Deep Feedforward Networks

Simon Jenni
(slides by Paolo Favaro)

UNIVERSITÄT BERN

Contents

- Introduction to Feedforward Neural Networks: definition, design, training

- Based on **Chapter 6** (and 4) of Deep Learning by Goodfellow, Bengio, Courville

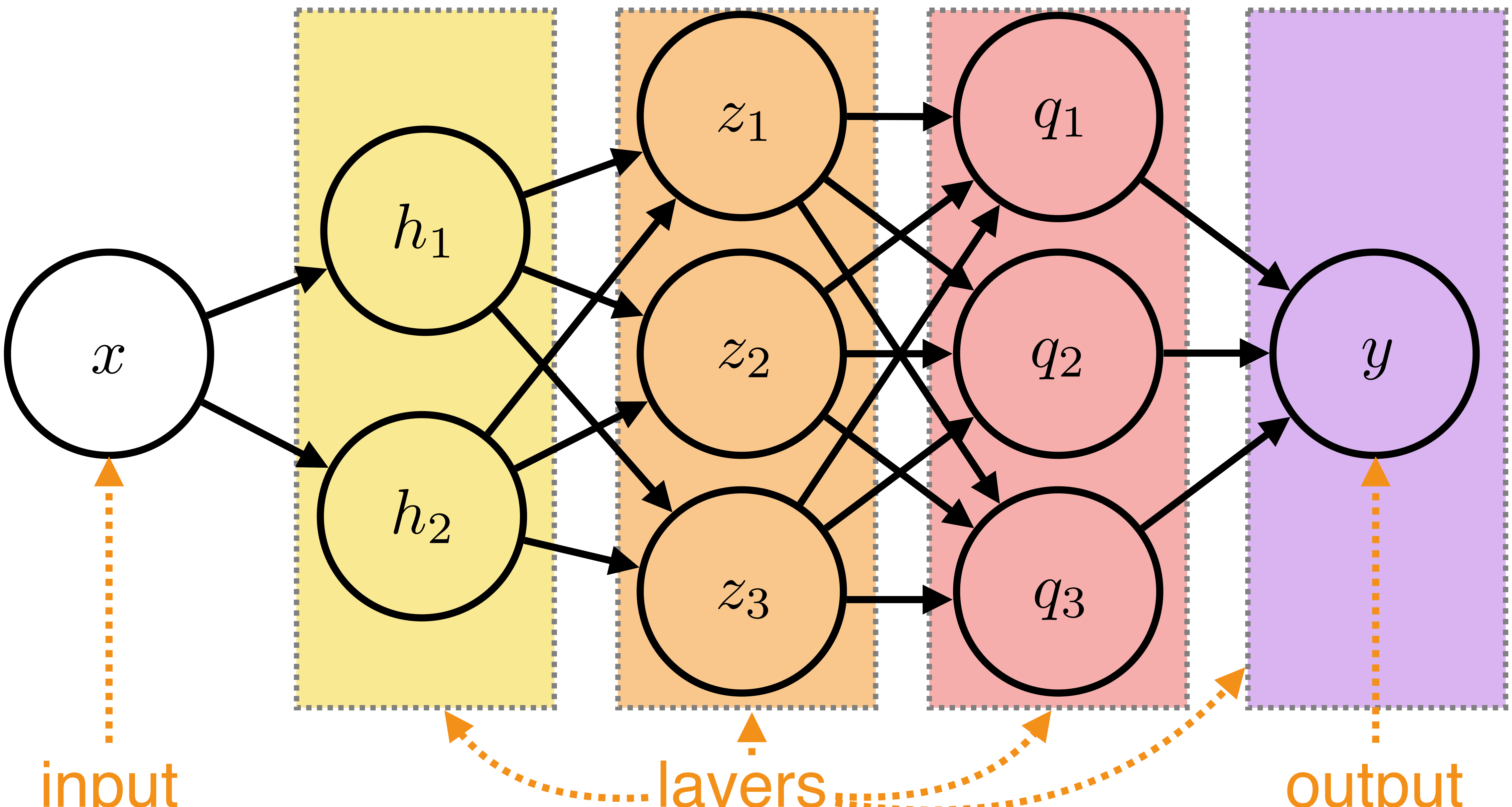
- References to Machine Learning and Pattern Recognition by Bishop

Resources

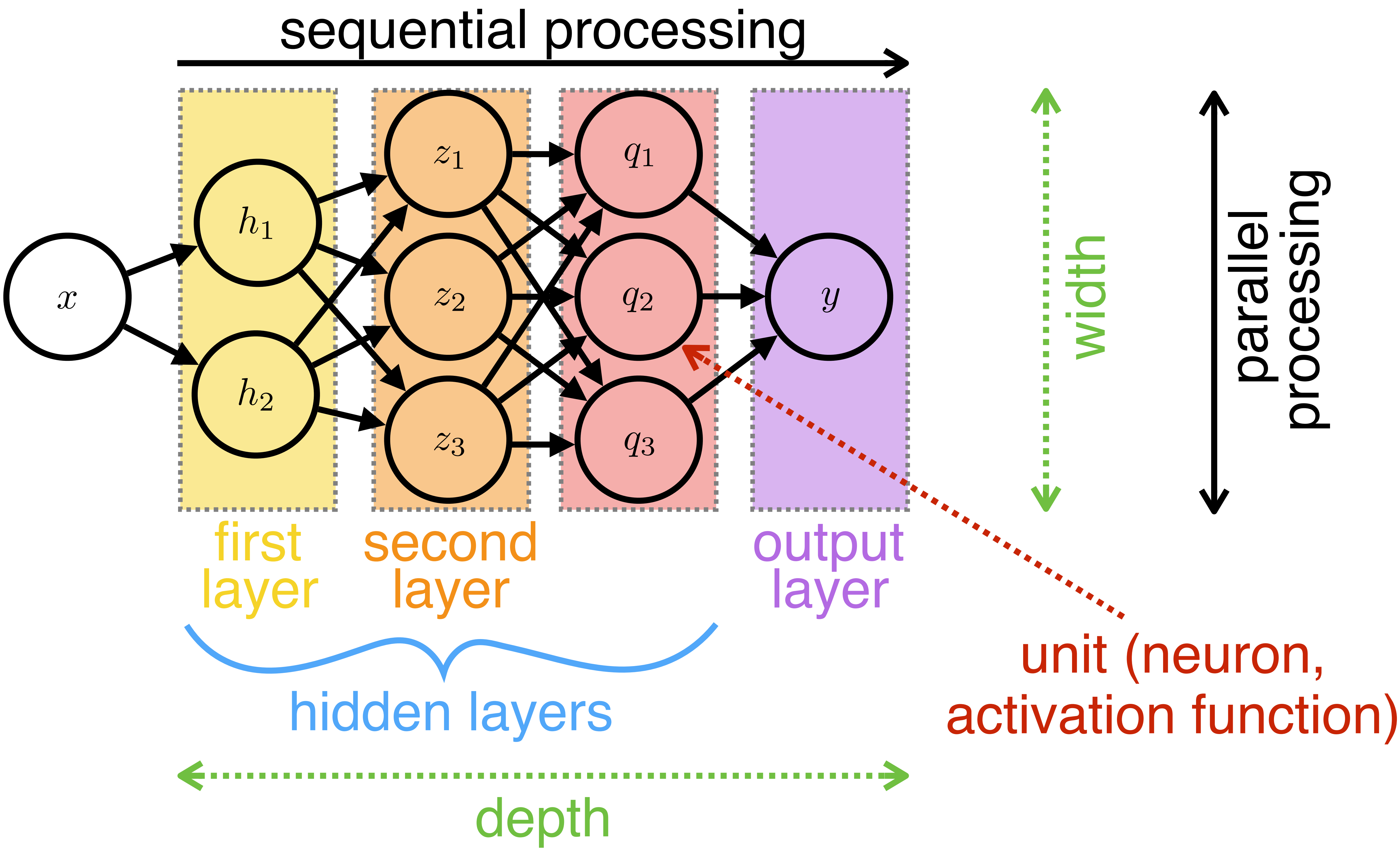
- Books and online material for further studies
  - CS231 @ Stanford (Fei-Fei Li)
  - **Pattern Recognition and Machine Learning**
    by Christopher M. Bishop
  - **Machine Learning: a Probabilistic Perspective**
    by Kevin P. Murphy

Feedforward Neural Networks

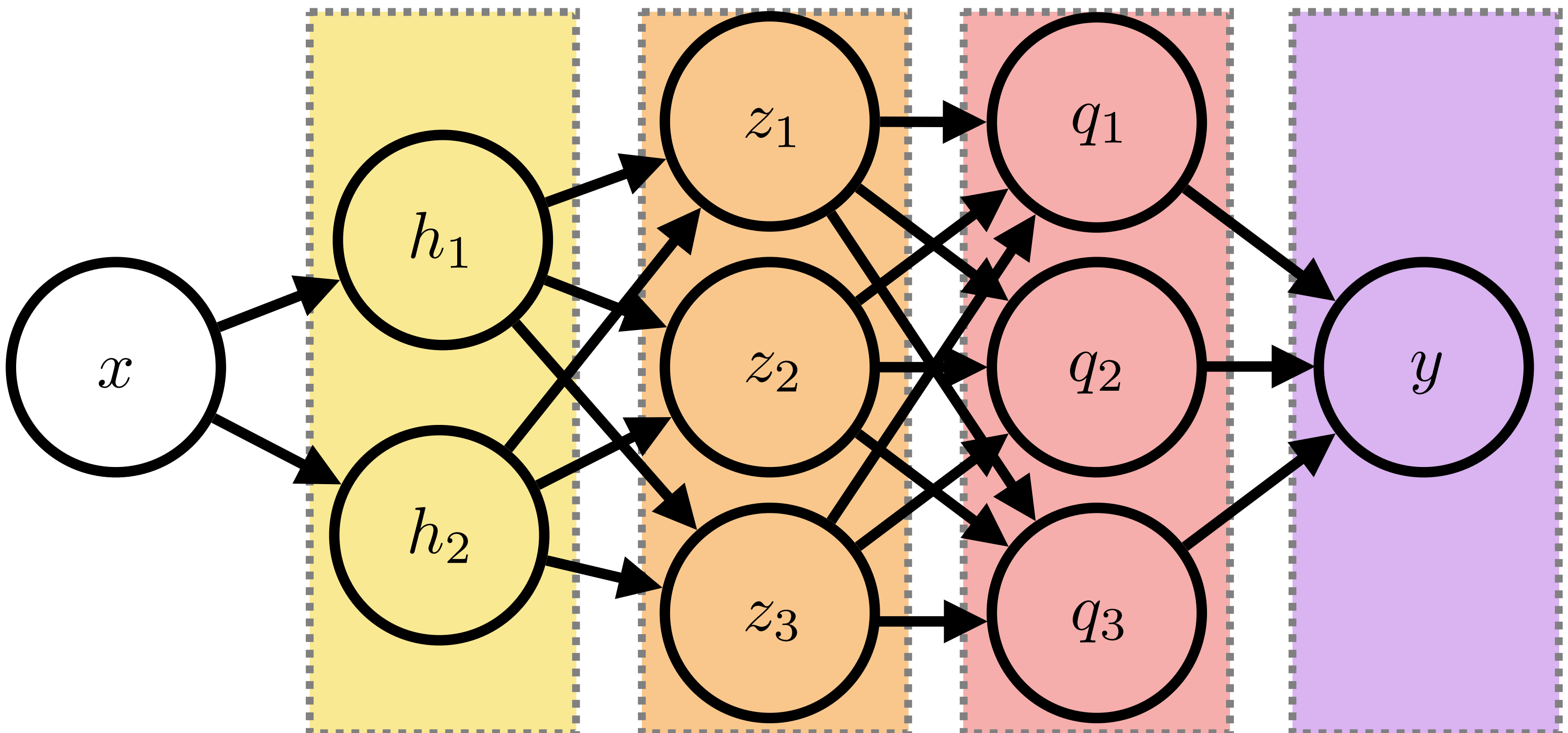
- Feedforward networks are a sequence of layers, each processing the output of the previous layer(s)



