1η ΕΡΓΑΣΙΑ-«Νευρωνικά Δίκτυα-Βαθιά Μάθηση»

Multi-Layer Perceptron Neural Network

ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Ιωαννίδης Χρήστος, τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΑΕΜ 9397

Εισαγωγή

Η παρούσα εργασία είναι μια αναφορά για τη δημιουργία, εκπαίδευση και τον έλεγχο ενός νευρωνικού δικτύου στην αναγνώριση hand-written δεκαδικών ψηφίων (0,1,..,9) που εμφανίζονται σε εικόνες που έχουν ληφθεί από τη βάση δεδομένων MNIST. Οι εικόνες, σε αριθμό 70000 και διαστάσεων 28x28 pixel, χωρίζονται σε σετ εκπαίδευσης (60000) και ελέγχου (10000) ώστε να υπολογιστούν μετρικές απόδοσης του νευρωνικού δικτύου.

Ο κώδικας υλοποίησης της εργασίας είναι σε γλώσσα Python, και χρησιμοποιήθηκαν οι βιβλιοθήκες Keras και Tensorflow.

Σκοπός της εργασίας είναι η σύγκριση της απόδοσης του αλγορίθμου back-propagation, με δοκιμές για διαφορετικές παραμέτρους (αριθμό εποχών, αριθμό νευρώνων και hidden layers, συνάρτηση ενεργοποίησης και optimizer) και της απόδοσης των αλγορίθμων πλησιέστερου γείτονα και πλησιέστερου κέντρου κλάσης.

Ο τρόπος λειτουργίας του δικτύου περιλαμβάνει την αναγνώριση των ψηφίων μέσω της φωτεινότητας του κάθε ενός από τα 784 pixels, το κάθε ένα στον άξονα άσπρο-γκρι-μαύρο και φυσικά της θέσης του στον πίνακα 28x28.

1. Multi-Layer Perceptron with Back-Propagation

Πρώτο βήμα, στο συγκεκριμένο σημείο, είναι να φορτώσουμε τα δεδομένα, τα οποία υπάρχουν ήδη στη βιβλιοθήκη keras, και να τα χωρίσουμε σε σετ εκπαίδευσης και ελέγχου. Στις μεταβλητές x_train, x_test αποθηκεύονται οι βαθμοί φωτεινότητας του κάθε pixel (28x28) της κάθε εικόνας ψηφίου σε έναν 3 dimensional array (60000x28x28/10000x28x28), ενώ τα y_train, y_test είναι τα labels, οι πραγματικές τιμές των ψηφίων.

Για να έχει πολύ μικρότερη υπολογιστική πολυπλοκότητα το πρόβλημα είναι σημαντικό να κάνουμε reshape στους πίνακες των δεδομένων, έτσι ώστε αντί να έχουμε κάθε εικόνα σαν 28x28 array, να την έχουμε σαν έναν μονοδιάστατο πίνακα 784x1, δηλαδή πρακτικά ένα διάνυσμα, και επίσης να κάνουμε κανονικοποίηση των αριθμών που αφορούν την φωτεινότητα (0-255 σε

0-1).

```
import tensorflow as tf
import matplotlib.pyplot as mplt
import numpy as np
import pandas as pd
import timeit
#Load the data, and convert them from 3D to 2D arrays
(x_train, y_train), (x_test, y_test) =tf.keras.datasets.mnist.load_data()
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
#Normalization
x_train=x_train/255
x_test=x_test/255
```

Επόμενο βήμα είναι ο σχεδιασμός του μοντέλου. Σε αυτό το σημείο θα γίνουν δοκιμές για μέτρηση της απόδοσης με διαφορετικές παραμέτρους.

Αρχικά δοκιμάζουμε με 1 hidden layers με 256 νευρώνες, 10 εποχές, adam optimizer, συνάρτηση ενεργοποίησης την relu στο hidden layer και softmax, όπως προτείνεται στη βιβλιογραφία, για output layer και σαν loss function sparse categorical crossentropy, όπως φαίνεται παρακάτω:

```
mlpmodel=tf.keras.models.Sequential()
#mlpmodel.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

mlpmodel.compile(optimizer="adam", loss='sparse_categorical_crossentropy', metrics=['accuracy'] )
t_start=timeit.default_timer()
mlpmodel.fit(x_train, y_train, epochs=10)

val_loss, val_acc=mlpmodel.evaluate(x_test,y_test)
t_stop=timeit.default_timer()
print("loss: ", val_loss, " accuracy: ", val_acc)
print("time: ", t_stop-t_start)
```

