Reordering Examples Helps during Priming-based Few-Shot Learning

Sawan Kumar
Indian Institute of Science, Bangalore
sawankumar@iisc.ac.in

Partha Talukdar
Indian Institute of Science, Bangalore
ppt@iisc.ac.in

Abstract

The ability to learn from limited data, or few-shot learning, is a desirable and often critical requirement for NLP systems. While many existing methods do poorly at learning from a handful of examples, large pretrained language models have recently been shown to be efficient few-shot learners. One approach to few-shot learning, which does not require finetuning of model parameters, is to augment the language model's input with priming text which is typically constructed using task specific descriptions and examples. In this work, we further explore priming-based few-shot learning, with focus on using examples as prompts. We show that presenting examples in the right order is key for generalization. We introduce PERO (Prompting with Examples in the Right Order), where we formulate few-shot learning as search over the set of permutations of the training examples. We show that PERO can learn to generalize efficiently using as few as 10 examples, in contrast to existing approaches. While the newline token is a natural choice for separating the examples in the prompt, we show that learning a new separator token can potentially provide further gains in performance. We demonstrate the effectiveness of the proposed method on the tasks of sentiment classification, natural language inference and fact retrieval. Finally, we analyze the learned prompts to reveal novel insights, including the idea that two training examples in the right order alone can provide competitive performance for sentiment classification and natural language inference.

1 Introduction

The ability to learn from a few examples, or few-shot learning, as generally understood to be possessed by humans, is a desirable property for Natural Language Processing (NLP) systems as well. It is critical in scenarios where collecting large amounts of data is expensive. It is also important to enable a personalized Artificial Intelligence (AI) experience, where a single user is expected to use an AI agent to perform a task demonstrated through a handful of examples.[1]

Pretrained language models (Devlin et al., 2019; Liu et al., 2019; Raffel et al., 2020) have recently been shown to be exceedingly good at several benchmark NLP tasks (Wang et al., 2018, 2019). Traditionally the parameters of these language models have been finetuned on task specific datasets to achieve the aforementioned performance gains, often requiring large amounts of data. Brown et al. (2020) show that large pretrained language models (GPT3) are also efficient few-shot learners. Few-shot learning is achieved using task descriptions and labeled examples as prompts. Remarkably, with this priming-based approach and without needing any parameter updates, GPT3 often performs comparable to traditional finetuning-based supervised systems which use much larger datasets. One could argue that the task performance achieved in the priming-based approach measures what the pretrained language model has already learned. Shin et al. (2020), operating in the same setting, use automatically generated prompts to measure task specific knowledge in a pretrained language model.

In this work, we further explore priming-based few-shot learning, while focusing on using examples as prompts. The training objective for a language model is typically the prediction of a token given a context. There is no clear incentive to treat a sequence of sentences in the context as equal and conveying examples of a concept. As a result, one could expect certain order of examples when used as a prompt to be more favorable at providing task

[1] See the Introduction section of Brown et al. (2020) for a discussion on further difficulties, relevant to the setting we consider in this work, regarding the need of a large dataset for every new task.

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 4507–4518
August 1–6, 2021.

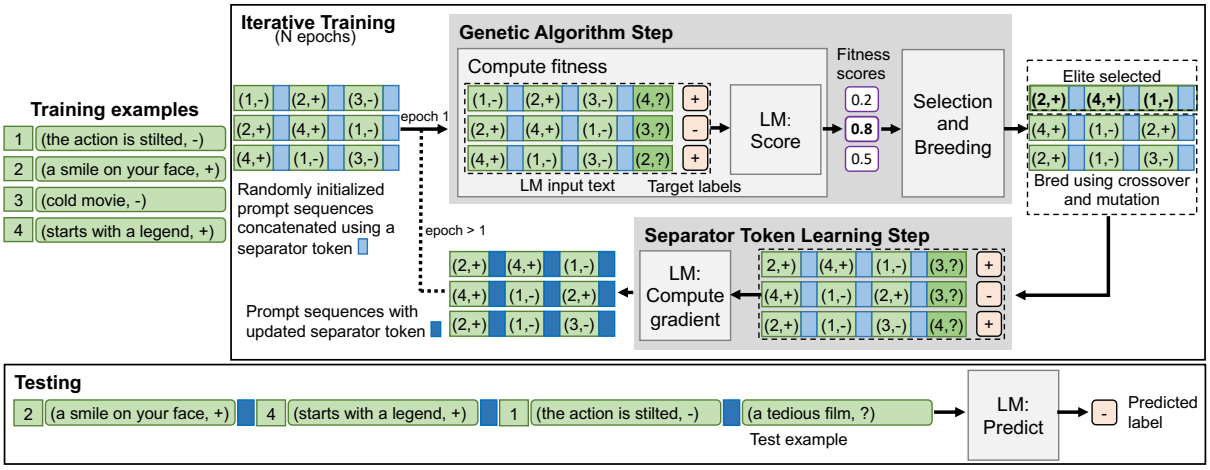

Figure 1: *Overview of PERO*: Given a set of training examples, PERO searches over permutations of training examples using a genetic algorithm (Section 4.2), and optionally also learns a separator token (Section 4.3) which is used to concatenate the training examples. Briefly, starting from a set of randomly initialized permutations, the genetic algorithm step computes the fitness of each permutation for making predictions using a pretrained LM. These fitness scores are then used for selection and subsequent breeding of new permutations using biologically inspired operations of mutation and crossover. The separator token learning step uses the updated set of permutations and uses gradient updates to improve the separator token. The two steps are performed iteratively for a fixed number of epochs and the best permutation and separator token are selected using a validation set. Please see Section 4 for details.

specific cues.

