به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	فاطمه جلیلی – سالار صفردوست
شماره دانشجویی	810199450 -810199398
تاریخ ارسال گزارش	1402/8/17

فهرست

شکل‌ها

پاسخ ۱- شبکه عصبی Mcculloch-Pitts

1-1. نمایشگر 7-segment

2-1. شبکه عصبی یک لایه

با توجه به اینکه ورودی یک بردار 1×7 است و خروجی برداری 1×4 است W باید ماتریسی به ابعاد 7×4 باشد .

برای پیاده سازی الگوریتم وزن های مربوط به LED های روشن را 1 و وزن های مربوط به LED های خاموش را 1- تعیین می کنیم برای مثال برای وردی اول (6) که تنها LED B خاموش است ردیف اول ماتریس W همگی 1 و عنصر دوم آن -1 مطابق شکل زیر تعیین می شوند همچنین به طور مشابه برای دیگر ورودی ها وزن ها را تعیین می کنیم.

برای تعیین threshold که ابعادی یکسان با ابعاد خروجی دارد زمانی که ورودی 6 باشد LED 6 روشن است پس عنصر اول threshold را برابر 5 تعیین می کنیم یعنی تعداد LED های روشن باید بزرگ تر از 5 باشد تا نورون خروجی اول 1 شود. به طور مشابه عنصر های دیگر threshold را به تعداد LED های روشن در هر ورودی منهای یک تعیین می کنیم:

```
x = np.zeros([7, 1])
W = np.zeros([4, 7])
thr = np.zeros([4, 1])
W[0, :] = [1, -1, 1, 1, 1, 1, 1]
W[1, :] = [1, 1, 1, -1, -1, -1, -1]
W[2, :] = [1, 1, 1, 1, 1, 1, 1]
W[3, :] = [1, 1, 1, 1, -1, 1, 1]
thr[:, 0] = [5, 2, 6, 5]
```

شکل 1. وزن ها و آستانه شبکه 1 لایه

برای تست عملکرد شبکه مطابق کد نوشته شده هر 4 حالت ورودی به همراه دو حالت خارج از ورودی های معرفی شده را تست می کنیم :

```
6 classified as :
[[1]
 [0]
 [0]
 [0]]

9 classified as :
[[0]
 [0]
 [0]
 [1]]

7 classified as :
[[0]
 [1]
 [0]
 [0]]

Other classified as :
[[0]
 [0]
 [0]
 [0]]

8 classified as :
[[0]
 [0]
 [1]
 [0]]

Other classified as :
[[0]
 [0]
 [0]
 [0]]
```

شکل 2. خروجی تست شبکه 1 لایه

همانطور که بدست آمده برای ورودی 6 نورون خروجی اول ، برای ورودی 7 نورون خروجی دوم ، برای ورودی 8 نورون خروجی سوم و برای ورودی 9 نورون خروجی چهارم 1 می شود و بقیه نورون ها صفر هستند ، در حالتی غیر از این چهار حالت همه ی نورون های خروجی 0 می شوند.

3-1. شبکه عصبی دولایه

الف)

با توجه به اینکه ورودی یک بردار 1×7 است و لایه پنهان دو نورون دارد ماتریس وزن بین لایه ورودی و لایه پنهان (W1) ماتریسی با ابعاد 7×2 است همچنین با توجه به این که لایه خروجی 4 نورون دارد ماتریس وزن بین لایه نهان و خروجی (W2) ماتریسی با ابعاد 2×4 است.

Threshold هر لایه ابعادی مطابق خروجی همان لایه دارد یعنی threshold لایه نهان 1×2 و threshold لایه خروجی مانند حالت قبل 1×4 است.

برای پیاده سازی الگوریتم وزن ها را طوری تعیین می کنیم که لایه نهان که دارای دو نورون است ، نورن اول روشن بودن همزمان LED های f, b (f & b) و نورون دوم لایه نهان روشن بودن همزمان LED های e, c (e & c) را تشخیص دهد چرا که مطابق ورودی های داده شده با داشتن از دو ویژگی از ورودی ها می توان آن ها را در لایه آخر از یکدیگر تفکیک کرد.

برای این کار وزن های مربوط به f, b را 1 و بقیه را 0 در ردیف اول W1 و وزن های مربوط به e, c را 1 و بقیه را 0 در ردیف دوم W1 تعیین می کنیم ، از آنجایی که می خواهیم روشن بودن همزمان این دو LED را بررسی کنیم باید هر دو عنصر threshold لایه نهان 1 تعیین شوند تا در صورتی که هر دو LED مربوطه روشن بودند یعنی خروجی قبل اعمال threshold 2 بود نورون های لایه نهان 1 شوند.

برای تعیین W2 روشن بودن همزمان LED های ذکر شده را در هر ورودی بررسی می کنیم اگر هر دو روشن بودند وزن مربوطه را 1 و در غیر اینصورت وزن مربوطه را 1- تعیین می کنیم، برای مثال برای ورودی 6 که f, b همزمان روشن نیستند عنصر اول ردیف اول W2 1- تعیین می شود و آز انجایی که e, c همزمان روشن هستند عنصر دوم ردیف اول W2 1 تعیین می شود ، همچنین به طور مشابه برای دیگر ورودی ها پیش می رویم.

برای تعیین threshold لایه خروجی هر ورودی را با توجه آنچه تا لایه خروجی بدست آمده مجزا بررسی می کنیم و threshold را تعیین می کنیم، برای مثال برای ورودی 6 چون لایه نهان خروجی $\binom{0}{1}$ دارد (یعنی f, b همزان روشن نیستند ولی e, c همزمان روشن هستند) و ردیف اول W2 به صورت $[-1, 1]$ تعیین شده است ، پس خروجی لایه آخر تا قبل از اعمال threshold ، 1 است پس عنصر اول threshold را 0 تعیین می کنیم ، یا برای مثال برای ورودی 7 چون لایه نهان خروجی $\binom{0}{0}$ دارد (یعنی f, b همزمان روشن نیستند و e, c هم همزمان روشن نیستند) و ردیف دوم W2 به صورت $[-1, -1]$ تعیین شده است ، پس خروجی لایه آخر تا قبل از اعمال threshold ، 0 است پس عنصر دوم threshold را 1- تعیین می کنیم به طور مشابه برای دیگر ورودی ها پیش می رویم.

```
#initializations
input = np.ones((7,1))
w1 = np.zeros((2, 7))
w2 = np.zeros((4, 2))
theta1 = np.zeros((2, 1))
theta2 = np.zeros((4, 1))

#Weighting
w1[0, :] = np.array((0, 1, 0, 0, 0, 1, 0))
w1[1, :] = np.array((0, 0, 1, 0, 1, 0, 0))

theta1 = np.matrix([[1], [1]])

w2 = np.matrix([[-1, 1], [-1, -1], [1, 1], [1, -1]])

theta2 = np.matrix([[0, -1, 1, 0]]).T
```

شکل 3. وزن ها و آستانه های شبکه 2 لایه

برای تست عملکرد شبکه مطابق کد نوشته شده هر 4 حالت ورودی را تست می کنیم :

```
out put when input = 6 :
 [[1]
 [0]
 [0]
 [0]]
out put when input = 7 :
 [[0]
 [1]
 [0]
 [0]]
out put when input = 8 :
 [[0]
 [0]
 [1]
 [0]]
out put when input = 9 :
 [[0]
 [0]
 [0]
 [1]]
```

شکل 4. خروجی تست شبکه 2 لایه

همانطور که بدست آمده برای ورودی 6 نورون خروجی اول ، برای ورودی 7 نورون خروجی دوم ، برای ورودی 8 نورون خروجی سوم و برای ورودی 9 نورون خروجی چهارم 1 می شود و بقیه نورون ها صفر هستند .

