# MLP Coursework 4

s1631854

March 21, 2017

## 1 Introduction

### 1.1 Dataset introduction

The experiments are based on CIFAR10 and CIFAR100 dataset.The CIFAR10 dataset (Krizhevsky, 2009) consists of ten classes of natural images with 50,000 examples for training and 10,000 for testing. Each example is a 32x32 RGB image taken from the tiny images dataset collected from the web [1]. Each image of this dataset is normalized before feeding into the neural network.CIFAR100 (Krizhevsky, 2009) is just like CIFAR-10, but with 100 categories. CIFAR100 is also scaled to [0, 1].

### 1.2 Architecture introduction

In MLP coursework 3, a basic architecture is used to classify CIFAR10 and CIFAR100. Based on 2 fully connected layers, Elu activation function, Momentum Optimizer, learning rate 0.001, the model can achieve 0.52 valid accuracy on CIFAR10 and 0.24 valid accuracy on CIFAR100. In MLP coursework 4, a more advanced architecture is going to be used for classification tasks. The following experiments are based on convolutional neural network, which has 3 convolutional layers and 2 fully connected layers and 1 output layer. More specially, the SAME padding and max-pooling are used and the patch size is 5x5.

### 1.3 Motivation introduction

Over-fitting refers to a model that models the training data too well.Over-fitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data[2]. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.
Convolutioanl neural network, such a complicated model, has a severe over-fitting problem. Therefore, this coursework will focus on this problem and try different methods to reduce the effect of over-fitting. More concretely, some methods, such as dropout, L2 regularization, early stopping and data augmentation, will be tested.

### 1.4 Research question introduction

(1) What is the difference between a model with dropout and without dropout? What is the optimal keeping probability for dropout? Where to put the dropout layer is the most suitable? What is the effect of dropout on different dataset size?
(2) What is the effect of L2 regularization? Comparing with dropout, what is the difference between these two methods?
(3) What is the effect of data augmentation? What is the difference of different data augmentation methods?
(4) How does the early stopping help to reduce over-fitting? When to stop?

# 2 Dropout

## 2.1 Methods

### 2.1.1 Method introduction

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to over-fitting and dropout is a technique that addresses this issue[3].
It prevents over-fitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, temporarily remove it from the network, along with all its incoming and outgoing connections[3]. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units[4].

### 2.1.2 Hyper-parameter setting

The experiment is based on convolutional neural network, which has 3 convolutioanl layers, 3 max pooling layers and 2 fully connected layers and 1 output layer. For the convolutional layer, the kernel size is 5x5 and stride is 1. For the max pooling layer, the kernel size is 2x2 and stride is 2 too. For the fully connected layer, the hidden units are 1024. The activation function is Relu and the optimizer is Adam with learning rate 0.001.

### 2.1.3 Implementation

The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights.
In tensor flow, tf.nn module has the dropout function: tf.nn.dropout(x, keep_prob),

where x is the input, keep_prob is the probability that each element is kept. The dropout layer can be placed after the output of convolutional layers or max pooling layers or fully connected layers.

