Deep Learning

Last update: 12 March 2024

Academic Year 2023 – 2024

Alma Mater Studiorum · University of Bologna

Contents

1 Neural networks expressivity

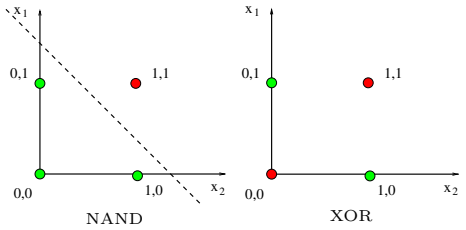
1.1 Perceptron

Single neuron that defines a binary threshold through a hyperplane:

$$\begin{cases} 1 & \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Expressivity A perceptron can represent a NAND gate but not a XOR gate.

Perceptron expressivity



Remark. Even if NAND is logically complete, the strict definition of a perceptron is not a composition of them.

1.2 Multi-layer perceptron

Composition of perceptrons.

Shallow neural network Neural network with one hidden layer.

Shallow NN

Deep neural network Neural network with more than one hidden layer.

Deep NN

Expressivity Shallow neural networks allow to approximate any continuous function

Multi-layer perceptron expressivity

$$f : \mathbb{R} \to [0, 1]$$

Remark. Still, deep neural networks allow to use less neural units.

2 Training

2.1 Gradient descent

Gradient descent

1. Start from a random set of weights w.
2. Compute the gradient $\nabla\mathcal{L}$ of the loss function.
3. Make a small step of size $-\nabla\mathcal{L}(w)$.
4. Go to 2., until convergence.

Learning rate

Learning rate Size of the step. Usually denoted with μ.

$$w = w + \mu\nabla\mathcal{L}(w)$$

Optimizer

Optimizer Algorithm that tunes the learning rate during training.

Stochastic gradient descent

Stochastic gradient descent Use a subset of the training data to compute the gradient.

Full-batch Use the entire dataset.

Mini-batch Use a subset of the training data.

Online Use a single sample.

Remark. SGD with mini-batch converges to the same result obtained using a full-batch approach.

Momentum

Momentum Correct the update v_t at time t considering the update v_{t-1} of time $t-1$.

$$\begin{aligned} w_{t+1} &= w_t + v_t \\ v_t &= \mu\nabla\mathcal{L}(w_t) + \alpha v_{t-1} \end{aligned}$$

Nesterov momentum

Nesterov momentum Apply the momentum before computing the gradient.

Overfitting

Overfitting Model too specialized on the training data.

Methods to reduce overfitting are:

- Increasing the dataset size.
- Simplifying the model.
- Early stopping.
- Regularization.
- Model averaging.
- Neurons dropout.

Underfitting

Underfitting Model too simple and unable to capture features of the training data.

2.2 Backpropagation

Chain rule Refer to `SMM for AI (Section 5.1.1)`. Chain rule

Backpropagation Algorithm to compute the gradient at each layer of a neural network. Backpropagation

The output of the i-th neuron in the layer l of a neural network can be defined as:

$$a_{l,i} = \sigma_{l,i}(\mathbf{w}_{l,i}^T \mathbf{a}_{l-1} + b_{l,i}) = \sigma_{l,i}(z_{l,i})$$

where:

- $a_{l,i} \in \mathbb{R}$ is the output of the neuron.
- $\mathbf{w}_{l,i} \in \mathbb{R}^{n_{l-1}}$ is the vector of weights.
- $\mathbf{a}_{l-1} \in \mathbb{R}^{n_{l-1}}$ is the vector of the outputs of the previous layer.
- $b_{l,i} \in \mathbb{R}$ is the bias.
- $\sigma_{l,i} : \mathbb{R} \to \mathbb{R}$ is the activation function[1].
- $z_{l,i}(\mathbf{w}_{l,i}, b_{l,i} | \mathbf{a}_{l-1}) = \mathbf{w}_{l,i}^T \mathbf{a}_{l-1} + b_{l,i}$ is the argument of the activation function and is parametrized on $\mathbf{w}_{l,i}$ and $b_{l,i}$.

Hence, the outputs of the l-th layer can be defined as:

$$\mathbf{a}_l = \sigma_l(\boldsymbol{W}_l^T \mathbf{a}_{l-1} + \mathbf{b}_l) = \sigma_l(\mathbf{z}_l(\boldsymbol{W}_l, \mathbf{b}_l | \mathbf{a}_{l-1}))$$

where:

- $\sigma_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is the element-wise activation function.
- $\boldsymbol{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$, $\mathbf{a}_{l-1} \in \mathbb{R}^{n_{l-1}}$, $\mathbf{b}_l \in \mathbb{R}^{n_l}$, $\mathbf{a}_l \in \mathbb{R}^{n_l}$.

Finally, a neural network with input $\mathbf{x}$ can be expressed as:

$$\begin{aligned} \mathbf{a}_0 &= \mathbf{x} \\ \mathbf{a}_i &= \sigma_i(\mathbf{z}_i(\boldsymbol{W}_i, \mathbf{b}_i | \mathbf{a}_{i-1})) \end{aligned}$$

Given a neural network with K layers and a loss function $\mathcal{L}$, we want to compute the derivative of $\mathcal{L}$ w.r.t. the weights of each layer to tune the parameters.

First, we highlight the parameters of each of the functions involved:

Loss $\mathcal{L}(a_K) = \mathcal{L}(\sigma_K)$ takes as input the output of the network (i.e. the output of the last activation function).

