Module 05 – Extra Class

MOMENTUM
Gradient Descent with Momentum

Nguyen Quoc Thai

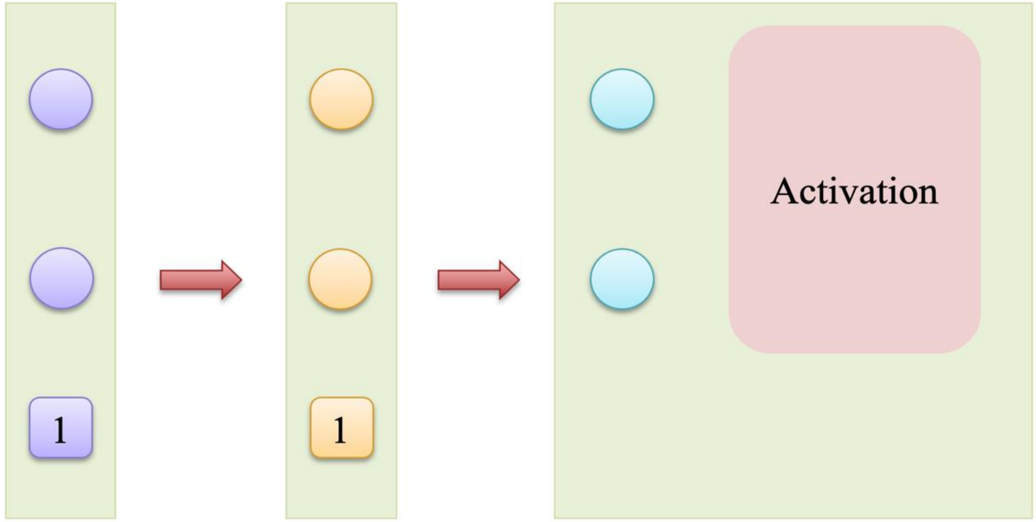
Code - Data

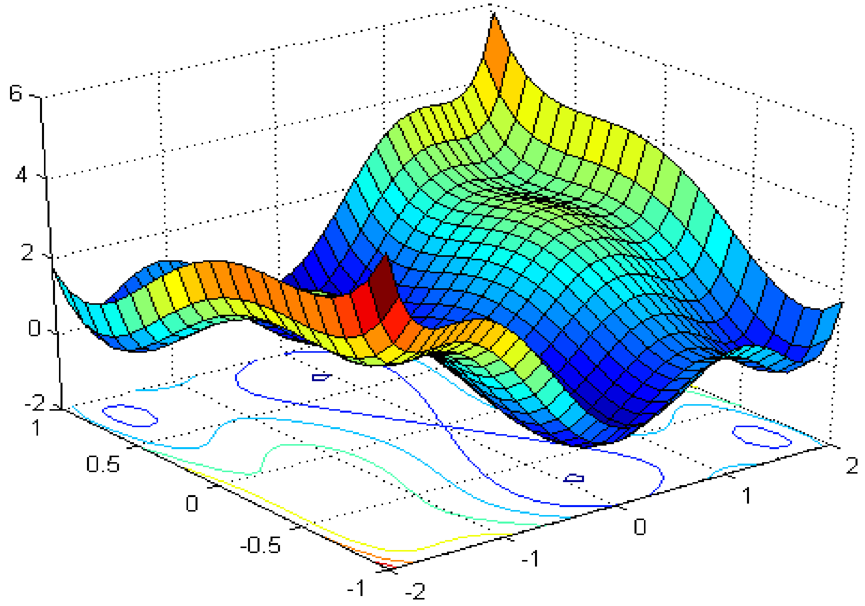
AI VIET NAM
@aivietnam.edu.vn

Objectives

Multi-layer Perceptron (Review)

- Hidden Layers
- Activation, Loss Function, Optimization
- Classification using MLP





Momentum

- Gradient Descent (Review)
- Gradient Descent with Momentum
- Nesterov Momentum

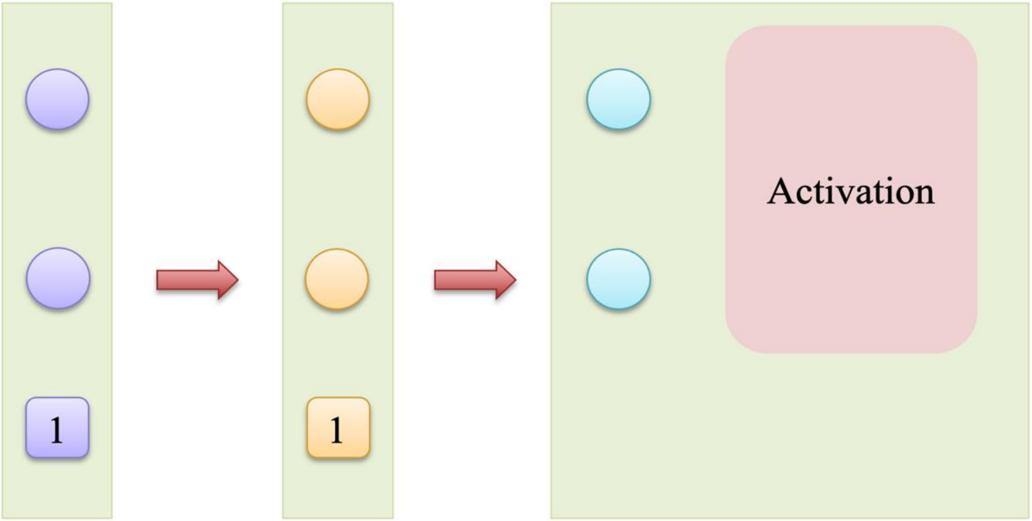
Outline

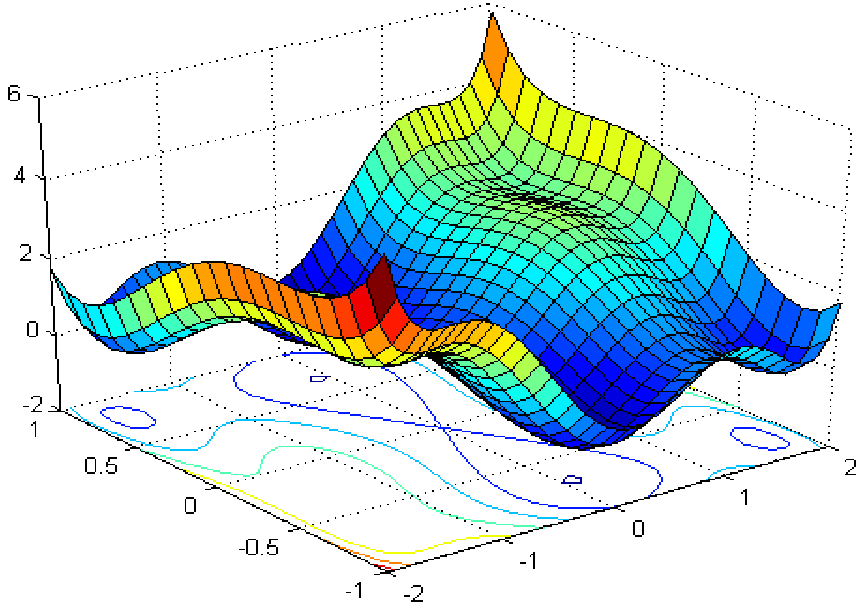
SECTION 1

Multi-layer Perceptron

SECTION 2

Momentum

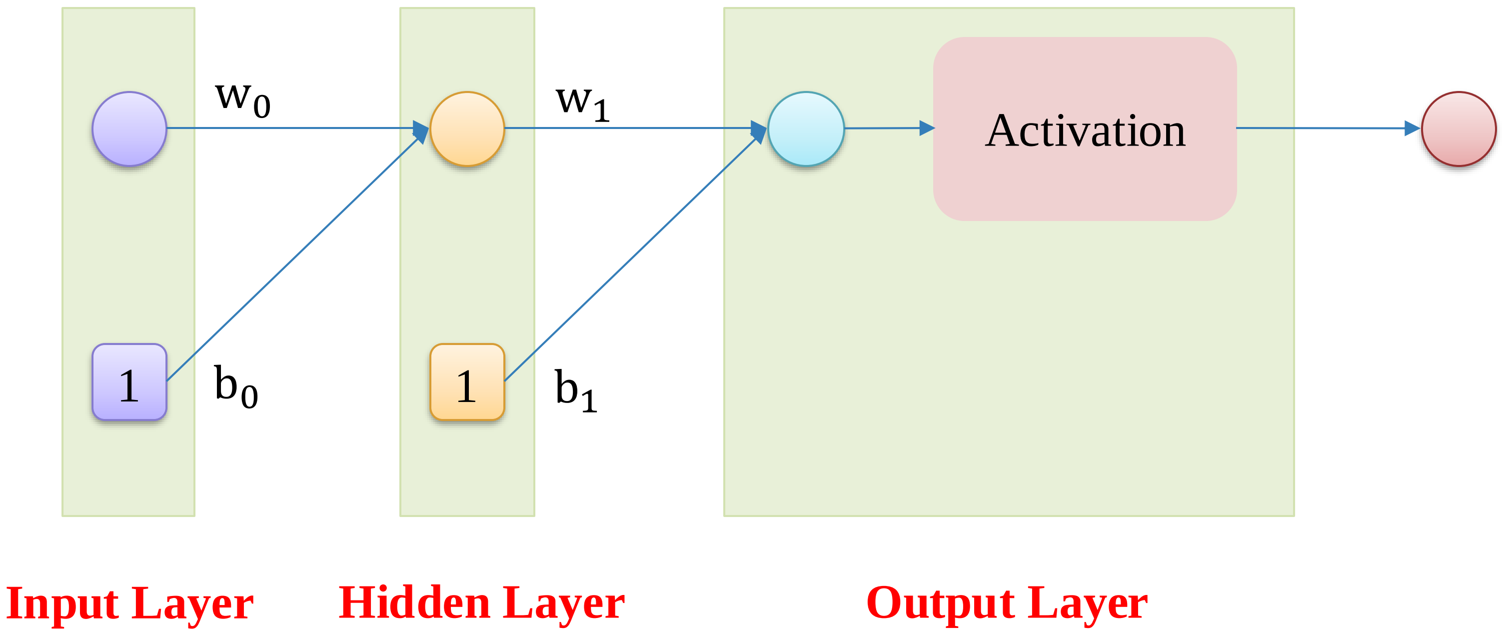




Multi-layer Perceptron

Hidden Layer

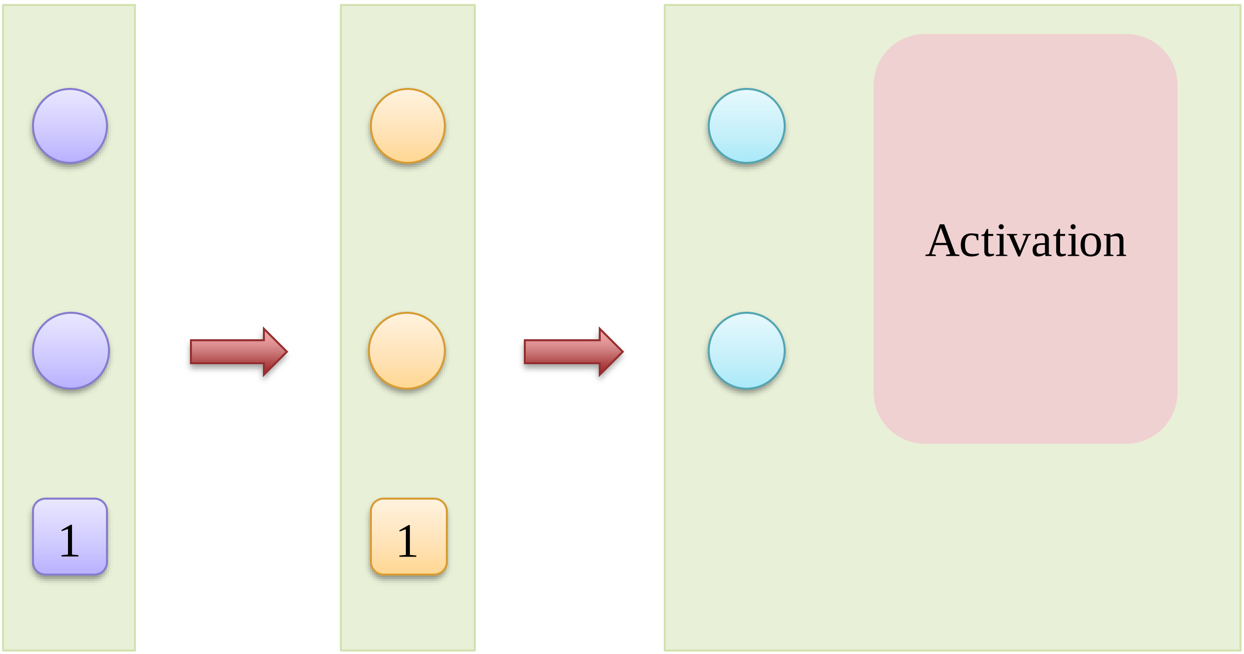
#parameters: 4



Multi-layer Perceptron

Multi-layer Perceptron

#parameters: 12

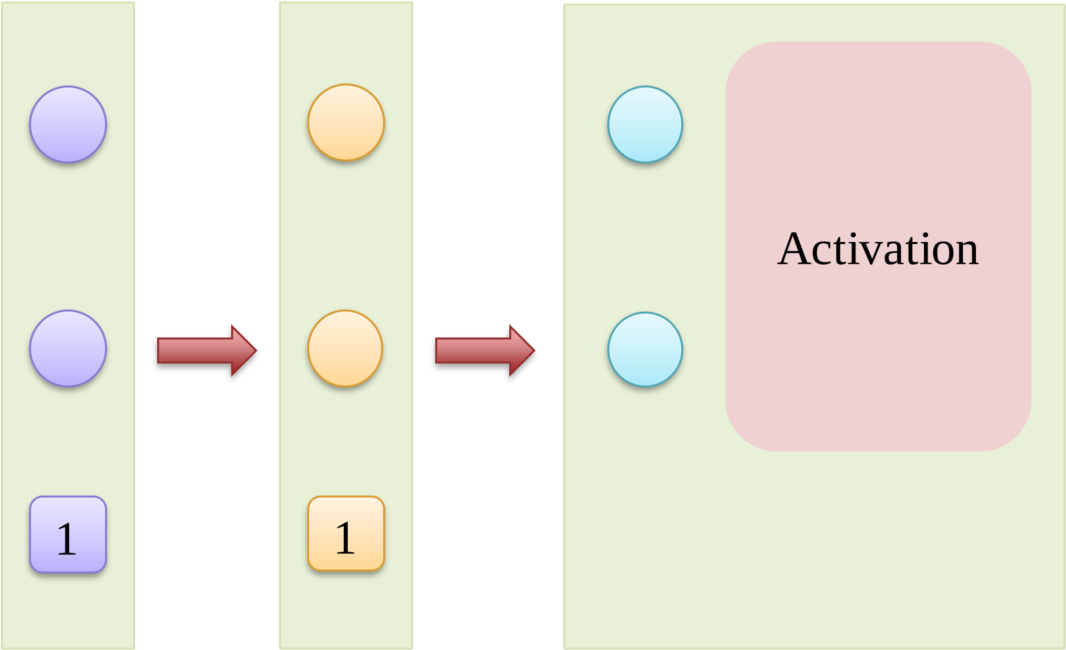


Input Layer   Hidden Layer   Output Layer

Multi-layer Perceptron

Multi-layer Perceptron using Pytorch

#parameters: 12




```
model = nn.Sequential(
    nn.Linear(2, 2),
    nn.Linear(2, 2),
    nn.Sigmoid()
)
```

```
summary(model, (10000000, 2))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1          [-1, 10000000, 2]               6
            Linear-2          [-1, 10000000, 2]               6
           Sigmoid-3          [-1, 10000000, 2]               0
================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 76.29
Forward/backward pass size (MB): 457.76
Params size (MB): 0.00
Estimated Total Size (MB): 534.06
----------------------------------------------------------------
```

