Learning to Few-Shot Learn Across Diverse Natural Language Classification Tasks

Trapit Bansal[*†] and **Rishikesh Jha**[*‡] and **Andrew McCallum**[†]
[†]University of Massachusetts, Amherst
[‡]Code for Science and Society
{tbansal,rishikeshjha,mccallum}@cs.umass.edu

arXiv:1911.03863v1 [cs.CL] 10 Nov 2019

Abstract

Self-supervised pre-training of transformer models has shown enormous success in improving performance on a number of downstream tasks. However, fine-tuning on a new task still requires large amounts of task-specific labelled data to achieve good performance. We consider this problem of learning to generalize to new tasks with few examples as a meta-learning problem. While meta-learning has shown tremendous progress in recent years, its application is still limited to simulated problems or problems with limited diversity across tasks. We develop a novel method, LEOPARD, which enables optimization-based meta-learning across tasks with different number of classes, and evaluate existing methods on generalization to diverse NLP classification tasks. LEOPARD is trained with the state-of-the-art transformer architecture and shows strong generalization to tasks not seen at all during training, with as few as 8 examples per label. On 16 NLP datasets, across a diverse task-set such as entity typing, relation extraction, natural language inference, sentiment analysis, and several other text categorization tasks, we show that LEOPARD learns better initial parameters for few-shot learning than self-supervised pre-training or multi-task training, outperforming many strong baselines, for example, increasing F1 from 49% to 72%.

1 Introduction

Learning to learn (Schmidhuber, 1987; Bengio et al., 1992; Thrun and Pratt, 2012) from limited supervision is an important problem with widespread application in areas where obtaining labelled data for training large models can be difficult or expensive. We consider this problem of *learning in k-shots* for natural language processing (NLP) tasks, that is, given k labelled examples of a new NLP task learn to efficiently solve the new task. Recently, self-supervised pre-training of large transformer models using language modeling objectives (Devlin et al., 2018; Radford et al., 2019; Yang et al., 2019) has achieved tremendous success in learning general-purpose parameters which are useful for a variety of downstream NLP tasks. While pre-training is beneficial, it is not optimized to allow fine-tuning with limited supervision and such models can still require large amounts of task-specific data for fine-tuning (Yogatama et al., 2019).

On the other hand, meta-learning methods have been proposed as effective solutions for few-shot learning. Such methods can be broadly categorized into metric-based and optimization-based methods. Metric-based meta-learning methods, such as matching networks (Vinyals et al., 2016), aim to learn a general-purpose neural model to encode data-points into an embedding space along with a (learned) distance metric which reduces the prediction problem to a matching problem. While optimization-based methods, such as model-agnostic meta-learning (MAML) (Finn et al., 2017), learn a good initialization point for a neural model from which learning using stochastic gradient descent can reach the optimal point for a new task with only a few examples.

Existing applications of such meta-learning methods have shown improved performance in few-shot learning for vision tasks such as learning to classify new image classes within a similar dataset. However, these applications are often limited to simulated datasets where each classification label is considered a task. Moreover, their application in NLP has followed a similar trend (Han et al., 2018; Yu et al., 2018; Guo et al., 2018; Mi et al., 2019; Obamuyide and Vlachos, 2019; Geng et al., 2019). We move beyond simulated tasks to investigate meta-learning performance on gener-

[*]Equal Contribution

alization outside the training tasks, and focus on a diverse task-set with different number of labels across tasks. Since the input space of natural language is shared across all NLP tasks, it is possible that a meta-learning approach generalizes to unseen tasks, as also conjectured by Yogatama et al. (2019).

As MAML-based approaches are agnostic to the model architecture and task specification, they are a lucrative candidate for learning to learn from diverse tasks. However, they require sharing model parameters, including softmax classification layers across tasks. Moreover, MAML learns a single initialization point across tasks. As such, they have been limited to fixed n-way classification-type tasks. This poses a barrier for learning to learn across diverse tasks, where different tasks can have potentially disjoint label spaces. Contrary to this, multi-task learning naturally handles disjoint label sets, while still benefiting from sharing statistical strength across tasks. However, to solve a new task at test time, multi-task learning would require training a new classification layer for the task.

On the other hand, metric-learning approaches (Vinyals et al., 2016; Snell et al., 2017) being non-parametric in nature can handle varied number of classes as long as they are provided with some examples of each class. However, as the number of labelled examples increase, these methods do not adapt to leverage the larger label set as nearest neighbor matching becomes computationally expensive. Due to this, their performance with increasing data size lags behind MAML-based methods, which can approach supervised models in the limit.

We address these concerns and make the following contributions: (1) we develop a MAML-based meta-learning method, **LEOPARD**[1], which can be applied across tasks with disjoint label spaces and can continuously learn to adapt from increasing amounts of labeled data; (2) we train LEOPARD with a large pre-trained transformer model, BERT (Devlin et al., 2018), as the underlying neural architecture, and show that it is possible to learn better initialization parameters for few-shot learning than that obtained from just self-supervised pre-training or pre-training followed multi-task learning; (3) we consider a broad and challenging set of NLP tasks including *entity typing, relation classification, natural language inference, sentiment classification, and various other text categorization tasks*; (4) we study how meta-learning, multi-task learning and fine-tuning (using pre-trained BERT model) perform for few-shot learning of completely new tasks, analyze merits/demerits of parameter efficient meta-training for LEOPARD, and study how various train tasks affect performance on target tasks.

To the best of our knowledge, this is the first application of meta-learning methods in NLP which evaluates on test tasks which are significantly different than training tasks and goes beyond simulated classification tasks or domain-adaptation tasks (where train and test tasks are similar but from different domains). We discuss relevant background in Section 2, introduce our approach LEOPARD in 3, discuss experimental results and analysis in Section 4 and discuss related work in Section 5.

2 Background

In supervised learning of a task T, we are given a dataset $\mathcal{D}_T = \{(x_j, y_j)\}$ sampled from a distribution: $\{(x_j, y_j)\} \sim P(\mathcal{X})P_T(\mathcal{Y}|\mathcal{X})$, where $\mathcal{X}$ is the input space, $\mathcal{Y}$ is the output space and $P_T(\mathcal{Y}|\mathcal{X})$ is the task-specific distribution over the output space given input. The aim is then to learn a parameterized model $f_\theta : x_j \rightarrow \hat{y}_j$, with parameters θ, that maps inputs x_j with true label y_j to a prediction $\hat{y}_j$, using a task-specific loss $\mathcal{L}_T$. For example, for the task of natural language inference, x is a pair of premise and hypothesis, and y is whether the hypothesis is true (entailment), false (contradiction) or undetermined (neutral) given the premise.

In meta-learning, we consider a meta goal of learning across multiple tasks and assume a distribution over tasks $T_i \sim P(\mathcal{T})$. We follow the episodic learning framework of Vinyals et al. (2016) which minimizes train/test mismatch for few-shot learning. We are given a set of M training tasks $\{T_1, \ldots, T_M\}$, where each task instance potentially has a large amount of training data. In order to simulate k-shot learning during training, in each episode (i.e. a training step) a training task T_i is sampled with a training set $\mathcal{D}_i^{tr} \sim T_i$, consisting of only k examples (per label) of the task and a validation set $\mathcal{D}_i^{val} \sim T_i$, containing several other examples of the same task. The model f is then trained on $\mathcal{D}_i^{tr}$ using the task loss $\mathcal{L}_i$,

[1] **L**earning to g**e**nerate s**o**ftmax **pa**rameters fo**r** **d**iverse classification

and then tested on $\mathcal{D}_i^{val}$. The model's performance on $\mathcal{D}_i^{val}$, measured using the loss, is then used to adjust the model parameters. Here the validation error of the sampled task T_i serves as the training error for the meta-learning process. At the end of training, the model is then evaluated on a new task $T_{M+1} \sim P(T)$ where again the train set of T_{M+1} consists of only k examples per label, and the model can use it's learning procedure to adapt to the task T_{M+1}. It is then evaluated on a large set of new test examples from this new task T_{M+1}. Note that the datasets $\mathcal{D}_i^{tr}$ and $\mathcal{D}_i^{val}$ are also referred to as *meta-training* and *meta-validation* sets. We next discuss two popular approaches to meta-learning which are pertinent to our work: MAML and Prototypical Networks. Refer to Section 5 for a discussion of other meta-learning methods.

