Neural Networks

30.11.2015

Lecturer: J. Matas

Authors: J. Matas, B. Flach, O. Drbohlav

Talk Outline

- Perceptron
- Combining neurons to a network
- Neural network, processing input to an output
- Learning
  - Cost function
  - Optimization of NN parameters
  - Back-propagation (a gradient descent method)
- Present and future

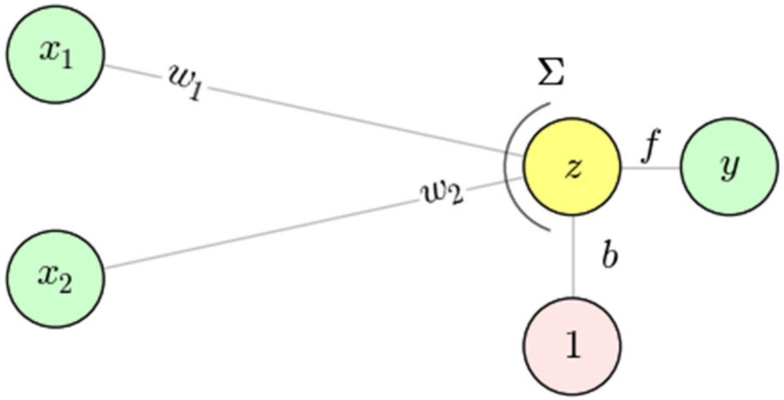
A Formal Neuron / Perceptron (1)

Binary-valued threshold neuron (McCulloch and Pitts ’49)

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

$$f(z) = \begin{cases} -1 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

- $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$ input
- $\boldsymbol{w} = (w_1, \dots, w_n) \in \mathbb{R}^n$ weights
- $b \in \mathbb{R}$ bias
- $y \in \{-1, 1\}$ output



Given the weights $\boldsymbol{w}$ and the bias b, the neuron produces an output $y \in \{-1, 1\}$ for any input $\boldsymbol{x}$.

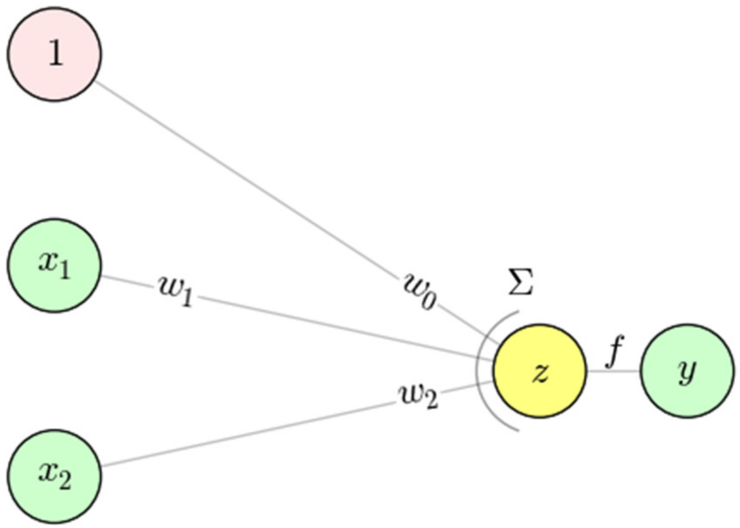
Note: This is a linear classifier, can be learned by the Perceptron Algorithm or SVM methods.

A Formal Neuron / Perceptron (2)

As usual, put the bias term b into the weights $\boldsymbol{w}$:

$$\begin{aligned} y &= f(\boldsymbol{w} \cdot \boldsymbol{x} + b) \\ &= f(\boldsymbol{w} \cdot \boldsymbol{x} + w_0 \cdot 1) \\ &= f(\boldsymbol{w}' \cdot \boldsymbol{x}') \end{aligned}$$

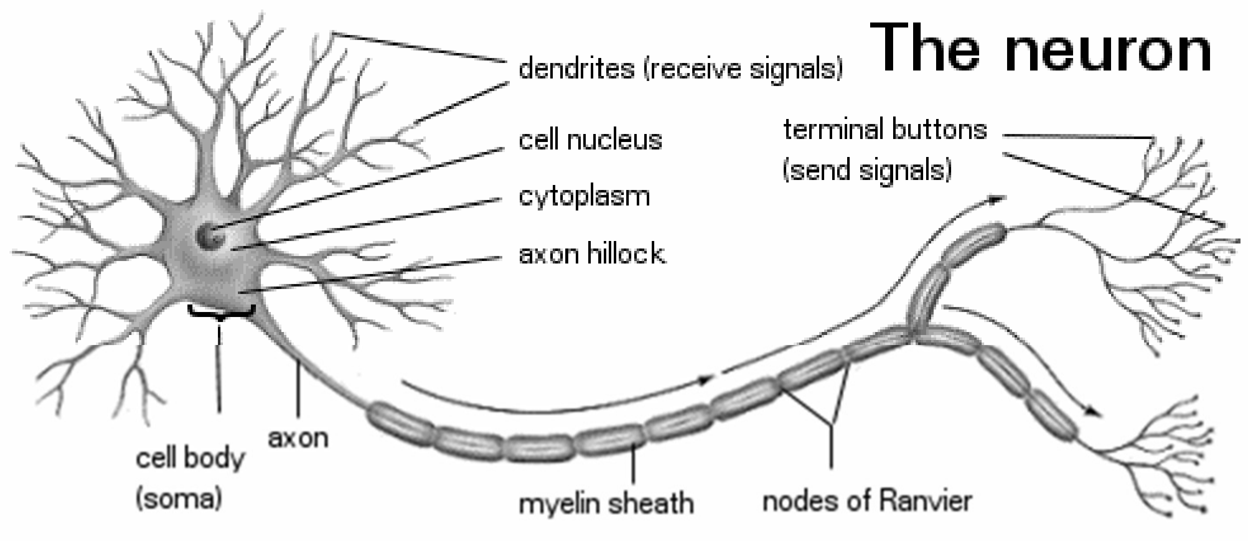
z … net activation
$y = f(z)$ … activation



