Paper Notes: Bag of tricks to improve Pretrain-Finetune Paradigm

Tianjian Li

Johns Hopkins University

May 24, 2022

1 BART

BART[16] is work by Facebook published at ACL 2020. It aims to generalize BERT [11]. BERT uses a bidirectional encoder that replaces random token with masks and is trained only on the loss between the masked tokens and the ground truth. GPT is trained in a unidirectional autoregressive way, generating words according to their leftward context.

BART combines the two models by including a denoising autoencoder that encodes the corrupted document then autoregressively maps the encoded document to its original uncorrupted counterpart. The encoder is the standard transformer[4], with a silght modification that changes the activating function from ReLU to GeLU[2]. In each layer of the decoder, the model calculates the cross attention over the final hidden layer of the encoder.

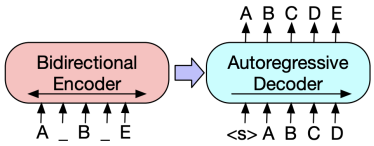

Figure 1: BART

The paper introduces a few corrupting tricks during the pretraining phase:

- Token Masking: Similar to BERT[6].
- Token Deleting : Random deletes input tokens
- Text Infilling : Sample text spans and replace them with a single mask token.

- Sentence Permutation
- Document Rotation:
  A token is chosen uniformly random and the document is rotated so that it begins with this token.

For various downstream tasks:

- Sequence Classification:
  Feed the pretrained encoder and decoder the same input and use let the decoder output a class token, similar to the CLS token in BERT.
- Token Classification(SQuAD):
  Feed the pretrained encoder and decoder the same input and use the final hidden state of the decoder as the representation for each word.
- Sequence Generation:
  The Encoder takes the input and the decoder outputs the generated sequence autoregressively.
- Machine Translation:
  The entire BART model is viewed as an decoder, combined with an additional encoder that takes the source language as input. The model is trained in two steps: 1) Freeze most the parameters of the BART model and only update the parameters of the new encoder, BART positional embeddings and the attention input matrix of the first layer of BART's encoder. 2) Train the entire model for a small number of steps.

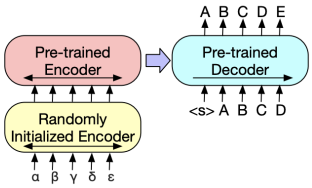

Figure 2: Illustration of Machine Translation Task

2 PET

PET[26]:Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference is published at EACL2021. GPT[11] teaches us that a pretrained model with the correct task descriptions can perform better in downstream tasks than the commonly used pretrain-fintune paradigm. PET proposes that cloze style input phrases can help language model understand a given task.

GPT provides hints for our language model to generalize to tasks(Reading Comprehension, Question Answering)in a zero-shot context. In contrast, PET provides task descriptions can be combined with supervised learning in few shot settings.

A *pattern function* P takes a sequence of sentences as its input and outputs **one** sentence that contains only **one** masked token. The output can be viewed as a cloze question. A *verbalizer* v is a injective function that maps a label to a word in the vocabulary.

We can find pair of P, v to solve specific tasks. For example, if we need to solve a Question Answering(QA) task, given training example Q, A. We define our P as follows:

$$P((Q, A)) = Q.\_\_\_\_, A.$$

The task now changes from having to assign a label without inherent meaning to answering whether the most likely choice for the masked position in a "Yes" or a "No".
During Training, the cross entropy loss between the predicted label and the groundtruth is minimized. Nevertheless, the nature of few shot supervised learning means that "catastrophic forgetting"[26] can occur. So the final loss is a combined loss of the masked language model and the cross entropy loss. (Experiment setting $\alpha = 1 * 10^{-4}$

$$L = (1 - \alpha)L_{CE} + \alpha \cdot L_{MLM}$$

2.1 Key Chanllenges of PET

One of the chanllenges of PET is that we need to manually search for a good pattern function P. The author uses knowledge distillation[1] to ensemble a group of potentially good pattern functions.

1. During finetuning, training different models fore each pattern function P.
2. During ensembling, computes scores by a uniformly weighted or accuracy weighted ensemble of the models. Use softmax to transform the scores into a probability distribution. Use $T = 2$ to get a soft distribution[1]. All of the pairs form a new training set T_C
3. Finetune PLM with a standard sequence classification on T_C.

3 Adapter Layers

Adapter Layers was introduced to NLP in [7] published at ICML 2019. The main paradigms in transfer learning are 1)feature based transfer and 2)Fine-tuning. While the former learns an embedding for each word/token then feeds the embedding into models for downstream tasks, the latter is not trained in an end to end way. We first pre-train to learn weights, then fine-tune for a small number of epochs.

However, as the author points out, both 1) and 2) are not parameter efficient because they require additional parameters to train in order to achieve performance, we can see this in the figure below.

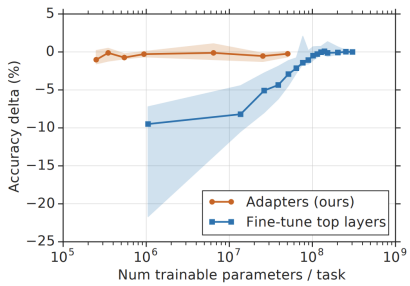

Figure 3: Fine-tuning vs Adapters from [7]

Adapters are new modules added between layers of a **pre-trained** network. The main properties of adapters are:

- Initialized as identity functions
- Very few parameters added

In this paper, the adapter layers is specifically designed, implemented and experimented on transformers[4] for NLP tasks. Although it is also useful in transfer learning in Computer Vision[3].

