Νευρωνικά Δίκτυα-Βαθιά Μάθηση
Πρώτη Εργασία

Ονοματεπώνυμο: Αθανάσιος Γιαπουτζής

ΑΕΜ: 3589

Η παρακάτω έκθεση είναι χωρισμένη σε τρία μέρη. Στο πρώτο κομμάτι γίνεται συνοπτική ανάλυση της βάσης *MNIST* η οποία χρησιμοποιήθηκε για το testing και training των αλγορίθμων/νευρωνικού. Στο δεύτερο κομμάτι συγκρίνονται οι αλγόριθμοι κατηγοριοποίησης Nearest Neighbor και Nearest Class Centroid, ενώ στο τρίτο κομμάτι αναλύεται εκτενώς το υλοποιημένο νευρωνικό δίκτυο και συγκρίνεται με τους δύο προηγούμενους κατηγοριοποιητές.

1 MNIST

Όπως προαναφέρθηκε, για την εκπαίδευση και το testing του δικτύου χρησιμοποιήθηκε η βάση MNIST. Αποτελείται από 60000 εικόνες ψηφίων για εκπαίδευση και 10000 για testing. Η αρχική αναπαράστατη τους γίνεται με μορφή τρισδιάστου πίνακα, δηλαδή με πίνακα 60000 θέσεων ο οποίος περιέχει δισδιάστατους πίνακες διαστάσεων 28x28. Κάθε κελί των παραπάνω δισδιάστατων πινάκων περιέχει τιμές από το 0 εως το 255, με το 0 να δηλώνει το μαύρο και το 255 το άσπρο.

2 Σύγκριση κατηγοριοποιητών

Ο κατηγοριοποιητής πλησιέστερου γείτονα με 1 γείτονα έδωσε τα εξής αποτελέσματα:

	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

Ο κατηγοριοποιητής πλησιέστερου γείτονα με 3 γείτονες έδωσε τα εξής αποτελέσματα:

	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

Ο κατηγοριοποιητής πλησιέστερου κέντρου έδωσε τα εξής αποτελέσματα:

	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

Σύμφωνα με τα παραπάνω metrics καλύτερος στην επίλυση του συγκεκριμένου προβλήματος φαίνεται να είναι ο KNN με τρεις πλησιέστερους γείτονα ένω σχεδόν όμοια αποτελεσματα έχει δώσει ο ίδιος αλγόριθμος με έναν πλησιέστερο γείτονα. Τέλος, η απόδοση του NC, είναι αρκετά χειρότερη, πετυχαίνοντας μόλις 82% accuracy.

3 Ανάλυση νευρωνικού δικτύου

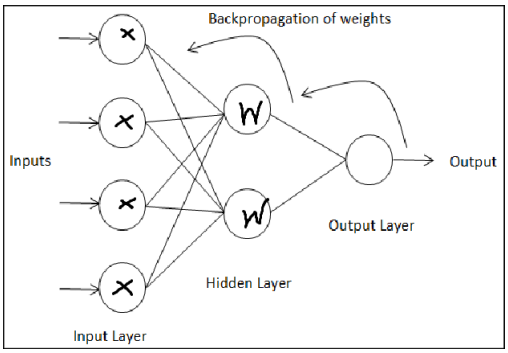
Για την υλοποίηση του νευρωνικού χρησιμοποιήθηκε deep learning αρχιτεκτονική της tensorflow.
Τα δεδομένα της βάσης φορτώνονται και τροποποιούνται κατάλληλα ώστε να γίνουν επεξεργάσιμα απο το δίκτυο, μετατρέπονται δηλαδή σε διανύσματα διάστασης 1x784. Στην συνέχεια κανονικοποιούμε τις τιμές για να αποφύγουμε μεγάλες τιμές της παραγώγου κατά το πέρασμα τους μέσα από το δίκτυο.
Τέλος η κωδικοποίηση των labels είναι sparse scalar γεγονός που δεν βολεύει την επεξαργασία τους, για αυτό και μετατρέπονται σε one-hot vectors.
Η υλοποίηση των παραπάνω με κώδικα είναι η εξής:

```
    #Loading data
    (train_images, train_labels), (test_images, test_labels) =
    mnist.load_data()

    #Reshaping data
    train_images = np.reshape(train_images, (-1, 784))
    test_images = np.reshape(test_images, (-1, 784))

    #Normalizing to avoid high gradient values
    train_images = train_images.astype('float32') / 255
    test_images = test_images.astype('float32') / 255

    #Converting labels into a one-hot vector
    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)
```