## 2.2 Results and Discussion

### 2.2.1 With dropout & Without dropout

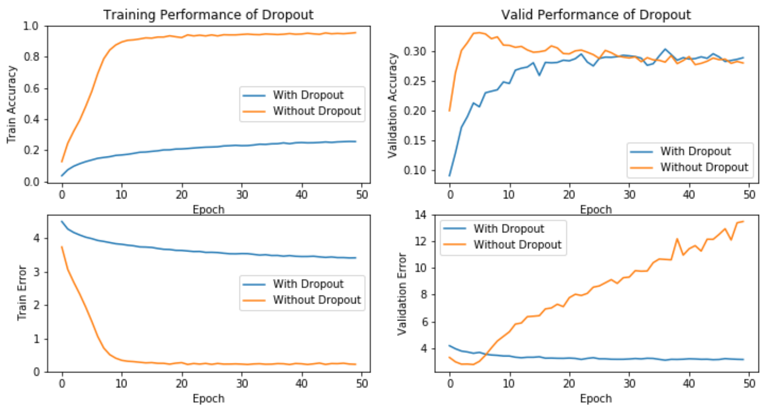

Figure 1: Performances of dropout on CIFAR100

In order to test the effect of dropout, classification experiments were done with networks with and without dropout keeping all other hyper-parameters. Figure 1 shows the train and valid performance. The same architectures trained with and without dropout have drastically different train and valid accuracy as seen as by the separate curves. In the training set, the architecture without dropout can achieve nearly 100% classification accuracy, while in the validation set, the accuracy is only 0.28. There exits a great gap between the training accuracy and valid accuracy. However, the gap between training accuracy and valid accuracy of the architecture with 0.5 dropout becomes very smaller. Also, from the error curve, we can see the training error curve without drop decreases gradually while the valid error curve decreases first and then increases fast with the increase of epoch. However, the trend of training and valid error curve of 0.5 dropout is consistent. From figure 2 we can see dropout also works on CIFAR10. Each training case effectively tries to train a different random architecture. Therefore, the gradients that are being computed are not gradients of the final architecture that will be used at test time. It is likely that this stochasticity prevents over-fitting. In conclusion, dropout gives a huge improvement across the experiments, which is also consistent with my hypothesis.

### 2.2.2 Different probability of dropout

Dropout has a tunable hyper-parameter p (the probability of retaining a unit in the network). In this experiment, the effect of varying this hyper-parameter will also be explored. Because this CNN architecture is complex , retaining 80%

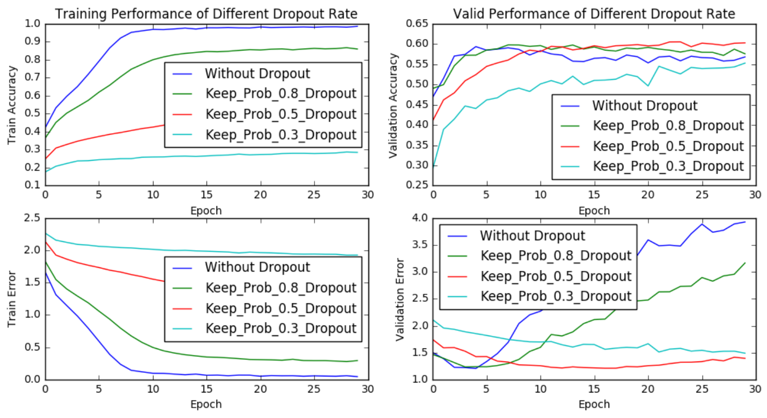

Figure 2: Performances of different probability dropout on CIFAR10

units in the network is still too much so that the valid error curve in figure 2 shows the same trend as the one without dropout. Besides, since retaining 30% units is too few, it makes the architecture loses too much useful information so that the valid accuracy is not as high as others. Figure 2 shows that keeping 50 % units is the most suitable because on the one hand it allows the model to gain enough useful information, on the other hand, it enables the model to be as simple as it can.

### 2.2.3 Different dropout methods

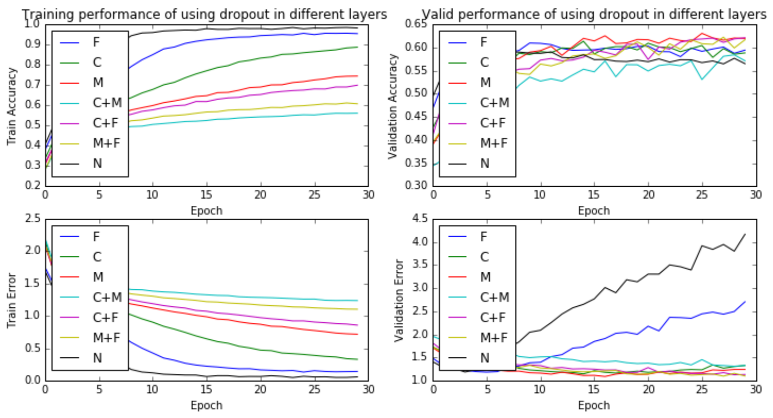

Figure 3: Performances of different locations to place dropout layer on CIFAR10(N means no dropout; F means fully connected dropout; C means convolutional dropout; M means max pooling dropout)

Various CNN models were also trained by separately and simultaneously using dropout in different layers on CIFAR10. Table 2 records the performance

