Prototypical Fine-tuning: Towards Robust Performance Under Varying Data Sizes

Anonymous submission

Abstract

In this paper, we move towards combining large parametric models with non-parametric prototypical networks. We propose prototypical fine-tuning, a novel prototypical framework for fine-tuning pretrained language models (LM), which automatically learns a bias to improve predictive performance for varying data sizes, especially low-resource settings. Our prototypical fine-tuning approach can automatically adjust the model capacity according to the number of data points and the model's inherent attributes. Moreover, we propose four principles for effective prototype fine-tuning towards the global optimum. Experimental results across various datasets show that our work achieves significant performance improvements under various low-resource settings, as well as comparable and usually better performances in high-resource scenarios.

Introduction

Pretrained language models (LM) have achieved substantial success on a variety of NLP applications (Brown et al. 2020). Their high discriminative power is partly attributed to a weak inductive bias, which poses little constraints on the model expressivity, but leads to potential overfitting and local optimum, especially in low-resource settings (McCoy, Pavlick, and Linzen 2019; Belinkov, Henderson et al. 2021). On the other hand, non-parametric models, such as prototypical networks, (Snell, Swersky, and Zemel 2017; Yang et al. 2018; Shao et al. 2017) introduce strong inductive biases that increase the probability for reaching global optimum by explicitly modeling the inter-class and intra-class relations within the data. These methods are promising for improving generalization and increasing sample efficiency, and have achieved great success in few-shot learning (Snell, Swersky, and Zemel 2017; Allen et al. 2019). However, as strong inductive biases restrict the hypothesis set of function approximators, these models introduce false assumptions that may limit the expressivity of models as the number of data points increases (Baxter 2000).

In this paper, we move towards combining large parametric models with non-parametric prototypical networks so that we automatically learn a bias to improve predictive performance on varying sizes of datasets, especially for low-resource settings. Achieving this goal is non-trivial due to two major challenges. First (*adaptation*), how to introduce prototype modeling into large-scale language models to jointly ensure superior performances under low-resource scenarios while mitigating the over-reliance of prototype learning on strong inductive biases? Second (*capacity*), different datasets have diverse levels of complexity, thus requires different volumes of prototypes and parameters to properly represent the data distribution (Mettes, van der Pol, and Snoek 2019). Choosing model capacity generally requires heuristics and knowledge about the data distribution (Allen et al. 2019). How to determine the number of prototypes automatically by jointly considering complexity of the data distribution and the model's inherent attributes to ensure robust performances on datasets with varying sizes?

To solve these two challenges, we propose **Prototypical Fine-tuning (PFit)**. Our method represents the data within each class as a set of prototypes, each modeled as a mixture component, while learning the number of components automatically based on the model capacity and the complexity of data distribution. At the initial stage of training, our method is equivalent to ProtoNet (Snell, Swersky, and Zemel 2017), which ensures a high-quality initialization of the prototypes and facilitates convergence towards a globally optimal solution. As training proceeds, *PFit* chooses from fitting each new example with existing prototypes or initiating a new prototype based on the data point, according to whether the example is in-distribution. We allow the number of prototypes to grow and shrink dynamically with respect to the data complexity. In all, our method ensures a hypothesis space large enough for viable solutions to the tasks being learned (Baxter 2000), but small enough for generalization. Thus, our method is well-suited for both data-scarce and data-rich tasks, especially offering substantial performance gain under low-resource scenarios. The contributions of this paper are summarized as follows:

1) We propose *Prototypical Fine-tuning (PFit)*, a novel prototypical method for fine-tuning pretrained language models by integrating the strengths of bayesian non-parametrics with parametric pretrained LMs, leading to superior predictive performance under varying data sizes.

2) We propose four principles for effectively leveraging non-parametric methods , which are mixture prototype initialization, infinite mixture prototype creation, adaptive prototype refinement, and dynamic diversity regularization, so that our method maintains a compact set of prototype embeddings conducive to its decision-making;

3) We conduct extensive experiments and show that *PFit* achieves considerable performance improvements under varying data sizes, as well as comparable or better performances for high-resource settings.

Related Work

Prototypical Learning

Prototypical Learning (Mettes, van der Pol, and Snoek 2019; Snell, Swersky, and Zemel 2017; Oreshkin, Rodríguez López, and Lacoste 2018) is closely related to metric learning (Goldberger et al. 2004; Oh Song et al. 2016) and case-based reasoning (Li et al. 2018; Kim, Rudin, and Shah 2014; Kolodner 1992). Prototypical learning has been used for improving model interpretability (Li et al. 2018; Chen et al. 2019) and few-shot classification (Snell, Swersky, and Zemel 2017; Gao et al. 2019). Most existing works combine bayesian non-parametrics with representation learning and learn a single prototype for each class (Vinyals et al. 2016; Snell, Swersky, and Zemel 2017), based on the uni-modality assumption of embedded class examples. As an extension, multi-modal methods (Hjort et al. 2010; Rasmussen 1999; Mensink et al. 2013; Ackerman and Dasgupta 2014) assume an infinite number of mixtures, but only a finite number are used to generate the data. In particular, infinite mixture prototype (IMP) (Allen et al. 2019; Kulis and Jordan 2012) adaptively fits the data distribution through a varied number of clusters for each class. Notably, existing works lack an efficient mechanism to be dynamically scaled on datasets with varying sizes.

Low-Resource

Due to the cost of acquiring high-quality labeled data, researchers have been working on improving model performances under low-resource settings. A special case is few-shot learning (Wang et al. 2021; Tsimpoukelli et al. 2021; Perez, Kiela, and Cho 2021) in which data is severely limited. Earlier works address data sparsity through large unlabeled datasets (Gunel et al. 2020; Artetxe et al. 2018), data augmentation (Salamon and Bello 2017; Fadaee, Bisazza, and Monz 2017), or the removal of redundant information in the pretraining representation (Belinkov, Henderson et al. 2021). These methods usually introduce extra data or auxiliary models that increase the computational cost. Moreover, these works are usually designed for specific low-resource settings, but cannot robustly scale to larger datasets with performance guarantee (Zaken, Ravfogel, and Goldberg 2021). In this work, we propose a non-parametric method suited for both low- and high- resource scenarios that achieves particularly superior performances under the former settings.

