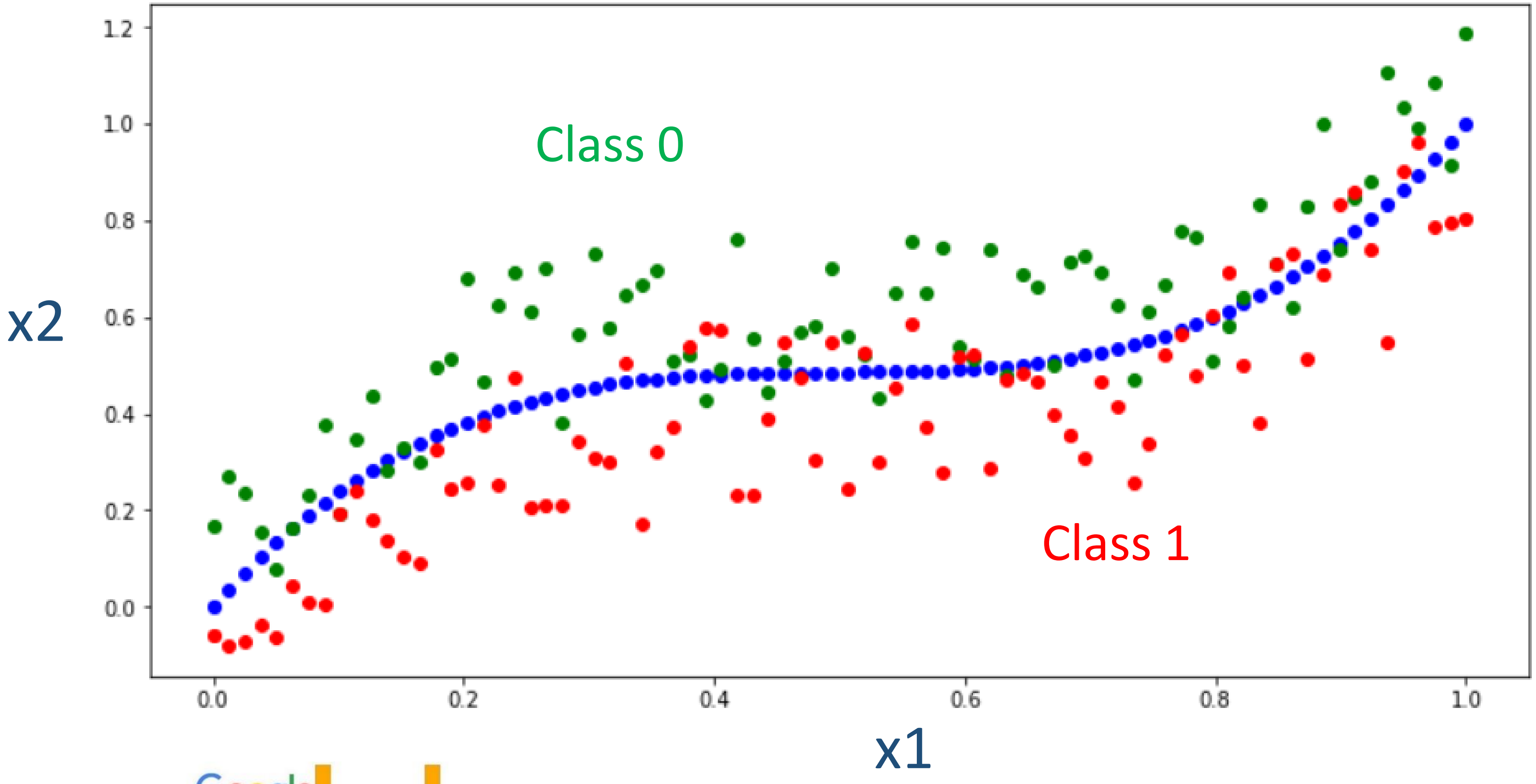
Run "4.1. Classification with MSE loss.ipynb"

Learn a function: $y = f(X)$

$X = (x1, x2)$

$Y = 0, 1$



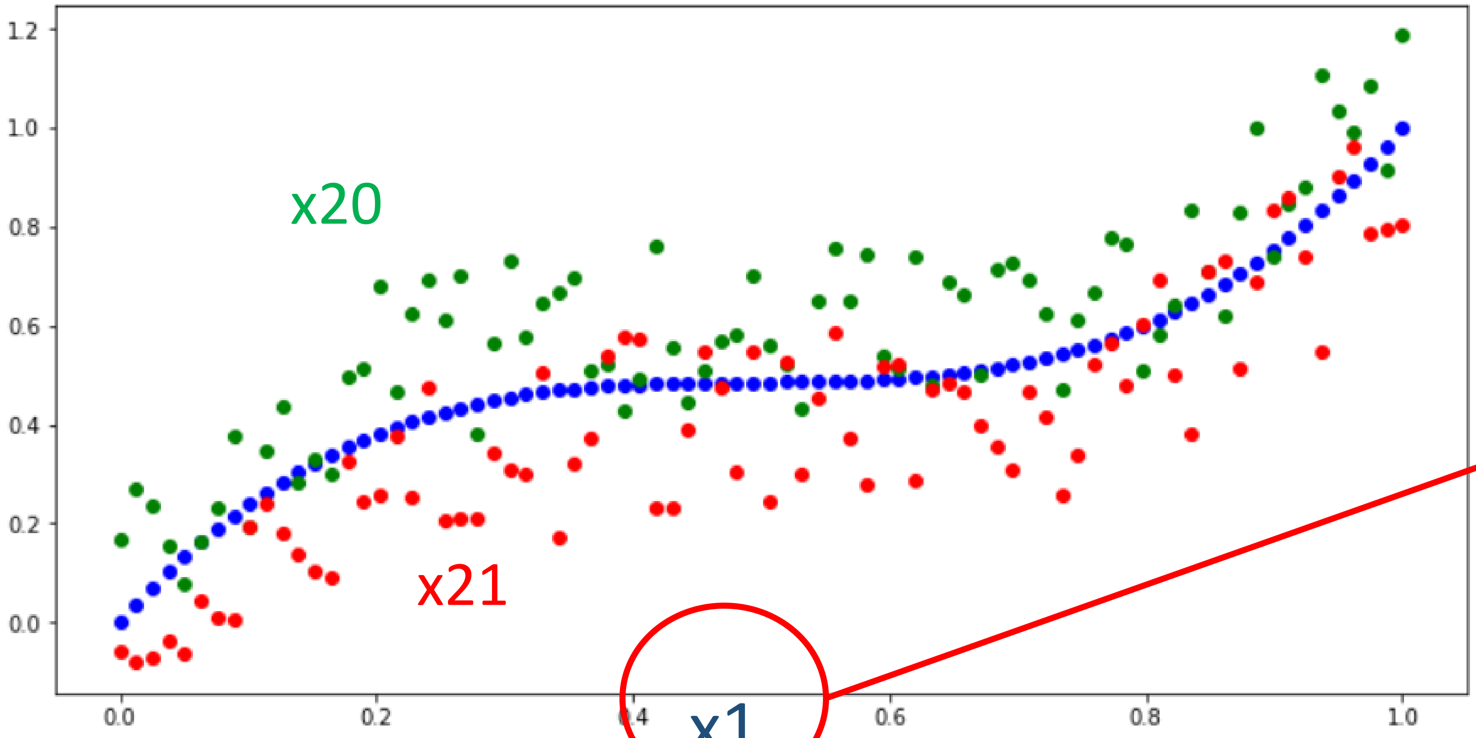
Generate list x1, x20, x21 for two classes

In [7]:

```
lstX20 = []
lstX21 = []
for i in range(len(lstX1)):
  lstX20.append(lstY1[i] + random.uniform(-0.1, 0.3))
  lstX21.append(lstY1[i] - random.uniform(-0.1, 0.3))
```

