Few-shot Sequence Learning with Transformers

Anonymous Author(s)
Affiliation
Address
email

Abstract

Few-shot algorithms aim at learning new tasks provided only a handful of training examples. In this work we investigate few-shot learning in the setting where the data points are sequences (or sets) of tokens and propose an efficient learning algorithm based on Transformers. In the simplest setting, we append a token to an input sequence which represents the particular task to be undertaken, and show that the embedding of this token can be optimized on the fly given few labeled examples. Our approach does not require complicated changes to the model architecture such as adapter layers nor computing second order derivatives as is currently popular in the meta-learning and few-shot learning literature. We demonstrate our approach on a variety of tasks, and analyze the generalization properties of several model variants and baseline approaches. In particular, we show that compositional task descriptors can improve performance. Experiments show that our approach works at least as well as other methods, while being more computationally efficient.

1 Introduction

The problem of learning a classifier from a handful examples has received considerable attention in the vision domain under the name of *few-shot learning* (Fink, 2005; Fei-Fei et al., 2006). In this work, we study the problem of sequence classification and modeling in the few-shot regime. Specifically, we assume there are several training tasks available for learning and, at test time, we are interested in performing few-shot adaptation to a given new task.

Transformers (Vaswani et al., 2017) have been very successful at modeling discrete sequences (Barrault et al., 2019; Devlin et al., 2018; Parisotto et al., 2019). Furthermore, they have been shown to use context tokens appended to an input to adapt their generations or to switch between different tasks (Lample et al., 2019; Zellers et al., 2019; Keskar et al., 2019). Thus, one might hope that such context tokens could be effectively used in the meta-learning setting for discrete sequences.

In this work, we show that this is indeed the case. Our approach to few-shot learning introduces a set of task specific parameters (a *task embedding*), in addition to the parameters of the model that are shared among all tasks. Unlike other approaches that require architectural changes (Houlsby et al., 2019), task embeddings are simply fed as input to the transformer. Learning a new task consists of inferring an appropriate task embedding for the task, leaving the shared model parameters intact. Towards this end, we propose a simple training algorithm where the task embedding is found via gradient based optimization, which is simpler and computationally less expensive than second order optimization methods (Finn et al., 2017; Zintgraf et al., 2019).

To summarize, our contributions in this work are as follows. First, we show that a simple alternating-minimization approach for few-shot learning works well in combination with the transformer architecture. Second, we show that a simple yet effective way to condition the transformer with task information is via input conditioning (i.e., feeding task information as input to the transformer); this naturally extends to compositional task information. Third, we introduce a battery of synthetic

Submitted to the 4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada. Do not distribute.

sequence classification and modeling tasks to benchmark in a controlled setting various baseline approaches and model variants for few shot learning of discrete sequences. And finally, we demonstrate that the proposed approach offers a better trade-off between few-shot performance and run time cost compared to other baselines, including meta-learning approaches.

2 Related work

Few-shot learning and meta-learning There is now a vast literature on learning methods designed for quickly adapting to new settings. At a coarse level, one can consider classes of methods that adapt the learning *algorithm* based on the task (and so are "meta-learners") (Schmidhuber, 1987; Hochreiter et al., 2001; Andrychowicz et al., 2016; Finn et al., 2017; Nichol & Schulman, 2018), or describe *model architectures* that can adapt to learn sample-efficiently over a task distribution (Vinyals et al., 2016; Snell et al., 2017). Many methods have elements of both of these, e.g. Mishra et al. (2018); Rusu et al. (2019); Zintgraf et al. (2019).

The method we describe in this work can be considered squarely in the class of model architectures for sample efficient learning. It is a descendant of Hinton & Plaut (1987); Schmidhuber (1992); Ba et al. (2016) and is closely related to Rusu et al. (2019); Zintgraf et al. (2019) in that we pick a subset of the weights of the model that are task specific (the "fast" weights), and update them using the training examples for a specific task; but update the other ("slow") weights on all training examples for all tasks. Our approach is closest to Zintgraf et al. (2019), but differs in the way the fast weights are used by the model, and because we do not use higher-order gradients for the slow weights, instead we use an alternating minimization type update.

