Concrete Machine Learning

Deep User : 2020 Summer Program

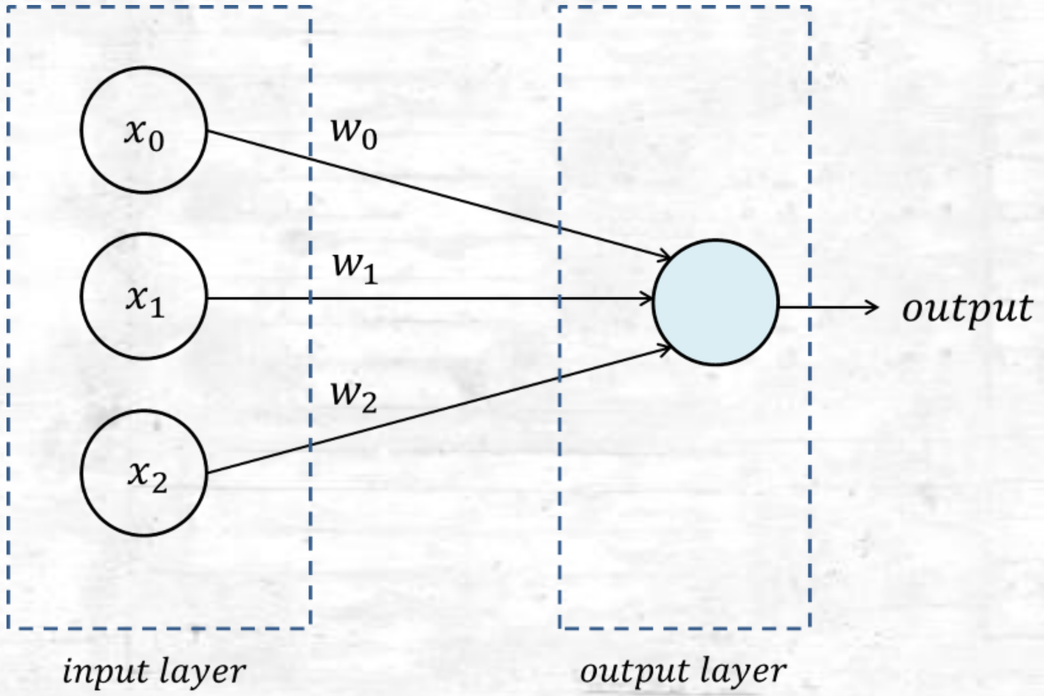
Single Layer Perceptron

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

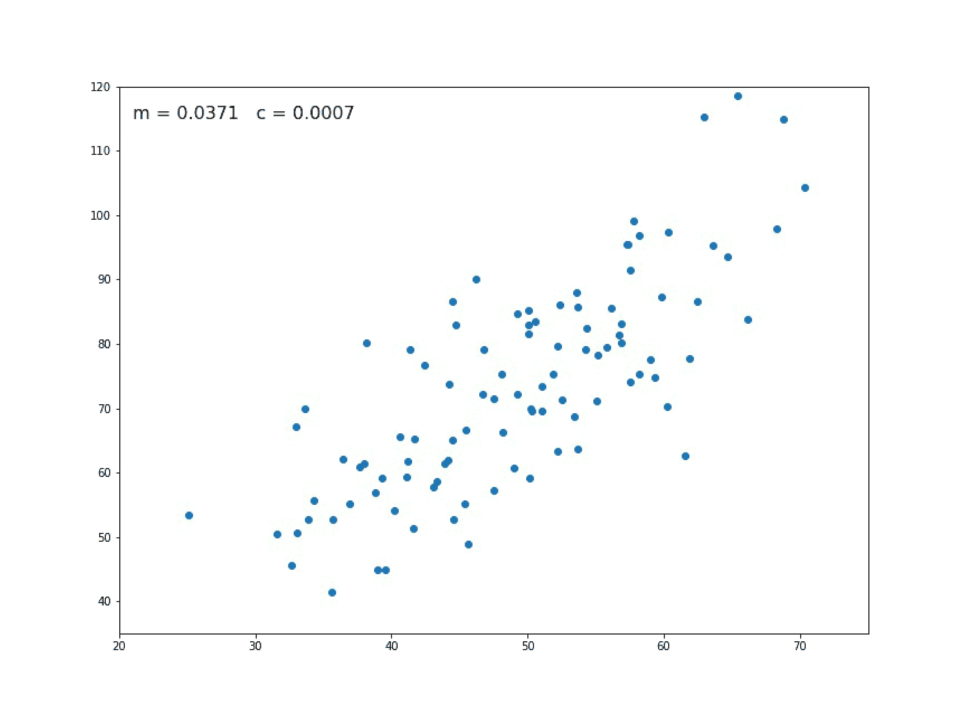
Keywords

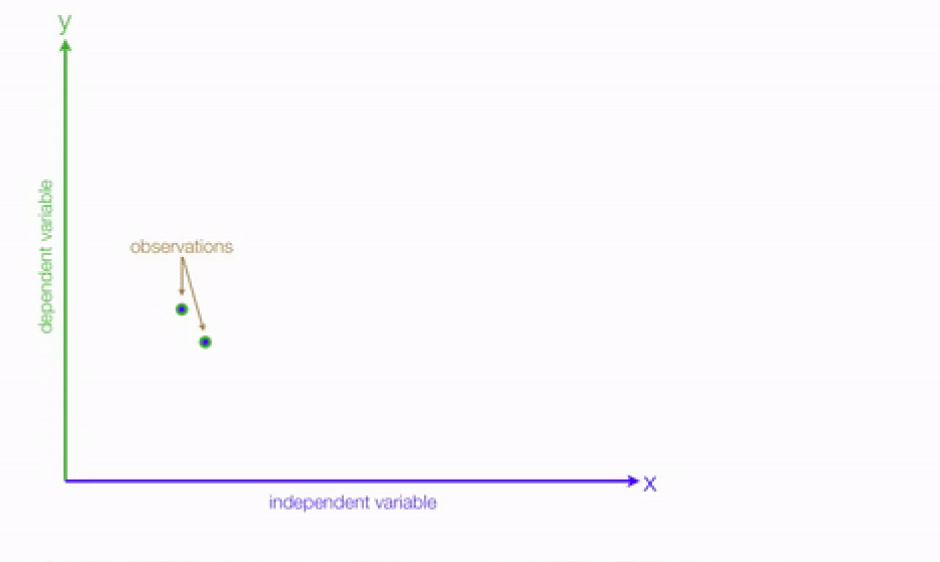
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

