Transformers

Dot Product

- Let's say we have two 3-dimensional vectors, x_1 and x_2:

$\boldsymbol{x_1}$ = [1, 2, 3] $\boldsymbol{x_2}$ = [4, 5, 6]

$\boldsymbol{x_1}.\boldsymbol{x_2}$= (1 x 4) + (2 x 5) + (3 x 6) = 4 + 10 + 18 =32

- **Activity 1:** Let's say we have two 2-dimensional vectors, $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, represented as follows: $\boldsymbol{x_1}$ = [3, 5] $\boldsymbol{x_2}$ = [-2, 4], dot product of A and $\boldsymbol{x_2}$?

- **Activity2:** Let's say we have two 4-dimensional vectors, $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$, represented as follows: $\boldsymbol{x_1}$ = [1, 2, 3, 4] $\boldsymbol{x_2}$ = [-4, 0, 2, 5], dot product of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$?

Self Attention

- Attention is comparison of input to other input elements.
  e.g value of $\boldsymbol{y_3}$ depends on the score of (x_3, x_1), (x_3, x_2) and (x_3, x_3)

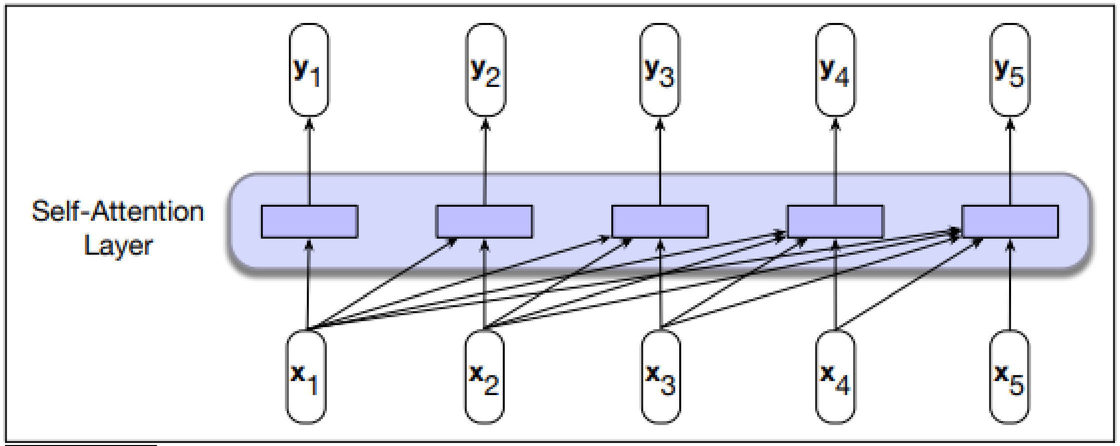


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

$$score(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i}.\boldsymbol{x_j}$$

Self Attention

$$\alpha_{i,j} = softmax\left(score(\boldsymbol{x}_i, \boldsymbol{x}_j)\right) \quad \forall j \leq i$$

$$\alpha_{i,j} = \frac{\exp\left(score(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)}{\sum_{k=1}^{i} \exp\left(score(\boldsymbol{x}_i, \boldsymbol{x}_k)\right)} \quad \forall j \leq i$$

$$\boldsymbol{y}_i = \sum_{j \leq i} \alpha_{ij} \boldsymbol{x}_j$$

Activity

Query, Key, Value

Considering the output y_3

$\boldsymbol{x}_3$ *query*

$\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ *key*

$\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ *value*

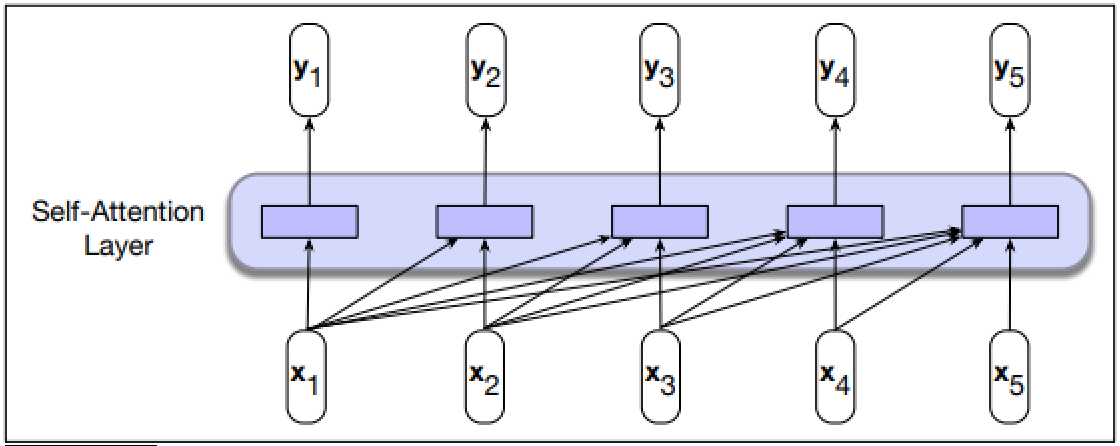


Figure 10.1 Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Attention Values

$score(\boldsymbol{x}_3, \boldsymbol{x}_1) = \alpha_{31}$ $score(\boldsymbol{x}_3, \boldsymbol{x}_2) = \alpha_{32}$ $score(\boldsymbol{x}_3, \boldsymbol{x}_3) = \alpha_{33}$

$$\boldsymbol{\alpha}_{31}\boldsymbol{x}_1 + \boldsymbol{\alpha}_{32}\boldsymbol{x}_2 + \boldsymbol{\alpha}_{33}\boldsymbol{x}_3 = \boldsymbol{y}_3$$

Key, Query and Value Matrices

- To capture the roles, transformer has 3 matrices

$$\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{W}^{\boldsymbol{K}}, \boldsymbol{W}^{\boldsymbol{V}}$$

- Transformer project each $\boldsymbol{x}_i$ into **query, key** and **value** vectors

$$\boldsymbol{q}_i = \boldsymbol{W}^{\boldsymbol{Q}} \boldsymbol{x}_i \ ; \boldsymbol{k}_i = \boldsymbol{W}^{\boldsymbol{K}} \boldsymbol{x}_i \ : \boldsymbol{v}_i = \boldsymbol{W}^{\boldsymbol{V}} \boldsymbol{x}_i$$

$$\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{W}^{\boldsymbol{K}}, \boldsymbol{W}^{\boldsymbol{V}} \in \boldsymbol{R}^{\boldsymbol{d} \times \boldsymbol{d}}, \qquad \boldsymbol{x}_i, \boldsymbol{y}_i \in \boldsymbol{R}^{\boldsymbol{1} \times \boldsymbol{d}}$$

