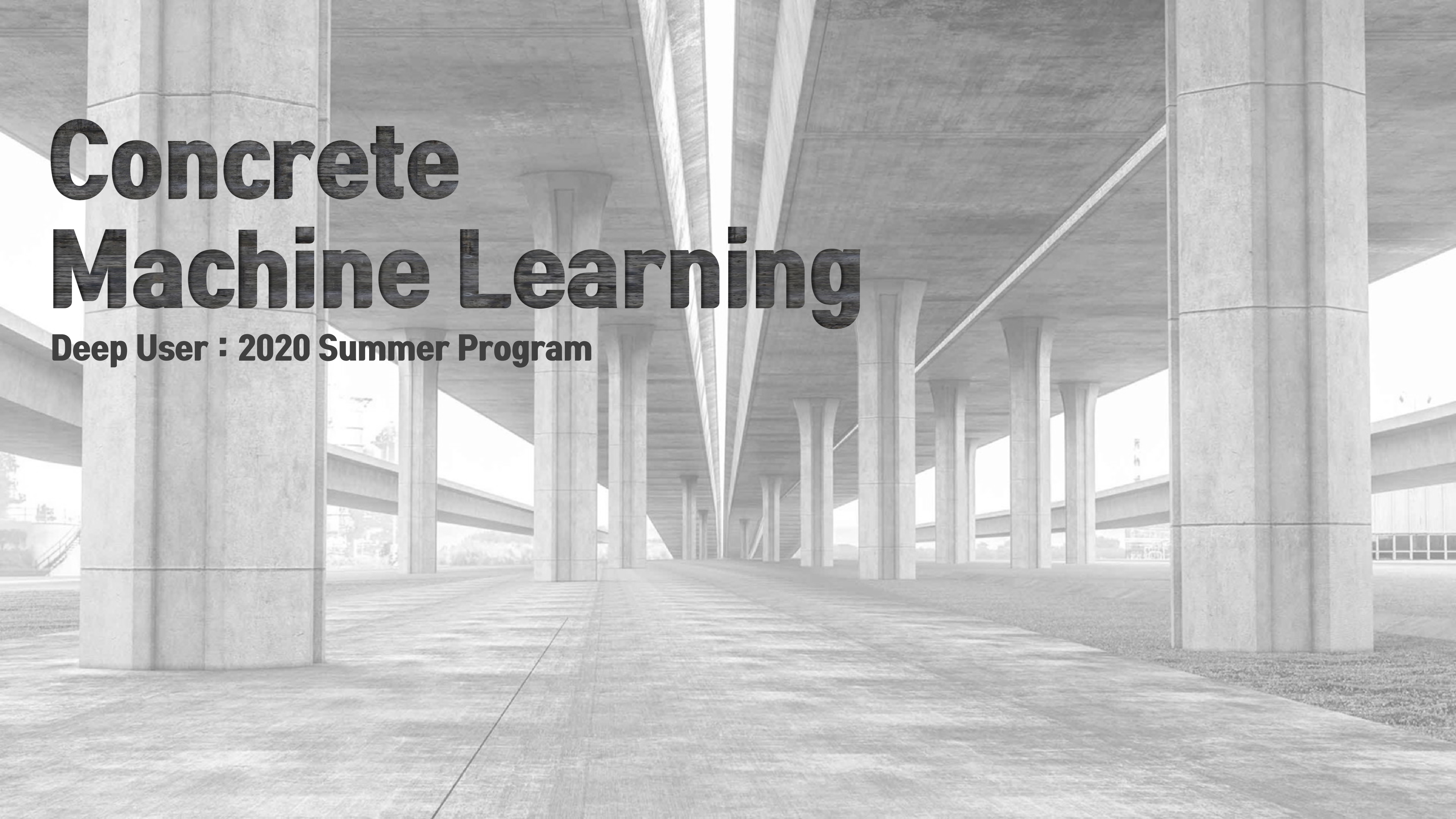


Concrete Machine Learning

Deep User : 2020 Summer Program

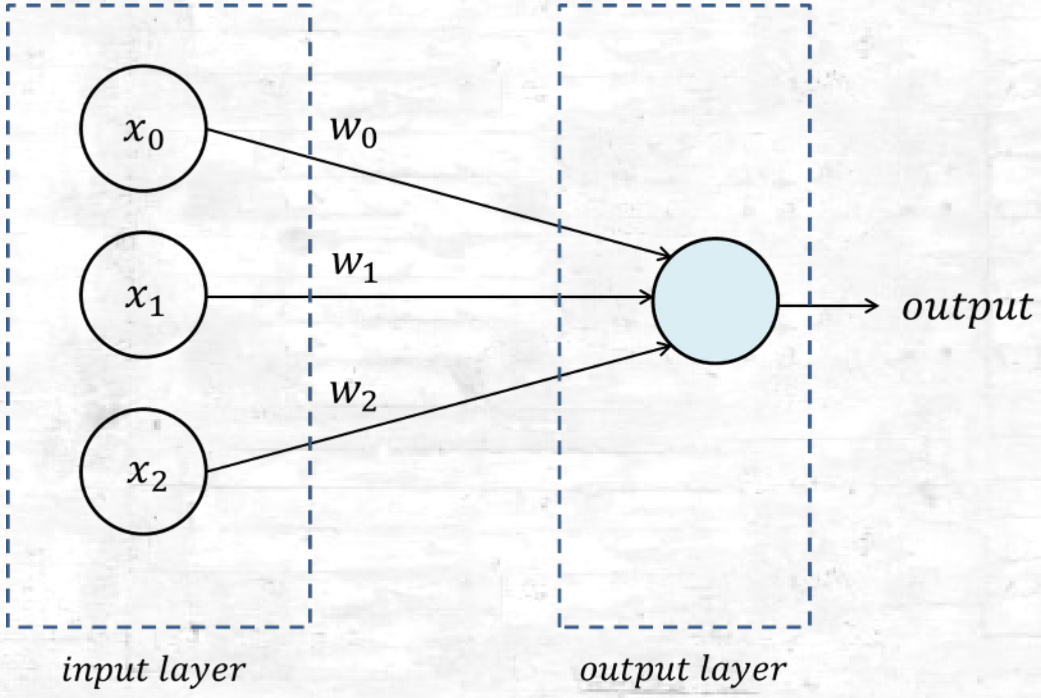
Single Layer Perceptron

Simple Deep Learning Model

First Neuromorphic Approach for solving problems

Simple and Intuitive

Basic of MLP / CNN / RNN ···



A | Single Layer Perceptron

-Main Goal [Predict Rings of Abalone]

