Project Report of the Course "Data Analysis Project", Summer Semester of 2024 at the University of Vienna

Jona Marie Hassenbach

1 Introduction

1.1 Motivation

Recent advancements in universal dependency parsing, specifically Udapter [1], have yielded valuable insights into the parsing of low resource languages. Without being trained on them, this universal dependency parser is able to successfully parse various languages that themselves lack the data to successfully train a dependency parser. Hence universal dependency parsers like Udapter provide a viable option for the parsing of low resource languages.

1.2 Project Proposition

Udapter, however, was meant to be universal and the results on non-training languages vary considerably. To analyze its performance when utilized for a specific target language, I trained the Udapter model on three languages that share different typological similarities with a chosen target language. This is done due to the unique architecture of Udapter, specifically the so-called *Contextual Parameter Generator*, which enables Udapter to learn parameters depending on the typologies of the in-training languages. Hence, with the major typological characteristics of the target language shared by one or several of the training languages, the resulting model should – in theory – be able to successfully parse the target language.

Going one step further, I also wanted to analyze the impact of the training set composition. For the original Udapter model the percentages in which the individual languages occurred during training were given by the size of their treebanks. To see whether different compositions of language occurrences, i.e. their *weight*, have an effect on the resulting parser, I trained Udapter on four different datasets instead of one (see below) and compared the performance of the four resulting models.

2 Theoretical Setup

2.1 Choosing the Languages

Out of personal interest, I chose Mandarin Chinese as my target language, even though it cannot be considered a low resource language. To match its main typologies, and with the help of Jeremy, I first considered Japanese, Russian, and Vietnamese as my three training languages. Japanese shares approximately 60 percent of its vocabulary

with Mandarin and exhibits some syntactic similarities too. Vietnamese, in addition to some shared vocabulary, is also tonal language like Mandarin.

However, further investigation [2] indicated that English might be a better fit as third training language compared to Russian, since English shares more language typologies with Mandarin: Both languages use SVO word order, there is little to no inflection in verbs and nouns, and no variation in adjectives for number, gender, or case. Russian, on the other hand, does not have a strict word order, instead, it relies on inflection to determine meaning. Hence Jeremy and I decided that I should work with English, Japanese, and Vietnamese.

2.2 Choosing the Treebanks

In consultation with Prof. Roth, Jeremy and Anastasiia, it was decided to use the treebanks of the original Udapter model to ensure maximum comparability. All three treebanks can be found on the Universal Dependencies website. The English one is called *UD English EWT*, the Japanese one *UD Japanese GSD* , and the Vietnamese one *UD Vietnamese VTB*.

2.3 Choosing the Multilingual pre-trained LM

Udapter uses contextualized word representations, which are obtained via mBERT. It is worth mentioning a recent study [3], which has found XLM-R to yield better results. After consulting with Prof. Roth however, it was decided to not introduce any additional new elements in order to ensure comparability.

3 Practical Steps

3.1 Setup of the Code

It should be mentioned that the Udapter code is quite old. It was written in 2020, it works with Python 3.6.8, and requires a number of packages that by now are outdated. Therefore I encountered a range of issues. These issues fall into two categories: On the one hand getting the code to work at all, on the other hand getting the model to run on GPU – as it was originally intended. While I succeeded on the first task, the same cannot be said for the second one. Therefore, all models mentioned below had to be trained on my own PC using its CPU, which took several hours or even days.

After pulling the Udapter code from the corresponding git repository, I encountered the first issue: the model requires `torch==1.1.0`, a version that is no longer supported by torch and cannot be installed the normal way using pip. With the help of the internet this problem was soon resolved: After creating the conda environment, the command `conda install pytorch-cpu==1.1.0 torchvision-cpu==0.3.0 cpu only -c pytorch` has to be run. Afterwards the other packages can be installed the normal way. Due to an error message I encountered, I followed the internet's advice and also installed the package `overrides==3.1.0`. The different steps of the setup can also be found in my python script.

