Lecture 15: Learning Parameters of Multilayer Perceptron with Backpropagation

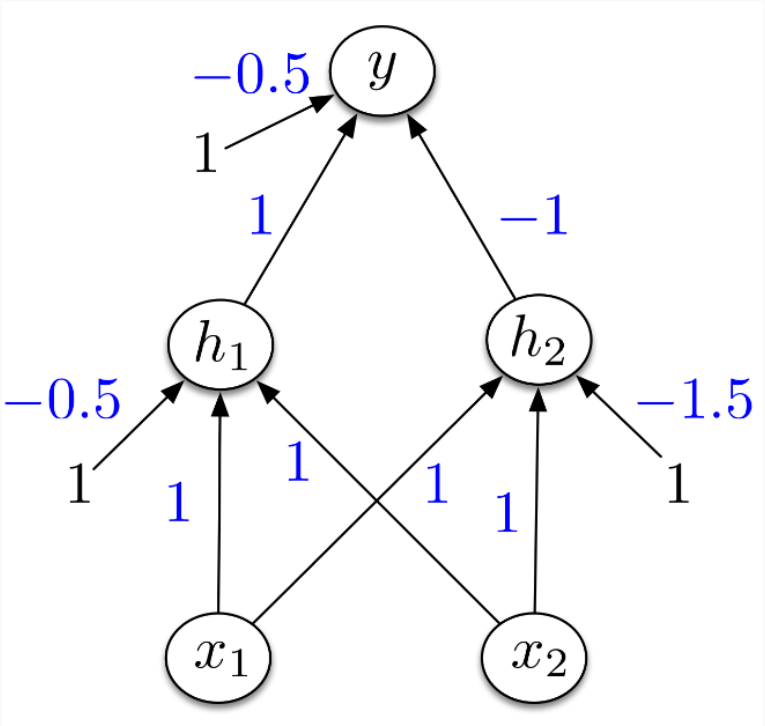
COMP90049

Semester 2, 2022

Joe West, CIS

A Multilayer Perceptron for XOR

How to obtain the optimal parameters?

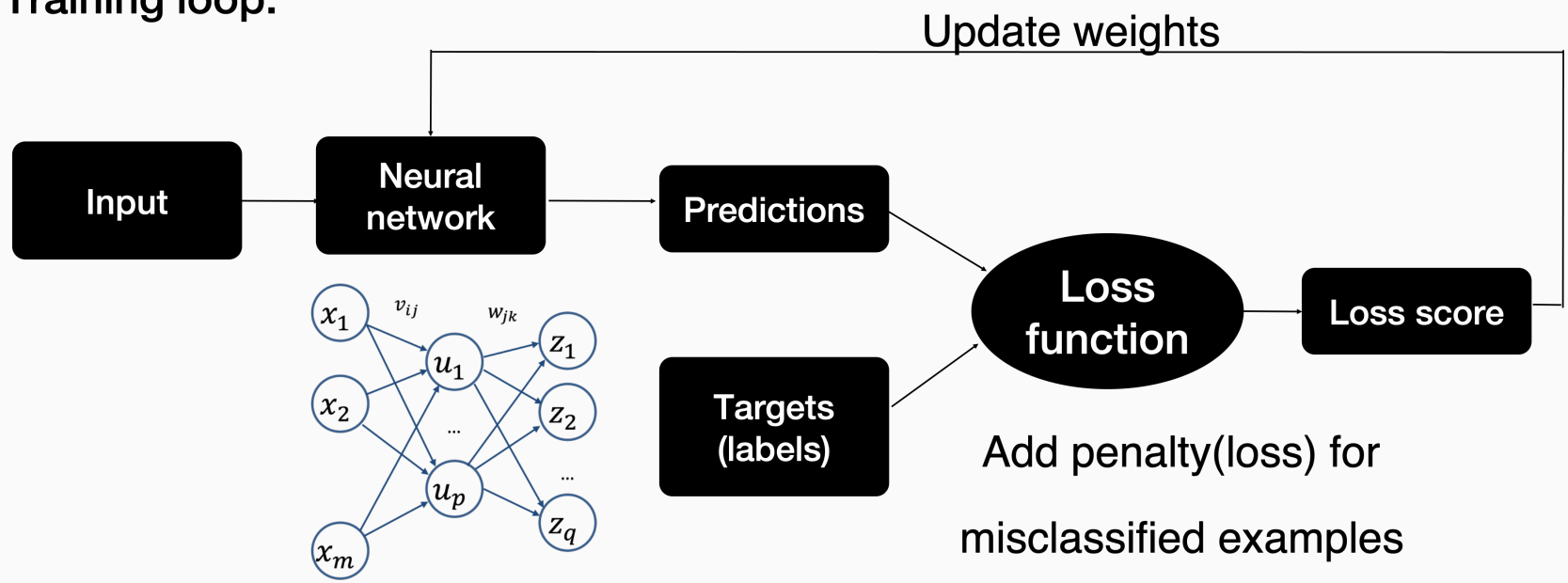


$$\phi(x) = \begin{cases} 1 & \text{if } x >= 0 \\ 0 & \text{if } x < 0 \end{cases} \quad \text{and recall: } h_j^{(l)} = \phi^{(l)}\left(\sum_i \theta_{ij}^{(l)} h_i^{(l-1)} + \theta_{0j}^{(l)} \right)$$

Source: *https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L05%20Multilayer%20Perceptrons.pdf*

“Training”: adjust weights to minimise loss

Training loop:



Loss Functions

- Regression Loss: typically mean-squared error (MSE)

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^{(i)})^2$$

- Binary classification loss

$$\hat{y}_1^{(i)} = p(y^{(i)} = 1|x^{(i)};\theta)$$
$$\mathcal{L} = \sum_i -[y^{(i)}\log(\hat{y}_1^{(i)}) \; + \; (1-y^{(i)})\log(1-\hat{y}_1^{(i)})]$$

- Multiclass classification

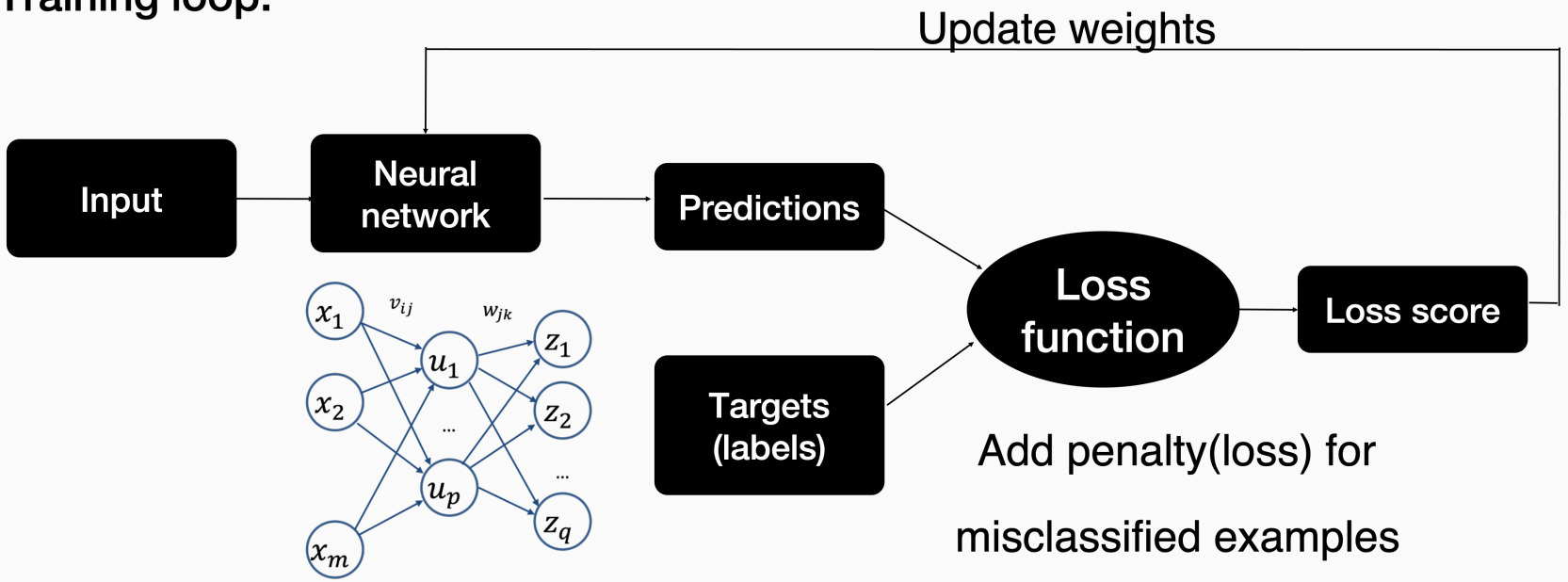
$$\hat{y}_j^{(i)} = p(y^{(i)} = j|x^{(i)};\theta)$$
$$\mathcal{L} = -\sum_i\sum_j y_j^{(i)}\log(\hat{y}_j^{(i)})$$

for j possible labels; $y_j^{(i)} = 1$ if j is the true label for instance i, else 0.