Η σωστή επιλογή μοντέλου έγινε έπειτα από εκτενή χρήση cross validation για την κατάλληλη επιλογή των παραμέτρων(layers, epochs).

Συνοπτική ανάλυση Backpropagation

Ο Backpropagation αποτελεί την κύρια μέθοδο για τον υπολογισμό των βαρών των ακμών του νευρωνικού δικτύου βάση των σφαλμάτων που παράχθηκαν στην προηγούμενη εποχή, ενώ αποτελεί και τον αλγόριθμο που χρησιμοποιεί η αρχιτεκτονική της tensorflow για εκπαίδευση. Ο κατάλληλος υπολογισμός των βαρών επιτρέπει την μείωση των σφαλμάτων κάνοντας το μοντέλο πιο αξιόπιστο. Πιο συγκεκριμένα ο αλγόριθμος υπολογίζει την παράγωγο μιας συνάρτησης ΄σφάλματος΄ για ένα συγκεκριμένο βάρος.



(i) Η είσοδος x φτάνει από το πρώτο επίπεδο
(ii) Μεταβάλλεται κατάλληλα η είσοδος (αύξηση/μείωση των διαστάσεων), σύμφωνα με τα βάρη του δικτύου (τα βάρη στην αρχή αρχικοποιούνται τυχαία).
(iii) Υπολογίζεται η έξοδος κάθε νευρώνα από το πρώτο επίπεδο, στα κρυφά επίπεδα και εν τέλει στο επίπεδο εξόδου.
(iv) Υπολογίζεται το σφάλμα εξόδου σύμφωνα με την παρακάτω φόρμουλα:

```
Error = ActualOutput - DesiredOutput
```

(v) Επιστροφή από το επίπεδο εξόδου προς τα πίσω, ενημερώνοντας κατάλληλα τα βάρη έτσι ώστε το σφάλμα να μειώνεται.
Η παραπάνω διαδικασία επαναλαμβάνεται εώς ότου επιτευχθεί το επιθυμητό accuracy.

Επιλογή εποχών

Πιο συγκεκριμένα:
Για 6 folds με 30 εποχές το κάθε ένα το δίκτυο πέτυχε τα εξής:

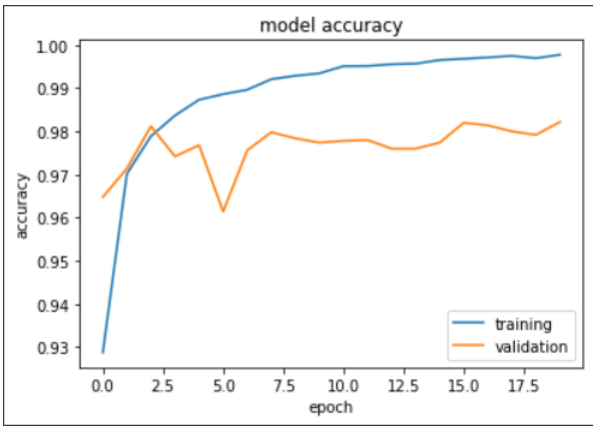
$$maxValAccuracy_1 = 0.9774, epoch = 16$$
$$maxValAccuracy_2 = 0.9766, epoch = 19$$
$$maxValAccuracy_3 = 0.9761, epoch = 14$$
$$maxValAccuracy_4 = 0.9769, epoch = 27$$
$$maxValAccuracy_5 = 0.9785, epoch = 26$$
$$maxValAccuracy_6 = 0.9777, epoch = 17$$

Αθροίζοντας τις εποχές και διαιρόντας με το πλήθος τους παίρνουμε μέσο όρο εποχών ίσο με 19.8333. Καταλληλότερος αριθμός εποχών άρα είναι οι 20.

