1) DATE OF PUBLICATION XX 0, 2024, DATE OF CURRENT VERSION XX 0, 2024.
Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Progress Estimation for End-to-End Training of Deep Learning Models with Online Data Preprocessing

Qifei Dong and Gang Luo
Department of Biomedical Informatics and Medical Education, University of Washington, Seattle, WA 98195, USA

Corresponding author: Gang Luo (luogang@uw.edu).

ABSTRACT Deep learning is the best machine learning algorithm for numerous analytical tasks. On a large data set, training a deep learning model frequently lasts several days to several months. Throughout this long period, it would be helpful to show a progress indicator, which continually projects the percentage of model training work accomplished as well as the outstanding model training time. We formerly invented the first method to support this function while allowing early stopping. This method assumes that the input data to the model have been preprocessed before model training starts. This is a limitation. In practice, online data preprocessing is often integrated into the model and done as part of the end-to-end model training. Ignoring online data preprocessing costs can cause our former method to produce inaccurate estimates. To overcome this limitation, this paper presents a new progress estimation method that explicitly considers online data preprocessing. We did a coding implementation of our new method in TensorFlow. Our tests unveil that for various deep learning models that integrate online data preprocessing and in comparison with our former method, our proposed new method produces more stable progress estimates for model training and on average lowers the error of the predicted outstanding model training time by 16.0%.

INDEX TERMS Deep learning, online data preprocessing, TensorFlow, progress indicator, model training

SYMBOL LIST

Symbol	Meaning
$\lfloor \rfloor$	Floor function.
$\lfloor \rceil$	Nearest integer function.
$\lceil \rceil$	Ceiling function.
b_{max}	Greatest number of batches that we allow in training the model.
B	Quantity of training instances that we handle in each batch.
c_0	Upon exiting the first raw point of validation, the cost of model training that we have spent ignoring the overhead that the progress indicator has incurred at the inserted points of validation to compute validation errors.
C	Upper limit of the cost of model training that we have spent upon exiting the 4th point of validation.
e_j	The model's validation error computed at the j-th raw point of validation.
g	Count of batches of model training that are done between two sequential raw points of validation.
K	Size of the sliding window of time that we use to compute the speed of model training.
m_e	Greatest number of epochs that we allow in training the model.
n_0	Number of points of validation inserted ahead of the first raw point of validation.
p	Patience.
P_1	Greatest permitted percentage rise in the cost of model training caused by the progress indicator between the model training start time and the time of exiting the first raw point of validation.
r	T-V cost ratio.
r_0	Starting learning rate that the exponential decay approach employs.
s_t	Latest speed of handling training instances.
s_v	Latest speed of handling validation instances.
U	Unit of work for handling the training instances.
v_{max}	Greatest number of raw points of validation that we allow in training the model.
V	Quantity of data instances the whole validation set contains.
V_{min}	Smallest quantity of data instances that the subset of the whole validation set employed at each inserted point of validation requires.

V' Fixed quantity of data instances that the subset of the whole validation set employed at each inserted point of validation contains.
W Unit of work for handling the validation instances.
δ min_delta.
ρ Constant that the exponential decay approach uses to decide the decay speed of the learning rate.

I. INTRODUCTION

Our former progress estimation method and its limitation

Deep learning is the best machine learning algorithm for numerous analytical tasks such as artificial intelligence art creation, text generation, and speech recognition [1]. Yet, on a large data set, training a deep learning model could last several days to several months [2]-[7] even if a cluster of tensor processing unit (TPU) or graphics processing unit (GPU) nodes is used. Throughout this long period, it would be helpful to display a progress indicator, which continually projects the percentage of model training work accomplished as well as the outstanding model training time (see Fig. 1). Providing this information can facilitate workload management and make model training more user friendly [8]-[10].

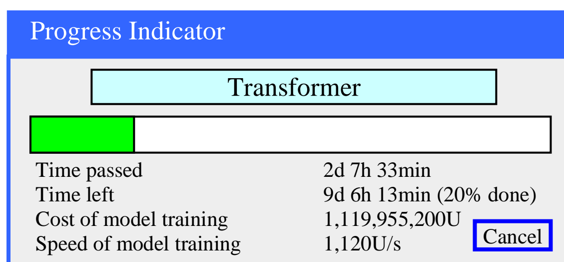


FIGURE 1. A progress indicator displayed in training a deep learning model.

We formerly invented the first progress estimation method that allows early stopping for training deep learning models [10], [11]. This method assumes that the input data to the deep learning model have been preprocessed before model training starts. In other words, we only do offline data preprocessing [12]. This is a limitation. In practice, part or all of the data preprocessing is often integrated into the model as online data preprocessing [12] and done as part of the end-to-end model training. Examples of online data preprocessing include rotating images, adjusting image contrast, adjusting image brightness, normalizing images, and embedding the tokens in textual documents. When online data preprocessing is used, it commonly takes a large percentage of model training time, e.g., 30% for an average deep learning job running in Google's data centers [13]. Ignoring online data preprocessing costs can cause our former progress estimation method to produce inaccurate estimates.

More specifically, in our former progress estimation method [10], [11], all operations in the deep learning model training job are assumed to be done on the same type of computer chips: central processing units (CPUs), GPUs, or TPUs. We define U, a unit of work, as the mean amount of work it requires to handle a training instance once during model training, which involves one forward and one backward propagation in the model. A validation instance is a data instance in the validation set. It can be shown that under the above assumption, the mean amount of work it requires to handle a validation instance once is $U/3$, which involves one forward propagation in the model. This result is the base for our former method to produce progress estimates. Yet, this result becomes invalid when online data preprocessing is used. In this case, handling a training instance once involves online data preprocessing as well as one forward and one backward propagation in the model. Handling a validation instance once typically involves both online data preprocessing and one forward propagation in the model. The online data preprocessing operation is often the same in handling either a training or a validation instance and can be done on CPUs. Forward and backward propagation in the model can be done on GPUs/TPUs. CPUs and GPUs/TPUs have vastly different processing speeds. This makes it difficult to define only one type of unit of work U, convert U to time in a uniform way for both the training and the validation instances, and produce good progress estimates.

To illustrate this point, we give a concrete example. During deep learning model training, we alternate between the training cycle and the validation cycle. In the training cycle, we handle the training instances and calculate changes to the model's parameter values. In the validation cycle, we handle the validation instances and calculate on the validation set the model's error rate. Each training and each validation cycle can take quite some time to run. For instance, when using one Nvidia Titan Xp GPU and the ImageNet-1k data set [14] to train the NASNet-A-Large model [15], it takes about 15 minutes to run one validation cycle [16]. When online data preprocessing is used and the model training job is the only job being executed in the system, the mean amount of time to handle a validation instance once can differ greatly from 1/3 of that to handle a training instance once. If we keep using $U/3$ as the mean amount of work it requires to handle a validation instance once, the speed of model training measured during the validation cycle can differ greatly from that measured during the training cycle. Consequently, during a typical validation cycle, the projected outstanding model training time can differ greatly from the genuine outstanding model training time.

Our contributions

To address our former progress estimation method's [10], [11] limitation, we come up with a novel progress estimation method for end-to-end training of deep learning models with online data preprocessing. In our new method, we define two types of unit of work, one for handling the training instances and the other for handling the validation instances. For handling the training instances, we use the type of unit of work for it to compute its speed and estimate its cost and outstanding

time. Handling the validation instances is done in a similar way. The outstanding model training time is = the outstanding time for handling the training instances + that for handling the validation instances.

We need to overcome two technical difficulties to complete the remaining parts of our new progress estimation method. First, when online data preprocessing is used, end-to-end model training is often done using asynchronous pipelining. There, multiple batches of training/validation instances appear at different stages of the processing pipeline simultaneously. This makes it non-trivial to gauge both the latest speed of handling training instances and that of handling validation instances. To address this problem, we enumerate all possible cases that ≥2 batches of training/validation instances appear at distinct stages of the pipeline simultaneously. For each such case, we design a distinct speed estimation approach tailored to it.

Second, our former progress estimation method [10], [11] inserts additional points of validation between the raw points of validation to more rapidly acquire decently good progress estimates. This insertion is controlled by several parameters, two of which are n_0 and V'. Our former progress estimation method sets the ratio of the mean amount of work required to handle a training instance once to that to handle a validation instance once to 3. This ratio is used to compute n_0 and V'. But when online data preprocessing is used, this ratio often differs greatly from 3 and is hard to compute. To address this problem, we show that regardless of this ratio's actual value, we can keep using a ratio of 3 to compute n_0 and V' without incurring any major performance problem for progress estimation.

We did a coding implementation of our new progress estimation method in the open-source software package TensorFlow [17]. We report our experimental results of training a transformer-based model, a convolutional neural network, and a recurrent neural network that all integrate online data preprocessing. Our results unveil that in comparison with our former progress estimation method [11], our proposed new method produces more stable progress estimates for model training and on average lowers the error of the predicted outstanding model training time by 16.0%.

Paper structure

The remainder of this paper has the following structure. Section II recaps our former progress estimation method. Section III reviews online data preprocessing. Section IV presents our new progress estimation method. Section V gives the experimental results. Section VI goes over the related work. Section VII lists some possible directions to do future work. Section VIII concludes this paper.

II. RECAP OF OUR FORMER PROGRESS ESTIMATION METHOD

This section first presents some concepts and notations the remainder of the paper will use, and then gives a summary of our former progress estimation method [10], [11]. In the remainder of this paper, wherever we mention GPUs, the same also applies to TPUs.

A. SOME CONCEPTS AND NOTATIONS

The user training the deep learning model sets 3 positive integers g, B, and m_e and an early stopping condition. During model training, all training instances are handled for one or more rounds termed epochs. We train the model in batches. In every batch, we handle B training instances and compute changes to the model's parameter values. Whenever g batches of model training are done, we arrive at a raw point of validation. There, we handle the validation instances and calculate on the whole validation set the model's error rate termed the validation error. We then evaluate whether the early stopping condition is satisfied. If so, we are done with model training. m_e is the greatest number of epochs that we allow in training the model. If upon finishing the m_e-th epoch, we still have not fulfilled the early stopping condition, we force model training to end. Accordingly, the greatest number of batches that we allow in training the model is

$$b_{max} = m_e \times \text{the number of data instances the training set contains} / B.$$

The greatest number of raw points of validation that we allow in training the model is

$$v_{max} = \lfloor b_{max} / g \rfloor.$$

$\lfloor\rfloor$ is the floor function, e.g., $\lfloor 5.7 \rfloor = 5$.

Like our prior work [10], [11], this work does not aim to handle every existing early stopping condition. Rather, we attend to a widely adopted early stopping condition [1], [18]. We do a case study on it to show that by explicitly addressing online data preprocessing, we can obtain better progress estimates for end-to-end training of deep learning models. Using a preset positive number called the patience p and a preset nonnegative number called the min_delta δ, this condition is satisfied at the first place where the validation error decreases by $< \delta$ for p sequential raw points of validation. That is, we stop model training at the k-th raw point of validation if we have $e_{k-p} - e_i < \delta$ for every i from $k-p+1$ to k. Here, e_j stands for the model's validation error that we compute at the j-th raw point of validation.

B. SUMMARY OF OUR FORMER PROGRESS ESTIMATION METHOD

This section gives a summary of our former progress estimation method for training deep learning models [10], [11]. The forecasted cost of model training is measured in Us. Each unit of work U is defined as the mean amount of work it requires to handle a training instance once during model training, which involves forward and backward propagation in the model in the absence of online data preprocessing. We start with a typically inaccurate guess of the cost of model training. During model training, we regularly collect statistics and use

them to revise the progress estimates for it. We keep computing the latest speed of model training = the quantity of Us finished per second in the previous $K = 10$ seconds. Each time we arrive at a point of validation, we use the information obtained at this and the prior points of validation to recompute the forecasted cost of model training. We keep predicting the outstanding model training time = the forecasted cost of model training left / the latest speed of model training. Every several seconds, the progress indicator is refreshed with the newest estimates. As model training continues, we keep gathering more accurate statistics of it and tend to obtain increasingly better progress estimates.

The raw points of validation could be sparse, causing a long delay to gather information at enough points of validation and obtain decently good progress estimates. To address this issue, we carefully insert additional points of validation between the raw points of validation. To reduce the progress estimation overhead, at each inserted point of validation, we evaluate the model's error rate, i.e., the validation error, on a subset randomly sampled from the whole validation set. In both the above paragraph and the remainder of this paper, wherever we speak of points of validation, we always refer to both inserted and raw points of validation unless we explicitly mention inserted or raw points of validation.

In the following, we review some details of how our former progress estimation method forecasts the cost of model training and inserts additional points of validation between the raw points of validation. These details are needed later to describe our new progress estimation method. We refer the reader to our prior papers [10], [11] for the other details of our former progress estimation method.

2) FORECASTING THE COST OF MODEL TRAINING

The cost of model training is roughly = the total cost of handling the training instances + the total cost of handling the validation instances. The total cost of handling the training instances is

= the mean amount of work it requires to handle a training instance once × the quantity of training instances that we handle in each batch × the quantity of batches it takes to train the model
= B × the quantity of batches it takes to train the model.

The total cost of handling the validation instances is

= the cost of handling the validation instances at the raw points of validation + the cost of handling the validation instances at the inserted points of validation.

We define the training-validation (T-V) cost ratio r = the mean amount of work it requires to handle a training instance once / the mean amount of work it requires to handle a validation instance once. As the numerator is 1 U, the denominator is

= U / r.

Let V stand for the quantity of data instances the whole validation set contains. The cost of handling the validation instances at the raw points of validation is

= the mean amount of work it requires to handle a validation instance once × the quantity of data instances the whole validation set contains × the quantity of raw points of validation it takes to train the model
= V / r × the quantity of raw points of validation it takes to train the model.

At each inserted point of validation, we employ a subset randomly sampled from the whole validation set. Let V' stand for the fixed count of data instances this subset contains. The cost of handling the validation instances at the inserted points of validation is

= V' / r × the number of inserted points of validation it takes to train the model.

When training the deep learning model in the absence of online data preprocessing, most of the training cost is spent on doing multiplication operations. We handle a training instance once by doing one forward and one backward propagation in the model. We handle a validation instance once by doing one forward propagation in the model. It takes about two times the quantity of multiplication operations to do one backward propagation than to do one forward propagation. Accordingly, we set the T-V cost ratio to 3.

In forecasting the cost of model training, the key is to project the quantity of raw points of validation it takes to train the model. We use the information obtained at the points of validation, maximum likelihood estimation, and Monte Carlo simulation to project this number. Unless early stopping occurs earlier, we can refine our initial and typically inaccurate estimate of this number for the first time only after we have obtained information from 4 points of validation.

3) INSERTING ADDITIONAL POINTS OF VALIDATION BETWEEN THE RAW POINTS OF VALIDATION

We use several parameters to control how we insert additional points of validation between the raw points of validation. In this section, we review how we set two of these parameters that are also used in describing our new progress estimation method.

Setting n_0

The first parameter to set is n_0, the number of points of validation to be inserted ahead of the first raw point of validation. When setting n_0, we attempt to meet two requirements if possible:

1) **Requirement 1**: Upon exiting the 4th point of validation, we have spent a cost of model training of $\leq C$ Us. C is a prechosen number whose default value is the quantity of CPUs or GPUs employed to train the model × 20,000. We adopt Requirement 1 to limit the amount of elapsed time before we wrap up at the 4th point of validation to refine

our initially guessed cost of model training for the first time.

2) **Requirement 2**: From the model training start time to the time of exiting the first raw point of validation, we incur a cost of $\leq c_0P_1$ to compute validation errors at the inserted points of validation. P_1 is a prechosen percentage with a default value of 5%. c_0 stands for upon exiting the first raw point of validation, the cost of model training that we have spent ignoring the overhead that the progress indicator has brought to compute validation errors at the inserted points of validation. We adopt Requirement 2 to limit this overhead.

As it is not always possible to fully meet both requirements, we treat them as soft requirements.

Setting V′

The second parameter to set is V', the fixed quantity of data instances that the subset of the whole validation set employed at each inserted point of validation contains. V' has to be $\leq V$, the number of data instances the whole validation set contains. To make one approximation used in our former progress estimation method accurate, we require V' to be $\geq$ a threshold V_{min}. Recall that r stands for the T-V cost ratio. When setting V', we attempt to meet the aforementioned Requirement 2 and set $V' = \min(\max(\lfloor rc_0P_1 / n_0 \rceil, V_{min}), V)$.

III. ONLINE DATA PREPROCESSING

In this section, we review online data preprocessing. In offline data preprocessing, the raw data are preprocessed and written to disk before we start training the deep learning model. The preprocessed data are usually as large as or several times larger than the raw data. During model training, the preprocessed data are read from disk and inputted to the model. When the raw data set (e.g., the 18-terabyte Open Images data set [19]-[21]) is large, writing the preprocessed data to and reading them from disk would incur high costs. To address this issue, one can do online data preprocessing. There, the raw data are preprocessed and then directly inputted to the model without being written to disk. No disk input/output is needed for handling the preprocessed data. Major deep learning software packages such as TensorFlow [17] and PyTorch [22] all support online data preprocessing.

Online data preprocessing can include one or more steps. Forward and backward (if any) propagation in the deep learning model is another step. One can do all these steps for each batch of data instances one by one. For instance, given a batch of training instances, we first normalize all training instance in it and then do forward and backward propagation for them in the model. After we finish handling one batch of training instances, we start handling the next batch. Alternatively, one can use asynchronous pipelining (see Fig. 2), a common approach to improve parallelism [23]. There, after a step is completed for a batch of data instances, we start this step for the next batch once the previous step is completed for the next batch.