Όπως φαίνεται στον κώδικα, η διαδικασία γίνεται πιο εύκολη από τις μεθόδους της βιβλιοθήκης keras, με τις μεθόδους add, compile, fit, evaluate. Τα αποτελέσματα ήταν τα εξής:

```
Epoch 1/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2251 - accuracy: 0.9345
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0923 - accuracy: 0.9719
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0613 - accuracy: 0.9811
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0452 - accuracy: 0.9861
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0331 - accuracy: 0.9898
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0254 - accuracy: 0.9921
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0200 - accuracy: 0.9935
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0155 - accuracy: 0.9952
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0136 - accuracy: 0.9958
Epoch 10/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.0104 - accuracy: 0.9965
313/313 [==============================] - 1s 3ms/step - loss: 0.0766 - accuracy: 0.9805
loss:  0.07659298926591873  accuracy:  0.9804999828338623
time:  61.4009116
```

Αρχικά, παρατηρούμε ότι το loss μικραίνει και το accuracy αυξάνεται με το πέρασμα των εποχών, οπότε δεν υπάρχει υποψία overfitting. Παρατηρούμε όμως ότι στο τέλος αυξανόταν το accuracy οριακά, οπότε καταλαβαίνουμε ότι δεν υπάρχει μεγάλο όφελος από το να αυξήσουμε τον αριθμό εποχών.

Επίσης, βλέπουμε τελικά ιδιαίτερα ικανοποιητικά νούμερα στο accuracy (98%) και στο loss (0.076). Ο συνολικός χρόνος μετρήθηκε 61.4 seconds.

Στη συνέχεια, δοκιμάζουμε να προσθέσουμε ένα ακόμα hidden layer. Το πρώτο hidden layer επιλέγουμε να έχει 256 νευρώνες ενώ το δεύτερο 128, και τα δύο με activation function τη relu. Δεν τροποποιούμε κάτι άλλο.

```
mlpmodel.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

Τα αποτελέσματα είναι τα εξής:

```
Epoch 1/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2065 - accuracy: 0.9390
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0842 - accuracy: 0.9738
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0594 - accuracy: 0.9811
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0433 - accuracy: 0.9860
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0342 - accuracy: 0.9887
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0287 - accuracy: 0.9905
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0230 - accuracy: 0.9926
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0203 - accuracy: 0.9931
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0178 - accuracy: 0.9947
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0153 - accuracy: 0.9952
313/313 [==============================] - 1s 3ms/step - loss: 0.0983 - accuracy: 0.9793
loss:  0.09829473495483398  accuracy:  0.9793000221252441
time:  86.1863156
```

Όπως παρατηρείται, ενώ ξεκινάει καλύτερα, δεν έχει την πρόοδο του προηγούμενου μοντέλου και έχει χαμηλότερες τιμές accuracy και ψηλότερες loss, φτάνοντας στο evaluation σε loss 0.098, και accuracy 97.93% σε αρκετά περισσότερο χρόνο (86.18 seconds)

Δοκιμάζουμε τώρα να αυξήσουμε τον αριθμό των νευρώνων στο 1ο hidden layer σε 512. Ενώ παρατηρείται μια μικρή βελτίωση του accuracy σε σχέση με τα προηγούμενα, βλέπουμε μεγαλύτερο loss από την περίπτωση του 1ος hidden layer και σαφώς μεγαλύτερο χρόνο και από τις 2 προηγούμενες περιπτώσεις

```
Epoch 1/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1871 - accuracy: 0.9430
Epoch 2/10
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0793 - accuracy: 0.9757
Epoch 3/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0548 - accuracy: 0.9820
Epoch 4/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0409 - accuracy: 0.9865
Epoch 5/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0302 - accuracy: 0.9897
Epoch 6/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0271 - accuracy: 0.9911
Epoch 7/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0217 - accuracy: 0.9931
Epoch 8/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0204 - accuracy: 0.9933
Epoch 9/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0170 - accuracy: 0.9946
Epoch 10/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0159 - accuracy: 0.9951
313/313 [==============================] - 2s 5ms/step - loss: 0.0925 - accuracy: 0.9811
loss:  0.09247665852308273  accuracy:  0.9811000227928162
time:  130.21442950000002
```

Παρόλο που η προσθήκη 2ου hidden layer δε βοήθησε, δοκιμάζουμε την διάταξη με ακόμα ένα, 3ο hidden layer.

