SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 10, 2021

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Drew Ligman

ENTITLED

Improved Hyperparameter Tuning for Graph Learning with Warm-Start Configuration

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Zhiqiang Tao
Zhiqiang Tao (Jun 10, 2021 15:41 PDT)

Thesis Advisor

Nam Ling
Nam Ling (Jun 10, 2021 15:50 PDT)

Department Chair

Improved Hyperparameter Tuning for Graph Learning with Warm-Start Configuration

by

Drew Ligman

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 10, 2021

Improved Hyperparameter Tuning for Graph Learning with Warm-Start Configuration

Drew Ligman

Department of Computer Science and Engineering
Santa Clara University
June 10, 2021

ABSTRACT

With the increasing size and complexity of machine learning datasets, obtaining highly performing prediction models in various tasks has become increasingly difficult. In particular, the processs of hyperparameter optimization (HPO) contributes a significant portion of this cost. This work examines a specific graph-machine learning model, graph convolutional networks (GCN), to derive a hyperparameter configuration with optimal performance across a variety of datasets. We motivate our configuration theoretically and validate it empirically through comprehensive experimentation. We find that for GCN semi-supervised classification tasks, our configuration performs nearly optimally when compared against traditional HPO while only requiring a fraction of the budget. We further propose using this configuration to warm-start subsequent HPO as a means of accelerating its convergence.

Table of Contents

List of Tables

Chapter 1

Introduction

In recent years, considerable effort has been made to apply machine learning to tasks involving arbitrarily-structured data like those of graphs. Most notably, [3] introduces graph convolutional networks (GCN), a fast and powerful layer-wise propagation rule for neural networks that generalizes convolutions to operate directly on graphs. In semi-supervised representation learning tasks like vertex classification, GCN offers significant performance increases over previous competing methods.

Despite the efficiency of GCN being best-in-class, the growing complexity of real-world datasets produces a multiplied increase in the complexity and cost of obtaining an accurate model. In particular, model selection, which involves the steps of model training and hyperparameter tuning, is acutely sensitive to this increase. As it stands currently, this model selection stage can consume weeks of compute resources and expert attention. Despite platforms like RayTune [4] offering distributed tuning and implementing efficient hyperparameter optimization (HPO) algorithms, the cost of performing HPO can still remain prohibitively expensive in sufficiently complex tasks.

In this work we derive a generalized hyperparameter configuration for GCN and show its performance to be nearly optimal across various datasets. In particular, we focus on the task of semi-supervised vertex-classification and consider the simultaneous performance over multiple datasets in our derivation. In contributing this, we hope to reduce the cost of hyperparameter tuning that currently prevents comprehensive HPO over massive datasets. That is, by using this configuration as a warm-start in subsequent tuning, convergence time is vastly accelerated. This practice of warm-starting HPO is preceded by similar work in literature [5]. Further, for those with especially limited compute budget, this warm-start can be used directly as a drop-in configuration to achieve nearly optimal performance even on unseen datasets.

Chapter 2

Methodology

2.1 Preliminary: On Graph Convolutional Networks

In motivating the development of graph machine learning models we introduce the problem of graph-based vertex-classification. That is, over a network in which only a small subset of nodes contain ground-truth labels, we wish to predict the labels for those remaining. This problem can be framed as a semi-supervised learning task in which label information is transformed through the network topology.

Various approaches have been presented in literature to solve this problem, but GCN [3] represents the most promising of these. The authors of [3] draw from the success of traditional convolutional neural networks (CNN) by deriving an approximate form for convolutions that operate directly on graph-structured data. For an in-depth derivation of this form and its relation to traditional CNNs, the reader is referred to [3]. In general, GCN allows for representation learning of many sorts, not just that of vertex-classification.

The layer-wise propagation rule of GCN is described in equation 2.1.

$$\boldsymbol{H}^{(l+1)} = \sigma(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}) \tag{2.1}$$

Here, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$ is the adjacency matrix with added self-loops. $\tilde{\boldsymbol{D}}$ is the diagonal degree matrix of the matrix $\tilde{\boldsymbol{A}}$, defined as $\boldsymbol{D}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. $\boldsymbol{W}^{(l)}$ is the matrix of trainable filter parameters for to the l-th layer. $\sigma(\cdot)$ can be any activation function such as ReLU or softmax. Finally, $\boldsymbol{H}^{(l)}$ is defined as the hidden feature matrix for the l-th layer. We note that values on the input layer are simply those provided from the initial node feature matrix: $\boldsymbol{H}^{(0)} = \boldsymbol{X}$.

