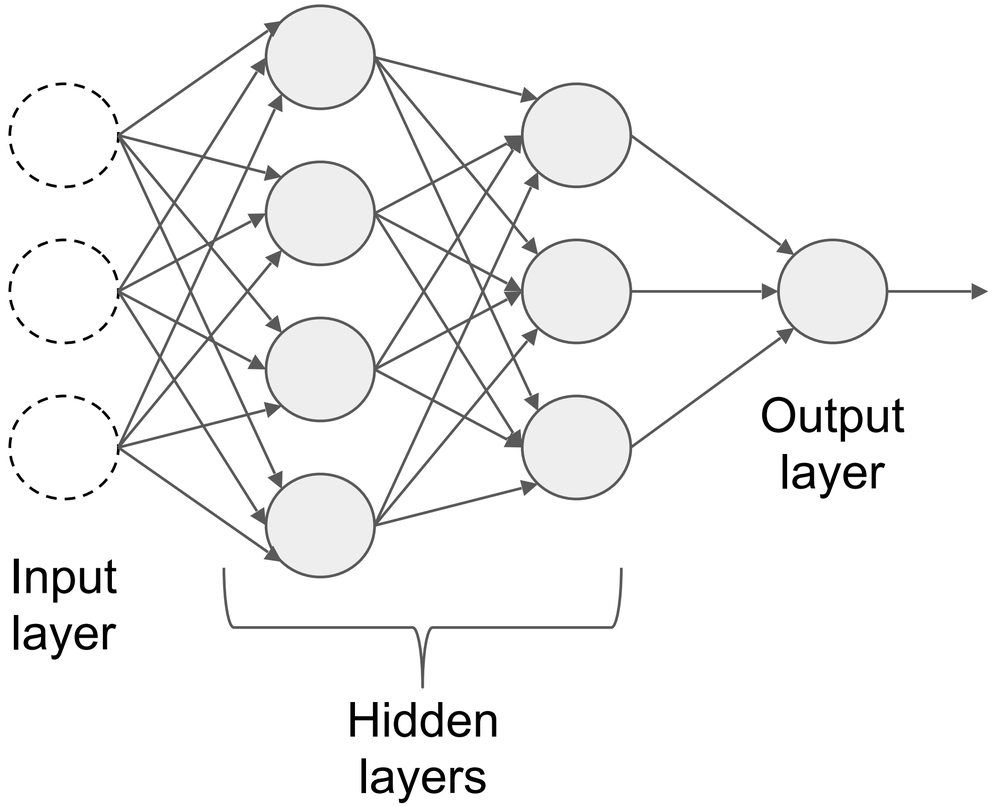
Part 2
FCNN, Learning, etc.

Frank Lindseth, IDI, NTNU

Neural Networks and Deep Learning (Nielsen)

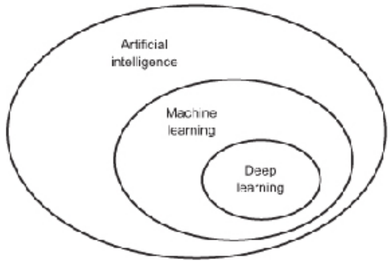
Key concepts

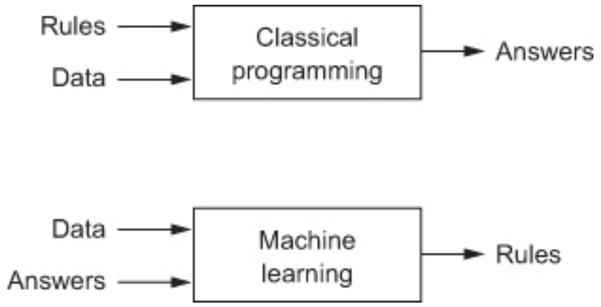
- Neuron (AN) / Unit
  - Weights, w-input
  - Activation functions & derivatives
- ANN (FF, FC, shallow)
  - Layers
- Learning
  - Inference, forward pass of input
  - GD, GT & Cost / Error / Loss functions
  - Training, GD, update-rule, back-propagation of error

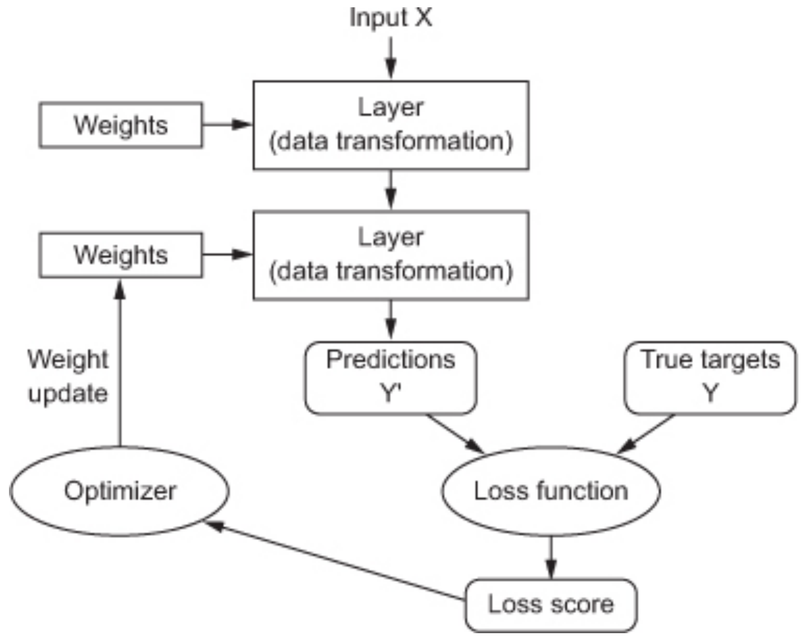


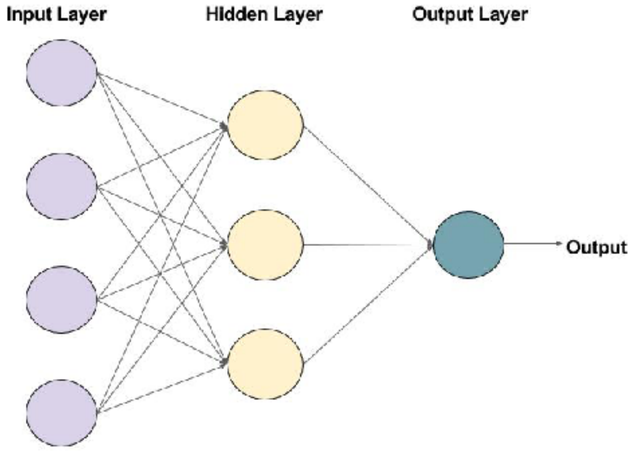
Shallow Fully-Connected NNs

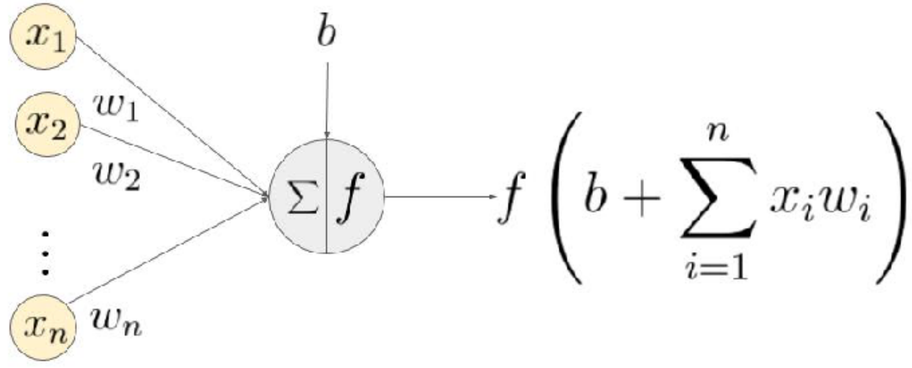
- Artificial Neural Networks (ANNs)
- Artificial Neurons (ANs)
- Activation Functions (AFs)
- Biological Neurons & NN (BNs & BNN)
- Matrix-based Notation
- Numerical Examples





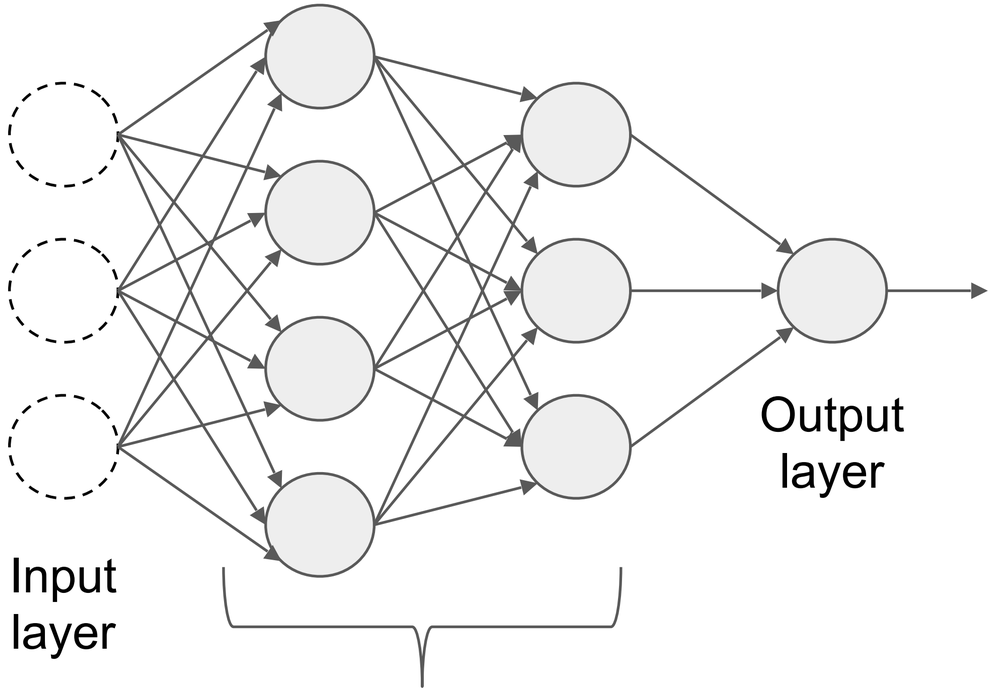


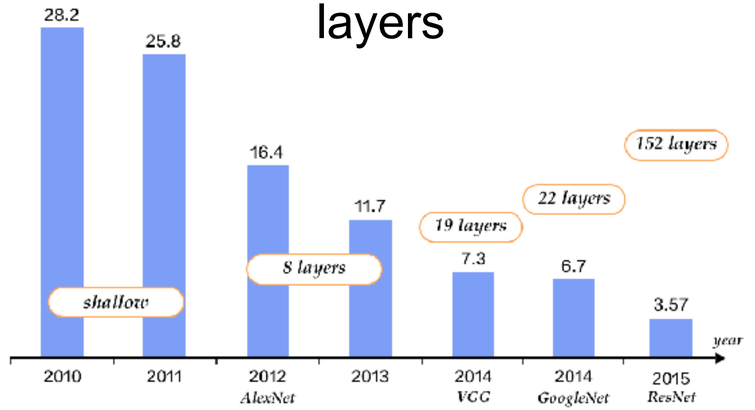




Artificial Neural Networks (ANNs)

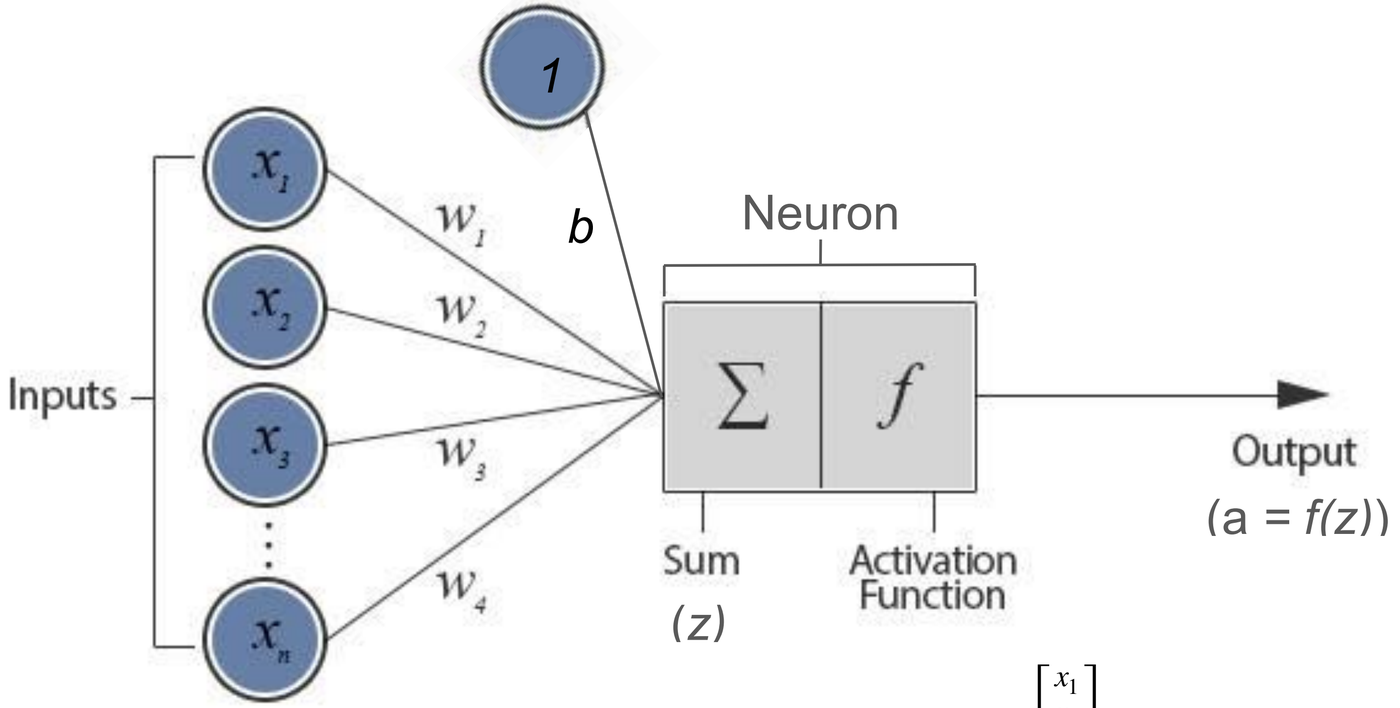
- Network **architecture** (model): shallow, fully-connected, feed-forward ANNetworks
  - An ANN consists of multiple **layers**
  - A layer consists of multiple **neurons** (units)
  - **Shallow** (vs. deep) ANNs: «a few» hidden layers
  - **Fully-connected** (vs. ConvNets) ANN: a given neuron is connected to all neurons in previous and next layer.
  - **Feed-Forward** (vs. Recurrent) ANNs: only forward links

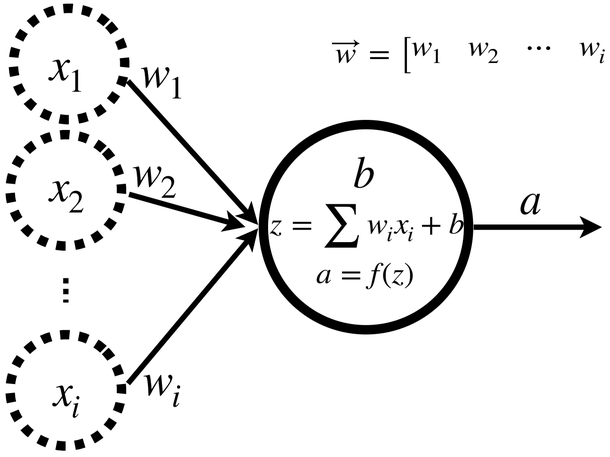




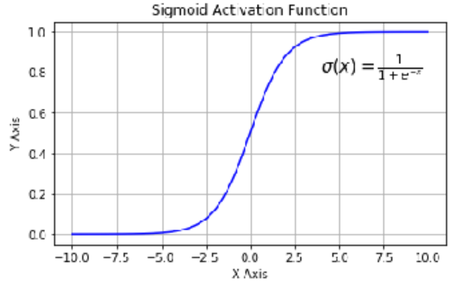
Artificial Neurons (ANs)

- Two operations:
  - weighted input: z = w*x + b
  - activation: a = f(z)
- Weights and Bias are learnable parameters.

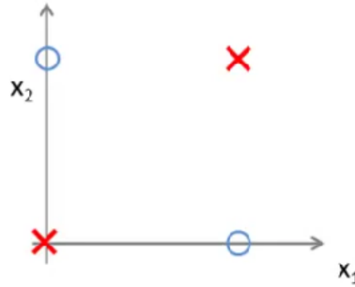




$$\vec{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_i & \cdots & w_{n_i} \end{bmatrix} \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_{n_i} \end{bmatrix}$$

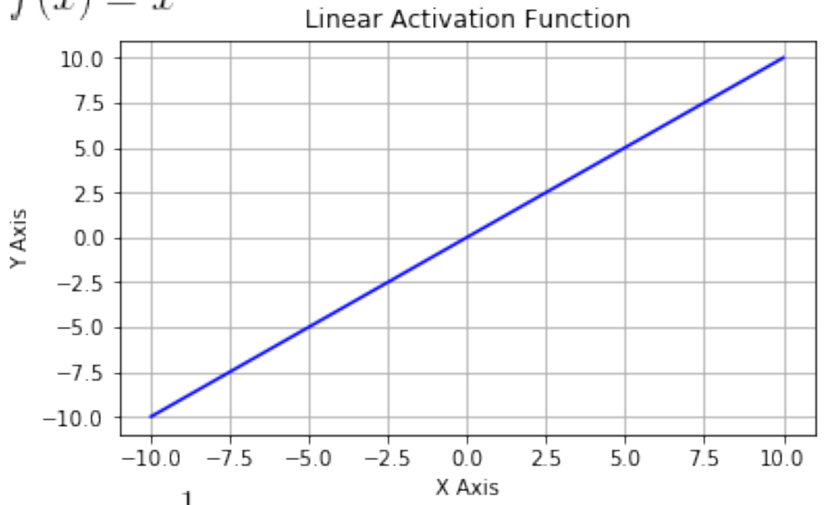


Activation Functions (AFs)

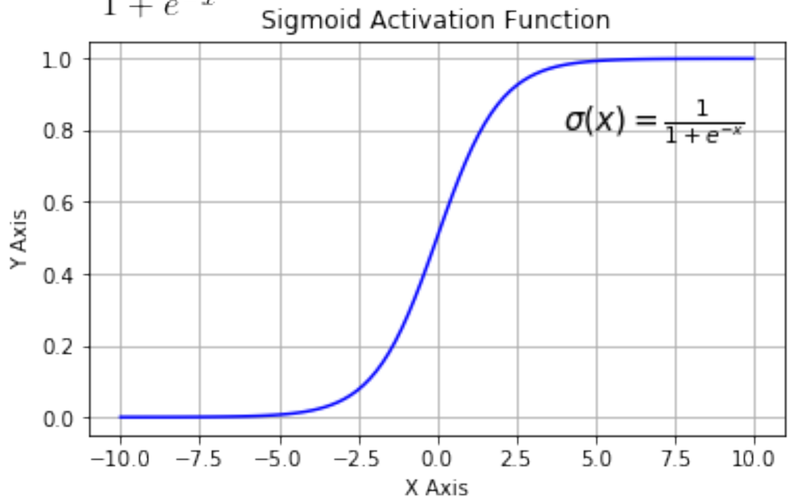


- Types: Linear vs. Non-Linear Activation Functions
- Wights and Bias transform the input signal linearly (XOR)
- A non-linearity (TF) allows us to learn arbitrarily complex transformations between the input and the output.
  - solve problems where a non-linear decision boundary is needed to separate the data
  - also used to squash the output of the neural network to be within certain bounds.
- Still an active area of research: want TFs that makes the neural network learn better and faster (sigmoid, tanh, relu, lrelu, prelu, swish, etc.)
  - GD and Backpropagation uses **gradients** in the learning process, i.e. **partial derivatives** of the loss function w.r.t the weights/biases are used to update the weights/biases.

$$f(x) = x$$



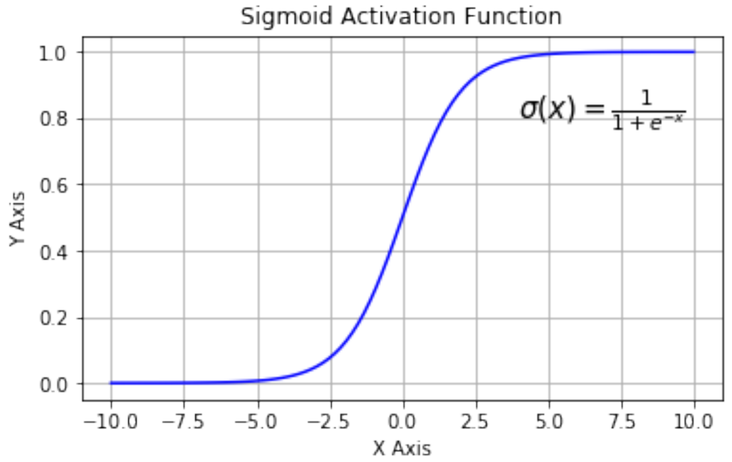
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

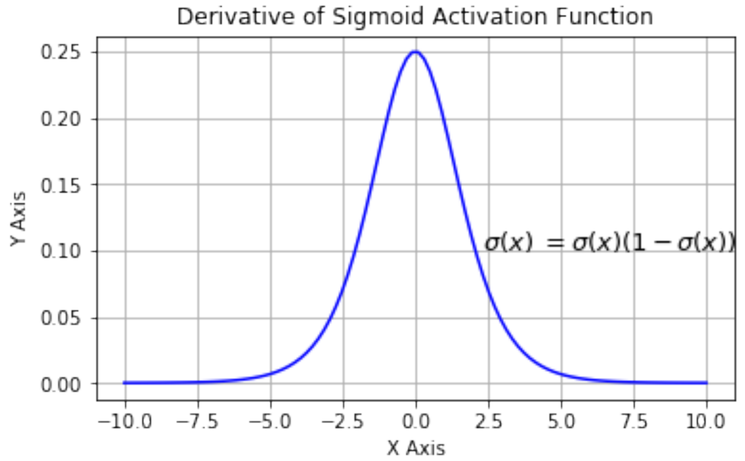


Sigmoid (Logistic) AF

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

- **Squashes** input between 0 (large negative numbers) and 1 (large positive numbers).
- Can be used in the output layer where our end goal is to predict **«probability»**.
- Drawbacks:
  - **Vanishing gradients**:
    - Sigmoid output close to 0/1 (flat) -> gradient / derivative close to 0 -> saturated neurons -> weights of neurons (or connected neurons) do not update -> network do not backpropagate, i.e. learn.
  - Not zero centered:
  - Computationally expensive:

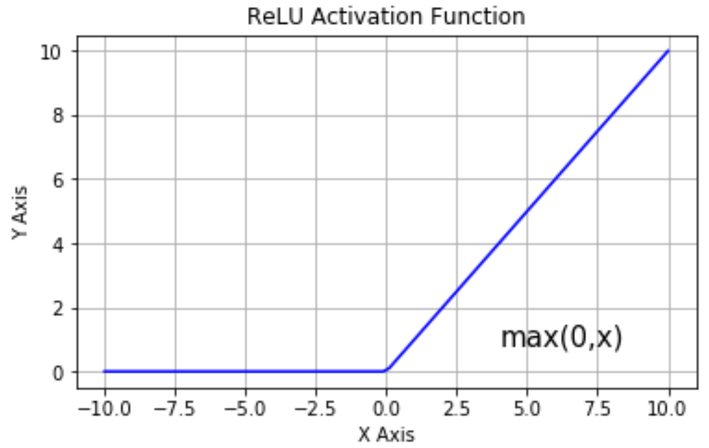


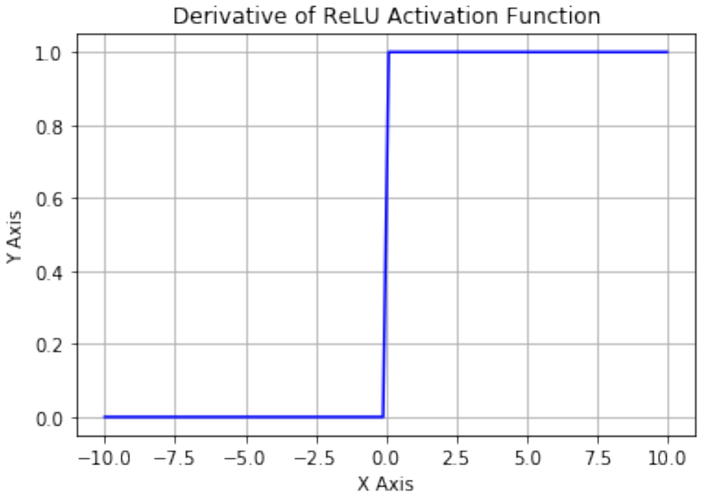


ReLU (Rectified Linear Unit) AF

$$f(x) = \max(0, x)$$

- ReLU is half-rectified (when the input x < 0 the output is 0 and if x > 0 the output is x).
- Network converge much faster (does not saturate, resistant to the vanishing gradient problem (positive region))
- ReLU is computationally very efficient.
- Drawbacks
  - Not zero-centered
  - if x < 0 during the forward pass, the neuron remains **inactive** and it **kills** the gradient during the backward pass. Thus weights do not get updated, and the network does not learn. When x = 0 the slope is undefined.





