Applied Machine Learning Mini Project 3 Report

Abstract

In this project we implemented two machine learning models, Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) and investigated their classification performances on an image dataset the CIFAR-10. The performances under different hyperparameter sets were compared. Model hyperparameters that we tuned include learning rate, batch size, activation function, network depth for MLP, and number of layers, optimizer and activation function for CNN. Our experimental results show that even though train accuracy is close to perfection for both models after a certain number of train epochs, the highest test accuracy is only 55.73% for MLP after 200 train epochs and 79.56% for CNN after 100 train epochs.

1 Introduction

In this project, two classification models — Multilayer Preceptron (MLP) and Convolutional Neural Network (CNN) are implemented from scratch and with *PyTorch*[1] respectively, on the *CIFAR-10* [2] image dataset for image classification task. In the following part, the basic ideas for these two models are introduced.

1.1 Multilayer Preceptron (MLP)

Multilayer Preceptron (MLP) is a class of feedforward artificial neural network (ANN). A MLP consists of an input layer, a certain number of hidden layers and an output layer. Each node in the hidden layer and output layer is a neuron that uses a nonlinear activation function, which distinguishes MLP from linear perceptron. Every layer except the input layer has weight matrix and bias as its parameters. MLP utilizes a supervised learning technique called backpropagation for training or learning its parameters. A MLP with a single hidden layer is represented graphically in Figure 1.

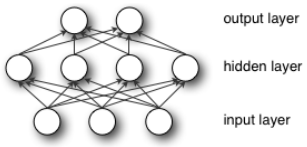


Figure 1: A MLP with a single hidden layer

1.2 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) specializes in analyzing visual images and videos. CNN is a version of multilayer perceptron (MLP) which employs convolution operations in hidden layers. Convolution operation is a special matrix multiplication, which simplifies image complexity by feeding different kernel filters (normally a n by n matrix for 2D images, where $1 \leq n \leq 10$), making images easier to process without losing critical features for predictions and also reducing the possibility of overfitting.

1.2.1 Residual Neural Network (ResNet)

Residual Neural Network, one of the CNN architectures, is built out of residual block, which enables CNN to train the dataset deeper than regular CNN by creating identity shortcut connection that skips over one or multiple layers. The formal definition of the residual block is given as

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + \mathbf{x}. \tag{1}$$

and the visualized structure is shown in Figure 2. Further, a visual demonstration of ResNet with 34 layers is shown in Figure 3.

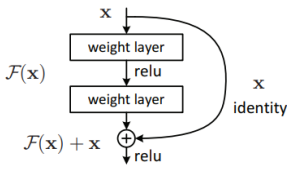


Figure 2: Residual learning: a building block (He et al., 2015a)

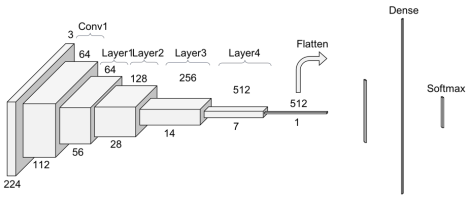


Figure 3: The architecture of ResNet-34 (He et al., 2015a)

The rest of this report is organized as follows: Section 2 gives brief descriptions of the dataset and how we preprocessed data, Section 3 introduces our experimental approaches. Section 4 reports experimental results, and finally in Section 5 we summarized our findings and provided future directions.

2 Dataset

The CIFAR-10 dataset consists of 60,000 32x32 colour images labeled in 10 classes. Every 6000 images are classified as *airplane, automobile, bird, cat, deer, dog, frog, horse, ship* and *truck*. Every instance has the size of 3x32x32. The image dataset, whose original range is [0,1], was normalized into range [-1,1]. The dataset contains 50000 train images, which we randomly split them into train set (95%) and validation set (5%) and 10000 test images. So far, the highest test accuracy 96.53% on this dataset is achieved in (Graham, 2014) using CNN.

