Persistent Selective Pruning

Kshitij Bharat Sanghvi
Department of Computer Science
Courant Institute of Mathematical Sciences
NY, USA
kbs391@nyu.edu

***Abstract*—In recent years there is a massive boom in hand-held computational devices. These devices have relatively lower memory and computation capabilities than a workstation. Software applications are incorporating neural networks on the local machine for varied purposes. Additionally, recent papers suggest that there is always a sub-network that can emulate similar performance, rendering the computation of the remaining neurons unnecessary and thus, reducing overall efficiency. To that end, I introduce a novel pruning algorithm for neuron selection. Each neuron has an associated value that measures its relative importance in terms of its contribution and updates at each step. For a defined interval, the neurons with associated values higher than a threshold - trade-off variable, are deemed necessary and added to the sparse network. Recent work advocates tuning a sparse network than retraining the model from scratch. For performance measure, we compare the results of a fully connected network to that of a sparse on the CIFAR-10 dataset. Finally, I propose a one-click deployment pipeline that is framework independent and incorporates the proposed pruning methodology.**

I. Introduction

The prowess of deep neural networks first came into the limelight when Alex Krizhevsky introduced Alexnet[1], significantly surpassing classification accuracies of traditional methods on the Imagenet[2]. This event was a turning point and resulted in spawning several varied species of neural networks. The computationally intense VGG models[3] and highly optimized GoogleNet[4], all came into existence within a few years. As the complexity of the models grew, distributed learning algorithms came into the picture to speed up the process. There remains a fundamental want to achieving the best performance at the cost of the least number of computations. This want is extremely desired in edge devices due to the limited computing problem and a required inference latency. Any kind of reduction in computation at any stage of the pipeline always improves efficiency. The question remains if there exists a sub-network within a network that performs equivalently[5]. If the claim is true, then the removal of the excess neurons can reduce the number of computations to a great extent. This leads to the introduction of pruning methods - removing neurons that are deemed unnecessary reducing overall complexity and computation requirement (Fig 1). I introduce a novel algorithm that takes into account the "activeness" of a particular neuron as a metric for its relative importance concerning a particular dataset. In my algorithm, I propose fine-tuning then retraining as the former provides better empirical results. With the advent of transfer learning, there exists a general trend of adopting pre-trained models and fine-tuning them, rather than training from scratch from datasets that display some kind of similarity[6]. Pruning can prove to be of special significance in such scenarios. However, the task imposes certain restrictions as pruning might result in fracturing fragile internal co-adaptations resulting in an irreparable loss in performance[7]. The proposed algorithm inherently prevents the fracturing of internal coadaptations.

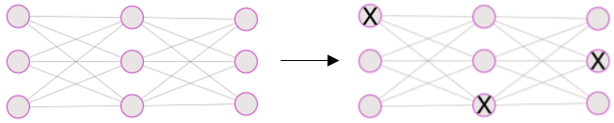


Fig. 1. Pruning.

II. Problem

The most basic problem at hand is to define a criterion that separates a "valuable" neuron from an unnecessary one. The absence of a provable algorithm that ranks the relative importance of a neuron in a neural network within each layer based on its attributes directs us to use empirical results to make our judgments. To present my intuition I use the following architecture (Fig. 2). The architecture is mainly dictated by the lack of distributed infrastructure and high-performance GPUs as the analysis runs into thousands of epochs and requires exorbitant amounts of memory. The hardware limits the use of a relatively simple data set like CIFAR-10 instead of Imagenet. Additionally, the proposed algorithm empirically works better for deeper networks and larger datasets.

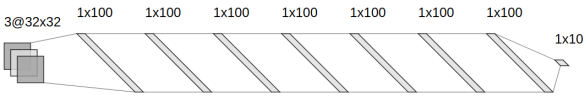


Fig. 2. Architecture.

To examine the behavior of neurons we look at their activation values for a particular batch of input data. I plot the histogram of the normalized activation values segregated across 50 bins (Fig 3 Fig 4).

