FVE3011

Multivariable linear regression/ logistic regression classifier/ SoftMax classifier

담당교수: 최 학남 (xncui@inha.ac.kr)

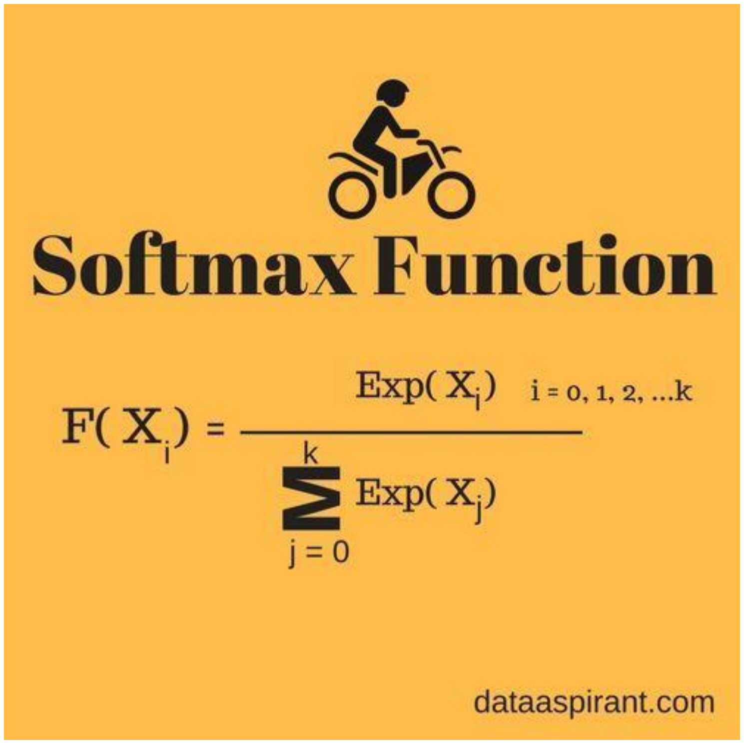
인하대학교

Contents

- Multivariable linear regression
- logistic regression classifier
- SoftMax classifier

