Homework: GLM and MLE

Problem 1

Follow the following steps, and answer the questions.

Step 1: Data loading

```
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader, TensorDataset
import pandas as pd

from sklearn.datasets import load_breast_cancer
cancer1 = load_breast_cancer()

print("Predictors: ", cancer1.feature_names)
print("\nResponse: ", cancer1.target_names)



cancer = pd.DataFrame(cancer1.data, columns=cancer1.feature_names)
cancer.columns = cancer.columns.str.replace(' ','_')
cancer.shape
```

```
Predictors:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']

Response:  ['malignant' 'benign']
(569, 30)
```

```
cancer1.target[:50] # 1 means malignant.
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1])
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

더블클릭 또는 Enter 키를 눌러 수정

Step 2: Preparing dataloader and train-test split

```
from sklearn.model_selection import train_test_split

X = torch.tensor(cancer.values, dtype=torch.float32)[:,[0,1,4]]
Y = torch.tensor(cancer1.target, dtype=torch.float32).reshape([-1,1])
#정규화 하기 전에 나눈 거/ Split the data into training and test sets, train/test data를 보면 안
X_unnormal_train, X_unnormal_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.
#train에 이름을 붙인 거.


X_train_mean = torch.mean(X_unnormal_train, axis=0)
X_train_std = torch.std(X_unnormal_train, axis=0)
```

```
X_train = ( X_unnormal_train -  X_train_mean ) / X_train_std
X_test = ( X_unnormal_test - X_train_mean ) / X_train_std #정규화



# Convert the data to TensorDataset to use with DataLoader

```

```
train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=True)
print(X_train.shape, Y_train.shape)
```

```
torch.Size([398, 3]) torch.Size([398, 1])
```

Question 1-1:

Using the data above, we want to build the logistic regression model to predict whether the tumor is malignant or benign. Use `mean_radious`, `mean_texture`, and `mean_smoothness` as explanatory variables. Name this model as **Model 1**. Train Model 1. You want to predict the probability of malignant for the first person `cancer.iloc[0, :]`. Calculate the probability of malignant for the first person.

Caution: this is not the normalized data.

```
X_unnormal_train[:1, :] # mean_radious, mean_texture, mean_smoothness

#환자의 병상태 파악

```

```
tensor([[13.7400, 17.9100,  0.0794]])
```

```
# Hint: Your answer is comparable to the following results.
# In my case (I am Jaeyoun), I cannot have the same weight.
!pip install ISLP
import statsmodels.api as sm
from ISLP import load_data

# Logistic regression
model = sm.GLM(Y_train.numpy(), X_train.numpy(), family=sm.families.Binomial())
results = model.fit()
print(results.summary())
```

