DYNAMICALLY LEARNING THE LEARNING RATES: ONLINE HYPERPARAMETER OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyperparameter tuning is arguably the most important ingredient for obtaining state of art performance in deep networks. We focus on hyperparameters that are related to the optimization algorithm, e.g. learning rates, which have a large impact on the training speed and the resulting accuracy. Typically, fixed learning rate schedules are employed during training. We propose *Hyperdyn* a dynamic hyperparameter optimization method that selects new learning rates *on the fly* at the end of each epoch. Our explore-exploit framework combines Bayesian optimization (BO) with a rejection strategy, based on a simple probabilistic *wait and watch* test. We obtain state of art accuracy results on CIFAR and Imagenet datasets, but with significantly faster training, when compared with the best manually tuned networks.

1 INTRODUCTION

Hyperparameter tuning is arguably the most important ingredient for obtaining state of art performance in deep neural networks. Currently, most networks are manually tuned after extensive trial and error, and this is derisively referred to as *graduate student descent*. Hyperparameter optimization (HPO), on the other hand, attempts to automate the entire process and remove the need for human intervention. Previous works on HPO propose various strategies to either adaptively select good configurations Snoek et al. (2012) or to speed up configuration evaluations Li et al. (2016).

One drawback behind existing HPO frameworks is that they do not distinguish between different kinds of hyperparameters and treat them all uniformly. Broadly there are two categories of hyperparameters: those that are fixed throughout the training process and those that need to be varied during training. The former class is mostly *structural* (e.g. network depth and width), while the latter is mostly related to the optimization algorithm (e.g. learning rates, regularization). The two classes of hyperparameters have very different behaviors. For structural hyperparameters, online evaluation is not possible, and we need to wait for the entire training process to be completed, which is very expensive. On the other hand, for time-varying hyperparameters, it is possible to select parameters *on the fly* without waiting for training to finish. In this work we keep the structural hyperparameters fixed and focus on optimizing the time-varying hyperparameters to improve efficiency, stability, and accuracy of the training process.

1.1 SUMMARY OF RESULTS

Our main contributions are as follows: (1) We propose *Hyperdyn* an automated approach for dynamically tuning hyperparameters during the training process, based on the past observations. It selects new hyperparameters at the end of each epoch by combining Bayesian optimization (BO) machinery with a simple rejection strategy. It is computationally efficient since it uses Gaussian processes (GP) and simple probabilistic rejection tests. (2) We show state of art performance on image classification benchmarks that match the accuracy of manually tuned networks while significantly improving the training speed. (3) We demonstrate that *Hyperdyn* is able to automatically decide regions of start, acceleration and slow down for the training process, and can also adapt to different conditions such as batch sizes, network architecture, datasets etc.

Although our framework is broadly applicable for any time-varying hyperparameter, we limit ourselves to selecting learning rates in the experiments. We now describe *Hyperdyn* in this context. A set of learning-rate proposals is randomly initialized along with the weights of the neural network. We choose the best learning rate based on the validation accuracy at the end of the first epoch. For

Dataset	Method	85%	90%	95%
CIFAR 10	SGD	1.5x	2x	2x
	ADAM	1x	1x	2x
Imagenet	SGD	1.5x	4x	4x

Table 1: Speed up in training (in terms of no. of iterations) over manually tuned to reach x% of the best reported top-1 validation accuracy

subsequent epochs, we employ standard Bayesian optimization (BO) to obtain new proposals based on the Gaussian process framework Shahriari et al. (2016). However, we do not always accept the outcomes of BO. We design a simple probabilistic *wait and watch* test to decide whether to accept the BO outcome or to stick to the previously chosen learning rate, based on the improvement of the validation accuracy over the past few epochs. This rejection test is very crucial for obtaining good performance. Our experiments show that if we naively switch the learning rate to the BO output at the end of each epoch, we have training instability and bad generalization performance. This rejection framework is philosophically similar to the hyperband framework Li et al. (2016) where more time is spent exploring the more *promising* choices. Here we require a more sophisticated framework that utilizes the temporal history to assess whether the current choice of learning rate is *promising* or if one should switch to a new learning rate, as proposed by BO.

