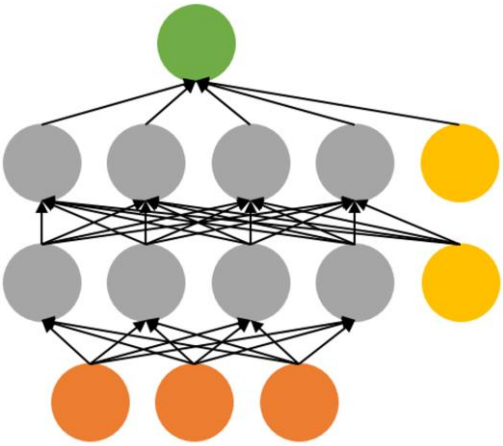

Phase I Report: Image Classification Study

Independent Study

Meng Chen
CSCI-8991

Objective

The objective of the phase I independent study project is study the theory, implementation and the practical usage of the deep feedforward neural network. The theory focuses on the mathematical modeling and backpropagation mechanism; the implementation focuses on the programming aspect of building a feedforward neural network using specialized tools; the practical usage part covers all the useful tips and skills on building a general neural network.

Problem Definition

Phase I study includes a well-defined project, which is used to validate learning. The task of the project is to classify CIFAR10 dataset, a well-known dataset used for many computer vision classification research. The dataset includes 50,000 images, with 5,000 per class, for training the network and 10,000 images for validating the network accuracy.

The goal is to achieve good classification accuracy for the 10 classes included in the dataset.

Theory and Methodology

The basic concept of artificial neural network is inspired by the biological neural network, and it is a simplified representation of how biological neural network works. In an artificial neural network, a neuron is the most basic unit and a number of neurons are grouped together to form a layer. There are three types of layers in a neural network: input layer, hidden layer and output layer. Between layers are the connections. In feedforward network, the connections are one-way connection, where information can only travel forward to the next layer. In a fully connected network, every neuron in a layer is connected to all the neurons in the previous layer.

Also, all the neuron has an attribute called the activation, which can be interpreted as the level of excitement in biological neural network. The activation of a neuron is determined by a function of the activations of all the neurons connected to it. The input layer activations are the raw input and output layer activations are the interpreted results.

What makes the output variable for a given model are the connection weights. Every neuron connection has a corresponding weight. A large connection weights means the receiver neurons are more sensitive to the activation level of the emitting neuron, and vice versa. The number of hidden layers are also important. A network that has more than one hidden layer is called deep network, and, usually, the deeper the network, the more capable it will be, but it will also be much harder to train. Because feedforward networks are trained using error back-propagation method, and if a network is too deep, the earlier layers' gradient (the connection

weights' influence having on the overall output layer error) will likely to suffer from vanishing gradient [1] or exploding gradient issue.

Following is an example of a fully connected deep feed-forward neural network with input size of 3 and output size of 1:

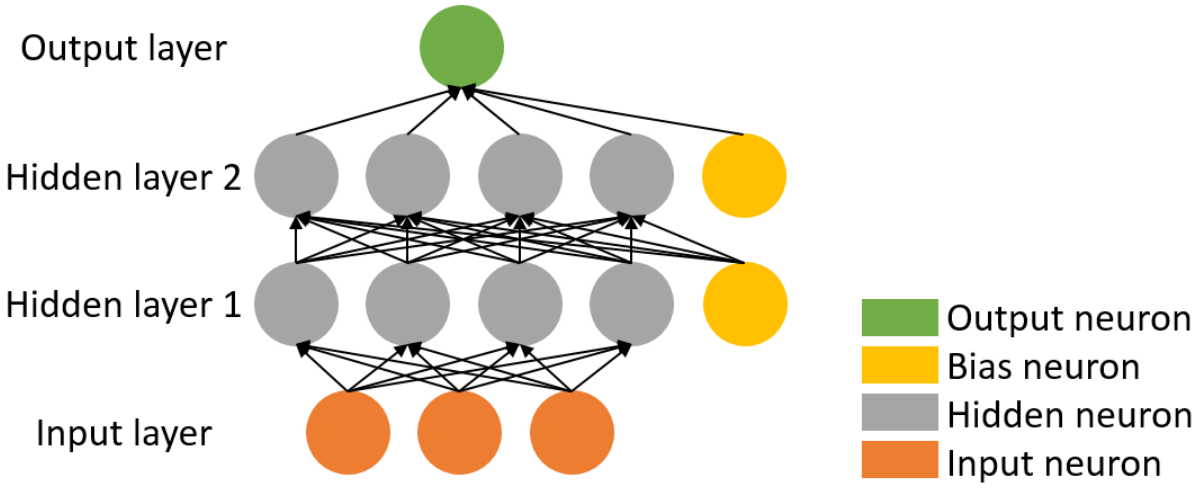

Figure 1 Example of a simple feed-forward deep neural network

Backpropagation

To further explain how back-propagation optimization works, I will first explain how the method works piece-wise, then use the above diagram to create a matrix representation of the same model.

The error is computed is mean squared error (MSE), which is traditionally used. A better error measure is using log-likelihood. For the sake of simplification, I will use MSE for this example. Therefore, the error is computed as following:

$$E = \frac{1}{K}\sum_{k=1}^{K}(o_k - t_k)^2$$

, where K is the number of neurons in the output layer, o_k is the output neuron activation and t_k is the target activation. Since E is the measure of overall network error, if we want to know the effect of individual neuron having on the overall error, error gradient with respect to the output activation of a neuron, we need to take derivative with respect to the individual neuron error:

$$\delta_k^o = \frac{(o_k - t_k)^2}{d(o_k - t_k)} = 2(o_k - t_k)$$

To step back further, we can compute the gradient of any weight for connection to the output layer using the chain rule:

$$\frac{\partial E}{\partial w_{ki}^{o}} = a_i^M \delta_k^o$$

, where a_i^M is the activation of the neuron i on hidden layer M. Here, we assumed that our output is linear, instead using the SoftMax output layer, for simplification purpose.

To trace back one more step, we can determine the output error with respect to the activation of the neuron i on layer M. Because in a fully connected neural network, this neuron is connected to all the neurons on the next layer, we must sum the effects to compute the gradient with respect to its activation:

$$\frac{\partial E}{\partial a_i^M} = \sum_{k=1}^{K} w_{ki}^o \delta_k^o$$

At this point, we have completed a back-propagation cycle, coming from error gradient with respect to the activations on the last layer, the output layer, to the activations of neurons on the second last layer, the last hidden layer. To compute gradients for all weights in the network, we just have to propagate layer by layer using the same technique.

