به نام خدا

درس یادگیری عمیق

تمرین اول

غزل زمانی نژاد

401722244

.1

الف)

1. $\text{output} = [4, 6, -10, -20]$

$$Y = [1, 0, 0, 0]$$

الف) $\hat{y} = \text{softmax}(\text{output})$

$$= [0.119, 0.880, 9.91 \times 10^{-8}, 4.5 \times 10^{-12}]$$

$$\text{loss} = CE(Y, \hat{Y}) = 2.126$$

$$\text{loss} = -\sum_{i=1}^{N} y_i \log(\text{softmax}(\text{output})) = -\sum_i y_i \log \frac{e^{o_i}}{\sum_j e^{o_j}}$$

$$= -\sum_i y_i \Big(\underbrace{\log e^{o_i}}_{o_i} - \log \sum_j e^{o_j}\Big) = \sum_i y_i \underbrace{\log \sum_j e^{o_j}}_{\text{ضریب ثابت}} - \sum_i y_i o_i =$$

$$= \log \sum_j e^{o_j} \times \underbrace{\sum_i y_i}_{1} - \sum_i y_i o_i = \log \sum_j e^{o_j} - \sum_i y_i o_i$$

$$\frac{\partial \text{loss}}{\partial o_i} = \frac{e^{o_i}}{\sum_j e^{o_j}} - y_i = \text{softmax}(o_i) - y_i$$

$$= [-0.88, 0.88, 9.91 \times 10^{-8}, 4.5 \times 10^{-12}]$$

ب)

ب) $\hat{y} = \text{sigmoid}(\text{output}) = [0.98, 0.99, 4.53\times10^{-5}, 2.06\times10^{-9}]$

$$loss = BCE(y, \hat{y}) = 6.02$$

$$\frac{\partial loss}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial output_i} = \left(-\left(y_i \frac{1}{\hat{y}_i} + (1-y_i)\frac{-1}{(1-\hat{y}_i)}\right)\right) \times \frac{-(-e^{-o_i})}{(1+e^{-o_i})^2} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right) \sigma(o_i)(1-\sigma(o_i))$$

$$= -\frac{y_i - \hat{y}_i}{\hat{y}_i(1-\hat{y}_i)} \sigma(o_i)(1-\sigma(o_i))$$

$$= [-1.79\times10^{-2}, +0.99, +4.53\times10^{-5}, +2.06\times10^{-9}]$$

2. نمودار آبی رنگ نشان دهنده تابع hinge و نمودار سبز نشان دهنده تابع cross entropy است.

تابع hinge دارای نمودار خطی است پس باید یکی از دو نمودار آبی یا نارنجی باشد. قبل از y.y_hat =1 یک تابع خطی با شیب ۱- داریم و مطابق اثبات ریاضی در نقطه y.y_hat =1 مقدار loss برابر با ۰ می شود.

تابع CE مطابق ریاضیات، در بعضی نقاط (y_hat < 0) مقداری کمتر از تابع hinge خواهد داشت و پایینتر از آن قرار می گیرد.

$$hinge = \max(0, 1 - t.y)$$

$$\hookrightarrow hinge = 0 \rightarrow 1 - t.y = 0 \rightarrow t.y \geq 1$$

$$CE = y_i \log \hat{y}_i = -y \log(1 + e^{-\hat{y}})$$

$$\hat{y} = -1 \rightarrow CE = \log(1 + e^{+1}) = 1.89$$

$$\hookrightarrow hinge = 1 - (-1) = 2 \rightarrow CE(-1) < hinge(-1)$$

.3

الف) در سلول های بخش dataset ابتدا دیتاست mnist را می خوانیم و چند نمونه از آن را چاپ می کنیم.

سپس داده را به دو بخش آموزش و ارزیابی تقسیم می کنیم.

سپس یک شبکه ساده با یک لایه flatten برای flat کردن تصاویر ورودی و یک لایه خطی دارای 28*28 نورون ورودی و ۱۰ نورون خروجی برای طبقه بندی داده ها می سازیم.

```
############
# Your code
# Start
in_features = img_tensor.shape[1] * img_tensor.shape[2]
num_classes = 10

class MNISTModel(nn.Module):
    def __init__(self, in_features=in_features, out=num_classes):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear = nn.Linear(in_features, num_classes)

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear(x)
        return logits
# End
```

در تابع run_gradient_descent بهینه سازی و یافتن بهترین پارامترهای مدل را پیاده سازی می کنیم. برای این کار از بهینه ساز stochastic gradient descent استفاده می کنیم زیرا میتواند به خوبی نقاط مینیمم را پیدا کند و پارامترهای بهینه را بیابد. همچنین از تابع ضرر cross entropy استفاده می کنیم زیرا مطابق با maximum likelihood estimation بهترین تابع ضرر برای مسائل دسته بندی چندکلاسه است.