[1] https://pytorch.org/

[2] https://www.cs.toronto.edu/ kriz/cifar.html

3 Proposed Approach

3.1 Multilayer Perceptron

A fully connected MLP was implemented from scratch. The output layer is a 10-class classification task. A softmax function was used to calculate the probability of each class and the standard cross-entropy was considered as loss. Back-propagation and mini-batch gradient descent algorithm was used for updating or learning weights and biases for all hidden layers plus output layer. Initially, a 6-hidden layer MLP architecture was employed and the size of each hidden layer is 1000, 500, 200, 100, 50, 20. For weight and bias initialization, Kaiming initialization was adopted (He et al., 2015b), where biases are all initialized to 0 and weights are generated by a normal distribution with the mean of 0 and the variance of the reciprocal of input size multiplied by 2. We set the initial activation function for all hidden layers, learning rate and batch size as ReLU, 0.1 and 128 respectively. The maximal number of training epoch was set as 200. For more implementation details please refer to our code.

3.2 Convolutional Neural Network

In order to understand the PyTorch implementation of CNN, we first adopted sample codes from the tutorial[3] with line by line comments. Then, we modified the codes to implement ResNet. We first compared performances on three ResNets with 18, 50 and 152 layers, and all three ResNets were applied with the same optimizer and activation function which are Adam and ReLU respectively. The learning rate for Adam was set as 0.001 and batch size as 128. After this process, we chose one ResNet architecture which has the best and the most stable performance and applied another two different optimizers (SGD and Adagrad) and one additional activation function Sigmoid to the chosen ResNet. The architectures of ResNet with 18, 50 and 152 layers are given by the table in Figure 4.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4: Architectures for ResNet with difference numbers of layers (He et al., 2015a).

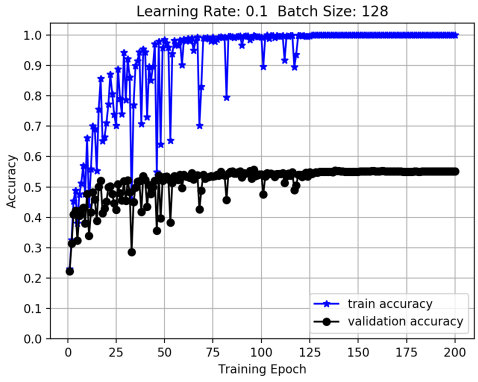
[3] https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

4 Results

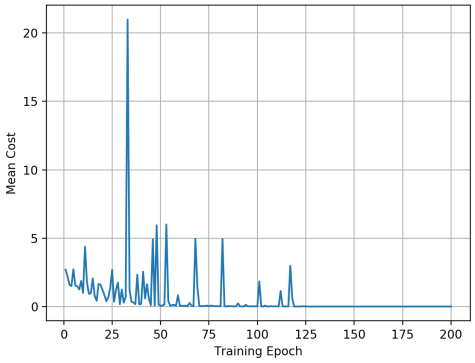
4.1 Multilayer Perceptron

4.1.1 Model Verification

The train and validation accuracy under initial hyperparameter settings after each training epoch are illustrated in Figure 5a. The mean cost (cross entropy) curve is shown in Figure 5b. Our MLP model is verified since after a certain number of training epochs, the mean cost converges to 0 and train accuracy tends to perfection. It can also be seen that both train and validation accuracy remain the same after roughly 125 training epochs.



(a) Train and validation accuracy.



(b) Mean Cost.

Figure 5: The accuracy and mean cost for MLP under initial hyperparameter settings (Learning rate = 0.1, Batch size = 128, Activation function = ReLU).

4.1.2 Learning Rate

In this part, we set batch size as 128, activation function as ReLU and investigated the effects of different learning rates on validation accuracy. The validation accuracy under several chosen learning rates (0.01, 0.02, 0.05, 0.1 and 0.2) after 200 training epochs is shown in Table 1. The train and validation accuracy curves under different learning rates are plotted in Appendix A. It can be seen from Table 1 that for

learning rate from 0.01 to 0.1, there is no significant difference among their validation accuracy, which is roughly 0.55. However, for learning rate 0.2, the validation accuracy is only 0.104 which is close to random guess because such learning rate is too large for our mini-batch gradient descent algorithm to converge to optimal solution. Appendix A states that smaller learning rate can make prediction reach stable state faster or within less training epochs. Since the highest validation accuracy is achieved with the learning rate of 0.02, such learning rate was adopted for the following experiments.

Table 1: Validation accuracy under different learning rates.