```
  Building wheel for autograd-gamma (setup.py) ... done
  Created wheel for autograd-gamma: filename=autograd_gamma-0.5.0-py3-none-any.whl size=4031 sha256=5a3e8b6f261b96ab
  Stored in directory: /root/.cache/pip/wheels/25/cc/e0/ef2969164144c899fedb22b338f6703e2b9cf46eeebf254991
Successfully built autograd-gamma
Installing collected packages: scipy, lightning-utilities, interface-meta, autograd-gamma, torchmetrics, pygam, formu
  Attempting uninstall: scipy
    Found existing installation: scipy 1.13.1
    Uninstalling scipy-1.13.1:
      Successfully uninstalled scipy-1.13.1
Successfully installed ISLP-0.4.0 autograd-gamma-0.5.0 formulaic-1.0.2 interface-meta-1.3.0 lifelines-0.30.0 lightnin
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  398
Model:                            GLM   Df Residuals:                      395
Model Family:                Binomial   Df Model:                            2
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -82.385
Date:                Fri, 08 Nov 2024   Deviance:                       164.77
Time:                        15:50:51   Pearson chi2:                     204.
No. Iterations:                     8   Pseudo R-squ. (CS):             0.5969
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -4.6121      0.540     -8.546      0.000      -5.670      -3.554
x2            -1.3166      0.230     -5.719      0.000      -1.768      -0.865
x3            -1.6985      0.253     -6.704      0.000      -2.195      -1.202
==============================================================================
```

Question 1-2:

Consider the first person, `X_unnormal_train[:1, :]`, whose mean_radius has increased from 13.740000 to 19.00000. What is the probability of malignant in Model 1 assuming all other explnatory variable does not change.

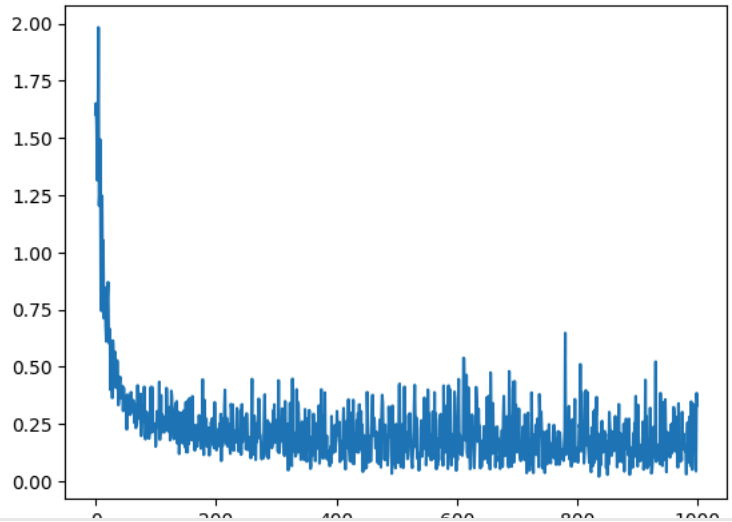
```
# Answer:
#악성 1로, 설명변수는 반지름, 질감, 딱딱함 정도 를 가지고 암인지 아닌지 판단 /암일 확률
#y가 categorical
beta=torch.tensor(torch.randn([3,1]),requires_grad=True)

bias=torch.tensor(torch.randn([1,]),requires_grad=True)

def forward(x): #x: [n,2]
  return torch.sigmoid( x @ beta + bias)


def loss_ftn(phat,y):
  return -torch.mean(y*torch.log(phat)+(1-y)*torch.log(1-phat))


  #m=torch.distributions.bernoulli.Bernoulli() #분포 정의
  #return -torch.mean(m.log_prob(y)) #negative log likelihood

lr=0.01
epochs=1000
history=[]
optimizer=torch.optim.SGD([beta,bias],lr=lr)

n_train=len(train_loader.dataset)#1000번정도 러닝

#def train(train_loader,forward,optimizer,loss_ftn,epochs)


for i in range(epochs):
  LOSS=0
  for xx,yy in train_loader:
    phat=forward(xx)
    loss=loss_ftn(phat,yy)
    LOSS+loss.item()*len(xx) #숫자로 더해 줌
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

```
38
39   history.append(loss.item())
40
```

```
<ipython-input-16-441ed458f510>:4: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  beta=torch.tensor(torch.randn([3,1]),requires_grad=True)
<ipython-input-16-441ed458f510>:6: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  bias=torch.tensor(torch.randn([1,]),requires_grad=True)
```

```
1 import matplotlib.pyplot as plt
2 plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x7c0fc5b72b30>]
```




```
1 myX=(X_unnormal_train[:1, :]-X_train_mean)/X_train_std
2 forward(myX)
```

```
tensor([[0.9733]], grad_fn=<SigmoidBackward0>)
```

```
1 add=torch.tensor([[5.26,0,0]],dtype=torch.float32)
2 myx=(X_unnormal_train[:1,:]+add-X_train_mean)/X_train_std
3 forward(myx)
```

```
tensor([[0.1285]], grad_fn=<SigmoidBackward0>)
```

Question 1-3:

Using the data above, we want to build the logistic regression model to predict whether the tumor is malignant or benign. Use `mean_radious` as the only explanatory variable. Name this model as Model 2. Train Model 2, and cauclate the test accuracy of Model 1 and 2. In terms of test accuracy which model is better?

