Medical Diagnostics

Tutorial 2: Deep Neural Networks

Spring 2025

Faculty of Biotechnology and Food Engineering
Technion Israel Institute of Technology

Table of contents

1 Intro

1.1 Ongoing security situation


התראות חירום
HW1 OUT NOW

1.2 Course updates

- new HW1 due date is July 10th
  - 10% of final grade
  - Use the **forum**, read existing posts
- HW2+Final project merged
  - 25% of final grade
  - Released on July 10th
  - Due date: September 7th
- Workshops (HW) are on Monday (15:30-17:30) and Wednesday (10:00-11:30)
- Office hours (course material) can be scheduled through mail

1.3 Slido

Open app.sli.do on your browser

Enter code **00660121**

Answer and ask questions in real-time

Example: How is Homework 1 going?

2 Tutorial 1 recap

2.1 Basic notations of ML

Symbol	Symbol
$\mathcal{X}$	$h_{\theta^*} \approx f$
$\mathcal{Y}$	$\mathcal{L}(h_\theta)$
$D = \{(x_i, y_i)\}_{i=1}^N$	θ
$f : \mathcal{X} \to \mathcal{Y}$	η
$\mathcal{A}$	$\nabla_\theta \mathcal{L}(h_\theta)$
$\mathcal{H}$	

Model training workflow:

1. Prepare data D
2. Choose model $\mathcal{H}$
3. **Train model:** Find h_{θ^*} s.t. $\mathcal{L}(h_{\theta^*}) = \min_{h_\theta \in \mathcal{H}} \mathcal{L}(h_\theta)$
    - Loss function
    - Optimization algorithm
4. **Tune hyperparameters:** Use validation set
5. **Evaluate:** Assess performance on test set
6. **Deploy:** Use trained model (h_θ) for predictions

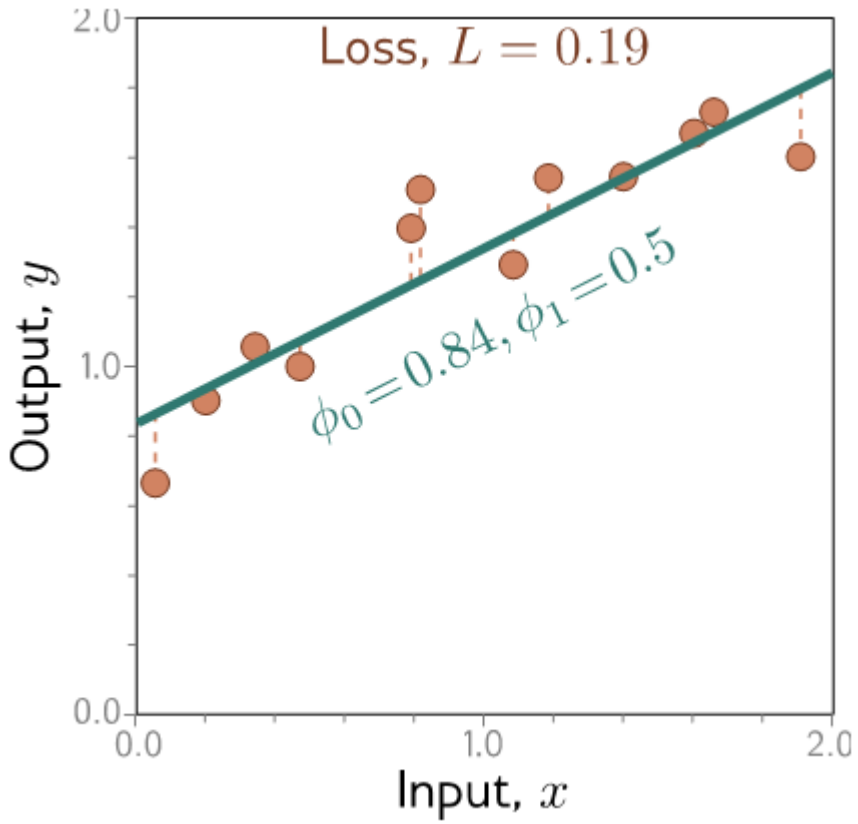
2.2 Gradient Descent (GD)

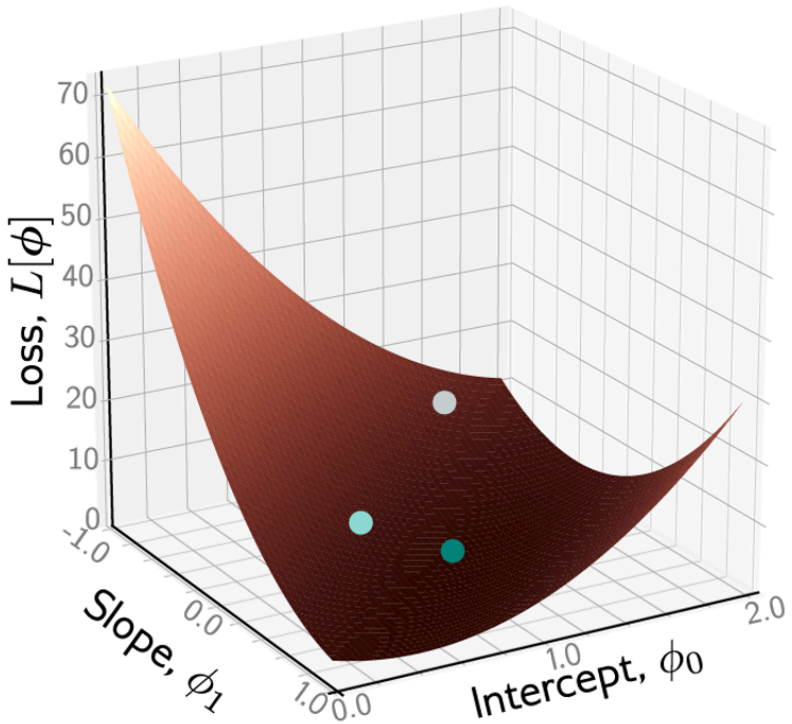
1. Initialize θ
2. Repeat:
   - Compute gradient $\nabla_\theta \mathcal{L}(\theta)$
   - Update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$
3. Until convergence

2.3 Linear Regression (LR)

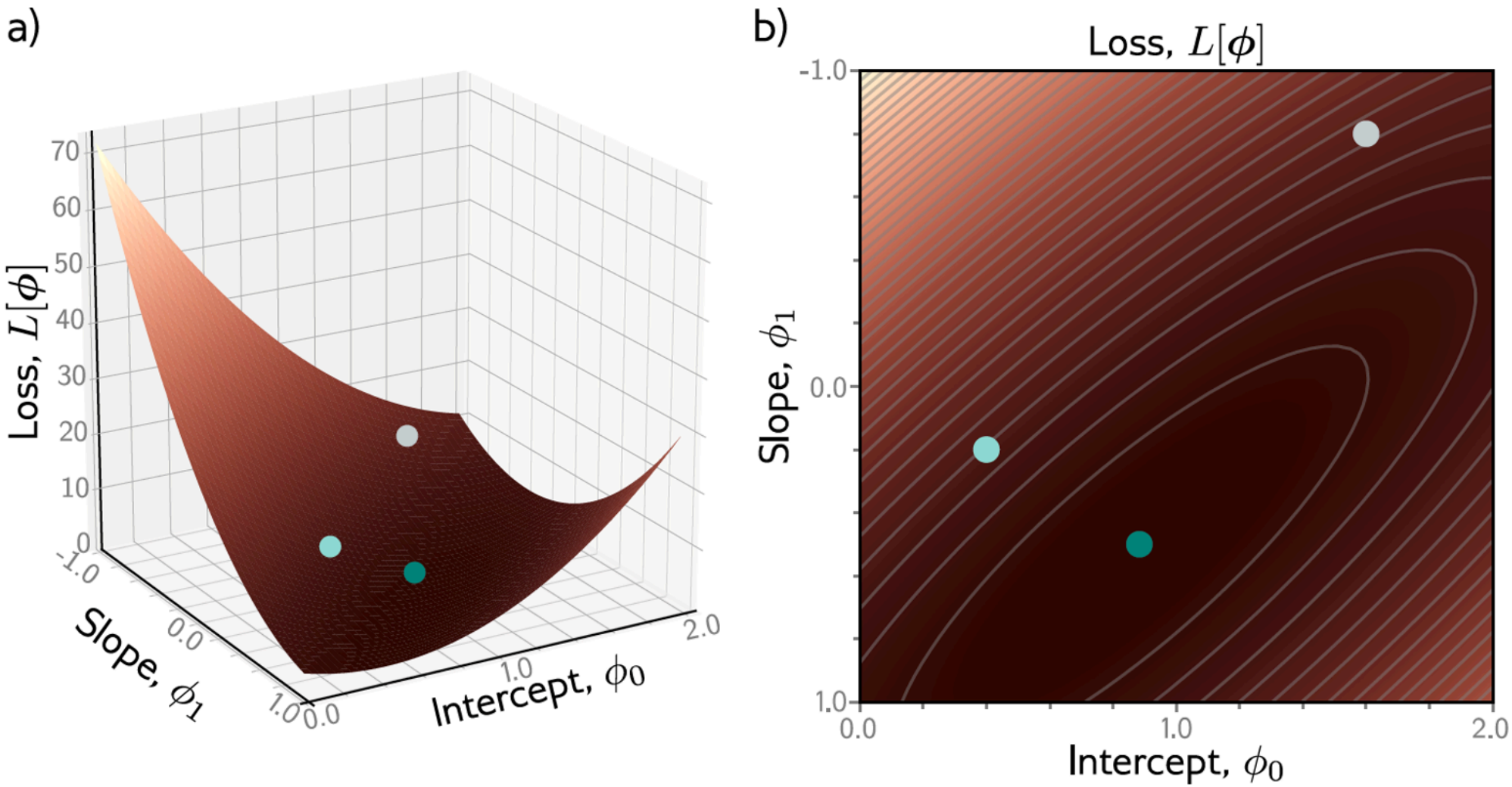
1. **Model:** $h_\theta(x) = \theta^T x$
2. **Loss function:** $\mathcal{L}(h_\theta) = \frac{1}{2} \sum_{i=1}^{N} (y_i - h_\theta(x_i))^2$
3. **Optimization:** Find θ^* that minimizes $\mathcal{L}(h_\theta)$
    - Gradient descent: Iteratively update θ using the gradient of the loss function