Self-attention (improved version)

$$score(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{q}_i . \boldsymbol{k}_j$$

$$score(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\boldsymbol{q}_i . \boldsymbol{k}_j}{\sqrt{d}}$$

$$\alpha_{i,j} = \frac{\exp\left(score(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)}{\sum_{k=1}^{i} \exp\left(score(\boldsymbol{x}_i, \boldsymbol{x}_k)\right)} \quad \forall j \leq i$$

$$\boldsymbol{y}_i = \sum_{j \leq i} \alpha_{ij} \boldsymbol{v}_j$$

Visual look of calculating $\boldsymbol{y}_3$

- Single output at single time step i
- Each output y_i is calculated independently

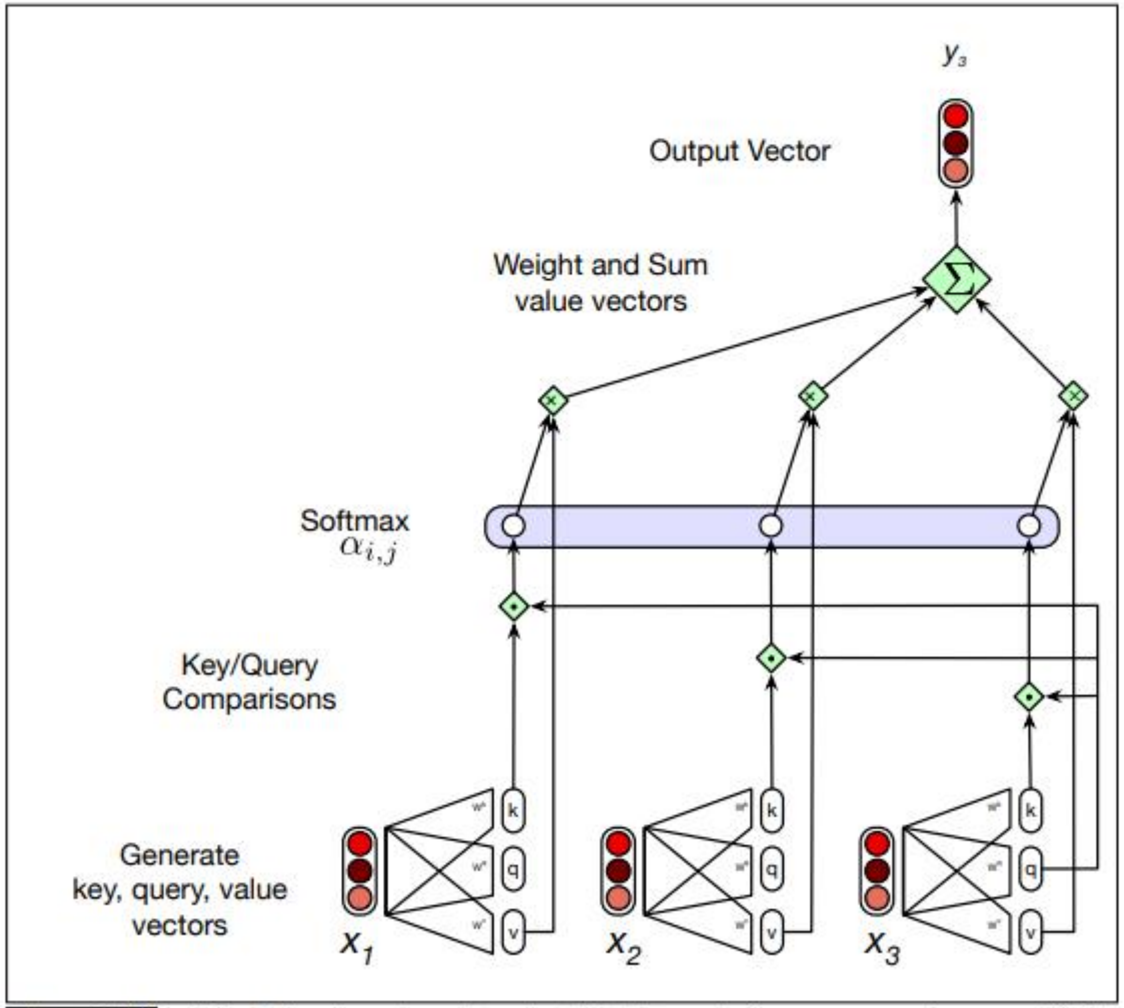


Figure 10.2 Calculating the value of $\mathbf{y}_3$, the third element of a sequence using causal (left-to-right) self-attention.

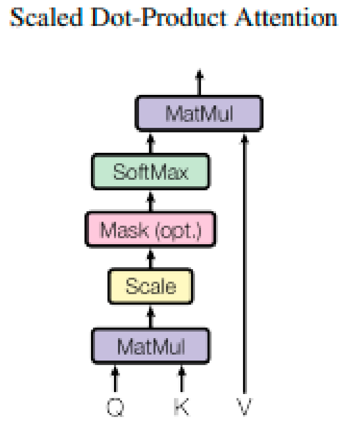
Vectorization

- Input $\boldsymbol{X}$: A sequence of N tokens with embedding dimension d

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{\boldsymbol{Q}} \; ; \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{\boldsymbol{K}} \; : \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{\boldsymbol{V}}$$

$$\boldsymbol{W}^{\boldsymbol{Q}}, \boldsymbol{W}^{\boldsymbol{K}}, \boldsymbol{W}^{\boldsymbol{V}} \in \boldsymbol{R}^{\boldsymbol{d}\times\boldsymbol{d}}, \qquad \boldsymbol{X} \in \boldsymbol{R}^{\boldsymbol{n}\times\boldsymbol{d}}$$

$$\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \boldsymbol{R}^{\boldsymbol{n}\times\boldsymbol{d}},$$



$$\boldsymbol{SelfAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\boldsymbol{T}}}{\sqrt{\boldsymbol{d}}}\right)V$$

Attention Values

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

Figure 10.3 The $N \times N$ $\mathbf{QK}^\intercal$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

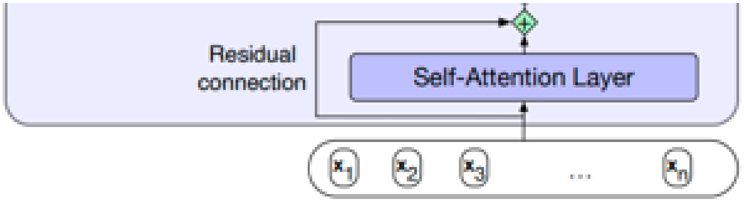
Attention is quadratic
In the length of input

Residual Connection

- In deep networks, residual connections are connections that pass information from a lower layer to a higher layer without going through the intermediate layer.
- Residual connections in transformers are implemented by adding a layer's input vector to its output vector before passing it forward.

