A. TRANSITION-BASED DEPENDENCY PARSER

Δίνεται κώδικας που υλοποιεί έναν dependency parser χρησιμοποιώντας το PyTorch[1]. Η προσέγγιση βασίζεται σε μεγάλο βαθμό στην εργασία των Chen & Manning (2014)[2] που προτείνουν έναν transition-based parser όπου η απόφαση για το αν υπάρχει ή όχι σχέση εξάρτησης μεταξύ δύο λέξεων βασίζεται σε ένα νευρωνικό δίκτυο. Στη συγκεκριμένη υλοποίηση ανιχνεύονται μόνο unlabeled dependencies, δηλ. δεν μας ενδιαφέρει ο τύπος (π.χ. nsubj, dobj) της εξάρτησης. Πιο συγκεκριμένα, κάθε περίπτωση αναπαριστάται με βάση ένα πλήθος features που αφορά στις λέξεις και τα μέρη-του-Λόγου από το stack και το buffer ενώ η απόφαση μπορεί να είναι LEFT-ARC, RIGHT-ARC ή SHIFT.

Η είσοδος του feed-forward δικτύου είναι τα embeddings λέξεων και μερών-του-Λόγου που καθορίζονται από τα features. Για τις λέξεις, τα αρχικά embeddings είναι προ-εκπαιδευμένα, ενώ για τα μέρη-του-Λόγου αρχικοποιούνται τυχαία. Και στις δύο περιπτώσεις, τα embeddings τροποποιούνται στη φάση της εκπαίδευσης. Ακολουθεί ένα κρυμμένο επίπεδο και το επίπεδο εξόδου όπου εφαρμόζεται η softmax. Σε σύγκριση με το Figure 2 του paper που παρουσιάζει την αρχιτεκτονική του δικτύου, στην συγκεκριμένη υλοποίηση δεν υπάρχουν τα arc labels (ο κώδικας εντοπίζει μόνο unlabeled arcs) ενώ η συνάρτηση ενεργοποίησης του κρυμμένου επιπέδου είναι η ReLU.

Παρακάτω δίνονται οι τιμές υπερ-παραμέτρων που χρησιμοποιούνται στο δίκτυο:

Διάσταση embeddings: 50
Πλήθος features: 36 (18 λέξεις + 18 μέρη-του-Λόγου)
Διάσταση hidden layer: 200
Συνάρτηση ενεργοποίησης hidden layer: ReLU
Πιθανότητα dropout: 0.5
Optimizer: Adam
Learning rate: 0.0005
Loss: Cross-Entropy
Batch size: 1024
Εποχές εκπαίδευσης: 10

Το δίκτυο μπορεί να εκπαιδευτεί χρησιμοποιώντας δεδομένα καθώς και τα pretrained embeddings που βρίσκονται στο φάκελο data. Υπάρχουν επίσης σύνολα δεδομένων επικύρωσης (validation set) και ελέγχου (test set) στον ίδιο φάκελο. Κατά την εκπαίδευση του δικτύου αποθηκεύεται το καλύτερο μοντέλο στο τέλος της κάθε εποχής και τελικά επιλέγεται το μοντέλο που μεγιστοποιεί την επίδοση (UAS[3]) στο validation set. Στη συνέχεια εφαρμόζεται το μοντέλο αυτό στο test set και αναφέρεται η επίδοση (UAS).

Η εκτέλεση του κώδικα είναι σχετικά γρήγορη και δεν απαιτείται η χρήση GPU.

[1] https://pytorch.org/
[2] https://aclanthology.org/D14-1082.pdf
[3] Το Unlabeled Attachment Score είναι το ποσοστό των σωστών dependencies που ανιχνεύτηκαν.