Learning Rate	Validation Accuracy
0.01	0.548
0.02	0.560
0.05	0.552
0.1	0.551
0.2	0.104

4.1.3 Batch Size

We fixed learning rate at 0.02 and activation function as ReLU and investigated the effects of different batch sizes on validation accuracy. The validation accuracy under several chosen batch sizes (32, 64, 128, 256 and 512) after 200 training epochs is shown in Table 2. The train and validation accuracy curves under different batch sizes are plotted in Appendix B. The results in Table 2 show that for batch size from 32 to 512, there is no significant difference among their validation accuracy. Also, Appendix B gives that smaller batch size makes the training process converge faster or in other words, within less training epochs. Since the highest validation accuracy is achieved when the batch size is 128, such batch size was adopted for the following experiments.

Table 2: Validation accuracy under different batch sizes.

Batch Size	Validation Accuracy
32	0.554
64	0.550
128	0.560
256	0.536
512	0.530

4.1.4 Activation Function

Another nonlinear activation function called leaky ReLU was tried here. Leaky ReLU is considered as an attempt to fix the 'dying ReLU' problem. For positive input leaky ReLU is the same as ReLU. But for negative input, leaky ReLU has a small learnable slope γ. The initial value of γ was set as 0.1. The change of the learnable slope γ for the first hidden layer's leaky ReLU was plotted in Figure 6.

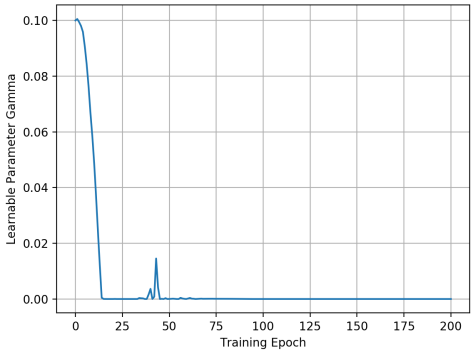
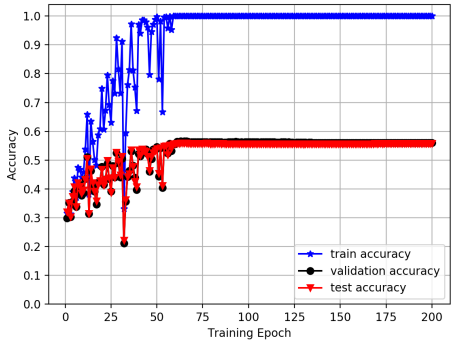
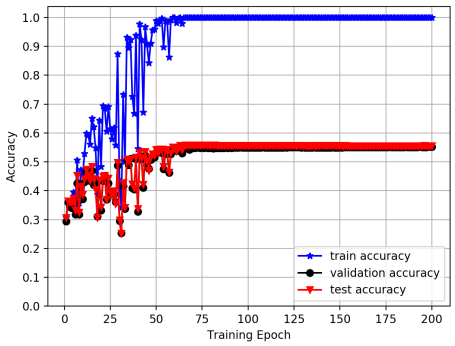


Figure 6: The change of leaky ReLU's learnable slope γ with the number of training epochs.

It can be seen that after some number of training epochs, γ converges to 0, which makes leaky ReLU beacome regular ReLU. The accuracy for MLP under the optimal hyperparameter set with ReLU and leaky ReLU as activation function was shown in Figure 7a and Figure 7b respectively.



(a) ReLU activation function.



(b) Leaky ReLU activation function.

Figure 7: The accuracy for MLP under the optimal hyperparameter set (Learning rate = 0.02, Batch size = 128) and different activation functions.

The test accuracy is very close to validation accuracy, which proves that validation accuracy is a good estimate of generalization error. The test accuracy after 200 training epochs is 0.557 and 0.556 for ReLU and leaky ReLU. There is no significant difference between ReLU and leaky ReLU probably because after the training process reaches convergence, the negative slope of leaky ReLU remains at 0, making it the same as ReLU.

