FH KREMS
UNIVERSITY OF APPLIED SCIENCES / AUSTRIA

Machine Learning, Artificial Intelligence, and Big Data Analytics (IL, 4th Semester)

Lecture 10

Agenda

- Introduction to Neural Networks
- The building blocks of a neural network
  - The perceptron: Neurons, weights, and activation function
- Deep Neural Network
- Training a neural network
  - Loss function, Gradient descent, backpropagation

- CODING.

Some history in pills

The Rosenblatt perceptron, 1960

- 1949: Donald Hebb proposes the Hebbian Learning principle
- 1951: Marvin Minsky creates the first ANN (Hebbian learning, 40 neurons).
- 1958: Frank Rosenblatt creates a perceptron to classify 20 × 20 images.

<1974-1980: 1st AI winter>

- 1980: Kunihiko Fukushima presents the Neocognitron, basis for convolutional NN
- 1982: Paul Werbos proposes back-propagation for ANN.

<1987-1993: 2nd AI winter>

- 21st century: Resurgence
- 2010-ongoing: AI spring, deep learning explosion

Why the new AI spring?

- Big data and cloud
  - Large datasets
  - Easier and cheaper collection & storage
- Hardware
  - GPUs/APUs
  - Parallelization

- Software
  - Frameworks and toolboxes

FH KREMS
UNIVERSITY OF APPLIED SCIENCES / AUSTRIA

Popular Deep Learning frameworks

TensorFlow + Keras

- Developed by **Alphabet**
- Written in C++
- Now integrates **Keras**

- Popular in production
- Trickier debugging

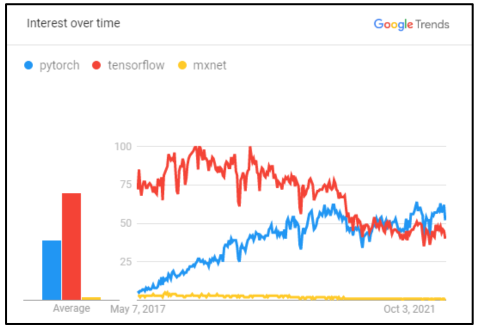
PyTorch

- Developer by **Meta**
- Written in Python/C++
- Based on **Torch**

- Popular in academia
- Easier debugging

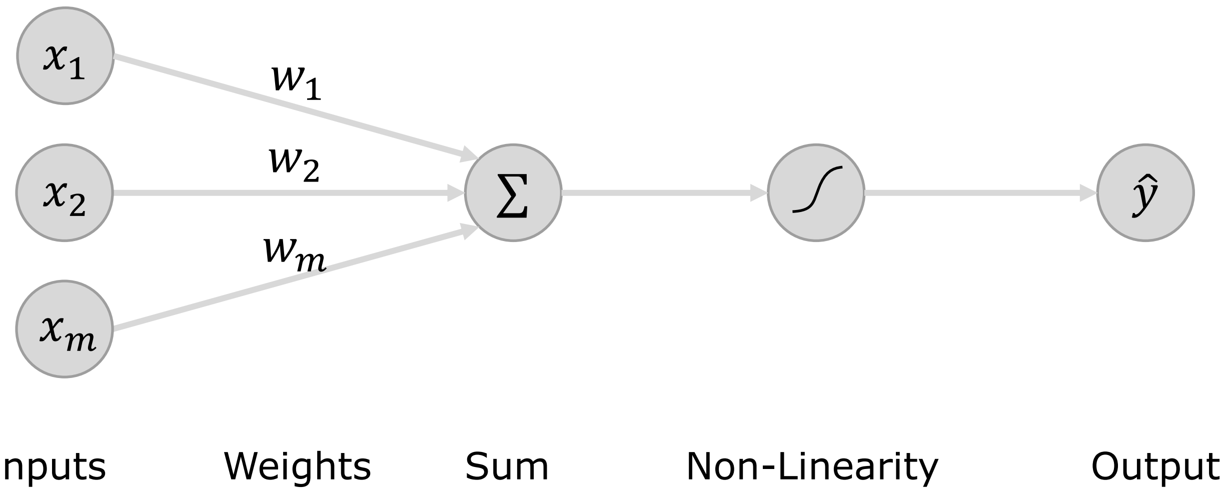
mxnet

- Academic and **Apache**
- Written in C++
- Multi-language support but less popular



The perceptron

The basic building block of a neural network



Output

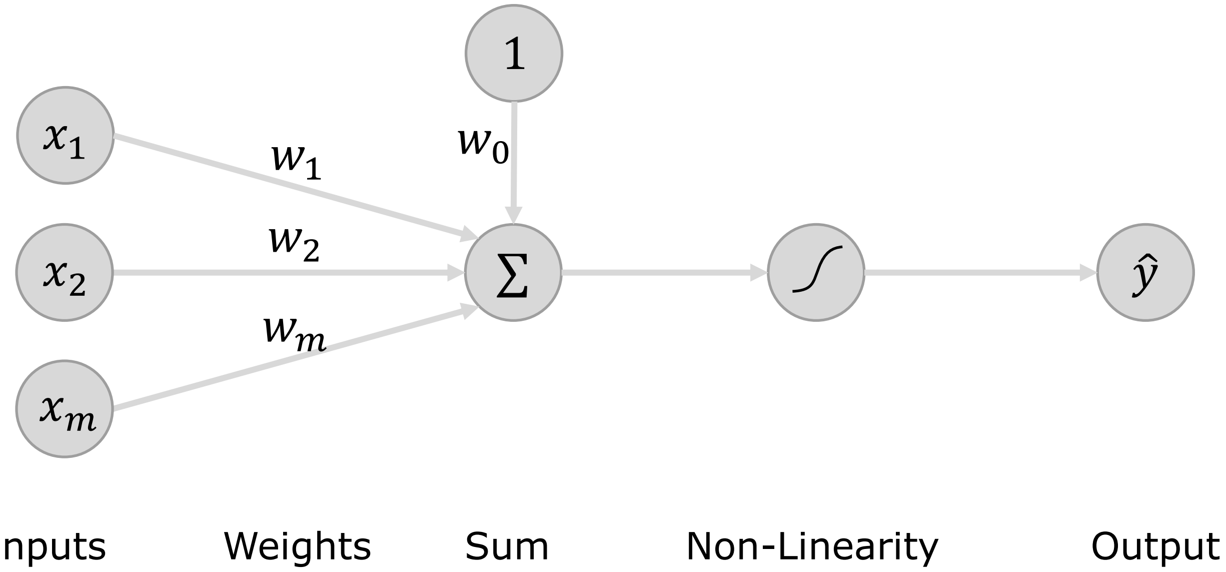
Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

The perceptron

The basic building block of a neural network



Output

Linear combination of inputs

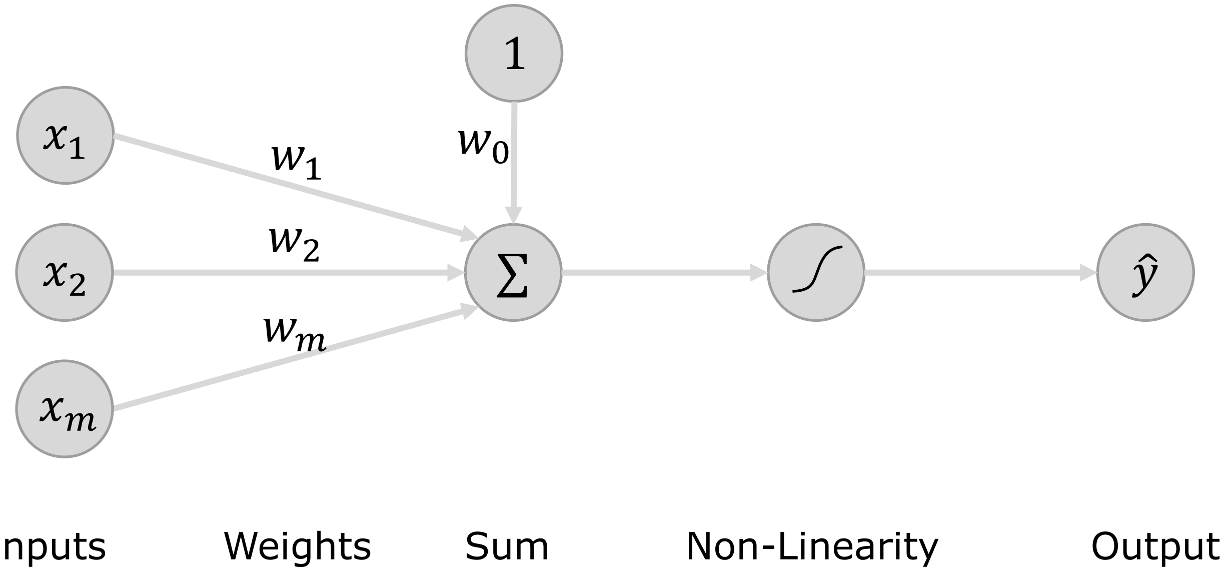
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