We investigate performance of *Hyperdyn* for tuning the learning rates of two most popular optimization algorithms, *viz.,* stochastic gradient descent (SGD) and Adam Kingma & Ba (2014), on CIFAR-10 and Imagenet datasets. The results are summarized in Table.1. Our method uniformly trains faster and can quickly reach to a significant % of the best validation accuracy, which was previously obtained (He et al. (2015)) after extensive manual tuning. In Section 4, we also show that *Hyperdyn* outperforms other strong baselines such as epoch-based BO (i.e. no rejection) and random 5x, i.e., at every epoch we invest 5x resources more in random search than *Hyperdyn* . Furthermore, we find that our method is stable and trains quickly even under larger batch sizes. We used a batch size of 1000 for our Imagenet experiments, while the manually tuned baseline was on a much smaller batch size of 256. Larger batches are preferred for distributed training since they reduce the relative communication overhead. However, training on larger batches is generally challenging, and can suffer from poor generalization Li (2017). The adaptivity of *Hyperdyn* allows it to overcome this challenge.

We conduct detailed empirical analysis of *Hyperdyn* under a variety of conditions. We find that the learning rates suggested by *Hyperdyn* for SGD eventually decay, but are not always monotonic in the beginning. This agrees with the previous results that using more adaptive algorithms such as Adam is more beneficial in the beginning than at a later stage Wilson et al. (2017). We find that learning rates chosen by *Hyperdyn* generally increase with the batch size, and this rule of thumb has been used for manual tuning Li (2017). We also verify that the learning rates suggested by *Hyperdyn* for Adam eventually converge to values that guarantee theoretical convergence. Further, we observe that SGD tuned with *Hyperdyn* outperforms more sophisticated algorithms (e.g. ones with momentum) that are manually tuned. This suggests the importance of tuning for good learning rates, compared to having more sophisticated optimization algorithms.

1.2 Related Work

Bayesian Optimization has been widely used to optimize blackbox functions. The most common frameworks use Gaussian processes (GP) for efficiently selecting good configurations Shahriari et al. (2016). Recently, Li et al. (2016) introduced Hyperband, which instead focused on speeding up configuration evaluations based on a simple random search strategy. A key feature of Hyperband is that it adaptively allocates resources using a principled early stopping mechanism. In the context of tuning for learning rates, these methods have been previously employed to tune the schedule of learning rate decay, while keeping the initial learning rate fixed (Li et al. (2016), Snoek et al. (2012)). However, this does not provide full flexibility in finding the best learning rates in each epoch. Moreover, these frameworks require for training to be completed in order to carry out their evaluations, which makes them expensive. On the other hand, *Hyperdyn* selects new configurations at the end of each epoch, without requiring for training to finish. Also, most previous works only compare results across different HPO frameworks, but not with the state of art manually tuned networks. Previous works report around 80% validation accuracy for the hyperband algorithm on CIFAR-10 and worse

results for other methods such as SMAC and TPE (tree-based BO) and Spearmint (Gaussian process) Li et al. (2016). None of these previous works report results on Imagenet-like large datasets.

For the task of finding good learning rates, other methods have been proposed that do not rely on hyperparameter optimization. One strategy is to incorporate more adaptivity into the optimization algorithm, as seen in Adam Kingma & Ba (2014), which has better performance in certain cases over SGD, but requires even more hyperparameters to be tuned. Another algorithm that automatically controls the learning rate of SGD, known as SALERA, was introduced in Schoenauer-Sebag et al. (2017). SALERA updates the learning rate by using an exponential moving average and deals with catastrophic events that occur in SGD (that lead to poor training accuracy) by allowing for backtracking. However, the learning rate update rule is fixed unless there is a catastrophic event, so the extent of adaptivity is limited. Additionally, the experiments were only conducted on smaller datasets such as MNIST and CIFAR10, so it is not clear how the algorithm behaves with larger datasets and larger batch sizes. An RL-based approach to learning *on the fly* was introduced in Li & Malik (2016). However, the RL framework in general requires a large amount of training data and the experiments in that work were not conducted on standard benchmark datasets such as CIFAR10 or MNIST. Recently, Goyal et al. (2017), You et al. (2017) proposed methods for large batch training on Imagenet dataset that achieve state of art Top-1% accuracy. The work there, however, was restricted to designing a learning rate scheduler for SGD for large batch sizes. *Hyperdyn* on the other hand is a general framework which can be applied to a wide range of hyperparameter optimization problems and different kinds of training methods.

