تمرین دوم شبکه عصبی

سوال اول

1- مبحث **overfitting** در شبکه های عصبی به این بر میگردد که مدل ما بیش اندازه داده ها را یاد بگیرد و مدل داده های train را با دقت بالایی یاد می گیرد اما در داده های تست دقت پایینی دارد.

2- overfitting زمانی رخ میدهد که مدل ما به جای یادگیری الگو های مهم به جزئیات داده های train می پردازد. هر چه فاصله بین دقت مدل برای داده های آموزشی و تستی بیشتر باشد می گوییم overfitting بیشتر رخ داده است.

3- روش های جلوگیری از overfitting:

- **کاهش پیچیدگی مدل:** هر چه مدل ساده تر باشد پارامتر ها و لایه ها کمتر است و از این رو مجبور به یادگیری ویژگی های مهم است.
- **توقف زود هنگام (Early Stopping):** این روش به این صورت کار میکند که داده های تست را در حین آموزش به مدل می دهد تا عملکرد مدل را بسنجد و زمانی که خطای مدل رو به افزایش باشد فرایند آموزش متوقف میشود چون مدل در حال یادگیری بیش از حد داده های آموزش است.
- **Cross-Validation:** این روش داده های آموزشی را به چند قسمت تقسیم می کند و آموزش مدل روی ترکیبی از این قسمت ها رخ می دهد و چون روی کل داده ها train نمی شود می تواند از overfitting جلوگیری کند.
- **استفاده از دیتاست های بزرگ:** اگر مدل داده های زیادی برای آموزش داشته باشد میتواند راحت تر الگو ها را یاد بگیرد و کمتر به نویز توجه میکند.

4- **Underfitting** زمانی رخ میدهد که مدل ما خیلی ساده باشد و الگو های اصلی در داده ها را نتواند یاد بگیرد که در این صورت مدل هم در داده های آموزشی و هم در داده های تستی دقت پایینی دارد و در واقع می توان گفت مدل ما چیزی یاد نگرفته.

دلاین وقوع این امر میتواند از سادگی مدل باشد. اگر مدل لایه و پارامتر های خیلی کمی داشته باشد ممکن است منجر به Underfitting شود. و یا زمانی که مدل زودتر از موعد متوقف شود امکان این امر وجود دارد. و در نهایت اگر داده ها دارای نویز باشد و نماینده خوبی برای هر کلاس نباشد مدل ممکن است نتواند الگو های مفید را پیدا کند.

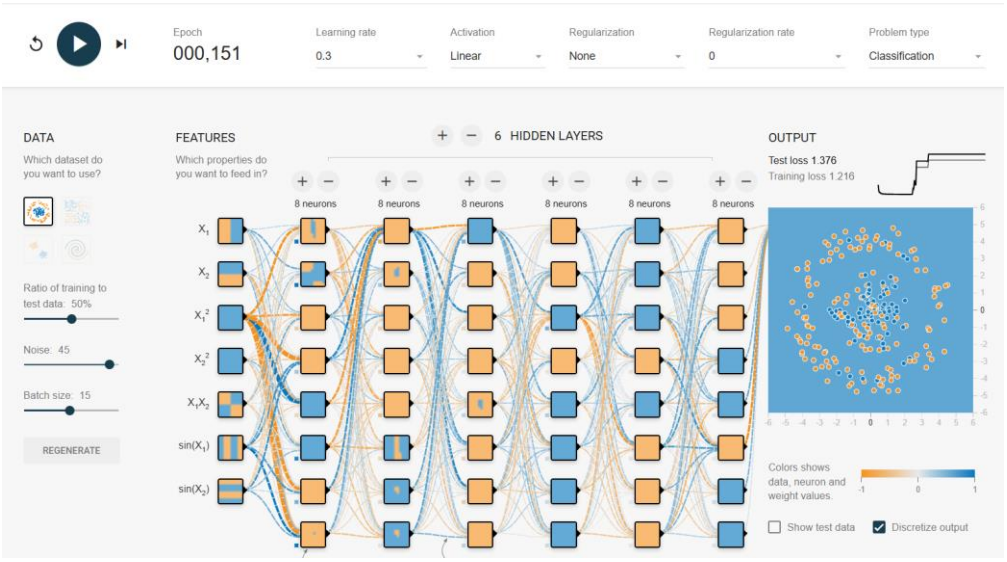
5- روش های جلوگیری از Underfitting:

- **افزایش پیچیدگی مدل:** یکی از راه های جلوگیری از این امر افزایش تعداد نورون ها و لایه ها است با این کار ظرفیت یادگیری مدل بالا می رود ولی از آن طرف ممکنه پیچیدگی خیلی زیاد دچار overfitting می شویم.
- **افزایش تعداد ایپوک ها:** یکی دیگر از روش های جلوگیری این است که دوره پردازش داده ها در مدل را افزایش دهیم تا مدل به خوبی الگو های موجود در داده ها را یاد بگیرد.
- **۴ .انتخاب ویژگی‌های مهم‌تر:** انتخاب ویژگی‌های مناسب یا حذف ویژگی‌های نویزی و غیرضروری می‌تواند به مدل کمک کند تا روی الگو های اصلی تمرکز کند.