Table 1: The performance of different dropout methods in CIFAR10

| Dropout method | train_err | valid_err | train_acc | valid_acc |
|---|---|---|---|---|
| No dropout | 0.06 | 4.16 | 0.98 | 0.57 |
| Fully connected dropout | 0.14 | 2.70 | 0.95 | 0.59 |
| Convolutional dropout | 0.33 | 1.33 | 0.89 | 0.59 |
| Max pooling dropout | 0.72 | 1.24 | 0.74 | 0.62 |
| Convolutional and Max pooling dropout | 1.24 | 1.33 | 0.56 | 0.57 |
| Convolutional and Fully connected dropout | 0.86 | 1.13 | 0.70 | 0.62 |
| Max pooling and Fully connected dropout | 1.10 | 1.10 | 0.61 | 0.62 |

of various dropout methods. Without dropout, the valid error is 4.16 and valid accuracy is 0.57. Figure 3 and Table 2 show that separately using convolutional, max-pooling and fully-connected dropout reduces the valid error and increases the valid accuracy. Convolutional dropout still underperforms max-pooling dropout. Again, simultaneously using convolutional and max-pooling dropout seems to easily result in too strong regularization, and thus decreases the valid performance. Simultaneously using convolutional and fully connected dropout or max-pooling and fully connected dropout achieves the best performance. This conclusion is consistent with this paper[4].

### 2.2.4 Using dropout in different dataset size

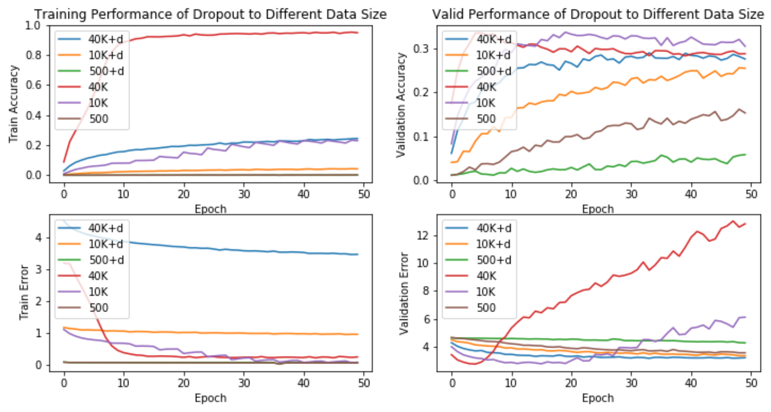

Figure 4: Performances of using dropout in different dataset size on CIFAR100

One test of a good regularizer is that it should make it possible to get good generalization error from models with a large number of parameters trained on small data sets. This section explores the effect of changing the data set size when dropout is used. Huge neural networks trained in the standard way must over-fit massively on small data sets. To see if dropout can help, classification experiments were carried out on CIFAR100 and vary the amount of data given to the network.

Table 2: The performance of using dropout in different dataset size in CIFAR100

| Dataset size | train_err | valid_err | train_acc | valid_acc |
|---|---|---|---|---|
| 500(without dropout) | 0.04 | 3.56 | 0.00 | 0.15 |
| 500(with dropout) | 0.05 | 4.27 | 0.00 | 0.06 |
| 10K(without dropout) | 0.07 | 6.11 | 0.23 | 0.31 |
| 10K(with dropout) | 0.95 | 3.35 | 0.04 | 0.26 |
| 40K(without dropout) | 0.23 | 12.82 | 0.95 | 0.29 |
| 40K(with dropout) | 3.47 | 3.21 | 0.24 | 0.28 |

The results of these experiments are shown in Figure 4. The network was given data sets of size 500,10K,40K chosen randomly from the CIFAR100 training set. The same network architecture was used for all data sets. Dropout with p = 0.5 was performed at all the hidden layers. It can be observed that for extremely small data sets (500) dropout does not give any improvements. The model has enough parameters that it can over-fit on the training data, even with all the noise coming from dropout. As the size of the data set is increased, the gain from doing dropout increases up. But according to this paper [3], for any given architecture and dropout rate, there is a "sweet spot" corresponding to some amount of data that is large enough to not be memorized in spite of the noise but not so large that over-fitting is not a problem anyways. Therefore, if the dataset size is large enough, the effect of dropout will decrease.

