Introduction to Deep Learning Project 2 Report

### Muhammet Oğuz Özcan

### oguz.ozcan@ozu.edu.tr

**Abstract**

In this project, CIFAR-10 data set is used to train and test a convolutional neural network for a multiclass classification problem. Ridge and Lasso regularization methods will be used during the training. The data consists of 10 different classes with 60000 images per class. How values of weights changes per epoch is analyzed and commented. The roles and differences of two regularization methods are commented.

# Introduction

The focus of this project is to gain experience on Convolutional Neural Networks by solving a classification problem with CIFAR-10 dataset. The architecture of the CNN is explained in detail. Furthermore, two different regularization methods, which are Ridge and Lasso is applied during the training and their affects and differences on the training is analyzed in this study. The changes on the weights are examined during each epoch, plotted and also commented.

# Problem and Dataset Information

The problem is to make a *classification* on the CIFAR-10 data set using CNN and two different regularization methods.. The data set is available on <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz> address for python implementation which is used in this project. Data set includes total of 60,000 32\*32 images in 10 classes, with 6,000 images per class. The 50,000 images are used for training and 10,000 is used for testing purposes. The dataset is divided into ***five training*** batches and ***one test*** batch, each with 10000 images. Test batch contains exactly 1000 randomly selected images for each class. Below, 10 classes and sample images are shown in the figure.

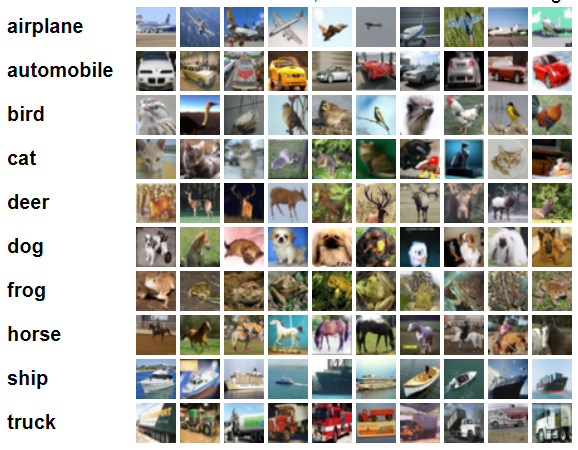


Figure 1: Random images from data set and 10 classes

Once the data is downloaded, it comes in a big tar file, which is about 170 MB size. The archive contains five training data called “data\_batch\_1, data\_batch\_2, ..., data\_batch\_5”, as well as test\_batch. Each of these files are pickled using cPickle [3], which is a serialization method used in Python. So, these files need to open and after that they return a dictionary.

The data can be opened like the following in python3:

def unpickle(file):

import pickle

with open(file, 'rb') as fo:

dict = pickle.load(fo, encoding='bytes')

return dict

If data is loaded in this way, each batch files contains dictionary with the data and label elements, their explanations are given below: [1]

* data -- a 10000x3072 [numpy](http://numpy.scipy.org/) array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.
* labels -- a list of 10000 numbers in the range 0-9. The number at index *i* indicates the label of the *i*th image in the array data.

The dataset contains another file, called batches.meta. It too contains a Python dictionary object. It has the following entries: [1]

* label\_names -- a 10-element list which gives meaningful names to the numeric labels in the labels array described above. For example, label\_names[0] == "airplane", label\_names[1] == "automobile", etc.

# Pre-processing

Since the TensorFlow is thought in this course and also suggested by the instructor, it is chosen for the implementation of the project. It is a widely used open source library for numerical computations. It supports many languages like Java, Python and C++ and Python API is used in this project. It has support for both CPU and GPU and since the computer used for this project does not hold a good GPU, the CPU supported version of the TensorFlow with version 1.5.1 is used.

Actually, in [4], downloading and preparing the CIFAR-10 data is easy as just calling the ‘maybe\_download\_and\_extract()’ function. However, this function gives an interesting error, which is ‘UnrecognizedFlagError: Unknown command line flag 'f'’. I tried to solve this problem; however I could not manage that. Therefore, I get help from some other open source projects which I’ve added to references section. I manually downloaded the images and load the training, test and labels in separate global array with some processing. This process was time consuming actually.

