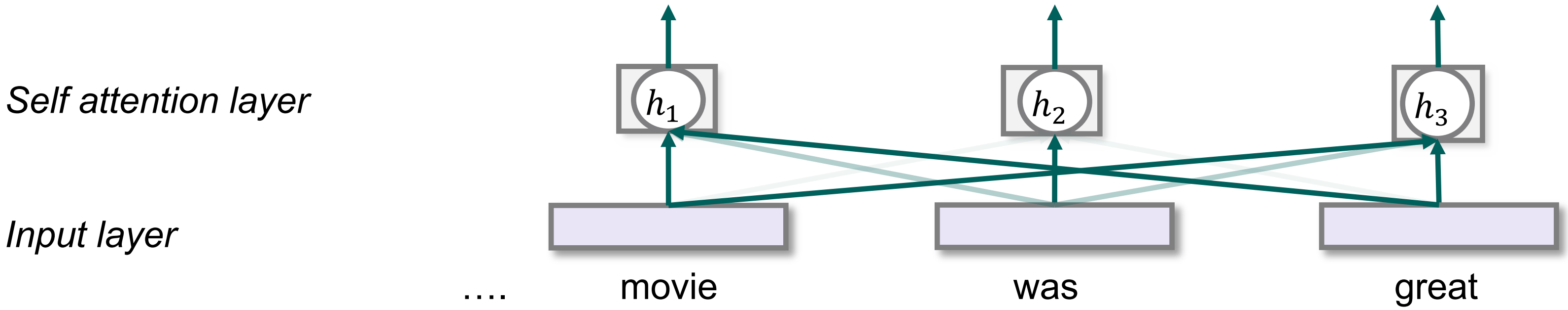
Self-Attention

- Scale and apply the softmax function to turn the score into attention weights that sum to 1:

$$\alpha(i,j) = \text{softmax}\left(\frac{\boldsymbol{W}^q \boldsymbol{x}_i \cdot \boldsymbol{W}^k \boldsymbol{x}_j}{\sqrt{d}}\right)$$

- $\alpha(i,j)$ indicates the proportion of attention we put on $\boldsymbol{x}_j$ in layer *n-1* when determining the hidden state $\boldsymbol{h}_i$ in layer *n* for token *i.*
- d is the number of dimensions of the embeddings in layer *n-1*

Self-Attention

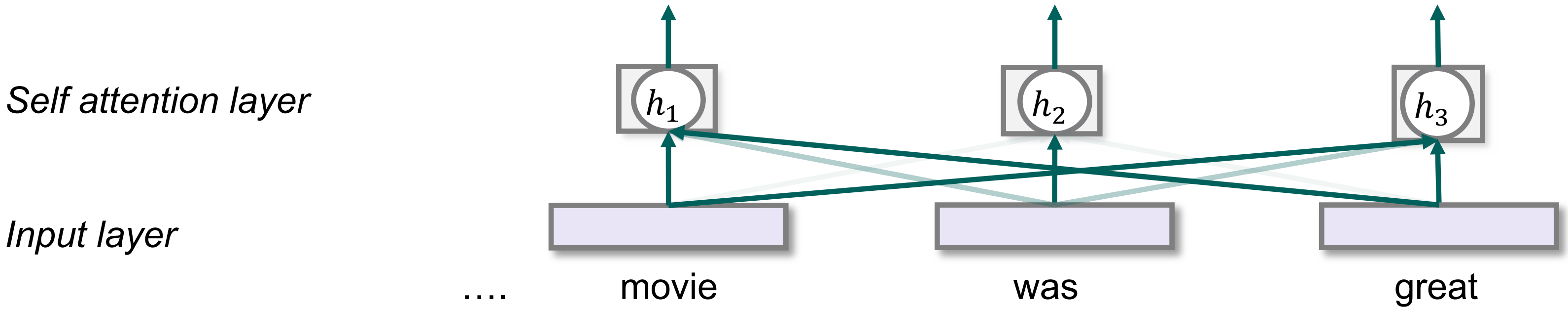


- Compute the hidden state $\boldsymbol{h}_i$ for layer *n* as a weighted sum over the input sequence:

$$\boldsymbol{h}_i = \sum_{j=1}^{N} \alpha(i,j)\boldsymbol{W}^v \boldsymbol{x}_j$$

