COMP4650/6490 Document Analysis

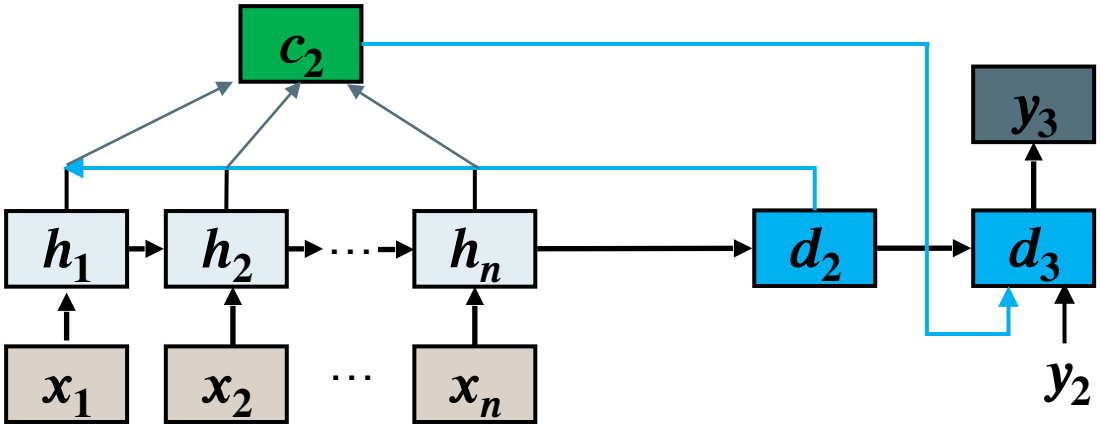
Transformers

ANU School of Computing

Recap

- RNNs to process sequential data
- Encoder-Decoder models for sequence-to-sequence mappings
- Attention helps the decoder

Recap



$$\tilde{a}_{j,i} = \text{score}(\boldsymbol{d}_j, \boldsymbol{h}_i),\ i = 1, \ldots, n$$

$$\boldsymbol{a}_j = \text{softmax}(\tilde{\boldsymbol{a}}_j)$$

$$\boldsymbol{c}_j = \sum_{i=1}^{n} a_{j,i} \boldsymbol{h}_i$$

$$\boldsymbol{d}_{j+1} = \text{Decoder}(\boldsymbol{d}_j, [\boldsymbol{c}_j, \boldsymbol{y}_j])$$

Transformers

Vaswani et al.
Attention is All you Need.
NeurIPS. 2017.

It was found that the recurrent part of the model was not necessary if you use a more sophisticated type of attention and some other tricks.

Attention Is All You Need

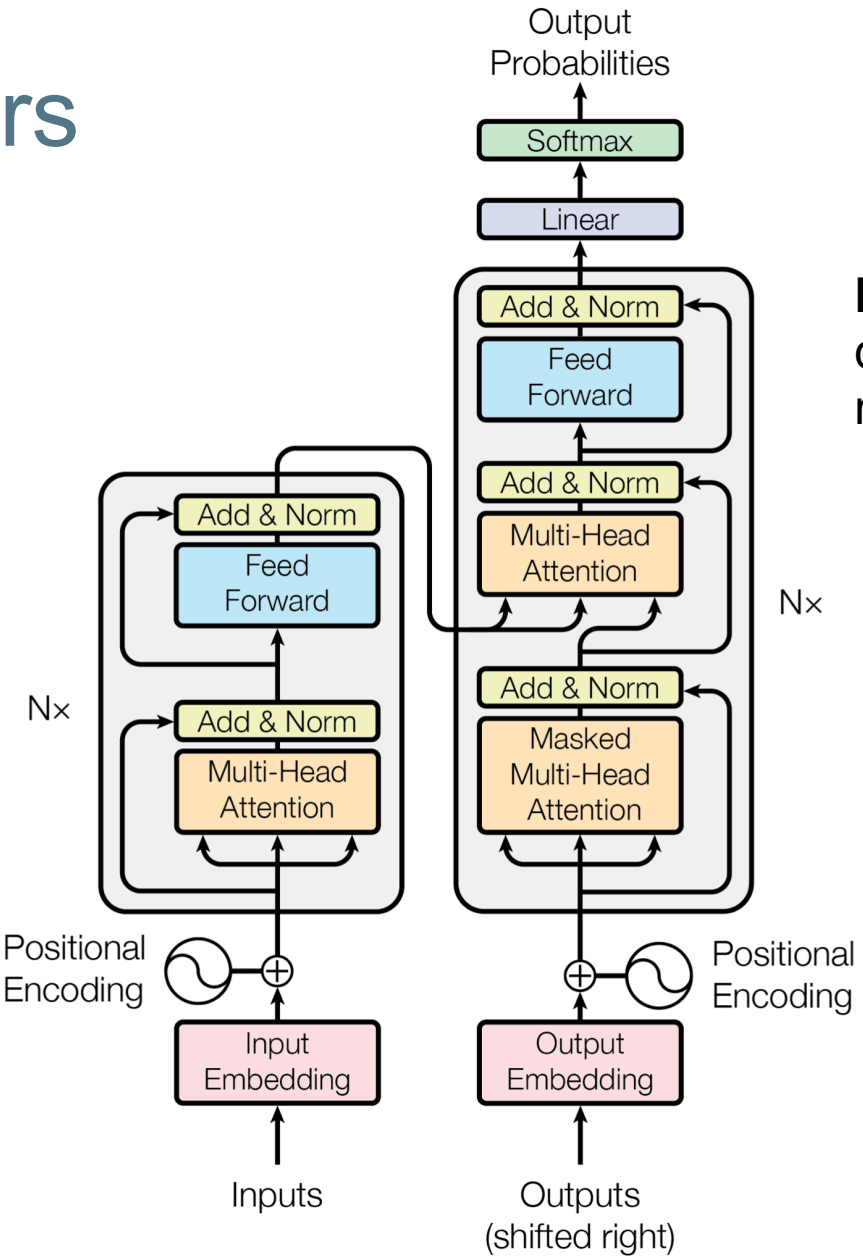
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

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

Transformers

Encoder:
encode the meaning in input



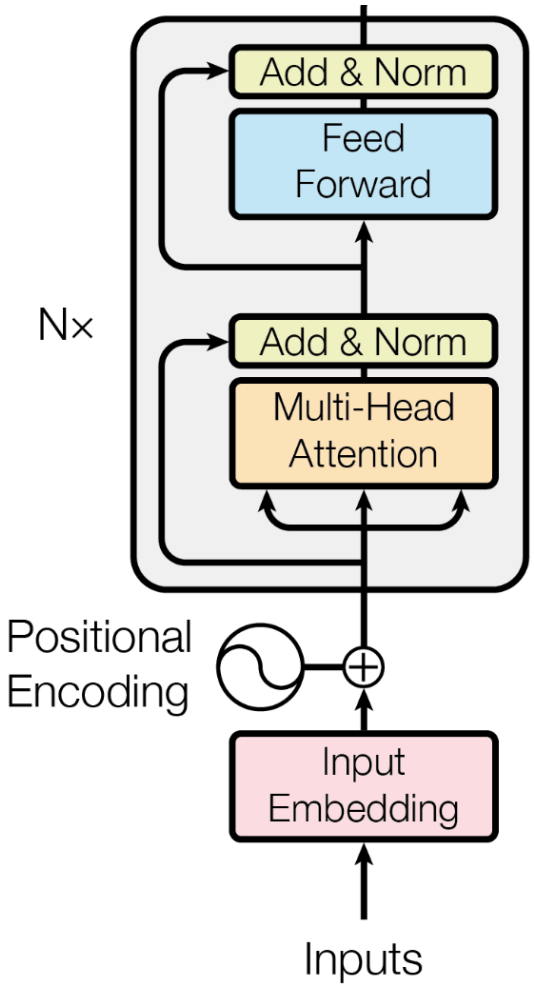
Decoder:
decode the meaning to output

Transformers

- Transformers do not do any recurrent processing of inputs. Instead, they use *self-attention* to create representations of the input sequence.
  - In self-attention (for encoder), every input vector is used as a query over all input vectors.
  - In contrast standard attention uses the previous decoder output as query over all encoder outputs.
- Transformers use *Query-Key-Value* attention.
- State-of-the-art models for text processing are now almost always transformer based.