ب)

نورن اول روشن بودن همزمان LED های f, b (f & b) و نورون دوم لایه نهان روشن بودن همزمان LED های e, c (e & c) را تشخیص می دهد.

ج)

مطابق توضیحات قسمت الف از آنجایی که در شبکه 1 لایه W ماتریسی با ابعاد 7×4 است و threshold برداری با ابعاد 1×4 است پس در کل شبکه 1 لایه 32 پارامتر دارد.

در شبکه دو لایه W1 و W2 ماتریس هایی با ابعاد 7×2 و 2×4 هستند و threshold لایه نهان و لایه خروجی به ترتیب بردارهایی با ابعاد 1×2 و 1×4 هستند پس در کل شبکه 2 لایه 28 پارمتر دارد.

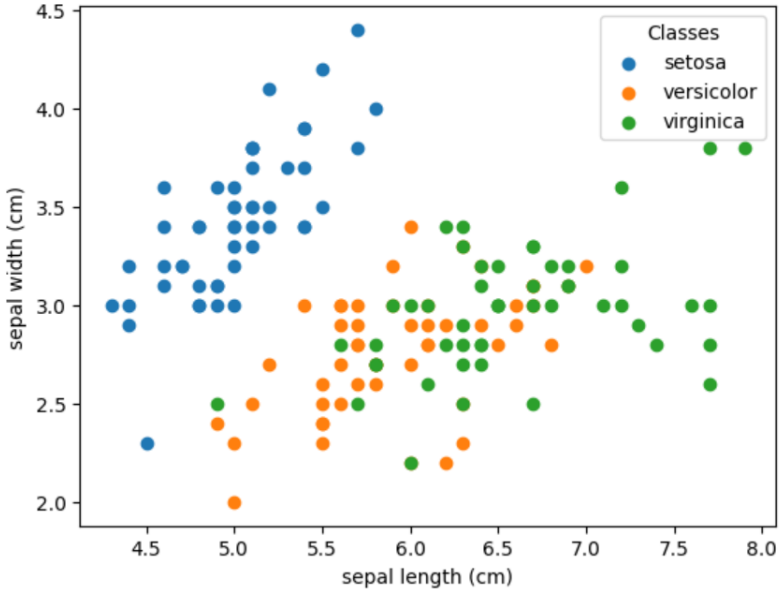
لذا تعداد پارامتر های شبکه دولایه از تعداد پارامتر های شبکه 1 لایه کم تر است.

.

پاسخ ۲ – آموزش شبکه های Adaline و Madaline

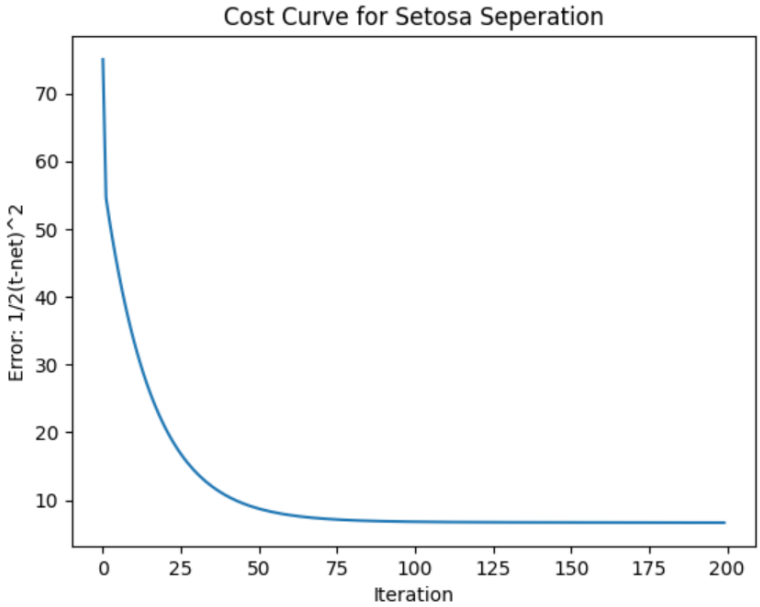
Adaline .1-2

الف)



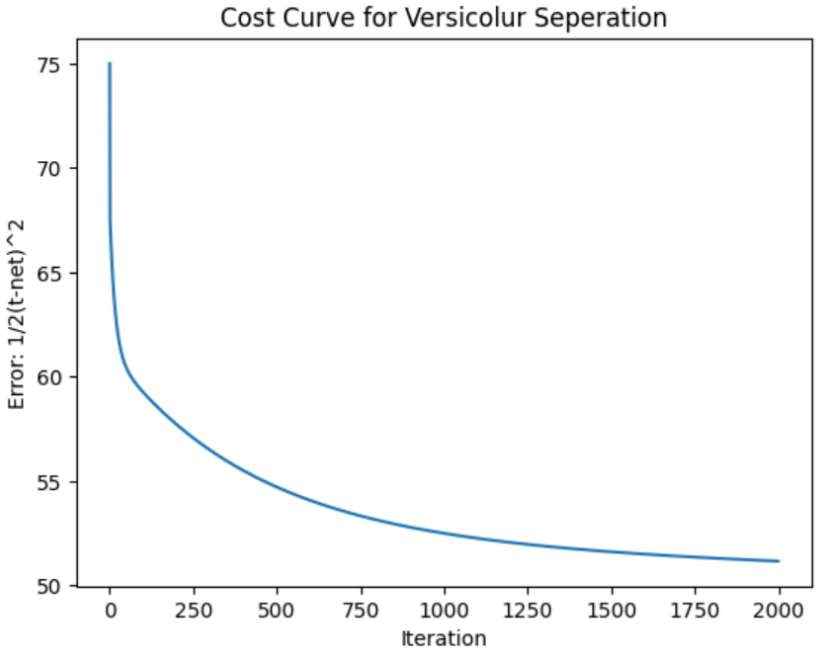
شکل 5. نمودار پراکندگی داده های iris

همانطور که دیده می شود داده Setosa از دو داده دیگر جدا پذیر تر است ولی دو داده دیگر در هم رفتگی بیش تری دارند.



شکل 6. نمودار تغییرات خطا Adaline برای جدا کردن داده Setosa

ب)



شکل 7. نمودار تغییرات خطا Adaline برای جدا کردن داده Versicolour

همانطور که دیده می شود با learning rate یکسان برای دو حالت جدا سازی، جدا سازی داده Versicolour ، iteration بسیار بیش تری لازم دارد تا خطا تقریبا ثابت شود و همچنین مقدار خطا حدود 52 و بسیار بیش تر از خطا در جدا سازی Setosa که در حدود 5 است می باشد چرا که همانطور که در قسمت الف در نمودار پراکندگی داده ها نشان داده شده است داده Setosa از دو داده دیگر جداپذیر تر است ولی دو داده دیگر در هم رفتگی بیش تری دارند لذا جدا سازی آن ها سخت تر خواهد بود.

Modaline .2-2

الف)

در این روش، یک لایه نهان با دو نورون و یک نورون در لایه خروجی وجود دارد.

