Optimization of Image Classification for CIFAR-10 dataset

Chen Lu
Washington University, St. Louis, MO 63130, USA
Corresponding author email: lu.chen@wustl.edu.

Abstract

Convolutional networks have shown high performance on image classification tasks, especially deep neural networks. In this project, I approached image classification problem by implementing and training from scratch one CNN model on the CIFAR-10 dataset. I tried to optimize the classification problem from three perspectives – regularization, optimization algorithm and loss function. The result showed that the image augmentation (regularization) helped to boost the accuracy a bit, but not very significantly. And the Adam optimizer outperformed sgd optimizer when we chose a very small number of epochs (<20). Lastly, in terms of loss functions, cross-entropy loss function clearly outperformed the two hinge functions in both convergence rate and accuracy. And the squared hinge function approved to be a better choice than hinge loss function.

1 Introduction

Convolutional networks(ConvNets) have enjoyed a great success in large-scale image and video recognition, and it has become popular due to large image dataset availability, such as ImageNet [1], and high-power computing systems, such as GPUs. ConvNets usually consists of by convolutional layer, pooling layer, and fully-connected Layer. These layers can be stacked to form a full ConvNet architecture [2] (Figure 1).

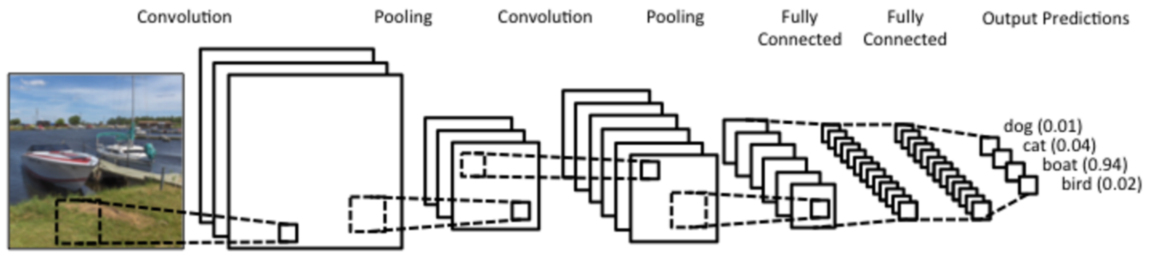


Figure 1. An example ConvNet architecture [3].

2 Related theory and practice.

2.1 Loss functions

On the last (fully-connected) layer of ConvNets, we have a loss function to measures how close the learned network is to the ground truth labels. Therefore, loss function plays a very important role in the neural network. One popular loss function used in classification is called cross-entropy (Figure 2A). In the equation, yˆi is the softmax function that takes the output scores of the network and normalizes them to have positive values. However, studies also showed that squared hinge loss (Figure 2B) and/or hinge loss (Fgiure 2C) could perform better than cross-entropy [4,5,6]. In the two equations, yk is the score of class k, which when training for a sample of class i will be 1 if k = i and -1 otherwise, and y^i is the estimated score value of class i and N the total number of classes. Tang et al found that one-vs-all squared hinge loss function reduced the error more than cross-entropy for three multi-class image classification tasks (Facial Recognition, MNIST, CIFAR-10) [4]. Zhao et al studied five different loss functions, i.e., cross-entropy, one-vs-all squared hinge loss (L2-SVM), one-vs-all-hinge loss (L1-SVM), one-vs-one hinge loss (L1-SVM) and rank loss on CIFAR-10 dataset. They concluded that L1-SVM and L2-SVM performed best and converged the quickest [5]. Therefore, it would be interesting to see the impact of loss functions in the image classification.

