ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΙΙ

ΕΡΓΑΣΙΑ 3

Πρόλογος

Σκοπός της εργασίας αυτής είναι να προπονηθεί ένα νευρωνικό δίκτυο και συγκεκριμένα ένα αναδρομικό **(RNN)** για ένα πρόβλημα κατηγοριοποίησης πολλαπλών κλάσεων με σκοπό την περαιτέρω ανάλυση του χαρακτήρα (**sentiment analysis**) κάποιων αναρτήσεων του twitter. Στην συνέχεια, θα αναλυθούν λεπτομερώς τόσο τα βήματα που ακολουθήθηκαν όσο και κάποιες παρατηρήσεις αλλά και συγκρίσεις μεταξύ διαφορετικών πρακτικών.

Γενική περιγραφή

Μερικά από τα κύρια βήματα που ακολουθήθηκαν είναι αρκετά παρόμοια με αυτά της προηγούμενης εργασίας. Πιο συγκεκριμένα, αφαιρέθηκαν αρκετά σημεία τα οποία είχαν σχέση με ζητούμενα της προηγούμενης άσκησης και προστέθηκαν ή τροποποιήθηκαν αρκετά άλλα που έχουν σχέση αποκλειστικά με τα RNNs. Δηλαδή η **φόρτωση** των training και validation **datasets**, η **προεπεξεργασία** τους, και ο τρόπος απεικόνισης των **αποτελεσμάτων** (roc curves, confusion matrices…) είναι τα ίδια. Η εξαγωγή **στατιστικών στοιχείων** είναι και αυτή η ίδια απλώς τώρα προστέθηκε και η εύρεση του μέσου μήκους των tweets το οποίο είναι αρκετά χρήσιμο για την επιλογή ενός ικανοποιητικού μήκους εισόδου στα RNNs. Επίσης, προστέθηκε και η δυνατότητα για εκτέλεση στη **GPU** μιας και υπήρχε τεράστια διαφορά στους χρόνους εκτέλεσης από τους χρόνους που έκανε η **CPU**. Το τελευταίο είναι αρκετά λογικό μιας και τα RNNs χρειάζεται να κάνουν πολλούς παράλληλους **υπολογισμούς** και η GPU τους κάνει πολύ πιο αποτελεσματικά.

Ουσιαστικά οι αλλαγές αρχίζουν από το σημείο που δηλώνονται κάποια βοηθητικά εργαλεία που θα χρησιμοποιηθούν από τη κλάση του μοντέλου.

Το πρώτο εργαλείο αναφέρεται στο δεύτερο σκέλος της άσκησης δηλαδή στη δημιουργία του **attention layer**. Ο μηχανισμός του attention είναι αρκετά απλός και το μόνο που χρειάζεται είναι ένα batch από **queries** και ένα batch από τα αντίστοιχα **contexts** έτσι ώστε το κάθε **query** να κάνει **attend** στα **keys** του αντίστοιχου context. Στο σε τι θα αντιστοιχεί το κάθε query και το αντίστοιχο context του θα περιγραφεί παρακάτω. Όσον αφορά την υλοποίηση αυτή έγινε με συνδυασμό ιδεών από τις διαφάνειες του μαθήματος, από ένα doc της βιβλιοθήκης pytorch nlp που παρέχει μια λίγο διαφορετική υλοποίηση και από ένα σχετικό άρθρο. Σε πρώτο στάδιο

λοιπόν, υπολογίζονται τα διαφορετικά **dot products** του query με το κάθε ένα key του context. Αυτό βολεύει αρκετά διότι το context είναι ένας πίνακας (**S x D**) και το query ένα διάνυσμα (**1 x D**). Δηλαδή το context είναι ένας πίνακας που περιέχει S διανύσματα/keys ίδιων διαστάσεων με το query τα οποία είναι και τα στοιχεία που το query θα πρέπει να κάνει attend. Οπότε παίρνοντας τον **ανάστροφο** πίνακα του context και κάνοντας τον πολλαπλασιασμό με το query γίνεται αρκετά εύκολα και μαζικά το dot product με όλα τα keys όπως αναφέρθηκε παραπάνω και έτσι υπολογίζονται κάποια **scores** τα οποία θα περαστούν στην συνέχεια από ένα **softmax layer** για προκύψουν τα βάρη του κάθε key. Πριν όμως εφαρμοστεί το softmax layer αν κάποια keys του context αποτελούν **θόρυβο** (π.χ padding) τότε δεν θα πρέπε κανονικά να υπολογιστεί score γι αυτά οπότε εφαρμόζεται κάποιο **masking** των scores τους με την τιμή **-∞** έτσι ώστε το softmax layer να δώσει **βάρος ίσο με 0** σε αυτά. Στο τέλος γίνεται ένα **weighted average/selective summary** με βάση τα βάρη που υπολογίστηκαν ως προς τα **values** (τα οποία είναι τα ίδια τα keys δηλαδή το ίδιο το context) και έτσι παράγεται το attention. Το attention αυτό **συνενώνεται** (γίνεται concatenation) με το αρχικό query (ή μπορεί και με κάποια άλλη αναπαράσταση/διάνυσμα αν έχει δοθεί) και το αποτέλεσμα διέρχεται από κάποιο **projection layer** ώστε να το επαναφέρει στις αρχικές διαστάσεις του αρχικού query. Επίσης, εφαρμόζεται και μια **tanh** activation function για να περιοριστεί το εύρος των τελικών τιμών (από -1 έως 1). Άρα, αυτό που περιγράφηκε αφορά το απλό attention με dot product όπου οι **διαστάσεις** του query και των keys θα πρέπει πάντα να είναι οι **ίδιες** πράγμα το οποίο δεν θα αποτελέσει πρόβλημα στη συνέχεια. Παρόλα αυτά υλοποιήθηκε και άλλος ένας τύπος attention ο οποίος μοιάζει αρκετά με τον προηγούμενο απλά εισάγει κάποιο layer χωρίς bias δηλαδή κάποια βάρη (**weights**) μεταξύ του query και του context. Αυτός ο τύπος συναντάται με το όνομα **mutliplicative** ή **general attention** ο οποίος έχει βάρη που τα μαθαίνει δυναμικά και δεν επηρεάζεται από το πρόβλημα των διαφορετικών διαστάσεων.