نحوه‌ی به دست آوردن وزن‌های نورون‌ها به شکل زیر خواهد بود:

1- پارامترهای نورون خروجی با اعداد کوچکی مقداردهی اولیه می‌شوند، همچنین یک Learning rate نسبتا کوچک نیز انتخاب می‌شود. تابع فعالساز هر سه نورون به شکل زیر می‌باشد:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0. \end{cases}$$

2- مراحل زیر تا تحقق شرط اتمام تکرار می‌شوند:

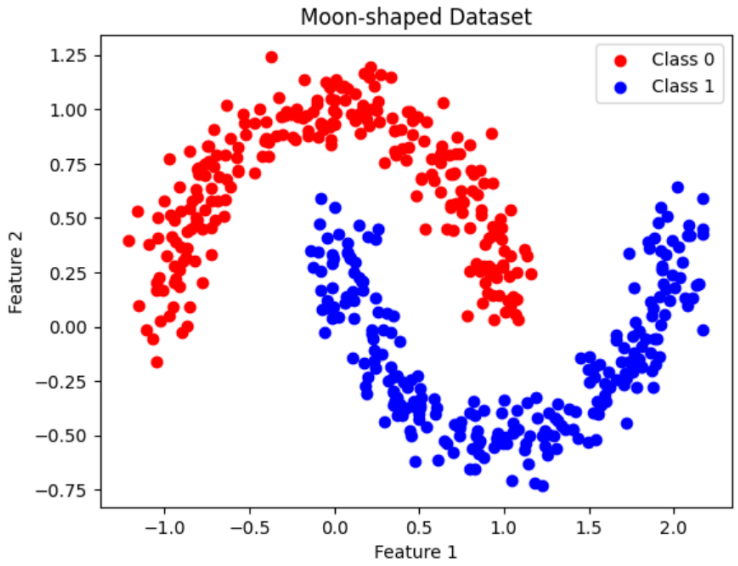
- برای هر عضو داخل داده‌ی آموزش مقدار خروجی شبکه را به دست می‌آوریم،
- گر مقدار خروجی با مقدار واقعی آن یکسان بود، به پارامترهای مسئله دست نمی‌زنیم.
- اگر مقدار خروجی 1 و با مقدار واقعی برابر نبود، پارامترهای نورونی از لایه پنهان که خروجی netاش به صفر نزدیک‌تر بود را به شکل زیر آپدیت می‌کنیم:

$$b_J(\text{new}) = b_J(\text{old}) + \alpha(1 - z\_in_J),$$
$$w_{iJ}(\text{new}) = w_{iJ}(\text{old}) + \alpha(1 - z\_in_J)x_i;$$

- اگر مقدار خروجی 0 و با مقدار واقعی آن یکسان نبود، پارامترهای نورون‌هایی از لایه پنهان که net مثبت دارند را به شکل زیر آپدیت می‌کنیم:

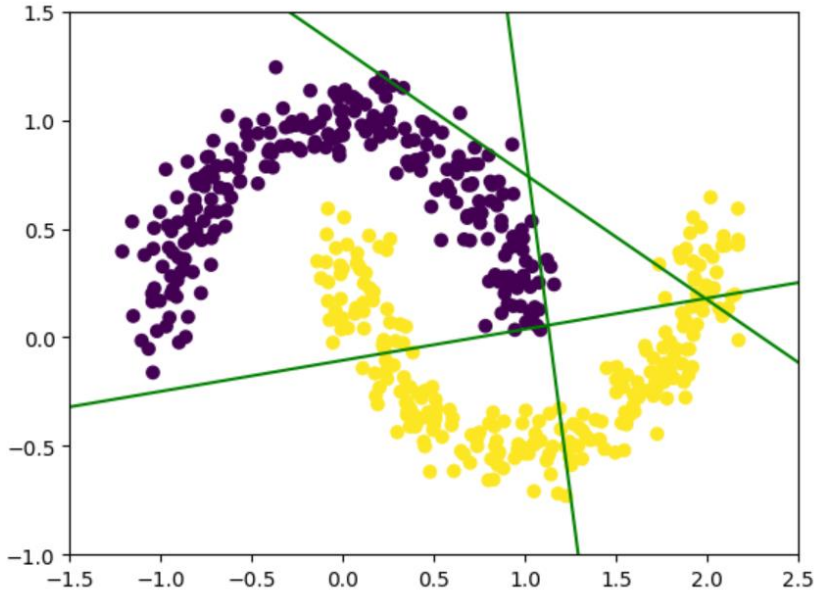
$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k),$$
$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i$$

ب، ج)

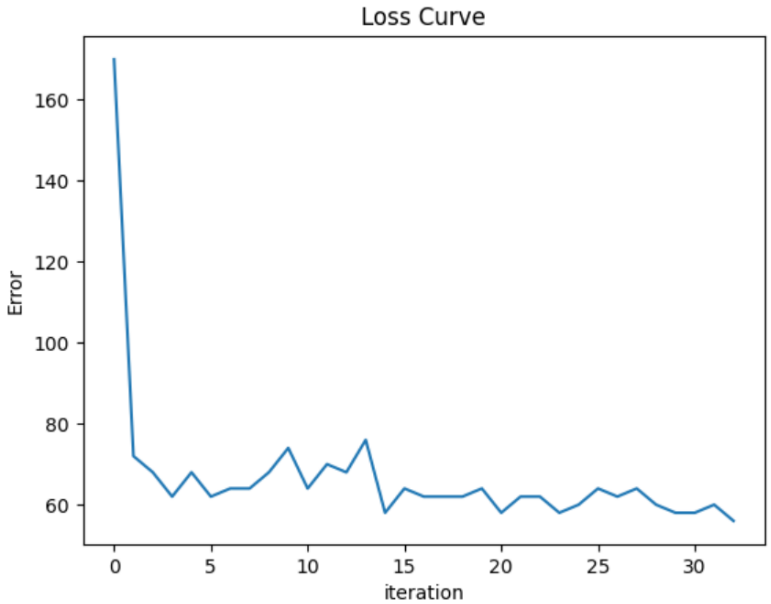


شکل 8. نمودار پراکندگی داده های Moon-shaped

```
Accuracy of prediction in test set with 3 hidden neorons is: 0.884
```

