Multilayer Perceptron

Building a **network** of **Perceptrons**

Faculty of Mathematics and Computer Science, University of Bucharest
and
Sparktech Software

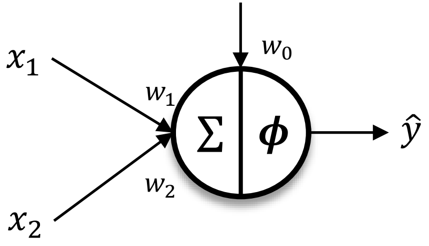
Academic Year 2018/2019, 1st Semester

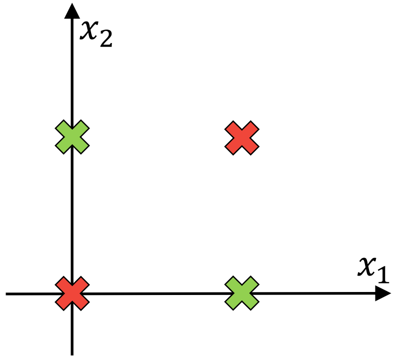
XOR with Perceptrons

- A single Perceptron cannot learn the XOR function because it is not linearly separable.

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



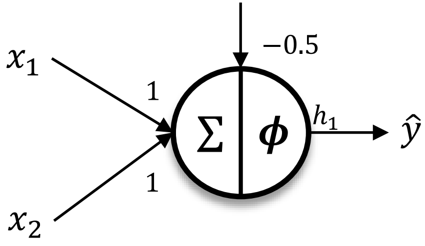


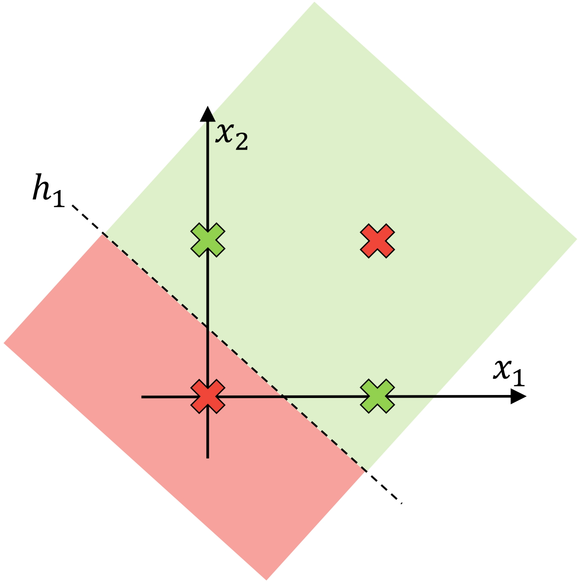
XOR with Perceptrons

- A single Perceptron cannot learn the XOR function because it is not linearly separable.

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



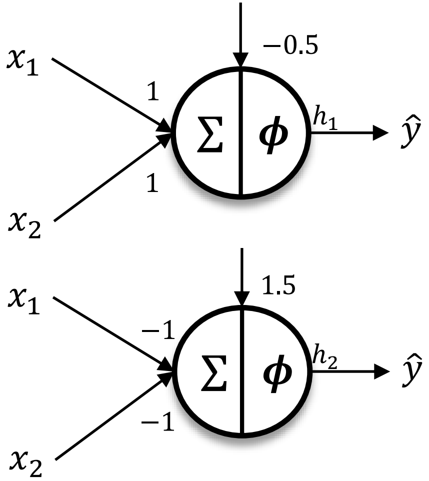


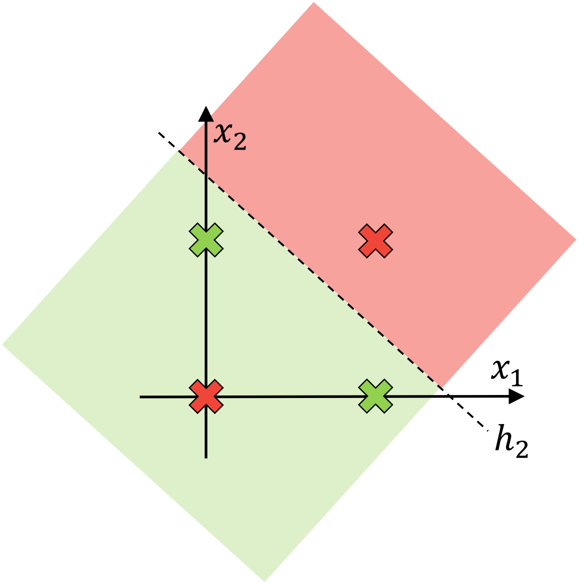
XOR with Perceptrons

- A single Perceptron cannot learn the XOR function because it is not linearly separable.