2.4 Example: 1D LR loss function

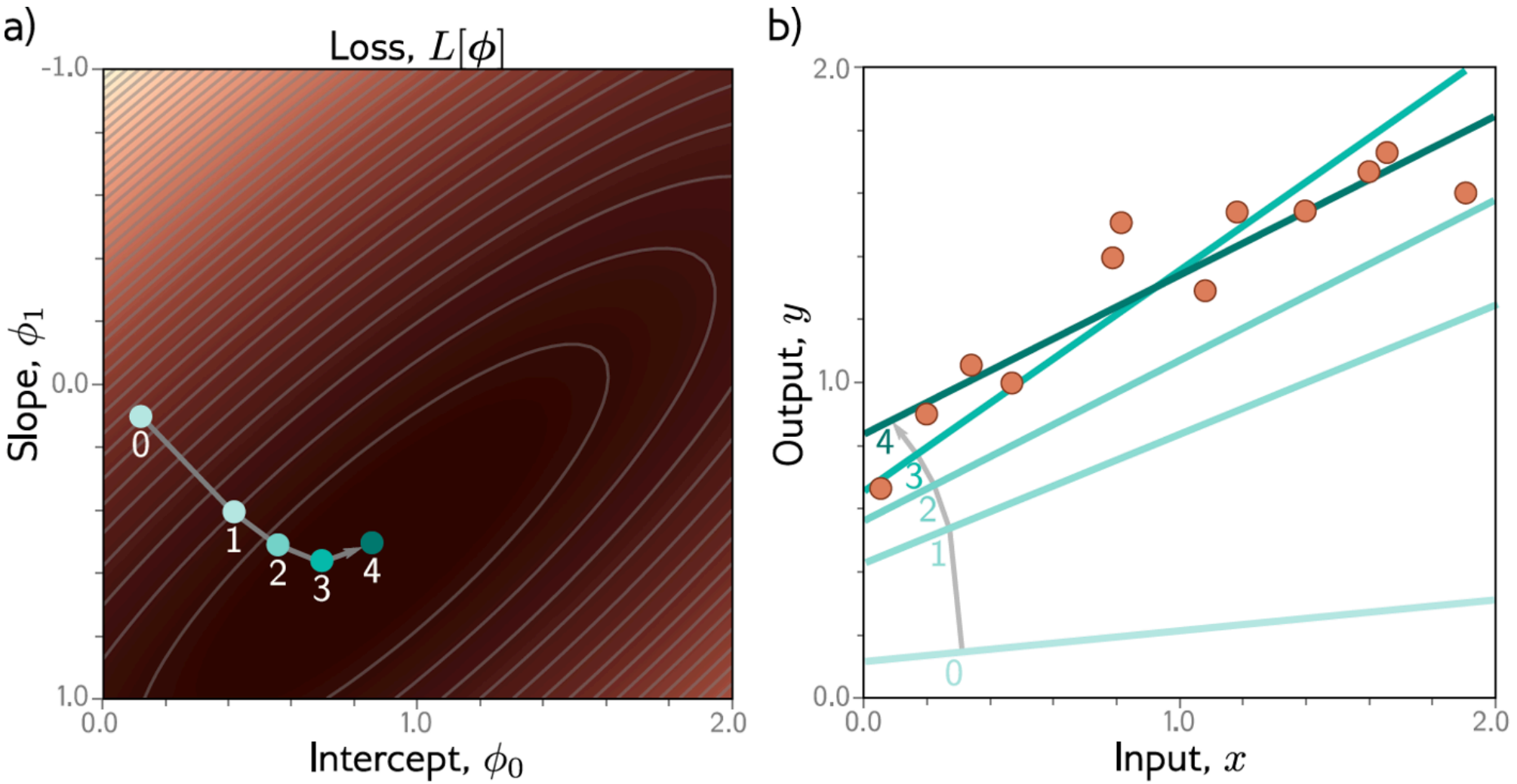




2.4 Example: 1D LR loss function

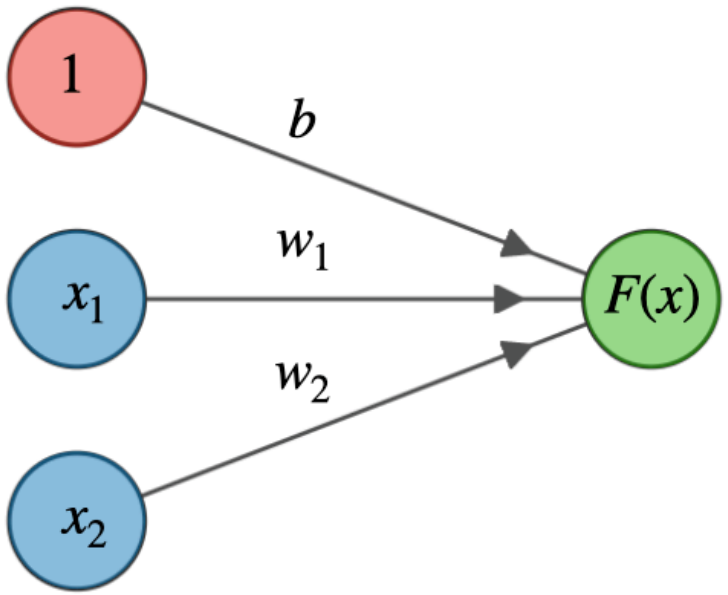


2.4 Example: 1D LR loss function



2.4 The perceptron: 1D LR as a shallow network

Linear regression



$$F(x) = w^T x + b$$

Input Layer $\in \mathbb{R}^3$ Output Layer $\in \mathbb{R}^1$

$$F(x) = w_1 \cdot x_1 + w_2 \cdot x_2 + 1 \cdot b$$

2.5 Possible problems with linear models

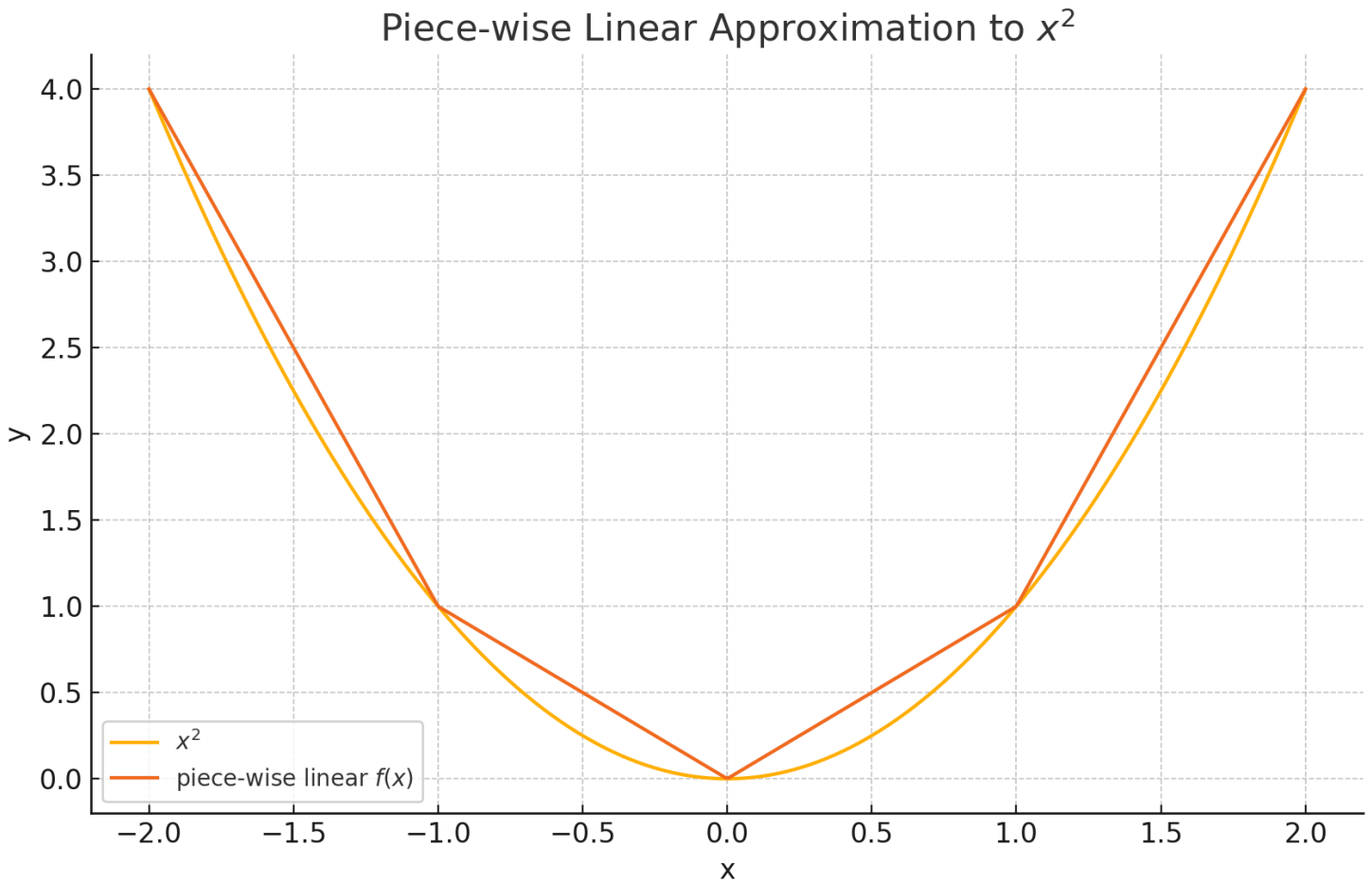
1. Want to be able to describe non-linear relationships
2. Want multiple inputs
3. Want multiple outputs

