به نام خدا
دانشگاه تهران
دانشکده مهندسی
برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین اول

نام و نام خانوادگی	سپهر آزردار – پوریا تاج محرابی
شماره دانشجویی	810199395 - 810199357
تاریخ ارسال گزارش	1402.08.17

فهرست

شکل‌ها

جدول‌ها

پاسخ 1. شبکه عصبی Mcculloch-Pitts

1-1. شبکه عصبی یک لایه

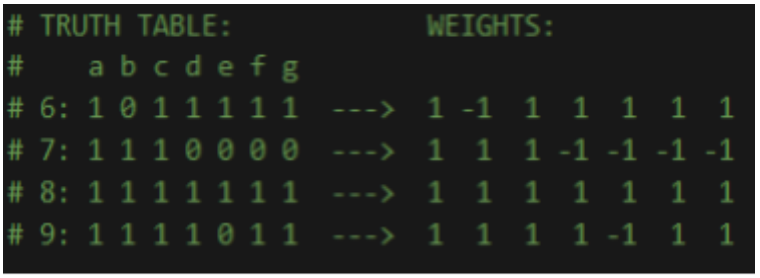
در اینجا خروجی ها مورد نظر را توانسته ایم با یک نورون ساده پیاده سازی کنیم و خروجی ها برابر با خروجی جدول صحت مورد نظر میباشد:

O_1	O_2	O_3	O_4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

```
[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 1]
[0, 0, 0, 0]
```

شکل1. خروجی شبکه عصبی یک لایه

در اینجا منطق استفاده شده برای انتخاب وزن یال ها به این شکل هست که اگر ورودی مورد نظر مان فعال باشد وزن مربوط به یال آن را 1 میگذاریم و در غیر این صورت وزن یال 1- میباشد. و آستانه برابر با جمع تعداد ورودی هایی هست که میخواهیم فعال باشند(تعدادیال هایی که وزرن 1 دارند.) و نکته این هست که که اگر برابر با آستانه باشد مجموع ورودی ها آنگاه نورون fire میکند. وزن یال های هر نورون (که وظیفه تشخیص دقیقا یک رقم را دارند) با این روش تعیین میشود.



شکل2.وزن یال های نورون ها شبکه عصبی یک لایه

بنابراین بدین شکل جدول صحت خواسته شده، پیاده سازی میشود و هر نورون (لایه خروجی))دقیقا به یک ردیف از این جدول درستی اشاره میکند. و فقط در صورتی تمام ورودی ها دقیقا مانند ردیف مورد نظر از جدول درستی باشند، fire میکنند و در غیر این صورت 0 میباشد.

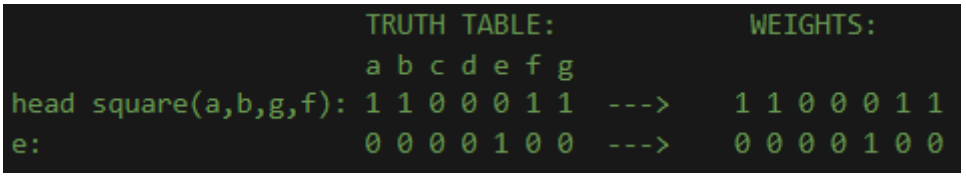
1-2. شبکه عصبی دو لایه

الف) وزن های این شبکه را به گونه ای تعیین که خروجی مورد انتظار را به ازای هر ورودی ایجاد کند.

در اینجا ما در لایه پنهان مان دو نورون داریم و در لایه اخر 4 نورون داریم. و هر نورون دو حالت بیشتر ندارد پس این دو نورون لایه ی میانی حداکثر میتوانند 4 حالت متمایز را ایجاد کنند و بر اساس آن ما می توانیم حداکثر 4 خروجی متمایز داشته باشیم. پس ما باید پترن ها و فیچر هایی را (توسط همین نورون ها میانی) از ورودی مان تشخیص دهیم که با ازای آن ها، نورون های میانی هر یک از 4 ورودی ما به یکی از 4 حالت گفته شده مپ کنند.

ب)

در اینجا همان طوری که در کد کامنت شده است. دو نورون در لایه نهان داریم. و هر کدام از آن ها دو خروجی به ما میدهند، بنابراین در مجموع ما 4 حالت مختلف خواهیم داشت. و باید به صورتی فیچر/pattern ها را تشخیص دهیم که هر کدام از آن ها یکی از این 4 حالت را نشان میدهند. در واقع باید به informative ترین حالت ممکن، این فیچر ها را استخراج کنیم تا بتوانیم هر چهار حالت را از یکدیگر تمیز دهیم. در واقع باید فیجر ها و پترن هایی را انتخاب کنیم که در لایه اخر، ورودی ها از همدیگر به صورت خطی جدا (linearly-separable) پذیر باشند.
پترن های متفاوتی در میتوان در نظر گرفت. در اینجا وجود یک مربع در قسمت بالایی و فعال بودن سگمنت e مورد بررسی قرار گرفتند و جدول درستی و متناظرا وزن یال ها در شکل زیر آورده شده است:



شکل3. وزن یال های میانی در شبکه عصبی دو لایه

نکته: جدول درستی داده شده در سوال اشتباه جزئی داشت. برای فعال بودن عدد 9 باید سگمنت e خاموش باشد. همانند شکل داده شده در صورت سوال اما به جای سگمنت c خاموش است. و این موضوع در اجرا برنامه لحاظ شده است.

یک نورون وجود مربع در قسمت بالایی را بررسی میکند، یعنی حضور

a b c d e f g

head square(a,b,g,f): 1 1 0 0 0 1 1

و نورون دیگر حضور المنت e به تنهایی را مورد بررسی قرار میدهد.

ج)

مقایسه پارامتر ها:

شبکه یک لایه: 7 تا ورودی داریم و 4 تا خروجی. و شبکه dense/fully connected میباشد بنابراین در مجموع 7*4=28 پارامتر داریم. وزن یال ها پارامتر های ما هستند.

شبکه دو لایه: 7 تا ورودی داریم و 4 تا خروجی. و دو نورون در لایه نهان. بنابراین 7 * 2 + 2 * 4 = 22 پارامتر داریم.

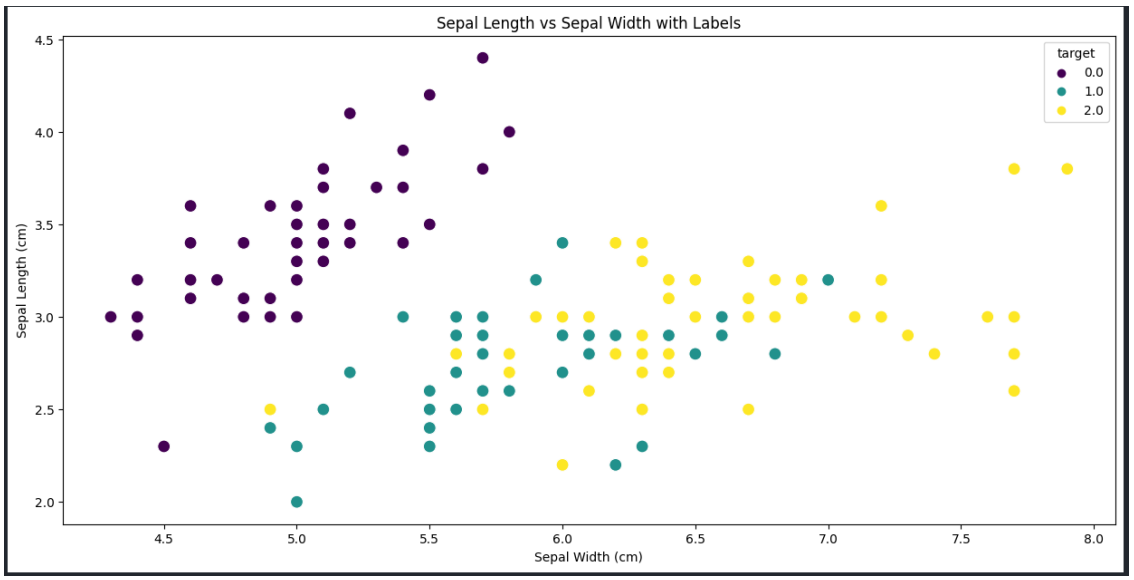
همان طور که انتظار میرفت با افزایش لایه ها قدرت شبکه بیشتر میشود و به نوعی حافظه یا sequence در شبکه بوجود میاید. بنابراین با پارامتر های کمتر می توان همان نتایج را بدست آورد و یا حتی بهتر عمل کرد زیرا که افزایش عمق باعث افزایش ظرفیت شبکه در prediction میشود.

