

# Foundations of Machine Learning in Python

---

Moritz Wolter

August 31, 2022

High-Performance Computing and Analytics Lab

Neural networks

Classification with neural networks

# Neural networks

---

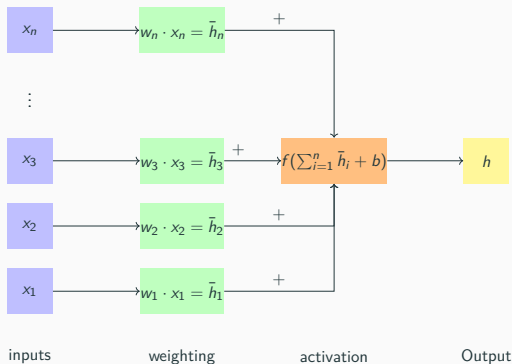
# The wonders of the human visual system



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

# The perceptron

Can computers recognize digits? Mimic biological neurons,



Formally a single perceptron is defined as

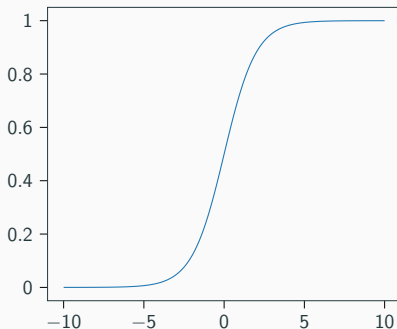
$$f(\mathbf{w}^T \mathbf{x}) = h \quad (1)$$

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

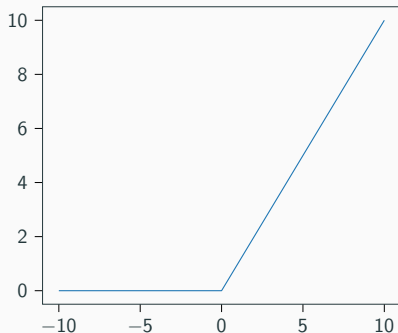
# The activation function $f$

Two popular choices for the activation function  $f$ .

Sigmoid  $\sigma(x)$

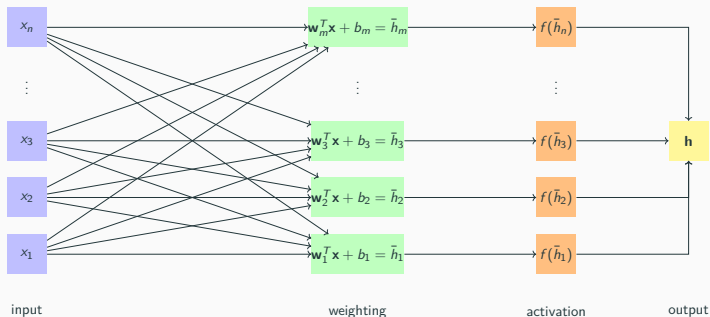


ReLU( $x$ )



# 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

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

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,

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

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.

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

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]

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

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

where  $\odot$  is the element-wise product.  $\delta$  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.

## Foundations of Machine Learning in Python

## └ Neural networks

## └ The gradient of a dense layer

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

$$\partial \mathbf{W} = [f'(\bar{\mathbf{h}}) \odot \Delta] \mathbf{x}^T, \quad \partial \mathbf{b} = f'(\bar{\mathbf{h}}) \odot \Delta, \quad (5)$$

$$\partial \mathbf{x} = \mathbf{W}^T [f'(\bar{\mathbf{h}}) \odot \Delta], \quad (6)$$

where  $\odot$  is the element-wise product.  $\delta$  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,

$$\bar{\mathbf{h}} = f(\bar{\mathbf{h}}) \quad (7)$$

$$\Rightarrow \delta \bar{\mathbf{h}} = f'(\bar{\mathbf{h}}) \odot \Delta \quad (8)$$

For the weight matrix,

$$\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (9)$$

$$\Rightarrow \delta \mathbf{W} = \delta \bar{\mathbf{h}} \mathbf{x}^T = [f'(\bar{\mathbf{h}}) \odot \Delta]^T \mathbf{x} \quad (10)$$

