The Mathematics Behind LLM Transformers

Carles Casacuberta

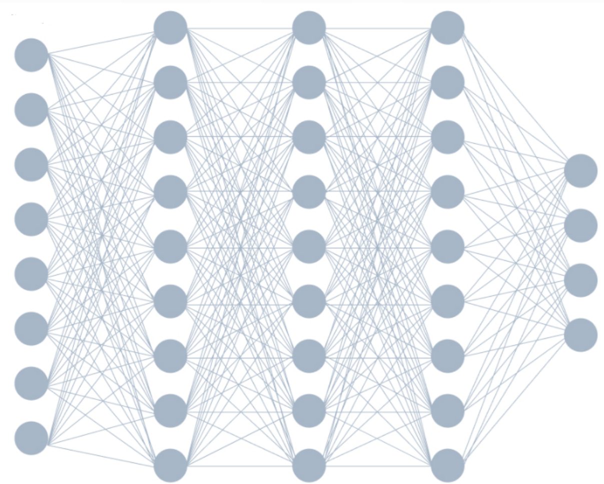
Universitat de Barcelona

Seminari Cuc

Proof Theory and Foundations of Mathematics

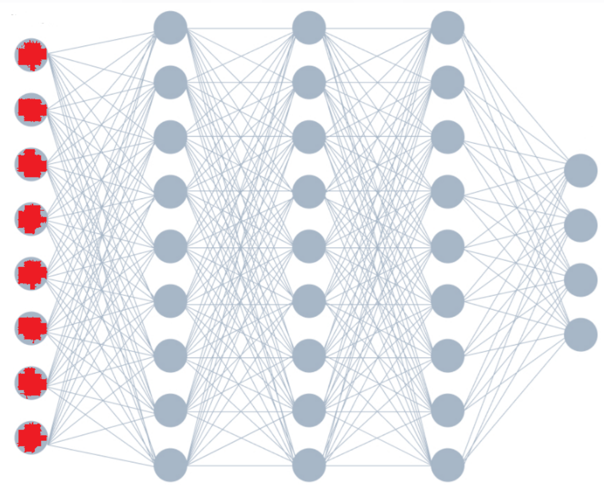
8 May 2025

Multilayer Perceptron



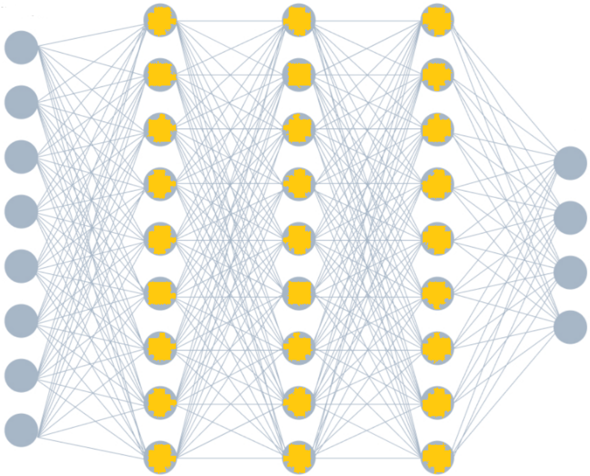
A fully connected **feed-forward** neural network

Multilayer Perceptron



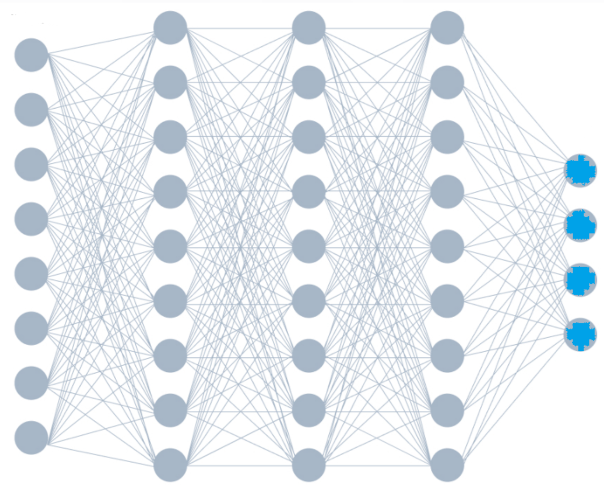
Input layer

Multilayer Perceptron



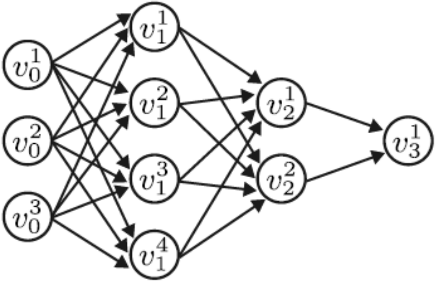
Hidden layers

Multilayer Perceptron



Output layer

Neurons



9

$\boldsymbol{v_0^1, \ldots, v_0^{N_0}}$**:** Neurons of the input layer.

