Speeding up Hyper-parameter Optimization by Extrapolation of Learning Curves using Previous Builds

Akshay Chandrashekaran, Ian R. Lane

Department of Electrical and Computer Engineering,
Carnegie Mellon University, Pittsburgh, PA
akshayc@cmu.edu lane@cmu.edu

Abstract. Recent work has shown that the usage of extrapolation of learning curves to determine when to terminate a training build has been shown to be effective in reducing the number of epochs of training required for finding a good performing hyper-parameter configuration. However, the current technique uses the information only from the current build to make the prediction. We propose the usage of a simple regression based extrapolation model that uses the trajectories from previous builds to make predictions of new builds. This can be used to terminate poorly performing builds and thus, speed up hyper-parameter search with performance comparable to non-augmented hyper-parameter optimization techniques. We compare the predictions made by our model against that of the existing extrapolation technique in different tasks. We incorporate our approach into a pre-existing termination criterion. We incorporate this termination criterion into an existing hyper-parameter optimization toolkit. We analyze the performance of our approach and contrast it against a baseline in terms of quality of prediction in three different tasks. We show that our approach yields builds with performance comparable to the non-augmented version with fewer epochs, and outperforms an existing parametric extrapolation technique for two out of three tasks in terms of number of required epochs.

Keywords: Hyper-parameter Optimization, Extrapolation of Learning Curves, Bayesian Optimization, Deep Neural Networks

1 Introduction

Deep neural networks have become ubiquitous in various fields in machine learning to solve complex problems like speech recognition, machine translation, image recognition, etc. Training the neural network involves specifying a model architecture, and passing training data in the form of randomized mini-batches to find the gradient of a selected loss function. This gradient is then propagated back through the network, whose parameters are changed to minimize the loss.

Typically, training a neural network this way requires iterating over multiple epochs of training. The surface of the loss distributed over the model architecture is not convex, and its properties are generally unknown, thus making structure specification a non-trivial problem. Automated black-box hyper-parameter optimization techniques have

shown to outperform manual and grid search for hyper-parameter optimization. However, these techniques rely on the complete training of the model for a given hyper-parameter setting.

In the presence of large amounts of data, training neural networks up to completion, which entails either a termination condition based on the performance over a validation data-set or a bound a maximum number of epochs can take multiple days to finish. Given this, training multiple neural network architectures is computationally very expensive, and can quickly become infeasible.

Humans can use information gained from previous model builds and current model trajectory to determine how a model is likely to perform at the end of training. Some work has been done to automate the above intuition using a parametric model in [5]. Empirically, learning curve trajectories across different hyper-parameter configurations look remarkably similar. Exploration on the usage of the above observation to build better extrapolation strategies have only recently gained interest [9]. We propose an alternative extension to a probabilistic model that extrapolates the performance of a model at the early stages of training using the trajectories of previous model builds to identify and terminate poorly performing builds quickly.

Our contributions in this paper are as follows:

- We present a simple, yet effective approach to leverage the learning curves of previously completed builds to predict what the learning curve of the current build is likely to terminate in section 3.1.
- We incorporate this approach with an existing termination criterion algorithm in section 3.2.
- We analyze the quality of predictions obtained from this approach in terms of overall fit to the actual learning curve, as well as the performance at the final epoch in section 4.2 for three different tasks.
- We incorporate this new termination criterion within an existing hyper-parameter optimization toolkit.
- We analyze the performance of the proposed termination criterion and compare it against the baseline technique in section 4.3.

2 Related Work

We first review hyper-parameter optimization techniques popular in modern literature and previous attempts to model learning curves.

2.1 Hyper-parameter Optimization Techniques

Other than manual tuning, one of the most utilized hyper-parameter optimization techniques in machine learning has been the grid search. Recently, simple random sampling [1] has been shown to outperform grid search, particularly in high dimensional problems. Sophisticated sequential model based global optimization techniques employing Bayesian Optimization [12], tree of Parzen estimators [2], SMAC [8] have been shown

to perform even better on several data-sets and tasks. For our task, we have used relatively few hyper-parameters to optimize. Hence, based on [6], we integrate our proposed approach with Spearmint [12] for our experiments. However, this approach can be integrated with any other hyper-parameter optimization technique with relative ease.

