Reconsidering the Past: Optimizing Hidden States in Language Models

Davis Yoshida **Kevin Gimpel**
Toyota Technological Institute at Chicago, IL, USA, 60637
{dyoshida,kgimpel}@ttic.edu

Abstract

We present Hidden-State Optimization (HSO), a gradient-based method for improving the performance of transformer language models at inference time. Similar to dynamic evaluation (Krause et al., 2018), HSO computes the gradient of the log-probability the language model assigns to an evaluation text, but uses it to update the cached hidden states rather than the model parameters. We test HSO with pretrained Transformer-XL and GPT-2 language models, finding improvement on the WikiText-103 and PG-19 datasets in terms of perplexity, especially when evaluating a model outside of its training distribution. We also demonstrate downstream applicability by showing gains in the recently developed prompt-based few-shot evaluation setting, again with no extra parameters or training data.

1 Introduction

Finetuning a pretrained transformer language model (LM) (Vaswani et al., 2017; Radford et al., 2018; Peters et al., 2018; Devlin et al., 2019) is now the default method for attacking a task in modern NLP. Due to the high cost of pretraining, much research has been focused on how better to apply the pretrained models, rather than just improving pretraining itself. However, even finetuning can be too costly, especially for models such as the 175 billion parameter GPT-3 (Brown et al., 2020). As such, researchers have sought low cost alternatives, such as finetuning a small set of auxiliary parameters (Houlsby et al., 2019), or more recently leaving the LM weights fixed and passing a textual context designed to elicit the desired behavior via token prediction, such as in Brown et al. (2020).

One direction for language modeling in particular is to leave the LM parameters fixed, but update its intermediate quantities (e.g., Dathathri et al., 2020 and Qin et al., 2020). In this paper, we introduce Hidden-State Optimization (HSO), a method that contributes to this line of work. HSO first computes the language modeling loss as usual, then modifies the LM hidden states using the gradient of the loss (but critically reports the original loss). This process is repeated for each window of 10-25 tokens, updating the cached hidden states each time. Attending to these modified hidden states creates higher quality predictions for future tokens.

As an example of how future information can help embed past tokens, consider the garden path sentence: "The old man the boat." The embedding for "man" will only depend on "The", "old", and "man", so it will not reflect that "man" is being used as a verb. HSO can be seen as a method of incorporating future information into the representation of a context while still using a left-to-right LM. BERT (Devlin et al., 2019) showed that bidirectional information passing improves embedding quality, which suggests that doing so should improve performance on downstream tasks.

We demonstrate HSO in the setting of language model evaluation on the WikiText-103 (Merity et al., 2017) and PG-19 (Rae et al., 2020) corpora, and find improvements in measured perplexity. In order to demonstrate that this translates into value for downstream applications we apply HSO to few-shot classification with the 1.5B parameter GPT-2, and find improvement in that setting as well.

2 Related Work

Learning during inference. HSO is related to methods that perform learning on the test set. One such method is dynamic evaluation (DE) (Krause et al., 2018, 2019), which was the inspiration for HSO. DE consists of using test inputs for learning after evaluating on them, which means a larger test set will result in a larger gain from its use. This is not reflective of the small amount of text present in a setting such as conditional generation or few-shot classification, while using HSO for LM evaluation is. HSO is also cheaper than DE because

it differentiates with respect to hidden states rather than the model parameters. See Section 4.2.2 for more discussion and results on this point.

Gradient-Based Optimization of Hidden States. Qin et al. (2020) proposed Delorean, a method that incorporates future tokens into LM predictions by using backpropagation into earlier intermediate vectors. However, their goal is to produce better generations for intermediate timesteps, using sampled intermediate tokens and ground truth future tokens. We instead use the LM loss to tune past hidden states to allow better prediction of unseen future tokens. They also only perform gradient updates to logits while we update hidden states.

Plug-and-Play language models (PPLM; Dathathri et al., 2020) modify the behavior of pretrained LMs by updating hidden states at inference time, but with the goal of controllable generation (e.g., controlling sentiment) rather than improved fidelity. Unlike HSO, PPLMs require an attribute classifier which must be trained with labeled data. Several methods have been developed to more efficiently achieve the same goal as PPLM (Madotto et al., 2020; Krause et al., 2020), and these ideas could potentially be applied in analogous ways to speed up HSO.

