WARP: Word-level Adversarial ReProgramming

Karen Hambardzumyan[1], **Hrant Khachatrian**[1,2], **Jonathan May**[3]
[1]YerevaNN, [2]Yerevan State University,
[3]Information Sciences Institute, University of Southern California
mahnerak@yerevann.com, hrant@yerevann.com, jonmay@isi.edu

Abstract

Transfer learning from pretrained language models recently became the dominant approach for solving many NLP tasks. A common approach to transfer learning for multiple tasks that maximize parameter sharing trains one or more task-specific layers on top of the language model. In this paper, we present an alternative approach based on adversarial reprogramming, which extends earlier work on automatic prompt generation. Adversarial reprogramming attempts to learn task-specific word embeddings that, when concatenated to the input text, instruct the language model to solve the specified task. Using up to 25K trainable parameters per task, this approach outperforms all existing methods with up to 25M trainable parameters on the public leaderboard of the GLUE benchmark. Our method, initialized with task-specific human-readable prompts, also works in a few-shot setting, outperforming GPT-3 on two SuperGLUE tasks with just 32 training samples.

1 Introduction

Language model pretraining has had a tremendous impact on solving many natural language processing tasks (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019; Liu et al., 2019). The most popular two approaches take a pretrained model and use a straightforward supervised learning objective. In the first approach, the parameters of the language model are frozen and a task-specific head is trained on top of them (Peters et al., 2018). The second approach fine-tunes all model parameters (Radford et al., 2018). The latter can sometimes yield better results (Peters et al., 2019), while the first one usually offers better stability for smaller datasets. The approach based on frozen features does not require storing task-specific language models.

A recent alternative is based on so called adapters (Houlsby et al., 2019; Pfeiffer et al., 2021), a technique that adds new weights at every layer of the pretrained language model while the original parameters are kept frozen. This enables a smaller set of task-specific parameters while achieving results comparable to the fine-tuning approach.

Another approach of leveraging pretrained language models for downstream tasks, introduced by Radford et al. (2019), provides "task descriptions" without using any labeled examples. GPT-3 (Brown et al., 2020) demonstrates impressive few-shot learning performance with *priming*: by providing the language model a few inputs and outputs ("analogies") as a context. The language model contextually "learns" from these examples and outputs the answer with a single forward pass without any trainable parameters. These methods, however, require huge language models (1.5B and 175B parameters, respectively).

The success of task reformulation-based approaches suggest that language models are capable of solving various natural language processing tasks given a well-crafted *prompt*. We hypothesize that it is possible to find such prompts. In other words, we can discover extra tokens that, when added to the input, can exploit language model capabilities better than the manually-designed ones.

In this paper, we introduce a novel technique to find optimal prompts. We call our method **WARP: Word-level Adversarial RePrograming**[1]. The method is inspired by adversarial reprogramming (Elsayed et al., 2019) — a method of adding adversarial perturbations to an input image that reprograms a pretrained neural network to perform classification on a task other than the one it was originally trained for.

[1]Our implementation is publicly available at: https://github.com/YerevaNN/WARP

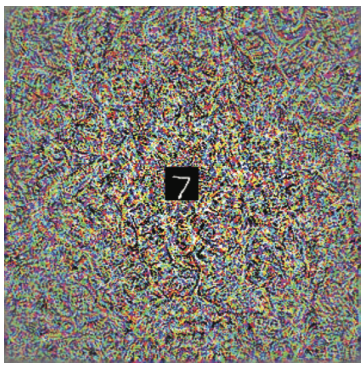

Figure 1: An example of an adversarial program that causes Inception V3 ImageNet model to function as an MNIST classifier, from Elsayed et al. (2019)

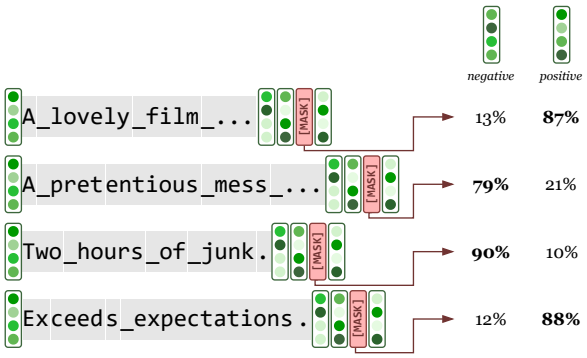

Figure 2: WARP adds a few trainable embeddings around the input, which causes the masked language model to predict the sentiment of the sentence.

We show that our method, using up to 25K trainable parameters per task, achieves 81.6 test score on the GLUE Leaderboard, outperforming all the other submissions that use up to three orders of magnitude more trainable parameters. We show that it is possible to inject knowledge into WARP models using manually designed initialization of the prompt, which is especially useful on tasks with a small number of examples. Moreover, WARP shows impressive few-shot performance on two tasks from the SuperGLUE benchmark with just 32 examples, outperforming GPT-3 results. Finally, we discuss the advantages of our method in real-life applications.

2 Related Work

2.1 Towards Fewer Trainable Parameters

Jiao et al. (2020) show that knowledge distillation may help reduce the size of their model 7.5 times while almost preserving the performance, but fine-tuning such models still requires storage of separate task-specific models. As seen in Section 6, this approach does not scale when we want to apply it to many tasks at once.