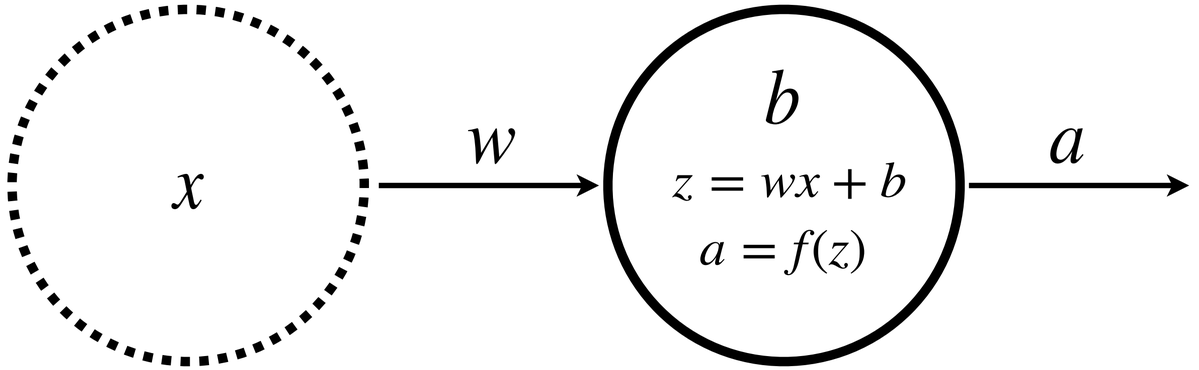
Softmax AF

- We sometimes want that the output should express probabilities (e.g. image classification problems)
- Use Softmax to scale the outputs
- Want outputs between 0 and 1 and should sum to 1

$$f(z) = \begin{bmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_j) \\ \vdots \\ f(z_m) \end{bmatrix}, f(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{m} e^{z_i}}$$

node	x	e^x	e^x / sum(e^x)
1	2	7.39	0.71
2	1	2.72	0.26
3	-1	0.37	0.04
total		10.48	1.00

Single example, single input - single layer, single neuron



x = input

w = weight

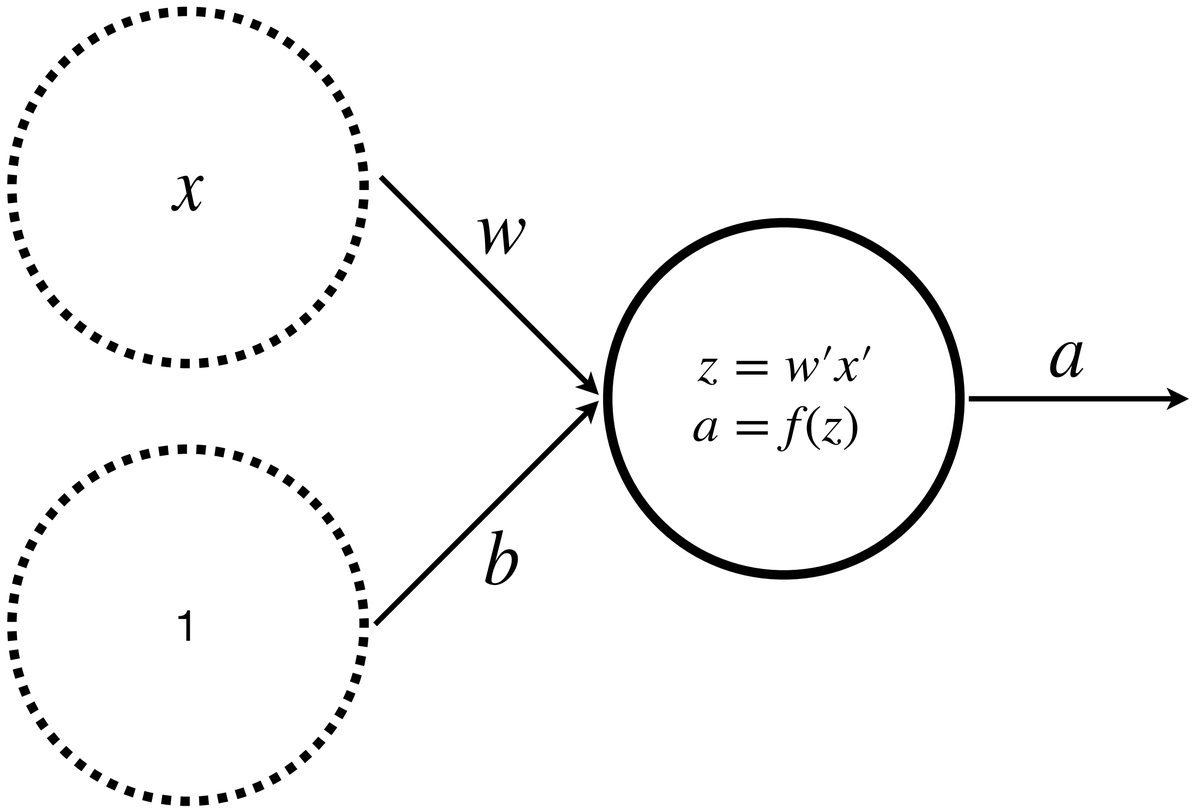
z = weighted input

f = activation function

a = activation

b = **bias**

Single example, single input - single layer, single neuron



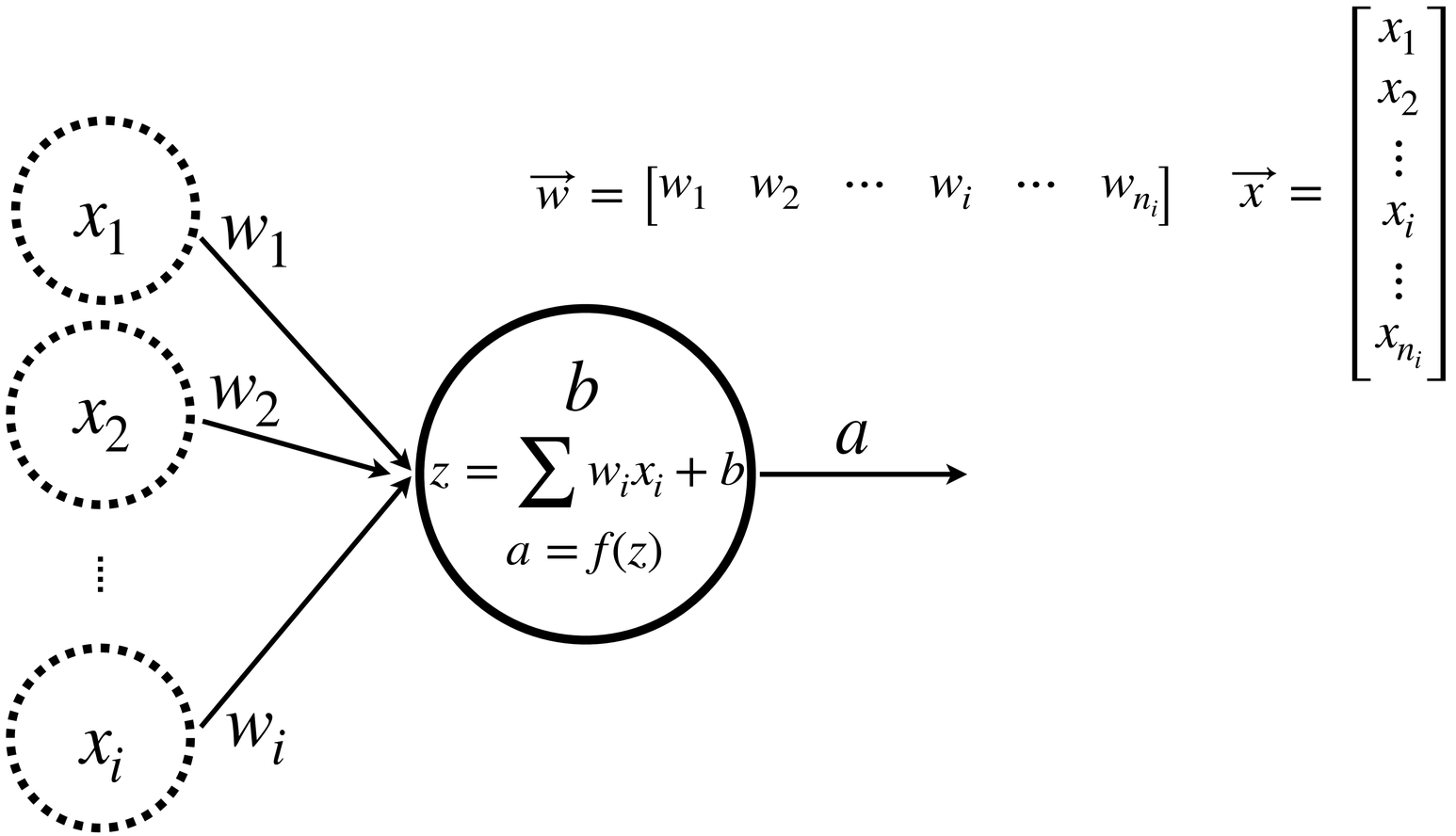
bias trick

$$w' = [w \quad b]$$

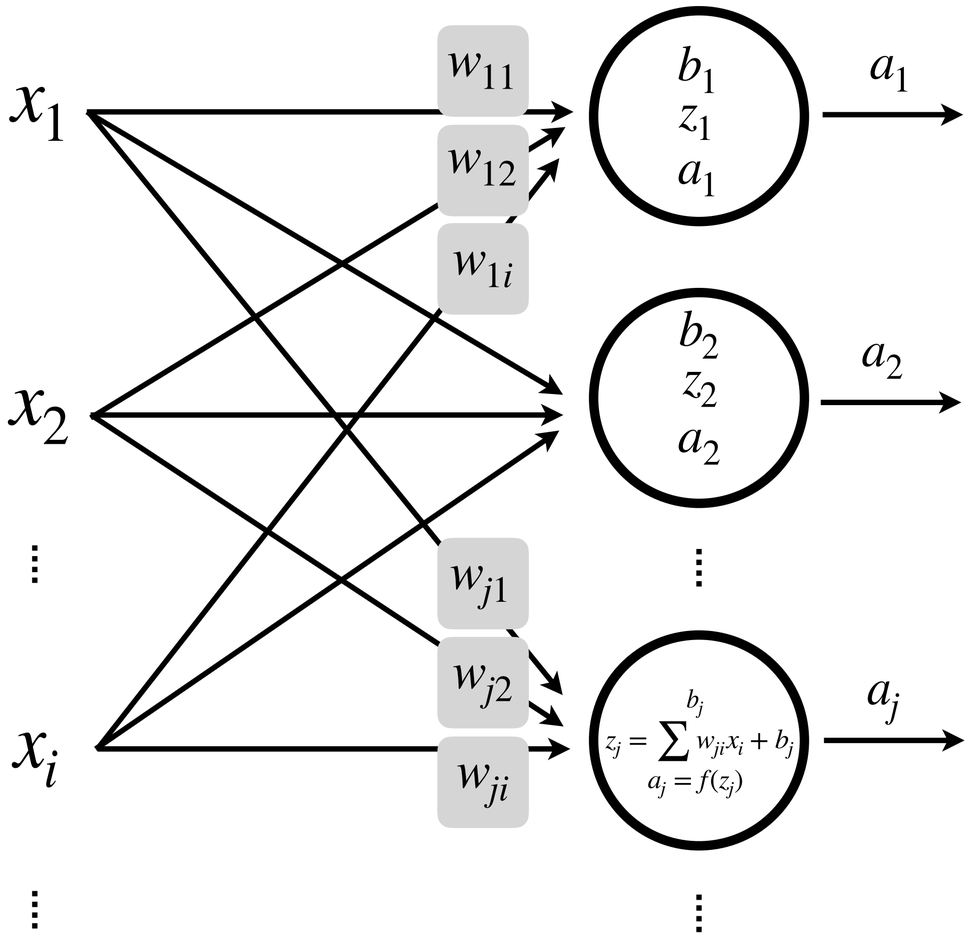
$$x' = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

$$z = w'x' = [w \quad b] \begin{bmatrix} x \\ 1 \end{bmatrix} = wx + b$$

Single example, multiple inputs - single layer, single neuron



Single example, multiple inputs - single layer, multiple neurons

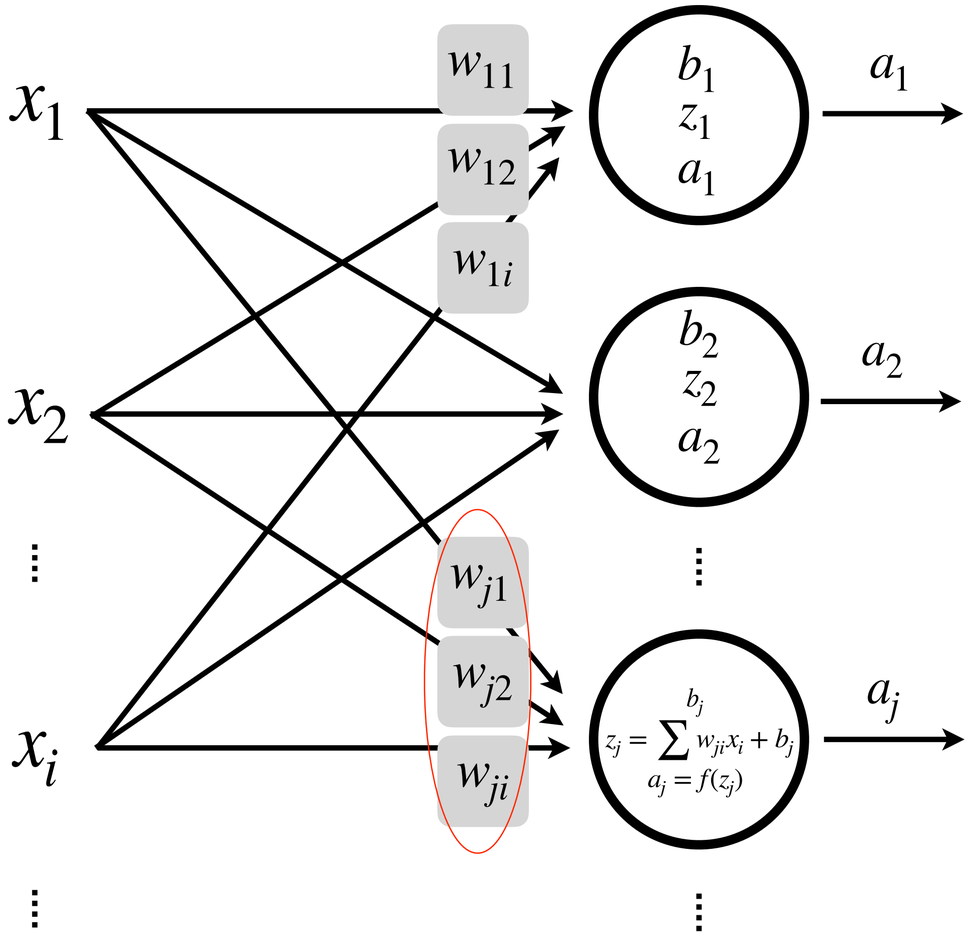


$$z_1 = \sum_i w_{1i}x_i + b_1$$
$$= w_{11}x_1 + w_{12}x_2 + \cdots + w_{1i}x_i + \cdots$$
$$a_1 = f(z_1)$$

$$z_2 = \sum_i w_{2i}x_i + b_2$$
$$= w_{21}x_1 + w_{22}x_2 + \cdots + w_{2i}x_i + \cdots$$
$$a_2 = f(z_2)$$

$$z_j = \sum_i w_{ji}x_i + b_j$$
$$= w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{ji}x_i + \cdots$$
$$a_j = f(z_j)$$

Single example, multiple inputs - single layer, multiple neurons



$$w = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1i} & \cdots \\ w_{21} & w_{22} & \cdots & w_{2i} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \end{pmatrix}$$

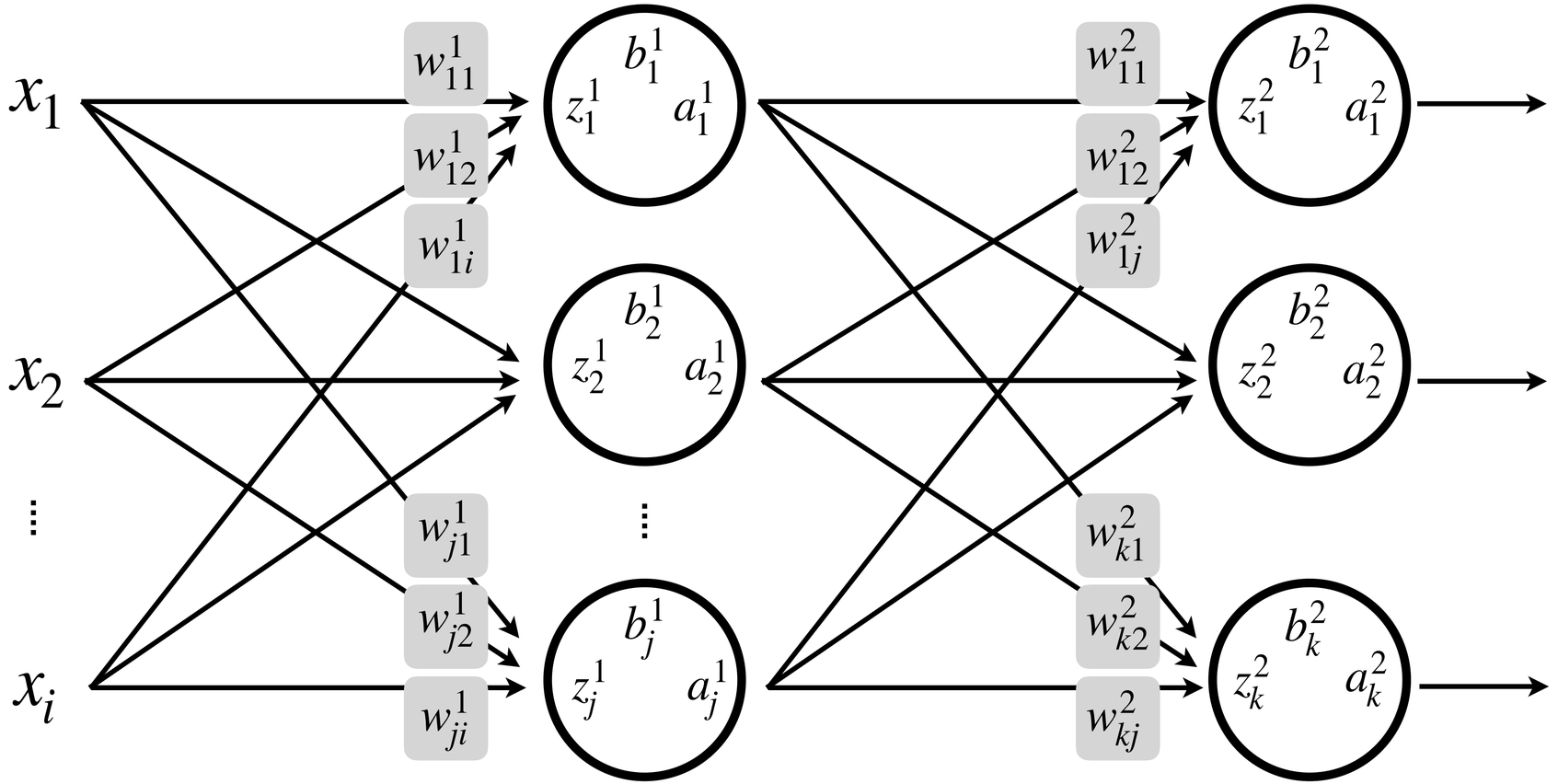
$$w_j = \vec{w}_j = \begin{bmatrix} w_{j1} & w_{j2} & \cdots & w_{ji} & \cdots \end{bmatrix}$$

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \\ \vdots \end{pmatrix} \quad z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_j \\ \vdots \end{pmatrix} \quad a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_j \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_j \\ \vdots \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1i} & \cdots \\ w_{21} & w_{22} & \cdots & w_{2i} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \\ \vdots \end{pmatrix}$$

$$z_j = \vec{w}_j \vec{x} + b_j \qquad a_j = f(z_j)$$

Single example, multiple inputs - multiple layers, multiple neurons



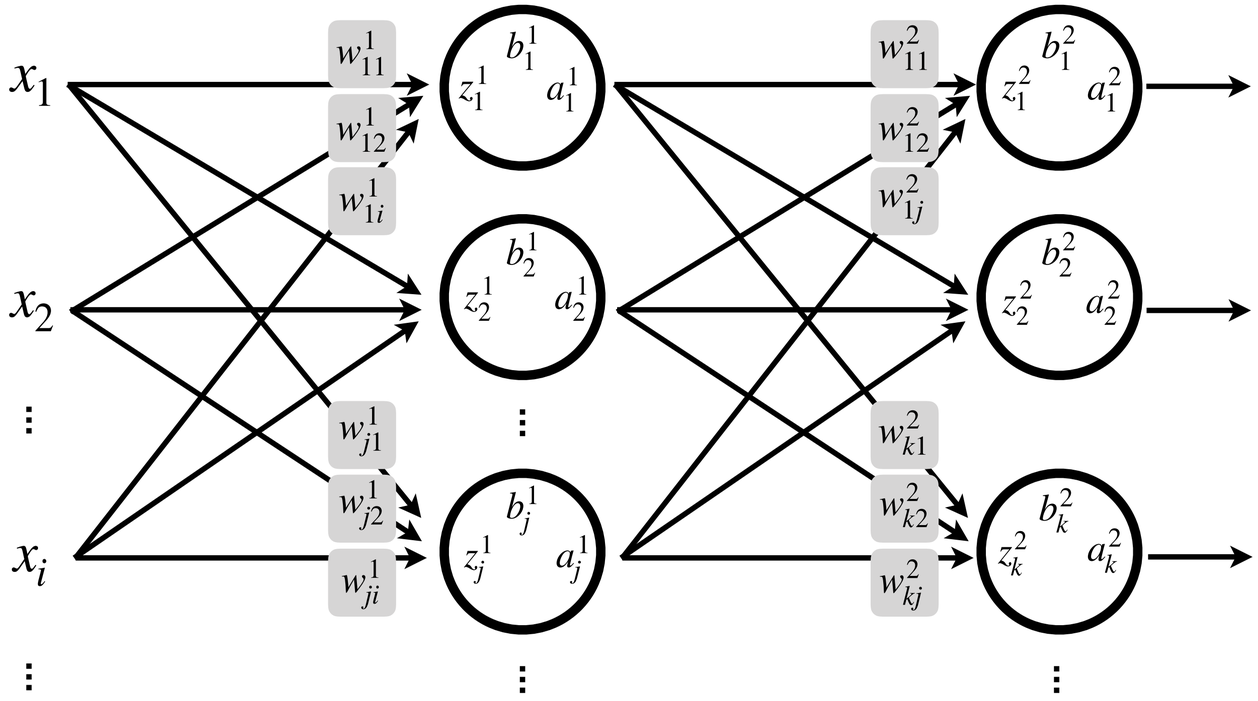
w^l_{ji} is the **weight** to neuron **j** in layer **l** from neuron **i** in layer **l-1**

z^l_j is the **weighted input** of neuron **j** in layer **l**

b^l_j is the **bias** of neuron **j** in layer **l**

a^l_j is the **activation** of neuron **j** in layer **l**

Single example, multiple inputs - multiple layers, multiple neurons



Vectorization:

$$x = \vec{x}$$
$$f(x) = f(\vec{x})$$
$$f(x)_j = f(x_j)$$
$$f(x_j) = x_j^2$$
$$f(\begin{bmatrix}1\\2\end{bmatrix}) = ?$$

a^0

w^1
b^1 z^1 a^1

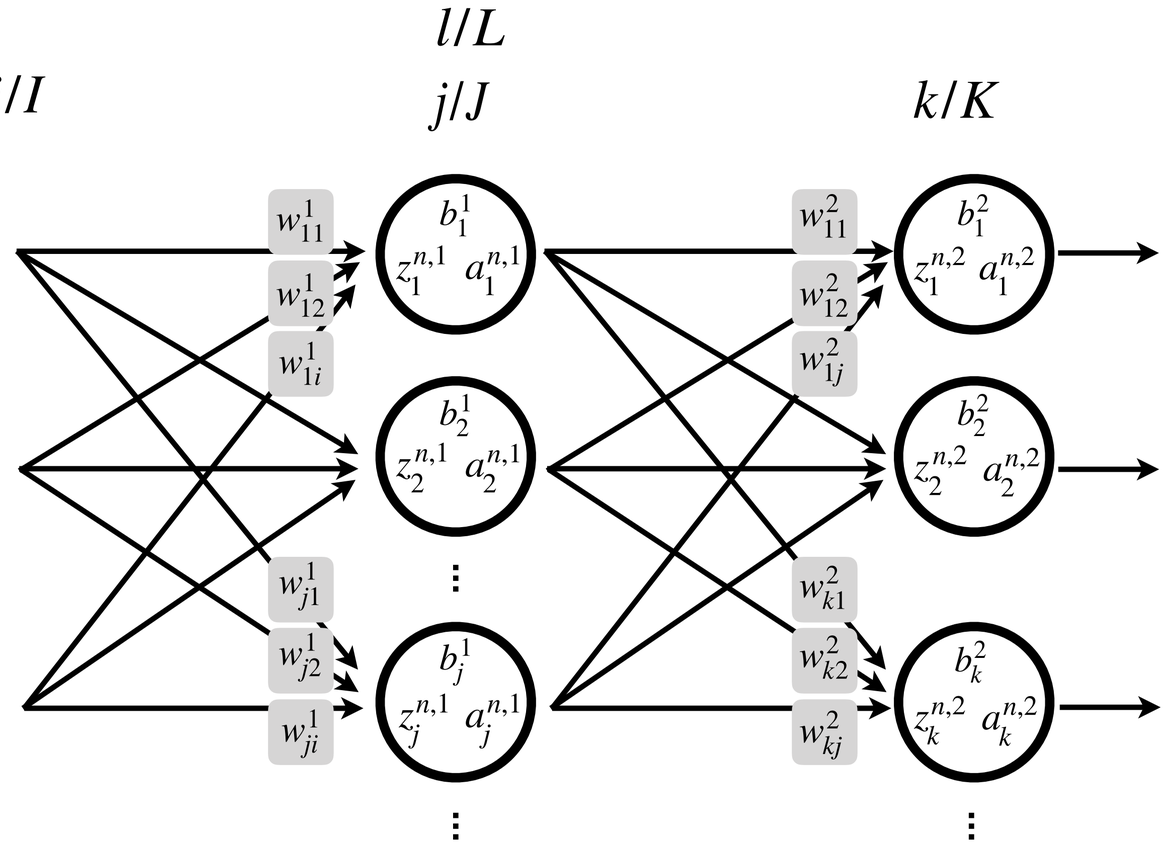
w^2
b^2 z^2 a^2

w^l
b^l z^l a^l

$$a^l = f(z^l)$$
$$z^l = w^l a^{l-1} + b^l$$

Multiple examples, multiple inputs - multiple layers, multiple neurons

$$x = \begin{pmatrix} x^1 & x^2 & \cdots & x^n & \cdots \end{pmatrix} = \begin{pmatrix} x_1^1 & x_1^2 & \cdots & x_1^n & \cdots \\ x_2^1 & x_2^2 & \cdots & x_2^n & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_i^1 & x_i^2 & \cdots & x_i^n & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$



index i = input i
index j = hidden neuron j (j1, j2..)
index k = output neuron k
index n = example n
index l = layer l

I = # inputs
J = # neurons in hidden layer
K = # neurons in output layer
N = # examples (batch size)
L = # layers (input = layer 0)