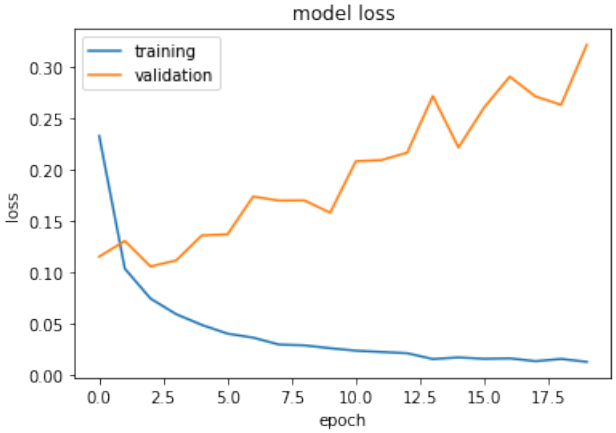
Επιλογή layers

Παρόμοια διαδικασία ακολουθήθηκε και για την επιλογή του αριθμού των κρυφών layers. Αρχικά δοκιμάστηκαν τέσσερα(4) layers(ένα εισόδου, ένα εξοδου και

δύο κρυφά):



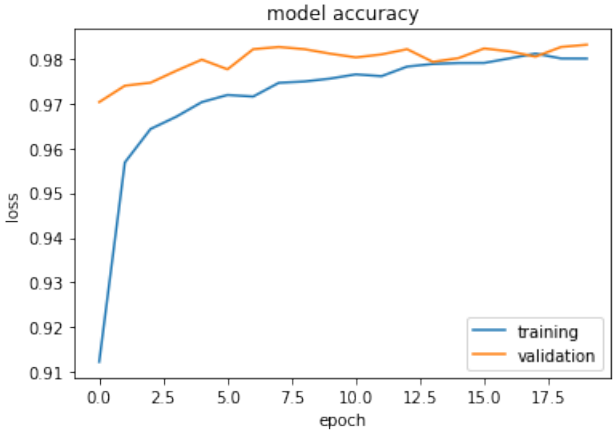
Παρατηρούμε ότι το training accuracy βελτιώνεται συνεχώς ενώ το validation accuracy έχει απρόβλεπτη συμπεριφορά.



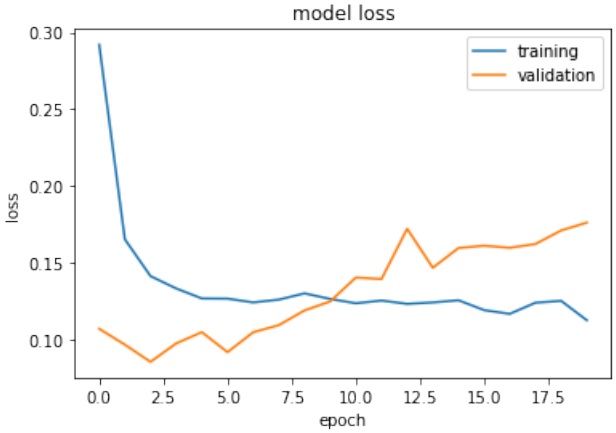
Παρατηρούμε ότι ενώ το σφάλμα του κατά την εκπαίδευση μειώνεται, το σφάλμα του validation συνεχώς αυξάνεται. Αυτό συμβαίνει διότι το μοντέλο υπερεκπαιδεύεται, κοινώς παπαγαλίζει. Για να το καταπολεμίσουμε αυτό εισάγουμε ένα Dropout κάτω από κάθε layer το οποίο ουστιαστικά είναι μια πιθανότητα να μην συνυπολογιστεί κάποιος νευρώνας στο επόμενο πέρασμα μέσα από το δίκτυο.

```
1    model.add(Dropout(0.3))
```