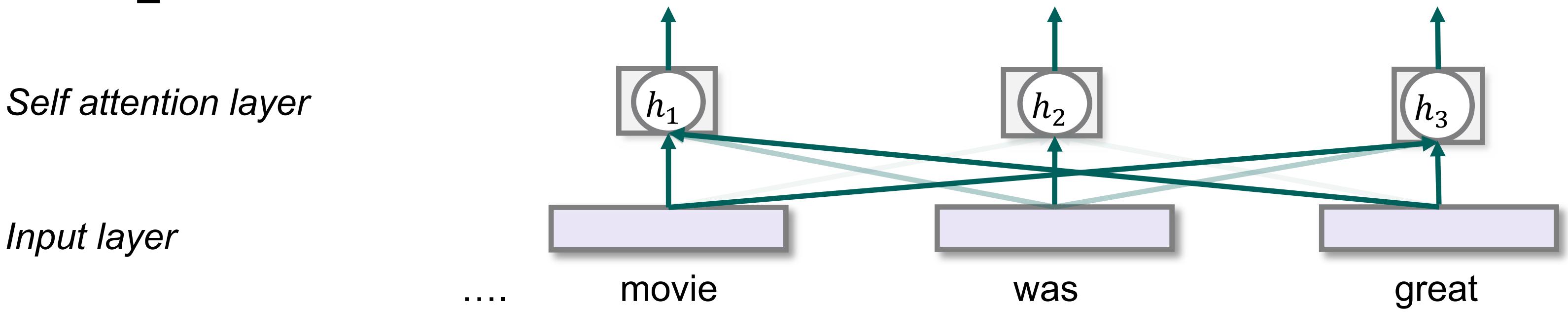
- $\boldsymbol{W}^v$ is a matrix of learnable weights (equivalent to $\boldsymbol{W}$ in the RNN)

Self-Attention



- Self-attention allows the model to select the relevant parts of the context when processing each token.
- Each hidden state $\boldsymbol{h}_i$ is computed independently, so can be computed in parallel.

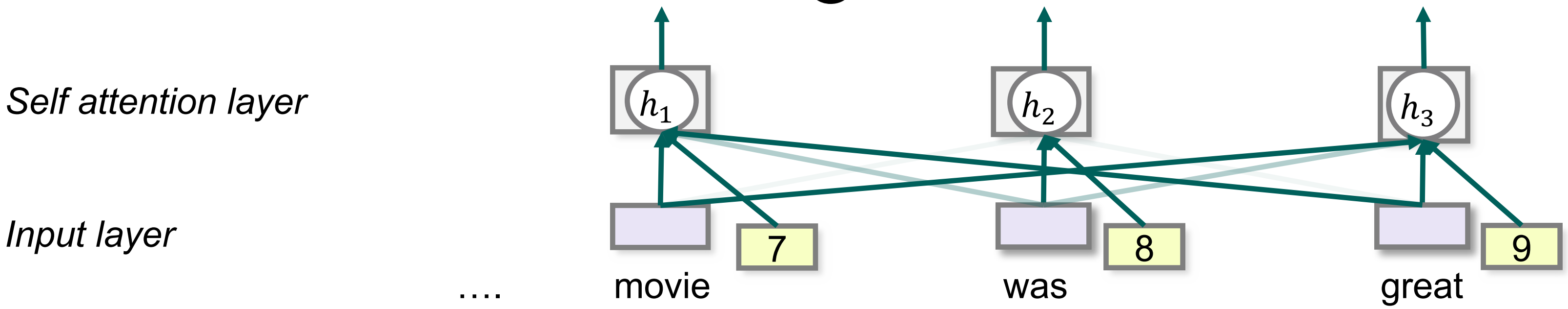
Sequential?



- The self-attention mechanism examines the whole sequence at once and doesn't know anything about word order…

If we ignore word order, what kind of model do we have?

Positional Embeddings



- The self-attention mechanism examines the whole sequence at once and doesn't know anything about word order…
- We're back to a bag of words!
- To provide information about word order, concatenate **positional embeddings** to the input embeddings.
- Positional embeddings are vectors that encode the position of each token in the sequence