A
$$L = -\frac{1}{N}\sum_i y_i \log(\hat{y}_i)$$

B
$$L = \frac{1}{N}\sum_k \max(0, 1 - y_k\hat{y}_i)^2$$

C
$$L = \frac{1}{N}\sum_k \max(0, 1 - y_k\hat{y}_i)$$

Figure 2. Loss functions. A) cross-entropy B) squared hinge loss C) hinge loss [7]

2.2 Optimization Algorithm

Optimization algorithms helps us to minimize the loss function. The choice of optimization algorithm can mean huge difference in training time. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. Considering the huge computation involved in deep neural networks, stochastic gradient descent is usually used instead of gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. Sometimes developers have to manually change learning rate in the process of training. Adam is another stochastic optimization algorithm works well in practice and compares favorably to other adaptive learning-method algorithms as it converges very fast, and the learning speed of the Model is quiet Fast and efficient, and also it rectifies every problem that is faced in other optimization techniques such as vanishing Learning rate, slow convergence or High variance in the parameter updates which leads to fluctuating Loss function [8].

2.3 Data Augmentation

Data augmentation is a common way we can reduce overfitting on models for image classification problems, where we increase the amount of training data using information only in our training data. Theoretically, it helps us train more accurate Various augmentation operations could be applied to different problems, includes shifting, scaling, rotating, flipping [9, 10].

3 Dataset and Features

This project used dataset CIFAR-10. CIFAR-10 consists of 10 classes with 5,000 training and 1,000 test images each, for a total of 50,000 training and 10,000 test images. Figure 3 showed an example image for each class. Each image is an RGB triplet of size 32x32. Each pixel in the image takes a 32-bit float value in the range 0 to 1. Data augmentation was used in the form of shifting images horizontally and vertically by at most 10%, and random horizontal flipping (Figure 4). Keras provides this functionality out of the box with the ImageDataGenerator class.

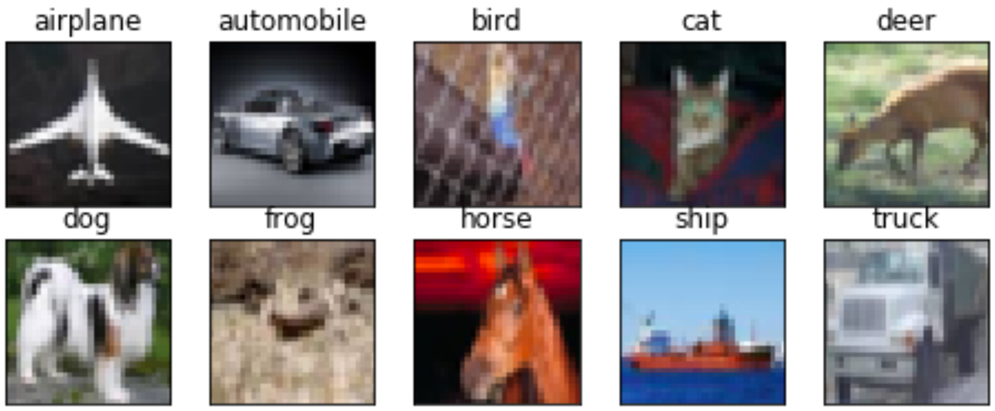
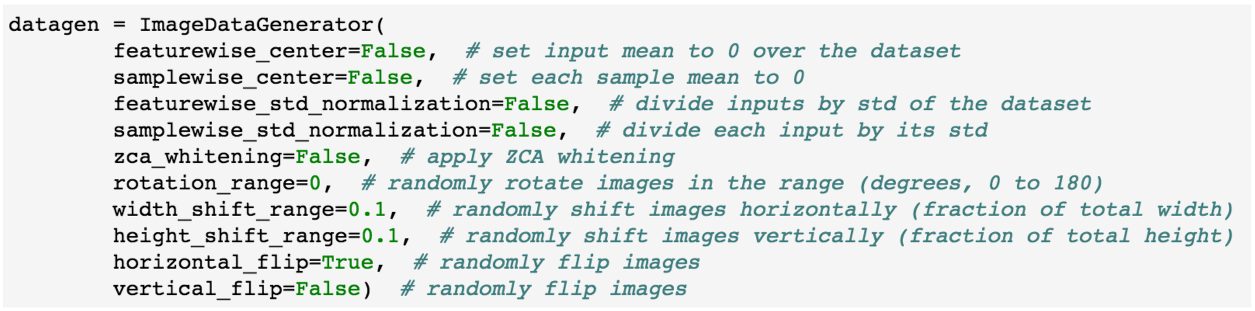


