It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners

Shadi Hamdan

I. Introduction

With the rise of deep learning, there has been a very sharp in demand for large amounts of data. Moreover, there has also been a strong trend towards larger models with more and more parameters. GPT-3 [1], with a whopping 175 billion parameters, showed impressive performance on many natural language benchmarks including SuperGLUE [2], while using just 32 labeled examples. This prompted more research into the usage of language models for few-shot learning. By reformulating natural language tasks into mask-filling tasks, Pattern-Exploiting Training [3] [4] can get better results than GPT-3 in the SuperGLUE benchmark [2] with approximately 0.1% of the parameters while still using just 32 labeled examples. This report is replication report of the original paper with the same title. The work was successfully replicated in Julia [5] using the Deep Learning framework Knet [6].

II. Background

A. Language Models

One of the most common ways to tackle natural language tasks is by fine-tuning masked language models. Modern language models are deep neural networks that are trained to model human language. Typical language models, such as GPT-3 [1], are trained to predict the next token conditioned on the previous tokens. These types of models are called auto-regressive models.

Masked language models, on the other hand, are trained to predict tokens conditioned on both previous and following tokens. This is done by masking. During training, masking is applied to training sentences by replacing a random subset of input tokens with a special reserved mask token. The model is then trained to predict this masked token. Unlike auto-regressive language models, masked language models can utilize context from both the left and the right. This modification has resulted in a noticeable improvement and soon many different variants were introduced.

Since the only training data required is monolingual data, there is no shortage of such data to pretrain language models. Language models are trained on large amounts of data from a multitude of sources, ranging from tens of gigabytes of uncompressed text for BERT [7] and ALBERT [8] to hundreds of gigabytes of text for RoBERTa [9], DeBERTa [10], and T5 [11].

B. Finetuning

After pre-training language models are trained on rich and diverse data, they are fine-tuned on downstream tasks such as text classification or natural language inference. During fine-tuning, additional layers are appended to the top of the language model in order to get an output that is compatible with the task at hand. For text classification, for example, a common approach is to add an affine transformation followed by a projection to the number of labels of the task. This inputs to this additional layer could be the averaged outputs of the model for each token in the input. The model is then fine-tuned for a few epochs on the task. Using similar methods for other types of tasks, masked language models have shown powerful performance and transfer learning and are now the de facto standard for many tasks.

III. Pattern-Exploiting Training

The common of approach of finetuning proves to be powerful and generalizes to many tasks. However, this performance does not translate well to smaller datasets [4], and results are generally dwarfed by gigantic models such as GPT-3 [1]. However, by modifying the approach slightly, it is possible to outperform GPT-3 using models that are orders of magnitude smaller. The main idea of Pattern-Exploiting Training is to reformulate language tasks to mask filling tasks. To do so, two components are needed: a pattern p and a verbalizer v. The pattern p takes an input instance x and reformulates it as a single string with at least one mask tokens. The verbalizer v takes the label of the instance y and converts it to a list of tokens equal in length to the number of mask tokens. An example of a pattern-verbalizer pair for a boolean question dataset:

- `p(x)` → `Correct or not? x. [MASK]`
- `v(true)` → `Yes`
- `v(false)` → `No`

Pattern-Exploiting Training (PET) consists of three steps: training an ensemble of masked language models (MLM) on the initial dataset, using the ensemble to label unlabeled data with soft labels, then finally using distillation to train a final model on soft-labeled data. By posing the tasks as masked filling tasks, they are much closer to what the MLMs have been pretrained to do. Moreover, unlike traditional methods of finetuning, this requires *no extra parameters* to be initialized. Thus, this method utilizes the information learned by the language model to a greater extent.

IV. Dataset

For this work, the SuperGLUE language understanding dataset [2] was used. It consists of 8 different datasets which comprise multiple language tasks such as question answering,

Dataset	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC
Task	Q/A	NLI	Q/A	Q/A	Q/A	NLI	WSD	coref.
|Train|	32	32	32	32	32	32	32	32
|Unlabeled|	9427	19999	400	456	17965	19998	5428	554
|Dev|	3270	57	100	953	10k	278	638	104
|Test|	3245	250	500	1800	10k	300	1400	146

TABLE I
THE DIFFERENT TASKS IN THE FEWGLUE DATASET, A SUBSET OF THE SUPERGLUE DATASET. Q/A STANDS FOR QUESTION ANSWERING, NLI FOR NATURAL LANGUAGE INFERENCE, WSD FOR WORD SENSE DISAMBIGUATION, AND COREF. FOR COREFERENCE (PRONOUN) RESOLUTION.

co-reference resolution, natural language inference, and word sense disambiguation. Since SuperGLUE is very widely used to benchmark the performance of language models, including GPT-3, it provides a powerful frame of reference of the performance of this approach which can directly be compared to other approaches.