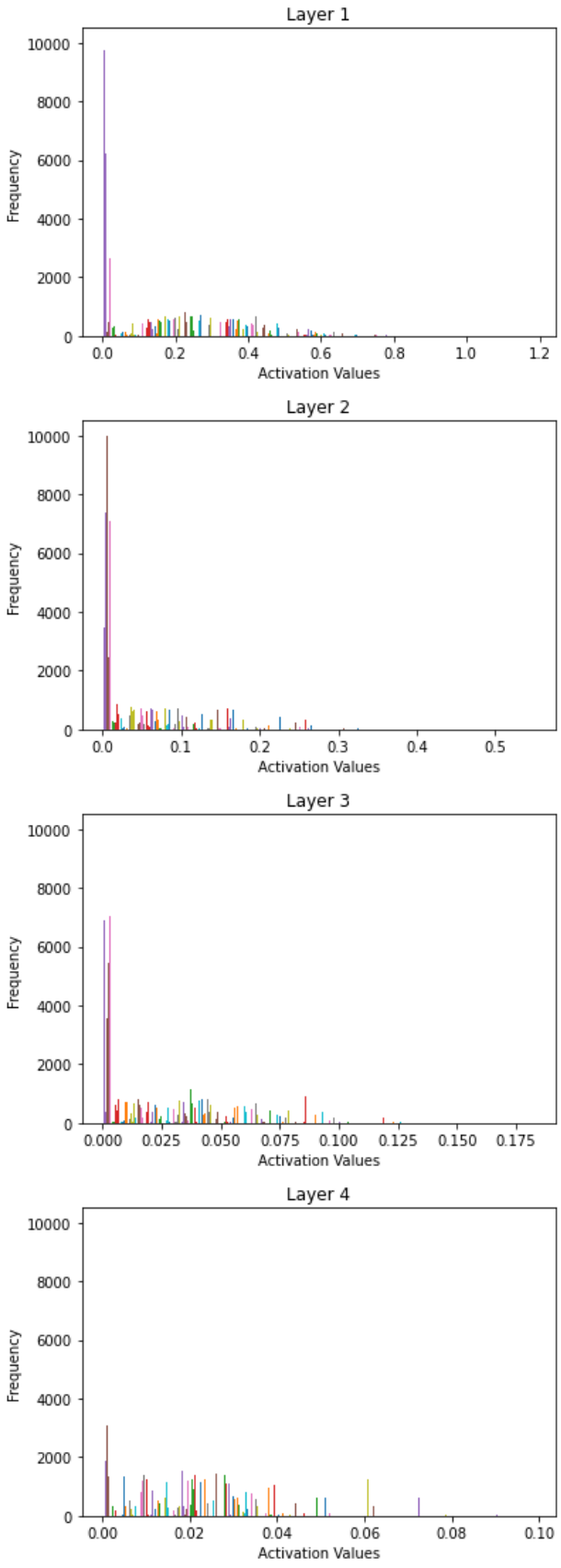


Fig. 3. Activation Frequency.

