Can random choices alleviate the burden of hyper-parameter optimization and model selection for neural networks?

S. Vishwak `cs15btech11043`
Harsh Agarwal `cs15btech11019`

1 Introduction

A critical step in hyper-parameter optimization and model selection in neural networks is to make a set of choices. These choices could be the learning rate, number of layers in a neural network, number of nodes in each layer of the neural network, weight decay parameters and so on. If we were to denote these set of choices by Λ, then the goal of hyper-parameter optimization and model selection is to find the ideal $\lambda^* \in \Lambda$ such that given a hyper-parameter objective Ψ (which could be cross-validation loss),

$$\lambda^* = \underset{\lambda \in \Lambda}{\operatorname{argmin}}\, \Psi(\lambda) \tag{1}$$

Generally, if provided with K such configurable parameters, then Λ would be a set of K-tuples. Let the set of "valid" choices for each of the K configurable (hyper)parameters be $C_i, \forall i \in \{1, \ldots, K\}$. Now, the space of valid parameter combinations is $\prod\limits_{i=1}^{K} C_i$, which is affected by the curse of dimensionality.

There have been multiple attempts at improving hyperparameter optimization over the years. The most commonly used method is **grid search**, which is embarrasingly parallel, and hence can be done "efficiently". However, the task of determining the sets $\{C_i\}_{i=1}^{K}$ remains, for which **manual search** is used, where one can identify regions in Λ that are more productive and promising, and these regions can be used to build the sets $\{C_i\}_{i=1}^{K}$. Recently Bergstra and Bengio (1) have provided an algorithm for hyper-parameter optimization called **random search**, which suggests that a search based entirely on random events is capable of finding, over the same domain Λ, hyper-parameters and models which are just as good or even better than the ones found by grid search, with a fraction of the computation time and consequently energy.

2 Proposed Methodology and Contributions

In this project, we expect on contributions to be two-fold:

- *Con. 1* Build a framework for random search proposed by Bergstra and Bengio (1) for `PyTorch` for optimizing hyper-parameters
- *Con. 2* Check if randomly built models are collectively able to perform better as an ensemble than those models designed with extremely careful hyper-parameter and model choices.

Con. 1 has not been attempted before. In addition to considering canonical hyper-parameter choices made in general deep learning tasks such as learning rate, weight decay parameters, architecture specifications (to mention a few), we will also make choices over the weight initialization schemes (e.g. by Glorot and Bengio (3), He et al. (4)), optimization algorithms (e.g. AdaGrad(Duchi et al. (2)), Adam(Kingma and Ba (6)), RMSProp(Tieleman and Hinton (9))), pre-processing techniques and so on. The idea is to be able to construct an API / wrapper for `PyTorch` to do this automatically.

Con. 2 is a spin-off idea. The intuition behind this arises from Boosting, where a group of weak-classifiers can become a strong classifier. So we would like to tests the hypothesis that poor hyper-parameter optimized models when combined could be "better" than a well-studied model.

3 Experiments to be conducted and Datasets to be used

Our experiments will involve multi-layer perceptrons (at least) with Batch Normalization(Ioffe and Szegedy (5)) and Dropout(Srivastava et al. (8)), of arbitrarily chosen sizes. Frameworks to be used are `PyTorch` and `scikit-learn`. `PyTorch` will be used for *Con. 1* and `scikit-learn` will be used for *Con. 2* where we have stable APIs for boosting and running multi-layer perceptrons.

Our experiments will involve the usage of 3 datasets - MNIST[1], CIFAR10[2] and SVHN[3]. Since CIFAR10 and SVHN are extremely high-dimensional when flattened out, we will using feature extracted versions of those datasets obtained from Wide-Resnets. If time permits, we are also interested in obtaining results from randomly generated datasets as well. We also will be using a series of datasets with many factors of variation[4] which are used in the experiments conducted by Larochelle et al. (7) and Bergstra and Bengio (1).

4 Expected Results and Performance Metrics

For *Con. 1*, we would like to verify the claim put forward by Bergstra and Bengio (1) that random search is indeed faster than the ordinary grid search. For this, we will be noting the number of trials, run-time and system specifications and the final results obtained -training and testing loss and/or training and testing accuracy, and comparing the effectiveness of these two search methods. For *Con. 2*, we will be comparing results with existing state-of-the-art results using just multi-layer perceptrons accuracy-wise.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 2012.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical report, EECS Department, University of California, Berkeley, 2010.

[3] 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*, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, 2015.

[6] Durk Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[7] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 2014.

[9] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[1] `http://yann.lecun.com/exdb/mnist/`

[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

[3] `http://ufldl.stanford.edu/housenumbers/`

[4] Datasets available here: `https://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007`