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



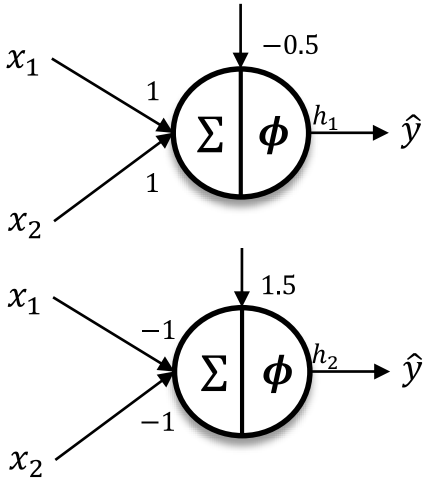


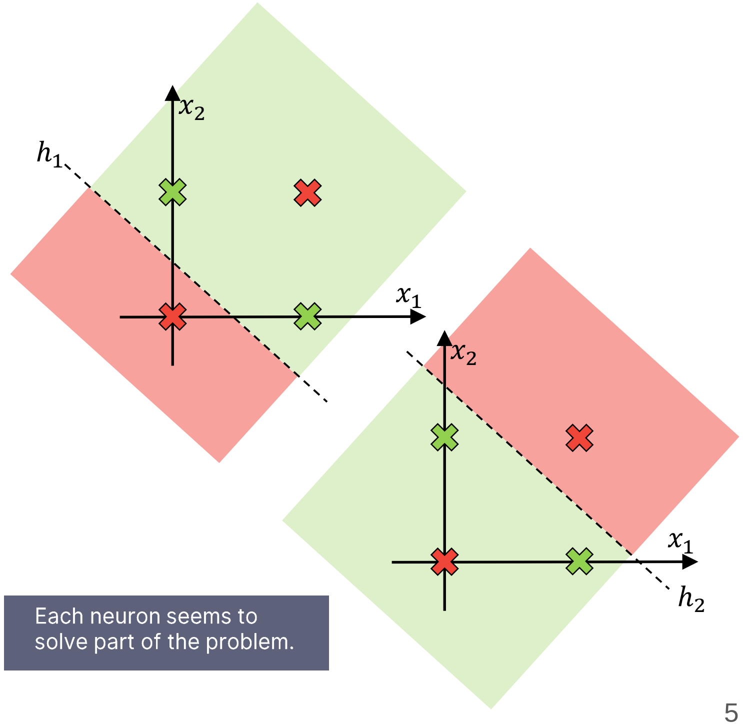
XOR with Perceptrons

- A single Perceptron cannot learn the XOR function because it is not linearly separable.

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





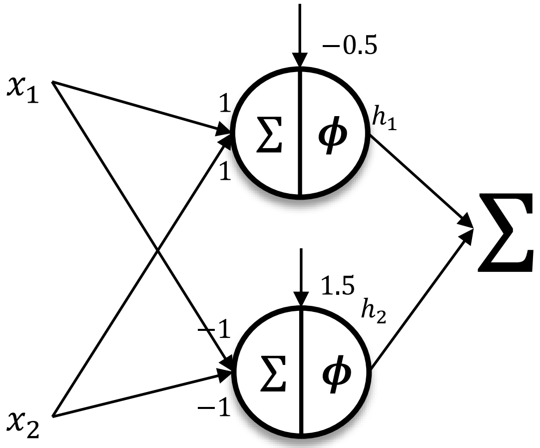
Each neuron seems to solve part of the problem.

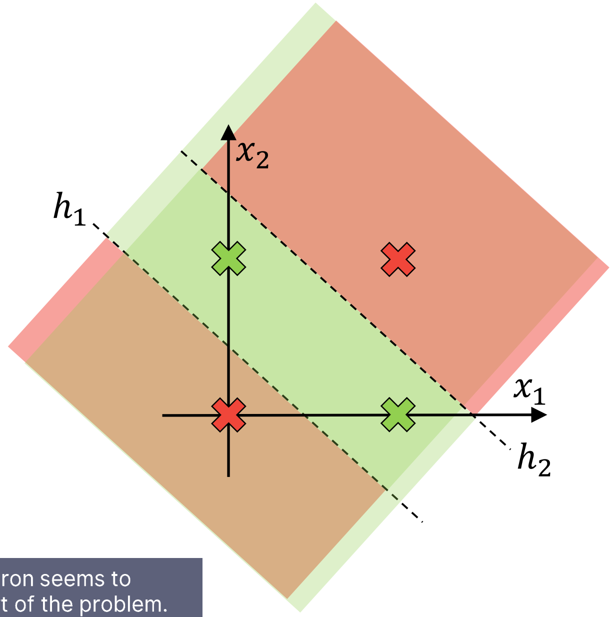
XOR with Perceptrons

- A single Perceptron cannot learn the XOR function because it is not linearly separable.
- This limitation can be overcome by "combining" the outputs of multiple Perceptrons.

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





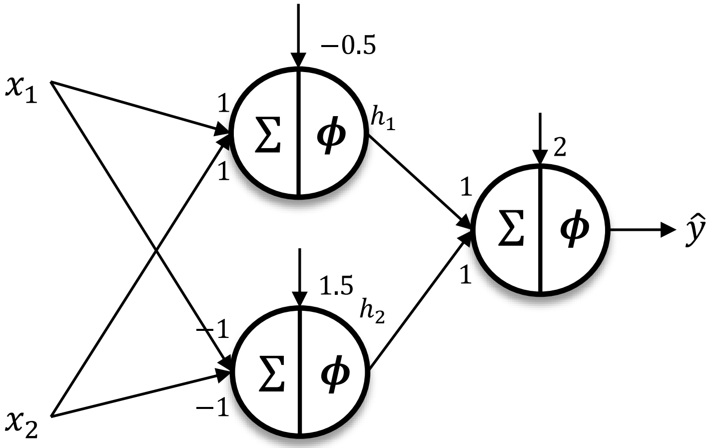
Each neuron seems to solve part of the problem.