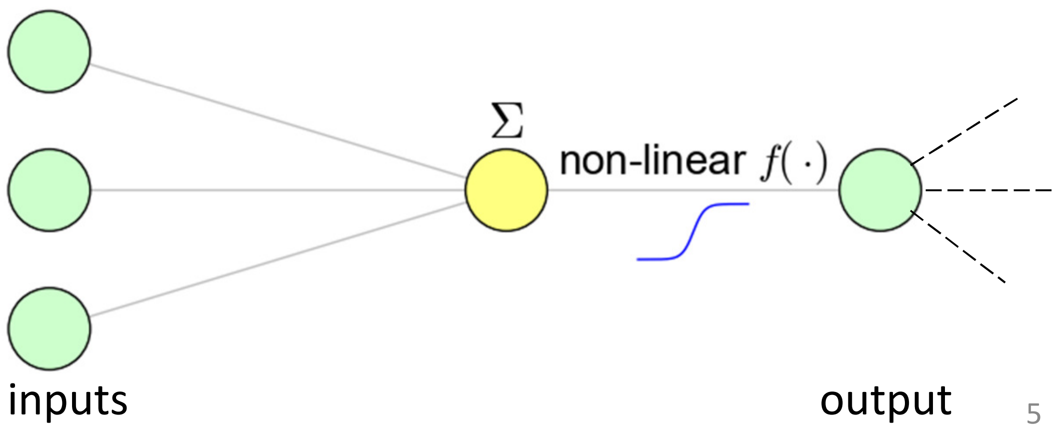
- $\boldsymbol{x}' = (1, x_1, \dots, x_n) \in \mathbb{R}^{n+1}$ input
- $\boldsymbol{w}' = (w_0, w_1, \dots, w_n) \in \mathbb{R}^{n+1}$ weights
- $f : \mathbb{R} \to \{-1, 1\}$ modified sign function
- $y \in \{-1, 1\}$ output

Neuron – Why That Name?

- A single neuron combines several inputs to an output
- Neurons are layered (outputs of neurons are used as inputs of other neurons)

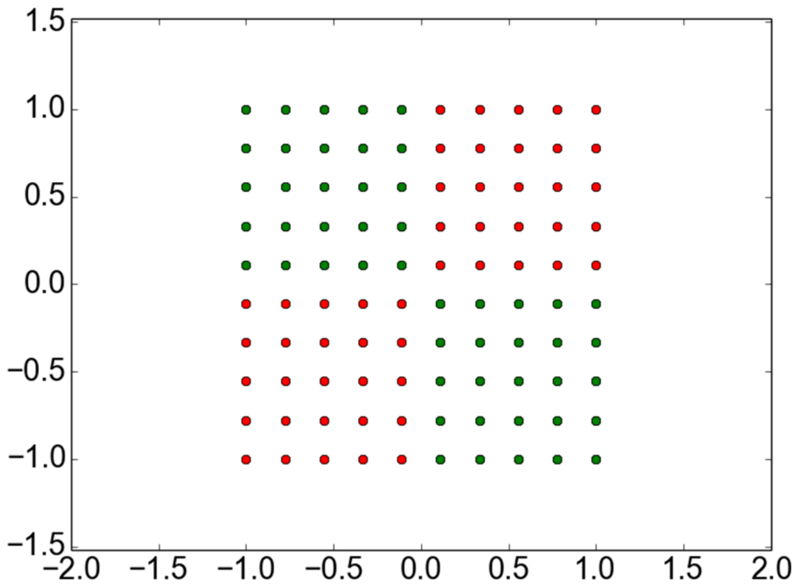


- A simple neuron model:



Historical perspective

- Perceptron (Rosenblatt, 1956) with its simple learning algorithm generated a lot of excitement
- Minsky and Papert (1969) showed that even a simple XOR cannot be learnt by a perceptron, this lead to skepticism

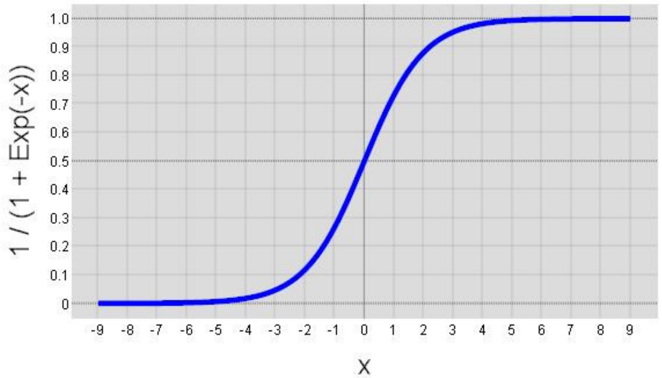


- The problem was solved by layering the perceptrons to a network (Multi-Layer Perceptron, MLP)

Graded Activation Function $f(\cdot)$

- Historically, the commonly used activation function $f(\cdot)$ is the sigmoid (cf. logistic regression)