Figure 3. Examples for 10 classes in CIFAR-10



```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

Figure 4. Data Augmentation Code

4 Approaches

4.1 Models

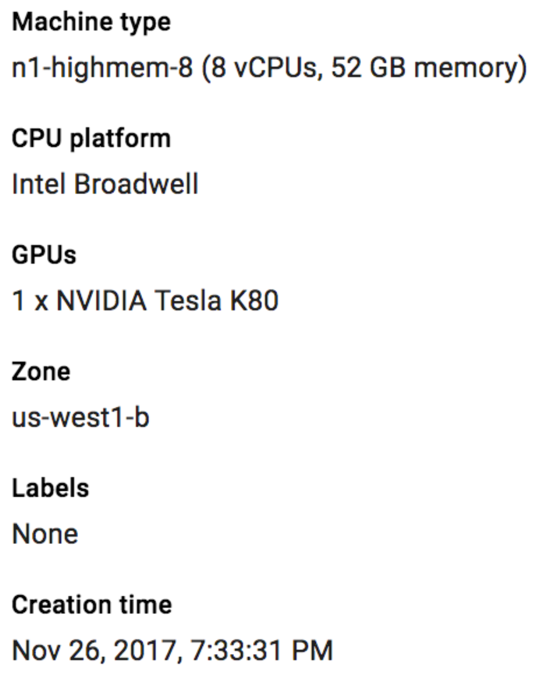
Due to time constraints, I only implemented a simple convolutional neural network. The model was introduced in this tutorial and achieved a test accuracy of 84.07% and 85.97% with and without data augmentation respectively. The detailed Model information is summarized in Table 1.

```
____________________________________________________________________________________
____________
Layer (type)                     Output Shape          Param #     Connected to

====================================================================================
============
convolution2d_7 (Convolution2D)  (None, 48, 32, 32)    1344        convolution2d_input_2
[0][0]
____________________________________________________________________________________
____________
activation_9 (Activation)        (None, 48, 32, 32)    0           convolution2d_7[0][0]

____________________________________________________________________________________
____________
convolution2d_8 (Convolution2D)  (None, 48, 30, 30)    20784       activation_9[0][0]

____________________________________________________________________________________
____________
activation_10 (Activation)       (None, 48, 30, 30)    0           convolution2d_8[0][0]

____________________________________________________________________________________
____________
maxpooling2d_4 (MaxPooling2D)    (None, 48, 15, 15)    0           activation_10[0][0]

____________________________________________________________________________________
____________
dropout_6 (Dropout)              (None, 48, 15, 15)    0           maxpooling2d_4[0][0]

____________________________________________________________________________________
____________
convolution2d_9 (Convolution2D)  (None, 96, 15, 15)    41568       dropout_6[0][0]

____________________________________________________________________________________
____________
activation_11 (Activation)       (None, 96, 15, 15)    0           convolution2d_9[0][0]

____________________________________________________________________________________
____________
convolution2d_10 (Convolution2D) (None, 96, 13, 13)    83040       activation_11[0][0]

____________________________________________________________________________________
____________
activation_12 (Activation)       (None, 96, 13, 13)    0           convolution2d_10[0]
[0]
____________________________________________________________________________________
____________
maxpooling2d_5 (MaxPooling2D)    (None, 96, 6, 6)      0           activation_12[0][0]

____________________________________________________________________________________
____________
dropout_7 (Dropout)              (None, 96, 6, 6)      0           maxpooling2d_5[0][0]

____________________________________________________________________________________
____________
convolution2d_11 (Convolution2D) (None, 192, 6, 6)     166080      dropout_7[0][0]

____________________________________________________________________________________
____________
activation_13 (Activation)       (None, 192, 6, 6)     0           convolution2d_11[0]
[0]
____________________________________________________________________________________
____________
convolution2d_12 (Convolution2D) (None, 192, 4, 4)     331968      activation_13[0][0]

