TaskSet: A Dataset of Optimization Tasks

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present TaskSet, a dataset of tasks for use in training and evaluating optimizers. TaskSet is unique in its size and diversity, containing over a thousand tasks ranging from image classification with fully connected or convolutional neural networks, to variational autoencoders, to non-volume preserving flows on a variety of datasets. As an example application of such a dataset we explore meta-learning an ordered list of hyperparameters to try sequentially. By learning this hyperparameter list from data generated using TaskSet we achieve large speedups in sample efficiency over random search. Next we use the diversity of the TaskSet and our method for learning hyperparameter lists to empirically explore the generalization of these lists to new optimization tasks in a variety of settings including ImageNet classification with Resnet50 and LM1B language modeling with transformers. As part of this work we have opensourced code for all tasks, as well as 29 million training curves for these problems and the corresponding hyperparameters.[1]

1 Introduction

As machine learning moves to new domains, collecting diverse, rich, and application-relevant datasets is critical for its continued success. Historically, research on learning optimization algorithms have only leveraged single tasks [4, 77], or parametric synthetic tasks [123], due to the difficulty of obtaining large sets of tasks.

1.1 TaskSet: A set of tasks

We present a set of tasks significantly larger than any optimizer dataset previously studied. We aim to better enable standardized research on optimizers, be that analysis of existing optimizers, or development of new learned learning algorithms. We call this suite of tasks TaskSet.

Much in the same way that learned features in computer vision outpaced hand designed features [62, 65], we believe that data driven approaches to discover optimization algorithms will replace their hand designed counterparts resulting in increased performance and usability. To this end, standardizing a large suite of optimization tasks is an important first step towards more rigorous learned optimizer research.

In this setting, a single "example" is an entire training procedure for a task defined by data, loss function, and architecture. Thus, TaskSet consists of over a thousand optimization tasks, largely focused on deep learning (neural networks). They include image classification using fully connected and convolutional models, generative models with variational autoencoders [58] or flows [30, 85], natural language processing tasks including both language modeling and classification, as well as

[1] redacted url

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

synthetic tasks such as quadratics, and optimization test functions. The problems themselves are diverse in size, spanning 7 orders of magnitude in parameter count, but remain reasonably fast to compute as almost all tasks can be trained 10k iterations on a CPU in under one hour. To demonstrate the breadth of this dataset we show an embedding of all the tasks in Appendix A.1 in Figure S1.

1.2 Amortizing hyperparameter search

Machine learning methods are growing ever more complex, and their computational demands are increasing at a frightening pace [3]. Unfortunately, most modern machine learning models also require extensive hyperparameter tuning. Often, hyperparameter search is many times more costly than the final algorithm, which ultimately has large economic and environmental costs [106].

The most common approach to hyperparameter tuning involves some form of quasi-random search over a pre-specified grid of hyperparameters. Building on past work [126, 90], and serving as a typical example problem illustrative of the sort of research enabled by TaskSet, we explore a hyperparameter search strategy consisting of a simple ordered list of hyperparameters to try. The idea is that the first few elements in this list will cover most of the variation in good hyperparameters found in typical machine learning workloads.

We choose the elements in this list by leveraging the diversity of tasks in TaskSet, by meta-learning a hyperparameter list that performs the best on the set of tasks in TaskSet. We then test this list of hyperparameters on new, larger machine learning tasks.

Although learning the list of hyperparameters is costly (in total we train $\sim$29 million models consisting of over 4,000 distinct hyper parameter configurations), our final published list is now available as a good starting guess for new tasks.

Furthermore, we believe the raw training curves generated by this search will be useful for future hyperparameter analysis and meta-learning research, and we release it as part of this work. We additionally release code in Tensorflow [2], Jax [17], and PyTorch [86] for a reference optimizer which uses our learned hyperparameter list, and can be easily applied to any model.