Bias

The perceptron

The basic building block of a neural network



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

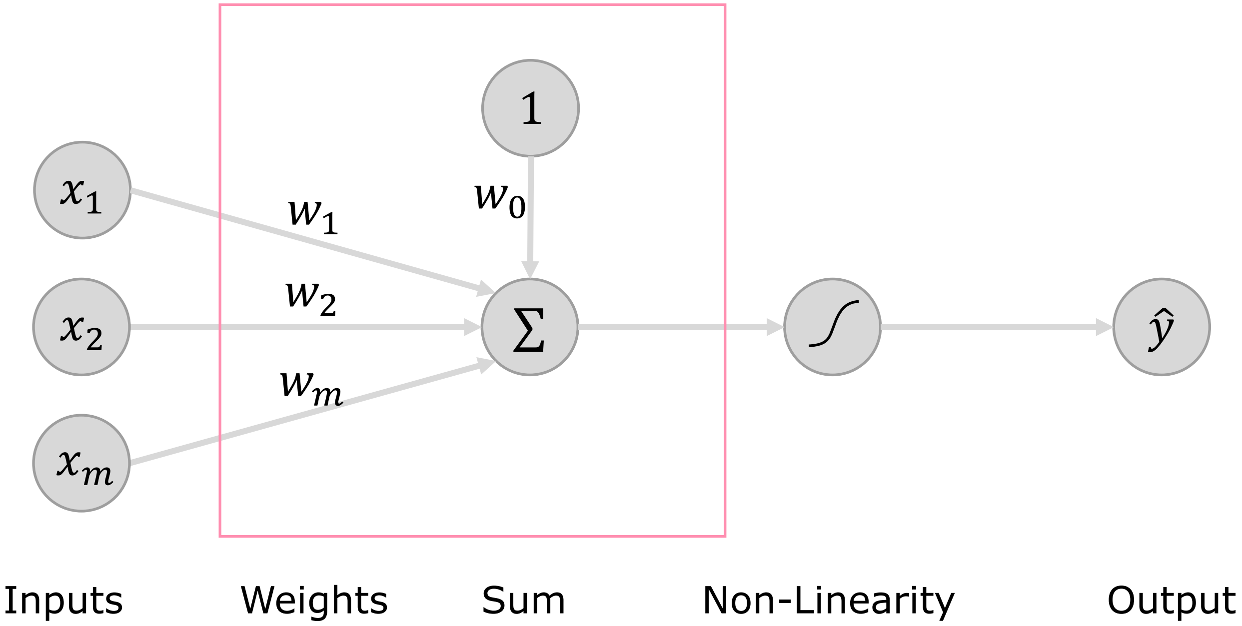
Bias

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$where\ \boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}\ and\ \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The perceptron

The basic building block of a neural network



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Non-linear activation function

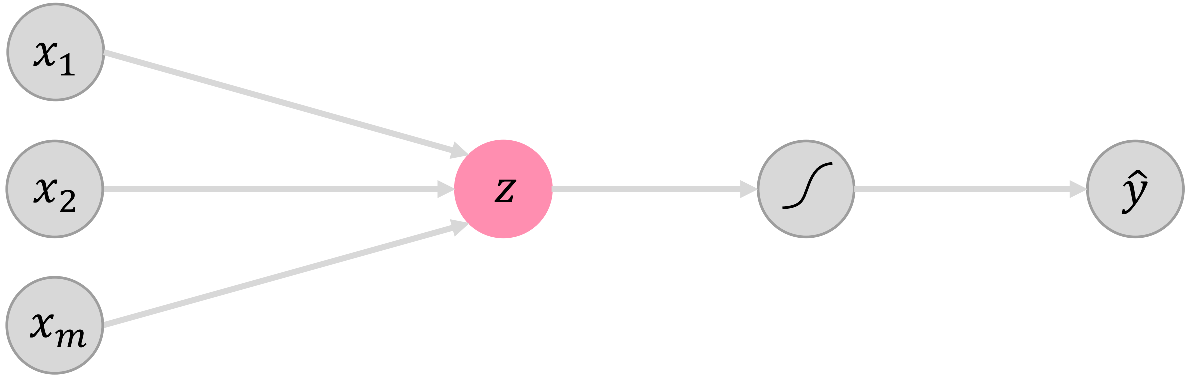
Bias

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$where\ \boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}\ and\ \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The perceptron

