IT5005:
Multi-Layer Perceptrons

Sirigina Rajendra Prasad

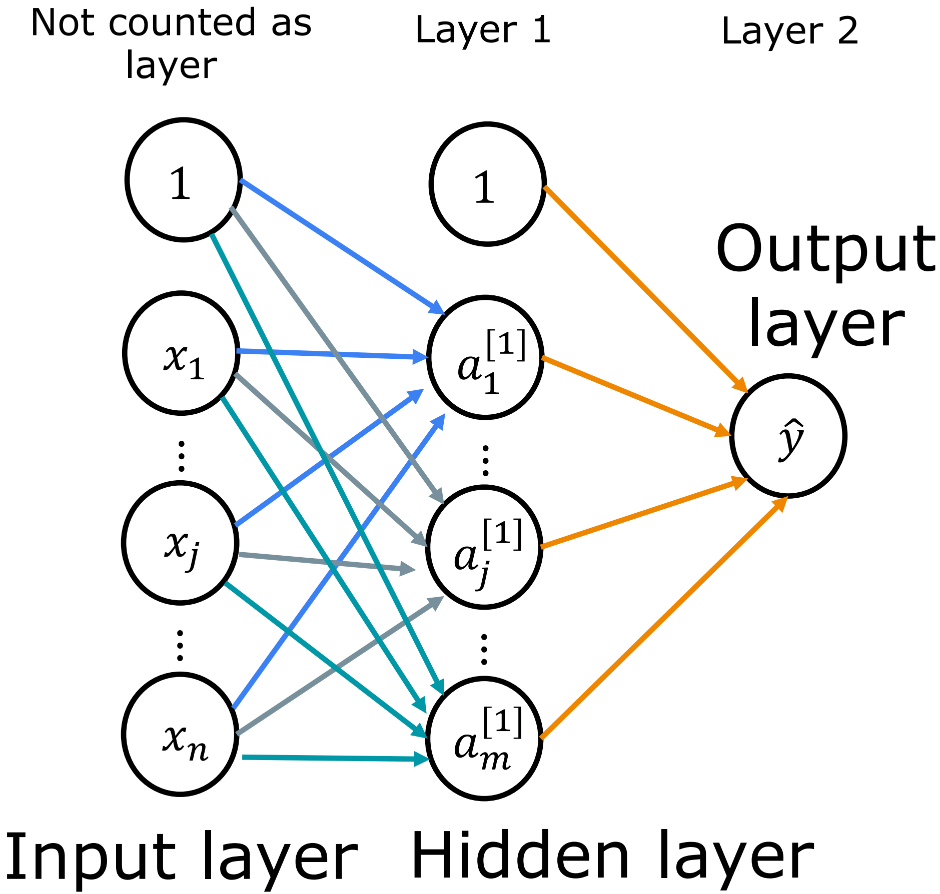
Slide Credit: Prof. Ben Leong
Revised for IT5005

Agenda

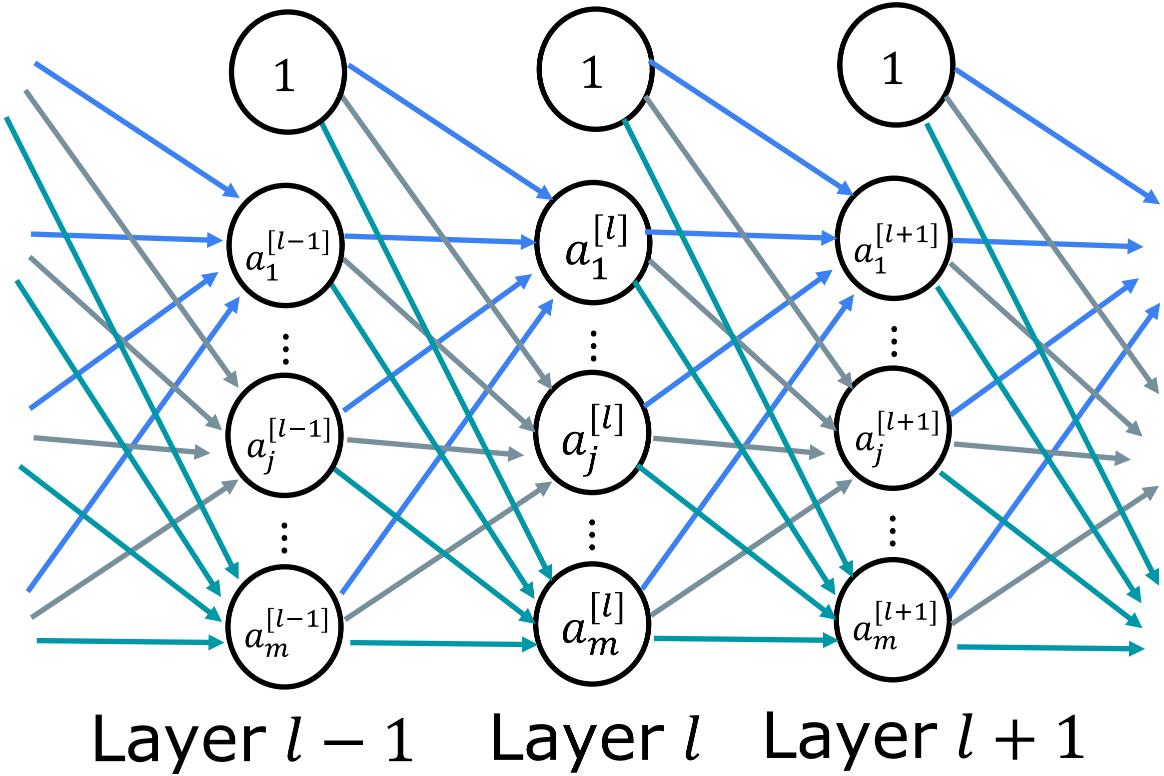
- Multilayer Perceptron
- Backpropagation

