Learning to Optimize with Recurrent Hierarchical Transformers

Abhinav Moudgil [1 2] **Boris Knyazev** [3] **Guillaume Lajoie** [2 4] **Eugene Belilovsky** [1 2]

Abstract

Learning to optimize (L2O) has received a lot of attention recently because of its potential to leverage data to outperform hand-designed optimization algorithms such as Adam. However, they can suffer from high meta-training costs and memory overhead. Recent attempts have been made to reduce the computational costs of these learned optimizers by introducing a hierarchy that enables them to perform most of the heavy computation at the tensor (layer) level rather than the parameter level. This not only leads to sublinear memory cost with respect to number of parameters, but also allows for a higher representation capacity for efficient learned optimization. To this end, we propose an efficient transformer-based learned optimizer which facilitates communication among tensors with self-attention and keeps track of optimization history with recurrence. We show that our optimizer converges faster than strong baselines at a comparable memory overhead, thereby suggesting encouraging scaling trends.

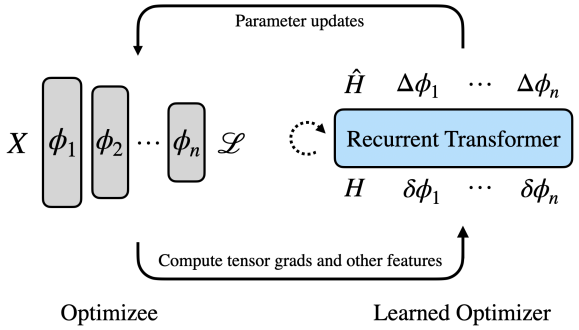

Figure 1. **Recurrent hierarchical transformer architecture for learned optimization.** We propose a novel learned optimizer based on a recurrent transformer which utilizes optimizee network structure for optimization. Optimizee network is assumed to have a stack of N neural modules or layers with parameters $\phi_1, \phi_2, \ldots, \phi_n$. After backpropagation with loss $\mathcal{L}$ on an input batch X, our optimizer takes as input tensor features like the mean of tensor gradients $\delta\phi_1, \delta\phi_2, \ldots, \delta\phi_n$, momentum, etc. along with a global hidden state H. This enables inter-tensor communication through self-attention after which weight updates for each parameter in a tensor are obtained by applying an MLP on top of these transformed tensor features. Finally, the weight updates are applied to the optimizee network to continue optimization.

1. Introduction

There has been immense progress in artificial intelligence through the use of deep neural networks which can learn features directly from the data. It has been empirically shown in various domains (Yan et al., 2015; Krizhevsky et al., 2017; Sutskever et al., 2014; Cho et al., 2014) that these data-driven features significantly outperform hand-crafted features in specific tasks. Training these neural networks can be challenging, and the scale of datasets and models continues to increase (Kaplan et al., 2020). However, the dominant optimizers used to train these neural networks to date like SGD (Robbins & Monro, 1951), Adam (Kingma & Ba, 2015), RMSProp (Graves, 2013), etc, are all hand-designed, and often heuristic. Hence, much remains to be explored when it comes to the "automation" of these underlying optimization algorithms. To this end, there has been significant interest in the area of learned optimization (Andrychowicz et al., 2016; Metz et al.; 2019; Almeida et al., 2021; Metz et al., 2022b), which aims to learn these underlying optimization algorithms themselves and outperform the hand-designed ones. These learned optimizers are typically meta-learned on "optimization tasks"(Metz et al., 2020a;b; 2022a) with each task specifying an objective function, neural network architecture and dataset. Moreover, since the learned optimizers are typically parameterized by neural networks, in principle, they could learn more complex optimization functions than the ones specified by hand-designed algorithms in order to optimize faster.

Several learned optimizers (Metz et al.; Almeida et al., 2021; Metz et al., 2020a) have been proposed in the past which can be broadly classified into two categories: hierarchical and non-hierarchical. Non-hierarchical optimizers (Metz

[1]Concordia University, Montréal, Canada [2]Mila [3]Samsung - SAIT AI Lab, Montréal [4]Universtié de Montréal. Correspondence to: Abhinav Moudgil <abhinav.moudgil@mila.quebec>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023.

et al.; 2022a) typically operate at the parameter level and do not take into account the in-built structure of optimizee neural networks. Since the non-hierarchical optimizers apply a complex function to each parameter independently, their memory overhead scales linearly with the number of parameters. Thus, to keep the memory overhead small, they are often parameterized by simple functions such as MLPs (Metz et al., 2022a; Harrison et al., 2022). This limits the representation capacity and hence the scalability of these optimizers.