We propose PERO[2] (Prompting with Examples in the Right Order), where we formulate the problem of few-shot learning as search over permutations of training examples. We find that choosing the right permutation is key to getting good task performance. In PERO, we use a genetic algorithm (Mitchell, 1998) to search over possible permutations of training examples. The selected examples are used for prompting publicly available pretrained language models (Devlin et al., 2019; Liu et al., 2019). We find that with as few as 10 examples, PERO can learn to generalize efficiently, in contrast to existing approaches. When concatenating examples to use as a prompt, the newline token is a natural choice as a separator token. We show that using a learned separator token can potentially provide further gains in performance. We evaluate the performance of PERO on the tasks of sentiment analysis, Natural Language Inference (NLI) and fact retrieval.

Finally, our analysis of the learned prompts (Section 5.5) leads to novel insights about few-shot learning using textual prompts. For instance, using only two examples, repeated and ordered using a learned label pattern, can provide performance comparable to and even exceeding the performance of existing few-shot baselines which use thousands of examples.

In summary, we make the following contributions:

1. We propose PERO, where we formulate the problem of few-shot learning as search over permutations of training examples, and optionally a separator token. As we don't update the parameters of the underlying language model, PERO serves as a probe for measuring task specific knowledge in pretrained language models.

2. We demonstrate the effectiveness of PERO over a recent baseline on the tasks of sentiment analysis, NLI and fact retrieval.

3. We analyze the learned prompts and provide novel insights about textual prompts that can lead to good task performance in the low-data regime. In particular, we provide an effective recipe for one-shot learning.

We have released the source code of PERO to aid reproducibility of the results.

[2]PERO source code is available at https://github.com/SawanKumar28/pero

Task	Template	Examples
Sentiment Classification	[Example Text] Answer: [Label Text]	(1) goes to absurd length Answer: false (2) A girl in white is dancing Answer: true
NLI	"[Premise Text]" implies "[Hypothesis text]" Answer: [Label Text]	(1) "Men are sawing logs" implies "Men are cutting wood" Answer: true (2) "There is no girl in white dancing" implies "A girl in white is dancing" Answer: false
Fact retrieval	[Subj] is located in [Obj] [Subj] is a subclass of [Obj]	(1) Directors Lounge is located in Berlin (2) gingerbread is a subclass of cookie

Table 1: *Formatting used to create textual inputs* for the tasks considered in this work. For sentiment classification, positive and negative sentiments correspond to the label text of *true* and *false* respectively. For NLI, entailment and contradiction labels correspond to the label text of true and false respectively.

2 Related Work

Pretrained Language Models using a transformer architecture (Vaswani et al., 2017) on large unsupervised corpora have recently been found to be efficient at learning downstream tasks, providing significant gains over existing standalone supervised systems, on a variety of NLP tasks (Wang et al., 2018, 2019). There have been two major approaches to learning language models: causal language models (CLM) and masked language models (MLM). CLMs (Radford et al., 2018, 2019; Brown et al., 2020) are typically trained by requiring a language model to predict the next token given a textual context. Masked language models (Devlin et al., 2019; Liu et al., 2019) on the other hand are trained by masking out a certain number of tokens in a textual context and requiring the language model to predict the masked out tokens. Typically, the parameters of the language model are then fine-tuned using task-specific training examples. For our experiments, we leverage publicly available pretrained masked language models (Devlin et al., 2019; Liu et al., 2019).

Few-shot learning using language models is a desirable and perhaps even an expected property of large pretrained language models, given the large amounts of data they are typically trained with. Brown et al. (2020) show that scaling up language models leads to improved few-shot learning, with their best model, GPT3, being able to achieve performance comparable to existing supervised systems, while using much fewer examples. Zero-shot and few-shot learning are achieved without needing parameter updates to the model but instead by prompting the language model with task specific description and task specific examples. In this work, we study the impact of the order in which examples are presented in a prompt and show that searching over them can lead to significant gains in few-shot performance, without needing updates to the model parameters.[3]

Measuring task performance of language models without any parameter updates can be seen as a measure of the knowledge (either descriptive, or procedural) that is already contained in the pretrained language model.

Probing knowledge contained in language models has been of interest, given the success of these models. Probing methods rely on creating cloze-style manual prompts (Petroni et al., 2019), or mining efficient natural language prompts (Jiang et al., 2020). Shin et al. (2020) rely on training examples to learn trigger tokens which when used as a prompt demonstrate the ability of language models to do sentiment analysis and NLI along with knowledge based completion, without needing any parameter updates. The learned trigger tokens however aren't very meaningful leading to difficulty in interpreting these results. In this work, we instead focus on using natural language training examples as prompts. While being more interpretable, the prompts used in this work lead to significant gains in performance in the low-data regime.