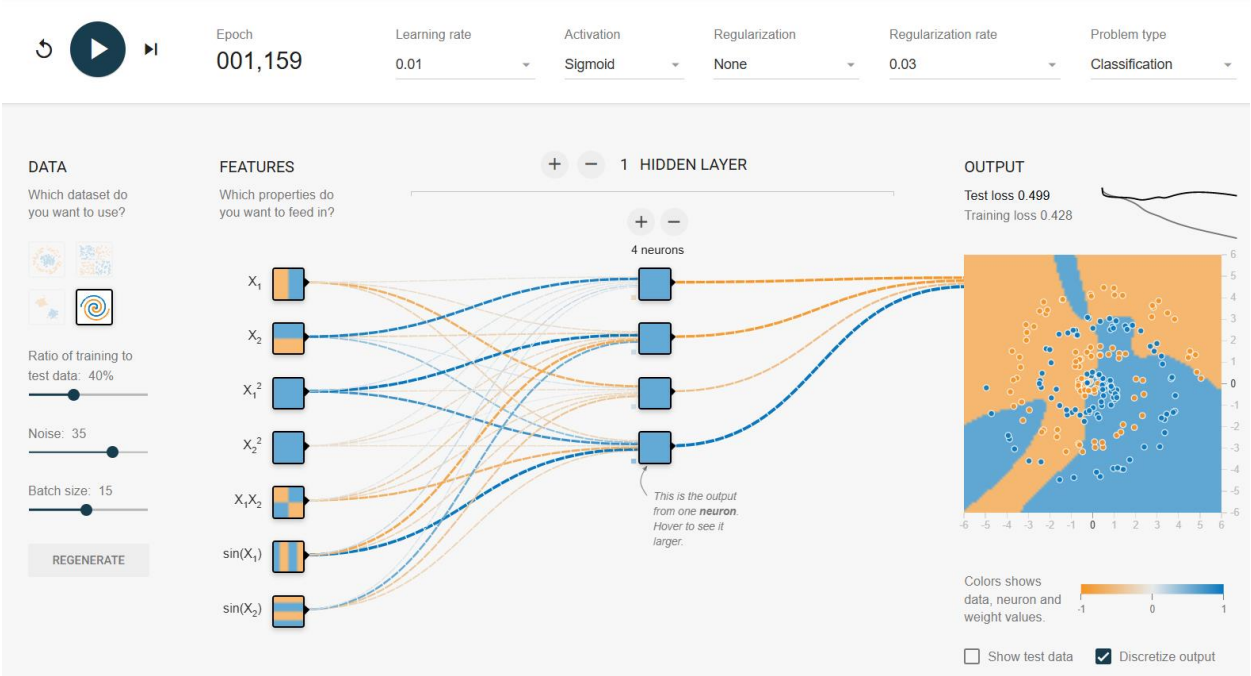
سوال دوم:

چند مدل برای underfit:

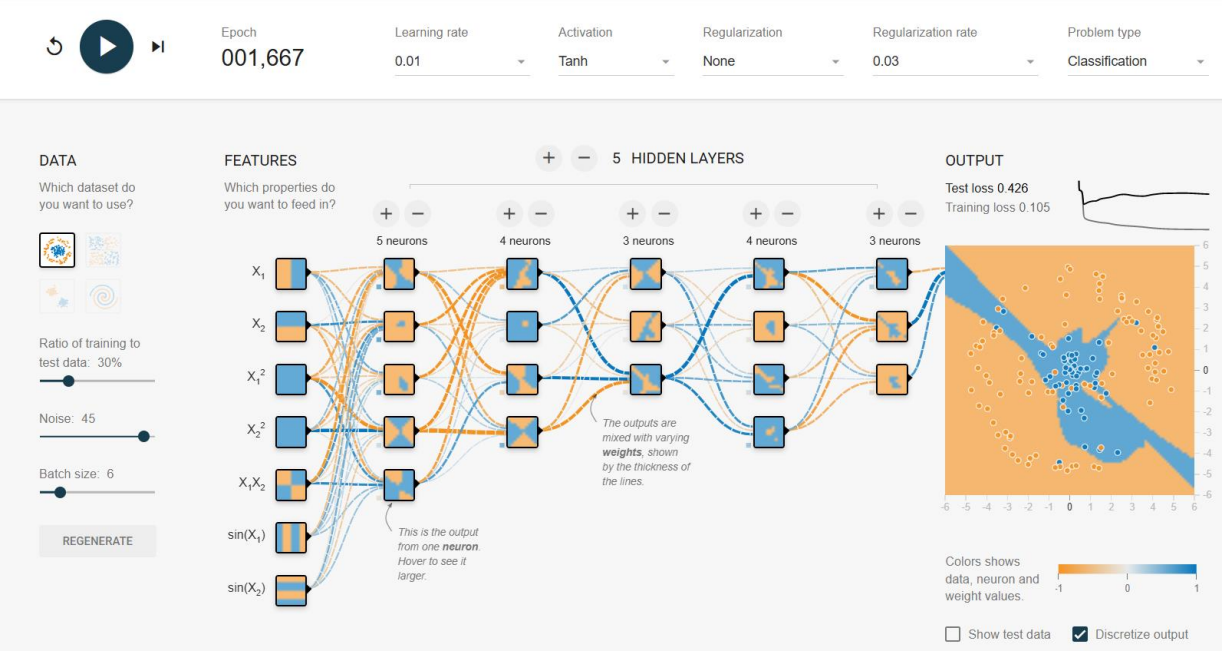
در این مثال با اینکه تعداد لایه ها و تعداد نورون ها زیاد است اما مدل ما خوب یاد نگرفته و هم خطای train و هم خطای test بالایی دارد.



