ARTIFICIAL INTELLIGENCE NEEDS REAL INTELLIGENCE

Artificial Neural Network II

Professor: Tianyuan ZHANG
tianyuan.zhang@kedgebs.com

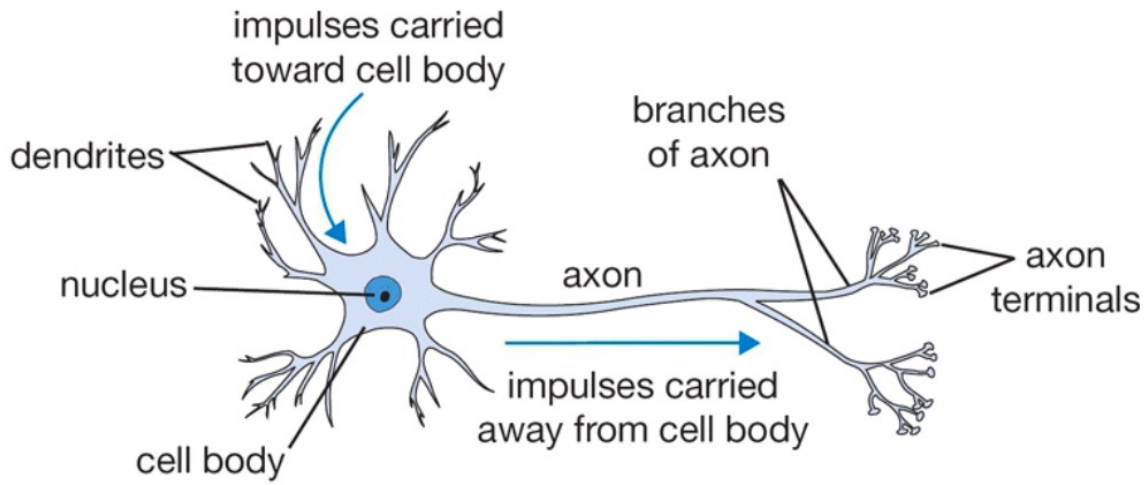
KEDGE BUSINESS SCHOOL

Recap of Previous Session

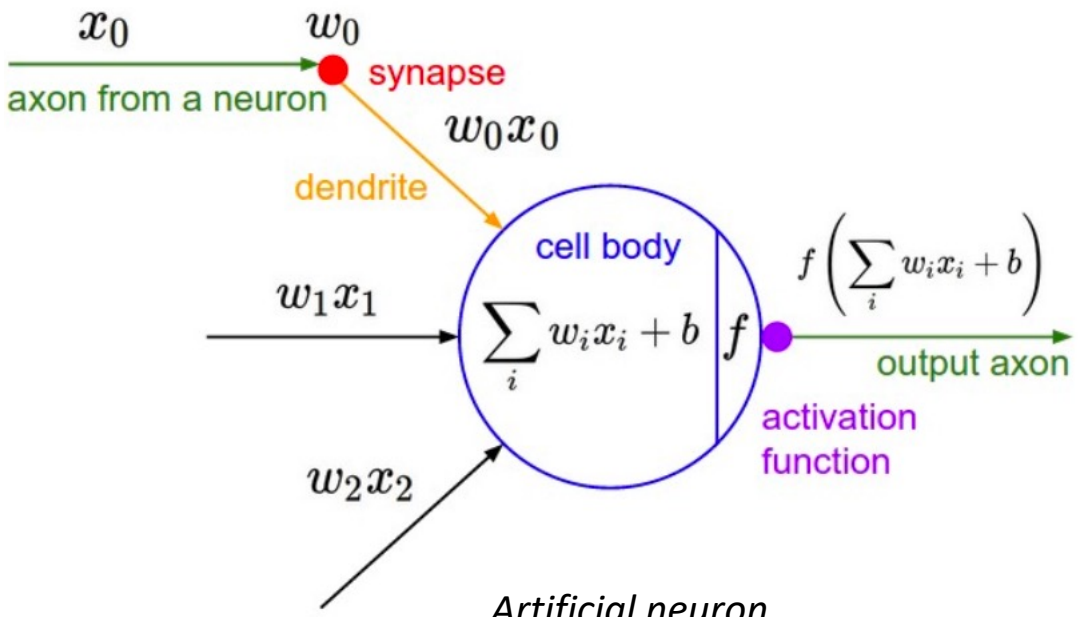
- Classical ML → **Deep Learning**
  - Parameter → Parameter of **artificial neuron**
  - Model → **Artificial neural networks** (consists of artificial neurons)
  - Learning process
    - Objective function → **Loss function**
      - **Conventionally lower is better**
    - Optimization algorithm → **Any optimization algorithm**
      - The dominant approach is based on **gradient descent**
- **Advantages** of deep learning
  - Artificial neural network and its variants can form very complex models (**deep**)
  - Various techniques to avoid over-fitting and ensure generalization

Recap of Previous Session

- **Artificial neuron**



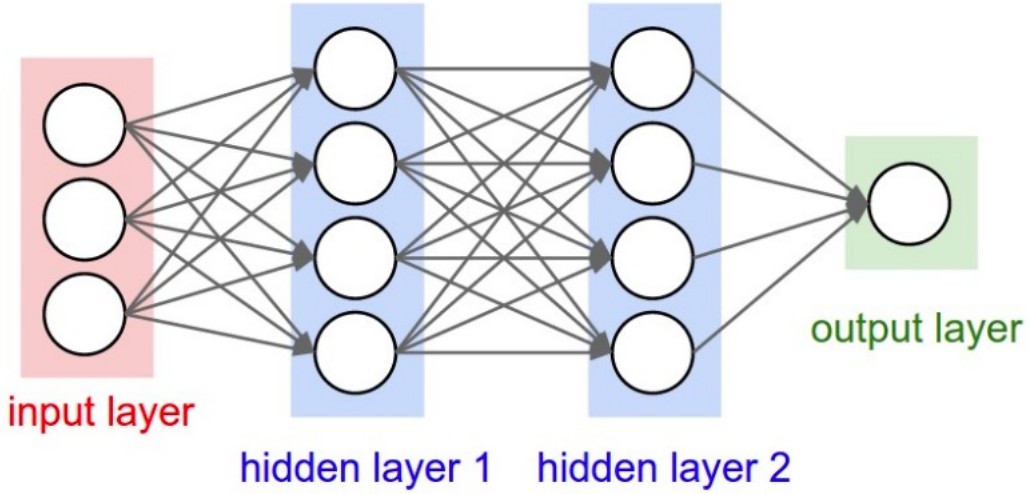
The basic computation unit of the brain: Neuron



Artificial neuron

Recap of Previous Session

- **Artificial Neural Network (ANN)**
  - A network consists of multiple connected artificial neurons arranged into different layers



A 3-layer neural network with two hidden layers

Naming conventions: a N-layer neural network

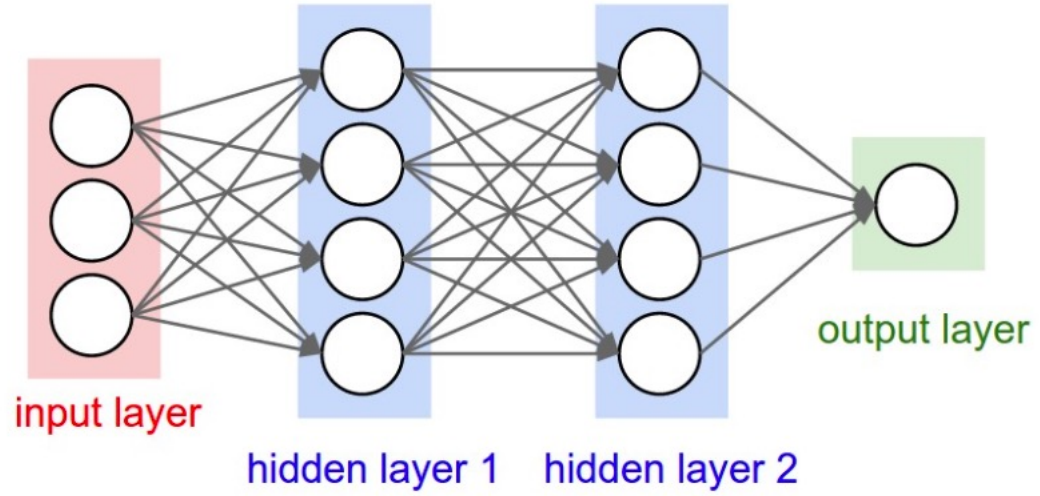
- N-1 hidden layers
- 1 output layer
- We do not count the input layer

Terminology:

- input nodes
- output nodes
- hidden layers
- connections
- weights
- activation function