Method

In this section, we first introduce the general workflow for fine-tuning language models. Then, we present the proposed approach *PFit* and show how it achieves superior performances under varying data sizes.

Problem Statement

Input The input for fine-tuning is a labeled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where each example $x_i \in X$ is an input text sequence, and $y_i \in \{1, \dots, C\}$ is the ground-truth label.
Output The outputs consist of 1) $\mathbb{P}(y \mid x_i)$, the prediction of each example x_i; 2) the fine-tuned model $f_\phi(\cdot) : X \to \mathbb{R}^C$ that maps an input sequence to a probability distribution over C classes.

Prototypical Fine-tuning

Pretrained language models are fine-tuned in a fully parametric way. The inference is conducted in a case-by-case manner by computing $\mathbb{P}(y \mid x_i)$. This way, performances of LMs predominantly stem from their large-scale parameters. In most cases, language models cannot efficiently leverage the class distribution and instance-level similarity among training examples.

To alleviate this problem, we explicitly model the inter-class and intra-class relations by introducing *prototypes*, which are embeddings in the same metric space as $f_\phi(x_i)$ that abstracts the essential semantics of multiple x_i's (Snell, Swersky, and Zemel 2017). In our case, we learn a compact set of prototypes $\mathbf{p}_{1\sim K}$ from $\mathcal{D}$ that serves as references for model prediction. We split the inference into two components: 1) **prototype prediction** of each $\mathbf{p}_k$ conditioned on the entire training set $\mathcal{D}$; 2) **prototype importance** that measures the compatibility between each $\mathbf{p}_k$ and x_i.

$$\mathbb{P}(y \mid x_i, \mathcal{D}) = \sum_{k=1}^{K} \underbrace{\mathbb{P}(y \mid \mathbf{p}_k, \mathcal{D})}_{\substack{\text{Prototype} \\ \text{Prediction}}} \underbrace{\mathbb{P}(\mathbf{p}_k \mid x_i)}_{\substack{\text{Prototype} \\ \text{Importance}}} \tag{1}$$

When infering on a new unclassified exemplar, prototypical networks calculates joint prediction probability $\mathbb{P}(y \mid x_i, \mathcal{D})$ and efficiently adapts existing prototypes to the new data rather than deriving answers from scratch. Under this formulation, prototypical networks focus on critical aspects of the data distribution and filter irrelevant or noisy features that are detrimental to their decision-making (Kolodner 1992) better at finding a globally optimal solution.

Prototypical Network Under the prototypical learning paradigm, each x_i is encoded into a D-dimensional feature vector $f_\phi(x_i) \in \mathbb{R}^D$ and matched against a set of prototypes $\mathbf{p}_k$ in a learned latent space according to some distance metric $d(\cdot, \cdot)$. The importance of prototype $\mathbf{p}_k$ with respect to x_i is measured by:

$$\mathbb{P}(\mathbf{p}_k \mid x_i) \propto \exp(-d(f_\phi(x_i), \mathbf{p}_k)) \tag{2}$$

where $d(\cdot, \cdot)$ is usually taken as the squared L2 distance. Prototypical methods (Snell, Swersky, and Zemel 2017) calculate the prototype of each class c as the averaged embedding of $\mathcal{D}_c \in \mathcal{D}$, or the support data within class c:

$$\mathbf{p}_c = \frac{1}{|\mathcal{D}_c|} \sum_{(x_i, y_i) \in \mathcal{D}_c} f_\phi(x_i) \tag{3}$$

Prediction is conducted by calculating the distance from $f_\phi(x_i)$ to each prototype vector.

$$\mathbb{P}(y = c \mid x_i) = \frac{\exp(-d(f_\phi(x_i), \mathbf{p}_c))}{\sum_{c'} \exp(-d(f_\phi(x_i), \mathbf{p}_{c'}))} \tag{4}$$

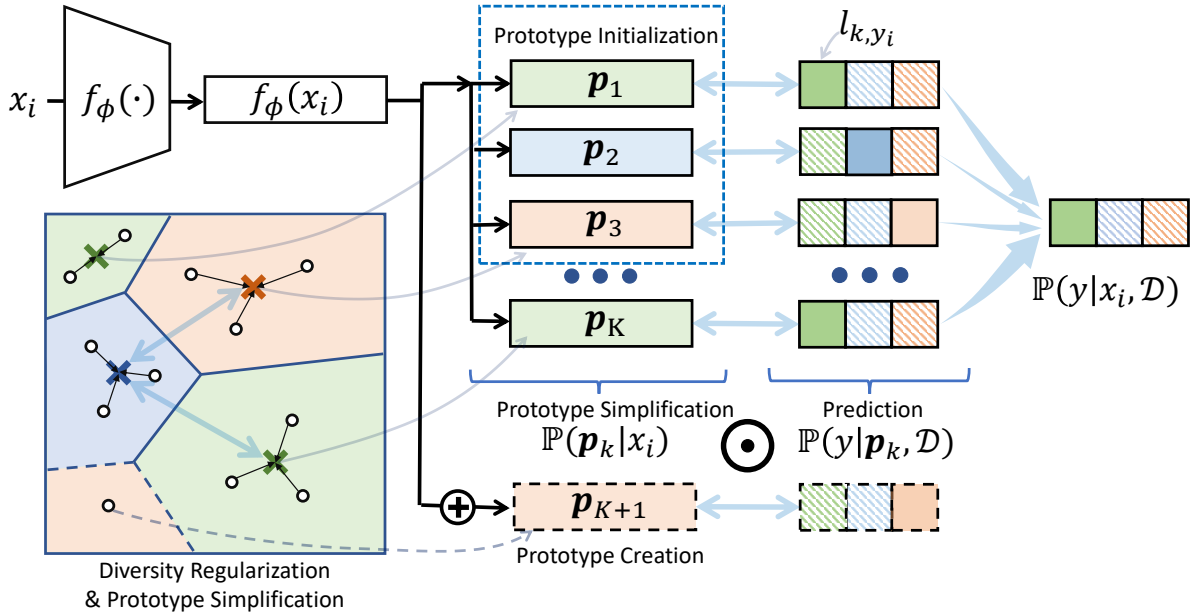

Figure 1: Our proposed *PFit* framework. During fine-tuning, the prediction of each example x_i is given by individual predictions of prototypes weighted by their distance to $f_\phi(x_i)$. *PFit* is able to dynamically add or prune prototypes, depending on whether existing prototypes can characterize the distribution of data.

