

Design 1

Run "4.1. Classification with MSE loss.ipynb"

- NN output layer has one node y .
- Use MSE as loss function to train NN.
- After training, if $y \leq 0.5 \rightarrow \text{class1}$; if $y > 0.5 \rightarrow \text{class2}$.

```
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, 1),
        )
MyNet.to(device)
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam
```

$X = (x_1, x_2)$

$\hat{y} = 0, 1$

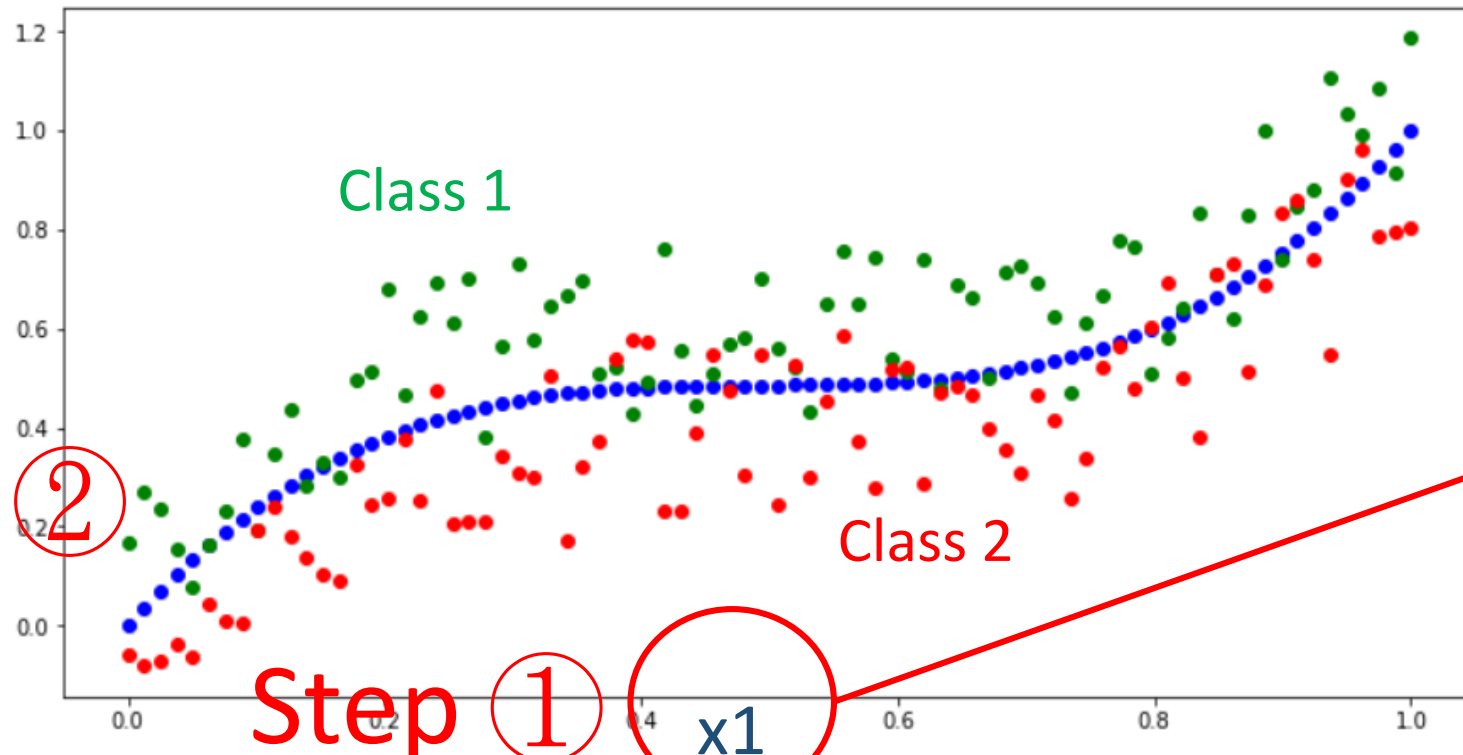
$y \leq 0.5 \rightarrow \text{class 1}$
 $y > 0.5 \rightarrow \text{class 2}$

$$L(w, b) = \sum_{n=1}^N (\hat{y}^n - y^n)^2$$

005)

Generate data

```
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))
```



```
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

Step ③

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)
```