____________________________________________________________________________________
____________
activation_14 (Activation)       (None, 192, 4, 4)     0           convolution2d_12[0]
[0]
____________________________________________________________________________________
____________
maxpooling2d_6 (MaxPooling2D)    (None, 192, 2, 2)     0           activation_14[0][0]

____________________________________________________________________________________
____________
dropout_8 (Dropout)              (None, 192, 2, 2)     0           maxpooling2d_6[0][0]

____________________________________________________________________________________
____________
flatten_2 (Flatten)              (None, 768)           0           dropout_8[0][0]

____________________________________________________________________________________
____________
dense_4 (Dense)                  (None, 512)           393728      flatten_2[0][0]

____________________________________________________________________________________
____________
activation_15 (Activation)       (None, 512)           0           dense_4[0][0]

____________________________________________________________________________________
____________
dropout_9 (Dropout)              (None, 512)           0           activation_15[0][0]

____________________________________________________________________________________
____________
dense_5 (Dense)                  (None, 256)           131328      dropout_9[0][0]

____________________________________________________________________________________
____________
activation_16 (Activation)       (None, 256)           0           dense_5[0][0]

____________________________________________________________________________________
____________
dropout_10 (Dropout)             (None, 256)           0           activation_16[0][0]

____________________________________________________________________________________
____________
dense_6 (Dense)                  (None, 10)            2570        dropout_10[0][0]
====================================================================================
============
Total params: 1,172,410
Trainable params: 1,172,410
Non-trainable params: 0
____________________________________________________________________________________
```

Table 1. model summary table

4.2 Approaches