Our PFit Framework Our goal is to learn a **Prototypical Fine-tuning (PFit)** network with robust performances under both low- and high-resource settings. The training procedure is detailed in Alg. 1. Compared with a transformer-based model, PFit contains an additional prototype learning module P that can flexibly adjust the capacity of its prototype embeddings $\mathbf{p}_k$ according to the data distribution. Our proposed PFit framework can be easily built on top of any backbone pretrained LMs. It pursues 4 approaches for robustness and expressivity under varying data sizes:

1) *Mixture Prototype Initialization.* We leverage the high-level semantic and linguistic features within pretrained representations to initialize the prototypes for faster convergence;

2) *Infinite Mixture Prototype Creation.* We extend infinite mixture prototype (Allen et al. 2019) to flexibly capture the complexity of the distribution. The prototype embeddings are fine-tuned along with the pretrained model in a data-driven manner;

3) *Adaptive Prototype Simplification.* To improve generalization and efficiency, we leverage simple inductive biases and maintain a compact set of prototypes that sufficiently represent the data distribution.

4) *Dynamic Diversity Regularization.* We dynamically enforce diversity among prototypes to ensure the expressivity and efficiency of our prototypical framework.

Mixture Prototype Initialization At the start of training, a simple yet strong inductive bias can ensure robust generalization since the amount of data is relatively limited (Snell, Swersky, and Zemel 2017). Previous studies have shown that transformers models encode high-level semantic and linguistic features in their representations (Tenney, Das, and Pavlick 2019; Jawahar, Sagot, and Seddah 2019; Hao et al. 2020; Jin et al. 2022; Yang et al. 2022). Therefore, we propose *mixture prototype initialization* to leverage the rich semantics in the sentence representations. Before fine-tuning starts, for each class c, we randomly sample a subset of n examples with label c, and aggregate their $f_\phi(x_i)$ with ϕ frozen:

$$\mathbf{p}_c = \text{Aggr}([f_\phi(x_1), \ldots, f_\phi(x_n)]) \quad (5)$$

In our case, we choose mean pooling as the aggregate function. We derive an initial set of C prototypes for all classes. Mixture prototype initialization is analogous to ProtoNet (Snell, Swersky, and Zemel 2017) and assumes that the classes form uni-modal clusters. Such initialization can empower the prototypes with the semantics of transformer word representations with little computational cost. Meanwhile, it does not require extra parameters or training, only the intrinsic features of language models.

Infinite Mixture Prototype Creation As the training proceeds, existing prototype embeddings become less capable of assimilating new data points. Higher capacity is required to model the growing data complexity. A straightforward remedy is to directly create a set of K prototypes and automatically learn meaningful representations from the training data. However, such method is largely subject to the choice of model capacity, either underfitting or overfitting the data (Snell, Swersky, and Zemel 2017; Ren et al. 2018). Second, in practice, this method usually yields highly similar prototypes (Ming et al. 2019) which hampers the expressivity of the prototype module and leads to sub-optimal convergence.

Motivated by (Allen et al. 2019), we seek informative samples from the training data that contributes to the diver-

sity of the prototypes and the expressivity of the fine-tuned models. Let P^c be the set of prototypes under class c. For each example $x_i \in \mathcal{D}_c$, if the minimum distance between $f_\phi(x_i)$ and any prototypes in P^c exceeds a threshold λ, a new prototype is created based on $f_\phi(x_i)$. The threshold distance λ is given by:

$$\lambda = 2\sigma \log \left(\frac{\alpha}{\left(1 + \frac{\rho}{\sigma}\right)^{d/2}} \right) \tag{6}$$

where σ is the cluster variance that is learned jointly with ϕ. ρ measures the standard deviation for the base distribution from which the cluster means are sampled. α is a hyperparameter that controls the concentration of clusters in the Chinese Restaurant Process (Wang and Blei 2009). This way, our approach can choose between fitting simple data distribution with low capacity and complex distribution with high capacity. As a further extension, we only allow the creation of prototypes after being fine-tuned for a certain number of steps. This way, the initial C prototypes are sufficiently trained along with ϕ so that meaningful prototypes can be created. For computational and storage efficiency, we restrict the size of prototypical networks to $|P_{max}|$.

Prediction During fine-tuning, language models classify new examples by applying an MLP on the word representations. PFit instead directly learns a distribution over all C classes for each $\mathbf{p}_k$. During fine-tuning, PFit learns a distribution over all C classes for each $\mathbf{p}_k$. Specifically, each $\mathbf{p}_k$ directly correlates with a learnable vector $\mathbf{l}_k \in \mathbb{R}^C$, which gives a probability distribution over all C classes:

$$\mathbb{P}(y = c \mid \mathbf{p}_k, \mathcal{D}) = \text{softmax}_c(l_{kc}); \;\; \mathbf{l}_k = [l_{k,1}, \ldots, l_{k,C}] \tag{7}$$

During initialization, for each newly created prototype $\mathbf{p}_k$, l_{k,y_i}, the value in $\mathbf{l}_k$ that corresponds to the ground-truth class y_i, is initialized to β, a positive real number. The value for all other classes $l_{k,c}, c \in \{1, \ldots, C\} \backslash y_i$ are initialized to $-\beta$. During fine-tuning, $\mathbf{l}_k$ is optimized together with its corresponding $\mathbf{p}_k$ in a data-driven manner. In order to keep the predicted label constant, l_{k,y_i}, which corresponds to the ground-truth class y_i, is constrained in $[0, +\infty)$. Predictions for other classes are constrained in $(-\infty, 0]$. Compared with existing prototypical approaches which usually use a constant value for prototype predictions, our method can achieve superior expressivity.

