Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing

- ***Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, Graham Neubig***

- https://arxiv.org/abs/2107.13586

Instead of the traditional model that takes x and predicts y as P(y|x), in prompt-based learning the input x in modified into a prompt x' that has some unfilled slots. The language model is then used probabilistically to fill the unfilled information to obtain the final string x, from which the final output y can be derived.

Changes in NLP

- Pre-2017: fully supervised learning.
- 2017-2019: pre-train and fine-tune: a pre-trained LM is adapted for different downstream tasks by introducing additional parameters and fine-tunning them with task-specific objective functions.
- 2021-: pre-train, prompt, and predict: instead of fine-tuning for the downstream tasks, these tasks are reformulated to look more like the original LM training with the help of the textual prompt. While being unsupervised or requiring very few examples (one-shot or zero-shot), it requires prompt engineering: finding the most appropriate prompt to allow the LM to solve the task at hand.

Prompting basics

Instead of training a model P(y|x), prompt-based learning circumvents this fine-tuning step by learning an LM that models the probability of the P(x) of the text directly and then uses this probability to predict y, removing the need for any annotated data.

1. Prompt addition:

We first apply a prompting function that applies a template to input x, where the template generally has two slots, [X] for the input, and [Z] for the slots the model will fill. The result in [Z] is then mapped to the output y. For example: template if [X] Overall, it was a [Z] movie. [X] will be "I love this movie", and then the model will be tasked by filling [Z].

2. Answer Search:

We then need to define the set of possible/permissible values for [Z], it can be the whole corpus or subset of it, for example: {"excellent", "good", "OK", "bad", "horrible"}, and each element in [Z] needs to be mapped into the output space Y. We then use our LM to search over the possible answers [Z], the search can be random or takes the argmax of the results.

3. Answer Mapping:

The next steps consist of mapping the best / highest scoring z into the corresponding output y. For language generation tasks, z itself is the output, but for other cases where multiple answers could results in the same output ("fabulous" & "wonderful" both can be mapped into a positive class), then a mapping between the searched answers and the output value is necessary.

Pretrained Language Models

Objective

The training objective of LM consists of objective predicting the probability of text x:

- Standard LM (SLM): they optimize the probability of text P(x) from a training corpus, where the text is generally predicted in an autoregressive manner from left to right, predicting one token at a time. The model is very suitable for prefix prompts or text generation tasks.
- Corrupted Text Reconstruction (CTR): it first applies from noising function to the input, and then the model is trained to restore the uncorrupted text by only computing the loss over the noised parts of the input sequence. These models are suitable for cloze prompt (fill the [Z]).
- Full-Text Reconstruction (FTR): these models are also trained using a denoising objective, but the loss is computed over the entirety of the input text, whether it has been noised or not. These models are suitable for language generation tasks.

Noising functions

Noising functions can take different forms and can be applied to 1 or 2 tokens, and to a given entity. The noising functions are: Masking (can be random or based on some prior), Replace, Delete, Permute, Rotate, Concatenate (different languages).

Directionality of representations

Another important factor in LM is the directionality of representations:

- Left-to-Right: the representation of a given word is computed on the word itself and all the previous ones. This representation is used for standard LM or when computing the output of an FTR objective.
- Bidirectional: the representation of each work is based on all the words in the sentence.
- Mixture: it is also possible to mix the two strategies together into a single model, or use some random permutation to generate the outputs.

Typical pertaining methods

- Left-to-right LM (L2R LM): a variety of auto-regressive LM, predict the next work (or assign a probability P(x)) based on the word to the left of the current one. The output probability is broken down using the chain rule: P(x) = P(x1) x ... x P(xn | x1 ... xn-1). This dependency is generally implemented using a diagonal attention mask. These models are popular for prompting methods since they are generally very large and hard to train, and often not publicly available.
- Masked Language Models (MLM): While L2R LM are very popular for modeling the probability of text, they require a L2R dependency, and when the focus is shifted from language generation to other downstream tasks that require optimal representation like classification, it is better to use other models like MLM, which predict the masked pieces of the input text based on the surrounded context. These models are very suitable for cloze prompts.
- Prefix and Encoder-Decoder: For conditional text generation or summarization, we need both a model that encodes the input, then generates the output sequentially. For this, they are two popular choices: 1) prefix language model which is a L2R where the encoding of the input and the generation are done by the same model, where the input is processed without an attention mask. and 2) encoder-decoder which uses a separate mode to encode the input without any masking, and then uses an autoregressive L2R model to generate the output. To learn better representation, a noising might be applied to the input in addition to training on predicting the output.