Another approach, called Adapters (Houlsby et al., 2019; Pfeiffer et al., 2021), introduces new task-specific parameters that are added at every layer of the Transformer network. Only these newly initialized weights are trained, which allows separation of general and task-specific knowledge. In contrast, our method does not inject task-specific knowledge inside the body of the pre-trained language model. Instead, it focuses on learning task-specific input-level prompts.

2.2 Task Reformulation

In GPT-2, Radford et al. (2019) introduce a completely unsupervised way for transferring knowledge to downstream tasks by reformulating various natural language understanding tasks into language modeling problems. This approach does not make use of the available training examples. Brown et al. (2020) demonstrate an effective few-shot transfer by reformulating downstream tasks into input-output analogies in the context without a need for further fine-tuning. Nonetheless, the number of training examples is limited to the context size and is not scalable to a traditional supervised learning scenario.

Schick and Schütze (2021b) show the effectiveness of reformulating a number of tasks into Cloze-style tasks by fine-tuning masked language models (Devlin et al., 2019). The method, called Pattern Exploited Training (PET), additionally uses training samples and performs few-shot learning even without huge models such as GPT-3.

Our method is also based on masked language models, but unlike PET, we focus on finding the best prompt using the training examples. This eliminates the need for manually-designed prompts, however, our method can also benefit from similar prior knowledge about the task by careful initialization of the prompts.

2.3 Adversarial Reprogramming

Adversarial Reprogramming (Elsayed et al., 2019) demonstrates the reprogramming of pretrained ImageNet classifiers by adding input-level adversarial perturbations to make them perform well on MNIST and CIFAR-10 image classification tasks. The adversarial perturbation is designed to be image padding added to the original input, as illus-

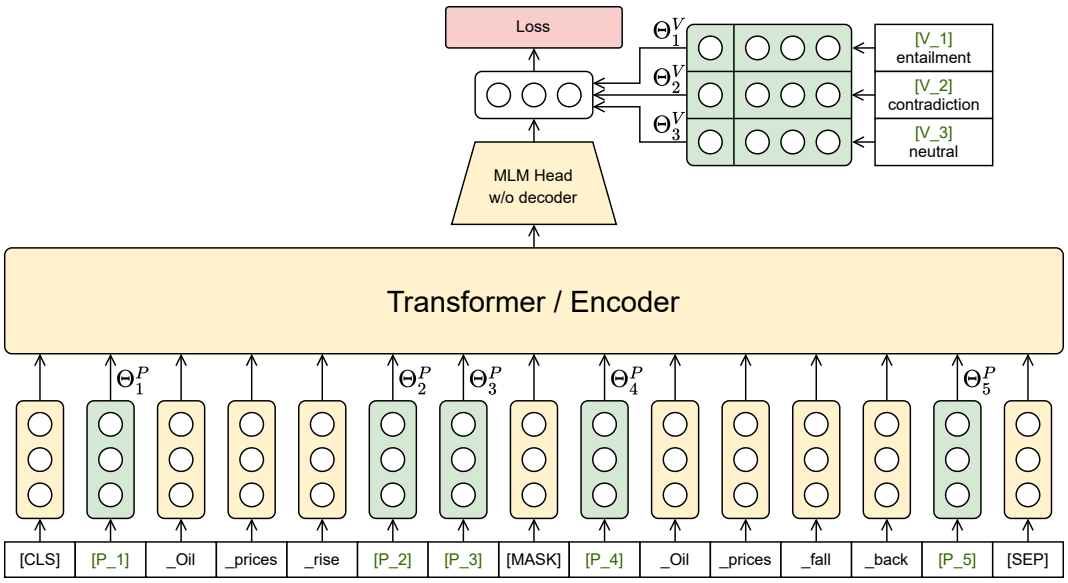

Figure 3: Illustration of WARP. The prompt tokens `[P_1]`, `[P_2]`, ..., `[P_N]` are inserted before, between, and after the sentences. Only the prompt and class embeddings are trainable (colored in green). The masked language modeling Head is applied without the decoder; instead, the matrix of `[V_1]`, `[V_2]`, ..., `[V_N]` is applied as a linear layer. Finally, a regular task-specific loss is computed on the resulting logits.

trated in Figure 1. Then the perturbation parameter is trained to optimize the target classification task objective using the annotated image data.

While in the case of image classification it is not obvious why adversarial reprogramming should ever work, e.g. why a network trained on ImageNet should have the capacity to solve MNIST when surrounded with a particular bitmap, for NLP tasks, there is more intuition. Many NLP tasks can be reformulated as language models, a shared space for both program and data.

Adversarial reprogramming has been adapted to text classification tasks with LSTM networks in (Neekhara et al., 2019). They operate in the vocabulary space and reprogram a model trained for one task to perform another task. More recently, AutoPrompt (Shin et al., 2020a) attempts to find prompts for large language models automatically without adding any parameters to the model. Unlike AutoPrompt, we perform gradient-based optimization in the space of word embeddings which gives our model more degrees of freedom and eventually better performance on the downstream tasks (Section 6.2).

