Team 21a's Submission to the SIGTYP 2024 Shared Task on Word Embedding Evaluation for Ancient and Historical Languages

Anonymous ACL submission

Abstract

In this paper, we describe Team 21a's submission to the constrained track of the SIGTYP 2024 Shared Task. Using only the data provided by the organizers, we pretrained a transformer-based multilingual model, then finetuned it on the Universal Dependencies (UD) annotations of a given language for a downstream task. We also explored the cross-lingual capability of our trained models. [Our systems achieved]$_{\text{LJ}}$

1 Introduction

This paper describes Team 21a's submission to the *constrained* track of the SIGTYP 2024 Shared Task on Word Embedding Evaluation for Ancient and Historical Languages. Our general approach involves pretraining a transformer-based multilingual model on the shared task dataset, and then finetuning the pretrained model using the Universal Dependencies (UD) annotations of each language. Throughout this paper, we will refer to the pretrained model as LiBERTus. We also explored data sampling and augmentation techniques during the pretraining step to ensure better generalization performance.

Our systems achieved...[stuff]$_{\text{LJ}}$. Table 1 shows our systems' performance on the shared task test set.

We detail our resource creation, model pretraining, and finetuning methodologies. The source code for all experiments can be found on GitHub: `https://anonymous.4open.science/r/sigtyp-1F05/`.

	POS tag.	Morph. annot.	Lemma.
CHU			
COP			
FRO			
GOT			
GRC			
HBO			
ISL			
LAT			
LATM			
LZH			
OHU			
ORV			
SAN			

Table 1: SIGTYP 2024 Shared Task final results as evaluated on the test set. Cells where we obtained the best competition score are highlighted in green.

2 Methodology

2.1 Model Pretraining

The main purpose of pretraining is to obtain context-sensitive word embeddings that we will finetune further for each downstream task. We approach this by training a multilingual language model akin to the XLM-RoBERTa (Conneau et al., 2020) and multilingual BERT (Devlin et al., 2019) architectures.

Preparing the pretraining corpora. We constructed the pretraining corpora using the annotated tokens of the shared task dataset. Initially, we explored several data augmentation techniques to ensure that each language is properly represented based on the number of unique tokens. However, we found pretraining to be unstable when we upsampled tokens to achieve the same count as LATM, the most overrepresented language. In the end, we found that leaving the token distribution as-is leads to more stable pretraining and lower validation scores.

Pretraining the base model. Using the pretraining corpora, we trained a model with 126M parameters that will serve as a base for finetuning downstream tasks. LiBERTus follows RoBERTa's pre-

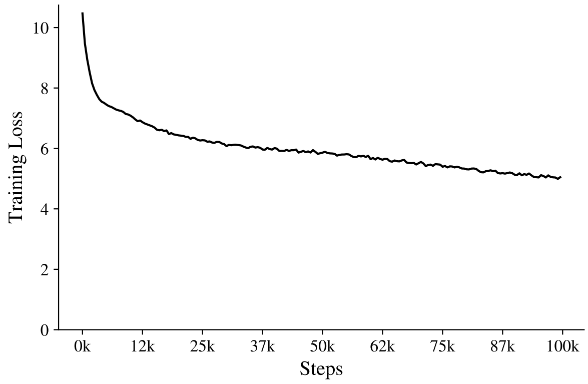

Figure 1: Training loss curve for the 126M-parameter model after 100k steps.

Hyperparameters	Value
Hidden size	768
Intermediate size	3072
Max position embed.	512
Num. attention heads	12
Hidden layers	12
Dropout	0.1

Table 2: Hyperparameter configuration for the LIBERTUS pretrained model.

training architecture (Liu et al., 2019) and takes inspiration from Conneau et al. (2020)'s work on scaling BERT models to multiple languages.

Our hyperparameter choices closely resemble that of the original RoBERTa implementation as seen in Table 2. We also trained the same BPE tokenizer (Sennrich et al., 2016) using the constructed corpora. During model pretraining, we used the AdamW optimizer with β_2=0.98 and a weight decay of 0.01. The base model underwent training for 100k steps with a learning rate of 2e-4. We used a learning rate scheduler that linearly warms up during the first 12k steps of the training process, then linearly decays for the rest. Figure 1 shows the training curve.