Before The Begin…

A | Single Layer Perceptron

Keywords

- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

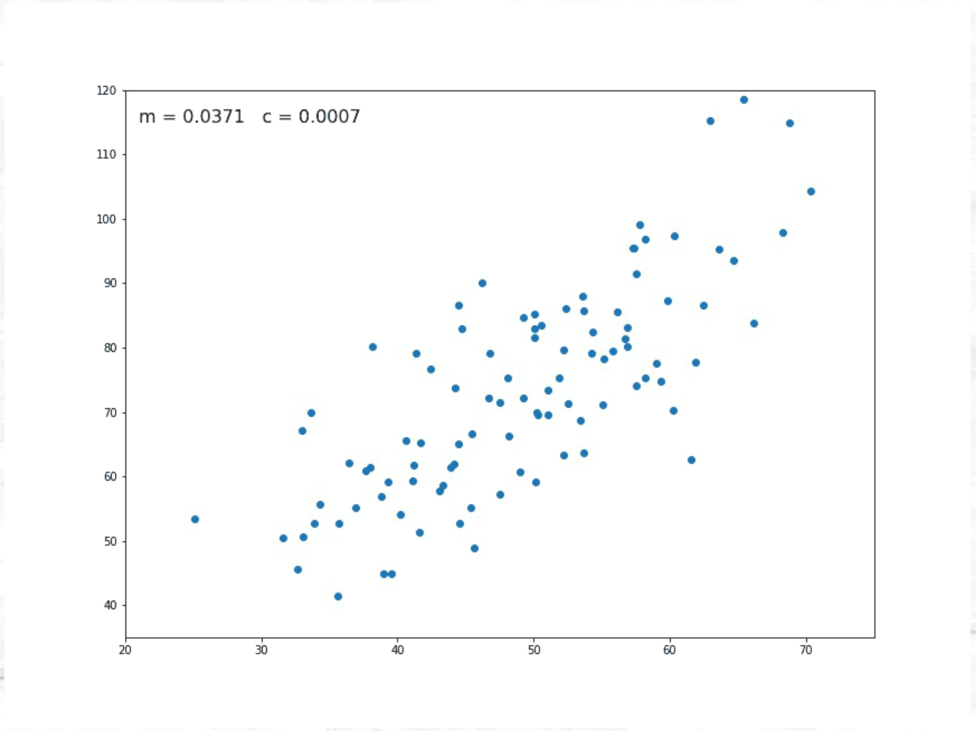
Keywords

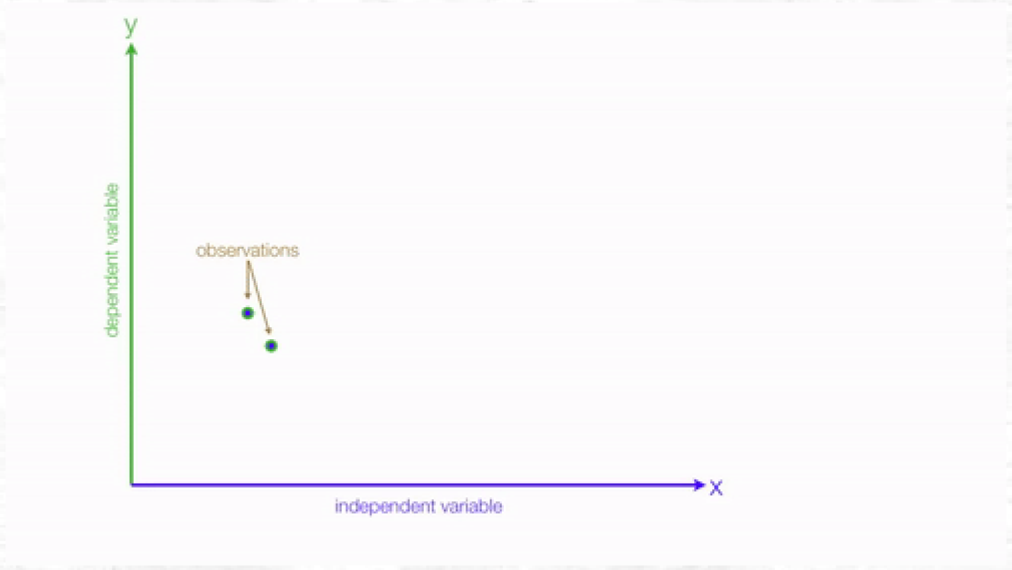
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

Regression

: Regression analysis is a set of statistical processes for estimating the relationships between a dependent variableand one or more independent variables





A | Single Layer Perceptron

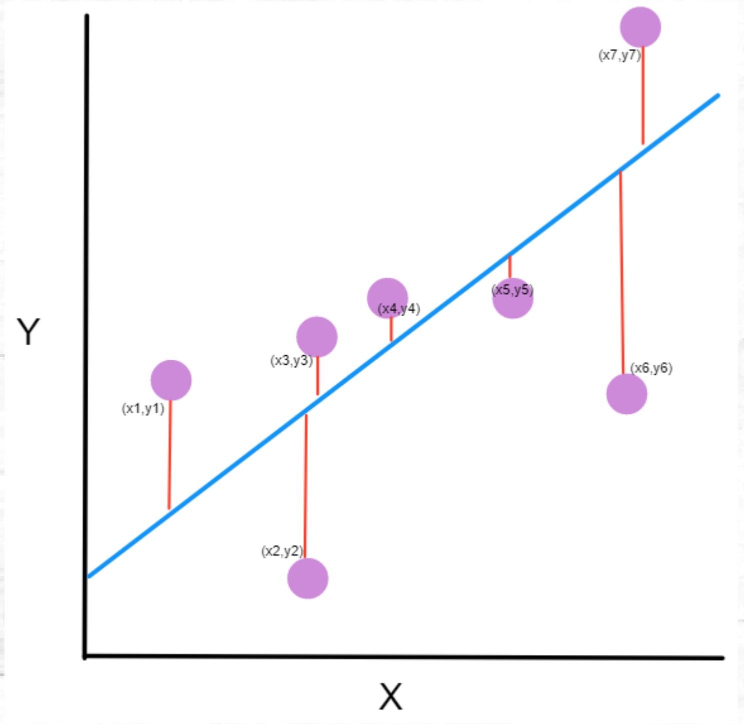
Keywords

- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

Mean Square Error

:MSE(Mean Square Error) used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed.



$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

A | Single Layer Perceptron

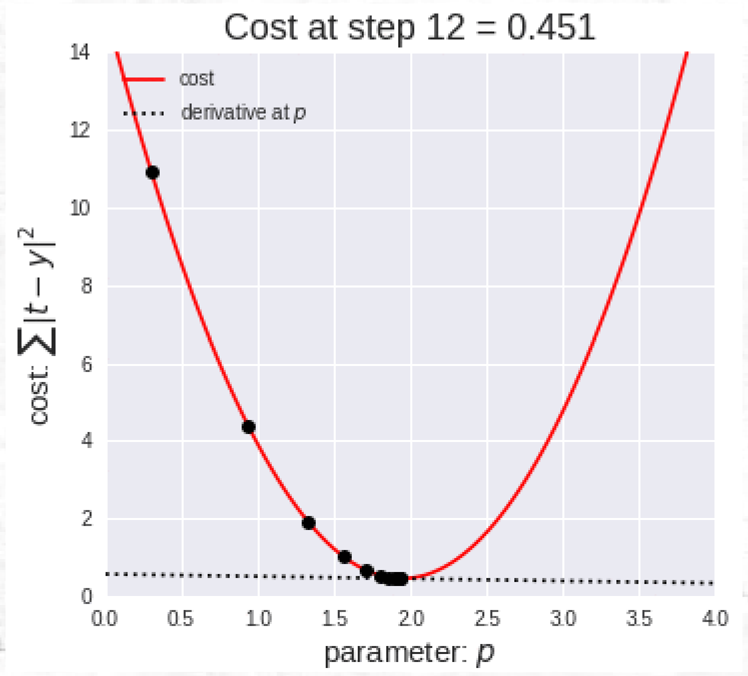
Keywords

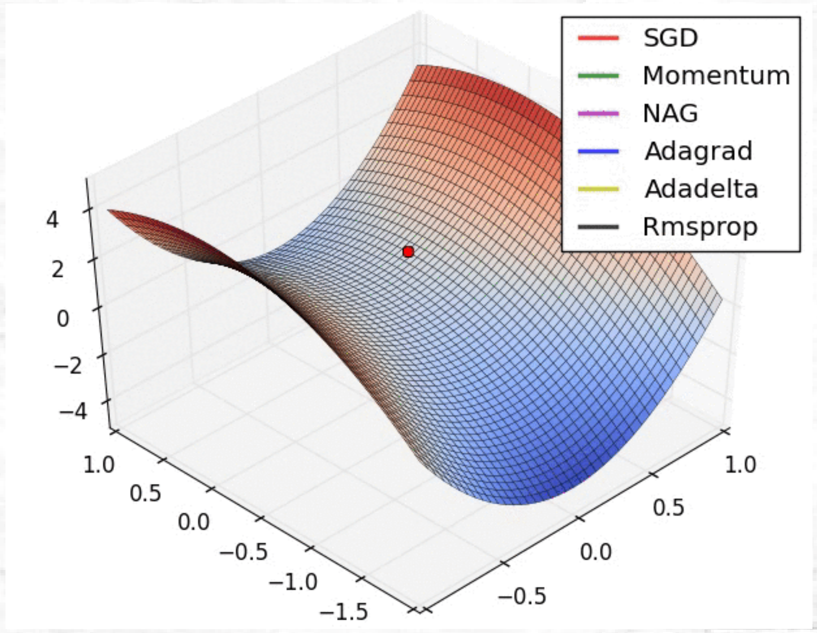
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

