

قسمت اول :

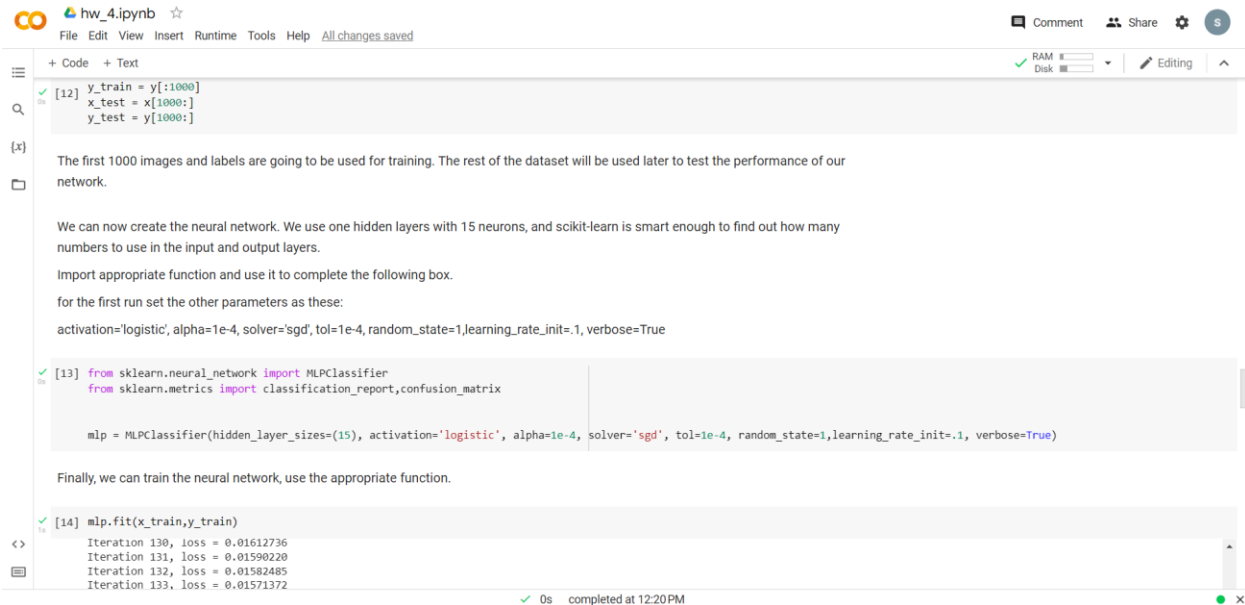
در بخش اول یک شبکه عصبی ایجاد شده و سعی شده با دقت بالا آن را آموزش داده و در نهایت مقایسه انجام شد. فراخوانی sklearn بارگذاری دیتایی شامل عکس از مجموعه اعداد دست نویس است. تصاویر اعداد از 0 تا 9 هست اما این تصاویر کیفیت بالایی ندارند و حتی انسان ها نیز ممکن است در تشخیص دچار اشتباه تشخیص شود. ما سعی می کنیم شبکه را به نحوی آموزش دهیم تا توانایی یادگیری و تشخیص اعداد را با دقت خوبی داشته باشد. داده های آموزش و تست جدا شده اند. 1770 داده داریم که 1000 تصویر برای آموزش و 770 تصویر برای تست در نظر گرفته شده است.

بررسی پارامترها و خروجی :

با تغییر تعداد پارامترهای نورون ها در لایه های پنهان و تغییر تعداد لایه های پنهان و استفاده از توابع فعال ساز متفاوت می توان به نتایج مختلفی رسید که در تصاویر زیر بخشی از آن ها نشان داده شده است در کل تا حدی بیشتر کردن نورون های لایه پنهان مفید خواهد بود و از جایی به بعد تاثیر ندارد. بهترین حالت بین پارامترهای تست شده تابع فعال ساز relu و 2 لایه نورون پنهان بود که به دقت 95% دست پیدا کرد. بیشتر از این مقدار به خاطر overfit شدن باعث کاهش دقت می شود.