On the other hand, hierarchical optimizers (Wichrowska et al., 2017; Metz et al., 2020a; 2022b; Peebles et al., 2022; Knyazev et al., 2021) have the potential to move beyond parameter level and process groups of parameters (layers or tensors) to learn more complex functions for faster optimization. Peebles et al. (2022) propose a diffusion transformer which takes all parameters along with the desired loss as input and returns the evolved parameters that achieve the desired loss in one step. This single-step paradigm, however, suffers from a generalization issue in that it can not be applied to new problems after being learned on a specific task. Metz et al. (2020a) propose a hierarchical RNN-based optimizer that maintains hidden states for tensors which also communicate among themselves and give per parameter updates through an MLP conditioned on these tensor states. This approach has been shown to generalize to unseen tasks.

In this work, we propose a novel hierarchical learned optimizer based on a *transformer* for modelling interactions among tensors while also maintaining a low compute overhead similar to prior work (Metz et al., 2020a). Our proposed optimizer (Fig. 1) performs a majority of the computation at the level of tensors (layers or structured groups of parameters), leading to a sub-linear memory cost in terms of the number of parameters. Moreover, it keeps track of optimization history with a single global hidden state instead of per-tensor hidden states as in prior work (Metz et al., 2020a; 2022b). We show that our optimizer outperforms strong baselines including both hand-designed and prior-learned optimizers. To the best of our knowledge, this is the first application of transformers in the learned optimization domain for neural network tasks and paves the way forward for future work.

2. Architecture

Our proposed architecture is hierarchical and recurrent at its core in order to do faster optimization within a limited computing budget (Fig. 1). It employs a transformer encoder (Vaswani et al., 2017; Devlin et al., 2019) with bi-directional attention to model interactions among tensor features to give parameter updates. Specifically, it takes input parameter values, gradients, and current state and gives updated state and parameter updates. It constructs tensor features which are derived from parameter values and gradients such as the mean of gradient values in a tensor, momentum, second-moment accumulation, shape of tensor, etc. We refer the reader to (Metz et al., 2020a) for a full description of these features. These tensors tensor features are encoded as tokens with a simple linear projection. A hidden state (whose initial value is also learned) is additionally concatenated with these tokens and fed into a transformer encoder model which synthesizes communication and gives transformed tokens.

These transformed tensor tokens are then passed through a linear layer individually to give a small conditioning embedding specific to each tensor. Finally, per parameter values along with their gradients, conditioning embedding and other derived features such as momentum, etc are passed through an MLP to give parameter updates (Metz et al., 2020a). Note that the same conditioning embedding is passed to the MLP for all parameters in a single tensor since it is specific to the tensor. A hidden state output from the transformer is simply passed as input to the next timestep without any additional processing.

Implementation details. Our transformer encoder consists of 4 layers with 4 attention heads and bi-directional masking. We embed all the tensor features and hidden state to size 64 and we use key size of 32 in multi-head attention for faster throughput without losing performance. Following (Metz et al., 2020a), we construct a total of 18 tensor features utilizing tensor gradients and parameter values; following the self-attention step, we obtain a conditional tensor embedding of size 17 through a linear projection of transformed tensor tokens which is then passed as input to the MLP. We implement our optimizer in JAX (Bradbury et al., 2018) using the learned optimization[1] open-source library (Metz et al., 2022a).

