Happy Mistakes: Why false promotions make ASHA robust to adversarial settings

Christian Belardi[1] **Rebecca Liu**[1] **Pakin Wirojwatanakul**[1]

Abstract

The simple greedy heuristic of the Successive Halving Algorithm (SHA) and its asynchronous counterpart, Asynchronous Successive Halving Algorithm (ASHA), suggests that these hyperparameter tuning algorithms perform sub-optimally in settings where the models that ultimately perform best perform poorly early on in training. In an attempt to expose this flaw by identifying these adversarial settings, we find that ASHA is surprisingly robust, while SHA selects a sub-optimal configuration as expected. Further study of this behavior shows that false promotions, thought to be a flaw of the algorithm, actually enable ASHA to perform well in adversarial settings. We provide a theoretical analysis of the two algorithms to show that ASHA has a non-zero probability of selecting the optimal configuration even when SHA is no longer able to do so. In addition, we provide empirical support demonstrating this phenomenon. Finally, we measure the increase in computation of ASHA by calculating the number of extra training epochs caused by false promotions.

1. Introduction

Hyperparameter tuning is a persistent challenge in machine learning, where searching is computationally expensive due to the cost of model training. Hyperparameters are often set manually using a priori information or optimized by searching over large hyperparameter spaces. In 2020, Li et al. proposed the Asynchronous Successive Halving Algorithm (ASHA), a parallelized hyperparameter optimization method inspired by the Successive Halving Algorithm (SHA), which is especially effective for large scale hyperparameter tuning (Li et al., 2020). By relaxing the synchronization constraint of SHA, ASHA outperforms SHA, as well as other hyperparameter tuning algorithms such as Bayesian Optimization and Population Based Training (Li et al., 2020).

Hyperparameter tuning is particularly challenging for high dimensional non-convex problems such as training neural networks, where reasoning about the effect of different hyperparameter combinations is intractable, or at the very least, inefficient. Therefore algorithms like ASHA are of great interest to the community.

2. Background

In this paper, we evaluate SHA and ASHA on adversarial settings where models that ultimately perform best—have the best validation error—perform poorly early on in training.

2.1. SHA

SHA (Karnin et al., 2013; Jamieson & Talwalkar, 2015) is a hyperparameter tuning algorithm that allots more training resources to configurations that perform well in training, and eliminates configurations that perform poorly. SHA gives a set number of resources for each configuration to begin training, r, and then evaluates all configurations and keeps the best $1/\eta$, where η is the reduction factor. Afterwards, these $1/\eta$ configurations are given η times more resources, and this process is continued until the configurations have used the maximum number of resources per configuration, R. Each round of training is a "rung," where rung 0 is the base, or initial, rung. SHA is presented in Algorithm 1.

2.2. ASHA

ASHA (Li et al., 2020) is a parallelized version of SHA. Unlike SHA, ASHA does not wait for all the configurations at a rung to complete before promoting to the next rung. ASHA will promote configurations to the next rung whenever there are at least η configurations at the rung. If no configurations are able to be promoted, ASHA adds a con-

[*]Equal contribution [1]College of Computer and Information Science, Cornell University, Ithaca, New York. Correspondence to: Christian Belardi <ckb73@cornell.edu>, Rebecca Liu <rll249@cornell.edu>, Pakin Wirojwatanakul <pw273@cornell.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

Algorithm 1 Successive Halving Algorithm: copied from (Li et al., 2020)

```
**Input:** number of configurations $n$, minimum resource $r$, maximum resource $R$, reduction factor $\eta$, minimum early-stopping rate $s$
$s_{max} = \lfloor \log_\eta(R/r) \rfloor$
**Assert:** $n \geq \eta^{s_{max}-s}$ so that at least one configuration will be allocated $R$.
$T = \texttt{get\_hyperparameter\_configuration}(n)$
**for** $i = 0$ **to** $s_{max} - s$ **do**
    $n_i = \lfloor n\eta^{-1} \rfloor$
    $r_i = r\eta^{i+s}$
    $L = \texttt{run\_then\_return\_val\_loss}\ (\theta, r_i) : \theta \in T$
    $T = \texttt{top\_k}\ (T, L, n_i/\eta)$
**end for**
**return** best configuration in $T$
```

Table 1. Model validation loss at each rung for false promotions examples in Section 2.3. The bold loss values show the loss of the models that would have been promoted at that rung.