4.1.5 Network Depth

In this part we investigated the influences of network depth on prediction accuracy. Our initial MLP model has 6 hidden layers. We fixed learning rate, batch size and activation function at their previous optimal values, 0.02, 128 and ReLU respectively. To reduce possible interference of the change of network width, we deleted our MLP model's last hidden layer successively and we can get a set of MLP models with 6, 5, 4, 3 and 2 hidden layers. For example, we deleted the initial model's last layer, whose size is 20, and got a MLP model with 5 hidden layers. The prediction accuracy after 200 train epochs for this set of MLP models with different number of hidden layers is shown in Table 3. It can be seen that less deeper MLP or less number of hidden layers can result in worse test accuracy but the difference is not significant. When the number of hidden layers decreases from 6 to 2, the test accuracy only drops by 3.4%.

Table 3: Prediction accuracy for a set of MLP models with different number of hidden layers.

# Hidden Layers	Validation Accuracy	Test Accuracy
2	0.538	0.523
3	0.551	0.543
4	0.545	0.550
5	0.546	0.553
6	0.560	0.557

4.2 Convolutional Neural Network

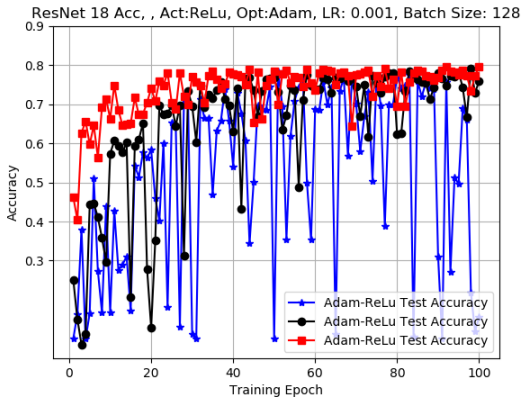
4.2.1 Performance Comparisons of Three ResNet Architectures

The performance results of ResNet with 18, 50 and 152 layers are shown in Appendix C. The test accuracy curves and the changes of mean cost for these three ResNet architectures are shown in Figure 8. The blue, black, and red lines represent ResNet with 152, 50 and 18 layers respectively. Table 4 provides the final train, validation and test accuracy after 100 training epochs.

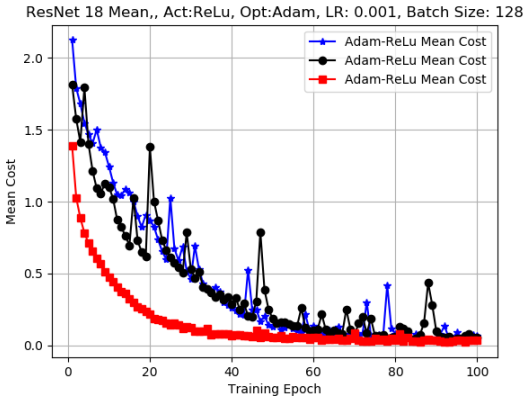
Apparently, ResNet with 18 layers has the best and the most stable performance than others, also its mean costs drop relatively faster than the other two architectures. Even though eventually all three ResNets achieve mean cost below 0.1, ResNet-18 is the first one. It is reasonable because less layers can contribute to faster convergence. Since the prediction accuracy for ResNet-152 is rather unstable over 100 training epochs, we conclude that more layers for ResNet cannot necessarily give better results. For the following experiments, ResNet with 18 layers was adopted.

Table 4: Three ResNets achieved accuracy after 100 epochs.

# Layers	Train Acc.	Validation Acc.	Test Acc.
18	0.9911	0.7986	0.7956
50	0.9668	0.7585	0.7579
152	0.1625	0.1643	0.1561



(a) ResNet 18, 50 and 152 overall test set accuracy.



(b) ResNet 18, 50 and 152 overall train set mean cost.

Figure 8: The test accuracy and mean costs results of ResNet with 18, 50 and 152 layers.

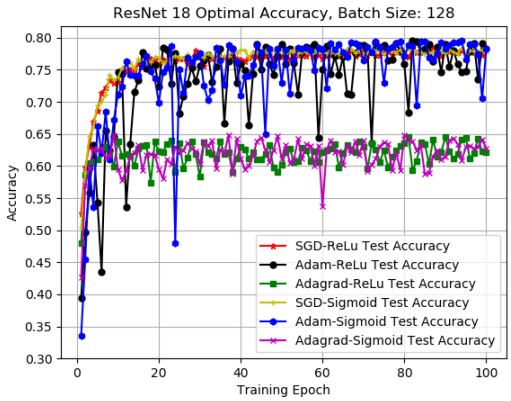
4.2.2 Performance Comparisons of ResNet-18 with Different Optimizers and Activation Functions

In this part, we investigated the influences of different optimizers and activation functions on prediction performance for ResNet-18 model. The optimizers that were tried here include Adam, Autograd and SGD. The activation functions contain ReLU and Sigmoid.