$w^1 \; b^1$

$z^{n,1} \; a^{n,1}$

$w^l \; b^l$

$a^{n,0} = x^n$

$z^{n,l} = w^l a^{n,l-1} + b^l$

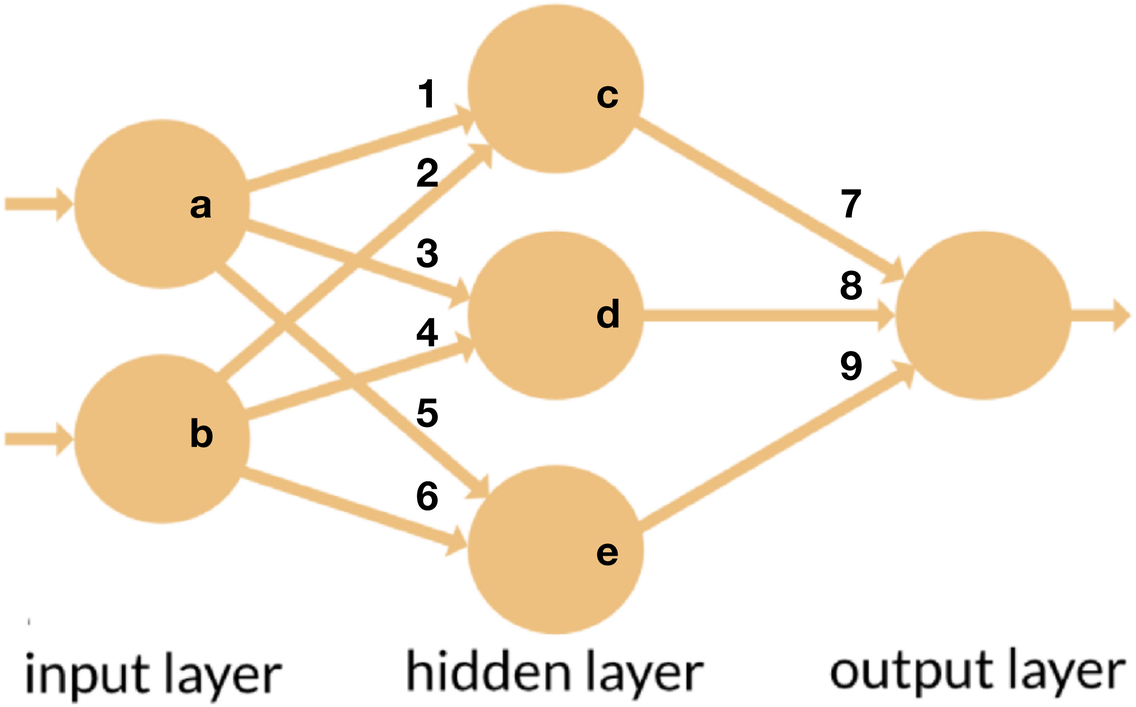
$a^{n,l} = f(z^{n,l})$

Forward pass

$$w^1 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \qquad w^2 = (7 \quad 8 \quad 9)$$

Given: two weight matrixes. **Provide**: figure with weights

Forward pass



Given: figure with weights **Provide**: two weight matrixes

Matrix-based Notation: Summary

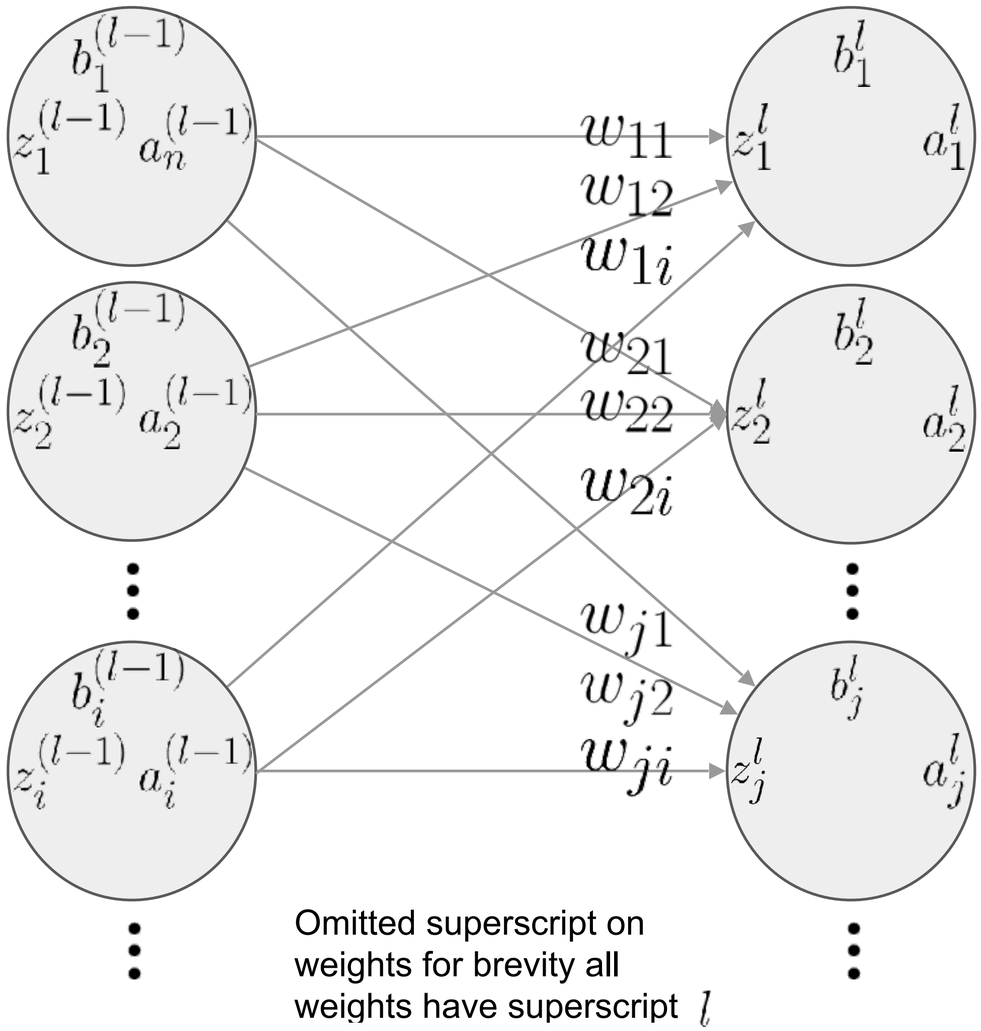
w^l_{ji}: weights between layer $l-1$ and l; node i in layer $l-1$; node j in layer l

b^l_j: bias for layer l; node j in layer l

z^l_j: weighted input for layer l; node j in layer l

a^l_j: activation for layer l; node j in layer l

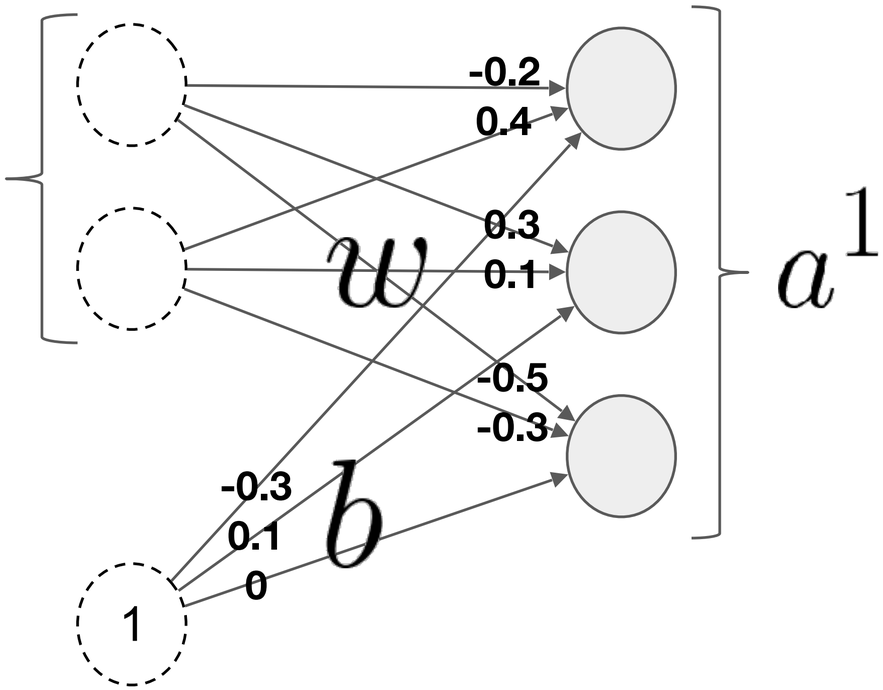
$$a^l = f\left(\overbrace{w^l a^{(l-1)} + b^l}^{z^l}\right)$$



Omitted superscript on weights for brevity all weights have superscript l

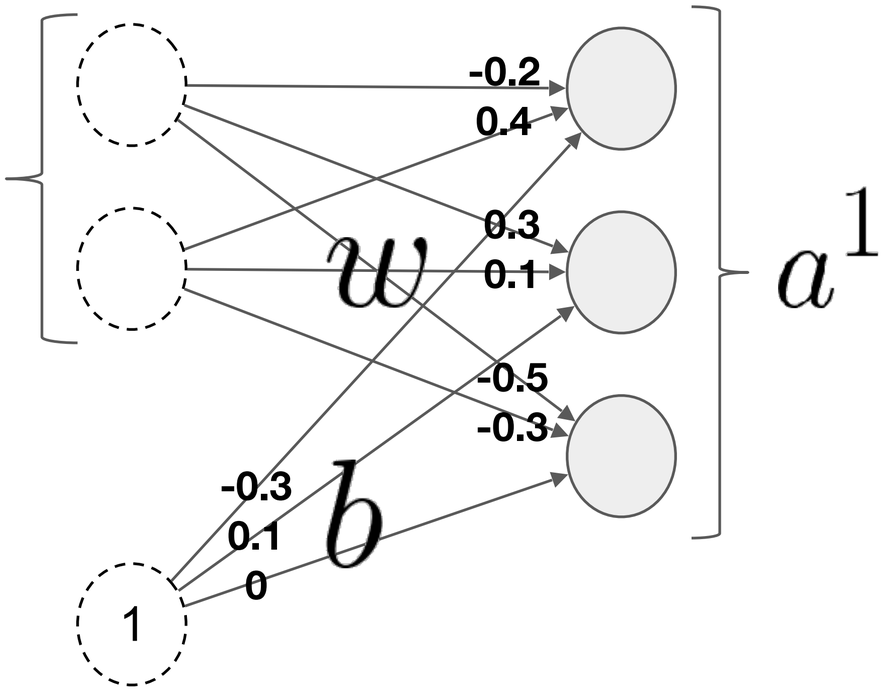
Numerical Example

$$x = a^0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad f : ReLU$$



$$x = a^0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad f : ReLU$$

$$w^1 = \begin{bmatrix} -0.2 & 0.4 \\ 0.3 & 0.1 \\ -0.5 & -0.3 \end{bmatrix} \; b^1 = \begin{bmatrix} -0.3 \\ .1 \\ 0 \end{bmatrix}$$



$$a^1 = f\left(w^1 a^0 + b^1\right)$$

$$a^1 = f\left(\begin{bmatrix} -0.2 & 0.4 \\ 0.3 & 0.1 \\ -0.5 & -0.3 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.3 \\ .1 \\ 0 \end{bmatrix}\right)$$

$$a^1 = f\left(\begin{bmatrix} 0.6 \\ -0.2 \\ 0.2 \end{bmatrix} + \begin{bmatrix} -0.3 \\ 0.1 \\ 0 \end{bmatrix}\right) \qquad a^1 = f\left(\begin{bmatrix} 0.3 \\ -0.1 \\ 0.2 \end{bmatrix}\right)$$

$$a^1 = \begin{bmatrix} 0.3 \\ 0 \\ 0.2 \end{bmatrix}$$

Bias trick

$$a^1 = f\left(\begin{bmatrix} -0.2 & 0.4 \\ 0.3 & 0.1 \\ -0.5 & -0.3 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.3 \\ .1 \\ 0 \end{bmatrix}\right)$$

$$a^1 = f\left(\begin{bmatrix} -0.2 & 0.4 & -0.3 \\ 0.3 & 0.1 & 0.1 \\ -0.5 & -0.3 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}\right)$$

Activation function: $f(z) = \dfrac{1}{1+e^{-z}}$

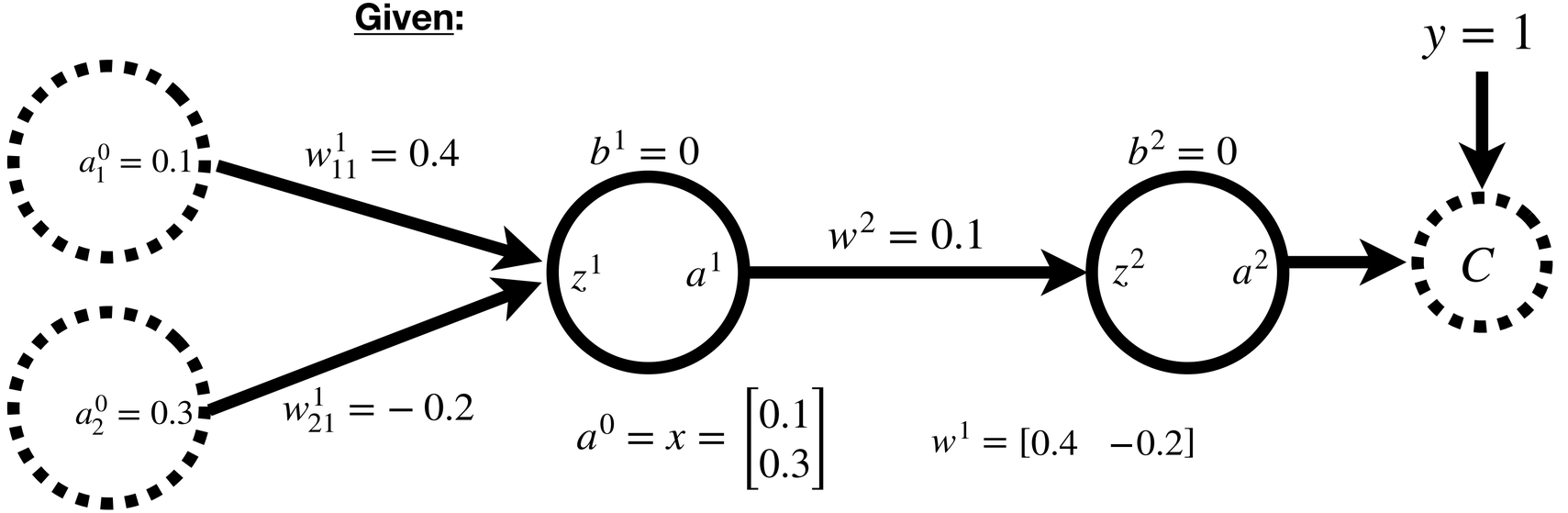
$$\frac{df(z)}{dz} = f(z)(1-f(z))$$

Cost function: $C = C(a^2) = \dfrac{1}{2}(y-a^2)^2$

$$\frac{\partial C}{\partial a^2} = a^2 - y$$

Learning rate: $\eta = 0.01$

Given:



$$a^0 = x = \begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix} \qquad w^1 = [0.4 \quad -0.2]$$

Find:

Forward pass:

$$z^1 = w^1 a^0 + b^1 = [0.4 \quad -0.2] \begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix} + 0 = 0.04 - 0.06 = -0.02$$

$$a^1 = f(z^1) = f(-0.02) = 0.495$$

$$z^2 = w^2 a^1 + b^2 = 0.1 \cdot 0.495 + 0 = 0.0495$$

$$a^2 = f(z^2) = f(0.0495) = 0.5124$$

Big picture / overview: Predictions

Image

Neural Network
(black box)

Cat → 0.97

Dog → 0.01

Other → 0.02

Input Layer Hidden Layer Output Layer

Output

Fully-Connected Net

Conv Layer 1 Pooling Layer 1 Conv Layer 2 Pooling Layer 2 Conv Layer 3 Pooling Layer 3 Fully Connected Layer Output layer

0

1

Convolutional NN

x_1 x_2 x_n w_1 w_2 w_n b Σ f

$$f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

Neuron

Repetition: Learning

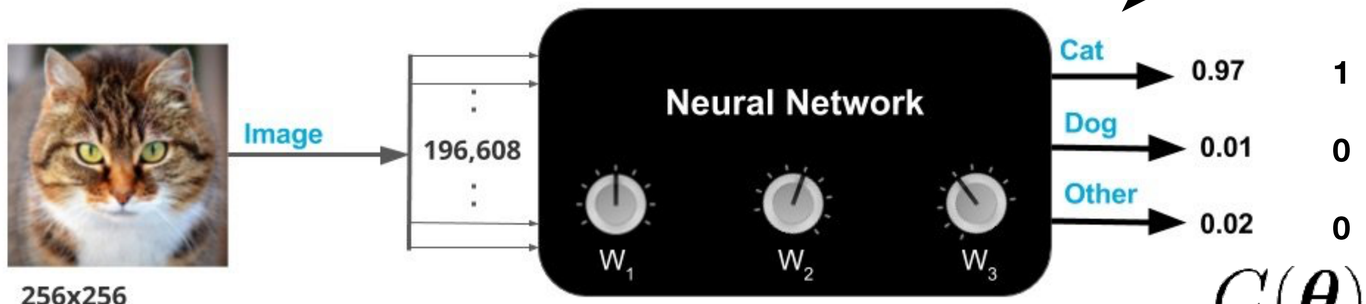
$f : \mathbb{R}^I \rightarrow \mathbb{R}^J$

$f(x) = f^{(L)}(f^{(L-1)}(\cdots f^{(1)}(x)))$

Given data: $(x^1, y^1), (x^2, y^2), \cdots (x^N, y^N)$

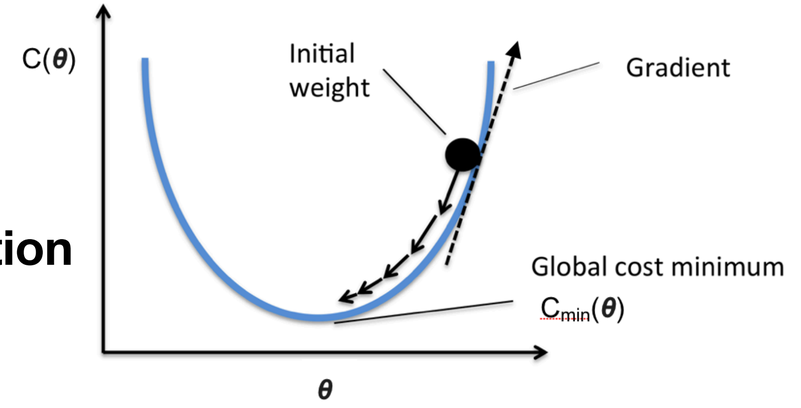
Find mapping: $f(x;\theta) = a^L = \hat{y}$ **=?** y **Desired output:**

Feed-forward **input**:



Back-propagate **error**

$C(\theta)$ **Cost function**



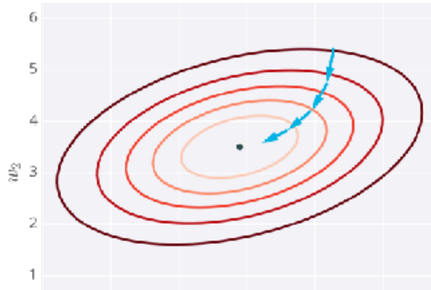
$$\frac{\partial C}{\partial w_{kj}^L} = \frac{\partial C}{\partial z^L}\frac{\partial z^L}{\partial w_{kj}^L} = \delta^L \frac{\partial z^L}{\partial w_{kj}^L}$$

$$\nabla C = \frac{\partial C}{\partial \theta_1}, \frac{\partial C}{\partial \theta_2} \cdots$$ **Gradients = partial derivatives**

$$\delta_k^L = \frac{\partial C}{\partial z_k^L} = \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_k^L} = \frac{\partial C}{\partial a_k^L} f'(z_k^L) \qquad a^L = f(z^L)$$

$$\frac{\partial z^L}{\partial w_{kj}^L} = a_j^{L-1} \qquad z^L = w^L a^{L-1} + b^L$$

$$\theta = \theta - \alpha \nabla C$$ **Update rule**



Cross-entropy (cost function)

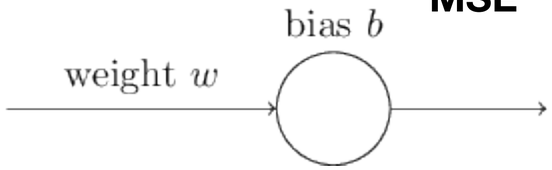
Cost functions:

MSE

Cross-entropy

Log-likelihood

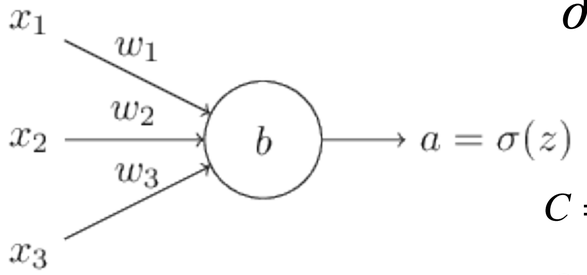
MSE



$$C = \frac{(y-a)^2}{2} \qquad x = 1, y = 0$$

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z)$$



$$C = -\frac{1}{N}\sum_n \left[y \ln a + (1-y)\ln(1-a)\right]$$

$$\frac{\partial C}{\partial w_j} = \frac{1}{N}\sum_n x_j(\sigma(z) - y)$$

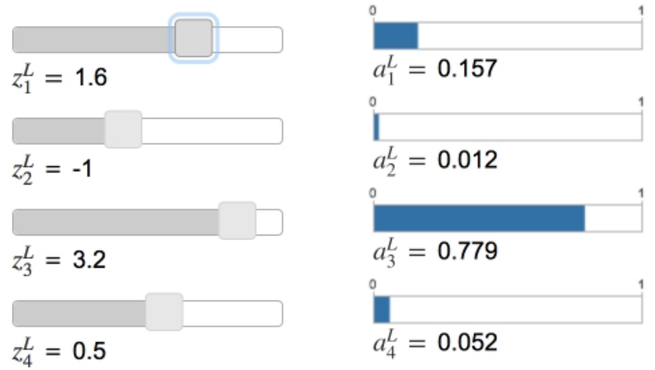
$$\frac{\partial C}{\partial b} = \frac{1}{N}\sum_n (\sigma(z) - y)$$

$$C = -\frac{1}{N}\sum_n \sum_j \left[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)\right]$$

Softmax (activation function) & Log-likelihood (cost function)

- Softmax **output** layer (weighted inputs same as before)
- Example: increase one input, inc. activation but decrees the others, sum over all activations remains 1, **probability distribution**, e.g. MNIST, i.e. simple interpretation of the output activations.
- Log-likelihood cost: Model confident -> high probability (close to 1) -> cost small (other way)
- Classification problems (e.g MNIST)

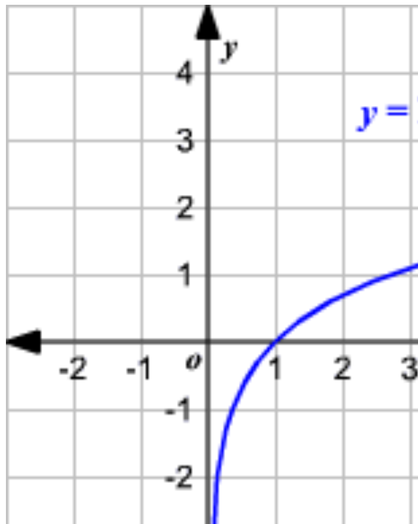
$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \qquad \sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1$$



$$C \equiv -\ln a_y^L$$

$$\frac{\partial C}{\partial w_{kj}^L} = a_j^{L-1}(a_k^L - y_k)$$

$$\frac{\partial C}{\partial b_k^L} = a_k^L - y_k$$



AF - Cost func. combinations

- Sigmoid output layer & quadratic cost
- Linear output layer & quadratic cost
- Sigmoid output layer & cross-entropy cost
- Softmax output layer & log-likelihood cost

Back-propagation: Basics

Problem:

Given a function: $f(x)$ (loss function, weights & biases)

Compute the derivative: $\nabla f(x)$)

Derivative: rate of change of f wrt x, small region around a point

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Operator: $\frac{d}{dx}$ Applied to: $f(x, y)$

Partial derivatives: $\frac{\partial f}{\partial x}$ $\frac{\partial f}{\partial y}$ (straight line, slope)

Back-propagation: Gradients

Eks:

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$$

$$x = 4, y = -3 \qquad f(x,y) = -12$$

$$\frac{\partial f}{\partial x} = -3 \qquad \frac{\partial f}{\partial y} = 4,$$

Sensitivity:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$(f(x+h) = f(x) + h\frac{df(x)}{dx})$$

Increase x with small h -> reduce f with 3h

Increase y with small h -> increase f with 4h

Local process (each gate):

- **Forward pass:**
  - Output value
  - Local gradient
- **Backward pass:**
  - Gradient (final output)
  - Chain rule