After downloading and preparing the data, data was in shape of (50000, 3072) and (10000, 3072) training and test data set respectively. To convert this to (50000, 32, 32, 3) and (10000, 32, 32, 3) shapes and function is used named ‘prepare\_input(data=None, labels=None). This function is taken from [5]

# Learning Algorithms

Convolutional Neural Networks is used in this project. To avoid overfitting of the neural networks there are some techniques exist like, cross validation sampling, reducing number of features, pruning and regularization. Regularization is used in this project and specifically Lasso and Ridge methods will be applied. What regularization does is, basically it gives penalty as the network model gets complex. [7]

Every dataset has noisy samples. The inaccuracies can lead to a low-quality model if not trained carefully. The model might end up memorizing the noise instead of learning the trend of the data. A visual example of a nonlinear over fitted model:

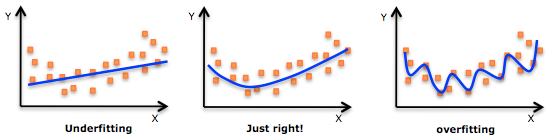


Figure 2: Underfitting and Overfitting

Ridge and Lasso regularization methods will be applied during training. Let’s explain what these methods are:

Lasso regularization: The abbreviation for “*Least Absolute Shrinkage*” and “*Selection Operator*” method, also called **L1** regularization. This method adds the absolute value of the coefficient as penalty to the loss function.



Figure 3: Lasso regularization

Ridge regularization: This is also called **L2** regularization or weight decay. This method adds squared magnitude of the coefficient as the penalty to the loss function.

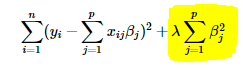


Figure 4: Ridge regularization

The difference between Lasso and Ridge regularizations is that:

Ridge regression enforces the β coefficients to be lower, but it does not enforce them to be zero. That is, it will not get rid of irrelevant features but rather minimize their impact on the trained model. Lasso method overcomes the disadvantage of Ridge regression by not only punishing high values of the coefficients β but actually setting them to zero if they are not relevant. Therefore, we might end up with fewer features included in the model than you started with, which is a huge advantage.

The network architecture can be seen in tensorboard. The tensorboard code is added to the jupyter notebook codes provided with this report. The tensorboard output is also provided in the below figure.

The methods used in this architecture are given below: There are three Conv-> Relu->MaxPool layers followed by a fully connected layer and a softmax at the at.

The methods used in this architecture are given below:

CONV -> RELU -> MAX\_POOL -> CONV -> RELU -> MAX\_POOL -> FC -> SOFTMAX

At first CONV layer, 32 5 \* 5 \* 3 filters are used. In the second CONV layer, 64 5 \* 5 \* 32 filters are used. At last 128 5 \* 5 \* 3 filters are used.

After the first trial, I’ve used Google Colab, which is explained detail in below. The network that is tried in Google Colab is:

CONV -> RELU -> MAX\_POOL -> CONV -> RELU -> MAX\_POOL -> CONV -> RELU -> MAX\_POOL -> CONV -> RELU -> MAX\_POOL -> FC -> SOFTMAX

At first CONV layer, 64 3 \* 3 \* 3 filters are used. In the second CONV layer, 128 3 \* 3 \* 32 filters are used. In the third layer 256 3 \* 3 \* 3 and at last 512 3 \* 3 \* 3 filters are used in the conv layers.