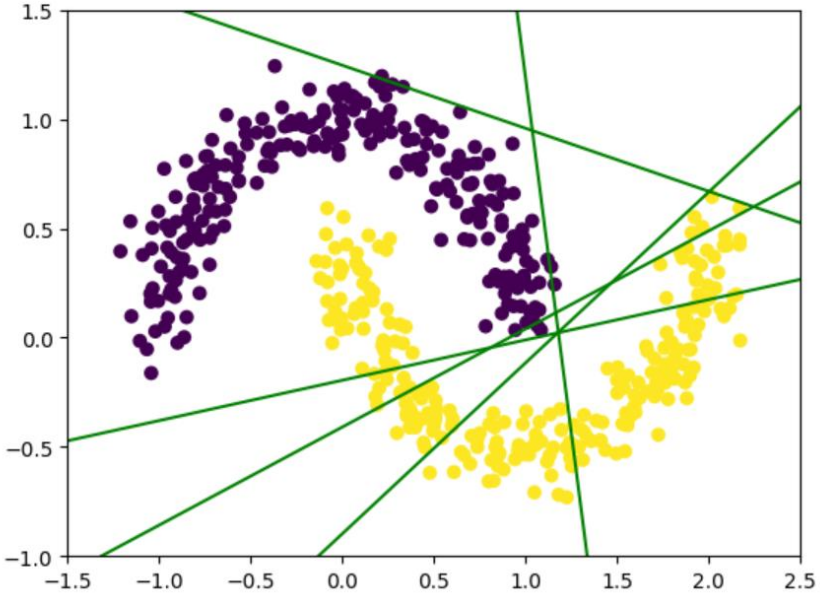


شکل 9. تفکیک نقاط با 3 نورون در لایه پنهان

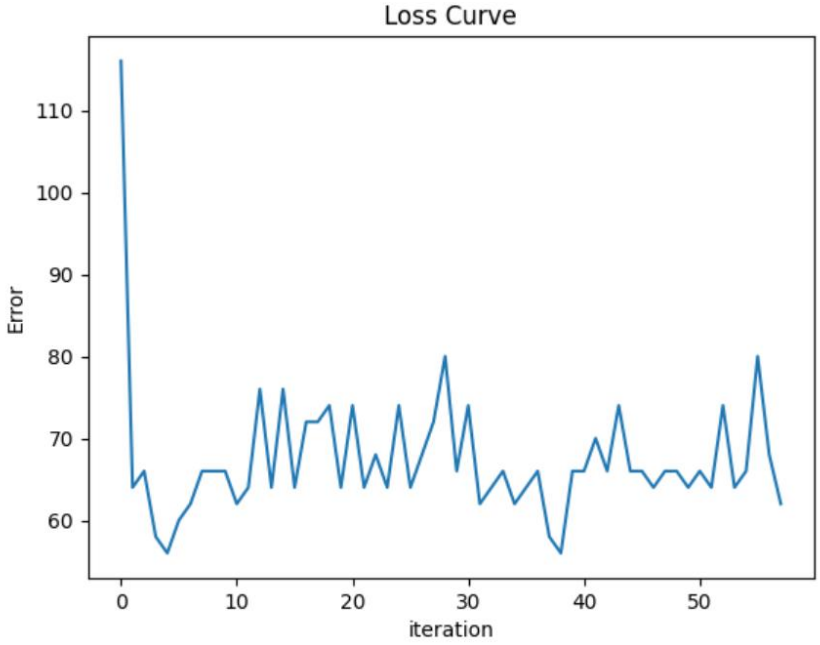


شکل 10. نمودار Loss با 3 نورون در لایه پنهان

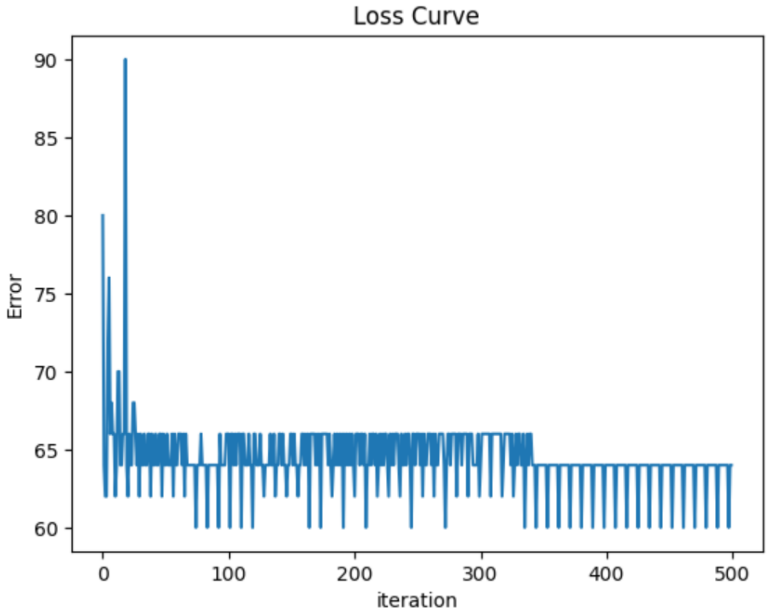
```
Accuracy of prediction in test set with 5 hidden neorons is: 0.888
```



شکل 11. تفکیک نقاط با 5 نورون در لایه پنهان

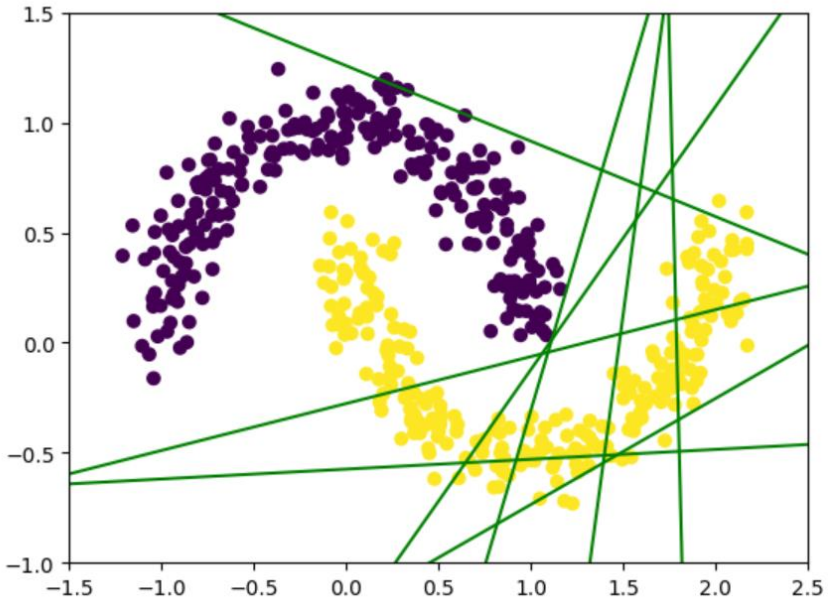


شکل 12. نمودار Loss با 5 نورون در لایه پنهان



شکل 13. تفکیک نقاط با 8 نورون در لایه پنهان

```
Accuracy of prediction in test set with 8 hidden neorons is: 0.884
```



شکل 14. نمودار Loss با 8 نورون در لایه پنهان

همانطور که در شکل های فوق نشان داده شده است دقت جدا سازی با 3 نورون و 8 نورون در لایه پنهان 88.4 ٪ و دقت جداسازی با 5 نورون در لایه پنهان 88.8٪ است.

مطابق شرط توقف تعریف شده اگر برای 3 iteration متوالی آپدیت وزن ها کم تر از حدی باشه روند متوقف می شود و یا اگر این شرط ارضا نشد با رسیدن به پایان ماکسیمم iteration تعریف شده روند متوقف می شود ، در نمودار های فوق می بینیم که در حالت 3 نورن و 5 نورون شرط اول در حدود 32 ایپاک و 59 ایپاک برقرار شده است ولی برای حالت 8 نورون تا انتهای 500 ایپاک تعریف شده پیش رفته است ، دلیل این امر این است که با زیاد تر شدن نورون ها یا خطوطی که می خواهیم توسط آن ها دیتا ها را تفکیک کنیم به دلیل شکل پراکندگی دیتا که درون خطوط صاف یا چند ضلعی به سختی جای می گیرند نیاز به ایپاک بیش تر داریم.

در نهایت مقدار خطا در حالت 3 نورن از دیگر حالت ها کم تر است و حدود 58 قرار می گیرد در حالتی که برای دو حالت دیگر بین 60 تا 65 قرار می گیرد.

پاسخ ۳ – خوشه بندی با استفاده از Autoencoder

1-3. پیاده سازی Deep Autoencoder برای کاهش ابعاد داده ها

1-1-3. برای پیاده‌سازی شبکه‌ی مشابه با شبکه‌ی ارائه شده در مقاله از کتابخانه‌ی PyTorch کمک گرفته شد و کلاس‌های Encoder و Decoder به صورت جداگانه ایجاد شد.

```python
class Encoder(nn.Module):
    def __init__(self, n_features):
        super(Encoder, self).__init__()
        self.linear1 = nn.Linear(n_features, 512)
        self.relu1 = nn.ReLU()
        self.tanh1 = nn.Tanh()
        self.linear2 = nn.Linear(512, 128)
        self.relu2 = nn.ReLU()
        self.tanh2 = nn.Tanh()
        self.linear3 = nn.Linear(128, 32)
        self.tanh3 = nn.Tanh()
        self.linear4 = nn.Linear(32, 10)

    def forward(self, x):
        x = self.tanh1(self.relu1(self.linear1(x)))
        x = self.tanh2(self.relu2(self.linear2(x)))
        x = self.tanh3(self.linear3(x))
        x = self.linear4(x)
        return x

class Decoder(nn.Module):
    def __init__(self, n_features):
        super(Decoder, self).__init__()
        self.linear1 = nn.Linear(10, 32)
        self.tanh1 = nn.Tanh()
        self.linear2 = nn.Linear(32, 128)
        self.tanh2 = nn.Tanh()
        self.linear3 = nn.Linear(128, 512)
        self.tanh3 = nn.Tanh()
        self.linear4 = nn.Linear(512, n_features)
        self.sigmoid4 = nn.Sigmoid()

    def forward(self, x):
        x = self.tanh1(self.linear1(x))
        x = self.tanh2(self.linear2(x))
        x = self.tanh3(self.linear3(x))
        x = self.sigmoid4(self.linear4(x))
        return x
```

