ΥΣ19 Artificial Intelligence II

Τέταρτη Εργασία

Ονοματεπώνυμο: Απόστολος Καρβέλας

Α.Μ.: 1115201800312

1. Το πρώτο υποερώτημα βρίσκεται στο ipynb αρχείο 1115201800312_hw4_1
2. Το δεύτερο υποερώτημα είναι στο 1115201800312_hw4_2
3. Ενώ για το τρίτο υποερώτημα, το training του μοντέλου στο SQuAD 2.0 dataset και το evaluation στα SQuAD 2.0 & TriviaQA γίνεται στο τέλος του 1115201800312_hw4_2 και το training σε TriviaQA και evaluation στα SQuAD 2.0 & TriviaQA είναι υλοποιημένο στο 1115201800312_hw4_3

Πρώτο υποερώτημα

ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ

Για να γίνει εκτέλεση των πειραμάτων πρέπει πρώτα να έχουν τρέξει τα αρχικά κελιά και συγκεκριμένα τα imports, preprocessing & print functions και το read files. Στην συνέχεια, υπάρχουν ξεχωριστά κελιά για κάθε πείραμα με τον αντίστοιχο τίτλο του και στο τέλος υπάρχει το καλύτερο μοντέλο, δηλαδή εκείνο με τα βέλτιστα αποτελέσματα. Για να τρέξει ένα κελί με διαφορετικά δεδομένα πρέπει πρώτα να αλλάξει τα paths των αρχείων που βρίσκονται στο read files segment.

ΕΙΣΑΓΩΓΗ

Στην εργασία αυτή θα ασχοληθούμε με το fine-tuning του pretrained μοντέλου BERT και συγκεκριμένα θα χρησιμοποιήσουμε:

- Bert-base-uncased Tokenizer encode plus του Bert
- BertForSequenceClassification bert-base-uncased
- AdamW optimizer
- Preprocessing
- Scheduler
- Clipping

Fine-tuning BERT

Το BERT είναι ένα νευρωνικό δίκτυο που αποτελείτε από 12 layers, 12 attention heads και πολλούς παραμέτρους, οπότε αν προσπαθήσουμε να εκπαιδεύσουμε ένα τέτοιο μοντέλο με τα δικά μας δεδομένα από την αρχή θα προκύψει προφανώς overflowing. Οπότε η λύση

είναι να χρησιμοποιήσουμε ένα pretrained μοντέλο που είναι trained σε πολύ μεγάλο dataset και συγκεκριμένα στα Wikipedia και Book Corpus. Ύστερα θα κάνουμε train το μοντέλο για πολύ λιγότερα δεδομένα και για μικρό αριθμό από epochs.

Διαδικασία

Στην αρχή του κώδικα αφού έχουμε διαβάσει τα αρχεία για training και validation πρέπει να τα κάνουμε tokenized για να τα καταλαβαίνει το μοντέλο, για αυτόν τον λόγω θα χρησιμοποιήσουμε τον bert-base-uncased tokenizer του BERT και συγκεκριμένα τον encoder plus που μαζί με το tokenization ταυτόχρονα προσθέτει τα special tokens, δημιουργεί το attention mask και κάνει padding στον μέγιστο αριθμό από επιτρεπόμενα στοιχεία. Η μεγαλύτερη πρόταση στο dataset που χρησιμοποιούμε είναι πάνω από 300 λέξεις, όμως αν βάλουμε batch_size να ισούται με 16, δεν θα φτάσει η μνήμης που μας προσφέρουν στην CUDA. Οπότε θα πειραματιστούμε με max_length 220, το οποίο σημαίνει ότι κατά το tokenization των κειμένων υπάρχει η πιθανότητα να κόψει ένα κομμάτι στο τέλος. Το πρόβλημα αυτό θα προσπαθήσουμε να το λύσουμε μετά με την χρήση preprocessing ώστε να μειωθεί η μεγαλύτερη πρόταση σε 205.

Αφού κάνουμε tokenization σε όλες τις προτάσεις των 2 dataset, φορτώνουμε τα στοιχεία σε Dataloader αφού πρώτα τα έχουμε βάλει σε TensorDataset και κάνουμε το μοντέλο να ισούται με το bert-base-uncased BertForSequenceClassification με αριθμό από labels 3.

