Grad2Task: Improved Few-shot Text Classification Using Gradients for Task Representation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large pretrained language models (LMs) like BERT have improved performance in many disparate natural language processing (NLP) tasks. However, fine tuning such models requires a large number of training examples for each target task. Simultaneously, many realistic NLP problems are "few shot", without a sufficiently large training set. In this work, we propose a novel conditional neural process-based approach for few-shot text classification that learns to transfer from other diverse tasks with rich annotation. Our key idea is to represent each task using gradient information from a base model and to train an adaptation network that modulates a text classifier conditioned on the task representation. While previous task-aware few-shot learners represent tasks by input encoding, our novel task representation is more powerful, as the gradient captures input-output relationships of a task. Experimental results show that our approach outperforms traditional fine-tuning, sequential transfer learning, and state-of-the-art meta learning approaches on a collection of diverse few-shot tasks. We further conducted analysis and ablations to justify our design choices.

1 Introduction

Transformer-based pretrained large-scale language models (LMs) have achieved tremendous success on many NLP tasks [10, 25], but require a large number of in-domain labeled examples for fine-tuning [41]. One approach to alleviate that issue is to fine-tune the model on a intermediate (source) task that is related to the target task. While previous work has focused on this setting for a single source task [24, 38], the problem of transferring from a set of source tasks has not been thoroughly considered. Wang et al. [39] showed that a naïve combination of multiple source tasks may negatively impact target task performance. Additionally, transfer learning methods typically consider the setting where more than a medium (N>100) number of training examples are available for both the source and target tasks.

In this work we develop a method to improve a large pre-trained LM for few-shot text classification problems by transferring from multiple source tasks. While it is possible to directly adapt recent advances in meta-learning developed for few-shot image classification, a unique challenge in the NLP setting is that source tasks do no share the same task structure. Namely, on standard few-shot image classification benchmarks, the training tasks are sampled from a “single” larger dataset, and the label space contains the same task structure for all tasks. In contrast, in text classification tasks, the set of source tasks available during training can range from sentiment analysis to grammatical acceptability judgment. Different source tasks could not only be different in terms of input domain, but also their task structure (i.e. label semantics, and number of output labels).

Approaches like MAML [11] and prototypical networks (ProtoNet) [32] were not designed to meta-learn from such a diverse collection of tasks. CNAP [28], a conditional neural process [12], explicitly

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

designed for learning from heterogeneous task distributions by adding a task conditional adaptation mechanism is better suited for solving tasks with a shifted task distribution. We design our approach based on the CNAP framework. Compared with CNAP, we use pretrained transformers for text sequence classification instead of convolutional networks for image classification. We insert adapter modules [15] into a pretrained LM to ensure efficiency and maximal parameter-sharing, and learn to modulate those adapter parameters conditioned on tasks, while CNAP learns to adapt the whole feature extractor. Additionally, we use gradient information for task representation as opposed to average input encoding in CNAP, since gradients can capture information from both input and ouput space.

In summary, our main contributions and findings are:

- We propose a novel conditional neural process-based approach that learns to modulate pretrained LMs for diverse text classification tasks using gradient information to represent task.
- We use pre-trained BERT with adapter modules [15] as the feature extractor and show that it works better than using only the BERT model, while also being more efficient to train.
- We compare our approach with traditional fine-tuning, sequential transfer learning, and state-of-the-art meta-learning approaches on a collection of diverse few-shot text classification tasks, and show that our approach achieves the best performance.

2 Related Work

2.1 Few-shot text classification

There is a wide range of text classification tasks, such as user intent classification, sentiment analysis, among others. Few-shot text classification tasks refer to those that contain novel classes unseen in training tasks and where only a few labeled examples are given for each class [42]. Meta-learning approaches have been proposed for this problem [6, 13, 4]. To increase the amount and diversity of training episodes, previous work trained few-shot learners by semi-supervised learning [31] and self-supervised learning [5]. We compare different meta-learning approaches following the experiment procedures of [4].

2.2 Meta-learning

Metric-based meta-learning approaches [37, 32, 34] try to learn embedding models on which simple classifiers can achieve good performance. Optimization-based approaches [11, 29, 21] are designed to learn good parameter initialization that can quickly adapt to new tasks within a few gradient descent steps. "Black box" or model-based meta-learning approaches use neural networks to embed task information and predict test examples conditioned on the task information [30, 27, 19, 22, 12, 28]. Our approach falls into this third category and is mostly related to CNAP [28]. Compared with CNAP we use a different feature extractor and different mechanisms for modulation, as well as focusing on different problems. An important difference with CNAP is that we use gradient information as features to represent tasks, instead of using average input encoding.

2.3 Transfer learning

Our work is also related to transfer learning. Instead of fine-tuning a pretrained language model directly on a target task, Phang et al. [24], Talmor and Berant [35] showed that intermediate training on data-rich supervised tasks is beneficial for downstream performance. Wang et al. [39] found that there is no guarantee that multi-task learning or intermediate training yields better performance than direct fine-tuning on target tasks. Transferability estimation is the task of predicting whether it is beneficial to transfer from some task to another one. There have been works estimating transferability of NLP tasks based on handcrafted features [7, 16], while data-driven approaches have been studied mainly for computer vision tasks [43, 36, 33, 20].