In a more general sense, guiding an NLP model with special tokens appended to the input is an even older idea. In particular, multilingual neural machine translation models use special tokens in the input to control the target language (Ha et al., 2016; Johnson et al., 2017) or politeness of the translation (Sennrich et al., 2016). Another method to reprogram a BERT-based model is proposed by Artetxe et al. (2020), where a model tuned on an English version of a particular task is transformed to work in another language by changing only the embedding matrices.

In parallel work, Li and Liang (2021) propose a similar method and successfully apply it on two text generation tasks. Apart from the different types of tasks and our characterization of the task as a form of Adversarial Reprogramming, the main difference between their approach and ours is that they use an additional parameterization trick to stabilize the training.

3 WARP

We follow a setup similar to Elsayed et al. (2019) with some NLP-specific modifications depicted in Figure 2.

Our goal is to find the best prompt that will make a pretrained masked language model predict the desired answer (verbalizer token) for a training example's masked token[2]. We search for such prompts in the (continuous) embedding space. In other words, we want to find parameters $\Theta = \{\Theta^P, \Theta^V\}$ for prompt and verbalizer embed-

[2]This approach can be easily extended to autoregressive language modeling.

dings, respectively, such that:

$$\Theta^* = \arg\max_{\Theta} \left(-\log P_\Theta(y|x)\right)$$

and the probabilities are given by:

$$P_\Theta(y|x) = \frac{\exp \Theta_y^V f(T_{\Theta^P}(x))}{\sum_{i \in C} \exp \Theta_i^V f(T_{\Theta^P}(x))}$$

where $T_{\Theta^P}(x)$ is the template that inserts the prompt embeddings Θ^P into predefined positions, C is the set of classes, and $f(x)$ is the masked language model output (without the last decoder layer, which is simply the transposed word embedding matrix). Both Θ^P and Θ^V are vectors in the same embeddings space as the word embeddings.

In Figure 2, the template $T_{\Theta^P}(x)$ prepends Θ_1^P and appends Θ_2^P, Θ_3^P, Θ_4^P parameters to the word embeddings and uses Θ_+^V and Θ_-^V to calculate the probabilities on the masked token position for positive and negative classes.

3.1 Method

Similar to Elsayed et al. (2019), we employ stochastic gradient descent to find the best adversarial perturbation on the text that will minimize the task objective. First, we insert special prompt tokens `[P_1]`, `[P_2]`, ... `[P_K]` and an additional `[MASK]` token into the input sequence. These tokens might be placed before or after the sentences, depending on the prompt template.

We set the optimization objective to a cross-entropy loss between the head output of the masked language model and the verbalizer tokens `[V_1]`, `[V_2]`, ..., `[V_C]` for classes $1...C$ accordingly.

The only trainable parameters are the word embeddings for `[P_1]`, ..., `[P_K]` and `[V_1]`, ... `[V_C]`. In case we want to train models for multiple tasks, these are the only task-specific parameters we need to store. The entire "body" of the large language model (all attention layers, feed-forward layers, and all other word embeddings) remains untouched.

Note that, unlike most adversarial attacks, we do not update the embeddings of the original tokens of the input. This follows the intuition from Elsayed et al. (2019), when the pixels of MNIST or CIFAR images are left untouched, and only padding pixels are updated.

We train these parameters by minimizing the loss on the training set of the downstream task.

3.2 Implementation Details

WARP is implemented in the AllenNLP framework. For all the GLUE benchmark tasks we use the `roberta-large` (Liu et al., 2019) model from the `PyTorch` implementation of `huggingface transformers` (Wolf et al., 2020) library. For the few-shot experiments, we use `albert-xxlarge-v2` in order to directly compare to iPET (Schick and Schütze, 2021b). For the GLUE and SuperGLUE tasks we use dataset loaders and metrics implementations from the `huggingface datasets` library.

The prompt tokens are initialized either with word embeddings of `[MASK]` or similar to the vectors from the word embedding layer. For the answer prompts, we use the masked language model head, which usually consists of a feed-forward network and a decoder on top of it, where the weights of the decoder are shared with the word embeddings used for the input. We calculate the softmax over the verbalizer tokens `[V_1]`, ... `[V_C]`.

We choose the Adam optimizer with a slanted triangular schedule for the learning rate with 6% warm-up steps and train for 10-20 epochs on each task. Each batch consists of examples containing at most 1024 tokens and 8 examples.

In order to speed up the training, we disable the dropout of the pretrained language model. All the experiments are performed on two Titan Vs and two RTX 3080 GPUs, with mixed precision training. In practice, WARP is 2.5-3 times faster than regular fine-tuning and 2 times slower than frozen-features experiments in terms of epoch duration with the same batch sizes.

Details about the hyperparameters can be found in the Supplementary material.

4 Experiments on GLUE

Following prior work, we evaluate our method on the GLUE Benchmark (Wang et al., 2019b), which consists of 9 natural language understanding tasks. Generally, we perform single-task WARP training, with early stopping and model selection using the original validation sets, if not stated otherwise.

