Foundations of Machine Learning in Python

Moritz Wolter

September 21, 2022

High-Performance Computing and Analytics Lab

Overview

Neural networks

Classification with neural networks

Neural networks

The wonders of the human visual system

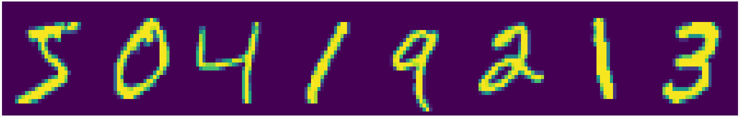


Figure: Most humans effortlessly recognize the digits 5 0 4 1 9 2 1 3.

Biological motivation

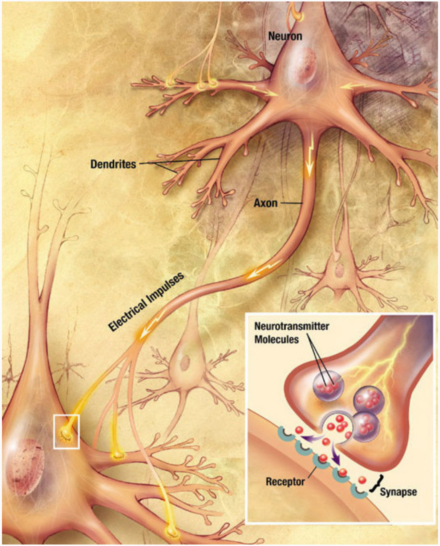


Image source: en.wikipedia.org

Biological motivation

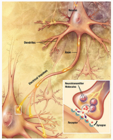
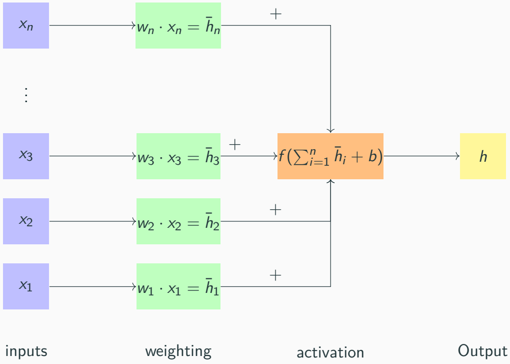


Image source: en.wikipedia.org

- A Human brain contains approximately 86 billion neurons.
- 10^{14} to 10^{15} synapses connect these neurons.
- Neurons recieve inputs from dentrites.
- and can produce outputs signals along its axon.
- Axons are connect neurons, modelled by weighting inputs wx.
- Neuron inputs can be inhibitive (negative weight) or
- excitory (positive weight).
- If enough inputs exite a neuron it fires.
- The activation function aims to mimic this behaviour.
- Even though neural networks started out as biologically motivated,
- engineering efforts have since diverged from biology.

The perceptron

Can computers recognize digits? Mimic biological neurons,



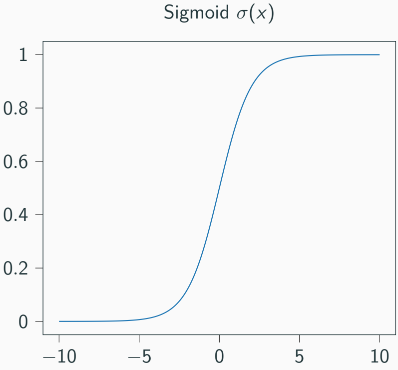
Formally a single perceptron is defined as

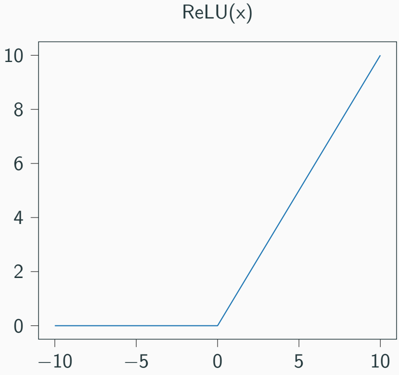
$$f(\mathbf{w}^T \mathbf{x}) = h \tag{1}$$

with $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^n$ and $h \in \mathbb{R}$.