✓ 0.0s Python

شکل 15: تعریف شبکه‌های encoder و decoder

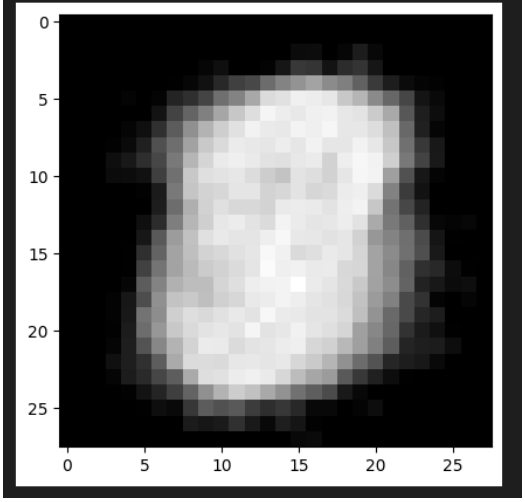
2-1-3. برای آموزش شبکه Autoencoder، در ابتدای امر نیاز به تعریف تابع هزینه داریم؛ با توجه به آنچه در مقاله ارائه شده بود، فرمول این تابع هزینه شامل پارامترهای ω_i بوده که نشان دهنده اهمیت هر پیکسل می‌باشند. برای محاسبه‌ی این مقادیر از بخشی از مجموعه ترین استفاده شد.

```
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)


T_subset = 100
indexes = np.random.randint(low = 0, high = T_train, size = T_subset)
subset = x_train[indexes, :]

n_features = subset.shape[1]
w = np.zeros(n_features)
s_equal = 0.0
s_nequal = 0.0
equal = 0
nequal = 0
for f in range(n_features):
    for i in range(T_subset):
        for j in range(i+1, T_subset):
            if(y_train[i] == y_train[j]):
                s_equal = s_equal + np.exp(-((subset[i, f]-subset[j, f])**2))
                equal = equal+1
            else:
                s_nequal = s_nequal + (1 - np.exp(-((subset[i, f]-subset[j, f])**2)))
                nequal = nequal+1
    w[f] = (s_equal/equal)*(s_nequal/nequal)
    s_equal = 0
    s_nequal = 0
    equal = 0
    nequal = 0
✓ 12.3s
```

شکل 16: به دست آوردن اهمیت هر پیکسل

نتایج حاصل از اجرای قطعه کد بالا به شکل تصویر زیر قابل نمایش است:

```
plt.imshow(w.reshape(28,28), cmap='gray')
w = torch.from_numpy(w)
✓ 0.1s
```



شکل 17: تصویر مربوط به اهمیت هر پیکسل

حال در مرحله‌ی بعد با داشتن ضرایب مورد نیاز، تابع هزینه را به شکل زیر تعریف می‌کنیم:

```
def loss_function(outputs, targets, model1, model2, w, beta):
    L_cmse = torch.mean(torch.mean(w*((outputs - targets)**2), dim=1))
    L2 = 0
    for param in model1.parameters():
        L2 += torch.sum(param**2)
    for param in model2.parameters():
        L2 += torch.sum(param**2)
    Lr = beta * L2
    return L_cmse + Lr
```

✓ 0.0s

شکل 18: تعریف تابع هزینه

برای آموزش شبکه از دیتاست MNIST استفاده می‌کنیم و دیتای ترین و تست را نرمالیزه می‌کنیم:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

[T_train, row, column] = x_train.shape
T_test = x_test.shape[0]

print(f"Train features shape: {x_train.shape}\nTrain label shape: {y_train.shape}")
print(f"Test features shape: {x_test.shape}\nTest labels shape: {y_test.shape}")
```

✓ 0.3s

```
Train features shape: (60000, 28, 28)
Train label shape: (60000,)
Test features shape: (10000, 28, 28)
Test labels shape: (10000,)
```

```
x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)

min_val = np.min(x_train)
max_val = np.max(x_train)
x_train = (x_train - min_val) / (max_val - min_val)
x_test = (x_test - min_val) / (max_val - min_val)
```

✓ 0.7s

شکل 19: نرمالایز کردن دیتاست

در انتها، در 40 epoch، شبکه را با batch sizeهایی که در هر iteration افزایش می‌یابند، آموزش می‌دهیم:

```python
encoder = Encoder(n_features)
decoder = Decoder(n_features)

beta = 1e-9
learning_rate = 0.008
epochs = 40
batch_size = 50

optimizer = torch.optim.Adam(list(encoder.parameters()) + list(decoder.parameters()), lr=learning_rate)

loss_list = []
for epoch in range(epochs):
    batch_size = (int(epoch/2)+1)*50
    data_loader = data.DataLoader(dataset=x_train, batch_size=batch_size, shuffle=True)
    for i, x in enumerate(data_loader):
        out_encoder = encoder(x)
        y = decoder(out_encoder)
        loss = loss_function(y, x, encoder, decoder, w, beta)
        loss_list.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if(i%34 == 0):
            print(f'Epoch {epoch+1}/{epochs} : {i+1}/{int(T_train/batch_size)}', end='\r')
    print(f"Epoch {epoch+1}/{epochs} : cost = {sum(loss_list[-batch_size:])/len(loss_list[-batch_size:])}")
```

✓ 8m 28.2s

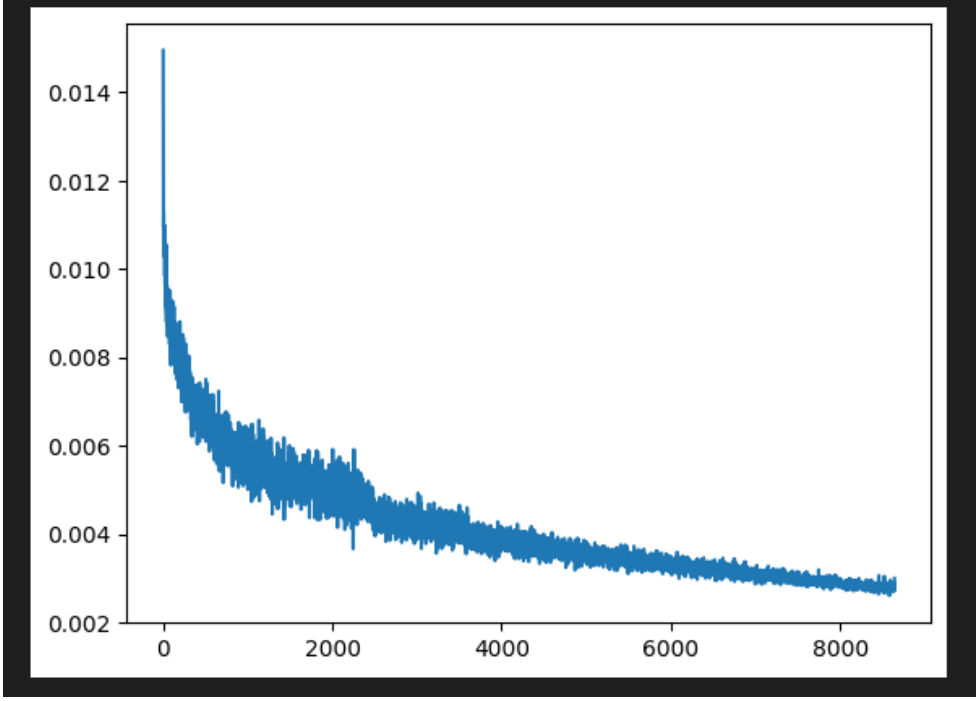
شکل 20: آموزش شبکه Autoencoder

می‌توان نتیجه را با نمایش مقادیر تابع لاس در هر تکرار و همچنین استفاده از imshow، خروجی اتوانکودرمشاهده کرد:

```python
plt.plot(loss_list)
```

✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x1c7a37925e0>]
```



شکل 21: نمودار تابع هزینه در هر تکرار

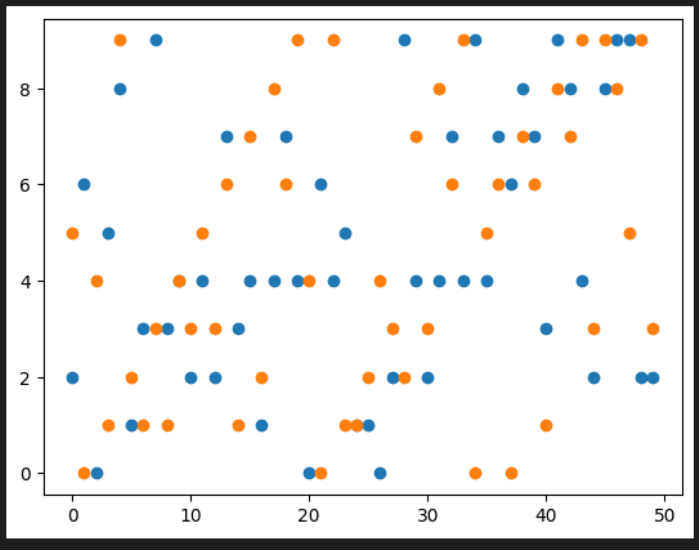
3-1-3 . حال می‌توانیم انکودر به دست آمده از مرحله‌ی قبل را از شبکه اتوانکودر جدا کنیم و خروجی آن را برای کلاستر کردن عکس‌ها استفاده کنیم. همچنین برای مقایسه‌ی نتیجه، یک بار کلاستر کردن را بر روی داده‌های خام انجام می‌دهیم.

```
from sklearn.cluster import KMeans

# using kmeans on the raw x_train
kmeans_ordinary = KMeans(n_clusters=10)
kmeans_ordinary.fit(x_train.detach().numpy())

predicted_ordinary_train = kmeans_ordinary.predict(x_train.detach().numpy())
predicted_ordinary_test = kmeans_ordinary.predict(x_test.detach().numpy())

#using kmeans on the encoded x_train
x_train_encoded = encoder(x_train).detach().numpy()

x_test_encoded = encoder(x_test).detach().numpy()

kmeans_encoded = KMeans(n_clusters=10)
kmeans_encoded.fit(x_train_encoded)

predicted_encoded_train = kmeans_encoded.predict(x_train_encoded)
predicted_encoded_test = kmeans_encoded.predict(x_test_encoded)
✓ 1m 8.3s
```

شکل 22: خوشه‌بندی کردن دیتای ترین خام و دیتای ترین انکودر

نمودار زیر نتیجه لیبل‌های داده شده به قسمتی از داده‌ی ترین در اثر کلاسترینگ می‌باشد و همزمان با آن لیبل‌های واقعی اعداد نیز نمایش داده شدند.

```
plt.scatter(np.array(range(len(predicted_encoded_train[0:50]))), predicted_encoded_train[0:50])
plt.scatter(np.array(range(len(y_train[0:50]))), y_train[0:50])
✓ 0.2s
<matplotlib.collections.PathCollection at 0x1c7b57a1130>
```



شکل 23: نمایش لیبل واقعی هر عضو دیتای ترین(آبی) به همراه لیبل به دست آماده از خوشه‌بندی(نارنجی)

دیده می‌شود بخش خوبی از اعداد واقع در یک لیبل (مثلا 6) در خروجی کلاستر شده لیبل عدد مشابه به هم را گرفته‌اند. (مثلا 5)

5-1-3. برای بررسی نهایی عملکرد، از معیار مشابه موجود در مقاله به نام ARI استفاده می‌کنیم:

```
from sklearn.metrics import adjusted_rand_score

ari_ordinary_train = adjusted_rand_score(y_train, predicted_ordinary_train)
ari_ordinary_test = adjusted_rand_score(y_test, predicted_ordinary_test)
ari_encoded_train = adjusted_rand_score(y_train, predicted_encoded_train)
ari_encoded_test = adjusted_rand_score(y_test, predicted_encoded_test)

print(f"ARI for ordinary train: {ari_ordinary_train}")
print(f"ARI for ordinary test: {ari_ordinary_test}")
print(f"ARI for encoded train: {ari_encoded_train}")
print(f"ARI for encoded test: {ari_encoded_test}")
```

✓ 0.0s

```
ARI for ordinary train: 0.36065812062604624
ARI for ordinary test: 0.36626426783515814
ARI for encoded train: 0.4623127568307444
ARI for encoded test: 0.4677684761795095
```

شکل 24: به دست آوردن مقدار ARI

پاسخ ۴ - شبکه های Multi-Layer Perceptron

1-4. آشنایی و کار با مجموعه دادگان (پیش پردازش)

الف) در شکل زیر، بعد اول هر array برابر تعداد سمپل و بعدهای دیگر موجود در x_train و x_test نشان‌دهنده‌ی سایز هر کدام از سمپل‌ها می‌باشد.

```
import torch
import numpy as np
from keras.datasets import mnist
import matplotlib.pyplot as plt
import torch.nn as nn

(x_train, y_train), (x_test, y_test) = mnist.load_data()

[T_train, row, column] = x_train.shape
T_test = x_test.shape[0]

print(f"Train features shape: {x_train.shape}\nTrain label shape: {y_train.shape}")
print(f"Test features shape: {x_test.shape}\nTest labels shape: {y_test.shape}")
```

✓ 0.5s

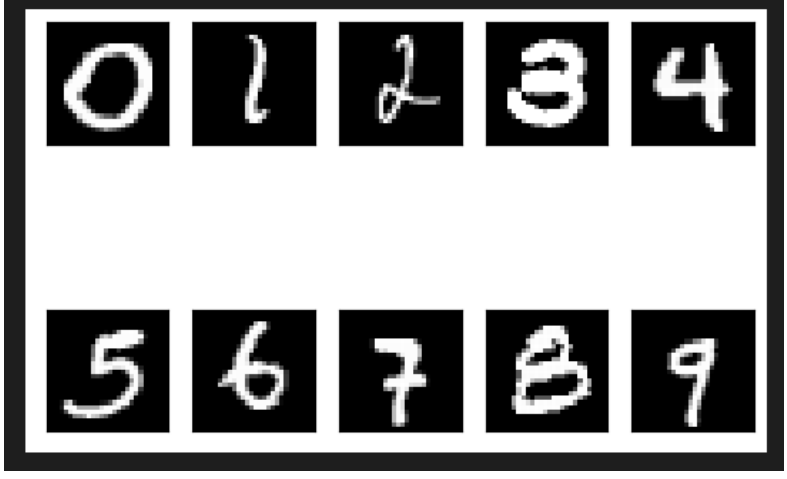
```
Train features shape: (60000, 28, 28)
Train label shape: (60000,)
Test features shape: (10000, 28, 28)
Test labels shape: (10000,)
```

