Exploring Optimization in Deep Learning with Various Algorithms

Jiahua Chen, Keying Chen, Jiaru Li, Sijia Li, University of Toronto

Abstract — The world has been fighting against COVID-19 for more than two years. Given the urgency of early and accurate detection of COVID-19 cases, deep learning is a promising technique for identifying COVID-19 cases and preventing the infection from spreading. This project aims to use the DenseNet121 architecture, a dense convolutional neural network, to diagnose COVID-19 patients from chest X-ray images. The project will compare the performances of various optimization algorithms, such as SGD, Adam, and their variants. Through tuning different sets of hyperparameters for each optimizer, applying early stopping based on validation error during training, and using Binary Cross Entropy as the loss function, the best non-adaptive and adaptive optimization algorithms are selected according to validation and test errors. They are SGD with Nesterov Momentum and Adamax. A trade-off between convergence speed and classification error for the two best performing optimizers is observed. Compared to Adamax, SGD with Nesterov Momentum has superior performance in terms of validation and test accuracies, but has a slower convergence speed. Moreover, our results show that SGD with Nesterov Momentum has the highest test accuracy of 94%, arriving at a different conclusion compared to other papers performing similar optimizer evaluations on COVID-19 chest X-ray images.

This project implementation can be found at:
https://github.com/lisijia6/Exploring-Optimization-in-Deep-Learning-with-Various-Algorithms

Index Terms—B.1.4.d Optimization, I.1.2.e Performance Evaluation of Algorithms and Systems, I.2.1.c Decision Support, I.2.6.g Machine Learning

1 Introduction

The COVID-19 pandemic has largely impacted people's lives. There is an increasing need to build a deep learning model that can quickly diagnose COVID-19 with a high accuracy as a cross-check method to avoid mis-diagnosis. Additionally, early and accurate detection of COVID-19 can be a promising technique to track and prevent the infection from spreading, for example, by isolating the positive patients detected. Convolutional neural network (CNN) architectures, such as DenseNet121, have been leveraged in prior studies [1] to perform the binary classification task on predicting COVID-19 patients using the chest X-ray images.

Since there is no single optimizer that outperforms others on all problems, we conduct a series of computational experiments to compare some commonly-used optimization algorithms with different hyperparameter combinations, and assess their performance both in terms of convergence speed (time and epochs) and testing accuracy on the COVID-19 classification problem. The optimization algorithms selected are Stochastic Gradient Descent (SGD), SGD with Momentum, SGD with Nesterov Momentum, SGD with Learning Rate Dropout, Adam, Adam with Learning Rate Dropout, Adam with L2 Regularization, Adam with Decoupled Weight Decay, and Adamax.

By comparing the performances of various optimization algorithms with different hyperparameter combinations, we can determine the optimizer and hyperparameter settings that work the best for the binary classification problem to predict COVID-19 patients from other pneumonia patients using the chest X-ray datasets. In addition, computational experiments will also allow us to understand empirically how different hyperparameters will affect the model performance, such as convergence time and testing accuracy.

2 Related Work

2.1 Deep Learning for Classification of COVID-19 Chest X-ray Images

There is a boom of papers exploring artificial intelligence aided images analysis as a promising alternative for identifying COVID-19 in 2020. Specifically, they use chest X-ray images and CNN models to predict COVID-19 cases.

Chauhan, Tavishee et al. uses DenseNet-121 and applies transfer learning to classify the normal-healthy images from COVID-19 images [1]. Comparative analysis of multiple optimizers, loss functions, and schedulers is performed to get the highest accuracy for the proposed system. The Adamax optimizer with Cross Entropy loss function and StepLR scheduler has the best performance with 98% accuracy for both normal-healthy X-ray images and COVID-19 images.

Another paper by Roberts, Michael et al. lists some common pitfalls and recommendations for leveraging machine learning to detect and forecast for COVID-19 using chest radiographs and computed tomography images [2]. Their search identifies 2,212 papers uploaded from Jan 1 to Oct 3, 2020 on this topic, but it turns out that none of the studies identified are of potential clinical use due to methodological flaws and/or underlying biases. Therefore, they stress the importance of having higher quality datasets (e.g. a reasonable number of participants or events-per-variable to lower the risk of bias), manuscripts with sufficient documentation to be reproducible (e.g. the image pre-processing method), and external validation (e.g. using test data from different sources than development) to increase the likelihood of models being taken forward and integrated into future clinical trials.

2.2 Optimization Algorithms in Deep Learning

In deep learning, optimization algorithms are used to minimize the loss function or the difference between the true labels and the model's predicted labels. In other words, the goal is to find a set of weights for the neural network to make accurate predictions, and hence, leading to a lower value of the loss function. Many gradient descent optimization algorithms have been developed to achieve this. Some algorithms are non-adaptive, meaning that the learning rates are fixed in the weight update rule. For example, SGD, SGD with momentum, SGD with Nesterov momentum, and SGD with learning rate dropout [3]. Other algorithms are adaptive with varying learning rates, such as Adam, Adam with learning rate dropout, AdamL2, AdamW, and Adamax [3, 4, 5]. However, there is no single optimizer that performs the best for all data. The choice of optimizer should depend on the input data and the neural network architecture. For instance, adaptive learning rate methods may be more suitable for training a deep and complex neural network and reaching convergence faster.

3 Datasets

3.1 Binary COVID-19 Dataset Description

The binary COVID-19 dataset used in this project is a combination of two sources. The first dataset is the COVID-19 Image Data Collection Project initiated by Dr. Joseph Paul Cohen, Paul Morrison, and Lan Dao in 2020. The COVID-19 ChestXray dataset consists of X-ray and CT scans of patients who are positive or suspected of COVID-19 or other viral and bacterial cases of pneumonia [5]. As of April 2022, there are 504 images of positive COVID-19 X-rays and 173 images of other diseases, as shown in Table 1. The metadata file in the COVID-19 ChestXray dataset is used to retrieve the images, and the "finding" field is used to match the images with disease labels, and the "modality" file is used to retrieve X-ray scans from all images.