The idea of using information learned in one task to improve on a future task, or meta–learning (Thrun & Pratt (2012)), is also related to the framework of using past information to improve hyperparameter choice, considered here. LSTMs were employed to learn the gradient descent updates in Andrychowicz et al. (2016); Lv et al. (2017). However, LSTMs require a large amount training data and it is not clear if the methods can scale to standard benchmark datasets. Currently, these techniques have been shown to work for only small scale problems.

The rest of paper is structured as follows – in section 2 we give a brief review of the bayesian optimization algorithm and gaussian processes that are at the center of *Hyperdyn* . In section 3 we describe its in detail. In section 4 we present the details of the experiments.

2 Mathematical Overview

Bayesian optimization has been widely used to find the minimize a black box function. In general, the black box functions are unknown or difficult to compute and the bayesian optimization algorithm suggests query points based on the optimization of an "easier" acquisition function. So if the black-box function was $f(\cdot)$ and we had to minimize $f(\cdot)$. Then we would use the bayesian algorithm as described in Algorithm 1. This version of Bayesian Optimization in Algorithm 1 is a one-step sim-

Algorithm 1 One_Step_BO

1: **input** Search Space: S, Mean Function: $\mu(\cdot)$, Variance Function: $\sigma(\cdot)$
2: select new η_0 by optimizing acquistion function $\alpha(\cdot;\cdot)$

$$\eta_0 = \arg\max_{\eta \in S} \alpha(\eta; \mu(\eta), \sigma(\eta))$$

3: Query objective function to obtain y_0
4: **return** η_0

plification of the one described in Shahriari et al. (2016). We next describe the acquisition function $\alpha(\cdot;\cdot)$ and other statistical details of Algorithm 1.

2.1 Gaussian Processes

A Gaussian Process is a nonparamteric statistical model that is characterized by $\mu_0, \sigma_0, \mathcal{K}(\cdot,\cdot)$ – initial mean, initial variance and kernel function respectively. Consider the sequence of points, $\eta_{1:n}$ (inputs) and $y_{1:n}$ (noisy observations). We introduce auxillary variables $f_{1:n}$, such that $f_{1:n}|\eta_{1:n} \sim \mathcal{N}(\mathbf{m}, \mathcal{K})$ where $\mathbf{m}_i = \mu_0(\eta_i)$, $K_{ij} = \mathcal{K}(\eta_i, \eta_j)$. Then we have that $y_{1:n}|f_{1:n}, \sigma_0^2 \sim \mathcal{N}(f_{1:n}, \sigma_0^2\mathbf{I})$. Given this GP, we "update" $\mu_n(\cdot), \sigma_n(\cdot)$ given some observation $\{(\eta_i, y_i)\}_{i=1}^n$, by Algorithm 2

Algorithm 2 Update_Statistical_Model

1: **input** Kernel Function: $\mathcal{K}$, Data: $\{(\eta_i, y_i)\}_{i=1}^n$, Initial Kernel Parameters: $\mu_0(\cdot), \sigma_0^2$
2: $\mathbf{k}(\eta)_i = \mathcal{K}(\eta, \eta_i), \mathbf{m}_i = \mu_0(\eta_i), K_{ij} = \mathcal{K}(\eta_i, \eta_j)$

$$\mu_n(\eta) = \mu_0(\eta) + \mathbf{k}(\eta)^T (K + \sigma_0^2 \mathbf{I})^{-1} (y_{1:n} - \mathbf{m})$$
$$\sigma_n^2(\eta) = \mathcal{K}(\eta, \eta) + \mathbf{k}(\eta)^T (K + \sigma_0^2 \mathbf{I})^{-1} (y_{1:n} - \mathbf{m})$$