$$\boldsymbol{x} + \boldsymbol{SelfAttention}(\boldsymbol{x}),$$

Residual Connection



- Suppose we have an input matrix X representing word embeddings of a sentence:

X = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]

- Assume we have already calculated the SelfAttention(Q, K, V) matrix using the self-attention mechanism:

SelfAttention(Q, K, V) =
[[0.5, 1.0, 1.5],
[2.0, 2.5, 3.0],
[3.5, 4.0, 4.5]]

- Residual_Connection =

X + SelfAttention(Q, K, V) =

[[1.5, 3.0, 4.5],
[6.0, 7.5, 9.0],
[10.5, 12.0, 13.5]]

Layer Normalization

- Given a batch of input vectors X with dimensions $(batch\_size, feature\_\mathrm{dim})$, a scale parameter vector γ and a bias parameter vector β, the layer normalization of X is computed as:

$$\mu = \frac{1}{batch\_size} \sum_{i=1}^{batch\_size} X_i$$

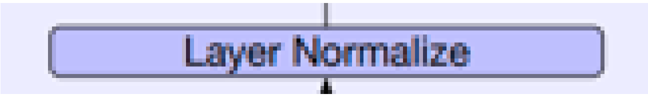
$$\sigma^2 = \frac{1}{batch\_size} \sum_{i=1}^{batch\_size} X_i - \mu$$

$$\hat{X} = \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$LayerNorm = \alpha\hat{X} + \beta$$

Where α and β are learnable parameters and ϵ is small constant

Layer normalization



- Step 1: Calculate the mean and standard deviation of each feature.
- Step 2: Normalize each feature using the mean and standard deviation
-

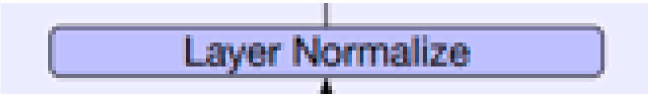
```
Residual_Connection = [
[1.5, 3.0, 4.5],
[6.0, 7.5, 9.0],
[10.5, 12.0, 13.5]
]
```

```
Mean = [
(1.5+6.0+10.5)/3, (3.0+7.5+12.0)/3, (4.5+9.0+13.5)/3
] = [6.0, 7.5, 9.0]
```

```
Standard Deviation = [
sqrt(((1.5-6.0)^2+(6.0-6.0)^2+(10.5-6.0)^2)/3),
sqrt(((3.0-7.5)^2+(7.5-7.5)^2+(12.0-7.5)^2)/3),
sqrt(((4.5-9.0)^2+(9.0-9.0)^2+(13.5-9.0)^2)/3)
] ≈ [3.674, 3.674, 3.674]
```

```
Normalized = [
[(1.5-6.0)/3.674, (3.0-7.5)/3.674, (4.5-9.0)/3.674],
[(6.0-6.0)/3.674, (7.5-7.5)/3.674, (9.0-9.0)/3.674],
[(10.5-6.0)/3.674, (12.0-7.5)/3.674, (13.5-9.0)/3.674]
] ≈ [
[-1.22, -1.22, -1.22],
[0.00, 0.00, 0.00],
[1.22, 1.22, 1.22]
]
```

Layer normalization



- Step 3: Apply learnable scale and shift parameters

Layer normalization also includes learnable parameters (gamma) for scaling and (beta) for shifting. For simplicity, assume gamma = 1 and beta = 0.

Layer_Normalized = gamma * Normalized + beta ≈

[[-1.22, -1.22, -1.22],

[0.00, 0.00, 0.00],

[1.22, 1.22, 1.22]]

Transformer Block

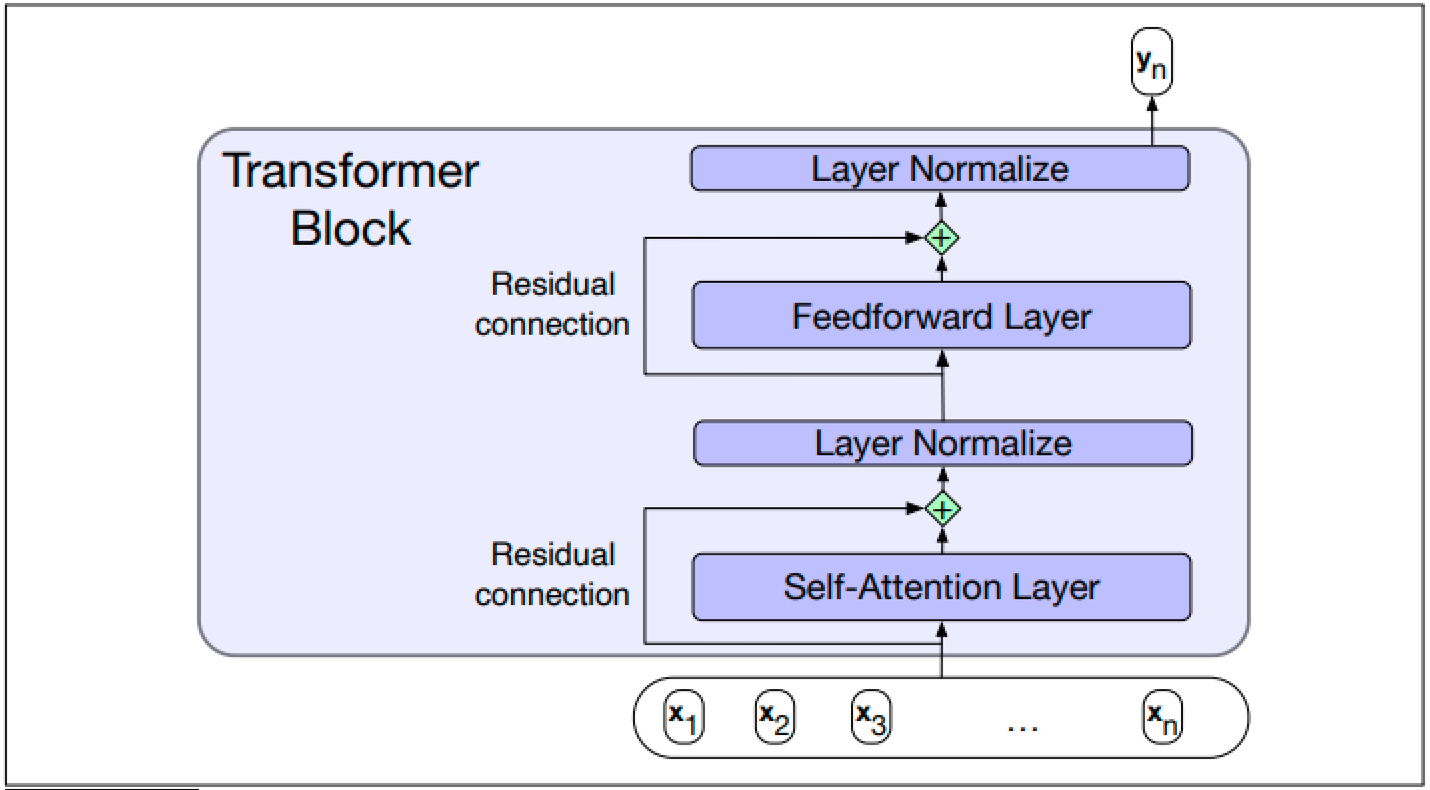


Figure 10.4 A transformer block showing all the layers.