Τα άλλα δύο εργαλεία αφορούν τον τρόπο κατασκευής των **batches** από τον DataLoader. Πιο αναλυτικά, ορίζεται αρχικά μια dataset κλάση (TweetDataset) η οποία θα αποθηκεύει όλα τα tweets με τα labels τους. Η ανάγκη για την δημιουργία του dataset αυτού έναντι του κλασικού TensorDataset που χρησιμοποιήθηκε στην προηγούμενη άσκηση προέκυψε από την ίδια την αναπαράσταση των tweets τα οποία τώρα έχουν **διαφορετικά μεγέθη** όπως θα φανεί και παρακάτω οπότε δεν υπήρχε κάποιος τρόπος να δημιουργηθεί μια ενιαία αναπαράσταση/tensor ώστε να δοθεί στο TensorDataset . Το άλλο εργαλείο αφορά τον τρόπο με τον οποίο θα γίνεται το **padding** αυτών των διαφορετικών μεγεθών/μηκών των ακολουθιών (**sequences**) μέσα σε ένα batch. Πιο συγκεκριμένα, αν έχει οριστεί κάποιο **φράγμα** για το μήκος των ακολουθιών γίνεται **αποκοπή** σε αυτό το μήκος αν το ξεπερνάει κάποια ακολουθία στο batch. Έτσι, στο τέλος πραγματοποιείται το padding μέχρι το μήκος της μεγαλύτερης ακολουθίας στο batch και επιπλέον υπολογίζονται και τα

διαφορετικά μήκη των ακολουθιών πριν το padding το οποίο θα φανεί χρήσιμο στη συνέχεια. Η συνάρτηση αυτή πρόκειται να συνδυαστεί με τον DataLoader ώστε να προσθέτει το padding αυτόματα κατά την δημιουργία των batches (αυτή η πρακτική συστήνεται και στα docs του pytorch).

Έπειτα ακολουθεί η κλάση του μοντέλου που θα δοκιμαστεί. Η κλάση αυτή διαφέρει κυρίως στα εξής σημεία από την κλάση του μοντέλου της προηγούμενης εργασίας:

- **Αρχικοποίηση**(__init__): Το βήμα αυτό διαφέρει τελείως. Γενικότερα, ορίζονται κάποιες παράμετροι του μοντέλου. Οι παράμετροι αυτές είναι ο τύπος του RNN με βάση τα cells (Vanilla, LSTM, GRU), ο αριθμός των layers, ο αριθμός των νευρώνων (**units**) του κάθε hidden state (**hidden dim/size**), ο βαθμός του dropout , ο αριθμός των κλάσεων, οι κατευθύνσεις, ένας αριθμός που δηλώνει κάθε πόσα layers θα γίνονται οι συνδέσεις για τα skip connections, ο τύπος του attention και ο τρόπος με τον οποίο θα γίνεται η τελική κωδικοποίηση (**encoding**) από τα hidden states στο τελευταίο layer του RNN.
  Παράλληλα, δημιουργούνται και κάποια δομικά στοιχεία. Αρχικά, δίνονται τα **pretrained embeddings** και δημιουργείται ένα embedding layer από αυτά. Τα **βάρη** του δεν θα μπορούν να αλλάζουν μιας και κάτι τέτοιο θα οδηγούσε στην μη αποδοτική και σωστή αξιοποίηση των embeddings. Στην συνέχεια, δημιουργούνται τα layers του RNN **ένα-ένα** όπου το κάθε layer είναι και ένα ξεχωριστό RNN με **ένα μόνο layer**. Ο λόγος για τον οποίο γίνεται αυτό είναι πως υπάρχει ανάγκη για την πρόσβαση όλων των outputs/hidden states του εκάστοτε layer μιας και ζητείται η προσθήκη των **skip connections**. Ένα άλλο στοιχείο που ορίζεται και αφορά τα skip connections είναι ένα επιπλέον layer και πιο συγκεκριμένα ένα σετ βαρών χωρίς bias το οποίο θα κάνει **projection** του αρχικού **input** που δίνεται **στο πρώτο layer** στις διαστάσεις του κάθε output από κάθε layer (**hidden size**) ώστε να μπορεί **να γίνει η ένωση** του input με οποιοδήποτε από αυτά. Αυτό αναφέρεται και στην βιβλιογραφία ως shortcut. Επίσης, ορίζεται και ένα άλλο layer το οποίο θα κάνει το **τελικό projection** του output του RNN στον **αριθμό των κλάσεων** ώστε να περαστεί από την **softmax** για να παραχθούν οι προβλέψεις. Τέλος, γίνεται και η δημιουργία του **attention layer** στην περίπτωση που ζητηθεί.
- **Forward pass** (forward): Το βήμα αυτό διαφέρει τελείως. Αρχικά, δίνεται ένα batch στο οποίο έχει ήδη εφαρμοστεί το padding σε κάθε ακολουθία (**sequence**) μέσα σε αυτό, όπου η κάθε μια αναφέρεται στις λέξεις ενός κειμένου με την σειρά που εμφανίζονται και αποτελείται όπως θα αναφερθεί και παρακάτω από τα indices των λέξεων αυτών στον πίνακα των embeddings, μαζί με το index για το padding (0) στο τέλος αν έχει προστεθεί.

Το batch αυτό περνάει από το **embedding layer** ώστε να γίνει η αντιστοίχιση των παραπάνω indices στα **πραγματικά vectors** της κάθε λέξης οπότε πρακτικά λειτουργεί σαν ευρετήριο. Αν έχει οριστεί και dropout, το input αυτό περνάει στη συνέχεια και από ένα **dropout layer** ώστε να γίνεται κάθε φορά η εκπαίδευση (**training**) με διαφορετικά κομμάτια του input πράγμα το οποίο μπορεί να βοηθήσει αρκετά στην αποφυγή του overfitting. Σε αυτό το σημείο αποθηκεύεται το παραπάνω input (έπειτα από την εφαρμογή του shortcut), ώστε να χρησιμοποιηθεί ως **skip connection** (αν έχει οριστεί) και έτσι να γίνει η **σύνδεση** του με κάποιο **output** από επόμενα layers. Έπειτα, πριν από κάθε layer γίνεται ένα **πακετάρισμα** στο input του, με βάση το πραγματικό μήκος που έχει υπολογιστεί και δοθεί από τον Dataloader, για την πιο αποτελεσματική επεξεργασία του από το αντίστοιχο RNN σε αυτό το layer μιας και αυτό το πακετάρισμα **αφαιρεί** το padding και το RNN μπορεί να λειτουργήσει γρηγορότερα χωρίς **άσκοπους υπολογισμούς** που θα έκανε και για εκείνο. Επίσης, αξίζει να σημειωθεί πως το τελευταίο έπαιζε και πολύ μεγάλο ρόλο στα αποτελέσματα του RNN μιας και όταν το padding ήταν αρκετά μεγάλο η **απόδοση** του ως προς όλες τις μετρικές ήταν πάρα πολύ χαμηλή (εφόσον τα τελευταία hidden states και τα cells επηρεάζονταν αρκετά από το padding). Οπότε, όταν το RNN έχει τελειώσει την επεξεργασία του input που του δόθηκε, το output του είναι και αυτό πακεταρισμένο οπότε ξεπακετάρεται αφού του προστεθεί πάλι padding. Αν έχει οριστεί και dropout περνάει στη συνέχεια και από ένα dropout layer. Τέλος, πριν περαστεί το output του τρέχοντος layer ως είσοδος στο επόμενο αν έχουν οριστεί **skip connections** και αν είναι το layer το οποίο έχει σειρά ώστε να γίνει η σύνδεση με το αποθηκευμένο προηγούμενο input τότε θα γίνει η πρόσθεση του output του με το τελευταίο αποθηκευμένο input ώστε να προκύψει η πρόσθεση με την ταυτοτική συνάρτηση (**f(x) + x**) και να γίνει έτσι προσομοίωση των **residual connections** (επειδή ορίζονται κυρίως για CNNs). Το αποτέλεσμα αυτό αποθηκεύεται ως το επόμενο νέο input που θα ενωθεί αργότερα.