Back-propagation

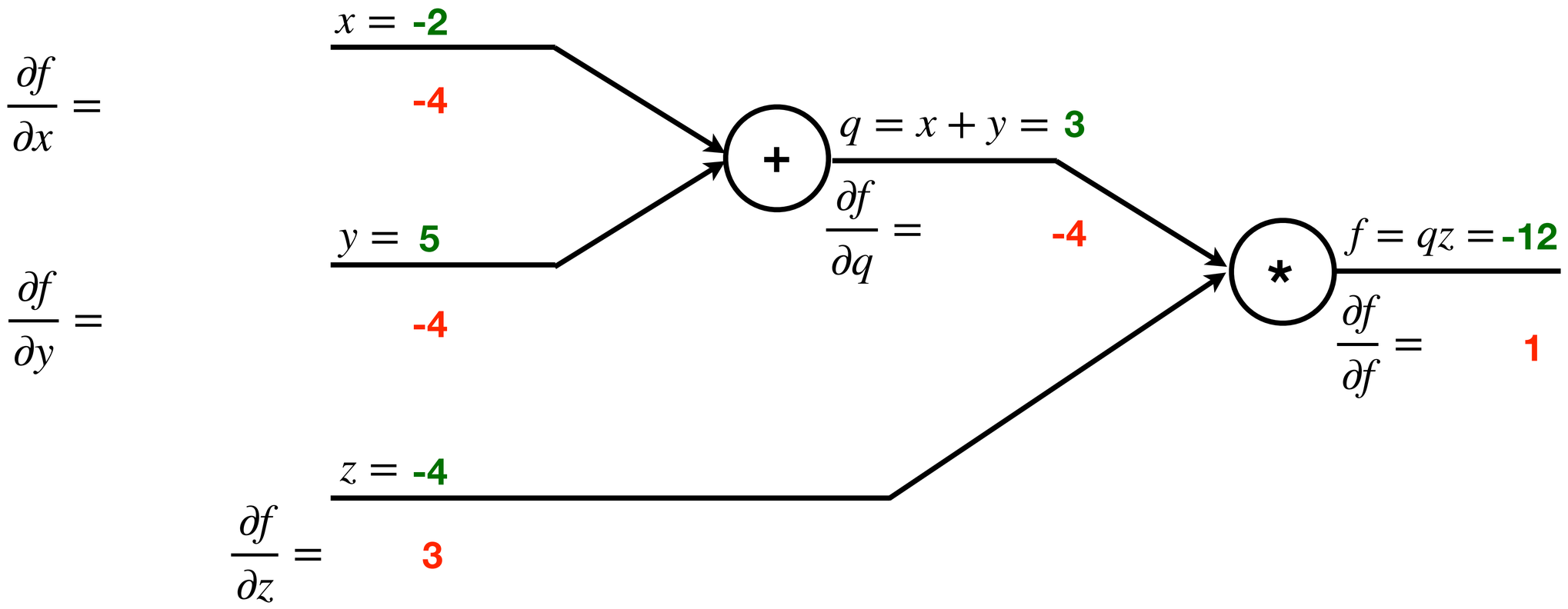
$$f(x, y, z) = f = (x + y)z = qz$$

$$q(x, y) = q = x + y$$

$$x = -2 \quad y = 5 \quad z = -4$$

$$\frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$



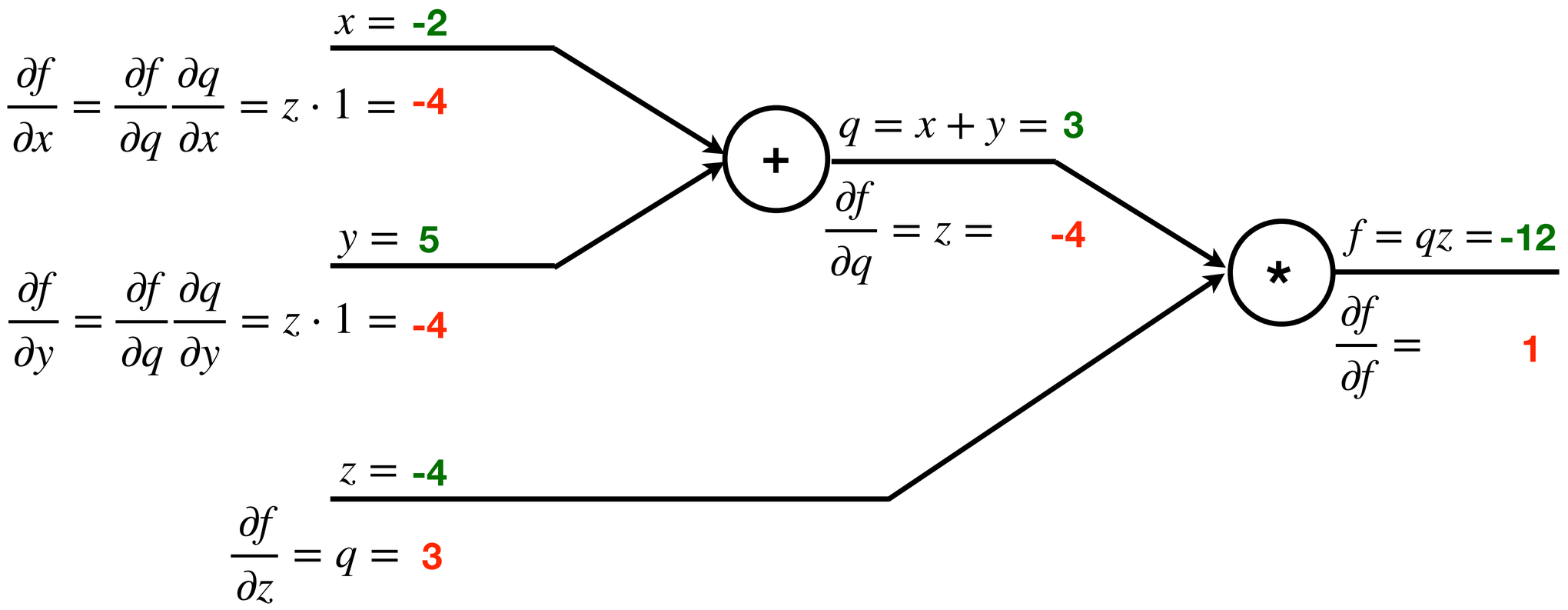
Back-propagation

$\Delta q = 1$ $\quad q : 3 \to 4$ $\quad f = qz = 4 \cdot -4 = -16$ $\quad f : -12 \to -16$ $\quad \Delta f = -4$

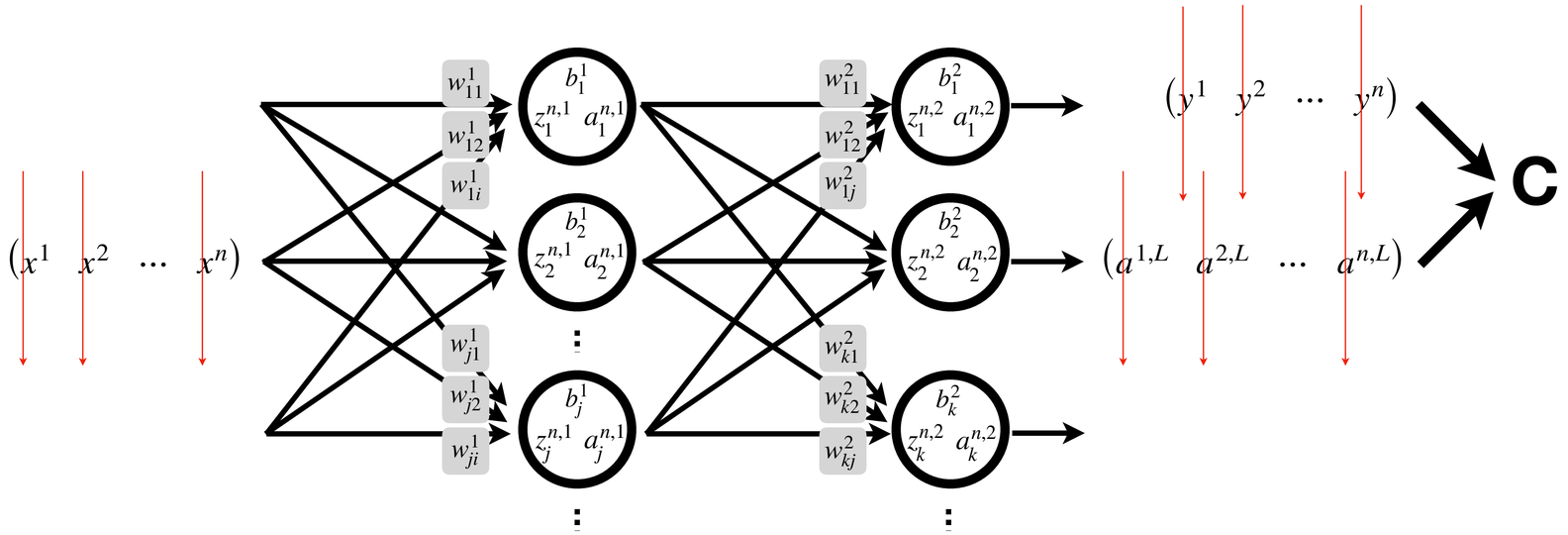
$\Delta z = 1$ $\quad z : -4 \to -3$ $\quad f = (x+y)z = (-2+5) \cdot -3 = 3 \cdot -3 = -9$ $\quad f : -12 \to -9$ $\quad \Delta f = 3$

$\Delta y = 1$ $\quad y : 5 \to 6$ $\quad f = (x+y)z = (-2+6) \cdot -4 = 4 \cdot -4 = -16$ $\quad f : -12 \to -16$ $\quad \Delta f = -4$

$\Delta x = 1$ $\quad x : -2 \to -1$ $\quad f = (x+y)z = (-1+5) \cdot -4 = 4 \cdot -4 = -16$ $\quad f : -12 \to -16$ $\quad \Delta f = -4$



Gradient descent & Backpropagation



Gradient descent:

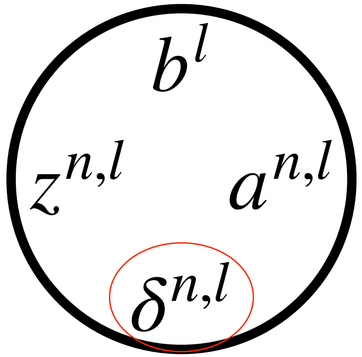
$\partial C/\partial w^l_{ji}$

$\partial C/\partial b^l_j$

- Error (of neuron) = Intermediate quantity
- Comp error & relate to param
- Error large -> Improve cost (sign)
- Error very small -> not much to do

z^l_j	Δz^l_j
$\sigma(z^l_j)$	$\sigma(z^l_j + \Delta z^l_j)$
C	$\frac{\partial C}{\partial z^l_j}\Delta z^l_j$.

Error:



$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$$

For each training example n: x^n

Input activation: $a^{n,0} = x^n$

Feed-forward: $z^{n,l} = w^l a^{n,l-1} + b^l$

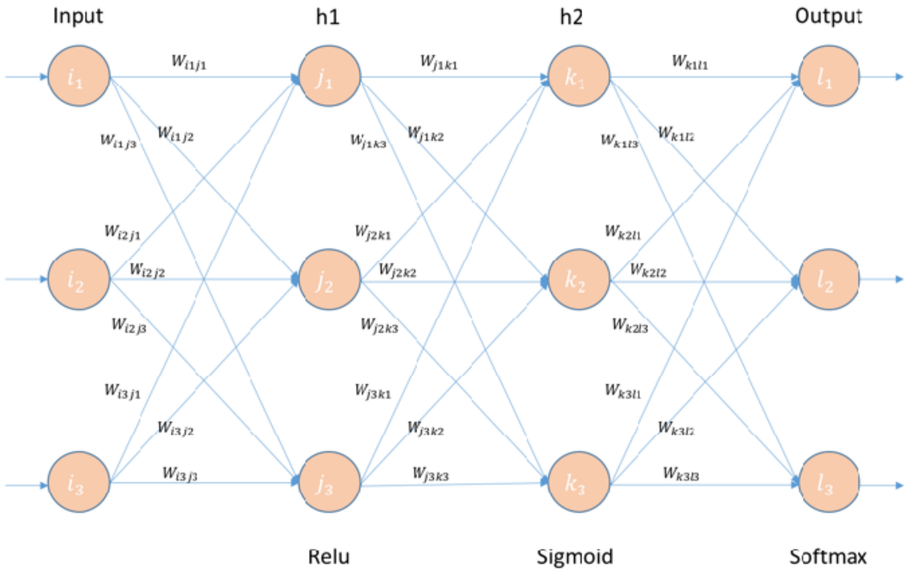
$a^{n,l} = f(z^{n,l}) \quad l = 1,2,\cdots,L$

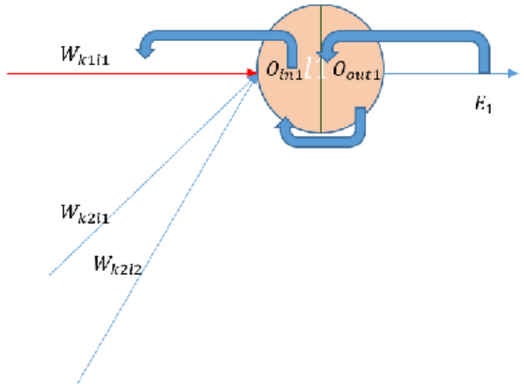
Output error: $\delta^{n,L} = \nabla_a C_n \odot f'(z^{n,L})$

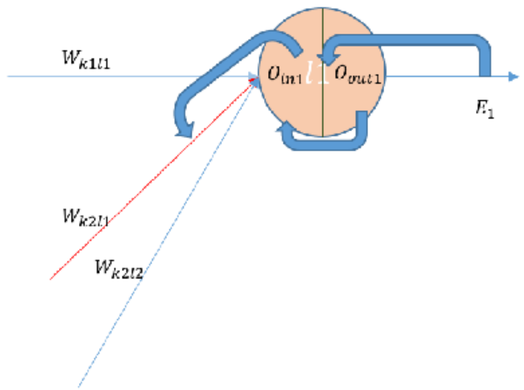
Back-propagate: $\delta^{n,l} = ((w^{l+1})^T \delta^{n,l+1} \odot f'(z^{n,l})$

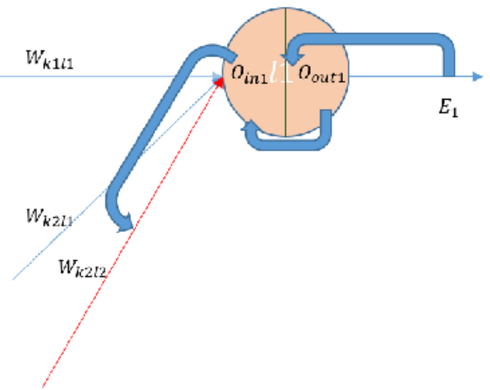
$l = L-1, L-2, \cdots, 1$

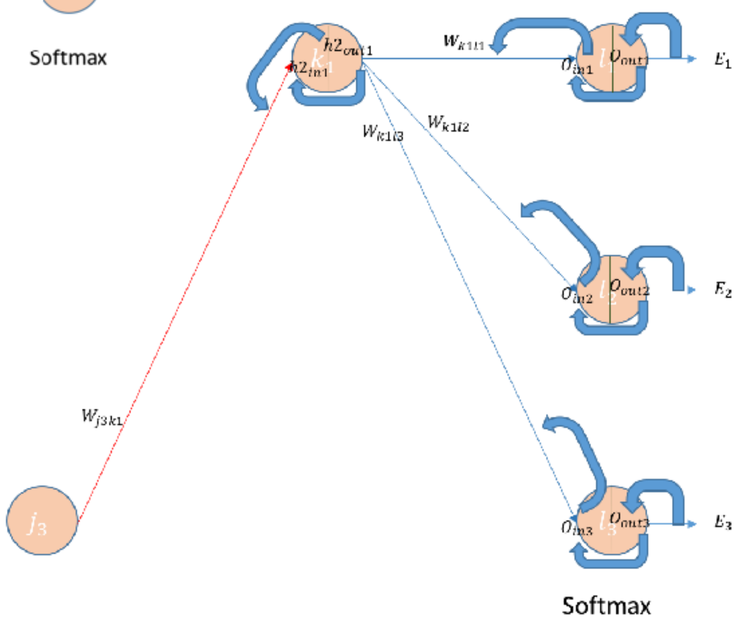
Backward pass

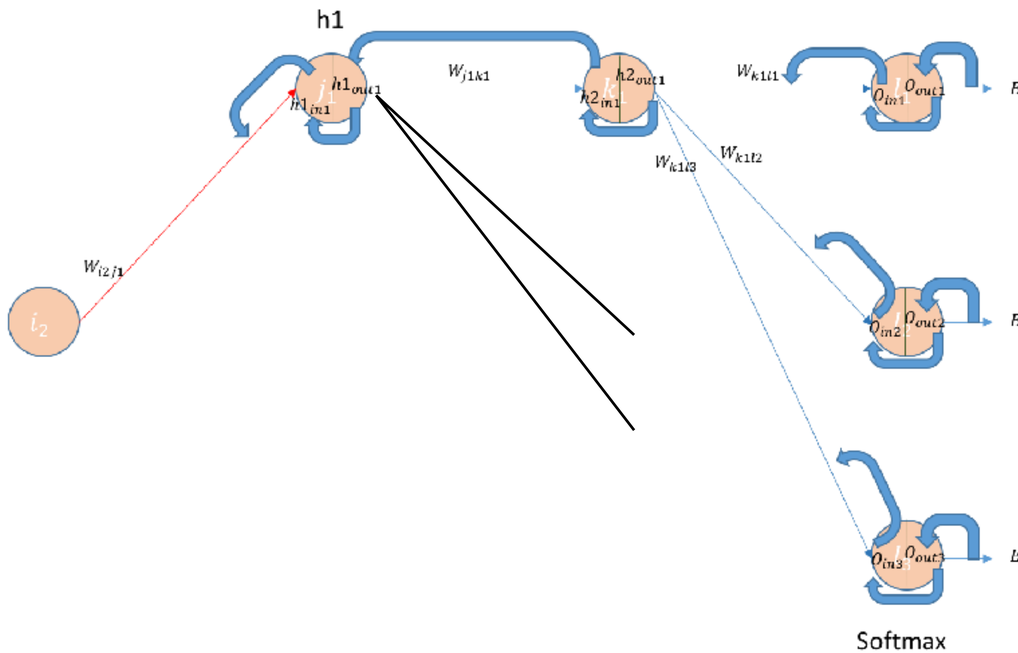












Repetition: Learning (2)

$f : \mathbb{R}^I \to \mathbb{R}^J$

$f(x) = f^{(L)}(f^{(L-1)}(\cdots f^{(1)}(x)))$

$(x^1, y^1), (x^2, y^2), \cdots (x^N, y^N)$

$f(x;\theta) = a^L = \hat{y} \quad \text{=?} \quad y$

$C(\boldsymbol{\theta})$

$\nabla C = \frac{\partial C}{\partial \theta_1}, \frac{\partial C}{\partial \theta_2} \cdots$

$\theta = \theta - \alpha \nabla C$

$\frac{\partial C}{\partial w_{kj}^L} = \frac{\partial C}{\partial z^L}\frac{\partial z^L}{\partial w_{kj}^L} = \delta^L \frac{\partial z^L}{\partial w_{kj}^L}$

$\frac{\partial C}{\partial b_k^L} = \frac{\partial C}{\partial z^L}\frac{\partial z^L}{\partial b_k^L} = \delta^L \frac{\partial z^L}{\partial b_k^L}$

$\delta_k^L = \frac{\partial C}{\partial z_k^L} = \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_k^L} = \frac{\partial C}{\partial a_k^L} f'(z_k^L)$

$a^L = f(z^L)$

$a_k^L = f(z_k^L)$

$\frac{\partial z^L}{\partial w_{kj}^L} = a_j^{L-1}$

$\frac{\partial z^L}{\partial b_k^L} = 1$

$z^L = w^L a^{L-1} + b^L$

$z_k^L = \sum_j w_{kj}^L a_j^{L-1} + b^L$

$\frac{\partial C}{\partial w_{kj}^L} = \frac{\partial C}{\partial a_k^L} f'(z_k^L)\frac{\partial z_k^L}{\partial w_{kj}^L} = \delta_k^L a_j^{L-1}$

$\frac{\partial C}{\partial b_k^L} = \frac{\partial C}{\partial a_k^L} f'(z_k^L)\frac{\partial z_k^L}{\partial b_k^L} = \delta_k^L$

SGD / BP

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L)$$

$$\delta^L = \nabla_a C \odot f'(z^L)$$

$$C = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

$$\partial C / \partial a_j^L = (a_j^L - y_j) \quad \nabla_a C = (a^L - y)$$

$$\delta^L = \nabla_a C \odot f'(z^L) = (a^L - y) \odot f'(z^L)$$

- For multiple epochs of training:
  - Generate a mini-batch of N training examples
  - For each training example n: x^n
    - Input activation: $a^{n,0} = x^n$
    - Forward pass / Feedforward the activation: $z^{n,l} = w^l a^{n,l-1} + b^l$

      $a^{n,l} = f(z^{n,l}) \qquad l = 1,2,\cdots,L$

      
    - Output error: $s \odot t$

      $(s \odot t)_j = s_j t_j$

      $\begin{bmatrix}1\\2\end{bmatrix} \odot \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1*3\\2*4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$

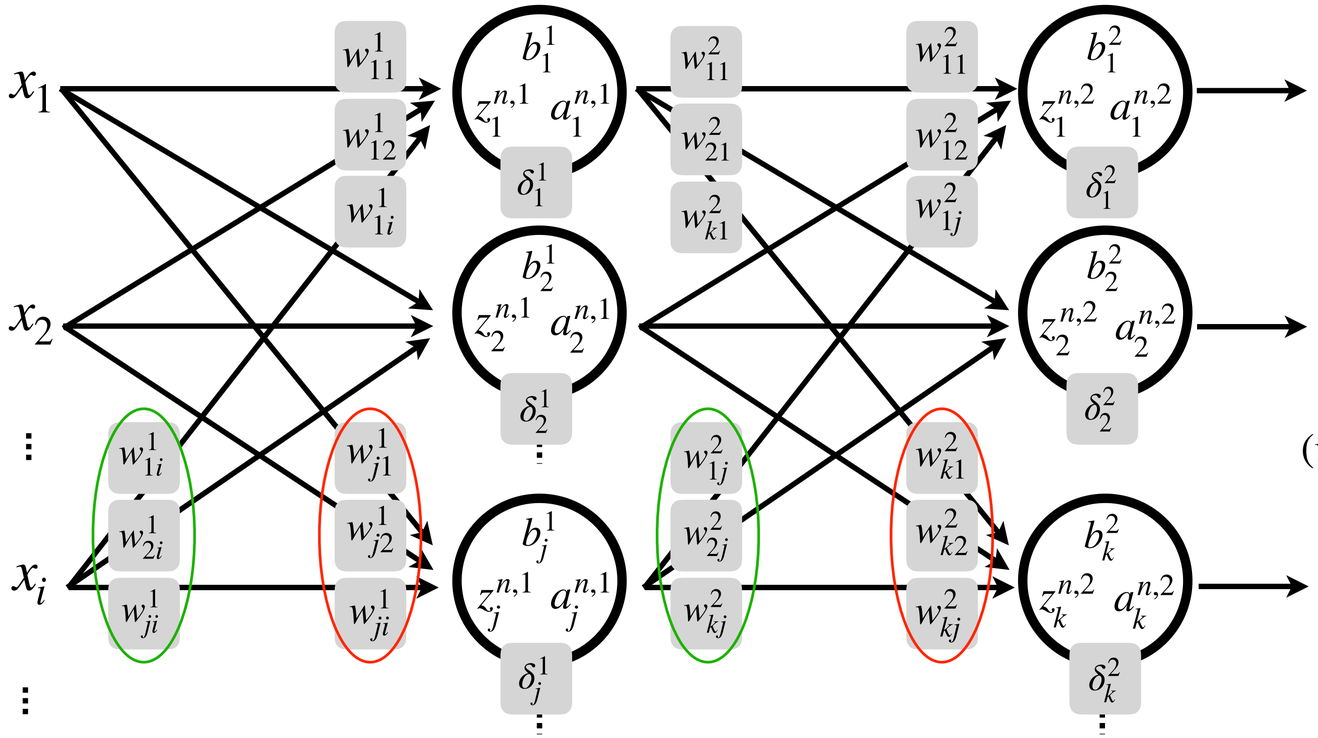
      $\delta^{n,L} = \nabla_a C_n \odot f'(z^{n,L}) \qquad \delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$
    - Backward pass / Backpropagate the error:

      $\delta^{n,l} = ((w^{l+1})^T \delta^{n,l+1} \odot f'(z^{n,l})$

      $l = L-1, L-2, \cdots, 1$
  - Gradient descent:

    $b^l \to b^l - \frac{\eta}{N}\sum_n \delta^{n,l} \qquad \frac{\partial C}{\partial b_j^l} = \delta_j^l$

    $l = L, L-1, \ldots, 1 \qquad w^l \to w^l - \frac{\eta}{N}\sum_n \delta^{n,l}(a^{n,l-1})^T \qquad \frac{\partial C}{\partial w_{ji}^l} = a_i^{l-1}\delta_j^l$



$$w^1 = \begin{pmatrix} w^1_{11} & w^1_{12} & \cdots & w^1_{1i} & \cdots \\ w^1_{21} & w^1_{22} & \cdots & w^1_{2i} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{j1} & w^1_{j2} & \cdots & w^1_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$(w^1)^T = \begin{pmatrix} w^1_{11} & w^1_{21} & \cdots & w^1_{j1} & \cdots \\ w^1_{12} & w^1_{22} & \cdots & w^1_{j2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{1i} & w^1_{2i} & \cdots & w^1_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$w^2 = \begin{pmatrix} w^2_{11} & w^2_{12} & \cdots & w^2_{1j} & \cdots \\ w^2_{21} & w^2_{22} & \cdots & w^2_{2j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{k1} & w^2_{k2} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$(w^2)^T = \begin{pmatrix} w^2_{11} & w^2_{21} & \cdots & w^2_{k1} & \cdots \\ w^2_{12} & w^2_{22} & \cdots & w^2_{k2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{1j} & w^2_{2j} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