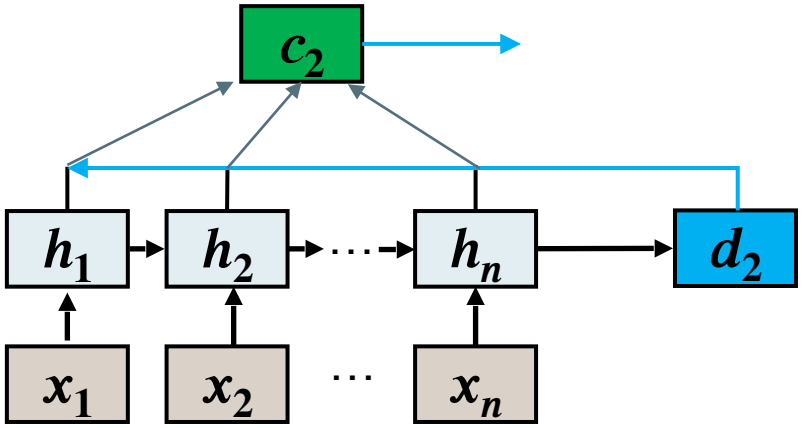
Transformer encoder

- Input embedding
- Positional encoding
- Multi-head self-attention
- Fully connect feed-forward network
- Residual connection
- Layer normalisation

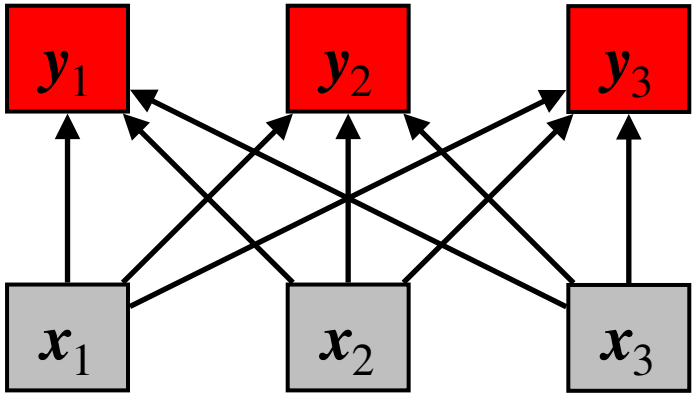


Self-attention

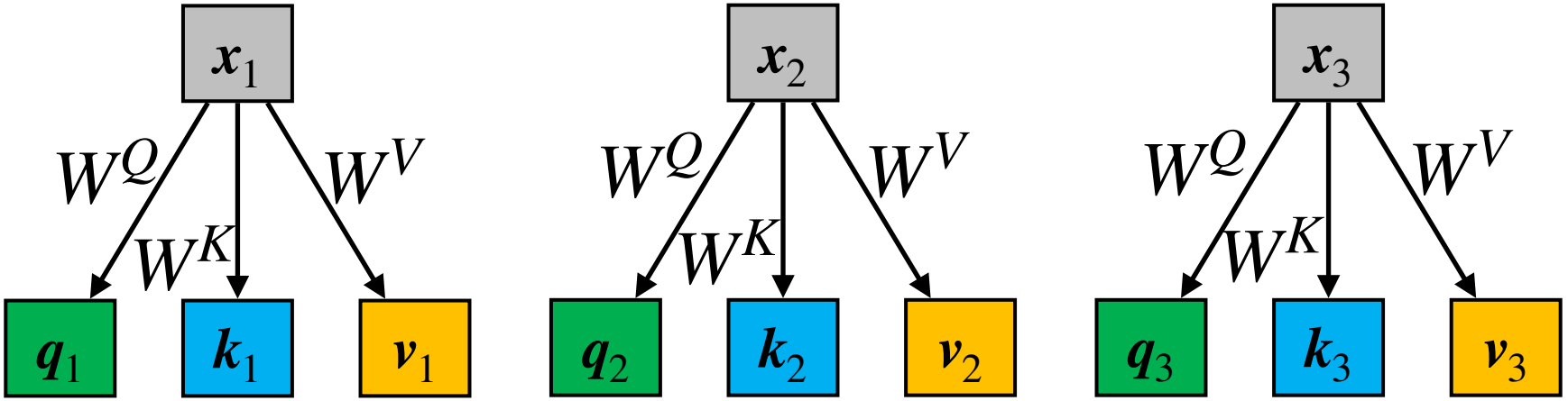
Previously we saw RNN attention:



Self-attention in transformers:



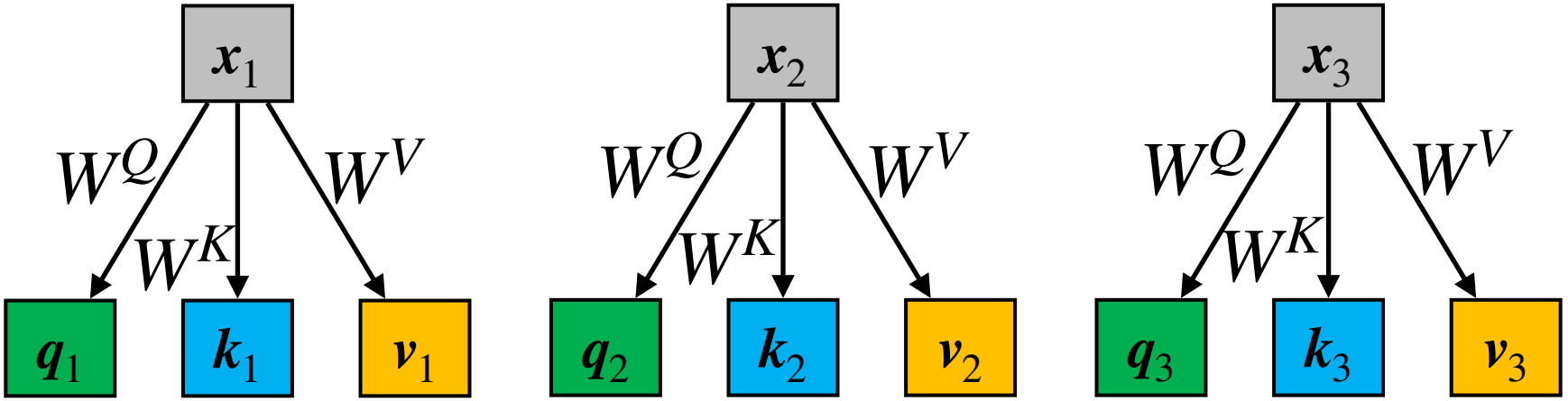
Self-attention



$$q_i = W^Q x_i \qquad k_i = W^K x_i \qquad v_i = W^V x_i \qquad i \in \{1,2,3\}$$

W^Q, W^K, W^V are learnable weight matrices that transform the inputs into query, key and value vectors.