2 TaskSet: A set of tasks

How should one choose what problems to include in a set of optimization tasks? In our case, we strive to include optimization tasks that have been influential in deep learning research over the last several decades, and will be representative of many common machine learning problems. Designing this dataset requires striking a balance between including realistic large-scale workloads and ensuring that tasks are fast to train so that using it for meta-learning is tractable. We construct our dataset largely out of neural network based tasks. Our chosen tasks have between ten thousand and one million parameters (much smaller than the billions commonly used today), as a result most problems can train in under an hour on a cloud CPU with 5 cores. We additionally focus on increased "task diversity" by including many different kinds of training algorithms, architectures, and datasets – inspired by past work in reinforcement learning which has demonstrated large numbers of problems and increased diversity around some domain of interest is useful for both training and generalization [50, 112, 27, 84]. Again though, a balance must be struck, as in the limit of too much diversity no learning can occur due to the no free lunch theorem [127]. Our dataset, TaskSet, is made up of 1162 tasks in total. We define a task as the combination of a loss function, a dataset, and initialization.

Specifically we define a task as a set of 4 functions:

- **Initialization**: () $\rightarrow$ parameter initial values
- **Data generator**: data split (e.g. train / valid / test) $\rightarrow$ batch of data
- **Forward pass**: (batch of data, params) $\rightarrow$ loss
- **Gradient function**: (input data, params) $\rightarrow$ gradients ($\frac{\text{dloss}}{\text{dparams}}$)

A task has no tunable hyperparameters and, coupled with an optimizer, provides all the necessary information to train using first order optimization. This makes experimentation easier, as each task

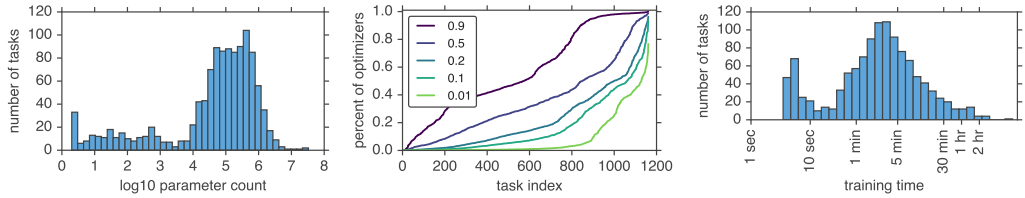

Figure 1: **(a)** A histogram of parameter counts for each problems in the task suite. Our task suite spans more than 7 orders of magnitude in model size. **(b)** Percentage of optimizers (y-axis) capable of reaching a given loss value (color) for tasks (x-axis). We find there exists around 100 "easy" tasks on which more than half of the optimizers perform well, and a large number of "difficult" tasks for which almost no optimizers perform well. **(c)** A histogram of training times. Almost all tasks can be trained in under an hour.

definition specifies hyperparameters such as batch size [102, 74] or initialization [100, 132, 131, 68, 91, 47, 56, 14, 48] that no longer need to be tuned.

We augment a set of "fixed" tasks which have been designed by hand, with "sampled" tasks that are randomly generated task instances.

2.1 Sampled families of tasks

Sampled tasks are created by sampling neural network architectures (e.g., MLPs, convnets), activation functions, datasets (e.g., images, text, quadratic functions, and synthetic tasks), and other properties. We organize these sampled tasks into similar *families* of tasks. See Appendix H for a complete description of these sampled tasks. Broadly, these are separated into tasks sampling image models (*mlp, mlp_ae* [51]*, mlp_vae* [58]*, conv_pooling, conv_fc, nvp* [30]*, maf* [85]), tasks sampling language models (*char_rnn_language_model* [44]*, word_rnn_language_model, rnn_text_classification*), quadratics (*quadratic*) and other synthetic tasks (*losg_tasks* [123]). Defining a sampling distribution that generates tasks that are always valid, and that run within a time constraint, is difficult. Instead, we define a broad distribution and make use of rejection sampling to remove tasks that are either too slow or that we are unable to optimize at all. By starting with a distribution that is too broad, and pruning it, we hope to achieve better coverage of tasks.

2.2 Hand designed tasks

In addition to the sampled tasks, we also include 107 hand designed tasks. These consist of more common tasks that both improve the coverage beyond the sampled tasks, and provide for better interpretability through a closer match to existing tasks in the literature. These tasks span image classification, text classification, language modeling, and generative modeling, as well as some synthetic tasks such as associative retrieval [5]. We leave the description of each one of these tasks to Appendix H.3.