Loss Function (Cost Function)

: Maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function





*MSE is good cost function for Regression model

A | Single Layer Perceptron

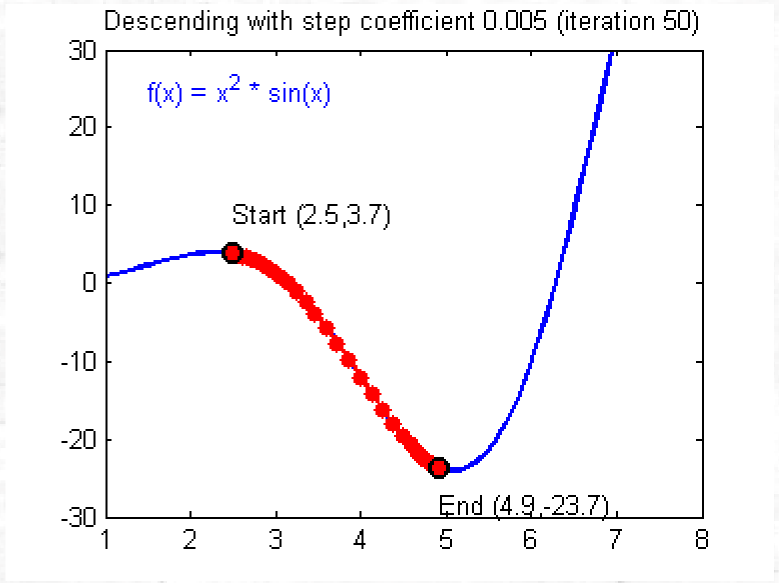
Keywords

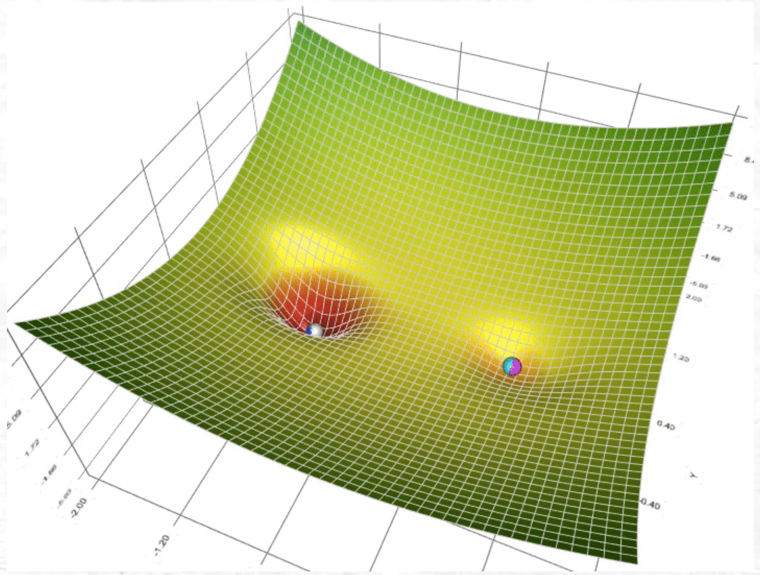
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

Gradient Descent Algorithm

: Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.



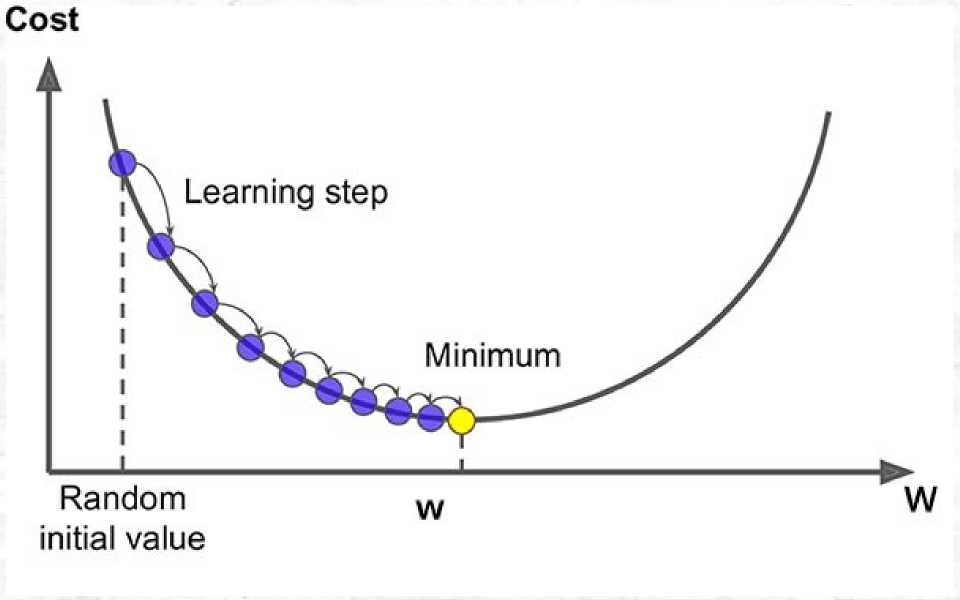


*MSE is good cost function for Regression model

A | Single Layer Perceptron

Gradient Descent Algorithm

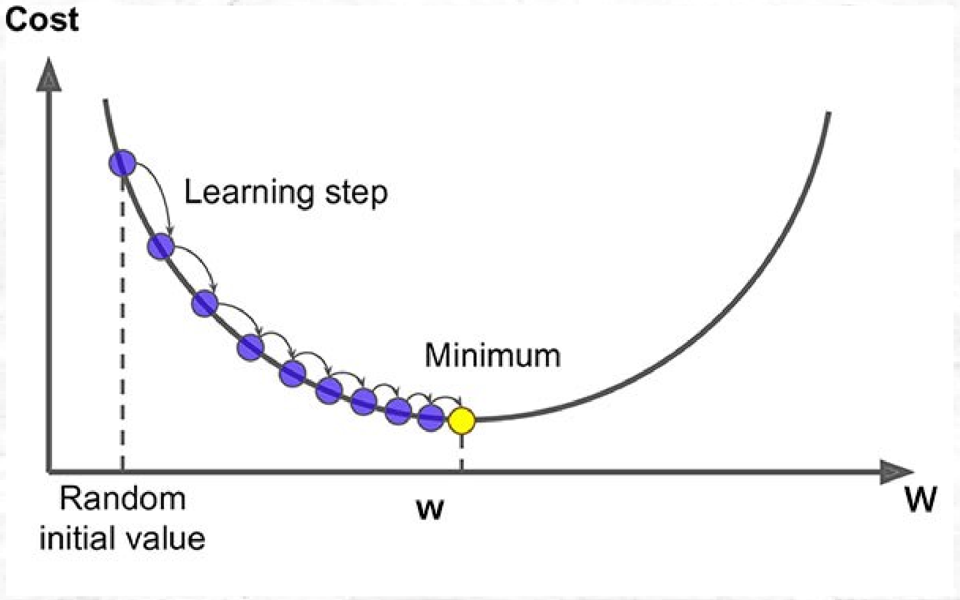
: Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.



