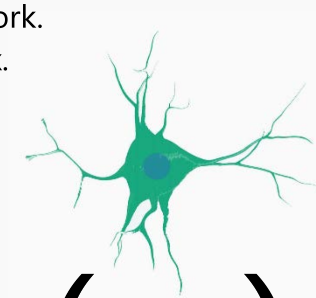
module5 · optimization-based methods

neural networks

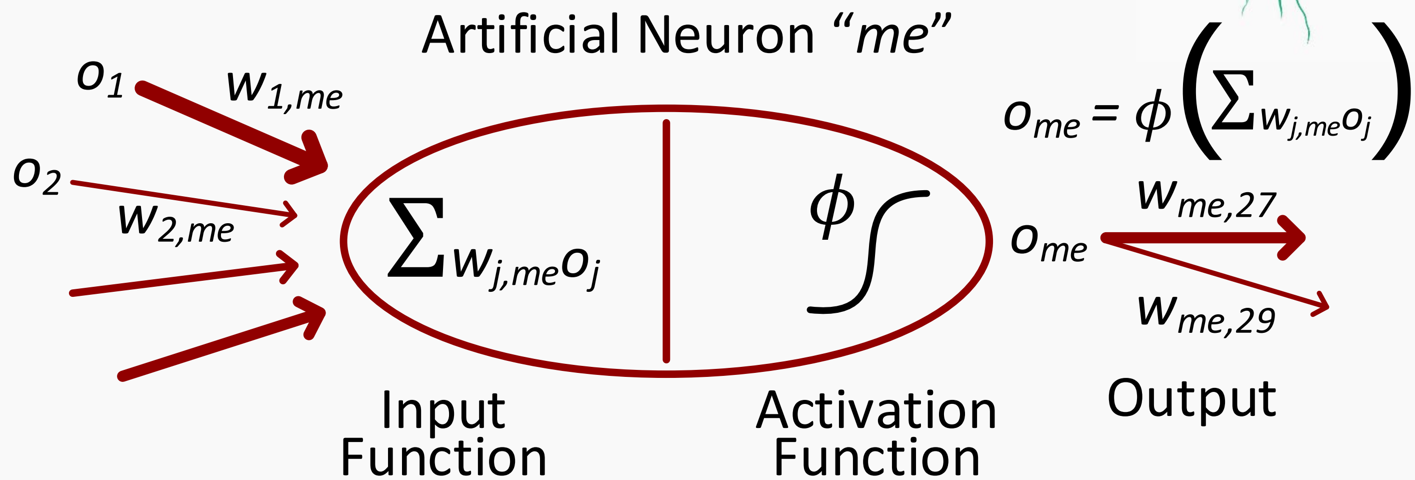
Neural Networks are particularly powerful for computer vision problems. **Artificial Neural Networks** are not the same as **real Neural Networks**:

- There are $\mathbf{10^{11}}$ neurons in a human brain with $\mathbf{10^{14}}$ synapses (connections).
- **Signals** are the electrical potential spikes that travel through the network.
- Thus, the order of magnitude is much larger in a **real Neural Network**.
- **Artificial Neural Networks**, in turn, compute like calculators.



McCulloh-Pitts "Neuron"

Illustrating an **Artificial Neuron** named "***me***":

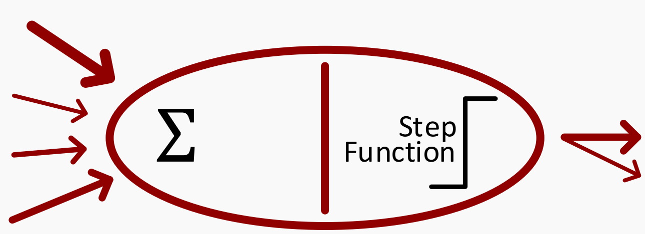


All other outputs $\boldsymbol{o_j}$ of the predecessor neurons are fed into "***me***" and **weighted** by their **connectivity**. The thickness of the predecessor outputs is representative of the magnitude (**weight**) of connectivity. The neuron "***me***" consequentially takes the sum of all the outputs from previous neurons as weighted by their relative connectivity: $\sum_j \boldsymbol{w_{j,me} o_j}$.