Feedforward Neural Networks



Feedforward Neural Networks



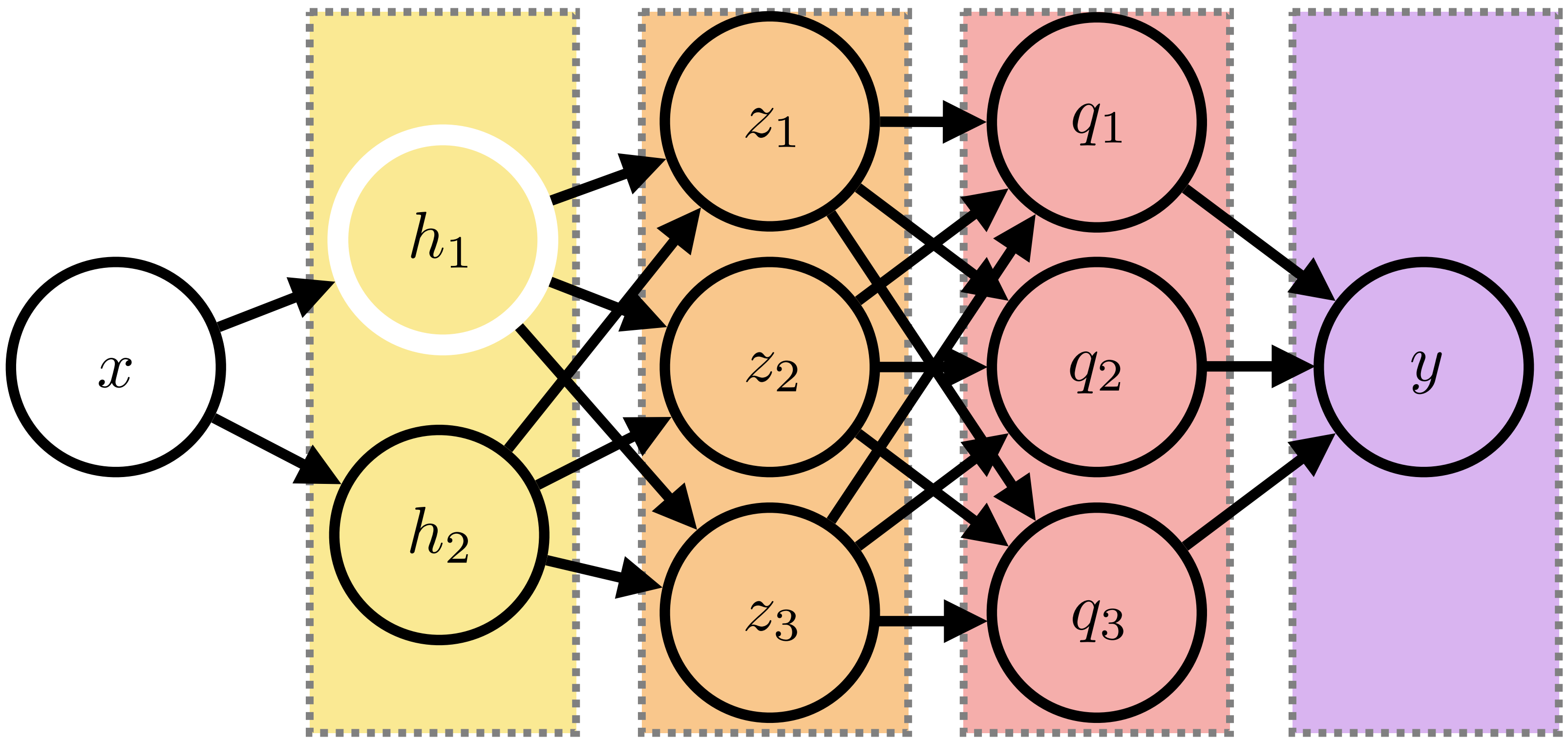
$$y = f_4(q_1, q_2, q_3)$$

$$h_1 = f_{1,1}(x)$$
$$h_2 = f_{1,2}(x)$$

$$z_1 = f_{2,1}(h_1, h_2)$$
$$z_2 = f_{2,2}(h_1, h_2)$$
$$z_3 = f_{2,3}(h_1, h_2)$$

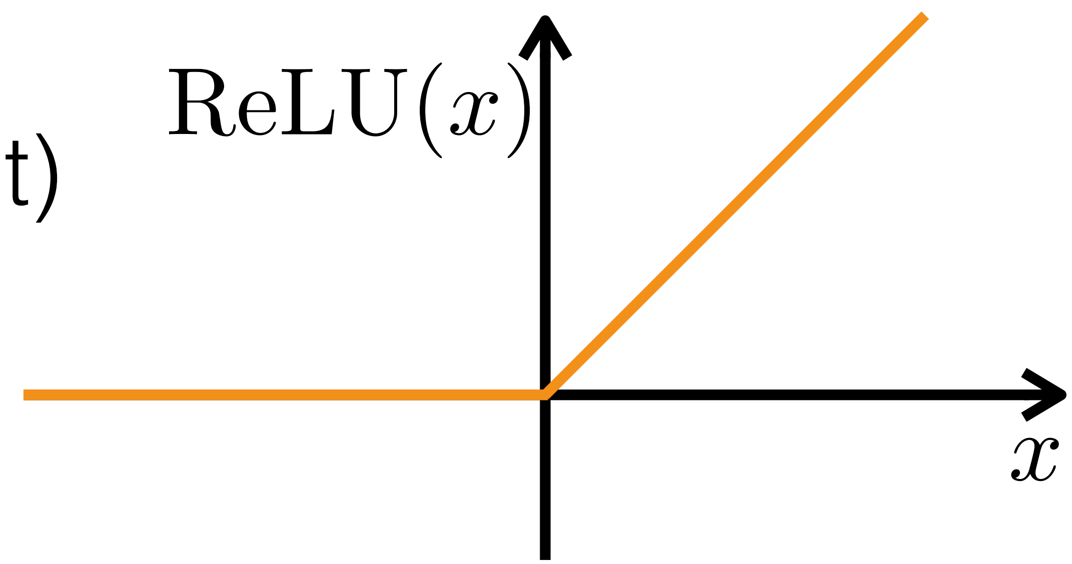
$$q_1 = f_{3,1}(z_1, z_2, z_3)$$
$$q_2 = f_{3,2}(z_1, z_2, z_3)$$
$$q_3 = f_{3,3}(z_1, z_2, z_3)$$

Feedforward Neural Networks

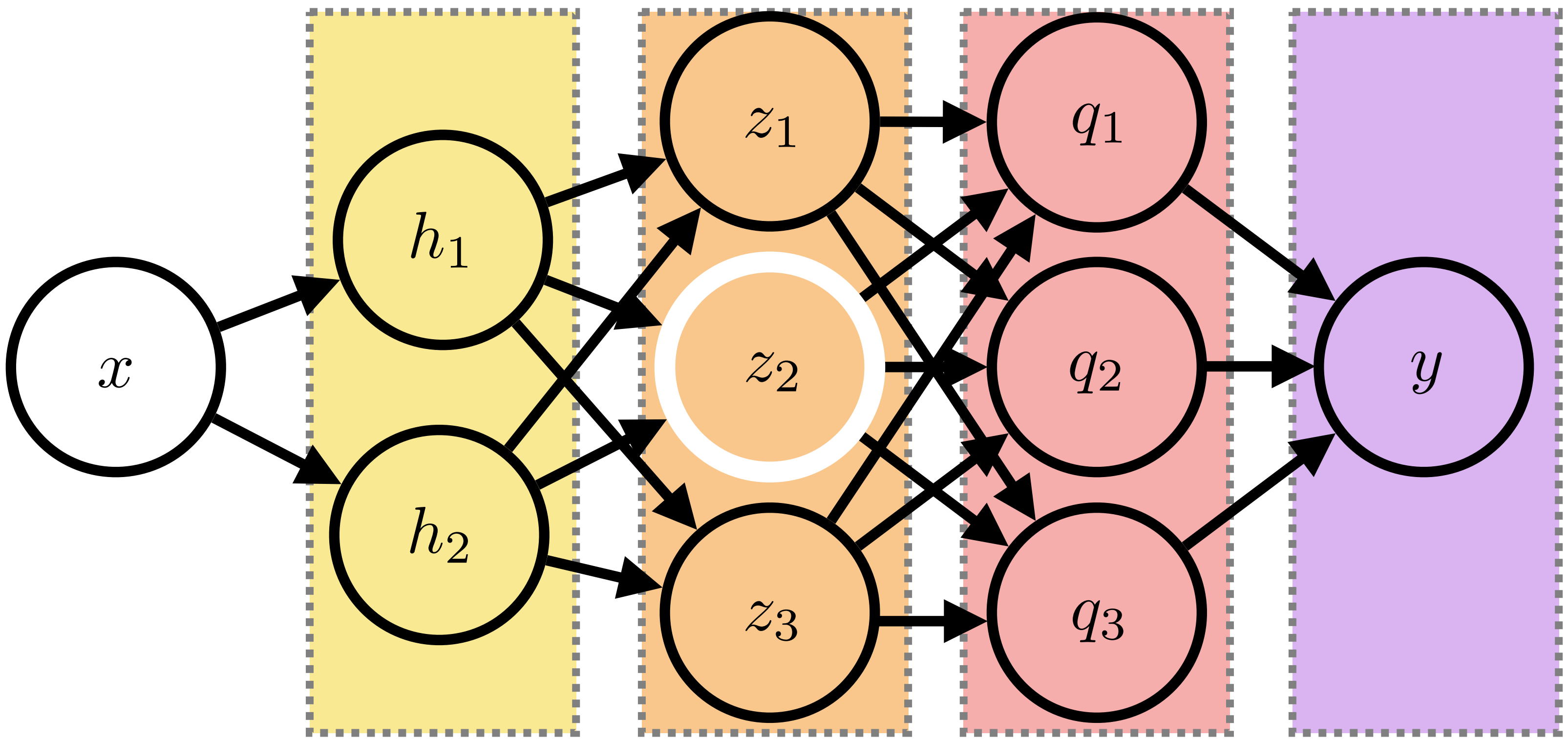


Example (rectified linear unit)