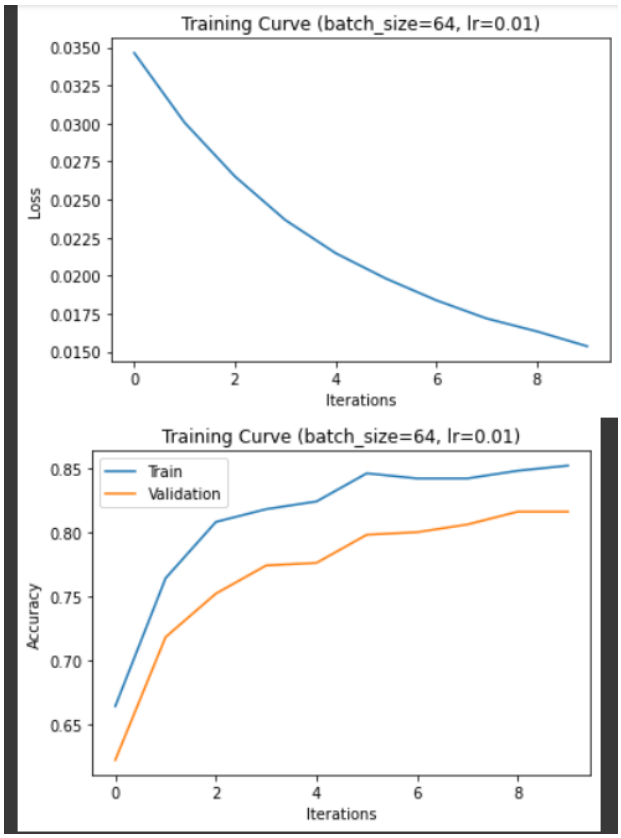
سپس در یک حلقه به تعداد epochها iterate می کنیم و مدل را آموزش می دهیم. ابتدا مدل را در حالت train قرار می دهیم. سپس batchهای مختلف از داده را لود می کنیم. و یک forward pass روی آنها اعمال می کنیم. میزان ضرر را محاسبه می کنیم و مقدار آن را ذخیره می کنیم. سپس عملیات back propagation را انجام می دهیم. مقادیر گرادیان ها را صفر می کنیم و پارامترها را آپدیت می کنیم. و بعد از هر ۱۰ batch مقادیر را چاپ می کنیم. در انتهای هر epoch مقادیر loss و دقت را ذخیره می کنیم تا در آینده آنها را پلات کنیم. و روی داده ارزیابی نیز مدل را می سنجیم.

```python
def run_gradient_descent(model,batch_size=64,learning_rate=0.01,weight_decay=0,num_epochs=10):
    #define the best optimizer and loss function

    ############
    # Your code
    # Start
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    loss_fn = nn.CrossEntropyLoss()
    # End

    iters, losses = [], []
    iters_sub, train_acc, val_acc = [], [] ,[]

    # PyTorch data loader
    train_loader = torch.utils.data.DataLoader(
        mnist_train,
        batch_size=batch_size, # batch size
        shuffle=True) # shuffle before each epoch

    size = len(train_loader.dataset)
    # training
    for epoch in range(num_epochs):

        ############
        # Your code
        # Start
        # iterate on batches
        print("Epoch", epoch)
        epoch_loss = 0
        model.train()
        for batch, (x, y) in enumerate(train_loader):
            # forward propagation
            pred = model(x)
            loss = loss_fn(pred, y)
            epoch_loss += loss.item()

            # backward propagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 10 == 0:
                loss, current = loss.item(), (batch + 1) * len(x)
                print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
        print("*"*20)

        # save loss for plot
        iters.append(epoch)
        losses.append(epoch_loss/size)
        # save accuracy for plot
        iters_sub.append(epoch)
        train_acc.append(get_accuracy(model, mnist_train))

        # evaluate model:
        model.eval()
        with torch.no_grad():
            val_acc.append(get_accuracy(model, mnist_val))
```

مدل را می سازیم و در ۱۰ ایپاک با نرخ یادگیری ۰.۰۱ و اندازه ۶۴ batch آن را آموزش می دهیم. نتیجه آموزش به شرح زیر است:

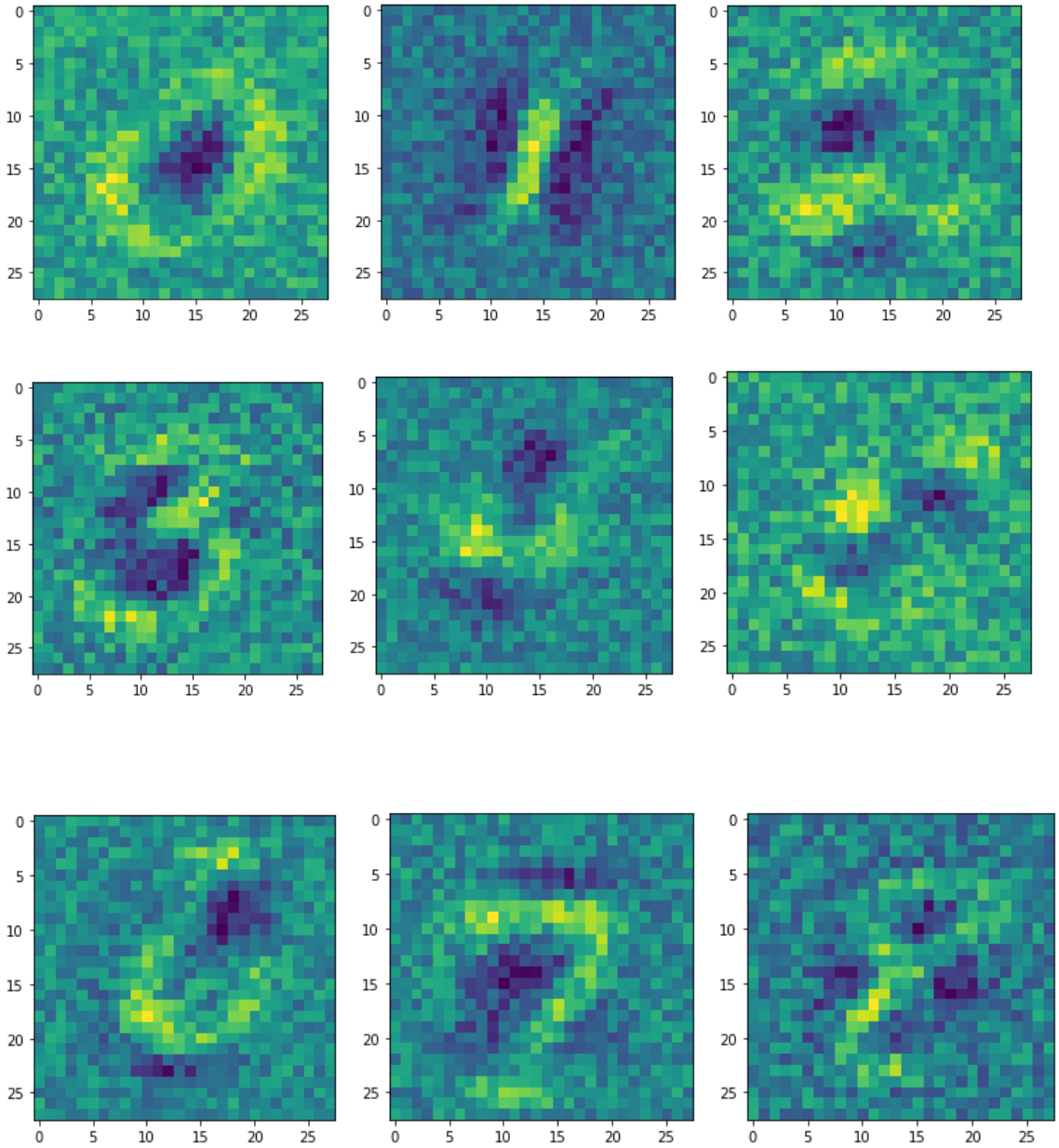
```
Epoch 0
loss: 2.319465  [   64/ 2000]
loss: 2.229431  [  704/ 2000]
loss: 2.132955  [ 1344/ 2000]
loss: 2.064138  [ 1984/ 2000]
********************
Epoch 1
loss: 1.954598  [   64/ 2000]
loss: 1.969585  [  704/ 2000]
loss: 1.880140  [ 1344/ 2000]
loss: 1.762660  [ 1984/ 2000]
********************
Epoch 2
loss: 1.771670  [   64/ 2000]
loss: 1.682876  [  704/ 2000]
loss: 1.676434  [ 1344/ 2000]
loss: 1.614405  [ 1984/ 2000]
********************
Epoch 3
loss: 1.557897  [   64/ 2000]
loss: 1.507629  [  704/ 2000]
loss: 1.516667  [ 1344/ 2000]
loss: 1.375341  [ 1984/ 2000]
********************
Epoch 4
loss: 1.422207  [   64/ 2000]
loss: 1.441802  [  704/ 2000]
loss: 1.264468  [ 1344/ 2000]
loss: 1.266092  [ 1984/ 2000]
********************
Epoch 5
loss: 1.170196  [   64/ 2000]
loss: 1.253586  [  704/ 2000]
loss: 1.200905  [ 1344/ 2000]
loss: 1.238269  [ 1984/ 2000]
********************
Epoch 6
loss: 1.214502  [   64/ 2000]
loss: 1.050314  [  704/ 2000]
loss: 1.002373  [ 1344/ 2000]
loss: 0.998873  [ 1984/ 2000]
********************
Epoch 7
loss: 1.212255  [   64/ 2000]
loss: 1.069430  [  704/ 2000]
loss: 0.912991  [ 1344/ 2000]
loss: 1.024760  [ 1984/ 2000]
********************
Epoch 8
loss: 0.941414  [   64/ 2000]
loss: 0.913626  [  704/ 2000]
loss: 0.976745  [ 1344/ 2000]
loss: 0.936738  [ 1984/ 2000]
********************
Epoch 9
loss: 0.850828  [   64/ 2000]
loss: 0.938331  [  704/ 2000]
```

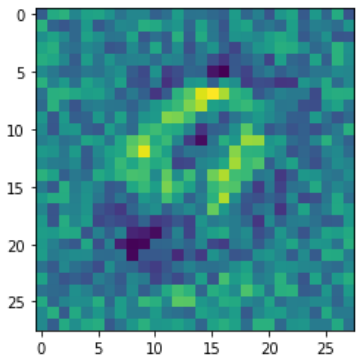
مقدار تابع ضرر پس از فرایند آموزش هم روی داده آزمون و هم داده ارزیابی کاهش یافت. همچنین دقت بر روی هر دو نوع داده افزایش یافت.