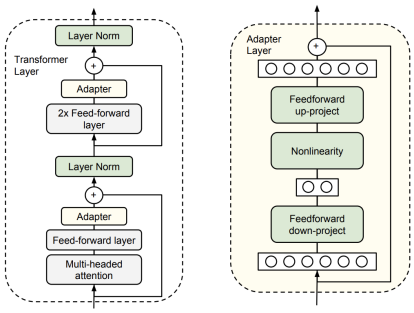

Figure 4: Illustration of adapter layers from [7]

A transformer block mainly contains one multi-head self attention layer and a feed forward network followed by a layer normalization. The adpater layers are added after the output of each layer and before the residual connection. In order to constraint the number of parameters, the adapter layer first projects the d dimensional input into a m dimensional tensor where $w << d$, apply a nonlinear function then projects back to its original d dimension. The down

projection adds $md+m$ parameters, the up projection adds $md+d$ parameters. The total added number of parameters are $2md + d + m$ including the bias. Importantly, the model also trains new layer normalization parameters for each downstream task.

4 Transformer-XL

Transformer-XL[5] is a architecture that enables learning dependency beyond a fixed length without disrupting temporal coherence based on the Transformers[4]. To design a model based on transformer to capture long range dependencies, we start with segmenting the inputs. We first cut the corpus into smaller segments, then only train our model within each segment.

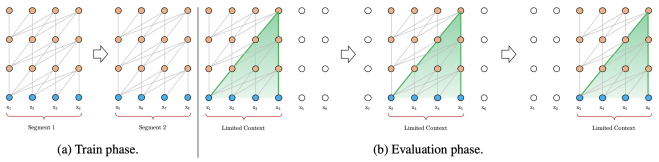

Figure 5: Vanilla Transformers, figure taken from [5]

Under this training paradigm, information never flows across segments in either the forward or backward pass. The raises two problems: 1) The largest possible dependencies captured by the model is strictly bounded by the length of the segment. 2) In practice we simply chunk the corpora into fixed size segments, paying no heed to the start and/or end of sentences. Cutting a sentence into two pieces and making them unable to learn context from each other results in a major performance drop.

Transformer-XL in-cooperates the idea of RNN into segmented Transformers. During training, the hidden state sequence computed for the previous segment is fixed and cached to be reused as an extended context when the model processes the next new segment.

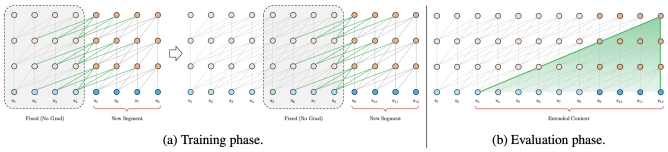

Figure 6: Transformers-XL, figure taken from [5]

Formally, if the first sentence is $s_i = [x_i^1, ..., x_i^L]$ and the next sentence is $s_i + 1 = [x_{i+1}^1, ...x_{i+1}^L]$ we have the following update rule for the hidden states h_{i+1}

$$\tilde{h_{i+1}} = \text{concat}(\text{stop\_gradient}(h_i), h_{i+1})$$
$$(q_{i+1}, k_{i+1}, v_{i+1}) = \text{linear\_transform}(h_{i+1}, \tilde{h_{i+1}}, \tilde{h_{i+1}})$$
$$h_{i+1} = \text{Transformer}(q_{i+1}, k_{i+1}, v_{i+1})$$

Another trick adopted by Transformer-XL is **Relative Positional Encoding**. This is straightforward because if we apply the original encoding methods in Transformers, the model fails to distinguish between the ith word of the previous sentence and the ith word of the current sentence. Observing that the difference between positions matters more than the actual position of our **query** token and **key** token, we use the difference between positions to encode the words. If we use E as the word embedding and P as the positional embedding, then attention weight in vanilla transformer(before scaling and softmax)is:

$$(E_Q + P_Q)W_Q^\top W_K(E_K + P_K)$$

$$= E_Q W_Q^\top W_K E_K + E_Q W_Q^\top W_K P_K + P_Q W_Q^\top W_K E_K + P_Q W_Q^\top W_K P_K$$

In Transformer-XL, the model trains two vectors q_E and q_{Diff}(u, v in the original paper)for $P_Q W_Q$ and uses the encoding of the difference as P_K, the attention score becomes:

$$E_Q W_Q^\top W_K E_K + E_Q W_Q^\top W_K \text{Diff}_{q,k} + q_E W_K E_K + q_{\text{Diff}} W_K \text{Diff}_{q,k}$$

There are a few implementing tricks and optimizations for Transformer-XL, they are beyond the scope of this note. The main takeaways are the two tricks for long range dependencies: **caching hidden states and relative positional encoding**.

5 XLNet

5.0.1 Model Design and Implementation

Autoregressive(AR) language modelling performs pretraining by maximizing the likelihood under the forward autoregressive factorization:

$$\max_\theta \log p_\theta(x) = \sum_{t=1}^{T} \log p_\theta(x_t | x_{<T}) = \sum_{t=1}^{T} \frac{\exp(h_\theta(x_{1:t-1})^\top e(x_t))}{\sum_{x'} \exp(h_\theta(x_{1:t-1})^\top e(x')))}$$

whereas **Autoencoding(AE)** language modelling aims to reconstruct the original text corpus from a corrupted text:

$$\max_\theta \log p_\theta(x_{\text{masked}} | x_{\text{corrupted}}) = \sum_{t=1}^{T} m_t \log p_\theta(x_t | x_{\text{corrupted}}) = \sum_{t=1}^{T} \frac{\exp(H_\theta(x_{\text{corrupted}})^\top e(x_t))}{\sum_{x'} \exp(h_\theta(x_{\text{corrupted}})^\top e(x')))}$$

here m_t is a indicator function of where the token at t is masked.

The following table compares two paradigm AR and AE models of Transformers[4] and BERT[6], respectively.

