Deep Learning - Foundations and Concepts

Chapter 12. Transformers

nonlineark@github

March 18, 2025

Outline

Attention

The fundamental concept that underpins a transformer is attention:

- This was originally developed as an enhancement to RNNs for machine translation (Bahdanau, Cho and Bengio, 2014)
- Later, it was found that significantly improved performance could be obtained by eliminating the recurrence structure and instead focusing exclusively on the attention mechanism (Vaswani et al., 2017).

Attention

Consider the following two sentences:

- I swam across the river to get to the other bank.
- I walked across the road to get cash from the bank.

Here the word "bank" has different meanings in the two sentences:

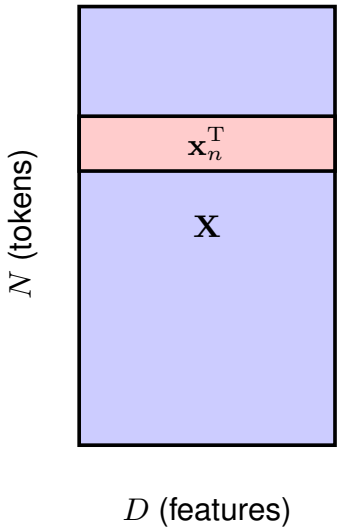
- In the first sentence, the words "swam" and "river" most strongly indicate that "bank" refers to the side of a river.
- In the second sentence, the word "cash" is a strong indicator that "bank" refers to a financial institution.

To determine the appropriate interpretation of "bank", a neural network processing such a sentence should:

- Attend to specific words from the rest of the sequence.
- The particular locations that should receive more attention depend on the input sequence itself.

Transformer processing

- The input data to a transformer is a set of vectors $\{x_n\}$ of dimensionality D, where $n = 1, \ldots, N$.
- We refer to these data vectors as tokens, and the elements of the tokens are called features.
- We will combine the tokens into a matrix X of dimension $N \times D$ in which the nth row comprises the token x_n^T.



Transformer processing

The fundamental building block of a transformer is a function that takes a data matrix X as input and creates a transformed matrix $\tilde{X}$ of the same dimensionality as the output:

$$\tilde{X} = \mathrm{TransformerLayer}(X)$$

A single transformer layer itself comprises two stages:

- The first stage, which implements the attention mechanism, mixes together the corresponding features from different tokens across the columns of the data matrix.
- The second stage acts on each row independently and transforms the features within each token.

Attention coefficients

Suppose we have a set of input tokens $x_1, \ldots, x_N$ and we want to map this set to another set $y_1, \ldots, y_N$:

- y_n should depend on all the tokens $x_1, \ldots, x_N$.
- This dependence should be stronger for those tokens x_m that are particularly important for determining the modified representation of y_n.

A simple way to achieve this is to define each output token y_n to be a linear combination of the input tokens:

$$y_n = \sum_{m=1}^{N} a_{nm} x_m$$

$$a_{nm} \geq 0 \qquad \sum_{m=1}^{N} a_{nm} = 1$$

Self-attention

The problem of determining the attention coefficients can be viewed from an information retrieval perspective:

- We could view the vector x_n as:
  - The key for input token n.
  - The value for input token n.
  - The query for output token n.
- To measure the similarity between the query x_n and the key x_m, we could use their dot product: $x_n^T x_m$.
- To make sure the attention coefficients define a partition of unity, we could use the $\mathrm{softmax}$ function to transform the dot products.

Self-attention

Dot-product self-attention:

$$y_n = \sum_{m=1}^{N} a_{nm} x_m$$

$$a_{nm} = \frac{\exp(x_n^T x_m)}{\sum_{m'=1}^{N} \exp(x_n^T x_{m'})}$$

Or write in matrix notation:

$$Y = \mathrm{softmax}(XX^T)X$$

where $\mathrm{softmax}(L)$ means to apply $\mathrm{softmax}$ to each row of the matrix L.

Network parameters

The current transformation from input tokens $\{x_n\}$ to output tokens $\{y_n\}$ has major limitations:

- The transformation is fixed and has no capacity to learn from data because it has no adjustable parameters.
- Each of the feature values within a token x_n plays an equal role in determining the attention coefficients.