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()
```

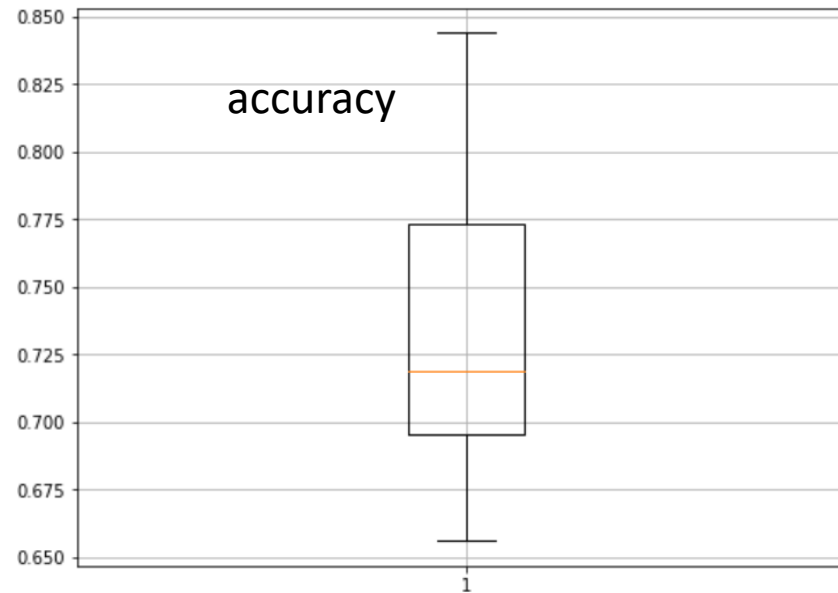
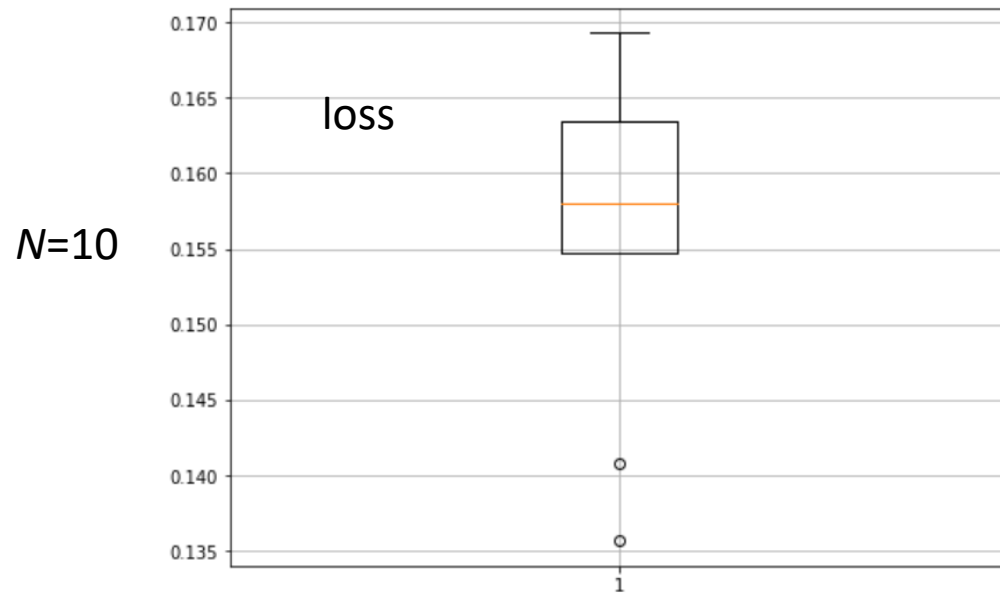
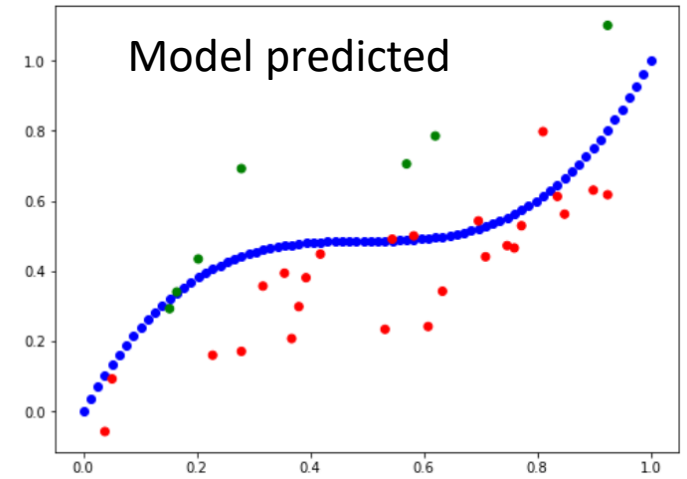
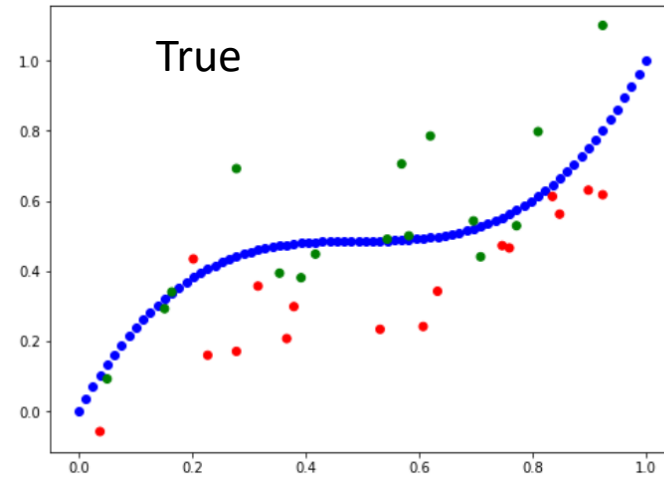
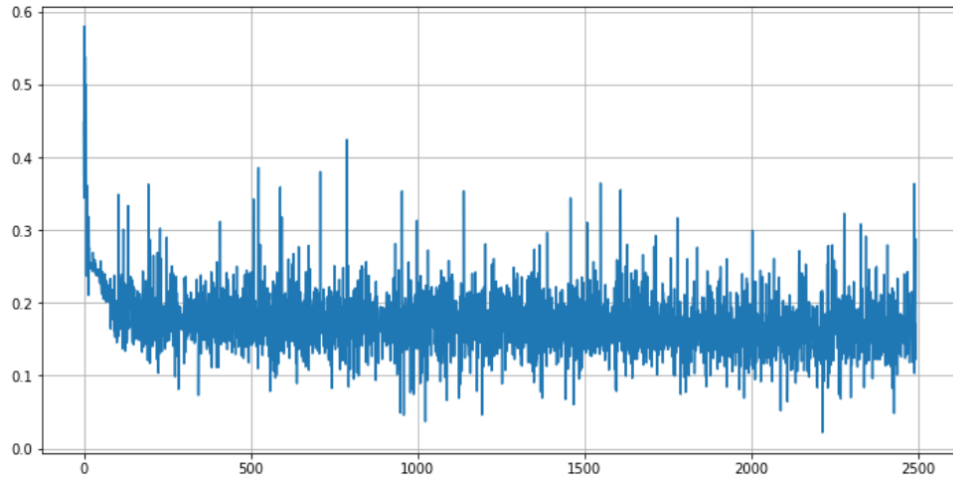
Train with whole data

```
lossLst = []
for epoch in range(1, 2000):
    tensorY = MyNet(tensorX)
    loss = loss_func(tensorY_hat, tensorY)
    loss1 = float(loss)
    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
```

Performance visualization



Design 2

Run " 4.2. Classification with CE loss"

- NN output layer contains two nodes, y_1 and y_2 , where $y_1 = P(C_1|x)$, $y_2 = P(C_2|x)$.
- Use cross entropy as loss function to train NN.