Self-attention

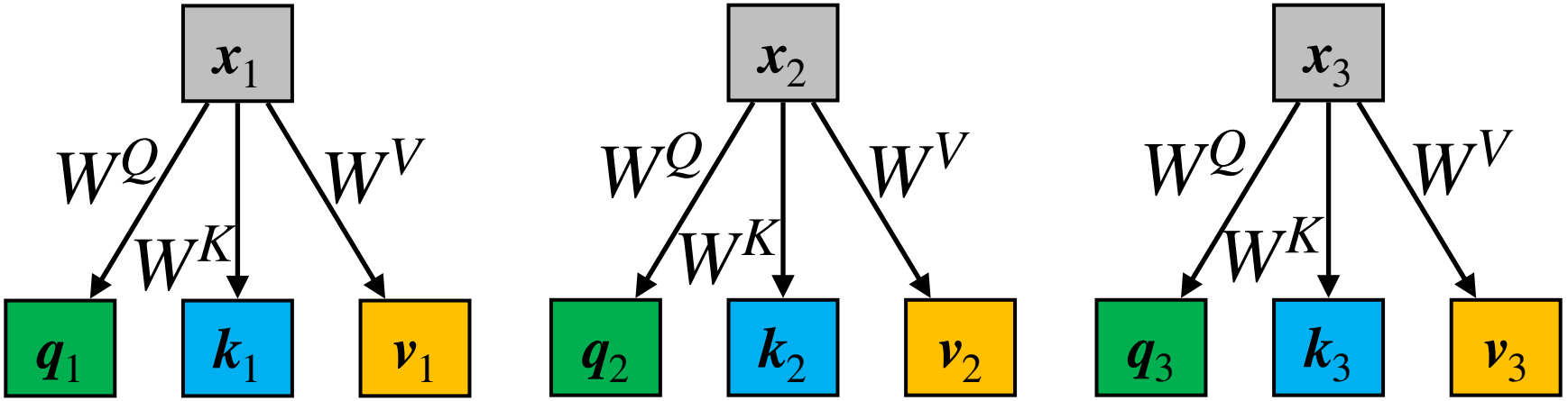


$\tilde{a}_{1,1} = \text{sim}(\boldsymbol{q}_1, \boldsymbol{k}_1)$ $\tilde{a}_{1,2} = \text{sim}(\boldsymbol{q}_1, \boldsymbol{k}_2)$ $\tilde{a}_{1,3} = \text{sim}(\boldsymbol{q}_1, \boldsymbol{k}_3)$

Typically, $\text{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j) = \boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d}$ where $\boldsymbol{q}_i, \boldsymbol{k}_j \in \mathbb{R}^{d \times 1}$

Compute *attention (weights) vector* using the *query* and *key* vectors: $\boldsymbol{a}_1 = \text{softmax}(\tilde{\boldsymbol{a}}_1)$

Self-attention



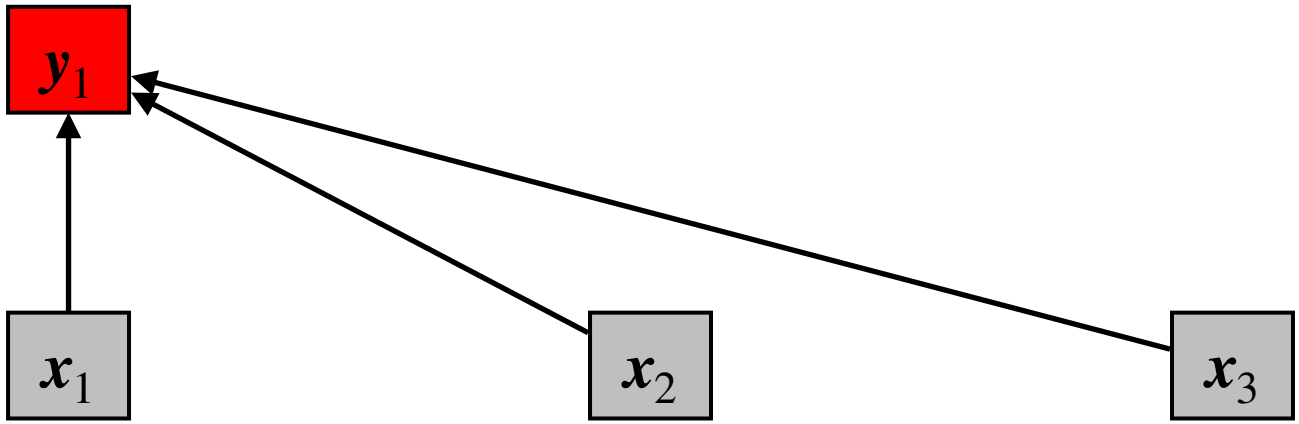
$\tilde{a}_{1,1} = \text{sim}(\boldsymbol{q}_1, \boldsymbol{k}_1)$ $\tilde{a}_{1,2} = \text{sim}(\boldsymbol{q}_1, \boldsymbol{k}_2)$ $\tilde{a}_{1,3} = \text{sim}(\boldsymbol{q}_1, \boldsymbol{k}_3)$

Compute *context vector* using the *attention weights* and *value* vectors:

$\boldsymbol{y}_1 = a_{1,1}\boldsymbol{v}_1 + a_{1,2}\boldsymbol{v}_2 + a_{1,3}\boldsymbol{v}_3$ where $\boldsymbol{a}_1 = \text{softmax}(\tilde{\boldsymbol{a}}_1)$

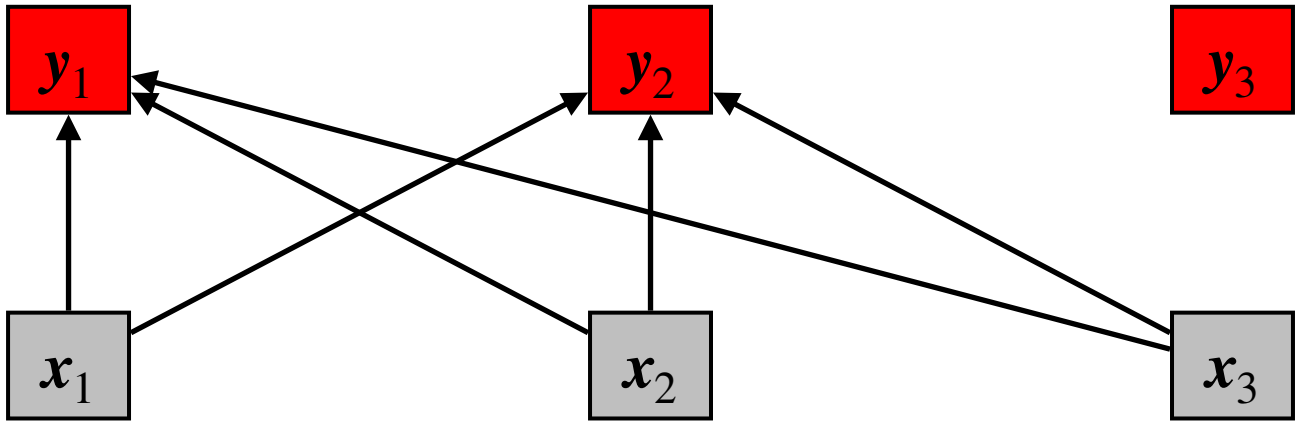
Self-attention

- We have calculated $\boldsymbol{y}_1$ — a new vector representation of $\boldsymbol{x}_1$
- We need to repeat this computation for every input $\boldsymbol{x}_i$
- In practice this can all be done in parallel



