WEIGHT-SHARING BEYOND NEURAL ARCHITECTURE SEARCH: EFFICIENT FEATURE MAP SELECTION AND FEDERATED HYPERPARAMETER TUNING

Mikhail Khodak [*1] **Liam Li** [*1] **Nicholas Roberts** [1] **Maria-Florina Balcan** [1] **Ameet Talwalkar** [1 2]

ABSTRACT

Hyperparameter optimization is a critical component of the machine learning pipeline. Although there has been much progress in this area, many methods for tuning model settings and learning algorithms are difficult to deploy in more restrictive settings such as federated learning. Recent progress in NAS has yielded a heuristic technique–weight-sharing, or the simultaneous optimization of multiple neural networks using the same parameters–that presents a promising new paradigm for hyperparameter optimization. In this paper we identify weight-sharing as a cheap, practical approach for more traditional hyperparameter optimization problems. We validate our claim with experiments on feature map selection problems where an approach combining weight-sharing with successive halving is able to find a good configuration much faster than full training. Finally, we propose a natural way of using weight-sharing to perform hyperparameter optimization for federated learning that enables learning a *tuned* model using data on *all* devices without significantly impacting on-device computation.

1 INTRODUTION

Weight-sharing has emerged as an useful optimization paradigm to reduce the computational cost of neural architecture search (NAS) (Pham et al., 2018; Liu et al., 2019; Cai et al., 2019). In lieu of training multiple architectures, weight-sharing reduces the training cost to that of a single super-network encompassing all possible architectures. While this computational gain comes at the expense of noisier signals of the quality of different architectures, there is evidence that weight-sharing is nonetheless able to provide useful signals for the selection of competitive architectures (Li & Talwalkar, 2019b; Guo et al., 2019; Zela et al., 2020).

Given NAS is a specialized instance of a hyperparameter optimization problem, many standard hyperparameter optimization methods are applicable to NAS. In this work, we instead study the efficacy of a *NAS-specific* method, i.e., weight-sharing, for general hyperparameter optimization problems. First, we validate the use of weight-sharing for hyperparameter optimization with two feature map selection experiments that demonstrate weight-sharing to be much faster than full training while still providing a strong enough signal to select a good configuration. Then, we propose a weight-sharing based algorithm for federated hyperparameter tuning and discuss the benefits of such an approach over traditional hyperparameter optimization methods.

1.1 Related Work

State-of-the-art hyperparameter optimization methods like Hyperband (Li et al., 2017), BOHB (Falkner et al., 2018), PBT (Jaderberg et al., 2017), and others (Kandasamy et al., 2017; Klein et al., 2017; Wu et al., 2019) typically exploit adaptive resource allocation in the form of early-stopping/partial training to dramatically speed up hyperparameter optimization. These methods work well in traditional settings where we have unlimited access to the underlying data and can train as many models as desired. However, these assumptions are typically violated in federated settings. We propose a novel weight-sharing based approach for federated hyperparameter tuning in Section 4 to address some of these limitations.

2 THE WEIGHT-SHARING APPROACH TO ARCHITECTURE SEARCH

In this section, we formalize the hyperparameter optimization problem solved by weight-sharing approaches as a bi-level optimization problem over a structured hypothesis space. We consider a structured hypothesis space $H(\mathcal{W}, \mathcal{C}) = \{h_w^{(c)} : w \in \mathcal{W}, c \in \mathcal{C}\}$ where $\mathcal{C}$ is a discrete set of configurations, with each c corresponding to an induced hypothesis subclass $H_c = \{h_w^{(c)} : w \in \mathcal{W}\}$ of functions $h_w^{(c)} : \mathcal{X} \mapsto \mathcal{Y}'$, parameterized by $\mathcal{W}$, mapping from some input space $\mathcal{X}$ to output space $\mathcal{Y}'$. For example, in neural architecture search $\mathcal{C}$ is the set of all possible

[*]Equal contribution [1]School of Computer Science, Carnegie Mellon University [2]Determined AI. Correspondence to: Mikhail Khodak <khodak@cmu.edu>, Liam Li <me@liamcli.com>.

