

Lecture 13: Learning Parameters of Multilayer Perceptron with Backpropagation

COMP90049

Semester 2, 2021

QiuHong Ke, CIS

©2021 The University of Melbourne



Last lecture

- From perceptrons to neural networks
- multilayer perceptron
- input layer, hidden layer, output layer
- features and limitations

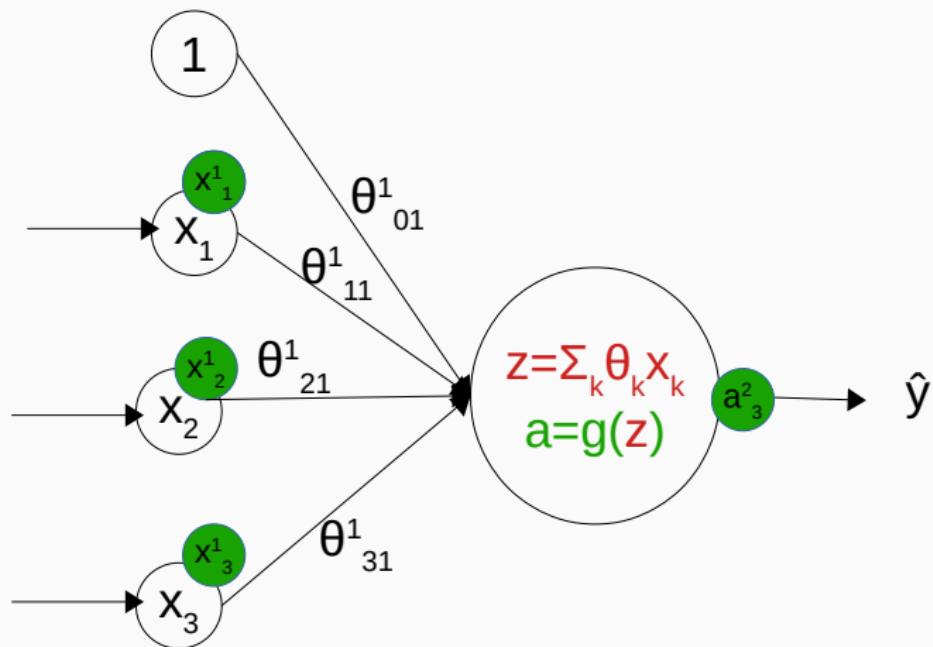
Today

- Training a neural network
- Chaining derivatives: the Backpropagation algorithm

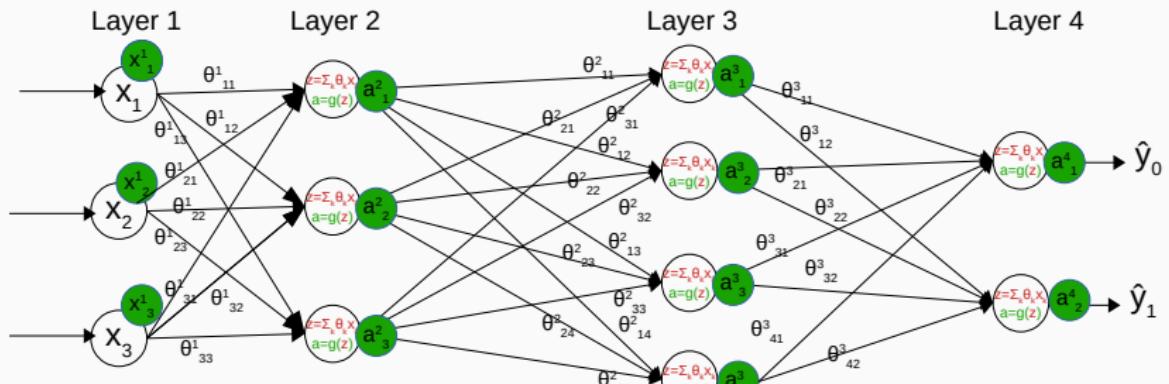


Training a neural network

Recap: perceptron



Recap: Multilayer perceptron

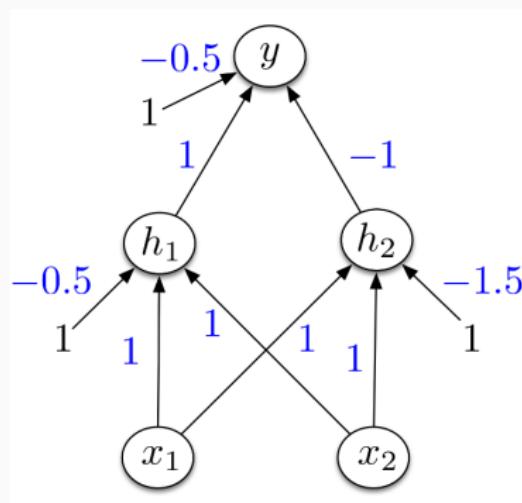


$$a_j^{(l)} = g\left(\sum_i \theta_{ij}^{(l)} a_i^{(l-1)} + \theta_{0j}^{(l)}\right)$$

- non-linearly separable data
- Hidden layers
- Non-linear activation functions

A Multilayer Perceptron for XOR

How to obtain the optimal parameters?



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

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



How to train your dragon Network?

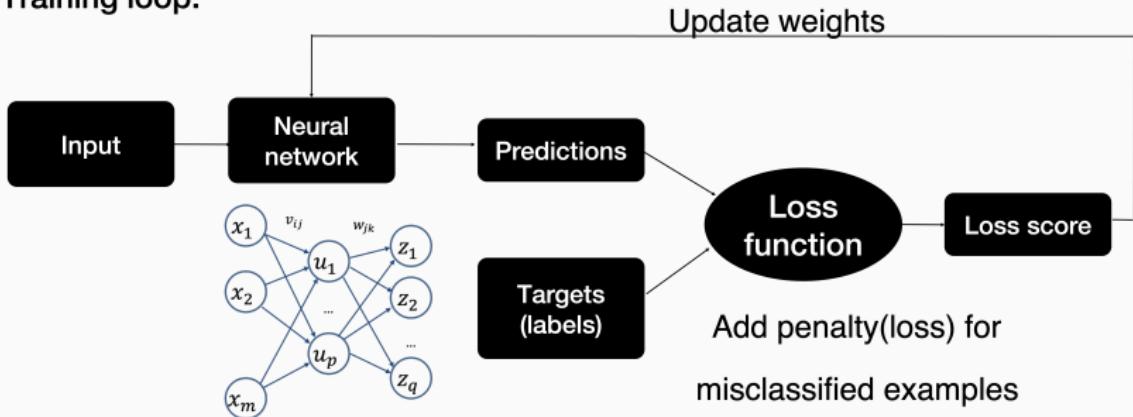


Adapted from Movie Poster from
Flickr user jdxyw (CC BY-SA 2.0)