Table 1 Number of X-ray Images of Various Diseases in the COVID-19 ChestXray Dataset

Disease	# of Images	Disease	# of Images
Covid 19	504	Mycoplasma	11
SARS	16	Influenza	5
Pneumocystis	30	Tuberculosis	18
Streptococcus	22	Aspergillosis	2
Chlamydophila	3	Herpes	3
E.Coli	4	Aspiration	1
Klebsiella	10	Nocardia	8
Legionella	10	MERS-CoV	10
Lipoid	13	MRSA	1
Varicella	6		

The second dataset is from a Kaggle competition, it consists of 5,863 X-ray images with two categories: Healthy or Pneumonia. There are 1,586 healthy X-ray images and 4,276 X-ray images that are diagnosed with Pneumonia [6].

Similar to a prior study [7], the binary COVID-19 dataset used in this project is built to distinguish COVID-19 X-ray images from X-ray images representing other diseases as well as healthy cases. To create a balanced negative COVID-19 class, 173 images of diseases other than COVID-19 are randomly sampled from the first data source, and 173 healthy X-ray images are randomly sampled from the second data source. Similarly, 346 COVID-19 X-ray images are selected to create a balanced COVID-19 positive class.

3.2 Data Preparation

The data processing method in this project is similar to a prior study using similar datasets [7]. As described in Section 3.1, the balanced binary dataset consists of 346 COVID-19 positive images and 346 COVID-19 negative images. In order to increase the size of the dataset as well as to improve generalization, various data augmentation techniques are applied. For each image within the original balanced dataset, two of the techniques from Table 2 are randomly selected and applied to the image. Finally, each image is resized to a 180 x 180 tensor object.

The final pre-processed COVID-19 dataset contains two classes: COVID-19 positive and COVID-19 negative. The dataset is divided into training, validation, and testing sets at the ratio of 70:15:15. The final dataset has a total of 1,038 images (726 for training, 156 for validation, and 156 for testing) for each class.

Table 2 Augmentation Techniques and Their Ranges

Augmentation Technique	Range
Brightness	[0.5, 1), (1, 1.5]
Rotation	(0, 15]
Vertical Flip	True
Horizontal Flip	True

4 Methods

4.1 Model Architecture

Transfer learning is a technique that uses a pre-trained network for other tasks [8]. Our model uses the pre-trained DenseNet121 architecture for feature extraction and then applies the transfer learning technique to classify COVID-19 X-ray images, as shown in Figure 1. The dense convolutional network architecture ensures the maximum information flow between layers in the network by connecting all layers directly with each other. Compared to other models such as CNN and ResNet, dense networks require much fewer parameters and can reduce overfitting on tasks with small training datasets.

In terms of architecture, DenseNet121 [9] begins with one convolutional and one pooling layer. After the first two layers, there are 3 dense blocks each followed by a transition layer. After the last transition layer, there is a fourth dense block followed by a classifier to make the final prediction. Each dense block consists of two convolutional layers (with 1x1 and 3x3 kernels) repeated 6, 12, 24, and 16 times for dense blocks 1, 2, 3, and 4, respectively. Each transition layer has one 1x1 convolutional layer and one 2x2 average pooling layer with a stride of 2.

After extracting features from DenseNet121, a new classifier is defined for our model to replace the classifier in DenseNet121. Our new classifier has two fully-connected layers. To introduce non-linearity, the ReLU activation function is applied after each convolutional and fully-connected layer, except for the last layer. Since COVID-19 X-ray image classification is a binary classification problem, the Binary Cross Entropy Loss function is used to train the model.

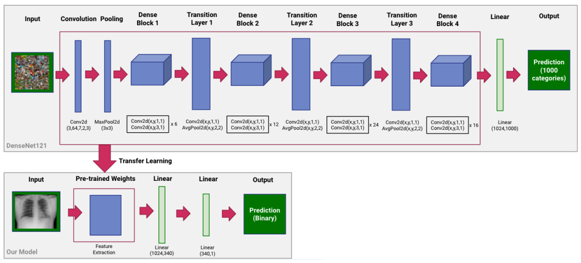


Figure 1. Model Architecture [9]

4.2 SGD and Its Variants

4.2.1 SGD

The first optimizer used in this project is SGD, an iterative method for optimizing an objective function with appropriate smoothness properties, such as differentiable or subdifferentiable. It is a stochastic approximation of gradient descent (GD) optimization, since it replaces the actual gradient calculated from the entire data set by an estimate that is calculated from a randomly selected subset of the data (Figure 2). SGD reduces the very high computational burden, especially in high-dimensional optimization problems, achieving faster iterations in trade for a lower convergence rate. However, because it takes and iterates one random subset of the data at a time, it tends to result in more noise than GD (Figure 3).

```
Algorithm 1 Stochastic Gradient Descent
  Require: Learning rate $\eta$.
  Require: Initial parameter $\theta$.
  while Stopping criterion not met do
      Sample a minibatch of $m$ examples from the training set $\{x^{(1)}, \dots, x^{(m)}\}$.
      Set $g = 0$
      for $i = 1$ to $m$ do
          Compute gradient estimate:
              $g \leftarrow g + \nabla_\theta L(f(x^{(i)}; \theta)), y^{(i)}; \theta)$.
      end for
      Apply update: $\theta \leftarrow \theta - \eta g$
  end while
```

Figure 2. SGD Algorithm [10]

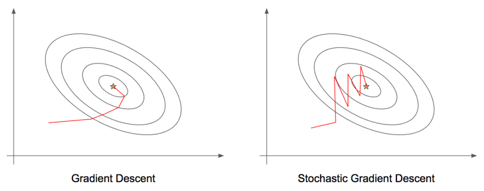


Figure 3. GD vs SGD Convergence Trajectory [11]

4.2.2 SGD with Momentum (Standard and Nesterov)

The original SGD has trouble navigating ravines, the areas where the surface curves are much steeper in one dimension than in another. Ravine can be a common case around local optima. Momentum is a widely used method that can help SGD converge faster and reduce oscillations by combining the update vector from the previous step with the current update vector. The momentum term μ is usually set to around 0.9 [12].