The basic building block of a neural network



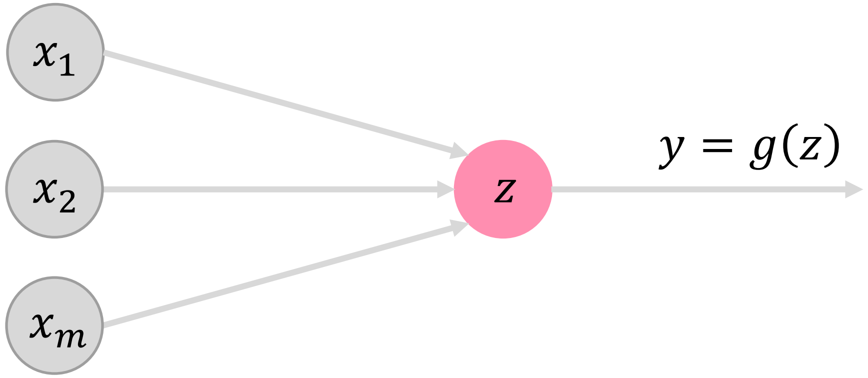
Inputs

$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

Output

The perceptron

The basic building block of a neural network

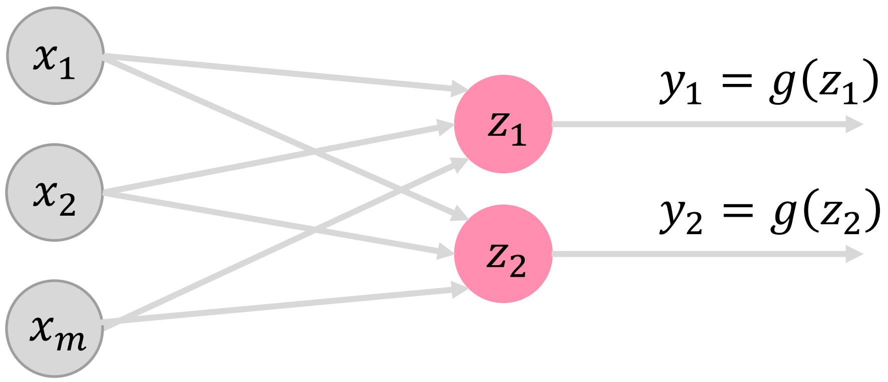


Inputs

$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

The perceptron

The basic building block of a neural network



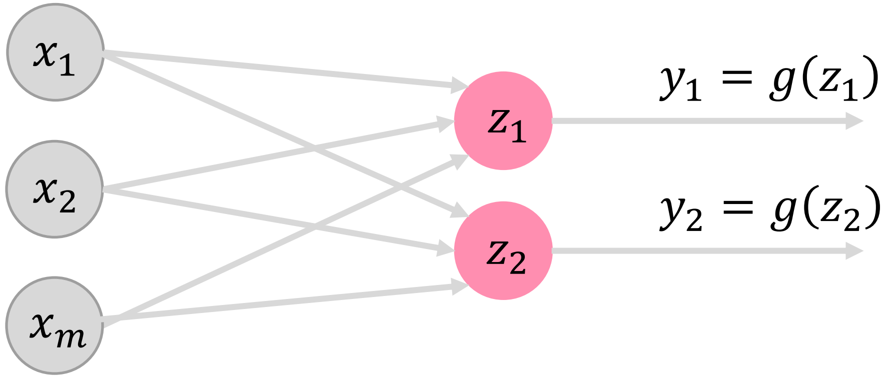
Inputs

$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$

Multi-output perceptron

The perceptron

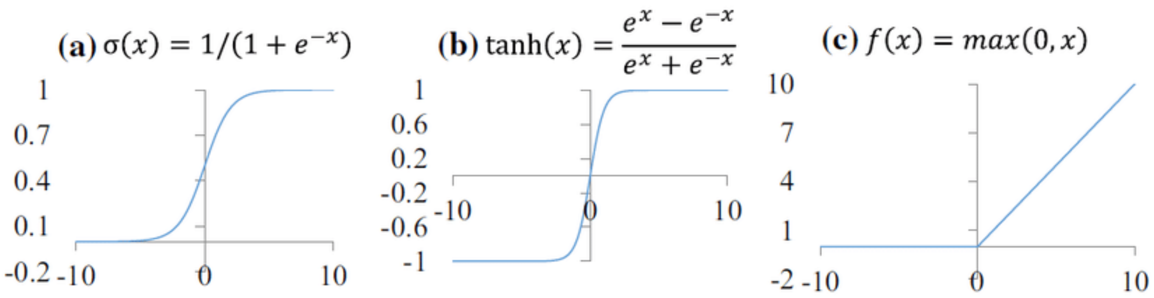
The basic building block of a neural network



Inputs

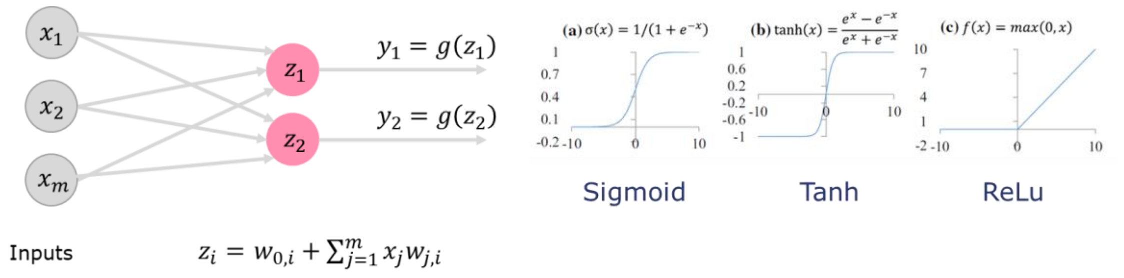
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

Common activation functions g



Sigmoid Tanh ReLu

the most used one



The perceptron

Example of output from the multi-output perceptron

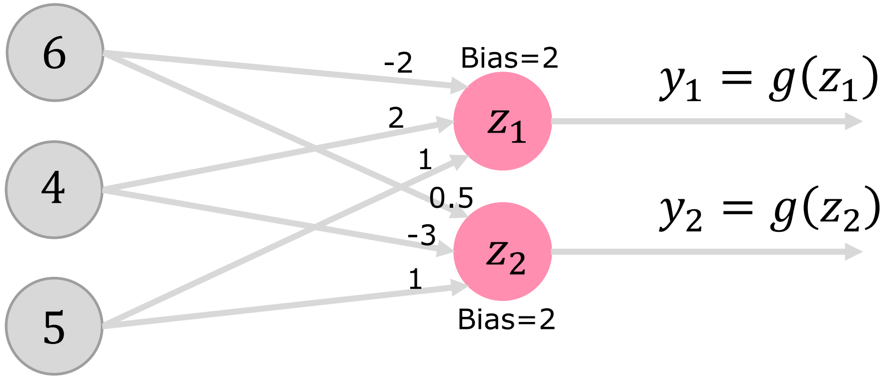
$x_1 = 6$
$x_2 = 4$
$x_3 = 5$

$w_{0,1} = 2$ $\quad w_{0,2} = 2$

$w_{1,1} = -2$ $\quad w_{1,2} = 0.5$

$w_{2,1} = 2$ $\quad w_{2,2} = -3$

$w_{3,1} = 1$ $\quad w_{3,2} = 1$



$$z_1 = 2 + (-2 * 6) + (2 * 4) + (1 * 5) = 3$$

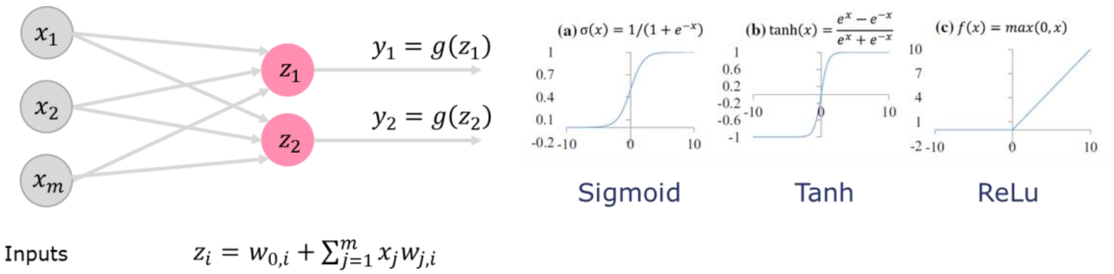
$$z_2 = 2 + (0.5 * 6) + (-3 * 4) + (1 * 5) = -2$$

input * weight + bias = Z

$$y_1 = g(z_1) = \begin{cases} \sigma(3) = \sim 0.952 \\ \tanh(3) = \sim 0{,}995 \\ \mathrm{ReLU}(3) = 3 \end{cases}$$

$$y_2 = g(z_2) = \begin{cases} \sigma(-2) = \sim 0.119 \\ \tanh(-2) = \sim -0{,}964 \\ \mathrm{ReLU}(-2) = 0 \end{cases}$$