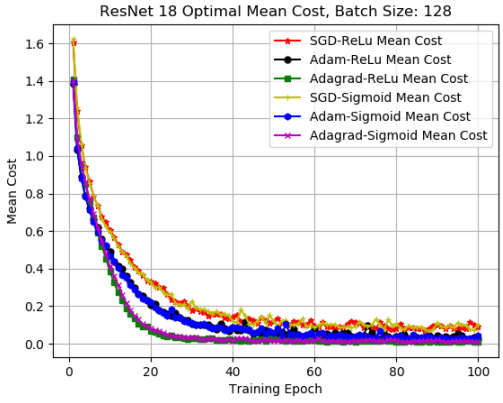
The individual performance results of ResNet-18 with three optimizers and two activation functions are shown in Appendix D Figure 13 and the final results after 100 epochs are listed in Table 5. The test accuracy curves and the changes of mean cost for all hyperparameter sets are shown in Figure 9.

Table 5: ResNet-18 achieved accuracy after 100 epochs with different settings. (The learning rate for every optimizer is set as 0.001)

Activation	Optimizer	Train Acc.	Test Acc.
ReLu	Adam	0.9911	0.7956
ReLu	Adagrad	0.9949	0.6220
ReLu	SGD	0.9828	0.7829
Sigmoid	Adam	0.9865	0.7884
Sigmoid	Adagrad	0.9811	0.6274
Sigmoid	SGD	0.9792	0.7764



(a) Test Accuracy.



(b) Mean Costs.

Figure 9: The test accuracy and mean costs results of ResNet 18 with different optimizers and activation functions.

It can be seen that compared with activation functions, optimizers make more significant differences to prediction accuracy. The train accuracy for all three optimizers tends to perfection after some train epochs. Even though Adagrad optimizer gives faster convergence on mean costs, its owns the worst validation and test accuracy, Comparing with Adam, SGD which can achieve more stable performances. In all, we conclude that SGD works better than the other two optimizers for ResNet-18 on this task.

5 Discussion and conclusion

In this work we designed and conducted a range of experiments to compare two machine learning models MLP and CNN on the CIFAR-10 image dataset. Experiment results consolidate our assumptions that CNN achieves better performance than MLP on image classification task. Besides, there exist great differences between train and test accuracy for both models and hence overfitting can be a common issue for both models. Since ResNet with 18 layers gives better and more stable results than ResNet with 50 and 152 layers on this classification task, it is concluded that more layers for ResNet cannot necessarily give better results. Also, we found that SGD optimizer works better for ResNet-18 model on this task.

Our conclusions can be meaningful for choosing suitable model for image classification task. Future directions could be focused on improving our model to achieve better classification accuracy comparable to that in recent literature.

6 Statement of contribution

Tianchi Ma and Qiutan Wu are responsible for Multilayer Preceptron and Shilei Lin is responsible for implementing Convolutional Neural Network. All team members contributed to report writing.

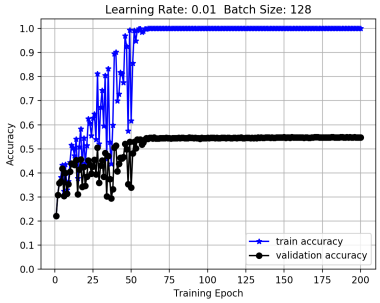
References

Benjamin Graham. 2014. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*.