The prototype importance $z_{i,k}$ is given by the normalized Gaussian density:

$$\mathbb{P}(\mathbf{p}_k \mid x_i) = z_{i,k} = \frac{\mathcal{N}(f_\phi(x_i); \mathbf{p}_k, \sigma_k)}{\sum_{k'} \mathcal{N}(f_\phi(x_i); \mathbf{p}_{k'}, \sigma_{k'})} \tag{8}$$

The joint probability of $\mathbf{p}_k$ predicting class c is given by Eq. 1. Intuitively, our model is analogous to a retrieval system or an attention-based model (Vaswani et al. 2017), in which a query example is mapped against a set of key-value pairs. The weight assigned to each value is computed by a compatibility function between the query and each key. The prediction result is a linear combination of values weighed by the compatibility between the query example $f_\phi(x_i)$ and existing prototypes $\mathbf{p}_k \in P$.

Algorithm 1: Prototypical Fine-tuning

Input: $\mathcal{D} = (x_i, y_i)$, where each $y_i \in \{1, \ldots, C\}$
Output: prototypes $\mathbf{p}_k$ and the classification of each example $\mathbb{P}(y \mid \mathbf{p}_{1 \sim K}, x_i)$

```
Perform Mixture Prototype Initialization
for minibatch B_r ∈ D do
    Perform Infinite Mixture Prototype Creation and estimate λ according to Eq. 6
    for x_i ∈ B_r do
        for k ∈ {1, ..., K} do
            {Compute the distance from x_i to all existing prototypes}
            Calculate d_{i,k} = d(f_φ(x_i), p_k) for p_k ∈ P^{y_i}, and d_{i,k} = +∞ for p_k ∉ P^{y_i}
        end for
        if min_k d_{i,k} > λ then
            Create the K + 1-th prototype p_{K+1} using f_φ(x_i); Increment K by 1
        end if
    end for
    if min_k d_{i,k} > λ then
        Create the K+1-th prototype p_{K+1} using f_φ(x_i); Increment K by 1
    end if
end for
```

Adaptive Prototype Simplification Simple inductive biases usually imply robust generalization to varying datasets (Baxter 2000). To improve generalization and efficiency, we propose *adaptive prototype simplification* to ensure that a compact set of prototypes sufficiently represent the data distribution. As prototypes are created with examples that are sufficiently distinct from existing prototype embeddings, a minority of prototypes may be created by outliers that are rarely observed in the training data. These prototypes may failed to generalize to new data and are likely to introduce noisy features or spurious correlations. To determine unimportant prototypes, a simple way is to calculate the average prototype importance $z_{i,k}$ in Eq. 8 between each $\mathbf{p}_k$ and $f_\phi(x_i)$ in the training set. This leads to two potential issues: 1) acquiring $f_\phi(x_i)$ and calculating $z_{i,k}$ among $f_\phi(x_i)$ and $\mathbf{p}_k$ (Eq. 8) requires $O(N|P_{max}|)$, both of which are computationally expensive. 2) a direct remedy is to store all $z_{i,k}$ calculated during training and use them during simplification. However, the actual value of $z_{i,k}$ is constantly changing, as $f_\phi(x_i)$ and ϕ are continuously updating.

To solve the above two challenges, we introduce a sliding window of length δ, which records all $z_{i,k}, i \in [T - \delta, T]$ at any training step T. Upon prototype simplification, we prune prototypes with an average prototype importance $\mathbb{P}(\mathbf{p}_k \mid x_i)$ that fall below a threshold ε:

$$\frac{1}{\delta} \sum_i \omega(i, T, \delta) z_{i,k} < \varepsilon \tag{9}$$

ε is a hyperparameter controlling the minimum average importance of $\mathbf{p}_k$ on all $f_\phi(x_i), i \in [T - \delta, T]$. $\omega(i, T, \delta) =$

$\frac{1}{\delta}(i - T + \delta)$ is a linear discount factor that assigns higher importance to data points closer to T in the sliding window. Our method not only avoids storing and indexing the entire set of representations, but also takes into account the fact that, at any timestep T, $f_\phi(x_i)$ that are computed closer to T are more accurate. During each training epoch, Prototype simplification is performed for a small number of M times per epoch.

Comparison to Previous Works Previous works in few-shot learning (Snell, Swersky, and Zemel 2017; Allen et al. 2019) use an alternative way to ensure the efficiency of prototypes. They compute a new prototype for each mixture component using the episode-end cluster mean, whereas in our model fine-tuning scenario, constant re-estimation of $\mathbf{p}_k$ will break the continuity of the model's learning, leading to poor convergence. In contrast, our approach carefully considers the dynamically changing nature of example representation; global information, the discount factor $\omega(i, T, \delta)$ also mitigates overfitting a small batch.

Optimization & Extension **Dynamic Diversity Regularization** We dynamically control the diversity among the prototype embeddings by restricting the minimum pairwise distance among $\mathbf{p}_k$'s to λ. As P grows, λ increases, enforcing higher pairwise difference among prototypes so that PFit maintains a set of meaningful prototypes out of $f_\phi(x_i)$ that sufficiently adds to the diversity of P.

$$\mathcal{L}_{div} = \sum_{j=1}^{K} \sum_{k=j+1}^{K} \max\left(0, \lambda - \|\mathbf{p}_j - \mathbf{p}_k\|_2\right)^2 \qquad (10)$$

Extension to Regression Our approach can be readily applied to regression tasks by partitioning the range of y_i into a number of $N_{reg} \in \mathbb{N}^*$ bins and sample n examples from each bin. Each prototype prediction $\mathbf{l}_k$ becomes a real number computed as the averaged ground-truth y_i in each bin. Upon inference, we compute $\hat{y}_i$ as $\hat{y}_i = \sum_k z_{i,k}\mathbf{l}_k$.