Type	AR	AE
Example	Transformers[4]	BERT[6]
Independence Assumption	No assumption	masked tokens are independent of each other(which is not always true)
Input Noise	No input noise	Corrupted input leads to discrepancy between pretrain and finetune
Context Dependency	Unidirectional	Bidirectional

To enable AR methods to capture bidirectional context, the authors of XLNet[10] proposes maximizing the expected likelihood over **all permutations** of the factorization order. During implementation, XLNet keeps the original sequence order, use the positional encoding corresponding to the original sequence, and rely on a proper attention mask in Transformers to achieve permutation of the factorization order.

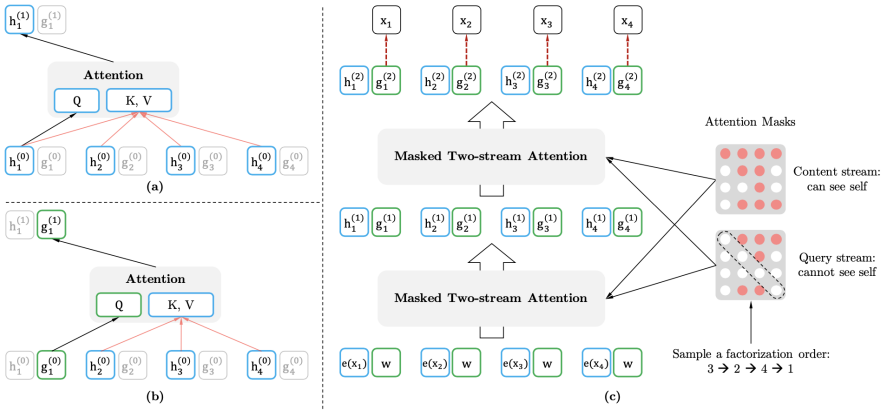

Figure 7: XLNet model architecture, figure from[10].

If we only use the hidden representations in Transformers[4]. Notice that the representation $h_\theta(x_{z<t})$ does not depend on which position it will predict, i.e., the value of z_t. Consequently, the same distribution is predicted regardless of the target position, which is not able to learn useful representations. To avoid this problem, we propose to re-parameterize the next-token distribution to be target position aware.

To resolve such a contradiction, XLNet propose to use two sets of hidden representations instead of one:

- The content representation $h_\theta(x_{z<t})$, or abbreviated as h_{z_t}, which serves

a similar role to the standard hidden states in Transformer. This representation encodes both the context and x_{z_t} itself.

- The query representation $g_\theta(x_{z<t}, z_t)$,which only has access to the contextual information $x_{z<t}$ and the position z_t, but not the content x_{z_t}.

Therefore XLNet[10] updates two sets of hidden outputs: g_{z_t}, which uses z_t but cannot see x_{z_t}, and h_{z_t}, which uses both the position z_t and also the content x_{z_t}. Formally:

$$g_{z_t}^{(m)} = \text{Attention}(Q = g_{z_t}^{(m-1)}, KV = h_{z_t}^{(m-1)}; \theta)$$

$$h_{z_t}^{(m)} = \text{Attention}(Q = h_{z_t}^{(m-1)}, KV = h_{z_t}^{(m-1)}; \theta)$$

The probability distribution $p(X_{z_t} = x|x_{z<t})$ of the next token is computed with the position aware and content blind hidden outputs $g_{z_t}^{(L)}$, where L denotes the number of transformer layers.

$$p(X_{z_t} = x|x_{z<t}) = \frac{\exp(e(x)^\top g_\theta(x_{z<t}, z_t))}{\sum_{x'} \exp(e(x')^\top g_\theta(x_{z<t}, z_t))}$$

5.1 Transformer-XL

As discussed in the previous note of Transformer-XL[5], Transformers fails to efficiently capture long range language dependencies. XLNet solves this issue by using Transformer-XL, to cut long sentences/documents into smaller chunks and caching the previous hidden state(the hidden state output of the previous chunk).

$$h_{z_t}^{(m)} = \text{Attention}(Q = h_{z_t}^{(m-1)}, KV = [\tilde{h}^{(m-1)}, h_{z_t}^{(m-1)}]; \theta)$$

Where $\tilde{h^{(m)}}$ denotes the cached hidden output of the previous segment/chunk. Note that we need to store the hidden output of all of the layers to acquire a hidden output of a specific layer m.

5.2 Future works

XLNet outperforms autoencoding methods(BERT) in a reading comprehension, text classification because these tasks needs to caputure long range dependencies and also need to find hidden relationships between different parts of the text. BERT assumes that the masked positions are independent, while XLNet does not depend on this assumption. XLNet was published in 2019. In year 2022, the idea of combining autoregressive models and autoencoding methods in few-shot/zero-shot learning scenarios with prompt tuning achieve great success[19].

6 ELECTRA

ELECTRA was proposed in **Pre-training Text Encoders as Discriminators Rather Than Generators**[13] in ICLR 2020. The major contributions of this paper are

- Introducing an more sample-efficient "replaced token detection"[13] pre-training objective as opposed to the *de facto* inefficient Masked Language Modeling(MLM).
- Experimented applying reinforcement learning methods an pointing out that Generative Adversarial Nets(GAN) generally fails in natural language understanding. (Note: For a more detailed explaination of why GAN doesn't work on language models, please refer to [12].)

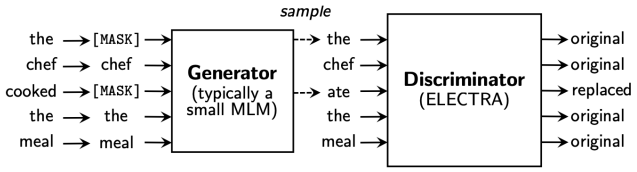

Figure 8: Overview of Replaced Token Detection. Copied from [13]

An higher order explanation of the replaced token detection objective is given an input sentence, we use a small generative network trained with a Masked Language Modeling(MLM) objective to reconstruct masked tokens. Then, instead of training a model that predicts the original identities of the "corrupted" tokens, we train a **discriminative** model that predicts whether each token in the corrupted input was replaced by a generator sample or not.