BERT with learning rate 1e-5

Σαν optimizer έχουμε τον AdamW με learning rate 1e-5 και θα κάνουμε train-validation το μοντέλο για 3 εποχές.

Training:

F1 score for

class 0 : 0.9408506710217994 class 1 : 0.8144123534458108 class 2 : 0.9275314086030452

Total f1 score: 0.8511099001095195

Validation:

F1 score for

class 0 : 0.8706256627783668 class 1 : 0.7186147186147186 class 2 : 0.8903614457831326

Total f1 score: 0.755492870748025

Παρατηρούμε ότι ακόμα και με τόσο απλούς παραμέτρους, δηλαδή χωρίς την χρήση preprocessing και άλλων τεχνικών που θα χρησιμοποιήσουμε μετά επιστρέφει πολύ καλό f1 score τόσο για το training όσο και για το validation. Αυτό συμβαίνει καθώς το μοντέλο μας είναι pretrained οπότε και με μικρό learning rate απλά θα βασίζεται περισσότερο στο dataset που έχει εκπαιδευτεί παρά το fine-tuning στο δικό μας dataset. Για αυτόν τον λόγω

θα προσπαθήσουμε να αυξήσουμε το learning rate για να βρούμε και καλύτερο αποτέλεσμα.

BERT with learning rate 2e-5

Οπότε με την αύξηση του learning rate σε 2e-5 έχουμε:

Training:

F1 score for

class 0 : 0.9579432723208048 class 1 : 0.8926282051282052 class 2 : 0.9560217056774923

Total f1 score: 0.9028925336650501

Validation:

F1 score for

class 0 : 0.8813025210084033 class 1 : 0.758909853249476 class 2 : 0.876829268292683

Total f1 score: 0.7615444495856143

Από τα δεδομένα βλέπουμε ότι με υψηλότερο learning rate κάνει καλύτερο tuning στο δικό μας dataset οπότε και επιστρέφει και κάπως μεγαλύτερο f1 score για τα datasets χωρίς να προκύπτει overflowing. Οπότε θα κρατήσουμε μέχρι στιγμής το learning rate αυτό.

BERT with lr 2e-5 and preprocessing

Όπως αναφέρθηκε παραπάνω ο tokenizer δέχεται max length 220 ενώ κάποιες προτάσεις έχουμε μεγαλύτερο αριθμό λέξεων από αυτό. Οπότε θα χρησιμοποιήσουμε preprocessing για να μειώσει τους χαρακτήρες της μεγαλύτερης πρότασης σε 205. Άρα παρατηρούμε:

Training:

F1 score for

class 0 : 0.9503166783954962 class 1 : 0.880669546436285 class 2 : 0.9497270431027459

Total f1 score: 0.8886012675736531

Validation:

F1 score for

class 0 : 0.8801261829652998 class 1 : 0.6575963718820861 class 2 : 0.8910295003010236

Total f1 score: 0.7489275231063723

Η προσθήκη preprocessing έχει ως αποτέλεσμα να μειωθούν τα f1 scores κατά λίγο, που σημαίνει ότι προφανώς δεν επηρεαζόταν τόσο από μερικές χαμένες λέξεις σε λίγες προτάσεις και μπορεί να μάθαινε συγκεκριμένα για αυτές τις προτάσεις. Ωστόσο, το preprocessing αφού βγάζει τις άχρηστες λέξεις, σε γενική περίπτωση, δηλαδή σε διαφορετικό dataset, είναι λογικό να επιστρέφει καλύτερα αποτελέσματα.

BERT with scheduler

Μπορούμε να ορίσουμε ένα scheduler στο οποίο το learning rate ενημερώνεται κατά τη διάρκεια της εκπαίδευσης με βάση την εποχή που βρίσκεται. Οπότε έχουμε:

Training:

F1 score for

class 0 : 0.948614929160499 class 1 : 0.8628634119583105 class 2 : 0.9428360534733045

Total f1 score: 0.8786465675481516

Validation:

F1 score for

class 0 : 0.8894681960375391 class 1 : 0.6931567328918322 class 2 : 0.8849878934624698

Total f1 score: 0.7625740544883226

Παρατηρούμε ότι με την χρήση του scheduler τα f1 scores αυξάνονται λίγο οπότε βοηθάει το μοντέλο κατά την διάρκεια του training όσο περνάει να μειώνει λίγο τον ρυθμό εκμάθησης. Άρα και το κρατάμε.