I implemented the convolutional neural network in Python 3.5.2 using Keras 1.2.0 [11]. Keras is a python library for neural networks and provides a high-level inteface to Theano or Tensorflow libraries. It is worth noting that using Keras 2.0 will lead to lots of errors because they take different arguments for quite a few libraries. In this project, I used tensorflow libraries. For hardware, I used Google Cloud platform (Figure 5). I followed the instructions in http://cs231n.github.io/gce-tutorial-gpus/ to set up the GPU.



Machine type
n1-highmem-8 (8 vCPUs, 52 GB memory)

CPU platform
Intel Broadwell

GPUs
1 x NVIDIA Tesla K80

Zone
us-west1-b

Labels
None

Creation time
Nov 26, 2017, 7:33:31 PM

Figure 5. CPU and GPU information used in the project

5 Experimental results and Discussion

Each experiment was run with the corresponding algorithms and parameters for 100 epochs (In fact, the result would be much clearer with more epochs such as 250. However, I shortened the time due to time constraints). It took about 1.5 hours to complete each set of experiments with around 55 seconds for each epoch. The values for the hyper-parameters were decided ad hoc as they are the default values given in Keras documentation. It would definitely be interesting to use K-fold cross validation to tune higher parameters and improve performance. However, I did not include it due to time limits.

5.1 Influence of data augmentation

I first tested the influence of data augmentation on image classification using Adam optimization algorithm. Figure 6 showed the results of fitting original data. Both graph showed that train accuracy and test accuracy started to diverge around 10 epochs. And it only took Adam algorithm 20 epochs to reach a test accuracy close to 80%. After 20 epochs, the test accuracy was not changing much whereas the train accuracy still increased slighted and finally was over 90%, indicating the model started to be overfitting. Figure 7 showed the results of fitting augmented dataset. The graph does not seem to converge yet. Up to 100 epochs, both train accuracy and test accuracy were still increasing and getting closer. We could see that using same parameters, data augmentation helped solved the overfitting issue in Figure 6. Table 2 showed that both experiments used same amount time of. And model trained with augmented data showed a bit higher accuracy on test data. Therefore, in this case, data augmentation could make model generalize better and be more accurate.

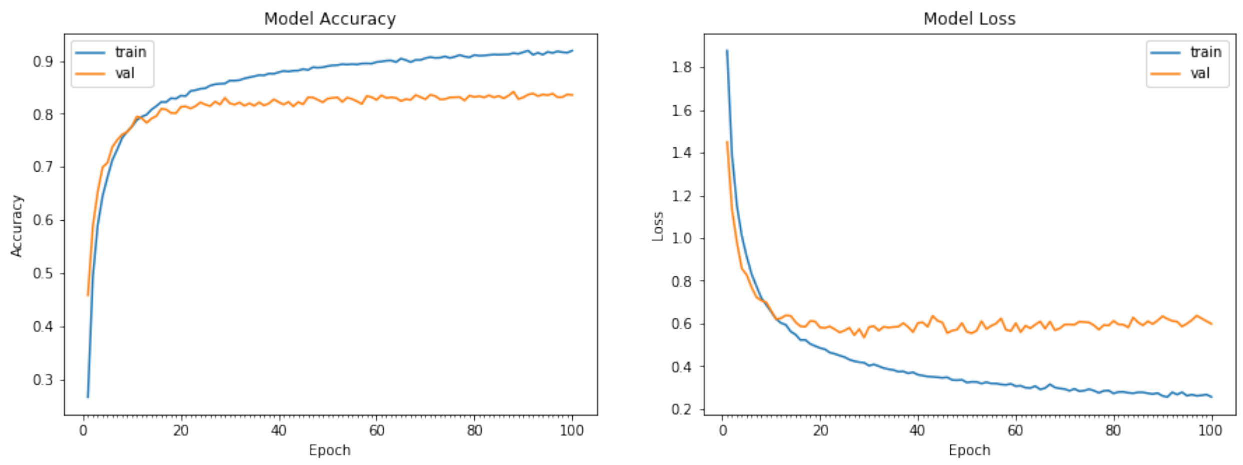


Figure 6. Loss scores and accuracies for training (blue) and validation (red) data for the Cross-entropy loss without data augmentation using Adam optimization algorithm

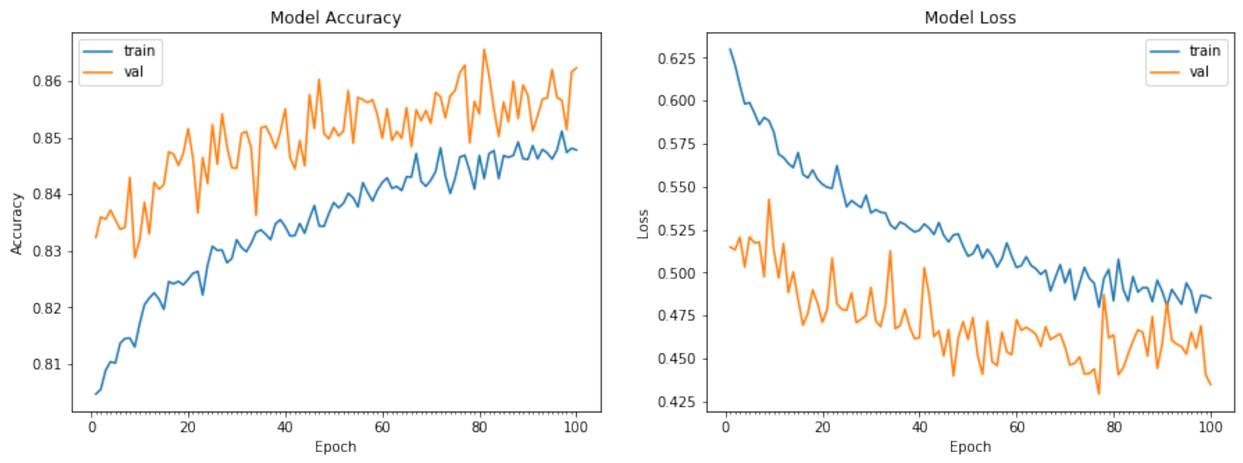