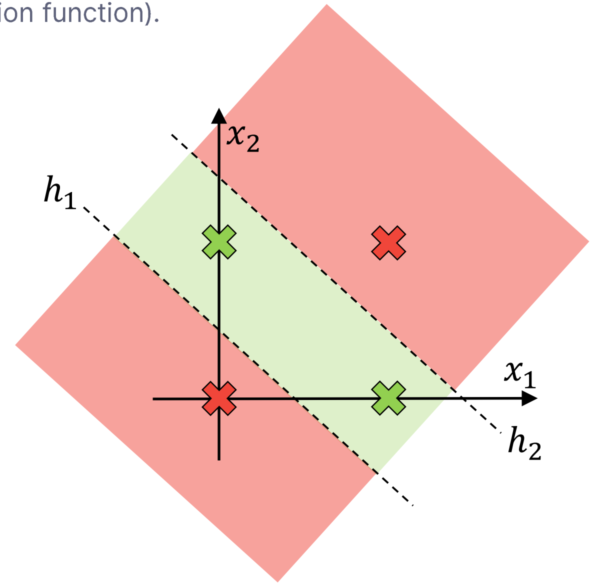
XOR with Perceptrons

- A single Perceptron cannot learn the XOR function because it is not linearly separable.
- This limitation can be overcome by "combining" the outputs of multiple Perceptrons.
    - We can combine them by using another Perceptron (weighed sum + activation function).

XOR

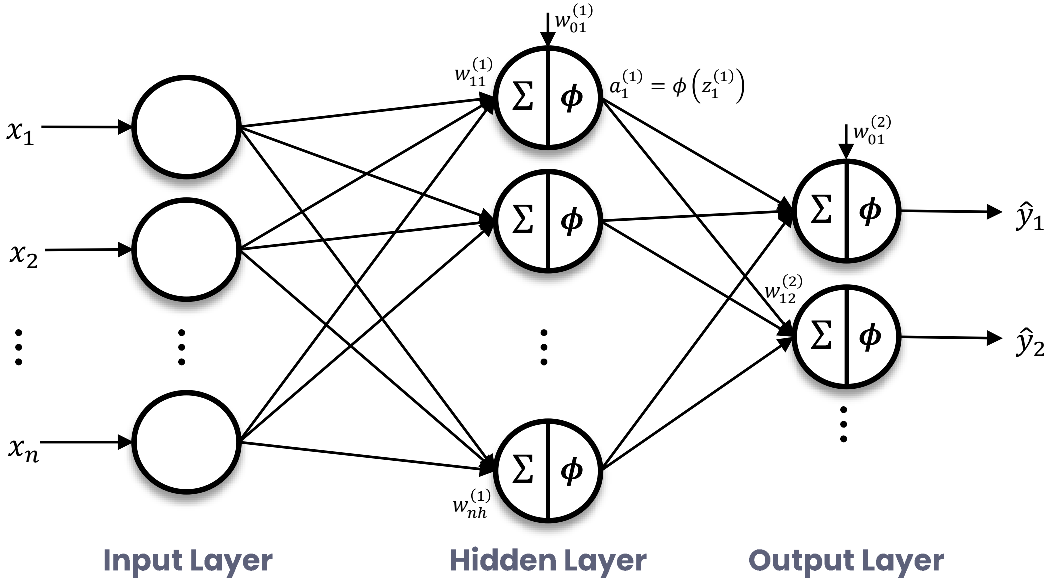
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0





The Multilayer Perceptron

- A **Multilayer Perceptron (MLP)** is a *feedforward artificial neural network* which has an *input layer*, one or more *hidden layers* and an *output layer*.
  - All neurons, excepts those from the input layer, apply an activation function to the weighted sum of inputs.
  - Each pair of neurons from consecutive layers has an associated weight.



- $w_{ij}^{(l)}$ is the weight from neuron i on layer $l-1$ to neuron j on layer l (input is layer 0).
- $w_{0j}^{(l)}$ is the bias of neuron j on layer l.
- $a_i^{(l)}$ is the output of neuron i on layer l.
- $z_i^{(l)}$ is the output before activation.