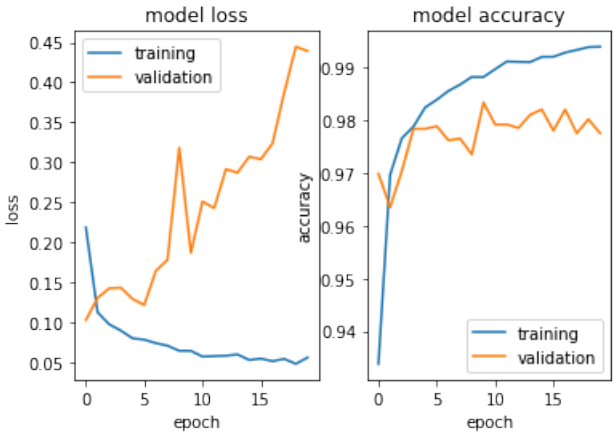
Παρατηρούμε ότι μετά την εισαγωγή του Dropout το validation accuracy βελτιώνεται κατα πολύ.



Βελτίωση παρατηρείται και στο validation loss μετά την εισαγωγή του Dropout. Είναι εμφανές ότι κάνοντας drop νευρώνες το δίκτυο δυσκολεύεται περισσότερο να μάθει, για αυτό και η αύξηση του training loss.



Αυξάνοντας τα layers σε πέντε(5) έχουμε τα εξής αποτελέσματα(χωρίς την χρήση Dropout)



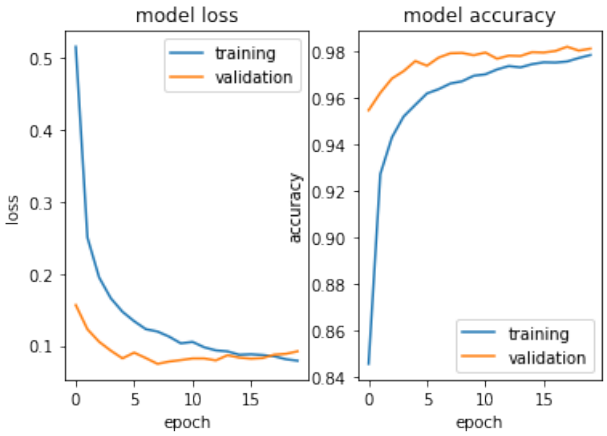
Ενώ θα περιμέναμε με την προσθήκη επιπέδων να μειωθεί το validation loss αυτό συνεχίζει να αυξάνεται. Αυτό συμβαίνει διότι ενισχύοντας το δίκτυο μας με περισσότερα επιπέδα ναι μεν το κάνουμε δυνατότερο αλλά ταυτόχρονα αυξάνουμε και την δύναμη στο να μαθαίνει το dataset με αποτέλεσμα να παπαγαλίζει πολύ ευκολότερα.
Η χρήση Dropout βελτιώνει κατά πολύ την κατάσταση των validation metrics, αλλά χειροτερεύει τα training metrics.
Η προσθήκη layers επομένως δεν είναι ωφέλιμη για το συγκεκριμένο πρόβλημα classification, άρα θα αρκεστούμε στα τέσσερα(4) layers.