Figure 7. Loss scores and accuracies for training (blue) and validation (red) data for the Cross-entropy loss with data augmentation using Adam optimization algorithm

	Without augmentation	With augmentation
train time(seconds)	5536.95	5574.6
accuracy on test	83.45%	86.23%

Table 2. Comparison between using and not using data augmentation

5.2 Influence of Optimization algorithm

Then I compared the running result between 2 optimization algorithms, i.e., sgd and adam. Based on lots of others' experience, adam was usually superior than sgd in terms of speed and accuracy. However, it was not our case. Table 3 showed that sgd achieved a higher test accurate than adam. Figure 8 showed the detailed results of running sgd algorithm on augmented dataset with cross-entropy loss function. With sgd algorithm, the accuracy started with very small number and dramatically increased as we increased epochs. With 20 epochs, it could achieve an accuracy of 80% for both test and train data and the increase rate was close to plateau after that. It was worth noting that the accuracy in Figure 7 passed 80% only with 1 epoch using adam, although the algorithm did not converge. This indicated that it would be better to use adam if we did not have much time. If we ran algorithm with less than 20 epochs, adam is definitely superior than sgd. It seemed like both models resolved overfitting issue with augmented data. Another interesting point to me was that the validation accuracy was higher than train accuracy for both models. I searched online and found one explain that if we used dropout regularization layer in the network, the validation error may be smaller than train error because usually dropout was activated when training but deactivated when evaluating on the validation set. [12]

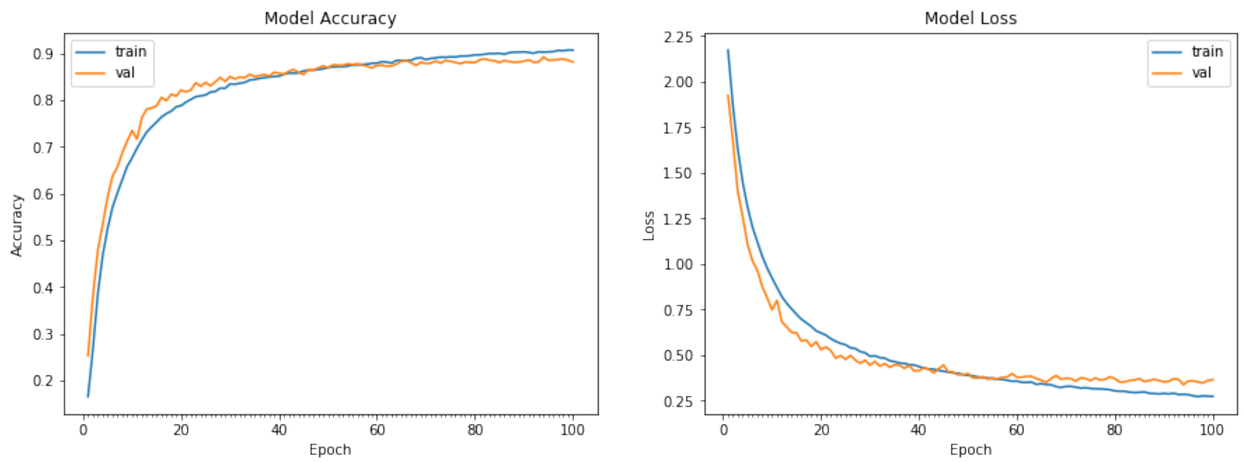


Figure 8. Loss scores and accuracies for training (blue) and validation (red) data for the Cross-entropy loss with data augmentation using SGD optimization algorithm

	adam	sgd
train time(seconds)	5574.6	5503.64
accuracy on test	86.23%	88.21%

Table 3. Comparison between adam and sdg with augmented dataset

5.3 Influence of loss function

At last, I tested three loss functions, categorical cross-entropy, hinge, and hinge squared. The three functions were run with a sgd optimizer (same parameters as in Figure 8) for 100 epochs. Different from what I expected, hinge loss and squared hinge failed to show better performance than cross-entropy loss (Table 4).

	Cross-entropy(sgd)	Hinge(sgd)	Squared-hinge(sgd)
train time(seconds)	5503.64	5501.13	5527.88
accuracy on test	88.21%	34.44%	59.53%