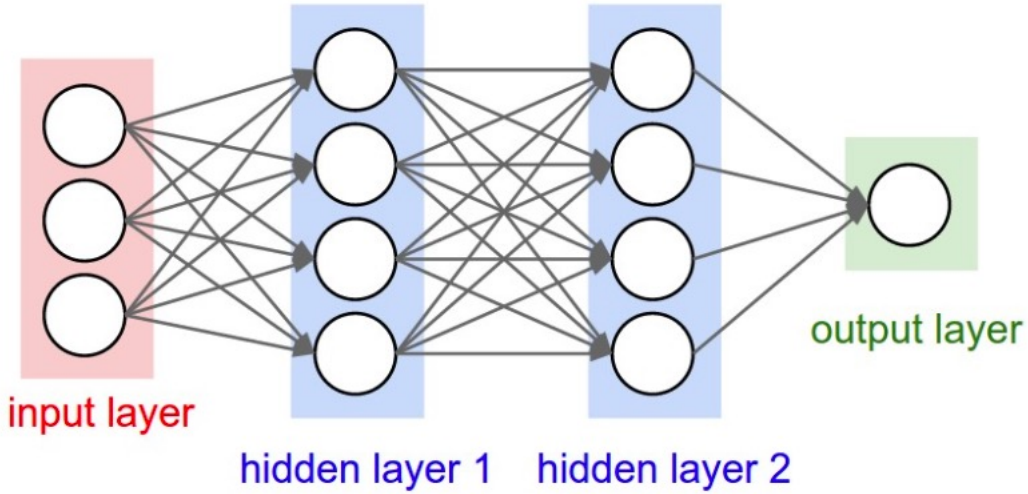
Recap of Previous Session

- **Artificial Neural Network (ANN)**
  - **Hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron
  - Parameters → **Learning parameters**
    - The weight of each connection
    - The bias of each neuron added to the weighted sum
  - The **capacity** (complexity, representational power) of a network can be assessed by **the number of learning parameters**



Recap of Previous Session

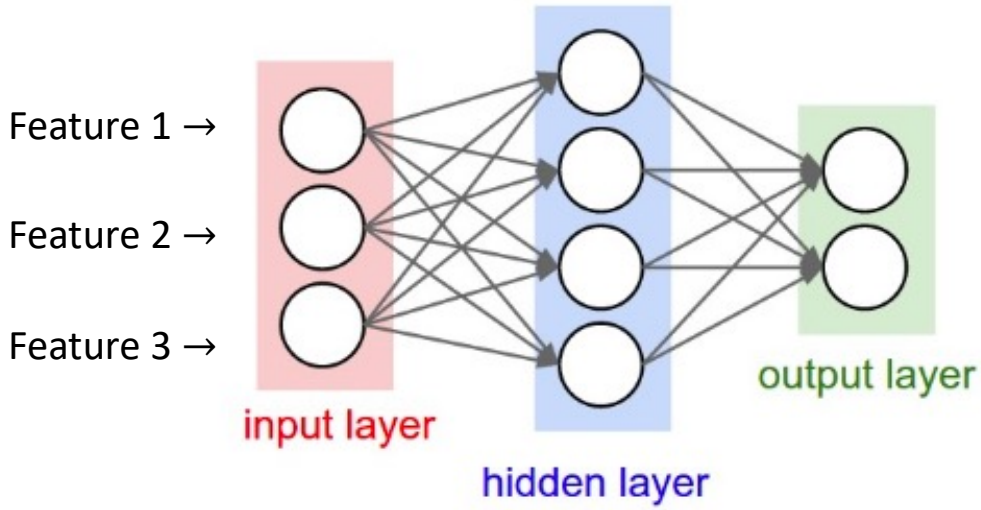
- How to **create** an ANN?
  - **Declare the hyper-parameters**
    - The number of layers
    - The number of neurons in each layer
    - The connections between neurons
    - The activation function of each neuron



- *The structure of the ANN determines its capacity:*
  - *If you ANN is under-fitted, consider to increase the capacity*
    - *Make it deeper (more layers, neurons, and connections)*
  - *Better to start with an over-fitted model which ensure your ANN having enough capacity*
    - *Then try to decrease the capacity by simplify the ANN*
    - *Or in most cases, applying other technique to prevent the ANN from overfitting*
      - *For example, early stopping*

Recap of Previous Session

- How does ANN perform **inference**?
  - Given an input instance
    - Feed the n features to the n input nodes
      - **The number of features should equal to the number of input nodes**
    - **Calculate** the output of each neuron
      - Layer by layer
      - Weights, bias, activation function
    - The output of the output node is the prediction made by the ANN
  - **Forward propagation**



Implemented using matrix operations:

- *Improved computation efficiency*
- *Parallelized and accelerated by GPU*

Recap of Previous Session

- How to **train** an ANN?
  - **Learning process** → Decide the values of **learning parameters**
    - **Loss function**
      - A formal measure of how good (or bad) the models are.
      - 'Loss' means lower value is better
      - Regression
        - **MSE, RMSE, MAE**
      - Classification
        - **Cross-entropy loss** (also called log loss)
    - **Optimization algorithm**
      - Search for the best values of learning parameters for minimizing the loss function
      - **Gradient descent**