پاسخ ۲ - آموزش شبکه‌های Adaline و Madaline

Adaline .2-1

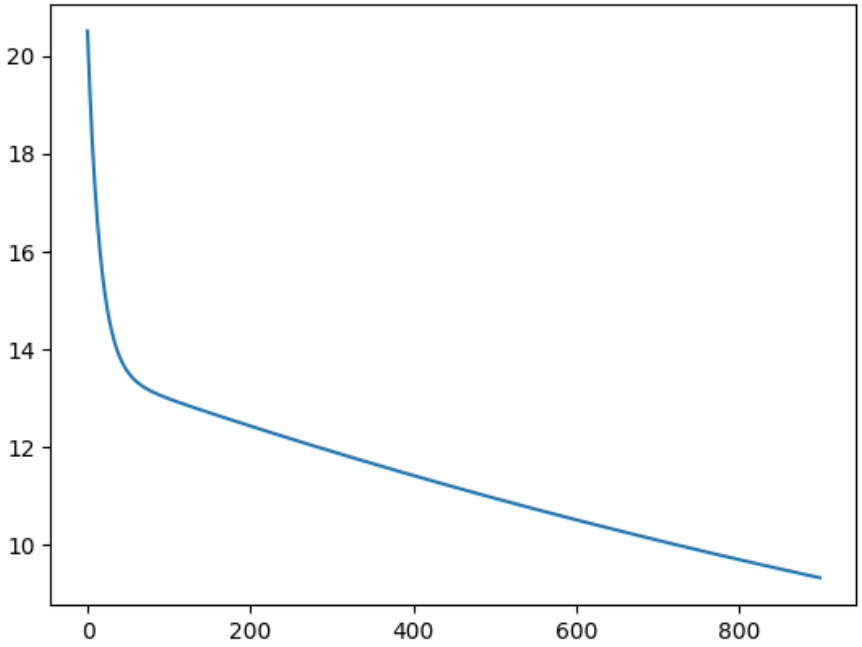
الف)

داده های ورودی متشکل از سه کلاس هستند که یک کلاس با لیبل ۰.۰ به صورت خطی قابل جدا کردن از بقیه داده ها هستند . داده های دو لیبل دیگر یعنی ۱.۰ و ۲.۰به وضوح به صورت خطی قابل جدا کردن نیستند



شکل۵. نمودار پراکندگی داده ها

با توجه به داده ها یک شبکه متشکل از یک نورون Adaline طراحی کردیم که epoch 900 آموزش داده شده است در نمودار زیر روند کاهش Loss دیده می شود .



شکل۶. نمودار Cost بر حسب Epoch

در نمودار زیر نتیجه تست کردن مدلمان را روی داده تست نشان می دهد که از تقسیم دادگان به نسبت ۰.۸ - ۰.۲ به دست آمده بود

جدول1. دقت تشخیص لیبل Setosa

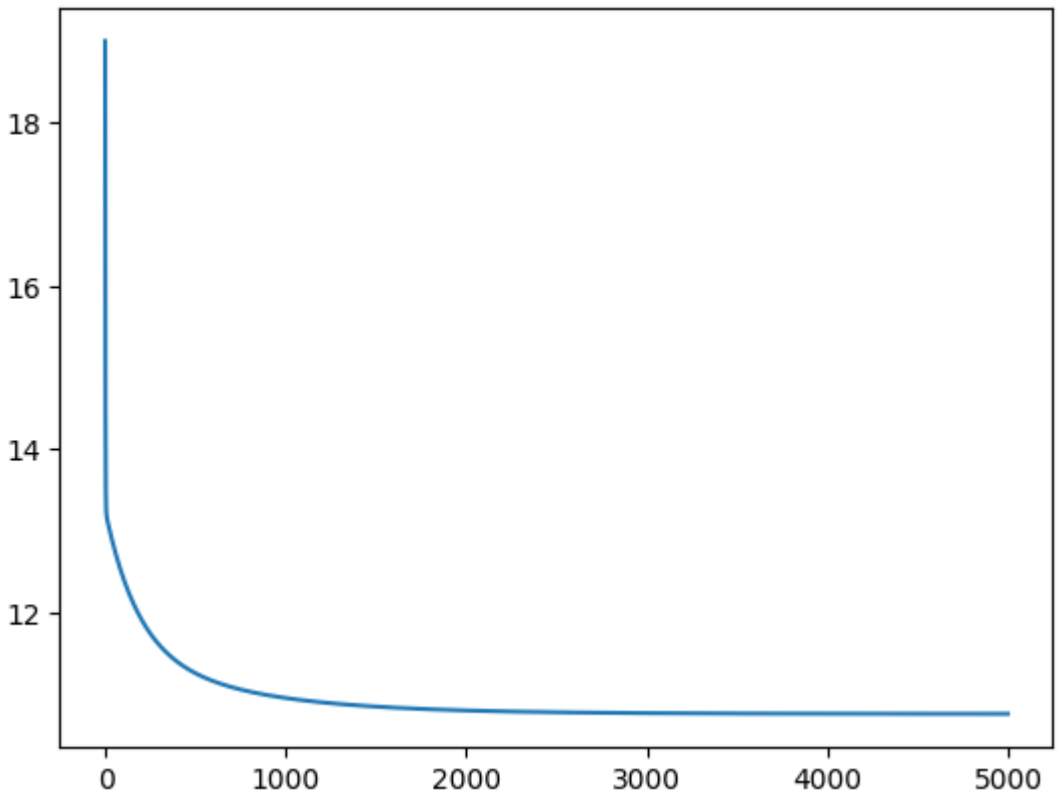
Accuracy / All of the dataset	Accuracy / Train	Accuracy / Test
0.66	0.76	0.73

همانطوری که مشهود بود می توانستیم با چشم نیز داده ها را خطی جدا کنیم و Adaline نیز موفق شد درصد قابل توجهی از Setosa ها را شناسایی کند

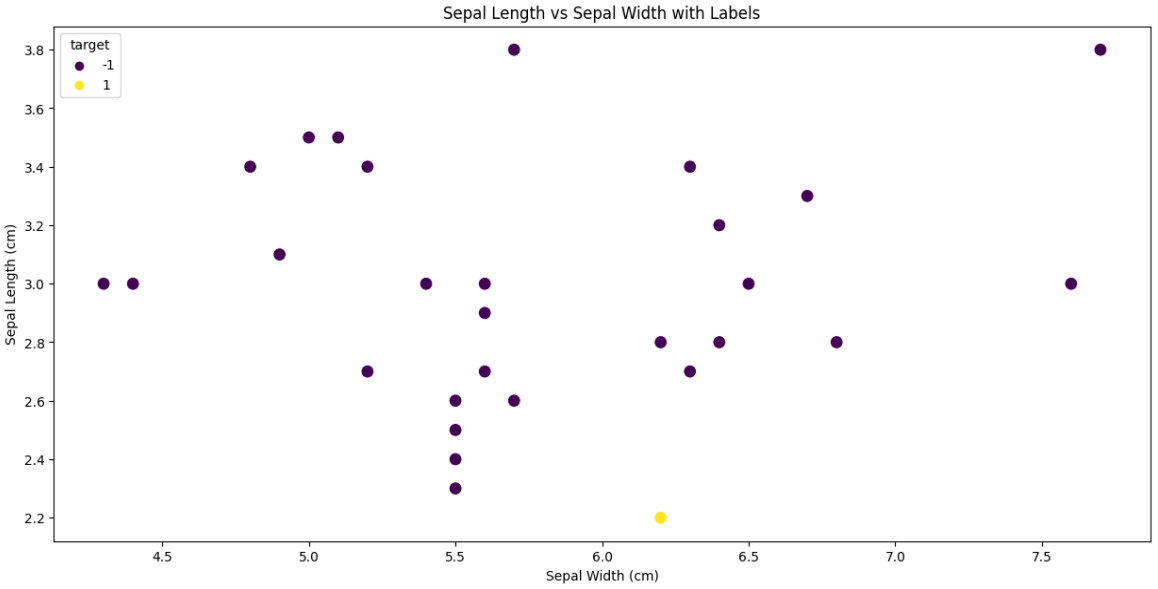
ب)

در این قسمت سعی میکنیم داده هایی که به صورت شهودی از لحاظ خطی قابل جدا سازی نبودند را جدا کنیم و نتیجه خوبی نمیگیریم .

در واقع Adaline نمی تواند رابطه های پیچیده تر را از هم جدا کند .



شکل ۷. نمودار Loss روی دیتای Train برای تشخیص داده ها با لیبل(Versicolor (2.0



شکل ۸. پیش بینی های مدل روی Versicolor در دادگان تست

دلیل اینکه در این قسمت نسبت به قسمت قبل نتیجه خوبی نمیگیریم به دلیل ذات متفاوت و رابطه غیر خطی Versicolor و Setosa و در هم تنیدگی غیر خطی Versicolor با Virginica است که با یک نورون قابل تمییز دادن نیست .