2.1 Model-Agnostic Meta-Learning (MAML)

MAML (Finn et al., 2017) is an approach to optimization based meta-learning where the goal is to find a good initial point for model parameters θ, which through few steps of gradient descent, can be adapted to yield good performance on a new task. Learning in MAML consists of an *inner loop*, which consists of gradient-based learning on the task-specific objective, and an *outer-loop* which refines the initial point in order to enable fast learning across the set of tasks. Thus, given a task T_i with training datasets $\mathcal{D}_i^{tr}$ sampled during an episode, MAML's inner loop adapts the model parameters θ as:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_i(\theta, \mathcal{D}_i^{tr}) \tag{1}$$

Typically, more than one step of gradient update are applied sequentially. The hyperparameter α can also be meta-learned in the outer-loop (Li et al., 2017). The model parameters θ are then trained by back-propagating through the inner-loop adaptation across tasks, with the meta-objective of minimizing the error on the respective task validation sets $\mathcal{D}_i^{val}$:

$$\theta \leftarrow \theta - \beta \sum_{T_i \sim P(\mathcal{T})} \mathcal{L}_i(\theta', \mathcal{D}_i^{val}) \tag{2}$$

More sophisticated routines for optimization, such as Adam (Kingma and Ba, 2014), can also be used here instead of vanilla SGD. Note that even though MAML is trained to generate a good initialization point for fast adaptation to a new task, since the inner-loop adaptation for each task also employs gradient-based learning to learn task-specific parameters, it's performance can approach regular supervised learning in the limit of large data.

2.2 Prototypical Networks

Prototypical networks (Snell et al., 2017) is a metric-learning based approach to meta-learning where the idea is to learn to embed data-points in an embedding space and use the few-shot training data as support to construct per class prototypes in this embedding space. The classification label for any new data point is then obtained as the class of the prototype that is closest to this data point in the embedding space. Concretely, the training dataset in an episode is first partitioned based on the class labels, $C_n = \{x_j | y_j = n\}$ where $n \in [N]$ and N is the number of classes in the current task. Then f_θ acts a neural encoder for data points in each C_n to generate a prototype for the n-th class as $\tilde{c}_n$. Now, given a query point $x^* \in \mathcal{D}_i^{val}$, the probability that x^* belongs to class n is given by computing the distance to the class prototype and passing the results through a softmax:

$$p(y^*|x^*, \mathcal{D}_i^{tr}) = \underset{n \in \{1,\ldots,N\}}{\text{softmax}} \{-d(\tilde{c}_n, f_\theta(x^*))\}$$

$$\tilde{c}_n = \frac{1}{|C_n|} \sum_{x_j \in C_n} f_\theta(x_j)$$

where $d(\cdot, \cdot)$ is the euclidean distance (Snell et al., 2017). Optimization of θ then proceeds by minimizing the validation loss on new query examples for each task T_i: $\sum_{T_i \sim P(\mathcal{T})} \mathcal{L}_{T_i}^{val}(f_{\theta'})$. While protoypical networks can perform well in few-shot regime and have nice interpretation as an end-to-end learned nearest neighbor, they don't learn to continuously adapt the parameters with data. Moreover, with more data computing per-class prototypes can be computationally expensive as size of C_n becomes large.

3 Model

In this section we describe our proposed method, LEOPARD, for learning new NLP classification tasks with k-examples. Fig. 1 shows a high-level description of the model. Our approach builds on the MAML framework and addresses some of it's limitations when applied to a diverse set of tasks with different number of classes across tasks. Our model consists of three main components: (1) a shared neural input encoder which generates feature representations useful across tasks; (2) a soft-

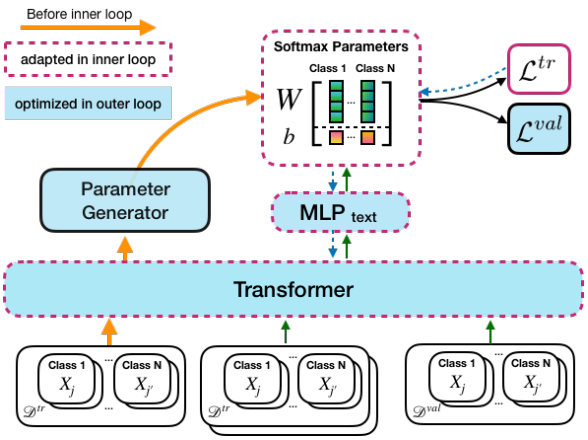

Figure 1: The proposed LEOPARD model. All inputs are first encoded using the Transformer. The first batch from the support set is passed through the parameter generator which learns a per-class set representation that is used to generate the initial softmax parameters. Subsequently, the support batches are used for adaptation of the generated parameters as well as the encoder parameters. Pink box (dashed) outline shows modules that are adapted in the inner loop, whereas blue boxes are optimized in the outer loop.

max parameter generator *conditioned on the meta-training dataset* for an N-way task, which generates the softmax parameters for the task; (3) a MAML-based adaptation method with a distinction between *task-specific parameters*, which are adapted per task, and *task-agnostic* parameters, which are shared across tasks, that can lead to parameter-efficient fine-tuning of large models.

3.1 Text Encoder

The input consists of natural language sentences, thus our models take sequences of words as input. Note that some tasks require classifying pairs of sentences (such as natural language inference) and phrases in a sentence (such as entity typing), and we discuss how these can also be encoded as a sequence of words in Section 4.1.1. Recently, Transformer models (Vaswani et al., 2017) have shown huge success as text encoders in a variety of NLP tasks (Devlin et al., 2018; Radford et al., 2019). Transformer incorporates multiple layers of multi-head self-attention among words in a sentence. This has been shown useful for learning both syntactic and semantic dependencies in different layers (Raganato and Tiedemann, 2018; Tenney et al., 2019), which is useful for various downstream tasks. Thus, we use a Transformer model as our text encoder. Concretely, we follow Devlin et al. (2018) and use their BERT-base model architecture. We denote the Transformer model by f_θ, with parameters $\theta = \{\theta_1, \ldots, \theta_{12}\}$ where θ_v are the parameters of layer v. Transformer takes a sequence of words $\mathbf{x}_j = [x_{j1}, \ldots, x_{js}]$ as input (s being the sequence length), and outputs d-dimensional representation at the final layer of multi-head self-attention. BERT adds a special CLS token (Devlin et al., 2018) to the start of every input, which can be used as a sentence representation. We thus use this as the fixed-dimensional input feature representation of the sentence: $\tilde{x}_j = f_\theta([x_{j1}, \ldots, x_{js}])$.

3.2 Generating Softmax Parameters for Task-specific Classification

Existing applications of MAML model all tasks as a fixed M-way classification. This limits applicability to multiple types of tasks, each of which would require a different number of classes for classification. Moreover, an unseen test task might have a completely different number of classes not seen at all at test time (for example, in our experiments, see Sec. 4). To remedy this, we introduce a method to generate *task-dependent* softmax parameters (both linear weights and bias). Given the training data for a task T_i in an episode, $\mathcal{D}_i^{tr} = \{(x_j, y_j)\}$, we first partition the input into the N_i number of classes for the task (available in $\mathcal{D}_i^{tr}$): $C_i^n = \{x_j | y_j = n\}$, where $n \in [N_i]$. Now, we perform a non-linear projection on the representations of the x_j in each class partition obtained from the text-encoder, and obtain a set representation for the the class m as:

$$w_i^n, b_i^n = \frac{1}{|C_i^n|} \sum_{x_j \in C_i^n} g_\psi(f_\theta(\mathbf{x}_j)) \qquad (3)$$

where g_ψ is multi-layer perceptron (MLP) with $tanh$ non-linearities between intermediate layers, w_i^n is a l-dimensional vector and b_i^n is a scalar. We treat the number of layer in g_ϕ and l as hyper-parameters (details in 4). w_i^n and b_i^n are then treated as the softmax linear weight and bias, respectively, for the class n:

$$\mathbf{W}_i = [w_i^1; \ldots; w_i^{N_i}] \quad \mathbf{b}_i = [b_i^1; \ldots; b_i^{N_i}] \qquad (4)$$

Thus, the softmax classification weights $\mathbf{W}_i \in \mathcal{R}^{N_i \times l}$ and bias $\mathbf{b}_i \in \mathcal{R}^{N_i}$ for task T_i are obtained by row-wise concatenation of the per-class weights in equation 3. Note that encoder $g_\psi(\cdot)$ would be shared across tasks in different episodes.

Now, given the softmax parameters, the prediction for a new data-point $\mathbf{x}^*$ is given as:

$$p(y|\mathbf{x}^*) = \text{softmax}\left\{\mathbf{W}_i h_\phi(f_\theta(\mathbf{x}^*)) + \mathbf{b}_i\right\} \quad (5)$$

where $h_\phi(\cdot)$ is another MLP with parameters ϕ and output dimension l, and the softmax is over the set of classes M_i for the task.

Note that if we use $x^* \in \mathcal{D}_i^{val}$, then the model is a form of a prototypical network (Snell et al., 2017) which uses a learned linear classification layer as opposed to euclidean distance. However, this would limit the model to not adapt it's parameters as more data becomes available for a task. We next discuss how we learn to adapt using the generated softmax parameters. It is important to note that *we do not introduce any task-specific parameters*, unlike multi-task learning (Caruana, 1997), and the existing parameters are used to generate a good starting point for parameters across tasks which can then be *adapted* to task-specific parameters using stochastic gradient based learning.

