System description for ProfNER - SMMH: Optimized fine tuning of a pretrained transformer and word vectors

David Fidalgo and **Daniel Vila-Suero** and **Francisco Aranda** and **Ignacio Talavera**
Recognai
https://www.recogn.ai
[david, daniel, francisco]@recogn.ai
ignaciotalaveracepeda@gmail.com

Abstract

This shared task system description depicts two neural network architectures submitted to the ProfNER track, among them the winning system that scored highest in the two sub-tasks 7a and 7b. We present in detail the approach, preprocessing steps and the architectures used to achieve the submitted results, and also provide a GitHub repository to reproduce the scores. The winning system is based on a transformer-based pretrained language model and solves the two sub-tasks simultaneously.

1 Introduction

The identification of professions and occupations (ProfNER) in Spanish[1] is part of the Social Media Mining for Health Applications (SMM4H) Shared Task 2021 (Magge et al., 2021). Its aim was to extract professions from social media to enable characterizing health-related issues, in particular in the context of COVID-19 epidemiology as well as mental health conditions.

ProfNER was the seventh track of the task and focused on the identification of professions and occupations in Spanish tweets. It consisted of two sub-tasks:

- **task 7a:** In this binary classification task, participants had to determine whether a tweet contains a mention of occupation, or not.
- **task 7b:** In this Named Entity Recognition (NER) task, participants had to find the beginning and end of occupation mentions and classify them into two categories: `PROFESION` (professions) and `SITUACION_LABORAL` (working status).

2 Our approach

We submitted two systems to each of the tasks described above, which share the same basic structure:

- a *backbone* model that extracts and contextualizes the input features
- a *task head* that performs task specific operations and computes the loss

In the backbone of both systems we take advantage of pretrained components, such as a transformer-based language model or skip-gram word vectors. The task head of both systems is very similar in that it solves task 7a and 7b simultaneously, and returns the sum of both losses.

For the first system we aimed to maximize the metrics of the competition with the constraint of using a single GPU environment. For the second system we tried to maximize the model's efficiency with respect to the model size and speed while maintaining acceptable performance.

Both systems were designed and trained using *biome.text*[2], a practical NLP open source library based on AllenNLP[3] (Gardner et al., 2017) and PyTorch[4] (Paszke et al., 2019).

2.1 Preprocessing

In a first step we transformed the given *brat*[5] annotations of task 7b to commonly used *BIO* NER tags(Ratinov and Roth, 2009). For this we used spaCy[6] (Honnibal et al., 2020) and a customized tokenizer of its "`es_core_news_sm`" language model, to make sure that the resulting word tokens and annotations always aligned well. In this step we excluded the entity classes not considered during evaluation. The same customized tokenizer was used to transform the predicted NER tags of our systems back to brat annotations during inference time.

[1] https://temu.bsc.es/smm4h-spanish/

[2] https://www.recogn.ai/biome-text

[3] https://allennlp.org/

[4] https://pytorch.org/

[5] http://brat.nlplab.org

[6] https://spacy.io/

tweet ID	word tokens	NER tags	classification label
1242604595463032832	[El, alcalde, ...]	[O, B-PROFESION, ...]	1
1242603450321506304	['', Trump, decide, ...]	[O, O, O, ...]	0
...	...	...	...

Table 1: Example of the format of our input data. NER tags are provided in the BIO encoding scheme.

To obtain the input data for our training pipeline, we added the tweet ID and the corresponding classification labels of task 7a to our word tokens and NER tags (see Table 1 for an example).

No data augmentation or external data was used for the training of our systems.

2.2 System 1: Transformer

In our first system, the backbone model consists of a transformer-based pretrained language model. More precisely, we use *BETO*, a BERT model trained on a big Spanish corpus (Cañete et al., 2020), which is distributed via Hugging Face's (Wolf et al., 2019) Model Hub[7] under the name `"dccuchile/bert-base-spanish-wwm-cased"`. For its usage we further tokenize the word tokens into word pieces with the corresponding BERT tokenizer, which also introduces the special BERT tokens `[CLS]` and `[SEP]` (Devlin et al., 2019). Since some of the word tokens cannot be processed by the tokenizer and are simply ignored (e.g. the newline character `"\n"`), we replace those problematic word tokens with a dummy token "æ", which is not ignored, and that allows the correct transformation of NER tags to brat annotations at inference time. The output sequence of the transformer is then passed on to the task head of the system.

In the task head we first apply a non-linear tanh activation layer to the `[CLS]` token, which we initialize with its pretrained weights (Devlin et al., 2019), before obtaining the logits of a linear classification layer that solves task 7a. The classification loss is calculated via the Cross Entropy loss function. To solve task 7b, we need to bridge the difference between the word piece features and predictions at a the level of word tokens. For this, we follow the approach of Devlin et al. (2019) who use a subword pooling in which the first word piece of a word token is used to represent the entire token, excluding the special BERT tokens. After the subword pooling we apply a linear classification layer and a subsequent Conditional Random Field (CRF) model that predicts a sequence of NER tags.

