ΥΣ19 Artificial Intelligence II

Δεύτερη εργασία.

Ονοματεπώνυμο: Απόστολος Καρβέλας

Α.Μ.: 1115201800312

Τρίτο υποερώτημα

ΕΙΣΑΓΩΓΗ

Στην εργασία αυτή έγινε υλοποίηση ενός multilabel vaccine sentiment classifier χρησιμοποιώντας feed-forward neural network. Στην αρχή χρησιμοποιούμε countervectorizer δοκιμάζοντας loss function και optimizers και συγκεκριμένα για το SGD optimizer πειραματιζόμαστε για κάθε και παράμετρο του.

CounterVectorizer

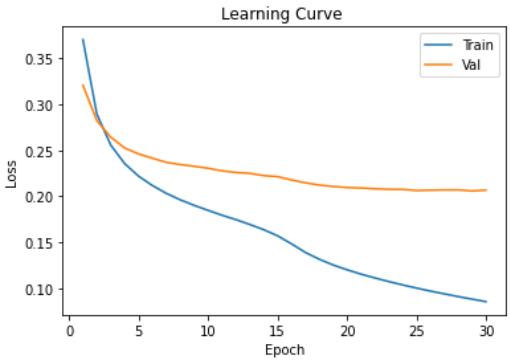
Για αρχή θα χρησιμοποιήσουμε CountVectorizer για το vectorization των tweets σε πίνακα με αριθμούς. Συγκεκριμένα, ο CountVectorizer είναι πολύ απλός για τον λόγο ότι απλά μετράει πόσες φορές εμφανίζεται η κάθε λέξη στο κείμενο και επιστρέφει έναν πίνακα με 15976 γραμμές, που είναι ο αριθμός των tweets, και 32656 στήλες, που ισούται με τον αριθμό των διαφορετικών λέξεων στα tweets. Χρησιμοποιούμε τον count vectorizer για να εξετάσουμε κάθε υπερπαράμετρο πριν υλοποιήσουμε κάποια πιο περίπλοκη δομή. Ο αριθμός των σειρών είναι πολύ μεγάλος αφού δεν έχουμε αφαιρέσει ακόμα τις άχρηστες λέξεις για το πρόγραμμα μέσω preprocessing και max features.

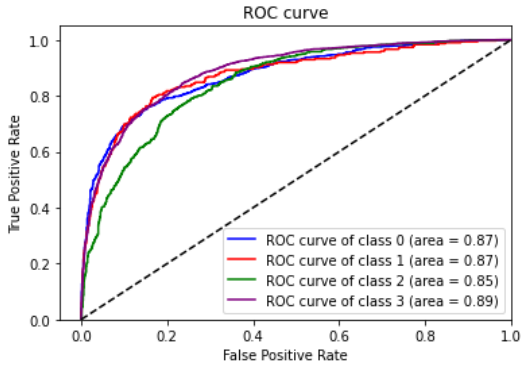
Σαν πρώτη δοκιμή έχει φτιαχτεί neural network χωρίς hidden layer αλλά μόνο input και output, με αυτόν τον τρόπο υλοποιείται ένας logistic regression. Σαν input βάζουμε τις διαστάσεις του torch με τα tweets του train και σαν output τον αριθμό 3, ένα για κάθε κλάση που έχουμε. Στην συνάρτηση forward του νευρωνικού δικτύου επιστέφουμε την γραμμική συνάρτηση (Linear) του pytorch αφού έχει πρώτα περάσει από την activation function softmax για multilabel σύστημα.

Για το training του νευρωνικού δικτύου χρειάζεται ένα loss function και έναν optimizer.

Στην συγκεκριμένη περίπτωση χρησιμοποιούμε L1 loss function με Adam optimizer.

Πέρα από αυτά τα βασικά στοιχεία έχουμε βάλει learning rate 0.001 με batch size 64 και αριθμό από epoch 30. Ο αριθμός των epoch είναι 30 για λόγους ταχύτητας.





Στο τελευταίο epoch έχουμε τα εξής στοιχεία:

Training loss = 0.0854

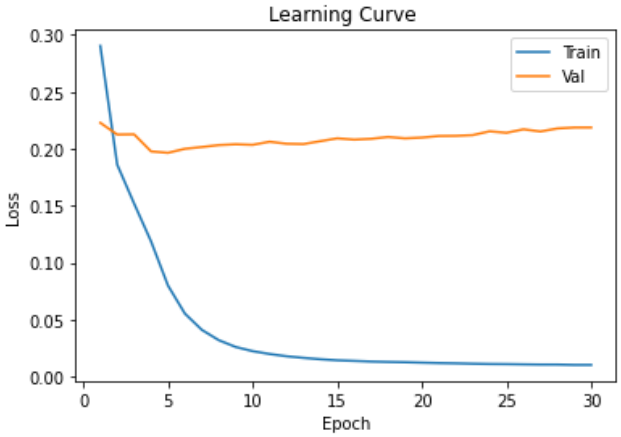
Validation loss = 0.2066

Training F1_score = 0.9191

Validation F1_score = 0.725

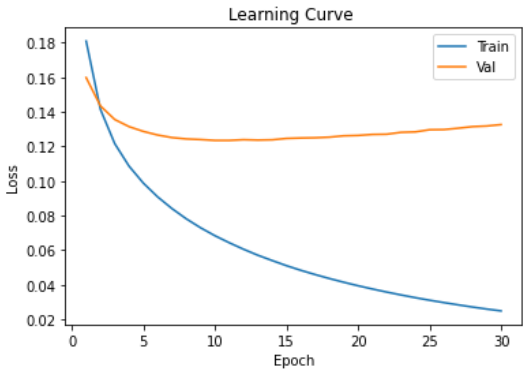
Από το learning curve παρατηρούμε ότι στο δίκτυο συμβαίνει πολύ overfitting, το οποίο είναι λογικό αφού δεν έχουμε κάνει preprocess και χρησιμοποιούμε απλό vectorizer. Επίσης η L1 function βρίσκει απλά το μέσο του σφάλματος και η Adam δεν είναι κατάλληλη για το σύστημα μας.

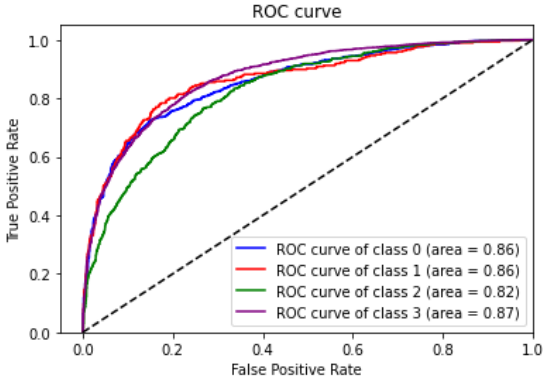
Στην συνέχεια θα δοκιμάσουμε με έναν hidden layer που έχει 50 κόμβους. Στην forward του μοντέλου θα χρησιμοποιήσουμε σαν activation function την ReLU για το hidden layer και θα συνεχίσουμε να επιστέφουμε την γραμμική συνάρτηση μέσω softmax.



Παρατηρούμε ότι το μοντέλο έχει ακόμα περισσότερο overfitting καθώς με την αύξηση των hidden layers γίνεται πιο περίπλοκο το μοντέλο. Οπότε για τώρα θα χρησιμοποιήσουμε χωρίς hidden layer.

Στην συνέχεια σαν loss function θα βάλουμε την MSE η οποία είναι με η L2 με αντίθεση την προηγούμενη L1 function. Η MSE τετραγωνίζει το σφάλμα πριν πάρει την μέση τιμή οπότε επηρεάζεται πολύ από τις ακραίες τιμές. Ενώ η L1 απλά παίρνει την μέση τιμή των σφαλμάτων. Οπότε με L2 function έχουμε:





Στο τελευταίο epoch έχουμε τα εξής στοιχεία:

Training loss = 0.0248

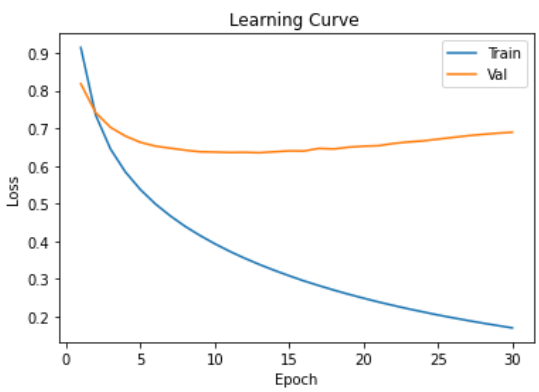
Validation loss = 0.1326

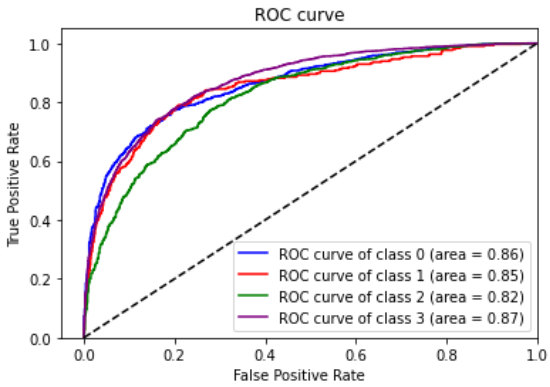
Training F1_score = 0.9762

Validation F1_score = 0.7034

Παρατηρούμε ότι χρησιμοποιώντας την MSE υπολογίζει παρόμοιες τιμές με την L1 function αλλά με λίγο περισσότερο overfitting γιατί τα δεδομένα μας περιέχουν πολλές ακραίες τιμές.

Τώρα θα χρησιμοποιήσουμε την cross entropy loss. Από το documentation βλέπουμε ότι η συνάρτηση αυτή εκτελεί nn.LogSoftmax() και nn.NLLLoss() μαζί, οπότε δεν θα χρησιμοποιήσουμε την softmax σαν output στην forward συνάρτηση του νευρωνικού δικτύου. Όμως στο τέλος πρέπει τα δεδομένα που βγάζουμε για την σχεδίαση γράφων να τα περνάμε από softmax συνάρτηση ώστε να είναι στην ίδια κλίμακα με τα άλλα.





Training loss = 0.1699

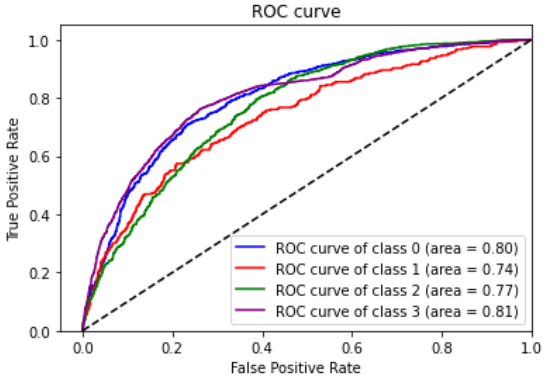
Validation loss = 0.6893

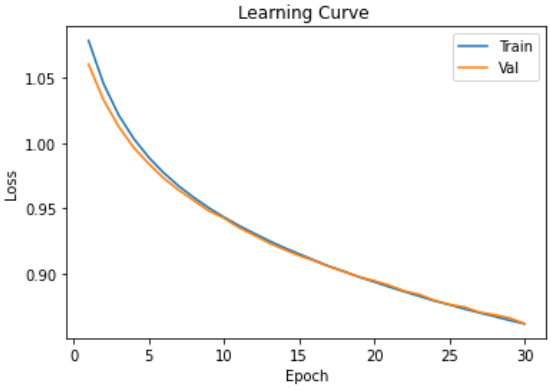
Training F1_score = 0.9673

Validation F1_score = 0.7103

Από τα γραφήματα καταλαβαίνουμε ότι με την χρήση cross entropy στο σύστημα γίνεται περισσότερο overfitting με το MSE ενώ έχουν ακόμα το ίδιο f1 score.