2.3 Aggregate Statistics of TaskSet

In Figure 1a we show histograms of compute times for all problems and find almost all problems train under an hour (see Appendix C for per task family histograms). In Figure 1c we plot a histogram of the number of parameters per tasks. Finally, in Figure 1b we show a distribution of task difficulty by plotting the fraction of optimizer configurations that achieve a certain loss value. We find that for some tasks as many as 50% of optimizers perform well while for others $< 1\%$ achieve a loss close to the smallest observed loss. For a qualitative visualization of TaskSet, see Appendix A

3 Amortized hyperparameter search

As a simple demonstration of using TaskSet for meta-learning research, we consider learning hyperparameter lists. This idea of learning lists of hyper parameters has been explored in [126, 90]. We define an optimizer as the pairing of an optimization algorithm and all its corresponding hyperparameters

(e.g. learning rate). While sometimes practitioners use a single optimizer – e.g. Adam [57] with default hyperparameters – most practitioners will often run multiple optimizers and use a validation set to select the best performer.

3.1 Optimizer families

We define different parameterizations of hand designed optimizers as an optimizer family. The optimizer families we consider consist of:

- *Adam1p*: One hyperparameter, the fixed learning rate α
- *Adam4p*: Four Adam hyperparameters, α, β_1, β_2, and ϵ
- *Adam6p*: Adam4p hyperparameters, and two additional hyperparameters controlling linear and exponential learning rate decays
- *Adam8p*: The hyperparameters in *Adam6p* plus two additional hyperparameters for $\ell 1$ and $\ell 2$ regularization terms
- *NAdamW*: A 10 hyperparameter search space based on NAdam [33] with cosine learning rate decay, and weight decay.

For the full update equations see Appendix D.1 for Adam and D.2 for NadamW. We chose Adam based on its use in existing work, and NAdam based on performance shown in [24].

3.2 Learned hyperparameter lists

Traditionally researchers tune hyperparameters on a per model basis. While this often results in performance gains; it comes at the cost of immense compute, and researchers are almost never able to expend enough compute to saturate model performance [102]. As an alternative to per-problem tuning, we proposes instead tuning the search strategy itself on a dataset of tasks and *transferring* the knowledge gained to new tasks of interest. This idea is already implicitly done by humans – e.g. we don't start a hyperparameter search with a learning rate of 10^6 – we use values that the community has found useful.

This dataset-based tuning has a number of desirable properties. First, the resulting search strategies are much more efficient, resulting in large speedups in sample efficiency on unseen tasks over a random search baseline. Second, we are less restricted by the number of optimizer parameters we search over or by needing to define reasonable search spaces. For example, if there are redundant regions of search space, our learned optimizer will be less likely to sample them repeatedly, unlike random search. If there is a region of hyperparameter space that performs poorly on all problems, the learned search strategy will avoid it.

In this work we parameterize the learned search strategy as an ordered list of optimizers to try (i.e. a list of hyperparameter configurations). Given a fixed number of task evaluations we would like to achieve the best possible performance on all tasks in the training set of tasks. For a length k list of optimizers we define our loss as:

$$J(\theta_{1,\ldots,k}) = \sum_{\tau \in \text{tasks}} \left[\min_{i \in 1..k} f(\tau, \theta_i) \right], \tag{1}$$

where θ_i are the optimizer hyperparameters for element i in the list, and f is an appropriately normalized loss computed after training task τ.

We seek to find an optimal list of optimizers as (similar to [126]):

$$\theta^*_{1,\ldots,k} = \arg\min_{\theta_{1,\ldots,k}} J(\theta_{1,\ldots,k}). \tag{2}$$

This is meant to serve as an example task, illustrative of the sort of research enabled by TaskSet. More advanced hyperparameter search strategies would no doubt yield even more performant results.

3.3 Scoring an optimizer by averaging over tasks

To score a task, we initialize the parameters of the task and run 10,000 iterations of an optimizer. We monitor loss on each data split (train, validation, test) every 200 steps using an average over 50 mini-batches per evaluation. For all data presented in this paper we also compute averages over 5 random task parameter initializations.