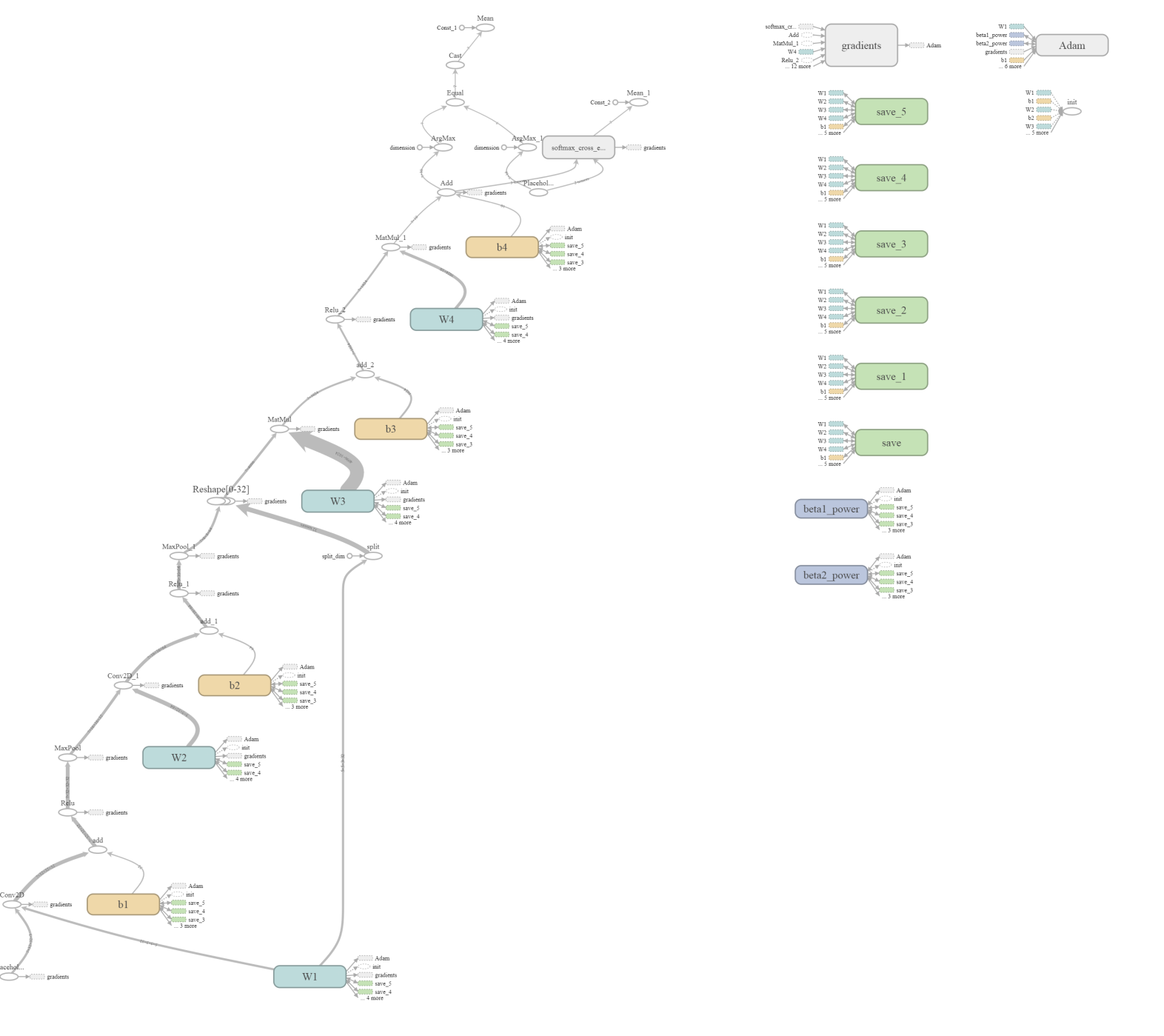


Figure 5: Network architecture drawn with tensorboard

## Convolutional Layer

This layer is the core of the CNN architecture where the most of the computation is done. The calculation is done using filters as weights, for example in size of [3, 3, 3] where the last digit represents the channel number.

## ReLU Layer

ReLu is one of the most popular activation functions used in Deep Learning community. Activation functions are used in the neurons to decide if the computation made in one neuron should matter or affect the others. Why ReLu is chosen among sigmoid or tanh. Because in big neural networks, which we will have in this project, sigmoid or tanh will process all the inputs and they produce an output which is not zero, in other words, all the output of these activation functions will be processed in the following neurons. This increases the computation complexity and time. However, in ReLu almost half of the network will produce zero result since ReLu creates 0 for negative values. This decreases the computation complexity. Moreover, the gradient of the ReLu function is constant. The constant gradient of ReLUs results in faster learning.

## Pooling Layer

In pooling layer, the spatial size is decreased and therefore the computational complexity is reduced by reducing the number of parameters. Inserting a pooling layer between Conv layers is a common applied technique. In our network

## Fully Connected Layer

Fully connected layers are generally used before output layers to flatten the high level features that are come out of Conv layers.

## Softmax

At the output, Softmax function is used. Since we are dealing with classification problem in this project we need to use softmax. What softmax does is, it squashes each unit to be between 0 and 1 and also the total sum of all units adds up to 1. The output of softmax creates a categorical probability distribution. It shows which of the classes that this input belongs to with the highest probability.