The perceptron

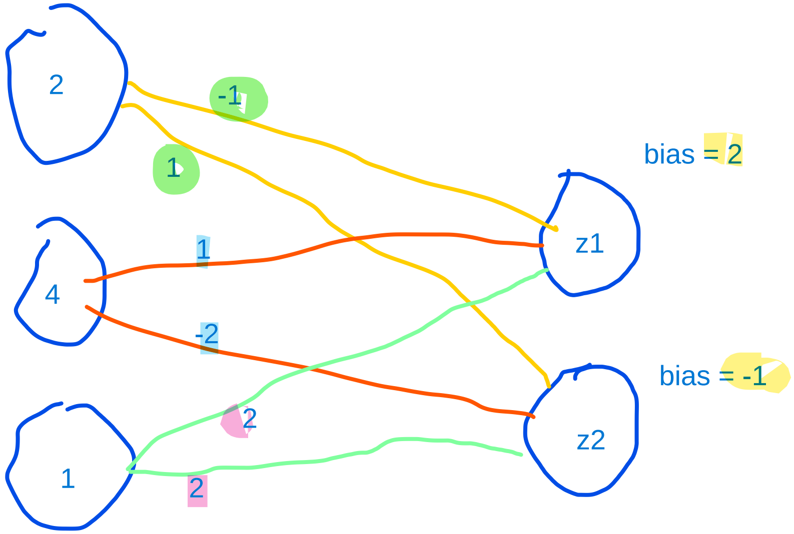


$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$

Exercise

$x_1 = 2$
$x_2 = 4$
$x_3 = 1$

$$W = \begin{pmatrix} 2 & -1 \\ -1 & 1 \\ 1 & -2 \\ 2 & 2 \end{pmatrix}$$



bias + (all the weights * inputs) = result

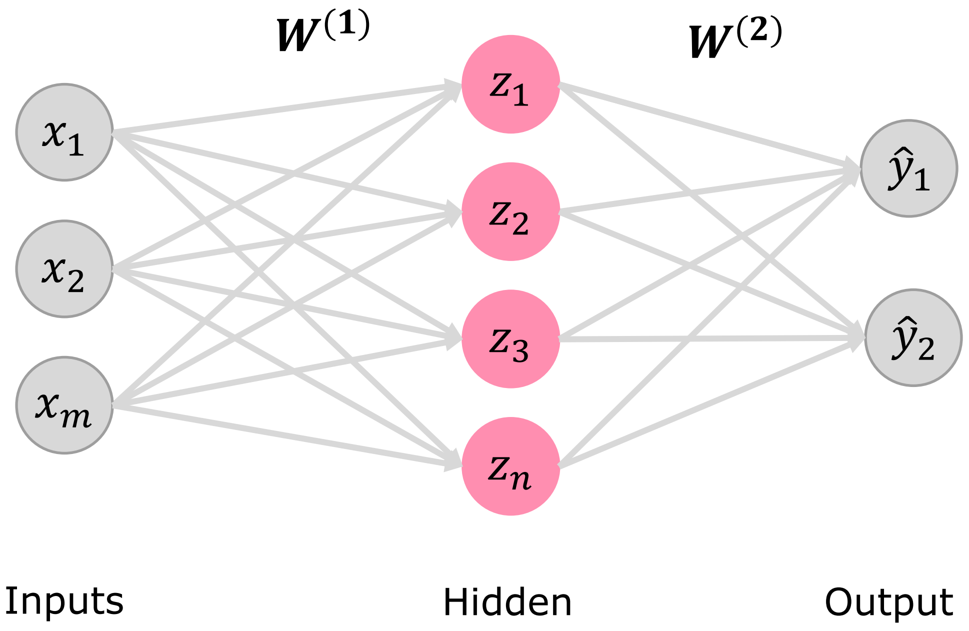
z1 = 2 + (2*-1) + (4*1) +(1*2) = 6

z2 = -1 + (2*1) + (4*-2) +(1*2) = -5

convert Z1 to
sigmoid(6) = 1
Tanh(6) = 1
Relu(6) = 6

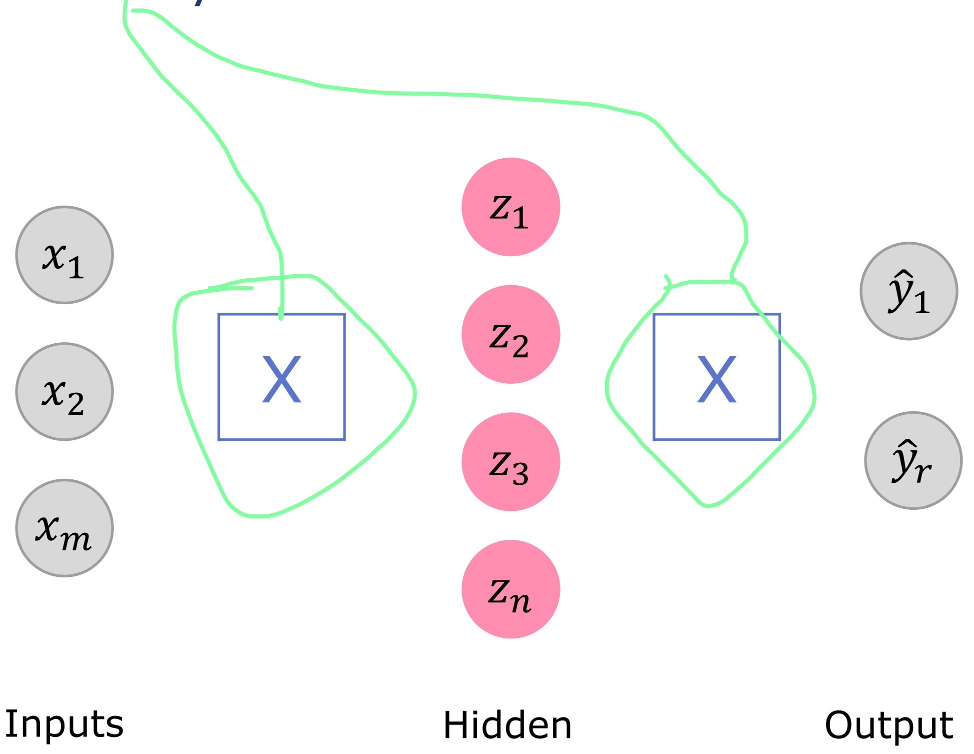
convert Z2
sigmoid(-5) = 0
Tanh(-5) = -1
Relu(-5) = 0

Single layer neural network



Single layer neural network

Dense layers



Since they are densely connected, they are called **dense layers**.

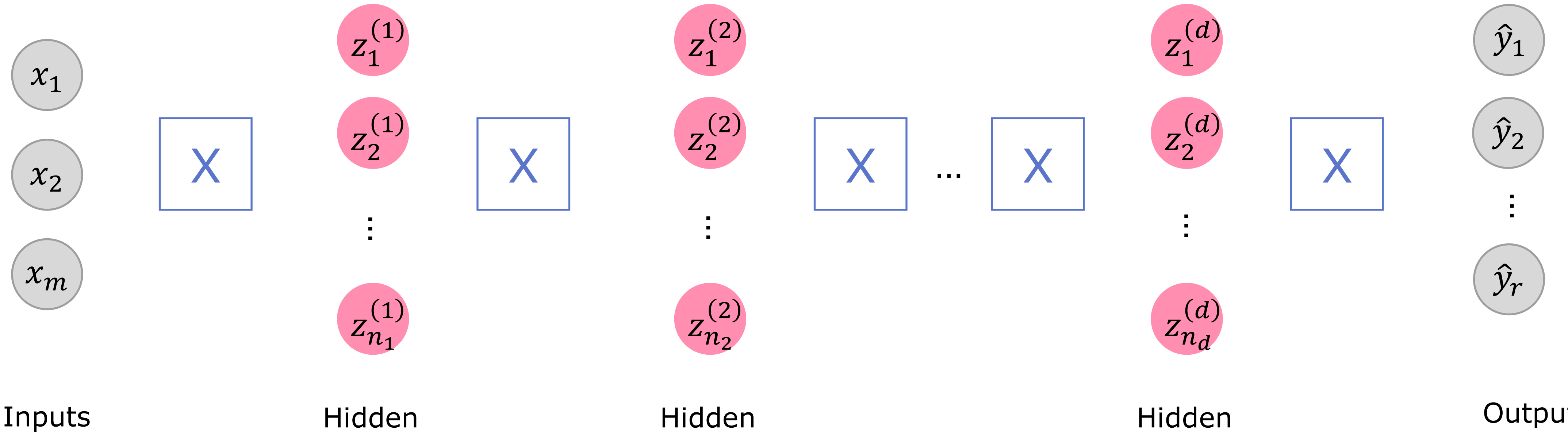
Here we have one input layer and two dense layers:

- 1 dense layer with m inputs and n outputs
- 1 dense layer with n inputs and 2 outputs

Multiple layer (deep) neural network

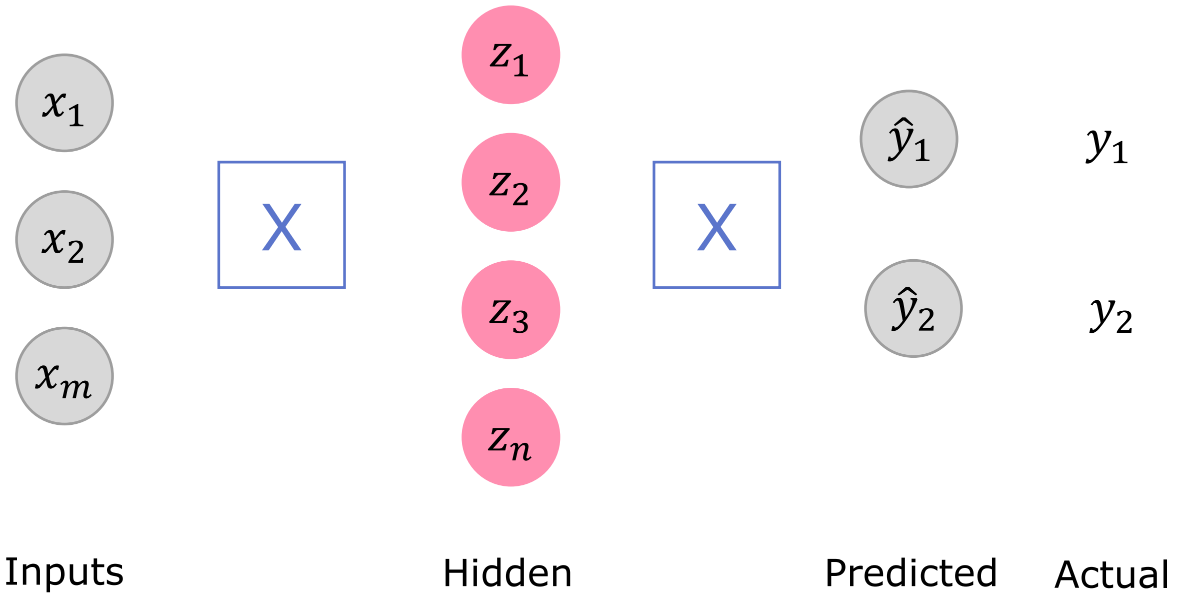
A LOT

Generalization



Training a neural network

Objective



<u>Goal</u>

Identifying parameters (biases and weights) such that the output of the neural network is as close as possible to the ground-truth.