A few notations on the methods are 1) If the generative model generates the groundtruth token, the token is considered to be "real" instead of "fake", meaning that the discriminator should predict such a token not been replaced by the generator. 2) The generator is trained to maximize the likelihood of the groundtruth token rather than trained to fool the discriminator. The latter, according to experiments, performs much worse though the reasons are not stated in the paper.

The authors conducted a series of experiments, notable conclusions includes

- The Replaced Token Detection method benefits from having a loss over all tokens rather than just a subset, evidenced by the fact that if you only use 15% of the tokens(the masked tokens) to compute the loss, the perfromance becomes much worse(GLUE score -3).

- BERT suffers from the pretrain - finetune mismatch in that during finetuning, BERT does not see the token [MASK]. Evidenced by Replace MLM slightly outperforming vanilla BERT(GLUE score +0.2).

6.1 My Thoughts

The proposed method of replaced token detection investigates every token from a given sentence as opposed to MLM which only examines the masked tokens. The discrepancy between pretrain and finetune, though influential in theory, actually makes little difference in GLUE dataset(+0.2). To use this training method on low resource languages could be more effective than high resource languages.

7 LM-BFF

LM-BFF[20] is proposed in **Making Pre-trained Language Models Better Few-shot Learners** in ACL 2021 by Tianyu Gao, Adam Fisch, and Danqi Chen. The major contributions of LM-BFF are

- Proposing a pipeline for automating prompt generation
- Proposing sampling semantically close demonstrations to facilitate learning of Language Models.

LM-BFF is built on the work of **Pattern-Exploiting Training**[26], which transforms various downstream tasks(Question Answering, Text Classification, Named Entity Recognition) into fill-in-the-blank cloze questions.

Another way to let pretrained language models perform a specific task is to use prompts and task demonstrations as the context of the input. GPT-3[11] achieves remarkable performance in few-shot learning.

The authors mentions two problems with GPT-3's in context learning: The demonstration examples are randomly chosen from different classes, which is hard to learn from; The amount of examples are bounded by the models maximum input length. Prompt based learning requires careful prompt engineering. In [25], the authors points out that the quality of prompts can have a huge impact on the performance of language model. To engineer a high quality prompt requires language-specific and task-specific prior knowledge.

7.1 Generating words given template

Therefore, task-agnostic and language-agnostic prompting is at the crux of prompt engineering. The work of LM-BFF aims to automate prompt engineering by using a **discrete template**. For contructing a set of label words of class c given a fixed template, we select all the training examples of class c,

plug them into the language model to generate possible words, and sum the log probabilities for each single candidate of label word and take the top k. Mathematically speaking, given a label word w and template T, its score is computed by

$$\sum_{x_{class}=c} \log P([MASK] = w|T(x_{class}))$$

We manually select k words with the highest scores for each class. The total number of label words are $C * k$.

7.2 Generating template given words

LM-BFF uses the T5[17] model to fill in missing spans in the template. The authors propose three simple meta-templates

$$< Input > \longrightarrow < Span1 > \text{label\_word} < Span2 > < Input >$$

$$< Input > \longrightarrow < Input > < Span1 > \text{label\_word} < Span2 >$$

$$< Input1 > < Input2 > \longrightarrow < Input1 > < Span1 > \text{label\_word} < Span2 > < Input2 >$$

and T5 is used to fill in the missing spans. Somehow these templates are semi-automated because you still need to manually define a meta-template. We select the template for each task(or for each training set)that maximizes the log conditional probability of the template(the sentence) given meta-template. Mathematically speaking, the score of a template given task t is given by

$$\sum_{i=1}^{|T|} \sum_{x_{train} \in t} \log P_{T5}(t_i|t_0, ...t_{i-1}, T(x_{train}))$$

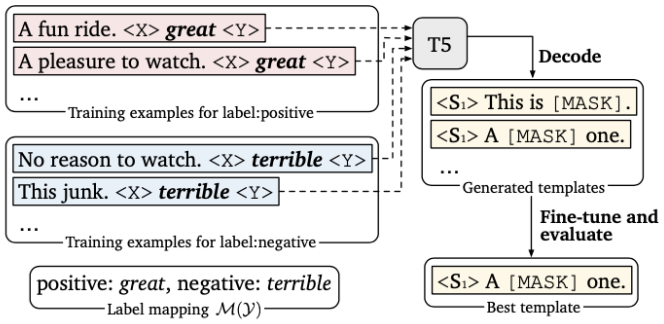

Figure 2: Our approach for template generation.

Figure 9: Template Generation, copied from [20].

7.3 Demonstrations based on similarity

The authors first encode the raw sentences into a sentence level encoder. Here they uses Sentence BERT(or SBERT)[18]. For given query x_{test} and a given label c, for all training examples x_{train} labelled as c, we calculate the cosine similarity of the embeddings of x_{train} and x_{test}. We sample from the top 50% similar sentences for each label as context.

7.4 My Thoughts

Although the experiments shows promising results of the automated search process, there have been various studies showing that a discrete template is sub-optimal compared to "soft prompts"[23][24]. The usage of "template of templates" also limits prompt engineering.

8 Prefix-Tuning

Prefix Tuning[22] is proposed in **Prefix-tuning: Optimizing continuous prompts for generation.** in ACL 2021 by Xiang Lisa Li and Percy Liang. The major contributions of Prefix-Tuning are

- Introducing an lightweight alternative to the state-of-the-art paradigm of finetuning every parameter for a specific task.
- Proposing that gradient based optimization of soft tokens/prompts can be effective as well as discrete tokens.