“Training”: adjust weights to minimise loss

How?: Use gradient descend

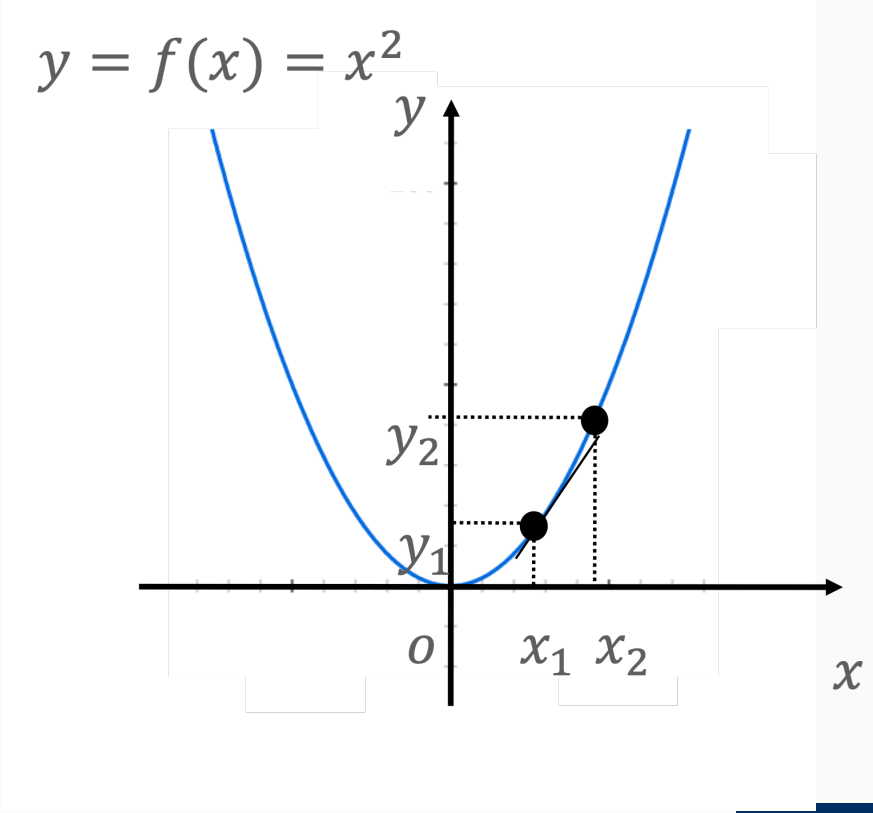
Training loop:



Derivative

$\Delta x \to 0 : f(x_1 + \Delta x) - f(x_1) = a\Delta x$

- derivative a: rate of change (slope) at x_1
- can be treated as a vector from origin along x dim.
- Move x along OPPOSITE direction of the vector: decrease f(x)

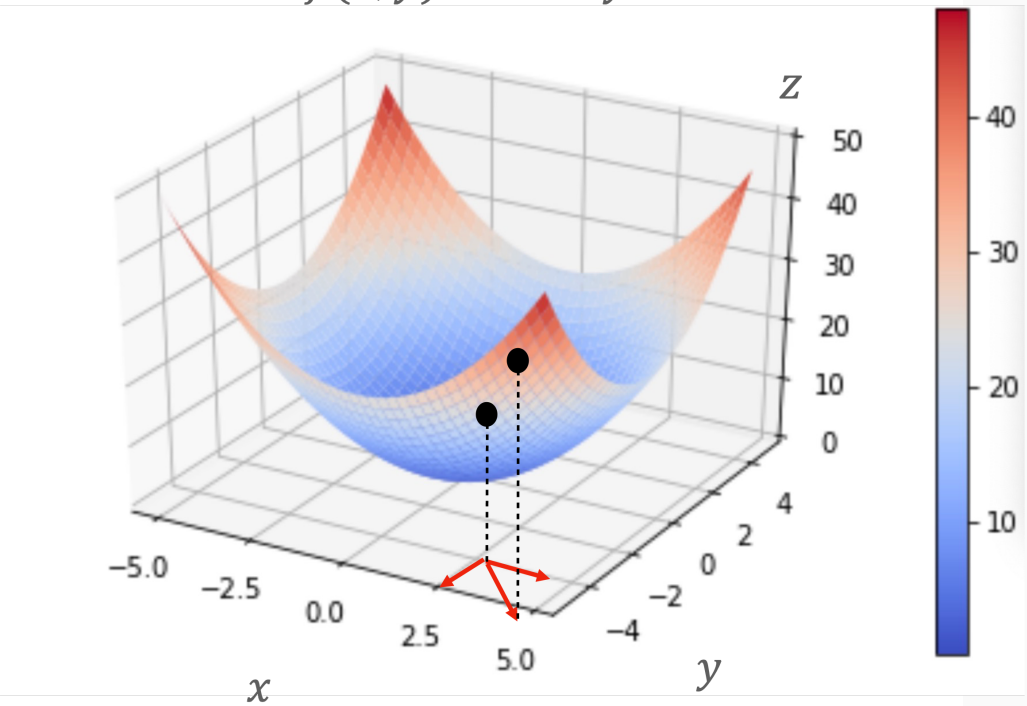


Gradient

$$G = [\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}]$$

Move input (x,y) along OPPOSITE direction of gradient vector: decrease output f(x,y)