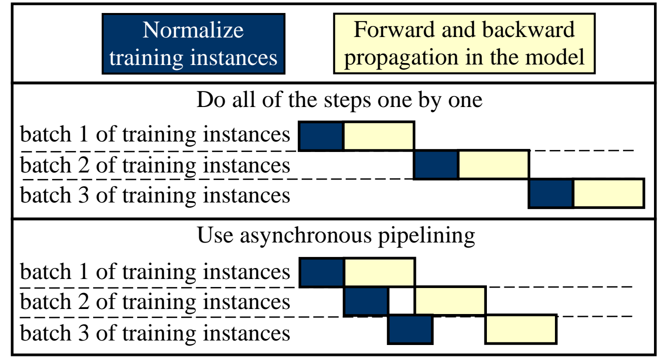


FIGURE 2. An example of handling training instances by doing all of the steps one by one vs. using asynchronous pipelining.

Online data preprocessing can be done on CPUs, GPUs, or a combination of both [24], [25]. In the last case, some online data preprocessing steps are done on CPUs, whereas the other online data preprocessing steps are done on GPUs.

IV. OUR NEW PROGRESS ESTIMATION METHOD

This section presents our new progress estimation method for end-to-end training of deep learning models with online data preprocessing. As in our prior paper [11], our presentation focuses on deep learning classification. Section IV-A shows how to predict the outstanding model training time. Section IV-B explains why we can keep using a T-V cost ratio of 3 to compute n_0 and V', two parameters used to control how we insert additional points of validation between the raw points of validation. Section IV-C presents the other changes made to our former progress estimation method [10], [11].

A. PREDICTING THE OUTSTANDING MODEL TRAINING TIME

1) OVERALL APPROACH

To address the limitation of our former progress estimation method [10], [11] when online data preprocessing is used, we separately estimate the progress of handling the training instances and the progress of handling the validation instances. Then we merge these estimates to obtain the overall progress estimates of model training.

More specifically, we use the same approach in our former progress estimation method [10], [11] to estimate the quantity of training instances and the quantity of validation instances needing to be handled to train the model. We define two types of unit of work:

(1) U for handling the training instances. As in our former progress estimation method [10], [11], each U is the mean amount of work it requires to handle a training instance once during model training.

(2) W for handling the validation instances. Each W is the mean amount of work it requires to handle a validation instance once during model training.

For handling the training instances, we use U to compute its speed and estimate its cost and outstanding time. Let s_t stand

for the latest speed of handling training instances measured by the quantity of *U*s finished per second. The predicted outstanding time for handling the training instances

= the forecasted remaining cost of handling the training instances / s_t

= the forecasted quantity of training instances that remain to be handled to train the model counting multiplicity / s_t.

For handling the validation instances, we use *W* to compute its speed and estimate its cost and outstanding time. Let s_v stand for the latest speed of handling validation instances measured by the quantity of *W*s finished per second. The predicted outstanding time for handling the validation instances

= the forecasted remaining cost of handling the validation instances / s_v

= the forecasted quantity of validation instances that remain to be handled to train the model counting multiplicity / s_v.

At any time, the predicted outstanding model training time

= the predicted outstanding time for handling the training instances + the predicted outstanding time for handling the validation instances.

Online data preprocessing can include applying data augmentation such as randomly flipping images to training instances. In this case, usually in each epoch, one augmented training instance is produced from every raw training instance. Only the augmented training instance is used to do forward and backward propagation in the deep learning model. After the forward and backward propagation is done for the augmented training instance, we count that one training instance has been handled.

Some data augmentation methods like CutMix [26] and MixUp [27] take multiple raw training instances as input to produce an augmented training instance. For instance, CutMix replaces a region in an image with a patch from another image. When such a data augmentation method is used, in each epoch, every raw training instance serves as the base and is combined with some other raw training instances to produce an augmented training instance exactly once. Only the augmented training instance is used to do forward and backward propagation in the deep learning model. After the forward and backward propagation is done for the augmented training instance, we count that one training instance has been handled.

Ideally, we should compute the latest speed of handling training instances s_t and the latest speed of handling validation instances s_v once every $K = 10$ seconds. When only training but no validation instances were handled in the previous *K* seconds, we compute s_t as the quantity of *U*s finished per second in the previous *K* seconds. When only validation but no training instances were handled in the previous *K* seconds, we compute s_v as the quantity of *W*s finished per second in the previous *K* seconds. In addition, we need to handle the following 3 cases:

1) Estimate s_t when no training instance was handled in the previous *K* seconds.
2) Estimate s_v when no validation instance was handled in the previous *K* seconds.
3) Estimate s_t and s_v when both training and validation instances were handled in the previous *K* seconds.

In the following, we discuss these 3 cases one by one.

2) ESTIMATING s_t WHEN NO TRAINING INSTANCE WAS HANDLED IN THE PREVIOUS *K* SECONDS

When no training but only validation instances were handled in the previous *K* seconds, we use the most recently estimated speed of handling training instances as the estimated latest speed of handling training instances. Taking this approximation will not greatly lower the accuracy of the predicted outstanding model training time. Typically, the training set is much larger than the validation set. For example, the ImageNet-1k data set contains ~1.3 million training instances and 50,000 validation instances [14]. During model training, we need to handle many more training instances than validation instances. Handling a training instance once takes more work than handling a validation instance once, as the former involves one forward and one backward propagation in the model, whereas the latter involves one forward propagation in the model. Due to these two factors, the time taken to handle the validation instances is much less than that taken to handle the training instances. In other words, the former is only a small fraction of the model training time. Only when we are handling the validation instances at a point of validation, the approximation used to estimate the latest speed of handling training instances will lead to estimation error in the predicted outstanding time for handling the training instances. But this will not last long. After a relatively short amount of time, we will finish handling the validation instances at the point of validation and move on to handling the training instances. At that time, we will recompute the correct latest speed of handling training instances.

3) ESTIMATING s_v WHEN NO VALIDATION INSTANCE WAS HANDLED IN THE PREVIOUS *K* SECONDS

When no validation but only training instances were handled in the previous *K* seconds, we use the most recently estimated speed of handling validation instances as the estimated latest speed of handling validation instances. This is an approximation that will not greatly lower the accuracy of the predicted outstanding model training time. As explained above, the time taken to handle the validation instances is only a small fraction of the model training time. The approximation will lead to estimation error in the predicted outstanding time for handling the validation instances. Yet, this estimation error will have only a small impact on the predicted outstanding model training time. When we predict that model training still needs quite some time to finish, the impact is by a small percentage. When we predict that model training is close to finish, the impact is by a small number.

By default, model training begins with handling the training instances. Before reaching the first point of validation, no

estimated speed of handling validation instances is available, making it impossible to estimate the outstanding time for handling the validation instances. To address this issue, when model training begins, we first randomly sample validation instances with replacement to obtain 5 batches of validation instances. Then we handle them to compute an initial estimated speed of handling validation instances. We set the number of batches to 5 to strike a balance between obtaining a relatively well estimated speed of handling validation instances and reducing the progress estimation overhead. When online data preprocessing is used, end-to-end model training is often done using asynchronous pipelining. When computing the initial estimated speed of handling validation instances in this case, we start the timer when the first batch of validation instances all exits the pipeline and do not count this batch. In this way, the latency resulting from initially filling in the pipeline would not negatively impact the precision of this speed computation.

4) ESTIMATING s_t AND s_v WHEN BOTH TRAINING AND VALIDATION INSTANCES WERE HANDLED IN THE PREVIOUS K SECONDS

Recall that s_t stands for the latest speed of handling training instances. s_v stands for the latest speed of handling validation instances. This section describes our approach to estimate s_t and s_v when both training and validation instances were handled in the previous K seconds. We first give an overview of our approach. Then we add some details needed in our approach.

Overview of our speed estimation approach

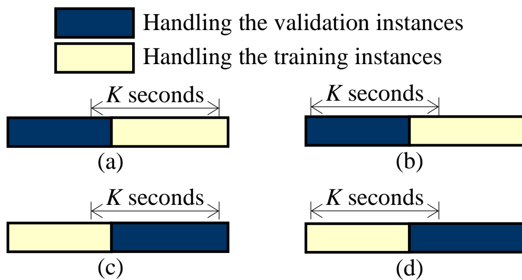


FIGURE 3. The 4 distinct scenarios in which only a few training or validation instances were handled in the previous K seconds.

When training the deep learning model, we alternate between handling training instances and handling validation instances. When we run into the case that both training and validation instances were handled in the previous K seconds, if only a few training (or validation) instances were handled in these K seconds (see Fig. 3), it can be hard to use these instances to well estimate the latest speed of handling training (or validation) instances.

To address this issue, we use a special speed estimation approach, in which we intentionally use no new parameter that needs to be set to an ad hoc number. We describe our approach mainly for the case of switching from handling validation instances to handling training instances in the previous K seconds (see Fig. 3(a) and 3(b)). The case of switching from handling training instances to handling validation instances in the previous K seconds (see Fig. 3(c) and 3(d)) can be handled similarly. Our approach includes two steps:

1) **Step 1:** As shown in Fig. 4, there are two possible cases:
   a. **Case 1:** We reached the most recent point of validation over K seconds ago (see Fig. 4(a)). In this case, the most recently estimated speed of handling validation instances was computed based on a K-second time window, in which only validation instances were handled. We use that estimated speed as the estimated latest speed of handling validation instances. As that estimated speed was computed only K seconds ago, it is usually a good estimate of the latest speed of handling validation instances.
   b. **Case 2:** We reached the most recent point of validation K seconds ago (see Fig. 4(b)). In this case, we estimate the latest speed of handling validation instances
      = the quantity of validation instances handled in the previous K seconds / the time spent on handling these validation instances
      = the quantity of validation instances handled in the previous K seconds / (the time of exiting the most recent point of validation – the starting time of K seconds ago).
2) **Step 2:** A timer is adopted to time the sliding time window employed to compute the speed of handling data instances. When we switch from handling validation instances to handling training instances, we restart the timer (see Fig. 4). This ensures that no other possible case needs to be considered in Step 1.

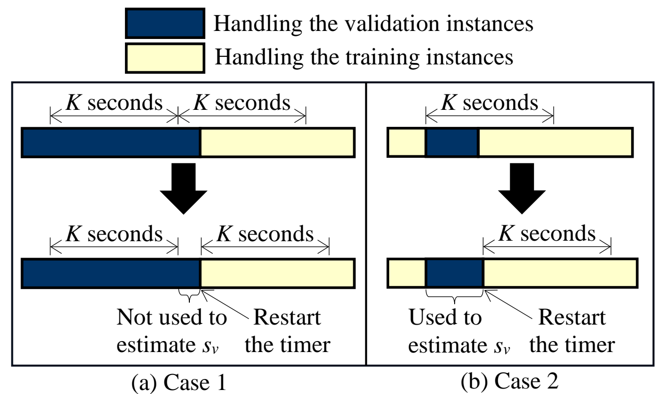


FIGURE 4. The two possible cases of switching from handling validation instances to handling training instances in the previous K seconds.

Additional details of Step 2 for handling the case of switching from handling validation instances to handling training instances in the previous K seconds

In this case, in Step 2, we do not restart the timer immediately upon switching from the previous round of handling validation instances to the current round of handling

training instances. When online data preprocessing is used, end-to-end model training is often done using asynchronous pipelining. In this case, the first training instance in the current round enters the pipeline before the last validation instance in the previous round exits the pipeline. To let estimating the latest speed of handling training instances not impacted by the last few validation instances in the previous round in the pipeline, we do not restart the timer until the first batch of training instances in the current round all exits the pipeline. When computing the latest speed of handling training instances, this batch of training instances is not counted.

To make speed computation doable, we need to ensure that ≥2 batches of training instances are handled between any two sequential points of validation. For this purpose, we only need to fulfill the requirement that ≥2 batches of model training are done before the first inserted point of validation, as our former progress estimation method [11] inserts more points of validation ahead of the first raw point of validation than between any two sequential raw points of validation. Recall that n_0 stands for the number of points of validation to be inserted ahead of the first raw point of validation. g stands for the count of batches of model training that are done between two sequential raw points of validation. To fulfill the requirement, we ensure that n_0 is $\leq \lfloor g / 2 - 1 \rfloor$. We do this by reusing the n_0 computed in our former progress estimation method [11] unless that computed number is $> \lfloor g / 2 - 1 \rfloor$, in which case we set n_0 to $\lfloor g / 2 - 1 \rfloor$.

Additional details of Step 2 for handling the case of switching from handling training instances to handling validation instances in the previous K seconds

In this case, in Step 2, we do not restart the timer immediately upon switching from the previous round of handling training instances to the current round of handling validation instances. Instead, we restart the timer when the first batch of validation instances in the current round all exits the pipeline. When computing the latest speed of handling validation instances, this batch of validation instances is not counted.

To make speed computation doable, we need to ensure that ≥2 batches of validation instances are handled at each point of validation. For this purpose, we only need to fulfill the requirement that ≥2 batches of validation instances are handled at each inserted point of validation. Recall that at each inserted point of validation, we use a subset of the whole validation set. V_{min} stands for the smallest count of validation instances demanded in this subset. To fulfill the requirement, we ensure that V_{min} is $\geq 2 \times$ the number of validation instances handled in each batch. That is, we set V_{min} to max(the V_{min} computed in our former progress estimation method [11], 2 × the number of validation instances handled in each batch).

B. WHY WE CAN KEEP USING A T-V COST RATIO OF 3 TO COMPUTE n_0 AND V'

Recall that the T-V cost ratio is = the mean amount of work it requires to handle a training instance once / the mean amount of work it requires to handle a validation instance once. We use the parameters n_0 and V' to control how we insert additional points of validation between the raw points of validation. In our former progress estimation method [10], [11], we set the T-V cost ratio to 3 and compute n_0 and V' accordingly. But when online data preprocessing is used, the actual T-V cost ratio can differ greatly from 3. In our new progress estimation method, we keep using a T-V cost ratio of 3 to compute n_0 and V'. This can cause two issues if the actual T-V cost ratio is < 3:

1) A longer time is needed before we can refine our initially guessed cost of model training for the first time.
2) The progress indicator incurs a higher runtime overhead.

Neither issue is a major one. In the following, for each of these two issues, we explain why it can occur but is not a major one.

1) A LONGER TIME IS NEEDED BEFORE WE CAN REFINE OUR INITIALLY GUESSED COST OF MODEL TRAINING FOR THE FIRST TIME

Recall that we can refine our initially guessed cost of model training for the first time only after we have obtained information from 4 points of validation. Upon exiting the 4th point of validation, we strive to incur a cost of model training of ≤ C Us (see Requirement 1 in Section II-B.2). When the actual T-V cost ratio is < 3 but we keep using a T-V cost ratio of 3, we underestimate the mean amount of work it requires to handle a validation instance once and subsequently the cost of handling validation instances at each point of validation. Thus, upon exiting the 4th point of validation, we could have spent a cost of model training of > C Us. This leads to the issue that a longer time is needed before we can refine our initially guessed cost of model training for the first time. This issue is not a major one because the T-V cost ratio has a lower bound of one, limiting the extent to which we underestimate the cost of model training that we would have spent upon exiting the 4th point of validation.

More specifically, when online data preprocessing is used, handling a training instance once involves online data preprocessing as well as one forward and one backward propagation in the model. Handling a validation instance once typically involves both online data preprocessing and one forward propagation in the model. The online data preprocessing steps for a training instance are often the same as those for a validation instance, but could include additional steps such as adjusting image contrast to add noise. Thus, the mean cost of doing online data preprocessing for a training instance is ≥ that for a validation instance. The T-V cost ratio

= (the mean cost of doing online data preprocessing for a training instance + the mean cost of doing one forward propagation in the model + the mean cost of doing one backpropagation in the model) / (the mean cost of doing online data preprocessing for a validation instance + the

mean cost of doing one forward propagation in the model)
> 1.

The T-V cost ratio has a lower bound of one. Thus, when we use a T-V cost ratio of 3, the actual cost of handling validation instances at each point of validation is $< 3 \times$ our estimated cost of doing that. Upon exiting the 4th point of validation, we have handled both training and validation instances with an expected cost of $\leq C$ Us (see Requirement 1 in Section II-B.2). The actual cost of model training that we have spent is usually $< 3 \times$ that cost, i.e., $3C$ Us. As is the case with C, $3C$ is a moderate number. Hence, the elapsed time before we wrap up at the 4th point of validation to refine our initially guessed cost of model training for the first time is relatively short, even if it is longer than what we initially expected.

2) THE PROGRESS INDICATOR INCURS A HIGHER RUNTIME OVERHEAD

When the actual T-V cost ratio is < 3 but we keep using a T-V cost ratio of 3, the actual cost of handling validation instances at each inserted point of validation is larger than our estimated cost of doing that. This leads to the issue that the progress indicator incurs a higher runtime overhead than what we initially expected.

This issue is not a major one. As explained before, when we use a T-V cost ratio of 3, the actual cost of handling validation instances at each inserted point of validation is $< 3 \times$ our estimated cost of doing that. As reviewed in Section II-B.2, we set P_I to 5% as the greatest permitted percentage rise in the cost of model training caused by the progress indicator between the model training start time and the time of exiting the first raw point of validation. In the worst case, the progress indicator incurs a rise of $< 3 \times 5\% = 15\%$ in the cost of model training. In practice, the actual rise is usually much less than 15%. For example, according to the computation done in our prior paper [11], in the case that at most 50 raw points of validation are allowed in training the model and model training ends at the 20th raw point of validation, we expect the progress indicator to incur a rise of ~1.2% in the cost of model training. The actual rise is $< 3.7\%$.

C. OTHER CHANGES MADE TO OUR FORMER PROGRESS ESTIMATION METHOD

In this section, we present the other two changes made to our former progress estimation method [11]. In Section IV-C.1, we show how to set the parameter C. In Section IV-C.2, we discuss how to display the progress estimates.