In the next step different minor bugs had to be fixed: with the help of Anastasiia we, Marlene and I, found that in several files the `open` statement needed an `encoding="utf-8"` to be added. In addition, the Json files in the *udapter/config* folder had to be corrected, possibly because their format was outdated. Hence comments had to be removed, multiplication terms had to be replaced by their results, and variable values had to be copied and pasted to the correct location. Eventually, later after running the first model and trying to evaluate it, it turned out that in line 28 of the *udapter/predict.py* file the default value needed to be changed from `None` to `-1`. All resulting code and Json files can be found on my git.

Now, with `cuda_device` set to `-1` in the *udapter/config/udify_base.json* file, I was able to start training a test model, which was running smoothly, but very slowly on my PC. Since the program predicted the training to last for at least 13 days, I consulted with Prof. Roth and we decided to stop the training, and for Marlene and I to start setting up the program on a device with a GPU, namely the *Galadriel* server of the University of Vienna. I'm assuming that Marlene is describing this part in detail, therefore, and since I did not end up using *Galadriel*, I will not depict the setting up process myself.

Though the setting up process was successful, the same cannot be said for the training on GPU. Our modified Udapter code was not able to connect with *Galadriel's* GPUs. Even with the help of Anastasiia and Johannes we were not able to resolve this issue. The problem is that `torch==1.1.0` needs CUDA 10.0 to run, but CUDA 10.0 is not compatible with the operating system of *Galadriel*, Ubuntu 22.4. To fix this issue, one would either have to update the whole Udapter code or build a docker container that runs on Ubuntu 18.4, the version required for CUDA 10.0 to work. Both of these solutions go beyond the scope of this project. Therefore we modified our approach (see below) and ran our models on CPU.

3.2 Setup of the Datasets

While we were still dealing with the abovementioned issues, I prepared my datasets. The original Udapter simply concatenated its different treebanks, hence the training data consisted of all sentences available in its chosen 13 training datasets. Therefore the *weight* of each language, i.e. the percentage of sentences the model saw during training that were in the specific language, was given by the number of sentences in its treebank divided by the total number of sentences. For the original Udapter the *weights* of the 13 languages were rather different.

I, on the other hand, wanted to see if the *weights* of the languages effect the performance of the resulting parser. Thus, I planned on comparing multiple models trained on the same training languages, but different *weights*. Hence, I wrote a program that randomly selects a given number of sentences from a treebank (see my python script). This way I could concatenate the resulting sub-treebanks and was able to control how many sentences per language are in my training data.

However, this alone is not sufficient to control the *weights* of the different languages. To up a languages *weight* one cannot simply select more sentences from its treebank, because this introduces a new example that might effect the performance of the parser

– not because it sees more sentences in a given language, but due to the sentences themselves. To ensure the comparability of the different models one has to up sample, i.e. include a language's sub-treebank several times in the concatenation process. This way the resulting training data now includes the corresponding sentences several times and the *weights* of the languages are adjusted accordingly.

This introduces a new issue: By up sampling my training data, the total number of sentences the model sees, i.e. its number of *updates*, also changes. The number of *updates* is given by the number of sentences in the training data multiplied with the number of epochs of the specific model. Hence, to ensure the comparability of the different models, the number of epochs had to be adjusted so that each model saw the same number of updates. Table 1. lists the properties of every model I ran.

Table 1. Properties of all models I ran.

