Τεχνητή Νοημοσύνη 2
(Homework 4)

Ονοματεπώνυμο: Δημήτριος Σιταράς
Αριθμός μητρώου: 1115201800178
Εξάμηνο: 7ο

Περιεχόμενα

Άσκηση 1

Γενικά

Παρακάτω αναλύω τον καλύτερο classifier που κατάφερα να υλοποιήσω μέσω fine-tuning πάνω στο pretrained BERT-base μοντέλο μετά από πολλούς πειραματισμούς που αναλύονται στην συνέχεια. Αρχικά, εφαρμόζω στα train και validation sets το απαραίτητο preprocessing (cleaning και lemmatize, χρησιμοποιώντας τις ίδιες συναρτήσεις με τις προηγούμενες εργασίες), στη συνέχεια προκειμένου να χρησιμοποιήσω το pre-trained BERT model, μέσω του Bert Tokenizer - 'bert-base-uncased', κάνω tokenize σε όλα τα tweets, προσθέτω τα special tokens σε κάθε tweet του train και του validation set, κάνω padding και trimming (truncation) κάθε tweet ώστε το max length να είναι σταθερό και ίσο με 80, μετατρέποντάς τα στο τέλος στα αντίστοιχα Ids. Έπειτα, κάνω load το μοντέλο BertForSequenceClassification - 'bert-base-uncased', δοκίμασα και το απλό δηλαδή το BertModel - 'bert-base-uncased' χωρίς να παρατηρώ κάποια διαφορά ως προς τα τελικά αποτελέσματα, και ορίζω στο αντίστοιχο config του μοντέλου τον αριθμό των labels να είναι ίσο με 3 για προφανείς λόγους (pro-vax, anti-vax, neutral). Ακολούθως, δημιουργώ δύο iterators για τα train και validation sets αντίστοιχα με το torch DataLoader, για την εκπαίδευση και την αξιολόγηση του μοντέλου. Αυτό βοηθά στην εξοικονόμηση μνήμης κατά τη διάρκεια της εκπαίδευσης, επειδή σε αντίθεση με μια for, χρησιμοποιώντας έναν iterator ολόκληρο το σύνολο δεδομένων δεν χρειάζεται γίνει load στην μνήμη.

Hyperparameters

Για το μοντέλο έχω επιλέξει τα εξής hyperparameters:

- *Batch Size:* Επέλεξα να το ορίσω ίσο με 32, καθώς δοκιμάζοντας (κυρίως) μεγαλύτερα αλλά και μικρότερα μεγέθη τα αποτελέσματα που παρατηρούσα δεν ήταν τα ιδανικά. Μάλιστα, για μεγαλύτερα μεγέθη υπάρχουν και μεγαλύτερες απαιτήσεις μνήμης με αποτέλεσμα η εκτέλεση του προγράμματος να τερμάτιζε ξαφνικά, λόγω της περιορισμένης RAM, δοκιμάζοντας μεγέθη batch πάνω από 64.

- *Learning Rate / lr = 1e-5* $(1 \cdot 10^{-5})$, καθώς για μεγαλύτερες τιμές επιταχυνόταν η εκμάθηση κατά την εκπαίδευση με αποτέλεσμα το μοντέλο να παρουσίαζε overfit και αστάθεια στα αποτελέσματα, ενώ για μικρότερες τιμές το μοντέλο χρειαζόταν περισσότερα epochs για να εκπαιδευτεί.

- *Epochs = 3*, καθώς αρκούν για να εκπαιδευτεί επαρκώς το μοντέλο. Με λιγότερα epochs το μοντέλο δεν εκπαιδευόταν πλήρως πάνω στο training set με

αποτέλεσμα να παρουσίαζε underfit. Ενώ, με περισσότερα epochs το μοντέλο «μάθαινε» υπερβολικά καλά το training set έτσι που να παρουσίαζε overfit.