The problem for finetuning happens when we are using a very large language model, usually over billions of parameters. For each downstream task, we need to store a task-specific copy of the parameters, which is "prohibitively expensive"[22].

Previous attempts to solve this problem often freezes most of the pretrained parameters and only finetune on a smaller subset of parameters. In previous section I have introduced the application of **adapters** in NLP[8], which inserts layers in transformer blocks for finetuning. Nevertheless, adapters still need to train 3.6% of the original parameters, which is huge compared to the amount of finetune parameters used in prefix-tuning(0.1%)

8.1 Prefix Tuning Method

As shown in the picture, prefix tuning appends a learn-able prefix to an already pretrained autoregressive language model. The dimension of the prefix is manually defined to be the same as the transformer dimension and the length of the prefix is a fixed hyper-parameter. The model is pretrained as the same as before but finetuned with additional prefixes.
One notable aspect of prefix tuning is that instead of randomly initializing the prefix embeddings, the author uses a smaller embedding and then use a linear

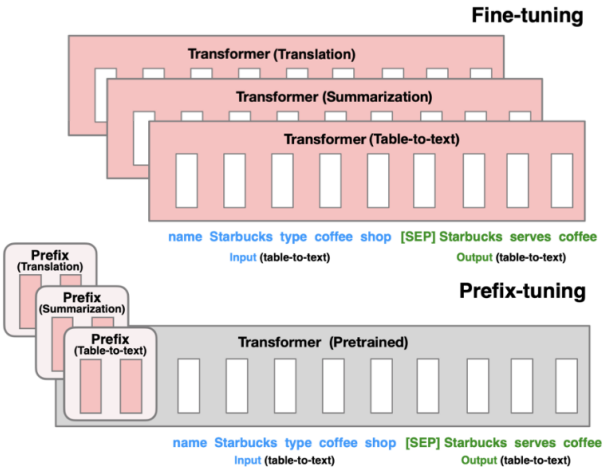

Figure 1: Fine-tuning (top) updates all LM parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the LM parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.

Figure 10: Prefix Tuning, copied from [22]

layer to transform it into the dimension of a transformer block. Quoting from the paper "Empirically, directly updating the P_θ parameters leads to unstable optimization and a slight drop in performance."[22] Also, "Initializing the prefix with activations of real words significantly improves generation."[22].

8.2 My Thoughts

The experiments results evidenced that prefix tuning(adding additional parameters in front of transformer blocks) achieve better results than adapters(adding additional parameters within transformer blocks) for a same pretrained model. Still prefix tuning still slightly underperforms full finetuning(which requires a lot of space and time).
Another problem is that the results are evaluated only on GPT-2(single directional), which might not be the case for other autoregressive models with bidirectional context. The authors also states the superiority in expressive power of continuous soft prompt as opposed to discrete prompts. Admitted that the

expressive power of soft prompts are better, they require a lot of computational power to search of a optimal prompt, even if the prompts were discretely initialized. Whereas the GPT-3 style in context transfer might yield better results.

9 GPT understands, too(P-Tuning)

This paper was published in 2021.3 by Xiao, Liu et al. The major contributions of this paper are

- Proposing trainable continuous prompt embeddings as a general method to improve LM's transfer learning ability.
- Showing that continuous prompt can achieve comparable, if not better results than discrete prompts proposed in [26].

P-Tuning defines a template T as

$$T = \{[P_{0:i}], x, [P_{i+1:m}], y\}$$

where x is [MASK] and y is [CLS](The classification head). A discrete prompting methods maps every token in the template to the word embedding. P-Tuning treats each P_i as psudo tokens and maps the template to

$$\{h_0, ..., h_i, \mathbf{e}(x), h_{i+1}, ..., h_m, \mathbf{e}(y)\}$$

Here $h_i \in R^d$ where d is the dimension of embedding vector(768 for BERT). The embedding might not corresponds to a discrete token.

One problem for this schema is that the target word embedding is discrete and while the prompt is continuous. The authors also states that if the embeddings are randomly initialized, it could easily fall into local minima. Also to associate each individual prompt token, the author uses a bidirectional LSTM network to encode the embeddings and then passes them through a single linear layer with ReLU activation function. (In Prefix Tuning[22], the author uses only a single layer MLP to transform low dimensional embeddings to a higher dimension).

9.1 My Thoughts

The paper is similar to [22] to propose continuous prompts, which achieve better results than Manuall Prompts + Finetuning(which is the case in [26]). In the following table, we can see that the gain on Language Models pretrained in a bidirectional denoising objective(BERT) is not as significant as the gain on unidirectional autoregressive models(GPT-2). It is possible for an ensemble of high quality manual prompts to achieve better performance.

Model	MP	FT	MP+FT	P-tuning
BERT-base (109M)	31.7	51.6	52.1	52.3 (+20.6)
-AutoPrompt (Shin et al., 2020)	-	-	-	45.2
BERT-large (335M)	33.5	54.0	55.0	54.6 (+21.1)
RoBERTa-base (125M)	18.4	49.2	50.0	49.3 (+30.9)
-AutoPrompt (Shin et al., 2020)	-	-	-	40.0
RoBERTa-large (355M)	22.1	52.3	52.4	53.5 (+31.4)
GPT2-medium (345M)	20.3	41.9	38.2	46.5 (+26.2)
GPT2-xl (1.5B)	22.8	44.9	46.5	54.4 (+31.6)
MegatronLM (11B)	23.1	OOM*	OOM*	**64.2** (+41.1)

* MegatronLM (11B) is too large for effective fine-tuning.

Figure 11: Experiments comparing MP+FT to P-Tuning, copied from [23]

10 Null Prompting

Null Prompting is proposed in **Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models**[21]. It aims to solve two major problems in transfer learning in NLP:

1. prompt-based finetuning([26], [20], [23]) have large memory requirements as they need to update all the model parameters as well as continuous prompt parameters if needed.

