Revisiting Random Search for Neural Network Hyperparameter Optimization

Ben Rabe, brabe2@illinois.edu

Abstract. The goal of most learning algorithms is to train some set of parameters θ over a set of training data $X^{(\text{train})}$. However, many of these learning algorithms have associated quantities that are not learned during the course of training, but rather are fixed at the beginning of the training process. These quantities are called hyperparameters (denote them λ). In neural networks, these hyperparameters include the number of hidden units, the regularization penalty, and the method with which the weights are initialized. In this way, we can denote the result of training with hyperparameter choice λ as an output function $f = \mathcal{A}_\lambda(X^{(\text{train})})$. For neural networks, it is common practice to use stochastic gradient descent (SGD) as the learning algorithm $\mathcal{A}$. However, the methods for finding optimal hyperparameters are not as straightforward. This paper will focus on this outer-loop optimization problem by describing a few existing methods, considering the advantages of each, and reproducing the results for the random search method introduced by Bergstra and Bengio [2].

1 Problem Characterization

The problem of hyperparameter optimization has the aim of minimizing the generalization error, $\mathbb{E}_{X\sim\mathcal{G}_X}[L(X;\mathcal{A}_\lambda(X^{(\text{train})}))]$, over the hyperparameter space Λ. This can be expressed as follows:

$$\lambda^* = \arg\min_{\lambda\in\Lambda} \mathbb{E}_{X\sim\mathcal{G}_X}[L(X;\mathcal{A}_\lambda(X^{(\text{train})}))] \tag{1}$$

An important thing to note is that for any problem in practice, there is no natural distribution but rather a finite set of samples taken from it to construct a training, validation, and test set. We should still consider the existence of a distribution $\mathcal{G}_X$ however, since in practice any trained classifier may be used on new data not included in the datasets gathered for training and evaluation. This raises the question of how to compute the expectation. The standard technique is *cross-validation* [4], which is approximating the expectation with the mean over a validation set $X^{(\text{valid})}$ that is drawn i.i.d. from $\mathcal{G}_X$. As long as the validation set is not seen by the learning algorithm during training (and under the i.i.d. assumption), cross-validation is unbiased. With this in mind, the optimization problem becomes:

$$\begin{aligned}
\lambda^* &\approx \arg\min_{\lambda\in\Lambda} \frac{1}{|X^{(\text{valid})}|} \sum_{X\in X^{(\text{valid})}} L(X;\mathcal{A}_\lambda(X^{(\text{train})})) && (2)\\
&\equiv \arg\min_{\lambda\in\Lambda}\ \Psi(\lambda) && (3)\\
&\approx \arg\min_{\lambda\in\{\lambda^{(1)},\ldots,\lambda^{(S)}\}}\ \Psi(\lambda) && (4)
\end{aligned}$$

We'll refer to Ψ from equations (3,4) as the hyperparameter response function, and in the future use it to refer to its use both validation and test datasets. Note that equation 4 suggests evaluating the response function at S trial points in the hyperparameter space, and simply choosing the one that minimizes the validation error. This approach is reasonable because we know very little about the behavior of the response function Ψ, so it cannot be solved in the same manner as more traditional optimization problems. Thus, the crux of the problem is in choosing the set of trial points $\{\lambda^{(1)}, ..., \lambda^{(S)}\}$, methods for which will be described in the next section.

2 Comparison of Methods

The first and most straightforward method is a manual search ("grad student descent") that is effectively choosing values for hyperparameters sequentially based on the intuition of the researcher gleaned from prior experiments. There is essentially no barrier to entry with this method, although more experienced researchers will tune their model more effectively. This method also gives some sense of intuition into the hyperparameter response function Ψ. On the other hand, it is not reproducible or extensible.

Another common and intuitive method is a grid search, in which a grid of reasonable hyperparameter values is iterated over in brute force. It is an embarrassingly parallel task, making it simple to implement and feasible if large amounts of resources are available. However, it suffers from the curse of dimensionality in the number of hyperparameters to optimize over. If each of the K hyperparameters may take values in a set $L^{(k)}$, then the number of trials needed in total is $S = \prod_{k=1}^{K} |L^{(k)}|$. While this is the case, grid search was still the standard until random search and bayesian models began to be considered [6].