# Evaluation

It is required in the assignment that the network must be saved onto disk. After some research, it is found that this can be done via ‘tf.train.Saver’ class provided in Tensorflow. [6] tf.saved\_model.simple\_save function easily saves the network in ‘model\_dir’, which is the directory that I defines as “/models\_no\_regression/” for the network that no regularization is used, “/models\_lasso/” is for network that lasso regularization is used and finally, “/models\_ridge/” is used for the network that ridge regularization is used.. After saving the network it can easily be restored using the following two lines of code:

new\_saver = tf.train.import\_meta\_graph(model\_path)

new\_saver.restore(sess, tf.train.latest\_checkpoint(checkpoint\_path))

After restoring the saved model, now the model is tested with the test data.

***Problem:***

***I’ve tried to save and load the model, however even thought I was successfully saved the model, when I load the model, I could not run the session and obtain the losses, therefore, I’ve tested my network with the test data in the same notebook as in the train data.***

Now the results that are obtained on trying the aforementioned algorithms are presented.

## Weights with no Regularization on Training

At first, I started to train my network with no regularization. Since I am using CPU based TensorFlow, I cannot use the whole set of data in once, so I am sending the train data in batches of size 1000 in 50 cycles. The epoch in other words the whole data is used 25 times. The accuracy is 92.88%. The change in the number of zeros and non-zero weights in the first weight which is W1 are given in the below figure:

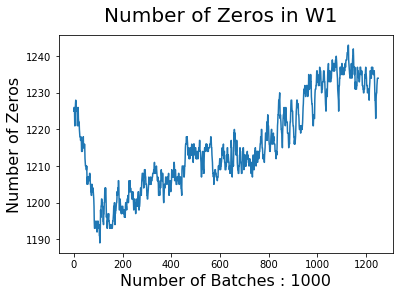
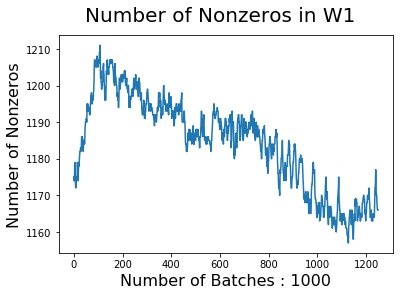


Figure 6: Change of the zero and non-zero weights per batches in W1

Then, I run the network with test data, the result of the accuracy was 66.87%, which is not a good solution, means my network is not strong enough.

I’ve tried to use Google’s Colab to train faster. I’ve encounter lots of problems, especially at downloading the cifar-10 data set to the Google Drive. However, with the help of [8], I’ve managed to do that, which is a lot faster and easier to train. But, I still could not manage to open saved model, so I still do train and testing in one jupyter notebook. I’ll add second wetransfer link to the end of document for the Colab networks.

The network I’ve used in Colab is runned with 50 epochs in 1000 batches. In training I’ve obtained 82.98% success and in the test I’ve obtained 70.12% success, which is acceptable I assume. The figures of the colab network is given below. For the sake of simplicity, I’ve only plotted changes in the first weight which is W1.

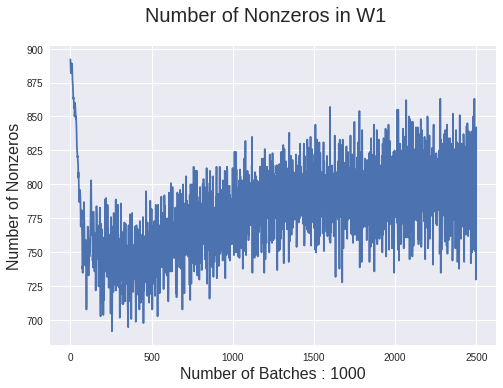
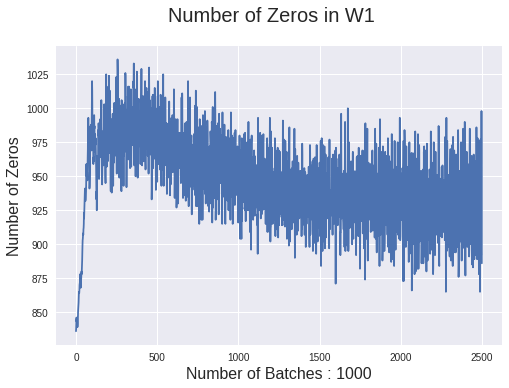


Figure 7: Change of the zero and non-zero weights per batches in W1 in Colab Network

Then, I added Lasso regularization, which is explained in section 5.2

## Weights with Lasso Regularization on Training

In this time, I added Lasso regularization as shown in below code piece to the architecture and observe the change in weights.