A side effect of the diverse task dataset is that losses span multiple orders of magnitude, making direct aggregation of performance problematic. To remedy this we normalize the loss values for all tasks linearly between 0 and 1 where 1 is validation loss at initialization and zero is the lowest validation loss achieved by any tested optimizer. Loss values greater than the loss at initialization are clipped to 1. To collapse an entire normalized training curve into a scalar cost, we compute the mean normalized loss over the 10,000 iterations. We find empirically that this choice is similar to taking the minimum (Appendix B.5). We leave exploring alternative methods such as performance profiles [31] and Nash averaging [6] for future work.

3.4 Greedy learning from random search

Optimizing Eq. 2 is combinatorially expensive. To tractably solve this optimization problem, we introduce two approximations [126]. First, we shift the unconstrained search over the full space of optimizers to search over a finite set of optimizers, Θ. This finite set can be computed ahead of time and decouples the expensive procedure of training each task with an optimizer from training the learned search space. Separating data and training in this way has been done for both hyperparameter search [35], and neural architecture search [59, 133]. In total we trained 1,000 optimizer configurations for each of Adam1p, Adam4p, Adam6p, Adam8p, and NAdamW on all 1,162 tasks with 5 random seeds per pair. Second, we use a greedy heuristic to approximate the combinatorial search over sets of k optimizers. For a single optimizer trial, $k = 1$, we select the best performing optimizer on average across all training tasks. We then continue to select optimizer parameters such that the minimum of all optimizer-parameters per task, aggregated over all tasks is minimized. This shifts the complexity from exponential in k to linear. Finding a length k set of optimizers can thus be efficiently computed as follows:

$$\theta_1^* = \arg\min_{\theta \in \Theta} \left[\sum_{\tau \in \text{tasks}} f(\tau, \theta) \right] \tag{3}$$

$$\theta_k^* = \arg\min_{\theta \in \Theta} \left[\sum_{\tau \in \text{tasks}} \left[\min\left(b, f(\tau, \theta)\right) \right] \right] \qquad \text{where} \quad b = \min_{i \in 1..(k-1)} f(\tau, \theta_i^*). \tag{4}$$

We note that the first argument of the outer min, b, can be computed once per set of hyperparameters as it does not depend on θ. Finally, as our tasks are stochastic, we order optimizers based on validation loss and report test loss [116].[2]

This training strategy requires an original search space from which to collect data and build Θ. The search space we use is described in Appendix E.2. While large, we find that the optimal parameters for each task end up covering almost the entire space. At some point, no improvement can be obtained on any of the tasks in the dataset. At this point, we simply randomly order the remaining optimizers though expect more sophisticated methods could be employed.

4 Experiments: Training and generalization of learned hyperparameter lists

With our dataset of tasks and data collected, we turn our attention to exploring training of the hyperparameter lists, and generalization beyond the suite of tasks in TaskSet. In this exploration, we hope to give a flavor of the types of research possible with TaskSet. Our main tool to show performance are figures that sweep the number of optimizers configurations on the x-axis, and show the best performance achieved for each number of optimizers tried, averaged over some set of tasks (Eq. 1).

[2] This technically means that increasing the number of optimizes could potentially decrease performance, but we find this rarely happens in practice.

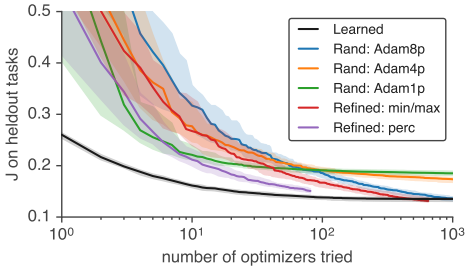

Figure 2: By learning a search space we achieve large speedups over random search. On the y-axis we show J, or the best aggregated and normalized performance achieved given some number of optimizer trials (x-axis). This is computed on heldout tasks not used to train the hyperparameter list. In solid we show median performance with 25-75 percentile shown with error bars over 50 resamplings of the train-test split of tasks, as well as random samplings. In black we show a learned search space computed from the Adam8p family of optimizes. In color we show various random search baselines. See §4.1 for description of these.

