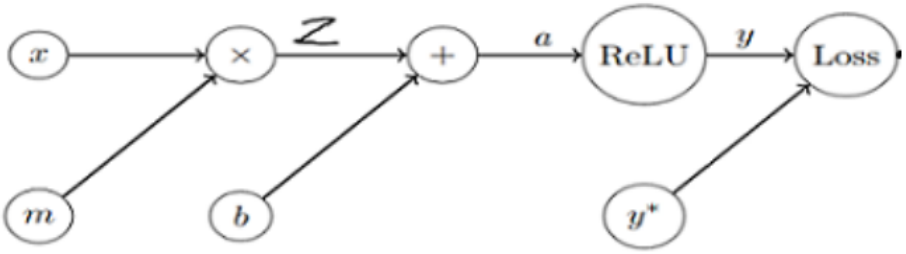
ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ-ΙΙ
ΕΡΓΑΣΙΑ 2
2021-22

ΑΣΚΗΣΗ 2

0.1 Exercise 2

We are given the below graph as input:



Forward prop steps:

- $y = max(0, a)$
- $a = z + b$
- $z = x * m$
- $output = o = (y - y*)^2$

Local Gradients:

- $\frac{dy}{da} = 1(a > 0)$
- $\frac{da}{dz} = b, \frac{da}{db} = z$
- $\frac{dz}{dx} = m, \frac{dz}{dm} = x$
- $\frac{do}{dy} = 2(y - y*), \frac{do}{dy*} = -2(y - y*)$

Using the $[downstream = upstream * localgradient]$ equation we can start the backpropagation from the output so the gradients of each node will be:

- $\frac{do}{da} = \frac{do}{dy}\frac{dy}{da} = 1 * 2(y - y*) = 2(max(0, x * m + b) - y*)$
- $\frac{do}{dz} = \frac{do}{da}\frac{da}{dz} = 2(max(0, x * m + b) - y*) * b$
- $\frac{do}{db} = \frac{do}{da}\frac{da}{db} = 2(max(0, x * m + b) - y*) * z$
- $\frac{do}{dx} = \frac{do}{dz}\frac{dz}{dx} = 2(max(0, x * m + b) - y*) * b * m$
- $\frac{do}{dm} = \frac{do}{dz}\frac{dz}{dm} = 2(max(0, x * m + b) - y*) * b * x$

The last 2 gradients (y,y*) are the same as their local gradients calculated in the start.

ΑΣΚΗΣΗ 3 (PYTORCH)

ΔΟΜΗ ΤΟΥ NOTEBOOK

Σε αυτή την εργασία δεν έβαλα ένα τεράστιο code block που τρέχει όλη την εργασία για δύο λόγους.

1) Ο κώδικάς μου είναι οργανωμένος σε συναρτήσεις και οι ειδικά οι συναρτήσεις plotting θα πρέπει να καλεστούν σε διαφορετικό κελί κώδικα για κάθε μοντέλο.
2) Σε αυτή την εργασία υπάρχουν πάρα πολλοί υπερπαράμετροι και μπορείτε αν θέλετε έτσι πολύ εύκολα να κάνετε κάποιους από τους πειραματισμούς που έκανα εγώ με τα μοντέλα απλά αλλάζοντας τα νούμερα στο αντίστοιχο block.

Οπότε θα πρέπει να τρέξετε τα code blocks ένα-ένα.

Για τα datasets εγώ συνήθως έφτιαχνα έναν φάκελο data στο collab και τα ανέβαζα εκεί μπορείτε να κάνετε το ίδιο αλλάζοντας μόνο τα ονόματα των αρχείων για να βάλετε τα δικά σας ή αλλάζοντας και το path αν θέλετε να χρησιμοποιήσετε κάποιο drive. Τα txt αρχεία του glove θα κατέβουν αυτόματα στο πρώτο code block με wget και unzip.