3: **return** $\mu_n(\cdot), \sigma_n(\cdot)$

2.2 Acquisition Functions

A plethora of acquisition functions are discussed in Shahriari et al. (2016), we specifically use the expected improvement function that we describe now. Consider the improvement function $I(\eta, v, \theta) = (v - \tau)\mathbb{I}(v > \tau)$; this captures the amount of improvement over a given threshold τ. Then expected improvement is just $\mathbb{E}[I(\eta, v, \theta)]$ assuming that v is normally distributed with some mean and variance parametrized by η. Then we have

$$\alpha_{\text{EI}}(\eta; \mu_n(\eta), \sigma_n(\eta)) = \mathbb{E}[I(\eta, v, \theta)] = (\mu_n(\eta) - \tau)\Phi\left(\frac{\mu_n(\eta) - \tau}{\sigma_n(\eta)}\right) + \sigma_n(\eta)\phi\left(\frac{\mu_n(\eta) - \tau}{\sigma_n(\eta)}\right) \tag{1}$$

Typically $\tau = \min_n y_n$, *i.e.*, the minimum of noisy observations.

It is not necessary that the kernel function in Algorithm 1 be stationary. In fact there is vast literature on non-stationary kernels for Bayesian Optimization (See Gramacy & Lee (2008)). These methods are, in general, very complicated. In the following section we propose a simple compositional non-stationary kernel that works well in practice.

2.3 The need for time-varying kernel

The simplicity of Bayesian optimization makes it amenable to blackbox optimizations. However, there are a few severe limitations of Bayesian optimization that prevent it from being applied directly to neural network training. First, as stated before, the framework of Bayesian optimization is such that only one setting of learning rate parameter can be used for the entire duration of training. Second, even if we were somehow able to use multiple learning rate parameter suggestions while training, there is no stationarity, *i.e.*, the same learning rates may produce very different results at different points of training.

In *Hyperdyn* we use different temporal estimates of the loss function change and use non stationary kernels to alleviate the two problems mentioned above. By using a simple compositional kernel we avoid the computational burden associated with general non stationary kernels.

Any training algorithm can be roughly summarized as follows

$$w_{t+1} = \mathcal{T}(w_t, \mathcal{B}, \text{Hyperparameters})$$

where $\mathcal{B}$ is a batch of data, $\mathcal{T}$ denotes the training algorithm and w_t are the parameters of some neural network at epoch t of the algorithm. $\mathcal{T}$ is run for some predefined number of iterations over the data. For *Hyperdyn* we define a new set of hyperparameters that includes the epoch number of the algorithm

$$\text{New Hyperparameters} = (\text{True Hyperparameters}, t)$$

Considering epoch number of the training process as a hyperparameter allows to build a composite kernel over t and the true hyperparameters; we now get kernels that are non-stationary without too much overhead from existing stationary ones. Let $(\eta_1, t_1), (\eta_2, t_2)$ be the new hyperparameters where t_i is the epoch number, then our kernel is of the form

$$K((\eta_1, t_1), (\eta_2, t_2)) = K_1(\eta_1, \eta_2) \times K_2(t_1, t_2) \tag{2}$$

Now, the $K_1(\cdot, \cdot)$ kernel can be one of RBF kernel or Matern-x kernels over the true hyperparameters, as in standard literature, we will describe $K_2(\cdot, \cdot)$, now referred to as the time kernel, in the following section. Unlike the true hyperparameters, we do not need to optimize over the epoch number, *i.e.*, t_is behave in a specific way $\{1, 2, \ldots, \}$ upto end of training process. Such a formulation

only helps in introducing non–stationarity in the kernels we use for Bayesian optimization. This is realized by changing the search space for epoch number in the Bayesian optimization to $[t, t]$ for every epoch number t.

2.4 Our Choice of Time Kernel

We use a similar approach to Swersky et al. (2014) Section 3 for training curves. The kernel $K_2(\cdot, \cdot)$ is of the form

$$K_2(t_1, t_2) = \int_{\lambda=0}^{\infty} \exp(-\lambda|t_1 - t_2|)\psi(d\lambda) = \frac{\beta^\alpha}{(|t_1 - t_2| + \beta)^\alpha}$$

where the last equality is obtained by choosing $\psi(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}\exp(-\lambda\beta)$. This construction is motivated from the observation that SGD decays as $O(1/N^{1/2})$ in N iterations, which in our case reduced to $\alpha = 1, \beta = 1/2$, and from work in Swersky et al. (2014).