“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

$$\hat{y}_j^{(i)} = p(y^{(i)} = j | x^{(i)}; \theta)$$

$$\mathcal{L} = - \sum_i \sum_j y_j^{(i)} \log(\hat{y}_j^{(i)})$$

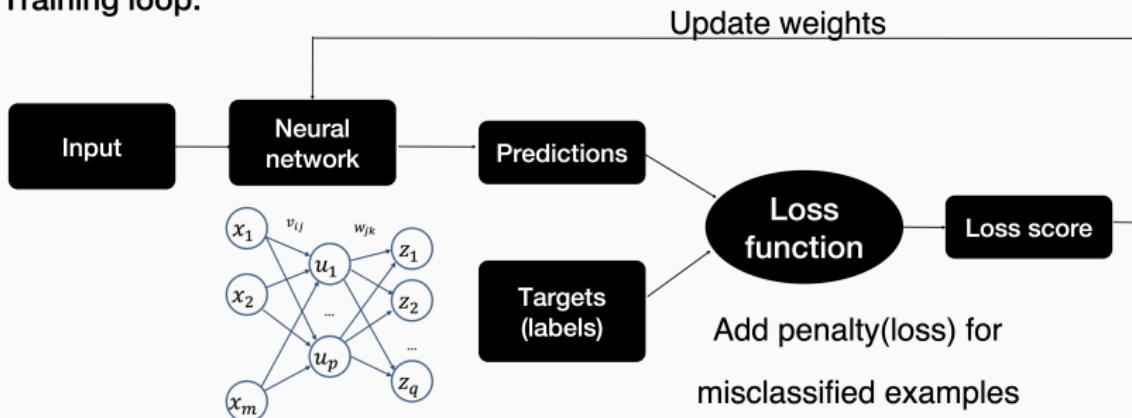
for j possible labels; $y_j^{(i)} = 1$ if j is the true label for instance i , else 0.



“Training”: adjust weights to minimise loss

How?: Use gradient descend

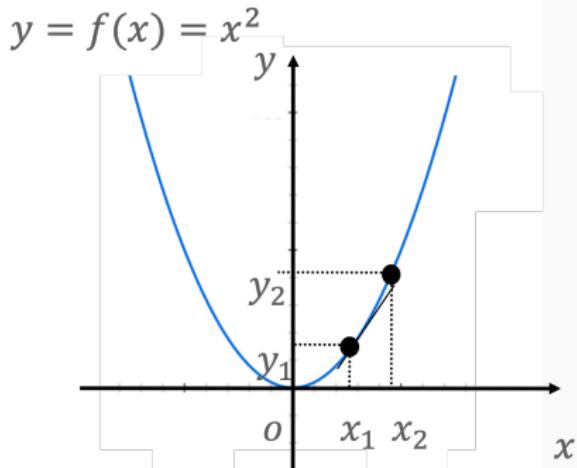
Training loop:



Derivative

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

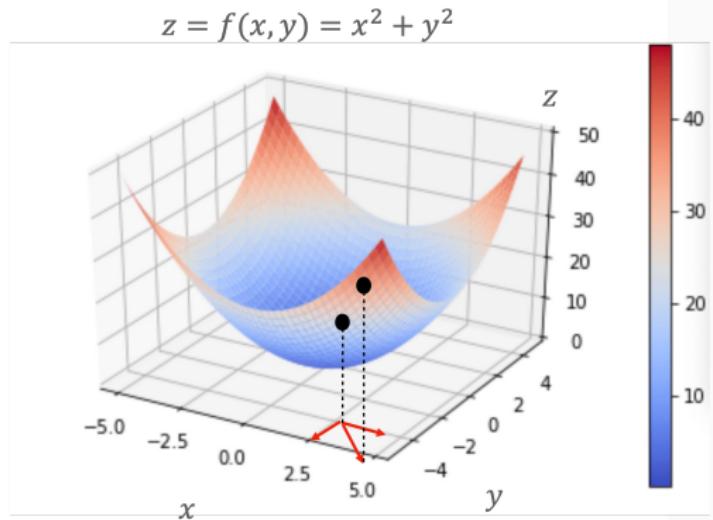
- derivative a : rate of change (slope) at x_1
- can be treated as a vector from origin along x dim.
- Move x along OPPOSITE direction of the vector: decrease $f(x)$



Gradient

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

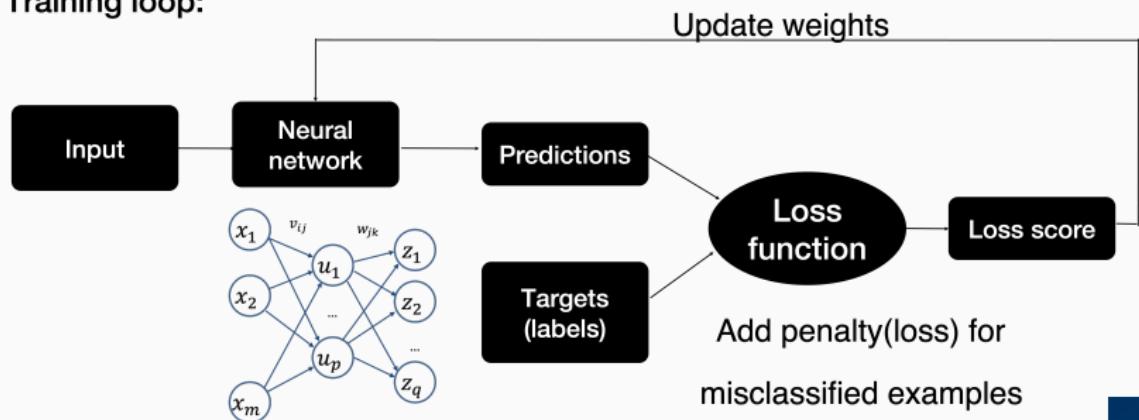
Move input (x,y) along
OPPOSITE direction of
gradient vector:
decrease output $f(x,y)$