Prompt Engineering

Prompt engineering is the process of creating a prompting function that results in the most effective performance on the downstream tasks. It consists of the following steps:

Prompt Shape

There are two types of prompt shapes, **cloze prompts** which fill the blanks of a textual string, and **prefix prompts** which continue a given prefix. Autoregressive LM works quite well with **prefix prompts**, while masked LM works well with **cloze prompts**. Full reconstruction models are versatile and can be used for both. In terms of tasks that require multiple inputs such as text pair classification, the input must also contain many inputs, eg, [X1] and [X2].

Template Creation

- *Manual Template Engineering:* The most straightforward way to create prompts is to create intuitive templates based on human introspection.
- *Automated Template Learning:* While the strategy of manually crafting templates is intuitive, defining and experimenting with these prompts takes time and the designed prompts may not be optimal. To address these problems, different methods were introduced to automatically learn such prompts:
- *Discrete:* where the prompt is an actual text string and consists of automatically searching for templates in the discrete space:

1. Mining: Automatically searches for templates given input x and output y, and then finds either the middle words or dependency paths between them.
2. Paraphrasing: paraphrases (back-translation, replacement from thesaurus or neural prompt rewriter) the inputs into many candidates prompts, then selects the one that achieves the highest training accuracy on the target task.
3. Gradient-Based Search: applies gradient-based search over actual tokens to find short sequences that can trigger the underlying LM to generate the target prediction.
4. Prompt Generation: Uses standard LM to generate the prompts.
5. Prompt Scoring: From a hand-crafted set of templates, a unidirectional LM is used to score these filled templates, the one with the heights LM probability is selected.

- **Continuous:** without the condition of having human interpretable natural language prompts, we can also create **continuous or soft prompts** that perform prompting directly in the embedding space. This way we remove the constraint of creating prompts of natural language words, and also remove the restriction of using LM parameters, where this time the prompts itself has their own parameters.

1. Prefix Tuning: consists of appending a set of continuous and task-specific vectors to the input while keeping the LM model parameters fixed, the appended prefix is then learned to maximize the LM log-likelihood. The added parameters could be for the input space only, or for each layer of the network.
2. Tunning Initialized with Discrete Prompts: Starting from a discrete set of discovered prompts, the embeddings of these prompts are fine-tuned to increase the task accuracy.
3. Hard-soft Prompt Hybrid Tunning: Instead of a fully learnable prompt, such methods insert a set of tunable embeddings into a hard prompt template.

Answer Engineering

Compared to prompt engineering that design the appropriate inputs, answer engineering design the answer space Z

so that there is a mapping from the model's output to the desired target space Y of a downstream task. Similar to prompt engineering, we first need to define the shape, design the search space, and define a method to search over it.

Answer Shape

The choice of the answer shape depends on the task we want to perform, some choices are 1) Token: either the whole vocabulary or some subset of it, 2) Span: a short multi-token span, which is usually used with cloze prompts and are the model's output corresponding to the blank space, and 3) Sentence, or a document, which are common with prefix prompts.

Manual Answer Design

The mapping from the output space Z to the target space Y is done by using either 1) an unconstrained design strategy with either a fixed span or a sequence of tokens, where the output space already matches the output space, and the mapping is an identity one. 2) Constrained spaces; in this case, the space of possible constrained for tasks with a limited label space such as text classification or entity recognition. In this case, we can design a predefined choice of token for each label, or for multiple-choice tasks, we can use an LM to score each answer.

Answer Search

In order to find the optimal mapping between the output space and the label space, we can do an automatic mapping search, either discrete or continuous search

- *Discrete Answer Search:*
- Answer Paraphrasing: starting from a small/initial answer space (answer -> label), these methods use paraphrasing (such as back-translation) to expand the space and broaden its coverage (paraphrased(answer) -> label), the final answer is then defined as the marginal probability distribution over all of the answers of this paraphrase set.
- Prune-then-search: from a set of several plausible answers, an algorithm is used to further search over this space and prune it to select the final set of answers.
- Label Decomposition: Consist of decomposing each relation label into its constituent words, and the probability will span the sum of each token's probability.
- *Continuous Answer Search:*

These methods directly optimize the answer token via gradient descent. It assigns a virtual token for each class label and optimizes the token embedding for each class together.

Multiple Prompt Learning

Instead of constructing a single prompt for input, we can use multiple prompts to further improve the efficacy of prompting methods, for either discrete or continuous prompts.

Prompt Ensembling

Consists of using multiple ****unanswered**** prompt and then combining the model's outputs at test time. This helps us 1) leverage complementary advantages of different prompts, 2) reduce the cost of prompt engineering since finding a single optimal prompt is challenging, and 3) stabilize the performances on the downstream tasks.