RUNG	MODEL A	MODEL B	MODEL C	MODEL D
0	2	2	**1.8**	**1.8**
1	1.4	1.4	**1.6**	1.7
2	0.5	0.5	1.5	1.5

Algorithm 2 Asynchronous Successive Halving Algorithm: copied from (Li et al., 2020)

```
**Input:** minimum resource $r$, maximum resource $R$, reduction factor $eta$, minimum early-stopping rate $s$
**function** ASHA()
    **repeat**
        **for** each free worker **do**
            $(\theta, k) = \texttt{get\_job()}$
            $\texttt{run\_then\_return\_val\_loss}(\theta, r\eta^{s+k})$
        **end for**
        **for** completed job $(\theta, k)$ with loss $l$ **do**
            Update configuration $\theta$ with loss $l$
        **end for**
    **until** desired
**end function**
**function** get_job()
    **for** $k = \lfloor log_\eta(R/r) - s - 1$ to 0 **do**
        candidates = $\texttt{top\_k}$(rung $k$, |rung $k|/\eta$)
        promotable = $\{t \in$ candidates : $t$ not promoted$\}$
        **if** |promotable| $> 0$ **then**
            **return** promotable[0], $k + 1$
        **end if**
    **end for**
    Draw random configuration $\theta$.
    **return** $\theta, 0$
**end function**
```

figuration to the base rung. ASHA also does not require the user to input the total number of configs due to the nature of the algorithm. ASHA is presented in Algorithm 2. ASHA avoids the performance ”bottleneck” of SHA that requires all configuration in a rung to finish training before selecting top $1/\eta$ configurations.

2.3. False Promotions

In the ASHA algorithm, a false promotion is when a configuration, c, trained at rung i is promoted to rung $i + 1$ even though c would not have ultimately been in the final best $1/\eta$ configurations after all configurations at rung i finished training. A false promotion could occur if a configuration c at rung i is promoted if c was added earlier to rung i (or finished training faster at rung i), but would not have been promoted if c was added later (or trained slower).

As an example, suppose the number of ASHA workers is 1 , the minimum resource r is 1, the maximum resource R is 4, the reduction factor η is 2, the minimum early-stopping rate s is 0, and the set of configurations consists of A, B, C, and D, with validation losses of each model at each rung presented in Table 1. We use one worker so we only need to consider the order configurations were added, not the order of training completion.

Suppose the configurations are added to the base rung in the following order: A, B, C, and D. Configuration A will be promoted to rung 1. However, if the configurations were added in the order of C, A, B, and D, then configuration A would not be promoted to rung 1. In the first scenario, A is falsely promoted at rung 0. Table 2 demonstrates the order of promotions. Since, there is only one worker, all the events happen sequentially.

The SHA algorithm does not have false promotions because configuration are compared at a rung only after all configurations have finished training. Thus, the order of the configurations does not matter. For the same example, SHA will always promote configuration C and D to rung 1, and promote configuration C to rung 2 as shown in Figure 1.

2.4. Implementation Decisions

There are many possible resources to choose from when running SHA and/or ASHA, such as wall-clock time of training and number of training examples. However, in this paper, we only consider the case where the resources refer to the number of training epochs.

Also, for SHA, in the final step in the for loop where the top_k function is called, instead of returning the top n_i/η configurations, we return the best $\lceil n_i/\eta \rceil$ configurations

Table 2. Promotion scheme for ASHA example in Section 2.3. The subscript denotes the event order. Since there is only one worker, all the events happen sequentially.