Επιλογή νευρώνων κρυφού επιπέδου

Έχοντας επιλέξει δύο κρυφά επίπεδα για το δίκτυο, σειρά έχει η επιλογή του αριθμού των νευρώνων των επιπέδων. Οι κύριοι κανόνες για αυτη την επιλογή είναι:
(i) Ο αριθμός των κρυφών νευρώνων πρέπει να είναι μεταξύ του μεγέθους της εισόδου και εξόδου του δικτύου.
(ii) Ο αριθμός των κρυφών νευρώνων πρέπει να είναι τα 2/3 του μεγέθους του επιπέδου εισόδου συν το μέγεθος του επιπέδου εξόδου.
Βέβαια η εφαρμογή των παραπάνω κανόνων δεν εγγυάται πάντα τα καλύτερα αποτελέσματα διότι την επιτυχία του μοντέλου επηρεάζουν και άλλα στοιχεία όπως, η συνάρτηση ενεργοποίησης, η πολυπλοκότητα των δεδομένων κλπ.
Στο συγκεκριμένο πρόβλημα η επιλογή του κατάλληλου αριθμού νευρώνων έγινε εμπειρικά, με την χρήση διαφορετικών αριθμών και παρατήρησης των metrics.

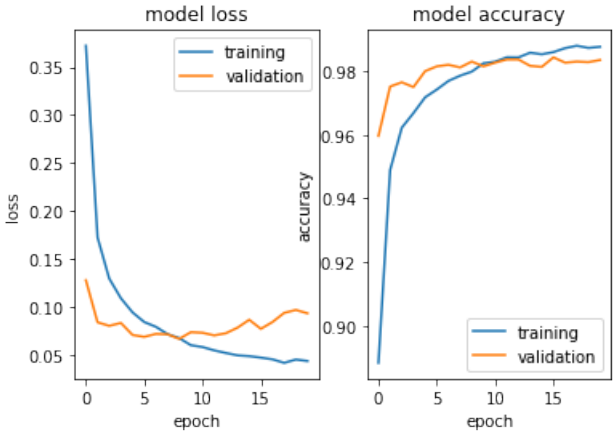
Η χρήση 128 νευρώνων για το πρώτο κρυφό επίπεδο και 64 για το δεύτερο έδωσε τα εξής αποτελέσματα:

```
Accuracy: 0.9786111116409302
Validation Accuracy: 0.9821666479110718
Loss: 0.07907987385988235
Validation Loss: 0.07470954209566116
```



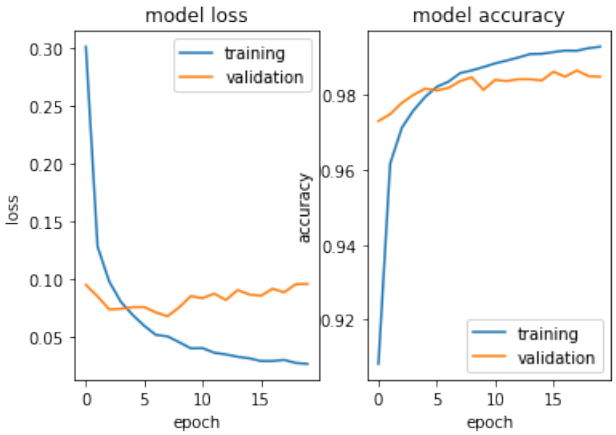
Η χρήση 256 νευρώνων για το πρώτο κρυφό επίπεδο και 128 για το δεύτερο έδωσε τα εξής αποτελέσματα:

```
Accuracy: 0.9879814982414246
Validation Accuracy: 0.984333336353302
Loss: 0.04186360538005829
Validation Loss: 0.06684502959251404
```