$$x_{i+1} = x_i - \alpha \frac{\partial f(x)}{\partial x}$$

Gradient Descent Algorithm

: Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.



$$x_{i+1} = x_i - \alpha \frac{\partial f(x)}{\partial x}$$

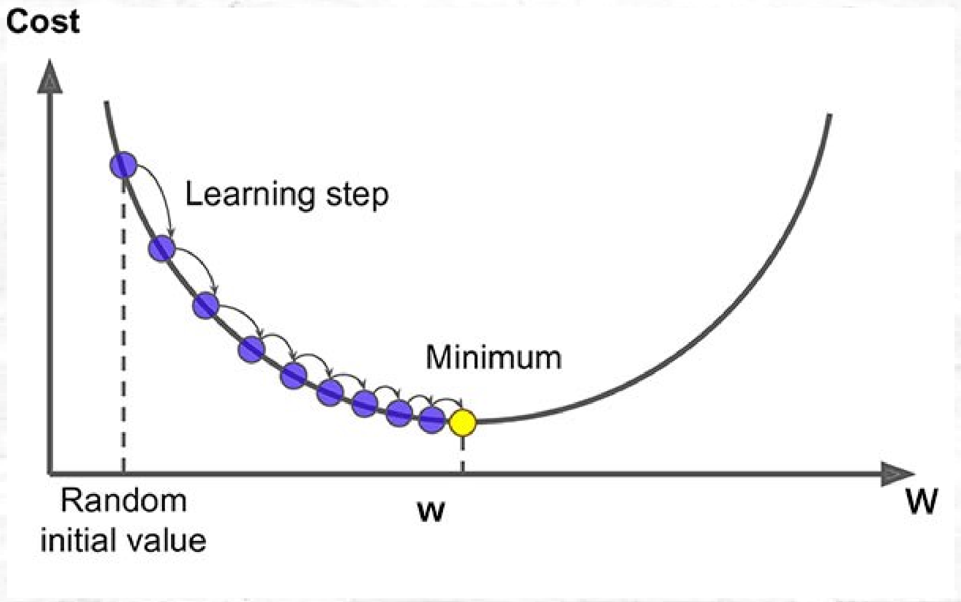
Why Not?

$$x_{i+1} = x_i - \alpha \frac{df(x)}{dx}$$

A | Single Layer Perceptron

Gradient Descent Algorithm

: Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.



$$x_{i+1} = x_i - \alpha \frac{\partial f(x)}{\partial x}$$

Why Not?

$$x_{i+1} = x_i - \alpha \frac{df(x)}{dx}$$

Complex

A | Single Layer Perceptron

Keywords

Regression

Mean Square Error

Loss Function

Gradient Descent Algorithm

Backward Propagation

Partial derivative

Hyperparameter

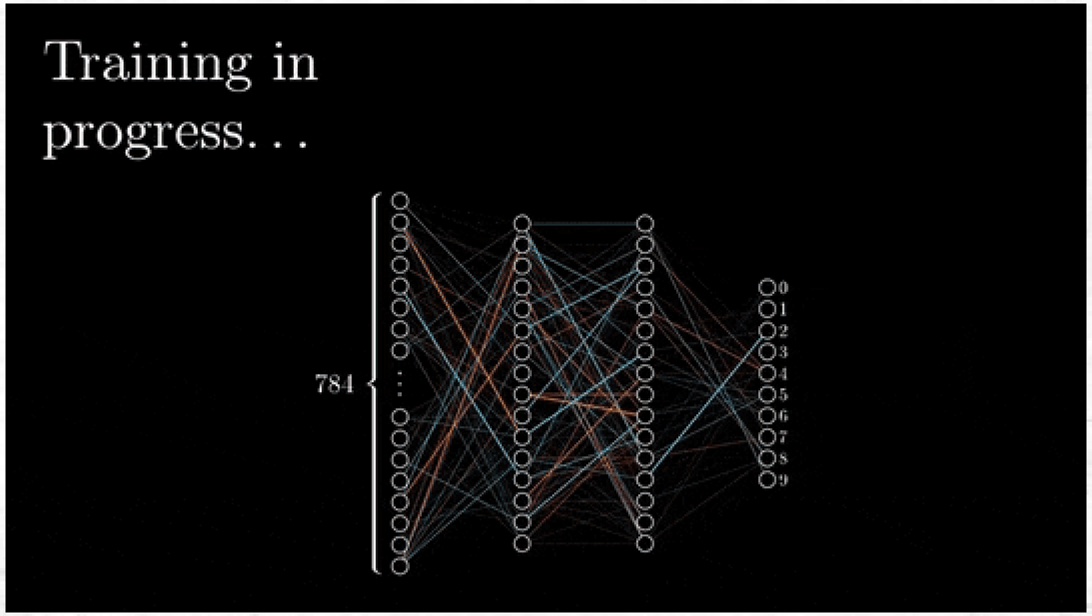
Non-linear Information

One-hot Vector

A | Single Layer Perceptron

Backward Propagation

: Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.



