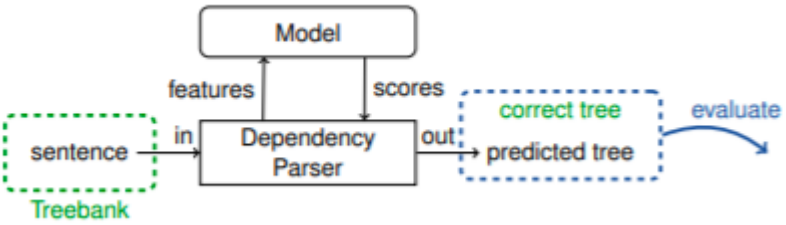
Statistical Dependency Parser Report[1]

Introduction:

The aim of this project was to learn to implement dependency parsers from scratch. There are two main types of parsers namely transition and graph parser. Following parsers are implemented in Python.

1) Graph-based Eisner (Working)
2) Transition-based Arc-standard (Not working due to some bugs).

The main components of a parser are feature extraction, decoder and a model such as perceptron. The dependency parser block in the figure contains a feature extractor and a decoder that enables to predict the dependency tree relations based on the scores obtained using the model.



Eisner Algorithm:

Eisner makes a projective spanning tree over a sentence and the implementation is done using the concepts of dynamic programming. Given a score matrix, created using the trained weights of the eisner perceptron, the eisner algorithm generates the arcs(tree) using back-tracking.

The perceptron model requires some features and the gold tree to train the model. The weights of the model are updated when the predicted tree does not match with the gold tree. Following unigram and bigram templates are used to make feature vectors. Trigram features are not added as it makes the vocabulary size large which make it difficult to train the model.

Unigram	"hform", "hpos", "hform+hpos", "dform", "dpos", "dform+dpos"
Bigram	“hform+hpos+dform+dpos”", "hpos+dform+dpos", "hform+dform+dpos", "hform+hpos+dform", "hform+hpos+dpos", "hform+dform", "hpos+dpos"

The eisner algorithm takes approx. 15 minutes per epoch. The metric stops improving considerably after 3-4 epochs.

Following are the bugs that were encountered during implementation:

[1] Figure from the SDP lecture slides.

- Features should be extracted from so-called potential trees for the Eisner algorithm. However, the mistake I made was to extract features from the gold trees.
- Feature vector should have the same fixed length. However, I was ignoring the features that were not present which made the vector length different.

Arc-standard:

The arc-standard uses a set of states/configurations and transitions to build a tree. A state is a triplet consisting of stack, buffer and created arcs. We start with the initial state and take some transitions to reach the terminal state. The created arcs of the terminal state is our predicted tree. There are three transitions namely left-arc, right-arc and shift that can take place based on some conditions. The left-arc and right-arc create an arc that is added to the set of created arcs of a state.

Following are the unigram and bigram templates used for feature extraction for each state.

Unigram	"S[0]-form", "S[0]-pos","S[0]-lemma" ,"B[0]-form","B[0]-lemma", "B[0]-pos","B[1]-pos", "S[1]-pos","ld(S[0])","rd(S[0])","ld(B[0])","rd(B[0])"
Bigram	"S[0]-form+S[0]-pos+B[0]-form+B[0]-pos", "S[0]-form+S[0]-pos+B[0]-form", "S[0]-form+B[0]-form+B[0]-pos", "S[0]-form+S[0]-pos+B[0]-pos"

The state features along with the correct transition labels are used to train the perceptron algorithm. The perceptron is used to assign scores of the three transitions for a particular state. Correct transition sequence is obtained using the Oracle parser, if the predicted transition label does not match with the gold transition label obtained form Oracle parser then the weights are updated. The trained model can then help to predict the correct set of transitions to go from the initial state to the terminal state.

Evaluation:

Unlabeled Attachment Score(UAS) is used as an evaluation metric. The table below reports the result obtained using the Eisner Algorithm. There are no results for Arc-standard due to some bugs in implementation.

	Development	**Test**
German	86	80
English	82	82

Conclusion:
The project provides implementation of graph-based Eisner and transition-based Arc-standard parser. The Eisner parser achieves decent UAS using unigram and bigram features. In the future, I plan to fix the bugs in the arc-standard parser so that the performance of models could be compared.