The activation function f

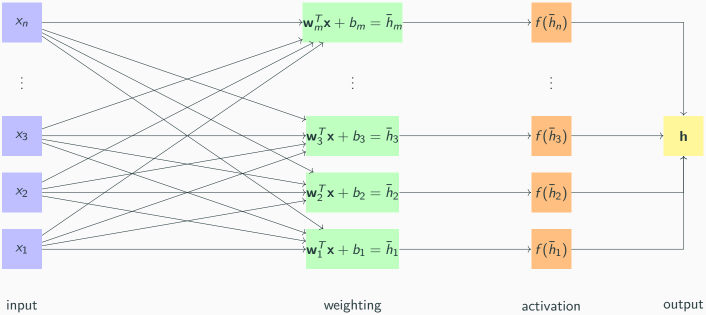
Two popular choices for the activation function f.





Arrays of perceptrons

Let's extend the definition to cover an array of perceptrons:



Every input is connected to every neuron. In matrix language, this turns into

$$\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b}, \qquad \mathbf{h} = f(\bar{\mathbf{h}}). \tag{2}$$

With $\mathbf{W} \in \mathbb{R}^{m,n}$, $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{h}, \bar{\mathbf{h}} \in \mathbb{R}^m$.

The loss function

To choose weights for the network, we require a quality measure. We already saw the mean squared error cost function,

$$C_{\text{mse}} = \frac{1}{2} \sum_{k=1}^{n} (\mathbf{y}_k - \mathbf{h}_k)^2 = \frac{1}{2} (\mathbf{y} - \mathbf{h})^T (\mathbf{y} - \mathbf{h}) \tag{3}$$

This function measures the squared distance from each desired output. $\mathbf{y}$ denotes the desired labels, and $\mathbf{h}$ represents network output.

The gradient of the mse-cost-function

Both the mean squared error loss function and our dense layer are differentiable.

$$\frac{\partial C_{\text{mse}}}{\partial \mathbf{h}} = \mathbf{h} - \mathbf{y} = \triangle_{\text{mse}} \tag{4}$$

The $\triangle$ symbol will re-appear. It always indicates incoming gradient information from above. If the labels are a vector of shape $\mathbb{R}^m$, $\triangle$ and the network output $\mathbf{h}$ must share this dimension.

The gradient of a dense layer

The chain rule tells us the gradients for the dense layer [Nie15]

$$\delta \mathbf{W} = [f'(\bar{\mathbf{h}}) \odot \triangle]\mathbf{x}^T, \qquad \delta \mathbf{b} = f'(\bar{\mathbf{h}}) \odot \triangle, \tag{5}$$

$$\delta \mathbf{x} = \mathbf{W}^T[f'(\bar{\mathbf{h}}) \odot \triangle], \tag{6}$$

where $\odot$ is the element-wise product. δ denotes the cost function gradient for the value following it [Gre+16].

Modern libraries will take care of these computations for you!

You can choose to verify these equations yourself by completing the optional deep learning project.

The gradient of a dense layer

The chain rule tells us the gradients for the dense layer [Nie15]

$$\delta\mathbf{W} = [f'(\bar{\mathbf{h}}) \odot \triangle]\mathbf{x}^T, \qquad \delta\mathbf{b} = f'(\bar{\mathbf{h}}) \odot \triangle, \tag{5}$$
$$\delta\mathbf{x} = \mathbf{W}^T[f'(\bar{\mathbf{h}}) \odot \triangle], \tag{6}$$

where $\odot$ is the element-wise product. δ denotes the cost function gradient for the value following it [Gre+16].
Modern libraries will take care of these computations for you!
You can choose to verify these equations yourself by completing the optional deep learning project.

On the board, derive: Recall the chain rule $(g(h(x)))' = g'(h(x)) \cdot h'(x)$.
For the activation function, we have,

$$\bar{\mathbf{h}} = f(\bar{\mathbf{h}}) \tag{7}$$
$$\Rightarrow \delta\bar{\mathbf{h}} = f'(\bar{\mathbf{h}}) \odot \triangle \tag{8}$$

For the weight matrix,

$$\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{9}$$
$$\Rightarrow \delta\mathbf{W} = \delta\bar{\mathbf{h}}\mathbf{x}^T = [f'(\bar{\mathbf{h}}) \odot \triangle]^T\mathbf{x} \tag{10}$$

