

Selectively Editable Language Models

Stanford CS224N Custom Project

Spencer Braun

Department of Statistics
Stanford University
spencerb@stanford.edu

Abstract

Pretrained language models abound in NLP applications, but they are largely static in time and retain stale representations of the world around them. We seek to selectively edit knowledge learned by a language model without affecting its outputs on unrelated samples. Specifically, we explore approaches that alter model understanding of named entities through novel training techniques. We build on methods first developed in general Model-Agnostic Meta-Learning (MAML) frameworks, which allow us to train model parameters on base language model objectives as well as a secondary "adaptability" task. Our experiments demonstrate that this technique improves knowledge editing with less performance degradation on unrelated samples than standard fine-tuning approaches.

External Mentor: Eric Mitchell, Department of Computer Science, em7@stanford.edu
Class Mentor: Megan Leszczynski

1 Introduction

Neural network utility and applications have grown exponentially in the last decade, with many domains deriving performance gains through larger and larger models. Natural language processing (NLP) in particular has seen an explosion in model size with the rise of the Transformer architecture, with state of the art models like OpenAI's 2019 GPT-2 making use of 1.5 billion parameters only to be surpassed in 2020 by GPT-3's 175 billion parameters [1][2]. Given the enormous computational requirements to train these models from scratch, most applications start with the model weights obtained from the original pretraining. The static nature of these language models encodes world knowledge into their parameters that quickly becomes stale; world leaders change, companies grow and shrink, but the models continue to reflect the world captured by their pretraining datasets. Therefore for these models to remain relevant and useful through time, it is fundamental that specific facts and knowledge can be updated without degrading the model's performance overall.

Most NLP applications make use of fine-tuning, additional training on a dataset of interest to encourage the model to produce similar output. For knowledge editing, fine-tuning is too broad a brush, altering language features ranging from syntax to tone without guarantees that specific facts or examples can be amended. To date, other approaches to correcting knowledge errors involve augmenting the training data or manually overriding machine learning systems, but neither allows for systematic and inexpensive model corrections on the fly [3]. Instead, efforts in Model-Agnostic Meta-Learning (MAML) have shown promising results by pushing model parameters towards a secondary objective using gradient-based techniques. Selective editing to correct classification errors recently received attention from Sinitsin et al. [4], with the authors successfully updating predictions on image classification examples without affecting overall performance.

In this paper, we adapt this MAML selective editing technique to the task of language generation using pretrained Transformer models. Our contributions include developing a novel dataset with altered named entities, incorporating these edited samples into a MAML training architecture built from scratch to encourage parameter adaptability, and creating relevant evaluation metrics and baselines

for the task. We focus specifically on training a language model to efficiently update its probabilities of producing a specific named entity given a single edited example, but the technique shows promise for a much richer set of language generation tasks.

2 Related Work

The idea of altering neural networks by introducing meta-learning goals has received attention since Finn, Abbeel, and Levine’s 2017 [5] proposal for a model-agnostic algorithm that trains a given model on a multiple tasks. They demonstrated that their MAML procedure produced models that achieved high performance on these alternate learning tasks after exposure to only a small number of training samples. By focusing on shifting parameter distributions during the training procedure, the algorithm improved generalization and allowed efficient finetuning using a limited number of gradient steps. Finn et al. [6] later showed this idea could be pushed further in an online setting, creating flexibility when only sequential updates are available.

MAML techniques were recently deployed for error correction in Editable Neural Networks [4], in which Sinitsin et al. propose a training approach that can reliably alter mistaken network output and minimize overall disturbance and computational requirements relative to other approaches. With their technique termed "Editable Training," the authors define an editor function, $\hat{\theta} = \text{Edit}(\theta, l_e)$, where $\hat{\theta}$ represents the edited parameters and l_e a task-specific constraint provided to the training procedure such that $l_e(\hat{\theta}) \leq 0$. While they focus on applications in computer vision, they provide a short neural machine translation example that shows the promise of MAML editing in the NLP domain.

Finally, the issue of static knowledge encoded in language models has recently received more attention, with Lazaridou et al. [7] examining the poor performance of pretrained Transformer models applied to out of sample domains. They find that static language models are unable to adapt and generalize about the future, a feature that is inflexible even in increasing model sizes. Our training procedure seeks to meet the authors’ call for a review of how we evaluate model performance and a focus on exploring more flexible architectures.