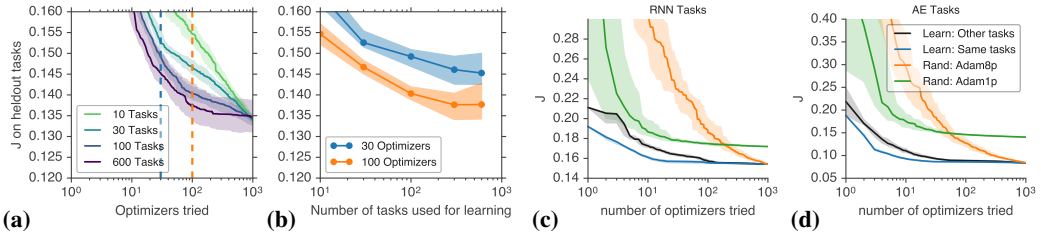

Figure 3: Using more tasks to train the search space results in improved performance on heldout tasks. **(a)** The number of optimizers tried vs performance on heldout tasks. In color, we show different numbers of tasks used to learn the search spaces. We show median performance with error bars denoting 25 and 75 percentile. **(b)** Performance at a fixed number of optimizers tried vs number of tasks used for meta-training. We find performance continues to improve as we meta-train on more tasks. These plots are slices out of (a), with colors matching the vertical dashed lines. **(c,d)** We show aggregate performance (J) as a function of number of optimizers tried when training the hyperparameter list on a different distributions of tasks than those we test on. We show testing on RNNs **(c)** and auto encoders **(d)**, and train on 700 tasks sampled from the remainder set of tasks. We find that these learned search spaces perform much better than random search in both the learning rate search space and in the original Adam8p search space. We additionally plot the best case performance – the case where we train and test on the same problem type. We show median and 25-75 percentile averaged over 50 different samplings.

4.1 Learned hyperparameter lists are more efficient than random search

To demonstrate the impact of learning a search space, we take the 1,162 tasks split them into even train and test tasks. We then learn a search strategy using optimizers from the Adam8p family following Eq. 4 on the train tasks. Results in Figure 3. As baselines, we use random search with different search spaces, including just learning rate (Rand: Adam1p), the default Adam hyper parameters (Rand: Adam4p), as well as the Adam 8 dimensional search space (Rand: Adam8p). To better get a sense of performance, we show two additional "Refined" baselines which involve random sampling from better search space. For min/max, we sample from the minimum bounding box containing the best hyperparameters for each task. To improve the search space quality, we shrink this bounding box so 90% of the best hyperparameters are enclosed. Further considerations regarding search space volume are treated in E.1, and the precise search spaces are specified in Appendix E.2. Finally, one difficulty of working with offline data is the difficulty of running online hyperparameter optimization methods such as Bayesian Optimization without running additional compute. Future work will explore offline Bayesian methods.

4.2 More tasks lead to better generalization

We next look at the effects of the number of training tasks on generalization. We take subsets of tasks of different size, and train hyperparameter lists using Eq.4. We compute test performance on the remainder of the tasks and plot loss averaged over different splits in Fig. 3. We find that a large number of tasks (more than 100) are required to achieve near-optimal test performance. This is surprising to us given how simple our learned search strategy is (simply a list of hyperparameters), but not wholly so given past work studying generalization in RL [27].

4.3 Generalization to different types of problem

For learned algorithms to be generally useful, some amount of generalization to unseen task *families* is required. To test this, we split our data into disjoint task types. We perform two splits: testing on RNN tasks and training on all others, and testing on autoencoder tasks and training on all others. As a best case baseline we additionally train search spaces on the test task families directly. We find an order of magnitude better sample efficiency than random search for both cases and find our learned search space is close in performance to search spaces trained on just the testing tasks (Fig. 3).

5 Experiments: Realistic problems

In §4.3 and §B.1 we explored generalization of learned hyperparameter lists to held out tasks within the TaskSet dataset. While useful for analysis, these tasks are still far from the workloads commonly employed to solve real problems. In this section, we explore the performance of our learned search space on a number of state of the art models. These models drastically differ from the training set of tasks in parameter count and compute cost. We see these experiments as evidence that the tasks presented in TaskSet capture enough of the structure of "realistic" problems that TaskSet can be used to improve larger scale workloads. For all experiments in this section we take the optimizer ordering using the NAdamW optimizer family on all TaskSet tasks then apply the resulting search space to the target problem. The final ordered list of hyperparameters used is in Appendix G. We show results for ResNet50 on ImageNet, and Transformers on LM1B. Additional results with reinforcement learning using PPO are in Appendix B.2.