3 Background: Genetic Algorithm

A genetic algorithm (Mitchell, 1998) is a search heuristic inspired by the biological process of natural selection. Briefly, it evolves a population of candidate solutions towards increasing fitness to an objective through biologically inspired operations such as selection, crossover and mutation. We now describe the key terminology:

[3]Note that the scope of this work is distinct from meta-learning approaches (Hospedales et al., 2020), where the goal is improve the learning algorithm using several learning episodes. In contrast, we only assume a pretrained language model and a few examples of the concept we are interested in.

Individual A single candidate solution, c, usually represented through a binary code but extensible to other types of codes. Generally, we will let c be denoted by the sequence of k integers $c = (s^1 s^2 ... s^k)$.

Population A set of individuals of size N_P, $P = \{c_i, i \in [N_P]\}$.

Fitness The measure of goodness for an individual for the task, $F(c_i)$.

Selection An operator to select fit individuals in a population which will be used to generate new individuals, through crossover and mutation. Better fitness leads to higher likelihood of selection.

Crossover An operator which typically takes two individuals c_1 and c_2 as inputs to produce new individuals d_1 and d_2, by combining subsequences from the two inputs. For example, consider two input sequences:

$$c_1 = (c_1(1)c_1(2)c_1(3)c_1(4))$$
$$c_2 = (c_2(1)c_2(2)c_2(3)c_2(4))$$

A single point crossover after the second position would lead to the individuals:

$$d_1 = (c_1(1)c_1(2)c_2(3)c_2(4))$$
$$d_2 = (c_2(1)c_2(2)c_1(3)c_1(4))$$

Mutation An operator which randomly flips some elements in an input sequence. For example, with input $c = (c(1)c(2)c(3)c(4))$, a typical mutation operation would lead to the output $d = (c(1)c(2)c'(3)c(4))$, where $c'(3) \neq c(3)$. Usually, each position is randomly altered with a mutation probability p_m.

We now present the sketch of a typical genetic algorithm:

1. Initialize a set of individuals to form a population $P = \{c_i, i \in [N_P]\}$. Repeat the following steps for N_{epochs} iterations.
2. Compute fitness of each individual in the population, $F(c_i), i \in [N_P]$.
3. Using the computed fitness, select individuals which will be used to breed the next generation.
4. With pairs of selected individuals, generate new individuals using the crossover operation.
5. Mutate the generated individuals using the mutation operator, to create a new population P'.
6. Set $P = P'$ and go to step 2.

4 PERO: Proposed Method

The overall architecture employed in PERO is shown in Figure 1. We introduce the notation in Section 4.1. We discuss how we employ a genetic algorithm to search over permutations of training examples in Section 4.2. We then discuss how we augment the search heuristic to learn a task specific separator token in Section 4.3.

4.1 Notation and Input Format

For both classification and knowledge base completion tasks, we denote a textual task input by x and the gold label as y. We denote the pretrained masked language model with the operator L, which takes a sequence of input tokens to output a sequence of the same length containing token probabilities over the token vocabulary. With tokens $(t_1 t_2 ... t_n)$, $L((t_1 t_2 ... t_n)) = (p_1 p_2 ... p_n)$, where p_i denotes a vector of probabilities over all tokens in the vocabulary.

For all our experiments, the input to the language model is formatted with exactly one mask token.[4] For brevity, we denote by L_{Mask} the operator which outputs the token probability at the mask token position.

The training data is denoted by the set of examples $(x_i, y_i), i \in [N_{\text{train}}]$. We denote a permutation, or an ordered subset of size k of the training data, by $c = (c(1)c(2)...c(k))$, where $c(j) \in [N_{\text{train}}]$.

For all tasks, we create an input text sequence by concatenating k examples using a permutation c of training examples, along with a test example x_{test}: "Format($x_{c(1)}$,$y_{c(1)}$) <Separator> Format($x_{c(2)}$,$y_{c(2)}$) .. <Separator> Format($x_{c(k)}$,$y_{c(k)}$) <Separator> Format(x_{test},mask)", where Format(,) formats the example text and label for a task, and <Separator> is either the new line character, or is learned as described in Section 4.3. The formatting details are provided in Table 1. We attempt to

[4] The mask token we use is the same as the one employed in pretraining.

use task agnostic formats and textual labels for classification tasks to the extent possible.

4.2 Genetic Algorithm: Search over Permutations of Examples

We employ a genetic algorithm for searching over permutations of training examples (see Section 3 for a brief introduction to genetic algorithms). We present the overall architecture in Figure 1.

Here, we detail how the various components and operators of a genetic algorithm are defined for searching over permutations of examples:

Individual An individual is defined as a vector of unique training example indices $c = (c(1)c(2)...c(k))$, where $c(j) \in [N_{\text{train}}]$.

Population A set of individuals.