In the actual computation, parameters and gradients will be stored in arrays. To use previous figure as an example, we can create following matrices, where v is for visible layer activation with subscript as the index; w is for connection weight with subscript for emitting neuron and superscript for receiving neuron; h is for hidden neuron, activation with subscript for index and superscript for layer index; when h is used for superscript, $h1{,}2$ stands for neuron on the first hidden layer index 2; b stands for bias and o stands for output:

$$input = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{bmatrix}$$

$$w_v^{h1} = \begin{bmatrix} w_{v1}^{h1,1} & w_{v2}^{h1,1} & w_{v3}^{h1,1} & w_b^{h1,1} \\ w_{v1}^{h1,2} & w_{v2}^{h1,2} & w_{v3}^{h1,2} & w_b^{h1,2} \\ w_{v1}^{h1,3} & w_{v2}^{h1,3} & w_{v3}^{h1,3} & w_b^{h1,3} \\ w_{v1}^{h1,4} & w_{v2}^{h1,4} & w_{v3}^{h1,4} & w_b^{h1,4} \end{bmatrix} \qquad h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \\ h_3^1 \\ h_4^1 \\ 1 \end{bmatrix}$$

$$w_{h1}^{h2} = \begin{bmatrix} w_{h1,1}^{h2,1} & w_{h1,2}^{h2,1} & w_{h1,3}^{h2,1} & w_{h1,4}^{h2,1} & w_b^{h2,1} \\ w_{h1,1}^{h2,2} & w_{h1,2}^{h2,2} & w_{h1,3}^{h2,2} & w_{h1,4}^{h2,2} & w_b^{h2,2} \\ w_{h1,1}^{h2,3} & w_{h1,2}^{h2,3} & w_{h1,3}^{h2,3} & w_{h1,4}^{h2,3} & w_b^{h2,3} \\ w_{h1,1}^{h2,4} & w_{h1,3}^{h2,4} & w_{h1,3}^{h2,4} & w_{h1,4}^{h2,4} & w_b^{h2,4} \end{bmatrix} \qquad h^2 = \begin{bmatrix} h_1^2 \\ h_2^2 \\ h_3^2 \\ h_4^2 \\ 1 \end{bmatrix}$$

$$w_{h2}^{o} = [w_{h2,1}^{o} \quad w_{h2,2}^{o} \quad w_{h2,3}^{o} \quad w_{h2,4}^{o} \quad w_{b}^{o}] \qquad output = [o]$$

From the matrix above, we can see that the number of parameters, weights, are 41, and it can increase very quickly as the size of the network increases. Now, we can also compute the error of the network using MSE measure:

$$E = (o - t)^2$$

$$\delta^{o} = 2(o - t)$$

, where t is the target output activation. Then, the gradient with respect to last layer weights, w_{h2}^{o}, can be determined by:

$$\frac{\partial E}{\partial w_{h2}^{o}} = h^2 \delta^{o} = \begin{bmatrix} 2h_1^2(o-t) \\ 2h_2^2(o-t) \\ 2h_3^2(o-t) \\ 2h_4^2(o-t) \\ 2(o-t) \end{bmatrix}$$

Then, we can also get:

$$\delta_{h2}^{o} = \frac{\partial E}{\partial h^2} = [w_{h2,1}^{o}\delta^{o} \quad w_{h2,2}^{o}\delta^{o} \quad w_{h2,3}^{o}\delta^{o} \quad w_{h2,4}^{o}\delta^{o} \quad w_{b}^{o}\delta^{o}]$$

At this point, we have again back-propagated one complete layer. To propagate further back, we get:

$$\frac{\partial E}{\partial w_{h1}^{h2}} = \begin{bmatrix} h_1^1 w_{h2,1}^{o}\delta^{o} & h_1^1 w_{h2,2}^{o}\delta^{o} & h_1^1 w_{h2,3}^{o}\delta^{o} & h_1^1 w_{h2,4}^{o}\delta^{o} \\ h_2^1 w_{h2,1}^{o}\delta^{o} & h_2^1 w_{h2,2}^{o}\delta^{o} & h_2^1 w_{h2,3}^{o}\delta^{o} & h_2^1 w_{h2,4}^{o}\delta^{o} \\ h_3^1 w_{h2,1}^{o}\delta^{o} & h_3^1 w_{h2,2}^{o}\delta^{o} & h_3^1 w_{h2,3}^{o}\delta^{o} & h_3^1 w_{h2,4}^{o}\delta^{o} \\ h_4^1 w_{h2,1}^{o}\delta^{o} & h_4^1 w_{h2,2}^{o}\delta^{o} & h_4^1 w_{h2,3}^{o}\delta^{o} & h_4^1 w_{h2,4}^{o}\delta^{o} \\ w_{h2,1}^{o}\delta^{o} & w_{h2,2}^{o}\delta^{o} & w_{h2,3}^{o}\delta^{o} & w_{h2,4}^{o}\delta^{o} \end{bmatrix}$$

The gradients with respect to the first hidden layer activations are:

$$\delta_{h1}^{o} = \frac{\partial E}{\partial h^1} = \left[\sum_{k=1}^{4} w_{h1,1}^{h2,k}\delta_{h2,k}^{0} \quad \sum_{k=1}^{4} w_{h1,2}^{h2,k}\delta_{h2,k}^{0} \quad \sum_{k=1}^{4} w_{h1,3}^{h2,k}\delta_{h2,k}^{0} \quad \sum_{k=1}^{4} w_{h1,4}^{h2,k}\delta_{h2,k}^{0} \quad \sum_{k=1}^{4} w_{b}^{h2,k}\delta_{h2,k}^{0}\right]$$