Softmax classifier

- Softmax function

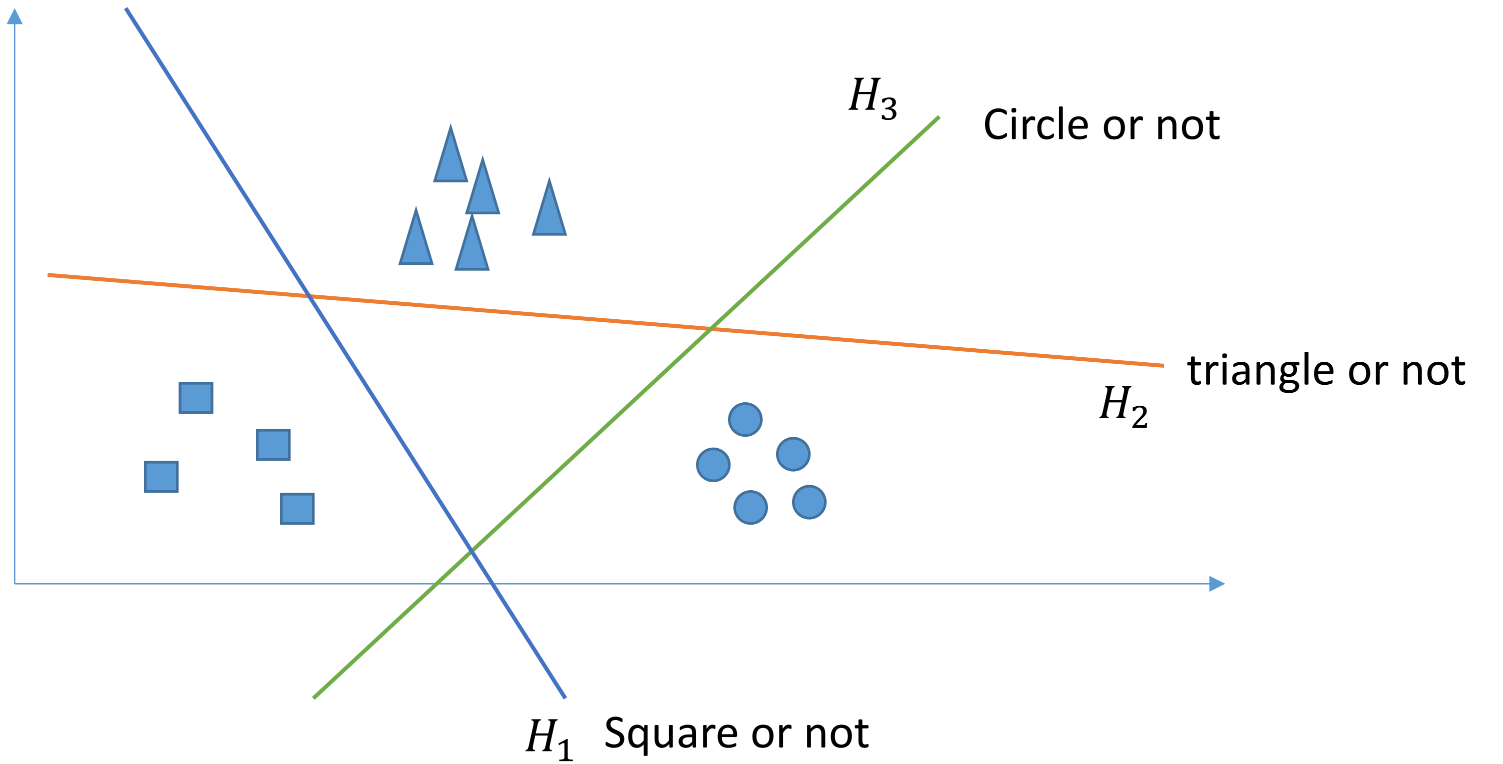


- The calculated probabilities will be in the range of 0 to 1.
- The sum of all the probabilities is equals to 1.
- Used in multiple classification logistic regression model.

```
def softmax(inputs):
    return np.exp(inputs) / float(sum(np.exp(inputs)))
```

```
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
softmax = torch.nn.Softmax()
linear = torch.nn.Linear(4, nb_classes, bias=True)
model = torch.nn.Sequential(linear, softmax)
```

Multinomial classification



$$H_1(X) = W_1 X$$
$$H_2(X) = W_2 X$$
$$H_3(X) = W_3 X$$

$$W_i = [w_{i1}, w_{i2}, w_{i3}] \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Multinomial classification (matrix form)

$$\boldsymbol{H}(X) = WX = \begin{bmatrix} w_{11}, w_{12}, w_{13} \\ w_{21}, w_{22}, w_{23} \\ w_{31}, w_{32}, w_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{bmatrix} = \begin{bmatrix} H_1(X) \\ H_2(X) \\ H_3(X) \end{bmatrix}$$

$$\begin{bmatrix} H_1(X) \\ H_2(X) \\ H_3(X) \end{bmatrix} = \begin{bmatrix} 4.0 \\ 2.0 \\ 0.9 \end{bmatrix} \Rightarrow \boxed{\text{Softmax}} \Rightarrow \mathrm{S}(\mathbf{H}) = \begin{bmatrix} 0.9 \\ 0.1 \\ 0.0 \end{bmatrix} \xrightarrow{\text{One-hot encoding}} \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