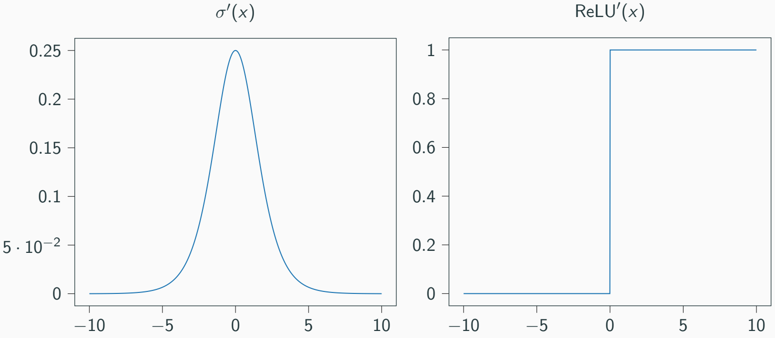
For the bias,

$$\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{11}$$
$$\Rightarrow \delta\mathbf{b} = 1 \odot \delta\bar{\mathbf{h}} = [f'(\bar{\mathbf{h}}) \odot \triangle] \tag{12}$$

Derivatives of our activation functions

$$\sigma'(x) = \sigma \cdot (1 - \sigma(x)) \tag{13}$$

$$\text{ReLU}' = H(x) \tag{14}$$



Perceptrons for functions

The network components described this far already allow function learning. Given a noisy input signal $x \in \mathbb{R}^m$ and a ground through output $y \in \mathbb{R}^m$, define,

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{15}$$

$$\mathbf{y}_{\text{net}} = \mathbf{W}_y \mathbf{h} \tag{16}$$

With $\mathbf{W} \in \mathbb{R}^{m,n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$. m and n denote the number of neurons and the input signal length. For signal denoising, input and output have the same length. Therefore $\mathbf{W}_y \in \mathbb{R}^{n,m}$.

Denoising a cosine

Training works by iteratively descending along the gradients. For $\mathbf{W}$ the weights at the next time step τ are given by,

$$\mathbf{W}_{\tau+1} = \mathbf{W}_{\tau} + \epsilon \cdot \delta \mathbf{W}_{\tau}. \tag{17}$$

The step size is given by ϵ. At $\tau = 0$ matrix entries are random. $\mathcal{U}[-0.1, 0.1]$ is a reasonable choice here. The process is the same for all other network components.

Denoising a cosine

Optimization for 500 steps leads to the output below:

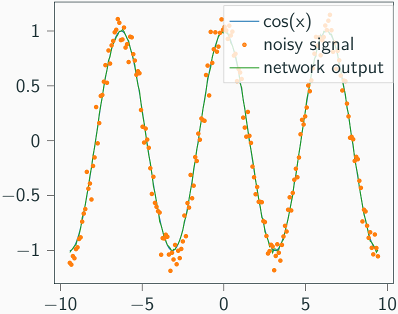


Figure: The cosine function is shown in blue. A noisy network input in orange, and a denoised network output in green.

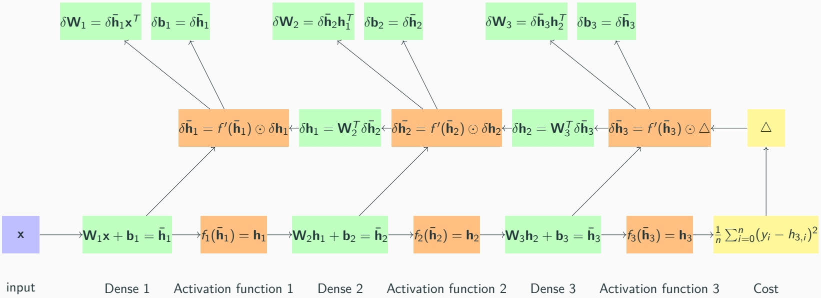
Classification with neural networks

Deep multi-layer networks

Stack dense layers and activations to create deep networks.

input		Dense 1		Activation function 1		Dense 2		Activation function 2		Dense 3		Activation function 3		Cost
$\mathbf{x}$	$\rightarrow$	$\mathbf{W}_1\mathbf{x} + \mathbf{b}_1 = \bar{\mathbf{h}}_1$	$\rightarrow$	$f_1(\bar{\mathbf{h}}_1) = \mathbf{h}_1$	$\rightarrow$	$\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 = \bar{\mathbf{h}}_2$	$\rightarrow$	$f_2(\bar{\mathbf{h}}_2) = \mathbf{h}_2$	$\rightarrow$	$\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 = \bar{\mathbf{h}}_3$	$\rightarrow$	$f_3(\bar{\mathbf{h}}_3) = \mathbf{h}_3$	$\rightarrow$	$\frac{1}{n}\sum(\mathbf{y} - \mathbf{h}_3)^2$