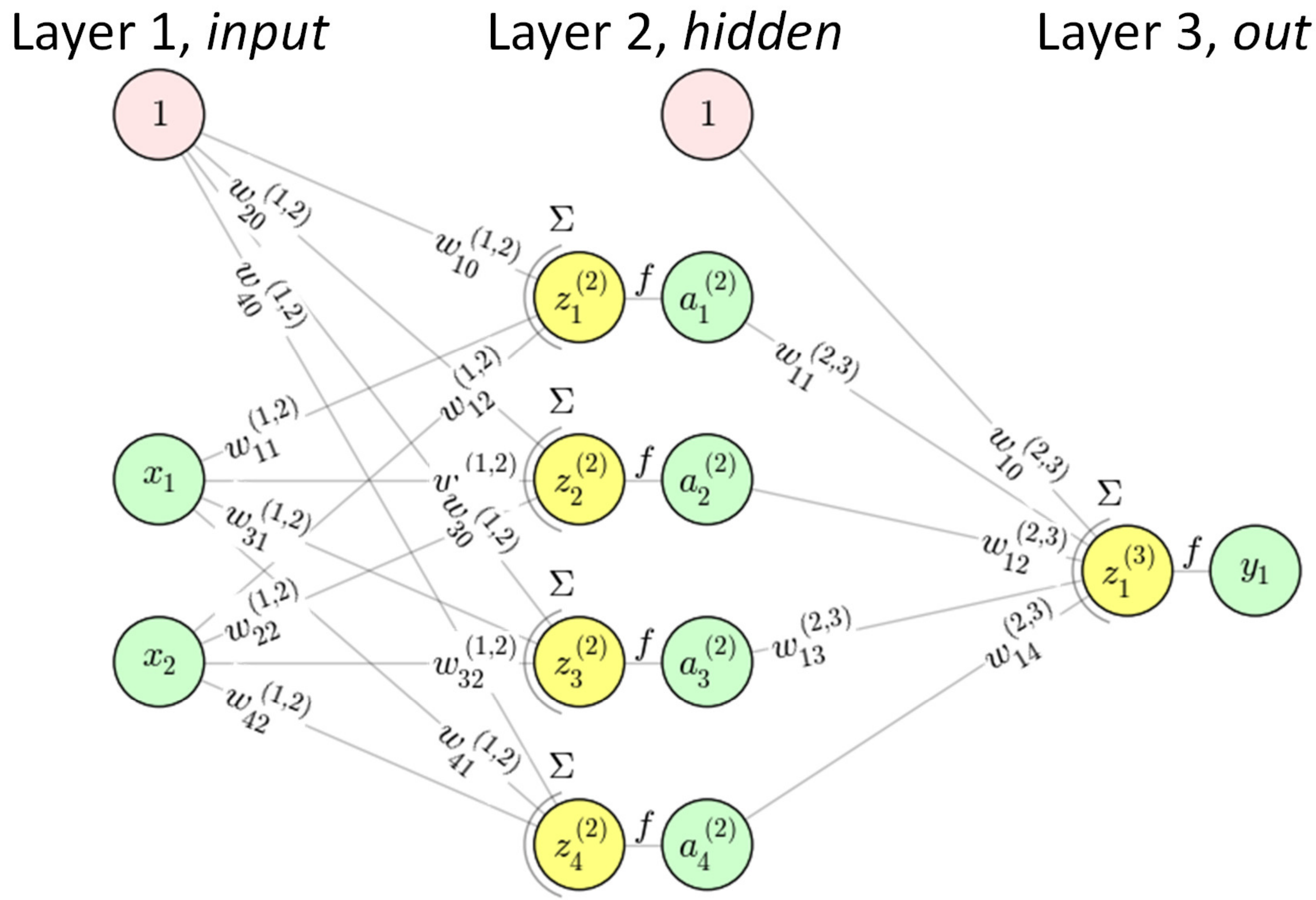
$$f(z) = \frac{1}{1 + e^{-z}}$$



- Its crucial properties are:
  – It is non-linear : if the activation function were linear, the multi-layer network could be rewritten (and would work the same as) a single-layer one
  – Differentiable : useful for fitting the coefficients of NN by gradient optimization

Three-Layer Neural Network (1/2)

- Input $\boldsymbol{x}$
- Output y_1



A 2-4-1 net

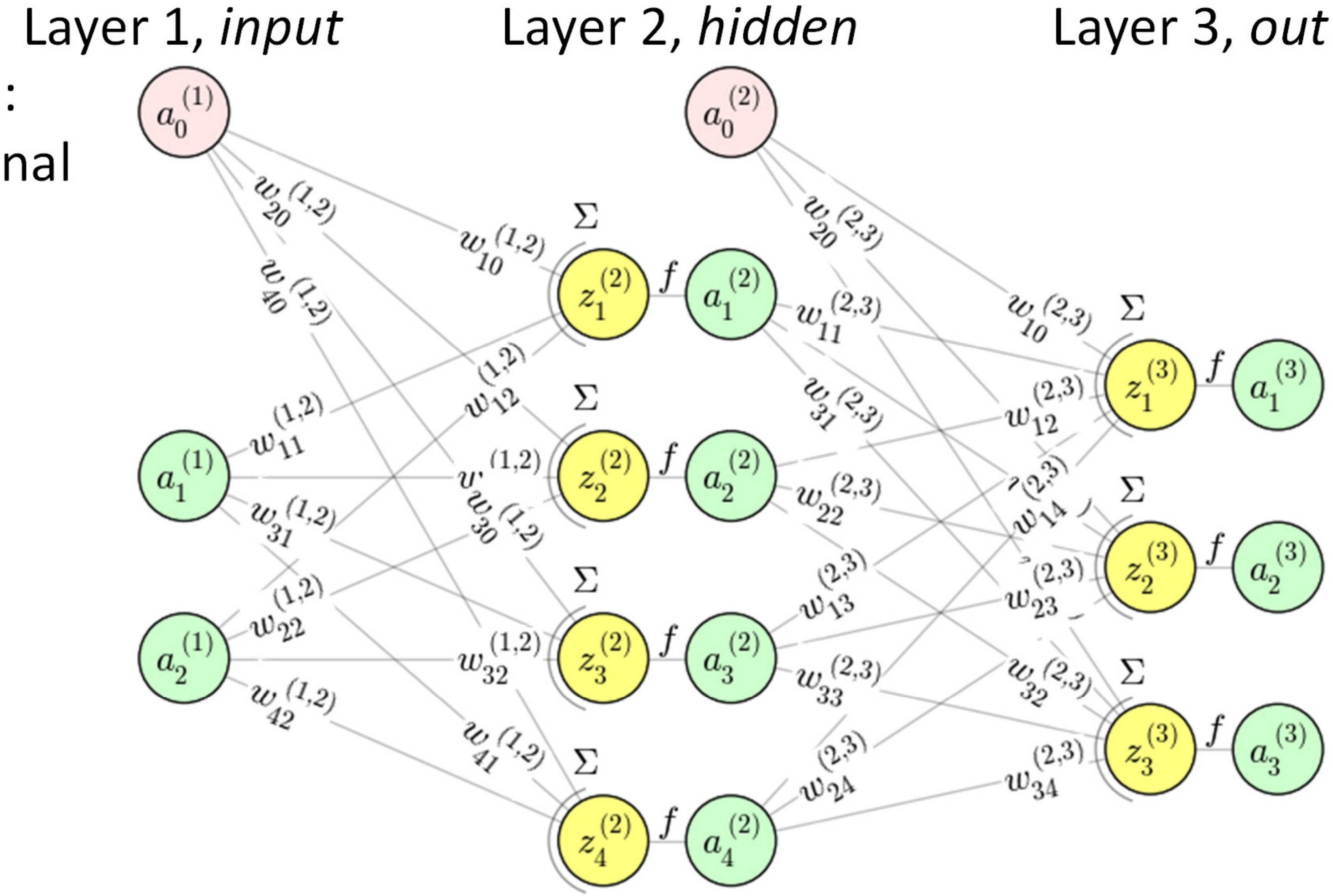
$z_4^{(2)} \longleftarrow$ Layer 2

$w_{42}^{(1,2)} \longleftarrow$ Weights between Layer 1 and 2

- Each neuron is a lin. combination of its inputs (incl. the bias term), followed by a non-linear transformation.