“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

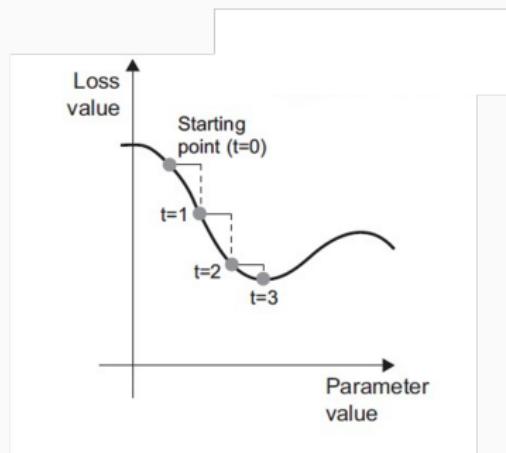


Figure 2.11 in Deep learning with python by Francois Chollet



“Training”: adjust weights to minimise loss

η : learning rate

Small η : local optimal value

Large η : random location

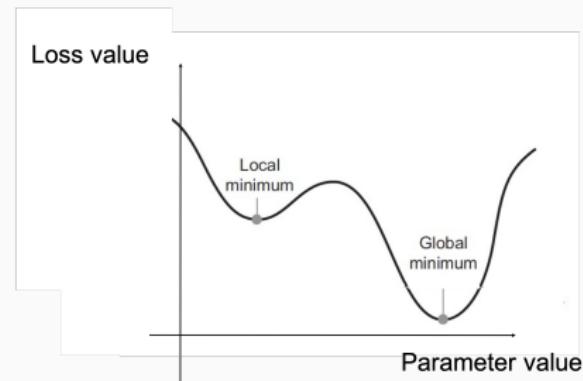


Figure 2.13 in Deep learning with python by Francois Chollet



Gradient Descent Algorithm

- Randomly shuffle/split all training examples in B batches
- Choose initial $\theta^{(1)}$
- For i from 1 to T
- For j from 1 to B
- Do gradient descent update using data from batch j
- Advantage of such an approach: computational feasibility for large datasets

Iterations over the entire dataset are called *epochs*



Stochastic Gradient Descent Algorithm

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

For i from 1 to T (epochs)

 For j from 1 to N (training examples)

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

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



Chaining derivatives: the Backpropagation algorithm

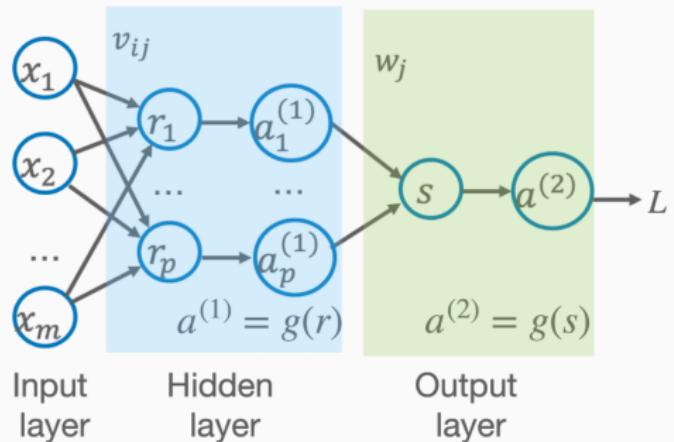
Multilayer perceptron

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

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

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

$g()$: Activation function



Chain rule

Given $z = g(u)$ $u = f(x)$



$$\frac{dz}{dx} = \frac{dz}{du} \frac{du}{dx}$$

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

$$z = \sin(u)$$



$$u = x^2$$

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



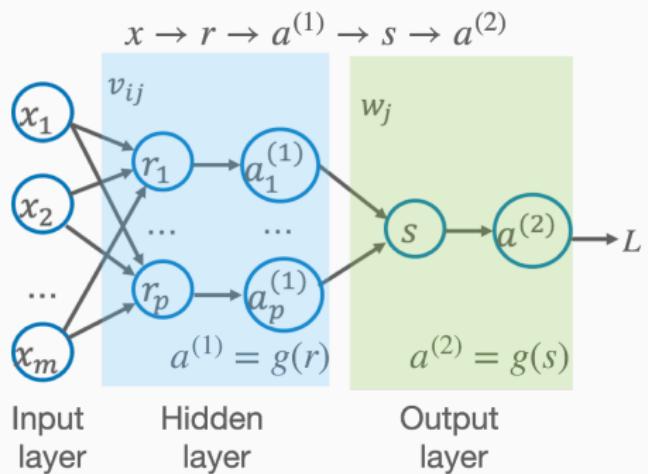
$$\frac{dz}{dx} = \frac{dz}{du} \frac{du}{dx} = 2x \cos(u)$$



Multilayer perceptron: Forward pass

Forward prediction

$$\begin{aligned} a^{(2)} &= g(s) \\ s &= \sum_{j=0}^p w_j a_j^{(1)} \\ a_j^{(1)} &= g(r_j) \\ r_j &= \sum_{i=0}^m x_i v_{ij} \end{aligned}$$



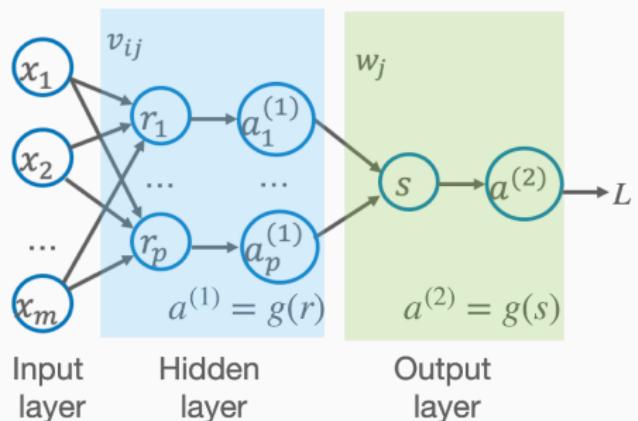
Multilayer perceptron: Loss

Regression task:

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

y : Ground-truth label

$$x \rightarrow r \rightarrow a^{(1)} \rightarrow s \rightarrow a^{(2)} \rightarrow L$$