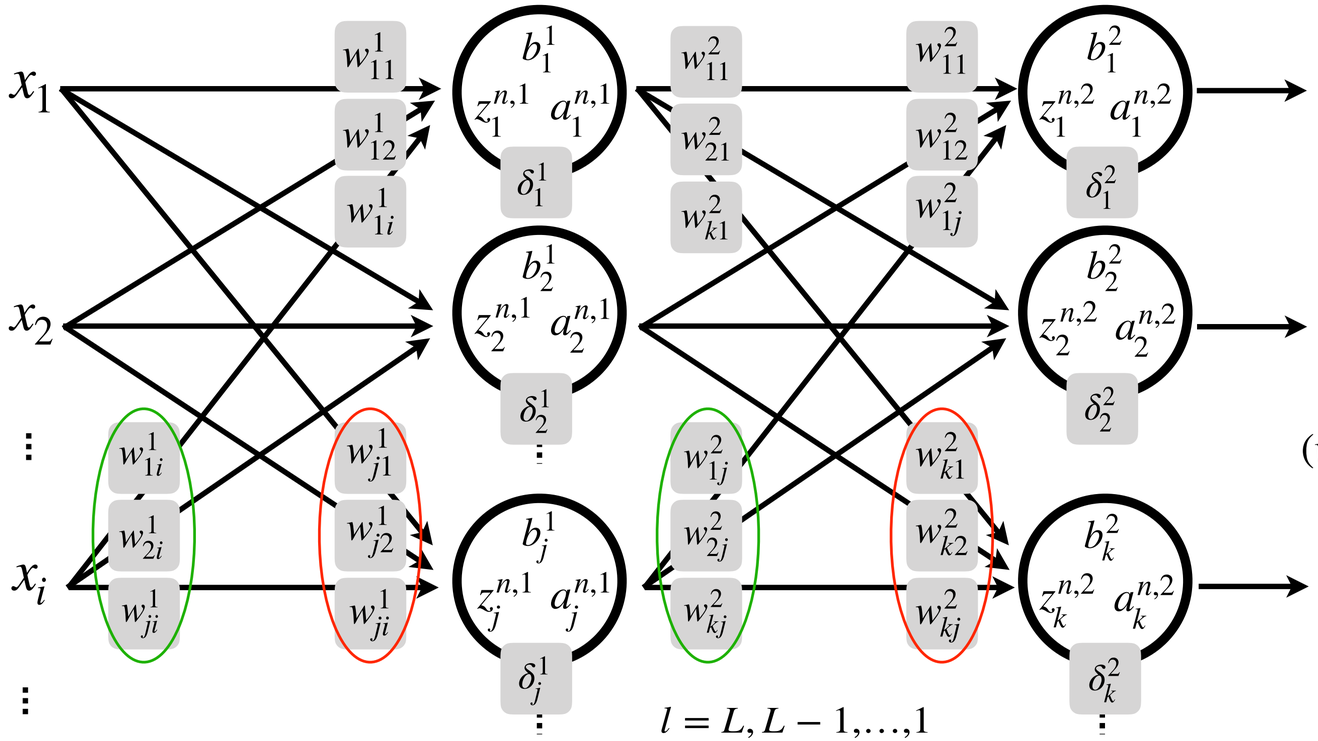
$$\delta^{n,l} = ((w^{l+1})^T \delta^{n,l+1} \odot f'(z^{n,l})$$
$$\delta^l = ((w^{l+1})^T \delta^{l+1} \odot f'(z^l)$$
$$\delta^1 = ((w^2)^T \delta^2 \odot f'(z^1)$$

$$\nabla_a C \quad \frac{\partial C}{\partial a^L_k}$$

$$\delta^{n,L} = \nabla_a C_n \odot f'(z^{n,L})$$
$$\delta^L = \nabla_a C \odot f'(z^L)$$
$$\delta^2 = \nabla_a C \odot f'(z^2)$$

? $$\begin{pmatrix} \delta^1_1 \\ \delta^1_2 \\ \vdots \\ \delta^1_j \\ \vdots \end{pmatrix} = \begin{pmatrix} w^2_{11} & w^2_{21} & \cdots & w^2_{k1} & \cdots \\ w^2_{12} & w^2_{22} & \cdots & w^2_{k2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{1j} & w^2_{2j} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} \delta^2_1 \\ \delta^2_2 \\ \vdots \\ \delta^2_k \\ \vdots \end{pmatrix} \odot f'\left(\begin{pmatrix} z^1_1 \\ z^1_2 \\ \vdots \\ z^1_j \\ \vdots \end{pmatrix} \right)$$

$$\begin{pmatrix} \delta^2_1 \\ \delta^2_2 \\ \vdots \\ \delta^2_k \\ \vdots \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial a^2_1} \\ \frac{\partial C}{\partial a^2_2} \\ \vdots \\ \frac{\partial C}{\partial a^2_k} \\ \vdots \end{pmatrix} \odot f'\left(\begin{pmatrix} z^2_1 \\ z^2_2 \\ \vdots \\ z^2_k \\ \vdots \end{pmatrix} \right)$$



$$w^1 = \begin{pmatrix} w^1_{11} & w^1_{12} & \cdots & w^1_{1i} & \cdots \\ w^1_{21} & w^1_{22} & \cdots & w^1_{2i} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{j1} & w^1_{j2} & \cdots & w^1_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$(w^1)^T = \begin{pmatrix} w^1_{11} & w^1_{21} & \cdots & w^1_{j1} & \cdots \\ w^1_{12} & w^1_{22} & \cdots & w^1_{j2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{1i} & w^1_{2i} & \cdots & w^1_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$w^2 = \begin{pmatrix} w^2_{11} & w^2_{12} & \cdots & w^2_{1j} & \cdots \\ w^2_{21} & w^2_{22} & \cdots & w^2_{2j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{k1} & w^2_{k2} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$(w^2)^T = \begin{pmatrix} w^2_{11} & w^2_{21} & \cdots & w^2_{k1} & \cdots \\ w^2_{12} & w^2_{22} & \cdots & w^2_{k2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{1j} & w^2_{2j} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \qquad \frac{\partial C}{\partial b^l_k} = \delta^l_k$$

$$l = L, L-1, \ldots, 1$$

$$b^l \to b^l - \frac{\eta}{N} \sum_n \delta^{n,l}$$

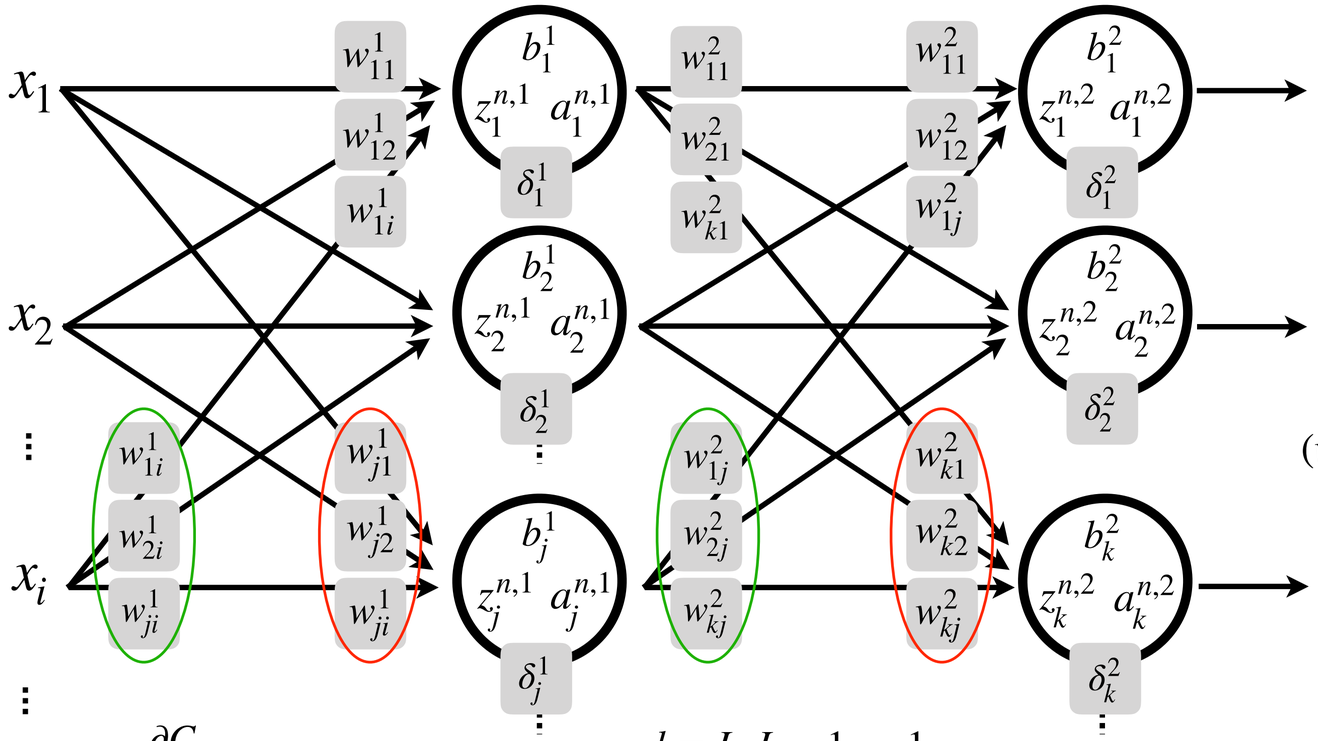
$$b^l \to b^l - \eta\delta^l = b^l - \eta \frac{\partial C}{\partial b^l}$$

$$b^1 \to b^1 - \eta\delta^1 = b^1 - \eta \frac{\partial C}{\partial b^1}$$

$$b^2 \to b^2 - \eta\delta^2 = b^2 - \eta \frac{\partial C}{\partial b^2}$$

$$\begin{pmatrix} b^1_1 \\ b^1_2 \\ \vdots \\ b^1_j \\ \vdots \end{pmatrix} \to \begin{pmatrix} b^1_1 \\ b^1_2 \\ \vdots \\ b^1_j \\ \vdots \end{pmatrix} - \eta \begin{pmatrix} \delta^1_1 \\ \delta^1_2 \\ \vdots \\ \delta^1_j \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} b^2_1 \\ b^2_2 \\ \vdots \\ b^2_k \\ \vdots \end{pmatrix} \to \begin{pmatrix} b^2_1 \\ b^2_2 \\ \vdots \\ b^2_k \\ \vdots \end{pmatrix} - \eta \begin{pmatrix} \delta^2_1 \\ \delta^2_2 \\ \vdots \\ \delta^2_k \\ \vdots \end{pmatrix}$$



$$w^1 = \begin{pmatrix} w^1_{11} & w^1_{12} & \cdots & w^1_{1i} & \cdots \\ w^1_{21} & w^1_{22} & \cdots & w^1_{2i} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{j1} & w^1_{j2} & \cdots & w^1_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$(w^1)^T = \begin{pmatrix} w^1_{11} & w^1_{21} & \cdots & w^1_{j1} & \cdots \\ w^1_{12} & w^1_{22} & \cdots & w^1_{j2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^1_{1i} & w^1_{2i} & \cdots & w^1_{ji} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$w^2 = \begin{pmatrix} w^2_{11} & w^2_{12} & \cdots & w^2_{1j} & \cdots \\ w^2_{21} & w^2_{22} & \cdots & w^2_{2j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{k1} & w^2_{k2} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$(w^2)^T = \begin{pmatrix} w^2_{11} & w^2_{21} & \cdots & w^2_{k1} & \cdots \\ w^2_{12} & w^2_{22} & \cdots & w^2_{k2} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{1j} & w^2_{2j} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\frac{\partial C}{\partial w^l_{ji}} = a^{l-1}_i \delta^l_j = \delta^l_j a^{l-1}_i$$

$$\frac{\partial C}{\partial w^l_{kj}} = a^{l-1}_j \delta^l_k = \delta^l_k a^{l-1}_j$$

$$w^1 \to w^1 - \eta \delta^1 (a^0)^T$$

$$l = L, L-1, \ldots, 1$$

$$w^l \to w^l - \frac{\eta}{N} \sum_n \delta^{n,l} (a^{n,l-1})^T$$

$$w^l \to w^l - \eta \delta^l (a^{l-1})^T$$

$$w^2 \to w^2 - \eta \delta^2 (a^1)^T$$

$$\begin{pmatrix} w^2_{11} & w^2_{12} & \cdots & w^2_{1j} & \cdots \\ w^2_{21} & w^2_{22} & \cdots & w^2_{2j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{k1} & w^2_{k2} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \to \begin{pmatrix} w^2_{11} & w^2_{12} & \cdots & w^2_{1j} & \cdots \\ w^2_{21} & w^2_{22} & \cdots & w^2_{2j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w^2_{k1} & w^2_{k2} & \cdots & w^2_{kj} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} - \eta \begin{pmatrix} \delta^2_1 \\ \delta^2_2 \\ \vdots \\ \delta^2_k \\ \vdots \end{pmatrix} \begin{pmatrix} a^1_1 & a^1_2 & \cdots & a^1_j & \cdots \end{pmatrix}$$

A practical step-by-step approach

Learning

- Neural networks deal with parametric learning
  - Learn an appropriate combination of parameters $\boldsymbol{\theta}$ (weights $\boldsymbol{w}$ + biases $\boldsymbol{b}$)
- Learn by examples
  - Parameters updated by the use of some optimization method
  - Examples guide the optimization method in the right direction
- Optimization methods
  - (Full (batch)) Gradient Descent
    - All training examples used before a single update of the parameters
    - A single batch containing all examples
    - Expensive
  - Mini-Batch Gradient Descents
    - Some examples (mini-batch size) responsible for a single update of the parameters
    - Parameters must be updated many times
  - (Single-example (Stochastic) Gradient Decent)

Cost Functions

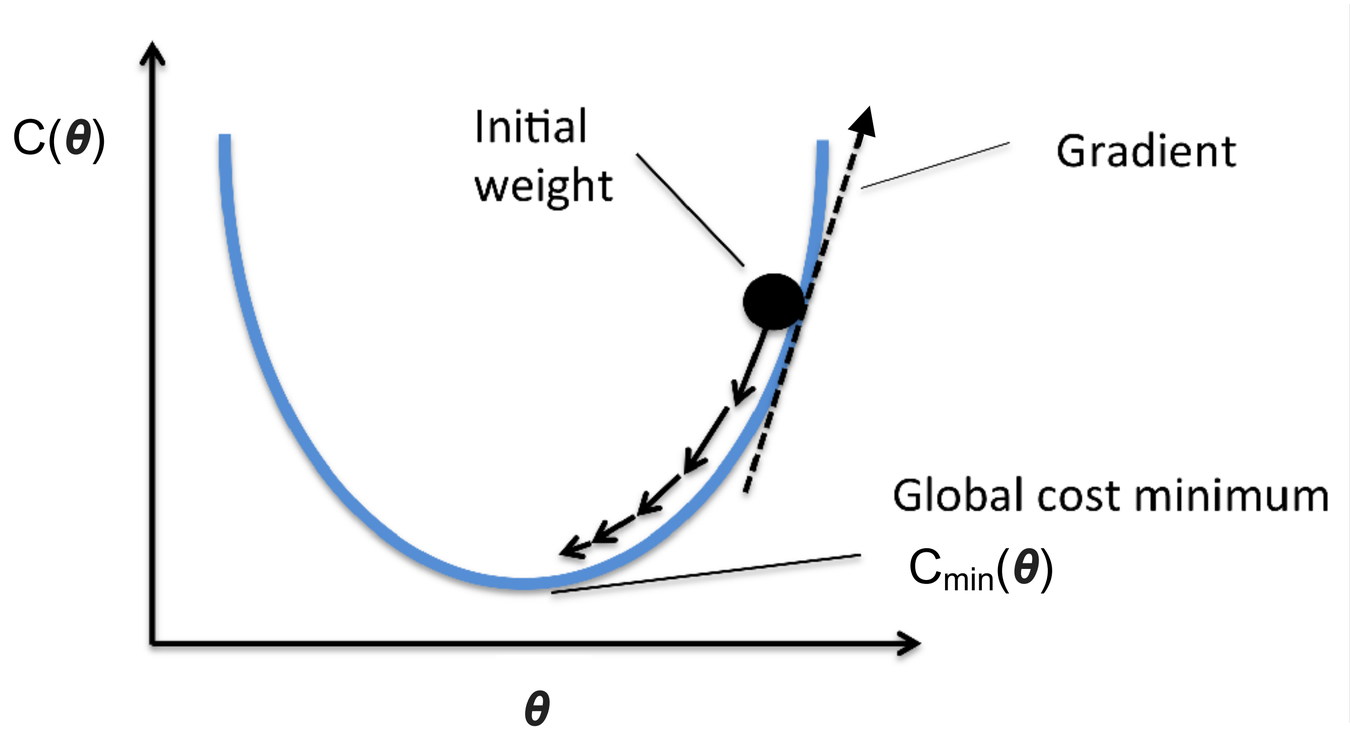
- How to determine if the model has chosen appropriate parameters $\boldsymbol{\theta}$ ($\boldsymbol{w}$, $\boldsymbol{b}$)?
  - A cost function $C(\boldsymbol{\theta})$ measures the strength of the model
    - A good choice of parameters should result in a relative low cost (minimize cost function)
  - Compute the cost function as a function of the prediction $\hat{\boldsymbol{y}}$ and the provided optimal value ($\boldsymbol{y}$)

Regression:
Mean absolute error:
Mean squared error (MSE)

Classification:
Cross entropy (logarithmic)
Log-likelihood

Gradient Descent

- Learning by moving the parameters in the opposite direction of the gradient of the cost function with respect to the parameters $\boldsymbol{\theta}$
  - The gradient defines the direction of most increase (partial derivative)
  - Move in opposite direction of the gradient as we want to obtain the minimum cost



Gradient Descent

- Rule for updating the parameters $\boldsymbol{\theta}$ given cost function $C(\boldsymbol{\theta})$ and learning rate $\boldsymbol{\lambda}$
  - Gradient of $C(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ = partial derivatives of $C(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$

$$\theta' = \theta - \lambda \nabla_\theta C(\theta)$$

$$= \theta - \lambda \frac{\partial C(\theta)}{\partial \theta}$$

Backpropagation

- Learning in neural networks enabled by backpropagating the gradients
- Algorithmic overview:
  - 1. Input ***x***
  - 2. Feed forward to compute ***z*** and ***a***
  - 3. Output error (equation 1)
  - 4. Backpropagate the error (equation 2)
  - 5. Update parameters (equation 3 and 4)

x: input vector

w^l: weight matrix of layer l

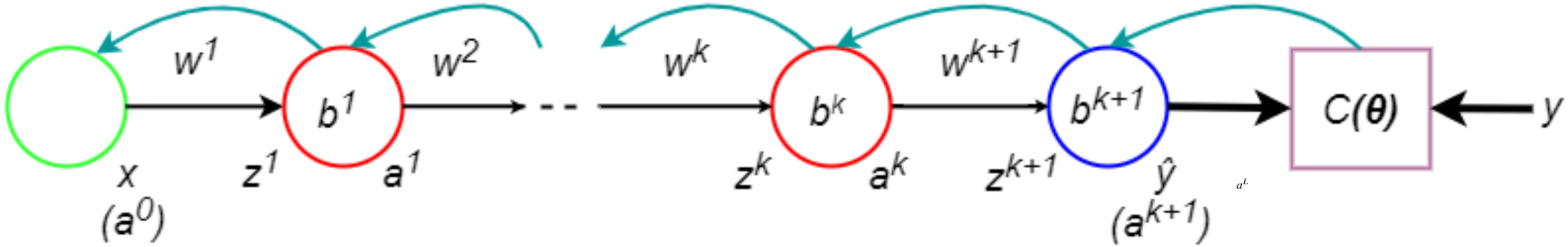
w_{ji}^l: weight between neuron j of layer l and neuron i of layer $(l-1)$

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector



Backpropagation

- Apply chain rule to backpropagate errors

$$\frac{\partial C}{\partial w_{ji}{}^{k+1}} = \frac{\partial C}{\partial z^{k+1}} \frac{\partial z^{k+1}}{\partial w_{ji}{}^{k+1}}$$

x: input vector

w^l: weight matrix of layer l

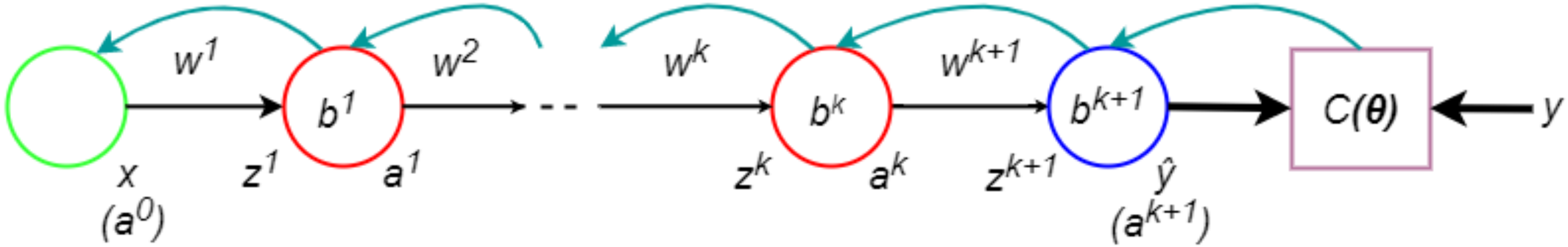
$w_{ji}{}^l$: weight between neuron j of layer l and neuron i of layer $(l-1)$

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector



Backpropagation

x: input vector

w^l: weight matrix of layer l

b^l: bias vector of layer l

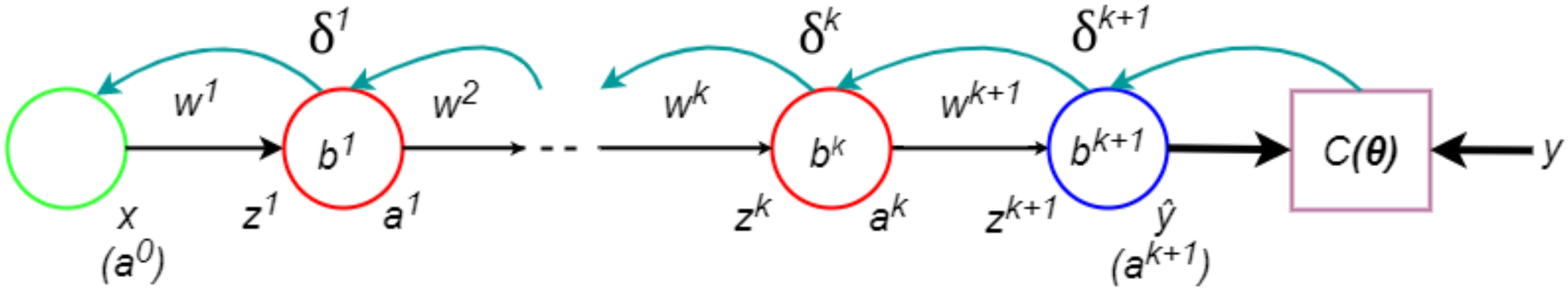
z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

- Delta rule
  - δ^l: error of layer l with respect to cost C
    - δ_j^l: error of neuron j of layer l with respect to cost C
    - Example:

$$\delta^l = \frac{\partial C}{\partial z^l} = \frac{\partial C}{\partial a^l}\frac{\partial a^l}{\partial z^l} = \begin{bmatrix} {\delta_1}^l \\ {\delta_2}^l \end{bmatrix}$$



Backpropagation

- Four equations (matrix-based): $_L$

1. Compute error of output layer *k+1* given cost *C*

$$\delta^{k+1} = \nabla_{\hat{y}} C \odot \nabla_{z^{k+1}} \hat{y}$$

$$\left(s \odot t = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \odot \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} s_1 t_1 \\ s_2 t_2 \end{bmatrix} \right)$$

x: input vector

w^l: weight matrix of layer *l*

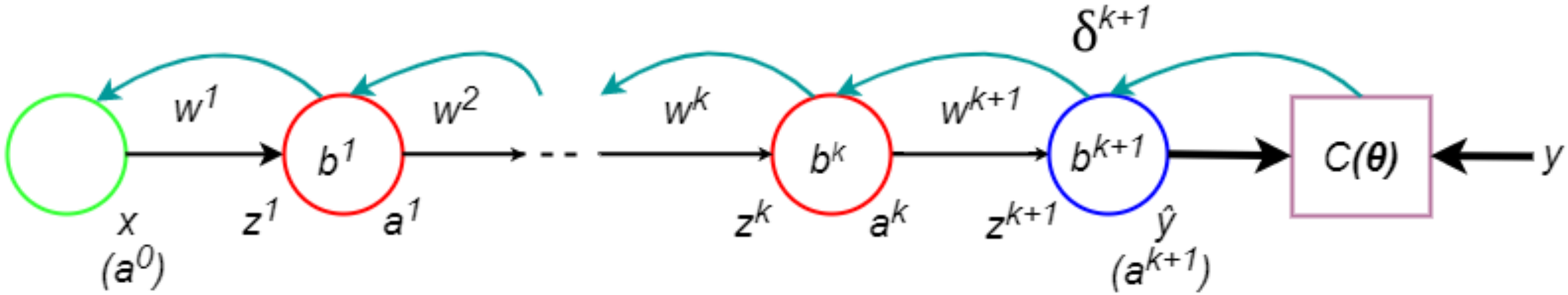
b^l: bias vector of layer *l*

z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*



Backpropagation

- Four equations (matrix-based):

2. Compute error of layer *l* given error of layer (*l+1*)

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \nabla_{z^l} a^l$$

x: input vector

w^l: weight matrix of layer l

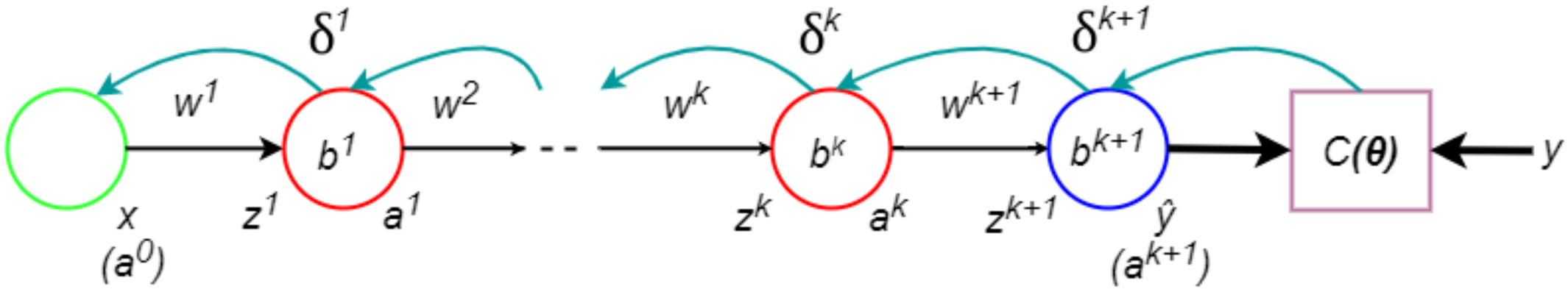
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

- Four equations (matrix-based):

3. Compute gradient of biases of layer *l*
given error of layer *l*

$$\nabla_{b_j{}^l} C = \frac{\partial C}{\partial b_j{}^l} = \delta_j{}^l$$

x: input vector

w^l: weight matrix of layer l

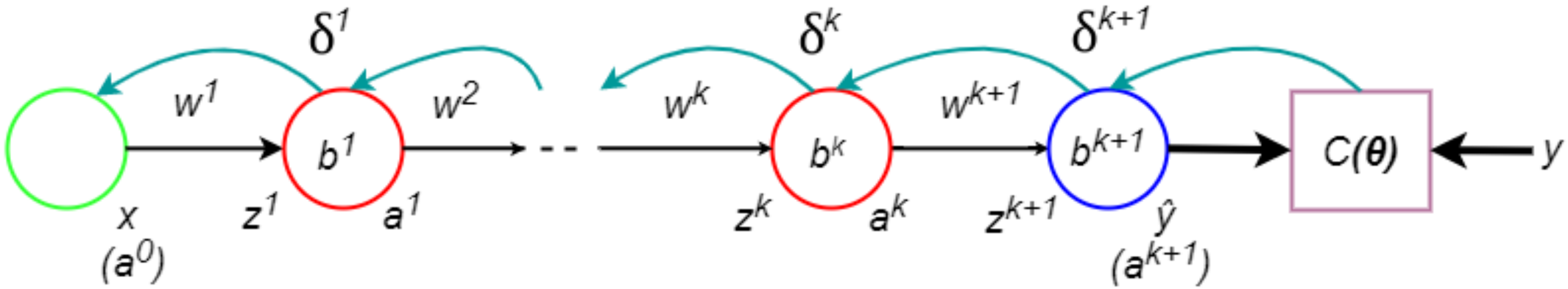
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