Cost function: cross entropy

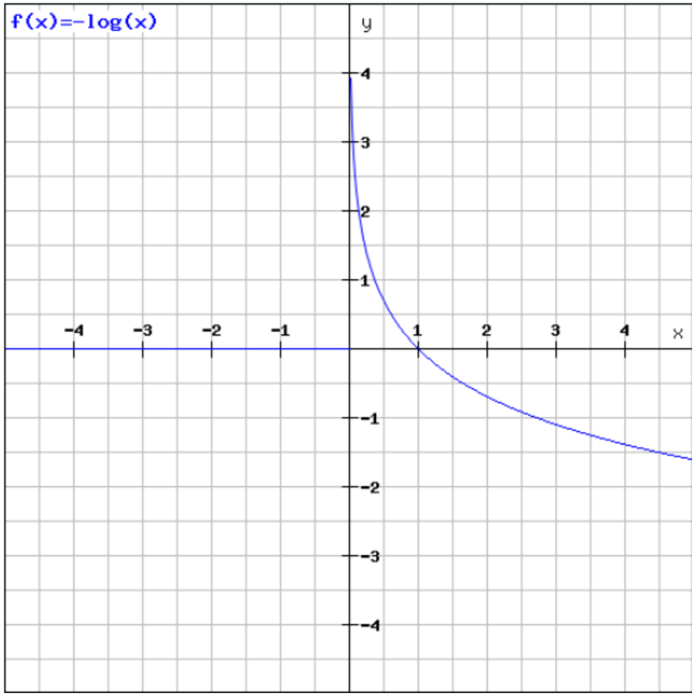
S(**H**) L

S(**H**)= $\begin{bmatrix} 0.9 \\ 0.1 \\ 0.0 \end{bmatrix}$

$$D(S,L) = -\sum_i L_i \log(S_i)$$

$\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$

$$D(S,L) = \sum_i L_i(-\log(S_i))$$



https://rechneronline.de/function-graphs/

Cost(loss) function

Loss function: $$\mathcal{L} = \frac{1}{N}\sum_{i} D(S(WX_i + b), L_i)$$

Difference Between Sigmoid Function and Softmax Function

	Softmax Function	Sigmoid Function
1	Used for multi-classification in logistic regression model.	Used for binary classification in logistic regression model.
2	The probabilities sum will be 1	The probabilities sum need not be 1.
3	Used in the different layers of neural networks.	Used as activation function while building neural networks.
4	The high value will have the higher probability than other values.	The high value will have the high probability but not the higher probability.

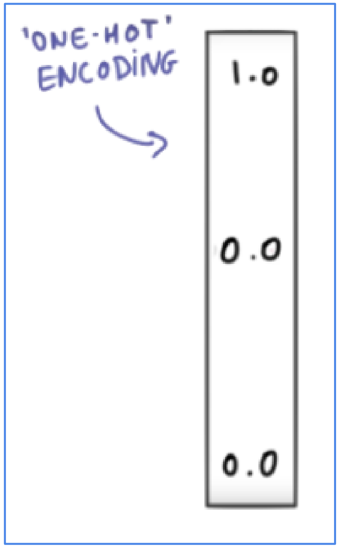
Cost function: cross entropy

$$\mathcal{L} = \frac{1}{N}\sum_{i} D(S(WX_i + b), L_i)$$

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

for step in range(2001):
    optimizer.zero_grad()
    hypothesis = model(X)
    # Cross entropy cost/loss
    cost = -Y * torch.log(hypothesis)
    cost = torch.sum(cost, 1).mean()
    cost.backward()
    optimizer.step()

    if step % 200 == 0:
        print(step, cost.data.numpy())
```

Training :One-hot encoding

```python
# Lab 6 Softmax Classifier
import torch
from torch.autograd import Variable

torch.manual_seed(777)  # for reproducibility

x_data = [[1, 2, 1, 1], [2, 1, 3, 2], [3, 1, 3, 4], [4, 1, 5, 5],
          [1, 7, 5, 5], [1, 2, 5, 6], [1, 6, 6, 6], [1, 7, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0],
          [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]]

X = Variable(torch.Tensor(x_data))
Y = Variable(torch.Tensor(y_data))
nb_classes = 3

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
softmax = torch.nn.Softmax()
linear = torch.nn.Linear(4, nb_classes, bias=True)
model = torch.nn.Sequential(linear, softmax)

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

for step in range(2001):
    optimizer.zero_grad()
    hypothesis = model(X)
    # Cross entropy cost/loss
    cost = -Y * torch.log(hypothesis)
    cost = torch.sum(cost, 1).mean()
    cost.backward()
    optimizer.step()

    if step % 200 == 0:
        print(step, cost.data.numpy())
```



Testing & one-hot encoding

```
# Testing & One-hot encoding
print('---------------')
a = model(Variable(torch.Tensor([[1, 11, 7, 9]])))
print(a.data.numpy(), torch.max(a, 1)[1].data.numpy())
```

```
[[ 1.38904958e-03  9.98601854e-01  9.06129117e-06]]
[1]
```

```
all = model(Variable(torch.Tensor([[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]])))
print(all.data.numpy(), torch.max(all, 1)[1].data.numpy())
```

```
[[ 1.38904958e-03   9.98601854e-01   9.06129117e-06]
 [ 9.31192040e-01   6.29020557e-02   5.90589503e-03]
 [ 1.27327668e-08   3.34112905e-04   9.99665856e-01]]