(2)

x2



x1

(1)

```
x = -10
while(x<10):
   y=3*x*x*x + 2*x*x + 5*x
   lstY1.append(y)
   lstX1.append(x)
   x = x + 0.25
print(len(lstX1), len(lstY1))

#normalized to [0,1]
lstX1= [(float(i)-min(lstX1))/(
lstY1= [(float(i)-min(lstY1))/(
```

Combine list x1, x20, x21 to generate X and Y

X = (x1, x2)
Y = 0, 1

In [9]:
```
lstX=[]
lstY=[]
for i in range(len(lstX1)):
  lstX.append([lstX1[i],lstX20[i]])
  lstY.append([0])
  lstX.append([lstX1[i],lstX21[i]])
  lstY.append([1])
numpyX = np.array(lstX)
numpyY = np.array(lstY)
print(numpyX.shape, numpyY.shape)
```

```
(160, 2) (160, 1)
```

Using MSE and gradient decent to learn

In [5]:
```
loss_func = torch.nn.MSELoss()
optimizer = torch.optim.Adam(MyNet.parameters(), lr=0.005)
```

Train with mini-batches

In [11]:
```
import torch.utils.data as Data
torch_dataset = Data.TensorDataset(tensorX, tensorY_hat)
```

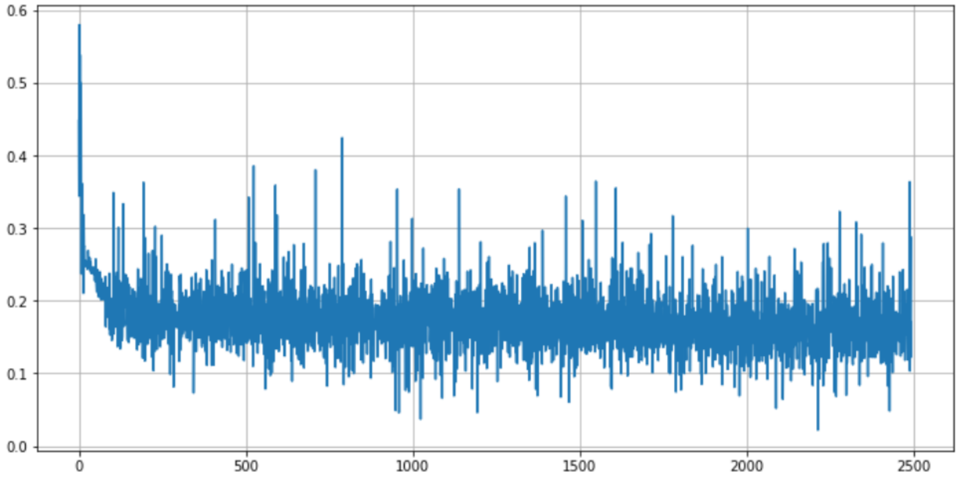
In [12]:
```
loader = Data.DataLoader(
    dataset=torch_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=0,    # subprocesses for loading data
)
```