In other words: **Minimize a loss function.**

Training a neural network

Loss functions

- Examples:

  Regression: Mean Squared Error Loss

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2$$

  Classification: Binary Cross Entropy Loss (Log-loss)

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

Training a neural network

Loss functions

- Examples:

  Regression: Mean Squared Error Loss

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

Example

$$\begin{array}{cc} \hat{y} & y \\ \begin{bmatrix}12\\23\\41\end{bmatrix} & \begin{bmatrix}10\\25\\40\end{bmatrix} \end{array} \quad L(y,\hat{y}) = \frac{(2)^2+(-2)^2+1^2}{3} = 3$$

  Classification: Binary Cross Entropy Loss (Log-loss)

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=0}^{N} -(y_i * \log(\hat{y}_i) + (1-y_i) * \log(1-\hat{y}_i))$$

Example

$$\begin{array}{cc} \hat{y} & y \\ \begin{bmatrix}0.1\\0.7\\0.2\end{bmatrix} & \begin{bmatrix}0\\1\\0\end{bmatrix} \end{array}$$

$$L(y,\hat{y}) = \\ = \frac{-(\log(1-0.1) + \log(0.7) + \log(1-0.2))}{3} \\ = \frac{0.29}{3} = \sim 0.1$$

possible to compute by hand in exam

Loss optimization

gradient points where the value increases

The gradient

- The loss is a mathematical function of the various weights in the network
- We want to find the set of weights that achieve the lowest loss

$\boldsymbol{W}^* = \operatorname{argmin}_w J(\boldsymbol{W})$, where J is the average Loss

- Gradient of a function: A vector that points in the direction of steepest ascent.
- If we take the opposite direction of the gradient we move towards a local minimum

Loss optimization

The gradient descent algorithm

Example in two dimensions

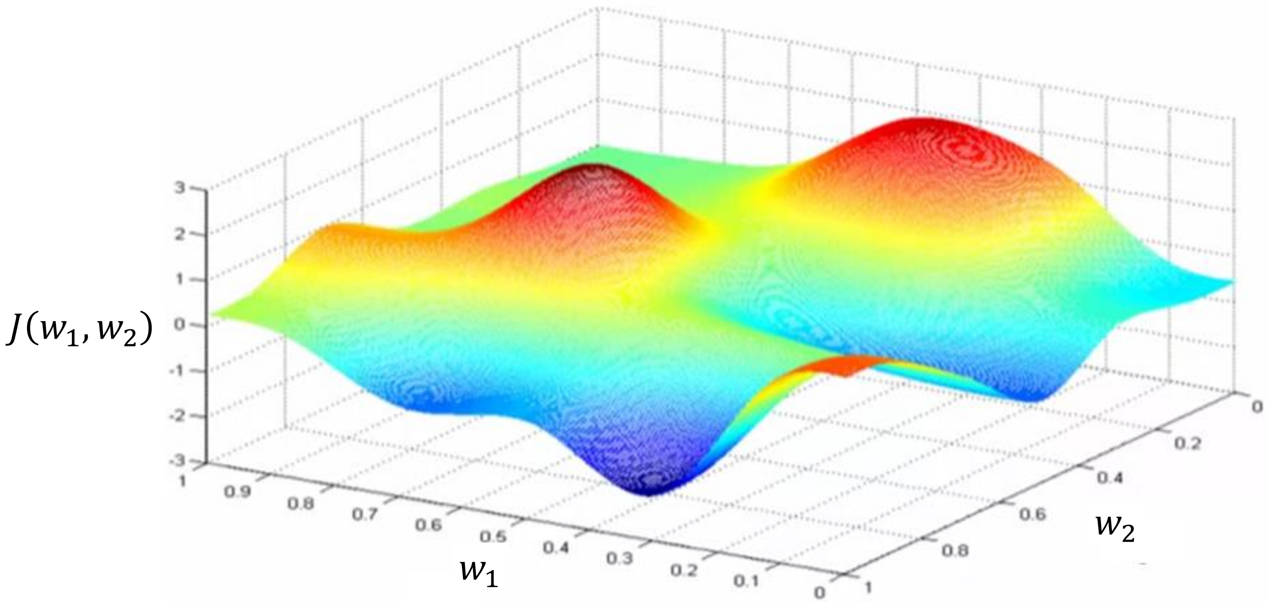


Image from: Andrew Ng (Coursera) https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent

Loss optimization

The gradient descent algorithm

Example in two dimensions

1. Pick a point (initialize weights)

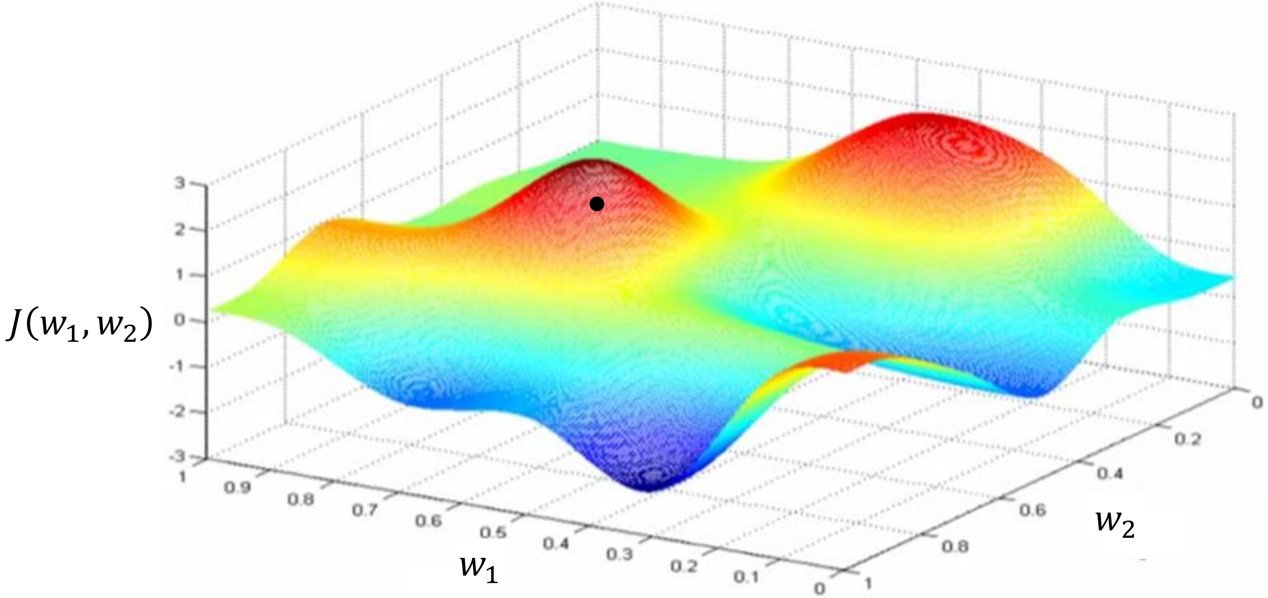


Image from: Andrew Ng (Coursera) https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent

Loss optimization

The gradient descent algorithm

Example in two dimensions

1. Pick a point (initialize weights)
2. Compute the gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

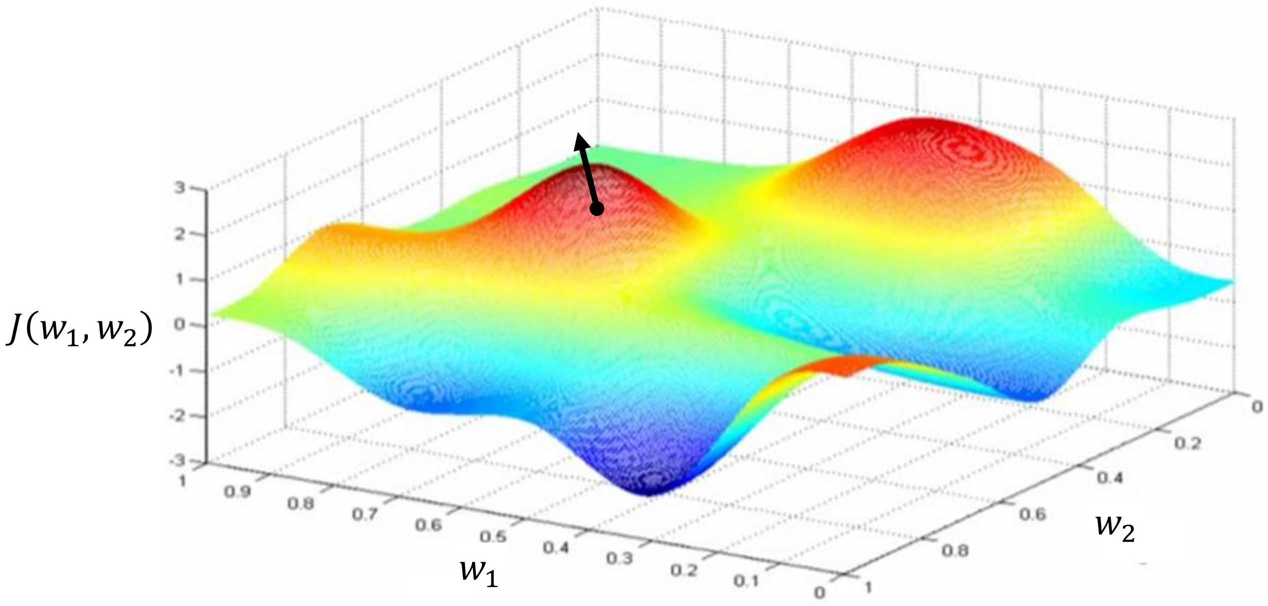


Image from: Andrew Ng (Coursera) https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent

Loss optimization

The gradient descent algorithm

1. Pick a point (initialize weights)
2. Compute the gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Take a step in the opposite direction

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

Example in two dimensions

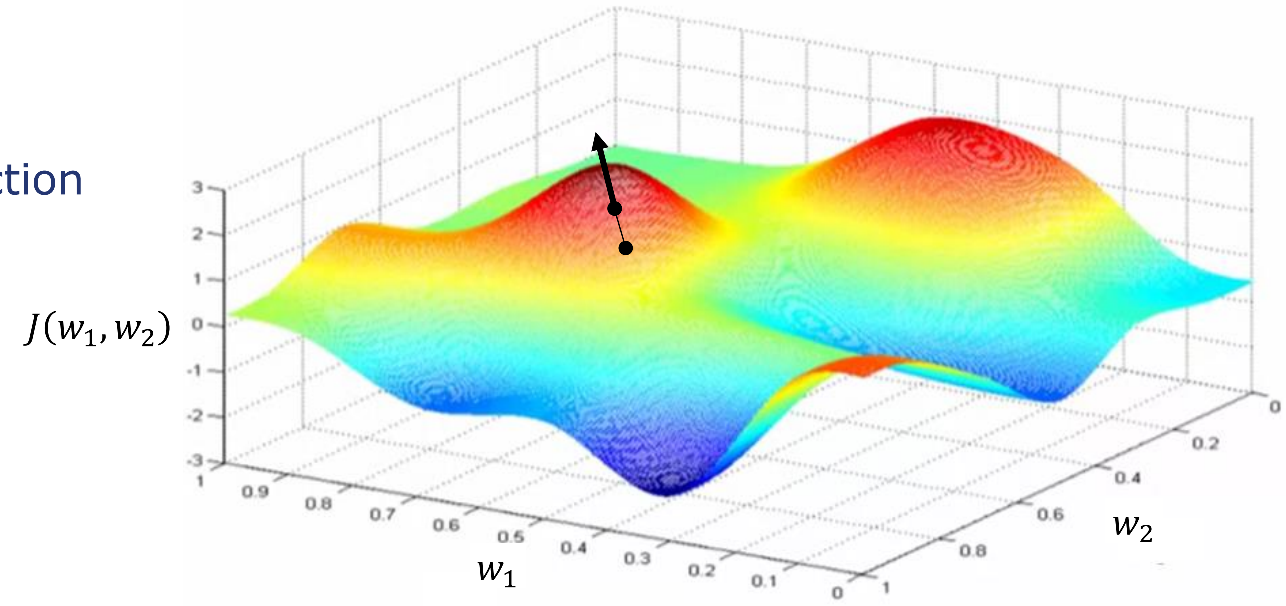


Image from: Andrew Ng (Coursera) https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent

Loss optimization

The gradient descent algorithm

1. Pick a point (initialize weights)
2. Compute the gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Take a step in the opposite direction

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

4. Repeat 2 and 3 until convergence

Example in two dimensions

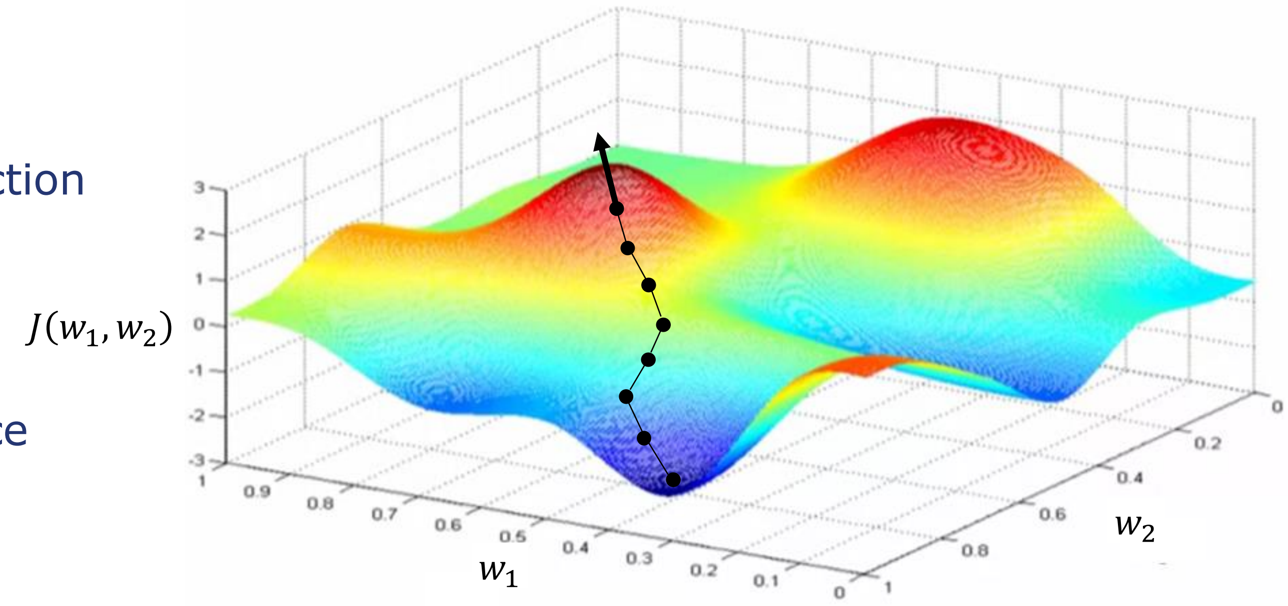


Image from: Andrew Ng (Coursera) https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent

Loss optimization

The gradient descent algorithm

1. Pick a point (initialize weights)
2. Compute the gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Take a step in the opposite direction