3.3 Learning to Adapt Efficiently

Given the task-specific classification loss computed at an episode, MAML takes multiple steps of SGD on the same training set $\mathcal{D}_i^{tr}$, as in equation 1. We employ MAML on the model parameters, including the generated softmax parameters, to adapt these parameters to the current task. However, the number of parameters in the BERT transformer architecture is substantially high ($\sim$ 110 million) and sometimes it can be beneficial to adapt a smaller number of parameters (Rusu et al., 2018; Houlsby et al., 2019; Zintgraf et al., 2019b). We thus separate the set of parameters into *task-specific* and *task-agnostic* parameters. For the transformer parameters for each layer $\{\theta_v\}$, we consider a threshold ν over layers, and consider $\theta_{\leq\nu} = \{\theta_v | v \leq \nu\}$ to be the parameters for first ν layers (closest to the input) and the rest of the parameters as $\theta_{>\nu}$. Then we consider these parameters for these initial ν layers of transformer, and the parameters ψ of the softmax generating function 3 as the set of task-agnostic parameters $\Theta = \theta_{\leq\nu} \cup \{\psi\}$. These task-agnostic parameters Θ need to generalize to produce good feature representations and good initial point for classification parameters *across tasks*. The remaining set of parameters for the higher layers of transformer, the input projection function in 5, and the softmax weights and bias generated in equation 4 are considered as the set of task-specific parameters $\Phi_i = \theta_{>\nu} \cup \{\phi, \mathbf{W}_i, \mathbf{b}_i\}$.

The task-specific parameters will be adapted for each task using stochastic gradient descent as in equation 1. Note that MAML usually does gradient descent steps on the same meta-train batch $\mathcal{D}_i^{tr}$ for a task in an episode. However, since we use $\mathcal{D}_i^{tr}$ to generate the softmax parameters in equation 3, using the same data to also take multiple gradient steps can lead to over-fitting. Thus, we instead sample $G > 1$ meta-train batches in each episode of training. Then only the first batch is used to generate the initial softmax parameters and subsequently all batches are used to adapt the softmax as well as other task-specific parameters using SGD. Apart from avoiding over-fitting and introducing stochasticity in the training from the different batches, this strategy further allows the learning to find a better initialization point that is robust to the inherent stochasticity in the optimization procedure that will also be used at test time, thus further minimizing the train and test mismatch. Task-specific adaptation in the inner loop does G steps of the following update, starting with $\Phi_i^{(0)} \leftarrow \Phi_i$, for $s := 0, \ldots, G-1$:

$$\Phi_i^{(s+1)} = \Phi_i^{(s)} - \alpha_s \, \mathbb{E}_{\mathcal{D}_i^{tr} \sim T_i}\left[\nabla_\Phi \mathcal{L}_i(\{\Theta, \Phi\}, \mathcal{D}_i^{tr})\right] \quad (6)$$

Note that we only take gradient with respect to the task-specific parameters Φ_i, however the updated parameter is also a function of Θ. After the G steps of adaptation, the final point (which consists of parameters Θ and Φ^G) is evaluated on the validation set for the task, $\mathcal{D}_i^{val}$, and the task-agnostic parameters Θ are updated (as in equation 2) to adjust the initial point across tasks. Note that optimization of the task-agnostic parameters requires back-propagating through the inner-loop gradient steps and requires computing higher-order gradients. Finn et al. (2017) proposed using a first-order approximation for computational efficiency. We use this approximation in this work, however we note that the distinction between task-specific and task-agnostic parameters can allow for higher order gradients when there are few task-specific parameters (for example, only the last layer).

Other Technical Details: Instead of scalar inner loop learning rates α_s, it has been shown beneficial to have per-parameter learning rates that are also learned (Li et al., 2017). However, this doubles the number of parameters and can be inefficient for large models such those explored here.

Instead, we follow Antoniou et al. (2018) and learn a per-layer, per-step learning rate for the inner loop. We apply layer normalization across layers of transformers (Vaswani et al., 2017; Ba et al., 2016) and also adapt their parameters in the inner loop. The number of layers to consider as task-specific parameters, ν, is also a hyper-parameter. We initialize the meta-training procedure of LEOPARD from pre-trained BERT model.

4 Experiments

Our experiments are focused on evaluating how different methods generalize to new NLP tasks with limited supervision. In this work, we focus on sentence-level classification tasks, including tasks which require classifying pairs of sentences (such as natural language inference) as well as tasks which require classifying a phrase in a sentence (such as entity typing). We consider 16 test datasets for evaluation from the following type of NLP tasks: natural language inference (NLI), sentiment classification, entity typing, relation classification, and various text categorization tasks.

4.1 Experimental Setting

We first discuss the training tasks considered, followed by our evaluation methodology, baselines, set of test tasks and datasets, and other implementation details.

4.1.1 Training tasks

We consider using high resource supervised tasks for training. To this end, we use the training data of the GLUE benchmark tasks[2] (Wang et al., 2018b) for training all the models. Such tasks are considered important for general linguistic intelligence and were created to promote models which go beyond superficial correlations between input and outputs (Wang et al., 2018b). It contains 8 datasets[3] for different kinds of sentence-level classification tasks with varying number of labels across tasks: NLI, sentence similarity, paraphrase detection, duplicate question detection, linguistic acceptability, and sentiment classification. In addition to the GLUE tasks, we also used the SNLI dataset (Bowman et al., 2015) as an additional training task. We use the corresponding validation sets for these tasks to select the best model during training and hyper-parameter tuning.

Data Augmentation: Meta-learning benefits from training across many tasks. Existing methods (Vinyals et al., 2016; Snell et al., 2017; Finn et al., 2017) treat each random sample of labels from a pool of labels (for example in image classification) as a task. In order to create more diversity during training, we also create multiple versions of each dataset that has more than 2 classes, by considering classifying between every possible pair of labels as a separate training task. This increases the number of training tasks for meta-learning and allows for more per-label examples in a batch for tasks with more than 2 classes. Note that this still allows meta-learning methods like prototypical and LEOPARD to fine-tune on a new task with a different number of classes. We train the meta-learning methods with this data augmentation. In addition, since one of the goals is to learn to classify phrases in a sentence, we modify the sentiment classification task (Stanford Sentiment Treebank) in GLUE, which contains annotations of sentiment for phrases, by providing a sentence in which the phrase occurs as part of the input. That is, the input to this task is the sentence followed by a separator token (Devlin et al., 2018) followed by the phrase to classify. Refer to the Appendix for an example. We use this version of the data for all the models to allow them to be fine-tuned for phrase-level classification tasks.

4.1.2 Evaluation and Baselines

We follow standard few-shot evaluation protocol (Vinyals et al., 2016; Snell et al., 2017; Finn et al., 2017). The models are trained on the training tasks (described above), and are then fine-tuned with k training examples per label for a target test task. We evaluate on $k \in \{8, 16, 32\}$[4]. The fine-tuned models are then evaluated on the entire test-set for the task. Since the number of fine-tuning examples are small, model performances can vary greatly based on the set of k examples chosen. Thus, for each task, we sample 10 such datasets for each k and report the mean and standard deviation. Supervised neural models are known to be sensitive to the hyper-parameters used and it is typical to use a validation set in order to select the hyper-parameters. While we use validation during training to select hyper-parameters, the best hyper-

[2] https://gluebenchmark.com

[3] We exclude WNLI from training since it has a very small training data compared to other tasks, and STS-B task since it is a regression task

[4] We choose k in multiples of 2 as higher increments (such as 5) would yield larger data for tasks with high N

parameters during fine-tuning stage can be quite different due to the small size of the training data. Since in the few-shot setting it is unreasonable to assume access to a large validation set, we instead tuned the hyper-parameters for all baselines on a held out validation task. We used SciTail, a scientific NLI dataset, as the held out validation task. For LEOPARD, we only tune the number of epochs for fine-tuning and use meta-SGD based learning rates and reuse remaining hyper-parameters. Refer to A.1 for more details.

We evaluate multiple transfer learning baselines as well as a meta-learning baseline, all based on the state-of-the-art pre-trained BERT architecture. Note that most existing applications of few-shot learning are tailored towards specific tasks and don't trivially apply to diverse tasks considered here. We evaluate the following methods:

$\text{BERT}_{\text{base}}$: We use the cased BERT-base model (Devlin et al., 2018) which is a state-of-the-art transformer (Vaswani et al., 2017) model for NLP. BERT uses self-supervised pre-training followed by supervised fine-tuning on a downstream task. The BERT-base model consists of 12 layers and 768 hidden dimension, with a total of about 110 million parameters. For fine-tuning, we tune all the parameters, in addition to the softmax, as it performed better on the validation task.

Multi-task BERT (MT-BERT): This is the BERT-base model trained in a multi-task learning setting on the set of training tasks. Our MT-BERT is comparable to the MT-DNN model of Liu et al. (2019) that is trained on the tasks considered here and uses the BERT-base as the initialization. We did not use the specialized stochastic answer network for NLI used by MT-DNN. For this model, we tune all the parameters during fine-tuning. Performance of MT-BERT on the validation set of training tasks can be found in Appendix.

$\text{MT-BERT}_{\text{softmax}}$: This is the multi-task BERT model above, where we only tune the softmax layer during fine-tuning.

Prototypical BERT (Proto-BERT): This is the prototypical network method of Snell et al. (2017) that uses the BERT-base model as the underlying neural model. We add a two layer MLP on the sentence representation to get the output embedding for each example. Following Snell et al. (2017), we used euclidean distance as the distance metric. Note that all the methods use the BERT-base architecture and the pre-trained BERT-base model as the initialization point. All parameters of MT-BERT and Proto-BERT are also tuned on the training data. We do not compare with MAML (Finn et al., 2017) as it does not trivially support varying number of classes.

4.1.3 Test Tasks and Datasets

The tasks and datasets we used for evaluating performance on few-shot learning are as follows:

1. Entity Typing: We use the following datasets for entity typing: CoNLL-2003 (Sang and De Meulder, 2003), MIT-Restaurant (Liu et al., 2013), and OpenEntity (Choi et al., 2018). Note that we consider each mention as a separate labelled example.

2. Relation Classification: We consider the SemEval-2010 task 8 dataset (Hendrickx et al., 2009).

3. Sentiment Classification: We use the Amazon Reviews dataset (Blitzer et al., 2007) which contains user reviews for various domains of products. We use the Books, DVD, Electronics, and Kitchen & Housewares domains, which are commonly used domains in the literature (Yu et al., 2018).

4. Text Categorization: We use multiple text categorization datasets from crowdflower[5] and Almeida et al. (2011). These involve classifying sentiments of tweets towards an airline, classifying whether a tweet refers to a disaster event, classifying emotional content of text, classifying the audience/bias/message of social media messages from politicians, and classifying whether a SMS is a spam or not. These tasks are quite different from the training tasks both in terms of the labels as well as the input domain.

5. NLI: We use the SciTail dataset (Khot et al., 2018), which is a dataset for entailment created from science questions.

4.1.4 Implementation Details

Since dataset sizes can be imbalanced, it is important to determine which task to sample in a particular batch, for both multi-task learning as well as meta-learning. Wang et al. (2018a) analyze this in detail for multi-task learning. We explored sampling tasks with a uniform probability

[5] https://www.figure-eight.com/data-for-everyone/

Entity Typing							
	N	k	$\text{BERT}_{\text{base}}$	$\text{MT-BERT}_{\text{softmax}}$	MT-BERT	Proto-BERT	LEOPARD
OpenEntity	7	8	13.93 ± 12.42	15.59 ± 10.92	26.25 ± 15.56	15.94 ± 6.77	**36.71 ± 7.55**
		16	24.57 ± 12.31	23.87 ± 12.16	30.15 ± 12.89	17.26 ± 8.18	**50.47 ± 3.63**
		32	34.31 ± 12.29	31.64 ± 12.71	40.83 ± 10.61	15.17 ± 4.15	**59.88 ± 5.39**
CoNLL	4	8	35.77 ± 08.46	29.67 ± 7.12	34.82 ± 6.52	31.57 ± 7.22	**57.28 ± 5.22**
		16	36.24 ± 11.22	33.55 ± 8.57	46.64 ± 5.01	33.23 ± 4.83	**67.53 ± 1.81**
		32	46.58 ± 15.47	45.32 ± 3.89	56.95 ± 7.47	36.71 ± 2.29	**77.82 ± 3.54**
MITR	8	8	17.82 ± 09.20	14.29 ± 3.71	27.66 ± 7.33	14.85 ± 2.75	**43.66 ± 3.38**
		16	18.05 ± 06.97	21.16 ± 6.76	36.43 ± 6.73	14.26 ± 1.51	**55.39 ± 4.06**
		32	38.27 ± 10.77	27.35 ± 6.36	49.04 ± 6.54	15.33 ± 1.91	**72.05 ± 1.49**

Table 1: Entity Typing results (micro F1) on 3 datasets with varying number of types (N).

Relation Classification							
	N	k	$\text{BERT}_{\text{base}}$	$\text{MT-BERT}_{\text{softmax}}$	MT-BERT	Proto-BERT	LEOPARD
SemEval-2010	10	8	11.76 ± 3.06	10.69 ± 2.84	11.71 ± 2.40	14.04 ± 2.42	**19.56 ± 3.08**
		16	14.20 ± 4.65	10.93 ± 2.53	14.20 ± 3.82	14.79 ± 1.13	**26.12 ± 3.77**
		32	22.26 ± 9.45	15.29 ± 2.61	23.00 ± 4.59	15.81 ± 1.19	**42.19 ± 2.69**

Table 2: Relation Classification results (micro F1) on the SemEval-2010 task 8 dataset with $N = 10$ relations.

and with a probability proportional to the square-root of the size of the task. We treated this as a hyper-parameter and found the latter to be beneficial for all models. We train MT-BERT, Proto-BERT and LEOPARD on 4 GPUs to benefit from larger batches and more tasks per batch. Hyper-parameter search ranges as well as best hyper-parameters can be found in A.1. Additional details about dataset statistics can be found in A.2.

4.2 Results

In this section we evaluate and analyze our model and the baselines on the aforementioned datasets. On an average, we observe *absolute gains* in accuracy of 4.62% on domain transfer, 4.52% on text categorization and 17.39% in F1 on entity typing and relation classification across low to moderate data regimes which confirms the efficacy of LEOPARD for a wide range of tasks. Further, in section 4.3 we perform ablation studies to better understand the performance of LEOPARD.

Entity Typing & Relation Classification Results for entity typing and relation classification are reported in Table 1 and 2 respectively. We can see that LEOPARD significantly outperforms all the baselines on these tasks. Note that these tasks were not seen while training any of the models. The results clearly show the importance of the meta-learning training procedure combined with task-specific classification layer for generalizing to new tasks with few labelled examples. We also notice that prototypical networks performs significantly worse on new training tasks. We hypothesize that this is because prototypical networks do not generate good class prototypes for new tasks and adaptation of class prototypes is important for improving performance. We also see that improved feature learning in MT-BERT with additional training tasks serves as a better initialization point for such held-out tasks than $\text{BERT}_{\text{base}}$. Fine-tuning just the softmax layer is slightly worse, as opposed fine-tuning entire MT-BERT, even in these data deficient regimes.

Text Categorization Table 3 summarizes the results on the social-media based text categorization tasks. LEOPARD outperforms the baselines which shows its robustness to varying number of labels and different text domains. We observe that even though LEOPARD uses the same training tasks as MT-BERT it can adapt quicker (i.e. with fewer examples) to new text domains. Some classification tasks such as classifying political bias and audiences prove to be quite difficult for all models in the few-shot setting.

Domain Transfer on Sentiment Classification & NLI We also evaluate the models on how they perform on new domains of tasks seen at training time. For this, we consider two tasks of Sentiment Classification and NLI. We introduce another relevant baseline here, $\text{MT-BERT}_{\text{reuse}}$, which reuses the trained softmax parameters of a related task at training. Results are summarized in Table 4.

Text Categorization							
	N	k	$\text{BERT}_{\text{base}}$	$\text{MT-BERT}_{\text{softmax}}$	MT-BERT	Proto-BERT	LEOPARD
Airline	3	8	34.45 ± 19.66	32.48 ± 12.10	33.86 ± 18.51	45.26 ± 8.51	**46.95 ± 10.23**
		16	31.93 ± 16.08	33.18 ± 15.21	49.37 ± 15.88	49.70 ± 10.90	**57.53 ± 04.64**
		32	33.99 ± 18.44	35.67 ± 18.58	42.04 ± 15.51	50.15 ± 10.53	**61.36 ± 03.58**
Disaster	2	8	54.34 ± 5.18	50.28 ± 4.67	51.21 ± 6.13	49.72 ± 1.52	**55.45 ± 2.85**
		16	53.23 ± 7.50	52.45 ± 5.60	51.95 ± 5.34	49.69 ± 0.67	**58.93 ± 4.80**
		32	52.41 ± 7.75	51.01 ± 5.90	52.37 ± 6.55	50.36 ± 3.28	**58.86 ± 3.19**
Emotion	13	8	05.48 ± 5.95	08.84 ± 4.4	07.22 ± 3.30	08.12 ± 3.95	**10.71 ± 2.61**
		16	10.43 ± 6.21	10.48 ± 2.57	06.81 ± 5.66	08.40 ± 2.79	**13.07 ± 1.97**
		32	07.64 ± 5.88	10.10 ± 3.77	09.29 ± 3.72	10.40 ± 3.33	**15.93 ± 1.15**
Political Message	9	8	06.16 ± 3.67	09.12 ± 7.39	09.88 ± 6.20	10.03 ± 3.73	**10.11 ± 4.14**
		16	10.44 ± 5.91	11.61 ± 6.66	08.84 ± 6.53	09.86 ± 5.66	**15.41 ± 6.33**
		32	09.37 ± 6.84	10.26 ± 5.50	10.81 ± 6.51	09.93 ± 4.96	**16.79 ± 6.67**
Political Bias	2	8	50.31 ± 0.92	49.71 ± 1.30	50.06 ± 0.86	50.66 ± 1.28	**51.63 ± 1.90**
		16	50.77 ± 1.52	50.99 ± 1.24	51.09 ± 2.29	50.91 ± 1.26	**51.93 ± 1.56**
		32	50.74 ± 2.08	49.86 ± 0.61	50.63 ± 1.70	50.08 ± 1.22	**51.23 ± 2.60**
Political Audience	2	8	50.38 ± 2.43	50.78 ± 1.99	49.57 ± 1.81	51.67 ± 1.91	**53.50 ± 2.61**
		16	50.16 ± 1.48	49.87 ± 1.49	50.13 ± 1.64	50.44 ± 2.03	**52.02 ± 2.52**
		32	51.28 ± 1.83	51.22 ± 2.28	51.52 ± 2.13	50.61 ± 2.31	**54.46 ± 2.55**
SMS Spam	2	8	70.31 ± 14.71	57.77 ± 08.13	68.95 ± 14.17	59.62 ± 7.32	**86.80 ± 4.27**
		16	79.86 ± 14.09	59.22 ± 10.75	74.02 ± 12.20	62.73 ± 7.39	**91.46 ± 1.91**
		32	90.13 ± 05.96	65.10 ± 10.64	84.50 ± 06.48	63.91 ± 1.91	**94.64 ± 0.90**

Table 3: Text Categorization accuracy on various social-media datasets.

First, we look at sentiment classification on 4 different domains of product reviews. This task is closely related to one of the tasks in the training phase: Stanford Sentiment Treebank (SST). Note that SST only contains phrase level sentiments and the models weren't trained to predict sentence-level sentiment, while the target tasks require sentence-level sentiment. Again, we observe that LEOPARD performs significantly better than the baselines. We also observe that reusing softmax parameters from SST serves as a good initialization for classification layer of MT-BERT further motivating generation of task dependent initial softmax parameters for tasks not seen during training. We also note that the performance of prototypical networks is significantly better for these tasks in comparison to that in Tables 3, 1 and 2 as it has learned to generate prototypes for a similar task during the training phase. Next, we look at NLI task on a new domain of scientific questions (SciTail). As the training procedure is focused towards NLI, with several large dataset as part of the training phase, the baselines perform significantly better. Prototypical performs best for small k, again as it has learned to generate prototypes for a the same task during the training phase. For larger $k = 32$, reusing softmax parameters from the related RTE task at train gets even better performance. While LEOPARD does not perform as well as Proto-BERT, we note that it performs significantly better than other fine-tuning baselines where the softmax parameters aren't reused. Also note that it is not always possible to have a large set of closely related dataset at hand for a target task. In our ablation study we also demonstrate that parameter efficient versions of our model have a performance comparable on such closely related tasks as well.

4.3 Ablation Study

In this section we analyze two important components of LEOPARD, that are parameter efficient training and effect of training tasks. For this analysis we focus on 4 tasks: the CoNLL-2003 entity typing task, SemEval 2010 relation classification task, Amazon reviews DVD domain sentiment classification task and SciTail NLI task.

4.3.1 Parameter Efficiency

We evaluate LEOPARD with parameter efficient training discussed in 3.3. We consider three variants: (1) LEOPARD_{10} where we take $\nu = 10$ and do not adapt the layers 0 to 10 including the word embeddings in the inner loop of the meta-training; (2) LEOPARD_{5} where we take $\nu = 5$;

	k	$\text{BERT}_{\text{base}}$	$\text{MT-BERT}_{\text{softmax}}$	MT-BERT	$\text{MT-BERT}_{\text{reuse}}$	Proto-BERT	LEOPARD
Natural Language Inference							
Scitail	8	59.38 ± 11.63	59.19 ± 18.18	61.16 ± 12.51	77.37 ± 2.08	**78.17 ± 1.13**	71.71 ± 1.45
	16	59.85 ± 08.77	63.88 ± 13.95	69.24 ± 11.29	78.39 ± 1.60	**78.69 ± 0.46**	73.87 ± 1.1
	32	65.18 ± 06.09	75.02 ± 09.56	63.28 ± 17.48	**79.27 ± 1.36**	78.84 ± 0.13	75.81 ± 4.16
Amazon Review Sentiment Classification							
Books	8	51.40 ± 2.17	53.78 ± 7.89	51.51 ± 03.75	67.91 ± 1.22	71.01 ± 10.71	**73.39 ± 7.15**
	16	52.63 ± 6.80	57.93 ± 6.38	65.12 ± 08.22	67.94 ± 0.12	71.25 ± 05.67	**77.50 ± 1.73**
	32	54.17 ± 6.27	59.20 ± 6.71	61.97 ± 10.02	68.65 ± 0.27	78.13 ± 02.29	**78.28 ± 1.59**
DVD	8	50.37 ± 0.70	54.99 ± 5.28	52.54 ± 6.20	66.74 ± 0.54	73.37 ± 1.75	**76.65 ± 2.44**
	16	50.32 ± 0.68	55.47 ± 6.27	54.70 ± 6.17	66.81 ± 0.11	72.38 ± 1.91	**76.86 ± 1.68**
	32	52.26 ± 3.05	57.54 ± 6.62	58.36 ± 8.82	70.11 ± 6.07	74.92 ± 1.12	**79.04 ± 1.17**
Electronics	8	51.94 ± 2.30	50.91 ± 5.63	52.27 ± 4.01	64.84 ± 1.10	57.00 ± 5.34	**72.22 ± 8.51**
	16	52.97 ± 4.38	51.76 ± 4.30	60.53 ± 6.65	65.18 ± 0.74	63.60 ± 8.10	**76.95 ± 1.99**
	32	54.97 ± 8.08	55.70 ± 5.51	57.66 ± 7.72	64.59 ± 8.33	67.42 ± 6.93	**77.46 ± 2.37**
Kitchen	8	52.29 ± 2.94	53.29 ± 4.39	56.13 ± 7.66	71.02 ± 8.87	62.40 ± 6.60	**78.71 ± 1.91**
	16	53.68 ± 5.06	51.96 ± 3.11	60.57 ± 9.71	67.51 ± 0.87	64.89 ± 6.44	**80.32 ± 2.04**
	32	51.78 ± 2.82	57.10 ± 6.63	64.82 ± 8.41	66.71 ± 7.04	67.02 ± 5.77	**81.00 ± 1.06**

Table 4: Domain transfer evaluation (accuracy) on NLI and Sentiment classification datasets.

k	Model	Relation Classification	Entity Typing	Sentiment Classification	NLI
8	LEOPARD_{10}	11.77 ± 4.98	39.59 ± 7.27	59.00 ± 5.63	75.05 ± 05.98
	LEOPARD_{5}	15.12 ± 4.74	56.12 ± 3.46	70.70 ± 3.78	67.47 ± 11.09
	LEOPARD	16.86 ± 2.45	59.71 ± 3.90	74.85 ± 4.75	73.20 ± 04.65
16	LEOPARD_{10}	10.72 ± 2.02	37.62 ± 7.37	58.10 ± 5.40	78.53 ± 1.55
	LEOPARD_{5}	17.84 ± 2.45	62.49 ± 4.23	71.50 ± 5.93	73.27 ± 2.63
	LEOPARD	22.69 ± 2.62	69.00 ± 4.76	76.65 ± 2.47	76.10 ± 2.21
32	LEOPARD_{10}	14.98 ± 2.93	44.58 ± 4.74	69.10 ± 8.32	78.06 ± 2.92
	LEOPARD_{5}	21.68 ± 2.40	68.96 ± 3.71	70.25 ± 5.05	75.97 ± 1.21
	LEOPARD	34.88 ± 2.79	79.89 ± 1.19	79.00 ± 2.00	78.98 ± 3.65

Table 5: Analyzing parameter efficiency: LEOPARD_{10} does not adapt layers 0-10 (inclusive) and LEOPARD_{5} does not adapt layers 0-5 in the inner loop, while LEOPARD adapts all parameters. Note that the outer loop during meta-training still optimizes all parameters for all the three models. For new tasks (like entity typing) adapting all parameters is beneficial while for tasks seen at training time (like NLI) adapting fewer parameters is better.

(3) LEOPARD where we adapt all parameters in the inner loop. Note that even for $\nu \neq 0$, the parameters are optimized in the outer loop to learn a better initialization point for few-shot learning. Table 5 shows the results on the dev-set of the 4 tasks considered. Interestingly, for all tasks except NLI we find that adapting all parameters in the inner loop is more efficient. Even with 8 examples per label, adapting all parameters can give significantly better results than only adapting the final layer and with more examples the difference becomes even more pronounced. On SciTail (NLI) we observe the opposite behaviour, suggesting that adapting fewer parameters is better for lesser examples. This is potentially because training tasks consisted of multiple datasets for NLI.

4.3.2 Performance based on training tasks

We analyze how the performance of MT-BERT and LEOPARD on target tasks is dependent on tasks used for training. For this experiment, we held out each training task one by one and trained both LEOPARD as well as MT-BERT. The trained models are then evaluated for their performance on the target tasks (using the development set), following the same protocol as before. Fig. 2 shows a visualization of the relative change in performance when each training task is held out. See Table 10 for the actual performance numbers. Since LEOPARD follows optimization procedure similar to MAML we expect it to converge to initial parameters which perform equally well across tasks i.e the initial parameters are *equidistant* to optimal parameters for a wide range of tasks. Fig. 2 confirms this intuition, as we see consistency in performance across target tasks, even when one training task is held out. When QNLI and SST are held out we see that LEOPARD consistently under-performs across all target tasks, while for

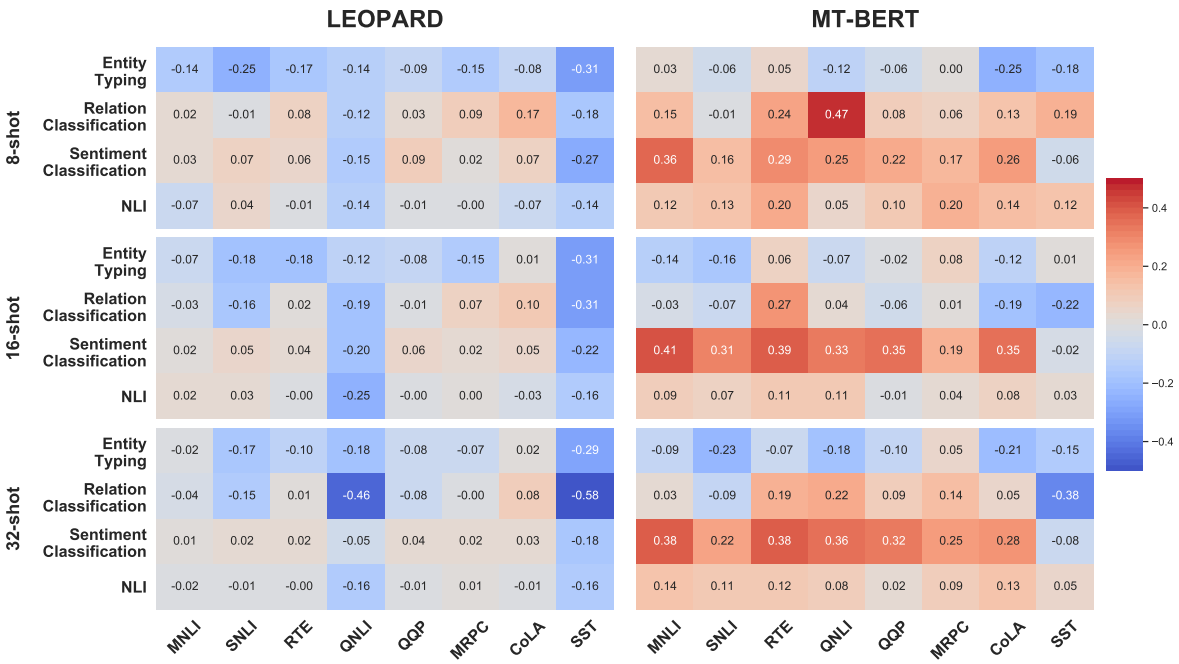

Figure 2: Analyzing target task performance as a function of training tasks (best viewed in color). Heatmaps on the left are for LEOPARD and on the right are for MT-BERT. Each column represents one held-out training task (name on x-axis) and each row corresponds to one target task (name on y-axis). Each cell is the relative change in performance on the target task when the corresponding training task is held-out, compared to training on all the train tasks. Dark blue indicates large drop, dark red indicates large increase and grey indicates close to no change in performance. In general, LEOPARD's performance is more consistent compared to MT-BERT indicating that meta-training learns more generalized initial parameters compared to multi-task training.

others its performance is consist. In contrast, MT-BERT's performance on target tasks varies greatly depending on the held-in training tasks. We also observe that removing CoLA benefits almost all the tasks for LEOPARD. This is likely because acceptability is a tough problem to learn in few-shots and forcing the model to train to perform this task hurts performance, while supervised learning is not constrained to obtain good performance in few-shots. Removing SST hurts performance on all tasks, potentially due to its large size.

5 Related Work

5.1 Meta-Learning

Meta-Learning approaches can be broadly classified into three categories: optimization-based, model-based and based on metric-learning. Optimization methods aim to modify the gradient descent based learning procedure such that quick adaptation to new task is possible. As discussed earlier, Finn et al. (2017) explicitly pose learning a good initialization of parameters as a optimization problem that can be solved using SGD. Al-Shedivat et al. (2018) studied continuous adaptation in a reinforcement learning setup using MAML. In contrast, Ravi and Larochelle (2017) learn a good initialization of parameters as well as an optimizer(Meta-Learner) which is modelled using a LSTM. Meta-SGD (Li et al., 2017) learn step sizes and update direction in a training procedure similar to MAML. Recently it has been shown that learning a set of task dependent model parameters improves performance for few-shot learning (Zintgraf et al., 2019a; Rusu et al., 2019). For example, Latent Embedding Optimization (LEO) (Rusu et al., 2019) learns a distribution of parameters conditioned on task instead of finding a single initialization point like in MAML. LEO generates high dimensional parameters but optimizes in low dimensional latent space of parameters so as to avoid difficulties in optimizing with few examples. In comparison we optimize directly in the generated parameter space since we only generate task specific parameters while preserving task-agnostic parameters. This allows us to use rich features representations in Transformer-like models along

with generating initial task specific parameters. Also, for the model considered in our method it is non-trivial to generate parameters while optimizing in low dimensional space. Model based learning typically involves specifically designing models for fast adaptation (Munkhdalai and Yu, 2017). Metric-learning approaches learn a metric function in order to compute similarity between support and query inputs in the learnt embedding space. Typically these methods are evaluated only for few-shot image classification with limited variance in task. In parallel to our work, Triantafillou et al. (2019) introduced a more diverse dataset for few-shot image classification and evaluated a suite of meta-learning algorithms.

5.2 Few-shot learning in NLP

The problem of few-shot learning is important in NLP as limited data is available for important sentence-level tasks, such as Machine Translation in low resource language pairs. Similarly, for entity level tasks such as Relation Classification, Named Entity Recognition, Entity Typing etc, data per entity can be very scarce. Traditional approach involves multi-task learning to learn a good feature representation and then fine-tune on the task at hand. These approaches are limited because there is a mismatch between train and test time. Further, if a test task is different from training tasks, then sharing classification layer becomes infeasible. Recently, meta-learning approaches have gained widespread popularity for few-shot learning. Gu et al. (2018) used MAML to simulate low resource machine translation by sampling from high resource language pairs, treating each language pair as a new task. Chen et al. (2018) learn HyperLSTM (Ha et al., 2016) model in a multi-task setting across various sentiment classification domains and show improvements when transferring to a new domain. Other approaches (Han et al., 2018; Obamuyide and Vlachos, 2019; Geng et al., 2019; Mi et al., 2019) train for a specific classification task and do not generalize beyond the training task itself. Metric based meta-learning is widely popular in NLP. Guo et al. (2018) learn a distance metric between example of a query domain with a set of elements from support domains, specifically for domain adaptation. Yu et al. (2018) learn multiple metrics to handle few-shot classification. Their meta-training procedure involves first clustering source tasks followed by learning a distance metric per cluster. Geng et al. (2019) employs capsules and dynamic routing algorithm (Sabour et al., 2017) for generating class prototypes in a framework similar to prototypical networks. More recent applications of meta-learning in NLP

Transfer Learning is a closely related research area. Recently, self-supervised pre-training using language modeling objectives has shown significant progress towards learning general purpose model parameters that improve downstream performance with fine-tuning (Peters et al., 2018; Howard and Ruder, 2018; Devlin et al., 2018; Radford et al., 2019; Yang et al., 2019). Fine-tuning, however, typically requires more training data. Peters et al. (2019) studied best practices for fine-tuning pre-trained models. Yogatama et al. (2019) studied general linguistic intelligence from pre-trained models and explored how such models generalize on unseen tasks. They demonstrated that BERT requires more training data than BERT trained on other supervised tasks, for generalizing to some target tasks such as QA. Phang et al. (2018) showed that training BERT on intermediate tasks such as NLI can improve performance for some other related tasks. Wang et al. (2018a) rigorously analyzed advantages of pre-training followed by intermediate training, as well as multi-task training and found multi-task training to be more beneficial for transfer. Liu et al. (2019) did multi-task training of BERT and showed that it can adapt to other similar NLI tasks. Our MT-BERT baseline is closely related to this. We refer the reader to Ruder (2019) for a more thorough discussion of transfer learning. Our work significantly advances these efforts on developing general linguistic intelligence by exploring a wide range of target tasks, in a few-shot setting.

6 Conclusions

Learning general linguistic intelligence has been a long-term goal of natural language processing. While humans, with all their prior knowledge, can quickly learn to solve new tasks with very few examples, machine-learned models still struggle to demonstrate such intelligence. Though there has been consistent progress in improving performance on many NLP tasks, we evaluated that atleast for few-shot learning, these improvements can be limited. To this end, we proposed LEOPARD, a meta-learning approach, and found that meta-training learns more general purpose parameters that better prime the model to solve com-

pletely new tasks with few examples, as compared to supervised multi-task training. While we see improvements using meta-learning, performance with few examples still lags significantly behind peak performance with large training data or human-level performance. We consider bridging this gap as a lucrative goal for the community to demonstrate true linguistic intelligence.

References

Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. 2018. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *Proceedings of the International Conference on Learning Representations*.

Tiago A Almeida, José María G Hidalgo, and Akebo Yamakami. 2011. Contributions to the study of sms spam filtering: new collection and results. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 259–262. ACM.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. 2018. How to train your maml. *arXiv preprint arXiv:1810.09502*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. 1992. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Junkun Chen, Xipeng Qiu, Pengfei Liu, and Xuanjing Huang. 2018. Meta multi-task learning for sequence modeling. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. *arXiv preprint arXiv:1807.04905*.

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*.

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 - Volume 70*, pages 1126–1135.

Ruiying Geng, Binhua Li, Yongbin Li, Xiaodan Zhu, Ping Jian, and Jian Sun. 2019. Induction networks for few-shot text classification.

Jiatao Gu, Yong Wang, Yun Chen, Kyunghyun Cho, and Victor OK Li. 2018. Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437*.

Jiang Guo, Darsh Shah, and Regina Barzilay. 2018. Multi-source domain adaptation with mixture of experts. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4694–4703.

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106*.

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.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2009. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, pages 94–99. Association for Computational Linguistics.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. Scitail: A textual entailment dataset from science question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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*.

Jingjing Liu, Panupong Pasupat, Scott Cyphers, and Jim Glass. 2013. Asgard: A portable architecture for multilingual dialogue systems. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8386–8390. IEEE.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Fei Mi, Minlie Huang, Jiyong Zhang, and Boi Faltings. 2019. Meta-learning for low-resource natural language generation in task-oriented dialogue systems. *arXiv preprint arXiv:1905.05644*.

Tsendsuren Munkhdalai and Hong Yu. 2017. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2554–2563. JMLR. org.

Abiola Obamuyide and Andreas Vlachos. 2019. Model-agnostic meta-learning for relation classification with limited supervision. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5873–5879.

Matthew Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv:1903.05987*.

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.

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Alessandro Raganato and Jörg Tiedemann. 2018. An analysis of encoder representations in transformer-based machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 287–297.

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

Sebastian Ruder. 2019. *Neural Transfer Learning for Natural Language Processing*. Ph.D. thesis, NATIONAL UNIVERSITY OF IRELAND, GALWAY.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. 2018. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*.

Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. 2019. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866.

Erik F Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Jürgen Schmidhuber. 1987. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Ph.D. thesis, Technische Universität München.

Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Sebastian Thrun and Lorien Pratt. 2012. *Learning to learn*. Springer Science & Business Media.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R Thomas McCoy, Roma Patel, Najoung Kim, Ian Tenney, Yinghui Huang, Katherin Yu, et al. 2018a. Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. *arXiv preprint arXiv:1812.10860*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018b. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomás Kociský, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, and Phil Blunsom. 2019. Learning and evaluating general linguistic intelligence. *CoRR*, abs/1901.11373.

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. *arXiv preprint arXiv:1805.07513*.

Luisa Zintgraf, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019a. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702.

Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. 2019b. Cavia: Fast context adaptation via meta-learning. In *International Conference on Machine Learning*.

A Appendix

Table 6 shows the dev-set accuracy of the MT-BERT model on the various training tasks.

A.1 Hyperparameters

Table 7 shows the hyper-parameter search range as well as the best hyper-parameters for MT-BERT, Proto-BERT and LEOPARD. We use same hyperparameters for prototypical networks except those not relevant to them. For fine-tuning we separately tune number of iterations, final layer dropout and batch size for each k shot for all the baselines. For MT-BERT we found 5 epochs , batch size 4 and dropout 0.3 to be best for 8-shot,2 epochs, batch size 8 and dropout 0.2 to be best for 16-shot and 1 epoch with 16 batch size and 0.2 dropout gave the best performance for 32 shot. For $\text{MT-BERT}_{\text{softmax}}$ we found 1 epoch, batch size 4 and dropout 0.0 to be best for 8-shot, we found 2 epochs, batch size 16 and dropout 0.0 to be best for 16-shot and 5 epochs with batch size 16 and dropout 0.1 gave the best performance for 32-shot. For $\text{BERT}_{\text{base}}$ 1 epochs, batch size 16, and dropout 0.0 for 8 shot, 1 epochs, 8 batch size, 0.2 dropout for 16 shot and 5 epochs, batch size 16 and dropout 0.0 for 32 shot gave the best performance. For $\text{MT-BERT}_{\text{reuse}}$ 1 epochs, batch size 4 and dropout 0.3 for 8-shot and 2 epochs, batch size 16 and dropout 0.1 for 16 shot and 5 epochs, batch size 16 and 0.0 dropout for 32 shot. Note, for LEOPARD we use Meta-SGD for training and found it beneficial to run for larger epochs. We find using 100 epochs gave best performance for 8 and 16 shot while 200 epochs for 32 shot was the best. As we learn per step SGD, we found it beneficial to use each steps learning rate evenly across epochs.

A.2 Datasets

We use the standard train, dev data split for GLUE and SNLI (Wang et al., 2018b; Bowman et al., 2015). For our ablation studies, on our target task we take 20% of the training data as dev set and sample from the remaining 80% to create the data for fine-tuning. For training MT-BERT we use dev data of the training task as the validation set. For meta-learning methods, prototypical network and LEOPARD we use additional validation datasets as is typical in meta learning methods (Finn et al., 2017; Snell et al., 2017). We use unlabelled Amazon review data from apparel, health, software, toys, video as categorization tasks and labelled data from music, toys, video as sentiment classification task. Details of the datasets are present in Table 8. For OpenEntity dataset, we use the high-level types, remove instances with multiple labels and those labels with fewer than 32 examples, details of actual data can be found in Choi et al. (2018).

An example of the input to all the models for the entity typing tasks can be found in Table 9

A.3 Importance of training tasks

Refer to Table 10 for actual numbers used for ablation study in Figure 2.

	MNLI(m/mm)	QQP	QNLI	SST-2	CoLA	MRPC	RTE	SNLI	Average
MT-BERT	82.11	89.92	89.62	90.7	81.30	84.56	78.34	89.97	**85.82**

Table 6: Dev-set accuracy on the set of train tasks for multi-task BERT.

Parameter	Search Space	MT-BERT	Proto-BERT	LEOPARD
Attention dropout	[0.1, 0.2, 0.3]	0.2	0.3	0.3
Batch Size	[16, 32]	32	16	16
Class Embedding Size	[128, 256]	—	256	256
Hidden Layer Dropout	[0.1, 0.2, 0.3]	0.1	0.2	0.2
Inner Loop Learning Rate	—	—	—	Meta-SGD (per-layer, per-step)
Min Adapted Layer (ν)	[0, 5, 8, 10, 11]	—	—	0
Outer Loop Learning Rate	[1e-4, 1e-5, 2e-5, 4e-5, 5e-5]	2e-05	2e-05	5e-05
Adaptation Steps (G)	[5]	—	—	5
Top layer [CLS] dropout	[0.45, 0.4, 0.3, 0.2, 0.1]	0.1	0.2	0.1
Train Word Embeddings(Inner Loop)	[True, False]	—	—	True
Data Sampling	[Square Root, Uniform]	Square Root	Square Root	Square Root
Lowercase text	False	False	False	False

Table 7: Hyper-parameter search space and best hyper-parameters for all models.