[1 0 2]
```

softmax_cross_entropy_with_logits

```
softmax = torch.nn.Softmax()

model = torch.nn.Linear(16, nb_classes, bias=True)
```

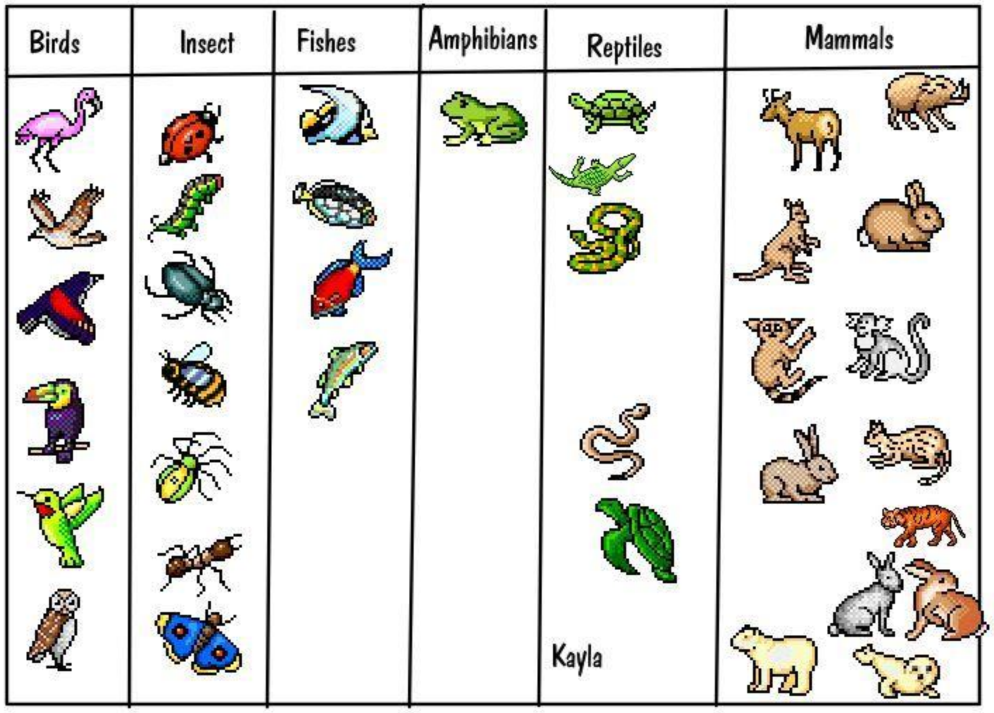
```
# Cross entropy cost/loss
criterion = torch.nn.CrossEntropyLoss()

# Softmax is internally computed.
```

Animal classification
with *softmax_cross_entropy_with_logits*

animalzoo.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
# https://archive.ics.uci.edu/ml/machine-learning-databases/zoo/zoo.data,,,,,
#   1. animal name:      (deleted),,,,,,,,,,,,,,,,
#   2. hair      Boolean",,,,,,,,,,,,,,,,
#   3. feathers      Boolean",,,,,,,,,,,,,,,,
#   4. eggs      Boolean",,,,,,,,,,,,,,,,
#   5. milk      Boolean",,,,,,,,,,,,,,,,
#   6. airborne      Boolean",,,,,,,,,,,,,,,,
#   7. aquatic      Boolean",,,,,,,,,,,,,,,,
#   8. predator      Boolean",,,,,,,,,,,,,,,,
#   9. toothed      Boolean",,,,,,,,,,,,,,,,
#  10. backbone      Boolean",,,,,,,,,,,,,,,,
#  11. breathes      Boolean",,,,,,,,,,,,,,,,
#  12. venomous      Boolean",,,,,,,,,,,,,,,,
#  13. fins      Boolean",,,,,,,,,,,,,,,,
#  14. legs      Numeric (set of values: {0",2,4,5,6,8}),,,,,,,,,,,
#  15. tail      Boolean",,,,,,,,,,,,,,,,
#  16. domestic      Boolean",,,,,,,,,,,,,,,,
#  17. catsize      Boolean",,,,,,,,,,,,,,,,
#  18. type      Numeric (integer values in range [0",6]),,,,,,,,,,,
1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,0
1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,0
0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,3
1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,0
1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,0
1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,0
1,0,0,1,0,0,0,1,1,1,0,0,4,1,1,1,0
```




```python
# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.txt', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

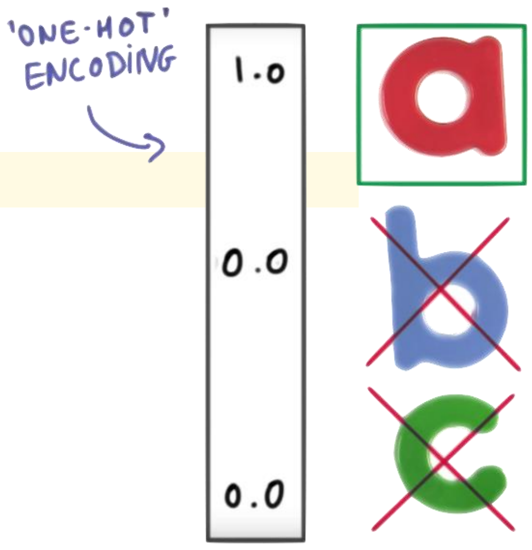
Tf.one_hot and reshape

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