First we explore ImageNet classification using a ResNet50. on We take the TPU implementation with default settings from the official Tensorflow models repository [110], and swap out different optimizers. We show accuracy computed over the course of training as well as best performance for a given hyperparameter budget in Figure 4. We find that the learned search space vastly outperforms learning rate tuned Adam.

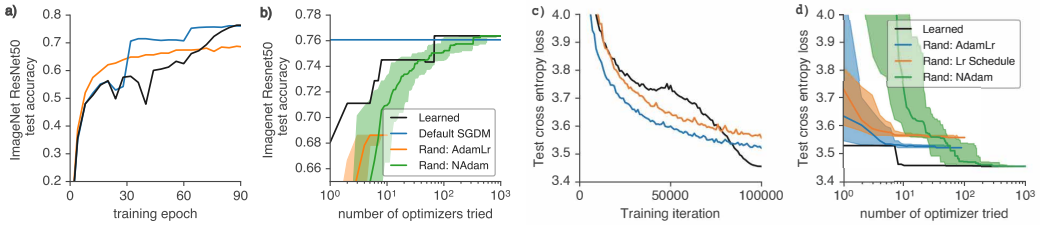

Figure 4: We find our learned optimizer list outperforms both learning rate tuned Adam and default training hyperparameters for ResNet50 and a 53M parameter Transformer trained on LM1B. **a)** Learning curves for the best hyperparameter from each optimizer class on the ResNet50 task. **b)** Number of optimizers tried vs best top 1 ImageNet accuracy achieved on the ResNet50 task. For NAdamW, we first train models with a validation set created by splitting the training set. We compute maxes over this set, and show the performance when retraining on the full training set on the official ImageNet validation set. For AdamLR, we simply compute a max over the official ImageNet validation set. **c)** Training curves for the best hyperparameter for each optimizer class on the Transformer task. **d)** Number of optimizers tried vs the test loss obtained given the best validation performance on the Transformer task.

Next we explore language modeling on LM1B with a Transformer. We take the transformer [118] example implemented in Jax [17] with Flax [39]. We train using a 2x2 TPU V2 configuration for 100k iterations. Once again we take all other hyperparameters as is and simply swap optimizer implementation. We find the learned hyperparameter list dramatically outperforms the default optimizer setting and the fixed learning rate baseline. Nevertheless, we emphasize that our method does not require any knowledge of the underlying problem to achieve faster results. See Appendix B.3 for this same transformer with a budget of 20k iterations.

6 Related Work

The idea of sets of tasks has been explored throughout machine learning. The majority of these suites are for use in evaluation where as our suite is targeted for meta-learning. The closest family of optimization tasks for evaluation to those presented here is DeepObs [99] which includes 20 neural network tasks. Our task suite focuses on smaller problems and contains 50x more tasks. Outside of evaluation, task suites in reinforcement learning such as Obstacle Tower [55], ProcGen [28], CoinRun [27], and Sonic [82] focus on training algorithms that work across a variety of settings.

The creation of TaskSet was motivated by the goal of learning learning algorithms, or meta-learning [98, 97, 53], and in particular learned optimizers [11, 4, 10, 123, 66, 71, 77, 78]. This use case is explored with this dataset in [79]. In this work we do not use this task suite to train learned optimizers, but instead focus on learning a hyperparameter search strategy. Tuning hyperparameters by leveraging multiple tasks has been explored within the contexts of Bayesian optimization [107, 87, 88] as well as meta-learning [94, 43, 38, 126, 125, 22, 90]. See Appendix F.1 for a full discussion of sets of tasks in machine learning, Appendix F.2 for more info on optimization in machine learning, and Appendix F.3 for a discussion on existing hyper parameter search methods.

7 Discussion

Learning optimization algorithms represents a promising direction for accelerating machine learning research. For the resulting algorithms to become useful tools, however, we must further understand the relationships between training tasks, meta-optimization, and both iid and out of distribution generalization.

This work takes steps towards this goal by introducing a significantly larger set of optimization tasks than ever previously considered. As an example use-case, we provide a thorough analysis of how TaskSet enables meta-optimization of simple, but performant hyperparameter lists. Despite this approach's simplicity, the training of learned learning algorithms is computationally expensive. We hope to explore alternative parameterizations which will increase efficiency by, e.g., leveraging previous evaluations or partial model training [108, 67].

