Deliverable 6.1 (6 points):

Consider a sentence of 10 words:

- In Project 1, we learned to build a text classifier. If the classifier is binary, there are 2 possible outputs for this sentence. In Project 2, we learned to build a sequence tagger. If each word has four possible labels, there are 4_{10}=1048576 possible outputs for this sentence. In this project, we learned to build a parser for **projective** dependency trees. **How many possible projective dependency trees are there for our sentence of 10 words? Why? Is this output space larger than the one above?**

  So, for one word, there is just one possible tree. For n=2 (2 words), there will be 3 trees possible. For n=3, there will be 7 words possible. Since we multiply by 2 every time we go up the tree until we reach the root of the tree, so accordingly there will be (2**10) - 1 possible dependency trees.

  N = 1 - > 1

  N = 2 - > 3

  N = 3 - > 7

  So, by recursion, we can see that for every N, there will be (2**N - 1) possible trees. We create an edge when we perform arc_left or arc_right action(2 possibilities) and no edge is created in the shift action. Therefore, for 10 words ie N= 10, there will (2**10) - 1 = 1024-1 -> 1023 possible trees. This is way less than the number mentioned above(which is equivalent to 2**20).

Deliverable 6.2 (6 points):

Briefly describe your bakeoff design.

English Bakeoff

- I made use of BiLstm word embedding and also created LSTMActionChooser with dropout as 0.4 which would use the current state of the parser and run them through an LSTM, returning log probabilities over actions. I trained the Transition parser(designed in Deliverable 3.1) using LSTMActionChooser and LSTMCombiner. I used Adam optimizer with the learning rate as 0.05 and also used a weight decay. I used ReduceLROnPlateau scheduler with the weight decay.
- I performed extensive parameter tuning to achieve an accuracy of 78.3% on the dev set for English language.

Norwegian Bakeoff

- Made use of BiLstm word embedding for training the network and created LSTMActionChooser with dropout as 0.2. I trained the Transition parser(designed in Deliverable 3.1) using LSTMActionChooser and LSTMCombiner designed during the course of the assignment. Used Adam optimizer with the learning rate as 0.001.
- Performed extensive parameter tuning to achieve an accuracy of 73.39% on the dev set for the Norwegian language.

Deliverable 6.3 (10 points):

You will select a research paper at ACL, EMNLP or NAACL that uses dependency trees for some downstream task. Summarize the paper, answering the following questions:

Paper Title: Punctuation Prediction with Transition-based Parsing

Authors : Dongdong Zhang, Shuangzhi Wu, Nan Yang, Mu Li

Venue : Association for Computational Linguistics, 2013

1. **What is the task that is being solved?**

- Standard automatic speech recognizers output unstructured streams of words. They neither perform a proper segmentation of the output into sentences nor predict punctuation symbols. The unavailability of punctuations acts as a barrier to many subsequent text processing tasks like summarization.
- The authors solve this problem by designing a system for predicting punctuations. The authors propose a model that jointly predicts the sequence of punctuation and performs parsing. This method jointly performs parsing and punctuation prediction by integrating a rich set of syntactic features when processing words from left to right.
- These syntactic features significantly improve the performance of punctuation prediction. Authors propose models with linear complexity that are capable of handling streams of words with any length. All the previous punctuation prediction methods were not able to incrementally process inputs and were not efficient for very long inputs. The proposed punctuation prediction framework solves these problems.

2. **Briefly (one sentence) explain the metric for success on this task.**

   The evaluation metrics used by authors are precision (prec.), recall (rec.), and F1-measure (F1).

3. **Why are dependency features expected to help with this task?**

Most previous methods for punctuation predictions involved word-level models that integrated local features like lexicons, prosodic cues. So, these models had a narrow view of the input. Punctuation prediction requires a more global context, especially for long-range dependencies. For instance, in English question sentences, the ending question mark is long-range dependent on the initial phrases. So, dependency features can be very useful in providing a more global context for punctuation prediction. The authors mention previous works that used syntactic features for this problem. However, those works treated punctuation prediction and syntactic prediction separately. Additionally, their methods were not able to incrementally process inputs and were not efficient for very long inputs.