Activation function $\sigma_i(\mathbf{z}_i)$ takes as input the value of the neurons at the i-th layer.

Neurons $\mathbf{z}_i(\boldsymbol{W}_i, \mathbf{b}_i)$ takes as input the weights and biases at the i-th layer.

Let $\odot$ be the Hadamard product. By exploiting the chain rule, we can compute the derivatives w.r.t. the weights going backward:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_K} = \frac{\partial \mathcal{L}}{\partial \sigma_K} \frac{\partial \sigma_K}{\partial \mathbf{z}_K} \frac{\partial \mathbf{z}_K}{\partial \boldsymbol{W}_K} = \underset{\mathbb{R}^{n_K \times 1}}{\nabla \mathcal{L}(\mathbf{a}_K)} \odot \underset{\mathbb{R}^{n_K \times 1}}{\nabla \sigma_K(\mathbf{z}_K)} \cdot \underset{1 \times \mathbb{R}^{n_{K-1}}}{\mathbf{a}_{K-1}^T} \in \mathbb{R}^{n_K \times n_{K-1}}$$

[1] Even if it is possible to have a different activation function in each neuron, in practice, each layer has the same activation function.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{K-1}} &= \frac{\partial \mathcal{L}}{\partial \sigma_K}\frac{\partial \sigma_K}{\partial \mathbf{z}_K}\frac{\partial \mathbf{z}_K}{\partial \sigma_{K-1}}\frac{\partial \sigma_{K-1}}{\partial \mathbf{z}_{K-1}}\frac{\partial \mathbf{z}_{K-1}}{\partial \boldsymbol{W}_{K-1}} \\ &= (\underset{\mathbb{R}^{n_K\times 1}}{\nabla \mathcal{L}(\mathbf{a}_K)} \odot \underset{\mathbb{R}^{n_K\times 1}}{\nabla \sigma_K(\mathbf{z}_K)})^T \cdot \underset{\mathbb{R}^{n_K}\times\mathbb{R}^{n_{K-1}}}{\boldsymbol{W}_K} \odot \underset{\mathbb{R}^{n_{K-1}\times 1}}{\nabla \sigma_{K-1}(\mathbf{z}_{K-1})} \cdot \underset{1\times\mathbb{R}^{n_{K-2}}}{\mathbf{a}_{K-2}^T} \in \mathbb{R}^{n_{K-1}\times n_{K-2}}\end{aligned}$$

$$\vdots$$

In the same way, we can compute the derivatives w.r.t. the biases:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}_K} = \frac{\partial \mathcal{L}}{\partial \sigma_K}\frac{\partial \sigma_K}{\partial \mathbf{z}_K}\frac{\partial \mathbf{z}_K}{\partial \mathbf{b}_K} = \underset{\mathbb{R}^{n_K\times 1}}{\nabla \mathcal{L}(\mathbf{a}_K)} \odot \underset{\mathbb{R}^{n_K\times 1}}{\nabla \sigma_K(\mathbf{z}_K)} \cdot 1 \in \mathbb{R}^{n_K}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{b}_{K-1}} &= \frac{\partial \mathcal{L}}{\partial \sigma_K}\frac{\partial \sigma_K}{\partial \mathbf{z}_K}\frac{\partial \mathbf{z}_K}{\partial \sigma_{K-1}}\frac{\partial \sigma_{K-1}}{\partial \mathbf{z}_{K-1}}\frac{\partial \mathbf{z}_{K-1}}{\partial \mathbf{b}_{K-1}} \\ &= (\underset{\mathbb{R}^{n_K\times 1}}{\nabla \mathcal{L}(\mathbf{a}_K)} \odot \underset{\mathbb{R}^{n_K\times 1}}{\nabla \sigma_K(\mathbf{z}_K)})^T \cdot \underset{\mathbb{R}^{n_K}\times\mathbb{R}^{n_{K-1}}}{\boldsymbol{W}_K} \odot \underset{\mathbb{R}^{n_{K-1}\times 1}}{\nabla \sigma_{K-1}(\mathbf{z}_{K-1})} \cdot 1 \in \mathbb{R}^{n_{K-1}}\end{aligned}$$

$$\vdots$$

It can be noticed that many terms are repeated from one layer to another. By exploiting this, we can store the following intermediate values:

$$\begin{aligned}\delta_K &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}_K} = \frac{\partial \mathcal{L}}{\partial \sigma_K}\frac{\partial \sigma_K}{\partial \mathbf{z}_K} = \nabla \mathcal{L}(\mathbf{a}_K) \odot \nabla \sigma_K(\mathbf{z}_K) \\ \delta_l &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l} = \delta_{l+1}^T \cdot \boldsymbol{W}_{l+1} \odot \nabla \sigma_l(\mathbf{z}_l)\end{aligned}$$

and reused them to compute the derivatives as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_l} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l}\frac{\partial \mathbf{z}_l}{\partial \boldsymbol{W}_l} = \delta_l \cdot \mathbf{a}_{l-1}^T \\ \frac{\partial \mathcal{L}}{\partial \mathbf{b}_l} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}_l}\frac{\partial \mathbf{z}_l}{\partial \mathbf{b}_l} = \delta_l \cdot 1\end{aligned}$$

Vanishing gradient As backpropagation consists of a chain of products, when a component is small (i.e. < 1), it will gradually cancel out the gradient when backtracking, causing the first layers to learn much slower than the last layers. Vanishing gradient

Remark. This is an issue of the sigmoid function. ReLU was designed to solve this problem.