MultiHead Attention

$$\text{MultiheadAttention}(\boldsymbol{X}) = (\boldsymbol{head_1} \oplus \boldsymbol{head_2} \oplus \cdots \boldsymbol{head_h})\boldsymbol{W^O}$$

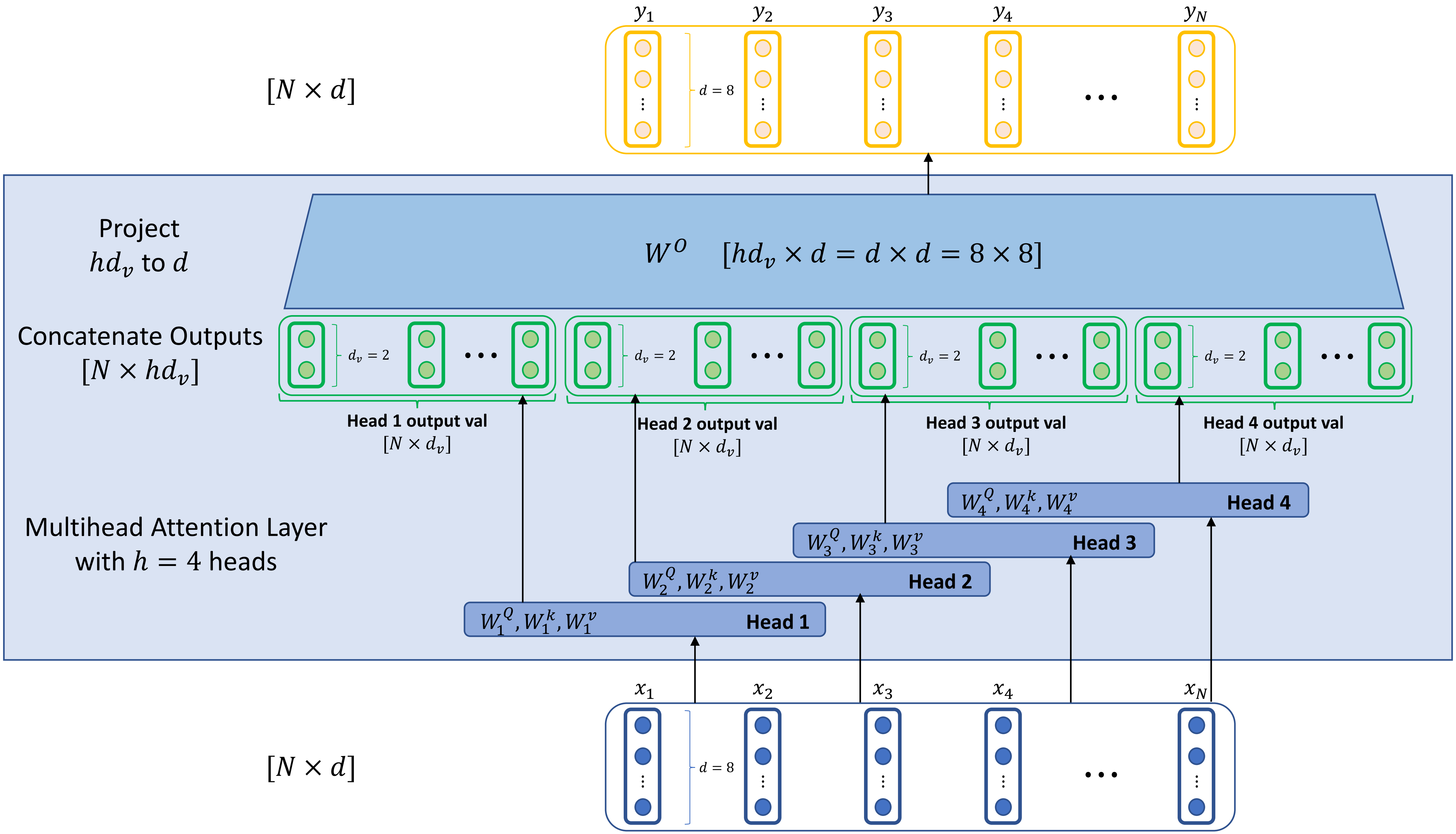
$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W_i^Q} : \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W_i^K} : \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W_i^V}$$

$$\boldsymbol{head_i} = SelfAttention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$$

$$\boldsymbol{W^Q} \in \boldsymbol{R^{d \times d_k}}, \boldsymbol{W^K} \in \boldsymbol{R^{d \times d_k}}, \boldsymbol{W^V} \in \boldsymbol{R^{d \times d_v}}, \qquad \boldsymbol{X} \in \boldsymbol{R^{n \times d}}$$

$$\boldsymbol{Q} \in \boldsymbol{R^{N \times d_k}}, \boldsymbol{K} \in \boldsymbol{R^{N \times d_k}}, \boldsymbol{V} \in \boldsymbol{R^{N \times d_v}},$$

$$\boldsymbol{W^O} \in \boldsymbol{R^{hd_v \times h}}$$



y_1
y_2
y_3
y_4
y_N
$[N \times d]$
$d = 8$
Project
hd_v to d
$W^O \quad [hd_v \times d = d \times d = 8 \times 8]$
Concatenate Outputs
$[N \times hd_v]$
$d_v = 2$
Head 1 output val
$[N \times d_v]$
Head 2 output val
$[N \times d_v]$
Head 3 output val
$[N \times d_v]$
Head 4 output val
$[N \times d_v]$
W_4^Q, W_4^k, W_4^v
Head 4
W_3^Q, W_3^k, W_3^v
Head 3
W_2^Q, W_2^k, W_2^v
Head 2
W_1^Q, W_1^k, W_1^v
Head 1
Multihead Attention Layer
with $h = 4$ heads
x_1
x_2
x_3
x_4
x_N
$[N \times d]$
$d = 8$