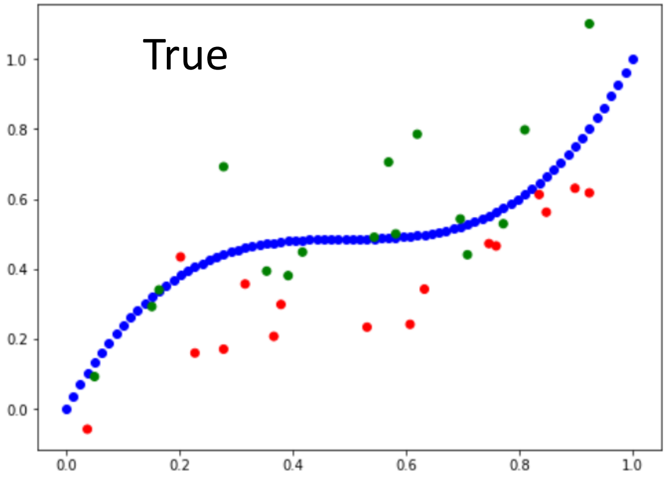
In [13]:
```
# initialize NN weights
for name, param in MyNet.named_parameters():
  if(param.requires_grad):
    torch.nn.init.normal_(param, mean=0.0, std=0.02)

lossLst = []
for epoch in range(1, 500):
  for (batchX, batchY_hat) in loader:
    tensorY = MyNet(batchX)
    loss = loss_func(batchY_hat, tensorY)
    lossLst.append(float(loss))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

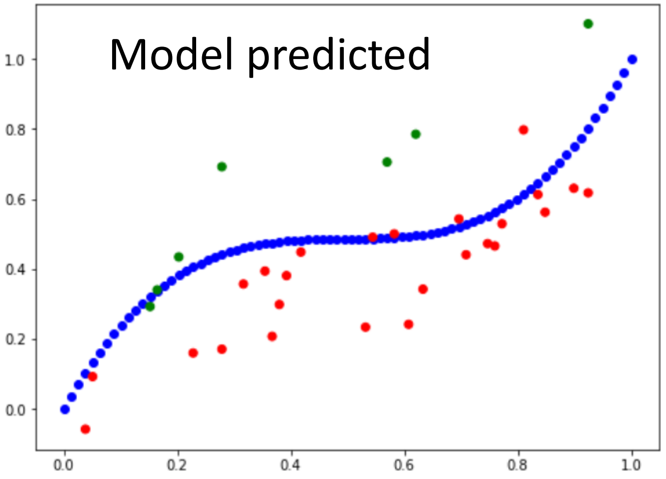
Classification with threshold = 0.5

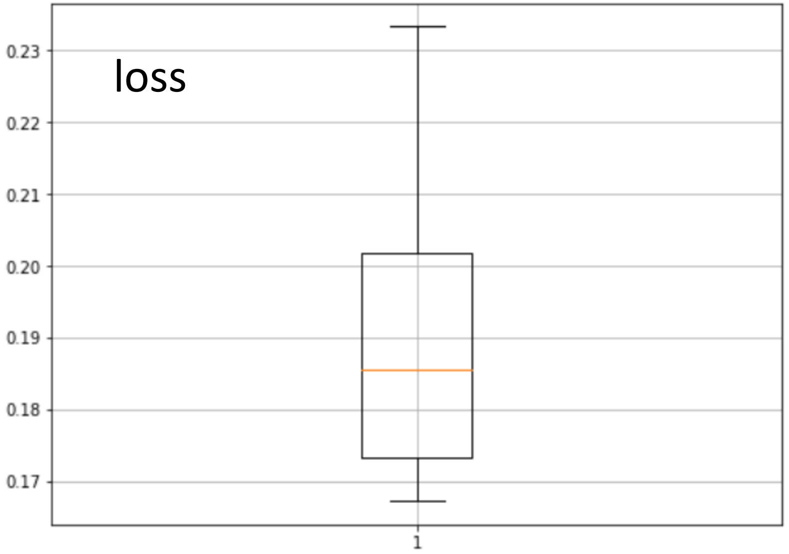
```
correctNo = 0
for i in range(Y.size):
  if(Y[i][0]<=0.5):
    lstColor.append("green")
    if(testY_hat[i][0]==0):
      correctNo = correctNo + 1
  else:
    lstColor.append("red")
    if(testY_hat[i][0]==1):
      correctNo = correctNo + 1
accuracy = correctNo/Y.size
```

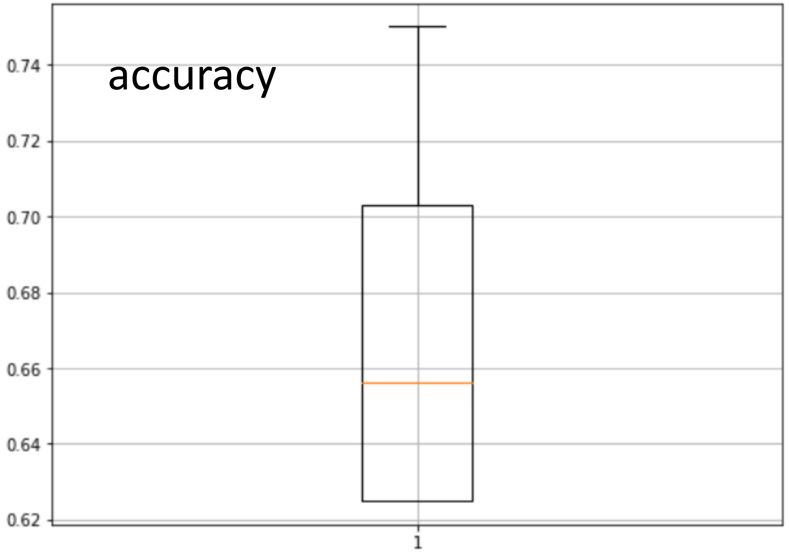
Model performance visualization











Other loss function for classification?

Probabilistic Generative Model

We can use Bayesian's rule to derive the probability of x belonging to a class C_i

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

Generative Model $P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$

To find the probability of observing x in class C_i, i.e., $p(x|C_1)$, we assume data points x^n are sampled from a Gaussian distribution, then we can use maximum likelihood to find the best Gaussian distribution behind them.

Reference: 李弘毅 ML Lecture 4 https://youtu.be/fZAZUYEeIMg

Probabilistic Generative Model

$$f_{\mu^1,\Sigma^1}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^1|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu^1)^T(\Sigma^1)^{-1}(x-\mu^1)\right\}$$

$$\mu^1 = \begin{bmatrix} 75.0 \\ 71.3 \end{bmatrix} \quad \Sigma^1 = \begin{bmatrix} 874 & 327 \\ 327 & 929 \end{bmatrix}$$

P(C1)
= 79 / (79 + 61) =0.56

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$f_{\mu^2,\Sigma^2}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^2|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu^2)^T(\Sigma^2)^{-1}(x-\mu^2)\right\}$$

$$\mu^2 = \begin{bmatrix} 55.6 \\ 59.8 \end{bmatrix} \quad \Sigma^2 = \begin{bmatrix} 847 & 422 \\ 422 & 685 \end{bmatrix}$$

P(C2)
= 61 / (79 + 61)
=0.44

If $P(C_1|x) > 0.5$ ➡ x belongs to class 1 (Water)

Reference: 李弘毅 ML Lecture 4 https://youtu.be/fZAZUYEeIMg

Posterior probability

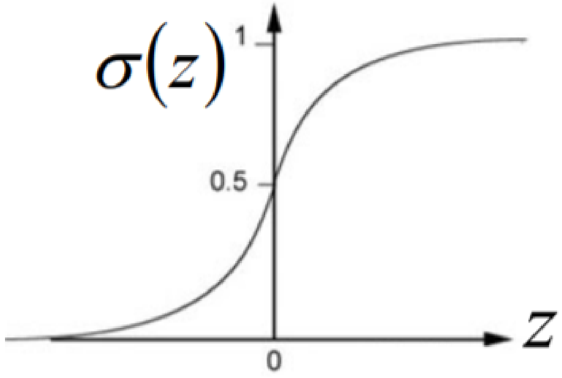
We can convert the posterior probability $P(C_i|x)$ as sigmoid function of z

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$= \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} = \frac{1}{1 + exp(-z)} = \sigma(z)$$

Sigmoid function

$$z = ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$



Reference: 李弘毅 ML Lecture 4 https://youtu.be/fZAZUYEeIMg

Posterior probability

If the covariance matrices of the two classes are the same, then the posterior probability could be represented as linear combination of x

$$P(C_1|x) = \sigma(z)$$

$$\boxed{\begin{aligned} z = \cancel{ln\frac{|\Sigma^2|^{1/2}}{|\Sigma^1|^{1/2}}} \cancel{-\frac{1}{2}x^T(\Sigma^1)^{-1}x} + (\mu^1)^T(\Sigma^1)^{-1}x - \frac{1}{2}(\mu^1)^T(\Sigma^1)^{-1}\mu^1 \\ \cancel{+\frac{1}{2}x^T(\Sigma^2)^{-1}x} - (\mu^2)^T(\Sigma^2)^{-1}x + \frac{1}{2}(\mu^2)^T(\Sigma^2)^{-1}\mu^2 + ln\frac{N_1}{N_2} \end{aligned}}$$

$$\Sigma_1 = \Sigma_2 = \Sigma$$

$$z = \underbrace{(\mu^1 - \mu^2)^T\Sigma^{-1}}_{\boldsymbol{w^T}}x \underbrace{- \frac{1}{2}(\mu^1)^T\Sigma^{-1}\mu^1 + \frac{1}{2}(\mu^2)^T\Sigma^{-1}\mu^2 + ln\frac{N_1}{N_2}}_{\text{b}}$$

$$P(C_1|x) = \sigma(w \cdot x + b)$$

How about directly find ***w*** and b?

In generative model, we estimate $N_1, N_2, \mu^1, \mu^2, \Sigma$

Then we have ***w*** and b

Logistic Regression

We want to find $P_{w,b}(C_1|x)$

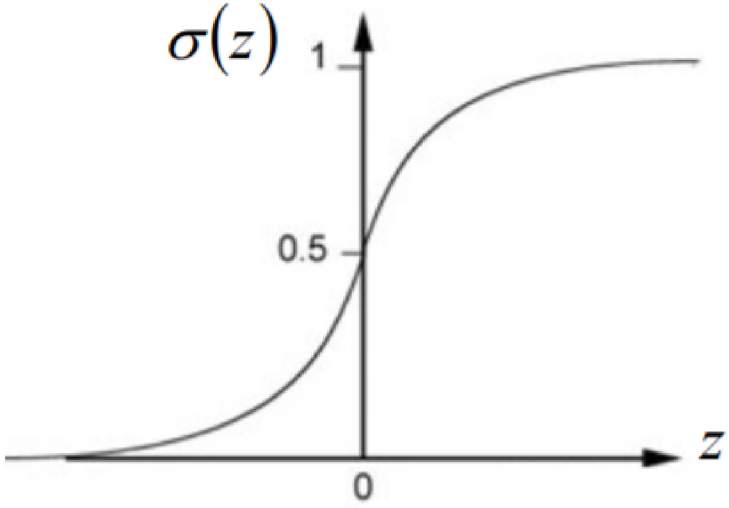
If $P_{w,b}(C_1|x) \geq 0.5$, output C_1

Otherwise, output C_2

$$P_{w,b}(C_1|x) = \sigma(z)$$

$$z = w \cdot x + b$$

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Logistic Regression vs Regression

Logistic Regression	Linear Regression
$f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$	$f_{w,b}(x) = \sum_i w_i x_i + b$
Output: between 0 and 1	Output: any value

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Loss function

Training Data	x^1	x^2	x^3	……	x^N
	C_1	C_1	C_2		C_1

Assuming the training data is generated from $P_{w,b}(C_1|x) = \sigma(w \cdot x + b)$, what is the probability of generating the data?

max $$L(w,b) = f_{w,b}(x^1) f_{w,b}(x^2) \left(1 - f_{w,b}(x^3)\right) \cdots f_{w,b}(x^N)$$

min $$-lnL(w,b) = -lnf_{w,b}(x^1) - lnf_{w,b}(x^2) - ln\left(1 - f_{w,b}(x^3)\right) \cdots$$

$\hat{y}^n$: 1 for class 1, 0 for class 2

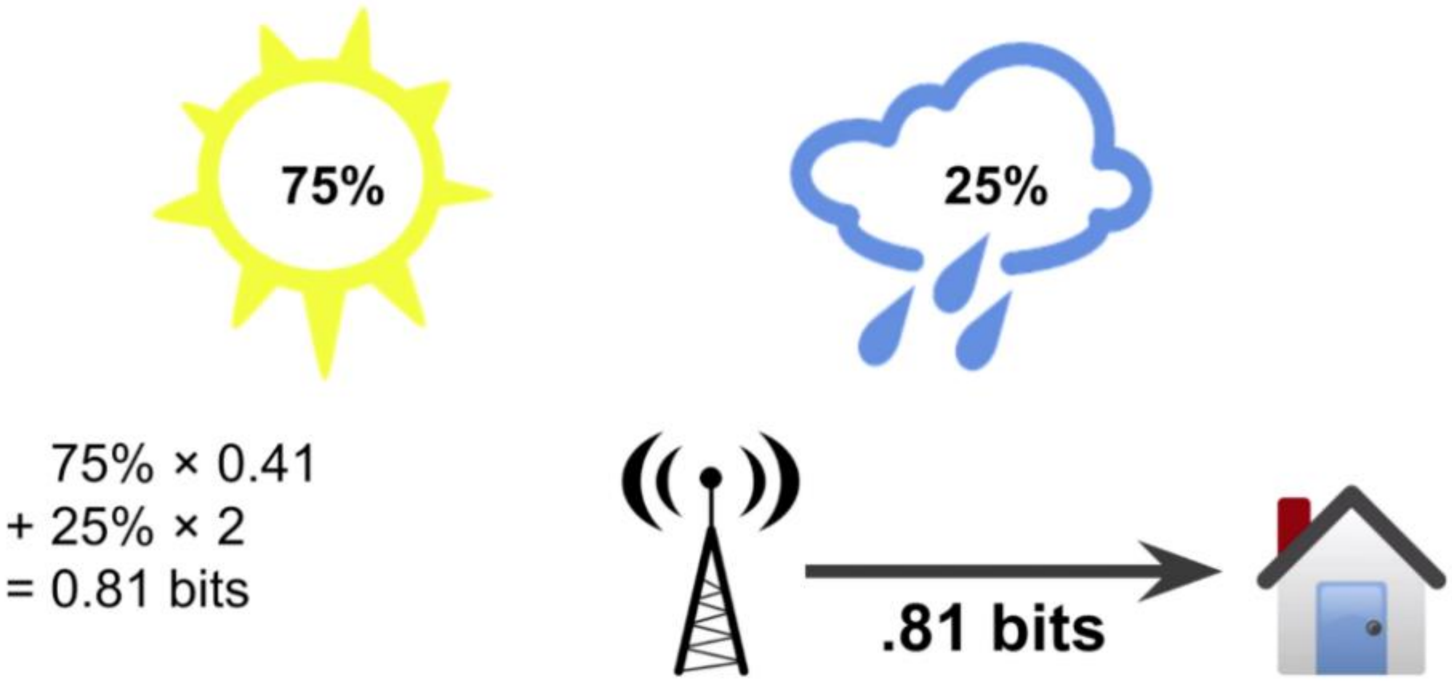
$$= \sum_n -\underbrace{\left[\hat{y}^n lnf_{w,b}(x^n) + (1 - \hat{y}^n) ln\left(1 - f_{w,b}(x^n)\right)\right]}_{\text{Cross entropy between two Bernoulli distribution}}$$

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Entropy

More information → more uncertain → larger entropy

$$Entropy = -\sum_i p_i log_2(p_i)$$



https://r23456999.medium.com/%E4%BD%95%E8%AC%82-cross-entropy-%E4%BA%A4%E5%8F%89%E7%86%B5-b6d4cef9189d

Cross entropy

Measures the differences between the true probability p_i and the predicted probability q_i

$$H(p,q) = -\sum_i p_i log_2(q_i)$$

動物	實際機率分佈	預測機率分佈	Entropy
Cat	0%	2%	0% * -log(2%) = 0
Dog	0%	30%	0% * -log(30%) = 0
Fox	0%	45%	0% * -log(45%) = 0
Cow	0%	0%	0% * -log(0%) = 0
Red Panda	100%	25%	100% * -log(25%) = 1.386
Bear	0%	5%	0% * -log(5%) = 0
Dolphin	0%	0%	0% * -log(0%) = 0
總計: cross-entropy = 1.386			

https://r23456999.medium.com/%E4%BD%95%E8%AC%82-cross-entropy-%E4%BA%A4%E5%8F%89%E7%86%B5-b6d4cef9189d

Loss function

Training data: $(x^n, \hat{y}^n)$

$\hat{y}^n$: 1 for class 1, 0 for class 2

$$L(f) = \sum_n C(f(x^n), \hat{y}^n)$$

Training data: $(x^n, \hat{y}^n)$

$\hat{y}^n$: a real number

$$L(f) = \frac{1}{2}\sum_n (f(x^n) - \hat{y}^n)^2$$

Cross entropy:

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n ln f(x^n) + (1 - \hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Run " 4.2. Classification with CE loss"