Using this form, we can therefore implement a multi-layer GCN by repeated composition of the forward-passing rule. We set the activation on the output layer as softmax and use existing methods of stochastic gradient descent to train our model. In doing so, we can ultimately obtain a GCN model capable of performing semi-supervised vertex-classification.

2.2 Preliminary: On Hyperparameter Optimization

The validation accuracy of a machine learning model can be expressed as a function $\mathcal{P} : \mathcal{X} \to \mathbb{R}$ of its hyperparameters $\boldsymbol{x} \in \mathcal{X}$. Thus, the HPO problem can be defined as finding the configuration of hyperparameters $\boldsymbol{x}^* \in \mathcal{X}$ such that $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{P}(\boldsymbol{x})$. Further, $\mathcal{P}$ is considered to be both blackbox and expensive. That is, $\boldsymbol{x}^*$ cannot be obtained analytically leaving the only method of finding $\boldsymbol{x}^*$ as repeated evaluation of $\mathcal{P}$ for various $\boldsymbol{x} \in \mathcal{X}$. Since dimensions of $\mathcal{X}$ are allowed to be continuous, evaluating any finite number of configurations $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k\}$ is not always guaranteed to produce $\boldsymbol{x}^*$. Recalling that $\mathcal{P}$ is also expensive to evaluate implies k is prevented from being very large. Because of both of these limitations, it is often considered sufficient to approximate $\boldsymbol{x}^*$ with a configuration $\boldsymbol{\xi}^*$ such that $\mathcal{P}(\boldsymbol{\xi}^*) \approx \mathcal{P}(\boldsymbol{x}^*) = \max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{P}(\boldsymbol{x})$.

Because of the well-studied nature of the blackbox optimization and HPO problems, efficient algorithms to obtain $\boldsymbol{\xi}^*$ have been proposed. In particular, [1] proposes a HPO algorithm that combines the approaches of Beyesian optimization (BO) and Hyperband (HB) to create a novel method called BOHB. [1] performs in-depth benchmarking comparisons to other state-of-the-art methods and finds that BOHB consistently outperforms competing methods on the basis of final model performance for a set budget. Thus, BOHB offers a means of obtaining $\boldsymbol{\xi}^*$ for a given performance function $\mathcal{P}$ and search space $\mathcal{X}$.

2.3 On Maximizing Simultaneous GCN Model Performance

Objective Function. For a single dataset, $\mathcal{S}$, the performance function $\mathcal{P}_{\mathcal{S}}$ of a GCN can be defined in any number of ways with the most common choices being validation accuracy and validation loss. Note with the latter, the problem becomes one of minimization. However, since $\mathcal{P}$ can be defined arbitrarily, we are able to employ BOHB to solve a more general problem provided $\mathcal{P}$ remains well-defined. In our case, we consider many datasets $U = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ and wish to find a single configuration $\boldsymbol{\xi}$ that performs nearly optimally over all datasets. That is, to obtain $\boldsymbol{\xi}$ satisfying the criteria in (2.2).

$$\boldsymbol{\xi} \in \mathcal{X} : \forall \mathcal{S}_i \in U, \mathcal{P}_{\mathcal{S}_i}(\boldsymbol{\xi}) \approx \max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{P}_{\mathcal{S}_i}(\boldsymbol{x}) \tag{2.2}$$

This configuration $\boldsymbol{\xi}$ represents the best trade-offs between individual datasets' performances that maximizes the universal performance. Thus, finding $\boldsymbol{\xi}$ becomes a problem of simultaneously maximizing all performance functions $\mathcal{P}_{\mathcal{S}_1}, \mathcal{P}_{\mathcal{S}_2}, \ldots, \mathcal{P}_{\mathcal{S}_n}$. To quantify this universal performance we denote $\mathcal{P}_U$ to be a function that aggregates $\{\mathcal{P}_{\mathcal{S}_1}, \mathcal{P}_{\mathcal{S}_2}, \ldots, \mathcal{P}_{\mathcal{S}_n}\}$. Thus finding such a configuration satisfying the initial optimization criteria reduces to solving for $\boldsymbol{\xi} = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{P}_U(\boldsymbol{x})$. This takes the identical form of the HPO problem above; therefore, we can now explicitly apply BOHB using $\mathcal{P}_U$ to obtain $\boldsymbol{\xi}$.