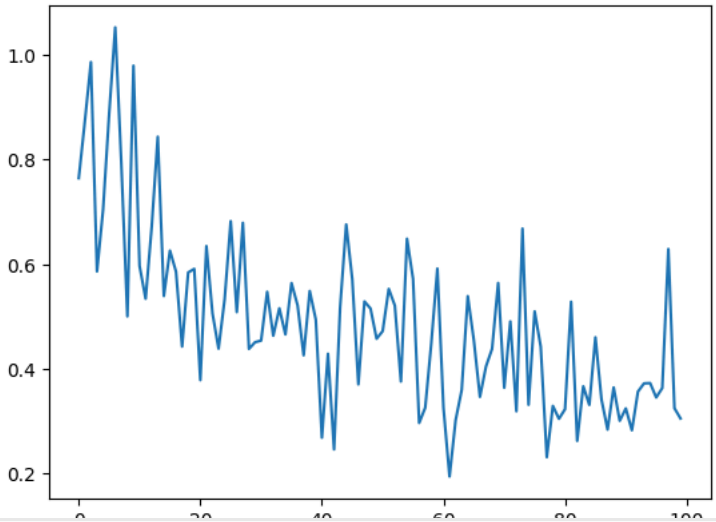
```
1 #반지름만 설명변수 Accuracy 계산
2 n_test=len(test_loader.dataset)
3 Correct_sum1=0
4 Correct_sum2=0
5 for xx, yy in test_loader:
6   phat1=forward(xx)
7   phat2=forward(xx[:,:1])
8
9   yhat1=(phat>0.5)*1.0
10  yhat2=(phat2>0.5)*1.0
11  Correct_sum1+=torch.sum(yhat1==yy)
12  Correct_sum2+=torch.sum(yhat2==yy)
13
14 Correct_sum1/n_test
15 Correct_sum2/n_test
```

```
#섞명변수, n,1만 사용하도록 슬라이싱

def forward2(x):
  return torch.sigmoid(x@beta2+bias)
beta2=torch.tensor(torch.randn([1,1]),requires_grad=True)
bias2=torch.tensor(torch.randn([1,]),requires_grad=True)

lr=0.01
epochs=100
history=[]
optimizer=torch.optim.SGD([beta2,bias2],lr=lr)


for i in range(epochs):
  LOSS=0
  for xx,yy in train_loader:
    phat=forward2(xx[:,:1]) #첫번째만 사용하는 것
    loss=loss_ftn(phat,yy)
    optimizer.step()
    optimizer.zero_grad()
    loss.backward()

    LOSS+=loss.item()*len(xx)
  history.append(loss.item())


#n_train=len(train_loader.dataset)#1000번정도 러닝
```

```
<ipython-input-34-a1e1468f7309>:5: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  beta2=torch.tensor(torch.randn([1,1]),requires_grad=True)
<ipython-input-34-a1e1468f7309>:6: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  bias2=torch.tensor(torch.randn([1,]),requires_grad=True)
```

```

plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x7c0fc52fae30>]
```



Problem 2:

Step 1: Preparation of data

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', '
# load dataset
pima = pd.read_csv("/content/drive/MyDrive/24-2프로그래밍/diabetes.csv", header=0, names=co
```

```
pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

다음 단계: pima변수로 코드 생성 추천 차트 보기 New interactive sheet

```
pima.shape
```

```
(768, 9)
```

```
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
X = torch.tensor(X.values, dtype=torch.float32)
Y = torch.tensor(y.values, dtype=torch.float32).reshape([-1,1])
```

Question 2-1:

`X` is the matrix for the explanatory variables, and `Y` is a corresponse response.