تصاویر خروجی :

تعداد نورون 15 با تابع فعالساز logistic: به دقت 0.91 رسید.



```
[12] y_train = y[:1000]
      x_test = x[1000:]
      y_test = y[1000:]

The first 1000 images and labels are going to be used for training. The rest of the dataset will be used later to test the performance of our network.

We can now create the neural network. We use one hidden layers with 15 neurons, and scikit-learn is smart enough to find out how many numbers to use in the input and output layers.

Import appropriate function and use it to complete the following box.

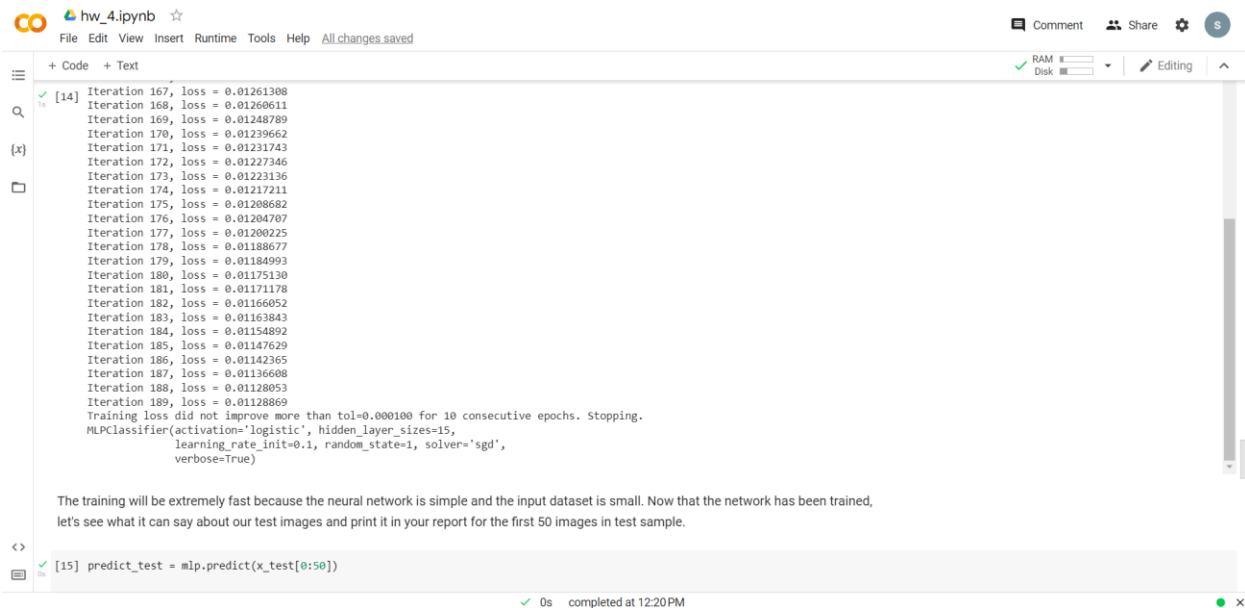
for the first run set the other parameters as these:
activation='logistic', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=.1, verbose=True

[13] from sklearn.neural_network import MLPClassifier
      from sklearn.metrics import classification_report, confusion_matrix

      mlp = MLPClassifier(hidden_layer_sizes=(15), activation='logistic', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=.1, verbose=True)

Finally, we can train the neural network, use the appropriate function.

[14] mlp.fit(x_train, y_train)
      Iteration 130, loss = 0.01612736
      Iteration 131, loss = 0.01590220
      Iteration 132, loss = 0.01582485
      Iteration 133, loss = 0.01571372
```