Proceedings of the 2^{nd} SysML Conference, Palo Alto, CA, USA, 2019.

architectures in the search space and $\mathcal{W}$ is a subset of $\mathbb{R}^d$, where d is the number of weights needed to parameterize the largest architecture in $\mathcal{C}$. Note we can treat any search space as discrete by taking a finite random sample from the search space, with the guarantee that we can find a good configuration as long as the sample is large enough.

As usual, the goal of learning is to find $h_w^{(c)} \in H(\mathcal{W}, \mathcal{C})$ with low population error $\ell_{\mathcal{D}}(h_w^{(c)}) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\ell(h_w^{(c)}(x), y)$ for loss $\ell : \mathcal{Y}' \times \mathcal{Y} \mapsto \mathbb{R}$ and some distribution $\mathcal{D}$ over sample space $\mathcal{X} \times \mathcal{Y}$. One reasonable approach is to solve the following bi-level optimization problem given training and validation sets $T, V \subset \mathcal{X} \times \mathcal{Y}$:

$$\begin{aligned} &\arg\min_{c\in\mathcal{C}} \quad \ell_V(w, c) \\ w \quad &\in \quad \arg\min_{w'\in\mathcal{W}} \quad \mathcal{L}_T(w', c) \end{aligned}$$

Here $\ell_S(w, c) = \frac{1}{|S|}\sum_{(x,y)\in S} \ell(h_w^{(c)}(x), y)$ is the empirical risk over S for any finite set $S \subset \mathcal{X} \times \mathcal{Y}$ and the regularized empirical risk $\mathcal{L}_T(w, c) = \ell_T(w, c) + R(w)$ for some regularization function $R : \mathcal{W} \mapsto \mathbb{R}$.

Weight-sharing (Pham et al., 2018) is an optimization heuristic for solving this problem in which a shared solution $w \in \mathcal{W}$ to the inner problem is found for a relaxation of the discrete search space (e.g. a parameterized distribution over configurations) and subsequently used for the outer problem to evaluate configurations c or update architecture parameters over the relaxed search space. It has been observed that the shared-weights solutions w often provide a strong signal of the quality of architecture c via $\ell_V(w, c)$; in fact, simply selecting the best architecture according to $\ell_V(w, c)$ from a set of uniformly sampled architectures will return a competitive configuration (Li & Talwalkar, 2019a).

For concreteness, we describe weight-sharing for neural architecture search with a stochastic relaxation of the search space, in which we maintain a distribution over $\mathcal{C}$. In this case, at each iteration we sample an architecture from this distribution, sample a mini-batch of examples from the training set T, and update the shared-weights using standard backpropagation on the sampled architecture. We then sample a mini-batch of examples from the validation set V and update the architecture distribution using e.g. a stochastic estimate of the architecture gradient.

This procedure heuristically handles an important question in any hyperparameter optimization algorithm–how to allocate resources to configurations– by giving more weight to favorable configurations as encoded by the distribution. However, unlike traditional hyperparameter optimization methods that may also perform adaptive resource allocation, weight-sharing maintains a single set of weights for the joint optimization problem over weights and configurations, instead of one set of weights *per* configuration. Although this approach has some drawbacks, such as the inability to directly tune learning parameters (e.g. regularization and optimization routine parameters), we show in Section 4 how these limitations can be overcome in the federated learning setting, where maintaining a single shared model is very natural and effective.

3 Feature Map Selection

Here we demonstrate how weight-sharing can be used as a tool to speed up general architecture search problems by applying it to two feature map selection problems.