جدول 2. دقت تشخیص لیبل Versicolor

Accuracy / All of the dataset	Accuracy / Train	Accuracy / Test
0.0066666666666666667	0~	0.03333333333333333

Madaline 2-2

الف) MR II : برای Train کردن و آپدیت کردن وزن ها استفاده می شود . در ابتدا وزن ها به صورت رندم ست می شوند سپس برای آپدیت شدن وزن ها که در واقع از نوعی Back Propagation با روش Gradient Descent استفاده می کند استفاده می شود.

این نوع Gradient Descent به شکل رسمی ای که در حال حاضر در شبکه های عصبی مدرن وجود دارد نیست و فقط برای دو لایه معنی دارد.

برای آپدیت وزن ها از قاعده زیر پیروی می کنیم:

اگر ارور صفر نبود به ازای هر unit موجود madaline وزن های آن از رابطه زیر پیروی می کند

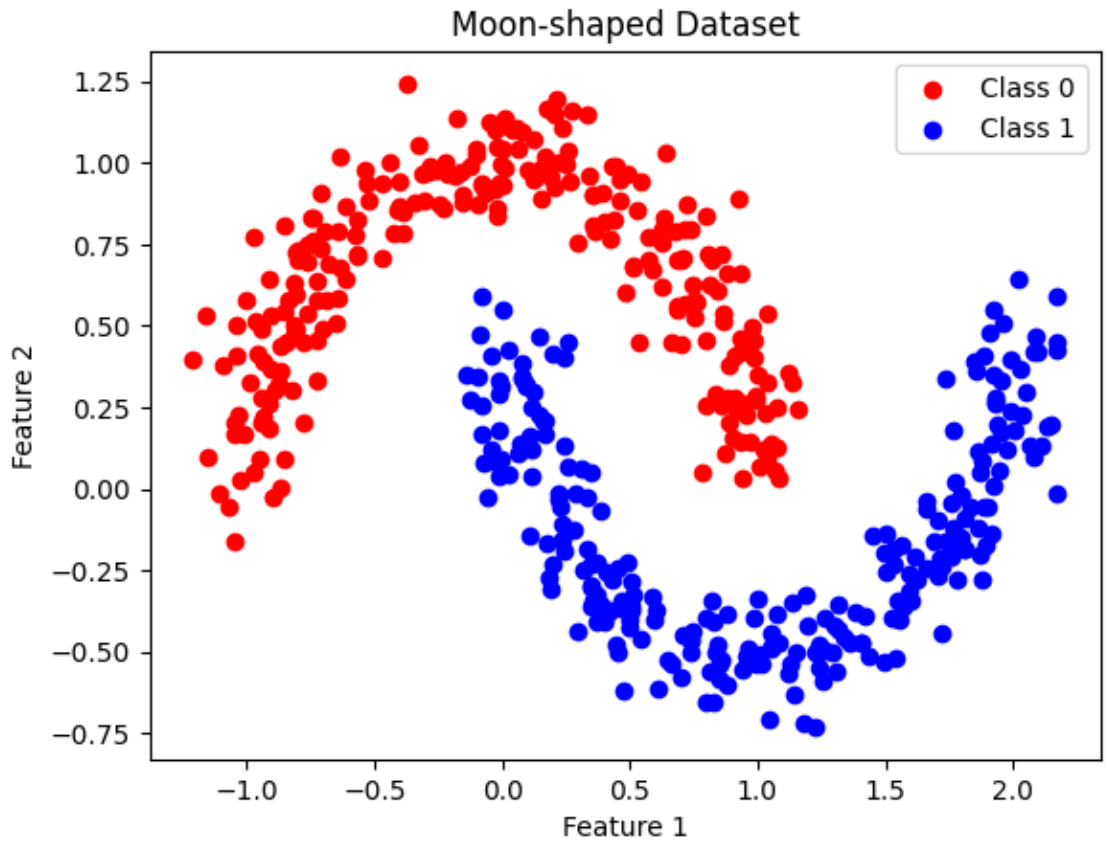
$$b_j(new) = b_j(old) + \alpha(1 - z_{inj})$$

$$w_{ij}(new) = w_{ij}(old) + \alpha(1 - z_{inj})x_i$$

شکل 9.فرمول های آپدیت وزن ها در Madaline دو لایه ای

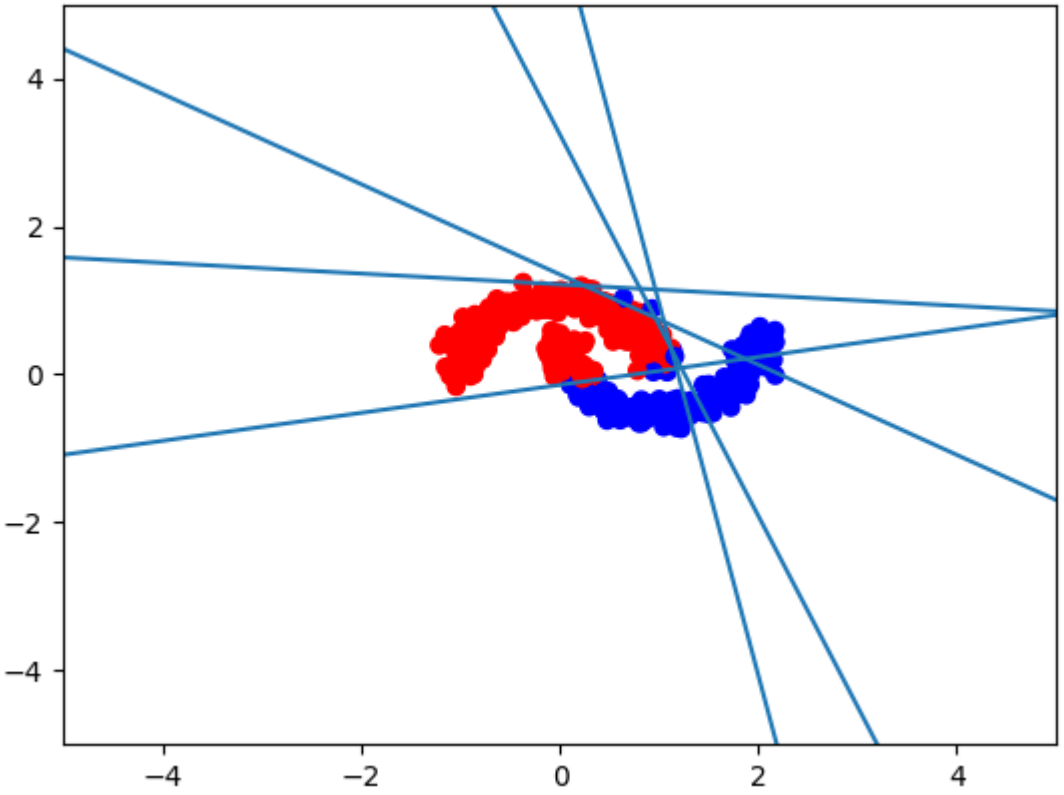
در این روش سعی می شود که تغییر وزن در جهتی باشد که اگر خروجی آن Neuron Flip کند بیشترین کاهش را در Loss / Error ایجاد کند یعنی وزن نورونی را تغییر می دهد که تغییر علامت آن بیشترین کاهش را در ارور ایجاد کند .

داده های moon shaped را load کرده و باز مشاهده می کنیم که به صورت خطی و از درجه اول قابل جدا سازی به شکل خوبی نیستند و استفاده از Madeline را توجیه می کند .