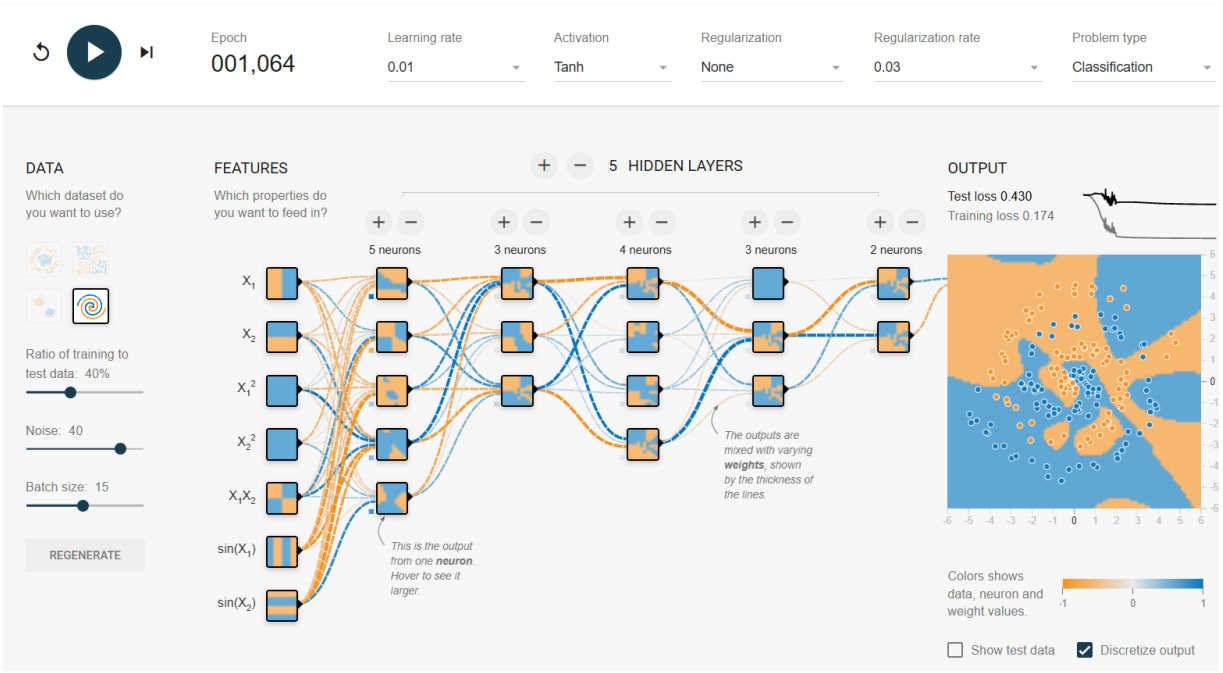
در اینجا هم با اینکه تعداد لایه ها و تعداد ورودی های مدل زیاد هست دچار خطای زیادی شدیم.