$$f_{1,1}(x) = \mathrm{ReLU}(x)$$



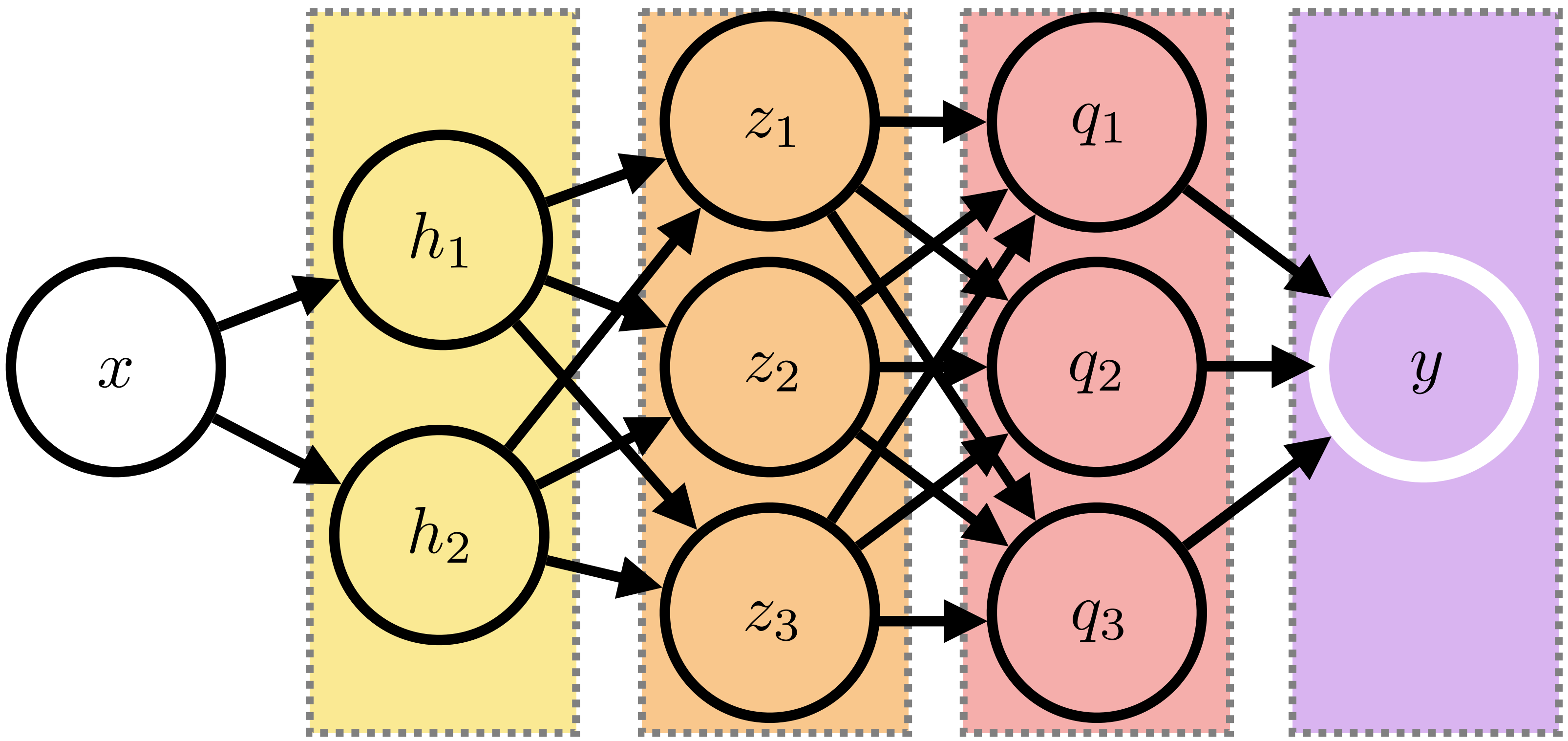
Feedforward Neural Networks



Example (fully connected unit)

$$f_{2,2}(h_1, h_2) = w_1 h_1 + w_2 h_2$$

Feedforward Neural Networks



$$y = f_4(q_1, q_2, q_3)$$

Hierarchical composition of functions

$$y = f_4(f_{3,1}(f_{2,1}(f_{1,1}(x), f_{1,2}(x)), \ldots), \ldots)$$

Feedforward Neural Networks

- Feedforward neural networks define a family of functions $f(x;\theta)$

- The goal is to find parameters θ that define the best mapping

$$y = f(x;\theta)$$

  between input **x** and output **y**

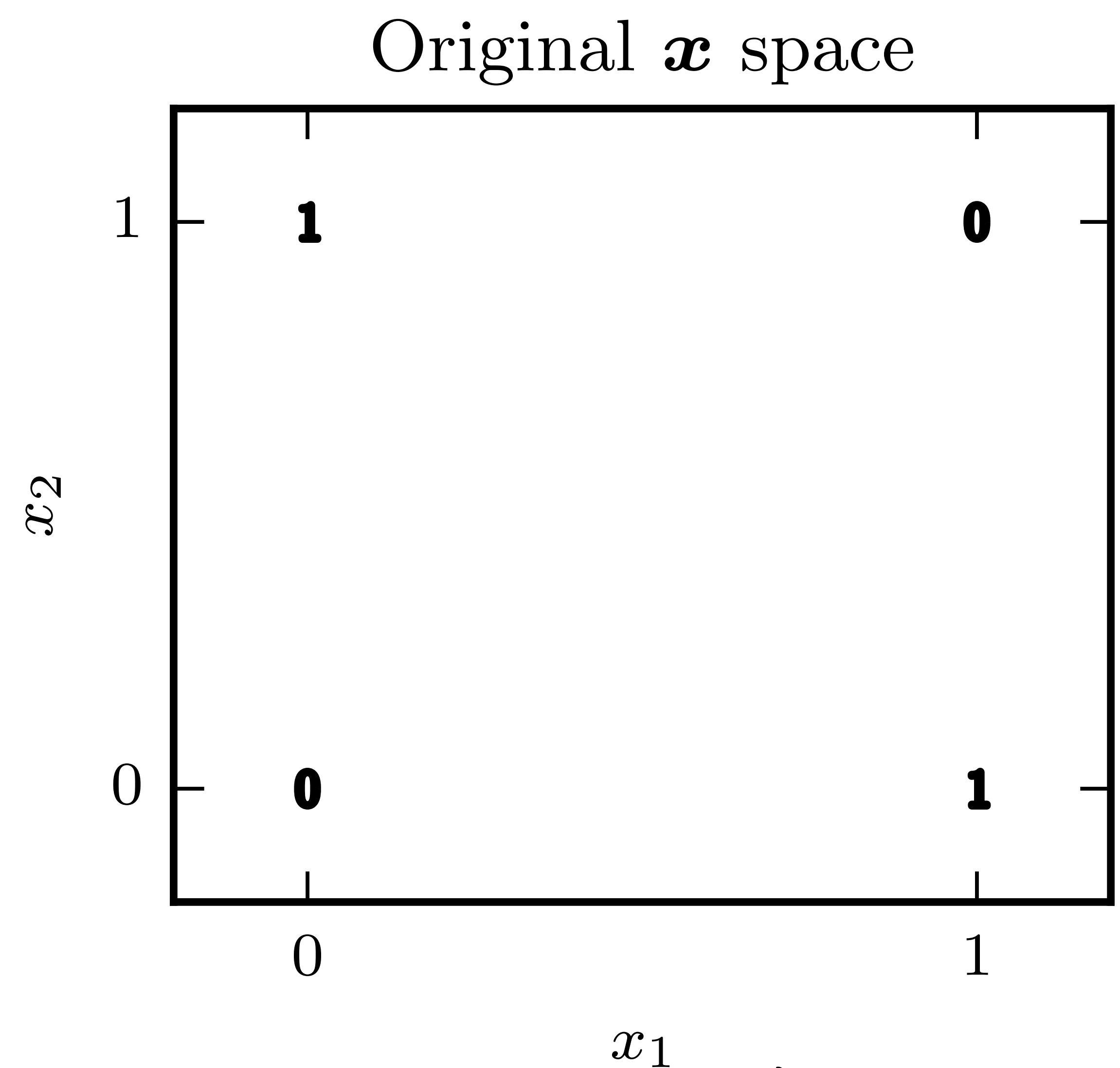
- The key constraints are the I/O dependencies

Deploying a Neural Network

- Given a **task** (in terms of I/O mappings)
- We need
    - **Cost function**
    - **Neural network model** (e.g., choice of units, their number, their connectivity)
    - **Optimization method** (back-propagation)

Example: Learning XOR

- Objective is the XOR operation between two binary inputs $\mathbf{x}_1$ and $\mathbf{x}_2$



- Training set of $(\mathbf{x},\mathbf{y})$ pairs

$$\left\{ \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, 0 \right), \left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}, 1 \right), \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, 1 \right), \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, 0 \right) \right\}$$

Cost Function

- Let us use the Mean Squared Error (MSE) as a first attempt

$$J(\theta) = \frac{1}{4} \sum_{i=1}^{4} \left(y^i - f(x^i; \theta)\right)^2$$

Linear Model

- Let us try a linear model of the form

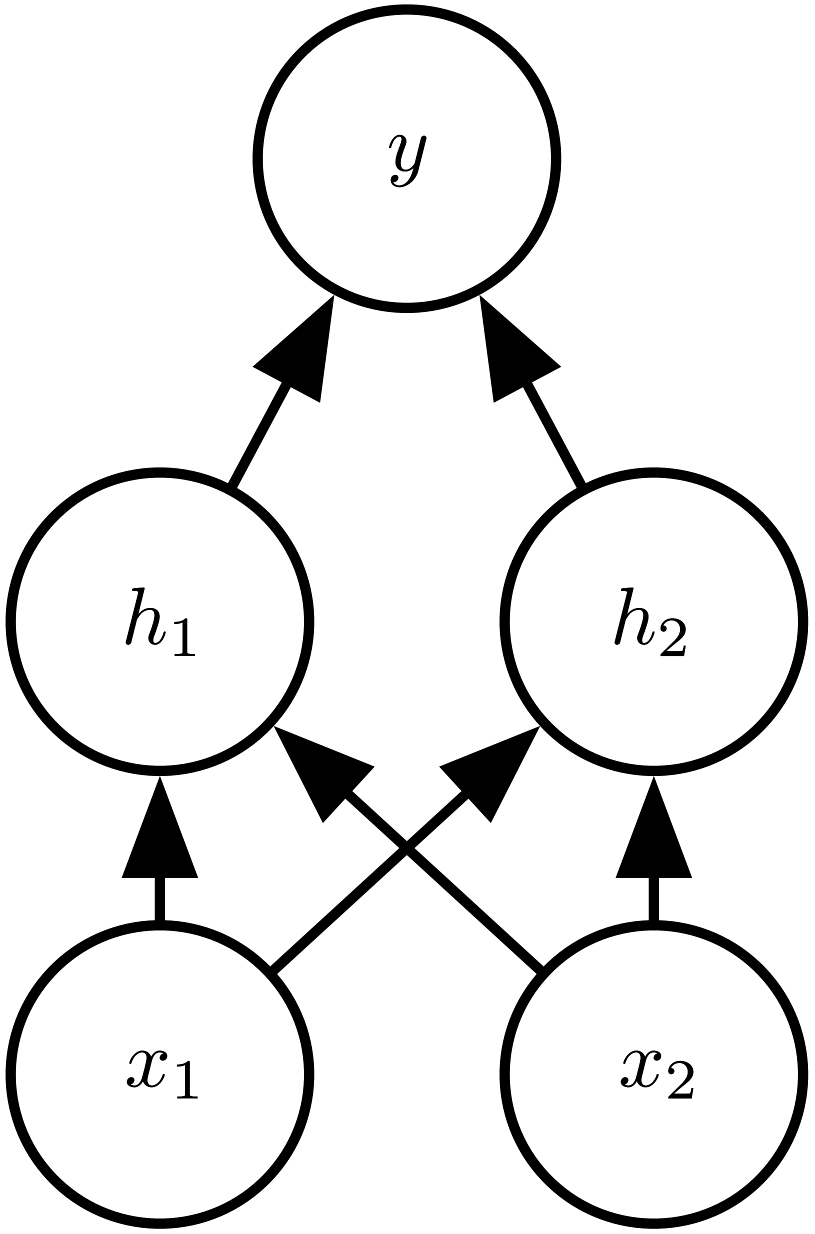
$$f(x; w, b) = w^\top x + b$$

- This choice leads to the normal equations (see slides on Machine Learning Review) and the following values for the parameters