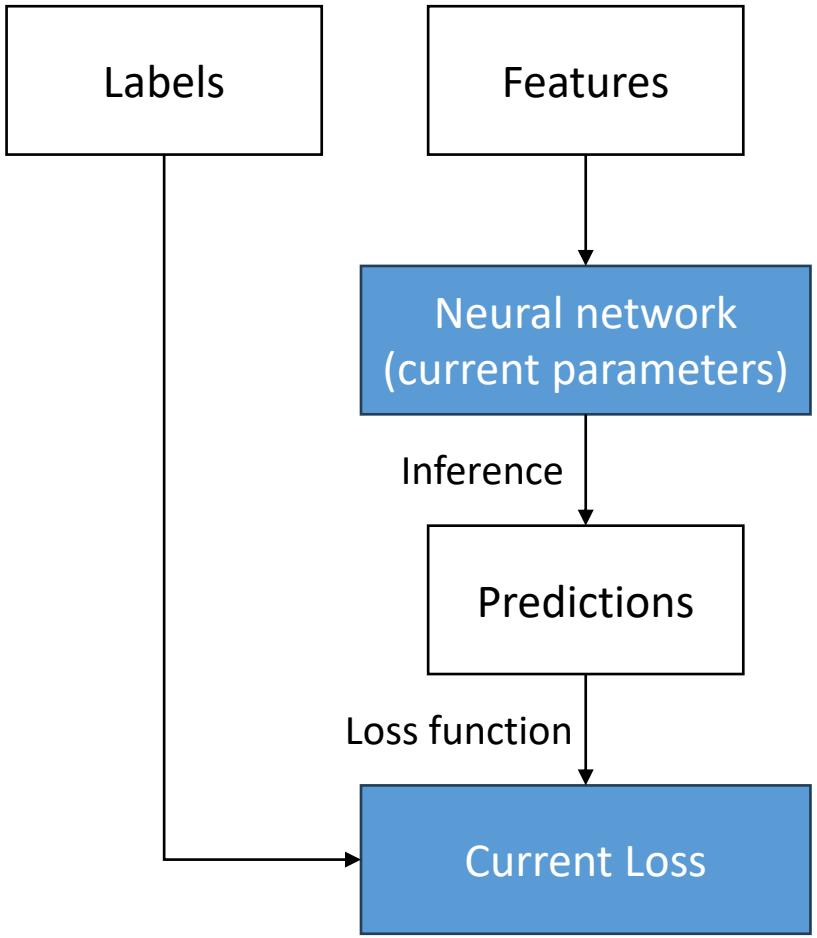
Outline

- **Gradient descent**
- Backpropagation
- Implement 1st ANN for regression

Gradient descent

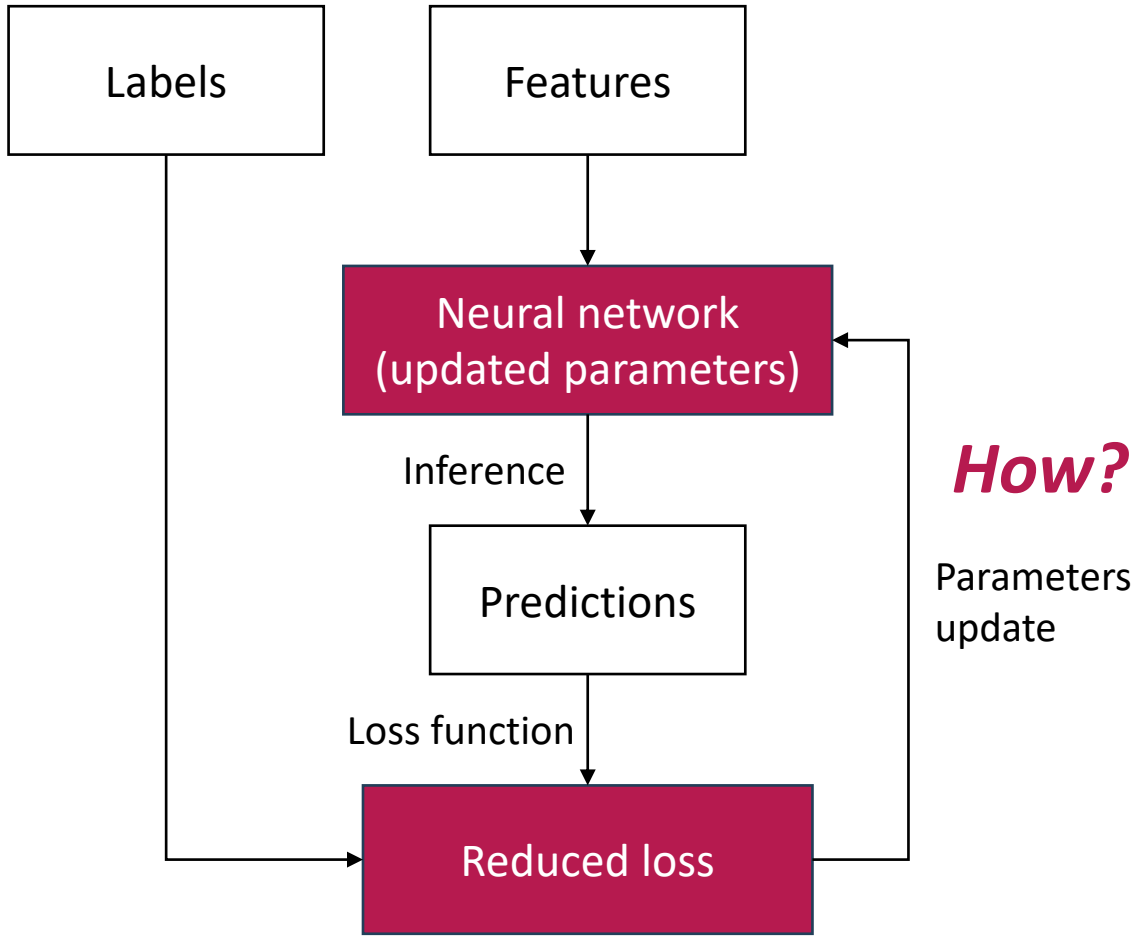
- How to **train** an ANN?
  - **Learning process**
    - **Decide the values of learning parameters**
      - Learn from the training data
      - Lead to the minimal loss value → best fitness
    - **An iterative process**

1. Make predictions on training data with current parameter values
2. Calculate loss by comparing predictions and labels
3. Update parameter values to reduce loss



Gradient descent

- How to **train** an ANN?
  - **Learning process**
    - **Decide the values of learning parameters**
      - Learn from the training data
      - Lead to the minimal loss value → best fitness
    - **An iterative process**
      1. Make predictions on training data with current parameter values
      2. Calculate loss by comparing predictions and labels
      3. **Update parameter values to reduce loss**



Gradient descent

- Gradient descent
  - An **iterative** algorithm
  - To find a **local minimum** of a **differentiable multivariate function**
    - Find the minimum value of a loss function
  - Take repeated steps in the opposite direction of the gradient of the function at the current point
    - The opposite direction of the gradient → The direction of steepest descent

Gradient descent

- An illustrative example
  - Suppose we have **an Artificial Neuron F**
    - With only **one weight parameter w**
  - Given a training sample (x, y)
    - The prediction is $\hat{y} = F(x, w)$
  - Use the pre-defined **loss function L**
    - The current loss is $L\big(y, F(x, w)\big) = L(x, y, w)$
    - **The gradient of L at w** is $\nabla w = \frac{\partial L}{\partial w}$
  - Update the parameter value
    - $\boldsymbol{w = w - \eta \times \nabla w}$

Take one step according to the gradient of the loss function at the point → **Gradient descent**

Gradient descent

- An illustrative example
  - Suppose we have **an Artificial Neuron F**
    - With only **one weight parameter w**
  - Given a training sample (x, y)
    - The prediction is $\hat{y} = F(x, w)$
  - Use the pre-defined **loss function L**
    - The current loss is $L\big(y, F(x, w)\big) = L(x, y, w)$
    - **The gradient of L at w** is $\nabla w = \frac{\partial L}{\partial w}$
  - Update the parameter value
    - $\boldsymbol{w = w - \eta \times \nabla w}$

Take one step according to the gradient of the loss function at the point → **Gradient descent**

- To the **opposite direction** of the gradient

Gradient descent

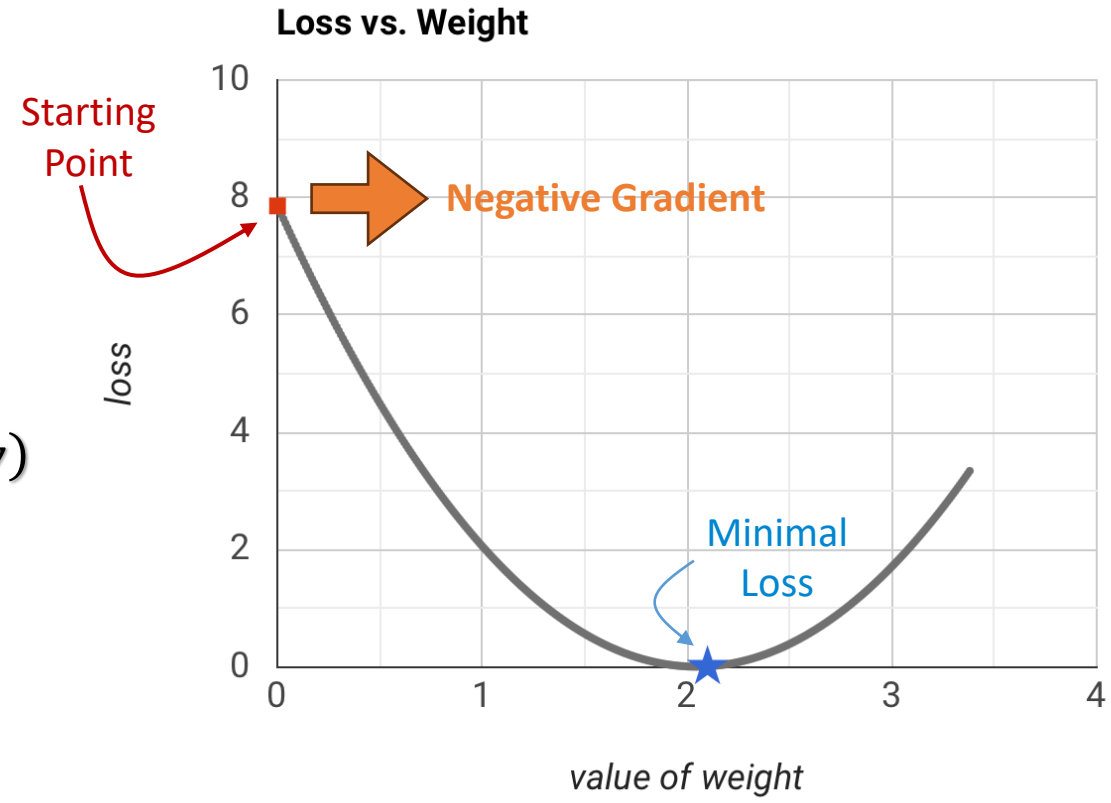
- An illustrative example
  - Suppose we have **an Artificial Neuron F**
    - With only **one weight parameter w**
  - Given a training sample (x, y)
    - The prediction is $\hat{y} = F(x, w)$
  - Use the pre-defined **loss function L**
    - The current loss is $L\big(y, F(x, w)\big) = L(x, y, w)$
    - **The gradient of L at w** is $\nabla w = \frac{\partial L}{\partial w}$
  - Update the parameter value
    - $w = w - \eta \times \nabla w$

Take one step according to the gradient of the loss function at the point → **Gradient descent**

- To the **opposite direction** of the gradient
- The **step size** is determined by:
  - The **magnitude of the gradient**: $|\nabla w|$
  - The **learning rate η** to scale the step size

