MO434 - Deep Learning
Fundamentals of (Deep) Neural Networks II

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

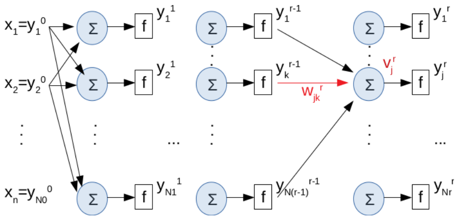
afalcao@ic.unicamp.br

Agenda

- A neural network with dense layers only – a Multi-Layer Perceptron (MLP).
- Activation and loss functions.
- Stochastic Gradient Descent (SGD) optimizer.
- The backpropagation algorithm.

Neural network with dense layers only

Consider a neural network with L dense layers and N_r neurons at layer $1 \leq r \leq L$.



- Each neuron $j \in [1, N_r]$ of a layer r has a weight vector $\boldsymbol{w}_j^r = (w_{j0}^r, w_{j1}^r, \ldots, w_{jN_{r-1}}^r)$ with bias w_{j0}^r,
- the input of layer r is the vector $\boldsymbol{y}^{r-1} = (1, y_1^{r-1}, y_2^{r-1}, \ldots, y_{N_{r-1}}^{r-1})$ and
- each perceptron j computes $v_j^r = \langle \boldsymbol{y}^{r-1}, \boldsymbol{w}_j^r \rangle$ followed by $f(v_j^r)$, where f is a differentiable activation function.

Examples of activation functions

Rectified Linear Unit (ReLU)

$$f(v) = \begin{cases} v & v > 0, \\ 0 & v \leq 0. \end{cases}$$

Logistic $(a > 0)$

$$f(v) = \frac{1}{1 + e^{-av}}.$$

Hyperbolic tangent

$$f(v) = tanh(v) = \frac{2}{1 + e^{-2v}} - 1$$

SoftPlus

$$f(v) = \log_e(1 + e^v)$$

ReLU derivative

$$f'(v) = \begin{cases} 1 & v > 0, \\ 0 & v \leq 0. \end{cases}$$

Logistic derivative

$$f'(v) = af(v)(1 - f(v)).$$

Hyperbolic tangent derivative

$$f'(v) = 1 - f^2(v)$$

SoftPlus derivative

$$f'(v) = \frac{1}{1 + e^{-v}}.$$

Examples of activation functions

Exponential Linear Unit (ELU)

$$f(v) = \begin{cases} a(e^v - 1) & v \leq 0, \\ v & v > 0. \end{cases}$$

ELU derivative

$$f'(v) = \begin{cases} ae^v & v \leq 0, \\ 1 & v > 0. \end{cases}$$

Scaled ELU (SELU)

$$f(v) = \lambda \begin{cases} a(e^v - 1) & v \leq 0, \\ v & v > 0. \end{cases}$$

SELU derivative

$$f'(v) = \begin{cases} \lambda ae^v & v \leq 0, \\ \lambda & v > 0. \end{cases}$$

Leaky RELU

$$f(v) = \lambda \begin{cases} av & v \leq 0, \\ v & v > 0. \end{cases}$$

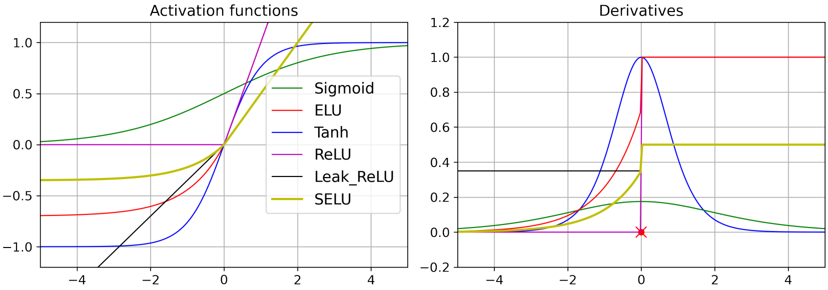
Leaky RELU derivative

$$f'(v) = \lambda \begin{cases} a & v \leq 0, \\ 1 & v > 0. \end{cases}$$

Activation functions

- RELU might set zero irreversibly to neuron outputs, which motivated the variants SELU, ELU, and Leaky RELU.

- To avoid gradient instabilities, SELU $>$ ELU $>$ Leaky RELU $>$ RELU $>$ tanh $>$ logistic, but RELU is the most popular.



Let's play with ACTIVATION FUNCTIONS.

Activation functions at the decision layer

At the decision layer, the choice of the activation function depends on the problem:

- Regression.
- Binary classification.
- Categorical classification.

Activation functions at the decision layer

- Regression: we do not usually want to limit the output of the NN, then RELU and SoftPlus can be used. Otherwise, we may limit it within [-1,1] using tanh or within [0,1] using logistic.