Network parameters

We can overcome these limitations by defining separate query, key and value matrices each having their own independent linear transformations:

- $Q = XW^{(q)}$, where $W^{(q)}$ has dimensionality $N \times D_k$.
- $K = XW^{(k)}$, where $W^{(k)}$ has dimensionality $N \times D_k$.
- $V = XW^{(v)}$, where $W^{(v)}$ has dimensionality $N \times D_v$.
- A typical choice is $D_k = D$.
- If we set $D_v = D$:
  - This will facilitate the inclusion of residual connections.
  - Multiple transformer layers can be stacked on top of each other.

The dot-product self-attention now takes the form:

$$Y = \text{softmax}(QK^T)V$$

Scaled self-attention

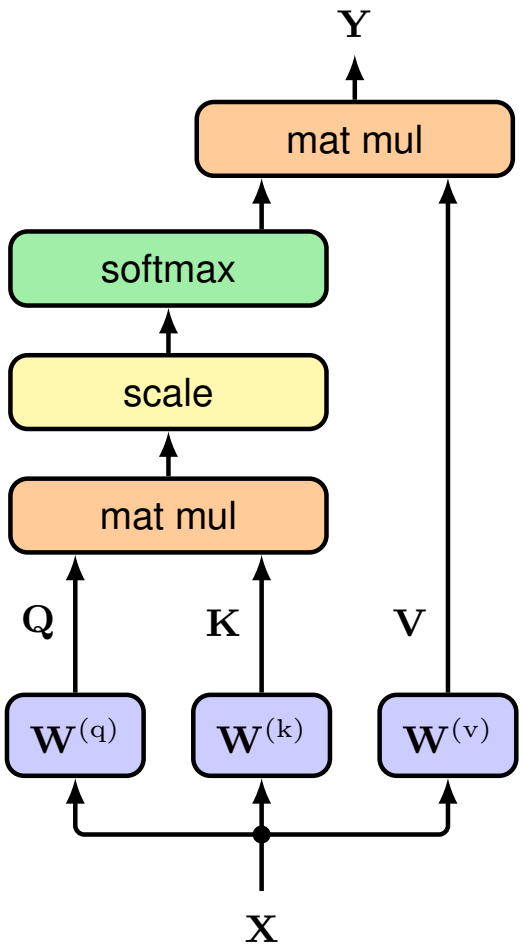
- When logits are too large, the $\mathrm{softmax}$ function will produce extremely small gradients, which is not desirable.
- We need to scale the logits before applying the $\mathrm{softmax}$ function.
- Notice that if $q, k \in \mathbb{R}^{D_k}$ and the elements of q and k are all independent random numbers with zero mean and unit variance, then $\mathrm{var}(q^T k) = D_k$. Thus it would be appropriate to scale the logits by the standard deviation $\sqrt{D_k}$.

The scaled dot-product self-attention now takes the form:

$$Y = \mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{D_k}})V$$

Scaled self-attention

Figure: Information flow in a scaled dot-product self-attention neural network layer



Scaled self-attention

Algorithm 1: Scaled dot-product self-attention

$Q \leftarrow XW^{(q)}$;

$K \leftarrow XW^{(k)}$;

$V \leftarrow XW^{(v)}$;

return $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{D_k}})V$;

Multi-head attention

We can use multiple attention heads in parallel to attend to multiple data-dependent patterns at the same time. Suppose we have C heads:

$$\begin{aligned} H_c &= \text{Attention}(Q_c, K_c, V_c) \\ &= \text{Attention}(XW_c^{(q)}, XW_c^{(k)}, XW_c^{(v)}) \end{aligned}$$