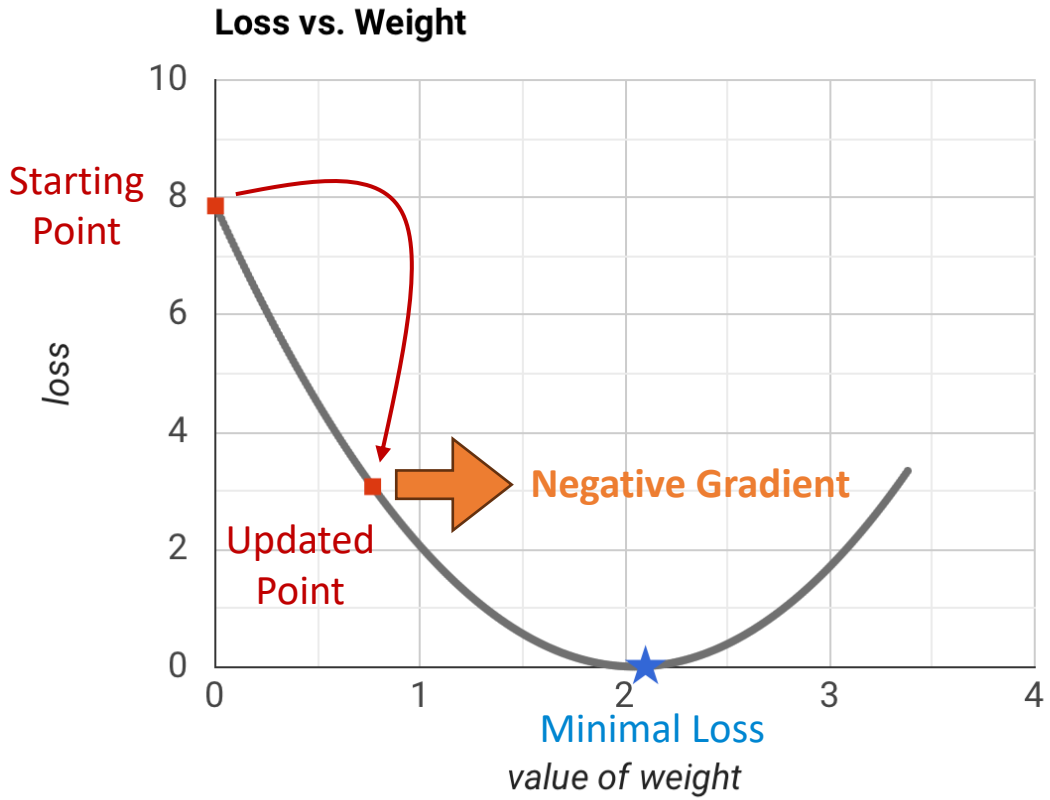
Gradient descent

- An illustrative example
  - Suppose we have **an Artificial Neuron F**
    - With only **one weight parameter w**
  - Given a training sample (x, y)
    - The prediction is $\hat{y} = F(x, w)$
  - Use the pre-defined **loss function L**
    - The current loss is $L\big(y, F(x, w)\big) = L(x, y, w)$
    - **The gradient of L at w** is $\nabla w = \frac{\partial L}{\partial w}$
  - Update the parameter value
    - $\boldsymbol{w = w - \eta \times \nabla w}$



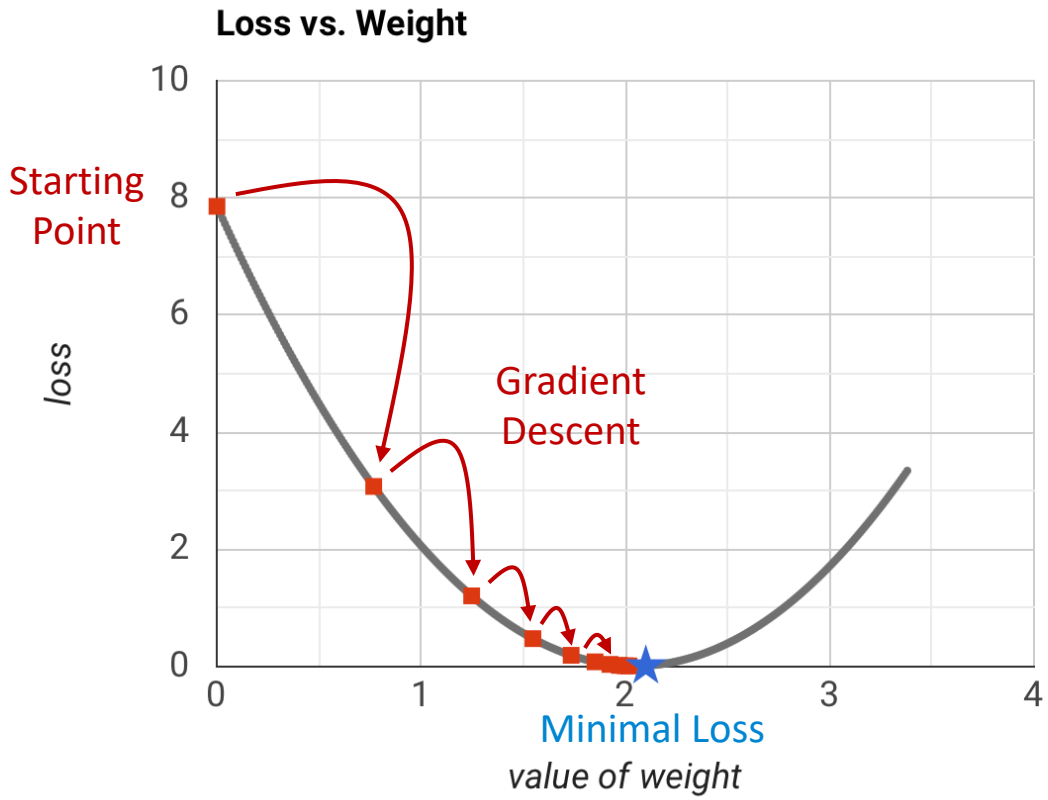
Gradient descent

- An illustrative example
  - Suppose we have **an Artificial Neuron F**
    - With only **one weight parameter w**
  - Given a training sample (x, y)
    - The prediction is $\hat{y} = F(x, w)$
  - Use the pre-defined **loss function L**
    - The current loss is $L\big(y, F(x, w)\big) = L(x, y, w)$
    - **The gradient of L at w** is $\nabla w = \frac{\partial L}{\partial w}$
  - Update the parameter value
    - $\boldsymbol{w = w - \eta \times \nabla w}$



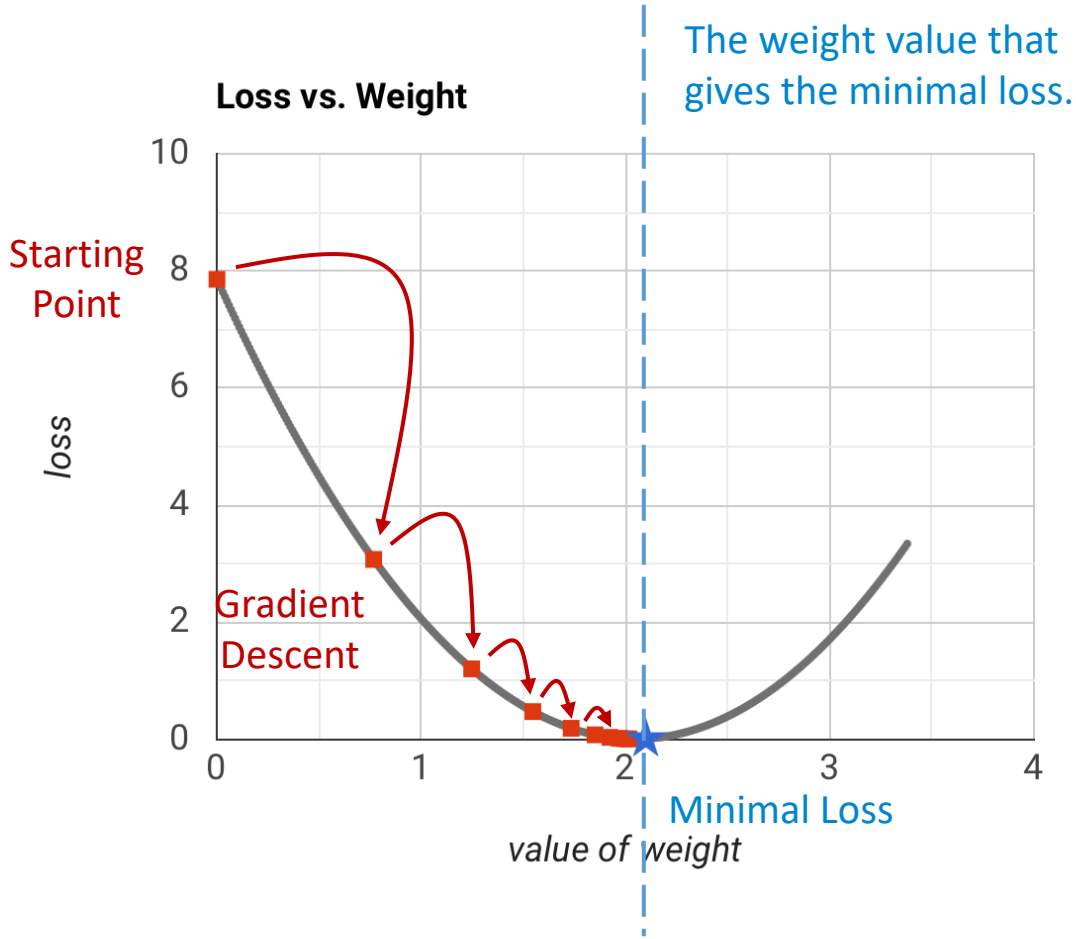
Gradient descent

- An illustrative example
  - Suppose we have **an Artificial Neuron F**
    - With only **one weight parameter w**
  - Given a training sample (x, y)
    - The prediction is $\hat{y} = F(x, w)$
  - Use the pre-defined **loss function L**
    - The current loss is $L\big(y, F(x, w)\big) = L(x, y, w)$
    - **The gradient of L at w** is $\nabla w = \frac{\partial L}{\partial w}$
  - Update the parameter value
    - $\boldsymbol{w = w - \eta \times \nabla w}$