Fitness For a given permutation of training example indices, fitness is defined as the average cross entropy loss over training examples when evaluated as in Figure 1. The cross entropy loss is computed over the set of possible labels for classifications tasks, and over all tokens in the vocabulary for knowledge base completion tasks.

Note that during search, a training example may occur both in the prompt and as well as the test example. This is generally not a problem as we are not finetuning the model and do not run the risk of learning to copy. When also training the separator token (Section 4.3), we ensure that the test example doesn't occur in the prompt by dropping it from the prompt if required.

Selection For selection, we use elitism, i.e., at each generation of individuals, we retain a certain percentage (elite ratio) of top performing individuals without any modifications. The rest of the population is created through crossover and mutation over a percentage (selection size) of top performing individuals.

Crossover We perform a single point crossover, while ensuring that the resulting individuals contain unique indices. Given two parents c_1 and c_2, first a random number j is sampled in the range $[k]$, the length of the individuals, to use as the crossover point. We define an operator $\text{First}_s(v, v')$ which selects the first s elements in vector v which do not occur in vector v'. Similarly, $\text{Last}_s(v, v')$ picks the last s elements in v which do not occur in vector v'. Denoting the subvector $c(i)c(i+1)...c(j)$ by $c^{i:j}$, four new individuals are then created:

$$d_1 = (c_1^{1:j}\text{Last}_{k-j}(c_2, c_1^{1:j}))$$
$$d_2 = (c_2^{1:j}\text{Last}_{k-j}(c_1, c_2^{1:j}))$$
$$d_3 = (\text{First}_j(c_2, c_1^{j+1:k})c_1^{j+1:k})$$
$$d_4 = (\text{First}_j(c_1, c_2^{j+1:k})c_2^{j+1:k})$$

This modification over a straightforward crossover ensures that the resulting individuals contain unique indices.

Mutation We perform mutation on an input candidate by changing each position with a mutation probability p_m. When changed, an index is replaced by a random choice from the other training examples. If the new index is already present in the input candidate, the value at that index is swapped with the selected index.

The Genetic algorithm is run for N_{epochs} (see Section 3 for the training flow). A validation set of the same size as the train set was used to select from the best performing individuals in each epoch.

4.3 Separator Token Learning

In addition to the search over permutations of training examples as described in the previous section, we optionally learn a separator token to concatenate the examples (see Figure 1).

We initialize a token embedding parameter with the token embedding of the newline character. At the end of each epoch of the genetic algorithm, we use gradient updates to estimate the token embedding. The training set is created using the individuals (prompts) in the population in the current generation, and replacing the answer of the final example with the mask token. Gradient updates are then done by requiring the model to predict the correct answer.

5 Experiments

In this section, we aim to answer the following questions:

Q1 How does PERO compare with existing approaches on task performance? (Section 5.3)

Q2 How do the components of PERO, namely genetic algorithm and separator token learning affect task performance? (Section 5.4)

Training	Prompt Type	P@1
Manual	LAMA (Petroni et al., 2019)	31.1^
Full supervision	LPAQA(top1) (Jiang et al., 2020)	34.1^
	Autoprompt (Shin et al., 2020)	43.3^
	PERO	46.6
10 examples	Autoprompt (Shin et al., 2020)	18.9
	PERO	40.3

Table 2: *Summary of fact retrieval experiments:* Precision @1 results on test sets are presented. ^indicates replicated numbers. PERO improves over the baselines when using all training data (up to 1000 examples) and provides significant gains when using limited training data. This indicates that PERO is more efficient at eliciting knowledge from language models. Please see Section 5.3 for details and Appendix A.2.2 for more detailed results.

Q3 What aspects of PERO are important for getting good performance? (Section 5.5)

The experimental setup is described in Section 5.2, and the datasets are described in Section 5.1.

5.1 Datasets

Sentiment Classification: We use SST2 (Socher et al., 2013), a binary sentiment classification task. The training data contains 67350 training, 873 validation and 1822 test examples.

NLI: We use label-balanced 2-class NLI dataset created by Shin et al. (2020) using the SICK-E dataset (Marelli et al., 2014). This dataset has 1289 training, 143 validation and 1427 test examples.

Fact Retrieval: We use the train, validation, and test splits created by Shin et al. (2020) (referred to as 'original' in the paper) for 41 relations. For our experiments, we use the manual prompts created by Petroni et al. (2019). Please see Appendix A.2.2 for relation wise prompts and training statistics.

5.2 Experimental Setup

Number of training examples: For most of our experiments, we limit to a total of 10 training examples. We chose this number as prior work (Shin et al., 2020) faced difficulty in enabling predictions using only 10 training examples, usually performing close to random prediction. We create 5 sets of size 10, chosen successively from the first 50 training examples, and report on average task performance. Although our focus is few-shot learning in the low data regime, we also present results with more examples (the first 100 and the first 1000 examples) for reference. For model selection, we use a label-balanced validation set (chosen from the beginning of the corresponding validation set) of the same size as the training data. In all cases, and irrespective of the number of training examples, we keep the prompt size fixed to 10 examples.