$$w = 0, \qquad b = \frac{1}{2}$$

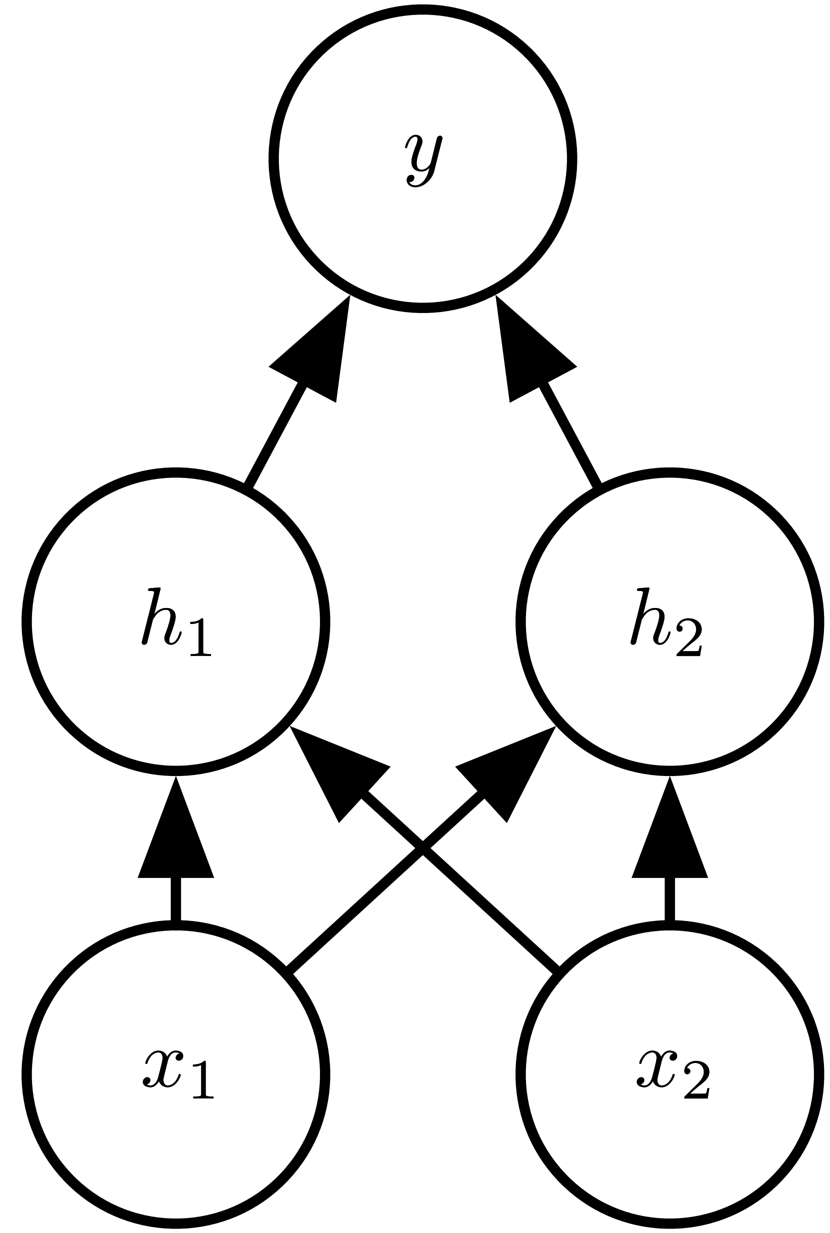
Nonlinear Model

- Let us try a simple feedforward network with one hidden layer and two hidden units



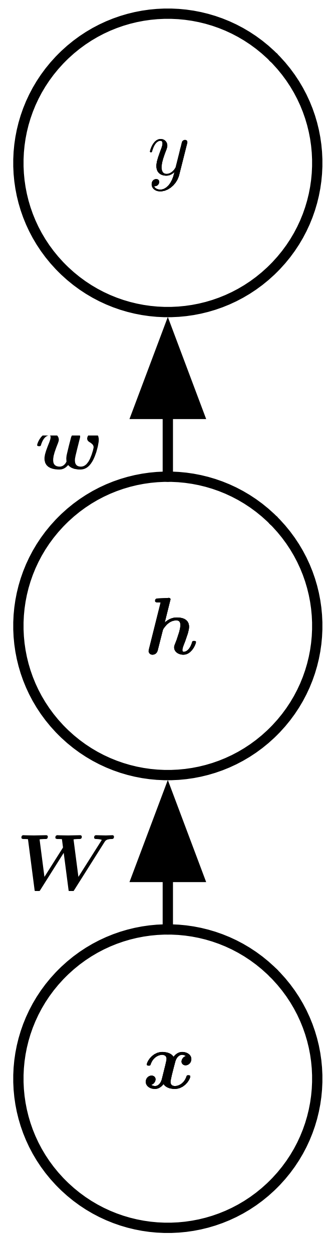
Nonlinear Model

- If each activation function is linear then the composite function would also be linear
- We would have the same poor result as before
- We must consider nonlinear activation functions



Nonlinear Model

$$f(x; W, c, w, b) = w^\top \max\{0, W^\top x + c\} + b$$



$$y(h) = w^\top h + b$$

$$h(x) = \mathrm{ReLU}(W^\top x + c)$$

Optimization

$$f(x; W, c, w, b) = w^\top \max\{0, W^\top x + c\} + b$$

At this stage we would use optimization to fit **f** to the **y** in the training set. In this example, we skip this step and assume that some oracle gives us the parameters

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

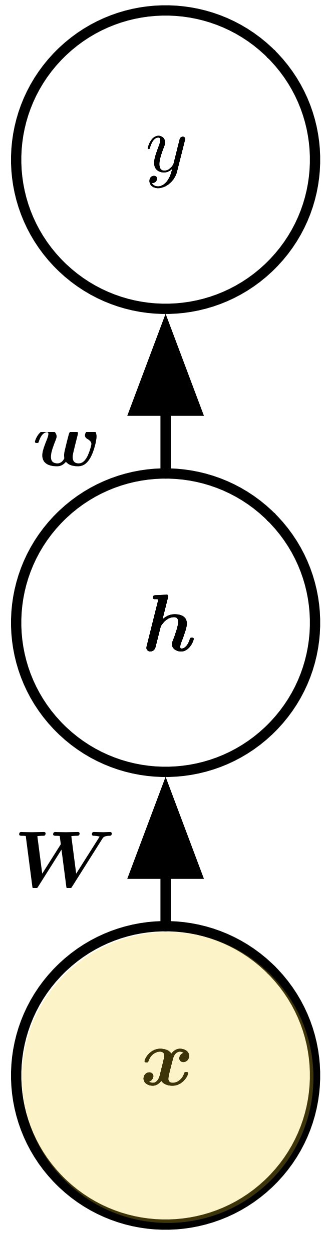
$$w = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \qquad b = 0$$

Simulation

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \Rightarrow XW + \mathbf{1}c = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

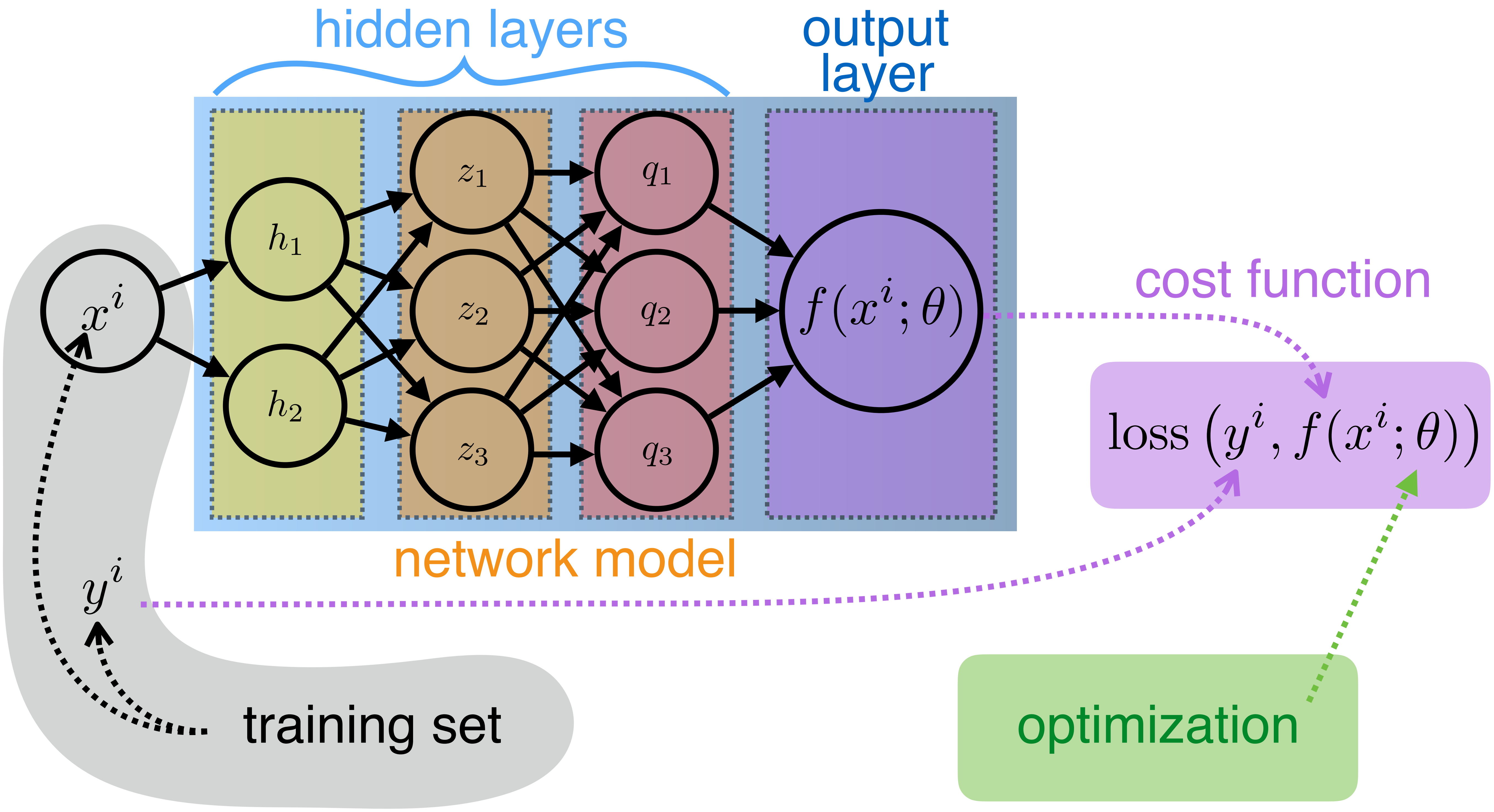
$$\Rightarrow \max\{0, XW + \mathbf{1}c\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\Rightarrow \max\{0, XW + \mathbf{1}c\}w + \mathbf{1}b = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



the XOR function
(matches Y)