## 2.3 Conclusion

From the experiments above, we can know dropout is a technique for improving neural networks effectively by reducing over-fitting. Also, using different dropout methods and different dropout probability have a totally different performance. The conclusion is consistent with this paper[4]. Certainly, using dropout on different dataset size has a different effect as well.
Standard back-propagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptations by making the presence of any particular hidden unit unreliable.
One of the shortcoming of dropout is that it increases training time. A dropout network typically takes longer time to train than a standard neural network of the same architecture. This creates a trade-off between over-fitting and training time. With more training time, one can use high dropout and suffer less over-fitting.

# 3 L2 Regularization

## 3.1 Methods

### 3.1.1 Method introduction

Another possible approach to reduce over-fitting is to reduce the size of our network. However, large networks have the potential to be more powerful than small networks, and so this is an option we'd only adopt reluctantly. Fortunately, there are other techniques which can reduce over-fitting, even when we have a fixed network and fixed training data. These are known as regularization techniques. One of the most commonly used regularization techniques is a technique sometimes known as weight decay or L2 regularization[5].
The idea of L2 regularization is to add an extra term to the cost function, a term called the regularization term. Here's the regularized cross-entropy:

$$C = C_0 + \frac{\lambda}{2n} \sum_w W^2 \tag{1}$$

The first term is just the usual expression for the cross-entropy. But we've added a second term, namely the sum of the squares of all the weights in the network. This is scaled by a factor $\lambda/2n$ , where $\lambda > 0$ is known as the regularization parameter, and n is, as usual, the size of our training set[5]. Importantly, the regularization term doesn't include the biases.

### 3.1.2 Hyper-parameter setting

The overall hyper-parameter setting is the same as dropout experiment.

### 3.1.3 Implementation

Intuitively, the effect of regularization is to make the network prefers to learn small weights. Large weights will only be allowed if they considerably improve the first part of the cost function.
So according to formula 1, the regularization should be added with cross-entropy. We need to rewrite the loss variable. In tensorflow, calling tf.nn.l2_loss() function can help us do the regularization. What we need to do is just add each weight into the function, such as the convolutional layer weights, the fully connected layer weights. It's also worth noting that the regularization term doesn't include the biases, so for example, the convolutional layer biases, don't need to be added. After summing up all the l2_loss, this regularization should be added to the original cross-entropy to be the new loss.

## 3.2 Results and Discussion

### 3.2.1 With L2 regularization & without L2 regularization

From Figure 5, we can observe that training with L2 regularization reduces over-fitting a lot. Without L2 training regularization, the training accuracy can achieve nearly 100%, while in the validation set, the accuracy is only 0.28. The performance of model with L2 regularization shows a little bit under-fit, but it is because the parameter $\gamma$ is too large, it penalizes the error too much.

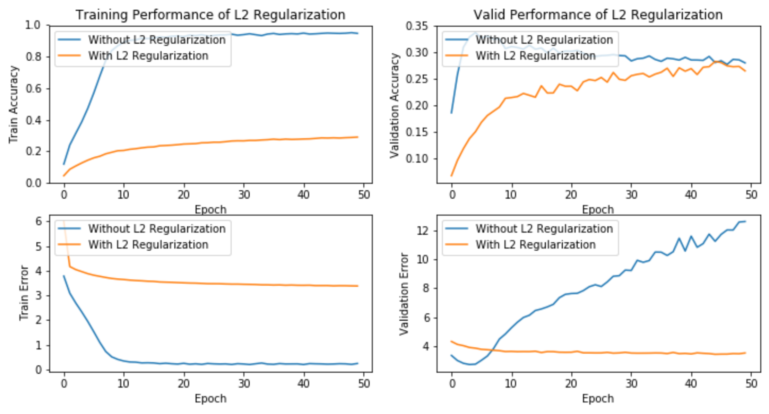

Figure 5: Performances of L2 regularization on CIFAR10

Even though, we can still observe that the gap between training accuracy and valid accuracy in the model with L2 regularization becomes smaller. Also, from the error curve we can notice that these two training errors decrease gradually with the increase of epoch, but the valid error without L2 regularization decreases first and increase rapidly later. However, the model with L2 regularization prevent this situation happening well. In conclusion, instead of requiring large coefficients to correctly represent sudden changes, incoherence, or other high-dimensionality phenomena shown in individual data points in the training data, L2 regularization limits the expressiveness of a model to "smooth" or low dimensional results, which might fit real world data better under most metrics.