RUNG	CONFIGURATION SEQUENCE: A, B, C, D
2	A_5
1	A_2 C_4
0	A_0 B_1 C_3

RUNG	CONFIGURATION SEQUENCE: C, A, B, D
2	C_6
1	C_2 D_5
0	C_0 A_1 B_3 D_4

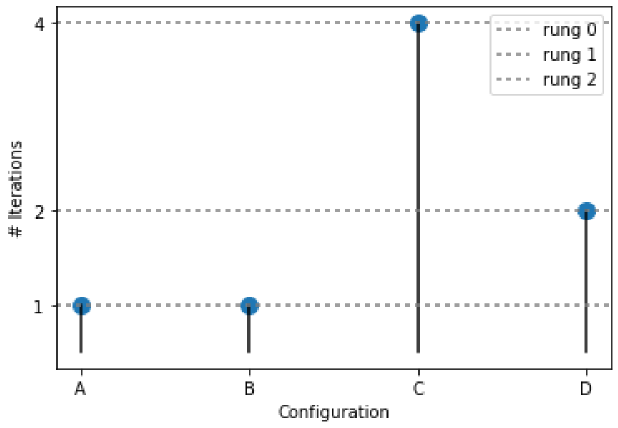

Figure 1. Promotion scheme for SHA Example in Section 2.3

in our implementation. We did this so that we could train the final configuration for as many rungs as we specified, and so that we would not run into the issue where zero configurations were promoted to the next rung.

In addition, ASHA does not require the user to limit the total number of configurations. However, for our experiments, we limit the total number of configurations to more easily compare the performance and results of SHA and ASHA.

3. Adversarial Settings

The simple greedy heuristic used by both SHA and ASHA promotes the configurations, with the lowest validation losses. This means that the configurations which achieve lower validation losses earlier in training will always be selected by a deterministic algorithm like SHA. We refer to these fast converging configurations as "precocious learners." By this heuristic, a deterministic algorithm like SHA will never select a configuration with high validation loss relative to the other configurations in the hyperparameter space. This is problematic when the configurations that ultimately achieve the lowest validation losses are not in the lowest $1/\eta$ of the configurations in the early rungs. We will call these superior, yet slow to converge, configurations "good learners." We define an adversarial setting to be any hyperparameter space where the set of configurations sampled from it can be partitioned into a set of good learners and a set of precocious learners. Furthermore, we can define transition points, or rungs, where the relationship between the good and precocious configurations changes. The first transition point will denote the last rung where all the good configurations have higher validation loss than all the precocious configurations. The second transition point will denote the first rung where all the good configurations have lower validation loss than all the precocious configurations. We call these transition points κ_1 and κ_2, and define them more formally in Section 4. See Figure 2 for a visualization of a real adversarial setting discussed further in Section 5.1.

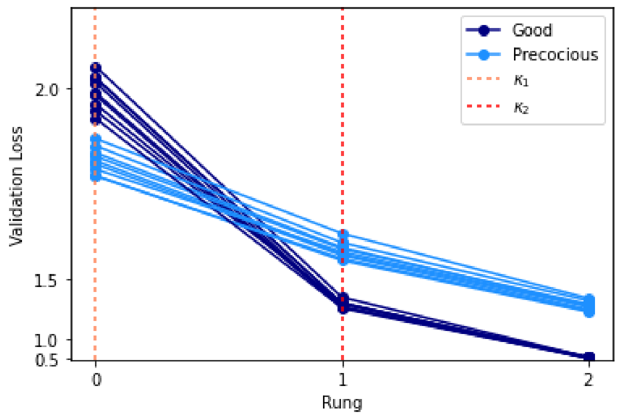

Figure 2. Plot of all configurations training curves from SGD vs. Adagrad adversarial setting discussed in Section 5.1

4. Theoretical Bounds

In this section we give a formal definition for an adversarial settings in terms of the two sets, good and precocious learners, as well as introduce additional notation to facilitate bounding SHA and ASHA. Let C represent some hyperparameter space, where samples from this space are the configurations we are optimizing over. Also, let the function $L_i : C \to \mathbb{R}$, take a configuration and return the validation loss of that configuration at rung i. Lastly, we assume all configuration validation losses are monotonic decreasing and that all configurations at each rung i take an equivalent amount of time to train.