Task transfer for transformers Our approach is also related to other recent work in natural language processing. We leverage the particular structure of the transformer architecture (Vaswani et al., 2017), which has been successful in many NLP tasks. Several works have shown that adding a token to an input can be used to switch between different tasks (Lample et al., 2019; Shen et al., 2019; Zellers et al., 2019; Keskar et al., 2019). Transformer language models trained on large corpora have also been recently shown to have impressive few-shot learning capabilities (Brown et al., 2020).

More generally, with the success of methods based on pretraining transformer models (Devlin et al., 2018), and finetuning on target tasks, there have been several works discussing how to adapt a pre-trained model without full finetuning (Houlsby et al., 2019; Stickland & Murray, 2019) but their focus has been on reducing the number of parameters subject to optimization at finetuning time as opposed to reducing the number of examples as in this study.

3 Approach

3.1 Architecture

In this work, we explore an adaptation of transformers to the few-shot regime. Previous works have shown that the behavior of a transformer model can be conditioned by appending special tokens describing the task to be performed on the input sequence (Lample et al., 2019; Zellers et al., 2019). We append a *task embedding* vector which represents information about the task of interest to the input sequence of token embeddings. We intend to control the overall behavior of the model for a particular task by altering the task embeddings while keeping the rest of the model parameters intact.

For classification tasks, we use a transformer encoder similar to the BERT model (Devlin et al., 2018). A classification head sits on the final layer representation of a special token at the beginning of the sequence. We replace this special token with the task embedding vector z in our model. We use a transformer decoder architecture for the transduction tasks and append the task embedding vector to the input sequence similar to the classification setting. See Fig. 1 in the appendix for an illustration.

In both settings we compute a log-likelihood of the form $\log p(y|x, z; \theta)$, where x is the input sequence, z is the task embedding and θ the model parameters. In the classification setting y is a categorical value. In the sequence transduction setting, y is a sequence and the log likelihood decomposes as the sum of the conditional log-likelihood terms $\log p(y|x, z; \theta) = \sum_i \log p(y_i|y_{i-1}, \cdots, y_1, x, z; \theta)$. Note that in practical applications θ can be high-dimensional, in the order of hundreds of millions. Our goal is to alleviate overfitting in the few-shot regime by adapting only z to learn a new task, where z is a small vector with at most a few hundred components. Next, we describe how we learn the model parameters θ and how we estimate the task embedding z for a given task.

Algorithm 1: TAM for k-shot Learning

Input : Training tasks $\mathcal{T}_1^{\text{train}}, \ldots, \mathcal{T}_N^{\text{train}}$
Output : Model parameters θ

repeat
  Sample a training task: $\mathcal{T}_i^{\text{train}}$;
  Sample $N_i \geq k$ training examples from the task $\{(x^j, y^j)_{j=1,\cdots,N_i}\} \sim \mathcal{T}_i^{\text{train}}$;
  Initialize: $z_{\mathcal{T}_i^{\text{train}}} = 0, \Delta\theta = 0$;
  while *loss improves and max number of updates not reached* **do**
    $z_{\mathcal{T}_i^{\text{train}}} \leftarrow z_{\mathcal{T}_i^{\text{train}}} - \nabla_{z_{\mathcal{T}_i^{\text{train}}}} \sum_j -\log p(y^j | x^j, z_{\mathcal{T}_i^{\text{train}}}; \theta)$;
    $\Delta\theta \leftarrow \Delta\theta - \nabla_\theta \sum_{j=1}^{N_i} -\log p(y^j | x^j, z_{\mathcal{T}_i^{\text{train}}}; \theta)$
  $\theta \leftarrow \theta + \Delta\theta$
until *max training iterations*;

3.2 Training and Inference Algorithm

We train our models with an alternating-minimization scheme similar to Maurer et al. (2013) and Kumar & Daume III (2012), that can be considered a simplification of the CAVIA approach in Zintgraf et al. (2019). See Algorithm 1 for pseudo-code. We separate the weights of the network defining the model into shared weights θ, and per-task weights, as in CAVIA. In our case, the per-task weights form the embedding z, one for each task; while all other parameters θ are shared.

We assume a distribution $p_{\text{data}}(\mathcal{T})$ over tasks from which disjoint sets of training, validation and test sets of tasks are drawn. The set of training tasks is denoted by $\{\mathcal{T}_i^{\text{train}}\}_{i=1}^N$, where each task $\mathcal{T}_i^{\text{train}}$ has an associated set of training examples $\{(x_j^i, y_j^i)_{j=1}^{N_i}\}$. Validation and test tasks are defined similarly, except each test task only has k training examples.