4.1 Tasks

Almost all the tasks from the GLUE Benchmark are either sentence classification or sentence pair classification tasks, so WARP requires very few modifications to adapt to each of the tasks.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS-B	AVG	#
Human Baselines	92.0 / 92.8	91.2	59.5 / 80.4	93.6	97.8	86.3 / 80.8	66.4	92.7 / 92.6	87.1	
DeBERT[3]	91.9 / 91.6	99.2	76.2 / 90.8	93.2	97.5	94.0 / 92.0	71.5	92.9 / 92.6	90.8	$3 \cdot 10^9$
RoBERTa	90.8 / 90.2	95.4	74.3 / 90.2	88.2	96.7	92.3 / 89.8	67.8	92.2 / 91.9	88.1	$355 \cdot 10^6$
$\text{BERT}_{\text{large}}$	86.7 / 85.9	92.7	72.1 / 89.3	70.1	94.9	89.3 / 85.4	60.5	87.6 / 86.5	80.5	$355 \cdot 10^6$
$\text{BERT}_{\text{base}}$	84.6 / 83.4	90.5	71.2 / 89.2	66.4	93.5	88.9 / 84.8	52.1	87.1 / 85.8	78.3	$110 \cdot 10^6$
TinyBERT_6	84.6 / 83.2	90.4	71.6 / 89.1	70.0	93.1	87.3 / 82.6	51.1	85.0 / 83.7	78.1	$67 \cdot 10^6$
TinyBERT_4	82.5 / 81.8	87.7	71.3 / 89.2	66.6	92.6	86.4 / 81.2	44.1	81.9 / 80.4	75.9	$15 \cdot 10^6$
$\text{ELECTRA}_{\text{small}}$	81.6 / 81.2	88.3	70.4 / 88.0	63.6	91.1	89.0 / 84.9	55.6	85.6 / 84.6	77.4	$14 \cdot 10^6$
Adapters (BERT)	85.4 / 85.0	92.4	71.5 / 89.4	71.6	94.3	88.7 / 84.3	59.2	87.3 / 86.1	80.2	$1.2 \cdot 10^6$
WARP (RoBERTa)	88.0 / 88.2	93.5	68.6 / 87.7	84.3	96.3	88.2 / 83.9	53.9	89.5 / 88.8	81.6	$< 25K$

Table 1: Test set results on GLUE Benchmark. The results are obtained from the GLUE Evaluation server. The subscript next to TinyBERT corresponds to the number of layers in the model. WARP for RTE, STS-B and MRPC are intialized from the MNLI parameters. Results for WNLI are not shown, although they are counted in the averaged GLUE score (AVG column). The last column **#** shows the number of *trainable* parameters. WARP's average performance is higher than all models with up to three orders of magnitude more trainable parameters. Fully fine-tuned RoBERTa and the current state-of-the-art method (DeBERT) score higher by 6.5 and 9.2 points, respectively.

SST-2 (Sentence Sentiment Treebank, Socher et al., 2013) is a single sentence binary classification task. For the prompt, we put a `[MASK]` token after the sentence, and the trainable prompt tokens are both appended and prepended to the sentence.

CoLA (Corpus of Linguistic Acceptability, Warstadt et al., 2019) is a single sentence classification task as well, so we treat both the same way with the only difference that as a validation metric we use accuracy for SST-2, and Matthew's correlation for CoLA.

MNLI (MultiNLI, Multi-Genre Natural Language Inference, Williams et al., 2018), **QNLI** (Question Natural Language Inference, Rajpurkar et al., 2016) and **RTE** (Recognizing Textual Entailment, Dagan et al., 2006; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009) are sentence pair classification tasks. Similar to Schick and Schütze (2021a), we may have prompt tokens before, after and between the two sentences, but the `[MASK]` token is always put between the sentences. For MNLI, we use matched accuracy as a validation metric and use the same model for the mismatched version. In our few-shot attempt for the RTE task, we use a different training and evaluation setup discussed in Section 5.2. **QQP** (Quora Question Pairs[4]) and **MRPC** (Microsoft Research Paraphrase Corpus, Dolan and Brockett, 2005) follow the same prompt pattern as NLI tasks. As a validation metric F_1 score is used.

STS-B (Semantic Textual Similarity Benchmark, Cer et al., 2017), unlike the other tasks in the benchmark, is formulated as a regression task. The prompt pattern is the same, but instead of introducing new embeddings for `[V_1]`, `[V_2]`, `...`, `[V_C]` verbalizer tokens, we add a regression head to the last hidden state of MLM head and use Mean Squares Error optimization objective, similar to (Liu et al., 2019). Pearson Correlation is used as the validation metric. During inference, we clip the scores within $[1, 5]$.

We follow Liu et al. and train models for MRPC, STS-B, and RTE tasks initialized with the parameters from the best MNLI model but do not apply any task-specific tricks to **WNLI** (Winograd Schema Challenge NLI, Levesque et al., 2011) and always predict the majority label.

4.2 Results

Table 1 presents the results on the test set obtained from the GLUE evaluation server. Besides our best WARP models, we also include the human baselines, current state-of-the-art model (He et al., 2020), the regular fine-tuned pretrained model we use, and also include relatively small language models, including (Jiao et al., 2020), (Clark et al., 2020), (Houlsby et al., 2019).

With the GLUE Score, WARP outperforms all the models that train less than 25 million parameters on the leaderboard. We explain the relatively strong WARP results on textual entailment tasks by the easier reformulation of such tasks. Likewise, we explain the relatively weak performance on CoLA by the difficulties of reformulating the

[4]https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS-B	AVG	#
train size	392702	104743	363846	2490	67349	3668	8551	5749		
Fine-Tuning	90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	88.9	$355 \cdot 10^6$
Adapters	90.4	94.7	88.5	83.4	96.3	92.9	67.4	92.5	88.3	$3 \cdot 10^6$
Linear Classifier	64.2	78.1	74.9	59.2	88.4	82.5	48.9	71.8	71.0	≤ 3072
WARP_0	70.9	78.8	77.1	72.2	89.8	83.8	32.8	73.8	72.4	≤ 3072
WARP_1	83.9	87.6	81.6	72.6	93.8	84.7	46.1	80.4	78.8	≤ 4096
WARP_2	85.4	88.0	81.5	69.7	94.3	85.3	54.4	80.8	79.9	≤ 5120
WARP_4	86.9	92.4	83.1	68.2	95.9	85.0	56.0	75.5	80.4	≤ 7168
WARP_8	87.6	93.0	83.8	72.9	95.4	85.6	57.4	81.0	82.1	$< 11K$
$\text{WARP}_{\text{init}}$	86.8	90.4	83.6	80.1	96.0	86.0	51.7	86.9	82.7	$< 11K$
WARP_{20}	88.2	93.5	84.5	75.8	96.0	90.8	60.6	88.6	84.8	$< 25K$
$\text{WARP}_{\text{MNLI}}$				86.3		91.2		91.0	86.4	$< 25K$

Table 2: Dev set results on GLUE tasks. The last column shows the number of *trainable* parameters only. WARP_i corresponds to WARP training with prompt consisting of i prompt tokens. $\text{WARP}_{\text{MNLI}}$ corresponds to WARP training initialized with the best MNLI parameters. All the models are based on pretrained `roberta-large`, and for Adapters and WARP-based approaches require to store $355 \cdot 10^6$ frozen parameters shared across all the GLUE tasks. We show the primary validation metric for each task, described at Subsection 4.1. The AVG column shows the average of shown metrics and is not comparable to the Test server GLUE Score. The number of parameters for WARP methods may vary because of a difference in the number of classes. Underlined numbers correspond to our GLUE submission.

task into a Cloze task.

To further analyze WARP, we conduct several experiments and focus on dev set results. In order to directly compare WARP with existing methods, we report in Table 2 different methods that use RoBERTa, including fine-tuning, linear classifiers on top, AutoPrompt, and Adapters.[5] For WARP experiments, we compare performance with different numbers of prompt tokens.

The WARP_0 model does not introduce any prompt parameters. The only difference between WARP_0 and Linear Classifier is that for WARP_0, `[MASK]` is added to the input of each sample, and we get sentence representations from the MLM head at the masked position. By contrast, in the case of the Linear Classifier, we use the average of non-special token embeddings as sentence representations. As we can see, pooling with MLM is significantly better.

Table 2 shows that, as we decrease the number of trainable prompt parameters, the performance decreases, but the model still works. Similar behavior was observed by Elsayed et al. (2019) in experiments with different padding parameter sizes. However, in contrast to WARP, the number of trainable parameters in that work are much greater than the size of the input.

An important benefit of using WARP is that it can be initialized with manual prompts. In addition to the regular models where we initialize with `[MASK]` tokens, we performed a run on the GLUE datasets with the same prompt [CLS] "S_1"? [MASK]. "S_2"! [SEP] for all the tasks (without S_2 for single-sentence tasks). We denote these results as $\text{WARP}_{\text{init}}$ in Table 2. $\text{WARP}_{\text{init}}$ outperforms WARP_8 on tasks with relatively few training examples — RTE, MRPC and STS-B, which indicates its potential in the low-data regime.

5 Few-Shot Experiments

The fact that WARP can be initialized using manually designed natural prompts suggests that we can similarly benefit from such human attribution similar to iPET (Schick and Schütze, 2021b), especially in scenarios with limited training data.

5.1 Setup

For our few-shot experiments we build WARP on top of ALBERT (Lan et al., 2020), the same pretrained model used by PET and iPET. To initialize WARP prompts, we use the same Prompt-Verbalizer Patterns (PVP) from Schick and Schütze (2021b): the embeddings for `[P_1], [P_2]... [P_N]` are initialized with PVP's prompt token embeddings, and embeddings for `[V_1], [V_2]... [V_C]` are initialized with verbalizer token embeddings for their corre-

[5] Unlike in Table 2, Adapters in Table 1 are built on `bert-large-uncased` model.

sponding classes. Unlike `roberta-large`, the `alberta-xxlarge-v2` uses word embeddings of size 128 (8 times smaller than RoBERTa).

5.2 Tasks

In order to compare with GPT-3, PET, and iPET, we use two tasks from **FewGLUE** (Schick and Schütze, 2021b), which is a few-shot subset of the SuperGLUE benchmark (Wang et al., 2019a) consisting of 32 examples for each task. The dataset also provides 20000 additional unlabeled examples, however, we do not make use of them and work in a purely supervised setup.

CB: CommitmentBank (de Marneffe et al., 2019) is a textual entailment task which we treat like the other sentence pair classification tasks. To initialize the prompt we use the template [CLS] "h"? [MASK]. "p" [SEP] . We also initialize `[V_1]`, `[V_2]`, `[V_3]` token embeddings with `_yes`, `_no` and `_maybe` (respectively for `entailment`, `contradiction` and `neutral`).

RTE: Unlike experiments on the RTE task for the full-sized training in the GLUE benchmark, we do not initialize the model with vectors from MNLI. Instead, the prompt is initialized exactly the same way as in the CB task. The only difference is that we have only the two tokens `[V_1]` and `[V_2]` initialized with `_yes` and `_instead` (for `entailment` and `not_entailment`, respectively).

5.3 Model Selection

Although all trainable parameters are manually initialized in this setup, different random seeds can yield different results because of the order the training examples appear during an epoch.

In the few-shot setup we cannot access the original validation set. Thus, we disable early stopping and simply pick the last checkpoint.

In order to find the best initial learning rate, we conduct 20 runs of WARP with the same learning rate each time by randomly choosing 16 training examples and taking the rest for a development set. We repeat this for all candidate learning rates and choose the one with the best average validation performance across all the random seeds.

Finally, in order to eliminate the effect of different random seeds, we build an ensemble model from 20 WARP runs using simple majority vote.

	Model	CB F_1 / Acc.	RTE Acc.
dev	GPT-3 Small	26.1 / 42.9	52.3
	GPT-3 Med	40.4 / 58.9	48.4
	GPT-3	57.2 / 82.1	72.9
	PET (ALBERT)	59.4 / 85.1	69.8
	iPET (ALBERT)	92.4 / 92.9	74.0
	$\text{WARP}_{\text{init}}$ (ALBERT)	84.0 / 87.5	71.8
test	GPT-3	52.0 / 75.6	69.0
	PET (ALBERT)	60.2 / 87.2	67.2
	iPET (ALBERT)	79.9 / 88.8	70.8
	$\text{WARP}_{\text{init}}$ (ALBERT)	70.2 / 82.4	69.1

Table 3: Results on SuperGLUE benchmark. The results for the test set are obtained from SuperGLUE evaluation server. We only show systems performing in a similar few-shot training setup using 32 examples.

5.4 Results

As seen in Table 3, WARP outperforms PET and GPT-3 baselines, but stays behind iPET on both tasks. GPT-3 has 170B parameters, but none of them is being trained for the given tasks. PET and iPET have 255M parameters, and *all* of them are trained for these tasks. Additionally, they leverage unlabeled examples using distillation. WARP has roughly the same 255M parameters, but only 1024 of them are trained for any single model. An ensemble of 20 WARP models has slightly more than 20K trainable parameters.

6 Discussion

6.1 Interpreting tokens learned by WARP

WARP learns prompt embeddings in a continuous space. In this section, we explore those embeddings by looking at the nearby token vectors. Table 6 in the Supplementary material lists the closest tokens (in terms of cosine similarity) to the learned embeddings. All GLUE tasks are initialized with `[MASK]` token, except for RTE, MRPC, and STS-B, which are initialized from the pretrained MNLI model. The prompt tokens of the solutions for those three tasks are quite close to the ones from the MNLI solution. We have seen similar behavior on SuperGLUE experiments with manual initializations. The solution for CoLA (which is one of the worst-performing tasks) is close to the initialized point.

We do not see any prompt tokens that are meaningful in the context of the tasks. As expected, the verbalized tokens are more interpretable. For

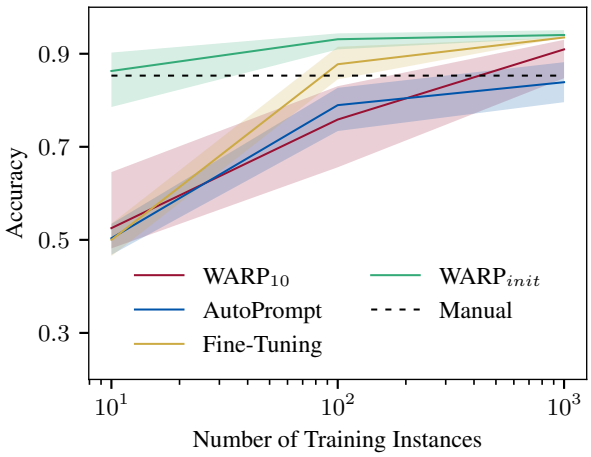

Figure 4: The effect of the training data size for SST-2 task (dev set). Horizontal axis is the number of training examples. Solid lines represent median over 10 runs, and the error bars show minimum and maximum performance. All methods use `roberta-large` model. The results for AutoPrompt and fine-tuning are taken from (Shin et al., 2020b)

.

example, the embedding for the "contradiction" class of MNLI is close to the token "Unless". The embeddings for "negative" and "positive" classes of SST-2 task are close to "defective" and "important", respectively. Other verbalized tokens are non-interpretable (e.g. "470" or word pieces with non-Latin characters).

6.2 Comparison with AutoPrompt

AutoPrompt (Shin et al., 2020b) learns a prompt for the given task in the finite space of vocabulary tokens. Their best version uses 3 or 6 prompt tokens and reaches 91.2% accuracy on the development set of SST-2. The search space of WARP is significantly larger, which allows WARP to get better performance with just a single prompt token (93.8%).

AutoPrompt does not achieve meaningful results on RTE or CB tasks. WARP succeeds on both without manual initialization. Moreover, with manual initialization, WARP gets good performance on both tasks even with just 32 examples (Table 3).