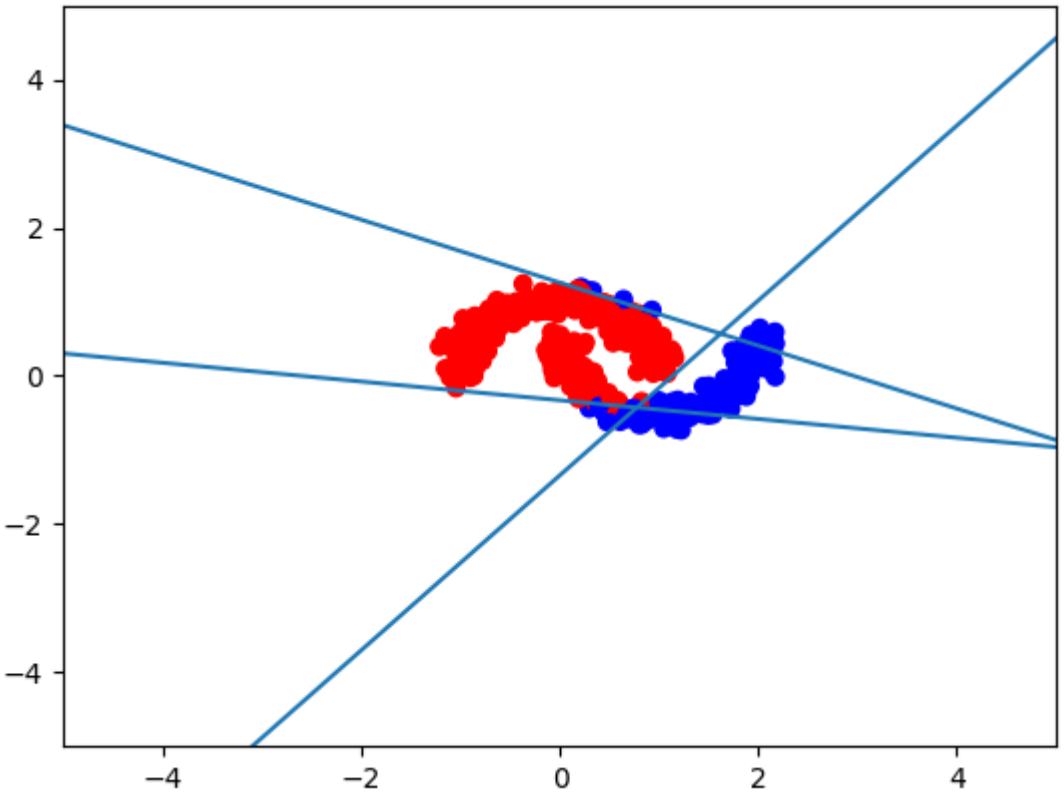


شکل 10. پیش بینی های مدل روی Versicolor در دادگان تست

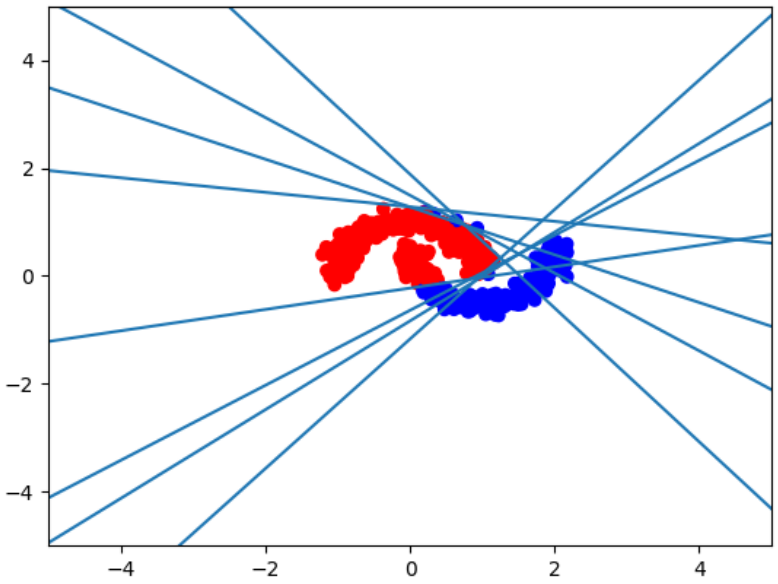
سعی شده که با کم نگه داشتن تعداد epoch ها از overfitting جلوگیری شود .
در صورت استفاده از ۵ نورون بهترین دقت جداسازی و در صورت استفاده از ۳ نورون بدترین دقت را میگیریم . دلیل این مورد می تواند Overfitting روی تعداد نورون بالا باشد .



شکل ۱۱ . استفاده از ۵ نورون



شکل ۱۲. استفاده از ۳ نورون



شکل ۱۴. ۸ نورون

جدول 3. دقت بر حسب نورون

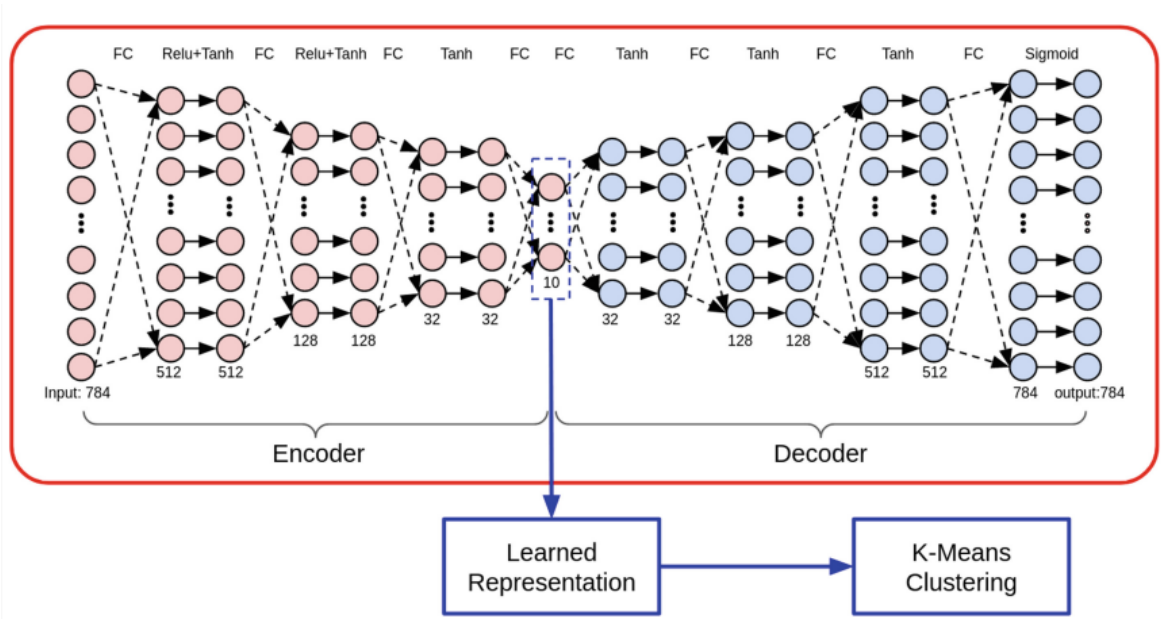
3 Neurons	5 Neurons	8 Neurons	تعداد نورون
0.816	**0.886**	0.874	دقت جداسازی

پاسخ ۳ – خوشه بندی با استفاده از Autoencoder

3-1. پیاده سازی Autoencoder Deep برای کاهش ابعاد داده‌ها

1. پیاده سازی autoencoder:

اتوانکودر گفته شده را با مشخصات گفته شده پیاده سازی کردیم و از ACTIVATION FUNCTION های RELU, TANH و SIGMOID استفاده شده است. در بعضی جاها که fc استفاده شده است و دقیقا توسط مقاله معلوم نشده بود که از چه تابعی استفاده کنیم. از تابع tanh استفاده کردیم.



شکل ۱۵. autoencoder مقاله DAC

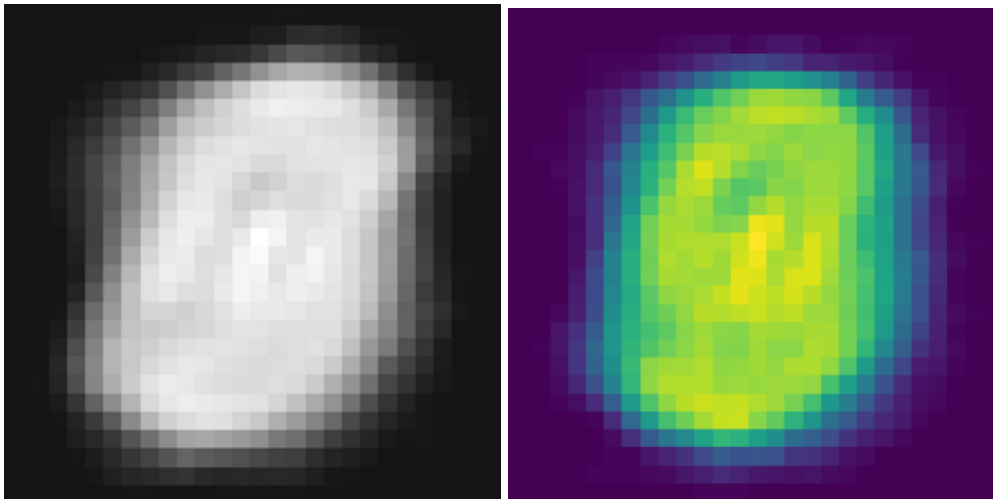
2. آموزش autoencoder:

ابتدا فرمول گفته شده بخش 4 مقاله، که فرمول(5) که مربوط به محاسبه وزن پیکسل ها میباشد را پیاده‌سازی کردیم.