Multihead Attention Transformer Block

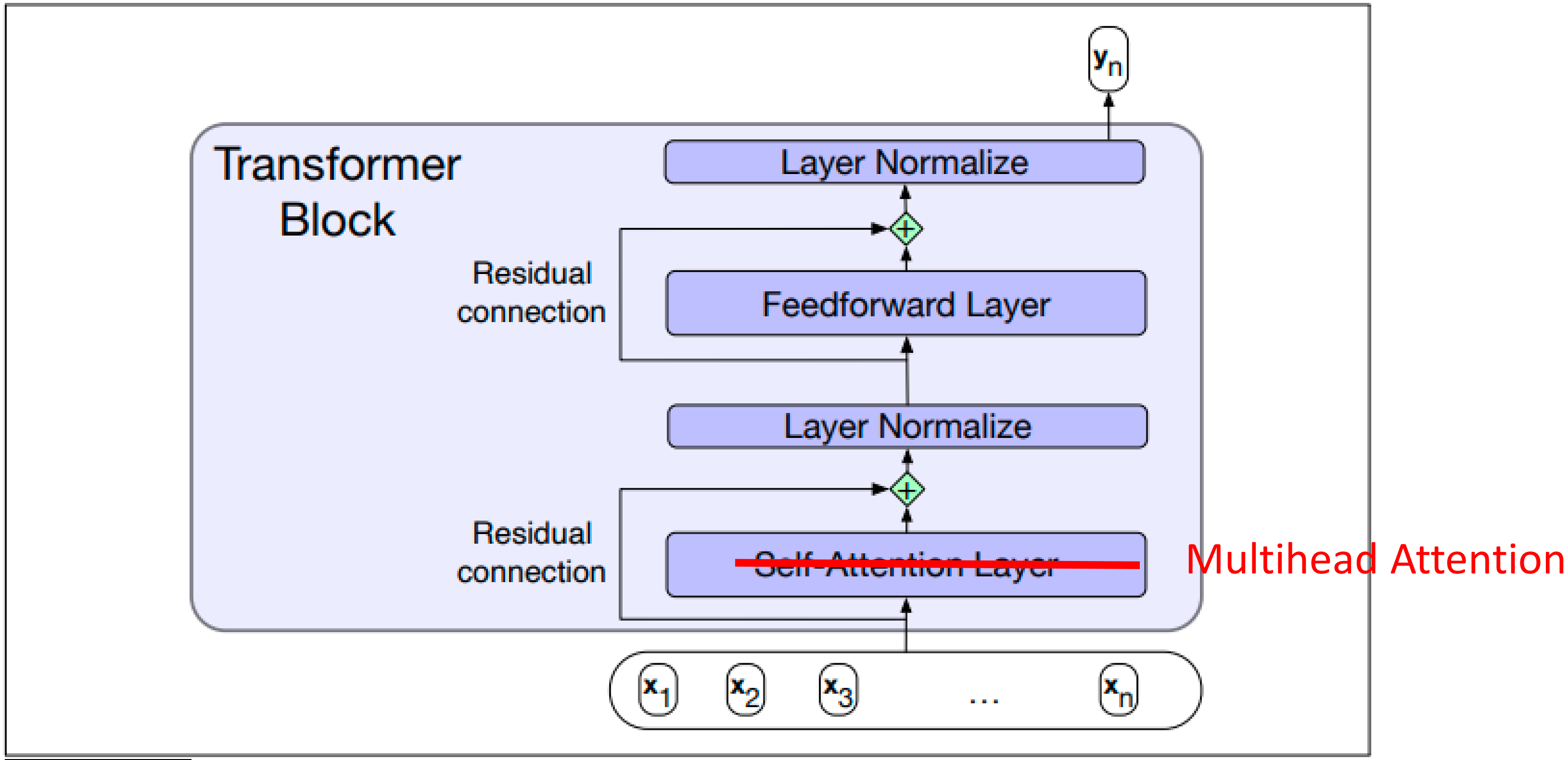


Figure 10.4 A transformer block showing all the layers.

Word Order

- Does transformer model the position of each token in the input sequence ?

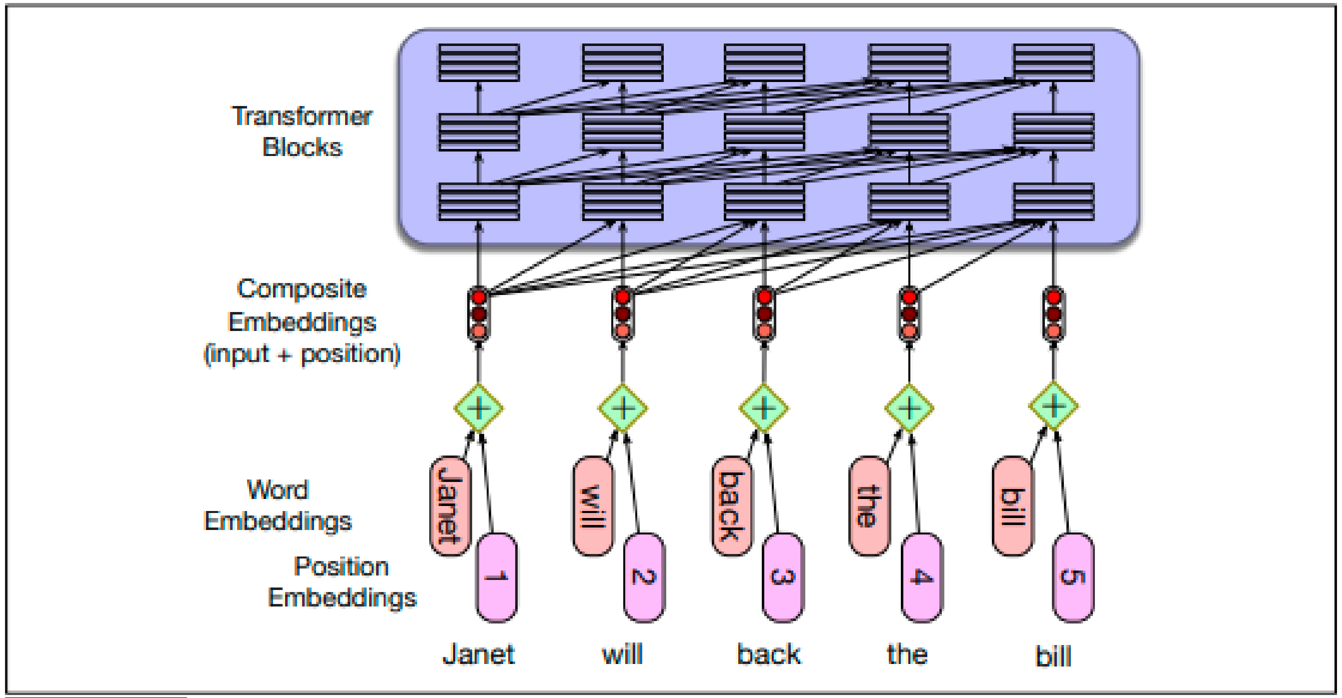


Figure 10.6 A simple way to model position: simply adding an embedding representation of the absolute position to the input word embedding to produce a new embedding of the same dimenionality.

