



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Επεξεργασία Φωνής & Φυσικής Γλώσσας

3ο Εργαστήριον

Γεωργίου Δημήτριος (03115106)

<el15106@central.ntua.gr>

Μπαζώτης Νικόλαος (03115739)

<el15739@central.ntua.gr>

Φεβρουάριος 2019

1 Εισαγωγή - Προετοιμασία δεδομένων

- Από το δύο datasets επιλέχθηκε το **Semeval2017A**, το οποίο κατά μέσο όρο απαιτεί περισσότερο χρόνο training στις ίδιες συνθήκες από το αντίστοιχο **MR**. Η επιλογή έγινε για να μπορέσουμε να έχουμε συγκριτικά αποτελέσματα μεταξύ των μοντέλων των παρακάτω ερωτημάτων, συμπεριλαμβανομένου και του bonus.
- **TODO:** tokenizer used
- Γενικότερα για τα παρακάτω ερωτήματα, θέλουντας να υπολογίσουμε τις μετρικές accuracy, F1 score (macro average), recall (macro average) των μεθόδων που χρησιμοποιούμε, έγινε δοκιμή διαφόρων:
 - activation non-linearity
 - batch size
 - word embeddings
 - sentence length
- **pre-trained word embeddings**
 - Έγινε χρήση pre-trained word embeddings. Επομένως τα word embeddings αρχικοποιούνται όχι τυχαία άλλα με αυτά τα pre-trained vectors. Λαμβάνουμε αυτά τα vectors απλώς συγκεκριμένοποιώντας ποια vectors θέλουμε, η υπόλοιπη δουλειά γίνεται από την συνάρτηση `load_word_vectors()` του **load_embeddings**. Συγκεκριμένα για τον σκοπό του project χρισμοποιούνται **glove.twitter.27B.50d.txt** και **glove.twitter.27B.200d.txt**, με σκοπό να δούμε το τι επιρροή έχουν οι διαστάσεις των embeddings στις παραπάνω μετρικές, στην απόδοση δηλαδή του μοντέλου μας. Το 27B υποδηλώνει ότι αυτά τα vectors έγιναν train πάνω σε 27 δισ. tokens και το Xd υποδηλώνει ότι είναι vectors X διαστάσεων. Κάθε λέξη δηλαδή αντιστοιχεί σε ένα vector X διαστάσεων
 - Η θεωρία πίσω από αυτό είναι ότι αυτά τα pre-trained vectors έχουν λέξεις με similar semantic meaning πιο κοντά στον διανυσματικό χώρο. Αυτό οδηγεί σε "καλή" αρχικοποίηση του embedding layers, εφόσον δεν χρειάζεται να μάθει μοντέλο μας αυτές τις σχέσεις μεταξύ των λέξεων από την αρχή.
- **batch size**
 - Για το train του μοντέλου, του δίνουμε τις προτάσεις σε batches, δηλαδή σπάμε το σύνολο των προτάσεων σε υποσύνολα batches που όλα έχουν συγκεκριμένο batch size (εκτός ίσως από το τελευταίο).
- **sentence length**
 - Επιπλέον, η λειτουργικότητα του μοντέλου βασίζεται στο ότι όλες οι προτάσεις έχουν ίδιο length, άρα βρίσκουμε το συνολικό max length του train set (και test set), και όσες προτάσεις είναι μικρότερου μεγέθους από αυτές, συμπληρώνουμε με 0 (zero padding). Σημειώνεται ότι στην προπαρασκευή έγινε χρήση ένος average length, και οδηγήθηκαμε σε descent αποτελέσματα συγκριτικά με χρήση max length (ελάχιστες διαφορές), έδω χρησιμοποιούμε όμως max length με σκοπό την διευκόλυνση του υπολογισμού του πραγματικού last timestamp στα ερώτηματα που απασχολούν την χρήση LSTM μοντέλων
- **activation non linearity**
 - Χρήση `tanh()` και `relu()` ως συναρτήσεις activation. Δεν υπάρχει καλύτερη συνάρτηση, η απόδοση της εξαρτάται από το περιέχομενο των προτάσεων, τις μενόδους και τα layers που χρησιμοποιούμε για το μοντέλο κλπ. Χρησιμοποιώντας διαφορετικές activation συναρτήσεις δεν επηρεάζουμε τι μπορεί το network μας να μάθει, άλλα πόσο γρήγορα (πόσα δεδομένα/εποχές χρειάζεται). Με όλες θα οδηγηθούμε στο ίδιο accuracy το οποίο αποδεικνύεται αν κάνουμε train το μοντέλο σε βάθος χρόνου, για αρκετά μεγάλη περίοδο.
- **epochs**
 - Χρήση 40 εποχών.
- **hidden size**
 - Γενικά το ορίσαμε ως embedding size, παρόλα αυτά προς τα τελευταία ερωτήματα το ορίσαμε 20, οδηγούσε σε λιγότερο overfitting και γενικά μέιωση των losses που ήταν και το ζητούμενο μεταξύ άλλων.
- **optimizer**
 - Χρήση του Adam. Ο SGD ενημερώνει όλες τις παραμέτρους με το ίδιο learning rate και το να διαλέξεις αυτό το learning rate μπορεί να είναι tricky. Ο Adam ενημερώνει το learning rate για κάθε παράμετρο, δίνοντας στις παραμέτρους που ενημερώνονται πιο συχνά lower learning rates και στις παραμέτρους που ενημερώνονται λιγότερο συχνά higher learning rate.

2 Ερώτημα 1

1.1 Mean Pooling - Max Pooling

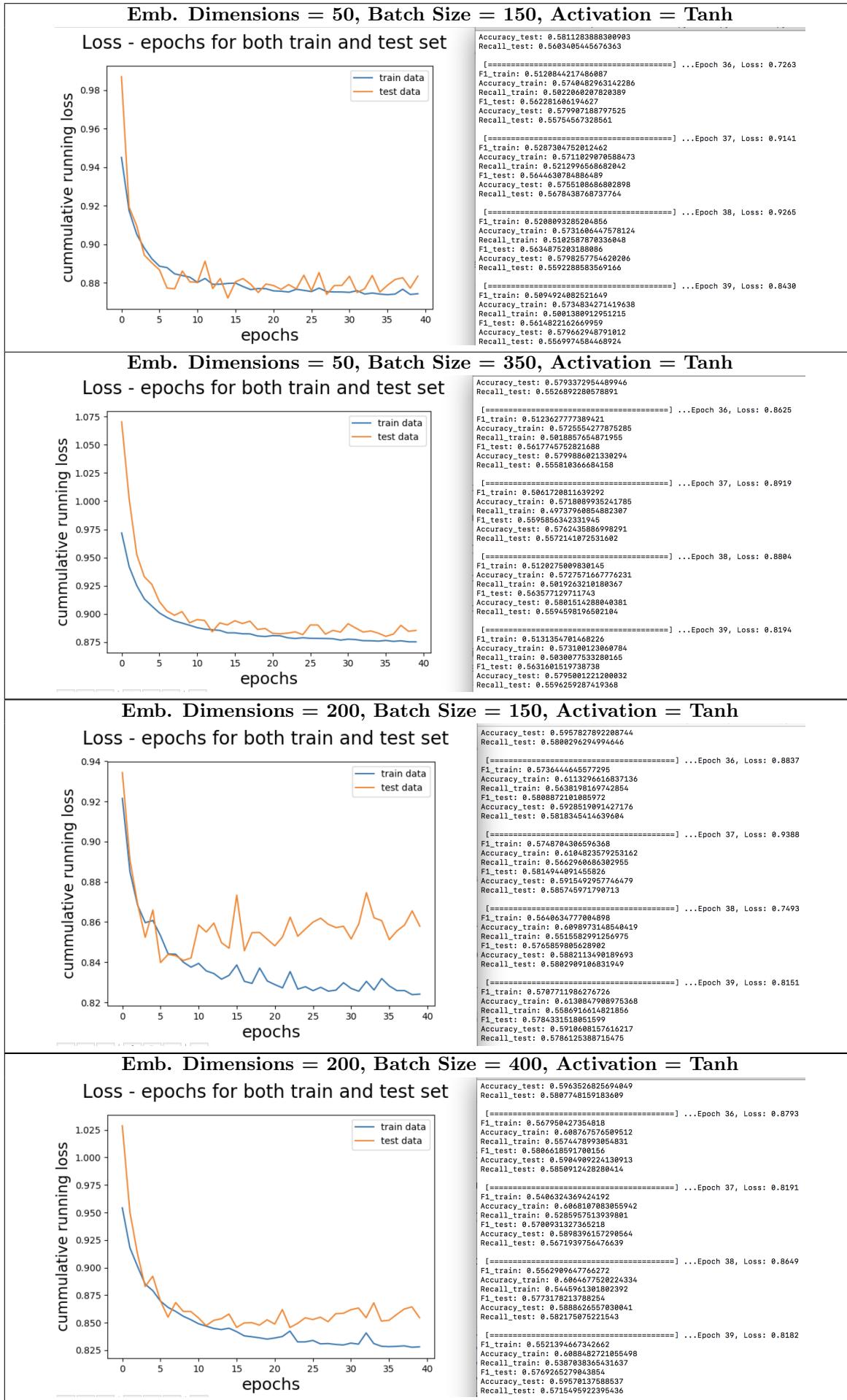
Η `forward()` για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