Given a few examples from the training task $\mathcal{T}_i^{\text{train}}$, we alternate training $z_{\mathcal{T}_i^{\text{train}}}$ (task embedding of task $\mathcal{T}_i^{\text{train}}$) for a few gradient descent steps keeping θ fixed, and then update θ based on the optimal task embedding. In practice, however, we found it helpful to update θ based on gradients accumulated for the intermediate values of the task embedding encountered in the inner loop optimization. We surmise that this optimization choice helps the model find better task embeddings as θ is updated to account for this search. Task embedding gradient updates are performed until the loss no longer improves or the maximum number of update steps has been reached. Note that unlike prior methods such as MAML or CAVIA we do not backpropagate gradients through an optimization process, which simplifies and speeds up our optimization. We call our method, Transformer trained with Alternating Minimization (TAM). At test time, given a new task $\mathcal{T}^{\text{test}}$, $z_{\mathcal{T}^{\text{test}}}$ is trained with a few steps of gradient descent, with all other parameters held fixed. Since TAM is trained to optimize task embeddings on the fly, we expect it to find good embeddings of the new task at test time as well.

4 Experiments

Model and Training Details In the classification setting, TAM is a bidirectional transformer that takes as input the input sequence x and the task embedding z, and outputs a distribution over classes. In the sequence transduction setting, TAM is a transformer decoder with a causal attention mechanism and takes as additional input the output sequence y up to the token before the last. In this case the model is trained to predict the sequence y at the last $|y|$ (length of sequence y) time steps. Since model parameters are shared across tasks, TAM needs to leverage the task embedding to perform the tasks well. Both classification and transduction models are trained with cross-entropy loss.

4.1 Baselines

We compare our approach against several baselines. **Task-Agnostic transformer** uses the same architecture as TAM but is not informed about the existence of different tasks at training time, i.e., no task embedding is fed at the input. The **Multitask transformer** is identical to TAM except all parameters including task embeddings are trained by standard back-propagation, without any alternating minimization. At test time, these baselines are fine-tuned on the k training examples from the test task. In addition, we consider Matching Networks (Vinyals et al., 2016), SNAIL (Mishra et al., 2018), MAML (Finn et al., 2017), CAVIA (Zintgraf et al., 2019) as baselines. More details about these baselines can be found in appendix C.

Model	Sequence Classification				Sequence Transduction				Path Finding			
	1	5	10	20	1	5	10	20	1	5	10	20
Task Agnostic	40.50	64.75	74.25	82.50	6.27	5.26	4.71	4.01	3.17	1.75	1.55	1.39
Multitask	38.75	66.00	77.50	87.50	13.80	6.80	5.18	2.91	6.39	1.98	1.64	1.44
Matching	**43.00**	58.75	64.50	67.00	–				–			
SNAIL	**43.00**	44.00	68.25	67.50	**2.48**	3.80	4.98	4.11	2.63	1.95	3.47	3.04
MAML	39.60	63.40	71.80	78.80	6.73	5.84	5.19	4.20	5.49	2.05	1.65	1.44
CAVIA	**43.00**	**78.00**	87.00	91.00	11.05	**2.75**	1.78	1.53	2.21	1.31	1.25	1.21
TAM	40.50	75.50	**89.50**	**94.50**	8.47	2.92	**1.47**	**1.15**	**1.82**	**1.27**	**1.22**	**1.17**

Table 1: k-shot sequence classification and sequence transduction experiments on our three benchmarks for $k \in \{1, 5, 10, 20\}$. The metric for sequence classification is average accuracy on test tasks (higher is better). On the transduction tasks, the performance metric is average perplexity on test tasks (lower is better). Random performance is at 25% accuracy (classification) and 12 perplexity points (other two tasks).