2.2 Modelling Learning Curves

In this work, we are trying to model the performance of an iterative ML algorithm as a function of iterations. We consider the performance of the ML algorithm in terms of validation accuracy over epochs to be the learning curve. The goal is to predict the validation performance and terminate a build that is unlikely to beat the best one so far. Freeze-thaw Bayesian Optimization [15] is a GP based Bayesian Optimization technique that includes a parametric exponential decay model for modelling the learning curve. They use this to perform exploration of several configurations by temporarily pausing the training of models, and then resuming the training. This technique has been shown to work on matrix factorization, online latent Dirichlet allocation (LDA) and logistic regression, but has not performed well on deep neural networks [5]. In [5], the authors model the learning curve for each selected hyper-parameter configuration using a weighted combination of parametrized exponential decay models. The following is an explanation of their termination criterion as understood from the code, which varies a bit compared to the explanation given in their paper. Every m epochs, the performance values are gathered, and Markov chain Monte Carlo (MCMC) is run to estimate the distribution of the parameters of the extrapolation model. Using this, the predicted termination value and uncertainty of the prediction is computed. If the standard deviation of the prediction falls below a specified threshold, the probability of the current model outperforming the best model so far is computed. If this probability falls below another specified threshold, the training is terminated, with the expected value returned in lieu of the real loss. Otherwise, the training is allowed to proceed. This method is shown to give builds with performance comparable to un-augmented optimization techniques while speeding up optimization by a factor of 2. We will be using this approach as the baseline for comparison.

In [9], the authors have built Bayesian neural networks based on trajectories obtained from random samples of hyper-parameter configurations, which have shown to have better fit compared to [5]. However, they require the presence of a large number of prior trajectories to effectively train their network, which is problematic if the task is new, and very computationally expensive.

To our knowledge, we are not aware of any prior research into prediction of learning curve trajectories that uses only a few previous builds to make accurate and fast predictions of the performance in the termination stage.

3 Proposed Approach

Given a machine learning algorithm $\mathcal{A}$ having hyper-parameters $\lambda_1, \ldots, \lambda_n$ with respective domains $\Lambda_1, \ldots, \Lambda_n$, we define the hyper-parameter space as the hyper-cube $\boldsymbol{\Lambda} = \Lambda_1 \times \ldots \Lambda_n$. For each hyper-parameter setting $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, we use $\mathcal{A}_{\boldsymbol{\lambda}}$ to denote the

learning algorithm $\mathcal{A}$ using this setting. We further use $l_k(\boldsymbol{\lambda}) = \mathcal{L}_k(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ to denote the best validation accuracy till epoch k that $\mathcal{A}_{\boldsymbol{\lambda}}$ achieves on data $\mathcal{D}_{valid}$ when trained on $\mathcal{D}_{train}$. We further denote The hyper-parameter optimization $l_*(\boldsymbol{\lambda}) = \max_k l_k(\boldsymbol{\lambda})$ to be the maximum validation accuracy obtained for the algorithm using $\boldsymbol{\lambda}$ over all epochs. The hyper-parameter optimization problem is to find $\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} l_*(\boldsymbol{\lambda})$.

3.1 Ensemble of Learning Curves Model

In this section, we describe how we probabilistically extrapolate from a short initial portion of the learning curve for a given model, and information from previous builds to predict the validation accuracy at a later point. For simplicity, let $y_{r,k}$ indicates the best validation accuracy observed till epoch k for the r^{th} build, i.e. $y_{r,k} = l_k(\boldsymbol{\lambda}_r)$.

$\mathbf{y}_{r,1:n} = [y_{r,1}, y_{r,2}, \ldots, y_{r,n}]$ denotes the observed performance values for the r^{th} build till epoch n. m is the final epoch after which a build will be terminated ($m > n$).

$\mathbf{Y}_{1:r-1,1:m} = [\mathbf{y}_{1,m}, \mathbf{y}_{2,m}, \ldots, \mathbf{y}_{r-1,m}]$ is the observed performance values from previous $r-1$ builds.

Our objective is, given $R-1$ completed previous builds ($\mathbf{Y}_{1:R-1,1:m}$), and n epochs from the current build ($\mathbf{y}_{R,1:n}$), to predict the performance of the current build at termination: $y_{R,m}$. We solve this problem using an ensemble of learning curve models as described below.

Our approach is to transform the data from previous builds to approximately match the current build. We propose that the learning curve for the current build is a simple affine transformation of a previous build with an additive noise component. For each previous build $r = 1 : R-1$ for a given epoch k,

$$\begin{aligned} \hat{y_{R_r,k}} &= a_r y_{r,k} + b_r + \epsilon \\ \epsilon &\sim \mathcal{N}(0, \sigma^2) \end{aligned} \tag{1}$$

We assume the noise ϵ to be normally distributed with variance σ^2.