Ύστερα από την επεξεργασία όλων των layers προκύπτει η ανάγκη της ενοποίησης (**pooling**) των διαφορετικών hidden states του τελευταίου layer. Ορίζονται λοιπόν 4 διαφορετικοί τρόποι για την επίτευξη του σκοπού αυτού. Ο πρώτος συναντάται σε αρκετές πηγές και μπορεί να εφαρμοστεί μόνο σε RNN δύο κατευθύνσεων (**bidirectional**) και ουσιαστικά χρησιμοποιεί μόνο το **τελευταίο hidden state από το forward** και μόνο το **τελευταίο hidden state από το backward** RNN τα οποία ύστερα τα **συνενώνει**. Ο δεύτερος και ο τρίτος αναφέρθηκαν και στο μάθημα και ουσιαστικά υπολογίζει ένα **element wise mean ή max όλων των hidden states** (στην περίπτωση του bidirectional εφαρμόζεται συνολικά έπειτα από την συνένωση των αντίστοιχων hidden states των δύο κατευθύνσεων η οποία γίνεται αυτομάτως). Ο τέταρτος και

τελευταίος ο οποίος είναι και ο προκαθορισμένος και ο πιο απλός χρησιμοποιεί απλά μόνο το **τελευταίο hidden state του forward** RNN το οποίο έρχεται μαζί **με το πρώτο hidden state του backward** αν είναι bidirectional (βέβαια αυτό πιθανώς να μην αποτελεί και την καλύτερη λύση μιας και το backward RNN δεν έχει δει όλο το sequence και δεν επιτυγχάνεται η πλήρης εκμετάλλευση του). Ταυτόχρονα, οι παραπάνω τρόποι έχουν διαμορφωθεί έτσι ώστε να λαμβάνουν υπόψη μόνο τα **έγκυρα hidden states** (μιας και υπάρχουν και hidden states που αποτελούν θόρυβο λόγω padding). Ύστερα, αν έχει οριστεί **attention layer** δίνεται το τελευταίο state (στην περίπτωση του bidirectional θεωρείται ως τελευταίο state η ένωση των τελευταίων hidden states των δύο κατευθύνσεων όπως στον πρώτο τρόπο που αναφέρθηκε παραπάνω) **ως query** με όλα τα **hidden states ως context** που παράχθηκαν στο τελευταίο layer ώστε να κάνει attend σε αυτά και έτσι να βρεθεί σε ποια αναφέρεται σε μεγάλο βαθμό. Επίσης, δίνεται και η **τελική αναπαράσταση** που παράχθηκε από κάποιον από τους παραπάνω τρόπους για να ενωθεί σε αυτή το attention (στη περίπτωση που θα εφαρμοστεί το attention στη συνέχεια στο καλύτερο μοντέλο η τελική αναπαράσταση θα είναι ίδια με το query μιας και ως τελική αναπαράσταση επιλέγεται επίσης η ένωση των τελευταίων hidden states των δύο κατευθύνσεων). Αυτό που περιγράφηκε αποτελεί ένα είδος **self attention** μιας και το πρόβλημα του classification είναι **seq2one** και όχι **seq2seq** αφού δεν υπάρχει decoder. Στο τέλος, γίνεται το **τελικό projection** της τελικής αναπαράστασης στον αριθμό των κλάσεων.
Κλείνοντας, αξίζει να σημειωθεί πως επειδή δεν ήταν ξεκάθαρο το πώς θα γίνει stacked το RNN κυρίως μαζί με το bidirectional κομμάτι η υλοποίηση επαληθεύθηκε και με τα αποτελέσματα της ίδιας της βίβλιοθήκης (υπάρχει σε σχόλιο ένα κομμάτι που βοήθησε στην επαλήθευση εφόσον έγινε χρήση των ίδιων των τιμών στα βάρη (**weights**) με το αντίστοιχο bidirectional stacked RNN της βιβλιοθήκης). Το συμπέρασμα από αυτό ήταν πως συνδέεται όλο το output (forward μαζί με backward hidden states έπειτα από συνένωση) ως είσοδος στο επόμενο layer και δεν πραγματοποιείται χωριστά από το forward και χωριστά από το backward όλη η διαδικασία και στο τέλος να γίνεται η συνένωση αυτή.

- **Training ανά batch** (batchTrain): Προστέθηκε μόνο η δυνατότητα για **gradient clipping** μετά το backward pass και πριν το βήμα του optimizer. Ο τρόπος που ακολουθήθηκε προτάθηκε και στο μάθημα και βασίζεται στο **max norm clipping** το οποίο κανονικοποιεί το gradient αν η νόρμα του ξεπεράσει κάποιο **threshold** ώστε με αυτόν τον τρόπο να αποφευχθούν τα **exploding gradients**.
- **Training** (trainNN): Αλλαγή στον Dataloader με την προσθήκη της συνάρτησης για το padding.

- **Training με επιλογές** (trainNNWithOptions): Αλλαγή στον Dataloader με την προσθήκη της συνάρτησης για το padding.
- **Προβλέψεις** (predict): Αλλαγή ώστε να γίνεται predict ανά batch.

Πριν τεθεί το μοντέλο σε εφαρμογή έγιναν επίσης και κάποιες αλλαγές στις συμπληρωματικές συναρτήσεις για την αξιολόγηση του μοντέλου. Αρχικά, προστέθηκε η δυνατότητα να γίνεται το **evaluation ανά batch** και επίσης διαμορφώθηκε το **grid search ως προς τις νέες παραμέτρους**.