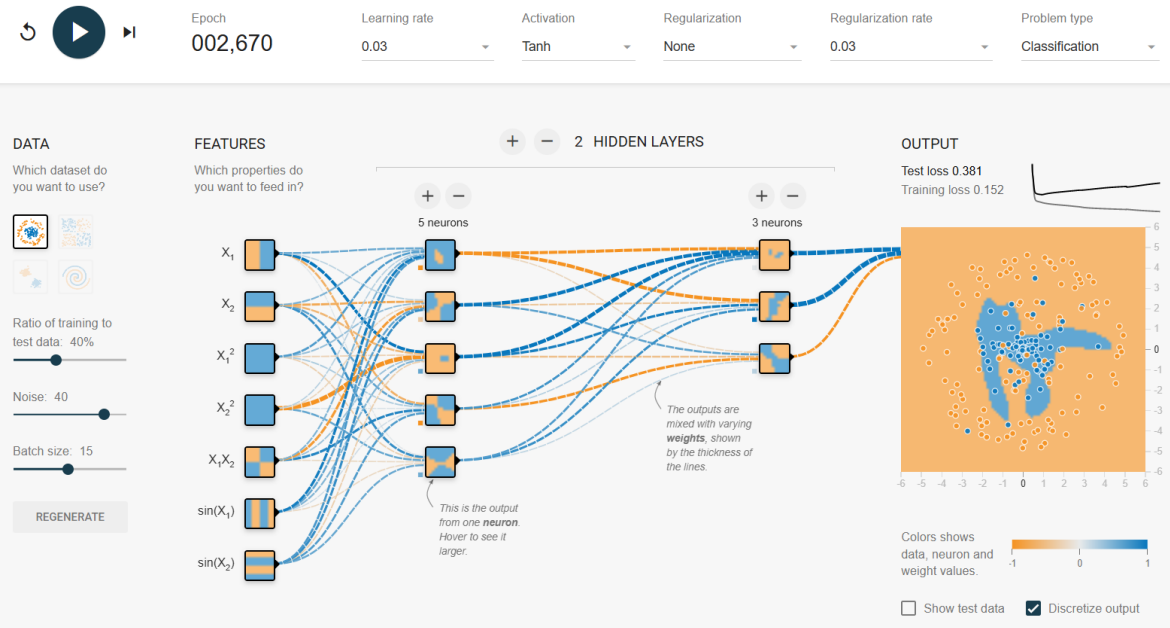


چند نمونه از overfitting:



در این مدل فقط داده های آموزشی را خوب یاد گرفته و overfit شده.





سوال سوم

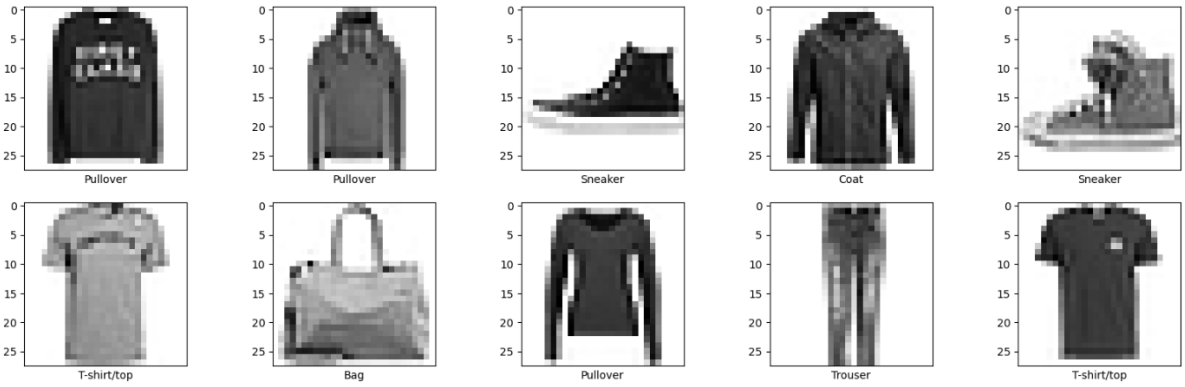
```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

ابتدا دیتاست مربوطه را لود میکنیم.

```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
plt.figure(figsize=(20, 6))
for i in range(10):
    index = np.random.choice(len(X_train),size=1)[0]
    plt.subplot(2, 5, i + 1)
    plt.xticks([])
    plt.imshow(X_train[index], cmap='binary')
    plt.xlabel(class_names[y_train[index]])
plt.show()
```

سپس برای رسم برخی از نمونه های دیتاست ابتدا اسم کلاس های مربوطه را به ترتیب می نویسم تا در کنار عکس نوع لباس را مشخص کنیم. پس از ساخت یک فیگور برای نمایش عکس ها یک حلقه فور ایجاد می کنیم تا 10 عکس را به صورت تصادفی نمایش دهیم.

از بین عکس های موجود یک ایندکس را به صورت رندوم انتخاب می کنیم و یک ساب پلات 2 در 5 برایش ایجاد می کنیم سپس مقادیر X آن عکس را برای ساخت عکس به تابع مربوطه می دهیم در نهایت برچسب آن را نیز در لیبل عکس نمایش می دهیم. این روند برای 10 عکس طی می شود که خروجی به شکل زیر است:



برای ساخت یک شبکه عصبی خطی ساده ما از کمترین پیچیدگی و تعداد لایه شروع کردیم که به صورت زیر است:

لایه ورودی: 784 تا (چون عکس ها 28 در 28 هستند --> 784 = 28 * 28)

لایه خروجی: 10 تا (چون تعداد کلاس های دیتاست ما 10 تا است)(از نوع Fully-connected)

تابع فعالساز لایه خروجی: برای مسائل multi-class بهتر است از تابع softmax استفاده کنیم. چرا که این تابع اعداد خروجی را به احتمال تبدیل می کند و به مدل کمک می کند تا کلاس مربوطه را با استفاده از بیشترین احتمال پیشبینی کند.