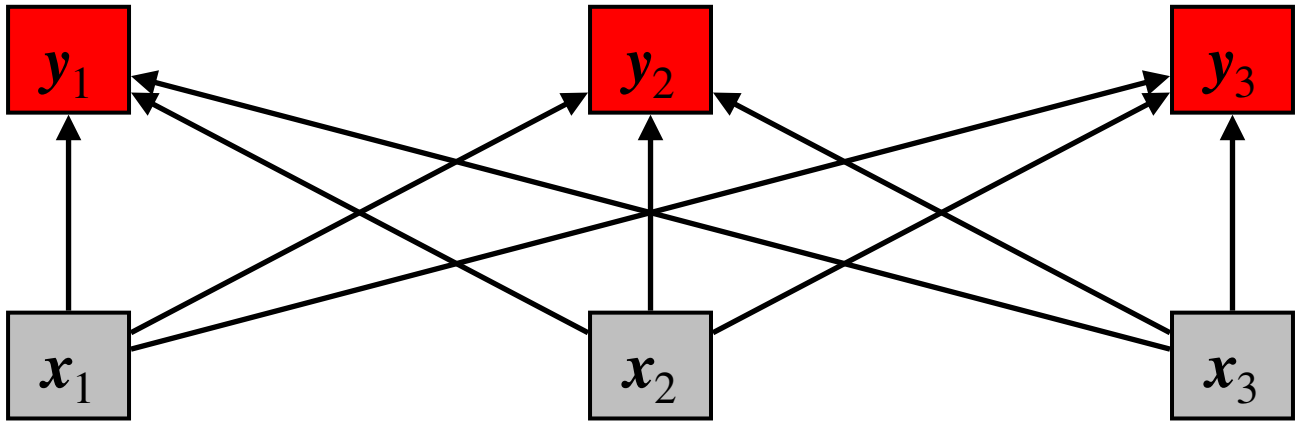
Self-attention

- We have calculated $\boldsymbol{y}_1$ — a new vector representation of $\boldsymbol{x}_1$
- We need to repeat this computation for every input $\boldsymbol{x}_i$
- In practice this can all be done in parallel



Self-attention

- We have calculated $\boldsymbol{y}_1$ — a new vector representation of $\boldsymbol{x}_1$
- We need to repeat this computation for every input $\boldsymbol{x}_i$
- In practice this can all be done in parallel

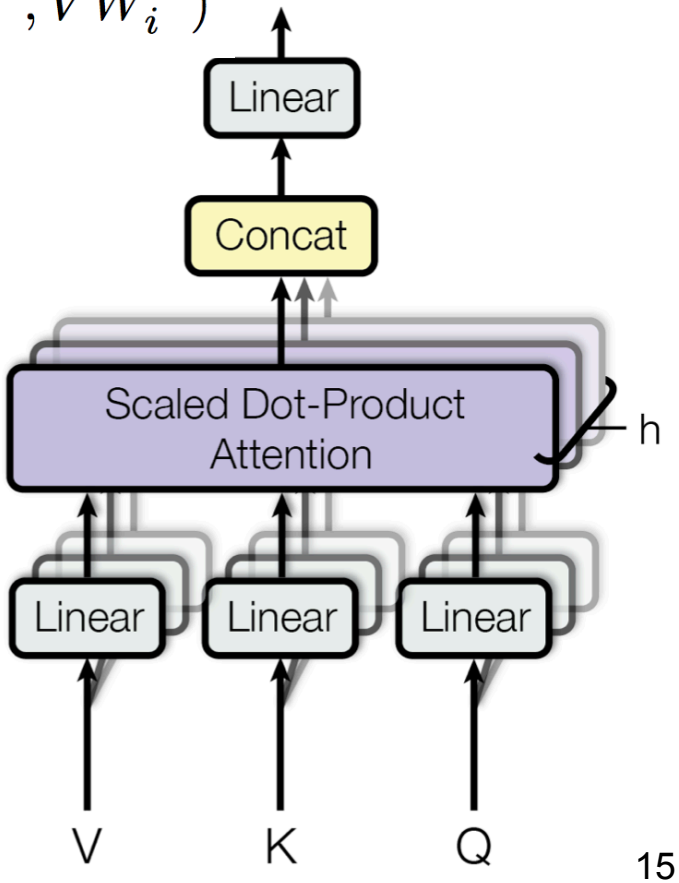


Multi-head attention

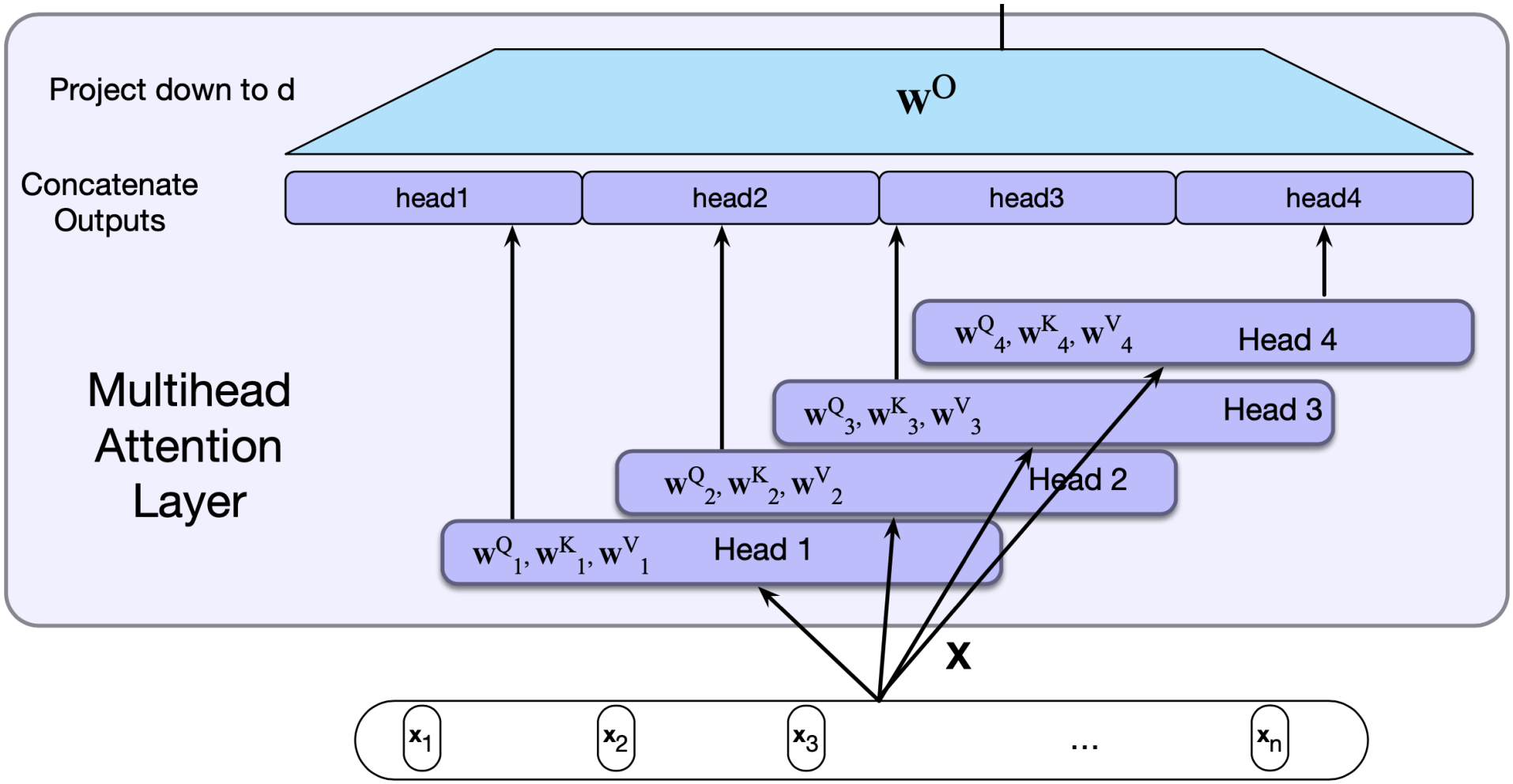
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