Transformer architecture

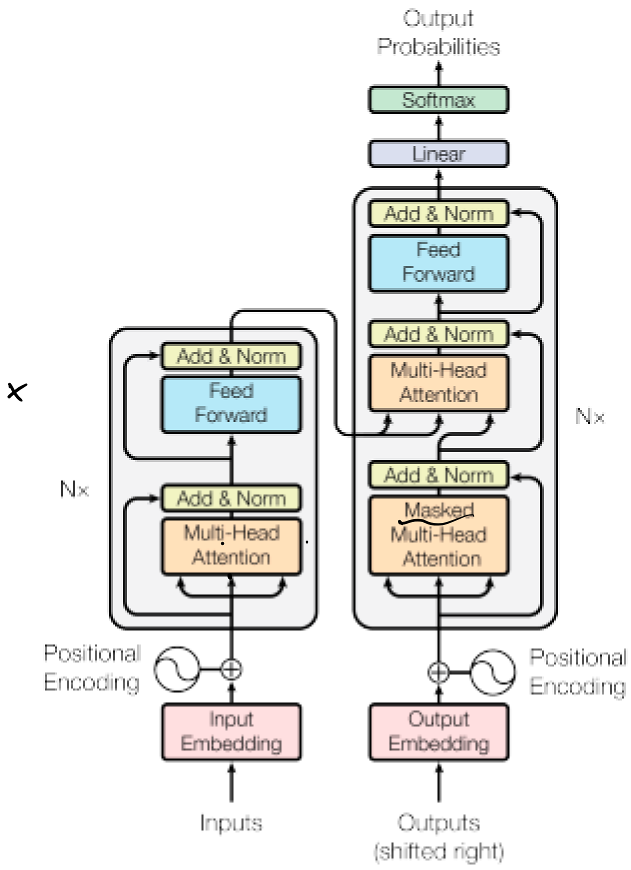


Figure 1: The Transformer - model architecture.

Transformers as Language Model

$$L_{CE}(\hat{y}_t, y_t) = -\log \hat{y}_t[w_{t+1}]$$

Where w_{t+1} is the correct next word in the input sequence

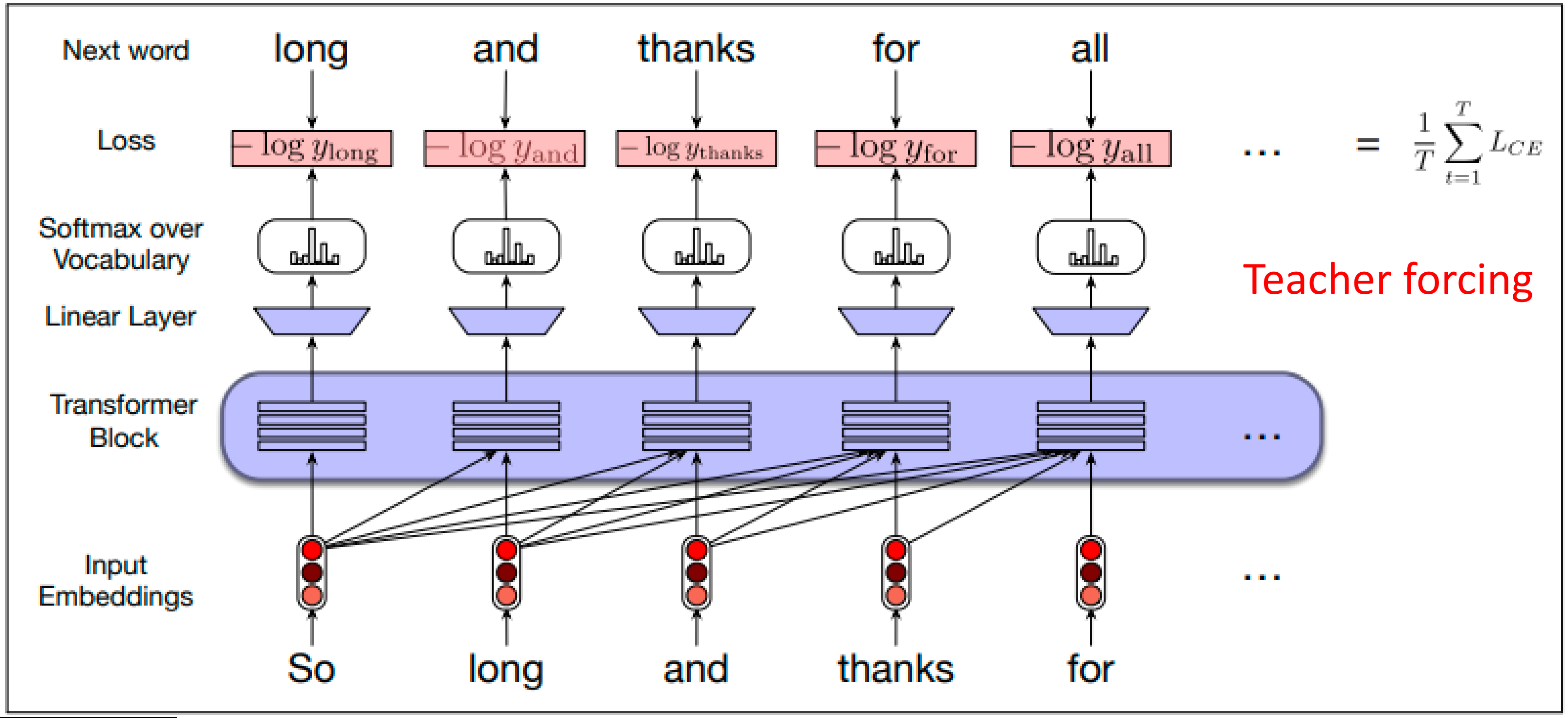
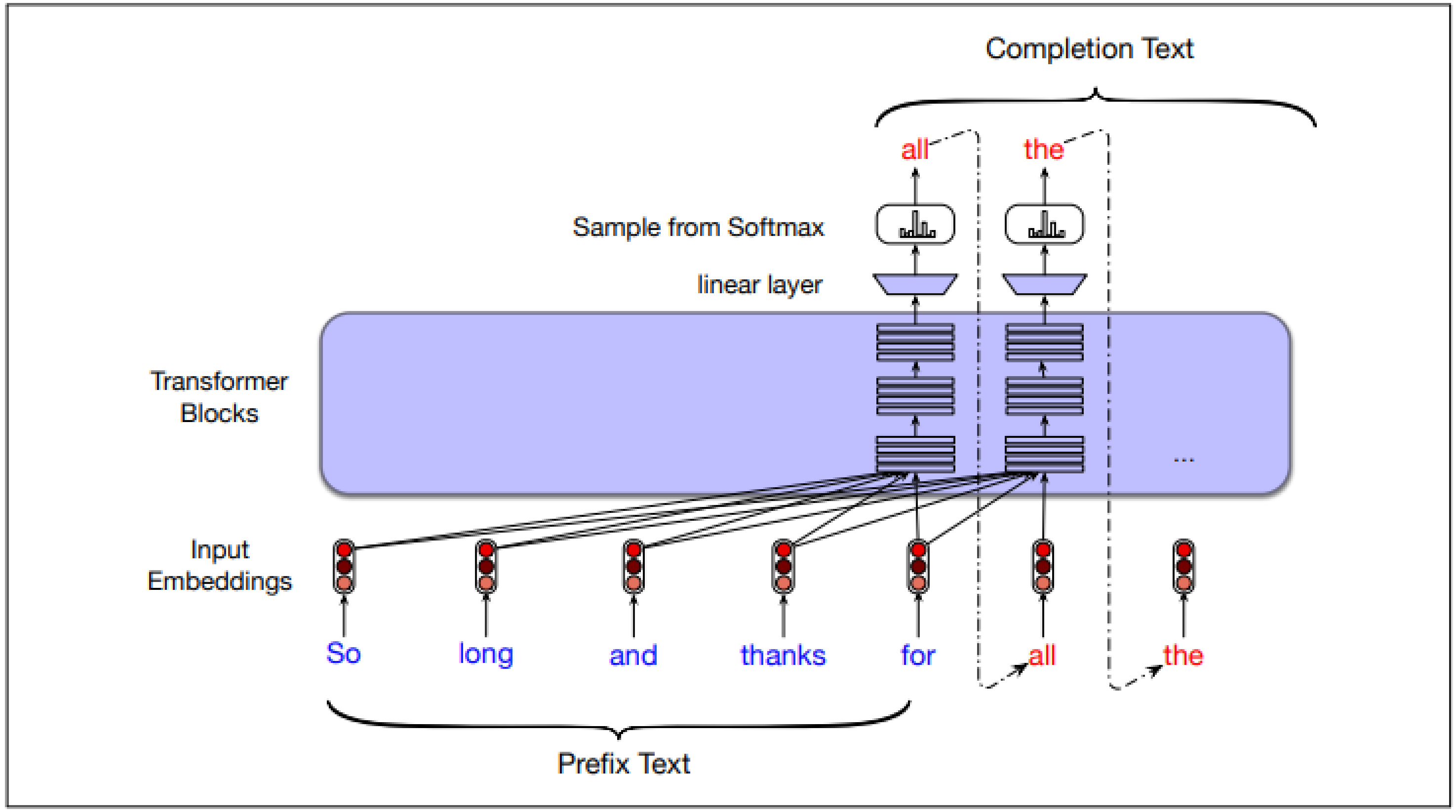