Definition 1. The set of precocious models P in hyperparameter space C for some κ_1 is $P \subseteq C$ s.t. $\forall p \in P$, $\forall c \in \{C \setminus P\}, \forall i \leq \kappa_1, L_i(p) < L_i(m)$.

Definition 2. The set of good models G in hyperparameter space C for some κ_2 is $G \subseteq M$ s.t. $\forall g \in G, \forall c \in \{C \setminus G\}$, $\forall i \geq \kappa_2, L_i(g) < L_i(m)$.

Definition 3. An adversarial setting is a hyperparameter space A if $\exists \kappa_1, \kappa_2, \exists G, P$ s.t. $G \cup P = A \wedge G \cap P = \emptyset$.

Notice that κ_1, κ_2 define three phases of training/tuning over this hyperparameter space. From rung 0 through κ_1, it is impossible for the greedy heuristic to promote a configuration from G to the next rung given full information about the current rung. That is, if we know all the configurations' validation losses at the current rung, a good configuration will never be promoted because by definition, there are $|P|$ configurations with lower validation losses. Notice that given only partial information about the current rung, the greedy heuristic may select a good configuration dependent upon the subset of configurations there is information about. The second phase, consisting of the rungs between κ_1 and κ_2, is where the good configurations and precocious configurations begin to cross. Finally, in phase three from κ_2 though the max rung, the good configurations will all have lower validation losses than all the precocious configurations.

A natural question to ask is: *Given a hyperparameter tuning algorithm and its inputs, how many good configurations, denoted by n_G, must exist in the hyperparameter space in order for the probability of promoting one to the top rung to be non-zero?* In the following subsections we address this question for both SHA and ASHA.

4.1. A Lower Bound on SHA

Since SHA is synchronous, the number of workers has no bearing on the output of the algorithm, only on the speed at which it can run. By definition 1, if the highest rung reached by a good configuration is less than $\kappa_1 + 1$, then there is no way for SHA to select a good configuration at the final rung. Consider the case where a good configuration reaches κ_1+1. By definition 2, we know that this good configuration is guaranteed to have a lower validation loss than at least one precocious configuration at this rung. However, it could be better than more than one precocious configuration; it could even be the best configuration in rung $\kappa_1 + 1$. This means that if a good configuration reaches rung $\kappa_1 + 1$, the probability of a good configuration being promoted to the top rung is non-zero.

From this, we can derive an exact expression for the number of good configurations needed in the hyperparameter space in order for one good configuration to be promote to the top rung.

$$n_G > n - \frac{n}{\eta^{\kappa_1+1}}$$

From n_G we can compute the percentage of all configurations that must be good which we will denote as τ_{SHA}.

$$\tau_{SHA} = \frac{n_G}{n}$$

Notice if $\frac{|G|}{n} > \tau_{SHA}$, SHA with some probability will pick a good configuration. Otherwise, SHA has a zero probability of selecting a good configuration.

4.2. A Lower Bound on ASHA

Let m denote the number of workers used to run ASHA. In order to bound ASHA, we must consider the behavior of ASHA $\forall m$. ASHA has the interesting behavior of promoting configurations whenever possible, even before running all configurations at the base rung. If the m workers all select good configurations initially because ASHA promotes from the top down, and we assume the run times of all configurations at a given rung are equal, then a precocious configuration will never catch up to a good configuration. Thus, the good configuration will reach the top rung. In the case where the number of workers is smaller than η^{κ_2}, we need enough good configurations for a good configuration to reach rung κ_2. This is because by definition 2, once a good configuration reaches κ_2 it will always be promoted over all precocious configurations, and thus reach the top rung.