3 Approach

Our approach incorporates several distinct steps to build and evaluate what we term the "language editing" procedure. First, we construct a novel dataset consisting of an unedited passage, a sentence of the passage with a permuted named entity, and a record of the named entity changed. Second, a training architecture is developed using the MAML library Higher [8] allowing for optimization over both a typical language model cross-entropy loss along with an "adaptability" loss. Finally, performance measures are created to understand how the language editing procedure altered model parameters and highlight areas for future improvement.

Data The dataset for this setting is a large corpus of text with identifiable named entities, divided into training, validation, and test sets. An off-the-shelf named entity recognition model identifies entity types along with start and end locations and appends each entity to a corpus-wide cache. Then to produce the edited dataset, a passage containing an entity is selected and a single sentence is extracted. Its named entity is then swapped with a different one selected at random from the cache, producing data triplets for each sample consisting of the original passage, the edited sentence, and the entity edits.

Model To ensure the task explored mimics a real-world application, we use a pretrained transformer model available through HuggingFace [9], DistilGPT2, as our base language model. Given large memory requirements to run MAML, DistilGPT2 is attractive for its reduced parameter count compared to the smallest standard GPT-2 model (82M vs. 110M parameters). Our methodology is model-agnostic so similar models could easily be substituted.

Training Architecture The language editing procedure prepares the model to be responsive to one-off edits made during evaluation. Let $f(x, \theta)$ be the functional form of the language model with text passages x , parameters θ , and task-specific loss function $L_{base}(x, \theta)$, here a standard language model cross-entropy loss between model predictions and ground truth text. For text passages x sampled

from the dataset, let x_e represent the edited passages with swapped named entities. The model passes through two optimization procedures, each specified for a different objective. An "outer-loop" optimization trains model parameters θ on a combination objective, while an "inner-loop" subroutine focuses on adaptability training.

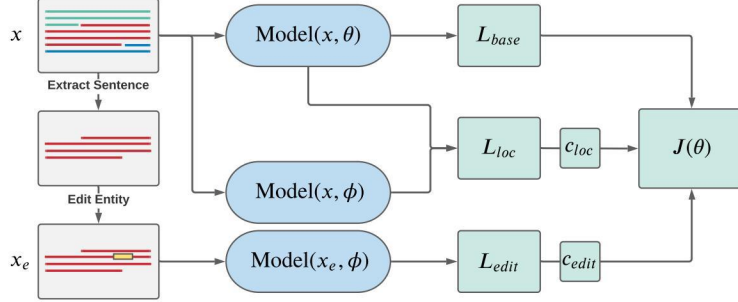


Figure 1: An overview of the language editing training procedure. The data are prepared for training (gray), passed through the differently parametrized models (blue), and outputs are used to evaluate losses (green).

The inner-loop optimization, what we could term the "edit step," pushes the model to learn altered parameters via the edit function, $\phi = \text{Edit}(x_e, \theta)$. This step clones the model parameters, θ , and sequentially optimizes parameters ϕ over an edit loss $L_{\text{edit}}(x_e, \phi)$, defined as the cross-entropy taken solely over the altered named entity tokens in the edited dataset.

The outer-loop optimization trains the model on the overall objective combining the base language model, language editing, and distribution shifts between the edited and base models. First, $L_{\text{base}}(x, \theta)$ is calculated over the unedited model, $f(x, \theta)$, identical to a typical fine-tuning loss. Second, the language editing loss $L_{\text{edit}}(x_e, \phi)$ is calculated after all edit steps in the inner-loop are completed and represents the final loss on the edited sample. Third, a locality loss $L_{\text{loc}}(x, \theta, \phi)$ penalizes distribution shifts in model output over unaltered data samples, measured by the Kullback-Leibler (KL) divergence between edited model predictions $p(y | x, \phi) = \text{softmax}[f(x, \phi)]$ and unedited model predictions $p(y | x, \theta) = \text{softmax}[f(x, \theta)]$. Finally, this is combined in a single objective $J(\theta) = L_{\text{base}} + c_{\text{edit}} \cdot L_{\text{edit}} + c_{\text{loc}} \cdot L_{\text{loc}}$ with hyperparameters c_{edit} , c_{loc} , and this loss is backpropogated through the entire network. Thus while only the parameters ϕ are updated in the inner-loop, they remain attached to the gradient-tape of the original model and eventually update θ through the L_{edit} and L_{loc} losses. This procedure is summarized by Figure 1 with losses specified below. For vocabulary size $|V|$, vocabulary tokens $y_i, i \in [1, |V|]$, edited entity tokens y_{e_i} , and length of input sequence J :