3 Proposed HPO Algorithm

Hyperdyn is comprised of some crucial moving parts. At the beginning we have no information and employ a purely exploratory approach as described in Algorithm 3. The function Validation_Accuracy$(\cdot, \cdot)$ takes in as input weights (first argument) and hyperparameter values (second argument) and updates weights. The output is top-1 accuracy on the validation set for the updated weights. Function Random_Start gives our main algorithm, which we describe in Algorithm 4, some initial information for a more exploitive approach. In Algorithm 4, Update_Weights simply updates the old weights using the chosen hyperparameter setting. For example, if we are doing SGD then

Algorithm 3 Random_Start

NN weights: w
Input: Hyperparameter Search Space: S, Number of Initial Search Points: k
Initialization $\mathcal{D} = \{\}$, $w \sim$ Normal$(0, 1)$
for $i = 1, 2, \ldots, k$ **do**
  Generate $p_i \in S$ uniformly at random
  $\mathcal{D} = \mathcal{D} \cup \{((p_i, 0), \text{Validation\_Accuracy}(w, p_i))\}$
end for
$k^* = \arg\max_{1 \le i \le k}$ Validation_Accuracy$(w, p_i))$
$w^* =$ Update_Weights(w, p_{k^*})
return $w^*, p_{k^*}, \mathcal{D}$, Validation_Accuracy$(w^*, 0)$

Algorithm 4 *Hyperdyn*

NN weights: w_t, Hyperparameters: η_t
Input: Hyperparameter Search Space: S, Number of Initial Search Points: k, Window: Δ
Kernel parameters: $\mu_0(\cdot), \sigma_0^2$, Kernel Function: $\mathcal{K}$
Initialization: $w_0, \eta_0, \mathcal{D}_0,$ acc $\leftarrow$ Random_Start(S, k)
for $t = 1, 2, \ldots, T$ **do**
  $\mu_t(\cdot), \sigma_t(\cdot) =$ Update_Statistical_Model$(\mathcal{K}, \mathcal{D}_{t-1}, \mu_0(\cdot), \sigma_0^2)$
  $(s_t, t) =$ One_Step_BO$(S \times [t, t], \mu_t(\cdot), \sigma_t(\cdot))$
  acc_diff$_0 \leftarrow$ Validation_Accuracy(w_{t-1}, η_{t-1}) − acc
  acc_diff$_1 \leftarrow$ Validation_Accuracy(w_{t-1}, s_t) − acc
  $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{((\eta_{t-1}, t), \text{acc\_diff}_0), ((s_t, t), \text{acc\_diff}_1)\}$
  if Check_Accuracy_Trend$(w_{1:t-1}, \Delta)$ **then**
    $w_t =$ Update_Weights$(w_{t-1}, \eta_{t-1}), \eta_t = \eta_{t-1}$
  else
    $w_t =$ Update_Weights$(w_{t-1}, s_t), \eta_t = s_t$
  end if
  acc $=$ Validation_Accuracy$(w_t, 0)$
end for

Update_Weights is

$$w_{t+1} \leftarrow w_t - \eta_t \sum_{x \in \mathcal{B}} \nabla f(x)$$

The function Update_Statistical_Model has been described in Section 2.1. We might be tempted to switch to the best performing hyperparameter value at every epoch (Lines $9-14$). As we will show, that such a myopic switching (between hyperparameters) strategy is a hindrance when it comes to good generalization performance. The function Check_Accuracy_Trend, described in Algorithm 5, prevents us from changing our hyperparameters unnecessarily. The idea is that if a hyperparameter choice has been improving on the validation accuracy, then there is no incentive to change.

Algorithm 5 Check_Accuracy_Trend

1: **Input**: $\{w_1, w_2, \ldots, w_\tau\}$, Window $= \Delta$, Temperature $= T$, offset $= b$
2: Generate $u \sim U[0, 1]$
3: $y_l =$ Validation_Accuracy$(w_l, 0) \, \forall \, l \in \{1, 2, \ldots, \tau\}$
4: **if** $\exp\left(T^{-1}(y_\tau - y_{\tau-\Delta}) - b\right) > u$ **then**
5: **return** True
6: **else**
7: **return** False
8: **end if**

An initial offset is provided so that there is a small probability of changing hyperparameter values even when current values are performing well.

