1η Εργασία Νευρωνικών Δικτύων

Περιεχόμενα

1. Εισαγωγή 1
2. Κώδικας 1
3. Ανάλυση Κώδικα 2
4. Δοκιμές 4
5. Σύγκριση με kNN και NCC 9

1. Εισαγωγή

Η εργασία που εκπονήθηκε, πραγματεύεται την υλοποίηση ενός νευρωνικού δικτύου πολυστρωματικού perceptron το οποίο εκπαιδεύεται πάνω στην βάση δεδομένων MNIST και αποσκοπεί στην εύρεση κλάσης αριθμών που γράφονται με το χέρι.

Ο κώδικας υλοποιήθηκε σε γλώσσα προγραμματισμού python σε περιβάλλον VScode και εμπεριέχει συναρτήσεις από το API **keras** τις βιβλιοθήκες **numpy**, **sklearn**, **pandas**, **matplotlib**, **sys**.

2.Κώδικας

```
from tensorflow import keras
from keras import datasets
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
import pandas as pd
from keras import models, layers
from matplotlib import pyplot as plt
import numpy as np
import sys
sys.stdout = open(r"C:\Users\Koukas\Desktop\Project 1 logs\test1.txt", "w")
num_features = 28*28
num_classes = 10

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

```
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)

x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
x_train, x_test = x_train/255., x_test/255.
model = keras.Sequential([layers.Flatten(input_shape=(num_features,)),
                          layers.Dense(128, activation='relu'),
                          layers.Dense(num_classes)])

model.compile(optimizer='sgd',
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

model.summary()

history = model.fit(x_train,y_train,
validation_data = (x_test, y_test),
epochs = 5)

plt.figure(1)
plt.plot(history.history['accuracy'], label = 'train')
plt.plot(history.history['val_accuracy'], label = 'test')
plt.legend()
plt.title('Performance on training and validation sets')
plt.show()
sys.stdout.close()
```

3. Ανάλυση Κώδικα

```
(x_train,y_train), (x_test,y_test) = datasets.mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)
```

Σε αυτό το σημείο του κώδικα κατεβάζουμε το dataset **MNIST**, και εν συνεχεία αλλάζουμε τις διαστάσεις των πινάκων των samples έτσι ώστε να είναι πίνακες από vectors (που είναι τα samples) για να μπορούμε να τα προωθήσουμε στις επόμενες συναρτήσεις μας.

```
x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
```

```
x_train, x_test = x_train/255., x_test/255.
```

Είναι σημαντικό να κανονικοποιήσουμε τα δεδομένα μας στο πεδίο [0,1] έτσι ώστε να μην έχουμε λανθασμένους υπολογισμούς των βαρών του νευρωνικού δικτύου από τυχόντα outliers. Επειδή ξέρουμε ότι οι gray-scale εικόνες που έχουμε παίρνουν τιμές σε κάθε pixel από 0 έως 255 τότε διαιρούμε όλα τα pixel με των αριθμό 255 κάτι που βγαίνει από τον τύπο:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
model.compile(optimizer='sgd',
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])

model.summary()

history = model.fit(x_train,y_train,
validation_data = (x_test, y_test),
epochs = 100)
```

Σε αυτό το σημείο του κώδικα καλούμε την συνάρτηση model.compile() στην οποία διαμορφώνουμε το νευρωνικό δέντρο που θέλουμε να κατασκευάσουμε και επιλέγουμε στον κώδικα μας τα arguments:

`optimizer`, `loss`, `metrics` όπου με το πρώτο επιλέγουμε μεθόδους που χρησιμοποιούνται για την αλλαγή των χαρακτηριστικών του νευρωνικού μας δικτύου, όπως τα βάρη και ο ρυθμός εκμάθησης, προκειμένου να μειωθούν οι απώλειες, το δεύτερο την αντικειμενική μας συνάρτηση και με το τρίτο την μετρική που θέλουμε να ακολουθήσουμε.

```
model.summary()
```