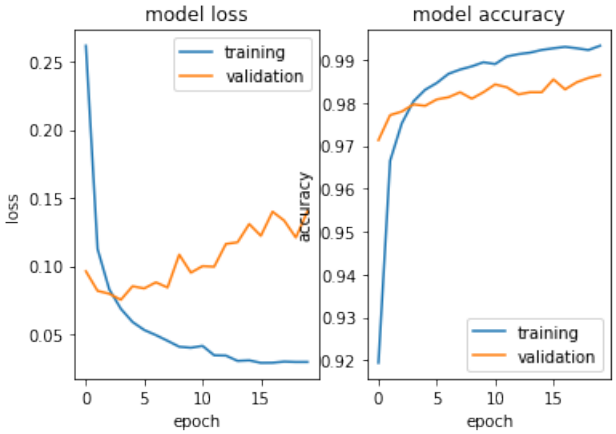
Η χρήση 512 νευρώνων για το πρώτο κρυφό επίπεδο και 256 για το δεύτερο έδωσε τα εξής αποτελέσματα:

```
Accuracy: 0.9928333163261414
Validation Accuracy: 0.9865000247955322
Loss: 0.02642500400543213
Validation Loss: 0.06765461713075638
```



Η χρήση 1024 νευρώνων για το πρώτο κρυφό επίπεδο και 512 για το δεύτερο έδωσε τα εξής αποτελέσματα:

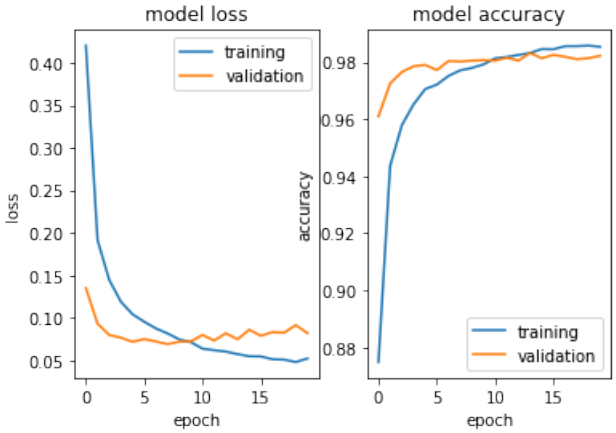
```
    Accuracy: 0.9933518767356873
    Validation Accuracy: 0.9865000247955322
    Loss: 0.029050879180431366
    Validation Loss: 0.07528889179229736
```



Από τα τέσσερα παραπάνω γραφήματα είναι εύκολο να παρατηρήσουμε το γεγονός ότι αυξάνοντας τους νευρώνες των κρυφών επιπέδων, ναι μεν πετυχαίνουμε πολύ καλά επίπεδα accuracy αλλα μετά από ένα σημείο το μοντέλο υπερεκπαιδεύεται διότι του παρέχουμε υπολογιστική ισχύ η οποία δεν χρειάζεται για το συγκεκριμένο πρόβλημα. Μπορούμε να το παρατηρήσουμε αυτό στα τελευταία γραφήματα όπου το μοντέλο τείνει να μάθει τέλεια το dataset αλλά η απόδοση του στο validation δείχνει να χειροτερεύει.
Επομένως, σύμφωνα με τα παραπάνω ο επιθυμητός αριθμός νευρώνων για τα δύο κρυφά επίπεδα του δικτύου είναι 256 και 64 αντίστοιχα. Να σημειωθεί οτι δεν εφαρμόστηκε prunning στο μοντέλο.