3.1 Weight-Sharing Algorithm

In the feature map selection problem we have a small set of configurations $\mathcal{C} = \{\phi_i : \mathcal{X} \mapsto \mathbb{R}^n \text{ for } i \in [k]\}$, each corresponding to some feature map of the input that we plan to pass to a linear classifier drawn from $\mathcal{W} = \mathbb{R}^n$; the hypothesis space is then $H(\mathcal{W}, \mathcal{C}) = \{\langle w, \phi_i(\cdot)\rangle : w \in \mathcal{W}, \phi_i \in \mathcal{C}\}$. Examples of feature maps one can consider are random Fourier features with preprocessing and bag-of-n-grams (BonG) featurizations of documents.

We propose the following simple combination of weight-sharing and successive halving for finding the best feature map using training data T and validation data V:

Assign probability $p_i = 1/|\mathcal{C}|$ to each feature map ϕ_i

For $t = 1, \ldots, \log_2 |\mathcal{C}|$ do

To each sample $(x, y) \in T$ assign map ϕ_{i_x} w.p. p_{i_x}

$$w \leftarrow \arg\min_{w'\in\mathbb{R}^d} \quad \lambda\|w'\|_2^2 \quad + \sum_{(x,y)\in T} \ell(\langle w', \phi_{i_x}(x)\rangle, y)$$

For each feature map ϕ_i:

Assign score $s_i \leftarrow \sum_{(x,y)\in V} \ell(\langle w, \phi_i(x)\rangle, y)$

Update probability $p_i \leftarrow 2p_i 1_{s_i \le \text{Median}(\{s_i : i\in[k]\})}$

Return ϕ_i w.p. p_i

Observe the equivalence to probabilistic NAS: at each step the classifier (shared parameter) is updated using random feature maps (architectures) on the training samples. The distribution over them is then updated using estimated validation performance. In addition to the above probabilistic update scheme, which uses successive halving, we also consider an exponentiated gradient (multiplicative weights) approach, which may be viewed as a softer version of successive elimination.

3.2 Empirical Results

The first problem we consider is kernel ridge regression over random Fourier features (Rahimi & Recht, 2008) on CIFAR-10. We also study logistic regression for IMDB

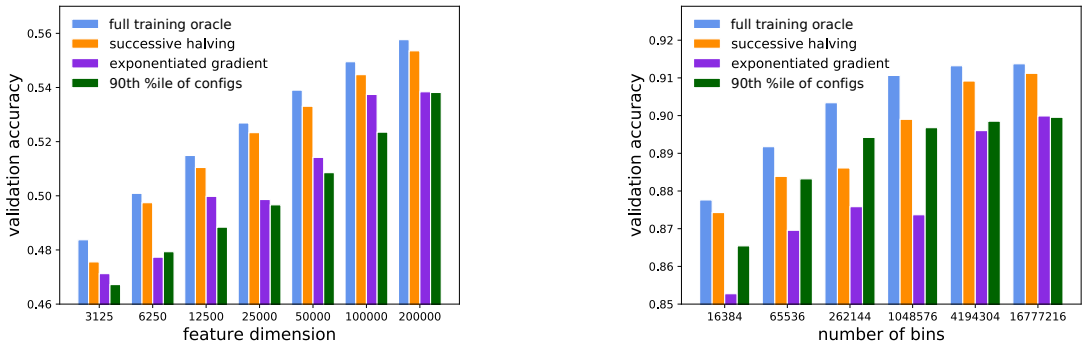

Figure 1. Validation accuracy on CIFAR-10 (left) and IMDB (right) of feature map selection with weight-sharing compared to a full sweep of random configurations. Average over 16 seeds.