- Four equations (matrix-based):

Update biases of layer *l* based on gradient

$$b_j^{l\prime} = b_j^{l} - \lambda \nabla_{b_j{}^l} C = b_j^{l} - \lambda \delta_j^{l}$$

x: input vector

w^l: weight matrix of layer l

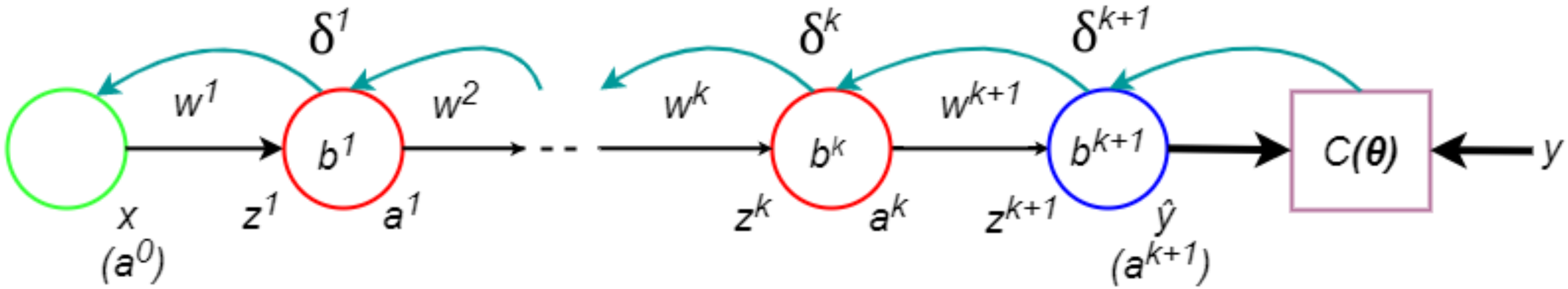
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

- Four equations (matrix-based):

4. Compute gradient of weights of layer *l* given error of layer *l*

$$\nabla_{w_{ji}{}^{l}} C = \frac{\partial C}{\partial w_{ji}{}^{l}} = a_i{}^{l-1} \delta_j{}^{l}$$

x: input vector

w^l: weight matrix of layer *l*

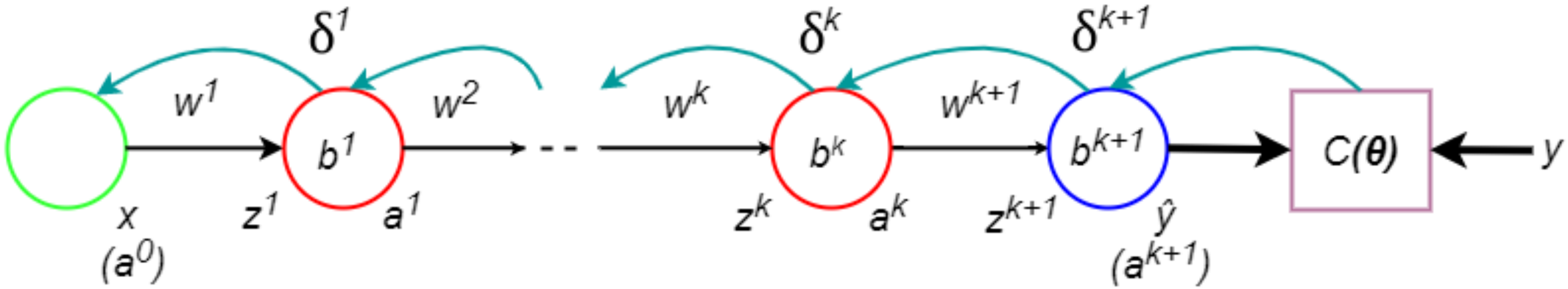
b^l: bias vector of layer *l*

z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*



Backpropagation

- Four equations (matrix-based):

Update weights of layer *l* based on gradient

x: input vector

w^l: weight matrix of layer *l*

b^l: bias vector of layer *l*

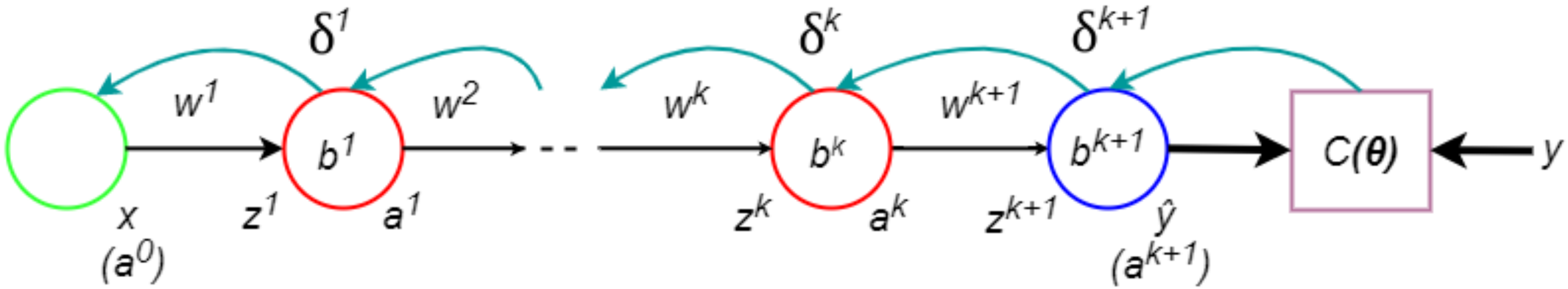
z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*

$$w_{ji}{}^{l\prime} = w_{ji}{}^{l} - \lambda \nabla_{w_{ji}{}^{l}} C = w_{ji}{}^{l} - \lambda a_i{}^{l-1} \delta_j{}^{l}$$



Backpropagation

- Given cost C, compute error δ^{k+1} by eq. 1
- Compute gradients of b^{k+1} and w^{k+1} by eq. 3 and 4
- Update b^{k+1} and w^{k+1}

x: input vector

w^l: weight matrix of layer l

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

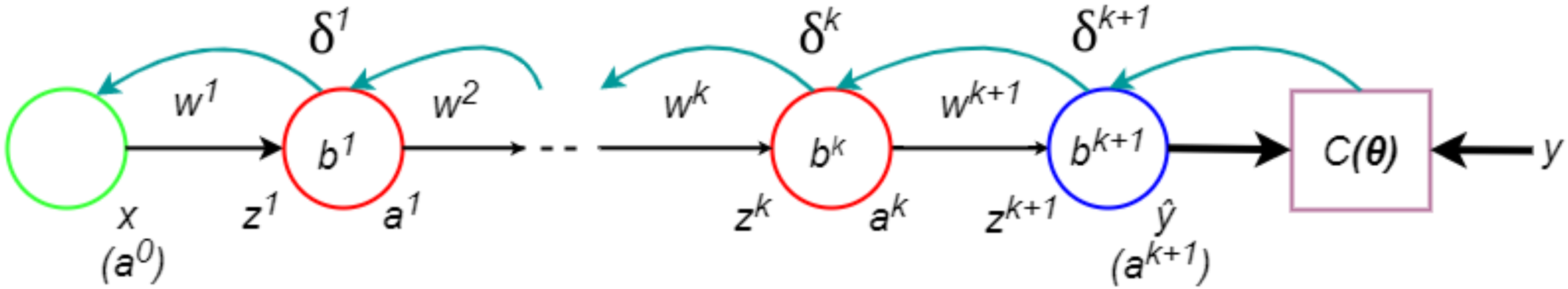
$\hat{y}$: output vector

δ^l: error of layer l

$$\delta^{k+1} = \nabla_{\hat{y}} C \odot \nabla_{z^{k+1}} \hat{y} \quad (1)$$

$$\nabla_{b_j{}^l} C = \delta_j{}^l \quad (2)$$

$$\nabla_{w_{ji}{}^l} C = a_i{}^{l-1} \delta_j{}^l \quad (3)$$



Backpropagation

- Given δ^{k+1}, compute error δ^{k} by eq. 2
- Compute gradients of b^k and w^k by eq. 3 and 4
- Update b^k and w^k

x: input vector

w^l: weight matrix of layer l

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

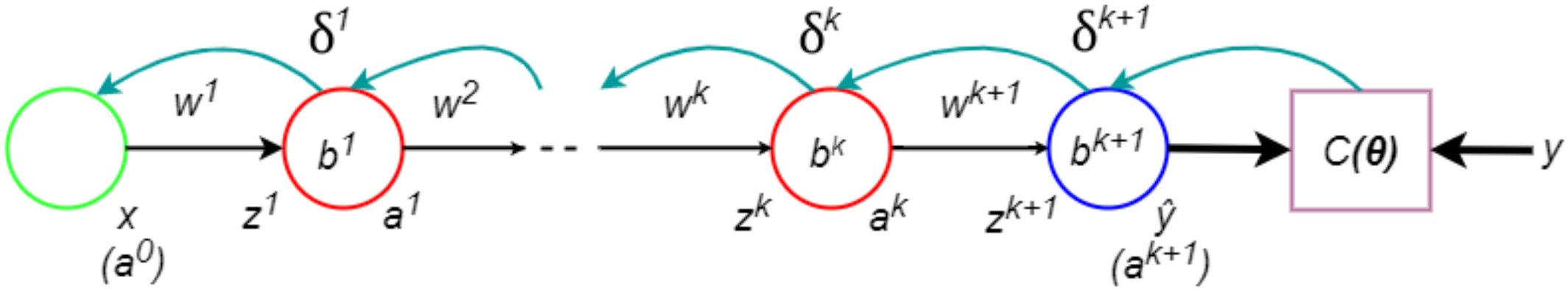
$\hat{y}$: output vector

δ^l: error of layer l

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \nabla_{z^l} a^l \quad (1.)$$

$$\nabla_{b_j{}^l} C = \delta_j{}^l \quad (2.)$$

$$\nabla_{w_{ji}{}^l} C = a_i{}^{l-1} \delta_j{}^l \quad (3.)$$



Backpropagation

- Proceed until parameters b^1 and w^1 of the first layer ($l = 1$) of the network are updated

x: input vector

w^l: weight matrix of layer l

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

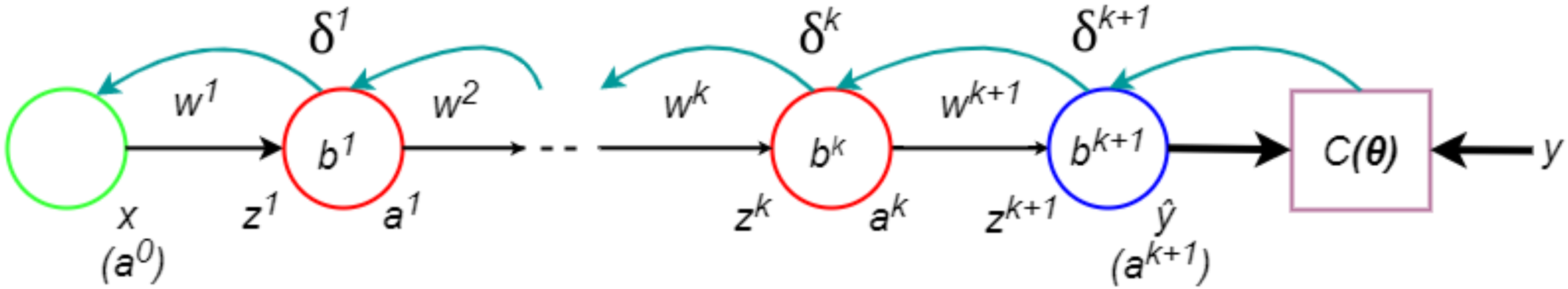
$\hat{y}$: output vector

δ^l: error of layer l

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \nabla_{z^l} a^l$$ 1.

$$\nabla_{b_j{}^l} C = \delta_j{}^l$$ 2.

$$\nabla_{w_{ji}{}^l} C = a_i{}^{l-1} \delta_j{}^l$$ 3.



Numerical Examples (NEs)

Activation function: $f(z) = \frac{1}{1+e^{-z}}$

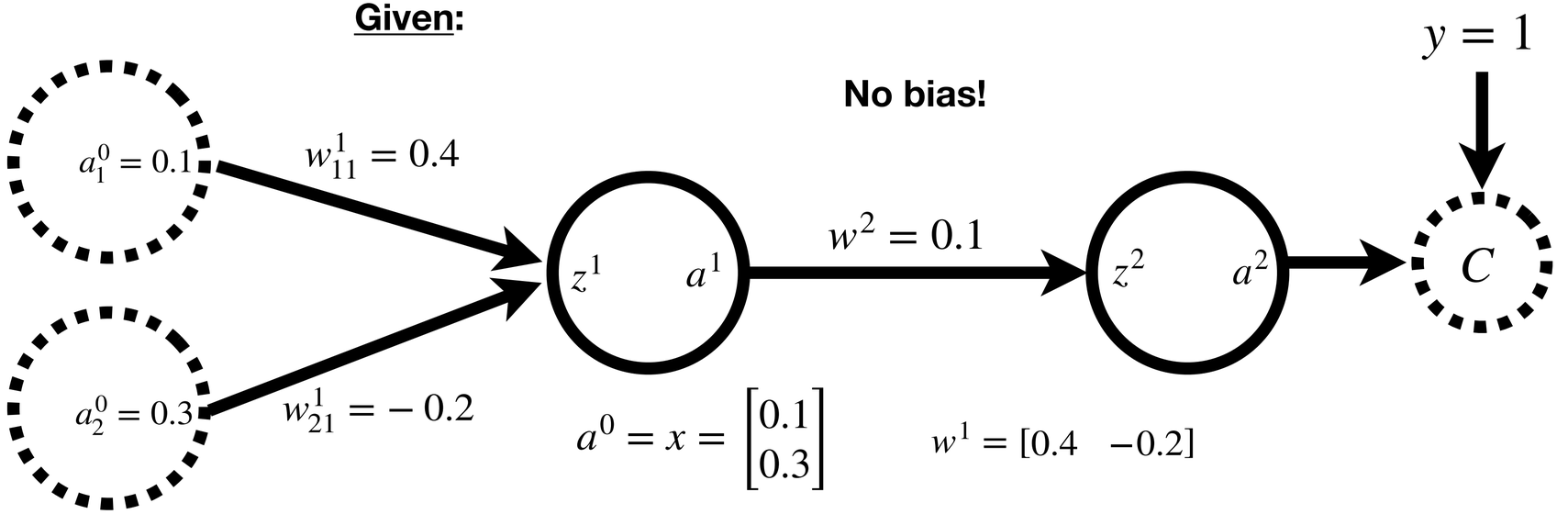
$$\frac{df(z)}{dz} = f(z)(1-f(z))$$

Cost function: $C = C(a^2) = \frac{1}{2}(y-a^2)^2$

$$\frac{\partial C}{\partial a^2} = a^2 - y$$

Learning rate: $\eta = 0.01$

Given:



$$a^0 = x = \begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix} \qquad w^1 = [0.4 \quad -0.2]$$

Find:

Forward pass:

$$z^1 = w^1 a^0 + b^1 = [0.4 \quad -0.2] \begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix} + 0 = 0.04 - 0.06 = -0.02$$

$$a^1 = f(z^1) = f(-0.02) = 0.495$$

$$z^2 = w^2 a^1 + b^2 = 0.1 \cdot 0.495 + 0 = 0.0495$$

$$a^2 = f(z^2) = f(0.0495) = 0.5124$$

Backward pass:

$$\delta^2 = \frac{\partial C}{\partial a^2} f'(z^2) = (a^2 - y) f'(z^2) = (0.5124 - 1) \cdot (0.5124 \cdot (1 - 0.5124)) = -0.4876 \cdot 0.5124 \cdot 0.4876 = -0.1218$$

$$\delta^1 = (w^2)^T \delta^2 \cdot f'(z^1) = 0.1 \cdot (-0.1218) \cdot f'(-0.02) = 0.1 \cdot (-0.1218) \cdot 0.495 \cdot (1 - 0.495) = -0.01218 \cdot 0.25 = -0{,}003045$$

$$\frac{\partial C}{\partial w^2} = \delta^2 a^1 = 0.495 \cdot -0.1218 = -0.060$$

NB: outer product in matrix notation

$$\frac{\partial C}{\partial w^1} = \delta^1 (a^0)^T = -0{,}003045 \cdot [0.1 \quad 0.3] = -\left[0{,}0003045 \quad 0{,}0009135\right]$$

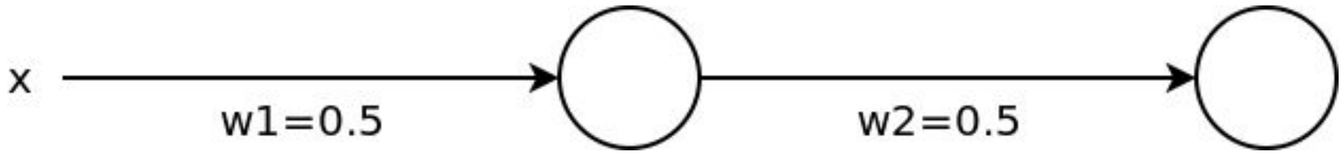
$$w_{new}^2 = w^2 - \eta \frac{\partial C}{\partial w^2} = 0.1 - 0.01 \cdot (-0.06) = 0.1 + 0.0006 = 0.1006$$

$$w_{new}^1 = w^1 - \eta \frac{\partial C}{\partial w^1} = [0.4 \quad -0.2] + 0.01 \left[0{,}0003045 \quad 0{,}0009135\right] = \left[0{,}400003045 \quad -0{,}199990865\right]$$

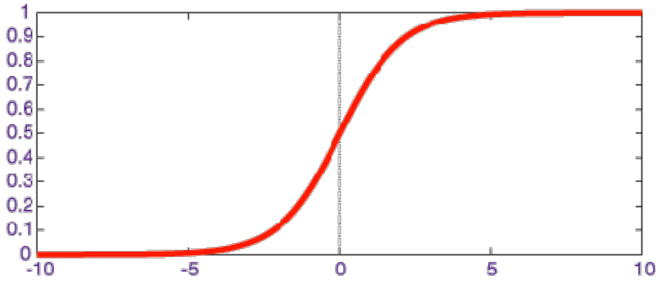
NE1

Backpropagation example

- Simple network
- Input: x=0
- Desired output: y=1
- Initial weights: 0.5



$$f(z) = \frac{1}{1 + e^{-z}}$$



Summary: the equations of backpropagation

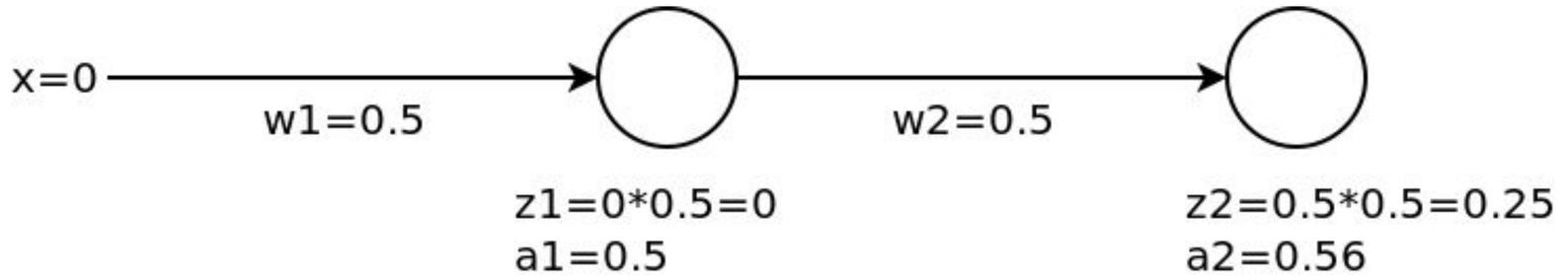
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation example



Summary: the equations of backpropagation

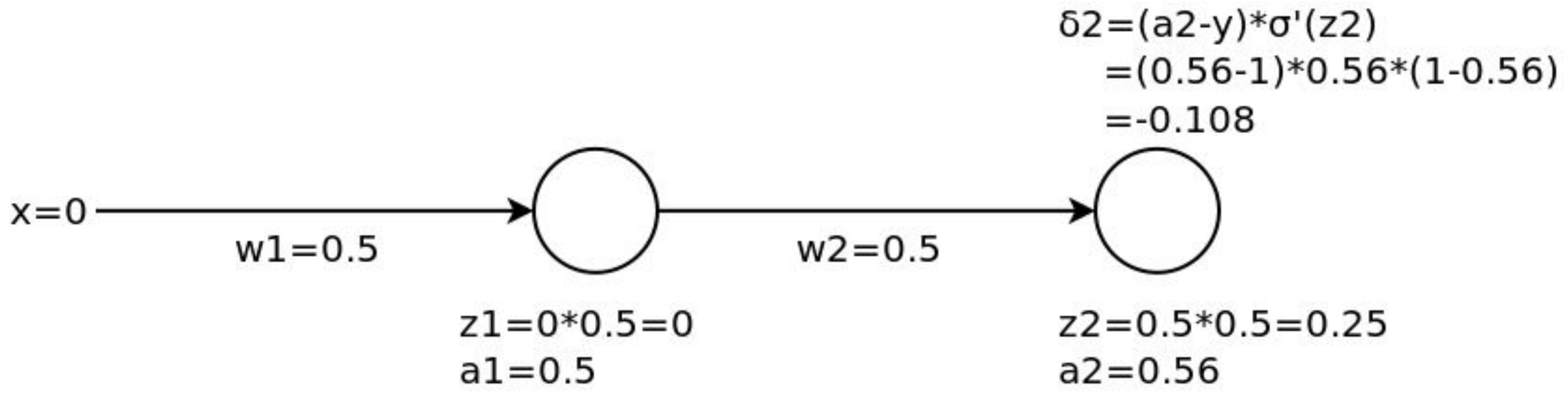
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation example



Summary: the equations of backpropagation

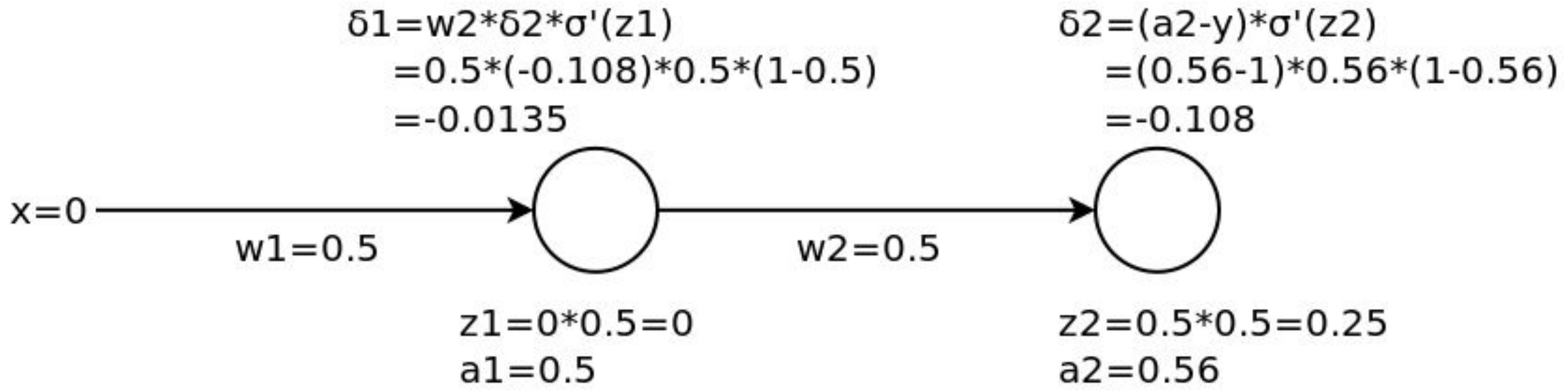
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation example



Summary: the equations of backpropagation

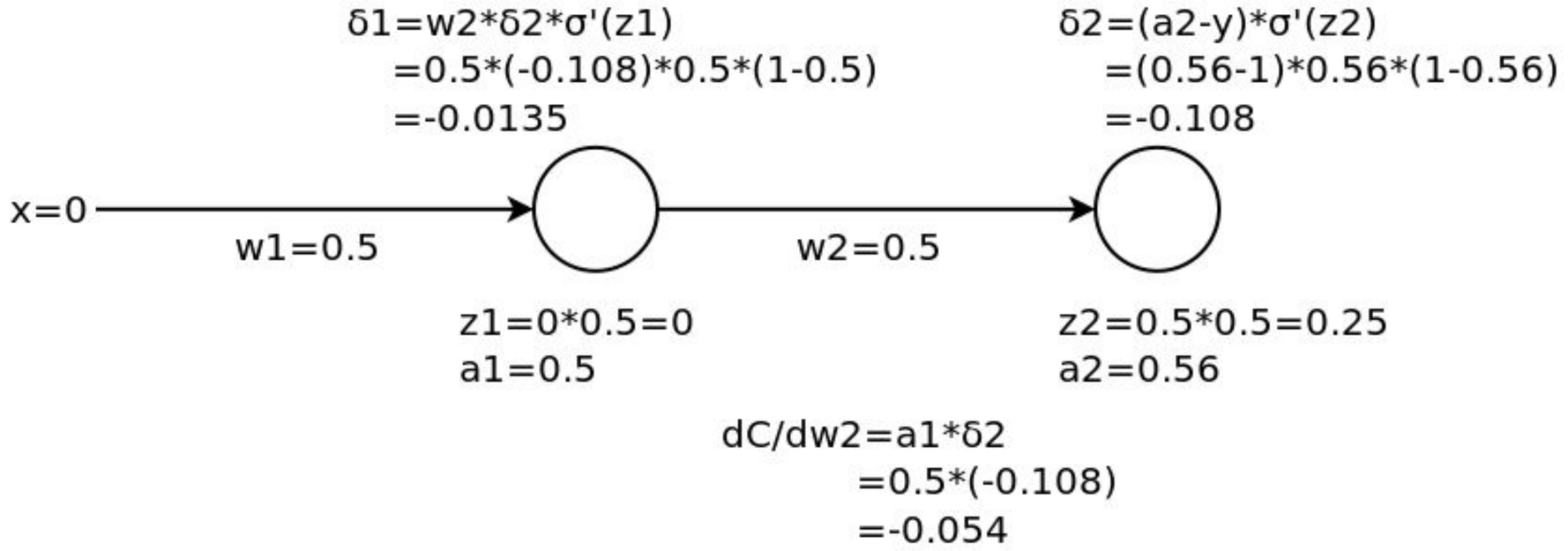
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation example