```
linear_model = Sequential()

linear_model.add(layers.Input(shape=(784,)))
linear_model.add(layers.Dense(10, activation='softmax'))

linear_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])

linear_model.summary()
```

در اینجا با استفاده از کتابخانه keras یک مدل خطی با تعداد لایه های گفته شده در بالا و تابع فعال ساز softmax ایجاد کردیم. در ادامه برای کامپایل مدل از بهینه ساز adam و تابع loss بالا استفاده می کنیم. خلاصه مدل ایجاد شده :

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape                   ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 10)                     │         7,850 │
└─────────────────────────────────┴────────────────────────────────┴───────────────┘
 Total params: 7,850 (30.66 KB)
 Trainable params: 7,850 (30.66 KB)
 Non-trainable params: 0 (0.00 B)
```

تعداد پارامتر ها : 7850

(784*10) + 10 = 7850

بایاس نورون ها = 10

برای شروع روند آموزش ابتدا داده ها را به 2 بعد تبدیل می کنیم و سپس فرایند نرمال سازی را با استفاده از کتابخانه standardscaler انجام می دهیم.

```
X_train = X_train.reshape(-1, 28*28)
X_test = X_test.reshape(-1, 28*28)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

در ادامه داده ها را با batch_size 32 و تعداد تکرار 10 روی مدل فیت کرده ایم.

```
linear_model.fit(X_train, y_train, epochs=10, batch_size=32)
```

حالا مقادیری که برای تست جدا کرده بودیم را توسط مدل پیش بینی می کنیم. در خط اول مقادیر همه ی 10 خروجی مدل برای هر x وجود دارد که ما باید نورونی که بیشترین احتمال را برای آن x در نظر گرفته را برای مقدار y در نظر بگیریم.

```
y_pred = linear_model.predict(X_test)
y_pred = tf.argmax(y_pred, axis=1).numpy()
```

در ادامه میزان دقت مدل و معیارهای ارزیابی را چاپ می کنیم.

```
accuracy = accuracy_score(y_test, y_pred)
classification_report = classification_report(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
print(classification_report)
```

خروجی نهایی:

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.8708 - loss: 0.3676
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8665 - loss: 0.3756
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8683 - loss: 0.3744
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8703 - loss: 0.3734
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8726 - loss: 0.3692
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8689 - loss: 0.3697
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8657 - loss: 0.3760
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8681 - loss: 0.3749
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8674 - loss: 0.3740
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8705 - loss: 0.3678
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
Test Accuracy: 0.8296
              precision    recall  f1-score   support

           0       0.80      0.75      0.78      1000
           1       0.96      0.96      0.96      1000
           2       0.73      0.71      0.72      1000
           3       0.82      0.84      0.83      1000
           4       0.69      0.81      0.75      1000
           5       0.96      0.86      0.91      1000
           6       0.62      0.53      0.57      1000
           7       0.84      0.97      0.90      1000
           8       0.93      0.93      0.93      1000
           9       0.95      0.93      0.94      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000

Confusion Matrix:
[[749   4  17  59   7   0 150   2  12   0]
 [  4 959   3  25   5   0   2   0   2   0]
 [ 22   4 708  15 161   3  78   2   7   0]
 [ 25  20  12 843  49   1  36   3  11   0]
 [  1   2  92  27 815   2  50   2   9   0]
 [  0   0   0   2   0 860   0 106   8  24]
 [119   5 127  46 144   1 534   0  23   1]
 [  0   0   0   0   0  10   0 971   0  19]
 [ 12   1  13   8   6   5  17   7 928   3]
 [  0   1   1   0   0  12   0  57   0 929]]
```

در ادامه به تست هایپرپارامتر های مختلف می پردازیم.

```
learning_rates = [0.01, 0.001, 0.0001]
batch_sizes = [16, 32, 64, 128]
epochs = [5, 10, 25, 50]
```

```
for lr in learning_rates:
    for batch_size in batch_sizes:
        for epoch in epochs:
            print(f"Training model with learning rate={lr}, batch size={batch_size}, epochs={epoch}")
            linear_model = Sequential()
            linear_model.add(layers.Input(shape=(784,)))
            linear_model.add(layers.Dense(10, activation='softmax'))
            linear_model.compile(optimizer = Adam(learning_rate=lr),
                                 loss='sparse_categorical_crossentropy',metrics=['accuracy'])

            linear_model.fit(X_train, y_train, epochs=epoch, batch_size=batch_size ,verbose=0)

            y_pred = linear_model.predict(X_test,verbose=0)
            y_pred = tf.argmax(y_pred, axis=1).numpy()
            accuracy = accuracy_score(y_test, y_pred)

            results.append({
                'learning_rate': lr,
                'batch_size': batch_size,
                'epochs': epoch,
                'accuracy': accuracy,
            })

            print(f"Accuracy: {accuracy:.4f}\n")
```