Alternatives to finetuning. Our method is related to those that reduce the computational cost of finetuning by updating a smaller number of parameters or avoid finetuning altogether. Houlsby et al. (2019) introduce adapter modules which are finetuned in lieu of the full model. Li and Liang (2021) introduce prefix-tuning, which adds a fixed set of learnable vectors to the beginning of the input sequence. The latter is related to using prompts for contextual generation, which has gained popularity both to extract information from language models (e.g., Radford et al., 2019, Jiang et al., 2020) and perform tasks directly without updating any model parameters (Brown et al., 2020). Follow-up work has sought to understand the effectiveness of prompting (Le Scao and Rush, 2021) and automatically find or learn better prompts (Shin et al., 2020; Liu et al., 2021; Qin and Eisner, 2021).

3 Method

Let f be a transformer language model computing the distribution for token x_t given tokens $x_{1:t-1}$:

$$p_t = f(x_{1:t-1})$$

In practice, one may cache the hidden states, $\boldsymbol{h}_t \in \mathbb{R}^{\ell \times d}$, where ℓ is the number of layers and d is the embedding size. We represent this by factoring f into f_h which computes hidden states (possibly depending on past hidden states) and f_p which computes output probabilities from the hidden states:

$$\begin{aligned} \boldsymbol{h}_t &= f_h\left(x_t, \boldsymbol{h}_{1:t-1}\right) \\ p_t &= f_p\left(\boldsymbol{h}_t\right) \end{aligned} \tag{1}$$

Given a loss function L which takes as arguments the ground truth next word and a distribution over word types, one can then compute its gradient with respect to both the present hidden states $\boldsymbol{h}_t$, and with respect to the cached hidden states $\boldsymbol{h}_{1:t-1}$:

$$\begin{aligned} \boldsymbol{g}_{\text{present}} &= \nabla_{\boldsymbol{h}_t} L\left(x_{t+1}, f_p(\boldsymbol{h}_t)\right) \\ \boldsymbol{g}_{\text{cached}} &= \nabla_{\boldsymbol{h}_{1:t-1}} L\left(x_{t+1}, f_p(f_h(x_t, \boldsymbol{h}_{1:t-1})\right) \end{aligned}$$

Denoting the concatenation of these two quantities along the time axis as $\boldsymbol{g}_t = \left[\boldsymbol{g}_{\text{cached}}; \boldsymbol{g}_{\text{present}}\right]$, we can make a gradient update to the hidden states:

$$\tilde{\boldsymbol{h}}_{1:t} = \boldsymbol{h}_{1:t} - \eta \boldsymbol{g}_t \tag{2}$$

where η is the step size. We apply Adam (Kingma and Ba, 2015) to this update, but with modifications described in Section 3.1.

In practice, we use standard cross entropy as our loss function L. So, intuitively, we are updating the hidden states to make the actual word at position $t+1$ more likely under the language model's distribution p_t by altering only the previously computed hidden states. Note that when we update the hidden states with gradient-based updates, it will no longer be the case that the set of hidden states follow the feedforward procedure defined by the architecture of the transformer language model.

While computing the hidden state for x_{t+1}, we then substitute $\tilde{\boldsymbol{h}}_{1:t}$ into Eq. 1 in place of $\boldsymbol{h}_{1:t-1}$:

$$\boldsymbol{h}_{t+1} = f_h\left(x_{t+1}, \tilde{\boldsymbol{h}}_{1:t}\right)$$

Provided that the loss for timestep t is computed with the unmodified hidden state $\boldsymbol{h}_t$ rather than $\tilde{\boldsymbol{h}}_t$, this may be done at test time without the loss being improved by "looking into the future." We continue to update all hidden states at each step.[1]

In practice taking a gradient step after each token is too costly, so we can process blocks of k tokens (which we will refer to as a *window size* of k):

[1] $\tilde{\boldsymbol{h}}_{1:t}$ is then a concatenation of hidden states which have been updated between 1 and t times.

$$
\begin{aligned}
\boldsymbol{h}_{t+1} &= f_h\left(x_{t+1}, \tilde{\boldsymbol{h}}_{1:t}\right) \\
p_{t+1} &= f_p\left(\boldsymbol{h}_{t+1}\right) \\
\boldsymbol{h}_{t+2} &= f_h\left(x_{t+2}, [\boldsymbol{h}_{t+1:t+1}; \tilde{\boldsymbol{h}}_{1:t}]\right) \\
&\vdots \\
\boldsymbol{h}_{t+k} &= f_h\left(x_{t+k}, [\boldsymbol{h}_{t+1:t+k-1}; \tilde{\boldsymbol{h}}_{1:t}]\right) \\
p_{t+k} &= f_p(\boldsymbol{h}_{t+k})
\end{aligned}
$$