#Create L1 regularization

l1\_regularizer = tf.contrib.layers.l1\_regularizer(scale=0.01, scope=None)

regularization = tf.contrib.layers.apply\_regularization(l1\_regularizer, vars)

regularized\_loss = s\_loss + regularization

The epoch in other words the whole data is used 25 times. The accuracy is 94.96%. The change in the number of zeros and non-zero weights in the first weight which is W1 are given in the below figure:

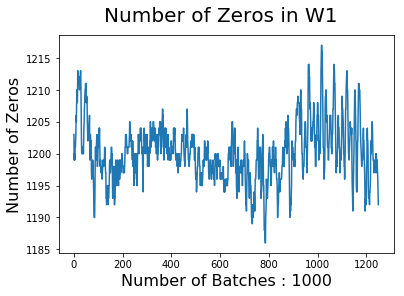
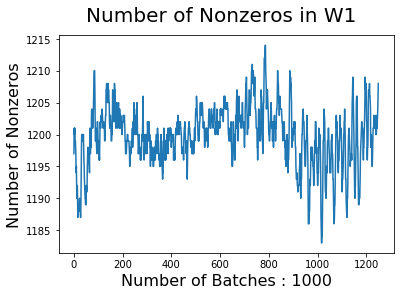


Figure 8: Change of the zero and non-zero weights per batches in W1

Then, I run the network with test data, the result of the accuracy was 66.87%, which is almost same as the non-regularized network.

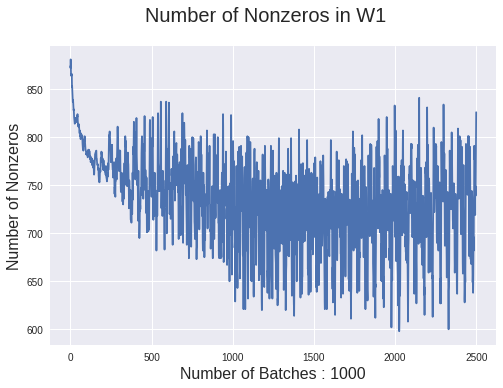
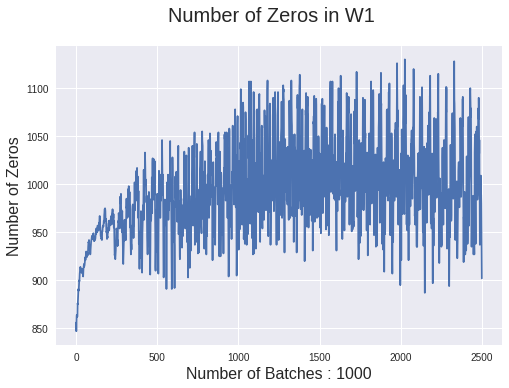
Then, I run the network in Colab with different architecture. While I was running L1 regularized network, at first my network started with 30% success, then it stucked at 10% and not moved. Then I’ve change learning rate of Adam Optimizer, then this time it stucked around 29%. Then, I decrease the learning rate and this time I’ve obtained success rate of 57.26% and test rate was 55.85%. The changes in the weights (W1) are given below.

Figure 9: Change of the zero and non-zero weights per batches in W1

## Weights with Ridge Regularization on Training

In this time, I added Ridge regularization as shown in below code piece to the architecture and observe the change in weights.