Using this definition, we construct a loss function comprised of all available information from the current build. This takes the form of a linear regression problem.

$$L_r = \frac{1}{n} ||\mathbf{y}_{R,1:n} - \hat{\mathbf{y}_{R_r,1:n}}||_2^2 + \frac{\theta_1}{2} \frac{(1-a_r)^2}{e^{\theta_2 n}} \tag{2}$$

where $\hat{\mathbf{y}_{R_r,1:n}} = [\hat{y_{R_r,1}}, \hat{y_{R_r,2}}, \ldots \hat{y_{R_r,n}}]$ The second term in the loss function is added to prevent pathological conditions of warping of the curves when only the initial epoch's information is present. This puts the majority of the contribution towards the loss to be by the offset term b_r for the first epoch, with this effect quickly dying out as more information becomes available.

The derivatives of the loss function with respect to the parameters a_r and b_r are given by:

$$\begin{aligned} \frac{\partial L_r}{\partial a_r} &= -\frac{1}{2n} (\mathbf{y}_{R,1:n} - \hat{\mathbf{y}_{R_r,1:n}})^T \mathbf{y}_{r,1:n} - \theta_1 \frac{(1-a_r)}{e^{\theta_2 n}} \\ \frac{\partial L_r}{\partial b_r} &= -\frac{1}{n} (\mathbf{y}_{R,1:n} - \hat{\mathbf{y}_{R_r,1:n}})^T \mathbf{1}_{1:n} \end{aligned} \tag{3}$$

where $.^T$ indicates the transpose of the vector, and $\mathbf{1}_{1:n}$ is an identity vector of length n.

Using the above loss and gradients, we perform simple gradient descent with multiple random start points for all the available previous builds. Once this is done, we select the top S parameter combinations and corresponding builds. We project the selected builds with their respective parameters. From this, we can get the expected value of the prediction and it's standard deviation for all subsequent epochs.

$$\begin{aligned} \mu_{\hat{y}_{R,k}} &= \frac{1}{S}\sum_{i=1}^{S} \hat{y_{R_{r_i},k}} \\ \sigma^2_{\hat{y}_{R,k}} &= \frac{1}{S-1}\sum_{i=1}^{S} (\hat{y_{R_{r_i},k}} - \mu_{\hat{y}_{R,k}})^2 \end{aligned} \tag{4}$$

Using the above mean and standard deviation definitions, we can estimate the probability that y_m exceeds a certain value $\tilde{y}$ as

$$P(y_m \geq \tilde{y} | \mathbf{y}_{R,1:n}, \mathbf{Y}_{1:R-1,1:m}) = 1 - \Phi(\tilde{y}; \mu_{\hat{y}_{R,m}}, \sigma^2_{\hat{y}_{R,m}}) \tag{5}$$

where $\Phi(.; \mu, \sigma^2)$ is the cumulative distribution function of a Gaussian with mean μ and variance σ^2.

The rationale behind the above method is inspired by the theory of wisdom of crowds [14] [16]. Our method satisfies the criteria required to form a wise crowd as described therein.

- *Diversity of Opinion*: Each estimator uses the validation accuracy trajectory of only it's own previous build.
- *Independence*: Each estimator trains its own parameters independent of any previous builds or concurrently built estimator.
- *Decentralization*: Each estimator's regression depends only on the validation accuracy trajectories of the current build and the one previous build it is based on.
- *Aggregation*: Equations 4 and 5 are the mechanisms to convert private judgements (estimate of prediction) to a collective decision.

3.2 Termination Criterion

We use a modified version of the termination criterion as described in [5] using the predictive model described above to speed up hyper-parameter search. We keep track of the best performance $\hat{y}$ found so far. Each time the optimizer queries the performance $l_*(\boldsymbol{\lambda})$ for a hyper-parameter setting $\boldsymbol{\lambda}$, we begin the training of the DNN using λ as usual. At the end of each epoch n, we record the performance over the validation set $y_{R,1:n}$. We run the algorithm described in section 3.1 to determine $m_{\hat{y}_{R,m}}, \sigma^2_{\hat{y}_{R,k}}$ and $P(y_m \geq \hat{y} | y_{R,1:n}, y_{1:R-1,1:m})$. We use this information in algorithm 1 to determine whether or not to terminate the current build. If the criterion dictates that the build be terminated, we replace it's final value $l_*(\boldsymbol{\lambda})$ to be the expected value derived by the ensemble method.