This brings us to random search. With this method, the value of each hyperparameter in each trial is drawn uniformly (or uniformly in log domain) from the same search space as a grid search. One of the most striking features of random search is its performance in high-dimensional hyperparameter spaces, especially those with *low effective dimensionality*. Low effective dimensionality is the property that Ψ is more sensitive to changes in some variables than others. In one dimension, for instance, we can think of this property as $f(x_1, x_2) \approx g(x_1)$, where the function f is more sensitive in changes to x_1 than x_2. As shown in figure 1, the grid search doesn't effectively cover the subspace formed by each dimension as it tests the same point in that dimension many times.

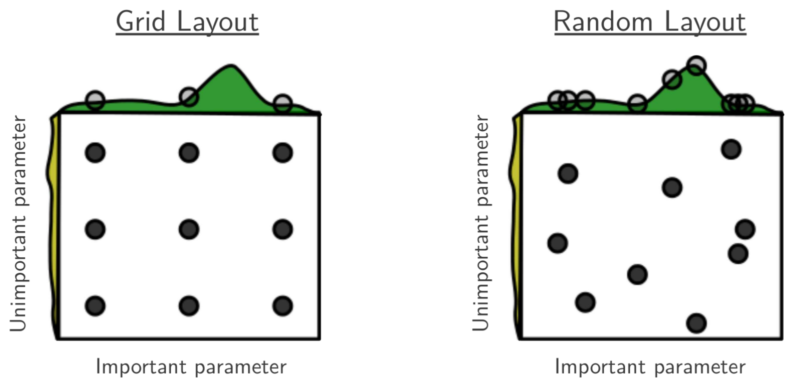


Fig. 1: Illustration of grid search and random search performance in the case of low effective dimensionality. Taken from [2].

3 Interpreting Results of Random Experiments

3.1 Estimating Generalization

The standard for evaluating performance of a model found by cross-validation is to consider the $\lambda^{(s)}$ that minimizes $\Psi^{(\text{valid})}(\lambda^{(s)})$ as optimal, and report $\Psi^{(\text{test})}(\lambda^{(s)})$ However, as discussed in the previous section, the training, validation, and test datasets are finite samples from an assumed natural distribution (extensible by assuming the inference task will be applied in practice on new data). As such, the test error is not monotone with the validation error i.e. the hyperparameter configuration that minimizes the validation error may not minimize the test error. We'd like to take this uncertainty in our choice of the "best" model into account so that we not only report a test error, but have some idea of a variance in that error. Note that this is not helpful in a practical sense; choosing the minimizer on the validation data is still our best chance at minimizing test error. However, this can be useful as an analysis tool as we will discuss. A procedure for considering this variation is outlined in [2]. Consider the following definitions:

$$\Psi^{(\text{valid})}(\lambda) = \frac{1}{|X^{(\text{valid})}|} \sum_{X \in X^{(\text{valid})}} L(X; \mathcal{A}_\lambda(X^{(\text{train})})) \tag{5}$$

$$\Psi^{(\text{test})}(\lambda) = \frac{1}{|X^{(\text{test})}|} \sum_{X \in X^{(\text{test})}} L(X; \mathcal{A}_\lambda(X^{(\text{train})})) \tag{6}$$

$$\mathbb{V}^{(\text{valid})}(\lambda) = \frac{\Psi^{(\text{valid})}\ (1 - \Psi^{(\text{valid})})}{|X^{(\text{valid})}| - 1} \tag{7}$$

$$\mathbb{V}^{(\text{test})}(\lambda) = \frac{\Psi^{(\text{test})}\ (1 - \Psi^{(\text{test})})}{|X^{(\text{test})}| - 1} \tag{8}$$

Note that this variance is only valid because the loss we are considering in this case is the error rate which is 0-1 loss, so Bernoulli variance is correct.

In light of the fact that $X^{(\text{valid})}$ is a finite sample of $\mathcal{G}_X$, we can view the test error of the best model in the set $\{\lambda^{(1)}, ..., \lambda^{(S)}\}$ as a random variable z. We'll model this random variable with a Gaussian mixture model with S mixture components. The mean and variance of each component are simply the mean test error and test variance respectively, or $\mu_s = \Psi^{(\text{test})}(\lambda^{(s)})$ and $\sigma_s^2 = \mathbb{V}^{(\text{test})}(\lambda^{(s)})$.