Αφού βλέπουμε ακόμα overfitting και το σύστημα είναι πολύ μακριά από το τέλειο θα δοκιμάσουμε διαφορετικό optimizer.Οπότε με την χρήση του SGD optimizer και Cross Entropy Loss function έχουμε:





Training loss = 0.8612

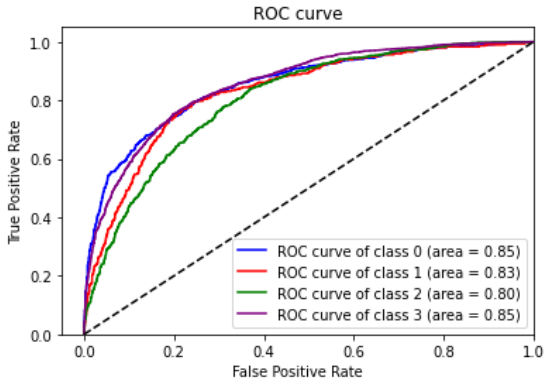
Validation loss = 0.8614

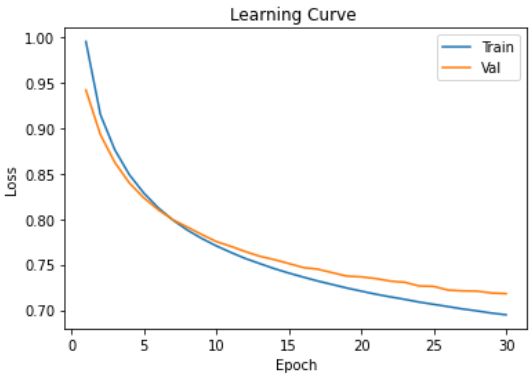
Training F1_score = 0.5957

Validation F1_score = 0.5959

Παρατηρούμε ότι δεν έχει καθόλου overfitting αλλά τώρα το loss είναι στους αριθμούς των 0.86, το οποίο για προφανώς δεν είναι επιτρεπτό. Για τον λόγο αυτόν θα δοκιμάσουμε να μεγαλώσουμε τον learning rate για να αλλάζει πιο γρήγορα τα βάρη ώστε να μάθει καλύτερα.

Οπότε χρησιμοποιώντας Cross Entropy Loss function με learning rate 0.01 :





Στο τελευταίο epoch έχουμε τα εξής στοιχεία:

Training loss = 0.6949

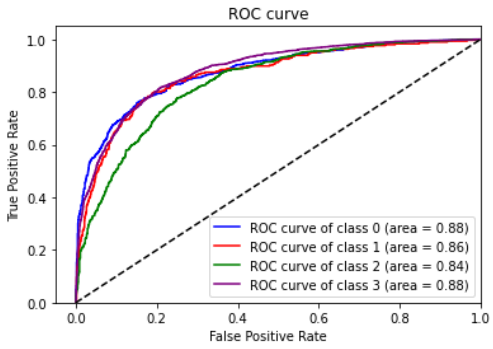
Validation loss = 0.7181

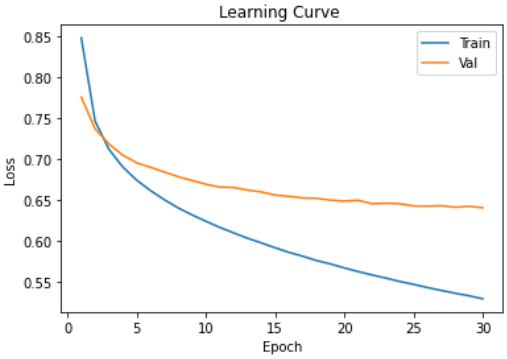
Training F1_score = 0.6813

Validation F1_score = 0.6705

Παρατηρούμε ότι το δίκτυο άρχισε να κάνει λίγο overfitting αλλά οι τιμές των loss κατέβηκαν σημαντικά. Οπότε θα συνεχίσουμε να αυξάνουμε το learning rate

Δοκιμάζοντας με learning rate 0.1 έχουμε:





Στο τελευταίο epoch έχουμε τα εξής στοιχεία:

Training loss = 0.5286

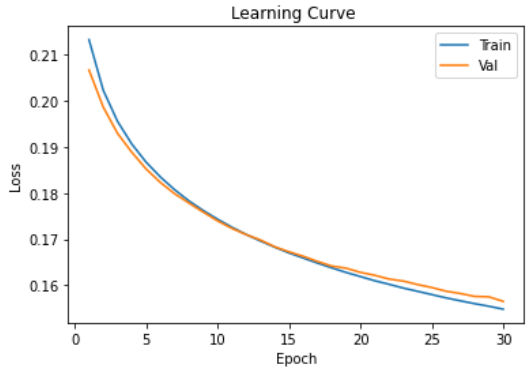
Validation loss = 0.6400

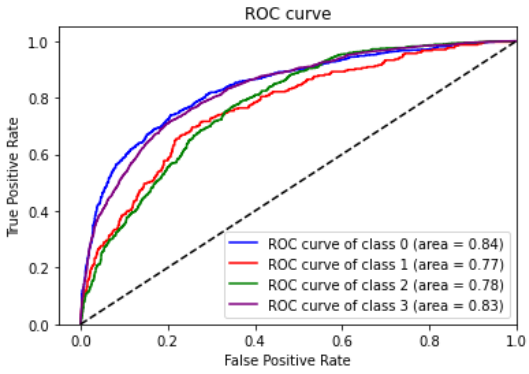
Training F1_score = 0.7889

Validation F1_score = 0.7256

Από τα γραφήματα παρατηρούμε ότι το f1 score αυξήθηκε αλλά χειροτέρεψε κατά πολύ το loss λόγω overfitting. Εν τέλει βλέπουμε ότι για learning rate 0.01 βρίσκουμε τα καλύτερα αποτελέσματα.

Από προηγούμενα πειράματα καταλήξαμε ότι η MSE υπολογίζει καλά loss για το train οπότε θα την χρησιμοποιήσομε. Επίσης με την SGD βλέπουμε το λιγότερο overfitting στα μοντέλα μας άρα θα δοκιμάσουμε τον συνδυασμό των 2.





Training loss = 0.1548

Validation loss = 0.1565

Training F1_score = 0.6212

Validation F1_score = 0.6190

Η θεωρία μας επιβεβαιώθηκε από το πείραμα αφού δεν παρουσιάζει σχεδόν καθόλου overfitting με καλές τιμές f1 score για το train και validation set και καλή αναλογία σωστών αποτελεσμάτων από το ROC curve.

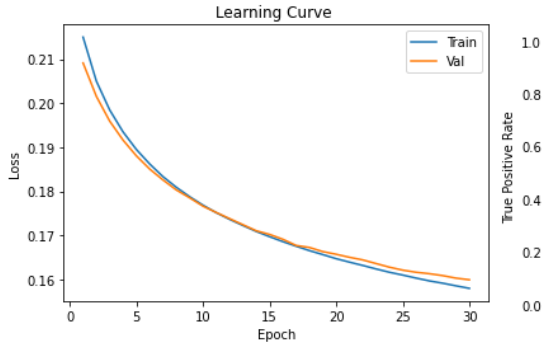
Όμως τα αποτελέσματα αυτά αν και καλά έχουν μερικά βασικά λάθη, όπως ο χρόνος εκτέλεσης γιατί παρουσιάζει υπερβολικά πολλά στοιχεία. Το πρόβλημα αυτό μπορεί να λυθεί με preprocessing.

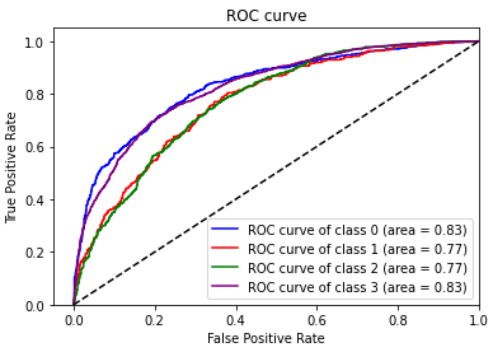
Κάνουμε preprocessing τα δεδομένα οπότε αφαιρούμε τα:

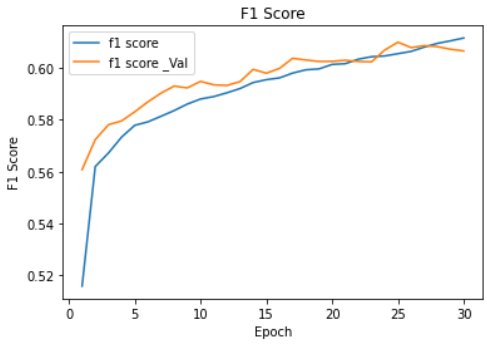
- Whitespaces μέσω την συνάρτηση strip()
- Links και URLs
- Emojis και σύμβολα
- Σημεία στίξης
- Accents με το module unicodedata
- νούμερα

Οπότε καταλήγουμε σε 26254 ξεχωριστές λέξεις.

Ξανατρέχουμε το καλύτερο πείραμα από πριν, δηλαδή SGD με MSE και 0.01 learning rate και υπολογίζει τα εξής στοιχεία







Στο τελευταίο epoch έχουμε τα εξής στοιχεία:

Training loss = 0.1579

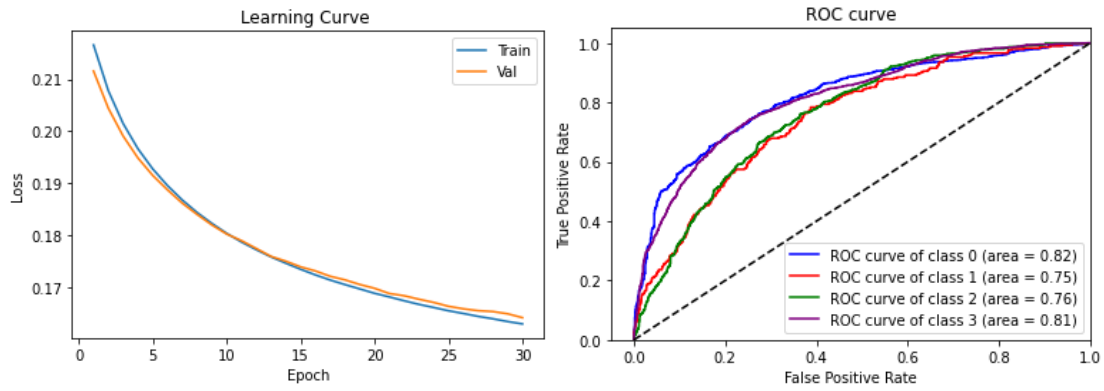
Validation loss = 0.1599

Training F1_score = 0.6114

Validation F1_score = 0.6064

Βλέπουμε ότι τα δεδομένα μας είναι παρόμοια με τα προηγούμενα αφού καταρχάς δεν αφαιρέσαμε πολλές λέξεις και εκείνες που βγάλαμε δεν είχαν μεγάλη βαρύτητα.

Τα αποτελέσματα όμως παίρνουν πολύ ώρα να υπολογιστούν οπότε μπορούμε να ασχοληθούμε με max_features 1000 δηλαδή με τις 1000 λέξεις που βρίσκονται περισσότερες φορές στα tweets. Η παράμετρος αυτή δεν πρέπει να αλλάξει πολύ το σύστημα αφού οι λέξεις που αφαιρέσαμε δεν εμφανίζονται πολλές φορές οπότε δεν είναι και σημαντικές.



Training loss = 0.1629

Validation loss = 0.1641

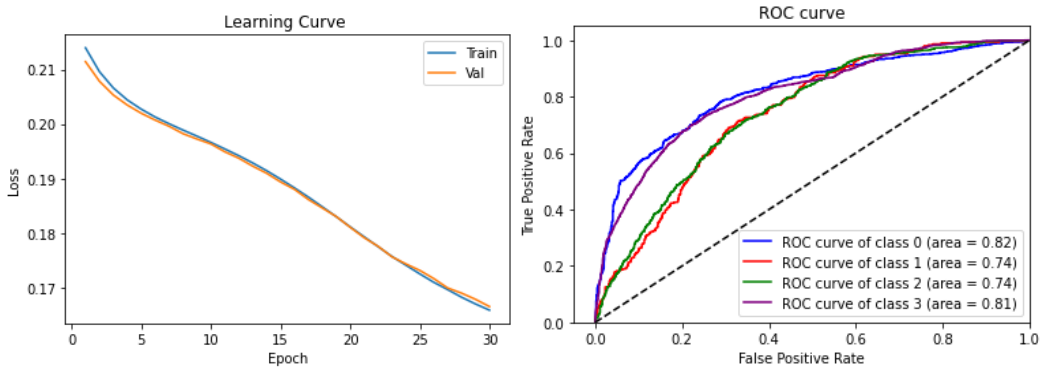
Training F1_score = 0.6027

Validation F1_score = 0.6034

Παρατηρούμε ότι η αλλαγή αυτή δεν άλλαξε τα αποτελέσματα για τον λόγο που αναφέρθηκε πριν.

Τώρα με την μείωση του χρόνου μας δίνει την δυνατότητα να ασχοληθούμε με hidden layers.

Αρχικά θα εκτελέσουμε το πρόγραμμα για 1 hidden layer με 100 κόμβους.