Εκπαίδευση του RNN μοντέλου με την χρήση GLoVe embeddings

Η διαδικασία φόρτωσης των GLoVe embeddings τοπικά παραμένει η ίδια με αυτή της προηγούμενης εργασίας. Το embedding αρχείο που επιλέχθηκε ήταν αυτό των **50 διαστάσεων**, το οποίο είναι και αυτό πάνω σε **tweets** και το οποίο λειτουργούσε με παρόμοια αποτελέσματα και πολλές φορές καλύτερα από αυτά των πιο μεγάλων διαστάσεων. Όταν το αρχείο φορτωθεί δημιουργείται ένα λεξιλόγιο (**vocabulary**) από όλες τις λέξεις που συναντώνται **αποκλειστικά** στο training set. Ύστερα, διαβάζεται **γραμμή-γραμμή** το αντίστοιχο αρχείο και σε κάθε τέτοια γραμμή η οποία αναφέρεται σε κάποια λέξη γίνεται ο έλεγχος για το αν αυτή η λέξη περιέχεται στο λεξιλόγιο. Άμα περιέχεται, διαχωρίζεται η λέξη από το διάνυσμα της που βρίσκεται και αυτό στην αντίστοιχη γραμμή και ύστερα αυτό τοποθετείται σε έναν **πίνακα** που διατηρεί και όλα τα υπόλοιπα embeddings. Επιπλέον, διατηρείται μια αντιστοιχία (**mapping**) της λέξης με τη θέση (**index**) του διανύσματος της στον πίνακα όπως ακριβώς στην προηγούμενη άσκηση. Βέβαια, για όσες λέξεις του λεξιλογίου **δεν βρέθηκε** κάποιο embedding στο αρχείο δημιουργείται ένα τυχαίο διάνυσμα (**random vector**). Παράλληλα, προστέθηκε και ένα ακόμη διάνυσμα για το **padding** το οποίο είναι ένα διάνυσμα μόνο με **μηδενικά** και παράλληλα προστέθηκε και ένα «default» διάνυσμα για τις λέξεις που **δεν υπάρχουν στο λεξιλόγιο** (**unkown**) το οποίο είναι ένα διάνυσμα που προκύπτει από τον **μέσο όρο** όλων των υπολοίπων στον πίνακα. Στο τέλος δημιουργείται για κάθε κείμενο του training set, μια ακολουθία (**sequence**) από τις θέσεις (**indices**) των διανυσμάτων που βρίσκεται **στον πίνακα** η **κάθε λέξη** που αποτελείται και κατασκευάζεται το dataset που αναφέρθηκε στην αρχή ώστε στην συνέχεια να δοθεί στον DataLoader ο οποίος θα φτιάχνει batches και θα προσθέτει αυτόματα το padding. Η ίδια διαδικασία ακολουθείται και για το validation set. Αξίζει να αναφερθεί πως υλοποιήθηκε ακριβώς η ίδια διαδικασία και με την χρήση της βιβλιοθήκης torchtext όμως τα αποτελέσματα ήταν παρόμοια οπότε επιλέχθηκε αυτή η υλοποίηση η έγινε από το μηδέν μιας και αυτό θα έδινε περισσότερη ευελιξία και περισσότερη βεβαιότητα στη χρήση.

Αξιολόγηση του μοντέλου

Η αξιολόγηση του μοντέλου θα γίνει ως προς τους διαφορετικούς **τύπους** για RNN (Vanilla, LSTM, GRU) και ως προς τις διαφορετικές **υπερπαραμέτρους** των μοντέλων στις εξής κατηγορίες:

1) Τιμές για learning rates
2) Αριθμός των νευρώνων (**units**) στα hidden states
3) Αριθμός των stacked layers και skip connections
4) Τιμές των max norms για gradient clipping
5) Τιμές για dropout
6) Κωδικοποίηση (**encoding**) των τελικών hidden states

+ **Συνδυασμός κάποιων καλών τιμών**

Επίσης υπάρχουν και κάποιες καλές παράμετροι (μετά από πειράματα) οι οποίες παραμένουν σταθερές σε όλες τις περιπτώσεις από 1 έως 6:

- Epochs : 15
- Batch size : 128
- Early stopping patience : 3
- L2 regularization : 0
- Sequence length : 50 (η επιλογή προέκυψε από το μέσο μήκος των tweets που είναι περίπου ίσο με 77 ώστε να καλύπτονται τα περισσότερα)
- 2 κατευθύνσεις (bidirectional)

Οι αξιολογήσεις των παραπάνω περιπτώσεων θα πραγματοποιηθούν **με την σειρά** που αναφέρθηκαν η οποία ήταν και η πιο λογική μιας και προέκυψε έπειτα από **αρκετά πειράματα**. Παράλληλα από τα πειράματα αυτά εκτός από τις παραπάνω σταθερές παραμέτρους βρέθηκαν και κάποιες **καλές τιμές** για κάθε περίπτωση (σε κάθε περίπτωση θα **υπογραμμίζεται**). Οπότε σε κάθε περίπτωση (εκτός από την περίπτωση για το dropout) θα συγκρίνεται κάθε τέτοια τιμή με κάποια μικρότερη/πιο απλή ή μεγαλύτερη/πιο σύνθετη της που ίσως να είναι καλύτερη από αυτή. Δηλαδή η τιμή που φάνηκε από τα πειράματα να **αποδίδει καλά** σε κάθε περίπτωση είναι μια **ενδιάμεση τιμή** που είτε θα πρέπει να **αλλαχθεί** (π.χ να αυξηθεί ή να μειωθεί) είτε θα **παραμένει ως έχει**. Για να γίνουν όμως αυτές οι συγκρίσεις σε κάθε βήμα **φιξάρονται** οι **καλύτερες τιμές** από τα προηγούμενα. Βέβαια κάθε τέτοιο βήμα απαιτεί και κάποιες τιμές από επόμενα βήματα οπότε οι τιμές αυτές θα παίρνουν τις αντίστοιχες ενδιάμεσες. Με λίγα λόγια είναι μια πιο

άπληστη προσέγγιση γι αυτό βέβαια υπάρχει και μία επιπλέον περίπτωση που συνδυάζει κάποιες καλές τιμές ώστε να προκύψει και το καλύτερο μοντέλο.

1) Τιμές για learning rates

Στην περίπτωση αυτή συγκρίνονται:

a. 0.1 με exponential schelduler
b. 0.005 χωρίς schelduler
c. 0.0003 χωρίς schelduler

Συμπέρασμα στο LSTM RNN: Η περίπτωση a δίνει καλά learning curves όμως με αρκετά **χαμηλά scores**. Η περίπτωση b δίνει πιο καλά scores όμως το φαινόμενο του **overfitting** παρατηρείται αρκετά έντονα. Τέλος, η καλύτερη περίπτωση φαίνεται να είναι η c μιας και δίνει **αρκετά καλά scores** με **αρκετά καλά learning curves**.