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

Learning rate

4. Repeat 2 and 3 until convergence

Example in two dimensions

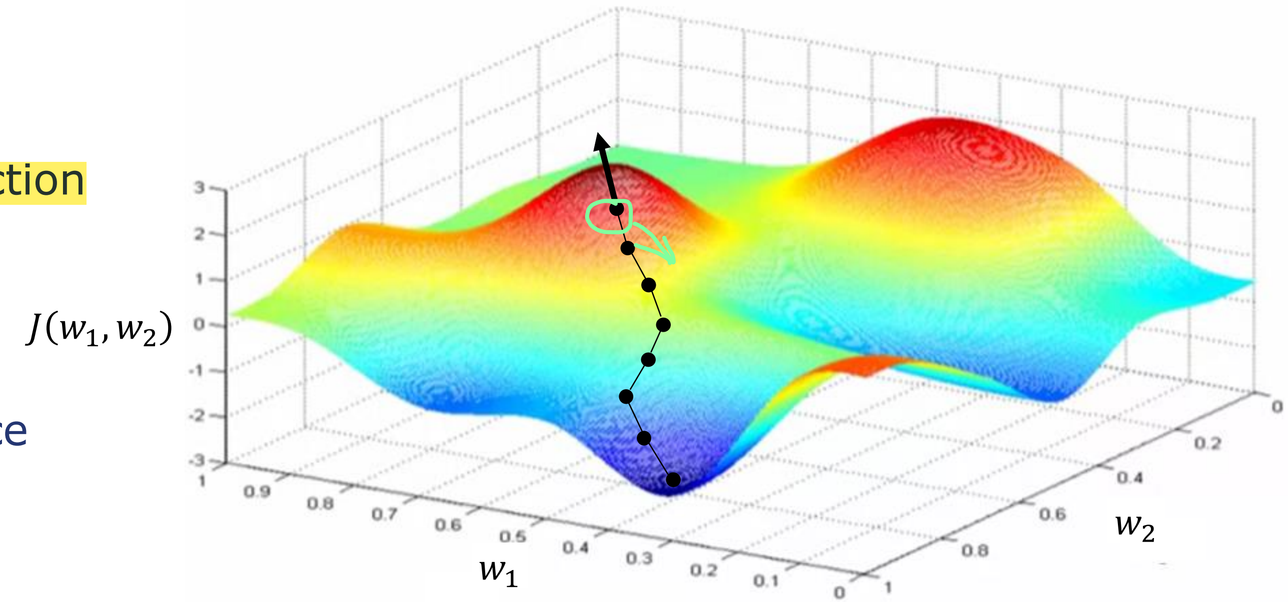
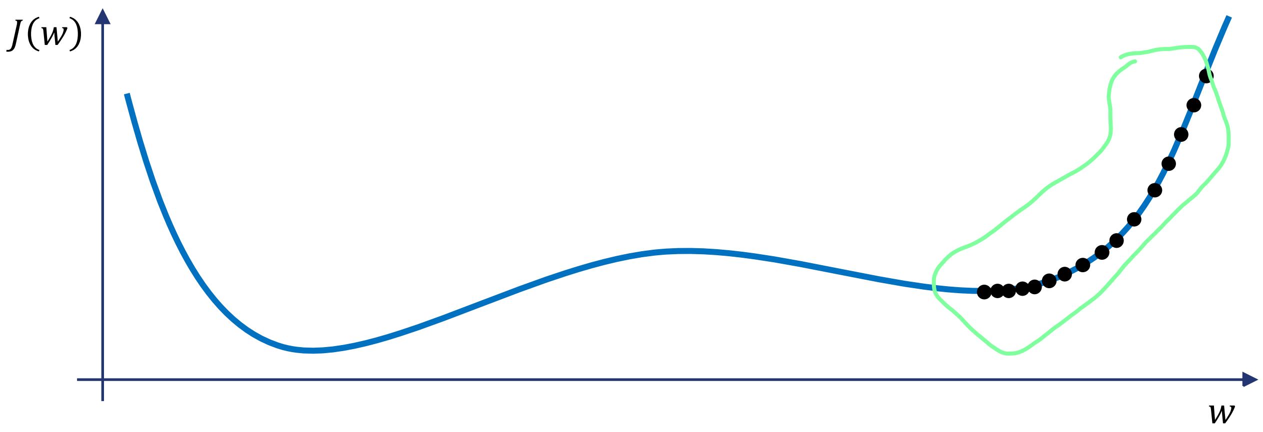


Image from: Andrew Ng (Coursera) https://www.coursera.org/learn/machine-learning/lecture/8SpIM/gradient-descent

Learning rate η

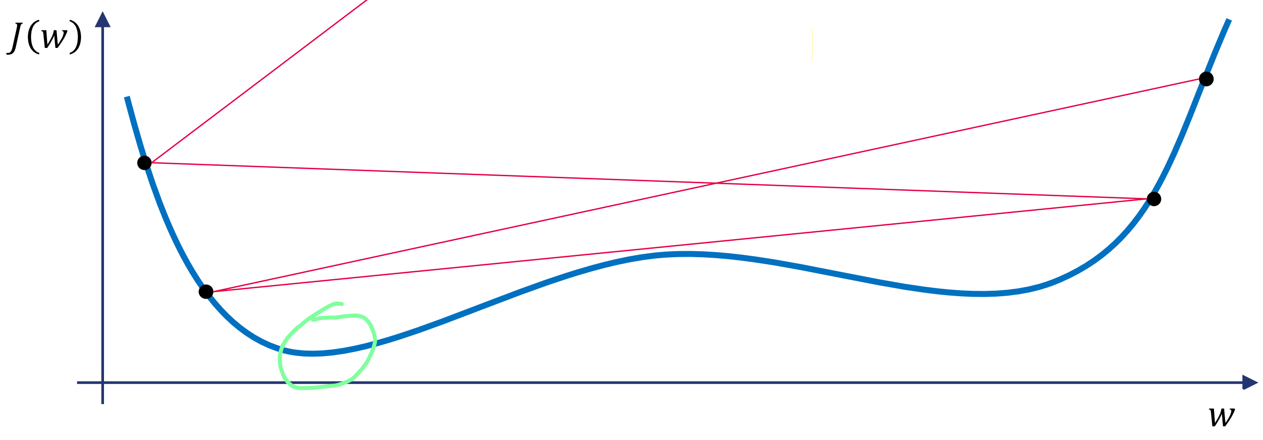
- **Small** learning rates increase the change of getting stuck into local minima



Learning rate η

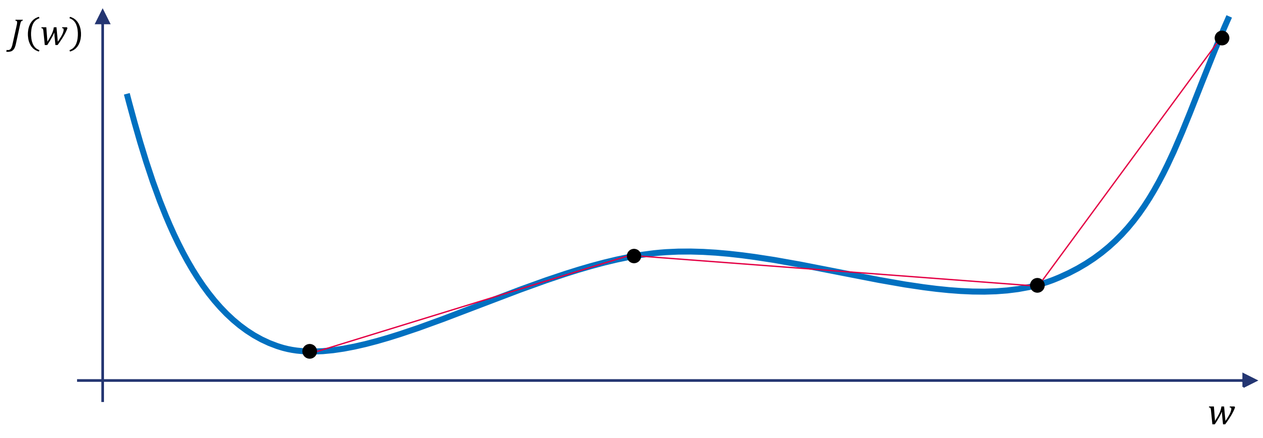
- **Large** learning rates increase the chance of divergence

skipping the local minima
jumping like crazy between the points and not finding the local minima



Learning rate η

- **Optimal learning rates** are hard to find



How to identify the right one?