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

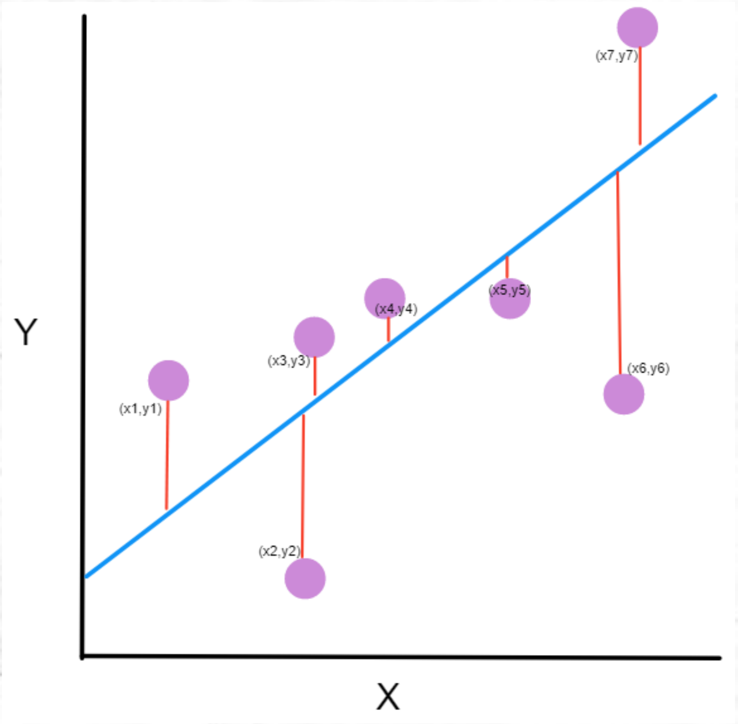
Non-linear Information

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.