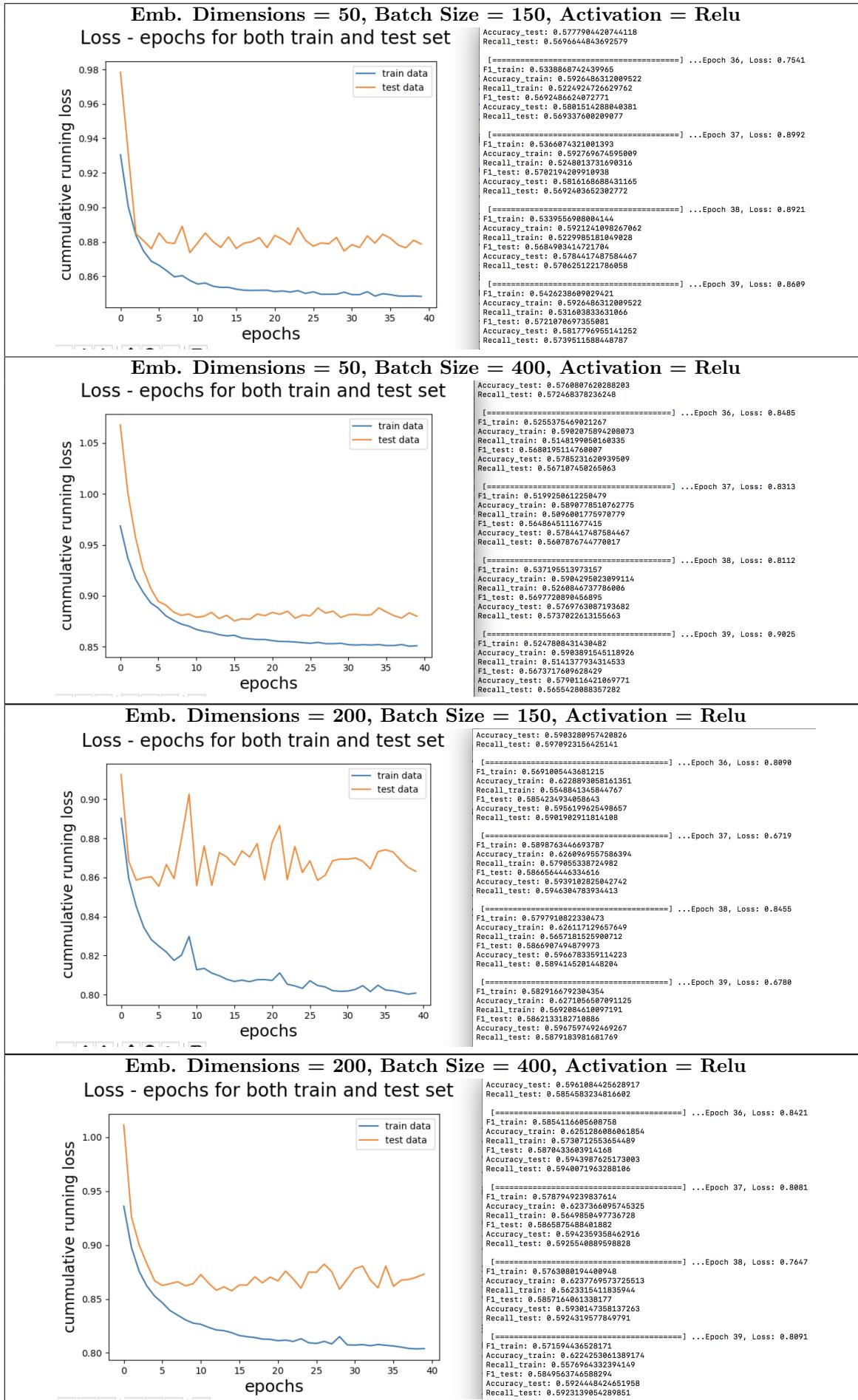
```
embeddings = self.embedding(x)
representations_mean = self.mean_pooling(embeddings, lengths)
representations_max = self.max_pooling(embeddings)
representations = torch.cat((representations_mean, representations_max), 1)
representations = self.non_linearityX(representations)1
logits = self.classifier(representations)
```

Σημειώνεται ακόμα ότι το μέγεθος εισόδου των `representations` που οδηγούνται στο τελευταίο linear layer, διπλασιάζεται άρα γίνεται η κατάλληλη αλλαγή στο `init()`:

```
self.classifier = nn.Linear(in_features = 2*emb_dim, out_features=output_size)
```

Για το αρχείο Semeval2017A - 27B twitter embeddings





1.2 Mean Pooling

Πρακτικά

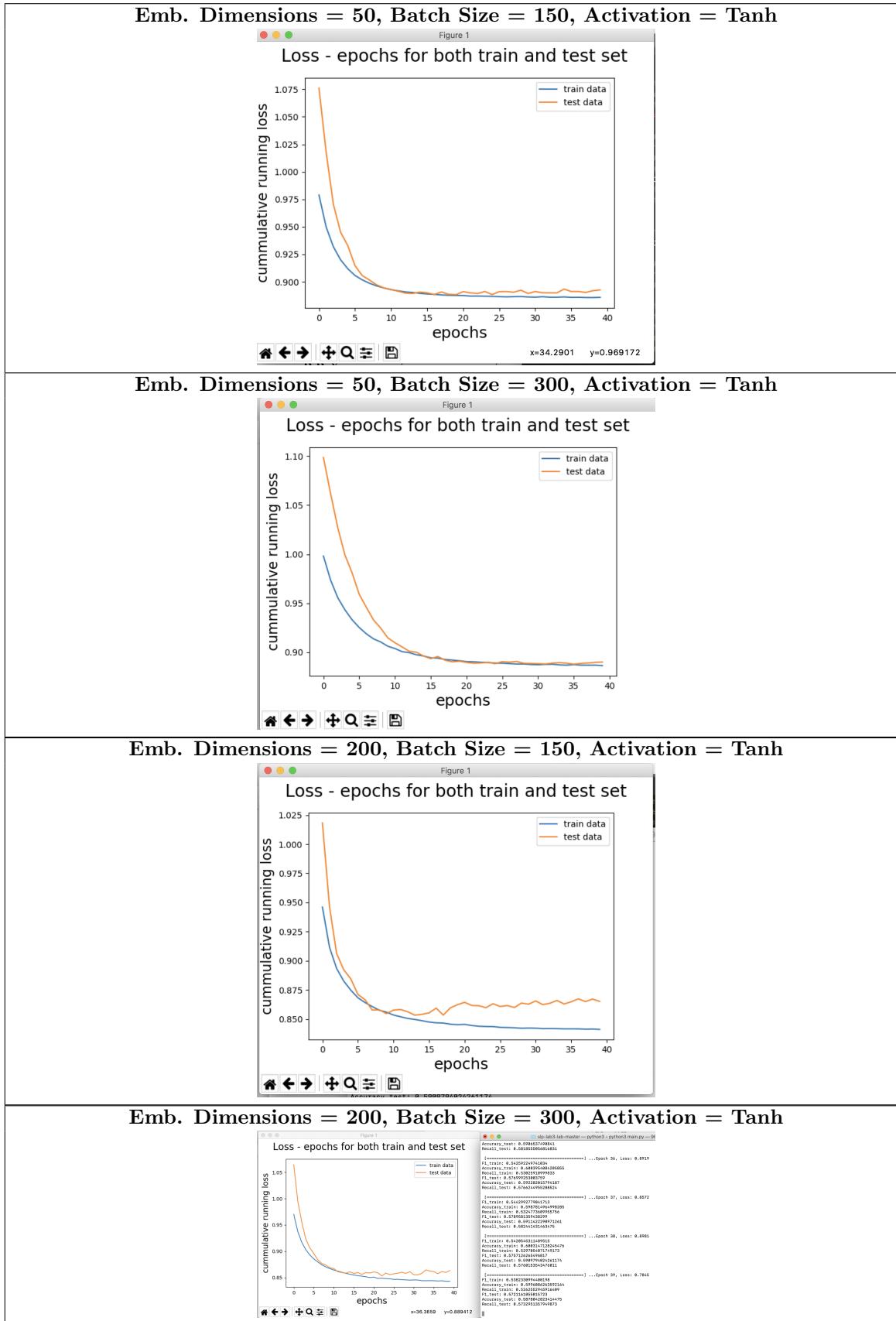
- Εφαρμόζουμε mean pooling και max pooling στα embeddings που έχουμε σχηματίσει και τα αποτελέσματα αυτών τα συνενώνουμε με απότοκο την παραγωγή ενός αποτελεσματικού feature descriptor. Συγχεκριμένα, ενώ στην προπαρασκευή μεταβαίνουμε από embeddings μεγέθους batch_size * seq_length * embedding_dimension σε representations batch_size * embedding_dimension, στο παρόν ερώτημα μεταβαίνουμε σε representations batch_size * (2 * embedding_dimension) που προκύπτει λόγω της παραπάνω συνένωσης.