There are two commonly used momentum computation methods: standard momentum and Nesterov momentum. The standard momentum method first computes the gradient at the current location and then takes a big jump in the direction of the updated accumulated gradient. Whereas the Nesterov momentum method first makes a big jump in the direction of the previously accumulated gradient (by SGD with standard momentum), and then measures the gradient where it ends up and makes a correction [10]. Figure 4 illustrates the vector visualization for the two methods. Typically, Nesterov momentum helps the SGD move in a more precise manner. However, in the stochastic gradient case, Nesterov momentum does not improve the rate of convergence [12].

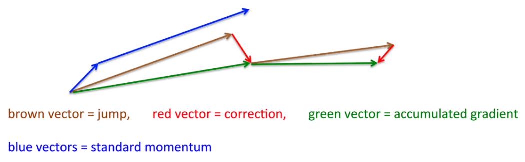

brown vector = jump,
red vector = correction,
green vector = accumulated gradient
blue vectors = standard momentum

Figure 4. Vector Visualization for Momentum Methods [12]

4.2.3 SGD with Learning Rate Dropout (LRD)

Learning Rate Dropout (LRD) is another technique that can be applied to the SGD optimizer [5]. At each iteration, each parameter has a probability to be dropped (i.e., the learning rate of the parameter is set to zero). Therefore, the corresponding weight parameter is not updated in that iteration. Standard dropout regularizes the hidden unit and the dropped unit does not appear in both forward and back propagation during training, but for LRD, the dropped learning rate does not affect forward and back propagation.

4.3 Adam and Its Variants

4.3.1 Adam

In the previous subsection, the non-adaptive learning rate optimization algorithms require setting a hyperparameter learning rate before training the model. However, setting the learning rate hyperparameter can be challenging. Hence, adaptive learning rate methods are developed by researchers to automatically tune the learning rate after initialization.

One such optimizer is Adam, which adapts the learning rate to parameters [4]. It keeps track of two exponentially decaying averages, one for past squared gradients (m_t) and one for past gradients (v_t). To avoid the bias towards zero that may result from initializing m_t and v_t to zero vectors, the vectors are adjusted to $\hat{m}_t$ and $\hat{v}_t$, given by equations (1) and (2). g_t is the vector of gradients with respect to the parameter vector at time step t. During each update step in (3), the learning rate is divided by the square root of $\hat{v}_t$ at each time step, hence for the "adaptive learning rate". ε in the denominator is a very small number to avoid division by zero.

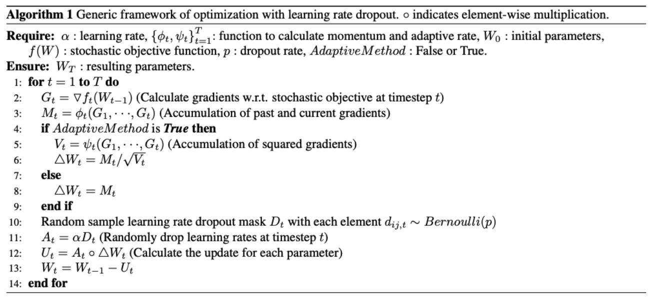
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad \hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{1}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \tag{3}$$

4.3.2 Adam with Learning Rate Dropout (LRD)

Similar to SGD with LRD, the LRD technique can also be applied to Adam. In Adam parameter updates, some learning rates are randomly set to zero for making no updates to some parameters. In particular, this technique is effective in helping Adam avoid suboptimal points and produce a better convergence result. This is because bad initialization may lead to a bad local minimum. With LRD, Adam will continue to look for other possible descent paths even if the optimizer has already reached a local minimum, therefore, able to escape from potentially bad local minima. Figure 5 shows the pseudocode for the generic framework of optimization with LRD [5].



Algorithm 1 Generic framework of optimization with learning rate dropout. $\circ$ indicates element-wise multiplication.

```
Require: $\alpha$ : learning rate, $\{\phi_t, \psi_t\}_{t=1}^{T}$: function to calculate momentum and adaptive rate, $W_0$ : initial parameters,
    $f(W)$ : stochastic objective function, $p$ : dropout rate, $AdaptiveMethod$ : False or True.
Ensure: $W_T$ : resulting parameters.
for $t = 1$ to $T$ do
    $G_t = \nabla f_t(W_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep $t$)
    $M_t = \phi_t(G_1, \cdots, G_t)$ (Accumulation of past and current gradients)
    if $AdaptiveMethod$ is True then
        $V_t = \psi_t(G_1, \cdots, G_t)$ (Accumulation of squared gradients)
        $\triangle W_t = M_t / \sqrt{V_t}$
    else
        $\triangle W_t = M_t$
    end if
    Random sample learning rate dropout mask $D_t$ with each element $d_{ij,t} \sim Bernoulli(p)$
    $A_t = \alpha D_t$ (Randomly drop learning rates at timestep $t$)
    $U_t = A_t \circ \triangle W_t$ (Calculate the update for each parameter)
    $W_t = W_{t-1} - U_t$
end for
```

Figure 5. Generic Framework of Optimization with LRD [5]

4.3.3 AdamL2

Overfitting will prevent the model from generalizing and hence will lead to a poor performance when given unseen new data. In order to prevent the model from overfitting, ℓ_2 regularization can be used.

