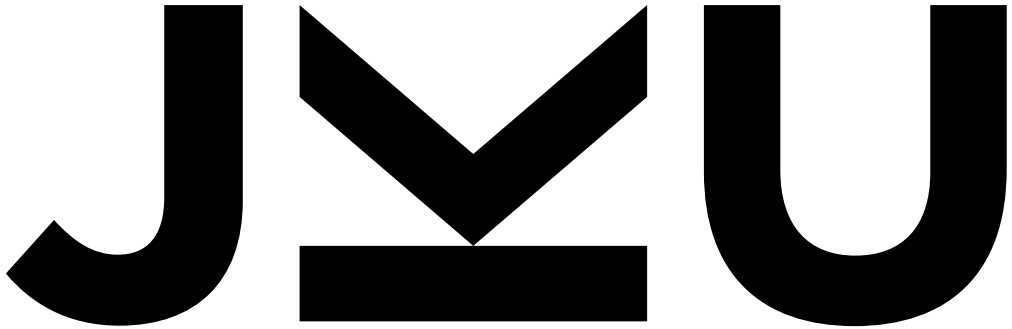
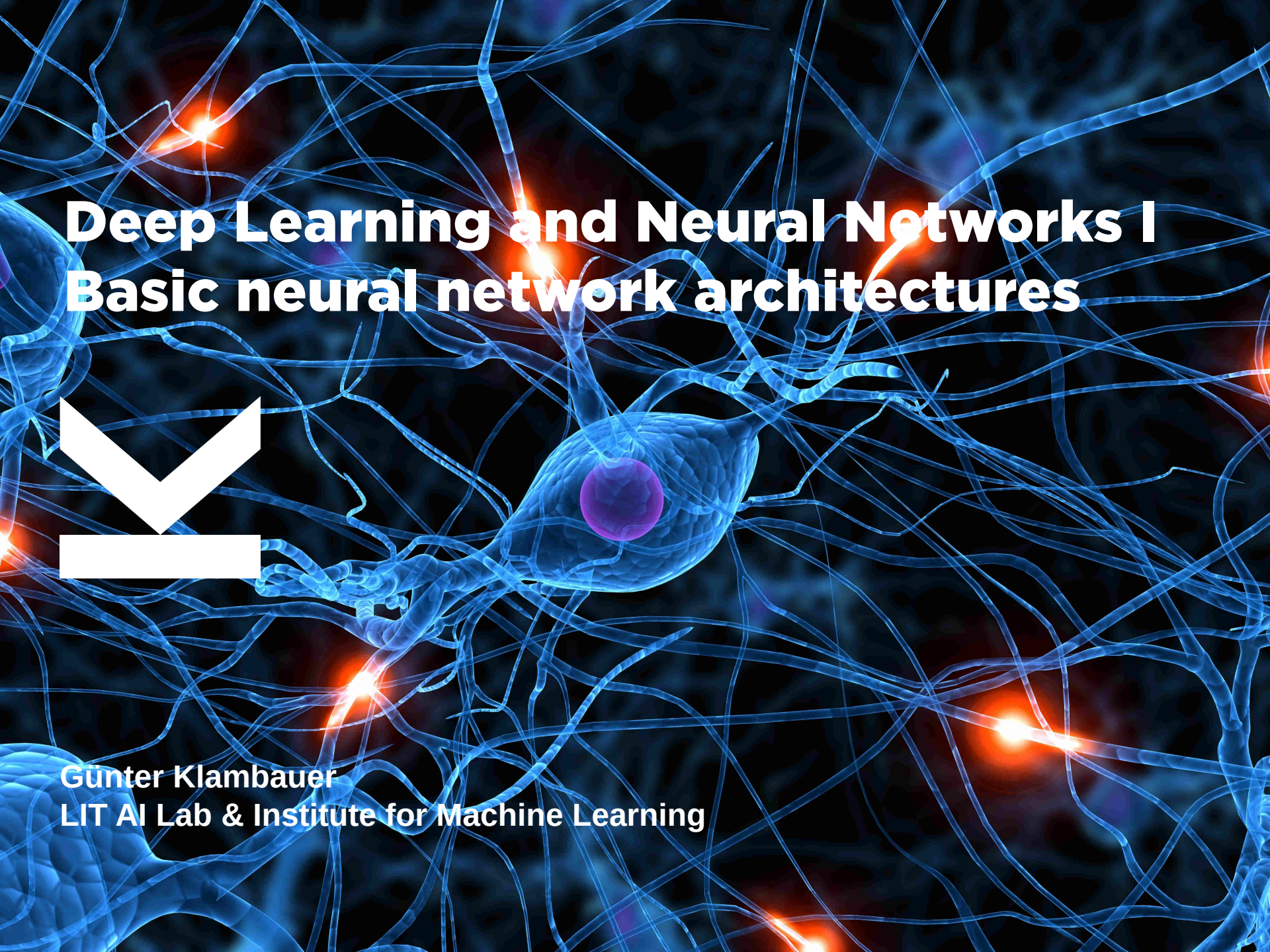


JOHANNES KEPLER
UNIVERSITY LINZ

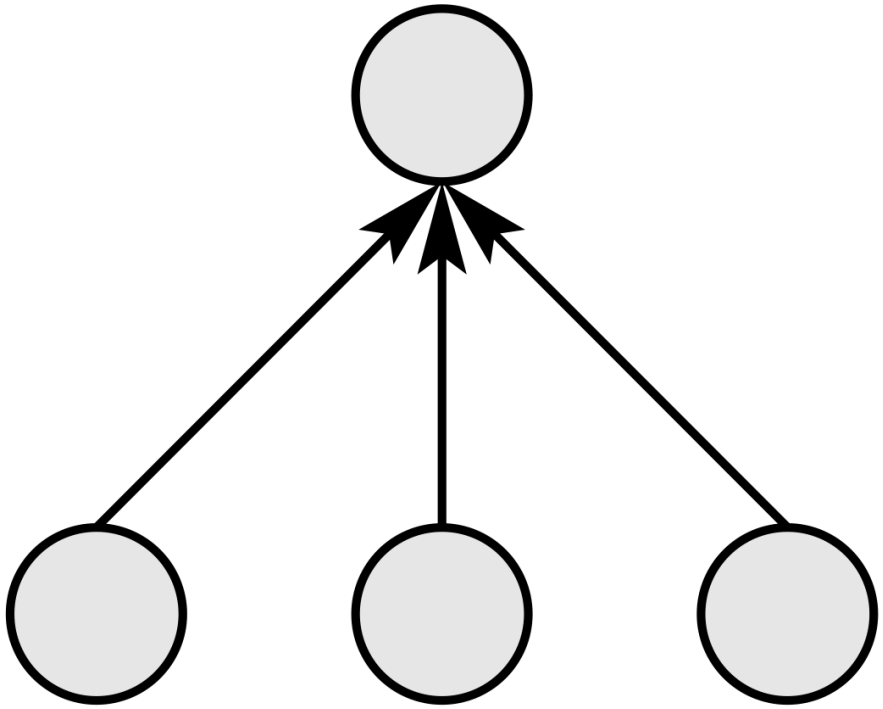


Deep Learning and Neural Networks I
Basic neural network architectures

Günter Klambauer
LIT AI Lab & Institute for Machine Learning

This material, no matter whether in printed or electronic form, may be used for personal and non-commercial educational use only. Any reproduction of this material, no matter whether as a whole or in parts, no matter whether in printed or in electronic form, requires explicit prior acceptance of the authors.