```
mlpmodel=tf.keras.models.Sequential()
mlpmodel.add(tf.keras.layers.Dense(512, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(256, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

Ενώ πετύχαμε ένα οριακά μεγαλύτερο accuracy(98.2%), το loss (0.081) ήταν μεγαλύτερο από την περίπτωση του ενός hidden layer και ο χρόνος σαφώς μεγαλύτερος από όλες τις περιπτώσεις (140.9 seconds)

```
Epoch 1/10
1875/1875 [==============================] - 15s 7ms/step - loss: 0.1945 - accuracy: 0.9403
Epoch 2/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0862 - accuracy: 0.9729
Epoch 3/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0632 - accuracy: 0.9800
Epoch 4/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0459 - accuracy: 0.9860
Epoch 5/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0401 - accuracy: 0.9880
Epoch 6/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0319 - accuracy: 0.9903
Epoch 7/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0294 - accuracy: 0.9908
Epoch 8/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0250 - accuracy: 0.9925
Epoch 9/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0219 - accuracy: 0.9936
Epoch 10/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0215 - accuracy: 0.9938
313/313 [==============================] - 2s 5ms/step - loss: 0.0817 - accuracy: 0.9820
loss:  0.08167263865470886  accuracy:  0.9819999933242798
time:  140.961503
```

Στη συνέχεια εξετάζουμε την ανταπόκριση αν αλλάξουμε τον optimizer σε sgd, και πιο συγκεκριμένα στην πρώτη περίπτωση του ενός hidden layer.

```
mlpmodel=tf.keras.models.Sequential()
mlpmodel.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
mlpmodel.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

mlpmodel.compile(optimizer="sgd", loss='sparse_categorical_crossentropy', metrics=['accuracy'] )
t_start=timeit.default_timer()
mlpmodel.fit(x_train, y_train, epochs=10)
```

Προκύπτει ότι το μοντέλο με sgd optimizer έχει σαφώς χειρότερη ανταπόκριση, ενώ ο ρυθμός αύξησης του accuracy μετά το πέρας των τελευταίων εποχών έχει γίνει πολύ μικρός, οπότε και πάλι δεν υπάρχει σημαντικό νόημα στην αύξηση των εποχών.

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6685 - accuracy: 0.8300
Epoch 2/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3398 - accuracy: 0.9047
Epoch 3/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2907 - accuracy: 0.9184
Epoch 4/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2605 - accuracy: 0.9274
Epoch 5/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2375 - accuracy: 0.9338
Epoch 6/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2184 - accuracy: 0.9394
Epoch 7/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2026 - accuracy: 0.9436
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1890 - accuracy: 0.9475
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1774 - accuracy: 0.9503
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1669 - accuracy: 0.9532
313/313 [==============================] - 1s 2ms/step - loss: 0.1623 - accuracy: 0.9517
loss:  0.16232533752918243  accuracy:  0.95169997215271
time:  30.290089999999996
```

Ενώ η προσθήκη κι άλλου κρυφού layer, ενώ βελτίωσε και το accuracy και το loss, είναι και πάλι λιγότερο αποδοτική από την χρήση του optimizer adam.

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5949 - accuracy: 0.8453
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2859 - accuracy: 0.9172
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2331 - accuracy: 0.9332
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1977 - accuracy: 0.9431
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1717 - accuracy: 0.9505
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1518 - accuracy: 0.9560
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1355 - accuracy: 0.9610
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1220 - accuracy: 0.9646
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1109 - accuracy: 0.9678
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1012 - accuracy: 0.9711
313/313 [==============================] - 1s 1ms/step - loss: 0.1085 - accuracy: 0.9671
loss:  0.10850853472948074  accuracy:  0.9671000242233276
time:  39.7313373
```

Έπειτα, ενώ επιστρέφουμε στην περίπτωση του adam, που φαίνεται πιο αποδοτικός από τα αποτελέσματα, δοκιμάζουμε τη χρήση του softmax και στα ενδιάμεσα layers, παρόλο που δεν προτείνεται από τη βιβλιογραφία.