So, the error gradient with respect to the weights connecting the input layer and the first hidden layer can be expressed as following:

$$\frac{\partial E}{\partial w_{v}^{h1}} = \begin{bmatrix} v_1\delta_{h1,1}^{o} & v_1\delta_{h1,2}^{o} & v_1\delta_{h1,3}^{o} & v_1\delta_{h1,4}^{o} \\ v_2\delta_{h1,1}^{o} & v_2\delta_{h1,2}^{o} & v_2\delta_{h1,3}^{o} & v_2\delta_{h1,4}^{o} \\ v_3\delta_{h1,1}^{o} & v_3\delta_{h1,2}^{o} & v_3\delta_{h1,3}^{o} & v_3\delta_{h1,4}^{o} \\ \delta_{h1,1}^{o} & \delta_{h1,2}^{o} & \delta_{h1,3}^{o} & \delta_{h1,4}^{o} \end{bmatrix}$$

$$= \begin{bmatrix} v_1 \sum_{k=1}^{4} w_{h1,1}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_1 \sum_{k=1}^{4} w_{h1,2}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_1 \sum_{k=1}^{4} w_{h1,3}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_1 \sum_{k=1}^{4} w_{h1,4}^{h2,k} w_{h2,k}^{o} 2(o-t) \\ v_2 \sum_{k=1}^{4} w_{h1,1}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_2 \sum_{k=1}^{4} w_{h1,2}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_2 \sum_{k=1}^{4} w_{h1,3}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_2 \sum_{k=1}^{4} w_{h1,4}^{h2,k} w_{h2,k}^{o} 2(o-t) \\ v_3 \sum_{k=1}^{4} w_{h1,1}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_3 \sum_{k=1}^{4} w_{h1,2}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_3 \sum_{k=1}^{4} w_{h1,3}^{h2,k} w_{h2,k}^{o} 2(o-t) & v_3 \sum_{k=1}^{4} w_{h1,4}^{h2,k} w_{h2,k}^{o} 2(o-t) \\ \sum_{k=1}^{4} w_{h1,1}^{h2,k} w_{h2,k}^{o} 2(o-t) & \sum_{k=1}^{4} w_{h1,2}^{h2,k} w_{h2,k}^{o} 2(o-t) & \sum_{k=1}^{4} w_{h1,3}^{h2,k} w_{h2,k}^{o} 2(o-t) & \sum_{k=1}^{4} w_{h1,4}^{h2,k} w_{h2,k}^{o} 2(o-t) \end{bmatrix}$$

Optimization

Here we can see, the gradient matrix carries the accumulated weights across all the previous layers, thus, potentially lead to vanishing or exploding gradient. After derived the gradient matrices and weight matrices, we can use stochastic gradient descent (SGD) method to train the network. Other optimization methods include Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm and Conjugate Gradient (CG) algorithm. For a very large dataset, SGD is the only viable method because the Hessian matrix size could become too large; while for smaller sets, BFGS and CG methods are generally faster than SGD methods. The theory regarding the performance of these algorithms are covered in CSCI 5302(Analysis of Numeric Algorithm), thus it is not a key part of this independent study and will not be discussed in details here.

Stochastic gradient descent method uses a parameter called learning rate. For small dataset and small iterations, it is safe to use a constant learning rate to get close enough to a local minimum. However, if the iteration number is large, it is necessary to implement learning rate decay for the SGD method to work. As parameters, we can define the decay ratio and decay period, for example, we can set the learning rate to decay by 50% after every 20 epochs (1 epoch is an iteration on the entire training dataset). Often, there will be an accuracy jump on the training accuracy (on the test accuracy as well if overfitting is not too severe), because the reduced learning rate allows the network to further descent down a valley, instead of oscillating.

Overfitting and Underfitting

At this point, the mathematical framework of a simple fully connected feedforward neural network should be clear. However, in order to have an accurate neural network, few design guidelines should be considered. Intuitively, we know that the more complex the network is, the more capable the network will be; just like the higher degree of a polynomial, the more complicated line shapes it can fit. This means, trying to solve a classification problem using an overly complicated network might achieve undesired results by trying too hard to fit all the

noise in the training data and produce a classifier that are not general enough to work on new data. The following graphs illustrates an overfitting scenario:

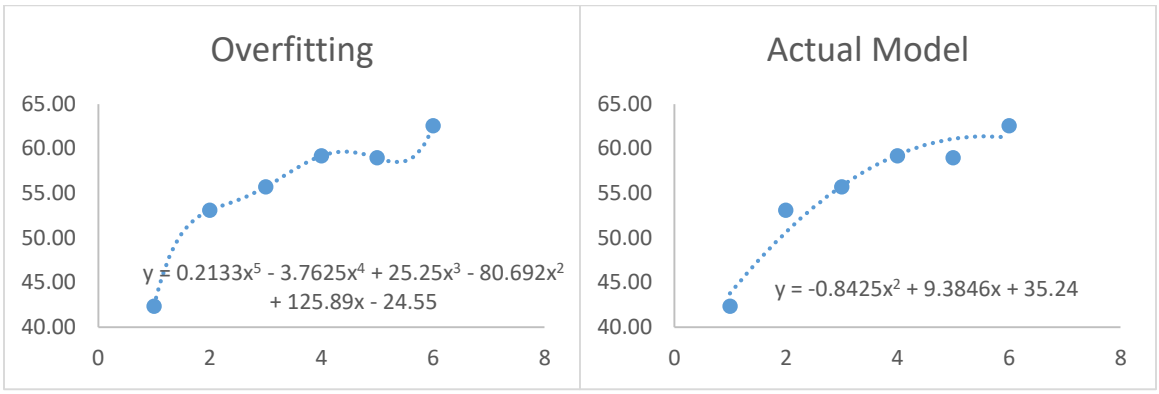

Figure 2 Overfitting example

Typically, when overfitting occurs, we will see a very high classification accuracy on the training set, but a much lower accuracy on the test set. This is because the network has learned the features that can classify data in the training set, but those features are not general enough to be used to classify new data.