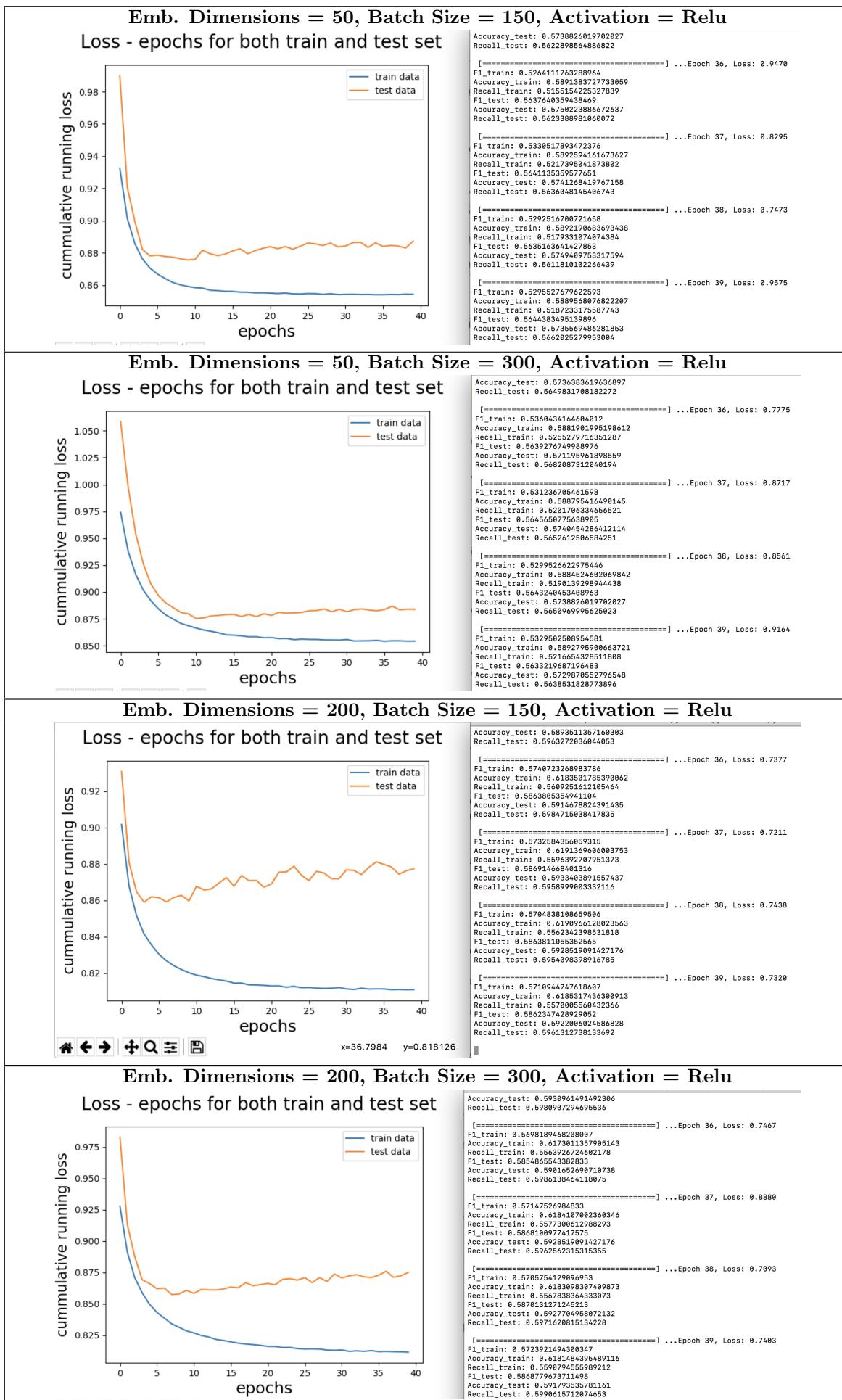
Θεωρητικά

- Γενικότερα η εφαρμογή pooling μεθόδων έχει αποδειχθεί αποτελεσματική στην ανάδειξη των σημαντικών περιοχών πληροφορίας. Το pooling layer σταδιακά μειώνει το spatial μέγεθος των αναπαραστάσεων, μειώνει το μέγεθος των παραμέτρων του μοντέλου και το υπολογιστικό κόστος, ενώ επιτρέπει να έχουμε έλεγχο πάνω στο overfitting. Στην συγκεκριμένη περίπτωση γίνεται εφαρμογή ενός hybrid pooling, που ενώ έχει μεγαλύτερο υπολογιστικό κόστος - από την εφαρμογή ενός απλού pooling -, είναι πολύ πιο ευέλικτο. Το max pooling χρησιμοποιεί την maximum τιμή από ένα cluster από neurons των embeddings, ενώ το average pooling την μέση τιμή αυτών.
- Για να υποστηρίξουμε το επιχειρήμα της παραπάνω πληροφορίας: Για παράδειγμα έστω ότι εξετάζουμε έναν διφυφορικό υπολογιστή. Δεν θέλουμε να ξέρουμε ένα κατά μ.ο τα pre-pooling κελιά έχουν πχ. ένα ποτάμι αλλά εάν κάποιο από αυτά έχουν ποτάμι, άρα αναζητάς το max. Αλλά το αν η εικόνα είναι 'ομιχλώδης', θες να ξέρεις αν είναι 'ομιχλώδης' κατά μ.ο. Επομένως με το να συμπεριλάβουμε και τις δύο μορφές pooling, έχουμε πρόσβαση σε όλο το μήκος κύματος πληροφορίας. Προσθέτοντας και τα δύο, επιτρέπεις στο νευρωνικό δίκτυο να διαλέξει τι δουλεύει καλύτερα χωρίς να χρειάζεται να πειραματιστούμε ξεχωριστά και με τις δύο μεθόδους.

Η `forward()` για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

```
embeddings = self.embedding(x)
representations_mean = self.mean_pooling(embeddings, lengths)
representations = representations_mean
representations = self.non_linearityX(representations)2
logits = self.classifier(representations)
```





Συμπεράσματα

- Σύγκριση των 2 μεθόδων (A) average pooling και (B) συνενωση max και average pooling. Γενικότερα διαπιστώθηκε ότι απ' όλες τις απόψεις η μέθοδος της συνένωσης οδηγεί σε καλύτερα αποτελέσματα.
 - Accuracy: Τάξης 0.01 μεγαλύτερο του (A)
 - Test Loss: Με χρήση `relu()` είναι πολύ πιο εμφανής η τάση του (B) για αύξηση του loss, ενώ οι διακυμάνσεις είναι πολύ μικρότερες σε σχέση με το (A).
 - Train Loss: Τάξης 0.03 μικρότερο με (B)
- Τώρα και για τις δύο μεθόδους³
 - Όσο αυξάνεται το batch size παρατηρούνται λιγότερες διακυμάνσεις στα losses, ενώ accuracy, train loss και test loss παραμένουν ίδια
 - Όσο αυξάνονται τα dimensions των pretrained embeddings που χρησιμοποιούνται έχουμε περισσότερες διακυμάνσεις στα losses, αυξάνεται το accuracy (0.58 σε 0.595, 0.573 σε 0.592), μειώνεται το train loss (0.875 σε 0.82, 0.85 σε 0.8, 0.9 σε 0.85) και μειώνεται και το test loss (0.88 σε 0.86)
 - Η χρήση της `relu()` ως activation function έναντι της `tanh()`, οδηγεί αύξηση του accuracy και μείωση του train loss (0.82 σε 0.8), ενώ το test loss παρουσιάζει τάση αύξησης.

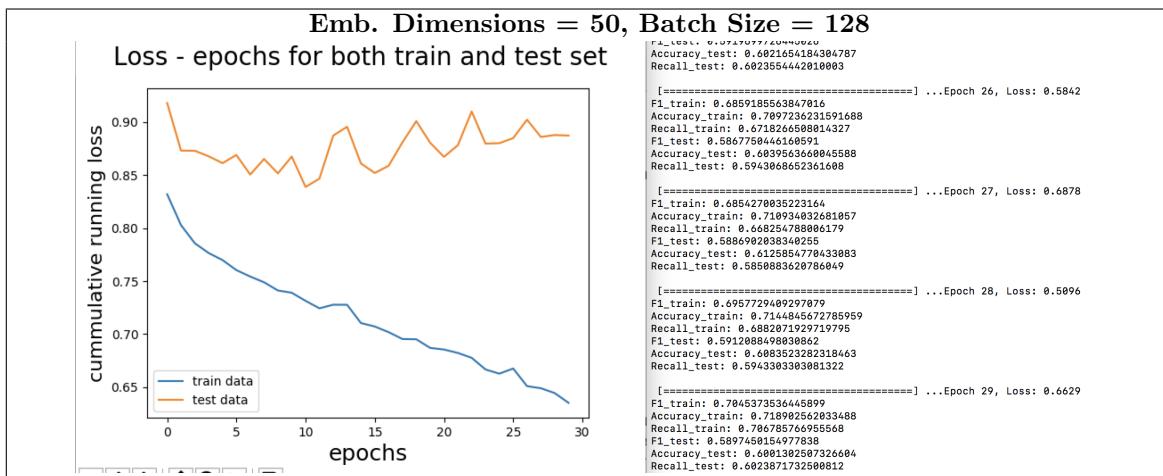
Επομένως, προφανώς τα embeddings των περισσότερων διαστάσεων οδηγούν σε καλύτερα αποτελέσματα εφόσον κάθε λέξη είναι πιο καλά ορισμένη στο context της. Επίσης αρκεί η χρήση ενός decent batch size, (πχ δύναμη του 2) αφού η χρήση μεγαλύτερου απλά προσφέρει λιγότερες διακυμάνσεις. Τέλος, αν κάνουμε train το μοντέλο για λίγες εποχές συνίσταται η χρήση της `relu()`. Επίσης συνίσταται η χρήση της μεθόδου (B) έναντι του (A) καθώς οδηγεί σε καλύτερο accuracy, και μικρότερο train loss.

3 Ερώτημα 2

2.1 LSTM

Η `forward()` για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

```
embeddings = self.embedding(x)
representations_LSTM, (hn, cn) = self.rnn(embeddings)
representations = self.last_timestep(representations_LSTM,lengths)
logits = self.classifier(representations)
```



- **Σχολιασμοί**

- Γενικά τα LSTM είναι ικανά να κάνουν capture τις ακολουθιακές εξαρτήσεις στις προτάσεις και μπορούν να ενισχύσουν τις λέξεις βάσει των εξαρτήσεων αυτών.
- Το training του LSTM προφανώς απαιτεί πολύ περισσότερο χρόνο (2 extra layers από βάρη), ενώ η διαδικασία θα μπορούσε να επιταχυνθεί με χρήση GPU. Τα LSTM δουλεύουν πολύ καλά εάν το πρόβλημα μας απαιτεί μόνο ένα output για κάθε σειρά input, αλλά είναι challenging να τα χρησιμοποιείς όταν έχεις μεγάλες σύνολα λέξεων, ειδικά στην περίπτωση μας που γίνεται χρήση max length έναντι του avg_length που χρησιμοποιήθηκε στην προπαρασκευή.