شکل 25: به دست آوردن ابعاد ماتریس‌ها

ب)

```
fig, ax = plt.subplots(2, 5)
for i in range(5):
    for j in range(2):
        ind = np.random.choice(np.where(y_train == 5 * j + i)[0])
        ax[j][i].set_xticks([])
        ax[j][i].set_yticks([])
        ax[j][i].imshow(x_train[ind], cmap='gray')
```

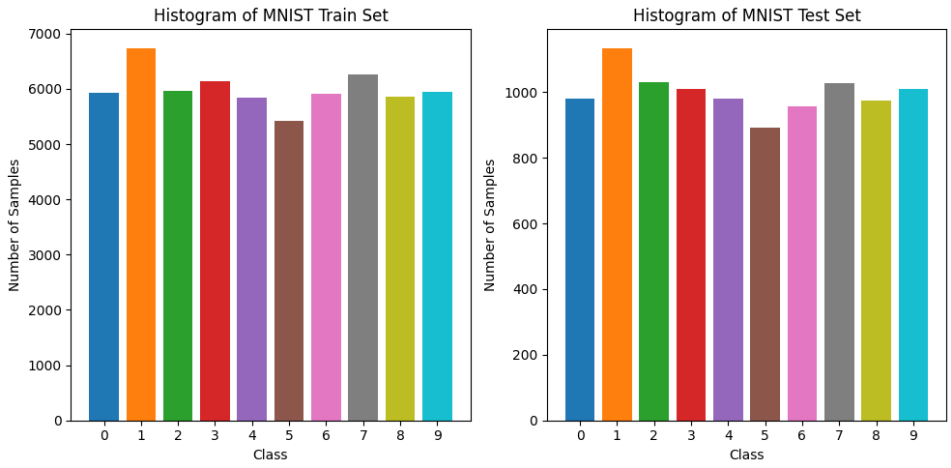
✓ 0.5s



شکل 26: رسم یک از هر کلاس داخل دیتاست

ج)

```
train_class_counts = np.bincount(y_train)
test_class_counts = np.bincount(y_test)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.bar(range(10), train_class_counts, color=plt.cm.tab10(range(10)))
plt.xlabel('Class')
plt.ylabel('Number of Samples')
plt.title('Histogram of MNIST Train Set')
plt.xticks(range(10), range(10))


plt.subplot(1, 2, 2)
plt.bar(range(10), test_class_counts, color=plt.cm.tab10(range(10)))
plt.xlabel('Class')
plt.ylabel('Number of Samples')
plt.title('Histogram of MNIST Test Set')
plt.xticks(range(10), range(10))

plt.tight_layout()
plt.show()
```



شکل 27: رسم هیستوگرام لیبل‌ها در داده‌ی ترین و تست

د)

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train = (x_train - np.min(x_train)) / (np.max(x_train) - np.min(x_train))
x_test = (x_test - np.min(x_train)) / (np.max(x_train) - np.min(x_train))
```

✓ 0.5s

شکل 28: نرمالایز کردن دیتاست

Teacher Network .2-4

```python
class Teacher(nn.Module):
    def __init__(self):
        super(Teacher, self).__init__()
        self.flatten = nn.Flatten()
        self.hidden_layer1 = nn.Linear(784, 1024)
        self.relu1 = nn.ReLU()
        self.hidden_layer2 = nn.Linear(1024, 512)
        self.relu2 = nn.ReLU()
        self.output_layer = nn.Linear(512, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.relu1(self.hidden_layer1(x))
        x = self.relu2(self.hidden_layer2(x))
        x = self.output_layer(x)
        return x
```

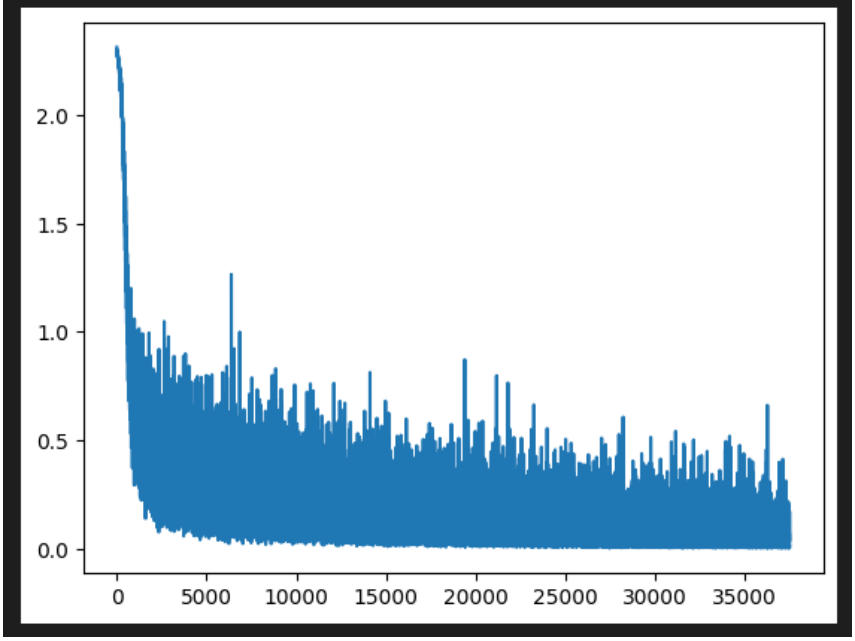
شکل 29: تعریف شبکه Teacher

```python
model_teacher = Teacher()

loss_fn = nn.CrossEntropyLoss()

learning_rate = 0.01
epochs = 20
batch_size = 32
data_loader = data.DataLoader(dataset=data.TensorDataset(x_train, y_train), batch_size=batch_size, shuffle=True)
optimizer = optim.SGD(model_teacher.parameters(), lr=learning_rate)

loss_list = []
for epoch in range(epochs):
    for i, (x, y_target) in enumerate(data_loader):
        y = model_teacher(x)
        loss = loss_fn(y, y_target)
        loss_list.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if(i%34 == 0):
            print(f'Epoch {epoch+1}/{epochs} : Progress: {i+1}/{int(T_train/batch_size)}', end='\r')
    print(f"Epoch {epoch+1}/{epochs} : cost = {sum(loss_list[-batch_size:])/len(loss_list[-batch_size:])}")
```

شکل 30: آموزش شبکه Teacher

```
plt.plot(loss_list)
```
✓ 0.3s

```
[<matplotlib.lines.Line2D at 0x1c7a3724f40>]
```



شکل 31: نمودار loss شبکه در هر تکرار

دقت شبکه روی داده‌های ترین و تست و تعداد پیش‌بینی‌های غلط به شکل زیر خواهد بود:

```
import torch.nn.functional as F

_, y_hat_train = torch.max(F.softmax(model_teacher(x_train), dim=1), dim=1)
_, y_hat_test = torch.max(F.softmax(model_teacher(x_test), dim=1), dim=1)

accuracy_test = accuracy_score(y_hat_test.detach().numpy(), y_test.detach().numpy())
accuracy_train = accuracy_score(y_hat_train.detach().numpy(), y_train.detach().numpy())
print(f"Accuracy on train: {accuracy_train}\t (Missclassification: {(y_hat_train != y_train).sum()}/{T_train})")
print(f"Accuracy on test: {accuracy_test}\t (Missclassification: {(y_hat_test != y_test).sum()}/{T_test})")
```
✓ 2.0s

```
Accuracy on train: 0.98675       (Missclassification: 795/60000)
Accuracy on test: 0.9733         (Missclassification: 267/10000)
```

شکل 32: دقت شبکه Teacher و تعداد تشخیص‌های اشتباه در هر دیتای ترین و تست

برای تشخیص کلاس از روی logitهای خروجی، کافیست مشابه عکس بالا مقادیر را از تابع softmax عبور دهیم تا برداری از احتمال برای حضور در هر لیبل را داشته باشیم. سپس از مقادیر این بردار ماکسیمم آن را پیدا کرده و لیبل آن را به عنوان خروجی تخمین زده شده گزارش می‌کنیم.

Student Network .3-4