For the weights of each component, we'd like some estimate on the probability that that component actually was the best model trained according to validation error. To do so via simulation, we'll draw hypothetical validation scores $Z^{(s)}$ from a normal distribution with mean $\Psi^{(\text{valid})}(\lambda^{(s)})$ and variance $\mathbb{V}^{(\text{valid})}(\lambda^{(s)})$, and assign each component a weight w_s equal to the fraction of simulation iterations where that component's score was best. A formal description of the simulation strategy is as follows:

$$w_s = P\left(Z^{(s)} < Z^{(s')}, \forall s' \neq s\right) \text{ where} \tag{9}$$

$$Z^{(s)} \sim \mathcal{N}(\Psi^{(\text{valid})}(\lambda^{(s)}), \mathbb{V}^{(\text{valid})}(\lambda^{(s)})) \tag{10}$$

Finally, bringing this all together gives the statistics of the random variable z representing the performance of the best model as follows:

$$\mu_z = \sum_{s=1}^{S} w_s \mu_s \tag{11}$$

$$\sigma_z^2 = \left(\sum_{s=1}^{S} w_s \left(\mu_s^2 + \sigma_s^2 \right) \right) - \mu_z^2 \tag{12}$$

This technique ends up averaging together the performance of the "best" model (the one with lowest validation error) with other models that were near-optimal, but could have been the best with variation in the dataset.

3.2 Random Experiment Efficiency Curves (REECs)

In the Bergstra paper [2], a plot style is introduced to interpret the results of random experiments. Collecting data for S trials can be interpreted as N independent experiments of s trials each so long as they are evenly divisible. An example of a REEC as well as an explanation of its features can be seen in figure 2 below.

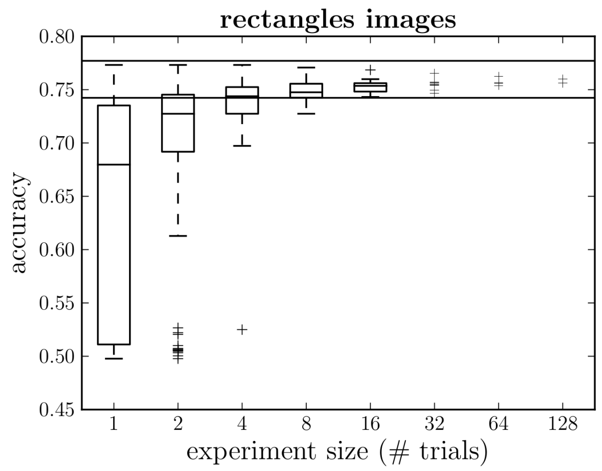


Fig. 2: An example of a random experiment efficiency curve. At each horizontal axis position, the S=256 total trials is reinterpreted as N experiments, each with the size listed on the axis. For instance, at position 128, we see two data points as there are N=2 experiments of size s=128. The box plot covers lower and upper quartiles with a line at the median. Whiskers show 1.5x the nearest inter-quartile range, with data points beyond that plotted as "+". Each data point listed follows from the generalization protocol listed in 3.1. The two black lines mark a 95% confidence interval on the s=256 case. Image taken from [2].

There are several features to note about the plot in figure 2. First, as the number of trials per experiment increases, we see the lower bound on accuracy increase as expected. This is because the poor-performing models get assigned low weights w_s in the GMM model of test error. However, we get a slight decrease in the accuracy of the highest-performing experiments. This is because due to our uncertainty about which trial is best, we're averaging better validation score trials with slightly worse ones as explained in the previous section.

The figures produced for this paper follow the same style, with the exception that the box plot has not been implemented for convenience as well as the final experiment (N = 1) plotted as a single data point instead of ommitted.