Πειραματισμός και μελέτη των νευρωνικών

Ξεκινώντας έτρεξα τον κώδικα που είδαμε στο φροντιστήριο με τα δικά μας dataset και τα word emmbedings και έπιασε 56.2% στο validation set μετά το training και με τεράστιο overfit παράλληλα. Από εκεί το πρώτο πράγμα που αλλάχτηκε ήταν το loss function που από SGD έγινε Cross Entropy Loss και δεν σκόπευα να δοκιμάσω κάποια διαφορετική συνάρτηση μέχρι να βρω ένα πολύ καλό μοντέλο καθώς είχαμε δει στο μάθημα ότι είναι από τις καλύτερες για classification. Επίσης κάτι το οποίο διάβασα για το pyTorch είναι ότι η nn.CrossEntropyLoss εφαρμόζει από μόνη της μία softmax οπότε δεν χρειάζται να βάλουμε έξτρα softmax στο output layer. Από εκεί η πρώτη βελτίωση ήρθε λίγο μετά όταν έβαλα την sigmoid για activation function και τον Adam optimizer η οποία ανέβασε το f1 score στο 63% και έριξε κατά 0.15 το τελικό loss. Επίσης σε αυτό το σημείο είχα μειώσει τον αριθμό των εποχών στο 50 γιατί από εκεί και μετά όσα μοντέλα είχα δοκιμάσει ξεκινούσαν πάρα πολύ μεγάλο overfit (εδώ να σημειωθεί ότι το τελικό μοντέλο εκπαιδεύεται πάλι για 50 εποχές γιατί λίγο μετά ξεκινάει το overfitting). Για την εφαρμογή του Adam καταστάλαξα μετά από λίγη έρευνα και το paper

(https://openreview.net/pdf?id=HygrAR4tPS) που εξηγεί για διάφορους optimizers είδα ότι από τους καλύτερους είναι ο Adam και ο RMSprop αλλά προς το παρόν εφάρμοσα Adam.

Κάνοντας tuning στο learning rate ή στα hidden layers και τα nodes τους δεν βελτίωσε κάτι οπότε συνέχισε την έρευνα και διάβαζα σε πολλά paper και ιστοσελίδες ότι γενικά η ReLU σαν activation function θα κάνει την δουλειά της αρκετά καλά αλλά είναι απλά overall καλή, λογικά δεν θα είναι η βέλτιστη για το κάθε πρόβλημα. Οπότε εφάρμοσα ReLU και κράτησα την sigmoid μόνο στο τελευταίο hidden layer γιατί επίσης κάπου είχα διαβάσει ότι είναι καλή πρακτική αυτό. Πρακτικά καμία απολύτως βελτίωση δοκίμασα και να ρίξω τον αριθμό των hidden layers ή να ανεβάσω τα nodes αλλά τίποτα. Μετά δοκίμασα LogSigmoid, Tanh δεν είδα καμία απολύτως βελτίωση.

Εκεί ξεκίνησα να ψάχνομαι περισσότερο και κοιτούσα στο site που εξηγεί τις συναρτήσεις των νευρωνικών (https://pytorch.org/docs/stable/nn.html) και ανακάλυψα για το batch normalization και το dropout για τα οποία διάβασα από εδώ (https://arxiv.org/abs/1502.03167) , (https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/). Το dropout πρακτικά πετάει ένα ποσοστό των αποτελεσμάτων για να γλυτώσει το overfit το μοντέλο αλλά εγώ ακόμα και με όλα αυτά δεν έβλεπα βελτίωση γιατί το πρόβλημα ήταν αλλού και δεν το έβλεπα μέχρι στιγμής. Συνέχιζα να δοκιμάζω πολλά πράγματα μερικά από αυτά δεν έκαναν καν train το μοντέλο ώσπου ανακαλύπτω ότι το batch size δεν το έχω αλλάξει και είναι κολλημένο στο 64. Με το ανέβασμα στο 100 και την προσθήκη dropout μετά από κάθε hidden layer δημιουργήθηκε το πρώτο καλό μοντέλο με πολύ μικρό overfit και 66% accuracy.

Μετά από εδώ πρόσθεσα τις εντολές model.eval() και model.train() στα κατάλληλα σημεία για να γνωρίζει το μοντέλο αν βρίσκεται σε κατάσταση εκπαίδευσης ή όχι. Επίσης πρόσθεσα και την εντολή torch.no_grad() κατά το σημείο του testing στο validation set μετά από κάθε εποχή (και στο τελικό) η οποία λέει στο μοντέλο να μην κάνει update τα gradients εκείνη τη στιγμή (επειδή τεστάρεται δεν εκπαιδεύεται).

Το επόμενο βήμα έγινε όταν έβαλα learning rate 5e-4, batch size 200 και για layers είχα 5 τα οποία τα είχα αλλάξει σε κάποια προηγούμενα test με αριθμό nodes = 100 100 50 25 3 αντίστοιχα. Αυτό ήταν το πρώτο μοντέλο μου που δεν είχε overfit και είχε 67% accuracy score. Εδώ να αναφέρω ότι το overfit το καταλάβαινα από το training loss σε κάθε εποχή και επειδή μετά από κάθε εποχή το test loss ανέβαινε αντί να πέφτει.