Universal Metric. The definition of $\mathcal{P}_U$ deserves discussion since it will affect the resulting configuration $\boldsymbol{\xi}$. An initial approach would be to define $\mathcal{P}_U$ as the classification accuracy over the union of all validation samples of datasets

in U as in equation (2.3).

$$\mathcal{P}_U(\boldsymbol{x}) = \frac{\sum_{\mathcal{S}_i \in U} \#\,\text{correct predictions}}{\sum_{\mathcal{S}_i \in U} \#\,\text{validation samples}} \tag{2.3}$$

However, as shown in section 3.2, the size of datasets vary considerably. This method would unintentionally weight larger datasets (those with larger validation splits) more heavily in this summation. With sufficient variance in dataset size, we may observe that $\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{\xi}) \approx \max_{\boldsymbol{x} \in \mathcal{X}} \mathcal{P}_{\mathcal{S}_i}(\boldsymbol{x})$ only when $|\mathcal{S}_i|$ is large. Maximizing such a function would therefore fail to provide a configuration satisfying the desired criteria in (2.2). We can solve this problem by defining $\mathcal{P}_U$ as the arithmetic mean of the classification accuracies $\mathcal{P}_{\mathcal{S}_1}, \mathcal{P}_{\mathcal{S}_2}, \ldots, \mathcal{P}_{\mathcal{S}_n}$ as shown in equation (2.4).

$$\mathcal{P}_U(\boldsymbol{x}) = \overline{P_{\mathcal{S}_i}}(\boldsymbol{x}) = \frac{1}{|U|} \sum_{\mathcal{S}_i \in U} \mathcal{P}_{\mathcal{S}_i}(\boldsymbol{x}) \tag{2.4}$$

However, in doing so, we potentially invite another concern. Under such a formulation $\mathcal{P}_U$ weights the classification accuracy of all datasets with equal regard. Consider two datasets S_h, S_l whose maximum performance differ by a significant margin ($\max \mathcal{P}_{\mathcal{S}_h} \gg \max \mathcal{P}_{\mathcal{S}_l}$). By nature of their respective magnitudes, any constant performance change to $\mathcal{P}_{\mathcal{S}_l}$ is relatively more significant than the equal change to $\mathcal{P}_{\mathcal{S}_h}$. Thus, an appropriate formulation for $\mathcal{P}_U$ would be one which maximizes the classification accuracy of all datasets relative to their maximum performance. We do so by weighting each dataset by a factor inversely proportional to its maximal accuracy as shown in (2.5)

$$\mathcal{P}_U(\boldsymbol{x}) = \frac{1}{|U|} \sum_{\mathcal{S}_i \in U} \frac{\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{x})}{\max \mathcal{P}_{\mathcal{S}_i}} \tag{2.5}$$

We note that this form for $\mathcal{P}_U$ has the added, non-trivial prerequisite of knowing of the maximal performance $\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{x}^*)$ for all datasets $\mathcal{S}_i \in U$. With $\mathcal{P}_U$ defined as above, our desired configuration $\boldsymbol{\xi}$ can be obtained by directly applying the BOHB algorithm to find its maximal argument.

2.4 On Tuning GCN with Warm-Start Configuration

The hope for this warm-start configuration is universal performance extending beyond the datasets known to it from U. That is, for some dataset $S_j \notin U$, of which $\boldsymbol{\xi}$ has no former information, we still hope $\boldsymbol{\xi}$ can generalize to this unseen dataset and perform nearly optimally ($\mathcal{P}_{\mathcal{S}_j}(\boldsymbol{\xi}) \approx \max \mathcal{P}_{\mathcal{S}_j}$). This objective is motivated by the argument that for a sufficiently large and diverse U, an unseen dataset S_j is likely to have structural similarity to at least one $S_i \in U$. Knowing $\boldsymbol{\xi}$ performs well on S_i, we can expect a similarly high performance for $\boldsymbol{\xi}$ on S_j.

