Practice 2 – Dependency parsing

Team: Pablo Fuentes Chemes, Andrés Nó Gómez and Pablo Páramo Telle.

User Manual

This project implements a dependency parsing model that processes sentences following the arc-eager algorithm using ANNs to predict the transitions between states.
To run the model, use the *main* function from the *main.py* file, which is developed to use the functions from *utils.py* to read the necessary files, build the vectorizers and encoders, load the trained model from disk, evaluate it using the development dataset and perform inference on the test dataset for evaluate their performance.
Additionally, there are several commented lines, one to save the vectorized dataset to disk to reduce execution time during development, another to test different hyperparameter configurations to evaluate the performance of various architectures and a final one to train the model and save it to disk after training.

Discussion about the implementation

Samples Generation

The *Algorithm.py* module is essential to transform the tokenized sentences from the dataset into the (State, Transition) samples used to train and evaluate the ANNs. This module includes the *Sample()* and *Arc_eager()* classes. The latter provides all necessary methods for this transformation:

To create the initial state for a sentence, the first token (ROOT) is pushed onto the Stack, while the rest are added to the Buffer. Initially, no arcs are generated. A final state can be identified by simply checking if the Buffer is empty.

To check the validity of the four possible transitions, the preconditions that the input state must meet are verified. For the LA (Left-Arc) and RA (Right-Arc) transitions, correctness is checked by ensuring the arc generated is a golden arc, i.e., one that exists in the dependency graph. These golden arcs for a sentence can be obtained using the *Arc_eager.gold_arcs()* method. As for the REDUCE transition correctness, the required condition in the state is checked. If no other transition is valid and correct, a SHIFT is performed. The implementation of the transitions is done through the *Arc_eager.apply_transition()* method, which directly modifies the state passed as an argument.

The most notable method in this class is *Arc_eager.oracle()*, which takes a sentence (list of tokens), computes its initial state, and iterates through the states and transitions an optimal parser would go through to reach a final state. This is done by checking the validity and correctness of each transition. This process yields the (State, Transition) samples used to train the neural network.

Instances of the *Sample()* class are not directly fed as inputs and targets to the model. Instead, states are transformed into feature lists, which serve as inputs to the ANNs. This transformation is done

using the *Sample.state_to_feats()* method. The generated lists include the FORM and UPOS tag of each token. The number of tokens from the Stack and Buffer included in the vector is adjustable via two parameters (*n_features*). Padding is applied if the number of elements in these lists is less than *n_features*.

Model Implementation

The model's implementation was carried out using the *ParserMLP* class, which includes the necessary functions to train and evaluate the model and to perform inference on the test sentences.

The class is initialized with several model hyperparameters, along with two *TextVectorizers*. These are essential as they allow the model to work with two independent numerical inputs: one for the words and another for the UPOS tags. Based on these vectors, two embeddings are created, one for each input type, enabling the model to establish relationships separately between words and UPOS tags. To flatten the multiple dimensions produced by the embeddings, *Flatten* layers are applied to each embedding before concatenating them. This concatenated output is then passed to two independent networks, each with one hidden layer with *ReLu* activation function and output *Softmax* layer. One of them is designed to predict the action of the transition, and the other one chooses the dependency label that will go with it. As for the output, both transitions and dependency labels are encoded by mapping them to integer numbers.

The training process is managed by the *Parser_MLP.train()* function. The samples, extracted from the training dataset sentences using the *Arc_eager.oracle()* method, are converted to features and then vectorized before training the model.

Evaluation is performed using the *evaluate* function, which, again, vectorizes the features that come from the samples extracted from the development dataset sentences. After that, it evaluates the transitions predicted by the model, and displays the resulting metrics.

Lastly, dependency graphs prediction on the test dataset sentences is conducted. These sentences are read with the *inference=True* option, so that the HEAD and DEPREL fields of the tokens come empty. They will be filled with the predictions of our model. In the *run* function, the initial arc-eager state of the sentences is computed, then, the ANN makes its prediction for the most convenient transitions ordered with *softmax* probabilities. Starting with the most probable one, their validity is checked, and if the conditions are satisfied, the transition is applied. At this point, if the action selected is “LEFT-ARC” or “RIGHT-ARC” and the dependency label predicted is “NONE”, then the second best model prediction for dependency is applied, because these two actions must come with a no-null value to form an arc. After applying the selected transition, a new state is reached, and this process is repeated until a final state is detected. Using the information that the arcs from the final state contain, it is possible to reconstruct the dependency graph of the sentence and fill the HEAD and DEPREL fields with the predictions. This whole process is carried out vertically taking the sentences in batches for computational efficiency purposes. Finally, after post-processing the predicted parsed trees, the evaluation is done by comparing them to the golden test trees.

Results and final discussion

Different model configurations were tested. It was observed that bigger embedding dimensions, number of units in the hidden layers, batch sizes and number of features passed to the models from the stack and buffer increased the accuracies of the ANNs in predicting both the Actions and Dependency Labels that conform the Transitions. The model selected was model 6, whose configuration and results are displayed along with other ones in the following table:

Model	W_emb	U_emb	Hidden_D	Epochs	Batch	n_buff	n_stack	Action_acc	Dependency_acc
1	50	50	50	15	50	5	5	**0'8373**	**0'8497**
2	100	100	100	15	50	8	8	**0'8425**	**0'8524**
3	200	200	200	15	100	12	12	**0'8469**	**0'8632**
4	300	300	300	20	100	16	16	**0'8511**	**0'8622**
5	400	400	400	20	200	20	20	**0'8507**	**0'8682**
6	**500**	**500**	**400**	**20**	**200**	**20**	**20**	**0'854**	**0'872**

The next table shows the evaluation metrics for the dependency parsing predictions using the model 6, calculated based on the comparison with the golden dependency graphs from the test sentences.

Metric	Precision	Recall	F1 Score	AligndAcc
Tokens	100.00	100.00	100.00	
Sentences	100.00	100.00	100.00	
Words	100.00	100.00	100.00	
UPOS	100.00	100.00	100.00	100.00
XPOS	100.00	100.00	100.00	100.00
UFeats	100.00	100.00	100.00	100.00
AllTags	100.00	100.00	100.00	100.00
Lemmas	100.00	100.00	100.00	100.00
UAS	74.50	74.50	74.50	74.50
LAS	67.08	67.08	67.08	67.08
CLAS	56.44	53.17	54.76	53.17
MLAS	54.35	51.21	52.73	51.21
BLEX	56.44	53.17	54.76	53.17

It is noteworthy that the accuracy achieved by the model in predicting the appropriate transitions for the states in the arc-eager algorithm is significantly better than the accuracy obtained in inferring the dependency graphs for the sentences in the test dataset. The reason behind this discrepancy is that, despite using the same model, when processing an entire sentence, an error in selecting the transition for a given state affects the state it produces, thus impacting the validity and correctness of the transitions recommended in the following steps.

Additionally, it is observed that the UAS is higher than the LAS. This is because LAS not only considers the origin and target of the arc—corresponding to the "action" prediction—but also accounts for the selected label, which corresponds to the "dependency" prediction.