ConservativeTerminationCriterion `Procedure`

Data: current learning curve: $\mathbf{y}_{R,1:n}$,
previous learning curves: $\mathbf{Y}_{1:R-1,1:m}$ $(m < n)$,
predictive mean: $\mu_{\hat{y}_{R,m}}$,
predictive standard deviation: $\sigma_{\hat{y}_{R,k}}$,
probability of exceeding best observed value: $P(y_m \geq \tilde{y}|\mathbf{y}_{R,1:n}, \mathbf{Y}_{1:R-1,1:m})$,
probability threshold: δ,
standard deviation threshold: σ_{thresh}

```
if y_n > ŷ then
    Continue Training till completion
else
    if P(y_m ≥ ỹ|y_{R,1:n}, Y_{1:R−1,1:m}) ≥ δ then
        Continue Training for next epoch
    else
        if σ_{ŷ_{R,k}} ≥ σ_{thresh} then
            Continue Training for next epoch
        else
            Terminate build
            return μ_{ŷ_{R,m}}
        end
    end
end
```

Algorithm 1: Procedure for the Conservative Termination Criterion

Practical Considerations Empirically, we have observed that on integrating the above with our hyper-parameter optimization toolkit, we observed that for some of the tasks, if either of the initial build points performed really well, then the termination criterion became very aggressive in pruning off paths, which could result in builds that would have become viable in later epochs getting pruned very quickly. Hence, we have enforced a condition that a few initial builds be trained till completion. This empirically results in a better spread of validation accuracy trajectories that could be used for comparison, resulting in lower prediction loss. Another reason for this enforcement is to satisfy the independence condition for the wisdom of crowds. In the initial builds, Bayesian optimization is much more explorative in nature. Hence, the first few builds can, in some sense be considered to be independent of each other.

Another empirical observation showed that the optimization technique usually underpredicted the performance of the current build, despite the enforced condition that the validation accuracy is non-decreasing. Hence, we have employed a recency weighting schema during training, which has a geometrically decreasing weight-age to the residual at earlier epochs as opposed the residual of later epochs.

Finally, we have also incorporated a monotonicity check during the gradient computation: The predicted accuracy of the estimator cannot be lesser than the best observed value so far. This is a logical check based on our definition of the validation accuracy at a given epoch.

4 Experimental Setup

4.1 Tasks

To test the validity of our proposed approach, we performed empirical tests in three different tasks. We describe the training and evaluation procedure for each of the tasks below.

MNIST Image Classification The MNIST image classification task deals with classifying hand-written digit images of size 28×28 pixels. We use a simple fully connected DNN network here to take in a single image and return a distribution over the 10 labels. The hyper-parameters and ranges are given in table 1. The data-set consists of 60000 training images and 10000 test images. Each network is trained up to 20 epochs, with the epoch giving the best test accuracy being selected as the final model for the given hyper-parameter configuration. The training was done using MXNet training toolkit using a single GPU. The performance metric is the accuracy of classification over the validation data-set, which is to be maximized.

Hyper-parameter	Type	Minimum	Maximum	Step
Number of layers	int	1	8	1
Number of neurons per layer	int	32	3200	32
Learning rate	float	0.001	0.1	-
Learning rate factor	float	0.5	0.9	0.1
Learning rate step epochs	list	{[4,8,12,16], [5,10,15], [10]}		

Table 1: Hyper-parameters for MNIST classification task

CIFAR-10 Image Classification The CIFAR-10 data-set [10] consists of 60000 32×32 images belonging to 10 different classes, with 6000 images per class as the training corpus, and 10000 images in the test corpus. For this task, we implement a variation of the ResNet [7] architecture for classifying an image into the 10 classes. The hyper-parameters and ranges for this model are given in table 2. For the number of conv units in each segment, we had a geometric progression based on the segment number. The first segment would have 16 conv units per layer, the second 32, the third 64, and so on. The training was done using MXNet training toolkit on a single GPU. The performance metric is the accuracy of classification over the validation data-set, which is to be maximized.

Large Vocabulary Continuous Speech Recognition For this task, we selected the optimization of a simple fully-connected deep neural network (DNN) hidden Markov model (HMM) hybrid speech recognition system trained on the Wall Street Journal (WSJ) corpus. This consists of 286 hours of training data, with a validation set of 503

Hyper-parameter	Type	Minimum	Maximum	Step
Number of residual segments	int	1	8	1
Number of Conv Layers per segment	int	1	5	1
Learning rate	float	0.001	0.1	-
Learning rate factor	float	0.5	0.9	0.1
Learning rate step epochs	list	{[30, 60, 90. ... , 270], [60, 120, 180, 240], [90, 180, 270], [120,240], [150], [200,250]}		