### 3.2.2 With L2 regularization & with dropout

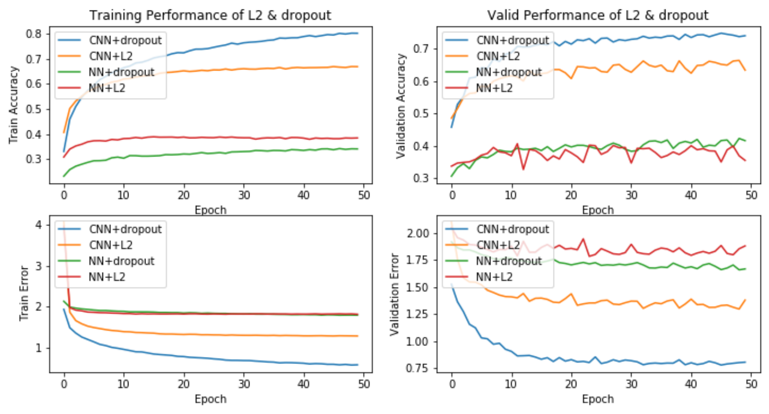

Figure 6: Performances of L2 regularization and dropout on CIFAR10

Table 3: The performance of L2 regularization and dropout on CIFAR10

| method | train_err | valid_err | train_acc | valid_acc |
|---|---|---|---|---|
| NN+dropout | 1.79 | 1.67 | 0.34 | 0.42 |
| NN+L2 | 1.81 | 1.88 | 0.38 | 0.35 |
| CNN+dropout | 0.58 | 0.81 | 0.80 | 0.74 |
| CNN+L2 | 1.28 | 1.38 | 0.67 | 0.63 |

Table 3 records different performances of dropout and L2 regularization with simple and complicated architecture. NN represents a simple architecture, which is the same architecture of coursework 3. CNN represents a complicated architecture, which consists of 3 convolutional layers,3 max-pooling layers and 2 fully connected layers and 1 output layer. From the Figure 6 we can observe that in simple architectures, the performance of using L2 regularization is slightly better than using dropout, but there is not a great difference. However, when the network grows larger and model has a large number of parameters, using dropout has a significant improvement than using L2 regularization, which can be seen from valid accuracy and error curve.

### 3.3 Conclusion

From the previous 2 experiments, it can be concluded that regularization helps to solve over-fitting problem in machine learning. Simple model will be a very poor generalization of data. At the same time, complex model may not perform well in test data due to over-fitting. We need to choose the right model in between simple and complex model. Regularization helps to choose preferred model complexity, so that model is better at predicting.
Experimental results also demonstrate that dropout training in a large network not only provides better performance improvement but it is more robust than L2 regularization. In contrast, the L2 regularization yields higher predictive accuracy than dropout in a small network since averaging learning model will enhance the overall performance when the number of sub-model is large and each of them must different from each other.

## 4 Data augmentation

### 4.1 Methods

#### 4.1.1 Method introduction

One way to avoid over-fitting is to collect a lot of data. But the cost of getting plenty of data is very high. Therefore, we can use data augmentation to make some more data.
Augmentation refers to the process of generating new training data from a smaller data set such that the new data set represents the real world data. In this experiment, five different data augmentation methods will be used.

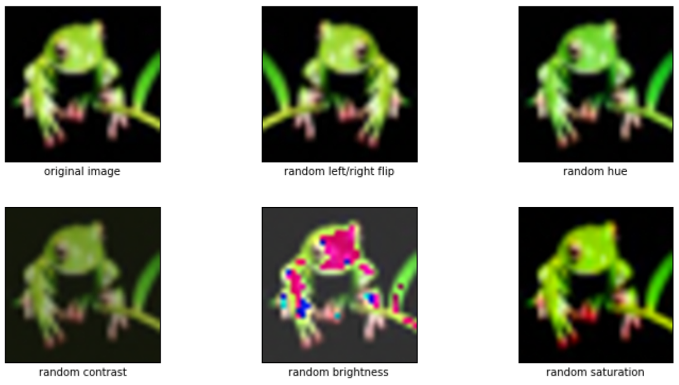

Figure 7: Different data augmentation methods

1. Brightness augmentation: changing brightness to simulate light conditions. We will generate images with different random brightness by converting images.
2. Left to right shift: shifting the images horizontally from left to right simulate the effect of object being at different positions in the scene.
3. Hue augmentation: in the real world, the same hue will look different in different light. So this method can reduce the hue effect for recognition.
4. Contrast augmentation: changing the difference between its darkest and lightest areas.
5. Saturation augmentation: changing the saturation to make colors more vivid or more muted.

#### 4.1.2 Hyper-parameter setting

The overall hyper-parameter setting is the same as dropout experiment except for the data augmentation part. Hyper-parameter of random_hue,max_delta, is set to 0.05. The hyper-parameters of random_contrast, lower is set to 0.3 and upper is set to 1.0. The hyper-parameters of random_brightness, max_delta is set to 0.2. The hyper-parameters of random_saturation, lower is set to 0.0 and upper is set to 2.0.

#### 4.1.3 Implementation

We should make full use of tensorflow property, graph. If you do the data augmentation after you get images from the batch each time, it is time-consuming. The wise way is to regard the data augmentation as the "first layer" in the architecture. This layer is different from such as, convolutional layer, but we can store it in the graph so that it can speed up processing. We can use tensorflow tf.image module to process the image, but it can only process one image at a

time.
So the first step is to unpack images from the batch:

```
with tf.name_scope('pre_process'):
    images = tf.map_fn(lambda image: pre_process_image(image), inputs)
```

The second step is to define pre_process_image() function. Because a total dimension of per input image is 3x32x32=3072, we need to reshape it into [3,32,32] and then change the axis of the image to be the tensorflow format [32,32,3]. After doing this, we can feed the image into tensorflow image process function:

```
def pre_process_image(image):
    image = tf.reshape(image, [3,32,32])
    image = tf.transpose(image, [1, 2, 0])
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_hue(image,max_delta=0.05)
    image = tf.image.random_contrast(image,lower=0.3, upper=1.0)
    image = tf.image.random_brightness(image, max_delta=0.2)
    image = tf.image.random_saturation(image, lower=0.0, upper=2.0)

    # Limit the image pixels between [0, 1] in case of overflow.
    image = tf.minimum(image, 1.0)
    image = tf.maximum(image, 0.0)
    return image
```

## 4.2 Results and Discussion

### 4.2.1 With data augmentation & without data augmentation

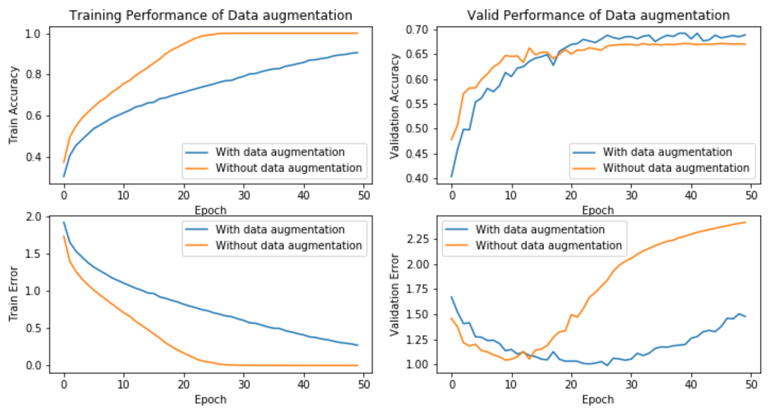

Figure 8: Performances of data augmentation on CIFAR10

From figure 8 and figure 9, we can observe that data augmentation has a significant influence on reducing over-fitting. Figure 8 shows the valid error curve without data augmentation goes down first and then increase, but the valid error curve with data augmentation decreases gradually and only has a

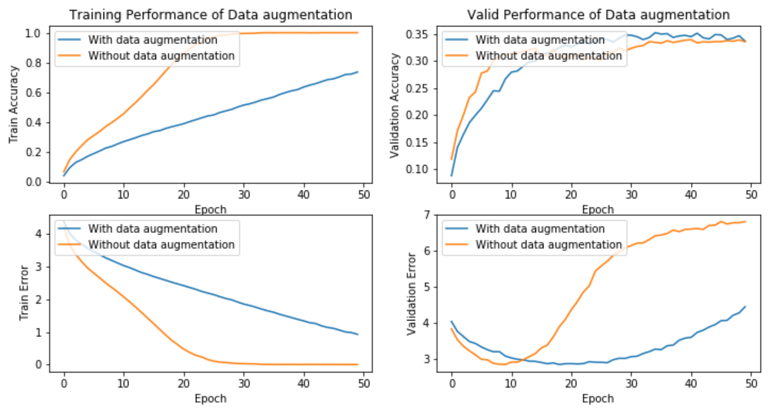

Figure 9: Performances of data augmentation on CIFAR100

slight fluctuation. But one thing worth noting is that the effect of the data augmentation in reducing over-fitting on CIFAR100 is better than CIFAR10. Because CIFAR100 has limited training data per class than CIFAR10, data augmentation has a significant effect on small training set.

### 4.2.2 Comparison of different data augmentation methods

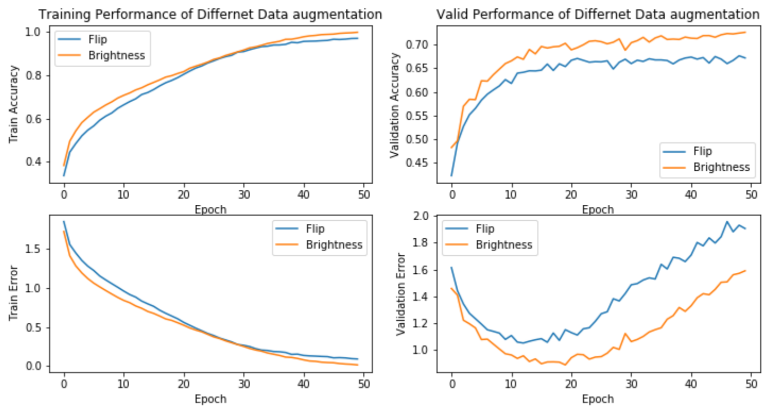

Figure 10: Performances of different data augmentation methods on CIFAR10

Figure 10 shows the performance of different data augmentation methods. We can notice that different data augmentation methods will result in different results. When the classifier is trained on the images on the left (or right), it will most likely wont be able to correctly identify the images on the right (or left) during testing, even though the images on the left and right are the same categories. In this case, mirroring the data in different orientations may eventually help in identifying a similar object in a different orientation. Similarly, adding random brightness to training images may help in identifying a similar object in a different light condition. But in this case, adding random brightness seems to work better than random flipping images from left to right.

## 4.3 Conclusion

Plentiful high-quality data is the key to great machine learning models. But good data does not grow on trees, and that scarcity can impede the development of a model. One way to get around a lack of data is to augment training dataset. The experiments above shows that data augmentation does great help in reducing the effect of over-fitting. But it is important to note that augmentation is only useful when semantically correct. For example, there is no reason to augment images of pedestrians crossing the street to be feet up and head down. On real data such a seen is incredibly unlikely, therefore such augmentation of the training data could hurt the results.

# 5 Early stopping

## 5.1 Methods

### 5.1.1 Method introduction

Early stopping is an approach to training complex machine learning models to avoid over-fitting.
It works by monitoring the performance of the model that is being trained on a separate test dataset and stopping the training procedure once the performance on the test dataset has not improved after a fixed number of training iterations. It avoids over-fitting by attempting to automatically select the inflection point where performance on the test dataset starts to decrease while performance on the training dataset continues to improve as the model starts to over-fit[6].
The performance measure may be the loss function that is being optimized to train the model (such as logarithmic loss), or an external metric of interest to the problem in general (such as classification accuracy).
There are a number of plausible stopping criteria. Here a simple stopping criteria is used: stop as so on as the valid accuracy doesn't exceed a certain threshold for certain epochs.

### 5.1.2 Hyper-parameter setting

The overall hyper-parameter setting is the same as dropout experiment.