Multilayer perceptron: Gradient I

$$(a^{(2)} - y)$$

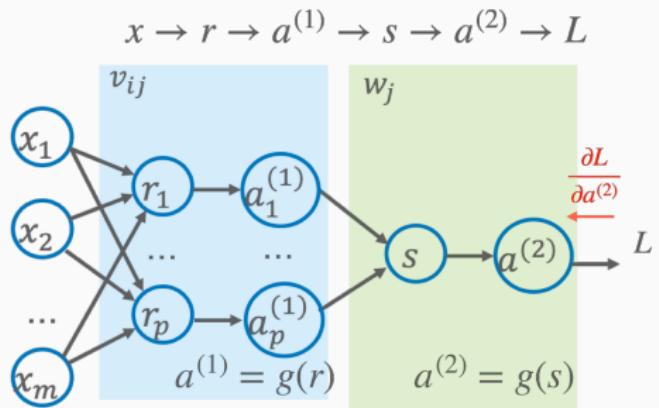
$$\frac{\partial L}{\partial a^{(2)}}$$

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

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

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

Forward



Multilayer perceptron: Gradient I

$$(a^{(2)} - y)g'$$

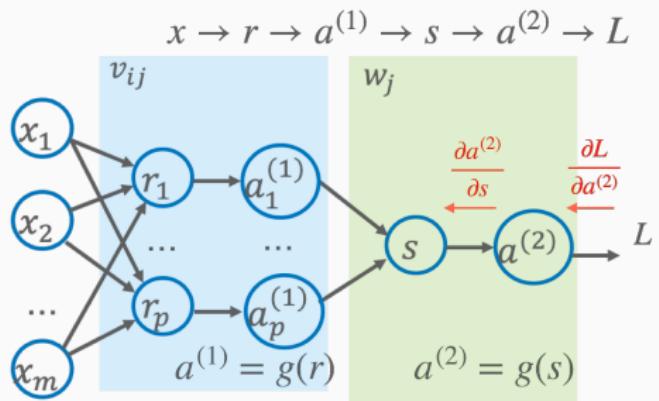
$$\frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s}$$

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

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

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

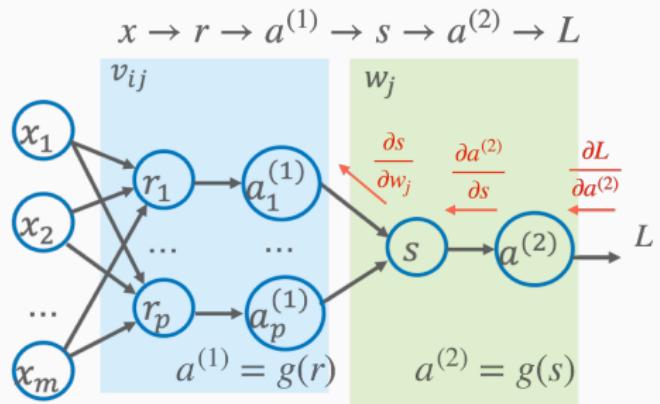
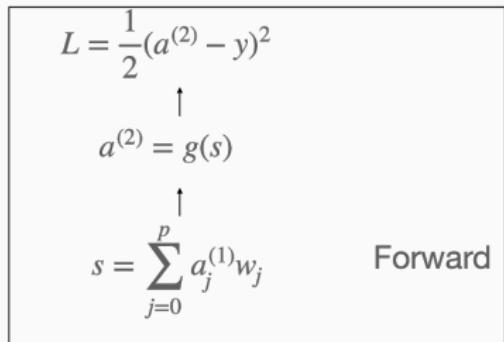
Forward



THE UNIVERSITY OF
MELBOURNE

Multilayer perceptron: Gradient I

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



Multilayer perceptron: Gradient I

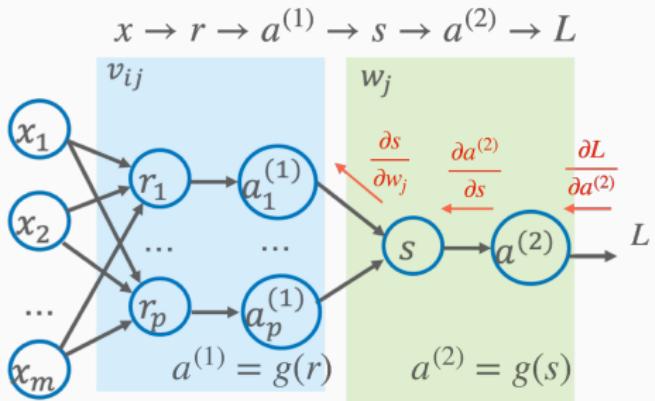
$$\begin{aligned}\frac{\partial L}{\partial w_j} &= (a^{(2)} - y) g' a_j^{(1)} \\ &= \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s} \frac{\partial s}{\partial w_j}\end{aligned}$$

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

$$\begin{array}{c} \uparrow \\ a^{(2)} = g(s) \end{array}$$

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

Forward



Multilayer perceptron: Update weights

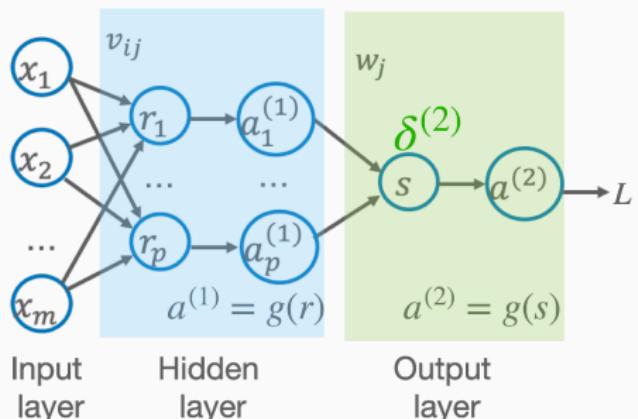
To reduce loss, update each weight in each layer:

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

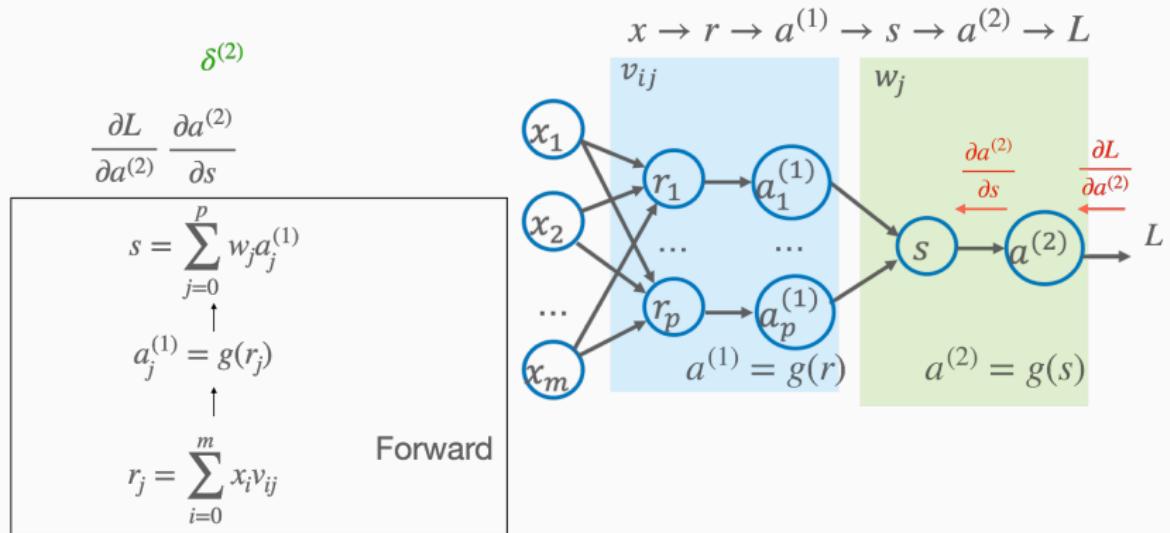
$$\begin{aligned} w_j^{new} &= w_j^{old} - \eta * \frac{\partial L}{\partial w_j} \\ &= w_j^{old} - \eta \delta^{(2)} a_j^{(1)} \end{aligned}$$

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

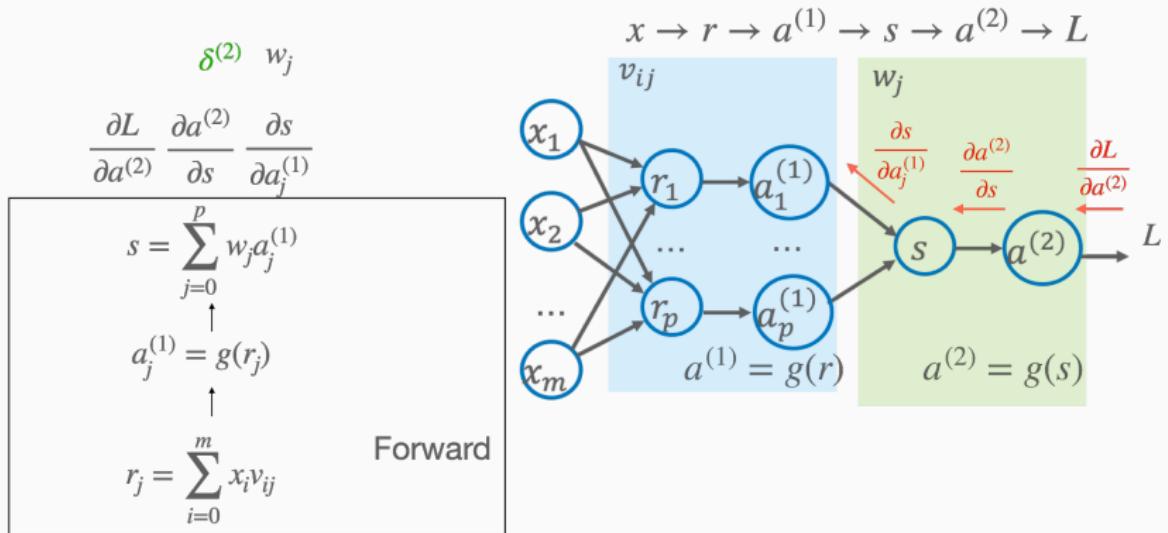
(Error at the output layer)