Step-by-Step Analysis



Cost Function

- Based on the **conditional distribution** $p_{\text{model}}(y|x;\theta)$
  - Maximum Likelihood (i.e., **cross-entropy** between model pdf and data pdf)

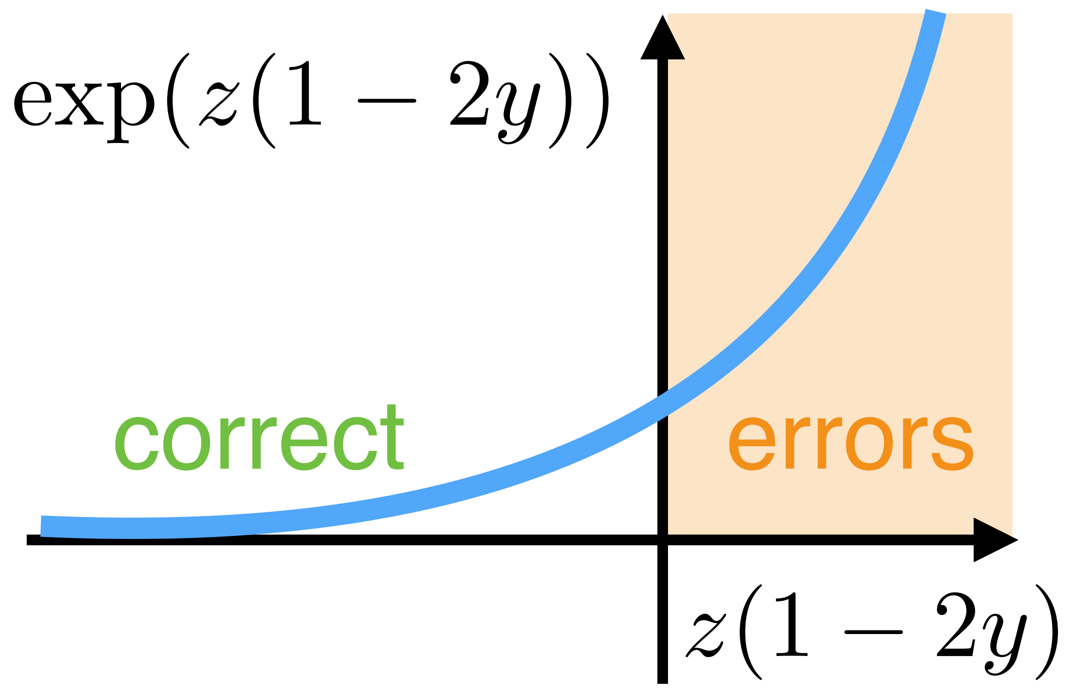
$$\min_{\theta} -E_{x,y\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(y|x;\theta)]$$

Saturation

- Functions that saturate (have flat regions) have a very small gradient and slow down gradient descent

- We choose loss functions that have a non flat region when the answer is incorrect (it might be flat otherwise)

- E.g., exponential functions saturate in the negative domain; with a binary variable $y \in \{0, 1\}$ map errors to the nonflat region and then minimize



- The logarithm also helps with saturation (see next slides)

Output Units

- The choice of the output representation (e.g., a probability vector or the mean estimate) determines the cost function

- Let us denote with

$$h = f(x; \theta)$$

the output of the layer before the output unit

Linear Units

- With a little abuse of terminology, linear units include **affine transformations**

$$\hat{y} = W^\top h + b$$

can be seen as the mean of the conditional Gaussian distribution (in the Maximum Likelihood loss)

$$p(y|x) = \mathcal{N}(y; \hat{y}, I)$$

- The Maximum Likelihood loss becomes

$$-\log p(y|\hat{y}) = |y - \hat{y}|^2 + \text{const}$$

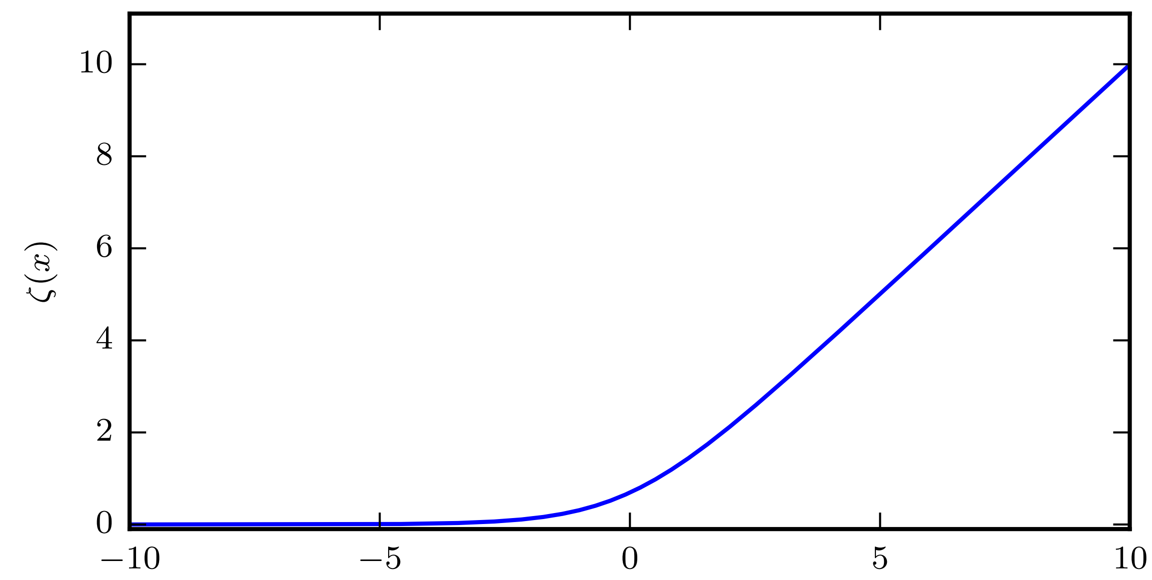
Softplus

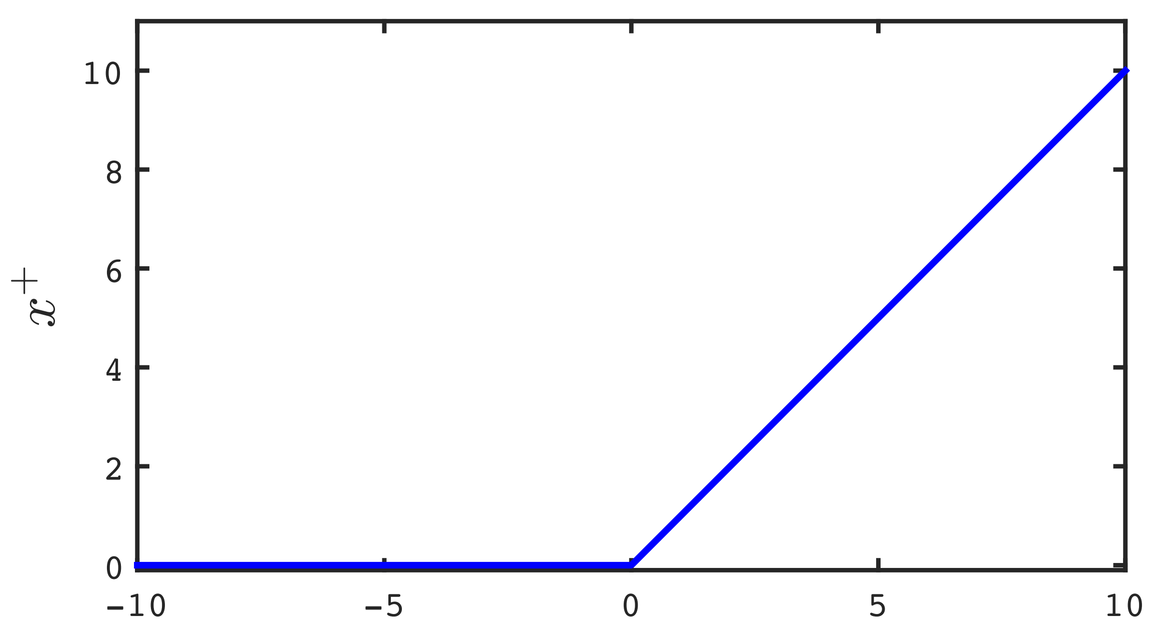
- The **softplus** function is defined as

$$\zeta(x) = \log(1 + \exp(x))$$

and it is a smooth approximation of the Rectified Linear Unit (ReLU)

$$x^{+} = \max(0, x)$$





Sigmoid Units

- Use to predict binary variables or to predict the probability of binary variables