ب) وزن هایی که مدل آموزش دیده است را پلات می کنیم.

```
w, b = model.parameters()

print("weight shape:", w.shape)
print("bias shape:", b.shape)
for w_class in w:
    w_reshaped = torch.reshape(w_class, (img_tensor.shape[1], img_tensor.shape[2]))
    plt.imshow(w_reshaped.detach().numpy())
    plt.show()
```


0
5
10
15
20
25



مطابق تصاویر، وزن بیشتر ارقام شباهت زیادی به همان رقم دارد. وزن رقم ۴ و ۵ کمترین میزان شباهت را به رقم اصلی دارد که می تواند نشان از سخت تر بودن این ارقام باشد.

.4

الف)

4) الف) $\text{arrival_time} = \underbrace{\frac{dist}{speed}}_{x_0} w_0 + \underbrace{traffic}_{x_1} \cdot w_1 + \underbrace{weather}_{x_2} \cdot w_2 + b + \varepsilon$

توزیع نرمال $\Rightarrow P(y|x_0, x_1, x_2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-1}{2\sigma^2}(y - x_0 w_0 - x_1 w_1 - x_2 w_2 - b)^2\right)$

$$P(y|x) = \prod_i^n P(y^{(i)} | x_0^{(i)}, x_1^{(i)}, x_2^{(i)})$$

بیشینه کردن log :

$$\log P(y|x) = \sum \log P(y^{(i)} | x_0^{(i)}, x_1^{(i)}, x_2^{(i)})$$

$$= \sum_{i=1}^{N} \frac{-1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y - x_0 w_0 - x_1 w_1 - x_2 w_2 - b)^2$$

کمینه کردن منفی log :

$$-\log P(y|x) = \sum_{i=1}^{N} \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(y - x_0 w_0 - x_1 w_1 - x_2 w_2 - b)^2$$

با فرض ثابت بودن σ^2 :

$$\min \sum_{i=1}^{N} (y - x_0 w_0 - x_1 w_1 - x_2 w_2 - b)^2$$

b را به w و یک مقدار 1 به هر $x^{(i)}$ الحاق می‌کنیم

$$w^* = \arg\min \|y - Xw\|^2 \Rightarrow \nabla_w \|y - Xw\|^2 = 2X^T(y - Xw)$$

$$X^T X w^* = X^T y \Rightarrow w^* = (X^T X)^{-1} X^T y = X^+ y$$

ب) تابع آماده:

ستون های دیتافریم را مطابق فرمول به عنوان ویژگی های داده در نظر می گیریم و x, y را آماده می کنیم. سپس ۰.۲ از داده ها را به عنوان داده آزمون در نظر می گیریم. در اینجا از یک random state ثابت استفاده می کنیم تا مطمئن شویم در بخش های بعدی داده ها به همین صورت هستند. از LinearRegression موجود در کتابخانه scikit استفاده می کنیم و مدل را بر روی داده آموزش فیت می کنیم. سپس میزان MSE و MAE مدل را بر روی هر دو نوع داده محاسبه می کنیم.

```
# prepare data
x0 = torch.tensor(df['distance'] / df['speed'])
x1 = torch.tensor(df['traffic'])
x2 = torch.tensor(df['weather_conditions'])
X = torch.cat((torch.unsqueeze(x0, dim=1), torch.unsqueeze(x1, dim=1), torch.unsqueeze(x2, dim=1)), 1)
y = torch.tensor(df['Time_Arrival'])

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
print("train shape", X_train.shape, y_train.shape)
print("test shape", X_test.shape, y_test.shape)

# linear regression model
reg = LinearRegression().fit(X_train, y_train)
train_prediction = reg.predict(X_train)
train_mse = mean_squared_error(y_train, train_prediction)
train_mae = mean_absolute_error(y_train, train_prediction)
print("MSE train", train_mse)
print("MAE train", train_mae)

# predict on test data
test_prediction = reg.predict(X_test)
test_mse = mean_squared_error(y_test, test_prediction)
test_mae = mean_absolute_error(y_test, test_prediction)
print("MSE test", test_mse)
print("MAE test", test_mae)
```

```
train shape torch.Size([1600, 3]) torch.Size([1600])
test shape torch.Size([400, 3]) torch.Size([400])
MSE train 2.827727381547115e-30
MAE train 1.3616885397027546e-15
MSE test 2.8330891705122892e-30
MAE test 1.3786194408282882e-15
```

از پایه با تابع ضرر MSE:

مشابه قسمت قبل ابتدا داده را آماده می کنیم. سپس تابع ضرر را مربع میزان خطا تعریف می کنیم. در یک حلقه به تعداد ایپاک ها مدل را آموزش می دهیم: در ابتدا output را با وزن های کنونی محاسبه می کنیم. سپس از تابع ضرری که تعریف کرده ایم برای محاسبه میزان loss استفاده می کنیم. بعد در back propagation

گرادیان ها را محاسبه کرده و ۴ پارامتر شبکه را آپدیت می کنیم. در آخر گرادیان ها را برای ایپاک بعدی صفر می کنیم. پس از پایان فرایند آموزش مقدار loss را برای داده تست محاسبه می کنیم.

```python
# prepare data
x0 = torch.tensor(df['distance'] / df['speed'])
x1 = torch.tensor(df['traffic'])
x2 = torch.tensor(df['weather_conditions'])
X = torch.cat((torch.unsqueeze(x0, dim=1), torch.unsqueeze(x1, dim=1), torch.unsqueeze(x2, dim=1)), 1)
y = torch.tensor(df['Time_Arrival'])

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

def mse_loss(output, target):
    loss = torch.mean((output - target)**2)
    return loss


for e in range(epoch):
    output = X_train.T[0] * w0 + X_train.T[1] * w1 + X_train.T[2] * w2 + b
    # MSE loss function
    loss = mse_loss(output, y_train)
    if e % 100 == 0:
        print(f"Epoch {e}: train MSE loss {loss}")

    # calculate gradients
    loss.backward()
    # update parameters
    with torch.no_grad():
        w0 -= step_size * w0.grad
        w1 -= step_size * w1.grad
        w2 -= step_size * w2.grad
        b -= step_size * b.grad

    # zero the gradients
    w0.grad.zero_()
    w1.grad.zero_()
    w2.grad.zero_()
    b.grad.zero_()

test_prediction = X_test.T[0] * w0 + X_test.T[1] * w1 + X_test.T[2] * w2 + b
print("test MSE loss", mse_loss(test_prediction, y_test))
```

میزان خطا به شرح زیر است:

```
Epoch 0: train MSE loss 17122.408552746467
Epoch 100: train MSE loss 32.67808298479283
Epoch 200: train MSE loss 27.2307204665066
Epoch 300: train MSE loss 23.232975078686465
Epoch 400: train MSE loss 20.103561670961902
Epoch 500: train MSE loss 17.540723378720344
Epoch 600: train MSE loss 15.380171379624729
Epoch 700: train MSE loss 13.526364706908153
Epoch 800: train MSE loss 11.918950455492368
Epoch 900: train MSE loss 10.516482270275016
Epoch 1000: train MSE loss 9.288173153023381
Epoch 1100: train MSE loss 8.20981642801157
Epoch 1200: train MSE loss 7.26156505507573
Epoch 1300: train MSE loss 6.42672416945466
Epoch 1400: train MSE loss 5.691043296723006
Epoch 1500: train MSE loss 5.042224981038125
Epoch 1600: train MSE loss 4.469609484386546
Epoch 1700: train MSE loss 3.9639119418398674
Epoch 1800: train MSE loss 3.5170376448878113
Epoch 1900: train MSE loss 3.121912840665345
Epoch 2000: train MSE loss 2.772349370704901
Epoch 2100: train MSE loss 2.462922576001187
Epoch 2200: train MSE loss 2.188883256331293
Epoch 2300: train MSE loss 1.9460616846786372
Epoch 2400: train MSE loss 1.7307949749030798
Epoch 2500: train MSE loss 1.5398670800661471
Epoch 2600: train MSE loss 1.3704509489051362
Epoch 2700: train MSE loss 1.2200565469371665
Epoch 2800: train MSE loss 1.0864922343835877
Epoch 2900: train MSE loss 0.9678274499743925
Epoch 3000: train MSE loss 0.8623593072499852
Epoch 3100: train MSE loss 0.7685846268885657
Epoch 3200: train MSE loss 0.6851781728509962
Epoch 3300: train MSE loss 0.6109682229045658
Epoch 3400: train MSE loss 0.5449196767936839
test MSE loss tensor(0.5016, dtype=torch.float64
```

از پایه با تابع ضرر MAE:

تمامی مراحل مشابه قسمت قبل است با این تفاوت که تعریف تابع ضرر در این قسمت فرق کرده است. پس از پایان آموزش میزان خطا به شرح زیر است:

```
Epoch 0: train MAE loss 118.88770880901967
Epoch 100: train MAE loss 48.277304796914635
Epoch 200: train MAE loss 5.124812519464308
Epoch 300: train MAE loss 5.085509007187506
Epoch 400: train MAE loss 5.046660001991217
Epoch 500: train MAE loss 5.00829776107772
Epoch 600: train MAE loss 4.970594210321275
Epoch 700: train MAE loss 4.933638137807737
Epoch 800: train MAE loss 4.897210820971126
Epoch 900: train MAE loss 4.861437356872408
Epoch 1000: train MAE loss 4.826178363048609
Epoch 1100: train MAE loss 4.7915538308930685
Epoch 1200: train MAE loss 4.757496340540033
Epoch 1300: train MAE loss 4.724047644833389
Epoch 1400: train MAE loss 4.690739737955052
Epoch 1500: train MAE loss 4.658053458304512
Epoch 1600: train MAE loss 4.625683446305234
Epoch 1700: train MAE loss 4.594087546661146
Epoch 1800: train MAE loss 4.56303647805294
Epoch 1900: train MAE loss 4.5323453770740025
Epoch 2000: train MAE loss 4.502028043237629
Epoch 2100: train MAE loss 4.471983263135915
Epoch 2200: train MAE loss 4.442340375175237
Epoch 2300: train MAE loss 4.412972959628217
Epoch 2400: train MAE loss 4.3840512741625535
Epoch 2500: train MAE loss 4.355613398342025
Epoch 2600: train MAE loss 4.327584027179052
Epoch 2700: train MAE loss 4.299590320450364
Epoch 2800: train MAE loss 4.2717573855969135
Epoch 2900: train MAE loss 4.2442285844623555
Epoch 3000: train MAE loss 4.2170274237998475
Epoch 3100: train MAE loss 4.190354574000582
Epoch 3200: train MAE loss 4.163896581341054
Epoch 3300: train MAE loss 4.137517424438196
Epoch 3400: train MAE loss 4.1114178849552
test MAE loss tensor(4.0898, dtype=torch.float64
```

در هر ۳حالت مقدار خطا در طول فرایند آموزش در حال کاهش است. در مقایسه دو تابع ضرر، مقدار MSE کمتر از MAE است. زمانی که پیش بینی مدل اشتباه باشد، تابع MSE میزان جریمه بیشتری برای مدل در نظر می گیرد و بهتر از تابع MAE می تواند پارامترها را بهینه کند. همچنین در مقایسه تابع آماده با تابعی که از پایه پیاده سازی شده است، میزان ضرر تابع آماده بسیار کمتر است.

پ) در ۱۰۰ آزمایش مختلف داده حاضر کرده و مدل را آموزش می دهیم. برای این کار در یک حلقه با استفاده از random.sample، سیصد ایندکس رندوم انتخاب می کنیم و x,y را بدین صورت تشکیل می دهیم. سپس وزن ها را دوباره با همان مقادیر reinitialize می کنیم. در یک حلقه به تعداد ایپاک ابتدا مقدار forward را محاسبه کرده و سپس با محاسبه گرادیان، پارامترها را آپدیت می کنیم. در پایان آموزش مقدار هر یک از پارامترها را در

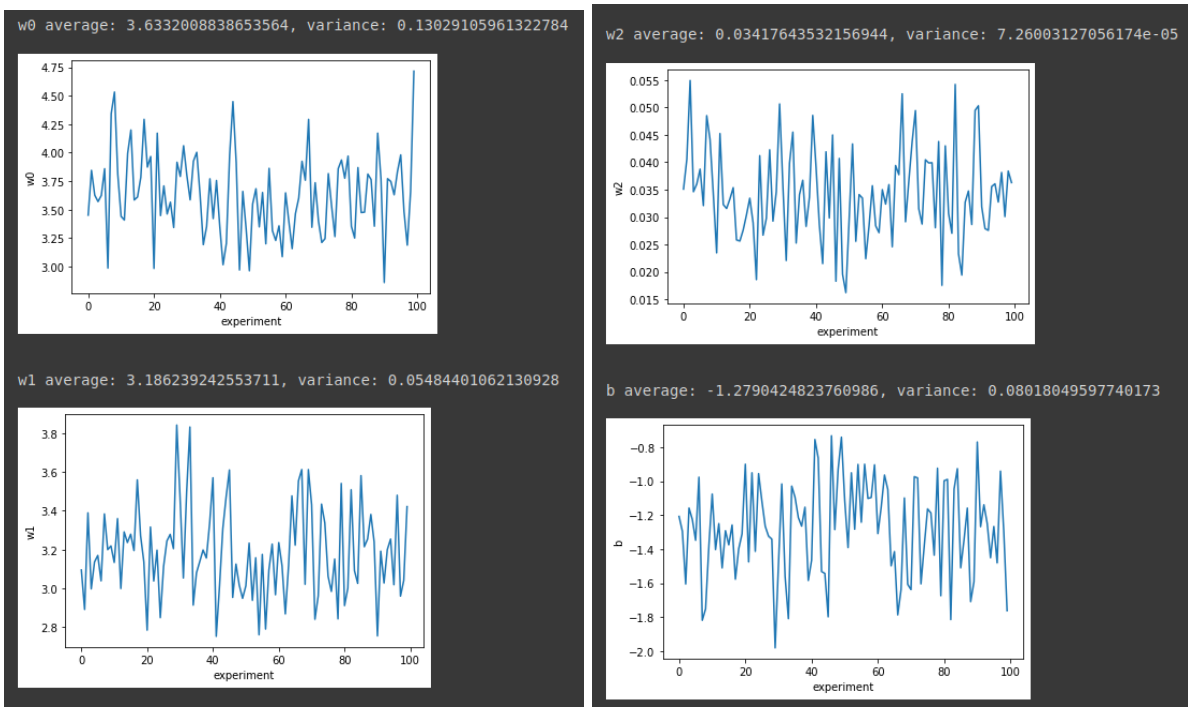
لیست مربوطه ذخیره می کنیم. (لیست ها: w0_experiments, b_experiments, w2_experiments, w1_experiments).

```python
# prepare data
x0 = torch.tensor(df['distance'] / df['speed'])
x1 = torch.tensor(df['traffic'])
x2 = torch.tensor(df['weather_conditions'])
X = torch.cat((torch.unsqueeze(x0, dim=1), torch.unsqueeze(x1, dim=1), torch.unsqueeze(x2, dim=1)), 1)
y = torch.tensor(df['Time_Arrival'])

indices = list(range(X.shape[0]))
w0_experiments = []
w1_experiments = []
w2_experiments = []
b_experiments = []

def mse_loss(output, target):
    loss = torch.mean((output - target)**2)
    return loss

for experiment in range(100):
    rnd_indices = random.sample(indices, 300)
    X_train = X[rnd_indices]
    y_train = y[rnd_indices]

    # reinitialize parameters
    w0 = torch.tensor(10.0, requires_grad=True)
    w1 = torch.tensor(15.0, requires_grad=True)
    w2 = torch.tensor(-10.0, requires_grad=True)
    b = torch.tensor(-20.0, requires_grad=True)

    for e in range(epoch):
        output = X_train.T[0] * w0 + X_train.T[1] * w1 + X_train.T[2] * w2 + b
        # MSE loss function
        loss = mse_loss(output, y_train)

        # calculate gradients
        loss.backward()
        # update parameters
        with torch.no_grad():
            w0 -= step_size * w0.grad
            w1 -= step_size * w1.grad
            w2 -= step_size * w2.grad
            b -= step_size * b.grad

        # zero the gradients
        w0.grad.zero_()
        w1.grad.zero_()
        w2.grad.zero_()
        b.grad.zero_()

    w0_experiments.append(w0)
    w1_experiments.append(w1)
    w2_experiments.append(w2)
    b_experiments.append(b)
```