Loss Function

Χρησιμοποιείται η Cross Entropy (η οποία αποτελεί την καλύτερη επιλογή στην περίπτωση του multiclass classification), της οποία η τιμή υπολογίζεται εσωτερικά του μοντέλου και επιστρέφεται από την αντίστοιχη συνάρτηση που κάνει το prediction.

Optimizer

Χρησιμοποιώ τον AdamW, ο οποίος αποτελεί μια βελτιωμένη εκδοχή του Adam ως προς διάσπαση των βαρών (weight decay). Βέβαια, δοκιμάζοντας και τους δύο αυτούς optimizers (AdamW και Adam) δεν παρατήρησα κάποια διαφορά στα αποτελέσματα.

Gradient Clipping

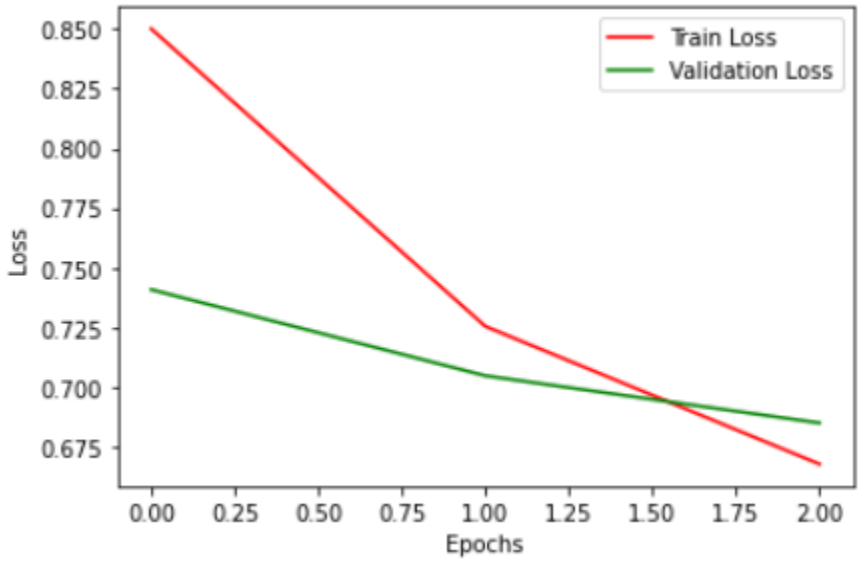
Κατά την διάρκεια της εκπαίδευσης χρησιμοποιώ Gradient Clipping προκειμένου να αποτρέψω «την εξαφάνιση των gradients», έχω δοκιμάσει διάφορες τιμές για το clipping στο εύρος [5,100] χωρίς να παρατηρηθεί κάποια ιδιαίτερη διαφορά, τελικά επέλεξα την τιμή 5. Σε δοκιμές όπου δεν χρησιμοποίησα Gradient Clipping μου αυξήθηκαν υπερβολικά τα train και τα validation losses, σε βαθμό που το μοντέλο να αδυνατούσε να εκπαιδευτεί.