Figure 4 shows the dependence of the accuracy on SST-2 development set from the number of training samples. Both WARP and AutoPrompt use 10 prompt tokens. With a few hundred training samples or fewer, the difference between the two algorithms is not significant. WARP starts to perform better with more training samples.

Approach	# of parameters to store
Linear probing	$M + ECN$
Full fine-tuning	MN
Single layer	$M + NE(E + C)$
TinyBERT	$M_0 N$
Adapters	$M + NEE'$
WARP	$M + NE(C + K)$

Table 4: The number of parameters to be stored to serve N text classification tasks with at most C classes each, using a pretrained language model with M parameters. E is the dimension of embeddings (1024 in the case of RoBERTa). In TinyBERT, M_0 can be up to 10 times less than M. In Adapters, E' is roughly equal to E, as the number of layers to which adapters are attached roughly compensates the smaller size of the bottleneck layer. In WARP, K is the number of prompts (usually fewer than 10).

Shin et al. (2020b) include results with a manually designed prompt[6] which performs pretty well (shown as a dashed line). We also compare with the manually initialized[7] version of WARP, which performs very well with just 100 examples.

6.3 Real-world applications

The importance of NLP systems like WARP can be demonstrated by the following application. Suppose we want to build a system that needs to serve $N >> 1$ classification tasks simultaneously. Let the number of classes for each task be bounded by C. The system can be based on a large pretrained language model with M parameters, using word embedding size E. How many parameters should the system store in the device memory to be able to serve all N tasks?

If we take the approach with frozen features, we can reuse M parameters for all tasks and store additional ECN task-specific parameters. This is optimal in terms of storage but will not perform well. The other extreme is to fine-tune the whole model for each task and store at least MN parameters. Table 4 shows the trade-offs offered by the other solutions. Methods like TinyBERT *decrease* the number of parameters from MN by only M. WARP, on the other hand, needs to *store* only $M + NE(C + K)$ parameters, where K is the number of trainable prompt tokens.

[6] *SENT.* this movie was ___. as a prompt, and "terrible" and "fantastic" as verbalizer tokens

[7] *SENT*, and finally, the movie overall was very ___! as a prompt, and "good" and "bad" as verbalizer tokens

In practice, WARP additionally allows performing inference on inputs for different tasks in parallel, using samples of multiple tasks in the same batch. Every input sentence can be concatenated with task-specific pretrained prompts in advance. Then, the forward pass of the network is identical for all tasks. The final task-specific linear layers can be concatenated to form a single large linear layer with at most NC output neurons.

This approach can be especially useful in the systems that provide machine learning models as a service. By storing one copy of a pretrained language model, it is possible to serve a large number of user-specific models in parallel with little overhead.

7 Conclusion

In this paper we have proposed an alternative way to transfer knowledge from large pretrained language models to downstream tasks by appending carefully optimized embeddings to the input text. The method outperforms existing methods with significantly more trainable parameters on GLUE benchmark tasks and shows an impressive performance in a few-shot setting on two SuperGLUE tasks. On the sentiment analysis task, the performance is comparable to the fully fine-tuned language models. This method can save a lot of storage in software applications designed to serve large numbers of sentence classification tasks.

Acknowledgments

This work is based in part on research sponsored by Air Force Research Laboratory (AFRL) under agreement number FA8750-19-1-1000. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation therein. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Laboratory, DARPA or the U.S. Government.

The work was supported by the RA Science Committee, in the frames of the research project No. 20TTAT-AIa024. Most experiments were performed on GPUs donated by NVIDIA.

References

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637, Online. Association for Computational Linguistics.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge.

T. Brown, B. Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, T. Henighan, R. Child, Aditya Ramesh, D. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, E. Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, J. Clark, Christopher Berner, Sam McCandlish, A. Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc Le, and Christopher D. Manning. 2020. Pre-training transformers as energy-based cloze models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 285–294, Online. Association for Computational Linguistics.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190. Springer.

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.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Gamaleldin F. Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. 2019. Adversarial reprogramming of neural networks. In *International Conference on Learning Representations*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

Thanh-Le Ha, Jan Niehues, and Alexander Waibel. 2016. Toward multilingual neural machine translation with universal encoder and decoder.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

N. Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and S. Gelly. 2019. Parameter-efficient transfer learning for nlp. In *ICML*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *ArXiv*, abs/1907.11692.

Marie-Catherine de Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The CommitmentBank: Investigating projection in naturally occurring discourse. *Proceedings of Sinn und Bedeutung*, 23(2):107–124.

Paarth Neekhara, Shehzeen Hussain, Shlomo Dubnov, and Farinaz Koushanfar. 2019. Adversarial reprogramming of text classification neural networks. 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 5216–5225, Hong Kong, China. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

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

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

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021a. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the*

16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 255–269, Online. Association for Computational Linguistics.

Timo Schick and Hinrich Schütze. 2021b. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, Online. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Controlling politeness in neural machine translation via side constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 35–40, San Diego, California. Association for Computational Linguistics.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020a. 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.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020b. 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 EMNLP*, pages 1631–1642.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

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 Hyperparameters

For each of the tasks, we performed hyperparameter search in the following space:

- **Learning rate** is chosen from the set $\{10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}, 10^{-4}, 3 \cdot 10^{-5}\}$,
- **Number of epochs** is chosen as either 10 or 20. This determines the behavior of the slanted triangular learning rate scheduler.
- **Initialization** is performed either with the embedding of the `[MASK]` token, or randomly initialized from a normal distribution, with the mean and variance taken from the matrix of RoBERTa's word embeddings.