The opposite of overfitting is underfitting, in which the neural network is too simple to reach the actual classification complexity of the problem. Below is an example of underfitting:

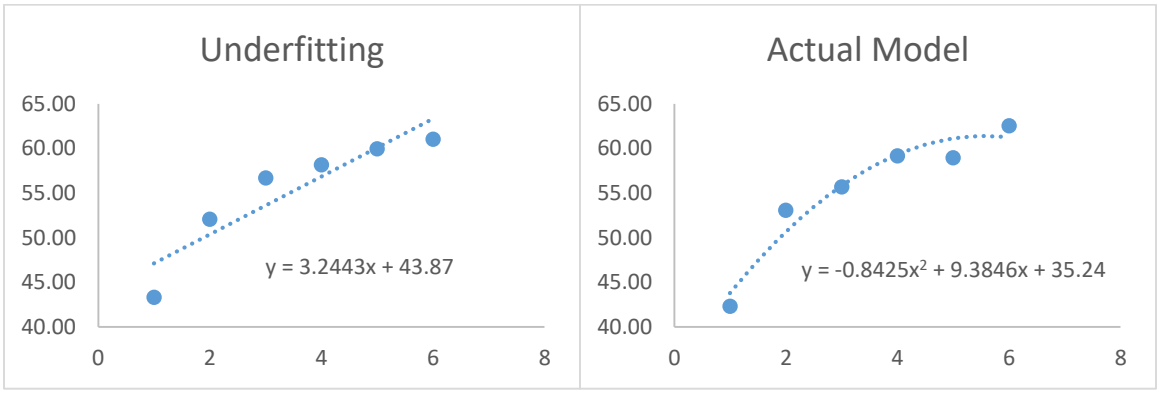

Figure 3 Underfitting example

When underfitting occurs, the network will have difficulties to achieve good accuracy on the training set.

Regularization

To improve a poorly performing neural network, we first need to know whether the network is suffering underfitting or overfitting using the diagnostic tips described previously. A underfitting problem can be resolved by simply increasing the network size or depth. More often, we will have to deal with an overfitting network and there are multiple techniques that we can use to tackle such problem.

The first and foremost technique is to make sure we have a large enough training set. A good indicator to use will be the number of examples to number of parameters ratio. The examples are the instances of labeled data in the training set. The number of parameters are the input layer size. Typically, the minimum ratio we should have in order to expect a reasonable training results is 10. This is not a mathematically derived number, but rather a rule of thumb used by many experienced researchers. In the CIFAR 10 dataset, we have 50,000 training images and the input size is 3,072 (3X32X32). Therefore, the overall ratio is 16.3 for the CIFAR 10 set and the dataset is large enough.

The second regularization technique is called L1 and L2 regularization [2]. This technique works by imposing penalties on the network evaluation criterion if the network weights are too large, thus prevent the classification boundary to curve too much around the data, and reduce the likelihood of overfitting. L2 regularization penalizes the network by adding a value proportional to the square of each weight parameters, while L1 regularization penalizes the network by adding a value proportional to the absolute value of each weight parameters:

$$L1\ penalty = \lambda_1 |w|$$

$$L2\ penalty = \lambda_2 w^2$$

$\lambda_1$and λ_2 are the coefficients that can be defined for each neural network in order to control the degree of regularization.

Dropout

Another very useful regularization technique is called dropout [3]. It involves randomly dropping neurons on a specified layer based on a given dropout probability during training in order to prevent the co-adaptation of neurons. If dropout of 50% probability were to be implemented on all hidden layers of the neural network illustrated in the Figure 1, we will have following network during the training:

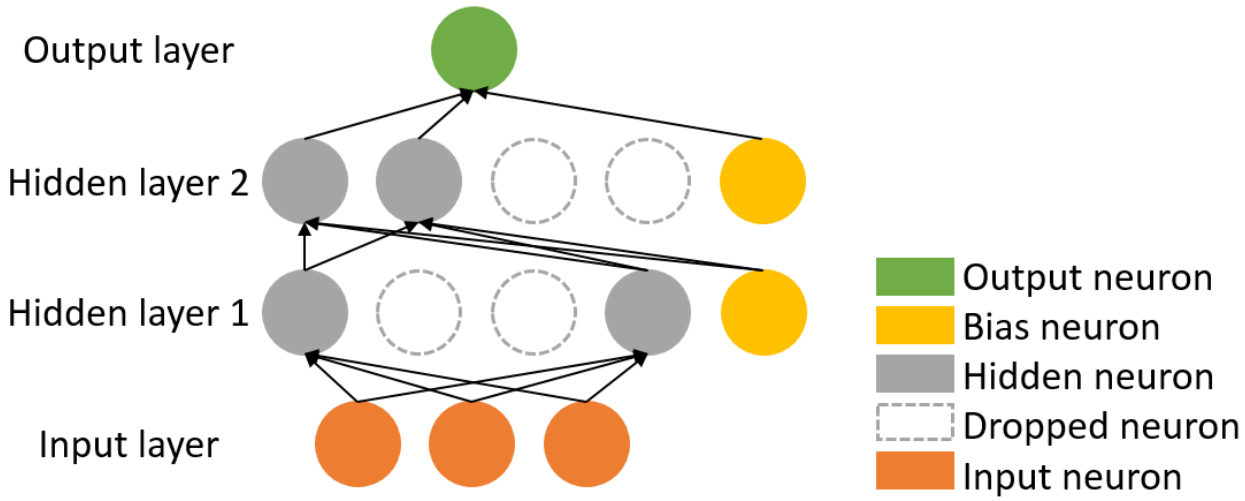

Figure 4 A simple feed-forward deep neural network with dropout regularization

Because the dropout occurs probabilistically, a hidden neuron cannot co-evolve with another, because they are likely to be exposed to different inputs during the training. During the test, dropout will be disabled, and its connection weights will be factored by the dropout probability associated with the layer.

Batch normalization

Last but not least, batch normalization [4] can also provide some regularization to the network, thus preventing overfitting. Batch normalization reduces the effects of outlier sample by using normalized mini-batch. Therefore, gradients based on mini-batch has better generality for classification compared to gradients based on a single sample. Other benefits of using batch normalization method includes the acceleration of training speed, because batched training method can utilize more efficient training data vectorization and better gradient quality also allows high learning rates to be used.

Implementation

Torch has been used to implement the artificial neural network for this project. Torch is a scientific computing framework, which uses Lua scripting language, and it supports multi-dimensional tensor computation. It includes an interface to C via LuaJIT. The main reason for choosing Torch as the primary tool is because that Torch has many libraries available. For example, the nn (which stands for neural network) library, enables the modular construction of a neural network. So, users, in most situation, do not need to construct model tensors; instead, users will specify the hyper-parameters that are necessary to construct a tensor and the structural logics of how those tensors (layers) are connected. This not only makes the

programming must easier, but also makes the code a lot more readable, and that model changes can be implemented faster.