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

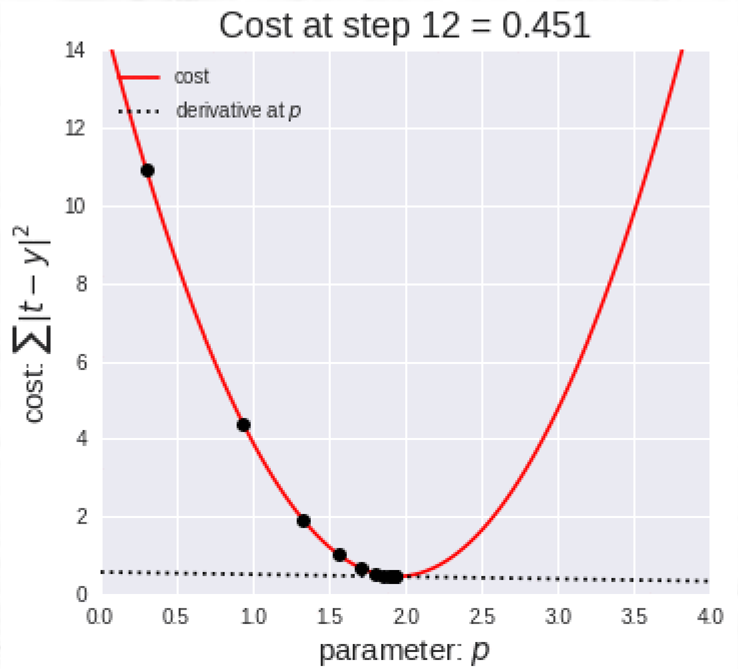
A | Single Layer Perceptron

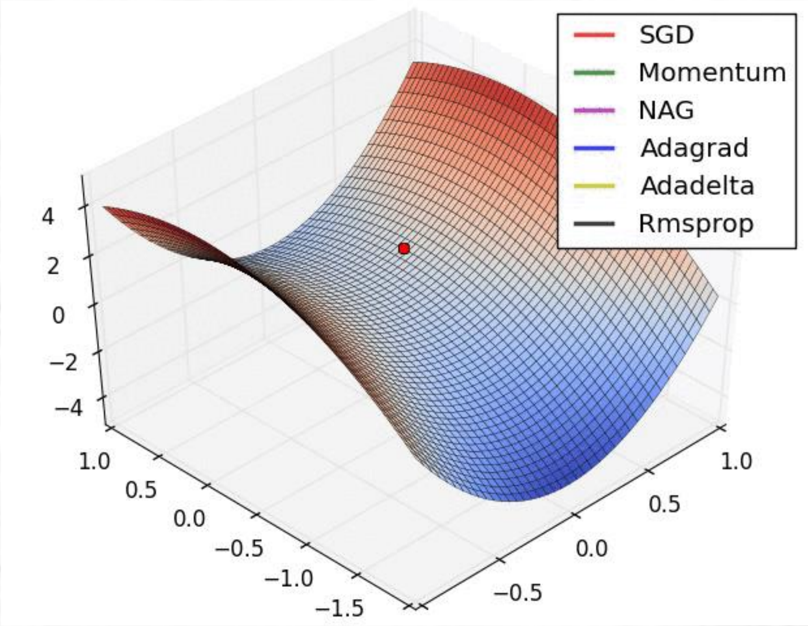
Keywords

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

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

Keywords

Regression

Mean Square Error

Loss Function

Gradient Descent Algorithm

Backward Propagation

Partial derivative

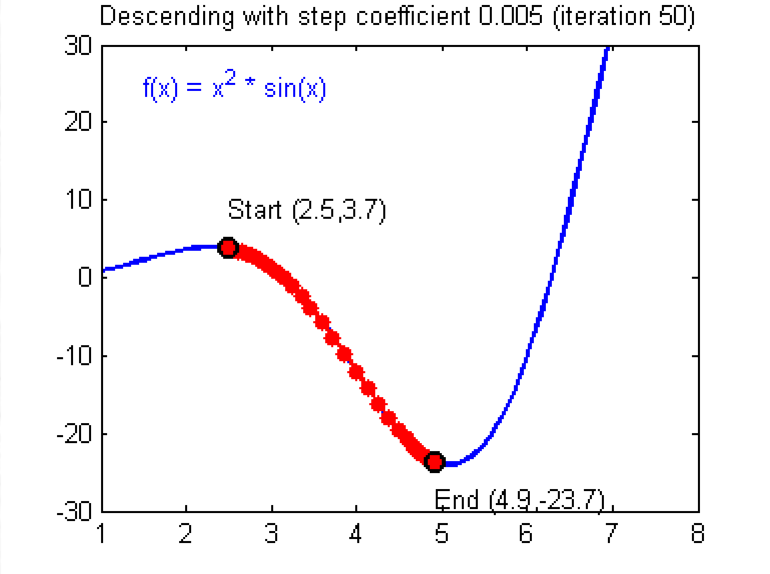
Hyperparameter

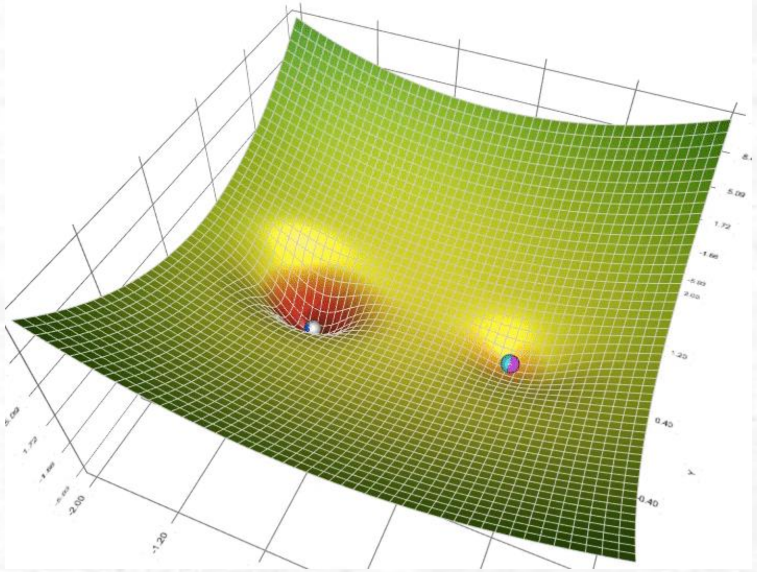
Non-linear Information

One-hot Vector

Gradient Descent Algorithm

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

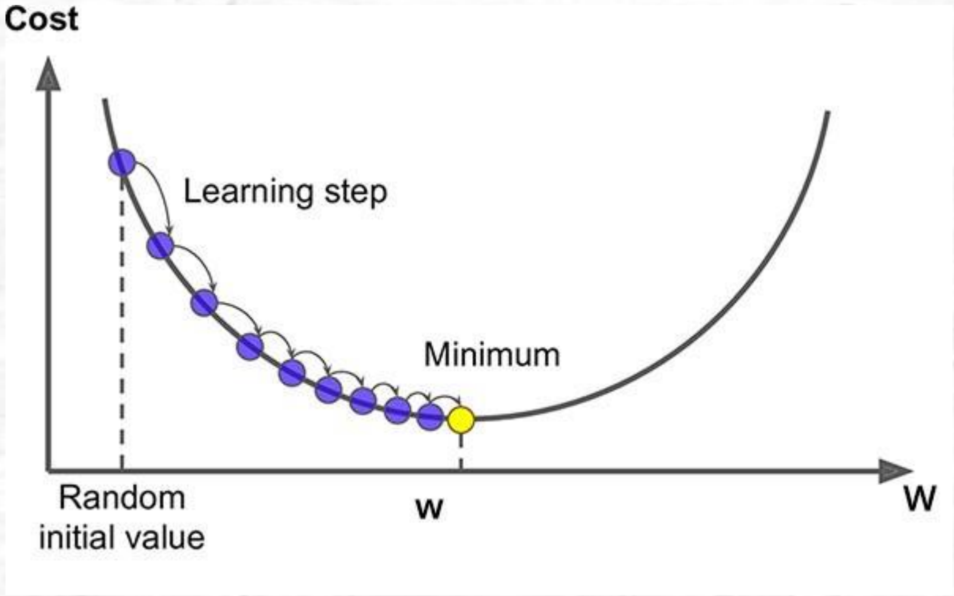




*MSE is good cost function for Regression model

Gradient Descent Algorithm

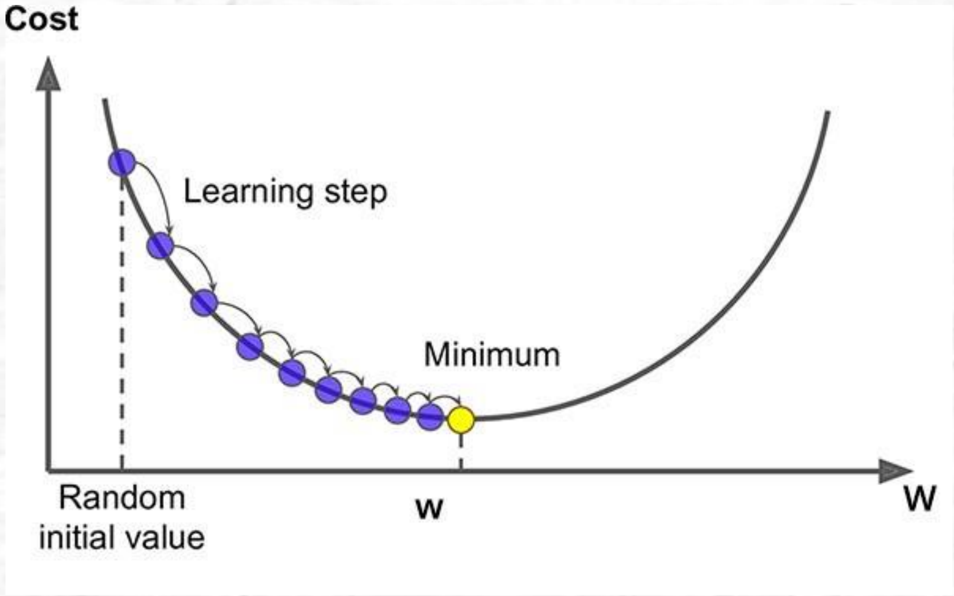
: Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.