Τυπώνει στο console log μία περίληψη του νευρωνικού που κατασκευάσαμε.

```
history = model.fit(x_train,y_train,
validation_data = (x_test, y_test),
epochs = 5)
```

Και εδώ γίνεται η εκπαίδευση του μοντέλου για δοσμένες εποχές που του δίνουμε πάνω στο training και test set μας.

```
plt.figure(1)
plt.plot(history.history['accuracy'], label = 'train')
plt.plot(history.history['val_accuracy'], label = 'test')
plt.legend()
plt.title('Performance on training and validation sets')
plt.show()
```

Κατασκευάζουμε κάποια διαγράμματα της ακρίβειας αλλά και της 'val_accuracy' η οποία είναι η ακρίβεια των προβλέψεων σε ένα τυχαία χωρισμένο validation set μετά από κάθε επανάληψη.

4. Δοκιμές

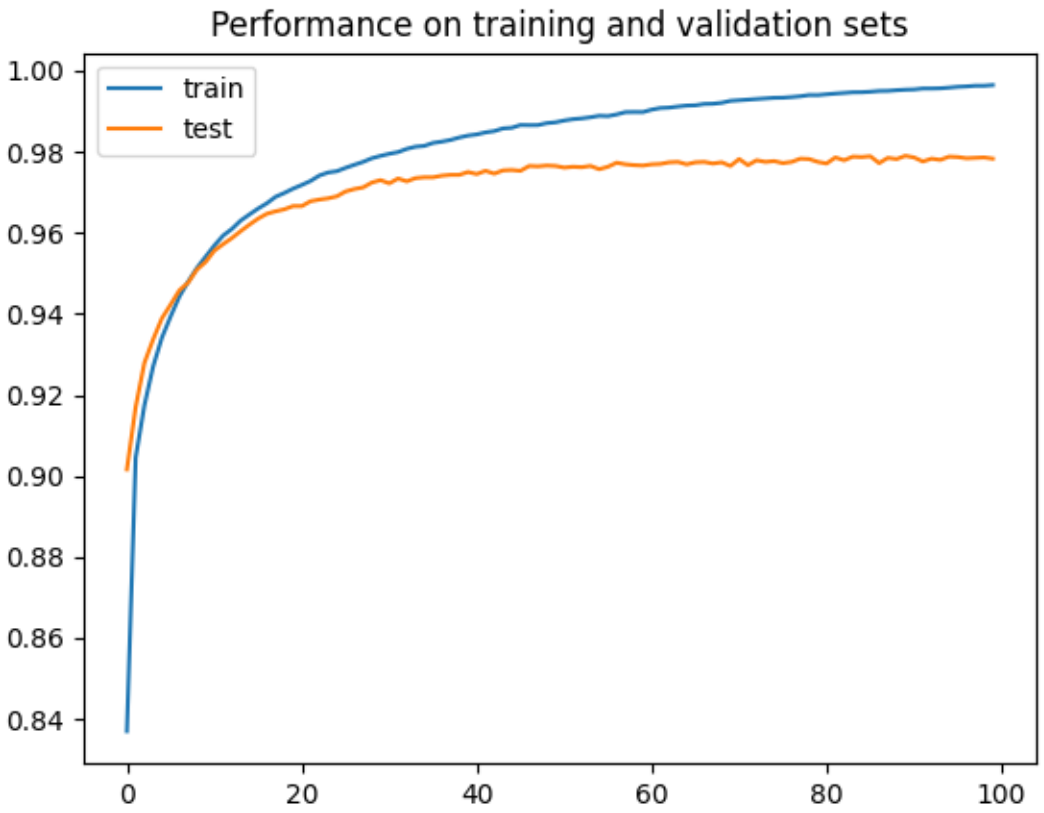
Στην πρώτη μας προσπάθεια θέτουμε τις εξής παραμέτρους:

```
layers.Dense(128, activation='relu')

optimizer='sgd'

loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)

epochs = 100
```

Επιλέξαμε την SparseCategoricalCrossentropy

Το batch size δεν το πειράζουμε οπότε είναι 32.