³Συγκρίνουμε τα αποτέλεσμα που αποφέρει η μεταβολή σε μία παράμετρο διατηρώντας σταθερές τις υπόλοιπες παραμέτρους

- Συμπεράσματα

- Όπως φαίνεται και από το σχήμα η σωστή λειτουργία του LSTM αποφαίνεται πλήρως από την μείωση που παρουσιάζει το train loss, το πόσο δηλαδή μαθαίνει το μοντέλο μας. Στις 30 εποχές το train loss = 0.6 , δηλαδή 0.2 λιγότερο από αυτό που μπορεί να φτάσει το μοντέλο του ερωτήματος 1.
- Αντιθέτως το test loss αρχίζει να αυξάνεται μετά από κάποιο σημείο με αρκετά μεγάλο ρυθμό πράγμα που υποδηλώνει το overfit των δεδομένων. Ισως μπορούσαμε να το καταπολεμήσουμε ως ένα βαθμό με χρήση dropout layer, ώστε να ξεχνάει το νευρωνικό μας αλλά αυτό δεν θα ήταν κατασταλτακός παράγοντας για το πρόβλημα του **overfit**. Η ρίζα του προβλήματος βρίσκεται στον tokenzier που χρησιμοποιούμε, ο οποίος είναι customized, και ο οποίος αναλύει τις προτάσεις σε σύνολο λέξεων και πολλές από αυτές δεν υπάρχουν στο λεξικό μας και ορίζονται ως unk:
- Στο σημείο αυτό αξίζει να σημειωθεί ότι αν γινόταν χρήση του tokenzier της βιβλιοθήκης ekphrasis, τότε θα οδηγούμασταν σε πολύ μικρότερο rate άγνωστων λέξεων και κατ' επέκταση μικρότερο overfit, καθώς ενδείκνυται για tweets, που έχουν πιο informal χαρακτήρα.
- Αξίζει να σημειωθεί ότι έχουμε accuracy = 0.583 περίπου ίδιο με το μοντέλο του προηγουμενου ερωτήματος ενώ το test loss φτάνει κάποια στιγμή και 0.83 τάξης 0.04 μικρότερο από αυτό του προηγούμενου ερωτήματος
- Χρησιμοποιώντας το τελευταίο timestep για κάθε πρόταση και όχι το πραγματικό βάσει μήκους οδήγησε σε χειρότερα αποτελέσματα.

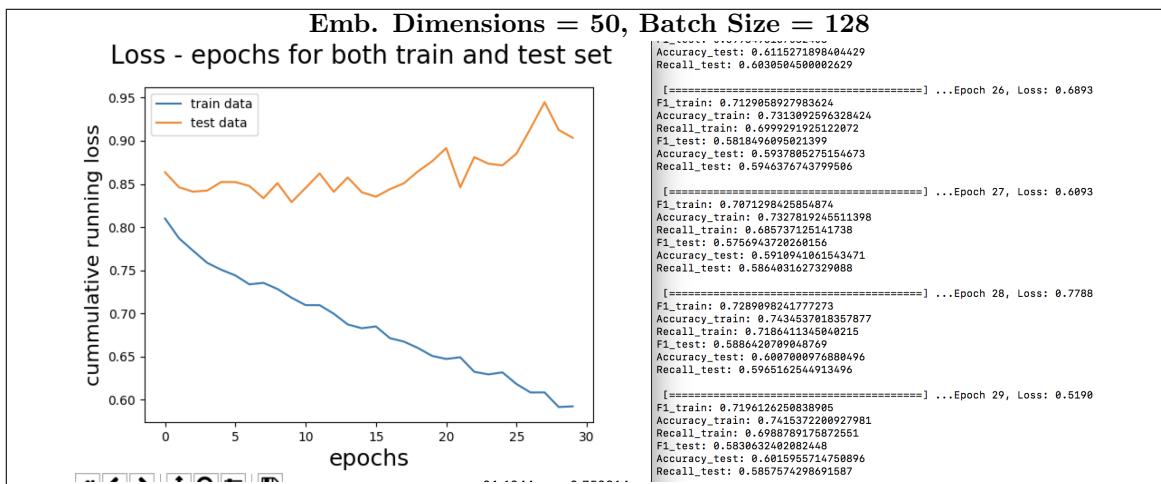
2.2 LSTM - Mean Pooling - Max Pooling

Η `forward()` για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

```
embeddings = self.embedding(x)
representations_LSTM, (hn, cn) = self.rnn(embeddings)
representations_mean = self.mean_pooling(representations_LSTM, lengths)
representations_max = self.max_pooling(representations_LSTM)
representations_LSTM = self.last_timestep(representations_LSTM,lengths)
representations = torch.cat((representations_mean,representations_max,representations_LSTM),1)
logits = self.classifier(representations)
```

Σημειώνεται ωκόμα ότι το μέγεθος εισόδου των representations που οδηγούνται στο τελευταίο linear layer, τριπλασιάζεται μετά την συνένωση άρα γίνεται η κατάλληλη αλλαγή στο `init()`:

```
self.classifier = nn.Linear(in_features = 3*emb_dim, out_features=output_size)
```



- Σχολιασμοί

- Max pooling και average pooling ειναι δύο τεχνικές που βελτιώνουν την απόδοση ενός μοντέλου, εδώ μάλιστα που τις χρησιμοποιούμε στα output του LSTM μαθαίνει πραγματικά να βρίσκει τα πιο salient σημειά μίας πρότασης και δουλεύει καλά όχι μόνο για classification, το οποίο μελετάμε στα πλαίσια αυτού του project, αλλά και ως general encoder.
- Γενικότερα να σημειωθεί ότι το pooling σαν τεχνική χρησιμοποιείται συνδυαστικά με attention layer, για να εμπλουτίσει το expressiveness του attention μηχανισμού.

- Συμπεράσματα

4 Ερώτημα 3

3.1 Attention Embeddings

Η `forward()` για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

```
embeddings = self.embedding(x)
representations, attentions = self.attention(embeddings, lengths)
logits = self.classifier(representations)
```

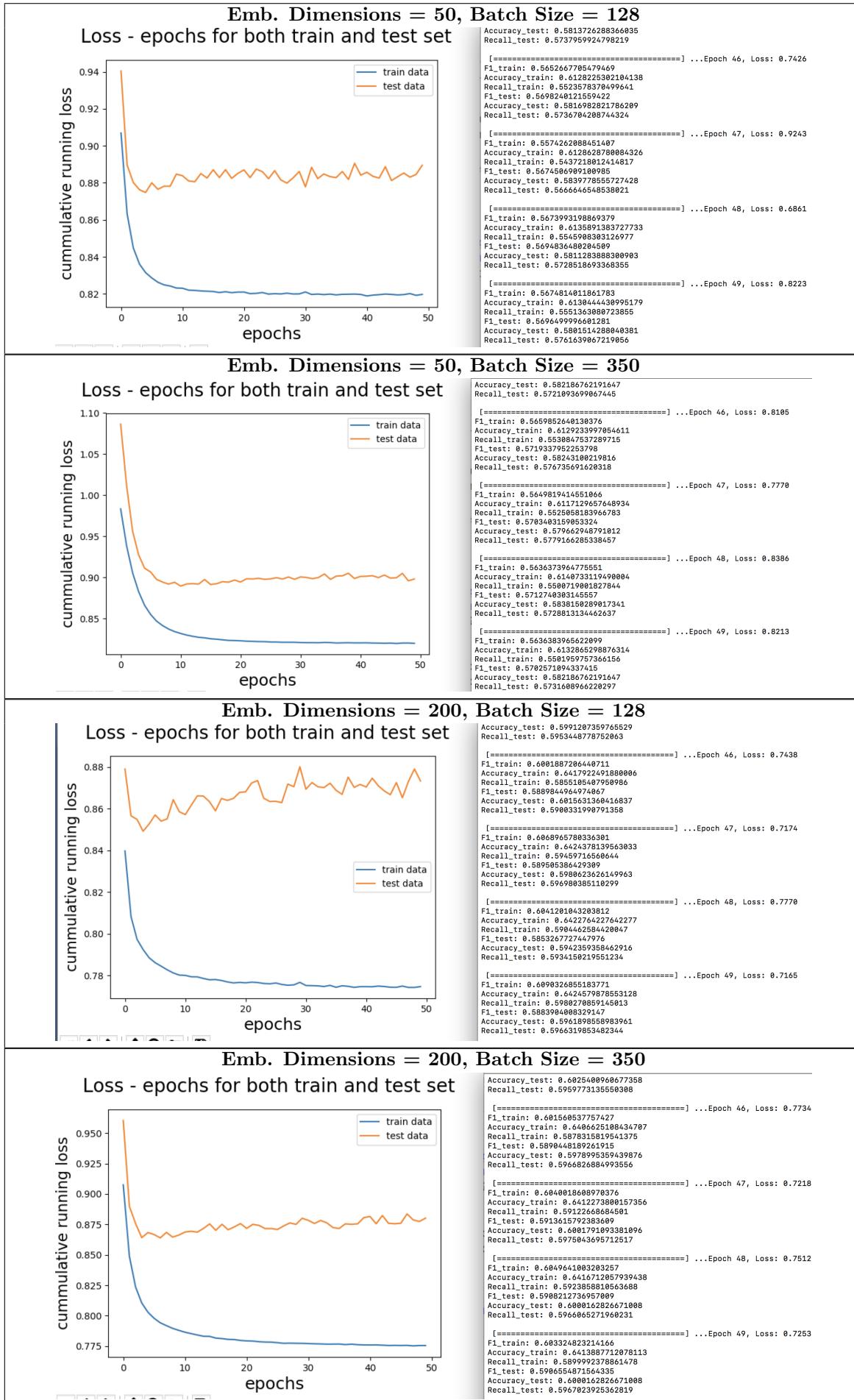
- **Σχολιασμοί**

- Σε κάθε βήμα επιτρέπουμε στο RNN να επιλέγει την πληροφορία που θα κοιτάξει από μια μεγάλη συλλογή πληροφοριών. Χρήση για capture generation ειδικά για CNN είναι πολύ χρήσιμο το attention layer. Ενισχύει την συνεισφορά πιο σημαντικών στοιχείων στην τελική αναπαράσταση αξιοποιώντας όλες τις ενδιάμεσες καταστάσεις ειδικά σε sentiment analysis, οπότε η χρήση του κρίνεται αναγκαία, εδικά εφόσον τα κανονικά RNN τείνουν να είναι "μυωπικά".
- Προφανώς ο χρόνος training θα ναι μεταξύ του χρόνου που απαιτείται για το train του μοντέλου στο ερώτημα 1 και αυτού στο ερώτημα 2, καθώς έχει ένα set βάρη περισσότερο από το πρώτο και ένα λιγότερο από το δεύτερο.