$$L_{\text{base}}(x, \theta) = \sum_i -\log p(y_i | x, \theta) \quad (1)$$

$$L_{\text{edit}}(x_e, \phi) = \sum_{e_i} -\log p(y_{e_i} | x_e, \phi) \quad (2)$$

$$L_{\text{loc}}(x, \theta, \phi) = \frac{1}{J} \sum_{j=1}^J D_{\text{KL}}(p(y | x, \theta) \| p(y | x, \phi)) \quad (3)$$

$$J(\theta) = L_{\text{base}} + c_{\text{edit}} \cdot L_{\text{edit}} + c_{\text{loc}} \cdot L_{\text{loc}} \quad (4)$$

At evaluation time, we test the model's ability to make a desired knowledge edit efficiently. A sentence with a permuted named entity is selected from the validation set and the edit step is run alone for a fixed number of iterations. This allows for a direct comparison of the model's probability of producing the new entity tokens before and after the edit steps are run.

Baselines As a first baseline, this evaluation procedure is performed on the pretrained DistilGPT2 without fine-tuning. As a more realistic baseline, the same procedure is applied to a DistilGPT2 model that is fine-tuned on 10,000 unaltered training samples from the WikiText-103 dataset, hereafter referred to as "DistilGPT2-FT."

4 Experiments

4.1 Data

Training, validation, and test datasets are built from WikiText-103 [10], a processed corpus of over 100 million tokens drawn from Wikipedia, and we select a random training sample of 100,000 passages to ensure a diverse vocabulary and topic space. SpaCy’s "en_core_web_md" model [11] is used to extract named entities labeled as "PERSON" and break passages into sentences. For passages containing at least one "PERSON" reference, a different named entity is chosen at random and swapped with the original in the passage. The unaltered passages define the base training set, used to finetune the language model on WikiText, while the corresponding edited passages train the adaptability of parameters.¹

Dataset	Sample
Original	During his time as a hotelier , Hinton made several overseas trips , the first of which was to attend the coronation of Queen Elizabeth II .
Edited	During his time as a hotelier , Charles Coghlan made several overseas trips , the first of which was to attend the coronation of Queen Elizabeth II .

Table 1: Example training sample. The "Original" sentence is unaltered from the WikiText-103 dataset and is used in the base language model loss, L_{base} . The "Edited" sentence differs only in the named entity present and is used to train the MAML adaptability objective, L_{edit} .

4.2 Evaluation method

The outcome of the training procedure is evaluated along two dimensions: editing success and degradation in model output on unrelated samples.

Editing Success At evaluation, the ultimate goal is to encourage the model to produce the edited named entity given its context in a validation sample sentence. Quantitatively, measuring this success could take many forms; most simply, let Success_{base} be the proportion of validation samples for which the probability (or negative log-likelihood) of producing the correct edited named entity increased. Given that this is a low bar, let $\text{Success}_{5\%}$ and $\text{Success}_{10\%}$ represent the proportion of samples for which the negative log-likelihood (logits) increased by 5 or 10 percent respectively after the edit steps. Formally, for sample j , named entity ent_j with tokens y_i , pre-edit parameters θ and post-edit parameters ϕ , we define

$$\text{Success}_{base} = 100 \times \frac{1}{J} \sum_{j=1}^J \mathbb{I} \left\{ \sum_{y_i \in ent_j} -\log p(y_i | x_j, \phi) > \sum_{y_i \in ent_j} -\log p(y_i | x_j, \theta) \right\}$$

$$\text{Success}_{5\%} = 100 \times \frac{1}{J} \sum_{j=1}^J \mathbb{I} \left\{ \sum_{y_i \in ent_j} -\log p(y_i | x_j, \phi) > 0.95 \sum_{y_i \in ent_j} -\log p(y_i | x_j, \theta) \right\}$$

with $\text{Success}_{10\%}$ defined similarly.