Εκεί πρόσθεσα στον διαχωρισμό του training set σε batches την παράμετρο shuffle=true για να ανακατεύει τα δεδομένα και παίζοντας με το dropout είχα την ιδέα να βάλω διαφορετικά dropout functions όπου σε κάθε layer θα πετιούνται όλο και λιγότερα

δεδομένα το οποίο πράγματι σε συνδυασμό με καινούριο αριθμό layers (256 128 64 32 3) είχε 66% f1 και 65% accuracy και καθόλου overfit.

Αυτό ήταν ένα καλούτσικο μοντέλο οπότε τώρα ξεκίνησα να δοκιμάζω άλλους optimizers κρατώντας τα layers σταθερά και αλλάζοντας μόνο το learning rate, batch size. Ο SGD έριξε το f1 στο 29% και το accuracy στο 46% και ότι και να έκανα δεν ανέβαιναν και πολύ. Επίσης έκανε πολύ μεγάλο overfit το μοντέλο ξανά.

Ο adamax ήταν καλούτσικος με f1 και accuracy 65% σχεδόν ίδιος με το μοντέλο που είχα πριν με σκέτο Adam.

Εδώ που είχα καλές βλέψεις ήταν ο RMSprop αλλά με απογοήτευσε κάνοντας σχεδόν τα ίδια score με ένα μικρό overfit πάντα το οποίο δεν μπόρεσα να λύσω.

Από εκεί και μετά αφού ο Adam ήταν ο καλύτερος optimizer που είχα δοκιμάσει αποφάσισα να πειραματιστώ με τις παραλλαγές του. Ξεκίνησα με τον RAdam ο οποίος είναι και αυτός που χρησιμοποίησα στο τελικό μου μοντέλο. Ο RAdam μου έδωσε τα ίδια score με τον κανονικό Adam αλλά οι καμπύλες του μοντέλου μου έμοιαζαν περισσότερο με αυτές ενός μοντέλου προς παράδειγμα όπως είχαμε δει στο μάθημα (βλ. διάγραμμα σελίδα 8 το PReLU Model Losses Per Epoch).

Μετά είπα να δοκιμάσω και τους υπόλοιπους όπως τον NAdam ο οποίος είχε τα ίδια score αλλά έκανε πολύ εύκολα overfit και τέλος τον Adagrad ο οποίος είναι σχετικά καλή παραλλαγή του Adam και πράγματι δεν έκανε overfit αλλά δεν μπορούσα να ανεβάσω το score του πάνω από 60%.

Εδώ τελειώνει ο πειραματισμός με τους optimizer και ξεκινάει ο πειραματισμός με activation functions(Μέχρι τώρα είχα ReLU). Η Tanh ήταν και αυτή μία καλή activation αλλά είχε 60% F1 score και σε αυτό το σημείο επέλεξα να αλλάξω μόνο την activation function και να δοκιμάσω 4-5 χωρίς να αλλάξω τίποτα άλλο πέρα από το batch size να δω στο περίπου ποια τα πάει καλύτερα.

Η επόμενη στην λίστα ήταν η PReLU που είναι και αυτή του τελικού μοντέλου μου που με την πρώτη ματιά δεν φάνηκε τόσο καλή γιατί έριξε τα score αλλά έριξε παραπάνω το loss και μου άρεσε πολύ ο τρόπος με τον οποίο εκπαιδεύει το μοντέλο και βεβαίως οι γραφικές παραστάσεις του μοντέλου χρησιμοποιώντας PReLU.

Μετά δοκίμασα και τις CeLU, SeLU που η πρώτη προκαλούσε overfitting και η δεύτερη είχε χαμηλά score. Κάπου εδώ κατάλαβα ότι πρέπει να αναπτύξω και άλλο την ιδέα της PReLU. Εδώ να πω ότι επειδή παράλληλα ανεβοκατέβαζα το batch size αντιλήφθηκα ότι για το δικό μας dataset είναι καλό από 300 και πάνω το batch size.