Three-Layer Neural Network (2/2)

- Generalization: multidimensional output $\boldsymbol{y}$
- Notation:
  $\boldsymbol{a}^{(1)} = [1, \boldsymbol{x}]$
  $\boldsymbol{a}^{(3)} = \boldsymbol{y}$



Three-Layer Neural Network (2/2)

- Generalization: multidimensional output $\boldsymbol{y}$
- Notation:
  $\boldsymbol{a}^{(1)} = [1, \boldsymbol{x}]$
  $\boldsymbol{a}^{(3)} = \boldsymbol{y}$
- All just works:

Given $\boldsymbol{a}^{(1)}$ (input)

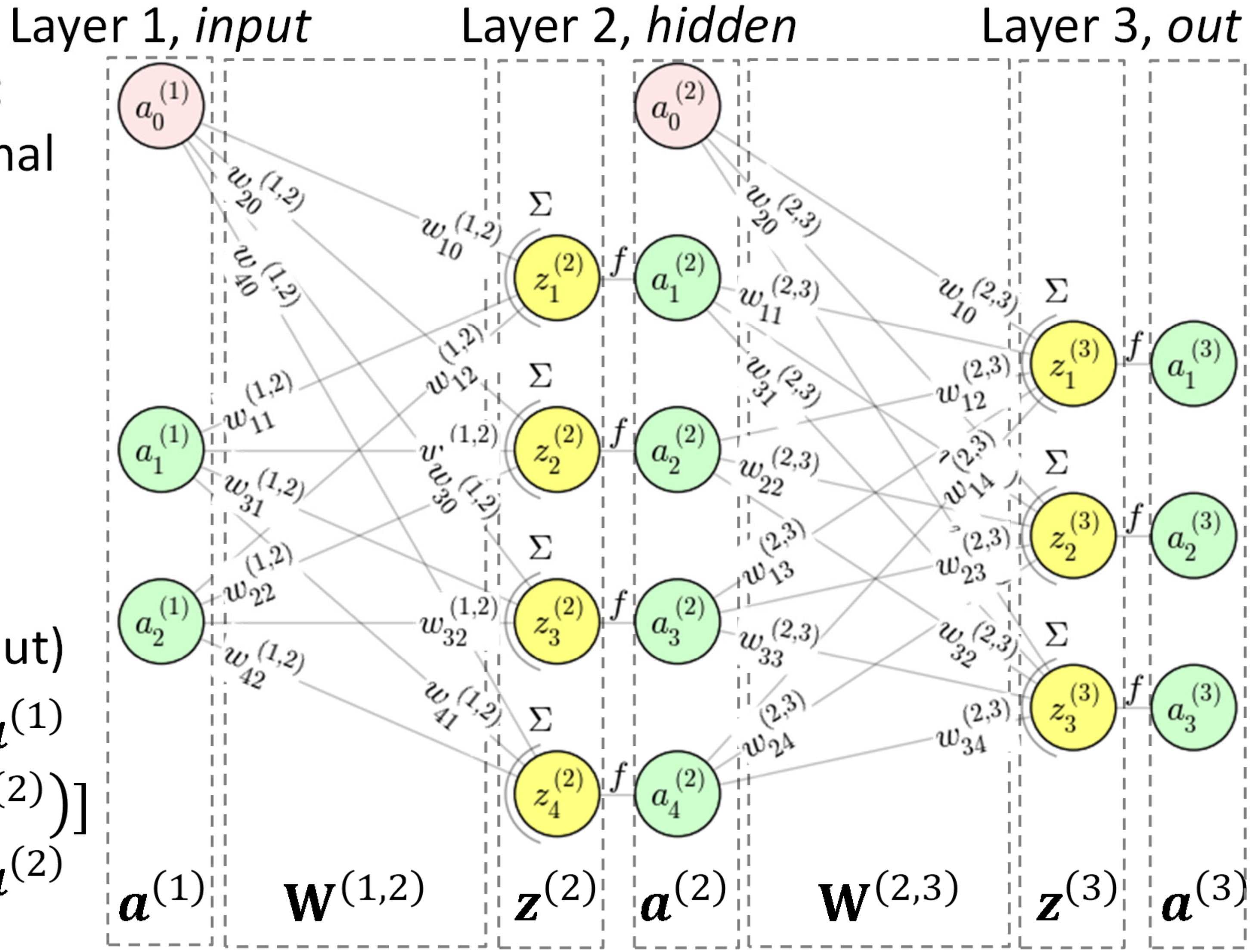
$\boldsymbol{z}^{(2)} = \mathbf{W}^{(1,2)}\boldsymbol{a}^{(1)}$

$\boldsymbol{a}^{(2)} = [1, f(\boldsymbol{z}^{(2)})]$

$\boldsymbol{z}^{(3)} = \mathbf{W}^{(2,3)}\boldsymbol{a}^{(2)}$

$\boldsymbol{a}^{(3)} = f(\boldsymbol{z}^{(3)})$

(= output)



Note: $f(\boldsymbol{z}) \stackrel{\text{def}}{=} \big(f(z_1), f(z_2), \dots f(z_n)\big)$
(f is applied element-wise)

K-Layer Neural Network

- Multilayer perceptron (MLP)
- Feed-forward computation
- Init:

  $$\boldsymbol{a}^{(1)} = [1, \boldsymbol{x}]$$

- Loop:

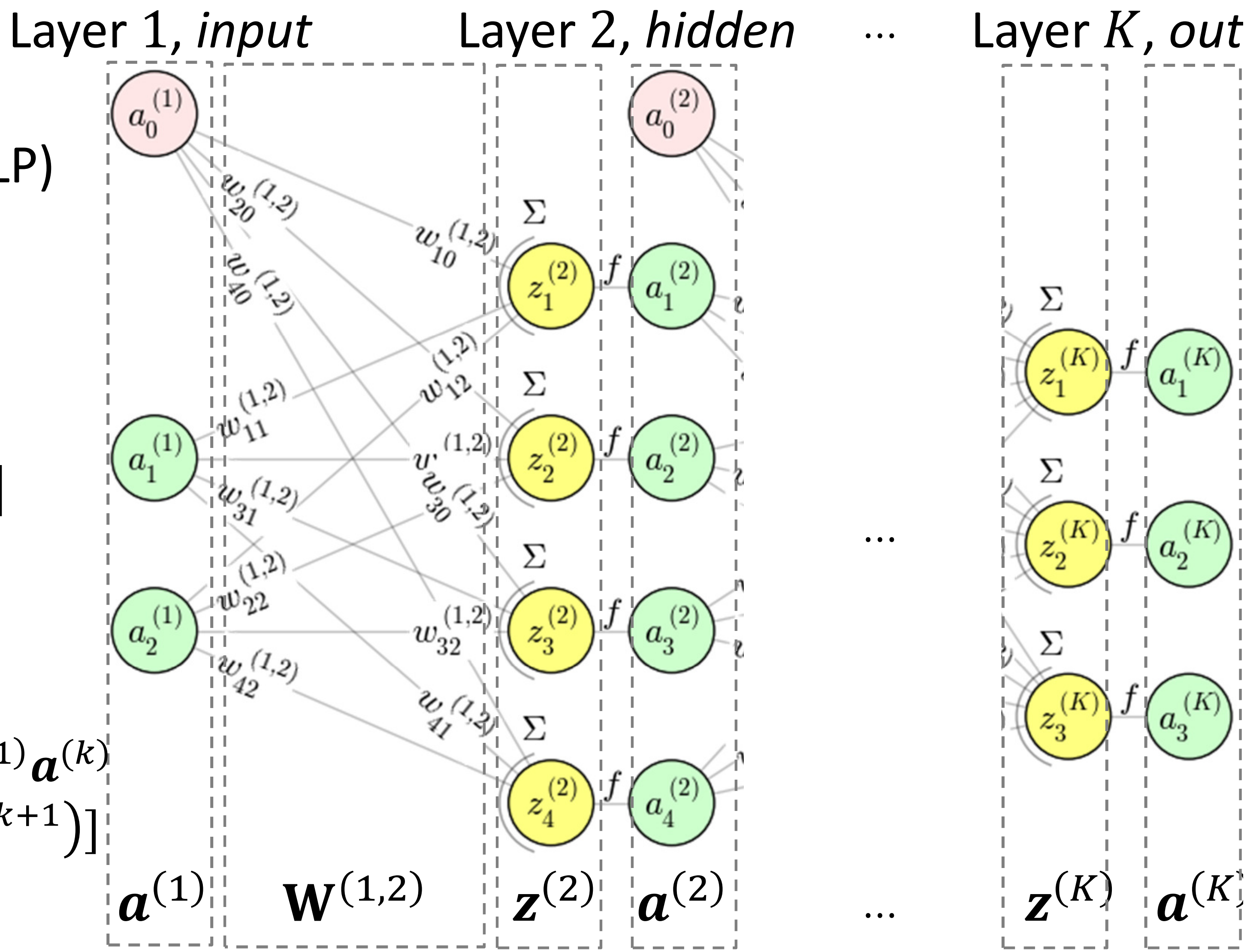
for $k = 1{:}\,\mathrm{K}-1$

$$\boldsymbol{z}^{(k+1)} = \mathbf{W}^{(k,k+1)}\boldsymbol{a}^{(k)}$$

$$\boldsymbol{a}^{(k+1)} = [1, f(\boldsymbol{z}^{k+1})]$$

- End:

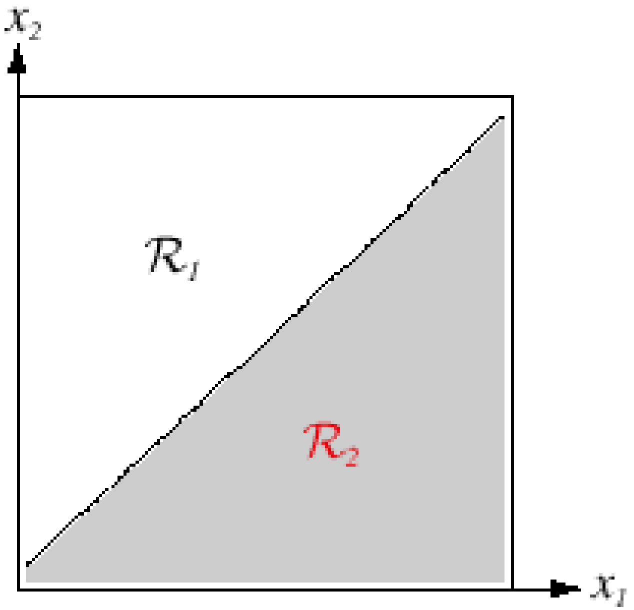
  $$\boldsymbol{y} = [\boldsymbol{a}^{(K)}]_{\emptyset}$$



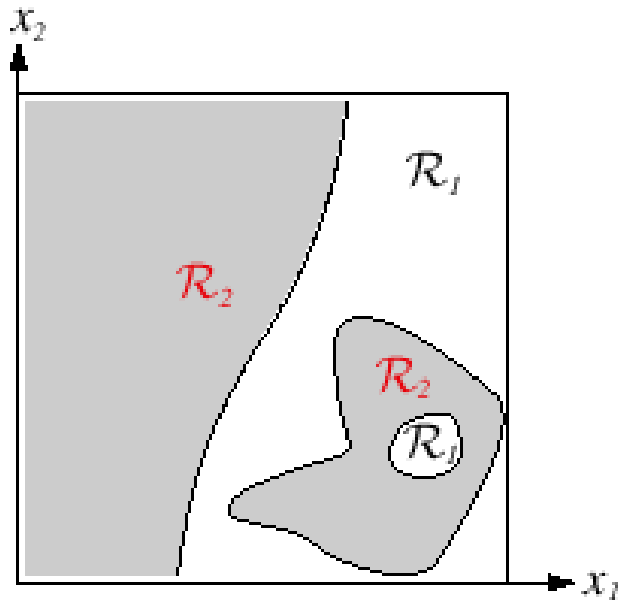
Operator $[\cdot]_{\emptyset}\colon \mathbb{R}^{D+1} \to \mathbb{R}^{D}$
$[(p_0, \dots, p_D)]_{\emptyset} = (p_1, \dots, p_D)$

Function approximation by a MLP

- Consider a simple case of K-layer NN with a single output neuron
- Such NN partitions space to two subsets $\mathcal{R}_1$ and $\mathcal{R}_2$



2-Layer NN: linear boundary between $\mathcal{R}_1$ and $\mathcal{R}_2$



K-Layer NN: can approximate increasingly more complex functions with increasing K

Images taken from Duda, Hart, Stork: Pattern Classification

Note: Remember the Adaboost example with weak *linear* classifiers? The strong classifier has been constructed as a linear combination of these. This is similar to what happens inside a 3-layer NN.

Regression, Classification, Learning (1)

- NNs can be employed for function approximation. Approximation from sample (training) points is the *regression* problem. Classification can be approached as a special case of regression.
- So far, the weight matrices $\mathbf{W}$ have been assumed to be already known.
- Learning the weight matrices is formulated as an optimization problem. Given the training set $\mathcal{T} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), i = 1..N\}$, we optimize

$$J_{\text{total}}(\{\mathbf{W}\}) = \sum_{i=1}^{N} J(\boldsymbol{y}_i, \boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x}_i)),$$

where $\boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x}_i)$ is the output of NN for $\boldsymbol{x}_i$, and $J(\cdot,\cdot)$ is the cost function.