Metrics

Accuracy	71 %
F1-score	73 %
Precision	78 %
Recall	71 %

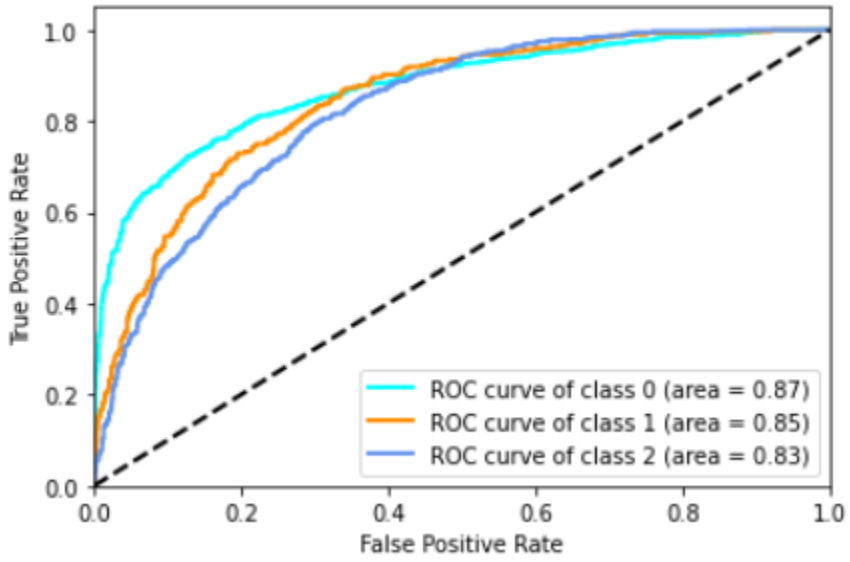
Loss Curve

Όσον αναφορά τώρα την loss curve που προκύπτει από τα loss του train και του validation set, είναι εμφανές πως το μοντέλο δεν παρουσιάζει overfit (αυτό που παρατηρείται στην γραφική είναι αμελητέο) ή underfit, που σημαίνει πως έχει εκπαιδευτεί «σωστά».



ROC Curve

Η ROC δείχνει την απόδοση ενός μοντέλου ταξινόμησης προβάλλοντας την διαγνωστική ικανότητά του. Συνεπώς, όσο πιο κοντά είναι οι καμπύλες της γραφικής παράστασης στην πάνω αριστερή γωνία τόσο μεγαλύτερο True Positive Rate έχει το μοντέλο. Έχω συγχωνεύσει, λοιπόν, 3 ROC curves σε ένα γράφημα για το συγκεκριμένο «πρόβλημα» (multiclass classification), κάθε καμπύλη αντιστοιχεί και επομένως αντιπροσωπεύει κάθε κλάση (0,1,2). Παρατηρώντας, λοιπόν, το γράφημα του μοντέλου μου συμπεραίνω πως αποδίδει καλύτερα για τα tweets που ανήκουν στην κλάση 0 (neutral) και «χειρότερα» αυτά που ανήκουν στις υπόλοιπες 2 κλάσεις (anti-vax και vax) με μικρή απόκλιση.



Παρατηρήσεις

Συγκρίνοντας το συγκεκριμένο μοντέλο με τα καλύτερα μοντέλα των τριών προηγούμενων εργασιών, παρατηρώ πως με βάση τα score των metrics:

	Hw1 (SoftMax Regression)	**Hw2 (Feed-Forward Neural Networks)**	**Hw3 (bidirectional stacked RNN with LSTM/GRU and Attention Layer)**	**Hw4 (fine-tuning BERT base-uncased model)**
Accuracy	71 %	66 %	70 %	71 %
F1-score	71 %	61 %	70 %	73 %
Precision	71 %	58 %	69 %	78 %
Recall	71 %	66 %	72 %	71 %

το pre-trained BERT model της συγκεκριμένης εργασίας για τα ίδια datasets αποδίδει καλύτερα, κάνοντας καλύτερη κατηγοριοποίηση των tweets.

Αξίζει να σημειωθεί πως η εκπαίδευση του BERT-base model είναι υπερβολικά χρονοβόρα και πραγματοποιήθηκε με χρήση GPU που παρέχει το Google Colab (καθώς είναι πολύ πιο γρήγορη από την CPU που χρησιμοποιήθηκε στις προηγούμενες εργασίες) προκειμένου να καταφέρω να εκπαιδεύσω το αντίστοιχο μοντέλο και να πάρω τα αποτελέσματα που ανέφερα παραπάνω σε σχετικά λογικούς χρόνους. Ειδικότερα, για να ολοκληρωθεί ένα epoch χρειάζεται περίπου μισή ώρα (30 λεπτά), συνεπώς όλη η εκπαίδευση του μοντέλου εφόσον επέλεξα 3 epochs διαρκεί περίπου 90 λεπτά. Δυστυχώς, δεν κατάφερα να μειώσω τον χρόνο. Επιδίωξα να μειώσω περισσότερο το μέγεθος των tweets στο στάδιο του preprocessing (εφαρμόζοντας και stemming), με σκοπό να επιταχυνθεί η διαδικασία εκπαίδευσης, όμως τελικά δεν κατάφερα να μειώσω σημαντικά τον χρόνο (~25 λεπτά /epoch), χάνοντας παράλληλα σημαντική πληροφορία από τα tweets με αποτέλεσμα τα score των metrics να μην είναι τόσο υψηλά.