Artificial neural networks: linear neuron



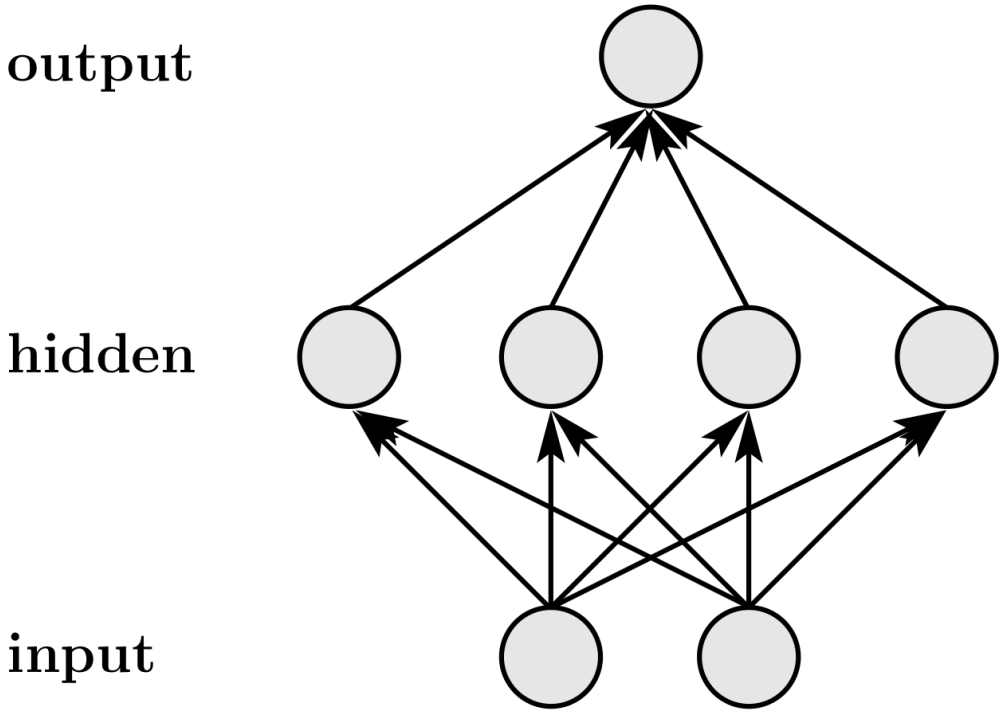
$$a = \boldsymbol{w}^T \boldsymbol{x}$$

$\boldsymbol{w}$

$\boldsymbol{x}$

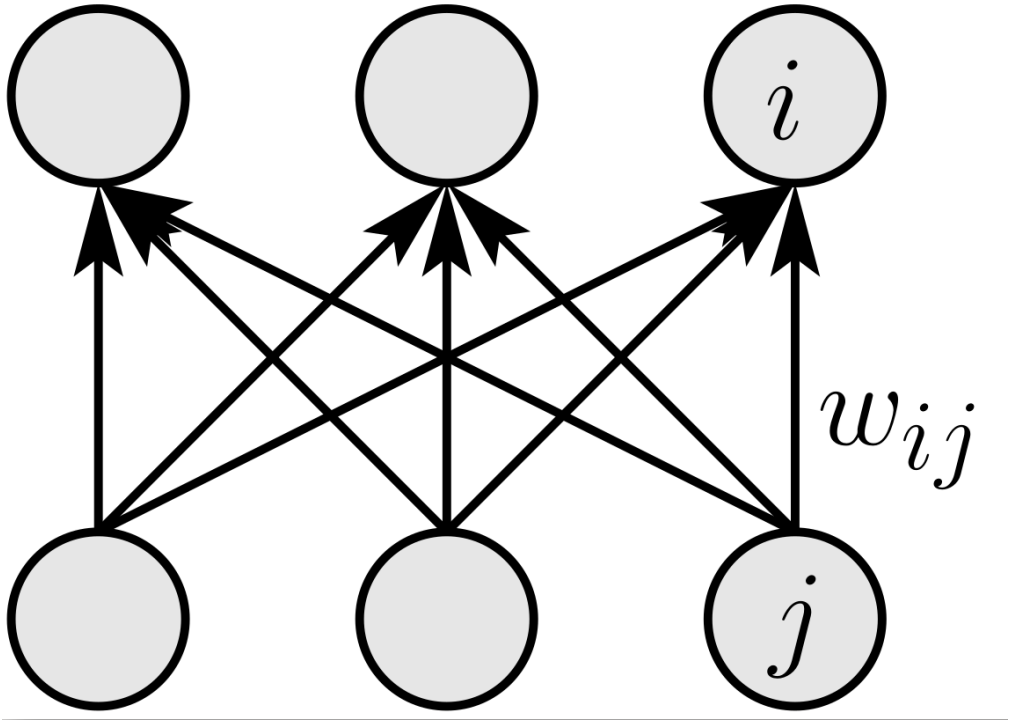
- Inputs directly connected to output
- Input: $\boldsymbol{x}$ adaptive weights: $\boldsymbol{w}$ output (activation): a

Artifical neural networks: 3-layer network



- Artificial neural networks: a 3-layered net with an input, hidden, and output layer