TASK2VEC is a task embedding approach based on the Fisher Information Matrix (FIM) [2]. Transferability between tasks can be easily evaluated by the distance between their task embeddings according to some simple metric. In the NLP domain, given a target task and a predefined set of source tasks,

Vu et al. [38] proposed to use task embedding based on FIM to measure task similarity and predict transferability according to the distance between task embeddings. We draw inspiration from this work to use gradients as features for task representation. Instead of one-to-one transfer, our work can be seen as transferring from a set of source tasks to new tasks through meta-learning.

3 Problem Definition

Following the terminologies in meta learning, each task (or episode), $t = (\mathcal{S}, \mathcal{Q})$, is specified by a support set (training set) $\mathcal{S}$ and a query set (test set) $\mathcal{Q}$, where $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{|\mathcal{S}|}$, $\mathcal{Q} = \{(x_i, y_i)\}_{i=1}^{|\mathcal{Q}|}$, x_i is a text sequence and y_i is a discrete value corresponding to a class. Different tasks could have different input domains, different numbers of classes, different label space, and so on.

Our goal is to train a text classification model on a set of labeled datasets $\{\mathcal{D}_i^s\}_{i=1}^N$ of N source tasks. The model is expected to achieve good performance on new target tasks after training. For each target task, a small labeled dataset, e.g., five shots for each class, is given for adaptation or fine-tuning and the full test dataset is used for evaluation. Note that when learning on target tasks, we assume no access to the source tasks nor other target tasks. This is different with multi-task learning where target tasks are learned together with source tasks.

4 Model Design

Figure 1 shows an overview of our model architecture, mainly consisting of three parts: a base model denoted by f, a task embedding network denoted by d, and an adaptation network denoted by a.

To make a prediction, our model takes the following steps: 1) given a support set, it uses the base model (Section 4.1) to compute gradients w.r.t. a subset of its parameters to use as input to the task embedding network (Section 4.2), 2) the task embedding network maps these gradients to task representations, 3) layer-wise adaptation networks (Section 4.3) take as input the task representation and output adaptation parameters, and lastly 4) the adaptation parameters are applied to the base model before it predicts on the query set.

4.1 Base Model: BERT & Bottleneck Adapters

Our base model is built upon transformer-based pretrained LMs. We use the pretrained $\text{BERT}_{\text{BASE}}$ model throughout this paper, although other transformer-based pretrained LMs are also applicable [26, 17]. Following Houlsby et al. [15], we insert two adapter modules containing bottleneck layers into each transformer layer of BERT. Each bottleneck adapter is denoted as $\alpha_l, l = 1...2L$ where L is the total number of transformer layers in BERT. We use the bottleneck adapters because, first, BERT with adapters are more efficient to train and less vulnerable to overfitting – a desirable property in the few-shot setting. Additionally, this simplifies our goal of using gradients for task representation. It is infeasible to use the gradients of the whole BERT model because the dimensionality is too large. Instead, we compute gradients w.r.t. parameters in these bottleneck layers (< 0.5% of the number of parameters in BERT).

Since text classification is a sequence classification task, we need a pooling method to represent each sequence by a single embedding before the embedding is fed into a classifier. Following Devlin et al. [10], this single embedding is obtained by applying a linear transformation on top of the contextual embedding of the special "[CLS]" token inserted at the beginning of every sequence. We further apply a fully connected layer on top of the BERT output. Together, we name the pretrained BERT with adapters and the last linear layer as the 'base model', denoted by f. We denote the parameters of the pretrained BERT as θ, bottleneck adapters as $\{\alpha_l\}_{l=1}^{2}L$, and linear layer as ω.

For the classifier, we use the ProtoNet classifier [32]. Given a support set S, a query example x and our base model f, a nearest cluster classifier can be described as follows:

$$p^{\text{base}}(y = c \,|\, x) = \text{softmax}(\text{euc}(f(x; \theta, \alpha, \omega), \mu^c)), \quad \mu^c = \frac{1}{|S^c|} \sum_{x_i \in \mathcal{S}^c} f(x_i; \theta, \alpha, \omega) \qquad (1)$$

where $S^c = \{(x_i, y_i) \,|\, (x_i, y_i) \in \mathcal{S}, y_i = c\}$, μ^c is the cluster center in the embedding space for support examples with class c, and euc refers to the Euclidean distance.