```
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), 2 classes
)
MyNet.to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(MyNet.parameters(), lr=0.005)
```

Entropy

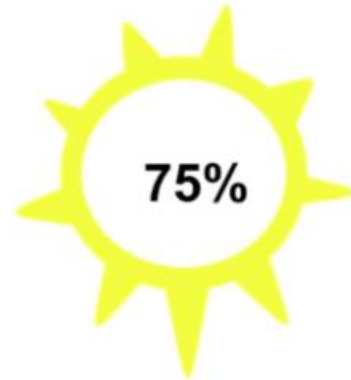
More information → more uncertain → larger entropy

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

Excel formula: =LOG(N, 2)

LOG(2, 2)=1

LOG(0.5, 2) = -1



Use Excel to verify

$$\begin{aligned} &75\% \times 0.41 \\ &+ 25\% \times 2 \\ &= 0.81 \text{ bits} \end{aligned}$$



Cross entropy

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

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

Excel formula: =LN(x)

Use Excel to verify

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			

CE vs MSE

Use Excel to compare CE vs MSE

p_i : Red Panda 99.4, others 0.01

q_i : Cat 99.4, others 0.01

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

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			

We can use Bayesian's rule to derive $y_i = p(C_i|x)$

$$y_1 = P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

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

Probabilistic Generative Model

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

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

$$P(C_1) = 79 / (79 + 61) = 0.56$$

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

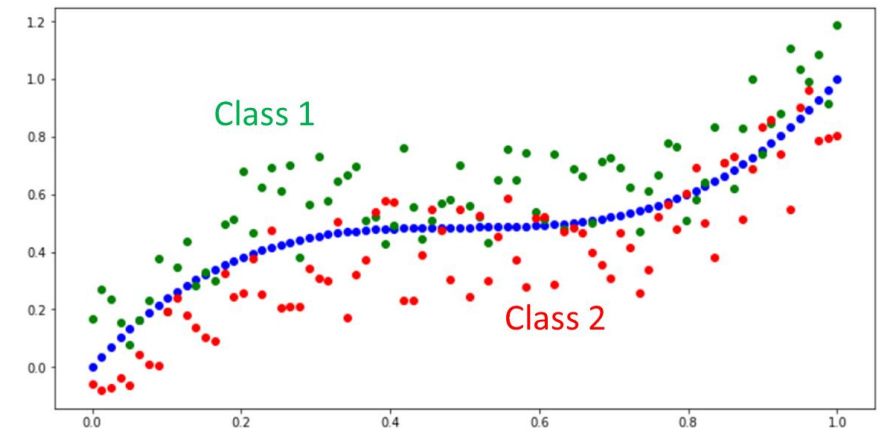
$$f_{\mu^2, \Sigma^2}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^2|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu^2)^T (\Sigma^2)^{-1} (x - \mu^2)\right\}$$

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

$$P(C_2) = 61 / (79 + 61) = 0.44$$

If $P(C_1|x) > 0.5$

Assuming x^n are sampled from a Gaussian distribution, then we can use maximum likelihood to find the best Gaussian distribution behind them.



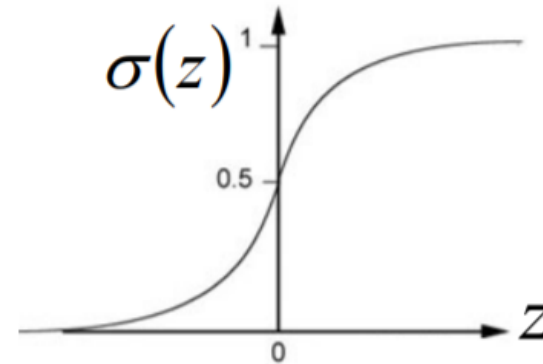
Posterior probability can be represented as sigmoid of z

$$y_1 = P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

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

Sigmoid function

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



Posterior probability represented as sigmoid of linear combination of x

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

Assuming the covariance matrices of the two classes are the same

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

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

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

$$y_1 = P(C_1|x) = \sigma(\mathbf{w} \cdot x + b)$$

How about directly find \mathbf{w} and b ?

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

Then we have \mathbf{w} and b

Logistic Regression

If we use gradient decent to find optimal w and b for the posterior probability $y_1 = p(C_1|x) = \sigma(w \cdot x + b)$, then the problem becomes logistic regression.

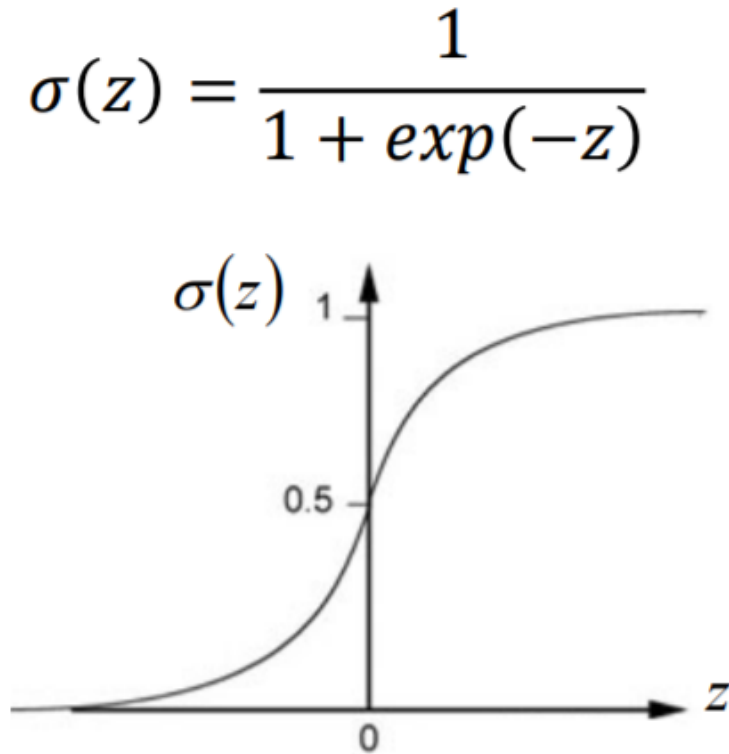
We want to find $P_{w,b}(C_1|x)$

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

Otherwise, output C_2

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

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



Logistic Regression vs Regression

Logistic Regression

$$f_{w,b}(x) = \sigma \left(\sum_i w_i x_i + b \right)$$

Output: between 0 and 1

Linear Regression

$$f_{w,b}(x) = \sum_i w_i x_i + b$$

Output: any value

Loss function for logistic regression

Assuming the training data is generated from $y_1 = P_{w,b}(C_1 | x) = \sigma(w \cdot x + b)$, what is the probability of generating the data?

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

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

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

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

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

Loss function for logistic regression vs regression

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:

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

Generate training data