4.2 Sequence Classification and Transduction

In this work we construct a new set of benchmarks involving synthetic sequential data, allowing us to evaluate models in a more controlled setting. We construct a synthetic few-shot *classification* benchmark as follows. The benchmark consists of tasks that involve a non-negative integer sequence as input and a discrete label as output. A task is constructed by applying a sequence of mathematical transformations to input sequences as follows: Element-wise transform (T_1) $\rightarrow$ Subsequence extraction (T_2) $\rightarrow$ Labeling function (T_3). The arrows indicate function composition and the whole sequence of transformations maps an input sequence to a single integer. The transformations are defined as follows. $T_1 \in S_1, T_2 \in S_2, T_3 \in S_3$ where, $S_1 = \{$*mul v, add v, div v, mod v*$\}$; $S_2 = \{$*(not) multiple of v, (not) greater than v, (do not) have exactly v divisors*$\}$; $S_3 = \{$*count, min, max, mean, median, mode, first, last, max-min, middle*$\}$, where $v \in \{1 \cdots n\}$ for some integer n. We randomly generate a large number of sequences X of integers from $\{0 \cdots N\}$. We apply the transformation sequence T_1, T_2, T_3 to these sequences $x \in X$ and get the corresponding outputs $T_3(T_2(T_1(x)))$. The C most frequent outputs are then defined to be the classes of interest for a C-way classification task. An example task is `mul 2` $\rightarrow$ `less than 5` $\rightarrow$ `count`, where the goal is to count the number of input elements which, when multiplied by 2, are less than 5. The semantics of each of the transforms are defined in the appendix. We set $C = 4$ in our experiments. Vocabulary size and input sequence length are set to 12 and 5, respectively.

We also construct two sequence transduction benchmarks. The first benchmark is constructed in a way similar to the classification tasks where we consider a sequence of transformations mapping an input sequence to an output sequence. An example task is `add 2` $\rightarrow$ `replace 2 with 1` $\rightarrow$ `reverse`, and an (input, output) sample drawn from this task is: $([0, 5, 0, 3, 6], [8, 5, 1, 7, 1])$. Our second transduction benchmark is a path finding task in a grid world. A task is defined by start and end positions in a square grid of size $N \times N$. Given the locations of obstacles in this grid, the objective of the task is to find the shortest path connecting start and end positions that avoids the obstacles. The source and target sequences correspond to the locations of obstacles and optimal path from start to end position avoiding the obstacles, respectively.

We use 500, 16, 64 tasks respectively for training, validation and testing for all three setups. Tasks are unique and randomly assigned to these sets, in other words we test generalization under the condition of distributional match between the training and the test set. Each training task has 500 examples.

4.2.1 Results

Table 1 reports the results on this benchmark (See appendix for error bars). In the extreme few-shot setting ($k = 1$), all methods perform poorly, although memory based methods such as matching networks and SNAIL fare the best. However, they start performing worse when more labelled data is available, where fine-tuning part or all of the model parameters could be beneficial. Both SNAIL and matching networks sometimes perform *absolutely* worse when more labeled examples are present, suggesting they are failing to effectively use their memory when confronted with longer sequences. Fine-tuning the whole model, particularly in the multitask setting, works remarkably well for larger values of k, although the best performance is achieved by TAM, suggesting the need for sample efficient task adaptation methods. For $k > 1$, TAM performs comparably or better than all baselines, including MAML and CAVIA. Further, TAM is more efficient to train than CAVIA (see section 4.4).

Model	Sequence Classification 1	5	10	20	Sequence Transduction 1	5	10	20	Path Finding 1	5	10	20
Multitask	57.50	74.00	81.00	88.5	43.32	7.50	3.48	2.16	3.08	1.62	1.32	1.23
Matching	61.25	69.50	72.25	67.5	–				–			
SNAIL	**63.5**	71.75	76.25	80.25	7.00	6.24	6.93	17.10	**1.53**	1.71	3.36	4.22
CAVIA	57.25	66.25	67.50	68.50	36.72	6.01	3.99	3.27	1.99	1.27	1.21	1.17
TAM (comp)	63.00	**76.50**	**82.75**	88.5	**6.15**	**3.43**	**2.69**	**2.13**	2.33	1.30	1.23	1.19
TAM (non-comp)	45.25	72.5	81.5	**89.75**	7.80	5.08	3.60	2.42	4.37	**1.27**	**1.17**	**1.11**

Table 2: Compositional models for few-shot sequence classification and sequence transduction. All models (except non-compositional TAM) get information on the primitives present in the tasks via extra tokens appended to the input sequence, except that one such primitive is unseen at test time. Non-compositional TAM is not given information about primitives, and estimates a single task embedding instead.

4.3 Compositional Task Representations

Compositional reasoning is arguably an important skill for few-shot learning (Lake, 2019; Purushwalkam et al., 2019). The underlying assumption is that there exist primitive skills which can be learned and combined together to solve entirely new tasks. If a learner can leverage the compositional structure of the learning task, it may learn with even less labeled data.