3 Perceptron → Multi-Layered Perceptron (MLP)

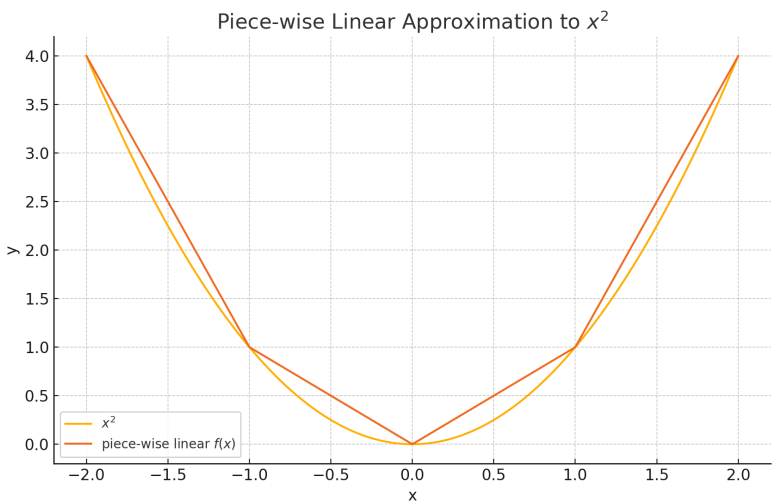
Graphics from 236781 Deep learning course at the Technion (Dr. Haim Baskin)

3.1 Piecewise linear functions

Consider the following function we would like to learn:



3.2 Piecewise linear functions



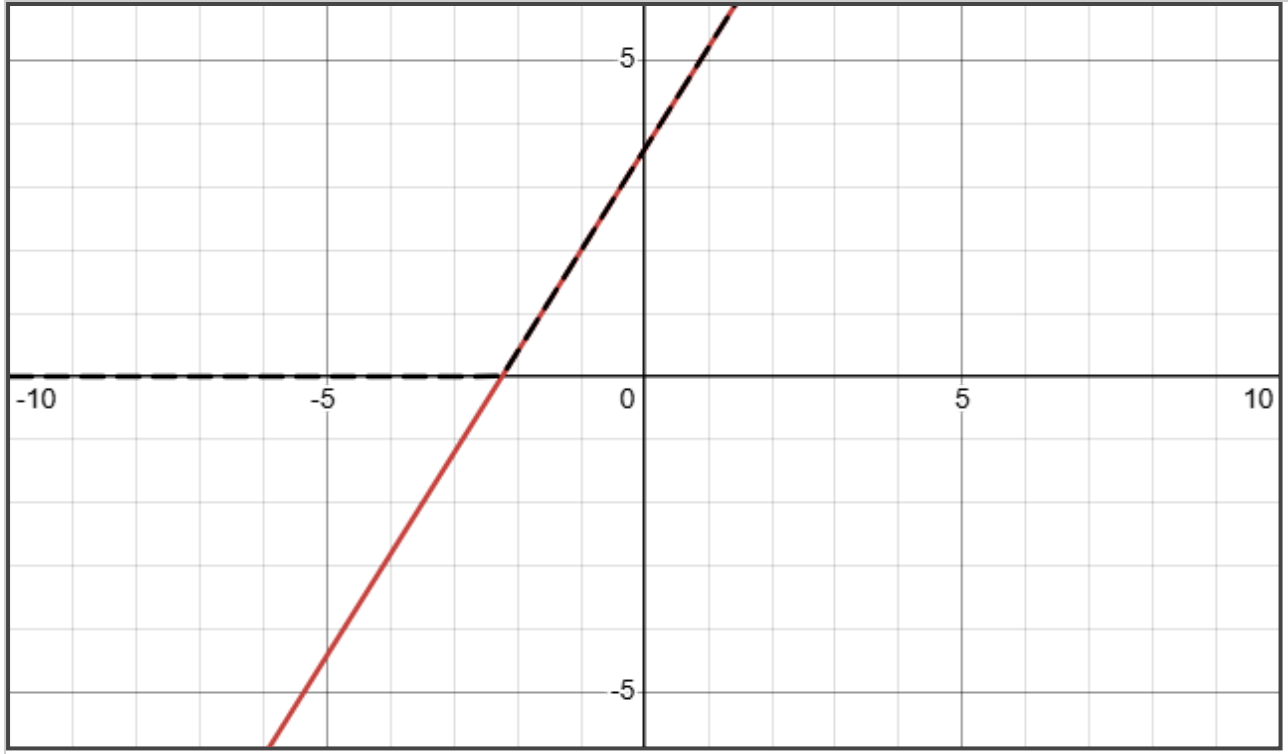
$$f(x) = \begin{cases} -3x - 2, & -2 \leq x \leq -1 \\ -x, & -1 < x \leq 0 \\ x, & 0 < x \leq 1 \\ 3x - 2, & 1 < x \leq 2 \end{cases}$$

- Combinatorical
  - Number of “joints” in the function: 4
  - Location of joints: $-1, 0, 1$
- Doesnt scale well to higher dimensions
- Differentiablity?

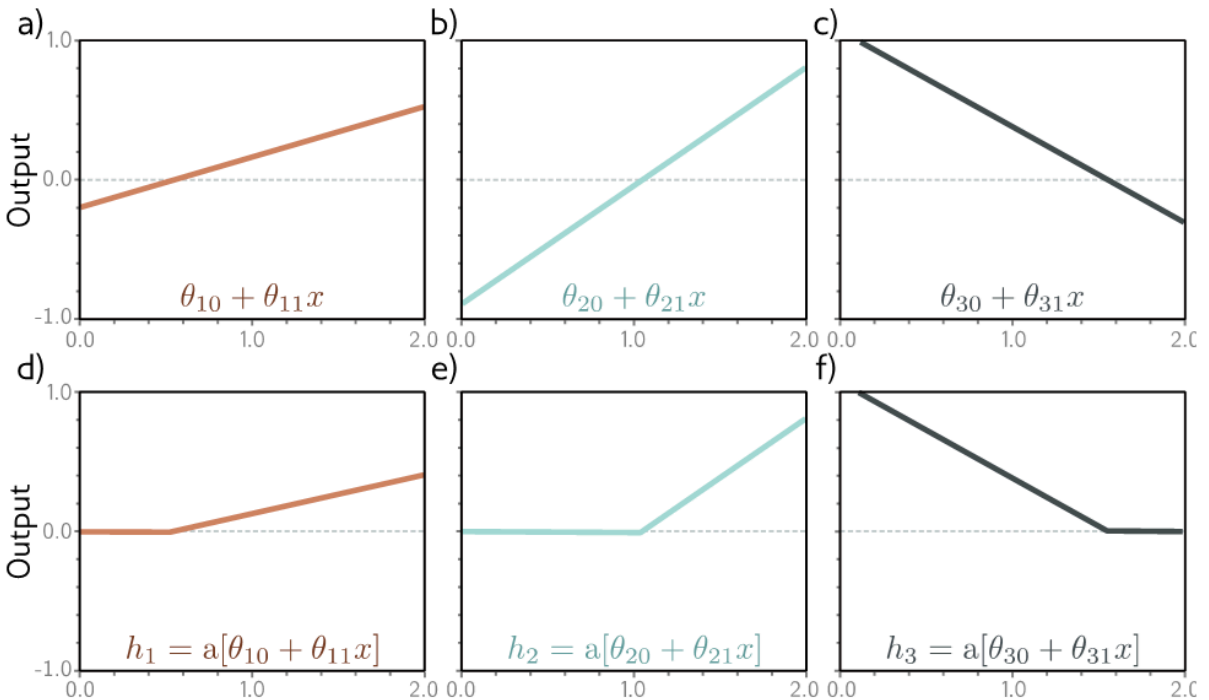
3.3 Activation functions: learnable joints!