This sequence of computations is done in a single forward pass, but we have broken it up by token to make clear how a mix of unmodified and modified hidden states is used to embed each token in the window. Once the loss function, L, is applied to $x_{t+2:t+k+1}$ and $p_{t+1:t+k}$, a backwards pass is done to compute the gradient of the sum of the losses with respect to the hidden states, at which point the modified hidden states $\tilde{\boldsymbol{h}}_{1:t+k}$ are computed.

k has a twofold effect on computational cost, as it controls both the number of gradient steps and the number of tokens processed at a time. A very small k will require many more forward passes and will not take advantage of GPU parallelism.

3.1 Modifications to Adam

One way of applying Adam to the HSO gradient update would be to view the past hidden states as a single $T \times \ell \times d$ tensor, where T is the maximum context size. This would allow use of just two moment estimate tensors $\boldsymbol{m}, \boldsymbol{v} \in \mathbb{R}^{T \times \ell \times d}$. This version of Adam performs very poorly, as a given value in the hidden state cache will not be consistently associated with the same moment estimate.

Instead, we keep first and second moment estimates $\boldsymbol{m}_i$ and $\boldsymbol{v}_i$ for each hidden state, discarding them once the corresponding hidden states are further in the past than the maximum attention length. This also requires maintaining a different optimizer step value for each block of k hidden states, as Adam's bias correction depends on how many updates have been made to a moment estimate. In terms of implementation, we do not actually keep a separate vector for each hidden state, but pack them into a tensor which is translated along with the cached hidden state tensor.

4 Experiments

We demonstrate HSO with the Transformer-XL (TXL) (Dai et al., 2019) and GPT-2[2] (Radford et al., 2019) models implemented using FLAX (Heek et al., 2020) and Haiku (Hennigan et al., 2020), on top of JAX (Bradbury et al., 2018). The TXL model is initialized from the HuggingFace Transformers (Wolf et al., 2020) model trained on WikiText-103 (WT-103). The GPT-2 models are initialized from the OpenAI checkpoints.

Method	WT-103	PG-19
Baseline	21.3/22.4	166.4/164.2
HSO	**20.7/21.7**	**140.0/145.7**

Table 1: Language modeling validation/test perplexity with Transformer-XL (pretrained on WT-103). Importantly, PG-19 is out of distribution for this model.

Method	WT-103	PG-19
Baseline	21.5/20.7	26.7/**26.5**
HSO	**21.0/20.3**	**25.1**/26.5

Table 2: Language modeling validation/test perplexity with GPT-2 (345M parameters).

4.1 Language modeling

We test HSO with the TXL and 345M parameter GPT-2 models on the pre-tokenized WikiText-103 (Merity et al., 2017) and PG-19 (Rae et al., 2020) datasets. As the TXL was trained on WT-103, this covers both an in-distribution and out-of-distribution (OOD) evaluation for it. We found that TXL was not stable in the OOD setting, but that resetting its hidden states to zeros upon reaching its maximum context size reduced the baseline perplexity significantly. We do not do this for HSO as it does not appear to need this stabilization. We evaluate GPT-2 with non-overlapping contexts for efficiency. The perplexities reported are per token, which differs between GPT-2 and the word based TXL. Out of vocabulary words are UNK-ed for TXL, but GPT-2 has an open vocabulary.

We used a window size of $k = 25$, a learning rate of 0.003, and 0.65/0.9 for Adam's β_1 and β_2 parameters. We found that some HSO hyperparameter settings gave better performance, especially for GPT-2, but for the sake of parsimony report our main results with consistent hyperparameters.

Our LM results are shown in Tables 1 and 2. HSO yields about a half a point improvement in perplexity on WT-103 with both architectures. While this is not a large improvement, recall that GPT-2's hidden states are reset every 1024 tokens, so

[2]For GPT-2, we backpropagate into the key and value vectors rather than the full embeddings at each layer for ease of implementation. They differ by only a linear transformation, so we do not expect this to be a critical difference.