2.2 Model Finetuning

For each language, we finetuned a multitask model using spaCy (Honnibal et al., 2020). We used spaCy's tokenization rules for the majority of languages except for LZH, where we segmented on characters. The final system consists of a parts-of-speech (POS) tagger, morphological analyzer, and lemmatizer.

Parts-of-speech (POS) tagger. We employed a standard classifier that predicts a vector of tag probabilities for each token. Each POS tag is a unique class that we assign exclusively to a token. We trained a model by taking the context-sensitive vectors from our pretrained embeddings, and passing it to a linear layer with a softmax activation. The network is then optimized using a categorical cross-entropy loss.

Morphological analyzer. Similar to the POS tagger, we treat morphological annotation as a token classification task. Instead of directly modeling each feature, we made every unique combination of morphological features as a class. The limitation of this approach is that it can only predict combinations that were present in the training corpora.

Lemmatizer. We trained a neural-based edit tree lemmatizer (Müller et al., 2015) by first extracting an edit tree for each token-lemma pair. Because this process can result to hundreds of edit trees, we treat the problem of picking the correct tree as a classification task. Here, each unique tree serves as a class and we compute a probability distribution over all trees for a given token. To obtain the most probable tree, we passed the context-sensitive embeddings from our pretrained model to a softmax layer and trained the network with a cross-entropy loss objective. We set the minimum frequency of an edit tree to 3, and used the surface form of the token as a backoff when no applicable edit tree is found. Finally, we ensured that the lemmatizer checks at least a single tree before resorting to backoff.

Finetuning the pipelines. We trained each component of the system in parallel, although the final "pipeline" assembles them together using the spaCy framework. For all components, the pretrained embeddings are passed on to linear layer with softmax activation. Sometimes, the tokenization from the multilingual model does not align one-to-one with spaCy's tokenization. In such case, we use a pooling layer that computes the average of each feature to obtain a single vector per spaCy token.

During finetuning, we used the Adam optimizer with β_1=0.9, β_2=0.999 and a learning rate of 0.001. The learning rate warms up linearly for the first 250 steps, and then decays afterwards.

3 Results

Table 1 shows the test scores for the shared task. [Talk about how you ranked against other

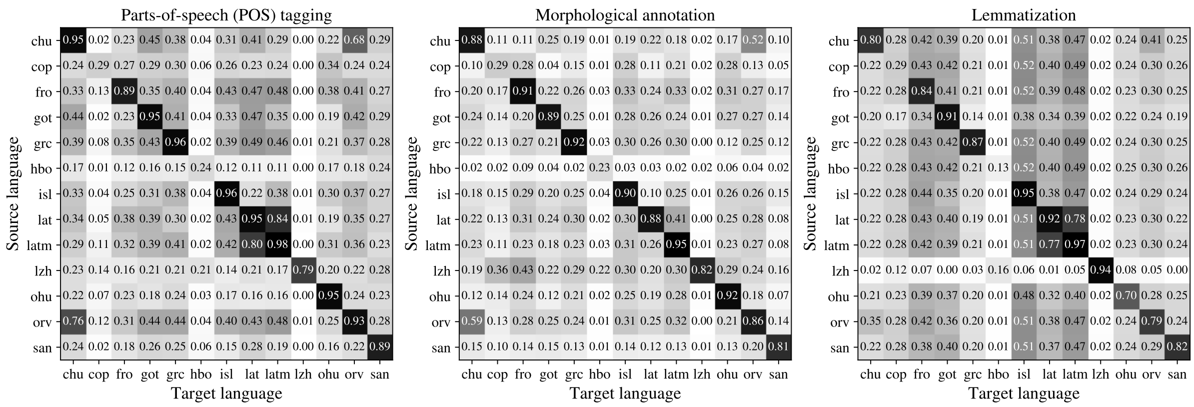

Figure 2: Cross-lingual evaluation given a monolingual model from one language and a validation set in another.

	POS tag.	Morph. annot.	Lemma.
CHU	0.947	0.876	0.803
COP	0.290	0.290	0.295
FRO	0.890	0.912	0.844
GOT	0.951	0.886	0.914
GRC	0.956	0.915	0.873
HBO	0.244	0.230	0.125
ISL	0.963	0.901	0.949
LAT	0.949	0.882	0.922
LATM	0.984	0.951	0.968
LZH	0.795	0.824	0.942
OHU	0.953	0.919	0.697
ORV	0.933	0.859	0.787
SAN	0.888	0.811	0.817

Table 3: F1-score results on the validation set. Cells with scores below 0.5 (random chance) are highlighted in red.

teams?]$_{\text{LJ}}$ In this section, we will outline our internal evaluations and benchmarking experiments.