AdamL2 is an adaptive gradient algorithm with an ℓ_2 regularization added to the loss function, as indicated in equation (4). λ, known as the regularization parameter, is the hyperparameter of the model. The larger the value of the regularization parameter is, the larger the regularization effect there will be.

$$loss = BCE + \lambda ||w||_2^2 = BCE + \lambda \sum_{i=1}^{n} w_i^2 \tag{4}$$

Since Adam is an adaptive algorithm, when updating, the sums of the gradient of the loss function and that of the ℓ_2 norm of the weights are adapted [13]. That is to say, the gradient of the regularizer will be normalized along with the gradient of the loss function by their summed magnitude. Hence, weights with varying magnitudes will be regularized with different rates.

4.3.4 AdamW

Another common strategy used to overcome overfitting is using decoupled weight decay. While for non-adaptive algorithms with fixed learning rates (such as SGD), weight decay has an equivalent impact as ℓ_2 regularization [13]. However, it is not the case for Adam with adaptive gradients. As previously mentioned, with ℓ_2 regularization, the gradient of the regularizer gets scaled with the gradient of the loss function. The decouple weight decay, on the other hand, regularizes all weights with the same rate λ regardless of its magnitude because the weight decay step and the adaptive gradient mechanism are separated [13].

A decouple weight decay will change the process from update (5) (without weight decay) to update (6) (with weight decay) as illustrated below. The hyperparameter λ is called the weight decay coefficient, and it has a similar impact as the regularization parameter.

$$W_{t+1} \leftarrow W_t - \alpha M_t \nabla f_t(Wt) \tag{5}$$

$$W_{t+1} \leftarrow (1 - \lambda) W_t - \alpha M_t \nabla f_t(Wt) \tag{6}$$

4.3.5 Adamax

The last variant of Adam used in this project is Adamax, based on the infinity norm (ℓ_∞). Although norms for large p values (ℓ_p) usually become numerically unstable, which is why ℓ_1 and ℓ_2 norms are more commonly used in practice, the ℓ_∞ norm generally exhibits stable behavior [14]. The v_t factor in the Adam update rule using the ℓ_2 norm (4.3.1 (2)) can be generalized to the ℓ_p norm (7). To avoid confusion with Adam, u_t is used to denote the ℓ_∞ norm constrained v_t (8) and plugged into the Adam update equation by replacing the update step $\sqrt{\hat{v}_t} + \varepsilon$ with u_t to obtain the Adamax update rule (9). Since u_t relies on the max operation, it is not as susceptible to bias towards zero as m_t and v_t in Adam, and thus there is no need to compute a bias correction for u_t. Compared to Adam, Adamax is sometimes superior in models with embeddings.

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p)|g_t|^p \tag{7}$$

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty)|g_t|^\infty \\ &= \max(\beta_2 \cdot v_{t-1}, |g_t|) \end{aligned} \tag{8}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \hat{m}_t \tag{9}$$

5 Experiments

5.1 Experiment Setup

The experiments are implemented using Google Colab, and the python notebooks used can be found here. To speed up the training process by parallel computing, CUDA GPU is used. PyTorch package is the machine learning framework used for the creation and training of the Deep Neural Network in this project. Existing code from Nicklas Hansen's repository [15] is also used for the implementation of various optimization algorithms for deep learning. In order to build a neural network that can solve the COVID-19 classification problem using different optimization algorithms, the code for DenseNet121 transfer learning from Kaggle [16] and for optimizers from Nicklas Hansen's repository [15] are integrated in our experiment setup.

We use a predefined dataset split (70:15:15) as discussed in Section 3. We choose not to do cross-validation because for the deep learning model: it is very costly to train k different models [17], let alone we have in total 7 different variants of optimization algorithms to tune. Instead, we use the hold-out method by using a subset of the training dataset as a hold-out to validate the model performance. It also helps us decide the best hyperparameter combinations for different optimization algorithms.

The performance metrics measured in the experiments include the convergence epoch and time, as well as accuracies on the training, validation and testing sets. Since it is a binary classification task in the medical field, two other commonly-used metrics are also measured, which are false positive (FP) rate and false negative (FN) rate. The FP rate measures the conditional likelihood that the patients will get a wrong positive result when they are not infected in reality, and the FN rate measures the probability that an infected patient will receive a wrongly negative diagnosis result. The FN rate is of a higher importance, as the COVID-19 and its variants are highly-infectious and can spread quickly. It is essential to diagnose the positive patients and have them isolated.

5.2 Experiment Results and Discussion

The experiment results of the best models for each optimizer tested are summarized in Table 3. The "Hyperparameters Setting" column lists the hyperparameter combinations for the model with the best performance in terms of validation error. Early stopping is used to prevent overfitting of the models. The stopping condition is when the validation loss and validation classification error cease to decrease. The table also contains the classification error rates on training, validation, and testing sets, as well as the FP and FN rates on the testing set.

5.2.1 SGD and Its Variants

As shown in the Appendix, for SGD, models trained with a batch size of 32 always perform better than models trained with a batch size of 64, regardless of their learning rate.

For SGD with standard momentum, models trained by setting a smaller momentum term (0.9) outperform those models trained with a larger momentum term (0.99) across various learning rates (0.01 and 0.001) and batch sizes (32 and 64). The best performing model for this optimizer is trained with batch size = 32 and learning rate = 0.001.

For SGD with Nesterov Momentum, setting a larger momentum (0.99) with a smaller learning rate (0.001) outperforms that of a larger learning rate (0.01) across different batch sizes (32 and 64). However, setting a larger learning rate with a larger batch size performs better than that of a smaller batch size, regardless of the momentum values (0.9, 0.95, and 0.99). Thus, our best model is obtained by taking the hyperparameter combinations of batch size = 64, learning rate = 0.01, and momentum = 0.9, as seen in Table 3. Its training curves are shown in Figure 6.

Comparing the two momentum computational methods, the latter outperforms the former when other hyperparameters are set to the same when comparing the error rate on the validation set, while the convergence speed is similar for both of the momentum methods. This result is aligned with a prior study [14].