Άσκηση 2

Γενικά

Παρακάτω αναλύω το καλύτερο question-answering μοντέλο που κατάφερα να υλοποιήσω μέσω fine-tuning πάνω στο pretrained BERT-base μοντέλο, χρησιμοποιώντας ως dataset το SQuAD 2.0. Ο πειραματισμός και η υλοποίηση έγιναν στο Google Colab και στο Kaggle (το Colab μετά από 4-5 ώρες χρήσης με μπλόκαρε και δεν με άφηνε να χρησιμοποιώ την GPU για το επόμενο 12ωρο, επομένως έπρεπε να βρω μια άλλη πλατφόρμα που θα με εξυπηρετεί, προκειμένου να τελειώσω την εργασία, έτσι κατέληξα στο Kaggle, το οποίο προσφέρει 36 ώρες συνεχόμενης χρήσης της GPU ανά εβδομάδα).

Αρχικά, η υλοποίηση μου όσον αφορά το preprocessing του SQuAD 2.0 dataset βασίστηκε στην ιστοσελίδα που δόθηκε στην εκφώνηση της εργασίας, huggingface (και συγκεκριμένα εδώ).

Σε πρώτο στάδιο, λοιπόν, αποθηκεύω σε αντίστοιχες λίστες τα κείμενα, τις ερωτήσεις και τις απαντήσεις του κάθε αρχείου .json (train-v2.0.json: περιέχει τα δεδομένα στα οποία θα γίνει train το μοντέλο, dev-v2.0.json περιέχει τα δεδομένα πάνω στα οποία θα γίνει το evaluation του μοντέλου), προκείμενου στη συνέχεια να κατασκευάσω από αυτές στη συνέχεια το train και το validation set.

Έπειτα, μέσω της συνάρτησης add_end_idx(…), «βρίσκω» για κάθε απάντηση τη θέση του χαρακτήρα όπου αυτή τελειώνει στο αντίστοιχο κείμενο και την αποθηκεύω, κάποιες φορές όμως οι απαντήσεις του SQuAD απέχουν έναν ή δύο χαρακτήρες σε σχέση με το κείμενο, έτσι το διαχειρίζομαι ανάλογα.

Ακολούθως, χρησιμοποιώ τον DistilBertTokenizerFast - distilbert-base-uncased, δίνοντας ως ορίσματα κάθε φορά τα αντίστοιχα κείμενα και τις αντίστοιχες ερωτήσεις που αφορούν το train και το validation. Έτσι, μέσω του tokenizer, κάνω σ ‘αυτά tokenize, προσθέτω τα special tokens, κάνω padding και trimming (truncation) κάθε κείμενο/ερώτηση ώστε το max length να είναι σταθερό και ίσο με 512 και τέλος τα μετατρέπω σε IDs, δημιουργώντας τελικά δύο encodings (δηλαδή dictionaries που έχουν ως keys: ‘input ids’ και ’attention mask’) που αφορούν το train και το validation. Επίσης, δοκίμασα και τον απλό BertTokenizer, όμως λόγω του μεγάλου όγκου των κειμένων και των ερωτήσεων που του έδινα ως ορίσματα καθυστερούσε υπερβολικά συγκριτικά με τον DistilBertTokenizerFast (επίσης κατανάλωνε πολύ RAM ως επακόλουθο ορισμένες φορές να γίνεται αυτόματο restart του notebook). Ο DistilBertTokenizerFast είναι παρόμοιος με τον BertTokenizerFast, τον οποίο μάλιστα δοκίμασα, προτιμώντας τελικά τον DistilBertTokenizerFast καθώς είναι ελάχιστα πιο γρήγορος. Τονίζω πως δοκίμασα διάφορα μεγέθη για το max length, από 50 έως και 512, (512 είναι το μέγιστο που υποστηρίζει ο tokenizer) καταλήγοντας τελικά στην παραπάνω επιλογή (**512**). Με μικρότερα μεγέθη χανόταν αρκετή πληροφορία με