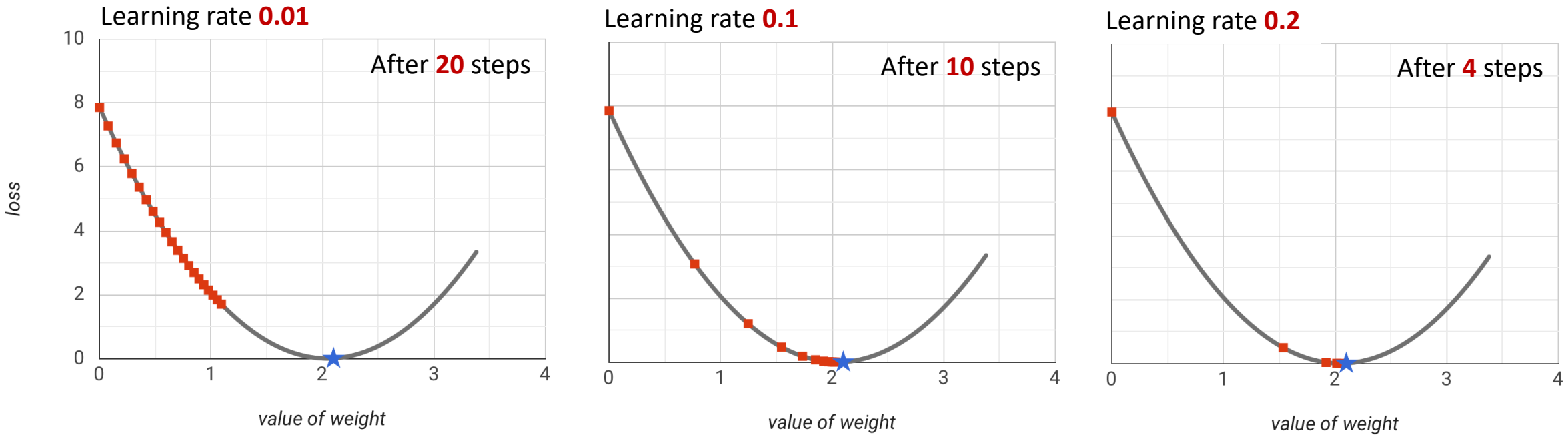
Gradient descent

- An illustrative example
  - Update the parameter value:
    - $w = w - \eta \times \nabla w$
  - **Gradient descent can't find exactly the minimal loss.**
    - As the minimal loss is approached, the gradient ∇w also approaches zero.
      - Even the direction is still correct, the step size approaches zero.
  - **After enough iterations, gradient descent can get close enough to the minimum loss**



Gradient descent

- **Learning rate η affects the step size.**



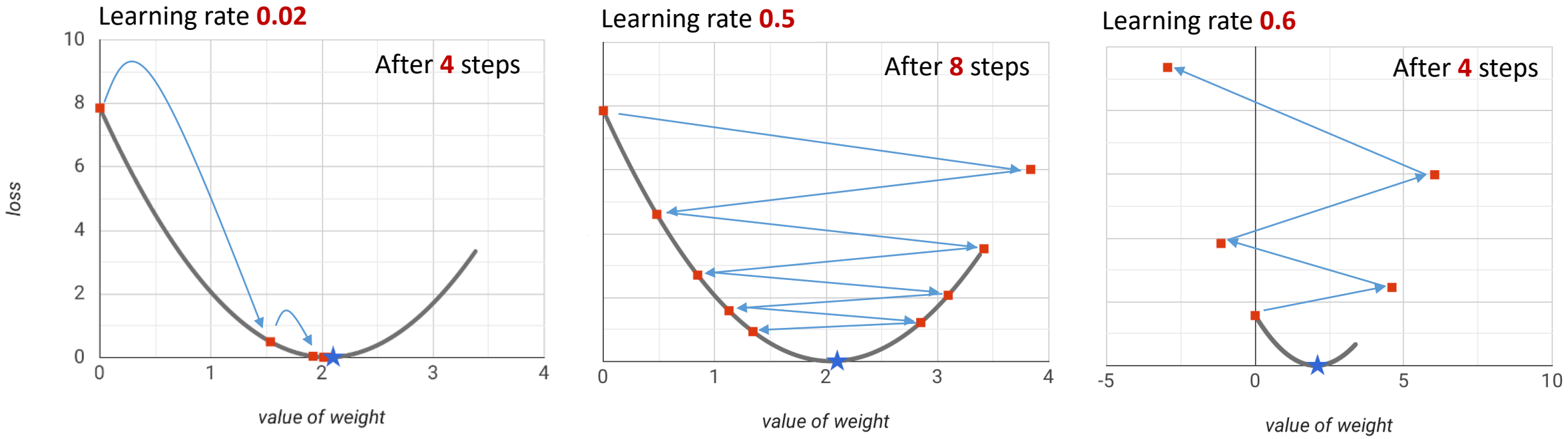
- A small learning rate will lead to more iterations, less efficient learning process.

Hands-on Exercise

- Optimizing Learning Rate:
  - Go to link: https://developers.google.com/machine-learning/crash-course/fitter/graph
  - Finish the 4 tasks:
    1. Set learning rate to 0.03. How many steps it take?
    2. Set learning rate to 0.01. How many steps it take?
    3. Set learning rate to 1, try to reach the minimal loss.
    4. Try to find the Goldilocks learning rate for this curve.
  - 3 minutes

Gradient descent

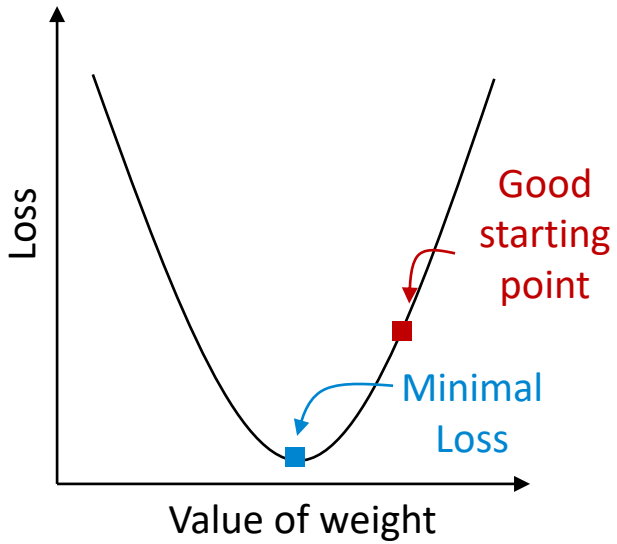
- **Learning rate η affects the step size.**



- A large learning rate may cross the minimal loss
- If learning rate is too large, it never reaches the minimal loss → Divergence

Gradient descent

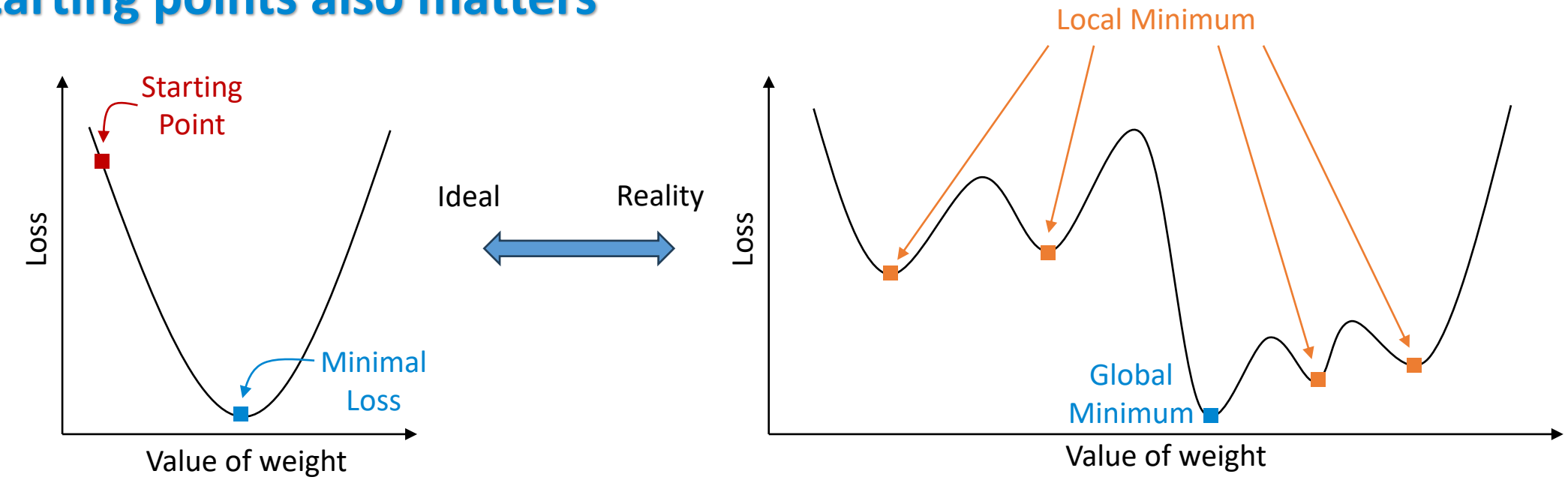
- **Starting points also matters**



- In most cases, the starting point is **randomly selected**.
  - When creating an ANN, the weight and bias parameters are initialized with random values.
  - A **good initialization** has a starting point close to the minimum, thus saving training time.