Modifications	Perplexity
None	25.1
$\eta = 3 \times 10^{-4}, \beta_1 = 0.8$	23.8
present-only	23.6
$k = 10$	24.4
$k = 10$, present-only	22.1
SGD, $\eta = 0.01$,	24.7
SGD, $\eta = 0.01$, present-only	25.1

Table 3: GPT-2 (345M) perplexity on the PG-19 validation set. η is learning rate, k is window size, "present-only" means only the last k hidden states are updated.

this represents improvement in prediction within the context of one attention window, rather than cumulative training on the test set as in DE.

On PG-19, the perplexity improvements are larger for the most part: 1.6 points for GPT-2 on the validation set and over 10 points for TXL (but a <0.1 point increase for GPT-2 on the test set). As we used the same hyperparameters for all LM evaluations, HSO seems to be fairly robust to the choice of architecture and dataset.

4.1.1 Modifying HSO

Table 3 shows the effect of various modifications to HSO on GPT-2's perplexity on the PG-19 validation set. Tuning Adam's parameters decreases perplexity by another point. Surprisingly, only updating the most recent window's hidden states ("present-only") improves perplexity on PG-19 (initial experiments on WT-103 did not find this to be the case). This also requires significantly less computation. Since Adam tries to estimate moments over many steps this might seem to imply it is not necessary. To investigate this, we tested stochastic gradient descent (SGD) with several learning rates but it performed worse than Adam for both full and "present-only" updates.[3]

4.2 Few-shot classification

While HSO can give gains in perplexity, we would like to see whether it benefits other tasks as well. So, we consider few-shot learning from examples in the LM's context, as in GPT-3 (Brown et al., 2020). Lacking GPT-3 access, we demonstrate our method with the 1.5B parameter GPT-2-XL model.

Dataset	n	Method			
		Baseline	DE[4]	HSO	HSO-2
SST-2	2	53.9	52.2 (55.1)	59.5	**64.0**
	4	58.3	55.6 (58.8)	63.1	**66.5**
	6	57.9	56.2 (59.4)	68.0	**69.2**
	8	58.4	59.9 (61.8)	**70.2**	70.2
AGNews	2	53.1	32.2 (35.0)	52.6	**54.3**
	4	**77.8**	52.2 (55.2)	77.2	77.6
	6	64.8	—	65.8	**66.2**
	8	63.3	—	68.5	**69.3**

Table 4: Effect of updating hidden states on few-shot classification accuracy of GPT-2-XL on SST-2 and AGNews, where n is the number of examples per prompt. Neither hidden states or weights are updated for the baseline. HSO-2 is HSO with two gradient steps per window of text.

We use the binary SST-2 (Socher et al., 2013) and 4-way AGNews (Zhang et al., 2015) classification datasets. We follow choices made by Zhao et al. (2021), including their prompt formats, but we made several changes to their procedure to reduce computational requirements and variance. Most importantly, we resampled a class-balanced prompt for every test example (but kept the prompt fixed between the baseline and HSO) rather than using a fixed prompt.[5] We used a learning rate of 0.01 and a window size of 10 tokens. Our experiments used a 24GB NVIDIA Quadro RTX 6000 GPU.

We also test DE, as in contrast to the LM setting, the amount of fine-tuning data will be the same between DE and HSO. We found that the learning rate of 0.01 led to the model collapsing to constant predictions, so we use a learning rate of 10^{-4} instead. We update the model every 10 tokens as with HSO, and recompute the hidden states after each update since the weights which produced them are no longer the model weights.

There are a few options to pick between when deciding what it meant to apply DE to this setting. One could choose to make a single gradient step based on the entire prompt, update the weights every 10 tokens but not recompute the hidden states, or perform multiple updates on the whole prompt. We chose what we believed was the closest comparison between HSO and DE, but did not experiment

[3] On the first step, Adam updates in the L_∞ steepest descent direction so it differs from SGD even for only one step.

[4] Due to the much higher running time for using dynamic evaluation, these are partial results from running on a random subset of the test set. The accuracy in parentheses is a hypergeometric 95% upper confidence bound. Future versions of this paper will have the full results. Furthermore, we exclude $n = 6, 8$ for AGNews due to running out of GPU memory on those input sizes.

[5] Zhao et al. (2021) reported high variance based on prompt choice, so we made this choice in order to only need to run each evaluation once. The other two changes were to sample 1200 examples from the AGNews test set to expedite the evaluation, and to only use examples with $\leq$35 tokens in our prompts to reduce the required memory.

with these other variations.