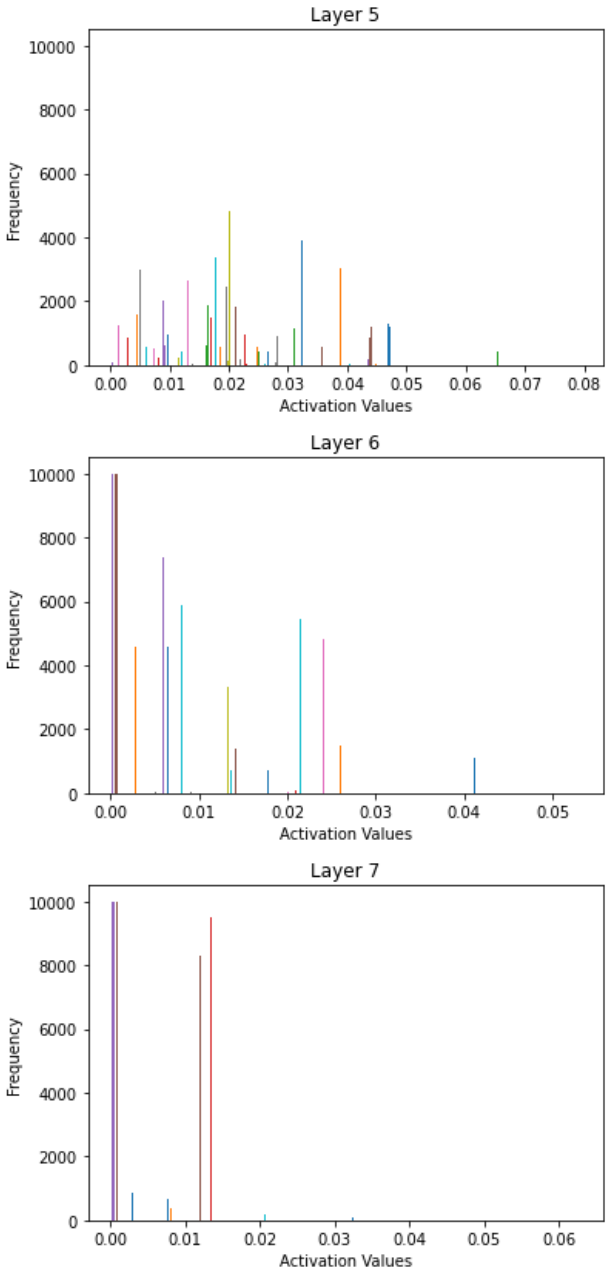


Fig. 4. Activation Frequency Contd.

We can observe that for most of the layers, the frequency of neurons being relatively ”dead” (low activation value) is high. I propose that neurons that remain active are can be considered being important. Next, I display the frequency of neurons in each layer and that have the mean activation values above a certain threshold (Fig 5).

This gives us an idea of how the activation values differ across layers. After testing various architectures, the general trend that followed was that as the depth increases the number of neurons above a certain threshold decreases. This is reinforced by the analogy that the shallow layers are responsible for identifying smaller features and the deeper layers are

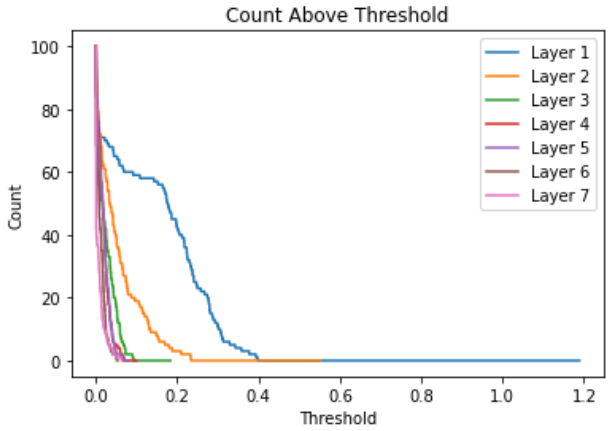


Fig. 5. Architecture.

looking for bigger patterns. Thus, as we now have a working methodology that helps us identify "important" neurons, I define the pruning algorithm.

III. Method

Algorithm 1 depicts the approach followed. The algorithm requires a trained model as input and a sample batch B of size b from the target dataset D. For each input data, the algorithm first performs a forward pass and records the activation values of each neuron in a layerwise fashion. Once all the forward passes are performed, it then calculates the mean activation values for each of the neurons within each layer. Subsequently, we can calculate the mean activation of each layer. We now have our pruning criterion involving a hyper-parameter γ and we can compute which neurons can be deemed unnecessary.

A. Abbreviations

D - The target Dataset
B - Sample Batch from the Dataset
I, y - Input and corresponding label
M - Model to Prune
M_{new} - Pruned Model
b - Cardinality of the batch B
L - Number of layers in M, $M.GetNumberOfLayers()$
N_l - Number of neurons in l^{th} of model M
$A_{n,l,i}$ - Activation value of n^{th} neuron in l^{th} layer for i^{th} input
$\bar{A}_{n,l}$ - Mean value of n^{th} neuron in l^{th} layer for all input in B
γ - Trade-off hyperparameter and $\gamma \in \mathcal{Z}^+$.