αποτέλεσμα το μοντέλο να μην εκπαιδευόταν σωστά (αφού τα κείμενα και οι ερωτήσεις «κοβόντουσαν» υπερβολικά, χωρίς να έχουν νόημα), έτσι που προεκύπταν πολύ χειρότερα scores. Ενώ με max length ίσο με 512, αξιοποιώ στο μέγιστο την δυνατότητα που μου προσφέρει ο tokenizer, χάνοντας όσο γίνεται λιγότερη πληροφορία από τα κείμενα και τις ερωτήσεις.

Στη συνέχεια, θέλω να προσθέσω στα παραπάνω encodings που επιστρέφει ο tokenizer τα "start-end token positions" των αντίστοιχων απαντήσεων. Συνεπώς, με την συνάρτηση add_token_positions(…), προσθέτω τα keys: 'start positions' and 'end positions', στα dictionaries/encodings του train και του validation. Καθένα από αυτά έχει μια λίστα που περιέχει τα start/end token positions της απάντησης που αντιστοιχεί στα ζεύγη κειμένου-ερώτησης. Άρα, τελικά τα dictionaries/encodings θα περιέχουν τα εξής keys: 'input ids', 'attention mask', 'start positions', 'end positions'.

Τέλος, υλοποιώ μια custom class η οποία αποτελεί υποκλάση της κλάσης torch.utils.data.Dataset (προσθέτοντας τρεις συναρτήσεις για την διαχείριση των dictionaries/encodings), προκειμένου να κατασκευάσω τα τελικά dataset (train και validation set) που θα χρησιμοποιηθούν στην συνέχεια για την εκπαίδευση και την αξιολόγηση του μοντέλου.

Μοντέλο

Χρησιμοποιώ το pre-trained BertForQuestionAnswering – 'bert-base-uncased' model. Βέβαια, δοκίμασα και το DistilBertForQuestionAnswering μοντέλο, το οποίο δεν είναι τόσο πολύπλοκο όσο το BertForQuestionAnswering (έχει λιγότερες παραμέτρους), με αποτέλεσμα να είναι αισθητά πιο γρήγορο στην απάντηση των ερωτήσεων, όμως «χάνει» σε ακρίβεια και συγκεκριμένα δεν καταφέρνει να δώσει σωστές απαντήσεις σε μεγάλο μέρος του validation set (πετυχαίνοντας τελικά συνολικό f1 score κάτω από 20% τις περισσότερες φορές).

Hyperparameters

Για το μοντέλο έχω επιλέξει τα εξής hyperparameters:

- *Batch Size: 16*, διότι για μεγαλύτερα μεγέθη που δοκίμασα (για το δεδομένο max length = 512) η GPU έσκαγε, ενώ για μικρότερα τα αποτελέσματα ήταν τα ίδια μόνο που η εκπαίδευση του μοντέλου καθυστερούσε πολύ παραπάνω. Μάλιστα παρατήρησα μια «αντιστρόφως ανάλογη σχέση» μεταξύ του batch size και την max length παράμετρο του tokenizer, δηλαδή όσο αυξάνω το batch size πρέπει να μειώνω ανάλογα και το max length και αντίστροφα, διαφορετικά γεμίζει η RAM της GPU (η GPU σκάει) κατά την εκπαίδευση του μοντέλου με

αποτέλεσμα να είναι αδύνατον να εκτελεστεί το αντίστοιχο cell του jupyter notebook.

- *Learning Rate / lr = 1e-5* $(1 \cdot 10^{-5})$, καθώς για μεγαλύτερες τιμές επιταχυνόταν η εκμάθηση κατά την εκπαίδευση με αποτέλεσμα το μοντέλο να παρουσίαζε overfit και αστάθεια στα αποτελέσματα, ενώ για μικρότερες τιμές το μοντέλο χρειαζόταν περισσότερα epochs για να εκπαιδευτεί.