Pretrained LM: We use RoBERTa-large (Liu et al., 2019) for all our experiments except for the fact retrieval task where we use the bert-large-cased model (Devlin et al., 2019) as this model has been shown to work better for the task (Shin et al., 2020). RoBERTa-large has 24 layers, with 16 attention heads and a hidden size of 1024 (355M parameters). Bert-large-cased uses the same architecture as RoBERTa-large. We use the implementation of transformer architectures provided by Wolf et al. (2020). We use the </s> token as the default separator token. When learning a new separator token, we initialize the token embedding by the token embedding of </s> token, and finetune the embedding as discussed in Section 4.3.

Genetic algorithm: We run the genetic algorithm for 100 epochs for classification tasks and 30 epochs for fact retrieval tasks. The population size was fixed to 100 and the mutation probability was set to 0.1. Elite ratio was set to 0.1, while the selection size was fixed to 25. When training a separator token embedding, the maximum number of training epochs for learning the embedding was set to 10 for classification tasks and 5 for fact retrieval tasks. Gradient updates were performed using the AdamW optimizer (Loshchilov and Hutter, 2018) with a learning rate of $1e-4$.

Baselines: We use Autoprompt (Shin et al., 2020) and the traditional finetuning approach as few-shot baselines. Please see Appendix A.1 for hyperparameter details.

5.3 Overall Results

In this section, we present the few-shot learning capability of PERO. For reference, we also report results when using more data.

We present fact retrieval results (Precision@1 scores) in Table 2. Relation wise results are provided in Appendix A.2.2. When using all training data, PERO improves the overall P@1, and is competitive or outperforms Autoprompt on all relations. When using only 10 training examples, PERO provides significant gains over Autoprompt

Number of training examples		10	100	1000
Sentiment Classification	Finetune	52.5 (2.36)	90.2	93.1
	Autoprompt	52.3 (2.60)	73.5	75.1
	PERO	**91.2** (1.83)	**93.8**	**94.2**
NLI	Finetune	57.4 (10.65)	**96.1**	**98.6**
	Autoprompt	58.6 (9.08)	76.2	86.5
	PERO	**81.3** (4.99)	78.5	83.2

Table 3: *Summary of classification tasks:* Test set label accuracies for PERO are presented for the tasks of sentiment classification and NLI. When using 10 examples, we also report the standard deviation across training splits. For both tasks, PERO provides significant gains over both Autoprompt and the traditional finetuning approach, when using 10 examples. For reference, we also present results when using more training examples. For both tasks, PERO is competitive with Autoprompt with increasing data. Overall, the results indicate that PERO is capable of learning to generalize with a handful of examples, in contrast to existing approaches. Please see Section 5.3 for details. Please see Appendix A.2.1 for additional comparison between Autoprompt and PERO using 10 examples across 100 training splits.

	Sentiment Classification	NLI
PERO	91.2	81.3
PERO-Sep learning	89.3	77.5

Table 4: *Impact of separator token learning Step:* Average test set label accuracies with and without the separator token learning are presented for training sets of size 10. The results indicate that the genetic algorithm step alone provides a strong baseline, while the separator token learning step provides further gains. Please see Section 5.4 for details.

on all relations. Overall, we show through PERO that simple manual prompts can be combined in relatively straightforward ways to create stronger probes while still being interpretable.[5]

We present the label accuracies of PERO for sentiment classification and NLI in Table 3. In each case, PERO is able to generalize well when using only 10 examples, while existing approaches perform close to random guess (50%). When using more data, PERO is competitive with Autoprompt for both tasks, while finetuning does better than PERO for NLI with larger training sizes. Overall, PERO provides an efficient approach to few-shot learning with pretrained language models.

Comparison when using larger training sizes: The results in Table 3 also suggest the use of finetuning when more data is available and the use of PERO when there isn't enough data for finetuning to generalize well. The relatively low performance of PERO with more data, especially for the NLI task, could be due to the much larger search space when using more training data. Since we keep the prompt size fixed to 10 examples, the search space is $10!$ for 10 training examples and $1000!/990!$ when using 1000 examples. While a better search strategy could potentially improve PERO's performance when using more data, we leave that as an interesting future work. Note, however, that the search space complexity is determined by the number of training examples irrespective of their labels. For example, PERO improves over the baselines on the fact retrieval task (Figure 2), despite a much larger number of labels.

For reference, we provide the label accuracies when using all available training data when using PERO, Autoprompt and finetuning respectively: 95.0, 91.4 and 96.7 for sentiment classification, and 79.5, 87.3 and 99.1 for NLI. When compared to the traditional fully supervised finetuning approach, PERO performs within 94.3% while using only 0.015% of the training data for sentiment classification, and within 82.1% while using only 0.77% of the training data for NLI.

5.4 Ablation on PERO's components

In this section, we present an ablation study to understand the role that the two components of PERO, namely genetic algorithm and separator token learning steps play. We present the label accuracies for sentiment classification and NLI with and without the separator token learning step (indicated as PERO-sep learning) in Table 4. The results indicate that the permutation search using the genetic algorithm step provides large gains by itself, while the separator token learning potentially improves it.

[5] Autoprompt's learned tokens are sometimes relevant to the task but generally hard to interpret.