Thus the overarching objective for $\boldsymbol{\xi}$ is to be a single, highly performing configuration for GCN vertex-classification tasks regardless of dataset used. We therefore encourage this generalization for our final configuration by including a number of diverse datasets in our experimentation (see section 3.2). We likewise perform generalization testing in section 3.4 to verify this claim.

Chapter 3

Experiments

We perform a number of experiments for the purposes of obtaining the warm-start configuration (section 3.3) and measuring its performance (section 3.4). The implementation of GCN is done using the deep learning framework provided by PyTorch [6]. For the experiments involving a BOHB search, we directly use the implementation provided by RayTune [4].

3.1 Hyperparameter Search Space

We enumerate our hyperparameter search space in Table 3.1. We note the inclusion of one extra hyperparameter "dropedge" not originally included in [3]. We incorporate the DropEdge technique as described in [7] and thus include its configurable hyperparameter, the dropedge preservation rate, in our experimentation. We confirm the results of [7] in finding that the inclusion of DropEdge decisively promotes the performance of GCN.

We further note the irregular shape of this space. Since we allow the dropout rate to be configurable on a per-layer basis, the number of dropout variables increases with the number of layers in our network. Likewise, the residual mechanism in deep learning is only defined over layers whose shapes are identical. Therefore, the inclusion of this hyperparameter is conditioned on the number of layers being greater than 2.

Table 3.1: Hyperparameter Space

Hyperparameter	**Type**	**Range**	**Description**
nlayers	integral	$\{2..4\}$	number of hidden layers
nhidden	integral	$\{8..64\}$	length of hidden feature vectors
bias	categorical	$\{\text{True}, \text{False}\}$	include bias parameter
residual	categorical	$\{\text{True}, \text{False}\}_{\text{nlayers}>2}$	apply residual mechanism
weight decay	real	$[0, 1e{-}3]$	L2 regularization weight
learning rate	real	$[0, 0.1]$	learning rate
dropedge	real	$[0, 1]$	edge preserving rate
dropout	real vector	$[0.1, 0.9]^{\text{nlayers}-1}$	layerwise dropout rates

Table 3.2: Dataset Statistics

Partition	Dataset	Type	Nodes	Edges	Classes	Features	Label Rate
Tuning	Cora	Citation Network	2708	5278	7	1433	0.0517
	CiteSeer	Citation Network	3327	4552	6	3703	0.0361
	Actor	Wikipedia Graph	7600	26659	5	932	0.0132
	WikiCS	Wikipedia Graph	11701	221447	10	300	0.0170
	Computers	Product Network	13752	245861	10	767	0.0145
	Photo	Product Network	7650	119081	8	745	0.0209
Validation	PubMed	Citation Network	19717	44324	3	500	0.0030
	Coauthor CS	Co-Authorship Graph	18333	81894	15	6805	0.0164

3.2 Datasets

We use a total of eight datasets in our experiments, summarized in Table 3.2. Each dataset represents some real-world network previously studied in literature. Copies of these datasets and their splits is provided in a consistent format by the PyTorch Geometric library in [2]. Label rate denotes the ratio of nodes in the training set to total nodes in the network.

We additionally partition the total collection of datasets into tuning and validation sets. In doing so, we can obtain $\boldsymbol{\xi}$ by optimizing over the tuning split and used the withheld validation datasets for generalization testing.

3.3 Obtaining the Warm-Start

To obtain the warm-start configuration $\boldsymbol{\xi}$ as described in section 2.3, we perform a BOHB search over the hyperparameter space defined in section 3.1 to maximize $\mathcal{P}_U$ of the form described in equation (2.4). We argue that the relative similarity between maximum performance $\mathcal{P}_{S_i}(\boldsymbol{\xi}^*)$ between datasets shown in section 3.5 justifies using this definition in place of the more robust form in equation (2.5). Though this leaves room for a more optimal configuration, we deem this improvement negligible against the stochasticity inherent to model training. We select the set of datasets, U, for this experiment to be only those in the tuning split described in section 3.2; those part of the validation split are withheld.