For the bias,

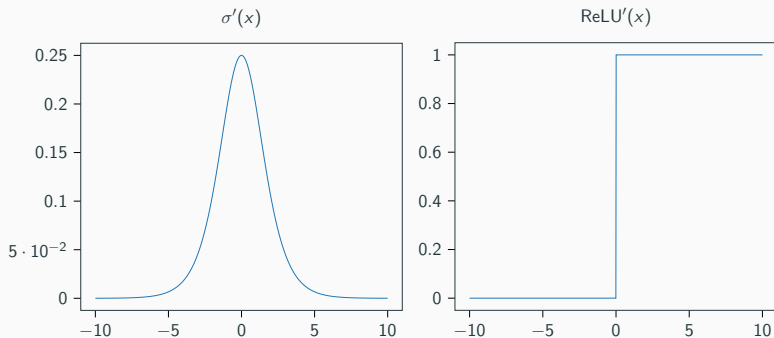
$$\bar{\mathbf{h}} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (11)$$

$$\Rightarrow \delta \mathbf{b} = 1 \odot \delta \bar{\mathbf{h}} = [f'(\bar{\mathbf{h}}) \odot \Delta] \quad (12)$$

## Derivatives of our activation functions

$$\sigma'(x) = \sigma \cdot (1 - \sigma(x)) \quad (13)$$

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



## Perceptrons for functions

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

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

$$\mathbf{y}_{\text{net}} = \mathbf{W}_y \mathbf{h} \quad (16)$$

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

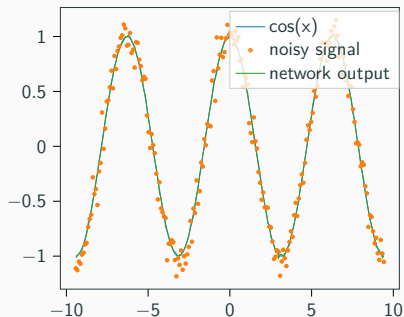
Training works by iteratively descending along the gradients. For  $\mathbf{W}$  the weights at the next time step  $\tau$  are given by,

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

The step size is given by  $\epsilon$ . 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.

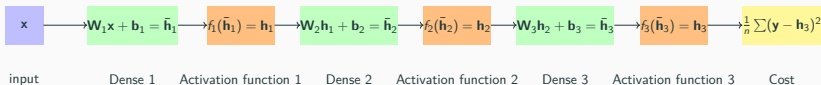
# Classification with neural networks

---

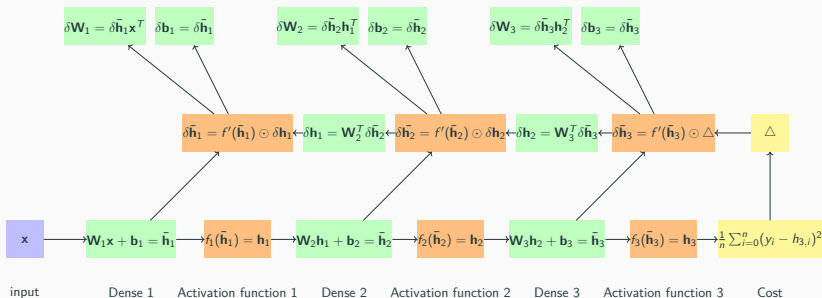


# Deep multi-layer networks

Stack dense layers and activations to create deep networks.



# Backpropagation



# The cross-entropy loss

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

$$C_{ce}(\mathbf{t}, \mathbf{o}) = - \sum_k^{n_o} (\mathbf{t}_k \ln \mathbf{o}_k) + (\mathbf{1} - \mathbf{t}_k) \ln(\mathbf{1} - \mathbf{o}_k). \quad (18)$$

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

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

# Foundations of Machine Learning in Python

## └ Classification with neural networks

### └ The cross-entropy loss

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

$$C_{ce}(\mathbf{t}, \mathbf{a}) = - \sum_k \left( t_k \ln a_k + (1 - t_k) \ln(1 - a_k) \right). \quad (18)$$

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

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

TODO: find the proof.

# The MNIST-Dataset



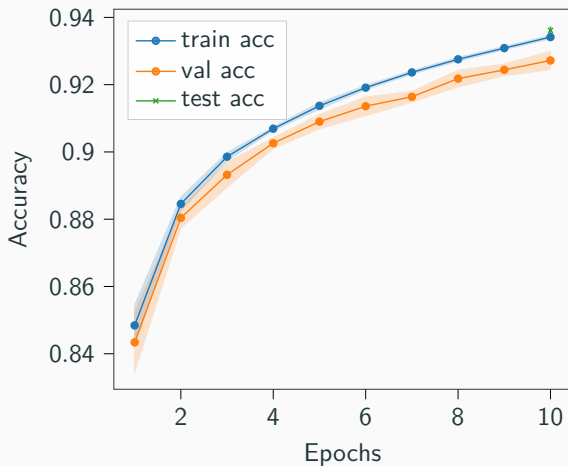
**Figure:** The MNIST-dataset contains 70k images of handwritten digits.

## 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.

# 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

---

- [Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.
- [Gre+16] 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.
- [Nie15] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.