Neural Network
= Multi-layer Perceptron

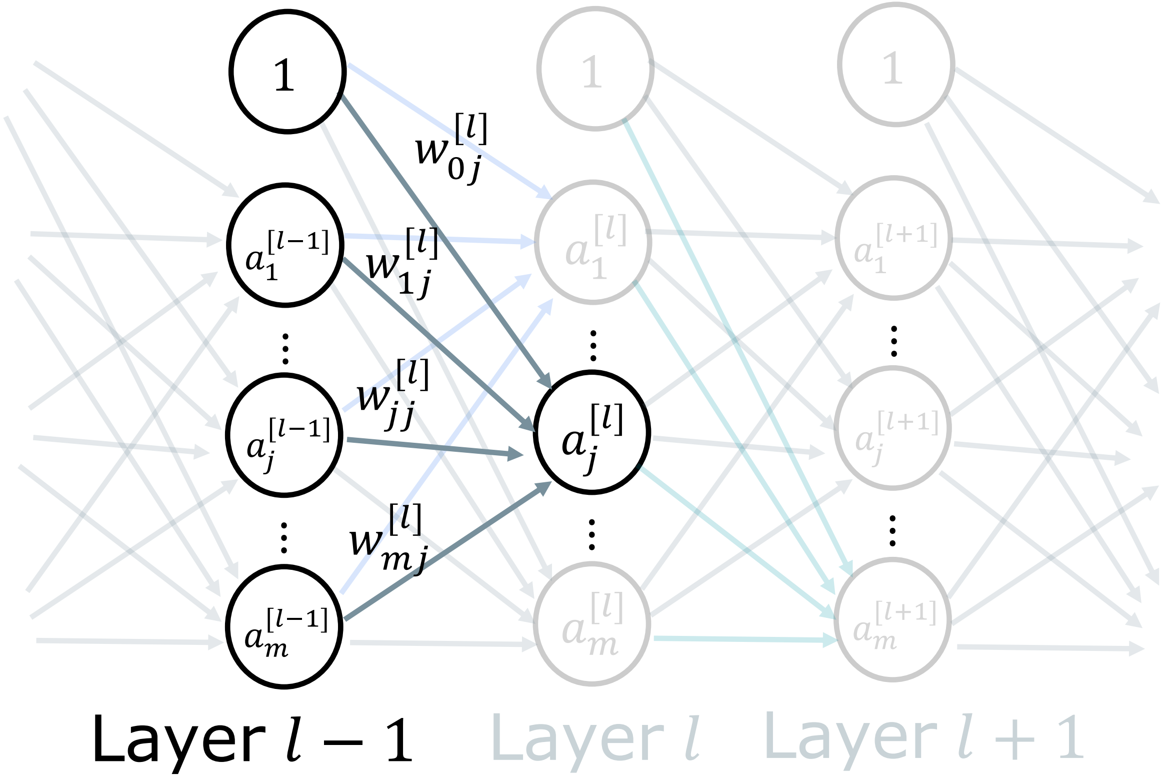
2-layer Perceptron



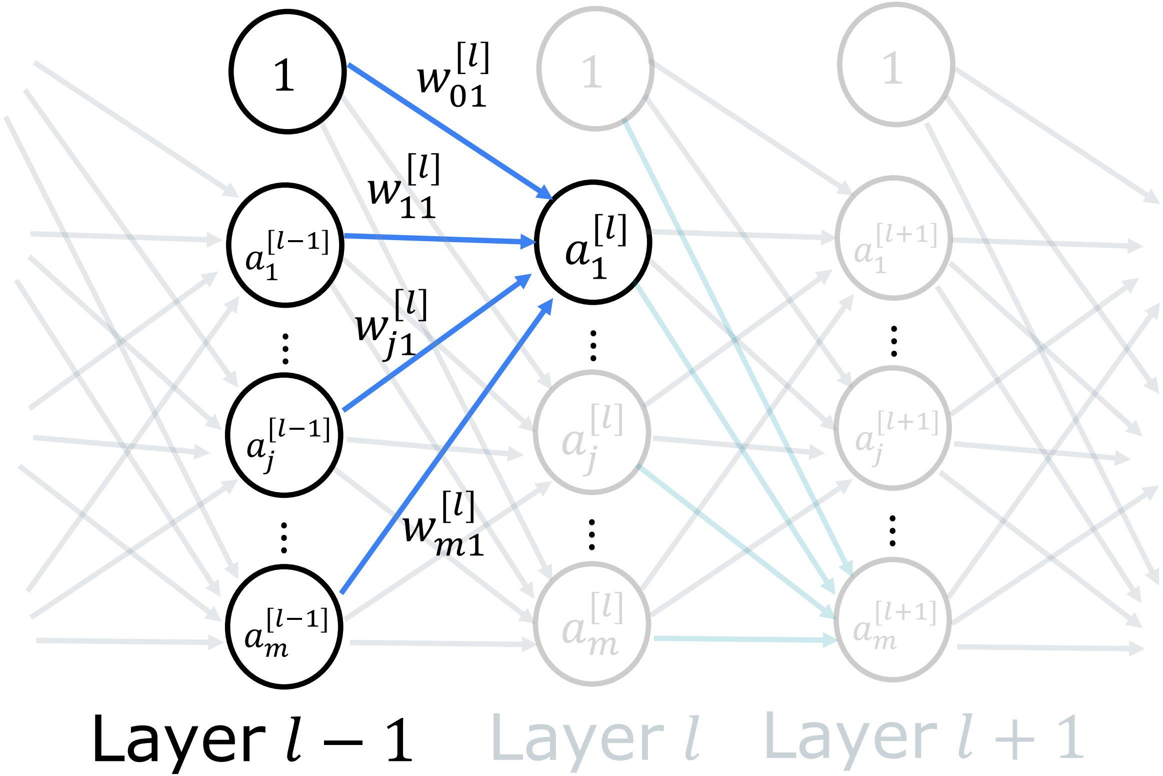
Multi-layer Perceptron



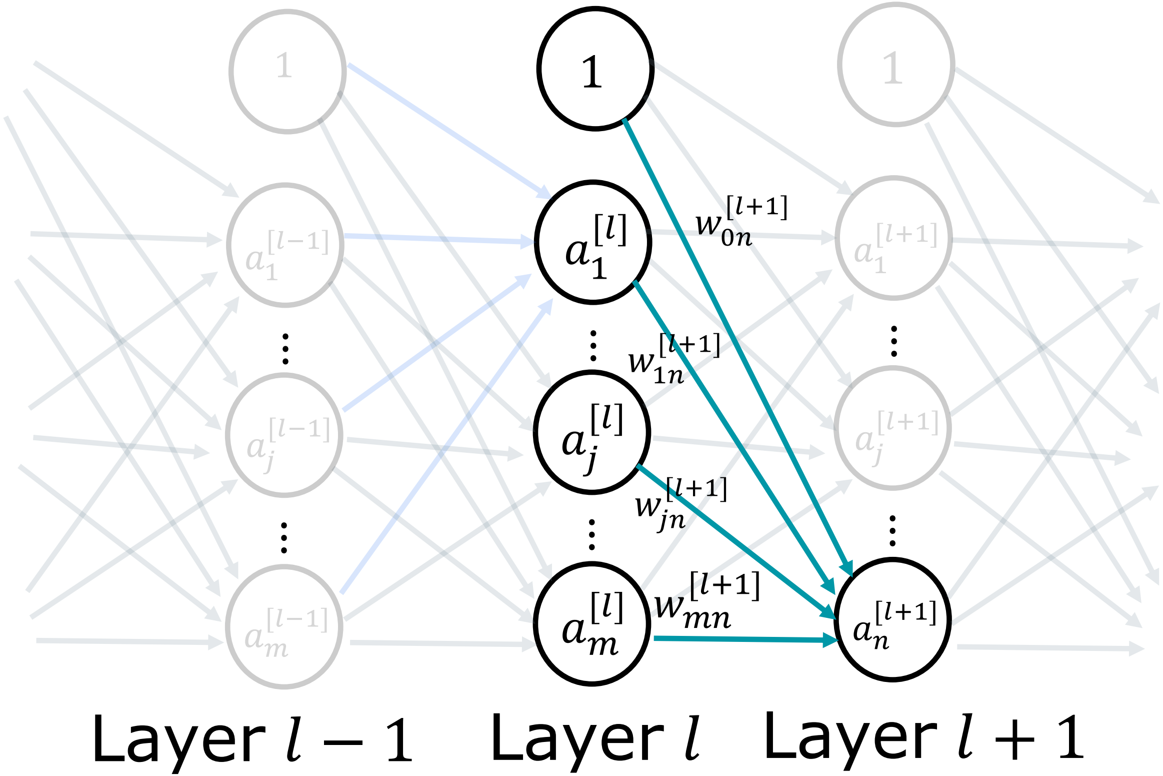
Multi-layer Neural Network



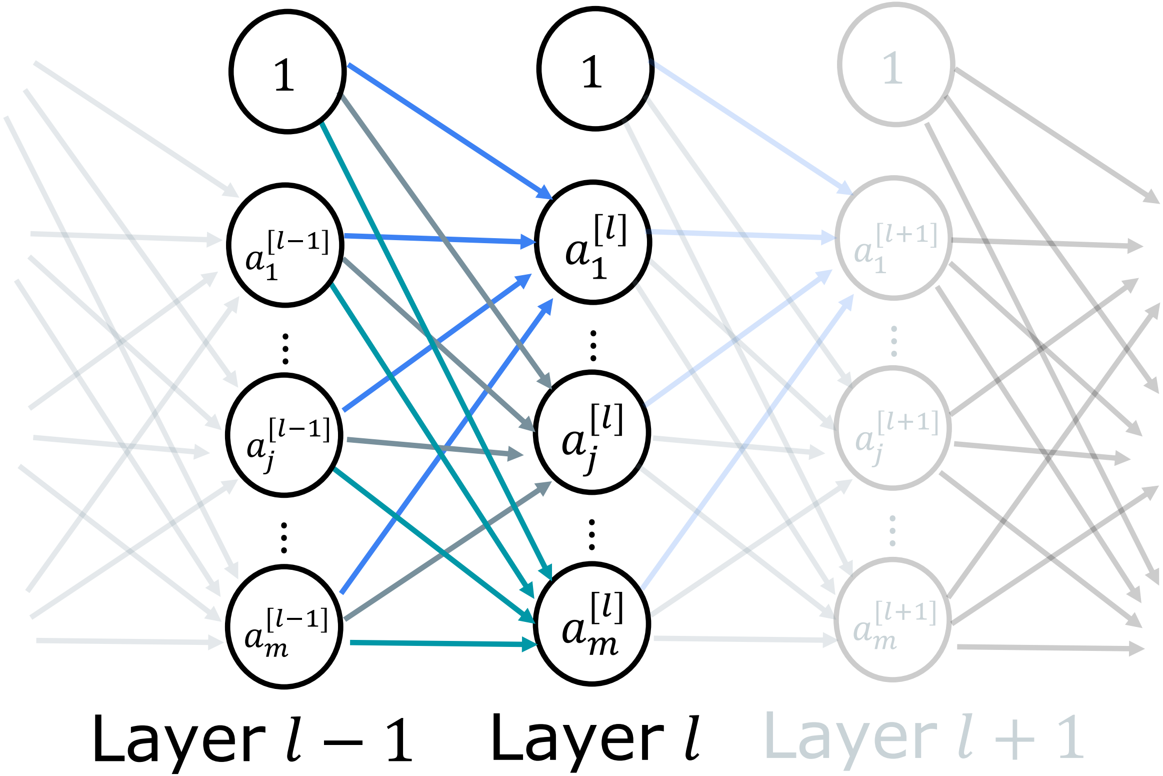
Multi-layer Neural Network



Multi-layer Neural Network



Multi-layer Perceptron



Layer Activation

$$a = g(f(\boldsymbol{x})), f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$$

Single-Layer Perceptron

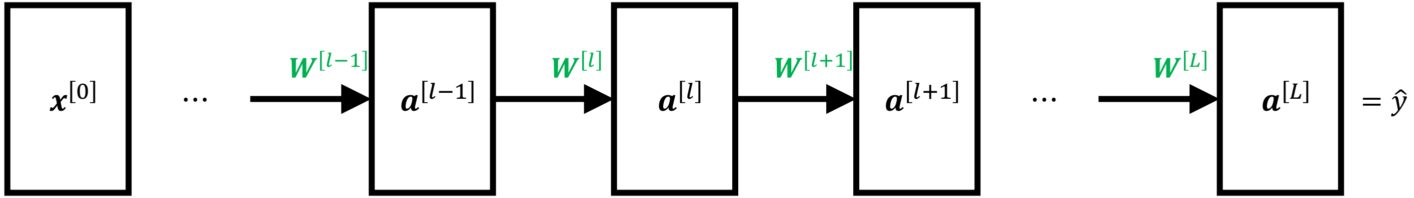
$$\underbrace{\boldsymbol{a}^{[l]}}_{\text{Layer } l \text{ Activations}} = \overbrace{g^{[l]}}^{\text{Layer } l \text{ Activation Function}}\left(\left(\overbrace{\boldsymbol{W}^{[l]}}^{\text{Layer } l \text{ Weights}}\right)^{\top} \underbrace{\boldsymbol{a}^{[l-1]}}_{\text{Layer } l-1 \text{ Activations}}\right)$$

Layer l in Neural Network

Forward Propagation

$$\boldsymbol{a}^{[l]} \equiv g^{[l]}(f^{[l]}), \;\; f^{[l]} \equiv \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

$$g^{[1]}\left(f^{[1]}(\boldsymbol{x}^{[0]})\right) = \boldsymbol{a}^{[1]} \qquad g^{[l]}\left(f^{[l]}(\boldsymbol{a}^{[l-1]})\right) = \boldsymbol{a}^{[l]} \qquad g^{[L]}\left(f^{[L]}(\boldsymbol{a}^{[L-1]})\right) = \boldsymbol{a}^{[L]}$$

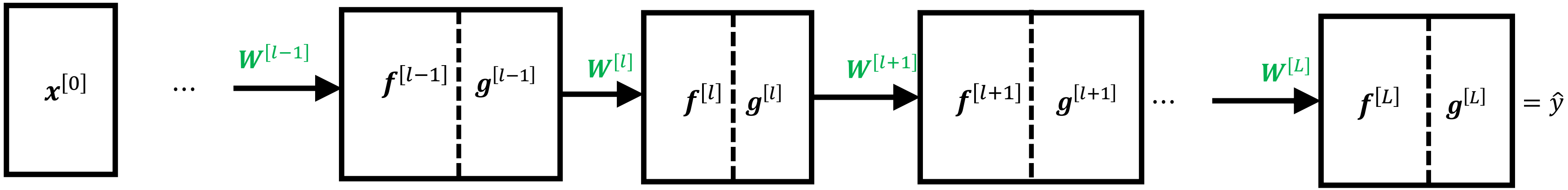


$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]})))))))))$$

Forward Propagation

$$\boldsymbol{a}^{[l]} \equiv g^{[l]}(f^{[l]}), \quad f^{[l]} \equiv \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

$$g^{[1]}\left(f^{[1]}(\boldsymbol{x}^{[0]})\right) = \boldsymbol{a}^{[1]} \qquad g^{[l]}\left(f^{[l]}(\boldsymbol{a}^{[l-1]})\right) = \boldsymbol{a}^{[l]} \qquad g^{[L]}\left(f^{[L]}(\boldsymbol{a}^{[L-1]})\right) = \boldsymbol{a}^{[L]}$$

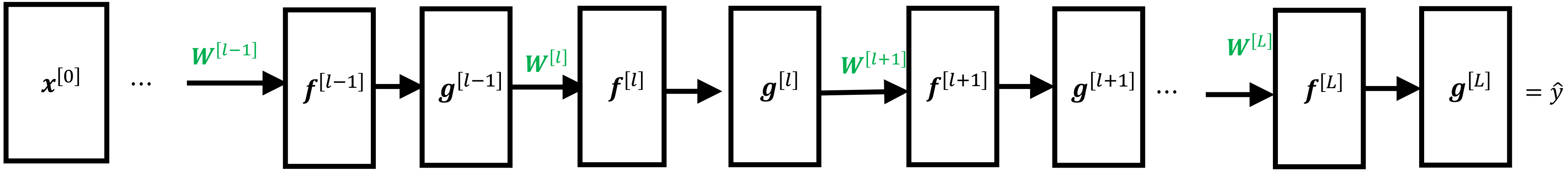


$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

Forward Propagation

$$\boldsymbol{a}^{[l]} \equiv g^{[l]}(f^{[l]}), \; f^{[l]} \equiv \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

$$g^{[1]}\left(f^{[1]}(\boldsymbol{x}^{[0]})\right) = \boldsymbol{a}^{[1]} \qquad g^{[l]}\left(f^{[l]}(\boldsymbol{a}^{[l-1]})\right) = \boldsymbol{a}^{[l]} \qquad g^{[L]}\left(f^{[L]}(\boldsymbol{a}^{[L-1]})\right) = \boldsymbol{a}^{[L]}$$



$$\hat{y}(x) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

How do we compute the weights $W^{[l]}$?