1) SETTING THE PARAMETER C

Upon exiting the 4th point of validation, we hope to have spent a cost of model training of $\leq C$ Us (see Requirement 1 in Section II-B.2). This helps limit the elapsed time before we can refine our initially guessed cost of model training for the first time. In our former progress estimation method [11], we assume that all operations in the deep learning model training job are done on either CPUs or GPUs, but not both. C is set to be the quantity of CPUs or GPUs employed to train the model $\times$ 20,000. When online data preprocessing is used, the end-to-end model training job can be done on a combination of CPUs and GPUs. In this case, we set C to be the total number of CPUs and GPUs employed to train the model $\times$ 20,000.

2) DISPLAYING THE PROGRESS ESTIMATES

In our former progress estimation method [11], at any time, we display one cost of model training and one speed of model training. In our new progress estimation method, we have one set of progress estimates for handling the training instances and another set of progress estimates for handling the validation instances. We display certain progress estimates in a different way from that in our former progress estimation method.

Costs

We display two costs, one of handling the training instances and the other of handling the validation instances.

Processing speeds

We display two speeds, one of handling training instances and the other of handling validation instances. When handling the training instances, we show the latest speed of handling training instances and leave the speed of handling validation instances empty. When handling the validation instances, we show the latest speed of handling validation instances and leave the speed of handling training instances empty.

Percentage of model training work accomplished

The percentage of model training work accomplished is computed as the model training work accomplished so far / the estimated cost of model training. The cost of model training

= the cost of handling the training instances + the cost of handling the validation instances
= the cost of handling the training instances in U + the cost of handling the validation instances in W / the T-V cost ratio.

The actual T-V cost ratio is hard to estimate. For computing the percentage of model training work accomplished, we use a T-V cost ratio of 3 as an approximation to calculate both the model training work accomplished so far and the cost of model training. The T-V cost ratio is > 1. Typically, the training set is much bigger than the validation set. Hence, the cost of handling the training instances is much larger than the cost of handling the validation instances. In this case, using the approximation will not greatly degrade the accuracy of the computed accomplished percentage of model training work.

V. PERFORMANCE

This section shows the experimental results of our new progress estimation method. We did a coding implementation

of this method in Version 2.9.0 of TensorFlow, a major open-source deep learning software package [17]. In every test, our progress indicators gave useful estimates updated once per 10 seconds with a small overhead. This meets the 3 progress estimation goals listed in our previous paper [8]: small overhead, reasonable pacing, and continuous updates.

A. EXPERIMENT DESCRIPTION

We ran TensorFlow and did experiments on a Digital Storm workstation. This workstation has a GeForce RTX 2080 Ti GPU, an 8-core Intel Core i7-9800X 3.8 GHz CPU, 64 GB memory, a 3 TB SATA disk, and a 500 GB solid-state drive and runs the Ubuntu 18.04.02 operating system. We used both the CPU and the GPU to train each deep learning model on an unloaded computer.

The Amazon reviews polarity data set [28] and ImageNet-1k [14] are two popular benchmark data sets. For each of them, we used a subset of it to do our tests (see Table 1). From the Amazon reviews polarity data set's training set, we randomly sampled 135,000 data instances. We used 130,000 of them as a training set and the other 5,000 as a validation set for our tests. For ImageNet-1k, we used a subset of it called ImageNet-100 [29]. This subset contains 130,000 training instances and 5,000 validation instances.

TABLE 1. The data sets we employed to test the progress estimation method.

Name	Total number of validation instances	Number of training instances	Number of classes	Data instance size
Subset of the Amazon reviews polarity data set	5,000	130,000	2	average number of tokens: 102
ImageNet-100	5,000	130,000	100	average height 402 × average width 500

For the early stopping condition, we set the patience p to 9, an integer chosen from [3, 10] randomly, and the min_delta δ to 0.00820, a number chosen from [0, 0.01] randomly.

We tested 3 major deep learning models:

1) Bidirectional Encoder Representations from Transformers (BERT) [5], a transformer-based model trained on the subset of the Amazon reviews polarity data set.
2) ResNet50 [30], a convolutional neural network model trained on ImageNet-100.
3) A Long Short-Term Memory (LSTM) model [31] trained on the subset of the Amazon reviews polarity data set.

BERT model

When training the BERT model, we started from Version 2 of the pretrained bert_en_uncased_L-8_H-256_A-4 [32] model [33]. We used a given learning rate of 2×10^{-5} and the adaptive moment estimation with decoupled weight decay (AdamW) optimization algorithm [34]. We handled 100 training instances in each batch and allowed at most 25 epochs. We set all other hyper-parameters to their default values [32].

In each epoch, we did the following online data preprocessing steps:

1) We used the CPU to shuffle the training instances [35].
2) We used the GPU to do the default preprocessing steps in Version 1 of TensorFlow Hub's bert_en_uncased_preprocess model [36] to transform the text in each data instance to a set of numeric vectors.

ResNet50 model

When training the ResNet50 model, we handled 50 training instances in each batch and allowed at most 100 epochs. We tested 4 major optimization algorithms: classical stochastic gradient descent (SGD) [37], root mean square propagation (RMSprop) [38], adaptive moment estimation (Adam) [39], and adaptive gradient (AdaGrad) [40]. For each optimization algorithm, we tested 3 learning rate decay approaches: exponential decay, step decay, and employing a given learning rate. In the exponential decay approach, the k-th epoch ($k \geq 1$) uses a learning rate of $r_0 e^{-(k-1)\rho}$. ρ is a positive constant specifying the decay speed of the learning rate. r_0 is the starting learning rate that is > 0. We set ρ to 0.05 and r_0 to 10^{-3}. In the step decay approach, we reduced the learning rate from 10^{-3} to 10^{-4} when the 20th epoch began and then to 10^{-5} when the 40th epoch began. We set all other hyper-parameters to their default values [41].

In each epoch, we used the CPU to do the following online data preprocessing steps:

1) We shuffled the training instances [35].
2) We replaced each image in the training set with a 224 × 224 pixels patch randomly cropped from the image to obtain an augmented training instance.
3) We replaced each image in the validation set with a 224 × 224 pixels patch cropped from the center of the image to obtain an augmented validation instance.
4) In each patch, we normalized its pixels to have a mean of 0 and a variance of 1.

In the second and third steps, if the height (or width) of the original image is < 224 pixels, we increased the height (or width) to 224 pixels before we did the cropping.

LSTM model

In the LSTM model, we put a fully connected layer on top of 3 stacked bidirectional LSTM layers. We set each LSTM cell's output dimension to 1,024 and the dimension of each token's embedding vector to 128. When training the LSTM model, we used the exponential decay approach to control the learning rate and set ρ to 0.05 and r_0 to 2×10^{-5}. We used Adam, handled 100 training instances in each batch, and allowed at most 25 epochs.

In each epoch, we did 3 online data preprocessing steps:

1) We used the CPU to shuffle the training instances [35].
2) We used the CPU to tokenize the text of each data instance.

3) We used the GPU to map each token to an embedding vector.

In this section, we give all experimental results of training the BERT model and those of training the ResNet50 model using Adam. The experimental results of training the ResNet50 model using the other 3 optimization algorithms are given in Section A of the Appendix. The experimental results of training the LSTM model are given in Section B of the Appendix.

B. ACCURACY MEASURE

As in Chaudhuri *et al.* [42], we employed the average prediction error to assess the accuracy of the progress estimates. The average prediction error is = the area of the space between a diagonal and a curve / the area of the triangle formed by the x-axis, the y-axis, and the diagonal (see Fig. 5). The diagonal depicts the genuine outstanding model training time. The curve depicts the forecasted outstanding model training time over time. The bigger the average prediction error, the worse the progress estimates are.

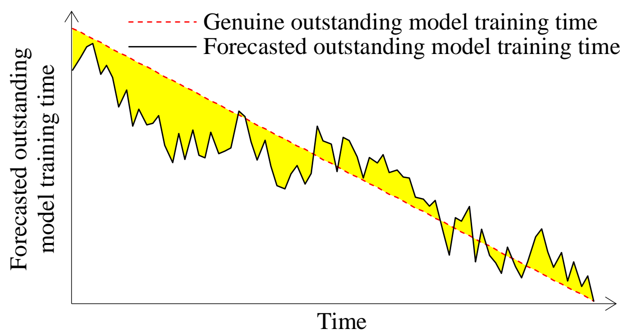


FIGURE 5. The areas of the spaces used to compute the average prediction error.

C. COMPARISON OF OUR FORMER AND NEW PROGRESS ESTIMATION METHODS

We compared the accuracy of the progress estimates given by our former [11] and new progress estimation methods. We did 14 tests, one for every (deep learning model, learning rate decay approach, optimization algorithm) combination listed in Section V-A. In every test, we trained the model 5 times, each in a different run. In every run, we employed each of our former and new progress estimation methods to give progress estimates. For every test, Table 2 lists for each of the two methods the mean and the standard deviation of the average prediction error over the 5 runs. There, we mark in bold the smaller mean between the two methods. Our new progress estimation method outperformed our former progress estimation method in every test. In comparison with our former method, on average our proposed new method cuts the error of the predicted outstanding model training time by 16.0%.

For each of the 14 tests, Table 3 lists the mean and the standard deviation of our new progress estimation method's runtime overhead, which is represented by the percentage rise in the model training time induced by the progress indicator. The mean runtime overhead across all tests is 4.69%.

In Section V-D, Section V-E, and the Appendix, we present the estimated outstanding model training time given by both our former and new progress estimation methods, as well as the other progress estimates given by our new progress estimation method. In each of the 14 tests, we trained the model 5 times. We randomly chose one of them and present the progress estimates given over time there.

D. EXPERIMENTAL RESULTS OF TRAINING THE BERT MODEL

This test employed a given learning rate as well as AdamW to train the BERT model. Fig. 6 displays the cost of handling the training instances forecasted over time by our new progress estimation method, with the horizontal dotted line marking the genuine cost of handling the training instances. Fig. 7 displays the cost of handling the validation instances forecasted over time by our new progress estimation method, with the horizontal dotted line marking the genuine cost of handling the validation instances. Within several hundred seconds after model training began, both forecasted costs became relatively accurate.

TABLE 2. For each combination of one of the 14 tests and one of our former and new progress estimation methods, the mean and the standard deviation of the average prediction error over the 5 runs.

Deep learning model	Learning rate decay approach	Optimization algorithm	Average prediction error: Our new progress estimation method	Average prediction error: Our former progress estimation method
BERT	Using a given learning rate	AdamW	**0.48 ± 0.27**	0.54 ± 0.28
LSTM	Exponential decay	Adam	**0.95 ± 0.23**	0.99 ± 0.23
ResNet50	Using a given learning rate	Adam	**0.36 ± 0.08**	0.48 ± 0.09
		RMSprop	**0.69 ± 0.08**	0.84 ± 0.09
		SGD	**0.56 ± 0.19**	0.67 ± 0.22
		AdaGrad	**0.65 ± 0.13**	0.79 ± 0.16
	Exponential decay	Adam	**0.77 ± 0.27**	0.97 ± 0.31
		RMSprop	**1.03 ± 0.24**	1.22 ± 0.27
		SGD	**0.30 ± 0.05**	0.38 ± 0.05

		AdaGrad	**0.49 ± 0.19**	0.60 ± 0.22
	Step decay	Adam	**0.44 ± 0.12**	0.56 ± 0.14
		RMSprop	**0.91 ± 0.08**	1.08 ± 0.10
		SGD	**0.91 ± 0.31**	1.06 ± 0.35
		AdaGrad	**0.99 ± 0.04**	1.18 ± 0.04
Over all runs in all tests			**0.68 ± 0.30**	0.81 ± 0.34

TABLE 3. For each of the 14 tests, the mean and the standard deviation of our new progress estimation method's runtime overhead over the 5 runs.

Deep learning model	Learning rate decay approach	Optimization algorithm	Runtime overhead
BERT	Using a given learning rate	AdamW	2.17% ± 0.30%
LSTM	Exponential decay	Adam	2.46% ± 0.09%
ResNet50	Using a given learning rate	Adam	5.65% ± 0.35%
		RMSprop	5.37% ± 0.25%
		SGD	3.90% ± 0.36%
		AdaGrad	4.93% ± 0.32%
	Exponential decay	Adam	6.47% ± 0.49%
		RMSprop	5.55% ± 0.49%
		SGD	3.45% ± 0.11%
		AdaGrad	4.53% ± 0.45%
	Step decay	Adam	5.48% ± 0.00%
		RMSprop	5.48% ± 0.00%
		SGD	4.73% ± 0.48%
		AdaGrad	5.48% ± 0.00%
Over all runs in all tests			4.69% ± 1.26%

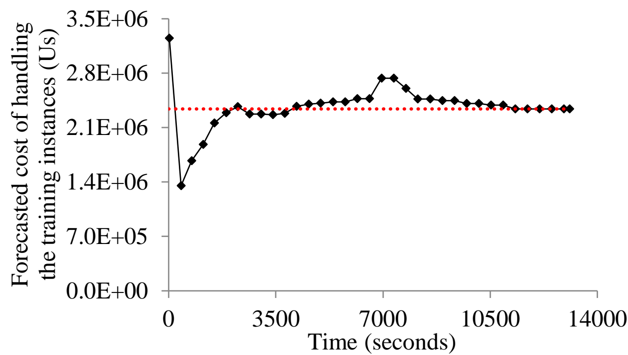


FIGURE 6. Cost of handling the training instances forecasted over time (employing a given learning rate as well as AdamW to train the BERT model).

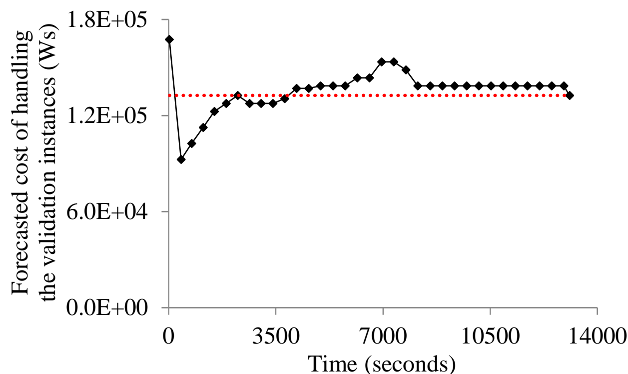


FIGURE 7. Cost of handling the validation instances forecasted over time (employing a given learning rate as well as AdamW to train the BERT model).

Fig. 8 displays both the speed of handling training instances and the speed of handling validation instances estimated by our new progress estimation method over time. Both estimated speeds were decently stable throughout the whole model training process.

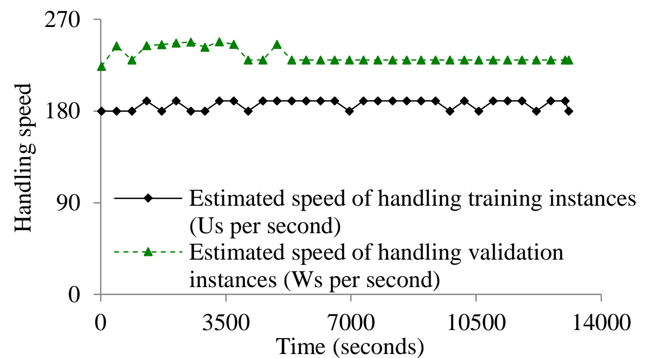


FIGURE 8. The speed of handling training instances and the speed of handling validation instances estimated over time (employing a given learning rate as well as AdamW to train the BERT model).

Fig. 9 and 10 display the outstanding model training time forecasted over time by our new and former progress estimation methods, respectively, with the dashed line marking the genuine outstanding model training time. For the reason given in the introduction, the outstanding model

training time forecasted by our former progress estimation method often differs greatly from the genuine outstanding model training time. Our new progress estimation method does not have this problem.

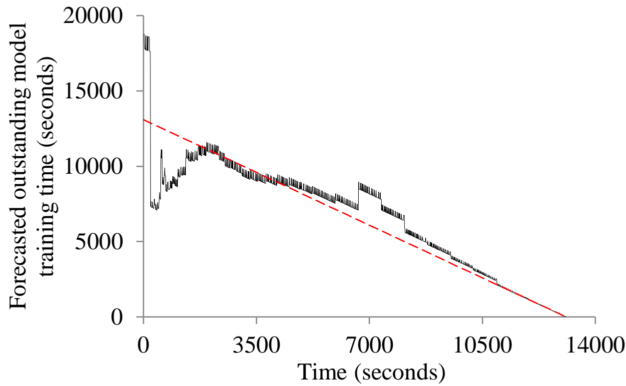


FIGURE 9. **Outstanding model training time forecasted by our new progress estimation method (employing a given learning rate as well as AdamW to train the BERT model).**

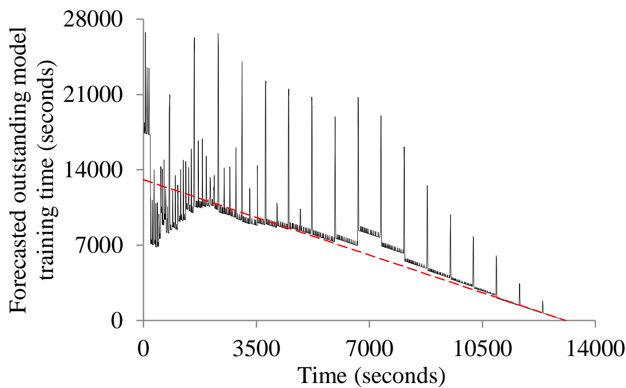


FIGURE 10. **Outstanding model training time forecasted by our former progress estimation method (employing a given learning rate as well as AdamW to train the BERT model).**

Fig. 11 displays the accomplished percentage of model training work estimated by our new progress estimation method over time. The curve depicting the forecasted accomplished percentage is decently near the dotted diagonal that joins the lower left and the upper right corners.