Following code is will generate the neural network illustrated in Figure 1:

```
require 'nn'
model = nn.Sequential()
    model:add(nn.Linear(3,4))
    model:add(nn.Linear(4,4))
    model:add(nn.Linear(4,1))
```

Compared to previous matrices layout, this code hides most of the non-essential details in the background, yet it still has enough flexibility to allow user to configure the network as desired. The default weight initialization used by Torch follows the method described in the paper "Efficient Backprop" by Yann LeCun [5].

In image processing, every pixel-value can be treated at an input. Therefore, a 1000 by 1000 pixels RGB image would have 3,000,000 inputs. Using fully connected network, such a large number of inputs would lead to an enormous network and would be practically impossible to train. Thus, when it comes to image processing, convolutional neural networks are often used. In a convolutional neural network, a window of sub-image is connected to a single neuron in the next layer, the weight parameters associated with the window is called filter. Often, many filters are used on the same sub-image to produce the corresponding number of neurons in the next layer. Also, the window is swept through the entire image, as the name convolution entails, to produce a four dimensional matrix on the next layer. In the case of the 1000 by 1000 RGB image, if we use 16 filters with 10 by 10 window size and stride of 2 (the distance between two adjacent windows), the second layer will have the size of 16 X 46 X 46, total of 33,856 neurons. Between the first layer and second layer, the number of weight parameters is only 16X10X10X3=4,800 (exclude biases), since we are applying the same sets of filter across the entire images. However, if we were to fully connect the first layer and the second layer, we will have 101,568,000,000 weight parameters (exclude biases)! Thus we can see, convolution is not only efficient, but also practically necessary.

First Generation Network

The primary purpose for the first network is to experiment with various network architecture and optimization methods. Following figure is a graphical representation of the model:

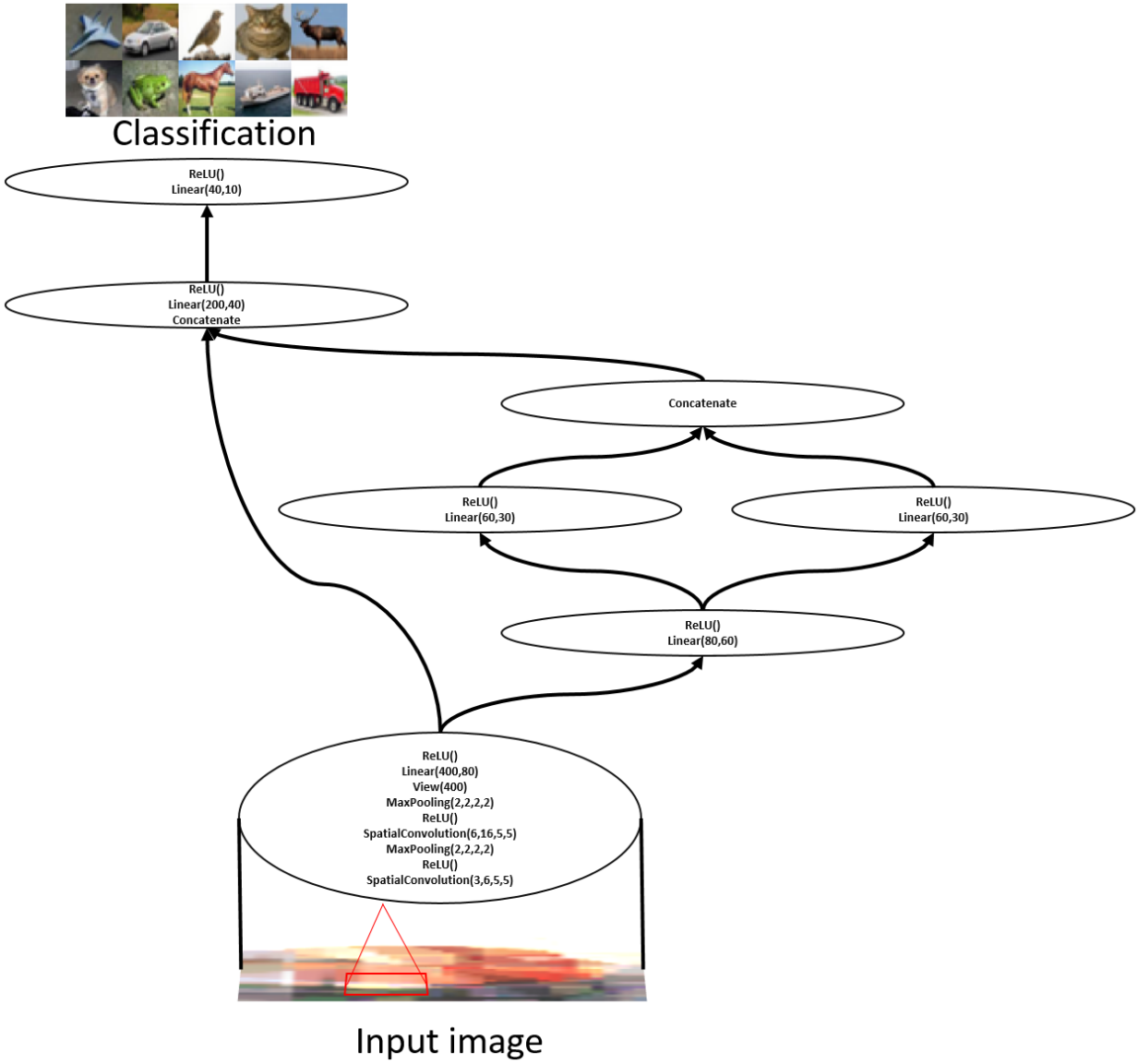

Figure 5 First Generation Network Diagram

As stated previous, the primary purpose of this network is to experiment with different construction of a neural network using Torch. So, the classification accuracy is not the primary concern yet. After the training, overfitting problem has been confirmed by training and test log information shown below:

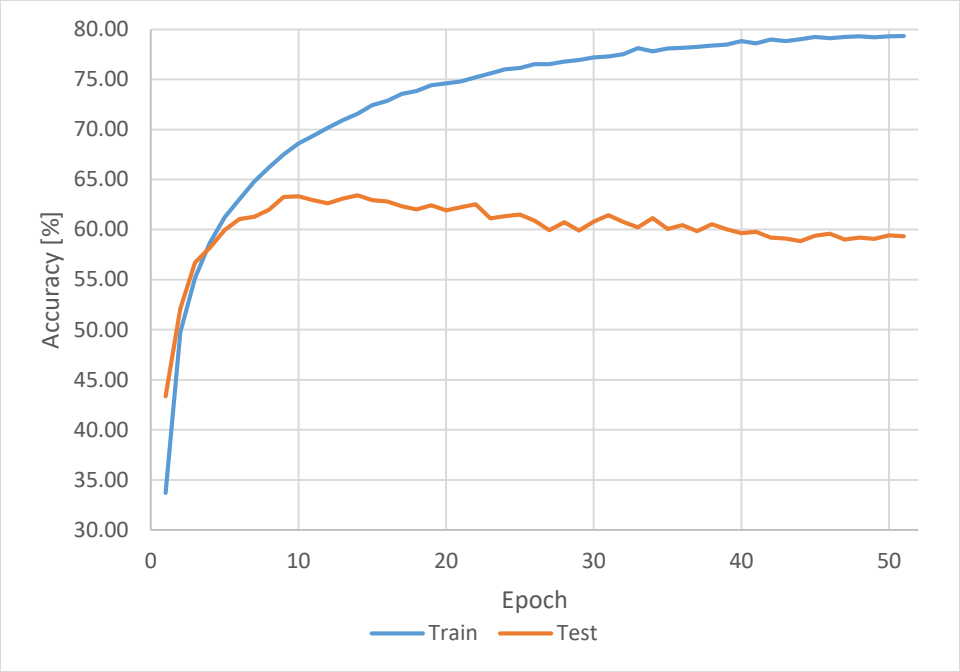

Figure 6 First Network Training and Testing Accuracy Log

From the figure above, we can see that in the very beginning of the training (under 5 epoch), the test accuracy is actually higher than the training accuracy. This is because that the training accuracy on individual images improves very fast in the beginning, while the training accuracy recorded is only the average accuracy throughout an entire iteration. And the difference between training accuracy and test accuracy always starts out to be small and grows as the training proceed. Therefore, in the beginning, the test accuracy is closer to the maximum training accuracy within each iteration, which is significantly larger than the average training accuracy. As the training proceed further, the differences between the train accuracy and test accuracy grows wider till convergence.

This network implemented L2 regularization with L2 coefficient to be 0.001. Even with L2 regularization, after 10^{th} epoch, the test accuracy starts to drop and the differences between train and test accuracy widens, which is a sign of overfitting. Even though, the training accuracy has not converged, it is not necessary to continue the training since the test accuracy has not made any progress after epoch 10. A practical stopping point that I found to be useful throughout this study in order to save time is 5 to 10 epoch after the first consecutive decrease in the test accuracy or 5 to 10 epoch after the first learning rate decay period. This

Second Generation Network

The second convolutional neural network is structurally simply compared to the first one. A graphical representation of the model is shown below:

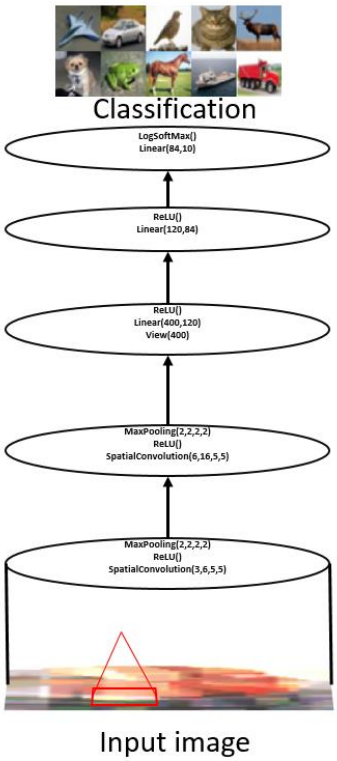

Figure 7 Second Generation Network Diagram

The number of optimizable parameters (number of connection weights) in this neural network is similar to the first generation model, thus, it is expected that this network will have similar capability as well. This network implemented the same L2 regularization with the same coefficient. The training and testing log validated our expectation and we have obtained similar overfitting optimization curves to the first one.

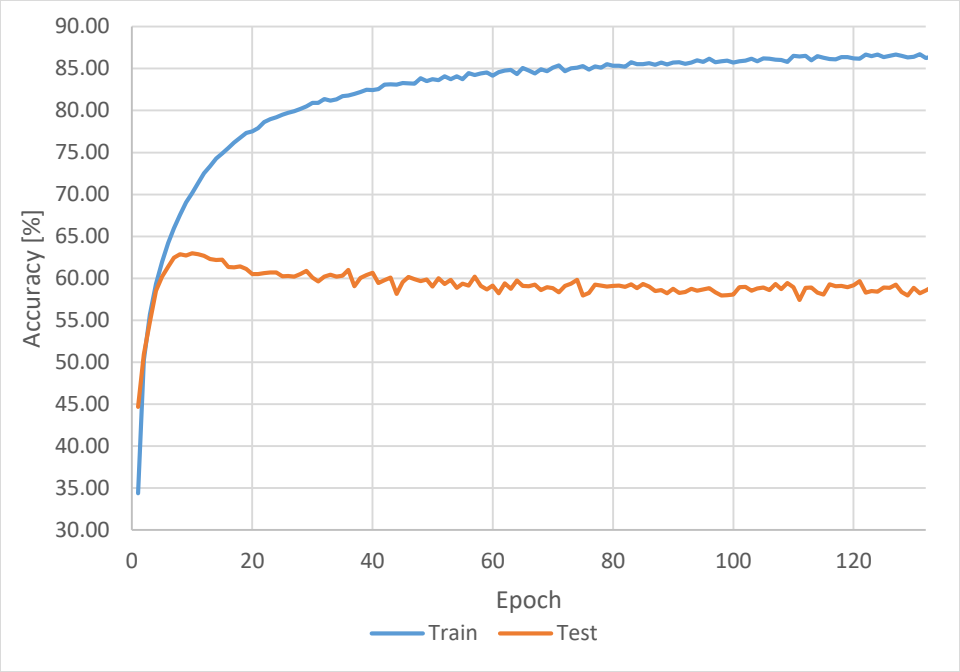

Figure 8 Second Network Training and Testing Accuracy Log

As an experimental study, it is worth finding out what would happen if we were to further simplify the network and reduce its complexity, which led to the third generation neural network.