First, make train and test split of ratio 7:3. Normalize. Then, normalize the explanatory variables in both train and test datasets. Finally, make `train_loader` and `test_loader` of batch size 32.

```
#1, make train and test split of ratio 7:3
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
#2, normalization
ux=torch.mean(X_train,axis=0)
std=torch.std(X_train,axis=0)
X_train=(X_train-ux)/std
X_test=(X_test-ux)/std

uy=torch.mean(Y_train,axis=0)
std=torch.std(Y_train,axis=0)
Y_train=(Y_train-uy)/std
Y_test=(Y_test-uy)/std

print(Y_test.shape, X_train.shape)

#3, make dataloader
from torch.utils.data import Dataset, DataLoader, TensorDataset
train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
```

```
torch.Size([231, 1]) torch.Size([537, 7])
```

```
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_unnormal_train, X_unnormal_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.



X_train_mean = torch.mean(X_unnormal_train, axis=0)
X_train_std = torch.std(X_unnormal_train, axis=0)
X_train = ( X_unnormal_train -  X_train_mean ) / X_train_std
X_test = ( X_unnormal_test - X_train_mean ) / X_train_std


train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=True)
print(X_train.shape, Y_train.shape)
```

```
torch.Size([537, 7]) torch.Size([537, 1])
```

Question 2-2:

Train the ***Model 1*** defined by

$$Y_i \sim \text{Ber}(p_i)$$

where

$$p_i = \sigma(\beta_0 + \beta_1 X_{i,1} + \cdots + +\beta_7 X_{i,7}).$$

You may want to use the batch learning using the `train_loader`.

You should use/modify the following as a forward function:

```
model = torch.nn.Sequential(
    torch.nn.Linear(7,1, bias=True),
    torch.nn.Sigmoid()
)
yhat = model(XX)
```

```
model = torch.nn.Sequential( #이 함수는 이 순서로 통과 시켜 줌.  함수를 만들어서 지나가게 만듬.

    torch.nn.Linear(7,1, bias=True),
    torch.nn.Sigmoid())


# Use `torch.sigmoid`
```

```
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader, TensorDataset
import pandas as pd
```

```
!pip install ISLP
import statsmodels.api as sm
from ISLP import load_data
```

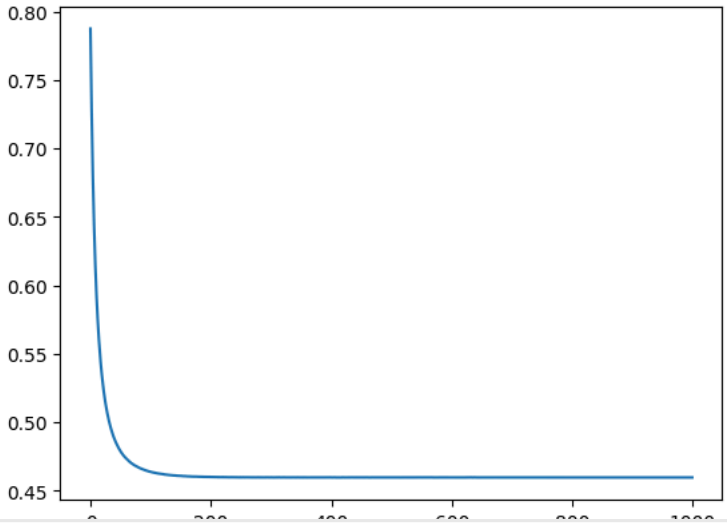
숨겨진 출력 표시

```
def loss_ftn(phat, y): #phat, y : [n,1]
  m = torch.distributions.bernoulli.Bernoulli( phat )
  return -torch.mean(m.log_prob(y))