1. Εκτελέστε τον κώδικα και βρείτε το UAS που επιτυγχάνεται στο test set με βάση τις υπάρχουσες τιμές υπερ-παραμέτρων.
2. Τροποποιήστε την αρχική έκδοση του κώδικα ώστε να μην χρησιμοποιούνται pretrained embeddings για τις λέξεις. Τα embeddings των λέξεων, όπως και των μερών-του-Λόγου, πρέπει να αρχικοποιούνται τυχαία και να μαθαίνονται στη φάση της εκπαίδευσης. Πώς συγκρίνονται τα αποτελέσματα με αυτά του Ερωτήματος 1;
3. Τροποποιήστε την αρχική έκδοση του κώδικα ώστε να χρησιμοποιούνται μόνο features που αφορούν λέξεις (να μην χρησιμοποιούνται features που αφορούν μέρη-του-Λόγου). Θα πρέπει να χρησιμοποιούνται τα pre-trained embeddings των λέξεων. Πώς συγκρίνονται τα αποτελέσματα με αυτά του Ερωτήματος 1;
4. Τροποποιήστε την αρχική έκδοση του κώδικα και προσθέστε ένα ακόμη κρυμμένο επίπεδο διάστασης 100 πάνω από το υπάρχον. Μεταξύ των 2 κρυμμένων επιπέδων αλλά και μετά το δεύτερο κρυμμένο επίπεδο να εφαρμόζεται η συνάρτηση ενεργοποίησης ReLU. Πώς συγκρίνονται τα αποτελέσματα με αυτά του Ερωτήματος 1;
5. Τροποποιήστε την αρχική έκδοση του κώδικα ώστε αντί για συνάρτηση ενεργοποίησης ReLU μετά το κρυμμένο επίπεδο να εφαρμόζεται η κυβική συνάρτηση ενεργοποίησης (cube activation function) όπως περιγράφεται από τους Chen & Manning (2014). Πώς συγκρίνονται τα αποτελέσματα με αυτά του Ερωτήματος 1;

B. GRAPH-BASED DEPENDENCY PARSER

Δίνεται κώδικας που υλοποιεί ένα graph-based neural dependency parser βασισμένο στην εργασία των Kiperwasser and Goldberg (2016)[4]. Η είσοδος στο δίκτυο είναι μια ολόκληρη πρόταση στην αρχή της οποίας έχει προστεθεί ένα ειδικό token (<ROOT>). Για κάθε token της πρότασης υπάρχει embedding λέξεων και embedding μέρους-του-Λόγου (τα embeddings αυτά μαθαίνονται στην διάρκεια της εκπαίδευσης). Τα embeddings συνενώνονται και οδηγούνται σε ένα δι-κατευθηντήριο LSTM (BiLSTM) που παίζει τον ρόλο του encoder. Η έξοδος του BiLSTM οδηγείται σε δύο multi-layer perceptrons (MLPs), το καθένα με ένα κρυμμένο επίπεδο. Το πρώτο MLP δίνει ένα score για κάθε πιθανή σχέση εξάρτησης μεταξύ των tokens της πρότασης ενώ το δεύτερο MLP ταξινομεί κάθε πιθανή σχέση εξάρτησης σε κάποια από τις κατηγορίες εξάρτησης (π.χ. nsubj, dobj, κτλ). Με βάση αυτές τις προβλέψεις προκύπτει το τελικό δέντρο εξαρτήσεων.

Παρακάτω φαίνονται οι τιμές των βασικών υπερ-παραμέτρων για το συνολικό δίκτυο:

Διάσταση embeddings λέξεων: 100
Διάσταση embeddings POS: 25
Διάσταση κρυμμένου επιπέδου LSTM: 125
Πλήθος στρωμάτων LSTM: 2
Διάσταση κρυμμένου επιπέδου MLP: 100
Συνάρτηση ενεργοποίησης κρυμμένου επιπέδου MLP: tanh
Optimizer: Adam
Learning rate: 0.001
Loss: Hinge
Εποχές εκπαίδευσης: 5

Δίνονται επίσης δεδομένα εκπαίδευσης (training), επικύρωσης (validation) και ελέγχου (test) στο φάκελο data. Τα δεδομένα αυτά είναι ακριβώς ίδια με αυτά που χρησιμοποιήθηκαν στο τμήμα Α της εργασίας (transition-based dependency parsing) οπότε τα αποτελέσματα της επίδοσης είναι άμεσα συγκρίσιμα (όσον αφορά το UAS).