B. Sample Batch: B

The algorithm suffers significantly if the sample batch is not uniformly distributed over all the labels. If the batch is under-representative of certain labels then those instances are bound to suffer. Moreover, including only those labels that are correctly classified by M has empirically shown to provide better results, given all the classes are equally distributed. This is in accordance with the expected behavior as the incorrect inferences activate another set of neurons, giving them higher importance than they hold. The best strategy is to uniformly sample data across all classes for which the model M has made the correct inference.

Algorithm 1: Persistent Pruning

Input: $B = ((I_1, y_1), \ldots, (I_b, y_b))$, M = Model to Prune, $M_{new} = \phi$, Dataset D
Output: $M_{new} = \phi$
for $i \leftarrow 1$ **to** b **do**
  $//PerformForward - Pass(M, I_i, y_i)$
  for $l \leftarrow 1$ **to** L **do**
    for $n \leftarrow 1$ **to** N_l **do**
      $A_{n,l,i} = A_{n,l,i} +$
       $M.GetNormalisedActivation(n, l, i)$
    end
  end
end
for $i \leftarrow 1$ **to** b **do**
  $\bar{A}_{n,l} = \sum_{i=1}^{b} A_{n,l,i}/b$
end
for $l \leftarrow 1$ **to** L **do**
  $t_l = \sum_{n=1}^{N_l} \bar{A}_{n,l}/N_l$
end
for $l \leftarrow 1$ **to** L **do**
  for $n \leftarrow 1$ **to** N_l **do**
    if $t_l^{\gamma} \leq \bar{A}_{l,n}$ **then**
      $M_{new}.AddNeuron(n, l)$;
  end
end
FineTune M_{new} with the dataset D
return M_{new}

C. Hyper-parameter: γ

The hyper-parameter controls the threshold value. As the weights are normalized, the higher the value of γ lower the threshold which results in a fewer number of neurons being dropped. The optimum value of γ can be computed by grid-search or the *hyperband* algorithm[11]. Empirically, the best value of γ lies in the range $\in (0, 5]$

D. Behavior

The algorithm adapts to different architectures and datasets. Depending on the distribution of the mean activation values within a layer, the algorithm decides on the degree of pruning. For instance, if the activation values are skewed the algorithm will extract the neurons associated with the high mean values As these neurons have been persistently shown to be active throughout the forward pass inherently giving them more importance. If the distribution of mean activation values is nearly equal across the neurons within a layer it becomes difficult for the algorithm to conclude if some are significantly better than the other.

IV. Performance Metrics

In order to quantify the extent of pruning for each layer l, I define a metric $Prune/Original$ (abbr PO) as

$$Prune/Original_l = \frac{N_l^{'}}{N_l} \tag{1}$$

where $N_l^{'}$ represents the number of neuron's within the layer l. To quantify the pruning extent of the entire model, we can simply define it in terms of PO_l.

$$PruningFactor_M = \sum_{l=1}^{L} PO_l \cdot N_l / \sum_{l=1}^{L} N_l \tag{2}$$

To measure the performance gain we can simply define ΔG as

$$\Delta G = M_{new}.TestAccuracy() - M.TestAccuracy() \tag{3}$$

For an archetype, the $PruningFactor$ should be as low as possible with a positive ΔG

V. Experimental Results

I present the results of the selective pruning algorithm on the model architecture as defined in Fig 2 with the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000, 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class.

For implementation, I have used TensorFlow 2.3 framework [for result reproducibility GitHub] .I have trained the model for 500 epochs with a batch size of 128 employing Adagrad[12] for optimization, followed by persistent pruning and fine-tuning. The performance is depicted in Fig 6.