Table 2: Hyper-parameters for CIFAR-10 classification task using the ResNet architecture

utterances. The following is a brief description of the training pipeline. An initial Gaussian Mixture Model based system is trained using the Kaldi recipes [11] up to the tri4b stage. Once this is done, a training corpus of extracted log-mel filter-bank features [4], with labels obtained by the alignment of the trained text to the audio is generated. We use 40 mel-filterbanks based on [13]. 10 iterations of DNN training using stochastic gradient descent is performed using the training framework used in [3]. The iteration giving the best validation set word error rate is selected as the final model trained for the given hyper-parameter configuration. The structure of the DNN model depends on the model hyper-parameters of number of input frames spliced, the number of hidden layers, and the number of neurons per hidden layer. The hyper-parameters and ranges for this task are given in table 3. The performance metric for this task is word error rate (WER), which is the ratio of the total number of substitutions, insertions, and deletions to be performed on a hypothesis to match the reference. This needs to be minimized for this task. To fit the description of the proposed approach, we take 1 - WER as the metric to maximize.

Hyper-parameter	Type	Minimum	Maximum	Step
Frame Context Splicing	int	0	9	1
Number of hidden layers	int	1	6	1
Number of neurons per hidden layer	int	512	2944	128

Table 3: Hyper-parameters for the WSJ LVCSR Task

4.2 Experiments on Quality of Predictions

The first set of experiments involve finding the quality of the predictions made by the proposed method. We look at two different metrics for this. The first, is the average root mean square error (RMSE) of the prediction across all the epochs. This gives us an idea of how well the overall prediction fits with the actual observations. The second is the average RMSE of the prediction over only the final epochs. This gives an indication of

the performance of the estimators for the termination criterion that will be integrated to any hyper-parameter optimization technique.

Experimental Setup We compute these numbers while progressively increasing the number of previous builds, and progressively increasing the number of available epochs from the current build. For the MNIST and CIFAR-10 tasks, we conducted 100 builds with randomly varied hyper-parameter setting to construct the initial data-set. For the WSJ task, we conducted 30 builds with random hyper-parameters. For each build, we select the specified number previous builds from the remaining builds randomly. We use our proposed approach to compute the predictions for all the remaining epochs for the current build. We conduct the above 50 times, and average the results over all the resultant combinations.

For the ensemble approach, we perform 100 iterations of gradient descent with random initialization of the affine parameters per previous build, and select the top 100 transforms with the least training loss to construct the ensemble estimator.

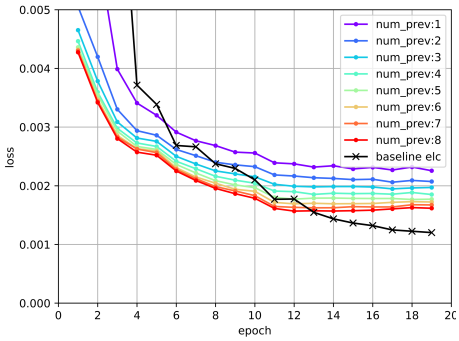
Results The results for assessing the quality of predictions are given in Fig. 1 and Fig. 2.

Figures (1a), (1c) and (1e) show the variation of the RMSE of the predictions over all epochs as a functions of number of epochs completed in the current build, while figures (1b), (1d) and (1f) show the corresponding average predictive standard deviations across all epochs.

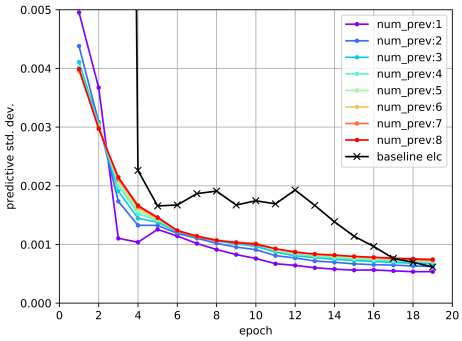
Figures (2a), (2c) and (2e) show the variation of the RMSE of the predictions of the final epoch as a functions of number of epochs completed in the current build, while figures (2b), (2d) and (2f) show the corresponding average predictive standard deviations over only the final epoch.

We observe the for all the cases, in the proposed method, using successively more number of previous builds to predict the result for the current build improves the prediction loss. Looking at the predictions, we observe the average RMSE of predictions across epochs and average RMSE of the prediction for the final epoch goes down as the number of previously completed builds increases. In the initial epochs for all three, our proposed approach performs significantly better than the baseline prediction method in both metrics. For all the tasks, we observe that,if given 5 initial builds, we match or outperform the baseline method for at least the initial 20% of the epochs. For the remaining epochs, the results are more of a mixed bag, but the proposed and baseline method perform comparably across all the tasks.