برای تست هایپرپارامتر های مختلف 3 حلقه ایجاد کردیم که برای هر یک از ترکیب های این هایپرپارامتر ها یک مدل همانند مدل قبل ایجاد کرده با این تفاوت که lr و batch_size و epochs در هر حلقه تغییر می کند.

دقت و هایپرپارمتر های هر مدل را درون متغیر results ذخیره کردیم.

```
best_result = max(results, key=lambda x: x['accuracy'])
print("\nBest Hyperparameters:")
print(f"Learning Rate: {best_result['learning_rate']}")
print(f"Batch Size: {best_result['batch_size']}")
print(f"Epochs: {best_result['epochs']}")
print(f"Accuracy: {best_result['accuracy']:.4f}")
```

در اینجا پس از اتمام حلقه فور نوبت به چاپ بهترین مدل می رسد که باید از متغیر results آن مدلی که میزان دقت آن از همه بیشتر است را پیدا میکنیم و مقادیر آن را نمایش می دهیم.

بخشی از خروجی کد :

```
Training model with learning rate=0.01, batch size=16, epochs=5
Accuracy: 0.7966

Training model with learning rate=0.01, batch size=16, epochs=10
Accuracy: 0.7871

Training model with learning rate=0.01, batch size=16, epochs=25
Accuracy: 0.7972

Training model with learning rate=0.01, batch size=16, epochs=50
Accuracy: 0.7953

Training model with learning rate=0.01, batch size=32, epochs=5
Accuracy: 0.8155
```

بهترین مدل:

```
Best Hyperparameters:
Learning Rate: 0.0001
Batch Size: 64
Epochs: 50
Accuracy: 0.8436
```

در اینجا مشاهده شد که با این که از هایپرپارمتر های مختلفی استفاده کردیم اما دقت مدل آن چنان تغییر نکرد. اما بنظر میرسد برای میزان دقت بیشتر باید تعداد لایه ها را بیشتر کنیم تا نتیجه بهتری دریافت کنیم.

سوال چهارم

```
my_colors = np.array([
    [255, 0, 0],
    [0, 255, 0],
    [0, 0, 255],
    [255, 255, 0]
])
my_colors = my_colors/255
```

برای این سوال ابتدا مقادیر اولیه را ست میکنیم که به شکل رو به رو است. سپس با تقسیم آن بر 255 داده ها را نرمال سازی میکنیم.

```
node_weights = np.array([[[100, 100, 100], [150, 150, 150]],
                         [[50, 50, 50], [200, 200, 200]]])
node_weights = node_weights / 255
```

در ادامه مقادیر وزن 4 نود را مشخص و سپس نرمالیزه میکنیم

```
learning_rate = 0.1
neighborhood_radius = 1
iter_count = 100
```

پس از آن هایپرپارامتر های مربوطه را ست می کنیم.

```
for iteration in range(iter_count):
    for input_color in my_colors:

        winner_index = None
        min_distance = float("inf")
        for x in range(2):
            for y in range(2):
                distance = np.linalg.norm(input_color - node_weights[x, y])
                if distance < min_distance:
                    min_distance = distance
                    winner_index = (x, y)
```

سپس یک حلقه فور با تکرار iter_count ایجاد می کنیم که مراحل زیر را انجام می دهد:

برای محسابه فاصله اقلیدسی هر رنگ تا هر نود به 3 حلقه نیاز داریم که حلقه اول برای رنگ های ورودی است و 2 حلقه دیگر برای نود ها که به صورت 2 بعدی هستند ایجاد کردیم. برای محاسبه فاصله اقلیدسی از نامپای کمک گرفتیم و ایندکس نودی که کمترین فاصله را برای هر رنگ دارد و فاصله آن ذخیره می کنیم.

```
for x in range(2):
    for y in range(2):
        dist_to_winner = np.sqrt((x - winner_index[0]) ** 2 + (y - winner_index[1]) ** 2)
        if dist_to_winner <= neighborhood_radius:

            influence = np.exp(-dist_to_winner / (2 * (neighborhood_radius ** 2)))
            node_weights[x, y] += learning_rate * influence * (input_color - node_weights[x, y])
```