3.1 Performance on the validation set

Table 3 shows the validation scores of our finetuned models. We achieved ≥70% performance in most language-task pairs. The top performers, calculated by taking the average across all tasks, are LATM (0.968), ISL (0.938), and LAT (0.918), whereas the bottom performers are SAN (0.839), COP (0.292), and HBO (0.200).

Error analysis for COP and HBO. We suspect that the main reason why we obtained worse scores on COP and HBO is due to our downstream tokenization process. To test this, we [look into how some sentences were tokenized vs. their actual tokenization?]$_{\text{LJ}}$

3.2 Evaluating cross-lingual capabilities

To test the cross-lingual capability of a language, we evaluated its finetuned model to the validation set of another. Figure 2 shows the results.

We found that it is possible to adapt a language onto another for morphological annotation and lemmatization. However, this does not extend to its morphology, as the validation set performs best only in the language it was trained on.

Some target languages tend to be cross-lingually receptive on lemmatization, i.e., many source languages can perform decently when applied to them. This observation is true for FRO, GOT, ISL, LAT, and LATM. Finally, there is also good cross-lingual compatibility between LAT and LATM—which is expected because they came from similar roots.

4 Conclusion

This paper describes Team 21a's system: a pretrained multilingual model (LiBERTus) finetuned on different languages for each downstream task. Our system obtained [describe your rank, how you stack, etc. etc.]$_{\text{LJ}}$

Our system's main strength is its downstream performance. The validation scores are high for the majority of languages. We also evaluated each language's cross-lingual capability and showed that transfer learning is possible especially on lemmatization. This approach can be a viable alternative on limited corpora.

Our training and benchmarking source code is on GitHub: `https://anonymous.4open.science/r/sigtyp-1F05/`. We will also release the pretrained multilingual model and finetuned pipelines in public shortly after the review period.

Limitations

Pretrained LM size. Due to compute constraints, we were only able to pretrain a model akin to the size of RoBERTa$_{\text{base}}$. We highly recommend pretraining a large LIBERTUS model to obtain performance gains if the resource allows.

Pretraining data mix. In the end, we didn't employ any sampling strategy to balance the token distribution of different languages during pretraining. [expand on this]$_{\text{LJ}}$

References

Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lester James Miranda, Ákos Kádár, Adriane Boyd, Sofie Van Landeghem, Anders Søgaard, and Matthew Honnibal. 2022. Multi hash embeddings in spaCy.

Thomas Müller, Ryan Cotterell, Alexander Fraser, and Hinrich Schütze. 2015. Joint lemmatization and morphological tagging with lemming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2268–2274, Lisbon, Portugal. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

A Appendix

A.1 Different sampling strategies on pretraining validation performance

We explored different sampling strategies and their effect on the pretraining validation loss curve as shown in Figure **??**. We ran the pretraining pipeline for 20k steps (one-fifths of the final training configuration) and measured the validation loss. The evaluation corpus was built from the validation set of the shared task, and we kept it the same throughout the experiment.

- **None:** we used the original dataset without any data sampling or augmentation.
- **Upsampling:** we upsampled each language to ensure that the number of their unique tokens is greater than or equal to the most dominant language.
- **Averaging:** we took the average number of unique tokens in the whole set and up-/downsampled each language based on this value.

Because any form of sampling resulted to unstable pretraining, we decided to stick with the dataset's original data distribution.

A.2 Finetuning a model per language vs. monolithic system

We investigated if finetuning a model per language is more effective against a monolithic system, i.e., training on the full multilingual annotated corpora. Here, we combined the training corpora for all languages, then shuffled them before batching. The merged dataset has 194,281 documents for training and 26,954 documents for validation. This means that the downstream model sees a language mix per training epoch.