Gradient descent

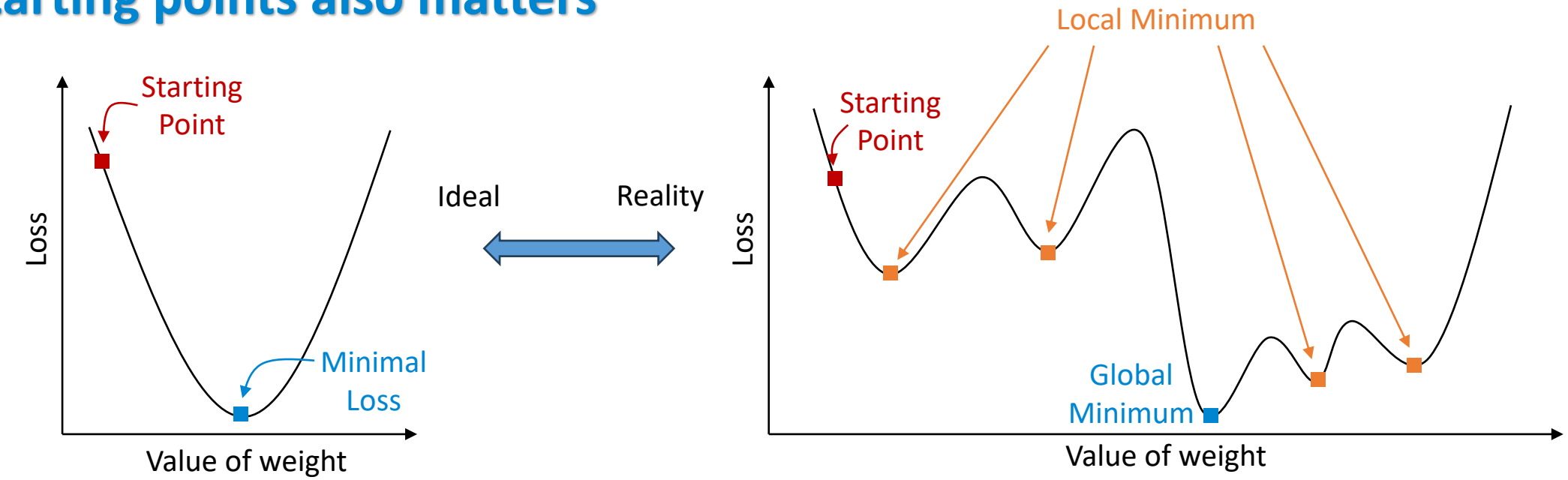
- **Starting points also matters**



- The real function is not a bowl-shape one → not convex
- There can be many **local minimum** and a **global minimum**

Gradient descent

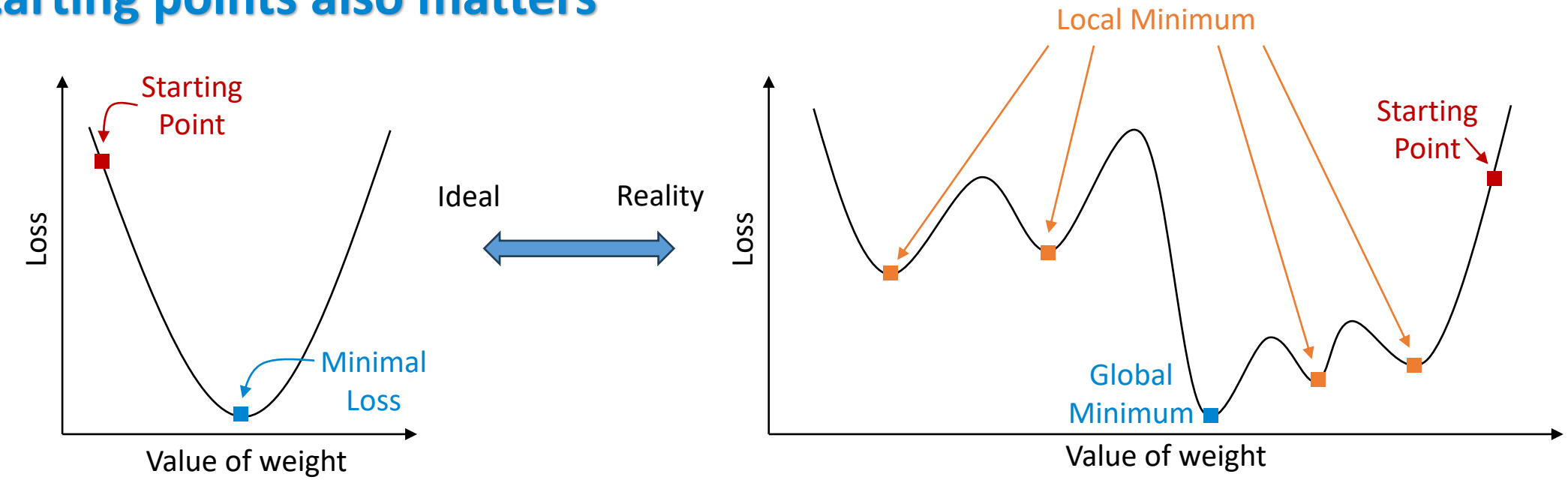
- **Starting points also matters**



- A poor staring point can get gradient descent algorithm trapped around a local minimum, that is much larger than the global minimum
- **Re-initialize the starting point / try a larger learning rate**

Gradient descent

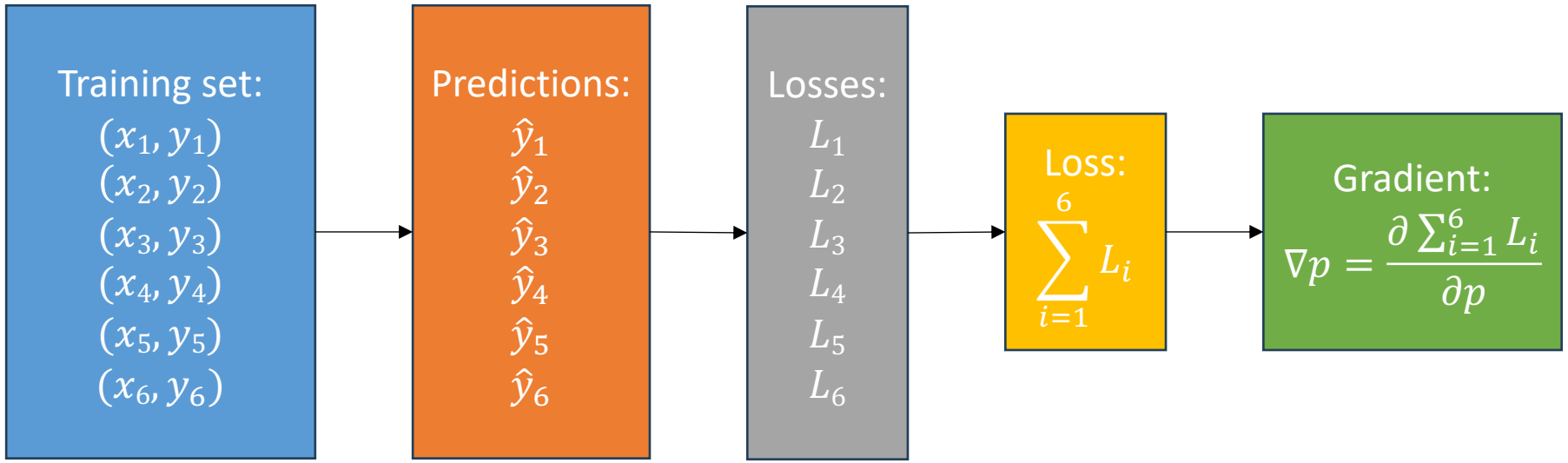
- **Starting points also matters**



- A good staring point can lead to the global minimum, or a local minimum whose value is close to the global minimum
- **We don't pursue the global minimum, as long as the local minimum is good enough**