Training loss = 0.1658

Validation loss = 0.1665

Training F1_score = 0.5966

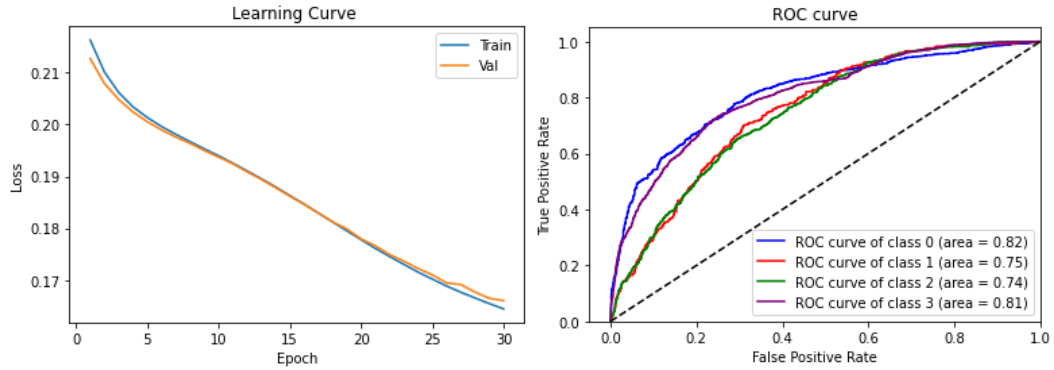
Validation F1_score = 0.5995

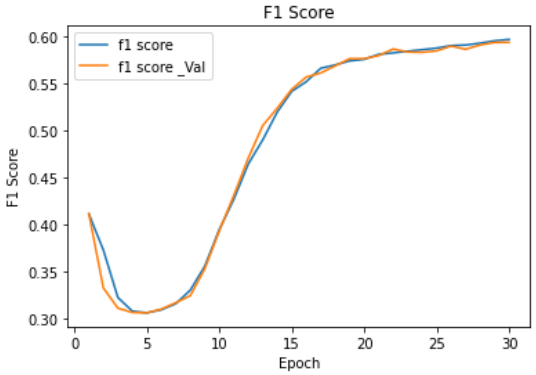
Οι μετρήσεις παραμένουν σχεδόν ίδιες με το προηγούμενο παράδειγμα με μια μικρή διαφορά των f1 score αλλά μέσω της learning curve καταλαβαίνουμε ότι θα αυξανόταν.

Από τα γραφήματα βλέπουμε ότι το learning curve κατεβαίνει πολύ πιο δραματικά από το προηγούμενο παράδειγμα που σημαίνει ότι για περισσότερο χρόνο (μεγαλύτερο αριθμό από epoch) θα βγάλει καλύτερο αποτέλεσμα. Οπότε κρατάμε αυτήν την σχεδίαση για τώρα.

Θα αυξήσουμε τον αριθμό των κόμβων για να κρατάει περισσότερα χαρακτηριστικά και να γίνει πιο περίπλοκο το πρόγραμμα ώστε να βγάζει και καλύτερα αποτελέσματα.

Οπότε για 500 κόμβους με 1 hidden layer έχουμε:





Training loss = 0.1644

Validation loss = 0.1660

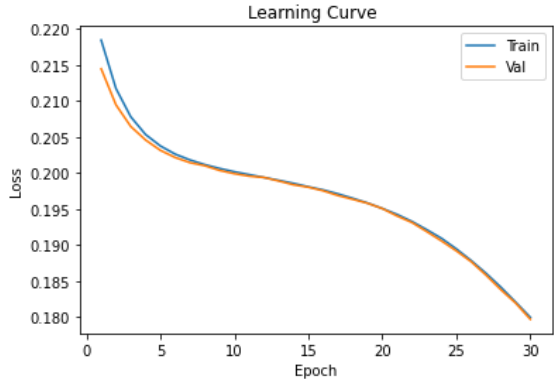
Training F1_score = 0.5972

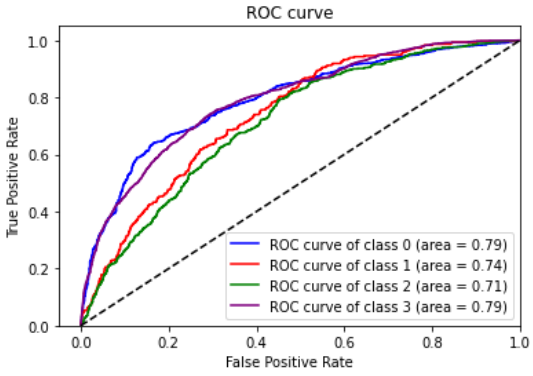
Validation F1_score = 0.5942

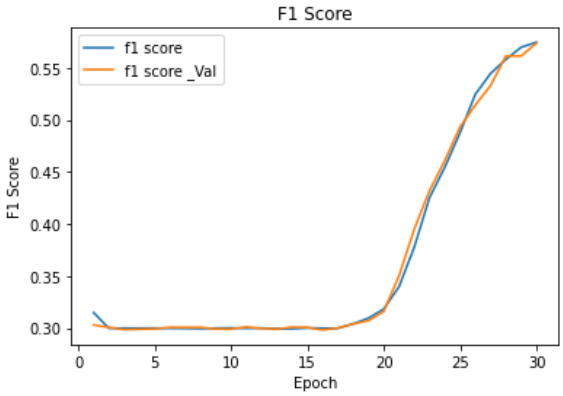
Με την αλλαγή αυτή δεν καταλαβαίνουμε κάποια σημαντική διαφορά μάλλον γιατί τα δεδομένα μας δεν είναι αρκετά περίπλοκα για να χρειάζεται και άλλους κόμβους αλλά την θεωρία αυτή θα την εξετάσουμε και στα επόμενα παραδείγματα.

Οπότε τώρα θα πειραματιστούμε με 2 hidden layers των 500 για να δούμε αν το σύστημα είναι υπερβολικά περίπλοκο για τα δεδομένα.

Για 2 hidden layers με 500 κόμβους έχουμε:

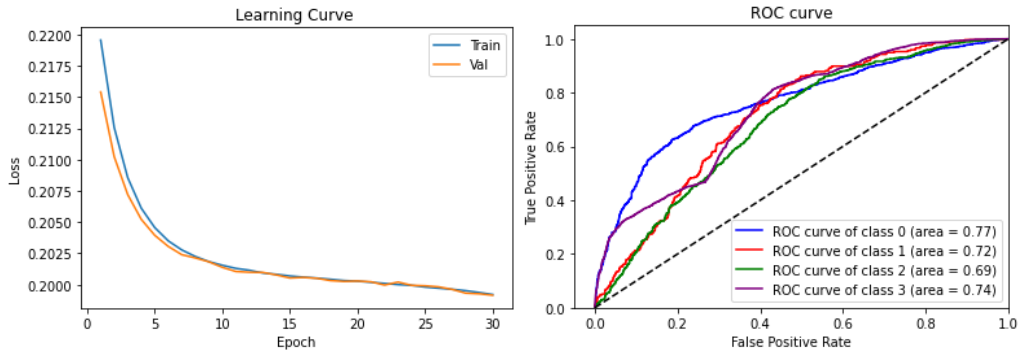


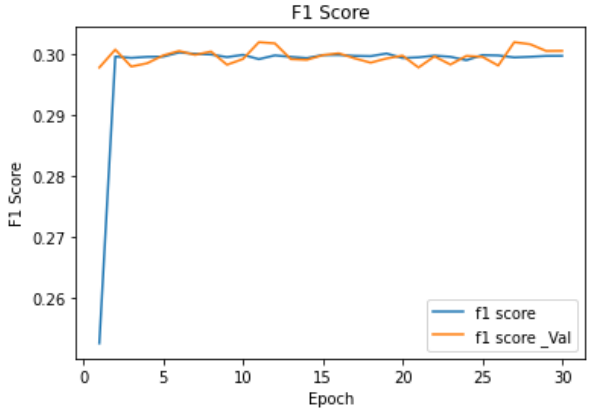




Παρατηρούμε ότι έχει ακόμα λιγότερο overfitting αλλά μέσω του γράφου με f1 scores χρειάζεται περισσότερα δεδομένα για να λειτουργήσει σωστά.

Αφού δεν βλέπουμε κάποια μεγάλη διαφορά θα δοκιμάσουμε 3 hidden layers των 500 κόμβων.





Training loss = 0.1992

Validation loss = 0.1991

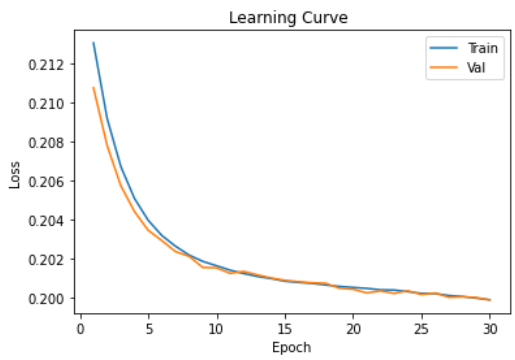
Training F1_score = 0.2996

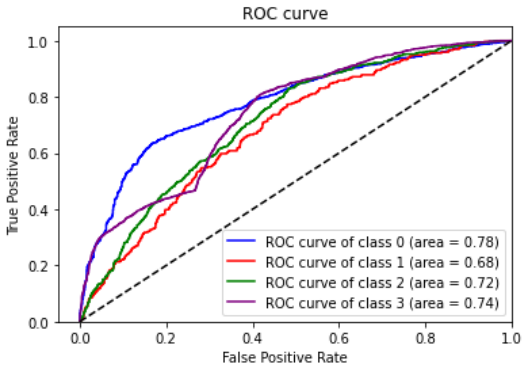
Validation F1_score = 0.3004

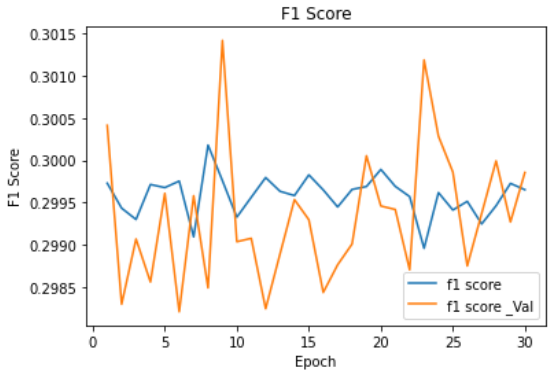
Παρατηρούμε ότι ενώ τα loss για το train και validation είναι καλά έχουν πέσει πολύ τα f1 scores οπότε το πρόγραμμα δεν μπορεί τόσο εύκολα να μάθει τα χαρακτηριστικά των δεδομένων αφού έχει υπερβολικά πολλούς κόμβους άρα και βάρη.

Άρα μια ιδέα είναι να δοκιμάσουμε για 3 hidden layers με 50 κόμβους για να έχουμε λιγότερα συνολικά βάρη.

Άρα 3 hidden με 50 κόμβους ο καθένας:







Training loss = 0.1998

Validation loss = 0.1998

Training F1_score = 0.2996

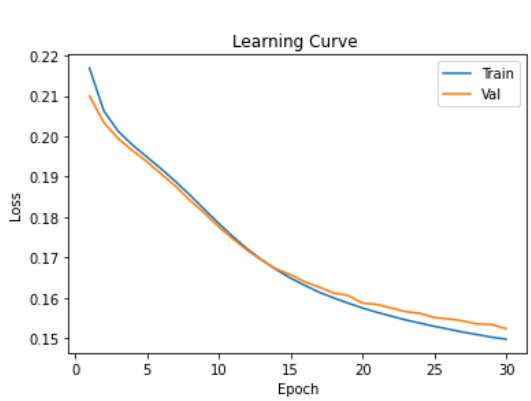
Validation F1_score = 0.2998

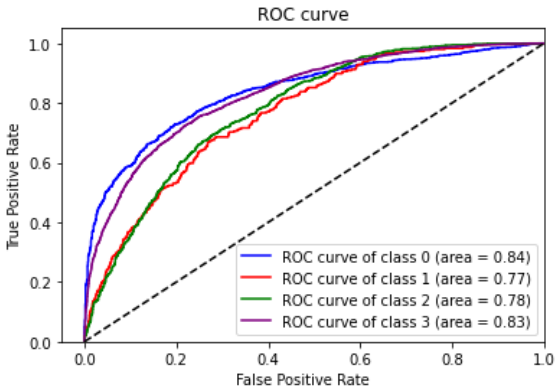
Από τα δεδομένα βλέπουμε ότι συνεχίζει να υπάρχει το ίδιο πρόβλημα με το προηγούμενο πείραμα, οπότε η χρήση 3 hidden layers δεν είναι σωστή υλοποίηση.