In terms of the effect of LRD, we observe faster convergence when LRD is applied. However, overall, the LRD does not improve validation error compared to other SGD optimizers without LRD. Comparing the different trials conducted with LRD, we find that 1) increasing the probability of dropping the learning rate, 2) applying a larger learning rate decreases the validation error, when other hyperparameter values are held the same. However, a larger batch size does not significantly influence the model performance.

5.2.2 Adam and Its Variants

As shown in the Appendix, for Adam, regardless of batch size, smaller learning rate always gives better performance in terms of training and validation errors. This suggests that for our problem, a learning rate of 0.01 may be too large such that the model may be learning a sub-optimal set of weights too fast. Moreover, regardless of learning rate, a smaller batch size always yields better performance. This observation is expected as smaller batch sizes can provide a regularization effect and generate lower error in unseen data [18]. These observations are also true for Adam with LRD.

For Adam with LRD, it is also observed that the combination of batch size = 32 and learning rate = 0.01 (regardless of LRD hyperparameter) will converge the fastest (in terms of number of epochs and training time) as compared to other combinations of batch size and learning rate. In addition, there is no correlation between the LRD hyperparameter with validation error. Comparing Adam and Adam with LRD, for any batch size and learning rate combination, Adam with LRD (regardless of LRD) always performs better than Adam in terms of validation error. This provides evidence that Adam with LRD is better as randomly setting some learning rates to zero can help the optimizer to escape from potentially bad local minima.

While both AdamL2 and AdamW are commonly used to avoid overfitting, they have different testing performance. As listed in Appendix, while other hyperparameters stay the same, the models with a smaller learning rate get better results than its counterpart with a larger learning rate. Tuning the regularization parameter (or weight decay coefficient for AdamW) using the validation error, there is a general trend that for AdamL2, l2_reg = 1e-5 normally works the best, while weight_decay at the default value of 1e-4 works better for AdamW. The results we obtained using a new dataset correspond to the conclusion from the previous study that the smaller the batch size is, the smaller the optimal regularization parameter is [13]. Moreover, it is discussed in prior research that AdamW generalizes better than AdamW [13], and it is also true in our case, with the exception that AdamL2 has a slightly lower FP rate.

For Adamax, when compared to Adam using the common set of hyperparameters (batch size and learning rate), Adamax's performance is superior to Adam's in terms of both convergence speed and validation error. In general, while other parameter values are held the same, a larger batch size (64) works better for Adamax. Moreover, setting the pair of betas to (0.9, 0.999) outperforms the other pairs in most cases. Finally, increasing the learning rate from 0.001 to 0.002 does not significantly affect Adamax's convergence time and validation error. The training curves of the best model of Adamax are shown in Figure 7.

5.2.3 Comparison of All SGD and Adam Models

Across models using various optimization algorithms, there is a general trend that the FP rate is higher than the FN rate as shown in Table 3. This is preferred since COVID-19 virus is highly infectious and thus any people who are potentially infected (tested FN) should be isolated as soon as possible. The models with the best performance in terms of validation and test classification errors among SGD and its variants as well as Adam and its variants are marked in green. They are

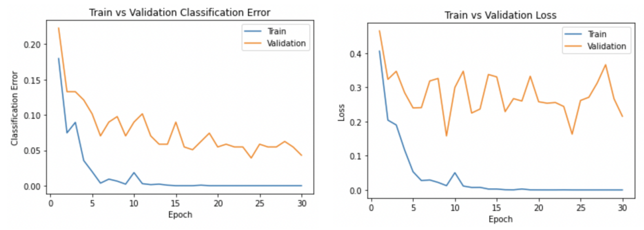


Figure 6. Training Curves of SGD with Nesterov Momentum

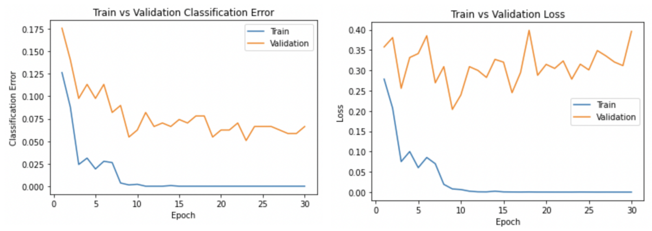


Figure 7. Training Curves of Adamax

Table 3. Experiment Results Summary

Optimizer	Hyperparameters Setting*	Convergence Epoch	Convergence Time (s)	Train Err	Val Err	Test Err	FP Rate	FN Rate
SGD	bs=32, lr=0.01	10	1098	0.001	0.108	0.104	0.087	0.017
SGD with Nesterov Momentum	**bs=64, lr=0.01, mu=0.9**	**30**	**1081**	**0**	**0.043**	**0.063**	**0.047**	**0.016**
SGD with Momentum	bs=32, lr=0.001, mu=0.9	20	908	0.002	0.063	0.101	0.079	0.021
SGD with Momentum and LRD	bs=64, lr=0.01, mu=0.9, lrd=0.6	20	701	0	0.051	0.086	0.070	0.016
Adam	bs=32, lr=0.01	20	935	0.028	0.087	0.104	0.063	0.042
Adam with LRD	bs=64, lr=0.001, lrd=0.3	30	1335	0.006	0.047	0.078	0.051	0.027
AdamL2	bs=32, lr=0.001, l2_reg=1e-5	20	955	0.015	0.080	0.097	0.065	0.031
AdamW	bs=32, lr=0.001, weight_decay=1e-4	20	1274	0.023	0.056	0.090	0.076	0.014
Adamax	**bs=64, lr=0.001, betas=(0.9, 0.999)**	**10**	**324**	**0.002**	**0.062**	**0.070**	**0.063**	**0.008**

**Hyperparameters Abbreviations: bs = batch size, lr = learning rate, mu = momentum term μ, lrd = learning rate dropout, l2_reg = L2 regularization, betas = a pair of tuples of coefficients used for computing running averages of gradient and its square*

SGD with Nesterov Momentum (NM) and Adamax. Between these two "best" optimizers, there is a small trade-off between the convergence speed and the validation error. Adamax (an adaptive algorithm) converges faster than SGD with NM (a non-adaptive algorithm). However, SGD with NM outperforms Adamax in terms of both validation and test classification errors. Looking at the FP and FN rates, SGD with NM outperforms Adamax on FP rate while Adamax outperforms SGD with NM on FN rate. Overall, if the goal is fast convergence, good generalization to unseen data, and low FN rate, Adamax may be the most suitable optimizer for this COVID-19 chest X-ray classification task. If higher generalization to unseen data is of more importance than fast convergence, SGD with NM may be the better choice in this case. Hence, neither optimizer outperforms the other in both convergence speed and classification errors.

