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

Vishwak Srinivasan
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 $\prod_{i=1}^{K} C_i$, 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 (2012) 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 Previous Work and Our Contributions

2.1 Previous Work

In addition to **grid / manual search** and **random search** specified above, there have been multiple efforts aimed at speeding up hyper-parameter optimization. Snoek et al. (2015) present a

Bayesian approach to hyper-parameter with certain theoretical guarantees, and evaluate their method for model selection for object recognition tasks. Maclaurin et al. (2015) compute hyper-parameter gradients by exactly reversing the dynamics of stochastic gradient descent with momentum. A bandit approach towards hyper-parameter optimization was also proposed in Li et al. (2016).

The idea of having an ensemble of neural networks dates back to 1990 stemming from the work of Hansen and Salamon (1990). A few years later, work by Krogh and Vedelsby (1995) also suggested the usage of neural network ensemble for improved generalization. However, a major drawback of these prior studies focus on improving the generalization performance, while few of them address the cost of training ensembles.

Schwenk and Bengio (2000) performed another empirical study where they propose three methods to couple AdaBoost and neural classifiers. The authors state in their paper that one method gave good generalization, but however required more training epochs and the second-order batch method such as conjugate gradients.

A very recent study conducted by Ju et al. (2017) showed that a variant called the "Super-Learner" gave a performance boost among many other variants in their work, with an accuracy bump of 1% on the CIFAR10 dataset.

2.2 Our Contributions

As the title may suggest, the goal of the project was to develop / build random neural network models and see if an ensemble of these "random" models would perform just as well or better than those models found after an extensive hyper-parameter search by either one of the methods specified above minus the time.

Running grid search for other model hyper-parameters for a neural network with 3 hidden layers took a lot of time, meaning a direct comparison would not be possible. As a solution to this issue, our first contribution was the implementation of **random search** over the set of model parameters.

Now that a baseline has been achieved, the next contribution of the work is the implementation of an ensemble of neural networks. The ensemble built was a simple weighted hard-soft voting classifier. Using this, we will be checking the hypothesis posed empirically.

3 Contribution 1: Random Search over model Hyper-parameters

The model hyper-parameters considered for performing a search over are listed below:

- Model Shape and Shape Parameters: We assume shapes for a given neural network: `decreasing`, `convex`, `concave` and `constant`. Associated with these "shapes" are two parameters. Examples of neural networks constructed with these shapes are shown in Figure 1.

- Number of Hidden layers: Number of hidden layers between the input and output layers

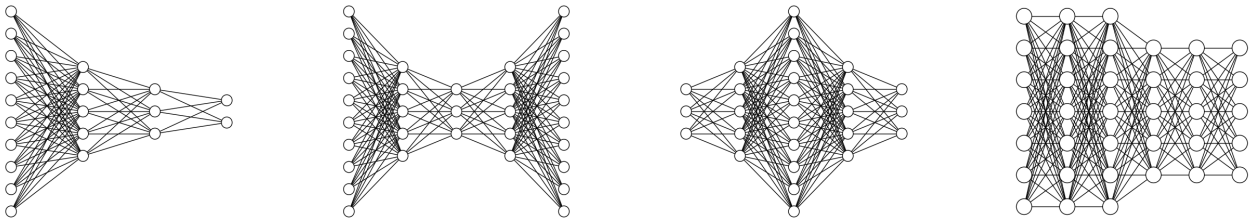

Figure 1: (L to R): `decreasing`, `convex`, `concave` and `constant` shapes respectively

- Dropout probabilities: Layer-wise dropout probability, where `Dropout(p)`, denotes that Dropout with probability p (Srivastava et al. (2014)).
- Existence of batch normalization: Layer-wise boolean vector representing the existence of batch normalization for that layer. Note that since we are using MLPs, a 1-D version of the Batch Normalization(Ioffe and Szegedy (2015)) is used.
- Weight Initialization technique: Considering the fact that weight initialization plays a crucial role in the performance of the neural network, we have decided to include this in the set of model hyper-parameters. The implemented initialization schemes are Glorot's initializatonGlorot and Bengio (2010), Kaiming He's initializationHe et al. (2015), a $\mathcal{U}[0, 1]$ initialization and the default initialization i.e., $\sqrt{\frac{1}{\text{fan-out}}}$
- Activation functions: One-activation-per-network is used, to reduce the dimensionality of the search space. The activations are `ReLU`, `sigmoid` and `tanh`.