2. In-context learning[11] "requires significant prompt engineering"[21], despite the fact that it does not require us to update any parameter of the model, essentially reusing the pretrained model in its "as is" form.

The major contributions of the paper are

- Proposing a novel prompting method that does not require prompt engineering(Although verbalizer engineering is required to achieve the best result).

- Experimenting with lightweight finetune alternatives, although on relatively small(number of params ¡ 500M) models.

- Discussing the significance of prompting and prompt engineering in natural language understanding tasks.

The null prompting methods is simple and straightforward. For example:

Prompt-Based Finetuning (Previous Works)	{Will GST affect the price level in India?}$_1$? [MASK], I want to know {Will GST effect the price level in India?}$_2$
Null Prompts	{Will GST affect the price level in India?}$_1$ {Will GST effect the price level in India?}$_2$ [MASK]

The above example removes all the prompting words "I want to know" and instead uses an [MASK] token to predict the answer. To answer the question of "Will GST affect the price level in India", one only need to restate the question

followed by a [MASK] token. Note that the verbalizer "Yes" still need to be manually defined. For a general question(Yes-no question) this can be simple. However for a classification question in Named Entity Recognition, for example, { What is the entity of Manchester in Manchester United} and the answer should be {Organization} instead of {Location} Since Manchester United is a Football Club. In this case the verbalizer word label {*location*} need to be manually defined.

You can see a comparison of different prompting methods in the following table.

Method	Finetuned Params	Prompt Design	Few-shot
AUTOPROMPT (Shin et al., 2020)	None	Learned (Discrete)	✗
Prompt Tuning (Lester et al., 2021)	Prompt Token Embeds	Learned (Continuous)	✗
OPTIPROMPT (Zhong et al., 2021)	Prompt Token Embeds	Learned (Continuous)	✗
Soft Prompts (Qin and Eisner, 2021)	All Contextualized Embeds	Learned (Continuous)	✗
GPT-3 (Brown et al., 2020)	None	Manual	✓
PET (Schick and Schütze, 2021a)	All	Manual	✓
LM-BFF (Gao et al., 2021)	All	Learned (Discrete)	✓
P-Tuning (Liu et al., 2021)	All + Prompt Token Embeds	Learned (Continuous)	✓
Null Prompts + Bitfit (Ours)	Bias Terms	None	✓

Figure 12: Comparison of different prompting methods, table copied from [21]

The experiment results shows that in ALBERT(223M)[15] and RoBERTa(330M)[9]

- **Manual Prompts** achieves the best performance, although a careful selection of prompts is required since prompts with out engineering perform worse.
- **Null Prompts + Manual Verbalizers** perform roughly the same, if not better than Prompt Tuning.
- **Bias-Only Finetuning** performs the best than other lightweight finetuning methods, and is comparable, if not better to finetuning all parameters.

10.1 My Thoughts

It is extremely important to notice that the result are obtain only from two models that are under 500M parameters. Which might not be the lase in either 1)Large Models($\geq$ 1B params) or 2)Autoregressive Models, therefore the results should be closely examined case by case(or model by model) in order to come up with a conclusion of different prompting methods.

11 Learning How to Ask: Querying LMs with Mixtures of Soft Prompts

This paper was published in NAACL 2021 by Guanghui Qin and Jason Eisner.(Hopefully I can enroll in his class and do research with him :p) It proposes an ensemble of soft prompts which can be optimized by gradient descent. In the abstract, the author makes two claims: 1) Their method "hugely outperforms" previous methods and 2) "random initialization is nearly as good as informed initialization."[24]

The major contributions of this paper are:

- Examining prompts from a semantic perspective and concluding that soft prompts are more expressive.
- Proposing soft prompting as an alternative to manually crafted prompts, which are particularly useful in **extracting relational knowledge from language models**.

In the first section, the authors mentions that discrete prompts can be "misleading, ambiguous, or overly specific"[24], which is solved by the expressiveness of continuous prompts.

The implementation is straightforward, as in [20], the authors replaces the embeddings of discrete prompt tokens with random initialized soft embeddings. While [20] and [23] uses MLP and BiLSTM, respectively, to learn the prompt embeddings, [24] directly optimizes the prompted embeddings and adds a small perturbation.

Experiment results shows that on different knowledge probing benchmarks(T-REx, Google-RE), soft prompting method outperforms previous models(LAMA, LPAQA and AutoPrompt)

11.1 My Thoughts

The paper mainly focused on probing knowledge from large language models using prompts. There are other NLU tasks that are more about understanding semantics, like Question Answering, Sentence Classification which may require different prompting methods. The superiority of soft prompting on other tasks still needs to be carefully examined

12 $(IA)^3$

$(IA)^3$ is proposed in **Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning**[27], to be published at ICLR 2022. The major contributions of this paper are:

	FT	PEFT	ICL
Example	BERT[6], T5[17]	Adapters[7], P-Tuning[23], Prefix-Tuning[22]	GPT-3[11]
Update	All parameters	Few parameters	No gradient update
Requirements	Computation Power	Prompt Engineering Model Design	Manually giving context Long process time

- Proposing a new Parameter Efficient Fine-Tuning(PEFT) method by element-wise multiplying a vector to K,V in attention blocks and the vector after the first ReLU(GeLU)[1] in feed-forward network blocks.
- Introducing three loss functions(two auxiliary loss and one Cross Entropy loss) to not only reward the model to produce correct outputs, but also punish them more severely for producing clearly incorrect ones.