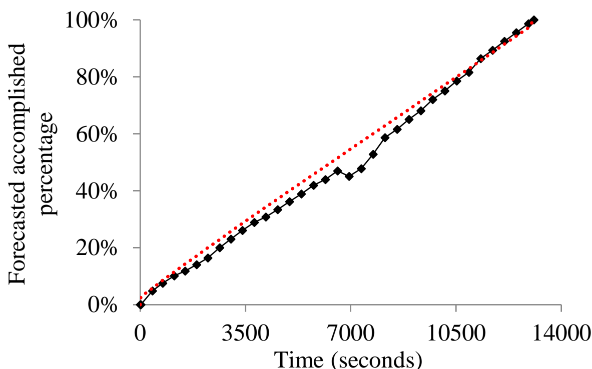


FIGURE 11. **Accomplished percentage forecasted over time (employing a given learning rate as well as AdamW to train the BERT model).**

E. EXPERIMENTAL RESULTS OF TRAINING THE RESNET50 MODEL

1) EXPERIMENTAL RESULTS OF EMPLOYING A GIVEN LEARNING RATE

This test employed a given learning rate as well as Adam to train the ResNet50 model. Fig. 12 displays the cost of handling the training instances forecasted over time by our new progress estimation method, with the horizontal dotted line marking the genuine cost of handling the training instances. Fig. 13 displays the cost of handling the validation instances forecasted over time by our new progress estimation method, with the horizontal dotted line marking the genuine cost of handling the validation instances. When model training just started, both forecasted costs diverged notably from the genuine costs. Once we reached the 4th point of validation and was able to refine our initially guessed costs within 243 seconds, both forecasted costs became much more accurate.

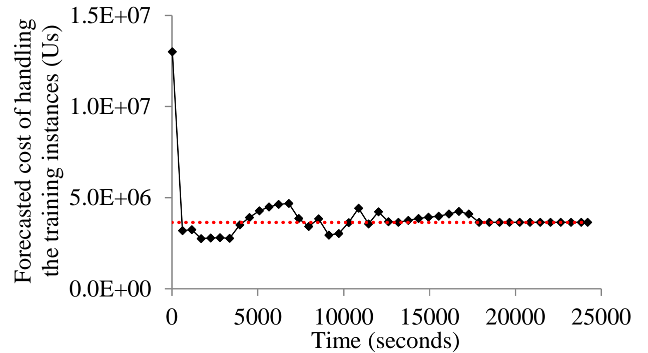


FIGURE 12. **Cost of handling the training instances forecasted over time (employing a given learning rate as well as Adam to train the ResNet50 model).**

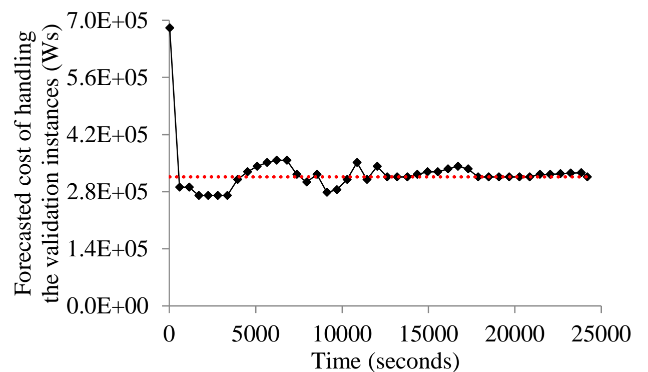


FIGURE 13. **Cost of handling the validation instances forecasted over time (employing a given learning rate as well as Adam to train the ResNet50 model).**

Fig. 14 displays both the speed of handling training instances and the speed of handling validation instances estimated by our new progress estimation method over time. Both estimated speeds were decently stable throughout the whole model training process.

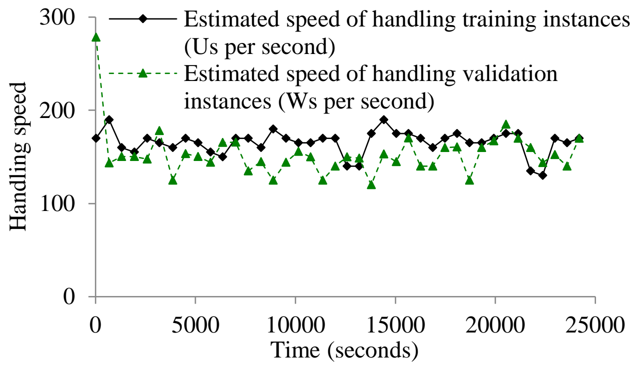


FIGURE 14. **The speed of handling training instances and the speed of handling validation instances estimated over time (employing a given learning rate as well as Adam to train the ResNet50 model).**

Fig. 15 and 16 display the outstanding model training time forecasted over time by our new and former progress estimation methods, respectively, with the dashed line depicting the genuine outstanding model training time. For the reason given in the introduction, the outstanding model training time forecasted by our former progress estimation method often differs greatly from the genuine outstanding model training time. Our new progress estimation method does not have this problem.

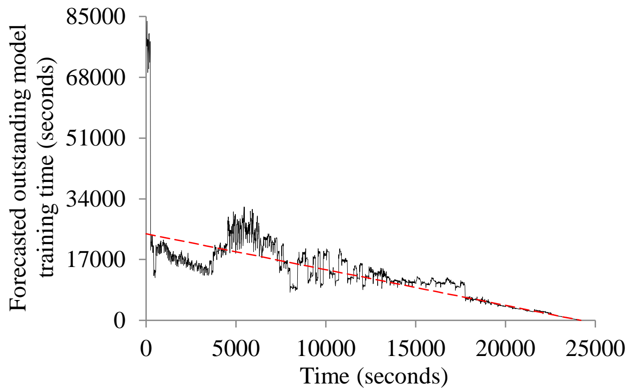


FIGURE 15. **Outstanding model training time forecasted by our new progress estimation method (employing a given learning rate as well as Adam to train the ResNet50 model).**

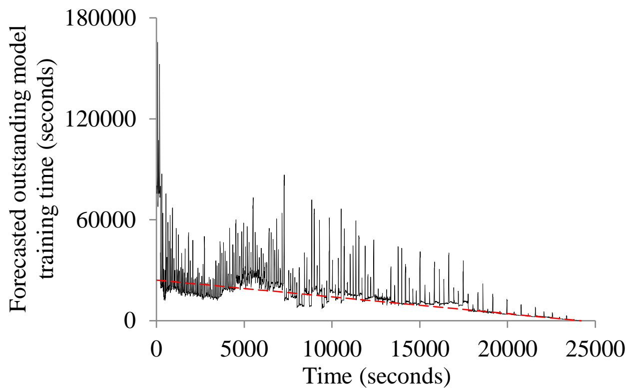


FIGURE 16. **Outstanding model training time forecasted by our former progress estimation method (employing a given learning rate as well as Adam to train the ResNet50 model).**

Fig. 17 displays the accomplished percentage of model training work estimated by our new progress estimation method over time. The curve depicting the forecasted accomplished percentage is decently near the dotted diagonal that joins the lower left and the upper right corners.

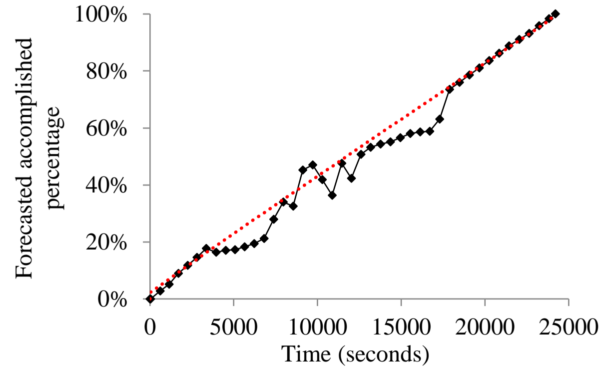


FIGURE 17. **Accomplished percentage forecasted over time (employing a given learning rate as well as Adam to train the ResNet50 model).**

2) EXPERIMENTAL RESULTS OF EMPLOYING THE EXPONENTIAL DECAY APPROACH TO CONTROL THE LEARNING RATE

This test employed the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model. Fig. 18-23 display this test's results. Overall, our new progress estimation method produced relatively good estimates of the cost of handling the training instances, the cost of handling the validation instances, and the outstanding model training time. Compared to our former progress estimation method, our new progress estimation method provided more stable estimates of the outstanding model training time.

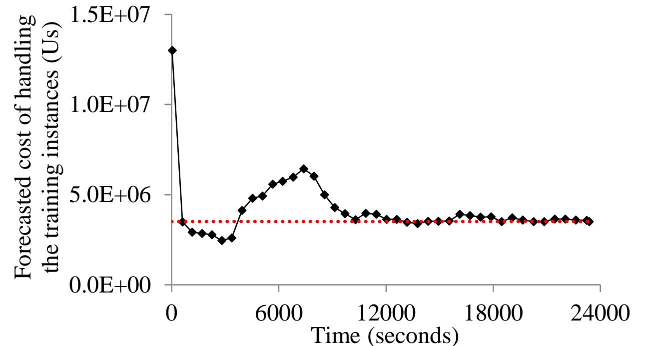


FIGURE 18. **Cost of handling the training instances forecasted over time (employing the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model).**

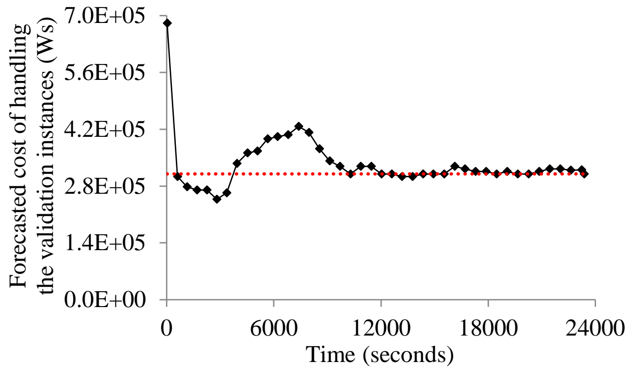


FIGURE 19. Cost of handling the validation instances forecasted over time (employing the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model).

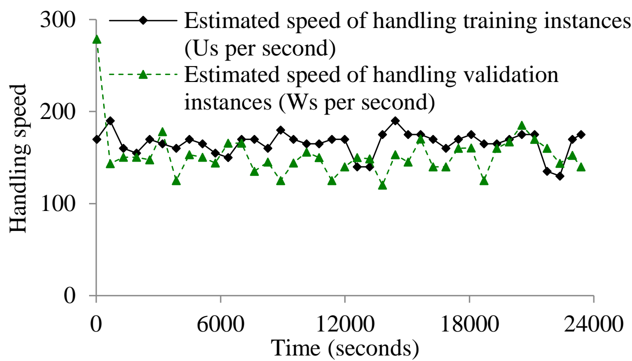


FIGURE 20. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model).

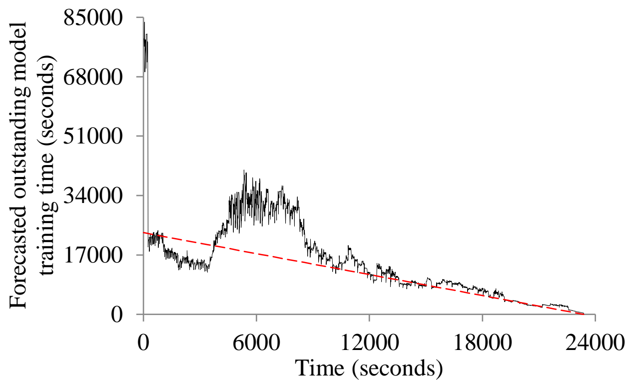


FIGURE 21. Outstanding model training time forecasted by our new progress estimation method (employing the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model).

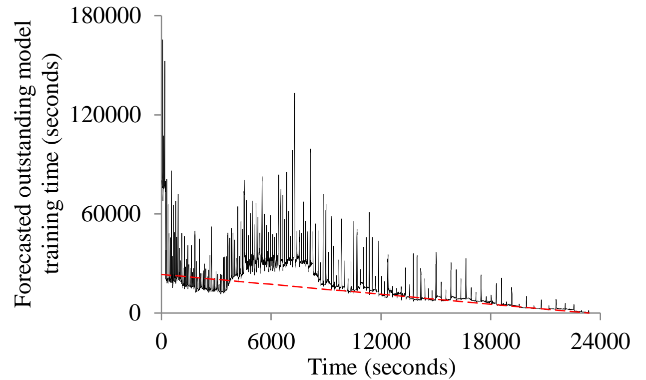


FIGURE 22. Outstanding model training time forecasted by our former progress estimation method (employing the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model).

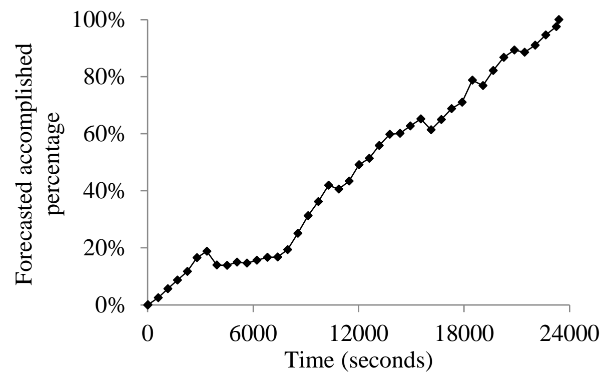


FIGURE 23. Accomplished percentage forecasted over time (employing the exponential decay approach to control the learning rate as well as Adam to train the ResNet50 model).

3) EXPERIMENTAL RESULTS OF EMPLOYING THE STEP DECAY APPROACH TO CONTROL THE LEARNING RATE

This test employed the step decay approach to control the learning rate as well as Adam to train the ResNet50 model. We reduced the learning rate from 10^{-3} to 10^{-4} when the 20th epoch began and then to 10^{-5} when the 40th epoch began. Early stopping occurred between the 20th epoch and the 40th epoch. Fig. 24-29 display this test's results. In each of these figures, we use a dash-dotted vertical line to show when the learning rate dropped. Overall, our new progress estimation method produced relatively good estimates of the cost of handling the training instances, the cost of handling the validation instances, and the outstanding model training time. Compared to our former progress estimation method, our new progress estimation method provided more stable estimates of the outstanding model training time.

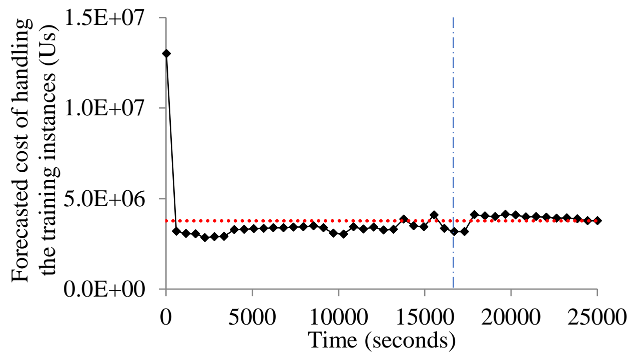


FIGURE 24. Cost of handling the training instances forecasted over time (employing the step decay approach to control the learning rate as well as Adam to train the ResNet50 model).

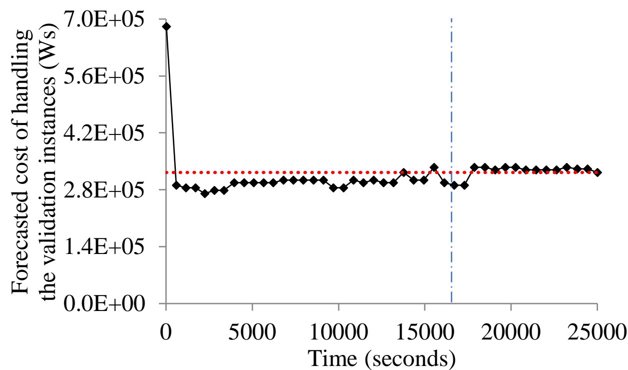


FIGURE 25. Cost of handling the validation instances forecasted over time (employing the step decay approach to control the learning rate as well as Adam to train the ResNet50 model).

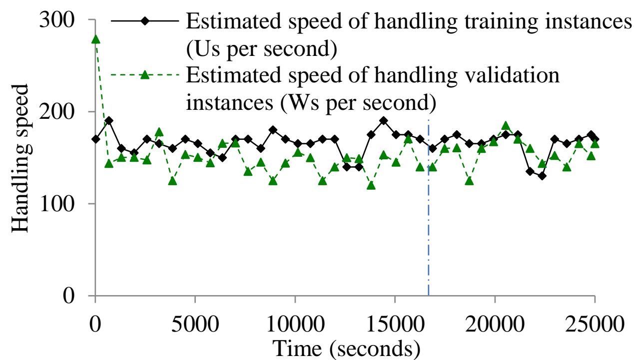


FIGURE 26. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the step decay approach to control the learning rate as well as Adam to train the ResNet50 model).

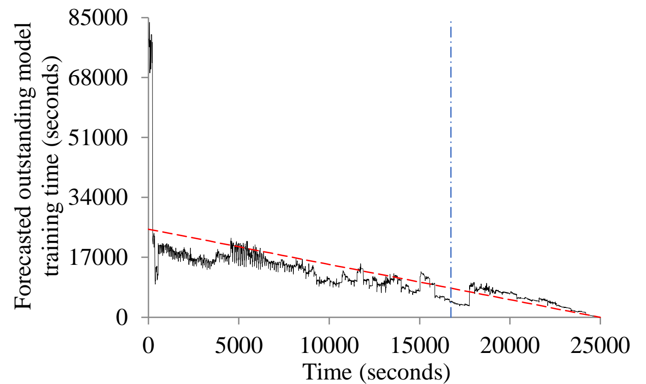


FIGURE 27. Outstanding model training time forecasted by our new progress estimation method (employing the step decay approach to control the learning rate as well as Adam to train the ResNet50 model).