- **Συμπεράσματα**

- Όσο αυξάνονται οι διαστάσεις αυξάνονται και οι διακυμάνσεις των train losses, ενώ παρατηρείται μείωση τους τάξης του 0.025 (0.88 σε 0.86, 0.9 σε 0.87), ενώ το accuracy επίσης αυξάνεται κατά 0.15 (0.58 σε 0.596, 0.582 σε 0.6)
- Όσο αυξάνεται το batch size αυξάνονται τα train losses, κάτι που μέχρι τώρα δε συνέβαινε απλά διατηρούνταν σταθερά.

Επομένως έως τώρα προτιμάται η χρήση του μοντέλου με attention layer συγκριτικά με το lstm, καθώς πετυχαίνει καλύτερο accuracy και όχι overfitting σε λιγότερο training χρόνο από το lstm.



3.2 LSTM Attention

H forward() για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

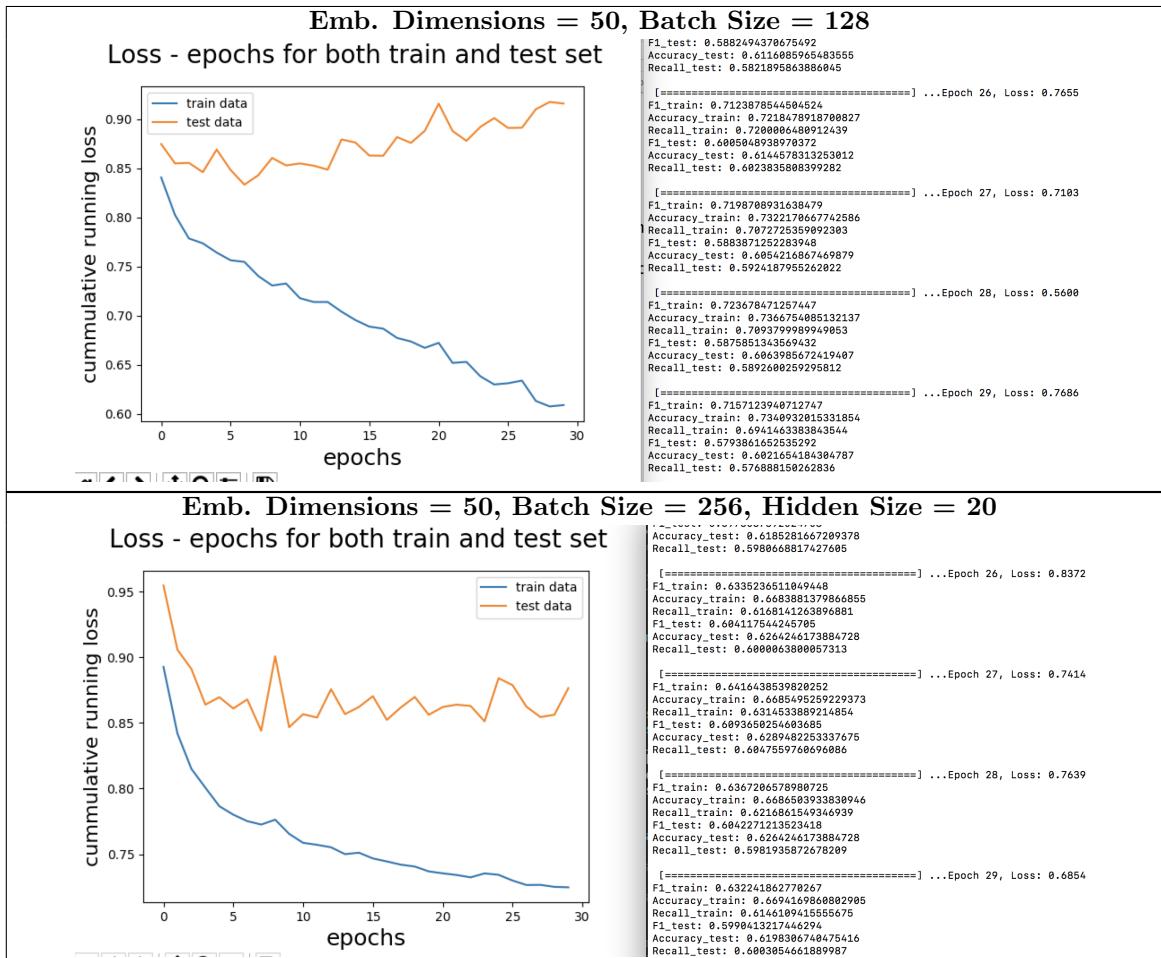
```
embeddings = self.embedding(x)
representations, attentions = self.attention(representations_LSTM, lengths)
logits = self.classifier(representations)
```

- **Σχολιασμοί**

- Αφιερώθηκε ιδιάτερος χρόνος για το συγκεκριμένο μοντέλο, υπάρχουν πολλά papers που επιχρωτούν την χρήση LSTM με attentions layer, οπότε άξιζε η συστηματική μελέτη του.

- **Συμπεράσματα**

- Από το πρώτο διάγραμμα—: Παρατηρείται και εδώ σημαντικό overfitting. Γενικά βγάζουμε τα ίδια συμπεράσματα με 3.2 που έγινε χρήση του lstm μαζί με average και max pooling, απλώς ίσως έχει ελάχιστα χειρότερες μετρικές.
- Από το δεύτερο διάγραμμα: Χρησιμοποιούμε μικρότερο hidden size = 20, μικρότερο δηλαδή από το embeddings dim, ενώ ταυτόχρονα γίνεται εφαρμογή των pretrained vectors 200d. Πράγματι έχουμε λιγότερο overfitting, δεν υπάρχει τάση για αύξηση του train loss, ενώ έχουμε καλύτερο accuracy από την πρώτη περίπτωση τάξης του 0.01 (έχουμε 0.62 περίπου), ενώ τα test loss είναι περίπου ίδια.



5 Ερώτημα 4

4.1 Bi-LSTM Mean Pooling Max Pooling

H forward() για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

```

embeddings = self.embedding(x)
representations_LSTM, (hn, cn) = self.bi_rnn(embeddings)
representations_mean = self.mean_pooling(representations_LSTM, lengths)
representations_max = self.max_pooling(representations_LSTM)
representations_LSTM_fw = representations_LSTM[:, :, :hidden_size]
representations_LSTM_bw = representations_LSTM[:, :, hidden_size:]
representations_LSTM_fw = self.last_timestep(representations_LSTM_fw, lengths)
representations_LSTM_bw = self.last_timestep(representations_LSTM_bw, lengths)
logits = self.classifier(representations)

```

Σημειώνεται ακόμα ότι το μέγεθος εισόδου των representations που οδηγούνται στο τελευταίο linear layer, εξαπλασιάζεται μετά την συνένωση όρα γίνεται η κατάλληλη αλλαγή στο *init()*:

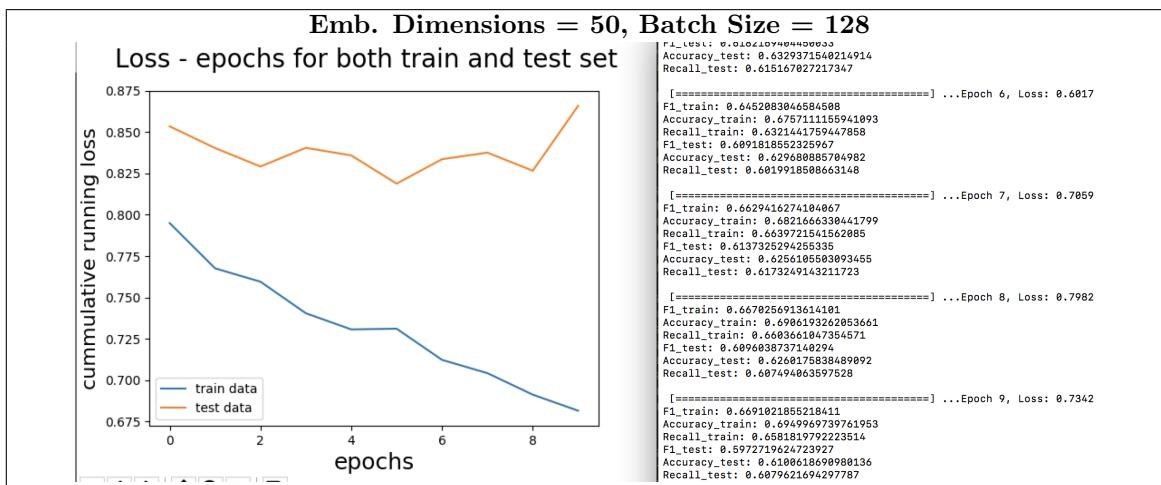
```
self.classifier = nn.Linear(in_features = 6*emb_dim, out_features=output_size)
```

• Σχολιασμοί

- Με την τεχνική αυτή, τον συνδυασμό δύο διαφορετικών RNN που το καθένα επεξεργάζεται με διαφορετική φορά, δημιουργούμε περίληψη του εγγράφου και από τις 2 κατευθύνσεις με σκοπό τον σχηματισμό καλύτερης αναπαράστασης. Στην Pytorch τα hidden state και cell state tensors που επιστρέφονται από τα forward και backward RNNs "στοιβάζονται" το ένα πάνω στο άλλο. Κάνουμε την πρόβλεψη συναισθήματος μας χρησιμοποιώντας το πραγματικό last timestep του hidden state του forward και το πραγματικό last timestep του hidden state του backward.
- Για το συγκεκριμένο μοντέλο χρησιμοποιούμε το **bi_rnn** μοντέλο, το οποίο στην *init()* έχει ως argument **bidirectional = True**. Αμέσως στην *forward()* ή έξοδος του δίνει διπλάσιο μέγεθος **batch_size x seq_len x 2* embedding_dim**. Πάνω στην έξοδο αυτή υπολογίζουμε mean και max pooling. Μετέπειτα, την σπάμε σε forward και backward και με την συνάρτηση *last_timestep()* customized, όπως και στα προηγούμενα ερωτήματα που αφορούν lstm, υπολογίζουμε το πραγματικό last timestep ένα για forward και ένα για backward. Τελικά, τα 4 αυτά τα συνενώνουμε και τα διοχετεύουμε στο τελευταίο linear layer, το οποίο στο ερώτημα αυτό δέχεται είσοδο **6 * embdding_dim**, 6 γιατί τα average και mean είναι διπλάσιου μεγέθους.