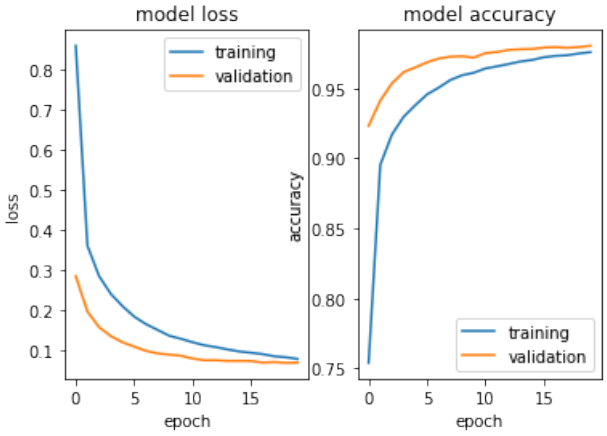
Επιλογή συνάρτησης ενεργοποίησης

Όλες οι παραπάνω δοκιμές έγιναν χρησιμοποιώντας σαν συνάρτηση ενεργοποίησης την ReLU.
Το γράφημα παρακάτω παρουσιάζει το μοντέλο έχοντας σαν συνάρτηση ενεργοποίησης την ReLU, με δύο κρυφά επίπεδα με 256 και 64 νευρώνες αντίστοιχα.




```
Accuracy: 0.9857592582702637
Validation Accuracy: 0.9831666946411133
Loss: 0.048112235963344574
Validation Loss: 0.06914559751749039
```

Το γράφημα παρακάτω παρουσιάζει το μοντέλο έχοντας σαν συνάρτηση ενεργοποίησης την σιγμοειδή με δύο κρυφά επίπεδα με 256 και 64 νευρώνες αντίστοιχα.




```
Accuracy: 0.9759073853492737
Validation Accuracy: 0.9804999828338623
Loss: 0.07962463051080704
Validation Loss: 0.06990333646535873
```

Παρατηρούμε ότι τα αποτελέσματα είναι σχεδόν όμοια, με την ReLU να τα πηγαίνει

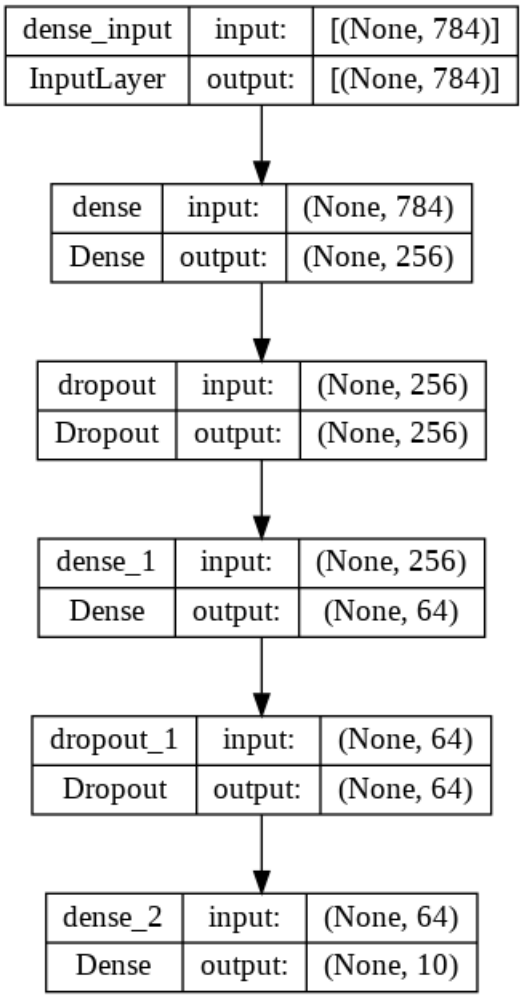
ελαφρώς καλύτερα. Επομένως επιλέγουμε την ReLU σαν συνάρτηση ενεργοποίησης.

Το τελικό μοντέλο - Περιγραφή κώδικα

Έχοντας επιλέξει τις κατάλληλες παραμέτρους για το δίκτυο, το υλοποιούμε με τις παρακάτω εντολές Python.