The author first compares three state-of-the-art methods in inducing large language models to perform a specific task. **Finetuning(FT)**, which is the *de facto* method performs gradient updates on **all parameters** in the model aligned to a specific task. While it achieves the best performance, it consumes a lot of memory and needs a substantial amount time to train. Storing 10-100GB of a checkpoint for a single task is undesirable. Another method to let LLMs perform task is **In Context Learning(ICL)**. GPT-3[11] concatenates demonstrations in front of our task input. For example, if our task is to perform English - French translation. ICL preappends a few examples: *red - rouge, cheeze - fromage, swimming - natation* to our actual input. ICL achieves great performance and an astounding ability to perform few-shot and/or zero shot tasks. There are several drawbacks to ICL, notably worse performance than FT and the huge time consumption to process the context, not to mention the variance induced from manually crafted examples. Recently, another category emerged in only training a relatively small proportion(usually 0.1%) of the parameters. Adapters[7], P-Tuning[23], Prefix-Tuning[22] and Soft Prompt Ensemble[24]are all ways of **Parameter Efficient Fine-Tuning(PEFT)**. However, from the figure we can see that most PEFT could only achieve "comparable" performance on transfer learning tasks.

The proposed PEFT method proposed in `(IA)`^{3}is simple: adding three additional vectors v_{key}, v_{value} and v_{ff} to each transformer block. Following the notation in [4], the attention output(disregarding the multi-head) is

$$\text{softmax}\left(\frac{Q(v_{key} \odot K^\top)}{\sqrt{d_k}}\right)(v_{value} \odot V)$$

[1]The original Transformer[4] uses ReLU and BERT[6] uses GeLU

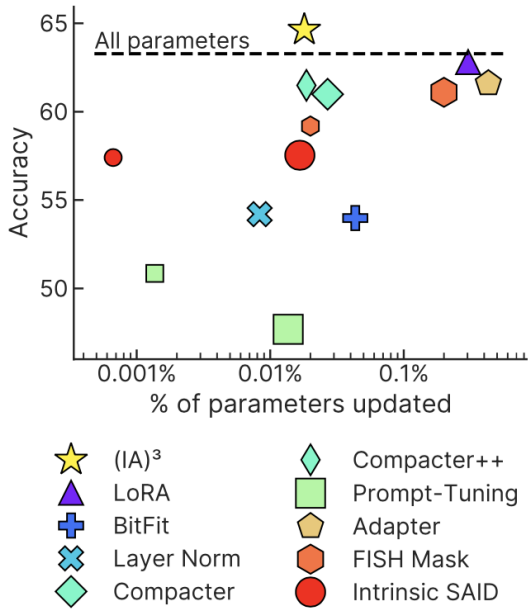

Figure 13: Experiments comparing different PEFT and FT, copied from [27]

The output of the Feed-Forward Network after scaling is:

$$(v_{ff} \odot \sigma(W_1 x))W_2$$

where σ is the nonlinear activation function.
The author also introduces two auxiliary loss function to perform rank classification. The standard Cross Entropy loss is given by

$$L_{CE} = -\frac{1}{T}\sum_{t=1}^{T} \log P(y_t|x, y_{<t})$$

Where T is the length of our input sentence.

The unlikelihood loss that penalizes incorrect examples to rank high is

$$L_{UN} = -\frac{\sum_{n=1}^{N}\sum_{t=1}^{T(n)} \log(1 - P(\hat{y}_t^{(n)}|x, \hat{y}_{<t}^{(n)})}{\sum_{n=1}^{N} T^{(n)}}$$

where n denotes the n'th sampled incorrect output.
Moreover, as Language Models tend to assign lower probabilities to longer sentences, we normalize the probability by

$$\beta(x, y) = \frac{1}{T}\sum_{t=1}^{T} P(y_t|x, y_{<t})$$

Then we maximize the log probability of the correct answer softmaxed with sampled incorrect answers.

$$L_{LN} = -\log \frac{\exp(\beta(x, y))}{\exp(\beta(x, y)) + \sum_{n=1}^{N} \exp(\beta(x, \hat{y}^{(n)}))}$$

The loss function is the unweighted sum of the three:

$$L = L_{CE} + L_{UL} + L_{LN}$$

12.1 My Thoughts

I believe that the only downside of PEFT to standard all parameter finetuning is their performance since PEFT is a strictly more lightweight version of FT. PEFT method `(IA)`^{3}achieves better performance than FT in a low resource context. The generalizability of `(IA)`^{3}is to be examined. I would like to try it on other models than T0.

13 REALM

REALM is proposed in **REALM: Retrieval-Augmented Language Model Pre-Training**[14], published at ICML 2020. It proposes a pipeline that incorporates relevant document retrieval to augment knowledge probing in large language models. It is an modification to the masked language modeling pretrain method proposed in BERT[6], as well as a finetune method used to query real world knowledge inside language models. The paper's major contributions are:

- Proposing a novel pretrain method by using document retrieval to allow language models attend to factual knowledge in documents.
- Solving the major problems that arises from measurement of document-sentence similarity, and giving out explanations of their methodology.

As we know, large language models stores factual knowledge within its parameters. We would like to explicitly retrieve the knowledge to perform Question-Answering(QA) or other tasks that require knowledge probing. REALM augments the **pre-train** objective with a learned knowledge retriever, so that during inference, the model can retrieve relevant documents from a large corpus.

13.1 Pipeline and Implementation Choices

Generally speaking, Language Models aims to optimize the conditional distribution of the next token given a context. $p(y|x)$. REALM first retrieve useful document from a knowledge corpus Z, which is measured by $p(z|x)$, and $z \in Z$. Then the model is pre-trained to generate the output y conditioned on z and x, that is $p(y|z, x)$.