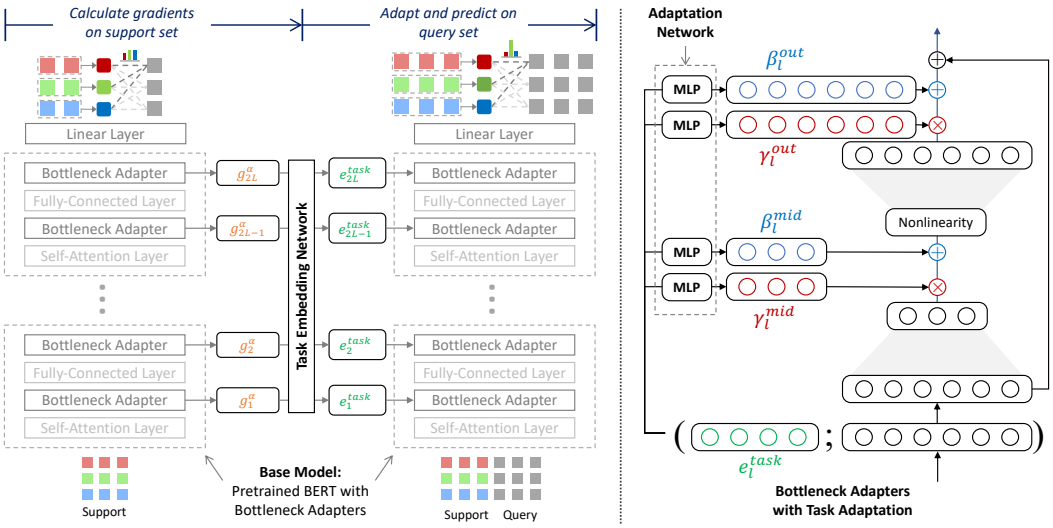

Figure 1: Model architecture overview. Left: The **base model** contains a pretrained BERT model with bottleneck adapters inserted in each transformer layer and a linear layer stacked on the top. The **task embedding network** is an RNN-based model that maps gradients from the base model into task representations in a layer-wise fashion, which are used for base model adaptation. Right: The **adaptation network** contains MLPs that takes as input the task representations and intermediate activate inside the base model, and outputs shifting and scaling parameters that are used to adapt the base model. Specifically, given the task embedding for a certain layer, we concatenate it with the input hidden representation to the current layer and map them into four shifting/scaling parameters by four MLP networks, which are applied on the hidden representations inside the adapter, referred as the auto-regressive adaptation.

4.2 Task Embedding Network for Per-Layer Task Encoding

Our task embedding network, denoted by d, is parametrized as a recurrent neural network (RNN) over the layers. It takes as input at each layer the gradient information of the adapter parameters at the given layer. Following Achille et al. [2], we use the gradient information defined by the FIM of the base model's parameters, denoted by $\Theta = \{\theta, \alpha, \omega\}$. The FIM is computed as:

$$F_\Theta = \mathbb{E}_{x,y\sim\hat{p}(x)p^{\text{base}}(y\,|\,x)} \nabla_\Theta \log p^{base}(y\,|\,x) \nabla_\Theta \log p^{base}(y\,|\,x)^T, \tag{2}$$

where $\hat{p}(x)$ is the empirical distribution defined by the dataset. Following Achille et al. [2], we only use the diagonal values of F_Θ. We only use those values corresponding to the adapter parameters, denoted by g_l^α for adapter layer l, $l = 1 \ldots 2L$.

Then, the RNN-based task embedding network d, parameterized by ϕ, maps each g_l^α into a task embedding:

$$e_l^{task} = d(h_l; \phi), \quad h_l = d(g_l^\alpha, h_{l-1}; \phi), \tag{3}$$

where h_l is the hidden representation, e_l^{task} is the task embedding at layer l, and h_0 is the learnable initial RNN hidden state.

4.3 Adaptation Network with Auto-Regressive Adaptation

As shown on the right side of Figure 1, at each layer l, the adaptation network output four adaptation parameters: γ_l^{mid} and β_l^{mid} are scaling and shifting adaptation parameters applied on the hidden representation after the middle layer, and γ_l^{out} and β_l^{out} are scaling and shifting adaptation parameters applied on the output hidden representation. The input to the adaptation network at each layer consists of both the task representation and the intermediate activation output by previous adapted layers. Due to the conditioning on previous modulated layers, we refer this adaptation method as auto-regressive adaptation, following Requeima et al. [28].

Algorithm 1: Training a task embedding network and adaptation network for quick adaptation to new tasks.

Input : T: A set of text classification tasks for training. $\mathcal{D}_i$: The labeled dataset of each task i. θ: Parameters of pretrained BERT. α: Parameters of the bottleneck layers learned from the first training stage.

Output : ϕ: Parameters of the task embedding model. ψ: Parameters of the adaptation network.

while *not converge* **do**
  Sample M episodes $\{t_i\}_{i=1}^{M}$, each episode $t_i = (\mathcal{S}_i, \mathcal{Q}_i), \mathcal{S}_i \subseteq \mathcal{D}_i, \mathcal{Q}_i \subseteq \mathcal{D}_i$
  for $i \leftarrow 1$ **to** M **do**
    for $s \leftarrow 1$ **to** S **do**
      Sample a few examples for each class from $\mathcal{S}_i$ and build prototypes $\{\mu^c\}$
      Sample another subset of examples $\hat{\mathcal{S}}_i \subseteq \mathcal{S}_i$
      $e_j = f(x_j; \theta, \alpha, \omega), (x_j, y_j) \in \hat{\mathcal{S}}_i$ `// encoding input in` $\hat{\mathcal{S}}_i$
      Sample $y'_j \sim p^{\text{base}}(y \,|\, x_j)$
      $g_s^{\alpha} = -\sum_{j=1}^{|\hat{\mathcal{S}}_i|} \nabla_{\alpha} \log p^{\text{base}}(y = y'_j \,|\, x_j)^2, (x_j, y_j) \in \hat{\mathcal{S}}_i$ `// calculating FIM and only // keep diagonal values`
    end
    $g^{\alpha} = \frac{1}{S} \sum_{s=1}^{S} g_s^{\alpha}$
    $e^{task} = d(g^{\alpha}; \phi)$ `// generating task representation`
    $\gamma^{in}, \beta^{in}, \gamma^{out}, \beta^{out} = a(e^{task}; \psi)$ `// generating adaptation parameters`
    $e_j = f(x_j; \theta, \alpha, \omega, \gamma^{in}, \beta^{in}, \gamma^{out}, \beta^{out})$ `// encoding input with adaptation // parameters`
    $\mu^c = mean(e_j), (x_j, y_j) \in \mathcal{S}_i, y_j = c$ `// build prototype of each class`
    $\ell_i = \sum_{j=1}^{|\mathcal{Q}_i|} -\log p^{\text{final}}(y = y_j | x_j)$ `// calculating loss on query`
  end
  $\{\phi, \psi\} \leftarrow \{\phi, \psi\} - lr \cdot \nabla_{\{\phi,\psi\}} \sum_{i=1}^{M} \ell_i$ `// gradient descent on` ϕ `and` ψ
end
return ϕ, ψ

Starting from the input layer, the adaptation network a produces adaptation parameters for the first adapter layer as follows:

$$\gamma_1^{mid}, \beta_1^{mid}, \gamma_1^{out}, \beta_1^{out} = a(e_1^{task}, x; \psi), \tag{4}$$

where x is input to the base model. We denote the bottleneck adapter network after modulation as α'.

For the subsequent layers, the adaptation network takes as input the combination of the task embedding and intermediate activation of the adapted base model, and outputs adaptation parameters as follows:

$$\gamma_l^{mid}, \beta_l^{mid}, \gamma_l^{out}, \beta_l^{out} = a(e_l^{task}, f_{l-1}(x; \theta, \{\alpha'_j\}_{j=1}^{l-1}); \psi),\ l = 2 \ldots 2L \tag{5}$$

where $f_{l-1}(x; \theta, \{\alpha'_j\}_{j=1}^{l-1})$ denotes the intermediate activation before layer l from the base model with the first $l-1$ adapters that have been modulated. Finally, the prediction from our modulated base model is written as:

$$p^{\text{final}}(y = c \,|\, x) = \text{softmax}(\cos(f(x; \theta, \alpha', \omega), \mu^c)), \quad \mu^c = \frac{1}{|S^c|} \sum_{x_i \in \mathcal{S}^c} f(x_i; \theta, \alpha', \omega) \tag{6}$$

Notice that the only difference between $p^{\text{final}}(y = c \,|\, x)$ and $p^{\text{base}}(y = c \,|\, x)$ is using α' or α, i.e. our task-specific modulation.

Our adaptation mechanism is similar with the Film layer [23] but different in terms of that here we only apply adaptation on the hidden representation of the "[CLS]", since only its embedding is used as the sequence embedding. This is analogous to only adapting the the first channel of each hidden representation to a Film layer. We also tried adapting embeddings of all tokens but performance is worse, as shown in Section 6.2.

4.4 Model Training

Similar with CNAP [28], we train our model in two stages. In the first stage, we train the base model using episodic training [32]. The training algorithm is showed in Appendix A. Given an episode

$t = (\mathcal{S}, \mathcal{Q})$, the loss is computed as:

$$\ell_{pn}(t) = \frac{1}{|\mathcal{Q}|} \sum_{x_i \in \mathcal{Q}} -\log p^{\text{base}}(y_i \mid x_i), \tag{7}$$

During this stage, only the parameters of the bottleneck adapters, layer normalization parameters and the top linear layer are updated while other parameters of the base model are frozen. The parameters are learned by:

$$\alpha^*, \omega^* = \arg\min_{\alpha,\omega} \mathbb{E}_{t \in T} \ell_{pn}(t). \tag{8}$$

In the second stage, the task embedding network and the adaptation network are trained episodically to generate good quality task embeddings and adaptation parameters for better performance on the query set of each episode. Specifically, as shown in Algorithm 1, we freeze the encoding network and only train the task embedding network d and the adaptation network a. In this stage, the frozen base model is used to generate task-specific gradient information and also be adapted to predict the query labels. Same with the first training stage, we also use the ProtoNet loss to train d and a but use the adapted based model. The parameters of the task embedding network ϕ and the adaptation network ψ is learned by:

$$\phi^*, \psi^* = \arg\min_{\phi,\psi} \mathbb{E}_{t \in T} \ell'_{pn}(t), \quad \ell'_{pn}(t) = \frac{1}{|\mathcal{Q}|} \sum_{x_i \in \mathcal{Q}} -\log p^{\text{final}}(y_i \mid x_i), \tag{9}$$

where the loss is computed based on the modulated model.

5 Experiments and Results

5.1 Experiment setup

We use datasets for training and testing that have different input domains and different numbers of labels. Each dataset appears during either training or testing – not both. Following [4], we use tasks from the GLUE benchmark [40] for training. Specifically, we use WNLI (m/mm), SST-2, QQP, RTE, MRPC, QNLI, and the SNLI dataset [8], to which we refer as our 'meta-training datasets'. The validation set of each dataset is used for hyperparameter searching and model selection. We train our model and other meta-learning models by sampling episodes from the meta-training tasks. The sampling process first selects a dataset and then randomly selects k-shot examples for each class as the support set and another k-shot as the query set. As with Bansal et al. [4], the probability of a task being selected is proportional to the square root of its dataset size.

We use the same test datasets as Bansal et al. [4], but also add several new datasets to increase task diversity, namely the Yelp [1] dataset to predict review stars, the SNIPS dataset [9] for intent detection, and the HuffPost [18] dataset for news headline classification. We refer to this set of tasks as our 'meta-testing datasets'. For each test task and a specific number of shot k, ten k-shot datasets are randomly sampled. Each time one of the ten datasets is given for model training or fine-tuning, the model is then tested on the full test dataset of the corresponding task. This is in line with real world applications where models built on a small training set are tested on the full test set.

Several test datasets used by Bansal et al. [4] contain very long sentences, while the $\text{BERT}_{\text{BERT}}$ (also used by Bansal et al. [4]) only allows for sequences up to 128 tokens. Truncating long sequences beyond the maximal length might lose important information for text classification, leading to noise in the test results. In order ensure the results can truly reflect text classification, we discard a few datasets used by Bansal et al. [4] that contain many very long sentences. See Appendix B for more details about the dataset we use.

5.2 Few shot text classification results

We use the cased version of the $\text{BERT}_{\text{BASE}}$ model for all experiments. We compare our proposed approach with the following approaches:

BERT. The pretrained BERT is simply fine-tuned on the labeled training data of the target task and then evaluated on the corresponding test data. No transfer learning happens with this model.

Table 1: Results on diverse few-shot text classification tasks. Results marked with '*' are from [4]. For Leopard, we reuse the their results on the first seven tasks and report the results of our implementation on the last three newly added tasks.