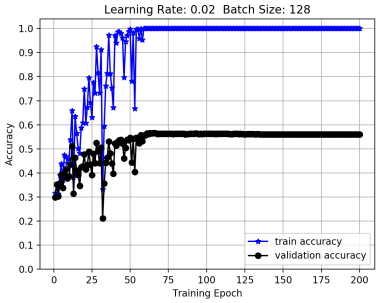
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015a. Deep residual learning for image recognition. *arXiv preprint arXiv: 1512.03385*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015b. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

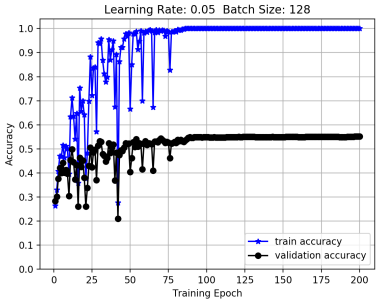
A Train and validation accuracy for MLP under different learning rates



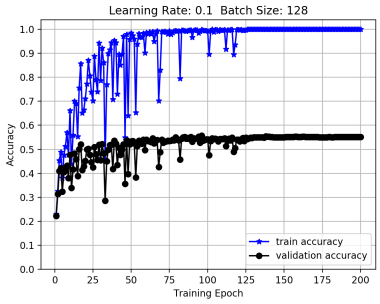
(a) Learning rate = 0.01



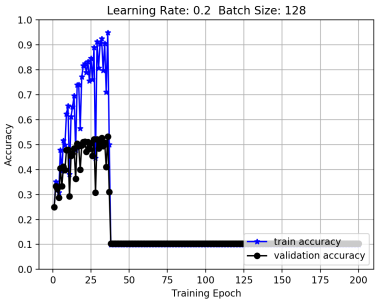
(b) Learning rate = 0.02



(c) Learning rate = 0.05



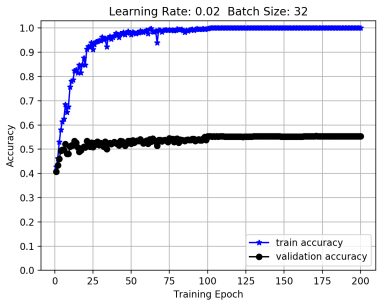
(d) Learning rate = 0.1



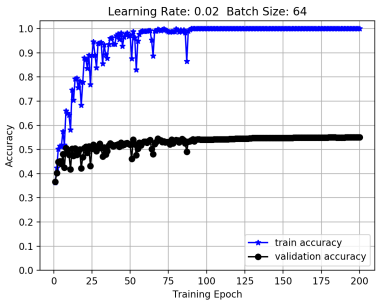
(e) Learning rate = 0.2

Figure 10: The train and validation accuracy for MLP under different learning rates with batch size 128 and activation function ReLU

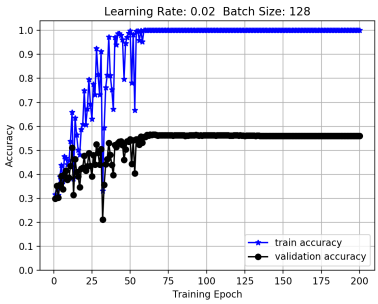
B Train and validation accuracy for MLP under different batch sizes



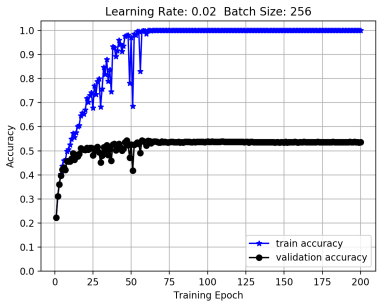
(a) Batch size = 32



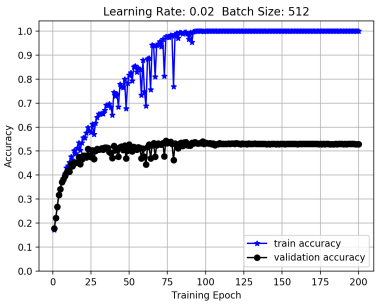
(b) Batch size = 64



(c) Batch size = 128



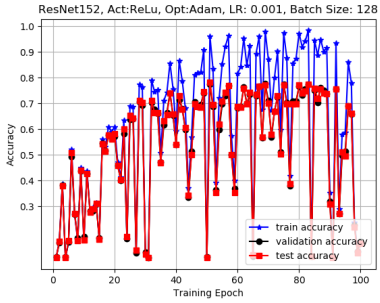
(d) Batch size = 256



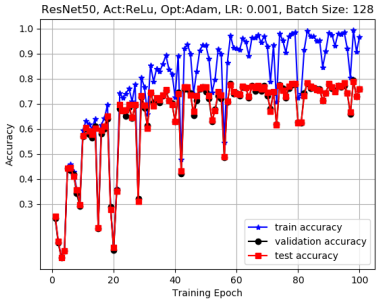
(e) Batch size = 512

Figure 11: The train and validation accuracy for MLP under different batch sizes with the fixed learning rate 0.02 and activation function ReLU