Multi-layer Perceptron

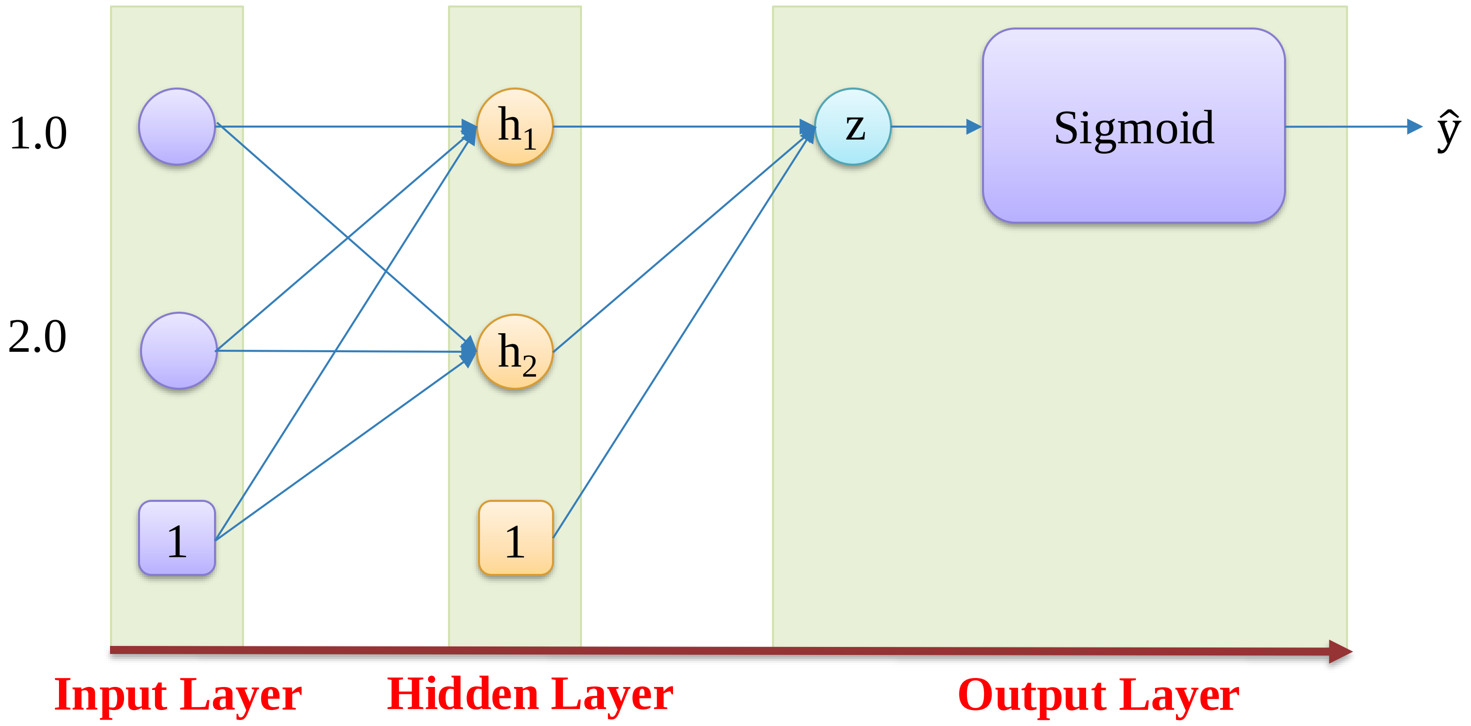
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$$

$$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$



$y = [0]$

Multi-layer Perceptron

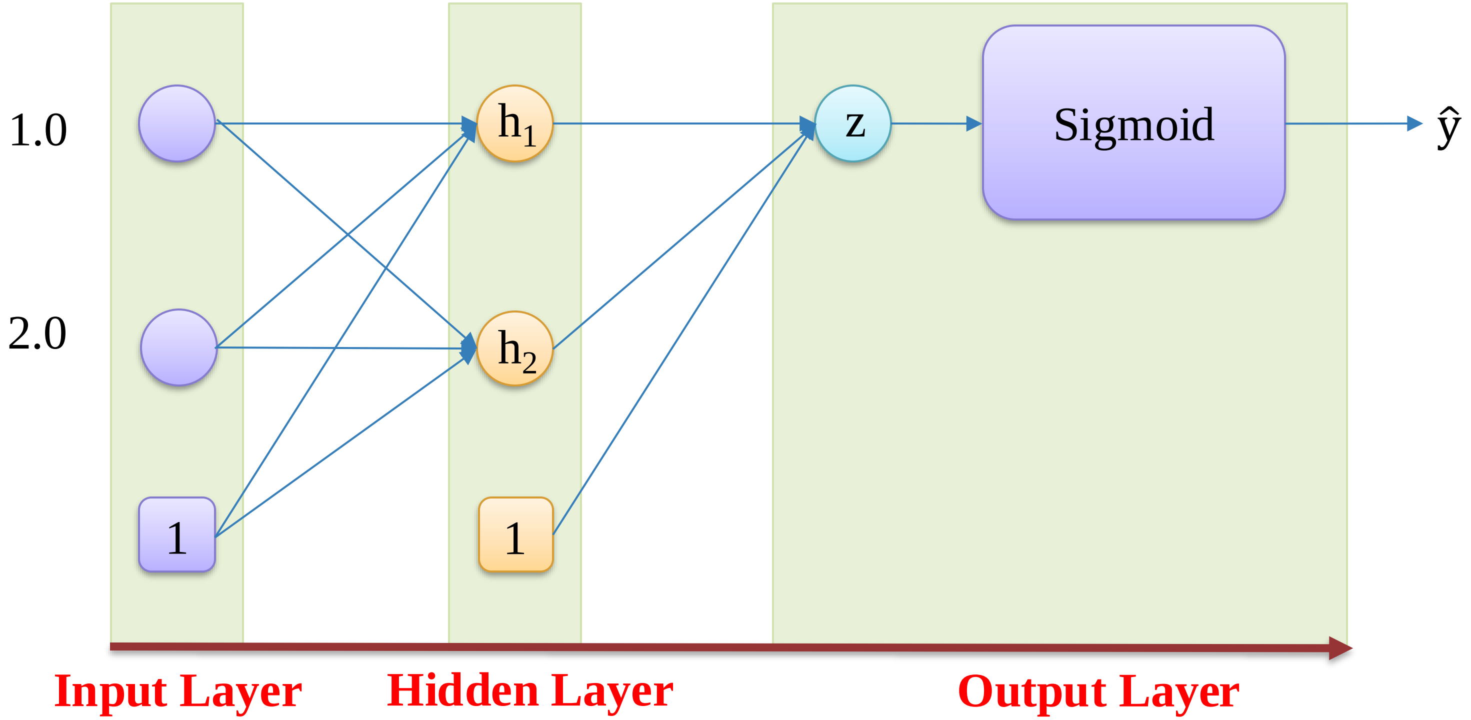
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$y = [0]$

$$h = [1.0\ x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$$

Multi-layer Perceptron

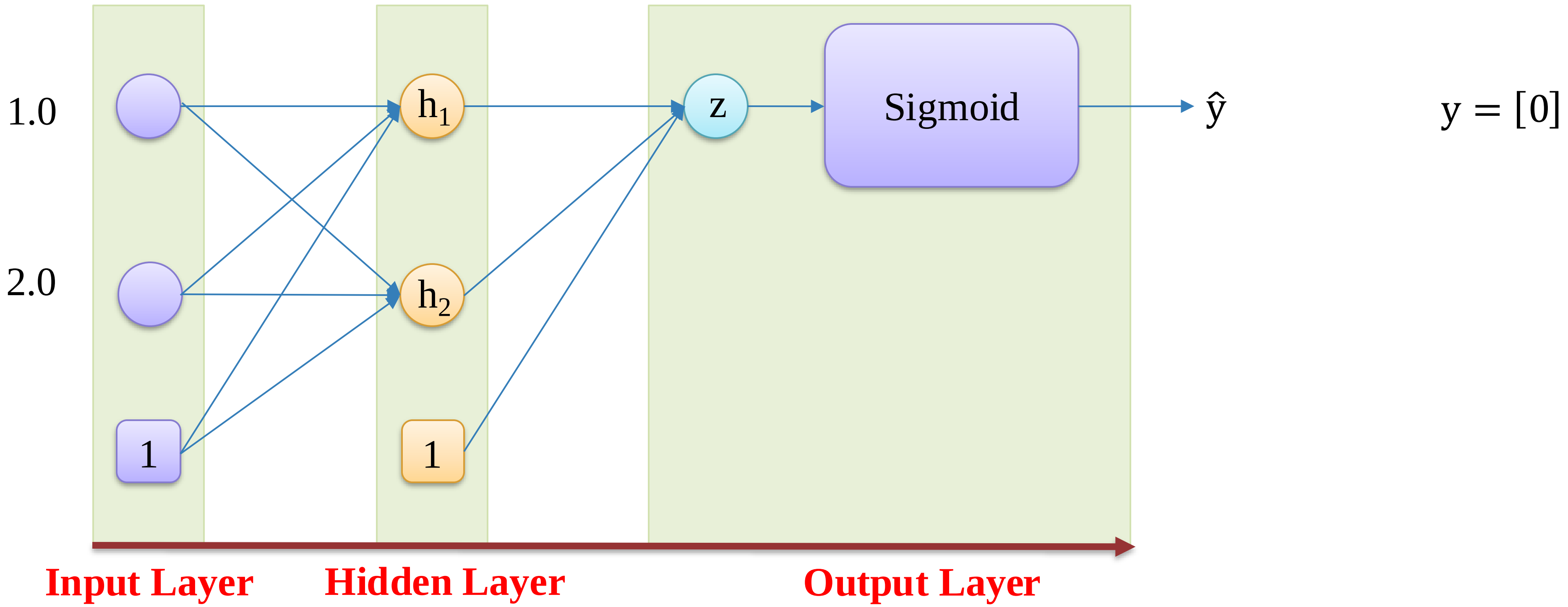
Forward

$x = [1.0 \quad 2.0\,]$

$W_h = [W_{h1} \quad W_{h2}\,]$

$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

$W_z = [W_z\,] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$



$h = [1.0\ x]W_h = [1.0 \quad 1.0 \quad 2.0\,] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4\,]$

$z = [1.0\ h]W_z = [1.0 \quad 0.4 \quad 0.4\,] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.18$

Multi-layer Perceptron

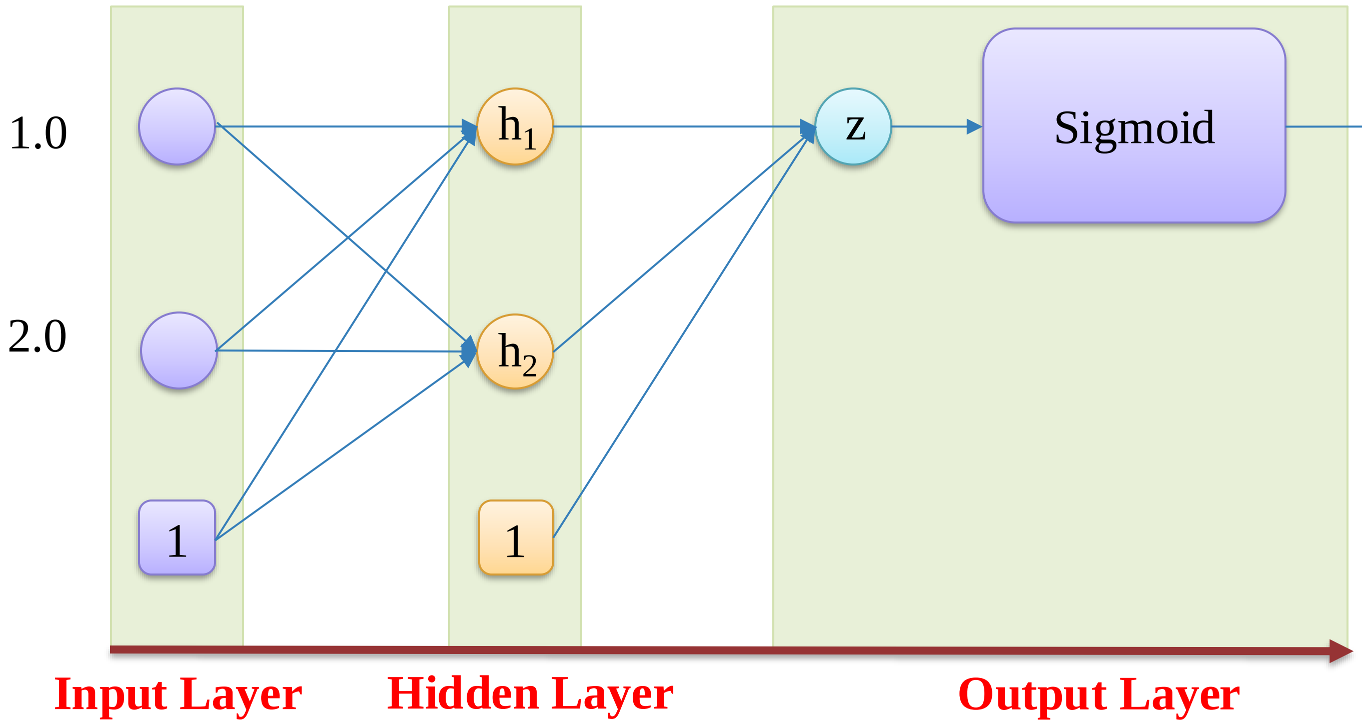
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$$

$$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$



$\hat{y} = [0.5449] \quad y = [0]$

$\hat{y} = \sigma(z)$
$= \sigma(0.18)$
$= 0.5449$

$$h = [1.0\ x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$$

$$z = [1.0\ h]W_z = [1.0 \quad 0.4 \quad 0.4] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.18$$

Multi-layer Perceptron

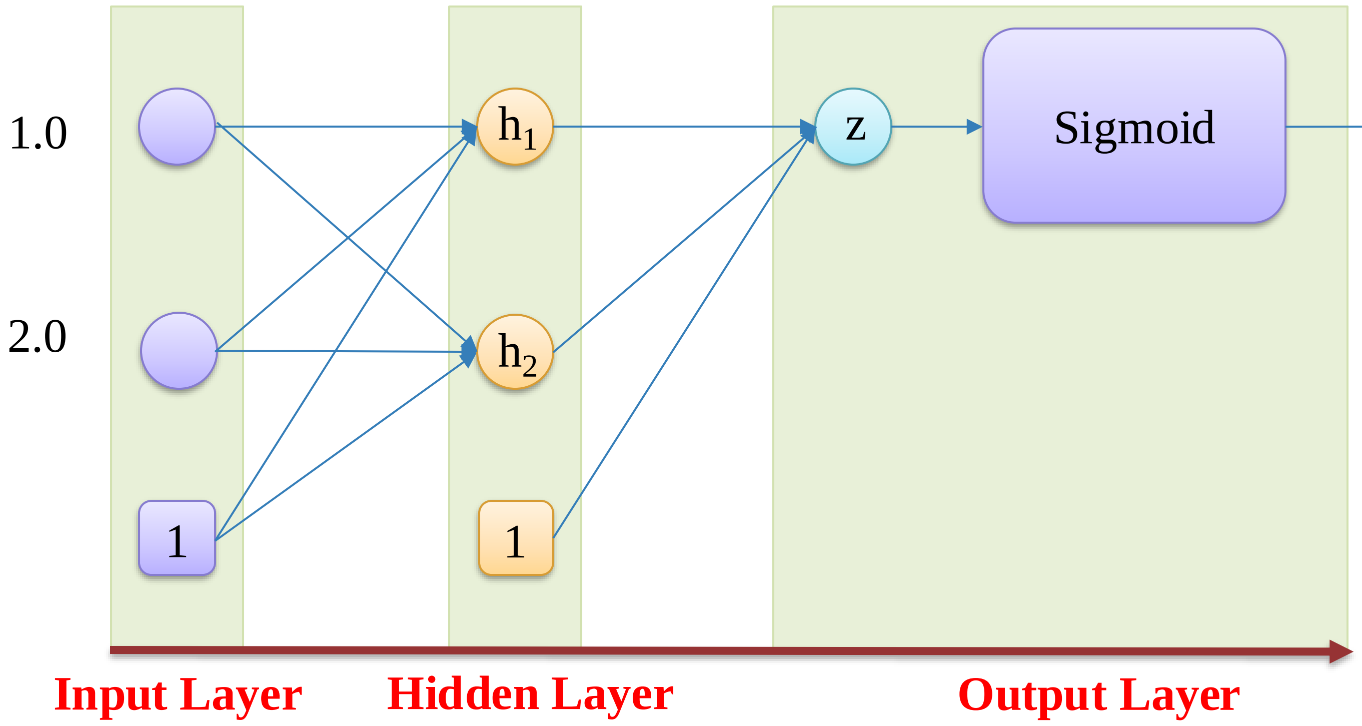
Forward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$$= \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$$

$$W_z = [W_z] = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$



$\hat{y} = [0.5449] \quad y = [0]$

$$\hat{y} = \sigma(z) = \sigma(0.18) = 0.5449$$

$L = 0.7872$

$$h = [1.0\ x]W_h = [1.0 \quad 1.0 \quad 2.0] \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.1 \\ 0.1 & 0.1 \end{bmatrix} = [0.4 \quad 0.4]$$

$$z = [1.0\ h]W_z = [1.0 \quad 0.4 \quad 0.4] \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} = 0.18$$

Multi-layer Perceptron

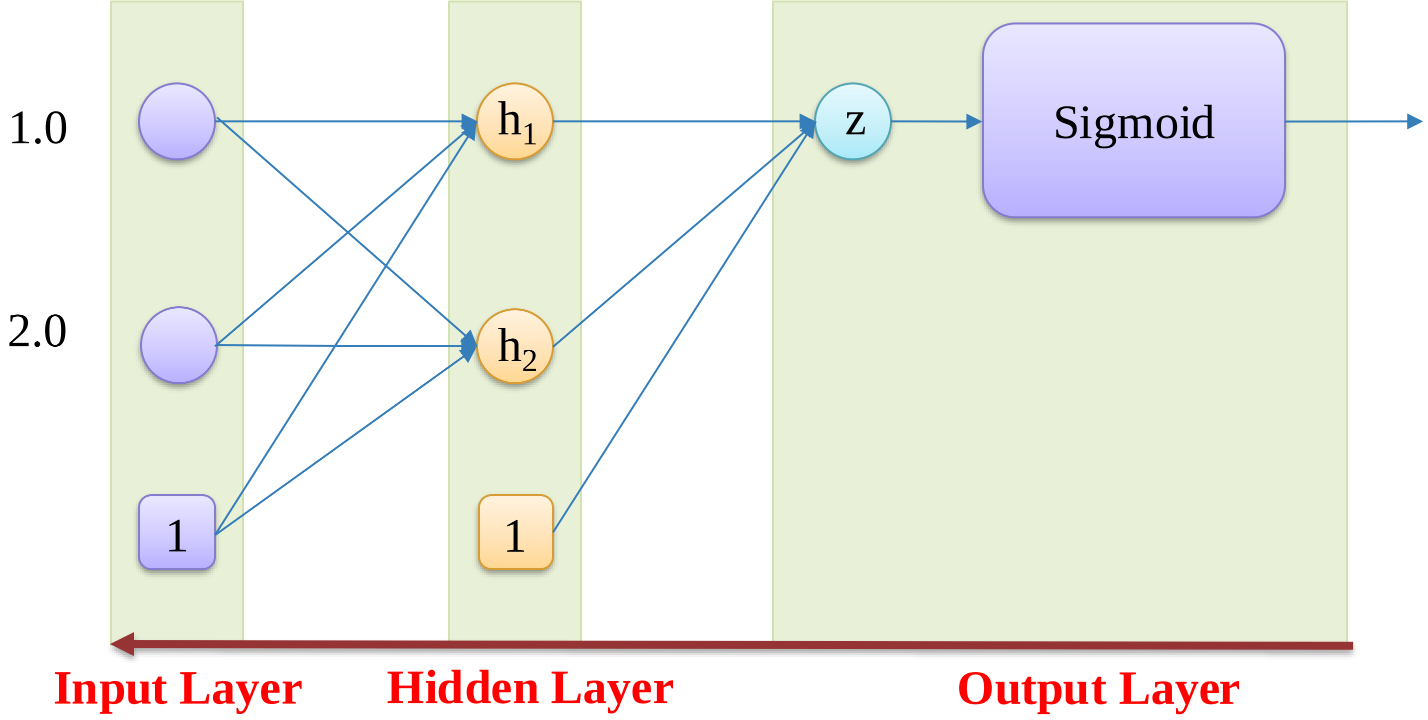
Backward

$x = [1.0 \quad 2.0]$

$W_h = [W_{h1} \quad W_{h2}]$

$$= \begin{bmatrix} 0.0946 & 0.0946 \\ 0.0946 & 0.0891 \\ 0.0946 & 0.0891 \end{bmatrix}$$

$$W_z = [W_z] = \begin{bmatrix} 0.0455 \\ 0.0782 \\ 0.0782 \end{bmatrix}$$



$\hat{y} = [0.5449] \quad y = [0]$

$$\hat{y} = \sigma(z) = \sigma(0.18) = 0.5449$$

$L = 0.7872$

Multi-layer Perceptron

Activation

```python
import torch.nn as nn

act = nn.Sigmoid()
input = torch.tensor([0.18, -0.18])
act(input)
```