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

Gradient Descent Algorithm

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



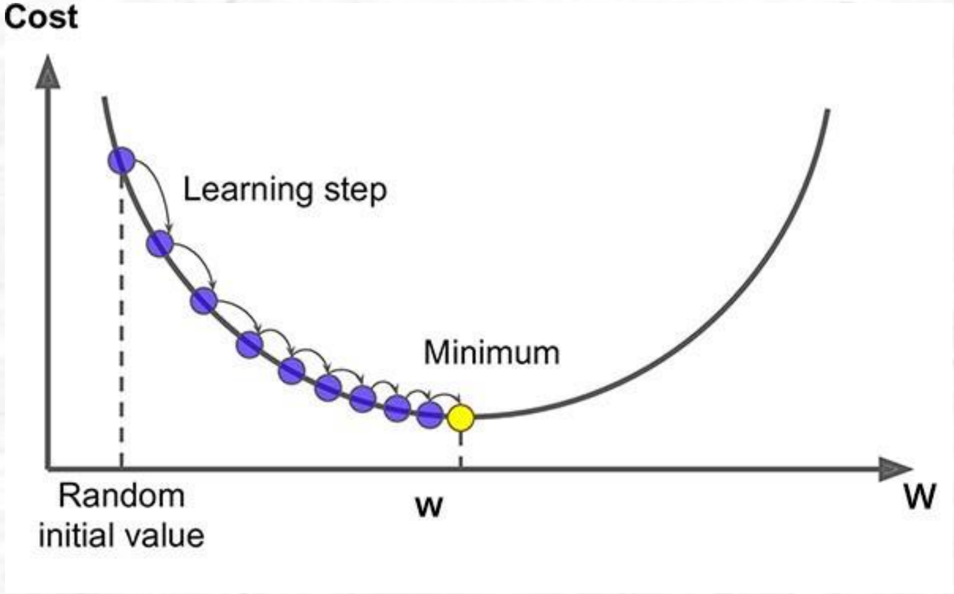
$$x_{i+1} = x_i - \alpha \frac{\partial f(x)}{\partial x}$$

Why Not?

