Evaluating a Beam Search Tagger and Parser on Different Dataset Sizes and using Fine-Tuning

Presented by NLP Group G14

Afshan Hashemi
Linus Roos
William Johansson
Yuting Huang

Intro

What is our goal?

- To find out how beam search affect the performance of transition-based tagger and parser
- To optimize the performance with different techniques

What have we done?

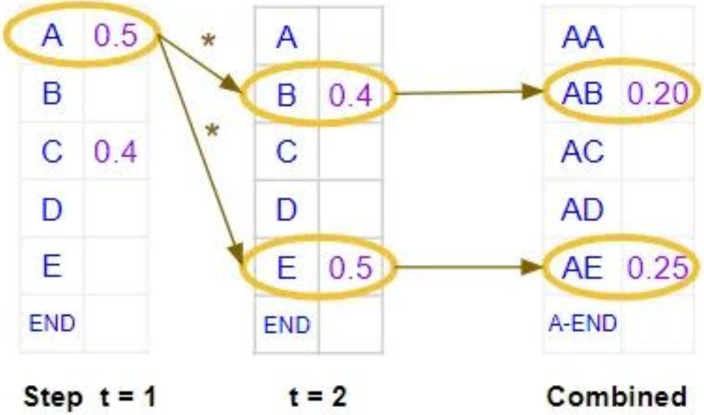
- Implemented a Tagger and Parser with Beam Search v.s.Greedy Search
- Trained and evaluated on 4 different languages: English/Swedish/Persian/Chinese
- Evaluated the system with different dataset sizes
- Optimized the performance with pre-training and fine-tuning

Beam Search

What is beam search and why to use it?

- With Greedy Search, we took just the single best word at each position. In contrast, Beam Search expands this and takes the best ’N’ words.
- It is casting the “light beam of its search” a little more broadly than Greedy Search.

Beam Search



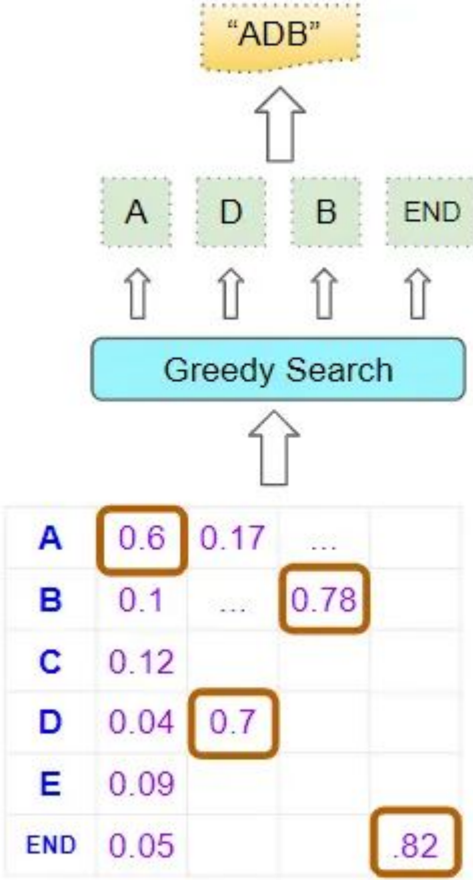
Prob (AB | input) = Prob (A | input) * Prob (B | A, input)

Prob (AB) = Prob (A) * Prob (B | A)

= 0.5 * 0.4

= 0.20

Greedy Search



Beam Search

- What is beam search and why to use it? ✔️

- How to implement beam search in the tree bank model?