BERT with AdamW epsilon and clipping.

Το eps του AdamW είναι ένα μικρό νούμερο που προστίθεται στον παρονομαστή για τη βελτίωση του αριθμητικού stability.

Το exploding gradient γίνεται όταν συσσωρεύονται μεγάλα gradients σφάλματος και έχουν ως αποτέλεσμα πολύ μεγάλες ενημερώσεις στα βάρη του μοντέλου κατά την εκπαίδευση. Αυτό έχει ως αποτέλεσμα το μοντέλο να μην μπορεί να μάθει νέα δεδομένα αφού τα προηγούμενα υπερτερούν πολύ αναλογικά στο βάρος. Μια λύση στο πρόβλημα αυτό είναι να χρησιμοποιήσουμε gradient clipping, κατά το οποίο αφού επιλέξουμε ένα threshold τότε

όλες οι νόρμες που είναι υψηλότερες ή χαμηλότερες του threshold κατεβαίνουν στην νόρμα του βάλαμε.

Ενώ το vanishing gradient συμβαίνει κατά το backpropagation όταν τα βάρη των προηγουμένων κόμβων χάνονται, δηλαδή οι τιμές φτάνουν κοντά στο μηδέν, οπότε το μοντέλο δεν μπορεί να θυμάται παλιά δεδομένα. Αυτό συμβαίνει επειδή η κλίση της συνάρτησης απώλειας μειώνεται εκθετικά με το χρόνο. Μια αρχική λύση είναι πάλι το gradient clipping για τον λόγω που προαναφέραμε.

Οπότε θα δοκιμάσουμε το προηγούμενο πείραμα αλλά αυτήν την φορά θα χρησιμοποιήσουμε gradient clipping για να μειώσουμε τα προβλήματα αυτά. Άρα κάνουμε το threshold να ισούται με 1 και έχουμε τα εξής αποτελέσματα:

Training:

F1 score for

class 0 : 0.9500950235799254 class 1 : 0.8702997275204359 class 2 : 0.9427493438320209

Total f1 score: 0.8816385010020971

Validation:

F1 score for

class 0 : 0.8830624016780283 class 1 : 0.7017543859649124 class 2 : 0.880243161094225

Total f1 score: 0.7548339815297995

Βλέπουμε ότι τα f1 scores αυξάνονται λίγο από το προηγούμενο πείραμα άρα βοηθάει λίγο η αριθμητική σταθερότητα που προσφέρει το eps αλλά λογικά το clipping δεν θα επηρεάζει πολύ το μοντέλο γιατί είναι pretrained οπότε είναι δύσκολο να εμφανίσει vanishing ή exploding gradient.

Δεύτερο υποερώτημα

ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ

Για να γίνει εκτέλεση των πειραμάτων πρέπει πρώτα να έχουν τρέξει τα αρχικά κελιά και συγκεκριμένα τα imports, preprocessing & print functions και το read files. Στην συνέχεια, υπάρχουν ξεχωριστά κελιά για κάθε πείραμα με τον αντίστοιχο τίτλο του και στο τέλος υπάρχει το καλύτερο μοντέλο, δηλαδή εκείνο με τα βέλτιστα αποτελέσματα.

ΕΙΣΑΓΩΓΗ

Τώρα θα ασχοληθούμε με το fine-tuning του pretrained μοντέλου BERT για question-answering στο SQuAD 2.0 dataset και θα χρησιμοποιήσουμε:

- Bert-base-uncased AutoTokenizer του Bert
- BertForQuestionAnswering bert-base-uncased
- AdamW optimizer
- Scheduler
- Clipping