We further configure RayTune's implementation of BOHB with the following: {num_samples $= 729$, reduction_factor $= \eta = 3$, max_t $= T\eta/(\eta - 1)$} This choice budget (max_t) for the experiment ensures that the longest trials run for exactly T epochs. The form of this expression derives from the geometric nature of Hyperband's successive trial reduction. We also explicitly set $T = 400$ as the maximum number of epochs since we find it to be a good estimate of the minimum necessary iterations for convergence. We note that GCN training occurs faster on some datasets, but our choice ensures sufficient iteration in the most difficult cases.

Thus, for each sample configuration $\boldsymbol{x} \in \mathcal{X}$ suggested by BOHB, we simultaneously train GCN models across

all tuning datasets to obtain $\mathcal{P}_{\mathcal{S}_1}(\boldsymbol{x}), \mathcal{P}_{\mathcal{S}_2}(\boldsymbol{x}), \ldots, \mathcal{P}_{\mathcal{S}_n}(\boldsymbol{x})$ for all $\mathcal{S}_i \in U$. We then aggregate and report the universal performance $\mathcal{P}_U(\boldsymbol{x})$ of that configuration for each epoch. After completing BOHB, we are left with the configuration $\boldsymbol{\xi}$ that maximizes $\mathcal{P}_U$ over $\mathcal{X}$. This final configuration is shown in section 3.5.

3.4 Measuring Warm-Start Performance

To verify that our obtained configuration $\boldsymbol{\xi}$ indeed satisfies the criteria described in Sections 2.3 and 2.4, we perform the following experiments: *(i)* model training with $\boldsymbol{\xi}$ across all datasets and *(ii)* comprehensive, independent HPO across all datasets. Thus, for each dataset $\mathcal{S}_i$ we are able to compare the performance $\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{\xi})$ from *(i)* with the maximum attainable performance $\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{\xi}^*)$ given by *(ii)*. In these experiments, we consider all datasets described in 3.2: both those in the tuning and validation splits are included.

(i). We perform direct model training using the configuration $\boldsymbol{\xi}$ previously obtained by the search described in section 3.3. Considering each dataset $\mathcal{S}_i$ individually, we conduct 32 repeated trials in which we allow GCN to train for $T = 400$ epochs. Finally, we record the highest test prediction accuracy over these trials as $\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{\xi})$. We repeat this process for all datasets in the tuning and validation splits.

(ii). We perform BOHB searchs nearly equivalent to the one described in section 3.3 with the sole change being to choice of performance function used. Instead of simultaneous evaluation, we now consider each dataset individually. In this way, we forgo the constraint of simultaneous maximization on $\mathcal{P}_U$ and instead measure the maximum attainable performance of $\mathcal{P}_{\mathcal{S}_i}$ considering solely that dataset. We configure BOHB identically to before $\{\text{num\_samples} = 729, \text{reduction\_factor} = \eta = 3, \text{max\_t} = T\eta/(\eta - 1)\}$ with $T = 400$ and record the test prediction accuracy of the highest performing trial as $\mathcal{P}_{\mathcal{S}_i}(\boldsymbol{\xi}^*)$. As before, we repeat this process for all datasets.

We recall that the validation datasets included in parts *(i)* and *(ii)* were previously withheld from the search that produced $\boldsymbol{\xi}$. Thus, we consider the performance over these validation datasets as a measure of $\boldsymbol{\xi}$'s ability to generalize to unseen datasets. These results are enumerated and compared in section 3.5.

3.5 Experimental Results

Warm-Start Configuration. We directly present the warm-start configuration $\boldsymbol{\xi}$ obtained by the BOHB search (see section 3.3) in Table 3.3. Of note is the inapplicability of the residual mechanism and there only being one dropout rate resulting from nlayers being 2.

Performance Measures. As described in section 3.4, we evaluate the warm-start configuration $\boldsymbol{\xi}$ on a variety of measures. For each dataset in column 1: column 2 shows the corresponding performance of our warm-start; column 3, the maximum attainable performance over that dataset; and column 4, the ratio between them. The values of columns 2 and 3 are obtained by the experiments (*i*) and (*ii*) from section 3.4 respectively.