The summation feeds into an **Activation Function** which is typically between 0 and 1; appearing as a **threshold function** $\boldsymbol{\phi}\left(\sum_j \boldsymbol{w_{j,me} o_j}\right)$ → the value of the activation function = the **output** of the neuron $\boldsymbol{o_{me}}$. Once computed, the output of neuron "***me***" feeds into all the successive neurons afterwards, also weighted by the connectivity strength.

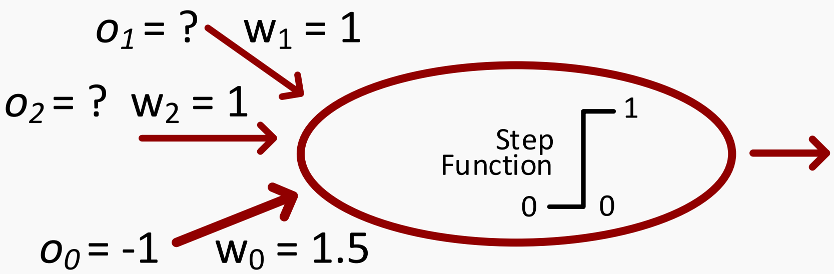
Assuming that the activation function $\boldsymbol{\phi}\left(\sum_j \boldsymbol{w_{j,me} o_j}\right)$ is simply a **step function** for the threshold:

Therefore, enough electrical input from prior neurons will make the sum $\sum_j \boldsymbol{w_{j,me} o_j}$ large;



Ultimately firing when the **activation function** is triggered and returning the higher value (**1**) as opposed to the lower value (**0**).

Applying Weights to Compute an Artificial Neuron's Output



o_1	o_2	output
0	0	0
1	0	0
0	1	0
1	1	1

$$o_1 = 0, o_2 = 0 \rightarrow \phi\left(\sum_j w_{j,me} o_j\right) \rightarrow (0 * 1) + (0 * 1) + (-1 * 1.5) = -1.5 < 0 \rightarrow \cancel{\text{activate}} \rightarrow o_{me} = 0$$

$$o_1 = 1, o_2 = 0 \rightarrow \phi\left(\sum_j w_{j,me} o_j\right) \rightarrow (1 * 0) + (0 * 1) + (-1 * 1.5) = -0.5 < 0 \rightarrow \cancel{\text{activate}} \rightarrow o_{me} = 0$$

$$o_1 = 0, o_2 = 1 \rightarrow \phi\left(\sum_j w_{j,me} o_j\right) \rightarrow (0 * 1) + (1 * 1) + (-1 * 1.5) = -0.5 < 0 \rightarrow \cancel{\text{activate}} \rightarrow o_{me} = 0$$

$$o_1 = 1, o_2 = 1 \rightarrow \phi\left(\sum_j w_{j,me} o_j\right) \rightarrow (1 * 1) + (1 * 1) + (-1 * 1.5) = 0.5 > 0 \rightarrow \text{activate} \rightarrow o_{me} = 1$$

The above illustration outputs the logic **where if one of the outputs from prior neurons = 0, then the output of the "*me*" neuron =0; if both outputs = 1, then the output of "*me*" = 1**.

In other words, the above Artificial Neuron computes the **logical table AND function**.

Additionally, Artificial Neural Networks can equally compute the logical table **OR** and **NOT** functions.

backpropagation

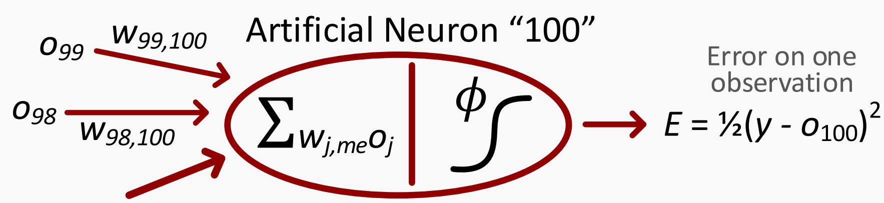
The **Activation Function** used in **Artificial Neural Networks** introduced above represents a threshold; the function is applied in a "smooth" nature due to being differentiable:

$$\phi\left(\sum_j w_{jme}\alpha_j\right) = \frac{\mathbf{1}}{(\mathbf{1}+\boldsymbol{e}^{-\boldsymbol{x}})} \rightarrow \textbf{Sigmoid Function used in Logistic Regression}$$

Single Layer Neural Network

An **Artificial Neural Network** learns by examining a particular input in **supervised learning** and comparing the predicted output; the Neuron's **error** on one observation: $E = \frac{1}{2}(y - o_{100})^2$

In a human brain, the synapses strengthen and weaken in order to learn; the same applies above. Therefore, the **error** will be minimized with respect to the weighted outputs from prior neurons.