The epoch in other words the whole data is used 25 times. The accuracy is 94.42%. The change in the number of zeros and non-zero weights in the first weight which is W1 are given in the below figure:

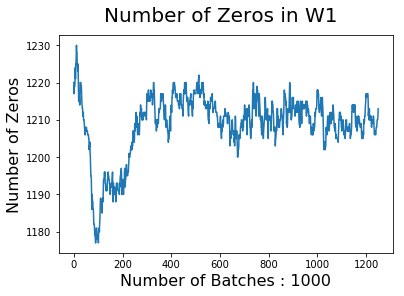
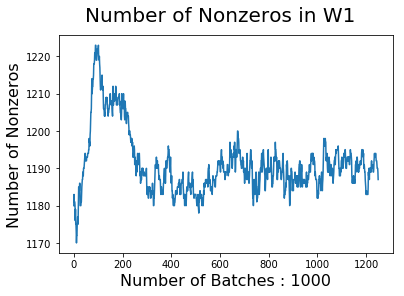


Figure 10: Change of the zero and non-zero weights per batches in W1

Then, I run the network with test data, the result of the accuracy was 65.42%, which is almost same as the non-regularized network.

Then, I run the Colab network with L2 regularization. The success rate of the network is 76.43% and the test data success result is 68.88%. There is a slight increase in the rate. The changes in weights are given below.

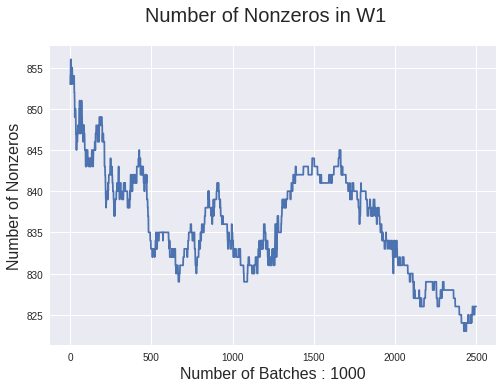
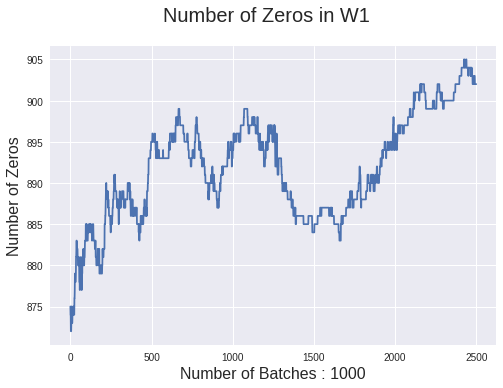


Figure 11: Change of the zero and non-zero weights per batches in W1

I know there is a problem at L1 regularized network, I wish I would like to analyze it more, but because of the problems I’ve faced that I’ve explained in the second paragraph of the conclusion, I could not do that.

# Conclusion

As mentioned earlier, the aim in this project is to get hands-on experience working on a deep learning neural networks problem, understanding CNN and to experience different regularization methods effects to the architecture.

I know my networks are a bit weak for this data set. There are lots of problems I’ve faced during this project. Firstly, it is really hard and trying to train these networks with CPU, it takes hours and my computer crashed a few times. Then I’ve decided to use Google Colab. It is a lot faster, but learning that environment, downloading the data was a pain in and also takes too much of my time. Also, I could not manage to load the saved network. This seems like a very basic task but I think I am missing a tiny point. These three problems take almost 80-90% of my time, and that’s why I could not play much with the network architecture to get a higher success. I’ve learned to use Colab and I hope in the next project I will have more time to play with the architecture.

Link to the created notebook is the following, as mentioned earlier, since I could not manage to run the loaded network, I’ve added only one notebook and saved models also to show that that part is completed.

***Link:*** <https://wetransfer.com/downloads/add61b21fa01aef8226dd164c80a03a520180413150945/688f49c05e577df5bc12ab02c8862a2f20180413150946/a205fc>

***Link for the Colab trined network:*** <https://wetransfer.com/downloads/8b2e52cac9879bb61d3bacebd96705bf20180416064622/062e095347dc4936047ce9098a962b8320180416064623/ee58b0>

# References

[1] CIFAR-10 data set repository - https://www.cs.toronto.edu/~kriz/cifar.html

[2] [Learning Multiple Layers of Features from Tiny Images](https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf), Alex Krizhevsky, 2009.

[3] https://www.python.org/doc/2.5/lib/module-cPickle.html

[4] https://www.tensorflow.org/tutorials/deep\_cnn#highlights\_of\_the\_tutorial

[5] https://github.com/kgeorge/kgeorge\_dpl/blob/master/notebooks/tf\_cifar.ipynb

[6] https://www.tensorflow.org/programmers\_guide/saved\_model

[7] <https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>

[8] https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d