3. Training

Following prior work (Metz et al., 2022a; Harrison et al., 2022), we use Persistent Evolutionary Strategies (PES) (Vicol et al., 2021) to meta-train our optimizer and all the learned optimizer baselines evaluated in this work. For a fair comparison with prior work (Metz et al., 2022a; Harrison et al., 2022), we meta-train on the same two tasks, namely Fashion MNIST with 2 hidden layers of 128 size each and a CIFAR-10 task with a 3-layer ConvNet with 32, 64, and 64 filters. Both these tasks use batch size of 128. We refer the reader to (Metz et al., 2022a) for all details about these tasks. In total, we do 100,000 outer iterations of meta-training with each outer iteration consisting of $\leqslant$2000 inner iterations of learned optimizer rollout and we use the mean training loss over these inner iterations as our meta-objective. The inner rollout length during meta-training is log uniformly sampled

[1] https://github.com/google/learned_optimization

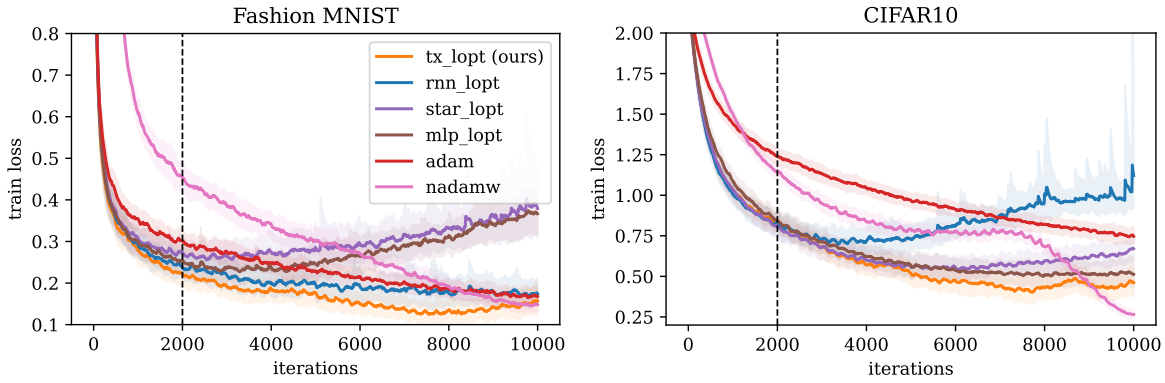

Figure 2. **Comparing our optimizer with strong learned and hand-designed optimizer baselines.** We meta-train all learned optimizers on a Fashion MNIST task with a 2-layer MLP (**left**) or a CIFAR-10 task with a ConvNet (**right**) with the maximum unroll length of 2K iterations. We test all optimizers on these tasks for 10K iterations, which is well beyond the meta-training length (2K) of our and baseline learned optimizers (see more details in Section 3). Our optimizer outperforms baseline learned optimizers (lopt) and performs competitively with hand-designed baselines heavily tuned on 10-1000s of tasks.

between 100 and 2000. We meta-train on 4 Nvidia RTX 8000 GPUs with a learning rate of 1e-4 and use truncation length of 50 with a standard deviation of 0.01 in PES.

4. Experiments

Following prior work (Harrison et al., 2022; Metz et al., 2022a), we meta-train our learned optimizer on each of the aforementioned Fashion MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky et al., 2009) tasks separately. We compare with the three recent learned optimizers: STAR LOpt (Harrison et al., 2022), RNN LOpt (Metz et al., 2020a), and MLP LOpt (Metz et al., 2022a). Although Velo (Metz et al., 2022b) is another recent learned optimizer, training it has an extremely large computational cost for meta-training (four thousand TPU months in the original work) in part due to using a full ES algorithm rather than the truncated PES one which we and the other learned baselines use. In our preliminary experiments, truncated PES could not stably train Velo. Therefore, we omit this comparison in this work and leave this direction for future work.

We also compare with strong hand-designed baselines, namely Adam (Kingma & Ba, 2015) and NAdamW (Metz et al.), which were tuned well for the Fashion MNIST and CIFAR-10 tasks in (Metz et al.; Harrison et al., 2022; Metz et al., 2022a). Specifically, Adam was tuned with 15 different learning rates sampled logarithmically and NAdamW was tuned with 1000 random trials with different hyperparameter settings serving as an extremely strong baseline for these tasks. For both of these baselines, we compare to the hyperparameters corresponding to the lowest training loss after 10K iterations (the maximum number of iterations we and the baseline learned optimizers typically consider). Following prior work (Harrison et al., 2022; Metz et al., 2022a), we focus on the training loss since that is the meta-objective with which we train our optimizer. We test each optimizer with 5 random seeds and show the training plots in Fig. 2.