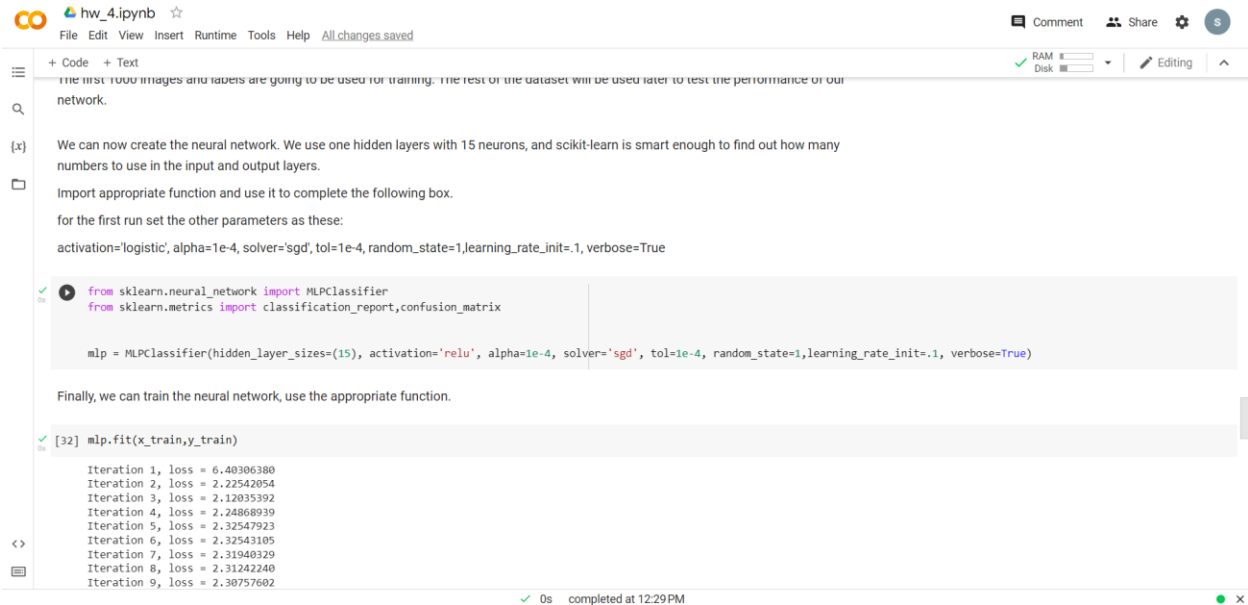
```
[14] Iteration 167, loss = 0.01261308
      Iteration 168, loss = 0.01260611
      Iteration 169, loss = 0.01248789
      Iteration 170, loss = 0.01239662
      Iteration 171, loss = 0.01231743
      Iteration 172, loss = 0.01227346
      Iteration 173, loss = 0.01223136
      Iteration 174, loss = 0.01217211
      Iteration 175, loss = 0.01208582
      Iteration 176, loss = 0.01204707
      Iteration 177, loss = 0.01200225
      Iteration 178, loss = 0.01188677
      Iteration 179, loss = 0.01184993
      Iteration 180, loss = 0.01175130
      Iteration 181, loss = 0.01171178
      Iteration 182, loss = 0.01166052
      Iteration 183, loss = 0.01163843
      Iteration 184, loss = 0.01154892
      Iteration 185, loss = 0.01147629
      Iteration 186, loss = 0.01142365
      Iteration 187, loss = 0.01136608
      Iteration 188, loss = 0.01128053
      Iteration 189, loss = 0.01128869
      Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
      MLPClassifier(activation='logistic', hidden_layer_sizes=15,
                    learning_rate_init=0.1, random_state=1, solver='sgd',
                    verbose=True)

The training will be extremely fast because the neural network is simple and the input dataset is small. Now that the network has been trained,
let's see what it can say about our test images and print it in your report for the first 50 images in test sample.

[15] predict_test = mlp.predict(x_test[0:50])
```


[illegible]

تعداد نورون 15 با تابع فعالساز relu: به دقت 0.099 رسید.