MLP in matrix format

$$\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad W^{(1)} = \begin{bmatrix} w_{01}^{(1)} & w_{11}^{(1)} & \cdots & w_{n1}^{(1)} \\ w_{02}^{(1)} & w_{12}^{(1)} & \cdots & w_{n2}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0h}^{(1)} & w_{1h}^{(1)} & \cdots & w_{nh}^{(1)} \end{bmatrix}_{h \times n+1} \qquad W^{(1)}\vec{x} = \begin{bmatrix} \sum_{i=0}^{n} x_i w_{i1}^{(1)} \\ \sum_{i=0}^{n} x_i w_{i2}^{(1)} \\ \vdots \\ \sum_{i=0}^{n} x_i w_{ih}^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ \vdots \\ z_h^{(1)} \end{bmatrix} = \vec{z}^{(1)}$$

$$\phi\left(\vec{z}^{(1)}\right) = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_h^{(1)} \end{bmatrix} \qquad \vec{a}^{(1)} = \begin{bmatrix} 1 \\ a_1^{(1)} \\ \vdots \\ a_h^{(1)} \end{bmatrix} \qquad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_o \end{bmatrix} = \phi\left(W^{(2)}\phi\left(W^{(1)}x\right)\right)$$

$$\boxed{\vec{x}} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)}$$

Training an MLP

- A multilayer perceptron is trained with **stochastic gradient descent**.
    - "*Stochastic*" because the gradient is computed only with respect to a single training example or a batch, not the entire dataset.
    - We need to compute the gradient of the error function with respect to each weight of the network and update the weights correspondingly.

$$E(\vec{x}) = \mathcal{L}(y, \hat{y})$$

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial E}{\partial w_{ij}^{(l)}}$$

$\mathcal{L}$ is the loss (a function which should be small if $\hat{y}$ is close to y and larger otherwise).

    - Or in matrix format:

$$\Delta W^{(l)} = -\eta \frac{\partial E}{\partial W^{(l)}}$$

The Chain Rule

$$\boxed{\vec{x}} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

- The error is a function which depends on all the weights of the network.

$$E(\vec{x}) = \mathcal{L}\left(y, \phi\left(W^{(2)}\phi\left(W^{(1)}\vec{x}\right)\right)\right)$$

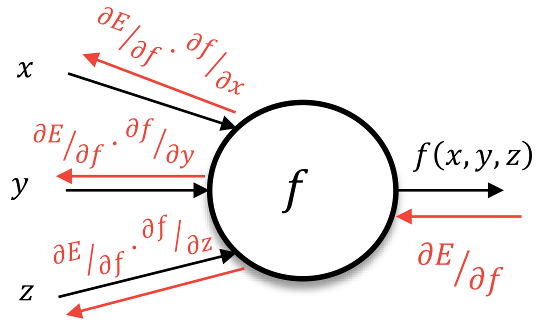
- We could pick any weight $w_{ij}^{(l)}$ and use the chain rule to compute the formula for $\frac{\partial E}{\partial w_{ij}^{(l)}}$.
- In matrix format:

$$\frac{\partial E}{\partial W^{(1)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial \vec{a}^{(1)}} \frac{\partial \vec{a}^{(1)}}{\partial \vec{z}^{(1)}} \frac{\partial \vec{z}^{(1)}}{\partial W^{(1)}}$$

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial W^{(2)}}$$

Backpropagation of error

- Computing the derivative *formula* (symbolic differentiation) for every weight in the network is both inefficient and very complex for larger networks.
- We are only interested in the numerical evaluation of the derivatives, we can focus on one gate at a time and we can used previously computed values.



$\partial E/\partial f$ is already computed numerically.

- This method is known as "**backpropagation**".
  - **„Learning representations by back-propagating errors"**

David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, 1986

Backpropagation of error

$$\boxed{\vec{x}} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

$$\frac{\partial E}{\partial W^{(2)}} =$$

$$\frac{\partial E}{\partial W^{(1)}} =$$

Backpropagation of error

$$\vec{x} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

$$\boxed{{}^{\partial E}/_{\partial \hat{y}}}$$

$$\frac{\partial E}{\partial W^{(2)}} =$$

$$\frac{\partial \boldsymbol{E}}{\partial \hat{\boldsymbol{y}}} = \boldsymbol{\mathcal{L}}'(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

$$\frac{\partial E}{\partial W^{(1)}} =$$

Backpropagation of error

$$
\begin{tikzcd}
\boxed{\vec{x}} \arrow[r] & \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \arrow[r] & \boxed{\vec{a}^{(1)} = [1; \phi(\vec{z}^{(1)})]} \arrow[r] & \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \arrow[r] & \boxed{\hat{y} = \vec{a}^{(2)} = \phi(\vec{z}^{(2)})} \arrow[r] & \boxed{E = \mathcal{L}(y, \hat{y})} \\
& & & & \boxed{\partial E / \partial \vec{z}^{(2)}} & \boxed{\partial E / \partial \hat{y}} \arrow[l]
\end{tikzcd}
$$

$$\frac{\partial E}{\partial W^{(2)}} =$$

$$\frac{\partial E}{\partial W^{(1)}} =$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

$$\frac{\partial \boldsymbol{E}}{\partial \vec{\boldsymbol{z}}^{(2)}} = \frac{\partial \boldsymbol{E}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \vec{\boldsymbol{z}}^{(2)}} = \frac{\partial \boldsymbol{E}}{\partial \hat{\boldsymbol{y}}} \boldsymbol{\phi}'(\vec{\boldsymbol{z}}^{(2)})$$