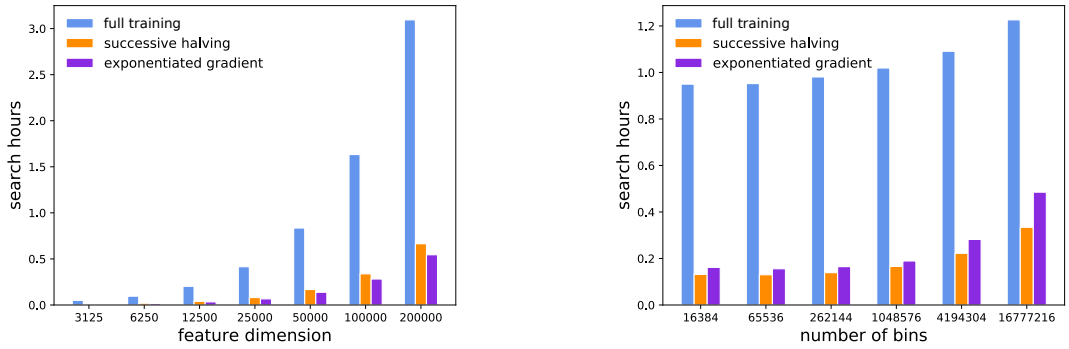

Figure 2. Search time on CIFAR-10 (left) and IMDB (right) for feature map selection. Weight-sharing finds a configuration with almost the same validation accuracy much faster than random search.

sentiment analysis of Bag-of-n-Gram (BonG) featurizations, a standard NLP baseline (Wang & Manning, 2012). See Appendix A for more details about the two search spaces considered.

To test the performance of weight-sharing for feature map selection, we randomly sample 64 configurations each for CIFAR-10 and IMDB and examine whether the above schemes converge to the optimal choice. The main comparison method here is thus random search, which runs a full sweep over these samples; by contrast successive halving will need to solve $6 = \log_2 64$ regression problems, while for exponentiated gradient we perform early stopping after five iterations. Note that weight-sharing can do *no better* than random search in terms of accuracy because they are picking a configuration from a space that random search sweeps over. The goal is to see if it consistently returns a good configuration much faster. As our results in Figures 1 and 2 show, successive halving indeed does almost as well as random search in much less time.[1] While exponentiated gradient usually does not recover a near-optimal solution, it does on average return a configuration in the top 10%. We also note the strong benefit of over-parameterization for IMDB–the n-gram vocabulary has size 4 million so the number of bins on the right is much larger than needed to learn in a single-configuration setting. Overall, these experiments show that weight-sharing can also be used as a fast way to obtain signal in regular learning algorithm configuration and not just NAS.

[1] We expect the speedups to be higher with more configurations and will validate this with future experiments.

4 Federated Hyperparameter Optimization

In this section we argue for the viability of using weight-sharing for hyperparameter optimization in the federated learning setting. We are motivated by the observation that hyperparameter tuning is a major challenge for federated learning (Kairouz et al., 2019). In particular, because of the ephemeral nature of multi-device learning, in which the data is bound to the device and the number of training rounds is limited due to computation and communication constraints, running standard cross-validation and more sophisticated hy-

perparameter optimization approaches necessarily involves making use of subsets of devices to train individual configurations. Hence, with existing methods, we cannot update all configurations using data from all devices, which can be problematic when data is non-i.i.d across devices.

The promise of weight-sharing in this setting is that, no matter the final model, the data from any given device will have contributed in some part to the final parameterization. Furthermore, by treating each device as a data-point to be allocated to some configuration in order to make a local update, weight-sharing naturally aligns with the dominant method in federated learning, Federated Averaging (FedAvg), in which at each communication round multiple devices run local SGD from a shared initialization before averaging the output (McMahan et al., 2017). Finally, by formalizing the problem as one of meta-learning, we are able to get around the issue of not being able to tune non-architectural parameters such as learning rates and regularization coefficients, as described below.