```
# one hot encoding
Y_one_hot = torch.zeros(Y.size()[0], nb_classes)
Y_one_hot.scatter_(1, Y.long().data, 1)
Y_one_hot = Variable(Y_one_hot)
print("one_hot", Y_one_hot.data)
```

Animal classification: full code

```python
import torch
from torch.autograd import Variable
import numpy as np
torch.manual_seed(777)  # for reproducibility
# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
print(x_data.shape, y_data.shape)
nb_classes = 7  # 0 ~
X = Variable(torch.from_numpy(x_data))
Y = Variable(torch.from_numpy(y_data))
# one hot encoding
Y_one_hot = torch.zeros(Y.size()[0], nb_classes)
Y_one_hot.scatter_(1, Y.long().data, 1)
Y_one_hot = Variable(Y_one_hot)
print("one_hot", Y_one_hot.data)
softmax = torch.nn.Softmax()
model = torch.nn.Linear(16, nb_classes, bias=True)
```



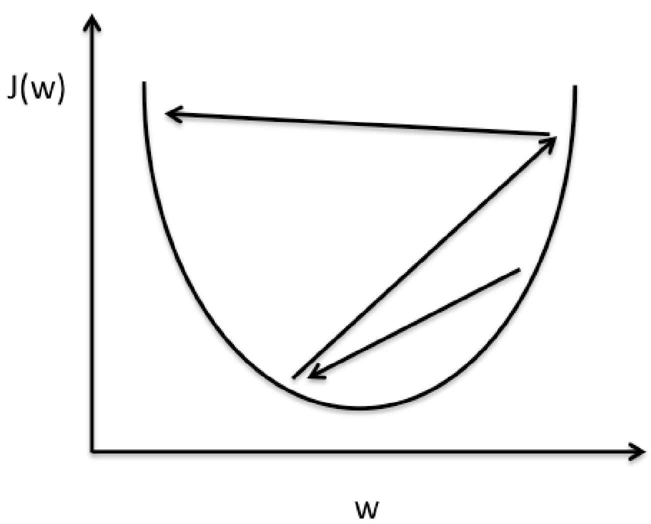
Animal classification: full code

```
model = torch.nn.Linear(16, nb_classes, bias=True)
# Cross entropy cost/loss
criterion = torch.nn.CrossEntropyLoss()  # Softmax is internally computed.
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
for step in range(201):
    optimizer.zero_grad()
    hypothesis = model(X)
    # Label has to be 1D LongTensor
    cost = criterion(hypothesis, Y.long().view(-1))
    cost.backward()
    optimizer.step()
    ...
    prediction = torch.max(torch.softmax(hypothesis,dim=1), 1)[1].float()
    ...
    correct_prediction = (prediction.data == Y.data.reshape(101))
    accuracy = correct_prediction.float().mean()
    ...

    if step % 100 == 0:
        print("Step: {:5}\tLoss: {:.3f}\tAcc: {:.2%}".format(step, cost.data.numpy(), accuracy))