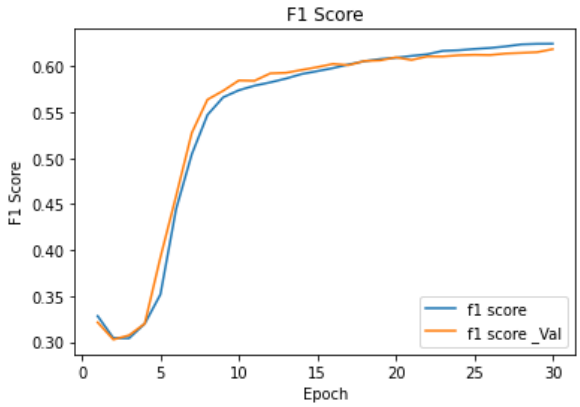
Οπότε θα χρησιμοποιήσουμε πάλι το καλύτερο παράδειγμα με ένα hidden layer των 500 κόμβων έχουμε.

Τώρα θα πειραματιστούμε με τις παραμέτρους του SGD:

Χρησιμοποιώντας momentum 0.5, η οποία θυμάται τα προηγούμενα δεδομένα και τα αλλάζει όταν βλέπει ότι κατευθύνεται σε αυτά.







Training loss = 0.1497

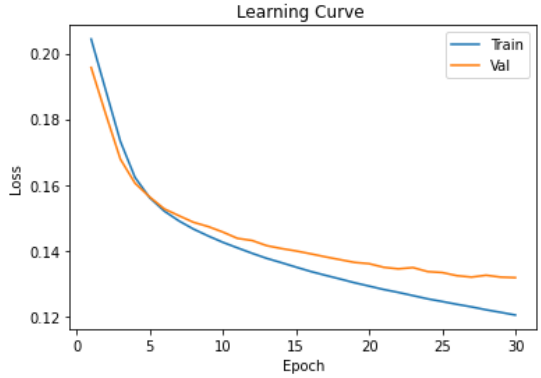
Validation loss = 0.1523

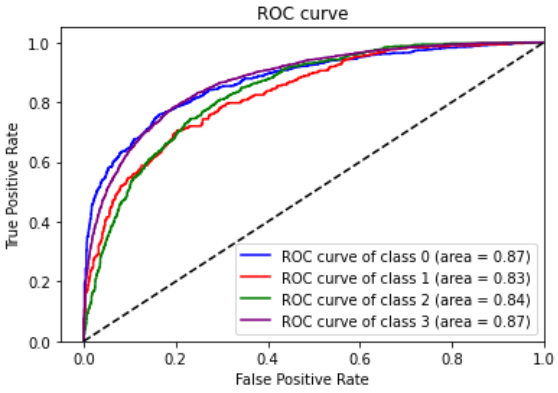
Training F1_score = 0.6245

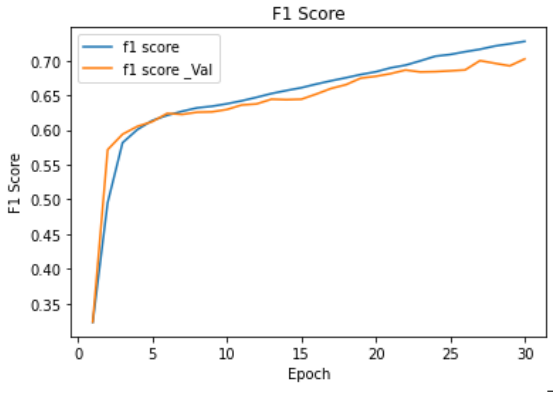
Validation F1_score = 0.6186

Παρατηρούμε καλύτερα αποτελέσματα στο f1 score ενώ με μια μικρή τάση για overfitting οπότε θα πειραματιστούμε και με υψηλότερες τιμές.

Με momentum 0.9:







Training loss = 0.1203

Validation loss = 0.1317

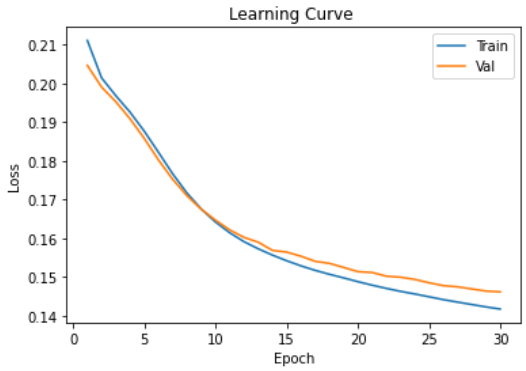
Training F1_score = 0.7276

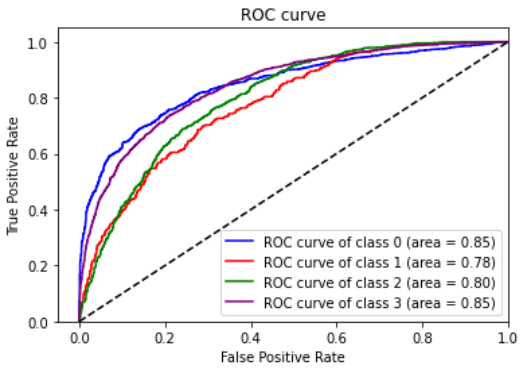
Validation F1_score = 0.7023

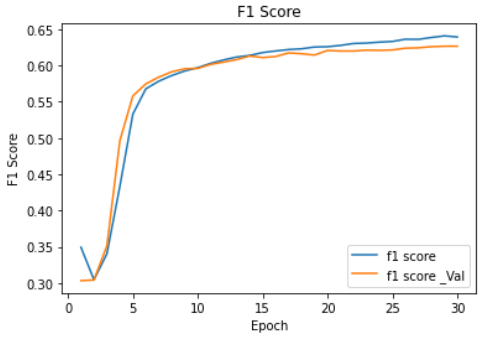
Βλέπουμε ότι κατεβαίνουν τα train και validation loss και ανεβαίνουν αρκετά τα f1 scores με καλή καμπύλη στην ROC curve με τάση για overfitting . Οπότε καταλήγουμε ότι αυτή η παράμετρος είναι πολύ καλή για το σύστημα μας.

Για να λύσουμε το μικρό πρόβλημα του overfitting θα χαμηλώσουμε λίγο το momentum για να δούμε τις διαφορές.

Οπότε με 0.7 momentum







Training loss = 0.1417

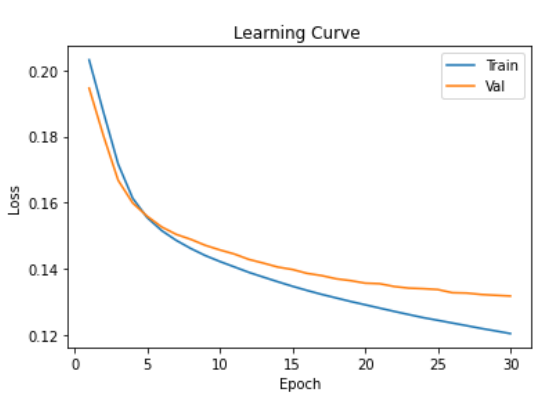
Validation loss = 0.1462

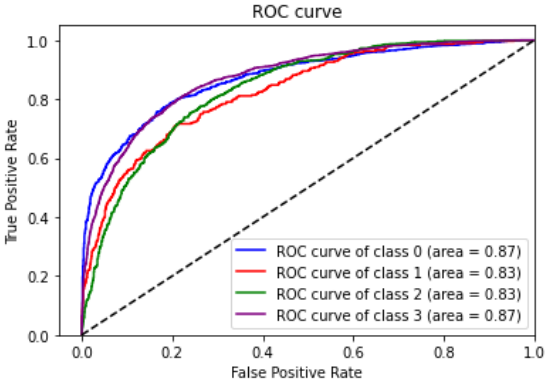
Training F1_score = 0.6391

Validation F1_score = 0.6264

Ξαναπέφτει το f1 score λύνοντας όμως το overfitting. Οπότε θα χρησιμοποιούμε momentum 0.9 για τα επόμενα πειράματα επειδή είχε υψηλές μετρήσεις.

Δοκιμάζοντας nesterov:





Training loss = 0.1205

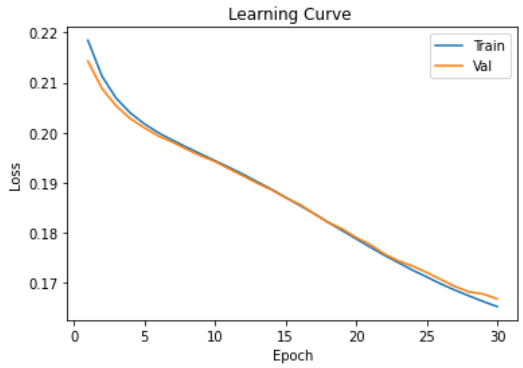
Validation loss = 0.1318

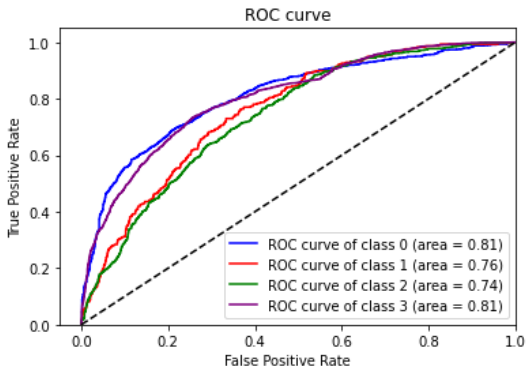
Training F1_score = 0.7268

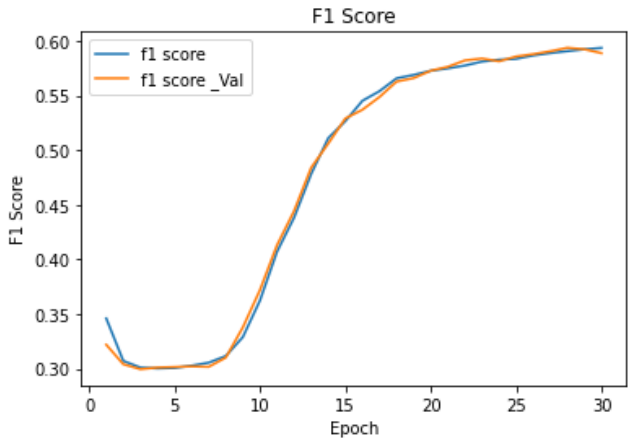
Validation F1_score = 0.6984

Βλέπουμε μεγαλύτερο overfitting οπότε δεν θα το διαλέξουμε.

Θα δοκιμάσουμε dampening 0.9 οπότε με momentum 0.9:







Training loss = 0.1652

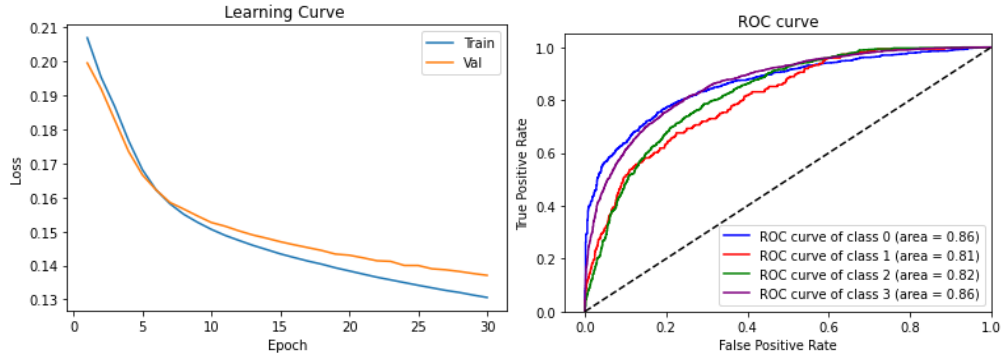
Validation loss = 0.1667

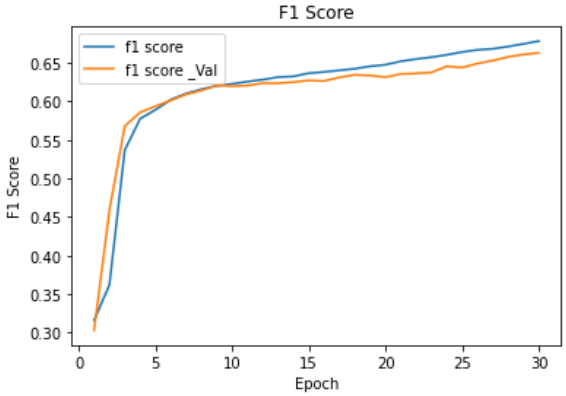
Training F1_score = 0.5935

Validation F1_score = 0.5887

Βλέπουμε μεγάλη κάθοδο στα f1 scores αλλά χωρίς overfitting και κλίση να κατεβαίνει στο learning curve οπότε είναι καλό.