Name	Weights (en, ja, vi)	Sentence Number	Epochs	Updates
en33_ja33_vi33	⅓ each	300	20	6,000
en20_ja20_vi60	20%, 20%, 60%	500	12	6,000
en20_ja60_vi20	20%, 60%, 20%	500	12	6,000
en60_ja20_vi20	60%, 20%, 20%	500	12	6,000
en20_ja40_vi40	20%, 40%, 40%	500	12	6,000
en25_ja25_vi50	25%, 25%, 50%	400	15	6,000
en25_ja50_vi25	25%, 50%, 25%	400	15	6,000
en40_ja20_vi40	40%, 20%, 40%	500	12	6,000
en40_ja40_vi20	40%, 40%, 20%	500	12	6,000
en50_ja25_vi25	50%, 25%, 25%	400	15	6,000
en33_ja33_vi33_big	⅓ each	1,200	5	6,000
en20_ja20_vi60_big	20%, 20%, 60%	2,000	3	6,000
en20_ja60_vi20_big	20%, 60%, 20%	2,000	3	6,000
en60_ja20_vi20_big	60%, 20%, 20%	2,000	3	6,000
en33_ja33_vi33_ultimate	⅓ each	36,000	10	360,000
en50_ja25_vi25_ultimate	50%, 25%, 25%	24,000	15	360,000
en33_ja50_vi17_ultimate	⅓, ½, ⅙	36,000	10	360,000
en33_ja17_vi50_ultimate	⅓, ⅙, ½	36,000	10	360,000

In case of the last four models, the *ultimate* ones, I had to up sample a lot, since the original Vietnamese treebank only had 1,200 sentences in their training data, while the English one had 12,543 and the Japanese one 7,133. To include as many sentences as possible while keeping the proportions, I let my program pick a subset of 12,000 sentences for English and 6,000 for Japanese, and included all 1,200 Vietnamese sentences. To reach the proportions listed above, I therefore had to include the Vietnamese training set up to 15.

I also used my code to generate development and test sets. These were the same for the first 10 models and *big* models, and they both consist of 300 sentences, 100 per language. For the *ultimate* models, I generated a bigger development and test set, consisting of 1,800 sentences each. Naturally, the development and test sets were not included in the up sampling process.

3.3 Running the models

While we (Marlene) were running a bigger model on *Galadriel*'s CPUs – to see if the resulting model actually works and while trying to get the GPU to work –, I decided to run several small models on my own PC. I first ran the 10 small models listed in Table 1., but the results were rather bad: the highest UAS score was 8.1% and the highest LAS score 0.35%. Thus, I ran the 4 *big* models, hoping that more examples (sentences) would help the models to yield better results. Though, to ensure comparability with the small models, I had to scale down the number of epochs. This, however, did not result in considerable performance changes: The UAS score improved slightly (up to 8.8%), but the LAS score stayed the same. To get meaningful results, I had to change my approach.

In the meantime two things had happened: First of all, the model on *Galadriel* had finished training and was working well, and, secondly, by this point it was obvious that we would not be able to fix the GPU issues in time. In consultation with Prof. Roth, we decided that I should try to train my models on as much data as possible, since this approach had worked well for Marlene. We also decided that I should limit myself to only training four models, but with considerable *weight* differences, so I settled on the four model configurations seen in Table 1., named the *ultimate* models. First, I tried to train my models on *Galadriel*, but the training got interrupted repeatedly, because apparently its memory was full. So I decided to leave *Galadriel* to Marlene and whoever else was using it and switched to my own PC. Training took between 40 and 44 hours per model, but after a week all four models were trained and ready to be compared.

4 Analysis

For the analysis two things had to be done: Firstly, the models had to be evaluated on their own test and development set to compare their performance on the in-training languages. Secondly, in accordance with my research question, the models had to be evaluated on the Chinese test set to analyze the impact of the different *weight* distributions. The results, measured by the models' LAS score, can be seen below (Fig. 1.).

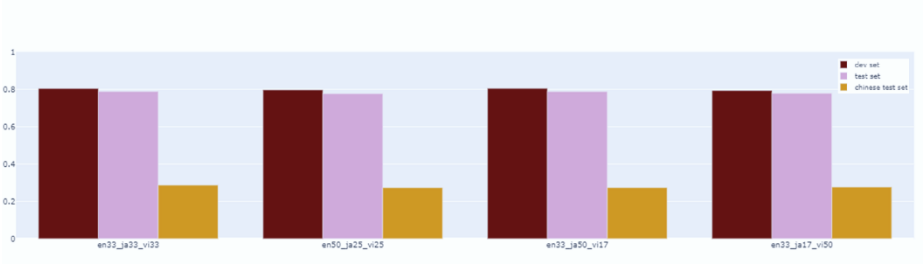

Fig. 1. The LAS scores of the four *ultimate* models, evaluated on their development set (red), test set (pink), and the Chinese test set (orange).