4 Experimental Setup

All the code written for the project is hosted at `https://github.com/bendrabe/bergstra12repro`. The training itself is carried out in `{mnist,rot,rectimages}.py`. The GMM estimation of generalization is carried out in `analysis/gmm.py`. TensorFlow [1] was used for generating and training the model. MLPython [5] was used for acquiring datasets. The code was run on an IBM 8335-GTH AC922 server with 4 NVIDIA V100 GPUs. The hyperparameter space sampled from can be seen in the following table:

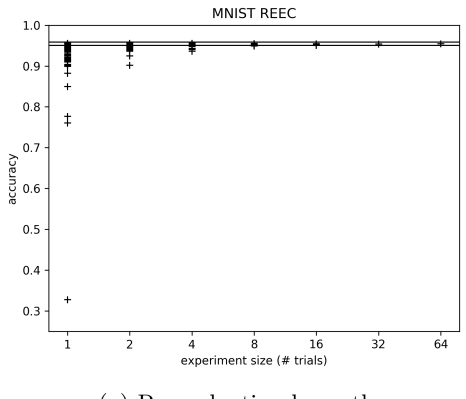
Hyperparameter	Space and sampling
Weight initialization distribution	Uniform on [-1,1] and unit normal
Weight initialization scaling	Multiple of $\sqrt{\text{IN}}$ or $\sqrt{\text{IN} + \text{OUT}}$
Number of hidden units	Drawn geometrically from 18 to 1024
Activation function	Sigmoid or tanh
Learning rate	Drawn exponentially from 1.0e-3 to 10.0
L2 regularization	Drawn exponentially from 3.1e-7 to 3.1e-5
Batch size	[20,100]

IN and OUT correspond to the number of inputs to and outputs from the hidden layer respectively. *Drawn geometrically* corresponds to drawing uniformly in the log domain, exponentiating, then rounding to an integer. *Drawn exponentially* is the same without rounding. If two methods or options are listed, they are chosen with equal probability. See the code itself for more details if desired. Note that the only divergence from the Bergstra paper is the lack of learning rate annealing.

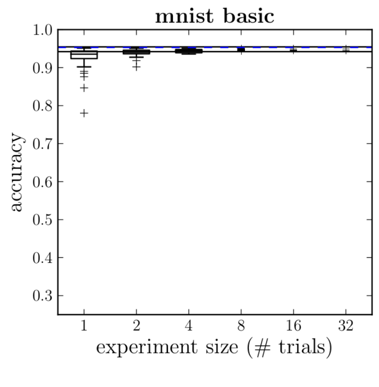
5 Results and Discussion

The three datasets chosen to reproduce from the Bergstra paper [2] were chosen for their widespread familiarity as well as the more interesting statistics that resulted in their analysis. For more details about the datasets, see [6].

In general, we see nearly identical results compared to the Bergstra paper. Basic MNIST (figure 3) reached an accuracy plateau at around 8 random trials. Grid search produced what appears to be a nearly optimal classifier in this case. There was very low variance in the generalization scores and as such, a very tight 95% confidence interval.



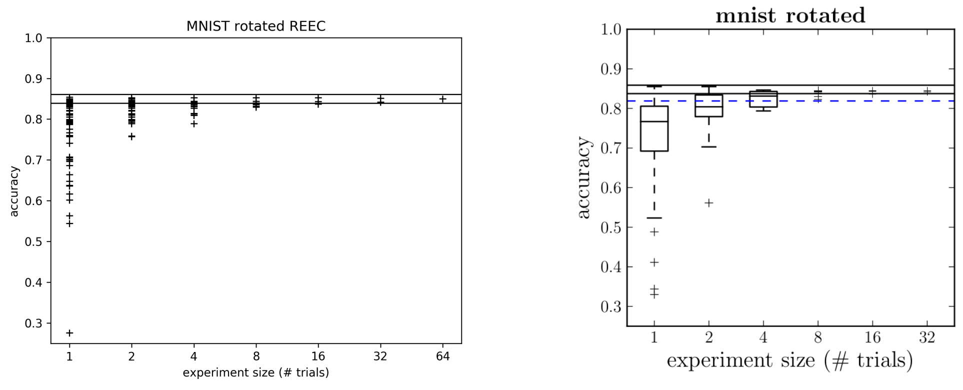
(a) Reproduction by author



(b) Taken from [2]

Fig. 3: Random experiment efficiency curves for MNIST dataset. The dashed blue line is the best model found in a 100 trial grid search.