- *Epochs = 2*, αρκούν για να εκπαιδευτεί επαρκώς το μοντέλο. Με ένα μόνο epoch το μοντέλο δεν εκπαιδευόταν πλήρως πάνω στο training set με αποτέλεσμα να παρουσίαζε underfit. Ενώ, με περισσότερα epochs το μοντέλο «μάθαινε» υπερβολικά καλά το training set έτσι που να παρουσίαζε overfit.

Optimizer

Χρησιμοποιώ τον AdamW, ο οποίος αποτελεί μια βελτιωμένη εκδοχή του Adam ως προς διάσπαση των βαρών (weight decay). Βέβαια, δοκιμάζοντας και τους δύο αυτούς optimizers (AdamW και Adam) δεν παρατήρησα κάποια διαφορά στα αποτελέσματα.

Gradient Clipping

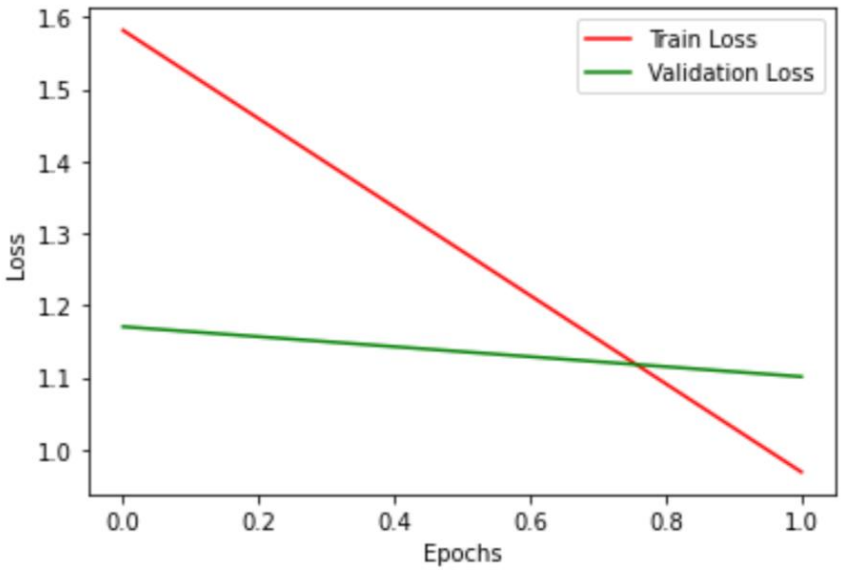
Κατά την διάρκεια της εκπαίδευσης χρησιμοποιώ Gradient Clipping προκειμένου να αποτρέψω «την εξαφάνιση των gradients», έχω δοκιμάσει διάφορες τιμές για το clipping στο εύρος [5,100] χωρίς να παρατηρηθεί κάποια ιδιαίτερη διαφορά, τελικά επέλεξα την τιμή 5. Σε δοκιμές όπου δεν χρησιμοποίησα Gradient Clipping μου αυξήθηκαν υπερβολικά τα train και τα validation losses, σε βαθμό που το μοντέλο να αδυνατούσε να εκπαιδευτεί.

Metrics

Exact	27 %
F1-score	32 %
Total	11873
HasAns exact	78 %
Recall	71 %
HasAns f1	51 %
NoAns exact	13 %
NoAns f1	13 %
NoAns total	5945

Loss Curve

Όσον αναφορά τώρα την loss curve που προκύπτει από τα loss του train και του validation set, είναι εμφανές πως το μοντέλο δεν παρουσιάζει overfit (αυτό που παρατηρείται στην γραφική είναι αμελητέο) ή underfit, που σημαίνει πως έχει εκπαιδευτεί «σωστά».