Gradient descent

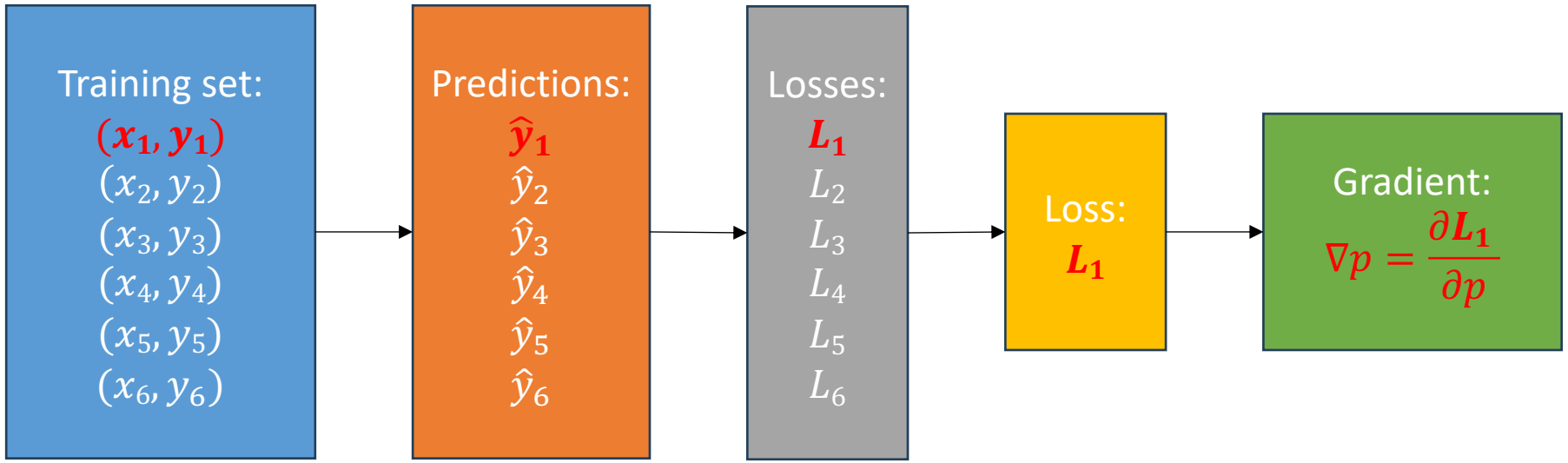
- Gradient descent
  - Use all data samples in the training set to compute the gradient in each iteration



  - More stable, but computationally expensive
  - Usually used in simple neural networks with less data

Gradient descent

- **Stochastic gradient descent**
  - Use a single sample in the training set to compute the gradient in each iteration

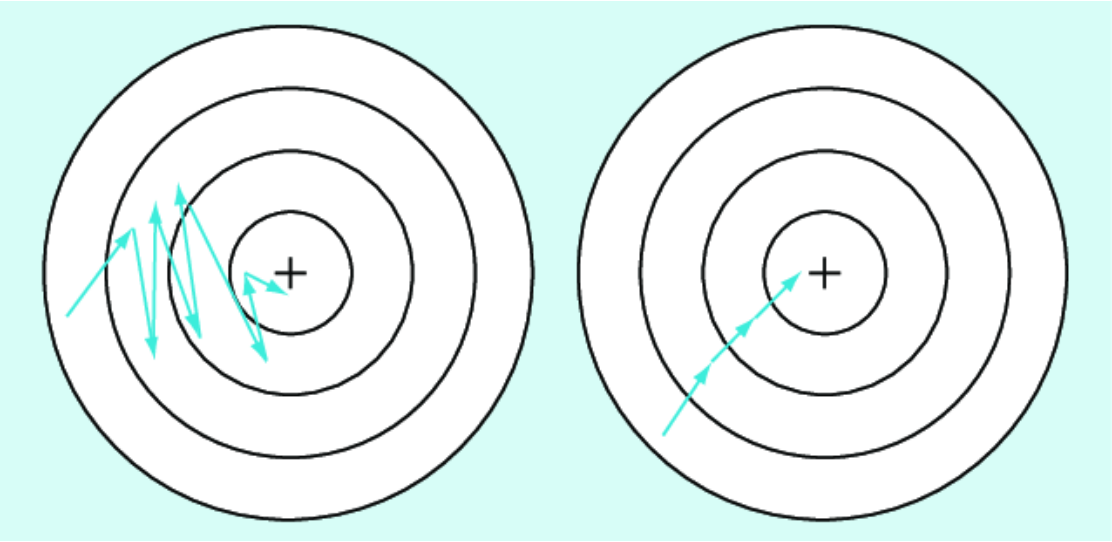


  - The single sample is chosen randomly
  - Faster and more efficient for each iteration

Gradient descent

- **Stochastic gradient descent**
  - Use a single sample in the training set to compute the gradient in each iteration
    - The randomness may lead less accurate gradient directions
    - Although each iteration is faster, it may require more iterations for optimization

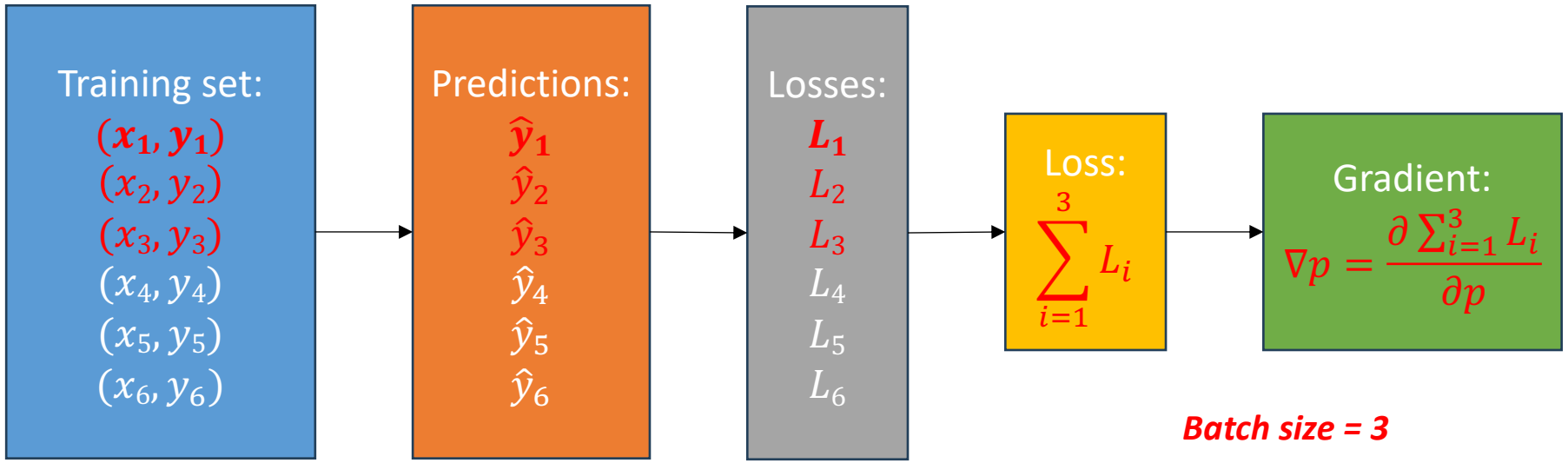
Stochastic Gradient Descent



Gradient Descent

Gradient descent

- **Mini-batch gradient descent**
  - Use a batch of training samples to compute the gradient in each iteration



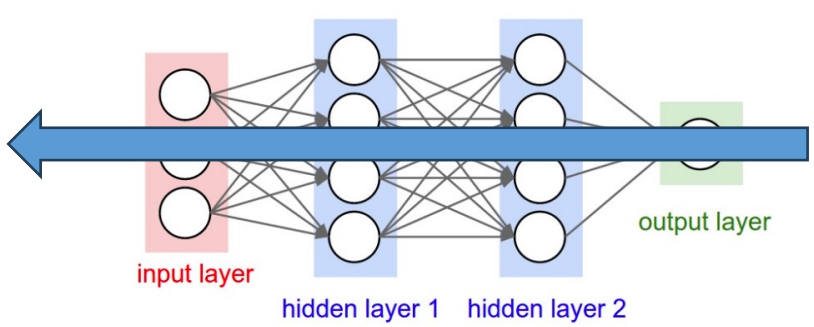
  - Reduce the randomness in SGD, trade-off between speed and stability
  - The batch size is usually 2^n, smaller is faster but might lead to a local minimal

Outline

- Gradient descent
- **Backpropagation**
- Implement 1st ANN for regression

Backpropagation

- For a single artificial neuron:
  - Given a loss function L, parameters can be updated by gradient descent:
    - Weight: $w = w - \eta \times \nabla w = w - \eta \times \frac{\partial L}{\partial w}$
    - Bias: $b = b - \eta \times \nabla b = b - \eta \times \frac{\partial L}{\partial b}$
- For a neural network with multiple artificial neuron:
  - Update parameters by **backpropagation**
    - Gradient descent + **Chain rule**
    - Compute gradients one neuron at a time
      - In practice, **one layer at a time** using matrix operations
    - Iterating backward from the last layer to the input layer