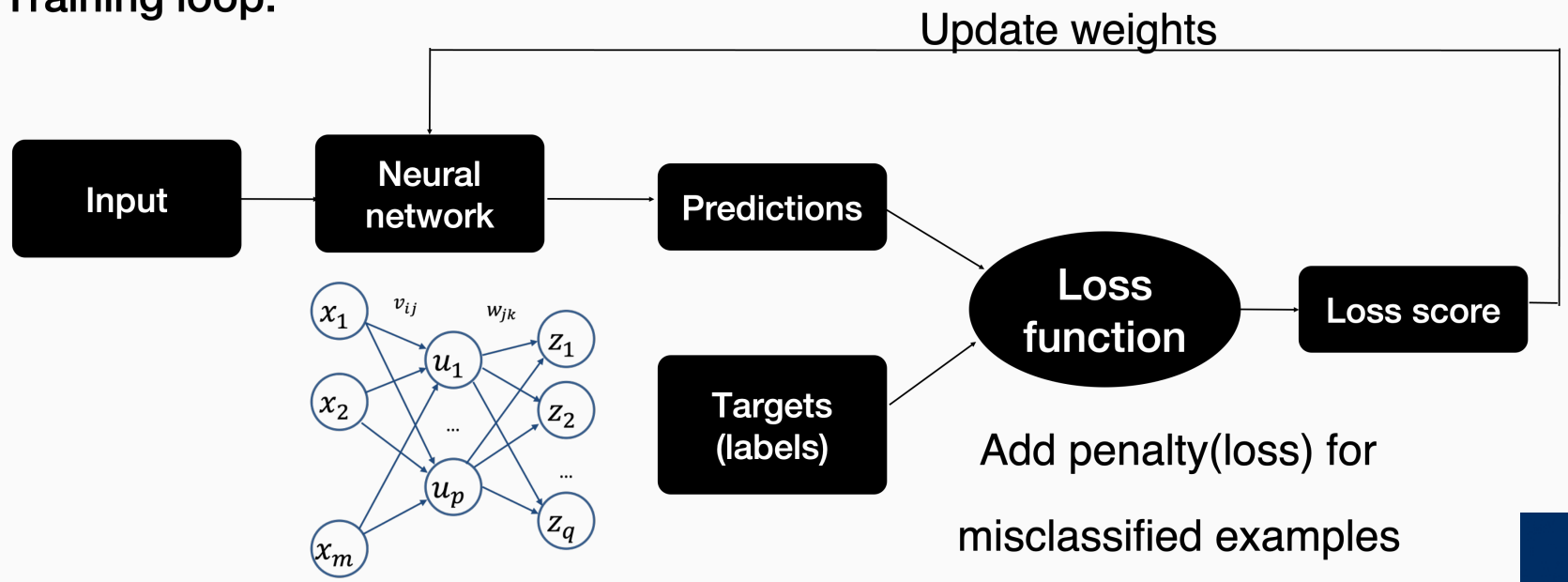
$$z = f(x, y) = x^2 + y^2$$



“Training”: adjust weights to minimise loss

- Loss = function (weights)
- Gradient descend (opposite direction of gradient vector) to minimize loss

Training loop:



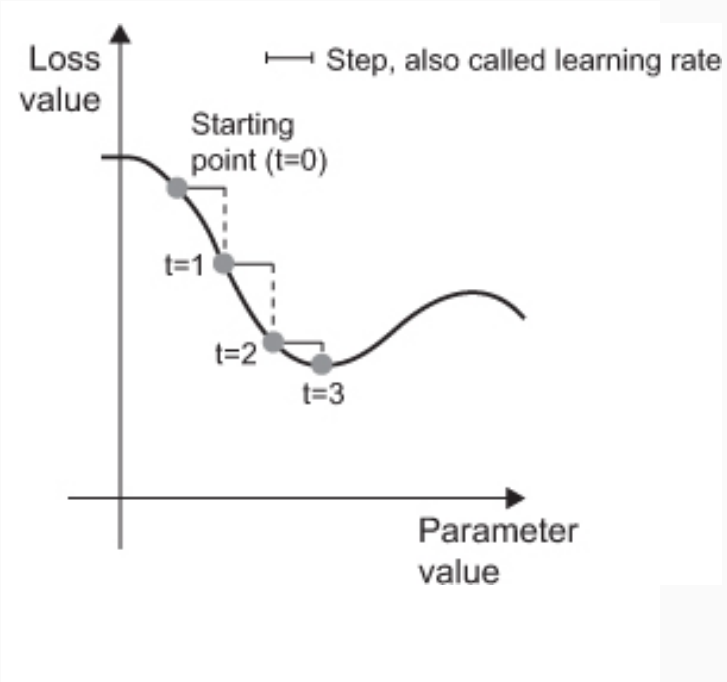
“Training”: adjust weights to minimise loss

To reduce loss, update weights:

$$w_i^{new} = w_i^{old} - \eta * \Delta L(w_i)$$

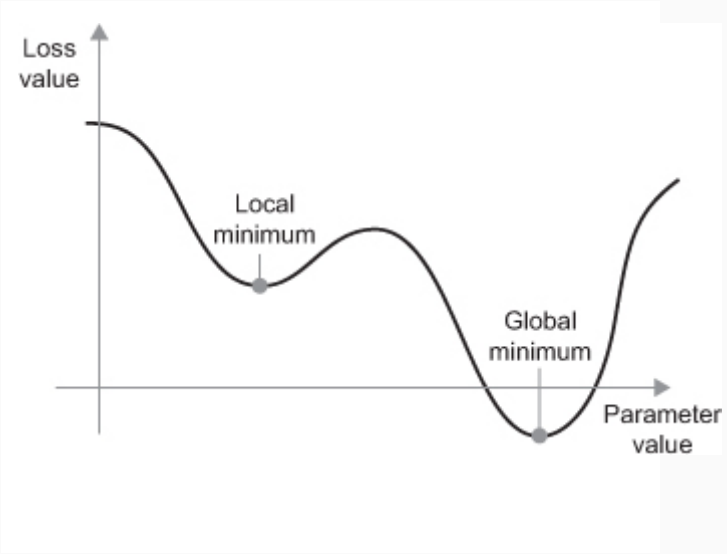
$$\Delta L(w_i) = \frac{\partial L}{\partial w_i}$$

η: learning rate



“Training”: adjust weights to minimise loss

η: learning rate
Small η: local minima
Large η: random location



Gradient Descent Algorithm

- Randomly shuffle/split all training examples in B batches
- Choose initial $\boldsymbol{\theta}^{(1)}$
- For i from 1 to T
- For j from 1 to B
- Do gradient descent update using data from batch j

Iterations over the entire dataset are called *epochs*

- Advantage of such an approach: computational feasibility for large datasets

Stochastic Gradient Descent Algorithm

Choose initial guess $\boldsymbol{w}^{(0)}$, $k = 0$

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

Update*: $\boldsymbol{w}^{(k++)} = \boldsymbol{w}^{(k)} - \eta \boldsymbol{\nabla} L(\boldsymbol{w}^{(k)})$

THE UNIVERSITY OF MELBOURNE

Chaining derivatives: the Backpropagation algorithm

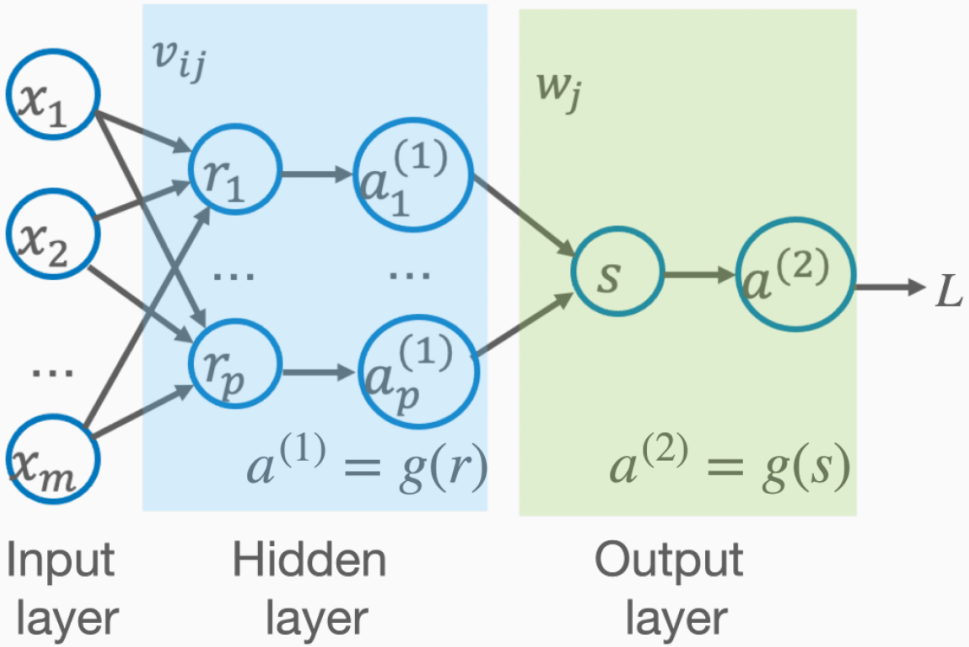
Multilayer perceptron

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

$$s = \sum_{j=0}^{p} a_j^{(1)} w_j$$

$a^{(i)}$: Activation the i^{th} layer

$g()$: Activation function



Chain rule

$$Given\ z = g(u) \quad u = f(x) \Rightarrow \frac{dz}{dx} = \frac{dz}{du}\frac{du}{dx}$$