```
In [5]: lstX=[]
        lstY=[]
        for i in range(len(lstX1)):
          lstX.append([lstX1[i],lstX20[i]])
          lstY.append(0)
          lstX.append([lstX1[i],lstX21[i]])
          lstY.append(1)
        numpyX = np.array(lstX)
        numpyY = np.array(lstY)
        print(numpyX.shape, numpyY.shape)

(160, 2) (160,)
```

Y is a vector

4.2. Classification with CE loss

```
In [9]: lstX=[]
        lstY=[]
        for i in range(len(lstX1)):
          lstX.append([lstX1[i],lstX20[i]])
          lstY.append([0])
          lstX.append([lstX1[i],lstX21[i]])
          lstY.append([1])
        numpyX = np.array(lstX)
        numpyY = np.array(lstY)
        print(numpyX.shape, numpyY.shape)

(160, 2) (160, 1)
```

Y is a matrix

4.1. Classification with MSE loss

Neural network and loss function

```
In [7]: MyNet = nn.Sequential(
            nn.Linear(2, 50),
            nn.ReLU(),
            nn.Linear(50, 100),
            nn.ReLU(),
            nn.Linear(100, 50),
            nn.ReLU(),
            nn.Linear(50, 2),
        )
        MyNet.to(device)
        loss_func = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(MyNet.parameters(), lr=0.005)
```

2 classes

Cross entropy loss

In [12]:
```
for (batchX, batchY_hat) in loader:
    break
print(batchX.shape, batchY_hat)
```

```
torch.Size([5, 2]) tensor([0, 0, 0, 1, 1], device='cuda:0')
```

Send batchX to NN

In [13]:
```
tensorY = MyNet(batchX)
print(tensorY.shape,"\n", tensorY)
```

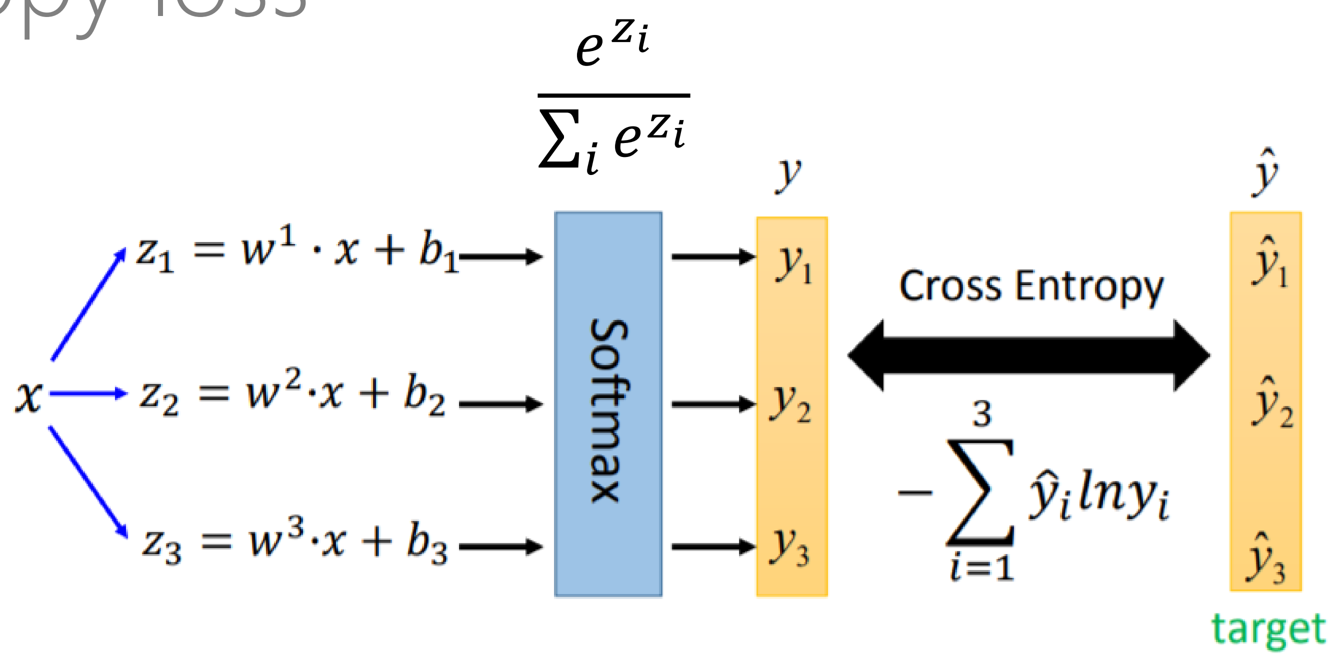
```
torch.Size([5, 2])
 tensor([[-0.0180,  0.0855],
        [-0.0244,  0.0741],
        [-0.0187,  0.0850],
        [-0.0258,  0.0687],
        [-0.0267,  0.0617]], device='cuda:0', grad_fn=<AddmmBackward>)
```

Calculate cross entropy between y and y-hat

In [14]:
```
loss = loss_func(tensorY, batchY_hat)
print(tensorY.shape, batchY_hat.shape, loss)
```