# Let's see if we can predict
pred = torch.max(torch.softmax(hypothesis,dim=1), 1)[1].float()

for p, y in zip(pred, Y):
    print("[{}] Prediction: {} True Y: {}".format(bool(p.data == y.data), p.data.int(), y.data.numpy()))
```

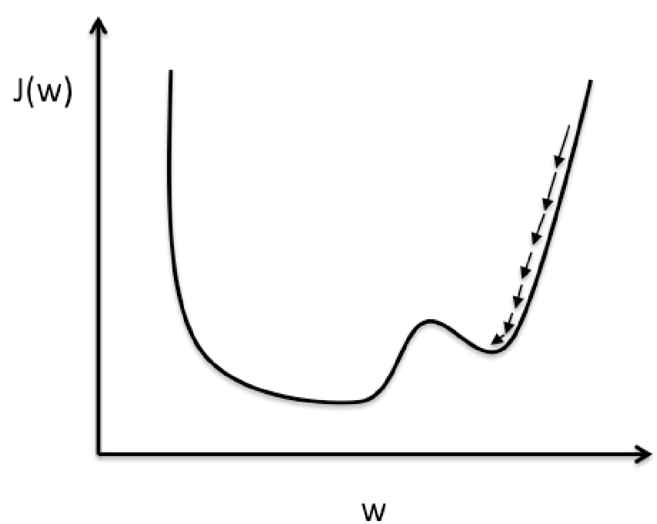
```
Step:  1100 Loss: 0.101 Acc: 99.01%
Step:  1200 Loss: 0.092 Acc: 100.00%
Step:  1300 Loss: 0.084 Acc: 100.00%
...
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
```

How to set the learning rate ?

- Try several learning rates
  - Observe the cost function
  - Check it goes down in a reasonable rate



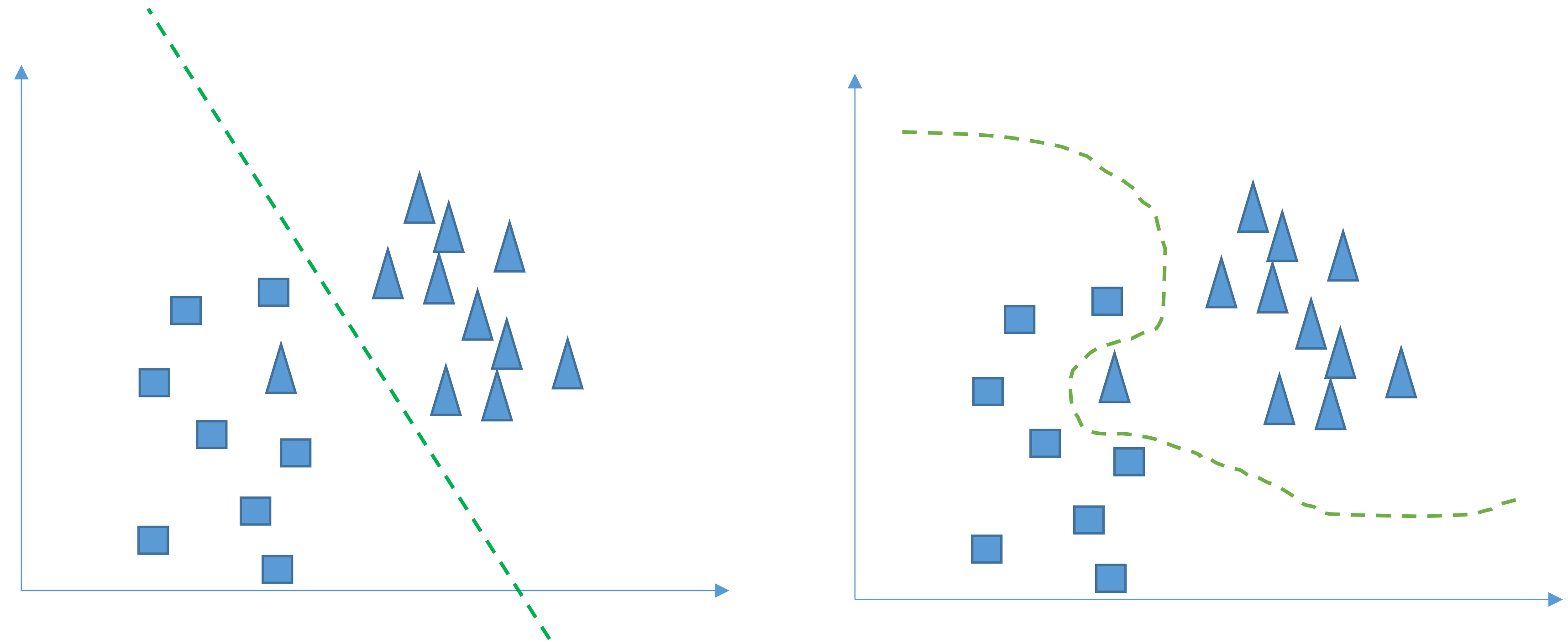
Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

Overfitting

- Our model is very good with training data set(with memorization)
- Not good at test dataset or in real use

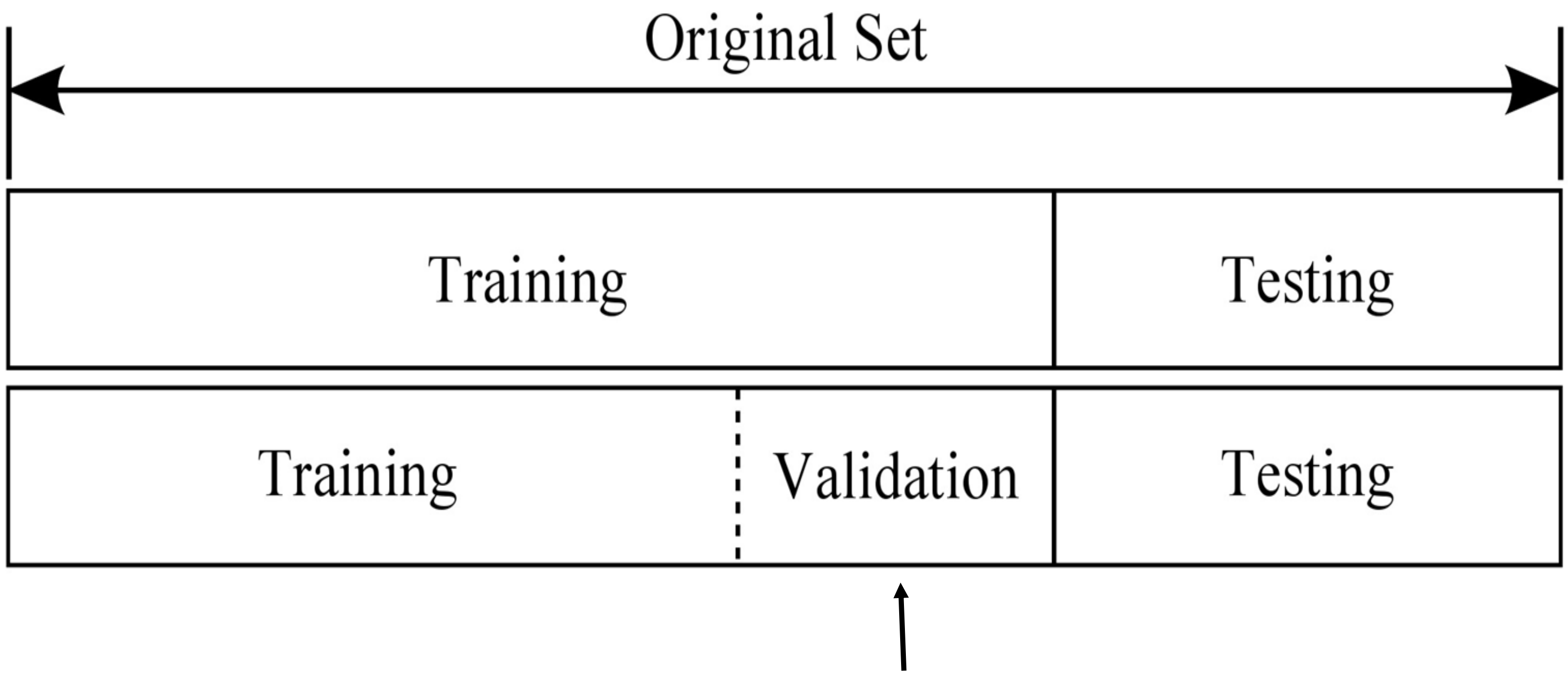