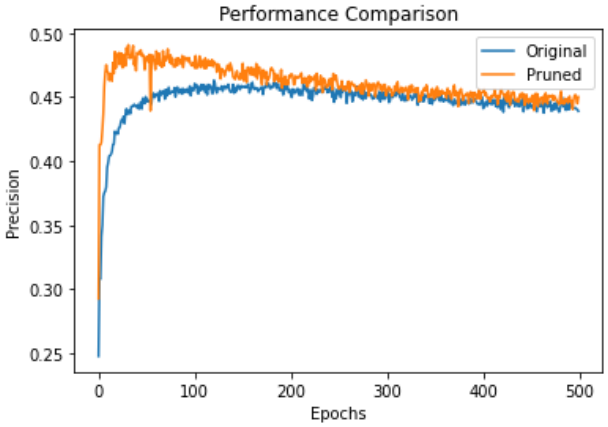


Fig. 6. Performance Comparison, $\gamma = 2$.

The extent of pruning is depicted in fig 7.

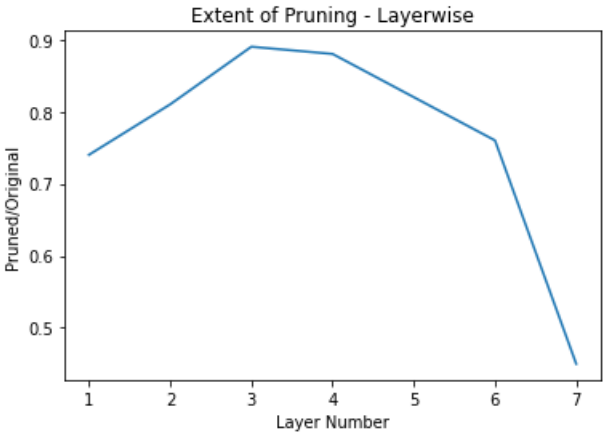


Fig. 7. Extent Of Pruning, $\gamma = 2$.

VI. Experiment Analysis

We can see that in Fig 6 there is a positive gain. For the later epochs, performance degradation can be attributed to over-fitting which can simply be avoided by using strategies such as early stopping. The accuracy is relatively low due to the absence of spatial cues as we have flattened the input in the initial layer for better CPU performance.

In Fig 7, the extent of pruning can be seen layer-wise. Layer 2 and Layer 3 have a higher PO than the remaining layers. As N_l is the same across all the layers we can simply calculate $PruningFactor$ by taking the mean values of all PO which gives us $PF \approx 0.76$

The value of hyper-parameter plays a vital role in the performance. As shown in Fig 8, for this value of γ we still have positive gain but the maximum accuracy achieved is not the same as shown in Fig 6.

Fig 8 represents the difference in peak performance when the model is fine-tuned instead of retraining from the scratch. No layers were frozen, thus, the increase in performance can be attributed to the initial weight values. We can conclude that finetuning produces a superior result.

To demonstrate how critical the value of γ is I select a different architecture as a 7 layer network is extremely powerful and can simulate complex functions even after substantial pruning. We can reduce the model potency by reducing its depth[10]. I run the pruning algorithm on a model with 100 neurons wide and 2 layers deep. For a low $PruningFactor$ the model performance should be capped. This behavior is indeed reflected in Fig 9, where I have deliberately fed a sub-optimal value of γ. Thus, we can consider γ in a way to be a trade-off variable.

VII. Pipeline Deployment

With the introduction of the ONNX[8] framework, a single model for a particular hardware configuration can be morphed into a different framework and hardware device. I now propose a pipeline architecture that can utilize the aforementioned pruning algorithm and simultaneously boost computational