Perplexity Drawdown Sinitsin et al. [4] evaluate model degradation through the concept of drawdown, a measure of error on unaltered samples before and after performing an edit on the model. A lower drawdown indicates that model locality has been preserved despite the editing procedure. We introduce perplexity drawdown, the percent increase in perplexity (PPL) over sequences generated by the model during evaluation. Over J validation samples, pre-edit parameters θ and post-edit parameters ϕ

$$\text{PPL}_{dd} = 100 \times \frac{1}{J} \sum_{j=1}^J \frac{PPL(\phi) - PPL(\theta)}{PPL(\theta)}$$

¹Code for data processing, model training, and evaluation can be found at https://github.com/spencerbraun/editable_nlp

4.3 Experimental details

Given the large potential search space within this flexible training framework, we ran a number of experiments modifying loss calculations, hyperparameter values for c_{edit} , c_{loc} , and number of optimization steps. Early experiments took the editing loss L_{edit} over the entire altered sentence instead of the specific edited named entity tokens; this loss proved too noisy to make progress on the adaptability task, and focusing on the edited tokens increased learning in the inner-loop optimization significantly. All experiments below trained only the last 3 Transformer layers in the pretrained DistilGPT2 model on the edit task to reduce memory overhead. (All layers were trained for the outer-loop optimization over the combination objective $J(\theta)$.)

Several learning rates were tried, but all experiments used a higher learning rate for the inner-loop optimization than the outer-loop, which improved progress on L_{edit} without overall degradation. For experiments below, the inner optimization was conducted with stochastic gradient descent with learning rate 0.001 while the outer optimization was run with Adam and hyperparameters $lr = 10^{-5}$, $\text{betas} = (0.9, 0.999)$, $\text{eps} = 10^{-8}$. During training, all models took a single inner-loop optimization step per edited sample. The outer-loop optimization made use of gradient accumulation, resulting in 5 inner-loop optimizer steps for every 1 outer-loop step.

4.4 Results

Model Comparisons Reviewing model differences on the validation set informs our view of the conditions needed to allow language editing to work. Table 2 highlights some interesting findings over a subset of attempted training and evaluation procedures. The column "# Edit Steps" refers to the maximum number of edit steps made at evaluation time; if the edit loss on a given sample dropped below 1, the optimization loop was stopped early. While experiments were run with up to 5 edit steps, most models showed declining performance as edit steps increased. Included in the table are two baselines, an unmodified DistilGPT2 model (DistilGPT2 below) and the fine-tuned model trained on WikiText over 10,000 samples (DistilGPT2-FT).

A few notable statistics weighed heavily on model selection. For model Editable 1.0, we see that $c_{edit} = c_{loc} = 1$ which resulted in great performance on the editing success metric but excessive perplexity drawdown; future runs weighed L_{loc} more heavily to increase model locality. Note also that most models performed well as measured by Success_{base} while the baselines performed much worse on $\text{Success}_{5\%}$. As we will explore, the Success_{base} metric sets the success bar too low allowing the baseline models to clear this threshold without truly adopting the desired edit. Finally it is interesting to note that longer training times, eg. model Editable 2.2, resulted in worse performance across metrics.

Model	c_{edit}	c_{loc}	# Samples	# Edit Steps	PPL_{dd}	Success_{base}	$\text{Success}_{5\%}$
DistilGPT2	-	-	-	1	0.14	80	0
DistilGPT2	-	-	-	3	0.72	78	8
DistilGPT2-FT	-	-	10k	1	0.02	82	0
DistilGPT2-FT	-	-	10k	3	0.51	84	8
Editable 1.0	1	1	10k	1	44.78	86	72
Editable 2.0	1	10	10k	1	0.71	84	56
Editable 2.1	1	10	10k	3	9.01	82	58
Editable 2.2	1	10	18k	1	1.29	66	52
Editable 3.0	1	15	10k	1	-0.18	74	54

Table 2: Model comparison along training hyperparameters, validation edit steps, and evaluation metrics.

Final Results From the results above, the model Editable 2.0 was selected for final evaluation. The editing procedure was also run for DistilGPT2 and DistilGPT2-FT models using samples from the test set. Table 3 shows the final results, which unsurprisingly are similar to the validation results above.