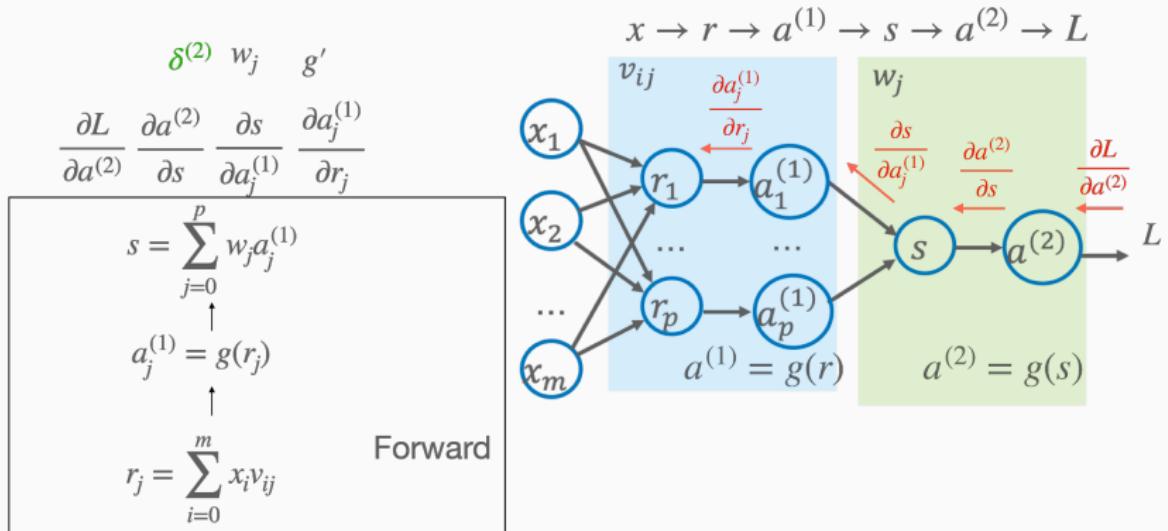
Multilayer perceptron: Gradient II



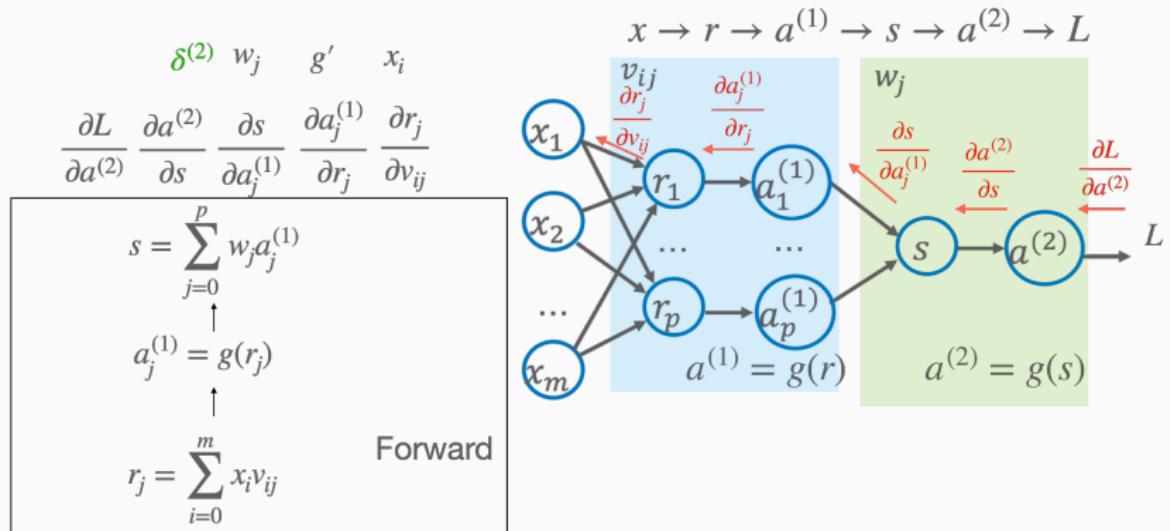
Multilayer perceptron: Gradient II



Multilayer perceptron: Gradient II



Multilayer perceptron: Gradient II



Multilayer perceptron: Gradient II

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

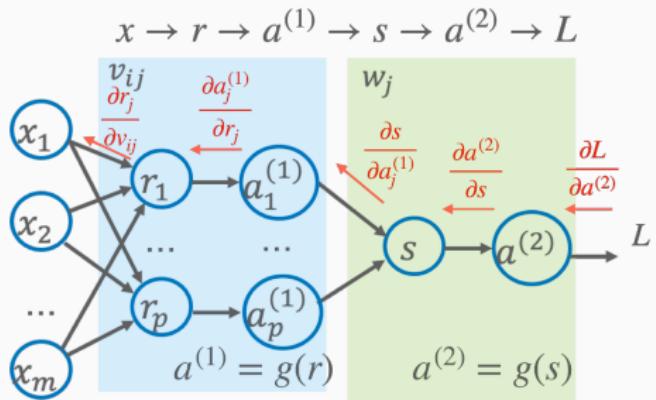
$$= \frac{\partial L}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial s} \frac{\partial s}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial r_j} \frac{\partial r_j}{\partial v_{ij}}$$

Forward

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

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

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



Multilayer perceptron: Update weights

To reduce loss, update each weight in each layer:

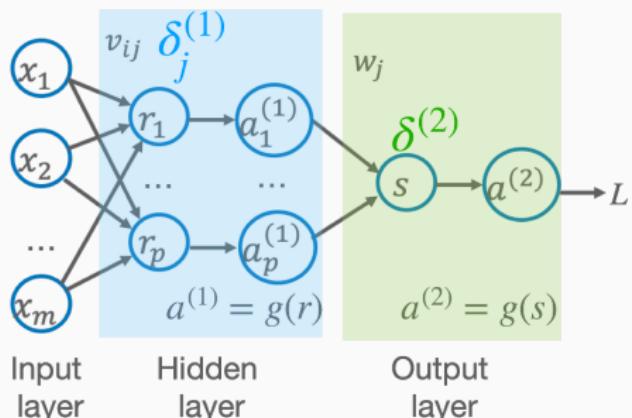
$$\frac{\partial L}{\partial v_{ij}} = \delta_j^{(2)} w_j g' x_i = \delta_j^{(1)} x_i$$

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

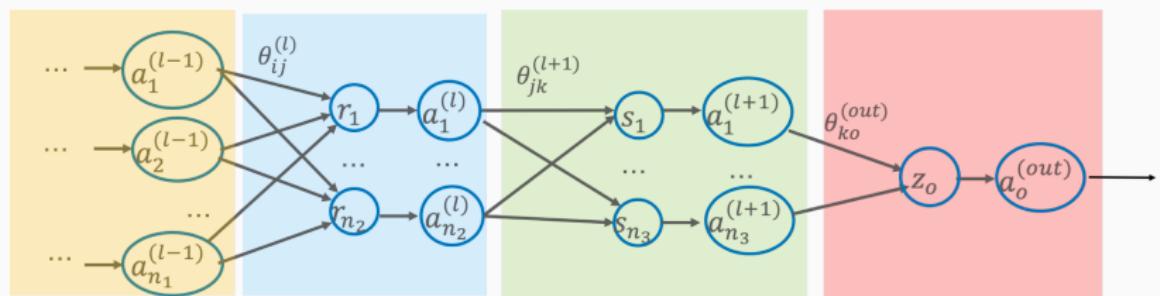
$$= v_{ij}^{old} - \eta \delta_j^{(1)} x_i$$

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

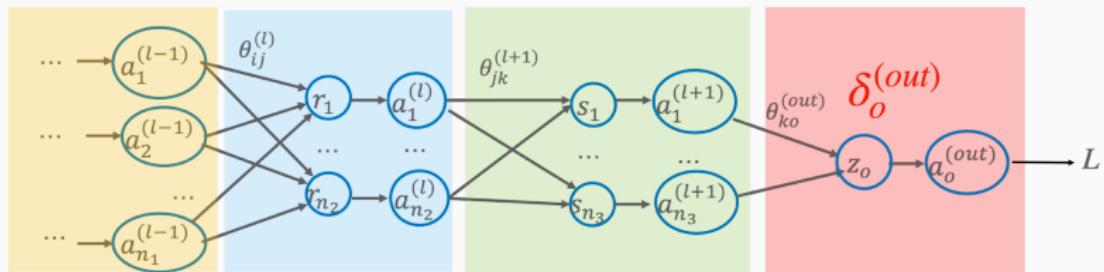
$$\delta_j^{(1)} = \delta^{(2)} w_j g' \quad (\text{error backpropagation})$$



Multilayer perceptron: more hidden layers



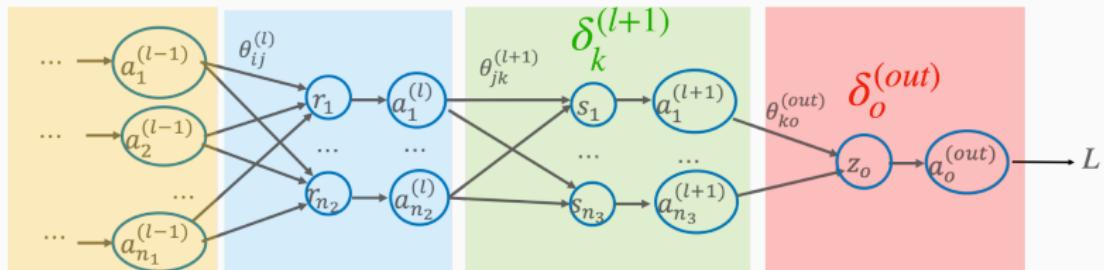
Backpropagation: error of output layer



$$\delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} \frac{\partial a^{(out)_o}}{\partial z_o} = \frac{\partial L}{\partial a_o^{(out)}} g'$$

$$\theta_{ko}^{(out_{new})} = \theta_{ko}^{(out_{old})} - \eta \delta_o^{(out)} a_k^{(l+1)}$$