```

```
lr=0.01
history = []
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
epochs=1000
n_train = len(train_loader.dataset)

for i in range(epochs):
  LOSS = 0
  for xx, yy in train_loader: #xx[n,3], yy[n,1]
    phat = model(xx)
    loss = loss_ftn(phat, yy)
    optimizer.zero_grad()
    LOSS += loss.item()*len(xx)
    loss.backward()
    optimizer.step()

  history.append(LOSS/n_train)
```

```
#답과 비교, 위에서 나왔던
plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x7c0fc547f9d0>]
```




```
#model.state_dict ()
```

```
OrderedDict([('0.weight',
              tensor([[-0.1957,  0.1332, -0.3096,  0.1107,  0.1944,  0.1693, -0.2827]])),
             ('0.bias', tensor([0.3393]))])
```

```
#이전에는
def model(xx):
  f1=torch.nn.Linear(7,1,bias=True)
  f2=torch.nn.Sigmoid()
  return(f2(f1(xx)))
```

```
# Hint: compare your result with the following result from the statistical optimization.
# Caution: the following data is using both train and test data.

import statsmodels.api as sm
from ISLP import load_data

# Logistic regression
model = sm.GLM(Y.numpy(), X.numpy(), family=sm.families.Binomial())
```

```
results = model.fit()
print(results.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  768
Model:                            GLM   Df Residuals:                      761
Model Family:                Binomial   Df Model:                            6
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -467.33
Date:                Fri, 08 Nov 2024   Deviance:                       934.65
Time:                        16:43:19   Pearson chi2:                     793.
No. Iterations:                     4   Pseudo R-squ. (CS):            0.07374
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.1284      0.029      4.484      0.000       0.072       0.185
x2             0.0007      0.001      1.046      0.295      -0.001       0.002
x3            -0.0047      0.010     -0.469      0.639      -0.024       0.015
x4            -0.0157      0.008     -1.861      0.063      -0.032       0.001
x5             0.0129      0.003      4.813      0.000       0.008       0.018
x6            -0.0303      0.005     -6.526      0.000      -0.039      -0.021
x7             0.3211      0.238      1.347      0.178      -0.146       0.789
==============================================================================
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

Question 2-3:

This time, train Model 1 where the `forward` function is defined as follows (properly modify it):

```
class my_logistic(torch.nn.Module):
    def __init__(self, in_units, units):
        super().__init__()
        self.linear = torch.nn.Linear(in_units, units, bias=True)

    def forward(self, X):
        ##Please fill in###
        return temp

forward = my_logistic(7, 1)
```

```
class my_logistic(torch.nn.Module):
    def __init__(self, in_units, units):
        super().__init__()
        self.linear = torch.nn.Linear(in_units, units, bias=True)

    def forward(self, X):
        ##Please fill in###
        return torch.sigmoid(self.linear(X))

forward = my_logistic(7, 1)
```

Question 2-4:

Model 1 defined by

$$Y_i \sim \text{Ber}(p_i)$$

where $$ p_i = \sigma(\beta_0 + \beta_1X_{i,1}+\cdots+

- \beta_7X_{i,7}). Train Model 2 defined by

$$Y_i \sim \text{Ber}(p_i)$$

where

$$p_i = \sigma(\beta_0 + \beta_1 Z_i).$$

where Z_i is a (standardized) explanatory variable for `bmi`. Calculate the test MSE under Model 1 and Modle 2. Which model do you prefer in terms of test MSE?

```
#1번에서 한 모델을 가지고
forward1, forward2 다시 정리
```

```
#standarized bmi
x=torch.tensor(pima['bmi'],dtype=torch.float32).reshape([-1,1])
print(x.shape)

X_train,X_test,Y_train,Y_test=train_test_split(x,Y,test_size=0.3,random_state=42)
ux=torch.mean(X_train,axis=0)
std=torch.std(X_train,axis=0)
X_train=(X_train-ux)/std
X_test=(X_test-ux)/std