Backpropagation



The cross-entropy loss

The cross entropy loss function is defined as [Nie15; Bis06]

$$C_{\text{ce}}(\mathbf{y}, \mathbf{o}) = -\sum_{k}^{n_o} (\mathbf{y}_k \ln \mathbf{o}_k) + (\mathbf{1} - \mathbf{y}_k) \ln(\mathbf{1} - \mathbf{o}_k). \tag{18}$$

If a sigmoidal activation function produced **o** the gradients can be computed using [Nie15; Bis06]

$$\frac{\partial C_{ce}}{\partial \mathbf{h}} = \sigma(\mathbf{o}) - \mathbf{y} = \triangle_{\text{ce}} \tag{19}$$

The cross-entropy loss

The cross entropy loss function is defined as [Nie15; Bis06]

$$C_{\text{ce}}(\mathbf{y}, \mathbf{o}) = -\sum_{k}^{n_o} (\mathbf{y}_k \ln \mathbf{o}_k) + (\mathbf{1} - \mathbf{y}_k) \ln(\mathbf{1} - \mathbf{o}_k). \quad (18)$$

If a sigmoidal activation function produced $\mathbf{o}$ the gradients can be computed using [Nie15; Bis06]

$$\frac{\partial C_{ce}}{\partial \mathbf{h}} = \sigma(\mathbf{o}) - \mathbf{y} = \triangle_{\text{ce}} \quad (19)$$

TODO: find the proof.

The MNIST-Dataset

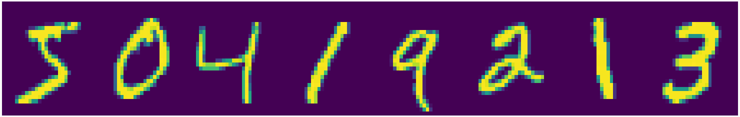


Figure: The MNIST-dataset contains 70k images of handwritten digits.

Validation and Test data splits

- To ensure the correct operation of the systems we devise, it is paramount to hold back part of the data for validation and testing.
- Before starting to train, split off validation and test data.
- The 70k MNIST samples could, for example, be partitioned into 59k training images. 1k validation images and 10k test images.

Input-preprocessing

Standard initializations and learning algorithms assume an approximately standard normal distribution of the network inputs. Consequently we must rescale the data using,

$$\mathbf{x}_s = \frac{\mathbf{x} - \mu}{\sigma} \tag{20}$$

With μ and σ the training set mean and standard deviation.

Label-encoding

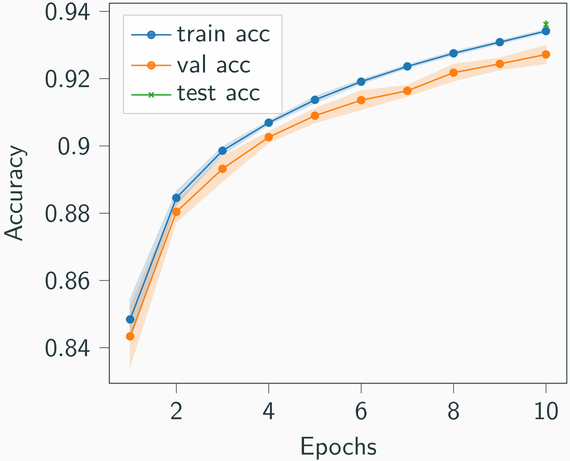
It has proven useful to have individual output neurons produce probabilities for each class. Given integer labels $1, 2, 3, 4, \dots \in \mathbb{Z}$. One-hot encoded label vectors have a one at the labels positions and zeros elsewhere. I.e.

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \vdots \end{pmatrix}, \dots \tag{21}$$

for the integer label sequence above.

MNIST-Classification

Training a three-layer dense network on mnist for five runs leads to:



Conclusion

- Artificial neural networks enable computers to make sense of images.
- The optimization result depends on the initialization.
- The initialization depends on the pseudo-randomness-seed.
- Seed-values must be recorded.

References

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Klaus Greff, Rupesh K Srivastava, Jan Koutnik, Bas R Steunebrink, and Jürgen Schmidhuber. "LSTM: A search space odyssey." In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.