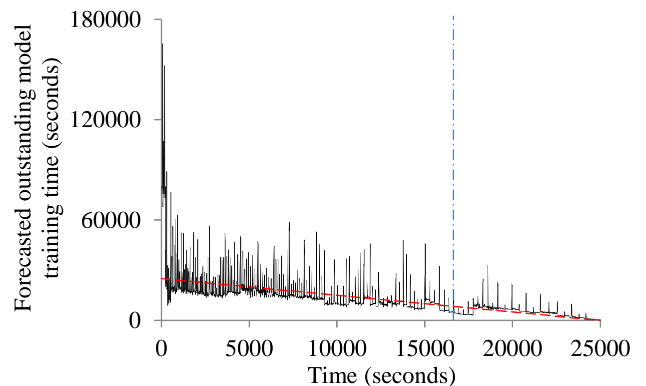


FIGURE 28. Outstanding model training time forecasted by our former progress estimation method (employing the step decay approach to control the learning rate as well as Adam to train the ResNet50 model).

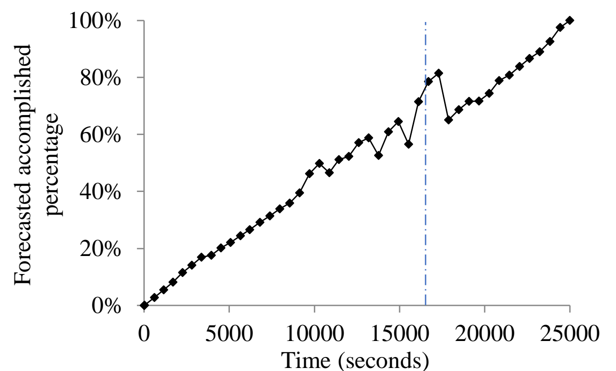


FIGURE 29. Accomplished percentage forecasted over time (employing the step decay approach to control the learning rate as well as Adam to train the ResNet50 model).

VI. RELATED WORK

This section briefly goes over the related work. Our prior paper [8] discusses the related work in detail.

Advanced progress indicators

Several researchers have designed advanced progress indicators for program compilation [43], software model checking [44], static program analysis [45], automatic machine learning model selection [46], [47], MapReduce jobs

[48], [49], database queries [9], [42], [50]-[52], and subgraph queries [53]. We have also designed advanced progress indicators for building several types of machine learning models like neural network, decision tree, and random forest [8], [10], [11], [54].

Predicting the deep learning model training time

To predict an epoch's runtime before one starts training a deep learning model, Justus *et al*. [55] designed a meta learning method. This method uses several features of the current model, the training data set adopted to build another deep learning model, and the computing resources, but predicts neither the time nor the number of epochs it takes to train the current model.

To predict the time needed to train a deep learning model before one starts training the model, multiple researchers have designed a Bayesian optimization method [56] as well as several meta learning methods using multivariate adaptive regression splines [57], polynomial regression [58], and support vector regression [59], respectively. The numbers predicted by these methods are often inaccurate, can differ a lot from the genuine time taken to train the model on a loaded computer, and are not revised continuously. In comparison, when our progress estimation method predicts the outstanding deep learning model training time, we consider the load on the computer and keep revising our predicted numbers.

Complexity analysis for neural network training

Multiple researchers have computed the time complexity of building a neural network model [60]-[62]. But, this information can neither give us an estimated model training time on a loaded computer nor help us create a progress indicator. Usually, time complexity disregards the data properties that affect the cost of model training and the lower order terms and the coefficients needed to predict that cost. During model training, a non-trivial progress indicator should keep revising its predicted cost of model training.

VII. POSSIBLE DIRECTIONS TO DO FUTURE WORK

This section lists several possible directions to do future work.

This work derives no upper bounds on the error of the predicted cost of handling the training instances and that of handling the validation instances. In the future, we could adopt an approach akin to what Chaudhuri *et al*. [63] used for database query progress estimation to compute such upper bounds.

As a case study, both this work and our prior work [10], [11] employ the same early stopping condition to show that we can create advanced progress indicators for deep learning model training. There are many other early stopping conditions [1], [64]-[66]. In the future, we will examine how to extend our current progress estimation techniques to accommodate other commonly used early stopping conditions.

This work addresses deep learning classification and uses error rate as a model performance metric. Deep learning is also used for regression, where mean squared error is used as a model performance metric. In the future, we will extend our current progress estimation techniques to handle that case.

VIII. CONCLUSION

This paper presents a new progress estimation method to handle end-to-end deep learning model training with online data preprocessing. This new method overcomes our former progress estimation method's limitation of ignoring online data preprocessing, which commonly takes a large percentage of model training time. Our tests show that when online data preprocessing is used and in comparison with our former method, our proposed new method produces more stable progress estimates for model training and on average lowers the error of the predicted outstanding model training time by 16.0%.

APPENDIX

A. OTHER EXPERIMENTAL RESULTS OF TRAINING THE RESNET50 MODEL

1) EXPERIMENTAL RESULTS OF EMPLOYING A GIVEN LEARNING RATE

Employing RMSprop

This test employed a given learning rate as well as RMSprop to train the ResNet50 model. Fig. A1-A6 display the experimental results, which resemble those displayed in Fig. 12-17.

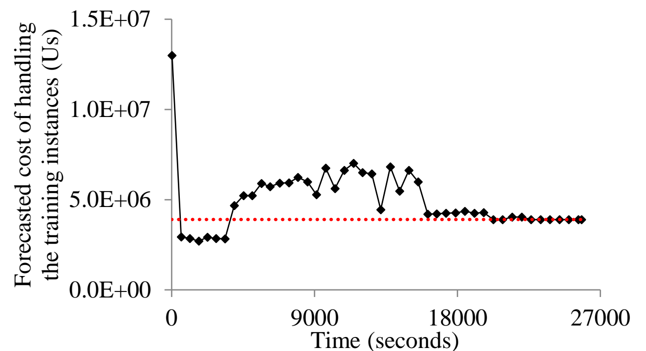


FIGURE A1. Cost of handling the training instances forecasted over time (employing a given learning rate as well as RMSprop to train the ResNet50 model).

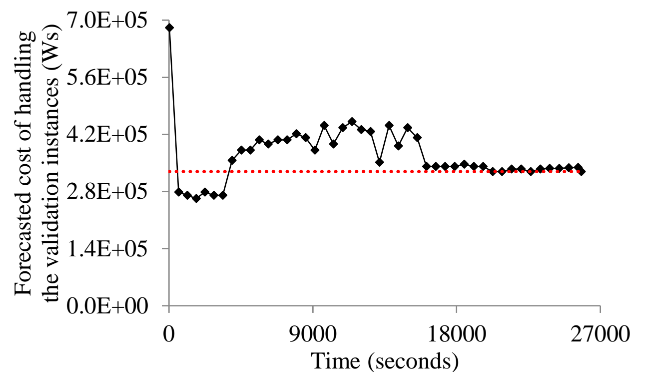


FIGURE A2. Cost of handling the validation instances forecasted over time (employing a given learning rate as well as RMSprop to train the ResNet50 model).

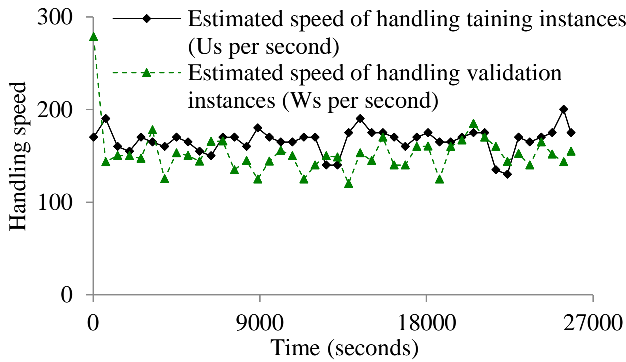


FIGURE A3. The speed of handling training instances and the speed of handling validation instances estimated over time (employing a given learning rate as well as RMSprop to train the ResNet50 model).

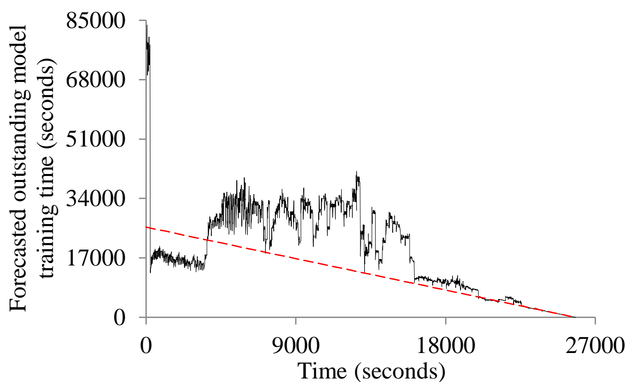


FIGURE A4. Outstanding model training time forecasted by our new progress estimation method (employing a given learning rate as well as RMSprop to train the ResNet50 model).

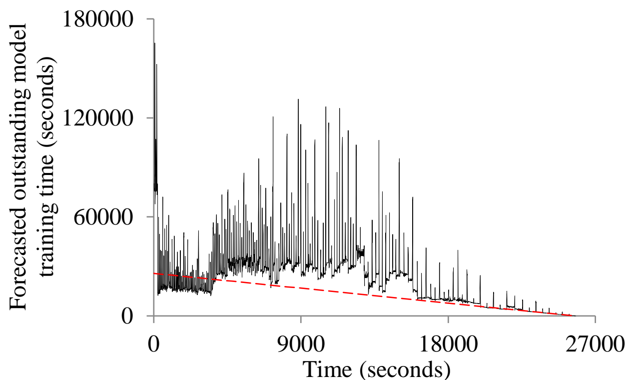


FIGURE A5. Outstanding model training time forecasted by our former progress estimation method (employing a given learning rate as well as RMSprop to train the ResNet50 model).

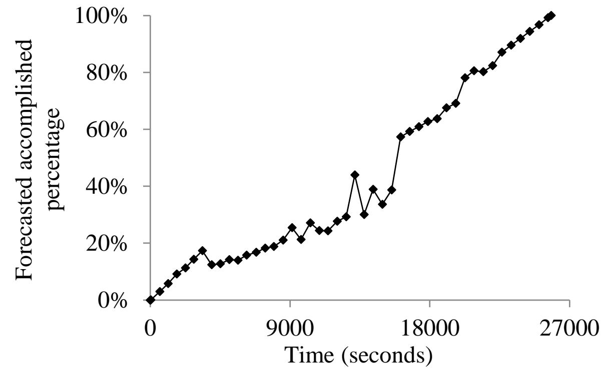


FIGURE A6. Accomplished percentage forecasted over time (employing a given learning rate as well as RMSprop to train the ResNet50 model).

Employing SGD

This test employed a given learning rate as well as SGD to train the ResNet50 model. Fig. A7-A12 display the experimental results, which resemble those displayed in Fig. 12-17.

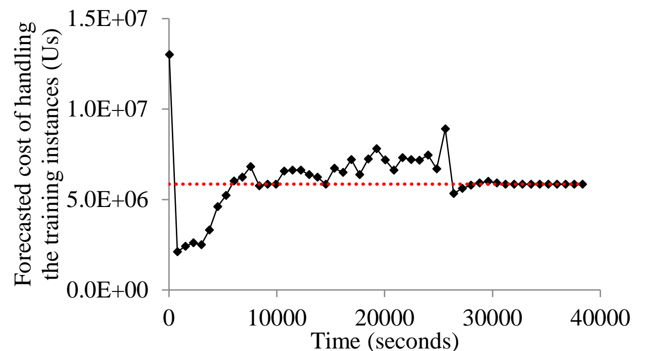


FIGURE A7. Cost of handling the training instances forecasted over time (employing a given learning rate as well as SGD to train the ResNet50 model).

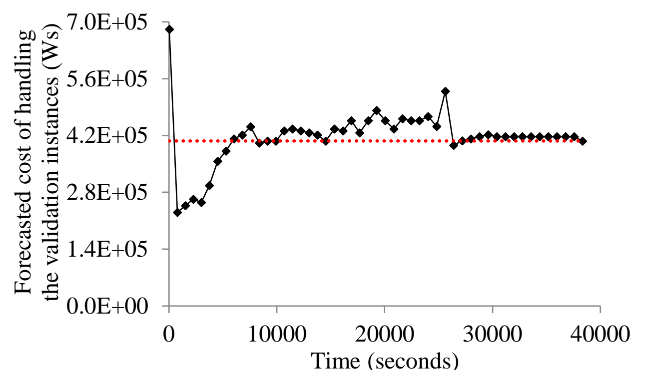


FIGURE A8. Cost of handling the validation instances forecasted over time (employing a given learning rate as well as SGD to train the ResNet50 model).

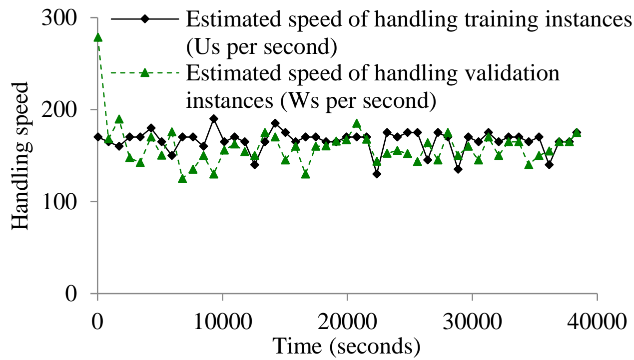


FIGURE A9. **The speed of handling training instances and the speed of handling validation instances estimated over time (employing a given learning rate as well as SGD to train the ResNet50 model).**

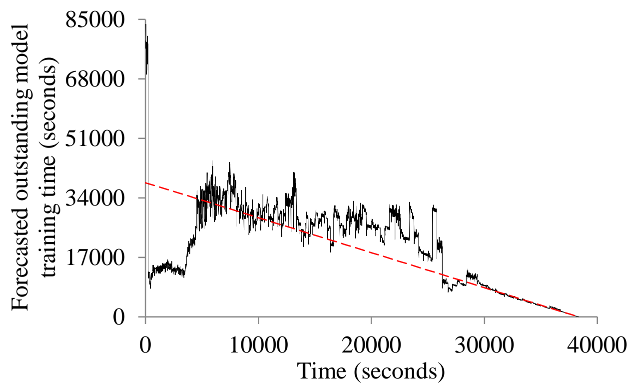


FIGURE A10. **Outstanding model training time forecasted by our new progress estimation method (employing a given learning rate as well as SGD to train the ResNet50 model).**

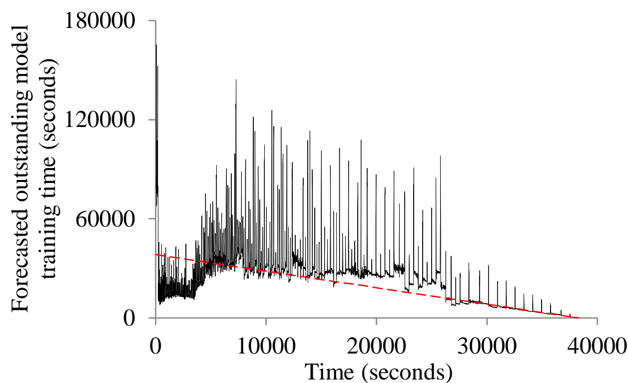


FIGURE A11. **Outstanding model training time forecasted by our former progress estimation method (employing a given learning rate as well as SGD to train the ResNet50 model).**

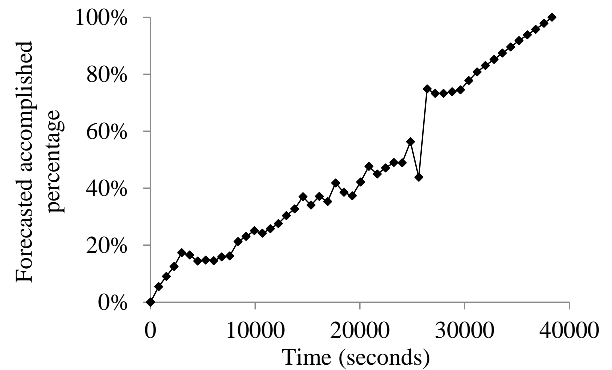


FIGURE A12. **Accomplished percentage forecasted over time (employing a given learning rate as well as SGD to train the ResNet50 model).**

Employing AdaGrad

This test employed a given learning rate as well as AdaGrad to train the ResNet50 model. Fig. A13-A18 display the experimental results, which resemble those displayed in Fig. 12-17.

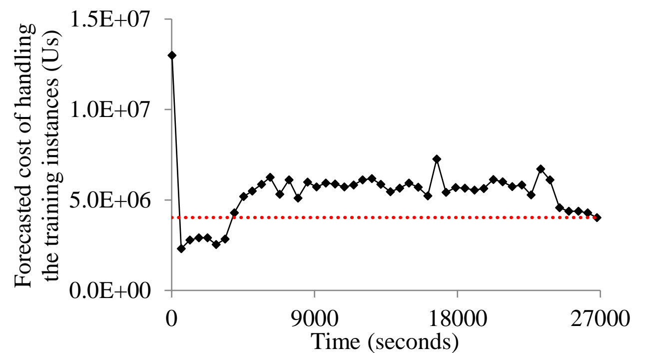


FIGURE A13. **Cost of handling the training instances forecasted over time (employing a given learning rate as well as AdaGrad to train the ResNet50 model).**

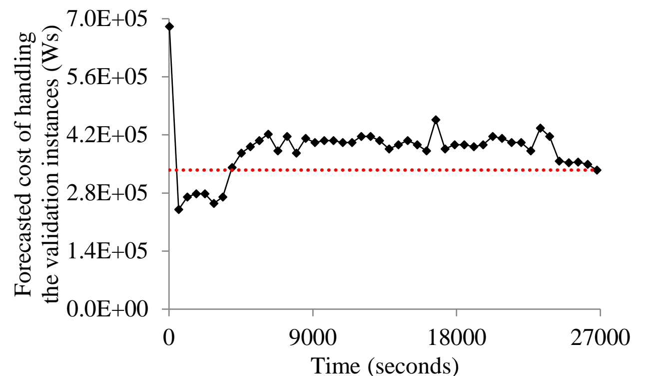


FIGURE A14. **Cost of handling the validation instances forecasted over time (employing a given learning rate as well as AdaGrad to train the ResNet50 model).**

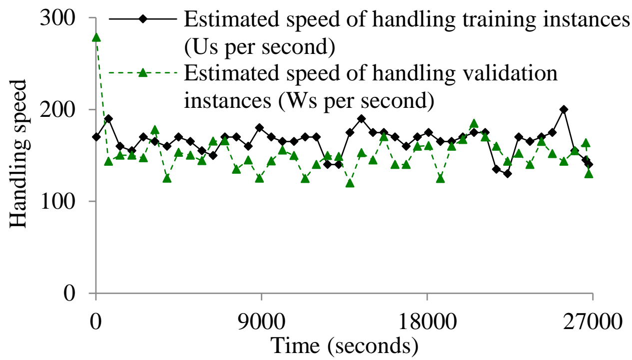


FIGURE A15. The speed of handling training instances and the speed of handling validation instances estimated over time (employing a given learning rate as well as AdaGrad to train the ResNet50 model).