$$Loss\ Function\ Gradient\ = \frac{\partial L}{\partial x}$$

A | Single Layer Perceptron

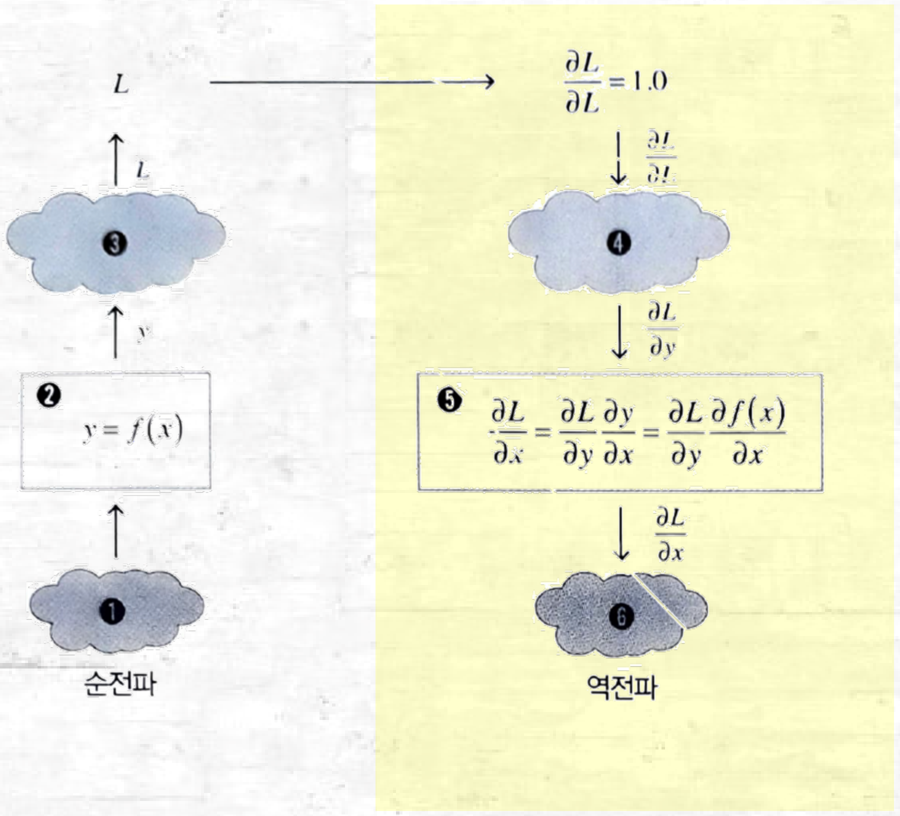
Keywords

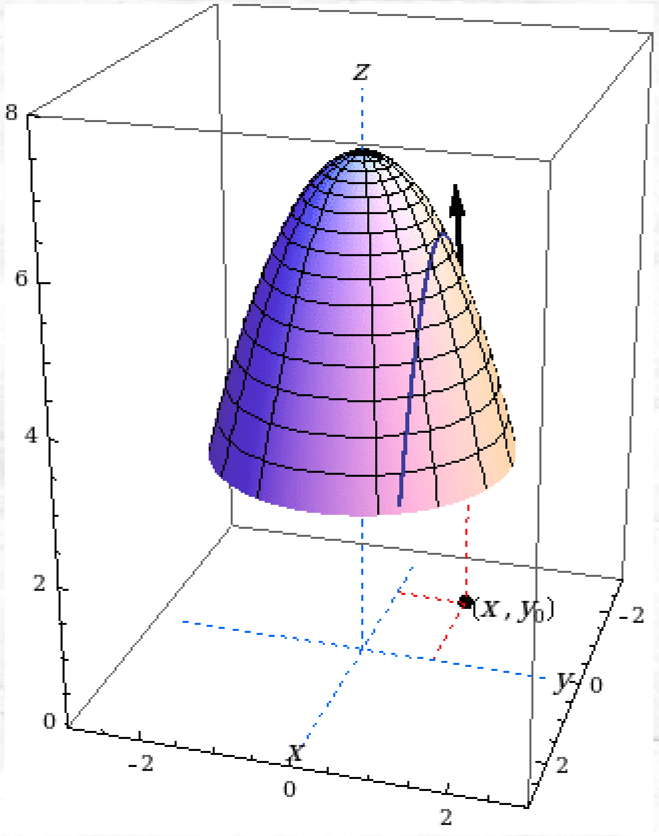
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

Partial derivative

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} \rightarrow differential\ equation$$





A | Single Layer Perceptron

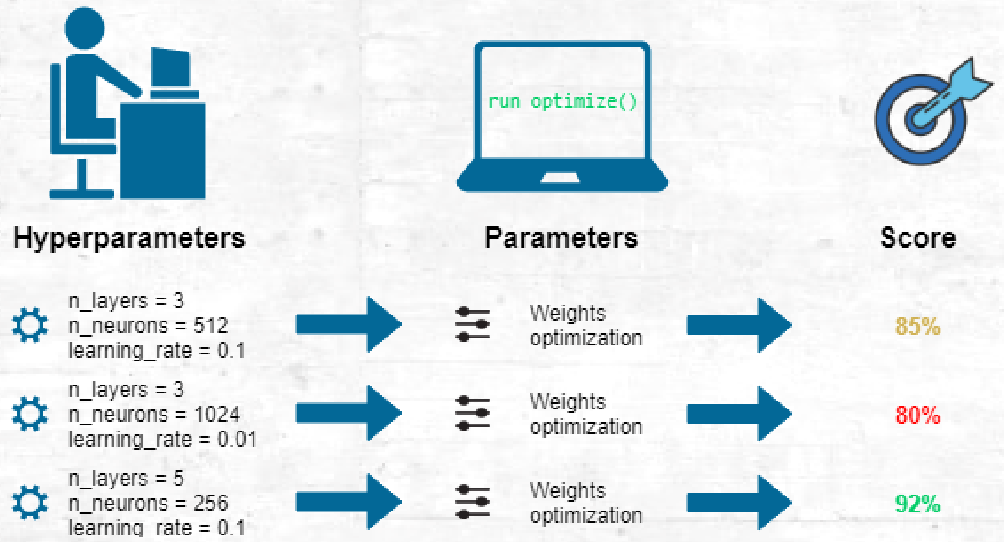
Keywords

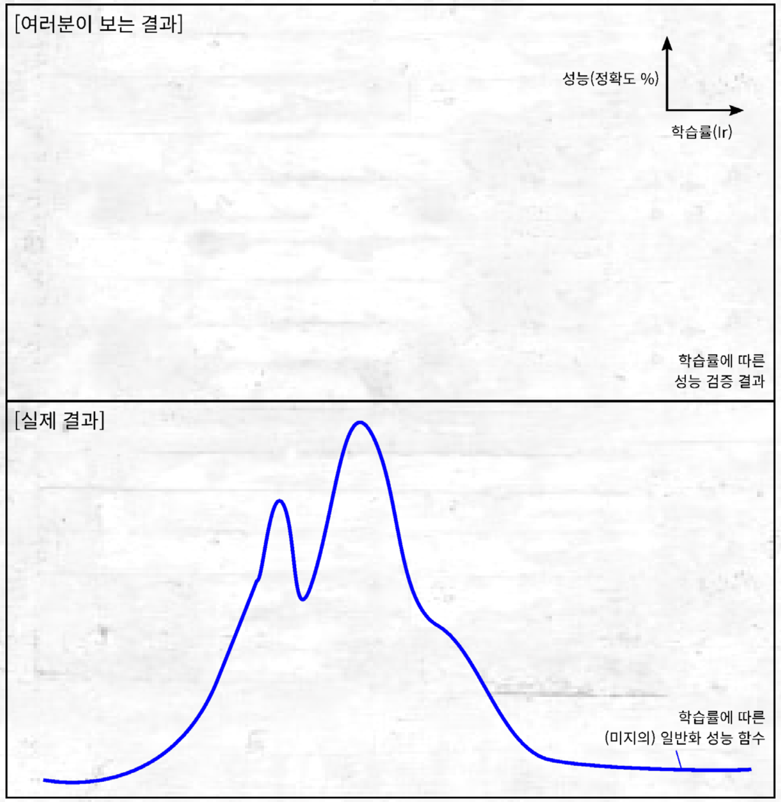
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

A | Single Layer Perceptron

Hyperparameter

: hyperparameter is a parameter whose value is used to control the learning process.
By contrast, the values of other parameters (typically node weights) are derived via training.