Summary: the equations of backpropagation

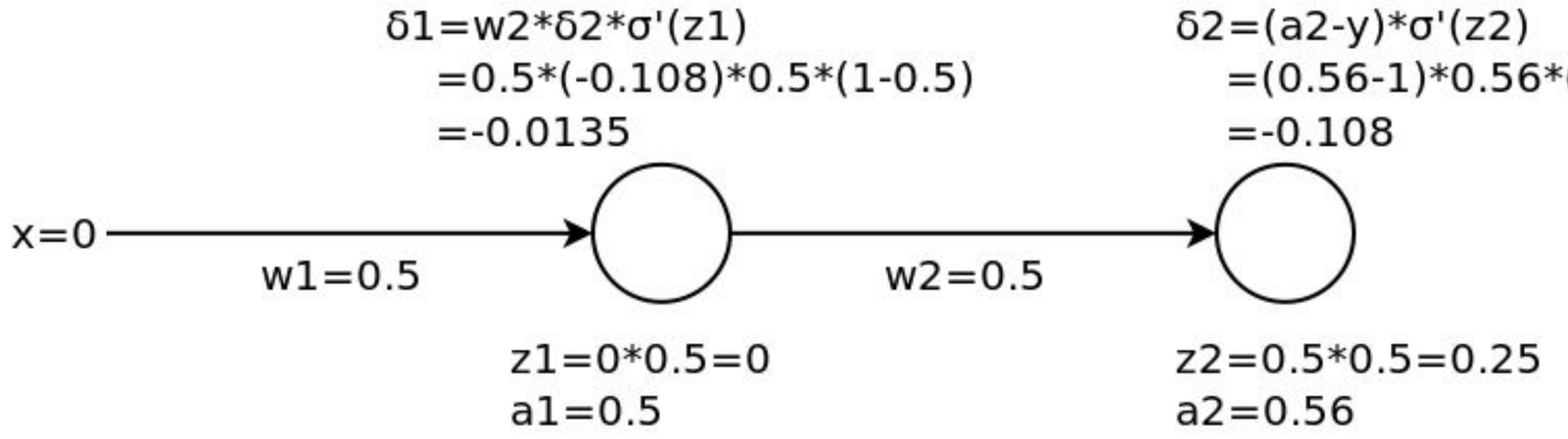
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation example



Summary: the equations of backpropagation

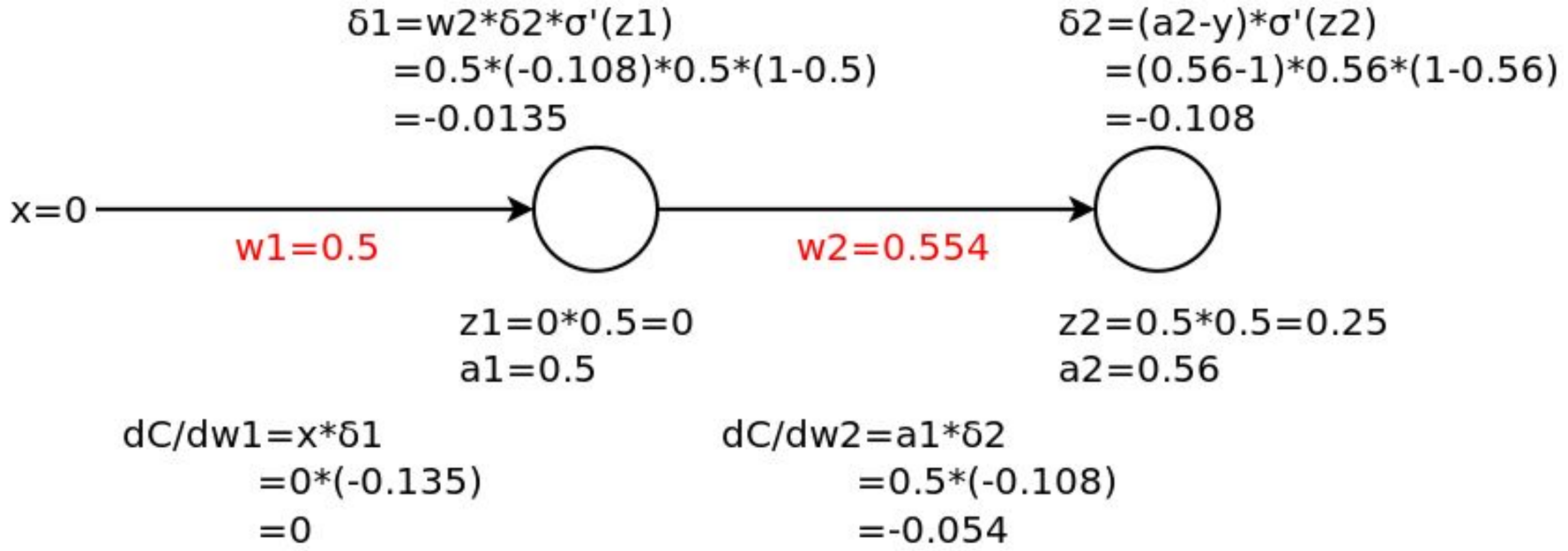
$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

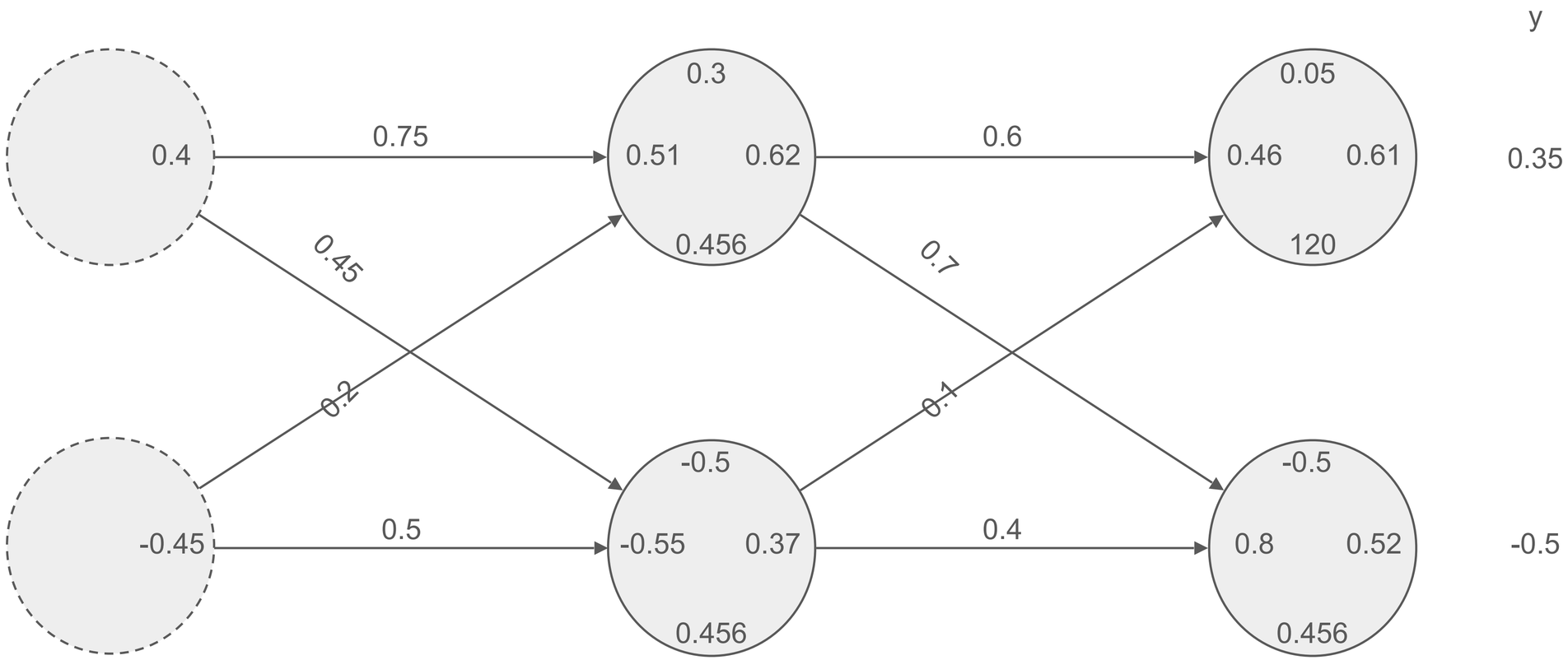
$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Backpropagation example



NE3

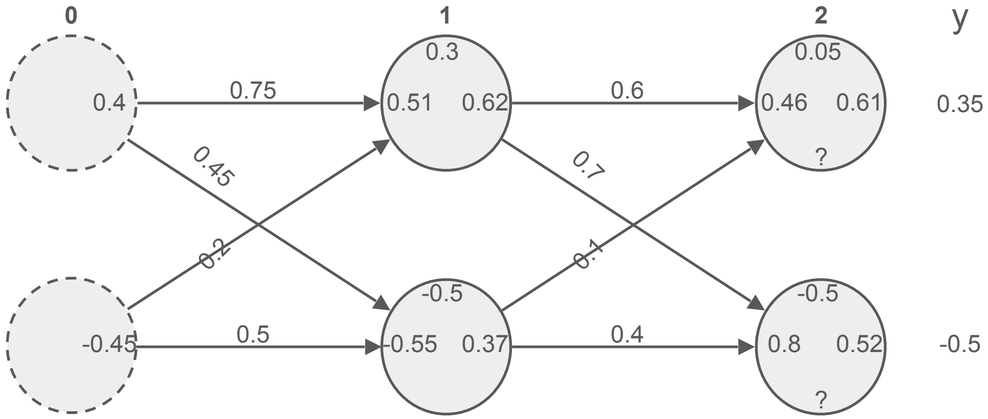
Backpropagation example



Backpropagation example

$$\nabla_a C = (a^L - y) \qquad \sigma'(z^L) = \sigma(z^L) \odot (1 - \sigma(z^L))$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L) = (a^L - y) \odot \sigma(z^L) \odot (1 - \sigma(z^L))$$

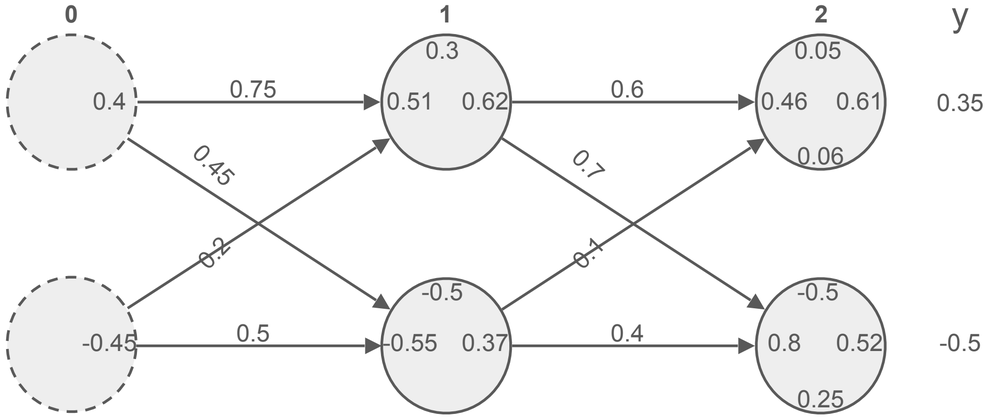


Backpropagation example

$$\delta^L = (a^L - y) \odot \sigma(z^L) \odot (1 - \sigma(z^L))$$

$$\delta^L = \left(\begin{bmatrix} 0.61 \\ 0.52 \end{bmatrix} - \begin{bmatrix} 0.35 \\ -0.5 \end{bmatrix} \right) \odot \left(\sigma \left(\begin{bmatrix} 0.46 \\ 0.8 \end{bmatrix} \right) \odot \left(1 - \sigma \left(\begin{bmatrix} 0.46 \\ 0.8 \end{bmatrix} \right) \right) \right)$$

$$\delta^L = \begin{bmatrix} 0.06 \\ 0.25 \end{bmatrix}$$

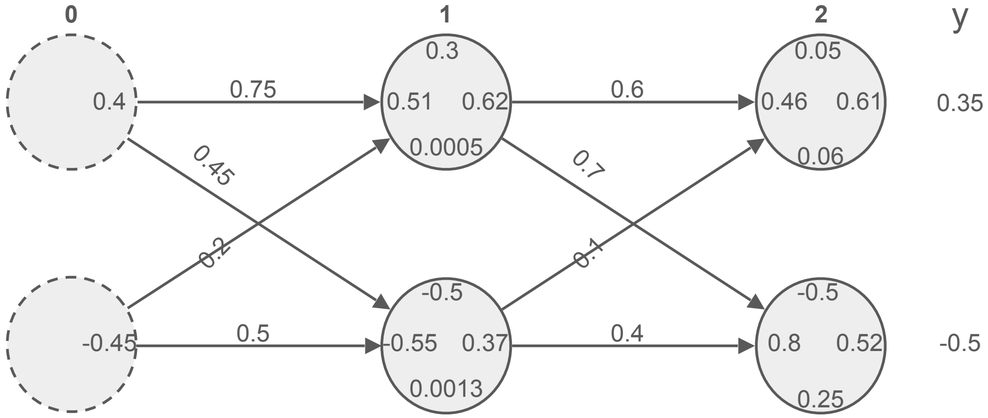


Backpropagation example

$$\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \left(\sigma(z^l) \odot (1 - \sigma(z^l))\right)$$

$$\delta^1 = \left(\left(\begin{bmatrix} 0.6 & 0.1 \\ 0.7 & 0.4 \end{bmatrix}\right)^T \begin{bmatrix} 0.06 \\ 0.25 \end{bmatrix}\right) \odot \left(\sigma\left(\begin{bmatrix} 0.51 \\ 0.55 \end{bmatrix}\right) \odot \left(1 - \sigma\left(\begin{bmatrix} 0.51 \\ 0.55 \end{bmatrix}\right)\right)\right)$$

$$\delta^1 = \begin{bmatrix} 0.0005 \\ 0.0013 \end{bmatrix}$$

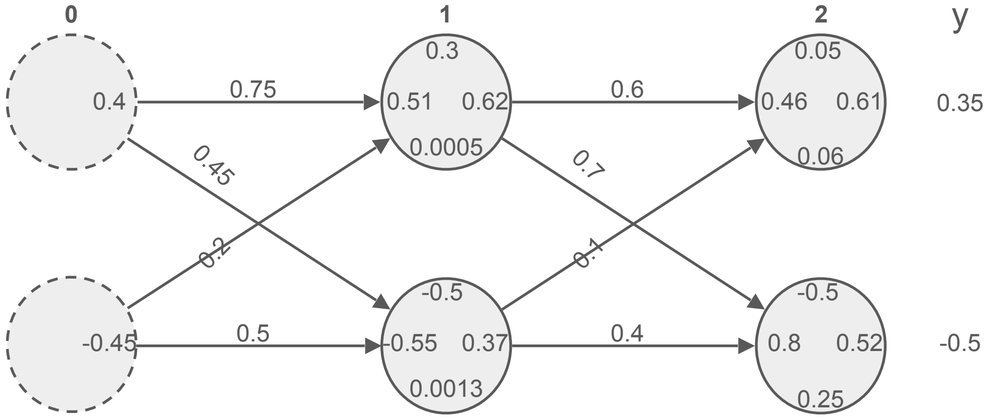


Backpropagation example

$$\delta^1 = \begin{bmatrix} 0.0005 \\ 0.0013 \end{bmatrix} \qquad a^0 = \begin{bmatrix} 0.4 \\ -0.45 \end{bmatrix}$$

$$\frac{\partial C}{\partial w^1} = \begin{bmatrix} a_0^0\delta_0^1 & a_1^0\delta_0^1 \\ a_0^0\delta_1^1 & a_1^0\delta_1^1 \end{bmatrix} = \begin{bmatrix} 0.00022 & -0.00025 \\ 0.00052 & -0.00059 \end{bmatrix}$$

$$\begin{pmatrix} \delta_1^2 \\ \delta_2^2 \\ \vdots \\ \delta_k^2 \\ \vdots \end{pmatrix} \begin{pmatrix} a_1^1 & a_2^1 & \cdots & a_j^1 & \cdots \end{pmatrix}$$



NE4

Backpropagation

- Fully connected network
  - 2 inputs
  - One hidden layer (shallow ANN) of 2 neurons
  - 1 output
- Sigmoid activation

$$a^1 = \sigma(z^1)$$

$$\hat{y} = \sigma(z^2)$$

- MSE cost function (1 example)

$$C(\theta) = (\hat{y} - y)^2$$

Usually, not here: $L = C = \frac{(y-a)^2}{2}$

x: input vector

w^l: weight matrix of layer l

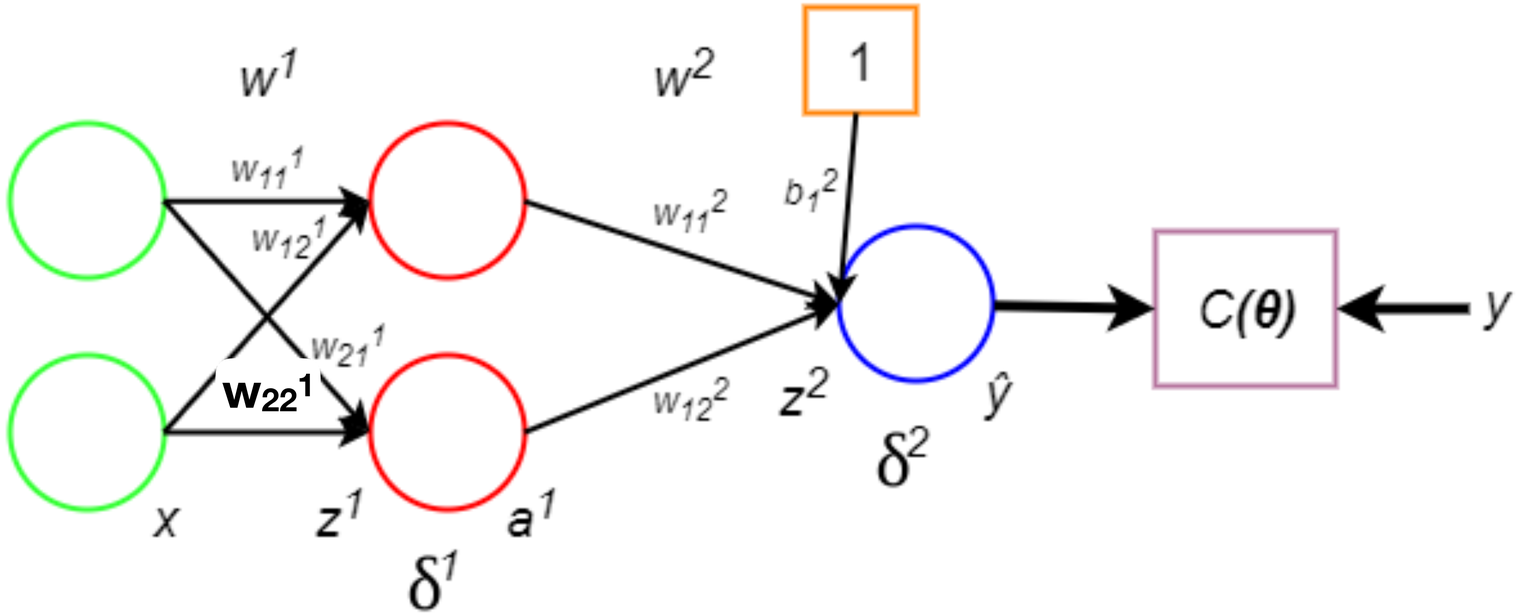
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

1. Compute error of output layer (l=2)

$$\delta^2 = \nabla_{\hat{y}} C \odot \nabla_{z^2} \hat{y}$$

- a) Derive gradient of cost C with respect to output $\hat{y}$

$$\nabla_{\hat{y}} C = \frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} (\hat{y} - y)^2 = 2(\hat{y} - y)$$

x: input vector

w^l: weight matrix of layer l

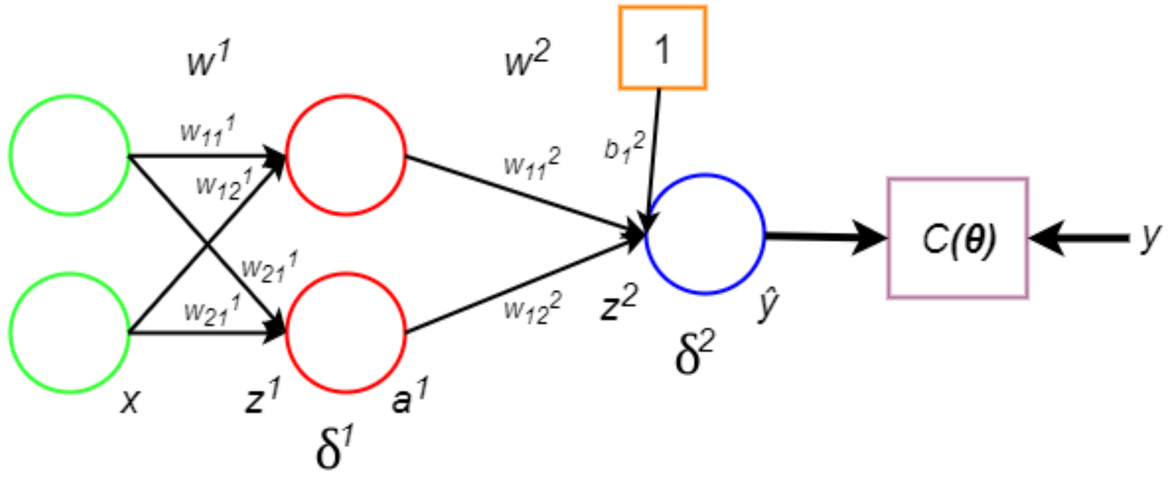
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

1. Compute error of output layer (*l*=2)

$$\delta^2 = \nabla_{\hat{y}} C \odot \nabla_{z^2} \hat{y}$$

- b) Derive gradient of output $\hat{y}$ with respect to weighted input (z^2) of output layer

$$\nabla_{z^2} \hat{y} = \frac{\partial \hat{y}}{\partial z^2} = \frac{\partial}{\partial z^2} \sigma(z^2)$$

$$= \sigma(z^2)\big(1 - \sigma(z^2)\big) = \hat{y}(1 - \hat{y})$$

x: input vector

w^l: weight matrix of layer l

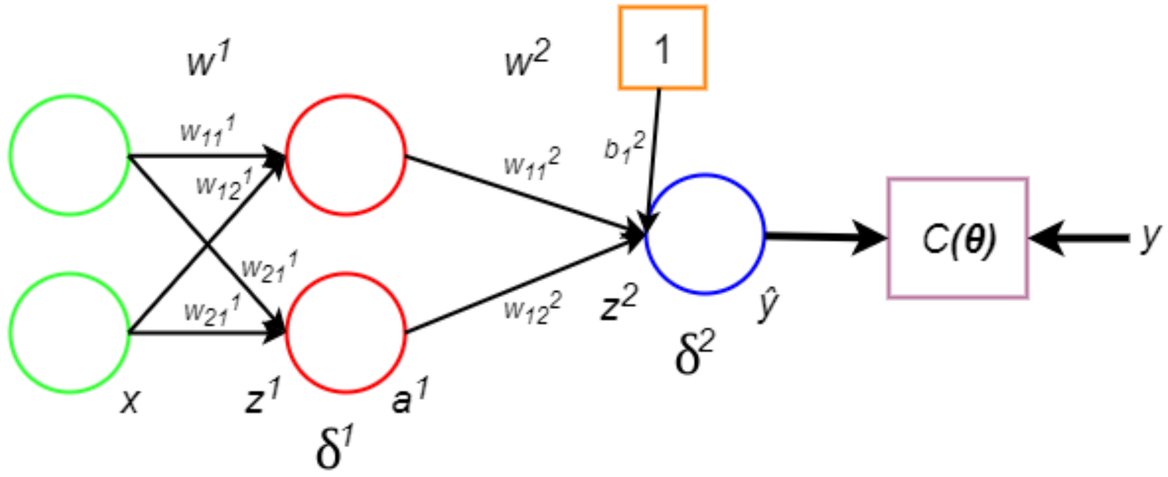
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

1. Compute error of output layer (l=2)

- We have:

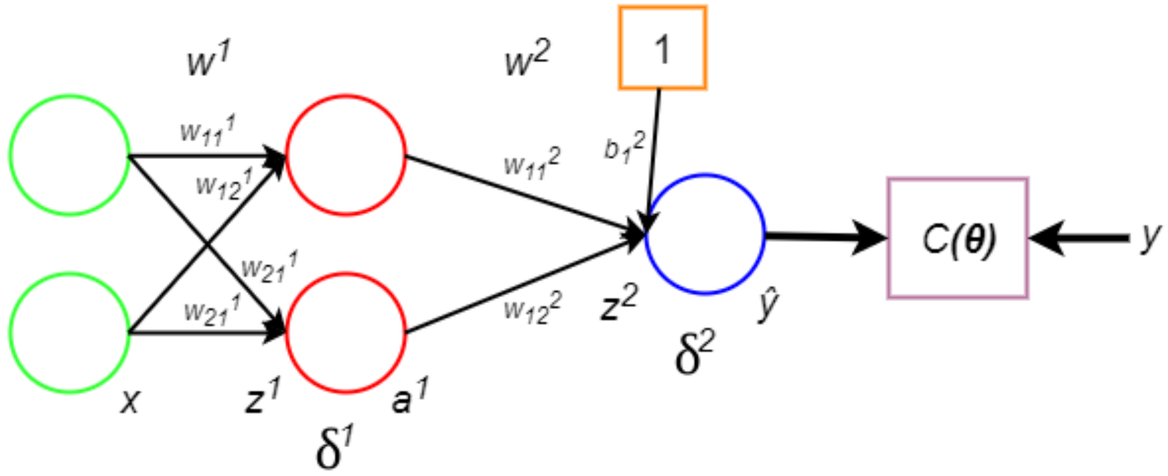
$$\hat{y} = 0.78 \qquad y = 1$$

- c) Calculate error of output layer

$$\nabla_{\hat{y}} C = 2(\hat{y} - y) = 2(0.78 - 1) = 0.44$$

$$\nabla_{z^2} \hat{y} = \hat{y}(1 - \hat{y}) = 0.78(1 - 0.78) = 0.20$$

$$\delta^2 = \nabla_{\hat{y}} C \odot \nabla_{z^2} \hat{y} = 0.44 \cdot 0.20 = 0.09$$



x: input vector

w^l: weight matrix of layer l

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l

Backpropagation

2. Update parameters of output layer (l=2)

- We have:

$$\delta^2 = 0.09 \qquad \lambda = 0.4 \qquad b^2 = 0.75$$

- a) Calculate gradient of bias and perform update

$$\nabla_{b_1{}^2} C = \delta_1{}^2 = 0.09$$

$$b_1{}^{2\prime} = b_1{}^2 - \lambda \nabla_{b_1{}^2} C$$
$$= 0.75 - 0.4 \cdot 0.09 = 0.71$$

x: input vector

w^l: weight matrix of layer l

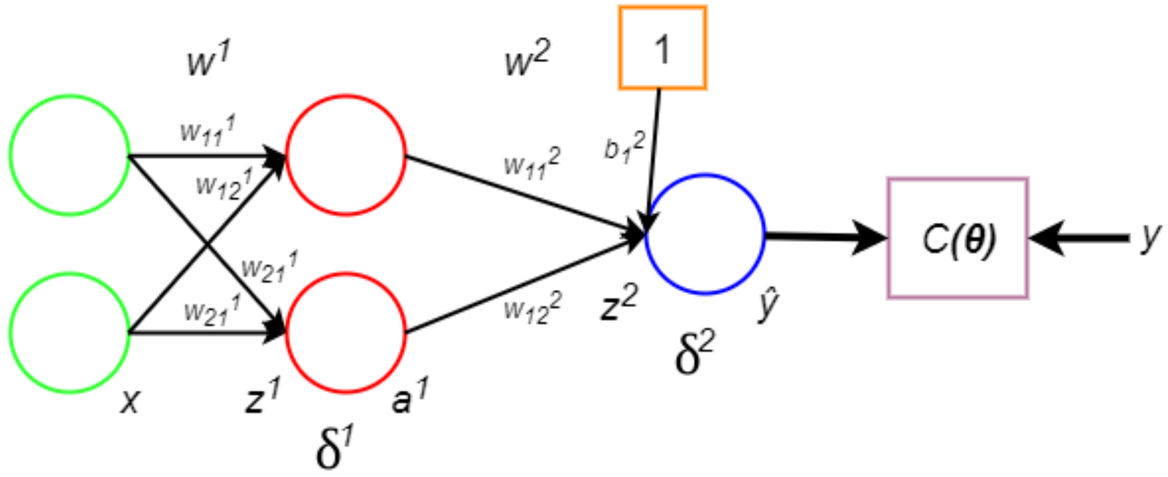
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l



Backpropagation

2. Update parameters of output layer (l=2)

- We have:

$$\delta^2 = 0.09 \quad a^1 = \begin{bmatrix} 0.62 \\ 0.5 \end{bmatrix} \quad \lambda = 0.4 \quad w^2 = [0.2 \quad 0.8]$$

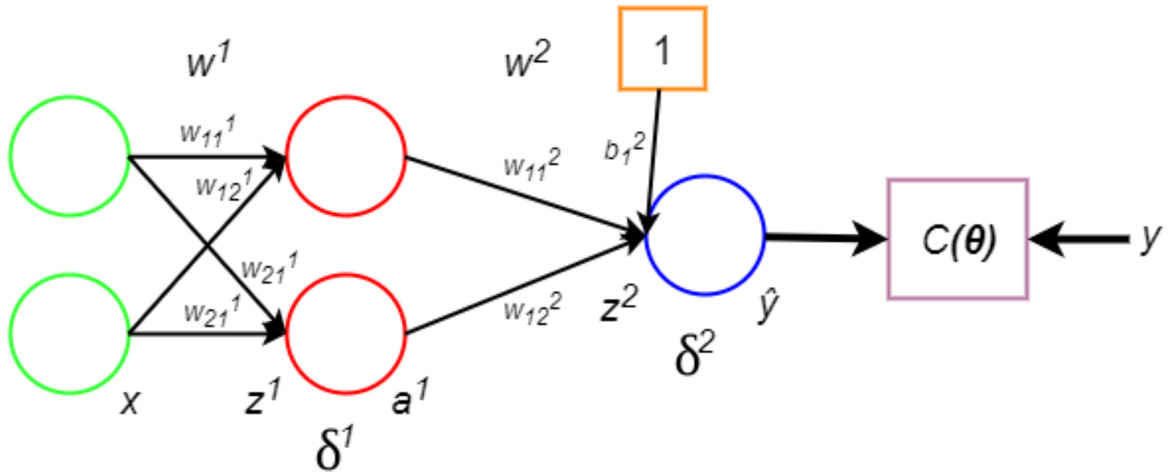
- b) Calculate gradients of weights and perform update

$$\nabla_{w_{11}{}^2} C = a_1{}^1 \delta_1{}^2 = 0.62 \cdot 0.09 = 0.06$$

$$\nabla_{w_{12}{}^2} C = a_2{}^1 \delta_1{}^2 = 0.5 \cdot 0.09 = 0.05$$

$$w_{11}{}^{2\prime} = w_{11}{}^2 - \lambda \nabla_{w_{11}{}^2} C = 0.2 - 0.4 \cdot 0.06 = 0.18$$

$$w_{12}{}^{2\prime} = w_{12}{}^2 - \lambda \nabla_{w_{12}{}^2} C = 0.8 - 0.4 \cdot 0.05 = 0.78$$



x: input vector

w^l: weight matrix of layer l

b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

δ^l: error of layer l

Backpropagation

3. Compute error of hidden layer (*l=1*)

$$\delta^1 = (w^2)^T \delta^2 \odot \nabla_{z^1} a^1$$

- a) Derive gradient of hidden layer output (***a¹***) with respect to weighted sum (***z¹***) of hidden layer

$$\nabla_{z^1} a^1 = \frac{\partial a^1}{\partial z^1} = \frac{\partial}{\partial z^1}\sigma(z^1) = \begin{bmatrix} \frac{\partial}{\partial z_1{}^1}\sigma(z_1{}^1) \\ \frac{\partial}{\partial z_2{}^1}\sigma(z_2{}^1) \end{bmatrix}$$

$$= \begin{bmatrix} \sigma(z_1{}^1)\left(1-\sigma(z_1{}^1)\right) \\ \sigma(z_2{}^1)\left(1-\sigma(z_2{}^1)\right) \end{bmatrix}$$

x: input vector

w^l: weight matrix of layer *l*

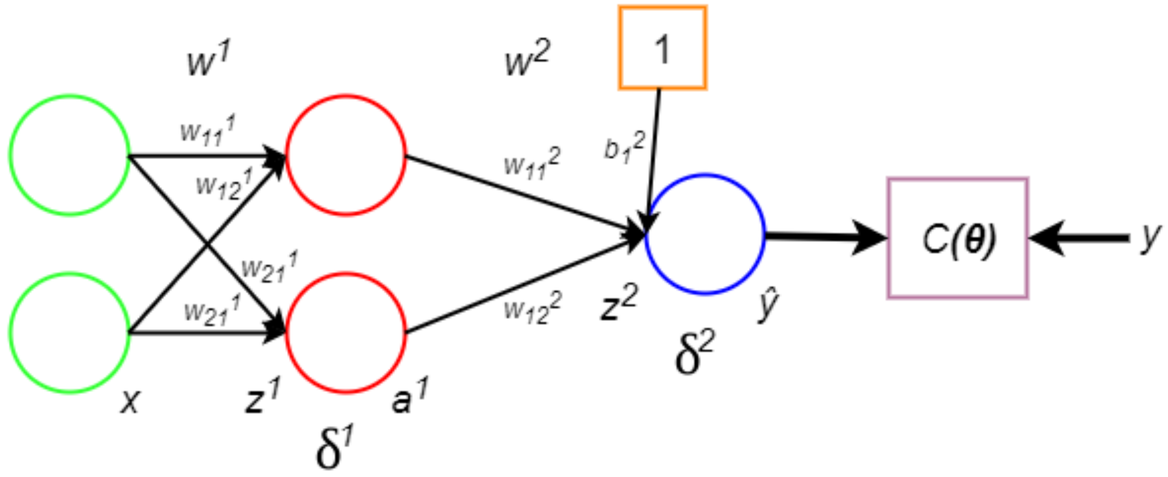
b^l: bias vector of layer *l*

z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*



Backpropagation

3. Compute error of hidden layer (*l=1*)

$$w^2 = [0.2 \quad 0.8] \quad \delta^2 = 0.09 \quad \sigma(z^1) = \begin{bmatrix} 0.62 \\ 0.5 \end{bmatrix}$$

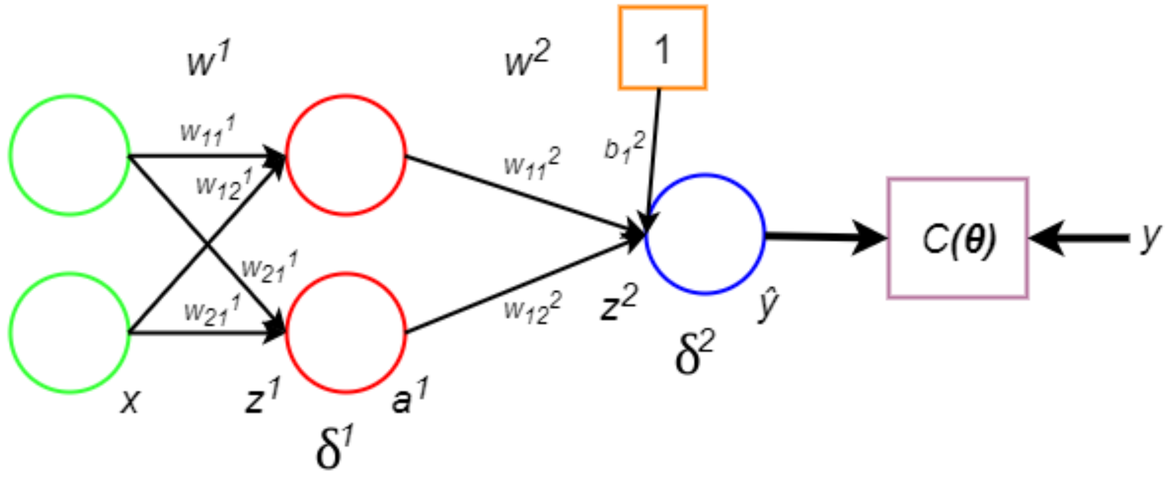
- b) Calculate gradients and error of hidden layer

$$(w^2)^T \delta^2 = \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix} 0.09 = \begin{bmatrix} 0.02 \\ 0.07 \end{bmatrix}$$

$$\nabla_{z^1} a^1 = \begin{bmatrix} \sigma(z_1{}^1)\left(1 - \sigma(z_1{}^1)\right) \\ \sigma(z_2{}^1)\left(1 - \sigma(z_2{}^1)\right) \end{bmatrix} = \begin{bmatrix} 0.62(1 - 0.62) \\ 0.5(1 - 0.5) \end{bmatrix} = \begin{bmatrix} 0.24 \\ 0.25 \end{bmatrix}$$

$$\delta^1 = (w^2)^T \delta^2 \odot \nabla_{z^1} a^1$$

$$= \begin{bmatrix} 0.02 \\ 0.07 \end{bmatrix} \odot \begin{bmatrix} 0.24 \\ 0.25 \end{bmatrix} = \begin{bmatrix} 0.02 \cdot 0.24 \\ 0.07 \cdot 0.25 \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.02 \end{bmatrix}$$



x: input vector

w^l: weight matrix of layer *l*

b^l: bias vector of layer *l*

z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*

Backpropagation

4. Update parameters of hidden layer (*l=1*)

- We have:

$$x = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \delta^1 = \begin{bmatrix} 0.01 \\ 0.02 \end{bmatrix}$$

- a) Calculate gradients of weight:

$$\nabla_{w_{11}{}^1} C = x_1 \delta_1{}^1 = 0 \cdot 0.01 = 0$$

$$\nabla_{w_{12}{}^1} C = x_2 \delta_1{}^1 = 1 \cdot 0.01 = 0.01$$

$$\nabla_{w_{21}{}^1} C = x_1 \delta_2{}^1 = 0 \cdot 0.02 = 0$$

$$\nabla_{w_{22}{}^1} C = x_2 \delta_2{}^1 = 1 \cdot 0.02 = 0.02$$

x: input vector

w^l: weight matrix of layer *l*

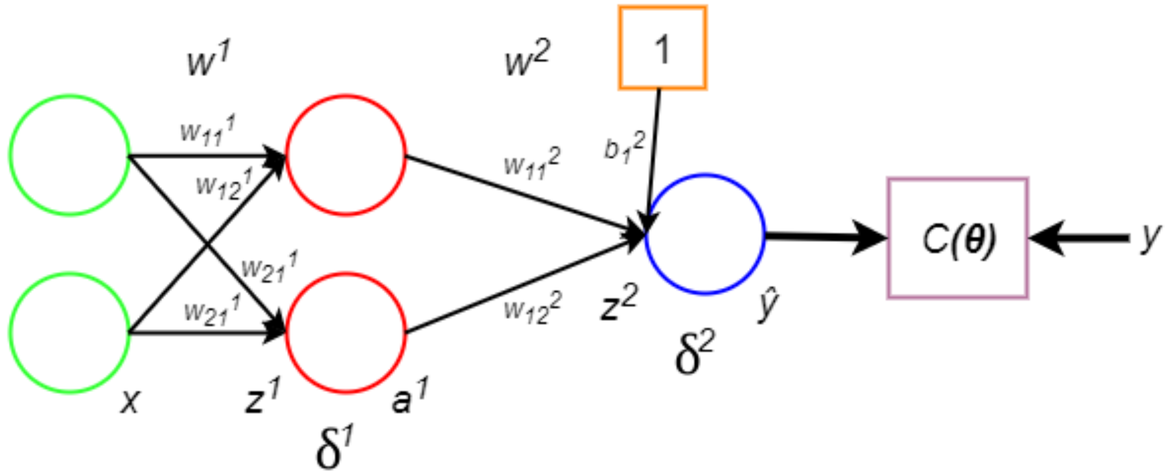
b^l: bias vector of layer *l*

z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*



Backpropagation

4. Update parameters of hidden layer (*l=1*)

- We have:

$$w^1 = \begin{bmatrix} 1 & 0.5 \\ 0.1 & 0 \end{bmatrix} \quad \lambda = 0.4 \quad \nabla_{w_{11}{}^1} C = 0 \quad \nabla_{w_{12}{}^1} C = 0.01 \quad \nabla_{w_{21}{}^1} C = 0 \quad \nabla_{w_{22}{}^1} C = 0.02$$

- b) Perform weight update

$$w_{11}{}^{1'} = w_{11}{}^1 - \lambda \nabla_{w_{11}{}^1} C = 1 - 0.4 \cdot 0 = 1$$

$$w_{12}{}^{1'} = w_{12}{}^1 - \lambda \nabla_{w_{12}{}^1} C = 0.5 - 0.4 \cdot 0.01 = 0.49$$

$$w_{21}{}^{1'} = w_{21}{}^1 - \lambda \nabla_{w_{21}{}^1} C = 0.1 - 0.4 \cdot 0 = 0.1$$

$$w_{22}{}^{1'} = w_{22}{}^1 - \lambda \nabla_{w_{22}{}^1} C = 0 - 0.4 \cdot 0.02 = -0.01$$

x: input vector

w^l: weight matrix of layer *l*

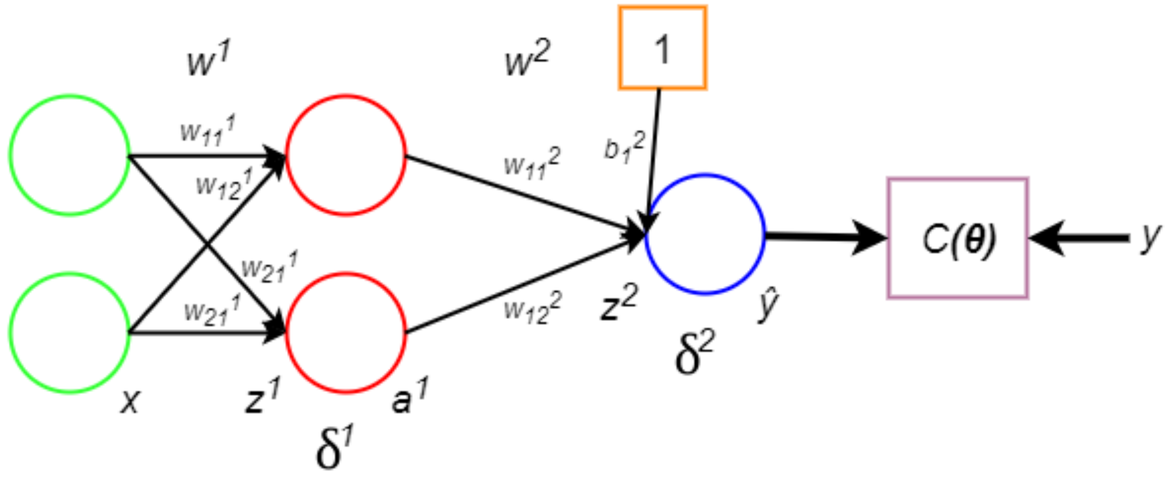
b^l: bias vector of layer *l*

z^l: weighted input of layer *l*

a^l: activation vector of layer *l*

$\hat{y}$: output vector

δ^l: error of layer *l*



Backpropagation

- Parameters before update

$$w^1 = \begin{bmatrix} {w_{11}}^1 & {w_{12}}^1 \\ {w_{21}}^1 & {w_{22}}^1 \end{bmatrix} = \begin{bmatrix} 1 & 0.5 \\ 0.1 & 0 \end{bmatrix}$$

$$w^2 = \begin{bmatrix} {w_{11}}^2 & {w_{12}}^2 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.8 \end{bmatrix}$$

$$b^2 = {b_1}^2 = 0.75$$

- Parameters after update

$$w^{1\prime} = \begin{bmatrix} {w_{11}}^{1\prime} & {w_{12}}^{1\prime} \\ {w_{21}}^{1\prime} & {w_{22}}^{1\prime} \end{bmatrix} = \begin{bmatrix} 1 & 0.49 \\ 0.1 & -0.01 \end{bmatrix}$$

$$w^{2\prime} = \begin{bmatrix} {w_{11}}^{2\prime} & {w_{12}}^{2\prime} \end{bmatrix} = \begin{bmatrix} 0.18 & 0.78 \end{bmatrix}$$

$$b^{2\prime} = {b_1}^{2\prime} = 0.71$$

x: input vector

w^l: weight matrix of layer l

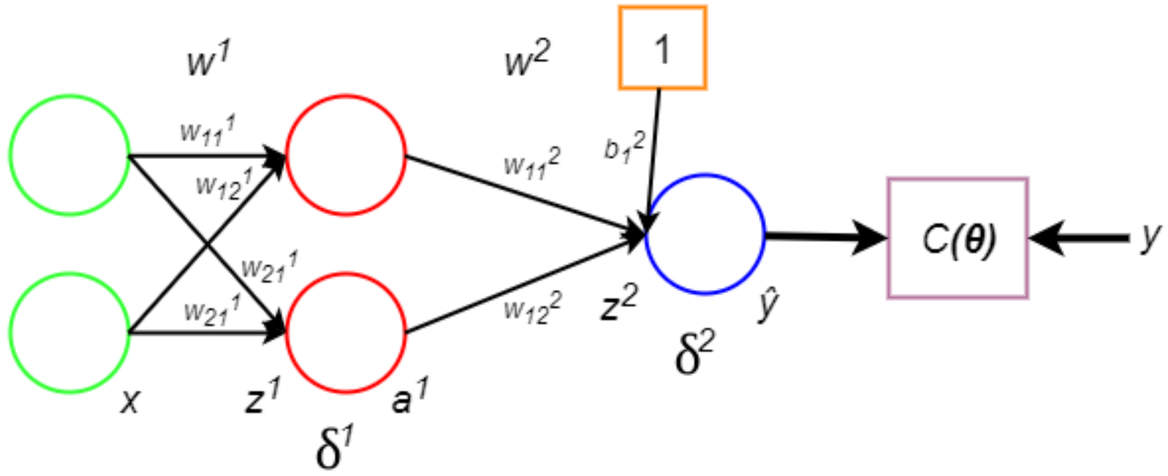
b^l: bias vector of layer l

z^l: weighted input of layer l

a^l: activation vector of layer l

$\hat{y}$: output vector

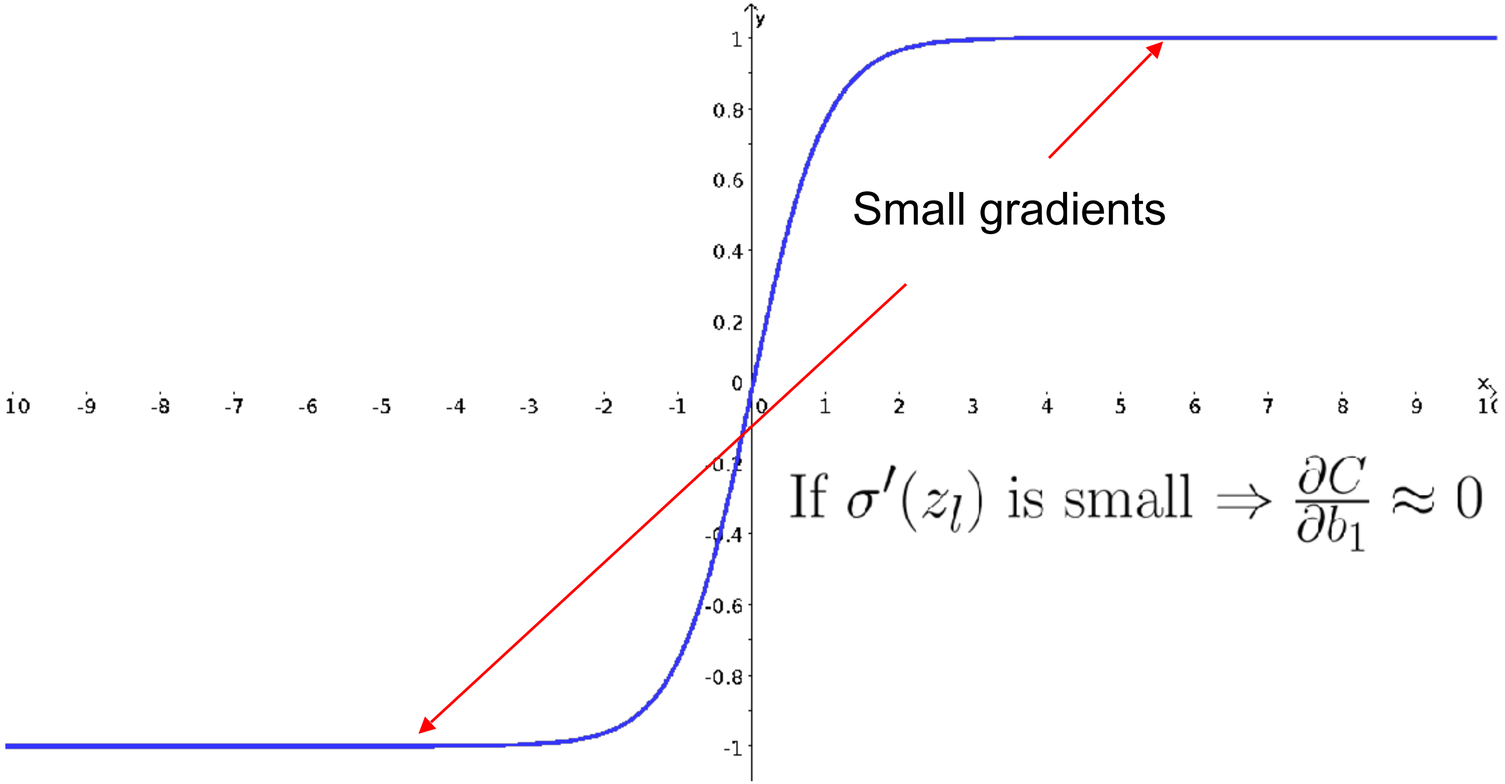
δ^l: error of layer l



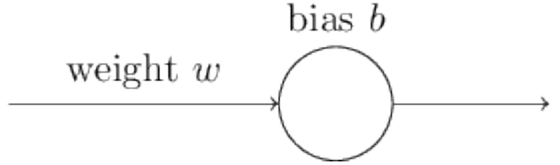
Vanishing gradients

Vanishing gradients

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



MSE:

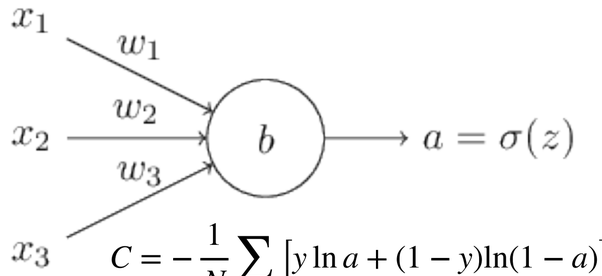


$$x = 1, y = 0 \qquad C = \frac{(y-a)^2}{2}$$

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z)$$

Cross-entropy:



$$C = -\frac{1}{N}\sum_n \left[y \ln a + (1-y)\ln(1-a)\right]$$

$$\frac{\partial C}{\partial w_j} = \frac{1}{N}\sum_n x_j(\sigma(z)-y) \qquad \frac{\partial C}{\partial b} = \frac{1}{N}\sum_n (\sigma(z)-y)$$

Log-likelihood:

$$C \equiv -\ln a_y^L \qquad a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

$$\frac{\partial C}{\partial w_{kj}^L} = a_j^{L-1}(a_k^L - y_k) \qquad \frac{\partial C}{\partial b_k^L} = a_k^L - y_k$$

Learning slowdown: AF(L) vs. CostF

- (Sigmoid output layer & quadratic cost)
- Linear output layer & quadratic cost
- Sigmoid output layer & cross-entropy cost
- Softmax output layer & log-likelihood cost