```python
class Student(nn.Module):
    def __init__(self):
        super(Student, self).__init__()
        self.flatten = nn.Flatten()
        self.hidden_layer1 = nn.Linear(784, 128)
        self.relu1 = nn.ReLU()
        self.hidden_layer2 = nn.Linear(128, 64)
        self.relu2 = nn.ReLU()
        self.output_layer = nn.Linear(64, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.relu1(self.hidden_layer1(x))
        x = self.relu2(self.hidden_layer2(x))
        x = self.output_layer(x)
        return x
```

✓ 37.5s

شکل 33: تعریف شبکه Student

```python
model_student_1 = Student()

loss_fn = nn.CrossEntropyLoss()

learning_rate = 0.01
epochs = 10
batch_size = 32
data_loader = data.DataLoader(dataset=data.TensorDataset(x_train, y_train), batch_size=batch_size, shuffle=True)
optimizer = optim.SGD(model_student_1.parameters(), lr=learning_rate)

lost_list = []
for epoch in range(epochs):
    for i, (x, y_target) in enumerate(data_loader):
        y = model_student_1(x)
        loss = loss_fn(y, y_target)
        lost_list.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if(i%34 == 0):
            print(f'Epoch {epoch+1}/{epochs} : Progress: {i+1}/{int(T_train/batch_size)}', end='\r')
    print(f"Epoch {epoch+1}/{epochs} : cost = {sum(lost_list[-batch_size:])/len(lost_list[-batch_size:])}")
```

شکل 34: آموزش شبکه Student

```python
import torch.nn.functional as F

_, y_hat_train = torch.max(F.softmax(model_student_1(x_train), dim=1), dim=1)
_, y_hat_test = torch.max(F.softmax(model_student_1(x_test), dim=1), dim=1)

accuracy_test = accuracy_score(y_hat_test.detach().numpy(), y_test.detach().numpy())
accuracy_train = accuracy_score(y_hat_train.detach().numpy(), y_train.detach().numpy())
print(f"Accuracy on train: {accuracy_train}\t (Missclassification: {(y_hat_train != y_train).sum()}/{T_train})")
print(f"Accuracy on test: {accuracy_test}\t (Missclassification: {(y_hat_test != y_test).sum()}/{T_test})")
```

✓ 0.3s

```
Accuracy on train: 0.9607333333333333    (Missclassification: 2356/60000)
Accuracy on test: 0.9491         (Missclassification: 509/10000)
```

شکل 35: دقت شبکه Student و تعداد تشخیص‌های اشتباه در هر دیتای ترین و تست

دیده می‌شود که نتایج حاصل شده از شبکه Student از شبکه Teacher ضعیف‌تر بوده است.

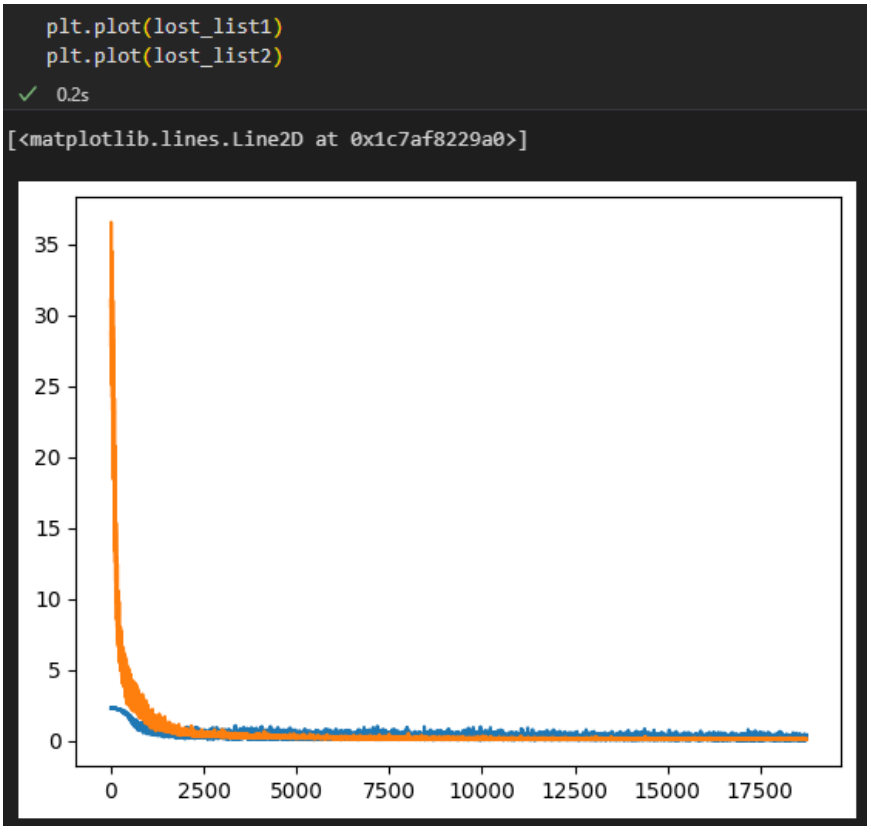
Knowledge Distillation .4-4

```
y_train_teacher = (model_teacher(x_train).detach())

model_student_2 = Student()

loss_fn = nn.MSELoss()

learning_rate = 0.01
epochs = 10
batch_size = 32
data_loader = data.DataLoader(dataset=data.TensorDataset(x_train, y_train_teacher), batch_size=batch_size, shuffle=True)

optimizer = optim.SGD(model_student_2.parameters(), lr=learning_rate)

lost_list2 = []
for epoch in range(epochs):
    for i, (x, y_target) in enumerate(data_loader):
        y = model_student_2(x)
        loss = loss_fn(y, y_target)
        lost_list2.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if(i%34 == 0):
            print(f'Epoch {epoch+1}/{epochs} : Progress: {i+1}/{int(T_train/batch_size)}', end='\r')
    print(f"Epoch {epoch+1}/{epochs} : cost = {sum(lost_list2[-batch_size:])/len(lost_list2[-batch_size:])}")

✓ 39.1s
```

شکل 36: آموزش شبکه Student با روش Knowledge Distillation

```
import torch.nn.functional as F

_, y_hat_train = torch.max(F.softmax(model_student_2(x_train), dim=1), dim=1)
_, y_hat_test = torch.max(F.softmax(model_student_2(x_test), dim=1), dim=1)

accuracy_test = accuracy_score(y_hat_test.detach().numpy(), y_test.detach().numpy())
accuracy_train = accuracy_score(y_hat_train.detach().numpy(), y_train.detach().numpy())
print(f"Accuracy on train: {accuracy_train}\t (Missclassification: {(y_hat_train != y_train).sum()}/{T_train})")
print(f"Accuracy on test: {accuracy_test}\t (Missclassification: {(y_hat_test != y_test).sum()}/{T_test})")

✓ 0.2s

Accuracy on train: 0.9775666666666667    (Missclassification: 1346/60000)
Accuracy on test: 0.9688         (Missclassification: 312/10000)
```

شکل 37: دقت شبکه Student آموزش یافته با Knowledge Distillation و تعداد تشخیص‌های اشتباه در هر دیتای ترین و تست



شکل 38: نمودار loss شبکه‌های Student در هر تکرار(آبی: شبکه آموزش یافته معمولی، نارنجی: شبکه آموزش یافته با روش Knowledge Distillation)

قابل مشاهده است که Knowledge Distillation موجب شده است نمودار loss این روش سریعتر از نمودار روش قبل