Όντως, όπως ήταν αναμενόμενο, παρατηρούμε χειρότερη απόδοση σε χρόνο, ακρίβεια και loss.

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 1.6436 - accuracy: 0.6257
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.6216 - accuracy: 0.7794
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5213 - accuracy: 0.8023
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4593 - accuracy: 0.8386
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4112 - accuracy: 0.8586
Epoch 6/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3736 - accuracy: 0.8884
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3043 - accuracy: 0.9195
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2577 - accuracy: 0.9325
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2327 - accuracy: 0.9395
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2132 - accuracy: 0.9443
313/313 [==============================] - 1s 2ms/step - loss: 0.2415 - accuracy: 0.9369
loss:  0.24145977199077606  accuracy:  0.9369000196456909
time:  55.3244918
```

Στην επόμενη μας δοκιμή εφαρμόζουμε στο hidden layer activation function sigmoid, με 128 νευρώνες.

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3584 - accuracy: 0.9017
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1780 - accuracy: 0.9487
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1229 - accuracy: 0.9640
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0907 - accuracy: 0.9733
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0693 - accuracy: 0.9801
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0546 - accuracy: 0.9844
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0426 - accuracy: 0.9879
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0333 - accuracy: 0.9909
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0266 - accuracy: 0.9933
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0203 - accuracy: 0.9954
313/313 [==============================] - 1s 2ms/step - loss: 0.0676 - accuracy: 0.9795
loss:  0.06755343079566956  accuracy:  0.9794999957084656
time:  40.5171784
```

Προκύπτουν αξιόλογα αποτελέσματα, με loss 0.067, accuracy 97.9% και χρόνο 40.5 seconds. Παρατηρούμε όμως ότι το accuracy συνεχίζει να αυξάνεται στις τελευταίες εποχές, οπότε δοκιμάζουμε να αυξήσουμε τον αριθμό τους σε 15:

```
Epoch 1/15
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3556 - accuracy: 0.9034
Epoch 2/15
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1792 - accuracy: 0.9475
Epoch 3/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1244 - accuracy: 0.9634
Epoch 4/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0927 - accuracy: 0.9731
Epoch 5/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0705 - accuracy: 0.9797
Epoch 6/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0546 - accuracy: 0.9843
Epoch 7/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0431 - accuracy: 0.9879
Epoch 8/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0338 - accuracy: 0.9911
Epoch 9/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0271 - accuracy: 0.9929
Epoch 10/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0209 - accuracy: 0.9947
Epoch 11/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0161 - accuracy: 0.9966
Epoch 12/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0128 - accuracy: 0.9976
Epoch 13/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0096 - accuracy: 0.9985
Epoch 14/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0075 - accuracy: 0.9990
Epoch 15/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0061 - accuracy: 0.9991
313/313 [==============================] - 1s 1ms/step - loss: 0.0683 - accuracy: 0.9805
loss:  0.06829217821359634  accuracy:  0.9804999828338623
time:  58.5788202
```

Αυτή τη φορά βλέπουμε ότι στο τέλος φτάνουμε στη μέγιστη απόδοση του αλγορίθμου πριν γίνει το overfitting, και ο αλγόριθμος πετυχαίνει την καλύτερη, ως τώρα, απόδοση, με accuracy 98.04%, loss 0.068 και χρόνο 58.57 seconds.

Δοκιμάζουμε, τώρα, να προσθέσουμε ένα ακόμα hidden layer. Τα αποτελέσματα είναι οριακά χειρότερα, όπως φαίνεται παρακάτω:

```
Epoch 1/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3818 - accuracy: 0.8939
Epoch 2/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1555 - accuracy: 0.9537
Epoch 3/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1024 - accuracy: 0.9693
Epoch 4/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0729 - accuracy: 0.9781
Epoch 5/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0542 - accuracy: 0.9831
Epoch 6/15
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0404 - accuracy: 0.9875
Epoch 7/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0307 - accuracy: 0.9905
Epoch 8/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0239 - accuracy: 0.9930
Epoch 9/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0180 - accuracy: 0.9948
Epoch 10/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0143 - accuracy: 0.9958
Epoch 11/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0121 - accuracy: 0.9962
Epoch 12/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0081 - accuracy: 0.9979
Epoch 13/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0084 - accuracy: 0.9973
Epoch 14/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0067 - accuracy: 0.9979
Epoch 15/15
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0054 - accuracy: 0.9985
313/313 [==============================] - 1s 2ms/step - loss: 0.0881 - accuracy: 0.9803
loss:  0.08806736022233963  accuracy:  0.9803000092506409
time:  65.6082383
```