$$w_i = \frac{\sum_{l_p=l_q} e^{-(x_{ip}-x_{iq})^2}}{\sum_{l_p=l_q} 1} \cdot \frac{\sum_{l_p\neq l_q} (1-e^{-(x_{ip}-x_{iq})^2})}{\sum_{l_p\neq l_q} 1}$$

شکل ۱۶. فرمول محاسبه وزن featureها

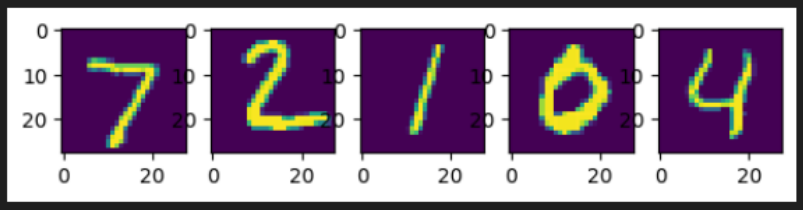
همان طوری که در مقاله ذکر شده بود برای محاسبه وزن هر پیکسل از چند سمپل لیبل خورده استفاده کردیم و عکس بدست آمده همانند عکس ذکر شده در مقاله شد. و در این عکس پیکسل هایی که اهمیت بیشتری دارند رنگ روشن تری دارند.(چپی عکس مقاله میباشد.)

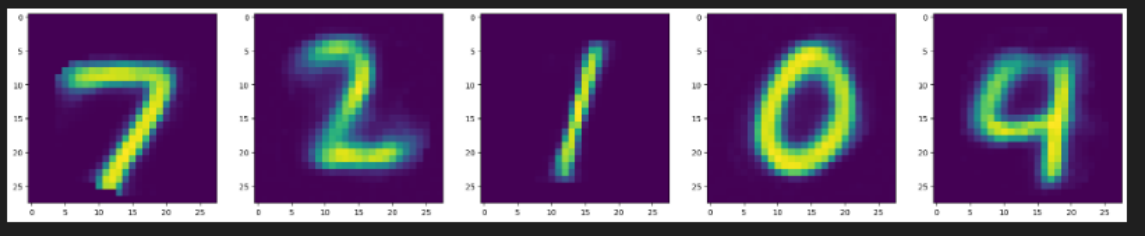


شکل ۱۷. عکس اهمیت پیکسل‌ها

سپس از فرمول های شماره 4 و6 برای پیاده‌سازی **loss function** استفاده کردیم. و با استفاده از autoencoder را آموزش دادیم. به خروجی زیر رسیدیم. از آنجایی که به پیکسل های اطراف وزنی اختصاص نمیدهد و وزن ها تقریبا 0 است. پس autoencoder سعی نمیکند که برای پیکسل هایی که در نواحی هستند که وزن آن ها 0 هست، خود را اپدیت بکند چرا که اصلا اروری به ازای آن نواحی دریافت نمیکند پس این نواحی هرچیزی میتوانند باشند. برای رفع این موضوع میتوان یک مقدار خیلی کمی به وزن کلی بدست امده برای فیچر ها اضافه کرد. با اینکار پیکسل های نواحی اطراف نیز به درستی predict میشوند. برای همین مقداری اندکی به w اضافه شده است. و همین طور به علت اینکه مقدار وزن بدست آمده برای هر فیچر کم بود، در یک عدد بزرگ ضرب شده است تا گرادیان به خوبی به عقب برگردد و یادگیری بهتر اتفاق بیفتد.

```
Epoch 9/10
118/118 [==============================] - 3s 29ms/step - loss: 2740618.7500 - val_loss: 2741544.7500 -
Epoch 10/10
118/118 [==============================] - 3s 29ms/step - loss: 2683971.5000 - val_loss: 2721794.2500 -
313/313 [==============================] - 1s 2ms/step - loss: 2721794.7500
Test Loss: 2721794.75
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
```





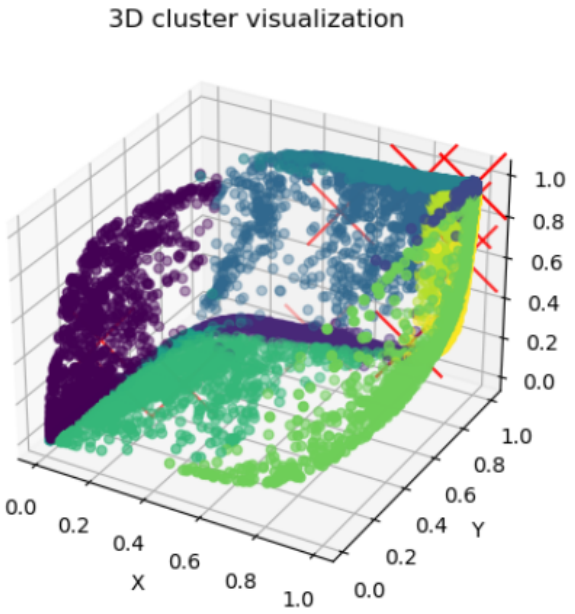
```
(60000, 784)
(10000, 784)
Epoch 1/10
118/118 [==============================] - 5s 31ms/step - loss: 5608625.0000 - val_loss: 5183535.0000 - lr: 0.0030
Epoch 2/10
118/118 [==============================] - 4s 31ms/step - loss: 4652151.5000 - val_loss: 4281727.0000 - lr: 0.0030
Epoch 3/10
118/118 [==============================] - 3s 29ms/step - loss: 3809012.0000 - val_loss: 3546791.5000 - lr: 0.0030
Epoch 4/10
118/118 [==============================] - 3s 30ms/step - loss: 3368858.7500 - val_loss: 3215300.2500 - lr: 0.0030
Epoch 5/10
118/118 [==============================] - 3s 29ms/step - loss: 3116859.7500 - val_loss: 3041157.2500 - lr: 0.0030
Epoch 6/10
118/118 [==============================] - 3s 29ms/step - loss: 2964973.7500 - val_loss: 2944246.2500 - lr: 0.0030
Epoch 7/10
118/118 [==============================] - 3s 27ms/step - loss: 2883724.7500 - val_loss: 2866510.5000 - lr: 0.0030
Epoch 8/10
118/118 [==============================] - 3s 29ms/step - loss: 2797506.7500 - val_loss: 2798983.5000 - lr: 0.0030
Epoch 9/10
118/118 [==============================] - 3s 29ms/step - loss: 2740618.7500 - val_loss: 2741544.7500 - lr: 0.0030
Epoch 10/10
118/118 [==============================] - 3s 29ms/step - loss: 2683971.5000 - val_loss: 2721794.2500 - lr: 0.0030
313/313 [==============================] - 1s 2ms/step - loss: 2721794.7500
Test Loss: 2721794.75
```

شکل ۱۸. خروجی آموزش autoencoder و predict چند نمونه

3. انجام تسک clustering با K-Means:

برای انجام این تسک لایه اخر encoder را که بخش اول autoencoder می باشد را به عنوان ورودی به K-means دادیم. در واقع هدف از استفاده از autoencoder برای این بود که dimention-reduction داشته باشیم. زیرا در بعد ها پایین تر الگوریتم K-means بهتر عمل میکنند. در اینجا لایه اخر encoder ابعاد را از 784(28*28) به 10 کاهش داده است و همچنین با استفاده از custom loss function گفته شده در مقاله، کل مدل و در نتیجه لایه اخر encoder سعی میکند که فیچر های بهتر و informative تر و مهم تری را یاد بگیرد.

4. ارزیابی با معیار ARI:




```python
from sklearn.metrics import adjusted_rand_score
true_labels = y_test
# Predicted labels from KMeans clustering
predicted_labels = kmeans.labels_

# Calculate the ARI
ari_score = adjusted_rand_score(true_labels, predicted_labels)

print("Adjusted Rand Index:", ari_score)
```

```
Adjusted Rand Index: 0.6635763283467994
```

شکل ۱۹.معیار ARI

جدول 4. مقایسه ARI روش ها مختلف

	K-Means	PCA	DAC
ARI	0.3833	0.38685	0.536089

در اینجا همان طور که مشاهده میکنیم در مقاله هم ذکر شده است. روش جدید از روش های دیگر بهتر عمل میکند و در اینجا مانند مقاله شاخص ARI، در حالت DAC بهتر از دو روش دیگر عمل کرده است. در صورت انتخاب کلاستر ها به صورت رندوم مقدار این معیار برابر با 0 مشود و حداکثر مقدار(در بهترین حالت) این معیار برابر با 1 میباشد. این معیار نسخه improved شده شاخص RI است. که نرمالایز شده است و مقدار آن بین 0.5- و 1 میباشد. فرمول آن به دین شکل میباشد: `(ARI = (RI - Expected_RI) / (max(RI) - Expected_RI` و این معیار زمانی قابل استفاده است که ما لیبل اصلی نمونه ها را میدانیم. این روش بین هر در نمونه محاسباتی رو انجام میدهد و به طور کلی هر چقدر داده به مرکز کلاستر خود نزدیک باشند و مرکز کلاستر ها از یکدیگر دور تر باشد، الگوریتم بهتر عمل کرده است.