Με dampening 0.4:





Training loss = 0.1305

Validation loss = 0.1370

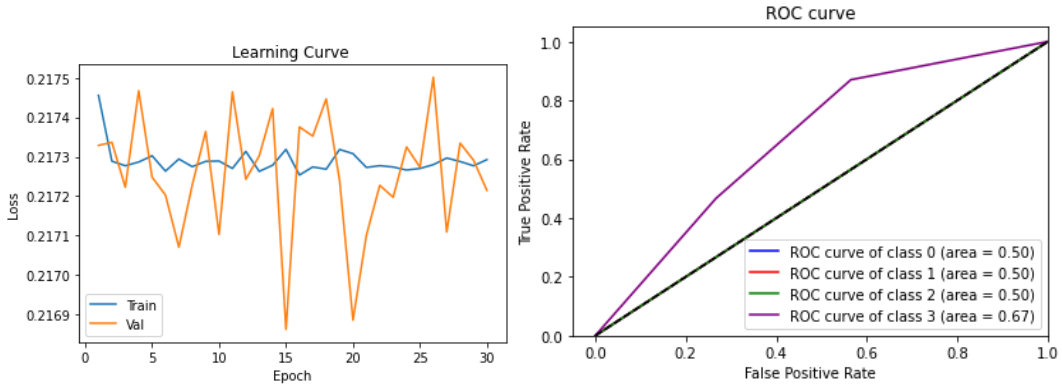
Training F1_score = 0.6781

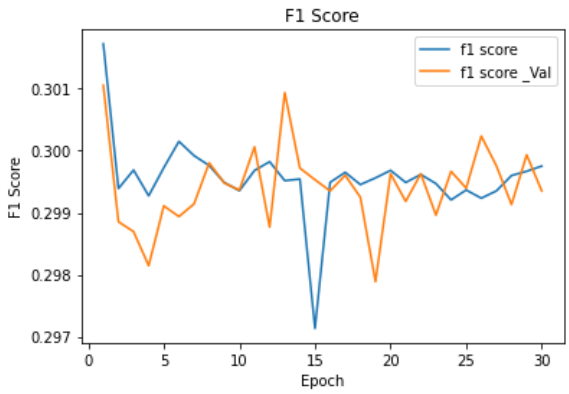
Validation F1_score = 0.6628

Με dampening 0.4 είναι λίγο μεγαλύτερα τα f1 scores αλλά ξαναφαίνεται λίγο overfitting οπότε θα προτιμήσουμε χωρίς καθόλου dampening ώστε να μην μειωθούν τα scores.

Θα πειραματιστούμε με την παράμετρο weight decay

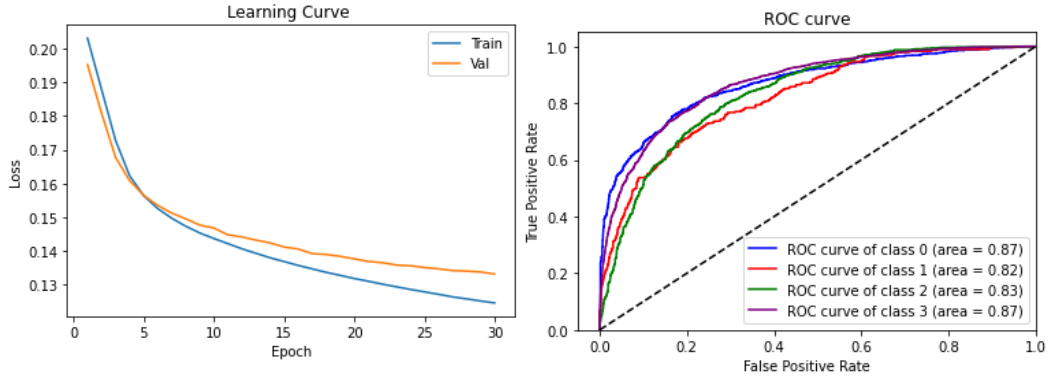
Με weight decay = 0.5 έχουμε:

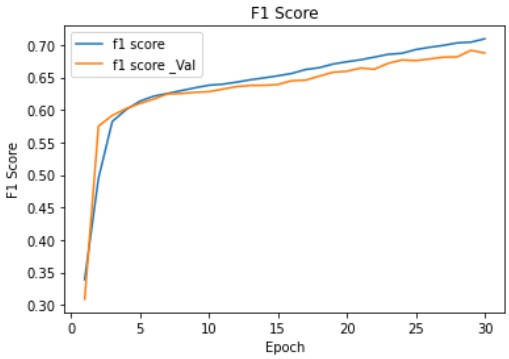




Για προφανή λόγους το μεγάλο weight decay δεν είναι καλό για το μοντέλο αφού θα μηδενίζει πολλά βάρη.

Οπότε θα δοκιμάσουμε για πιο λογικά weight decays όπως 0.0005





Training loss = 0.1244

Validation loss = 0.1330

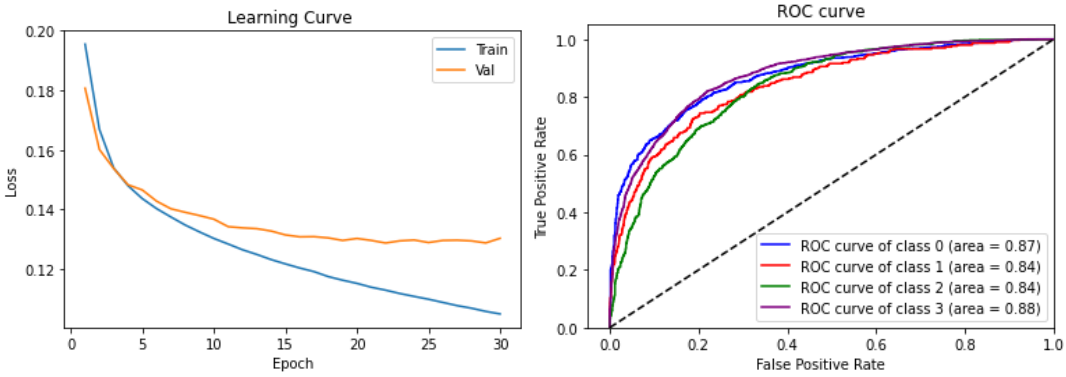
Training F1_score = 0.7096

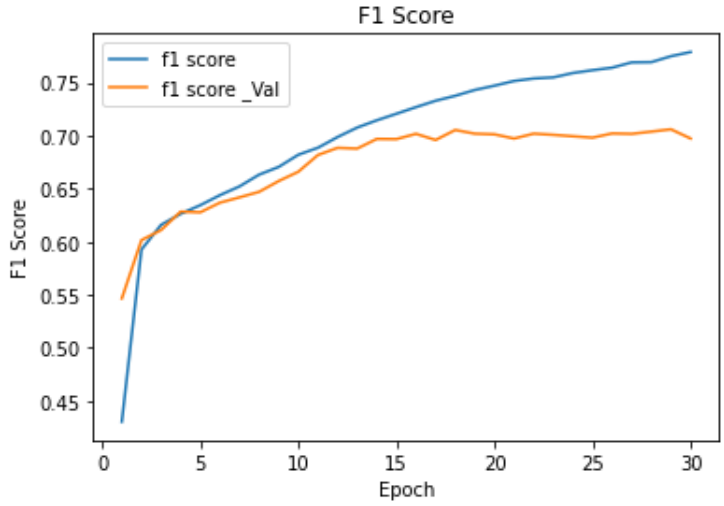
Validation F1_score = 0.6877

Δεν παρατηρούμε κάποια σημαντική αλλαγή με το πείραμα χωρίς weight decay οπότε δεν θα τον χρησιμοποιήσουμε.

Τώρα για πιο μικρά batch sizes το πρόγραμμα θα αργεί περισσότερο στην εκτέλεση και το μοντέλο δέχεται λιγότερα δεδομένα την φορά για να μάθει. Οπότε

Με 32 batch size:





Training loss = 0.1049

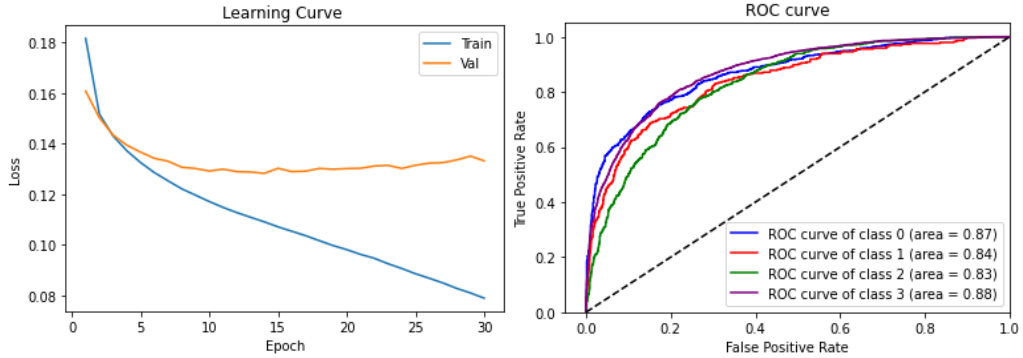
Validation loss = 0.1303

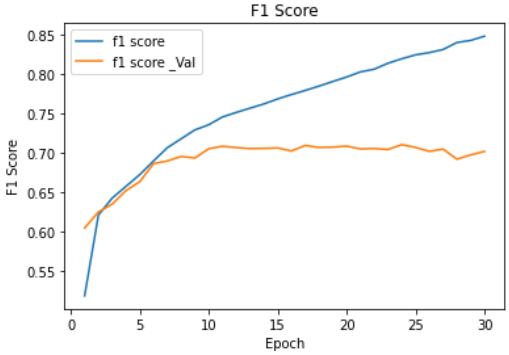
Training F1_score = 0.7787

Validation F1_score = 0.6972

Παρατηρούμε σημαντική μείωση των loss και αύξηση του f1 score αλλά έχει overfitting το οποίο είναι λογικό αφού μαθαίνει με λιγότερες τιμές την φορά.

Με 16 batch:





Training loss = 0.07903

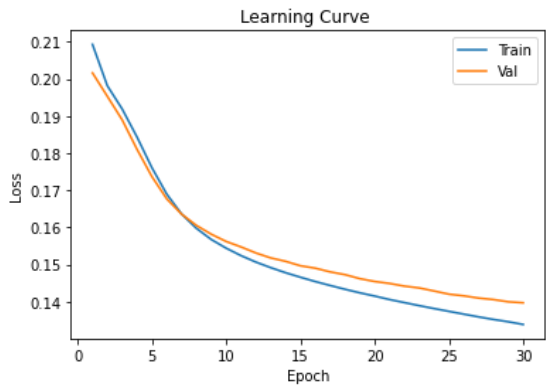
Validation loss = 0.1331

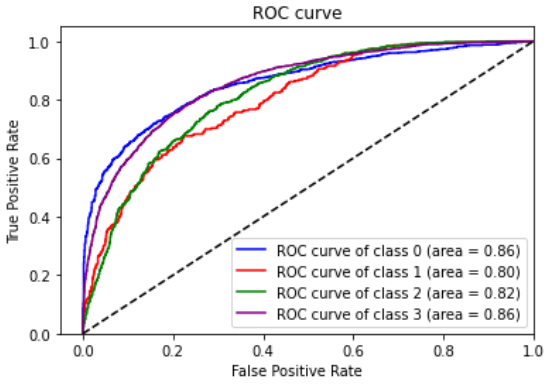
Training F1_score = 0.8475

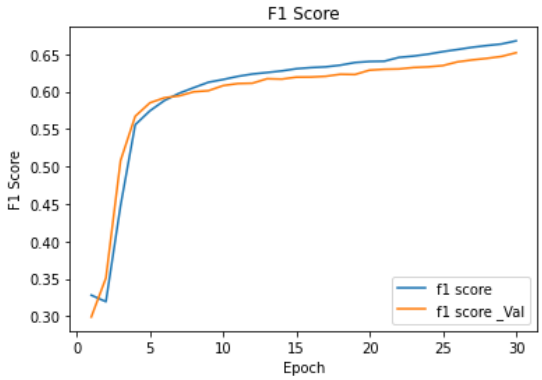
Validation F1_score = 0.7012

Προφανώς αν μικρύνουμε και άλλο το batch size θα έχει ακόμα περισσότερο overfitting.