- Uniform averaging: takes a simple average over multiple prompts. Instead of using all available prompts, we can first filter them and only take the top K prompts with the heights output scores.

- Weighted averaging: Instead of using the same weight for every prompt, a pre-specified set of weights can be defined or optimized for each prompt using the training set.
- Majority voting: majority voting can also be used to combine the results of different prompts.
- Knowledge distillation: an ensemble of models can be distilled into a single model using knowledge distillation.
- Prompt ensembling for text generation: For language generation tasks where the output is a sequence of tokens instead of a single one, in this case, we can either use the ensembled probabilities at time step t to generate the next token. Otherwise, we can also first decode the generations and then score each generation by averaging the probabilities across all models.

Prompt Augmentation

Consists of providing a few additional examples of ****answered**** prompts that can be used to demonstrate bow the LM should provide the answer for a given prompt. For example, instead of just providing a prompt of "China's capital is [Z]", the prompt can be prefaced by a few examples such as "Great Britain's capital is London." and "Japan's capital is Tokyo." Then we pass the prompt we want to answer: China’s capital is [Z]. Although the idea of prompt augmentation is simple, there are several aspects that make it challenging: (1) Sample Selection: how to choose the most effective examples? (2) Sample Ordering: How to order the chosen examples with the prompt?

- Sample Selection: to find the appropriate sample to seed the model, we can find the sentences with embeddings close to the prompt to be used and use them to seed the model. Using both positive and negative samples can also help to show the model both the desired and undesired outputs.
- Sample Ordering: Since the order in which the answered prompts are presented to the model is important, the best order can be found with different methods such as entropy-based methods scoring of different permutations.

Prompt Composition and Decomposition

- Composition: For a task that is composable like relation extraction which aims to extract the relation of two entities, we can break the task into several sub-ones including identifying the characteristics of entities and classifying the relationships between entities.
- Decomposition: For tasks where multiple predictions should be performed for one sample such as sequence labeling, directly defining a holistic prompt with regards to the entire input text x is challenging. One intuitive method to address this problem is to break down the holistic prompt into different sub-prompts, and then answer each sub-prompt separately. For example, given an input: Mike went to New York yesterday. the prompts can be [X] Mike is [MASK] entity type and [X] New York is [MASK] entity type.

Training Strategies

While the prompting methods can be used without any explicit training of the LM for the downstream task using its output probability over the tokens directly, which amounts to a zero-shot learning strategy. We can also use training data to train the model in convert with the prompting methods. Either full-data learning using a reasonably large number of training examples or few-shot learning where a very small number of examples are used to train the model. In terms of the parameters to be fine-tunned, we can categorize things into three buckets:

- LM Params: whether the parameters of the underlying LM are tuned or not,
- Prompt Params: whether there are additional prompt-related parameters,
- Tuning Prompt Params: there are additional prompt-related parameters, whether those parameters are tuned or not.

Promptless Fine-tuning

This is the standard strategy of using a pretrained LM, in which the parameters of the pre-trained LM will be updated via gradients induced from downstream training samples. While simple and does not require any prompt design, the LM might overfit or not learn stable on such a small dataset and may also induce catastrophic forgetting.

Tuning-free Prompting

Tuning-free prompting directly generates the answers without changing the parameters of the pre-trained LMs based only on a prompt, while efficient and applicable in a zero-shot setting, they require heavy engineering of the prompts to achieve good results.

Fixed-LM Prompt Tuning

In the scenario where additional prompt-relevant parameters are introduced besides parameters of the pre-trained model, fixed-LM prompt tuning updates only the prompts' parameters using the supervision signal obtained from the downstream training samples while keeping the entire pre-trained LM unchanged. While having the LM frozen we can avoid overfitting and catastrophic forgetting, we cannot use this method in a zero-shot scenario and the prompts are usually not human-interpretable or manipulable.

Fixed-prompt LM Tuning

Fixed-prompt LM tuning tunes the parameters of the LM, as in the standard pre-train and fine-tune paradigm, but additionally uses prompts with fixed parameters to specify the model behavior. This can help with more efficient learning by specifying the prompts beforehand and only fine-tuning the model, especially in a few-shot setting. However, this might now transfer from one task to the other and it requires prompt engineering.

Prompt+LM Tuning

In this setting, there are prompt-relevant parameters, which can be fine-tuned together with all or some of the parameters of the pre-trained models. These methods are suitable for scenarios where the data is readily available, however, they might be too expressive for a small dataset and induce overfitting, and is more computationally expensive than the rest of the methods since we need to store both the prompt and the LM model parameters.