Regression, Classification, Learning (2)

- For a 2-class classification, the last layer has one neuron, and the output $\boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x}_i)$ is thus 1-dimensional.
- For K-class classification, a common choice is to encode the class by an M-dimensional vector:

$$y = (0, 0, \dots, 1, \dots, 0)^T \, ,$$

1 at k-th coordinate if $\boldsymbol{x}$ belongs to k-th class.

Each class $k \in \{1, 2, .., K\}$ has an associated weight vector w_k.

The conditional probability for the k-th function is computed using the **softmax** function:

$$p(k|x) = \frac{e^{w_k x}}{e^{w_1 x} + e^{w_2 x} + \dots + e^{w_K x}} \, . \tag{40}$$

Regression, Classification, Learning (3)

- A frequent choice for $J(\cdot,\cdot)$ is the quadratic loss:

$$J(\boldsymbol{y}, \boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x})) = \frac{1}{2} \|\boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x}) - \boldsymbol{y}\|^2$$

- Other possibility: cross entropy, etc.

Graded Activation Function $f(\cdot)$

$$J_{\text{total}}(\{\mathbf{W}\}) = \sum_{i=1}^{N} J(\boldsymbol{y}_i, \boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x}_i))$$

- Ready to optimize J_{total} ?
  - $J(\cdot,\cdot)$ is a quadratic loss (no problem)
  - $\boldsymbol{y}^{(K)}$ is a composition of two types of functions:
    - Linear combination (no problem)
    - Activation function $f(\cdot)$ – must be differentiable (modified signum function is not)

Learning: Minimize J

$$\{\mathbf{W}'\} = \underset{\{\mathbf{W}\}}{\operatorname{argmin}}\, J_{\text{total}}(\{\mathbf{W}\}) = \underset{\{\mathbf{W}\}}{\operatorname{argmin}} \sum_{i=1}^{N} J(\boldsymbol{y}_i, \boldsymbol{y}(\{\mathbf{W}\}, \boldsymbol{x}_i))$$

Apply gradient descent.

Compute gradient / partial derivatives w.r.t. all weights:

$$\frac{\partial J_{\text{total}}}{\partial w_{pq}^{(k,k+1)}} = \sum_{i=0}^{N} \frac{\partial J(x_i)}{\partial w_{pq}^{(k,k+1)}}$$

Gradient of J (1/4)

Example for NN with number of layers $K = 3$, output dimensionality D, and quadratic loss function:

$$\frac{\partial J(x)}{\partial w_{pq}^{(k,k+1)}} = \sum_{j=1}^{D} \left[y(W,x) - y\right]_j \frac{\partial \left[y(W,x)\right]_j}{\partial w_{pq}^{(k,k+1)}} =$$

$$= \sum_{j=1}^{D} \underbrace{\left[y(W,x) - y\right]_j}_{D_j} \frac{\partial a_j^{(3)}(x)}{\partial w_{pq}^{(k,k+1)}}$$

Output discrepancy

Dep. of j-th output neuron on that weight

Note: $[\cdot]_j$ is j-th component.

Gradient of J (2/4)

So, we have that:
$$\frac{\partial J(x)}{\partial w_{pq}^{(k,k+1)}} = \sum_{j=1}^{D} D_j \frac{\partial a_j^{(3)}(x)}{\partial w_{pq}^{(k,k+1)}}$$

Let us have a look at the gradient patterns, based on some examples (note: f' is the derivative of f, $*$ is element-wise multiplication):

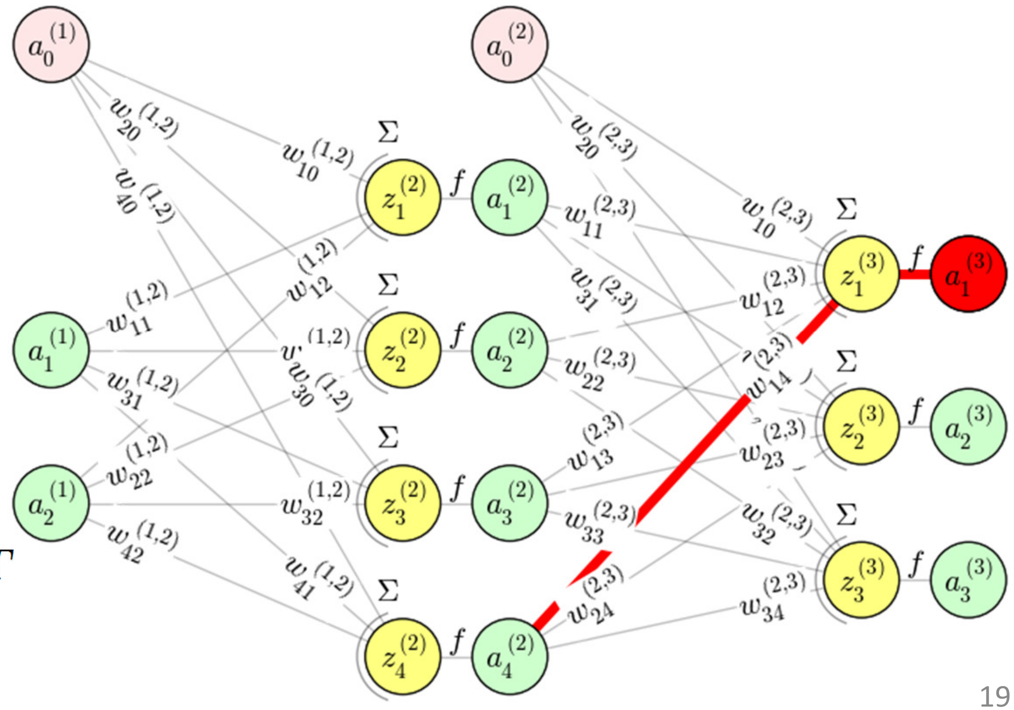
$$\frac{\partial a_j^{(3)}}{\partial w_{14}^{(2,3)}} = \frac{\partial a_j^{(3)}}{\partial z_j^{(3)}} \frac{\partial z_j^{(3)}}{\partial w_{14}^{(2,3)}} = \begin{cases} f'(z_1^{(3)}) a_4^{(2)} & \text{if } j = 1 \\ 0 & \text{otherwise} \end{cases}$$

Thus, for $\mathbf{W}^{(2,3)}$:

$$\frac{\partial J(x)}{\partial w_{pq}^{(2,3)}} = D_p f'(z_p^{(3)}) a_q^{(2)}$$

In vector notation:

$$\frac{\partial J(x)}{\partial W^{(2,3)}} = \left[D * f'(z^{(3)}) \right] a^{(2)T}$$