```
torch.Size([5, 2]) torch.Size([5]) tensor(0.7066, device='cuda:0', grad_
```

Cross entropy loss



A	B	C	D	E	F	G	H
z1	z2	y-hat	EXP(A)	EXP(B)	D+E	(D or E)/ (D+E)	-1*LN(G)
-0.018	0.0855	0	0.982	1.089	2.071	0.474	0.74624
-0.0244	0.0741	0	0.976	1.077	2.053	0.475	0.74361
-0.0187	0.085	0	0.981	1.089	2.070	0.474	0.74634
-0.0258	0.0687	1	0.975	1.071	2.046	0.524	0.64701
-0.0267	0.0617	1	0.974	1.064	2.037	0.522	0.64992
							0.706624521

Soft max and torch.max

Calculate accuracy

```
In [15]: print(tensorY.shape,"\n", tensorY)

torch.Size([5, 2])
 tensor([[-0.0180,  0.0855],
        [-0.0244,  0.0741],
        [-0.0187,  0.0850],
        [-0.0258,  0.0687],
        [-0.0267,  0.0617]], device='cuda:0', grad_fn=<
```

```
In [16]: # apply softmax
         tensorY = torch.softmax(tensorY, 1)
         print(tensorY.shape,"\n", tensorY)

torch.Size([5, 2])
 tensor([[0.4742, 0.5258],
        [0.4754, 0.5246],
        [0.4741, 0.5259],
        [0.4764, 0.5236],
        [0.4779, 0.5221]], device='cuda:0', grad_fn=<So
```

```
In [17]: MaxOfEachRow = torch.max(tensorY, 1)
         print(MaxOfEachRow)

torch.return_types.max(
values=tensor([0.5258, 0.5246, 0.5259, 0.5236, 0.5221],
       grad_fn=<MaxBackward0>),
indices=tensor([1, 1, 1, 1, 1], device='cuda:0'))
```

Torch.max

```
In [17]: MaxOfEachRow = torch.max(tensorY, 1)
         print(MaxOfEachRow)
```

```
torch.return_types.max(
values=tensor([0.5258, 0.5246, 0.5259, 0.5236, 0.5221], device='c
       grad_fn=<MaxBackward0>),
indices=tensor([1, 1, 1, 1, 1], device='cuda:0'))
```

```
In [18]: MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
         print(MaxIdxOfEachRow)
```

```
tensor([1, 1, 1, 1, 1], device='cuda:0')
```

```
In [19]: correct = 0
         MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
         for i in range(batchY_hat.shape[0]):
           print(int(MaxIdxOfEachRow[i]), int(batchY_hat[i]), end="==>")
           if (int(MaxIdxOfEachRow[i]) == int(batchY_hat[i])):
             print("correct")
             correct += 1
           else:
             print("wrong")
         print(correct)
         accuracy = correct/batchY_hat.shape[0]
         print("%.2f" % accuracy)
```

```
1 0==>wrong
1 0==>wrong
1 0==>wrong
1 1==>correct
1 1==>correct
2
0.40
```

Mini-batch training

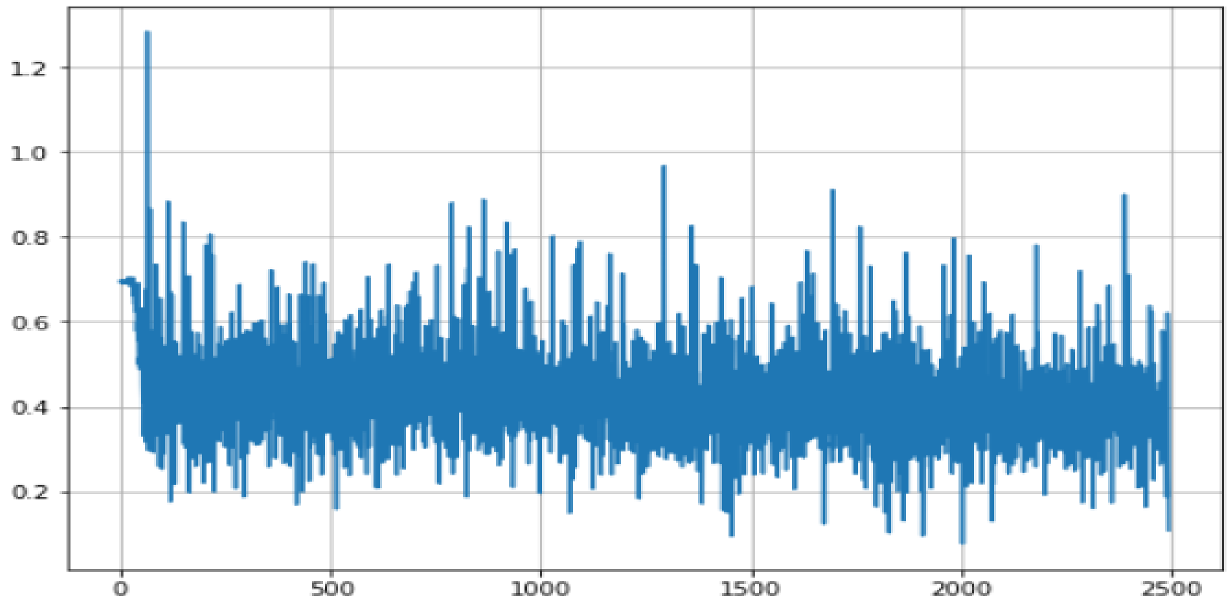
```
for epoch in range(1, 500):
  for (batchX, batchY_hat) in loader:
    tensorY = MyNet(batchX)
    tensorY = torch.softmax(tensorY, 1)
    loss = loss_func(tensorY, batchY_hat)
    lossLst.append(float(loss))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    correct = 0
    MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
    for i in range(batchY_hat.shape[0]):
      if (int(MaxIdxOfEachRow[i]) == int(batchY_hat[i])):
        correct += 1
    accuracy = correct/batchY_hat.shape[0]
    accuracyLst.append(accuracy)
```

4.2. Classification with CE loss

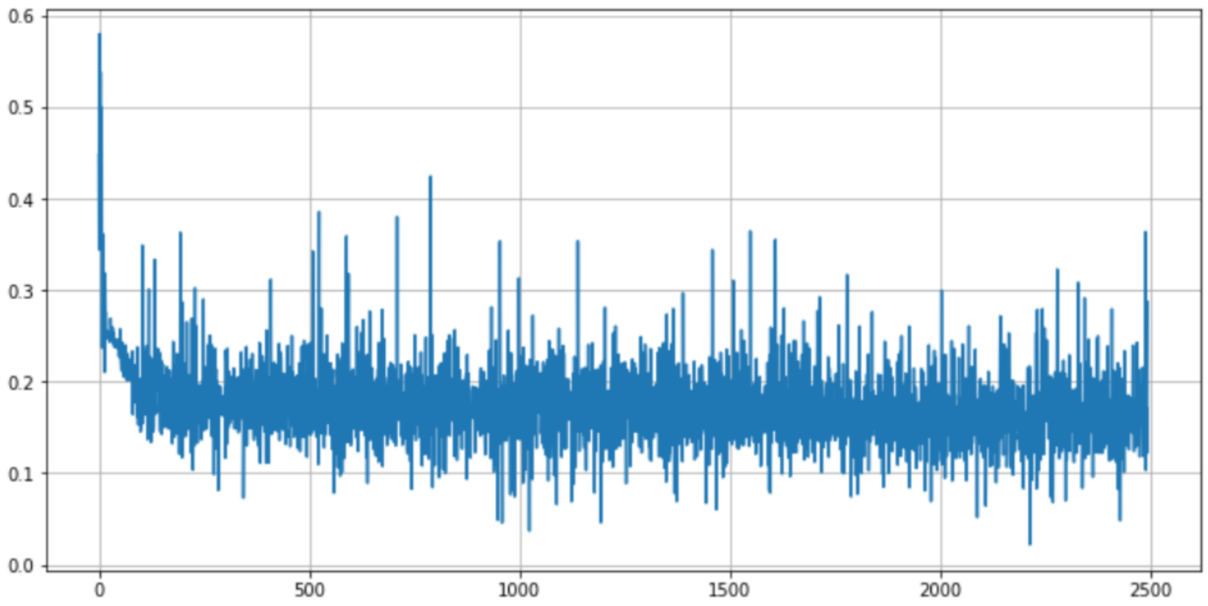
```
for epoch in range(1, 500):
  for (batchX, batchY_hat) in loader:
    tensorY = MyNet(batchX)
    loss = loss_func(batchY_hat, tensorY)
    lossLst.append(float(loss))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