	Sentiment Classification	NLI
Default fitness	89.3	77.5
Inverse fitness	78.3	58.8

Table 5: *Searching for bad permutations:* We use the genetic algorithm described in Section 4 to search for bad permutations, by inverting the definition of fitness and report on test accuracy. Average results when using training sizes of 10 examples are presented and contrasted with search using default fitness. The results indicate that good permutations found by PERO are not trivial and that choosing the right permutation is necessary for generalization. Please see Section 5.5.1 for details.

5.5 Analyzing Learned Permutations

5.5.1 Do bad solutions exist which PERO learns to avoid?

With the same search strategy as discussed in Section 4, we search for potentially bad permutations, by inverting the definition of fitness. For this experiment, to focus on the role of permutations, we do not train the separator token for this experiment. We present the average test set accuracies across training splits for the best and the worst permutations in Table 5. Additionally, we also evaluate 100 random permutations for each training split. The mean (and standard deviation) test accuracy across training splits and random permutations was 85.6(9.08) for sentiment classification and 67.9(8.99) for NLI.

The results indicate that PERO's learned permutations provide significant gains over other permutations constructed using the same examples. Selecting the right permutation, therefore, is important for generalization.

5.5.2 How many examples does PERO need for good performance?

One could see a permutation learned by PERO as a combination of a label pattern (the sequence of labels corresponding to the sequence of examples) and particular examples of the respective labels. To understand the importance of the learned label pattern, we search for pairs of examples, one example for each label[6], and repeat them using the learned label pattern. The examples are selected from within the set of examples in the learned permutation. We present the accuracy with the original permutation and the best and worst accuracies when using only two training examples in Table 6. Remarkably, two examples alone, when selected well, can go a long way towards good performance.

Additionally, using the learned label pattern provides at least a 10 point improvement in accuracy when compared with a sequence without repetitions (details omitted). This indicates a potential recipe for one-shot learning which we discuss next.

[6]Learning from one example per class is usually referred to as one-shot learning.

Accuracies on SST2 dev set using only 2 examples			
Label sequence	PERO-10	Best-2	Worst-2
−−−−++++−−	91.4	91.5	81.2
−++−−−−+−−	81.1	89.5	66.5
++++−−−+++	81.4	85.5	53.9
−−−++++−−	91.5	87.4	69.3
−−−−++++−−	91.4	90.8	74.9

Table 6: *Evaluating learned label patterns with one example per label:* Best and worst accuracies obtained when using only two examples (one unique example per label) are compared with the accuracy of PERO with 10 distinct examples for the same label pattern (denoted by PERO-10). The best accuracy possible with two distinct examples is competitive with PERO-10. The results indicate that learned label patterns are useful for generalization, along with the choice of selected examples. Please see Section 5.5.2 for details.

		Sentiment Classification	NLI
Proposed 1-shot	Worst	56.2	56.3
	Best	90.6	84.5
10 examples	Finetune	52.5	57.4
	Autoprompt	52.3	58.6
	PERO	91.2	81.3

Table 7: *One-shot learning:* Best and worst test set label accuracies with one-shot learning using training example pairs obtained from the first 10 training examples are presented. The best possible accuracies with the proposed one-shot learning approach is competitive with PERO using 10 examples, while improving over finetuning and Autoprompt using 10 examples. Please Section 5.5.3 for details.

5.5.3 Can insights gained from this work lead to one-shot learning recipes?

To answer this question, we provide an example one-shot learning (one training example per class) algorithm which greedily grows a prompt sequence. In contrast to Section 4, we don't use an additional validation set to select a good prompt sequence. We update the definition of fitness to prevent it from being biased towards one class by defining it to be the minimum and not the average of the cross

SST-2	
Best (Acc: 90.6)	-ve sentiment: on the worst revenge-of-the-nerds clichés the filmmakers could dredge up +ve sentiment: demonstrates that the director of such hollywood blockbusters as patriot games can still turn out a small , personal film with an emotional wallop .
Worst (Acc: 56.2)	-ve sentiment: remains utterly satisfied to remain the same throughout +ve sentiment: of saucy

Table 8: *Example training pairs for one-shot learning* corresponding to the best and worst test set accuracies for sentiment classification. Please see Section 5.5.3 for details.

entropy loss over the training examples. This is equivalent to minimizing the negative probability of the least probable target label.

Following Section 5.5.2, we allow an example to be repeated in a prompt sequence. Setting the maximum possible length of the prompt sequence, i.e., number of (potentially repeated) examples in the prompt sequence to $l_{\max}$, the algorithm then is comprised of the following steps:

1. Initialize an empty prompt, $c = ()$
2. Create all possible prompts, P, formed by inserting exactly one example to c. If we denote the length of c as l_c and the number of labels as N_{labels}, the size of the set is given by $N_P = (l_c + 1) * N_{\text{labels}}$.
3. Compute the fitness of prompts in P.
4. Select prompt $c' \in P$ with the best fitness.
5. Set $c' = c$ and go to step 2 if $l_c < l_{\max}$.