A | Single Layer Perceptron

Keywords

Regression

Mean Square Error

Loss Function

Gradient Descent Algorithm

Backward Propagation

Partial derivative

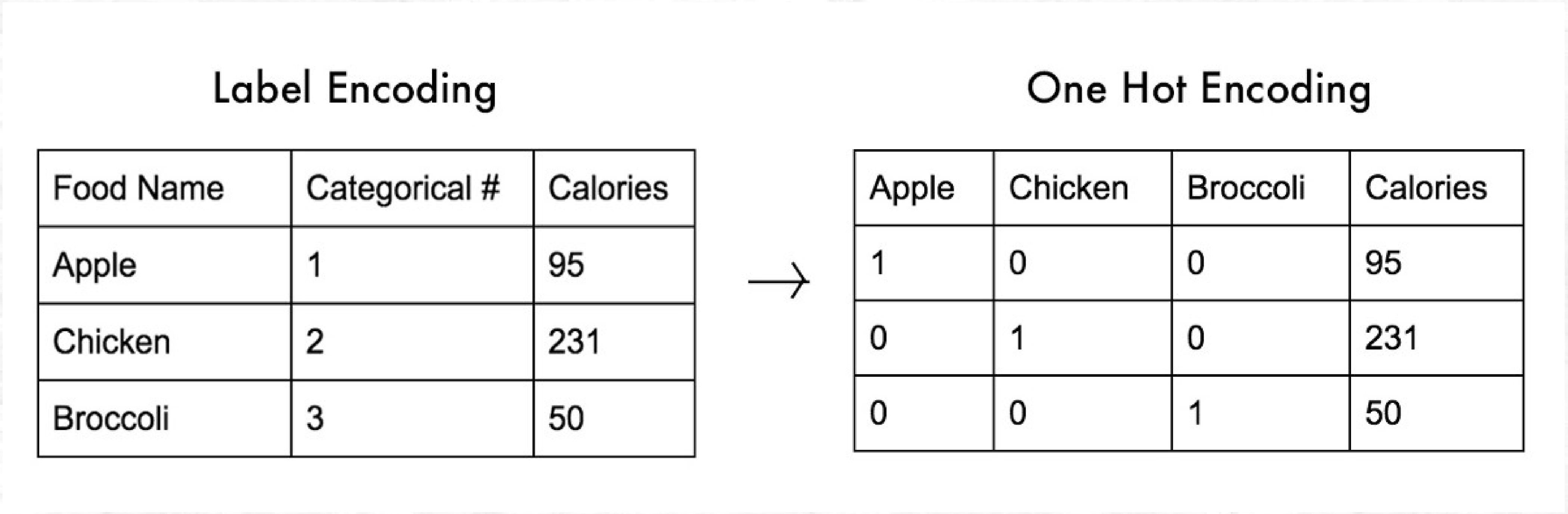
Hyperparameter

Non-linear Information

One-hot Vector

A | Single Layer Perceptron

Non-linear Information & One-hot Vector



Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

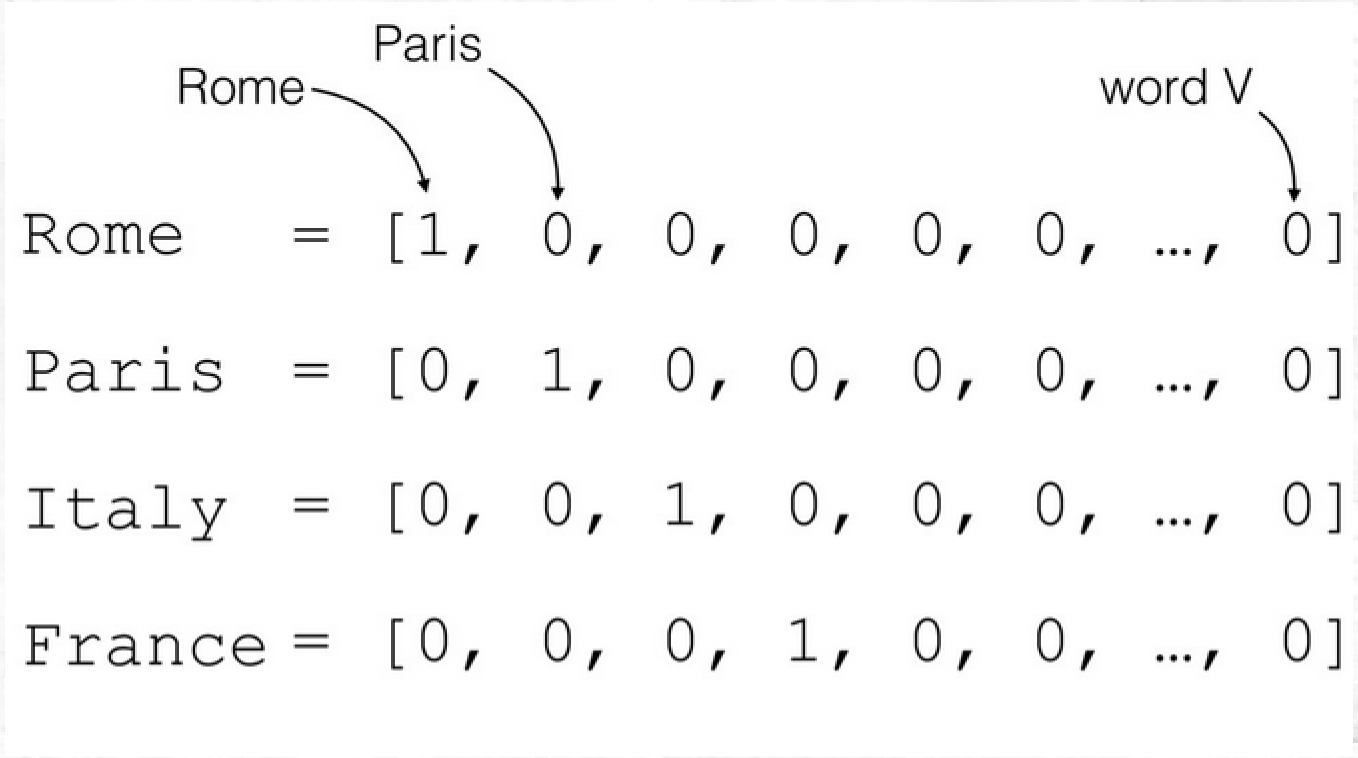
$\rightarrow$

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

A | Single Layer Perceptron

Non-linear Information & One-hot Vector



```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

A | Single Layer Perceptron

-Main Goal [Predict Rings of Abalone]

Welcome Back!

A | Single Layer Perceptron

-Main Goal [Predict Rings of Abalone]

Dataset

Abalone

https://archive.ics.uci.edu/ml/datasets/Abalone

maik3141 • updated 3 years ago (Version 1)

Name	Data Type	Measurement Unit	Description
Sex	nominal	--	M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer	--	+1.5 gives the age in years

A | Single Layer Perceptron

-Main Goal [Predict Rings of Abalone]

Abalone

https://archive.ics.uci.edu/ml/datasets/Abalone

maik3141 • updated 3 years ago (Version 1)

Name	/ Data Type	/ Measurement Unit	/ Description
Sex	/ nominal	/ --	/ M, F, and I (infant)
Length	/ continuous	/ mm	/ Longest shell measurement
Diameter	/ continuous	/ mm	/ perpendicular to length
Height	/ continuous	/ mm	/ with meat in shell
Whole weight	/ continuous	/ grams	/ whole abalone
Shucked weight	/ continuous	/ grams	/ weight of meat
Viscera weight	/ continuous	/ grams	/ gut weight (after bleeding)
Shell weight	/ continuous	/ grams	/ after being dried
Rings	/ integer	/ --	/ +1.5 gives the age in years

A | Single Layer Perceptron

- Implement

Abablone_exec

```
import numpy as np
import csv
import time

np.random.seed(1234)
def randomize(): np.random.seed(time.time())
```

```
RND_MEAN = 0
RND_STD = 0.0030

LEARNING_RATE = 0.001
```

A | Single Layer Perceptron

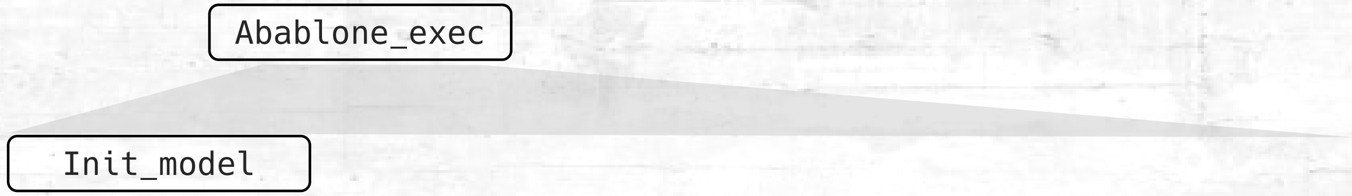
- Implement

Abablone_exec

```
def abalone_exec(epoch_count=10, mb_size=10, report=1):
    load_abalone_dataset()
    init_model()
    train_and_test(epoch_count, mb_size, report)
```

A | Single Layer Perceptron

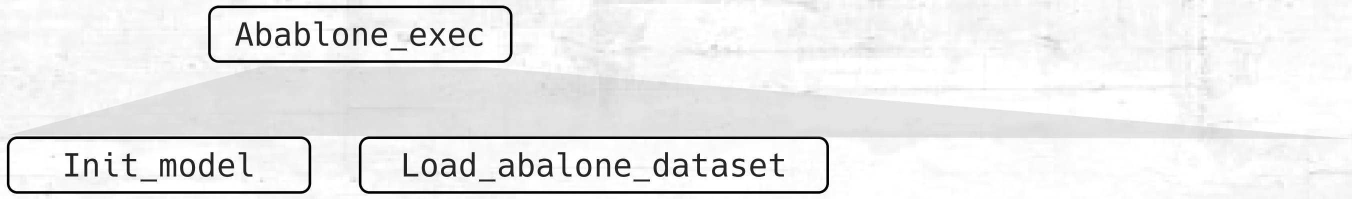
- Implement