Optimization We minimize cross-entropy loss for classification and mean squared error for regression. The full objective $\mathcal{L}$ linearly combines $\mathcal{L}_0$ and $\mathcal{L}_{div}$ using a hyperparameter ρ_d.

$$\mathcal{L}_0 = \begin{cases} \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} -y_{i,k}\log(\hat{y}_{i,k}) & \text{(classification)} \\ \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 & \text{(regression)} \end{cases}$$

$$\mathcal{L} = \mathcal{L}_0 + \rho_d \mathcal{L}_{div}$$

Interpretability

There are 3 ways to interpret the prototype embeddings $\mathbf{p}_k$: 1) jointly train a decoder that transforms $\mathbf{p}_k$ back to the input space (Li et al. 2018); 2) project $\mathbf{p}_k$ onto the nearest sequence in the input space (Ming et al. 2019); 3) project $\mathbf{p}_k$ onto examples that are sufficiently close to the prototype within a threshold τ: $\{x_i \mid \|\mathbf{p}_k - f_\phi(x_i)\|_2 < \tau\}$.

Experiment

Implementation Details We apply a batch size of 32 for the base models and 16 for the large models. We use $n = 8$ examples to initialize each prototype. The sequence length is set to 128. We first conduct a grid-search on α within $\{1, 0.5, 0.1, 0.05, 0.01\}$, ρ_d in $\{1e-4, 1e-5, 1e-6\}$, and ϵ in $\{1e-3, 3e-3, 1e-2\}$ using a subset of SST-2 with 1000 data points, and fix $\alpha = 0.1$, $\rho_d = 1e-5$, and $\epsilon = 1e-3$ to conduct all experiments without further tuning. We use the Adam optimizer (Kingma and Ba 2015) without weight decay. For both PFit and the baselines, we search the best learning rate from $[1e-5, 2e-5, 3e-5, 5e-5, 1e-4]$. Other hyperparameters of the baselines are set to their default values in their original papers. All experiments were run for a maximum of 5 epochs with early stopping. For hardware usage, experiments are run on servers with NVIDIA Tesla P100 (16GB GPU memory) or V100 (32GB GPU memory).

Robustness Across Varying Data Sizes

Experiment Setup Our goal is to demonstrate the superior performances of PFit across a wide range of data sizes, especially under low-resource scenarios. Thus, we evaluate our model across a diverse spectrum of data sizes using 4 representative datasets from the General Language Understanding Evaluation (GLUE) benchmark (Wang et al. 2019), including QNLI, MNLI-m, and MNLI-mm, 3 textual entailment datasets, as well as SST-2, a common benchmark for sentiment analysis. The datasets are chosen following (Gunel et al. 2020) so that both single sentence and sentence-pair classification datasets are included. We report our results on MNLI-m/mm in Tab. 1 and SST-2/QNLI in Tab. 3. We validate the performances using a moderate-sized model BERT-Base (Devlin et al. 2019) and a stronger baseline RoBERTa-Large (Liu et al. 2019) as our backbones. We randomly sample 100 to 5000 examples from the training set as our training set, taking into account the class distribution of the full training set, and test our model on the full dev set. We conduct experiments with 5 different seeds and report the accuracy and standard deviation.

Baselines We compare our PFit method against 6 baselines, including: 1) 4 non-parametric networks built on top of the backbone language models, including DP-Means (Kulis and Jordan 2012), ProtoNet (Snell, Swersky, and Zemel 2017), ProSeNet (Ming et al. 2019), and IMP (Allen et al. 2019); 2) VIBERT, a network designed for varying-resource fine-tuning (Belinkov, Henderson et al. 2021); 3) BSS, a regularization approach to suppress untransferable spectral components and improve fine-tuning (Wei, Xie, and Ma 2021) 4) the original LM.

Model Performances The definition of low-resource settings varies across data sets (Belinkov, Henderson et al. 2021; Gao et al. 2018). We follow existing literature (Jin et al. 2020; Belinkov, Henderson et al. 2021) to define low-resource settings as datasets with $\leq$ 5000 data points. Compared with the baselines, our PFit model consistently achieves the best performance among all datasets and data sizes. For MNLI-m/mm, our method leads to a maximum of 5.6 and 11.1 percent improvement (Tab. 1) compared to the best baseline models. For QNLI/SST-2, our method leads to improvement of 5.1 and 3.8 percent (Tab. 3). Results of RoBERTa-Large are shown in the Appendix.

MNLI-m	DPMeans	ProtoNet	ProSeNet	IMP	BSS	ViBERT	Raw	PFit	Impr. (b) %
100	38.6 ± 2.1	35.5 ± 0.6	36.3 ± 1.2	36.7 ± 0.7	36.5 ± 0.9	39.3 ± 1.1	33.4 ± 0.8	40.6 ± 2.2	3.2
200	39.1 ± 1.3	37.5 ± 1.6	38.6 ± 0.4	41.2 ± 1.3	40.0 ± 0.7	40.1 ± 0.4	35.3 ± 0.9	42.4 ± 1.9	2.8
500	41.4 ± 1.5	43.2 ± 0.5	44.7 ± 0.8	47.5 ± 0.8	50.1 ± 0.5	49.2 ± 0.7	38.5 ± 0.8	52.9 ± 1.8	5.6
1000	53.0 ± 1.0	50.5 ± 0.3	52.2 ± 0.4	55.3 ± 0.4	57.4 ± 0.4	57.1 ± 0.3	49.0 ± 0.7	58.5 ± 1.0	1.9
2000	62.5 ± 0.7	59.4 ± 0.7	59.0 ± 0.2	66.0 ± 0.3	65.6 ± 0.5	60.6 ± 0.3	59.6 ± 0.3	67.0 ± 0.7	1.5
5000	67.8 ± 0.5	67.4 ± 0.4	66.9 ± 0.1	71.1 ± 0.2	72.0 ± 0.1	71.4 ± 0.3	66.3 ± 0.1	72.3 ± 0.6	0.5
MNLI-mm									
100	38.8 ± 0.7	36.4 ± 0.4	36.8 ± 1.2	37.2 ± 1.1	35.2 ± 0.7	36.8 ± 1.8	34.1 ± 0.7	39.6 ± 2.1	1.8
200	39.5 ± 0.5	39.4 ± 1.2	38.6 ± 1.2	41.0 ± 0.2	41.5 ± 0.7	39.3 ± 0.8	35.7 ± 0.2	43.2 ± 1.8	4.0
500	46.4 ± 0.4	47.6 ± 1.1	40.7 ± 1.3	49.0 ± 2.2	49.5 ± 2.0	46.4 ± 1.4	45.9 ± 2.1	54.9 ± 1.0	11.1
1000	53.6 ± 0.6	55.4 ± 0.3	52.2 ± 0.4	54.8 ± 0.3	56.1 ± 0.2	48.6 ± 0.4	50.1 ± 0.3	61.5 ± 0.6	9.6
2000	64.3 ± 0.8	62.4 ± 0.3	61.0 ± 0.3	67.1 ± 0.2	64.5 ± 0.5	60.9 ± 0.3	58.6 ± 0.2	67.8 ± 0.2	1.1
5000	69.8 ± 0.3	70.0 ± 0.1	68.9 ± 0.2	71.5 ± 0.4	71.7 ± 0.3	68.3 ± 0.2	69.2 ± 0.3	71.9 ± 0.1	0.4