Neural network with adaptive weights



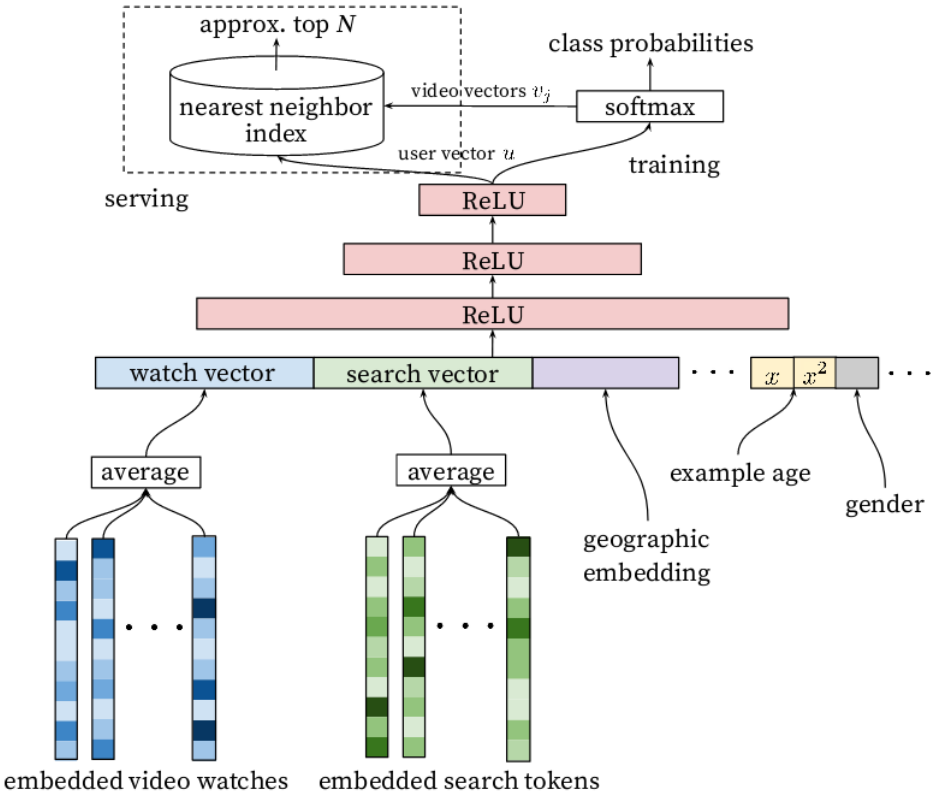
Artifical neural networks: units and weights. The weight w_{ij} gives the weight, connection strength, or synaptic weight from unit j to unit i

Overview

- 4. Basic neural network architectures
  - 4.5 Single-layer networks cannot solve XOR
  - 4.6 Multi-layer perceptron
    - 4.6.1. Model and forward pass of an MLP
    - 4.6.2. Gradient descent and the backpropagation algorithm
    - 4.6.3. Training MLPs
  - 4.7. (Deep) feed-forward neural networks
    - Backpropagation in DNNs

Why MLPs? Deep MLP!?

- MLPs are still heavily used!
- Many neurons, many layers



Deep Neural Networks for YouTube Recommendations

Paul Covington, Jay Adams, Emre Sargin
Google
Mountain View, CA
{pcovington, jka, msargin}@google.com

Covington, P., Adams, J., & Sargin, E. (2016, September). Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems (pp. 191-198). ACM.

4.5 Single-layer networks cannot solve XOR

- Assume we have

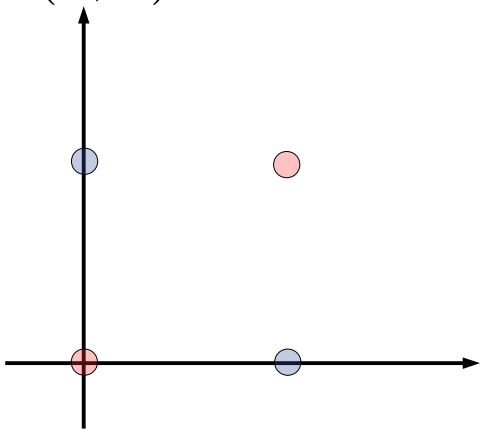
  $\boldsymbol{x}_1 = (0, 0),\ \boldsymbol{x}_2 = (1, 0),\ \boldsymbol{x}_3 = (0, 1),\ \text{and}\ \boldsymbol{x}_4 = (1, 1)$

- With the labels

  $y_1 = 0, y_2 = 1,\ y_3 = 1,\ \text{and}\ y_4 = 0$

- And a linear (single-layer) network

$$g_1(\boldsymbol{x}; \boldsymbol{w}) = x_1 w_1 + x_2 w_2$$



- There are no parameters that solve this problem

4.5 Single-layer networks cannot solve XOR

- No parameters solve this problem

$$
\begin{aligned}
0 &= 0\ w_1 + 0\ w_2 \\
1 &= 1\ w_1 + 0\ w_2 \\
1 &= 0\ w_1 + 1\ w_2 \\
0 &= 1\ w_1 + 1\ w_2.
\end{aligned}
$$

- Adding sigmoids or softmax activations does not help
- Solutions?

4.5 Single-layer networks cannot solve XOR

- Use a hidden layer or two-layer network

$$g_2(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{w}^T \max(0, \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{w}^T \boldsymbol{h}$$

- We can find parameters that solve the problem. For example:

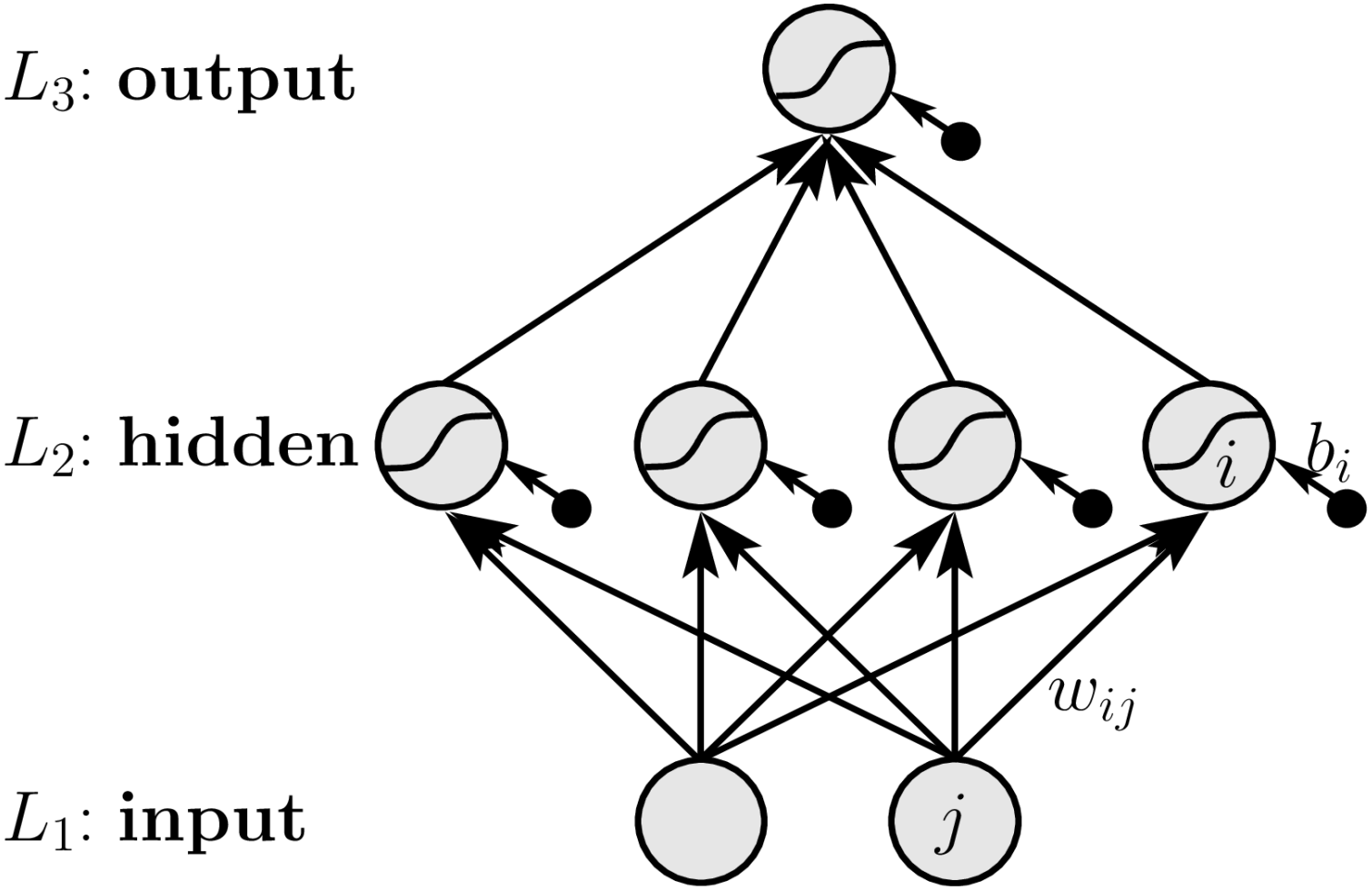
$$\boldsymbol{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \text{ and } \boldsymbol{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