```
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



Calculate 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)
```

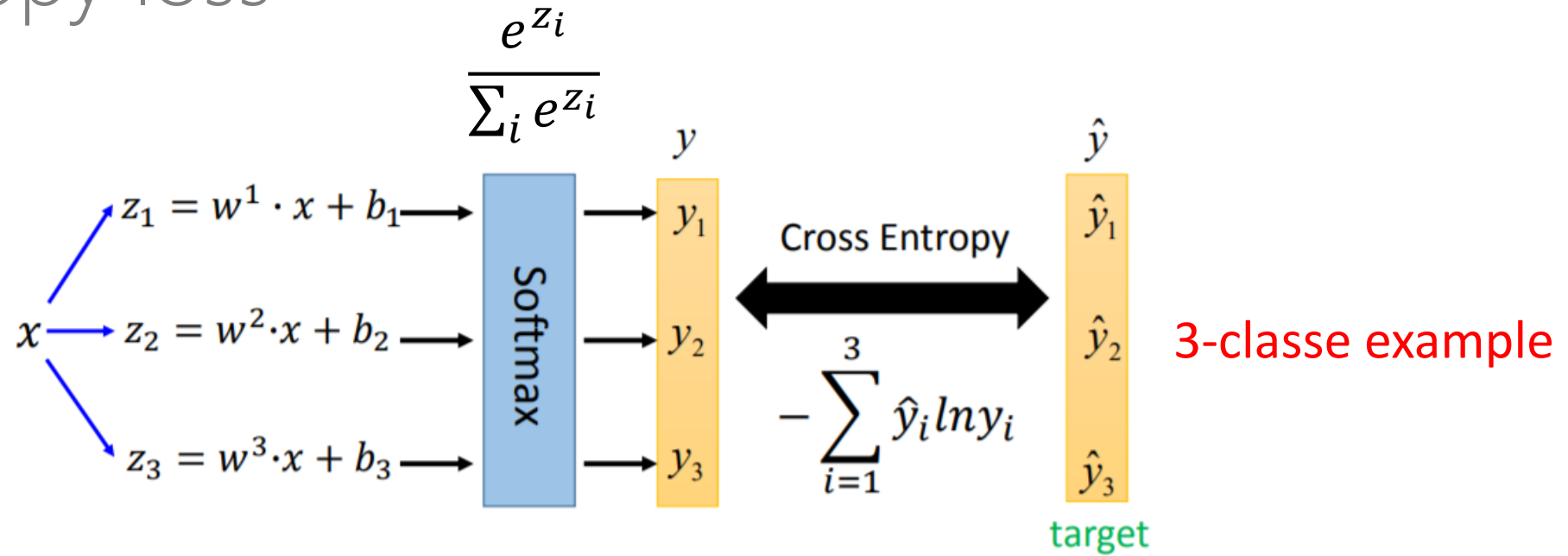
```
torch.Size([5, 2])
tensor([[ -0.0180,  0.0855],
        [ -0.0244,  0.0741],
        [ -0.0187,  0.0850],
        [ -0.0258,  0.0687],
        [ -0.0267,  0.0617]], device='cuda:0', grad_fn=<AddmmBackward>)
```

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

[illegible]

Soft max and torch.max

```
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.softmax(tensor, 1)`

0: ↓

1: →

`torch.max(tensor, 1)`

$$\frac{e^{y_1}}{e^{y_1} + e^{y_2}}$$

$$\frac{e^{y_2}}{e^{y_1} + e^{y_2}}$$

Torch.max

```
tensor([[0.4742, 0.5258],
        [0.4754, 0.5246],
        [0.4741, 0.5259],
        [0.4764, 0.5236],
        [0.4779, 0.5221]],
```