پاسخ ۴ – شبکه‌ی Multi-Layer Perceptron

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

الف)

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

print("Number of training samples:", len(x_train))
print("Number of testing samples:", len(x_test))
print("Shape of training data:", x_train.shape)
print("Shape of testing data:", x_test.shape)
print("Shape of labels:", y_train.shape)

```

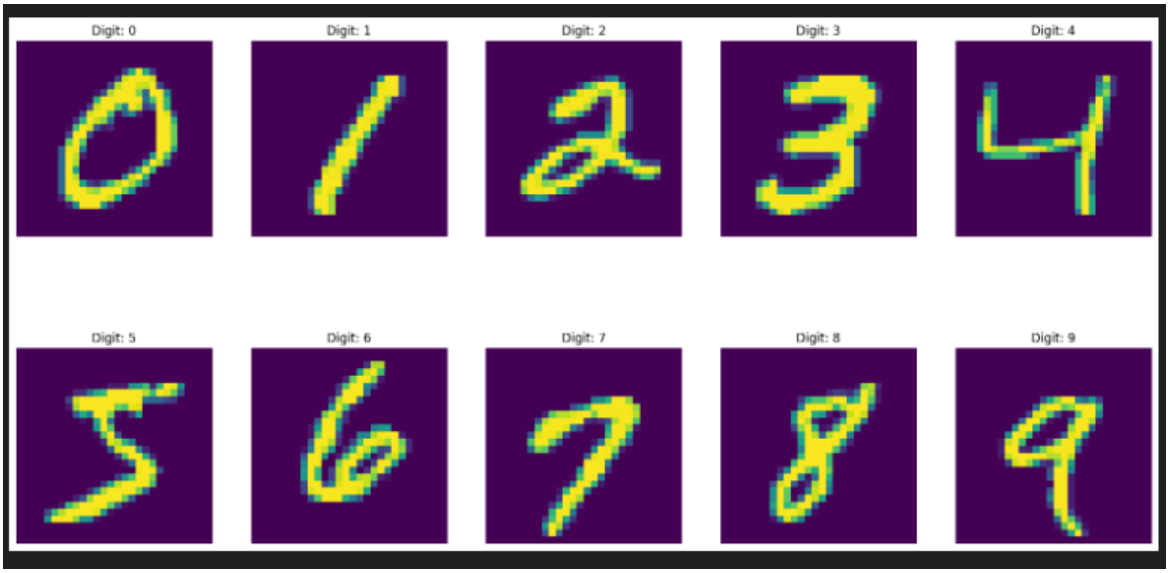
```
Number of training samples: 60000
Number of testing samples: 10000
Shape of training data: (60000, 28, 28)
Shape of testing data: (10000, 28, 28)
Shape of labels: (60000,)
```

شکل ۲۱. ابعاد و تعداد MNIST

همان طوری که مشاهده میکنید، دیتاست MNIST شامل 70000 عکس از ارقام 0 تا 9 میباشد. این عکس ها بع صورت سیاه و سفید میباشند و ابعاد هر عکس برابر با 28*28 است که برابر با یک آرایه یک بعدی به اندازه 784 است. 10000 تا نمونه برای داده تست داریم و 60000 تا برای داده آموزش داریم.

ب)

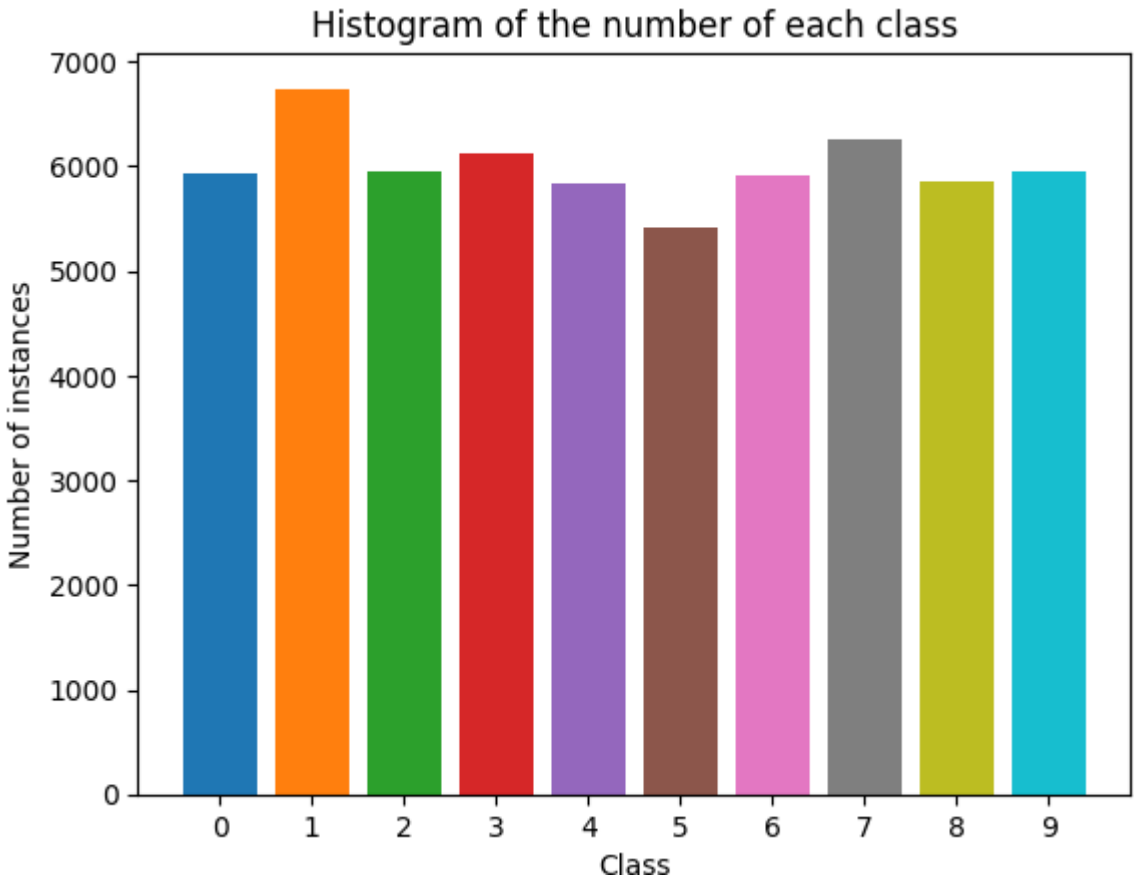
همان طور که گفته شد این دیتا ست مربوط به راقام 0 تا 9 هست و شامل 10 کتگوری میشود.



شکل ۲۲. نمونه ای از هر کتگوری

ج)

تقریبا تعداد نمونه ها هر کتگوری باهم برابر است و نیازی OVERSMAPLING و UNDERSAMPLING نیست.



شکل ۲۳. نمودار هیستوگرام توزیع نمونه بین کتگوری ها

د)

از آنجایی که عکس ها grayscale هستند، حداقل مقدار آن ها برابر با 0 است و حداکثر مقدار آن برابر با 255 هست. با این کار حتما تمام داد ها در بازه 0 تا 1 میفتند که همان هدف نرمالایز کردن است.

