What is neural network and how it works?

First, I began with this question: “What is neural network?”. I learned its design (and “why should it be like that?”), its inputs, outputs, its application fields and its examples (Standard NN, Convolutional NN, Recurrent NN). Then I started to learn its implementation techniques in order to design a basic standard neural network. For example, I have been told that I should avoid using "for loop" in the lessons I have taken so far, but I did not know the methods to do this. When I didn't want to use it, I had to change my method. But there is a way that performs exactly the same function: Vectorization. By using vectorization, I could get the data to be used, in a faster way. Therefore, we have solved the problem of dealing with big data that may come up in the first stage.

Then I continued with binary classification. The first step of understanding binary classification is logistic regression which is an algorithm used when the output labels in a learning problem are either zero or one. In order to train the parameters of a logistic regression model, we should use loss function (used for a single training example) and cost function (for the entire set). Then I have learned the implementation details and logic of gradient descent used to minimize total cost. At first, I had hard times to understand its mechanism but then visualization helped me to handle it. It was like finding the lowest point of a valley, that's what we wanted to do while minimizing cost.

After covering these topics, I did my first and a simple assignment to understand how to combine the subjects I learned. Thus, I could easily use it on more advanced models. I implemented the project with numpy. I began with implementing sigmoid function (an activation function) to labeling the output correctly. Then I computed gradients by using derivatives. Then I took the 3D input image and converted into 1D array by reshaping it. To make the gradient descent faster, I used normalization on data. Then I used softmax function which outputs a vector that represents the probability distributions of a list of potential outcomes. Then I applied vectorization and loss functions. Thus, I implemented the functions I learned, and I’ve experienced on numpy for the first time.

Before moving on to other topics, I implemented a cat recognition model, a simple binary classification example. I worked on the dataset "data.h5". After loading the data and required libraries, I created a model structure. After initializing the model's parameters, I should calculate current loss, gradient and update the parameters respectively. To do this, I did forward propagation, backward propagation (including activation function and cost function). I implemented gradient descent (to get the learned parameters by updating parameters which will give the least cost) in optimization part. Then I implemented test part to see whether the given images include cat or not. Although I got %99 training accuracy, I got 70% accuracy from the test set. As a result, the model needed to be improved.

The result of the project required me to examine the structure more deeply. I used the sigmoid function in the project because I saw it in the examples I examined. But I had to examine the others to find out which activation function was more suitable for my work. So, I have examined the pros and cons of other functions. I learned that I could build more complex structures and get more accurate results by building deeper networks and started another project to implement it. This time I implemented 4-layer neural network and got better performance (with 80% accuracy) for cat recognition model.

Improving my network

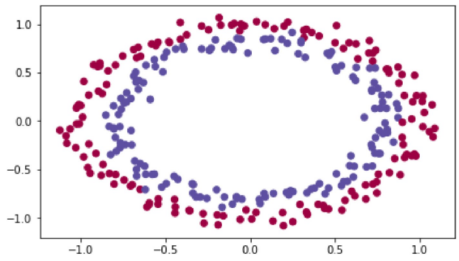
There were some important points to consider when developing my network such as regularization, dropout regularization, normalization, gradient checking, how to divide my data into sets etc. I started with learning how to set up the data. I needed to divide my data into three parts: Train set, development set and test set. I learned that it makes more sense giving the big piece to the train set while working with big data. Because there were still enough data to develop and test the model.

There were some problems I might encounter while testing my model: High bias (which is about training data performance), high variance (which is about development set performance) or both. After understanding how to detect these problems, I learned the necessary techniques to solve them such as using bigger network, training longer, getting more data, regularization (with data augmentation, early stopping) etc. also I learned how these techniques solve those problems. Then I applied this learning in two separate projects: First one is about initialization techniques and the second one is about regularization.

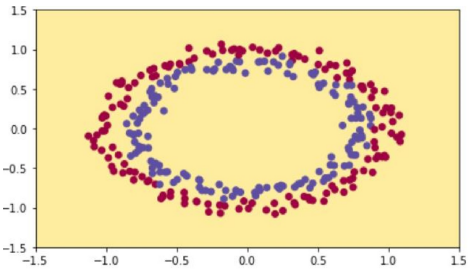
In first project, I implemented three methods (zero initialization, random initialization and he

initialization) while initializing parameters in order to see which works better.