	Model	**BERT***		**MT-BERT***		**Leopard**		**PN-BERT**		**PN-BN**		**Grad2Vec**	
#		**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
	airline	42.76	13.50	46.29	12.26	54.95	11.81	65.39	12.73	65.33	7.95	**70.64**	3.95
	disaster	**55.73**	10.29	50.61	8.33	51.45	4.25	54.01	2.90	53.48	4.76	55.43	5.89
	emotion	9.20	3.22	9.84	2.14	11.71	2.16	11.69	1.87	12.52	1.32	**12.76**	1.35
	political_audience	51.89	1.72	51.53	1.80	52.60	3.51	**52.77**	5.86	51.88	6.37	51.28	5.74
	political_bias	54.57	5.02	54.66	3.74	60.49	6.66	58.26	10.42	**61.72**	5.65	58.74	9.43
4	political_message	15.64	2.73	14.49	1.75	15.69	1.57	17.82	1.33	20.98	1.69	**21.13**	1.97
	rating_kitchen	34.76	11.20	36.77	10.62	50.21	9.63	**58.47**	11.12	55.99	9.85	57.09	9.74
	huffpost_10	-	-	-	-	11.8	1.41	14.97	1.69	16.81	2.52	**18.5**	2
	snips	-	-	-	-	21.36	2.7	28.99	3.93	46.29	3.91	**52.51**	2.68
	yelp	-	-	-	-	36.95	2.98	42.84	2.66	42.64	2.93	**43**	3.55
	Average	-	-	-	-	36.72	4.67	40.52	5.45	42.76	4.70	**44.11**	4.63
	airline	38.00	17.06	49.81	10.86	61.44	3.90	69.14	4.84	69.37	2.46	**72.04**	2.58
	disaster	56.31	9.57	54.93	7.88	55.96	3.58	54.48	3.17	53.85	3.03	**57.49**	5.36
	emotion	8.21	2.12	11.21	2.11	12.90	1.63	13.10	2.64	13.87	1.82	**13.99**	1.90
	political_audience	52.80	2.72	54.34	2.88	54.31	3.95	**55.17**	4.28	53.08	6.08	52.60	5.55
	political_bias	56.15	3.75	54.79	4.19	61.74	6.73	63.22	1.96	**65.36**	2.03	64.06	1.12
8	political_message	13.38	1.74	15.24	2.81	18.02	2.32	20.40	1.12	**21.64**	1.72	21.31	1.16
	rating_kitchen	34.49	8.72	47.98	9.73	53.72	10.31	57.08	11.54	56.27	10.70	**58.35**	9.83
	huffpost_10	-	-	-	-	12.73	2.23	16.52	1.48	19.03	2.18	**21.12**	1.69
	snips	-	-	-	-	20.51	2.93	32.19	1.85	52.74	2.74	**57.19**	2.77
	yelp	-	-	-	-	38.31	3.52	**44.7**	1.68	43.83	2.45	43.66	1.65
	Average	-	-	-	-	38.96	4.11	42.60	3.46	44.90	3.52	**46.18**	3.36
	airline	58.01	8.23	57.25	9.90	62.15	5.56	71.06	1.60	69.83	1.80	**72.30**	1.75
	disaster	**64.52**	8.93	60.70	6.05	61.32	2.83	55.30	2.68	57.38	5.25	59.63	3.11
	emotion	13.43	2.51	12.75	2.04	13.38	2.20	12.81	1.21	14.11	1.12	**13.72**	1.24
	political_audience	**58.45**	4.98	55.14	4.57	57.71	3.52	56.16	2.81	57.23	2.77	55.46	3.34
	political_bias	60.96	4.25	60.30	3.26	65.08	2.14	61.98	6.89	**65.38**	1.71	63.83	0.74
16	political_message	20.67	3.89	19.20	2.20	18.07	2.41	21.36	0.86	**24.00**	1.39	22.22	1.20
	rating_kitchen	47.94	8.28	53.79	9.47	57.00	8.69	61.00	9.17	59.45	8.33	**61.72**	6.38
	huffpost_10	-	-	-	-	13.78	1.05	17.74	1.42	21.43	1.53	**23.57**	1.76
	snips	-	-	-	-	24.49	1.62	33.84	3.08	54.64	2.06	**59.47**	1.91
	yelp	-	-	-	-	39.7	2.31	**45.5**	1.72	44.78	1.8	44.87	2.09
	Average	-	-	-	-	41.27	3.23	43.67	3.14	46.82	2.78	**47.68**	2.35

MT-BERT. The pretrained BERT is first trained on the meta-training tasks via multi-task learning and then fine-tuned and evaluated on each test task.

ProtoNet-BERT. ProtoNet using pretrained BERT plus a linear layer as the feature extractor. The model is trained episodically on the meta-training datasets.

ProtoNet-BN. Similar with ProtoNet-BERT but with adapter modules inserted in the transformer layers. Only the parameters of the adapters, linear layer, and layer normalization layers inside BERT are updated during training. The model is trained episodically on the meta-training datasets.

MAML based approach. We compare with the MAML-based approach, named Leopard, proposed by Bansal et al. [4] with first-order approximation and meta-learned per-layer learning rates. We re-implemented this model, and include both their reported results and results of our implementation for fair comparison. See Appendix C for more details.

Grad2Task. This is our proposed model built upon pretrained ProtoNet-BERT BN and trained episodically to learn to adapt.

Results are shown in Table 1. Overall, our proposed method achieves the best performance. Our model is built upon ProtoNet-BN but keeps its parameters untouched and only learns to adapt the bottleneck adapter modules for different tasks. On average, our approach improves over ProtoNet-BN by 1.02%, indicating task conditioning can further improve a strong baseline.

Surprisingly, we find that ProtoNet-BERT achieves very good performance and outperforms the optimization-based method, Leopard, while Bansal et al. [4] reported the opposite results. We thus re-implemented both Leopard and ProtoNet-BERT and confirm that our MAML implementation has similar performance with Leopard but our ProtoNet-BERT results are much better than theirs (see Appendix C for a more detailed comparison).

Although we observe that ProtoNet-BERT has better performance and faster convergence rates during training and validation, it is outperformed by ProtoNet-BN which has orders of magnitude fewer parameters to learn (fewer than 0.5% of the number of BERT's parameters). We hypothesize this is because ProtoNet-BERT is more vulnerable to overfitting on the meta-training tasks.

6 Analysis and Ablations

6.1 Same/different task classification

We conduct a toy experiment to evaluate whether gradients are capable of representing tasks. This experiment is about classifying whether two few-shot datasets are sampled from the same task. For each few-shot dataset, we further split it into two subsets – one as the support set and the other as the query set. We then calculate the ProtoNet loss and its gradients using the base model. We repeat this process on the other few-shot training set and feed the gradients on the two datasets into a single linear layer, respectively. Prediction is given by the cosine similarity between the two representations after linear transformation and the binary cross entropy loss is used for training.

We train the MLP classifier on few-shot dataset pairs sampled from our meta-training datasets, and test it on dataset pairs sampled from our meta-testing datasets. Figure 2 shows the AUC curves on the testing set. Although the model is simple, it can predict same/different labels reasonably well on unseen tasks during training. With more shots, the gradient becomes more reliable, thus the model achieves better performance, resulting in an AUC score of 0.84 with 16 shots.

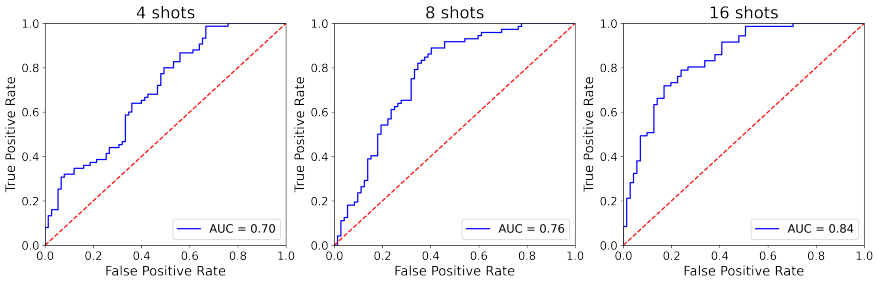

Figure 2: Results on the same/different experiments. Each column shows the results of classifying pairs of dataset with a certain number of shots.

6.2 Ablation Study

We conduct ablation studies to justify our decision choices. We report the average accuracy on meta-testing datasets of different model variants in Table 2. Results show that our approach achieves the best average accuracy among all model variants, resulting in average accuracy of 45.99%. The full results are included in Appendix D

Does the adaptation really help? Starting from the base model trained after the first stage, we compare our approach with just training the base model using the ProtoNet loss for the same number of steps. Shown as "ProtoNet Longer Training" in Table 2, the performance of this approach is worse than our proposed approach. This justifies that we can do better by adapting the base model than training it for more steps.

Table 2: Ablation results.

Model	Mean Acc.
Grad2Task w/ Gradients	**45.99**
ProtoNet Longer Training	45.10
Grad2Task w/ X	45.66
Grad2Task w/ X&Y	45.16
Grad2Task Adapt All	44.57
Grad2Task w/ Pretrained TaskEmb	45.68
Hypernetwork	44.79

How are gradients compared with other task representations? We compare our approach with other CNP-based approaches with different methods of task embedding. Similar to Requeima et al. [28], we use the average embeddings of the training sequences as representations for a task, and keep the other model architecture unchanged.

We also consider using the embeddings of both input sequences and labels for task representation. We treat labels as normal text instead of discrete values and encode them using the BERT model. The label embedding and average input embeddings are concatenated as the task representation. Shown as "Grad2Task w/ X" and "Grad2Task w/ XY", respectively, in Table 2, both of these two approach underperform our gradient-based method. Note that we can also combine those methods for task representation, which we leave for future work.

Adapt all or just "[CLS]"? The only difference between this model and ours is whether we apply the generated adaptation parameters only on the hidden representation of the "[CLS]" token or on all tokens in each sequence. Shown as "Grad2Task Adapt All" in Table 2, this model performs worse, resulting in 44.57% on average.

Generating model parameters or adaptation parameters? We also experimented with a hypernetwork [14]-based approach. Conditioned on the task representation, we use a neural network (hypernetwork) to output the parameters of bottleneck adapters directly, instead of freezing the bottleneck adapters and only generating scaling/shifting parameters, as in our proposed model. The model with a hypernetwork has higher flexibility for task adaptation, but the size of the hypernework is large, as it maps high-dimensional gradient information to high-dimensional model parameters, which brings challenges for optimization and suceptibility for overfitting. As shown in Table 2, it achieves accuracy of 44.79% on average and is worse than our proposed approach.

Pretraining task embedding? For this model, we first pretrain the task embedding model for the same/different tasks, then freeze and combine it with other components following our proposed approach. We observe that it has close but slightly worse performance (45.68%) than Grad2Task without task embedding pretraining. We hypothesize the task embedding trained with the same/different tasks is not good enough. We expect this model will benefit from pretraining the task embedding model on a set of datasets with higher diversity and training the task embedding model using more advanced metric-learning approaches, which we leave for future work.

7 Discussion and Conclusion

In this paper, we propose a novel model-based meta-learning approach for few-shot text classification and show the feasibility of using gradient information for task conditioning. Our approach is explicitly designed for learning from diverse text classification tasks how to adapt a base model conditioned on different tasks. We use pretrained BERT with bottleneck adapters as the base model and modulate the adapters conditioned on task representations based on gradient information.

Our work is an inaugural exploration of using gradient-based task representations for meta-learning. It has several limitations. First, the way we use neural networks to encode high-dimensional data significantly increases the number of parameters to train. For future work, we will explore more efficient ways for FIM calculation and better ways to handle high-dimensional gradient information. Second, we only focus on text sequence classification tasks in this paper. There are various other types of NLP tasks to explore, such as question answering and sequence labeling tasks, which present many distinct challenges. Also, we only consider transferring between text classification tasks. As shown in Vu et al. [38], positive transfer can happen among different types of NLP tasks. Thus, we intend to explore meta-learning or transfer learning approaches to learning from different types of tasks.