- The transformation $\max(0, \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ has mapped the points into a space where they are linearly separable

Summary on single-layer networks and XOR

- We have indicated that single-layer networks cannot solve XOR
- A simple two-layer network has solved it (using *ReLU non-linearity*)
- Multi-layer perceptrons are even universal function approximators
  - → Universal function approximator theorem (Hornik, 1989)

Multi-layer perceptron



Multi-layer perceptron: quantities

- a_i: *activity* of the i-th unit

Multi-layer perceptron: quantities

- s_i: *network input* (pre-activation) to the i-th unit ($i > D$) computed as

$$s_i = \sum_{j=0}^{Q} w_{ij} a_j \tag{1}$$

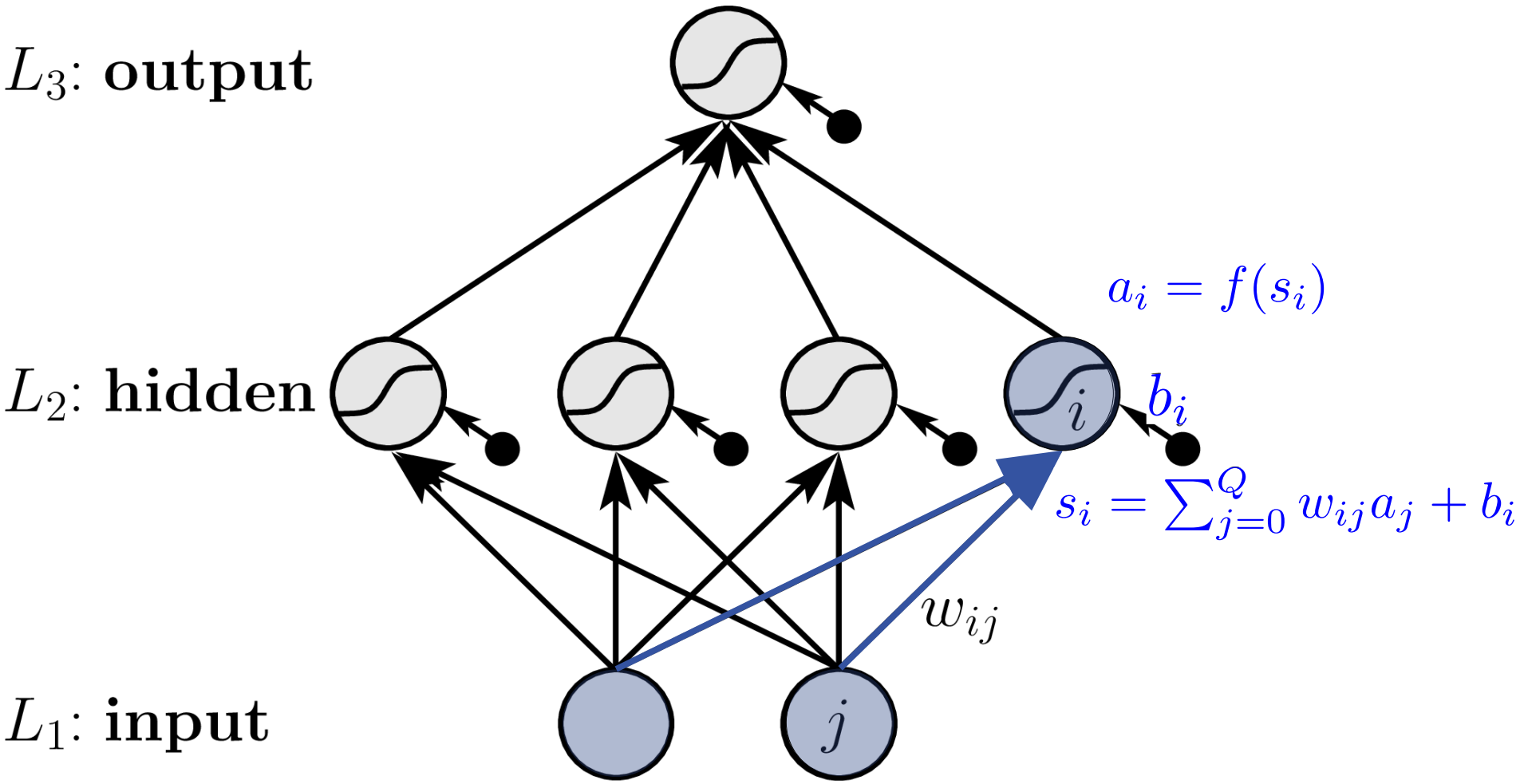
- f: *activation function* with

$$a_i = f(s_i) \tag{2}$$

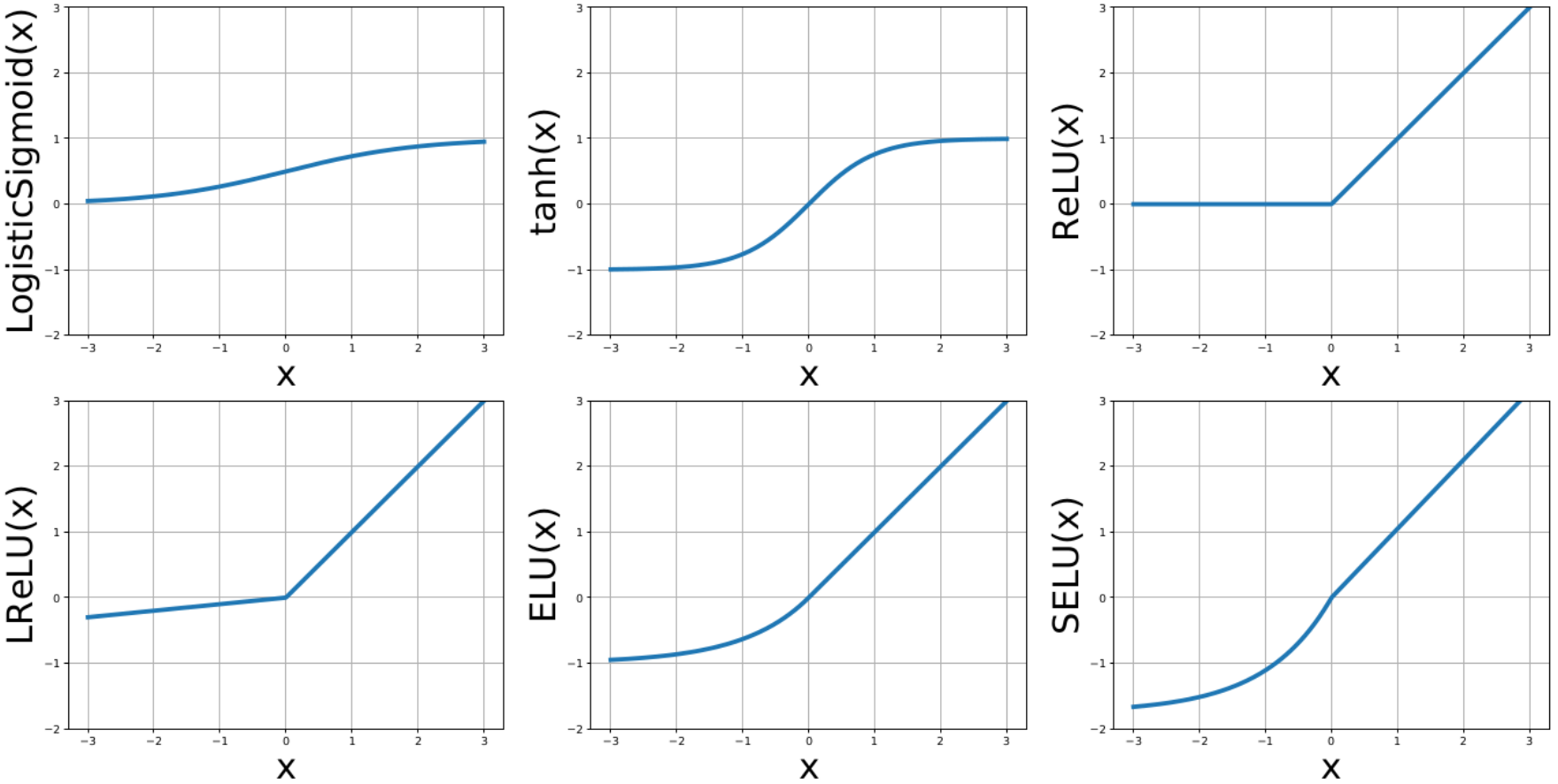
  It is possible to define different activation functions f_i for different units. The activation function is sometimes called *transfer function* (in more realistic networks one can distinguish between activation of a neuron and the signal which is transferred to other neurons).

- the *architecture* of a neural network is given through number of layers, units in the layers, and defined connections between units – the activations function may be accounted to the architecture.