Gradient of J (3/4)

So, we have that:

$$\frac{\partial J(x)}{\partial w_{pq}^{(k,k+1)}} = \sum_{j=1}^{D} D_j \frac{\partial a_j^{(3)}(x)}{\partial w_{pq}^{(k,k+1)}}$$

$$\frac{\partial a_j^{(3)}}{\partial w_{30}^{(1,2)}} = \frac{\partial a_j^{(3)}}{\partial z_j^{(3)}} \frac{\partial z_j^{(3)}}{\partial a_3^{(2)}} \frac{\partial a_3^{(2)}}{\partial z_3^{(2)}} \frac{\partial z_3^{(2)}}{\partial w_{30}^{(1,2)}} = f'(z_j^{(3)}) w_{j3}^{(2,3)} f'(z_3^{(2)}) a_0^{(1)}$$

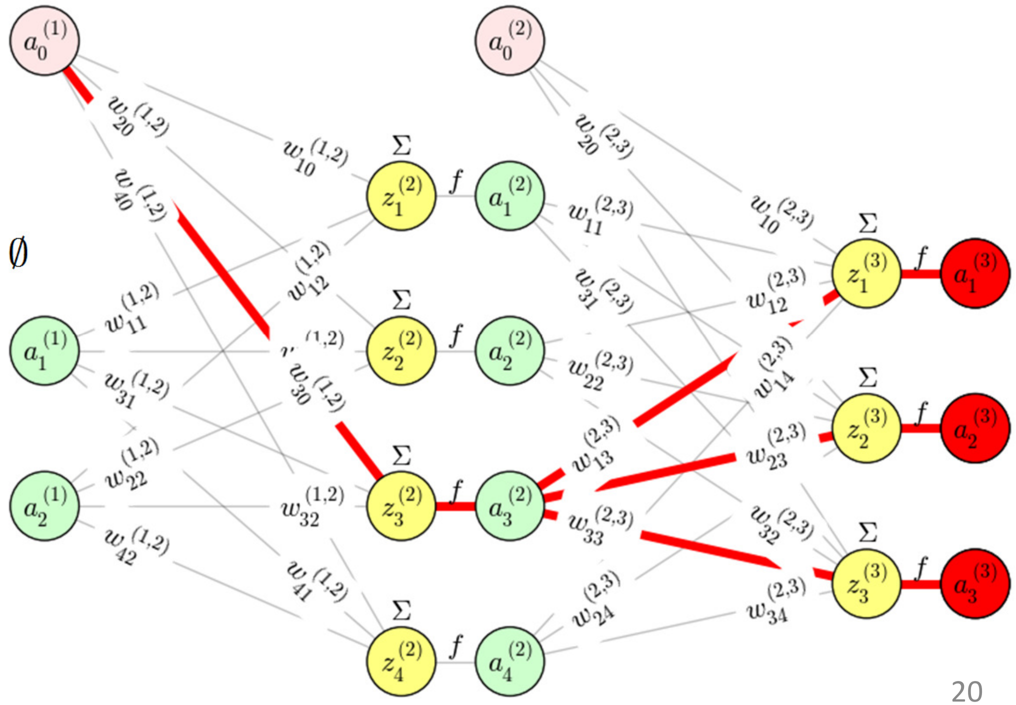
$$\frac{\partial J(x)}{\partial w_{pq}^{(1,2)}} = \sum_{j=1}^{D} D_j f'(z_j^{(3)}) w_{jp}^{(2,3)} f'(z_p^{(2)}) a_q^{(1)}$$

In vector notation:

$$\frac{\partial J(x)}{\partial W^{(1,2)}} = \left[W^{(2,3)T} \left[D * f'(z^{(3)}) \right] \right]_{\emptyset} * f'(z^{(2)}) a^{(1)T}$$

Cf.

$$\frac{\partial J(x)}{\partial W^{(2,3)}} = \left[D * f'(z^{(3)}) \right] a^{(2)T}$$



Gradient of J (4/4)

Define:

$$\boldsymbol{\Delta}^{(k,k+1)} = \frac{\partial J}{\partial \mathbf{W}^{(k,k+1)}}$$

Compute:

output from feed-forward

desired output

$$\boldsymbol{\delta}^{(9)} = \left(\left[\boldsymbol{a}^{(9)}(\boldsymbol{x})\right]_{\emptyset} - \boldsymbol{y}\right) * f'\left(\boldsymbol{z}^{(9)}\right)$$

$$\boldsymbol{\delta}^{(8)} = \left[\mathbf{W}^{(8,9)T}\boldsymbol{\delta}^{(9)}\right]_{\emptyset} * f'\left(\boldsymbol{z}^{(8)}\right)$$

$$\boldsymbol{\delta}^{(7)} = \left[\mathbf{W}^{(7,8)T}\boldsymbol{\delta}^{(8)}\right]_{\emptyset} * f'\left(\boldsymbol{z}^{(7)}\right)$$

$$\dots$$

$$\boldsymbol{\delta}^{(2)} = \left[\mathbf{W}^{(2,3)T}\boldsymbol{\delta}^{(3)}\right]_{\emptyset} * f'\left(\boldsymbol{z}^{(2)}\right)$$

Compute gradient of J:

$$\boldsymbol{\Delta}^{(8,9)} = \boldsymbol{\delta}^{(9)}\boldsymbol{a}^{(8)T}$$

$$\boldsymbol{\Delta}^{(7,8)} = \boldsymbol{\delta}^{(8)}\boldsymbol{a}^{(7)T}$$

$$\dots$$

$$\boldsymbol{\Delta}^{(1,2)} = \boldsymbol{\delta}^{(2)}\boldsymbol{a}^{(1)T}$$

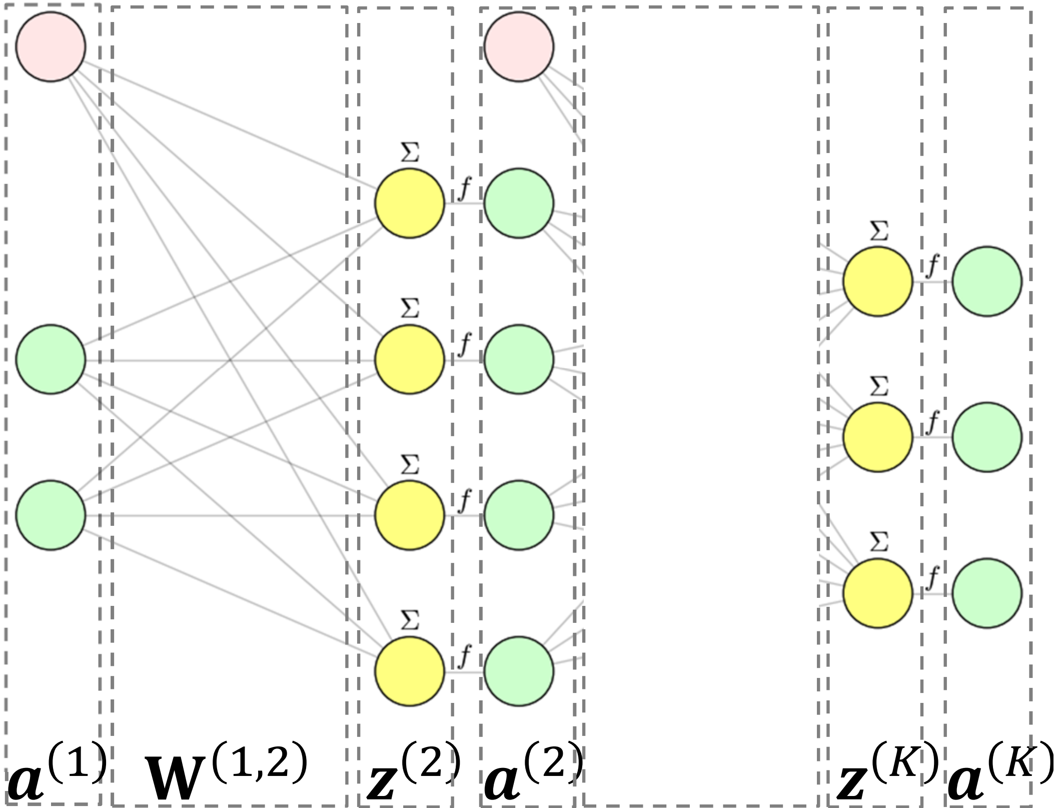
Notes:

$K = 9$ used as an example

T = transposition

$*$ = elementwise multiplication

$[\cdot]_{\emptyset}$: remove the first vector component



Back-propagation algorithm (1/2)

Given $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}$

Do forward propagation.

compute predicted output for x

Compute the gradient.

Update the weights:

$\mathbf{W}^{(k,k+1)} \leftarrow \mathbf{W}^{(k,k+1)} + \beta \boldsymbol{\Delta}^{(k,k+1)}$

β… learning rate

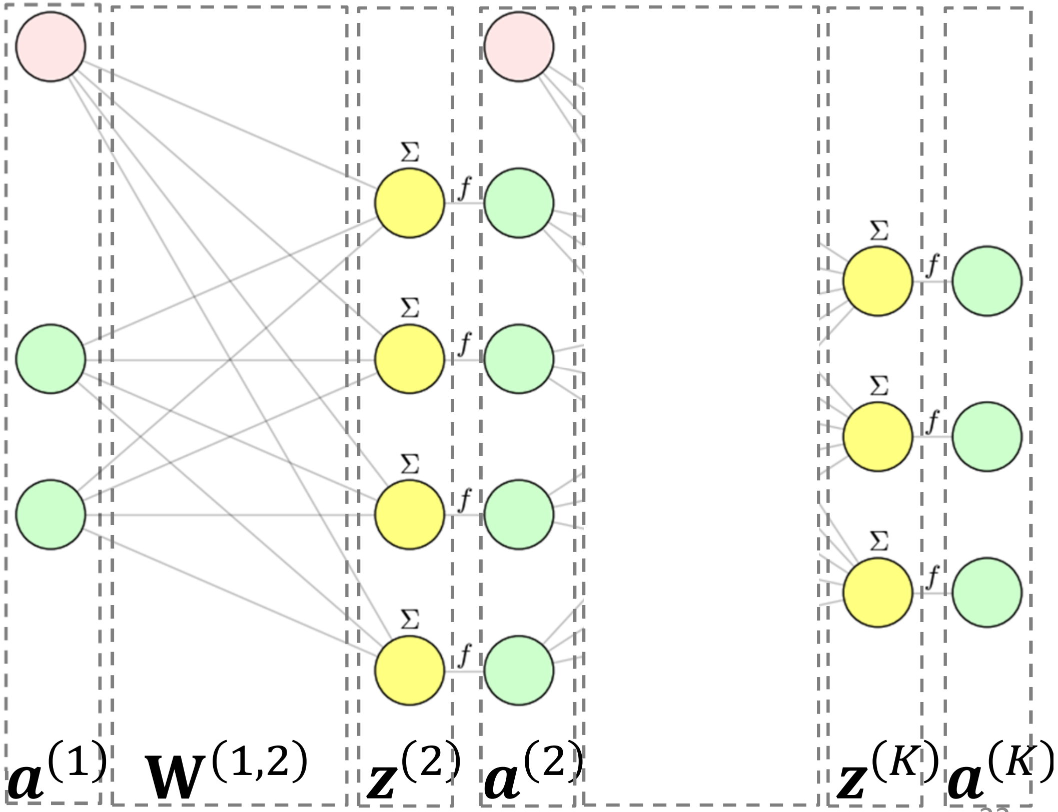
Repeat until convergence.

Notes:

$K = 9$ used as an example

T = transposition

$*$ = elementwise multiplication



Back-propagation algorithm (2/2)

- Update computation was shown for 1 training sample only for the sake of clarity
- This variant of weight updates can be used (loop over the training set like in the Perceptron algorithm)
- Back-propagation is a gradient-based minimization method.
- Variants: construct the weight update using the entire batch of training data , or use mini-batches as a compromise between exact gradient computation and computational expense
- The step size (learning rate) could be found by line search algorithm as in standard gradient-based optimization
- Many variants for the cost function – logistic regression-type, regularization term, etc. This will lead to different update rules.

NN by back-propagation - properties

Advantages:

- Handles well the problem with multiple classes
- Can do both classification and regression
- After normalization, output can be treated as aposteriori probability

Disadvantages:

- No guarantee to reach the global minimum

Notes:

- Ways to choose network structure?
- Note that we assumed the activation functions to be identical throughout the NN. This is not a requirement though.

Deep NNs

- Deep learning – “hot” topic, unsupervised discovery of features
- Renaissance of NNs
- What is different from the past? Massive amounts of data, regularization, sparsity enforcement, drop-out
- Used in computer vision, speech recognition, general classification problems

Deep NNs

- A common alternative to the sigmoid: RELU (rectified linear unit)