The expression for n_G w.r.t. to ASHA comes from this explanation.

$$n_G \geq \max(\eta^{\kappa_2}, m)$$

From n_G, we can compute the percentage of all configurations that must be good, which we will denote as τ_{ASHA}.

$$\tau_{ASHA} = \frac{n_G}{n}$$

Notice if $\frac{|G|}{n} \geq \tau_{ASHA}$, ASHA will pick a good configuration with some probability. Otherwise, ASHA has a zero probability of selecting a good configuration.

5. Empirical Evaluation

The first experiment demonstrates that ASHA has a higher probability of promoting good configurations than SHA as implied by the theoretical bounds of SHA and ASHA. The second experiment demonstrates that even in settings where the distinction between good and precocious learners is not obvious, ASHA can, with some likelihood, outperform SHA—in some cases significantly.

We use the MNIST dataset of hand written digits for all experiments (Deng, 2012). We also use the same neural network architecture for all experiments. We define a block in this architecture as a 3x3 convolutional layer, batch normalization layer, and activation. The activation is by default the ReLU activation function. The CNN architecture is defined as follows: block, 2x2 max pooling layer, block, adaptive average pooling (to 1x1 feature maps), and finally a linear layer. The first convolutional layer has 3 input channels, and 32 output channels. The second convolutional layer has 32 input and output channels. The linear layer takes in 32 dimensional vectors and outputs 10 dimensional vectors.

5.1. Probability of Selecting Good Configuration Experiment

All the configurations were created using the dataset and CNN architecture described above with batch size 16, learning rate of 1e-3, and weight decay of 1e-6. The precocious set of configurations had Adagrad as the optimizer and the good set of configurations had SGD as the optimizer. The validation losses achieved by the precocious and good configurations at each rung are displayed in Figure 2.

For a given fraction of good configurations g, a set of configurations C_g has $|C_g|g$ good configurations and $|C_g|(1-g)$ precocious configurations. The good configurations and precocious configurations are sub-sampled from the set of all configurations.

The SHA/ASHA algorithms were configured as such: The minimum resource r is 3, the maximum resource R is 27, the reduction factor η is 3, the minimum early-stopping rate s is 0, and the set of configurations is C_g of size 9 (n is 9). The number of workers for ASHA is 2. We use $g \in \{0, \frac{1}{9}, \frac{2}{9}, ..., \frac{8}{9}, 1\}$.

In order to determine the models selected by SHA and ASHA for a given a given set of configurations C_g, we trained C_g for the total number of epochs ASHA and SHA would have used in the setting specified above, and reported the validation loss and error at each rung: training epochs 3, 9, and 27.

The ASHA algorithm was simulated 500 times for each g using these calculated validation loss values, and the percentage of good configurations selected, the error achieved by the the selected model, and the number of false promotions for ASHA were recorded. SHA was simulated one time for each g, as it is deterministic.

From Figure 2, we observe that $\kappa_1 = 0$ and $\kappa_2 = 1$. From these values we can calculate τ_{SHA} and τ_{ASHA},

$$\tau_{SHA} = \frac{9 - \frac{9}{3^{0+1}}}{9} = \frac{2}{3}$$

$$\tau_{ASHA} = \frac{\max(3^1, 2)}{9} = \frac{1}{3}$$

Our empirical results as shown in Figure 3 demonstrate that when the percentage of good configurations in the hyper parameter space is between τ_{ASHA} and τ_{SHA}, ASHA has a non zero probability of selecting a good configuration, whereas SHA has a zero probability of selecting a good configuration.

Consequently, the configurations selected by ASHA have a lower average validation error as shown in Figure 4. Note that in Figure 4, the SHA graph is not flat prior to τ_{SHA} because the configuration set C_g evaluated is different at each point, and the particular precocious learner selected had an optimal validation loss, but not an optimal validation error.