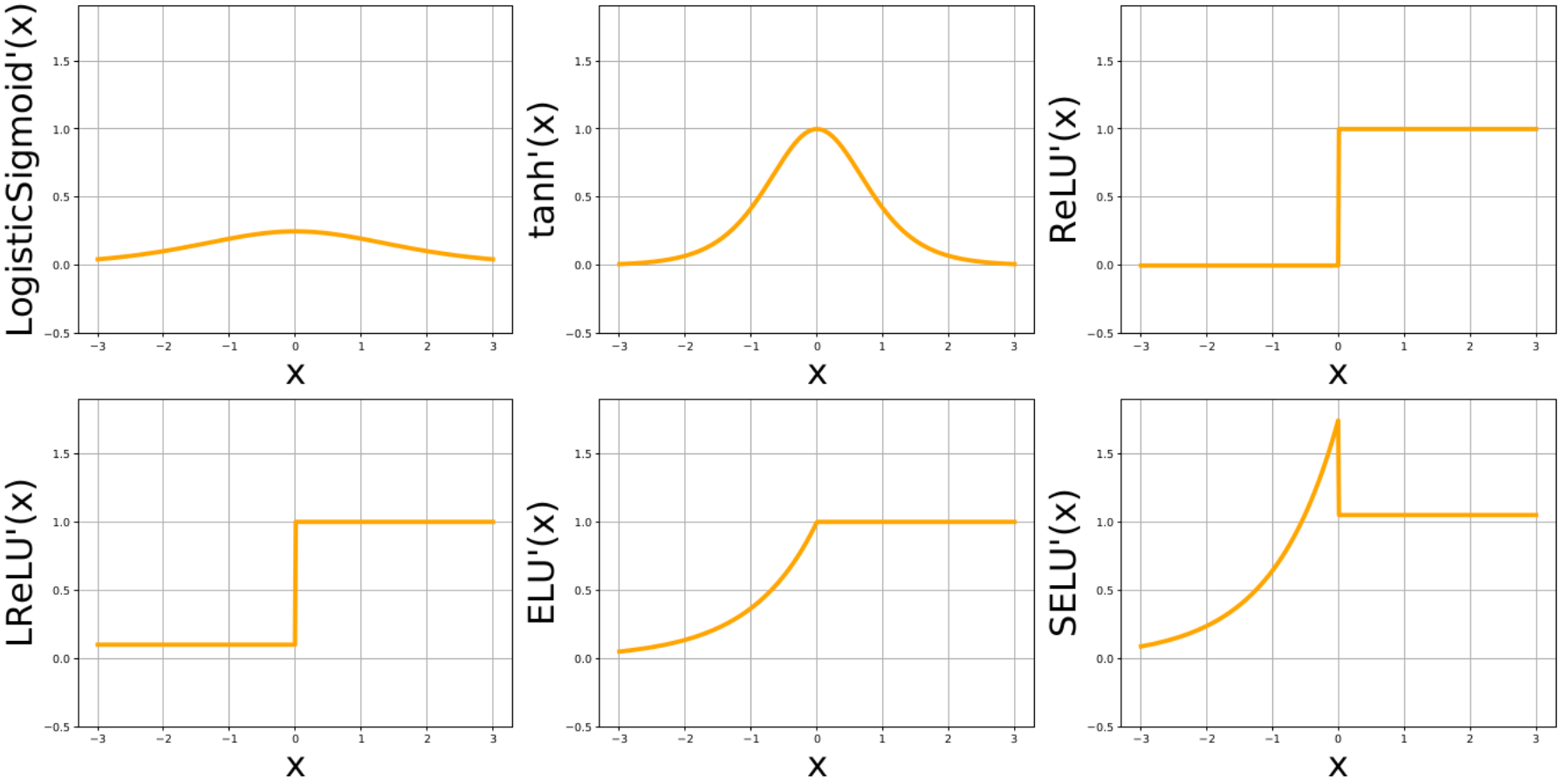
Multi-layer perceptron



A note on activation functions

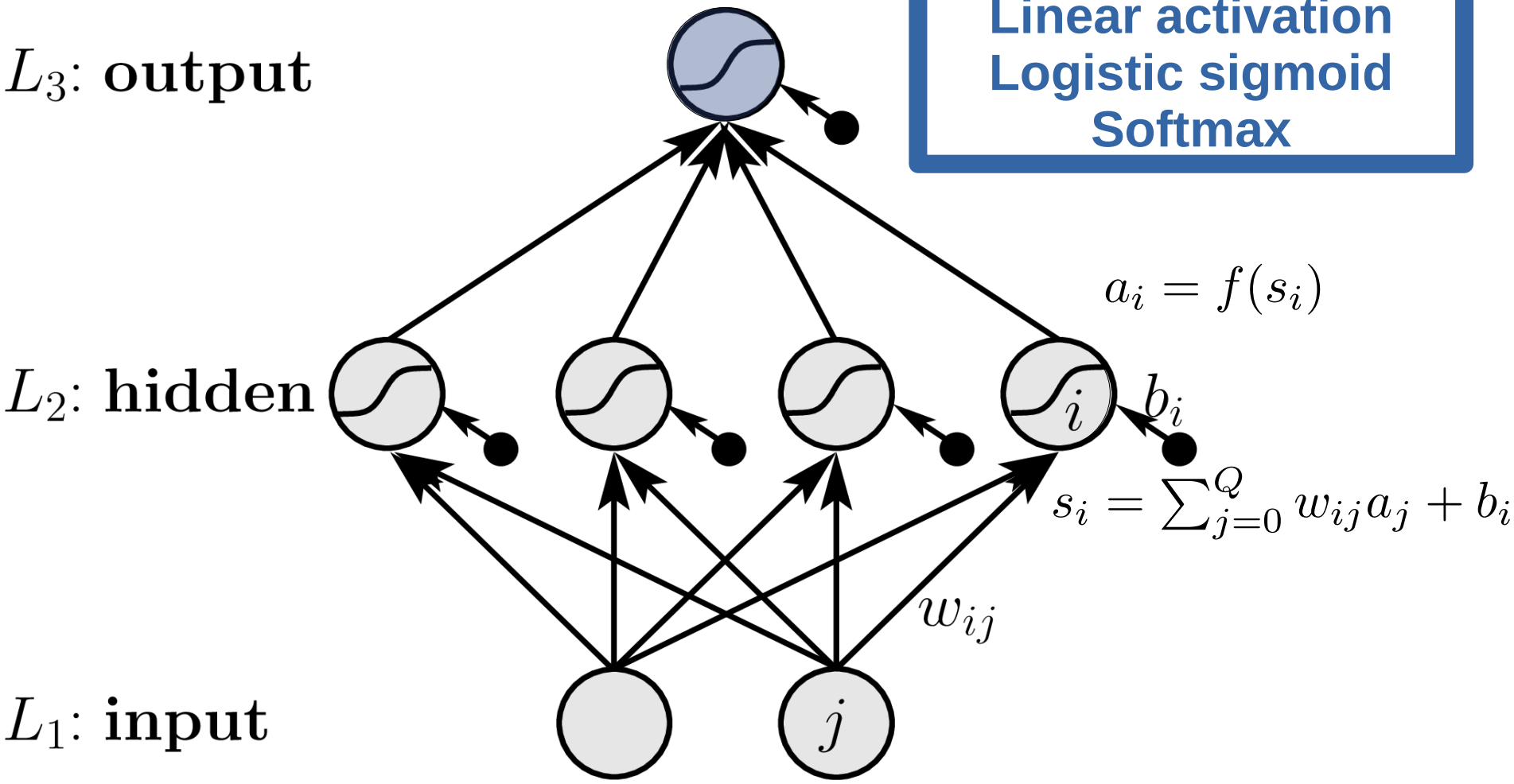


A note on activation functions: derivatives



Multi-layer perceptron: activations at output layer

Usually:
Linear activation
Logistic sigmoid
Softmax



Forward-pass of an MLP: pseudo-code

Algorithm .1 Forward Pass of an MLP

BEGIN initialization
  provide input $\boldsymbol{x}$
  for all $(i = 1; i \leq D;\ i++)$ **do**
    $a_i = x_i$
  end for
END initialization
BEGIN Forward Pass
  for $(\nu = 2; \nu \leq L; \nu++)$ **do**
    for all $i \in L_\nu$ **do**

$$s_i = \sum_{j=0; w_{ij} \text{ exists}}^{Q} w_{ij} a_j$$

      $a_i = f(s_i)$
    end for
  end for
  provide output $\hat{y}_i = (g(\boldsymbol{x}; \boldsymbol{w}))_i = a_i$, for all $Q - K + 1 \leq i \leq Q$
END Forward Pass

4.6.2. Gradient descent and the backpropagation algorithm

- Training of MLPs by *back-propagation*
- Famous algorithm made popular by Rumelhart et al. (1986); proposed earlier by Werbos (1974).
- Also called *delta-propagation*

Backpropagation

- Same setting as before: we aim at minimizing the empirical error by gradient descen

$$R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{N} \sum_{n=1}^{N} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right)$$

  - $g(\boldsymbol{x}^n, \boldsymbol{w})$ is the network and $\boldsymbol{w}$ contains all weights (weight matrices) in all layers and (potentially) biases

- We want to perform a gradient descent update

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \, \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{w})$$

  or for a single data point:

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \, \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w})$$

Backpropagation: derivative w.r.t. weights

- We derive the loss w.r.t. the weights:

$$\frac{\partial}{\partial w_{ij}} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right)$$

- Using the chain rule, we obtain:

$$\begin{aligned}\frac{\partial}{\partial w_{ij}} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) &= \frac{\partial}{\partial s_i} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) \frac{\partial s_i}{\partial w_{ij}} \\ &= \frac{\partial}{\partial s_i} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) a_j,\end{aligned}$$

where we used $\frac{\partial s_i}{\partial w_{ij}} = a_j$

Backpropagation: Defining delta errors

- We define the delta error at unit i as:

$$\delta_i := \frac{\partial}{\partial s_i} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right)$$