Third Generation Neural Network

The third generation neural network's capability has been significantly reduced compared to previous ones, containing less than one tenth the number of connection weights contained in previous networks. The object for attempting to train such a simple network is to estimate the minimum complexity required for the classification task and get a generalized starting point for adding complexing later. It is necessary to use a network more complex than the minimum requirement, especially with more regularization techniques used, however, an overly-complex network will make regularization task much more difficult, since the network will learn more redundant features.

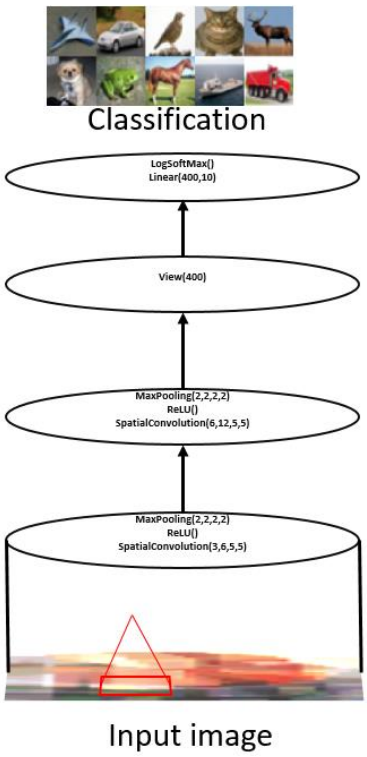

Figure 9 Third Generation Network Diagram

Compare to the second generation network, this network has removed the second fully connected layer and reduced the filter number from 16 to 12 on the second convolutional layer. Even though, the train accuracy has significantly reduced compared to previous network, the test accuracy is at similar level:

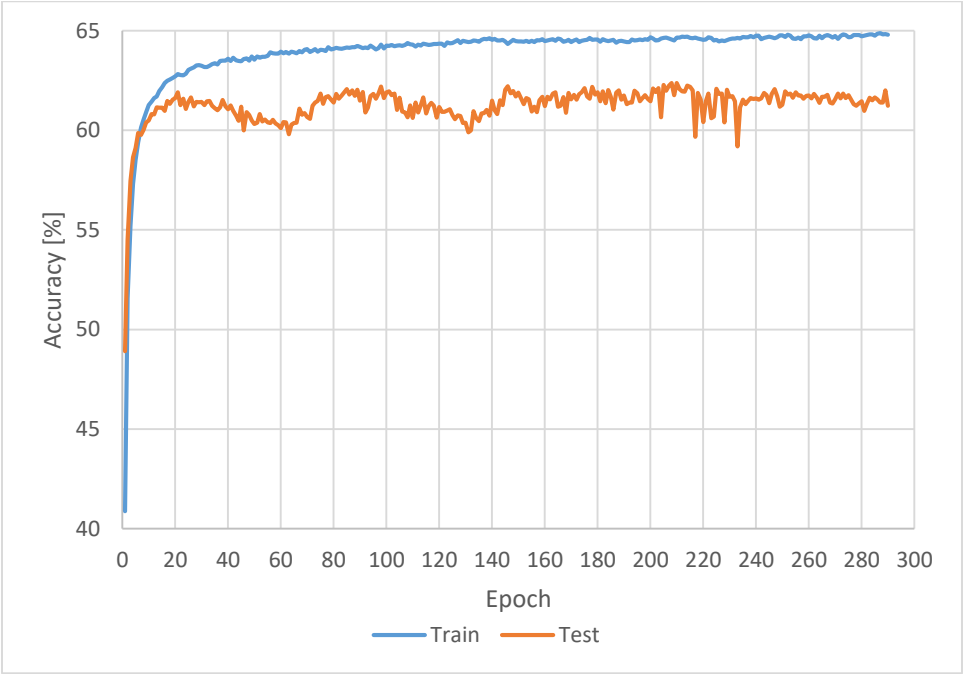

Figure 10 Third Network Training and Testing Accuracy Log

Because the relative small gap between the train accuracy and test accuracy, I know that the existing optimization generalize well over the test data. I can incrementally increase the complexity of the network, and regularize each time I increment the complexity of the net to keep the gap between the train accuracy and test accuracy as small as possible.

Fourth Generation Network

Now, I can try to increase the network complexity from the third generation, then use necessary regularization technique to maintain the generalization of the network. The diagram on the left shows the structure of the network with increased complexity from the third generation and the diagram on the right shows the structure of the network on the left with added dropout regularization. Compare to the third generation network, the new network increased the number of filter from 12 to 64.

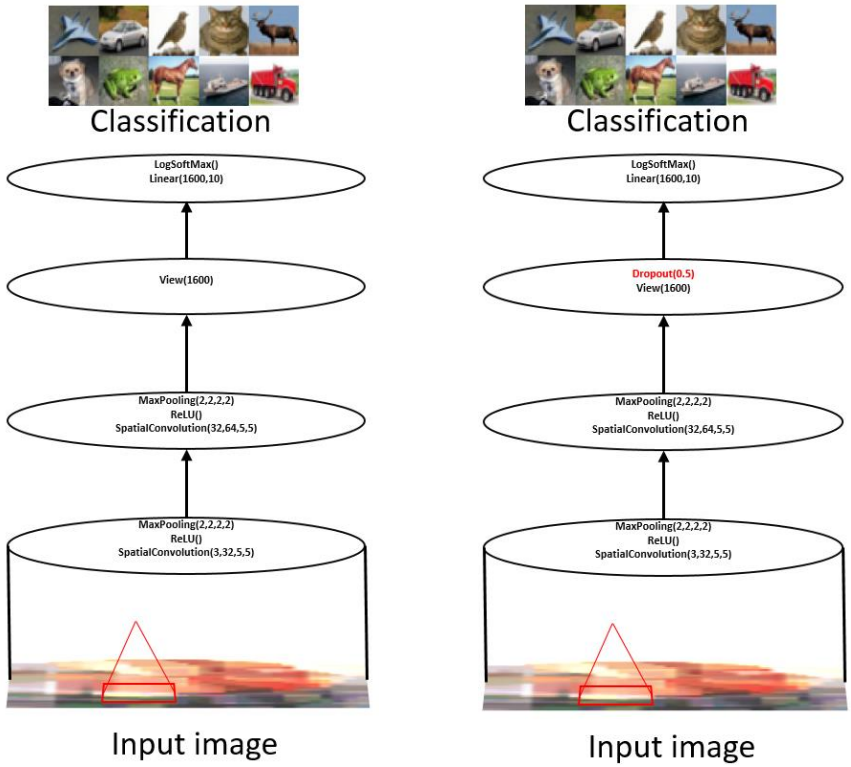

Figure 11 Fourth Generation Network Diagram with Un-regularized and Regularized Structure