We now discuss the results of using this one-shot learning approach over the tasks of sentiment classification and NLI. In each case, we consider the first 10 examples in the training set and create all possible training example pairs for one-shot learning, selecting one example from each class. This leads to 24 training example pairs in each case. We set the max length $l_{\max}$ to 10, and ensure that the prompt sequence is label-balanced at each step. We summarize the results in Table 7. The results indicate that the proposed algorithm is an effective approach to one-shot learning. In Table 8, we show the training examples corresponding to the best and worst cases for the task of sentiment classification. While there is indication that more representative examples (such as longer examples) are more informative and thus more useful for one-shot learning, we leave a more thorough analysis as interesting future work.

6 Conclusion

In this paper, we propose PERO, a promising approach for few-shot learning, where we formulate learning as search over permutations of training examples, and optionally a separator token. We show the effectiveness of PERO for few-shot learning on the tasks of sentiment classification, NLI and fact retrieval tasks. We demonstrate that PERO provides an interpretable and a more accurate way to probe the knowledge contained in pretrained language models. Our analysis of the learned prompts reveals novel insights and cues for further research on few-shot learning, including one-shot learning.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work is supported by the Ministry of Human Resource Development (Government of India).

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2020. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. In *International Conference on Learning Representations*.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? 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 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

A Appendix

A.1 Experimental Setup

A.1.1 Autoprompt Experiments

For Autoprompt experiments, following Shin et al. (2020), we set the number of trigger tokens to 10, number of label tokens to 3, and candidate set size to 10. Label search was run for 50 iterations and trigger token search was run for 180 iterations. Experiments were conducted on the same splits as PERO.

A.1.2 Finetuning Experiments

For the finetuning experiments, following the recommended settings for small datasets by Mosbach et al. (2020), we trained models for 20 epochs, using AdamW (Loshchilov and Hutter, 2018), with learning rate linearly increasing to $2e-5$ in the first 10% epochs and then linearly decreasing to 0. The experiments were conducted on the same splits as PERO.

A.1.3 Training Time

Training time for PERO was approximately 3 hours for each experiment in the case of classification tasks, and approximately 30 minutes for each experiment of fact retrieval tasks.

A.1.4 Computing Infrastructure

We used Nvidia's GeForce GTX 1080 Ti GPUs for all our models. Each experiment was run on a single GPU.

A.1.5 Data

The experiments were done in the evaluation framework of Shin et al. (2020) who provide instructions for downloading the corresponding data splits at `https://github.com/ucinlp/autoprompt`.

Here, we provide more details on the classification datasets used. Details on the fact retrieval data are presented in Section A.2.2.

Sentiment Classification: We used the SST-2 dataset, the binarized version of the sentiment classification dataset created by Socher et al. (2013). The training examples are constructed using movie review excerpts collected from `rottentomatoes.com` website, and labels obtained using Amazon Mechanical Turk's crowdsourcing platform. The percentage of examples labeled with positive sentiment in train, validation and test sets are 55.78%, 50.92% and 49.64% respectively. The number of examples labeled with positive sentiment in the training sets of size 10 used in the work are 4, 3, 7, 5 and 4. See Section 5.2 for selection and other details.

NLI: We use the label-balanced 2-class NLI dataset created by Shin et al. (2020) using the SICK-E dataset (Marelli et al., 2014). The dataset was created using sentences from the 8K ImageFlickr data set[7] and the SemEval 2012 STS MSRVideo Description data set[8]. Labels were obtained using Amazon Mechanical Turk's crowdsourcing platform. The number of examples labeled with entailment relation in the training sets of size 10 used in the work are 4, 3, 5, 6 and 5. See Section 5.2 for selection and other details.

Number of training examples		10
Sentiment Classification	Autoprompt	54.73 (4.72)
	PERO	**91.81** (1.87)
NLI	Autoprompt	62.31 (8.23)
	PERO	**78.61** (6.73)

Table 9: *Additional results on classification tasks:* Test set label accuracies (and standard deviation) for AutoPrompt and PERO are presented for the tasks of sentiment classification and NLI across 100 training splits of size 10.

A.2 Additional Results

A.2.1 Sentiment Classification

With the experimental setup described in Section 5, we performed additional comparison between Autoprompt and PERO by creating 100 training splits of size 10, chosen successively from the first 1000 training examples in each dataset. We report on the average (and standard deviation) test accuracy with Autoprompt and PERO in Table 9.

A.2.2 Fact Retrieval

We present relation wise training details and LAMA (Petroni et al., 2019) prompts which we used for our experiments along with the detailed relation wise test results in Table 10.

A.3 Validation Set Results

In this section, we provide the validation set results omitted from the main text.

For sentiment classification, PERO's accuracy on validation set with 10, 100 and 1000 examples respectively are 91.2%, 93.8% and 94.1%. For NLI, PERO's accuracy on validation set with 10, 100 and 1000 examples respectively are 81.3%, 78.5% and 83.2%. The validation set accuracy of PERO-Sep learning which was trained on 10 training examples was 91.2% for sentiment classification and 79.4% for NLI.