Activation functions at the decision layer

- Regression: we do not usually want to limit the output of the NN, then RELU and SoftPlus can be used. Otherwise, we may limit it within [-1,1] using tanh or within [0,1] using logistic.

- Binary classification: it can be solved as a regression problem with a single output value within [0,1], such that samples with output $0 \leq y^L < 0.5$ are assigned to class ω_1 and output $0.5 \leq y^L \leq 1$ to class ω_2. Hence, the logistic can be used.

Activation functions at the decision layer

- Regression: we do not usually want to limit the output of the NN, then RELU and SoftPlus can be used. Otherwise, we may limit it within [-1,1] using tanh or within [0,1] using logistic.

- Binary classification: it can be solved as a regression problem with a single output value within [0,1], such that samples with output $0 \leq y^L < 0.5$ are assigned to class ω_1 and output $0.5 \leq y^L \leq 1$ to class ω_2. Hence, the logistic can be used.

- For categorical classification SoftMax is usually adopted. For each neuron $j \in [1, N_L]$, where N_L is the number of classes,

$$f(v_j^L) = \frac{e^{v_j^L}}{\sum_{k=1}^{N_L} e^{v_k^L}}.$$

Then class $\omega_c = \arg\max_{k \in [1,N_L]} \{f(v_k^L)\}$, $c \in [1, N_L]$ is chosen.

Examples of loss functions

The weights of the NN are optimized based on a loss function, whose choice also depends on the problem.

The weights of the NN are optimized based on a loss function, whose choice also depends on the problem.

- For regression, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are classical loss functions.

Examples of loss functions

The weights of the NN are optimized based on a loss function, whose choice also depends on the problem.

- For regression, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are classical loss functions.
- Binary cross-entropy (BCE) is used for binary classification.

Examples of loss functions

The weights of the NN are optimized based on a loss function, whose choice also depends on the problem.

- For regression, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are classical loss functions.
- Binary cross-entropy (BCE) is used for binary classification.
- Categorical cross-entropy (CCE) is commonly used for multi-class classification.

Examples of loss functions

The weights of the NN are optimized based on a loss function, whose choice also depends on the problem.

- For regression, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are classical loss functions.
- Binary cross-entropy (BCE) is used for binary classification.
- Categorical cross-entropy (CCE) is commonly used for multi-class classification.

Let $s \in \mathcal{Z}_{tr}$ be a sample of a training set with N samples, $\boldsymbol{x}(s)$ be its feature vector (input of the NN), and the desired and estimated outputs at the decision layer be $\boldsymbol{y}(s)$ and $\boldsymbol{y}^L(s)$, respectively.

Examples of loss functions

They are all based on $\frac{1}{|\mathcal{Z}_{tr}|}\sum_{s\in\mathcal{Z}_{tr}}\mathcal{E}(s)$, where

- for MSE,

$$\mathcal{E}(s) = \frac{1}{N_L}\sum_{j=1}^{N_L}(y_j(s) - y_j^L(s))^2,$$

- for MAE,

$$\mathcal{E}(s) = \frac{1}{N_L}\sum_{j=1}^{N_L}|y_j(s) - y_j^L(s))|,$$

- for BCE, $y(s)$ and $y^L(s)$ must be in [0,1],

$$\mathcal{E}(s) = -(y(s)\log(y^L(s)) + (1-y(s))\log(1-y^L(s))),$$

- and for CCE, $y_j(s)$ and $y_j^L(s)$ must be in [0,1],

$$\mathcal{E}(s) = -\sum_{j=1}^{N_L} y_j(s)\log(y_j^L(s)).$$

SGD optimizer

For $\boldsymbol{w}_j^r$, each iteration i adjusts its weights by

$$
\begin{aligned}
\boldsymbol{w}_j^r(i+1) &= \boldsymbol{w}_j^r(i) + \Delta\boldsymbol{w}_j^r, \\
\Delta\boldsymbol{w}_j^r &= -\mu\frac{\partial J}{\partial \boldsymbol{w}_j^r}, \\
J &= \sum_{s\in\mathcal{Z}_{tr}} \mathcal{E}(s)
\end{aligned}
$$

for a fixed learning rate μ and error function $\mathcal{E}$.

SGD optimizer

For $\mathbf{w}_j^r$, each iteration i adjusts its weights by

$$\begin{aligned} \mathbf{w}_j^r(i+1) &= \mathbf{w}_j^r(i) + \Delta\mathbf{w}_j^r, \\ \Delta\mathbf{w}_j^r &= -\mu\frac{\partial J}{\partial \mathbf{w}_j^r}, \\ J &= \sum_{s\in\mathcal{Z}_{tr}} \mathcal{E}(s) \end{aligned}$$

for a fixed learning rate μ and error function $\mathcal{E}$.