$$Example : z = \sin(x^2)$$

$$z = sin(u) \qquad \frac{dz}{du} = \cos(u)$$

$$\uparrow \qquad \downarrow$$

$$u = x^2 \qquad \frac{dz}{dx} = \frac{dz}{du}\frac{du}{dx} = 2x\cos(u)$$

Multilayer perceptron: Forward pass

Forward prediction

$$a^{(2)} = g(s)$$

$$\uparrow$$

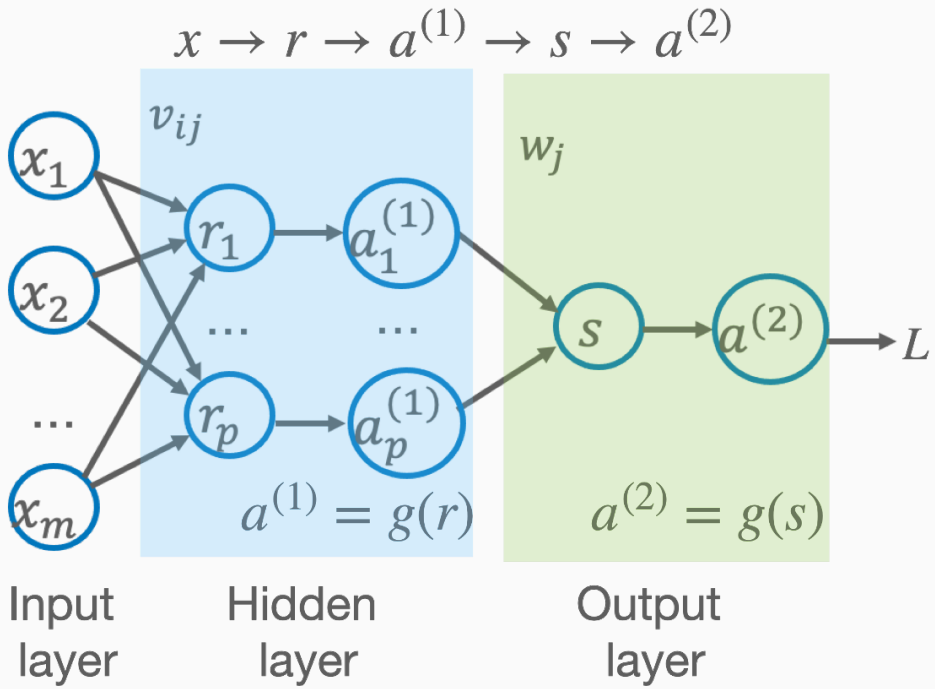
$$s = \sum_{j=0}^{p} w_j a_j^{(1)}$$

$$\uparrow$$

$$a_j^{(1)} = g(r_j)$$

$$\uparrow$$

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

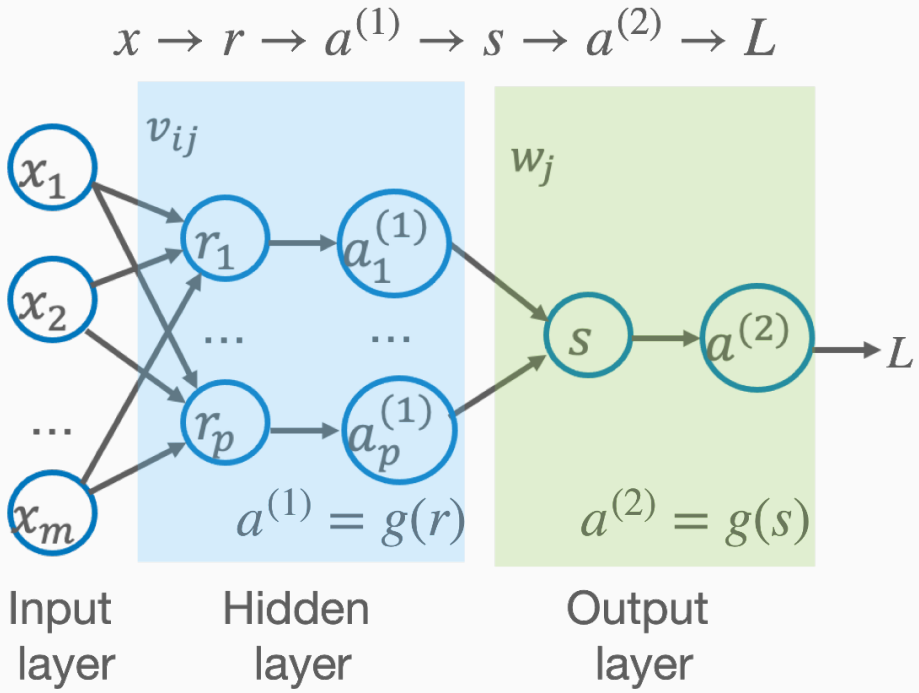


Multilayer perceptron: Loss

Regression task:

$$L = \frac{1}{2}(a^{(2)} - y)^2$$

y : Ground-truth label



Multilayer perceptron:Gradient I

$$\frac{\partial L}{\partial w_j} = (a^{(2)} - y) g' a_j^{(1)}$$

$$= \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s} \frac{\partial s}{\partial w_j}$$

$$L = \frac{1}{2}(a^{(2)} - y)^2$$

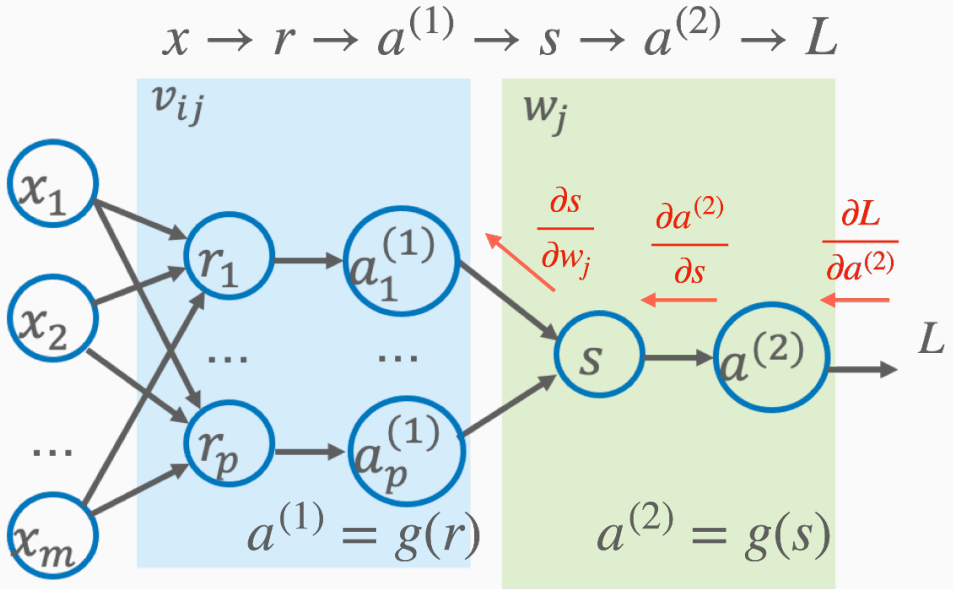
$$\uparrow$$

$$a^{(2)} = g(s)$$

$$\uparrow$$

$$s = \sum_{j=0}^{p} a_j^{(1)} w_j$$

Forward



Multilayer perceptron: Update weights

To reduce loss, update each weight in each layer:

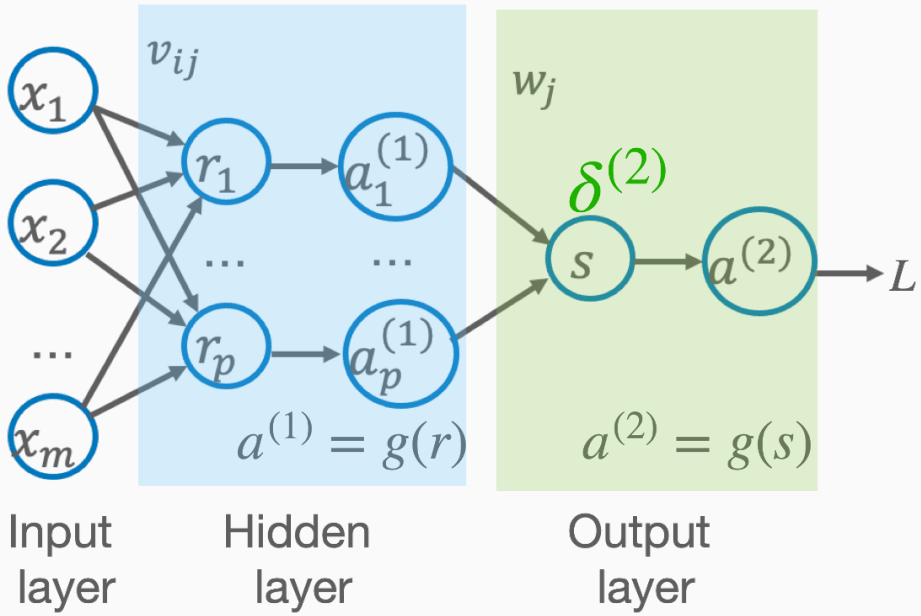
$$\frac{\partial L}{\partial w_j} = (a^{(2)} - y) g' \, a_j^{(1)} = \delta^{(2)} a_j^{(1)}$$

$$w_j^{new} = w_j^{old} - \eta * \frac{\partial L}{\partial w_j}$$

$$= w_j^{old} - \eta \delta^{(2)} a_j^{(1)}$$

$$\delta^{(2)} = \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s} = (a^{(2)} - y) g'$$

(Error at the output layer)



Multilayer perceptron: Gradient II

$$\frac{\partial L}{\partial v_{ij}} = \delta^{(2)}\ w_j\ g'\ x_i$$

$$= \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s} \frac{\partial s}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial r_j} \frac{\partial r_j}{\partial v_{ij}}$$

$$s = \sum_{j=0}^{p} w_j a_j^{(1)}$$

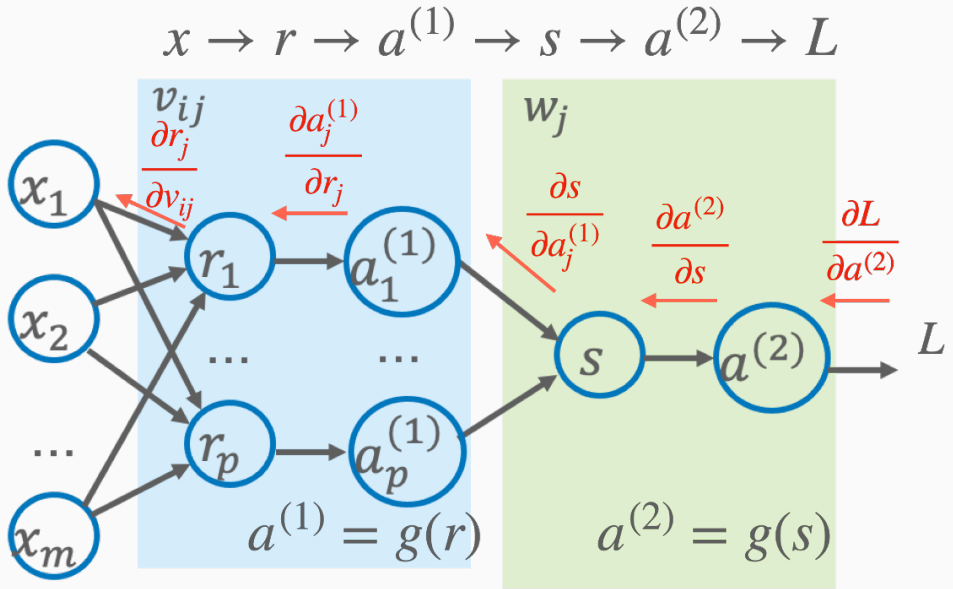
$$\uparrow$$

$$a_j^{(1)} = g(r_j)$$

$$\uparrow$$

$$r_j = \sum_{i=0}^{m} x_i v_{ij}$$

Forward



Multilayer perceptron: Update weights

To reduce loss, update each weight in each layer:

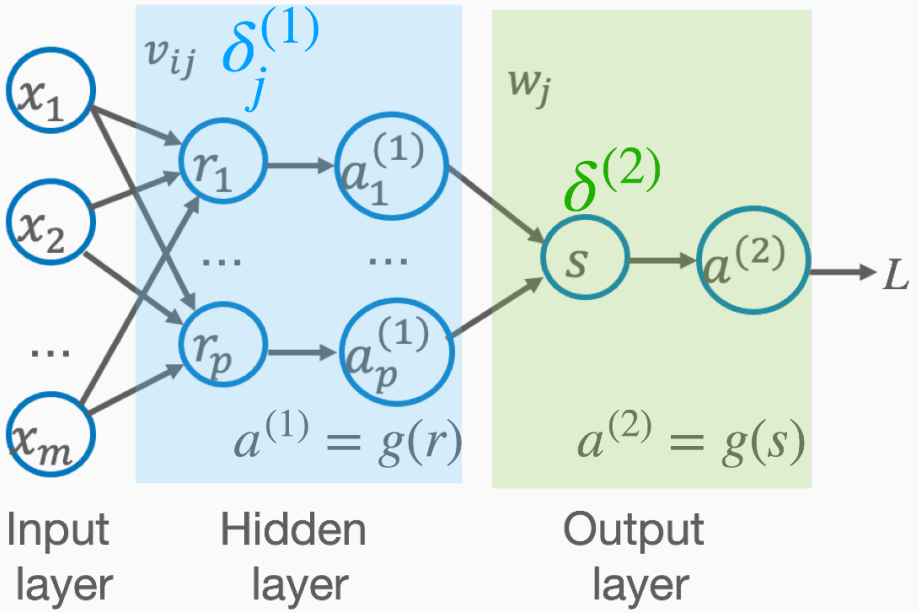
$$\frac{\partial L}{\partial v_{ij}} = \delta^{(2)} w_j g' x_i = \delta_j^{(1)} x_i$$

$$v_{ij}^{new} = v_{ij}^{old} - \eta * \frac{\partial L}{\partial v_{ij}}$$

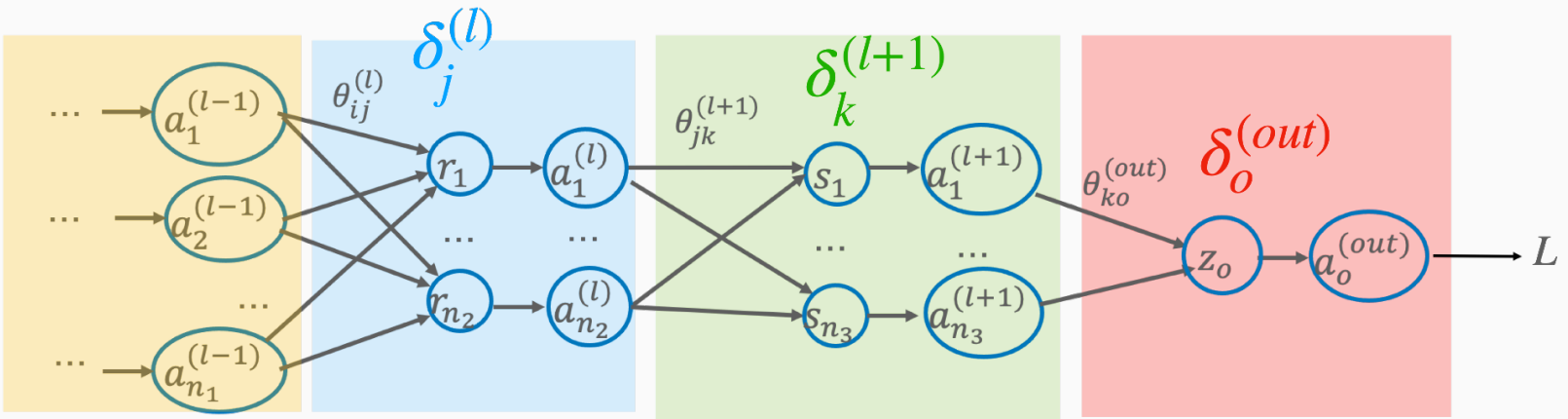
$$= v_{ij}^{old} - \eta \delta_j^{(1)} x_i$$