1. Trying multiple learning rates
2. Adaptive learning rates
   - SGD
   - Adam
   - Adadelta
   - Adagrad
   - …

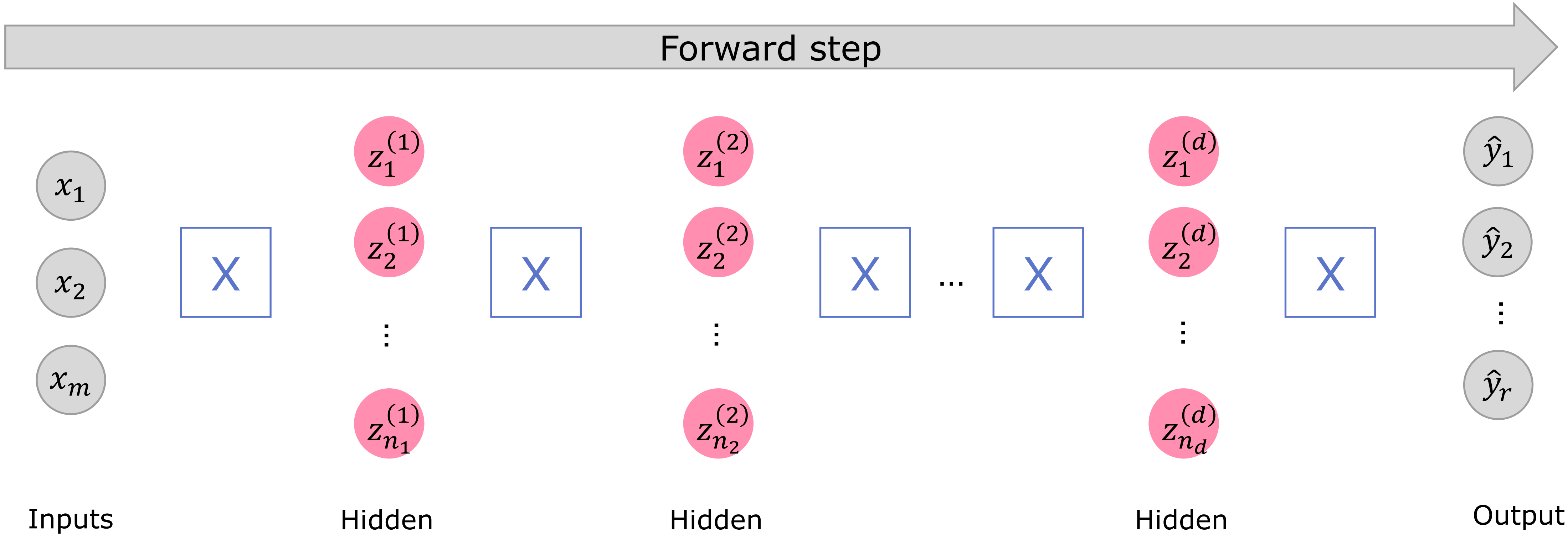
Many available in TF & pyTorch

Back-propagation

Why is it called backpropagation?

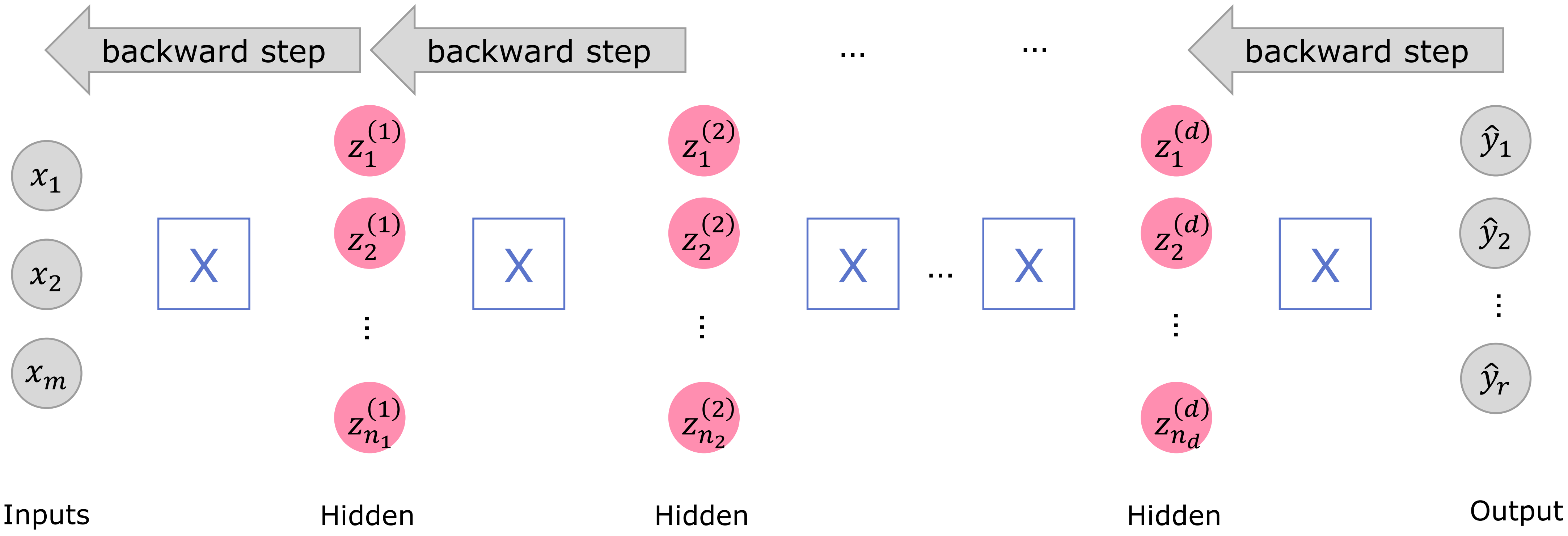
Train and adjust weights in the network
How it works:
comparing the network's predicted output to the expected output.
It calculates the difference between these two values, which is called the "error." The goal of backpropagation is to minimize this error.

From Output layer -> To Input
Technique: Gradient descend



Back-propagation

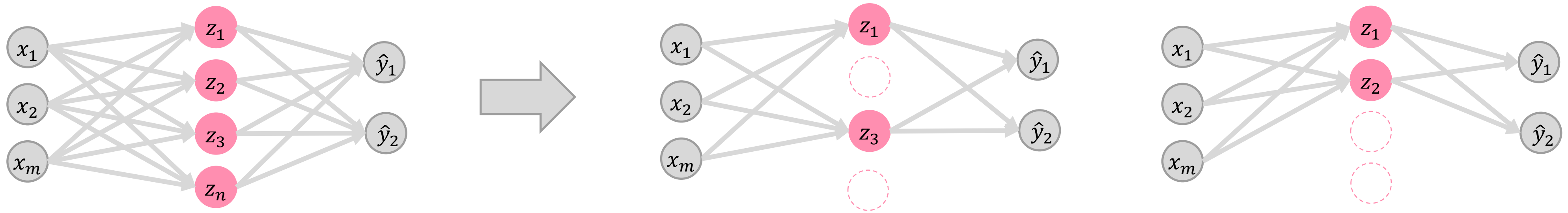
Why is it called backpropagation?



Overfitting in deep learning

Typical approaches

- Drop-out (randomly set some activation functions to 0).



- Early stop (stop training as soon as test error starts increasing)
  - Very similar to the traditional ML approach

Summary

- Fundamental building block: **The Perceptron**
- Stacking multi-output perceptrons sequentially: **Feed-Forward Neural Network**
- Training: finding the weights that **minimize a loss** function
  - Gradient Descent and back-propagation
- Overfitting: **Drop-out** or **early-stop**

- A short video and then… coding session

Exercise 4

Neural Networks

1. **Load the *Dry_Bean dataset***
   (HINT: library *readxl* to read excel files)
2. **Explore the dataset** (e.g. how many observations? How many classes? How many observations per class? How is each numeric variable distributed among classes? Are the classes distinguishable? etc.)
3. Create a **feed forward neural network** to predict the class of the beans by using the other variables as predictors
   1. Split the data (with ratio 80-20)
   2. Convert to tensors
   3. Experiments with different num of layer and nodes.
   4. Show results on training and test set

- **Due date: ??, 23.59 CET (Late submission +1week, 7 pts)**
- R Students: Use Rmarkdown/Rnotebook/Jupyter.
- Py Students: Use Jupyter
- Reports must contain code and results (no need to rerun)