Τέλος, αφού καταλήξαμε στο ένα μόνο κρυφό layer, με optimizer adam, και activation functions sigmoid και softmax, δοκιμάζουμε να αυξήσουμε τον αριθμό των νευρώνων στο ενδιάμεσο layer σε 256, και τα αποτελέσματα είναι οριακά χειρότερα από την περίπτωση των 128, με accuracy 97.78%, loss 0.074 και χρόνο 52.82 seconds.

```
Epoch 1/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3957 - accuracy: 0.8972
Epoch 2/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1962 - accuracy: 0.9430
Epoch 3/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1428 - accuracy: 0.9589
Epoch 4/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1111 - accuracy: 0.9679
Epoch 5/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0890 - accuracy: 0.9744
Epoch 6/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0730 - accuracy: 0.9791
Epoch 7/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0607 - accuracy: 0.9832
Epoch 8/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0508 - accuracy: 0.9858
Epoch 9/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0431 - accuracy: 0.9886
Epoch 10/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0362 - accuracy: 0.9906
Epoch 11/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0304 - accuracy: 0.9925
Epoch 12/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0256 - accuracy: 0.9941
Epoch 13/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0214 - accuracy: 0.9957
Epoch 14/15
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0181 - accuracy: 0.9968
Epoch 15/15
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0153 - accuracy: 0.9973
313/313 [==============================] - 1s 1ms/step - loss: 0.0743 - accuracy: 0.9778
loss:  0.07434379309415817  accuracy:  0.9778000116348267
time:  52.82857010000001
```

Τελευταίο πράγμα που θα ελέγξουμε είναι η ανταπόκριση του μοντέλου αν αλλάξουμε το loss function από sparse categorical crossentropy σε Mean Squared Error.

Αρχικά, με το mean squared error, παρατηρούνται πολύ χειρότερα αποτελέσματα.

```
mlpmodel=tf.keras.models.Sequential()
mlpmodel.add(tf.keras.layers.Dense(128, activation=tf.nn.sigmoid))
mlpmodel.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

mlpmodel.compile(optimizer="adam", loss='mean_squared_error', metrics=['accuracy'] )
t_start=timeit.default_timer()
mlpmodel.fit(x_train, y_train, epochs=15)
```

Πιο συγκεκριμένα, έχουμε accuracy 7.4%, loss 27.25 και χρόνο 48.1 seconds.

```
Epoch 1/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0986
Epoch 2/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.1010
Epoch 3/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0994
Epoch 4/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0997
Epoch 5/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0987
Epoch 6/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0999
Epoch 7/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3045 - accuracy: 0.0997
Epoch 8/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.1009
Epoch 9/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0993
Epoch 10/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.1007
Epoch 11/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0981
Epoch 12/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.0993
Epoch 13/15
1875/1875 [==============================] - 4s 2ms/step - loss: 27.3046 - accuracy: 0.1016
Epoch 14/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.1009
Epoch 15/15
1875/1875 [==============================] - 3s 2ms/step - loss: 27.3046 - accuracy: 0.1024
313/313 [==============================] - 1s 2ms/step - loss: 27.2503 - accuracy: 0.0744
loss:  27.25031280517578  accuracy:  0.07440000027418137
time:  48.1250283
```

Έτσι καταλήγουμε στο βέλτιστο μοντέλο μας, με 1 hidden layer με 128 νευρώνες, activation function sigmoid σε αυτό και softmax στο output layer, loss function sparse categorical crossentropy και optimizer adam.