Πειράζοντας πολλά πράγματα πλέον όπως το dropout percentage, layers, learning rate και batch size το επόμενο πολύ καλό μοντέλο και είναι σχεδόν ίδιο με το τελικό ήταν μία δοκιμή που έκανα με τα παρακάτω χαρακτηριστικά: learning rate = 2e-4, hidden layers = (125 125 75 30 3), batch size = 450, dropout1 = 30% (from 50%). Αυτό το μοντέλο είχε 0.76 loss και 64% f1 και accuracy scores.

Κάπου εδώ ξαναέδωσα μία ευκαιρία στον RMSprop αλλά πάλι δεν τα πήγε καλά και επίσης δοκίμασα τον AdamW optimizer ο οποίος προκαλούσε overfitting. Ύστερα δοκίμασα και τον AdaDelta με τον οποίο δεν εκπαιδευόταν καν το μοντέλο.

Η προτελευταία ομάδα δοκιμών μου ήταν διάφορες αλλαγές στο παραπάνω μοντέλο χωρίς να δω κάποια βελτίωση και μετά αντικατέστησα την PReLU με την tanh η οποία με λίγο tuning στις υπόλοιπες hyperparameters μου έδωσε το δεύτερο καλύτερο μοντέλο και αυτό το οποίο χρησιμοποίησα παρακάτω για να κάνω τις συγκρίσεις. Αυτές ήταν: learning rate=9e-4, batchsize=400, hidden layers = 256 128 64 32 3. Το αποτέλεσμα ήταν loss < 0.7, 66% f1 score, 67% accuracy score.

Τέλος, ήρθε η ώρα να δοκιμάσω και άλλες loss functions στα 2 καλύτερα μοντέλα μου προσθέτοντας και sfotmax στο output layer παρόλο που δεν πίστευα ότι κάποια από αυτές που έχει το nn είναι καλύτερη για classification. Η SGD που δοκιμάστηκε στο καλύτερο μοντέλο μου έριξε τα score στο μισό παρά όποιες αλλαγές σε συγκεκριμένες παραμέτρους έκανα δεν φαινόταν και τόσο καλή. Μετά δοκίμασα τις αρνητικές όπως nn.GaussianNLLLoss, NLLLoss, PoissonNLLLoss αλλά δεν έκαναν καν τη δουλειά τους και το loss του μοντέλου έμενε σταθερό σε κάθε εποχή. Δοκίμασα και κάποιες άλλες που χρησιμοποιούνται κυρίως για regression και το έγραφε ξεκάθαρα στην περιγραφή όπως KLDivLoss, L1 αλλά επίσης δεν έκαναν καν το μοντέλο να δουλέψει. Τελευταίες δοκίμασα την MultiMarginLoss που εφαρμόζεται σίγουρα σε classification προβλήματα αλλά παρόλο που το μοντέλο στην τελευταία εποχή κατάφερε να έχει μόνο 0.25 όλα τα υπόλοιπα ήταν τα ίδια με το καλύτερο μοντέλο μου εκτός από το f1 score που έπεσε κατά 1%. Μπορούμε α πούμε ότι εν τέλει και αυτή είναι μία πολύ καλή loss function.

Μια τελευταία βελτίωση που σκέφτηκα να δοκιμάσω αλλά περίμενα να μην δουλέψει ήταν να χρησιμοποιήσω το glove.6B.300d.txt αντί για το glove.6B.50d.txt που χρησιμοποιούσα μέχρι αυτή τη στιγμή. Ο λόγος που δεν περίμενα να δουλέψει ήταν το averaging που κάνουμε στα διανύσματα των λέξεων έτσι όλες οι προτάσεις να έχουν τις ίδιες διαστάσεις και θεωρώ ότι χάνεται πάρα πολύ πληροφορία εκεί. Οπότε με το 300d είναι ακόμα περισσότερες οι διαστάσεις θα γίνονται σε περισσότερα διανύσματα averaging άρα ακόμα περισσότερη έλλειψη πληροφορίας. Όμως προς έκπληξή μου με το

300d το f1 score ανέβηκε από 66% σε 71,1% και το accuracy από 66% σε 71.4%. Αυτό ήταν και το τελικό καλύτερο μοντέλο μου για αυτό το project (Αναλύεται περαιτέρω παρακάτω στις συγκρίσεις).