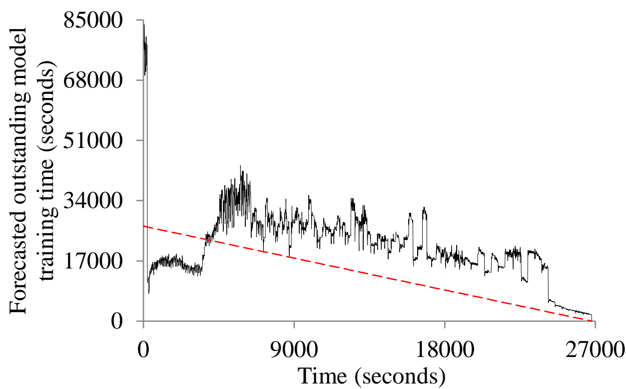


FIGURE A16. Outstanding model training time forecasted by our new progress estimation method (employing a given learning rate as well as AdaGrad to train the ResNet50 model).

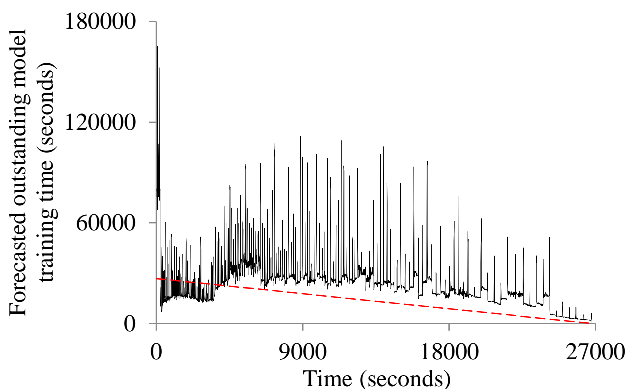


FIGURE A17. Outstanding model training time forecasted by our former progress estimation method (employing a given learning rate as well as AdaGrad to train the ResNet50 model).

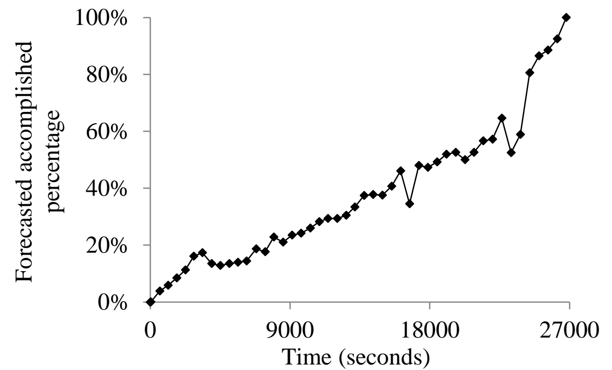


FIGURE A18. Accomplished percentage forecasted over time (employing a given learning rate as well as AdaGrad to train the ResNet50 model).

2) EXPERIMENTAL RESULTS OF EMPLOYING THE EXPONENTIAL DECAY APPROACH TO CONTROL THE LEARNING RATE

Employing RMSprop

This test employed the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model. Fig. A19-A24 display the experimental results, which resemble those displayed in Fig. 18-23.

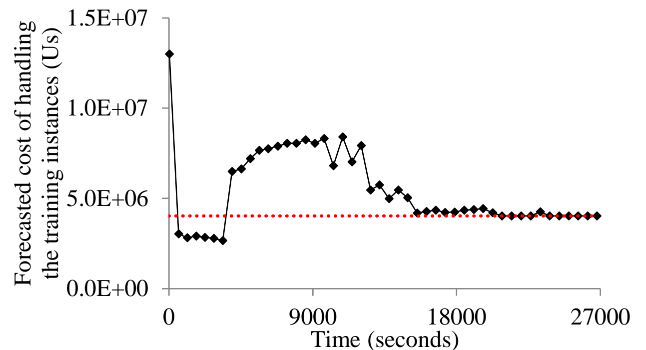


FIGURE A19. Cost of handling the training instances forecasted over time (employing the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

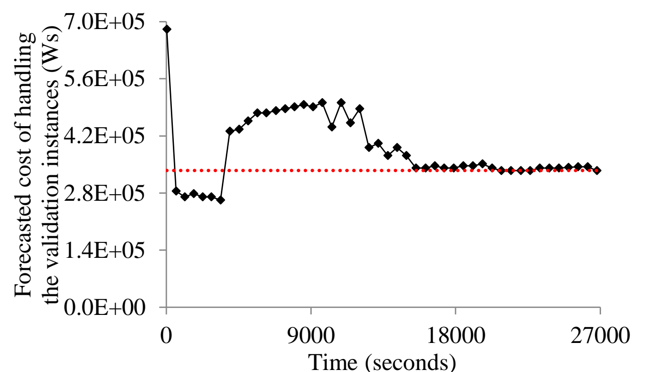


FIGURE A20. Cost of handling the validation instances forecasted over time (employing the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

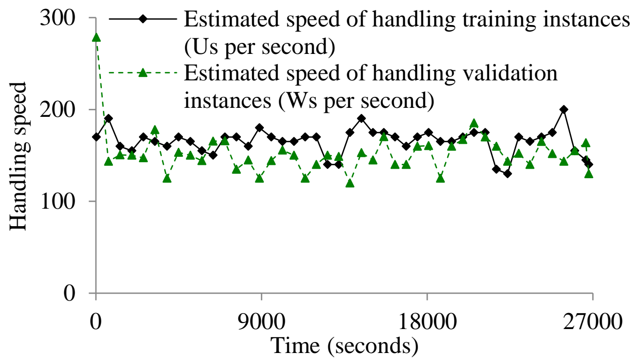


FIGURE A21. **The speed of handling training instances and the speed of handling validation instances estimated over time (employing the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).**

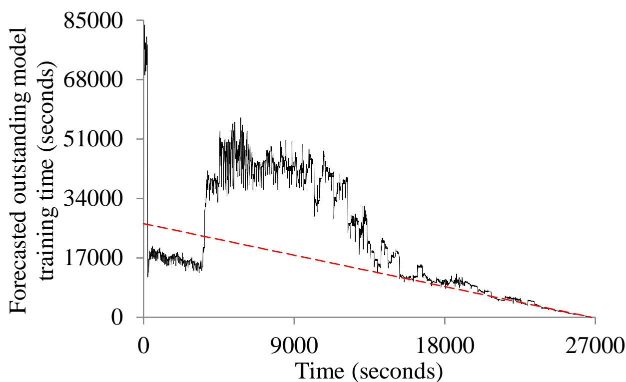


FIGURE A22. **Outstanding model training time forecasted by our new progress estimation method (employing the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).**

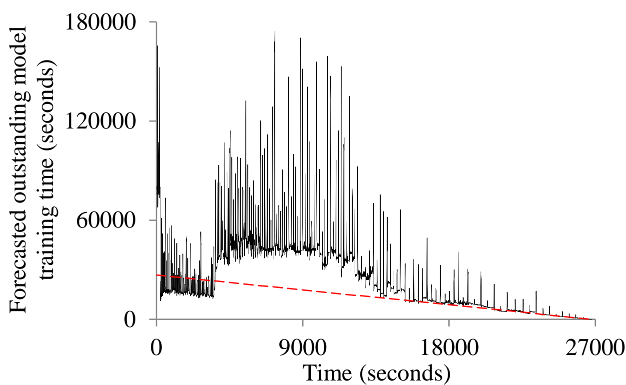


FIGURE A23. **Outstanding model training time forecasted by our former progress estimation method (employing the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).**

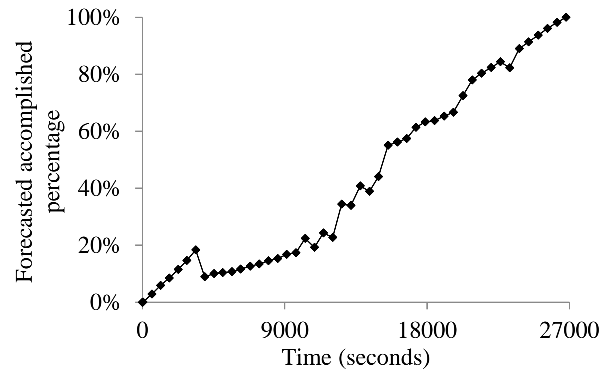


FIGURE A24. **Accomplished percentage forecasted over time (employing the exponential decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).**

Employing SGD

This test employed the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model. Fig. A25-A30 display the experimental results, which resemble those displayed in Fig. 18-23.

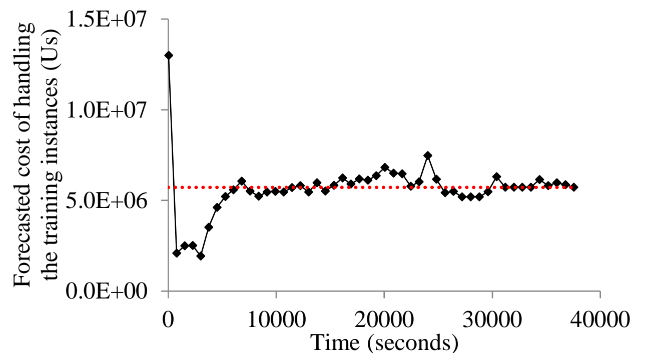


FIGURE A25. **Cost of handling the training instances forecasted over time (employing the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model).**

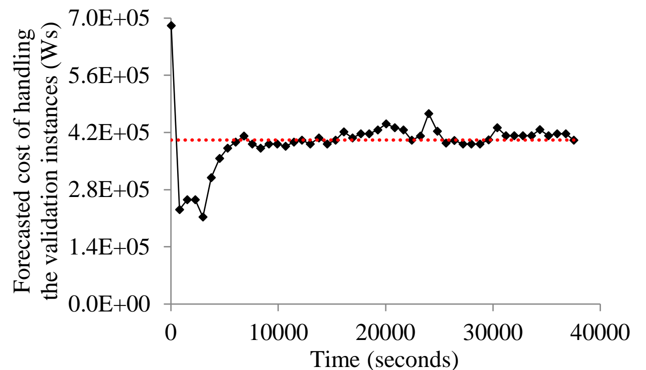


FIGURE A26. **Cost of handling the validation instances forecasted over time (employing the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model).**

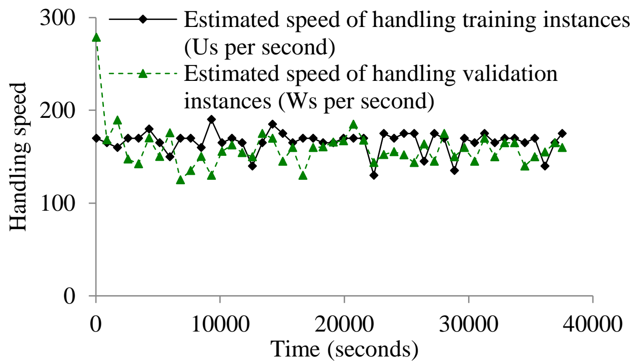


FIGURE A27. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model).

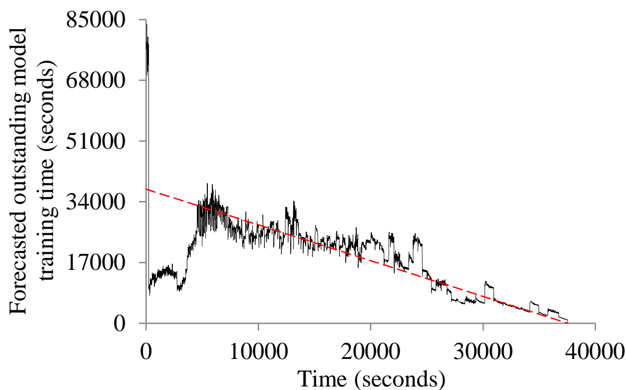


FIGURE A28. Outstanding model training time forecasted by our new progress estimation method (employing the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model).

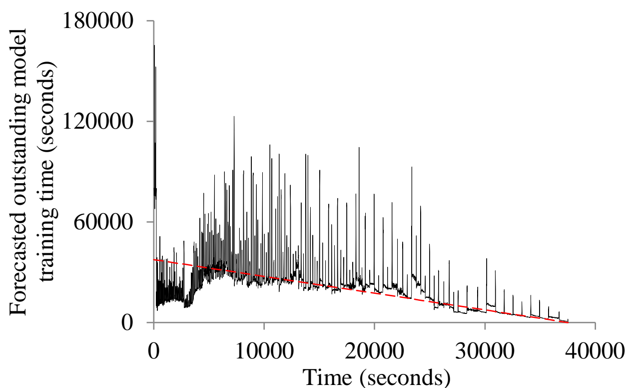


FIGURE A29. Outstanding model training time forecasted by our former progress estimation method (employing the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model).

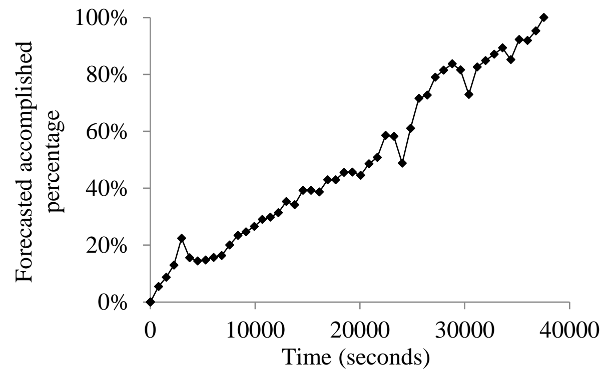


FIGURE A30. Accomplished percentage forecasted over time (employing the exponential decay approach to control the learning rate as well as SGD to train the ResNet50 model).

Employing AdaGrad

This test employed the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model. Fig. A31-A36 display the experimental results, which resemble those displayed in Fig. 18-23.

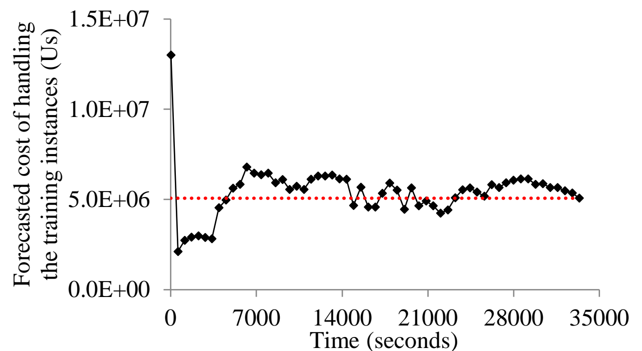


FIGURE A31. Cost of handling the training instances forecasted over time (employing the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

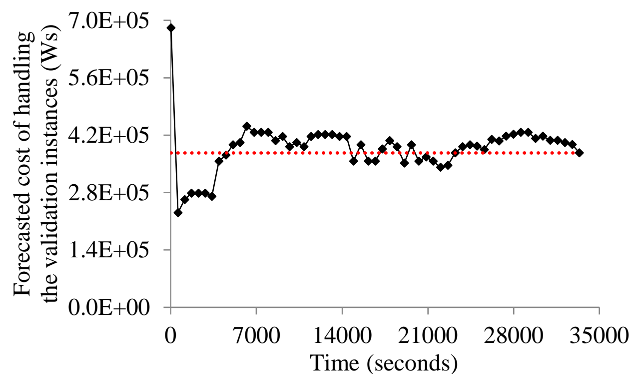


FIGURE A32. Cost of handling the validation instances forecasted over time (employing the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

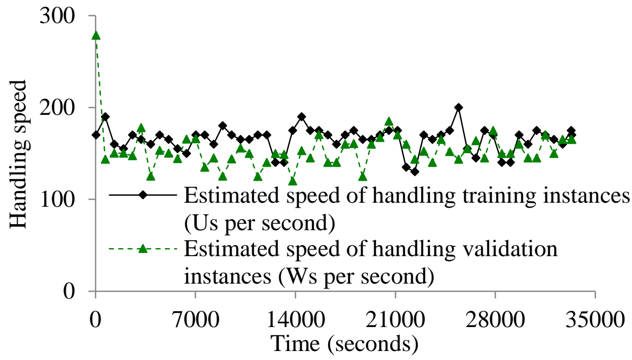


FIGURE A33. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

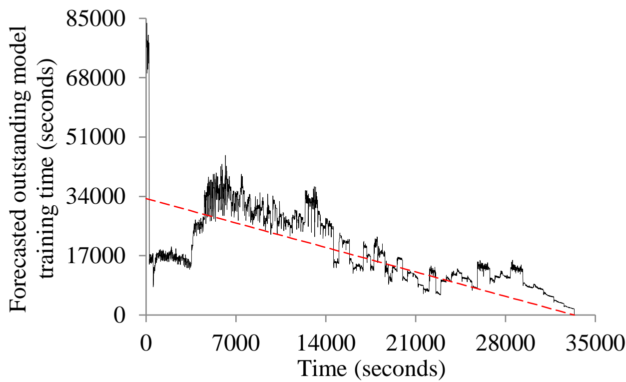


FIGURE A34. Outstanding model training time forecasted by our new progress estimation method (employing the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

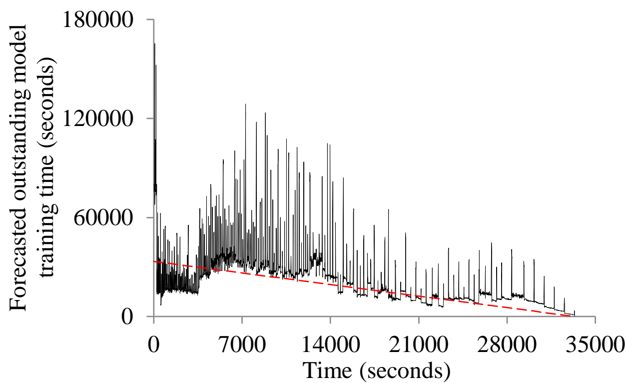


FIGURE A35. Outstanding model training time forecasted by our former progress estimation method (employing the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

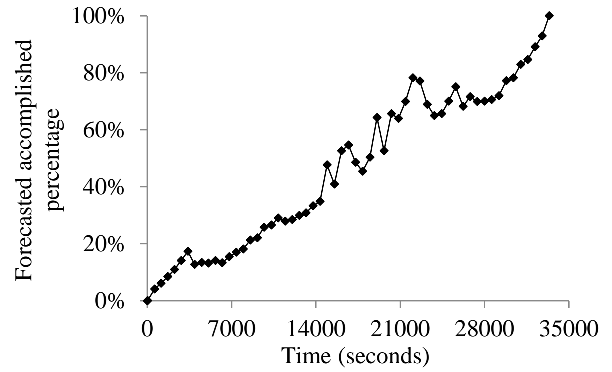


FIGURE A36. Accomplished percentage forecasted over time (employing the exponential decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

3) EXPERIMENTAL RESULTS OF EMPLOYING THE STEP DECAY APPROACH TO CONTROL THE LEARNING RATE

In each test that employed the step decay approach to control the learning rate to train the ResNet50 model, we reduced the learning rate from 10^{-3} to 10^{-4} when the 20th epoch began and then to 10^{-5} when the 40th epoch began. Early stopping occurred between the 20th epoch and the 40th epoch. In each figure displayed in this section, we employ a dash-dotted vertical line to show when the learning rate dropped.