- And we obtain:

$$\frac{\partial}{\partial w_{ij}} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) = \delta_i a_j.$$

- Compare a result from before (single-layer nets):

$$\frac{\partial}{\partial w_{ij}} R_{\text{emp}}(\boldsymbol{w}, \boldsymbol{x}, y) = \underbrace{\left(\text{softmax}(\boldsymbol{W}\boldsymbol{x})_i - y_i\right)}_{=\delta_i} \underbrace{x_j}_{=a_j},$$

Backpropagation: delta error at output units

- Delta errors at output units:

$$\delta_k = \frac{\partial}{\partial a_k} L\left(\boldsymbol{y}^n, g\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) f'(s_k)$$

  where we have

  - activations $a_k = g\left(\boldsymbol{x}^n; \boldsymbol{w}\right)$ $\quad Q - K + 1 \leq k \leq Q$
  - pre-activations s_k $\quad Q - K + 1 \leq k \leq Q$
    at the output layer

- This is the same as for linear, log., softmax regr

Backpropagation: delta error at hidden units

- The delta errors at the hidden units are

$$\delta_j = \frac{\partial}{\partial s_j} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) = \sum_i \frac{\partial}{\partial s_i} L\left(\boldsymbol{y}^n, \boldsymbol{g}\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) \frac{\partial s_i}{\partial s_j}$$

$$= f'(s_j) \sum_i \delta_i w_{ij}$$

- where the sum goes over all units for which the weight exists (the units in the layer above)

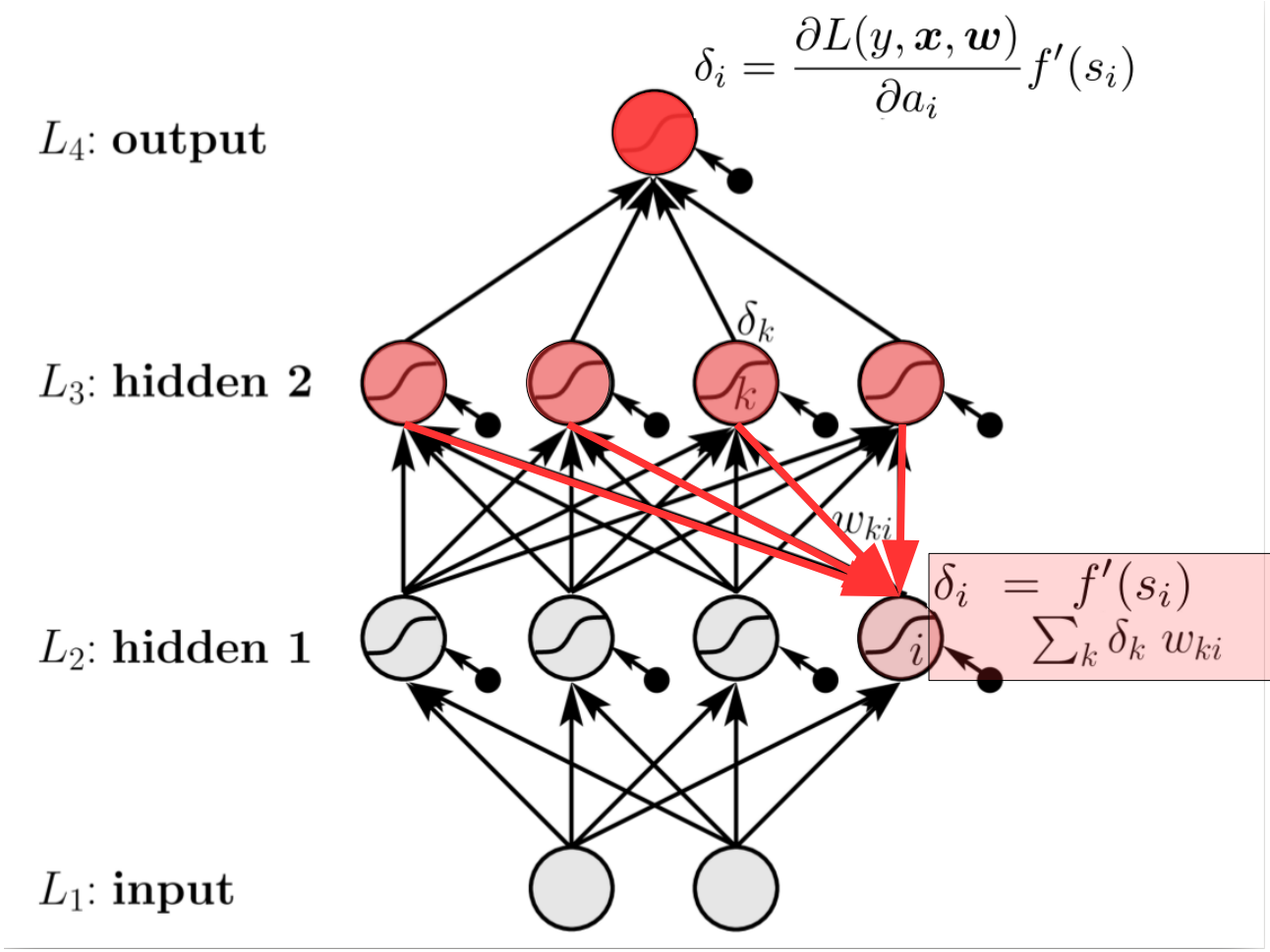
Canonical link functions

Loss function		canonical link	delta-error
squared loss	$1/2(a-y)^2$	linear: $f(x) = x$	$\delta = (a - y)$
binary CE	$-y\log(a) - (1-y)\log(1-a)$	sigmoid: $f(x) = \frac{1}{1+e^{-x}}$	$\delta = (a - y)$
categorical CE	$-\sum_{k=1}^{K} y_k \log(a_k)$	softmax: $f(\boldsymbol{x}) = \text{softmax}(\boldsymbol{x})$	$\boldsymbol{\delta} = (\boldsymbol{a} - \boldsymbol{y})$