Dataset	Labels	Training Size	Validation Size	Testing Size	Source
ARSC Domains	2	800	200	1000	(Blitzer et al., 2007)
CoLA	2	8551	1042	—	(Wang et al., 2018b)
MRPC	2	3669	409	—	(Wang et al., 2018b)
QNLI	2	104744	5464	—	(Wang et al., 2018b)
QQP	2	363847	40431	—	(Wang et al., 2018b)
RTE	2	2491	278	—	(Wang et al., 2018b)
SNLI	3	549368	9843	—	(Bowman et al., 2015)
SST-2	2	67350	873	—	(Wang et al., 2018b)
MNLI (m/mm)	3	392703	19649	—	(Wang et al., 2018b)
Scitail	2	23,596	1,304	2,126	(Khot et al., 2018)
Airline	3	7320	—	7320	https://www.figure-eight.com/data-for-everyone/
Disaster	2	4887	—	4887	https://www.figure-eight.com/data-for-everyone/
Political Bias	2	2500	—	2500	https://www.figure-eight.com/data-for-everyone/
Political Audience	2	2500	—	2500	https://www.figure-eight.com/data-for-everyone/
Political Message	9	2500	—	2500	https://www.figure-eight.com/data-for-everyone/
Emotion	13	20000	—	20000	https://www.figure-eight.com/data-for-everyone/
CoNLL	4	23499	5942	5648	(Sang and De Meulder, 2003)
SemEval-2010 Task 8	10	8000	—	2717	(Hendrickx et al., 2009)
MIT-Restaurant	8	12474	—	2591	https://groups.csail.mit.edu/sls/downloads/restaurant/
OpenEntity*	7	1362	—	1287	(Choi et al., 2018)

Table 8: Dataset statistics for all the datasets used in our analysis. "-" represent data that is either not available or not used in this study. We have balanced severely unbalanced datasets(Political Bias, Audience and Message) as our training data is balanced. To create training data for few shot experiments we sample 10 datasets for each k-shot. *Sec A.2 for more details

Input	Label
are there any [authentic mexican]^1 restaurants in [the area]^2	^{1}Cuisine, ^{2}Location
are there any authentic mexican restaurants in the area [SEP] authentic mexican	Cuisine
are there any authentic mexican restaurants in the area [SEP] the area	Location

Table 9: An example of an input from the MIT restaurants dataset. The first line is the actual example with two mentions. The next two lines are the input to the models – one for each mention.

Natural Language Inference									
		CoNLL		Relation Classification		Sentiment Classification		SciTail	
Heldout	k	MT-BERT	LEOPARD	MT-BERT	LEOPARD	MT-BERT	LEOPARD	MT-BERT	LEOPARD
—	8	35.54 ± 8.88	59.71 ± 3.9	10.63 ± 3.06	16.86 ± 2.45	55.0 ± 8.81	74.85 ± 4.75	57.22 ± 14.29	73.2 ± 4.6
	16	45.0 ± 10.56	69.0 ± 4.76	14.67 ± 3.67	22.69 ± 2.62	54.55 ± 6.97	76.65 ± 2.47	69.64 ± 9.51	76.1 ± 2.2
	32	58.06 ± 6.4	79.89 ± 1.19	19.04 ± 4.37	34.88 ± 2.79	57.25 ± 10.09	79.0 ± 2.0	69.02 ± 13.07	78.98 ± 3.6
MNLI	8-	36.57 ± 14.32	51.28 ± 6.42	12.23 ± 3.68	17.25 ± 2.34	74.95 ± 6.76	77.45 ± 6.62	64.01 ± 10.16	68.42 ± 11.9
	16	38.87 ± 3.35	63.85 ± 4.67	14.19 ± 3.89	22.0 ± 2.44	76.7 ± 8.3	77.9 ± 4.72	76.01 ± 2.44	77.89 ± 1.8
	32	52.99 ± 8.73	78.37 ± 2.82	19.55 ± 6.42	33.65 ± 3.02	79.2 ± 6.5	79.85 ± 1.7	78.49 ± 2.57	77.63 ± 10.2
SNLI	8	33.53 ± 7.34	44.91 ± 7.04	10.54 ± 4.44	16.63 ± 2.39	63.7 ± 11.19	80.3 ± 3.05	64.46 ± 11.3	76.38 ± 2.7
	16	37.74 ± 8.05	56.25 ± 3.44	13.7 ± 3.67	18.96 ± 2.19	71.55 ± 9.32	80.1 ± 1.62	74.54 ± 3.37	78.01 ± 1.2
	32	44.92 ± 10.35	65.95 ± 5.7	17.29 ± 2.87	29.79 ± 2.23	70.05 ± 10.17	80.75 ± 5.42	76.89 ± 1.9	77.99 ± 1.2
RTE	8	37.39 ± 6.69	49.55 ± 5.29	13.18 ± 5.08	18.13 ± 2.59	71.1 ± 11.56	79.25 ± 1.97	68.73 ± 12.86	72.6 ± 3.7
	16	47.83 ± 11.43	56.54 ± 5.22	18.64 ± 3.24	23.05 ± 2.01	75.95 ± 3.25	79.45 ± 2.27	77.38 ± 3.43	75.88 ± 1.3
	32	54.25 ± 7.56	71.65 ± 5.14	22.71 ± 6.48	35.37 ± 2.7	79.15 ± 5.19	80.8 ± 2.32	76.98 ± 5.89	78.69 ± 1.5
QNLI	8	31.38 ± 7.33	51.53 ± 5.92	15.67 ± 3.61	14.8 ± 3.35	68.65 ± 10.19	63.5 ± 12.19	60.19 ± 13.07	63.28 ± 9.3
	16	42.01 ± 9.18	60.85 ± 6.29	15.21 ± 3.42	18.32 ± 2.66	72.45 ± 10.03	61.7 ± 9.45	77.34 ± 2.67	56.75 ± 9.8
	32	47.58 ± 10.69	65.39 ± 11.87	23.2 ± 3.11	18.86 ± 5.6	77.75 ± 4.46	75.35 ± 3.01	74.74 ± 5.46	66.64 ± 8.0
Semantic similarity and Paraphrase									
QQP	8	33.5 ± 4.77	54.58 ± 3.62	11.49 ± 3.15	17.44 ± 2.68	66.85 ± 13.5	81.55 ± 2.53	63.18 ± 9.48	72.47 ± 3.9
	16	44.3 ± 12.47	63.48 ± 6.13	13.84 ± 4.69	22.54 ± 2.26	73.65 ± 5.26	81.05 ± 1.94	68.79 ± 3.04	75.75 ± 1.2
	32	52.47 ± 5.43	73.89 ± 4.23	20.72 ± 4.71	32.22 ± 1.78	75.65 ± 4.78	81.8 ± 1.69	70.59 ± 3.09	78.24 ± 1.8
MRPC	8	35.68 ± 6.42	50.92 ± 6.15	11.29 ± 5.73	18.39 ± 2.97	64.2 ± 12.28	76.05 ± 5.08	68.51 ± 7.53	72.93 ± 3.6
	16	48.54 ± 12.32	58.78 ± 6.34	14.77 ± 5.36	24.35 ± 1.76	64.85 ± 10.58	78.55 ± 1.99	72.41 ± 5.61	76.41 ± 2.1
	32	60.84 ± 13.48	74.55 ± 4.39	21.8 ± 4.69	34.72 ± 1.65	71.5 ± 9.52	80.85 ± 1.43	75.49 ± 5.27	79.9 ± 1.8
Single Sentence Classification									
CoLA	8	26.73 ± 6.4	54.86 ± 5.99	12.01 ± 3.23	19.75 ± 2.27	69.15 ± 10.29	80.35 ± 3.15	65.34 ± 11.47	68.25 ± 6.1
	16	39.78 ± 11.35	69.88 ± 3.9	11.91 ± 4.45	25.05 ± 2.03	73.85 ± 6.53	80.35 ± 2.28	75.09 ± 4.43	74.1 ± 2.0
	32	46.14 ± 11.69	81.13 ± 2.23	20.08 ± 2.84	37.8 ± 2.59	73.0 ± 8.38	81.75 ± 1.89	77.7 ± 4.5	77.96 ± 2.3
SST	8	28.99 ± 7.76	41.33 ± 7.45	12.6 ± 3.89	13.86 ± 3.79	51.7 ± 3.42	54.6 ± 4.82	64.12 ± 11.84	63.24 ± 10.3
	16	45.5 ± 6.58	47.66 ± 10.71	11.42 ± 2.89	15.68 ± 2.73	53.4 ± 6.71	59.45 ± 5.9	71.68 ± 2.76	64.23 ± 9.9
	32	49.52 ± 8.65	56.48 ± 11.77	11.82 ± 4.31	14.77 ± 4.51	52.8 ± 3.03	64.9 ± 5.86	72.32 ± 2.61	66.61 ± 5.

Table 10: Analyzing target task performance as a function of training tasks.