Table 1: Results on MNLI-m/mm with varying data sizes for BERT-Base. "Raw" indicates the original language model. The leftmost column shows the number of examples used in training. *Impr.(b)* indicates the improvement over the best baseline

Runtime	SST-2	QNLI	MNLI
IMP	523	816	3139
raw	18	28	105
PFit	18	29	109

Table 2: Comparison of per-epoch runtime (in minutes) among the original model, IMP (Allen et al. 2019), and PFit on RoBERTa-Large (Liu et al. 2019) with batch size of 16.

We also observe that the most significant improvement (Tab. 3) occurs with smaller datasets (100-200 examples) on SST-2, compared with medium-sized datasets (500-2000 examples) on QNLI and MNLI. This is potentially because sentiment analysis has a larger reliance on lower-level phrasal information and surface features, which are encoded in the pretrained word representations (Jawahar, Sagot, and Seddah 2019) and can be directly leveraged in the construction of prototypes. Our proposed mixture prototype initialization can effectively leverage the rich phrasal and sentiment clues and assist PFit towards achieving a globally optimal solution. For more complex tasks such as textual entailment, larger data sizes are needed for modeling higher-level complex semantic features and class relationships among examples.

Scenarios for Applying PFit We observe that PFit achieves great performance improvements for datasets with varying data sizes under varying-resource settings. Meanwhile, due to the higher variance it incurs on small-scale datasets (≤ 100 data points), PFit is less advised for smaller datasets. In those cases, models with simpler inductive biases like ProtoNet (Snell, Swersky, and Zemel 2017) may achieve slightly worse but more stable performances with less parameters. On larger datasets, prototypical models with flexible capacity (Allen et al. 2019; Kulis and Jordan 2012) generally outperform methods with fixed capacity (Snell, Swersky, and Zemel 2017; Ming et al. 2019).

Complexity & Runtime

The time complexity for PFit is $O(N|P_{max}| + NM)$. This consists of $O(N|P_{max}|)$ for prototype creation and $O(NM)$ for prototype simplification. While prototypical methods like IMP can achieve satisfying performances, they may incur expensive computational costs. IMP takes $O(N^2|P_{max}|)$ due to recomputing a collection of prototype vectors at the end of each training step. For most data sizes, IMP (Allen et al. 2019) outperforms other baselines and sometimes achieves comparable results as PFit. However, IMP is computationally inefficient for a feature-learning setting as it requires constant re-estimation of the prototypes. As shown in Tab. 2, fine-tuning IMP requires 523 and 816 minutes per epoch on SST-2 and QNLI, respectively. In contrast, PFit takes 18 and 29 minutes. Compared with the raw model, PFit achieves significant improvement over the original language model with only a marginal runtime increase.

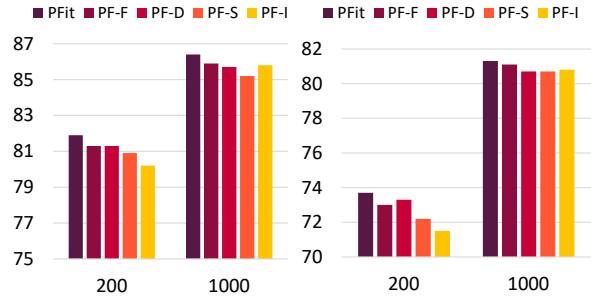

Figure 2: Results of variations of *PFit* model on SST-2 (Left) and QNLI (Right) with 1000 data points.

Ablation Study

To demonstrate the effectiveness of each major component, we evaluate 4 variations of PFit: 1) PF-I removes mixture prototype initialization and randomly initializes $\mathbf{p}_k$; 2) PF-S disables *adaptive prototype simplification*; 3) PF-D applies no diversity regularization $\mathcal{L}_{div}$; 4) PF-F uses a fixed threshold for $\mathcal{L}_{div}$. We observe that mixture prototype initialization benefits the training under low-resource settings. The performance drops most significantly when we disabled *adaptive prototype simplification*. We conjecture that some prototypes were created from a minority of out-of-distribution samples, which may overfit older training data but fail to generalize to new data.