Although several prior papers show that adaptive learning rate methods, such as Adamax, achieve better performance in deep neural networks [1], our study shows that non-adaptive learning rate methods, such as SGD with NM, when carefully tuned can outperform the former. Consequently, for any future work, it is always useful to conduct experiments for comparing and selecting the most suitable optimizer for the task of interest.

6 Conclusion and Future Work

This project examines various optimization algorithms in deep learning and evaluates the performance of the algorithms on the task of classifying COVID-19 chest X-ray images. This is a binary classification task for COVID-19 positive and negative classes. The final dataset used for this project consists of 1,038 images for each of the two classes (726 for training, 156 for validation, and 156 for testing). It is concluded that SGD with NM and Adamax have the best performances in terms of validation and test classification errors. SGD with NM has lower validation and test errors, but has a slower convergence speed compared to Adam. This shows a convergence speed and classification error trade-off between the 'best' non-adaptive algorithm (among SGD and its variants) and adaptive algorithm (among Adam and its variants). Although SGD with NM and Adamax are the 'best' algorithms for this task, other optimization algorithms may be more suitable and produce better results for other tasks. Hence, experiments should always be conducted to compare various optimizers and select the 'best' one for the task of interest.

Although it was previously discussed that FN rate is more important in the medical diagnosis field, FP diagnosis will also bring inconvenience to people and hence should also be considered. As illustrated in Section 5.2 that the FP rate is higher than the FN rate across models using different optimization algorithms, it could be an interesting future work to analyze why this is the case. Another important extension of the work could be to better understand the COVID-19 chest X-ray images dataset. For example, the loss landscape could be plotted [19]. It would show if this problem has a clear minimum, which could lead to good model performance in general. Lastly, with this research project focusing on SGD (and its variants) and Adam (and its variants), there are more promising optimization algorithms proposed in prior research. More studies could be conducted to test their performances as well.

7 End Sections

7.1 Acknowledgments

The authors wish to thank Professor Elias Khalil for the continuous feedback provided after each phase of the project as well as for expanding students' knowledge about optimization in machine learning. In addition, the authors would also like to thank the teaching assistants for hosting extremely helpful lab sessions.

7.2 Contribution of Team Members

RS = research, FD = first draft, ET = Edit for grammar and spelling, CM = compile elements into the complete document, WC = write code

Task	Jiahua C.	Keying C.	Jiaru L.	Sijia L.
Literature Review	RS	RS	RS	RS
Project Proposal	FD, ET	FD, ET	FD, ET	FD, ET
Data Preprocessing		WC	WC	
Model Architecture and Training Functions	WC			WC
Hyper- parameter Tuning	WC	WC	WC	WC
Presentation	FD, ET	FD, ET	FD, ET	FD, ET
	Final Report			
Abstract	FD	ET	ET	ET
1. Introduction	ET	ET	FD	ET
2. Related Work	FD	ET	FD	ET
3. Datasets	ET	FD	ET	ET
4. Methods	FD (4.2), ET	FD (4.2), ET	FD (4.3), ET	FD (4.1, 4.3), ET
5. Experiments	FD (5.2.1), ET	FD (5.2.1), ET	FD (5.1, 5.2.2), ET	FD (5.2.2, 5.3), ET
6. Conclusion and Future Work	ET	ET	FD	FD
7. End Sections	FD, ET	FD, ET	FD, ET	FD, ET
All Sections	CM	CM	CM	CM

7.3 References

[1] T. Chauhan, H. Palivela, and S. Tiwari, "Optimization and fine-tuning of DenseNet model for classification of COVID-19 cases in medical imaging," International Journal of Information Management Data Insights, Nov-2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8189817. [Accessed: 11-Apr-2022]

[2] M. Roberts, D. Driggs, M. Thorpe, J. Gilbey, M. Yeung, S. Ursprung, A. I. Aviles-Rivero, C. Etmann, C. McCague, L. Beer, J. R. Weir-McCall, Z. Teng, E. Gkrania-Klotsas, J. H. F. Rudd, E. Sala, and C.-B. Schönlieb, "Common pitfalls and recommendations for using machine learning to detect and prognosticate for covid-19 using chest radiographs and CT scans," *arXiv.org*, 05-Jan-2021. [Online]. Available: https://arxiv.org/abs/2008.06388. [Accessed: 18-Apr-2022].

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv.org*, 15-Jun-2017. [Online]. Available: https://arxiv.org/abs/1609.04747. [Accessed: 18-Apr-2022].

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv.org*, 30-Jan-2017. [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed: 18-Apr-2022].

[5] H. Lin, W. Zeng, X. Ding, Y. Huang, C. Huang, and J. Paisley, "Learning rate dropout," *arXiv.org*, 05-Dec-2019. [Online]. Available: https://arxiv.org/abs/1912.00144. [Accessed: 18-Apr-2022].

[6] P. Mooney, "Chest X-ray images (pneumonia)," *Kaggle*, 24-Mar-2018. [Online]. Available: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia. [Accessed: 18-Apr-2022].