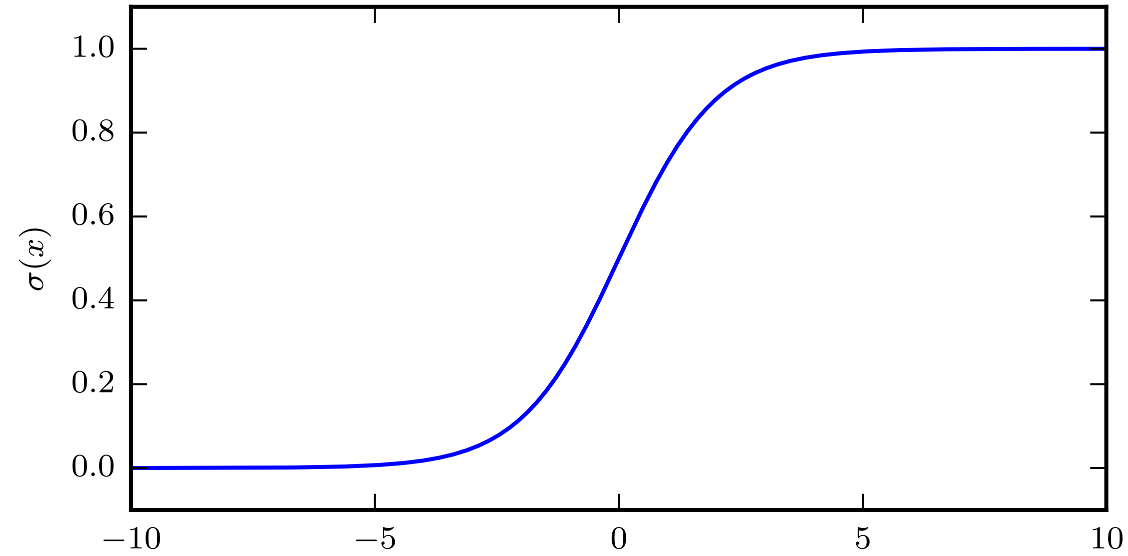
$$p(y = 0|x) \in [0, 1]$$

- The sigmoid unit defines a suitable mapping

$$\hat{y} = \sigma(w^\top h + b)$$

where we have used the **logistic sigmoid** function

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



Sigmoid Cross-Entropy

- Let $z = w^\top h + b$. Then, we can define the Bernoulli distribution

$$p(y|z) = \sigma(z)^y (1 - \sigma(z))^{(1-y)}$$

- The loss function with Maximum Likelihood is then

$$-\log p(y|z) = y \log \sigma(z) + (1 - y) \log(1 - \sigma(z))$$

and saturation occurs only when the output is correct (y=0 and z<0 or y=1 and z>0)

Smoothed Max

- An extension to the softplus function is the smoothed max

$$\log \sum_j \exp(z_j)$$

which gives a smooth approximation to $\max_j z_j$

- If we rewrite the softplus function as

$$\log(1 + \exp(z)) = \log(\exp(0) + \exp(z))$$

we can see that it is the case with $z_1 = 0, z_2 = z$

Softmax Units

- An extension of the logistic sigmoid to multiple variables
- Used as the output of a multi-class classifier
- The **Softmax** function is defined as

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- Shift-invariance: $\mathrm{softmax}(z + \mathbf{1}c) = \mathrm{softmax}(z)$

  gives numerically stable implementation

$$\mathrm{softmax}(z - \max_j z_j) = \mathrm{softmax}(z)$$

Softmax Cross-Entropy

- In Maximum Likelihood we have

$$-\log p(y|z) = \sum_{i=1}^{K} y_i \log \mathrm{softmax}(z)_i$$

$$\log \mathrm{softmax}(z)_i = z_i - \log \sum_j \exp(z_j)$$

- Recall the smoothed max, then we can write

$$\log \mathrm{softmax}(z)_i \simeq z_i - \max_j z_j$$

- Maximization, with $i = \arg\max_j z_j$, yields

$$\mathrm{softmax}(z)_i = 1 \quad \text{and} \quad \mathrm{softmax}(z)_{j \neq i} = 0$$

Softmax Units

- Softmax is an extension to the logistic sigmoid where we have 2 variables and $z_1 = 0, z_2 = z$

$$p(y = 1|x) = \mathrm{softmax}(z)_1 = \sigma(z_2)$$

- Softmax is a winner-take-all formulation

- Softmax is more related to the arg max function than the max function

Hidden Units

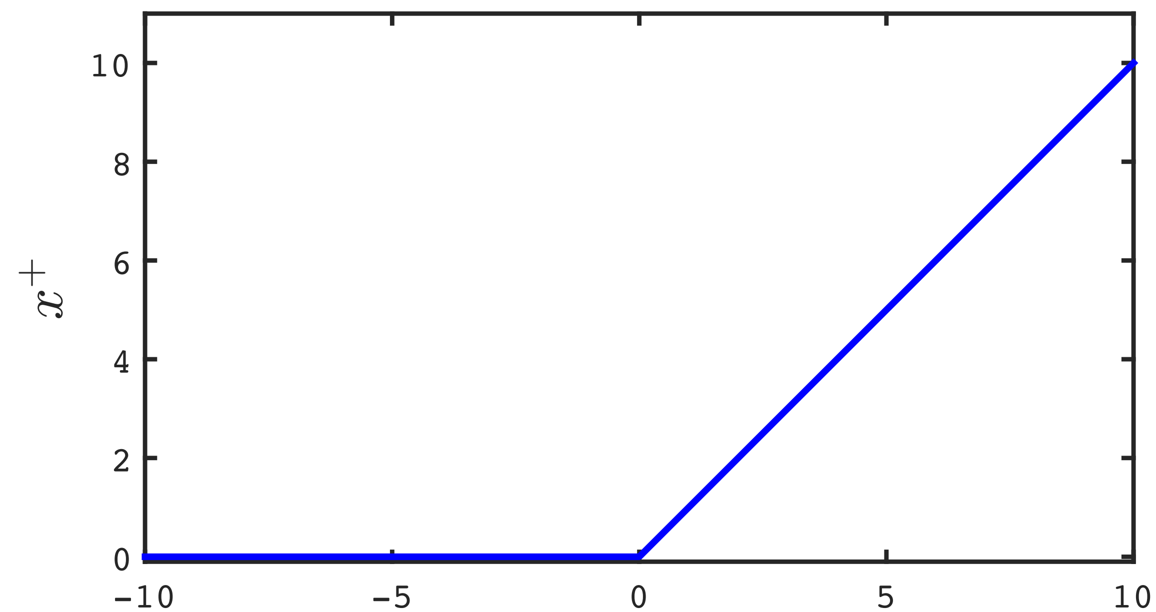
- The design of a neural network is so far still an art
- The basic principle is the **trial and error** process:
  1. Start from a known model
  2. Modify
  3. Implement and test (go back to 2. if needed)
- A good choice is to always use ReLUs
- In general the hidden unit picks a g for

$$h(x) = g(W^\top x + b)$$

Rectified Linear Units

- ReLUs typically use also an affine transformation

$$g(z) = \max\{0, z\}$$



- Good initialization is b = 0.1 (initially, a linear layer)

- Negative axis cannot learn due to null gradient

- Generalizations help avoid the null gradient

Leaky ReLUs and More

- A generalisation of ReLU is

$$g(z, \alpha) = \max\{0, z\} + \alpha \min\{0, z\}$$

- To avoid a null gradient the following are in use

1. Absolute value rectification $\alpha = -1$
2. Leaky ReLU $\alpha = 0.01$
3. Parametric ReLU α learnable
4. Maxout Units

$$g(z)_i = \max_{j \in S_i} z_j$$

$$\cup_i S_i = [1, \ldots, m]$$

$$S_i \cap S_j = \varnothing \quad i \neq j$$

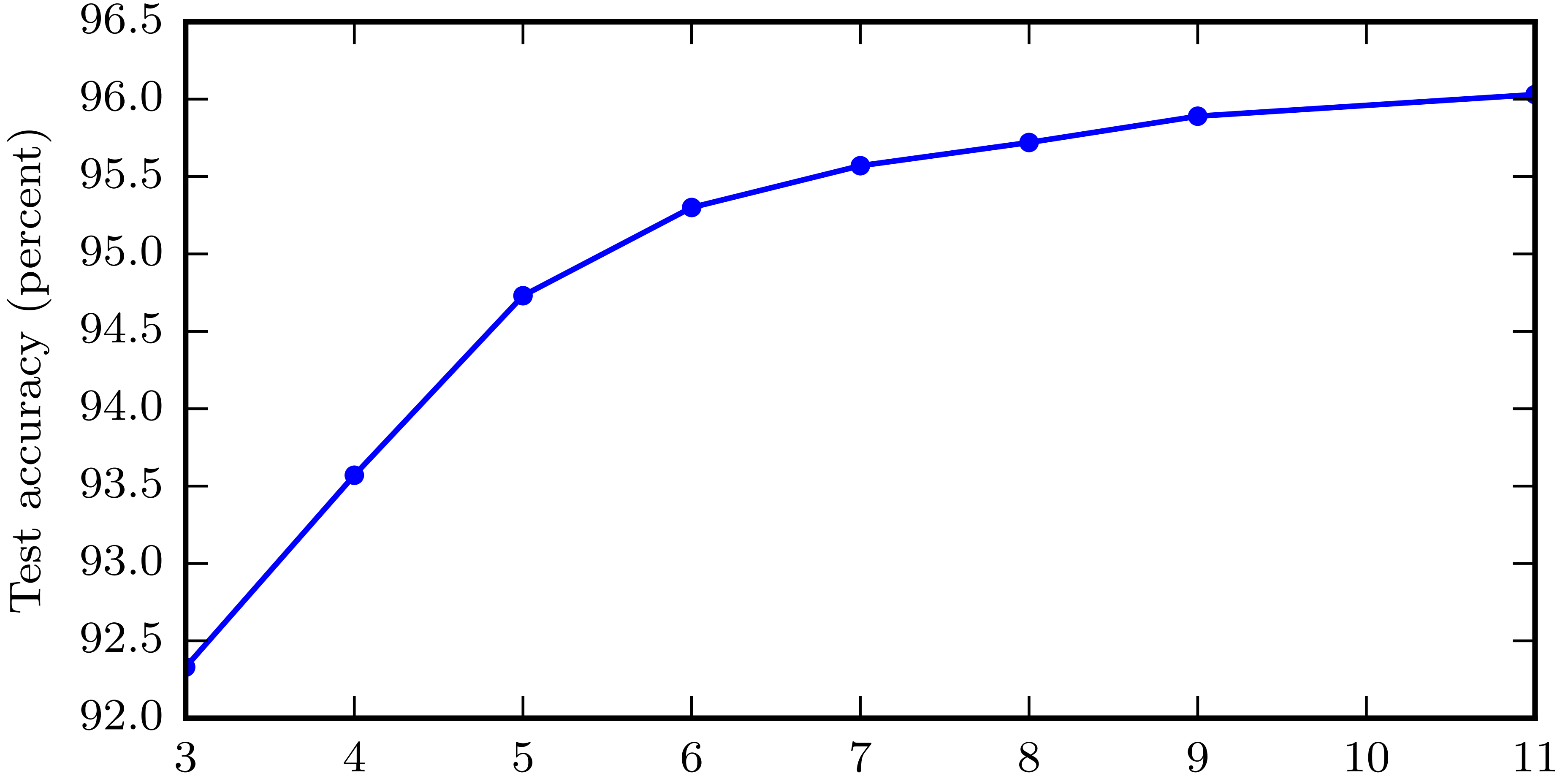
Network Design

- The **network architecture** is the overall structure of the network: number of units and their connectivity

- Today, the design for a task must be found experimentally via a careful analysis of the training and validation error