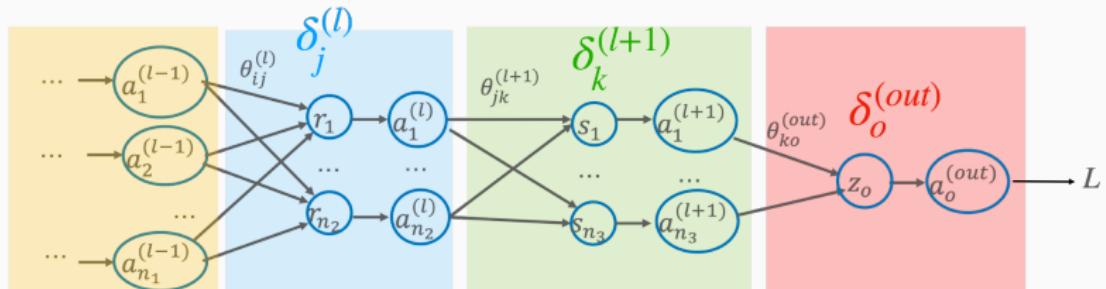
Backpropagation: error of hidden layer



$$\delta_k^{(l+1)} = \sum_o \delta_o^{(out)} \theta_{ko}^{(out)} g' \quad \delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} \frac{\partial a^{(out)_o}}{\partial z_o} = \frac{\partial L}{\partial a_o^{(out)}} g'$$

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

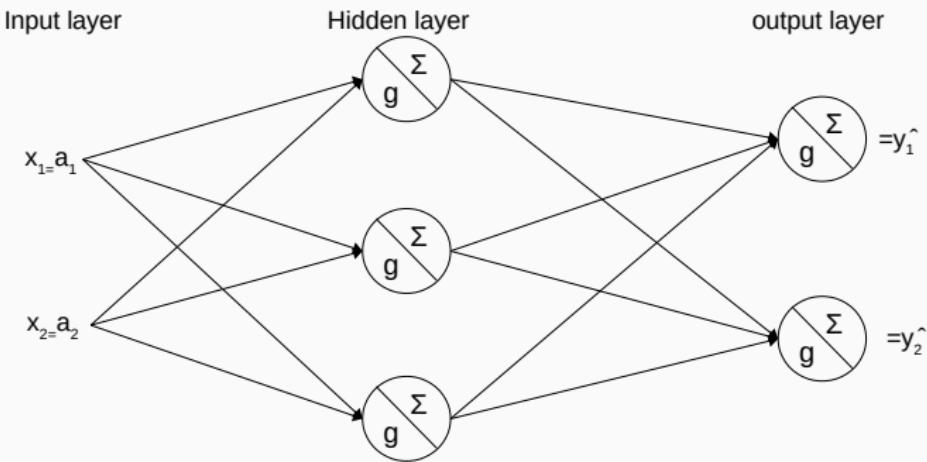
Backpropagation: error of hidden layer



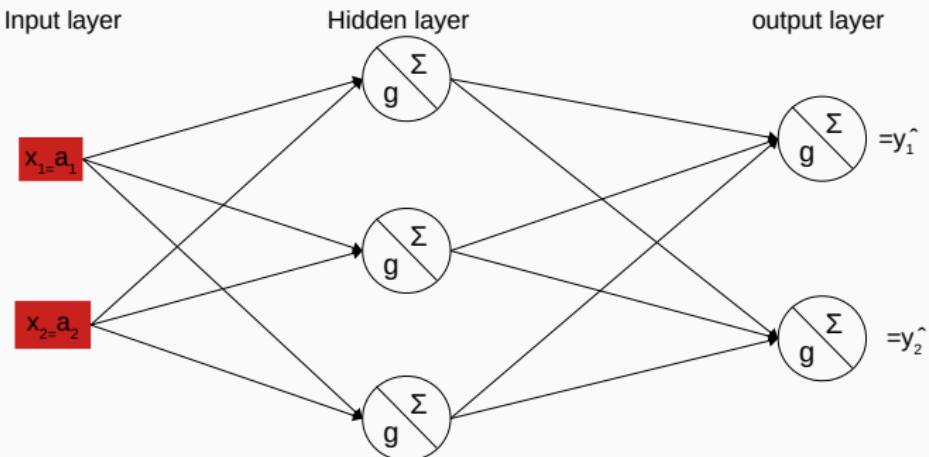
$$\delta_j^{(l)} = \sum_{k=1}^{n_3} \delta_k^{(l+1)} \theta_{jk}^{(l+1)} g' \quad \delta_k^{(l+1)} = \sum_o \delta_o^{(out)} \theta_{ko}^{(out)} g' \quad \delta_o^{out} = \frac{\partial L}{\partial a_o^{(out)}} \frac{\partial a_o^{(out)}_o}{\partial z_o} = \frac{\partial L}{\partial a_o^{(out)}} g'$$