ASHA's improved ability to select good configurations comes at the expense of false promotions. The higher the rung the false promotion happens, the more resources consumed. Since rung r_i uses $r\eta^{i+s}$ epochs, a false promotion at rung r_i would incur $r\eta^{i+s}$ extra epochs relative to SHA. Figure 5 demonstrates the number extra epochs used for 500 trials of C_4. We find that most trials for C_4 have around 1500-1750 extra epochs.

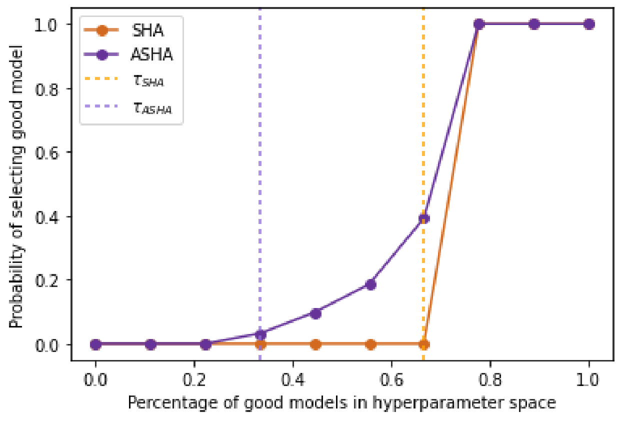

Figure 3. SGD vs. Adagrad Adversarial Setting: Showing probability of selecting good configuration as a function of percentage of good configurations in model space. SHA points are from 1 trial for each C_g. ASHA points are from 500 trials for each C_g

5.2. Experiment in non-trivial hyperparameter space

In this experiment, we compare the errors of the configurations selected by SHA and ASHA. We use a similar approach here as we did in the experiment described in 5.1. SHA and ASHA were given a minimum resource of

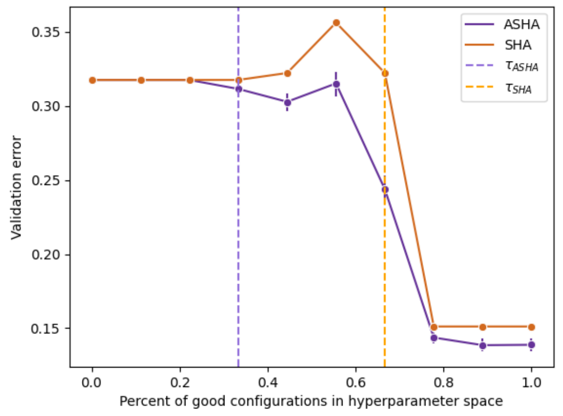

Figure 4. SGD vs. Adagrad Adversarial Setting: Showing validation error as a function of percentage of good models in model space. The SHA points are from 1 trial of SHA for each C_g. The ASHA points are the average of 500 trials for each C_g. The error bars for ASHA show a 95% confidence interval.

3, maximum resource of 27, and early-stopping rate of 0. We run all configurations for the total amount of resources, and then simulate both algorithms on the cached training curves. SHA is simulated once, since it is deterministic, while ASHA is simulated 100 times. For this experiment we only use 1 worker, so the execution times of the different configurations do not bias ASHA. The default setting for the hyperparameters is the following: batch size of 16, learning rate of 1e-3, weight decay of 1e-6, ReLU activation, and Nesterov optimizer with a momentum value of 0.9. Each setting evaluated has one hyperparameter, which we have manipulated. Every setting only has 9 models.

The setting ACTIVATIONS optimizes over three different activations: Leaky ReLU, ReLU, and Mish—resulting in 3 models for each activation (Misra, 2019).

The setting MOMENTUM optimizes over three different momentum values: 0.9, 0.99, and 0.999—resulting in 3 models for each momentum value.

The setting NESTEROV optimizes over three different optimizers: SGD, SGD with Momentum, and SGD with Nesterov Momentum—resulting in 3 models for each momentum value. Note that the same momentum value, 0.9, is used for both SGD with Momentum and SGD with Nesterov Momentum.