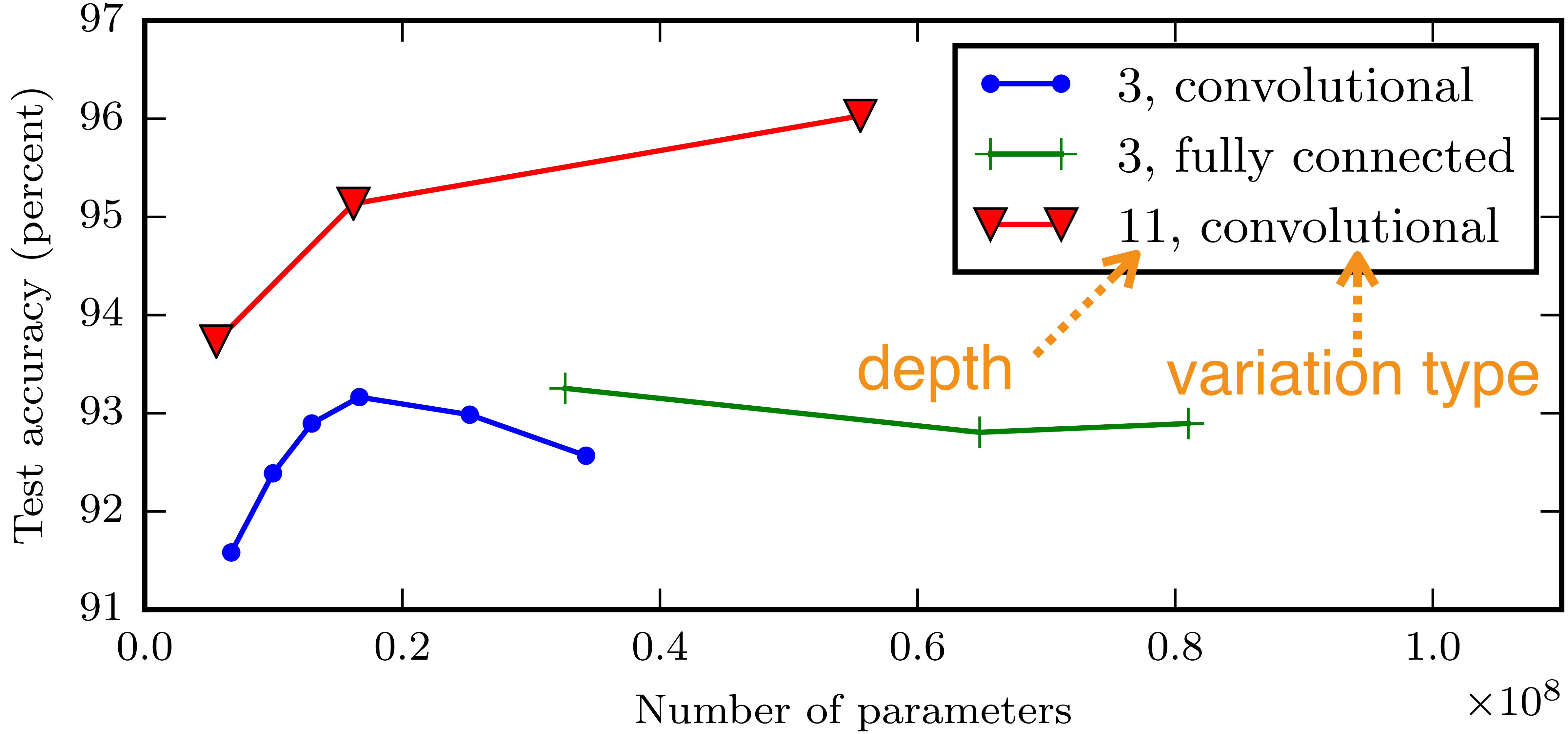
Depth

- A general rule is that depth helps generalization
- It is better to have many simple layers than few highly complex ones



Depth

- Other network modifications do not have the same effect

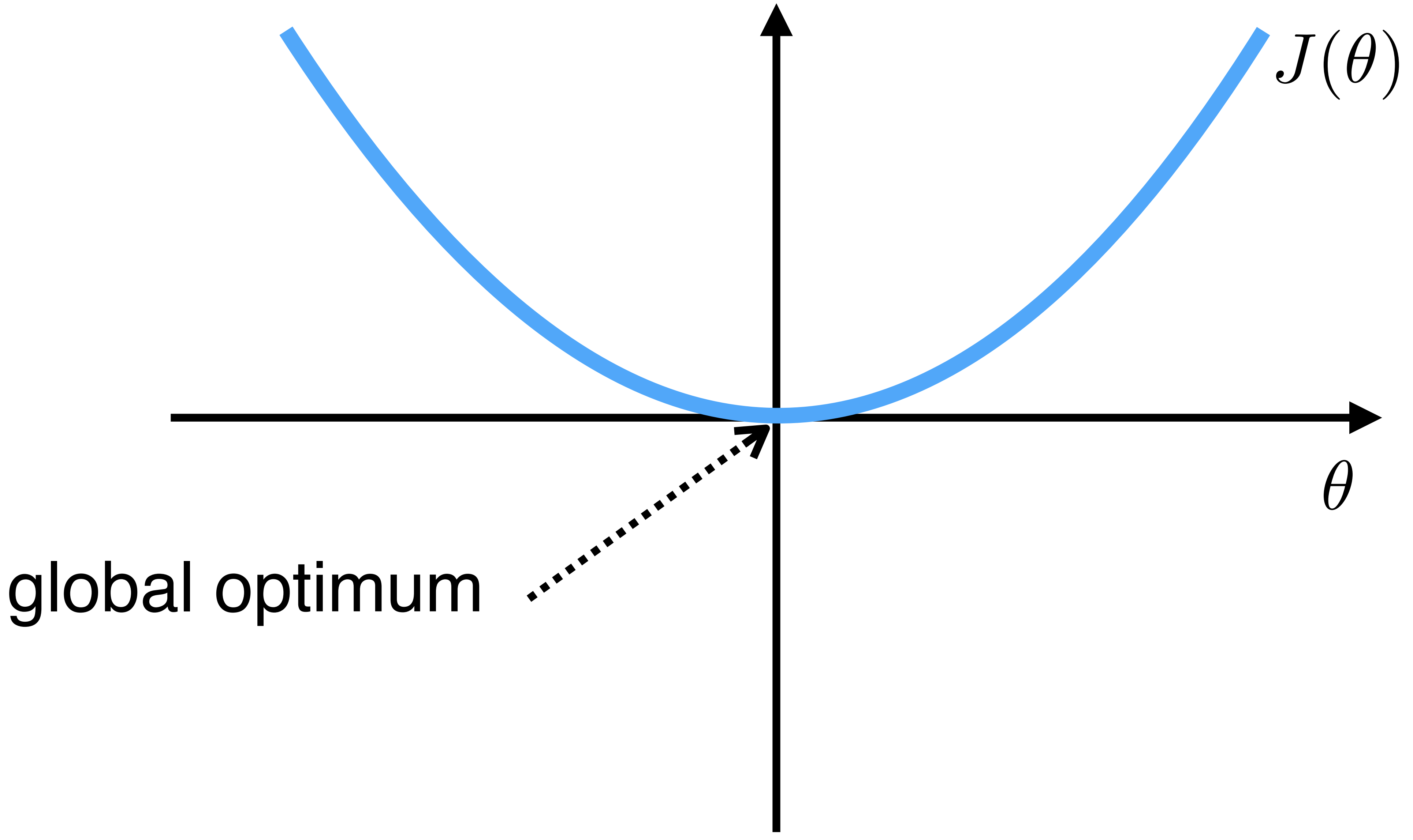


Optimization

- Given a task we define
  - The training data $\{x^i, y^i\}_{i=1,\ldots,m}$
  - A network design $f(x;\theta)$
  - The loss function $J(\theta) = \sum_{i=1}^{m} \text{loss}\left(y^i, f(x^i;\theta)\right)$
- Next, we **optimize** the network parameters θ
- This operation is called **training**

Optimization

- The MSE cost function $J(\theta)$ is **convex** with a linear model



Optimization

- However, since the cost function $J(\theta)$ is typically **non convex** in the parameters, we use an iterative solution

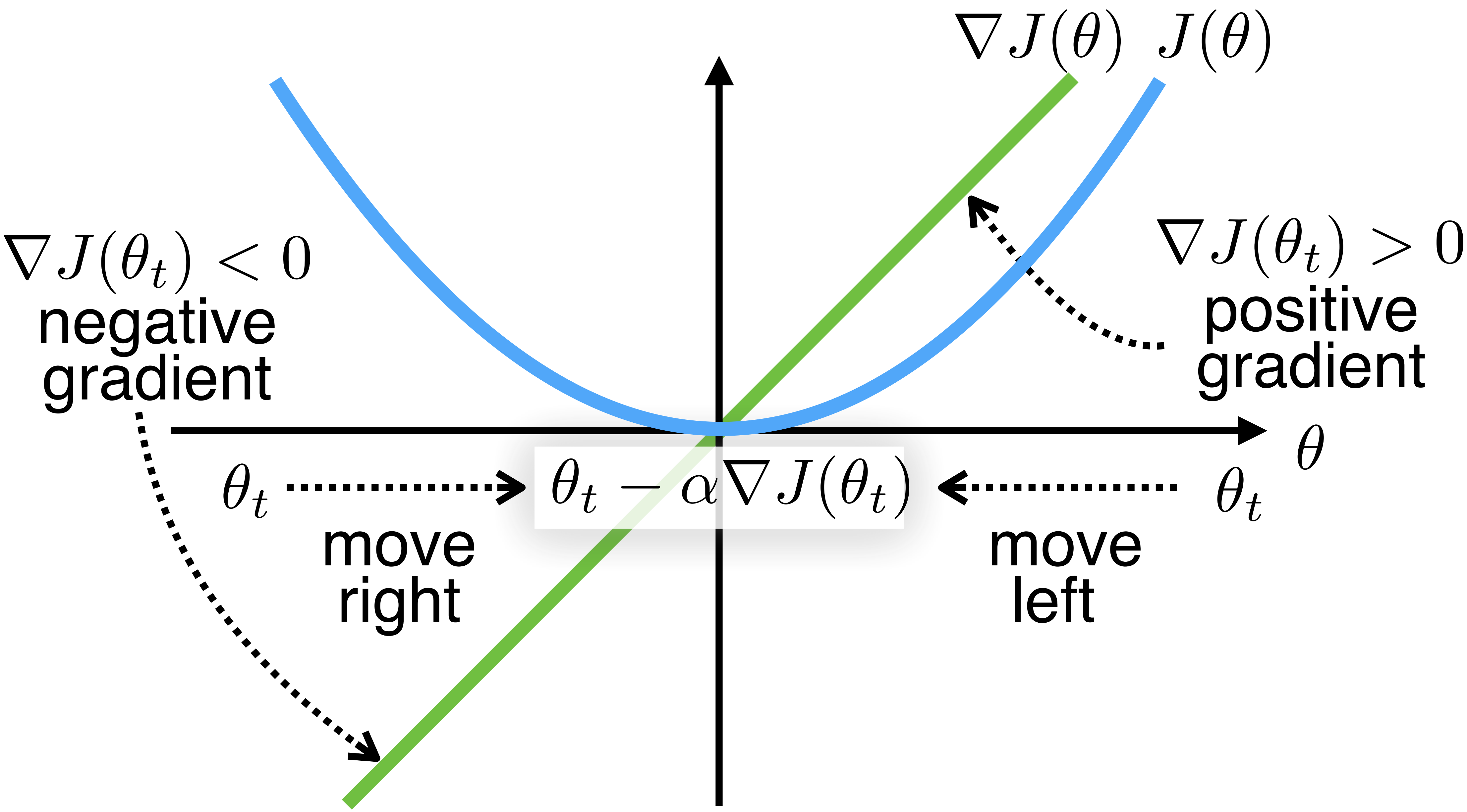
- We consider the **gradient descent** method