Model	PPL_{dd}	Success _{base}	Success _{5%}	Success _{10%}
DistilGPT2	-0.02	71.0	0.0	0.0
DistilGPT2-FT	0.07	78.0	0.0	0.0
Editable	0.76	89.0	70.0	43.0

Table 3: Final Evaluation

In some sense, it seems logical that introducing a secondary training objective increases model flexibility, as one could view the training edit step as a form of regularization on the model’s fit to the WikiText dataset. Indeed while perplexity drawdown is minimal after the evaluation edit step, the language edited model produces output with an average raw perplexity score of 61.6, higher than DistilGPT2-FT’s score of 41.7 and lower than the DistilGPT2 model’s 74.6. Thus the model has not been allowed to fully fit to the WikiText dataset. This difference could potentially be remedied through additional re-weightings of hyperparameters c_{edit} , c_{loc} .

However, the degree of performance change induced by the language editing procedure is still surprising. Table 3 indicates that without the adaptability training objective, neither baseline model is able to adjust a single edited entity’s logits by more than 10%, while the selected Editable model does so on nearly half of the test samples. First, this result emphasizes the issues encountered when relying on static language models, since while we can easily decrease perplexity with a fine-tuning procedure, this additional training does little to encourage the model to adapt to content drift over time. Second, the language editing procedure was able to achieve good results without much additional computational overhead. The best performance comes from taking a single edit step for each data sample and a modest number of optimization steps. This result shows promise for broader adaptability goals, as modest training modifications can have significant impact on secondary model objectives.

5 Analysis

Model Outputs The experimental results demonstrate a large gap between models when measured by a metric like Success_{5%} but little difference when we consider Success_{base}. This outcome may leave some ambiguity in our judgement of the success of the procedure or in what circumstances the language editing procedure might be relevant. Looking more closely at the model outputs helps identify where these differences lie.

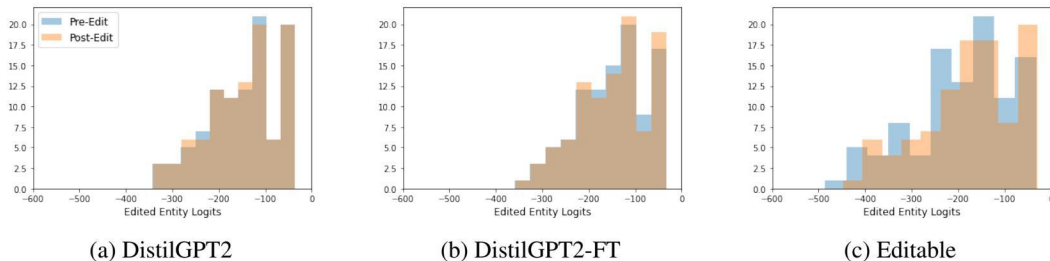


Figure 2: Distribution of the negative log-likelihoods (logits) of producing tokens associated with an altered named entity before and after an attempted edit.

Figure 2 displays histograms of the model output logits for the edited entity tokens before and after performing 100 different attempted edits drawn from the test set. Panel (a) plots the logits from the DistilGPT2 model, and it can be difficult to discern any difference before and after an edit is made. Thus while the logits for the new entity may be pushed slightly higher by the editing process, and therefore meeting the success criteria of Success_{base}, it is unlikely we would ever see any difference in the actual language produced by this model. If we perform a similar exercise for the baseline DistilGPT2-FT in Panel (b), we can see slightly more movement after an edit but still far from the dramatic difference we are hoping to produce.

Finally, Panel (c) shows the pre- and post-edit logits produced by the Editable model after the language editing training procedure. The difference is dramatic, as the distribution for the desired

edits has been shifted rightward significantly. The change is even more apparent in Figure 3, which presents the percent change in logit values for the edited tokens after the edit is made.

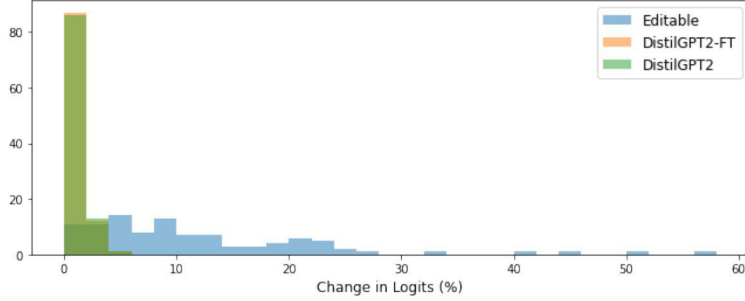


Figure 3: Percent change in sum of logit values for edited named entity tokens output before and after an attempted edit is made. Shown for 100 test set samples.