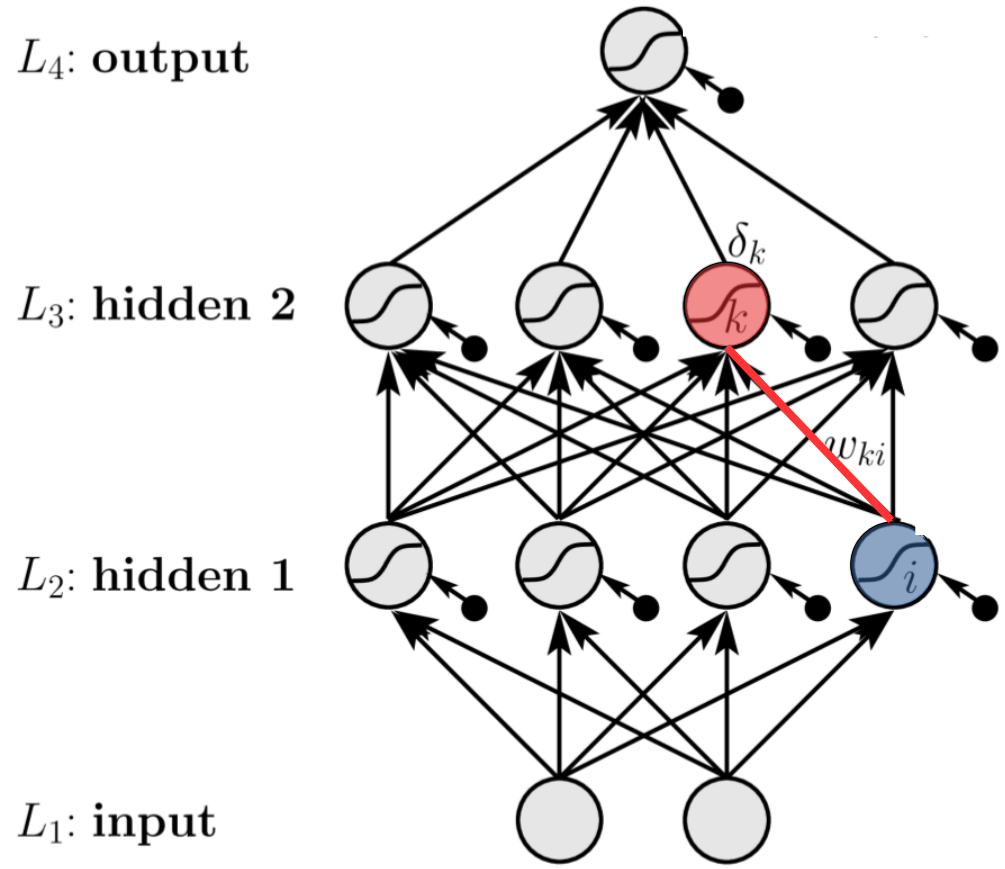
Table 1: Frequently used loss functions for neural networks and their canonical link activation functions.

- Potential problems of other combinations?

Delta propagation



Weight update



$$\frac{\partial L}{\partial w_{ki}} = \delta_k a_i$$

Recap: backpropagation

- Select a sample x randomly from the training set and

  perform a forward-pass. Memorize activations and preactivations of all neurons.
- Calculate delta-errors at output units: $\delta_k = \frac{\partial}{\partial a_k} L\left(\boldsymbol{y}^n, g\left(\boldsymbol{x}^n; \boldsymbol{w}\right)\right) f'(s_k)$
- Calculate delta-errors for hidden layers starting from the hidden layers closest to the output layer, thus backpropagating the error signal according to $\delta_j = f'(s_j) \sum_i \delta_i w_{ij}$
- Calculate gradients for weights according to $\frac{\partial L}{\partial w_{ij}} = \delta_i a_j$
- Update the weights by performing a gradient descent step using the weight changes $\Delta w_{ij} = -\eta \delta_i \; a_j$

Backpropagation: pseudo-code

Algorithm .1 Backward Pass of an MLP

BEGIN initialization
provide activations a_i of the forward pass and the label y
for $(i = Q - K + 1; i \leq Q; i++)$ **do**
$\delta_i = \dfrac{\partial L(y, \boldsymbol{x}, \boldsymbol{w})}{\partial a_i} f'(s_i)$
for all $j \in L_{L-1}$ **do**
$\Delta w_{ij} = -\eta \delta_i a_j$
end for
end for
END initialization
BEGIN Backward Pass
for $(\nu = L - 1; \nu \geq 2; \nu --)$ **do**
for all $i \in L_\nu$ **do**
$\delta_i = f'(s_i) \sum_k \delta_k w_{ki}$
for all $j \in L_{\nu-1}$ **do**
$\Delta w_{ij} = -\eta \delta_i a_j$
end for
end for
end for
END Backward Pass

Remark: efficiency of backprop

- The name “backprop” refers to the fact that errors (deltas) because the delta errors of one layer are used to calculate the deltas for the layer below
- Complexity is $\mathcal{O}(W)$
- A similar approach can be used to calculate the second derivative (not detailed now)

Remark: full batch, stochastic and online learning

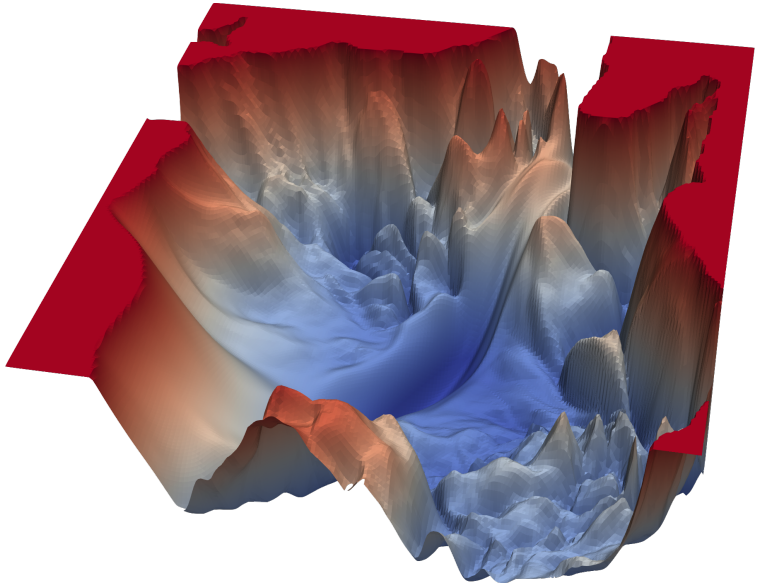
- The gradient can be calculated on the full training set, and then an update step can be taken: *full-batch gradient*
- Calculate gradient on single sample and perform update: *online learning*
- Sample a small subset, e.g. 32, samples, calc gradient and perform update: *minibatch training or stochastic gradient desc.*

Training MLPs: architecture and hyperparams

- MLPs now offer many choices
  - Number of layers
  - Number of neurons
  - Activation function
  - Learning rate
- Minimizing the empirical error is not sufficient (see Chapter 5)
  - Optimize architecture on *validation set*
  - Evaluate final model on *test set*

Remark: optimization problem and loss surface

- Loss surface is highly non-convex
- Uniqueness? No! *Weight space symmetries*
  - If there is a parameter set with horizontal tangent, we can easily switch signs to get an equivalent set of parameters (e.g. tanh)
  - Also the order of neurons can be changed
- However: Some theoretic results that optimization by SGD still yields ‘good’ minima (see Part II)



Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems (pp. 6389-6399).

MLPs: Summary

- XOR problems
- Architecture of MLPs
  - Forward pass
  - Backward pass (backpropagation)
- Remarks on optimization, etc