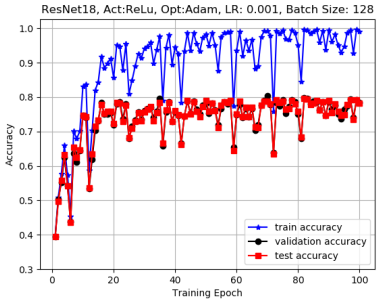
C Performances for ResNet with 18, 50 and 152 layers



(a) ResNet-152



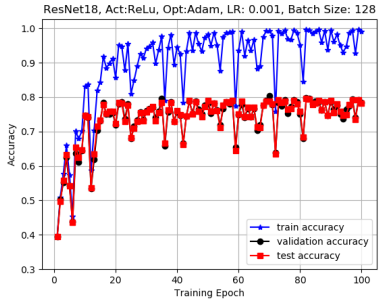
(b) ResNet-50



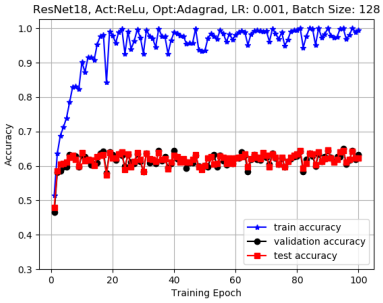
(c) ResNet-18

Figure 12: The train, validation and test accuracy for ResNet with 18, 50 and 152 layers, applied with batch size: 128, optimizers: Adam, learning rate: 0.001 and activation function: ReLU.

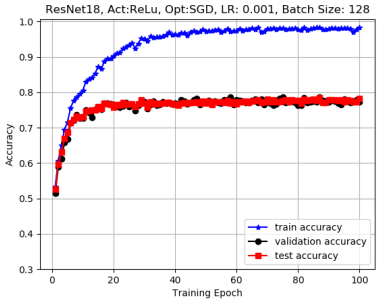
D Performances for ResNet-18 with different settings



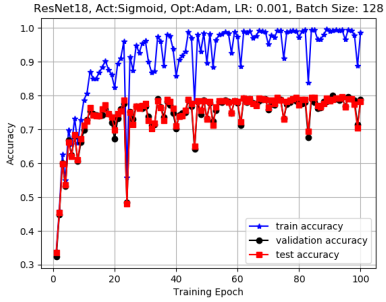
(a) Train, validation and test accuracy by using ReLU and Adam.



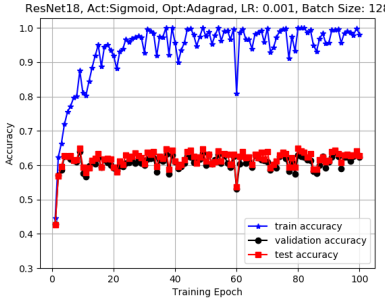
(b) Train, validation and test accuracy by using ReLU and Adgrad.



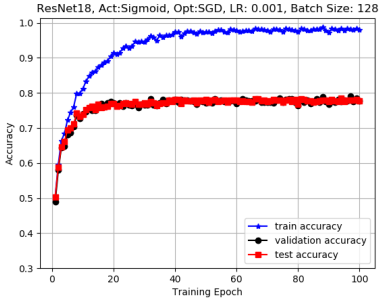
(c) Train, validation and test accuracy by using ReLU and SGD.



(d) Train, validation and test accuracy by using Sigmoid and Adam.



(e) Train, validation and test accuracy by using Sigmoid and Adgrad.



(f) Train, validation and test sets accuracy by using Sigmoid and SGD.

Figure 13: The train, validation and test accuracy for ResNet with 18 layers, applied with fixed learning rate: 0.001, fixed batch size: 128, three different optimizers: Adam, SGD and Adgrad, and two activation functions: ReLU and Sigmoid.