```
nrmd_x_train = x_train.astype('float32') / 255.
nrmd_x_test = x_test.astype('float32') / 255.
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
print(np.max(nrmd_x_train), np.min(nrmd_x_train),
      np.max(nrmd_x_test), np.min(nrmd_x_test))
print(x_train.shape)
print(x_test.shape)
```

```
1.0 0.0 1.0 0.0
(60000, 28, 28)
(10000, 28, 28)
```

شکل ۲۴. حداکثر مقدار و حداقل مقدار پس از normalization

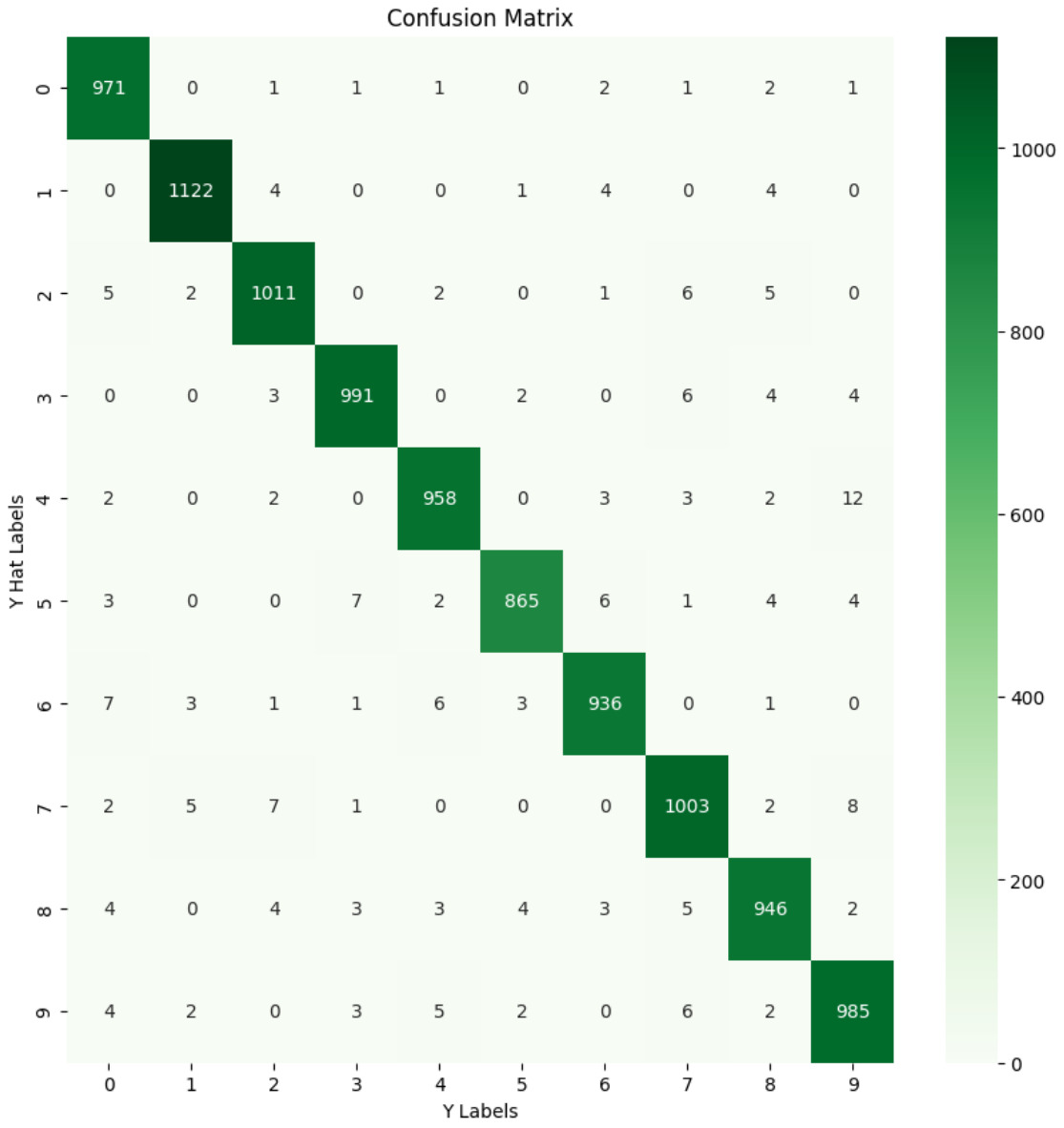
4-2. Teacher Network

همانطور که مشاهده میکنید به روی داده های آزمون accuracy 0.97 داریم. و با استفاده از confusion matrix می توان فهمید دقیقا هر نمونه یک کتگوری، چگونه misclassified شده است. تعداد کل misclassification ها برابر با 209 میباشد. همان طور که در کد هم آورده شده است، کتگوریی که مقدار بیشتری دارد را، به عنوان کتگوری تشخیص داده شده، توسط مدل در نظر میگیریم. و بدین شکل مدل سعی میکند، به کتگوری که درست است، مقدار بیشتری را نسبت به بقیه نسبت دهد.

```
1  teacher_loss, teacher_acc = teacher_model.evaluate(nrmd_x_test,y_test)
2  print(f"Loss : - Accuracy :", teacher_loss, teacher_acc)
3  # number of misclassifications:
4  teacher_predictions = teacher_model.predict(nrmd_x_test)
5  predicted_classes = np.argmax(teacher_predictions, axis=1)
6  true_classes = np.argmax(y_test, axis=1)
7  num_misclassifications = np.count_nonzero(predicted_classes != true_classes)
8  print("Number of misclassifications:", num_misclassifications)
9  print((1 - teacher_acc) * len(y_test))
```

✓ 3.0s

```
313/313 [==============================] - 1s 5ms/step - loss: 0.0666 - accuracy: 0.9791
Loss : - Accuracy : 0.06661488860845566 0.9790999889373779
313/313 [==============================] - 1s 4ms/step
Number of misclassifications: 209
209.0001106262207
```

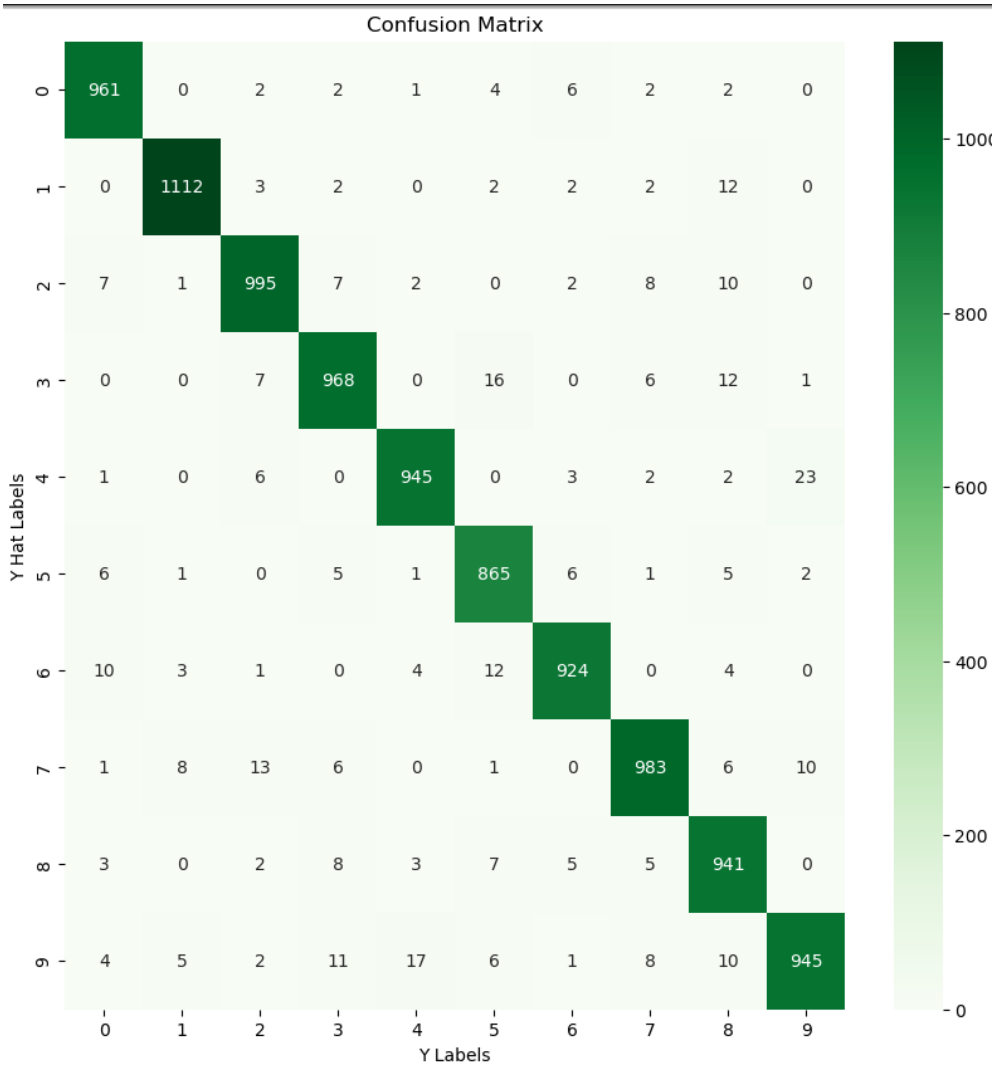


شکل ۲۵. accuracy & confusion matrix

4-3. Student Networks

در اینجا لاس بیشتر میشود و accuracy نسبت به حالت قبل کمتر میشود. زیرا که مدل student ظرفیت کمتری نسبت به مدل Teacher دارد. تعداد نورون های لایه ها student نسبت به teacher کمتر است. و همچنین روی تعداد epoch ها کمتری ترین میشود. بنابراین accuracy کمتری نسبت به teacher دارد. تعداد غلط ها برابر با 361 میباشد.

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1195 - accuracy: 0.9639
Number of misclassifications: 361
360.9997034072876
```