• Συμπεράσματα

- Λόγω του μεγάλου πλήθους συνενώσεων, ο χρόνος training του μοντέλου αυξάνεται σημαντικά. Επομένως στα πλαίσια του project το κάναμε train για 10 εποχές για να λάβουμε decent απλά αποτελέσματα και να έχουμε ένα μέτρο σύγκρισης με τα υπόλοιπα μοντέλα. Συγκεκριμένα, το πιο σημαντικό απ'όλα είναι ότι το train loss μειώνεται σημαντικά συγκριτικά με τα υπόλοιπα μοντέλα σε 0.815, αρκετά μικρότερο (0.85 το καλύτερο έως τώρα)
- Επίσης παρατηρήθηκε μια μικρή αύξηση στο accuracy που χτύπησε 0.625



4.2 Bi-LSTM Attention

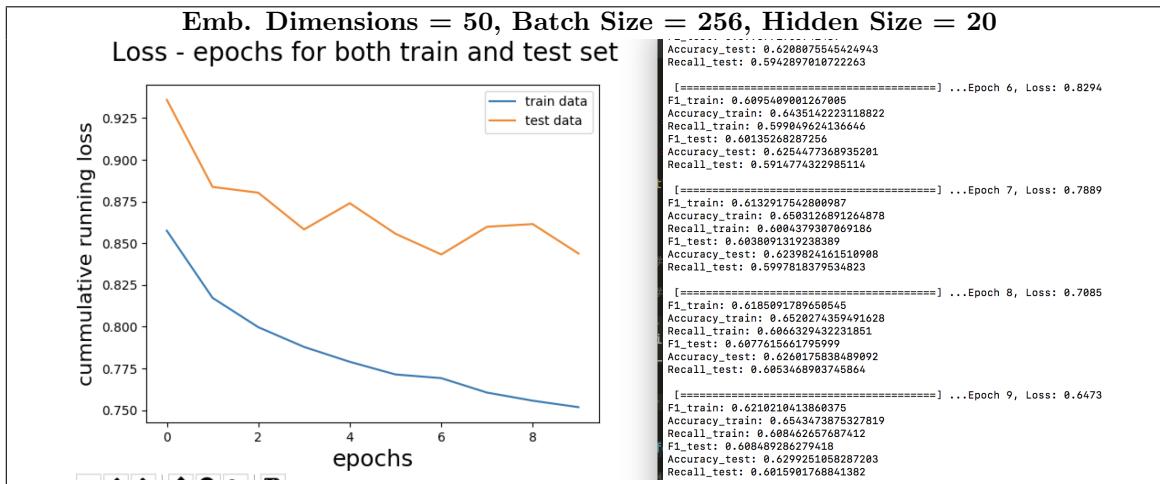
Η *forward()* για την συγκεκριμένη προσέγγιση του μοντέλου έχει υλοποιηθεί ως εξής:

```
embeddings = self.embedding(x)
representations_LSTM, (hn, cn) = self.rnn(embeddings)
representations_LSTM_fw = representations_LSTM[:, :, :hidden_size]
representations_LSTM_bw = representations_LSTM[:, :, hidden_size:]
representations_LSTM_fw, attentions_fw = self.attention(representations_LSTM_fw,
lengths)
representations_LSTM_bw, attentions_bw = self.attention(representations_LSTM_bw,
lengths)
representations =
torch.cat((representations_LSTM_fw, representations_LSTM_bw), 1)
logits = self.classifier(representations)
```

- ### • Σχολιασμοί

• Συμπεράσματα

- Ενώ έχει loss γύρω στο 0.85, έδωσε καλύτερο accuracy από το προηγουμένο μοντέλο της τάξης το 0.05, έδωσε 0.63 συγκεκριμένα. Είμαστε αφετά ευχαριστημένοι παρατηρώντας την πρόοδο των μεθόδων, η εκφύνηση ήταν σαφώς διαμερισμένη και κατατμησμένη ώστε να γίνονται εμφανές οι διαφορές των μοντέλων αν προγραμματιστούν σωστά.
 - Παρατηρείται λιγότερο overfitting συγχριτικά με πριν, μπορεί να οφείλεται στο ότι χρησιμοποιήσαμε hidden size = 20 έναντι με hidden size = 50 και 200 d word embeddings.



6 Ερώτημα 5

5.1 Προβλέψεις σε .txt

Τα βήματα που ακολουθήθηκαν είναι τα εξής:

1. Βρήκαμε ποιο model οδηγεί στο καλύτερο performance από άποψη loss. Αυτό είναι το Bidirectional LSTM του ερωτήματος 4.1.
 2. Ορίσαμε ως MODEL_PATH = '**model.pt**' και θα αποτελεί το checkpoint μας
 3. Στην loop που αφορά το training του μοντέλου μας, για κάθε εποχή τσεκάρουμε αν το παρόν test loss (το χριήριο της εκφώνησης του υποερωτήματος) είναι μικρότερο από τα μέχρι τώρα test losses, και σε περίπτωση που είναι κάνε `torch.save()` το μοντέλο συγκεκριμένα την epoch, το `model_state_dict`, το `optimizer_state_dict` και το test loss
 4. Στην συνέχεια αφού τελειώσει το training, κάνε load το model από το checkpoint και για ακόμα μία φορά κάνε evaluation το test set, για να παράξεις τα αποτελέσματα κυρίως τα attentions και τα predictions
 5. Στην συνέχεια γράφουμε στο αρχείο '**model.txt**' τα interger predicted labels όπως αυτά προκύπτουν από το προηγούμενο βήμα.

5.2 NeAt-vision

Τα βήματα που ακολουθήθηκαν είναι τα εξής:

1. Ορίσαμε 'data.json' κάνοντας κατάλληλη διαχείριση αρχείων και συμβουλευόμενοι την μορφή δειγμάτων που υπήρχε στο NeAt-vision. Στην κατεύθυνση αυτή χρειάστηκε να γίνουν ορισμένες αλλαγές στον κώδικα των αρχείων **models.py**, **main.py**, **training.py** με την έννοια ότι η **forward()** θα επιστρέψει πλέον και τα attentions.
2. Ορίσαμε 'labels.json', βάσει αντιστοιχίσεων μεταξύ indexes και tags όπως αυτά ορίστηκαν στην προπαρασκευή.
3. Παράδειγμα ενός sample από data και του labels:

```

275  {
276      "text": [
277          "SIDE",
278          "TO",
279          "SIDE",
280          "",
281          "arianagrande",
282          "sidetoside",
283          "arianagrande",
284          "musically",
285          "comunidadadgay",
286          "lgbt",
287          "",
288          "LOTB",
289          "https://t.co/tEd8rftAxV"
290      ],
291      "label": 1,
292      "prediction": 1,
293      "attention": [
294          0.07690763,
295          0.07459653,
296          0.073241055,
297          0.07248827,
298          0.07209335,
299          0.071897194,
300          0.07180214,
301          0.07175307,
302          0.07172346,
303          0.07170552,
304          0.07170532,
305          0.07172747,
306          0.06230873,
307          0.06605227,
308          0.0,
309          0.0
310      ],
311      "id": "sample_4"
312  }
313  0.0,
314  0.0,
315  0.0,
316  0.0
317  ],
318  "id": "sample_4"
319  }
320  }
321  }
322  }
323  }
324  }
325  }
326  }
327  }
328  }
329  }
330  }
331  }
332  }
333  }
334  }
335  }
336  }
337  0.0,
338  0.0,
339  0.0,
340  0.0
341  ],
342  "id": "sample_4"
343  }
344  }
345  }
346  }
347  }
348  }
349  }
350  }
351  }
352  }
353  }
354  }
355  }
356  }
357  }
358  }
359  }
360  }
361  }
362  }
363  }
364  }
365  }
366  }
367  }
368  }
369  }
370  }
371  }
372  }
373  }
374  }
375  }
376  }
377  }
378  }
379  }
380  }
381  }
382  }
383  }
384  }
385  }
386  }
387  }
388  }
389  }
390  }
391  }
392  }
393  }
394  }
395  }
396  }
397  }
398  }
399  }
400  }
401  }
402  }
403  }
404  }
405  }
406  }
407  }
408  }
409  }
410  }
411  }
412  }
413  }
414  }
415  }
416  }
417  }
418  }
419  }
420  }
421  }
422  }
423  }
424  }
425  }
426  }
427  }
428  }
429  }
430  }
431  }
432  }
433  }
434  }
435  }
436  }
437  }
438  }
439  }
440  }
441  }
442  }
443  }
444  }
445  }
446  }
447  }
448  }
449  }
450  }
451  }
452  }
453  }
454  }
455  }
456  }
457  }
458  }
459  }
460  }
461  }
462  }
463  }
464  }
465  }
466  }
467  }
468  }
469  }
470  }
471  }
472  }
473  }
474  }
475  }
476  }
477  }
478  }
479  }
480  }
481  }
482  }
483  }
484  }
485  }
486  }
487  }
488  }
489  }
490  }
491  }
492  }
493  }
494  }
495  }
496  }
497  }
498  }
499  }
500  }
501  }
502  }
503  }
504  }
505  }
506  }
507  }
508  }
509  }
510  }
511  }
512  }
513  }
514  }
515  }
516  }
517  }
518  }
519  }
520  }
521  }
522  }
523  }
524  }
525  }
526  }
527  }
528  }
529  }
530  }
531  }
532  }
533  }
534  }
535  }
536  }
537  }
538  }
539  }
540  }
541  }
542  }
543  }
544  }
545  }
546  }
547  }
548  }
549  }
550  }
551  }
552  }
553  }
554  }
555  }
556  }
557  }
558  }
559  }
560  }
561  }
562  }
563  }
564  }
565  }
566  }
567  }
568  }
569  }
570  }
571  }
572  }
573  }
574  }
575  }
576  }
577  }
578  }
579  }
580  }
581  }
582  }
583  }
584  }
585  }
586  }
587  }
588  }
589  }
590  }
591  }
592  }
593  }
594  }
595  }
596  }
597  }
598  }
599  }
600  }
601  }
602  }
603  }
604  }
605  }
606  }
607  }
608  }
609  }
610  }
611  }
612  }
613  }
614  }
615  }
616  }
617  }
618  }
619  }
620  }
621  }
622  }
623  }
624  }
625  }
626  }
627  }
628  }
629  }
630  }
631  }
632  }
633  }
634  }
635  }
636  }
637  }
638  }
639  }
640  }
641  }
642  }
643  }
644  }
645  }
646  }
647  }
648  }
649  }
650  }
651  }
652  }
653  }
654  }
655  }
656  }
657  }
658  }
659  }
660  }
661  }
662  }
663  }
664  }
665  }
666  }
667  }
668  }
669  }
670  }
671  }
672  }
673  }
674  }
675  }
676  }
677  }
678  }
679  }
680  }
681  }
682  }
683  }
684  }
685  }
686  }
687  }
688  }
689  }
690  }
691  }
692  }
693  }
694  }
695  }
696  }
697  }
698  }
699  }
700  }
701  }
702  }
703  }
704  }
705  }
706  }
707  }
708  }
709  }
710  }
711  }
712  }
713  }
714  }
715  }
716  }
717  }
718  }
719  }
720  }
721  }
722  }
723  }
724  }
725  }
726  }
727  }
728  }
729  }
730  }
731  }
732  }
733  }
734  }
735  }
736  }
737  }
738  }
739  }
740  }
741  }
742  }
743  }
744  }
745  }
746  }
747  }
748  }
749  }
750  }
751  }
752  }
753  }
754  }
755  }
756  }
757  }
758  }
759  }
759  }
760  }
761  }
762  }
763  }
764  }
765  }
766  }
767  }
768  }
769  }
770  }
771  }
772  }
773  }
774  }
775  }
776  }
777  }
778  }
779  }
779  }
780  }
781  }
782  }
783  }
784  }
785  }
786  }
787  }
788  }
789  }
789  }
790  }
791  }
792  }
793  }
794  }
795  }
796  }
797  }
798  }
799  }
799  }
800  }
801  }
802  }
803  }
804  }
805  }
806  }
807  }
808  }
809  }
809  }
810  }
811  }
812  }
813  }
814  }
815  }
816  }
817  }
818  }
819  }
819  }
820  }
821  }
822  }
823  }
824  }
825  }
826  }
827  }
828  }
829  }
829  }
830  }
831  }
832  }
833  }
834  }
835  }
836  }
837  }
837  }
838  }
839  }
839  }
840  }
841  }
842  }
843  }
844  }
845  }
846  }
847  }
848  }
849  }
849  }
850  }
851  }
852  }
853  }
854  }
855  }
856  }
857  }
858  }
859  }
859  }
860  }
861  }
862  }
863  }
864  }
865  }
866  }
867  }
868  }
869  }
869  }
870  }
871  }
872  }
873  }
874  }
875  }
876  }
877  }
878  }
879  }
879  }
880  }
881  }
882  }
883  }
884  }
885  }
886  }
887  }
888  }
889  }
889  }
890  }
891  }
892  }
893  }
894  }
895  }
896  }
897  }
898  }
899  }
899  }
900  }
901  }
902  }
903  }
904  }
905  }
906  }
907  }
908  }
909  }
909  }
910  }
911  }
912  }
913  }
914  }
915  }
916  }
917  }
918  }
919  }
919  }
920  }
921  }
922  }
923  }
924  }
925  }
926  }
927  }
928  }
929  }
929  }
930  }
931  }
932  }
933  }
934  }
935  }
936  }
937  }
938  }
939  }
939  }
940  }
941  }
942  }
943  }
944  }
945  }
946  }
947  }
948  }
949  }
949  }
950  }
951  }
952  }
953  }
954  }
955  }
956  }
957  }
958  }
959  }
959  }
960  }
961  }
962  }
963  }
964  }
965  }
966  }
967  }
968  }
969  }
969  }
970  }
971  }
972  }
973  }
974  }
975  }
976  }
977  }
978  }
979  }
979  }
980  }
981  }
982  }
983  }
984  }
985  }
986  }
987  }
988  }
989  }
989  }
990  }
991  }
992  }
993  }
994  }
995  }
996  }
997  }
998  }
999  }
999  }
1000  }
1001  }
1002  }
1003  }
1004  }
1005  }
1006  }
1007  }
1008  }
1009  }
1009  }
1010  }
1011  }
1012  }
1013  }
1014  }
1015  }
1016  }
1017  }
1018  }
1019  }
1019  }
1020  }
1021  }
1022  }
1023  }
1024  }
1025  }
1026  }
1027  }
1028  }
1029  }
1029  }
1030  }
1031  }
1032  }
1033  }
1034  }
1035  }
1036  }
1037  }
1038  }
1039  }
1039  }
1040  }
1041  }
1042  }
1043  }
1044  }
1045  }
1046  }
1047  }
1048  }
1049  }
1049  }
1050  }
1051  }
1052  }
1053  }
1054  }
1055  }
1056  }
1057  }
1058  }
1059  }
1059  }
1060  }
1061  }
1062  }
1063  }
1064  }
1065  }
1066  }
1067  }
1068  }
1069  }
1069  }
1070  }
1071  }
1072  }
1073  }
1074  }
1075  }
1076  }
1077  }
1078  }
1079  }
1079  }
1080  }
1081  }
1082  }
1083  }
1084  }
1085  }
1086  }
1087  }
1088  }
1089  }
1089  }
1090  }
1091  }
1092  }
1093  }
1094  }
1095  }
1096  }
1097  }
1098  }
1098  }
1099  }
1099  }
1100  }
1101  }
1102  }
1103  }
1104  }
1105  }
1106  }
1107  }
1108  }
1109  }
1109  }
1110  }
1111  }
1112  }
1113  }
1114  }
1115  }
1116  }
1117  }
1118  }
1119  }
1119  }
1120  }
1121  }
1122  }
1123  }
1124  }
1125  }
1126  }
1127  }
1128  }
1129  }
1129  }
1130  }
1131  }
1132  }
1133  }
1134  }
1135  }
1136  }
1137  }
1138  }
1138  }
1139  }
1140  }
1141  }
1142  }
1143  }
1144  }
1145  }
1146  }
1147  }
1148  }
1148  }
1149  }
1150  }
1151  }
1152  }
1153  }
1154  }
1155  }
1156  }
1157  }
1158  }
1158  }
1159  }
1160  }
1161  }
1162  }
1163  }
1164  }
1165  }
1166  }
1167  }
1168  }
1169  }
1169  }
1170  }
1171  }
1172  }
1173  }
1174  }
1175  }
1176  }
1177  }
1178  }