Solution for overfitting

- More training data!
- Reduce the number of features
  - Remove the similar feature
- Regularization

$$\mathcal{L} = \frac{1}{N}\sum_i D(S(WX_i + b), L_i) + \lambda \sum W^2$$

Training and testing sets



Validation set
use to tune hyperparameters

Hyperparameters: learning rate , lambda etc.

Training epoch/batch

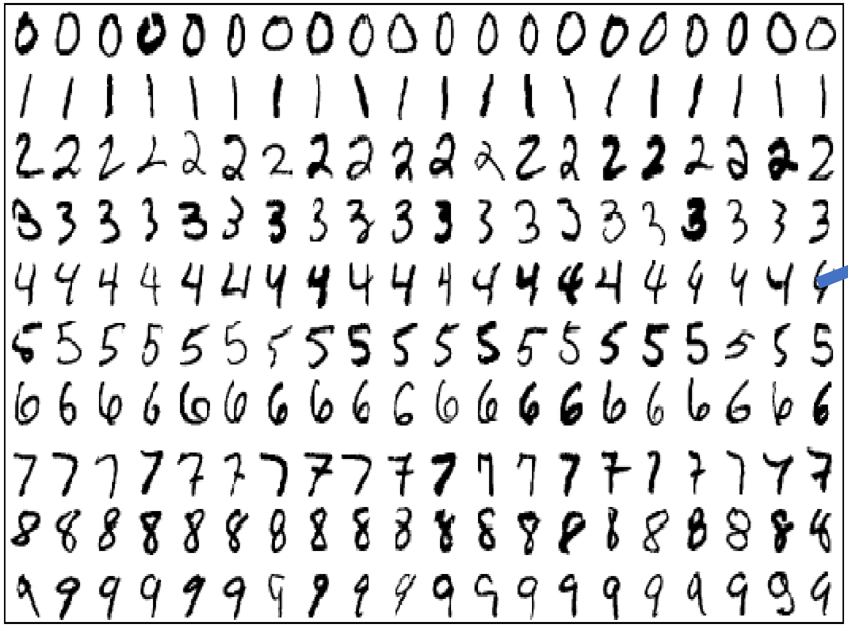
- In the neural network terminology:
  - **one epoch** = one forward pass and one backward pass of all the training examples
  - **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
  - **number of iterations** = number of passes, each pass using [batch size] number of examples. To be clear,
    - one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).
- Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

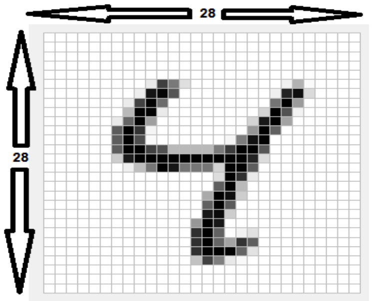
Training epoch/batch