Ο κώδικας απαιτεί GPU για να εκτελεστεί σε λογικό χρόνο (η αρχική έκδοση απαιτεί λιγότερο από μία ώρα για μία εποχή). Συστήνεται η χρήση του **Google Colab**[5]. Αν έχετε προβλήματα χρόνου στην εκτέλεση του κώδικα, μπορείτε να μειώσετε το πλήθος των εποχών.

[4] https://transacl.org/ojs/index.php/tacl/article/viewFile/885/198 (Section 5)
[5] https://colab.research.google.com/?utm_source=scs-index

Δημιουργήστε μία αναφορά στην οποία περιγράφετε τις αλλαγές που έγιναν στην αρχική έκδοση του κώδικα και τα αποτελέσματα (UAS και LAS στο test set) που παρήχθησαν για τις παρακάτω περιπτώσεις:

1. Εκτελέστε τον κώδικα για την περίπτωση που έχουμε ένα μόνο στρώμα BiLSTM encoder. Και στα παρακάτω ερωτήματα να χρησιμοποιείτε ένα μόνο στρώμα BiLSTM. Ποια είναι η επίδοση (UAS και LAS[6] στο test set); Πώς συγκρίνεται η επίδοση UAS με αυτή του transition-based dependency parser από το τμήμα Α;
2. Αντικαταστήστε τα embeddings λέξεων που μαθαίνονται με προ-εκπαιδευμένα embeddings. Χρησιμοποιήστε τα embeddings Glove-6B-100d[7]. Πώς συγκρίνεται η επίδοση (UAS και LAS) με αυτή του ερωτήματος 1 και το UAS του τμήματος Α;
3. Αλλάξτε την συνάρτηση ενεργοποίησης του κρυμμένου επιπέδου των MLPs και αντί για tanh να χρησιμοποιείται ReLU. Πώς συγκρίνεται η επίδοση (UAS και LAS) με αυτή του ερωτήματος 1 και το UAS του τμήματος Α;
4. Αντικαταστήστε το BiLSTM encoder με το προ-εκπαιδευμένο γλωσσικό μοντέλο BERT[8]. Η είσοδος στο BERT πρέπει να είναι μία ολόκληρη πρόταση (με το ειδικό token <ROOT> στην αρχή) και η έξοδος του BERT θα οδηγείται στα MLPs. Προσοχή: το BERT εκτελεί subword tokenization για τα tokens εκτός λεξιλογίου του οπότε είναι πιθανόν μία λέξη της πρότασης να διαχωριστεί σε περισσότερα του ενός subword tokens. Σε αυτές τις περιπτώσεις να λαμβάνεται υπόψη η έξοδος του BERT μόνο για το πρώτο subword token της κάθε λέξης. Πώς συγκρίνεται η επίδοση (UAS και LAS) με αυτή του ερωτήματος 1 και το UAS του τμήματος Α;
5. Σχολιάστε τα αποτελέσματα που προέκυψαν από τα τμήματα Α και Β. Ποια πλεονεκτήματα/μειονεκτήματα έχει ο transition-based dependency parser και ποια ο graph-based dependency parser; Ποιες αλλαγές βοήθησαν στην βελτίωση της επίδοσης του κάθε μοντέλου;

Οδηγίες Παράδοσης:

Δημιουργήστε μία αναφορά στην οποία για κάθε ερώτημα να περιγράφετε τις αλλαγές που έγιναν στην αρχική έκδοση του κώδικα και τα (συγκριτικά) αποτελέσματα που ζητούνται.

[6] Το Labeled Assignment Score είναι το ποσοστό των σωστών τύπων εξαρτήσεων που ανιχνεύθηκαν.
[7] https://nlp.stanford.edu/projects/glove/
[8] https://huggingface.co/bert-base-uncased