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

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

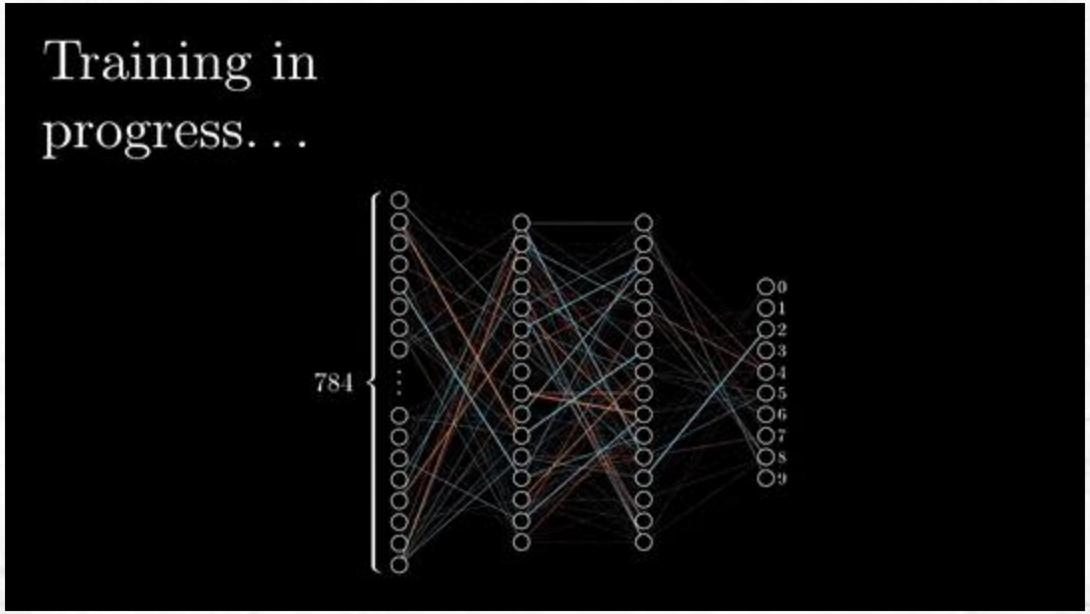
A | Single Layer Perceptron

Keywords

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

Backward Propagation

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



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

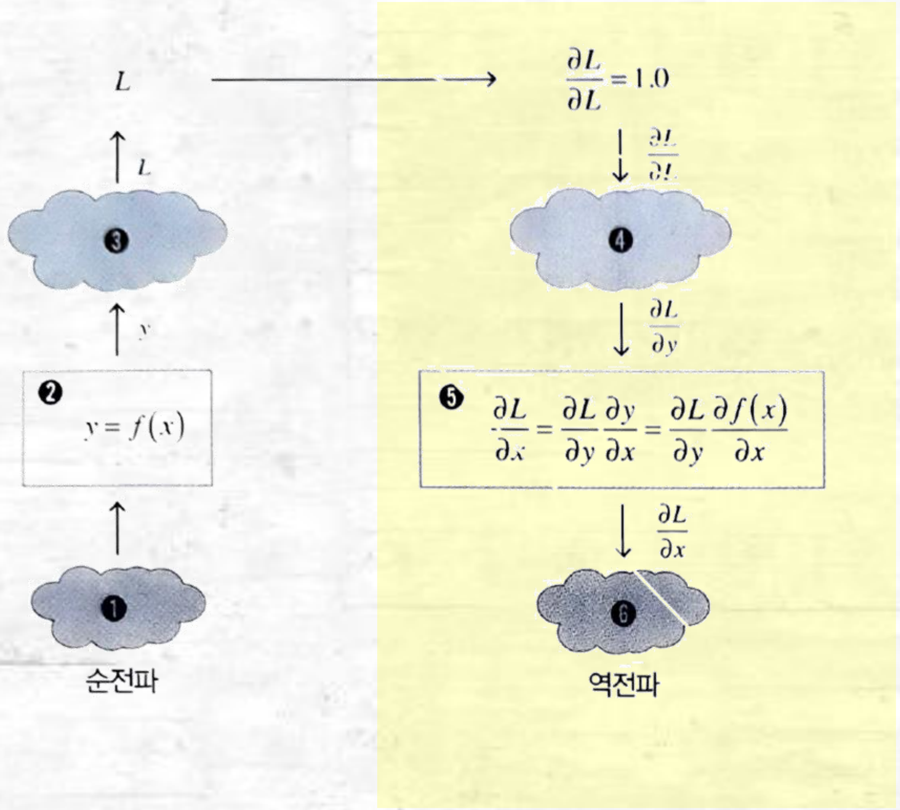
A | Single Layer Perceptron

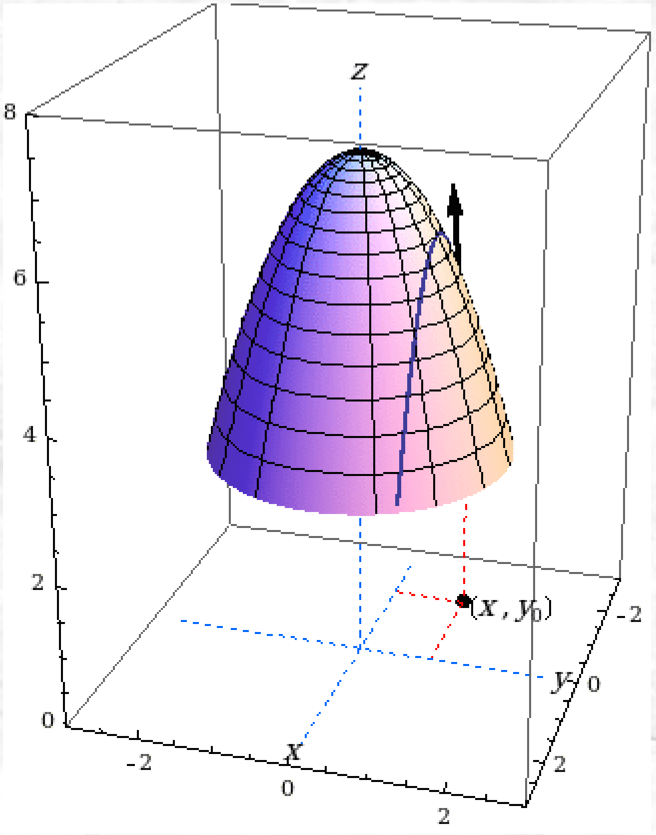
Keywords

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

Partial derivative

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





A | Single Layer Perceptron

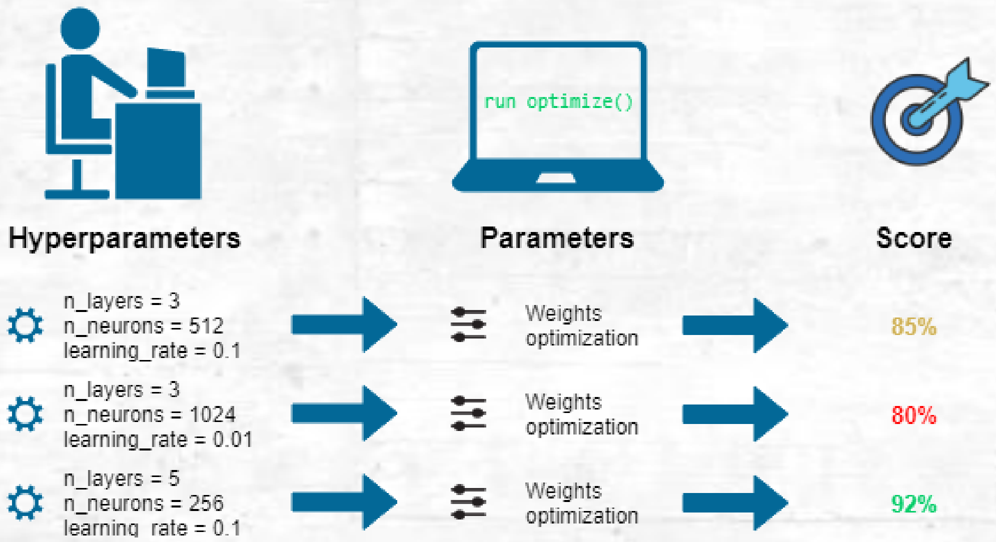
Keywords

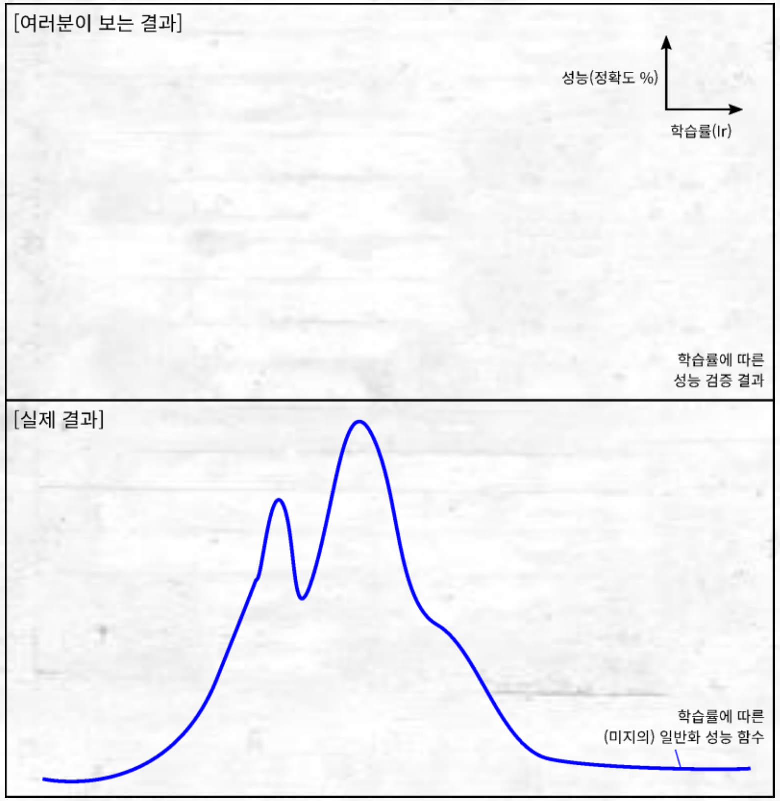
- Regression
- Mean Square Error
- Loss Function
- Gradient Descent Algorithm
- Backward Propagation
- Partial derivative
- Hyperparameter
- Non-linear Information
- One-hot Vector

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.