```
def init_model():
    global weight, bias, input_cnt, output_cnt
    weight = np.random.normal(RND_MEAN, RND_STD,[input_cnt, output_cnt])
    bias = np.zeros([output_cnt])
```

A | Single Layer Perceptron

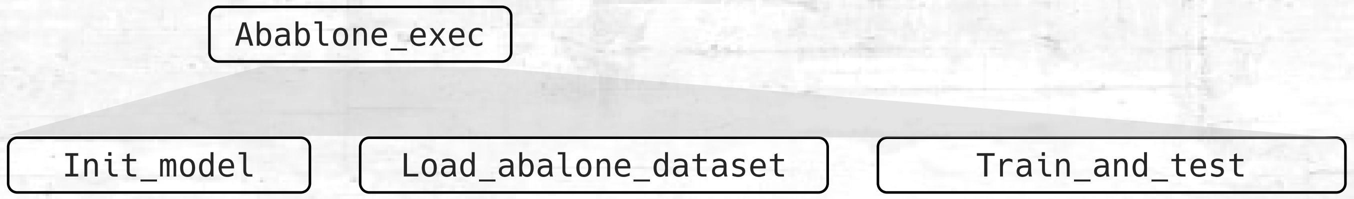
- Implement



```
def load_abalone_dataset():
    with open('../../data/chap01/abalone.csv') as csvfile:
        csvreader = csv.reader(csvfile)
        next(csvreader, None)
        rows = []
        for row in csvreader:
            rows.append(row)

    global data, input_cnt, output_cnt
    input_cnt, output_cnt = 10, 1
    data = np.zeros([len(rows), input_cnt+output_cnt])

    for n, row in enumerate(rows):
        if row[0] == 'I': data[n, 0] = 1
        if row[0] == 'M': data[n, 1] = 1
        if row[0] == 'F': data[n, 2] = 1
        data[n, 3:] = row[1:]
```

A | Single Layer Perceptron

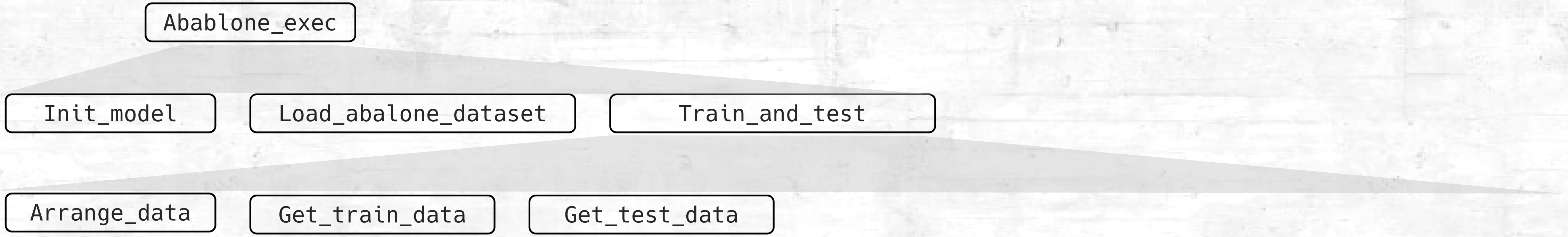
- Implement



```
def train_and_test(epoch_count, mb_size, report):
    step_count = arrange_data(mb_size)
    test_x, test_y = get_test_data()

    for epoch in range(epoch_count):
        losses, accs = [], []

        for n in range(step_count):
            train_x, train_y = get_train_data(mb_size, n)
            loss, acc = run_train(train_x, train_y)
            losses.append(loss)
            accs.append(acc)

        if report > 0 and (epoch+1) % report == 0:
            acc = run_test(test_x, test_y)
            print('Epoch {}: loss={:5.3f}, accuracy={:5.3f}/{:5.3f}'. ₩
                  format(epoch+1, np.mean(losses), np.mean(accs), acc))

    final_acc = run_test(test_x, test_y)
    print('₩nFinal Test: final accuracy = {:5.3f}'.format(final_acc))
```

A | Single Layer Perceptron

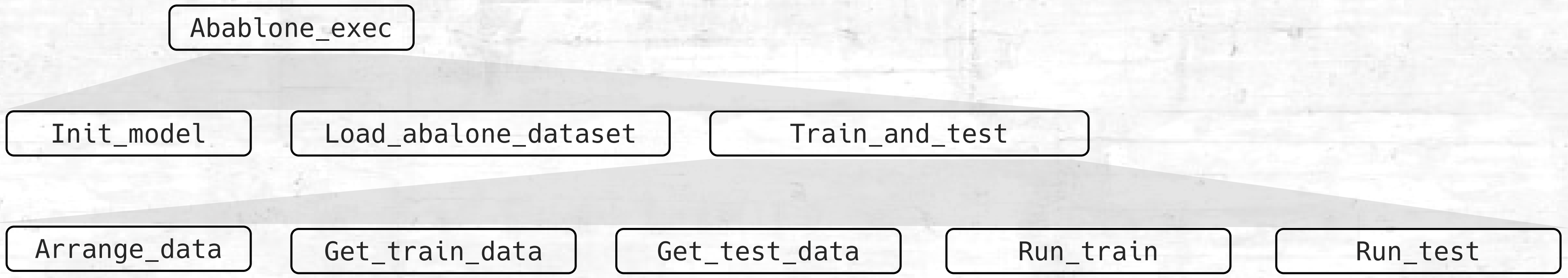
- Implement



```
def arrange_data(mb_size):
    global data, shuffle_map, test_begin_idx
    shuffle_map = np.arange(data.shape[0])
    np.random.shuffle(shuffle_map)
    step_count = int(data.shape[0] * 0.8) // mb_size
    test_begin_idx = step_count * mb_size
    return step_count

def get_test_data():
    global data, shuffle_map, test_begin_idx, output_cnt
    test_data = data[shuffle_map[test_begin_idx:]]
    return test_data[:, :-output_cnt], test_data[:, -output_cnt:]

def get_train_data(mb_size, nth):
    global data, shuffle_map, test_begin_idx, output_cnt
    if nth == 0:
        np.random.shuffle(shuffle_map[:test_begin_idx])
    train_data = data[shuffle_map[mb_size*nth:mb_size*(nth+1)]]
    return train_data[:, :-output_cnt], train_data[:, -output_cnt:]
```

A | Single Layer Perceptron

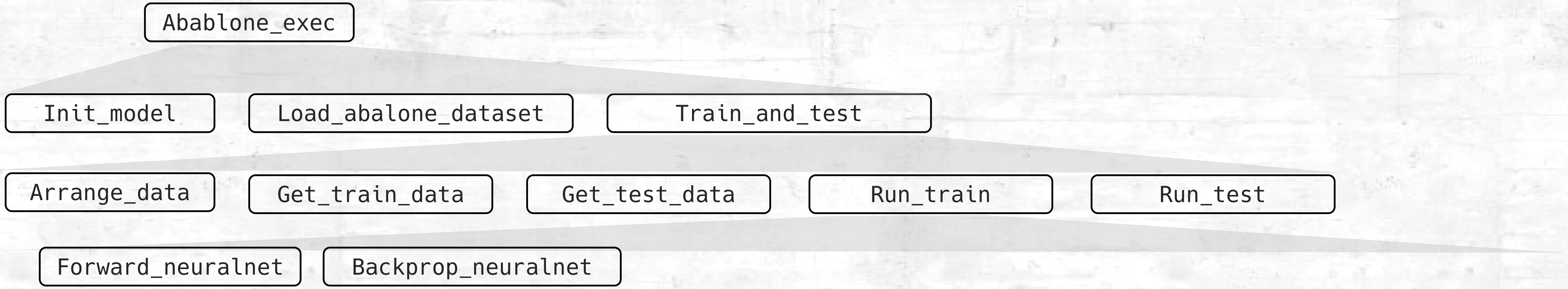
- Implement