Τώρα θα δοκιμάσουμε να αυξήσουμε το batch size se 128







Training loss = 0.13383661782741546

Validation loss = 0.13967910698718494

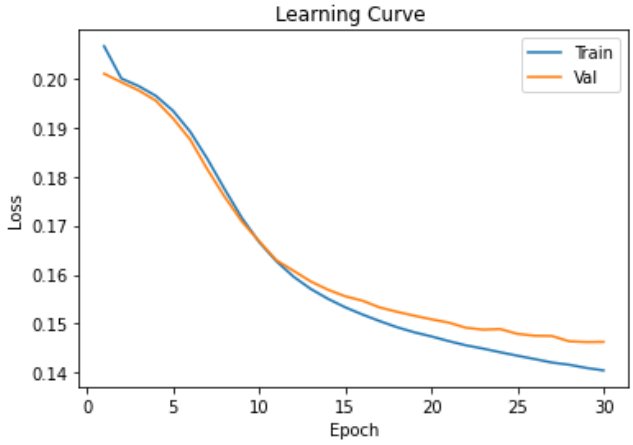
Training F1_score = 0.667844637618506

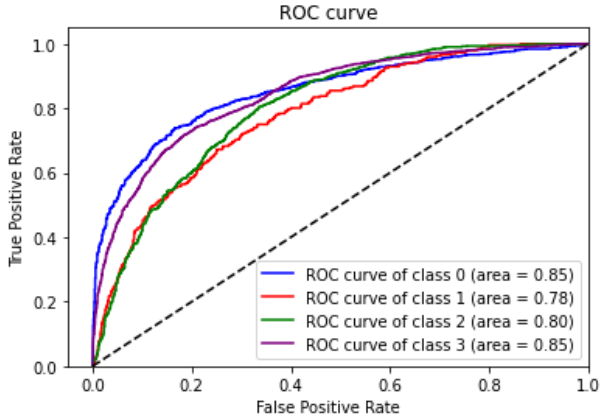
Validation F1_score = 0.6519301214090296

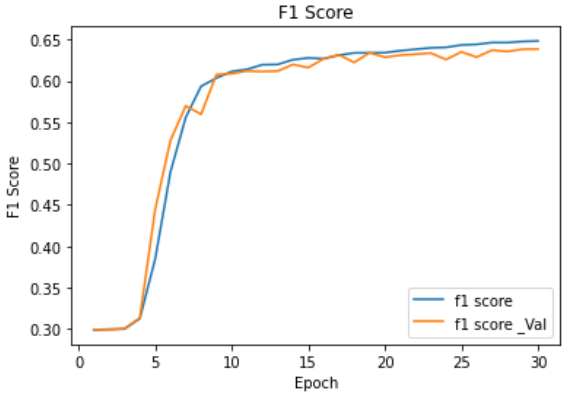
Βλέπουμε ότι οι τιμές των f1 scores πέφτουν αφού του βάζουμε κάθε φορά να μάθει με υπερβολικά πολλά δεδομένα. Όμως ο χρόνος εκτέλεσης μειώθηκε.

Τώρα θα δοκιμάσουμε με tfidf που είναι σαν τον counter με έναν αλγόριθμό normalize

TFIDF vectorizer:





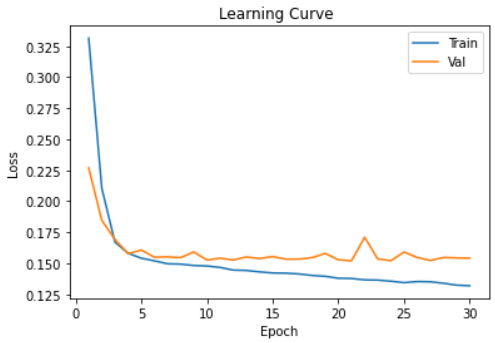


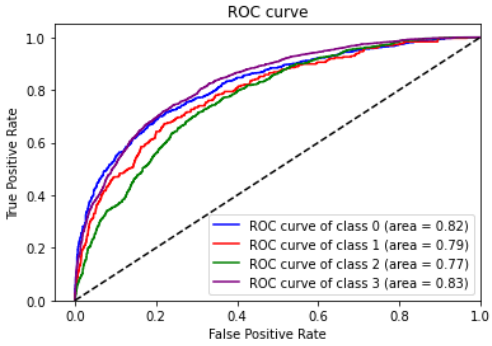
Training loss = 0.14047052127122878 Validation loss = 0.14628545629481474

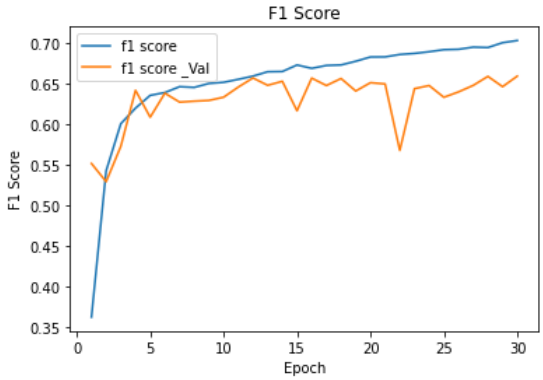
Training F1_score = 0.6484660637230011 Validation F1_score = 0.6386040655472346

Παρατηρούμε περίπου τις ίδιες μετρήσεις αλλά με λιγότερο overfitting που είναι λογικό αφού δέχεται μερικές λέξεις που είναι πιο σημαντικές για τον καθορισμό των προτάσεων.

Τώρα θα χρησιμοποιήσουμε GloVe για αρχή με 50 διαστάσεις χρησιμοποιούμε το καλύτερο μοντέλο από πριν με glove:





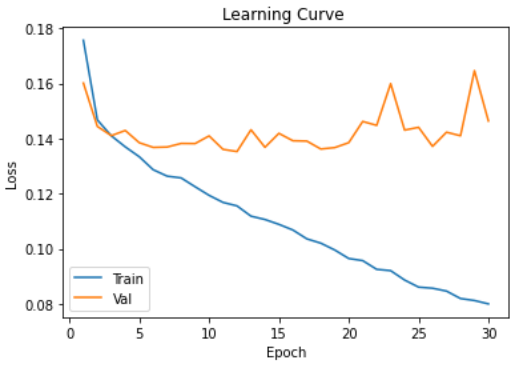


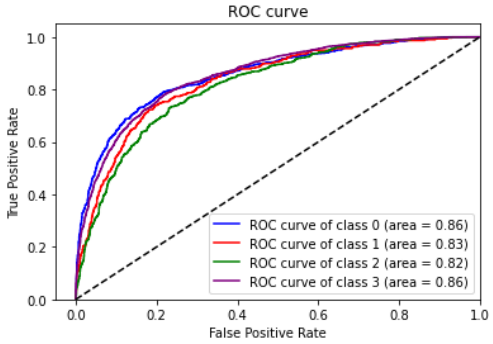
Training loss = 0.13189456504583358 Validation loss = 0.15420825800134075

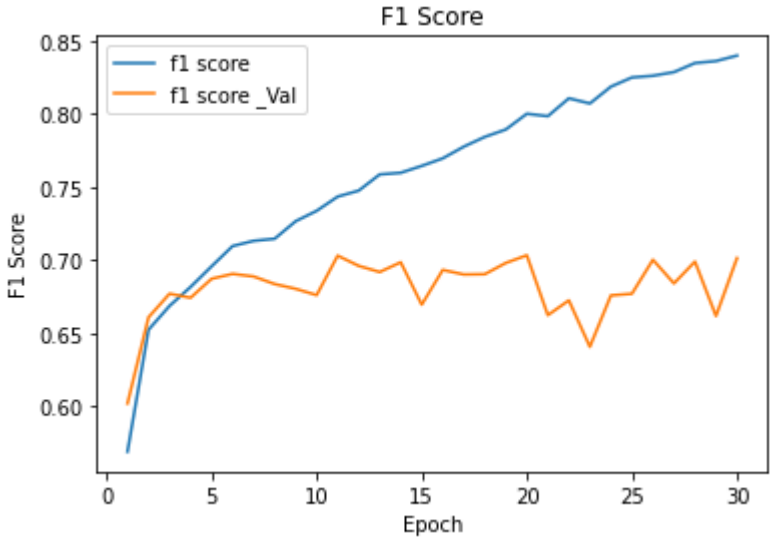
Training F1_score = 0.703314107588502 Validation F1_score = 0.6593411546403201

Βλέπουμε ότι το μοντέλο αυτό είναι κάπως χειρότερο από το προηγούμενο και αυτό οφείλεται γιατί κατά την δημιουργία των tweets σε πίνακα μιας στήλης προσθέσαμε κάθε νούμερο των γραμμών σε 1. Αυτό προφανώς δεν είναι καλή υλοποίηση αφού χάνονται πολλές χρήσιμες πληροφορίες.

Θα δοκιμάσουμε τώρα με 300 διαστάσεις:





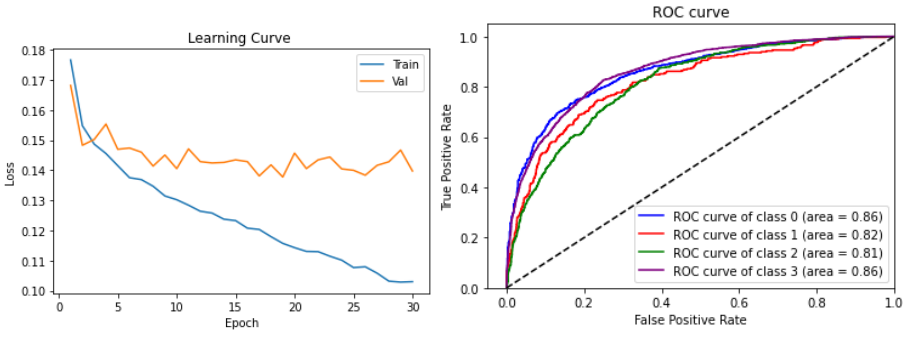


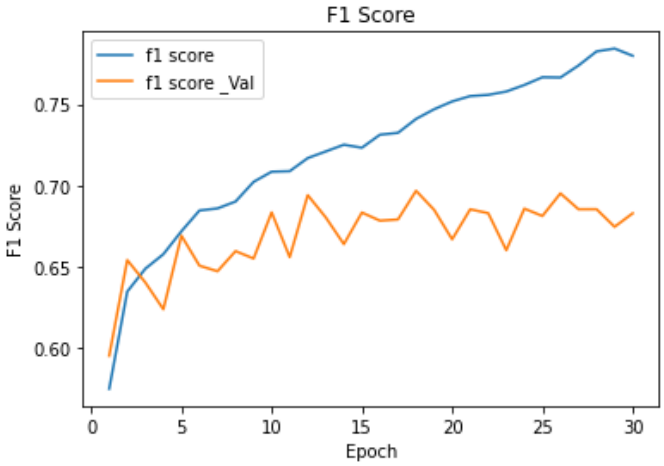
Training loss = 0.08003395217657089 Validation loss = 0.1464030018283261

Training F1_score = 0.8399123093091565 Validation F1_score = 0.7015126695975258

Παρατηρούμε ότι με 300 διαστάσεις έχει μεγάλο overfitting καθώς το learning rate είναι υπερβολικά μεγάλο για το μοντέλο αυτό.

Για GloVe με 200D:



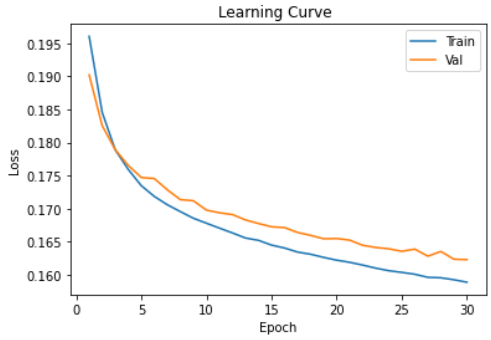


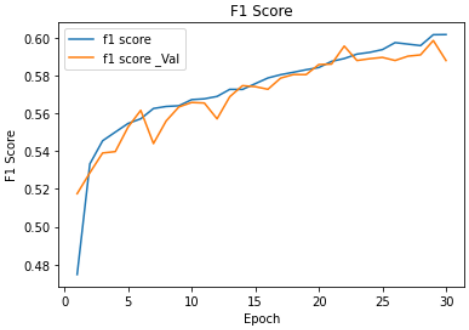
Training loss = 0.102993675917387 Validation loss = 0.13973293577631316

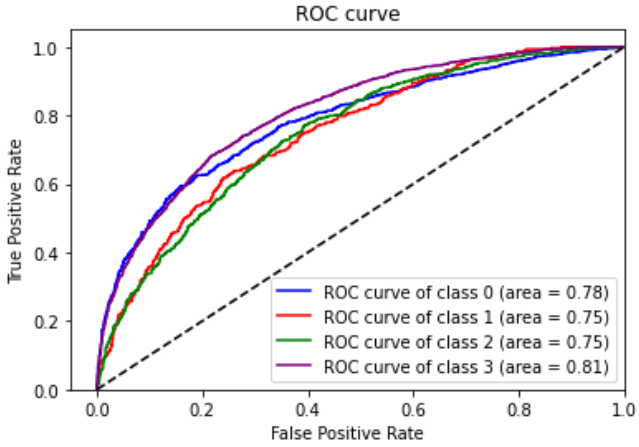
Training F1_score = 0.7798957528259732 Validation F1_score = 0.6827818026641101

Βλέπουμε ότι έχει καλύτερα αποτελέσματα από τις 300 διαστάσεις αλλά και πάλι εμφανίζεται το πρόβλημα με το overfitting.