1178  }
1179  }
1180  }
1181  }
1182  }
1183  }
1184  }
1185  }
1186  }
1187  }
1187  }
1188  }
1189  }
1190  }
1191  }
1192  }
1193  }
1194  }
1195  }
1196  }
1197  }
1197  }
1198  }
1199  }
1199  }
1200  }
1201  }
1202  }
1203  }
1204  }
1205  }
1206  }
1207  }
1208  }
1209  }
1209  }
1210  }
1211  }
1212  }
1213  }
1214  }
1215  }
1216  }
1217  }
1218  }
1219  }
1219  }
1220  }
1221  }
1222  }
1223  }
1224  }
1225  }
1226  }
1227  }
1228  }
1229  }
1229  }
1230  }
1231  }
1232  }
1233  }
1234  }
1235  }
1236  }
1237  }
1238  }
1238  }
1239  }
1240  }
1241  }
1242  }
1243  }
1244  }
1245  }
1246  }
1247  }
1248  }
1248  }
1249  }
1250  }
1251  }
1252  }
1253  }
1254  }
1255  }
1256  }
1257  }
1258  }
1258  }
1259  }
1260  }
1261  }
1262  }
1263  }
1264  }
1265  }
1266  }
1267  }
1268  }
1268  }
1269  }
1270  }
1271  }
1272  }
1273  }
1274  }
1275  }
1276  }
1277  }
1278  }
1279  }
1279  }
1280  }
1281  }
1282  }
1283  }
1284  }
1285  }
1286  }
1286  }
1287  }
1288  }
1289  }
1289  }
1290  }
1291  }
1292  }
1293  }
1294  }
1295  }
1296  }
1297  }
1297  }
1298  }
1299  }
1299  }
1300  }
1301  }
1302  }
1303  }
1304  }
1305  }
1306  }
1307  }
1308  }
1309  }
1309  }
1310  }
1311  }
1312  }
1313  }
1314  }
1315  }
1316  }
1317  }
1318  }
1319  }
1319  }
1320  }
1321  }
1322  }
1323  }
1324  }
1325  }
1326  }
1327  }
1328  }
1328  }
1329  }
1330  }
1331  }
1332  }
1333  }
1334  }
1335  }
1336  }
1337  }
1337  }
1338  }
1339  }
1339  }
1340  }
1341  }
1342  }
1343  }
1344  }
1345  }
1346  }
1346  }
1347  }
1348  }
1349  }
1349  }
1350  }
1351  }
1352  }
1353  }
1354  }
1355  }
1356  }
1357  }
1358  }
1358  }
1359  }
1360  }
1361  }
1362  }
1363  }
1364  }
1365  }
1366  }
1367  }
1367  }
1368  }
1369  }
1369  }
1370  }
1371  }
1372  }
1373  }
1374  }
1375  }
1375  }
1376  }
1377  }
1377  }
1378  }
1379  }
1379  }
1380  }
1381  }
1382  }
1383  }
1384  }
1385  }
1385  }
1386  }
1387  }
1387  }
1388  }
1389  }
1389  }
1390  }
1391  }
1392  }
1393  }
1394  }
1394  }
1395  }
1396  }
1396  }
1397  }
1398  }
1398  }
1399  }
1399  }
1400  }
1401  }
1402  }
1403  }
1404  }
1405  }
1406  }
1407  }
1408  }
1409  }
1409  }
1410  }
1411  }
1412  }
1413  }
1414  }
1415  }
1416  }
1417  }
1418  }
1418  }
1419  }
1420  }
1421  }
1422  }
1423  }
1424  }
1425  }
1426  }
1427  }
1427  }
1428  }
1429  }
1429  }
1430  }
1431  }
1432  }
1433  }
1434  }
1435  }
1436  }
1436  }
1437  }
1438  }
1438  }
1439  }
1440  }
1441  }
1442  }
1443  }
1444  }
1445  }
1446  }
1446  }
1447  }
1448  }
1448  }
1449  }
1450  }
1451  }
1452  }
1453  }
1454  }
1455  }
1456  }
1456  }
1457  }
1458  }
1458  }
1459  }
1460  }
1461  }
1462  }
1463  }
1464  }
1465  }
1466  }
1466  }
1467  }
1468  }
1468  }
1469  }
1470  }
1471  }
1472  }
1473  }
1474  }
1475  }
1475  }
1476  }
1477  }
1477  }
1478  }
1479  }
1479  }
1480  }
1481  }
1482  }
1483  }
1484  }
1485  }
1485  }
1486  }
1487  }
1487  }
1488  }
1489  }
1489  }
1490  }
1491  }
1492  }
1493  }
1494  }
1495  }
1495  }
1496  }
1497  }
1497  }
1498  }
1499  }
1499  }
1500  }
1501  }
1502  }
1503  }
1504  }
1505  }
1506  }
1507  }
1508  }
1509  }
1509  }
1510  }
1511  }
1512  }
1513  }
1514  }
1515  }
1516  }
1517  }
1518  }
1518  }
1519  }
1520  }
1521  }
1522  }
1523  }
1524  }
1525  }
1526  }
1526  }
1527  }
1528  }
1528  }
1529  }
1530  }
1531  }
1532  }
1533  }
1534  }
1535  }
1536  }
1536  }
1537  }
1538  }
1538  }
1539  }
1540  }
1541  }
1542  }
1543  }
1544  }
1545  }
1546  }
1546  }
1547  }
1548  }
1548  }
1549  }
1550  }
1551  }
1552  }
1553  }
1554  }
1555  }
1556  }
1556  }
1557  }
1558  }
1558  }
1559  }
1560  }
1561  }
1562  }
1563  }
1564  }
1565  }
1566  }
1566  }
1567  }
1568  }
1568  }
1569  }
1570  }
1571  }
1572  }
1573  }
1574  }
1575  }
1575  }
1576  }
1577  }
1577  }
1578  }
1579  }
1579  }
1580  }
1581  }
1582  }
1583  }
1584  }
1585  }
1585  }
1586  }
1587  }
1587  }
1588  }
1589  }
1589  }
1590  }
1591  }
1592  }
1593  }
1594  }
1595  }
1595  }
1596  }
1597  }
1597  }
1598  }
1599  }
1599  }
1600  }
1601  }
1602  }
1603  }
1604  }
1605  }
1606  }
1607  }
1608  }
1609  }
1609  }
1610  }
1611  }
1612  }
1613  }
1614  }
1615  }
1616  }
1617  }
1618  }
1618  }
1619  }
1620  }
1621  }
1622  }
1623  }
1624  }
1625  }
1626  }
1626  }
1627  }
1628  }
1628  }
1629  }
1630  }
1631  }
1632  }
1633  }
1634  }
1635  }
1636  }
1636  }
1637  }
1638  }
1638  }
1639  }
1640  }
1641  }
1642  }
1643  }
1644  }
1645  }
1646  }
1646  }
1647  }
1648  }
1648  }
1649  }
1650  }
1651  }
1652  }
1653  }
1654  }
1655  }
1656  }
1656  }
1657  }
1658  }
1658  }
1659  }
1660  }
1661  }
1662  }
1663  }
1664  }
1665  }
1666  }
1666  }
1667  }
1668  }
1668  }
1669  }
1670  }
1671  }
1672  }
1673  }
1674  }
1675  }
1676  }
1676  }
1677  }
1678  }
1678  }
1679  }
1680  }
1681  }
1682  }
1683  }
1684  }
1685  }
1686  }
1686  }
1687  }
1688  }
1688  }
1689  }
1690  }
1691  }
1692  }
1693  }
1694  }
1695  }
1696  }
1696  }
1697  }
1698  }
1698  }
1699  }
1700  }
1701  }
1702  }
1703  }
1704  }
1705  }
1706  }
1707  }
1708  }
1709  }
1709  }
1710  }
1711  }
1712  }
1713  }
1714  }
1715  }
1716  }
1717  }
1718  }
1718  }
1719  }
1720  }
1721  }
1722  }
1723  }
1724  }
1725  }
1726  }
1726  }
1727  }
1728  }
1728  }
1729  }
1730  }
1731  }
1732  }
1733  }
1734  }
1735  }
1736  }
1736  }
1737  }
1738  }
1738  }
1739  }
1740  }
1741  }
1742  }
1743  }
1744  }
1745  }
1746  }
1746  }
1747  }
1748  }
1748  }
1749  }
1750  }
1751  }
1752  }
1753  }
1754  }
1755  }
1756  }
1756  }
1757  }
1758  }
1758  }
1759  }
1760  }
1761  }
1762  }
1763  }
1764  }
1765  }
1766  }
1766  }
1767  }
1768  }
1768  }
1769  }
1770  }
1771  }
1772  }
1773  }
1774  }
1775  }
1776  }
1776  }
1777  }
1778  }
1778  }
1779  }
1780  }
1781  }
1782  }
1783  }
1784  }
1785  }
1786  }
1786  }
1787  }
1788  }
1788  }
1789  }
1790  }
1791  }
1792  }
1793  }
1794  }
1795  }
1796  }
1796  }
1797  }
1798  }
1798  }
1799  }
1800  }
1801  }
1802  }
1803  }
1804  }
1805  }
1806  }
1807  }
1808  }
1809  }
1809  }
1810  }
1811  }
1812  }
1813  }
1814  }
1815  }
1816  }
1817  }
1818  }
1818  }
1819  }
1820  }
1821  }
1822  }
1823  }
1824  }
1825  }
1826  }
1826  }
1827  }
1828  }
1828  }
1829  }
1830  }
1831  }
1832  }
1833  }
1834  }
1835  }
1836  }
1836  }
1837  }
1838  }
1838  }
1839  }
1840  }
1841  }
1842  }
1843  }
1844  }
1845  }
1846  }
1846  }
1847  }
1848  }
1848  }
1849  }
1850  }
1851  }
1852  }
1853  }
1854  }
1855  }
1856  }
1856  }
1857  }
1858  }
1858  }
1859  }
1860  }
1861  }
1862  }
1863  }
1864  }
1865  }
1866  }
1866  }
1867  }
1868  }
1868  }
1869  }
1870  }
1871  }
1872  }
1873  }
1874  }
1875  }
1876  }
1876  }
1877  }
1878  }
1878  }
1879  }
1880  }
1881  }
1882  }
1883  }
1884  }
1885  }
1886  }
1886  }
1887  }
1888  }
1888  }
1889  }
1890  }
1891  }
1892  }
1893  }
1894  }
1895  }
1896  }
1896  }
1897  }
1898  }
1898  }
1899  }
1900  }
1901  }
1902  }
1903  }
1904  }
1905  }
1906  }
1907  }
1908  }
1909  }
1909  }
1910  }
1911  }
1912  }
1913  }
1914  }
1915  }
1916  }
1917  }
1918  }
1918  }
1919  }
1920  }
1921  }
1922  }
1923  }
1924  }
1925  }
1926  }
1926  }
1927  }
1928  }
1928  }
1929  }
1930  }
1931  }
1932  }
1933  }
1934  }
1935  }
1936  }
1936  }
1937  }
1938  }
1938  }
1939  }
1940  }
1941  }
1942  }
1943  }
1944  }
1945  }
1946  }
1946  }
1947  }
1948  }
1948  }
1949  }
1950  }
1951  }
1952  }
1953  }
1954  }
1955  }
1956  }
1956  }
1957  }
1958  }
1958  }
1959  }
1960  }
1961  }
1962  }
1963  }
1964  }
1965 
```

μικρών διαστάσεων. Ωστόσο το πρόβλημα προκύπτει στις μικρές διστάσεις όταν το πλήθυσμα των δεδομένων δεν είναι αρκετό για να γίνει το feature extraction σωστά και τα word embeddings αποτυνχάνουν σε σχέση με το Bag of Words που δεν επηρεάζεται.

- Άρα όταν θέλουμε να κάνουμε classification σε ένα μικρό πλήθυσμα λέξεων είναι θεμιτό να χρησιμοποιήσουμε ως γνώμονα τις συχνότερα εμφανιζόμενες λέξεις κάτι το οποίο κάνει το BoW

ΠΑΡΑΡΤΗΜΑ

- Έγινε χρήση περισσότερο αγγλικών όρων για την περιγραφή ώστε να μην υπάρχουν αφαιρισμές στα νοήματα.