Figure 10.7 Training a transformer as a language model.

Autoregressive text Completion



Generating a text from Language model

- Once language model trained we generate the text in the following way
  - ✓ Sample a word in the output from the softmax distribution that results from using the beginning of sentence marker, <s>, as the first input.
  - ✓ Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.
  - ✓ Continue generating until the end of sentence marker, </s>, is sampled or a fixed length limit is reached.

$$\widehat{y_t} = argmax_{w \in V} P(w | y_1 y_2 \ldots y_{t-1})$$

greedy decoding

Greedy approach problem

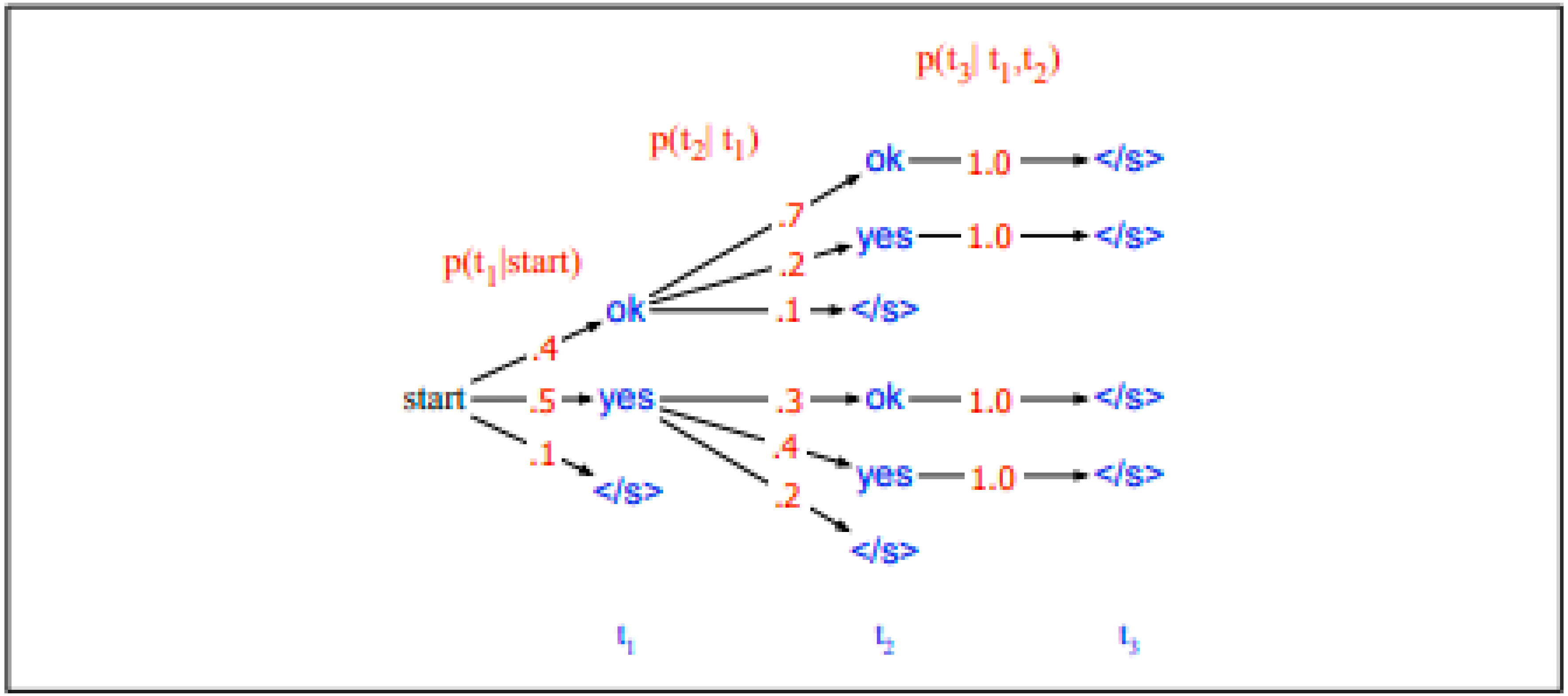


Figure 10.8 A search tree for generating the target string $T = t_1, t_2, \ldots$ from the vocabulary $V = \{\text{yes}, \text{ok}, \text{<s>}\}$, showing the probability of generating each token from that state. Greedy search would choose *yes* at the first time step followed by *yes*, instead of the globally most probable sequence *ok ok*.