We are releasing the optimal hyperparameter list we have found as a drop-in replacement optimizer in a variety of deep learning frameworks (Tensorflow [2], PyTorch [86], and JAX [17]) in the hopes that the research community finds them useful. We believe this represents a new set of reasonable optimizer defaults for new problems. Finally, we hope TaskSet encourages more standardized research on general purpose optimizers.

References

[1] URL https://s3.amazonaws.com/amazon-reviews-pds/readme.html.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

[3] Dario Amodei and Danny Hernandez. Ai and compute. *Heruntergeladen von https://blog. openai. com/aiand-compute*, 2018.

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

[5] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pp. 4331–4339, 2016.

[6] David Balduzzi, Karl Tuyls, Julien Perolat, and Thore Graepel. Re-evaluating evaluation. In *Advances in Neural Information Processing Systems*, pp. 3268–3279, 2018.

[7] David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech M Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. *arXiv preprint arXiv:1901.08106*, 2019.

[8] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[9] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[10] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc Le. Neural optimizer search with reinforcement learning. 2017. URL https://arxiv.org/pdf/1709.07417.pdf.

[11] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1990.

[12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[13] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 24*, pp. 2546–2554. Curran Associates, Inc., 2011.

[14] Yaniv Blumenfeld, Dar Gilboa, and Daniel Soudry. A mean field theory of quantized deep networks: The quantization-depth trade-off. *arXiv preprint arXiv:1906.00771*, 2019.

[15] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

[16] Olivier Bousquet, Sylvain Gelly, Karol Kurach, Olivier Teytaud, and Damien Vincent. Critical hyper-parameters: No random, no cry. *arXiv preprint arXiv:1706.03200*, 2017.

[17] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[18] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[19] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[21] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013. URL `http://arxiv.org/abs/1312.3005`.

[22] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 748–756. JMLR. org, 2017.

[23] Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*, 2018.

[24] Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

[25] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[26] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[27] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

[28] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2019.

[29] Ian Dewancker, Michael McCourt, Scott Clark, Patrick Hayes, Alexandra Johnson, and George Ke. A stratified analysis of bayesian optimization methods. *arXiv preprint arXiv:1603.09441*, 2016.

[30] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[31] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[32] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[33] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[34] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[35] Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[36] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[37] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[38] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Using meta-learning to initialize bayesian optimization of hyperparameters. In *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection-Volume 1201*, pp. 3–10. Citeseer, 2014.

[39] Flax Developers. Flax: A neural network library for jax designed for flexibility, 2020. URL `https://github.com/google-research/flax/tree/prerelease`.

[40] Wikimedia Foundation. Wikimedia downloads. URL `https://dumps.wikimedia.org`.

[41] Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, and Jonathan M Cohen. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *arXiv preprint arXiv:1905.11286*, 2019.

[42] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[43] Taciana AF Gomes, Ricardo BC Prudêncio, Carlos Soares, André LD Rossi, and André Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012.

[44] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[45] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`, 2018. URL `https://github.com/tensorflow/agents`. [Online; accessed 25-June-2019].

[46] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *arXiv preprint arXiv:1603.08785*, 2016.

[47] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the selection of initialization and activation function for deep neural networks. *arXiv preprint arXiv:1805.08266*, 2018.

[48] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. Mean-field behaviour of neural tangent kernel for deep neural networks, 2019.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[50] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[51] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[52] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[53] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

[54] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.). *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

[55] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378*, 2019.

[56] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of fisher information in deep neural networks: mean field approach. 2018.

[57] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[58] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[59] Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

[60] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. 2016.

[61] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[63] Manoj Kumar, George E Dahl, Vijay Vasudevan, and Mohammad Norouzi. Parallel architecture and hyperparameter search via successive halving and classification. *arXiv preprint arXiv:1805.10255*, 2018.

[64] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[65] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[66] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[67] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[68] Ping Li and Phan-Minh Nguyen. On random deep weight-tied autoencoders: Exact asymptotic analysis, phase transitions, and implications to training. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJx54i05tX`.

[69] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[70] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[71] Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. *arXiv preprint arXiv:1703.03633*, 2017.