In this section we assess how much better TAM works when we expose the compositional structure of the tasks described in §4.2. Specifically, we assess the ability to learn new tasks which are composed of primitives, some of which were unseen during training. To present an example from the classification setting, assume the models know that tasks are composed of three transforms $T_1 \in S_1, T_2 \in S_2, T_3 \in S_3$. We henceforth refer to the elements of $S_1 \cup S_2 \cup S_3$ as *primitives*. Further assume the model never saw the *add* k primitive during training. Given a new test task for which $T_1 =$ *add* 3 (and T_2, T_3 are known primitives seen during training), we expect the model to infer the concept of *add* from few training examples.

4.3.1 Task Construction

In the compositional setting, we provide models with information about the primitives used to construct the task. For the classification and transduction tasks, the training and test tasks are constructed as follows. Assume the set of primitives available for the three transforms to be S_1, S_2, S_3. We hold out a subset of primitives S'_1, S'_2, S'_3 respectively from each of these three sets, which shall constitute the *unseen* primitives. The training tasks are made up of primitives from $S_1 - S'_1, S_2 - S'_2, S_3 - S'_3$, which we will refer to as *seen* primitives. The test tasks are made up of seen and unseen primitives where exactly one primitive is unseen (For instance, $T_1 \in S_1 - S'_1, T_2 \in S'_2, T_3 \in S_3 - S'_3$). Model performance is averaged over multiple (8) different choices of S'_1, S'_2, S'_3.

We also define a compositional path-finding task as follows. In addition to finding the optimal path from start, end positions while avoiding obstacles, we now require the path to lie on a specified way-point (See F.2 for an example). The locations of the start, end and way points thus define the primitives that make up a task. Similar to the previous settings, we hold out sets of values for each of these points and construct the train/test tasks in an analogous manner.

4.3.2 Training

For all the models, a sequence of primitive ids representing the primitives that make up the task is appended to the input sequence. These primitive embeddings θ_e are learned along with the other model parameters. To simulate the testing conditions, at training time we pretend some primitives are unknown. For the multitask, matching network and SNAIL baselines we learn an *unknown primitive embedding*, which is used to initialize embeddings of unknown primitives encountered at test time. Although the tasks themselves are harder (because entire primitives are unseen), modeling them is easier because the primitive information is given to the model. For CAVIA and TAM, we infer embeddings for unknown primitives on the fly using gradient descent during train and test. See Algorithm 2 in the appendix for the complete algorithm. We use 5000 training tasks, and 100 validation and test tasks each. Each training task has 500 examples.

Arch	Training	Sequence Classification 1	5	10	20
LSTM	Multi	0.38	0.57	0.72	0.87
	Alt	0.35	**0.78**	0.83	0.85
Transf	Multi	0.39	0.68	0.80	0.88
	Alt	**0.41**	0.76	**0.89**	**0.94**

Table 3: k-shot accuracy when using different architectures. We consider two architectures and two training methods, multitask learning and the proposed alternating minimization algorithm.

Model	Classification Acc., Time	Transduction Ppl., Time	Path-finding Ppl., Time
Multitask	67.4, **0.5h**	7.2, **0.5h**	2.9, **0.3h**
CAVIA	74.8, 3h	4.5, 5.3h	1.5, 3.7h
TAM	**75.0**, 2h	**1.5**, 2.3h	**1.3**, 2.7h

Table 4: Training efficiency: Time taken by each training algorithm to reach the best model (identified using validation tasks) and corresponding model performance (non-compositional setting). Performance and time are averaged across $k \in \{1, 5, 10, 20\}$ shots.

4.3.3 Results

Table 2 summarizes the results in the compositional setting. We observe similar trends as before for the non-compositional case. Multitask learning becomes competitive only for larger values of k. Vice versa, matching networks and SNAIL suffer with long sequences (larger values of k). TAM performs comparably if not better than methods relying on second order derivatives like CAVIA. Finally, the compositional version of TAM often yields higher accuracy than the corresponding non-compositional version, showing that the model is able to leverage the additional knowledge about a subset of primitives (two out of three) that compose the new task. Compositionality is particularly helpful with fewer shots (e.g., 1-shot) – with sufficient training examples (e.g., 20-shot) models benefit less from compositionality.