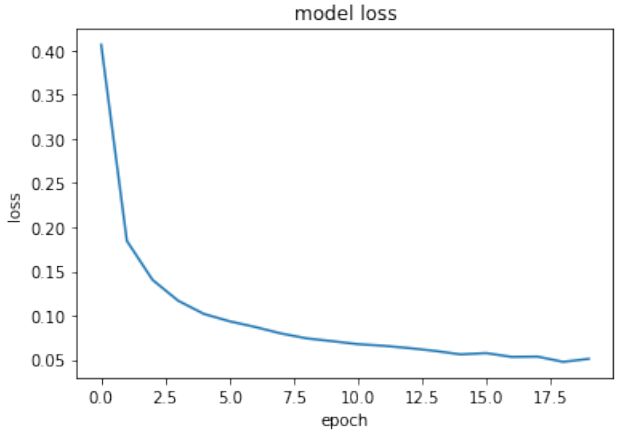
```
def create_dense():
  model = Sequential()

  #Adding 4 layers
  model.add(Dense(256, activation='relu', input_dim=784))
  model.add(Dropout(0.3))
  model.add(Dense(64, activation='relu'))
  model.add(Dropout(0.3))
  model.add(Dense(10, activation='softmax'))

  #Model compilation
  model.compile(optimizer='RMSprop', loss='
  categorical_crossentropy', metrics=['accuracy'])

  return model
```

Επιπλέον το παρακάτω διάγραμμα κάνει πολύ καλό visualization του μοντέλου και της ροής που θα έχουν τα δεδομένα μέσα σε αυτό.



Στην συνέχεια γίνεται fit το μοντέλο με τις παρακάτω εντολές. Το διάγραμμα δείχνει την πορεία του training loss κατά το πέρασμα των εποχών.

```
history = model.fit(train_images, train_labels, epochs=20,
batch_size=128, verbose=False)
```



Έχοντας εκπαιδεύσει το δίκτυο, έχει σειρά να το κάνουμε evaluate χρησιμοποιώντας τα testing δεδομένα.

```
loss, accuracy = model.evaluate(test_images, test_labels,
verbose=False)
```

Το loss και το accuracy του μοντέλου είναι τα παρακάτω:

```
Testing loss:0.11527601629495621
Testing accuracy: 0.9778000116348267
```

Στον κώδικα επιπλέον έχει υλοποιηθεί και κομμάτι το οποίο υπολογίζει διάφορα metrics(precision, recall, f1 score)

```
#Classification report
y_pred = model.predict(test_images.reshape(-1, 784))
y_pred_classes = np.argmax(y_pred, axis=1)
y_pred = y_pred[:,0]
test_labels = np.argmax(test_labels, axis=1)
cr = classification_report(test_labels, y_pred_classes)
print(cr)
```

Σύγριση MLP με τους κατηγοριοποιητές

Έφοσον η σύγκριση των δύο κατηγοριοποιητών έχει γίνει σε προηγούμενο κομμάτι, εδώ απλά θα συγκριθούν με το MLP.
Εξάγοντας το classification report του νευρωνικού παρατηρούμε ότι έχει πετύχει πάρα πολύ καλά ποσοστά precision, recall, και f1 score.

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.98	0.99	0.99	1135
2	0.98	0.98	0.98	1032
3	0.96	0.99	0.98	1010
4	0.98	0.98	0.98	982
5	0.99	0.97	0.98	892
6	0.98	0.98	0.98	958
7	0.98	0.97	0.98	1028
8	0.98	0.97	0.98	974
9	0.98	0.97	0.98	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

Το MLP επομένως είναι ανώτερο των δύο προηγούμενων κατηγοριοποιητών. Πιο συγκεκριμένα ο KNN έδωσε πολύ ικανοποιητικά αποτελέσματα κοντράροντας το νευρωνικό και φτάνοντας σε 97% accuracy.
Αντίθετα, ο NC κατάφερε μακράν χειρότερα αποτελέσματα με μόλις 82% accuracy. Καταλήγουμε άρα στο ότι το νευρωνικό αποτελεί την αποδοτικότερη και αποτελεσματικότερη τεχνολογία για την επίλυση του συγκεκριμένου προβλήματος classification.

4 Πηγές

https://adventuresinmachinelearning.com/python-tensorflow-tutorial/

https://rukshanpramoditha.medium.com/acquire-understand-and-prepare-the-mnist-dataset-3d71a84e07e7

https://www.tensorflow.org/guide/keras/