$$\delta^{(2)} = \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s} = (a^{(2)} - y) g'$$

$$\delta_j^{(1)} = \delta^{(2)} w_j g' \quad \text{(error backpropogation)}$$



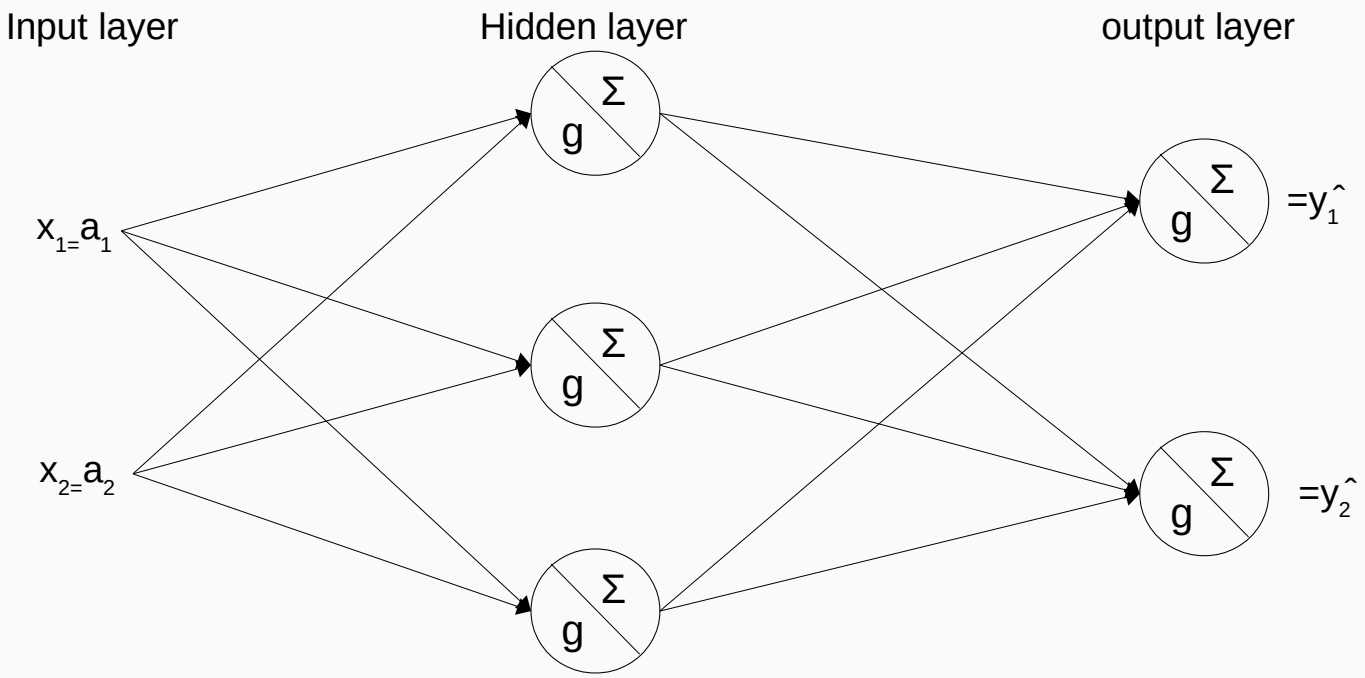
Backpropagation: error of hidden layer - Multiple Layers



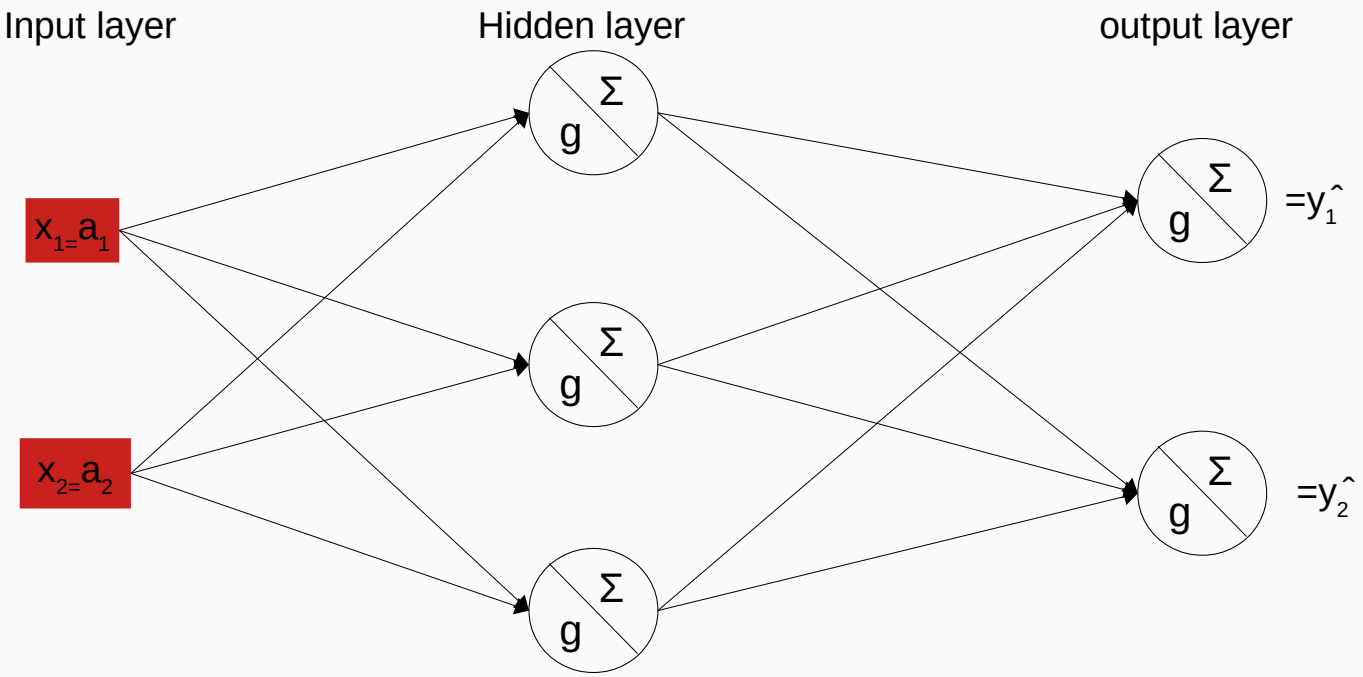
$$\delta_j^{(l)} = \sum_{k=1}^{n_3} \delta_k^{(l+1)} \theta_{jk}^{(l+1)} g' \qquad \delta_k^{(l+1)} = \sum_{o}^{1} \delta_o^{(out)} \theta_{ko}^{(out)} g' \qquad \delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} \frac{\partial a^{(out)_o}}{\partial z_o} = \frac{\partial L}{\partial a_o^{(out)}} g'$$

$$\theta_{ij}^{(l_{new})} = \theta_{ij}^{(l_{old})} - \eta \delta_j^{(l)} a_i^{(l-1)} \qquad \theta_{jk}^{(l+1_{new})} = \theta_{jk}^{(l+1_{old})} - \eta \delta_k^{(l+1)} a_j^{(l)} \qquad \theta_{ko}^{(out_{new})} = \theta_{ko}^{(out_{old})} - \eta \delta_o^{(out)} a_k^{(l+1)}$$