2. K-Nearest-Neighbors

Στο επόμενο στάδιο θα συγκρίνουμε την απόδοση του προηγούμενου δικτύου με τη μέθοδο των πλησιέστερων γειτόνων. Και πάλι εφαρμόζουμε την κανονικοποίηση και την και το reshape των data, και χρησιμοποιούμε τη μέθοδο από την βιβλιοθήκη sklearn, μια φορά με 3 και μια φορά με 1 κοντινότερο γείτονα, όπως φαίνεται παρακάτω:

```python
import tensorflow as tf
import matplotlib.pyplot as mpl
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score
import timeit
(x_train, y_train), (x_test, y_test) =tf.keras.datasets.mnist.load_data()
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)

x_train=x_train/255
x_test=x_test/255

knn=KNeighborsClassifier(n_neighbors=3)
t_start=timeit.default_timer()
knn.fit(x_train, y_train)
y_pred=knn.predict(x_test)
t_stop=timeit.default_timer()

report=classification_report(y_test, y_pred)
print("classification report:\n", report)

acc=accuracy_score(y_test, y_pred)
print("accuracy: ", {acc:.4})
print("time: ", t_stop-t_start)
```

Για 3 γείτονες, τα αποτελέσματα είναι τα εξής:

```
classification report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.96      1.00      0.98      1135
           2       0.98      0.97      0.97      1032
           3       0.96      0.97      0.96      1010
           4       0.98      0.97      0.97       982
           5       0.97      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

accuracy:  {0.9705: 0.4}
time:  16.0653785
```

Όπως φαίνεται, το συνολικό μας accuracy είναι 97%, ενώ ο αλγόριθμος ήταν ιδιαίτερα αποτελεσματικός, όπως φαίνεται από το classification report, στην αναγνώριση του αριθμού 8, με accuracy 99%. Έτσι ο αλγόριθμος είναι

οριακά λιγότερο αποτελεσματικός από το προηγούμενο δίκτυο. Ήταν, βέβαια, αρκετά γρηγορότερος, με συνολικό χρόνο 16 seconds.

Αν, τώρα, βάλουμε n_neighbors=1, προκύπτουν τα εξής αποτελέσματα:

```
classification report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

accuracy:  {0.9691: 0.4}
time:  13.6507012
```

Έχουμε, δηλαδή, λίγο μικρότερη απόδοση από την περίπτωση των 3 πλησιέστερων γειτόνων.

3. Nearest Centroid

Τώρα παρουσιάζεται η εφαρμογή της μεθόδου των πλησιέστερων κέντρων κλάσης με μετρική euclidean, που επίσης εφαρμόστηκε μέσω της βιβλιοθήκης sklearn, όπως φαίνεται παρακάτω:

```python
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.neighbors import NearestCentroid
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd
import tensorflow as tf
import timeit
#Load the data, and convert them from 3D to 2D arrays
(x_train, y_train), (x_test, y_test) =tf.keras.datasets.mnist.load_data()

x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
#Normalization
x_train=x_train/255
x_test=x_test/255

t_start=timeit.default_timer()
model= NearestCentroid(metric='euclidean')
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
t_stop=timeit.default_timer()
print("Classification report:\n", classification_report(y_test,y_pred))
print("accuracy: ", accuracy_score(y_test, y_pred))
print("time: ", t_stop-t_start)
```

Και τα αποτελέσματα φαίνονται παρακάτω:

```
Classification report:
              precision    recall  f1-score   support

           0       0.91      0.90      0.90       980
           1       0.77      0.96      0.86      1135
           2       0.88      0.76      0.81      1032
           3       0.77      0.81      0.78      1010
           4       0.80      0.83      0.81       982
           5       0.75      0.69      0.72       892
           6       0.88      0.86      0.87       958
           7       0.91      0.83      0.87      1028
           8       0.79      0.74      0.76       974
           9       0.77      0.81      0.79      1009

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000

accuracy:  0.8203
time:  0.2684690000000005
```

Βλέπουμε ότι έχουμε μικρότερο accuracy (82%) από τους άλλους 2 αλγορίθμους, με μικρότερη την ικανότητα του μοντέλου να ανιχνεύσει τον αριθμό 5 (75%), και συνολικό χρόνο 22 δευτερόλεπτα.

<u>Συμπεράσματα</u>

Ως καλύτερο μοντέλο έχει προκύψει το παρακάτω:

```
mlpmodel=tf.keras.models.Sequential()
mlpmodel.add(tf.keras.layers.Dense(128, activation=tf.nn.sigmoid))
mlpmodel.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

mlpmodel.compile(optimizer="adam", loss='sparse_categorical_crossentropy', metrics=['accuracy'] )
t_start=timeit.default_timer()
mlpmodel.fit(x_train, y_train, epochs=15)
```

Με χρήση του αλγορίθμου back-propagation με τις παραμέτρους που φαίνονται στον κώδικα.