[72] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

[73] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[74] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[75] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

[76] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[77] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pp. 4556–4565, 2019.

[78] Luke Metz, Niru Maheswaranathan, Jonathon Shlens, Jascha Sohl-Dickstein, and Ekin D Cubuk. Using learned optimizers to make models robust to input noise. *arXiv preprint arXiv:1906.03367*, 2019.

[79] Luke Metz, Niru Maheswaranathan, C Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020.

[80] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-20). 1996.

[81] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pp. 543–547, 1983.

[82] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

[83] Alex Olsen, Dmitry A. Konovalov, Bronson Philippa, Peter Ridd, Jake C. Wood, Jamie Johns, Wesley Banks, Benjamin Girgenti, Owen Kenny, James Whinney, Brendan Calvert, Mostafa Rahimi Azghadi, and Ronald D. White. DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning. *Scientific Reports*, 9(2058), 2 2019. doi: 10.1038/s41598-018-38343-3. URL `https://doi.org/10.1038/s41598-018-38343-3`.

[84] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *arXiv preprint*, 2019.

[85] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

[87] Valerio Perrone and Huibin Shen. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pp. 12751–12761, 2019.

[88] Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pp. 6845–6855, 2018.

[89] Johann Petrak. Fast subsampling performance estimates for classification algorithm selection. In *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, pp. 3–14. Citeseer, 2000.

[90] Florian Pfisterer, Jan N van Rijn, Philipp Probst, Andreas Müller, and Bernd Bischl. Learning multiple defaults for machine learning algorithms. *arXiv preprint arXiv:1811.09409*, 2018.

[91] Arnu Pretorius, Elan van Biljon, Steve Kroon, and Herman Kamper. Critical initialisation for deep signal propagation in noisy rectifier neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 5717–5726. Curran Associates, Inc., 2018.

[92] Prajit Ramachandran, Barret Zoph, and Quoc Le. Searching for activation functions. 2017.

[93] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

[94] Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta-learning for evolutionary parameter optimization of classifiers. *Machine learning*, 87(3):357–380, 2012.

[95] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[96] Tom Schaul, Ioannis Antonoglou, and David Silver. Unit tests for stochastic optimization. *arXiv preprint arXiv:1312.6055*, 2013.

[97] Juergen Schmidhuber. On learning how to learn learning strategies. 1995.

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

[99] Frank Schneider, Lukas Balles, and Philipp Hennig. Deepobs: A deep learning optimizer benchmark suite. *International Conference on Learning Representations*, 2019.

[100] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.

[101] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[102] Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

[103] Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and François Fleuret. On the tunability of optimizers in deep learning. *arXiv preprint arXiv:1910.11758*, 2019.

[104] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[105] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180, 2015.

[106] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[107] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pp. 2004–2012, 2013.

[108] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

[109] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. Technical report, DeepMind, January 2018. URL `https://arxiv.org/abs/1801.00690`.

[110] Tensorflow. tensorflow tpu resnet50, Oct 2019. URL `https://github.com/tensorflow/tpu/tree/master/models/official/resnet`.

[111] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

[112] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30. IEEE, 2017.

[113] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

[114] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[115] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[116] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[117] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL `http://doi.acm.org/10.1145/2641190.2641198`.

[118] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[119] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

[120] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[121] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint arXiv:1905.00537*, 2019.

[122] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[123] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *International Conference on Machine Learning*, 2017.

[124] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

[125] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pp. 1–10. IEEE, 2015.

[126] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Sequential model-free hyperparameter tuning. In *2015 IEEE international conference on data mining*, pp. 1033–1038. IEEE, 2015.

[127] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[128] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger B Grosse. Understanding short-horizon bias in stochastic meta-optimization. pp. 478–487, 2016.

[129] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[130] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3485–3492, June 2010. doi: 10.1109/CVPR.2010.5539970.

[131] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, 2018.

[132] Ge Yang and Samuel Schoenholz. Mean field residual networks: On the edge of chaos. In *Advances In Neural Information Processing Systems*, 2017.

[133] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. *arXiv e-prints*, art. arXiv:1902.09635, Feb 2019.

[134] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. The visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[135] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[136] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017. URL `https://arxiv.org/abs/1611.01578`.