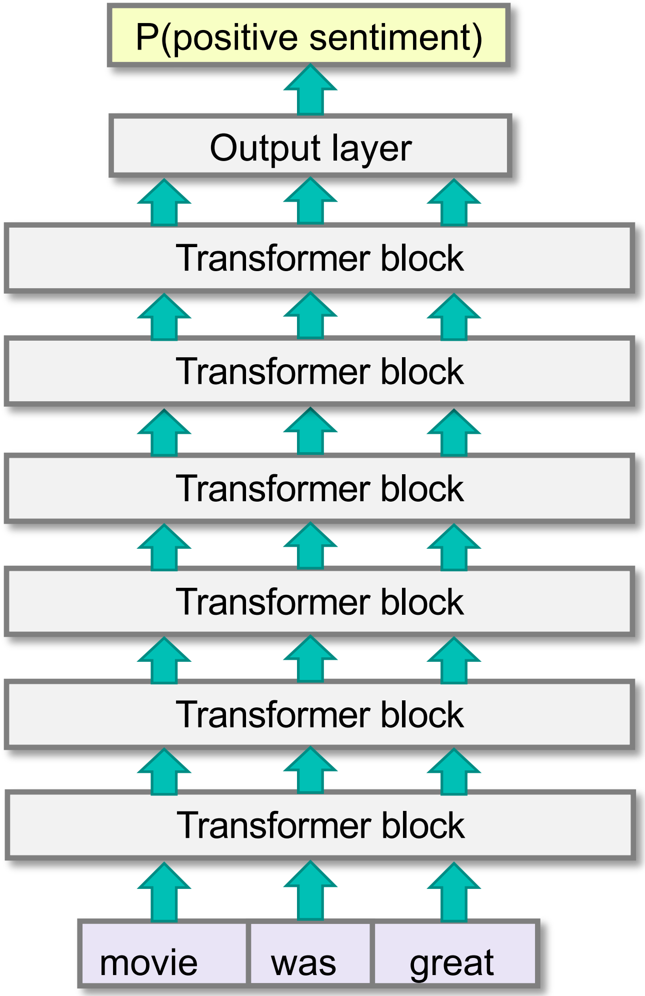
Building a Model with Self-Attention

- In the NLP pipeline, we saw how successive processing steps extract syntactic and semantic information at different levels.
- A self-attention layer composes information at only one level of abstraction, so a single layer is not very powerful on its own.

How can we emulate this compositional behaviour with a neural network?

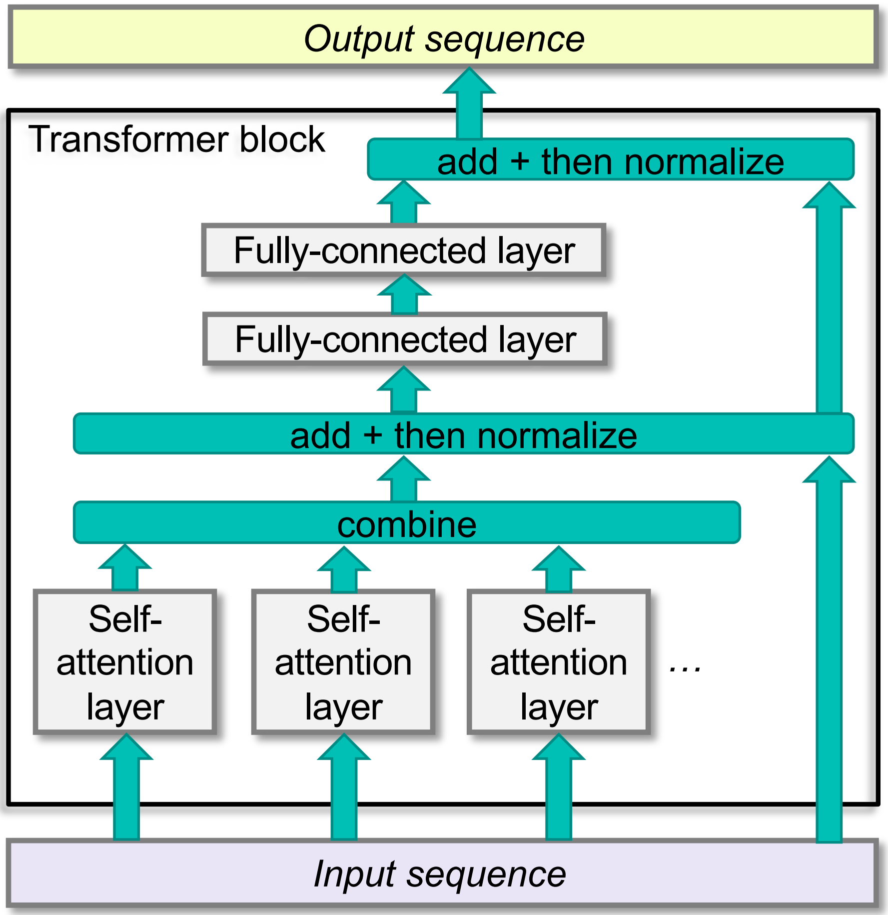
Transformers

- Stack multiple self attention layers to compose features at different levels of abstraction
- In a transformer, the self-attention layers are contained inside **transformer blocks**
- A transformer is a stack of transformer blocks →