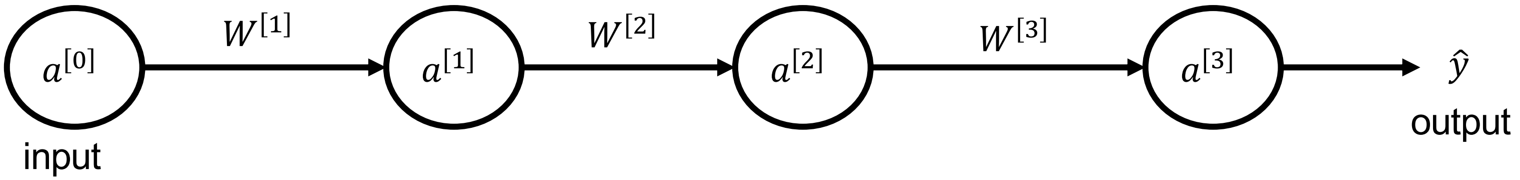
Recap: Gradient Descent

1. Decide on some loss function $\mathcal{E}$, which is general some function of $\hat{y} - y$
2. Compute $\dfrac{\partial \varepsilon}{\partial \boldsymbol{w}}$
3. Iterate until convergence:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial \varepsilon}{\partial \boldsymbol{w}}$$

Backpropagation

Backpropagation: Scalar Example

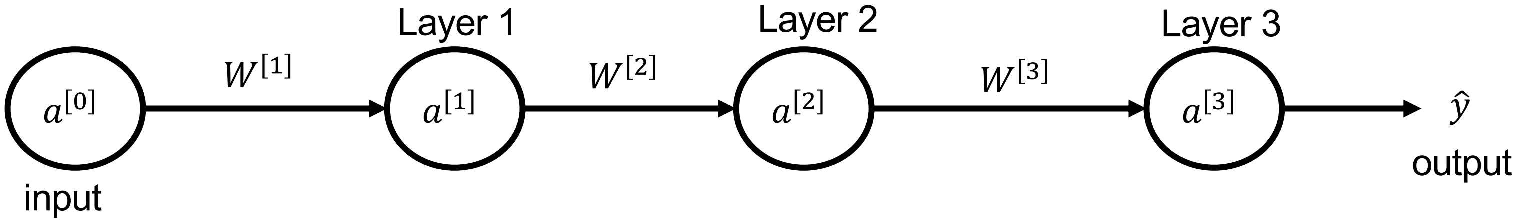


A three-layer NN with single perceptron in each layer

scalar variables

Back Propagation: Scalar Example

A three-layer NN with single perceptron in each layer (scalar variables)

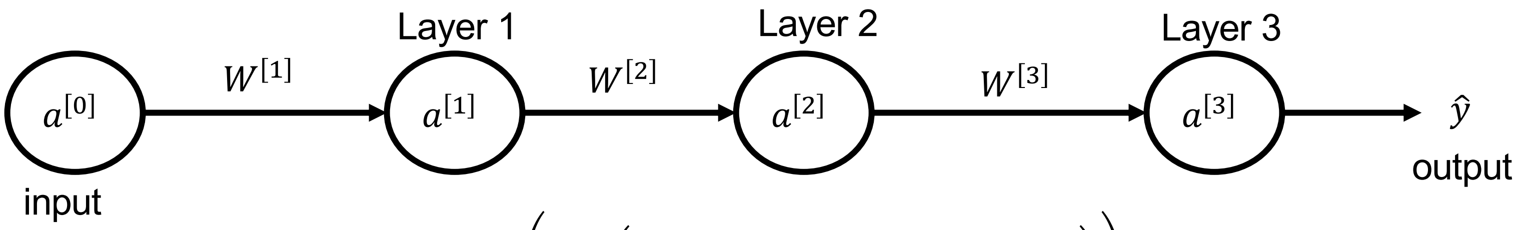


$$\hat{y} = a^{[3]} = g^{[3]}\left(f^{[3]}\left(g^{[2]}\left(f^{[2]}\left(g^{[1]}\left(f^{[1]}\left(a^{[0]}\right)\right)\right)\right)\right)\right)$$

$a^{[l]}$: Output of perceptron at l-th layer
$f^{[l]}$: Input of perceptron at l-th layer
$W^{[l]}$: Weights of l-th layer
Input Layers : Layer 0
Hidden Layers : Layer 1 and Layer 2
Output Layer : Layer 3

Back Propagation: Scalar Example

A three-layer NN with single perceptron in each layer (scalar variables)



$$\hat{y} = a^{[3]} = g^{[3]}\left(f^{[3]}\left(g^{[2]}\left(f^{[2]}\left(g^{[1]}\left(f^{[1]}\left(a^{[0]}\right)\right)\right)\right)\right)\right)$$

$$\hat{y} = a^{[3]}$$
$$a^{[3]} = g^{[3]}\left(f^{[3]}\right)$$
$$a^{[2]} = g^{[2]}\left(f^{[2]}\right)$$
$$a^{[1]} = g^{[1]}\left(f^{[1]}\right)$$
$$a^{[0]} = \mathrm{x}$$

$$f^{[3]} = W^{[3]}\ a^{[2]}$$
$$f^{[2]} = W^{[2]}\ a^{[1]}$$
$$f^{[1]} = W^{[1]}\ a^{[0]}$$

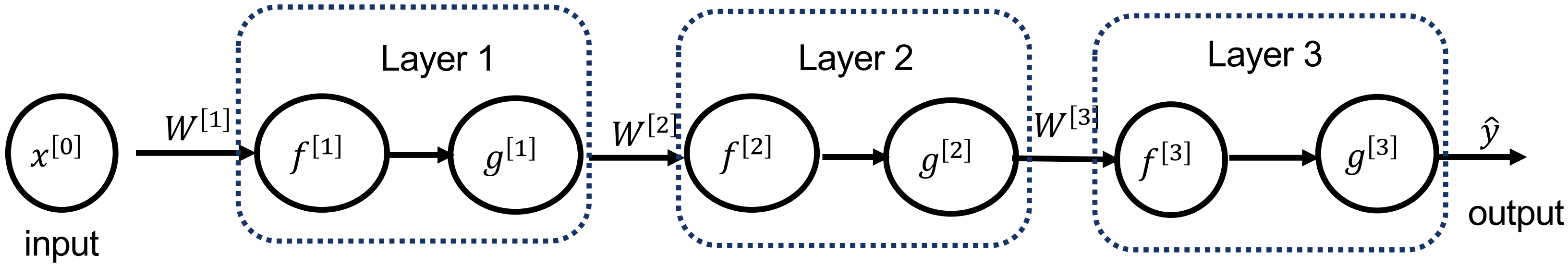
Loss function: MSE

$$\mathcal{E} = \frac{1}{2}(\hat{y} - \mathrm{y})^2$$

$$W^{[i]} := W^{[i]} - \eta \frac{\partial \varepsilon}{\partial W^{[i]}}$$

Back Propagation: Scalar Example

A three-layer NN with single perceptron in each layer (scalar variables)



$$\hat{y} = a^{[3]} = g^{[3]}\left(f^{[3]}\left(g^{[2]}\left(f^{[2]}\left(g^{[1]}\left(f^{[1]}(a^{[0]})\right)\right)\right)\right)\right)$$

$$\begin{aligned} \hat{y} &= a^{[3]} \\ a^{[3]} &= g^{[3]}(f^{[3]}) \\ a^{[2]} &= g^{[2]}(f^{[2]}) \\ a^{[1]} &= g^{[1]}(f^{[1]}) \\ a^{[0]} &= \mathrm{x} \end{aligned}$$

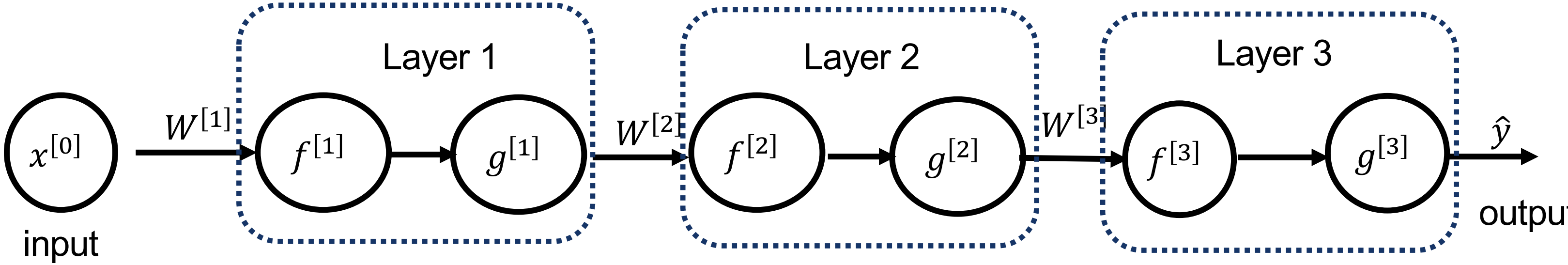
$$\begin{aligned} f^{[3]} &= W^{[3]}\, a^{[2]} \\ f^{[2]} &= W^{[2]}\, a^{[1]} \\ f^{[1]} &= W^{[1]}\, a^{[0]} \end{aligned}$$

Loss function: MSE

$$\mathcal{E} = \frac{1}{2}(\hat{y} - \mathrm{y})^2$$

$$W^{[i]} := W^{[i]} - \eta \frac{\partial \varepsilon}{\partial W^{[i]}}$$

$$\mathcal{E} = \frac{1}{2}(\hat{y} - y)^2 \qquad \frac{\partial \mathcal{E}}{\partial W^{[l]}} = (\hat{y} - y)\frac{\partial \hat{y}}{\partial W^{[l]}}$$



$$\hat{y} = a^{[3]}$$
$$a^{[3]} = g^{[3]}(f^{[3]})$$
$$a^{[2]} = g^{[2]}(f^{[2]})$$
$$a^{[1]} = g^{[1]}(f^{[1]})$$

$$\frac{\partial \hat{y}}{\partial W^{[3]}} = \frac{\partial g^{[3]}}{\partial W^{[3]}} = \frac{\partial g^{[3]}}{\partial f^{[3]}} \frac{\partial f^{[3]}}{\partial W^{[3]}}$$

Store this value and reuse for gradients w.r.t $W^{[1]}$ and $W^{[2]}$

$$f^{[3]} = W^{[3]}\, a^{[2]}$$

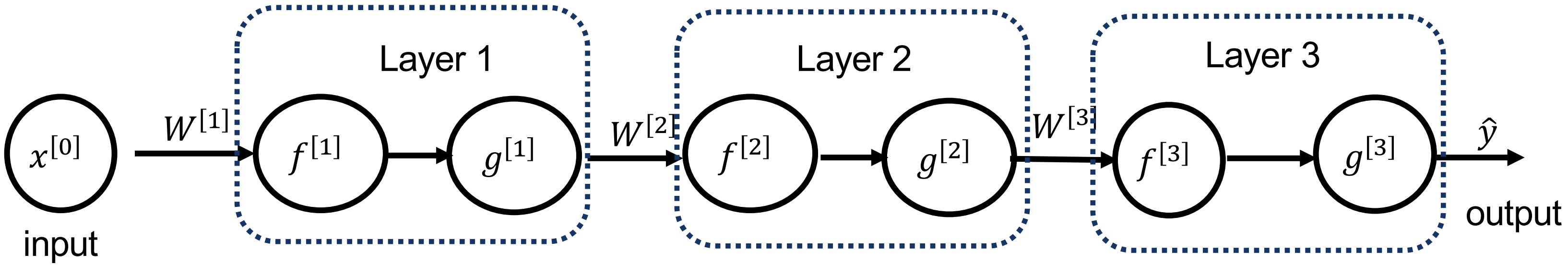
$$\frac{\partial f^{[3]}}{\partial W^{[3]}} = a^{[2]}$$

$$\frac{\partial \hat{y}}{\partial W^{[3]}} = \delta^{[3]} a^{[2]} \quad where\ \delta^{[3]} = \frac{\partial g^{[3]}}{\partial f^{[3]}}$$

$$f^{[3]} = W^{[3]}\, a^{[2]}$$
$$f^{[2]} = W^{[2]}\, a^{[1]}$$
$$f^{[1]} = W^{[1]}\, a^{[0]}$$

$$\delta^{[l]} = \frac{\partial g^{[L]}}{\partial f^{[l]}}$$

$$\mathcal{E} = \frac{1}{2}(\hat{y} - y)^2 \qquad \frac{\partial \mathcal{E}}{\partial W^{[l]}} = (\hat{y} - y)\frac{\partial \hat{y}}{\partial W^{[l]}}$$



From Layer 3

Store this value and reuse for gradients w.r.t $W^{[1]}$

$$\frac{\partial \hat{y}}{\partial W^{[2]}} = \frac{\partial g^{[3]}}{\partial W^{[2]}} = \underbrace{\frac{\partial g^{[3]}}{\partial f^{[3]}} \frac{\partial f^{[3]}}{\partial g^{[2]}} \frac{\partial g^{[2]}}{\partial f^{[2]}}} \frac{\partial f^{[2]}}{\partial W^{[2]}}$$

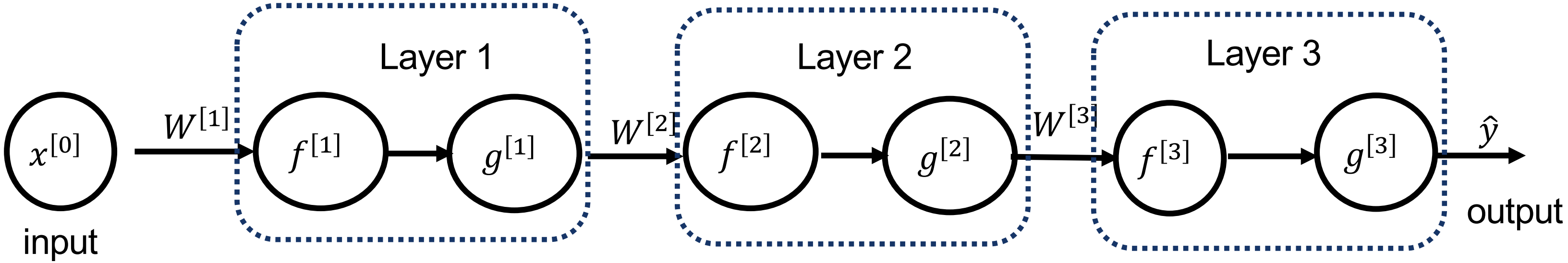
$$= \delta^{[3]} W^{[3]} \frac{\partial g^{[2]}}{\partial f^{[2]}} a^{[1]}$$

$$\frac{\partial \hat{y}}{\partial W^{[2]}} = \delta^{[2]} a^{[1]}$$ where $\delta^{[2]} = \frac{\partial g^{[3]}}{\partial f^{[2]}} = \delta^{[3]} W^{[3]} \frac{\partial g^{[2]}}{\partial f^{[2]}}$

$$\hat{y} = a^{[3]}$$
$$a^{[3]} = g^{[3]}(f^{[3]})$$
$$a^{[2]} = g^{[2]}(f^{[2]})$$
$$a^{[1]} = g^{[1]}(f^{[1]})$$

$$f^{[3]} = W^{[3]}\, a^{[2]}$$
$$f^{[2]} = W^{[2]}\, a^{[1]}$$
$$f^{[1]} = W^{[1]}\, a^{[0]}$$

$$\mathcal{E} = \frac{1}{2}(\hat{y} - y)^2 \qquad \frac{\partial \mathcal{E}}{\partial W^{[l]}} = (\hat{y} - y)\frac{\partial \hat{y}}{\partial W^{[l]}}$$



$$\frac{\partial \hat{y}}{\partial W^{[1]}} = \frac{\partial g^{[3]}}{\partial W^{[1]}} = \overbrace{\frac{\partial g^{[3]}}{\partial f^{[3]}}\frac{\partial f^{[3]}}{\partial g^{[2]}}\frac{\partial g^{[2]}}{\partial f^{[2]}}}^{\frac{\partial g^{[3]}}{\partial f^{[2]}} \text{ from layer 2}}\frac{\partial f^{[2]}}{\partial g^{[1]}}\frac{\partial g^{[1]}}{\partial f^{[1]}}\frac{\partial f^{[1]}}{\partial W^{[1]}}$$

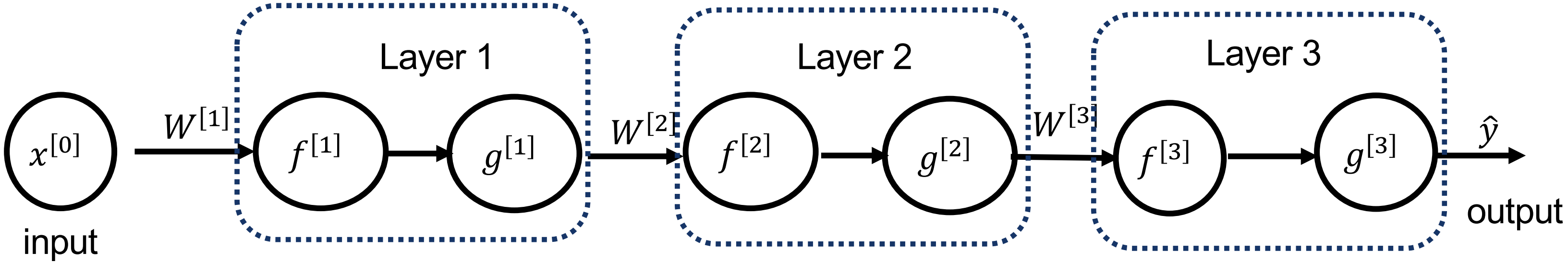
$$= \delta^{[2]} W^{[2]} \frac{\partial g^{[1]}}{\partial f^{[1]}} a^{[0]}$$

$$\frac{\partial \hat{y}}{\partial W^{[1]}} = \delta^{[1]} a^{[0]}, \quad where\ \delta^{[1]} = \delta^{[2]} W^{[2]} \frac{\partial g^{[1]}}{\partial f^{[1]}}$$

$$\hat{y} = a^{[3]}$$
$$a^{[3]} = g^{[3]}(f^{[3]})$$
$$a^{[2]} = g^{[2]}(f^{[2]})$$
$$a^{[1]} = g^{[1]}(f^{[1]})$$

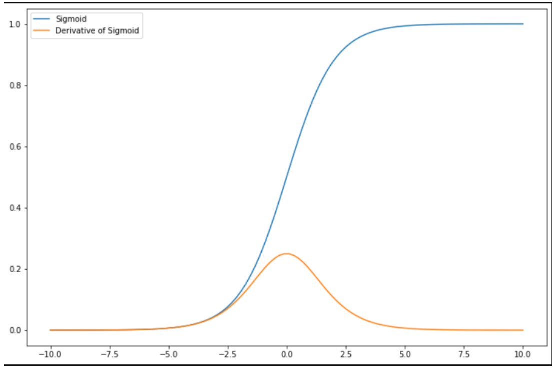
$$f^{[3]} = W^{[3]}\ a^{[2]}$$
$$f^{[2]} = W^{[2]}\ a^{[1]}$$
$$f^{[1]} = W^{[1]}\ a^{[0]}$$

Vanishing Gradient Problem



$$\frac{\partial \hat{y}}{\partial W^{[1]}} = \frac{\partial g^{[3]}}{\partial W^{[1]}} = \frac{\partial g^{[3]}}{\partial f^{[3]}} \frac{\partial f^{[3]}}{\partial g^{[2]}} \frac{\partial g^{[2]}}{\partial f^{[2]}} \frac{\partial f^{[2]}}{\partial g^{[1]}} \frac{\partial g^{[1]}}{\partial f^{[1]}} \frac{\partial f^{[1]}}{\partial W^{[1]}}$$

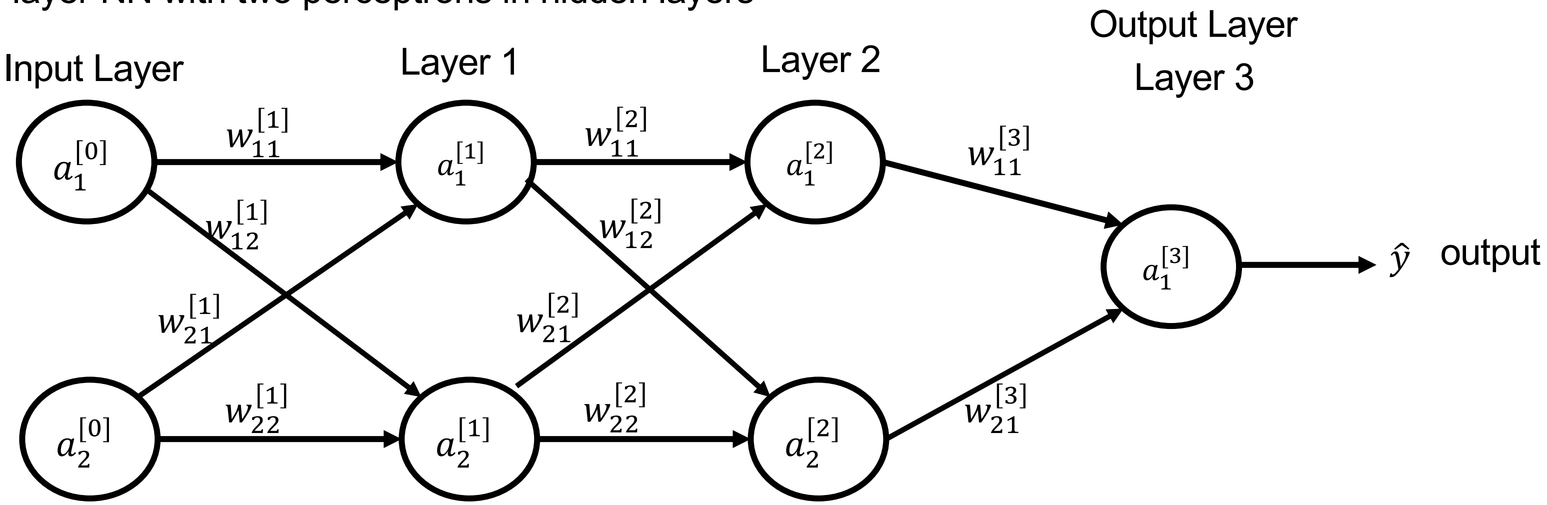
$<$ 0.25 for Sigmoid



The gradients approach zero as number of layers increase
Zero gradients mean, no update to the weights

Back Propagation: Vector Example

A three-layer NN with two perceptrons in hidden layers

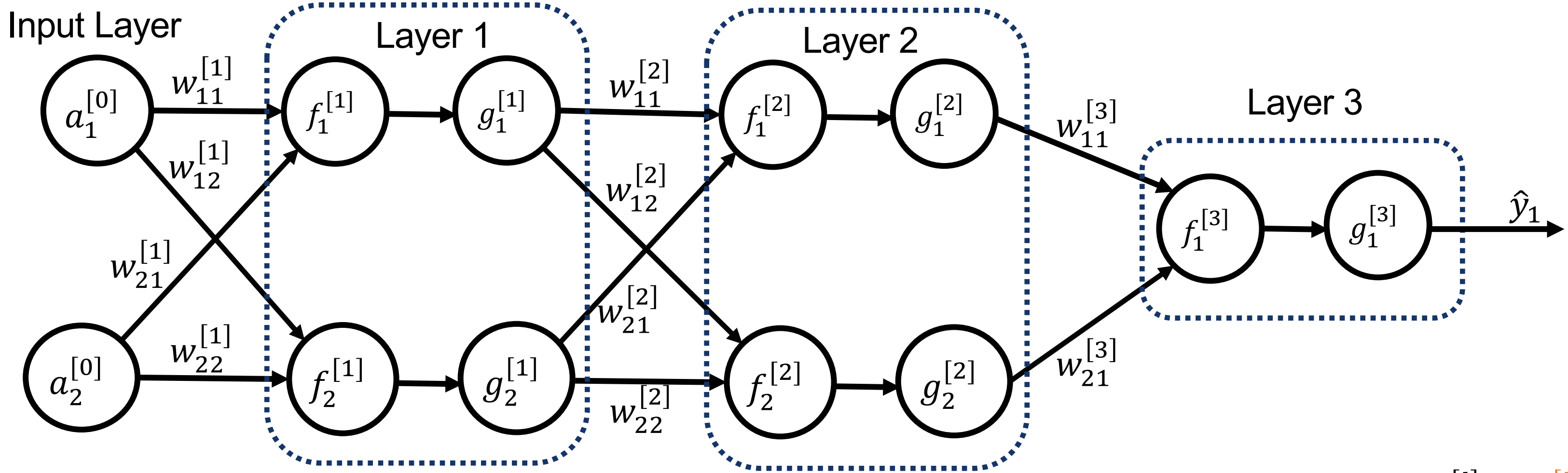


$$\hat{y} = \boldsymbol{a}^{[3]} = \boldsymbol{g}^{[3]}\left(\boldsymbol{f}^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}\left(W^{[1]}, \boldsymbol{a}^{[0]}\right)\right)\right)\right)\right)\right)$$

$$W^{[l]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} \quad \boldsymbol{a}^{[l]} = \begin{bmatrix} a_1^{[l]} \\ a_2^{[l]} \end{bmatrix} \quad \boldsymbol{f}^{[l]} = \begin{bmatrix} f_1^{[l]} \\ f_2^{[l]} \end{bmatrix} \quad \boldsymbol{g}^{[l]} = \begin{bmatrix} g_1^{[l]} \\ g_2^{[l]} \end{bmatrix} \qquad \widehat{\boldsymbol{y}} = a_1^{[3]}$$

$\boldsymbol{a}^{[l]}$: Output of perceptron at l-th layer
$\boldsymbol{f}^{[l]}$: Input of perceptron at l-th layer
$\boldsymbol{g}^{[l]}$: Activation at l-th layer
$W^{[l]}$: Weights of l-th layer
Input Layers : Layer 0
Hidden Layers : Layer 1 and Layer 2
Output Layer : Layer 3

Back Propagation: Vector Example



$$\boldsymbol{a}^{[l]} \equiv \boldsymbol{g}^{[l]}(\boldsymbol{f}^{[l]})$$

$$\hat{y} = \boldsymbol{a}^{[3]} = \boldsymbol{g}^{[3]}\left(\boldsymbol{f}^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}(W^{[1]}, \boldsymbol{a}^{[0]})\right)\right)\right)\right)\right)$$

$$\boldsymbol{f}^{[l]} \equiv \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

$$W^{[l]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} \quad \boldsymbol{a}^{[l]} = \begin{bmatrix} a_1^{[l]} \\ a_2^{[l]} \end{bmatrix} \quad \boldsymbol{f}^{[l]} = \begin{bmatrix} f_1^{[l]} \\ f_2^{[l]} \end{bmatrix} \quad \boldsymbol{g}^{[l]} = \begin{bmatrix} g_1^{[l]} \\ g_2^{[l]} \end{bmatrix}$$

$$\boldsymbol{f}^{[3]} = \left[f_1^{[3]}\right]$$

$$\boldsymbol{g}^{[3]} = \left[g_1^{[3]}\right]$$

Back Propagation: Vector Example

$$\boldsymbol{a}^{[0]} \xrightarrow{W^{[1]}} \boldsymbol{f}^{[1]} \Rightarrow \boldsymbol{g}^{[1]} \xrightarrow{W^{[2]}} \boldsymbol{f}^{[2]} \Rightarrow \boldsymbol{g}^{[2]} \xrightarrow{W^{[3]}} \boldsymbol{f}^{[3]} \Rightarrow \boldsymbol{g}^{[3]} \Rightarrow \hat{y}$$

$$\hat{y} = \boldsymbol{a}^{[3]} = \boldsymbol{g}^{[3]}\left(\boldsymbol{f}^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}\left(\boldsymbol{a}^{[0]}\right)\right)\right)\right)\right)\right)$$

$$= g_1^{[3]}\left(f_1^{[3]}\left(\boldsymbol{g}^{[2]}\right)\right)$$

Refer slide 8 of “MatrixCalculus_January2025.pdf” for the proof

$$\hat{y}'\left(W^{[3]}\right) = \frac{\partial \boldsymbol{g}^{[3]}}{\partial W^{[3]}} = \frac{\partial f_1^{[3]}}{\partial W^{[3]}} \frac{\partial g_1^{[3]}}{\partial f_1^{[3]}} = \boldsymbol{a}^{[2]} \left(\frac{\partial g_1^{[3]}}{\partial f_1^{[3]}}\right)^T$$

$$f_1^{[3]}\left(\boldsymbol{g}^{[2]}\right) = \left(W^{[3]}\right)^{\mathsf{T}} \boldsymbol{a}^{[2]}$$

$$= \boldsymbol{a}^{[2]} \left(\delta^{[3]}\right)^T$$

$$\delta^{[3]} = \frac{\partial g_1^{[3]}}{\partial f_1^{[3]}}$$

Back Propagation: Vector Example

$$\boldsymbol{a}^{[0]} \xrightarrow{W^{[1]}} \boldsymbol{f}^{[1]} \rightarrow \boldsymbol{g}^{[1]} \xrightarrow{W^{[2]}} \boldsymbol{f}^{[2]} \rightarrow \boldsymbol{g}^{[2]} \xrightarrow{W^{[3]}} \boldsymbol{f}^{[3]} \rightarrow \boldsymbol{g}^{[3]} \rightarrow \hat{y}$$

$$\hat{y} = a^{[3]} = \boldsymbol{g}^{[3]}\left(\boldsymbol{f}^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}\left(\boldsymbol{a}^{[0]}\right)\right)\right)\right)\right)\right)$$

$$= g_1^{[3]}\left(f_1^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\right)\right)\right)\right)$$

Refer slide 8 of “MatrixCalculus_January2025.pdf” for the proof

$$f^{[2]}\left(\boldsymbol{g}^{[1]}\right) = \left(W^{[2]}\right)^{\mathsf{T}} \boldsymbol{a}^{[1]}$$

$$\hat{y}'\left(W^{[2]}\right) = \frac{\partial g_1^{[3]}}{\partial W^{[2]}} = \frac{\partial \boldsymbol{f}^{[2]}}{\partial W^{[2]}} \frac{\partial g_1^{[3]}}{\partial \boldsymbol{f}^{[2]}} = \boldsymbol{a}^{[1]}\left(\frac{\partial g_1^{[3]}}{\partial \boldsymbol{f}^{[2]}}\right)^T$$

$$= \boldsymbol{a}^{[1]}\left(\frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[2]}} \frac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}} \frac{\partial g_1^{[3]}}{\partial f_1^{[3]}}\right)^T$$

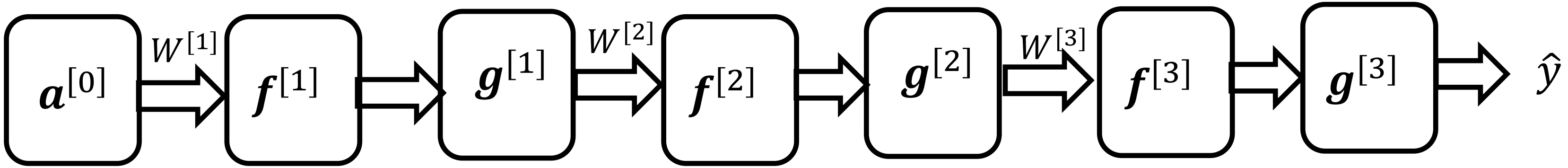
$$= \boldsymbol{a}^{[1]}\left(\frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[2]}} \frac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}} \delta^{[3]}\right)^T$$

$$= \boldsymbol{a}^{[1]}\left(\boldsymbol{\delta}^{[2]}\right)^T$$

$$\delta^{[3]} = \frac{\partial g_1^{[3]}}{\partial f_1^{[3]}}$$

$$\boldsymbol{\delta}^{[2]} = \frac{\partial g_1^{[3]}}{\partial \boldsymbol{f}^{[2]}} = \frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[2]}} \frac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}} \boldsymbol{\delta}^{[3]}$$

Back Propagation: Vector Example



$$\hat{y} = a^{[3]} = g_1^{[3]}\left(f_1^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}\left(\boldsymbol{a}^{[0]}\right)\right)\right)\right)\right)\right)$$

$$\hat{y}'\left(\boldsymbol{W}^{[1]}\right) = \frac{\partial g^{[3]}}{\partial \boldsymbol{W}^{[1]}} = \frac{\partial \boldsymbol{f}^{[1]}}{\partial \boldsymbol{W}^{[1]}}\frac{\partial g^{[3]}}{\partial \boldsymbol{f}^{[1]}} = \boldsymbol{a}^{[0]}\left(\frac{\partial g^{[3]}}{\partial \boldsymbol{f}^{[1]}}\right)^T$$

Refer slide 8 of “MatrixCalculus.pdf” for the proof

$$f^{[1]}\left(\boldsymbol{a}^{[0]}\right) = \left(W^{[1]}\right)^{\top}\boldsymbol{a}^{[0]}$$

$$= \boldsymbol{a}^{[0]}\left(\frac{\partial \boldsymbol{g}^{[1]}}{\partial \boldsymbol{f}^{[1]}}\frac{\partial \boldsymbol{f}^{[2]}}{\partial \boldsymbol{g}^{[1]}}\frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[2]}}\frac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}}\frac{\partial g_1^{[3]}}{\partial f_1^{[3]}}\right)^T$$

$$\delta^{[3]} = \frac{\partial g_1^{[3]}}{\partial f_1^{[3]}}$$

$$= \boldsymbol{a}^{[0]}\left(\frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[1]}}\frac{\partial \boldsymbol{f}^{[3]}}{\partial \boldsymbol{g}^{[2]}}\boldsymbol{\delta}^{[2]}\right)^T$$

$$\boldsymbol{\delta}^{[2]} = \frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[2]}}\frac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}}\delta^{[3]}$$

$$\boldsymbol{\delta}^{[1]} = \frac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[1]}}\frac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}}\delta^{[2]}$$

$$= \boldsymbol{a}^{[0]}\left(\boldsymbol{\delta}^{[1]}\right)^T$$

Back Propagation: Vector Example

$$\boldsymbol{a}^{[0]} \xrightarrow{W^{[1]}} \boldsymbol{f}^{[1]} \rightarrow \boldsymbol{g}^{[1]} \xrightarrow{W^{[2]}} \boldsymbol{f}^{[2]} \rightarrow \boldsymbol{g}^{[2]} \xrightarrow{W^{[3]}} \boldsymbol{f}^{[3]} \rightarrow \boldsymbol{g}^{[3]} \rightarrow \hat{y}$$

$$\hat{y} = a^{[3]} = \boldsymbol{g}^{[3]}\left(\boldsymbol{f}^{[3]}\left(\boldsymbol{g}^{[2]}\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}\left(\boldsymbol{a}^{[0]}\right)\right)\right)\right)\right)\right)$$