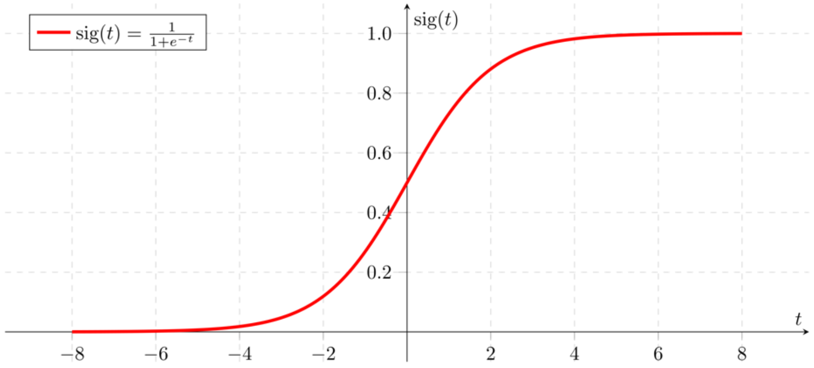
```
tensor([0.5449, 0.4551])
```

```python
import torch.nn as nn

act = nn.ReLU()
input = torch.tensor([0.18, -0.18])
act(input)
```

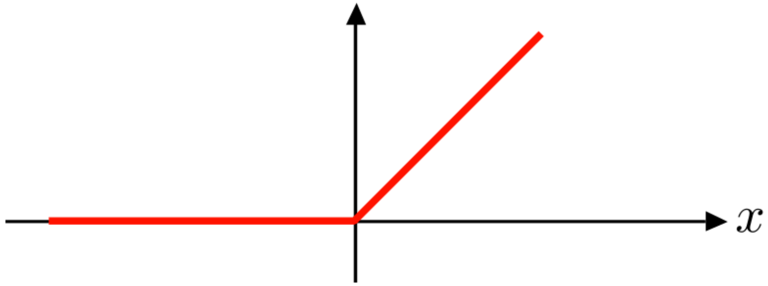
```
tensor([0.1800, 0.0000])
```

```python
import torch.nn as nn

act = nn.Tanh()
input = torch.tensor([0.18, -0.18])
act(input)
```

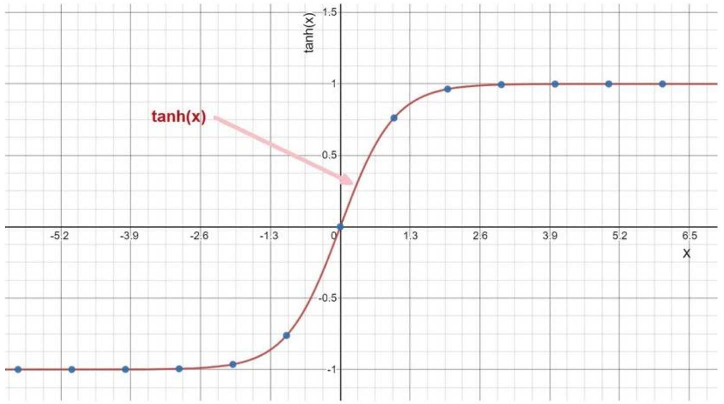
```
tensor([ 0.1781, -0.1781])
```



$$\mathrm{ReLU}(x) \triangleq \max(0, x)$$



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



Multi-layer Perceptron

Loss Function

```
import torch.nn as nn
loss_fn = nn.BCELoss()
```

```
y_pred
```

```
tensor([0.5449], grad_fn=<SigmoidBackward0>)
```

```
y
```

```
tensor([0.])
```

```
loss = loss_fn(y_pred, y)
loss
```

```
tensor(0.7872, grad_fn=<BinaryCrossEntropyBackward0>)
```

```
import torch.nn as nn
loss_fn = nn.CrossEntropyLoss()
```

```
y = torch.tensor(0)
y
```

```
tensor(0)
```

```
y_pred
```

```
tensor([0.0580, 0.4275], grad_fn=<AddBackward0>)
```

```
loss_fn(y_pred, y)
```

```
tensor(0.8949, grad_fn=<NllLossBackward0>)
```

Multi-layer Perceptron

Optimizer (SGD)

```
learning_rate = 0.1
optimizer = optim.SGD(model.parameters(), learning_rate)
```

```
loss.backward()
```

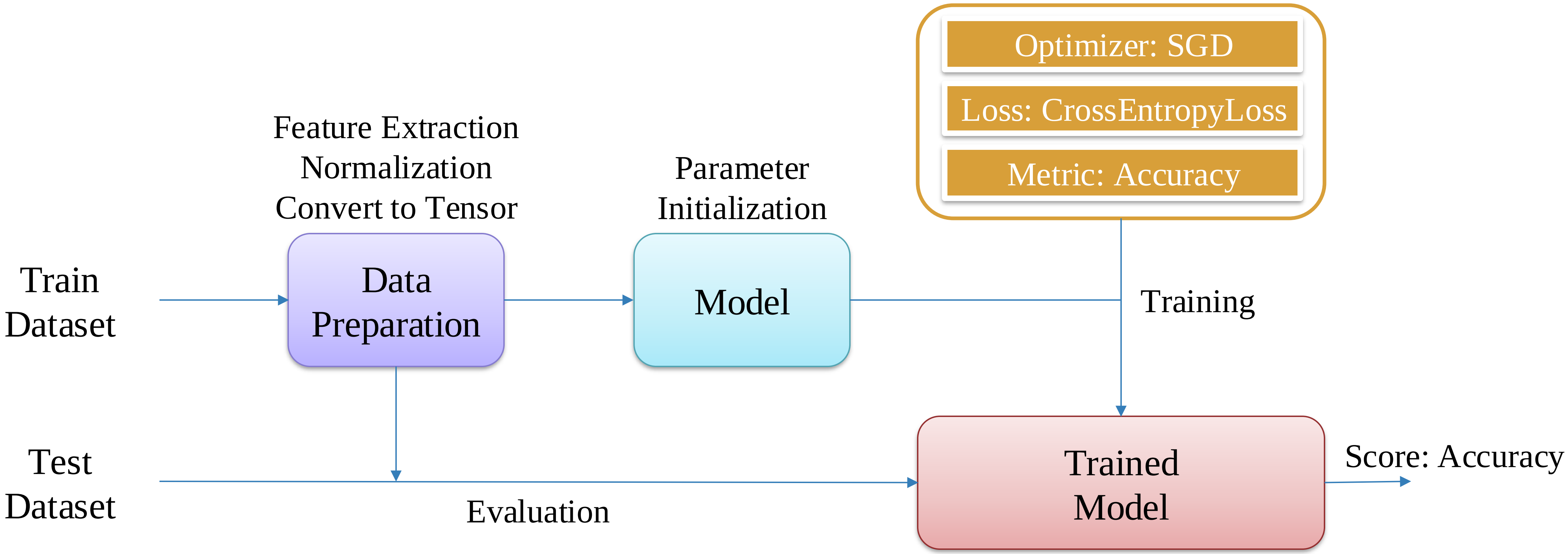
```
optimizer.step()
```

```
for layer in model.children():
    print(layer.state_dict())
```

```
OrderedDict([('weight', tensor([[0.0946, 0.0891],
        [0.0946, 0.0891]])), ('bias', tensor([0.0946, 0.0946]))])
OrderedDict([('weight', tensor([[0.0782, 0.0782]])), ('bias', tensor([0.0455]))])
OrderedDict()
```

Classification using MLP

IRIS Dataset



AI VIET NAM
@aivietnam.edu.vn

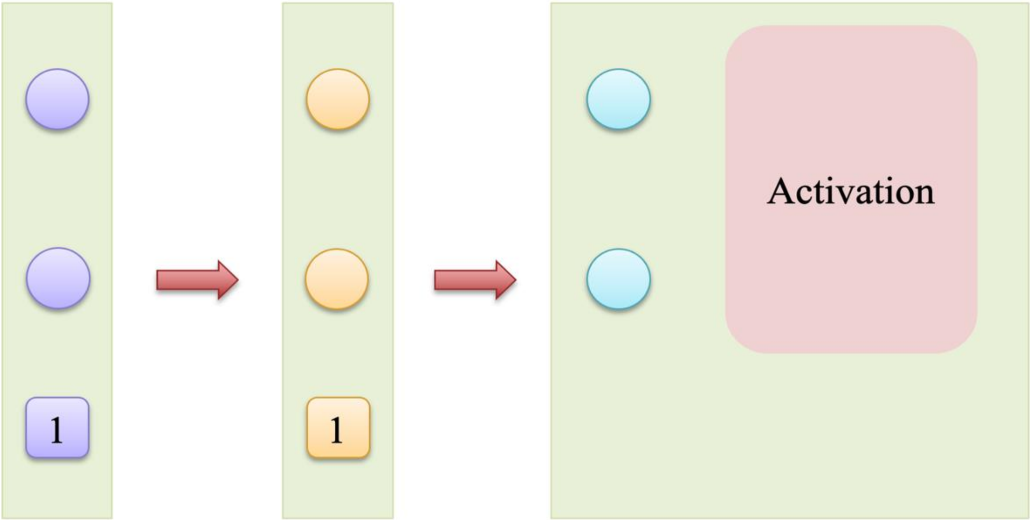
Outline

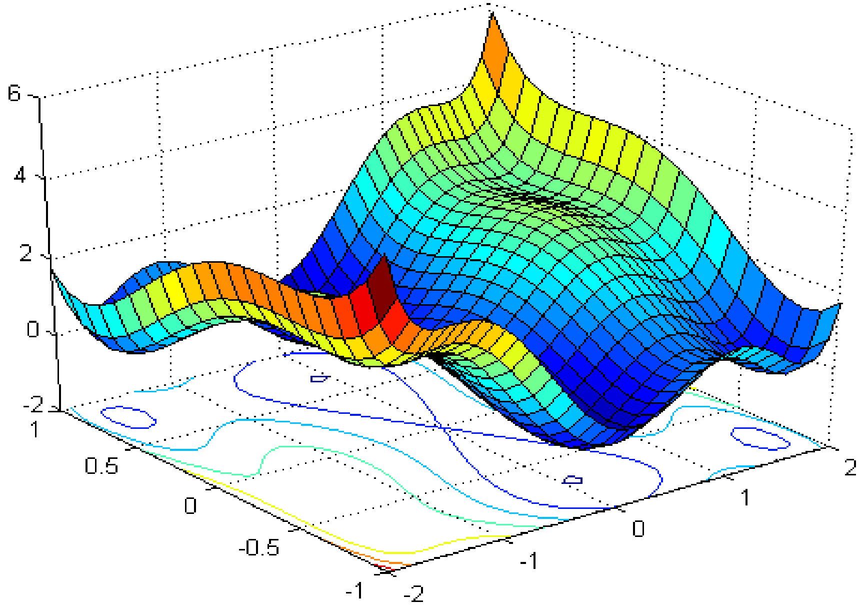
SECTION 1

Multi-layer Perceptron

SECTION 2

Momentum





Momentum

SGD

1) Pick a sample (x, y) from training data

2) Compute output $\hat{y}$

$$\hat{y} = \boldsymbol{\theta}^T \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{\theta}$$

3) Compute loss

$$L = (\hat{y} - y)^2$$

4) Compute derivative

$$\nabla_\theta \mathrm{L} = 2\boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_\theta \mathrm{L}$$

η is learning rate

1) Pick a sample (x, y) from training

2) Compute output $\hat{y}$

$$z = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

3) Compute loss

$$L(\boldsymbol{\theta}) = (-\mathrm{y}\log\hat{\mathrm{y}} - (1-\mathrm{y})\log(1-\hat{\mathrm{y}}))$$

4) Compute derivative

$$\nabla_\theta \mathrm{L} = \boldsymbol{x}(\hat{y} - y)$$

5) Update parameters

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_\theta \mathrm{L}$$

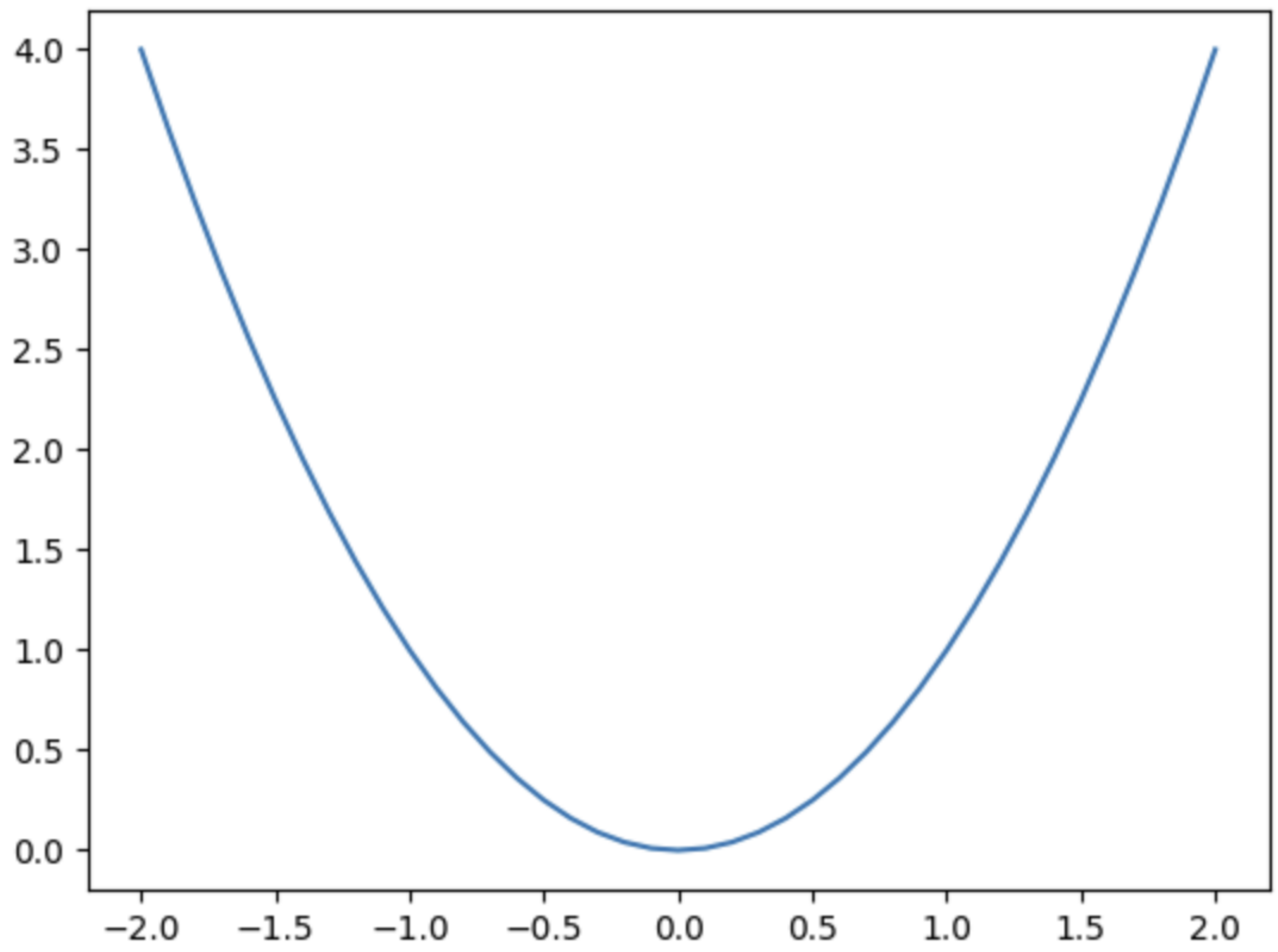
η is learning rate

Momentum

SGD

- Objective function: x^2
- Derivative: $2x$
- Gradient:

$$x = x - \eta * f'(x)$$



Momentum

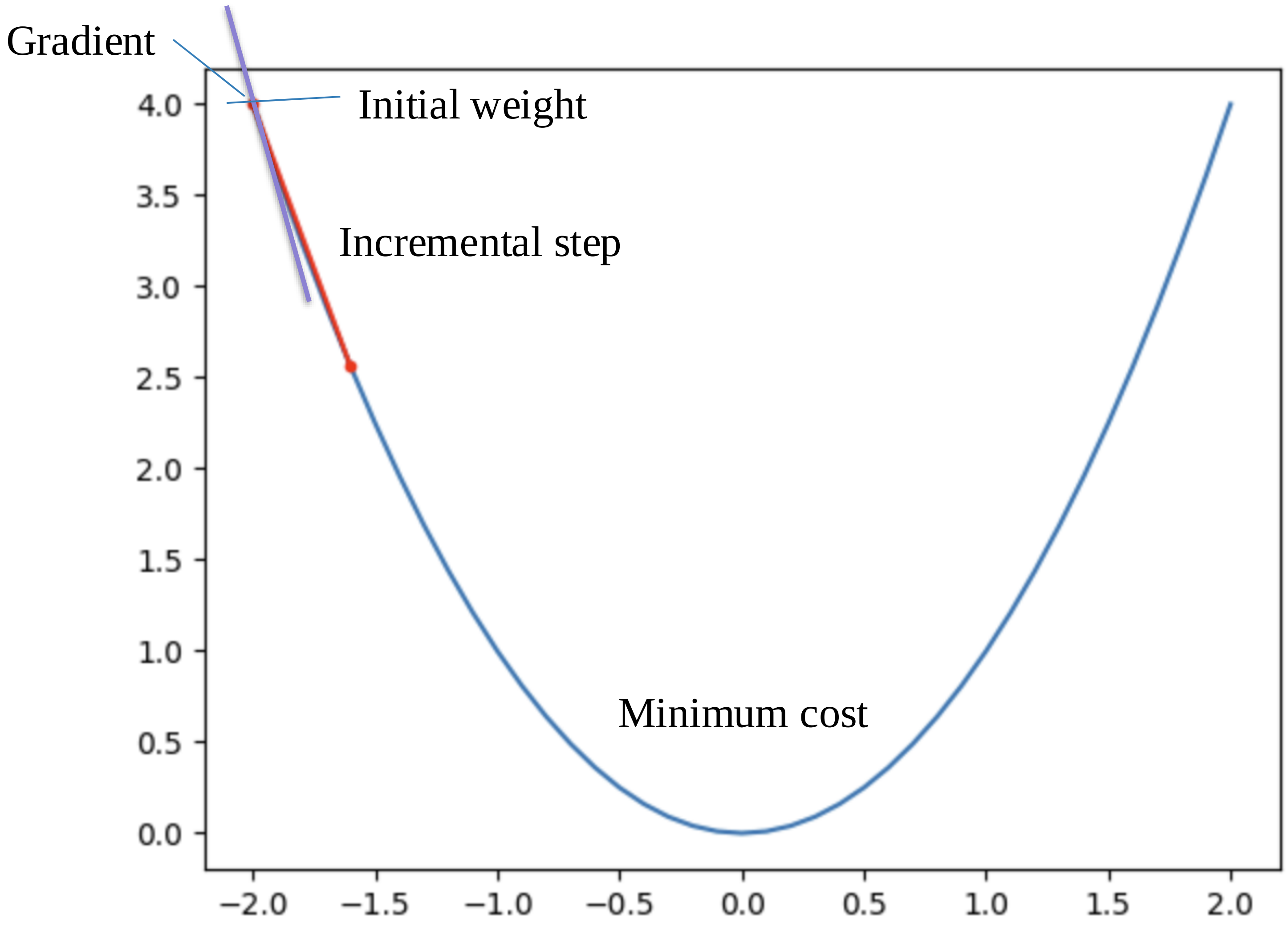
SGD

- Objective function: x^2
- Derivative: $2x$
- Gradient:

$$x = x - \eta * f'(x)$$

$$x = -2.0$$
$$\eta = 0.1$$

$$x = -2.0 - 0.1 * (2 * -2.0)$$
$$= -1.6$$

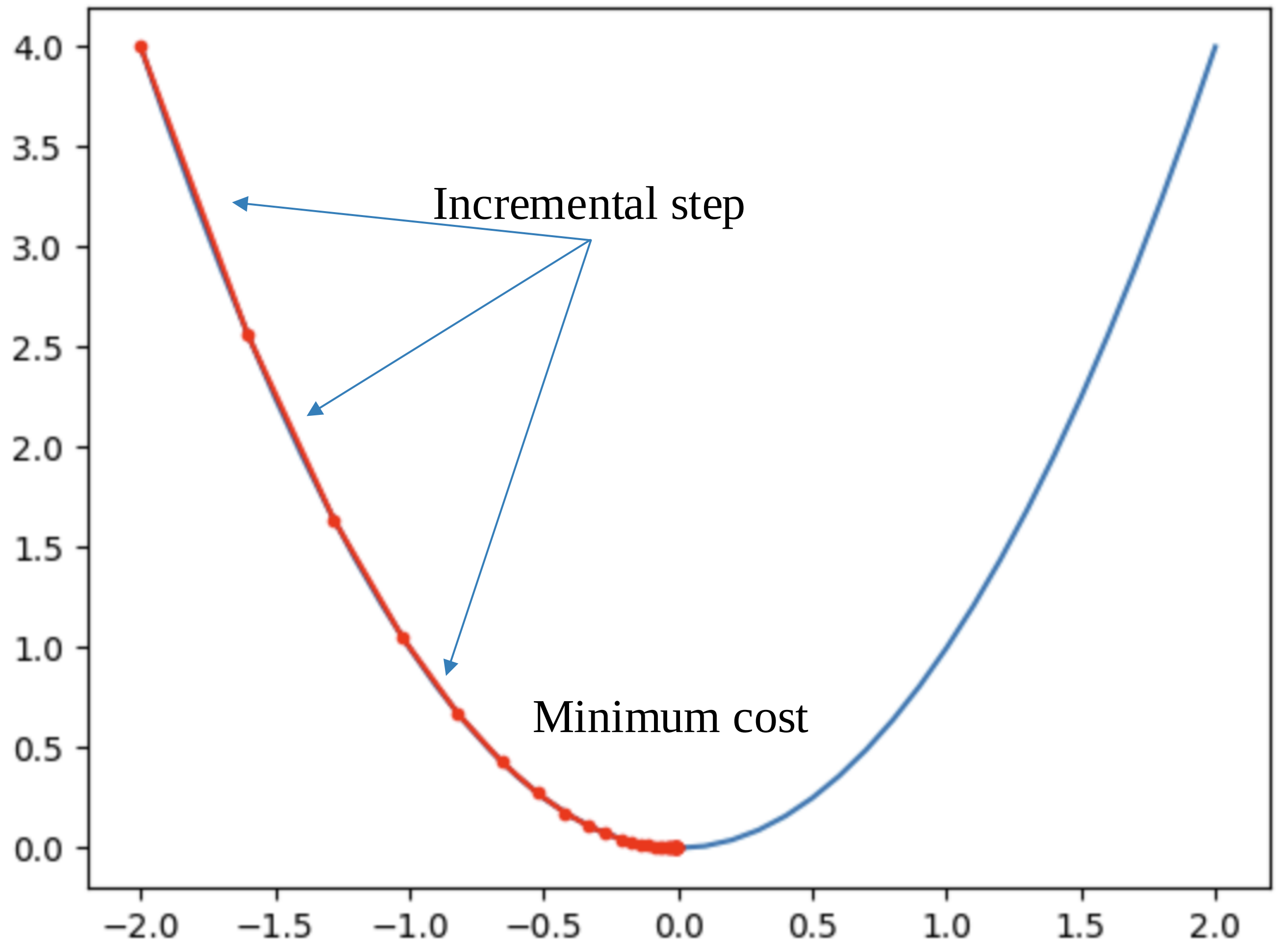


Momentum

SGD

- Objective function: x^2
- Derivative: $2x$
- Gradient:

$$x = x - \eta * f'(x)$$



AI VIET NAM
@aivietnam.edu.vn

Momentum

SGD

```python
def init_inputs(r_min=-2.0, r_max=2.0):
    inputs = torch.arange(r_min, r_max+0.1, 0.1)
    return inputs
```

```python
init_inputs(r_min=-2.0, r_max=2.0)
```

```
tensor([-2.0000e+00, -1.9000e+00, -1.8000e+00, -1.7000e+00, -1.6000e+00,
        -1.5000e+00, -1.4000e+00, -1.3000e+00, -1.2000e+00, -1.1000e+00,
        -1.0000e+00, -9.0000e-01, -8.0000e-01, -7.0000e-01, -6.0000e-01,
        -5.0000e-01, -4.0000e-01, -3.0000e-01, -2.0000e-01, -1.0000e-01,
        -5.9605e-09,  1.0000e-01,  2.0000e-01,  3.0000e-01,  4.0000e-01,
         5.0000e-01,  6.0000e-01,  7.0000e-01,  8.0000e-01,  9.0000e-01,
         1.0000e+00,  1.1000e+00,  1.2000e+00,  1.3000e+00,  1.4000e+00,
         1.5000e+00,  1.6000e+00,  1.7000e+00,  1.8000e+00,  1.9000e+00,
         2.0000e+00])
```

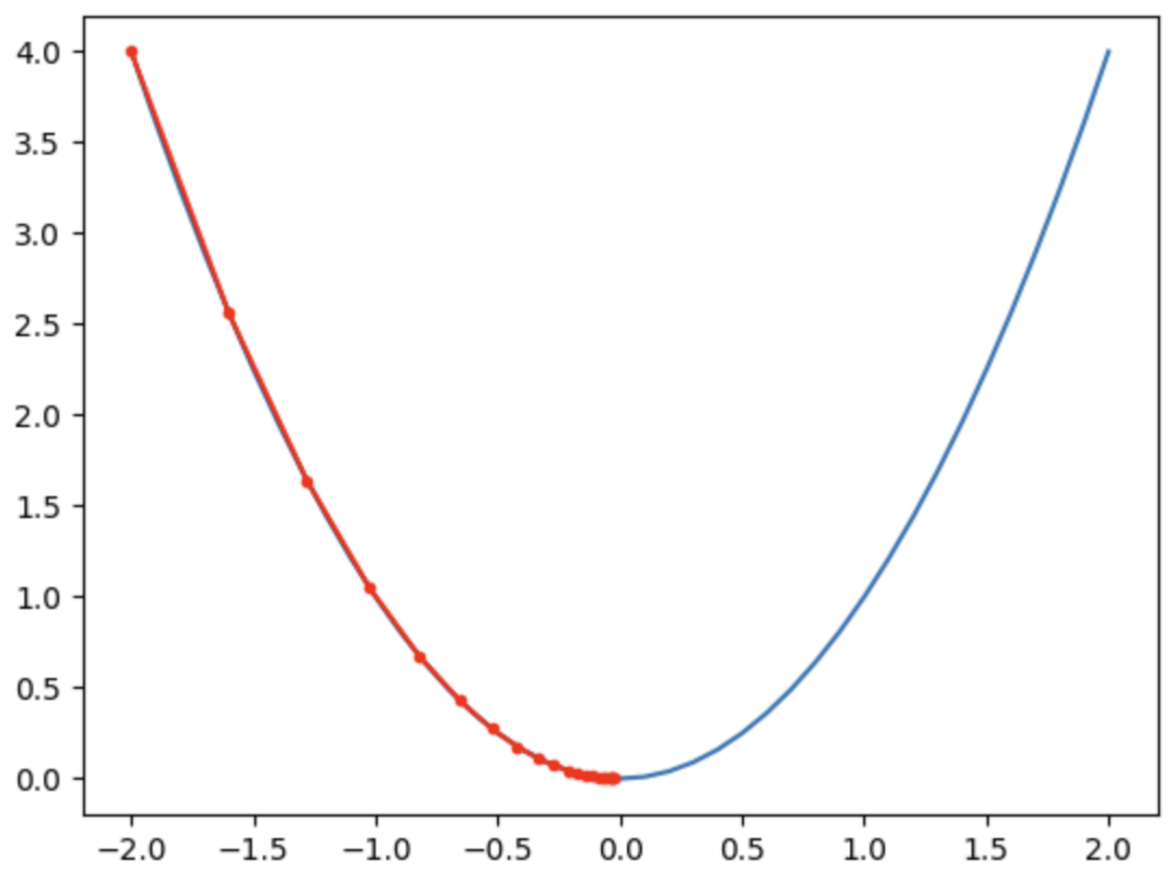
```python
def objective(x):
    return x**2.0

def derivative(x):
    return 2.0*x

def sgd(input, num_epochs, learning_rate):
    solutions, scores = [], []
    solution = input
    solution_eval = objective(solution)
    solutions.append(solution)
    scores.append(solution_eval)

    for epoch in range(num_epochs):
        # calculate gradient
        gradient = derivative(solution)
        # take a step
        solution = solution - learning_rate * gradient
        # evaluate candidate point
        solution_eval = objective(solution)
        # store solution
        solutions.append(solution)
        scores.append(solution_eval)
        # report progress
        print('Epoch: %0.2d -- f(%0.3f) = %.5f' % (
            epoch, solution, solution_eval))
    return solutions, scores
```

Momentum

SGD

```
num_epochs = 20
learning_rate = 0.1
inputs = init_inputs()
input = inputs[0]
solutions, scores = sgd(input, num_epochs, learning_rate)
```

```
Epoch: 00 -- f(-1.600) = 2.56000
Epoch: 01 -- f(-1.280) = 1.63840
Epoch: 02 -- f(-1.024) = 1.04858
Epoch: 03 -- f(-0.819) = 0.67109
Epoch: 04 -- f(-0.655) = 0.42950
Epoch: 05 -- f(-0.524) = 0.27488
Epoch: 06 -- f(-0.419) = 0.17592
Epoch: 07 -- f(-0.336) = 0.11259
Epoch: 08 -- f(-0.268) = 0.07206
Epoch: 09 -- f(-0.215) = 0.04612
Epoch: 10 -- f(-0.172) = 0.02951
Epoch: 11 -- f(-0.137) = 0.01889
Epoch: 12 -- f(-0.110) = 0.01209
Epoch: 13 -- f(-0.088) = 0.00774
Epoch: 14 -- f(-0.070) = 0.00495
Epoch: 15 -- f(-0.056) = 0.00317
Epoch: 16 -- f(-0.045) = 0.00203
Epoch: 17 -- f(-0.036) = 0.00130
Epoch: 18 -- f(-0.029) = 0.00083
Epoch: 19 -- f(-0.023) = 0.00053
```



Momentum

SGD

- Objective function:

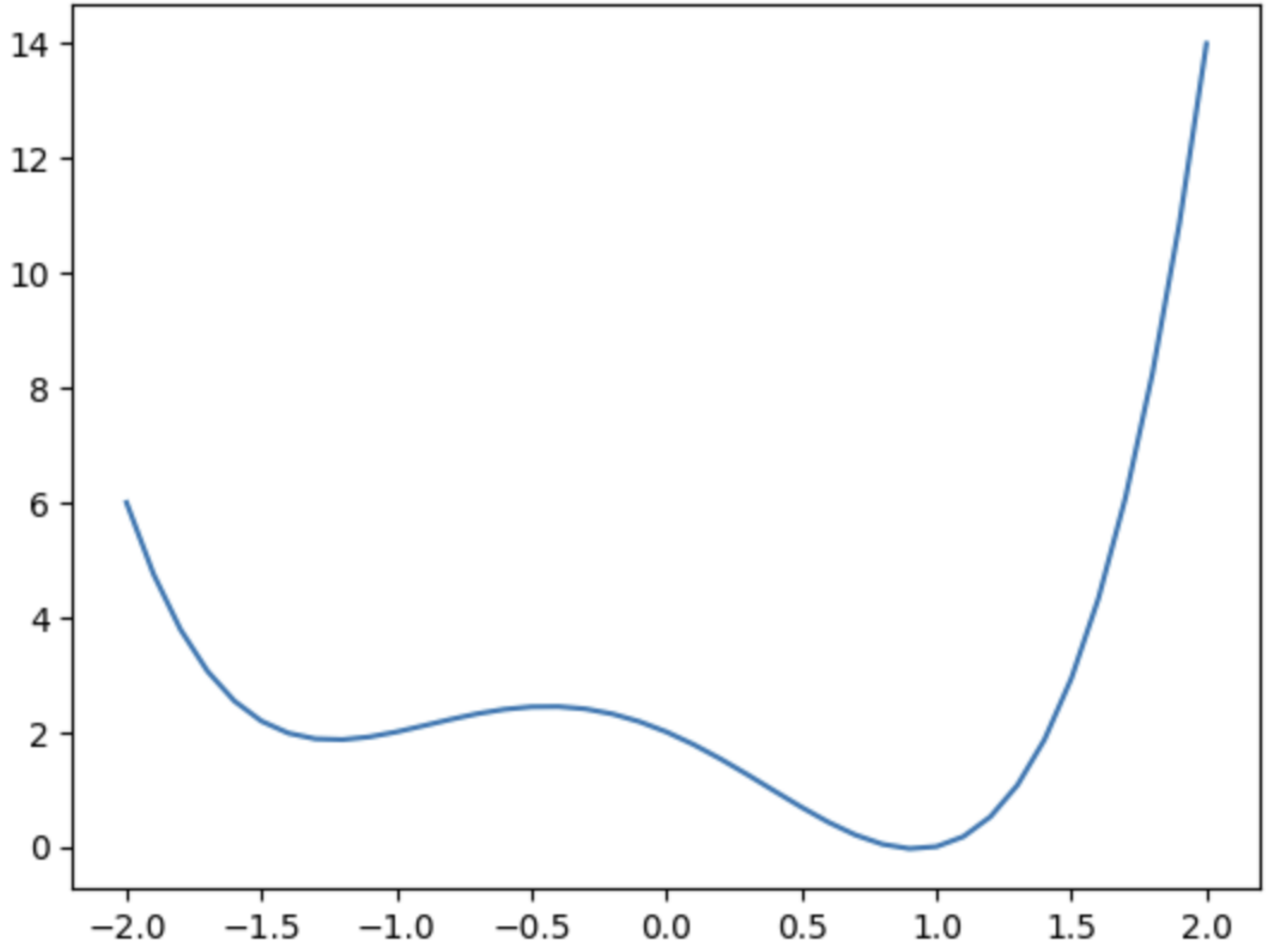
$$x^4 + x^3 - 2x^2 - 2x + 2$$

- Derivative:

$$4x^3 + 3x^2 - 4x - 2$$

- Gradient:

$$x = x - \eta * f'(x)$$



AI VIET NAM
@aivietnam.edu.vn

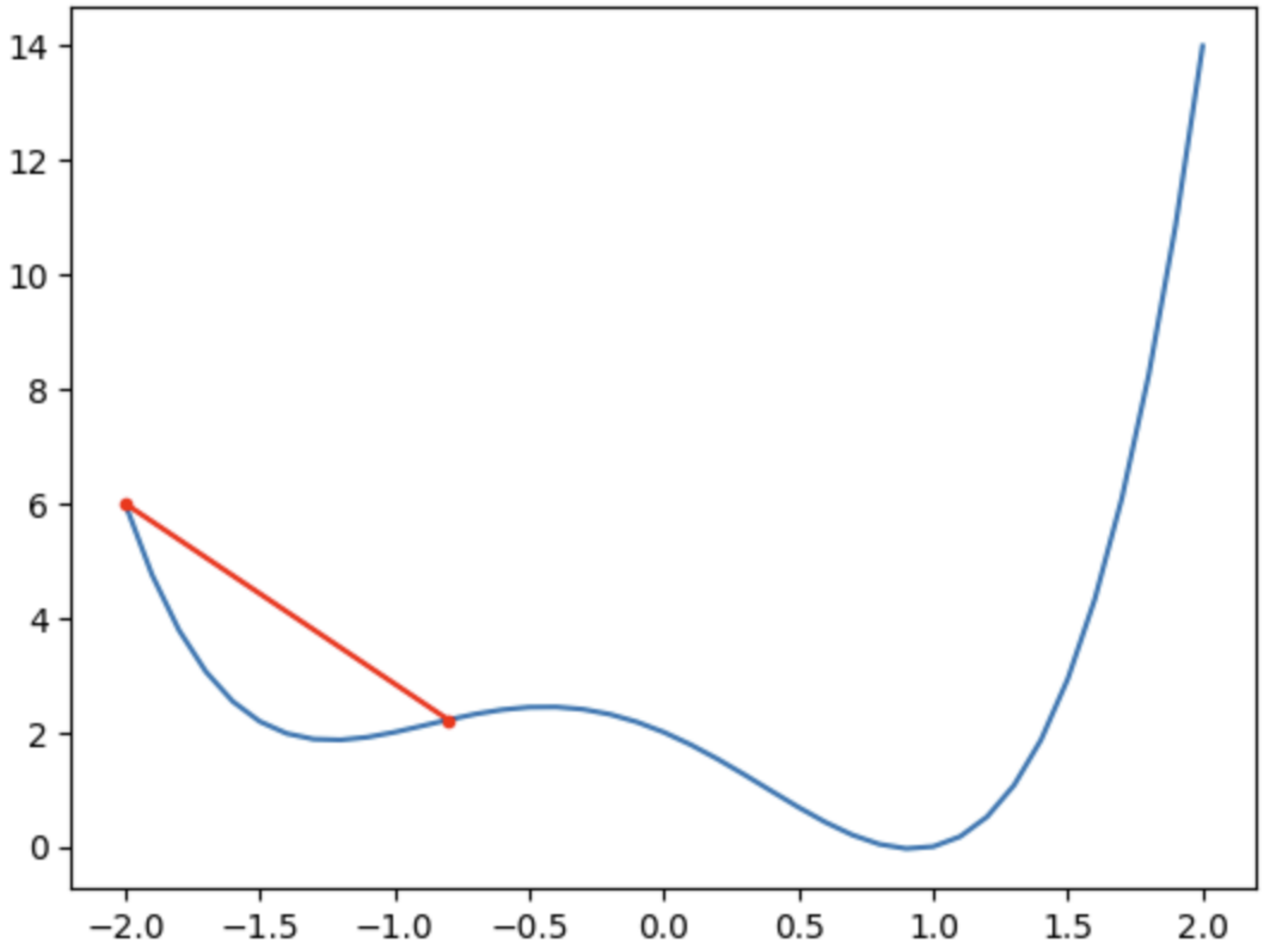
Momentum

SGD

- Objective function:
  $$x^4 + x^3 - 2x^2 - 2x + 2$$
- Derivative:
  $$4x^3 + 3x^2 - 4x - 2$$
- Gradient:
  $$x = x - \eta * f'(x)$$

$$x = -2.0$$
$$\eta = 0.1$$

$$x = -2.0 - 0.1 * (-14)$$
$$= -0.6$$



Momentum

SGD

- Objective function:

  $$x^4 + x^3 - 2x^2 - 2x + 2$$

- Derivative:

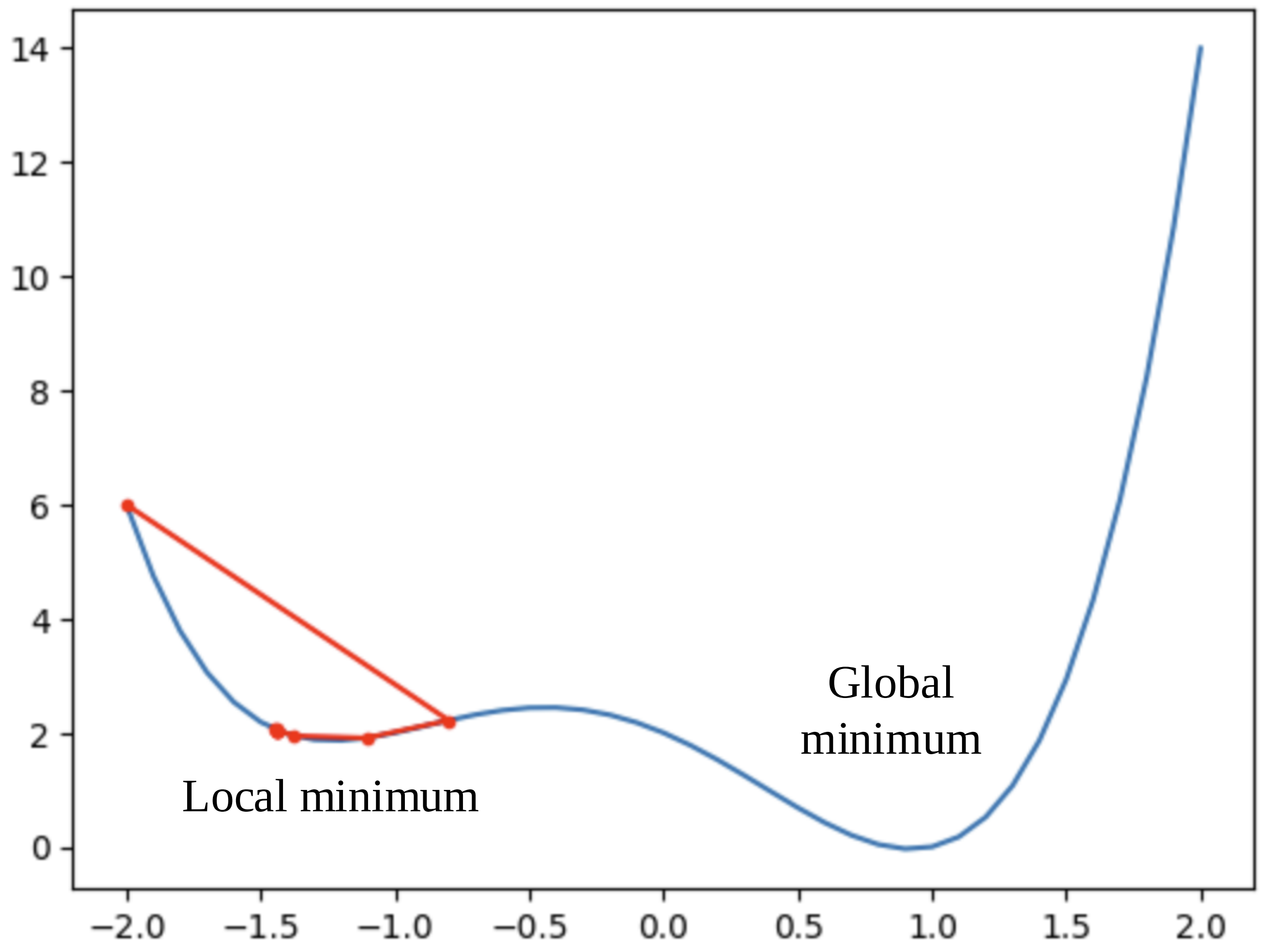
  $$4x^3 + 3x^2 - 4x - 2$$

- Gradient:

  $$x = x - \eta * f'(x)$$

$x = -2.0$
$\eta = 0.1$
num_epochs = 5

Cost = 2.05

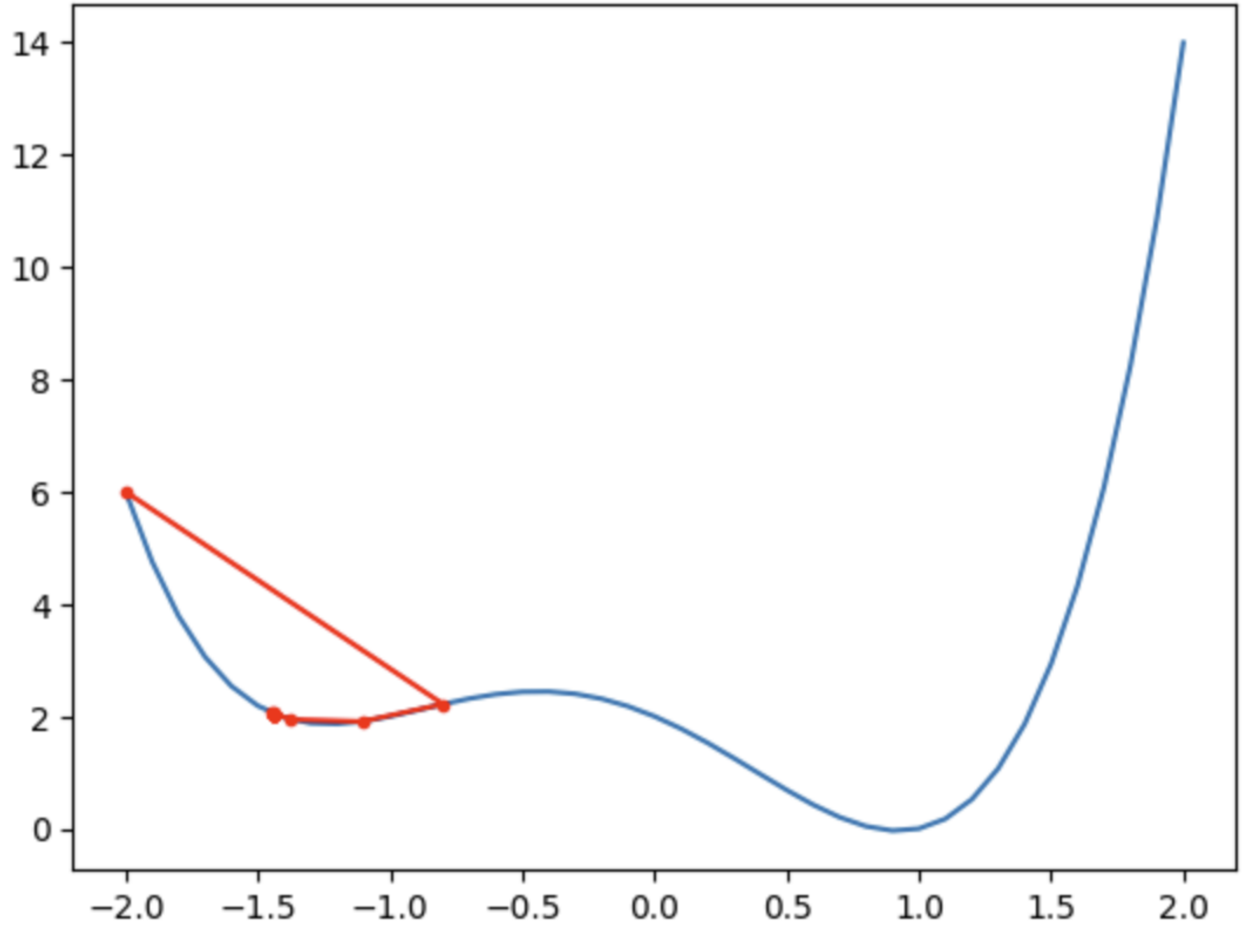


Momentum

SGD

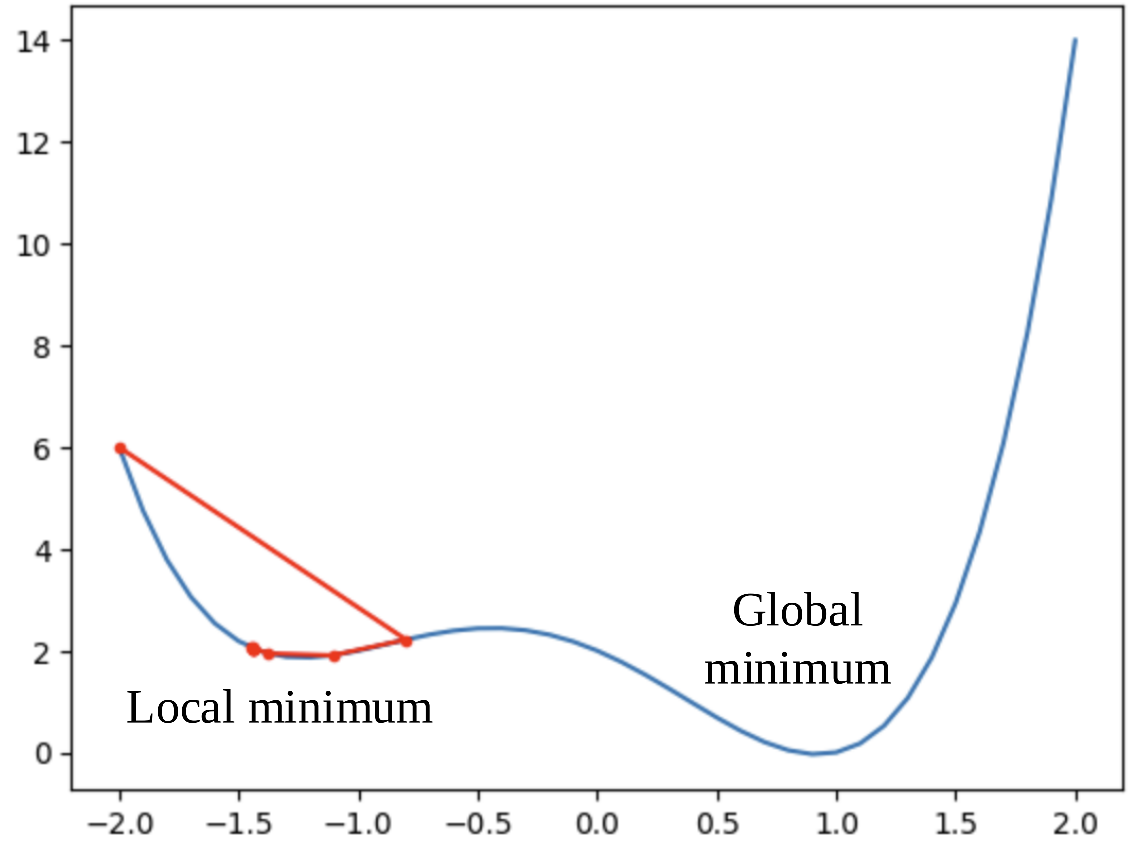
```
# objective function
def objective(x):
 return x**4+x**3-2*x**2-2*x + 2

# derivative of objective function
def derivative(x):
    return 4*x**3 + 3*x**2 - 4*x - 2

num_epochs = 20
learning_rate = 0.1
inputs = init_inputs()
input = inputs[0]
solutions, scores = sgd(input, num_epochs, learning_rate)
```