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

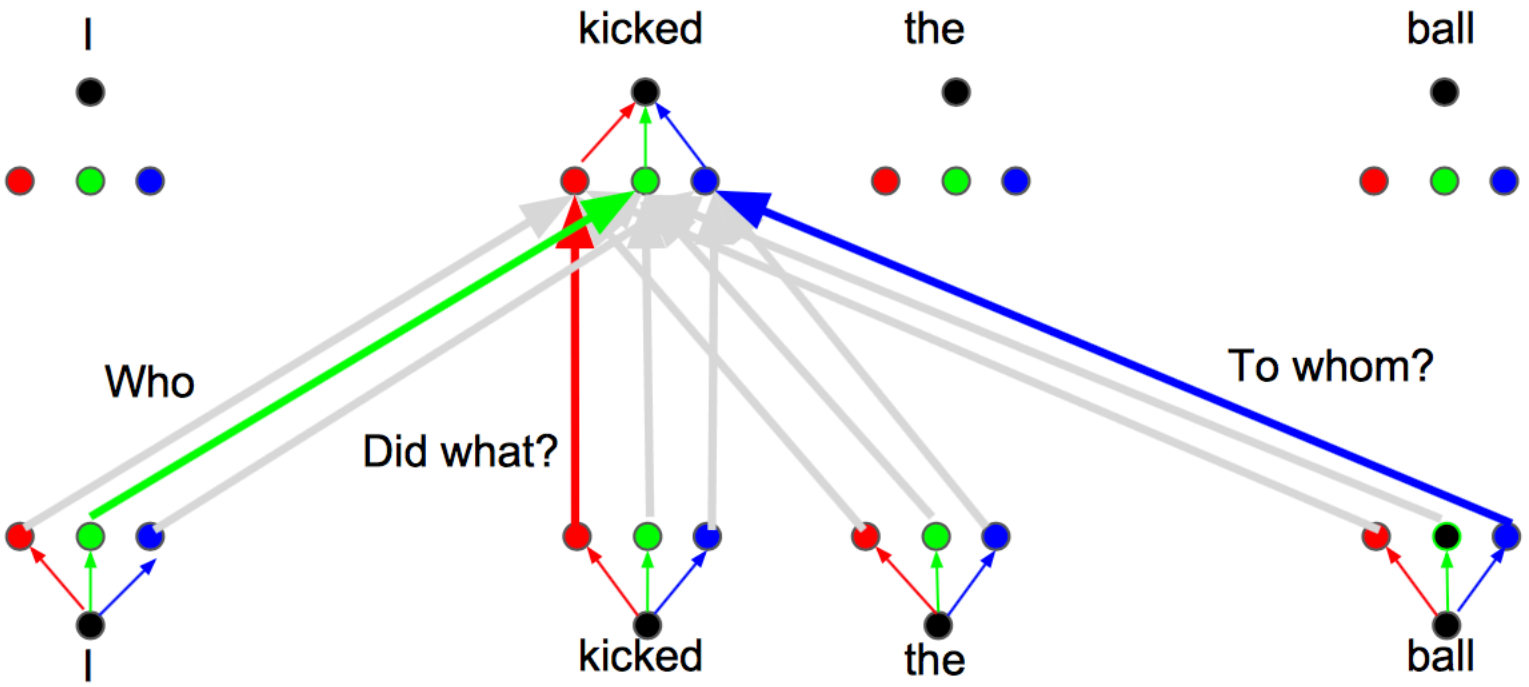
- Have multiple attention heads, so the model can focus on more input regions.
- A linear layer (W^O) combines all the attention heads
- For transformer encoder:
  $Q = K = V = X$ (the input)
- For transformer decoder:
  Similar to self-attention in the encoder but masking out subsequent terms
- For cross attention in encoder-decoder transformers:
  Q, K, V are different (query from decoder, key and value from encoder).



Multi-head attention



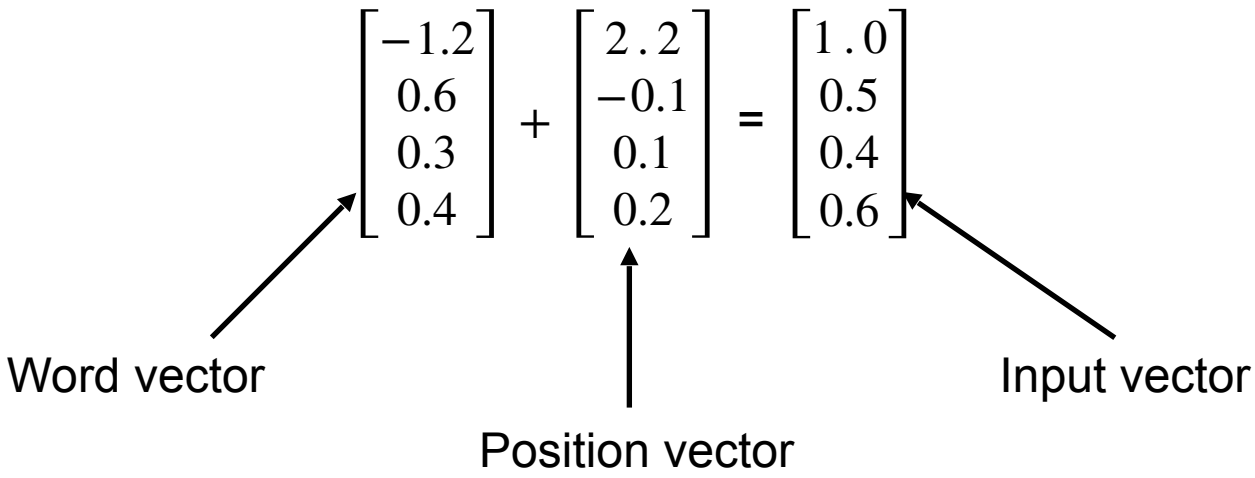
Multi-head attention visualisation



Note: red, green, blue indicate different attention heads.

Positional encodings

- Unlike RNN, the self attention mechanism does not take into account the order of words.
- We could encode positional information directly into the input vectors.
- The t-th input element is created by adding the embedding of the t-th word and a vector that represents the t-th position.



Positional encodings

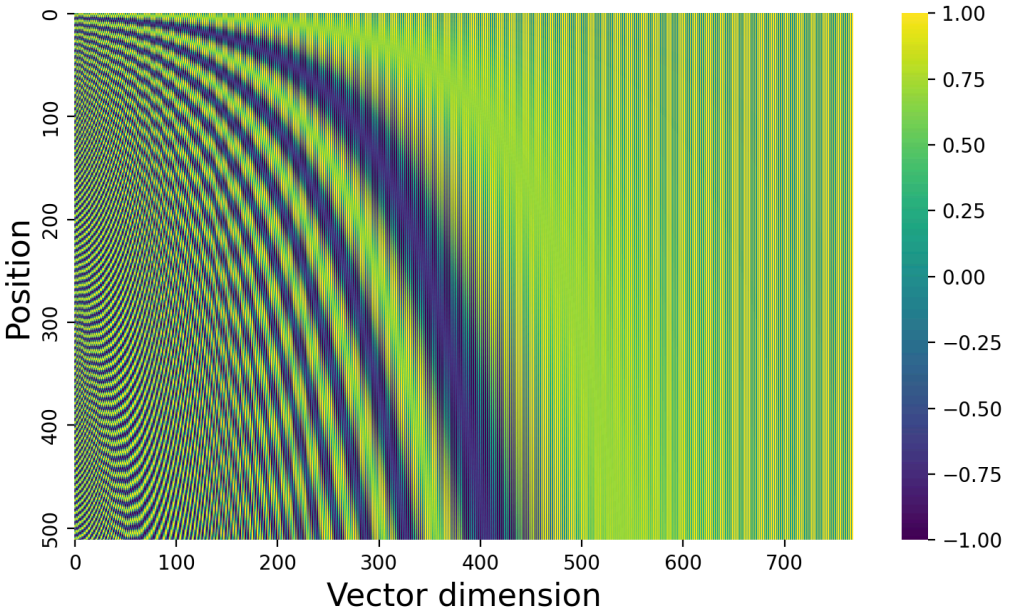
- Positional encodings/embeddings may be learned (similar to word embeddings)
- Or using fixed position encoding vectors, for example, vectors constructed from sine and cosine functions

$$\text{PE}_{t,2i} = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right) \qquad \text{PE}_{t,2i+1} = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right)$$