SuperGLUE tasks have training sets that are much larger than 32 examples. For their work, Schick et al. [4] form a smaller dataset that is a subset of SuperGLUE. To create this dataset, they randomly sample 32 examples from the training sets. For the remaining instances, they strip the labels and use it as unlabeled data for the second step of PET. This smaller dataset is named FewGLUE and the authors have publicly shared it[1] to make comparisons between approaches in the future more fair.

V. IMPLEMENTATION

In the original implementation, the masked language model that was used was used was ALBERT [8]. Since it was not implemented in Knet [6] yet, the first step was to implement it and use the publicly available weights.

A. ALBERT

ALBERT [8] is a variant of masked language models. Like most masked language models, ALBERT's backbone is a transformer [12] encoder. The unique feature of ALBERT is that all the encoder blocks are completely shared. This allows for the hidden vector size to be increased by a large factor while still being similar in size compared to other masked language models. In the largest version of ALBERT, the hidden size is Other than that, there are a few modifications to the regular encoder block of the transformer. For the implementation, the HuggingFace library [13] was used for reference.

1) Embeddings: By increasing the hidden size, the size of the embedding matrix becomes very large. To alleviate this, ALBERT uses a *embedding size* that is much smaller the *hidden size*. After the embedding layer, an affine operation is applied to the embedding vectors to project them to the hidden size.

Classical Embedding: $V \times H$ Parameters
ALBERT Embedding: $V \times E + E \times H$ Parameters

Where V is the size of the vocabulary, E is the embedding size, and H is the hidden dimension size. When E is much smaller than V, this results in a considerable decrease in the number of parameters, without affecting performance.

2) Shared Encoder Layers: In ALBERT, all the encoder layers share the same weights. By sharing the weights, the number of parameters is significantly reduced. In the largest version of ALBERT, there are 12 encoder layers with shared weights. Since the parameters are greatly reduced by this modification, this allows for the hidden layer dimension to be greatly increased while keeping the model size reasonable. In the largest version of ALBERT, the hidden dimension is 4096, and the feed-forward dimension in the encoder block is 16384.

B. Ensemble Training

In the first step of PET, different combinations of patterns are verbalizers are used to train an ensemble of models. Since the results from a single iteration vary widely with different runs and different seeds, each pattern-verbalizer pair is used to train 3 models. This helps alleviate the noise in the resultant models. The optimizer used is Adam with weight decay. Weight decay is applied to all the parameters aside from the biases and layer normalization layers. Additionally, a linear decay weight scheduler is used during training to decrease the learning rate gradually to 0 by the end of training. The loss function is cross entropy loss for just the mask tokens. The model is trained for 250 updates on the 32 examples. Shuffling the training data is necessary for good performance.

C. Labeling Unlabeled Data

After the models are trained, each model is individually used to evaluate and label the *unlabeled* set. In this stage, the labels are soft, i.e. they are not strictly one label or the other. In soft labeling, the output scores of the model for each label are used as the soft labels. This helps preserve information about how confident each model is about its prediction. After each instance has been labeled by all the models, the labels from the different models are then combined to create the final labels to be used in the next step. To combine the labels, a weighted mean of the soft labels is used. The weights for the labels of each model are the accuracies of the models with their respective pattern-verbalizer pairs on the training set *before training*. After training, the models typically overfit the training set, and the accuracy doesn't provide any useful information about each model since it's 100% for all the models. Equation (1) is the weighted mean equation, where $q(y|x_i)$ is the resultant labels from the weighted average for

[1]https://github.com/timoschick/fewglue

Model	Params (M)	BoolQ (Acc.)	CB (Acc.)	COPA (Acc.)	RTE (Acc.)	WiC (Acc.)	WSC (Acc.)	MultiRC (EM/F1)	ReCoRD (Acc.)
GPT-3 Med	350	60.6	58.9	64	48.4	55	60.6	11.8/55.9	77.2
GPT-3	175,000	77.5	82.1	92	72.9	**55.3**	75	32.5/74.8	**89**
PET	223	79.4	85.1	95	69.8	52.4	**80.1**	37.9/77.3	86
iPET	223	**80.6**	**92.9**	**95***	**74**	52.2	**80.1***	33 / 73	86*
PET.jl	223	79.2	85.7	94	**74**	51.7	78.8	**38/77**	84.6
iPET.jl	223	80.4	84	94*	73.6	50.6	78.8*	**38/77**	84.6*

TABLE II
THE RESULTS OF GPT-3, GPT-3 MEDIUM, AS WELL AS PET AND IPET. REPLICATION RESULTS ARE APPENDED WITH .JL, WHICH IMPLIES THE USAGE OF JULIA. THE PARAMS COLUMN REFERS TO THE NUMBER OF PARAMETERS IN THE MODEL. TASKS MARKED WITH AN ASTERISK (*) ARE RESULTS THAT ARE DIRECTLY FROM PET

the i^{th} instance, w_j is the accuracy of model and pattern-verbalizer pair j, and $s_j(y|x_i)$ is the scores or the soft labels of the j^{th} model for the i^{th} instance.