Derivatives	Recursion
$\hat{y}'\left(W^{[3]}\right) = \boldsymbol{a}^{[2]}\left(\delta^{[3]}\right)^T$	$\delta^{[3]} = \dfrac{\partial g_1^{[3]}}{\partial f_1^{[3]}}$
$\hat{y}'\left(W^{[2]}\right) = \boldsymbol{a}^{[1]}\left(\boldsymbol{\delta}^{[2]}\right)^T$	$\boldsymbol{\delta}^{[2]} = \dfrac{\partial \boldsymbol{g}^{[2]}}{\partial \boldsymbol{f}^{[2]}} \dfrac{\partial f_1^{[3]}}{\partial \boldsymbol{g}^{[2]}} \delta^{[3]}$
$\hat{y}'\left(W^{[1]}\right) = \boldsymbol{a}^{[0]}\left(\boldsymbol{\delta}^{[1]}\right)^T$	$\boldsymbol{\delta}^{[1]} = \dfrac{\partial \boldsymbol{g}^{[1]}}{\partial \boldsymbol{f}^{[1]}} \dfrac{\partial \boldsymbol{f}^{[2]}}{\partial \boldsymbol{g}^{[1]}} \boldsymbol{\delta}^{[2]}$

$$\boldsymbol{a}^{[l]} \equiv \boldsymbol{g}^{[l]}\left(\boldsymbol{f}^{[l]}\right)$$

$$\boldsymbol{f}^{[l]} \equiv \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

$$\boldsymbol{\delta}^{[l]} = \frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l]}}$$

$\dfrac{\partial \boldsymbol{g}^{[l]}}{\partial \boldsymbol{f}^{[l]}}$: Diagonal Matrix

Generalizing to the L-layer MLP

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

Gradient relative to $\boldsymbol{W}^{[l]}$

$$\hat{y}'(\boldsymbol{W}^{[l]}) = \frac{\partial \hat{y}}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{W}^{[l]}} = \frac{\partial \boldsymbol{f}^{[l]}}{\partial \boldsymbol{W}^{[l]}} \frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l]}} = \boldsymbol{a}^{[l-1]} \left(\boldsymbol{\delta}^{[l]}\right)^T$$

$$\boxed{\begin{aligned} \boldsymbol{a}^{[l]} &= \boldsymbol{g}^{[l]}(\boldsymbol{f}^{[l]}) \\ \boldsymbol{f}^{[l]} &= \left(W^{[l]}\right)^{\mathsf{T}} \boldsymbol{a}^{[l-1]} \end{aligned}}$$

$$\boldsymbol{\delta}^{[l]} = \frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l]}} = \frac{\partial \boldsymbol{g}^{[l]}}{\partial \boldsymbol{f}^{[l]}} \frac{\partial \boldsymbol{f}^{[l+1]}}{\partial \boldsymbol{g}^{[l]}} \frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l+1]}}$$

$$\boxed{\frac{\partial \boldsymbol{g}^{[l]}}{\partial \boldsymbol{f}^{[l]}} = \boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]})} \quad \boxed{\frac{\partial \boldsymbol{f}^{[l+1]}}{\partial \boldsymbol{g}^{[l]}} = \frac{\partial \boldsymbol{f}^{[l+1]}}{\partial \boldsymbol{a}^{[l]}} = W^{[l+1]}} \quad \boxed{\frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l+1]}} = \boldsymbol{\delta}^{[l+1]}}$$

$$\boxed{\hat{y}'(\boldsymbol{W}^{[l]}) = \boldsymbol{a}^{[l-1]} \left(\boldsymbol{\delta}^{[l]}\right)^T} \quad \boxed{\boldsymbol{\delta}^{[l]} = \boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]}) \ \boldsymbol{W}^{[l+1]} \ \boldsymbol{\delta}^{[l+1]}}$$

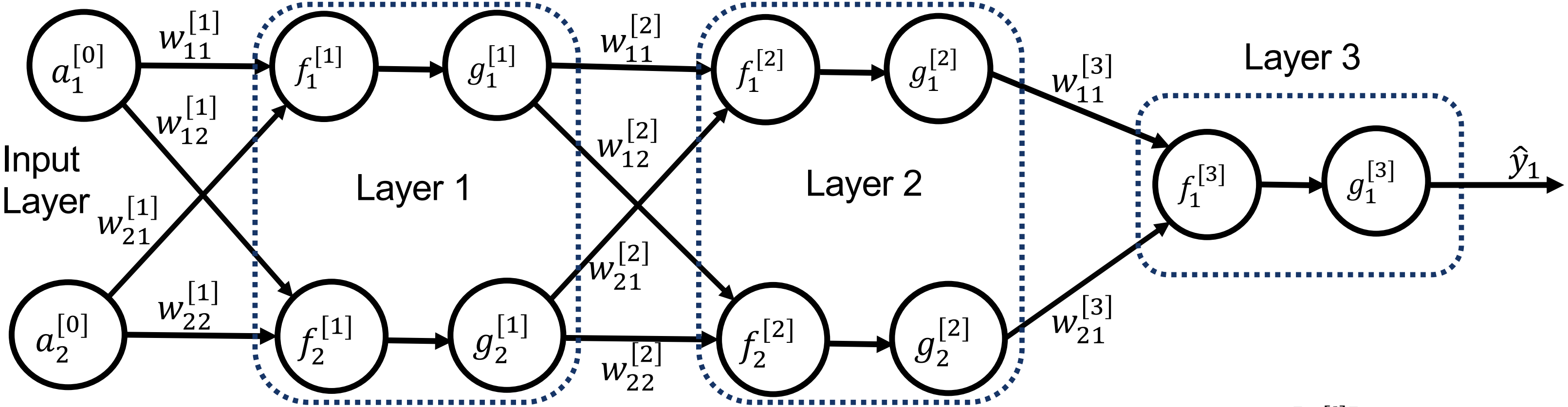
Recursion

A Closer Look at $\boldsymbol{\delta}^{[l]}$:

$$\underset{n^{[l]}\times 1}{\boldsymbol{\delta}^{[l]}} = \underset{\substack{n^{[l]}\times n^{[l]}\\ \text{Diagonal Matrix}}}{\boldsymbol{g}'^{[l]}\left(\boldsymbol{f}^{[l]}\right)} \underbrace{\overset{n^{[l]}\times n^{[l+1]}}{\boldsymbol{W}^{[l+1]}} \; \overset{n^{[l+1]}\times 1}{\boldsymbol{\delta}^{[l+1]}}}_{n^{[l]}\times 1}$$

$n^{[l]}$: # of perceptrons in l-th layer

$n^{[l]} = 2, \forall l$



$$\boldsymbol{g}'^{[l]}\left(\boldsymbol{f}^{[l]}\right) = \frac{\partial \boldsymbol{g}^{[l]}\left(\boldsymbol{f}^{[l]}\right)}{\partial \boldsymbol{f}^{[l]}} = \begin{bmatrix} \frac{\partial g_1^{[l]}\left(f_1^{[l]}\right)}{\partial f_1^{[l]}} & \frac{\partial g_2^{[l]}\left(f_2^{[l]}\right)}{\partial f_1^{[l]}} \\ \frac{\partial g_1^{[l]}\left(f_1^{[l]}\right)}{\partial f_2^{[l]}} & \frac{\partial g_2^{[l]}\left(f_2^{[l]}\right)}{\partial f_2^{[l]}} \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1^{[l]}\left(f_1^{[l]}\right)}{\partial f_1^{[l]}} & 0 \\ 0 & \frac{\partial g_2^{[l]}\left(f_2^{[l]}\right)}{\partial f_2^{[l]}} \end{bmatrix} = diag\left(\begin{bmatrix} \frac{\partial g_1^{[l]}\left(f_1^{[l]}\right)}{\partial f_1^{[l]}} \\ \frac{\partial g_2^{[l]}\left(f_2^{[l]}\right)}{\partial f_2^{[l]}} \end{bmatrix}\right)$$

$$\boldsymbol{g}^{[l]} = \begin{bmatrix} g_1^{[l]} \\ g_2^{[l]} \end{bmatrix}$$

$$\boldsymbol{f}^{[l]} = \begin{bmatrix} f_1^{[l]} \\ f_2^{[l]} \end{bmatrix}$$

Exploiting Diagonal Matrix

- Product of a diagonal matrix and a vector is equivalent to Hadamard product of diagonal elements of the matrix and vector

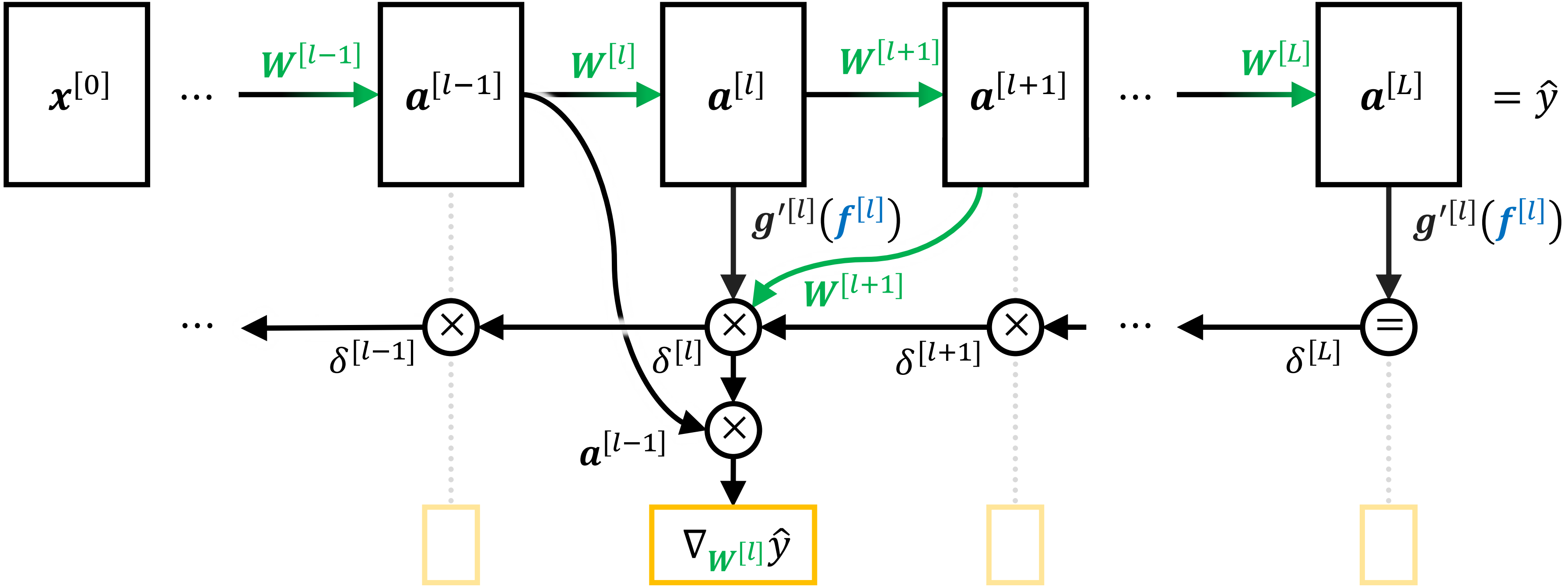
Hadamard Product: elementwise multiplication

$$\begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} xa \\ yb \\ zc \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \circ \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$\underset{n^{[l]}\times 1}{\boldsymbol{\delta}^{[l]}} = \underset{\substack{n^{[l]}\times n^{[l]} \\ \text{Diagonal Matrix}}}{\boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]})} \underbrace{\boldsymbol{W}^{[l+1]} \;\; \boldsymbol{\delta}^{[l+1]}}_{n^{[l]}\times 1} \quad \Longrightarrow \quad \underset{n^{[l]}\times 1}{\boldsymbol{\delta}^{[l]}} = \underset{\substack{n^{[l]}\times 1 \\ \text{Vector with} \\ \text{Diagonal Elements}}}{\boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]})} \circ \underbrace{\boldsymbol{W}^{[l+1]} \;\; \boldsymbol{\delta}^{[l+1]}}_{n^{[l]}\times 1}$$