4.1 Extending FedAvg with Hyperparameter Optimization

We consider federated learning in the non-i.i.d. device setting with personalization, also known as meta-learning. Note that here the sample space $\mathcal{X} \times \mathcal{Y} = \{(\tilde{T}, \tilde{V}) : \tilde{T}, \tilde{V} \subset \tilde{\mathcal{X}} \times \tilde{\mathcal{Y}}, |\tilde{T}| < \infty, |\tilde{V}| < \infty\}$ consists of tasks/devices represented by finite training and validation set pairs $\tilde{T}, \tilde{V}$ over data space $\tilde{\mathcal{X}} \times \tilde{\mathcal{Y}}$. In the most general setting the configuration space $\mathcal{C}$ consists of both parameters that configure the on-device (within-task) update algorithm (e.g. learning rate, weight-decay, dropout, etc.) and potentially the model architecture, but we will focus only on configuring the within-task algorithm. The shared-weight parameter space determines the initialization of the within-task update algorithm configured by a configuration c. Thus we have $H(\mathcal{W}, \mathcal{C}) = \{h_w^{(c)} : w \in \mathcal{W}, c \in \mathcal{C}\}$ s.t. $h_w^{(c)} : \mathcal{X} \mapsto \mathcal{Y}'$, where $\mathcal{Y}' = \{\tilde{h}_{\tilde{w}} : \tilde{w} \in \mathcal{W}\}$ consists of predictors $\tilde{h}_{\tilde{w}} : \tilde{\mathcal{X}} \mapsto \tilde{\mathcal{Y}}$ on the underlying data space. The target loss function to minimize is then $\ell_{\mathcal{D}}(h_w^{(c)}) = \mathbb{E}_{(\tilde{T},\tilde{V})\sim\mathcal{D}} \frac{1}{|\tilde{V}|} \sum_{(x,y)\in\tilde{V}} \tilde{\ell}(h_w^{(c)}(\tilde{T})(x), y)$, i.e. the validation loss of a model $h_w^{(c)}(\tilde{T}) \in \mathcal{Y}'$ trained using initialization w and algorithm configuration c on dataset $\tilde{T}$, with the validation and training data drawn as pairs from some distribution $\mathcal{D}$. Here $\tilde{\ell} : \tilde{Y} \times \tilde{Y} \mapsto \mathbb{R}$ is a loss function on the underlying data-space. Note that, in the standard meta-learning setting, $\mathcal{D}$ is a meta-distribution over task-distributions, so sampling $\tilde{T}, \tilde{V}$ consists of drawing such a distribution and then sampling from it i.i.d.

We propose the following simple approach using multiplicative-weights on top of random search:

Randomly sample a shared initialization $w \in \mathcal{W}$
Randomly sample k configurations $c_i \in \mathcal{C}$ and assign to each an initial probability $p_i = 1/|\mathcal{C}|$
For communication round $t = 1, \ldots, n$ do
  For each device $j \in [b]$ in round t do
    Get on-device data $\{(\tilde{T}_{tj}, \tilde{V}_{tj})\}$
    Get configuration $c_{i_{tj}}$ w.p. $p_{i_{tj}}$
    Obtain predictor $\tilde{h}_{\tilde{w}_{tj}} = h_w^{(c_{i_{tj}})}(\tilde{T}_{tj})$ by training with configuration $c_{i_{tj}}$ on dataset $\tilde{T}_{tj}$
    Score predictor using on-device validation data:

$$s_{tj} = \frac{1}{|\tilde{V}_{tj}|} \sum_{(x,y)\in\tilde{V}_{tj}} \tilde{\ell}(\tilde{h}_{\tilde{w}_{tj}}(x), y)$$

  Update shared-weights: $w \leftarrow \frac{1}{b} \sum_{j\in[b]} \tilde{w}_{t_j}$
  Multiplicative weights update: for each $j \in [b]$ do
    $p_{i_{tj}} \leftarrow p_{i_{tj}} \exp\left(-s_{tj}/\sqrt{n}\right)$
  Re-normalize probabilities: $p_i \leftarrow p_i / \sum_{i\in[k]} p_i$

Note that if the configurations consist of just one variant of SGD we recover the basic FedAvg algorithm (McMahan et al., 2017). Thus in each round we send model initializations and some random algorithm configuration to each device in a batch; each device updates the model using this algorithm over on-device training data and the server then aggregates these updates; finally, each device updates the distribution over algorithm configurations using multiplicative weights over on-device validation data. The advantage of this approach is that by treating the on-device algorithm as a model, we transform most of the learning hyperparameters of FedAvg (in-particular the on-device learning rate) into architectural hyperparameters.