Οπότε αντί να προσθέτουμε τις γραμμές θα προσπαθήσουμε να τις κάνουμε average σε μια γραμμή





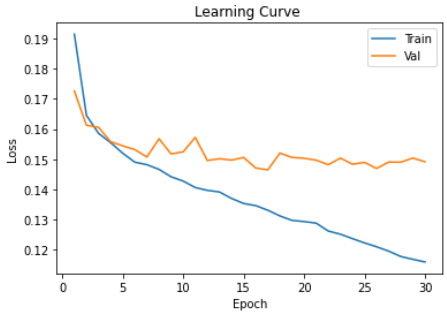


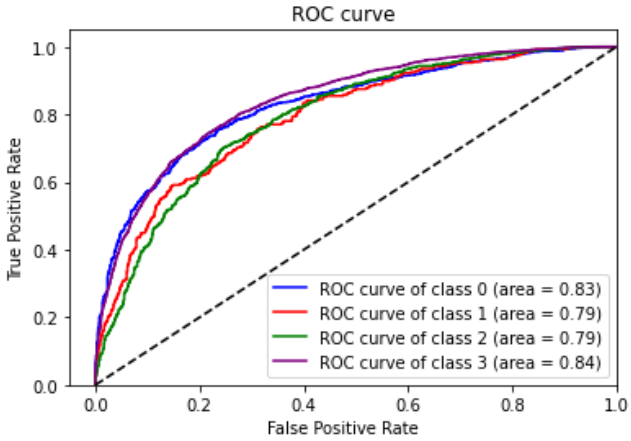
Training loss = 0.15885887917876243 Validation loss = 0.16227383870217535

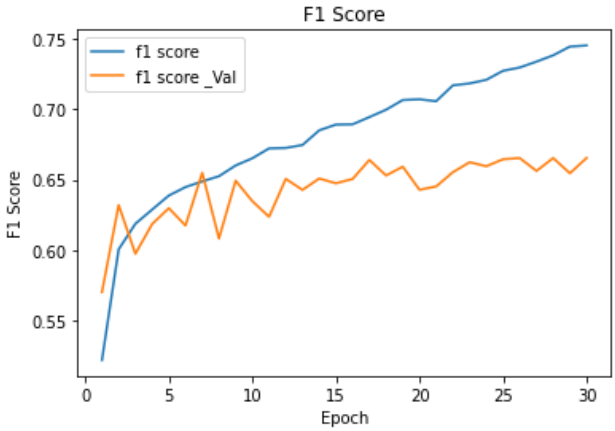
Training F1_score = 0.6015717719559107 Validation F1_score = 0.5878388099912173

Βλέπουμε ότι με κάνοντας average τις τιμές χειροτερεύουν σημαντικά από τα προηγούμενα παραδείγματα οπότε δεν θα το χρησιμοποιήσουμε στα επόμενα.

Με 2 hidden layers H1 = 500 H2 = 250





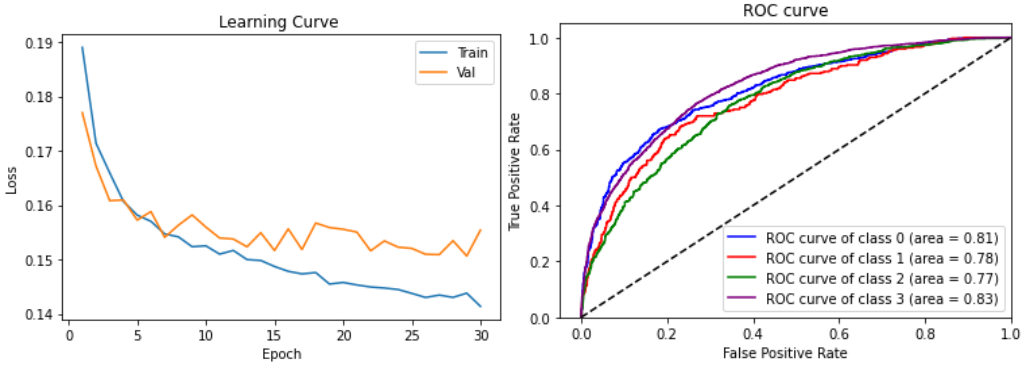


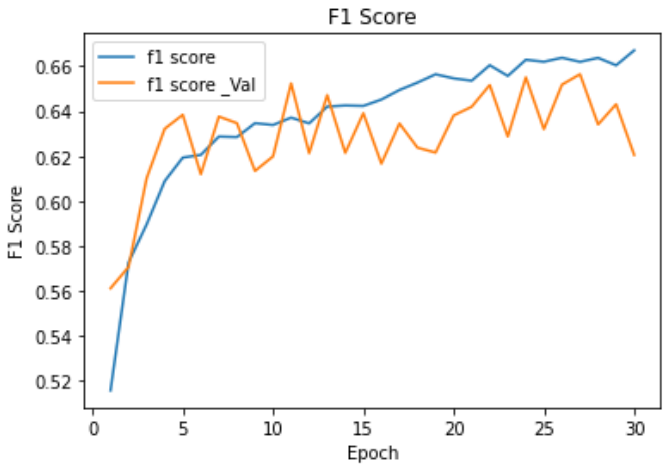
Training loss = 0.11598428440093994 Validation loss = 0.14913691663079792

Training F1_score = 0.7452458794820194 Validation F1_score = 0.6654662213963959

Το μοντέλο μας τώρα έχει πολλούς κόμβους οπότε και πολλά βάρη σε αυτά οπότε το σύστημα είναι υπερβολικά περίπλοκος και έχει overfitting.

Για τον λόγο αυτό θα μειώσουμε τους συνολικούς κόμβους σε 50 και 50



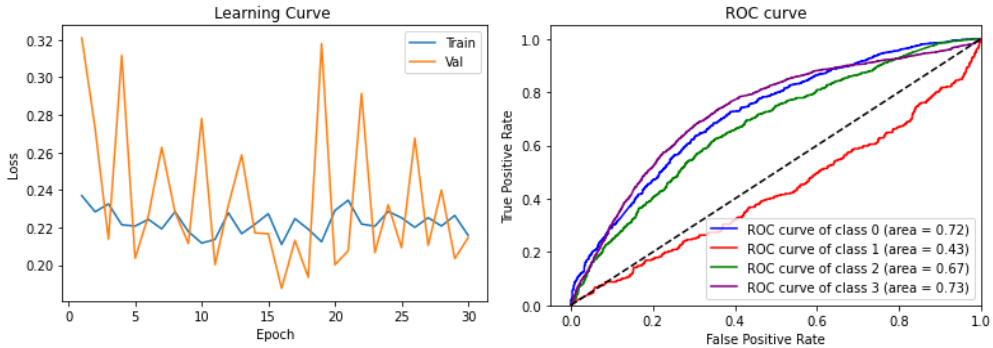


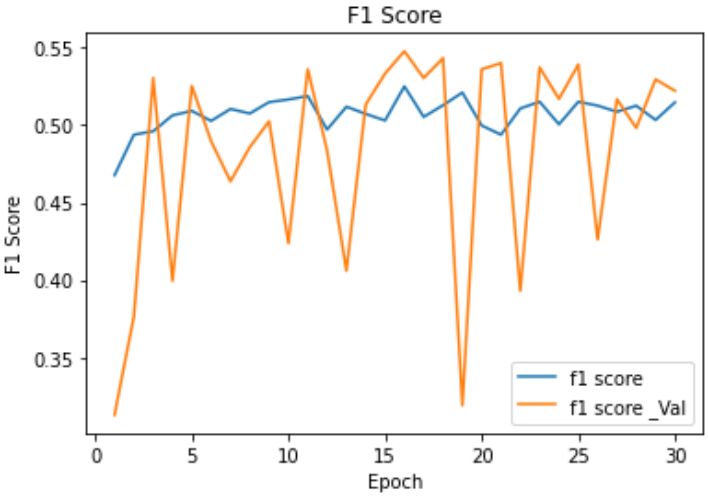
Training loss = 0.14136312448978425 Validation loss = 0.15540794862641227

Training F1_score = 0.6670381225981928 Validation F1_score = 0.6205298315392427

Από το learning curve παρατηρούμε μικρότερη αλλά και πάλι έντονο overfitting και τα f1 score βλέπουμε ότι η πιθανή αιτία είναι το μεγάλο learning rate

Τώρα θα δοκιμάσουμε το άλλο άκρο, δηλαδή κανένα hidden layer



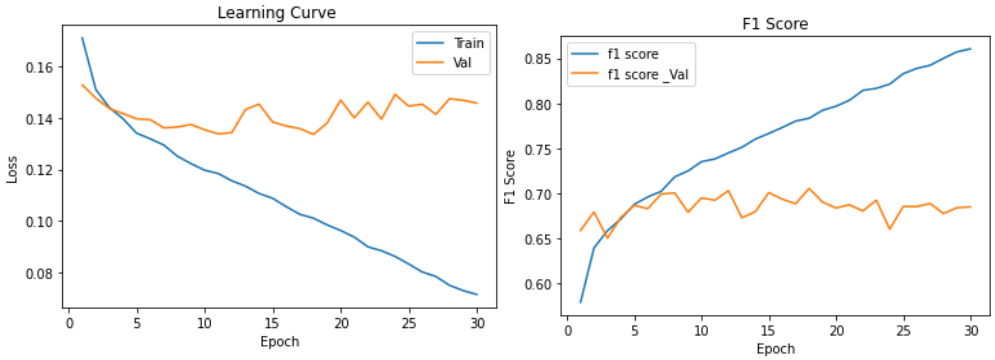


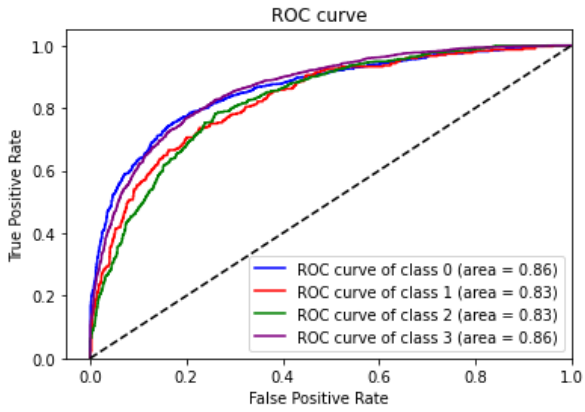
Training loss = 0.21583393579721452 Validation loss = 0.21452618017792702

Training F1_score = 0.5146385918214439 Validation F1_score = 0.5221194955179672

Παρατηρούμε από την f1 score ότι το σύστημα αδυνατεί να μάθει αφού δεν μπορεί να σταθεροποιηθεί σε ένα σημείο

Τώρα θα δοκιμάσουμε να χρησιμοποιήσουμε 300 διαστάσεις με 2 hidden layers των 500 κόμβων





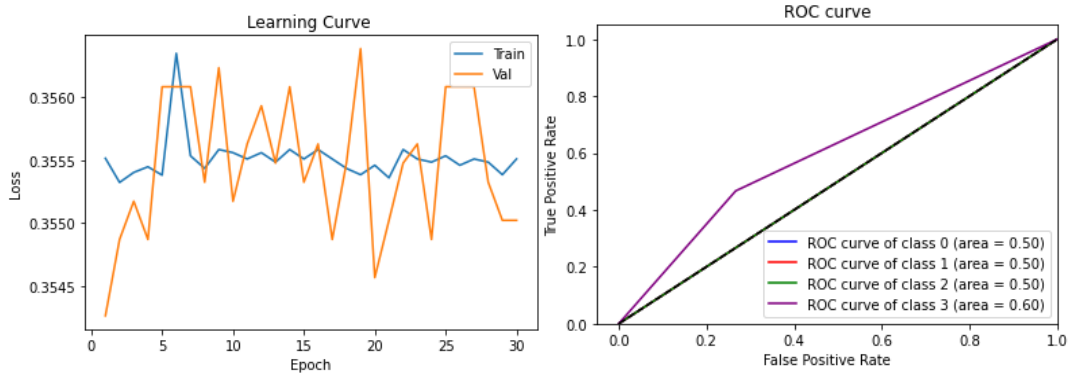
Training loss = 0.071628785058856 Validation loss = 0.14569667747451198

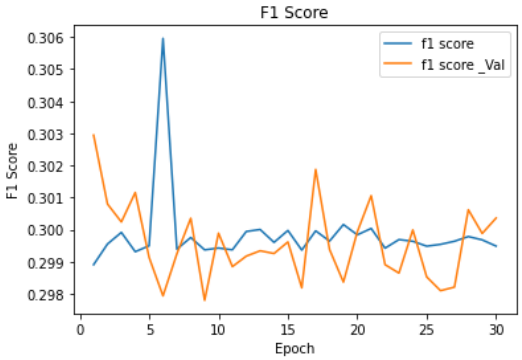
Training F1_score = 0.860479676586431 Validation F1_score = 0.6848338201688111

Βλέπουμε ότι συνεχίζει να έχει πολύ overfitting καθώς υπάρχουν πολλά βάρη από τους κόμβους. Επίσης μια άλλη πιθανή αιτία είναι το learning rate το οποίο είναι υπερβολικά υψηλό και δεν προλαβαίνει να μάθει όλα τα βάρη.