Backpropagation: Demo

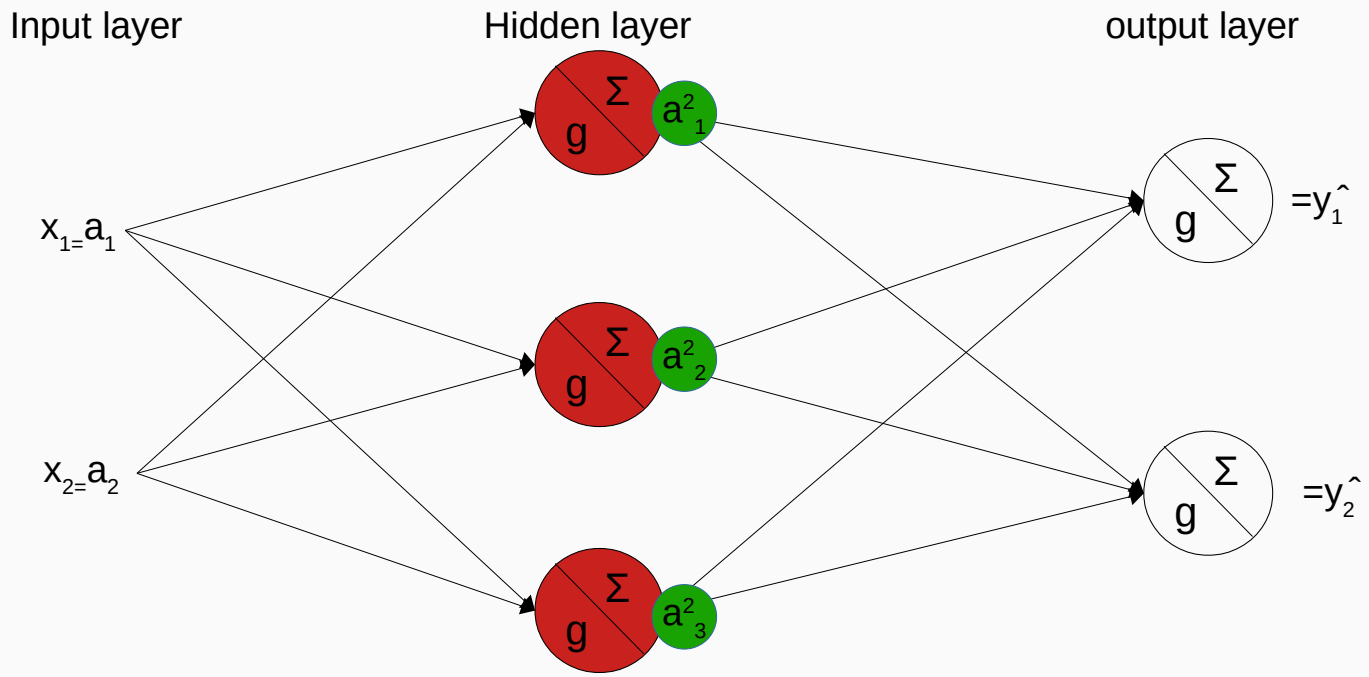


Backpropagation: Demo



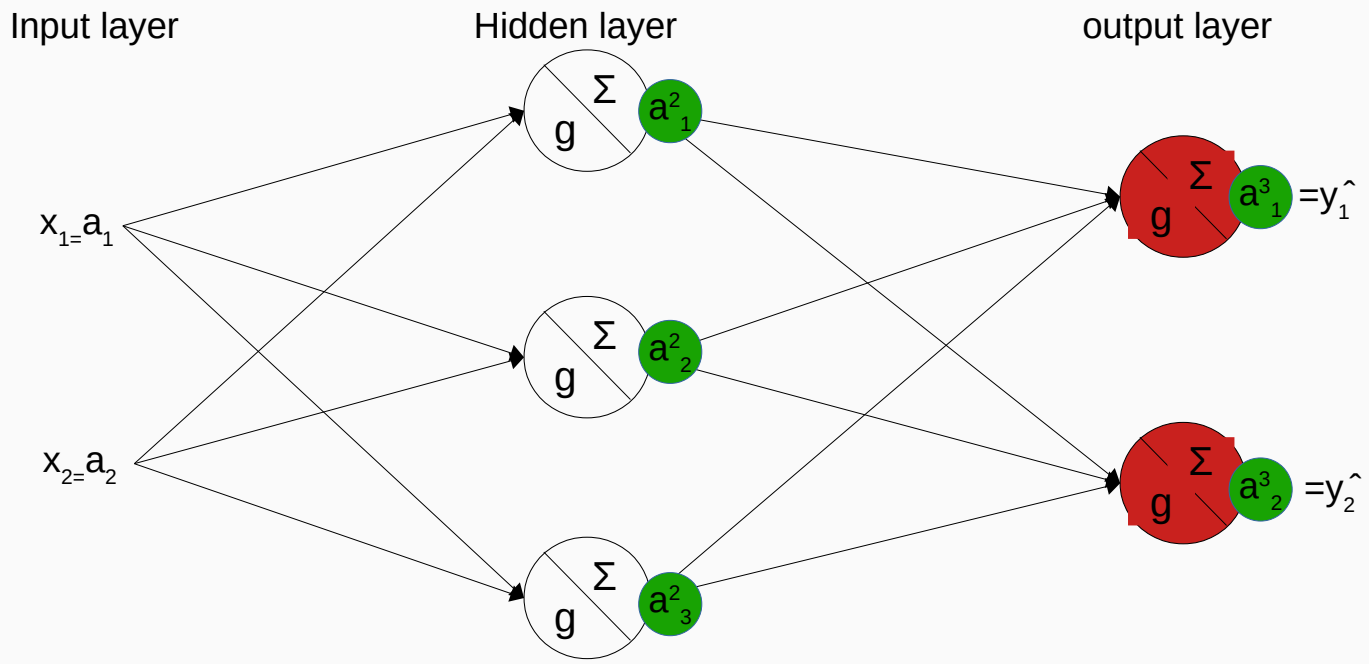
- Receive input

Backpropagation: Demo



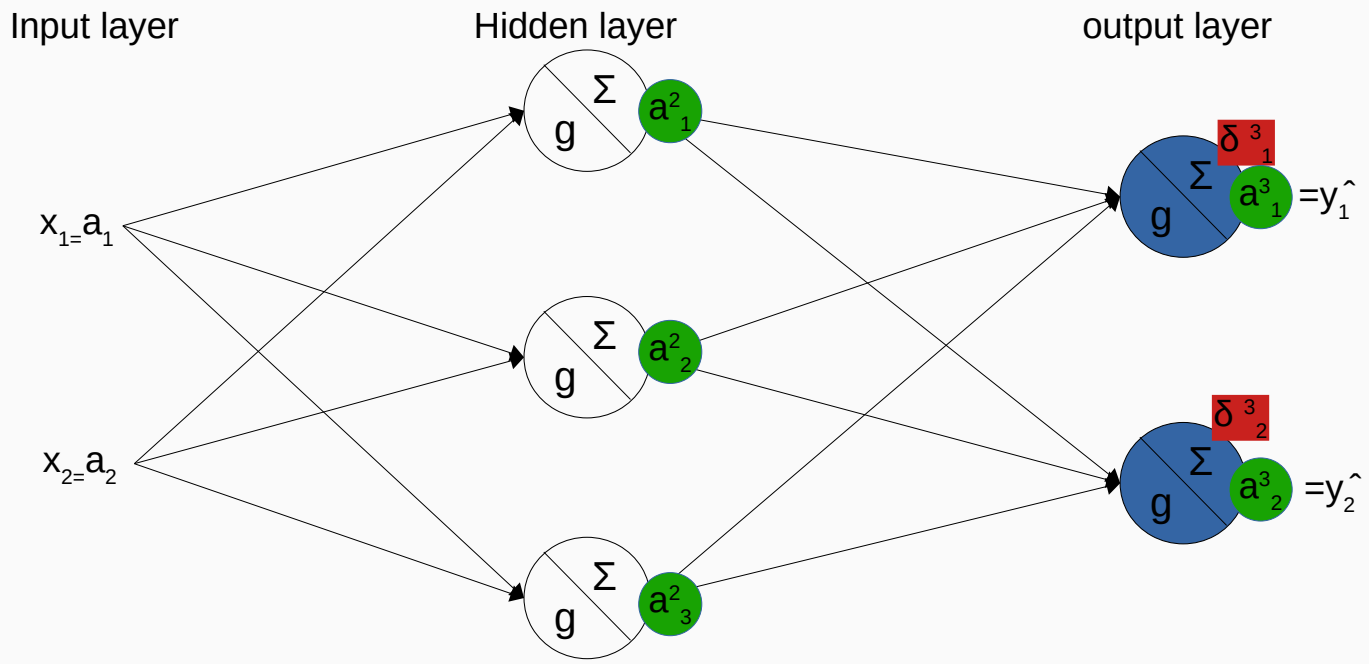
- Receive input
- Forward pass: propagate activations through the network