For $\ell = 1, \ldots, L$ (number of hidden layers plus output layer),

$\boldsymbol{v_\ell^1, \ldots, v_\ell^{N_\ell}}$**:** Neurons of the ℓ-th layer.

Activation

For $x \in \mathbb{R}^{N_0}$ and $\ell = 0, \ldots, L$, compute **hidden states** $h_\ell(x) \in \mathbb{R}^{N_\ell}$ recursively as $h_0(x) = x$ and

$$\boxed{h_\ell(x) = \sigma\Big(W_\ell\, h_{\ell-1}(x) + b_\ell\Big)}$$

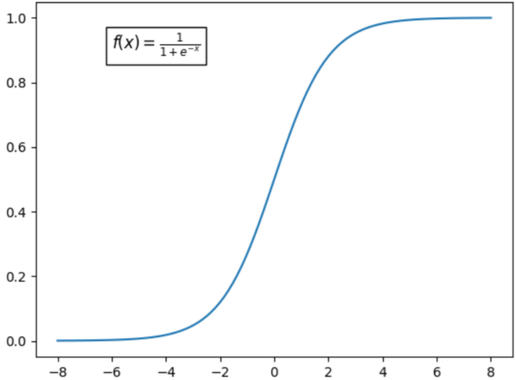
where σ is a nonlinear, almost everywhere smooth **activation function** applied entrywise, W_ℓ is a matrix of **weights,** and b_ℓ is a vector of **biases.** The set θ of entries of W_ℓ and b_ℓ is the set of **learnable parameters** of the network.

For an ordered collection $x_1, \ldots, x_n$ of **input data,** the vector

$$(h_\ell(x_1)^k, \ldots, h_\ell(x_n)^k)$$

is the **activation vector** of the neuron v_ℓ^k at $x_1, \ldots, x_n$.

Activation



The **sigmoid** is a common activation function

Loss Function

In a **supervised task,** one considers a function $\phi\colon X \to Y$, where X is a set of **inputs** and Y is a set of **labels**, e.g., $Y = \mathbb{R}$ for a **regression** task, or Y finite for a **classification** task.

In practice,

- a **training set** $S_{\text{train}} \subseteq X \times Y$ and
- a **test set** $S_{\text{test}} \subseteq X \times Y$ are chosen.

If $\mathcal{L}\colon Y \times Y \to \mathbb{R}$ is a suitable **loss function** and a subset $S = \{(x_i, y_i)\}_{i=1}^{n} \subseteq X \times Y$ is given, the **empirical risk** is

$$R_S = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\Big(h_L(x_i), y_i, \theta\Big)$$

where $h_L(x_i)$ are **predicted** values and y_i are **true** values.

Generalization Gap

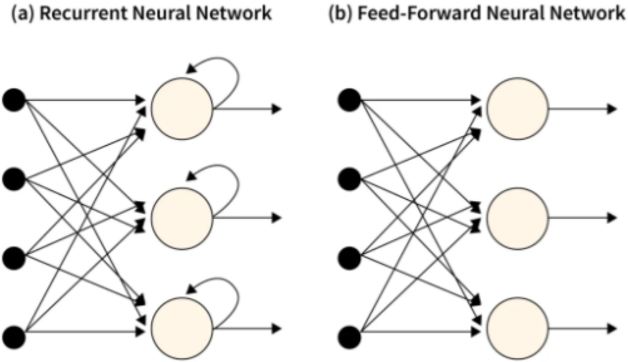
After learning with **backpropagation** the parameter set θ by minimizing the empirical risk R_{train}, the **generalization gap** is

$$\boxed{G = R_{\text{test}} - R_{\text{train}}}$$

To prevent overfitting and reduce the generalization gap, a **regularization function** $\mathcal{T}$ can be used as follows:

$$\widetilde{R}_S = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\Big(h_L(x_i) + \mathcal{T}(x_i), y_i, \theta\Big).$$

Recurrent Neural Networks



Recurrent Neural Networks

For a recurrent neural network, input data (**tokens**) are **sequentially** ordered: $w_1, \ldots, w_T$.