Keywords

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

Non-linear Information & One-hot Vector

Label Encoding

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

→

One Hot Encoding

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

Non-linear Information & One-hot Vector

$$\begin{array}{lcccccccc}
 & \text{Rome} & \text{Paris} & & & & & & \text{word } V \\
\text{Rome} = & [1, & 0, & 0, & 0, & 0, & 0, & \dots, & 0] \\
\text{Paris} = & [0, & 1, & 0, & 0, & 0, & 0, & \dots, & 0] \\
\text{Italy} = & [0, & 0, & 1, & 0, & 0, & 0, & \dots, & 0] \\
\text{France} = & [0, & 0, & 0, & 1, & 0, & 0, & \dots, & 0]
\end{array}$$

A | Single Layer Perceptron

-Main Goal [Predict Rings of Abalone]

Welcome Back!

-Main Goal [Predict Rings of Abalone]

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

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

A | Single Layer Perceptron

-Main Goal [Predict Rings of Abalone]

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()                            //For load datasets
    init_model()                                      //Model Initiallize
    train_and_test(epoch_count, mb_size, report)      //Train process eval process
```

A | Single Layer Perceptron

- Implement

Ababalone_exec

Init_model

```
def init_model():                                            //Initialize model
    global weight, bias, input_cnt, output_cnt
    weight = np.random.normal(RND_MEAN, RND_STD,[input_cnt, output_cnt]) //Start point is also crucial
    bias = np.zeros([output_cnt])
```

A | Single Layer Perceptron

- Implement

Ababalone_exec

Init_model

Load_abalone_dataset

```
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):        //One-hot Encoding
        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

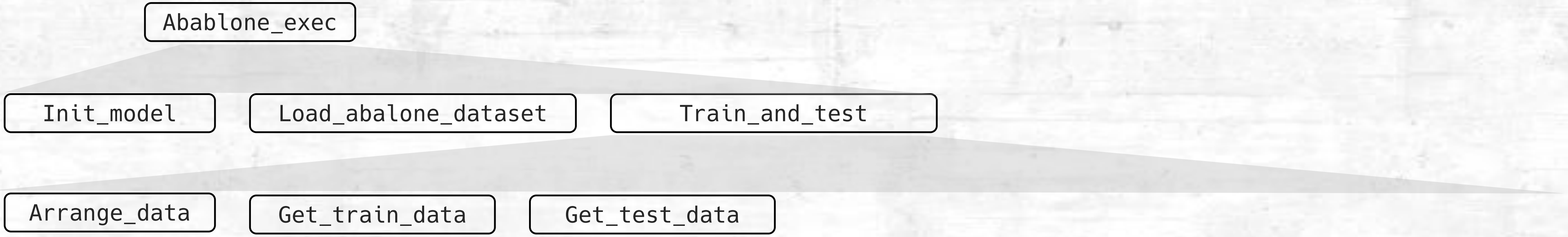
Ablablone_exec

Init_model | Load_abalone_dataset | Train_and_test

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

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

        for n in range(step_count): //Number of Minibatch
            train_x, train_y = get_train_data(mb_size, n)//Define minibatch
            loss, acc = run_train(train_x, train_y)//Actual Train step
            losses.append(loss)
            accs.append(acc)

        if report > 0 and (epoch+1) % report == 0: //Report state (when Condition is True)
            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

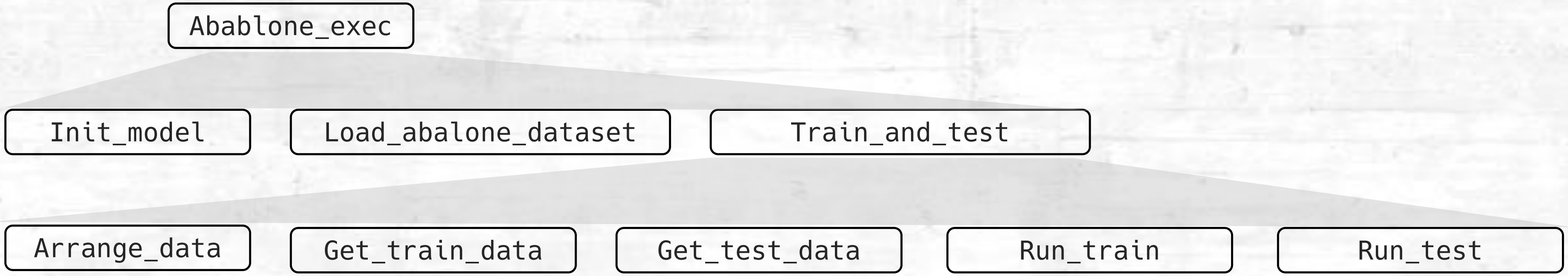
- Implement




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

def get_test_data(): //Data spliter
    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)://Data spliter
    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)    //Forward Step get output from input matrix 'x'
    loss, aux_pp = forward_postproc(output, y)  //Forward Step get loss val from input 'output' and 'y'
    accuracy = eval_accuracy(output, y) //Report Acc step