Συμπέρασμα στο GRU RNN: Ομοίως με LSTM.

Συμπέρασμα στο Vanilla RNN: Παρόμοια με τα παραπάνω όμως παρατηρούνται **χειρότερα scores**. Επιπλέον παρατηρούνται και **χειρότερα learning curves** ειδικά στην περίπτωση b και λίγο λιγότερο στην c όπου παρατηρείται όμως αρκετό overfitting. Βέβαια και εδώ η περίπτωση c είναι λίγο καλύτερη από τις υπόλοιπες.

Γενικό Συμπέρασμα: Η περίπτωση c δίνει τα καλύτερα αποτελέσματα. Με την περίπτωση c παρατηρήθηκε και τεράστια βελτίωση στην **αντιμετώπιση** του **overfitting** που ήταν ένα αρκετά σημαντικό πρόβλημα που παρουσιαζόταν αρχικά στο μοντέλο. Σε αυτό βέβαια συμβάλει και ο **Adam optimizer** που προσαρμόζει κατάλληλα το learning rate.

2) Αριθμός των νευρώνων (**units**) στα hidden states

Στην περίπτωση αυτή συγκρίνονται:

a. 64
b. 128
c. 256

Συμπέρασμα στο LSTM RNN: Όλες οι περιπτώσεις έχουν παρόμοια καλά scores. **Βέβαια όσο αυξάνονται τα hidden units παρατηρείται όλο και περισσότερο overfitting**.

Συμπέρασμα στο GRU RNN: Ομοίως με LSTM.

Συμπέρασμα στο Vanilla RNN: Παρόμοια με τα παραπάνω. Παρατηρούνται και εδώ χειρότερα scores και επίσης φαίνεται πως δεν μπορεί να λειτουργήσει **καθόλου καλά** για μεγάλο αριθμό από hidden units όπως στην περίπτωση c.

Γενικό Συμπέρασμα: Τα hidden units επηρεάζουν αρκετά τον βαθμό του overfitting. **Όσο μεγαλύτερος αριθμός από hidden units τόσο μεγαλύτερο overfitting**. Άρα το μοντέλο μπορεί να μάθει καλά με λιγότερα hidden units. Αυτό πιθανώς να σχετίζεται αρκετά και με τη διάσταση του input στο μοντέλο η οποία είναι και αυτή μικρή και ίση με 50. Δηλαδή περίπτωση a είναι η καλύτερη μιας και είναι η πιο κοντινή στη διάσταση αυτή του input.

3) Αριθμός των stacked layers και skip connections

Στην περίπτωση αυτή συγκρίνονται:

a. 1 layer χωρίς skip connections
b. 2 layers χωρίς skip connections
c. 2 layers με skip connections ανά 2 layers (ενώνεται το αρχικό input με το τελικό output δηλαδή μοίαζει με ένα απλό residual block με 2 layers)
d. 3 layers χωρίς skip connections
e. 3 layers με skip connections ανά 1 layer (ενώνεται κάθε input με κάθε output)
f. 4 layers χωρίς skip connections
g. 4 layers με skip connections ανά 2 layers (ενώνεται το αρχικό input με το ενδιάμεσο output (στο layer 2) και αυτό με το τελικό output δηλαδή σαν να υπάρχουν 2 residual blocks με 2 layers το καθένα)

Συμπέρασμα στο LSTM RNN: Το μοντέλο φαίνεται να τα πηγαίνει αρκετά καλά με ικανοποιητικά scores και learning curves χωρίς πολύ overfitting **και μόνο με 1 layer**. Χρησιμοποιώντας μερικά layers ακόμα δηλαδή έχοντας 2 ή 3 ή 4 layers δεν αλλάζουν αρκετά τα learning curves όμως στην μεγαλύτερη πλειοψηφία των εκτελέσεων παράγουν παρόμοια ή και καλύτερα scores.

Η προσθήκη των skip connections **δεν βοηθάει απαραίτητα** στην βελτίωση των scores όμως προκαλεί αλλαγή στα learning curves μιας και φαίνεται να βοηθάει το μοντέλο να **συγκλίνει πιο απότομα** αφού το **training curve** έχει **πιο απότομη κλίση** και σταματάει πιο χαμηλά στις ίδιες επαναλήψεις. Αυτό βέβαια **δεν συμβαίνει** και για το **validation curve** με αποτέλεσμα να φαίνεται **εντονότερο το overfitting**. Το φαινόμενο αυτό παρατηρείται αρκετά στα πιο πολλά layers (3 και 4) μιας και είναι λογικό αφού τα gradients «ρέουν» πιο εύκολα (λόγω του **αθροίσματος** που γίνεται κατά την ένωση διευκολύνεται η ροή των gradients στο backpropagation εξαλείφοντας και τυχόν vanishing gradients μεταξύ των layers)

Συμπέρασμα στο GRU RNN: Ομοίως με LSTM. Παρόλα αυτά φαίνεται πως όσο αυξάνεται ο αριθμός των layers υπάρχει περίπτωση να παράγονται λίγο λιγότερο ομαλά learning curves.

Συμπέρασμα στο Vanilla RNN: Παρόμοια με τα παραπάνω με την μόνη διαφορά πως όσο αυξάνονται τα layers δεν φαίνεται να βοηθάει το μοντέλο όπως φαίνεται και από τα learning curves λόγω του overfitting. Και εδώ τα scores είναι μικρότερα.

Γενικό Συμπέρασμα: Φαίνεται πως το μοντέλο τα πηγαίνει αρκετά καλά για όλους τους διαφορετικούς αριθμούς από layers, ωστόσο τα πηγαίνει πιο καλά κυρίως για **μικρότερους αριθμούς**. Αυτό σημαίνει πως είναι πιο **εύκολο** το **πρόβλημα** κατηγοριοποίησης που έχει να επιλύσει το μοντέλο μιας και τα δεδομένα που δίνονται για να εκπαιδευτεί δεν είναι και πάρα πολλά. Ο αριθμός των layers που ξεχώρισε ήταν τα 2 και τα 3 layers (περίπτωση b, c) όπου είχαν παρόμοια αποτελέσματα με το 1 layer αλλά κατά πλειοψηφία καλύτερα. Τα skip connections βοήθησαν αρκετά τον μεγαλύτερο αριθμό από layers να **συγκλίνει πιο απότομα χωρίς** ωστόσο να συμβάλλουν ουσιαστικά στην **βελτίωση της απόδοσης του**.