Each token w_t is converted into a vector $x_t \in \mathbb{R}^d$ by means of a suitable **embedding** followed by **positional encoding,** which preserves the token order in the sequence $x_1, \ldots, x_T$.

Hidden states for each input x_t are computed at layer ℓ as

$$\boxed{h_\ell(x_t) = \sigma\Big(W_\ell^x\, h_{\ell-1}(x_t) + W_\ell^h\, h_\ell(x_{t-1}) + b_\ell\Big)}$$

where W_ℓ^x and W_ℓ^h are matrices of weights to be learned, and b_ℓ is a bias vector, also to be learned.

Recurrent Neural Networks

The **output** of a RNN is, for each t,

$$y_t = W^{\text{out}}\, h_L(x_t) + b^{\text{out}}.$$

Here W^{out} is a matrix of dimensions $N \times d$, where N is the size of the vocabulary. Outputs are converted with the **softmax** function:

$$\hat{y}_t[i] = \text{softmax}(y_t)[i] = \frac{\exp(y_t[i])}{\sum_{j=1}^{N} \exp(y_t[j])}, \quad i = 1, \dots, N.$$

The model is trained using **cross-entropy** as loss function:

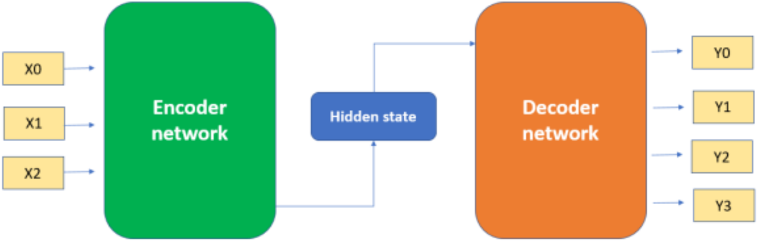
$$\mathcal{L} = -\log \hat{y}_t[\,y_t],$$

where $\hat{y}_t[w]$ is the probability assigned by $\hat{y}_t$ to a token w.

Main difficulty: **Vanishing gradient** during backpropagation.

Encoders

Encoder-decoder architectures are used for sequence learning in **generative** artificial intelligence.



Self-Attention

Transformers rely on **self-attention** in both encoders and decoders, which replaces recurrence by enabling all tokens to interact with each other simultaneously.

Self-attention assigns to each state $h_t \in \mathbb{R}^d$ (either an input or a hidden state from a previous step) three vectors: **queries** (q_t), **keys** (k_t), and **values** (v_t), by means of linear projections:

$$q_t = W^Q h_t, \qquad k_t = W^K h_t, \qquad v_t = W^V h_t,$$

where W^Q, W^K and W^V are $d \times d$ learnable matrices, and d is called **model length.**

Self-Attention

Then **alignment scores** are defined for each pair h_i, h_j as

$$e_{ij} = \frac{k_i \cdot q_j}{\sqrt{d}},$$

where d is the model length.

Alignment scores are normalized using the **softmax** function:

$$a_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{t=1}^{T} \exp(e_{it})}.$$

Thus, a_{ij} can be interpreted as a probability that state h_i is relevant to state h_j.

Alignment Scores

Normalized alignment scores are used to compute a weighted average of values:

$$C = (c_1, \ldots, c_T), \qquad c_i = \sum_{t=1}^{T} a_{it} v_t,$$

which is called the **attention matrix** of $H = (h_1, \ldots, h_T)$:

$$C = \mathrm{Att}(H) = \mathrm{Att}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^{\mathrm{t}}}{\sqrt{d}} V\right).$$

Multi-head Attention

There is a **multi-head attention** layer in the transformer architecture. The multi-head attention layer works as **multiple parallel attention mechanisms,** called **heads.**