The hyperparameter search took roughly 4 days on two Titan V GPUs. The final choices for each task are shown in Table 5. Initialization with `[MASK]` performed better than the random initialization.

We disable all dropouts inside Transformer. We use *huggingface* implementation of AdamW optimizer with weight decay disabled. The gradient is normalized to the value 1.0. For the batch sampling we use bucketing with padding noise of 0.1. In order to use the device memory more effectively, we also set maximum number of tokens per batch to 2048. The maximum sequence length is truncated to 512 tokens. We enable mixed precision and pad all sequence lengths to the multiples of 8 for the effective usage of TensorCores[8].

[8] https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html

Task	Learning rate	Epochs	Init.
MNLI	0.001	10	[MASK]
QNLI	0.001	10	[MASK]
QQP	0.0003	20	[MASK]
RTE	0.001	20	MNLI
SST-2	0.003	20	[MASK]
MRPC	0.001	20	MNLI
CoLA	0.001	20	[MASK]
STS-B	0.001	20	MNLI

Table 5: Hyperparameters of our best-performing models. `[MASK]` means the prompts are intialized with the word embedding of same token, and MNLI means the prompt is initialized with the prompts of out best MNLI run.

B Learned Tokens

Table 6 lists the closest vocabulary words to the learned embeddings. Most tasks have two input sentences, so the prompts consist of three parts: one is added before the first sentence, the second one is added between the sentences and the third one is appended next to the second sentence. For the single-sentence tasks, the second and third parts of the prompt are simply concatenated. Each task has trainable verbalizer tokens, one per output class.

The prompts of RTE, MRPC and STS-B are pretty similar to MNLI's prompts, as the models for these tasks were initialized from pretrained MNLI models. The other tasks were initialized with `[MASK]` tokens. The final model for CoLA didn't move too far from its initialization.

MNLI	**Prompts**	**before**	A-A-A-A-A-A-A-A-A-A-A-A-A-A-A-A- Tomorrow Ale .aGj _*.
		between	_MUCH irin [/ _a _(@ **[MASK]** _dL aHJ E [MASK] _aKH
		after	_<!– _informing inyl _entit dim
	Verbalizers	**entailment**	_categories
		neutral	gomery
		contradiction	Unless
QNLI	**Prompts**	**before**	*. _neigh [MASK] U _{{
		between	aG—aG— [MASK] olitan _pronouns [MASK] **[MASK]** [MASK] @@@@ [MASK] _Choi [MASK]
		after	
	Verbalizers	**entailment**	_VIDE
		not_entailment	470
QQP	**Prompts**	**before**	_resembling _swarm _Paramount _Calm _Membership
		between	_derive rics [MASK] _alias iary **[MASK]** _omnip [MASK] [MASK] [MASK] _sham
		after	[MASK] _forb [MASK] _Firefly _THEY
	Verbalizers	**not_duplicate**	ende
		duplicate	_sugg
RTE	**Prompts**	**before**	A-A-A-A-A-A-A-A-A-A-A-A-A-A-A-A- Tomorrow ALE .aGj _*.
		between	_MUCH irin [/ _a _(@ **[MASK]** _ aHJ femin [MASK] _aK
		after	ahiahi _informing # _entit OOOO
	Verbalizers	**entailment**	e!
		not_entailment	_blames
SST-2	**Prompts**	**before**	choes _charms _sorely _”... akijakij
		between	a afe Pae _charred _masked **[MASK]** _Fall _babys _smartest ik /
		after	dL forums _bio _mang A+-
	Verbalizers	**negative**	_defective
		positive	_important
MRPC	**Prompts**	**before**	A-A-A-A-A-A-A-A-A-A-A-A-A-A-A-A- Tomorrow rison .aGj _*.
		between	_MUCH irin [/ _a jay **[MASK]** _dL aHJ femin [MASK] .?
		after	_> _informing # _entit OOOO
	Verbalizers	**entailment**	_categories
		neutral	gomery
CoLA	**Prompts**	**before**	[MASK] [MASK] [MASK] [MASK] [MASK]
		between	[MASK] [MASK] [MASK] [MASK] [MASK] **[MASK]** [MASK] [MASK] [MASK] [MASK] [MASK]
		after	[MASK] [MASK] [MASK] [MASK] [MASK]
	Verbalizers	**unacceptable**	_additionally
		acceptable	o
STS-B	**Prompts**	**before**	A-A-A-A-A-A-A-A-A-A-A-A-A-A-A-A- Tomorrow Ale .aGj [MASK]
		between	_Kers irin [/ _a _(@ **[MASK]** _dL AhAHAhAH femin [MASK] _aKH
		after	A-A-A-A-A-A-A-A-A-A-A-A-A-A-A-A- _repertoire inyl _Idea dim
	Verbalizers	**regression**	cH

Table 6: The closest words to the prompt and verbalizer token embeddings for the best model for each task. We use cosine distance to measure the distance. **[MASK]** tokens highlighted in bold indicate the positions we use to output the prediction.