*Μέχρι αυτό το σημείο έχει φανεί πως το LSTM και το GRU είναι αρκετά **ανταγωνιστικά** μεταξύ τους. Από τους χρόνους που κάνουν τις περισσότερες φορές το **GRU είναι πιο γρήγορο** (βέβαια δεν είναι τόσο αισθητό με τη χρήση της GPU) μιας και έχει λιγότερη πολυπλοκότητα στην αρχιτεκτονική του.*

*Επίσης φαίνεται πως το απλό **Vanilla** RNN **δεν μπορεί να ξεπεράσει** σε καμία περίπτωση το LSTM και το GRU. Αυτό είναι αρκετά αναμενόμενο αφού τα δύο τελευταία χρησιμοποιούν ένα **είδος μνήμης** (cells) η οποία τα βοηθάει να θυμούνται αρκετά προηγούμενα states (αυτό επιτυγχάνεται σε συνδυασμό με τα διαφορετικά **gates** που μαθαίνει το μοντέλο δυναμικά βοηθώντας την ροή των gradients κατά το backpropagation through time). Βέβαια τα αποτελέσματα του δεν είναι και πολύ χαμηλά και αυτό μπορεί να οφείλεται στο γεγονός πως τα tweets δεν είναι αρκετά μεγάλα σε μήκος. Παρόλα αυτά είναι μέχρι στιγμής **χειρότερο** από τα άλλα δύο και γι αυτό θα συνεχίσουν με αυτά οι υπόλοιπες συγκρίσεις οι οποίες αφορούν κυρίως κάποια κομμάτια περεταίρω βελτιστοποίησης.*

4) Τιμές των max norms για gradient clipping

Στην περίπτωση αυτή συγκρίνονται:

a. Χωρίς gradient clipping
b. 1
c. 5
d. 10

Συμπέρασμα στο LSTM RNN: Όλες οι τιμές των διαφορετικών max norms έχουν **παρόμοια επίδραση** σε σχέση με την περίπτωση που δεν χρησιμοποιείται καθόλου gradient clipping.

Συμπέρασμα στο GRU RNN: Ομοίως με LSTM.

Γενικό Συμπέρασμα: Φαίνεται πως η εφαρμογή του **gradient clipping** δεν έχει **καμία επίδραση**. Αυτό σημαίνει πως **δεν συναντάται καθόλου το πρόβλημα των exploding gradients** μιας και αυτός είναι ο κύριος λόγος που γίνεται το gradient clipping. Δηλαδή θα περίμενε κανείς στην περίπτωση a που δεν εφαρμόζεται gradient clipping τεράστιες ενημερώσεις στα βάρη του RNN τα οποία θα οδηγούσαν σε απρόσμενα αποτελέσματα. Παρόλα αυτά καλό είναι να εφαρμόζεται ένα **φράγμα** ώστε να υπάρχει πάντοτε μια εγγύηση πως δεν θα υπάρξει πρόβλημα με τυχόν exploding gradients. Γι αυτό θα επιλεγεί η περίπτωση c μιας και αποτελεί μια ενδιάμεση τιμή.

5) Τιμές για dropout

Στην περίπτωση αυτή συγκρίνονται:

a. 0.2
b. 0.4
c. 0.6

Συμπέρασμα στο LSTM RNN: Φαίνεται πως αρκεί μια **μικρή τιμή** πιθανότητας για το dropout η οποία οδηγεί στην τελική σύγκλιση μεταξύ του training και του validation learning curve. Αυτό σημαίνει πως **εξαλείφεται το overfitting** όπως φαίνεται στην περίπτωση a. Βέβαια αν τεθεί και μια τιμή λίγο μεγαλύτερη όπως στην περίπτωση b φαίνεται το overfitting να τείνει στο να εξαλειφθεί αλλά χρειάζονται **ακόμα μερικές επαναλήψεις (epochs)** για επιτευχθεί αυτό. Ωστόσο μια μεγάλη τιμή όπως αυτή της περίπτωσης c έχει ως αποτέλεσμα τα training και validation learning curves **να απέχουν αρκετά** οπότε δεν αντιμετωπίζεται αποτελεσματικά το overfitting.

Συμπέρασμα στο GRU RNN: Ομοίως με το LSTM.

Γενικό Συμπέρασμα: Όπως είναι γνωστό και από την θεωρία η τεχνική του dropout λειτουργεί ως **regularizer** για την αποτελεσματική αντιμετώπιση του overfitting μιας και **δεν γίνεται η χρήση κάποιων units**. Το μοντέλο χρησιμοποιεί dropout στο αρχικό input από το embedding layer αλλά και ανάμεσα στα stacked layers που σημαίνει πως το κάθε layer εκπαιδεύεται **με ένα κομμάτι** του input του. Έτσι το μοντέλο μαθαίνει και συνδυάζει ένα σύνολο από επιμέρους διαφορετικά μοντέλα πράγμα που το καθιστά αρκετά ισχυρό. Ταυτόχρονα, φάνηκε πως η καλύτερη τιμή για το μοντέλο είναι η πιο μικρή (περίπτωση a) όμως και η λίγο πιο μεγάλη (περίπτωση b) έχει αρκετές προοπτικές. Τέλος, μπορεί κανείς να παρατηρήσει πως το validation loss είναι πάντα πιο χαμηλά από το training loss. Αυτό συμβαίνει διότι το validation loss προκύπτει από προβλέψεις του μοντέλου που χρησιμοποιούνται όλα τα units.

6) Κωδικοποίηση (**encoding/pooling**) των τελικών hidden states

Στην περίπτωση αυτή συγκρίνονται:

a. Συνδυασμός του τελευταίου hidden state του forward RNN μαζί με το πρώτο hidden state του backward RNN του τελευταίου layer
b. Συνδυασμός του τελευταίου hidden state του forward RNN μαζί με το τελευταίο hidden state του backward RNN του τελευταίου layer
c. Element wise mean των αναπαραστάσεων που προκύπτουν από την συνένωση των αντίστοιχων hidden states και των δύο κατευθύνσεων του τελευταίου layer
d. Element wise max των αναπαραστάσεων που προκύπτουν από την συνένωση των αντίστοιχων hidden states και των δύο κατευθύνσεων του τελευταίου layer

Συμπέρασμα στο LSTM RNN: Όλες οι περιπτώσεις παράγουν **αρκετά παρόμοια** και καλά αποτελέσματα τα οποία είναι δύσκολο να διακριθούν μεταξύ τους.

Συμπέρασμα στο GRU RNN: Ομοίως με LSTM.