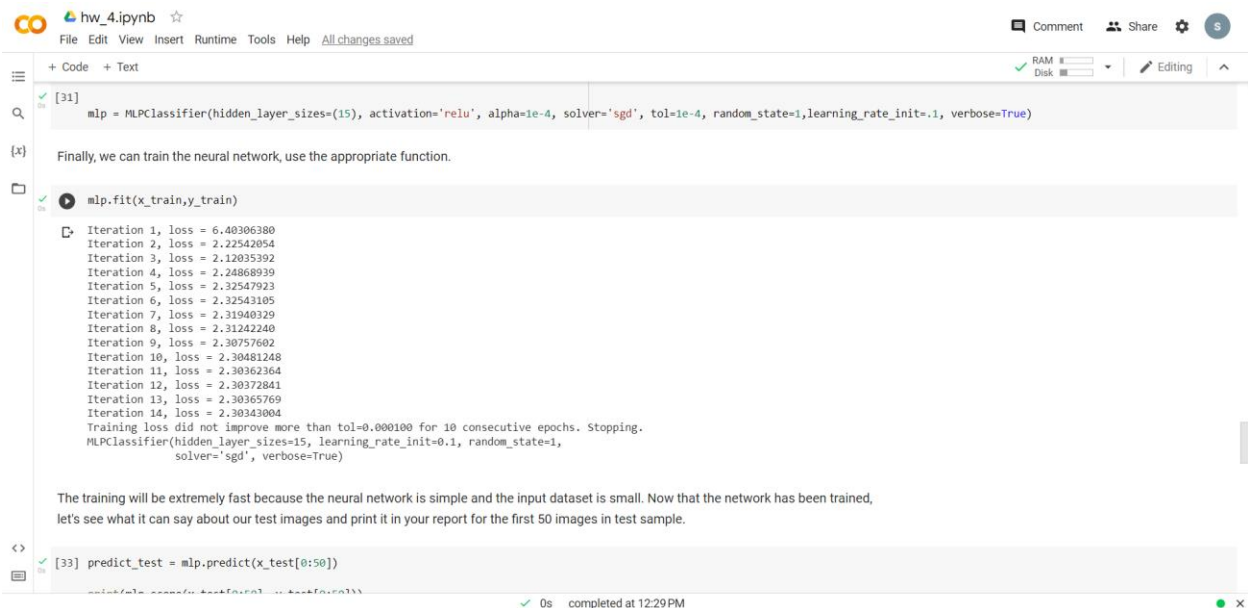
The screenshot shows a Jupyter Notebook titled 'hw_4.ipynb'. The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with icons for RAM, Disk, and Editing. The notebook content consists of several text cells and one code cell. The text cells describe the dataset split (first 1000 images for training, the rest for testing) and the parameters for the MLPClassifier. The code cell defines the classifier with the following parameters: hidden_layer_sizes=(15), activation='relu', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=.1, verbose=True. Below the code cell, the output of the fit method is shown, displaying the loss for each of the 9 iterations.

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix

mlp = MLPClassifier(hidden_layer_sizes=(15), activation='relu', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=.1, verbose=True)

mlp.fit(x_train, y_train)
```

Iteration 1, loss = 6.40306380
Iteration 2, loss = 2.22542054
Iteration 3, loss = 2.12035392
Iteration 4, loss = 2.24868939
Iteration 5, loss = 2.32547923
Iteration 6, loss = 2.32543105
Iteration 7, loss = 2.31940329
Iteration 8, loss = 2.31242240
Iteration 9, loss = 2.30757602



The screenshot shows the continuation of the Jupyter Notebook. The code cell defines the MLPClassifier with the same parameters as in the previous screenshot. The output of the fit method is shown, displaying the loss for each of the 14 iterations. The training process is complete, and the classifier is ready for testing. The text cells describe the training process and the next steps for testing the classifier on the test set.

```
mlp = MLPClassifier(hidden_layer_sizes=(15), activation='relu', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=.1, verbose=True)

mlp.fit(x_train, y_train)
```

Iteration 1, loss = 6.40306380
Iteration 2, loss = 2.22542054
Iteration 3, loss = 2.12035392
Iteration 4, loss = 2.24868939
Iteration 5, loss = 2.32547923
Iteration 6, loss = 2.32543105
Iteration 7, loss = 2.31940329
Iteration 8, loss = 2.31242240
Iteration 9, loss = 2.30757602
Iteration 10, loss = 2.30481248
Iteration 11, loss = 2.30362364
Iteration 12, loss = 2.30372841
Iteration 13, loss = 2.30365769
Iteration 14, loss = 2.30343004
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(hidden_layer_sizes=15, learning_rate_init=0.1, random_state=1, solver='sgd', verbose=True)

The training will be extremely fast because the neural network is simple and the input dataset is small. Now that the network has been trained, let's see what it can say about our test images and print it in your report for the first 50 images in test sample.