print(x.shape)
train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
```

```
torch.Size([768, 1])
torch.Size([768, 1])
```

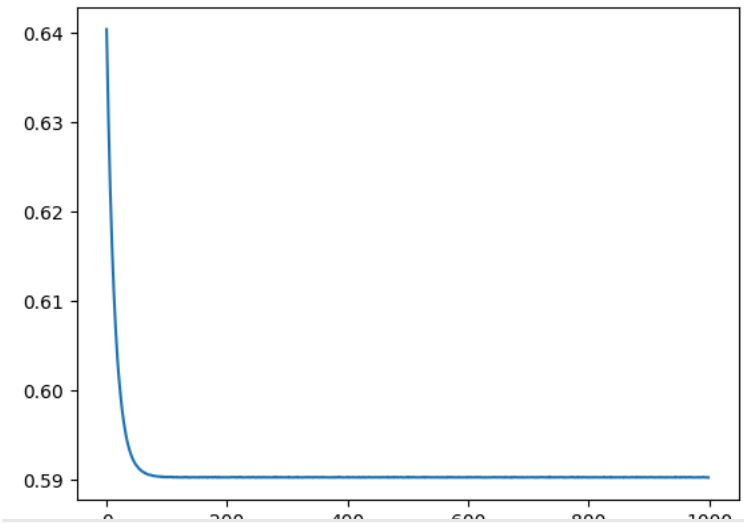
```
model2 = torch.nn.Sequential( #이 함수는 이 순서로 통과 시켜 줌.  함수를 만들어서 지나가게 만듬.

    torch.nn.Linear(1,1, bias=True),
    torch.nn.Sigmoid())
```

```
def loss_ftn(phat, y): #phat, y : [n,1]
  m = torch.distributions.bernoulli.Bernoulli( phat )
  return -torch.mean(m.log_prob(y))
```

```
lr=0.01
history = []
optimizer = torch.optim.SGD(model2.parameters(), lr=lr)
epochs=1000
n_train = len(train_loader.dataset)

for i in range(epochs):
  LOSS = 0
  for xx, yy in train_loader: #xx[n,3], yy[n,1]
    phat = model2(xx)
    loss = loss_ftn(phat, yy)
    optimizer.zero_grad()
    LOSS += loss.item()*len(xx)
    loss.backward()
    optimizer.step()

  history.append(LOSS/n_train)
```

```
plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x7c0fc535aa40>]
```




```
#Model1, Model2 MSE 비교
n_test = len(test_loader.dataset)
Correct1=0
for xx, yy in test_loader:
  phat=model(xx)
  yhat=(phat>0.5)*1.0
  Correct1+=torch.sum(yhat==yy)
accu_m1=Correct1/n_test

Correct2=0
for xx, yy in test_loader:
  phat=model2(xx[:,5].reshape([-1,1]))
  yhat=(phat>0.5)*1.0
  Correct2+=torch.sum(yhat==yy)
accu_m2=Correct2/n_test
print(accu_m1,accu_m2)

#두 번째 모델의 성능이 더 좋은 걸 알 수 있다.
```

```
tensor(0.4242) tensor(0.6364)
```

Problem 3: Heart attack patients

Suppose that we are working with some doctors on heart attack patients. The dependent variable is whether the patient has had a second heart attack (Y)within 1 year (yes = 1). We have two independent variables, one is whether the patient completed a treatment consistent of anger control practices (W_1) (yes=1). The other is a score on a trait anxiety scale (W_2) (a higher score means more anxious). You want to model

$Y \sim \text{Ber}(p_i)$

where $p_i = \sigma(x_i W + b)$ with $W = \begin{pmatrix} W_1 \\ W_2 \end{pmatrix}$.

Answer the following questions:

1. Build the following

- Model 1: Y is explained by W1 and W2

which explain the heart attack mechanism. You may want to standardize the explanatory variables before define the model.

2. Using the model, train the model.
3. What is the probability that a person of

`Treatment of Anger =1` and `Trait Anxiety =75`

will have a heart attack.

4. Calculate the train accuracy of the model.

Person	2nd Heart Attack	Treatment of Anger	Trait Anxiety
1	1	1	70
2	1	1	80
3	1	1	50
4	1	0	60
5	1	0	40
6	1	0	65
7	1	0	75
8	1	0	80
9	1	0	70
10	1	0	60
11	0	1	65
12	0	1	50
13	0	1	45
14	0	1	35
15	0	1	40
16	0	1	50
17	0	0	55
18	0	0	45
19	0	0	50
20	0	0	60