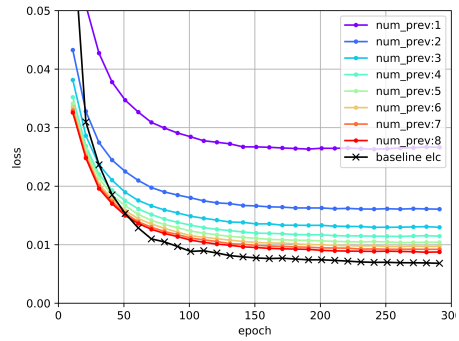
We observe that for the CIFAR-10 task, we observe a higher loss in terms of prediction over all the different scenarios, however the baseline is dramatically more confident in it's prediction. This may result in potentially terminating builds in the initial epochs, even if those builds can potentially get better at later stages. In contrast, the standard deviations of our proposed ensemble approach is more in line with the average RMSE across all tasks.



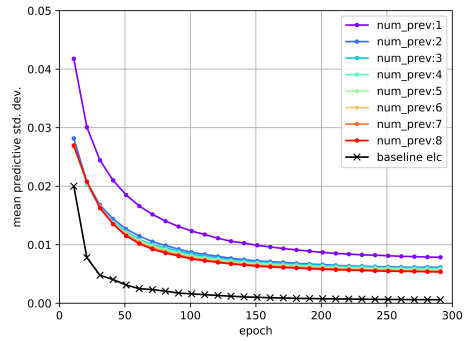
(a) MNIST: Average Prediction RMSE



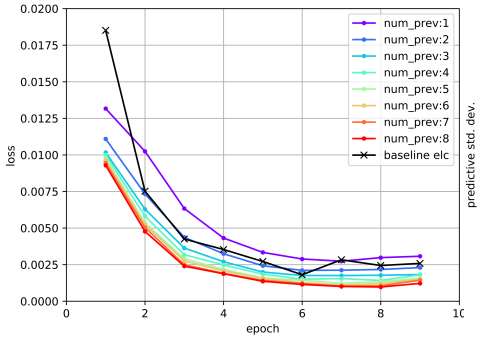
(b) MNIST: Average Prediction Standard Deviation



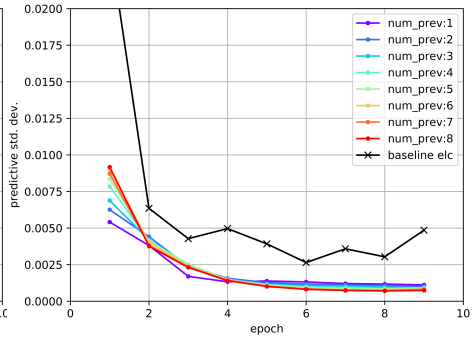
(c) CIFAR-10: Average Prediction RMSE



(d) CIFAR-10: Average Prediction Standard Deviation

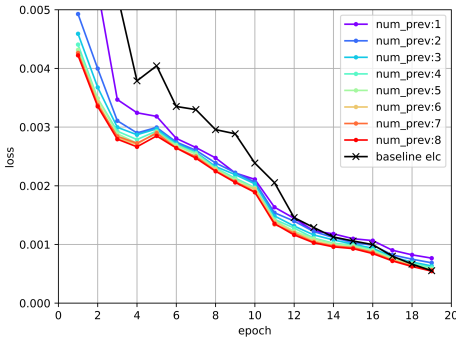


(e) WSJ: Average Prediction RMSE

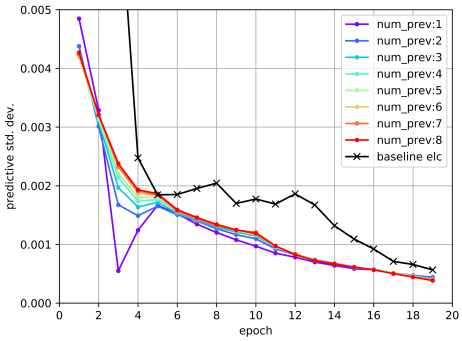


(f) WSJ: Average Prediction Standard Deviation

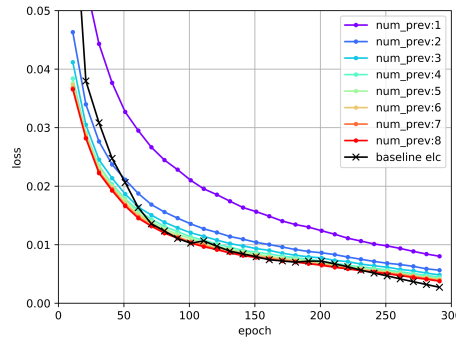
Fig. 1: Quality of Predictions in three tasks over all epochs. The left plots show the average RMSE, while the right shows the predictive standard deviation over all epochs. num_prev indicates the number of previous builds used to build the estimator.