پس از پیدا کردن نزدیک ترین نود به رنگ ورودی دوباره 2 حلقه برای تغییر وزن نود و همسایه هایش ایجاد کردیم. در ابتدا فاصله هر نود تا نود برنده را محاسبه می کنیم و اگر این فاصله از شعاع همسایگی کمتر باشد مقادیر وزن نود را با استفاده از learning-rate آپدیت می کنیم. این روند برای هر 4 رنگ تکرار می شود.

```
print("Final SOFM weights after training:")
node_weights = np.round((node_weights * 255),2)
print(node_weights)
```

در نهایت پس از اتمام حلقه مقادیر وزن هر نود را در 255 ضرب میکنیم تا به فرم قبلی برگردد، و مقادیر را با تقریب 2 رقم اعشار چاپ می کنیم که به شکل زیر است:

```
Final SOFM weights after training:
[[[125.2  133.09  50.93]
  [187.38 191.48   0.  ]]

 [[ 63.52  67.62 123.87]
  [131.49 131.49 123.51]]]
```

برای اینکه بفهمیم این مقادیر وزن پایدار هستند یا نه کد را با 1000 iter_count اجرا کردیم و خروجی باز هم به شکل قبل شد. پس می توانیم نتیجه بگیریم که وزن های نود پایدار شدند.

```
for input_color in my_colors:
    winner_index = None
    min_distance = float("inf")
    for x in range(2):
        for y in range(2):
            distance = np.linalg.norm(input_color - node_weights[x, y])
            if distance < min_distance:
                min_distance = distance
                winner_index = (x, y)
    print(input_color*255,"==>",winner_index)
```

برای اینکه بفهمم هر رنگ به چه نودی اختصاص داده شده با استفاده از کد بالا آن نودی که کمترین فاصله اقلیدسی با هر رنگ را دارد پیدا می کنیم و نمایش می دهیم که به شکل زیر است:

```
[255.   0.   0.] ==> (0, 0)
[  0. 255.   0.] ==> (0, 0)
[  0.   0. 255.] ==> (1, 0)
[255. 255.   0.] ==> (0, 1)
```

سوال پنجم

```
import numpy as np
import matplotlib.pyplot as plt

my_colors = np.random.randint(0,256,(256,3))
plt.figure(figsize=(15, 3))
plt.imshow([my_colors], aspect='auto')
plt.title("Randomly Generated 256 Colors")
plt.show()
```

برای حل این سوال ابتدا یک دیتاست رنگی ایجاد میکنیم که در آن 256 رنگ با فرمت RGB داریم که به صورت رندوم انتخاب شده است. در ادامه میتوانیم این رنگ های انتخاب شده را مشاهده کنیم.

در ادامه یک شبکه 10 در 10 برای som ایجاد می کنیم و برای وزن های آن یک رنگ با فرمت RGB به صورت رندوم ایجاد می کنیم. شکل نهایی وزن های شبکه به صورت 100تا(10*10) رنگ 3 تایی است.

```
grid_size = (10, 10)
neuron_weights = np.random.rand(10, 10, 3)

plt.figure(figsize=(5, 5))
plt.imshow(neuron_weights, aspect='auto')
plt.axis('off')
plt.title("Initial SOM Weights")
plt.show()
```

می توانیم به صورت بصری نتیجه کد بالا را مشاهده کنیم.

حال برای شروع روند آموزش هایپرپارامتر ها را ست می کنیم.

```
learning_rate = 0.1
epochs = 500
neighborhood_radius = 5
```

```
for epoch in range(epochs):
    current_lr = learning_rate * (1 - (epoch / epochs))
    np.random.shuffle(my_colors)
    for color in my_colors:
        distances = np.linalg.norm(neuron_weights - color, axis=2)
        winner_index = np.unravel_index(np.argmin(distances), grid_size)

        for i in range(10):
            for j in range(10):
                neuron_distance = np.sqrt((i - winner_index[0]) ** 2 + (j - winner_index[1]) ** 2)

                if neuron_distance <= neighborhood_radius:
                    influence = np.exp(-neuron_distance ** 2 / (2 * (neighborhood_radius ** 2)))
                    neuron_weights[i, j] += current_lr * influence * (color - neuron_weights[i, j])
```

در کد بالا یک حلقه فور به تعداد ایپوک ها داریم تا روند آموزش به این تعداد بار تکرار شود. در ادامه مقدار Learning-rate کاهش می یابد چرا که در فرایند som برای اینکه به یک نتیجه پایدار همگرا شویم باید مقدار تغییرات وزن نورون ها را کاهش دهیم. در هر روند آموزش داده ها شافل میشوند.

یک حلقه اینجا داریم که برای هر رنگ ورودی فاصله اقلیدسی آن رنگ را با همه نورون ها حساب می کند و در ادامه ایندکس آن نورونی که از همه به رنگ مورد نظر نزدیک تر است را ذخیره می کنیم.