Το log αυτού του run μαζί με τα διαγράμματα παρατίθενται παρακάτω:



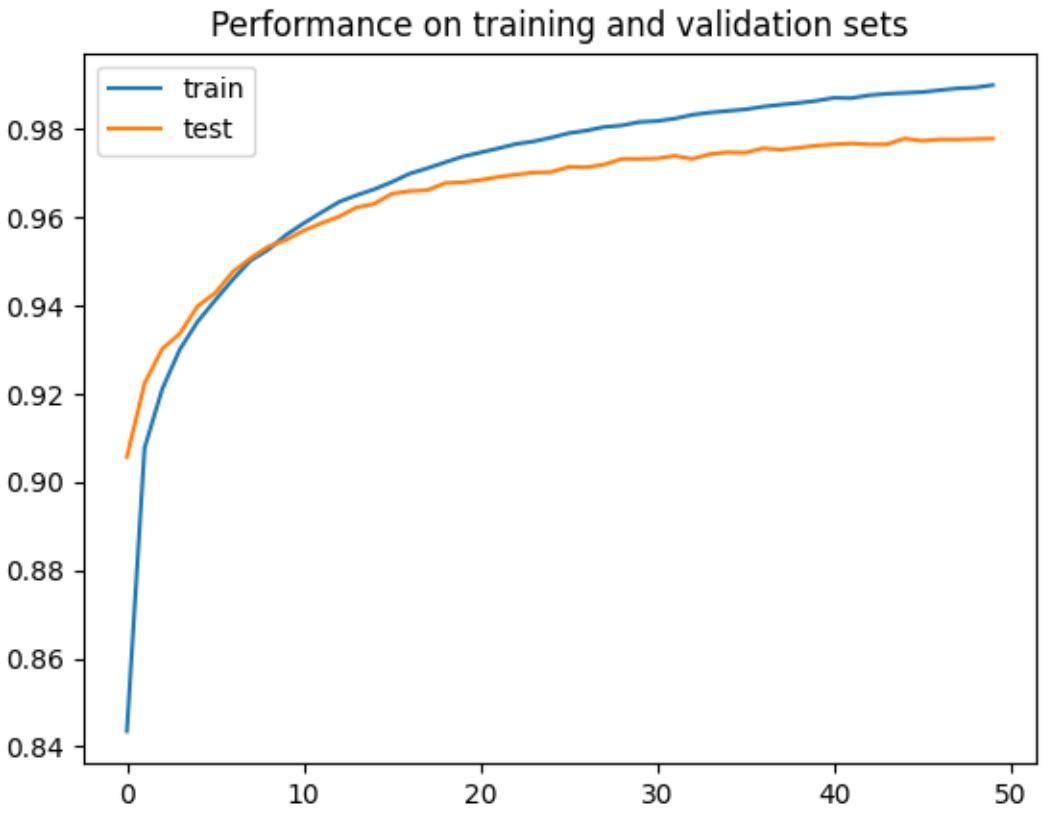
Μετά από 20-25 εποχές παρατηρούμε σύγκλιση της ακρίβειας στο test set με το μέτρο της να σταθεροποιείται περί το 0.97.

Εν συνεχεία δοκίμαζουμε τις εξής παραμέτρους:

```
layers.Dense(256, activation='relu')
epochs = 50
```

Αυξάνουμε δηλαδή τον αριθμό των κρυφών layers και μειώνουμε τις εποχές:

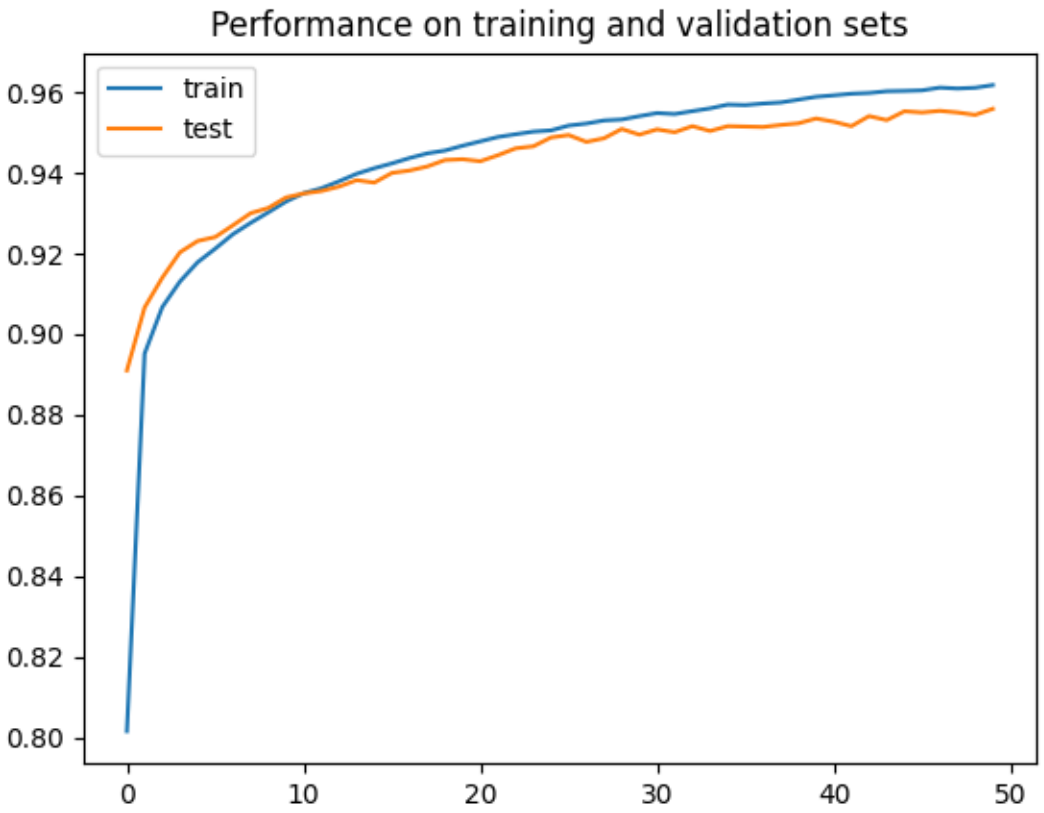
Τα αποτελέσματα:



Δεν βλέπουμε μεγάλη διαφορά στα αποτελέσματα μας εκτός του ότι συγκλίνει σε λιγότερες εποχές η ακρίβεια.

Εάν μειώσουμε όμως τα hidden layers σε 16 θα έχουμε γρηγορότερη εκπαίδευση αλλά όχι τόσο μεγάλη ακρίβεια.

Όπως φαίνεται και εδώ:



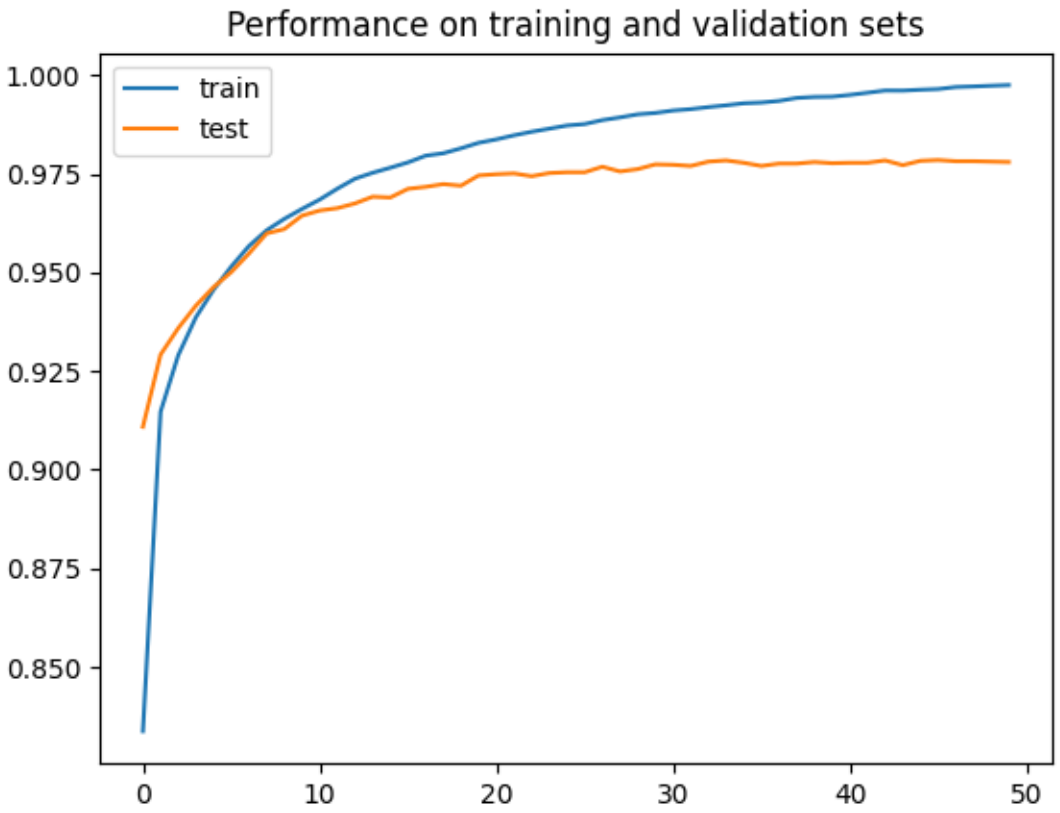
Στον αρχικό μας αλγόριθμο,

```
layers.Dense(128, activation='relu')

optimizer='sgd'

loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)

epochs = 50
```

προσθέτω ακόμα ένα στρώμα 128 layers:

```
layers.Dense(128, activation='relu')
```

Εδώ λόγω των λιγότερων epochs έχουμε γρήγορη εκτέλεση αλλά λόγω των 2 στρωμάτων πολύ μεγάλη ακρίβεια:



Σε μία ακόμη δοκιμή θα παραμετροποιήσουμε το δίκτυο μας ως εξής:

```
optimizer='Adam'
```

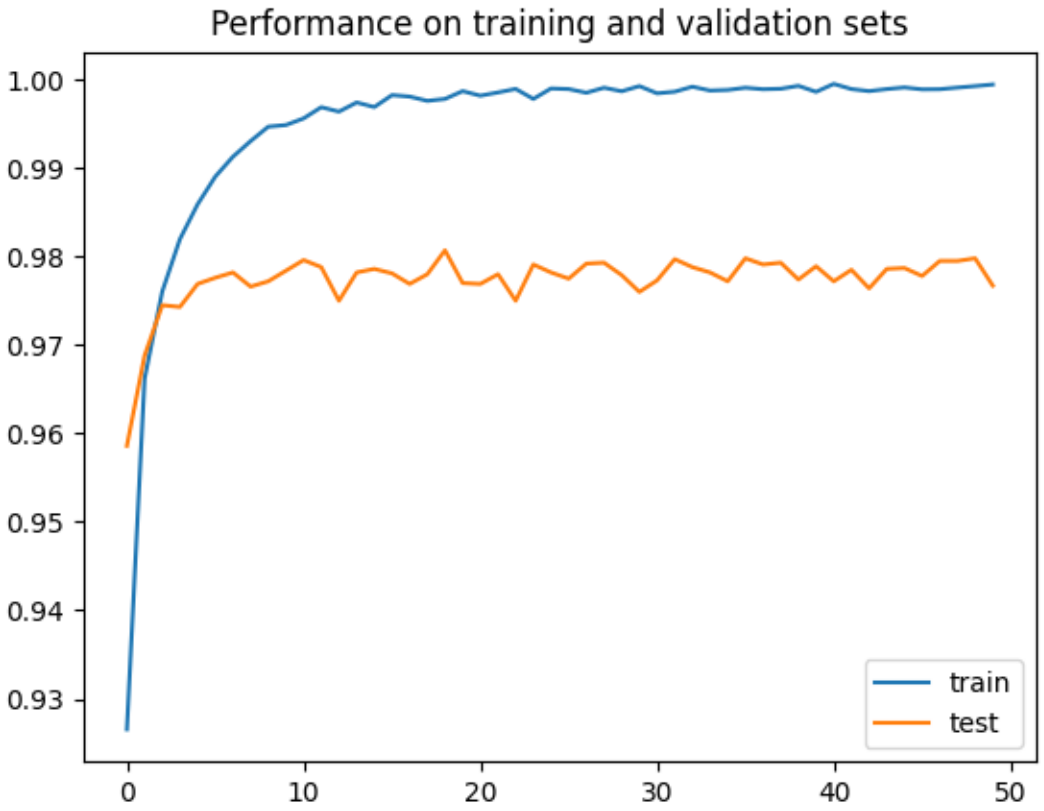
με ένα στρώμα 128 layers

```
layers.Dense(128, activation='relu')
```