```
predict_test = mlp.predict(x_test[0:50])
```


hw_4.ipynb

File Edit View Insert Runtime Tools Help Saving...

Comment Share 8

+ Code + Text

RAM Disk Editing

Iteration 65, loss = 0.02684366
Iteration 66, loss = 0.02641765
Iteration 67, loss = 0.02602925
Iteration 68, loss = 0.02641758
Iteration 69, loss = 0.02611710
Iteration 70, loss = 0.02525083
Iteration 71, loss = 0.02486286
Iteration 72, loss = 0.02438828
Iteration 73, loss = 0.02419459
Iteration 74, loss = 0.02541653
Iteration 75, loss = 0.02863149
Iteration 76, loss = 0.02398655
Iteration 77, loss = 0.02419088
Iteration 78, loss = 0.02514810
Iteration 79, loss = 0.02422368
Iteration 80, loss = 0.02391635
Iteration 81, loss = 0.02363310
Iteration 82, loss = 0.02339628
Iteration 83, loss = 0.02305659
Iteration 84, loss = 0.03130819
Iteration 85, loss = 0.05275030
Iteration 86, loss = 0.03679163
Iteration 87, loss = 0.02590564
Iteration 88, loss = 0.02873643
Iteration 89, loss = 0.02562219
Iteration 90, loss = 0.02479340
Iteration 91, loss = 0.02308579
Iteration 92, loss = 0.02306781
Iteration 93, loss = 0.02354917
Iteration 94, loss = 0.02335379
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(activation='tanh', hidden_layer_sizes=15, learning_rate_init=0.1,
random_state=1, solver='sgd', verbose=True)

The training will be extremely fast because the neural network is simple and the input dataset is small. Now that the network has been trained,

0s completed at 12:27 PM

hw_4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share 8

+ Code + Text

RAM Disk Editing

1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 1 1 5 7 9 5 4 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

[1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 1 1 5 7 9 5 4 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0]

But can we be a bit more quantitative? We can compute the accuracy of the classifier, which the probability for a digit to be classified in the right category. Again, scikit-learn comes with a handy tool to do that. write the output in your report.

[29] predictions = mlp.predict(x_test)
predictions[0:50]

array([1, 4, 0, 5, 3, 6, 9, 6, 1, 7, 5, 4, 4, 7, 2, 8, 2, 2, 5, 7, 9, 5,
4, 4, 9, 0, 8, 9, 8, 0, 1, 2, 9, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4,
5, 6, 7, 8, 5, 0])

from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)

0.903387703889586

This number is the probability for the digits in the test sample to be classified in the right category.
We managed to get this accuracy with this very simple neural network. Not too bad!
However, this is only a first try. Now please change different parameters and discuss in your report about the effect of changing these parameters. (number of hidden layer neuruns, learning rate, the method of solver and activation function, ...)
Your grade for this homework mainly is dependent to your complete report.

0s completed at 12:27 PM

hw_4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

activation='logistic', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=1, verbose=True

[37]

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix

mlp = MLPClassifier(hidden_layer_sizes=(15), activation='identity', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=.1, verbose=True)
```

Finally, we can train the neural network, use the appropriate function.

[38]

```
mlp.fit(x_train, y_train)

Iteration 1, loss = 28.21374118
Iteration 2, loss = 14126855.27471844
Iteration 3, loss = 1589542228262309.000000000
Iteration 4, loss = 84521198197524177879040.000000000
Iteration 5, loss = 2066722309942148931984893171815552.000000000
Iteration 6, loss = 9465205723562376346556282046185996288000.000000000
Iteration 7, loss = 2095082623811985027477322840897465041193686532096.000000000
Iteration 8, loss = 1113489228936867806117037492732221308696402793168570941440.000000000
Iteration 9, loss = 2204136480404943019846718337812919127214481541811123364053804646400.000000000
Iteration 10, loss = 132117507324634162504056817611156422586818396042371640558621463141280645120.000000000
Iteration 11, loss = 2399394128225415272068039720979633762623496896765059933605825075866167384934973440.000000000
Iteration 12, loss = 12157514685122366376733431713667677810803679445989757783298864846187158438916059181745176576.000000000
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(activation='identity', hidden_layer_sizes=15,
              learning_rate_init=0.1, random_state=1, solver='sgd',
              verbose=True)
```

The training will be extremely fast because the neural network is simple and the input dataset is small. Now that the network has been trained, let's see what it can say about our test images and print it in your report for the first 50 images in test sample.

The screenshot shows a Jupyter Notebook window titled "hw_4.ipynb". The top bar includes standard file operations (File, Edit, View, Insert, Runtime, Tools, Help) and indicates "All changes saved".

The notebook contains three visible cells:

- A code cell with a comment "# [40]" followed by two arrays of handwritten digits from the MNIST dataset:

```
[1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 1 1 5 7 9 5 5 4 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0]
[1 4 0 5 3 6 9 6 1 7 5 4 4 7 2 8 2 2 5 7 9 5 4 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0]
```
- A text cell asking: "But can we be a bit more quantitative? We can compute the accuracy of the classifier, which the probability for a digit to be classified in the right category. Again, scikit-learn comes with a handy tool to do that. write the ouput in your report."
- A code cell containing two snippets of Python code:

```
# predicions = mlp.predict(x_test)
predicions[0:50]

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])

# from sklearn.metrics import accuracy_score
accuracy_score(y_test, predicions)
```

The output of the second code snippet is displayed below it: `0.10037641154328733`.