[7] D. Ezzat, A. E. Hassanien, and H. A. Ella, "An optimized deep learning architecture for the diagnosis of COVID-19 disease based on gravitational search optimization," *Applied Soft Computing*, 22-Sep-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494620306803?casa_token=4ieLFywpVeUAAAAA%3APeXRbdBj46bl9y-0SZOOE30EekImtXEMDC592HLOaOSNjKVMLBh5a9-MKeVbrATUJE8RwrSl#b42. [Accessed: 18-Apr-2022].

[8] D. (D. J. Sarkar, "A comprehensive hands-on guide to transfer learning with real-world applications in Deep learning," *Medium*, 17-Nov-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a. [Accessed: 18-Apr-2022].

[9] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks ," *arxiv.org*, 2018. [Online]. Available: https://arxiv.org/pdf/1608.06993.pdf%C2%A0. [Accessed: 18-Apr-2022].

[10] "An overview on optimization algorithms in deep learning 1," Taihong Xiao, 04-Feb-2016. [Online]. Available: https://prinsphield.github.io/posts/2016/02/overview_opt_alg_deep_learning1/. [Accessed: 11-Apr-2022].

[11] "Stochastic gradient descent vs gradient descent: A head-to-head comparison," SDS Club. [Online]. Available: https://sdsclub.com/stochastic-gradient-descent-vs-gradient-descent-a-head-to-head-comparison/. [Accessed: 18-Apr-2022].

[12] G. Hinton, N. Srivastava, and K. Swersky, "Lecture 6a Overview of mini-batch gradient descent," *Lecture Slides - cs.toronto.edu*. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec9.pdf. [Accessed: 18-Apr-2022].

[13] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arxiv.org*, 04-Jan-2019. [Online]. Available: https://arxiv.org/abs/1711.05101. [Accessed: 18-Apr-2022].

[14] S. Ruder, "An overview of gradient descent optimization algorithms," *Sebastian Ruder*, 20-Mar-2020. [Online]. Available: https://ruder.io/optimizing-gradient-descent/index.html#adamax. [Accessed: 18-Apr-2022].

[15] N. Lashansen, "Nicklashansen/neural-net-optimization: PYTORCH implementations of recent optimization algorithms for deep learning.," *GitHub*, 2019. [Online]. Available: https://github.com/nicklashansen/neural-net-optimization. [Accessed: 18-Apr-2022].

[16] Leigh, "Densenet121: Pytorch," *DenseNet121 | pytorch*, 01-Jul-2019. [Online]. Available: https://www.kaggle.com/code/leighplt/densenet121-pytorch/notebook. [Accessed: 18-Apr-2022].

[17] V. Lyashenko and A. Jha, "Cross-validation in Machine Learning: How To Do It Right," *neptune.ai*, 18-Mar-2022. [Online]. Available: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right. [Accessed: 18-Apr-2022].

[18] J. Brownlee, "How to control the stability of training neural networks with the batch size," *Machine Learning Mastery*, 27-Aug-2020. [Online]. Available: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/. [Accessed: 18-Apr-2022].

[19] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *arXiv.org*, 07-Nov-2018. [Online]. Available: https://arxiv.org/abs/1712.09913. [Accessed: 18-Apr-2022].

7.4 Appendix

The appendix contains the detailed experiment results for the optimizers and their performance under various hyperparameter settings.

Table A1. Performance of SGD and Its Variants

Optimizer	BS	LR	Other Hyperparameters	Convergence Epoch	Convergence Time (sec)	Train Error	Validation Error
SGD	32	0.01	None	10	1098	0.00139	0.10764
	32	0.001	None	30	1332	0.03472	0.15278
	64	0.01	None	20	715	0	0.11328
	64	0.001	None	30	1078	0.15909	0.19531
SGD with Nesterov Momentum	32	0.01	mu = 0.99	30	1371	0.02778	0.11806
	32	0.001	mu = 0.99	30	1368	0.00069	0.05208
	64	0.01	mu = 0.99	20	723	0.01491	0.11328
	64	0.001	mu = 0.99	20	724	0.00355	0.05859
	32	0.01	mu = 0.95	30	1985	0.00069	0.06597
	32	0.01	mu = 0.9	20	909	0.00069	0.08681
	32	0.001	mu = 0.95	20	908	0.00069	0.0625
	32	0.001	mu = 0.9	30	1359	0	0.07639
	64	0.01	mu = 0.95	20	719	0.00142	0.06250
	64	0.01	mu = 0.9	30	1081	0	0.04297
	64	0.001	mu = 0.95	20	1094	0	0.10547
	64	0.001	mu = 0.9	20	737	0	0.10938
SGD with Momentum	32	0.01	mu = 0.8	20	905	0.00069	0.07300
	32	0.001	mu = 0.8	30	1357	0.00069	0.10069
	32	0.01	mu = 0.9	20	902	0.00625	0.09028
	32	0.001	mu = 0.9	20	908	0.00208	0.06250
	32	0.01	mu = 0.99	30	1493	0.06458	0.11806
	32	0.001	mu = 0.99	25	1130	0.00208	0.08333
	64	0.01	mu = 0.8	25	905	0.0	0.09375
	64	0.01	mu = 0.9	20	724	0.0	0.07422
	64	0.001	mu = 0.9	15	449	0.0	0.11719
	64	0.01	mu = 0.99	20	724	0.00284	0.08594
SGD with Momentum & LRD	32	0.01	mu = 0.9, lrd = 0.4	20	913	0.00069	0.08333
	32	0.001	mu = 0.9, lrd = 0.4	16	1022	0.00417	0.12847
	32	0.01	mu = 0.9, lrd = 0.5	15	687	0.0	0.07639
	32	0.001	mu = 0.9, lrd = 0.5	18	748	0.00069	0.10069
	32	0.01	mu = 0.9, lrd = 0.6	27	1239	0.0	0.05208
	32	0.001	mu = 0.9, lrd = 0.6	16	704	0.00139	0.09722
	64	0.01	mu = 0.9, lrd = 0.4	10	533	0.00426	0.08203
	64	0.001	mu = 0.9, lrd = 0.4	20	700	0.01492	0.13281
	64	0.01	mu = 0.9, lrd = 0.5	20	702	0.0	0.07031
	64	0.001	mu = 0.9, lrd = 0.5	25	875	0.0	0.14844
	64	0.01	mu = 0.9, lrd = 0.6	20	701	0.0	0.05078
	64	0.001	mu = 0.9, lrd = 0.6	25	871	0.0	0.12891

