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

Tianjian Li

March 2022

1 BART

BART[12] is work by Facebook published at ACL 2020. It aims to generalize BERT [5] and GPT[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.

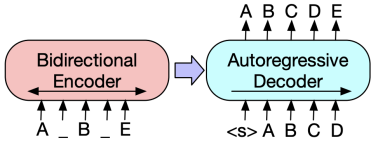

Figure 1: BART

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

- Token Masking: Similar to BERT[5].
- 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.

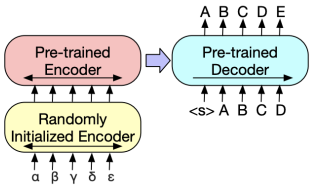

Figure 2: Illustration of Machine Translation Task

2 PET

PET[21]: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"[21] 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.

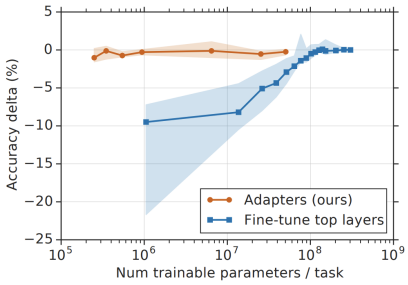

Figure 3: Fine-tuning vs Adapters from [7]

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].

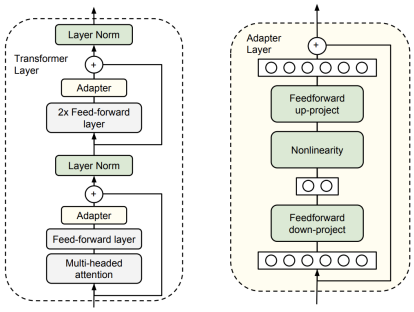

Figure 4: Illustration of adapter layers from [7]

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[6] 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.

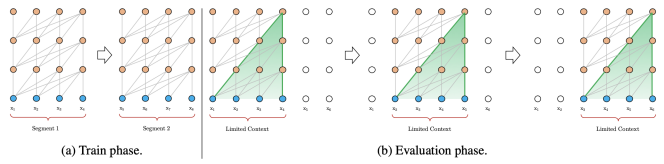

Figure 5: Vanilla Transformers, figure taken from [6]

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.

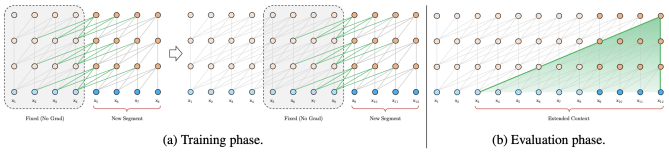

Figure 6: Transformers-XL, figure taken from [6]

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[5], respectively.

Type	AR	AE
Example	Transformers[4]	BERT[5]
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[9] 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.

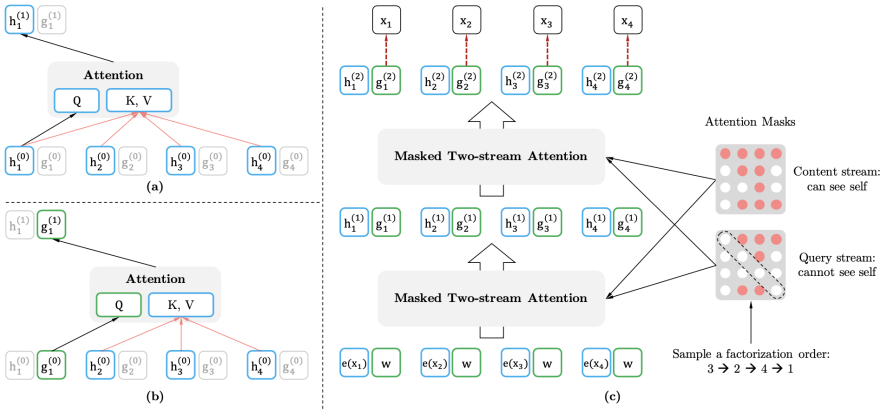

Figure 7: XLNet model architecture, figure from[9].

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_{zt}, 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 contex-

tual information $x_{z<t}$ and the position z_t, but not the content x_{z_t}.

Therefore XLNet[9] 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[6], 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[15].

6 LM-BFF

LM-BFF[16] 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**[21], 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[10] 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 [20], 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.

6.1 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 $C * k$.

6.2 Generating template given words

LM-BFF uses the T5[13] 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

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

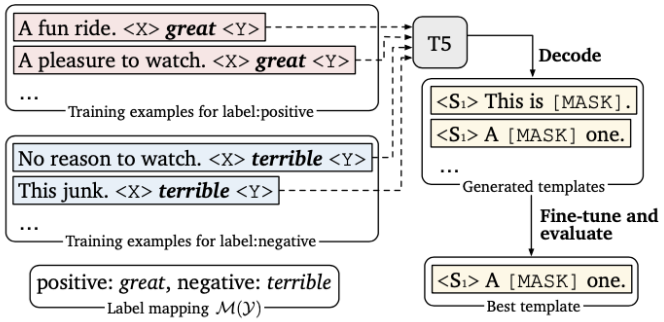

Figure 2: Our approach for template generation.

Figure 8: Template Generation, copied from [16].

6.3 Demonstrations based on similarity

The authors first encode the raw sentences into a sentence level encoder. Here they uses Sentence BERT(or SBERT)[14]. 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.

6.4 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"[18][19]. The usage of "template of templates" also limits prompt engineering.

7 Prefix-Tuning

Prefix Tuning[17] 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"[17].

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%)

7.1 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 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."[17] Also, "Initializing the prefix with activations of real words significantly improves generation."[17].

7.2 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

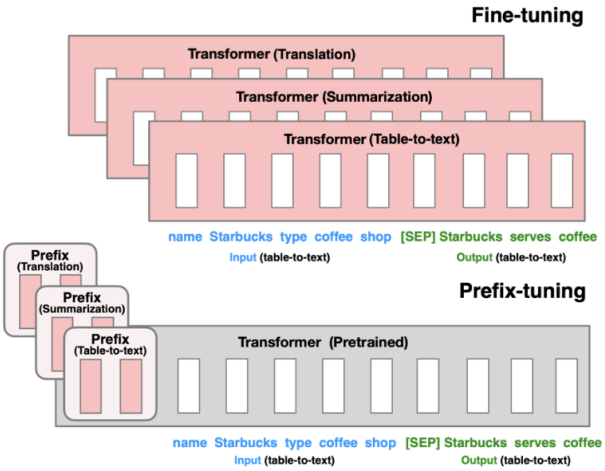

Figure 1: Fine-tuning (top) updates all LM parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the LM parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix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

Figure 9: Prefix Tuning, copied from [17]

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.

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] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[6] 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.

[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] 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.

[10] 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.

[11] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.