SST-2	DPMeans	ProtoNet	ProSeNet	IMP	BSS	ViBERT	Raw	PFit	Impr. (b) %
100	69.0 ± 0.7	74.9 ± 1.2	72.8 ± 0.9	74.7 ± 0.5	66.4 ± 0.7	67.5 ± 1.1	64.7 ± 0.9	78.7 ± 3.2	5.1
200	69.6 ± 2.5	72.9 ± 0.6	75.5 ± 1.2	80.0 ± 1.3	68.6 ± 0.4	68.7 ± 0.6	67.1 ± 1.2	81.9 ± 1.1	2.4
500	73.9 ± 0.6	80.7 ± 0.9	80.6 ± 0.6	82.3 ± 0.6	69.8 ± 0.7	74.4 ± 0.1	77.7 ± 0.6	84.6 ± 0.9	2.8
1000	84.6 ± 0.6	84.8± 0.4	85.2 ± 0.6	84.8 ± 0.7	84.3 ± 0.6	85.6 ± 0.3	86.0 ± 0.3	86.4 ± 0.6	1.0
2000	87.2 ± 0.3	85.2 ± 0.4	86.8 ± 0.5	87.5 ± 0.4	87.9 ± 0.4	87.4 ± 0.1	87.9 ± 0.6	88.5 ± 0.2	0.6
5000	88.3 ± 0.5	87.0 ± 0.3	88.2 ± 0.1	89.6 ± 0.2	89.4 ± 0.1	89.1 ± 0.3	89.6 ± 0.2	90.7 ± 0.2	1.2
QNLI									
100	56.0 ± 2.7	64.8 ± 0.4	63.2 ± 1.7	61.2 ± 2.0	59.0 ± 1.3	60.3 ± 1.2	56.4 ± 1.4	67.0 ± 2.8	3.4
200	67.2 ± 0.4	67.4 ± 1.3	69.0 ± 1.5	71.0 ± 1.2	69.2 ± 0.5	69.2 ± 1.3	62.1 ± 1.6	73.7 ± 1.0	3.7
500	71.1 ± 0.2	76.6 ± 0.5	77.0 ± 1.1	77.0 ± 0.6	75.8 ± 0.5	72.7 ± 0.1	68.4 ± 0.2	78.1 ± 0.3	1.3
1000	78.6 ± 0.1	80.2 ± 0.4	78.3 ± 0.5	78.0 ± 0.2	77.6 ± 0.3	75.9 ± 0.2	73.3 ± 0.3	81.3 ± 0.2	3.8
2000	81.6 ± 0.2	80.7 ± 0.1	81.4 ± 0.1	81.3 ± 0.2	82.1 ± 0.2	80.9 ± 0.2	77.6 ± 0.1	82.6 ± 0.3	0.6
5000	82.7 ± 0.1	82.9 ± 0.2	84.3 ± 0.1	84.3 ± 0.3	84.5 ± 0.1	84.4 ± 0.2	82.5 ± 0.1	84.9 ± 0.3	0.4

Table 3: Results on QNLI/SST-2 with varying data sizes for BERT-Base.

Prototype A	Prototype B	Prototype C
I had to look away - this was god **awful**.	It's of the quality of a lesser harrison ford movie	Bad.
I **have to say** the star and director are the **big problems** here.	I can take infantile humor ... **but** this is the sort of infantile that makes you wonder about changing the director and ...	Very bad.
Eventually, every idea in this film is flushed down the **latrine** of heroism.	**I am sorry that** i was unable to get the full brunt of the comedy .	Yes, dull ...

Table 4: The top 3 closest examples of each learned prototype on SST-2. The closest examples all have ground-truth labels of "negative." However, each prototype represents a group of semantically similar negative samples.

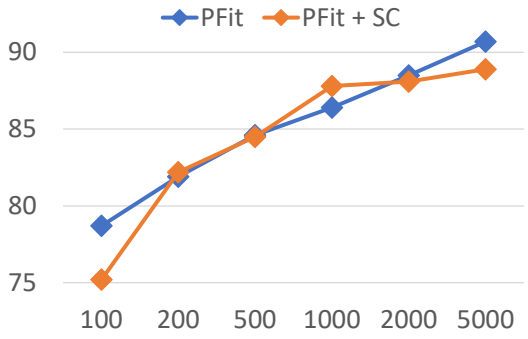

Figure 3: Results for the original *PFit* model and *PFit* with skip connection (PFit-SC).

Interpreting the Prototypes

Besides performance improvements, prototypes can bring the extra advantage of explainability. To interpret the prototype vectors, we project each prototype to its closest 10 examples. We observe that 1) prototypes manifest inter-class distinction as the classes are discriminable from the prototypes; and 2) prototypes demonstrate intra-class distinction as there are clustering emergent within the prototypes.

Inter-class Distinctions If we project each prototype onto its nearest examples, each prototype shows strong polarity towards one specific class. Here, for simplicity, we refer to prototypes that predict positive class as "positive prototypes" and those predicting negative as "negative prototypes". On SST-2, the average percentage of positive examples is 91% for positive prototypes, and that of negative examples for negative prototypes is 87%.

Intra-class Distinctions Tab. 4 is an example for the binary sentiment classification dataset SST-2. The top 3 closest examples to each prototype are shown. The closest examples all have ground-truth labels of "negative" However, each prototype represents a group of negative samples that are semantically similar. Examples of prototype A generally express stronger emotions ("awful", "big problem", "latrine"). Examples of B express weaker emotions. Examples of C mainly focus on succinct negative expressions. Methods such as ProtoNet (Snell, Swersky, and Zemel 2017) only use one prototype for each class, which may not sufficiently characterize such intra-class distinctions.

We experimented with connecting non-contextualized BERT (Devlin et al. 2019) embeddings with contextualized representations using a residual network, leading to performance improvements on SST-2 with 200 and 1000 examples, and decreases in other settings. We believe that merging the contextualized embeddings with non-contextualized signals without sacrificing performances can potentially increase the explainability of our prototypical framework. We will explore this method comprehensively in future works.

Conclusion

We propose Prototypical Fine-tuning (PFit), a prototypical framework for effectively fine-tuning pretrained language models. Extensive experiments across various datasets show that our work offers significant performance improvements for language models under varying data sizes, as well as explainability.

References

Ackerman, M.; and Dasgupta, S. 2014. Incremental clustering: The case for extra clusters. In *NIPS*.

Allen, K.; Shelhamer, E.; Shin, H.; and Tenenbaum, J. 2019. Infinite mixture prototypes for few-shot learning. In *ICML*, 232–241. PMLR.

Artetxe, M.; Labaka, G.; Agirre, E.; and Cho, K. 2018. Unsupervised neural machine translation. In *ICLR*.