Backpropagation of error

$$\boxed{\vec{x}} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

$$\boxed{{}^{\partial E}/_{\partial \vec{a}^{(1)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \vec{z}^{(2)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \hat{y}}}$$

$$\frac{\partial E}{\partial W^{(2)}} =$$

$$\frac{\partial E}{\partial W^{(1)}} =$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

$$\frac{\partial E}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \phi'\left(\vec{z}^{(2)}\right)$$

$$\boldsymbol{\frac{\partial E}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} W^{(2)}}$$

Backpropagation of error

$$\vec{x} \rightarrow \vec{z}^{(1)} = W^{(1)}\vec{x} \rightarrow \vec{a}^{(1)} = [1; \phi(\vec{z}^{(1)})] \rightarrow \vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)} \rightarrow \hat{y} = \vec{a}^{(2)} = \phi(\vec{z}^{(2)}) \rightarrow E = \mathcal{L}(y, \hat{y})$$

$$\partial E / \partial \vec{a}^{(1)} \quad \partial E / \partial W^{(2)} \leftarrow \partial E / \partial \vec{z}^{(2)} \leftarrow \partial E / \partial \hat{y}$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial \boldsymbol{E}}{\partial \vec{\boldsymbol{z}}^{(2)}} \frac{\partial \vec{\boldsymbol{z}}^{(2)}}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial \boldsymbol{E}}{\partial \vec{\boldsymbol{z}}^{(2)}} \vec{\boldsymbol{a}}^{(1)}$$

$$\frac{\partial E}{\partial W^{(1)}} =$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

$$\frac{\partial E}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \phi'(\vec{z}^{(2)})$$

$$\frac{\partial \boldsymbol{E}}{\partial \vec{\boldsymbol{a}}^{(1)}} = \frac{\partial \boldsymbol{E}}{\partial \vec{\boldsymbol{z}}^{(2)}} \frac{\partial \vec{\boldsymbol{z}}^{(2)}}{\partial \vec{\boldsymbol{a}}^{(1)}} = \frac{\partial \boldsymbol{E}}{\partial \vec{\boldsymbol{z}}^{(2)}} \boldsymbol{W}^{(2)}$$

Backpropagation of error

$$\vec{x} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

$$\boxed{{}^{\partial E}/_{\partial \vec{z}^{(1)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \vec{a}^{(1)}} \quad {}^{\partial E}/_{\partial W^{(2)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \vec{z}^{(2)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \hat{y}}}$$

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \vec{a}^{(1)}$$

$$\frac{\partial E}{\partial W^{(1)}} =$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

$$\frac{\partial E}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \phi'\left(\vec{z}^{(2)}\right)$$

$$\frac{\partial E}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} W^{(2)}$$

$$\boldsymbol{\frac{\partial E}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \frac{\partial \vec{a}^{(1)}}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \phi'\left(\vec{z}^{(1)}\right)}$$

Backpropagation of error

$$\vec{x} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right)} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

$$\boxed{{}^{\partial E}\!/_{\partial W^{(1)}}} \leftarrow \boxed{{}^{\partial E}\!/_{\partial \vec{z}^{(1)}}} \leftarrow \boxed{{}^{\partial E}\!/_{\partial \vec{a}^{(1)}} \quad {}^{\partial E}\!/_{\partial W^{(2)}}} \leftarrow \boxed{{}^{\partial E}\!/_{\partial \vec{z}^{(2)}}} \leftarrow \boxed{{}^{\partial E}\!/_{\partial \hat{y}}}$$

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \vec{a}^{(1)}$$

$$\boldsymbol{\frac{\partial E}{\partial W^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(1)}} \frac{\partial \vec{z}^{(1)}}{\partial W^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(1)}} \vec{x}}$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

$$\frac{\partial E}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \phi'\left(\vec{z}^{(2)}\right)$$

$$\frac{\partial E}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} W^{(2)}$$

$$\boldsymbol{\frac{\partial E}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \frac{\partial \vec{a}^{(1)}}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \phi'\left(\vec{z}^{(1)}\right)}$$

Backpropagation of error

$$\boxed{\vec{x}} \rightarrow \boxed{\vec{z}^{(1)} = W^{(1)}\vec{x}} \rightarrow \boxed{\vec{a}^{(1)} = [1; \phi(\vec{z}^{(1)})]} \rightarrow \boxed{\vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)}} \rightarrow \boxed{\hat{y} = \vec{a}^{(2)} = \phi(\vec{z}^{(2)})} \rightarrow \boxed{E = \mathcal{L}(y, \hat{y})}$$

$$\boxed{{}^{\partial E}/_{\partial W^{(1)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \vec{z}^{(1)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \vec{a}^{(1)}} \quad {}^{\partial E}/_{\partial W^{(2)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \vec{z}^{(2)}}} \leftarrow \boxed{{}^{\partial E}/_{\partial \hat{y}}}$$

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \vec{a}^{(1)}$$

$$\frac{\partial E}{\partial W^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(1)}} \frac{\partial \vec{z}^{(1)}}{\partial W^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(1)}} \vec{x}$$

$$\boldsymbol{\Delta W^{(l)} = \eta \frac{\partial E}{\partial W^{(l)}}}$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

$$\frac{\partial E}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \phi'(\vec{z}^{(2)})$$

$$\frac{\partial E}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} W^{(2)}$$

$$\frac{\partial E}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \frac{\partial \vec{a}^{(1)}}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \phi'(\vec{z}^{(1)})$$