$\mathrm{ReLU} = \max(0, f(x))$ (Rectified Linear Unit)

$$\mathrm{ReLU}(f(x)) = \begin{cases} 0, & f(x) < 0 \\ f(x), & f(x) \geq 0 \end{cases}$$

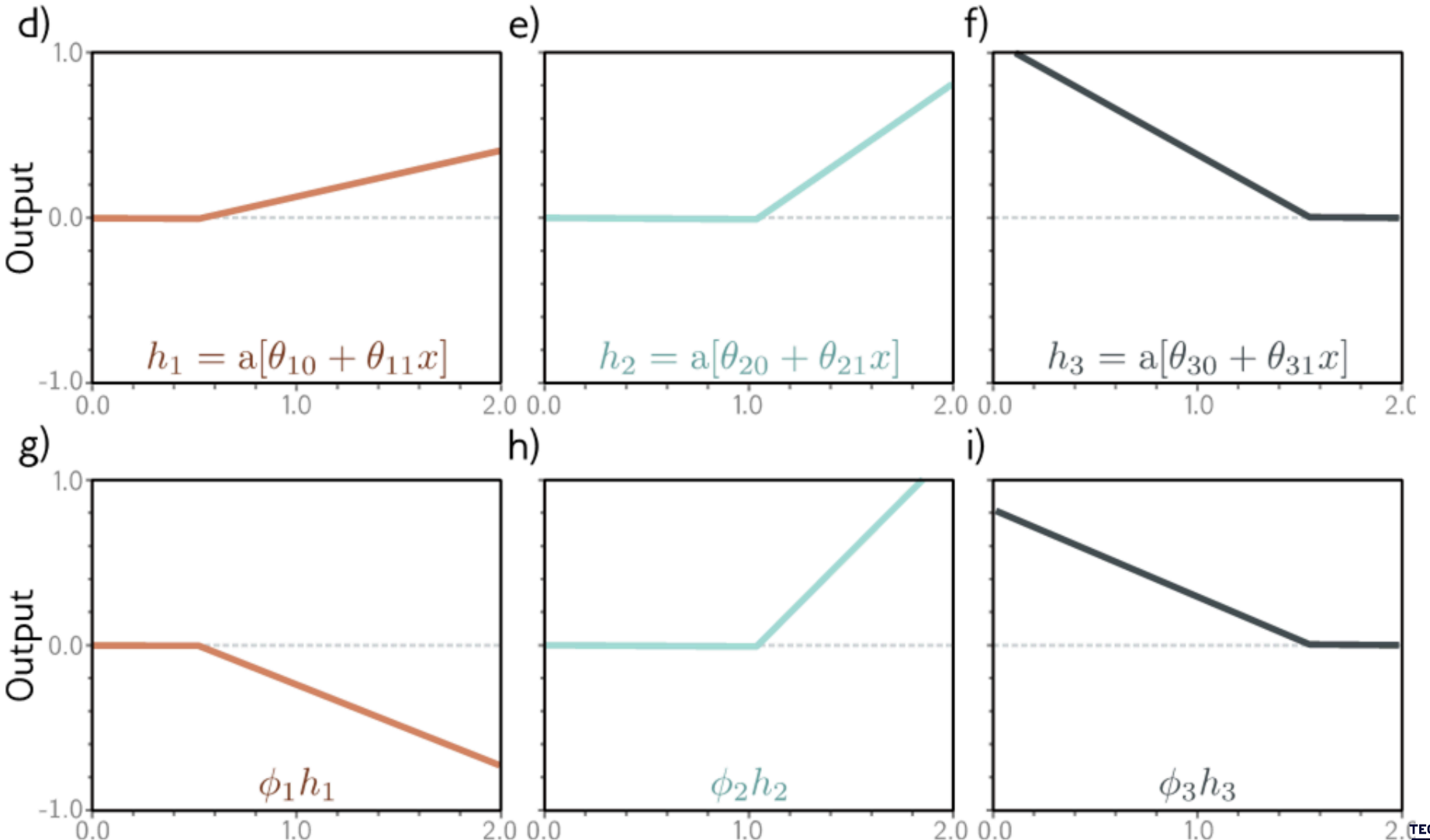

5
-10
-5
0
5
10
-5

3.4 Pass through ReLU functions: hidden units

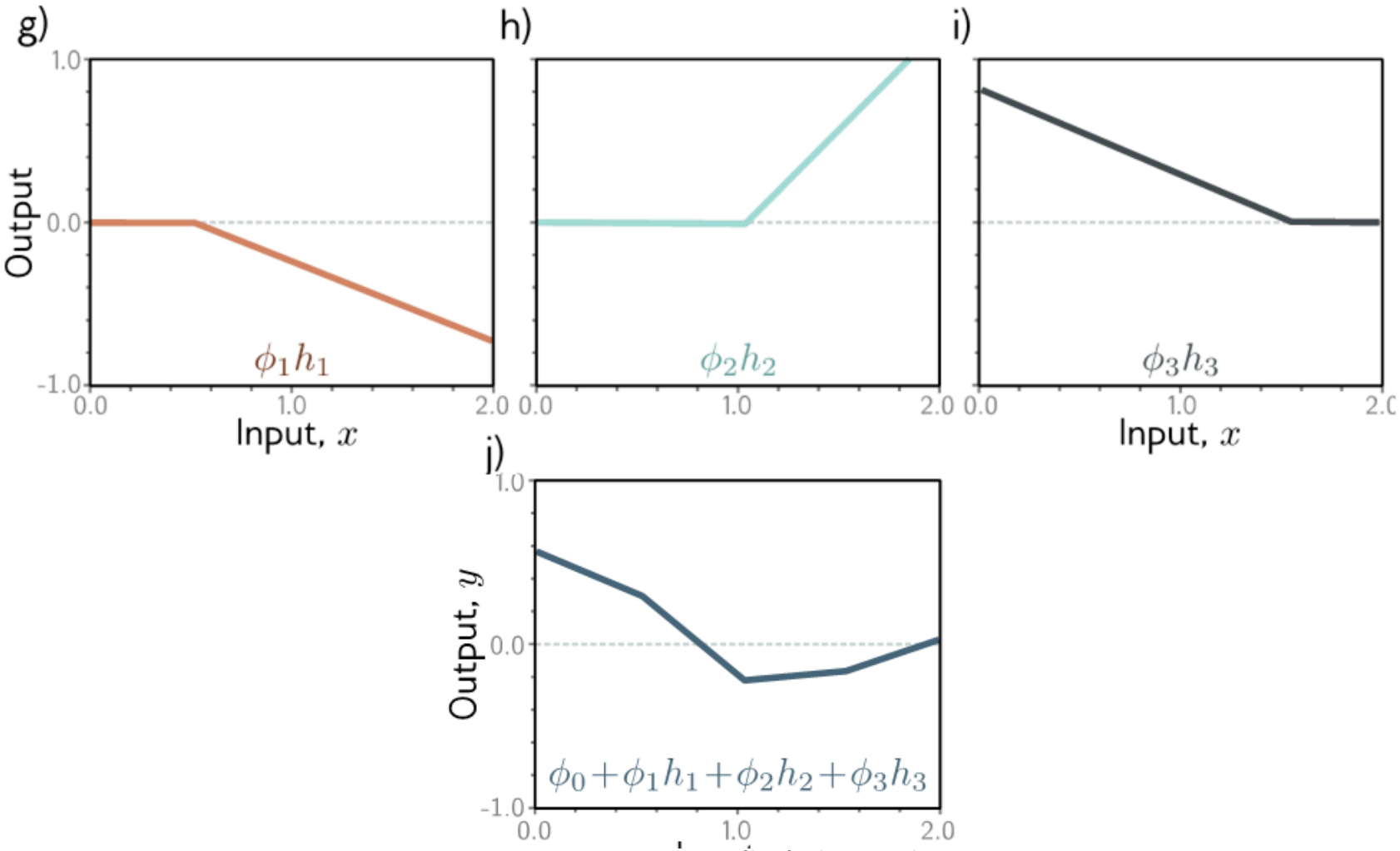


- Other activation functions: sigmoid, tanh, softmax, etc.