    G_loss = 1.0  //loss Gradient  ∂L/∂L = 1
    G_output = backprop_postproc(G_loss, aux_pp)  //Backward Step get 'G_output' from input 'G-loss'
    backprop_neuralnet(G_output, aux_nn)//Backward Step parameter update (Actual learning process)

    return loss, accuracy

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

A | Single Layer Perceptron

- Implement

Abablone_exec

Init_model | Load_abalone_dataset | Train_and_test

Arrange_data | Get_train_data | Get_test_data | Run_train | Run_test

Forward_neuralnet | Backprop_neuralnet

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

def backprop_neuralnet(G_output, x): //Backward Process (Get G_output [loss gradient of forward output 'output']
    global weight, bias
    g_output_w = x.transpose() //partial gradient between x and output
    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
    //subtraction by Learning Rate
    //(Ref. Partial derivative)
```

$output = x(input\ matrix) * w(weight\ matrix) + b(bias)$

$[10, N]$, *when* N *is size of mini batch*

$[10,1]$ G_w $[N,1]$

Weight matrix W loss cost gradient W [Weight Loss Gradient]

$$\frac{\partial L}{\partial B_j} = T_{k1}G_{1j} + T_{k2}G_{2j} + \cdots + T_{km}G_{mj} \rightarrow \frac{\partial L}{\partial W} = TG = X^T G \qquad {}^*T = X^T$$

bias matrix B loss cost gradient B [Bias Loss Gradient]

$$\frac{\partial L}{\partial B_j} = G_{1j} + G_{2j} + \cdots + G_{mj}$$

Get these values simply by get sum of the each G matrix's row

A | Single Layer Perceptron

- Implement

Abablone_exec

Init_model | Load_abalone_dataset | Train_and_test

Arrange_data | Get_train_data | Get_test_data | Run_train | Run_test

Forward_neuralnet | Backprop_neuralnet | Forward_postproc | Backprop_postproc

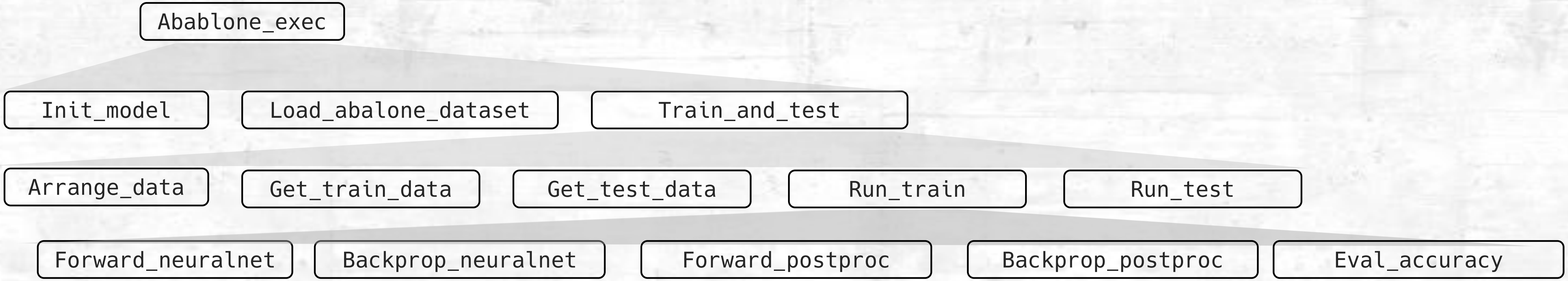
```
            def forward_postproc(output, y)://Get MSE
Matrix [N,1]    diff = output - y
Matrix [N,1]    square = np.square(diff)
Single Scalar   loss = np.mean(square)
                return loss, diff

def backprop_postproc(G_loss, diff)://Backward Process
    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 //Mean, Square, Loss backward step
    G_diff = g_square_diff * G_square
    G_output = g_diff_output * G_diff

    return G_output
```

Also Can be represent as

```
def backprop_postproc_oneline(G_loss, diff):
    return 2 * diff / np.prod(diff.shape)
```

A | Single Layer Perceptron

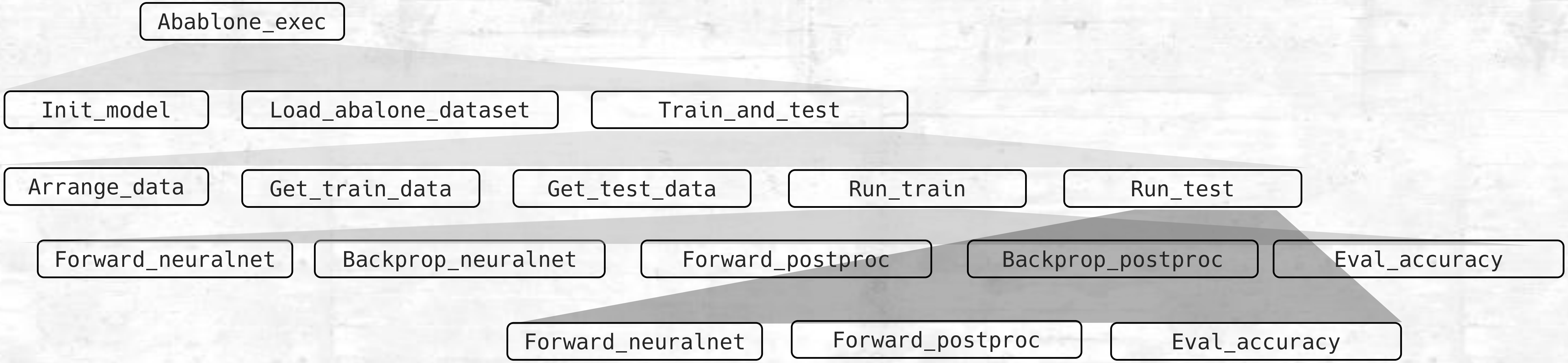
- Implement




```
def eval_accuracy(output, y): //eval process
    mdiff = np.mean(np.abs((output - y)/y))
    return 1 - mdiff
```

A | Single Layer Perceptron

- Implement



A | Single Layer Perceptron

- Implement