Transformer Blocks

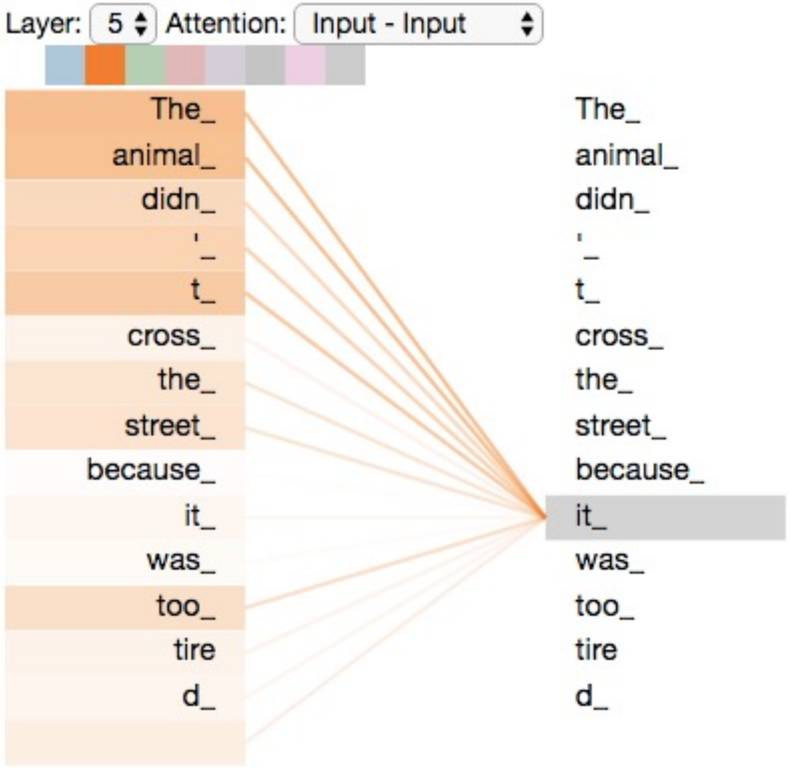
- Several self attention layers run in parallel: **multi-headed attention**
- Each attention head models a different kind of relation between the words in the input sequence.
- Fully-connected layers learn nonlinear functions



More on Transformers

- By avoiding recurrent connections, it is possible to train very large, deep transformers that excel at a wide range of language understanding and generation tasks
- More on transformers in the next video when we show how they can be used to provide contextualised embeddings
- See also:
  - Attention is all you need, Vaswani et al., 2017
  - The illustrated Transformer

Self-Attention Example

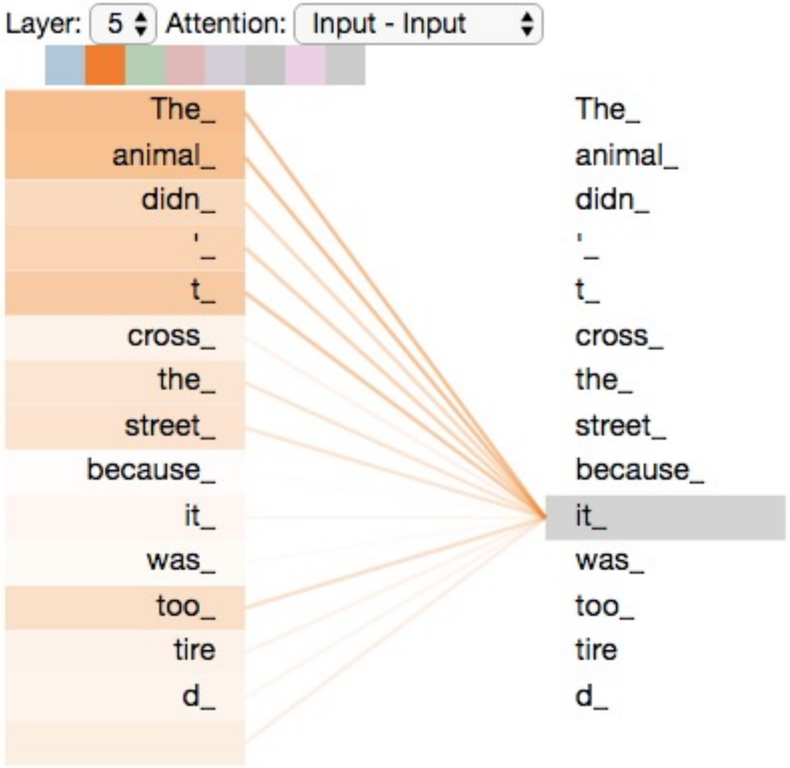


- The colours show the attention weights when processing the token ‘it_’.

Can you think why certain input words have stronger attention weights for ‘it_’?

Could this be useful in a task like relation extraction?

Self-Attention Example



- The colours show the attention weights when processing the token ‘it_’.
- Strongest weights are on ‘the animal’.
- Self-attention resolves ‘it’ to ‘the animal’ so the network can determine an encoding of ‘it’ that captures its meaning in this context.

Summary

- Parallel computation is not possible with RNNs and their handling of long-range dependencies is still limited.
- Self-attention layers allow the network to select the relevant context words from anywhere in the sequence and take a weighted sum of their values.
- Positional embeddings ensure that word order is considered.
- Transformers arrange many self-attention layers into large, deep models that extract different kinds of syntactic and semantic information and are highly effective in many tasks.