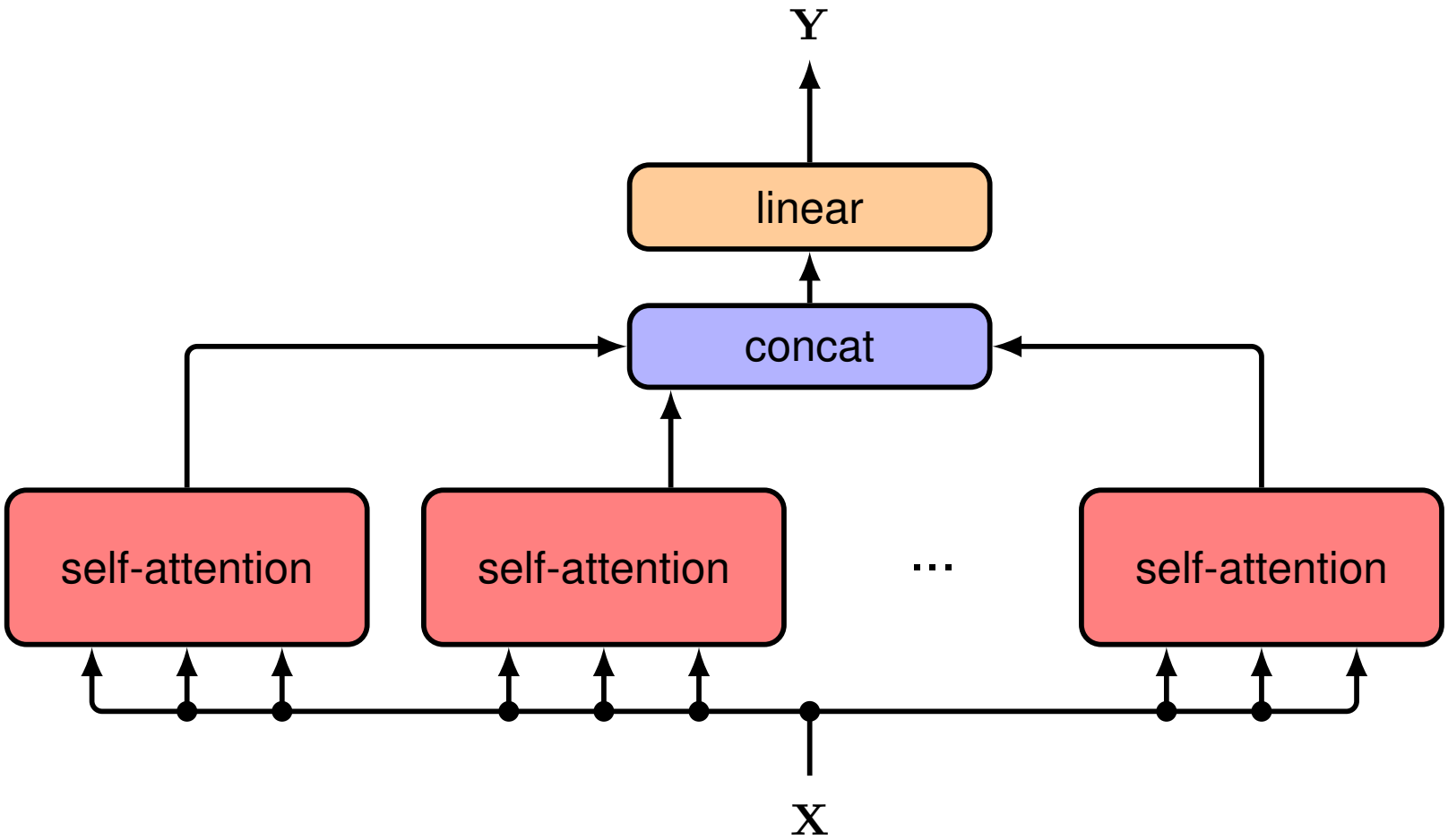
The heads are first concatenated into a single matrix, and the result is then linearly transformed using a matrix $W^{(o)}$ to give a combined output:

$$Y = (H_1, \ldots, H_C)W^{(o)}$$

The matrix $W^{(o)}$ has dimension $HD_v \times D$, so that the final output matrix Y has dimension $N \times D$.