Backpropagation of error

$$\vec{x} \rightarrow \vec{z}^{(1)} = W^{(1)}\vec{x} \rightarrow \vec{a}^{(1)} = \left[1; \phi\left(\vec{z}^{(1)}\right)\right] \rightarrow \vec{z}^{(2)} = W^{(2)}\vec{a}^{(1)} \rightarrow \hat{y} = \vec{a}^{(2)} = \phi\left(\vec{z}^{(2)}\right) \rightarrow E = \mathcal{L}(y, \hat{y})$$

$$\partial E/_{\partial W^{(1)}} \leftarrow \partial E/_{\partial \vec{z}^{(1)}} \leftarrow \partial E/_{\partial \vec{a}^{(1)}} \;\; \partial E/_{\partial W^{(2)}} \leftarrow \partial E/_{\partial \vec{z}^{(2)}} \leftarrow \partial E/_{\partial \hat{y}}$$

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial W^{(2)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \vec{a}^{(1)}$$

$$\frac{\partial E}{\partial W^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(1)}} \frac{\partial \vec{z}^{(1)}}{\partial W^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(1)}} \vec{x}$$

$$\boldsymbol{\Delta W^{(l)} = \eta \frac{\partial E}{\partial W^{(l)}}}$$

$$\frac{\partial E}{\partial \hat{y}} = \mathcal{L}'(y, \hat{y})$$

Derivative of activation function.

$$\frac{\partial E}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \vec{z}^{(2)}} = \frac{\partial E}{\partial \hat{y}} \phi'\left(\vec{z}^{(2)}\right)$$

$$\frac{\partial E}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} \frac{\partial \vec{z}^{(2)}}{\partial \vec{a}^{(1)}} = \frac{\partial E}{\partial \vec{z}^{(2)}} W^{(2)}$$

$$\frac{\partial E}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \frac{\partial \vec{a}^{(1)}}{\partial \vec{z}^{(1)}} = \frac{\partial E}{\partial \vec{a}^{(1)}} \phi'\left(\vec{z}^{(1)}\right)$$

Choosing an activation function

- In order to do backpropagation we need to compute the *derivative of the activation function*.
  - The *unit step function* of the perceptron is not differentiable.
  - The *linear activation* used for *Adaline* does not benefit from multiple layers:

$$\hat{y} = \phi_{\text{linear}}\left(W^{(2)}\phi_{\text{linear}}\left(W^{(1)}x\right)\right) = W^{(2)}W^{(1)}x = W'x$$

- We are looking for a step-like differentiable function.

Choosing an activation function

- In order to do backpropagation we need to compute the *derivative of the activation function.*
  - The *unit step function* of the perceptron is not differentiable.
  - The *linear activation* used for *Adaline* does not benefit from multiple layers:

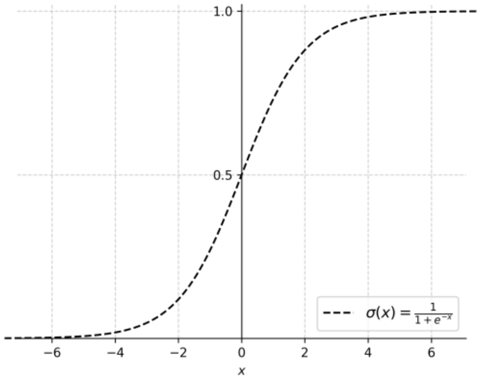
$$\hat{y} = \phi_{\text{linear}}\left(W^{(2)}\phi_{\text{linear}}\left(W^{(1)}x\right)\right) = W^{(2)}W^{(1)}x = W'x$$

- We are looking for a step-like differentiable function.
  - **Standard Logistic Function**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)\big(1-\sigma(x)\big)$$

Usually referred to as "the" **sigmoid**.



Choosing a loss function

- Adaline used a **least squares loss** function:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \qquad \mathcal{L}'(y, \hat{y}) = -(y - \hat{y})$$

Choosing a loss function

- Adaline used a **least squares loss** function:

$$\mathcal{L}(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2 \qquad \mathcal{L}'(y,\hat{y}) = -(y-\hat{y})$$

- The **Cross-entropy loss** is much more common in practice.

$$\mathcal{L}(y,\hat{y}) = -y\log\hat{y} - (1-y)\log(1-\hat{y}) \qquad \mathcal{L}'(y,\hat{y}) = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$

- Typically, the output layer uses **Softmax** activation, instead of a *sigmoid* for each neuron, to make it suitable for probabilistic interpretation.

$$\hat{y}_j = \frac{e^{z_j^{(2)}}}{\sum_{\mathrm{k=1}}^{\mathrm{o}} e^{z_k^{(2)}}} \qquad \text{instead of} \qquad \hat{y}_j = a_j^{(2)} = \sigma\left(z_j^{(2)}\right)$$

Universal approximation theorem

- The **universal approximation theorem** states that a multilayer perceptron with a single hidden layer containing a finite number of neurons is sufficient to represent **any function** given appropriate parameters.
    - However, it does not say anything about the learnability of those parameters.
- This implies that the model which the MLP learns can be *arbitrarily complex*, which can rapidly lead to **overfitting**.