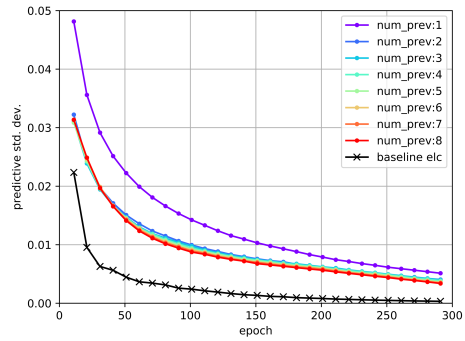
(a) MNIST: Average Prediction RMSE



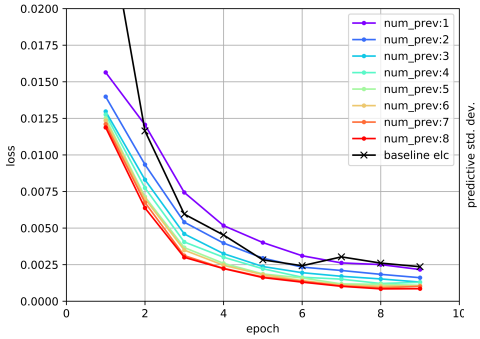
(b) MNIST: Average Prediction Standard Deviation



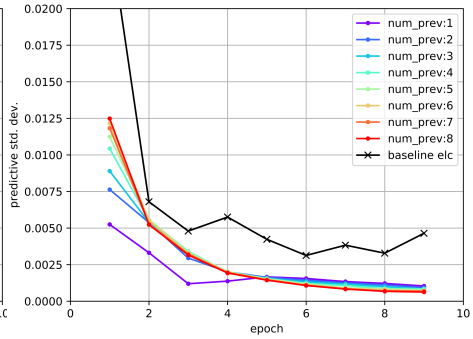
(c) CIFAR-10: Average Prediction RMSE



(d) CIFAR-10: Average Prediction Standard Deviation



(e) WSJ: Average Prediction RMSE



(f) WSJ: Average Prediction Standard Deviation

Fig. 2: Quality of Predictions in three tasks at the final epoch. The left side shows the average RMSE at the final epoch, while the right shows the average standard deviation over the final epoch. num_prev indicates the number of previous builds used to build the estimator.

4.3 Experiments on Integration with Bayesian Optimization

The second set of experiments evaluate the performance of integrating the proposed termination criterion with an existing hyper-parameter optimization toolkit. The experiments performed here use a combination of integral and floating point hyper-parameters which do not have any hierarchy. Also, the number of hyper-parameters to be optimized is low for our current test cases. Hence, based on the findings of [6], we have selected Bayesian Optimization as the sequential model based optimization method for our overarching automated hyper-parameter optimization technique. We have modified the Spearmint toolkit [12] to incorporate the termination criterion.

Experimental setup For each task, we perform three sub-experiments. The first performs normal Bayesian Optimization without any early stopping (Non-augmented). So, each build will run till completion. The second experiment incorporates the termination criterion using the baseline extrapolation of learning curves as described in [5] into Bayesian optimization (Baseline ELC). The third incorporates the termination criterion using the proposed extrapolation of learning curves technique as described in section 3 into the Bayesian optimization (Proposed ELC).

For each sub-experiment for MNIST and WSJ, we perform 30 builds in a single iteration of optimization. For CIFAR-10, we perform 20 builds for a single optimization iteration. For all the sub-experiments, we analyze the best validation performance metric obtained at the end of the optimization run, and the total number of epochs of learning required to reach the end of optimization. We are aware that each epoch within each model build can have dramatically different time requirements. However, because we are trying to reduce the number of epochs required for training, we do not report the actual wall time required to finish the optimization run.

Results Table. 4 shows a comparison of the performance of the two extrapolation approaches. Each sub-table performs the comparison for each individual task. We observe that in the MNIST and WSJ tasks, the proposed ensemble approach requires fewer number of epochs to find the best performing build compared to the baseline extrapolation technique, with little to no loss in performance. In the case of the CIFAR-10 task, though the baseline performs fewer epochs, there is a substantial decrease in performance in terms of accuracy. We presume this is due to the learning rate annealing feature (the last row in table 2). Empirically, we saw that at the epochs where the learning rate was annealed, there was a significant improvement in the validation accuracy. This information of probable future improvement is not captured by the baseline extrapolation technique. However, since similar behavior was probably observed in the previously completed builds used by the proposed extrapolation method, the termination criterion does not kill the job immediately. This can also be tied in to the observations of the loss and standard deviation trajectories in figures (2c) and (2d).

Although we have shown only one run of hyper-parameter optimization for each, we do not expect to see a large variance between multiple runs.