Table A2. Performance of Adam and Its Variants

Optimizer	BS	LR	Other Hyperparameters	Convergence Epoch	Convergence Time (sec)	Train Error	Validation Error
Adam	32	0.01	None	40	2176	0.04514	0.14236
	32	**0.001**	**None**	**20**	**935**	**0.02847**	**0.08681**
	64	0.01	None	30	1083	0.11577	0.13281
	64	0.001	None	20	729	0.00994	0.08984
Adam with LRD	32	0.01	lrd = 0.3	40	1049	0.04722	0.10417
	32	0.01	lrd = 0.4	40	928	0.03750	0.09375
	32	0.01	lrd = 0.5	40	929	0.025	0.13541
	32	0.01	lrd = 0.6	50	1161	0.01181	0.11458
	32	0.001	lrd = 0.3	20	466	0.00694	0.05556
	32	0.001	lrd = 0.4	20	466	0.01667	0.07292
	32	0.001	lrd = 0.5	30	702	0.00139	0.07986
	32	0.001	lrd = 0.6	30	703	0.00139	0.06597
	64	0.01	lrd = 0.3	40	785	0.01420	0.11719
	64	0.01	lrd = 0.4	40	781	0.02912	0.10547
	64	0.01	lrd = 0.5	40	780	0.03906	0.08203
	64	0.01	lrd = 0.6	30	1507	0.07670	0.10156
	64	**0.001**	**lrd = 0.3**	**30**	**1335**	**0.00639**	**0.04688**
	64	0.001	lrd = 0.4	30	1341	0.00639	0.07031
	64	0.001	lrd = 0.5	40	843	0.00608	0.06771
	64	0.001	lrd = 0.6	30	650	0.00710	0.05469
AdamL2	32	0.01	l2_reg = 1e-3	20	966	0.20625	0.22569
	32	0.001	l2_reg = 1e-3	50	3046	0.01806	0.10069
	32	0.01	l2_reg = 1e-4	20	1287	0.19514	0.19444
	32	0.001	l2_reg = 1e-4	30	1995	0.01111	0.10416
	32	0.01	l2_reg = 1e-5	50	3047	0.07361	0.14931
	32	**0.001**	**l2_reg = 1e-5**	**20**	**955**	**0.01458**	**0.07986**
	64	0.01	l2_reg = 1e-3	30	1112	0.15554	0.17578
	64	0.001	l2_reg = 1e-3	30	1340	0.02201	0.09766
	64	0.01	l2_reg = 1e-4	20	734	0.19034	0.21875
	64	0.001	l2_reg = 1e-4	20	1082	0.02486	0.10938
	64	0.01	l2_reg = 1e-5	30	1115	0.15270	0.12891
	64	0.001	l2_reg = 1e-5	10	372	0.02628	0.09766
AdamW	32	0.01	weight_decay = 1e-3	20	950	0.19653	0.22222
	32	0.001	weight_decay = 1e-3	40	1892	0.03333	0.08333
	32	0.01	weight_decay = 1e-4	20	1291	0.16181	0.20486
	32	**0.001**	**weight_decay = 1e-4**	**20**	**1274**	**0.02292**	**0.05555**
	32	0.01	weight_decay = 1e-5	40	1894	0.06042	0.14236
	32	0.001	weight_decay = 1e-5	15	639	0.13164	0.125
	64	0.01	weight_decay = 1e-3	30	1339	0.14631	0.14453
	64	0.001	weight_decay = 1e-3	10	383	0.05184	0.125
	64	0.01	weight_decay = 1e-4	40	1453	0.03835	0.12111
	64	0.001	weight_decay = 1e-4	30	1094	0.00497	0.05859
	64	0.01	weight_decay = 1e-5	30	1087	0.12784	0.16406
	64	0.001	weight_decay = 1e-5	10	381	0.02344	0.08984

Adamax	32	0.001	betas = (0.9, 0.999)	20	959	0.00139	0.07986
	32	0.001	betas = (0.9, 0.995)	20	781	0.01042	0.08681
	32	0.001	betas = (0.95, 0.999)	10	391	0.00625	0.07639
	32	0.001	betas = (0.85, 0.999)	20	779	0.00694	0.09028
	64	**0.001**	**betas = (0.9, 0.999)**	**20**	**647**	**0**	**0.0625**
	64	0.001	betas = (0.9, 0.995)	20	646	0.00213	0.0742
	64	0.001	betas = (0.95, 0.999)	20	646	0.00071	0.06859
	64	0.001	betas = (0.85, 0.999)	20	646	0.00426	0.07813
	32	0.002	betas = (0.9, 0.999)	20	533	0.00764	0.08681
	32	0.002	betas = (0.9, 0.995)	30	523	0.00069	0.09028
	32	0.002	betas = (0.95, 0.999)	20	349	0.01181	0.08681
	32	0.002	betas = (0.85, 0.999)	20	348	0.01042	0.07639
	64	0.002	betas = (0.9, 0.999)	10	166	0.00994	0.07422
	64	0.002	betas = (0.9, 0.995)	10	165	0.01491	0.07422
	64	0.002	betas = (0.95, 0.999)	20	329	0.00781	0.08984
	64	0.002	betas = (0.85, 0.999)	10	165	0.01705	0.07813