Tagger

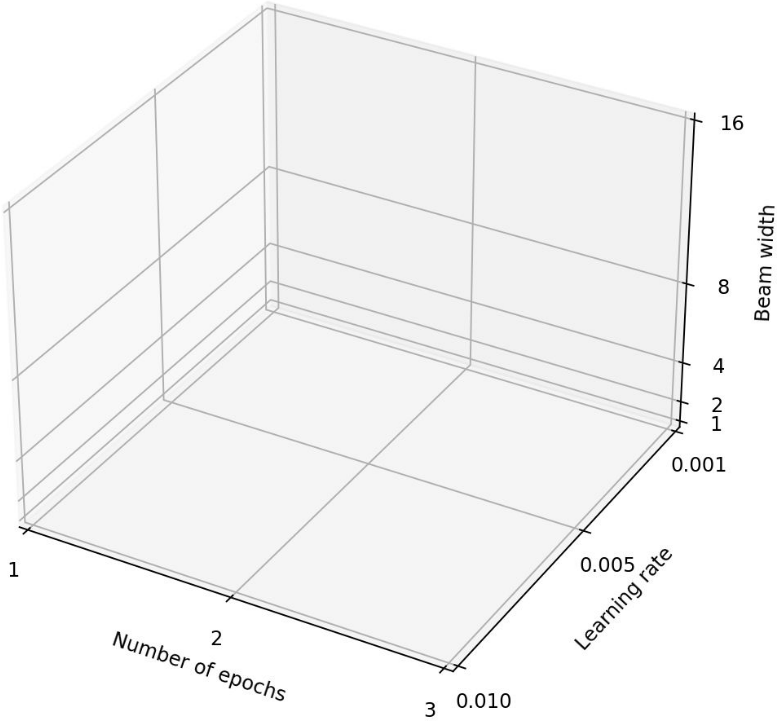
```python
def predict(self, words):
    words = [self.w2i.get(w, UNK_IDX) for w in words]
    beam = [(0, [])]  # (score, sequence)

    for i in range(len(words)):
        new_beam = []
        for score, sequence in beam:
            output = self.model.forward(self.featurize(words, i, sequence))
            output = F.softmax(output, dim=0)  # Apply softmax to the output scores
            for tag in range(len(self.i2t)):
                new_sequence = sequence + [tag]
                new_score = score + output[tag].item()  # Use the softmax output here
                new_beam.append((new_score, new_sequence))
        beam = sorted(new_beam, key=lambda x: x[0], reverse=True)[:self.beam_width]

    return [self.i2t[i] for i in max(beam, key=lambda x: x[0])[1]]
```

Same for Parser

Hyperparameter optimization

- Hyperparmeters:
    - Epochs
    - Learning rate
    - Beam width

- Grid search in 3 dimensions



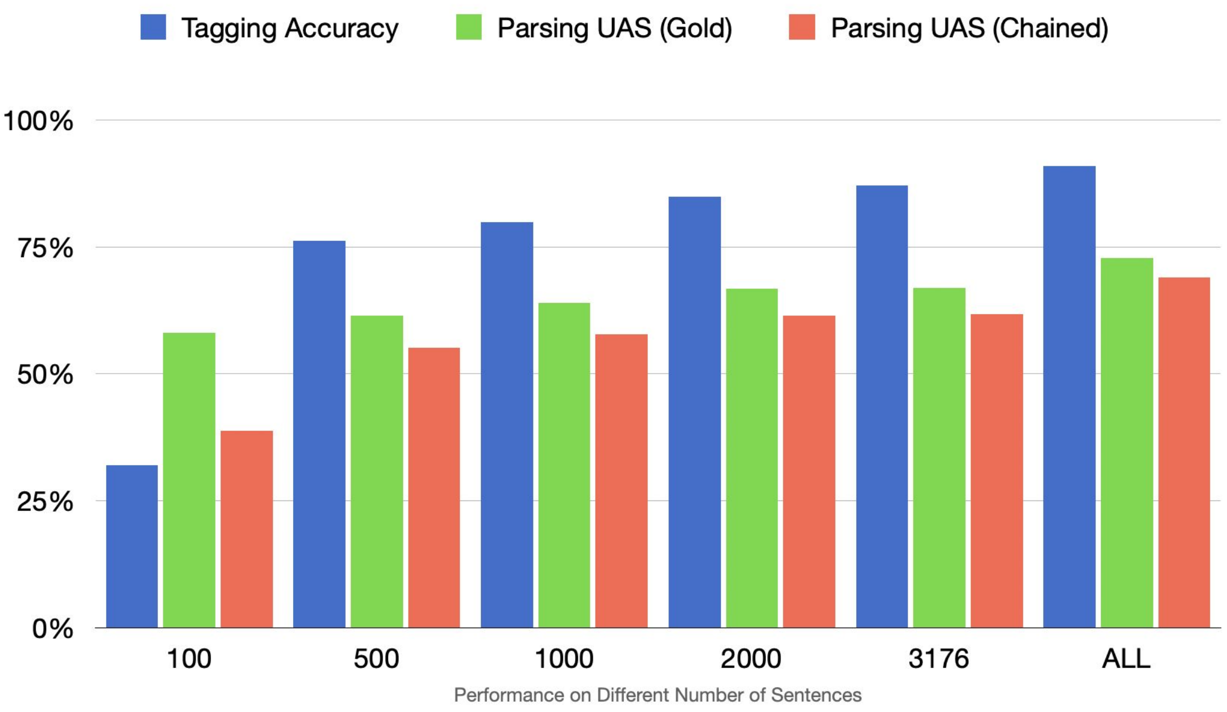
Hyperparameter optimization - Results

- 2 Epochs
- 0.005 Learning rate

BeamWidth	Tagging accuracy	Parsing uas (gold)
1	90.83%	**70.41%**
2	90.90%	70.26%
4	90.83%	70.28%
8	**90.91%**	69.85%
16	90.87%	69.74%