```
def run_train(x, y):
    output, aux_nn = forward_neuralnet(x)
    loss, aux_pp = forward_postproc(output, y)
    accuracy = eval_accuracy(output, y)

    G_loss = 1.0
    G_output = backprop_postproc(G_loss, aux_pp)
    backprop_neuralnet(G_output, aux_nn)

    return loss, accuracy

def run_test(x, y):
    output, _ = forward_neuralnet(x)
    accuracy = eval_accuracy(output, y)
    return accuracy
```

A | Single Layer Perceptron

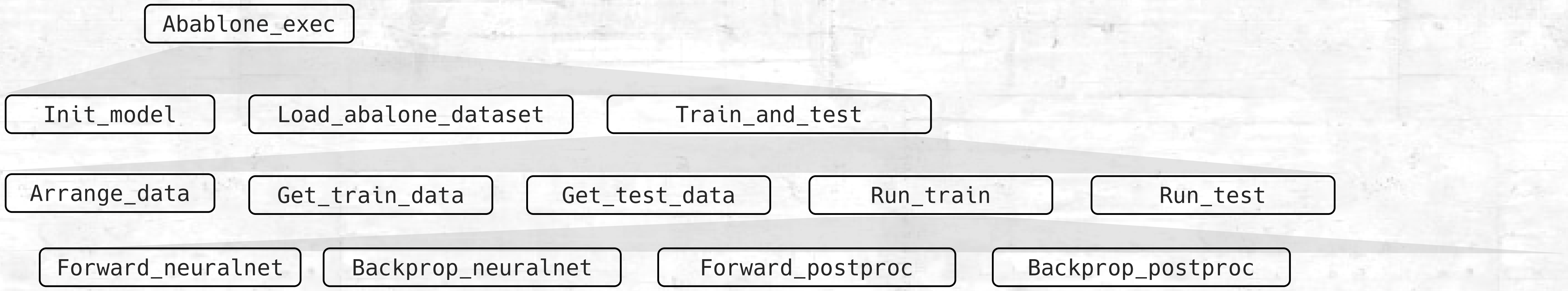
- Implement



```python
def forward_neuralnet(x):
    global weight, bias
    output = np.matmul(x, weight) + bias
    return output, x

def backprop_neuralnet(G_output, x):
    global weight, bias
    g_output_w = x.transpose()

    G_w = np.matmul(g_output_w, G_output)
    G_b = np.sum(G_output, axis=0)

    weight -= LEARNING_RATE * G_w
    bias -= LEARNING_RATE * G_b
```

A | Single Layer Perceptron

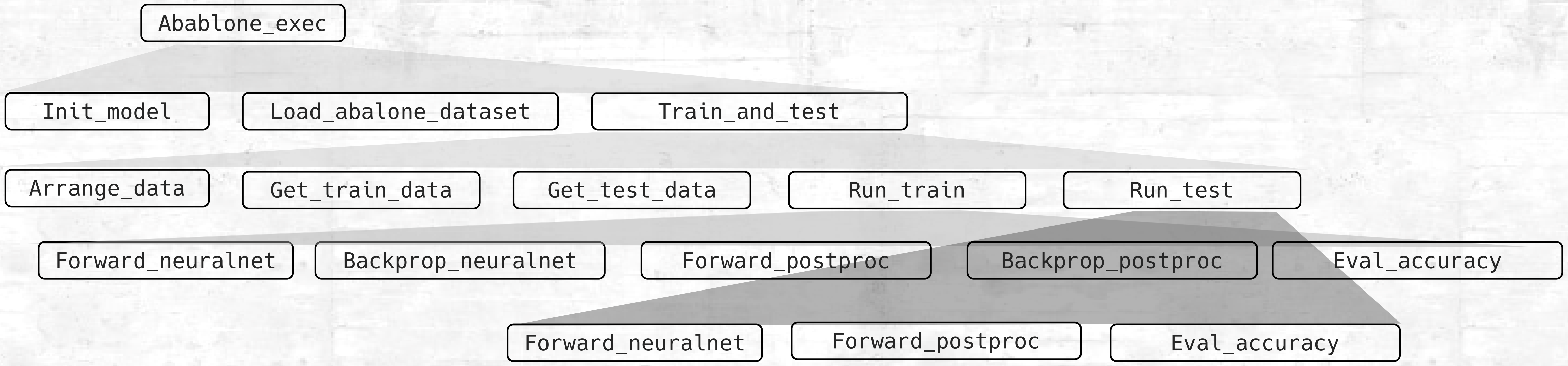
- Implement



```python
def forward_postproc(output, y):
    diff = output - y
    square = np.square(diff)
    loss = np.mean(square)
    return loss, diff

def backprop_postproc(G_loss, diff):
    shape = diff.shape

    g_loss_square = np.ones(shape) / np.prod(shape)
    g_square_diff = 2 * diff
    g_diff_output = 1

    G_square = g_loss_square * G_loss
    G_diff = g_square_diff * G_square
    G_output = g_diff_output * G_diff

    return G_output
```

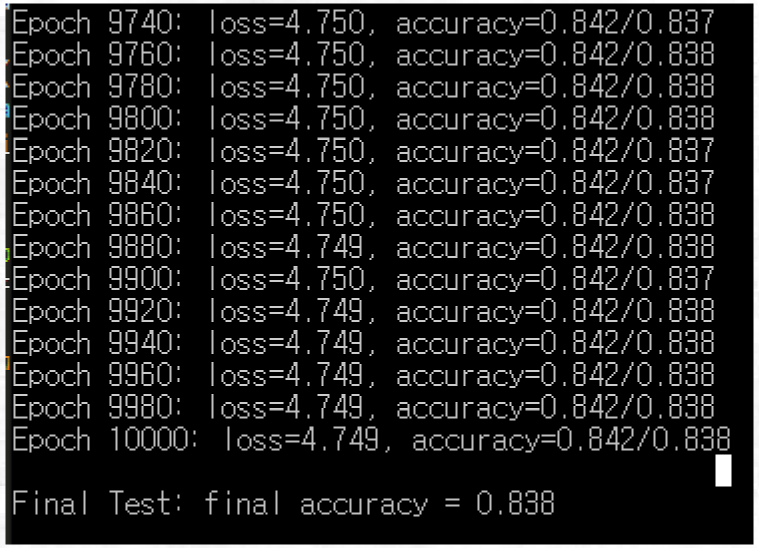
A | Single Layer Perceptron

- Implement



A | Single Layer Perceptron

- Implement



```
Epoch 9740: loss=4.750, accuracy=0.842/0.837
Epoch 9760: loss=4.750, accuracy=0.842/0.838
Epoch 9780: loss=4.750, accuracy=0.842/0.838
Epoch 9800: loss=4.750, accuracy=0.842/0.838
Epoch 9820: loss=4.750, accuracy=0.842/0.837
Epoch 9840: loss=4.750, accuracy=0.842/0.837
Epoch 9860: loss=4.750, accuracy=0.842/0.838
Epoch 9880: loss=4.749, accuracy=0.842/0.838
Epoch 9900: loss=4.750, accuracy=0.842/0.837
Epoch 9920: loss=4.749, accuracy=0.842/0.838
Epoch 9940: loss=4.749, accuracy=0.842/0.838
Epoch 9960: loss=4.749, accuracy=0.842/0.838
Epoch 9980: loss=4.749, accuracy=0.842/0.838
Epoch 10000: loss=4.749, accuracy=0.842/0.838

Final Test: final accuracy = 0.838
```

DeepUser

Single Layer Perceptron

THANKS