Initially, we attempted at running grid search over a discretized search space (where the continuous variables were discretized) resulting in ≈ 7000 grid points with the negative log likelihood loss over the validation set as a scoring function. The neural networks built on this grid were made to train for 3 epochs on every dataset. Despite using 3 GPUs[1] for 2 days, the experiments failed to complete, causing us to time-out the experiments in the interest of freeing resources. The power during this period was 80W per GPU (on average), thus equating to 240W overall (on average)[2].

Going with the random search algorithm, we consider 1000 samples drawn with replacement while maintaining the same scoring function as specified above. This took relatively less time to finish ($\approx$ 7 hrs) on 1 GPU. After obtaining these results, we finetuned the architecture to get our best results. Below tabulated are those results. These results are averaged over 5 runs. `BN` indicates a Batch Normalization layer, and `DP(p=p)` indicates a Dropout layer with probability p.

Dataset	**Accuracy**	**Architecture**
MNIST	$98.294 \pm 0.112\%$	784 - 1097 - `ReLU` - 329 - `BN` - `ReLU` - 98 - `BN` - `ReLU` - 10 - `Softmax`
CIFAR10	$95.018 \pm 0.021\%$	256 - 153 - `ReLU` - 99 - `ReLU` - `DP(p=0.25)` - 64 - `ReLU` - `DP(p=0.5)` - 10 - `Softmax`
SVHN	$96.262 \pm 0.027\%$	192 - 96 - `ReLU` - `DP(p=0.25)` - 124 - `ReLU` - `DP(p=0.25)` - 161 - `ReLU` - `DP(p=0.5)` - 10 - `Softmax`

All these networks were initialized using Glorot's initialization technique.

[1] NVIDIA 1080Ti

[2] Power values where obtained using `nvidia-smi`

4 Contribution 2 - Ensembles

The main motivation for the usage of ensembles is mostly due to the time it takes for performing a search. Computation costs still remain the bottleneck of such methods, and hence why not train multiple random architectures and treat them as an ensemble instead of wasting valuable compute time, albeit some additional memory expenditure?

Our random networks are experimented with ensembles of size 3, 4 and 5 i.e., the number of random neural networks. These random neural networks are generated by randomizing all the hyper-parameters stated above. The ensemble under consideration is a simple voting ensemble. This voting ensemble allows for parallelization of training of candidate networks. The voting ensemble also takes in a set of weights representing the amount of confidence of judgement of each neural network. These weights are assigned based of network features:

- Uniform
- Number of hidden layers
- Number of Dropout layers
- Number of Batch Normalization layers

Additionally, the ensemble can also choose to take a hard decision or a soft decision. A soft decision is based on the weighted average of class probabilities, and a hard decision is based on the maximum weight for a class.

We evaluated this ensemble on the same datasets above, and the results are tabulated below. These results were averaged over 5 trials.

Dataset	**# networks**	**Voting type**	**Weighing scheme**	**Accuracy**
MNIST	3	Soft	Number of Hidden Layers	97.74%
CIFAR10	3	Hard	Uniform	95.04%
SVHN	5	Soft	Number of Hidden Layers	96.24%

The results obtained are not significantly different from the results obtained using Random Search. This could be attributed to the nature of the ensemble being a voting-based ensemble. The major takeaway from this result is the time taken to obtain these results. Note that the results obtained using Random Search took about 7 hrs, whereas these results took a maximum of 30 minutes (without parallelization). This fact is motivating, which means that if we spent more time performing a partial random search over the set of hyper-parameters and left the rest to an ensemble, we should till be able to get close results as compared to those obtained using complete Random Search.

5 Scope for Improvement

MNIST, CIFAR10 and SVHN are considered to be "solved" datasets. It would be interesting to try this approach on different datasets, for instance, Tiny-ImageNet. The ensemble method could also be changed to AdaBoost or any batch-agnostic ensemble methods, which might resolve some problems that could have occurred with the voting ensemble. An extension to Convolutional Neural Networks can also be considered, which might involve shape checking and reshaping the outputs of the networks.

References

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

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.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1990.

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

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

C. Ju, A. Bibaut, and M. J. van der Laan. The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification. *ArXiv e-prints*, 2017.

Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, 1995.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*, 2016.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.

Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural Comput.*, 2000.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.

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.