	Tagging accuracy	Parsing uas (gold)
Baseline	89.87%	70.34%

Testing on Dataset Sizes



Purpose:

- If the dataset size would affect the accuracy
- If we can use a smaller dataset for further training

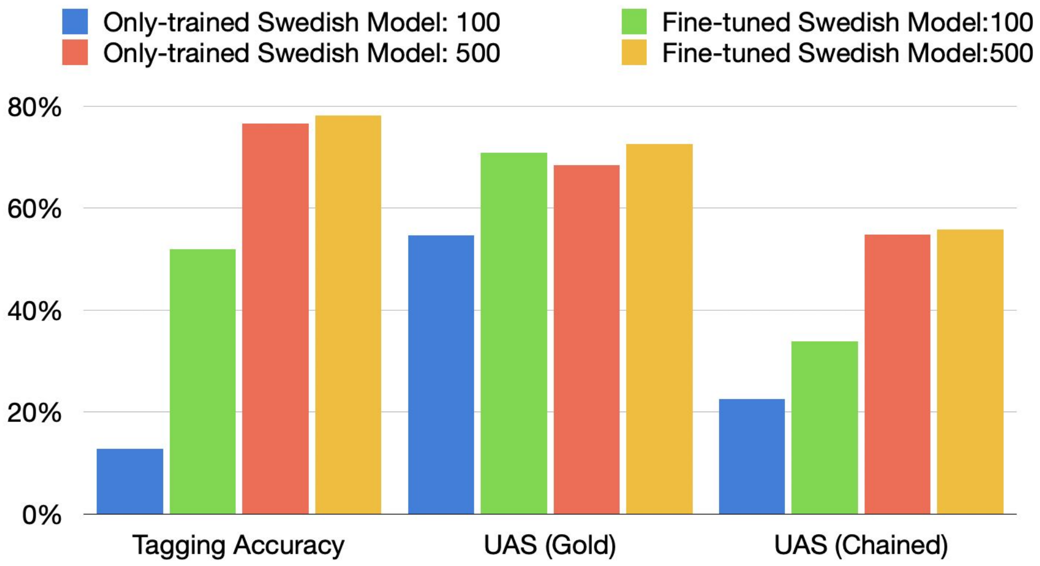
Result:

- A dataset with at least 2000 sentences yields approximately 95% accuracy compared to the full dataset.