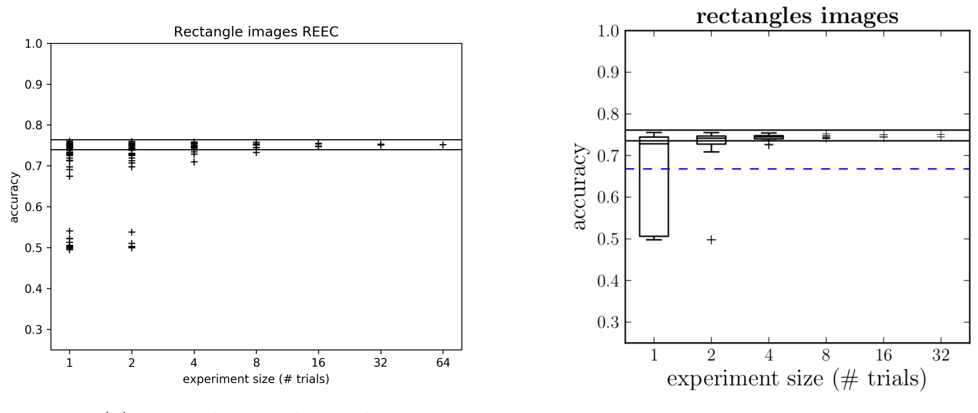
The MNIST rotated dataset is the same as base MNIST with a random rotation applied to each image. Again we see very neat agreement with the Bergstra results, but this time we see that an experiment size of 8 trials is enough to beat the 100 trial grid search accuracy in both the original and reproduction.



(a) Reproduction by author (b) Taken from [2]

Fig. 4: Random experiment efficiency curves for MNIST rotated dataset. Note that in both the reproduction (a) and the original paper (b), 8 random search trials are sufficient to beat the performance of a 100 trial grid search (dashed blue line).

The rectangles images dataset is composed of rectangles of random height, width, and location with an image patched inside and a background patched outside the rectangle. The label is whether the rectangle has larger width or height. As the experiment size increased, we saw very similar results to Bergstra. However, for small experiments (1 or 2 trials), we had outliers that were not present in the original. We attribute these outliers to the lack of learning rate annealing, as the outliers occurred when minibatches were small and learning rates were high. In these cases, gradient descent carried out massive overcorrection with each gradient application. Another possible hyperparameter to explore with this in mind is SGD with momentum (Nesterov or otherwise) [7].



(a) Reproduction by author (b) Taken from [2]

Fig. 5: Random experiment efficiency curves for rectangles images dataset. Note that in both the reproduction (a) and the original paper (b), 4 random search trials are sufficient to beat the performance of a 100 trial grid search (dashed blue line).

6 Future Work

There are two directions for extension of these results. The first is repeating this kind of study on larger, more complex classification problems such as CIFAR-10 or even ImageNet. The training process for a single trial on a modest system (like the 4-GPU server used in this work) would take on the order of 1 hour and 10 hours respectively for CIFAR and ImageNet.

To combat this higher resource demand, researchers have attempted to find methods of choosing promising hyperparameter combinations based on the results of previous trials. An example is sequential model based optimization (SMBO) [3]. For sequential hyperparameter tuning, the expensive step is training the classifier f and computing the validation score. SMBO attempts to identify some surrogate for f that roughly models its response to changing hyperparameters. This surrogate is then minimized to choose the next hyperparameter combination to try. The expensive step must still be carried out, but it will be run with hyperparameters chosen to minimize the surrogate, so a good choice of surrogate function can be very useful.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research **13**(Feb), 281–305 (2012)
3. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems. pp. 2546–2554 (2011)
4. Bishop, C.M., et al.: Neural networks for pattern recognition, pp. 372–375. Oxford university press (1995)
5. Hugo Larochelle and group: MLPython, `https://bitbucket.org/HugoLarochelle/mlpython/src`
6. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y.: An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th international conference on Machine learning. pp. 473–480. ACM (2007)
7. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International conference on machine learning. pp. 1139–1147 (2013)