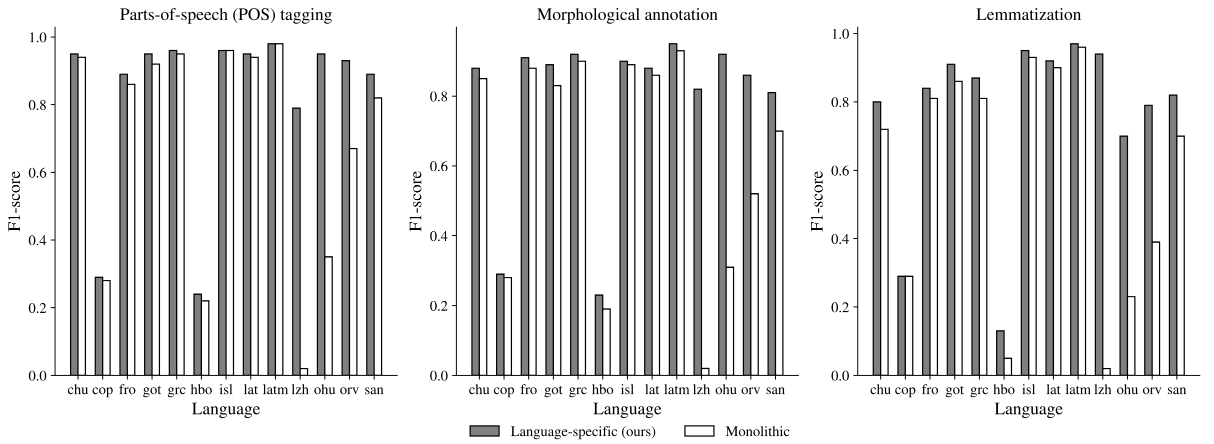

Figure 3: Comparison between training language-specific models versus a single monolithic model as evaluated on the validation set.

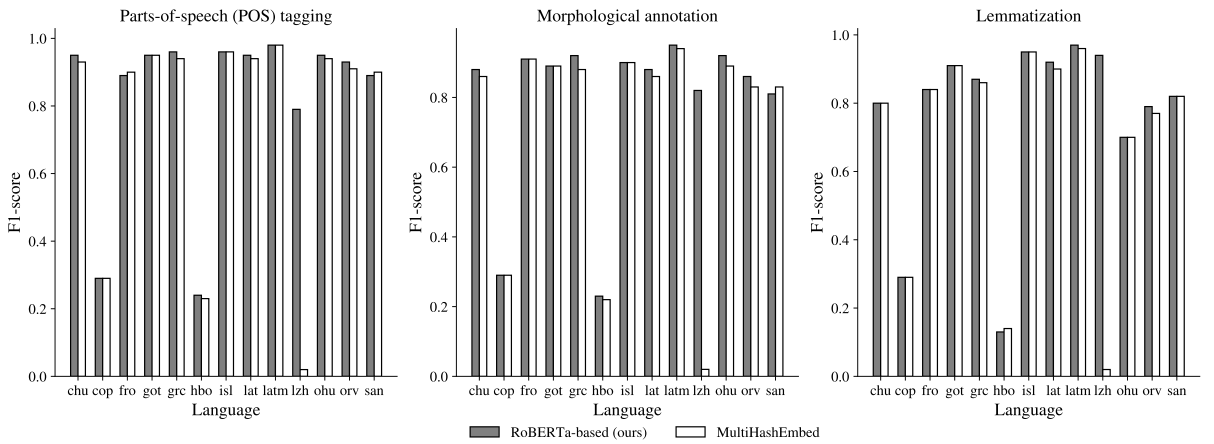

Figure 4: Comparison between a RoBERTa-based pretrained model and multi-hash embeddings (Miranda et al., 2022) as evaluated on the validation set.

As shown in Figure 3, finetuning a model per language still yields the best results. [talk more about what will happen]LJ

A.3 Alternative approach we considered

We considered using multi-hash embeddings (Miranda et al., 2022) as an alternative approach. Instead of pretraining, these embeddings use orthographic features (e.g., prefix, suffix, norm, shape) to create a word vector table. This approach also applies the hashing trick, inspired by Bloom filters (Bloom, 1970), to decrease the vector table's memory footprint.

Figure 4 shows the results in comparison to our final system. It is notable that simple orthographic features are competitive with our transformer-based model. However, we chose to submit the transformer-based pipeline as our final system because it still outperforms the multi-hash embed method in the majority of our language-task pairs. We still recommend investigating this approach further because the hash-embed method has noticeable efficiency gains in terms of model size.