[7] https://huggingface.co/models

parameter	search space
learning rate	loguniform(5e-6, 1e-4)
weight decay	loguniform(1e-3, 1e-1)
warm-up steps	randint(0, 200)
batch size	choice([8, 16])
num epochs	choice([3, 4, 5])

Table 2: List of hyperparameters tuned during training. Search spaces define valid values for the hyperparameters and how they are sampled initially. They are provided as Ray Tune search space functions.

2.2.1 Training

For the parameter updates we used the AdamW algorithm (Loshchilov and Hutter, 2019) and schedule the learning rate with warm-up steps and a linear decay afterwards. We optimized the training parameters listed in Table 2 by means of the *Ray Tune* library[8] (Liaw et al., 2018) which is tightly integrated with *biome.text*. Our Hyperparameter Optimization (HPO) consisted of 50 runs (see Figure 1) using a tree-structured Parzen Estimator[9] as search algorithm (Bergstra et al., 2011) and the ASHA trial scheduler to terminate low-performing trials (Li et al., 2018). The reference metric for both algorithms was the overall F1 score of task 7b. The HPO lasted for about 6 hours on a *g4dn.xlarge* AWS machine with one Tesla T4 GPU.

We took the best performing model of the HPO, performed a quick sweep across several random seeds for the initialization[10] and finally employed the best configuration to train the system on the combined train and validation data set.

In further experiments, we tried to improve the validation metrics by switching to BILOU tags (Ratinov and Roth, 2009) or by including the entity classes not considered for the final evaluation, but could not find any significance differences.

[8] https://docs.ray.io/en/master/tune/

[9] https://github.com/hyperopt/hyperopt

[10] In hindsight, it would have been better to perform this sweep before the HPO and include the best performing random seeds in the HPO itself.

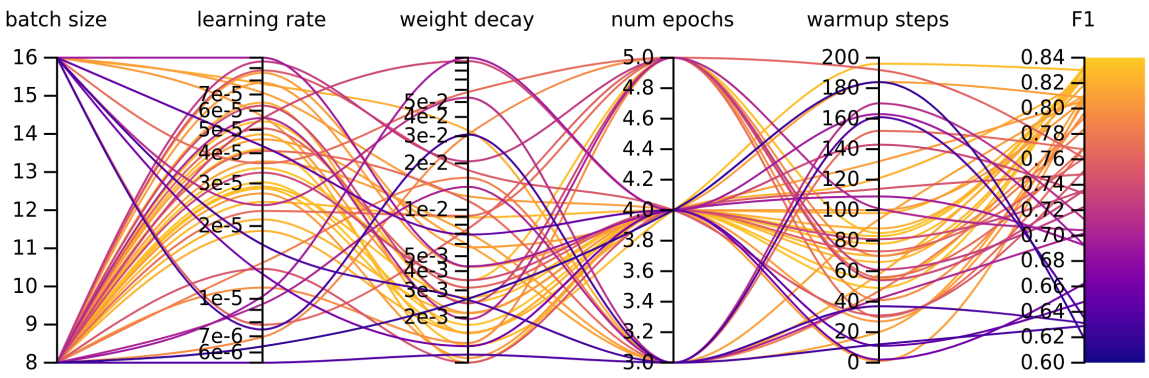

Figure 1: Distribution of the hyperparameters during the HPO for system 1. In total we executed 50 trials using a tree-structured Parzen Estimator as search algorithm and the ASHA trial scheduler to terminate low-performing trials early. The trial with the highest F1 NER score had a batch size of 8, a learning rate of 3.03e-05, a weight decay of 1.79e-3, was trained for 4 epochs and had 49 warm-up steps.

2.3 System 2: RNN

In our second system, the backbone model extracts word and character features, and combines them at a word token level. For the word feature we start from a cased version of skip-gram word vectors that were pretrained on 140 million Spanish tweets[11]. We concatenate these word vectors with the output of the last hidden state of a bidirectional Gated Recurrent Unit (GRU, Cho et al., 2014) that takes as input the lower cased characters of a word token. These embeddings are then fed into another larger bidirectional GRU, where we add contextual information to the encoding, and whose hidden states are passed on to the task head of the system.

In the task head we pool the sequence by means of a bidirectional Long short-term memory (LSTM, Hochreiter and Schmidhuber, 1997) unit and pass the last hidden state to a classification layer to solver task 7a. The classification loss is calculated via the Cross Entropy loss function. To solve task 7b, we pass each embedding from the backbone sequence through a feedforward network with a linear classification layer on top. The outputs of the classification layer are fed into a CRF model that predicts a sequence of NER tags.

The architectural choice of using GRU or LSTM units was solved via an HPO as described in the following training subsection.

2.3.1 Training

For the parameter updates we apply the same optimization algorithm and learning rate scheduler as for system 1. The comparatively small size of system 2 allowed us to perform extensive HPOs, not only for the training parameters but also for the architecture, and to some extent Neural Architecture Searches (NAS).

Backbone	
Word feature	300 dim, pretrained word vectors
Char feature	64 dim char vectors pooled by a GRU (bidirectional, 1 layer, 64 hidden size)
Backbone encoder	GRU (bidirectional, 1 layer, 512 hidden size)
Task Head	
Classification pooler	LSTM (bidirectional, 1 layer, 64 hidden size)
Feedforward	1 layer, 128 hidden size

Table 3: Details of our best RNN architecture.

In a first optimization run of 200 trials, we allowed wide ranges for almost all hyperparameters and tried out different RNN architectures, that is either LSTMs or GRUs. An example of a clearly preferred choice are the word embeddings pretrained with a skip-gram model over the ones pretrained with a a CBOW model (Mikolov et al., 2013). In a second run, we fixed obviously preferred choices and narrowed down the search spaces to the most promising ones.

For both HPO runs we applied the same search algorithm and trial scheduler as for system 1, and proceeded the same way to obtain the submitted version of system 2.

The resulting best RNN architecture is detailed in Table 3.

[11] https://zenodo.org/record/4449930

System	F1 Test (task 7a)	F1 Test (task 7b)	F1 Valid. (task 7a)	F1 Valid. (task 7b)	Model size (nr of params)	Inference time* (for 1 prediction)
1: Transformer	0.93	0.839	0.92	0.834	$\sim 1.1 \times 10^8$	24.5 ms $\pm$ 854 µs
2: RNN	0.88	0.764	0.85	0.731	$\sim 1.5 \times 10^7$	3.7 ms $\pm$ 103 µs

Table 4: Results for the two systems. Test results are provided with the systems trained on the combined training and validation data set, while the validation metric is taken from the best performing HPO trial. System 1 was the winning system in both ProfNER sub-tracks, while system 2 still scored above the arithmetic median of 0.85 and 0.7605 in both tasks.
*Mean value, computed on an i7-9750 H CPU with 6 cores.

3 Results

Table 4 presents the evaluation metrics of both systems on the validation and the test data sets, as well as the model size and its inference speed. With system 1 we managed to score highest on both ProfNER 7a and 7b sub-tracks (F1:0.93/P:0.9251/R:0.933 and F1:0.839/P:0.838/R:0.84, respectively), with an average of 8 points above the arithmetic median of all submissions. The much smaller and faster (by a factor of $\sim$ 7) system 2 still manages to score above the competitions median (F1:0.88/P:0.9083/R:0.8553 and F1:0.764/P:0.815/R:0.718, respectively), but performs significantly worse when compared to system 1.

We find a clear correlation between the classification F1 score and the F1 score of the NER task in our HPO runs, which signals that the feedback loop between the two tasks is in general beneficial and advocates solving both tasks simultaneously.

When comparing system 1 and 2, it seems that the amount of training data provided to the RNN architecture was not sufficient to match the transfer capabilities of the pretrained transformer, even with dedicated architecture searches and extensive hyperparameter tuning. This is corroborated by the fact that adding the validation data to the training data led to a clear performance boost for system 2, while the performance of system 1 stayed almost the same (compare the F1 Test and Validation metrics for task 7b in Table 4).

A possible path to improve system 1, which was not pursued due to time constraints, could be the inclusion of the gazetteers provided during the ProfNER track. We consider this path especially promising given the fact that the precision was always lower than the recall for both tasks.

We conclude that the exploitation of the transfer capabilities of a pretrained language model and its optimized fine tuning to the target domain, provides an conceptually easy system architecture and seems to be the most straight forward method to achieve competitive performance, especially for tasks where training data is scarce.

To help to reproduce our results, we provide a GitHub repository at `https://github.com/recognai/profner`.

Acknowledgments

This work was supported by the Spanish Ministerio de Ciencia, Inonvacion y Universidades through its Ayuda para contratos Torres Quevedo 2018 program with the reference number PTQ2018-009909.

References

James Bergstra, R. Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*, volume 24 of *Advances in Neural Information Processing Systems*, Granada, Spain. Neural Information Processing Systems Foundation.

José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. 2020. Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

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

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2018. A System for Massively Parallel Hyperparameter Tuning. *arXiv e-prints*, page arXiv:1810.05934.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv e-prints*, page arXiv:1807.05118.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Arjun Magge, Ari Klein, Ivan Flores, Ilseyar Alimova, Mohammed Ali Al-garadi, Antonio Miranda-Escalada, Zulfat Miftahutdinov, Eulàlia Farré-Maduell, Salvador Lima López, Juan M Banda, Karen O'Connor, Abeed Sarker, Elena Tutubalina, Martin Krallinger, Davy Weissenbacher, and Graciela Gonzalez-Hernandez. 2021. Overview of the sixth social media mining for health applications (# smm4h) shared tasks at naacl 2021. In *Proceedings of the Sixth Social Media Mining for Health Applications Workshop & Shared Task*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Lev Ratinov and Dan Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv e-prints*, page arXiv:1910.03771.