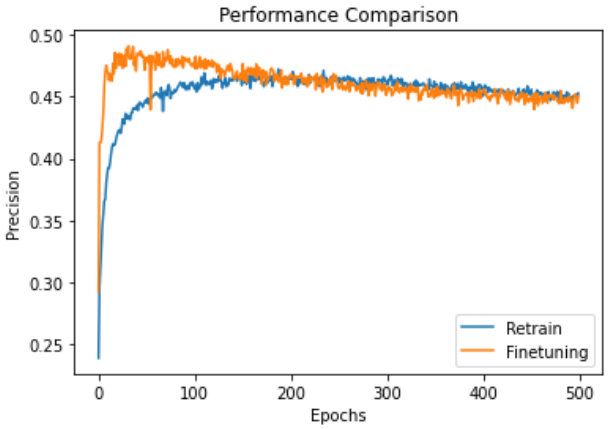
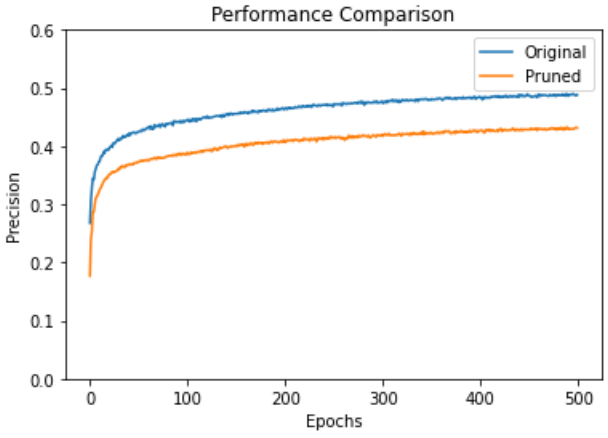


Fig. 8. Performance Difference between Retraining and Fine-tuning



. Fig. 9. Incorrect Value of $\gamma = 1, M_{[100\times2]}$

performance for reduced training, finetuning, and inference time. A pipeline receives the model and the dataset as input. It first converts the input model into its preferred framework using the ONNX API, in our case TensorFlow. It then prunes the model using the algorithm. Once the model is pruned, it no longer enjoys the computational accelerations which are limited to dense layers. Thus, if there is a method to transform a sparse network to a dense network without an increase in model complexity, the hardware accelerations can be utilized. However, this a fundamentally difficult problem and I currently exclude it from the pipeline and reserve it for future works. Finally, the model can be transformed into .onnx format and deployed at the target hardware where the ONNX Run-time environment can be used for inferencing which automatically optimizes depending on the underlying platform. The pipeline is summarised in Fig 10 and can be realized by a single click deployment.

VIII. Conclusion and Future work

In this paper, I present a novel self-adaptive pruning algorithm that improves performance and at the same time reduces the model size backed by empirical evidence.

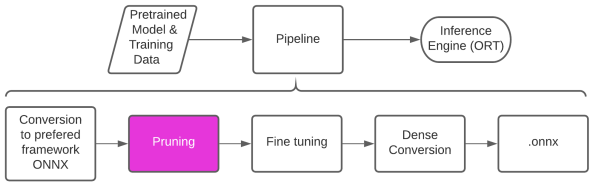


Fig. 10. Proposed Pipeline.

Future work involves collaboration for theoretical analysis of the proposed algorithm and searching solutions for the sparse-to-dense conversion problem or an alternate method. Also, apply the algorithm on complex models and datasets by gaining access to powerful hardware infrastructure.

References

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.
[2] "ImageNet", Image-net.org, 2020. [Online]. Available: http://www.image-net.org/. [Accessed: 12- Nov- 2020].
[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
[4] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. arXiv:1412.1441v2, 2015.
[5] Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In Proceedings of the 37th International Conference on Machine Learning, 2020.
[6] Bishwaranjan Bhattacharjee, John R Kender, Matthew Hill, Parijat Dube, Siyu Huo, Michael R Glass, Brian Belgodere, Sharath Pankanti, Noel Codella, and Patrick Watson. P2l: Predicting transfer learning for images and semantic relations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 760–761, 2020.
[7] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In NIPS, 2014.
[8] "ONNX — Home", Onnx.ai, 2020. [Online]. Available: https://onnx.ai/. [Accessed: 14- Dec- 2020].
[9] Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. The International Conference on Learning Representations, 2019b.
[10] Md Zahangir Alom, Theodore Josue, Md Nayim Rahman, Will Mitchell, Chris Yakopcic, and Tarek M Taha. 2018. Deep Versus Wide Convolutional Neural Networks for Object Recognition on Neuromorphic System. arXiv preprint arXiv:1802.02608 (2018).
[11] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: bandit-based configuration evaluation for hyperparameter optimization. In International Conference on Learning Representations, 2017.
[12] Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res., 12:2121–2159, July 2011. ISSN 1532-4435.