Παρατηρήσεις

Πειραματίστηκα συμπεριλαμβάνοντας και τα impossible questions, δηλαδή τις ερωτήσεις του SQuAD 2.0. που δεν έχουν απάντηση, στο train data set. Τότε το μέγεθός του σχεδόν διπλασιαζόταν με αποτέλεσμα η διαδικασία της εκπαίδευσης να καθυστερούσε πολύ περισσότερο, βελτιώνοντας όμως ελάχιστα ως και καθόλου τα τελικά scores. Τελικά, δεν τα συμπεριέλαβα, διότι το μοντέλο είναι προφανές πως δεν μπορεί να εκπαιδευτεί πάνω σε ερωτήσεις που δεν έχουν απαντήσεις.

Αξίζει να σημειωθεί πως η εκπαίδευση του BertForQuestionAnswering model είναι υπερβολικά χρονοβόρα και πραγματοποιήθηκε με χρήση GPU. Για να ολοκληρωθεί ένα epoch χρειάζεται περίπου μια μισή ώρα (1 ώρα και 30 λεπτά), συνεπώς όλη η εκπαίδευση του μοντέλου εφόσον έχω επιλέξει 2 epochs διαρκεί λίγο παραπάνω από 3 ώρες. Ο χρόνος αυτός είναι σχεδόν αδύνατον να μειωθεί, χωρίς να μεταβληθούν τα scores προς το χειρότερο, γιατί πρέπει να μειωθεί το max_length στον tokenizer, ώστε να αυξηθεί το batch size, χάνοντας έτσι αρκετή πληροφορία. Επίσης, δοκίμασα να μειώσω και το dataset του SQuAD 2.0, χρησιμοποιώντας, για παράδειγμα το 70% των δεδομένων, προκειμένου να μπορέσω να αυξήσω το batch size (χωρίς να «σκάει» η GPU), μειώνοντας έτσι τον συνολικό χρόνο εκπαίδευσης, όμως η απόδοση του ήταν χειρότερη, διότι το μοντέλο δεν εκπαιδευόταν σε όλο το σύνολο των δεδομένων αλλά σε ένα μέρος αυτού.

Η εκπαίδευση και η αξιολόγηση του τελικού μοντέλου πραγματοποιήθηκε στο Kaggle, γιατί το Collab με μπλόκαρε λόγω των μικρών χρονικών πλαισίων και δεν μπορούσα να ολοκληρώσω ούτε καν την εκπαίδευση του μοντέλου.

Άσκηση 3

Πίνακας

		Evaluated on				
		SQuAD	**TriviaQA**	**NQ**	**QuAC**	**NewsQA**
Fine-tuned on	**SQuAD**	31.7 %	19.8 %	26.3 %	12.5 %	16.2 %
	TriviaQA	18.5 %	28.6 %	21 %	8.5 %	13 %
	NQ					
	QuAC	18.0 %	11.7 %	18.6 %	18.9 %	7.7 %
	NewsQA	25.1 %	15.1 %	29.9 %	9.8 %	30.8 %

Παρατηρήσεις

Για όλα τα Datasets (TriviaQA, QuAC, NewsQA) που έκανα fine-tuned χρησιμοποίησα το ίδιο ακριβώς μοντέλο που χρησιμοποίησα για να το SQuAD 2.0, το οποία ανέλυσα παραπάνω. Έκανα την επιλογή αυτή επειδή ό,τι διαφορετικό και αν δοκίμασα (αλλάζοντας κυρίως τις υπερπαραμέτρους και το max_length του tokenizer) τα αποτελέσματα που προέκυπταν ήταν χειρότερα, επίσης παρατήρησα πως τα datasets αυτά είναι παρόμοια με το Squad, επομένως έκρινα πως το μοντέλο που υλοποίησα προηγουμένως είναι το ιδανικότερο. Το μόνο που άλλαξα στο QuAC και στο NewsQA είναι το learning rate που το έκανα από 1e-5 σε 2e-5 και από 1e-5 σε 3e-5 αντίστοιχα, καθώς τα loss ήταν πολύ υψηλά και σύμφωνα με το αντίστοιχο loss curve δεν επιτυγχανόταν σύγκλιση.