در پایان مقادیر ذخیره شده را پلات می کنیم:

```
def plot_weights(weight, title):
    w = [x.detach().numpy() for x in weight]
    avg = np.mean(w)
    var = np.var(w)
    print(f'{title} average: {avg}, variance: {var}', '\n')

    plt.plot(w)
    plt.xlabel("experiment")
    plt.ylabel(title)
    plt.show()

    print('\n')

plot_weights(w0_experiments, "w0")
plot_weights(w1_experiments, "w1")
plot_weights(w2_experiments, "w2")
plot_weights(b_experiments, "b")
```

نمودار مربوط به هر یک از پارامترها به صورت زیر است:



نمودارها نوسانی هستند اما واریانس پارامترها بسیار کم است و میانگین هر یک از آنها به مقدار بیشتر داده ها در ۱۰۰ آزمایش نزدیک است. می توان نتیجه گرفت وزن ها وابستگی زیادی به داده ها ندارد.

.5

5) الف) $\frac{644}{4238} \simeq 0.15$

ب) $P(y|x) = \hat{y}^{y} \cdot (1-\hat{y})^{1-y} \xrightarrow[\hat{y}=p]{} P(y|x) = p^{y}(1-p)^{1-y}$

طبق MLE: $P(y|x) = \max \prod_{i=1}^{n} P(y^{(i)}|x^{(i)}) = \max \prod_{i}^{n} p^{y}(1-p)^{1-y}$

min کردن منفی log :

$$\Rightarrow -\log P(y|x) = -\sum_{i=1}^{n} (y_i \log p + (1-y_i)\log(1-p))$$

$$= -\log p \cdot \sum_{i=1}^{n} y_i - \log(1-p) \sum_{i=1}^{n}(1-y_i) = -n\bar{y}\log p - n(1-\bar{y})\log(1-p)$$

$$\xrightarrow{\text{مشتق } /\partial p} -n\bar{y} \cdot \frac{1}{p} - n(1-\bar{y})\frac{-1}{1-p} = 0 \Rightarrow n\left(\bar{y}/p - \frac{1-\bar{y}}{1-p}\right) = 0 \Rightarrow \frac{\bar{y}(1-p)-(1-\bar{y})p}{p(1-p)} = 0$$

$$\xrightarrow[p \neq 1]{p \neq 0 \wedge} \bar{y}(1-p) - (1-\bar{y})p = 0 \Rightarrow \bar{y} - p\bar{y} - p + \bar{y}p = 0 \Rightarrow \boxed{p = \bar{y}}$$

ت) در ابتدا فایل csv داده شده را می خوانیم و هد آن را چاپ می کنیم تا سطرهای داده را مشاهده کنیم. سپس با یک پیش پردازش کوچک، سطرهایی که مقادیر null دارند را از داده حذف می کنیم.

```python
df_disease = pd.read_csv('./heart_disease.csv')
df_disease.head()
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRa
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	8
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	9
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	7
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	6
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	8

