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?**

  For n nodes, there are a total of n**(n-1) possible trees. Out of these trees, we have to consider only projective dependency trees ie the trees where there is no crossing between the edges. We know that for n = 1, there is one possible tree. For n=2, there should be 2 possible trees. Eg, if A and B are 2 nodes(words), then it is possible to have A -> B and A <- A. For n=3, there will be a total of 9 possible trees out of which only 7 are projective.

  So, now, if I think of n=4, we are adding another node to the previous 3 nodes, so every 7 possible trees for n=3 will have 2 edges possible(one outwards[edge going from left to right] and one inwards[edge going from right to left]) for new added node, so that will make it 14 new trees. Also, the 4th node can be added before and after 3 nodes, so that will give 2 permutations making it 28. If we consider the middle connection where both the edges go outward[nodes where edges go outward in both directions <- ->], that will add 2 more trees making it 30 trees for n = 4.

  Similarly, for n=5, we will consider the n=4 tree and add one more node to it. This way, we will have 2 possible combinations for each of these 30 trees, one combination in which edges go from left to right and another where edges go from right to left. There will also be 2 ways of adding the node, to the front, or to the end, we will get 120 trees. Now, if we consider, the combinations for middle connections, we will get 12 more trees. This makes it to be 136 trees.

  Extending this logic to different n, we get the following sequence:

  N = 1 - > 1

  N = 2 - > 2

  N = 3 - > 7

  N = 4 - > 30

N = 5 - > 132

N = 6 - > 560

N = 7 -> 2300

N = 8 -> 9296

N = 9 -> 37440

N = 10 -> 150272 trees

So, for n=10 words, we will get **150272 projective trees.** The output space is clearly less than 4**10.

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. 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.
- I performed extensive parameter tuning to achieve an accuracy of 78.3% on the dev set for English language. I trained the parser for 10 epochs.

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. I trained the parser for 8 epochs.

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 do not perform segmentation of the output into sentences and do not predict punctuation symbols. The unavailability of punctuations acts as a barrier to many subsequent text processing tasks like summarization as, without punctuations, the meaning of the sentence can be completely different.
- 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. They solve a joint optimization problem for the task. This method jointly performs parsing and punctuation prediction by integrating a rich set of syntactic features when processing words from left to right.
- The use of syntactic features significantly improves the performance of punctuation prediction.
- Authors propose 2 models with linear complexity for solving the joint optimization problem: **Unified punctuation prediction model (UPP)** and **Cascaded punctuation prediction model (CPP).**
- Both these models are capable of handling text of large lengths. 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, an English question sentence ending with a question mark may have a long-range dependence on the initial part of the phrase. 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. They did not solve the problem jointly as proposed by the authors. 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 lower complexity and are capable of dealing with input streams of any length. In addition, the uses of syntactic features greatly improve the performance of punctuation prediction in comparison to the baseline method.

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:

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

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

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

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

1) **Cascaded punctuation prediction model (CPP) -** This method performs transition actions of parsing followed by punctuation predictions in a cascaded

way. So, the parsing problem is solved first, followed by punctuation prediction. As the words in the stream are consumed, each computation of transition actions is followed by a computation of punctuation prediction. This process continues until the optimal parsing tree and optimal punctuation symbols are generated.

2) **Unified punctuation prediction model (UPP) -** In this model, the computation of the parsing tree and punctuation prediction are combined into one model. This one model which outputs a sequence of transition action which 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(both models UPP and CPP) 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

Impact of the proposed algorithm:

- performance of the proposed method is much higher than that of the baseline.
- UPP yields slightly better performance than CPP on full-stop and mixed punctuation prediction, and much better performance on middle-paused punctuation prediction. The better performance of UPP is due to a closer connection between parsing and punctuation prediction due to cascading.
- All methods perform poorly on middle-punctuation prediction, however, the performance of UPP & CPP is still much better than baseline. The low accuracy values indicate that middle-punctuation is a difficult problem to solve.

Overall, the authors proofed that solving punctuation prediction problem jointly along with parsing leads to better results.