Σημειώνω πως χρησιμοποίησα το 70% του train dataset του TriviaQA ώστε να μπορέσω να εκπαιδεύσω το μοντέλο, χρησιμοποιώντας παραπάνω από το 70 % «η GPU έσκαγε», πράγμα το οποίο ήταν αδύνατον να αντιμετωπίσω με μείωση ή αύξηση του batch size και του max length αναλόγως.

Ομοίως, χρησιμοποίησα το 70% του train dataset του QuAC ώστε να μπορέσω να εκπαιδεύσω το μοντέλο σε λογικό χρονικό διάστημα (2 ώρες συνολικά,1 ανά epoch), χρησιμοποιώντας ολόκληρο το dataset η μια εποχή διαρκούσε παραπάνω από 5 ώρες (εκτός και προέκυψε κάποιο bug στο Kaggle) με αποτέλεσμα να σταματάω την εκτέλεση.

Ομοίως, χρησιμοποίησα το 60% του train dataset του NewsQA ώστε να μπορέσω να εκπαιδεύσω το μοντέλο σε λογικό χρονικό διάστημα (2 ώρες συνολικά,1 ανά epoch), , χρησιμοποιώντας παραπάνω από το 60 % «η GPU έσκαγε», πράγμα το οποίο ήταν

αδύνατον να αντιμετωπίσω με μείωση ή αύξηση του batch size και του max length αναλόγως.

Η εκπαίδευση όλων των τελικών μοντέλων πραγματοποιήθηκε στο Kaggle, ενώ το evaluate αυτών πάνω στα υπόλοιπα datasets έγινε στο Collab (μετά το πέρας της εκπαίδευσης στο Kaggle αποθήκευα τα μοντέλα με torch.save() ώστε να τα μεταφέρω στο Collab για το evaluate).

Δεν κατάφερα να κάνω το μοντέλο train με το NQ dataset, γιατί είναι 41 GB μόνο το train dataset και είναι αδύνατο να χωρέσει στο Kaggle (το οποίο έχει συνολικό χώρο 20 GB). Οριακά χωράει σε ένα άδειο Google Drive, όμως δεν υπάρχει μετά κάποιος τρόπος να το μεταφέρω στο Kaggle χωρίς να το αποθηκεύσω. Ωστόσο, δοκίμασα να κάνω την εκπαίδευση του μοντέλου στο Colab, αλλά μετά από λίγο χρόνο μου διέκοπτε την δυνατότητα χρήσης της GPU, με αποτέλεσμα να μην καταφέρνω να πάρω ποτέ αποτέλεσμα, ούτε για μια εποχή (λόγω του τεράστιου όγκου των δεδομένων).

Γενικά, δεν κατάφερα να πετύχω τόσα υψηλά scores όσο το αντίστοιχο paper που δίνεται στην εκφώνηση, μάλιστα υπάρχει αρκετά μεγάλη απόκλιση, αυτό ίσως οφείλεται στο γεγονός των περιορισμένων πόρων που είχα στην διάθεση μου (δωρεάν χρήση του Colab και του Kaggle). Ίσως με κάποιο premium πλάνο από τις αντίστοιχες πλατφόρμες (Colab και Kaggle), το οποίο θα μου έδινε την δυνατότητα να χρησιμοποιήσω πολύ παραπάνω πόρους και διαφορετική GPU, να κατάφερνα να πετύχω πιο υψηλά scores.

Τέλος, συμπεριλαμβάνω στο παραδοτέο zip αρχείο όλα τα μοντέλα που υλοποίησα (που έκανα δηλαδή fine-tuned), το κάθε ένα βρίσκεται σε ξεχωριστό Jupiter notebook, συγκεκριμένα:

- Fine-tuned on SQuAD: *hw4_Squad_part2part3.ipynb*
- Fine-tuned in TriviaQA: *hw4_Trivia_part3.ipynb*
- Fine-tuned in QuAC: *hw4_quac_part3.ipynb*
- Fine-tuned in NewsQA: *hw4_newsqa_part3.ipynb*