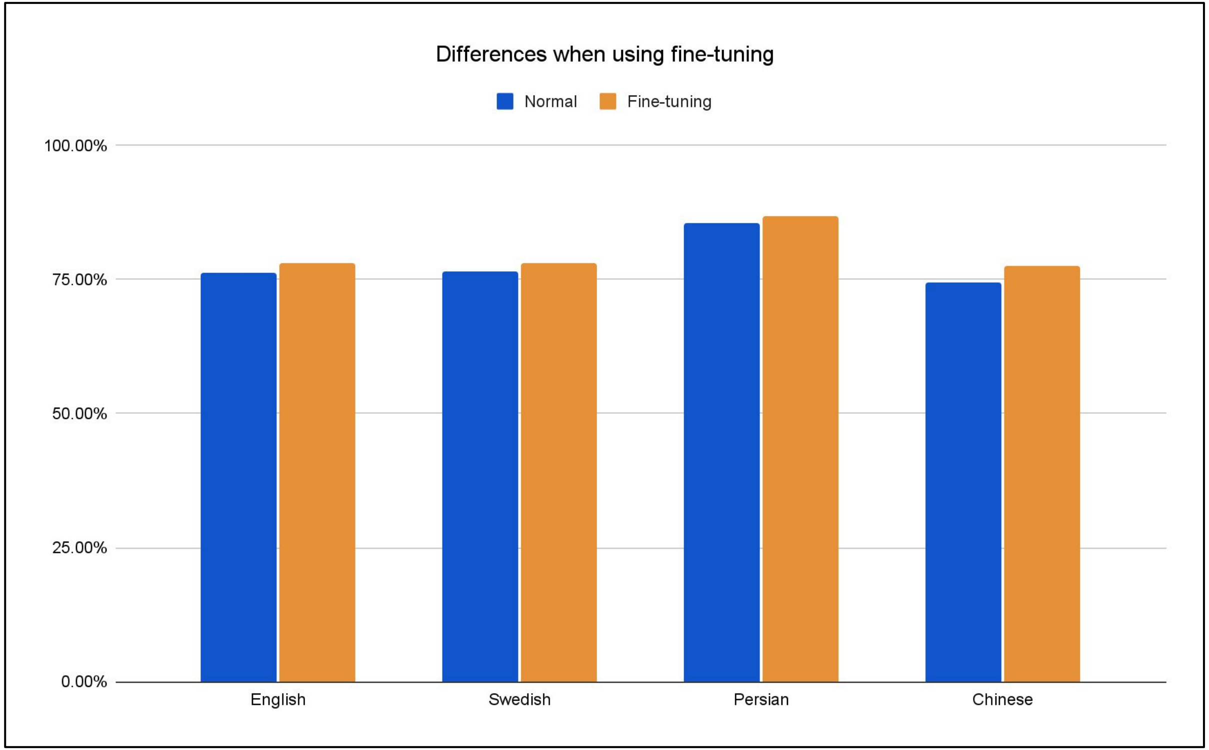
Pre-training and Fine-tuning

Fine-tune both the tagger and the parser:

- Pre-trained on the Training data (e.g. English treebank en-ewt)
- Fine-tuned on 100 and 500 sentences from a target language(e.g. Swedish)
- Evaluated the model on Dev-Data(e.g Swedish)

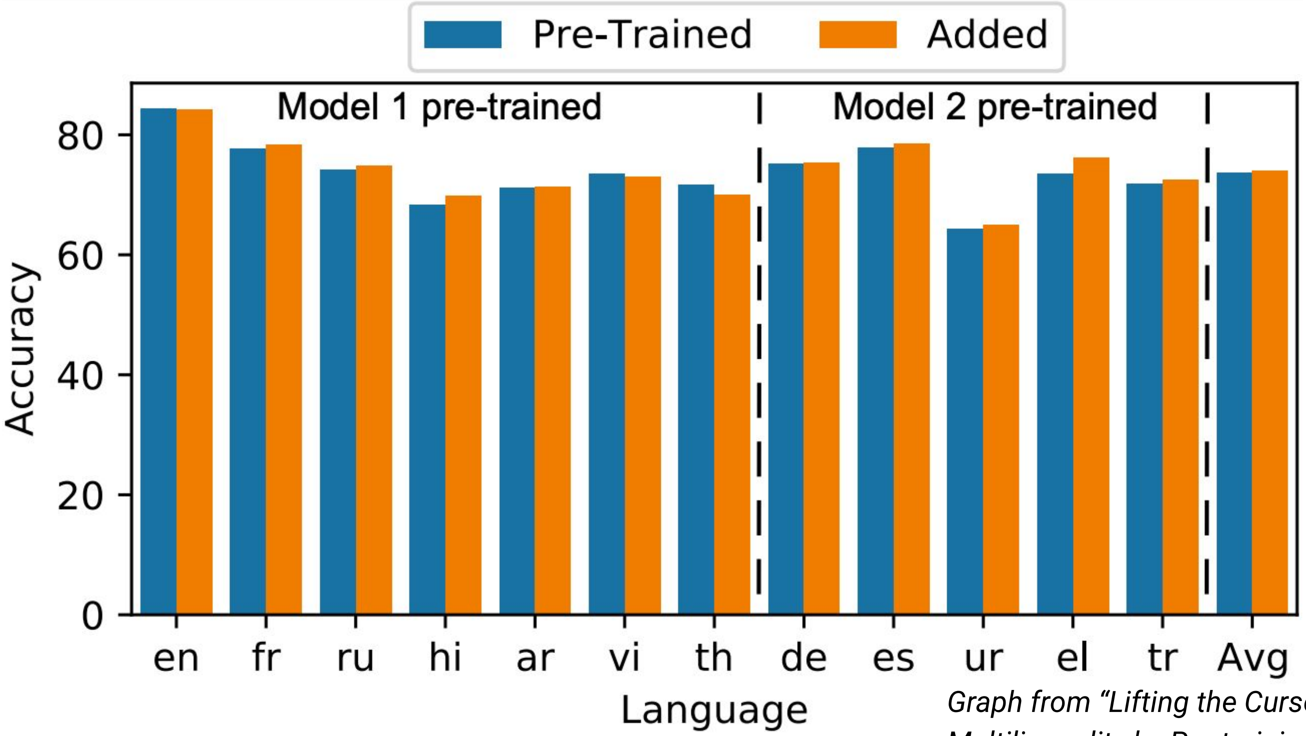


Comparison



Differences when using fine-tuning
Normal
Fine-tuning
100.00%
75.00%
50.00%
25.00%
0.00%
English
Swedish
Persian
Chinese

Comparison to research



Graph from "Lifting the Curse of Multilinguality by Pre-training Modular Transformers" by Pfeiffer et al

Conclusion

- Better than baseline
- Beam width – local and global approach
- Comparable to existing research

Global beam search proposed in "Globally Normalized Transition-Based Neural Networks" by Andor et al