4.4 Discussion

Importance of Transformer Architecture The experiments presented in this paper so far have used a transformer. We study the impact of swapping out the transformer with a recurrent model in Table 3 on non-compositional tasks. We use a bidirectional LSTM with a comparable number of parameters to the transformer. The classifier head acts on the final representation of the final layer of the LSTM. We examine the performance of the two architectures when trained using both multitasking and the proposed alternating minimization algorithm. First, we observe that the transformer generally performs better than the recurrent model. Second, the proposed training algorithm yields consistent improvements over the multitask baseline for the transformer. This shows that the proposed algorithm is general, but particularly effective when used in conjunction with the transformer.

Optimizing Task Embeddings We observed that both CAVIA and TAM generally attain better performance when trained with a larger number of inner loop updates. In this work, we use a maximum of 25 inner loop updates for TAM because it strikes a good balance between finding an optimal task embedding and containing training time. CAVIA performed best with 10 inner loop updates, beyond which we hit the computational limitations of our hardware. We also found that TAM works better when trained with a number of examples per task much greater than k, in our case 300. All these empirical findings suggest that optimizing for the task embedding and replacing the second order optimization with TAM's first order is an intrinsically difficult problem that requires more iterations and a larger number of examples.

Training Efficiency We discuss the training efficiency of different models in Table 4. The multitask baseline is not expensive to train, but it doesn't perform well on few-shot scenarios. CAVIA does well especially in the extreme few-shot scenarios, but has stability issues. TAM is simple, easy to implement, performs comparably or better than the baselines and trains more efficiently than CAVIA.

First vs. Second Order Gradients Double backprop has become a standard method of meta-learning. In our settings, we have found that first order gradients (via alternating minimization) are sufficient if done correctly, despite being simpler and more efficient. Although CAVIA sometimes outperforms TAM, especially for very small numbers of test examples, TAM is always competitive; with more test examples, TAM is usually superior. TAM always outperforms MAML.

5 Conclusion

In this work we have shown how task embeddings naturally fit with Transformers for few-shot learning. In our settings, this approach yields comparable or superior performance to approaches relying on second order derivatives while being computationally more efficient.

References

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pp. 3981–3989, 2016.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pp. 4331–4339, 2016.

Barrault, L., Bojar, O., Costa-jussà, M. R., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Huck, M., Koehn, P., Malmasi, S., Monz, C., Müller, M., Pal, S., Post, M., and Zampieri, M. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pp. 1–61, Florence, Italy, August 2019. Association for Computational Linguistics.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Fei-Fei, L., Fergus, R., and Perona, P. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4), 2006.

Fink, M. Object classification from a single example utilizing class relevance metrics. In *In Advances in Neural Information Processing Systems*, pp. 449—456, 2005.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

Hinton, G. E. and Plaut, D. C. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pp. 177–186, 1987.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, 2019.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kumar, A. and Daume III, H. Learning task grouping and overlap in multi-task learning. In *International Conference on Machine Learning*, 2012.

Lake, B. M. Compositional generalization through meta sequence-to-sequence learning. *arXiv preprint arXiv:1906.05381*, 2019.

Lample, G., Subramanian, S., Smith, E., Denoyer, L., Ranzato, M., and Boureau, Y.-L. Multiple-attribute text rewriting. In *International Conference on Learning Representations*, 2019.

Maurer, A., Pontil, M., and Romera-Paredes, B. Sparse coding for multitask and transfer learning. In *International conference on machine learning*, pp. 343–351, 2013.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Nichol, A. and Schulman, J. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M., Heess, N., and Hadsell, R. Stabilizing transformers for reinforcement learning. *arXiv:1910.06764*, 2019.

Purushwalkam, S., Nickel, M., Gupta, A., and Ranzato, M. Task-driven modular networks for zero-shot compositional learning. *arXiv preprint arXiv:1905.05908*, 2019.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Shen, T., Ott, M., Auli, M., and Ranzato, M. Mixture models for diverse machine translation: Tricks of the trade. In *International Conference on Machine Learning*, 2019.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

Stickland, A. C. and Murray, I. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Zellers, R., Holtzman, A., Rashkin, H., Bisk, Y., Farhadi, A., Roesner, F., and Choi, Y. Defending against neural fake news. In *Neural Information Processing Systems*, 2019.

Zintgraf, L. M., Shiarlis, K., Kurin, V., Hofmann, K., and Whiteson, S. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, 2019.