και 50 εποχές

```
epochs = 50
```

με αποτελέσματα:



Κάτι που μας δείχνει ότι ακρίβεια συγκλίνει σε πολύ λίγες εποχές σε μία πολύ υψηλή τιμή με τον κώδικα να τρέχει σε αποδεκτό χρόνο.

5. Σύγκριση με kNN και NCC

Κώδικας ενδιάμεσης εργασίας:

```
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from sklearn.metrics import accuracy_score
import time
```

```
clf1 = KNeighborsClassifier(1)
clf2 = KNeighborsClassifier(3)
CentroidClassifier = NearestCentroid()




(x_train,y_train), (x_test,y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)

start_time1 = time.time()
clf1.fit(x_train,y_train)
pred1 = clf1.predict(x_test)
print("Execution Time for kNN-1 Classifier --- %s seconds ---" % (time.time() -
start_time1))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred1))

start_time2 = time.time()
clf2.fit(x_train,y_train)
pred2 = clf2.predict(x_test)
print("Execution for kNN-3 Classifier --- %s seconds ---" % (time.time() -
start_time2))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred2))


start_time3 = time.time()
CentroidClassifier.fit(x_train,y_train)
pred3 = CentroidClassifier.predict(x_test)
print("Execution Time for Centroid Classifier --- %s seconds ---" % (time.time()
- start_time3))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred3))

#Χρησιμοποιήσαμε την μετρική Accuracy για να εκτιμήσουμε την απόδοση των
Classifier μας στο συγκεκριμένο dataset.
#Με την βοήθεια της βιβλιοθήκης time καταφέραμε να μετρήσουμε τον Χρόνο εκτέλεσης
του κάθε ένα από τους Classifiers.
#Αφού δεν μπορούμε να εισάγουμε στο fit 3D data πρέπει να τα ανασχηματίσουμε σε
διαστάσεις που μας βολεύουν.
```

Με output στην κονσόλα:

Execution Time for kNN-1 Classifier --- 21.646045684814453 seconds ---

Accuracy for kNN1 --- 0.9691 ---

Execution for kNN-3 Classifier --- 18.950039625167847 seconds ---

Accuracy for kNN1 --- 0.9705 ---

Execution Time for Centroid Classifier --- 0.12199664115905762 seconds ---

Accuracy for kNN1 --- 0.8203 ---

Παρατηρούμε ότι ο Centroid Classifier δεν τα καταφέρνει καλά στο να ανεβάσει την μετρική σε επιθυμητά σημεία αλλά ο kNN καταφέρνει σε πολύ μικρό χρόνο να σχηματίσει ένα μοντέλο που έχει αρκετά μεγάλη ακρίβεια στο MNIST dataset, εφάμιλλη αυτής που δίνουν τα νευρωνικά δίκτυα που δοκιμάσαμε προηγουμένως. Την καλή απόδοση του kNN την αποδίδουμε στην χαμηλή πολυπλοκότητα του MNIST Dataset σε άλλα datasets περιμένουμε μεγαλύτερη απόδοση από το νευρωνικό μοντέλο.

Τα log files του κάθε run εμπεριέχονται στο zip μαζί με το αρχείο py.

Κούκας Γεώργιος 9486