4.1. Classification with MSE loss

Loss plot



4.2. Classification with CE loss



4.1. Classification with MSE loss

Model performance on test data

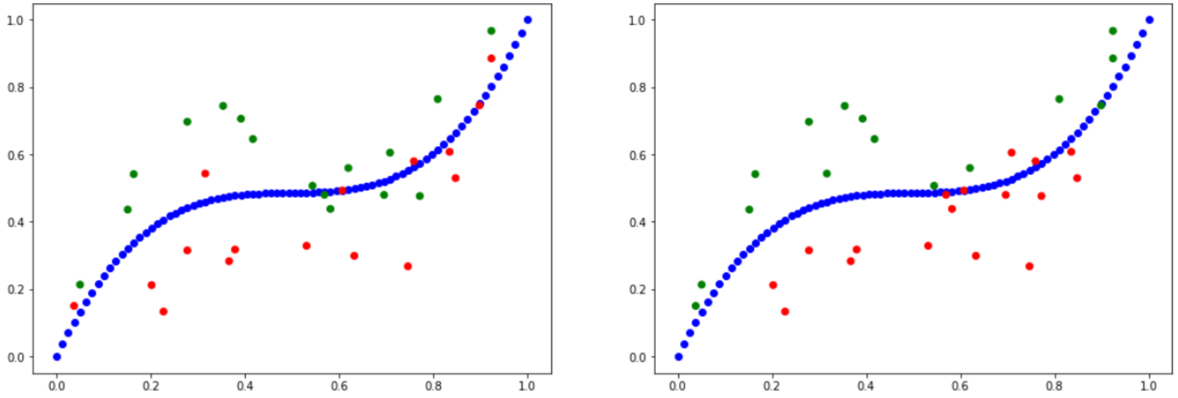
```python
# show model predicted classification
lstColor = []
correctNo = 0
MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
for i in range(tensorY.shape[0]):
  if (int(MaxIdxOfEachRow[i]) == 0):
    lstColor.append("green")
    if(int(testY_hat[i])==0):
      correctNo += 1
  else:
    lstColor.append("red")
    if(testY_hat[i]==1):
      correctNo = correctNo + 1
print(correctNo)
accuracy = correctNo/tensorY.shape[0]
```

4.2. Classification with CE loss

```python
# show model predicted classification
lstColor = []
correctNo = 0
for i in range(Y.size):
  if(Y[i][0]<=0.5):
    lstColor.append("green")
    if(testY_hat[i][0]==0):
      correctNo = correctNo + 1
  else:
    lstColor.append("red")
    if(testY_hat[i][0]==1):
      correctNo = correctNo + 1
accuracy = correctNo/Y.size
```