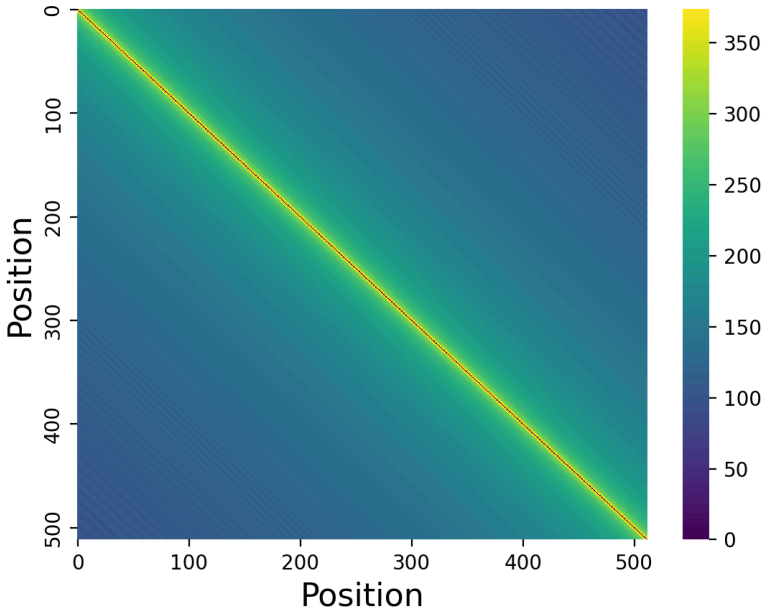
where

  - d is the dimension of the positional encoding/embedding
  - t indexes the position in the input sequence
  - i indexes the dimension, $i \in \{0, 1, \ldots, d-1\}$
  - Different positions t will have different position vectors PE_t
  - You can think of the vector PE_t as a 'clock'
    (Also see this blog post for a good explanation of positional encoding: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/)

Positional encodings



Positional encoding vectors



Dot product of positional encodings

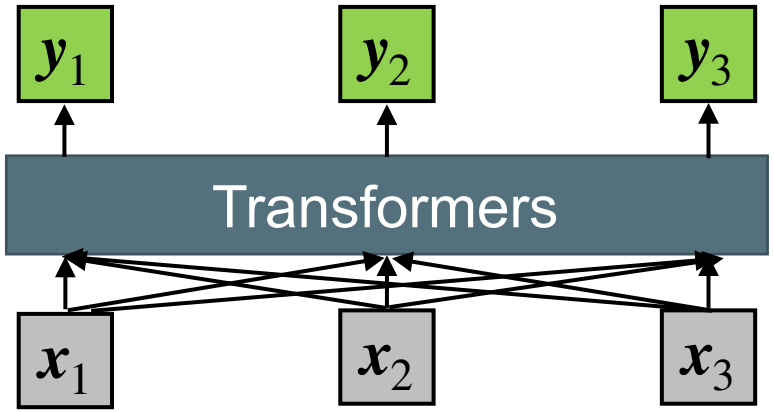
Positional encodings that are closest to each other have the largest dot product.

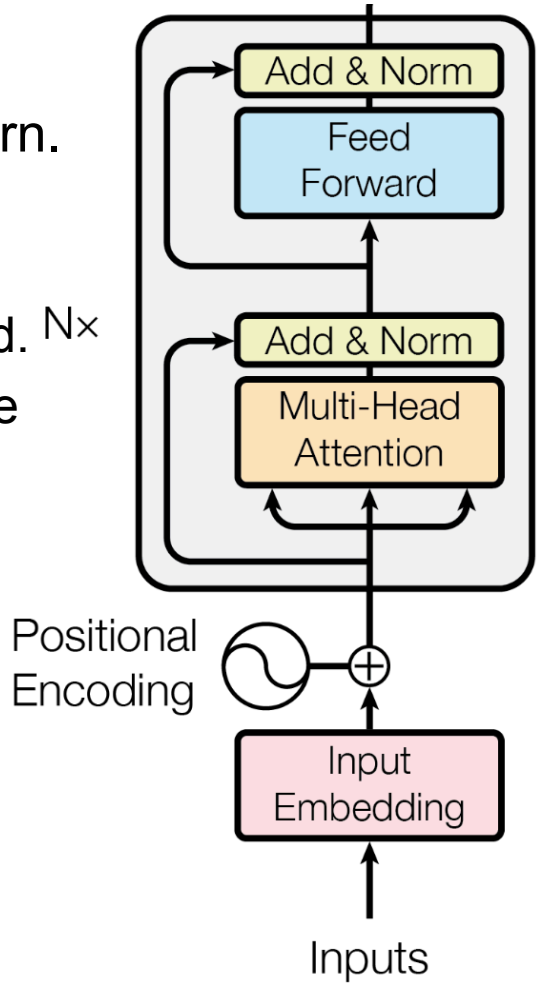
Note: the dot product of a positional encoding with itself is set to 0.

Transformers

Transformers are created by stacking self-attention layers and feed-forward layers in an alternating pattern.

- Self-attention extracts features from all the inputs.
- Feed-forward layers perform a non-linear transform independently on each input vector. ReLU is often used.
- The input is a sequence the output is a sequence of the same length (just like with the RNN models)





Transformers

Usually, transformers also include residual connections and layer normalisations

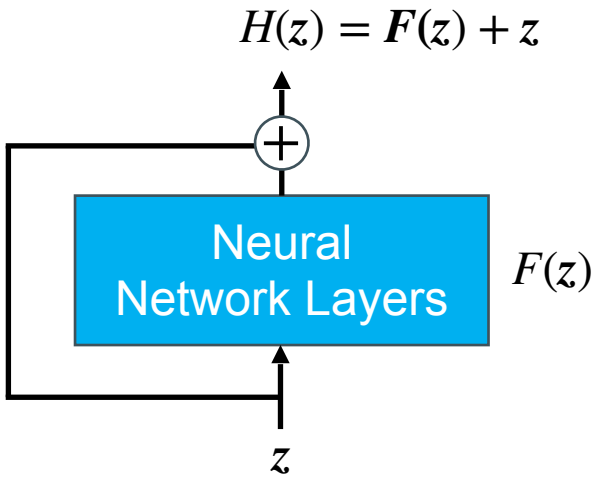
- Residual connections skip a layer and then add the input to the output of that layer.
- Layer normalisation means that each layer output is normalised (subtract mean, divide by standard deviation).

These tricks help with training very deep networks.

Residual layer connection

The residual layer copies the input and adds it to the output of one or more stacked layers.

This means the network is learning a residual: $F(z) = H(z) - z$



Layer normalisation

- During training, the weights change.
- Changes in the weights cause layer outputs to change.
- If the output changes too much, then the next layer may not have been trained on the new input space.
- This is called the “covariate shift” problem.
- Layer normalisation could reduce “covariate shift”.