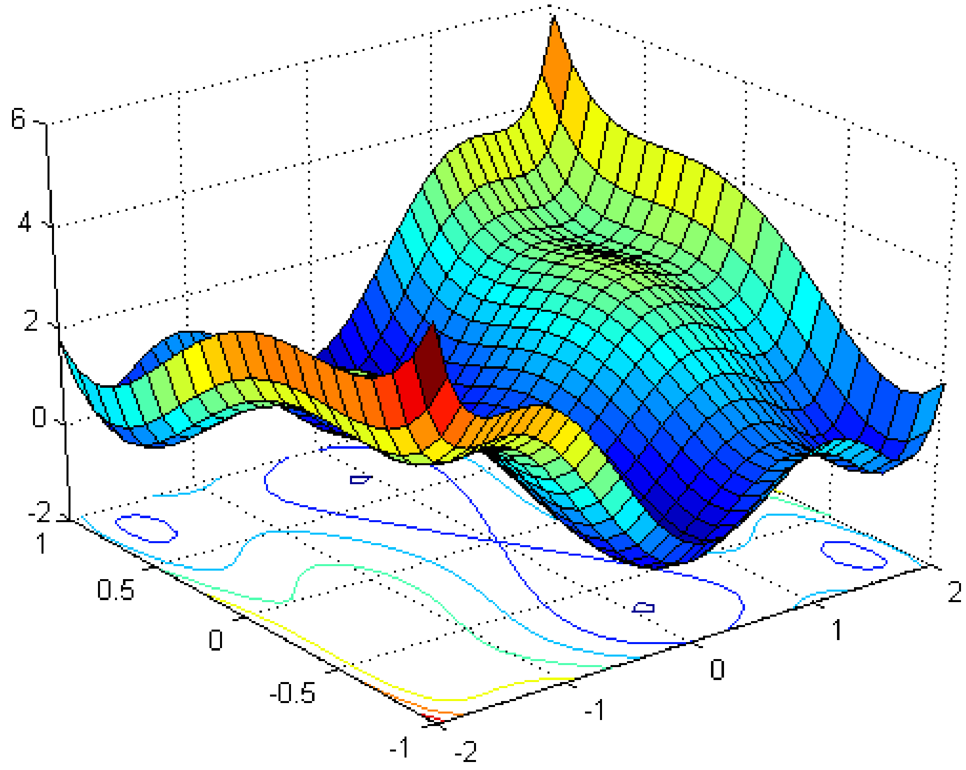
```
Epoch: 00 -- f(-0.600) = 2.39360
Epoch: 01 -- f(-0.662) = 2.34977
Epoch: 02 -- f(-0.742) = 2.27775
Epoch: 03 -- f(-0.840) = 2.17372
Epoch: 04 -- f(-0.951) = 2.05123
Epoch: 05 -- f(-1.059) = 1.94550
Epoch: 06 -- f(-1.144) = 1.88629
Epoch: 07 -- f(-1.195) = 1.86667
Epoch: 08 -- f(-1.219) = 1.86278
Epoch: 09 -- f(-1.228) = 1.86225
Epoch: 10 -- f(-1.231) = 1.86219
Epoch: 11 -- f(-1.232) = 1.86218
Epoch: 12 -- f(-1.232) = 1.86218
Epoch: 13 -- f(-1.232) = 1.86218
Epoch: 14 -- f(-1.232) = 1.86218
Epoch: 15 -- f(-1.232) = 1.86218
Epoch: 16 -- f(-1.232) = 1.86218
Epoch: 17 -- f(-1.232) = 1.86218
Epoch: 18 -- f(-1.232) = 1.86218
Epoch: 19 -- f(-1.232) = 1.86218
```



Momentum

SGD





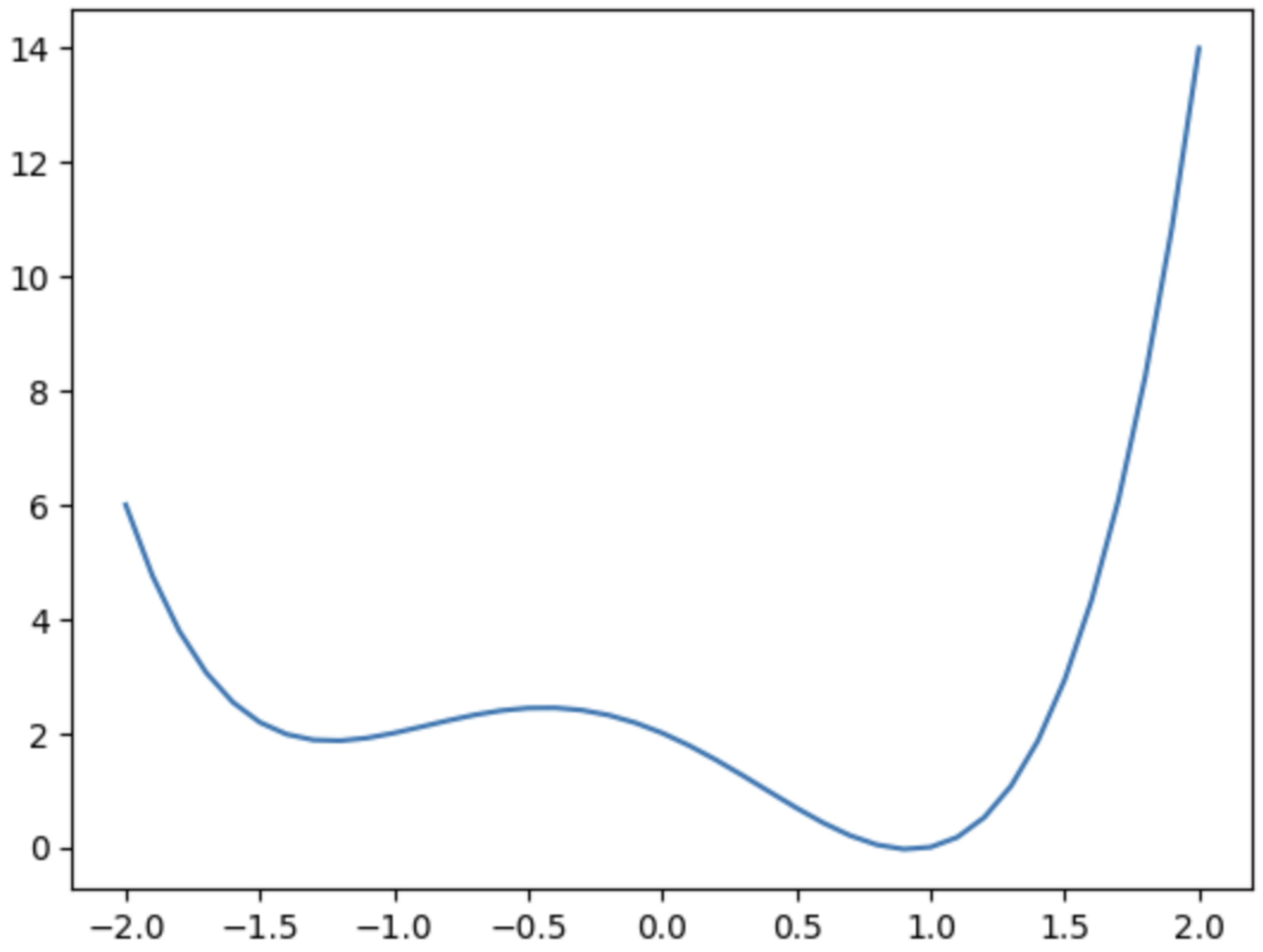
Momentum

SGD with Momentum

- Objective function:
  $$x^4 + x^3 - 2x^2 - 2x + 2$$
- Derivative:
  $$4x^3 + 3x^2 - 4x - 2$$
- Gradient with Momentum:

$$v_t = \gamma v_{t-1} + \eta f'(x)$$
$$x = x - v_t$$

γ: momentum



Momentum

SGD with Momentum

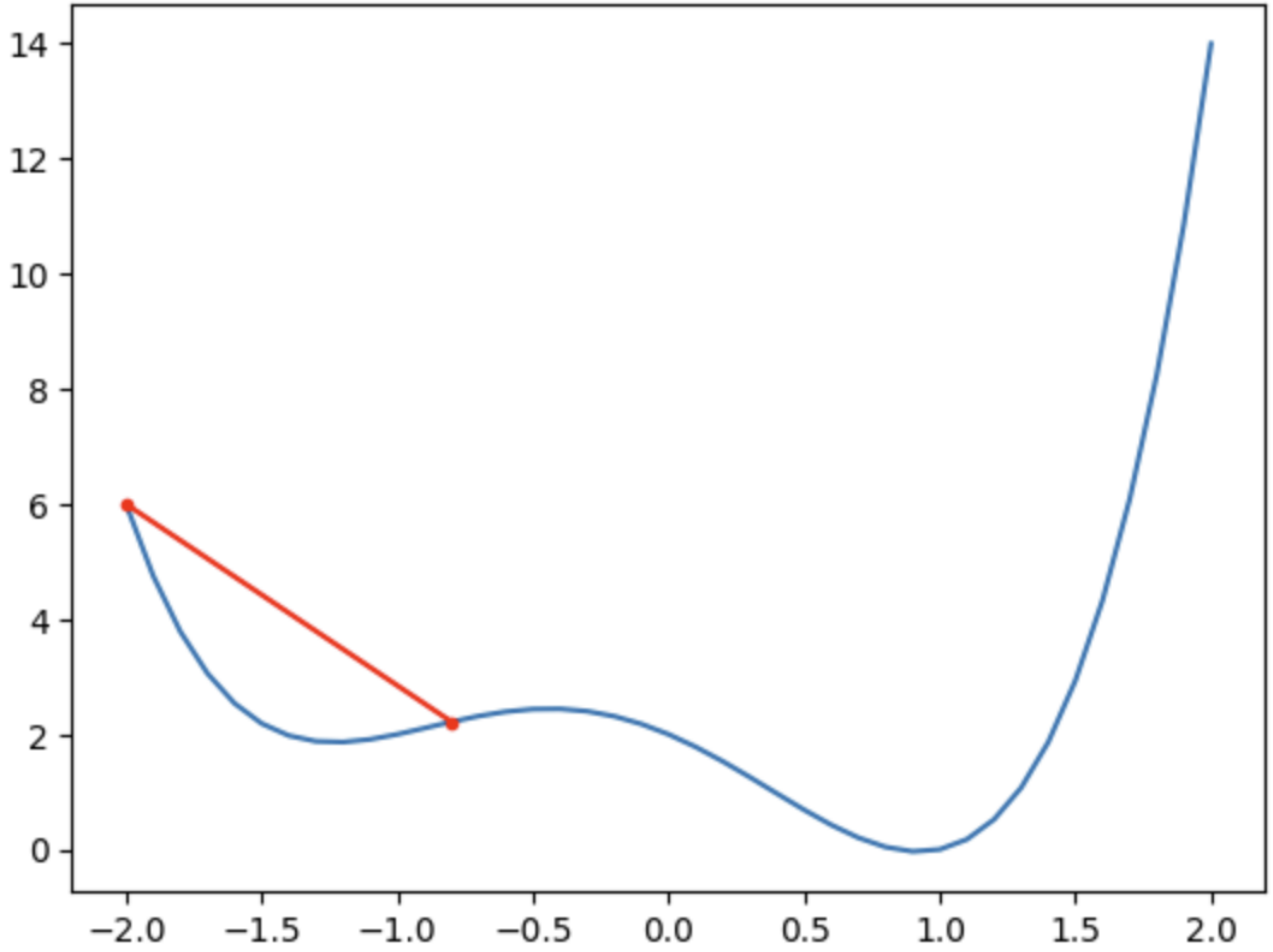
- Objective function:
  $$x^4 + x^3 - 2x^2 - 2x + 2$$
- Derivative:
  $$4x^3 + 3x^2 - 4x - 2$$
- Gradient with Momentum:

$$v_t = \gamma v_{t-1} + \eta f'(x)$$
$$x = x - v_t$$

$x = -2.0 \quad \eta = 0.1 \quad \gamma = 0.8 \quad v_0 = 0.0$

$v_1 = 0.8 * 0.0 + 0.1 * (-14) = -1.4$

$x = -2.0 - (-1.4) = -0.6$



Momentum

SGD with Momentum

- Objective function:

$$x^4 + x^3 - 2x^2 - 2x + 2$$

- Derivative:

$$4x^3 + 3x^2 - 4x - 2$$

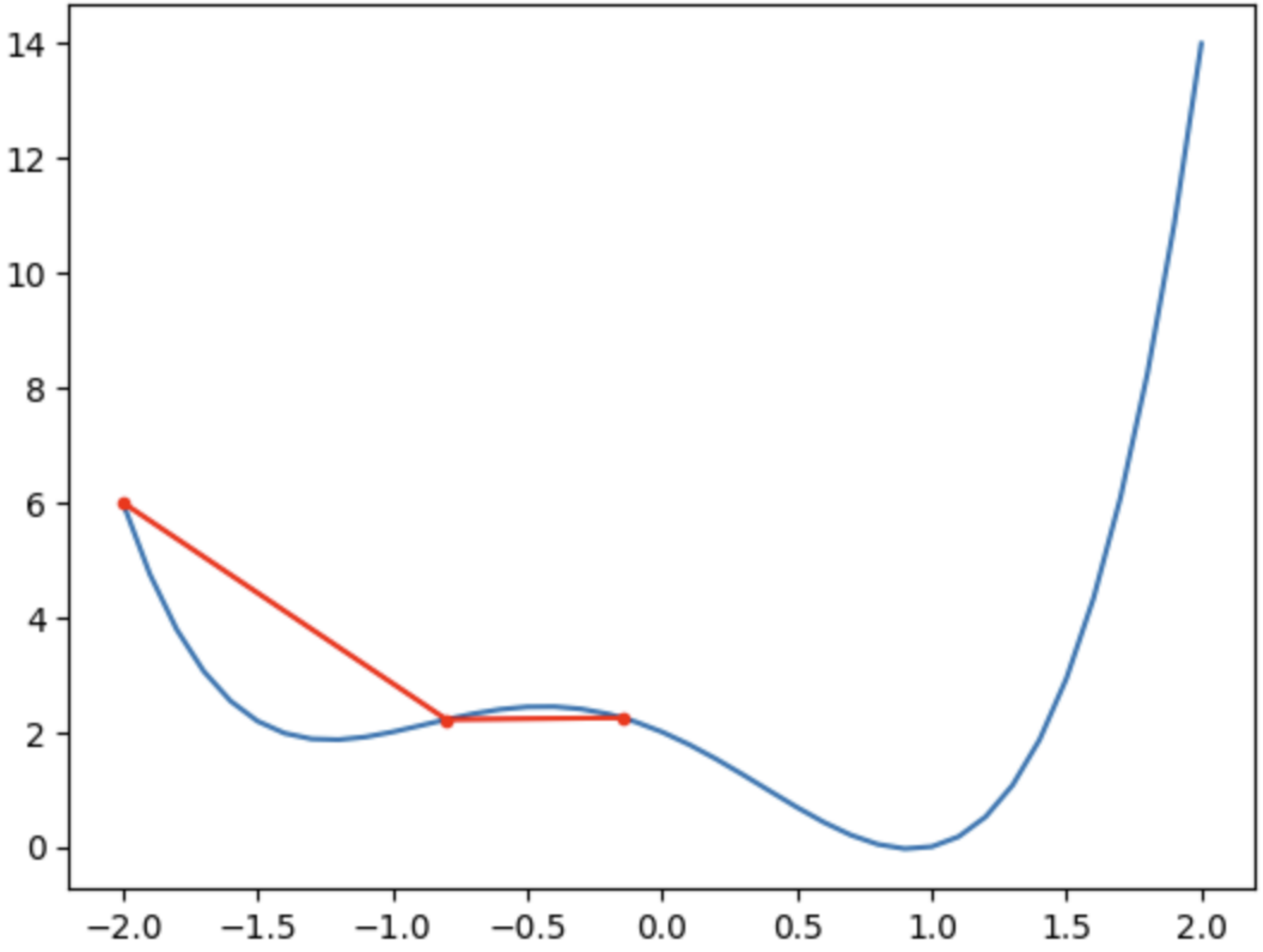
- Gradient with Momentum:

$$v_t = \gamma v_{t-1} + \eta f'(x)$$
$$x = x - v_t$$

$x = -2.0 \quad \eta = 0.1 \quad \gamma = 0.8 \quad v_1 = -1.2$

$v_1 = 0.8 * (-1.4) + 0.1 * (0.62) = -1.05$

$x = -0.6 - (-1.05) = 0.45$



AI VIET NAM
@aivietnam.edu.vn

Momentum

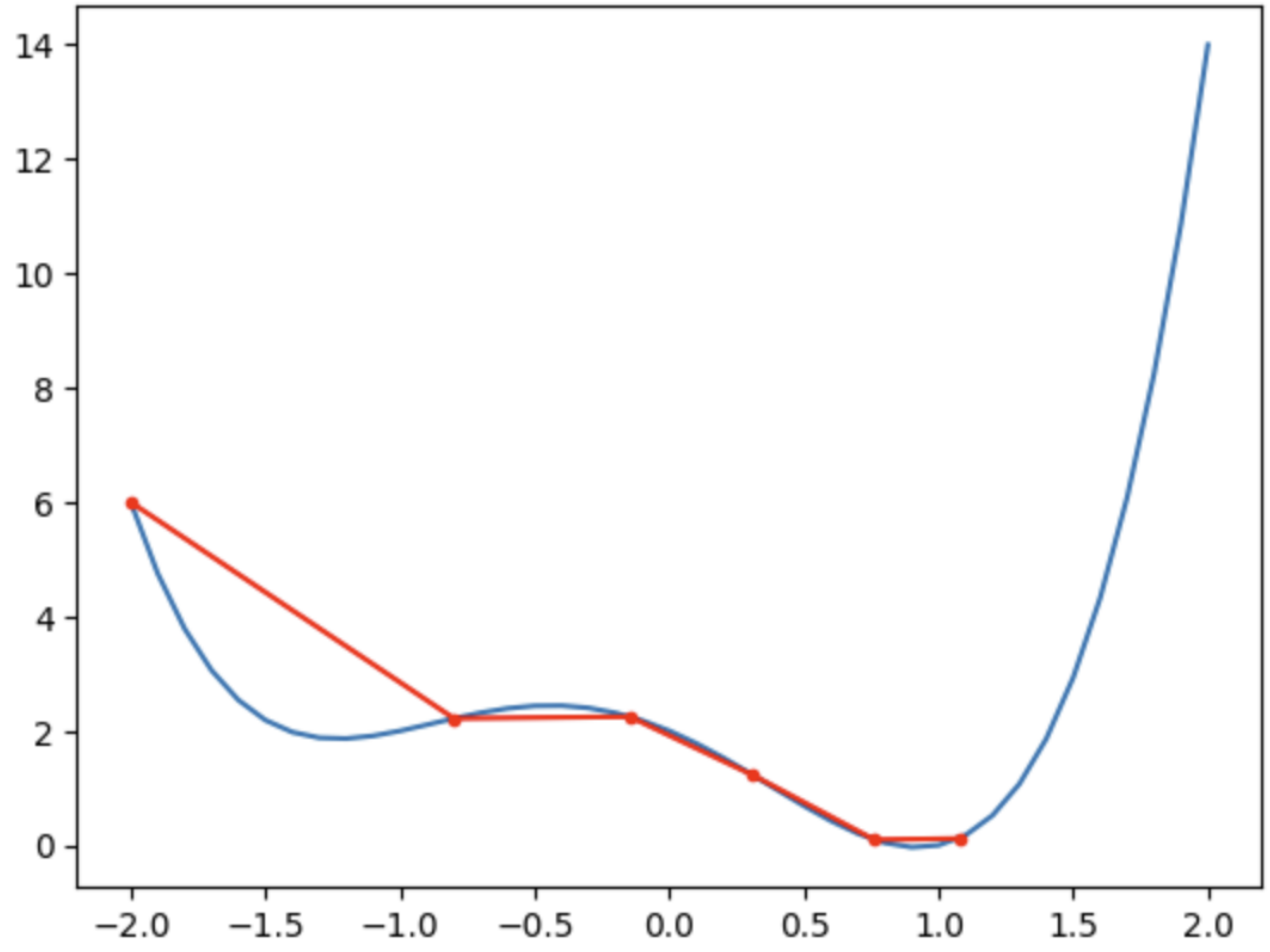
SGD with Momentum

- Objective function:
  $$x^4 + x^3 - 2x^2 - 2x + 2$$
- Derivative:
  $$4x^3 + 3x^2 - 4x - 2$$
- Gradient with Momentum:

$$v_t = \gamma v_{t-1} + \eta f'(x)$$
$$x = x - v_t$$

$x = -2.0 \quad \eta = 0.1 \quad \gamma = 0.8 \quad v_0 = 0.0$

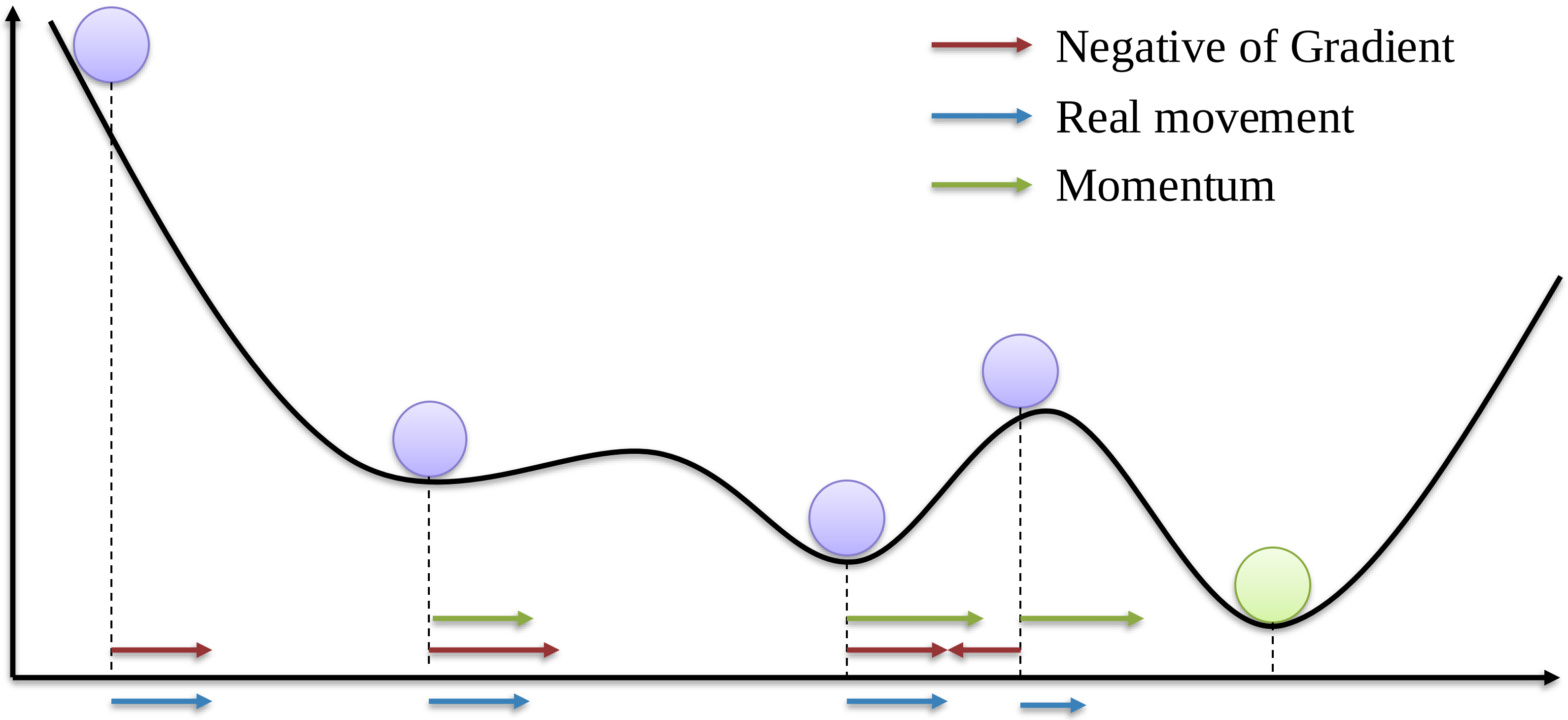
num_epochs = 5



Momentum

SGD with Momentum

$$v_t = \gamma v_{t-1} + \eta f'(x)$$
$$x = x - v_t$$



Movement = Negative of Gradient + Momentum

Momentum

SGD with Momentum

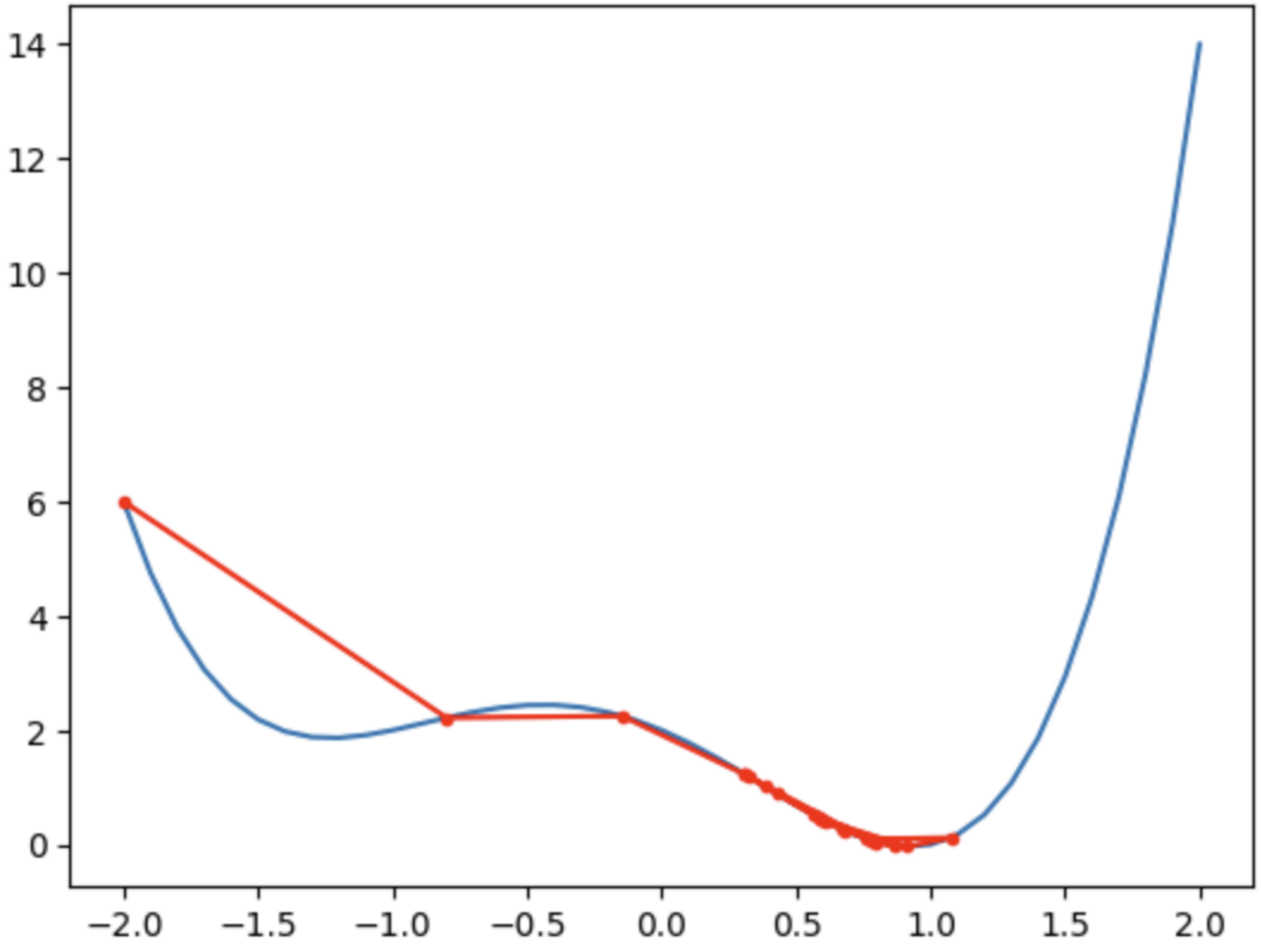
```python
# gradient descent algorithm
def gradient_descent_with_momentum(input, num_epochs, learning_rate, momentum):
    solutions, scores = [], []
    solution = input
    # keep track of the change
    change = 0.0
    solution_eval = objective(solution)
    solutions.append(solution)
    scores.append(solution_eval)

    for epoch in range(num_epochs):
        # calculate gradient
        gradient = derivative(solution)
        # calculate update
        new_change = learning_rate * gradient + momentum * change
        # take a step
        solution = solution - new_change
        # save the change
        change = new_change
        # evaluate candidate point
        solution_eval = objective(solution)
        # store solution
        solutions.append(solution)
        scores.append(solution_eval)
        # report progress
        print('Epoch: %0.2d -- f(%0.3f) = %.5f' % (
            epoch, solution, solution_eval))
    return solutions, scores
```

Momentum

SGD with Momentum

```python
num_epochs = 20
learning_rate = 0.1
inputs = init_inputs()
input = inputs[0]
# define momentum
momentum = 0.8
solutions, scores = gradient_descent_with_momentum(
    input, num_epochs, learning_rate, momentum
)
```

```
Epoch: 00 -- f(-0.800) = 2.21760
Epoch: 01 -- f(-0.147) = 2.24834
Epoch: 02 -- f(0.311) = 1.22418
Epoch: 03 -- f(0.761) = 0.09618
Epoch: 04 -- f(1.075) = 0.11696
Epoch: 05 -- f(0.913) = -0.03723
Epoch: 06 -- f(0.594) = 0.44088
Epoch: 07 -- f(0.387) = 1.00798
Epoch: 08 -- f(0.308) = 1.23381
Epoch: 09 -- f(0.327) = 1.17774
Epoch: 10 -- f(0.428) = 0.89005
Epoch: 11 -- f(0.593) = 0.44244
Epoch: 12 -- f(0.774) = 0.07689
Epoch: 13 -- f(0.863) = -0.01795
Epoch: 14 -- f(0.799) = 0.04293
Epoch: 15 -- f(0.672) = 0.26025
Epoch: 16 -- f(0.582) = 0.46945
Epoch: 17 -- f(0.563) = 0.51926
Epoch: 18 -- f(0.606) = 0.41078
Epoch: 19 -- f(0.684) = 0.23563
```



QUIZ TIME

Momentum

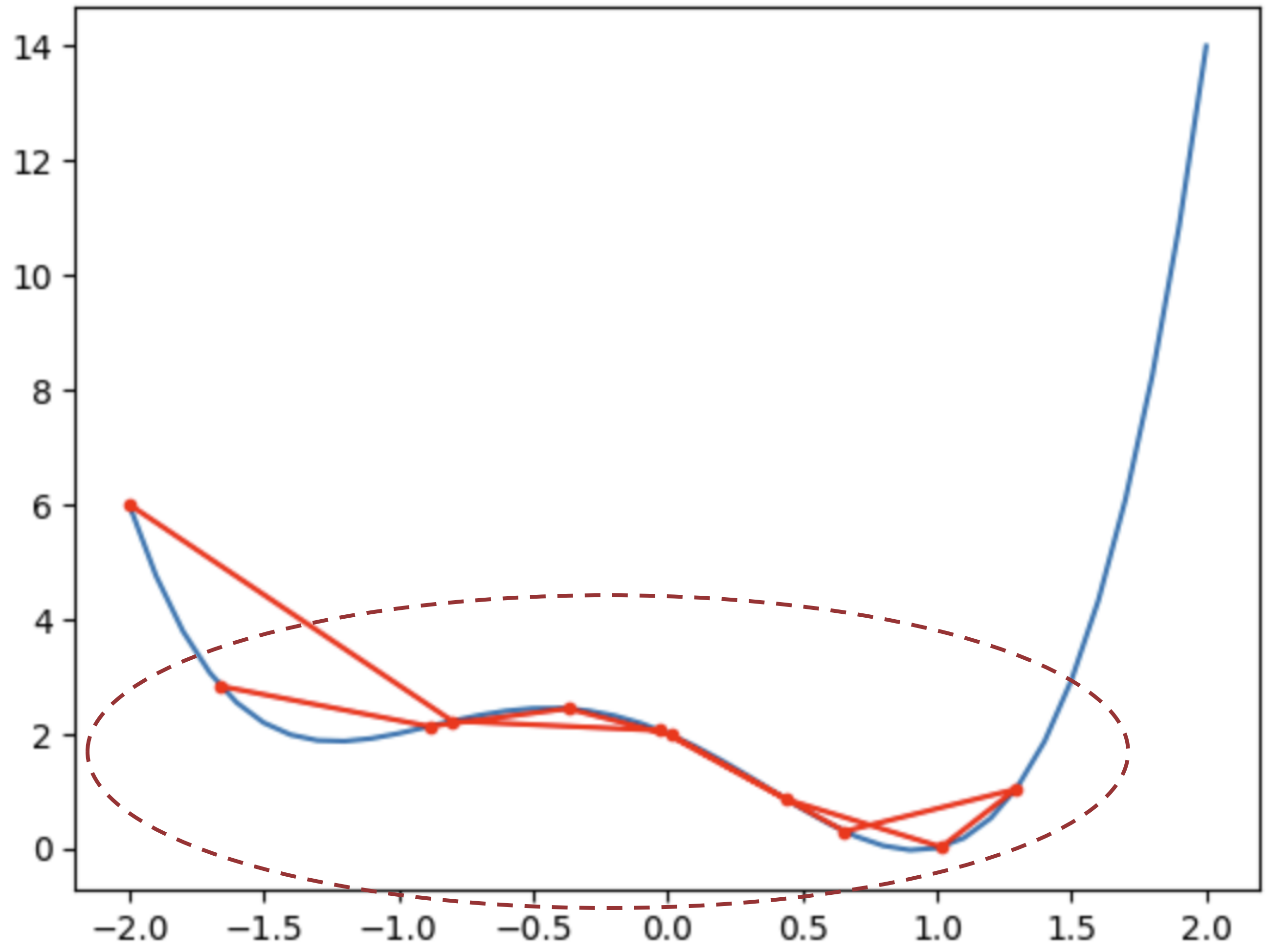
Problem of SGD with Momentum

- Objective function:
  $$x^4 + x^3 - 2x^2 - 2x + 2$$
- Derivative:
  $$4x^3 + 3x^2 - 4x - 2$$
- Gradient with Momentum:

$$v_t = \gamma v_{t-1} + \eta f'(x)$$
$$x = x - v_t$$

$x = -2.0$ $\eta = 0.1$ $\gamma = 0.9$ $v_0 = 0.0$

num_epochs = 10



AI VIET NAM
@aivietnam.edu.vn

Momentum

Nesterov Momentum

- Objective function:

$$x^4 + x^3 - 2x^2 + 2$$

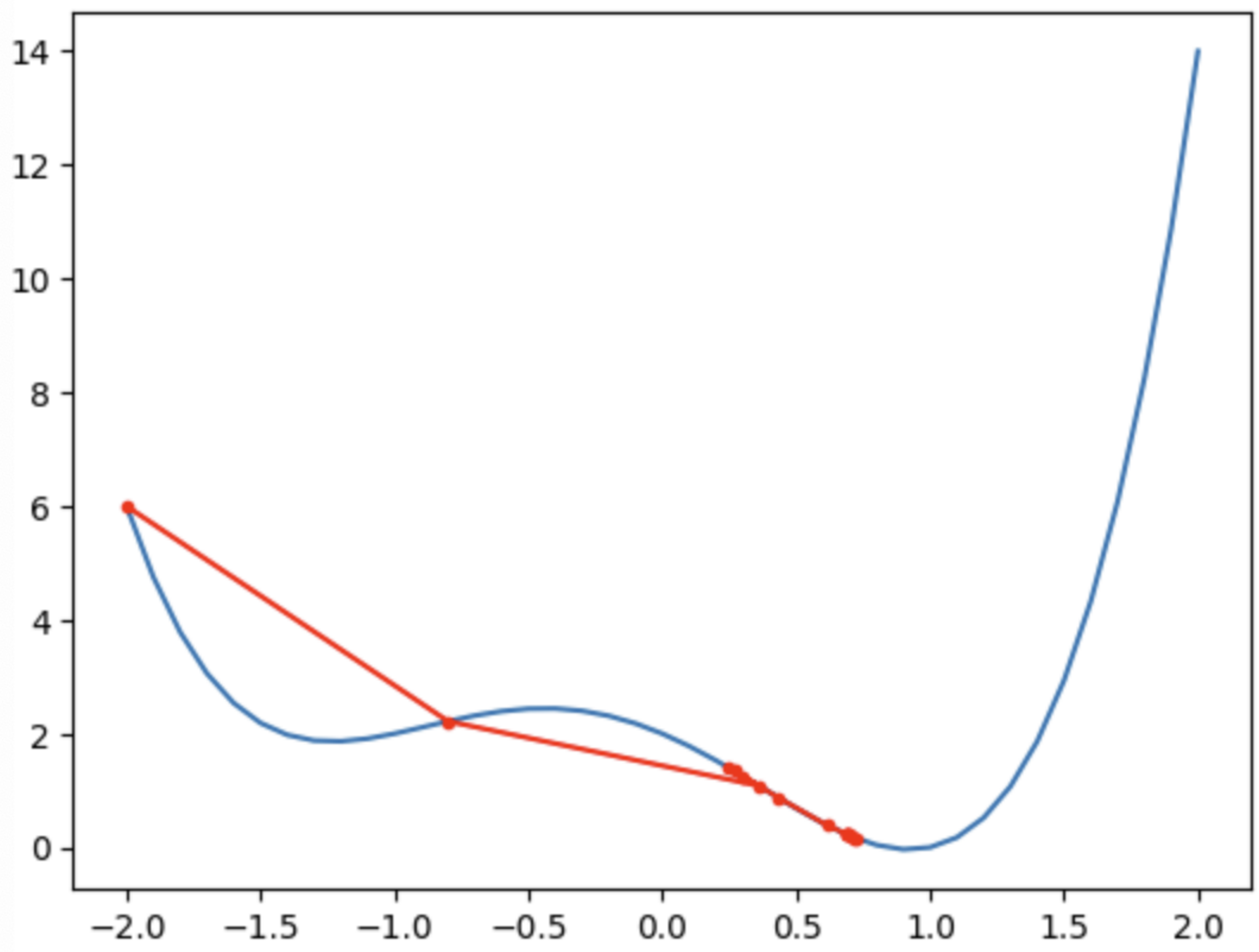
- Derivative:

$$4x^3 + 3x^2 - 4x - 2$$

- Gradient with Momentum:

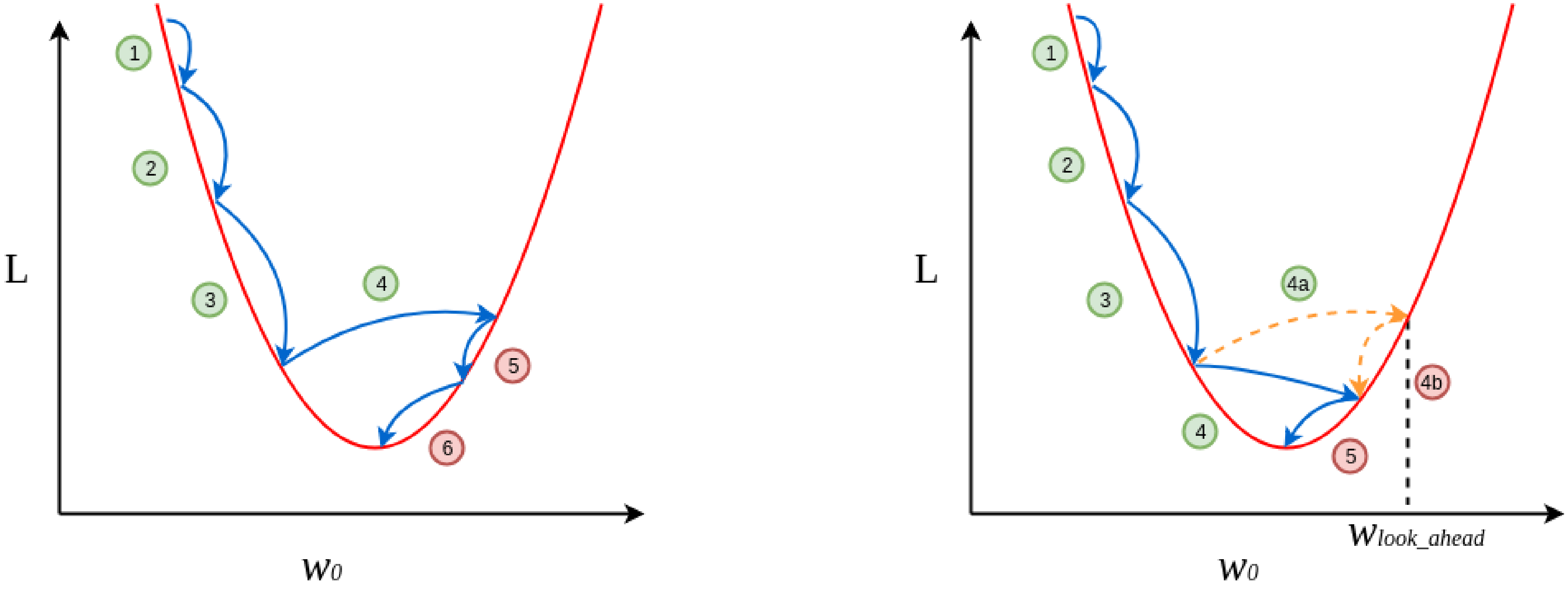
$$v_t = \gamma v_{t-1} - \eta f'(x + \gamma * v_{t-1})$$
$$x = x + v_t$$

$x = -2.0 \quad \eta = 0.1 \quad \gamma = 0.9 \quad v_0 = 0.0$



Momentum

Nesterov Momentum



(a) Momentum-Based Gradient Descent

(b) Nesterov Accelerated Gradient Descent

$\Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$ (green)

$\Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$ (red)

Momentum

Nesterov Momentum

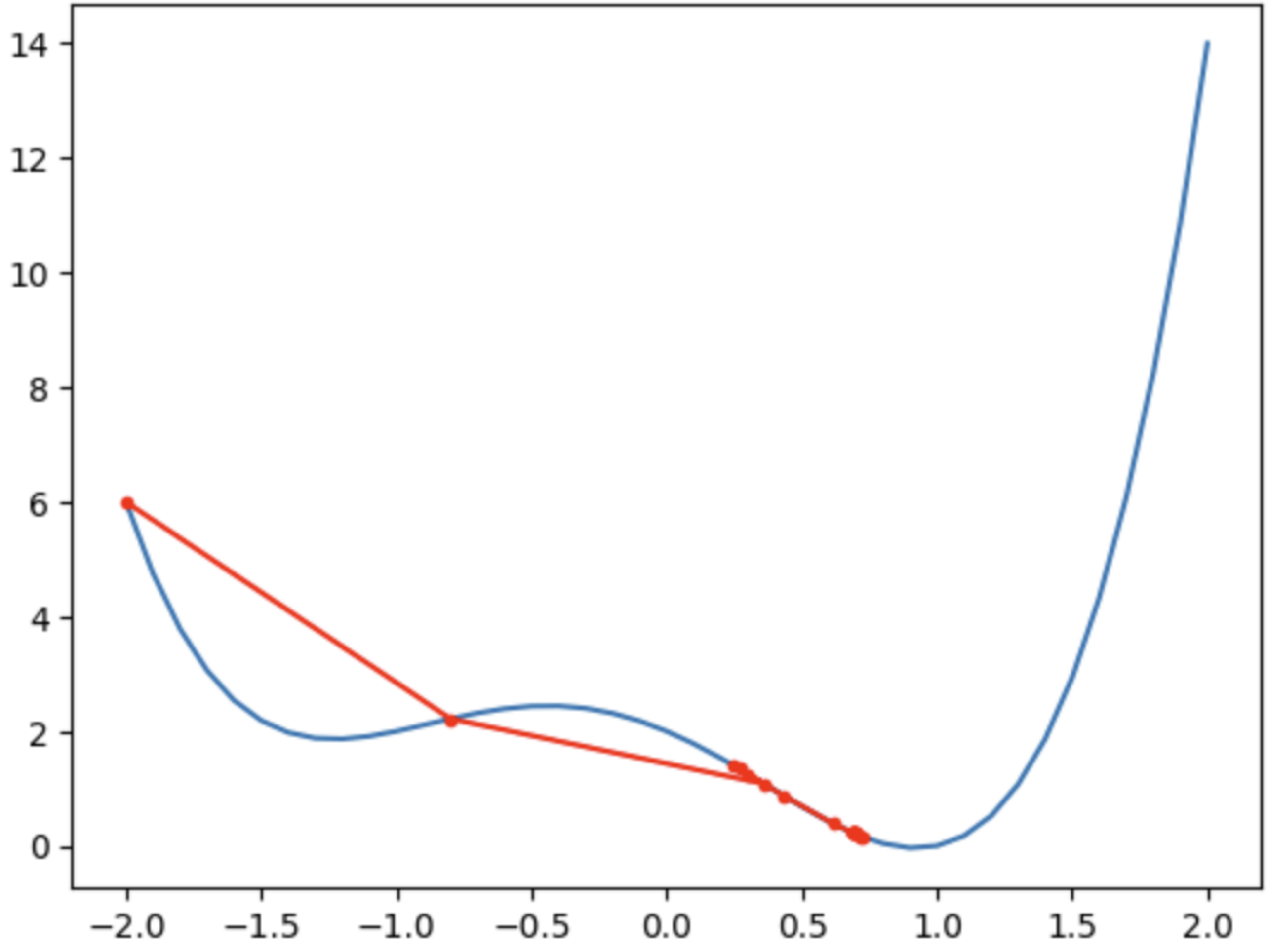
```python
# gradient descent algorithm
def nesterov_momentum(input, num_epochs, learning_rate, momentum):
    solutions, scores = [], []
    solution = input
    # keep track of the change
    change = 0.0
    solution_eval = objective(solution)
    solutions.append(solution)
    scores.append(solution_eval)

    for epoch in range(num_epochs):
        # calculate the projected solution
        projected = solution + momentum * change
        # calculate gradient
        gradient = derivative(projected)
        # calculate update
        new_change = momentum * change - learning_rate * gradient
        # take a step
        solution = solution + new_change
        # save the change
        change = new_change
        # evaluate candidate point
        solution_eval = objective(solution)
        # store solution
        solutions.append(solution)
        scores.append(solution_eval)
        # report progress
        print('Epoch: %0.2d -- f(%0.3f) = %.5f' % (
            epoch, solution, solution_eval))
    return solutions, scores
```

Momentum

Nesterov Momentum

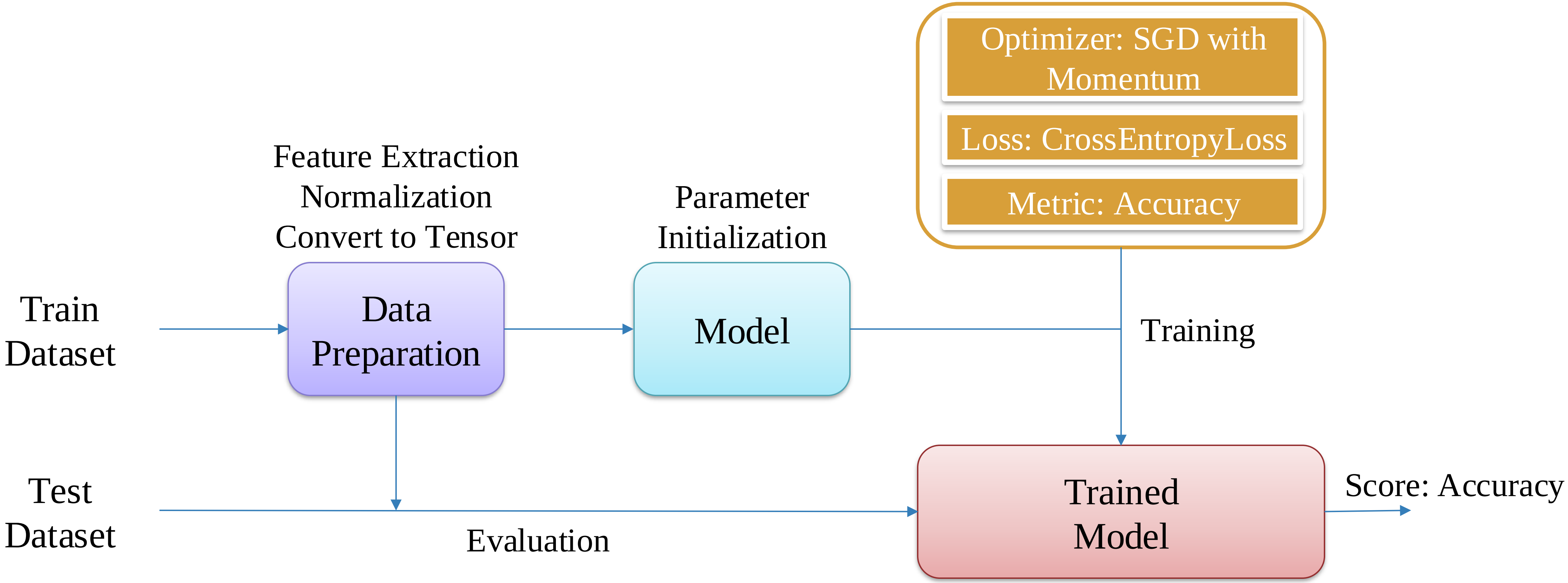
```
num_epochs = 20
learning_rate = 0.1
inputs = init_inputs()
input = inputs[0]
# define momentum
momentum = 0.9
solutions, scores = nesterov_momentum(
    input, num_epochs, learning_rate, momentum
)
```

```
Epoch: 00 -- f(-0.800) = 2.21760
Epoch: 01 -- f(0.360) = 1.08511
Epoch: 02 -- f(0.268) = 1.34411
Epoch: 03 -- f(0.247) = 1.40184
Epoch: 04 -- f(0.300) = 1.25650
Epoch: 05 -- f(0.432) = 0.87709
Epoch: 06 -- f(0.614) = 0.39136
Epoch: 07 -- f(0.719) = 0.16670
Epoch: 08 -- f(0.725) = 0.15602
Epoch: 09 -- f(0.707) = 0.19025
Epoch: 10 -- f(0.692) = 0.21938
Epoch: 11 -- f(0.687) = 0.22943
Epoch: 12 -- f(0.689) = 0.22596
Epoch: 13 -- f(0.692) = 0.21937
Epoch: 14 -- f(0.694) = 0.21555
Epoch: 15 -- f(0.694) = 0.21499
Epoch: 16 -- f(0.694) = 0.21595
Epoch: 17 -- f(0.693) = 0.21690
Epoch: 18 -- f(0.693) = 0.21728
Epoch: 19 -- f(0.693) = 0.21723
```



Classification

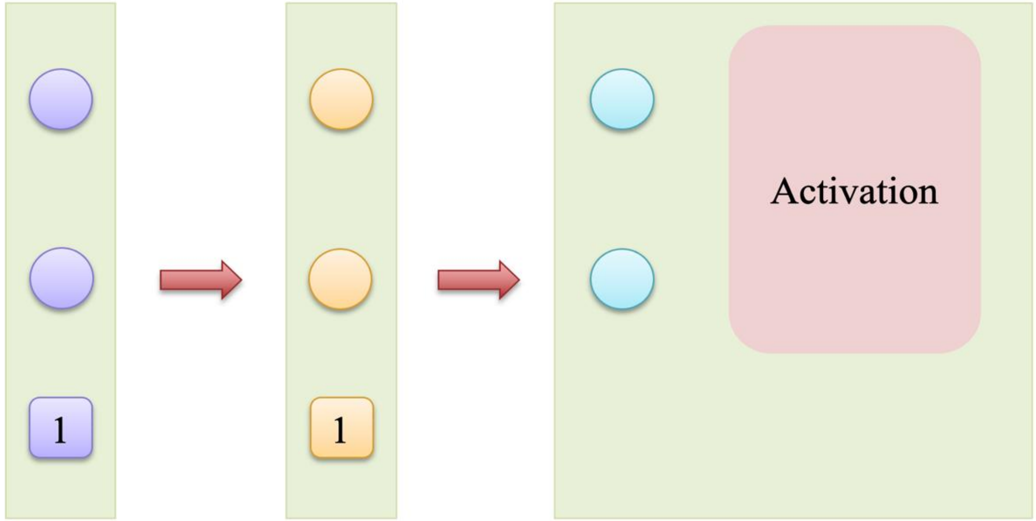
IRIS Dataset

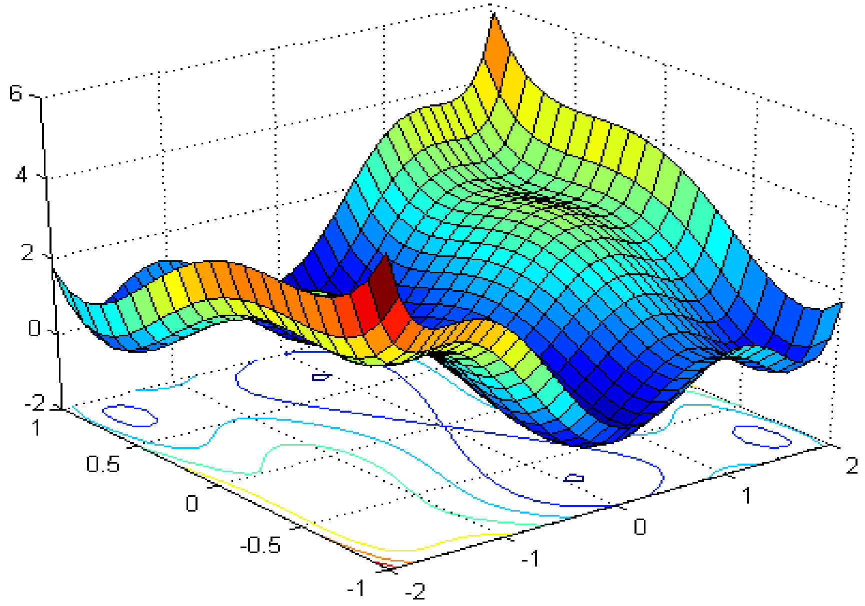


Objectives

Multi-layer Perceptron (Review)

- ❖ Hidden Layers
- ❖ Activation, Loss Function, Optimization
- ❖ Classification using MLP





Momentum

- ❖ Gradient Descent
- ❖ Gradient Descent with Momentum
- ❖ Nesterov Momentum

Thanks!

Any questions?