`torch.max(tensor, 1)[1]`

[1]: The 2nd item of
torch.max results

```
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()

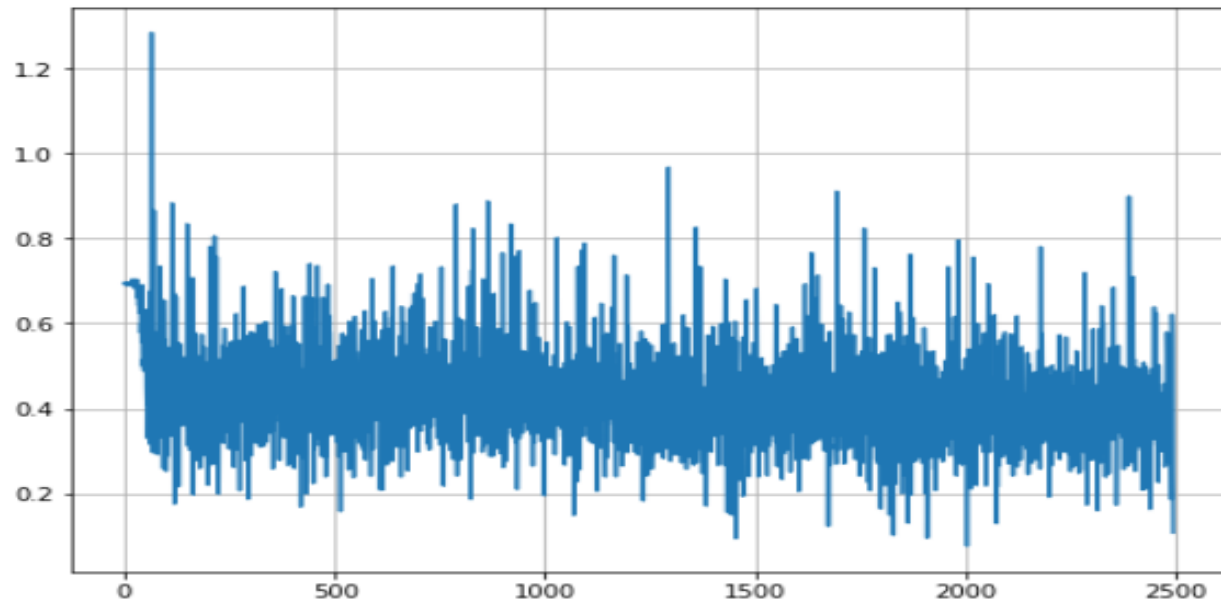
        correct = 0
        MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
        for i in range(batchY_hat.shape[0]):
            if (int(MaxIdxOfEachRow[i]) == int(batchY_hat[i])):
                correct += 1
        accuracy = correct/batchY_hat.shape[0]
        accuracyLst.append(accuracy)
```

4.2. Classification with CE loss

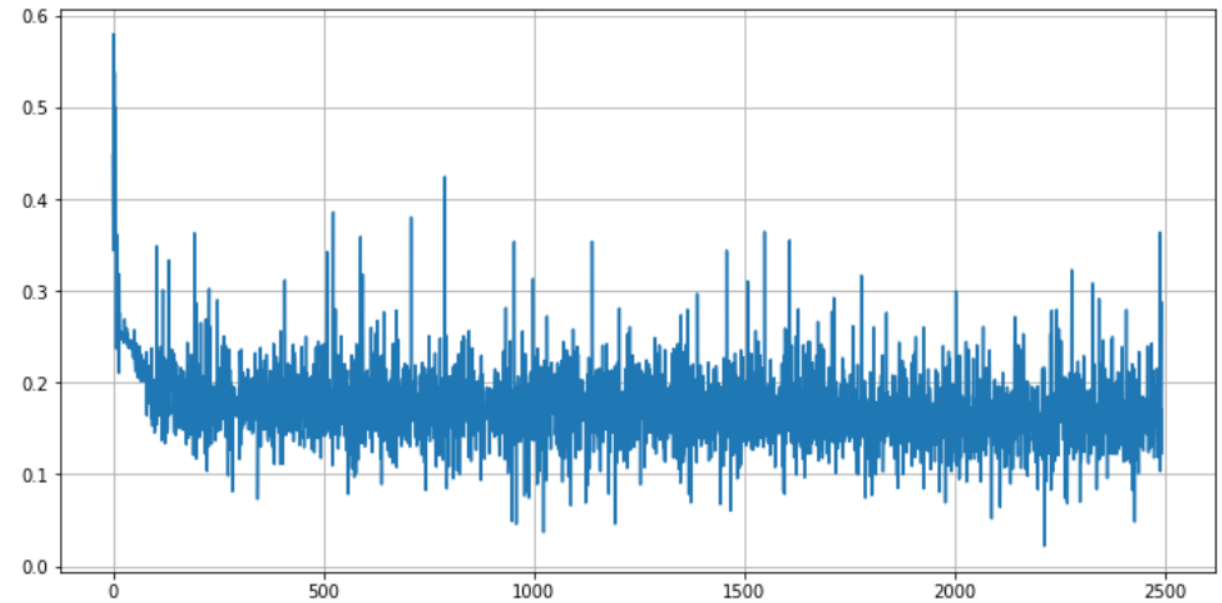
```
for epoch in range(1, 500):
    for (batchX, batchY_hat) in loader:
        tensorY = MyNet(batchX)
        loss = loss_func(batchY_hat, tensorY)
        lossLst.append(float(loss))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

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

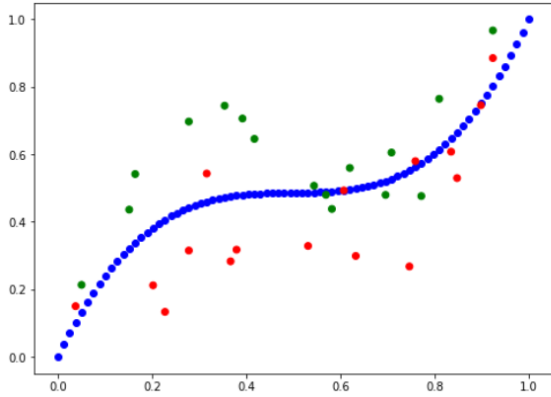
```
# show model predicted classification
lstColor = []
correctNo = 0
MaxIdxOfEachRow = torch.max(tensorY, 1)[1]
for i in range(tensorY.shape[0]):
    if (int(MaxIdxOfEachRow[i]) == 0):
        lstColor.append("green")
        if(int(testY_hat[i])==0):
            correctNo += 1
    else:
        lstColor.append("red")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