$$\mu = \frac{1}{d}\sum_{i=1}^{d} z_i$$

$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(z_i - \mu)^2}$$

$$\text{LayerNorm}(z) = \gamma \odot \left(\frac{z-\mu}{\sigma}\right) + \beta$$

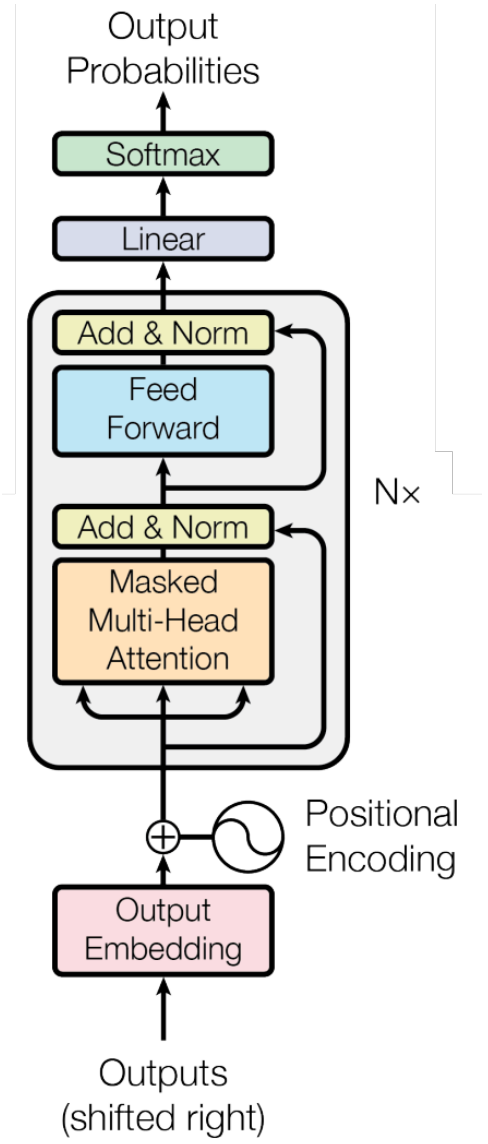
d - dimension of the input

z - vector input to layer normalisation

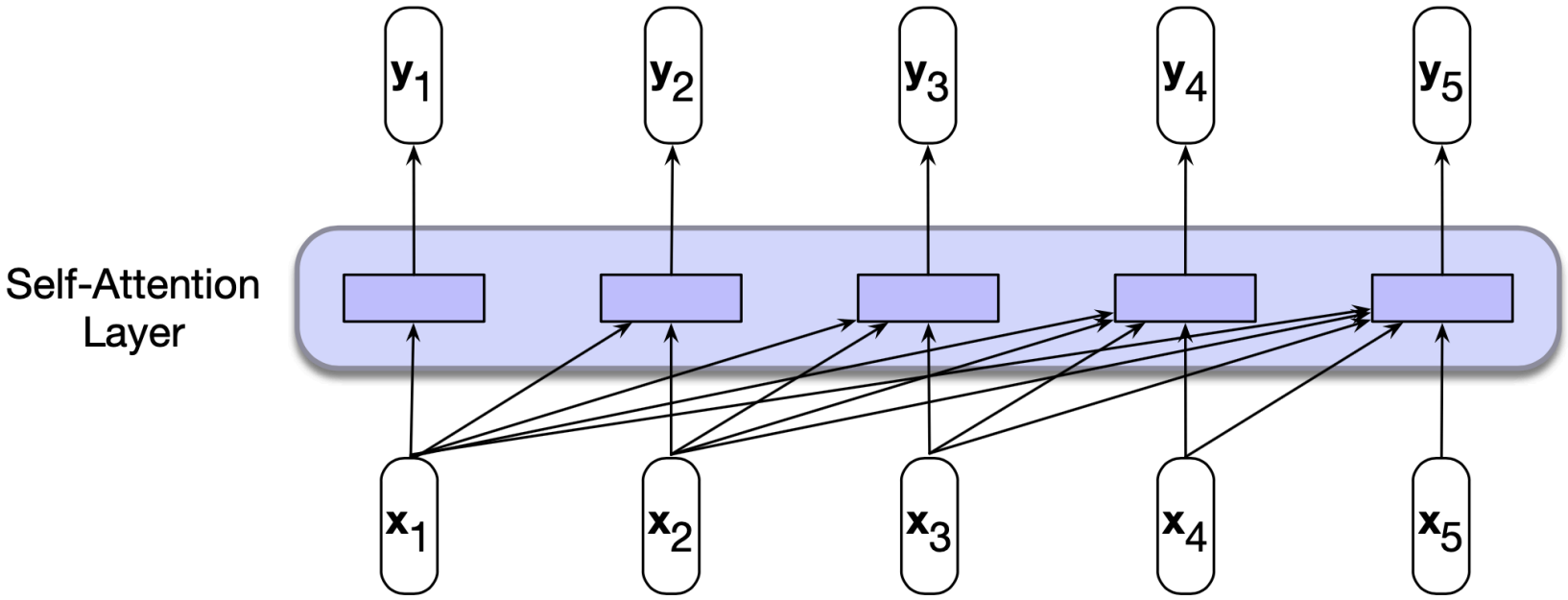
$\beta, \gamma \in \mathbb{R}^d$ - the offset and gain (learnable parameters)

Transformer decoder

- Embedding and positional encoding
- Masked multi-head self-attention
- Fully connect feed-forward network
- Residual connection
- Layer normalisation
- Linear and softmax layer
- Many transformer language models (e.g. GPT models) use the Transformer decoder architecture



Masked self-attention



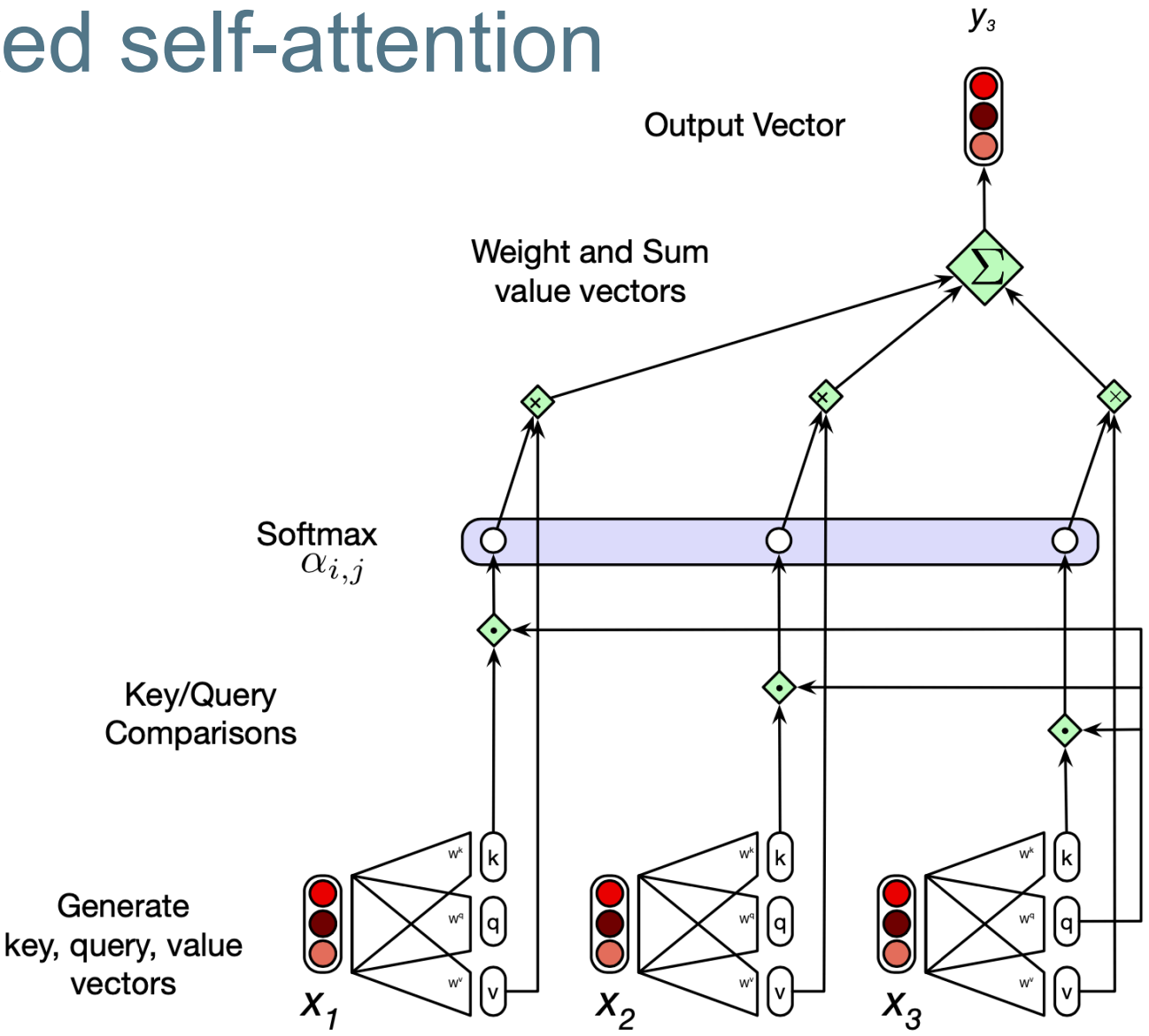
Masked self-attention

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

Masking out subsequent terms to prevent a position from attending to subsequent positions.