```python
# prepocess data and delete null rows
new_df_disease = df_disease.dropna()
print("data rows:", df_disease.shape[0])
print("data rows after preprocess:", new_df_disease.shape[0])
```

```
data rows: 4238
data rows after preprocess: 3656
```

برای آماده سازی داده، کل دیتافریم به جز ستون TenYearCHD را به عنوان X و ستون TenYearCHD را به عنوان برچسب Y در نظر می گیریم و داده را به آموزش و آزمون تقسیم می کنیم. و در نهایت یک مدل Logistic Regression بر روی داده آموزشی train می کنیم.

```python
from sklearn.linear_model import LogisticRegression

X = new_df_disease.drop('TenYearCHD', axis=1)
y = new_df_disease['TenYearCHD']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
```

میزان دقت مدل بر روی داده آموزش و آزمون می سنجیم:

```python
train_acc = clf.score(X_train, y_train)
test_acc = clf.score(X_test, y_test)

print("train data accuracy", train_acc)
print("test data accuracy", test_acc)
```

```
train data accuracy 0.8512311901504788
test data accuracy 0.8387978142076503
```

همانگونه که مشاهده می شود میزان دقت مدل بر روی داده آموزش ۸۵ و داده آزمون ۸۳ درصد است که نشان از عملکرد خوب مدل و عدم overfit دارد.

تفاوت linear regression و logistic regression:

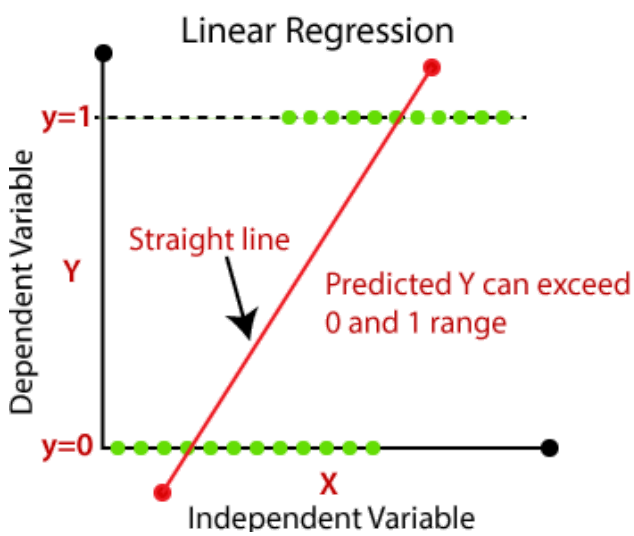
رگرسیون خطی برای مسائل رگرسیون استفاده می شود در حالی که برای logistic regression در مسائل طبقه بندی استفاده می شود.

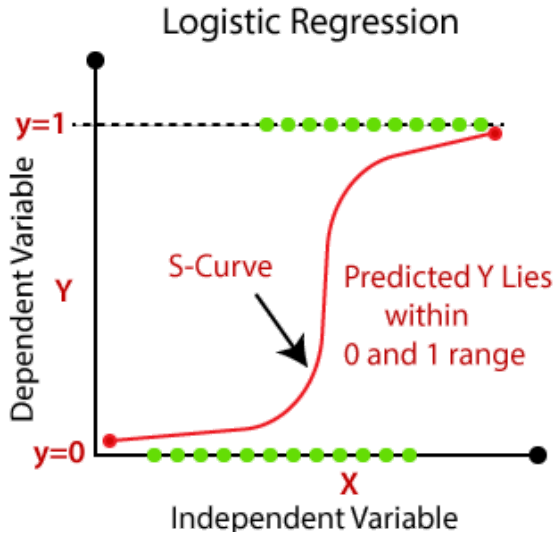
رگرسیون خطی یک خروجی پیوسته ارائه می دهد اما خروجی logistic regression گسسته است.

هدف از رگرسیون خطی یافتن بهترین خطی است که بتواند بر داده fit شود در حالی که logistic regression بدنبال یافتن بهترین منحنی است که بتواند داده را آموزش ببیند. در تصویر زیر به خوبی فرق بین خط و منحنی پیش بینی شده توسط هر دو روش مشاهده می شود.

در رگرسیون خطی از تابع ضرر MSE استفاده می شود در حالی که برای logistic regression از maximum likelihood estimation استفاده می شود.

رگرسیون خطی برای پیش‌بینی متغیر وابسته پیوسته با استفاده از مجموعه معینی از متغیرهای مستقل استفاده می‌شود. خروجی‌های تولید شده باید یک مقدار پیوسته مانند قیمت و سن باشند. اما از logistic regression برای پیش‌بینی متغیر وابسته categorical با استفاده از مجموعه معینی از متغیرهای مستقل استفاده می‌شود و خروجی‌های تولید شده باید مقادیر categorical مانند 0 یا 1، بله یا خیر باشد.





منابع

https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/