Γενικό Συμπέρασμα: Τελικά διαπιστώθηκε πως οι δύο πρώτες περιπτώσεις δεν διαφέρουν και πάρα πολύ. Συνήθως η περίπτωση b βγάζει καλύτερα scores αφού το forward και το backward RNN **έχουν σαρώσει όλο το sequence** αντίστοιχα. Επίσης θα περίμενε κανείς τις δύο τελευταίες περιπτώσεις να είναι με διαφορά οι καλύτερες μιας και λαμβάνουν υπόψη όλα τα hidden states του τελευταίου layer αλλά τελικά παρατηρήθηκε πως **δεν μπορούν συνεισφέρουν κάτι αρκετά καλύτερο** με βάση και τις υπόλοιπες υπερπαραμέτρους.

Καλύτερο μοντέλο

Από τον **συνδυασμό των καλύτερων τιμών** προέκυψε το εξής καλύτερο μοντέλο:

- Τύπος: GRU (τελικά φάνηκε πως λειτουργεί πιο καλά από το LSTM λόγω των περιορισμένων δεδομένων)
- Learning rate: 0.0003 με performance schelduler (ο schelduler βοήθησε το validation loss να γίνει λίγο πιο ομαλό μιας και έκανε αρκετά ανεβοκατεβάσματα)
- Hidden units: 64
- Layers/Stacks: 2
- Χωρίς skip connections
- Max clip norm: 5
- Dropout: 0.2
- Encoding/Pooling: Τελευταία states από forward και backward RNNs
- Epochs : 35 (αυξήθηκαν διότι το μοντέλο μπορούσε να μάθει ακόμα καλύτερα με περισσότερά)
- Batch size : 128
- Early stopping patience : 5
- L2 regularization : 0
- Sequence length : 50
- 2 κατευθύνσεις (bidirectional)

Το μοντέλο αυτό είχε αρκετά ικανοποιητική επίδοση μιας και έχει καταφέρει περίπου **72.1% accuracy** και **71.6% f1 score**. Σε άλλες εκτελέσεις μπορεί να πέφτουν λίγο τα scores αυτά αλλά στην πλειοψηφία των εκτελέσεων διατηρούνται πάνω από **71%**. Επίσης τα learning curves του δεν είναι και τα τέλεια όπως μπορεί να προκύπτουν από άλλες περιπτώσεις όμως οι περιπτώσεις αυτές χάνουν πολύ από τα scores που παράγουν. Άρα αν ισοσταθμιστούν οι δύο παράγοντες αυτοί το μοντέλο αυτό αποτελεί τη καλύτερη επιλογή.

Όσον αφορά τα **roc curves** της κάθε κλάσης παρατηρείται πως απέχουν αρκετά **μακριά** από την **διαγώνιο** οπότε είναι μια καλή ένδειξη ότι οι προβλέψεις δεν είναι τελείως τυχαίες. Βέβαια, δεν αποτελούν και τις καλύτερες καμπύλες εφόσον απέχουν αρκετά από την πάνω αριστερά γωνία. Ωστόσο, σύμφωνα με το roc auc score καταλαβαίνει κανείς πως η κλάση neutral που έχει το μεγαλύτερο auc score (89%) μπορεί **να διακριθεί πιο εύκολα** (πράγμα αρκετά καλό μιας και αποτελεί την ενδιάμεση κλάση που εύκολα θα μπορούσε να κάνει λάθος το μοντέλο) από το μοντέλο σε σχέση με τις μη neutral (antivax + provax μαζί) ενώ αντίστοιχα στη συνέχεια

το ίδιο ισχύει σε έναν μικρότερο βαθμό για την antivax κλάση (πράγμα αρκετά καλό μιας και δεν υπάρχουν τόσα tweets για αυτή την κλάση όσα και στις υπόλοιπες) σε σχέση με τις μη antivax (neutral + provax μαζί) και σε ακόμα πιο μικρό βαθμό για την provax κλάση σε σχέση με τις μη provax (neutral + antivax μαζί), χωρίς όμως να πέφτει γενικά το auc score κάτω από το 84% που σημαίνει πως οι προβλέψεις της κάθε κλάσης για άγνωστες οντότητες είναι αρκετά καλές.

Σύγκριση με το καλύτερο μοντέλο της Εργασίας 1 και 2

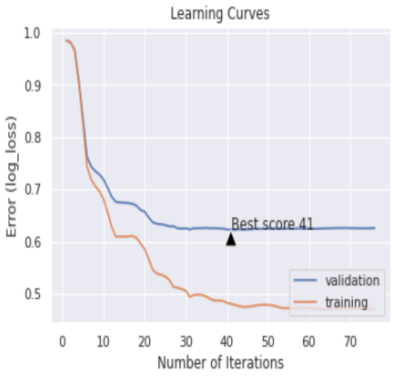
Softmax Regression

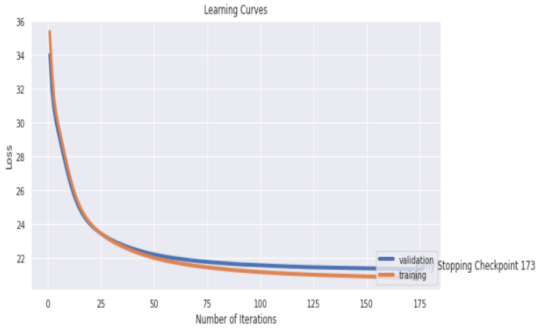
	precision	recall	f1-score
neutral	0.78525	0.80000	0.79256
antivax	0.71739	0.44595	0.55000
provax	0.69694	0.76656	0.73009
accuracy			0.74058
macro avg	0.73319	0.67083	0.69088
weighted avg	0.74081	0.74058	0.73589

Feed Forward Neural Network

	precision	recall	f1-score
neutral	0.764759	0.790610	0.777470
antivax	0.701493	0.317568	0.437209
provax	0.664756	0.755700	0.707317
macro avg	0.710336	0.621293	0.640665
weighted avg	0.716192	0.715162	0.705021

Accuracy: 0.71516

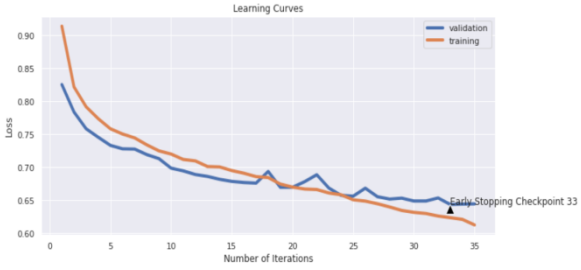




Recurrent Neural Network (**GRU**)

	precision	recall	f1-score
neutral	0.797348	0.790610	0.793965
antivax	0.567961	0.395270	0.466135
provax	0.673529	0.745928	0.707883
macro avg	0.679613	0.643936	0.655994
weighted avg	0.717622	0.721297	0.716700