MLP
Regularization Techniques

MLP Regularization

- MLP models can become *very complex* and this makes them prone to **overfitting**, especially when the training set is not large enough.

MLP Regularization

- MLP models can become *very complex* and this makes them prone to **overfitting**, especially when the training set is not large enough.
- One way to reduce this problem is to try *simpler model structures* (i.e. less layers, less neurons per layer) and check how much it affects *validation error.*
    - This approach can be very time consuming.

MLP Regularization

- MLP models can become *very complex* and this makes them prone to **overfitting**, especially when the training set is not large enough.
- One way to reduce this problem is to try *simpler model structures* (i.e. less layers, less neurons per layer) and check how much it affects *validation error.*
    - This approach can be very time consuming.
- Another method is to employ **regularization techniques.**
    - Modifying the error function by adding a term which *penalizes model complexity*.
    - Techniques which automatically *reduce model capacity* during or after training.
    - Artificially *increasing training set size.*

Weight Decay

- Similarly to *Ridge Regression* and *SVMs*, keeping the weights small makes the represented function smoother and less prone to overfitting.

Weight Decay

- Similarly to *Ridge Regression* and *SVMs*, keeping the weights small makes the represented function smoother and less prone to overfitting.

$$E = \mathcal{L}(y, \hat{y}) + {}^{\lambda}/_{2} \cdot \|\vec{w}\|_2^2$$

Tikhonov regularization.

L_2 regularization.

Weight Decay

- Similarly to *Ridge Regression* and *SVMs*, keeping the weights small makes the represented function smoother and less prone to overfitting.

$$E = \mathcal{L}(y, \hat{y}) + {}^{\lambda}/_{2} \cdot \|\vec{w}\|_2^2$$

Tikhonov regularization.

L_2 regularization.

- By applying *gradient descent* to the new error function we obtain:

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} - \eta \lambda w_{ij}^{(l)}$$

Subtracting a fraction of the weight.

Weight Decay

- Similarly to *Ridge Regression* and *SVMs*, keeping the weights small makes the represented function smoother and less prone to overfitting.

$$E = \mathcal{L}(y, \hat{y}) + {}^{\lambda}/_{2} \cdot \|\vec{w}\|_2^2$$

Tikhonov regularization.

L_2 regularization.

- By applying *gradient descent* to the new error function we obtain:

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} - \eta \lambda w_{ij}^{(l)}$$

Subtracting a fraction of the weight.

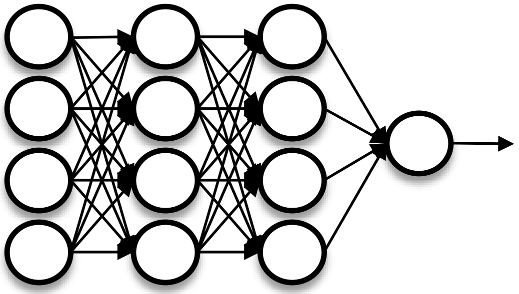
- We can also add the L_1 norm (like in *Lasso Regression*):

$$E = \mathcal{L}(y, \hat{y}) + \lambda \|\vec{w}\|_1 \quad \Longrightarrow \quad \Delta w_{ij}^{(l)} = -\eta \frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} - \eta \lambda \cdot \operatorname{sign}\left(w_{ij}^{(l)}\right)$$

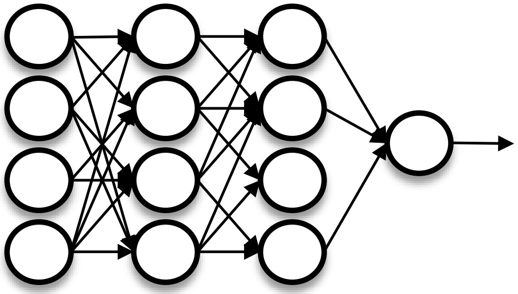
L_1 regularization.

Optimal Brain Damage

- After training, completely remove some weights (set them to 0) if it does not affect validation error too much.
    - Similar to pruning decision trees.
    - **“Optimal Brain Damage”,** Yann LeCun, John S. Denker and Sara A. Solla, 1990



During Training

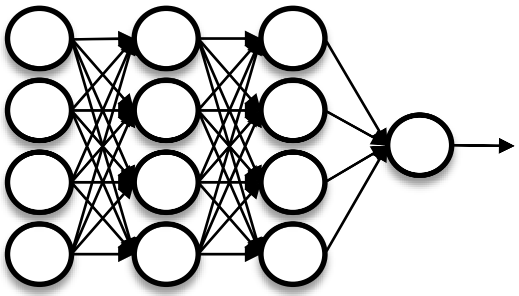


After Training

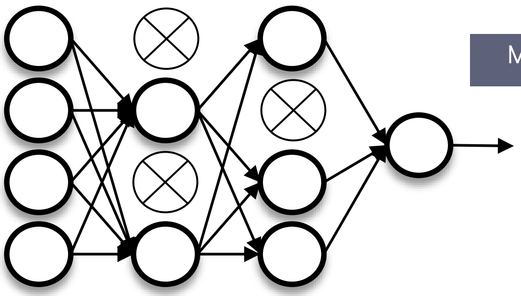
Dropout

- **Dropout** is a regularization technique which randomly disables certain neurons during some steps of the training phase.
  - **"Dropout: A Simple Way to Prevent Neural Networks from Overfitting"**

    N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014



Without Dropout



With Dropout

Much newer technique!