The train and test log data are plotted in the figure below:

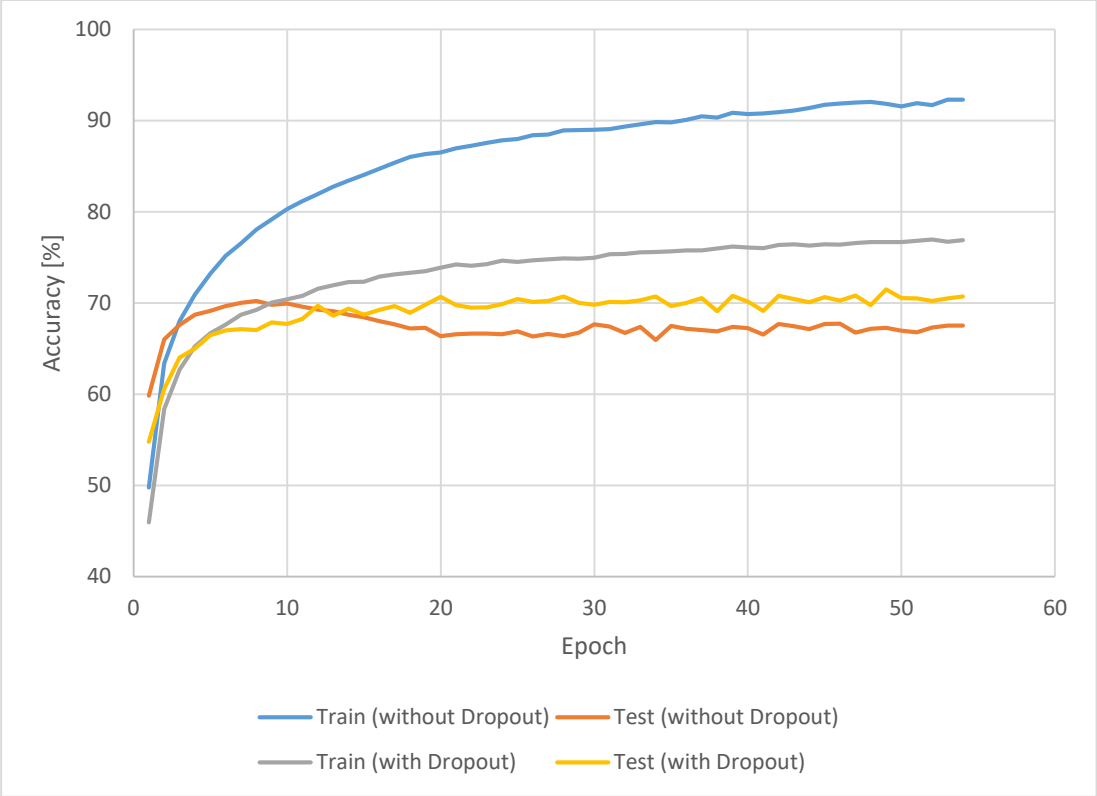

Figure 12 Fourth Network Training and Testing Accuracy Log

Without the dropout regularization, the added complexity improved train accuracy much more than the test accuracy, meaning the additional capability are mostly not general enough to be useful. However, with the implementation of the dropout regularization on the fully connected layer, those redundant capability has largely been eliminated. Even though the train accuracy reduced significantly, the test accuracy actually improved compared to an un-regularized net, and the differences between the train and test accuracy is maintained at a similar level compared to the third generation network. Such incremental approach can be done iteratively to build a more capable and general classification network.

Results and Limitation

The maximum classification accuracy on the CIFAR 10 dataset achieved in this study is 71.91%, which is lower than the accuracy in the benchmark paper [6] used for this project, 78.9%. Nevertheless, it is a significant improvement over the first neural network. The biggest challenge of achieving good classification results is resolving overfitting. Underfitting is fairly easy to detect and fix, it can be fixed by increasing the network complexity. However, to resolve overfitting problem, it involves using more sophisticated regularization techniques. In this

study, three regularization techniques, L2 regularization, dropout and batch normalization, are used. Though, more in-depth study could be conducted on each regularization technique given more time.

Another constraint is the computational power available for this study. The study uses virtual machine running Ubuntu 16 operating system with two cores and 10 GB memory. Ideally, the program would utilize a CUDA capable GPU to parallelize the computation and achieve fast training. But the host computer does not have a CUDA capable GPU, therefore, all the training is done using CPU. On average, the training took about 8 hours (overnight) in this study.

Learning

Through this study, I have learned the basics theories of the feedforward neural network, and the construction of a convolutional neural network for image classification task. The learning included following concepts:

Main Area	Key Concept
Theory	• Input layer, hidden layer, output layer • Bias • Transfer functions (ReLU, SoftMax, Tanh) • Convolution filter
Data Processing	• Normalization • Color space transformation • Dataset size evaluation
Training	• Stochastic gradient descent (best for large dataset) • Conjugate gradient descent and BFGS (best for small dataset) • Newton's method (not practical for large network) • Learning rate and how to choose learning rate • Momentum • Criterion for network evaluation • When to stop training • Network performance diagnosis
Regularization	• L1 and L2 regularization • Dropout • Batch normalization
Evaluation	• Overfitting • Underfitting
Programming	• Lua Programming Language
Tool	• Torch, iTorch(by Facebook)

All neural network models and training/testing log information can be found through following link:

https://github.com/11dtech/gitTorch

Reference

[1] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Aistats*. Vol. 9. 2010.

[2] Ng, Andrew Y. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance." *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

[3] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580* (2012).

[4] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

[5] LeCun, Yann A., et al. "Efficient backprop." *Neural networks: Tricks of the trade*. Springer Berlin Heidelberg, 2012. 9-48.

[6] Krizhevsky, Alex, and G. Hinton. "Convolutional deep belief networks on cifar-10." *Unpublished manuscript* 40 (2010).