Given the pairs $(\mathbf{x}(s), \mathbf{y}(s))$, $s \in \mathcal{Z}_{tr}$, with the input and desired output vectors, one can choose $\mathcal{E}(s)$ as

$$\mathcal{E}(s) = \frac{1}{2}\|\mathbf{y}^L(s) - \mathbf{y}(s)\|^2 = \frac{1}{2}\sum_{m=1}^{N_L}(y_m^L(s) - y_m(s))^2 = \frac{1}{2}\sum_{m=1}^{N_L} e_m^2(s),$$

where $\mathbf{y}^L(s)$ is the estimated output vector.

SGD optimizer

For $\Delta \mathbf{w}_j^r$, we must compute $\frac{\partial J}{\partial \mathbf{w}_j^r} = \sum_{s \in \mathcal{Z}_{tr}} \frac{\partial \mathcal{E}(s)}{\partial \mathbf{w}_j^r}$. By the chain rule,

$$\frac{\partial \mathcal{E}(s)}{\partial \mathbf{w}_j^r} = \frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} \frac{\partial v_j^r(s)}{\partial \mathbf{w}_j^r}.$$

SGD optimizer

For $\Delta \boldsymbol{w}_j^r$, we must compute $\frac{\partial J}{\partial \boldsymbol{w}_j^r} = \sum_{s \in \mathcal{Z}_{tr}} \frac{\partial \mathcal{E}(s)}{\partial \boldsymbol{w}_j^r}$. By the chain rule,

$$\frac{\partial \mathcal{E}(s)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} \frac{\partial v_j^r(s)}{\partial \boldsymbol{w}_j^r}.$$

Given that $v_j^r(s) = \sum_{m=0}^{N_{r-1}} w_{jm}^r y_m^{r-1}(s) = \langle \boldsymbol{w}_j^r, \boldsymbol{y}^{r-1}(s) \rangle$,

$$\frac{\partial v_j^r(s)}{\partial \boldsymbol{w}_j^r} = \boldsymbol{y}^{r-1}(s).$$

For $\Delta \boldsymbol{w}_j^r$, we must compute $\frac{\partial J}{\partial \boldsymbol{w}_j^r} = \sum_{s \in \mathcal{Z}_{tr}} \frac{\partial \mathcal{E}(s)}{\partial \boldsymbol{w}_j^r}$. By the chain rule,

$$\frac{\partial \mathcal{E}(s)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} \frac{\partial v_j^r(s)}{\partial \boldsymbol{w}_j^r}.$$

Given that $v_j^r(s) = \sum_{m=0}^{N_{r-1}} w_{jm}^r y_m^{r-1}(s) = \langle \boldsymbol{w}_j^r, \boldsymbol{y}^{r-1}(s) \rangle$,

$$\frac{\partial v_j^r(s)}{\partial \boldsymbol{w}_j^r} = \boldsymbol{y}^{r-1}(s).$$

Let us now define $\frac{\partial \mathcal{E}(s)}{\partial v_j^r(s)} = \delta_j^r(s)$, such that

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{s \in \mathcal{Z}_{tr}} \delta_j^r(s) \boldsymbol{y}^{r-1}(s).$$

The computation of $\delta_j^r(s)$ starts from $r = L$ and propagates backward for $1 \leq r < L$, deriving the name backpropagation algorithm.

The computation of $\delta_j^r(s)$ starts from $r = L$ and propagates backward for $1 \leq r < L$, deriving the name backpropagation algorithm.

For $r = L$ and $1 \leq j \leq N_L$,

$$
\begin{aligned}
\delta_j^L(s) &= \frac{\partial \mathcal{E}(s)}{\partial v_j^L(s)} = \frac{\partial \left(\frac{1}{2} \sum_{m=1}^{N_L} \left(f(v_m^L(s)) - y_m(s) \right)^2 \right)}{\partial v_j^L(s)} \\
\delta_j^L(s) &= (f(v_j^L(s)) - y_j(s)) \frac{\partial f(v_j^L(s))}{\partial v_j^L(s)} = e_j(s) f'(v_j^L(s)) \\
\delta_j^L(s) &= e_j(s) f'(v_j^L(s)).
\end{aligned}
$$

SGD optimizer

For $r < L$ and $1 \leq j \leq N_{r-1}$, $v_j^{r-1}(s)$ affects all $v_k^r(s)$, $k = 1, 2, \ldots, N_r$. Therefore, the chain rule must be applied.