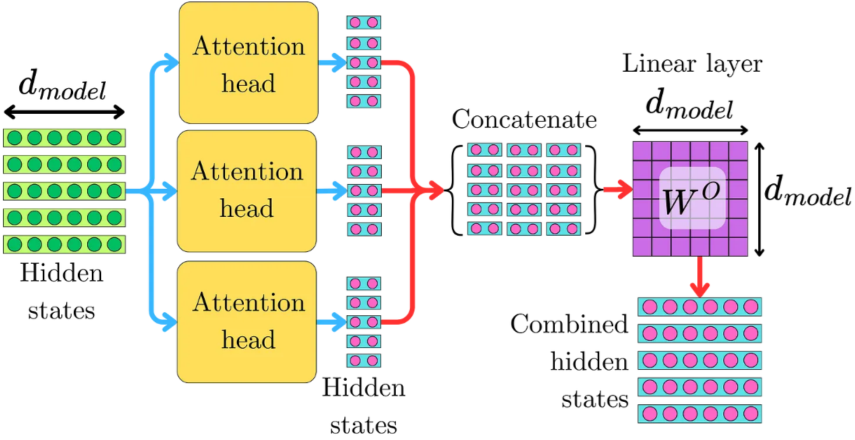
The number n of heads must divide the model length d.

Denote by $H = (h_1, \ldots, h_T)$ the incoming hidden states, and let

$$\mathrm{Mult}(H) = \mathrm{concat}(\mathrm{head}_1, \ldots, \mathrm{head}_n)\, W^O, \quad \mathrm{head}_i = \mathrm{Att}_i(H),$$

where n is the number of heads and W^O is another learnable matrix. Here the matrices W_i^Q, W_i^K and W_i^V used to compute Att_i for each i are of size $d \times (d/n)$, so W^O is again a $d \times d$ matrix.

Multi-head Attention



Transformers

In a transformer, each **encoder** has two layers: the self-attention mechanism followed by a two-step feed-forward neural network:

$$h_t^{\text{enc}} = W_2\,\sigma(W_1 h_t + b_1) + b_2.$$

Thus, an encoder receives a sequence of tokens $w_1, \ldots, w_T$. Each token w_t is converted into a vector $x_t \in \mathbb{R}^d$ using **positional encoding.** The vectors $x_1, \ldots, x_T$ pass through several **attention blocks,** and the encoder returns a matrix H^{enc} of size $T \times d$.

Transformers

The **decoder** receives H^{enc} together with $x_1, \ldots, x_T$ during training (**teacher forcing**), or the previous predicted values during generation (**autoregression**). The decoder generates an output matrix H^{dec} by means of a **masked** self-attention layer, a **cross-atention** layer, and a feed-forward neural network.

Masked self-attention has

$$\text{Att}(\boldsymbol{q_t}, K, V) = \sum_{j=1}^{\boldsymbol{t}} a_{tj} v_j;$$

hence, **future** tokens $j > t$ are not included.

Cross-attention recovers information from H^{enc} as follows: the query matrix Q is computed as $W^Q H^{\text{dec}}$ but the key and value matrices are computed as $W^K H^{\text{enc}}$ and $W^V H^{\text{enc}}$.

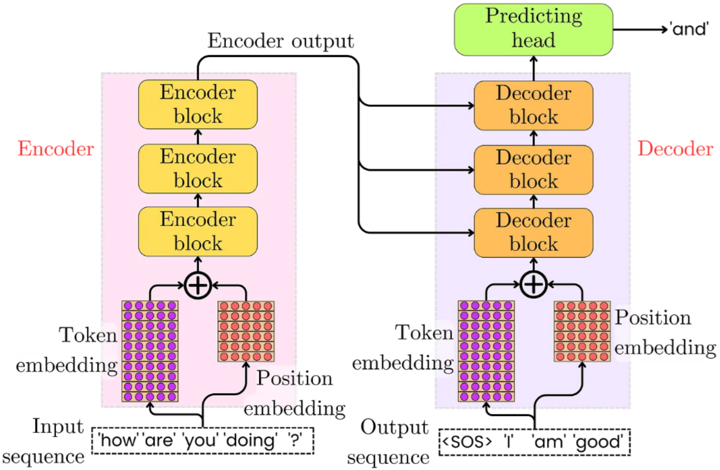
Transformers

The transformer ends with a **predicting head:** A classifier over the whole token vocabulary made out of a linear layer followed by softmax, converting the decoder's final hidden states into **token probabilities** to predict the next word in a sequence:

$$\hat{y}_t = \mathrm{softmax}\big(W^{\mathrm{out}} h_t^{\mathrm{dec}} + b\big).$$

Here h^{dec} is the decoder's output and $\hat{y}_t \in \mathbb{R}^N$ are probabilities for each token, where N is the length of the vocabulary. The size of the matrix W^{out} is $N \times d$.

Transformers



Transformers