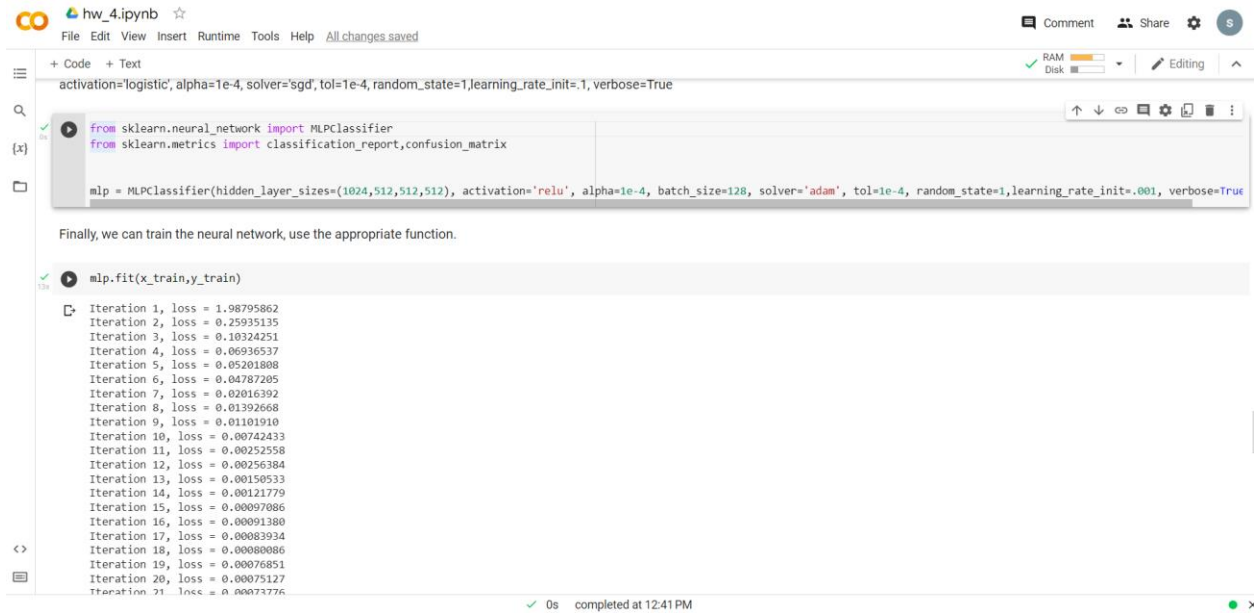
Below the code cells, there is explanatory text: "This number is the probability for the digits in the test sample to be classified in the right category." followed by a bold statement: "We managed to get this accuracy with this very simple neural network. Not too bad!".

Further text explains: "However, this is only a first try. Now please change different parameters and discuss in your report about the effect of changing these parameters. (number of hidden layer neurons, learning rate, the method of solver and activation function, ...)"

The final sentence states: "Your grade for this homework mainly is dependent to your complete report."

At the bottom status bar, it says "0s completed at 12:32 PM".

دو لایه پنهان نورون با تعداد های 1024، 512، 512، 512 و تابع فعالساز relu: به دقت 0.957 رسید که بیشترین دقت بود.



hw_4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

activation='logistic', alpha=1e-4, solver='sgd', tol=1e-4, random_state=1, learning_rate_init=1, verbose=True