Hadamard Product: Elementwise multiplication of matrices

Backward Propagation

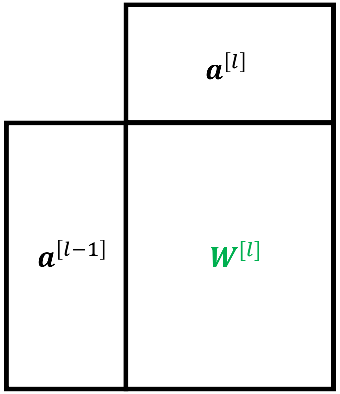


$\hat{y}'(\boldsymbol{W}^{[l]}) = \boldsymbol{a}^{[l-1]}(\boldsymbol{\delta}^{[l]})^{\top}$	$\boldsymbol{\delta}^{[l]} = [\boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]})] \circ (\boldsymbol{W}^{[l+1]}\boldsymbol{\delta}^{[l+1]})$

Matrix multiplication

nRows×nCols $\quad n^{[l-1]}\times n^{[l]} \quad n^{[l-1]}\times 1 \quad n^{[l]}\times 1 \quad n^{[l]}\times n^{[l+1]} \quad n^{[l+1]}\times 1$

$$\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \boldsymbol{a}^{[l-1]} \quad \boldsymbol{\delta}^{[l]} \qquad \boldsymbol{\delta}^{[l]} = \underbrace{\boldsymbol{g}'^{[l]}\left(\boldsymbol{f}^{[l]}\right)}_{n^{[l]}\times n^{[l]}} \quad \underbrace{\boldsymbol{W}^{[l+1]} \quad \boldsymbol{\delta}^{[l+1]}}_{n^{[l]}\times 1}$$



$n^{[l]}\times 1$

Diagonal Matrix

Diagonal elements arranged as a vector

Hadamard product on same-shape matrices, since we are multiplying each dimension independently

$n^{[l-1]}\times 1 \quad 1\times n^{[l]} \qquad n^{[l]}\times 1$

$$\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \boldsymbol{a}^{[l-1]}\left(\boldsymbol{\delta}^{[l]}\right)^{\mathsf{T}} \qquad \boldsymbol{\delta}^{[l]} = \left[\boldsymbol{g}'^{[l]}\left(\boldsymbol{f}^{[l]}\right)\right] \circ \underbrace{\left(\boldsymbol{W}^{[l+1]}\boldsymbol{\delta}^{[l+1]}\right)}_{n^{[l]}\times 1}$$

Gradients in MLP: Intuition

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(\boldsymbol{f}^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(\boldsymbol{f}^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(\boldsymbol{f}^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

$$\boxed{\hat{y}'(\boldsymbol{W}^{[l]}) = \boldsymbol{a}^{[l-1]} \ (\boldsymbol{\delta}^{[l]})^T}$$

$$\boxed{\boldsymbol{\delta}^{[l]} = \boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]}) \ \boldsymbol{W}^{[l+1]} \ \boldsymbol{\delta}^{[l+1]}}$$

Recursion

$$\boxed{\frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l]}} = \boldsymbol{\delta}^{[l]}}$$

$\boldsymbol{\delta}^{[l]}$ indicates how much the output (error) of the NN will change in response to small changes in the weighted sum of inputs to the neurons at the l-th layer

$$\boxed{\frac{\partial \boldsymbol{g}^{[l]}}{\partial \boldsymbol{f}^{[l]}} = \boldsymbol{g}'^{[l]}(\boldsymbol{f}^{[l]})}$$

local gradient

Indicates how much output of a neuron changes in response to a change in weighted sum of its inputs

$$\boxed{\frac{\partial \boldsymbol{f}^{[l+1]}}{\partial \boldsymbol{g}^{[l]}} \frac{\partial \boldsymbol{g}^{[L]}}{\partial \boldsymbol{f}^{[l+1]}} = W^{[l+1]} \boldsymbol{\delta}^{[l+1]}}$$

Backpropagated Error Signal

This product represents the error signal from the next layer $(l+1)$ propagated backward through the weights $W^{[l+1]}$ to l–th layer

Back propagation

Backpropagation **efficiently** computes the **<u>gradient</u>** by

- Avoiding duplicate calculations
- Not computing unnecessary intermediate values,
- Computing the **<u>gradient</u>** of ***each*** **layer**

In particular, the gradient of the weighted input of each layer is calculated from back $[l+1]$ to front $[l]$:

$\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \boldsymbol{a}^{[l-1]}\left(\boldsymbol{\delta}^{[l]}\right)^{\mathsf{T}}$	$\boldsymbol{\delta}^{[l]} = \left[g'^{[l]}\left(f^{[l]}\right)\right] \circ \left(\boldsymbol{W}^{[l+1]}\boldsymbol{\delta}^{[l+1]}\right)$

Tensors

A Tensor is a multi-dimensional "matrix" containing elements of a single data type

- 0-D Tensor $X \Rightarrow$ scalar
- 1-D Tensor $X[i] \Rightarrow$ vector
- 2-D Tensor $X[i, j] \Rightarrow$ matrix
- 3-D Tensor $X[i, j, k]$

Tensors

- Similar to numpy arrays https://rickwierenga.com/blog/machine%20learning/numpy-vs-pytorch-linalg.html
- Hardware wise, tensors can be loaded onto CUDA enabled GPUs for faster computations,
- Software wise, tensors keep track of additional information (Computational Graph) in order to compute the gradients.

Implementation in PyTorch

```
import torch  # Import PyTorch

x = torch.tensor([0.])
w1 = torch.tensor([0.], requires_grad=True)
w2 = torch.tensor([0.], requires_grad=True)

y = w2*torch.sigmoid(w1*x)
y.backward()

print(x,w1,w2)
print(w1.grad)
print(w2.grad)
print(x.grad)
```

```
tensor([0.]) tensor([0.],
requires_grad=True) tensor([0.],
requires_grad=True)
tensor([0.])
tensor([0.5000])
None
```

How do we know that PyTorch is doing the right thing?

```
epsilon = 0.001
def y2(w1,w2,x):
    def sigmoid(x):
        return 1/(1+math.exp(-x))
    return w2*sigmoid(w1*x)

def gradient_w1(w1,w2,x):
    return (y2(w1+epsilon,w2,x)-\
            y2(w1-epsilon,w2,x))/(2*epsilon)
```

$$\hat{y} = w_2\sigma(w_1 x)$$

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

$$\frac{\partial \hat{y}}{\partial w_i} \approx \frac{\hat{y}(w_i+\varepsilon) - \hat{y}(w_i-\varepsilon)}{2\varepsilon}$$

How do we know that PyTorch is doing the right thing?

```
def gradient_w2(w1,w2,x):
    return (y2(w1,w2+epsilon,x)-\
            y2(w1,w2-epsilon,x))/(2*epsilon)

print(gradient_w1(0,0,x)) →0.0
print(gradient_w2(0,0,x)) →0.5
```

What happened to the loss function ε?

Key insight:

$\frac{\partial \varepsilon}{\partial w_i} = \frac{\partial \varepsilon}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_i}$ so once we have all the $\frac{\partial \hat{y}}{\partial w_i}$'s, we can compute any $\frac{\partial \varepsilon}{\partial \boldsymbol{w}}$ for use with gradient descent

Recap: Gradient Descent

1. Decide on some loss function $\mathcal{E}$, which is general some function of $\hat{y} - y$
2. Compute $\dfrac{\partial \varepsilon}{\partial \boldsymbol{w}}$
3. Iterate until convergence:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial \varepsilon}{\partial \boldsymbol{w}} = \frac{\partial \varepsilon}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \boldsymbol{w}}$$

Auto Differentiation for Backprop

- Even with backprop, implementing the gradients is tedious
- Deep learning APIs have automated differentiation.
  - Tensor Flow autodiff
  - PyTorch autograd
  - Implement derivatives of many common functions
  - You just need to implement your layers and neurons; API will handle gradients

Caution

- If you want to implement **custom functions/layers** (not simple weighted sum)
- They need to be **differentiable** to be able to calculate their **gradients**
- Otherwise, backprop cannot update weights accurately

So how do we go about training a Neural Network?

Let's train a NN for $y = x^2$

```
import torch.nn as nn

class MultilayerPerceptron(nn.Module): # 3 layers, with 2
hidden layers of the same size

  def __init__(self, input_size, hidden_size):
    # Call to the __init__ function of the super class
    super(MultilayerPerceptron, self).__init__()

    # Bookkeeping: Saving the initialization parameters
    self.input_size = input_size
    self.hidden_size = hidden_size
```

Let's train a NN for $y = x^2$

```
    # Defining of our layers
    self.linear = nn.Linear(self.input_size, \
                                self.hidden_size)
    self.linear2 = nn.Linear(self.hidden_size, \
                                 self.hidden_size)
    self.linear3 = nn.Linear(self.hidden_size, 1)
    self.relu = nn.ReLU()

def forward(self, x):
    linear = self.linear(x)
    linear2 = self.linear2(self.relu(linear))
    linear3 = self.linear3(self.relu(linear2))
    return linear3
```

Auto-randomized weights. w.o this, all the neurons in the same layer will "learn" the same function/update to the same value.