Accuracy: 0.72130



Φαίνεται πως το καλύτερο μοντέλο αυτής της άσκησης προκύπτει ως **κάτι ενδιάμεσο** των δύο άλλων. Πιο συγκεκριμένα, τα **scores** που παράγει είναι πιο **κοντινά** και **λίγο χειρότερα** από αυτά του **softmax regression** που είχε φανεί το καλύτερο σε scores και παραμένει μέχρι στιγμής αλλά φαίνεται να έχει πολύ **καλύτερα learning curves** από αυτό χωρίς τόσο μεγάλο overfitting. Αντιθέτως έχει **καλύτερα scores** από το **feed forward neural network** σε αρκετά **λιγότερες επαναλήψεις** όμως δεν καταφέρνει να έχει **τόσο ομαλά learning curves** όπως αυτό. Παρόλα αυτά δεν τίθεται κάποιο πρόβλημα μιας και τα training και validation curves παραμένουν αρκετά κοντά ειδικά στο τέλος. Επιπλέον, το καλύτερο μοντέλο αυτό παράγει και λίγο καλύτερα **roc curves** με παρόμοια ή λίγο μεγαλύτερα auc scores που σημαίνει πως μπορεί να **διαχωρίσει/ξεχωρίσει** καλύτερα τη κάθε κλάση όπως αυτό διαπιστώνεται και στα **confusion matrices**. Να σημειωθεί πως τα αποτελέσματα του feed forward neural network που παρουσιάζονται αφορούσαν την περίπτωση του **TF-IDF** που ήταν καλύτερη από αυτή του **GloVe**. Άρα φαίνεται πως το μοντέλο αυτό της άσκησης χρησιμοποίησε τα GloVe embeddings πολύ **πιο αποτελεσματικά** μιας και τώρα η κάθε αναπαράσταση της κάθε λέξης δινόταν ως έχει σε κάθε timestep και δεν χρειαζόταν να γίνει κάποια πρόσμιξη (**aggregation**). Κλείνοντας αξίζει να αναφερθεί ακόμα πως το **λεξιλόγιο** που χρησιμοποιήθηκε σε αυτή την άσκηση είναι **αρκετά μικρότερο** των άλλων δύο και παρόλα αυτά το καλύτερο μοντέλο αυτής της άσκησης είναι αρκετά ανταγωνιστικό απέναντι στα καλύτερα μοντέλα τους. Αυτό και όλα τα παραπάνω δείχνουν και την **ισχύ** που έχουν τα RNNs λόγω της αποθήκευσης της κατάστασης παλαιότερων καταστάσεων η οποία ενισχύεται σε μεγάλο βαθμό και με την μνήμη που αυτά πιθανώς να διαθέτουν και επιπλέον της ακολουθιακής αλλά και διπλής σάρωσης που κάνουν για τις ακολουθίες που δέχονται ως input και από τις δύο κατευθύνσεις.

Προσθήκη Attention στο καλύτερο μοντέλο

Στο καλύτερο μοντέλο προστέθηκε και ένα attention layer και πιο συγκεκριμένα **ένα για κάθε διαφορετικό τύπο** που αναφέρθηκε και παραπάνω (**dot product** και **multiplicative**). Τα αποτελέσματα **δεν** ήταν τα **αναμενόμενα** μιας και κανείς θα περίμενε την **εκτόξευση των scores** αφού βοηθάει το μοντέλο να εστιάσει σε συγκεκριμένες σημαντικές λέξεις μέσα σε μια πρόταση αλλά παρόλα αυτά τα scores παρέμειναν αρκετά παρόμοια και λίγο πιο χαμηλά. Πολλοί είναι οι λόγοι που πιθανώς να παρατηρήθηκε αυτό το φαινόμενο. Ένας λόγος είναι πως γενικά το καλύτερο μοντέλο **είχε φτάσει ήδη** σε ένα πολύ καλό επίπεδο στα scores του σε σύγκριση με τα υπόλοιπα

μοντέλα που δοκιμάστηκαν οπότε μπορεί κανείς να καταλάβει τα περιθώρια βελτίωσης. Ένας άλλος λόγος είναι πως η εφαρμογή του attention σε ένα **seq2one** μοντέλο μπορεί να μην είναι τόσο αποτελεσματική όσο σε κάποιο **seq2seq** όπου για τέτοια μοντέλα είχε προταθεί το attention. Δηλαδή, υπάρχουν κάποιοι άλλοι μηχανισμοί όπως το **seq2seq** και οι **transformers** που ίσως να εκμεταλλεύονται καλύτερα τα οφέλη του attention.

Οδηγίες για τη δοκιμή με το test set

Για να εξεταστεί το test set θα πρέπει αρχικά να φορτωθεί με το όνομα του validation set ή να δοθεί το σωστό μονοπάτι. Θα πρέπει επίσης να εκτελεστούν όλα τα κελία από την αρχή χωρίς όμως την υποπεριοχή που διεξάγονται όλα τα διαφορετικά πειράματα που περιγράφηκαν παραπάνω (Model Evaluation & Tuning Section) μιας και διαρκούν αρκετή ώρα και χρησιμοποιούσαν το validation set. **Προσοχή**: Θα πρέπει να γίνει αντικατάσταση της κλήσης model.trainNNWithOptions με την κλήση

```
model.trainNN(glove_training_set, optimizer, loss_functio
n, None, seq_length, max_clip_norm, epochs, batch_size)
```

Όπως φαίνεται δεν χρησιμοποιείται ο performance schelduler μιας και απαιτεί την παρουσία του validation set.

Βιβλιογραφία/Πηγές

- Διαφάνειες μαθήματος
- Pytorch documentation (tutorials + examples) & discussions
- Stack Overflow
- Stack Exchange
- Quora
- Reddit
- Πάρα πολλά άρθρα, blogs και forums για ιδέες πάνω στο θέμα όπως:
  https://towardsdatascience.com/multiclass-text-classification-using-lstm-in-pytorch-eac56baed8df
  https://www.analyticsvidhya.com/blog/2020/01/first-text-classification-in-pytorch
  https://towardsdatascience.com/building-efficient-custom-datasets-in-pytorch-2563b946fd9f

https://suzyahyah.github.io/pytorch/2019/07/01/DataLoader-Pad-Pack-Sequence.html
https://gist.github.com/HarshTrivedi/f4e7293e941b17d19058f6fb90ab0fec
https://www.kaggle.com/dannykliu/lstm-with-attention-clr-in-pytorch
https://androidkt.com/how-to-apply-gradient-clipping-in-pytorch
https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections

…

Ορφέας Τσουράκης,
1115201700175