Τεχνητή Νοημοσύνη 2
Εργασία 3

Ιωάννης Καπετανγεώργης
1115201800061

Περιεχόμενα

Γενικά Σχόλια

- Στο παραδοτέο αρχείο (.ipynb) παρουσιάζονται τα εξής μοντέλα:
    - Το βέλτιστο μοντέλο χρησιμοποιώντας LSTM cells που προέκυψε από την μελέτη του πρώτου μέρους και παρουσιάζεται τελικά στην παράγραφο Βέλτιστο μοντέλο LSTM.
    - Το βέλτιστο μοντέλο χρησιμοποιώντας GRU cells που προέκυψε από την μελέτη του πρώτου μέρους και παρουσιάζεται τελικά στην παράγραφο Βέλτιστο μοντέλο GRU.
    - Το βέλτιστο μοντέλο που προέκυψε προσθέτοντας attention στα δύο παραπάνω μοντέλα και παρουσιάζεται στην παράγραφο Μέρος Β - Προσθήκη Attention στο βέλτιστο μοντέλο.

  Πιο συγκεκριμένα στο παραδοτέο αρχείο υπάρχουν τα εξής 7 εκτελέσιμα «κελιά»:
    1. Γίνονται όλα τα απαραίτητα imports για την εκτέλεση των παρακάτω κελιών.
    2. Ορίζονται τα paths των αρχείων που περιέχουν το train και το validation set.
    3. Download του glove vocabulary για την δημιουργία των embedding. Αρκετά χρονοβόρο λόγω μεγέθους (περίπου 7 λεπτά).
    4. Ορίζονται όλες οι βοηθητικές συναρτήσεις οι οποίες χρησιμοποιούνται για την παραγωγή των μοντέλων. Μεταξύ άλλων οι συναρτήσεις αυτές χρησιμοποιούνται για:
        a. Προ επεξεργασία των δεδομένων.
        b. Ορισμό και δημιουργία των Batchers που περιγράφονται στην παράγραφο Δημιουργία αναπαραστάσεων των tweets - Batching.
        c. Plot των loss vs epochs curves
    5. Η συνολική υλοποίηση του Βέλτιστου μοντέλου LSTM, όπου ακολουθείται η εξής διαδικασία: Προ-επεξεργασία δεδομένων, ορισμός του RNN και των αντίστοιχων υπερπαραμέτρων, εκπαίδευση του μοντέλου και evaluation του μοντέλου.
    6. Η συνολική υλοποίηση του Βέλτιστου μοντέλου GRU, όπου ακολουθείται η αντίστοιχη διαδικασία με το κελί 5.
    7. Η συνολική υλοποίηση του Βέλτιστου μοντέλου με attention, όπου ακολουθείται η αντίστοιχη διαδικασία με τα κελιά 5 και 6 συν τον ορισμό της κλάσης του Attention mechanism.

  Για την εκτέλεση των κελιών 5, 6 και 7 απαιτείται να έχουν εκτελεστεί πρώτα τα κελιά 1, 2, 3 και 4 με αυτήν την σειρά. Τα κελιά 5, 6 και 7 είναι ανεξάρτητα μεταξύ τους και μπορούν να εκτελεστούν με οποιαδήποτε σειρά.
- Η υλοποίηση των νευρωνικών δικτύων των μοντέλων έγινε αποκλειστικά με την χρήση του pytorch.
- Για συνεκτικότητα των αποτελεσμάτων προτείνεται η εκτέλεση χωρίς χρήση της GPU του Colab.
- **ΣΗΜΑΝΤΙΚΟ**: Για την δοκιμή του μοντέλου στο test set χρειάζεται να αλλάξει το path που ορίζεται από την εντολή στο κελί 2:
  ```
  validation_set_location = r'vaccine_validation_set.csv'
  ```
  Έτσι θα αντικατασταθεί το validation set με το test set και κατ' επέκταση ο classifier θα δοκιμαστεί και θα αξιολογηθεί για το test set.
- Στις παραγράφους που ακολουθούν θα περιγράψω αναλυτικά την διαδικασία/μελέτη που ακολούθησα έτσι ώστε να καταλήξω στην τελική μου υλοποίηση.

Μέρος Α - Bidirectional stacked RNN

Στην εργασία αυτή θα κατασκευάσω έναν sentiment classifier χρησιμοποιώντας bidirectional recurrent NN με LSTM/GRU cells για την κατηγοριοποίηση των tweets των datasets που μας δόθηκαν σε τρεις κλάσεις (neutral, anti-vax and pro-vax).
Πιο συγκεκριμένα, θα ακολουθήσει η μελέτη μέσω της οποίας κατέληξα στο τελικό βέλτιστο μοντέλο. Στην μελέτη αυτή πειραματίστηκα με:

- Διάφορες τεχνικές προ επεξεργασίας των δεδομένων
- Τον αριθμό των stacked RNNs
- Το μέγεθος του hidden state του κάθε RNN (hidden_size)
- LSTM και GRU cells
- Dropout μεταξύ των layers
- Gradient clipping
- Skip Connections
- Υπερπαραμέτρους όπως: learning rate, epochs, batch size

Μεθοδολογία

Η μεθοδολογία και τα βήματα που ακολούθησα για την παραγωγή του classifier είναι:

- Ανάγνωση των αρχείων train και validation και αποθήκευση των δεδομένων τους σε δυο dataframes (trainSet και validationSet αντίστοιχα).
- Προεπεξεργασία των δεδομένων τόσο του train set όσο και του validation set. "Καθαρίζουμε" τα δεδομένα έτσι ώστε κάθε tweet να περιέχει μόνο στοιχεία τα οποία μπορούν να προσφέρουν κάποια χρήσιμη πληροφορία.
- Προετοιμασία των δεδομένων έτσι ώστε στην συνέχεια να δημιουργηθούν οι αναπαραστάσεις των προτάσεων μέσω των GloVe pre-trained words embedding vectors. Λεπτομέρειες για την δημιουργία των αναπαραστάσεων εξηγούνται στην αντίστοιχη παράγραφο.
- Ορισμός του νευρωνικού δικτύου, δηλαδή του αριθμού των stacked RNN που θα περιέχει καθώς και τον τύπο αυτών, του αριθμού των νευρώνων από τους οποίος θα αποτελείται το κάθε ένα, καθώς και οποιαδήποτε άλλη από τις τεχνικές που αναφέρθηκαν παραπάνω (π.χ. dropout).
- Ορισμός του loss function που θα χρησιμοποιηθεί καθώς και του optimizer μαζί με τις αντίστοιχες παραμέτρους του. Το loss function και ο optimizer είναι προκαθορισμένα από την εκφώνηση της εργασίας (Cross-Entropy loss και Adam αντίστοιχα), επομένως πειραματιζόμαστε μόνο με παραμέτρους αυτών (π..χ learning rate).
- Εκπαίδευση του μοντέλου για προκαθορισμένο αριθμό epochs. Σε κάθε epoch «δοκιμάζουμε» το μοντέλο μας και για το validation set υπολογίζοντας το validation loss, validation accuracy και validation f1-score τα οποία και εκτυπώνονται για κάθε ένα epoch μαζί με το train loss.
- Δοκιμή του εκπαιδευμένου μοντέλου στο validation set και αξιολόγηση του classification με βάση τις προβλέψεις που έκανε το μοντέλο μας.
  Για την αξιολόγηση χρησιμοποιείται: f1 score, accuracy, precision και recall.
  Επίσης κατασκευάζονται και παρουσιάζονται τα loss vs epoch curves τόσο για το test set όσο και για το validation καθώς και τα ROC curves για κάθε μία κλάση έτσι ώστε να παρακολουθήσουμε την συμπεριφορά του μοντέλου μας κατά την εκπαίδευση και να ανιχνεύσουμε φαινόμενα όπως overfit ή underfit.

Προεπεξεργασία Δεδομένων

Μέσω της προεπεξεργασίας των δεδομένων κειμένου (στην περίπτωση μας tweets) καταφέρνουμε να απαλείψουμε λέξεις/στοιχεία τα οποία δεν προσφέρουν κάποια χρήσιμη πληροφορία.
Έτσι με την κατάλληλη προεπεξεργασία, προκύπτει μια καλύτερη αναπαράσταση η οποία δεν περιέχει άχρηστες πληροφορίες.
Πιο συγκεκριμένα οι τεχνικές που δοκίμασα είναι οι εξής:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions (αναφορά σε άλλους χρήστες του twitter), αφαίρεση των αριθμών και αφαίρεση των hashtags. Για τα παραπάνω χρησιμοποίησα τον tweet-preprocessor ο οποίος έχει κατασκευαστεί για αυτόν ακριβώς τον σκοπό.
- Αφαίρεση των emojis καθώς και αντικατάσταση τους από raw text (π.χ. 😀 → grinning_face)
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση των stop_words.
- Lemmatization μέσω του WordNetLemmatizer
- Stemming μέσω του SnowballStemmer

Δοκίμασα όλες τις παραπάνω τεχνικές σε διαφορετικούς συνδυασμούς. Οι περισσότερες αποδείχθηκαν αποδοτικές σε συνδυασμό με το νευρωνικό δίκτυο.
Στην μελέτη/σύγκριση που ακολουθεί παρουσιάζεται αναλυτικά η απόδοση της κάθε τεχνικής καθώς και το ποιες από αυτές επέλεξα να εφαρμόσω τελικά.

Δημιουργία αναπαραστάσεων των tweets - Batching

Για κάθε ένα tweet πρέπει να δημιουργήσουμε μια αναπαράσταση από αριθμούς η οποία στην συνέχεια θα δοθεί σαν είσοδος στο νευρωνικό δίκτυο.
Για την αναπαράσταση αυτή χρησιμοποίησα pre-trained word embedding vectors του Glove, όπως άλλωστε υποδεικνύεται και από την εκφώνηση της εργασίας.

Αντίστοιχα με την προηγούμενη εργασία, δοκιμάζοντας όλα τα παρεχόμενα αρχεία, παρατήρησα καλύτερα αποτελέσματα με το αρχείο:

- glove.twitter.27B.200d

Το οποίο και θα χρησιμοποιήσω για την δημιουργία των αναπαραστάσεων.

Μέσω της βιβλιοθήκης torchtext παίρνω το glove dictionary του παραπάνω αρχείου. Συγκεκριμένα μέσω της εντολής:

```
glove = torchtext.vocab.GloVe(name="twitter.27B", dim=EMBEDDING_SIZE)
```

(EMBEDDING_SIZE = 200)

Για την δημιουργία των αναπαραστάσεων των tweets θα χρησιμοποιήσω ένα Embedding layer εντός του νευρωνικού. Το embedding layer δημιουργείται από το pretrained μοντέλο του glove (δηλαδή δεν είναι trainable). Πιο συγκεκριμένα, δημιουργώ το embedding layer χρησιμοποιώντας: **nn.Embedding.from_pretrained()**.
Το embedding layer δέχεται σαν input το tweet χωρισμένο σε λέξεις/tokens και παράγει την αναπαράσταση του η οποία προκύπτει από το σύνολο των αναπαραστάσεων για κάθε λέξη/token.

Έτσι, αρχικά (κατά την προεπεξεργασία των δεδομένων) χωρίζω το κάθε tweet σε λέξεις/tokens. Δηλαδή κάθε tweet πλέον αντιστοιχεί σε μία λίστα από tokens.
Στην συνέχεια για βελτιστοποίηση της απόδοσης μετατρέπω την κάθε λέξη στο αντίστοιχο index του glove dictionary. Έτσι πλέον κάθε tweet αντιστοιχεί σε μια λίστα από indexes (αριθμούς) αντί σε μια λίστα από strings.

Έχοντας το index του κάθε token, στο Embedding layer χρησιμοποιώ τα glove vectors, αντί του glove dictionary. Δηλαδή, ορίζω το embedding layer ως εξής:

```
nn.Embedding.from_pretrained(glove.vectors)
```

(όπου το glove ορίστηκε παραπάνω)

Σε αντίθεση με την προηγούμενη εργασία, η αναπαράσταση που θα δημιουργήσουμε για κάθε tweet θα αποτελείται από πολλά vectors. Συγκεκριμένα, ένα για κάθε λέξη.
Έτσι, κάθε tweet θα έχει διαφορετικού μεγέθους αναπαράσταση ανάλογα με τον αριθμό των λέξεων του.

Λόγω αυτού δεν μπορούμε να ακολουθήσουμε την ίδια διαδικασία με την εργασία 2 καθώς οι dataloaders δέχονται δεδομένα ίδιου μεγέθους.

Η αρχική μου προσέγγιση ήταν να εφαρμόσω padding σε όλες τις αναπαραστάσεις χρησιμοποιώντας το μέγεθος του μεγαλύτερου σε μήκος tweet και κάνοντας pad όλα τα υπόλοιπα tweets έτσι ώστε να έχουν το ίδιο μέγεθος.
Για την εφαρμογή του padding χρησιμοποίησα την συνάρτηση pad_sequence του pytorch.
Το RNN αγνοεί τα features τα οποία έχουν προκύψει από το padding, έτσι δεν επηρεάζουν το τελικό αποτέλεσμα.

Ωστόσο, το padding έχει σημαντική επίπτωση στην χρονική απόδοση του μοντέλου καθώς μεγαλώνει το μέγεθος όλων των δεδομένων.

Έτσι, αποφάσισα να ακολουθήσω μια διαφορετική προσέγγιση.
Πιο συγκεκριμένα, δημιούργησα έναν δικό μου DataLoader ο οποίος χωρίζει τα δεδομένα σε batches χωρίς να χρειάζεται η εφαρμογή padding.
Τα tweets ταξινομούνται με βάση το μήκος τους και κάθε batch περιέχει tweets ίδιου μήκους.

Αναλυτικότερα αρχικά δημιουργώ ένα λεξικό το οποίο περιέχει σαν κλειδιά διάφορα μήκη των tweets και σαν values όλα τα tweets τα οποία έχουν το αντίστοιχο μήκος.
Στην συνέχεια, χρησιμοποιώντας αυτό το λεξικό, για κάθε ένα μήκος που περιέχεται στα κλειδιά του λεξικού δημιουργώ έναν DataLoader ο οποίος χωρίζει σε batches όλα τα tweets τα οποία αντιστοιχούν στο συγκεκριμένο μήκος.
Τέλος, ορίζω την συνάρτηση __iter__ έτσι ώστε να κάνω iterable την κλάση μου και να επιστρέφει με τυχαία σειρά τα batches κατά την διάρκεια του training.
Συνοψίζοντας:

- Δημιουργώ έναν DataLoader για κάθε ένα από τα μήκη των tweets που υπάρχουν στο dataset.
- Σε κάθε έναν DataLoader δίνω σαν input όλα τα tweets τα οποία έχουν το αντίστοιχο μήκος, και αυτός τα χωρίζει σε batches με βάση το batch size που έχω ορίσει.
- Μέσω της συνάρτησης __iter__ ο Batcher που κατασκεύασα είναι iterable επιστρέφοντας με τυχαία σειρά batches κατά το training (εντός ενός for loop, όπως ακριβώς χρησιμοποιείται και ο κλασσικός DataLoader)

Χρησιμοποιώντας αυτήν την μέθοδο παρατήρησα βελτίωση στον χρόνο εκπαίδευσης για αυτό και την επέλεξα. Όσον αφορά τα τελικά αποτελέσματα, δεν υπάρχει διαφορά ανεξαρτήτως της μεθόδου που χρησιμοποιείται (padding ή ο Batcher που υλοποίησα).

Συνοψίζοντας όλα τα παραπάνω, η συνολική διαδικασία που ακολουθείται είναι:

- Αρχικά χρησιμοποιώντας την βιβλιοθήκη torchtext, κάνω download το αρχείο και δημιουργώ το embedding dictionary μέσω της εντολής:
  glove = torchtext.vocab.GloVe(name="twitter.27B", dim=EMBEDDING_SIZE)
- Αφού έχει γίνει η κατάλληλη Προεπεξεργασία Δεδομένων, «σπάω» κάθε πρόταση/tweet σε λέξεις/tokens.
- Μετατρέπω τις λέξεις/tokens στα αντίστοιχα indexes του glove. Έτσι κάθε tweet πλέον αντιστοιχεί σε μια λίστα από indexes.
- Μέσω του batcher που υλοποίησα, δημιουργώ batches από τα tweets. Κάθε batch περιέχει tweets (λίστες από indexes) ίδιου μήκους.
- Όταν ένα batch δίνεται σαν είσοδος στο RNN, μέσω του Embedding layer δημιουργείται η αναπαράσταση μέσω glove vectors για κάθε ένα tweet και στην συνέχεια δίνεται σαν input στα layers του νευρωνικού.

Ορισμός του Νευρωνικού Δικτύου

Για την υλοποίηση του νευρωνικού δικτύου χρησιμοποίησα την βιβλιοθήκη pytorch.

Αρχικά ορίζω τα εξής:

- **num_epochs** : ο επιθυμητός αριθμός των epochs για την εκπαίδευση του μοντέλου
- **batch_size**: το μέγεθος των batches που θα δίνονται κάθε φορά σαν input στο μοντέλο κατά την εκπαίδευση.
- **learning_rate**: παράμετρος του optimizer για την ταχύτητα εκμάθισης/προσαρμογής.
- **input_size**: το μέγεθος του κάθε feature που δέχεται σαν είσοδο το πρώτο layer LSTM/GRU. Δηλαδή είναι ίσο με το Embedding size (= 200).
- **hidden_size**: ο αριθμός των features του hidden layer του κάθε LSTM/GRU.
- **num_layers**: ο αριθμός των stacked RNNs (LSTM/GRU).
- **dropout_between_layers**: το ποσοστό του dropout ενδιάμεσα των stacked RNNs
- **final_dropout** : το ποσοστό του dropout στο output του τελευταίου RNN (δηλαδή πριν το γραμμικό output layer).
- **gradient_clipping**: (True/False) ορίζει το αν θα εφαρμοστεί gradient clipping κατά την εκπαίδευση. Η υλοποίηση του gradient clipping είναι αντίστοιχη με αυτήν που παρουσιάστηκε στο φροντιστήριο.
- **skip_connections**: (True/False) ορίζει το αν θα εφαρμοστούν skip connections στο νευρωνικό δίκτυο.
- **model_type**: (LSTM/GRU) ο τύπος του cell που θα χρησιμοποιηθεί.

Στην συνέχεια ορίζω το RNN. Δηλαδή, ορίζω την κλάση RNN η οποία υλοποιεί το νευρωνικό δίκτυο. Η κλάση αυτή είναι υποκλάση της nn.Module (όπου nn: torch.nn). Μέσω της __init__() ορίζω:

- Το Embedding Layer
- Τα stacked bidirectional RNNs που θα χρησιμοποιηθούν
- Τον τύπο αυτών (LSTM/GRU)
- Activation/Normalization functions
- Το τελικό γραμμικό layer έτσι ώστε να παραχθεί το output για κάθε κλάση
- Αρχικοποιώ τα weights των cells. Η αρχικοποίηση των weights γίνεται με βάση των κώδικα που παρουσιάστηκε στο φροντιστήριο της εργασίας.

Αφού οριστεί το RNN, ορίζω:

- Το loss function που θα χρησιμοποιηθεί κατά την εκπαίδευση (Cross Entropy Loss)
- Ο optimizer που θα χρησιμοποιηθεί (Adam) καθώς και οι αντίστοιχες παράμετροι του.

Στην συνέχεια δημιουργούνται οι Batchers τόσο για το train set όσο και για το validation set.

Τέλος εκπαιδεύουμε το RNN.
Ορίζοντας τον επιθυμητό αριθμό epochs, η εκπαίδευση γίνεται επαναληπτικά για κάθε ένα epoch ως εξής:

- Για κάθε ένα batch του train set:
    - Διαγράφουμε τα αποθηκευμένα gradients από το προηγούμενο epoch.
    - Εκτελούμε το μοντέλο με το τρέχον batch, παίρνοντας το αντίστοιχο output για κάθε ένα tweet.

    - Υπολογίζουμε το loss μέσω της loss function που έχει οριστεί.
    - Εφαρμόζεται backpropagation με το loss που υπολογίστηκε.
    - Εφαρμόζεται gradient clipping εφόσον η μεταβλητή gradient_clipping έχει οριστεί σε True.
    - Μέσω του optimizer ανανεώνουμε τα weights του μοντέλου με βάση τα gradients που προέκυψαν από το backpropagation
    - Τέλος, αποθηκεύουμε τα predictions που προέκυψαν έτσι ώστε να αξιολογήσουμε το μοντέλο στο τέλος του epoch
- Αφού ολοκληρωθεί το training για αυτό το epoch αξιολογούμε το μοντέλο χρησιμοποιώντας το validation set. Αρχικά καλούμε την συνάρτηση model.eval() έτσι ώστε να θέσουμε το μοντέλο σε “evaluation mode”. Στην συνέχεια για κάθε ένα batch του validation set:
    - Εκτελούμε το μοντέλο με το τρέχον batch, παίρνοντας το αντίστοιχο output για κάθε ένα tweet.
    - Υπολογίζουμε το loss μέσω της loss function που έχει οριστεί.
    - Αποθηκεύουμε τα predictions που προέκυψαν έτσι ώστε να αξιολογήσουμε το μοντέλο αφού ολοκληρωθεί η διαδικασία για όλα τα batches.
- Τέλος, υπολογίζουμε και εκτυπώνουμε για το τρέχον epoch τα εξής:
    - Train Loss
    - Validation Loss
    - Train F1-Score
    - Validation F1-Score
    - Validation Accuracy

Αφού ολοκληρωθεί η παραπάνω διαδικασία για όλα τα epochs, το μοντέλο έχει εκπαιδευτεί και βρίσκεται σε evaluation mode έτσι ώστε να γίνει το τελικό evaluation του.

Σημαντική Παρατήρηση

Το Loss Function που χρησιμοποιείται είναι το nn.CrossEntropyLoss() καθώς το πρόβλημα μας είναι ένα multiclass classification.
Όπως αναφέρεται στο documentation της nn.CrossEntropyLoss(), η συνάρτηση αυτή συνδιάζει τα nn.LogSoftmax() and nn.NLLLoss().
Αυτό σημαίνει ότι έπειτα από το output layer δεν χρειάζεται να εφαρμοστεί Softmax καθώς αυτό εφαρμόζεται εσωτερικά την nn.CrossEntropyLoss().
Πηγές:

- Documentation nn.CrossEntropyLoss()
- https://stackoverflow.com/questions/55675345/should-i-use-softmax-as-output-when-using-cross-entropy-loss-in-pytorch

https://programmerall.com/article/5058212346/
« So when we use PyTorch to build a classification network, there is no need to manually add a softmax layer after the last fc layer.»

Αξιολόγηση του μοντέλου

Έπειτα από την εκπαίδευση του μοντέλου (και εφόσον το μοντέλο βρίσκεται ήδη σε evaluation mode) εκτελούμε το μοντέλο δίνοντας του σαν input το validation set. Αυτό μας επιστρέφει τις προβλέψεις του για κάθε ένα tweet (pred). Συγκεκριμένα, για κάθε ένα tweet μας επιστρέφει τρείς τιμές οι οποίες αντιστοιχούν στην πιθανότητα να ανήκει το tweet στην αντίστοιχη κλάση.

Για την αξιολόγηση της κατηγοριοποίησης που προέκυψε από το μοντέλο χρησιμοποιώ τις εξής μετρικές:

- F1 score
- Precision
- Recall

Επίσης κατά την εκπαίδευση του μοντέλου υπολογίζω και παρουσιάζω τις καμπύλες Loss vs Epochs τόσο για το train set όσο και για το validation. Πιο συγκεκριμένα, όπως αναφέρθηκε την παράγραφο Ορισμός του Νευρωνικού Δικτύου, για κάθε ένα epoch υπολογίζεται το loss για το train set και για το validation set.
Μετά το πέρας της εκπαίδευσης σχεδιάζονται οι δύο καμπύλες.

Τέλος, παρουσιάζω τα ROC curves για κάθε μία από τις τρείς κλάσεις. Πιο συγκεκριμένα, μέσω των προβλέψεων του μοντέλου (y_pred) υπολογίζω τα ROC curves για κάθε μία κλάση μέσω της συνάρτησης roc_curve της sklearn και στην συνέχεια παρουσιάζω τις τρείς καμπύλες στο ίδιο διάγραμμα.

Μέσω των μετρικών αλλά και των καμπυλών μπορούμε να μελετήσουμε αναλυτικά την απόδοση του μοντέλου αλλά και να εντοπίσουμε προβλήματα όπως overfit και underfit.

Για περαιτέρω μελέτη της συμπεριφοράς/απόδοσης του μοντέλου χρησιμοποιώ:

- Accuracy metric
- Χρησιμοποιώ την συνάρτηση classification_report της sklearn, η οποία παρουσιάζει τα precision, recall και f1-score για κάθε ένα label ξεχωριστά.

Μελέτη για την εύρεση του βέλτιστου μοντέλου

Αρχικά κατασκευάζω ένα πολύ απλό RNN το οποίο αποτελείται από:

- Το Embedding Layer
- 1 LSTM layer με **hidden_size**=32
- Το Linear output layer με input 64 (διπλάσιο του hidden_size λόγω του bidirectional LSTM) και output 3 (ίσο με τον αριθμό των κλάσεων)

Επίσης ορίζω:

- 30 epochs
- **learning_rate** = 0.0001
- **batch_size** = 16
- **dropout** = 0 , δηλαδή δεν εφαρμόζεται καθόλου dropout
- **gradient_clipping** = False, δηλαδή δεν εφαρμόζεται gradient clipping

Το μοντέλο είναι το εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 32, batch_first=True, bidirectional=True)
  (activation): ReLU()
  (dropout): Dropout(p=0, inplace=False)
  (fc): Linear(in_features=64, out_features=3, bias=True)
)
```

Εύρεση βέλτιστης προεπεξεργασίας των δεδομένων

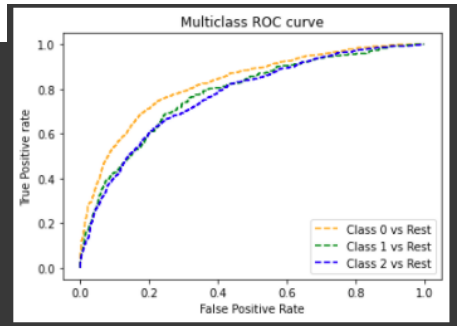
Αρχικά δοκιμάζουμε το παραπάνω μοντέλο χωρίς να εφαρμόσουμε καμία προεπεξεργασία στα δεδομένα.

Τα αποτελέσματα είναι τα εξής:


```
Accuracy: 66.78%
f1 score: 66.42%
Precision: 66.25%
Recall: 66.78%
              precision    recall  f1-score   support

           0       0.74      0.74      0.74      1062
           1       0.44      0.35      0.39       293
           2       0.64      0.68      0.66       918

    accuracy                           0.67      2273
   macro avg       0.61      0.59      0.60      2273
weighted avg       0.66      0.67      0.66      2273
```



Παρατηρούμε σχετικά ικανοποιητικά scores. Ωστόσο μέσω των train vs loss curves παρατηρούμε κακή συμπεριφορά κατά την εκπαίδευση.

Πιο συγκεκριμένα παρατηρούμε ότι:

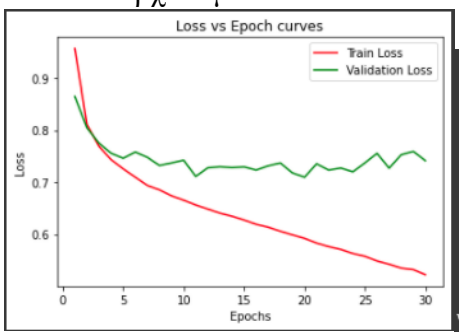
- Το validation loss συνεχώς αυξάνεται αντί να μειώνεται.
- Αντίθετα το train loss μειώνεται, ωστόσο δεν έχει μεγάλη κλίση.
- Τελικά οι δύο καμπύλες έχουν τεράστια απόκλιση, δηλαδή το μοντέλο παρουσιάζει σε πολύ μεγάλο βαθμό overfit.

Στην συνέχεια δοκιμάζω όλες τις τεχνικές προεπεξεργασίας που αναφέρθηκαν στην παράγραφο Προεπεξεργασία Δεδομένων σε διαφορετικούς συνδυασμούς.

Τελικά, την καλύτερη απόδοση πετυχαίνω εφαρμόζοντας την εξής προεπεξεργασία στα δεδομένα:

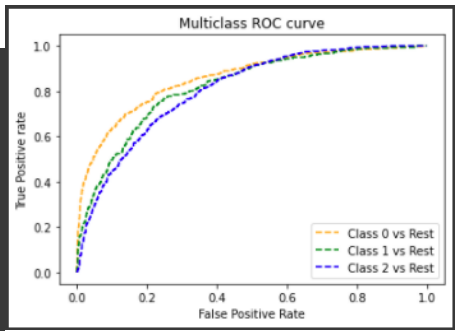
- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face).
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση stopwords.
- Lemmatization.

Εφαρμόζοντας την παραπάνω προεπεξεργασία στα δεδομένα και εκπαιδεύοντας ξανά το αρχικό μοντέλο τα αποτελέσματα είναι τα εξής:



Accuracy: 69.28%
f1 score: 68.81%
Precision: 69.07%
Recall: 69.28%

	precision	recall	f1-score	support
0	0.78	0.74	0.76	1065
1	0.52	0.35	0.42	296
2	0.64	0.74	0.69	921
accuracy			0.69	2282
macro avg	0.65	0.61	0.62	2282
weighted avg	0.69	0.69	0.69	2282



Παρατηρούμε πολύ καλύτερη συμπεριφορά από το μοντέλο. Πιο συγκεκριμένα:

- Πλέον το validation loss δεν αυξάνεται. Αρχικά μειώνεται και στην συνέχεια παραμένει σχεδόν σταθερό.
- Το train loss μειώνεται συνεχώς.
- Τα scores των μετρικών έχουν αυξηθεί, δηλαδή το μοντέλο έχει καλύτερη απόδοση στο validation set.
- Τα ROC curves έχουν καλή συμπεριφορά και για τις τρείς κλάσεις.
- Ωστόσο οι καμπύλες απέχουν αρκετά, δηλαδή εξακολουθεί να παρουσιάζεται overfit.

Πειραματισμός με την δομή των μοντέλων LSTM/GRU

Έχοντας βρει και εφαρμόσει την βέλτιστη προεπεξεργασία στα δεδομένα μας, θα τροποποιήσουμε το μοντέλο μας έτσι ώστε:

- Να εξαλειφθεί το μεγάλο overfit που παρουσιάζει το τρέχον μοντέλο.
- Να αυξηθούν όσο το δυνατόν περισσότερο τα scores των μετρικών.

Όλες τις δοκιμές που παρουσιάζονται στην συνέχεια αλλά και όλες όσες έκανα χωρίς να τις παρουσιάζω εδώ, τις πραγματοποίησα τόσο χρησιμοποιώντας LSTM cells όσο και GRU cells έτσι ώστε να συγκρίνω σε κάθε βήμα την απόδοση τους.

Αρχικά για την βελτίωση της συμπεριφοράς και την μείωση του overfit, δοκιμάζω:

- Να εφαρμόσω **gradient clipping** κατά την εκπαίδευση
- Να εφαρμόσω dropout έπειτα από το LSTM/GRU layer. Θέτω το **dropout** ίσο με 0.5.

Έτσι τα δύο μοντέλα μας είναι τα εξής (LSTM και GRU αντίστοιχα):

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 32, batch_first=True, bidirectional=True)
  (activation): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=64, out_features=3, bias=True)
)
```

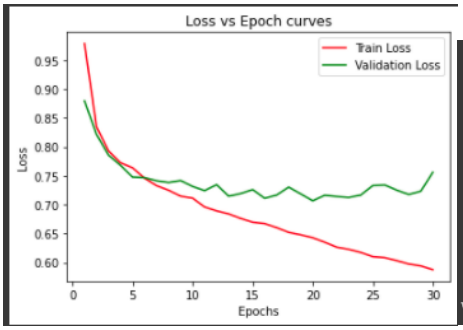
```
RNN(
  (emb): Embedding(1193514, 200)
  (gru): GRU(200, 32, batch_first=True, bidirectional=True)
  (activation): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=64, out_features=3, bias=True)
)
```

Και οι υπόλοιπες παράμετροι:

- 30 epochs
- learning_rate = 0.0001
- batch_size = 16
- final_dropout = 0.5
- gradient_clipping = True

Τα αποτελέσματα είναι τα εξής:

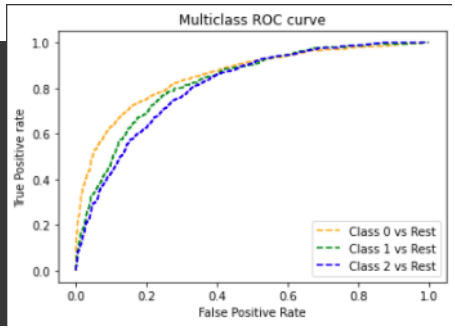
- LSTM



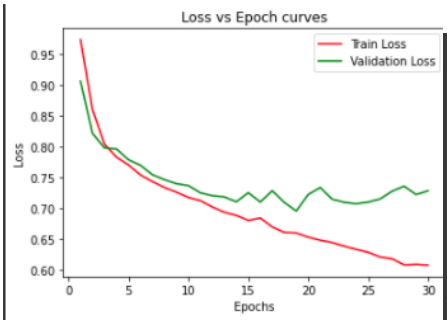

```
Accuracy: 69.54%
f1 score: 68.69%
Precision: 68.66%
Recall: 69.54%
              precision    recall  f1-score   support

           0       0.77      0.77      0.77      1065
           1       0.49      0.29      0.37       296
           2       0.66      0.74      0.69       921

    accuracy                           0.70      2282
   macro avg       0.64      0.60      0.61      2282
weighted avg       0.69      0.70      0.69      2282
```



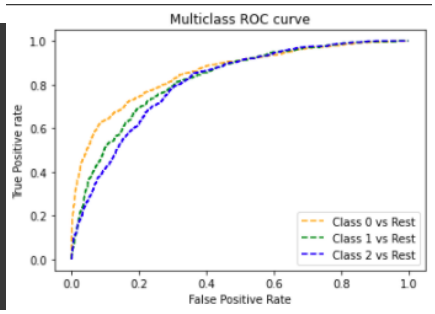
- GRU




```
Accuracy: 68.89%
f1 score: 67.90%
Precision: 67.91%
Recall: 68.89%
              precision    recall  f1-score   support

           0       0.72      0.81      0.76      1065
           1       0.54      0.31      0.40       296
           2       0.67      0.67      0.67       921

    accuracy                           0.69      2282
   macro avg       0.65      0.60      0.61      2282
weighted avg       0.68      0.69      0.68      2282
```



Παρατηρούμε:

- Χρησιμοποιώντας LSTM πετυχαίνουμε καλύτερα αποτελέσματα από ότι με τον GRU κάτι που φαίνεται από τα scores των μετρικών. Η συμπεριφορά του LSTM έχει βελτιωθεί ελάχιστα όσον αφορά τα loss curves.
- Τα ROC curves χρησιμοποιώντας LSTM και GRU είναι παρόμοια. Και στις δύο περιπτώσεις υποδεικνύουν καλή συμπεριφορά των μοντέλων όσον αφορά την αναγνώριση των τριών κλάσεων. Αυτό φαίνεται από το γεγονός ότι πλησιάζουν την πάνω αριστερά γωνία του γραφήματος δηλαδή τα true positives υπερισχύουν και οι καμπύλες έχουν αρκετά υψηλότερες τιμές από την ευθεία y=x (δηλαδή τον random classifier).
- Και τα δύο μοντέλα συνεχίζουν να παρουσιάζουν overfit καθώς οι καμπύλες train και validation loss απέχουν αρκετά μεταξύ τους.

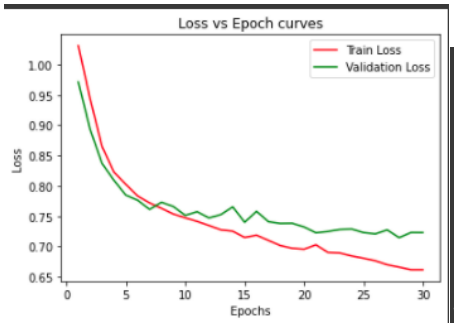
Εφόσον το overfit δεν μειώθηκε χρησιμοποιώντας dropout, δοκιμάζω να απλοποιήσω το μοντέλο.
Μειώνοντας σταδιακά τον αριθμό των features του hidden state του LSTM/GRU layer (δηλαδή την παράμετρο hidden_size), καταλήγω στο μοντέλο με hidden_size=16.
Όλες οι υπόλοιπες παράμετροι παραμένουν ως έχουν.
Έτσι τα δύο μοντέλα είναι τα εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (gru): GRU(200, 16, batch_first=True, bidirectional=True)
  (activation): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 16, batch_first=True, bidirectional=True)
  (activation): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

Τα αποτελέσματα είναι τα εξής:

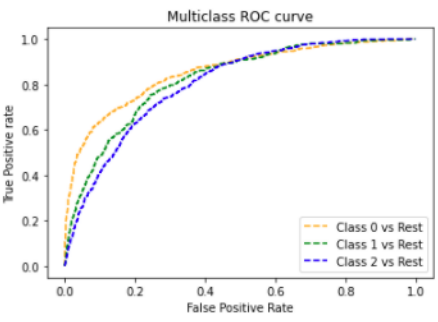
- LSTM



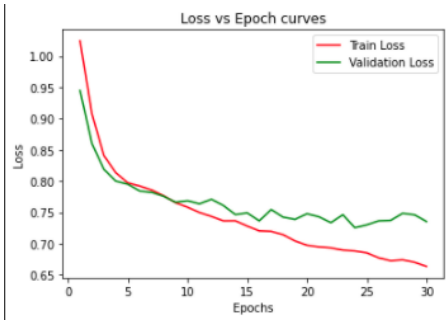

```
Accuracy: 67.75%
f1 score: 66.71%
Precision: 67.37%
Recall: 67.75%
              precision    recall  f1-score   support

           0       0.76      0.73      0.75      1065
           1       0.52      0.26      0.34       296
           2       0.62      0.75      0.68       921

    accuracy                           0.68      2282
   macro avg       0.64      0.58      0.59      2282
weighted avg       0.67      0.68      0.67      2282
```



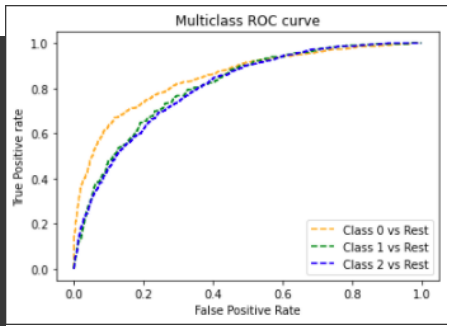
- GRU




```
Accuracy: 69.33%
f1 score: 68.27%
Precision: 68.80%
Recall: 69.33%
              precision    recall  f1-score   support

           0       0.76      0.76      0.76      1065
           1       0.57      0.28      0.37       296
           2       0.64      0.74      0.69       921

    accuracy                           0.69      2282
   macro avg       0.66      0.60      0.61      2282
weighted avg       0.69      0.69      0.68      2282
```



Παρατηρούμε:

- Το overfit έχει μειωθεί σημαντικά καθώς πλέον οι δύο καμπύλες βρίσκονται πολύ πιο κοντά. Ωστόσο δεν έχει εξαλειφθεί πλήρως.
- Χρησιμοποιώντας LSTM τα scores των μετρικών έχουν μειωθεί, δηλαδή έχει πέσει η απόδοση του μοντέλου μας.
- Αντίθετα, χρησιμοποιώντας GRU, η απόδοση έχει αυξηθεί σε σχέση με πριν ενώ ταυτόχρονα μειώθηκε και το overfit. Πλέον το GRU έχει καλύτερη απόδοση από το LSTM.

Με στόχο την πλήρη εξάλειψη του overfit και την βελτίωση της απόδοσης του μοντέλου δοκιμάζω να εφαρμόσω activation functions στο output του LSTM/GRU, δηλαδή πρίν από το Linear output layer.
Δοκίμασα να εφαρμόσω ReLU, Tanh και Sigmoid.
Ωστόσο η απόδοση γινόταν χειρότερη.

Έτσι, στην συνέχεια δοκίμασα να τροποποιήσω την δομή του μοντέλου προσθέτοντας stacked layers LSTM/GRU.
Η προσθήκη των stacked layers γίνεται μέσω της παραμέτρου num_layers των LSTM και GRU, η τιμή της οποίας αντιστοιχεί στον αριθμό των stacked layers.

Δοκιμάζοντας πολλούς συνδυασμούς num_layers και hidden_size (δηλαδή του αριθμού των features του κάθε hidden layer), κατέληξα στον εξής συνδυασμό:

- **num_layers** = 2
- **hidden_size** = 8

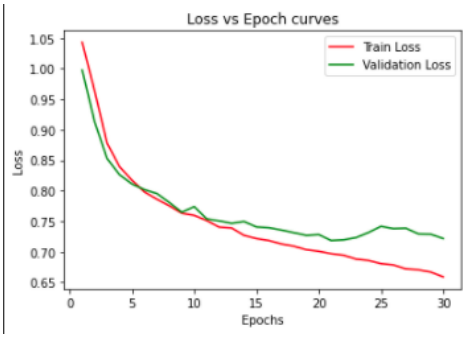
Δηλαδή έχουμε δύο stacked LSTM/GRU layers μεγέθους 8.
Οι υπόλοιπες παράμετροι παραμένουν ως έχει.
Τα δύο μοντέλα είναι τα εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (gru): GRU(200, 8, num_layers=2, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=16, out_features=3, bias=True)
)
```

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 8, num_layers=2, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=16, out_features=3, bias=True)
)
```

Τα αποτελέσματα είναι τα εξής:

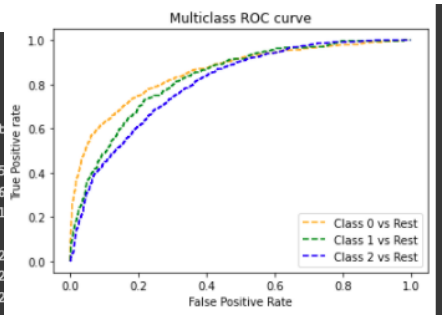
- LSTM



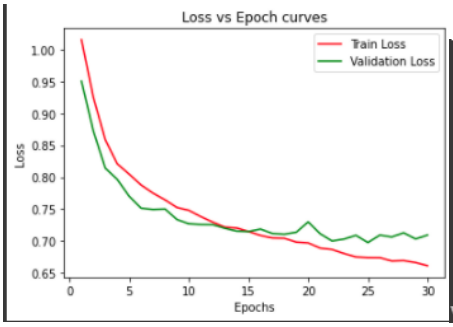

```
Accuracy: 68.36%
f1 score: 67.49%
Precision: 67.75%
Recall: 68.36%
              precision    recall  f1-score   support

           0       0.76      0.76      0.76      1065
           1       0.53      0.29      0.38       296
           2       0.63      0.72      0.67       921

    accuracy                           0.68      2282
   macro avg       0.64      0.59      0.60      2282
weighted avg       0.68      0.68      0.67      2282
```



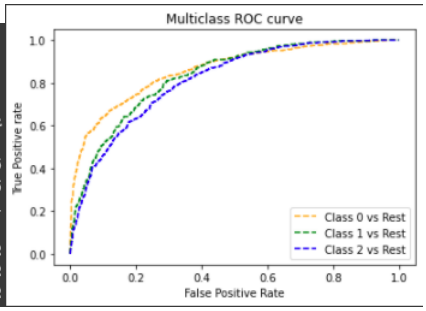
- GRU




```
Accuracy: 68.54%
f1 score: 68.25%
Precision: 68.17%
Recall: 68.54%
              precision    recall  f1-score   support

           0       0.76      0.75      0.76      1065
           1       0.49      0.40      0.44       296
           2       0.65      0.70      0.68       921

    accuracy                           0.69      2282
   macro avg       0.63      0.62      0.62      2282
weighted avg       0.68      0.69      0.68      2282
```



Παρατηρούμε:

- Η απόδοση του LSTM έχει αυξηθεί ξανά. Ωστόσο συνεχίζει να παρουσιάζει ένα μικρό overfit αντίστοιχο με του προηγούμενου μοντέλου. Τόσο τα scores των μετρικών όσο και τα ROC curves είναι αρκετά ικανοποιητικά.
- Όσον αφορά το GRU τα scores των μετρικών καθώς και τα ROC curves είναι αντίστοιχα με προηγουμένως. Ωστόσο έχει μειωθεί το overfit καθώς τα δύο loss curves συγκλίνουν περισσότερο.

Όσον αφορά το GRU δεν κατάφερα να εξαλείψω τελείως το overfit ή να βελτιώσω περισσότερο την απόδοση του.

Όσον αφορά το LSTM, έτσι ώστε να εξαλειφθεί πλήρως το overfit εφαρμόζω τα εξής:

- **dropout_between_layers** = 0.6 , δηλαδή εφαρμόζω dropout μεταξύ των δύο LSTM/GRU layers
- **final_dropout** = 0.5 , ομοίως με προηγουμένως εφαρμόζω dropout στο output του δεύτερου LSTM/GRU layer (δηλαδή πριν το linear output layer)
- **learning_rate** = 0.000125 , αυξάνω ελάχιστα το learning rate

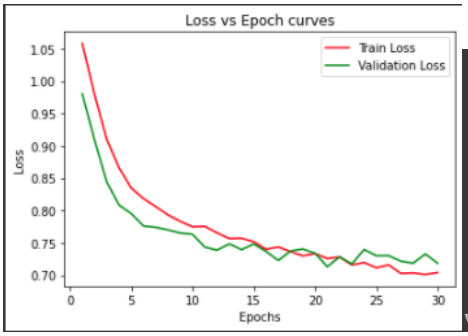
Τα υπόλοιπα παραμένουν ως έχουν.

Επομένως το μοντέλο είναι το εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 8, num_layers=2, batch_first=True, dropout=0.6, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=16, out_features=3, bias=True)
)
```

Τα αποτελέσματα είναι τα εξής:

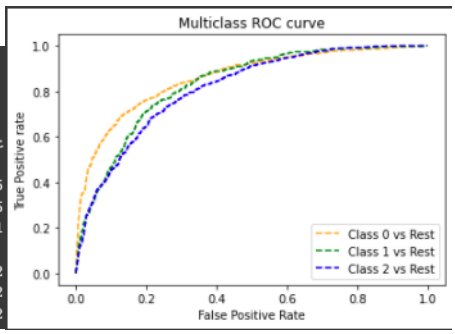
- LSTM




```
Accuracy: 69.37%
f1 score: 68.89%
Precision: 69.12%
Recall: 69.37%
              precision    recall  f1-score   support

           0       0.79      0.74      0.76      1065
           1       0.48      0.34      0.40       296
           2       0.65      0.75      0.70       921

    accuracy                           0.69      2282
   macro avg       0.64      0.61      0.62      2282
weighted avg       0.69      0.69      0.69      2282
```



Παρατηρούμε:

- Η απόδοση του LSTM έχει αυξηθεί ελάχιστα.
- Ταυτόχρονα έχει εξαλειφθεί το overfit καθώς οι δύο καμπύλες train και validation loss τελικά συγκλίνουν και απέχουν ελάχιστα.
- Η καλή συμπεριφορά του μοντέλου επιβεβαιώνεται και από τα ROC curves καθώς για κάθε κλάση τα true positives υπερισχύουν και οι καμπύλες έχουν αρκετά υψηλότερες τιμές από την ευθεία y=x (δηλαδή τον random classifier).

Στην συνέχεια δοκίμασα:

- Να αυξήσω των αριθμό των LSTM/GRU stacked layers για διάφορους συνδυασμούς hidden_size.
- Να εφαρμόσω activation functions ενδιάμεσα των LSTM/GRU layers καθώς και στο output του τελευταίου layer. Η εφαρμογή των activation function ενδιάμεσα από τα LSTM/GRU layers έγινε μέσω της τροποποίησης του μοντέλου που εξηγείται στην παράγραφο Skip Connections καθώς δεν γίνεται να εφαρμοστούν έχοντας ορίσει το stacked LSTM/GRU μέσω της παραμέτρου num_layers.
- Να εφαρμόσω διαφορετικές τιμές για το dropout.

Ωστόσο τίποτα από τα παραπάνω δεν βελτίωσε την απόδοση του μοντέλου.

Skip Connections

Στην συνέχεια, στην προσπάθεια να βελτιώσω το μοντέλο θα εφαρμόσω skip connections στα βέλτιστα μοντέλα που προέκυψαν από την παραπάνω μελέτη χρησιμοποιώντας LSTM και GRU.

Για την εφαρμογή του χρειάζεται να ορίσω με διαφορετικό τρόπο τα stacked LSTM/GRU layers.
Πιο συγκεκριμένα, έχοντας 2 stacked LSTM/GRU layers χρειαζόμαστε το output του πρώτου layer. Προηγουμένως είχα ορίσει stacked LSTM/GRU layers μέσω της παραμέτρου num_layers. Έτσι το output του πρώτου layer δινόταν απευθείας σαν είσοδος στο δεύτερο layer χωρίς να έχουμε την δυνατότητα να το χρησιμοποιήσουμε.

Έτσι ορίζω δύο ξεχωριστά LSTM/GRU layers θέτοντας στο κάθε ένα την παράμετρο num_layers ίση με 1. Το μέγεθος του hidden layer (δηλαδή το hidden_size) παραμένει ίδιο με πριν, δηλαδή 8.
Το πρώτο layer έχει το ίδιο input_size με προηγουμένως δηλαδή ίσο με το Embedding size.
Το δεύτερο layer έχει input size διπλάσιο από το output του πρώτου layer (λόγω του bidirection) δηλαδή 16.
Επίσης ορίζω και ένα dropout layer ενδιάμεσα των δυο LSTM/GRU layers για να εφαρμόσουμε το ενδιάμεσο dropout όπως και προηγουμένως.
Λόγω του συνδυασμού (concatenate) των outputs των δυο LSTM/GRU layers το τελικό output έχει διπλάσια διάσταση σε σχέση με πριν. Έτσι διπλασιάζω το input size του τελευταίου linear output layer.

Τέλος, στην συνάρτηση forward του νευρωνικού κάνω τις εξής αλλαγές:

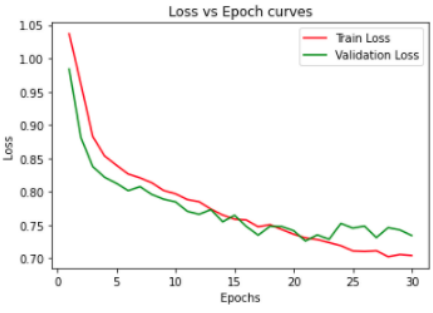
- Έπειτα από την εκτέλεση του πρώτου LSTM/GRU layer αποθηκεύω το output του layer και στην συνέχεια το δίνω σαν input στο δεύτερο layer.
- Έπειτα από την εκτέλεση και του δεύτερου layer αποθηκεύω και το output του δεύτερου layer. Το output του δεύτερου layer είναι ίδιο με το output το Stacked LSTM/GRU που χρησιμοποιούσαμε πριν το skip connection.
- Τέλος, συνδυάζω το output του πρώτου layer με το output του δεύτερου layer κάνοντας τα concatenate δημιουργώντας ένα νέο output μέσω του skip connection.
- Από εκεί και πέρα η διαδικασία συνεχίζεται όπως και προηγουμένως, δηλαδή το νέο output δίνεται σαν είσοδος στο linear output layer.

Όσον αφορά το LSTM, χρησιμοποιώ τις ίδιες παραμέτρους με το βέλτιστο μοντέλο που προέκυψε στην παραπάνω μελέτη.

Δηλαδή το δίκτυο είναι το εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm1): LSTM(200, 8, batch_first=True, bidirectional=True)
  (dropout_between_layers): Dropout(p=0.6, inplace=False)
  (lstm2): LSTM(16, 8, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

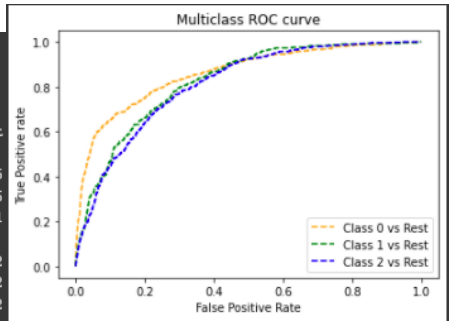
Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 68.23%
f1 score: 67.22%
Precision: 68.11%
Recall: 68.23%
              precision    recall  f1-score   support

           0       0.80      0.72      0.76      1065
           1       0.46      0.24      0.32       296
           2       0.62      0.78      0.69       921

    accuracy                           0.68      2282
   macro avg       0.63      0.58      0.59      2282
weighted avg       0.68      0.68      0.67      2282
```



Παρατηρούμε:

- Η απόδοση του LSTM έχει μειωθεί σημαντικά. Αυτό φαίνεται από τα scores των μετρικών.
- Όσον αφορά τα loss curves, έχουν καλή συμπεριφορά. Ωστόσο έχουν λίγο μεγαλύτερη απόκλιση μεταξύ τους σε σχέση με το βέλτιστο μοντέλο LSTM χωρίς skip connections.

Κάθε προσπάθεια για βελτίωση της απόδοσης του μοντέλου είχε ως αποτέλεσμα σημαντική αύξηση στο overfit. Επομένως το μοντέλο δεν βελτιώνεται περαιτέρω χρησιμοποιώντας skip connections και υστερεί αρκετά σε σχέση με το βέλτιστο μοντέλο που προέκυψε στην παραπάνω μελέτη χωρίς skip connections.

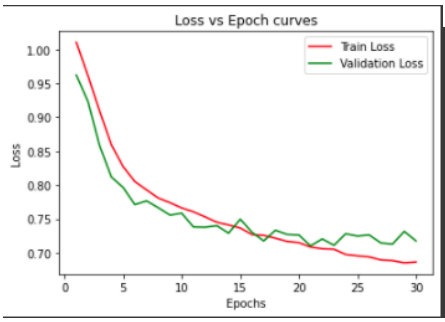
Όσον αφορά το GRU, χρησιμοποιώ τις ίδιες παραμέτρους με το βέλτιστο μοντέλο που προέκυψε στην παραπάνω μελέτη.
Η μόνη διαφορά είναι στα dropout layers όπου ορίζω:

- dropout_between_layers = 0.5
- final_dropout = 0.25

Δηλαδή το δίκτυο είναι το εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (gru1): GRU(200, 8, batch_first=True, bidirectional=True)
  (dropout_between_layers): Dropout(p=0.5, inplace=False)
  (gru2): GRU(16, 8, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.25, inplace=False)
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

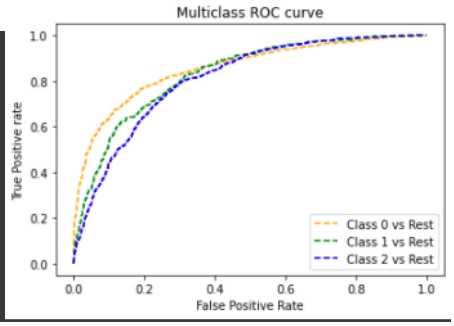
Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 69.72%
f1 score: 69.33%
Precision: 69.58%
Recall: 69.72%
              precision    recall  f1-score   support

           0       0.79      0.75      0.77      1065
           1       0.52      0.37      0.43       296
           2       0.65      0.75      0.69       921

    accuracy                           0.70      2282
   macro avg       0.65      0.62      0.63      2282
weighted avg       0.70      0.70      0.69      2282
```



Παρατηρούμε:

- Σε σχέση με το βέλτιστο μοντέλο χωρίς skip connections, παρατηρούμε αύξηση στα scores των μετρικών. Πιο συγκεκριμένα παρατηρούμε σημαντική αύξηση σε όλα τα metrics f1-score, recall, precision και accuracy.
- Όσον αφορά τα loss curves, καταλήγουν αρκετά κοντά. Δηλαδή το μοντέλο δεν παρουσιάζει overfit.
- Τα ROC curves επαληθεύουν την καλή συμπεριφορά του μοντέλου.
- Άρα το νέο μοντέλο GRU υπερτερεί σε σχέση με το βέλτιστο μοντέλο GRU χωρίς skip connections.

Activation Functions ενδιάμεσα των LSTM/GRU layers

Τέλος, έχοντας ορίσει τα stacked LSTM/GRU layers με αυτόν τον τρόπο μας δίνεται η δυνατότητα να εφαρμόσουμε activation functions ενδιάμεσα των LSTM/GRU layers.
Πιο συγκεκριμένα, πριν εφαρμόσω τα skip connections δοκίμασα να εφαρμόσω activation functions στα βέλτιστα μοντέλα LSTM/GRU που είχαν προκύψει στην μελέτη χωρίς skip connections.
Στην συνέχεια, αφού κατέληξα στα παραπάνω βέλτιστα μοντέλα με skip connections δοκίμασα να εφαρμόσω και σε αυτά activation functions ενδιάμεσα των stacked layers.
Ωστόσο, σε καμία από τις δύο περιπτώσεις δεν βελτίωσαν την απόδοση των μοντέλων.

Βέλτιστο μοντέλο LSTM

Για την παραγωγή του βέλτιστου μοντέλου LSTM αρχικά εφαρμόζουμε την εξής προεπεξεργασία στα δεδομένα:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face).
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση stopwords.
- Lemmatization.

Οι αναπαραστάσεις των tweets δημιουργούνται με τον τρόπο που εξηγήθηκε αναλυτικά στην παράγραφο Δημιουργία αναπαραστάσεων των tweets - Batching.

Το νευρωνικό δίκτυο αποτελείται από:

- Το Embedding Layer
- 2 stacked bidirectional LSTM layers με hidden_size = 8 και dropout = 0.6 μεταξύ των δύο layers
- Ένα dropout layer με dropout = 0.5 το οποίο εφαρμόζεται στο output των stacked LSTM layers.
- Ένα linear output layer έτσι ώστε να παραχθεί το output για τις 3 κλάσεις του προβλήματος.

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 8, num_layers=2, batch_first=True, dropout=0.6, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=16, out_features=3, bias=True)
)
```

Παρατήρηση: Όπως αναφέρθηκε και στην παραπάνω μελέτη τα 2 stacked LSTM layers ορίζονται μέσω της παραμέτρου num_layers καθώς δεν χρειάζεται να κάνουμε τίποτα ενδιάμεσα των δύο layers πλην του dropout το οποίο γίνεται μέσω της αντίστοιχης παραμέτρου. Όλες οι άλλες τεχνικές ενδιάμεσα των layers (π.χ. activation functions, skip connections) δοκιμάστηκαν (και αναφέρθηκαν στην μελέτη) και αποδείχθηκαν μη αποδοτικές.

Συνολικά για να προκύψει το μοντέλο ορίζονται τα εξής:

- **num_epochs** : 30
- **batch_size**: 16
- **learning_rate**: 0.000125
- **input_size**: 200 (= Embedding size)
- **hidden_size**: 8
- **num_layers**: 2
- **dropout_between_layers**: 0.6
- **final_dropout** : 0.5
- **gradient_clipping**: True
- **skip_connections**: False
- **model_type**: LSTM

(Οι παραπάνω μεταβλητές/παράμετροι εξηγούνται στην παράγραφο Ορισμός του Νευρωνικού Δικτύου)

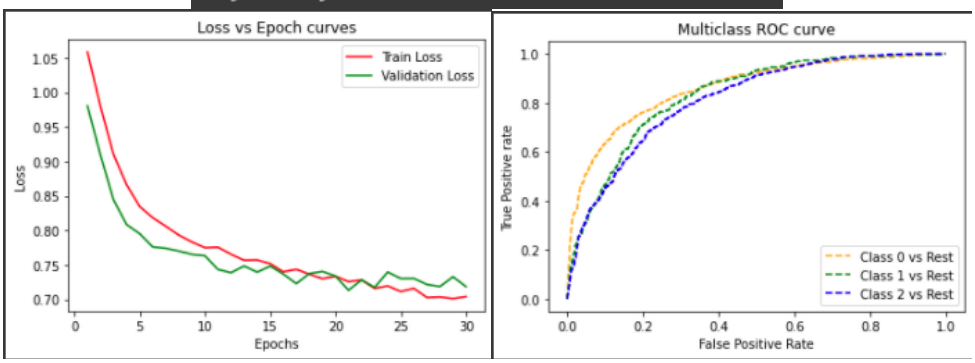
Αξιολόγηση μοντέλου

Έπειτα από την εκπαίδευση του παραπάνω μοντέλου τα αποτελέσματα των μετρικών είναι τα εξής:

```
Accuracy: 69.37%
f1 score: 68.89%
Precision: 69.12%
Recall: 69.37%
              precision    recall  f1-score   support

           0       0.79      0.74      0.76      1065
           1       0.48      0.34      0.40       296
           2       0.65      0.75      0.70       921

    accuracy                           0.69      2282
   macro avg       0.64      0.61      0.62      2282
weighted avg       0.69      0.69      0.69      2282
```



Συμπεράσματα:

- Το μοντέλο μας έχει πολύ καλά scores με βάση τις μετρικές που παρουσιάζονται παραπάνω. Υστερεί στην αναγνώριση των tweets της κλάσης 1 σε σχέση με τις άλλες δύο κλάσεις, κάτι το οποίο οφείλεται στον μικρό όγκο δειγμάτων κλάσης 1 τόσο στο train όσο και στο validation set.
- Το μοντέλο δεν παρουσιάζει overfit κάτι που συμπεραίνουμε τόσο από τα loss curves τα οποία βρίσκονται πολύ κοντά όσο και από τα train/validation f1-scores τα οποία επίσης είναι πολύ κοντά.
- Παρατηρούμε ότι για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο validation set από το train set, δηλαδή παρουσιάζει underfit. Αυτό οφείλεται στην απλότητα του νευρωνικού δικτύου καθώς και στα dropouts που έχουν προτεθεί ενδιάμεσα των layers. Ωστόσο, με την πάροδο των epochs το φαινόμενο του underfit όλο και μειώνεται και τελικά εξαλείφεται. Δηλαδή το μοντέλο μας, μετά το τέλος της εκπαίδευσης δεν παρουσιάζει underfit.
- Από τα ROC curves επαληθεύεται η καλή συμπεριφορά του μοντέλου μας και για τις τρείς κλάσεις. Πιο συγκεκριμένα παρατηρούμε πως τα curves πλησιάζουν προς την «πάνω αριστερά γωνία» δηλαδή τα true positives υπερισχύουν.

Γενικά Συμπεράσματα:

- Αυξάνοντας την πολυπλοκότητα του δικτύου παρατηρήσαμε πως δεν αυξάνεται η απόδοση του μοντέλου. Αντίθετα, χρησιμοποιώντας ένα πιο απλό δίκτυο με 2 μόνο stacked layers με πολύ μικρά hidden sizes και προσαρμόζοντας κατάλληλα τις παραμέτρους πετύχαμε ένα πολύ καλύτερο αποτέλεσμα.

Βέλτιστο μοντέλο GRU

Για την παραγωγή του βέλτιστου μοντέλου GRU αρχικά εφαρμόζουμε την εξής προεπεξεργασία στα δεδομένα:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών.
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face).
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση stopwords.
- Lemmatization.

Οι αναπαραστάσεις των tweets δημιουργούνται με τον τρόπο που εξηγήθηκε αναλυτικά στην παράγραφο Δημιουργία αναπαραστάσεων των tweets - Batching.

Το νευρωνικό δίκτυο αποτελείται από:

- Το Embedding Layer
- 2 stacked bidirectional GRU layers με hidden_size = 8 και dropout = 0.5 μεταξύ των δύο layers
- Ένα dropout layer με dropout = 0.25 το οποίο εφαρμόζεται στο output των stacked GRU layers.
- Ένα linear output layer έτσι ώστε να παραχθεί το output για τις 3 κλάσεις του προβλήματος.

```
RNN(
  (emb): Embedding(1193514, 200)
  (gru1): GRU(200, 8, batch_first=True, bidirectional=True)
  (dropout_between_layers): Dropout(p=0.5, inplace=False)
  (gru2): GRU(16, 8, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.25, inplace=False)
  (fc): Linear(in_features=16, out_features=3, bias=True)
)
```

Στο μοντέλο αυτό εφαρμόζουμε skip connections. Έτσι ο ορισμός των GRU layers γίνεται ορίζοντας δύο ξεχωριστά GRU layers και θέτοντας στο κάθε ένα την παράμετρο n_layers=1.
Επίσης, γίνεται ο κατάλληλος ορισμός του input size στο κάθε layer. Λόγω της ξεχωριστής δήλωσης των layers δεν μπορούμε να εφαρμόσουμε το dropout μεταξύ των δύο layers μέσω της παραμέτρου dropout της κλάσης GRU (όπως κάναμε στο Βέλτιστο μοντέλο LSTM).
Έτσι χρειάζεται να ορίσουμε και ένα dropout layer μεταξύ των δύο GRU layers.
Ο ορισμός του μοντέλου καθώς και η υλοποίηση του Skip Connection εξηγείται αναλυτικά στην παράγραφο Skip Connections.

Συνολικά για να προκύψει το μοντέλο ορίζονται τα εξής:

- **num_epochs** : 30
- **batch_size**: 16
- **learning_rate**: 0.0001
- **input_size**: 200 (= Embedding size)
- **hidden_size**: 8
- **num_layers**: 1
- **dropout_between_layers**: 0.5
- **final_dropout** : 0.25
- **gradient_clipping**: True
- **skip_connections**: True
- **model_type**: GRU

(Οι παραπάνω μεταβλητές/παράμετροι εξηγούνται στην παράγραφο Ορισμός του Νευρωνικού Δικτύου)

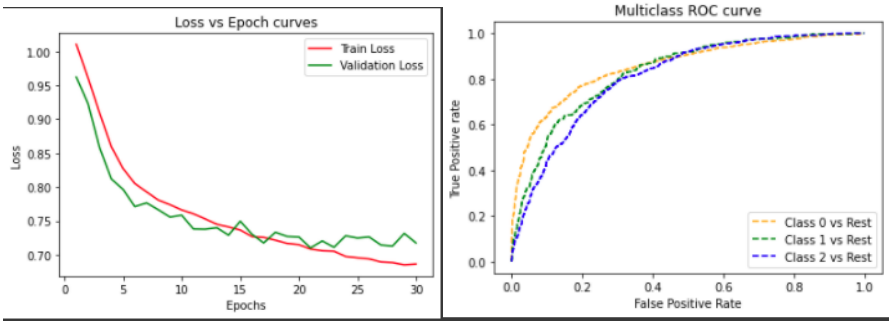
Αξιολόγηση μοντέλου

Έπειτα από την εκπαίδευση του παραπάνω μοντέλου τα αποτελέσματα των μετρικών είναι τα εξής:

```
Accuracy: 69.72%
f1 score: 69.33%
Precision: 69.58%
Recall: 69.72%
              precision    recall  f1-score   support

           0       0.79      0.75      0.77      1065
           1       0.52      0.37      0.43       296
           2       0.65      0.75      0.69       921

    accuracy                           0.70      2282
   macro avg       0.65      0.62      0.63      2282
weighted avg       0.70      0.70      0.69      2282
```



Τα συμπεράσματα που βγάζουμε από τα scores των μετρικών αλλά και από τα curves είναι αντίστοιχα με αυτά του βέλτιστου μοντέλου LSTM, δηλαδή:

- Το μοντέλο μας έχει πολύ καλά scores με βάση τις μετρικές που παρουσιάζονται παραπάνω. Υστερεί στην αναγνώριση των tweets της κλάσης 1 σε σχέση με τις άλλες δύο κλάσεις, κάτι το οποίο οφείλεται στον μικρό όγκο δειγμάτων κλάσης 1 τόσο στο train όσο και στο validation set.
- Το μοντέλο δεν παρουσιάζει overfit κάτι που συμπεραίνουμε τόσο από τα loss curves τα οποία βρίσκονται πολύ κοντά όσο και από τα train/validation f1-scores τα οποία επίσης είναι πολύ κοντά. Τα δύο curves έχουν μια μικρή απόκλιση ωστόσο όχι τόσο μεγάλη ώστε να θεωρηθεί overfit. Επίσης παρατηρούμε πως το train loss συνεχίζει να μειώνεται στα τελευταία epochs και δεν σταθεροποιείται όπως θα θέλαμε ιδανικά. Σταματώντας όμως την εκπαίδευση σε αυτό το σημείο έχουμε την επιθυμητή συμπεριφορά χωρίς το μοντέλο να προλάβει να παρουσιάσει overfit.
- Παρατηρούμε ότι για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο validation set από το train set, δηλαδή παρουσιάζει underfit. Αυτό οφείλεται στην απλότητα του νευρωνικού δικτύου καθώς και στα dropouts που έχουν προτεθεί ενδιάμεσα των layers. Ωστόσο, με την πάροδο των epochs το validation loss μειώνεται με μικρότερο ρυθμό και το φαινόμενο του underfit τελικά εξαλείφεται. Δηλαδή το μοντέλο μας, μετά το τέλος της εκπαίδευσης δεν παρουσιάζει underfit.
- Από τα ROC curves επαληθεύεται η καλή συμπεριφορά του μοντέλου μας και για τις τρείς κλάσεις. Πιο συγκεκριμένα παρατηρούμε πως τα curves πλησιάζουν προς την «πάνω αριστερά γωνία» δηλαδή τα true positives υπερισχύουν.

Γενικά Συμπεράσματα:

- Αυξάνοντας την πολυπλοκότητα του δικτύου παρατηρήσαμε πως δεν αυξάνεται η απόδοση του μοντέλου. Αντίθετα, χρησιμοποιώντας ένα πιο απλό δίκτυο με 2 μόνο stacked layers με πολύ μικρά hidden sizes και προσαρμόζοντας κατάλληλα τις παραμέτρους πετύχαμε ένα πολύ καλύτερο αποτέλεσμα.

Σύγκριση βέλτιστων μοντέλων LSTM και GRU

Στην συνέχεια θα συγκρίνω τα δυο καλύτερα μοντέλα που προέκυψαν χρησιμοποιώντας LSTM και GRU και παρουσιάστηκαν στις παραγράφους Βέλτιστο μοντέλο LSTM και Βέλτιστο μοντέλο GRU αντίστοιχα.

Όσον αφορά τα scores των μετρικών και συγκεκριμένα f1-score, accuracy, recall και precision, παρατηρούμε πως το μοντέλο GRU υπερτερεί σε όλα.
Αυτό σημαίνει ότι καταφέρνει να αναγνωρίσει καλύτερα και να κατηγοριοποιήσει με μεγαλύτερη επιτυχία τα tweets του validation set.
Ωστόσο η διαφορά σε σχέση με το μοντέλο LSTM δεν είναι πολύ μεγάλη.

Όσον αφορά τα loss vs epochs curves παρατηρούμε πως τελικά και στα δύο μοντέλα τα train και validation loss καταλήγουν αρκετά κοντά. Αυτό σημαίνει ότι τα μοντέλο μας είναι εξίσου αποδοτικά τόσο στο σύνολο με το οποίο εκπαιδεύτηκαν όσο και στο validation set. Δηλαδή κανένα από τα δύο μοντέλα δεν παρουσιάζουν overfit.
Ωστόσο παρατηρούμε λίγο καλύτερη συμπεριφορά στο μοντέλο LSTM καθώς τα curves έχουν λίγο μικρότερη απόκλιση σε σχέση με το GRU.

Τέλος, όσον αφορά τα ROC curves παρατηρούμε αντίστοιχη συμπεριφορά και στα δύο μοντέλα. Και στις δύο περιπτώσεις υποδεικνύουν καλή συμπεριφορά των μοντέλων όσον αφορά την αναγνώριση των τριών κλάσεων. Αυτό φαίνεται από το γεγονός ότι πλησιάζουν την πάνω αριστερά γωνία του γραφήματος δηλαδή τα true positives υπερισχύουν και οι καμπύλες έχουν αρκετά υψηλότερες τιμές από την ευθεία y=x (δηλαδή τον random classifier).

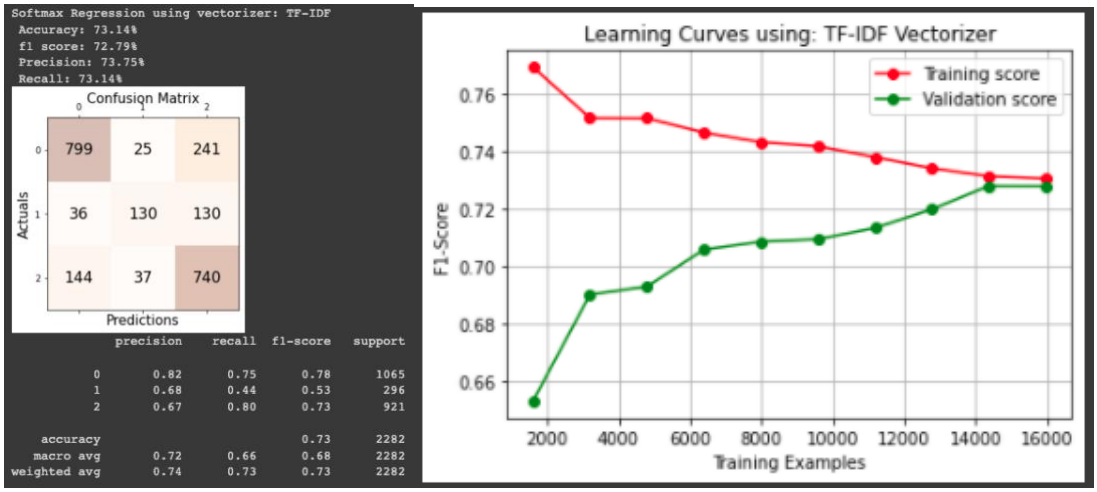
Έτσι καταλήγουμε στο ότι τα δύο μοντέλα έχουν παρόμοια (και αρκετά ικανοποιητική) συμπεριφορά. Το μοντέλο GRU υπερτερεί ελάχιστα σε σχέση με το LSTM.

Σύγκριση με τα μοντέλα των προηγούμενων εργασιών

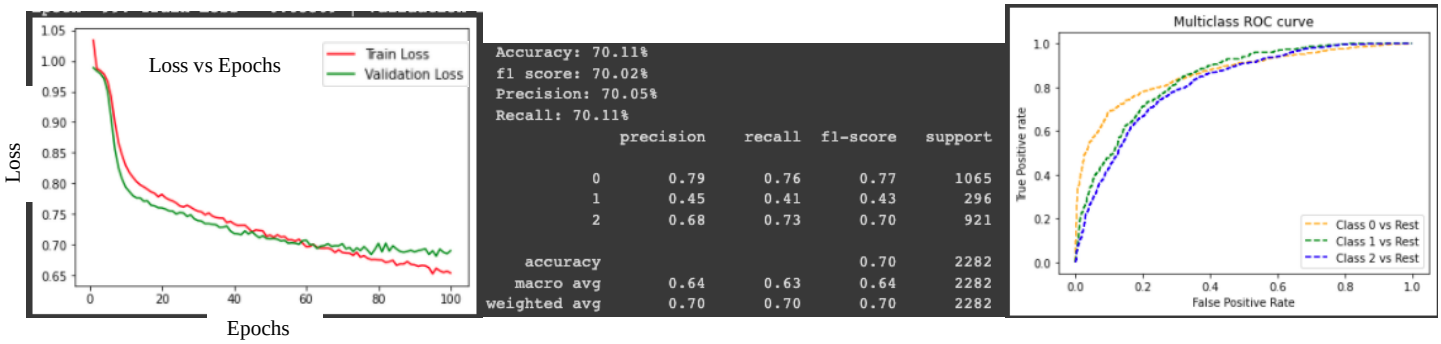
Θα συγκρίνω το βέλτιστο μοντέλο που προέκυψε από την παραπάνω μελέτη (δηλαδή το Βέλτιστο μοντέλο GRU) με τα βέλτιστα μοντέλα των εργασιών 1 και 2.

Αρχικά παραθέτω τα αποτελέσματα των βέλτιστων μοντέλων των πρώτων δύο εργασιών.

- Εργασία 1



- Εργασία 2



Συγκρίνοντας το Βέλτιστο μοντέλο GRU με το μοντέλο της εργασίας 1 παρατηρούμε:

- Το μοντέλο της εργασίας 1 (Softmax) υπερισχύει σε σχέση με αυτό την εργασίας 3 τόσο στα συνολικά f1-score/precision/recall όσο και στα επιμέρους scores ανά κλάση. Δηλαδή, το μοντέλο της πρώτης εργασίας αναγνωρίζει καλύτερα τα tweets του validation set και τα κατηγοριοποιεί με μεγαλύτερη επιτυχία σε κάθε περίπτωση.
- Και τα δύο μοντέλα έχουν ομαλή συμπεριφορά και δεν παρουσιάζουν overfit ή underfit. Πιο συγκεκριμένα, στο μοντέλο της εργασίας 1 παρατηρούμε ότι τα train/validation f1-score curves συγκλίνουν και τελικά ταυτίζονται, δηλαδή δεν παρουσιάζεται καθόλου overfit. Αντίστοιχα στο μοντέλο της εργασίας 3 τα train/validation loss curves ακολουθούν παρόμοια πορεία και τελικά καταλήγουν αρκετά κοντά, με μια μικρή απόκλιση η οποία όμως δεν υποδηλώνει overfit.

Από τα παραπάνω συμπεραίνουμε πως το μοντέλο της πρώτης εργασίας (Softmax regression) είναι αποδοτικότερο από αυτό της τρίτης εργασίας (bidirectional stacked RNN).

Συγκρίνοντας το Βέλτιστο μοντέλο GRU με το μοντέλο της εργασίας 2 παρατηρούμε:

- Το μοντέλο της εργασίας 2 (NN) υπερισχύει σε σχέση με αυτό την εργασίας 3 τόσο στα συνολικά f1-score/precision/recall όσο και στα επιμέρους scores ανά κλάση. Ωστόσο η διαφορά είναι πολύ μικρή (μικρότερη του 1%) σε όλες τις μετρικές. Επομένως μπορούμε να θεωρήσουμε πως τα δύο μοντέλα έχουν αντίστοιχη απόδοση.
- Όσον αφορά τα train/validation loss curves, παρατηρούμε αντίστοιχη εικόνα και στα δύο μοντέλα. Πιο συγκεκριμένα τα train/validation loss curves ακολουθούν παρόμοια πορεία και τελικά καταλήγουν αρκετά κοντά, με μια μικρή απόκλιση η οποία όμως δεν υποδηλώνει overfit.
- Τέλος, λαμβάνοντας υπόψιν τα εξής:
    - Και στα δύο μοντέλα έχει οριστεί το ίδιο batch size (= 16) κατά την εκπαίδευση και προφανώς το train set είναι ίδιο.
    - Στο μοντέλο της εργασίας 2 έχει οριστεί αρκετά μεγαλύτερο learning rate σε σχέση με το μοντέλο της εργασίας 3. Πιο συγκεκριμένα έχουν οριστεί learning_rate=0.0025 και learning_rate=0.0001 αντίστοιχα.
    - Το μοντέλο της εργασίας 2 χρειάζεται 100 epochs για να εκπαιδευτεί ενώ το μοντέλο της εργασίας 3 χρειάζεται μόλις 30 epochs.
    - Τα δύο μοντέλα έχουν τελικά παρόμοια απόδοση και συμπεριφορά.

  Συμπεραίνουμε πως το μοντέλο της εργασίας 3 εκπαιδεύεται πολύ πιο εύκολα σε σχέση με αυτό της εργασίας 2.

Από τις δύο παραπάνω συγκρίσεις συμπεραίνουμε πως το μοντέλο της πρώτης εργασίας είναι το αποδοτικότερο από τα τρία μοντέλα για το συγκεκριμένο πρόβλημα.
Αυτό οφείλεται στην απλότητα του προβλήματος και του μικρού όγκου των δεδομένων μας. Ως γνωστών τα νευρωνικά δίκτυα (Εργασία 2) και πόσο μάλλον τα bidirectional stacked recursive νευρωνικά δίκτυα (Εργασία 3) είναι πολύ αποδοτικά για δύσκολα προβλήματα και απαιτούν μεγάλο όγκο δεδομένων για να εκπαιδευτούν. Αυτό αποδεικνύεται και στην πράξη μέσω της μελέτης μας καθώς έχοντας το σχετικά μικρό dataset εκπαίδευσης που μας δόθηκε, τα νευρωνικά δίκτυα δεν βελτίωσαν καθόλου την απόδοση σε σχέση με το «απλό» μοντέλο του Softmax Regression.

Μέρος Β - Προσθήκη Attention στο βέλτιστο μοντέλο

Στο δεύτερο μέρος της εργασίας θα προσθέσω attention στα βέλτιστα μοντέλα LSTM/GRU που προέκυψαν από την μελέτη του πρώτου μέρους και παρουσιάστηκαν τελικά στις παραγράφους Βέλτιστο μοντέλο LSTM και Βέλτιστο μοντέλο GRU αντίστοιχα.

Πιο συγκεκριμένα θα εφαρμόσω self-attention.
Το attention εφαρμόζεται στο output των stacked LSTM/GRU layers.

Για την υλοποίηση του attention mechanism ορίζω την κλάση Attention.
Η κλάση Attention είναι υπό-κλάση της nn.Module.
Η συνάρτηση __init__ της κλάσης Attention δέχεται σαν όρισμα την διάσταση του input που πρόκειται να δεχθεί (στην περίπτωση μας την διάσταση του output των LSTM/GRU layers) και ορίζει τα εξής:

- Ένα linear (input) layer με input_size την διάσταση του input και output size επίσης την διάσταση του input (δηλαδή 2*hidden_size).
- Ένα softmax layer.
- Ένα linear (output) layer με input_size διπλάσιο από την διάσταση του input (4*hidden_size) και output size την διάσταση του input (δηλαδή 2*hidden_size).
- Μια activation function, συγκεκριμένα την tanh.

Η εφαρμογή του attention γίνεται μέσω της συνάρτησης forward και η διαδικασία που ακολουθείται είναι η εξής:

- Αρχικά δέχεται δύο ορίσματα (query, context) τα οποία αντιστοιχούν στα δεδομένα στα οποία θα εφαρμόσουμε το attention και στα δεδομένα με τα οποία θα εφαρμόσουμε το attention πάνω στα πρώτα. Εφόσον θα εφαρμόσουμε self-attention, τα query και context ταυτίζονται και αντιστοιχούν στο output των LSTM/GRU layers.
- Κάνοντας το κατάλληλο reshape στο query set, εφαρμόζουμε το πρώτο γραμμικό input layer πάνω σε αυτό.
- Στην συνέχεια, μέσω της συνάρτησης torch.bmm υπολογίζουμε το batch product των δύο matrices: query και context. Όπου query πλέον είναι το output του πρώτου linear layer και context είναι το output των LSTM/GRU layers. Στην ουσία πολλαπλασιάζουμε κάθε ένα token ενός tweet με όλα τα υπόλοιπα tokens του tweet (έχοντας «περάσει» το ένα σύνολο από tokens από ένα γραμμικό layer). Έτσι προκύπτουν τα αρχικά attention scores.
- Έπειτα εφαρμόζουμε softmax στα attention scores που προέκυψαν από το multiplication του προηγούμενου βήματος. Έτσι προκύπτουν τα attention weights.
- Στην συνέχεια, μέσω της συνάρτησης torch.bmm υπολογίζουμε το γινόμενο του κάθε token με το αντίστοιχο attention weight που προέκυψε από την εφαρμογή της softmax. Δηλαδή κάνουμε matrix multiplication του context (δηλαδή των tokens του tweet) με το matrix που προέκυψε από το προηγούμενο βήμα.
- Συνδυάζεται το output του multiplication του προηγούμενου βήματος με το context μέσω ενός concatenate.
- Εφαρμόζεται το δεύτερο γραμμικό (output) layer. Έτσι τα δεδομένα «επιστρέφουν» στην αρχική τους διάσταση.
- Τέλος, εφαρμόζεται η tanh.

Για την υλοποίηση του Attention, βασίστηκα στην υλοποίηση που παρουσιάζεται στο Pytorch-NLP documentation.

Για να προσθέσουμε το attention layer στο νευρωνικό μας δίκτυο, το ορίζουμε εσωτερικά της κλάσης __init__ ως εξής:

```
self.attention_layer = Attention(hidden_size*2)
```

Για να εφαρμόσουμε το attention, προσθέτουμε το εξής έπειτα από την εκτέλεση των LSTM/GRU layers και πριν το τελευταίο γραμμικό output layer:

```
attention_out, weights = self.attention_layer(out,out)
```

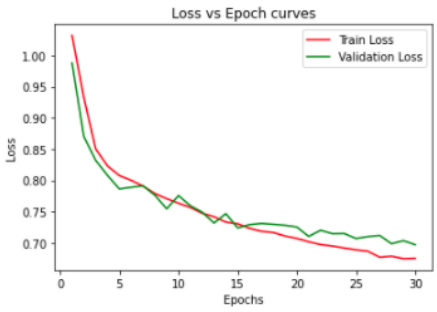
(όπου out το τελικό output LSTM/GRU layers)

Στην συνέχεια εφαρμόζω το attention mechanism στο Βέλτιστο μοντέλο LSTM.
Έπειτα από πειραματισμό, για την επίτευξη του βέλτιστου αποτελέσματος μειώνω την τιμή του learning rate σε 0.00007.
Οι υπόλοιπες παράμετροι παραμένουν ως έχουν.

Το μοντέλο, πλέον, είναι το εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (lstm): LSTM(200, 8, num_layers=2, batch_first=True, dropout=0.6, bidirectional=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (attention_layer): Attention(
    (linear_in): Linear(in_features=16, out_features=16, bias=False)
    (softmax): Softmax(dim=-1)
    (linear_out): Linear(in_features=32, out_features=16, bias=False)
    (tanh): Tanh()
  )
  (label): Linear(in_features=16, out_features=3, bias=True)
)
```

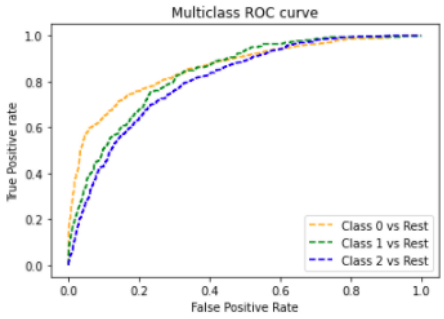
Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 68.97%
f1 score: 68.32%
Precision: 68.64%
Recall: 68.97%
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      1065
           1       0.50      0.31      0.39       296
           2       0.64      0.75      0.69       921

    accuracy                           0.69      2282
   macro avg       0.64      0.60      0.61      2282
weighted avg       0.69      0.69      0.68      2282
```



Παρατηρούμε:

- Τα αποτελέσματα είναι αντίστοιχα με αυτά του Βέλτιστο μοντέλο LSTM, με ελάχιστα χειρότερα scores στις μετρικές.
- Ισχύουν οι ίδιες παρατηρήσεις που αναφέρονται στην Αξιολόγηση μοντέλου.

Δοκιμάζοντας αντίστοιχες τεχνικές με αυτές που δοκιμάστηκαν στην μελέτη του πρώτου μέρους, δεν κατάφερα να βελτιώσω την απόδοση του μοντέλου.

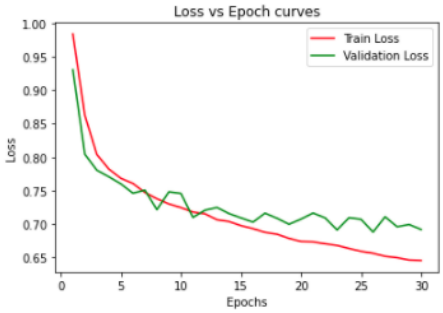
Επομένως συμπεραίνουμε ότι προσθέτοντας attention στο Βέλτιστο μοντέλο LSTM δεν βελτιώνεται η απόδοση του.

Στην συνέχεια εφαρμόζω το attention mechanism στο Βέλτιστο μοντέλο GRU.
Όλες παράμετροι παραμένουν ως έχουν.

Το μοντέλο, πλέον, είναι το εξής:

```
RNN(
  (emb): Embedding(1193514, 200)
  (gru1): GRU(200, 8, batch_first=True, bidirectional=True)
  (dropout_between_layers): Dropout(p=0.5, inplace=False)
  (gru2): GRU(16, 8, batch_first=True, bidirectional=True)
  (dropout): Dropout(p=0.25, inplace=False)
  (attention_layer): Attention(
    (linear_in): Linear(in_features=32, out_features=32, bias=False)
    (linear_out): Linear(in_features=64, out_features=32, bias=False)
    (softmax): Softmax(dim=-1)
    (tanh): Tanh()
  )
  (fc): Linear(in_features=32, out_features=3, bias=True)
)
```

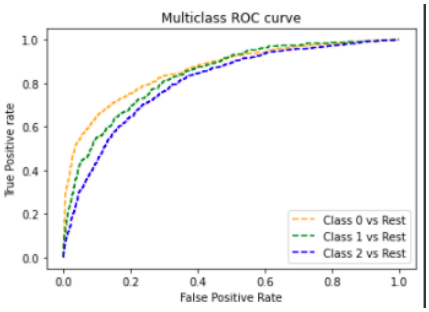
Τα αποτελέσματα είναι τα εξής:




```
Accuracy: 70.16%
f1 score: 69.60%
Precision: 70.07%
Recall: 70.16%
              precision    recall  f1-score   support

           0       0.78      0.74      0.76      1065
           1       0.58      0.36      0.44       296
           2       0.65      0.76      0.70       921

    accuracy                           0.70      2282
   macro avg       0.67      0.62      0.64      2282
weighted avg       0.70      0.70      0.70      2282
```



Παρατηρούμε:

- Τα αποτελέσματα όσον αφορά τα scores των μετρικών είναι καλύτερα από αυτά του Βέλτιστο μοντέλο GRU. Γενικότερα είναι καλύτερα από όλα τα μοντέλα που υλοποίησα στην εργασία αυτή.
- Ωστόσο υπάρχει μια μικρή απόκλιση μεταξύ των δύο καμπυλών train/validation loss, δηλαδή το μοντέλο παρουσιάζει ελάχιστο overfit.
- Κατά τα άλλα ισχύουν οι ίδιες παρατηρήσεις που αναφέρονται στην Αξιολόγηση μοντέλου.

Δοκιμάζοντας αντίστοιχες τεχνικές με αυτές που δοκιμάστηκαν στην μελέτη του πρώτου μέρους, δεν κατάφερα να βελτιώσω την απόδοση του μοντέλου.
Επομένως συμπεραίνουμε ότι προσθέτοντας attention στο Βέλτιστο μοντέλο GRU αυξάνεται η απόδοση του μοντέλου όσον αφορά τα scores των μετρικών αλλά ταυτόχρονα το μοντέλο παρουσιάζει ελάχιστο overfit.

Γενικό συμπέρασμα

Προσθέτοντας attention στο βέλτιστο μοντέλο LSTM που προέκυψε από την μελέτη του πρώτου μέρους της εργασίας, συμπεραίνουμε πως δεν βελτιώνεται η απόδοση του. Συγκεκριμένα, η απόδοση του μοντέλου είναι ελάχιστα χειρότερη ή ίδια σε σχέση με το αντίστοιχο μοντέλο χωρίς attention.
Όσον αφορά το μοντέλο GRU παρατηρούμε αύξηση στα scores των μετρικών αλλά ταυτόχρονα το μοντέλο παρουσιάζει ελάχιστο overfit.