Finally, the setting RANDOMINIT leaves all hyperparameters the same as the default. The only difference between the configurations is the randomly initialized weights.

As expected, we find that ASHA in some settings can select

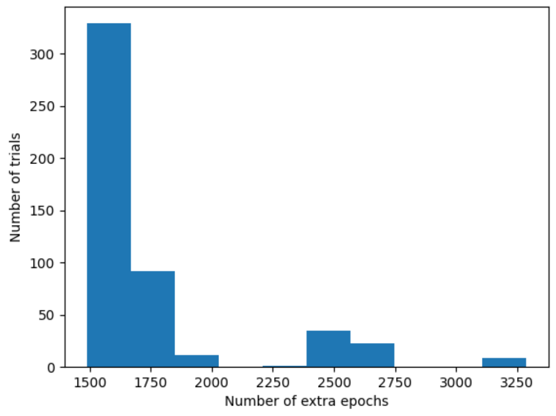

Figure 5. Histogram showing number of trials vs. number of extra epochs for 500 trials of C_4 (Section 5.1)

Table 3. Table of validations errors of configurations selected by SHA in various non-trivial settings, as well as average, max, and min validation errors of configurations selected by ASHA.

SETTINGS	SHA	ASHA AVERAGE	ASHA MAX	ASHA MIN
ACTIVATION	2.45%	2.45%	2.45%	**2.44%**
MOMENTUM	2.14%	2.08%	2.14%	**1.64%**
NESTEROV	2.60%	2.59%	2.60%	**2.44%**
RANDOMINIT	2.66%	2.65%	2.66%	**2.45%**

configurations that perform significantly better than those chosen by SHA. Furthermore, our results show that the configuration chosen by ASHA will never perform worse than the configuration chosen by SHA. These results are displayed in Table 3.

6. Discussion

ASHA with more than one worker is biased towards faster training configurations. If the set of configurations contains many fast precocious learners, ASHA will select a sub-optimal configurations. However, if the set of configurations contains many fast good learners, ASHA will select an optimal final configuration.

In regards to Table 3, we refer to these adversarial settings as non-trivial because the hyperparameter space does not closely resemble the exemplary hyperparameter space as is the case with the SGD vs. Adagrad example discussed above.

Another interesting observation is that our theoretical bound for ASHA is not dependent upon the size of the hyperparameter space but rather η and m.

7. Conclusion & Future Work

The asynchronous aspect of ASHA contributes much more than just a systems speed up to the algorithm. The asynchronicity introduces stochasticity to the sampling of configurations, which results in ASHA having a non-zero probability of selecting a configuration with a lower validation loss than SHA. In this paper, we only consider the specific case where resources are measured in number of epochs and configurations are evaluated by validation loss , the latter of which is used in SHA and ASHA (Li et al., 2020). A different selection of resource and/or evaluation metric may potentially avoid some of the pitfalls we have highlighted in this work. Specifically, defining resources in wall-clock time or amount of training data are interesting alternatives which may not exhibit the same behavior as we have seen for when resources are measured in epochs. These different cases are possible avenues for future work in this area. In addition, further analysis of the cases addressed here but at larger scales, such as a greater total number of configurations, would help illuminate the presence of this phenomenon in the the field.

References

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Jamieson, K. G. and Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. *CoRR*, abs/1502.07943, 2015. URL http://arxiv.org/abs/1502.07943.

Karnin, Z., Koren, T., and Somekh, O. Almost optimal exploration in multi-armed bandits. In Dasgupta, S. and McAllester, D. (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1238–1246, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/karnin13.html.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A. A system for massively parallel hyperparameter tuning. *Proceedings of the 3 rd MLSys Conference*, 2020.

Misra, D. Mish: A self regularized non-monotonic neural activation function. *CoRR*, abs/1908.08681, 2019. URL http://arxiv.org/abs/1908.08681.