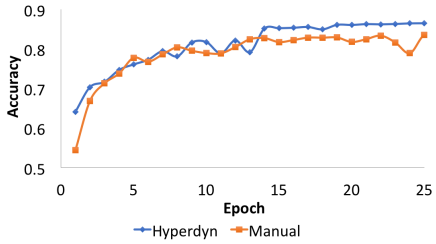
4 EXPERIMENTS

The experiments were conducted on Amazon AWS p2x16 EC2 machines. We only use data parallelism for gradient computation in SGD across different GPUs, *i.e.*, if there are k GPUs and a batch size of α then each GPU calculates the gradient on a batch size of α/k and this is summed across GPUs before weight update. The datasets used were CIFAR10 and Imagenet 1K. We employed simple augmentations as described in He et al. (2015). We also consider their results as the state of art since the structural setting (neural network architecture, augmentation settings etc.) there is closest to the one here. Further for batch sizes not included there we use scaling rules described in Goyal et al. (2017). The number of initial search points for Random_Start was 5. In our experiments on *Hyperdyn* we only use learning rate and momentum for hyperparameter optimization. The window size used is 4, temperature is 1 and offset is 0.01. The window size should not be too small ~ 1 as it leads to very high variance while training, and for the experiments that we run older observations become obsolete fast and a large window size is unnecessary.

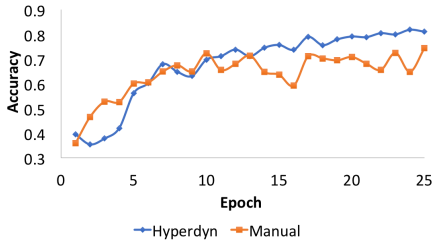
4.1 EXPERIMENTS ON CIFAR10

Comparison to Manually Tuned Networks: In this section we presents results when the weight update mechanism is momentum-SGD. We compare the performance of *Hyperdyn* -tuned momentum-SGD versus the manually tuned momentum-SGD as described in He et al. (2015) on a ResNet-20. Each epoch indicates $(\text{num\_points}/\text{batch\_size})$ number of iterations. As shown in Fig. 1(a)–1(c), *Hyperdyn* tuning is always faster (at reaching superior generalization performance) than the manually tuned one. Table 1 shows the number of iterations after which a certain top-1 accuracy is reached for a ResNet-20 with batch size 128. *Hyperdyn* tuned SGD is substantially faster than the fixed schedule manually tuned SGD. There is no additional tuning required for *Hyperdyn* and as a result we circumvent hours of trial and error to get the hyperparameters right.

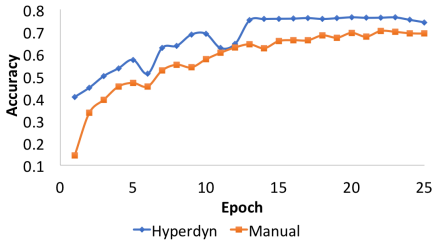
Dynamics of *Hyperdyn* : For any stable recursive stochastic algorithm (for details see Gelfand & Mitter (1991)) it must be true that $\eta_k \to 0$ as $k \to \infty$, where η_k is the learning rate at iteration k. In Fig. 2(a) we observe that *Hyperdyn* tuning automatically tends to 0 as the learning process progresses. Empirically it has been found that larger learning rates are needed for larger batches in the manual setting Li (2017) and *Hyperdyn* is automatically able to achieve this. As can be seen in Fig. 2(a) the average learning rate increases as we increase the batch size from 128 to 2k. However, the same trend does not translate, when the batch size is doubled further. A possible explanation to this maybe that we are already in a very large batch regime $\sim 10\%$ of dataset size and the learning rate dynamics behave very differently. Further, we study *Hyperdyn* in Figure 2(b) for tuning SGD



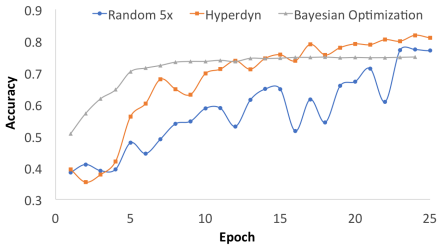
(a) *Hyperdyn* vs Manual (batch size=128)