Όλες αυτές οι μετρήσεις και ακόμα περισσότερες είναι γραμμένες στο excel αρχείο notes που υπάρχει μέσα στο zip το οποίο είναι αυτό μέσα στο οποίο έγραφα την ώρα που πειραματιζόμουν με τα μοντέλα και αν θελήσετε μπορείτε να το κοιτάξετε. Είναι εκεί σημειωμένες οι υπερπαράμετροι για κάθε test, τα score που πέτυχε, το loss και το overfit. Με πράσινο είναι χρωματισμένα κάποια αρκετά καλά μοντέλα που ήθελα να ξανακοιτάξω αργότερα γιατί ήτα αξιόλογα. Επίσης η τελεία '.' σημαίνει ότι η συγκεκριμένη παράμετρος δεν άλλαξε τιμή από την προηγούμενη φορά. Όταν κάτι αλλάζει σημειώνεται ξεκάθαρα στην αντίστοιχη στήλη. Επίσης η πρώτη στήλη είναι οι αλλαγές οι οποίες έχουν να κάνουν με την κλάση του μοντέλου (στο forward του).

Σύγκριση του Tanh με το PReLU μοντέλο

Το μοντέλο που χρησιμοποιεί PReLU και έχει hidden layers με μικρότερο αριθμό κόμβων και μικρότερο learning rate είναι το καλύτερο μοντέλο της μελέτης που έκανα για αυτή την εργασία και προφανώς καλύτερο από το Tanh μοντέλο. Αρχικά ας κάνουμε έναν πίνακα με τα διάφορες score των δύο μοντέλων στο testset.

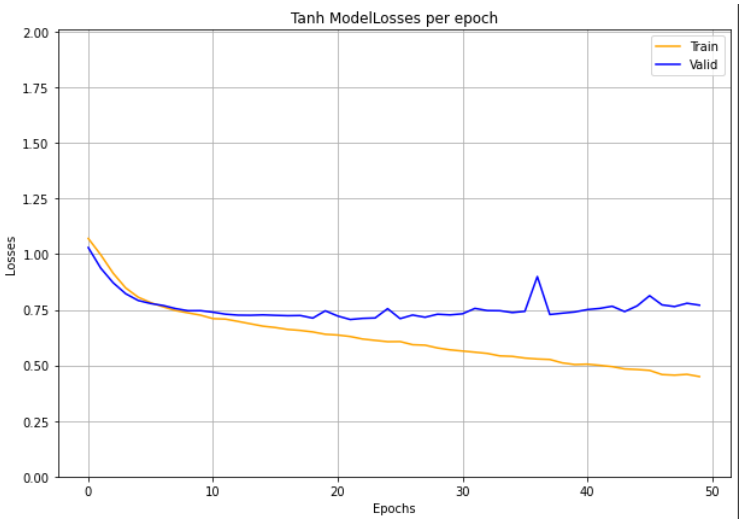
Scores	Tanh	PReLU
Accuracy	70.5%	71.4%
F1	70.6%	71.1%
Recall	70.5%	71.4%
Precision	71.2%	71.5%
CrossEntropyLoss	0.77	0.67

Πολύ εύκολα παρατηρούμε ότι σε καμία σειρά το Tanh μοντέλο δεν έχει καλύτερο score από το PReLU. Αλλά η αλήθεια είναι ότι οι αριθμοί δεν έχουν μεγάλη απόκλιση οπότε πρέπει να παρατηρήσουμε και άλλα δεδομένα για τα 2 μοντέλα. Το Tanh μοντέλο αν παρατηρήσουμε προσεκτικά το loss που έχει σε κάθε εποχή υπάρχουν μερικά σημεία που το loss ανεβαίνει λίγο από την προηγούμενη εποχή και μετά πέφτει το οποίο είναι ένα σημάδι μικρού overfitting:

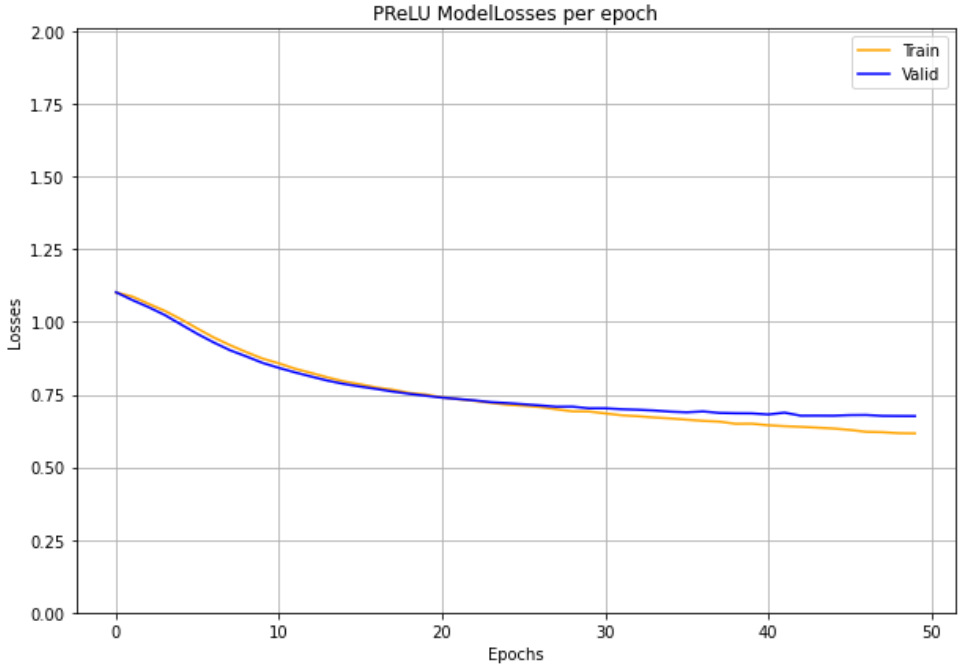
```
Epoch  24: Loss = 0.60636
Epoch  25: Loss = 0.60688
```

```
Epoch  46: Loss = 0.45865
Epoch  47: Loss = 0.45595
Epoch  48: Loss = 0.45954
```

Το οποίο μετά επιβεβαιώνεται στην καμπύλη με τα δεδομένα από το testset:

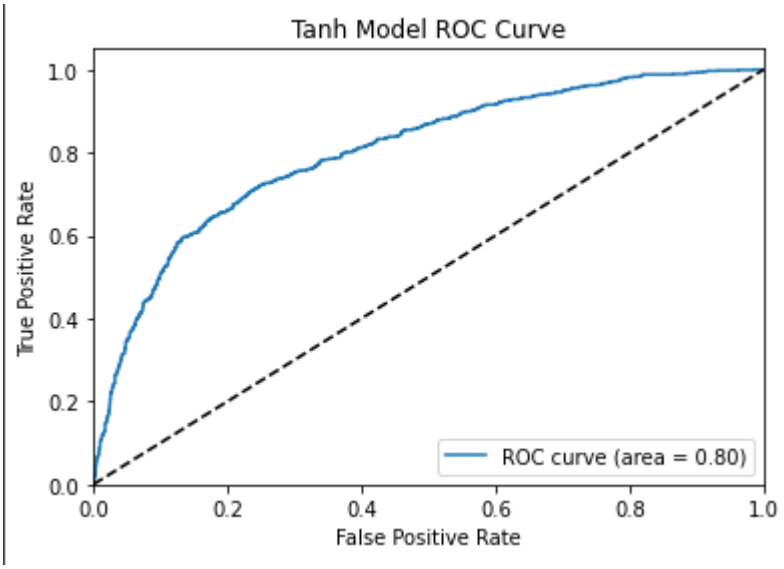


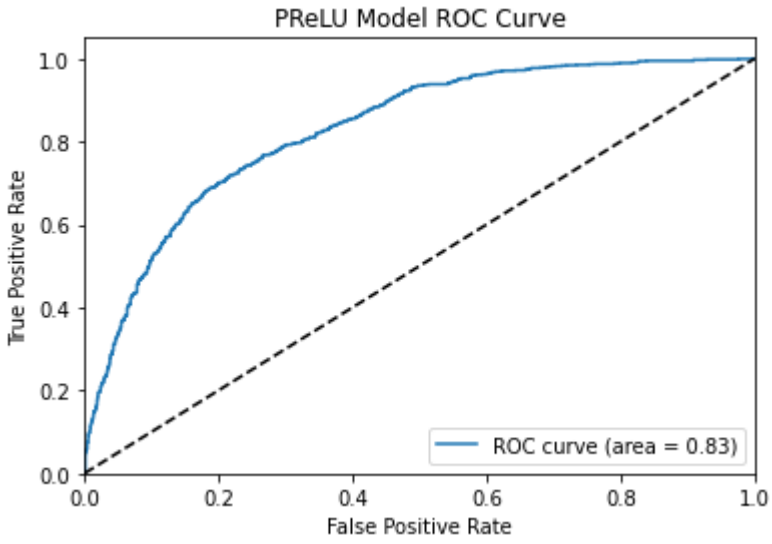
Εδώ βλέπουμε το overfitting στην καμπύλη του validation set που από ένα σημείο και μετά ενώ το μοντέλο εκπαιδεύεται το loss στο test set ανεβαίνει. Ενώ στο PReLU μοντέλο τα πράγματα είναι πολύ πιο καλά:



Η αλήθεια είναι ότι θα μπορούσα να επιλέξω ένα από τα άλλα μοντέλα μου σαν δεύτερο που να μην κάνει overfit αλλά επέλεξα αυτό γιατί είχε το δεύτερο καλύτερο score και επίσης ήθελα να δείξω αυτό το φαινόμενο του overfit το οποίο το είχαν το 75% των μοντέλων μου και λύθηκε χρησιμοποιώντας κατά βάση dropout και tuning στο batch size.

Τέλος, συγκρίνοντας και τα ROC curve των δύο μοντέλων παρατηρούμε επίσης πόσο καλύτερο είναι το PReLU μοντέλο καθώς η καμπύλη του φτάνει πιο κοντά στην πάνω αριστερή γωνία η οποία συμβολίζει το τέλειο μοντέλο.



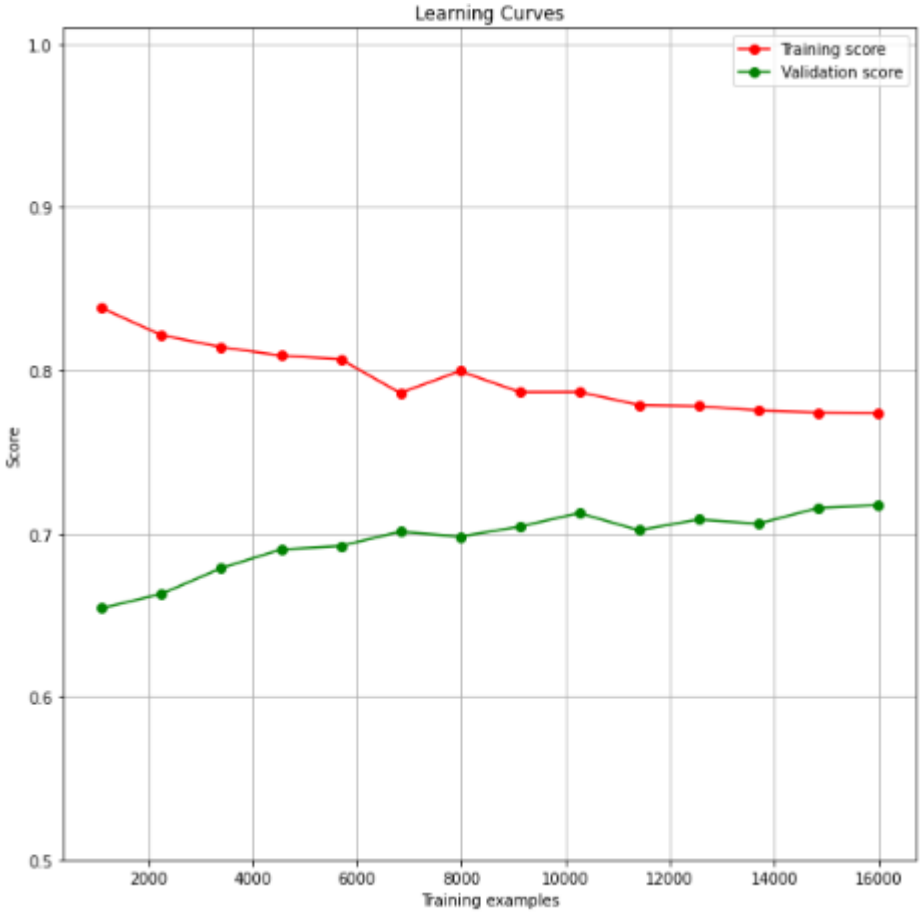


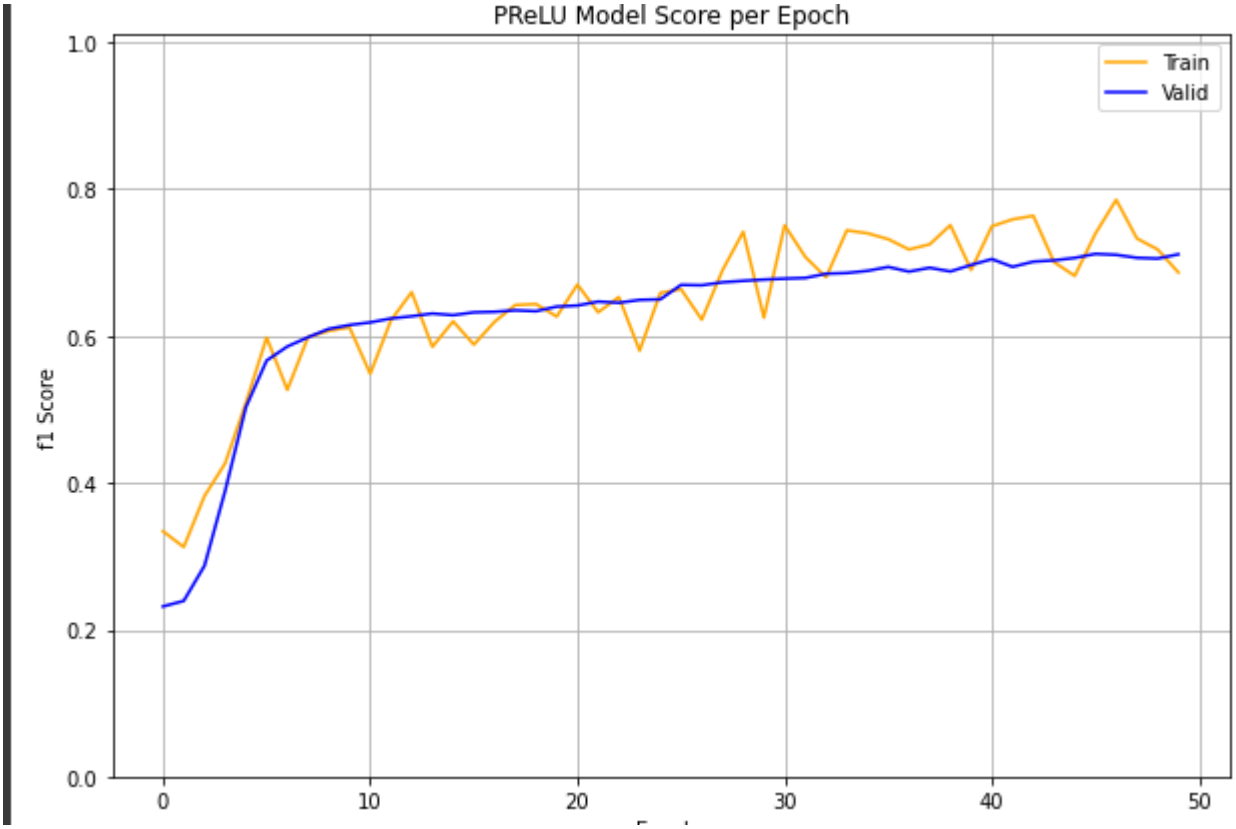
Σύγκριση του PReLU μοντέλου με το Softmax Regression using HashingVectorizer της εργασίας 1

Θα ξεκινήσουμε συγκρίνοντας τα score των δύο μοντέλων βάση του accuracy και του f1 (το recall και precision ήταν το ίδιο με το f1 στην εργασία 1 λόγω micro averaging).

Scores	Softmax Regression	PReLU NN
Accuracy	71.3%	71.4%
F1	71.3%	71.1%

Τα δύο μοντέλα έχουν σχεδόν την ίδια αποδοτικότητα στο test set με το softmax regression να είναι λίγο καλύτερο και ο λόγος πάλι πιστεύω όπως ανέφερα και πριν πως είναι το averaging που κάνουμε στα διανύσματα των word emmbedings και χάνεται πολύ πληροφορία. Υπό κανονικές συνθήκες προφανώς να έπρεπε ένα nn να είναι αρκετά καλύτερο αλλά δεν κατάφερα να το κάνω. Ας δούμε τώρα και τις καμπύλες των score βέβαια έχουν διαφορετικό άξονα x αλλά δεν μας πειράζει.





Αυτό που μπορούμε με σιγουριά να συμπεράνουμε είναι ότι τα nn εκπαιδεύονται πολύ πιο γρήγορα από τα regression μοντέλα και εκτός αυτού στην αρχή ξεκινάνε με ποσοστό που είναι σαν το μοντέλο να απαντάει στην τύχη ενώ στο softmax regression δεν συμβαίνει αυτό.