```
abalone_exec(epoch_count = 100, mb_size = 50, report = 20)
```

```
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
```

Predict 'Rings' label in abalone dataset
Get more than 0.80 Acc

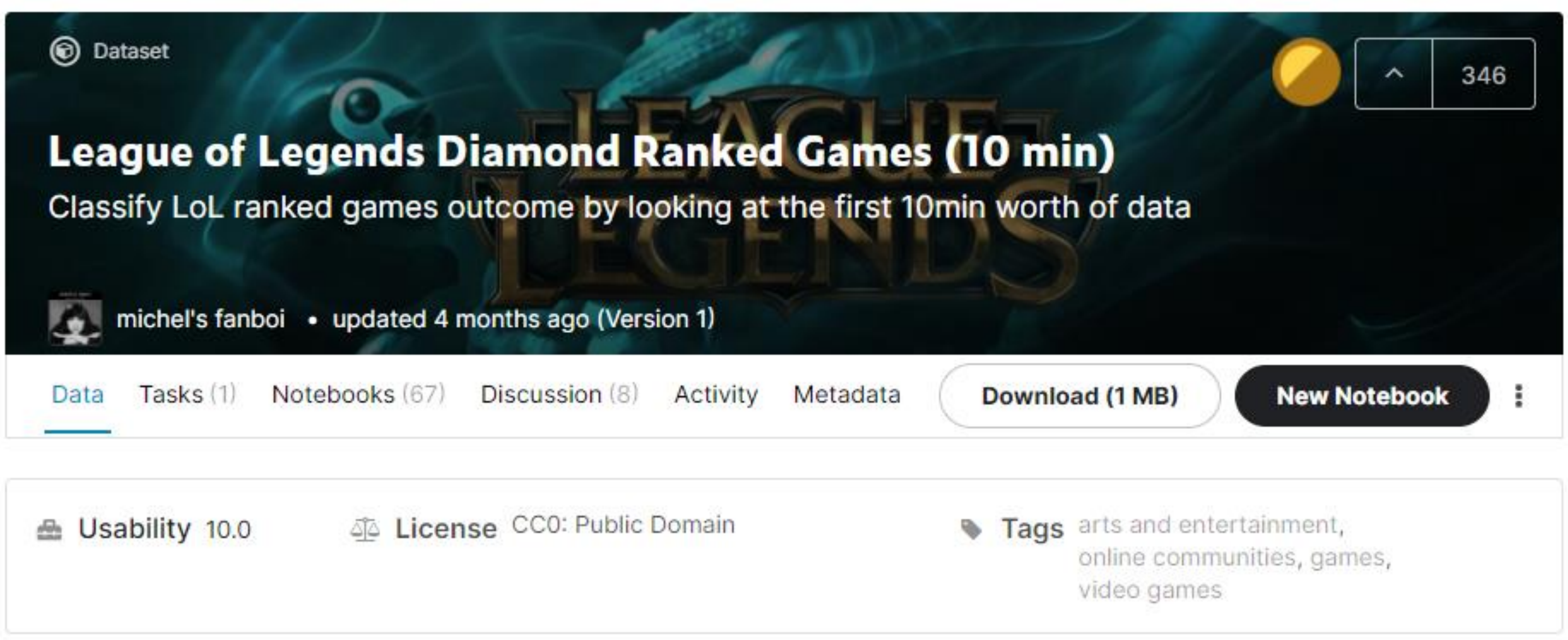
	A	B	C	D	E	F	G	H	I
1	M	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	15
2	M	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07	7
3	F	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21	9
4	M	0.44	0.365	0.125	0.516	0.2155	0.114	0.155	10
5	I	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055	7
6	I	0.425	0.3	0.095	0.3515	0.141	0.0775	0.12	8
7	F	0.53	0.415	0.15	0.7775	0.237	0.1415	0.33	20
8	F	0.545	0.425	0.125	0.768	0.294	0.1495	0.26	16
9	M	0.475	0.37	0.125	0.5095	0.2165	0.1125	0.165	9
10	F	0.55	0.44	0.15	0.8945	0.3145	0.151	0.32	19
11	F	0.525	0.38	0.14	0.6065	0.194	0.1475	0.21	14
12	M	0.43	0.35	0.11	0.406	0.1675	0.081	0.135	10
13	M	0.49	0.38	0.135	0.5415	0.2175	0.095	0.19	11
14	F	0.535	0.405	0.145	0.6845	0.2725	0.171	0.205	10
15	F	0.47	0.355	0.1	0.4755	0.1675	0.0805	0.185	10
16	M	0.5	0.4	0.13	0.6645	0.258	0.133	0.24	12

X label

Y label

A | Single Layer Perceptron

-One More! [Predict Anything in dataset]



https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-mins

A | Single Layer Perceptron

-One More! [Predict Anything in dataset]

* LOL_data.csv

	B	C	D	E	F	G	H	I	J	K	L	M
1	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDragoı	blueHerald	blueTowers	blueTotalG
2	0	28	2	1	9	6	11	0	0	0	0	17210
3	0	12	1	0	5	5	5	0	0	0	0	14712
4	0	15	0	0	7	11	4	1	1	0	0	16113
5	0	43	1	0	4	5	5	1	0	1	0	15157
6	0	75	4	0	6	6	6	0	0	0	0	16400
7	1	18	0	0	5	3	6	1	1	0	0	15899
8	1	18	3	1	7	6	7	1	1	0	0	16874
9	0	16	2	0	5	13	3	0	0	0	0	15305
10	0	16	3	0	7	7	8	0	0	0	0	16401
11	1	13	1	1	4	5	5	1	1	0	0	15057
12	0	20	3	1	4	4	6	0	0	0	0	15474
13	0	33	2	1	11	11	7	1	0	1	0	16695
14	1	18	1	1	7	1	11	1	1	0	0	17865
15	0	14	3	0	4	9	1	1	0	1	0	14979
16	1	15	3	1	4	4	4	0	0	0	0	15722
17	0	17	1	0	3	7	3	0	0	0	0	15015
18	1	14	1	1	10	2	8	0	0	0	0	19733
19	0	43	3	0	3	7	3	1	0	1	0	14852
20	1	21	4	1	5	4	11	0	0	0	0	16282
21	0	11	3	0	5	9	5	0	0	0	0	14994
22	1	14	3	1	11	6	15	1	1	0	0	18606
23	0	13	1	0	4	13	5	0	0	0	0	15878
24	0	17	2	0	4	6	3	0	0	0	0	15773
25	0	78	4	0	4	3	4	2	1	1	0	15906

Predict ‘any y’ label in abalone dataset

Recommend y label

- blueWardsPlaced
- blueTotalGold
- blueAvgLevel
- blueTotalExperience
- blueTotalMinionsKilled
- blueGoldPerMin

DeepUser

Single Layer Perceptron

THANKS