(b) *Hyperdyn* vs Manual (batch size=1k)



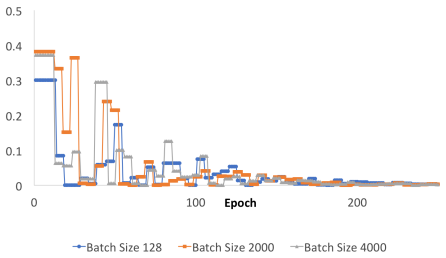
(c) *Hyperdyn* vs Manual (batch size=4k)



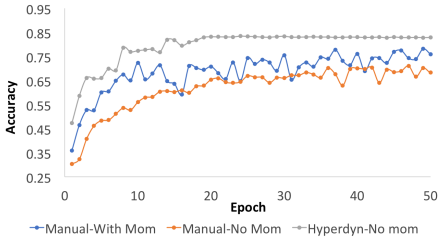
(d) *Hyperdyn* vs Other Algorithms

Figure 1: *Hyperdyn* Performance

when the momentum term is set to 0. We observe that it outperforms manually tuned SGD with momentum, which is a more sophisticated algorithm. This suggests that it is more important to find good learning rates for a simple algorithm, rather than attempt to design a more sophisticated algorithm. Moreover, the *wait and watch* strategy of *Hyperdyn* is incorporating past information for stability, similar in principle to momentum. This could also explain why SGD tuned with *Hyperdyn* can outperform manually tuned SGD with momentum.



(a) Learning Rate variation in *Hyperdyn*

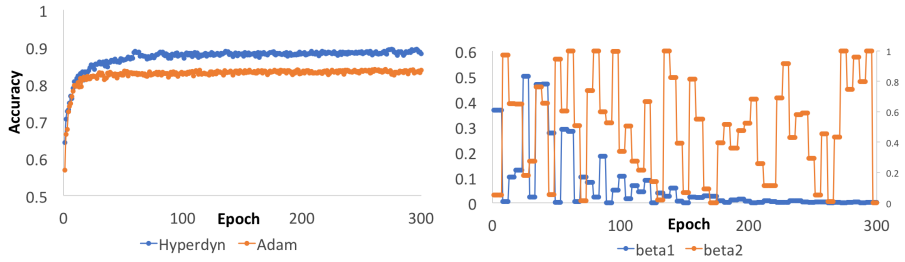


(b) No Momentum *Hyperdyn*

Figure 2: Dynamics of *Hyperdyn*

Comparison of *Hyperdyn* with Other Schemes: We compare *Hyperdyn* to a greedy version of it, where we switch to the best performing hyperparameter in every epoch. This greedy method, or epoch based Bayesian optimization, is obtained by setting the offset *i.e.*, $b = \infty$ in Check_Accuracy_Trend sub-routine; which is nothing but the standard Bayesian optimization method with compositional kernel incorporating the temporal dependence. However, as Fig. 1(d) shows, the epoch based BO has poor generalization. We set the batch size of 1000 on ResNet-20. This observation necessitates the need for a "wait-and-watch" strategy, as realized by the Check_Accuracy_Trend module. Although the epoch based BO version outperforms in the initial few epochs, it plateaus quickly and is overtaken by *Hyperdyn* . The choice of batch size is arbitrary for Figure 1(d) as this observation is consistent across different batch sizes. We also compare *Hyperdyn* with a version of random search 5x, *i.e.*, at every epoch we invest 5x resources more in random search than in *Hyperdyn* . We find that random search is susceptible to higher variance than *Hyperdyn* , especially at larger batch sizes. Figure 1(d) shows a typical comparison of random search 5x and *Hyperdyn* . We employed a variant of Hyperband where we reduced the search space of learning rate by 0.1 instead of successive halving and did not find any improvement in performance over random search 5x.