To evaluate this method, we propose using the LEAF dataset of federated learning benchmarks (Caldas et al., 2018), in particular the common language-modeling datasets such as Shakespeare. While recurrent neural network architectures such as LSTMs (Hochreiter & Schmidhuber, 1997) have seen heavy development for such tasks, they still require a great deal of learning-parameter tuning, specifically of parameters such as learning rate, weight-decay, dropout, and so on. We thus propose to focus on non-architectural parameters and compare the performance of the shared-weights tuned hyperparameters against human-tuning approaches (McMahan et al., 2017).

REFERENCES

Arora, S., Liang, Y., and Ma, T. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Caldas, S., Wu, P., Li, T., Konečn, J., McMahan, H. B., Smith, V., and Talwalkar, A. LEAF: A benchmark for federated settings. arXiv, 2018.

Chen, X., Xie, L., Wu, J., and Tian, Q. Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation. *arXiv e-prints*, art. arXiv:1904.12760, Apr 2019.

Devries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. 2017.

Falkner, S., Klein, A., and Hutter, F. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. 2019.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. Population based training of neural networks. 2017.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečn, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. Advances and open problems in federated learning. arXiv, 2019.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. Multi-fidelity bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, 2017.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast bayesian optimization of machine learning hyperparameters on large datasets. *International Conference on Artificial Intelligence and Statistics*, 2017.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2019a.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, 2019b.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. *International Conference on Learning Representation*, 17, 2017.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL http://jmlr.org/papers/v18/16-558.html.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. DARTS+: Improved Differentiable Architecture Search with Early Stopping. *arXiv e-prints*, art. arXiv:1909.06035, Sep 2019.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Loper, E. and Bird, S. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*, 2002.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artifical Intelligence and Statistics*, 2017.

Nayman, N., Noy, A., Ridnik, T., Friedman, I., Jin, R., and Zelnik-Manor, L. Xnas: Neural architecture search with expert advice. 2019.

Noy, A., Nayman, N., Ridnik, T., Zamir, N., Doveh, S., Friedman, I., Giryes, R., and Zelnik-Manor, L. ASAP: Architecture Search, Anneal and Prune. *arXiv e-prints*, art. arXiv:1904.04123, Apr 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine

learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2008.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Wang, S. and Manning, C. D. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2012.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

Wu, J., Toscano-Palmerin, S., Frazier, P. I., and Wilson, A. G. Practical multi-fidelity Bayesian optimization for hyperparameter tuning. In *Uncertainty in Artificial Intelligence*, 2019.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019.

Zela, A., Siems, J., and Hutter, F. {NAS}-{bench}-1{shot}1: {BENCHMARKING} {and} {dissecting} {one}-{shot} {neural} {architecture} {search}. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJx9ngStPH.

A FEATURE MAP SELECTION DETAILS

Solvers provided by `scikit-learn` (Pedregosa et al., 2011) were used for ridge regression and logistic regression. For CIFAR-10 we use the kernel configuration setting from Li et al. (2018) but replacing the regularization parameter by the option to use the Laplace kernel instead of Gaussian. The regularization was fixed to $\lambda = \frac{1}{2}$ and the data split was the standard 40K/10K/10K.

For IMDB we consider the following configuration choices:

1. tokenizer: {remove punctuation | split on punctuation | `nltk.word_tokenize`} (Loper & Bird, 2002)
2. stopwords: {remove stopwords | do not remove stopwords}
3. lowercasing: {lowercase | do not lowercase}
4. n-gram order (n includes all k-grams for $1 \leq k \leq n$): $n \in [3]$
5. binarizing: {binarize features | do not binarize features}
6. feature-weights: {naive-Bayes | smoothed inverse frequency} (Wang & Manning, 2012; Arora et al., 2017)
7. feature-weight smoothing parameter α: $\log_{10} \alpha \in [-5, 2]$
8. preprocessing: {None | ℓ_2-normalization | averaging by number of tokens}

The regularization was fixed to $C = 1$ and the data split was 25K/12.5K/12.5K