As can be seen above, all four models perform well when parsing the languages they were trained on. The LAS score for the development set lies between 80.5% and 79.3%,

and for the test set between 78.8% and 77.6%. In the case of Chinese, the performance dropped significantly for all four models: the LAS score lies between 28.6% and 27.3%. Hence, to answer the first part of my research question: No, the approach of taking three languages with typological similarities to "approximate" a target language, did not result in a working parser. The reasons for this could be twofold: Either the model simply needs more training data (without increasing its number of training languages), or it would profit from a wider variety of training languages – or both. Further research would be needed to evaluate either hypothesis.

The question remained whether the performance difference of the *ultimate* models when parsing Chinese is statistically significant. While the difference measured in percentages seemed miniscule, it was not as clear when looking at the absolute numbers:

Table 2. Performance of the *ultimate* models

Name	Weights (en, ja, vi)	LAS score in %	Correct LAS scores (out of 12012)
en33_ja33_vi33_ultimate	⅓ each	28.65	3441
en50_ja25_vi25_ultimate	50%, 25%, 25%	27.27	3276
en33_ja50_vi17_ultimate	⅓, ½, ⅙	27.30	3279
en33_ja17_vi50_ultimate	⅓, ⅙, ½	27.63	3319

The model *en33_ja33_vi33_ultimate* seemed to perform considerably better than the other three. Hence, I ran a statistical significance test to evaluate whether these performance differences could be explained by the flip of a coin and therefore were likely to just be a coincidence, or not. I can conclude: the difference in performance is statistically significant in all three cases with a 95% confidence, and the *en33_ja33_vi33_ultimate* model is indeed the best one. Furthermore, the model *en50_ja25_vi25_ultimate* is the model that comes closest to a model trained on the three languages "naturally" weighted by their treebank sizes, and it performs the worst.

This answers the second part of my research question: when it comes to the *weights* of the different languages in the training set, their composition does matter. When dealing with a set of languages with very different treebank sizes, the simple act of sampling up to equal *weights* does – at least in my case – result in Udapter performing significantly better. However, it is too early to say, if this result can be applied generally, or only holds true for the four languages I picked. It would be interesting to set up further tests, because, if generally applicable, this result would offer a simple and low cost tool for improving the performance of universal language parsers like Udapter.

5 Final Remarks

I wanted to test if it is possible to use the Udapter code in a more specialized setting. If one is only interested in parsing one language, rather than building a universal parser, can one make use of the special architecture of Udapter, pick typologically relevant languages, and run a much smaller model – resulting in a working parser? Regarding

this question, the results of my project are inconclusive. With more data it might be possible, or the model might need additional training languages.
However, I did come to a surprising conclusion: as mentioned above, the simple act of up sampling *without adding additional data* improved the performance of my model. For me this was rather unexpected. It would be interesting to test this idea on the original Udapter: Following my results, a model trained on the same 13 languages, but with the languages' *weights* adjusted to an even distribution, should yield better results. Though, keeping in mind the many issues in setting up and running the model, for now this has to remain a project for another time.

References

1. Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. *UDapter: Language adaptation for truly Universal Dependency parsing*. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 2302–2315, Online. Association for Computational Linguistics.
2. Comrie, B. (2018). *The World's Major Languages* (Third edition.). London :: Taylor and Francis.
3. Stefan Grünewald, Annemarie Friedrich, and Jonas Kuhn. 2021. *Applying Occam's Razor to Transformer-Based Dependency Parsing: What Works, What Doesn't, and What is Really Necessary*. In Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021), pages 131–144, Online. Association for Computational Linguistics.