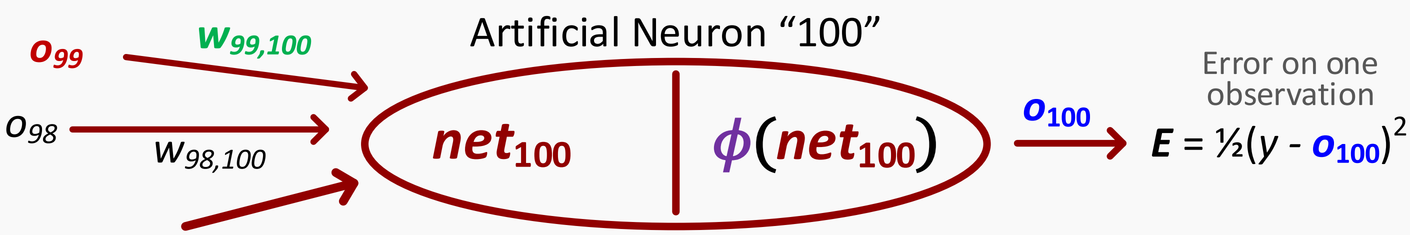
Backpropagation Algorithm

Minimization of the Error in an Artificial Neuron is the objective of **Backpropagation**.

- An algorithm that trains the weights of a neural network
- Require information to propagate backwards through the network, then forwards, then backwards, etc.; reiterating as such
- Backwards Propagation = the chain rule from calculus

Backpropagation through a Single Layer Neural Network

$\boldsymbol{net}_{100}$ is simply the weighted sum from prior neurons weighted by their respective connectivity. The objective is to adjust the weight $\boldsymbol{w}_{99,100}$ in order to minimize training error on the Neuron. The derivative of the error will be taken with respect to the weight $\boldsymbol{w}_{99,100}$ in order to take steps along the gradient and reduce the error:



Below represents the derivative of error $\boldsymbol{E}$ using the **chain rule** from calculus:

$$\frac{dE}{d\boldsymbol{w}_{99,100}} = \frac{dE}{d\boldsymbol{o}_{100}}\frac{d\boldsymbol{o}_{100}}{d\boldsymbol{net}_{100}}\frac{d\boldsymbol{net}_{100}}{d\boldsymbol{w}_{99,100}}$$

$$\boldsymbol{\phi}(z) = \frac{1}{1+e^z}$$

$$\boldsymbol{\phi}'^{(z)} = \frac{d\boldsymbol{\phi}(z)}{dz} = \boldsymbol{\phi}(z)\big(1-\boldsymbol{\phi}(z)\big)$$

The above represents the derivative of $\boldsymbol{E}$ with respect to the output $\boldsymbol{o_{100}}$, the derivative of the output $\boldsymbol{o_{100}}$ with respect to the net $\boldsymbol{net_{100}}$, and the derivative of the net $\boldsymbol{net_{100}}$ with respect to the weight $\boldsymbol{w_{99,100}}$.

In order to calculate at least one of the terms from the chain rule, a factor must be applied as the activation $\boldsymbol{\phi}$. The derivative of phi $\boldsymbol{\phi}$ shown above is phi $\boldsymbol{\phi(z)}$ itself times 1 minus phi $\left(\mathbf{1} - \boldsymbol{\phi(z)}\right)$.