```
# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch:%4d] cost =%.9f"%(epoch + 1, avg_cost))

print('Learning Finished!')

# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)
```

- Classify the MNIST data using SoftMax classifier.

  - MNIST contains 60,000 training images and 10,000 testing images





- Data handling using pytorch

```
import torch
from torch.autograd import Variable
import torchvision
import random
torch.manual_seed(777)  # reproducibility
# parameters
learning_rate = 0.001
training_epochs = 3
batch_size = 64
# MNIST dataset
mnist_train = torchvision.datasets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=torchvision.transforms.ToTensor(),
                          download=True)

mnist_test = torchvision.datasets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=torchvision.transforms.ToTensor(),
                         download=True)
# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)
```

Practice : reference code

```
# MNIST and softmax
import torch
from torch.autograd import Variable
import torchvision
import random
torch.manual_seed(777)  # reproducibility
# parameters
learning_rate = 0.001
training_epochs = 3
batch_size = 64
# MNIST dataset
mnist_train = torchvision.datasets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=torchvision.transforms.ToTensor(),
                          download=True)

mnist_test = torchvision.datasets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=torchvision.transforms.ToTensor(),
                         download=True)
# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)

# model

# define cost/loss & optimizer
```

```python
# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch:%4d] cost =%.9f"%(epoch + 1, avg_cost))

print('Learning Finished!')

# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)

# Get one and predict
r = random.randint(0, len(mnist_test) - 1)
X_single_data = Variable(mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float())
Y_single_data = Variable(mnist_test.test_labels[r:r + 1])

print("Label: ", Y_single_data.data)
single_prediction = model(X_single_data)
print("Prediction: ", torch.max(single_prediction.data, 1)[1])
```