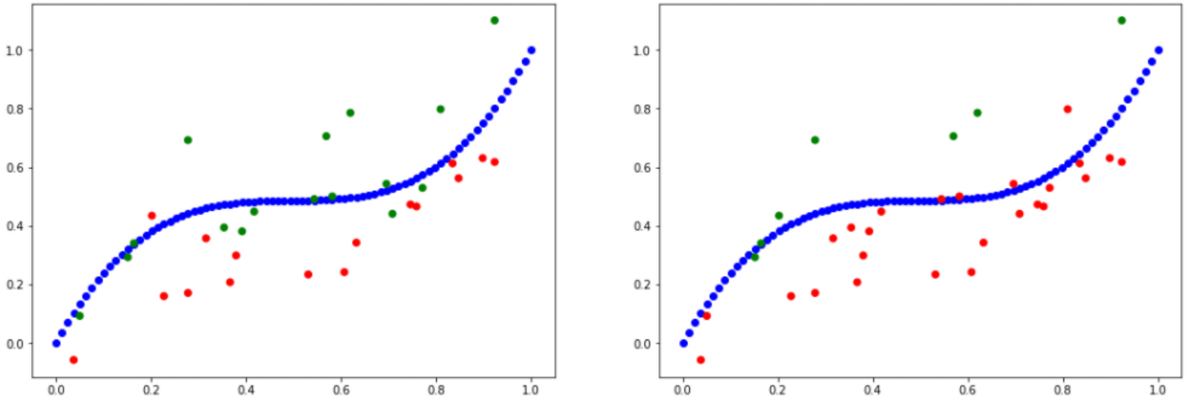
4.1. Classification with MSE loss

Model performance on test data



correct No.= 23 , accuracy = 0.72

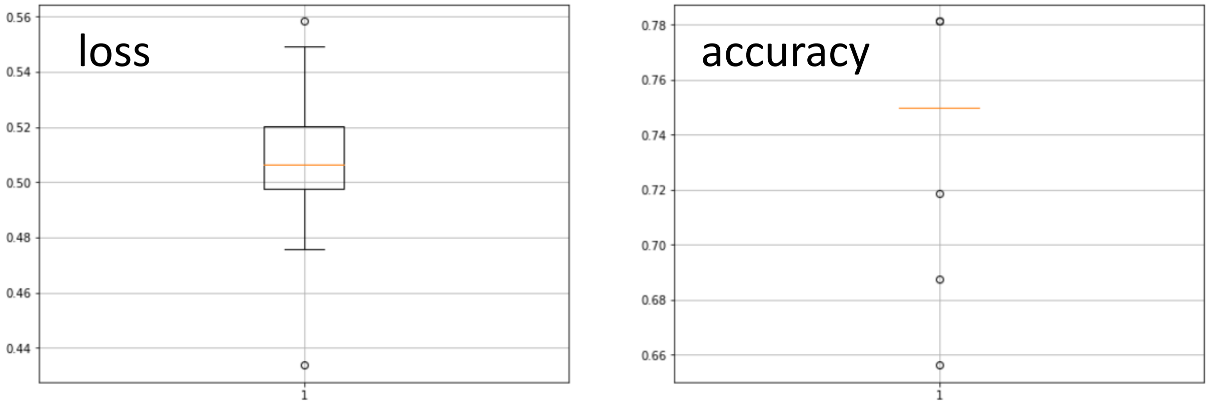
4.2. Classification with CE loss



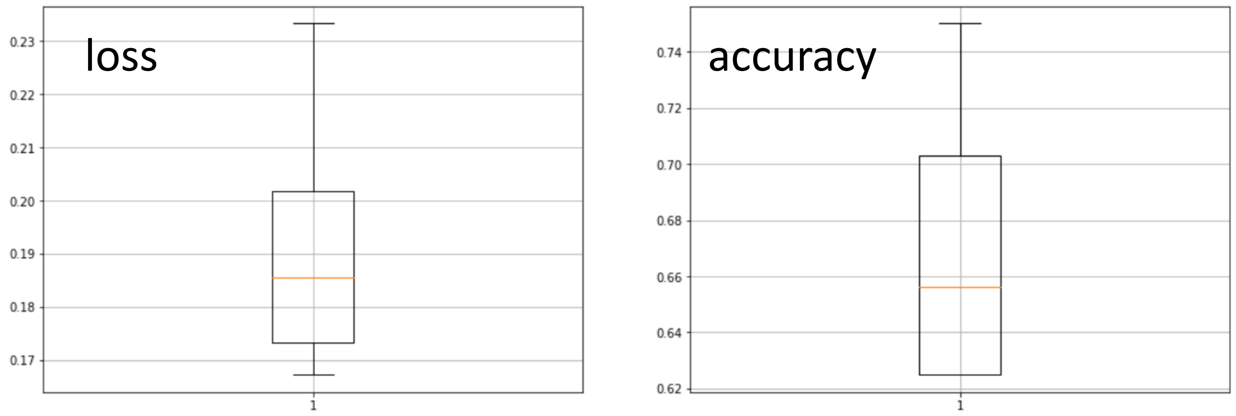
correct No.= 21 , accuracy = 0.66

4.1. Classification with MSE loss

Variance of model performance on test data

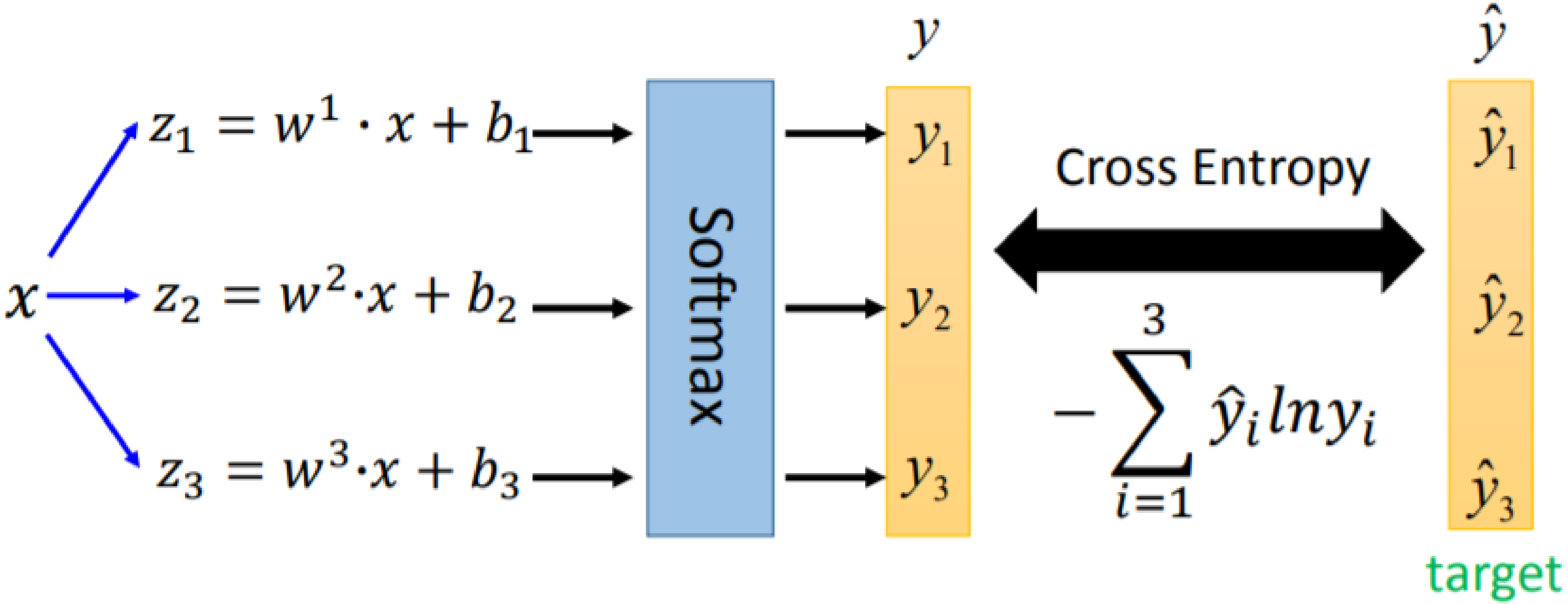


4.2. Classification with CE loss



4.1. Classification with MSE loss

Multi-class classification



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Multi-class classification

C_1: w^1, b_1 $\quad z_1 = w^1 \cdot x + b_1$

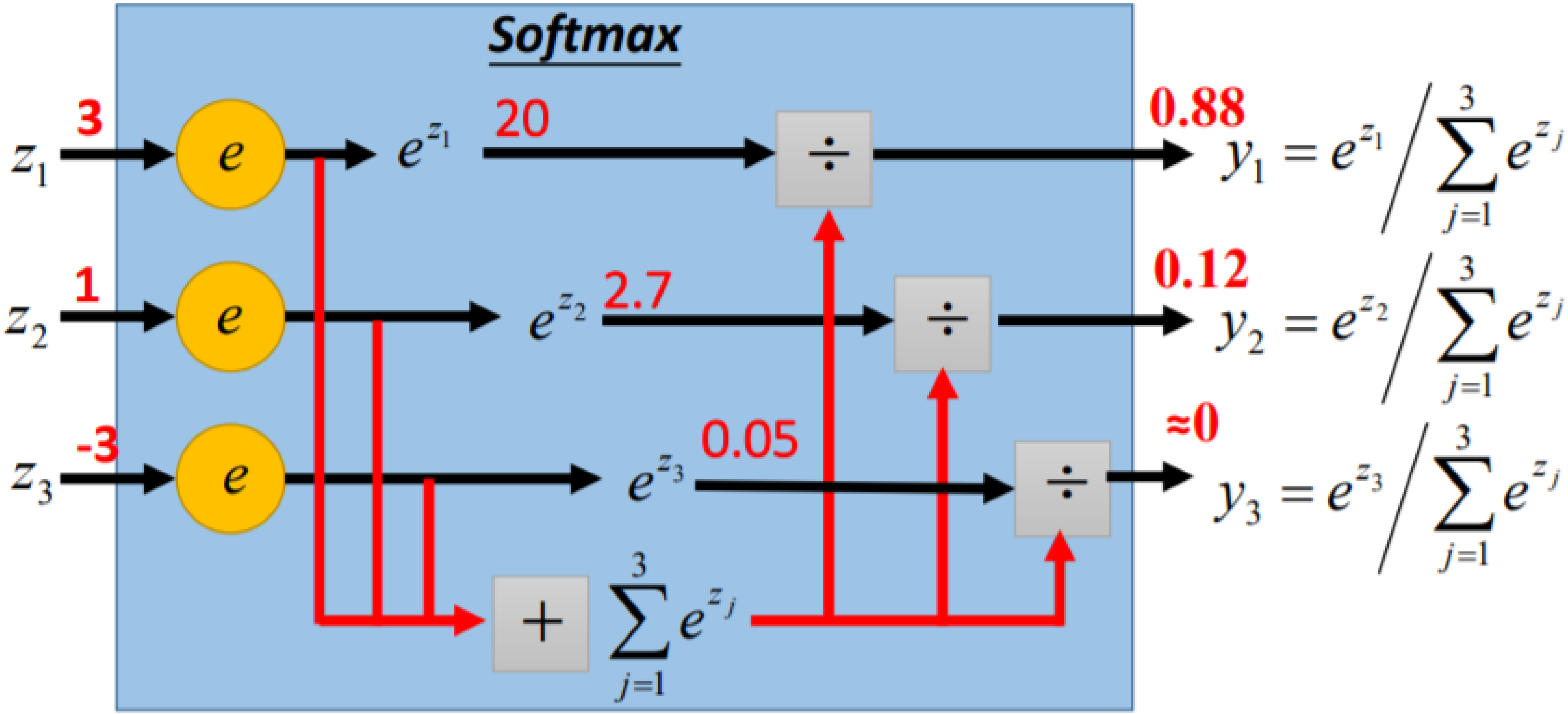
C_2: w^2, b_2 $\quad z_2 = w^2 \cdot x + b_2$