Stepping through the Derivative of the Error with Respect to the Weight

$$\frac{dE}{d\boldsymbol{o_{100}}} = \frac{1}{2}2(y - \boldsymbol{o_{100}})(-1) = -(y - \boldsymbol{o_{100}})$$

$\boldsymbol{E}$ is a function of $\boldsymbol{o_{100}}$ and thus the derivative of $\boldsymbol{E}$ with respect to $\boldsymbol{o_{100}}$ is applied appropriately.

$$\frac{d\boldsymbol{o_{100}}}{d\boldsymbol{net_{100}}} = \frac{d\boldsymbol{\phi}(\boldsymbol{net_{100}})}{d\boldsymbol{net_{100}}} = \boldsymbol{\phi}'(\boldsymbol{net_{100}})\left(1 - \boldsymbol{\phi}(\boldsymbol{net_{100}})\right) = \boldsymbol{o_{100}}(1 - \boldsymbol{o_{100}})$$

The derivative of $\boldsymbol{o_{100}}$ with respect to $\boldsymbol{net_{100}}$is related through phi $\boldsymbol{\phi}$; thus requiring the derivative of phi $\boldsymbol{\phi}$ and $\boldsymbol{net_{100}}$represented through $\boldsymbol{o_{100}}$, appropriately, as illustrated above.

$$\frac{d\boldsymbol{net_{100}}}{d\boldsymbol{w_{99,100}}} = \frac{d\left(\boldsymbol{w_{99,100}}\boldsymbol{o_{99}} + w_{98,100}o_{98} + w_{97,100}o_{97} + \cdots\right)}{d\boldsymbol{w_{99,100}}} = \boldsymbol{o_{99}}$$

$\boldsymbol{net_{100}}$ is simply the sum of the weighted outputs from prior Neurons, with only one containing the weight $\boldsymbol{w_{99,100}}$. Thus, the numerator represents the weighted sum of outputs with one occurrence of $\boldsymbol{w_{99,100}}$. The derivative of $\boldsymbol{net_{100}}$ with respect to $\boldsymbol{w_{99,100}}$ is simply the output of neuron "99" $\boldsymbol{o_{99}}$.

The result of the calculation above is as follows:

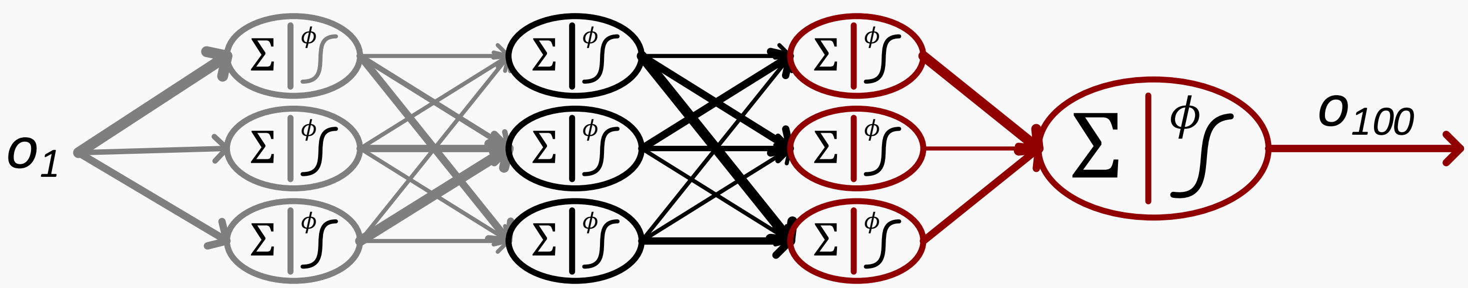
$$\frac{dE}{d\boldsymbol{w_{99,100}}} = \frac{dE}{d\boldsymbol{o_{100}}}\frac{d\boldsymbol{o_{100}}}{d\boldsymbol{net_{100}}}\frac{d\boldsymbol{net_{100}}}{d\boldsymbol{w_{99,100}}} = -(y - \boldsymbol{o_{100}})\boldsymbol{o_{100}}(1 - \boldsymbol{o_{100}})\boldsymbol{o_{99}}$$

Adjusting the notation for application for the term that only depends on **node "100"**:

$$\frac{dE}{d\boldsymbol{o_{100}}}\frac{d\boldsymbol{o_{100}}}{d\boldsymbol{net_{100}}} = \boldsymbol{\delta_{100}} \to \frac{dE}{d\boldsymbol{w_{99,100}}} = \boldsymbol{\delta_{100}}\boldsymbol{o_{99}}$$

The above illustration represents the calculations for the final layer of an **Artificial Neural Network**:

Feedforward Neural Network



Each layer is composed of series of Neurons that contribute weighted outputs to the final layer.