Authors propose jointly performing punctuation prediction and transition-based dependency parsing over transcribed speech text. Their models have linear complexity and are capable of handling streams of words with any length. In addition, the computation of models use a rich set of syntactic features, which can improve the complicated punctuation predictions from a global view, especially for the long-range dependencies.

4. How are dependency features incorporated into the solution?

The punctuation prediction problem is formulated as follows:

Input = a stream of words from an automatic transcription of speech text, denoted by $w_1^n := w1, w2, \dots, wn$.

Output = sequence of punctuation symbols $S^n_1 := s1, s2, \dots, sn$ where si is attached to wi to form a sentence

The prediction of the best sequence of punctuations can be formulated as follows:

$S^* = \text{argmax } P(S_1^n | w^n_1)$ (1)

Authors convert the problem in (1) by introducing the transition-based parsing tree T to guide the punctuation prediction as shown below:

$S^* = \text{argmax} \sum P(T|w^n_1) \times P(S_1^n|T, w^n_1)$

Instead of enumerating all possible transition trees, authors convert the above problem into jointly optimizing the punctuation prediction model and the transition-based parsing model with the form:

$(S^*, T^*) = \text{argmax}\ P(T \mid w^n_1) \times P(S_1^n \mid T, w^n_1)$

Let T_i^1 be the constructed partial tree when w^1_i is consumed from the queue.

$$(S^*, T^*) = \text{argmax} \prod P(T_i^1 \mid T_{i-1}^1, w^1_i) \times P(s_i \mid T_i^1, w^1_i)$$

Authors proposed the following 2 ways of solving the above joint optimization problem:

1) **Cascaded punctuation prediction model (CPP) -** This method conduct transition actions of parsing followed by punctuation predictions in a cascaded way. In this method, the parsing problem is solved first and then punctuation prediction. As the words in the stream are consumed, each computation of transition actions is followed by a computation of punctuation prediction. Thus, the two sub-models are computed in a cascaded way, until the optimal parsing tree and optimal punctuation symbols are generated.
2) **Unified punctuation prediction model (UPP) -** Here, the computation of the parsing tree and punctuation prediction is unified into one model where the sequence of transition action outputs uniquely determines the punctuations attached to the words.

5. Does the paper evaluate whether dependency features improve performance on the downstream task? If so, what is their impact? If not, why not?

Yes, the paper evaluates the performance of the proposed method. Authors test the performance of their method on both the correctly recognized texts and automatically recognized texts. Authors show that their framework for punctuation prediction performed better than a baseline method based on the CRF model(which incorporated the features of bag of words and POS tags) for punctuation prediction.

Test data information - Correctly recognized texts consisted of 10% of IWSLT09 training set, which consists of 19,972 sentences from BTEC (Basic Travel Expression Corpus) and 10,061 sentences from CT (Challenge Task). The automatically recognized text was taken from TDT4 data.

The tables below show a comparison between the proposed methods (CPP and UPP) and CRF in terms of accuracy, precision, and F1 scores.

	Measure	Middle-Paused	Full-stop	Mixed
Baseline (CRF)	*prec.*	33.2%	81.5%	78.8%
	rec.	25.9%	83.8%	80.7%
	F_1	29.1%	82.6%	79.8%
CPP	*prec.*	51%	89%	89.6%
	rec.	50.3%	93.1%	92.7%
	F_1	50.6%	91%	91.1%
UPP	*prec.*	52.6%	93.2%	92%
	rec.	59.7%	91.3%	92.3%
	F_1	55.9%	92.2%	92.2%

Table 4. Punctuation prediction performance on correctly recognized text

	Measure	Full-stop
Baseline (CRF)	*prec.*	37.7%
	rec.	60.7%
	F_1	46.5%
CPP	*prec.*	63%
	rec.	58.6%
	F_1	60.2%
UPP	*prec.*	73.9%
	rec.	51.6%
	F_1	60.7%

Table 5. Punctuation prediction performance on automatically recognized text

The performance of the proposed method is much higher than that of the baseline. The UPP yields slightly better performance than CPP on full-stop and mixed punctuation prediction, and much better performance on middle-paused punctuation prediction. This could be because the interaction of parsing and punctuation prediction is closer together in UPP than in CPP. The performance of middle-paused punctuation prediction is fairly low between all methods, which shows predicting middle-paused punctuations is a difficult task.