C_3: w^3, b_3 $\quad z_3 = w^3 \cdot x + b_3$

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

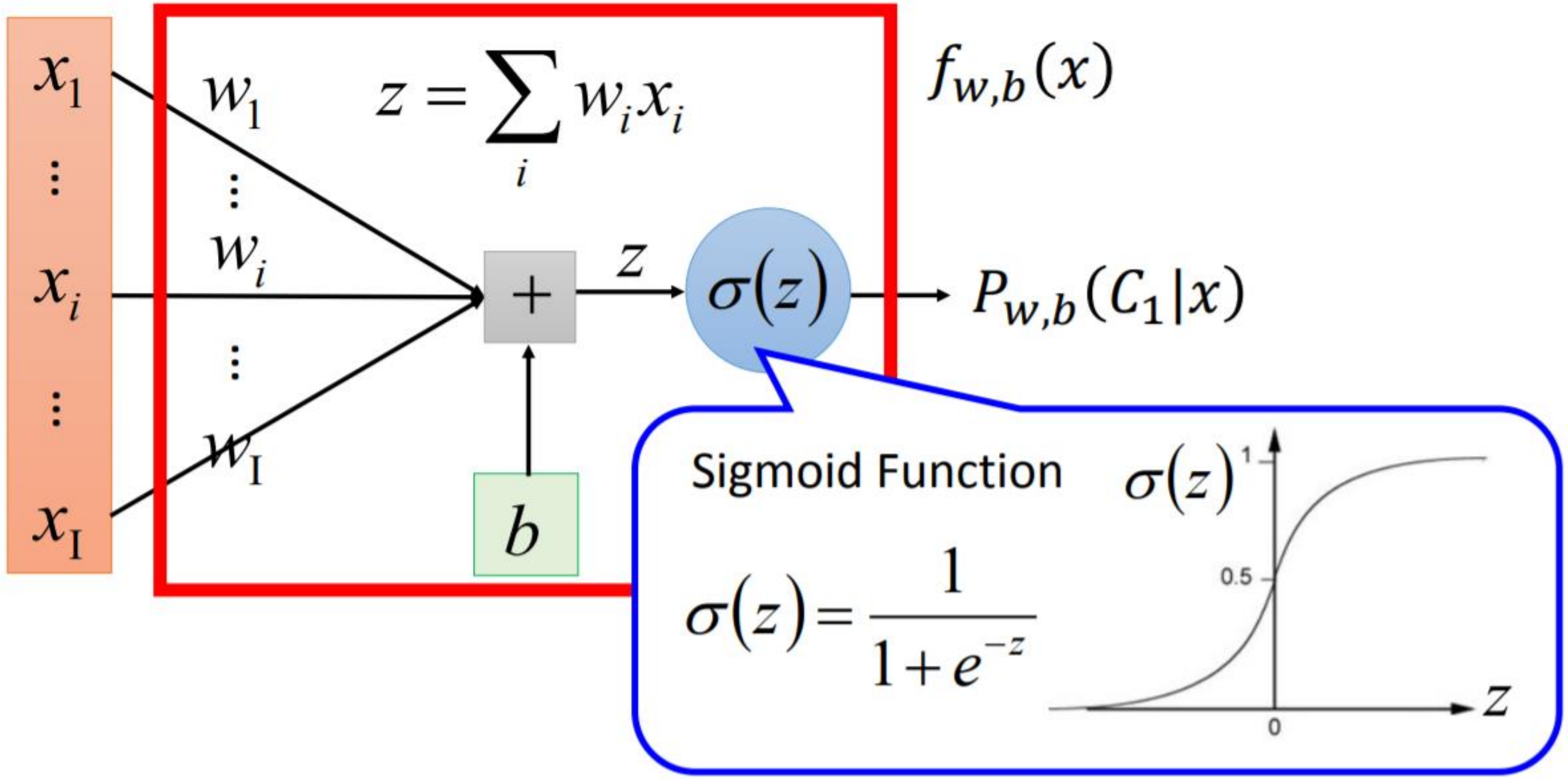
$$y_i = P(C_i \mid x)$$



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

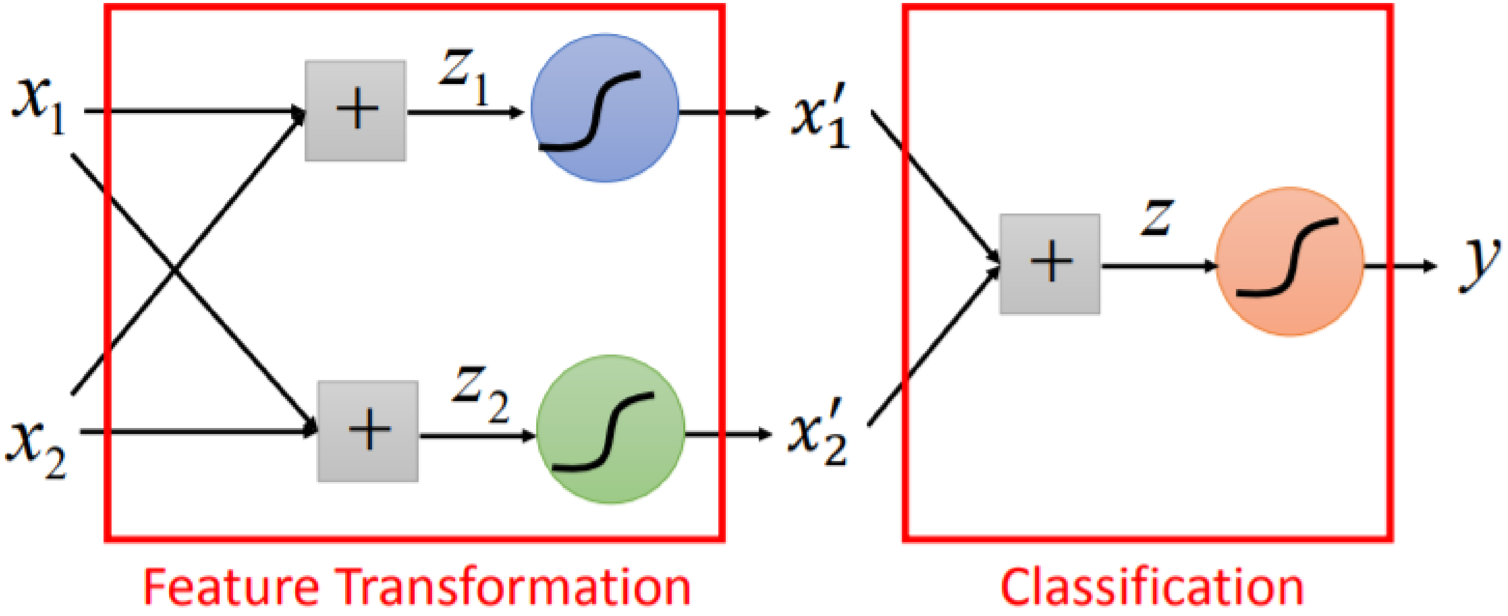
Logistic regression represented as NN

$$P(C_1 | x) = \sigma(w \cdot x + b) = \sigma\left(\sum_i w_i x_i + b\right)$$



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA

Cascading logistic regression models



Reference: 李弘毅 ML Lecture 5 https://youtu.be/hSXFuypLukA