- Each neuron has a probability $p_i^{(l)}$ of being dropped.
- After training, during inference, all neurons are used.

In practice, all neurons on a layer have the same dropout probability.

Popularity of Neural Networks

Popularity of Neural Networks

- Even with all the developments and breakthroughs from the original *Perceptron*, NNs were not a very popular ML technique during the 90s and early 2000s in comparison to other algorithms (especially *SVMs*).

Popularity of Neural Networks

- Even with all the developments and breakthroughs from the original *Perceptron*, NNs were not a very popular ML technique during the 90s and early 2000s in comparison to other algorithms (especially *SVMs*).
- Neural networks have *many parameters to learn*.
    - This means that they require *considerably more training examples* than traditional techniques and, therefore, lots of *computational resources*.

Popularity of Neural Networks

- Even with all the developments and breakthroughs from the original *Perceptron*, NNs were not a very popular ML technique during the 90s and early 2000s in comparison to other algorithms (especially *SVMs*).
- Neural networks have *many parameters to learn*.
    - This means that they require *considerably more training examples* than traditional techniques and, therefore, lots of *computational resources*.
- They also have many *hyperparameters* to tune.
    - *Learning rate, Batch size, Regularization strength, Dropout probability*.
    - But most importantly, the **architecture of the network itself**.

Popularity of Neural Networks

- Even with all the developments and breakthroughs from the original *Perceptron*, NNs were not a very popular ML technique during the 90s and early 2000s in comparison to other algorithms (especially *SVMs*).
- Neural networks have *many parameters to learn*.
    - This means that they require *considerably more training examples* than traditional techniques and, therefore, lots of *computational resources*.
- They also have many *hyperparameters* to tune.
    - *Learning rate, Batch size, Regularization strength, Dropout probability*.
    - But most importantly, the **architecture of the network itself**.
- Simply put, they were slow, hard to train and did not work as well as other techniques.

Popularity of Neural Networks

- Three key factors determined the large increase in performance and, hence, popularity of neural networks, under the name “**Deep Learning**”, in the late 2000s and early 2010s.

Popularity of Neural Networks

- Three key factors determined the large increase in performance and, hence, popularity of neural networks, under the name “**Deep Learning**”, in the late 2000s and early 2010s.
- **Training Data**
  - Large neural networks require huge amounts of data for training.
  - Many large, high-quality datasets have been published over the past few years, which were not available to researchers in the 90s.
  - More data is generated everyday nowadays than was generated in years a decade ago.

Popularity of Neural Networks

- Three key factors determined the large increase in performance and, hence, popularity of neural networks, under the name “**Deep Learning**”, in the late 2000s and early 2010s.
- **Training Data**
  - Large neural networks require huge amounts of data for training.
  - Many large, high-quality datasets have been published over the past few years, which were not available to researchers in the 90s.
  - More data is generated everyday nowadays than was generated in years a decade ago.
- **Computing Power**
  - The use of GPUs and dedicated architectures, like TPUs, for training has significantly improved the speed and allowed the use of much larger neural networks.
  - *“a training run that takes one day on a single TPU device would have taken a quarter of a million years on an 80486 from 1990”*. - Shane Legg, cofounder of Google DeepMind.

Popularity of Neural Networks

- Three key factors determined the large increase in performance and, hence, popularity of neural networks, under the name "**Deep Learning**", in the late 2000s and early 2010s.
- **Training Data**
  - Large neural networks require huge amounts of data for training.
  - Many large, high-quality datasets have been published over the past few years, which were not available to researchers in the 90s.
  - More data is generated everyday nowadays than was generated in years a decade ago.
- **Computing Power**
  - The use of GPUs and dedicated architectures, like TPUs, for training has significantly improved the speed and allowed the use of much larger neural networks.
  - *"a training run that takes one day on a single TPU device would have taken a quarter of a million years on an 80486 from 1990"*. - Shane Legg, cofounder of Google DeepMind.
- **Algorithms**
  - Improvements in network structure (Convolutions, Recurrent Networks), activation functions (ReLu) and optimizers (RMSprop, ADAM) have greatly improved the performance of NNs.

Keywords

Perceptron

McCulloch-Pitts Neuron

Hebb's Rule

Linear Separability

AI Winter

Learning Rate

Batch Update

Gradient Descent

Chain Rule

Delta Rule

Multilayer Perceptron

MLP

Hidden Layer

Backpropagation

Universal approximation theorem

Weight Decay

Optimal Brain Damage

Dropout

Warren McCulloch

Walter Pits

Donald Hebb

Frank Rosenblatt

Marvin Minsky

Seymour Papert

Bernard Widrow

Ted Hoff

Geoffrey Hinton