```
import numpy as np
import torch
import matplotlib.pyplot as plt
import pandas as pd
data = np.array([[1,1,70], [1,1,80], [1,1,50],[1,0,60], [1,0,40], [1, 0, 65], [1,0,75],
data
Y = data[:, :1] # 2nd Heart Attack
X = data[:, 1:] # where the first column is "Treatment of Anger" and the first column is
```

1: Y is explained by W1 and W2 which explain the heart attack mechanism. You may want to standardize the explanatory variables before define the model.

```
x=torch.tensor(X, dtype=torch.float32).reshape([-1,2])
yy=pd.get_dummies(data[:,0])
y=torch.tensor(np.array(pd.get_dummies(data[:,0])),dtype=torch.float32)[:,1:]
print(x.shape,y.shape)

```

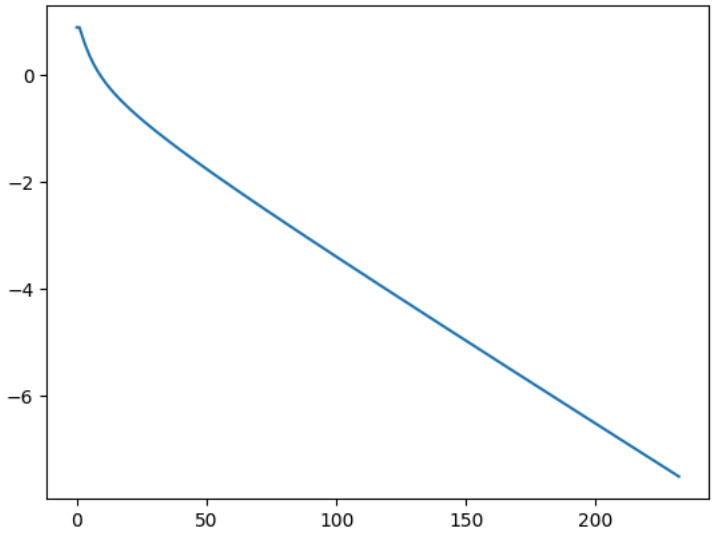
```
torch.Size([20, 2]) torch.Size([20, 1])
```

```
#standarize
ux=torch.mean(x,axis=0)
std=torch.std(x,axis=0)
x=(x-ux)/std

uy=torch.mean(y,axis=0)
stdy=torch.std(y,axis=0)
y=(y-uy)/stdy


```

```
#Using the model, train the model. ber
import torch
def forward(x):
  return torch.sigmoid(x@beta+bias)
beta=torch.tensor(torch.randn([2,1]),requires_grad=True)
bias=torch.tensor(torch.randn([1,]),requires_grad=True)
def loss_ftn(phat,y):
```

```
  return -torch.mean(y*torch.log(phat)+(1-y)*torch.log(1-phat))
#def forward(x):
 # return logistic(beta[0]+beta[1]*x+bias)
lr=0.2
history=[]
optimizer=torch.optim.SGD([beta,bias],lr=lr)
epochs=10000

for i in range(epochs):
  phat=forward(x)
  loss=loss_ftn(phat,y)
  optimizer.step()
  optimizer.zero_grad()
  loss.backward()
  history.append(loss.item())

plt.plot(history)
```

```
<ipython-input-19-87f1a6834ea7>:5: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  beta=torch.tensor(torch.randn([2,1]),requires_grad=True)
<ipython-input-19-87f1a6834ea7>:6: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  bias=torch.tensor(torch.randn([1,]),requires_grad=True)
[<matplotlib.lines.Line2D at 0x79b8132fa260>]
```




```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
xx=torch.tensor([[1],[75.0]])
xx=(xx-ux)/std

#ones=torch.ones([2,1])
#myx=torch.concat([xx,ones],axis=1)
```