Backpropagation: Demo



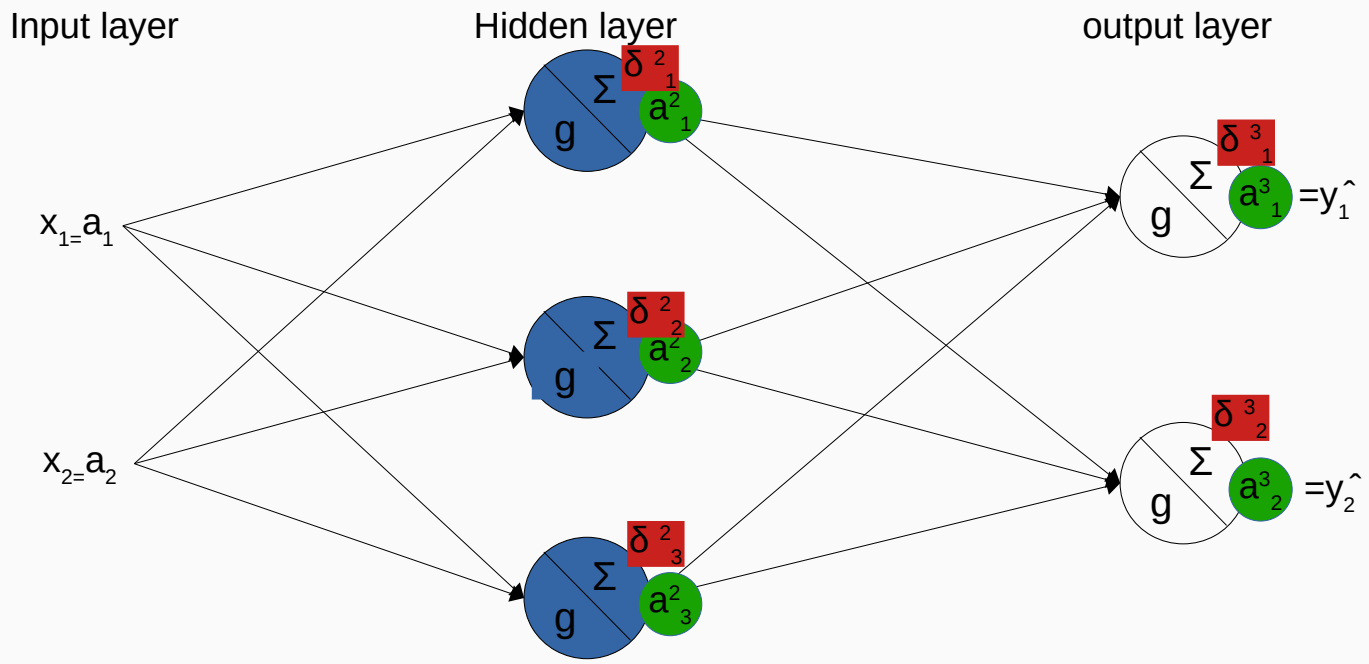
- Receive input
- Forward pass: propagate activations through the network

Backpropagation: Demo



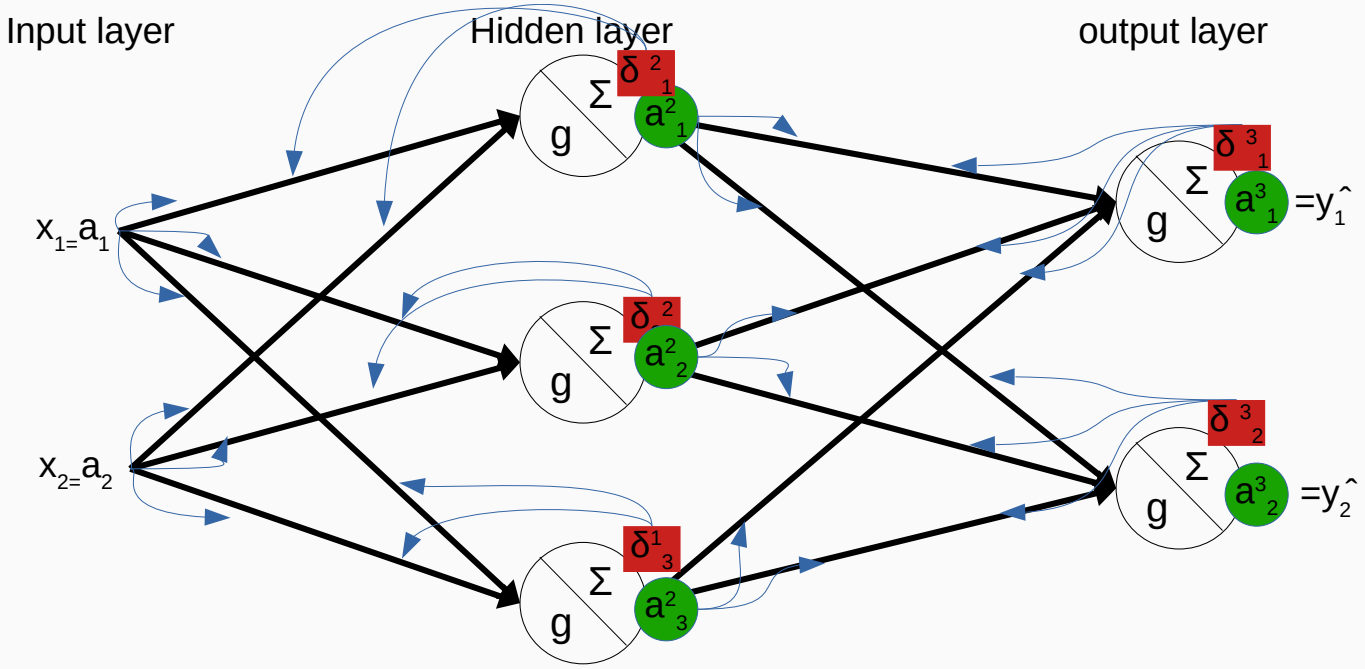
- Receive input
- Forward pass: propagate activations through the network
- Compute Error of output layer: $\delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} g'$

Backpropagation: Demo



- Receive input
- Forward pass: propagate activations through the network
- Compute Error of output layer: $\delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} g'$
- Backward pass: propagate error terms through the network
  $\delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} \theta_{jk}^{(l+1)} g'$

Backpropagation: Demo



- Receive input
- Forward pass: propagate activations through the network
- Compute Error of output layer: $\delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} g'$
- Backward pass: propagate error terms through the network $\delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} \theta_{jk}^{(l+1)} g'$
- Calculate $\frac{\partial L}{\partial \theta_{ij}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$ for all $\theta_{ij}^{(l)}$
- Update weights $\theta_{ij}^{(l)} \leftarrow \theta_{ij}^{(l)} - \eta \frac{\partial L}{\partial \theta_{ij}^{(l)}}$

Backpropagation Algorithm

Design your neural network

Initialize parameters θ

repeat

for training instance x_i **do**

1. **Forward pass** the instance through the network, compute activations, determine output
2. Compute the **error** $\delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} g'$
3. Propagate error **back** through the network, and compute for all weights between nodes ij in all layers l

$$\delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} \theta_{jk}^{(l+1)} g'$$

4. Update **all** parameters **at once**

$$\theta_{ij}^{(l)} \leftarrow \theta_{ij}^{(l)} - \eta \frac{\partial L}{\partial \theta_{ij}^{(l)}} = \theta_{ij}^{(l)} - \eta \delta_j^{(l)} a_i^{(l-1)}$$

until stopping criteria reached.

Summary

After this lecture, you be able to understand

- How to use Gradient Descent to optimize the parameters for neural network
- How Backpropagation allows us to efficiently compute the gradients of all weights wrt. the error in Multilayer perceptron