Table 3.3: Warm-Start Configuration $\boldsymbol{\xi}$

Hyperparameter	**Value**
nlayers	2
nhidden	54
bias	True
residual	-
weight decay	8.218e−7
learning rate	7.52e−2
dropedge	0.8457
dropout	[0.2190]

Table 3.4: Dataset Performance Measures

Dataset	**Warm-Start Performance** $\mathcal{P}(\boldsymbol{\xi})$	**Maximum Attainable Performance** $\mathcal{P}(\boldsymbol{\xi}^*)$	**Relative Warm-Start Performance** $\mathcal{P}(\boldsymbol{\xi})/\mathcal{P}(\boldsymbol{\xi}^*)$
Cora	0.804	0.823	0.977
CiteSeer	0.725	0.725	0.943
Actor	0.752	0.759	0.991
WikiCS	0.750	0.752	0.997
Computers	0.820	0.834	0.983
Photo	0.911	0.915	0.996
PubMed	0.775	0.798	0.971
Coauthor CS	0.901	0.907	0.993

In all datasets tested we find that the performance of $\boldsymbol{\xi}$ nearly equals that of the maximum. That is, over all the datasets considered, $\boldsymbol{\xi}$ attains an average of 0.981 the performance of the optimal configuration. The generalization performance of our configuration is likewise demonstrated by the nearly optimal performance of $\boldsymbol{\xi}$ even over the validation datasets (PubMed, Coauthor CS) to which it had no introduction during tuning. We can thus expect $\boldsymbol{\xi}$ to perform optimally on other unseen datasets. Therefore, we consider the results of Table 3.4 as sufficient in satisfying the criteria set forth for $\boldsymbol{\xi}$ in section 2.3.

Chapter 4

Conclusion

4.1 Summary

We present a hyperparameter configuration for GCN vertex-classification tasks that we show generalizes across various datasets. By incorporating multiple diverse datasets in our derivation, we encourage the ability of this configuration to generalize to real-world datasets beyond merely those discussed in this work. Extensive experimentation shows that this configuration performs nearly optimally relative to the maximum performance attainable with a comprehensive HPO tuning process. Considering the order-of-magnitude cost increase to perform HPO when compared against simply performing model training with this configuration, the slight performance reduction (< 0.02) may be justified in many cases. In other cases where marginal performance is extremely important, this configuration can assume the role of a warm-start in accelerating the convergence of subsequent HPO.

4.2 Considerations

As this project differs from adjacent design work, the considerations likewise differ considerably: we discuss these presently.

Ethically, this project represents no immediate error; however, we note that the pervasiveness of predictive modeling can indeed invite moral wrongdoing. This has been thoroughly discussed recent years and is almost always attributable to laziness in implementation. With predictive modeling taking the place of human actors in a variety of functions, implicit trust is being given to these models. These models are fundamentally limited by the structure and content of the training data, leaving considerable room for error should this responsibility not be taken seriously. Especially in a "high-stakes" environment where inaccurate model predictions can feasibly result in suffering, we must tread extremely carefully in transferring our agency to these algorithms. We echo the recommendations of machine learning experts who continually stress the need for extensive testing, especially with respect to unintended bias.

We designed this project with the purpose of furthering the knowledge and abilities of the field of machine learning. In doing so, we subscribe to the idea that expanding the body of scientific knowledge is a noble pursuit with the ultimate

benefactor being that of humanity. We likewise agree with the hope that, when used properly, the results of this project and all engineering research will eventually manifest some increase in the standards of living for all of society.

We also warn of the environmental impact inherent to massive compute problems of which machine learning (and HPO especially) belong. Despite a pursuit for improvements in efficiency, these problems often yet require massive compute budgets. Since compute is nearly directly tied to energy use, we acknowledge the importance of this factor when performing studies in this field. As it stands, all experiments performed in this project require a relatively low budget (run on a single 6-core CPU on a timescale of hours), and further, our results explicitly encourage cost-reduction in future work. However, this consideration is one which we weight heavily in our decisions, and thus communicate it here.

Bibliography

[1] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale, 2018.

[2] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[4] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[5] Gaurav Mittal, Chang Liu, Nikolaos Karianakis, Victor Fragoso, Mei Chen, and Yun Fu. Hyperstar: Task-aware hyperparameters for deep networks, 2020.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[7] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification, 2020.

[8] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2021.