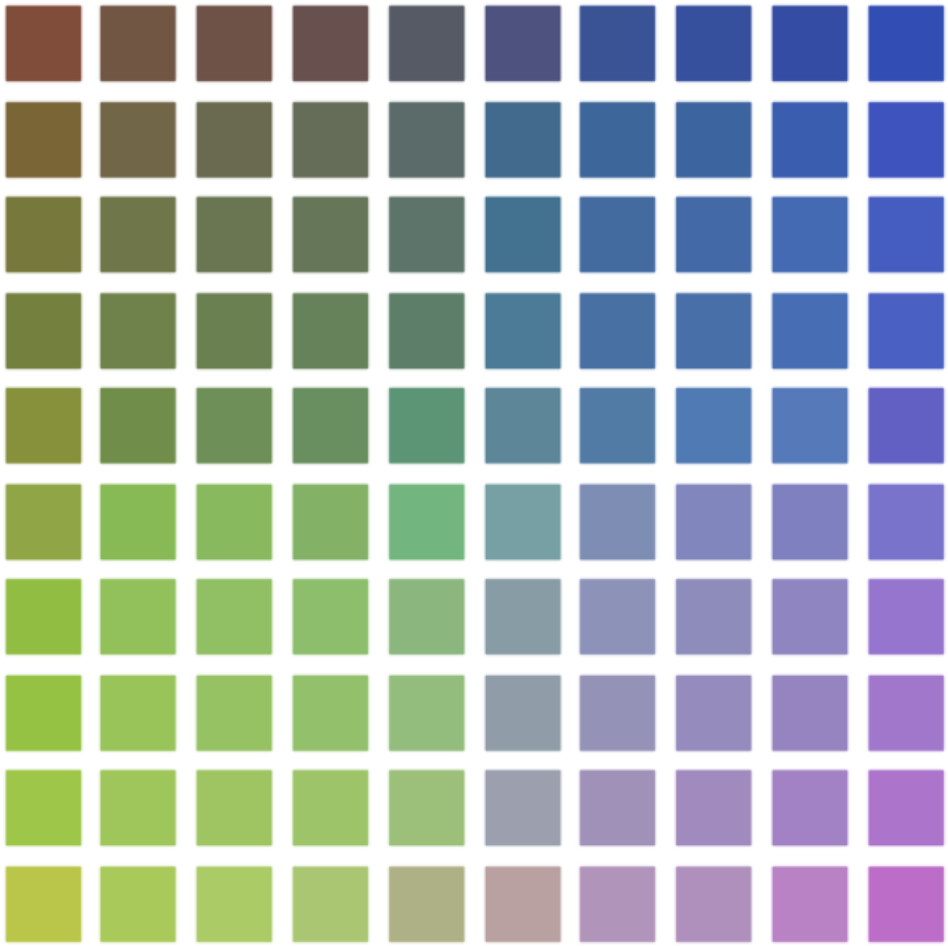
سپس درون 2 حلقه بعدی که 100 بار اجرا میشود(هر نورن یکبار) فاصله نورون تا فاصله نورون برنده محاسبه می شود و اگر از شعاع همسایگی کمتر یا مساوی باشد مقدار وزن آن نورون با استفاده از learning-rate و influence که فرمولی برای تعیین تاثیر میزان آپدیت وزن های همسایه ها است آپدیت می شود. این روند برای هر 100 نورون اتفاق میوفتد.

در نهایت تمامی این روند ها برای هر رنگ و به تعداد ایپاک ها تکرار میشود.

```
neuron_weights = neuron_weights / 255
plt.figure(figsize=(7, 7))
for i in range(10):
    for j in range(10):
        plt.plot(i, j, 's', color=neuron_weights[i, j], markersize=30)
plt.axis('off')
plt.show()
```

حال نوبت به نمایش وزن های شبکه میرسد که پس از این همه تکرار برای نمایندگی رنگ های ما انتخاب شده اند که به صورت زیر است.

خروجی نهایی



همان طور که قابل پیش بینی شد نورون هایی که رنگ های مشابهی دارند در کنار هم قرار گرفتند. اما این شبکه یکتا نیست و اگر دوباره کد را اجرا کنیم شبکه تغییر می کند و ممکن است جای رنگ ها عوض شد.

در خروجی بالا قابل مشاهده است که رنگ ها از تیره به روشن میرود و باعث ساخت گردنیت میشود که از رنگ زرد به سبز و از سبز به آبی می رود.

اگر آموزش SOM موفقیت‌آمیز بوده باشد، هر سلول باید یک رنگ میانگین یادگرفته شده از مجموعه داده‌ها را نشان دهد. و اگر مجموعه داده های رنگی ما شامل تنوع گسترده‌ای از رنگ‌ها باشد، SOM باید گردینت هایی را نشان دهد که به طور طبیعی از یک رنگ به رنگ دیگر جریان یابند.

محمد حقیقت - 403722042