(softmax turns −∞ to 0)

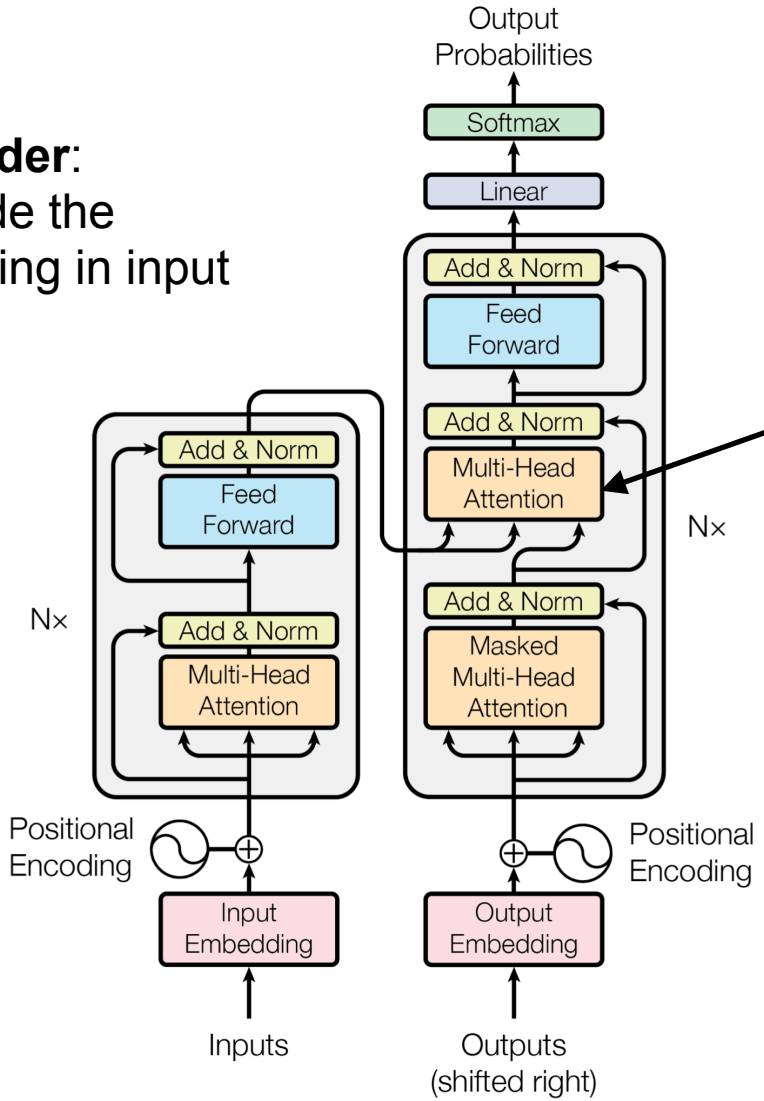
Masked self-attention



Transformer encoder-decoder

Encoder: encode the meaning in input

Decoder: decode the meaning to output



Cross-attention: queries are from the decoder, keys and values are from the output of encoder (similar to the standard attention in seq2seq models).

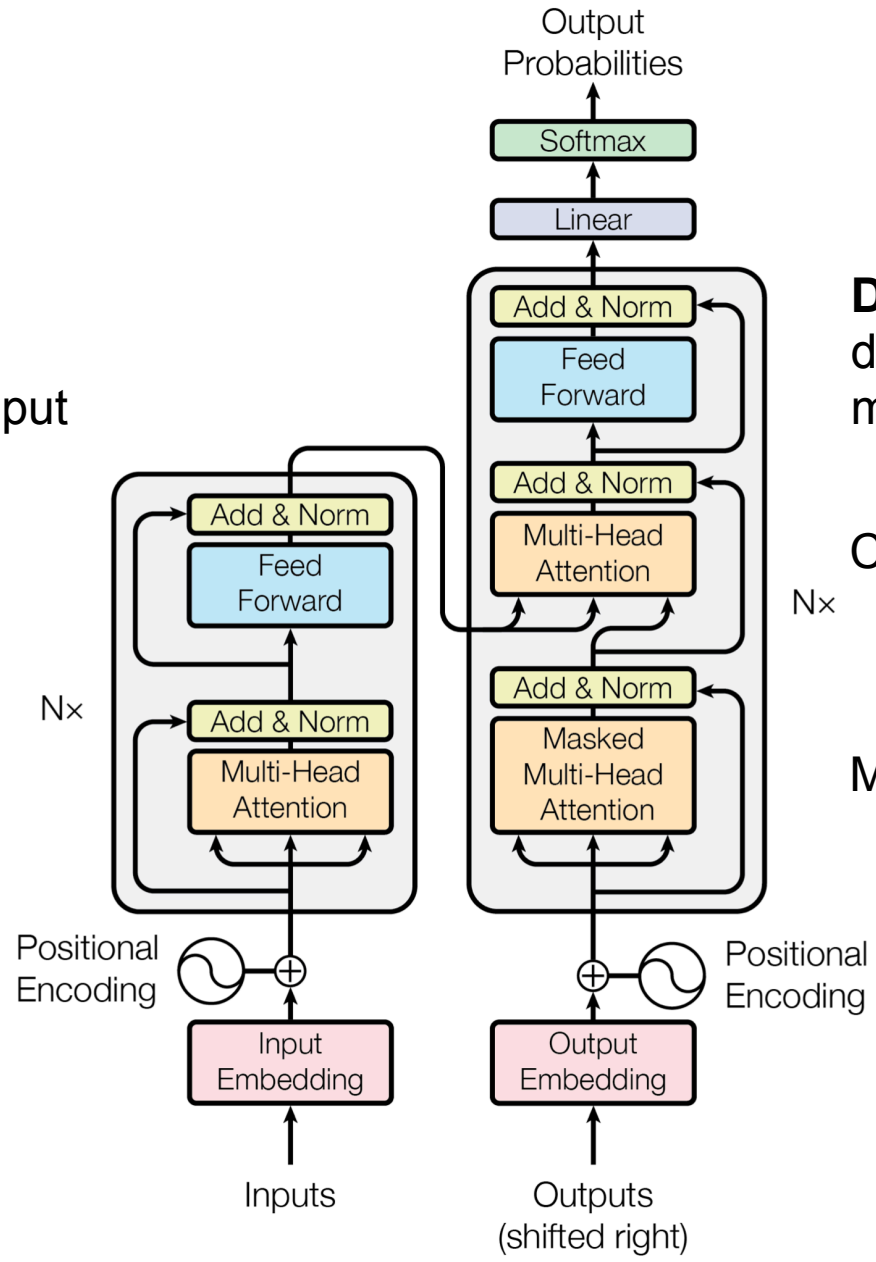
Limitations of transformers

- Expensive in terms of both computation and memory.
- Fixed context window
- May require more training data compared to recurrent models
  - The transformer must learn sequential structure (with the help of positional encoding/embeddings), whereas recurrent models are natively sequential.

Summary

Encoder:
encode the
meaning in input

Self-attention



Decoder:
decode the
meaning to output

Cross-attention

Masked self-attention

References

Sections 10.1, 10.2, Speech and Language Processing.