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix

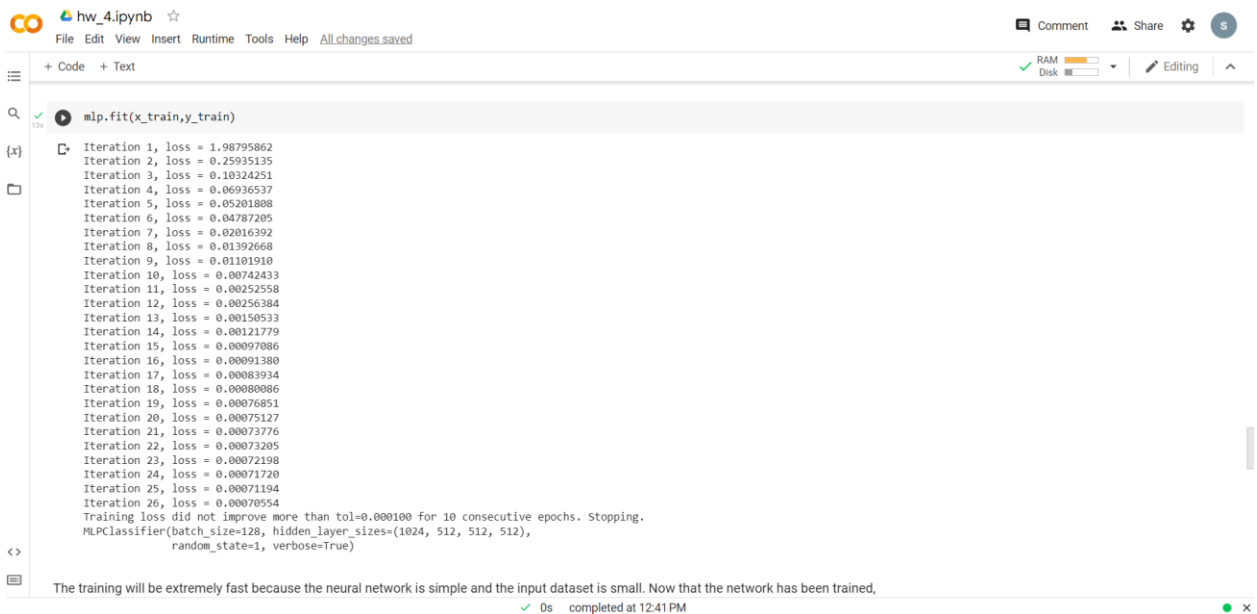
mlp = MLPClassifier(hidden_layer_sizes=(1024, 512, 512, 512), activation='relu', alpha=1e-4, batch_size=128, solver='adam', tol=1e-4, random_state=1, learning_rate_init=.001, verbose=True)
```

Finally, we can train the neural network, use the appropriate function.

```
mlp.fit(x_train, y_train)
```

```
Iteration 1, loss = 1.98795862
Iteration 2, loss = 0.25935135
Iteration 3, loss = 0.10324251
Iteration 4, loss = 0.06936537
Iteration 5, loss = 0.05201808
Iteration 6, loss = 0.04787205
Iteration 7, loss = 0.02016392
Iteration 8, loss = 0.01392668
Iteration 9, loss = 0.01101910
Iteration 10, loss = 0.00742433
Iteration 11, loss = 0.00252558
Iteration 12, loss = 0.00256384
Iteration 13, loss = 0.00150533
Iteration 14, loss = 0.00121779
Iteration 15, loss = 0.00097086
Iteration 16, loss = 0.00091380
Iteration 17, loss = 0.00083934
Iteration 18, loss = 0.00080086
Iteration 19, loss = 0.00076851
Iteration 20, loss = 0.00075127
Iteration 21, loss = 0.00073776
```

0s completed at 12:41 PM



hw_4.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
mlp.fit(x_train, y_train)
```

```
Iteration 1, loss = 1.98795862
Iteration 2, loss = 0.25935135
Iteration 3, loss = 0.10324251
Iteration 4, loss = 0.06936537
Iteration 5, loss = 0.05201808
Iteration 6, loss = 0.04787205
Iteration 7, loss = 0.02016392
Iteration 8, loss = 0.01392668
Iteration 9, loss = 0.01101910
Iteration 10, loss = 0.00742433
Iteration 11, loss = 0.00252558
Iteration 12, loss = 0.00256384
Iteration 13, loss = 0.00150533
Iteration 14, loss = 0.00121779
Iteration 15, loss = 0.00097086
Iteration 16, loss = 0.00091380
Iteration 17, loss = 0.00083934
Iteration 18, loss = 0.00080086
Iteration 19, loss = 0.00076851
Iteration 20, loss = 0.00075127
Iteration 21, loss = 0.00073776
Iteration 22, loss = 0.00073205
Iteration 23, loss = 0.00072198
Iteration 24, loss = 0.00071720
Iteration 25, loss = 0.00071194
Iteration 26, loss = 0.00070554
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(batch_size=128, hidden_layer_sizes=(1024, 512, 512, 512), random_state=1, verbose=True)
```