3.5 Weight the hidden units

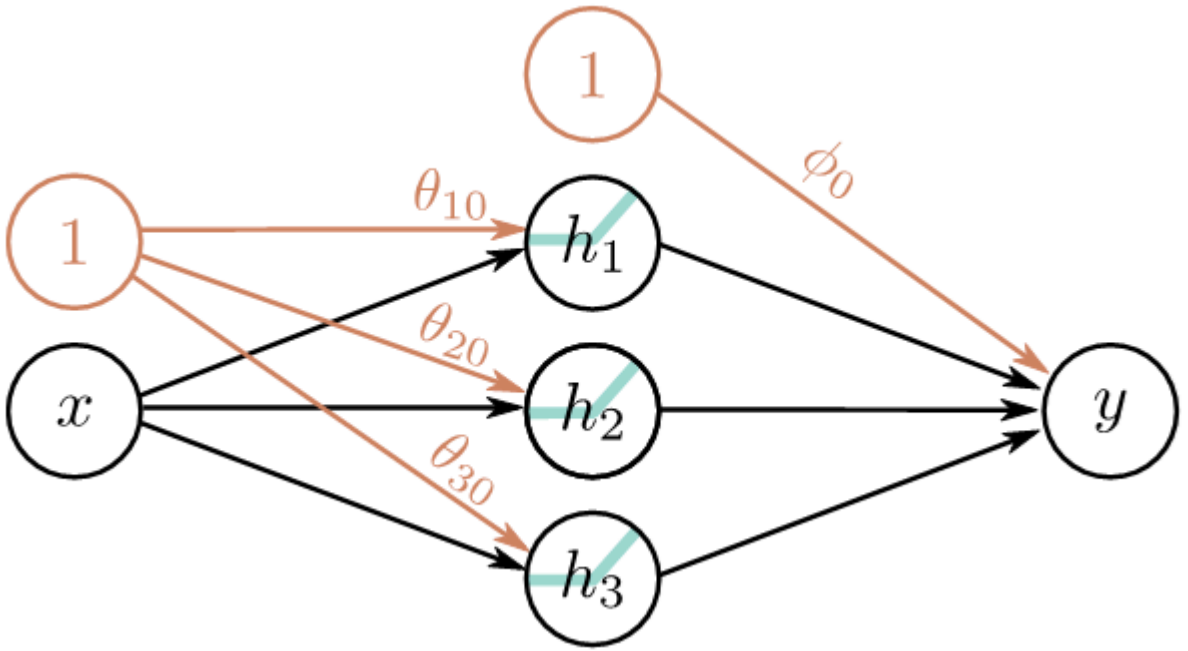


3.6 Sum the weighted hidden units to output



$$
\begin{aligned}
y = \phi_0 &+ \phi_1 \operatorname{ReLU}\left(\theta_{10} + \theta_{11} x\right) \\
&+ \phi_2 \operatorname{ReLU}\left(\theta_{20} + \theta_{21} x\right) \\
&+ \phi_3 \operatorname{ReLU}\left(\theta_{30} + \theta_{31} x\right)
\end{aligned}
$$

3.7 Lets draw the network again



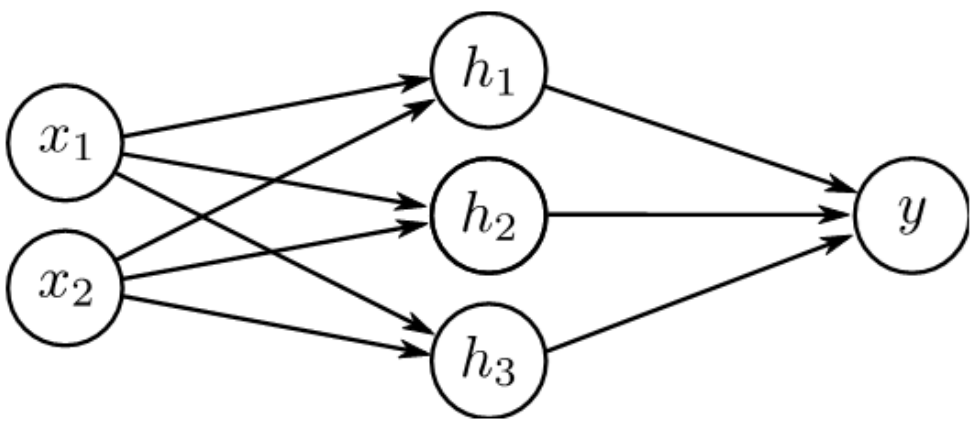
$$y = \phi_0 + \phi_1 \,\mathrm{ReLU}\left(\theta_{10} + \theta_{11}x\right) + \phi_2 \,\mathrm{ReLU}\left(\theta_{20} + \theta_{21}x\right) + \phi_3 \,\mathrm{ReLU}\left(\theta_{30} + \theta_{31}x\right)$$

3.8 Multiple inputs

- 2 inputs, 3 hidden units, 1 output

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

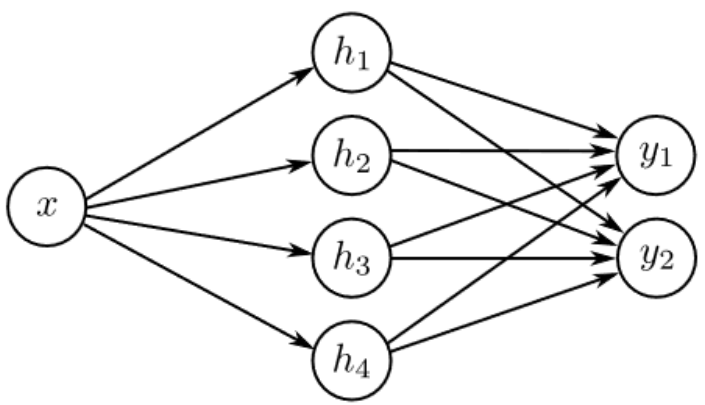


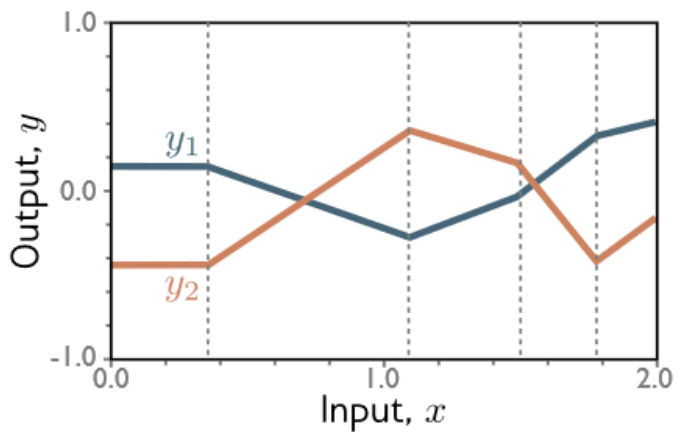
3.9 Multiple outputs

- 1 input, 4 hidden units, 2 outputs

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$
$$h_4 = \mathrm{a}[\theta_{40} + \theta_{41}x]$$

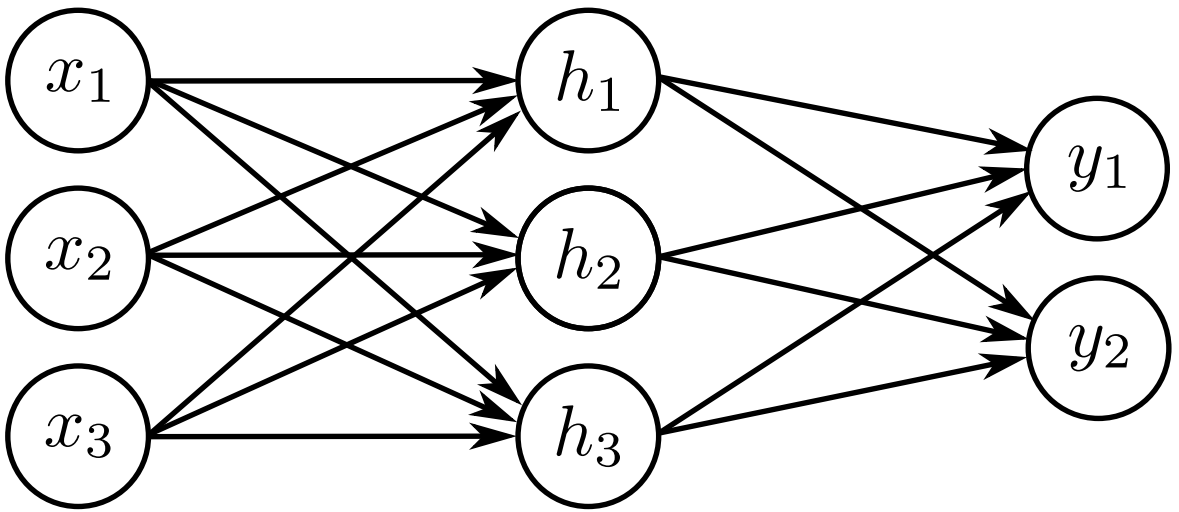
$$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4$$
$$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4$$



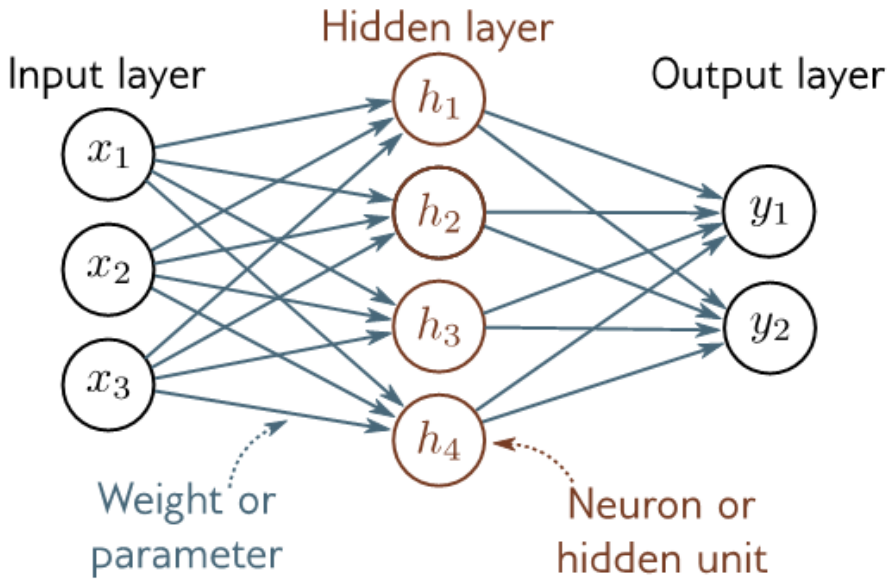


3.10 Question

- How many parameters does this model have?



3.11 Nomenclature

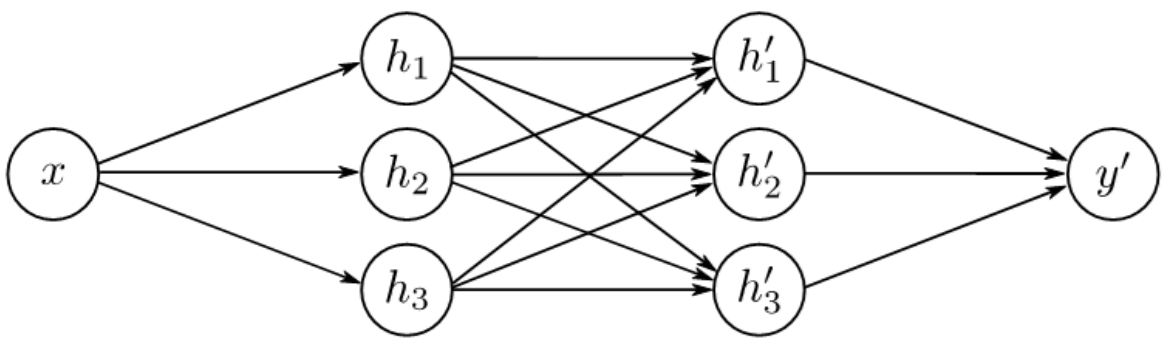


- Everything in one layer connected to everything in the next = fully connected network
- No loops = feedforward network
- Values after ReLU (activation functions) = activations
- One hidden layer = shallow neural network

3.12 MLP: Deep networks

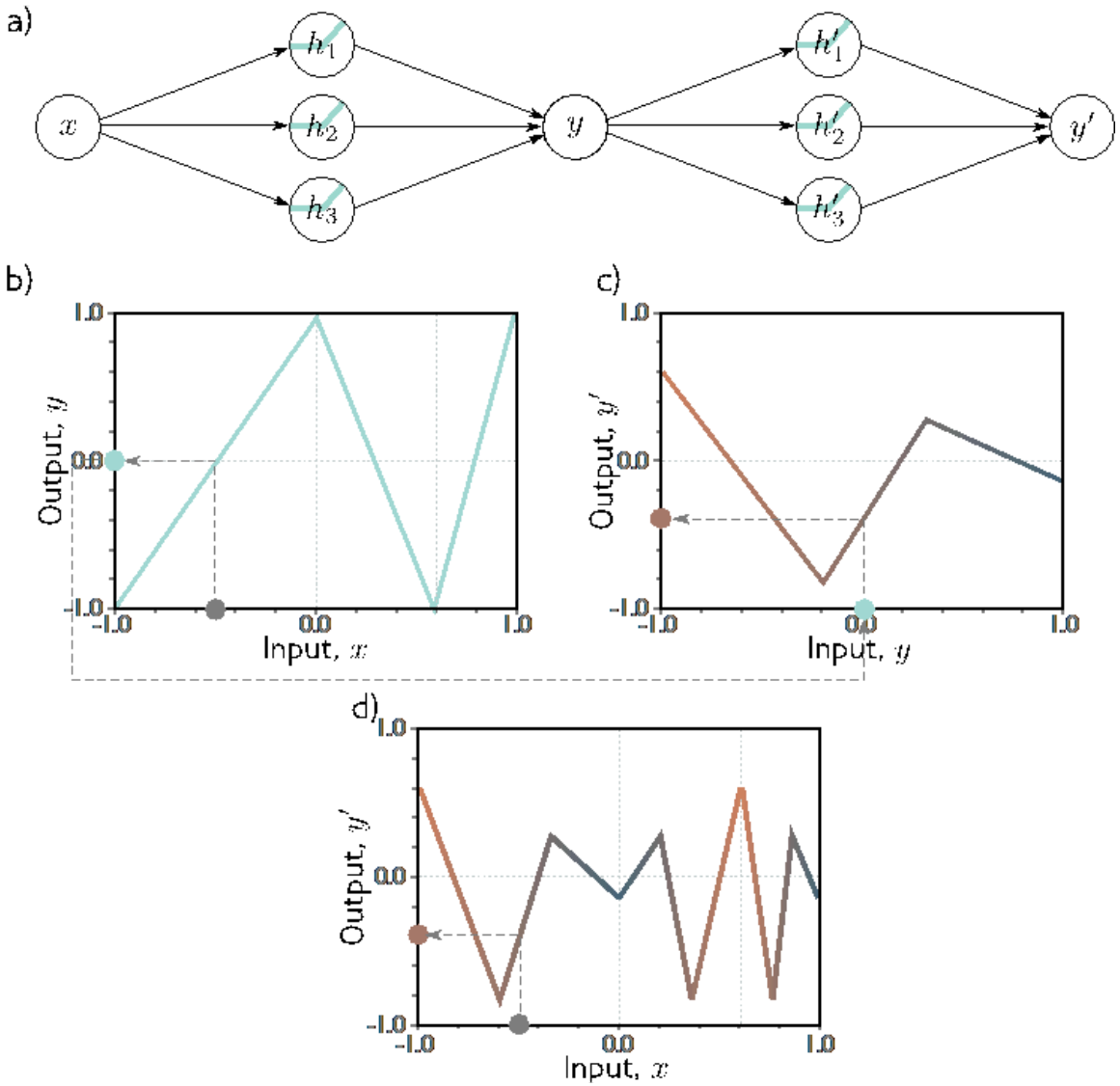
$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x] \qquad h'_1 = \mathrm{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x] \qquad h'_2 = \mathrm{a}[\psi_{20} + \psi_{21}h_2 + \psi_{22}h_2 + \psi_{23}h_3]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x] \qquad h'_3 = \mathrm{a}[\psi_{30} + \psi_{31}h_2 + \psi_{32}h_2 + \psi_{33}h_3]$$

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$

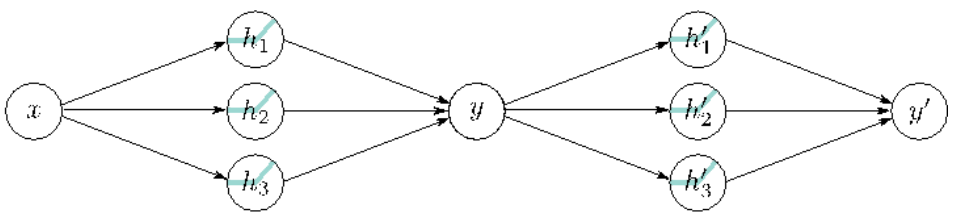


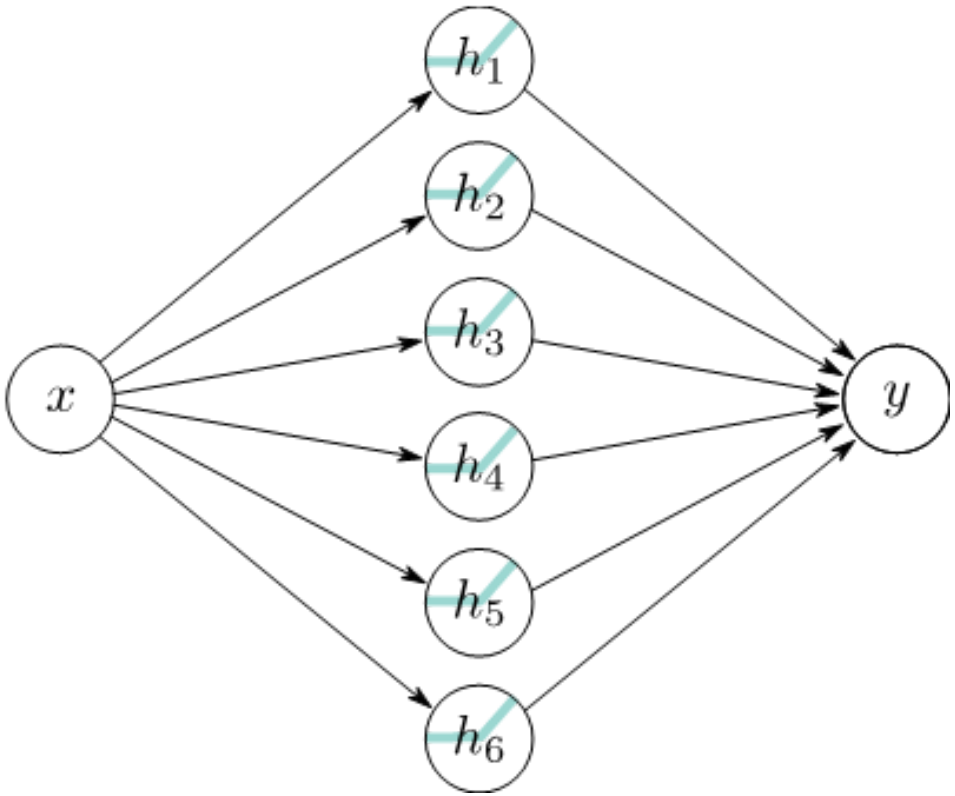
- Hyperparameter: number of hidden layers (Depth)
- Hyperparameter: number of hidden units per layer (Width)

3.13 Intuition: Folding



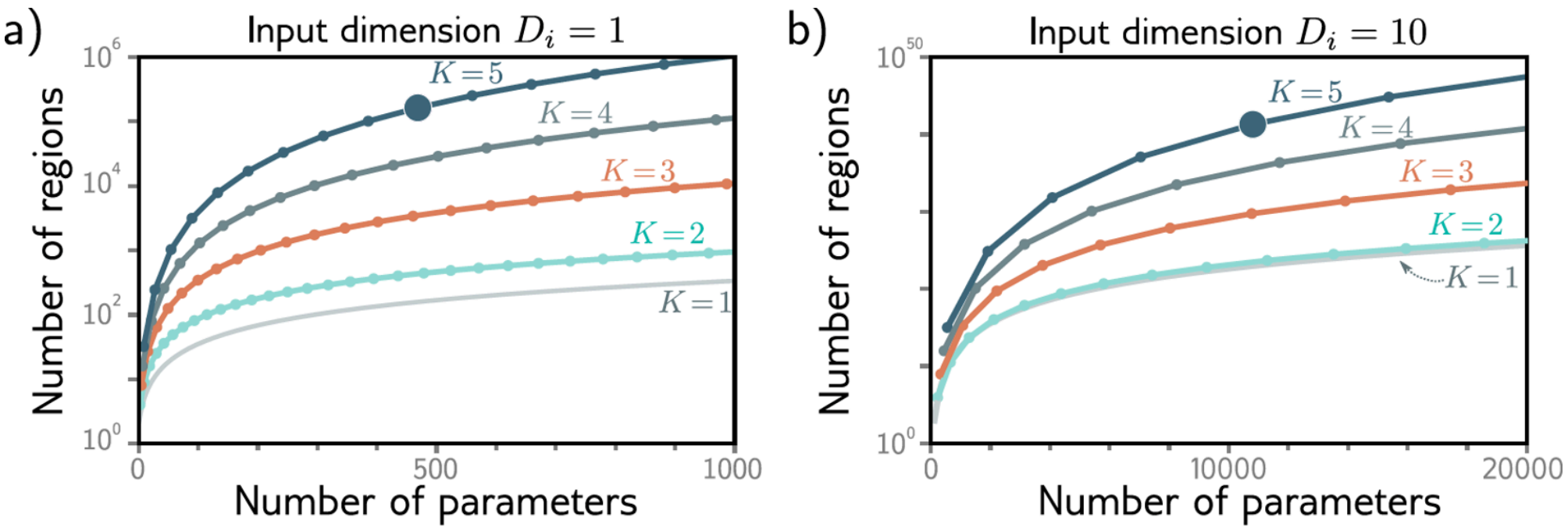
3.14 Shallow VS Deep





- Shallow: 19 parameters, max 7 regions
- Deep: 20 parameters, at least 9 regions

3.15 Shallow VS Deep viusalized



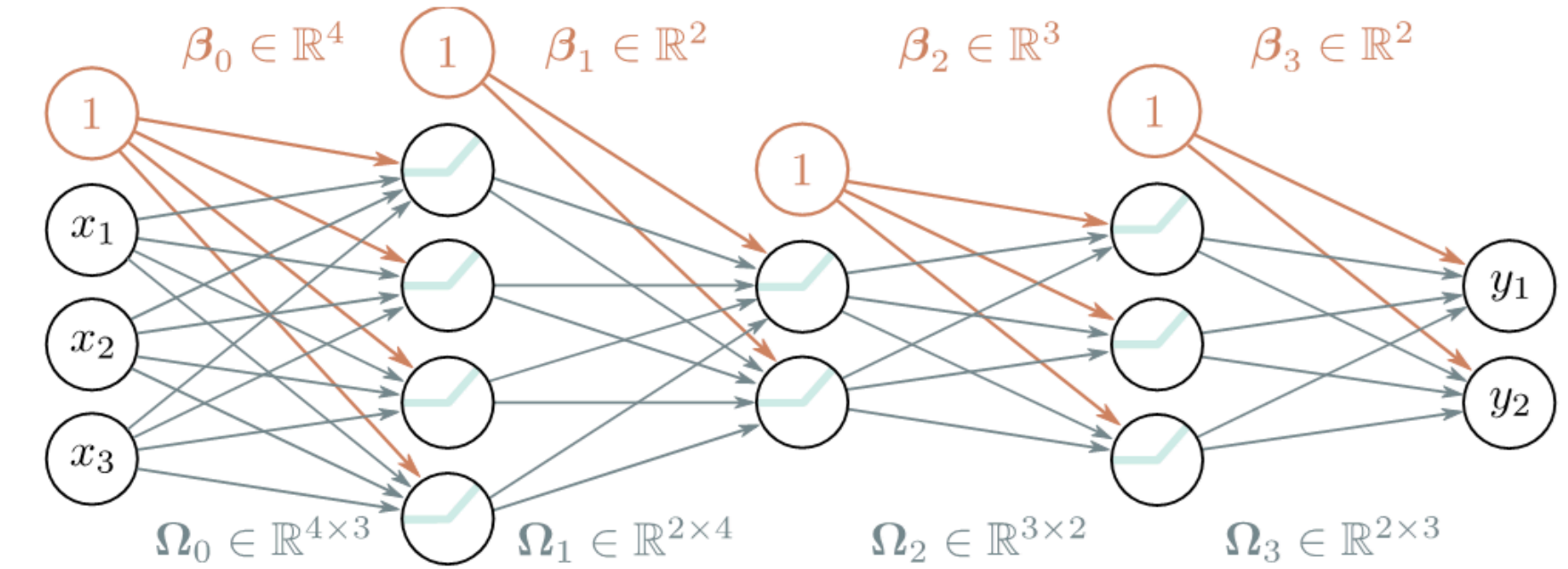
3.16 Notations

$$\mathbf{h} = \mathbf{a}\left(\begin{bmatrix}\theta_{10}\\ \theta_{20}\\ \theta_{30}\end{bmatrix} + \begin{bmatrix}\theta_{11}\\ \theta_{21}\\ \theta_{31}\end{bmatrix} x\right) = \mathbf{a}(\boldsymbol{\theta}_0 + \boldsymbol{\theta} x)$$

$$\mathbf{h}' = \mathbf{a}\left(\begin{bmatrix}\psi_{10}\\ \psi_{20}\\ \psi_{30}\end{bmatrix} + \begin{bmatrix}\psi_{11} & \psi_{12} & \psi_{13}\\ \psi_{21} & \psi_{22} & \psi_{23}\\ \psi_{31} & \psi_{32} & \psi_{33}\end{bmatrix}\begin{bmatrix}h_1\\ h_2\\ h_3\end{bmatrix}\right) = \mathbf{a}(\boldsymbol{\psi}_0 + \boldsymbol{\Psi}$$

$$y = \begin{bmatrix}\phi_1 & \phi_2 & \phi_3\end{bmatrix}\begin{bmatrix}h'_1\\ h'_2\\ h'_3\end{bmatrix} + \phi_0 = \boldsymbol{\phi}^\top \mathbf{h}' + \phi_0$$

3.17 Deep networks



Input, $\mathbf{x}$ — $D_i = 3$

Hidden layer, $\mathbf{h}_1$ — $D_1 = 4$

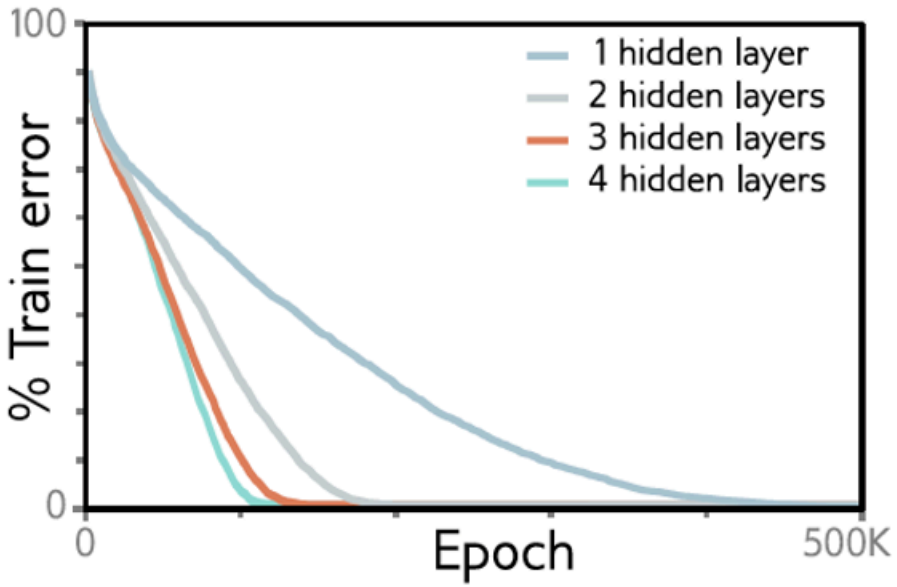
Hidden layer, $\mathbf{h}_2$ — $D_2 = 2$

Hidden layer, $\mathbf{h}_3$ — $D_3 = 3$

Output, $\mathbf{y}$ — $D_o = 2$

3.18 Deep networks

Figure 20.2 MNIST-1D training. Four fully connected networks were fit to 4000 MNIST-1D examples with random labels using full batch gradient descent, He initialization, no momentum or regularization, and learning rate 0.0025. Models with 1,2,3,4 layers had 298, 100, 75, and 63 hidden units per layer and 15208, 15210, 15235, and 15139 parameters, respectively. All models train successfully, but deeper models require fewer epochs.



3.19 Deep Neural Network playground

4 Backpropagation

How do we find $\nabla \mathcal{L}(h_\theta)$ for deep networks?

$$L'(\theta, \psi, \phi) = \ldots ?$$

4.1 Chain rule (1-D recap)

> If $y = f(g(x))$
> and $u = g(x)$ then

$$\frac{dy}{dx} = \frac{dy}{du}\,\frac{du}{dx}.$$

One derivative “chain” per link.

4.2 Chain rule example

Let $u = g(x) = x^2, \quad y = f(u) = \sin u, \quad x = 0.3$

- In Calculus class:
  - $y = f(g(x)) = \sin(x^2)$
  - $f'(x) = \cos(x^2) \cdot 2x = \cos(u) \cdot u'$
- Forward
  - $u = g(0.3) = 0.09, \quad y = f(u) = \sin 0.09 \approx 0.0899$
- Gradients
  - $\frac{dy}{du} = \cos u = 0.996 \quad , \quad \frac{du}{dx} = 2x = 0.6$
  - $$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = 0.996 \times 0.6 \approx 0.598$$

4.3 Motivation for backpropagation

Take the toy model:

$$f(x,y,z) = (x+y)\,z = qz, \qquad x, y \xrightarrow{+} q \xrightarrow{\times z} f.$$

To get every gradient we build three separate chains, what is the problem?

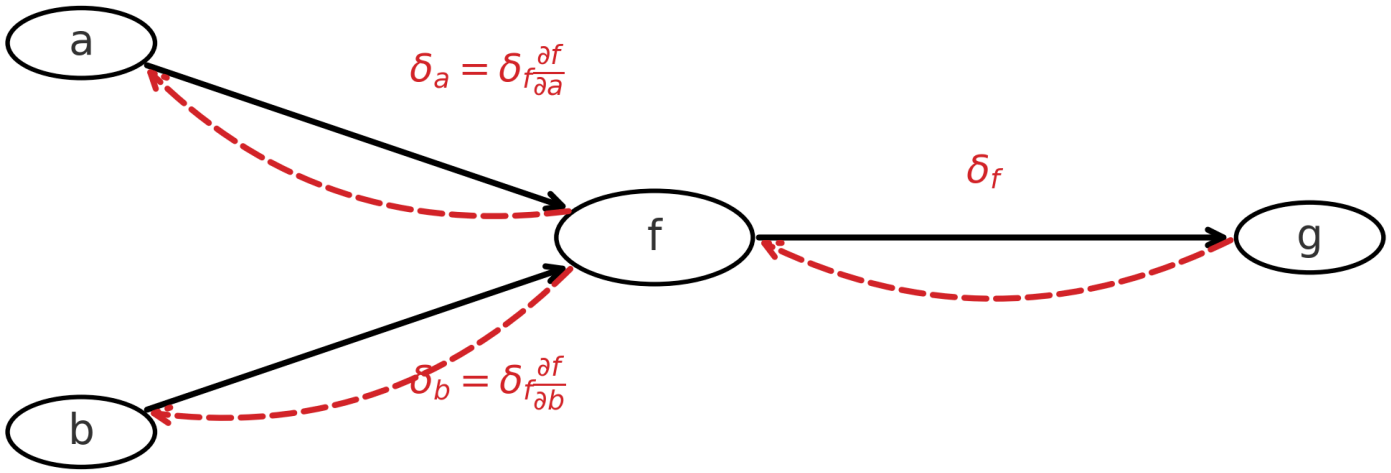
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\,\frac{\partial q}{\partial x} = z \cdot 1 = z$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\,\frac{\partial q}{\partial y} = z \cdot 1 = z$$

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

4.3 Backpropagation

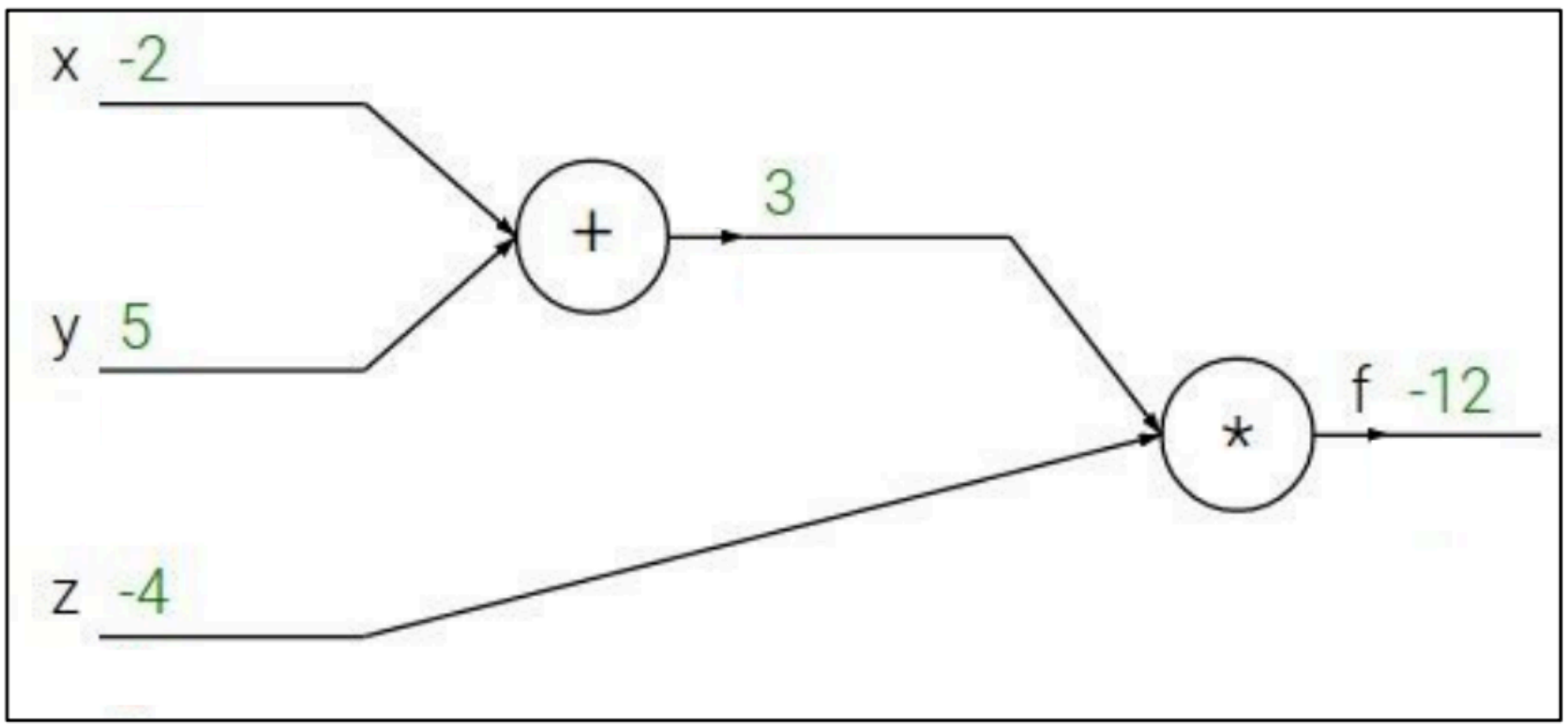
- We need to compute the gradient of the loss function with respect to each parameter in the network. $\frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial \psi}, \frac{\partial L}{\partial \phi}$
- Use the chain rule to compute gradients efficiently by reusing intermediate results from the forward pass.


a
b
f
g
$\delta_a = \delta_f \frac{\partial f}{\partial a}$
$\delta_b = \delta_f \frac{\partial f}{\partial b}$
δ_f

4.4 Backpropagation example

$$f(x,y,z) = (x+y)\,z = qz, \qquad x, y \xrightarrow{+} q \xrightarrow{\times z} f.$$

- Let $x = -2, y = 5, z = -4$. Need



4.5 Backprop algorithm (simplified)

1. Build the computational graph
2. Forward pass: compute all intermediate values
3. Backward pass: compute gradients (once!) using the chain rule
4. Update parameters using gradients

5 Implementing and training models

5.1 Defining the model

```
W1 =  HeNormal([h1_dim, in_dim])     # weight matrix
b1 =  zeros([h1_dim, 1])

W2 =  HeNormal([h2_dim, h1_dim])
b2 =  zeros([h2_dim, 1])

W3 =  HeNormal([out_dim, h2_dim])
b3 =  zeros([out_dim, 1])

forward(X):
  z1 = W1 @ X + b1
  a1 = ReLU(z1)

  z2 = W2 @ a1 + b2
  a2 = ReLU(z2)

  z3 = W3 @ a2 + b3
  ŷ  = z3
```

5.2 The deep learning training loop

```
import loss_function, SGD, backprop, forward

optimizer = SGD(learning_rate=0.01)

for epoch in range(num_epochs):
  optimizer.zero_grad()

  # Forward pass
  Y_hat = forward(X)
  loss = loss_function(Y_hat, Y)

  # Backward pass
  gradients = backprop(loss, W1, b1, W2, b2, W3, b3)

  # Update parameters
  optimizer.step(gradients)
```