The retriever is defined using a inner product model.

$$p(z|x) = \frac{\exp f(x,z)}{\sum_{z'} \exp f(x,z')}$$

$$f(x,z) = \text{Embed}_{\text{input}}(x)^\top \text{Embed}_{\text{doc}}^\top(z)$$

Here $\text{Embed}_{\text{input}}$ and $\text{Embed}_{\text{doc}}$ are embedding functions that maps an input sentence x and a document z to a vector $v \in R^d$, the relevance score is computed by the dot product of the embeddings, then all scores are softmaxed. The embedding is a linear projection of the BERT [CLS] token of the input and document.

After a document is retrieved, the model concatenates the input x and the document z and passes them into a transformer model. During pre-training, the authors uses the same objective of BERT[6], which predicts the word/token at [MASK] positions. During fine-tuning, assuming that the answer can be explicitly found in a document rather than being generated, we can define $p(y|z,x)$ as:

$$p(y|z,x) \propto \sum_{s \in S(z,y)} \exp(\text{MLP}([h_{\text{START}(s)}; h_{\text{END}(s)}]))$$

$$h_{\text{START}(s)} = BERT_{\text{START}(s)}(join_{BERT}(x, z_{body}))$$

$$h_{\text{END}(s)} = BERT_{\text{END}(s)}(join_{BERT}(x, z_{body}))$$

where $S(z,y)$ denotes the set of spans matching y in z. Honestly I am confused by the notations here but I think it means that given a set of spans, we use a BERT model to encode the starting token and ending token into vectors, concatenate them and then feed them into a feed forward network to compute the score of a span. The "score" of a particular set of spans(set of answers) is summed by the score of each single span.

To implement REALM, two of the biggest challenges are to compute the softmaxed document score within a huge amount of documents. So the author pre-computes the scores of the document and performs a Maximum Inner Product Search(MIDS). However, as the parameters of the document retriever(which is a BERT model) gets updated, the pre-computed vectors, and therefore the dot product scores becomes stale. The author solve(or attempts to solve) this problem by asynchronously updating the retriever(BERT encoder) parameters Once in a while, the model sends the update of the retriever parameters to the document vector pre-computing agency and let it update the stale vectors.

13.2 My Thoughts

This paper is the pioneering work of using retrieval to augment language model pretraining to let it perform better in knowledge probing. I believe that some of its engineering choices could be better(e.g. selecting the encoding of start and end of spans). The generalizability of this method is to be examined. Using retrieval only during fine-tuning to get answer from a large document or collection of document is of great potential.

References

[1] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[2] Dan Hendrycks and Kevin Gimpel. "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units". In: *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415.

[3] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. "Learning multiple visual domains with residual adapters". In: *CoRR* abs/1705.08045 (2017). arXiv: 1705.08045. URL: http://arxiv.org/abs/1705.08045.

[4] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[5] Zihang Dai et al. "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context". In: *CoRR* abs/1901.02860 (2019). arXiv: 1901.02860. URL: http://arxiv.org/abs/1901.02860.

[6] Jacob Devlin et al. "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)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[7] Neil Houlsby et al. "Parameter-Efficient Transfer Learning for NLP". In: *CoRR* abs/1902.00751 (2019). arXiv: 1902.00751. URL: http://arxiv.org/abs/1902.00751.

[8] Neil Houlsby et al. "Parameter-Efficient Transfer Learning for NLP". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

[9] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[10] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *CoRR* abs/1906.08237 (2019). arXiv: 1906.08237. URL: http://arxiv.org/abs/1906.08237.

[11] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[12] Massimo Caccia et al. "Language GANs Falling Short". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=BJgza6VtPB.

[13] Kevin Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *ICLR*. 2020. URL: https://openreview.net/pdf?id=r1xMH1BtvB.

[14] Kelvin Guu et al. "Retrieval Augmented Language Model Pre-Training". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 3929–3938. URL: https://proceedings.mlr.press/v119/guu20a.html.

[15] Zhenzhong Lan et al. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=H1eA7AEtvS.

[16] Mike Lewis et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://aclanthology.org/2020.acl-main.703.

[17] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

[18] Nils Reimers and Iryna Gurevych. "Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2020. URL: https://arxiv.org/abs/2004.09813.

[19] Zhengxiao Du et al. "All NLP Tasks Are Generation Tasks: A General Pretraining Framework". In: *CoRR* abs/2103.10360 (2021). arXiv: 2103.10360. URL: https://arxiv.org/abs/2103.10360.

[20] Tianyu Gao, Adam Fisch, and Danqi Chen. "Making Pre-trained Language Models Better Few-shot Learners". In: *Association for Computational Linguistics (ACL)*. 2021.

[21] Robert L. Logan IV et al. *Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models*. 2021. arXiv: 2106.13353 [cs.CL].

[22] Xiang Lisa Li and Percy Liang. "Prefix-Tuning: Optimizing Continuous Prompts for Generation". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4582–4597. DOI: 10.18653/v1/2021.acl-long.353. URL: https://aclanthology.org/2021.acl-long.353.

[23] Xiao Liu et al. "GPT Understands, Too". In: *arXiv:2103.10385* (2021).

[24] Guanghui Qin and Jason Eisner. "Learning How To Ask: Querying LMs with Mixtures of Soft Prompts". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Best Short Paper Award. Online, June 2021, pp. 5203–5212. URL: http://cs.jhu.edu/~jason/papers/#qin-eisner-2021.

[25] Laria Reynolds and Kyle McDonell. "Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm". In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI EA '21. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380959. DOI: 10.1145/3411763.3451760. URL: https://doi.org/10.1145/3411763.3451760.

[26] Timo Schick and Hinrich Schütze. "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*. Online: Association for Computational Linguistics, Apr. 2021, pp. 255–269. DOI: 10.18653/v1/2021.eacl-main.20. URL: https://aclanthology.org/2021.eacl-main.20.

[27] Haokun Liu et al. "Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning". In: *arXiv preprint arXiv:2205.05638* (2022).