4.1. Results

We benchmark our learned optimizer for 10K inner iterations after meta-training as done in (Harrison et al., 2022) and show training plots in Fig. 2. As evident from the plots, our learned optimizer achieves the lowest training loss and optimizes faster than all the other learned optimizers. It is important to note that unrolling the learned optimizers for 10K iterations is a strong test of generalization, since our approach and all the learned optimizer baselines were meta-trained for a maximum of 2K iterations. Moreover, we do not add any extra features (as in (Harrison et al., 2022)) which could help our learned optimizer generalize better, so this is purely a zero-shot generalization of our learned optimizer to longer context lengths.

In order to be fair in comparison, we do not use weight decay in any learned optimizer as used in some prior works (Harrison et al., 2022; Metz et al., 2022b). Without weight decay, we found that STAR optimizer (Harrison et al., 2022) overfits more than the MLP baseline and consequently performs slightly poorly in benchmarking. It is also interesting that the RNN baseline with per-tensor hidden states generalizes well beyond its meta-training length of 2K iterations on the simple Fashion MNIST task, but performs poorly on the CIFAR-10 ConvNet task. In contrast, our optimizer employing a global hidden state with tensor-level self-attention is able to generalize on both tasks. Our optimizer outperforms Adam on the CIFAR10 task and achieves a similar loss as Adam at the end on the Fashion MNIST task. A well-tuned NAdamW achieves the lowest training loss at the

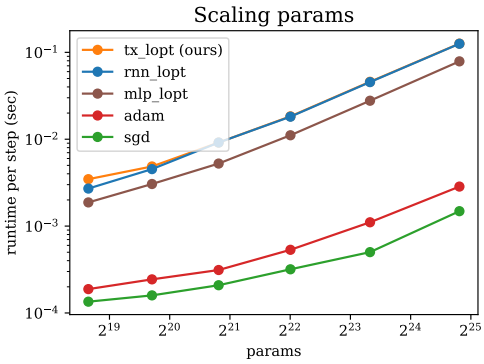

Figure 3. **Runtime of our optimizer scales linearly with the number of parameters demonstrating negligible transformer cost.** Since our optimizer is hierarchical, the per-parameter update cost heavily dominates the runtime cost of the recurrent transformer leading to linear scale-up like all the other baselines.

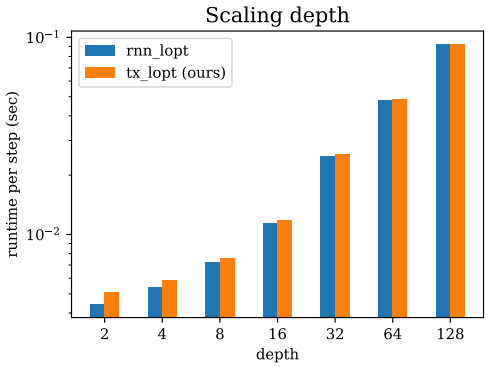

Figure 4. **Runtime gap between our optimizer and RNN diminishes as the depth increases.** This allows us to do heavy computation at tensor-level (self-attention with quadratic complexity instead of RNN) for better performance without sacrificing runtime at scale. Even at the lowest depth of 2, the absolute difference in runtime between RNN and ours is still minimal (0.6ms per step).

end (slightly below ours), but we note this is a strong handcrafted optimizer baseline tuned on thousands of tasks (Metz et al.; 2022b). On the other hand, our optimizer achieves the strongest performance in the 2K step window for which it is meta-trained. By scaling meta-training to more tasks and a longer training horizon, it could be possible to substantially outperform this NAdamW baseline for longer time horizons.

4.2. Runtime Analysis

We analyze the runtime of our proposed optimizer along two axes: (1) scaling the number of parameters, (2) scaling the number of layers, i.e. depth in the underlying task. We perform these runtime experiments on the CIFAR10 task with an MLP and measure runtime per step (in seconds). In this section, we analyze how the runtime of our proposed optimizer scales with respect to others.