Employing RMSprop

This test employed the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model. Fig. A37-A42 display the experimental results.

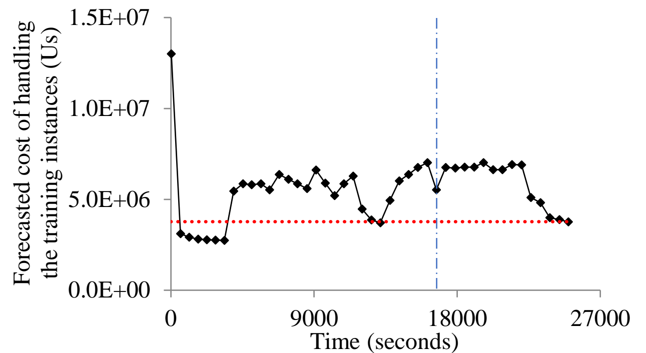


FIGURE A37. Cost of handling the training instances forecasted over time (employing the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

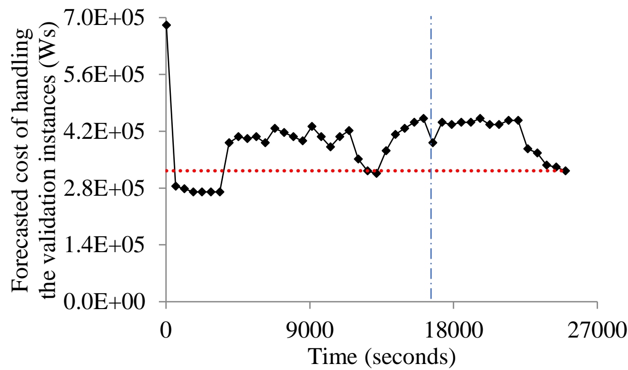


FIGURE A38. Cost of handling the validation instances forecasted over time (employing the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

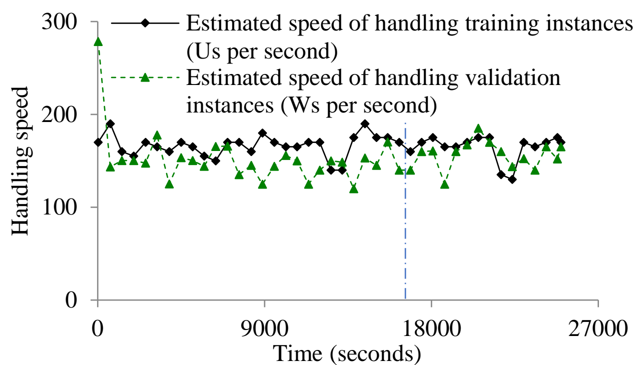


FIGURE A39. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

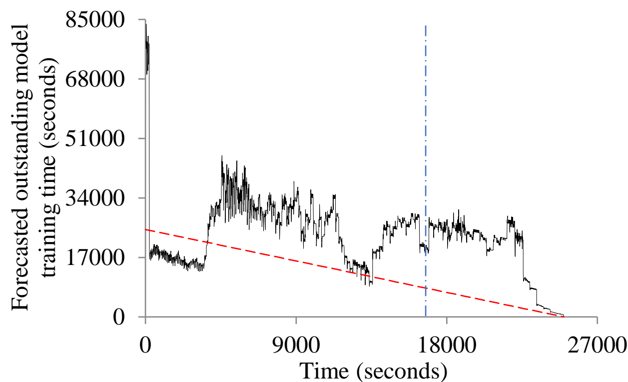


FIGURE A40. Outstanding model training time forecasted by our new progress estimation method (employing the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

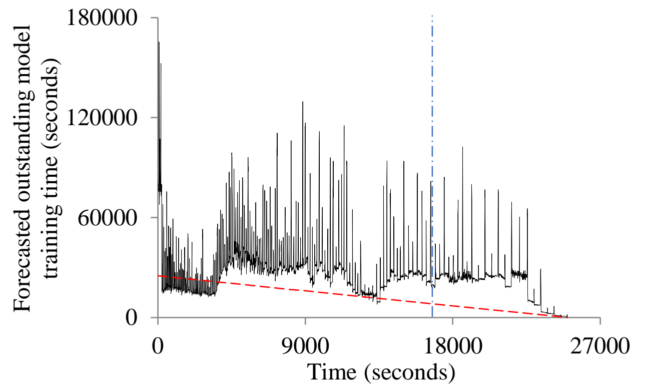


FIGURE A41. Outstanding model training time forecasted by our former progress estimation method (employing the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

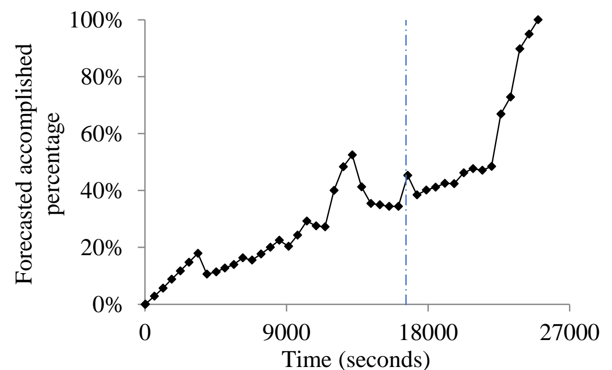


FIGURE A42. Accomplished percentage forecasted over time (employing the step decay approach to control the learning rate as well as RMSprop to train the ResNet50 model).

Employing SGD

This test employed the step decay approach to control the learning rate as well as SGD to train the ResNet50 model. Fig. A43-A48 display the experimental results.

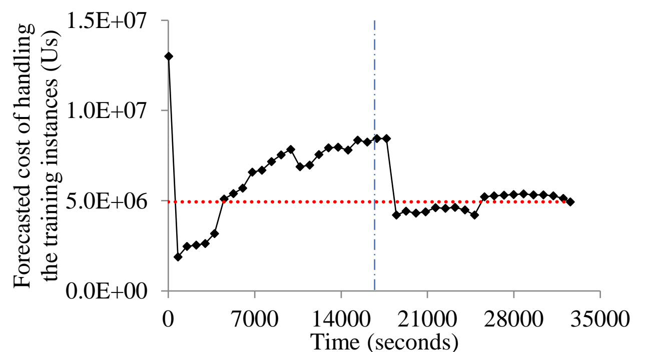


FIGURE A43. Cost of handling the training instances forecasted over time (employing the step decay approach to control the learning rate as well as SGD to train the ResNet50 model).

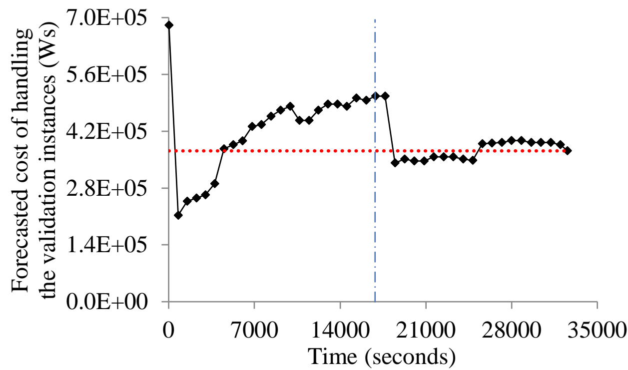


FIGURE A44. Cost of handling the validation instances forecasted over time (employing the step decay approach to control the learning rate as well as SGD to train the ResNet50 model).

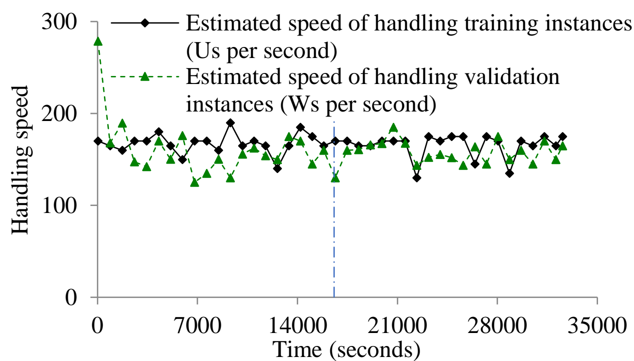


FIGURE A45. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the step decay approach to control the learning rate as well as SGD to train the ResNet50 model).

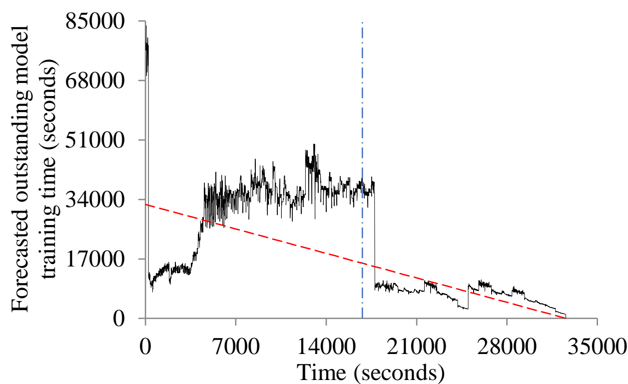


FIGURE A46. Outstanding model training time forecasted by our new progress estimation method (employing the step decay approach to control the learning rate as well as SGD to train the ResNet50 model).

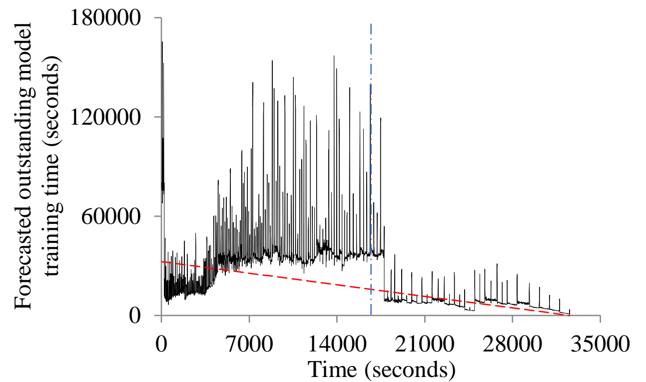


FIGURE A47. Outstanding model training time forecasted by our former progress estimation method (employing the step decay approach to control the learning rate as well as SGD to train the ResNet50 model).

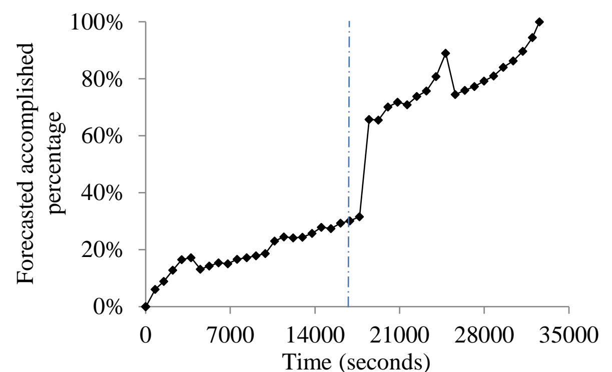


FIGURE A48. Accomplished percentage forecasted over time (employing the step decay approach to control the learning rate as well as SGD to train the ResNet50 model).

Employing AdaGrad

This test employed the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model. Fig. A49-A54 display the experimental results, which resemble those displayed in Fig. A43-A48.

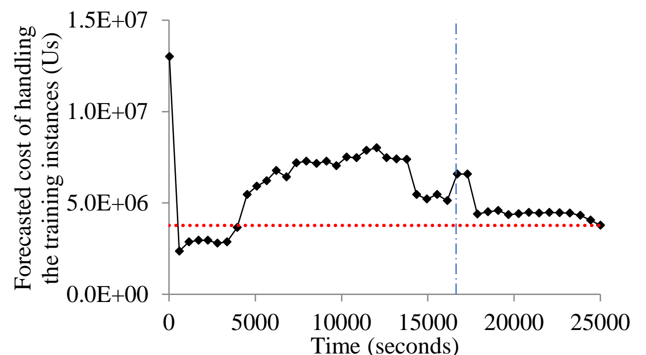


FIGURE A49. Cost of handling the training instances forecasted over time (employing the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

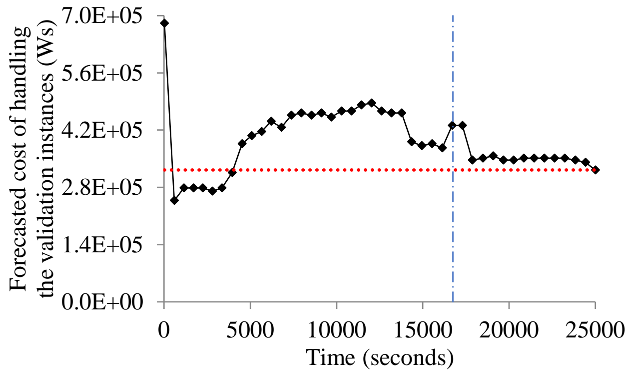


FIGURE A50. Cost of handling the validation instances forecasted over time (employing the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

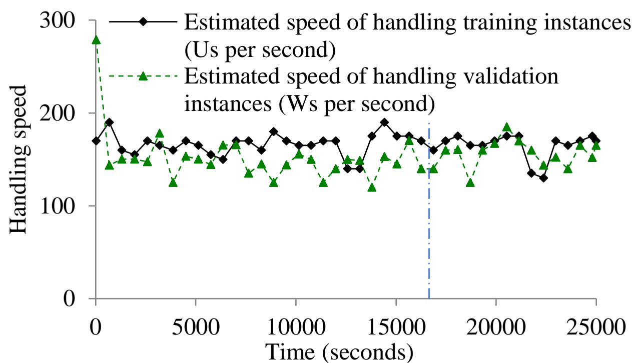


FIGURE A51. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

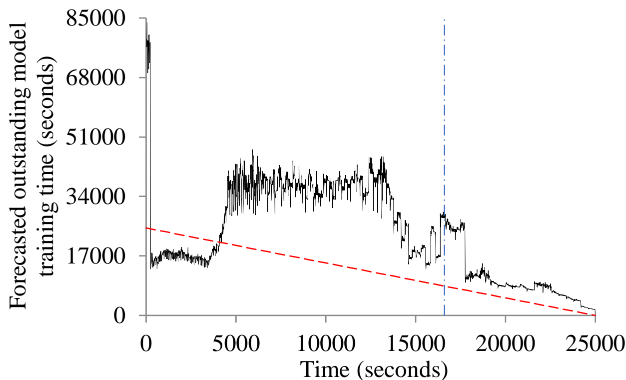


FIGURE A52. Outstanding model training time forecasted by our new progress estimation method (employing the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

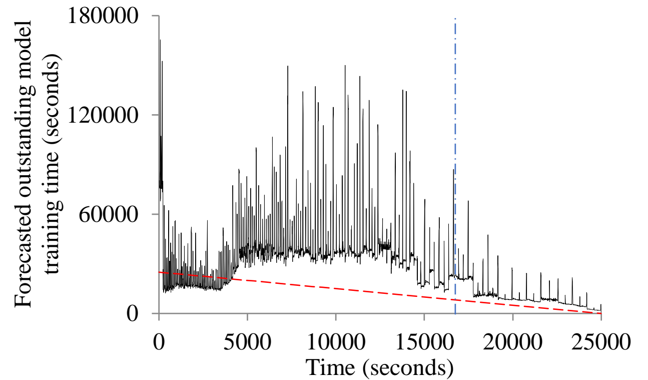


FIGURE A53. Outstanding model training time forecasted by our former progress estimation method (employing the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

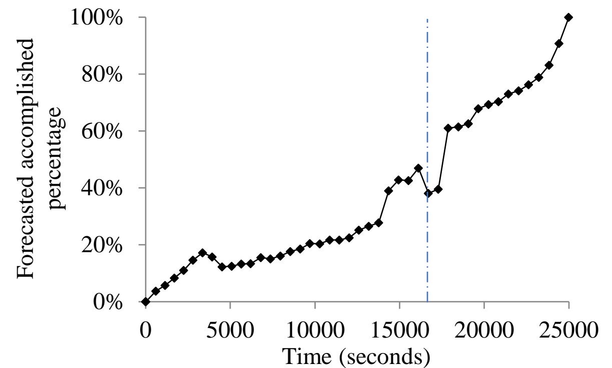


FIGURE A54. Accomplished percentage forecasted over time (employing the step decay approach to control the learning rate as well as AdaGrad to train the ResNet50 model).