Table 4. Comparison between 3 loss functions

Figure 9 showed the model loss score and accuracies for training and test data using the standard hinge loss function. The model did not converge after 100 epochs. Compared Figure 9 with Figure 8, we could see the model converged much smaller when using hinge loss. Although there was no overfitting to the training data, the final accuracy of hinge loss function was just around 35% for test data, which was much smaller than what we achieved using cross-entropy loss.

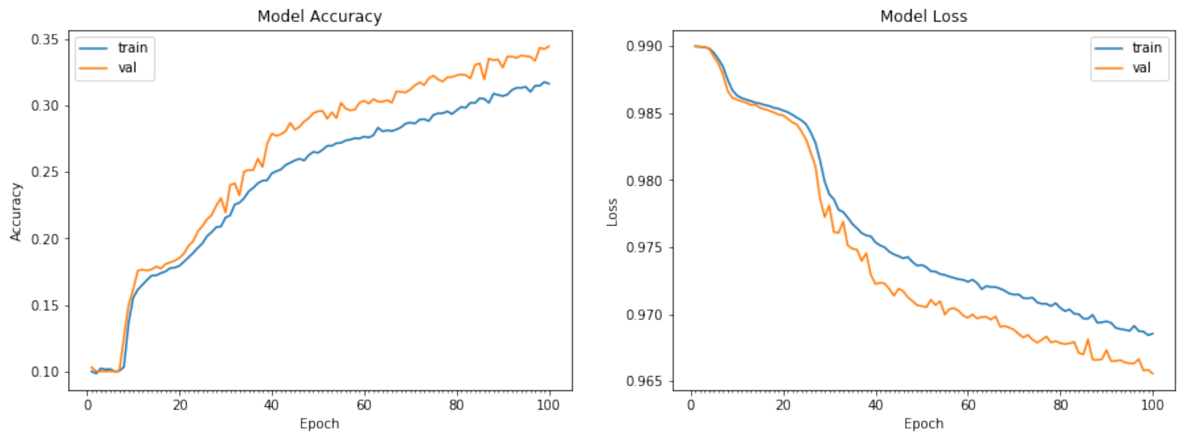


Figure 9. Loss scores and accuracies for training (blue) and validation (red) data for the hinge loss with data augmentation using SGD optimization algorithm

Figure 10 showed the model loss score and accuracies for train and test data using the squared hinge loss function. Similar to what we saw in Figure 9, the model failed to converge after 100 epochs and converged very slowly. There was no overfitting issue with augmented data. However, the final accuracy of squared hinge loss was just, which was also much smaller than that of cross-entropy loss. However, it was clear that the squared hinge loss function outperformed hinge loss function in terms of convergence and final accuracy.

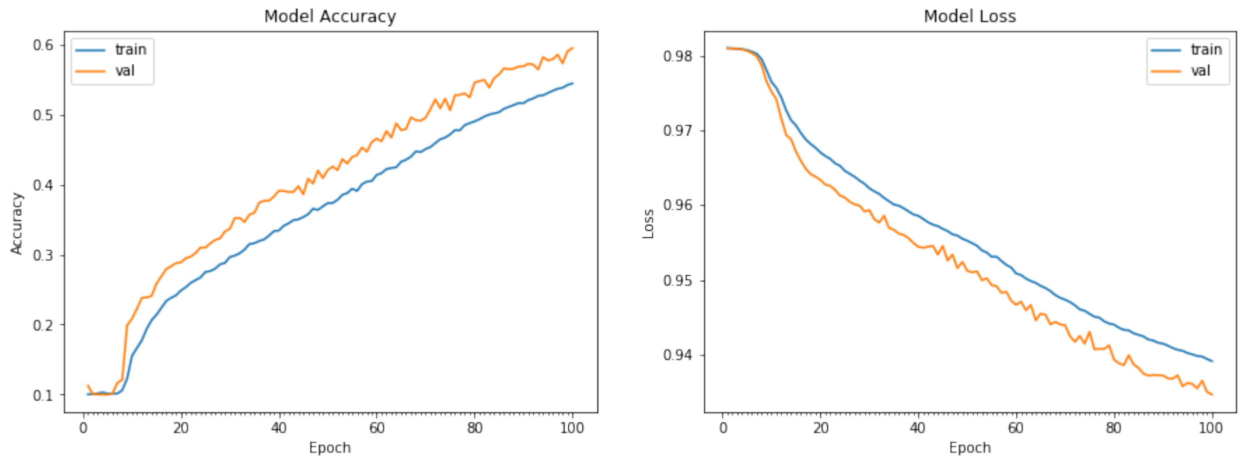


Figure 10. Loss scores and accuracies for training (blue) and validation (red) data for the Squared hinge loss with data augmentation using SGD optimization algorithm

6 Conclusions and Future Work

In this project, I implemented and trained from scratch a CNN model on CIFAR-10 dataset. I studied three aspects of influence, i.e., data augmentation, optimizers and loss functions. As expected, the image augmentation helps to boost the accuracy a bit, but not very significantly. As for optimizers, adam outperformed sgd if we chose a very small number of epochs. However, if we used 100 epochs, the sgd showed better results than adam. Finally, in terms of loss functions, cross-entropy loss function clearly outperformed the two hinge functions not only in convergence rate but also in accuracy. And the squared hinge function achieved much better result than standard hinge function. The conclusions stated here was not perfect since most of experiments failed to converge. I did not have enough time to run larger epochs. In the future, it would be necessary to see what results are with larger number of epochs. Another future study would be interesting to me is to see how these factors influence the deeper neural networks or other famous deep nets like ResNet, VGG, GoogLenet and so on. I was trying to implement this experiments using pytorch libraries. However, it took too much time and I couldn't finish everything if I chose this topic.