For fact retrieval. the average P@1 was 48.95 when using all training data, and 42.56 when using only 10 training examples.

[7] `http://nlp.cs.illinois.edu/HockenmaierGroup/data.html`

[8] `http://www.cs.york.ac.uk/semeval-2012/task6/index.php?id=data`

Relation, Manual Prompt (LAMA) (#train)	Full data set				Size 10 dataset	
	LAMA	LPAQA	Auto Prompt	PERO	Auto Prompt	PERO
P1001, [X] is a legal term in [Y] (1000)	70.47	72.75	82.45	84.88	71.50	79.00
P101, [X] works in the field of [Y] (864)	9.91	5.32	12.79	18.25	3.60	11.32
P103, The native language of [X] is [Y] (1000)	72.16	72.16	82.09	81.88	23.40	76.64
P106, [X] is a [Y] by profession (1000)	0.63	0	14.72	14.30	0.60	6.78
P108, [X] works for [Y] (376)	6.79	5.74	8.62	8.09	1.80	7.57
P127, [X] is owned by [Y] (548)	34.79	32.46	35.95	47.89	13.90	39.13
P1303, [X] plays [Y] (1000)	7.59	18.02	15.38	23.71	15.10	16.23
P131, [X] is located in [Y] (1000)	23.27	22.81	37.46	39.95	12.00	30.58
P136, [X] plays [Y] music (1000)	0.75	16.76	55.42	55.42	9.30	55.81
P1376, [X] is the capital of [Y] (310)	73.93	59.83	40.17	56.84	26.10	55.81
P138, [X] is named after [Y] (856)	61.55	59.69	66.05	72.40	18.20	70.26
P140, [X] is affiliated with the [Y] religion (445)	0.63	59.83	75.26	63.00	49.30	61.02
P1412, [X] used to communicate in [Y] (1000)	65.02	64.71	71.21	74.20	49.80	74.01
P159, The headquarter of [X] is in [Y] (1000)	32.37	35.57	35.47	39.71	10.20	28.67
P17, [X] is located in [Y] (1000)	31.29	35.48	52.15	59.14	17.20	56.56
P176, [X] is produced by [Y] (1000)	85.64	81.67	87.78	87.88	55.20	82.46
P178, [X] is developed by [Y] (560)	62.84	59.12	66.72	67.23	29.50	52.30
P19, [X] was born in [Y] (1000)	21.08	20.87	19.92	22.56	6.50	17.80
P190, [X] and [Y] are twin cities (895)	2.41	1.91	2.31	2.61	1.00	2.63
P20, [X] died in [Y] (1000)	27.91	27.91	31.16	32.53	11.90	30.62
P264, [X] is represented by music label [Y] (1000)	9.56	10.26	43.82	38.46	9.90	28.76
P27, [X] is [Y] citizen (1000)	0	41.51	46.69	48.96	25.80	46.63
P276, [X] is located in [Y] (1000)	41.5	41.5	44.11	48.38	20.80	42.50
P279, [X] is a subclass of [Y] (1000)	30.74	14.75	54.93	63.28	22.40	51.95
P30, [X] is located in [Y] (1000)	25.44	18.56	70.36	79.69	43.80	73.23
P31, [X] is a [Y] (1000)	36.66	36.66	51.95	53.90	15.40	45.55
P36, The capital of [X] is [Y] (1000)	62.16	62.16	60.6	63.44	14.70	63.36
P361, [X] is part of [Y] (1000)	23.61	31.44	17.7	41.09	1.70	7.96
P364, The original language of [X] is [Y] (1000)	44.51	43.93	48.48	53.04	16.60	44.02
P37, The official language of [X] is [Y] (311)	54.55	56.83	62.63	67.29	13.00	57.12
P39, [X] has the position of [Y] (1000)	7.96	16.14	30.72	37.33	23.10	33.50
P407, [X] was written in [Y] (1000)	59.18	65.22	68.42	72.63	41.80	66.50
P413, [X] plays in [Y] position (1000)	0.53	23.74	41.7	41.70	19.10	41.70
P449, [X] was originally aired on [Y] (1000)	20.89	9.08	34.39	35.19	15.90	28.06
P463, [X] is a member of [Y] (679)	67.11	57.33	54.22	65.78	25.10	39.20
P47, [X] shares border with [Y] (1000)	13.67	13.34	19.52	15.84	5.30	14.51
P495, [X] was created in [Y] (1000)	16.5	32.23	36.63	40.37	9.90	37.51
P527, [X] consists of [Y] (1000)	11.07	10.55	25.61	27.66	3.30	23.77
P530, [X] maintains diplomatic relations with [Y] (927)	2.81	3.92	3.11	3.41	0.90	2.21
P740, [X] was founded in [Y] (1000)	7.59	13.68	13.89	15.71	8.80	9.25
P937, [X] used to work in [Y] (1000)	29.77	39.1	38.36	44.23	13.80	39.81

Table 10: *Fact retrieval Precision @1*. PERO outperforms or matches Autoprompt for all except one relation when using all training data. With 10 examples, PERO performs significantly better than the Autoprompt for all relations. Please see Section 5.3 in the main text for details.