Using *Hyperdyn* with Adam: In addition to SGD, Adam is commonly used for optimization. It is adaptive, computationally and memory efficient, but also works for non-stationary objectives and noisy environments Kingma & Ba (2014). Typically, Adam is employed for the initial part of the training process while SGD is used near the end. We use *Hyperdyn* to optimize the hyperparameters of Adam, i.e. learning rates β_1, β_2 in Kingma & Ba (2014). *Hyperdyn* tuned Adam is much faster than manually tuned Adam (in terms of number of iterations) and also generalizes better on ResNet-20 with batch size 128. As our HPO algorithm proceeds, we observe convergence of learning rates



(a) Top-1% Accuracy (b) β_1, β_2

Figure 3: *Hyperdyn* tuned Adam

as $\beta_1 \to 0$ and β_2 is around 1, i.e. the momentum coefficient β_1 becomes very small (Fig. 3(b)). It turns out that these limits are also needed to establish good theoretical properties for Adam; see Corollary 4.2 in Kingma & Ba (2014). This also matches the previous observation in Sutskever et al. (2013) that the momentum coefficient needs to be reduced to obtain good convergence.

4.2 Experiments on Imagenet

For Imagenet we used ResNet-50 and a batch size of 1000. This batch size is considerably larger than typical Imagenet experiments, where the batch sizes are $\sim$ 256. We compare in Figure 4 a *Hyperdyn* tuned Imagenet training process to a manually tuned one. Note that the batch size for the two are different because a manual Imagenet tuning is hard for a batch size of 1000. As a result we achieve the same accuracy much faster than a manually tuned training process. Additionally, results in You et al. (2017) suggest that (upto 40 epochs) *Hyperdyn* gives the best performance.

5 Conclusion

In this work we introduced a general framework for a certain class of hyperparameter optimization. The algorithm here has the advantage that it is fast, flexible – can be used on any objective function and training method, and stable on larger batch sizes. We demonstrated that our proposed optimizer is faster and at least as accurate as SGD (with momentum) or Adam. We also show that *Hyperdyn* is stable for larger batch sizes on Imagenet (achieves acceptable accuracy with 4x speed). We demonstrate how too much exploration can be detrimental to generalization accuracy of a training process, and propose a probabilistic "wait-and-watch" strategy.

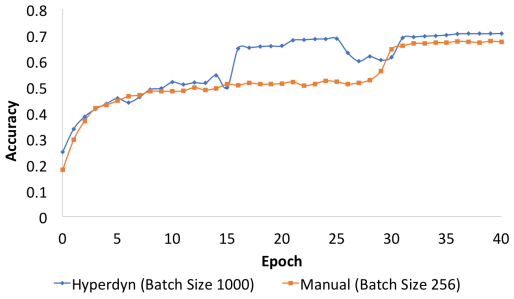


Figure 4: Top-1% Acc. of *Hyperdyn* vs Manual

Currently we do not parallelize *Hyperdyn* ; however, computing Validation_Accuracy on the suggestion from One_Step_BO can be easily parallelized. At each epoch we make only suggestion and the validation accuracy on this suggestion can be computed independently of the current hyperparameter setting. In the general case, when we make multiple suggestions we can parallelize in a similar fashion to Snoek et al. (2012). We also observe that epoch-based BO in Fig. 1(d) outperforms *Hyperdyn* in the initial epochs. One future direction maybe to have time varying temperature in Check_Accuracy_Trend based on epoch. We can also exploit the temporal gains obtained by using a backtrack algorithm at the later stages of the algorithm – at a stage when accuracy is sufficiently high but more sophisticated tuning is required to get the best error rates.

REFERENCES

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.

Saul B Gelfand and Sanjoy K Mitter. Recursive stochastic algorithms for global optimization in r^d. *SIAM Journal on Control and Optimization*, 29(5):999–1018, 1991.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Robert B Gramacy and Herbert K H Lee. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

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

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

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.

Mu Li. *Scaling Distributed Machine Learning with System and Algorithm Co-design*. PhD thesis, Intel, 2017.

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

A. Schoenauer-Sebag, M. Schoenauer, and M. Sebag. Stochastic Gradient Descent: Going As Fast As Possible But Not Faster. *ArXiv e-prints*, September 2017.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016.

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.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

K. Swersky, J. Snoek, and R. Prescott Adams. Freeze-Thaw Bayesian Optimization. *ArXiv e-prints*, June 2014.

Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *ArXiv e-prints*, May 2017.

Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer. 100-epoch ImageNet Training with AlexNet in 24 Minutes. *ArXiv e-prints*, September 2017.