4.2. Classification with CE loss

```
# 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
```

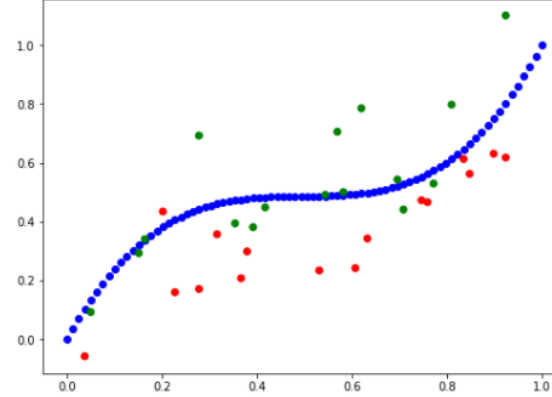
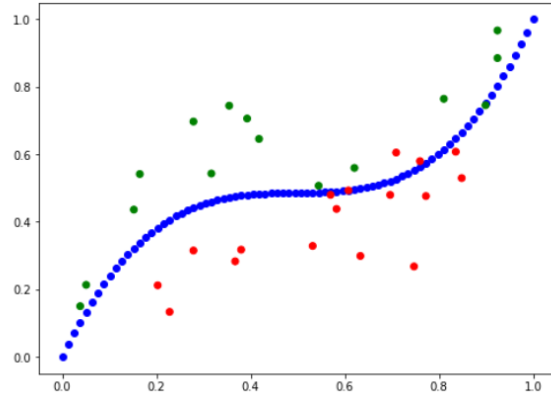
4.1. Classification with MSE loss

Model performance on test data



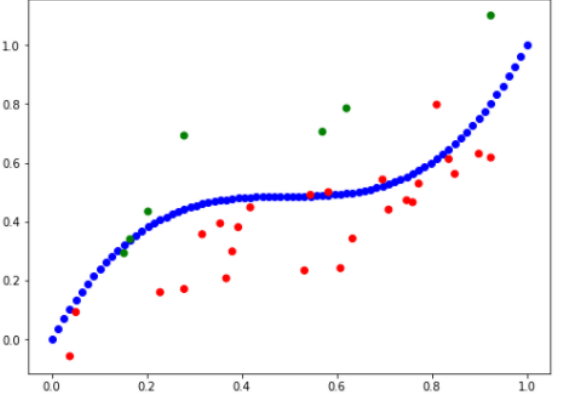
correct No.= 23 , accuracy = 0.72

4.2. Classification with CE loss

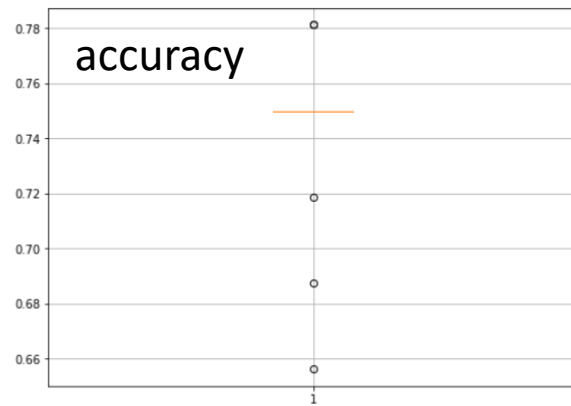
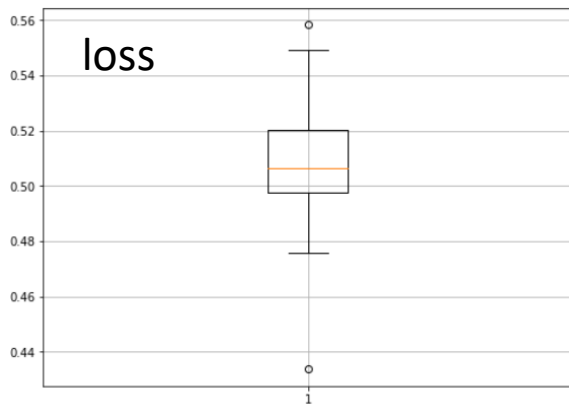


correct No.= 21 , accuracy = 0.66

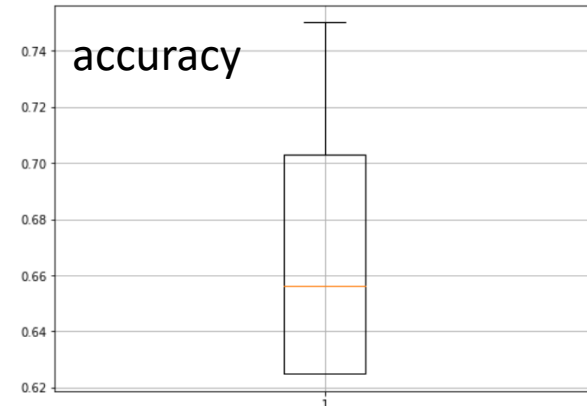
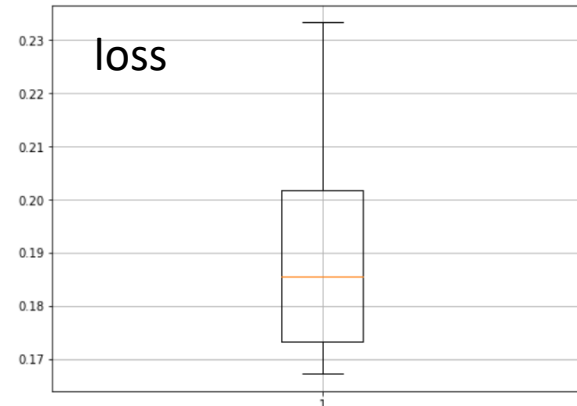
4.1. Classification with MSE loss