Ethical Concerns. Our work advances the field of few-shot text classification. While not having any direct negative societal impact, one can imagine that a few-shot text classifier is abused by a malicious user. For example, classifying trendy keywords on the internet naturally falls into the application area of our method as example phrases containing the new keyword can be limited in number. Toxic language is known to be a pervasive problem in the Internet [3], and a malicious user could use a good few-shot text classifier to further perpetuate these problems (e.g., by classifying toxic text as benign and fooling the end user). On the other hand, the same system can be used to promote fairness.

References

[1] Yelp open dataset. https://www.yelp.com/dataset. (Accessed on 05/20/2021).

[2] A. Achille, M. Lam, R. Tewari, A. Ravichandran, S. Maji, C. C. Fowlkes, S. Soatto, and P. Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6430–6439, 2019.

[3] R. Adragna, E. Creager, D. Madras, and R. Zemel. Fairness and robustness in invariant learning: A case study in toxicity classification. *arXiv preprint arXiv:2011.06485*, 2020.

[4] T. Bansal, R. Jha, and A. McCallum. Learning to few-shot learn across diverse natural language classification tasks. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5108–5123, 2020.

[5] T. Bansal, R. Jha, T. Munkhdalai, and A. McCallum. Self-supervised meta-learning for few-shot natural language classification tasks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 522–534, 2020.

[6] Y. Bao, M. Wu, S. Chang, and R. Barzilay. Few-shot text classification with distributional signatures. In *International Conference on Learning Representations*, 2019.

[7] J. Bingel and A. Søgaard. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 164–169, 2017.

[8] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. 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, 2015.

[9] A. Coucke, A. Saade, A. Ball, T. Bluche, A. Caulier, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone, T. Lavril, et al. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*, 2018.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[11] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[12] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2018.

[13] R. Geng, B. Li, Y. Li, X. Zhu, P. Jian, and J. Sun. 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, 2019.

[14] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[15] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[16] E. Kerinećć, A. Søgaard, and C. Braud. When does deep multi-task learning work for loosely related document classification tasks? In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 1–8. Association for Computational Linguistics, 2018.

[17] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[18] R. Misra. News category dataset. https://www.kaggle.com/rmisra/news-category-dataset. (Accessed on 05/20/2021).

[19] T. Munkhdalai and H. Yu. Meta networks. In *International Conference on Machine Learning*, pages 2554–2563. PMLR, 2017.

[20] C. Nguyen, T. Hassner, M. Seeger, and C. Archambeau. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, pages 7294–7305. PMLR, 2020.

[21] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[22] B. N. Oreshkin, P. Rodriguez, and A. Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 719–729, 2018.

[23] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[24] J. Phang, T. Févry, and S. R. Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.

[25] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[26] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67, 2020.

[27] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. 2016.

[28] J. Requeima, J. Gordon, J. Bronskill, S. Nowozin, and R. Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. *Advances in Neural Information Processing Systems 32 (NIPS 2019)*.

[29] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2018.

[30] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[31] T. Schick and H. Schütze. Exploiting cloze questions for few-shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

[32] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4080–4090, 2017.

[33] J. Song, Y. Chen, J. Ye, X. Wang, C. Shen, F. Mao, and M. Song. Depara: Deep attribution graph for deep knowledge transferability. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3922–3930, 2020.

[34] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

[35] A. Talmor and J. Berant. MultiQA: An empirical investigation of generalization and transfer in reading comprehension. *arXiv preprint arXiv:1905.13453*, 2019.

[36] A. T. Tran, C. V. Nguyen, and T. Hassner. Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1395–1405, 2019.

[37] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3637–3645, 2016.

[38] T. Vu, T. Wang, T. Munkhdalai, A. Sordoni, A. Trischler, A. Mattarella-Micke, S. Maji, and M. Iyyer. Exploring and predicting transferability across nlp tasks. *arXiv preprint arXiv:2005.00770*, 2020.

[39] A. Wang, J. Hula, P. Xia, R. Pappagari, R. T. McCoy, R. Patel, N. Kim, I. Tenney, Y. Huang, K. Yu, et al. Can you tell me how to get past Sesame Street? Sentence-level pretraining beyond language modeling. *arXiv preprint arXiv:1812.10860*, 2018.

[40] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[41] D. Yogatama, C. d. M. d'Autume, J. Connor, T. Kocisky, M. Chrzanowski, L. Kong, A. Lazaridou, W. Ling, L. Yu, C. Dyer, et al. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*, 2019.

[42] M. Yu, X. Guo, J. Yi, S. Chang, S. Potdar, Y. Cheng, G. Tesauro, H. Wang, and B. Zhou. Diverse few-shot text classification with multiple metrics. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1206–1215, 2018.

[43] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 7.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [N/A] Not aware of any

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We only use datasets that publicly available.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No personal information is included.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]