Διαδικασία

Στην αρχή του κώδικα αφού έχουμε διαβάσει το SQuAD 2.0 dataset για training και validation πρέπει να τα πάρουμε τις χρήσιμες πληροφορίες, οπότε τρέχουμε τα datasets και προσθέτουμε στις λίστες για κάθε απάντηση, την ερώτηση και το κείμενο που του αντιστοιχούν. Στις απαντήσεις δεν δίνει το τέλος της απάντησης, οπότε μπορούμε να το βρούμε προσθέτοντας το start index με το μέγεθος της απάντησης. Στην συνέχεια, κάνουμε tokenize για να τα καταλαβαίνει το μοντέλο, έτσι θα χρησιμοποιήσουμε τον bert-base-uncased autotokenizer του BERT. Όπως στο προηγούμενο ερώτημα αν βάλουμε batch_size να ισούται με 16, δεν θα φτάσει η μνήμης που μας προσφέρουν στην CUDA. Οπότε θα πειραματιστούμε με max_length 220, το οποίο σημαίνει ότι κατά το tokenization των κειμένων υπάρχει η πιθανότητα να κόψει ένα κομμάτι στο τέλος. Στα επόμενα πειράματα θα αυξήσουμε το max_length και θα μειώσουμε το batch_size. Μετά από το tokenization και στα 2 datasets βρίσκουμε τα καινούργια start και end positions διατρέχοντας τα datasets και μέσω της εντολής char_to_token βρίσκει την πραγματική αρχή και τέλους της ερώτησης.

Στην συνέχεια φορτώνουμε τα στοιχεία σε Dataloader αφού πρώτα τα έχουμε βάλει σε Dataset και κάνουμε το μοντέλο να ισούται με το bert-base-uncased BertForQuestionAnswering. Αφού έχουμε κάνει train το μοντέλο μέσω του train dataloader πρέπει να βρούμε το validation του μοντέλου. Για να γίνει αυτό τρέχουμε το validation dataset και για κάθε ερώτηση κάνουμε tokenize το κείμενο και την ερώτηση και βρίσκει το prediction του μοντέλου. Μετά αφού υπολογίσει το logits start and end index, βρίσκουμε το prediction answer και με το compute_f1 συνάρτηση επιστρέφει το ολικό f1 σε όλο το set.

BERT base for Question Answering

Στον tokenizer έχουμε max length 220 και batch size 16. Σαν optimizer χρησιμοποιούμε learning rate 4e-5 και eps 5e-9. Οπότε στο κελί με το validation επιστρέφει f1 score 0.55. Το φαινόμενο γίνεται λόγω της έλλειψης πληροφορίας στο μοντέλο αφού εντοπίζει μόνο τις πρώτες 220 λέξεις του κειμένου άρα δεν καταλαβαίνει πολλές και από τις ερωτήσεις. Οπότε για να μην υπάρχει αυτό το πρόβλημα θα προσπαθήσουμε με DistilBert καθώς δεν χρειάζεται τόσο μνήμη, έχει λιγότερες παραμέτρους από το bert-base-uncased και τρέχει πιο γρήγορα. Άρα μπορούμε να φορτώσουμε περισσότερα στοιχεία για να μάθει καλύτερα.

DistilBert for Question Answering

Για learning rate 2e-5 και batch size 450 παρατηρούμε ότι έχει f1 0.47 οπότε το μοντέλο δεν μαθαίνει είτε λόγω χαμηλού learning rate είτε χαμηλού αριθμού από στοιχεία εισαγωγής.

Τρίτο υποερώτημα

Στο τρίτο ερώτημα μας έχει ζητηθεί να πειραματιστούμε με το fine tuning διαφορετικών datasets και να τα κάνουμε evaluate με άλλα. Οπότε για αρχή θα δοκιμάσουμε να φορτώσουμε το triviaqa dataset. Για να γίνει αυτό πρέπει πρώτα να μετατρέψουμε το dataset σε μορφής SQuAD για να μπορέσουμε να χρησιμοποιήσουμε το μοντέλο από το προηγούμενο ερώτημα. Αφού κατεβάσουμε το dataset καλούμε την συνάρτηση triviaqa_to_squad_format ώστε να γίνει στην μορφή αυτή. Στην συνέχεια καλούμε το πρόγραμμα για squad με τιμές:

- Max length = 350 γιατί δεν είχε χώρο η cpu για μεγαλύτερο αριθμό
- Batch size = 4
- Learning rate = 2e-5
- Eps = 5e-9

Το μοντέλο που έχει γίνει fine-tuned σε SQuAD όπως είδαμε και παραπάνω, όταν το κάνουμε evaluation πάλι στο SQuAD επιστρέφει f1 score 0.55 ενώ για evaluation στο Triviaqa παρατηρούμε ότι υπολογίζει 0.14 f1 score.

Ενώ το μοντέλο το οποίο είναι fine-tunes σε Triviaqa, όταν το κάνουμε evaluate με το ίδιο dataset υπολογίζει f1 score 0.23 ενώ για evaluation στο SQuAD έχει f1 score 0.18.