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

where $\alpha > 0$ is the learning rate

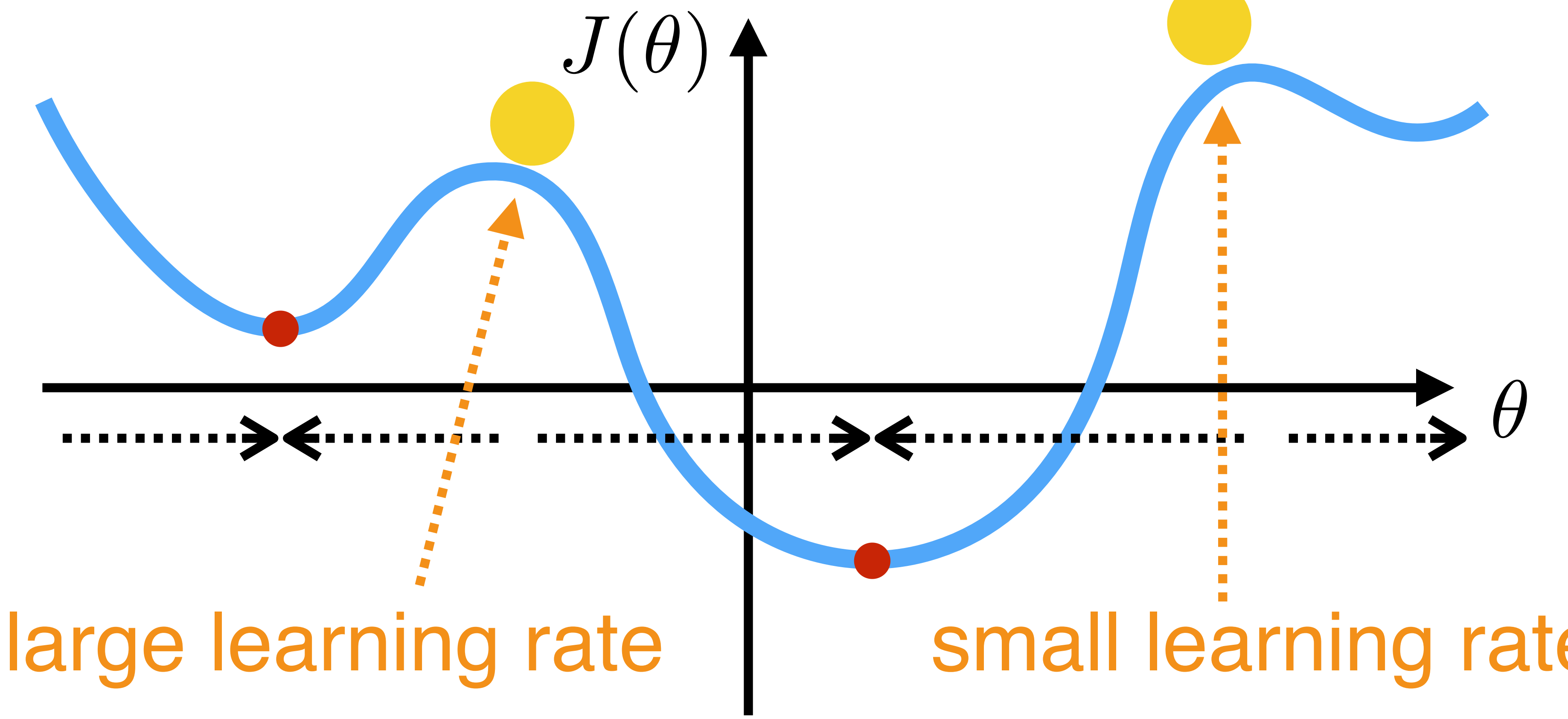
Optimization

gradient descent $\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$



Local Minima

- Does gradient descent reach a (local) minimum even with a non convex function?



$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

Optimization

- For more efficiency, we use the **stochastic gradient descent** method

- The gradient of the loss function is computed on a small set of samples from the training set

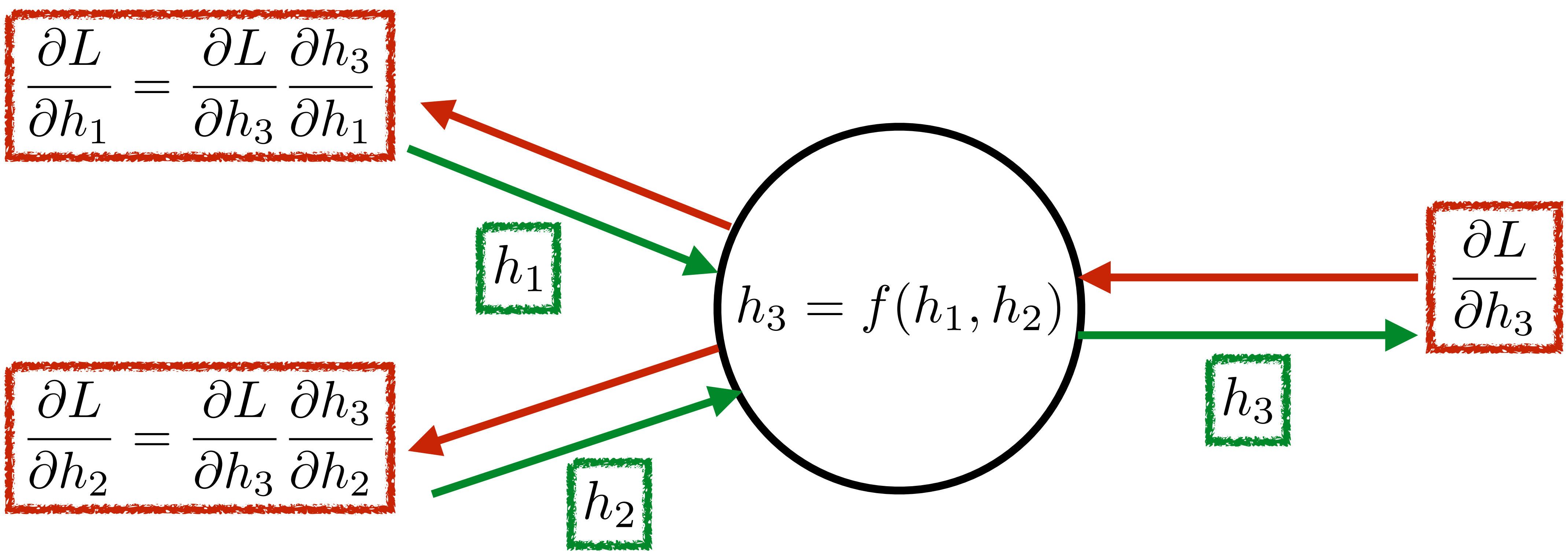
$$\tilde{J}(\theta) = \sum_{i \sim [1,\ldots,m]} \operatorname{loss}\left(y^i, f\left(x^i; \theta\right)\right)$$

and the iteration is as before

$$\theta_{t+1} = \theta_t - \alpha \nabla \tilde{J}(\theta_t)$$

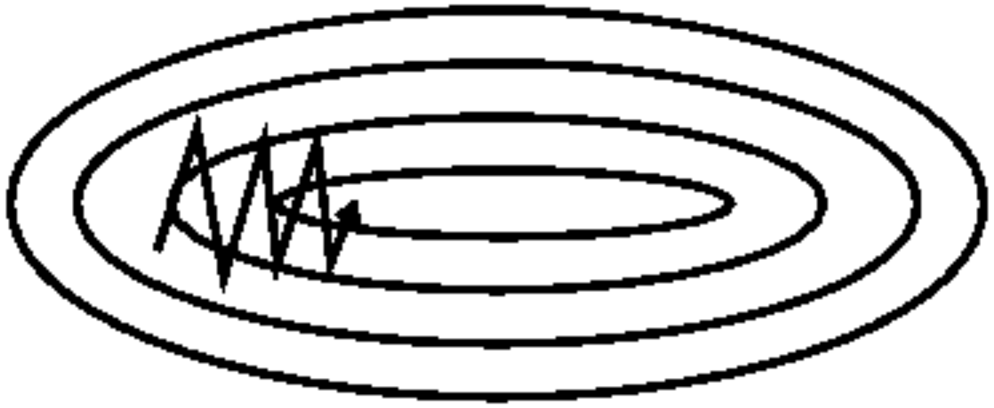
Backpropagation Algorithm

- An efficient implementation of the chain-rule to compute derivatives w.r.t. the network weights
- It is applied automatically by all popular frameworks

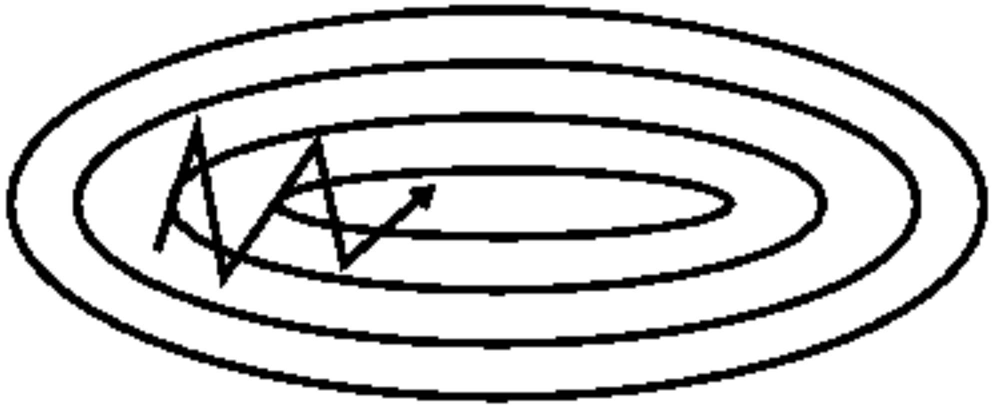


Momentum

- Standard gradient descent is often very slow
- Accelerated gradient methods are therefore the standard in practice
- The most basic is Gradient Descent + Momentum



without momentum



with momentum

Gradient Descent + Momentum

- The update rule for Gradient Descent with Momentum is:

$$\mathbf{m}_{t+1} \leftarrow \beta \mathbf{m}_t - \alpha \nabla_\theta J\left(\theta_t\right)$$
$$\theta_{t+1} \leftarrow \theta_t + \mathbf{m}_{t+1}$$

- The hyper-parameter β controls the amount of momentum

- $\beta = 0.9$ is often a good choice

Conclusion

- Introduced the basic building blocks of Neural Networks:
  - Cost Function & Output Units:
    - Sigmoid + binary cross-entropy
    - Softmax + categorical cross-entropy
    - Linear + MSE
  - Network Design:
    - Typically ReLU variants in the hidden layers
    - Deeper often generalizes better
    - Don’t reinvent the wheel
  - Optimization:
    - Gradient Descent (+ Momentum)

Thank you for your attention!

Questions?