Lastly, this is the first time I use GPU and implement convolutional neural networks. I heavily referred a lot of online tutorials and github repositories. But I do learn a lot through the process. I have to say this is not a conference level paper, but some learning summaries. A recently interesting and provoking debate is about whether deep learning is "new electricity" or "alchemy". I do feel like I did tons of "alchemy" these days, since I couldn't explain why some conditions achieved better result whereas others failed. It is also weird to me lots of validation error was smaller than training error. The happy news is there is still a lot to learn.

References:

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, 2009, pp. 248–255.
[2] “https://cs231n.github.io/convolutional-networks/.” .
[3] D. Britz, “Understanding Convolutional Neural Networks for NLP,” WildML, 07-Nov-2015. [Online]. Available: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/. [Accessed: 10-Dec-2017].
[4] Y. Tang, “Deep learning using support vector machines,” CoRR, abs/1306.0239, vol. 2, 2013.
[5] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, “Is L2 a Good Loss Function for Neural Networks for Image Processing?,” ArXiv e-prints, vol. 1511, 2015.
[6] S. Chen and Y. Wang, “Convolutional neural network and convex optimization,” Dept. of Elect. and Comput. Eng., Univ. of California at San Diego, San Diego, CA, USA, Tech. Rep, 2014.
[7] S. Dierauf, “cifar-classifier.” [Online]. Available: https://github.com/sdierauf/cifar-classifier.
[8] “An overview of gradient descent optimization algorithms,” Sebastian Ruder, 19-Jan-2016. [Online]. Available: u=http://ruder.io/optimizing-gradient-descent/. [Accessed: 10-Dec-2017].
[9] “What is Data Augmentation? - Definition from Techopedia,” Techopedia.com. [Online]. Available: https://www.techopedia.com/definition/28033/data-augmentation. [Accessed: 10-Dec-2017].
[10] “Convolutional Neural Networks (CNN) for CIFAR-10 Dataset,” Parneet Kaur. [Online]. Available: http://parneetk.github.io/blog/cnn-cifar10/. [Accessed: 10-Dec-2017].
[11] F. Chollet, keras: Deep Learning library for Python. Runs on TensorFlow, Theano, or CNTK. 2017.
[12] “machine learning - Validation Error less than training error? - Cross Validated.” [Online]. Available: https://stats.stackexchange.com/questions/187335/validation-error-less-than-training-error. [Accessed: 10-Dec-2017].