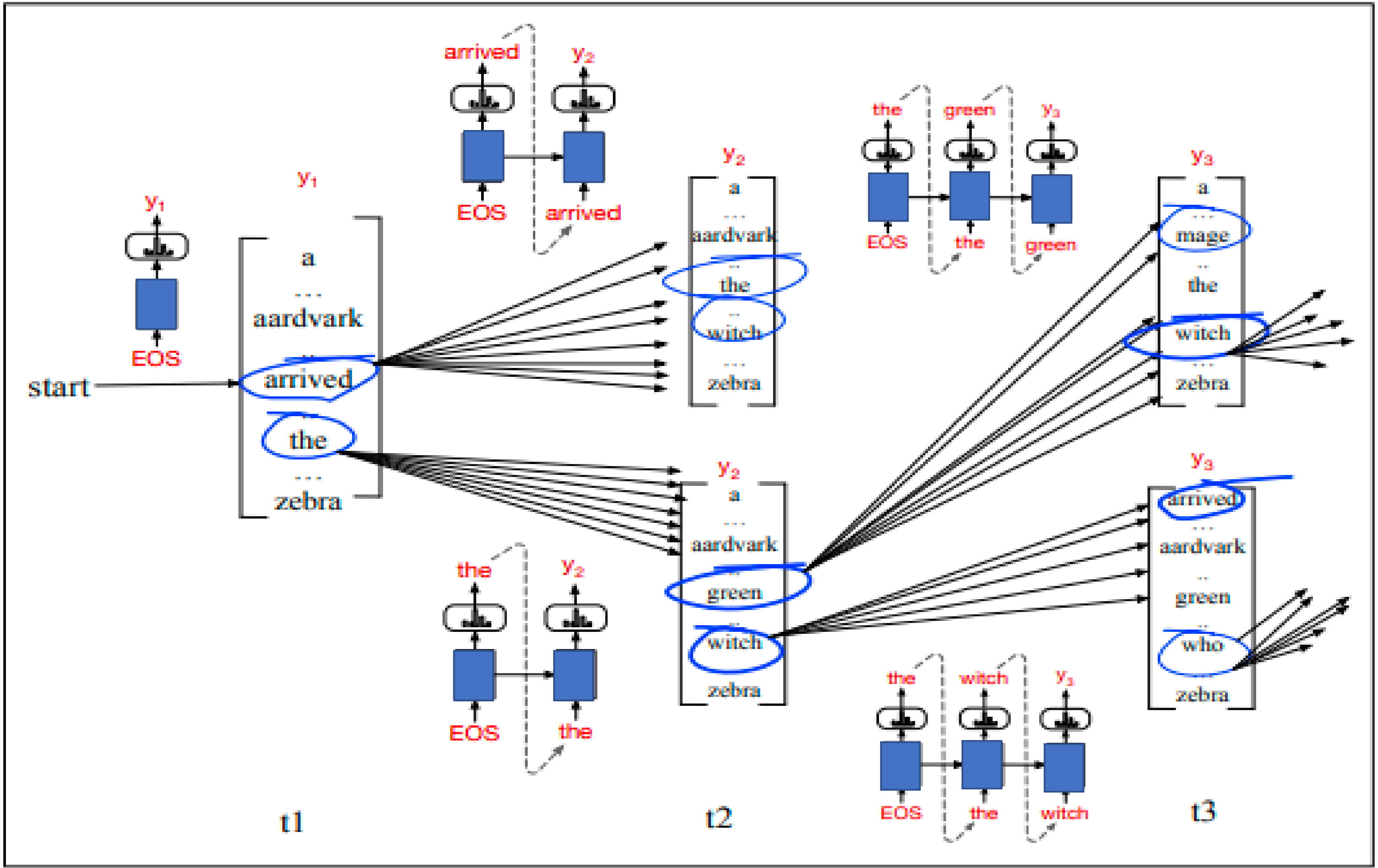


Figure 10.9 Beam search decoding with a beam width of $k = 2$. At each time step, we choose the k best hypotheses, compute the V possible extensions of each hypothesis, score the resulting $k * V$ possible hypotheses

Calculating probabilities

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log\left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)\ldots P(y_t|y_1,\ldots,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,\ldots,y_{i-1},x) \qquad (10.22)
\end{aligned}
$$

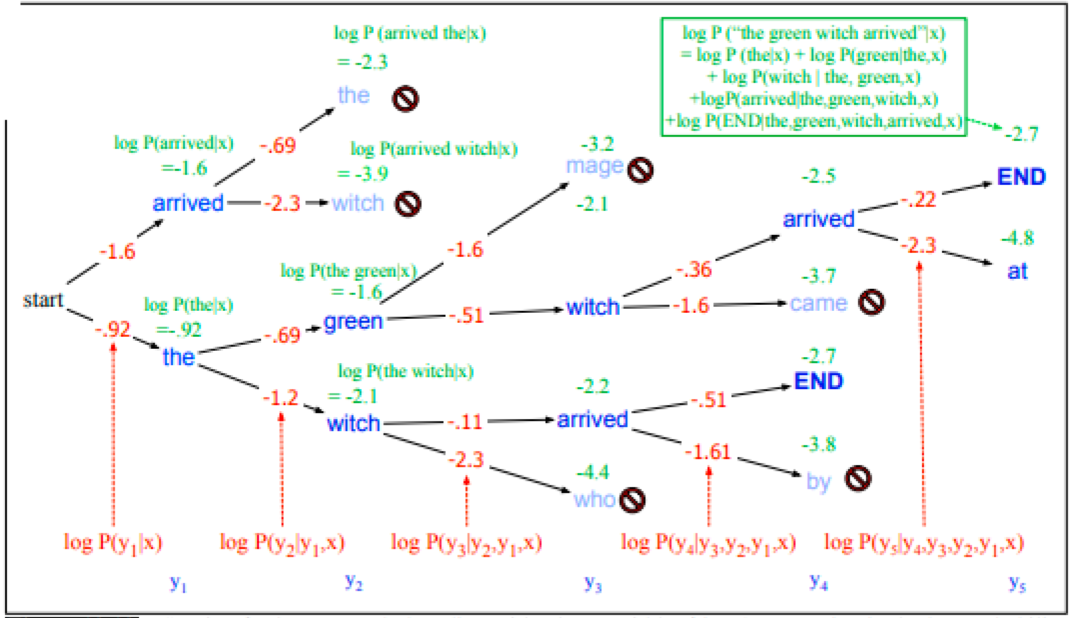


Figure 10.10 Scoring for beam search decoding with a beam width of $k = 2$. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step.