B. EXPERIMENTAL RESULTS OF TRAINING THE LSTM MODEL

This test employed the exponential decay approach to control the learning rate as well as Adam to train the LSTM model. Fig. A55-A60 display the experimental results. Overall, our new progress estimation method produced relatively good estimates of the cost of handling the training instances, the cost of handling the validation instances, and the outstanding model training time. Compared to our former progress estimation method, our new progress estimation method provided more stable estimates of the outstanding model training time.

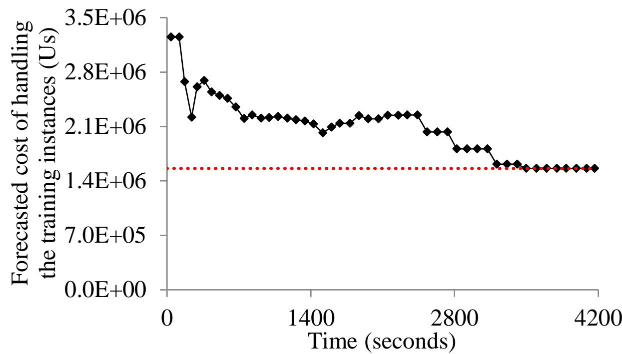


FIGURE A55. Cost of handling the training instances forecasted over time (employing the exponential decay approach to control the learning rate as well as Adam to train the LSTM model).

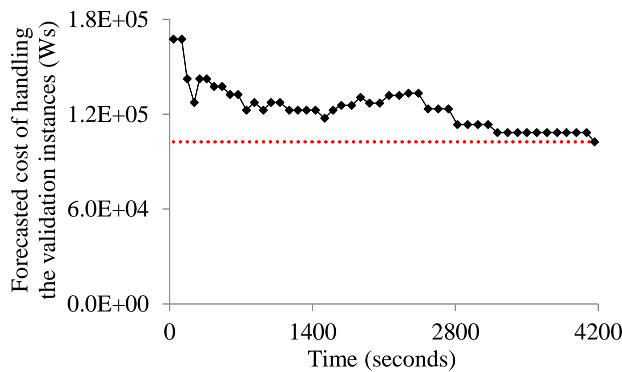


FIGURE A56. Cost of handling the validation instances forecasted over time (employing the exponential decay approach to control the learning rate as well as Adam to train the LSTM model).

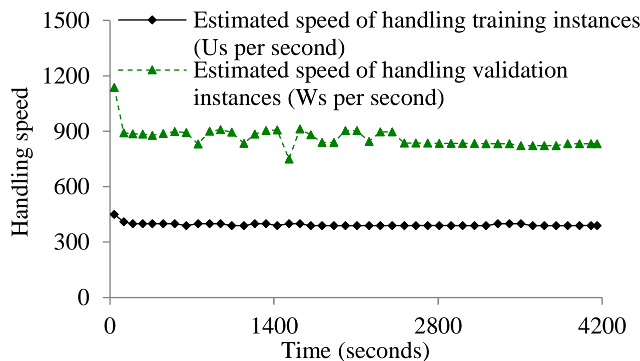


FIGURE A57. The speed of handling training instances and the speed of handling validation instances estimated over time (employing the exponential decay approach to control the learning rate as well as Adam to train the LSTM model).

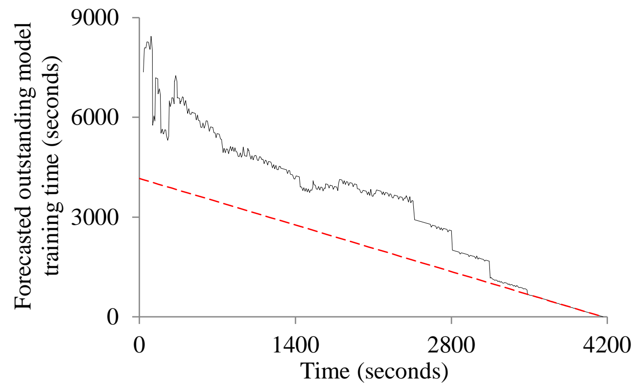


FIGURE A58. Outstanding model training time forecasted by our new progress estimation method (employing the exponential decay approach to control the learning rate as well as Adam to train the LSTM model).

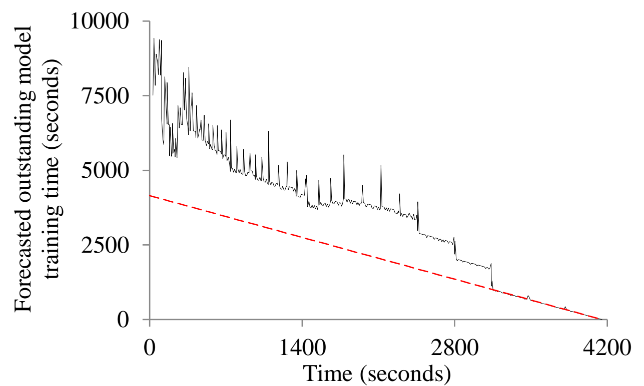


FIGURE A59. Outstanding model training time forecasted by our former progress estimation method (employing the exponential decay approach to control the learning rate as well as Adam to train the LSTM model).

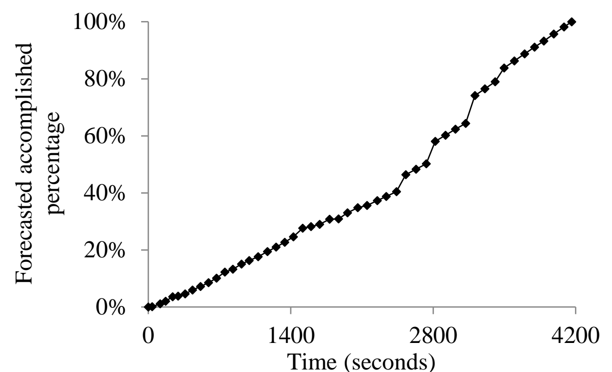


FIGURE A60. Accomplished percentage forecasted over time (employing the exponential decay approach to control the learning rate as well as Adam to train the LSTM model).

ACKNOWLEDGMENT

We thank Brian Kelly for useful discussions.

AUTHORS' CONTRIBUTIONS

QD participated in the study design, wrote the initial draft of the paper and the computer code, and did the literature review and the experiments. GL participated in the study design and

rewrote the entire paper. Both authors read and approved the final version of the paper.

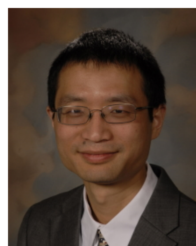
REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[2] C. Li. "OpenAI's GPT-3 language model: a technical overview." Lambda. https://lambdalabs.com/blog/demystifying-gpt-3 (accessed Dec. 10, 2023).
[3] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54-63, Dec. 2020.
[4] K. Ni, R. A. Pearce, K. Boakye, B. Van Essen, D. Borth, B. Chen, and E. X. Wang, "Large-scale deep learning on the YFCC100M dataset," 2015, *arXiv: 1502.03409*.
[5] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171-4186.
[6] J. Hui. "How to scale the BERT training with Nvidia GPUs?" Nvidia. https://medium.com/nvidia-ai/how-to-scale-the-bert-training-with-nvidia-gpus-c1575e8eaf71 (accessed Dec. 10, 2023).
[7] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proc. ICCV*, 2017, pp. 843-852.
[8] G. Luo, "Toward a progress indicator for machine learning model building and data mining algorithm execution: a position paper," *SIGKDD Explorations*, vol. 19, no. 2, pp. 13-24, Dec. 2017.
[9] G. Luo, J. F. Naughton, and P. S. Yu, "Multi-query SQL progress indicators," in *Proc. EDBT*, 2006, pp. 921-941.
[10] Q. Dong and G. Luo, "Progress indication for deep learning model training: a feasibility demonstration," *IEEE Access*, vol. 8, pp. 79811-79843, Apr. 2020.
[11] Q. Dong, X. Zhang, and G. Luo, "Improving the accuracy of progress indication for constructing deep learning models," *IEEE Access*, vol. 10, pp. 63754-63781, Jun. 2022.
[12] A. Klimovic. "Rethinking data storage and preprocessing for ML." ACM SIGARCH. https://www.sigarch.org/rethinking-data-storage-and-preprocessing-for-ml (accessed Dec. 10, 2023).
[13] D. G. Murray, J. Šimša, A. Klimovic, and I. Indyk, "tf.data: a machine learning data processing framework," *Proc. VLDB Endow.*, vol. 14, no. 12, pp. 2945-2958, Jul. 2021.
[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211-252, Dec. 2015.
[15] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. CVPR*, 2018, pp. 8697-8710.
[16] S. Bianco, R. Cadène, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270-64277, Oct. 2018.
[17] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: a system for large-scale machine learning," in *Proc. OSDI*, 2016, pp. 265-283.
[18] "tf.keras.callbacks.EarlyStopping." TensorFlow. https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/keras/callbacks/EarlyStopping (accessed Dec. 10, 2023).
[19] A. Kuznetsova, H. Rom, N. Alldrin, J. R. R. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari, "The Open Images Dataset V4," *Int. J. Comput. Vis.*, vol. 128, no. 7, pp. 1956-1981, Jul. 2020.
[20] "Open Images Dataset V7 and extensions." https://storage.googleapis.com/openimages/web/index.html (accessed Dec. 10, 2023).
[21] "Open Images Dataset." GitHub. https://github.com/cvdfoundation/open-images-dataset (accessed Dec. 10, 2023).
[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: an imperative style, high-performance deep learning library," in *Proc. NeurIPS*, 2019, pp. 8024-8035.
[23] M. A. Franklin and T. Pan, "Clocked and asynchronous instruction pipelines," in *Proc. MICRO*, 1993, pp. 177-184.
[24] "tf.data: build TensorFlow input pipelines." TensorFlow. https://www.tensorflow.org/guide/data (accessed Dec. 10, 2023).
[25] "NVIDIA data loading library." Nvidia. https://developer.nvidia.com/dali (accessed Dec. 10, 2023).
[26] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "CutMix: regularization strategy to train strong classifiers with localizable features," in *Proc. ICCV*, 2019, pp. 6022-6031.
[27] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: beyond empirical risk minimization," in *Proc. ICLR*, 2018.
[28] X. Zhang, J. J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *NIPS*, 2015, pp. 649-657.
[29] A. Shekhar. "ImageNet100: a sample of ImageNet classes." Kaggle. https://www.kaggle.com/datasets/ambityga/imagenet100 (accessed Dec. 10, 2023).
[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770-778.
[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735-1780, 1997.
[32] "bert/bert-en-uncased-l-8-h-256-a-4." kaggle/Models. https://www.kaggle.com/models/tensorflow/bert/frameworks/tensorFlow2/variations/bert-en-uncased-l-8-h-256-a-4/versions/2 (accessed Dec. 10, 2023).
[33] "TensorFlow Hub." Tensorflow. https://www.tensorflow.org/hub (accessed Dec. 10, 2023).
[34] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. ICLR*, 2019.
[35] "tf.data.Dataset/shuffle." Tensorflow. https://www.tensorflow.org/api_docs/python/tf/data/Dataset#shuffle (accessed Dec. 10, 2023).
[36] "bert/en_uncased_preprocess." Kaggle. https://www.kaggle.com/models/tensorflow/bert/frameworks/tensorFlow2/variations/en-uncased-preprocess/versions/1 (accessed Dec. 10, 2023).
[37] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*, 2010, pp. 177-186.
[38] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.
[39] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Proc. ICLR*, 2015.
[40] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121-2159, 2011.
[41] "tf.keras.applications.resnet50.ResNet50." TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50 (accessed Dec. 10, 2023).
[42] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, "Estimating progress of execution for SQL queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 803-814.
[43] G. Luo, T. Chen, and H. Yu, "Toward a progress indicator for program compilation," *Softw.: Pract. and Experience*, vol. 37, no. 9, pp. 909-933, July 2007.
[44] K. Wang, H. Converse, M. Gligoric, S. Misailovic, and S. Khurshid, "A progress bar for the JPF search using program executions," in *Proc. Java PathFinder Workshop at ESEC/FSE*, 2018.
[45] W. Lee, H. Oh, and K. Yi, "A progress bar for static analyzers," in *Proc. SAS*, 2014, pp. 184-200.
[46] G. Luo, "PredicT-ML: a tool for automating machine learning model building with big clinical data," *Health Inf. Sci. Syst.*, vol. 4, Article 5, Dec. 2016.
[47] G. Luo, B. L. Stone, M. D. Johnson, P. Tarczy-Hornoch, A. B. Wilcox, S. D. Mooney, X. Sheng, P. J. Haug, and F. L. Nkoy, "Automating construction of machine learning models with clinical big data: proposal rationale and methods," *JMIR Res. Protoc.*, vol. 6, no. 8, pp. e175, Aug. 2017.

[48] K. Morton, M. Balazinska, and D. Grossman, “ParaTimer: a progress indicator for MapReduce DAGs,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 507-518.
[49] K. Morton, A. L. Friesen, M. Balazinska, and D. Grossman, “Estimating the progress of MapReduce pipelines,” in *Proc. IEEE Int. Conf. Data Eng.*, 2010, pp. 681-684.
[50] K. Lee, A. C. König, V. R. Narasayya, B. Ding, S. Chaudhuri, B. Ellwein, A. Eksarevskiy, M. Kohli, J. Wyant, P. Prakash, R. V. Nehme, J. Li, and J. F. Naughton, “Operator and query progress estimation in Microsoft SQL Server Live Query Statistics,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 1753-1764.
[51] G. Luo, J. F. Naughton, C. J. Ellmann, and M. Watzke, “Increasing the accuracy and coverage of SQL progress indicators,” in *Proc. IEEE Int. Conf. Data Eng.*, 2005, pp. 853-864.
[52] G. Luo, J. F. Naughton, C. J. Ellmann, and M. Watzke, “Toward a progress indicator for database queries,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 791-802.
[53] X. Xie, Z. Fan, B. Choi, P. Yi, S. S. Bhowmick, and S. Zhou, “PIGEON: progress indicator for subgraph queries,” in *Proc. IEEE Int. Conf. Data Eng.*, 2015, pp. 1492-1495.
[54] G. Luo, “Progress indication for machine learning model building: a feasibility demonstration,” *SIGKDD Explorations*, vol. 20, no. 2, pp. 1-12, Dec. 2018.
[55] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, “Predicting the computational cost of deep learning models,” in *Proc. BigData*, 2018, pp. 3873-3882.
[56] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in *Proc. NIPS*, 2012, pp. 2960-2968.
[57] T. Doan and J. Kalita, “Predicting run time of classification algorithms using meta-learning,” *Int. J. Mach. Learn. and Cybern.*, vol. 8, no. 6, pp. 1929-1943, Dec. 2017.
[58] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell, “OBOE: collaborative filtering for AutoML model selection,” in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2019, pp. 1173-1183.
[59] M. Reif, F. Shafait, and A. Dengel, “Prediction of classifier training time including parameter optimization,” in *Proc. KI*, 2011, pp. 260-271.
[60] R. Livni, S. Shalev-Shwartz, and O. Shamir, “On the computational efficiency of training neural networks,” in *Proc. NIPS*, 2014, pp. 855-863.
[61] L. L. Fredenslund. “Computational complexity of neural networks.” https://lunalux.io/computational-complexity-of-neural-networks (accessed Dec. 10, 2023).
[62] M. Anthony and P. L. Bartlett, Neural Network Learning: Theoretical Foundations. New York, NY, USA: Cambridge Univ. Press, 2002.
[63] S. Chaudhuri, R. Kaushik, and R. Ramamurthy, “When can we trust progress estimators for SQL queries?” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 575-586.
[64] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig, “Early stopping without a validation set,” 2017, *arXiv: 1703.09580*.
[65] D. Duvenaud, D. Maclaurin, and R. P. Adams, “Early stopping as nonparametric variational inference,” in *Proc. AISTATS*, 2016, pp. 1070-1077.
[66] L. Prechelt, “Early stopping-but when?” in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 1996, pp. 55-69.

QIFEI DONG received the B.S. degree in electrical engineering from Zhejiang University, Hangzhou, Zhejiang Province, P.R. China, in 2016 and the M.S. degree in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2018. He is currently pursuing the PhD degree in biomedical informatics and medical education at the University of Washington, Seattle, WA, USA.

Since 2018, he has been a Research Assistant with the University of Washington Clinical Learning, Evidence and Research Center for Musculoskeletal Disorders, Seattle, WA, USA. His research interests include machine learning, computer vision, natural language processing, and clinical informatics.



GANG LUO received the B.S. degree in computer science from Shanghai Jiaotong University, Shanghai, P.R. China, in 1998, and the PhD degree in computer science from the University of Wisconsin-Madison, Madison, WI, USA, in 2004.

From 2004 to 2012, he was a Research Staff Member at IBM T.J. Watson Research Center, Hawthorne, NY, USA. From 2012 to 2016, he was an Assistant Professor in the Department of Biomedical Informatics at the University of Utah, Salt Lake City, UT, USA. He is currently a Professor in the Department of Biomedical Informatics and Medical Education at the University of Washington, Seattle, WA, USA. He is the author of over 90 papers. His research interests include machine learning, information retrieval, database systems, and health informatics.