$$
\begin{aligned}
\delta_j^{r-1}(s) &= \sum_{k=1}^{N_r} \frac{\partial \mathcal{E}(s)}{\partial v_k^r(s)} \frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} = \sum_{k=1}^{N_r} \delta_k^r(s) \frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} \\
\frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} &= \frac{\partial \left(\sum_{m=0}^{N_{r-1}} w_{km}^r y_m^{r-1}(s) \right)}{\partial v_j^{r-1}(s)} = \frac{\partial \left(\sum_{m=0}^{N_{r-1}} w_{km}^r f(v_m^{r-1}(s)) \right)}{\partial v_j^{r-1}(s)} \\
\frac{\partial v_k^r(s)}{\partial v_j^{r-1}(s)} &= w_{kj}^r \frac{\partial f(v_j^{r-1}(s))}{\partial v_j^{r-1}(s)} = w_{kj}^r f'(v_j^{r-1}(s)) \\
\delta_j^{r-1}(s) &= \left(\sum_{k=1}^{N_r} \delta_k^r(s) w_{kj}^r \right) f'(v_j^{r-1}(s))
\end{aligned}
$$

SGD optimizer

In summary,

$$
\begin{aligned}
\boldsymbol{w}_j^r(i+1) &= \boldsymbol{w}_j^r(i) + \Delta \boldsymbol{w}_j^r, \\
\Delta \boldsymbol{w}_j^r &= -\mu \sum_{s \in \mathcal{Z}_{tr}} \delta_j^r(s) \boldsymbol{y}^{r-1}(s) \\
\delta_j^r(s) &= \begin{cases} (f(v_j^r(s)) - y_j^r) f'(v_j^r(s)) & r = L \\ \left(\sum_{k=1}^{N_{r+1}} \delta_k^{r+1}(s) w_{kj}^{r+1} \right) f'(v_j^r(s)) & r < L \end{cases}
\end{aligned}
$$

For the logistic function,

$$
f'(v_j^r(s)) = af(v_j^r(s))(1 - f(v_j^r(s)))
$$

and for ReLU,

$$
f'(v_j^r(s)) = \begin{cases} 1 & v_j^r(s) > 0, \\ 0 & \text{otherwise.} \end{cases}
$$

Start from $(\mathbf{x}(s), \boldsymbol{y}(s))$, $s \in \mathcal{Z}_{tr}$, a given network architecture with random weight initialization, learning rate μ, maximum number $T > 0$ of iterations (epochs), and minimum error $\epsilon > 0$.

01. Set $i \leftarrow 1$.
02. Do
03.   Set $\mathcal{E} \leftarrow 0$.
04.   For each $s \in \mathcal{Z}_{tr}$ do
05.   For $r = 1$ to L and $j = 1$ to N_r do
06.   Compute $v_j^r(s)$ and $y_j^r(s) = f(v_j^r(s))$.
07.   For $j = 1$ to N_L do
08.   Set $\mathcal{E} \leftarrow \mathcal{E} + \frac{1}{2}(y_j^L(s) - y_j(s))^2$
09.   For $r = 1$ to L and $j = 1$ to N_r do
10.   Set $\Delta \boldsymbol{w}_j^r \leftarrow \mathbf{0}$.

Backpropagation algorithm

11. For each $s \in \mathcal{Z}_{tr}$ do
12. For $r = L$ to 1 and $j = 1$ to N_r do
13. Compute $\delta_j^r(s)$ and $\Delta \boldsymbol{w}_j^r \leftarrow \Delta \boldsymbol{w}_j^r - \mu \delta_j^r(s) \boldsymbol{y}^{r-1}(s)$.
14. For $r = 1$ to L and $j = 1$ to N_r do
15. Set $\boldsymbol{w}_j^r \leftarrow \boldsymbol{w}_j^r + \Delta \boldsymbol{w}_j^r$.
16. Set $i \leftarrow i + 1$.
17. While $\mathcal{E} > \epsilon$ and $i \leq T$.

Although it can be optimized, lines 4-8 represent a forward pass, lines 9-10 set gradients to 0, and lines 11-15 represent a backward pass, in which the weights are updated.

Backpropagation algorithm

11. For each $s \in \mathcal{Z}_{tr}$ do
12. For $r = L$ to 1 and $j = 1$ to N_r do
13. Compute $\delta_j^r(s)$ and $\Delta \boldsymbol{w}_j^r \leftarrow \Delta \boldsymbol{w}_j^r - \mu \delta_j^r(s) \boldsymbol{y}^{r-1}(s)$.
14. For $r = 1$ to L and $j = 1$ to N_r do
15. Set $\boldsymbol{w}_j^r \leftarrow \boldsymbol{w}_j^r + \Delta \boldsymbol{w}_j^r$.
16. Set $i \leftarrow i + 1$.
17. While $\mathcal{E} > \epsilon$ and $i \leq T$.

Although it can be optimized, lines 4-8 represent a forward pass, lines 9-10 set gradients to 0, and lines 11-15 represent a backward pass, in which the weights are updated.

More details about SGD and other tricks to train deep neural networks will be discussed in the next lecture.