Meta-Learning for Few-Shot Named Entity Recognition

Cyprien de Lichy
Amazon, Alexa AI
101 Main St
Cambridge, MA, USA
cllichy@amazon.com

Hadrien Glaude
Amazon, Alexa AI
101 Main St
Cambridge, MA, USA
hglaude@amazon.com

William Campbell
Amazon, Alexa AI
101 Main St
Cambridge, MA, USA
cmpw@amazon.com

Abstract

Meta-learning has recently been proposed to learn models and algorithms that can generalize from a handful of examples. However, applications to structured prediction and textual tasks pose challenges for meta-learning algorithms. In this paper, we apply two meta-learning algorithms, Prototypical Networks and Reptile, to few-shot Named Entity Recognition (NER), including a method for incorporating language model pre-training and Conditional Random Fields (CRF). We propose a task generation scheme for converting classical NER datasets into the few-shot setting, for both training and evaluation. Using three public datasets, we show these meta-learning algorithms outperform a reasonable fine-tuned BERT baseline. In addition, we propose a novel combination of Prototypical Networks and Reptile.

1 Introduction

The usage of Natural Language Understanding (NLU) technologies has spread widely in the last decade thanks to the recent jump in accuracy due to Deep Neural Networks (DNN). In addition, DNN libraries have made easier than ever the productization of NLU technologies. Applications have spread in quality and quantity with the broadened usage of chat bots by customer services, the development of virtual assistants (e.g. Amazon Alexa, Google Home, Apple's Siri or Microsoft Cortana) and the need of document parsing (e.g. medical reports, receipts, tweets, news articles) for data extraction. These applications often rely on NER to locate and classify named entities in text. NER aims at extracting named entities (e.g. "artist", "city" or "restaurant_type") from a sequence of words. This problem is often approached (McCallum and Li, 2003) as a sequence labeling task that assigns to each word one of the different entity types or the "other" label for words that do not belong to any named entity.

The wide variety of applications has made the need for domain specific data the main bottleneck to train or fine-tune statistical models. This data is often acquired by running the application itself and collecting user inputs. Then, the annotation effort can be significantly reduced using active learning (Peshterliev et al., 2019) or semi-supervised learning (Cho et al., 2019b). However, to reach this bootstrapping stage, statistical models have to perform reasonably before being exposed to users. Indeed, low performing models can turn away users or shift the input distribution as users lose engagement with features that do not work.

Transfer learning (Do and Gaspers, 2019) is an efficient way to cope with the data shortage by extracting task-agnostic high-level features. In particular, for NER, fine-tuning language models (Peters et al., 2018; Devlin et al., 2018; Conneau and Lample, 2019) allows achieving state-of-the-art performances (Wang et al., 2018a). However, fine tuning to specific tasks still requires a reasonable amount of data, especially for a task like NER with large structured label spaces. In certain cases, for example to learn personalized models or for products with restricted budgets, only a handful "reference" examples are available. As we will show, in such scenarios where very few training examples are available, transfer learning has its limitations.

Few-Shot Learning (FSL) is a rapidly growing field of research, reviewed in Section 2, that aims at building models that can generalize from very few examples as detailed in (Miller et al., 2000; Koch et al., 2015). This area of research is motivated by the ability of humans and animals to learn object categories from few examples, and at a rapid pace. In particular, inductive bias (Mitchell, 1980) has been identified for a long time as a key component to fast generalization to new inputs. Previous work has suggested that meta-learning (Schmidhuber, 1987) can help quickly acquire knowledge from few examples by learning an inductive bias from a distribution of similar tasks but with different categories.

In this paper, we leverage recent progress made in transfer learning and meta-learning to address few-shot NER. First, we provide a novel definition of few-shot NER in Section 3.1 where few-shot NER aims at building models to solve NER tasks given only a handful of labeled utterances per entity type. Then, in Section 3.2, we define a transfer learning baseline consisting in fine-tuning a pre-trained language model (BERT Devlin et al., 2018) using only few examples. In addition, we introduce an extension of Prototypical Networks (Snell et al., 2017), a metric-based model, capable of handling structured prediction. In particular, we detail how it can be combined with Conditional Random Fields (CRF) (Lafferty et al., 2001). In Section 3.3, we explain how such models can be trained using meta-learning. In addition, we introduce the application of an optimization-based algorithm to NER, Reptile (Nichol et al., 2018), capable of meta-learning a better initialization model. We also propose a novel combination of Prototypical Networks and Reptile that brings the best of both worlds, performance and the ability to handle a different number of classes between training and testing. Finally, in Section 3.4, we show how to generate diverse and realistic FSL tasks, corresponding to the bootstrapping phase of NER systems, from classical NER datasets either for meta-training or meta-testing.

In Section 4, we conduct an extensive evaluation on three public datasets: SNIPS (Coucke et al., 2018), Task Oriented Parsing (TOP Gupta et al., 2018) and Google Schema-Guided Dialogue State Tracking (DSTC8 Rastogi et al., 2019) where we compare our three meta-learning approaches to the transfer learning baseline. Source code and datasets will be made available online.

2 Related Work

Few-shot learning has been addressed using metric-learning, data augmentation and meta-learning. Metric-learning relies on learning how to compare pairs (Koch et al., 2015) or triplets (Ye and Guo, 2018) of examples and use that distance function to classify new examples. Data augmentation through deformation has been known to be effective in image recognition tasks. More advanced approaches rely on generative models (Gupta, 2019; Hou et al., 2018; Zhao et al., 2019; Guu et al., 2018; Yoo et al., 2018), paraphrasing (Cho et al., 2019a) or machine translation (Johnson et al., 2019). All the methods above rely somewhat on transfer learning with the hope that representations learned in one domain can be applied to another one.

Meta-learning takes a different approach by trying to learn an inductive bias on a distribution of similar tasks that can be utilized to build models from very few examples. There are four common approaches. Model-based meta-learning relies on a meta-model to update or predict the weights of a task specific model (Munkhdalai and Yu, 2017). Generation-based meta-learning (Zhang et al., 2018; Schwartz et al., 2018) produces generative models able to quickly learn how to generate task specific examples, often in the feature space (Kumar et al., 2019). The other two approaches are explained in detail below.

Metric-based meta-learning is similar to nearest neighbors algorithms. In particular, several metric-based meta-learning methods (Vinyals et al., 2016; Snell et al., 2017; Rippel et al., 2015) have been proposed for few-shot classification where an embedding space or a metric is meta-learned and used at test time to embed the few support examples of new categories and the queries. Prediction is performed by comparing embedded queries and support examples. In many cases, the loss function is based on a distance between the supports and the queries. More advanced losses have been proposed in (Triantafillou et al., 2017; Wang et al., 2018b; Sung et al., 2018) for example based on triplet, ranking and max-margin losses. One of the issues with approaches listed above is that the distance is the same for all categories. Thus, Fort (2017); Hilliard et al. (2018) have explored scaling the distance for new categories.

Optimization-based meta-learning explicitly meta-learns an update rule or weight initialization that enables fast learning during meta-testing. In Ravi and Larochelle (2017), they use an LSTM meta-learner trained to be an optimization algorithm. However, this approach incurs a high complexity. In Finn et al. (2017), the authors explored with success using ordinary gradient descent in the learner and meta-learning the initialization weights. However, this algorithm named MAML, requires to back propagate through gradient updates and so rely on second order derivatives which are expensive to compute. They also proposed an algorithm, FOMAML, relying only on first order derivatives. This idea has been extended by Nichol et al. (2018) to propose an algorithm, Reptile, that does not need a training-test split for each task as explained in Section 3.3. Note that, Triantafillou et al. (2019) gives an overview of many meta-learning algorithms and propose a set of benchmarks to evaluate them. Finally, instead of just learning a model initialization, Li et al. (2017) propose to learn a full-stack Stochastic Gradient Descent (SGD), including update direction, and learning rate.

Few-Shot Learning on textual data has been explored recently, mostly for text classification tasks. Yu et al. (2018) propose to meta-learn a set of distances and learn a task-specific weighted combination of those. Jiang et al. (2018) build on top of MAML and attention mechanisms to propose an algorithm for text classification. Geng et al. (2019) focuses on sentiment and intent classification. Cheng et al. (2019) propose to use metric-based meta-learning to learn task-specific metrics that can handle imbalanced datasets. Recently, Bansal et al. (2019) proposed a new optimization-based meta-learning algorithm, LEOPARD, that outperforms strong baselines on several text classification problems (entity typing, natural language inference, sentiment analysis). Few-shot relation classification has also attracted some attention in the past two years, thanks to Han et al. (2018) who proposed a new dataset and using Prototypical Networks. Several works built on top of this to combine Prototypical Networks with attention models (Sun et al., 2019; Ye and Ling, 2019).

NER has been addressed in several works. In (Fritzler et al., 2019; Yang and Katiyar, 2020) the task of interest consists of recognizing one class of named entities, for tag set extension or domain transfer. In our work, we extend the N-way K-shot setting to structured prediction. (Hou et al., 2020) propose a CRF with coarse-grained transitions between abstract classes. In (Krone et al., 2020) the authors propose a task sampling algorithm based on intents which can result in leakage between meta-training and meta-testing sets. In (Hofer et al., 2018) the authors don't use pre-trained language models. As we will show subsequently our work differs significantly from those. First, our task sampling method, that can generate a very large amount of tasks, is key to learn efficiently an inductive bias. Second, we utilize pre-trained language models. Third, using a fine-grained CRF, amenable to meta-learning, our model can learn sequential dependencies between labels. Fourth, we fine-tune our meta-learned Prototypical Network per task and even utilize optimization-based meta-learning to improve the fine-tuning. Those contributions are central in achieving the best performance on few-shot NER as shown in Section 4.

3 Few-Shot Named Entity Recognition

3.1 Task Definition

We define the few-shot NER problem by describing what is a task. A task is defined by a set of N target entity types (examples of entity types could be "song", "city" or "date"), a small training set of $N \times K$ utterances (with their labels) called support set and another disjoint set of labeled utterances called query set. Similarly to Triantafillou et al. (2019), we refer to this setting as N-way-K-shot with the difference that we have a total of $N \times K$ support utterances rather than K examples for each of the N entity types, which is not feasible as one utterance might contain several entities. Thus, the number of mentions per entity type can be imbalanced. In addition, the support set follows the same distribution as the query set. Evaluation is performed by sampling a set of tasks from the meta-testing set. For each task, an NER model is learned from the support set. This model is evaluated on the query set. The performance is finally averaged across tasks. During meta-training, an additional set of meta-training tasks is available with disjoint entity types from the meta-testing set. Queries are used to train the meta-model. At meta-testing, this meta-model is tailored to the task using the support examples as mentioned above.

3.2 Prototypical Networks for NER

This paper builds on top of Prototypical Networks, introduced by Snell et al. (2017). Their model embeds support and query examples into a vector space. Then, one prototype per category is computed by taking the mean of its supports. Finally, queries are compared to prototypes using the euclidean distance. The distances are converted to probabilities using a Gibbs distribution. The model is meta-trained to predict the query labels using only few examples. This Section details the architecture of Prototypical Networks for sequence labeling. The next Section explains how the embedding function is meta-learned. Without meta-learning the architecture of Prototypical Networks does not bring any advantage over classical ones.

For a sequence labeling task, like NER, the difference is that to each word is assigned one label. Let $S = \{(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^n, \mathbf{y}^n)\}$ be a small support set of n labeled sequences where $\mathbf{x}^i = (x_1^i, \ldots, x_L^i)$ is an utterance of length L and $\mathbf{y}^i = (y_1^i, \ldots, y_L^i)$ a sequence of entity labels. For each entity type k, we compute a prototype c_k by embedding all words tagged as k using an embedding function f_θ where θ represents the meta-learned parameters. The fundamental difference with the common implementation of Prototypical Networks is that the embedding function f_θ utilizes the context of the current word to compute its representation in a vector space. Although, we should formally note $f_\theta(x_j^i; \mathbf{x}^i)$ the representation of x_j^i in the embedding space, we will just write $f_\theta(x_j^i)$ in the sequel to not overload equations. Thus, prototypes are defined by

$$c_k = \frac{1}{|S_k|} \sum_{x \in S_k} f_\theta(x), \tag{1}$$

where $S_k = \{x_j^i \mid y_j^i = k, (\mathbf{x}^i, \mathbf{y}^i) \in S\}$, i.e. the set of all tokens with a particular label k. Note that we compute one prototype per entity type and also one for "other". As mentioned in Section 5, we leave better handling of "other" for future work.

In this paper, we use BERT to generate embeddings for each word. More specifically, we used the pre-trained English BERT Base uncased model from (Wolf et al., 2019). This BERT model has 12 layers, 768 hidden states, and 12 heads. Then, we followed recommendation from Souza et al. (2019) to fine-tune BERT. Since BERT uses WordPiece sub-word units and NER labels are aligned to words, we elected to pick the last sub-word representation of a word as the final word representation. Then, we sum the outputs of the last 4 layers to get a word-level representation and then add dropout and a linear layer. [1] For our baseline model, the linear layer output size is the number of entity types plus "other". When using Prototypical Networks, the linear layer output size is 64. Then, distances to prototypes are computed for every word, giving the same output size than for the baseline model.

[1]In our experiments, we also tried an alternative architecture consisting of a frozen BERT model topped with three ELU-activation linear layers with dropout (Clevert et al., 2016), motivated by the fact that fine-tuning a large capacity model with very few examples might degrade the performances. As the first architecture worked better by a significant margin for the baseline, we did not pursue further this alternative.

Finally, in our experiments, we tried two different decoders. For the first one, we simply feed the distances into a SoftMax layer and use the negative log-likelihood (NLL) summed over all positions for the loss function, as follow,

$$p(y_t = k \mid \mathbf{x}) = \frac{e^{-\|f_\theta(x_t) - c_k\|^2}}{\sum_{k'} e^{-\|f_\theta(x_t) - c_{k'}\|^2}}, \tag{2}$$

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_t p(y_t \mid \mathbf{x}, \{c_k\}). \tag{3}$$

For our second decoder, we use a CRF, as Lample et al. (2016) have shown they are effective for NER when combined with neural networks. Using a CRF instead of making independent tagging decisions allows to model the dependencies between labels by considering a transition score between labels in addition to the standard emission scores to obtain a probability distribution,

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\left(\sum_t \left[U(x_t, y_t) + T(y_t, y_{t+1})\right]\right)}{\mathcal{Z}(\mathbf{x})}, \tag{4}$$

$$Z(x) = \sum_{\mathbf{y}'} \exp\left(\sum_t U(x_t, y_t') + T(y_t', y_{t+1}')\right) \tag{5}$$

where, T is a transition matrix, U the emission network and $\mathcal{Z}$ the partition function - a normalization factor used so that the probabilities sum to 1, equal to the sum of the scores over all label sequences. The loss function is the standard NLL. The emission network is the same as the SoftMax decoder.

For our baseline, the transition matrix is just a parameter of our network. However, estimating transitions between labels in the FSL setting is very prone to over-fitting as many transition pairs are likely to be absent from the limited training data. This intuition will be confirmed empirically in Section 4. Hence, we make use of prototypes and transfer learning to estimate the transition matrix. More specifically,

$$U(x_t, y_t) = -\|f_\theta(x_t) - c_{y_t}\|^2 \text{ and} \tag{6}$$

$$T(y_t, y_{t+1}) = g_\psi(c_{y_t}, c_{y_{t+1}}), \tag{7}$$

where the weights ψ of our neural network g are learned across tasks during meta-training and eventually fine-tuned during meta-testing. In our experiments, g is implemented as a feed-forward neural network on stacked prototype representation with one hidden layer of size 64 and ELU activation function. Looking only at the learning of the transition matrix during meta-training, this setting is equivalent to a standard training procedure that uses classes, represented by prototypes, as training examples and tries to predict transitions between them. Hence, we rely on the generalization capability of our transition DNN during meta-testing to handle new classes. We will see in Section 4, that using our Prototypical CRF decoder is very beneficial compared to a standard CRF.

3.3 Meta-Learning

In this Section, we introduce meta-learning and how it can be used to meta-learn initialization weights for the baseline architecture using Reptile, the embedding function in Prototypical Networks or both. In most cases, meta-learning algorithms, i.e. algorithms that learn how to learn, are typically comprised of two processes. The inner process is a traditional learning process capable of learning quickly using only a small number of task-specific examples. The outer loop, or meta-learning loop, slowly learns the inductive bias across a set of tasks. Thus, the objective of the outer loop is to improve generalization during the inner learning process. This is often achieved thanks to a meta-model. For Prototypical Networks the meta-model is the embedding function that defines the prototypes and the distance. For Reptile, the meta-model are the initialization weights that will be fine-tuned during meta-testing. During meta-testing, task specific models are derived from the meta-model and the support examples, for example by building prototypes or by gradient descent. Then, all queries are used to evaluate the task-specific model.

Meta-training runs in episodes. For each episode, a task or a batch of tasks is sampled. In our setting, we are only considering one task at a time. Then, from the current meta-model, a task specific model is built using the inner process and the support examples. The loss is computed using the queries and back-propagated through the inner process to update the meta-model. Good performance is often achieved when the inner process at meta-training and meta-testing are alike.

In the case of Prototypical Networks for sequence labeling, the meta-learner learns a representation amenable to generalization where queries can be compared to prototypes built from few support examples. Hence, the inner process just builds one prototype per entity type $k \in \mathcal{E}$, where $\mathcal{E}$ is the set of entity types for this task (including “other”) as described in Algorithm 1.

Algorithm 1 ProtoNet

```
INITIALIZE $\theta$
while has not converged do
    $\mathcal{E}, S, Q \leftarrow$ SAMPLETASK$(\mathcal{T}, K, N)$
    for all entity type $k$ in $\mathcal{E}$ do
        $c_k \leftarrow \frac{1}{|S_k|} \sum_{x \in S_k} f_\theta(x)$ as in eq. (1)
    end for
    $L \leftarrow$ NLL$(p,$ BATCH$(Q))$ where $p$ is defined in eq. (3) or eq. (4)
    $\theta \leftarrow$ UPDATE$(\theta, \frac{\partial L}{\partial \theta})$
end while
```

During meta-testing, we can simply compute the prototypes from the support examples as in eq. (1), in that case training is done without any backpropagation. However, in our experiments, see Section 4, we found that fine-tuning the meta-model using the task-specific supports was improving the performance. To fine-tune the model we further split the supports into two subsets using 80% to build the prototypes and the remaining to compute the loss and backpropagating it to update the model. By introducing this additional fine-tuning step at test time, the inner process now differs between meta-training and meta-testing. Similarly, for our baseline, we fine-tune our BERT-based model using the support utterances at meta-test time. In both cases, to better align meta-training and meta-testing, we turned to optimization-based meta-learning. Optimization-based meta-learning encompasses methods where the inner process consists in fine-tuning the meta-model. Back-propagating through the inner optimization loop allows computing a meta-gradient to update the meta-model as done in MAML. However doing so requires to compute second order derivatives. Instead, Reptile builds a first order approximation as shown in Algorithm 2, where T is the number of steps used to compute the first order approximation.

In addition, for MAML, the inner-loop optimization uses support examples, whereas the loss is computed using the queries. This way MAML optimizes for generalization. However, Reptile does not require a query-support split to compute the meta-gradient, which makes it a better candidate to be combined with Prototypical Networks.

Algorithm 2 Reptile

```
INITIALIZE $\theta_0$
while has not converged do
    $\mathcal{E}, S, Q \leftarrow$ SAMPLETASK$(\mathcal{T}, K, N)$
    for $t \in 1..T$ do
        $L \leftarrow$ NLL$(p,$ BATCH$(S \cup Q))$
        $\theta_t \leftarrow$ UPDATE$(\theta_{t-1}, \frac{\partial L}{\partial \theta_{t-1}})$
    end for
    $\theta_0 \leftarrow$ UPDATE$(\theta_0, \theta_T - \theta_0)$
end while
```

To combine MAML and Prototypical Networks, Triantafillou et al. (2019) use the same support examples to compute prototypes and to compute the loss for backpropagation in the MAML inner loop. However, having two disjoints support sets is preferable so as not to compare examples to prototypes computed from the same examples. With Reptile, this issue is alleviated altogether as shown in Algorithm 3.

Algorithm 3 Proto-Reptile

```
INITIALIZE $\theta_0$
while has not converged do
    $\mathcal{E}, S, Q \leftarrow$ SAMPLETASK$(\mathcal{T}, K, N)$
    for all entity type $k$ in $\mathcal{E}$ do
        $c_k \leftarrow \frac{1}{|S_k|} \sum_{x \in S_k} f_\theta(x)$ as in eq. (1)
    end for
    for $t \in 1..T$ do
        $L \leftarrow$ NLL$(p,$ BATCH$(Q))$
        $\theta_t \leftarrow$ UPDATE$(\theta_{t-1}, \frac{\partial L}{\partial \theta_{t-1}})$
    end for
    $\theta_0 \leftarrow$ UPDATE$(\theta_0, \theta_T - \theta_0)$
end while
```

In Algorithms 1 to 3, NLL stands for the negative log-likelihood function, BATCH for a function that samples a batch. $\mathcal{T}$ is the training set, K the number of shots, N the number of ways, S the support set and Q the query set, T is the number of steps in Reptile. In addition, UPDATE can be any optimizer, such that SGD or Adam (Kingma and Ba, 2015). In our experiments, we use Adam in Algorithm 1, and in the inner loop of Algorithm 3. For the outer loop of Algorithm 3, we use the classical SGD update rule without any momentum. Note that, each loop has its own learning rate. In addition, we used different learning rates for the BERT encoder and the rest of the network.

3.4 Generating Tasks for Training or Testing

To generate training and testing data from classical NER datasets, we first randomly partition entity types and utterances to either the train, the validation or the test split. Utterances are assigned based on the majority split of its entity types, counted per word. In other words, for a given utterance we count the number of words for entity types that are in each split and utterances are assigned to the partition that was the most represented in that utterance. In case of tie, priority is given to the test split, then the valid split and finally to the train split. Any entity contained in an utterance that is not in the corresponding partition is replaced with "other" to ensure, e.g., no test entities are seen during training. Finally, utterances with no entities are dropped. This task sampling procedure can both simulate a realistic few-shot NER testing setting and generate a large number of training tasks. During meta-training, having a diverse enough distribution of training tasks is crucial to learn an inductive bias effectively, similarly to having many examples helps generalization.

4 Experiments

4.1 Datasets and Pre-Processing

Experiments were conducted on the SNIPS (Coucke et al., 2018), Task Oriented Parsing (TOP Gupta et al., 2018) and Google Schema-Guided Dialogue State Tracking (DSTC8 Rastogi et al., 2019) datasets. For evaluation, we sampled 50 tasks from the meta-test set to average the Micro F1 across tasks. We use the Micro F1 metric introduced in (Tjong Kim Sang, 2002) that does not give any credit to partial matches. For SNIPS, we combine B and I labels from the BIO (Ramshaw and Marcus, 1995) encoding into a single label. For DSTC8, we used utterances from both the system and user, we discarded utterances containing more than 1 frame. For the TOP dataset, which contains hierarchical labels for slot labels and intents, we used the finest-grained entity types (the leaf nodes) as labels and discarded intents. We did not adhere to any pre-defined train, valid and test partitions, but followed our own task-based procedure defined in Section 3.4. Additional details about data preparation and datasets statistics are given in the appendix.

4.2 Hyper-Parameter Tuning

During meta-testing, only a few support examples are available to fine-tune the task specific model derived from the meta-model. As such, it is impractical to set aside some as a validation set for early stopping. However, early stopping is really important in the few-shot setting as the model can easily overfit. Hence, we find the best number of fine-tuning epochs on the validation split and then use it during meta-testing. For the baseline, this is the only purpose of meta-training.

For each algorithm (Baseline, ProtoNet, Reptile, Proto-Reptile) and decoder (SoftMax or CRF), we conducted an extensive hyper-parameter optimization (HPO) procedure using the built-in Bayesian optimization of AWS SageMaker (Amazon Web Services, 2017) on the SNIPS meta-validation dataset. The search space, the best hyper-parameters, the best performance and the training times are given in the appendix. We used the same hyper-parameters in all our experiments. However, after HPO, we retrained all our models with a number of meta updates and updates manually tuned per algorithm on each meta-validation dataset to avoid (meta-)stopping too early. All results on the meta-validation set and training times can be found in the appendix.

4.3 Results

We conducted four types of experiments. First, we compared all approaches on the three datasets using $N = 4$ and $K = 10$ in Table 1. Fine-tuning produces the largest gains, especially on SNIPS and TOP (less on DSTC8). Indeed, starting with the baseline, fine-tuning a pre-trained BERT model with aggressive dropout (0.9) is quite effective. Chen et al. (2019); Tian et al. (2020) also observed that transfer learning baselines are often competitive and neglected in FSL works. We also evaluated Prototypical Networks without fine-tuning at meta-test time using the supports. We refer to those algorithms by ProtoNet* and Proto-Reptile*. Compared to previous work on image recognition (Chen et al., 2019), fine-tuning the Prototypical Network seems to be extremely beneficial for textual application that builds on top of pre-trained language models instead of solely building the prototypes. Hence, combining optimization-based and metric-based meta-learning sounds a natural idea.

Comparing ProtoNet and Reptile, we can see that the Prototypical Network architecture helps generalization in the low data regime thanks to being instance-based. In addition, gains are even larger when combined with a CRF, with or without fine-tuning, in particular on DSTC8. Indeed, the CRF can only be slightly beneficial compared to using a simple SoftMax decoder for the Baseline and for Reptile. On the other hand, using our Prototypical CRF achieves a significant jump in Micro F1, especially on DSTC8, demonstrating that the transition network can generalize to new classes unseen at meta-training. We believe that, Reptile's meta-learning approach is inefficient because the initialization weights of the transition matrix do not have enough capacity to encode an inductive bias. Maybe other optimization-based meta-learning methods relying on external neural networks with larger capacity, e.g. a network that predicts the update direction as proposed by Li et al. (2017), could be more efficient than relying solely on the initialization weights to learn the inductive bias.

Comparing Reptile to Baseline and Proto-Reptile to ProtoNet, we see that optimization-based meta-learning can help significantly with fine-tuning. Although the gap is less impressive between Proto-Reptile to ProtoNet, Proto-Reptile obtains the best result in most cases. Comparing results between datasets, DSTC8 high diversity seems to be a real game changer for meta-learning. Indeed, all meta-learning approaches achieve twice or more the Baseline Micro F1. We argue that, the richer the task distribution, the better the learned inductive bias.

In our second experiment, we evaluated cross-domain transfer learning of the inductive bias by meta-training on TOP or DTSC8 and meta-testing on SNIPS. Note that early stopping was calibrated on the source meta-validation set, which gives an unfair advantage to the baseline to avoid overfitting. On inductive bias transfer, Proto and Proto-Reptile outperform the baseline by a small but statistically significant margin. As already observed, DTCS8 diversity is better to learn an inductive bias that can transfer across domain. Showing that task diversity is key to meta-learning.

In the third experiment, we varied N and K on the DSTC8 dataset to observe the performance gap between Proto-Reptile and the baseline. Results are plotted in the first row of Figure 1. As expected, Micro F1 increases when there are fewer entity types to discriminate (smaller N) or more examples

Meta-train dataset Meta-test dataset		SNIPS SNIPS	TOP SNIPS	DSTC8 SNIPS	TOP TOP	DSTC8 DSTC8
Baseline	CRF	76.84 ± 3.75	N/A	N/A	51.09 ± 5.06	34.57 ± 4.70
	SoftMax	73.68 ± 3.41	N/A	N/A	48.18 ± 4.78	35.18 ± 3.27
ProtoNet	CRF	$\mathbf{89.67 \pm 0.63}$	78.78 ± 1.14	82.88 ± 0.99	64.99 ± 3.51	75.69 ± 2.53
	SoftMax	87.11 ± 1.26	78.49 ± 1.37	80.37 ± 1.51	62.08 ± 3.58	66.39 ± 2.73
ProtoNet*	CRF	58.56 ± 1.78	44.75 ± 1.92	52.97 ± 2.04	29.53 ± 4.40	71.49 ± 3.81
	SoftMax	54.52 ± 1.82	43.23 ± 2.08	45.77 ± 1.26	28.34 ± 3.74	60.07 ± 2.62
Reptile	CRF	80.08 ± 3.58	74.85 ± 3.47	75.06 ± 3.32	57.18 ± 6.02	70.50 ± 2.60
	SoftMax	80.00 ± 3.51	75.82 ± 3.48	75.14 ± 3.45	57.64 ± 5.96	71.06 ± 2.77
Proto-Reptile	CRF	89.20 ± 0.89	$\mathbf{80.50 \pm 1.24}$	$\mathbf{82.96 \pm 1.19}$	$\mathbf{67.34 \pm 3.87}$	$\mathbf{78.96 \pm 1.60}$
	SoftMax	88.09 ± 0.90	77.53 ± 1.30	79.83 ± 1.74	64.06 ± 3.75	62.56 ± 2.14
Proto-Reptile*	CRF	49.98 ± 2.02	48.09 ± 1.85	51.63 ± 1.37	33.78 ± 3.41	75.22 ± 2.44
	SoftMax	58.41 ± 1.63	44.14 ± 1.88	37.93 ± 1.23	24.63 ± 3.68	58.09 ± 2.55

Table 1: Micro F1 averaged over 50 tasks. Results are reported with a Gaussian 95% confidence interval. Asterisks indicate that prototypes were not finetuned. The best result per column is in bold.

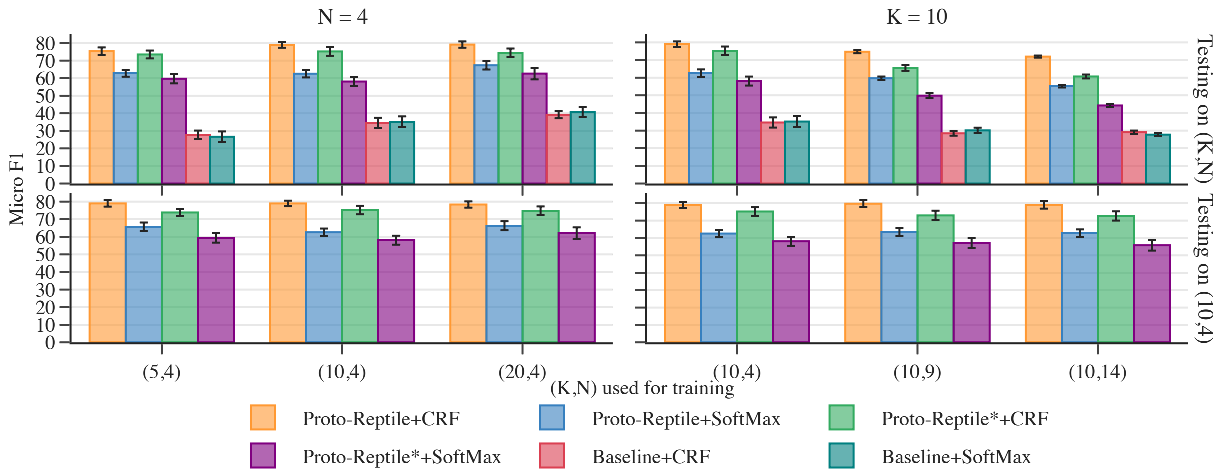

Figure 1: Micro F1 averaged over 50 tasks on N-way-K-shot DTSC8 for different value of (K, N). Error bars represent Gaussian 95% confidence intervals. In the first row of plots, (K, N) match between training and testing. In the second row, models trained on different N-way-K-shot settings are tested on 4-way-10-shot.

for each entity type (larger K). Indeed, either the problem becomes easier — fewer entity types to discriminate — or we get more data per entity type. Nevertheless, the Micro F1 increases faster with K for the baseline. We expect that, in the high data regime (very large K), the baseline would catch up to our approach. However, comparing those approaches in the high data regime would not be very relevant and the meta-learning would not scale.

Finally, we looked at meta-training on N-way-K-shot datasets but meta-testing on the 4-way-10-shot dataset in the second row of Figure 1. Training with more shots or more ways does not seem to improve or decrease performances significantly for Proto-Reptile. This demonstrate our approach is robust to variations in the meta-testing scheme, compared to what is usually observed in the few-shot literature. This is probably because we sample imbalanced support sets. All results in Figure 1 are reported numerically in the appendix.

5 Conclusions

In this paper, we have proposed a new definition of few-shot learning for NER, not relying a coarse-grain approach, like in (Fritzler et al., 2019), based on the intent to generate tasks. We have shown that, combining fine-tuning language models, CRF, diverse task generation, optimization-based and metric-based meta-learning, can significantly and consistently outperform transfer learning on three datasets. Also, our combination of Prototypical Network and Reptile is quite robust to mismatches in the number of shots or ways between meta-training and meta-testing. Thus, our approaches are effective to bootstrap NLU systems.

For future works, one specificity of few-shot NER has not been properly addressed yet. Although different in every tasks, the definition of the background class ("other") is partially shared between tasks. This assumption could be better leveraged in our approaches to transfer some of that knowledge across tasks instead of treating the background class as a different entity type in every tasks. Another interesting direction to explore is few-shot integration, when we have to build a model that performs well on tasks made of entity types seen and unseen during meta-training.

References

Amazon Web Services. 2017. AWS SageMaker. https://aws.amazon.com/sagemaker/.

Trapit Bansal, Rishikesh Jha, and Andrew McCallum. 2019. Learning to few-shot learn across diverse natural language classification tasks. *arXiv preprint arXiv:1911.03863*.

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. 2019. A closer look at few-shot classification. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Yu Cheng, Mo Yu, Xiaoxiao Guo, and Bowen Zhou. 2019. Few-shot learning with meta metric learners. In *Proceedings of the 3rd Workshop on Meta-Learning (MetaLearn 2019)*.

Eunah Cho, He Xie, and William M Campbell. 2019a. Paraphrase generation for semi-supervised learning in nlu. In *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*.

Eunah Cho, He Xie, John P Lalor, Varun Kumar, and William M Campbell. 2019b. Efficient semi-supervised learning for natural language understanding by optimizing diversity. In *Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019, Singapore, December 14-18, 2019*. IEEE.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7059–7069. Curran Associates, Inc.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Quynh Do and Judith Gaspers. 2019. Cross-lingual transfer learning with data selection for large-scale spoken language understanding. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1455–1460.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2185–2194, Hong Kong, China. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*. JMLR.org.

Stanislav Fort. 2017. Gaussian prototypical networks for few-shot learning on omniglot. *arXiv preprint arXiv:1708.02735*.

Alexander Fritzler, Varvara Logacheva, and Maksim Kretov. 2019. Few-shot classification in named entity recognition task. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 993–1000.

Ruiying Geng, Binhua Li, Yongbin Li, Xiaodan Zhu, Ping Jian, and Jian Sun. 2019. Induction networks for few-shot text classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3895–3904.

Rahul Gupta. 2019. Data augmentation for low resource sentiment analysis using generative adversarial networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. *arXiv preprint arXiv:1810.07942*.

Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. 2018. Generating sentences by editing prototypes. *Transactions of the Association of Computational Linguistics*.

Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. FewRel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4803–4809, Brussels, Belgium. Association for Computational Linguistics.

Nathan Hilliard, Lawrence Phillips, Scott Howland, Artëm Yankov, Courtney D Corley, and Nathan O Hodas. 2018. Few-shot learning with metric-agnostic conditional embeddings. *arXiv preprint arXiv:1802.04376*.

Maximilian Hofer, A. Kormilitzin, Paul Goldberg, and A. Nevado-Holgado. 2018. Few-shot learning for named entity recognition in medical text. *ArXiv*, abs/1811.05468.

Yutai Hou, Wanxiang Che, Yongkui Lai, Zhihan Zhou, Yijia Liu, Han Liu, and Ting Liu. 2020. Few-shot slot tagging with collapsed dependency transfer and label-enhanced task-adaptive projection network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Yutai Hou, Yijia Liu, Wanxiang Che, and Ting Liu. 2018. Sequence-to-sequence data augmentation for dialogue language understanding. *arXiv preprint arXiv:1807.01554*.

Xiang Jiang, Mohammad Havaei, Gabriel Chartrand, Hassan Chouaib, Thomas Vincent, Andrew Jesson, Nicolas Chapados, and Stan Matwin. 2018. Attentive task-agnostic meta-learning for few-shot text classification. In *Proceedings of the 2nd Workshop on Meta-Learning (MetaLearn 2018)*.

Andrew Johnson, Penny Karanasou, Judith Gaspers, and Dietrich Klakow. 2019. Cross-lingual transfer learning for japanese named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 182–189.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*.

Jason Krone, Yi Zhang, and Mona Diab. 2020. Learning to classify intents and slot labels given a handful of examples. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*.

Varun Kumar, Hadrien Glaude, Cyprien de Lichy, and Wlliam Campbell. 2019. A closer look at feature space data augmentation for few-shot intent classification. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 1–10, Hong Kong, China. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 282–289. Morgan Kaufmann.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2017. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

Andrew McCallum and Wei Li. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003*, pages 188–191. ACL.

Erik G Miller, Nicholas E Matsakis, and Paul A Viola. 2000. Learning from one example through shared densities on transforms. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1, pages 464–471. IEEE.

Tom M. Mitchell. 1980. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, NJ.

Tsendsuren Munkhdalai and Hong Yu. 2017. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning*.

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms.

Stanislav Peshterliev, John Kearney, Abhyuday Jagannatha, Imre Kiss, and Spyros Matsoukas. 2019. Active learning for new domains in natural language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language*

Technologies, Volume 2 (Industry Papers), pages 90–96, Minneapolis, Minnesota. Association for Computational Linguistics.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2019. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. *arXiv preprint arXiv:1909.05855*.

Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *In International Conference on Learning Representations*.

Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. 2015. Metric learning with adaptive density discrimination. *arXiv preprint arXiv:1511.05939*.

Jurgen Schmidhuber. 1987. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May.

Eli Schwartz, Leonid Karlinsky, Joseph Shtok, Sivan Harary, Mattias Marder, Abhishek Kumar, Rogerio Feris, Raja Giryes, and Alex Bronstein. 2018. Delta-encoder: an effective sample synthesis method for few-shot object recognition. In *Advances in Neural Information Processing Systems*.

Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc.

Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. 2019. Portuguese Named Entity Recognition using BERT-CRF. *arXiv e-prints*.

Shengli Sun, Qingfeng Sun, Kevin Zhou, and Tengchao Lv. 2019. Hierarchical attention prototypical networks for few-shot text classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 476–485, Hong Kong, China. Association for Computational Linguistics.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. 2018. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B. Tenenbaum, and Phillip Isola. 2020. Rethinking few-shot image classification: a good embedding is all you need? *CoRR*, abs/2003.11539.

Erik F. Tjong Kim Sang. 2002. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.

Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. 2017. Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems*.

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. 2019. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. 2016. Matching networks for one shot learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Yong Wang, Xiao-Ming Wu, Qimai Li, Jiatao Gu, Wangmeng Xiang, Lei Zhang, and Victor O. K. Li. 2018b. Large margin few-shot learning.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yi Yang and Arzoo Katiyar. 2020. Simple and effective few-shot named entity recognition with structured nearest neighbor learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Meng Ye and Yuhong Guo. 2018. Deep triplet ranking networks for one-shot recognition. *arXiv preprint arXiv:1804.07275*.

Zhi-Xiu Ye and Zhen-Hua Ling. 2019. Multi-level matching and aggregation network for few-shot relation classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational*

Linguistics, pages 2872–2881, Florence, Italy. Association for Computational Linguistics.

Kang Min Yoo, Youhyun Shin, and Sang goo Lee. 2018. Data augmentation for spoken language understanding via joint variational generation.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. 2018. Diverse few-shot text classification with multiple metrics. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. 2018. Metagan: An adversarial approach to few-shot learning. In *Advances in Neural Information Processing Systems*.

Zijian Zhao, Su Zhu, and Kai Yu. 2019. Data augmentation with atomic templates for spoken language understanding. *arXiv preprint arXiv:1908.10770*.

6 Appendix

6.1 Dataset preparation and statistics

This Section details how data was prepared. First, utterances without any named entities and the ones that are longer than 40 sub-word units (given by the BERT tokenizer) were removed. For each dataset, less than 1% of utterances were longer than 40 subwords. Removing long utterances allowed us to increase the computation efficiency significantly without impacting the results too much. datasets statistics are given in Table 2. For SNIPS, we used the data preprocessed in `https://github.com/MiuLab/SlotGated-SLU/`.

6.2 Hyper-parameters Tuning

This section describes the search space for hyper-parameters of each algorithm. The dropout parameter is the dropout of the additional layers on trop of BERT. In all settings, we used 0.1 for the BERT dropout and 64 for the batch size. During validation, we fine-tuned the current meta-model for 10 epochs, each epoch consisting of 64 batches, for each tasks. Validation Micro F1 was averaged over 5 sampled tasks with 128 queries each, using the same tasks in-between epochs to reduce the randomness. In the outer loop, we used early stopping with a patience of 4 and a maximum of 12 meta-epochs. At every meta-epoch, we reported the best epoch during the validation fine-tuning, to be used for meta-testing. The number of task per meta-epoch varies per algorithm and so is given in Tables 3 to 6 along with all the other parameters optimized. Bayesian optimization ran with 4 workers in parallel and a total of 30 training jobs, optimizing for the validation Micro F1. For Reptile-based algorithm, the number of steps stands for the number of steps used to compute the first order approximation (T in algorithms 2 and 3 of the main paper). Note that, Reptile was quite sensitive to hyper-parameter tuning and less stable than other approaches.

Training times are reported in Table 8. We used p2.xlarge AWS instances to train our models. Most of the training time actually is spent in validation that requires fine-tuning the meta-model.

In Figure 2, we reported how the performance of the best model increased overtime during hyper-parameters tuning. Because, we used Bayesian optimization instead of random search, it would have been very computationally intensive to compute the expected validation performance as suggested by (Dodge et al., 2019). Indeed, because random search produces i.i.d. trials, they can build an estimator of the validation performance and its variance at no cost. In our case, trials are dependant from the previous ones. We believe, Figure 2 provides a decent estimation of the budget needed for hyper-parameters tuning and how it affects the performance.

The best hyper-parameters per algorithm and per decoder is reported in Table 7 and the best validation Micro F1 is reported in Table 8.

6.3 Number of parameters

All our models used almost the same number of parameters. The differences introduced by the CRFs are negligible compared to BERT (110 millions parameters). Putting aside BERT, without Prototypical Networks, the linear layer on top of BERT adds $768 \times 4 \times N$ parameters and the CRF transition matrix adds $N \times N$ parameters. With Prototypical Networks, the linear layer on top of BERT adds $768 \times 4 \times 64$ parameters and the CRF transition network adds 64×64 parameters.

6.4 Results on the meta-validation set

Table 9 list the validation Micro F1, the training time, the best number of meta-epochs and the best number of epochs that is reused to stop the training during meta-testing. Note that most of the training time of meta-training is spend during validation.

	SNIPS			TOP			DSTC8		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
Utterances	9166	3832	1486	12868	13316	11547	107763	26562	26851
Entity types	27	5	7	20	6	8	84	18	20

Table 2: Datasets statistics.

Hyper-parameter	Range/Values	Scaling
Learning rate	$[5 \times 10^{-5}, 0.001]$	Logarithmic
BERT learning rate	$[1 \times 10^{-5}, 2 \times 10^{-4}]$	Logarithmic
Dropout	$[0.1, 0.9]$	Linear

Table 3: Hyper-parameter search space for the baseline.

Hyper-parameter	Range/Values	Scaling
# tasks	2048	Static
Learning rate	$[5 \times 10^{-5}, 0.001]$	Logarithmic
BERT learning rate	$[1 \times 10^{-5}, 2 \times 10^{-4}]$	Logarithmic
Meta learning rate	$[5 \times 10^{-5}, 0.001]$	Logarithmic
Meta BERT learning rate	$[1 \times 10^{-5}, 2 \times 10^{-4}]$	Logarithmic
Dropout	$[0.1, 0.9]$	Linear

Table 4: Hyper-parameter search space for ProtoNet.

Hyper-parameter	Range/Values	Scaling
# task	1024	Static
Learning rate	$[5 \times 10^{-5}, 0.001]$	Logarithmic
BERT learning rate	$[1 \times 10^{-5}, 2 \times 10^{-4}]$	Logarithmic
Meta learning rate	$[0.1, 1]$	Linear
Meta BERT learning rate	$[0.1, 1]$	Linear
Dropout	$[0.1, 0.9]$	Linear
# steps	$[1..10]$	Discrete

Table 5: Hyper-parameter search space for the Reptile.

Hyper-parameter	Range/Values	Scaling
# task	512	Static
Learning rate	$[5 \times 10^{-5}, 0.001]$	Logarithmic
BERT learning rate	$[1 \times 10^{-5}, 2 \times 10^{-4}]$	Logarithmic
Meta learning rate	$[0.1, 1]$	Linear
Meta BERT learning rate	$[0.1, 1]$	Linear
Dropout	$[0.1, 0.9]$	Linear
# steps	$[1..10]$	Discrete

Table 6: Hyper-parameter search space for Proto-Reptile.

Algorithm	Decoder	# steps	Meta LR	Meta BERT LR	LR	BERT LR	Dropout
Baseline	SoftMax	N/A	N/A	N/A	4.35×10^{-4}	8.94×10^{-5}	0.9
	CRF	N/A	N/A	N/A	1×10^{-3}	3.94×10^{-5}	0.897
ProtoNet	SoftMax	N/A	8.2×10^{-4}	6.88×10^{-5}	9.53×10^{-4}	1.99×10^{-5}	0.393
	CRF	N/A	9.73×10^{-4}	6.21×10^{-5}	3.54×10^{-4}	2.24×10^{-5}	0.558
Reptile	SoftMax	10	0.909	0.126	3.32×10^{-4}	7.91×10^{-5}	0.104
	CRF	10	0.107	0.188	7.97×10^{-4}	4.45×10^{-5}	0.71
Proto-Reptile	SoftMax	2	0.641	0.580	4×10^{-4}	1.05×10^{-5}	0.496
	CRF	10	0.847	0.329	6.92×10^{-4}	1.15×10^{-5}	0.446

Table 7: Best hyper-parameters found using Bayesian optimization.

Algorithm	Decoder	Micro F1	Best # meta-epochs	Best # epochs	Training time
Baseline	SoftMax	61.04 ± 5.23	N/A	10	01:33:44
	CRF	59.49 ± 3.12	N/A	9	01:43:42
ProtoNet	SoftMax	70.48 ± 3.83	5	10	11:19:57
	CRF	73.65 ± 2.92	8	5	14:53:58
Reptile	SoftMax	71.88 ± 2.19	8	3	13:57:09
	CRF	70.64 ± 2.30	4	2	16:56:16
Proto-Reptile	SoftMax	70.89 ± 2.98	10	5	10:57:57
	CRF	76.18 ± 4.22	6	8	24:18:03

Table 8: Best validation run found using Bayesian optimization. Micro F1 is averaged over 5 tasks. Results are reported with Gaussian 95% confidence interval. However, note that the same 5 validations tasks are used for every algorithms and models, which introduces a beneficial dependency.

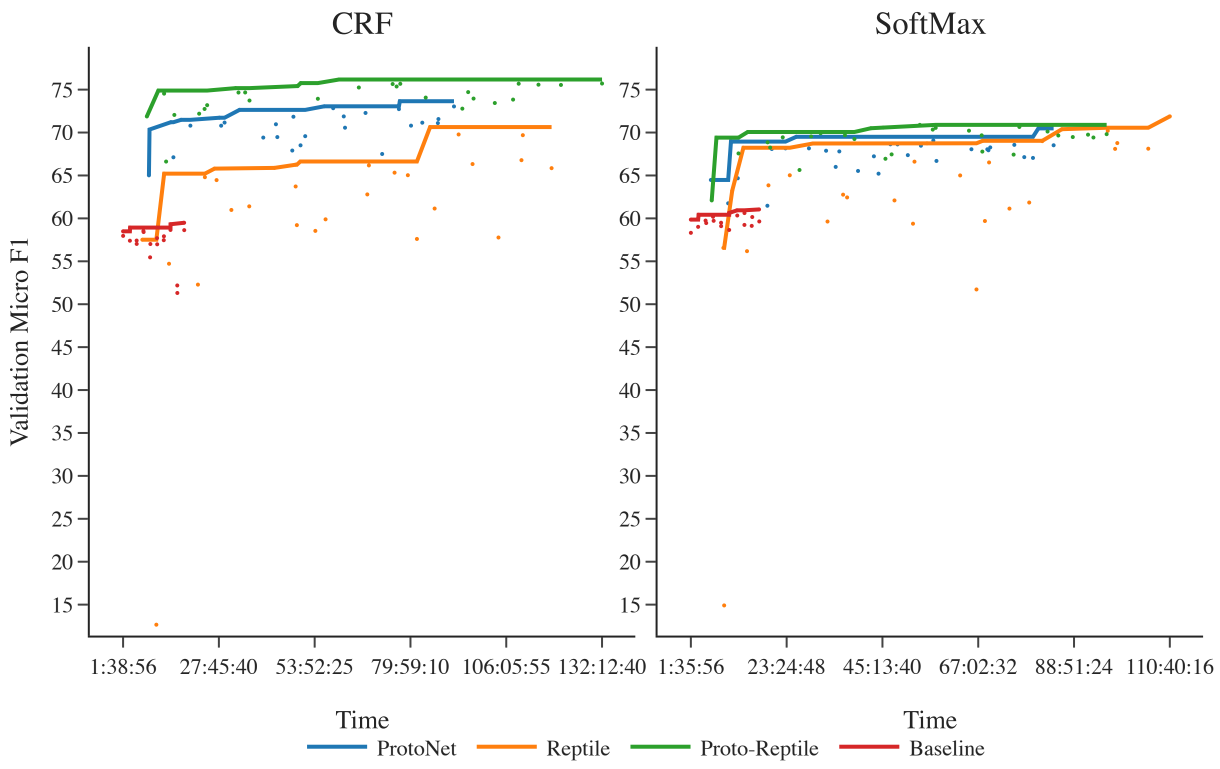

Figure 2: Averaged Micro F1 over the same 5 tasks randomly drawn from the SNIPS validation split during Bayesian optimization of the hyper-parameters. Each dot represents one meta-training. The lines indicate the best model performance overtime.

Dataset	Algorithm	Decoder	Micro F1	# Tasks	# Meta Epochs	# Epochs	Time
SNIPS	ProtoNet	CRF	63.60 ± 5.43	5	9	2	50:31:06
		SoftMax	60.61 ± 5.04	5	11	3	54:30:54
	Reptile	CRF	60.02 ± 5.30	5	20	12	11:55:21
		SoftMax	58.18 ± 4.54	5	20	13	11:14:58
	Proto-Reptile	CRF	67.13 ± 4.01	5	4	13	55:05:08
		SoftMax	62.05 ± 3.38	5	7	13	33:50:06
	Baseline	CRF	48.82 ± 4.37	8	N/A	14	5:16:26
		SoftMax	45.01 ± 4.75	8	N/A	11	4:47:26
TOP	ProtoNet	CRF	71.16 ± 5.77	5	8	1	72:04:31
		SoftMax	67.68 ± 5.05	5	4	4	65:59:54
	Reptile	CRF	59.16 ± 6.38	5	10	10	8:51:06
		SoftMax	60.87 ± 5.55	5	5	4	5:44:05
	Proto-Reptile	CRF	72.29 ± 4.37	5	2	14	72:06:05
		SoftMax	69.90 ± 4.55	5	12	12	72:06:16
	Baseline	CRF	59.16 ± 4.26	8	N/A	14	10:38:27
		SoftMax	55.85 ± 4.61	8	N/A	5	9:31:50
DSTC8	ProtoNet	CRF	82.29 ± 4.13	5	17	15	72:08:26
		SoftMax	73.56 ± 6.46	5	5	8	35:56:08
	Reptile	CRF	75.03 ± 5.62	5	18	2	16:36:53
		SoftMax	75.01 ± 3.35	5	22	5	17:30:08
	Proto-Reptile	CRF	83.83 ± 4.13	5	6	10	72:07:55
		SoftMax	75.87 ± 4.80	5	12	10	33:42:35
	Baseline	CRF	47.08 ± 7.02	8	N/A	14	10:42:08
		SoftMax	42.17 ± 8.23	8	N/A	1	10:00:21

Table 9: Validation Micro F1 with Gaussian 95% confidence interval and training times.