### 5.1.3 Implementation

The way to implement early stopping is quite easy. There should be a variable to record the max valid accuracy. Once 10(hyper-parameter) consecutive valid accuracies are less than max valid accuracy, we regard it as over-fitting, so we can break the loop.

## 5.2 Results and Discussion

Figure 11 is the same with my assumption. From the valid accuracy curve, we can see that it stops training after achieving 10 consecutive max accuracies. Also, from the valid error curve we can see when the model starts to over-fit, the model stops training. Comparing with the model without early stopping, the model with early stopping saves nearly half of the training time because for the model without early stopping, the rest of the training time is meaningless.

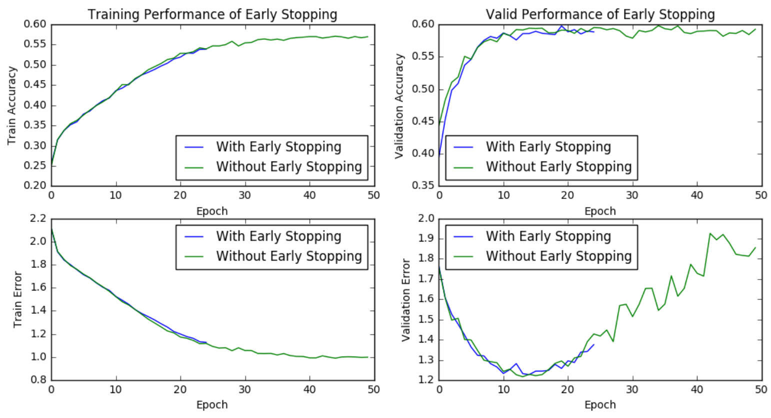

Figure 11: Performances of early stopping on CIFAR10

## 5.3 Conclusion

From the experiment above, early stopping rules can provide guidance as to how many iterations can be run before the learner begins to over-fit. However, for most of neural network trainings, the validation set error does not evolve as smoothly as shown in Figure 11. The validation error can still go further down after it has begun to increase. So choosing a suitable stopping criterion becomes a very important question. Stopping criterion predominantly involves a trade-off between training time and generalization error. Some other stopping criteria may typically find better trade-offs.

# 6 Conclusion

In machine learning, over-fitting occurs when a learning model customizes itself too much to describe the relationship between training data and the labels. Over-fitting tends to make the model very complex by having too many parameters. By doing this, it loses its generalization power, which leads to poor performance on new data.
The experiments show that there are some ways to solve this issue. Firstly, dropout is a technique for improving neural networks by reducing over-fitting. Facing with different dataset size, choosing different dropout probability and using different dropout methods will result in different performance. Secondly,

L2 regularization reduces the effect of over-fitting by penalizing large weights. In a small scale neural network, L2 regularization works more efficiently than dropout. But in a deep neural network, dropout reduces over-fitting more significantly. Besides, another way to avoid over-fitting is to use data augmentation to create more data. The main reason over-fitting happens is the small dataset. The algorithm will have greater control over this small dataset and it will make sure it satisfies all the data points exactly. But if you have a large number of data points, then the algorithm is forced to generalize and come up with a good model that suits most of the points. Finally, early stopping is also a method for avoiding over-fitting and requires a method to assess the relationship between the generalization accuracy(or error) of the learned model and the training accuracy(or error).

## References

[1] Krizhevsky A. Convolutional Deep Belief Networks on CIFAR-10[J]. 2012.

[2] Jason Brownlee, Machine Learning Algorithms, March 21, 2016.

[3] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. Journal of Machine Learning Research, 2014, 15(1):1929-1958.

[4] Wu H, Gu X. Towards dropout training for convolutional neural networks[J]. Neural Networks the Official Journal of the International Neural Network Society, 2015, 71(C):1-10.

[5] Michael Nielsen . (2016). Neural networks and deep learning. Pro PHP MVC. Berkeley, CA: Apress, pp.81-83.

[6] Jason Brownlee.(2016).Avoid Overfitting By Early Stopping With XGBoost In Python.Retrieved from http://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python/

[7] Ekachai Phaisangittisagul. An Analysis of the Regularization between L2 and Dropout in Single Hidden Layer Neural Network[J]. International Conference on Intelligent Systems, Modelling and Simulation, 2016, 15(7).