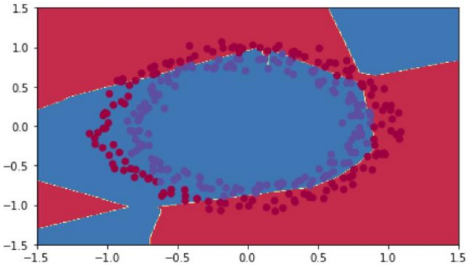
First image after loading the dataset



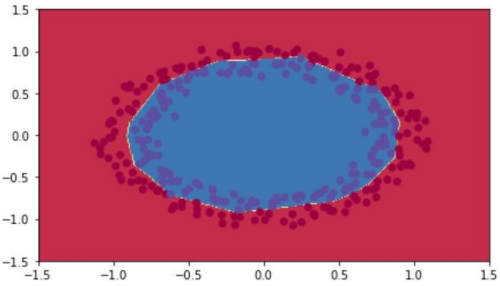
Model with zero initialization



Model with random initialization



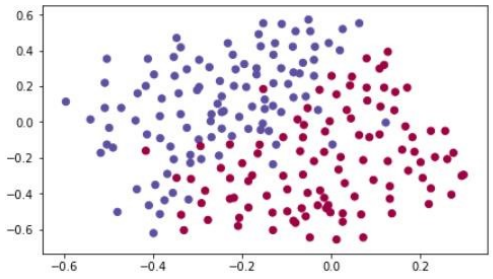
Model with He initialization



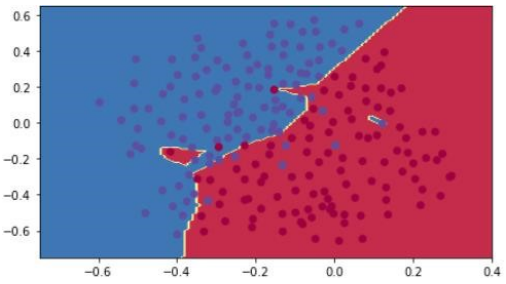
After implementing parameters with zero initialization, I trained model and I realized that the cost value did not change at any iteration. It was always 0.69 (with learning rate 0.01). The accuracy of train and test sets are the same: 0.5. These values meant nothing for the improvement of the model. Then I continued with random initialization. After implementing random initialization, the model seemed to mean something more than zero initialization. The cost value almost fell by half in the first 7000 iteration, but the decreases following it were very low. The accuracies obtained from train and test sets are 0.83 and 0.86 respectively. The areas with blue and red dots, although not exactly, were highly discernible. Later, I tried another method: He initialization. After implementing he initialization, the result getting from the model was much more satisfactory. The cost value dropped significantly compared to previous values, it was 0.07 after 14000^{th} iteration. On the training set, the accuracy was 0.99. On the test set, the accuracy was 0.96. Now the areas with red and blue dots were almost completely discernible. I already knew that different initialization techniques would bring different results, but I learned here that random initialization serves to break symmetry and the ReLU activation function works so well with He initialization.

After learning and implementing initialization techniques, I have moved on to the second project where I will study regularization techniques.

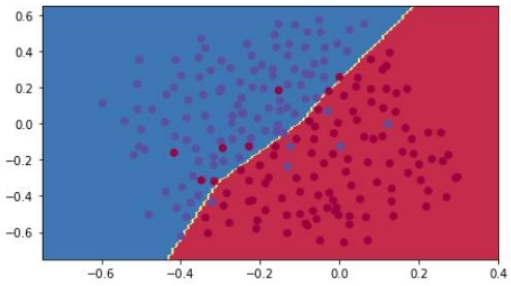
1: First image after loading the dataset



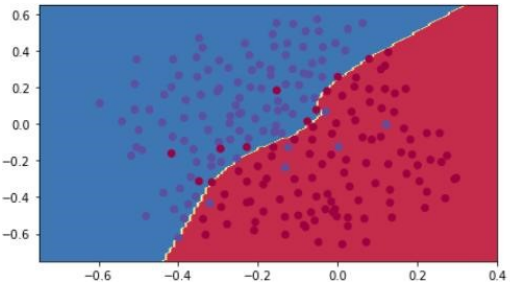
2: Model without regularization



3: Model with L2- Regularization



4: Model with dropout



In this project after loading the dataset, at first, I did not apply any regularization method to the model, and I got the result shown in second image. Cost value declined significantly in the first 500 iterations, but there was no significant change after that (It was 0.163 after iteration 10000, and 0.138 after iteration 20000). The train accuracy was 94.8% while the test accuracy is 91.5%. Areas with blue and red dots could be distinguished but the model was overfitting the training set (because of high variance). Therefore, the model needed to be improved by reducing overfitting. So, I had to apply one of the techniques to handle the high variance problem. So, I implemented L-2 Regularization. First, I computed new cost (cross-entropy cost + L2 regularization cost) value for the new model. I reorganized the backward propagation part according to the new cost value. Then the model was trained and tested. Cost value declined significantly in the first 500 iterations, but there was no significant change after that as in the previous model (It was 0.2684 after iteration 10000, and 0.2680 after iteration 20000). The accuracy of the train set was 93.8% and the accuracy of the test set was 93.0%. The accuracy obtained from the test set was more satisfactory. The situation of the model after applying the L-2 regularization can be seen from the third picture. L2