Extrapolation type	**Total Epochs (Relative improvement)**	**Best Accuracy(%) (Relative Improvement)**
Non-augmented	600	98.76%
Baseline ELC	370 (38.3)%	98.76% (0.0%)
Proposed ELC	257 (57.16%)	98.63% (-0.1%)

(a) Task: MNIST

Extrapolation type	**Total Epochs (Relative improvement)**	**Best Accuracy(%) (Relative Improvement)**
Non-augmented	6000	91.359%
Baseline ELC	4292 (28.9%)	90.5% (-0.9%)
Proposed ELC	5019 (16.3%)	90.92% (-0.4%)

(b) Task: CIFAR-10

Extrapolation type	**Total Epochs (Relative improvement)**	**Best WER(%) (Relative Improvement)**
Non-augmented	300	7.85%
Baseline ELC	201 (33%)	7.85% (0%)
Proposed ELC	171 (43%)	7.85% (0%)

(c) Task: WSJ

Table 4: Performance comparison on integration with Bayesian Optimization

5 Limitations and Future Directions

The proposed approach currently uses only the information from completed builds to make the prediction for new builds. Due to this, it can make predictions only up to the number of epochs in the current build. The baseline method [5], although not explored, does not suffer from this limitation. One possible work-around could be to use parametric models like the baseline approach on the previously completed builds, and train an ensemble predictor using those.

In it's present form, it relies on the presence diversity in the initial builds to come up with good extrapolation models for new builds. Though we have used a single probability threshold, standard deviation threshold, and minimum number of required initial builds across the tasks, there is no guarantee that this value will produce the optimal result, i.e. best build in least number of epochs. Other tasks could potentially have differing thresholds.

Our technique relies on the random initialization of the affine transformation parameters to result in the presence of a variety in the training loss values which will guide which weak estimators are selected. This can be made more formal using Markov Chain Monte Carlo (MCMC), which we plan to explore in the future. Using this could also incorporate some of the practical considerations mentioned in 3.2 in a more formal manner.

Finally, the proposed approach does not use the build hyper-parameters directly to guide the ensemble construction. This is another future direction we plan to explore.

6 Conclusion

We have studied an ensemble approach for modelling learning curves of machine learning algorithms. We have analyzed the performance of this technique across three different tasks. We have demonstrated its performance when integrated with an automated optimization technique over these three tasks. This approach requires relatively small overhead for the estimation, and performs remarkably well in a majority of the tasks. It lends itself well to sequential optimization techniques since it relies on the presence of few builds that have run to completion to make predictions for subsequent builds, and shows improvement in prediction accuracy as more builds get completed. We have discussed the strengths and limitations of this new approach compared to previous approaches, and have discussed some potential ideas for further improvement of this technique.

Note

The source code is available at
`https://github.com/akshayc11/Spearmint`
under the branch `elc`. Please refer to this for more details.

References

1. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13(Feb), 281–305 (2012)
2. 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)
3. Chan, W., Lane, I.: Deep recurrent neural networks for acoustic modelling. arXiv preprint arXiv:1504.01482 (2015)
4. Davis, S., Mermelstein, P.: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE transactions on acoustics, speech, and signal processing 28(4), 357–366 (1980)
5. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: International Joint Conference on Artificial Intelligence (IJCAI), 2015. pp. 3460–3468. AAAI Press / International Joint Conferences on Artificial Intelligence (2015)
6. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: NIPS workshop on Bayesian Optimization in Theory and Practice. pp. 1–5 (2013)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
8. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration (extended version). Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep. (2010)
9. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with bayesian neural networks. Proc. of ICLR 17 (2017)
10. Krizhevsky, A.: Learning multiple layers of features from tiny images. Citeseer (2009)
11. Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al.: The kaldi speech recognition toolkit. In: IEEE 2011 workshop on automatic speech recognition and understanding. IEEE Signal Processing Society (2011)
12. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems. pp. 2951–2959 (2012)
13. Soltau, H., Saon, G., Kingsbury, B.: The ibm attila speech recognition toolkit. In: Spoken Language Technology Workshop (SLT), 2010 IEEE. pp. 97–102. IEEE (2010)
14. Surowiecki, J.: The wisdom of crowds. Anchor (2005)
15. Swersky, K., Snoek, J., Adams, R.P.: Freeze-thaw bayesian optimization. arXiv preprint arXiv:1406.3896 (2014)
16. Yampolskiy, R.V., Ashby, L., Hassan, L.: Wisdom of artificial crowds-a metaheuristic algorithm for optimization. Journal of Intelligent Learning Systems and Applications 4(2), 98 (2012)