Variance of model performance on test data

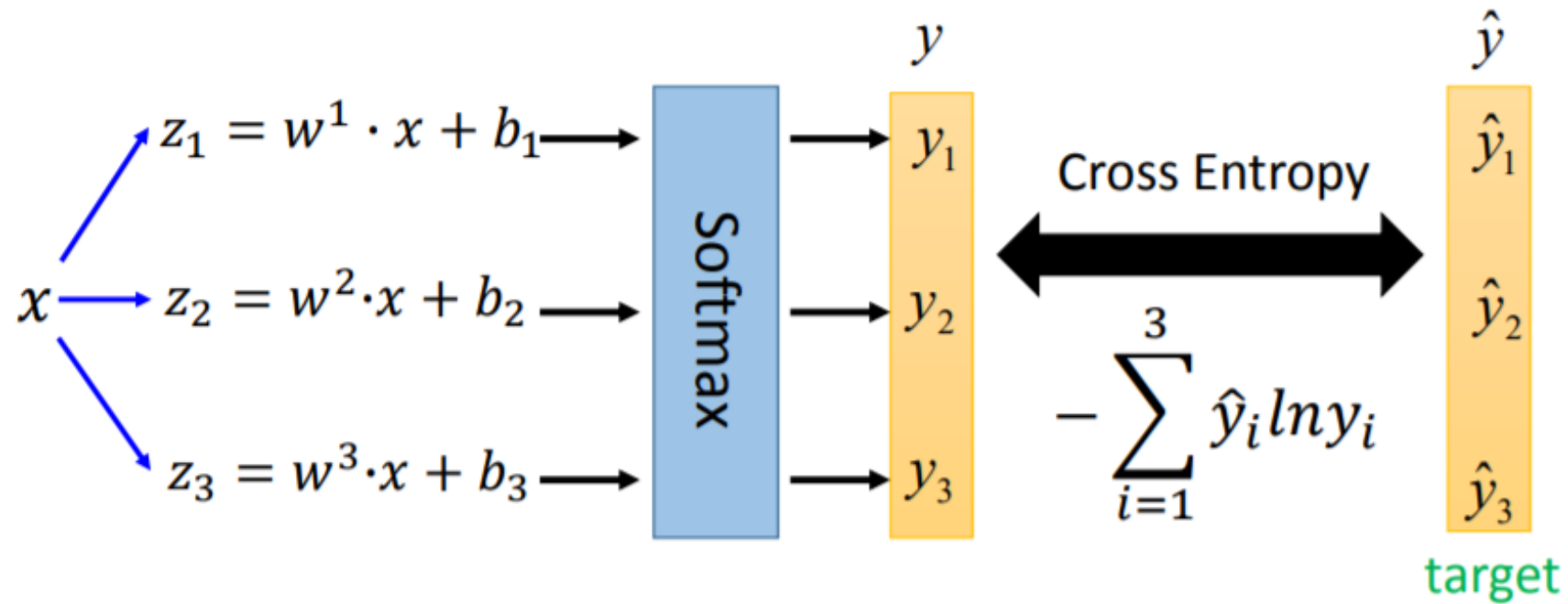


4.2. Classification with CE loss



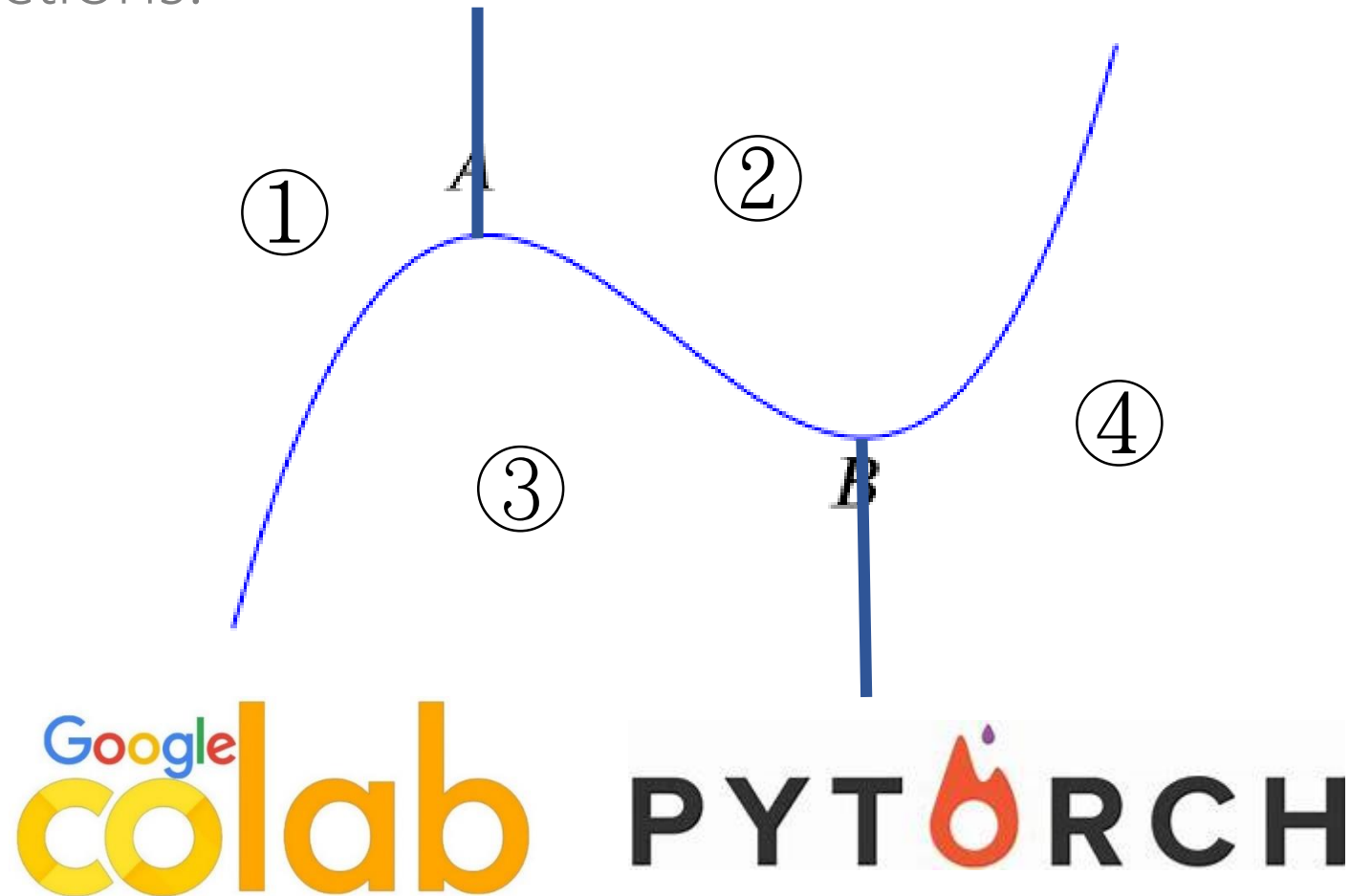
4.1. Classification with MSE loss

Multi-class classification



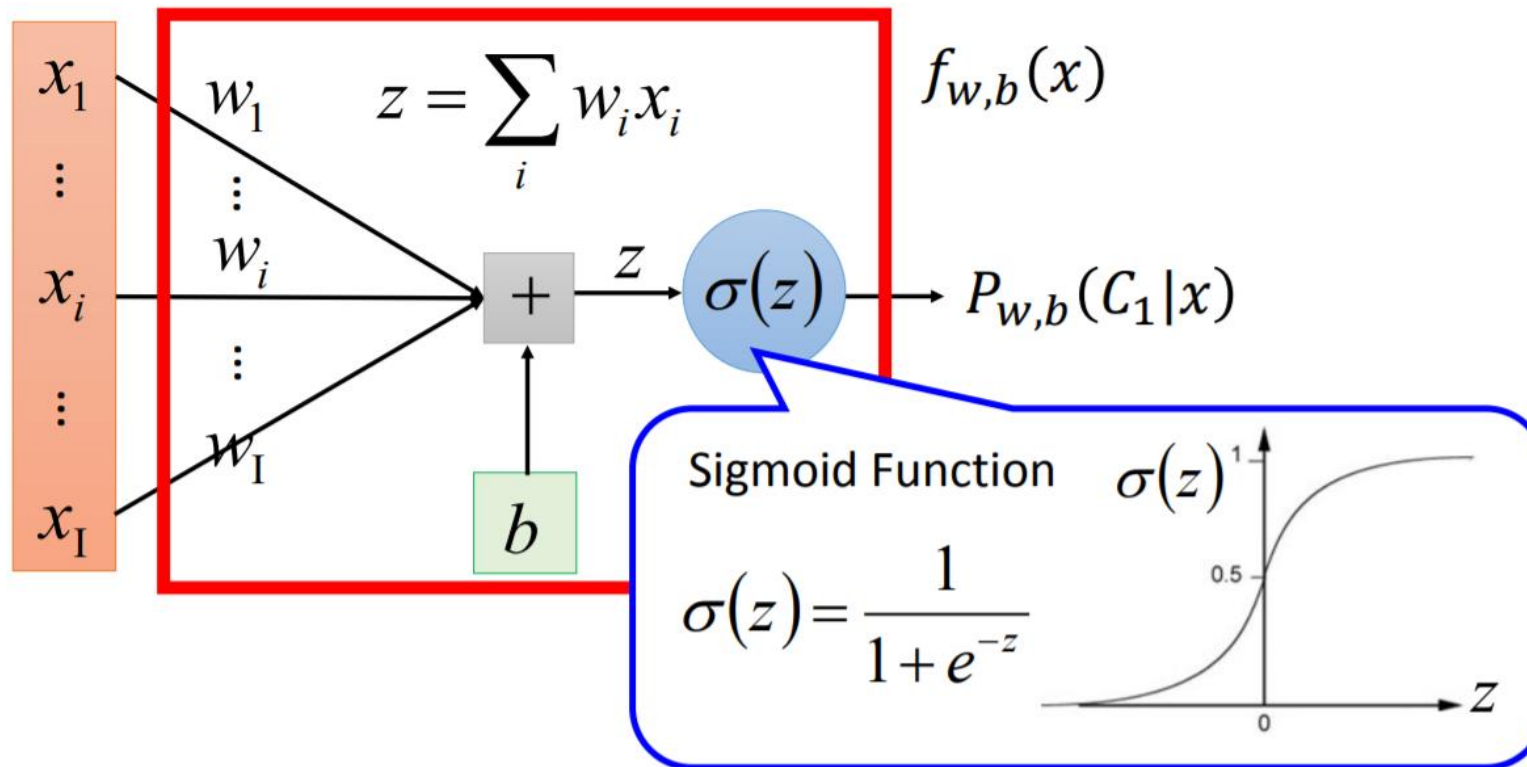
HW4

- Extend the example to 4 classes. Compare the classification performance (loss plot, scatter plot, box plot) between MSE and CE loss functions.



Each neuron in a NN performs logistic regression to classify its inputs

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



A neural network can be seen as cascading logistic regression models