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

Backpropagation: Demo

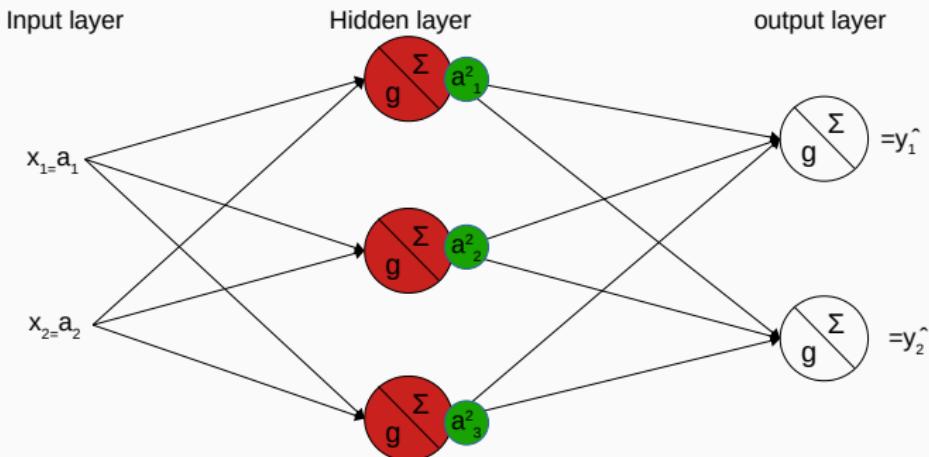


Backpropagation: Demo



- Receive input

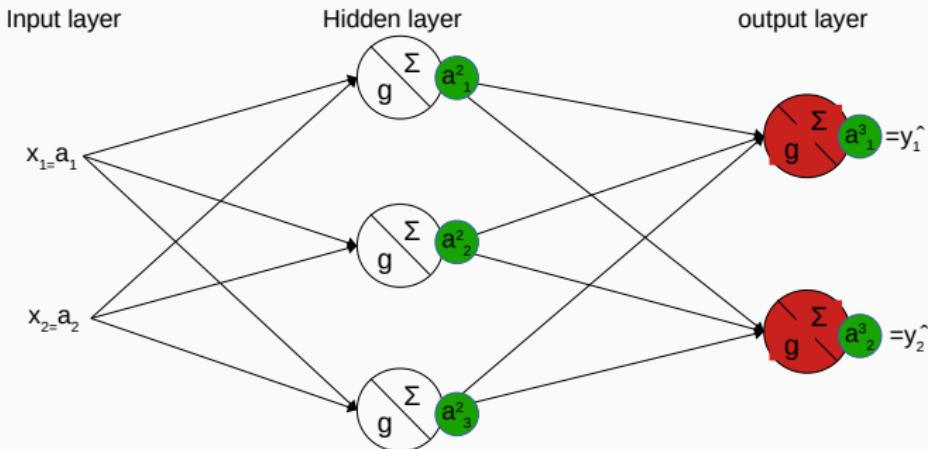
Backpropagation: Demo



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

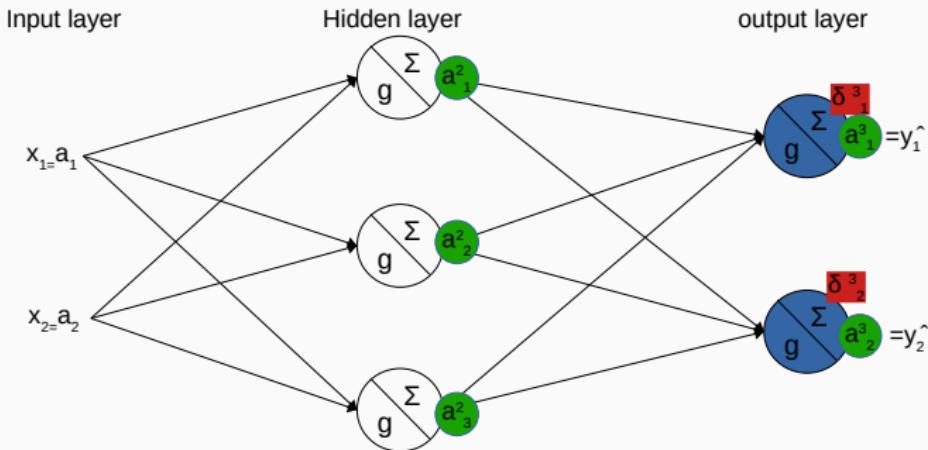


Backpropagation: Demo



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

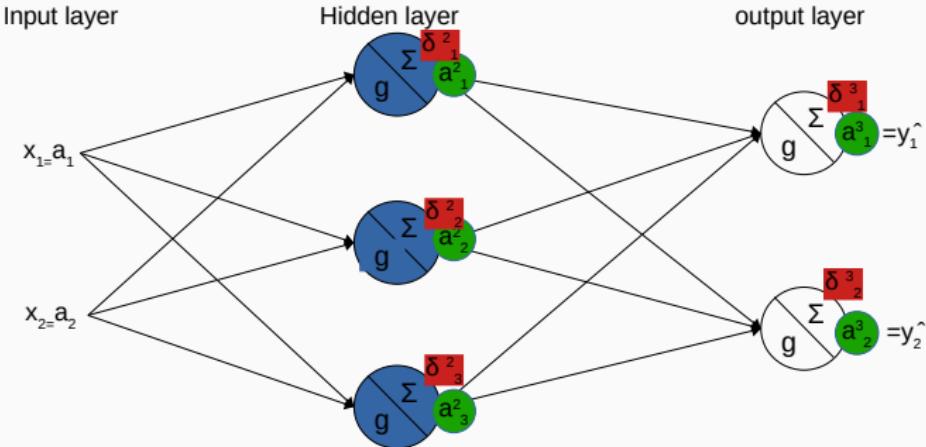
Backpropagation: Demo



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

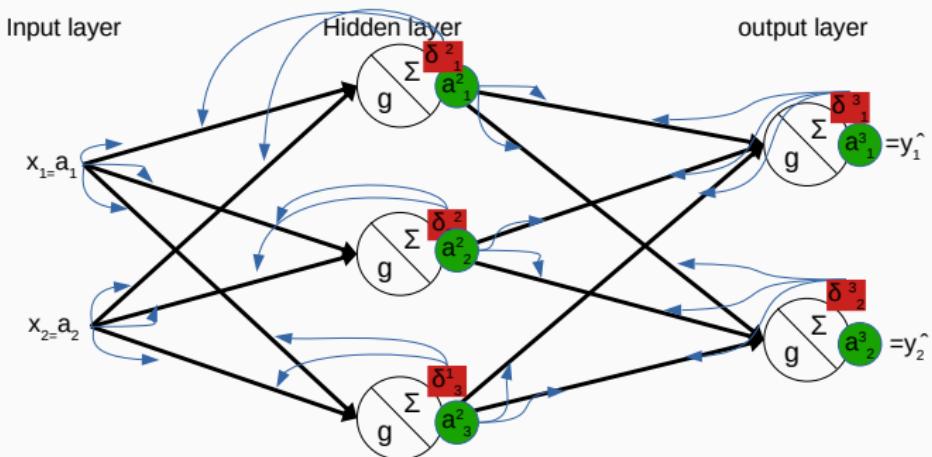


Backpropagation: Demo



- 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 I

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

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

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

until stopping criteria reached.



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