Θα δοκιμάσουμε διαφορετικό optimizer αρχικά με υψηλό learning rate

Οπότε θα δοκιμάσουμε Adam με 0.1 learning rate



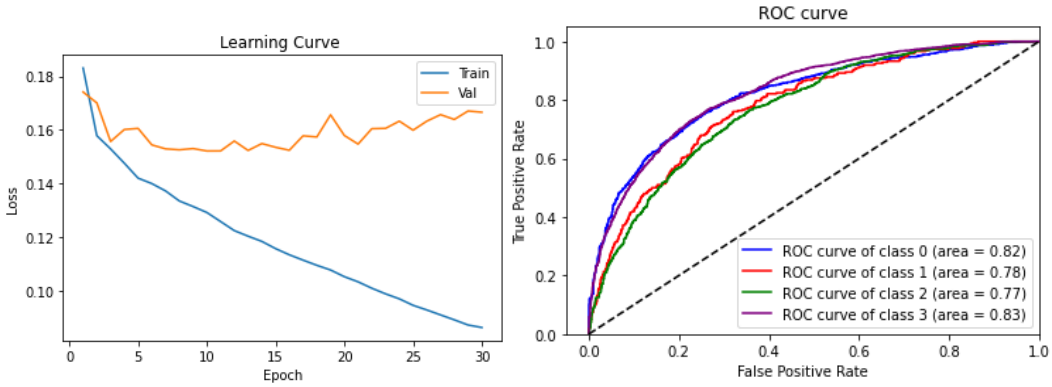


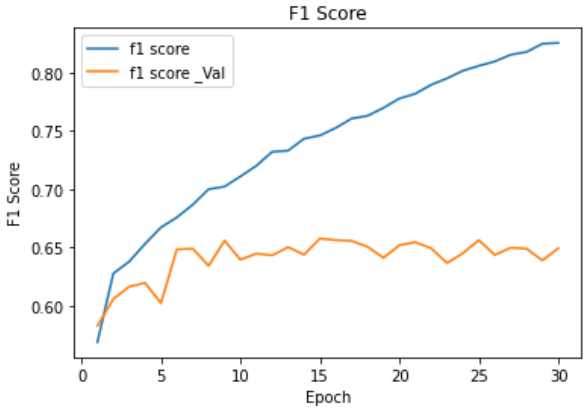
Training loss = 0.35550833320617675 Validation loss = 0.35502094361517167

Training F1_score = 0.2994862806263181 Validation F1_score = 0.3003592857248314

Προφανώς ο Adam optimizer είναι υπερβολικά γρήγορος για να μάθει και χρειάζεται μικρότερο learning rate

Adam με 0.001:



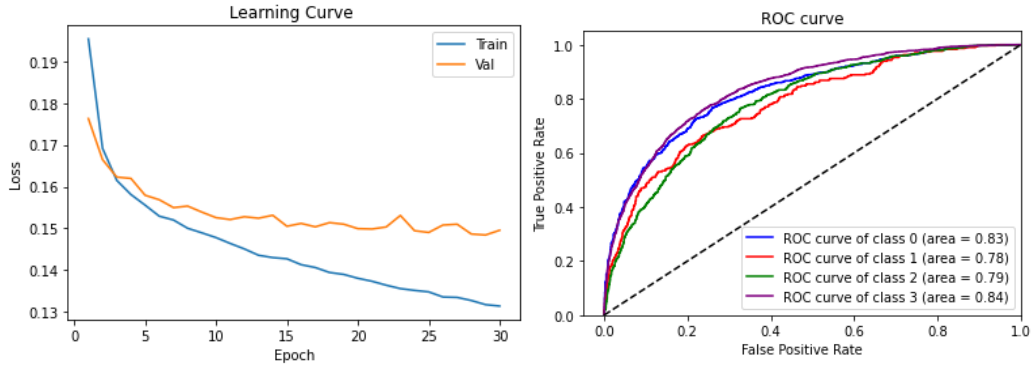


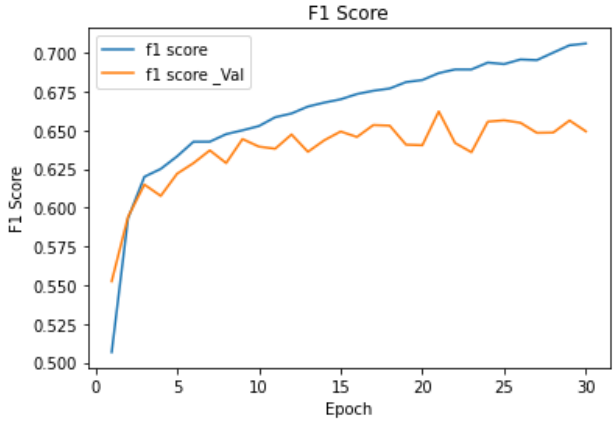
Training loss = 0.08630071644484996 Validation loss = 0.1665958207514551

Training F1_score = 0.8253574950871061 Validation F1_score = 0.6492953563186074

Παρατηρούμε ότι τα νούμερα είναι κάπως καλύτερα από τα προηγούμενα αλλά και πάλι δεν είναι αρκετά καλό και θέλει ακόμα λιγότερο learning rate

Με 0.0001:



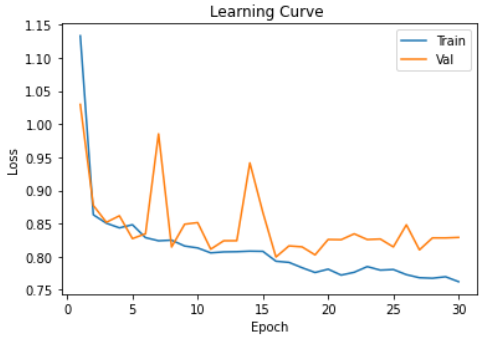


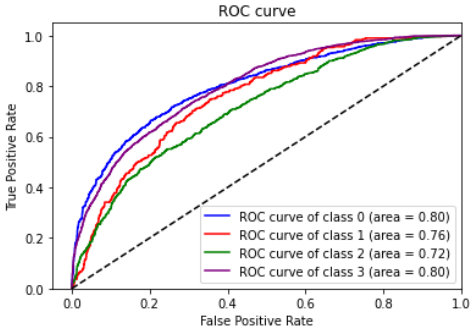
Training loss = 0.13141270160675048 Validation loss = 0.14955587602323955

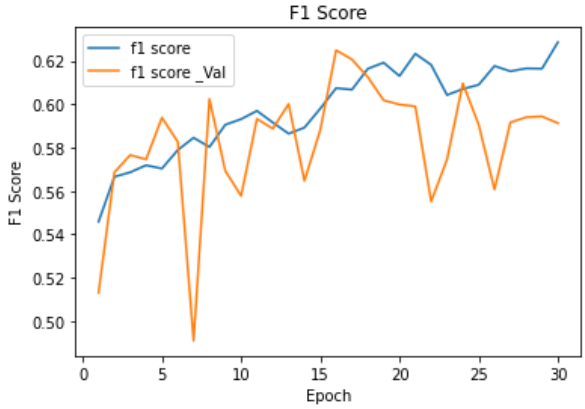
Training F1_score = 0.7060884310716928 Validation F1_score = 0.6493539444200166

Βλέπουμε πολύ πιο λογικά νούμερα από τα προηγούμενα πειράματα αλλά και πάλι δεν είναι στο επίπεδο του MSE οπότε το απορρίπτουμε.

Τέλος θα δοκιμάσουμε με cross entropy loss:







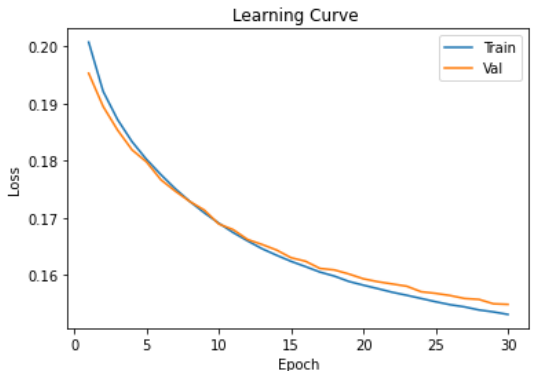
Training loss = 0.7622248024940491 Validation loss = 0.8290840652253892

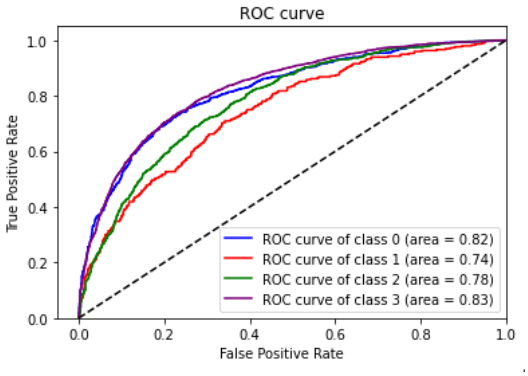
Training F1_score = 0.6286228708696565 Validation F1_score = 0.5912835989329532

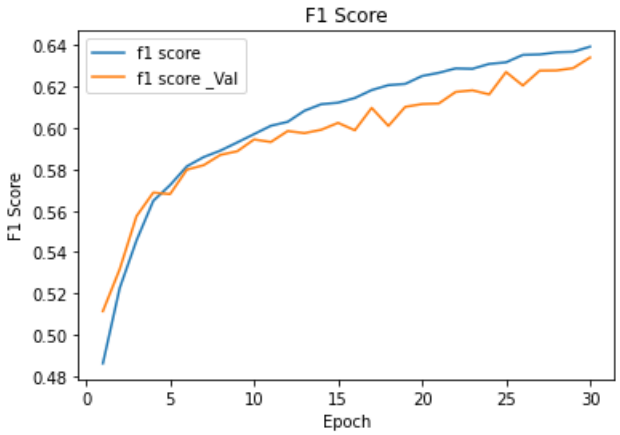
Παρατηρούμε ότι το μοντέλο μας αδυνατεί να μάθει αφού στο γράφημα f1 score δεν είναι σταθερή τιμή. Επίσης παρουσιάζει overfitting με πολύ μεγάλες τιμές loss οπότε δεν θα την χρησιμοποίσουμε.

Τέλος θα δοκιμάσουμε την Glove με 300 διαστάσεις και πολύ χαμηλό learning rate με 1 hidden layer των 250 κόμβων.

Αυτή η επιλογή πρέπει να είναι και η καλύτερη αφού με περισσότερες διαστάσεις, το μοντέλο μαθαίνει και περισσότερα χαρακτηριστικά για τα δεδομένα μας και με χαμηλό learning rate 0.0001 δεν πρόκειται να κάνει τόσο εύκολα overfit. Επίσης το 1 hidden layer των 250 κόμβων δεν κάνει πολύ περίπλοκο το πρόγραμμα και δεν έχει υπερβολικά πολλά βάρη μεταξύ αυτών.







Training loss = 0.1530

Validation loss = 0.1548

Training F1_score = 0.6393

Validation F1_score = 0.6341

Το σύστημα αυτό δεν φαίνεται να κάνει overfitting και τελειώνει σε με καλά f1, precision και recall score με αρκετά χαμηλά loss για το train και validation. Οπότε αυτό το πρόγραμμα είναι και το καλύτερο για την εργασία αυτή. Όμως στο f1 score παρατηρούμε μερικά ανεβοκατεβάσματα το οποίο είναι λογικό αφού έχει πολύ μικρό learning rate.

Πρώτο υποερώτημα

Από το softmax gradient ξέρουμε ότι :

$$y'1 = \frac{e^{x1}}{\sum_k e^{xk}}$$

$$L = -\sum yi \ln (y'1)$$

$$\frac{DL}{dy'i} = -\sum_{i} \frac{yi}{y'i}$$

$$\frac{DL}{dxj} = -\sum_{i!-j} \frac{yi}{y'i}\frac{dy'i}{dxj} - \frac{yj}{y'j}\frac{dyi}{dxj} = = \sum_{i!-j} yi\, y'i + yiy'i - yi = \sum_{i!=j} yi\, y'j - yi = y'i - yi$$