What's in the model?

```
# Instantiate the model
model = MultilayerPerceptron(1,8)

list(model.named_parameters())
```

```
[('linear.weight',
  Parameter containing:
  tensor([[-8.4808e-01],
          [-1.6513e+00],
          [ 7.3262e-04],
          [ 1.2931e+00],
          [-1.2794e+00],
          [ 6.2609e-01],
          [ 1.0877e+00],
          [ 1.0415e+00]], requires_grad=True)),
 ('linear.bias',
  Parameter containing:
  tensor([-7.2229, -4.7316, -1.6215, -2.8653, -1.5121, -0.3011, -1.2276, -7.6588],
         requires_grad=True)),
 ('linear2.weight',
  Parameter containing:
  tensor([[ 4.6861, -0.4884,  0.3506, -0.3301,  0.3303, -0.5205, -0.5355,  6.3255],
          [-0.5538, -0.5617, -0.3647, -0.1368, -0.5342, -0.7102, -0.8007, -0.3282],
          [ 0.5950,  1.6070, -0.1568,  1.1529,  0.0298,  0.0076,  0.4839,  0.9624],
          [ 0.6179,  1.8320, -0.2905,  1.0212, -0.4870, -0.1435,  0.5969,  0.6689],
          [ 0.3527,  1.6112,  0.0848,  1.5503, -0.8342,  0.3396,  0.8760,  0.8615],
          [-0.5923, -0.6296, -0.6366, -0.4786, -0.5051, -0.5808, -0.8134, -0.6187],
          [ 1.6187,  1.2003,  0.0634,  0.7204,  2.4347,  1.3357,  1.4979,  1.7267],
          [ 1.5565,  1.7537, -0.1973,  1.4202,  0.1672, -0.0336,  0.6651,  0.6998]],
         requires_grad=True)),
......
```

What’s in the model?

……

```
 ('linear2.bias',
  Parameter containing:
  tensor([-1.3254, -0.6541, -5.4646, -6.0203, -4.0982, -0.4350,  0.0093, -6.5684],
         requires_grad=True)),
 ('linear3.weight',
  Parameter containing:
  tensor([[0.7400, 0.4930, 0.6233, 0.5519, 0.7737, 0.4144, 1.4362, 0.9378]],
         requires_grad=True)),
 ('linear3.bias',
  Parameter containing:
  tensor([0.5673], requires_grad=True))]
```

Let's generate the training data

```
# Create the y data
x = 5*torch.randn(100, 1)
```

100 samples of Normally distributed random numbers with mean 0 and variance 1

```
# Add some noise to our goal y to generate our x
# We want out model to predict our original data, albeit
the noise
y = torch.square(x) + torch.randn_like(x)
```

x^2

Same size vector with Normally distributed random numbers with mean 0 and variance 1

Simulate $y = x^2$ with white noise

Set up for training

```
import torch.optim as optim

# Instantiate the model
model = MultilayerPerceptron(1,8)

# Define the optimizer
adam = optim.Adam(model.parameters(), lr=1e-1)

# Define loss using a predefined loss function
loss_function = nn.MSELoss()

# Calculate how our model is doing now
y_pred = model(x)
```

Training Loop

```
n_epoch = 10000
for epoch in range(n_epoch):
  # Set the gradients to 0
  adam.zero_grad()

  # Get the model predictions
  y_pred = model(x)

  # Get the loss
  loss = loss_function(y_pred, y)

  # Print stats
  if epoch%1000==0:
      print(f"Epoch {epoch}: traing loss: {loss}")

  # Compute the gradients
  loss.backward()

  # Take a step to optimize the weights
  adam.step()
```

Backpropagation

```
Epoch 0: traing loss: 0.741550087928772
Epoch 1000: traing loss: 0.6229936480522156
Epoch 2000: traing loss: 1.784149169921875
Epoch 3000: traing loss: 0.6484881043434143
Epoch 4000: traing loss: 0.5825478434562683
Epoch 5000: traing loss: 0.7236675024032593
Epoch 6000: traing loss: 0.5898232460021973
Epoch 7000: traing loss: 0.658348560333252
Epoch 8000: traing loss: 0.5854230523109436
Epoch 9000: traing loss: 0.5815950036048889
```

Let's see how well we do!

```
x2 = 5*torch.randn(5, 1)
print(x2.tolist())

y2 = torch.square(x2)
print(y2.tolist())

y_pred = model(x2)
print(y_pred.tolist())
```

Ground truth with no noise

```
[[5.367788791656494], [0.8833026885986328], [-0.6179562211036682],
[4.864236831665039], [5.316009998321533]]
[[28.813156127929688], [0.7802236676216125], [0.3818698823451996],
[23.660799026489258], [28.25996208190918]]
[[29.595108032226562], [0.8340979814529419], [0.5545129179954529],
[23.755483627319336], [28.972129821777344]]
```

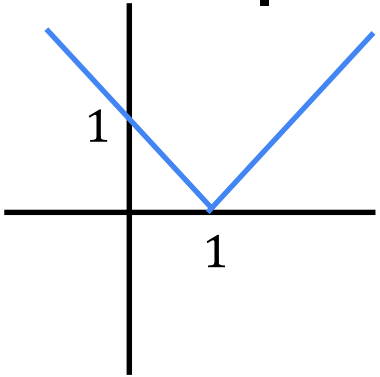
Install PyTorch and try out the examples in this lecture

https://web.stanford.edu/class/cs224n/materials/CS224N_PyTorch_Tutorial.html

Why did we use 8x8 with ReLU?

Stack Overflow suggested! ☺

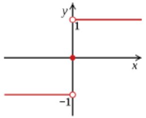
Neural Network for $\hat{y} = |x - 1|$



Which activation function(s)?

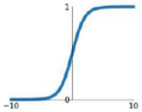
Step

$$\operatorname{sgn}(x) = \begin{cases} +1 & z > 0 \\ -1 & z \leq 0 \end{cases}$$



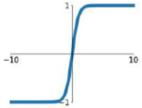
Sigmoid

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



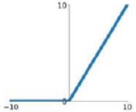
tanh

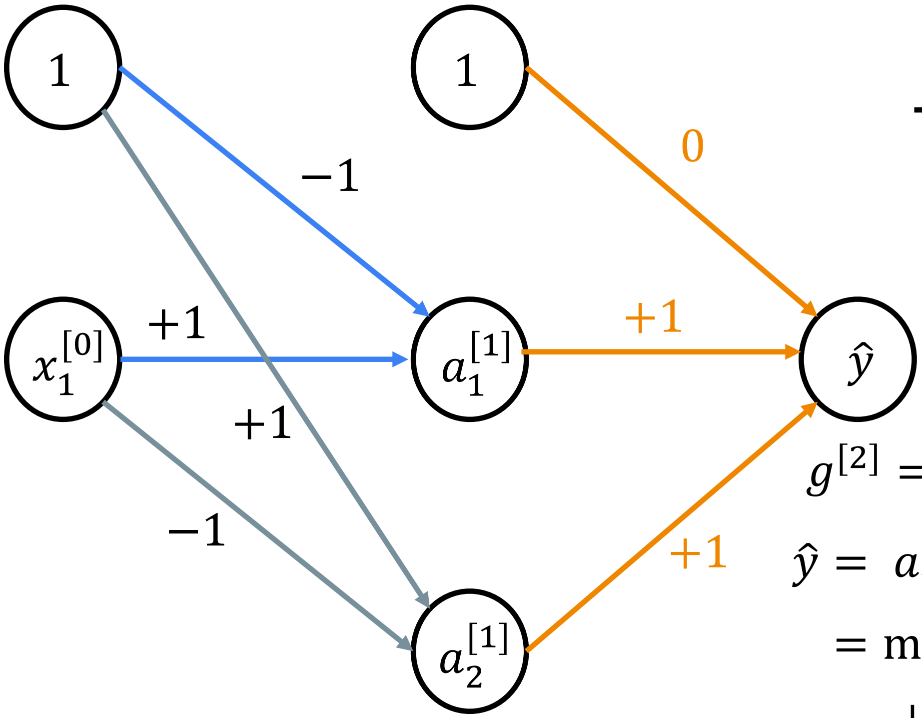
$$\tanh(x)$$



ReLU

$$\max(0, x)$$





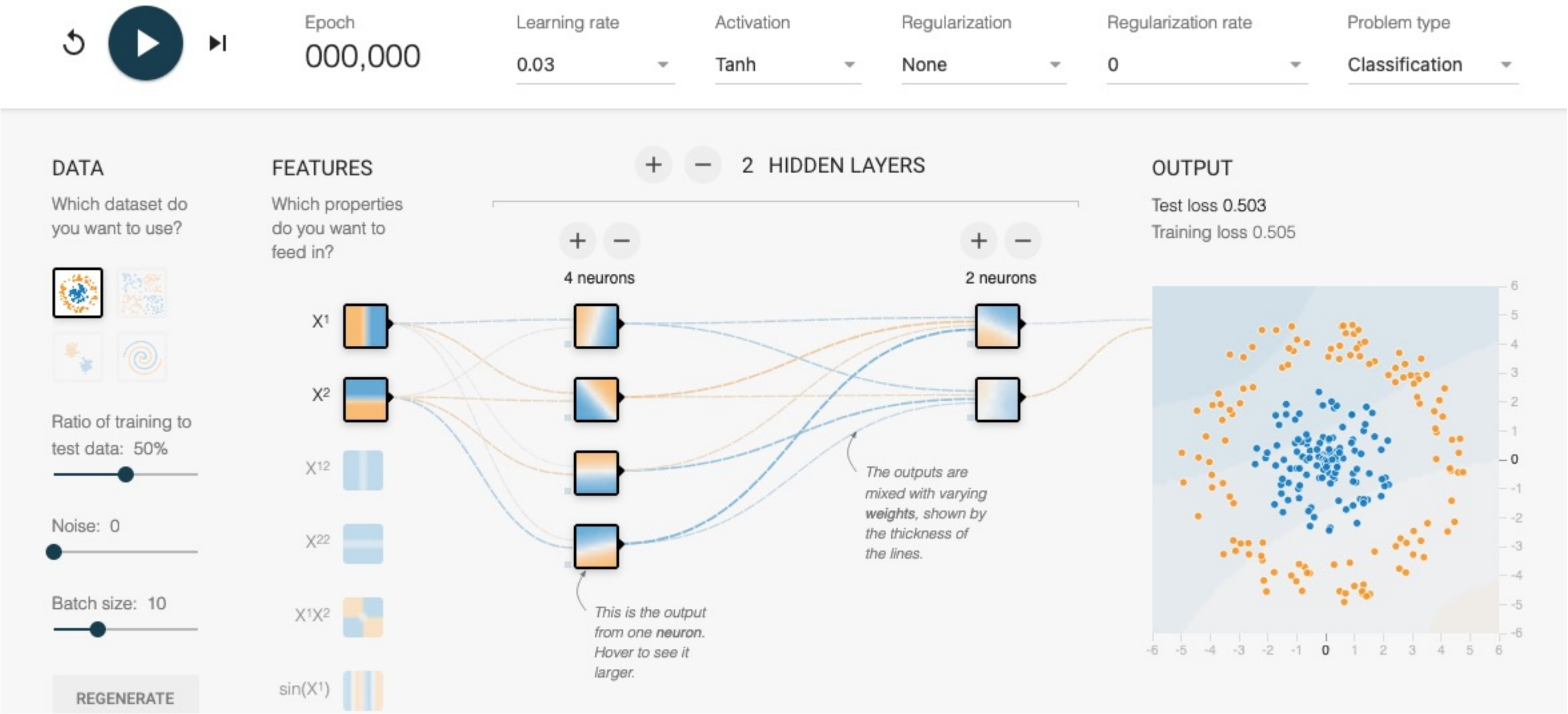
$g^{[2]} = I$

$$\hat{y} = a_1^{[1]} + a_2^{[1]}$$
$$= \max(0,\ x_1^{[0]} - 1)$$
$$+\max(0,\ 1 - x_1^{[0]})$$

$g^{[1]}$= ReLu $\max(0, x)$

Use PyTorch to train a network for $\hat{y} = |x - 1|$

https://playground.tensorflow.org/



Summary

- Back Propagation
- Intro to Pytorch
- Training Neural Network using Gradient Descent