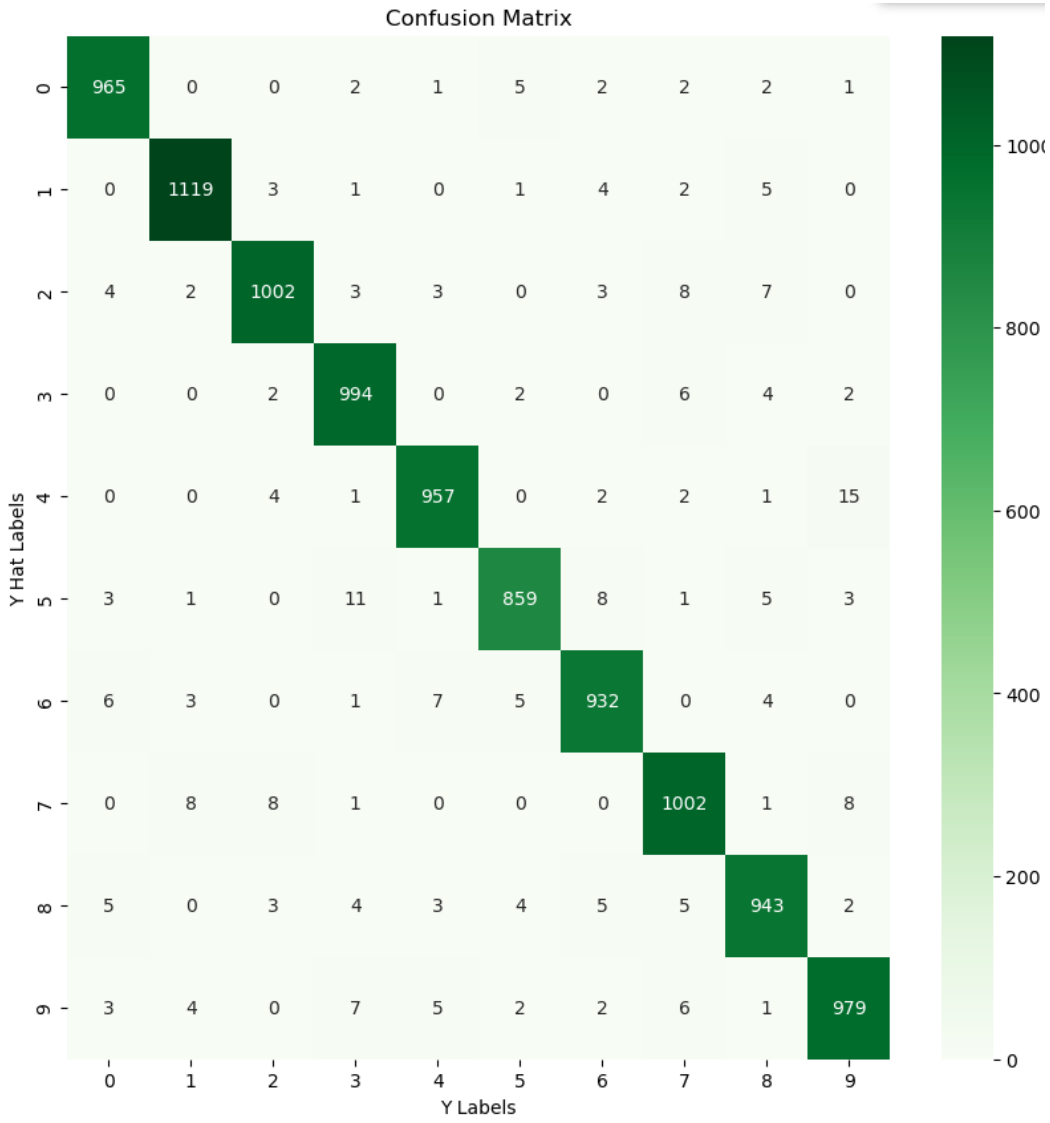


شکل ۲۶. student accuracy & confusion matrix

4-4. Knowledge Distillation

همان طور که مشاهده میکنید، accuracy یا همان دقت، مدل student بهتر شده است و تعداد خطا عایش کاهش یافته است و از 361 به 248 رسیده است. این نشان دهنده، کارایی این روش میباشد.

```
313/313 [==============================] - 1s 2ms/step - loss: 28.2220 - accuracy: 0.9752
Number of misclassifications: 248
248.00002574920654
```



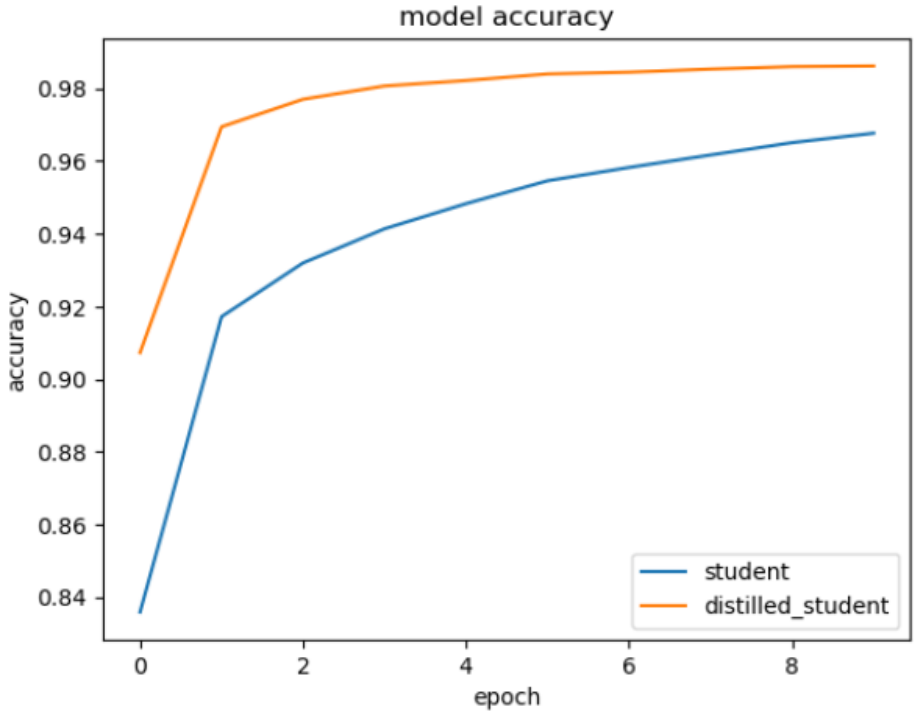
شکل ۲۷. Knowledge Distillation accuracy & confusion matrix

الف)

همان طور که گفته شد تعداد غلط ها کاهش یافته است و دقت مدل student بالاتر رفته است. در واقع مدل student در روش **Knowledge Distillation** سعی میکند، تا جایی که میتواند، مانند مدل teacher رفتار کند با این کار در واقع دانش و generalization مدل capable تر به مدل کوچکتر(student منتقل میشود) و مدل کوچکتر بهتر عمل میکند. در واقع در اینجا دقت مدل student از حالت قبلی خودش بهتر و مدل teacher کمتر است. زیرا سعی میکند که رفتار مدل teacher را تا جایی که میتواند تقلید کند و بنابراین از مدل teacher دقت کمتری دارد. نکته ای که وجود دارد این است، علت اینکه این اتفاق میفتد این است که مدل student بسیار بهتر میتواند رفتار مدل teacher که soft , smooth تر از واقعیت است، را تقلید کند. و به خاطر همین از حالت عادی بهتر عمل میکند.

ب)

همان طور که مشاهده میکنید، از ابتدا دقت روش **Knowledge Distillation** بیشتر است. زیرا، در این روش مدل student هدف اسان تری دارد و در واقع تقلید از teacher بیسار اسان تر از تقلید واقعیت میباشد. در واقع گویی واقعیت یکبار از فیلتر teacher عبور کرده است و smooth تر شده است. برای مثال داده هایی که تشخیص آنها برای خود teacher هم سخت بوده اند، به صورت soft تر در اختیار student قرار گرفته اند. و این امر باعث میشود که student بتواند generaliztion معلم(teacher) را بخوبی تقلید کند. همچنین با اینکار، خروجی ها informative تری به student میرسد و میتواند بهتر عمل کند. در واقع teacher نسخه generalized شده را به student میدهد، بنابراین این دانش به student منتقل میشود و بهتر عمل میکند.



شکل ۲۸. Knowledge Distillation accuracy VS Typical model accuracy