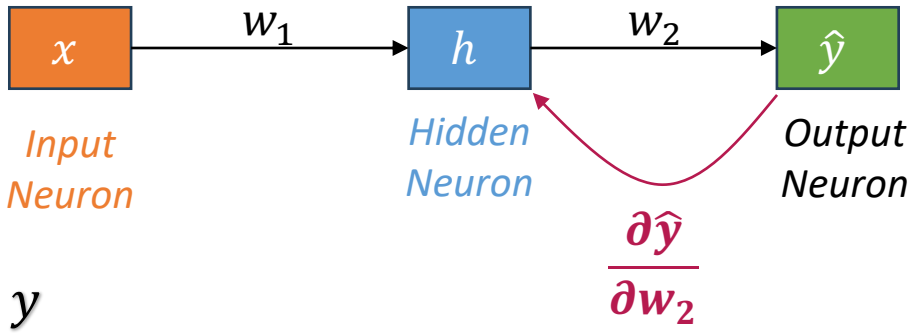
Backpropagation

- Chain rule:
  - Suppose for the function $y = f\big(g(x)\big)$:
    - Two underlying functions:
      - $y = f(u)$
      - $u = g(x)$
    - The chain rule states that:
      - $\frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}$

Backpropagation

- An illustrative example:
  - Suppose we have a 2-layer ANN:
    - Suppose the bias values are zero
    - $h = w_1 x$
    - $\hat{y} = w_2 h = w_2 w_1 x$

    } *Forward propagation*
    - The loss function $L = \frac{1}{2}(\hat{y} - y)^2, \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$



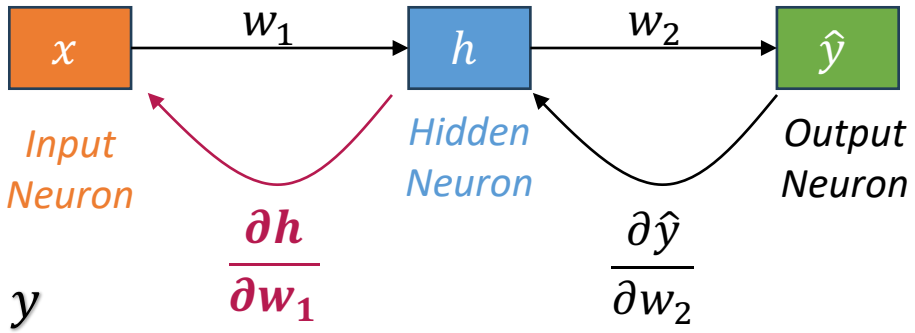
  - Backpropagation:
    - $w_2 \leftarrow w_2 - \eta \nabla w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} = w_2 - \eta \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} = w_2 - \eta (\hat{y} - y) \frac{\partial \hat{y}}{\partial w_2}$

To update w_2, instead of computing ∇w_2, only $\frac{\partial \hat{y}}{\partial w_2}$ needs to be computed

Already knew, stored in the memory

Backpropagation

- An illustrative example :
  - Suppose we have a 2-layer ANN:
    - Suppose the bias values are zero
    - $h = w_1 x$
    - $\hat{y} = w_2 h = w_2 w_1 x$ } *Forward propagation*
    - The loss function $L = \frac{1}{2}(\hat{y} - y)^2, \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$



  - Backpropagation:
    - $w_2 \leftarrow w_2 - \eta \nabla w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} = w_2 - \eta \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} = w_2 - \eta(\hat{y} - y) \frac{\partial \hat{y}}{\partial w_2}$
    - $w_1 \leftarrow w_1 - \eta \boldsymbol{\nabla w_1} = w_1 - \eta \frac{\partial L}{\partial w_1} = w_1 - \eta \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \boldsymbol{\frac{\partial h}{\partial w_1}} = w_1 - \eta(\hat{y} - y) w_2 \boldsymbol{\frac{\partial h}{\partial w_1}}$

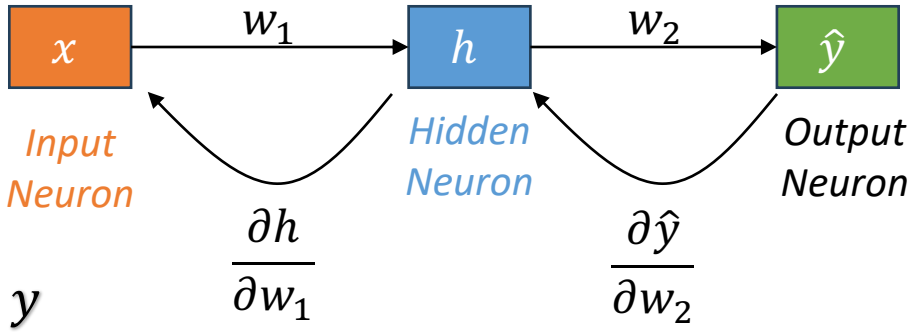
To update w_1, instead of computing ∇w_1, only $\frac{\partial h}{\partial w_1}$ needs to be computed

Already knew, stored in the memory

Backpropagation

- An illustrative example :
  - Suppose we have a 2-layer ANN:
    - Suppose the bias values are zero
    - $h = w_1 x$
    - $\hat{y} = w_2 h = w_2 w_1 x$ } *Forward propagation*
    - The loss function $L = \frac{1}{2}(\hat{y} - y)^2, \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$
  - Backpropagation:
    - $w_2 \leftarrow w_2 - \eta \nabla w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} = w_2 - \eta \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} = w_2 - \eta (\hat{y} - y) \frac{\partial \hat{y}}{\partial w_2}$
    - $w_1 \leftarrow w_1 - \eta \nabla w_1 = w_1 - \eta \frac{\partial L}{\partial w_1} = w_1 - \eta \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1} = w_1 - \eta (\hat{y} - y) w_2 \frac{\partial h}{\partial w_1}$

To update the weight of a connection between two layers, only the partial derivative between these two layers needs to be computed

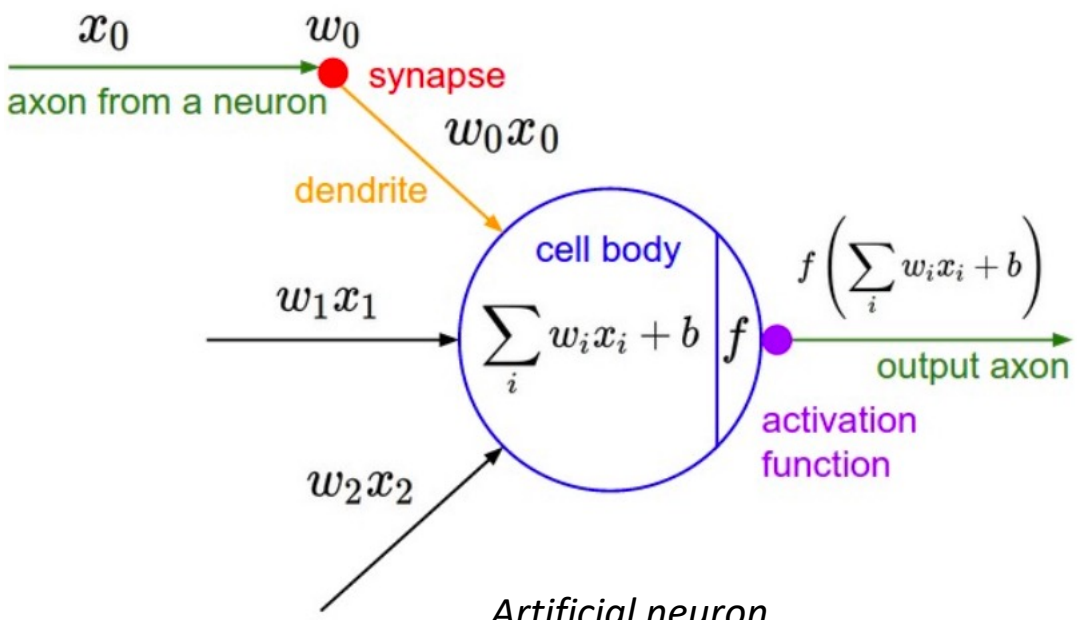


Outline

- Gradient descent
- Backpropagation
- **Implement 1st ANN for regression**

Implement 1st ANN for regression

- Use a single artificial neuron as a linear regression model
  - If the activation function of the artificial neuron is linear:
    - $f(z) = z$
  - Then the output of this artificial neuron is:
    - $\hat{y} = \sum_i w_i x_i + b$
    - The same as a linear regression model



Artificial neuron

Implement 1st ANN for regression

- Implementation using PyTorch
  1. Build the data pipeline
  2. Create the artificial neural network
  3. Training by gradient descent
  4. Save and load a trained model
  5. Make predictions and evaluation

Hands-on Exercise

- Exercise 04 ANN for Regression – Instruction
  - Build a one-layer ANN as a linear regression model
  - Try to add more layers to get a multi-layer ANN
  - Try to introduce non-linearity into the ANN
- Exercise 04 ANN for Regression - Assignment