While the Editable model increases its probability of producing the new named entities by up to 60%, the baseline models are clustered near zero, allowing at most a single percent increase in edited logits. The language editing procedure has clearly impacted the model’s ability to adapt to proposed changes, even with a single edit step.

Model Parameters Beyond the outputs of each model, we might also consider how the edit affects model internals. In the experiments above, the last three transformer layers are optimized in the language editing procedure, and it is unclear from the outputs how localized an edit may be to a single layer and how consistently the MAML training affected adaptability of specific parameters.

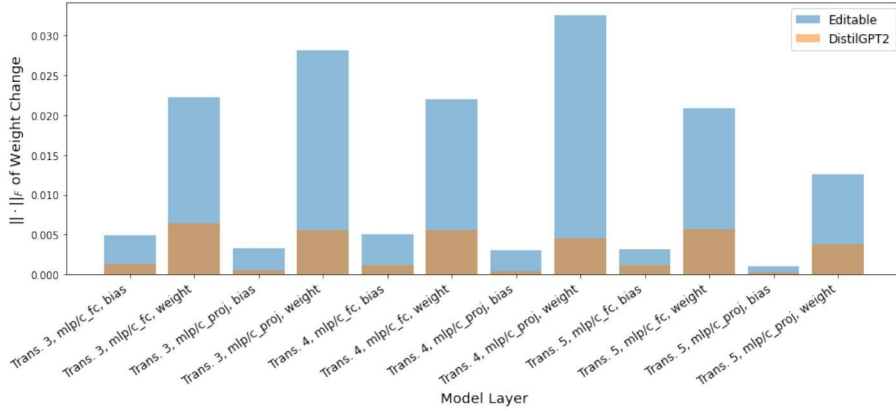


Figure 4: Frobenius norm of parameter changes from pre-edit to post-edit by Transformer feed-forward layer.

Figure 4 shows the Frobenius norm of the entry-wise difference for each Transformer parameter matrix before and after an attempted edit is made to a model. Specifically, it considers the weight and bias matrices for the feed-forward layers in Transformers 4-6 for both the Editable model and baseline DistilGPT2. The Editable model shows robust change across Transformer layers that far exceeds any change seen in the baseline. The language editing training seems to have impacted all layers included in the inner-loop optimization relatively consistently. In short, the experiment points to significant change in the model’s response to a proposed edit across internal and external measures.

6 Conclusion

We introduce a simple MAML training procedure to increase the adaptability of language models to selective edits and demonstrate its empirical efficacy against realistic baselines. We show that large

pretrained language models can adopt edits successfully without significant performance degradation on unrelated text samples. This project contributes to the larger goal of increasing the dynamism of NLP models, but it is only a first step in exploring knowledge editing in this setting.

While we considered the performance of single edits to a trained model, a more useful experiment would modify the procedure to evaluate multiple sequential edits to the same model. This would mimic a realistic application setting in which a model in production over a longer period of time must be corrected to reflect many changes in the real world. It is likely that additional steps would need to be taken to prevent dramatic degradation or locality loss after several edits.

This project also explored a limited section of potential training architectures, and future work may find new gains in some combination of simple changes. These alterations may include training all Transformer layers, creating distinct "adaptable" and "knowledge" sets of model parameters, experimenting with larger and richer language models, and incorporating a replay buffer to reduce distribution shift. Ultimately there are many promising avenues for exploration, and we hope this project serves only as a starting point for the creation of continuously editable language models.

References

- [1] A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [3] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.
- [4] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkun, Sergei Popov, and Artem Babenko. Editable neural networks. In *International Conference on Learning Representations*, 2020.
- [5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.
- [6] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1920–1930. PMLR, 09–15 Jun 2019.
- [7] Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d’Autume, Sebastian Ruder, Dani Yogatama, Kris Cao, Tomas Kocisky, Susannah Young, and Phil Blunsom. Pitfalls of static language modelling, 2021.
- [8] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.
- [9] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.
- [10] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2016.
- [11] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.