Multi-head attention

Figure: Information flow in a multi-head attention layer



Multi-head attention

Algorithm 2: Multi-head attention

for $c \leftarrow 1$ **to** C **do**

$Q_c = XW_c^{(q)}$;

$K_c = XW_c^{(k)}$;

$V_c = XW_c^{(v)}$;

$H_c = \text{Attention}(Q_c, K_c, V_c)$;

end

$H = (H_1, \ldots, H_C)$;

return $Y = HW^{(o)}$;

Transformer layers

To improve training efficiency, we can introduce residual connections and layer normalization:

$$Z = \mathrm{LayerNorm}(Y(X) + X)$$

Or using pre-norm:

$$Z = Y(\mathrm{LayerNorm}(X)) + X$$

Until now, the output data matrix Y is still a linear transformation of the input data matrix X, and this limits the expressive capabilities of the attention layer. We can enhance the flexibility by post-processing the outputs using a standard nonlinear neural network denoted MLP:

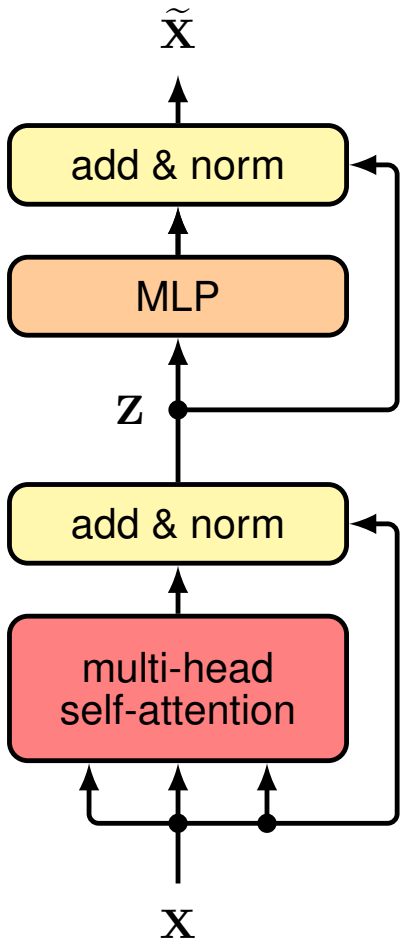
$$\tilde{X} = \mathrm{LayerNorm}(\mathrm{MLP}(Z) + Z)$$

Again, we can use a pre-norm instead:

$$\tilde{X} = \mathrm{MLP}(\mathrm{LayerNorm}(Z)) + Z$$

Transformer layers

Figure: One layer of the transformer architecture



Computational complexity

- In the attention layer:
  - Calculate the matrices Q, K and V: $\mathcal{O}(ND^2)$.
  - Calculate the dot products QK^T: $\mathcal{O}(N^2D)$.
  - Calculate the matrix Y: $\mathcal{O}(N^2D)$.
- In the neural network layer:
  - Calculate the matrix $\tilde{X}$: $\mathcal{O}(ND^2)$.

Positional encoding

- A transformer is equivariant with respect to input permutations due to the shared matrices $W_c^{(q)}$, $W_c^{(k)}$, $W_c^{(v)}$ and the shared subsequent neural network.
- The lack of dependence on token order becomes a major limitation when we consider sequential data, so we need to find a way to inject token order information into the network.

Positional encoding

The requirements for a positional encoding:

- The token order should be encoded in the data itself instead of having to be represented in the network architecture.
- We will construct a position encoding vector r_n associated with each input position n and then combine this with the associated input token x_n:
  - [No] $\tilde{x}_n = (x_n, r_n)$.
  - [Yes] $\tilde{x}_n = x_n + r_n$.
- r_n should be bounded.
- r_n should generalize well to new input sequences that are longer than those used in training.
- r_n should be unique for a given position.
- r_n should have a consistent way to express the number of steps between any two input tokens irrespective of their absolute position.

Positional encoding

Positional encoding based on sinusoidal functions:

$$r_{ni} = \begin{cases} \sin(\frac{n}{L^{\frac{i}{D}}}), \text{if } i \text{ is even} \\ \cos(\frac{n}{L^{\frac{i-1}{D}}}), \text{if } i \text{ is odd} \end{cases}$$