Baxter, J. 2000. A model of inductive bias learning. *JAIR*, 12: 149–198.

Belinkov, Y.; Henderson, J.; et al. 2021. Variational Information Bottleneck for Effective Low-Resource Fine-Tuning. In *ICLR*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. In *NIPS*.

Chen, C.; Li, O.; Tao, D.; Barnett, A.; Rudin, C.; and Su, J. K. 2019. This looks like that: deep learning for interpretable image recognition. In *NIPS*, volume 32.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 4171–4186.

Fadaee, M.; Bisazza, A.; and Monz, C. 2017. Data Augmentation for Low-Resource Neural Machine Translation. In *ACL*, 567–573.

Gao, H.; Shou, Z.; Zareian, A.; Zhang, H.; and Chang, S.-F. 2018. Low-shot learning via covariance-preserving adversarial augmentation networks. In *NIPS*, volume 31.

Gao, T.; Han, X.; Liu, Z.; and Sun, M. 2019. Hybrid attention-based prototypical networks for noisy few-shot relation classification. In *AAAI*, 6407–6414.

Goldberger, J.; Hinton, G. E.; Roweis, S.; and Salakhutdinov, R. R. 2004. Neighbourhood components analysis. In *NIPS*.

Gunel, B.; Du, J.; Conneau, A.; and Stoyanov, V. 2020. Supervised Contrastive Learning for Pre-trained Language Model Fine-tuning. In *ICLR*.

Hao, Y.; Dong, L.; Wei, F.; and Xu, K. 2020. Investigating learning dynamics of BERT fine-tuning. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, 87–92.

Hjort, N. L.; Holmes, C.; Müller, P.; and Walker, S. G. 2010. *Bayesian nonparametrics*, volume 28. Cambridge University Press.

Jawahar, G.; Sagot, B.; and Seddah, D. 2019. What does BERT learn about the structure of language? In *ACL*.

Jin, S.; Wiseman, S.; Stratos, K.; and Livescu, K. 2020. Discrete Latent Variable Representations for Low-Resource Text Classification. In *ACL*, 4831–4842.

Jin, Y.; Wang, X.; Yang, R.; Sun, Y.; Wang, W.; Liao, H.; and Xie, X. 2022. Towards Fine-Grained Reasoning for Fake News Detection. In *AAAI*.

Kim, B.; Rudin, C.; and Shah, J. A. 2014. The bayesian case model: A generative approach for case-based reasoning and prototype classification. *NIPS*, 27.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Kolodner, J. L. 1992. An introduction to case-based reasoning. *Artificial intelligence review*, 6(1): 3–34.

Kulis, B.; and Jordan, M. I. 2012. Revisiting k-means: new algorithms via Bayesian nonparametrics. In *ICML*, 1131–1138.

Li, O.; Liu, H.; Chen, C.; and Rudin, C. 2018. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *AAAI*.

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

McCoy, T.; Pavlick, E.; and Linzen, T. 2019. Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference. In *ACL*, 3428–3448.

Mensink, T.; Verbeek, J.; Perronnin, F.; and Csurka, G. 2013. Distance-based image classification: Generalizing to new classes at near-zero cost. *PAMI*, 35(11): 2624–2637.

Mettes, P.; van der Pol, E.; and Snoek, C. 2019. Hyperspherical prototype networks. *NIPS*, 32.

Ming, Y.; Xu, P.; Qu, H.; and Ren, L. 2019. Interpretable and steerable sequence learning via prototypes. In *KDD*, 903–913.

Oh Song, H.; Xiang, Y.; Jegelka, S.; and Savarese, S. 2016. Deep metric learning via lifted structured feature embedding. In *CVPR*, 4004–4012.

Oreshkin, B.; Rodríguez López, P.; and Lacoste, A. 2018. Tadam: Task dependent adaptive metric for improved few-shot learning. *NIPS*, 31.

Perez, E.; Kiela, D.; and Cho, K. 2021. True few-shot learning with language models. *NIPS*, 34.

Rasmussen, C. 1999. The infinite Gaussian mixture model. *NIPS*, 12.

Ren, M.; Triantafillou, E.; Ravi, S.; Snell, J.; Swersky, K.; Tenenbaum, J. B.; Larochelle, H.; and Zemel, R. S. 2018. Meta-Learning for Semi-Supervised Few-Shot Classification. In *ICLR*.

Salamon, J.; and Bello, J. P. 2017. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal processing letters*, 24(3): 279–283.

Shao, J.; Huang, F.; Yang, Q.; and Luo, G. 2017. Robust prototype-based learning on data streams. *TKDE*, 30(5): 978–991.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. *NIPS*, 30.

Tenney, I.; Das, D.; and Pavlick, E. 2019. BERT Rediscovers the Classical NLP Pipeline. In *ACL*, 4593–4601.

Tsimpoukelli, M.; Menick, J. L.; Cabi, S.; Eslami, S.; Vinyals, O.; and Hill, F. 2021. Multimodal few-shot learning with frozen language models. *NIPS*, 34: 200–212.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *NIPS*.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Wang, C.; and Blei, D. 2009. Variational inference for the nested Chinese restaurant process. *NIPS*, 22.

Wang, J.; Wang, K.-C.; Rudzicz, F.; and Brudno, M. 2021. Grad2Task: Improved Few-shot Text Classification Using Gradients for Task Representation. *NIPS*, 34.

Wei, C.; Xie, S. M.; and Ma, T. 2021. Why do pretrained language models help in downstream tasks? an analysis of head and prompt tuning. *NIPS*, 34: 16158–16170.

Yang, H.-M.; Zhang, X.-Y.; Yin, F.; and Liu, C.-L. 2018. Robust classification with convolutional prototype learning. In *CVPR*, 3474–3482.

Yang, R.; Wang, X.; Jin, Y.; Li, C.; Lian, J.; and Xie, X. 2022. Reinforcement Subgraph Reasoning for Fake News Detection. In *KDD*.

Zaken, E. B.; Ravfogel, S.; and Goldberg, Y. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.