4.2.1 Results

Table 4 shows our results. HSO with a single gradient step leads to consistent improvements in accuracy across prompt sizes, and larger improvement with more prompt examples. The exceptions are AGNews with 2 and 4 example prompts, for which there is a slight decrease in accuracy. DE has similar performance to the baseline on SST-2, and degrades significantly on AGNews.

A longer prompt means both more examples to learn from and more gradient steps, so to disentangle the effect, we also tried two gradient steps per window (last column). This yields further improvement in 7 out of 8 cases. Surprisingly, for the cases where one gradient step was harmful, a second gradient step increases accuracy rather than causing further degradation. Also, a second gradient step generally causes a larger increase in accuracy for shorter prompts (e.g., for SST-2, two steps with two examples beats one step with four examples).

4.2.2 Compute costs for HSO and DE

As we noted earlier, DE is not intended to be applied to a very small amount of text, so this is not an apples-to-apples comparison of methods, but can still help emphasize the differences between the two. In this setting, DE uses a much smaller amount of data (less than a single full GPT-2 window) to make updates to the entire transformer's weights. As such, it is not surprising it does not improve greatly over the baseline.

In terms of memory, the parameters and Adam moment estimates for DE of GPT-2-XL require more than 18GB in total. As the parameters are updated separately for each example, batching multiplies this overhead by the batch size, making DE infeasible for use on prompts coming from different distributions. HSO's extra overhead is the moment estimates for the hidden states, which cost ~1.2MB per token of input, for a total of ~1.3GB on a maximum size input. Furthermore, DE requires storing an additional copy of the model parameters, as they must be reset after each example. To avoid storing this extra copy on the GPU, we transferred it from RAM to GPU memory each time.

While the primary performance advantage over DE is reduced overhead and batching, we examine runtimes for each method in Table 5. We additionally benchmark the 345M parameter GPT-2 for a speed comparison without the extra parameter transfer to the GPU. It is important to note that taking a single step per example instead of once per k tokens would be much faster than either method, as both DE and HSO require $\lceil \frac{N}{k} \rceil$ backward passes for a length N input.

Method	n	GPT-2 parameters	
		345M	1558M
DE	2	1.1	11.7
	8	3.3	30.6
HSO	2	0.4	2.2
	8	1.0	6.6

Table 5: Seconds per example for few-shot evaluation using HSO and DE on SST-2. Because DE with GPT-2-XL requires copying the parameters from RAM to GPU memory every step, we also include speeds for GPT-2-medium which does not have that additional overhead.

5 Conclusion and Future Work

We presented a method that optimizes transformer language model hidden states, which improves LM perplexity and prompt-based few-shot classification, without additional parameters or data.

Future work will explore improving the cost of HSO by further investigation into updating only a subset of hidden weights, and approximation of the exact gradient update. Other directions we will explore are its application to conditional generation by improving the representation of the context, and its interaction with other methods for improving prompt-based few-shot classification.

Acknowledgements

Thank you to the reviewers for their time and feedback, which helped us to improve the paper.

References

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. 2020. Flax: A neural network library and ecosystem for JAX.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. 2020. Haiku: Sonnet for JAX.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the International Conference for Learning Representations (ICLR)*.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq R. Joty, Richard Socher, and Nazneen Fatema Rajani. 2020. GeDi: Generative discriminator guided sequence generation. *CoRR*, abs/2009.06367.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2018. Dynamic evaluation of neural sequence models. In *International Conference on Machine Learning*, pages 2766–2775. PMLR.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2019. Dynamic evaluation of transformer language models. *arXiv preprint arXiv:1904.08378*.

Teven Le Scao and Alexander Rush. 2021. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2627–2636, Online. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *CoRR*, abs/2101.06804.

Andrea Madotto, Etsuko Ishii, Zhaojiang Lin, Sumanth Dathathri, and Pascale Fung. 2020. Plug-and-play conversational models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2422–2433, Online. Association for Computational Linguistics.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Guanghui Qin and Jason Eisner. 2021. 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*, pages 5203–5212, Online. Association for Computational Linguistics.

Lianhui Qin, Vered Shwartz, Peter West, Chandra Bhagavatula, Jena D. Hwang, Ronan Le Bras, Antoine Bosselut, and Yejin Choi. 2020. Back to the future: Unsupervised backprop-based decoding for counterfactual and abductive commonsense reasoning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 794–805, Online. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.

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. 2020. 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. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 649–657.

Tony Z Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. *arXiv preprint arXiv:2102.09690*.