$$q(y|x_i) = \exp \sum_j (w_j s_j(y_i|x_i)) \tag{1}$$

D. Training The Final Classifier

After the soft labels are combined to form the final training set, the final model is trained on the softly labeled data. Unlike the first step, the final model is trained using the conventional approach of appending layers to a pretrained language model. Since the training set is significantly larger than 32 instances, the final issue does not face the same generalization issues that arise when training on a small amount of data. This process is called distillation and has been outlined in detail in (Hinton et al. [14]). Since the labels in this case are soft, KL-Divergence is used to compute the loss. It is very similar to cross entropy loss:

$$p_i = Softmax(\frac{p(y|x_i)}{T}) \tag{2}$$

$$q_i = Softmax(\frac{q(y|x_i)}{T}) \tag{3}$$

$$KL(p_i|q_i) = \sum_c^K p_i \log(\frac{p_{i,c}}{q_{i,c}}) \times T^2 \tag{4}$$

The divergence loss for the i^{th} instance is denoted by $KL(p_i|q_i)$, where p_i is the output of the classifier model for instance x_i and q_i is the target soft labels for the instance. p_i and q_i are vectors of size K, and $p_{i,c}$ is the smoothed predicted probability for instance x_i and the c^{th} class. Temperature parameter T is a hyper-parameter and is used to smooth the labels [15].

E. Iterative PET

In iterative PET, step 1 and 2 of PET are repeated multiple times before training the final model in step 3. After the second step, the softly labeled data is appended to the original training set and used to train another ensemble of models, starting from step 1. Each step of model ensembles is called a generation. For more robustness, the new data for the j^{th} model in the i^{th} generation, denoted $model_{i,j}$, is created from the softly labeled data from N other models in generation $i-1$, where $k! = j$ for all N models $model_{i-1,k}$. By using models with different patterns and verbalizers to append the data for a model, the results become more stable and robust after multiple generations.

VI. REPLICATION

The approach was replicated exactly as in Schick et al. [3] [4]. The patterns for all the datasets can be found in the original paper. During replication, the following issues were found with the original code and model.

A. Projection Bias

In the original implementation of ALBERT, the biases of the projection layer, which projects from hidden vectors to the vocabulary do not load properly, and are entirely zeroes when initialized from pretraining weights. Initial experiments with fixing this issue and loading the weights properly in this implementation showed slight improvement in the results of the first step of PET which did not translate to the final results. Due to the slight difference in performance, the biases were loaded exactly as in the problematic implementation to remain true to the original implementation. Experiments with this can be future work.

B. Token Type Embeddings

In traditional transformers, the embedding layer has both positional and token embeddings. In addition masked language models usually introduce another type of embedding called token type embeddings. In the original PET implementation, these tokens were not utilized. Experimenting with using these embeddings has not been carried out due to time limitations and are left for future work.

VII. RESULTS

The results of the replication attempt compared to the original results and GPT-3's results can be seen in Table II. For most tasks, the results are within the margin of error of the original implementation. One noticable difference is in CB, where the original implementation of iPET outperforms iPET.jl. However, this is still within the margin of error, since CB only has 56 instances in the Dev set. For ReCoRD, COPA, and WSC, the results in Table II for iPET are the same as PET, since iPET was not used due to the limited size of the unlabeled data.

REFERENCES

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[2] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," 2019.

[3] T. Schick and H. Schütze, "Exploiting cloze questions for few shot text classification and natural language inference," 2020.

[4] T. Schick and H. Schutze, "It's not just size that matters: Small language models are also few-shot learners," *ArXiv*, vol. abs/2009.07118, 2020.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, p. 65–98, Jan 2017. [Online]. Available: http://dx.doi.org/10.1137/141000671

[6] J. D. Yuret, "Knet : beginning deep learning with 100 lines of," 2016.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Proceedings of the 2019 Conference of the North*, 2019. [Online]. Available: http://dx.doi.org/10.18653/v1/N19-1423

[8] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2019.

[9] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[10] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," 2020.

[11] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2019.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[13] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2019.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[15] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," 2017.

APPENDIX

A. Training Hyperparameters

The hyperparameters used for the replication are in the table in the following page in Table III.

Task	Learning Rate	Updates (Step 1)	Batch size	Temperature	Weight Decay	Updates (Step 3)	Max Input Length
BoolQ	1e-5	250	16	2	0.01	5000	256
CB	1e-5	250	16	2	0.01	5000	256
COPA	1e-5	250	16	2	0.01	5000	96
RTE	1e-5	250	16	2	0.01	5000	256
WiC	1e-5	250	16	2	0.01	5000	256
WSC	1e-5	250	16	2	0.01	5000	128
MultiRC	1e-5	250	16	2	0.01	5000	256
ReCoRD	1e-5	250	16	2	0.01	5000	512

TABLE III
THE HYPERPARAMETERS USED TO REPLICATE THE ORIGINAL PAPER'S RESULTS.