The training will be extremely fast because the neural network is simple and the input dataset is small. Now that the network has been trained,

0s completed at 12:41 PM

epoch	train_loss	valid_acc	valid_loss	dur
1	0.5880	0.9038	0.3164	144.5404
2	0.2031	0.9265	0.2364	140.6138
3	0.1415	0.9458	0.1708	140.6175
4	0.1093	0.9580	0.1374	140.5089
5	0.0894	0.9631	0.1229	140.6072
6	0.0746	0.9639	0.1133	140.4930
7	0.0634	0.9681	0.1032	140.6170
8	0.0544	0.9684	0.0997	140.4845
9	0.0472	0.9712	0.0928	140.5628
10	0.0404	0.9709	0.0943	140.5686

```
<class 'skorch.classifier.NeuralNetClassifier'>[initialized](
  module_=CNN(
    (network): Sequential(
      (0): Conv2d(1, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
      (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU()
      (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): ReLU()
      (6): Flatten(start_dim=1, end_dim=-1)
```

#channel for each layer \ optimizer

	SGD	Adam
(128, 128, 128)	97.0	10.4
(256, 256, 256)	97.4	10.4
(512, 512, 512)	97.38	10.4

epoch	train_loss	valid_acc	valid_loss	dur
1	0.5976	0.9063	0.3078	140.8147
2	0.1995	0.9246	0.2448	140.5222
3	0.1399	0.9468	0.1681	140.6707
4	0.1091	0.9588	0.1300	140.6153
5	0.0895	0.9648	0.1105	140.6230
6	0.0752	0.9675	0.1014	140.6110
7	0.0643	0.9702	0.0953	140.6292
8	0.0554	0.9719	0.0929	140.5753
9	0.0481	0.9727	0.0897	140.6479
10	0.0416	0.9724	0.0915	140.7525

```
<class 'skorch.classifier.NeuralNetClassifier'>[initialized](
  module_=CNN(
    (network): Sequential(
      (0): Conv2d(1, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
      (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU()
      (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): ReLU()
      (6): Flatten(start_dim=1, end_dim=-1)
```

```

10         0.0416         0.9724         0.0915  140.7525
<class 'skorch.classifier.NeuralNetClassifier'[initialized](
  module_=CNN(
    (network): Sequential(
      (0): Conv2d(1, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
      (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU()
      (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): ReLU()
      (6): Flatten(start_dim=1, end_dim=-1)
      (7): Linear(in_features=401408, out_features=512, bias=True)
      (8): ReLU()
      (9): Linear(in_features=512, out_features=10, bias=True)
    )
  ),
)

```

Write down the **validation accuracy** of your model under different hyperparameter settings.

#channel for each layer	validation accuracy
(128, 128, 128)	97.47

- Code + Text

Write down the **validation accuracy** of your model under different hyperparameter settings.

#channel for each layer	validation accuracy
(128, 128, 128)	97.47
(128, 256, 512)	97.15
(256, 256, 256)	97.48
(256, 512, 1024)	97.40
(512, 512, 512)	97.30
(512, 1024, 2048)	97.48

Your Answer: 97.67%

What can you conclude for the design of CNN structure?

Your Answer: سه لایه پیشی با 512، 1024، 2048 نرون و همچنین 256، 256، 256 نرون، بالاترین دقت را داشتند.

• Subtask 2-1: Completing the Table

We have provided the following table for different combinations of optimizers and learning rate, please write down the **validation accuracy** of your model with different optimizers and learning rates.

	0.1	0.01	0.001	0.0001
SGD	98.3	97.5	93.9	64.71
Adam	10.2	10.4	98.38	98.5
RMSprop	9.8	10.4	98.2	98.55

- **Subtask 2-2: Explaining your Observations**

Based on your results, briefly explain your observations, e.g., which optimizer works the best, what is the optimal learning rate for each optimizer?

Your Answer: RMSprop از الگوریتم ها RMSprop و Adam بهترین نتیجه را به دست آورد برای برخی از الگوریتم ها نرخ یادگیری هرچه کمتر بود نتیجه بهتر شد مثل SGD برعکس این موضوع برقرار بود مثل



validation Accuracy

- **Subtask 3-2: Answering the Question**

Is it always better to train a model for more epoches? How can we decide when should we stop training?

Your Answer: نشود overfit بیشتر بشد نتیجه بهتر است تا جایی که epoch در کل هر چه تعداد: