Τεχνητή Νοημοσύνη 2
Εργασία 4

Ιωάννης Καπετανγεώργης
1115201800061

Περιεχόμενα

Γενικά σχόλια

- Η μελέτη που ακολουθεί είναι αρκετά αναλυτική και αρκετά μεγάλη σε μήκος έτσι ώστε να καλυφθεί κάθε λεπτομέρεια της υλοποίησης χωρίς ασάφειες. Είναι χωρισμένη κατάλληλα σε παραγράφους και υπό-παραγράφους για εύκολη πλοήγηση.
- Έχουν υλοποιηθεί όλα τα ζητούμενα της εργασίας. Όσον αφορά την άσκηση 3, έχουν συμπληρωθεί όλα τα κελία του Table 3 του paper.
- Στο παραδοτέο υπάρχει εκτελεσμένος ο κώδικας για όλα τα ερωτήματα της εργασίας. Τα περιεχόμενα του παραδοτέου εξηγούνται στην αμέσως επόμενη παράγραφο.
- Ο κώδικας είναι πλήρως σχολιασμένος.
- Για την εκτέλεση του κώδικα είναι απαραίτητη η χρήση GPU για την επιτάχυνση των υπολογισμών.
- Η υλοποίηση/εκτέλεση της πρώτης άσκησης της εργασίας έγινε μέσω της πλατφόρμας Google Colab.
- Η υλοποίηση/εκτέλεση των ασκήσεων 2 και 3 έγινε μέσω της πλατφόρμας Kaggle καθώς:
    - Οι περιορισμοί όσον αφορά την χρήση μνήμης είναι πιο «χαλαροί» και έτσι υπάρχει η δυνατότητα να εκτελεστεί κώδικας με μεγαλύτερες απαιτήσεις πόρων σε σχέση με το Google Colab.
    - Δίνει την δυνατότητα για συνεχόμενη πολύωρη εκτέλεση με χρήση της GPU, κάτι που φάνηκε απαραίτητο για την υλοποίηση των ασκήσεων 2 και 3. Αντίθετα, το Colab δεν επιτρέπει την πολύωρη συνεχόμενη εκτέλεση, επιβάλλοντας time-out στο account.
    - Ταχύτερη εκτέλεση σε σχέση με το Colab.
- Λόγω των παραπάνω, ενδεχομένως η εκτέλεση των ασκήσεων 2 και 3 του παραδοτέου αρχείου μου να είναι αδύνατες μέσω του Google Colab, καθώς μπορεί να υπερβαίνουν τα όρια χρήσης της μνήμης ή της vram της GPU. Ωστόσο, η εκτέλεση γίνεται κανονικά μέσω του Kaggle.
- Η μόνη περίπτωση εκτέλεσης στο Collab για την άσκηση 3 είναι η δημιουργία κάποιων datasets καθώς ο χώρος του output περιορίζεται στα 20GB. Χαρακτηριστικά, για την δημιουργία του dataset Natural Questions απαιτείται χώρος τουλάχιστον 40GB. Έτσι δημιούργησα τα αρχεία του dataset μέσω του Colab, τα μετέτρεψα σε SQuAD format, τα κατέβασα τοπικά και τέλος τα μετέφερα στο Kaggle ανεβάζοντας τα έτσι ώστε να τα χρησιμοποιήσω.
- **ΣΗΜΑΝΤΙΚΟ**: Για την δοκιμή του μοντέλου της πρώτης άσκησης με κάποιο test set χρειάζεται να αλλάξει το path που ορίζεται από την εξής εντολή στο πρώτο κελί του Μέρους Α:
  validation_set_location = r'vaccine_validation_set.csv'
  Δηλαδή, πρέπει να αντικατασταθεί το path του validation set, με το path του test set. Ωστόσο, πρέπει να σημειωθεί ότι η εκπαίδευση του μοντέλου είναι αρκετά χρονοβόρα (περίπου 30-40 λεπτά συνολικά).
  Για την εκτέλεση της πρώτης άσκησης, απαιτείται η εκτέλεση όλων των κελιών που βρίσκονται στο section “Μέρος Α” του παραδοτέου αρχείου. Δηλαδή μέχρι ΚΑΙ το κελί με τίτλο «Model Evaluation».

Παραδοτέο αρχείο

Το παραδοτέο αρχείο .ipynb χωρίζεται σε 2 sections: Μέρος Α και Μέρος Β.

Το Μέρος Α περιέχει την υλοποίηση για την πρώτη άσκηση της εργασίας. Έχει χωριστεί στις εξής υπό-ενότητες ανάλογα με την λειτουργία που εκτελείτε από τον αντίστοιχο κώδικα. Περιέχει τις εξής υπό-ενότητες:

- Data pre-processing
- Tokenization
- Finetuning
- Model Evaluation

Για τις ενότητες αυτές, υπάρχουν τα αντίστοιχα κεφάλαια στην αναλυτική μελέτη που ακολουθεί έτσι ώστε να εξηγηθεί πλήρως η λειτουργία του καθενός.
Για την εκτέλεση της πρώτης άσκησης (π.χ. για δοκιμή του μοντέλου σε test set) απαιτείται η εκτέλεση όλων των κελιών που περιέχονται στο section Μέρος Α (και των αντίστοιχων υπό-παραγράφων του).

Το Μέρος Β περιέχει την υλοποίηση των ασκήσεων 2 και 3 της εργασίας. Η άσκηση 2 αποτελεί υπό-ερώτημα της άσκησης 3 έτσι παρουσιάζονται μαζί στο παραδοτέο αρχείο.

Το Μέρος Β αποτελείται από 2 βασικές παραγράφους:

- Datasets Creation : όπου πραγματοποιείται η δημιουργία των 5 datasets και η μετατροπή τους (πλην του SQuAD) σε SQuAD format. Η διαδικασία εξηγείται αναλυτικά στην παράγραφο 3.1 Δημιουργία των datasets.
- Fine-Tuning and Testing : όπου πραγματοποιείται το fine-tuning και το evaluation των μοντέλων.

Αναλυτικότερα η παράγραφος Fine-Tuning and Testing αποτελείται από 5 υπό-παραγράφους, μία για κάθε dataset.
Κάθε υπό-παράγραφος, αντιστοιχεί σε ένα dataset (π.χ. στο SQuAD) και περιέχει:

- Κώδικα για το finetuning του μοντέλου χρησιμοποιώντας το training set του SQuAD.
- Evaluation του finetuned μοντέλου μέσω του dev set του SQuAD.
- Evaluation του finetuned μοντέλου μέσω των dev sets των υπόλοιπων τεσσάρων datasets, δηλαδή μία υπό-υπό-παράγραφο για κάθε ένα dataset.

Το παραδοτέο αρχείο είναι μεγάλο σε έκταση, ωστόσο περιλαμβάνονται σε αυτό όλες οι εκτελέσεις που έκανα έτσι ώστε να είναι προφανές ότι υλοποίησα στην πράξη όλα αυτά που αναφέρω στην συνέχεια.
Επίσης, περιέχει αρκετά κομμάτια duplicate κώδικα καθώς πολλές εκτελέσεις έγιναν σε διαφορετικά Notebooks όπως είναι αναμενόμενο. Ωστόσο τα κράτησα όλα ως έχουν έτσι ώστε να φαίνεται και το αντίστοιχο output από κάθε εκτέλεση.

Το Notebook έχει χωριστεί στα κατάλληλα sections και υπό-sections έτσι ώστε να είναι πιο κατανοητό, και να είναι ευκολότερη η πλοήγηση σε αυτό μέσω και του Table of Contents που παρέχει το Colab.

Μέρος Α –Sentiment Classification with Bert

Στην πρώτη άσκηση της εργασίας θα αναπτύξω έναν sentiment classifier για την κατηγοριοποίηση των tweets των datasets που μας δόθηκαν σε τρεις κλάσεις (neutral, anti-vax and pro-vax). Ο classifier θα προκύψει έπειτα από fine-tuning του pretrained BERT-base μοντέλου.
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
- Πειραματισμός με το tokenization
- Χρήση διαφορετικών παραλλαγών του μοντέλου BERT (BertModel, BertForSequenceClassification)
- Πειραματισμός με τις υπερπαραμέτρους για το fine tuning (batch_size, learning_rate, epochs)

1.1 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.
- Tokenization των tweets για την δημιουργία των αναπαραστάσεων που θα δοθούν σαν είσοδος στο Bert.
- Ορισμός του μοντέλου Bert. Στην περίπτωση της χρήσης του BertModel ορίζεται μια κλάση που υλοποιεί τον sentiment classifier. Στην περίπτωση της χρήσης BertForSequenceClassification, απλώς ορίζουμε το αντίστοιχο μοντέλο το οποίο γίνεται import. Περισσότερες λεπτομέρειες στις αντίστοιχες παραγράφους.
- Ορισμός του loss function που θα χρησιμοποιηθεί καθώς και του optimizer.
- Ορισμός των υπερπαραμέτρων για το finetuning του μοντέλου (learning_rate, batch_size και epochs).
- Μετατροπή των δεδομένων σε Tensors και δημιουργία των dataloaders για τον διαχωρισμό των δεδομένων σε batches.
- 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.

1.2 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 μοντέλο.
Στην μελέτη/σύγκριση που ακολουθεί παρουσιάζεται αναλυτικά η απόδοση της κάθε τεχνικής καθώς και το ποιες από αυτές επέλεξα να εφαρμόσω τελικά.

1.3 Tokenization

Για το tokenization των tweets χρησιμοποιώ τον BertTokenizer και συγκεκριμένα το μοντέλο bert-base-uncased.
Δηλαδή ο ορισμός του tokenizer γίνεται ως εξής:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

Κατά το tokenization μπορεί να οριστεί το μέγιστο μήκος της tokenized αναπαράστασης, δηλαδή ο μέγιστος αριθμός από tokens για κάθε tweet μέσω της μεταβλητής max_length.
Για μεγάλο αριθμό από tokens ο χρόνος εκπαίδευσης αυξάνεται αρκετά. Έτσι αρχικά επέλεξα να ορίσω της μεταβλητή max_length=65 έτσι ώστε να κάνω τις πρώτες δοκιμές της μελέτης μου.
Στην συνέχεια όπως παρουσιάζεται και στην συνέχεια της μελέτης αυξήσα σταδιακά τον αριθμό max_length έτσι ώστε να πετύχω το βέλτιστο αποτέλεσμα.
Έχοντας ορίσει την παράμετρο max_length, γίνεται περικοπή των τελευταίων tokens σε όσα tweets προκύπτει μεγαλύτερη αναπαράσταση από ότι το max_length ενώ για όσα tweets προκύπτει μικρότερη αναπαράσταση εφαρμόζεται padding.
Είναι προφανές ότι από τα tweets με μεγαλύτερες αναπαραστάσεις «χάνεται» πληροφορία λόγω του max_length κάτι που έχει συνέπειες και στην τελική απόδοση του μοντέλου έπειτα από το fine-tuning.

Έτσι η μελέτη που έκανα όσον αφορά το tokenization είναι το να βρω την «χρυσή τομή» για την παράμετρο max_length έτσι ώστε να μην χάνεται χρήσιμη πληροφορία ενώ ταυτόχρονα δεν δημιουργούνται πολύ μεγάλες αναπαραστάσεις μέσω τον οποίων αυξάνεται πολύ τόσο ο χρόνος εκπαίδευσης όσο και οι απαιτήσεις πόρων.

1.4 Batching

Έπειτα από το tokenization των tweets τόσο των του train set όσο και του validation set μεταξύ άλλων ο tokenizer μας επιστρέφει για κάθε ένα tweet τα input_ids (δηλαδή τα ids των tokens που προέκυψαν για το κάθε tweet) και το attention_mask (το οποίο υποδεικνύει στο μοντέλο ποια tokens να λάβει υπόψιν του κατά την εκπαίδευση).
Επίσης για κάθε tweet έχουμε και το αντίστοιχο label του.
Τα τρία αυτά στοιχεία θα χρησιμοποιήσουμε σαν δεδομένα κατά την εκπαίδευση. Δηλαδή για κάθε tweet έχουμε:

- input_ids
- attention_mask
- label

Μετατρέπουμε τα παραπάνω σε tensors και στην συνέχεια δημιουργούμε δύο DataLoaders έναν για το train set και έναν για το validation set αντίστοιχα. Λόγω του padding που εφαρμόζεται κατά το tokenization από τον tokenizer όλα τα tokenized tweets έχουν ίδιο μήκος και έτσι μπορούμε εύκολα να κατασκευάσουμε τους Dataloaders.
Έτσι τα δεδομένα μας χωρίζονται σε batches έτσι ώστε να γίνει η εκπαίδευση (fine tuning).

1.5 Ορισμός του Μοντέλου

Για τον ορισμό του pretrained μοντέλου που θα χρησιμοποιηθεί ακολουθείται διαφορετική διαδικασία ανάλογα με τον τύπο του.

Πιο συγκεκριμένα, στην περίπτωση του BertForSequenceClassification, αρκεί απλά να γίνει import και να οριστεί το συγκεκριμένο μοντέλο που θέλουμε να χρησιμοποιήσουμε καθώς και ο αριθμός των κλάσεων του προβλήματος μας.
Τα μοντέλα BertForSequenceClassification, όπως φαίνεται και από το όνομα, είναι κατασκευασμένα ειδικά για classification, έτσι έχοντας ορίσει τον κατάλληλο αριθμό των κλάσεων, το μοντέλο κάθε φορά επιστρέφει τα αντίστοιχα logits των κλάσεων που αντιστοιχούν στο input που δόθηκε.
Χρησιμοποιώ το bert-base-uncased μοντέλο και ο ορισμός του γίνεται ως εξής:

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=3)
```

Όσον αφορά την χρήση του BertModel μοντέλου, ο ορισμός του είναι διαφορετικός. Το BertModel είναι ένα μοντέλο γενικής χρήσης. Έτσι, για να το χρησιμοποιήσουμε για το classification πρέπει να ορίσουμε τα κατάλληλα layers έτσι ώστε το output του Bert να μετατραπεί στα αντίστοιχα logits των κλάσεων. Η λογική των επιπλέον layers είναι ίδια με αυτήν των Feed-Forward NN.
Για την δημιουργία του μοντέλου ορίζω την αντίστοιχη κλάση με όνομα BertClassifier. Αναλυτικότερη εξήγηση για τα περιεχόμενα της κλάσης καθώς και τον πειραματισμό σχετικά με τα layers του μοντέλου εξηγούνται στην αντίστοιχη παράγραφο της μελέτης που ακολουθεί.

1.6 Εκπαίδευση μοντέλου (Fine Tuning)

Έχοντας ορίσει τον επιθυμητό αριθμό epochs, το fine tuning 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.
    - 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, έχει ολοκληρωθεί το fine tuning του μοντέλου και βρίσκεται σε evaluation mode έτσι ώστε να γίνει το τελικό evaluation του.

1.7 Loss Function

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().
Έτσι στην περίπτωση χρήσης του BertModel, δεν χρειάζεται να προσθέσουμε στο μοντέλο μας Softmax layer.
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.»

Επίσης, το μοντέλο BertForSequenceClassification υπολογίζει εσωτερικά το loss.
Ωστόσο έχω επιλέξει να το υπολογίζω χειροκίνητα μέσω της συνάρτησης nn.CrossEntropyLoss() χρησιμοποιώντας τα logits που επίσης επιστρέφει το μοντέλο.

1.8 Optimizer

Οι δύο optimizers που χρησιμοποιώ είναι ο Adam και ο AdamW, με τους οποίους βέβαια παρατήρησα παρόμοια αποτελέσματα
Πιο συγκεκριμένα, με το μοντέλο BertModel χρησιμοποιώ τον Adam καθώς παρατήρησα ελάχιστα καλύτερη συμπεριφορά.
Αντίθετα με το μοντέλο BertForSequenceClassification χρησιμοποιώ τον AdamW καθώς υπερτερούσε ελάχιστα σε σχέση με τον Adam.

Όπως αναφέρεται και στις 2 μελέτες, δοκίμασα να χρησιμοποιήσω και διαφορετικούς optimizers ωστόσο η απόδοση του μοντέλου ήταν χειρότερη.

1.9 Αξιολόγηση του μοντέλου

Έπειτα από το fine tuning 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 1.6 Εκπαίδευση μοντέλου (Fine Tuning), 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.

1.10 Μελέτη για την εύρεση του βέλτιστου μοντέλου

Όπως αναφέρθηκε και παραπάνω, πραγματοποίησα δύο μελέτες χρησιμοποιώντας δύο διαφορετικά μοντέλα. Χρησιμοποίησα το BertModel και το BertForSequenceClassification. Στην συνέχεια παρουσιάζονται η δύο μελέτες για την εύρεση του καλύτερου μοντέλου στις δύο περιπτώσεις. Προφανώς δεν παρουσιάζονται όλες οι δοκιμές οι οποίες έκανα αλλά παρουσιάζεται μια σύνοψη αυτών.

1.10.1 BertModel

Για την χρήση του μοντέλου BertModel για το sentiment classification χρειάζεται να ορίσουμε μια κλάση έτσι ώστε να προσθέσουμε επιπλέον layers τα οποία λαμβάνουν ως είσοδο του output του BertModel και τελικά παράγουν τα αντίστοιχα logits που αντιστοιχούν στις 3 κλάσεις του προβλήματος.

Αρχικά ορίζω το μοντέλο με τα εξής layers (τα οποία εκτελούνται διαδοχικά):

- BertModel.from_pretrained('bert-base-uncased',num_labels=3)
- (dropout): Dropout(p=0.5, inplace=False)
- (linear): Linear(in_features=768, out_features=100, bias=True)
- (dropout2): Dropout(p=0.5, inplace=False)
- (relu): ReLU()
- (linear2): Linear(in_features=100, out_features=3, bias=True)

Δηλαδή τα βήματα που ακολουθούνται όταν δίνεται ένα input στο μοντέλο είναι:

- Εκτελείται το μοντέλο BertModel, το οποίο επιστρέφει output μεγέθους 768.
- Στο output του BertModel εφαρμόζεται dropout με dropout rate = 0.5
- Εφαρμόζεται ένα γραμμικό layer το οποίο λαμβάνει είσοδο μεγέθους 768 και παράγει έξοδο μεγέθους 100
- Στο output του γραμμικού layer εφαρμόζεται dropout με dropout rate = 0.5
- Στο output του dropout layer εφαρμόζεται activation function relu
- Τέλος εφαρμόζεται ένα γραμμικό layer το οποίο λαμβάνει είσοδο μεγέθους 100 (η οποία προέκυψε από το προηγούμενο γραμμικό layer καθώς και τα dropout και activation layers) και παράγει έξοδο μεγέθους 3 (δηλαδή όσο και ο αριθμός των κλάσεων). Από το γραμμικό layer αυτό παράγονται τελικά τα logits που αντιστοιχούν στην κάθε κλάση.

Όσον αφορά την προεπεξεργασία των δεδομένων, αρχικά εφαρμόζω την ίδια προεπεξεργασία η οποία αποδείχθηκε καλύτερη στην προηγούμενη εργασία. Δηλαδή εφαρμόζω:

- 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.

Όπως αναφέρθηκε και παραπάνω έχω επιλέξει αρχικά να εφαρμόσω το tokenization με max_length = 65 και στην συνέχεια της μελέτης να το αυξήσω. Για το tokenization χρησιμοποιώ τον BertTokenizer και συγκεκριμένα το bert-base-uncased.

Όσον αφορά τις υπερπαραμέτρους αρχικά επιλέγω τις εξής τιμές:

- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 0.00001

Το loss function που χρησιμοποιώ είναι το nn.CrossEntropyLoss().
Ο optimizer που χρησιμοποιώ είναι ο Adam. Δοκιμάζοντας και διαφορετικούς optimizers η απόδοση ήταν χειρότερη σε όλες τις περιπτώσεις με εξαίρεση τον AdamW ο οποίος παρουσίασε παρόμοια απόδοση.

Εκπαιδεύοντας το μοντέλο 2 φορές με ακριβώς τις ίδιες υπερπαραμέτρους/προεπεξεργασία/tokenization τα αποτελέσματα είχαν μεγάλη διαφορά μεταξύ τους. Τα αποτελέσματα της πρώτης εκτέλεσης (τα οποία είναι και καλύτερα) δεν κατάφερα να τα πετύχω ξανά.

Πιο συγκεκριμένα, στην πρώτη εκπαίδευση τα αποτελέσματα ήταν τα εξής:

- Πληροφορίες ανά epoch

```
Epoch  0: Train Loss = 0.87597 | Validation Loss = 0.75326 | Accuracy = 66.3015 | Train-f1 = 0.5575 | Valid-F1 = 0.6216
Epoch  1: Train Loss = 0.75230 | Validation Loss = 0.70677 | Accuracy = 68.4049 | Train-f1 = 0.6477 | Valid-F1 = 0.6505
Epoch  2: Train Loss = 0.68584 | Validation Loss = 0.68150 | Accuracy = 71.5600 | Train-f1 = 0.7007 | Valid-F1 = 0.7048
```

- Τελικά αποτελέσματα μετρικών

Accuracy: 71.56%
f1 score: 70.48%
Precision: 73.40%
Recall: 71.56%

Στην δεύτερη εκπαίδευση τα αποτελέσματα ήταν τα εξής:

- Πληροφορίες ανά epoch

```
Epoch  0: Train Loss = 0.89351 | Validation Loss = 0.76135 | Accuracy = 66.1700 | Train-f1 = 0.5450 | Valid-F1 = 0.6197
Epoch  1: Train Loss = 0.75432 | Validation Loss = 0.71344 | Accuracy = 67.9229 | Train-f1 = 0.6405 | Valid-F1 = 0.6458
Epoch  2: Train Loss = 0.68959 | Validation Loss = 0.74667 | Accuracy = 68.1858 | Train-f1 = 0.6953 | Valid-F1 = 0.6771
```

- Τελικά αποτελέσματα μετρικών

Accuracy: 68.19%
f1 score: 67.71%
Precision: 67.53%
Recall: 68.19%

Στην πρώτη περίπτωση παρατηρούμε πως το μοντέλο δεν παρουσιάζει καθόλου overfit και ταυτόχρονα τα αποτελέσματα των μετρικών είναι αρκετά ικανοποιητικά.

Αντίθετα στην δεύτερη περίπτωση παρατηρούμε πως το μοντέλο παρουσιάζει αρκετό overfit (κάτι που φαίνεται από την μεγάλη διαφορά των train/validation loss) ενώ ταυτόχρονα τα scores των μετρικών έχουν μειωθεί αρκετά.

Δοκιμάζοντας μια τρίτη εκπαίδευση του μοντέλου, τα αποτελέσματα ήταν παρόμοια με αυτά της δεύτερης.

Έτσι δοκίμασα να τροποποιήσω το μοντέλο.

Αρχικά αφαίρεσα το ReLU activation layer, η υπόλοιπη δομή του μοντέλου καθώς και οι υπερπαραμέτροι παραμένουν ως έχουν.
Εκπαιδεύοντας ξανά το μοντέλο τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.88532 | Validation Loss = 0.75194 | Accuracy = 66.6082 | Train-f1 = 0.5513 | Valid-F1 = 0.6253
Epoch 1: Train Loss = 0.74622 | Validation Loss = 0.70640 | Accuracy = 68.6240 | Train-f1 = 0.6544 | Valid-F1 = 0.6602
Epoch 2: Train Loss = 0.67864 | Validation Loss = 0.70405 | Accuracy = 69.8510 | Train-f1 = 0.7055 | Valid-F1 = 0.6963

- Τελικά αποτελέσματα μετρικών

Accuracy: 69.85%
f1 score: 69.63%
Precision: 69.49%
Recall: 69.85%

Παρατηρούμε πως:

- Το overfit έχει μειωθεί αρχικά καθώς το train loss και το validation loss είναι αρκετά πιο κοντά
- Τα scores των μετρικών έχουν αυξηθεί σε σχέση με το προηγούμενο μοντέλο
- Δοκιμάζοντας να εκπαιδεύσω ξανά το μοντέλο τα αποτελέσματα ήταν παρόμοια

Έτσι η αφαίρεση του activation layer αποδείχθηκε αποδοτική.

Με στόχο την πλήρη εξάλειψη του overfit αλλά και την αύξηση της απόδοσης του μοντέλου δοκιμάζω περαιτέρω τροποποίηση του μοντέλου.
Έχοντας αφαιρέσει το activation layer, τροποποιώ το hidden size του πρώτου hidden layer από 100 σε 64 έτσι ώστε να απλοποιηθεί λίγο το μοντέλο.
Επίσης λόγω της απλοποίησης του μοντέλου, αυξάνω το batch_size από 32 σε 64.
Έτσι το μοντέλο είναι πλέον το εξής:

- BertModel.from_pretrained('bert-base-uncased',num_labels=3)
- (dropout): Dropout(p=0.5, inplace=False)
- (linear): Linear(in_features=768, out_features=64, bias=True)
- (dropout2): Dropout(p=0.5, inplace=False)
- (linear2): Linear(in_features=64, out_features=3, bias=True)

Οι υπερπαραμέτροι είναι:

- TRAINING_EPOCHS = 3
- batch_size = 64
- learning_rate = 0.00001

Έπειτα από το fine tuning, τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.88206 | Validation Loss = 0.76210 | Accuracy = 66.6959 | Train-f1 = 0.5532 | Valid-F1 = 0.6247
Epoch 1: Train Loss = 0.76538 | Validation Loss = 0.73174 | Accuracy = 67.9229 | Train-f1 = 0.6252 | Valid-F1 = 0.6430
Epoch 2: Train Loss = 0.72306 | Validation Loss = 0.70561 | Accuracy = 68.8431 | Train-f1 = 0.6589 | Valid-F1 = 0.6614

Παρατηρούμε υψηλότερο train loss σε σχέση με πριν ενώ ταυτόχρονα το validation loss είναι χαμηλότερο, δηλαδή παρουσιάζεται underfit.

Έτσι αυξάνω το learing_rate σε 0.00002

Τα αποτελέσματα έπειτα από την τροποποίηση του learning rate είναι:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.84633 | Validation Loss = 0.74278 | Accuracy = 67.7038 | Train-f1 = 0.5751 | Valid-F1 = 0.6343
Epoch 1: Train Loss = 0.73756 | Validation Loss = 0.70862 | Accuracy = 68.3173 | Train-f1 = 0.6415 | Valid-F1 = 0.6614
Epoch 2: Train Loss = 0.66624 | Validation Loss = 0.69238 | Accuracy = 69.6757 | Train-f1 = 0.7025 | Valid-F1 = 0.6851

- Τελικά αποτελέσματα μετρικών

Accuracy: 69.68%
f1 score: 68.51%
Precision: 69.21%
Recall: 69.68%

Παρατηρούμε πως:

- Πλέον δεν παρουσιάζεται underfit. Αντίθετα, το train loss είναι χαμηλότερο από το validation loss και παρουσιάζεται ελάχιστο overfit.
- Ωστόσο, τα scores των μετρικών είναι χαμηλότερα σε σχέση με πριν.
- Δηλαδή, το μοντέλο συνεχίζει να παρουσιάζει αντίστοιχο overfit με πριν (σε μικρό βαθμό) ενώ ταυτόχρονα τα scores των μετρικών έχουν μειωθεί.

Άρα, η τροποποίηση του μοντέλου δεν αποδείχθηκε χρήσιμη.

Στην συνέχεια, δοκιμάζω να απλοποιήσω ακόμη περισσότερο το μοντέλο.
Πιο συγκεκριμένα, έχοντας ήδη αφαιρέσει το activation layer, αφαιρώ και το πρώτο γραμμικό layer μαζί με το αντίστοιχο dropout layer.
Επίσης επαναφέρω τις υπερπαραμέτρους σε αυτές που αποδείχθηκαν αποδοτικότερες σύμφωνα με τις παραπάνω εκτελέσεις.
Δηλαδή το μοντέλο είναι το εξής:

- BertModel.from_pretrained('bert-base-uncased',num_labels=3)
- (dropout): Dropout(p=0.5, inplace=False)
- (linear): Linear(in_features=768, out_features=3, bias=True)

Οι υπερπαραμέτροι είναι:

- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 0.00001

Τα αποτελέσματα είναι:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.83230 | Validation Loss = 0.72966 | Accuracy = 68.0105 | Train-f1 = 0.5855 | Valid-F1 = 0.6376
Epoch 1: Train Loss = 0.71333 | Validation Loss = 0.69694 | Accuracy = 70.1139 | Train-f1 = 0.6661 | Valid-F1 = 0.6882
Epoch 2: Train Loss = 0.64306 | Validation Loss = 0.69256 | Accuracy = 70.6836 | Train-f1 = 0.7132 | Valid-F1 = 0.6999

- Τελικά αποτελέσματα μετρικών

Accuracy: 70.68%
f1 score: 69.99%
Precision: 72.48%
Recall: 70.68%

Παρατηρούμε πως:

- Τα scores των μετρικών έχουν αυξηθεί και είναι τα καλύτερα μέχρι τώρα (εξαιρώντας την πρώτη εκτέλεσης).
- Παρατηρούμε μια μικρή διαφορά (αντίστοιχη με προηγουμένως) μεταξύ του train loss και του validation loss, δηλαδή το μοντέλο παρουσιάζει λίγο overfit.
- Εκπαιδεύοντας ξανά το ίδιο μοντέλο με τις ίδιες υπερπαραμέτρους τα αποτελέσματα ήταν αντίστοιχα.

Οποιαδήποτε άλλη τροποποίηση του μοντέλου ή των υπερπαραμέτρων δεν βελτίωσαν την απόδοση του μοντέλου.

Έτσι στην συνέχεια θα αυξήσω των μέγιστο αριθμό από tokens που προκύπτουν για τα tweets κατά το tokenization.
Για την εύρεση της κατάλληλης τιμής για το max_length έκανα το εξής:

- Χρησιμοποιώντας τον tokenizer έκανα tokenize το κάθε tweet ξεχωριστά χωρίς να επιβάλλω κάποιον περιορισμό στο max_length
- Αφού ολοκληρώθηκε το tokenization για κάθε ένα tweet, ελέγχοντας κάθε ένα tweet, εντόπισα το μεγαλύτερο πλήθος από tokens που προέκυψε για ένα tweet.

Ο αριθμός που προέκυψε είναι 140.

Έτσι ορίζω το max_length ίσο με 150 έτσι ώστε να μην χάνεται καθόλου πληροφορία από κανένα tweet κατά το tokenization. Για τα tweets που προκύπτουν αναπαραστάσεις με λιγότερα από 150 tokens εφαρμόζεται padding στα tokens που απομένουν για την επίτευξη του max_length.

Έχοντας τα νέα tokenized tweets ακολουθώ την ίδια διαδικασία με παραπάνω.
Αρχικά χρησιμοποιώ το πρώτο μοντέλο που παρουσίασα παραπάνω
Δηλαδή το μοντέλο είναι το εξής:

- BertModel.from_pretrained('bert-base-uncased',num_labels=3)
- (dropout): Dropout(p=0.5, inplace=False)
- (linear): Linear(in_features=768, out_features=100, bias=True)
- (dropout2): Dropout(p=0.5, inplace=False)
- (relu): ReLU()
- (linear2): Linear(in_features=100, out_features=3, bias=True)

Οι υπερπαραμέτροι είναι:

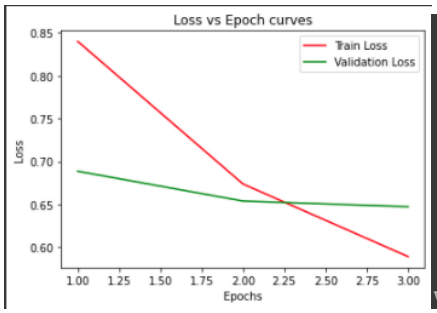
- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 0.00001

Τα αποτελέσματα από το fine tuning είναι τα εξής:

- Πληροφορίες ανά epoch

```
Epoch  0: Train Loss = 0.83989 | Validation Loss = 0.68857 | Accuracy = 70.4645 | Train-f1 = 0.5916 | Valid-F1 = 0.6936
Epoch  1: Train Loss = 0.67377 | Validation Loss = 0.65384 | Accuracy = 72.2612 | Train-f1 = 0.7172 | Valid-F1 = 0.7268
Epoch  2: Train Loss = 0.58909 | Validation Loss = 0.64717 | Accuracy = 73.3129 | Train-f1 = 0.7688 | Valid-F1 = 0.7335
```

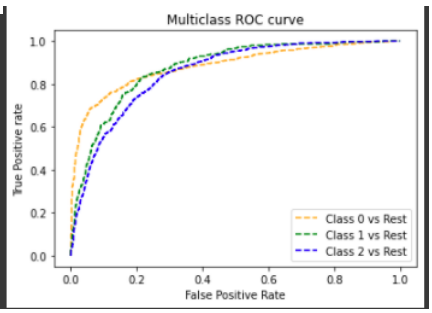
- Τελικά αποτελέσματα μετρικών




```
Accuracy: 73.31%
f1 score: 73.35%
Precision: 73.95%
Recall: 73.31%
              precision    recall  f1-score   support

           0       0.84      0.75      0.80      1065
           1       0.54      0.49      0.51       296
           2       0.68      0.79      0.73       921

    accuracy                           0.73      2282
   macro avg       0.69      0.68      0.68      2282
weighted avg       0.74      0.73      0.73      2282
```



Παρατηρούμε πως:

- Τα scores των μετρικών έχουν αυξηθεί αρκετά σε σχέση με την καλύτερη εκτέλεση μέχρι τώρα και είναι πολύ ικανοποιητικά.
- Παρατηρούμε πως στο τελευταίο epoch το validation loss σχεδόν παραμένει σταθερό ενώ ταυτόχρονα το train loss μειώνεται. Δηλαδή το μοντέλο παρουσιάζει overfit.

Έτσι, ακολουθώ την ίδια διαδικασία με παραπάνω έτσι ώστε να εξαλείψω το overfit και να βελτιώσω την συμπεριφορά του μοντέλου.

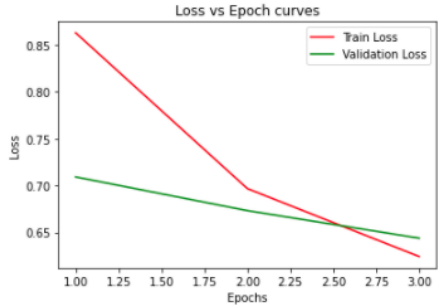
Αρχικά κρατώντας ίδια την δομή του μοντέλου πειραματίζομαι με τις υπερπαραμέτρους. Μειώνω το learning rate σε 0.0000075 ενώ κρατώ το batch_size και τον αριθμό των epochs ως έχουν.

Τα αποτελέσματα από το fine tuning είναι τα εξής:

- Πληροφορίες ανά epoch

```
Epoch  0: Train Loss = 0.86308 | Validation Loss = 0.70920 | Accuracy = 69.4128 | Train-f1 = 0.5744 | Valid-F1 = 0.6730
Epoch  1: Train Loss = 0.69673 | Validation Loss = 0.67323 | Accuracy = 70.6836 | Train-f1 = 0.6992 | Valid-F1 = 0.7124
Epoch  2: Train Loss = 0.62421 | Validation Loss = 0.64381 | Accuracy = 72.7432 | Train-f1 = 0.7471 | Valid-F1 = 0.7269
```

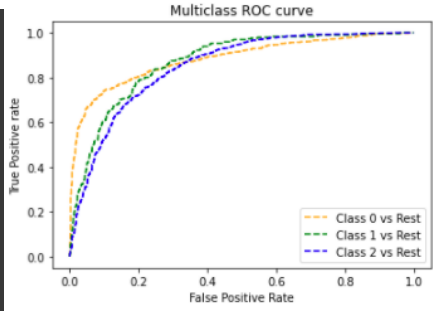
- Τελικά αποτελέσματα μετρικών




```
Accuracy: 72.74%
f1 score: 72.69%
Precision: 73.34%
Recall: 72.74%
              precision    recall  f1-score   support

           0       0.85      0.75      0.80      1065
           1       0.52      0.44      0.48       296
           2       0.67      0.79      0.73       921

    accuracy                           0.73      2282
   macro avg       0.68      0.66      0.67      2282
weighted avg       0.73      0.73      0.73      2282
```



Παρατηρούμε πως:

- Τα scores των μετρικών έχουν μειωθεί ελάχιστα σε σχέση με πριν.
- Το μοντέλο δεν παρουσιάζει overfit καθώς το train loss και το validation loss τελικά είναι πολύ κοντά.

Στην συνέχεια δοκίμασα να μειώσω την πολυπλοκότητα του μοντέλου (όπως και πριν την αλλαγή του max_length).
Πιο συγκεκριμένα, όπως και παραπάνω, δοκίμασα να «κρατήσω» μόνο ένα γραμμικό layer στο μοντέλο, να αφαιρέσω την ReLU activation function και να πειραματιστώ με τις υπερπαραμέτρους. Η καλύτερη επίδοση που πέτυχα ήταν η εξής:

Μοντέλο:

- BertModel.from_pretrained('bert-base-uncased',num_labels=3)
- (dropout): Dropout(p=0.5, inplace=False)
- (linear): Linear(in_features=768, out_features=3, bias=True)

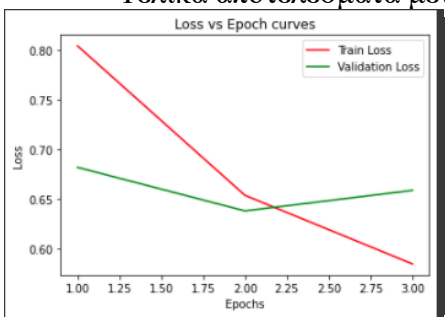
Υπερπαράμετροι:

- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 0.0000075

Πληροφορίες ανά epoch:

Epoch 0: Train Loss = 0.80443 | Validation Loss = 0.68204 | Accuracy = 69.8948 | Train-f1 = 0.6117 | Valid-F1 = 0.6716
Epoch 1: Train Loss = 0.65385 | Validation Loss = 0.63819 | Accuracy = 72.2612 | Train-f1 = 0.7153 | Valid-F1 = 0.7171
Epoch 2: Train Loss = 0.58478 | Validation Loss = 0.65884 | Accuracy = 72.8747 | Train-f1 = 0.7558 | Valid-F1 = 0.7236

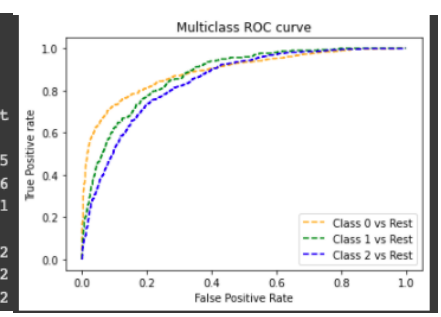
Τελικά αποτελέσματα μετρικών:




```
Accuracy: 72.87%
f1 score: 72.36%
Precision: 74.10%
Recall: 72.87%
              precision    recall  f1-score   support

           0       0.86      0.73      0.79      1065
           1       0.60      0.36      0.45       296
           2       0.65      0.84      0.73       921

    accuracy                           0.73      2282
   macro avg       0.70      0.65      0.66      2282
weighted avg       0.74      0.73      0.72      2282
```



Παρατηρούμε πως:

- Τα scores των μετρικών έχουν μειωθεί σε σχέση πριν.
- Το μοντέλο παρουσιάζει έντονα το φαινόμενο του overfit κάτι που φαίνεται από την μεγάλη απόκλιση των train και validation loss και από το γεγονός ότι στο τελευταίο epoch το validation loss αυξάνεται σημαντικά ενώ το train loss συνεχίζει να μειώνεται.

Έτσι η μείωση της πολυπλοκότητας του μοντέλου δεν αυξάνει καθόλου την απόδοση του, αντίθετα με πριν την αύξηση του max_length.

Επίσης δοκιμάζοντας διαφορετικούς συνδυασμούς προεπεξεργασίας των δεδομένων η απόδοση δεν αυξανόταν.

1.10.1.1 Βέλτιστο μοντέλο BertModel

Με βάση την παραπάνω μελέτη τα καλύτερα αποτελέσματα τα παίρνουμε ως εξής:

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.

Στην συνέχεια εφαρμόζουμε tokenization χρησιμοποιώντας τον:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

και ορίζοντας την παράμετρο max_length ίση με 150.
Κατά το tokenization δεν «χάνεται» καθόλου πληροφορία καθώς το μέγιστο μήκος από tokens που προκύπτει είναι 140.

Όσον αφορά το μοντέλο χρησιμοποιώ το BertModel (bert-base-uncased) προσθέτοντας επιπλέον layers μετά από αυτό. Ο ορισμός του συνολικού μοντέλου γίνεται μέσω μιας κλάσης (όπως ένα Feed-Forward NN).
Πιο συγκεκριμένα το μοντέλο ορίζεται ως εξής:

```
class BertClassifier(nn.Module):

    def __init__(self,num_labels=3, dropout=0.5):

        super(BertClassifier, self).__init__()

        self.bert = BertModel.from_pretrained('bert-base-uncased',num_labels=num_labels)
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(768, 100)
        self.dropout2 = nn.Dropout(dropout)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(100, num_labels)

    def forward(self, input_id, mask):

        _, pooled_output = self.bert(input_ids= input_id, attention_mask=mask,return_dict=False)
        dropout_output = self.dropout(pooled_output)
        linear_output = self.linear(dropout_output)
        final_layer = self.dropout2(linear_output)
        final_layer = self.relu(final_layer)
        final_layer = self.linear2(final_layer)
        return final_layer
```

Οι υπερπαραμέτροι που χρησιμοποίησα για το fine tuning είναι:

- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 0.0000075

Επίσης κατά την εκπαίδευση εφάρμοσα gradient clipping μέσω της συνάρτησης:

```
nn.utils.clip_grad_norm_
```

Το Loss Function που χρησιμοποίησα είναι το nn.CrossEntropyLoss().
Ο optimizer που χρησιμοποίησα ήταν ο Adam.

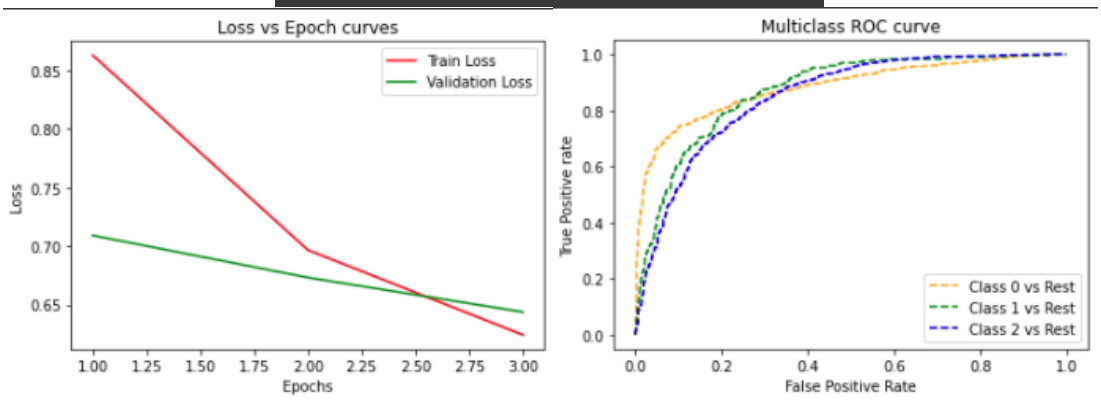
1.10.1.2 Αξιολόγηση βέλτιστου μοντέλου BertModel

Έπειτα από το fine tuning του παραπάνω μοντέλου τα αποτελέσματα των μετρικών είναι τα εξής:

```
Accuracy: 72.74%
f1 score: 72.69%
Precision: 73.34%
Recall: 72.74%
              precision    recall  f1-score   support

           0       0.85      0.75      0.80      1065
           1       0.52      0.44      0.48       296
           2       0.67      0.79      0.73       921

    accuracy                           0.73      2282
   macro avg       0.68      0.66      0.67      2282
weighted avg       0.73      0.73      0.73      2282
```



Συμπεράσματα:

- Το μοντέλο μας έχει αρκετά 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. 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.

1.10.2 BertForSequenceClassification

Το μοντέλο BertForSequenceClassification είναι κατασκευασμένο για classification, επομένως (σε αντίθεση με το BertModel) για την χρήση του αρκεί μόνο να ορίσουμε τον αριθμό των κλάσεων του προβλήματος μας.

Ο ορισμός του μοντέλου γίνεται μέσω της παρακάτω εντολής:

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=3)
```

Όσον αφορά την προεπεξεργασία των δεδομένων, εφαρμόζω την ίδια προεπεξεργασία η οποία αποδείχθηκε καλύτερη τόσο στην προηγούμενη εργασία όσο και στην μελέτη με το BertModel. Δοκιμάζοντας διαφορετικούς συνδυασμούς προεπεξεργασίας κατά την ακόλουθη μελέτη η απόδοση δεν αυξανόταν.
Δηλαδή εφαρμόζω:

- 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.

Αντίστοιχα με την παραπάνω μελέτη έχω επιλέξει αρχικά να εφαρμόσω το tokenization με max_length = 65 λόγο του αρκετά μικρότερου χρόνου εκτέλεσης έτσι ώστε να κάνω τις αρχικές παρατηρήσεις και στην συνέχεια της μελέτης να το αυξήσω. Για το tokenization χρησιμοποιώ τον BertTokenizer και συγκεκριμένα το bert-base-uncased.

Όσον αφορά τις υπερπαραμέτρους αρχικά επιλέγω τις εξής τιμές:

- TRAINING_EPOCHS = 3
- batch_size = 16
- learning_rate = 0.000006

Επίσης:

- Το loss function που χρησιμοποιώ είναι το nn.CrossEntropyLoss().
- Ο optimizer που χρησιμοποιώ είναι ο AdamW. Δοκιμάζοντας και διαφορετικούς optimizers η απόδοση ήταν χειρότερη σε όλες τις περιπτώσεις με εξαίρεση τον Adam ο οποίος παρουσίασε παρόμοια (ελάχιστα χειρότερη) απόδοση.

Πραγματοποιώντας το πρώτο fine tuning με βάση τα παραπάνω τα αποτελέσματα ήταν τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.80432 | Validation Loss = 0.73508 | Accuracy = 67.5285 | Train-f1 = 0.6001 | Valid-F1 = 0.6337
Epoch 1: Train Loss = 0.70425 | Validation Loss = 0.69715 | Accuracy = 69.7634 | Train-f1 = 0.6735 | Valid-F1 = 0.6957
Epoch 2: Train Loss = 0.64281 | Validation Loss = 0.70302 | Accuracy = 70.2892 | Train-f1 = 0.7192 | Valid-F1 = 0.6979

- Τελικά αποτελέσματα μετρικών

Accuracy: 70.29%
f1 score: 69.79%
Precision: 69.60%
Recall: 70.29%

Παρατηρούμε:

- Τα scores των μετρικών είναι σχετικά ικανοποιητικά
- Ωστόσο το μοντέλο παρουσιάζει overfit καθώς έχουμε αρκετή απόκλιση μεταξύ train/validation loss έπειτα από το πέρας του τρίτου epoch.

Στην συνέχεια δοκιμάζω να τροποποιήσω τις υπερπαραμέτρους αυξάνοντας το learning rate και παράλληλα αυξάνοντας το batch size. Ο αριθμός των epochs μένει ίδιος.
Πιο συγκεκριμένα ορίζω:

- batch_size = 32
- learning_rate = 0.00001

Τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.80819 | Validation Loss = 0.73197 | Accuracy = 66.8273 | Train-f1 = 0.5936 | Valid-F1 = 0.6276
Epoch 1: Train Loss = 0.70944 | Validation Loss = 0.70366 | Accuracy = 69.6319 | Train-f1 = 0.6615 | Valid-F1 = 0.6890
Epoch 2: Train Loss = 0.63947 | Validation Loss = 0.69227 | Accuracy = 71.0342 | Train-f1 = 0.7204 | Valid-F1 = 0.7054

- Τελικά αποτελέσματα μετρικών

Accuracy: 71.03%
f1 score: 70.54%
Precision: 72.59%
Recall: 71.03%

Παρατηρούμε:

- Τα scores των μετρικών αυξηθεί σε σχέση με πριν και το τελικό validation loss είναι μικρότερο από την προηγούμενη εκπαίδευση.
- Ωστόσο το μοντέλο συνεχίζει να παρουσιάζει overfit.

Με στόχο την εξάλειψη του overfit και σε δεύτερη φάση την αύξηση της απόδοσης του μοντέλου δοκιμάζω διαφορετικές τιμές για το learning rate.

Για **learning_rate = 2e-6** τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.98502 | Validation Loss = 0.88081 | Accuracy = 60.7800 | Train-f1 = 0.4768 | Valid-F1 = 0.5610
Epoch 1: Train Loss = 0.85252 | Validation Loss = 0.78964 | Accuracy = 63.8913 | Train-f1 = 0.5777 | Valid-F1 = 0.5998
Epoch 2: Train Loss = 0.79226 | Validation Loss = 0.76262 | Accuracy = 65.5565 | Train-f1 = 0.6131 | Valid-F1 = 0.6299

- Τελικά αποτελέσματα μετρικών

Accuracy: 65.56%
f1 score: 62.99%
Precision: 66.08%
Recall: 65.56%

Παρατηρούμε:

- Τα scores των μετρικών έχουν μειωθεί πολύ και το train loss είναι πολύ υψηλό.
- Το μοντέλο παρουσιάζει underfit καθώς το validation loss είναι αρκετά χαμηλότερο από το train loss.

Είναι προφανές ότι το learning_rate πρέπει να αυξηθεί.

Για **learning_rate = 7.5e-6** τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.81871 | Validation Loss = 0.74191 | Accuracy = 66.4330 | Train-f1 = 0.5903 | Valid-F1 = 0.6237
Epoch 1: Train Loss = 0.71844 | Validation Loss = 0.70693 | Accuracy = 69.8948 | Train-f1 = 0.6601 | Valid-F1 = 0.6871
Epoch 2: Train Loss = 0.65804 | Validation Loss = 0.69867 | Accuracy = 70.2892 | Train-f1 = 0.7101 | Valid-F1 = 0.6943

- Τελικά αποτελέσματα μετρικών

Accuracy: 70.29%
f1 score: 69.43%
Precision: 72.69%
Recall: 70.29%

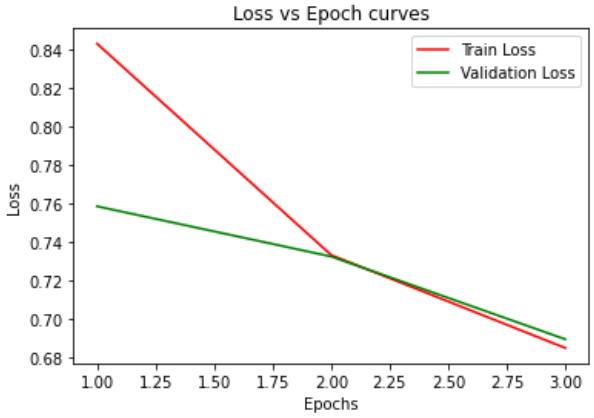
Παρατηρούμε:

- Τα scores των μετρικών έχουν αυξηθεί ξανά.
- Ωστόσο το μοντέλο πάλι παρουσιάζει overfit (σε μικρότερο βαθμό σε σχέση με πριν).

Τελικά επιλέγω να ορίσω το **learning_rate = 5e-6** και τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.84278 | Validation Loss = 0.75831 | Accuracy = 66.7835 | Train-f1 = 0.5752 | Valid-F1 = 0.6251
Epoch 1: Train Loss = 0.73302 | Validation Loss = 0.73224 | Accuracy = 68.3611 | Train-f1 = 0.6431 | Valid-F1 = 0.6798
Epoch 2: Train Loss = 0.68474 | Validation Loss = 0.68929 | Accuracy = 70.9027 | Train-f1 = 0.6890 | Valid-F1 = 0.7058



- Τελικά αποτελέσματα μετρικών

Accuracy: 70.90%
f1 score: 70.58%
Precision: 70.60%
Recall: 70.90%

Παρατηρούμε:

- Τα scores των μετρικών είναι τα καλύτερα έως τώρα.
- Το μοντέλο δεν παρουσιάζει καθόλου overfit η underfit. Αντίθετα τα train/validation loss curves σχεδόν ταυτίζονται.

Έτσι, η απόδοση του μοντέλου είναι αρκετά ικανοποιητική.

Στην συνέχεια δοκιμάζω να χρησιμοποιήσω τον Adam optimizer, χρησιμοποιώντας ακριβώς τις ίδιες παραμέτρους με παραπάνω.
Τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.84274 | Validation Loss = 0.75833 | Accuracy = 66.6959 | Train-f1 = 0.5750 | Valid-F1 = 0.6242
Epoch 1: Train Loss = 0.73321 | Validation Loss = 0.73040 | Accuracy = 68.5364 | Train-f1 = 0.6433 | Valid-F1 = 0.6822
Epoch 2: Train Loss = 0.68475 | Validation Loss = 0.68889 | Accuracy = 70.7274 | Train-f1 = 0.6898 | Valid-F1 = 0.7035

- Τελικά αποτελέσματα μετρικών

Accuracy: 70.73%
f1 score: 70.35%
Precision: 70.41%
Recall: 70.73%

Παρατηρούμε την ίδια συμπεριφορά και ελάχιστα χειρότερα scores στις μετρικές.

Δεν κατάφερα να βελτιώσω περισσότερο την απόδοση του μοντέλου.
Έτσι, στην συνέχεια, αυξάνω την παράμετρο max_length κατά το tokenization ορίζοντας την σε 150. Όπως αναφέρθηκε και παραπάνω, κατά το tokenization δεν «χάνεται» καθόλου πληροφορία καθώς το μέγιστο μήκος από tokens που προκύπτει είναι 140.

Έχοντας τα νέα tokenized tweets εκπαιδεύω ξανά το τελευταίο μοντέλο που είχε την βέλτιστη απόδοση έως τώρα.
Οι υπερπαράμετροι παραμένουν ως έχουν, δηλαδή:

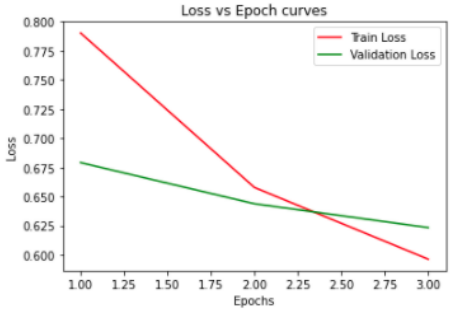
- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 5e-6

Τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

Epoch 0: Train Loss = 0.79033 | Validation Loss = 0.67909 | Accuracy = 70.6836 | Train-f1 = 0.6156 | Valid-F1 = 0.6912
Epoch 1: Train Loss = 0.65779 | Validation Loss = 0.64380 | Accuracy = 73.0500 | Train-f1 = 0.7125 | Valid-F1 = 0.7257
Epoch 2: Train Loss = 0.59623 | Validation Loss = 0.62333 | Accuracy = 73.7949 | Train-f1 = 0.7523 | Valid-F1 = 0.7377

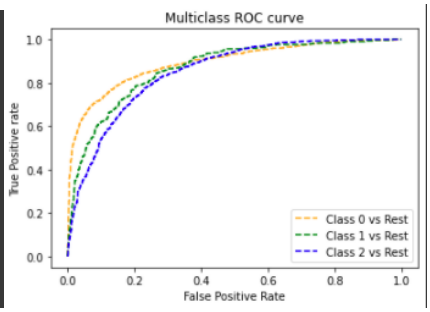
- Τελικά αποτελέσματα μετρικών




```
Accuracy: 73.79%
f1 score: 73.77%
Precision: 73.94%
Recall: 73.79%
              precision    recall  f1-score   support

           0       0.82      0.78      0.80      1065
           1       0.58      0.52      0.55       296
           2       0.69      0.76      0.72       921

    accuracy                           0.74      2282
   macro avg       0.70      0.69      0.69      2282
weighted avg       0.74      0.74      0.74      2282
```



Παρατηρούμε:

- Τα scores των μετρικών έχουν αυξηθεί και είναι τα καλύτερα έως τώρα και στις δύο μελέτες.
- Παρατηρούμε μια μικρή απόκλιση μεταξύ του train και του validation loss. Ωστόσο λόγω της μικρής διαφοράς δεν θεωρείται ότι το μοντέλο παρουσιάζει overfit.

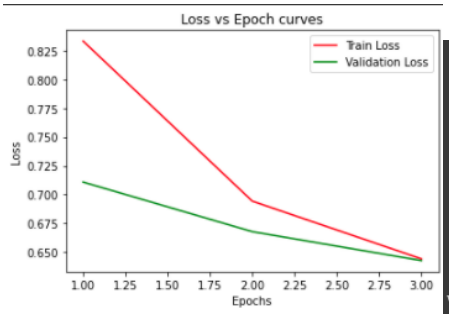
Τέλος, δοκιμάζω να τροποποιήσω το learning_rate, θέτοντας το σε 3e-6.

Τα αποτελέσματα είναι τα εξής:

- Πληροφορίες ανά epoch

```
Epoch   0: Train Loss = 0.83336 | Validation Loss = 0.71087 | Accuracy = 68.4487 | Train-f1 = 0.5982 | Valid-F1 = 0.6506
Epoch   1: Train Loss = 0.69432 | Validation Loss = 0.66781 | Accuracy = 71.4724 | Train-f1 = 0.6784 | Valid-F1 = 0.7090
Epoch   2: Train Loss = 0.64416 | Validation Loss = 0.64267 | Accuracy = 72.8309 | Train-f1 = 0.7211 | Valid-F1 = 0.7289
```

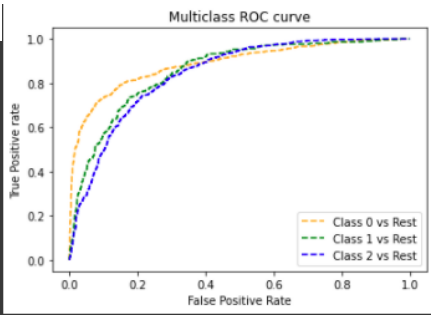
- Τελικά αποτελέσματα μετρικών




```
Accuracy: 72.83%
f1 score: 72.89%
Precision: 73.43%
Recall: 72.83%
              precision    recall  f1-score   support

           0       0.84      0.76      0.80      1065
           1       0.51      0.47      0.49       296
           2       0.68      0.77      0.72       921

    accuracy                           0.73      2282
   macro avg       0.68      0.67      0.67      2282
weighted avg       0.73      0.73      0.73      2282
```



Παρατηρούμε:

- Τα scores των μετρικών έχουν μειωθεί ελάχιστα σε σχέση με το τελευταίο βέλτιστο μοντέλο.
- Το μοντέλο δεν παρουσιάζει καθόλου overfit καθώς τα train και validation loss τελικά ταυτίζονται.

Δοκιμάζοντας και άλλες τιμές για τις παραμέτρους δεν κατάφερα να βελτιώσω περισσότερο το μοντέλο.

Τελικά σαν βέλτιστο μοντέλο επιλέγω το προ-τελευταίο, το οποίο πέτυχε και τα υψηλότερα scores.
Παρόλο που υπάρχει μια μικρή απόκλιση μεταξύ train και validation loss δεν θεωρείται πως παρουσιάζει overfit. Το τελευταίο μοντέλο που έχει καλύτερη συμπεριφορά όσον αφορά τα loss curves ωστόσο υστερεί όσον αφορά τα scores των μετρικών.
Έτσι επιλέγω το πρώτο από τα δύο.

1.10.2.1 Βέλτιστο μοντέλο BertForSequenceClassification

Με βάση την παραπάνω μελέτη τα καλύτερα αποτελέσματα τα παίρνουμε ως εξής:

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.

Στην συνέχεια εφαρμόζουμε tokenization χρησιμοποιώντας τον:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

και ορίζοντας την παράμετρο max_length ίση με 150.
Κατά το tokenization δεν «χάνεται» καθόλου πληροφορία καθώς το μέγιστο μήκος από tokens που προκύπτει είναι 140.

Όσον αφορά το μοντέλο χρησιμοποιώ το BertForSequenceClassification (bert-base-uncased).
Ο ορισμός του συνολικού μοντέλου γίνεται ως εξής:

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=3)
```

Οι υπερπαραμέτροι που χρησιμοποίησα για το fine tuning είναι:

- TRAINING_EPOCHS = 3
- batch_size = 32
- learning_rate = 5e-6

Επίσης κατά την εκπαίδευση εφάρμοσα gradient clipping μέσω της συνάρτησης:

```
nn.utils.clip_grad_norm_
```

Το Loss Function που χρησιμοποίησα είναι το nn.CrossEntropyLoss().
Ο optimizer που χρησιμοποίησα ήταν ο AdamW.

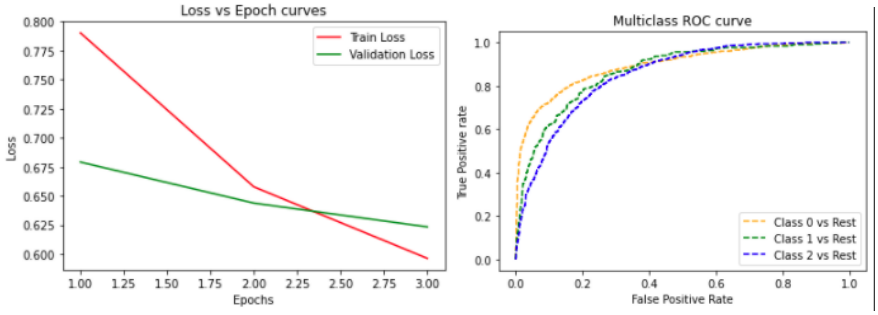
1.10.2.2 Αξιολόγηση βέλτιστου μοντέλου BertForSequenceClassification

Έπειτα από το fine tuning του παραπάνω μοντέλου τα αποτελέσματα των μετρικών είναι τα εξής:

```
Accuracy: 73.79%
f1 score: 73.77%
Precision: 73.94%
Recall: 73.79%
              precision    recall  f1-score   support

           0       0.82      0.78      0.80      1065
           1       0.58      0.52      0.55       296
           2       0.69      0.76      0.72       921

    accuracy                           0.74      2282
   macro avg       0.70      0.69      0.69      2282
weighted avg       0.74      0.74      0.74      2282
```



Συμπεράσματα:

- Το μοντέλο μας έχει πολύ καλά scores (τα καλύτερα που παρουσιάστηκαν σε ολόκληρη την μελέτη) με βάση τις μετρικές που παρουσιάζονται παραπάνω. Υστερεί ελάχιστα 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.
- Παρατηρούμε μια μικρή απόκλιση μεταξύ train και validation loss ωστόσο δεν θεωρείται πως παρουσιάζει overfit καθώς οι τιμές είναι αρκετά κοντά.
- Παρατηρούμε ότι για τα πρώτα epochs το μοντέλο τα πηγαίνει καλύτερα στο validation set από το train set, δηλαδή παρουσιάζει underfit. 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.

1.11 Βέλτιστο Μοντέλο

Συγκρίνοντας τα δύο βέλτιστα μοντέλα που προέκυψαν χρησιμοποιώντας το BertModel και το BertForSequenceClassification παρουσιάστηκαν στις παραγράφους 1.10.1.1 Βέλτιστο μοντέλο BertModel και 1.10.2.1 Βέλτιστο μοντέλο BertForSequenceClassification αντίστοιχα, το καλύτερο από τα δύο είναι αυτό που προέκυψε με το BertForSequenceClassification.
Παρατηρούμε ότι υπερτερεί σε σχέση με το πρώτο όσον αφορά τα scores όλων των μετρικών.
Όσον αφορά τα loss curves και στα δύο μοντέλα παρατηρούμε παρόμοια εικόνα, με μια μικρή απόκλιση μεταξύ train και validation loss, χωρίς όμως να θεωρείται overfit.

Έτσι το βέλτιστο μοντέλο για την άσκηση 1 (το οποίο παρουσιάζεται και στο παραδοτέο αρχείο .ipynb) είναι αυτό που παρουσιάστηκε στην παράγραφο 1.10.2.1 Βέλτιστο μοντέλο BertForSequenceClassification.

1.12 Αξιολόγηση Βέλτιστου Μοντέλου

Η αξιολόγηση του βέλτιστου μοντέλου παρουσιάστηκε στην παράγραφο 1.10.2.2 Αξιολόγηση βέλτιστου μοντέλου BertForSequenceClassification.

1.13 Σύγκριση με τα μοντέλα των προηγούμενων εργασιών

	F1 score	Precision	Recall	Accuracy
Project 1 – Softmax Regression	72.79%	73.75%	73.14%	73.14%
Project 2 – Feed-Forward NN	70.02%	70.05%	70.11%	70.11%
Project 3 – LSTM bidirectional Stacked RNN	68.89%	69.12%	69.37%	69.37%
Project 3 – GRU bidirectional Stacked RNN	69.33%	69.58%	69.72%	69.72%
Project 3 – GRU bidirectional Stacked RNN with attention	69.60%	70.07%	70.16%	70.16%
Project 4 – Finetuned BertForSequenceClassification	73.77%	73.94%	73.79%	73.79%

Με βάση τα scores των μετρικών που παρουσιάζονται στον παραπάνω πίνακα συμπεραίνουμε πως το βέλτιστο μοντέλο αυτής της εργασίας έχει την καλύτερη απόδοση από όλα.
Όσον αφορά τα μοντέλα των εργασιών 2 και 3 παρατηρούμε ότι η διαφορά είναι μεγαλύτερη (περίπου 2-3%), δηλαδή το μοντέλο υπερτερεί σαφώς.
Όσον αφορά το μοντέλο της εργασίας 1 (το οποίο είναι και το πιο «απλό» μοντέλο από όλα) παρατηρούμε μικρή διαφορά στα scores των μετρικών (περίπου 1%).
Λαμβάνοντας υπόψιν την τεράστια διαφορά στην πολυπλοκότητα καθώς και στην εκπαίδευση των μοντέλων της εργασίας 1 και της εργασίας 4 καθώς και της μικρής διαφοράς των scores των μετρικών συμπεραίνουμε πως η σημαντική αύξηση της πολυπλοκότητας του μοντέλου δεν έχει ως αποτέλεσμα σημαντική αύξηση στην απόδοση του μοντέλου για το δοθέν dataset.
Αυτό́ οφείλεται στην απλότητα του προβλήματος και του μικρού́ όγκου των δεδομένων μας. Ως γνωστών τα feed-forward νευρωνικά δίκτυα (Εργασία 2), τα bidirectional stacked recursive νευρωνικά δίκτυα (Εργασία 3) και πόσο μάλλον ένα Deep Bidirectional Transformer μοντέλο όπως το Bert 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, μοντέλα αυτά δεν βελτίωσαν σημαντικά την απόδοση σε σχέση με το «απλό» μοντέλο του Softmax Regression.

Μέρος Β – Question Answering on SQuAD 2.0

Στην πρώτη άσκηση της εργασίας θα αναπτύξω έναν μοντέλο question answering για το dataset SQuAD2.0. Το μοντέλο θα προκύψει έπειτα από fine-tuning του pretrained BERT-base μοντέλου πάνω στο train set του SQuAD dataset.
Πιο συγκεκριμένα, θα ακολουθήσει η μελέτη μέσω της οποίας κατέληξα στο τελικό βέλτιστο μοντέλο. Στην μελέτη αυτή πειραματίστηκα κυρίως με:

- Πειραματισμός με το tokenization
- Χρήση διαφορετικών παραλλαγών του μοντέλου BERT (BertForQuestionAnswering, DistilBertForQuestionAnswering)
- Πειραματισμός με τις υπερπαραμέτρους για το fine tuning (batch_size, learning_rate, epochs)

2.1 Μεθοδολογία

Συνοπτικά, η μεθοδολογία και τα βήματα που ακολούθησα για την παραγωγή και αξιολόγηση του μοντέλου είναι:

- Λήψη των αρχείων train και dev set του SQuAD dataset (μέσω της εντολής wget).
- Ανάγνωση των αρχείων μέσω της συνάρτησης read_file και αποθήκευση των αντίστοιχων δεδομένων σε αντίστοιχες δομές. Εξηγείται αναλυτικά στην παράγραφο 2.2 Προετοιμασία Δεδομένων.
- Ορισμός του tokenizer που θα χρησιμοποιηθεί και tokenization των δεδομένων τόσο του train set όσο και του dev set. Εξηγείται αναλυτικά στην παράγραφο 2.3 Tokenization.
- Ορισμός του μοντέλου Bert που θα χρησιμοποιηθεί, ορισμός του optimizer και ορισμός των υπερπαραμέτρων για το fine tuning του μοντέλου.
- Δημιουργία των DataLoaders για τον διαχωρισμό των δεδομένων train/dev σε batches.
- Εκπαίδευση του μοντέλου για προκαθορισμένο αριθμό epochs. Σε κάθε epoch «δοκιμάζουμε» το μοντέλο μας και για το validation set υπολογίζοντας το validation loss και validation accuracy τα οποία και εκτυπώνονται για κάθε ένα epoch μαζί με το train loss και το train accuracy.
- Δοκιμή του εκπαιδευμένου μοντέλου στο validation (dev) set και αξιολόγηση του question answering με βάση τις απαντήσεις που έδωσε το μοντέλο μας.
  Η αξιολόγηση γίνεται χρησιμοποιώντας το evaluation script που παρέχεται από τους δημιουργούς του SQuAD (Εξηγείται αναλυτικά στην παράγραφο 2.9 Αξιολόγηση του μοντέλου).

2.2 Προετοιμασία Δεδομένων

Αρχικά μέσω της εντολής wget γίνεται λήψη (απευθείας στο notebook) των δύο αρχείων train και dev set. Στην εντολή δίνουμε τα αντίστοιχα links από το site του hugging face. Πιο συγκεκριμένα η λήψη των δύο αρχείων γίνεται ως εξής:

```
!wget https://rajpurkar.github.io/SQuAD-explorer/dataset/train-v2.0.json
!wget https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v2.0.json
```

Έχοντας κατεβάσει τα αρχεία, σειρά έχει η ανάγνωση τους και η αποθήκευση των δεδομένων σε αντίστοιχες δομές.
Η διαδικασία αυτή πραγματοποιείται από την συνάρτηση **read_file**.
Μια ερώτηση μπορεί είτε να έχει μία/πολλές απαντήσεις είτε να μην έχει απάντηση (impossible). Στόχος μας είναι να δημιουργήσουμε ένα σετ ερωτήσεων-απαντήσεων.
Για κάθε ερώτηση αποθηκεύω τις εξής πληροφορίες:

- Question: η ερώτηση.
- Context: το κείμενο μέσα από το οποίο προκύπτει η απάντηση στην ερώτηση.
- Answer: η πραγματική απάντηση της ερώτησης.
- Impossible: (True/False) ανάλογα με το αν υπάρχει απάντηση στην ερώτηση εσωτερικά του αντίστοιχου context.
- Answer Start: το index του string του context στο οποίο ξεκινά η απάντηση. Η πληροφορία αυτή υπάρχει για κάθε απάντηση στα αρχεία του SQuAD.
- Answer End: το index του string του context στο οποίο τελειώνει η απάντηση. Η πληροφορία αυτή δεν υπάρχει στα αρχεία του SQuAD και έτσι την υπολογίζω εσωτερικά της συνάρτησης κατά την δημιουργία των δεδομένων. Ο τρόπος υπολογισμού του answer_end index εξηγείται στην συνέχεια.

Επίσης ισχύουν τα εξής:

- Σε ένα context αντιστοιχούν πολλές ερωτήσεις.
- Σε περίπτωση που μια ερώτηση έχει παραπάνω από μια απαντήσεις, τότε εισάγω μια «εγγραφή» στα δεδομένα για κάθε μία απάντηση. Για παράδειγμα, αν μια ερώτηση έχει 3 σωστές απαντήσεις, τότε εισάγω στα δεδομένα μου 3 ξεχωριστές φορές την ερώτηση όπου σε κάθε εισαγωγή αντιστοιχεί και μία από τις σωστές απαντήσεις.
- Σε περίπτωση που μια ερώτηση δεν έχει απάντηση (impossible), τότε στο πεδίο «Answer» εισάγω την κενή συμβολοσειρά και στα πεδία answer_start και answer_end εισάγω την τιμή 0.

Η συνάρτηση read_file δημιουργεί 4 λίστες σαν output οι οποίες:

- Context: περιέχει το context που αντιστοιχεί στην κάθε ερώτηση. Το context στην θέση i της λίστας αντιστοιχεί στην ερώτηση στην θέση i της λίστας με τα Questions.
- Question: περιέχει το σύνολο των ερωτήσεων.
- Answer: περιέχει την απάντηση που αντιστοιχεί στην κάθε ερώτηση. Η απάντηση στην θέση i της λίστας αντιστοιχεί στην ερώτηση στην θέση i της λίστας με τα Questions. Κάθε στοιχείο της λίστας είναι ένα dictionary όπου περιέχει τα εξής:
    - text: την ερώτηση σε μορφή κειμένου.
    - answer_start: το index του string του context από οποίο ξεκινά η απάντηση.
    - answer_end: το index του string του context από οποίο τελειώνει η απάντηση.
- Impossible: περιέχει την πληροφορία για το κατά πόσο η αντίστοιχη ερώτηση έχει απάντηση.

Όσον αφορά το answer_end index ο υπολογισμός του γίνεται ως εξής:

answer_end = answer_start + len(answer)

Δηλαδή στο index answer_start προτίθεται το μήκος της απάντησης.
Ωστόσο σε ορισμένες περιπτώσεις το answer_start που παρέχεται από τα αρχεία του SQuAD απέχει ελάχιστα από το πραγματικό (συνήθως 1-2 index). Έτσι μέσω ενός while loop ελέγχεται αυτή η ενδεχόμενη απόκλιση και διορθώνεται.

Τέλος, κάποιες σημαντικές παρατηρήσεις:

- Η επιλογή να εισάγω στα δεδομένα μου πολλαπλές φορές την ίδια ερώτηση, κάθε φορά με μία από τις πιθανές σωστές απαντήσεις έγινε έτσι ώστε το μοντέλο να έχει περισσότερα διαφορετικά δεδομένα για εκπαίδευση και κατ' επέκταση να εκπαιδευτεί καλύτερα στην εύρεση σωστών απαντήσεων.
- Όπως αναφέρθηκε και παραπάνω στα δεδομένα μου συμπεριλαμβάνω και τις «impossible» ερωτήσεις θεωρώντας σαν σωστή απάντηση την κενή συμβολοσειρά. Έτσι το μοντέλο εκπαιδεύεται στο να αναγνωρίζει το κατά πόσο για μια ερώτηση δεν υπάρχει απάντηση εσωτερικά του δοθέντος context και να επιστρέφει την κενή συμβολοσειρά.

2.3 Tokenization

Για το tokenization των δεδομένων χρησιμοποιώ τον BertTokenizerFast και συγκεκριμένα το μοντέλο bert-base-uncased.
Ο ορισμός του tokenizer γίνεται ως εξής:

```
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
```

Χρησιμοποιώ τον BertTokenizerFast αντί του BertTokenizer καθώς:

- Είναι πολύ γρηγορότερος και επιτρέπει το tokenization μεγαλύτερου όγκου δεδομένων με μεγαλύτερο μήκος από tokens (max_length). Η χρήση του BertTokenizer για το tokenization μεγάλου όγκου δεδομένων με μεγάλο max_length είναι αδύνατη στις πλατφόρμες εκτέλεσης του κώδικα (GoogleColab και Kaggle) καθώς είναι πολύ απαιτητικός όσον αφορά την χρήση μνήμης και δεν το επιτρέπουν τα όρια των πλατφορμών.
- Δίνει την δυνατότητα εύρεση του αντίστοιχου index του token δίνοντας του το αντίστοιχο index της συμβολοσειράς. Πιο συγκεκριμένα μέσω της char_to_token μέσω του index answer_start μπορούμε να βρούμε το αντίστοιχο index στο tokenized text, κάτι που είναι απαραίτητο καθώς πρέπει να δοθεί σαν είσοδος στο μοντέλο για την εκπαίδευση. Περισσότερες λεπτομέρειες εξηγούνται στην συνέχεια.

Κατά το tokenization, ο tokenizer δέχεται σαν είσοδο τα contexts και τα questions. Για κάθε ένα σετ context/question δημιουργεί την αντίστοιχη αναπαράσταση. Ο tokenizer προσθέτει αυτόματα τα απαραίτητα tokens, δηλαδή το [CLS] token στην αρχή και στο τέλος και το [SEP] token ενδιάμεσα από το context και το question.
Ο tokenizer επιστέφει τα input_ids, δηλαδή τα ids των tokens της tokenized αναπαράστασης καθώς και το attention_mask.

Όπως παρουσιάζεται και στην μελέτη που ακολουθεί, πειραματίστηκα αρκετά με την παράμετρο max_length κατά το tokenization.
Η παράμετρος max_length ορίζει το μέγιστο μήκος της tokenized αναπαράστασης που μπορεί να προκύψει, δηλαδή το μέγιστο μήκος από tokens.

Μικρή τιμή του max_length έχει ως αποτέλεσμα μικρότερες αναπαραστάσεις και κατ΄ επέκταση μείωση του χρόνου εκπαίδευσης. Ωστόσο, πολλές φορές έχει ως αποτέλεσμα απώλεια πληροφορίας καθώς μπορεί να αποκοπεί περιεχόμενο, κάτι που μπορεί να έχει επίπτωση στην απόδοση του μοντέλου.
Αντίθετα μεγάλη τιμή για το max_length έχει ως αποτέλεσμα όσο το δυνατόν λιγότερη απώλεια πληροφορίας, αλλά ταυτόχρονα αύξηση του χρόνου εκπαίδευσης.

Έπειτα από το tokenization πρέπει να μετατρέψουμε τα index των απαντήσεων (answer_start και answer_end) στα αντίστοιχα indexes των token.
Η διαδικασία αυτή υλοποιείται από την συνάρτηση **index_answer**.
Πιο συγκεκριμένα για κάθε σετ ερώτησης απάντησης:

- Αν η ερώτηση δεν έχει απάντηση ορίζω σαν answer_start και answer_end το index του πρώτου [CLS] token (δηλαδή το index 0). Επίσης, δοκίμασα να ορίζω σαν start και end index το index του [SEP] token, ωστόσο η απόδοση ήταν αντίστοιχη. Περισσότερες λεπτομέρειες σχετικά με αυτήν την επιλογή εξηγούνται στην παράγραφο 2.10.1 Διαχείριση ερωτήσεων χωρίς απάντηση.
- Αν η ερώτηση έχει απάντηση, τότε μέσω της συνάρτησης char_to_token και δίνοντας της σαν όρισμα τα answer_start και answer_end (που αντιστοιχούν στα indexes του context πριν το tokenization) εντοπίζουμε τα αντίστοιχα indexes των tokens.
  Επίσης:
  - Το answer_start index του token μπορεί να μην βρεθεί καθώς λόγω του περιορισμένου μήκους max_length του tokenization μπορεί να έχει αποκοπεί. Στην περίπτωση αυτή ορίζουμε σαν answer_start το μέγιστο μήκος του max_length (512), καθώς η απάντηση δεν περιέχεται στην tokenized αναπαράσταση.
  - Στην περίπτωση που δεν βρεθεί answer_end index του token μειώνουμε σταδιακά το answer_end έως ότου βρεθεί ένα token. Έτσι συμπεριλαμβάνουμε και την περίπτωση που η απάντηση έχει κοπεί στην μέση και την περίπτωση που το answer_end «δείχνει» σε κενό χαρακτήρα.

Τελικά έχουμε για κάθε σετ context/question τα αντίστοιχα start και end token indexes σε μορφή δύο λιστών οι οποίες και επιστρέφονται από την συνάρτηση index_answer. Τα στοιχεία των δύο αυτών λιστών αντιστοιχούν στα στοιχεία των λιστών που επιστράφηκαν από την συνάρτηση read_file.

Ολόκληρη η παραπάνω διαδικασία εφαρμόζεται τόσο για το train set όσο και για το validation/dev set.

2.4 Batching

Όπως αναφέρθηκε και παραπάνω, ο tokenizer για κάθε ένα σετ context-question επιστρέφει τα input_ids και τα attention_masks.
Επίσης, για κάθε ένα σετ context-question έχουμε δημιουργήσει τα start και end indexes των tokens των απαντήσεων.
Αυτές είναι και οι πληροφορίες που θα χρησιμοποιηθούν για την εκπαίδευση, δηλαδή για κάθε σετ context-question έχουμε:

- Input_ids
- Attention_mask
- Start_index
- End_index

Μετατρέπουμε τα παραπάνω σε tensors και στην συνέχεια δημιουργούμε δύο DataLoaders έναν για το train set και έναν για το validation set αντίστοιχα. Λόγω του padding που εφαρμόζεται κατά το tokenization από τον tokenizer όλα τα tokenized tweets έχουν ίδιο μήκος και έτσι μπορούμε εύκολα να κατασκευάσουμε τους Dataloaders.
Έτσι τα δεδομένα μας χωρίζονται σε batches έτσι ώστε να γίνει η εκπαίδευση (fine tuning).

2.5 Ορισμός του Μοντέλου

Το μοντέλο που θα χρησιμοποιήσω είναι το BertForQuestionAnswering και συγκεκριμένα το bert-base-uncased.
Όπως φαίνεται και από το όνομα του, το μοντέλο αυτό είναι κατασκευασμένο για question answering. Έτσι, δέχεται σαν είσοδο τα 4 δεδομένα που αναφέρθηκαν στην παράγραφο batching (input_ids, attention_mask, start_index, end_index) και υπολογίζει εσωτερικά το loss κατά την εκπαίδευση.
Κατά την διάρκεια του evaluation, δέχεται σαν όρισμα τα input_ids και το attention mask και επιστρέφει τις προβλέψεις του για το start και end index.
Ο ορισμός του μοντέλου γίνεται ως εξής:

```
model = BertForQuestionAnswering.from_pretrained("bert-base-uncased")
```

Επίσης δοκίμασα να χρησιμοποιήσω και το μοντέλο DistilBertForQuestionAnswering λόγω του σημαντικά μικρότερου χρόνου που χρειάζεται για το fine tuning. Ωστόσο, η απόδοση του ήταν σε κάθε περίπτωση χειρότερη από αυτήν του BertForQuestionAnswering και έτσι δεν παρουσιάζεται στην μελέτη που ακολουθεί.

2.6 Loss Function

Το μοντέλο BertForQuestionAnswering υπολογίζει εσωτερικά το loss, επομένως δεν χρειάζεται ο ορισμός ενός loss function.

2.7 Optimizer

Ο optimizer που χρησιμοποιώ είναι ο AdamW.
Δοκίμασα να χρησιμοποιήσω και τον κλασσικό Adam, με τον οποίο παρατήρησα παρόμοια (ελάχιστα χειρότερη) απόδοση.

2.8 Εκπαίδευση μοντέλου (Fine Tuning)

Έχοντας ορίσει τον επιθυμητό αριθμό epochs, το fine tuning 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 context/question.
    - Στο output περιέχεται το loss το οποίο υπολογίστηκε εσωτερικά του μοντέλου.
    - Εφαρμόζεται backpropagation με το loss που υπολογίστηκε.
    - Εφαρμόζεται gradient clipping.
    - Μέσω του optimizer ανανεώνουμε τα weights του μοντέλου με βάση τα gradients που προέκυψαν από το backpropagation
    - Τέλος, υπολογίζω το accuracy των προβλέψεων. Το accuracy υπολογίζεται συγκρίνοντας τα start και end logits που επιστράφηκαν από το μοντέλο, συγκρίνοντας τα με αυτά που του δόθηκαν σαν όρισμα (δηλαδή σαν σωστή απάντηση).
- 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 context/question.
    - Αποθηκεύουμε το loss το οποίο υπολογίστηκε εσωτερικά του μοντέλου έτσι ώστε να παρατηρήσουμε την πορεία του..
    - Αντίστοιχα υπολογίζω το accuracy των προβλέψεων του validation set έτσι ώστε να παρακολουθήσω την απόδοση του μοντέλου και στο validation set.
- Τέλος, υπολογίζουμε και εκτυπώνουμε για το τρέχον epoch τα εξής:
    - Train Loss
    - Validation Loss
    - Train Accuracy
    - Validation Accuracy

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

Παρατήρηση: Η μετρική accuracy που υπολογίζεται κατά την εκπαίδευση του μοντέλου είναι αρκετά σχετική και δεν αποτελεί στιβαρό κριτήριο για την αξιολόγηση του μοντέλου. Όπως αναφέρθηκε συγκρίνονται τα start/end logits που επιστρέφει το μοντέλο σε σχέση με αυτά που δέχθηκε σαν όρισμα. Ωστόσο, για παράδειγμα λόγω του περιορισμένου μήκους του input (max_length) μπορεί η απάντηση ενός context να έχει περικοπεί. Έτσι, τα start και end index “δείχνουν” στο τέλος του input (και όχι στην πραγματική απάντηση). Σε περίπτωση που το μοντέλο “προβλέψει” αυτά τα start/end logits τότε το accuracy θα βγει 100%. Όμως, είναι προφανές ότι το μοντέλο δεν έχει εντοπίσει την απάντηση, καθώς αυτή δεν περιέχεται στο context λόγω της αποκοπής. Παρόλα αυτά, την χρησιμοποιώ σαν μια επιπλέον εκτίμηση της απόδοσης κατά την εκπαίδευση.

2.9 Αξιολόγηση του μοντέλου

Έπειτα από το fine tuning 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 μια ερώτηση, δηλαδή επιστέφει τα start και end logits.
Η διαδικασία αυτή γίνεται τόσο χρησιμοποιώντας τις ερωτήσεις που έχουν απάντηση όσο και τις ερωτήσεις που δεν έχουν απάντηση (impossible). Για της impossible ερωτήσεις, σωστή απάντηση θεωρείται η κενή συμβολοσειρά.

Για την αξιολόγηση των προβλέψεων χρησιμοποιώ το evaluation script που υπάρχει στην επίσημη σελίδα του SQuAD.
Το evaluation script είναι το εξής:
https://worksheets.codalab.org/rest/bundles/0x6b567e1cf2e041ec80d7098f031c5c9e/contents/blob/

Για την χρήση του εκτελώ τα εξής:

- Μέσω της βιβλιοθήκης urllib.request «διαβάζω» τον κώδικα που περιέχεται στο παραπάνω URL.
- Δημιουργώ ένα αρχείο (evaluation.py) και «γράφω» τον κώδικα που διαβάστηκε στο αρχείο αυτό.
- Έτσι, έχει δημιουργηθεί ένα python αρχείο που περιέχει το evaluation script. Επομένως, για την χρήση του αρκεί να εκτελεστεί δίνοντας του τα κατάλληλα ορίσματα.

Το evaluation script δέχεται τα εξής δύο ορίσματα:

- Το dev αρχείο του SQuAD dataset.
- Ένα αρχείο το οποίο περιέχει ένα «εκτυπωμένο» dictionary με τις απαντήσεις που προέκυψαν από το μοντέλο για κάθε μία ερώτηση.

Έπειτα από την εκτέλεση του, το evaluation script επιστρέφει:

- Exact : το συνολικό Exact Match score όλων των ερωτήσεων του validation set.
- F1 : το συνολικό F1 score όλων των ερωτήσεων του validation set.
- HasAns_exact : το Exact Match score των ερωτήσεων του validation set που έχουν απάντηση.
- HasAns_f1 : το F1 score των ερωτήσεων του validation set που έχουν απάντηση.
- HasAns_total : ο αριθμός των ερωτήσεων που δόθηκαν και έχουν απάντηση.
- NoAns_exact : το Exact Match score των ερωτήσεων του validation set που δεν έχουν απάντηση.
- NoAns_f1 : το F1 score των ερωτήσεων του validation set που δεν έχουν απάντηση.
- NoAns_total : ο αριθμός των ερωτήσεων που δόθηκαν και έχουν απάντηση.

Συνολικά, η διαδικασία που ακολουθείται για το evaluation του μοντέλου είναι:

- Ανάγνωση του αρχείου dev και αποθήκευση όλων των ερωτήσεων, του context που αντιστοιχεί σε κάθε ερώτηση και το id της κάθε ερώτησης.
- Tokenization των contexts/questions. Κατά το tokenization δεν επιβάλεται κάποιος περιορισμός όσον αφορά το max_length.

- Δημιουργία των αντίστοιχων tensors και στην συνέχεια δημιουργία DataLoaders έτσι ώστε τα δεδομένα να χωριστούν σε batches για την επιτάχυνση της διαδικασίας των προβλέψεων (δηλαδή να μην δίνονται μια προς μια οι ερωτήσεις στο μοντέλο).
- Για κάθε μια ερώτηση το μοντέλο επιστρέφει τα αντίστοιχα start/end logits. Αρχικά καλώ την συνάρτηση convert_ids_to_tokens για να εντοπίσω τα tokens τα οποία ανήκουν στο διάστημα μεταξύ των start end logits. Στην συνέχεια μέσω της συνάρτησης convert_tokens_to_string μετατρέπω τα tokens αυτά στο αντίστοιχο string. Έτσι προκύπτει η τελική απάντηση του μοντέλου για την κάθε ερώτηση. Χρησιμοποιώντας το id της κάθε ερώτησης (σαν κλειδί) αποθηκεύω τις απαντήσεις για κάθε ερώτηση σε ένα dictionary.
- Έτσι τελικά έχει προκύψει ένα dictionary όπου σαν κλειδιά έχει τα ids όλων των ερωτήσεων του dev set και σαν τιμές έχει την απάντηση που έδωσε το μοντέλο για κάθε μια ερώτηση.
- «Γράφω» το dictionary σε ένα αρχείο με όνομα results.txt
- Δημιουργώ το evaluation script (όπως εξηγήθηκε παραπάνω) και εκτελώ το script δίνοντας του σαν όρισμα το αρχείο dev και το αρχείο results.txt.
- Τέλος, μέσω του evaluation script εκτυπώνονται τα scores των μετρικών.

2.10 Μελέτη για την εύρεση του βέλτιστου μοντέλου

Όπως έχει αναφερθεί και παραπάνω στις αντίστοιχες παραγράφους, στην μελέτη που ακολουθεί χρησιμοποιώ το μοντέλο BertForQuestionAnswering και τον tokenizer BertTokenizerFast.
Οι εκτελέσεις που παρουσιάζονται παρακάτω έγιναν μέσω της πλατφόρμας Kaggle. Κάποιες από αυτές (κυρίως προς το τέλος της μελέτης) ενδεχομένως να μην μπορούν να πραγματοποιηθούν μέσω της πλατφόρμας Google Colab λόγω των μικρότερων περιορισμών όσον αφορά την χρήση υπολογιστικών πόρων (π.χ. χρήση μνήμης).

Αρχικός στόχος μου είναι να πειραματιστώ με διάφορες τιμές των υπερπαραμέτρων και να παρατηρήσω την απόδοση του μοντέλου.
Επίσης, για να μειώσω περαιτέρω τον χρόνο εκτέλεσης κατά τις πρώτες πειραματικές εκτελέσεις, αρχικά:

- Μείωσα την παράμετρο max_length του tokenizer έτσι ώστε να προκύπτουν μικρότερες αναπαραστάσεις.
- Μείωσα τον όγκο των δεδομένων κρατώντας κάθε φορά μόνο sets context/question για τα οποία ίσχυε:
  πλήθος λέξεων context + πλήθος λέξεων question < max_length

Όπως παρουσιάζεται και στην συνέχεια, σταδιακά αυξάνω το max_length καθώς και το πλήθος των δεδομένων για να προκύψει το τελικό μοντέλο.

Οι υπερπαράμετροι με τις οποίες πειραματίστηκα είναι:

- Number of epochs
- Learning rate
- Batch size

Και όπως ήδη αναφέρθηκε: max_length και πλήθος δεδομένων

Στην συνέχεια παρουσιάζω κάποιες από τις αρχικές πειραματικές εκτελέσεις μαζί με τα αντίστοιχα αποτελέσματα. Στα αποτελέσματα παρουσιάζω τα στατιστικά που προέκυψαν για κάθε ένα epoch (loss και accuracy) και τα αποτελέσματα από το evaluation script.

Επίσης, αρχικά χρησιμοποίησα μόνο τις ερωτήσεις οι οποίες είχαν απάντηση, δηλαδή στα δεδομένα δεν περιλαμβάνονταν τα impossible questions.

Εκτέλεση 1

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 150
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 150
- learning_rate = 1e-1
- batch_size = 64
- impossible questions: δεν συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 5.17534 | Validation Loss = 5.01064 | Train Accuracy = 0.0065 | Validation Accuracy = 0.0088
Epoch 1: Train Loss = 5.15425 | Validation Loss = 5.01064 | Train Accuracy = 0.0066 | Validation Accuracy = 0.0077

Evaluation:

- "exact": 50.07159100480081,
- "f1": 50.07159100480081,
- "total": 11873,
- "HasAns_exact": 0.0,
- "HasAns_f1": 0.0,
- "HasAns_total": 5928,
- "NoAns_exact": 100.0,
- "NoAns_f1": 100.0,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πολύ μεγάλες τιμές για το loss κατά την εκπαίδευση το οποίο σχεδόν παραμένει σταθερό και στα 2 epochs.
- Τόσο από τα scores των μετρικών όσο και έπειτα από έλεγχο των απαντήσεων του μοντέλου, φαίνεται πως το μοντέλο δίνει κάθε φορά σαν απάντηση την κενή συμβολοσειρά.
- Συμπεραίνουμε ότι το learning rate έχει υπερβολικά μεγάλη τιμή και το μοντέλο αδυνατεί να εκπαιδευτεί.

Εκτέλεση 2

Μειώνω το learning rate. Τα υπόλοιπα παραμένουν ως έχουν.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 150
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 150
- learning_rate = 1e-2
- batch_size = 64
- impossible questions: δεν συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 5.02252 | Validation Loss = 5.01064 | Train Accuracy = 0.0064 | Validation Accuracy = 0.0068
Epoch 1: Train Loss = 5.01568 | Validation Loss = 5.01064 | Train Accuracy = 0.0065 | Validation Accuracy = 0.0060

Evaluation:

- "exact": 49.869451697127936,
- "f1": 49.869451697127936,
- "total": 11873,
- "HasAns_exact": 0.0,
- "HasAns_f1": 0.0,
- "HasAns_total": 5928,
- "NoAns_exact": 99.59629941126998,
- "NoAns_f1": 99.59629941126998,
- "NoAns_total": 5945

Παρατηρήσεις:

- Τα αποτελέσματα είναι αντίστοιχα με πριν. Το learning rate εξακολουθεί να είναι υπερβολικά μεγάλο.

Εκτέλεση 3

Μειώνω ακόμη περισσότερο το learning rate. Επίσης μειώνω και το μήκος του max_length. Τα υπόλοιπα παραμένουν ως έχουν.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 100
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 100
- learning_rate = 5e-3
- batch_size = 64
- impossible questions: δεν συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 4.62334 | Validation Loss = 4.60517 | Train Accuracy = 0.0088 | Validation Accuracy = 0.0088
Epoch 1: Train Loss = 4.61006 | Validation Loss = 4.60517 | Train Accuracy = 0.0089 | Validation Accuracy = 0.0098

Evaluation:

- "exact": 29.857660237513688,
- "f1": 31.338494611798513,
- "total": 11873,
- "HasAns_exact": 0.0,
- "HasAns_f1": 2.9659154058511152,
- "HasAns_total": 5928,
- "NoAns_exact": 59.62994112699748,
- "NoAns_f1": 59.62994112699748,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πως αυξήθηκε ελάχιστα το f1 score των εφικτών ερωτήσεων. Δηλαδή το μοντέλο δεν επιστρέφει πάντα την κενή συμβολοσειρά.
- Ωστόσο η τιμή του learning rate παραμένει μεγάλη.

Εκτέλεση 4

Μειώνω ακόμη περισσότερο το learning rate. Επίσης αυξάνω το μήκος του max_length σε 175. Τα υπόλοιπα παραμένουν ως έχουν.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 175
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 175
- learning_rate = 5e-5
- batch_size = 64
- impossible questions: δεν συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 1.53036 | Validation Loss = 1.37682 | Train Accuracy = 0.5405 | Validation Accuracy = 0.5649
Epoch 1: Train Loss = 0.93787 | Validation Loss = 1.42082 | Train Accuracy = 0.6154 | Validation Accuracy = 0.5618

Evaluation:

- "exact": 21.704708161374548,
- "f1": 24.857968333579848,
- "total": 11873,
- "HasAns_exact": 40.45209176788124,
- "HasAns_f1": 46.767654862448296,
- "HasAns_total": 5928,
- "NoAns_exact": 3.010933557611438,
- "NoAns_f1": 3.010933557611438,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πολύ μικρότερες τιμές και λογικές τιμές τόσο στο train και validation loss όσο και στο accuracy.
- Πλέον το μοντέλο έχει αρχίσει και εκπαιδεύεται. Αυτό φαίνεται και από τα scores των μετρικών. Πιο συγκεκριμένα παρατηρούμε ότι τα scores για τις εφικτές ερωτήσεις έχουν αυξηθεί πολύ σε σχέση με πριν. Αντίθετα, τα scores για τις impossible ερωτήσεις είναι πολύ μικρά, κάτι που είναι λογικό καθώς δεν συμπεριλαμβάνονται στο training set.
- Ωστόσο μέσω των train/validation loss παρατηρούμε πως το μοντέλο παρουσιάζει overfit, καθώς στο δεύτερο epoch το train loss μειώνεται ενώ το validation loss αυξάνεται.

Εκτέλεση 5

Με στόχο την εξάλειψη του overfit, μειώνω ακόμη περισσότερο το learning rate και αυξάνω το μήκος του max_length σε 200. Τα υπόλοιπα παραμένουν ως έχουν.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 200
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 200
- learning_rate = 2e-5
- batch_size = 64
- impossible questions: δεν συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 1.98018 | Validation Loss = 1.30910 | Train Accuracy = 0.4636 | Validation Accuracy = 0.6066
Epoch 1: Train Loss = 1.14283 | Validation Loss = 1.17667 | Train Accuracy = 0.5571 | Validation Accuracy = 0.6237

Evaluation:

- "exact": 32.68761054493388,
- "f1": 38.2035730269766,
- "total": 11873,

- "HasAns_exact": 59.49730094466936,
- "HasAns_f1": 70.54504428969172,
- "HasAns_total": 5928,
- "NoAns_exact": 5.954583683767872,
- "NoAns_f1": 5.954583683767872,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πως τα scores των μετρικών για τις εφικτές ερωτήσεις έχουν αυξηθεί πολύ και είναι πολύ ικανοποιητικά.
- Το μοντέλο πλέον δεν παρουσιάζει overfit, καθώς τα train και validation loss είναι πολύ κοντά μεταξύ τους.
- Το μοντέλο εξακολουθεί να έχει κακή απόδοση όσον αφορά τα impossible questions, γεγονός αναμενόμενο καθώς εξακολουθώ να μην συμπεριλαμβάνω τα impossible questions στα δεδομένα εκπαίδευσης.

Εκτέλεση 6

Έτσι ώστε να αυξήσω την απόδοση του μοντέλου, αυξάνω ελάχιστα το learning rate (από 2e-5 σε 3e-5) και αυξάνω το μήκος του max_length από 200 σε 300. Τα υπόλοιπα παραμένουν ως έχουν.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 300
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 300
- learning_rate = 3e-5
- batch_size = 64
- impossible questions: δεν συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 1.44941 | Validation Loss = 1.07446 | Train Accuracy = 0.5919 | Validation Accuracy = 0.6879
Epoch 1: Train Loss = 0.80800 | Validation Loss = 1.07876 | Train Accuracy = 0.6681 | Validation Accuracy = 0.6907

Evaluation:

- "exact": 35.820769813863386,
- "f1": 40.98904039399743,
- "total": 11873,
- "HasAns_exact": 67.91497975708502,
- "HasAns_f1": 78.26634220612878,
- "HasAns_total": 5928,
- "NoAns_exact": 3.8183347350714887,
- "NoAns_f1": 3.8183347350714887,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πως τα scores των μετρικών για τις εφικτές ερωτήσεις έχουν αυξηθεί ακόμη περισσότερο.

Το μοντέλο μας έχει πάρα πολύ καλή απόδοση για τις εφικτές ερωτήσεις πετυχαίνοντας **F1-Score 78%** σε αυτές.

Έχοντας αυξήσει πολύ την απόδοση του μοντέλου όσον αφορά τις εφικτές ερωτήσεις, σειρά έχει η διαχείριση των impossible ερωτήσεων.

2.10.1 Διαχείριση ερωτήσεων χωρίς απάντηση

Όπως έχει ήδη αναφερθεί το dataset SQuAD2.0 περιέχει ερωτήσεις οι οποίες δεν έχουν απάντηση εντός του δοθέντος context.
Για της ερωτήσεις αυτές, σωστή απάντηση θεωρείται η κενή συμβολοσειρά.
Μέσω των εκτελέσεων που παρουσιάστηκαν παραπάνω, παρατηρήσαμε πως εκπαιδεύοντας το μοντέλο χωρίς να συμπεριλάβουμε τις impossible ερωτήσεις στο training set, αυτό αδυνατεί να αναγνωρίσει πως είναι impossible, με αποτέλεσμα να μην «απαντά» με την κενή συμβολοσειρά.
Αυτό φαίνεται ξεκάθαρα από τα επιμέρους score των μετρικών όσον αφορά τις impossible ερωτήσεις τα οποία είναι μικρότερα από 5%.
Έτσι πρέπει να εκπαιδεύσουμε το μοντέλο μας να αναγνωρίζει τις impossible ερωτήσεις και να απαντά σε αυτές την κενή συμβολοσειρά.

Αρχικά πρέπει να συμπεριλάβουμε τις impossible ερωτήσεις στο training set.
Στην συνέχεια πρέπει να ορίσουμε τα κατάλληλα start και end token index τα οποία θα δέχεται το μοντέλο, έτσι ώστε να εκπαιδευτεί και τελικά να επιστρέφει αυτά κάθε φορά που του δίνεται σαν είσοδος ένα impossible question.
Όσον αφορά τα start και end indexes δοκίμασα δύο διαφορετικές προσεγγίσεις.

- Το start και end token index να αντιστοιχεί στο index του πρώτου [CLS] token. Κάθε tokenized αναπαράσταση ξεκινά με ένα [CLS] token. Έτσι, η προσέγγιση αυτή είναι ισοδύναμη με το να ορίσουμε το start και end index ίσα με 0. Το μοντέλο θα εκπαιδευτεί και τα start και end logits που θα επιστρέφει θα είναι ίσα με 0, δηλαδή θα επιστρέφει το πρώτο [CLS] token.
- Το start και end token index να αντιστοιχεί στο index του [SEP] token. Κάθε tokenized αναπαράσταση περιέχει το context και το question. Αυτά τα δύο διαχωρίζονται μεταξύ τους με το [SEP] token, δηλαδή το [SEP] token σηματοδοτεί το τέλος του context (και αντίστοιχα την αρχή του question). Το μοντέλο θα εκπαιδευτεί και τα start και end logits που θα επιστρέφει θα αντιστοιχούν πάντα στο [SEP] token του input.

Δοκιμάζοντας τις δύο αυτές προσεγγίσεις, η απόδοση ήταν παρόμοια. Έτσι, επέλεξα να εφαρμόσω την πρώτη.
Κάθε φορά που το μοντέλο επιστρέφει start και end logits ίσα με 0 (δηλαδή επιστρέφει το [CLS] token) τότε ορίζω σαν σωστή απάντηση την κενή συμβολοσειρά.

Έχοντας εφαρμόσει τα παραπάνω στον κώδικα, εκπαιδεύω ξανά το μοντέλο, με στόχο την αναγνώριση των impossible ερωτήσεων.

Εκτέλεση 7

Εφαρμόζω όσα αναφέρθηκαν στην παράγραφο 2.10.1 Διαχείριση ερωτήσεων χωρίς απάντηση έτσι ώστε να συμπεριλάβω τις impossible ερωτήσεις στο training set. Επίσης, λόγω της αύξησης των δεδομένων και την αύξηση της δυσκολίας του προβλήματος (αναγνώριση και των impossible questions) μειώνω το batch_size από 64 σε 32.
Τα υπόλοιπα παραμένουν ως έχουν.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = 300
- Πλήθος δεδομένων: μόνο όσα sets έχουν context_words + question_words < 300
- learning_rate = 3e-5
- batch_size = 32
- impossible questions: συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 1.49799 | Validation Loss = 1.27287 | Train Accuracy = 0.5363 | Validation Accuracy = 0.6084
Epoch 1: Train Loss = 0.81859 | Validation Loss = 1.32071 | Train Accuracy = 0.6369 | Validation Accuracy = 0.6271

Evaluation:

- "exact": 63.95182346500463,
- "f1": 68.8645354414101,
- "total": 11873,
- "HasAns_exact": 64.1531713900135,
- "HasAns_f1": 73.99268375436225,
- "HasAns_total": 5928,
- "NoAns_exact": 63.751051303616485,
- "NoAns_f1": 63.751051303616485,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πως τα scores των μετρικών για τις ανέφικτές ερωτήσεις έχουν αυξηθεί πολύ. Πλέον, το μοντέλο αναγνωρίζει τα impossible questions και «απαντά» την κενή συμβολοσειρά για αυτά.
- Παρατηρούμε μια πολύ μικρή μείωση όσον αφορά τα scores των εφικτών ερωτήσεων. Αυτό είναι αναμενόμενο καθώς το μοντέλο πλέον δεν ειδικεύεται μόνο στην απάντηση εφικτών ερωτήσεων.
- Τέλος, παρατηρούμε πως το μοντέλο παρουσιάζει ελάχιστο overfit καθώς στο δεύτερο epoch το train loss μειώνεται ενώ το validation loss αυξάνεται.

Εκτέλεση 8

Για την εξάλειψη του overfit αλλά και την αύξηση της απόδοσης του μοντέλου αφαιρώ τον περιορισμό του max_length κατά το tokenization έτσι ώστε να προκύψουν μεγαλύτερες αναπαραστάσεις για τα δεδομένα και να μην υπάρχει απώλεια πληροφορίας. Όταν δεν υπάρχει περιορισμός, το μέγιστο max_length που μπορεί να παράξει ο tokenizer είναι 512. Επίσης συμπεριλαμβάνω όλα τα δεδομένα τόσο στο training set.
Λόγω της αύξησης του μεγέθους των tokenized αναπαραστάσεων μειώνω το batch size σε 16 καθώς διαφορετικά υπερβαίνει τα όρια χρήσης της μνήμης της GPU και είναι αδύνατη η εκπαίδευση τόσο μέσω της πλατφόρμας Kaggle όσο και μέσω του Google Colab.
Τέλος, λόγω της αύξησης των δεδομένων, την μείωση του batch size αλλά και του overfit που παρουσιάστηκε στην προηγούμενη εκτέλεση, μειώνω το learning rate από 3e-5 σε 1e-5.

Δεδομένα και υπερπαραμέτροι:

- Epochs = 2
- MAX_LENGTH = κανένας περιορισμός (by default 512)
- Πλήθος δεδομένων: όλα τα δεδομένα (130319 sets ερωτήσεων/απαντήσεων)
- learning_rate = 1e-5
- batch_size = 16
- impossible questions: συμπεριλαμβάνονται στα training data.

Πληροφορίες κατά την εκπαίδευση:

Epoch 0: Train Loss = 1.59697 | Validation Loss = 1.39957 | Train Accuracy = 0.5348 | Validation Accuracy = 0.5698
Epoch 1: Train Loss = 0.94147 | Validation Loss = 1.33266 | Train Accuracy = 0.6183 | Validation Accuracy = 0.6011

Evaluation:

- "exact": 64.92040764760381,
- "f1": 69.37908304492896,
- "total": 11873,
- "HasAns_exact": 60.82995951417004,
- "HasAns_f1": 69.76009665864375,
- "HasAns_total": 5928,
- "NoAns_exact": 68.99915895710681,
- "NoAns_f1": 68.99915895710681,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πως τα scores των μετρικών για τις ανέφικτες ερωτήσεις έχουν αυξηθεί ακόμη περισσότερο.
- Παρατηρούμε μια ακόμη μια μικρή μείωση όσον αφορά τα scores των εφικτών ερωτήσεων.
- Ωστόσο, τα συνολικά scores (exact match και f1) έχουν αυξηθεί.
- Τέλος, παρατηρούμε πως το μοντέλο δεν παρουσιάζει πια overfit καθώς τόσο το train loss όσο και το validation loss μειώνονται στο δεύτερο epoch. Βέβαια παρατηρούμε μια απόκλιση μεταξύ τους καθώς το train loss είναι χαμηλότερο από το validation loss.

Εκτελώντας περισσότερα πειράματα δεν κατάφερα να βελτιώσω περισσότερο την απόδοση του μοντέλου.

2.11 Βέλτιστο Μοντέλο

Έπειτα από την μελέτη που παρουσιάστηκε παραπάνω, κατέληξα στο βέλτιστο μοντέλο μέσω της Εκτέλεσης 8.
Έτσι παρουσιάζω αναλυτικότερα την διαδικασία παραγωγής του συγκεκριμένου μοντέλου.

Αρχικά δημιουργώ το training και το dev set μέσω των αντίστοιχων αρχείων squad.
Τόσο στο training όσο και στο dev set περιέχονται όλες οι ερωτήσεις (εφικτές και ανέφικτες).
Όπως εξηγείται αναλυτικά στην παράγραφο 2.2 Προετοιμασία Δεδομένων, οι ερωτήσεις οι οποίες έχουν πολλαπλές απαντήσεις εισάγονται πολλές φορές στα sets, μια φορά για κάθε σωστή απάντηση.
Συνολικά για το training set προκύπτουν 130.319 sets ερωτήσεων-απαντήσεων.

Στην συνέχεια εφαρμόζω tokenization χρησιμοποιώντας τον BertTokenizerFast και συγκεκριμένα το μοντέλο bert-base-uncased.
Κατά το tokenization δεν επιβάλω κάποιον περιορισμό για το max_length και έτσι εφαρμόζεται το μέγιστο δυνατό, δηλαδή 512. Επίσης εφαρμόζεται padding στις αναπαραστάσεις με μήκος μικρότερο από 512.

Έπειτα εφαρμόζω τα εξής:

- Μετατροπή των start και end indexes στα αντίστοιχα indexes της tokenized μορφής (εξηγείται αναλυτικά στην παράγραφο 2.3 Tokenization). Όσον αφορά τα impossible questions η μετατροπή των start και end indexes περιγράφεται στην παράγραφο 2.10.1 Διαχείριση ερωτήσεων χωρίς απάντηση.
- 2.4 Batching

Το μοντέλο που χρησιμοποιώ είναι το BertForQuestionAnswering και συγκεκριμένα το pre-trained μοντέλο bert-base-uncased.

Ο optimizer που χρησιμοποιώ είναι ο AdamW.

Τέλος ορίζω τις εξής υπερπαραμέτρους:

- epochs = 2
- learning_rate = 1e-5
- batch_size = 16

Κατά την διάρκεια της εκπαίδευσης (fine-tuning) εφαρμόζω gradient clipping. Η διαδικασία του fine tuning εξηγείται αναλυτικά στην παράγραφο 2.8 Εκπαίδευση μοντέλου (Fine Tuning).

2.12 Αξιολόγηση βέλτιστου μοντέλου

Η αξιολόγηση του μοντέλου γίνεται χρησιμοποιώντας το επίσημο evaluation script από την σελίδα του SQuAD.
Η διαδικασία μέσω της οποίας προκύπτουν τα αποτελέσματα των μετρικών εξηγείται αναλυτικά στην παράγραφο 2.9 Αξιολόγηση του μοντέλου.

Τα αποτελέσματα των μετρικών είναι:

- "exact": 64.92040764760381,
- "f1": 69.37908304492896,
- "total": 11873,
- "HasAns_exact": 60.82995951417004,
- "HasAns_f1": 69.76009665864375,
- "HasAns_total": 5928,
- "NoAns_exact": 68.99915895710681,
- "NoAns_f1": 68.99915895710681,
- "NoAns_total": 5945

Παρατηρήσεις:

- Παρατηρούμε πως το μοντέλο έχει αρκετά ικανοποιητική απόδοση. Παρουσιάζει αντίστοιχη απόδοση τόσο στις εφικτές όσο και στις ανέφικτες ερωτήσεις.
- Όπως παρουσιάστηκε και στην παραπάνω μελέτη το μοντέλο μπορεί να επιτύχει υψηλότερα scores όταν λαμβάνουμε υπόψιν μόνο τις εφικτές ερωτήσεις. Ωστόσο προσθέτοντας και τις ανέφικτες, το score των εφικτών μειώνεται αλλά ταυτόχρονα το μοντέλο μας έχει την δυνατότητα να αναγνωρίζει ανέφικτες ερωτήσεις.
- Τέλος, παρατηρούμε πως το μοντέλο δεν παρουσιάζει πια overfit καθώς τόσο το train loss όσο και το validation loss μειώνονται στο δεύτερο epoch. Βέβαια παρατηρούμε μια απόκλιση μεταξύ τους καθώς το train loss είναι χαμηλότερο από το validation loss.

Μέρος Γ – Bert Finetuning on more datasets

Στην τρίτη άσκηση της εργασίας θα συμπληρώσουμε τον πίνακα 3 του paper: "What do Models Learn from Question Answering Datasets?".

Αρχικά ακολουθεί η αντίστοιχη μελέτη και η αναλυτική παρουσίαση του κάθε μοντέλου και των αποτελεσμάτων αυτών. Δηλαδή, η διαδικασία με την οποία προέκυψαν όλες οι τιμές του πίνακα.
Τελικά παρουσιάζεται ο συμπληρωμένος πίνακας στην παράγραφο 3.8 Πίνακας αποτελεσμάτων (Table 3).

Πιο συγκεκριμένα, έχοντας τα εξής datasets:

- SQuAD 2.0
- TriviaQA
- Natural Questions (NQ)
- QuAC
- NewsQA

Για κάθε ένα από αυτά θα εκπαιδεύσουμε ένα μοντέλο (finetune) και στην συνέχεια θα το κάνουμε evaluate τόσο στο dev set του ίδιου datasets όσο και στα dev sets των υπόλοιπων datasets.

Έτσι για κάθε ένα dataset ακολουθούμε την εξής διαδικασία:

- Fine-tuning ενός Bert μοντέλου χρησιμοποιώντας το training set του dataset.
- Evaluation του μοντέλου χρησιμοποιώντας το dev set του dataset.
- Evaluation του μοντέλου χρησιμοποιώντας τα dev sets των υπόλοιπων τεσσάρων datasets.

Όσον αφορά το SQuAD dataset, το fine-tuning και το evaluation του μοντέλου με το dev set του (δηλαδή τα δύο πρώτα bullets) έχουν ήδη γίνει στο Μέρος Β της εργασίας.

Όπως αναφέρεται και στην παράγραφο Παραδοτέο αρχείο, όσον αφορά τα Μέρη Β και Γ της εργασίας, το παραδοτέο αρχείο περιέχει:

- Ένα κεφάλαιο με όνομα Datasets Creation στο οποίο δημιουργούνται τα datasets. Το κεφάλαιο αυτό περιέχει 5 παραγράφους (μία για κάθε dataset), όπου στην κάθε μία δημιουργείται το αντίστοιχο dataset.
- Ένα κεφάλαιο με όνομα Fine-Tuning and Testing στο οποίο πραγματοποιείται το finetuning και το evaluation των μοντέλων. Πιο συγκεκριμένα, το κεφάλαιο αυτό περιέχει 5 παραγράφους (μία για κάθε dataset), όπου κάθε παράγραφος περιέχει τα εξής:
    - Fine-tuning ενός Bert μοντέλου χρησιμοποιώντας το training set του dataset.
    - Μία υπο-παράγραφο με το evaluation του μοντέλου χρησιμοποιώντας το dev set του dataset/
    - Τέσσερις υπο-παράγραφους με το evaluation του μοντέλου χρησιμοποιώντας τα dev sets των υπόλοιπων τεσσάρων datasets.

3.1 Δημιουργία των datasets

Για την δημιουργία των datasets ακολούθησα τις οδηγίες που παρουσιάζονται στο paper και συγκεκριμένα στην σελίδα: https://github.com/amazon-research/qa-dataset-converter

Πιο συγκεκριμένα, όπως αναφέρεται και στο paper: *"To consistently compare and experiment across models, we convert all datasets into a SQuAD 2.0 JSON format"*
Δηλαδή, έπειτα από την λήψη του κάθε dataset, το μετατρέπω σε μορφή αντίστοιχη με αυτή των αρχείων SQuAD χρησιμοποιώντας τις συναρτήσεις μετατροπής που παρέχονται στην παραπάνω ιστοσελίδα.
Έτσι μπορεί να γίνει επαναχρησιμοποίηση του κώδικα που έγραψα για το μέρος Β της εργασίας.
Προφανώς η μετατροπή αυτή, δεν ισχύει για τα αρχεία SQuAD και αρκεί μόνο η λήψη τους.

Λόγω του αρκετού χρόνου που απαιτείται για την δημιουργία κάποιων από τα datasets, καθώς και των πολλαπλών εκτελέσεων που πραγματοποίησα, έπειτα από την πρώτη δημιουργία των datasets (και την μετατροπή τους σε SQuAD format) αποθήκευσα τα τελικά αρχεία σαν datasets στην πλατφόρμα Kaggle (όπου έγιναν και όλες οι εκτελέσεις). Έτσι, κάθε φορά έκανα import το αντίστοιχο datasets και τα χρησιμοποιούσα απευθείας.
Προφανώς τα paths των αρχείων ήταν διαφορετικά από αυτά του παραδοτέου αρχείου.
Τα paths που υπάρχουν στο παραδοτέο αρχείο αντιστοιχούν στο τρέχον directory του Notebook στο οποίο θα αποθηκευτούν τα αρχεία εφόσον εκτελεστούν τα αντίστοιχα κελία που τα δημιουργούν, η εφόσον «ανέβουν χειροκίνητα» τα αντίστοιχα αρχεία.
Τα datasets μπορούν να βρεθούν στα εξής link:

- SQuAD: www.kaggle.com/dataset/6e922d6088651199b6768578f8c38c435e162329e321fd7d1d91b27732952e71
- TriviaQA: www.kaggle.com/dataset/a86f2afc121f1cd127831352fbbf1c62238008ec392f7c7faa4661d672f48cb5
- NewsQA: www.kaggle.com/dataset/c823fc6fcda1669d1d36885d06d44761cba7430978a70f7fa486572be74cb504
- Natural Questions: www.kaggle.com/dataset/eb0f7f399a2e1ddb0ea35dbe13581668a10b51207825998c14226ab1cfaaaae5
- QuAC: www.kaggle.com/dataset/64cd771bd7b197f40a35cc067eb7eb50bc98729ab72a39f216a0222b36917277

Στην συνέχεια παραθέτω περισσότερες λεπτομέρειες για την δημιουργία του κάθε dataset.

3.1.1 SQuAD

Όπως αναφέρθηκε και στην παράγραφο 2.2 Προετοιμασία Δεδομένων του Μέρους Β, αρκεί η λήψη των δύο αρχείων train και dev μέσω των παρακάτω εντολών:

```
!wget https://rajpurkar.github.io/SQuAD-explorer/dataset/train-v2.0.json
!wget https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v2.0.json
```

3.1.2 TriviaQA

Η λήψη των αρχείων του TriviaQA dataset γίνεται μέσω της εντολής:

```
!wget https://nlp.cs.washington.edu/triviaqa/data/triviaqa-rc.tar.gz
```

Ακολουθούμενη από την εντολή:

```
!tar -xzf triviaqa-rc.tar.gz
```

Έτσι ώστε να αποσυμπιεστεί το αρχείο που λήφθηκε.

Στην συνέχεια για την μετατροπή των αρχείων σε SQuAD format, έχω αντιγράψει τον κώδικα των απαραίτητων συναρτήσεων για την μετατροπή και συγκεκριμένα:

- Τον κώδικα του αρχείου triviaqa_to_squad.py : https://github.com/amazon-research/qa-dataset-converter/tree/main/triviaqa
- Καθώς και τις βοηθητικές συναρτήσεις από το TriviaQA repo : https://github.com/mandarjoshi90/triviaqa οι οποίες γίνονται import από το αρχείο triviaqa_to_squad.py.
- Τέλος, έχοντας τις απαραίτητες συναρτήσεις, καλώ την συνάρτηση triviaqa_to_squad τόσο για το train αρχείο όσο και για το dev αρχείο

Ως αποτέλεσμα έχουν παραχθεί δύο αρχεία train και dev σε μορφή SQuAD, με ονόματα:

- **triviaqa_train_SquadFormat.json**
- **triviaqa_dev_SquadFormat.json**

3.1.3 NewsQA

Για την δημιουργία των αρχείων του dataset NewsQA αρχικά ακολούθησα τα βήματα που περιγράφονται στο github repo του NewsQA: https://github.com/Maluuba/newsqa
Πιο συγκεκριμένα, πραγματοποίησα το Manual Setup, μέσω των εξής εντολών:

- Αρχικά έκανα clone το repo του NewsQA μέσω της εντολής:

```
!git clone https://github.com/Maluuba/newsqa
```

- Λήψη των αρχείων για της ερωτήσεις απαντήσεις μέσω της σελίδας: https://msropendata.com/datasets/939b1042-6402-4697-9c15-7a28de7e1321
  Η λήψη τον αρχείων έγινε τοπικά και στην συνέχεια τα ανέβασα «χειροκίνητα» στα δεδομένα του Notebook καθώς για την λήψη απαιτείται σύνδεση με λογαριασμό και έτσι δεν είναι δυνατή η λήψη μέσω μιας εντολής από το Notebook (όπως παραπάνω).
- Λήψη των αρχείων CNN Stories μέσω της σελίδας http://cs.nyu.edu/~kcho/DMQA/
  Και σε αυτήν την περίπτωση κατέβασα τα αρχεία τοπικά και στην συνέχεια τα ανέβασα στα δεδομένα του Notebook καθώς δεν ήταν δυνατή η απευθείας λήψη των αρχείων από το Notebook.
- Λήψη των JAR αρχείων τα οποία παρέχονται από βιβλιοθήκες του Stanford και είναι απαραίτητες για το tokenization των δεδομένων. Η λήψη έγινε μέσω της εντολής:

```
!wget https://nlp.stanford.edu/software/stanford-postagger-2015-12-09.zip
```

- Λήψη του Conda και δημιουργία περιβάλλοντος Conda.
- Λήψη των requirements του Conda.
- Δημιουργία των αρχείων train, dev και test μέσω της εντολής:

```
!python2.7 newsqa/maluuba/newsqa/data_generator.py
```

- Έλεγχος για τον αν τα αρχεία δημιουργήθηκαν σωστά, μέσω της εντολής:

```
!python2.7 -m unittest discover ./newsqa
```

Έχοντας δημιουργήσει τα αρχεία του NewsQA dataset, σειρά έχει η μετατροπή τους σε SQuAD format. Αντιγράφω τον κώδικα του αρχείου newsqa_to_squad.py και στην συνέχεια καλώ την συνάρτηση newsqa_to_squad τόσο για το train set όσο και για το dev set.

Έτσι δημιουργούνται τα αντίστοιχα αρχεία train και dev σε SQuAD format, με ονόματα:

- **newsqa_train_SquadFormat.json**
- **newsqa_dev_SquadFormat.json**

3.1.4 Natural Questions (NQ)

Η λήψη των αρχείων του NQ dataset γίνεται μέσω της εντολής:

```
!gsutil -m cp -R gs://natural_questions/v1.0 .
```

Και έτσι κατεβαίνουν τα απαραίτητα αρχεία για το train και το dev set στον τρέχον φάκελο.

Έχοντας κατεβάσει τα αρχεία του NQ dataset, σειρά έχει η μετατροπή τους σε SQuAD format. Αντιγράφω τον κώδικα του αρχείου nq_to_squad.py και στην συνέχεια καλώ την συνάρτηση nq_to_squad_format τόσο για το train set όσο και για το dev set.

Έτσι δημιουργούνται τα αντίστοιχα αρχεία train και dev σε SQuAD format, με ονόματα:

- **nq_train.json**
- **nq_dev.json**

3.1.5 QuAC

Η λήψη των αρχείων train και dev set του QuAC dataset γίνεται μέσω των εντολών:

```
!wget https://s3.amazonaws.com/my89public/quac/train_v0.2.json
!wget https://s3.amazonaws.com/my89public/quac/val_v0.2.json
```

αντίστοιχα.

Έχοντας κατεβάσει τα δύο αρχεία του QuAC dataset, σειρά έχει η μετατροπή τους σε SQuAD format. Αντιγράφω τον κώδικα του αρχείου quac_to_squad.py και στην συνέχεια καλώ την συνάρτηση quac_to_squad τόσο για το train set όσο και για το dev set.

Έτσι δημιουργούνται τα αντίστοιχα αρχεία train και dev σε SQuAD format, με ονόματα:

- **quac_train_SquadFormat.json**
- **quac_dev_SquadFormat.json**

Σημαντική παρατήρηση: Όσον αφορά το QuAC dataset, στο paper αναφέρεται:
"To standardize training, we do not model contextual information, but we include QuAC to see how models trained without context handle context-dependent questions."
Επομένως, σαν πληροφορία για την απάντηση των ερωτήσεων χρησιμοποιείται το πεδίο "background" και όχι το πεδίο "context" όπως στα υπόλοιπα datasets.
Περισσότερες πληροφορίες εξηγούνται στην συνέχεια, στην αντίστοιχη παράγραφο 3.7 QuAC.

3.2 Finetuning and Testing

Έχοντας μετατρέψει όλα τα datasets σε SQuAD format η διαδικασία που ακολουθείται έτσι ώστε να προκύψει ένα finetuned μοντέλο άλλα και στην συνέχεια να γίνει evaluate είναι η ίδια με αυτήν που περιεγράφηκε αναλυτικά στο Μέρος Β.
Περιγραφικά για κάθε ένα dataset η διαδικασία που ακολουθείται για να προκύψει το αντίστοιχο μοντέλο είναι:

- Ανάγνωση των αρχείων και δημιουργία των αντίστοιχων δομών που περιέχουν τα δεδομένα των train και dev set. Περιγράφεται αναλυτικά στην παράγραφο 2.2 Προετοιμασία Δεδομένων.
- Tokenization των δεδομένων και μετατροπή των start/end indexes στα αντίστοιχα indexes των tokens. Περιγράφεται αναλυτικά στην παράγραφο 2.3 Tokenization.
- 2.4 Batching.
- 2.5 Ορισμός του Μοντέλου και των υπερπαραμέτρων που θα χρησιμοποιηθούν για την εκπαίδευση. Το μοντέλο που χρησιμοποιείται είναι το ίδιο σε όλες τις περιπτώσεις. Ωστόσο οι υπερπαραμέτροι αλλάζουν και εξηγούνται στην συνέχεια για κάθε περίπτωση ξεχωριστά.
- Εκπαίδευση του μοντέλου όπως περιγράφεται στην παράγραφο 2.8 Εκπαίδευση μοντέλου (Fine Tuning).
- Αξιολόγηση του μοντέλου χρησιμοποιώντας το dev file του dataset με το οποίο εκπαιδεύτηκε το μοντέλο. Η διαδικασία είναι ακριβώς ίδια με αυτήν που περιγράφεται στην παράγραφο 2.9 Αξιολόγηση του μοντέλου.
- Αξιολόγηση του μοντέλου χρησιμοποιώντας τα dev files των υπόλοιπων τεσσάρων datasets. Η διαδικασία είναι ακριβώς η ίδια, με την διαφορά ότι κάθε φορά αλλάζει το path του dev file ανάλογα με ποιο datasets θέλουμε να χρησιμοποιήσουμε για το evaluation.

Επιπλέον παρατηρήσεις:

- Παράμετροι όπως το max_length του tokenizer ή υπερπαράμετροι όπως το learning rate αλλάζουν ανάλογα το μοντέλο. Ωστόσο η διαδικασία παραμένει ίδια. Συγκεκριμένες λεπτομέρειες εξηγούνται στην συνέχεια για κάθε μια περίπτωση ξεχωριστά.
- Το evaluation σε όλες τις περιπτώσεις γίνεται χρησιμοποιώντας το evaluation script του SQuAD, εφόσον έχουμε μετατρέψει όλα τα datasets σε SQuAD format. Το evaluation script δημιουργείται μια φορά στην αρχή (με τρόπο που εξηγείται στην παράγραφο 2.9 Αξιολόγηση του μοντέλου) και χρησιμοποιείται κάθε φορά που κάνουμε evaluation.

3.3 SQuAD

Η διαδικασία παραγωγής του μοντέλου το οποίο έγινε finetuned στο SQuAD dataset είναι αντικείμενο του Μέρους Β και εξηγήθηκε πολύ αναλυτικά.
Έχοντας το εκπαιδευμένο μοντέλο, κάνουμε evaluate τόσο στο dev set του SQuAD όσο και στα dev sets των υπόλοιπων datasets.

Το εκπαιδευμένο μοντέλο το έχω αποθηκεύσει σαν dataset εδώ:
www.kaggle.com/dataset/91a03213c19a0012bc1282b5a9c9e5fb446f86b938ed15a95f9b1f9b07ea2a31
Έτσι, μπορεί να γίνει λήψη του μοντέλου και απευθείας χρήση του, μέσω της εντολής model.load(…).

3.3.1 Finetuned model evaluation

Το evaluation του finetuned μοντέλου έγινε επίσης στο Μέρος Β και παρουσιάστηκε στην παράγραφο 2.12 Αξιολόγηση βέλτιστου μοντέλου.
Παρουσιάζω ξανά τα αποτελέσματα του evaluation script.

- "exact": 64.92040764760381,
- "f1": 69.37908304492896,
- "total": 11873,
- "HasAns_exact": 60.82995951417004,
- "HasAns_f1": 69.76009665864375,
- "HasAns_total": 5928,
- "NoAns_exact": 68.99915895710681,
- "NoAns_f1": 68.99915895710681,
- "NoAns_total": 5945

3.3.2 Evaluation for TriviaQA

Για να κάνουμε evaluate το μοντέλο, αντικαθίσταται το path του dev file με το path του αντίστοιχου αρχείου dev του TriviaQA dataset και εκτελείται ο ίδιος κώδικας (και το evaluation script).
Τα αποτελέσματα είναι τα εξής:

- "exact": 36.39047016656125,
- "f1": 40.263927771473604,
- "total": 14229,
- "HasAns_exact": 17.98414956309693,
- "HasAns_f1": 23.584172755567558,
- "HasAns_total": 9842,
- "NoAns_exact": 77.68406656029177,
- "NoAns_f1": 77.68406656029177,
- "NoAns_total": 4387

3.3.3 Evaluation for NewsQA

Αντίστοιχα κάνουμε evaluate το SQuAD finetuned μοντέλο χρησιμοποιώντας το dev file του NewsQA dataset.
Τα αποτελέσματα είναι τα εξής (Το NewsQA περιέχει μόνο εφικτές ερωτήσεις):

- "exact": 16.78281068524971,
- "f1": 25.44817225388683,
- "total": 5166,
- "HasAns_exact": 16.78281068524971,
- "HasAns_f1": 25.44817225388683,
- "HasAns_total": 5166

3.3.4 Evaluation for Natural Questions (NQ)

Τα αποτελέσματα του evaluation του SQuAD finetuned μοντέλου χρησιμοποιώντας το dev file του NQ dataset είναι τα εξής:

- "exact": 37.399821905609976,
- "f1": 44.70900438891064,
- "total": 3369,
- "HasAns_exact": 36.79966044142615,
- "HasAns_f1": 47.25154320298811,
- "HasAns_total": 2356,
- "NoAns_exact": 38.79565646594274,
- "NoAns_f1": 38.79565646594274,
- "NoAns_total": 1013

3.3.5 Evaluation for QuAC

Τα αποτελέσματα του evaluation του SQuAD finetuned μοντέλου χρησιμοποιώντας το dev file του QuAC dataset είναι τα εξής:

- "exact": 11.53113951590971,
- "f1": 13.938658490054106,
- "total": 7354,
- "HasAns_exact": 0.34083162917518744,
- "HasAns_f1": 3.358025653690845,
- "HasAns_total": 5868,
- "NoAns_exact": 55.720053835800805,
- "NoAns_f1": 55.720053835800805,
- "NoAns_total": 1486

3.4 TriviaQA

Η μελέτη που πραγματοποίησα έτσι ώστε να προκύψει το finetuned μοντέλο είναι αντίστοιχη με αυτήν που πραγματοποιήθηκε και εξηγήθηκε αναλυτικά στο Μέρος Β.

Ωστόσο λόγο του μεγάλου μήκους των contexts/questions αλλά και το μεγάλο πλήθος αυτών δεν κατάφερα να πραγματοποιήσω το tokenization χρησιμοποιώντας ολόκληρο το σύνολο των δεδομένων του training set λόγω των περιορισμών μνήμης τόσο της πλατφόρμας Colab όσο και του Kaggle.
Έτσι, αρχικά δοκίμασα να περιορίσω το μέγεθος max_length, δηλαδή το μέγιστο μήκος των tokenized αναπαραστάσεων, μειώνοντας την τιμή του. Ωστόσο το αποτέλεσμα ήταν το ίδιο.
Έτσι, δοκίμασα να περιορίσω το πλήθος των δεδομένων μειώνοντας το έως ότου το tokenization να είναι εφικτό.
Ταυτόχρονα το max_length το είχα ορίσει με την μέγιστη δυνατή τιμή του, δηλαδή 512.
Τελικά κατέληξα στο εξής:

- Χρησιμοποιώ μόνο όσα sets context/questions έχουν αθροιστικό πλήθος λέξεων μικρότερο από 700.

Η επιλογή αυτή έγινε καθώς με αυτόν τον τρόπο ταυτόχρονα με την μείωση του πλήθους των δεδομένων, επιτυγχάνω να υπάρχει όσο τον δυνατόν μικρότερη απώλεια πληροφορίας στα sets που «κρατάω». Αυτό καθώς το max_length είναι περιορισμένο (512) και σε contexts/questions με μεγαλύτερο πλήθος από tokens εφαρμόζεται truncation.
Έτσι, με την επιλογή να «κρατάω» τα μικρότερα σε πλήθος tokens, το ποσοστό αποκοπής είναι πολύ μικρότερο και η πιθανότητα να αποκοπεί η απάντηση που βρίσκεται εσωτερικά του context είναι ακόμα μικρότερη.
Έχοντας περιορίσει έτσι το πλήθος των δεδομένων του training set, το tokenization είναι πλέον εφικτό.

Άρα συνολικά, ορίζω max_length=512 και «κρατάω» όσα context/questions έχουν πλήθος λέξεων μικρότερο από 700. Τελικά το training set αποτελείται από 42.628 sets.
Όσον αφορά το training, επιλέγω τις εξής υπερπαραμέτρους:

- epochs = 2
- learning_rate = 3e-5
- batch_size = 16

Το μοντέλο που χρησιμοποιώ (όπως και σε όλες τις περιπτώσεις) είναι το BertForQuestionAnswering και συγκεκριμένα το bert-base-uncased.
Ο optimizer που χρησιμοποιώ είναι ο AdamW.

Το εκπαιδευμένο μοντέλο το έχω αποθηκεύσει σαν dataset εδώ:
www.kaggle.com/dataset/0f12ddeef5f462171e508b1e8a788d637247a3e74913b2b666b282d211e0a52b
Έτσι, μπορεί να γίνει λήψη του μοντέλου και απευθείας χρήση του, μέσω της εντολής model.load(…).

3.4.1 Finetuned model evaluation

Έπειτα από το finetuning του μοντέλου το κάνουμε evaluate χρησιμοποιώντας το dev set του.
Τα αποτελέσματα είναι τα εξής:

- "exact": 38.40747768641507,
- "f1": 39.422830131417484,
- "total": 14229,
- "HasAns_exact": 12.131680552733185,
- "HasAns_f1": 13.599618973779668,
- "HasAns_total": 9842,
- "NoAns_exact": 97.35582402552997,
- "NoAns_f1": 97.35582402552997,
- "NoAns_total": 4387

3.4.2 Evaluation for SQuAD

Αντικαθιστώ το dev path του TriviaQA με αυτό του SQuAD dev file και κάνω evaluate το μοντέλο με αυτό. Τα αποτελέσματα είναι τα εξής:

- "exact": 13.324349364103428,
- "f1": 16.115175977555154,
- "total": 11873,
- "HasAns_exact": 13.090418353576249,
- "HasAns_f1": 18.680074963142967,
- "HasAns_total": 5928,
- "NoAns_exact": 13.557611438183347,
- "NoAns_f1": 13.557611438183347,
- "NoAns_total": 5945

3.4.3 Evaluation for NewsQA

Αντίστοιχα κάνω evaluate το μοντέλο χρησιμοποιώντας το dev set του NewsQA.
Τα αποτελέσματα είναι τα εξής:

- "exact": 5.555555555555555,
- "f1": 8.057651464334763,
- "total": 5166,
- "HasAns_exact": 5.555555555555555,
- "HasAns_f1": 8.057651464334763,
- "HasAns_total": 5166

3.4.4 Evaluation for Natural Questions (NQ)

Τα αποτελέσματα του evaluation του TriviaQA finetuned μοντέλου χρησιμοποιώντας το dev file του NQ dataset είναι τα εξής:

- "exact": 15.256752745621846,
- "f1": 20.5597419800712,
- "total": 3369,
- "HasAns_exact": 16.850594227504246,
- "HasAns_f1": 24.4336887652207,
- "HasAns_total": 2356,
- "NoAns_exact": 11.549851924975322,
- "NoAns_f1": 11.549851924975322,
- "NoAns_total": 1013

3.4.5 Evaluation for QuAC

Τέλος, τα αποτελέσματα του evaluation του TriviaQA finetuned μοντέλου χρησιμοποιώντας το dev file του QuAC dataset είναι τα εξής:

- "exact": 3.24993200979059,
- "f1": 5.65380821082426,
- "total": 7354,
- "HasAns_exact": 0.22154055896387184,
- "HasAns_f1": 3.2341693221543504,
- "HasAns_total": 5868,
- "NoAns_exact": 15.208613728129206,
- "NoAns_f1": 15.208613728129206,
- "NoAns_total": 1486

3.5 NewsQA

Αντίστοιχα ακολούθησα παρόμοια διαδικασία μελέτης με το Μέρος Β για την δημιουργία του finetuned μοντέλου.

Αντίστοιχα με το 3.4 TriviaQA, λόγω του μεγάλου όγκου των δεδομένων αλλά και το μεγάλο μήκος των contexts, το tokenization χρησιμοποιώντας ολόκληρο το training set ήταν αδύνατο λόγω των περιορισμών στην χρήση μνήμης.
Έτσι, πραγματοποίησα τους ίδιους πειραματισμούς μειώνοντας σταδιακά την παράμετρο max_length ή/και το πλήθος των δεδομένων.

Τελικά κατέληξα στο εξής:

- Ορίζω το max_length = 300
- Μειώνω το πλήθος των δεδομένων, χρησιμοποιώντας μόνο όσα sets context/question έχουν συνολικό πλήθος λέξεων μικρότερο από 750.
  Ο λόγος που επέλεξα να μειώσω το πλήθος των δεδομένων με αυτήν την «τεχνική» εξηγήθηκε στην παράγραφο 3.4 TriviaQA.

Έτσι τελικά το training set αποτελείται από 53.572 sets context/question.

Όσον αφορά το training, επιλέγω τις εξής υπερπαραμέτρους:

- epochs = 2
- learning_rate = 3e-5
- batch_size = 16

Το μοντέλο που χρησιμοποιώ (όπως και σε όλες τις περιπτώσεις) είναι το BertForQuestionAnswering και συγκεκριμένα το bert-base-uncased.
Ο optimizer που χρησιμοποιώ είναι ο AdamW.

Το εκπαιδευμένο μοντέλο το έχω αποθηκεύσει σαν dataset εδώ:
www.kaggle.com/dataset/b8bd1d1955db4a037b2576c1db60f88d6dabaeb30d1cac42f33090b045335bd7
Έτσι, μπορεί να γίνει λήψη του μοντέλου και απευθείας χρήση του, μέσω της εντολής model.load(…).

Το NewsQA datasets περιέχει μόνο εφικτές ερωτήσεις και επομένως το μοντέλο εκπαιδεύεται μόνο σε αυτές. Έτσι, περιμένουμε σχετικά μικρή απόδοση κατά το evaluation στα υπόλοιπα datasets όσον αφορά τις impossible ερωτήσεις.

3.5.1 Finetuned model evaluation

Έπειτα από το finetuning του μοντέλου το κάνουμε evaluate χρησιμοποιώντας το dev set του.
Τα αποτελέσματα είναι τα εξής:

- "exact": 33.15911730545877,
- "f1": 48.22154510226573,
- "total": 5166,
- "HasAns_exact": 33.15911730545877,
- "HasAns_f1": 48.22154510226573,
- "HasAns_total": 5166

3.5.2 Evaluation for SQuAD

Αντικαθιστώ το dev path του NewsQA με αυτό του SQuAD dev file και κάνω evaluate το μοντέλο με αυτό. Τα αποτελέσματα είναι τα εξής:

- "exact": 29.66394340099385,
- "f1": 36.76369805667687,
- "total": 11873,
- "HasAns_exact": 51.40013495276653,
- "HasAns_f1": 65.62000455919768,
- "HasAns_total": 5928,
- "NoAns_exact": 7.9899074852817495,
- "NoAns_f1": 7.9899074852817495,
- "NoAns_total": 5945

Όπως ήταν αναμενόμενο παρατηρούμε πολύ καλή απόδοση όσον αφορά τις εφικτές ερωτήσεις και αντίθετα χαμηλή απόδοση όσον αφορά τις impossible ερωτήσεις.

3.5.3 Evaluation for TriviaQA

Αντίστοιχα κάνω evaluate το μοντέλο χρησιμοποιώντας το dev set του TriviaQA.
Τα αποτελέσματα είναι τα εξής:

- "exact": 23.325602642490686,
- "f1": 29.818630894827496,
- "total": 14229,
- "HasAns_exact": 27.778906726275146,
- "HasAns_f1": 37.166155151646095,
- "HasAns_total": 9842,
- "NoAns_exact": 13.334852974697972,
- "NoAns_f1": 13.334852974697972,
- "NoAns_total": 4387

3.5.4 Evaluation for Natural Questions (NQ)

Τα αποτελέσματα του evaluation του NewsQA finetuned μοντέλου χρησιμοποιώντας το dev file του NQ dataset είναι τα εξής:

- "exact": 36.182843573760756,
- "f1": 44.80130253666088,
- "total": 3369,
- "HasAns_exact": 47.15619694397284,
- "HasAns_f1": 59.48030061375658,
- "HasAns_total": 2356,
- "NoAns_exact": 10.661401776900297,
- "NoAns_f1": 10.661401776900297,
- "NoAns_total": 1013

3.5.5 Evaluation for QuAC

Τέλος, τα αποτελέσματα του evaluation του NewsQA finetuned μοντέλου χρησιμοποιώντας το dev file του QuAC dataset είναι τα εξής:

- "exact": 3.1003535490889313,
- "f1": 6.508353102635553,
- "total": 7354,
- "HasAns_exact": 0.3067484662576687,
- "HasAns_f1": 4.577782671571566,
- "HasAns_total": 5868,
- "NoAns_exact": 14.131897711978466,
- "NoAns_f1": 14.131897711978466,
- "NoAns_total": 1486

3.6 Natural Questions (NQ)

Αντίστοιχα ακολούθησα παρόμοια διαδικασία μελέτης με το Μέρος Β για την δημιουργία του finetuned μοντέλου.

Το NQ datasets περιέχει σχετικά μικρού μήκους context και έτσι μπορεί να χρησιμοποιηθεί ολόκληρο το training dataset (σε αντίθεση με τα TriviaQA και NewsQA).
Έτσι:

- Ορίζω το max_length = 512, δηλαδή την μέγιστη δυνατή τιμή
- Χρησιμοποιώ ολόκληρο το training dataset.

Το training set αποτελείται από 110.865 sets context/question.

Όσον αφορά το training, επιλέγω τις εξής υπερπαραμέτρους:

- epochs = 2
- learning_rate = 3e-5
- batch_size = 16

Το μοντέλο που χρησιμοποιώ (όπως και σε όλες τις περιπτώσεις) είναι το BertForQuestionAnswering και συγκεκριμένα το bert-base-uncased.
Ο optimizer που χρησιμοποιώ είναι ο AdamW.

Το εκπαιδευμένο μοντέλο το έχω αποθηκεύσει σαν dataset εδώ:
www.kaggle.com/dataset/0f035f273680c9db304319efce624bf7a3b9734b5218b396d0881be5adc613a6
Έτσι, μπορεί να γίνει λήψη του μοντέλου και απευθείας χρήση του, μέσω της εντολής model.load(…).

3.6.1 Finetuned model evaluation

Έπειτα από το finetuning του μοντέλου το κάνουμε evaluate χρησιμοποιώντας το dev set του.
Τα αποτελέσματα είναι τα εξής:

- "exact": 64.49985158800831,
- "f1": 68.7583873447665,
- "total": 3369,
- "HasAns_exact": 57.088285229202036,
- "HasAns_f1": 63.177846759133345,
- "HasAns_total": 2356,
- "NoAns_exact": 81.73741362290227,
- "NoAns_f1": 81.73741362290227,
- "NoAns_total": 1013

3.6.2 Evaluation for SQuAD

Αντικαθιστώ το dev path του NQ με αυτό του SQuAD dev file και κάνω evaluate το μοντέλο με αυτό. Τα αποτελέσματα είναι τα εξής:

- "exact": 44.175861197675395,
- "f1": 47.00064431383382,
- "total": 11873,
- "HasAns_exact": 33.8225371120108,
- "HasAns_f1": 39.480204105625575,
- "HasAns_total": 5928,
- "NoAns_exact": 54.49957947855341,
- "NoAns_f1": 54.49957947855341,
- "NoAns_total": 5945

3.6.3 Evaluation for TriviaQA

Αντίστοιχα κάνω evaluate το μοντέλο χρησιμοποιώντας το dev set του TriviaQA.
Τα αποτελέσματα είναι τα εξής:

- "exact": 31.899641577060933,
- "f1": 36.44250312583861,
- "total": 14229,
- "HasAns_exact": 21.011989433042064,
- "HasAns_f1": 27.579798514281354,
- "HasAns_total": 9842,
- "NoAns_exact": 56.32550718030545,
- "NoAns_f1": 56.32550718030545,
- "NoAns_total": 4387

3.6.4 Evaluation for NewsQA

Τα αποτελέσματα του evaluation του NQ finetuned μοντέλου χρησιμοποιώντας το dev file του NewsQA dataset είναι τα εξής:

- "exact": 9.930313588850174,
- "f1": 13.07485522398131,
- "total": 5166,
- "HasAns_exact": 9.930313588850174,
- "HasAns_f1": 13.07485522398131,
- "HasAns_total": 5166

3.6.5 Evaluation for QuAC

Τέλος, τα αποτελέσματα του evaluation του NQ finetuned μοντέλου χρησιμοποιώντας το dev file του QuAC dataset είναι τα εξής:

- "exact": 17.555072069621975,
- "f1": 18.321486880235494,
- "total": 7354,
- "HasAns_exact": 0.2556237218813906,
- "HasAns_f1": 1.2161238100292884,
- "HasAns_total": 5868,
- "NoAns_exact": 85.86810228802153,
- "NoAns_f1": 85.86810228802153,
- "NoAns_total": 1486

3.7 QuAC

Όσον αφορά το QuAC dataset, στο paper αναφέρεται:

- *"To standardize training, we do not model contextual information, but we include QuAC to see how models trained without context handle context-dependent questions."*
- *"For QuAC, we ignored all context-related fields and treated each example as an independent question, so we see lower results than models built on the full dataset."*

Επομένως δεν χρησιμοποιείται το context σαν πληροφορία για την απάντηση των ερωτήσεων. Έτσι, χρησιμοποιώ το πεδίο background αντί του context (όπως αναφέρθηκε και στο piazza). Ωστόσο το πεδίο background δεν περιέχει τις ίδιες πληροφορίες και πολλές φορές δεν περιέχει την απάντηση στην ερώτηση. Έτσι, τα αποτελέσματα του μοντέλου που έγινε finetuned στο Quac dataset είναι αρκετά χαμηλά σε σχέση με τα υπόλοιπα.

Επίσης, στο dev set το οποίο χρησιμοποιήθηκε για το evaluation των υπόλοιπων μοντέλων περιέχει επίσης το background σαν πληροφορία αντί του context.
Συγκεκριμένα, τα αποτελέσματα που παρουσιάστηκαν στις παραγράφους:

- 3.3.5 Evaluation for QuAC
- 3.4.5 Evaluation for QuAC
- 3.5.5 Evaluation for QuAC
- 3.6.5 Evaluation for QuAC

στον τελικό πίνακα 3.8 Πίνακας αποτελεσμάτων (Table 3) αλλά και στο παραδοτέο Notebook, περιέχουν το background σαν πληροφορία. Για αυτό και τα scores των μετρικών είναι σχετικά χαμηλά.

Έτσι, για το finetuning με το QuAC dataset έγινε χρησιμοποιώντας το background. Ωστόσο, σαν επιπλέον μελέτη ακολούθησα την ίδια διαδικασία και χρησιμοποιώντας στο context σαν πληροφορία έτσι ώστε να συγκρίνω τα αποτελέσματα μεταξύ των δύο προσεγγίσεων. Η επιπλέον αυτή μελέτη παρουσιάζεται στην συνέχεια.

Όσον αφορά την διαδικασία για την παραγωγή του finetuned μοντέλου:

- Ορίζω το max_length = 512, δηλαδή την μέγιστη δυνατή τιμή.
- Χρησιμοποιώ ολόκληρο το training dataset.

Το training set αποτελείται από 83.568 sets context/question.
Όσον αφορά το training, επιλέγω τις εξής υπερπαραμέτρους:

- epochs = 2
- learning_rate = 3e-5
- batch_size = 16

Το μοντέλο που χρησιμοποιώ (όπως και σε όλες τις περιπτώσεις) είναι το BertForQuestionAnswering και συγκεκριμένα το bert-base-uncased.
Ο optimizer που χρησιμοποιώ είναι ο AdamW.

Το εκπαιδευμένο μοντέλο το έχω αποθηκεύσει σαν dataset εδώ:
www.kaggle.com/dataset/372778bd761b3e988e8544b92578adf63830702d43204b8f4987bb07562eefa0
Έτσι, μπορεί να γίνει λήψη του μοντέλου και απευθείας χρήση του, μέσω της εντολής model.load(…).

3.7.1 Finetuned model evaluation

Έπειτα από το finetuning του μοντέλου το κάνουμε evaluate χρησιμοποιώντας το dev set του. Το dev set που χρησιμοποιήθηκε για τα παρακάτω αποτελέσματα περιέχει το background σαν πληροφορία, αντίστοιχα με το training set χρησιμοποιήθηκε για το finetuning.
Τα αποτελέσματα είναι τα εξής:

- "exact": 18.30296437313027,
- "f1": 20.626521960802588,
- "total": 7354,
- "HasAns_exact": 0.0,
- "HasAns_f1": 2.9119704328123097,
- "HasAns_total": 5868,
- "NoAns_exact": 90.57873485868102,
- "NoAns_f1": 90.57873485868102,
- "NoAns_total": 1486

3.7.2 Evaluation for SQuAD

Αντικαθιστώ το dev path του Quac με αυτό του SQuAD dev file και κάνω evaluate το μοντέλο με αυτό. Τα αποτελέσματα είναι τα εξής:

- "exact": 31.525309525814873,
- "f1": 32.90385135917714,
- "total": 11873,
- "HasAns_exact": 0.0,
- "HasAns_f1": 2.7610369749511596,
- "HasAns_total": 5928,
- "NoAns_exact": 62.96047098402018,
- "NoAns_f1": 62.96047098402018,
- "NoAns_total": 5945

3.7.3 Evaluation for TriviaQA

Αντίστοιχα κάνω evaluate το μοντέλο χρησιμοποιώντας το dev set του TriviaQA.
Τα αποτελέσματα είναι τα εξής:

- "exact": 25.23719165085389,
- "f1": 25.376298053152556,
- "total": 14229,
- "HasAns_exact": 0.0,
- "HasAns_f1": 0.20111207054551883,
- "HasAns_total": 9842,
- "NoAns_exact": 81.85548210622294,
- "NoAns_f1": 81.85548210622294,
- "NoAns_total": 4387

3.7.4 Evaluation for NewsQA

Τα αποτελέσματα του evaluation του QuAC finetuned μοντέλου χρησιμοποιώντας το dev file του NewsQA dataset είναι τα εξής:

- "exact": 0.07742934572202866,
- "f1": 0.5363297972466945,
- "total": 5166,
- "HasAns_exact": 0.07742934572202866,
- "HasAns_f1": 0.5363297972466945,
- "HasAns_total": 5166

3.7.5 Evaluation for Natural Questions (NQ)

Τέλος, τα αποτελέσματα του evaluation του QuAC finetuned μοντέλου χρησιμοποιώντας το dev file του NQ dataset είναι τα εξής:

- "exact": 13.119620065301277,
- "f1": 18.6813919599444,
- "total": 3369,
- "HasAns_exact": 0.0,
- "HasAns_f1": 7.953144954606389,
- "HasAns_total": 2356,
- "NoAns_exact": 43.63277393879566,
- "NoAns_f1": 43.63277393879566,
- "NoAns_total": 1013

3.7.6 Σχολιασμός αποτελεσμάτων QuAC finetuned model

Με βάση τα αποτελέσματα των μετρικών τόσο κατά το evaluation του μοντέλου με το dev set του QuAC, όσο και με τα dev sets των υπόλοιπων datasets είναι φανερό ότι το μοντέλο υστερεί πολύ στην αναγνώριση των εφικτών ερωτήσεων.
Πιο συγκεκριμένα, σε κάθε περίπτωση τα επιμέρους scores που αφορούν τις εφικτές ερωτήσεις είναι πολύ χαμηλά. Επίσης, χαρακτηριστικά είναι τα αποτελέσματα κατά την δοκιμή με το dev set του NewsQA τα οποία είναι σχεδόν μηδενικά αφού το NewsQA περιέχει μόνο εφικτές ερωτήσεις.

Η χαμηλή αυτή απόδοση οφείλεται στην χρήση του background (αντί του context) σαν πληροφορία για την απάντηση των ερωτήσεων. Στις περισσότερες περιπτώσεις η απάντηση στις ερωτήσεις δεν περιέχεται στο background, και έτσι το μοντέλο αδυνατεί να εκπαιδευτεί στην εύρεση των σωστών απαντήσεων.
Πιο συγκεκριμένα:

- Στο σύνολο των 83.568 sets ερωτήσεων/απαντήσεων του training set, στις 45.242 δεν υπάρχει η απάντηση μέσα στο κείμενο του background.
- Στο σύνολο των 7.354 sets ερωτήσεων/απαντήσεων του validation set, στις 3.918 δεν υπάρχει η απάντηση μέσα στο κείμενο του background.

Επομένως είναι λογικό το μοντέλο να δυσκολεύεται να εκπαιδευτεί και σε συνέχεια να αδυνατεί να απαντήσει τις εφικτές ερωτήσεις.

Λόγω των παραπάνω, προχώρησα σε επιπλέον μελέτη, εκτελώντας την ίδια διαδικασία και χρησιμοποιώντας το context σαν πληροφορία. Η μελέτη αυτή παρουσιάζεται στην επόμενη παράγραφο.

3.7.7 Χρήση του context ως πληροφορία

Όπως αναφέρθηκε παραπάνω, λόγω της κακής απόδοσης του μοντέλου χρησιμοποιώντας το background σαν πληροφορία, δοκιμάζω να χρησιμοποιήσω το context (όπως και σε όλα τα υπόλοιπα datasets).

Η διαδικασία του finetuning είναι η ίδια, δηλαδή:

- Ορίζω το max_length = 512, δηλαδή την μέγιστη δυνατή τιμή.
- Χρησιμοποιώ ολόκληρο το training dataset.

Όσον αφορά το training, επιλέγω τις εξής υπερπαραμέτρους:

- epochs = 2
- learning_rate = 3e-5
- batch_size = 16

Το μοντέλο που χρησιμοποιώ είναι το BertForQuestionAnswering και συγκεκριμένα το bert-base-uncased και ο optimizer που χρησιμοποιώ είναι ο AdamW.

Το εκπαιδευμένο μοντέλο το έχω αποθηκεύσει σαν dataset εδώ:
www.kaggle.com/dataset/6ae450af0e0d57894ae3c9dc99588e1d0e2c9b854887b0a0f32a094827afaf82
Έτσι, μπορεί να γίνει λήψη του μοντέλου και απευθείας χρήση του, μέσω της εντολής model.load(…).

Ο κώδικας για την εκπαίδευση δεν υπάρχει στο παραδοτέο αρχείο, καθώς αποτελεί επιπλέον μελέτη. Ωστόσο μπορείτε να τον δείτε εδώ:
https://drive.google.com/file/d/1oK6Cc5ZN7wRtHaOYS8_CPGT0HLElVJPk/view?usp=sharing
(Η εκτέλεση έγινε στην πλατφόρμα Kaggle, ωστόσο είχα αποθηκεύσει το αρχείο μόνο τοπικά και όχι online στο Kaggle. Έτσι το παρουσιάζω μέσω του google drive.)

Τα αποτελέσματα κάνοντας evaluate το μοντέλο με το dev set του QuAC (το οποίο επίσης περιέχει το context ως πληροφορία) είναι:

- "exact": 21.498504215392984,
- "f1": 33.96829706897533,
- "total": 7354,
- "HasAns_exact": 7.498295841854124,
- "HasAns_f1": 23.125912857062875,
- "HasAns_total": 5868,
- "NoAns_exact": 76.78331090174966,
- "NoAns_f1": 76.78331090174966,
- "NoAns_total": 1486

Παρατηρούμε πως είναι υψηλότερα σε σχέση με αυτά του μοντέλου που εκπαιδεύτηκε στο background (3.7.1 Finetuned model evaluation)

Στην συνέχεια κάνω evaluate το μοντέλο χρησιμοποιώντας τα dev sets των υπόλοιπων datasets. Ο κώδικας με το evaluation υπάρχει εδώ:
https://www.kaggle.com/giannhskp/askisi3-quac-context/notebook

Συνοπτικά τα αποτελέσματα είναι:

- **SQUAD:**

"exact": 25.68853701676072,
"f1": 32.96025327996062,
"total": 11873,
"HasAns_exact": 3.9642375168690958,
"HasAns_f1": 18.528523480596117,
"HasAns_total": 5928,
"NoAns_exact": 47.35071488645921,
"NoAns_f1": 47.35071488645921,
"NoAns_total": 5945

- **TriviaQA:**

"exact": 25.982149132054257,
"f1": 30.65994564488643,
"total": 14229,
"HasAns_exact": 1.9813046128835603,
"HasAns_f1": 8.744194938131766,
"HasAns_total": 9842,
"NoAns_exact": 79.82676088443128,
"NoAns_f1": 79.82676088443128,
"NoAns_total": 4387

- **NewsQA:**

"exact": 2.3615950445218736,
"f1": 12.669742333736068,
"total": 5166,
"HasAns_exact": 2.3615950445218736,
"HasAns_f1": 12.669742333736068,
"HasAns_total": 5166

- **Natural Questions (NQ):**

"exact": 17.77975660433363,
"f1": 31.399399999760448,
"total": 3369,
"HasAns_exact": 6.32427843803056,
"HasAns_f1": 25.799906026822168,
"HasAns_total": 2356,
"NoAns_exact": 44.42250740375123,
"NoAns_f1": 44.42250740375123,
"NoAns_total": 1013

Συγκρίνοντας τα αποτελέσματα αυτά σε σχέση με τα αντίστοιχα αποτελέσματα από την χρήση του background σαν πληροφορία έχουμε:

Πίνακας με τα συνολικά F1-Scores

	SQuAD	TriviaQA	NewsQA	NQ	QuAC
Μοντέλο finetuned για το QuAC με το background	32.90%	25.37%	0.53%	18.68	20.62%
Μοντέλο finetuned για το QuAC με το context	32.96%	30.65%	12.66%	31.39%	33.96%

Table 1 - Σύγκριση εκπαίδευσης με background/context

Παρατηρούμε πως σε κάθε περίπτωση το μοντέλο το οποίο έχει εκπαιδευτεί χρησιμοποιώντας το context έχει καλύτερη απόδοση. Εξαιρώντας το SQuAD όπου η απόδοση είναι παρόμοια, σε όλες τις άλλες περιπτώσεις η διαφορά είναι σημαντική.
Άρα είναι προφανές ότι η μειωμένη απόδοση του πρώτου μοντέλου οφείλεται καθαρά στην επιλογή του background ως πληροφορία.

Επίσης, όπως έχει ήδη αναφερθεί, για το evaluation των μοντέλων των υπόλοιπων τεσσάρων μοντέλων χρησιμοποιήθηκε το dev set του QuAC το οποίο έχει το background ως πληροφορία για την απάντηση των ερωτήσεων.
Έτσι, στην συνέχεια πραγματοποιώ τα ίδια evaluations χρησιμοποιώντας το context ως πληροφορία.
Τα scores αυτά αντιστοιχούν στην στήλη QuAC του πίνακα 3.8 Πίνακας αποτελεσμάτων (Table 3).

Πίνακας με τα συνολικά F1-Scores

Μοντέλα	Datasets		
		QuAC with background	QuAC with context
	SQuAD	13.93%	18.80%
	TriviaQA	5.65%	4.83%
	NewsQA	6.50%	14.06%
	NQ	18.32%	2.15%
	QuAC (trained on background)	20.62%	8.67%

Table 2 - Απόδοση μοντέλων σε σχέση με το context/background του dev αρχείου

Η πρώτη στήλη του παραπάνω πίνακα έχει προκύψει από την βασική μελέτη του Μέρους Γ και έχει παρουσιαστεί παραπάνω στις αντίστοιχες παραγράφους. Επίσης, τα αποτελέσματα αυτά παρουσιάζονται και στο παραδοτέο Notebook.
Η δεύτερη στήλη αποτελεί αντικείμενο της επιπλέον μελέτης και ο κώδικας για την παραγωγή των αποτελεσμάτων μπορεί να βρεθεί εδώ:
https://www.kaggle.com/giannhskp/test-quac-evaluation-with-context/notebook

Παρατηρούμε πως σε ορισμένες περιπτώσεις τα μοντέλα έχουν καλύτερη απόδοση με το αρχείο που χρησιμοποιεί το background ενώ σε άλλες με το αρχείο που χρησιμοποιεί το context. Έτσι, δεν μπορούμε να βγάλουμε κάποιο ξεκάθαρο συμπέρασμα.

Προφανώς το μοντέλο QuAC έχει καλύτερη απόδοση στο αρχείο με τα background καθώς έχει εκπαιδευτεί σε αυτό. Ωστόσο όπως είδαμε στον παραπάνω πίνακα, το μοντέλο QuAC που έχει εκπαιδευτεί με το context πετυχαίνει καλύτερο score στο αντίστοιχο dev set του.

3.8 Πίνακας αποτελεσμάτων (Table 3)

Με βάση όσα αναφέρθηκαν παραπάνω και την αναλυτική παρουσίαση των αποτελεσμάτων για κάθε ένα μοντέλο, συμπληρώνω τον πίνακα (Table 3) του paper, το οποίο είναι και το ζητούμενο του Μέρους 3.
Οι τιμές που παρουσιάζονται στο παρακάτω πίνακα αντιστοιχούν στα συνολικά F1-Scores, δηλαδή στο πεδίο "f1" του output του evaluation script.

Fine-tuned on \ Evaluated on	SQuAD	TriviaQA	NQ	QuAC	NewsQA
SQuAD	**69.37%**	40.26%	44.70%	13.93%	25.44%
TriviaQA	16.11%	**39.42%**	20.55%	5.65%	8.05%
NQ	47%	36.44%	**68.75%**	18.32%	13.07%
QuAC	32.9%	25.37%	18.68%	**20.62%**	0.53%
NewsQA	36.76%	29.81%	44.80%	6.5%	**48.22%**

Παρατηρήσεις:

- Το πώς προέκυψε η κάθε τιμή του παραπάνω πίνακα εξηγείται στην αντίστοιχη παράγραφο της παραπάνω μελέτης.
- Οι χαμηλές τιμές στην γραμμή και την στήλη του QuAC dataset οφείλονται στην χρήση του background σαν πληροφορία για την απάντηση των ερωτήσεων. Αυτό, εξηγείται αναλυτικά στην παράγραφο 3.7.7 Χρήση του context ως πληροφορία. Επίσης, στην ίδια παράγραφο έχουν δημιουργηθεί η αντίστοιχη γραμμή και η αντίστοιχη στήλη του QuAC χρησιμοποιώντας το context. Η γραμμή αντιστοιχεί στο Table 1 και η στήλη αντιστοιχεί στο table 2 που παρουσιάζονται στην παράγραφο 3.7.7.