Scaling parameters. We perform the first scaling experiment on the CIFAR10 task with a 2-layer MLP with 128 units. We scale the number of parameters by increasing the width in this MLP by a factor of 2 till we reach the width of 4096 and show runtime for all the considered optimizers in Fig. 3. As expected, in all the optimizers, runtime per step scales linearly with the number of parameters. Our optimizer shows negligible overhead over the RNN LOpt baseline and its runtime matches the runtime of RNN LOpt at scale. Since both these optimizers are hierarchical, their runtime can be roughly broken down into parts (Metz et al., 2022b): (1) fixed overhead, (2) per-parameter update cost which scales linearly with the number of parameters. The per-parameter cost becomes dominant with the increase in the number of parameters, hence leading to identical runtimes of our approach and RNN LOpt at scale.

Scaling depth. We increase the number of layers in the aforementioned CIFAR10 task from 4 to 128 (keeping the width of 256 fixed) and compare the runtime of our optimizer with the hierarchical RNN baseline in Fig. 4. The key result here is that although we use a transformer whose runtime cost scales quadratically with the increase in the number of layers (depth), we still observe vanishing overhead with respect to RNN LOpt. Due to the hierarchical structure of our optimizer, the additional cost of our recurrent transformer diminishes in comparison with the update cost of parameters by the MLP resulting in overall similar runtimes. This shows that the runtime of our hierarchical optimizer only depends on per-parameter cost at scale and hence the cost of our proposed recurrent transformer becomes negligible. The maximum gap between the runtimes of our approach and the RNN baseline across depths is still quite low (0.6ms per step at depth 2).

Overall, our learned optimizer enjoys similar runtime and memory overhead as the RNN LOpt baseline while maintaining superior performance.

5. Conclusion

We propose an efficient learned optimizer which uses a transformer with recurrence and leverages the structure of neural networks to perform optimization. We test our optimizer on two image recognition tasks and show that it outperforms prior learned approaches and is on par or better than heavily tuned hand-designed baselines. In addition, we show that the recurrent transformer in our optimizer architecture has minimal overhead and its runtime cost vanishes at scale, due to its hierarchical structure. To the best of our knowledge, this is the first work which utilizes a transformer in a learned optimizer for neural network tasks and paves the way forward for future work in this direction.

References

Almeida, D., Winter, C., Tang, J., and Zaremba, W. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021. 1

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. *Advances in neural information processing systems*, 29, 2016. 1

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax. 2

Cho, K., Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014. 1

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 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)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423. 2

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 1

Harrison, J., Metz, L., and Sohl-Dickstein, J. A closer look at learned optimization: Stability, robustness, and inductive biases. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=cxZEBQFDoFK. 2, 3

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980. 1, 3

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero, A. Parameter prediction for unseen deep architectures. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=vqHak8NLk25. 2

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6(1):1, 2009. 3

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 1

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. Learning unsupervised learning rules. 1, 3, 4

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pp. 4556–4565. PMLR, 2019. 1

Metz, L., Maheswaranathan, N., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020a. 1, 2, 3

Metz, L., Maheswaranathan, N., Sun, R., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*, 2020b. 1

Metz, L., Freeman, C. D., Harrison, J., Maheswaranathan, N., and Sohl-Dickstein, J. Practical tradeoffs between memory, compute, and performance in learned optimizers. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2022a. URL http://github.com/google/learned_optimization. 1, 2, 3

Metz, L., Harrison, J., Freeman, C. D., Merchant, A., Beyer, L., Bradbury, J., Agrawal, N., Poole, B., Mordatch, I., Roberts, A., et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022b. 1, 2, 3, 4

Peebles, W., Radosavovic, I., Brooks, T., Efros, A., and Malik, J. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022. 2

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951. 1

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014. 1

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2

Vicol, P., Metz, L., and Sohl-Dickstein, J. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning*, pp. 10553–10563. PMLR, 2021. 2

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Freitas, N., and Sohl-Dickstein, J. Learned optimizers that scale and generalize. In *International conference on machine learning*, pp. 3751–3760. PMLR, 2017. 2

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 3

Yan, L. C., Yoshua, B., and Geoffrey, H. Deep learning. *nature*, 521(7553):436–444, 2015. 1