regularization made the boundary smoother. This result was very close to the result I wanted, but I also implemented it to see the result of dropout regularization. I implemented dropout method by choosing only a subset of the neurons (84% of them) in the model at each iteration which means I modified my model at each iteration. Cost value progressed like other models but the last value of it was much better. It was 0.06 after iteration 20000. The accuracies of the train and test sets are 92.8% and 95.0% respectively. The test accuracy increased again. And the model was not overfitting the train set anymore. The last situation of the model can be seen from the 4th image above. With this project I have observed the effect of regularization techniques on the model.

I continued with a technique which would make my algorithm faster while processing big data easily: Mini-batch gradient descent. I learned choosing the variables appropriately for this optimization algorithm in order to avoid noisiness and too much time. I learned exponentially weighted averages, bias correction and gradient descent with momentum to understand the mechanism of another optimization algorithm: RMSprop. Then I continued with Adam optimization algorithm which I would use more often. I have also learned another method, learning rate decay, which could help me in addition to these algorithms. Because parameter tuning is very important for fast and efficient algorithm, I learned how to do parameter search and also learned batch normalization which makes easier this search. Then I have examined how softmax activation function works on a neural network with more than one hidden layer.

Different frameworks

Until now, I've been implementing deep learning algorithms from scratch using python and numpy. But to implement more complex and deep models such convolutional neural networks, I had to use additional frameworks. There were many deep learning frameworks with advantages and disadvantages compared to each other, I continued with Tensorflow. I got knowledge about how to use by examining code examples. Then I implemented my first project in tensorflow.

In this project, I worked with SIGNS dataset. First, I used tensors that correspond to the variables I have used so far and created placeholders for the first time. I implemented a linear function, sigmoid function. Then I computed the cost. The operations were the same as the ones I have used so far, only my implementation method has changed. I implemented one_hot_matrix function to take one vector of labels

and the total number of classes and return one hot encoding. After initializing the array with its shape, I continued with building my first neural network in tensorflow. I separated the data into two parts: training set (with 1080 pictures (64 by 64 pixels) of signs representing numbers from 0 to 5) and test set (with 120 pictures (64 by 64 pixels) of signs representing numbers from 0 to 5.) With one hot encoding we could show an image of third class with this array [0, 0, 0, 1, 0, 0]. After some operations such as loading the dataset, flattening it, normalizing image vectors and converting training and test labels to one hot matrices, I created placeholder which allows me to later pass the training data in when I run the session. While initializing parameters, I used Xavier initialization for weights and zero initialization for biases. While implementing forward propagation and completing the forward pass, I used ReLU activation function. In order to give the last linear layer's output as an input to computing loss function, I did not calculate last activation. Then I implemented compute cost function using "tf.reduce_mean" and softmax cross entropy. It was time to implement backward propagation and updating parameters. After all functions are completed, model was created by combining all these functions in model function. Now, we could train and test it. I trained the model with learning rate = 0.0001, number of epochs = 1500 and minibatch size = 32. The cost value of the model was decreasing as the number of epochs reaches towards its end. The last value of cost was 0.05 after 1400 iteration. Although the accuracy of training set was 0.99, the accuracy of test set was 0.71. There was again the same problem: High variance. So, I needed to get more data or regularize the model or search a new neural network architecture that would work on a smaller data set. I left this project here because I have previously solved the same problem using one of the regularization techniques (L2 regularization). In this way, I made my first project in tensorflow.

I continued my work by examining the idea: Orthogonalization. And continued with understanding evaluation metrics, optimizing it, how to set up the distributions of train/dev/test sets.

While developing the performance of our model, we face some limiting factors such as human level. Since models are developed by humans, the lack of perfection of humans makes the performance of the model not reach a hundred percent. Then I learned how to avoid this situation. The things to do were also about biases and variances.

To develop my model, I've looked at some other problems I might encounter. Incorrectly labeled data is one of them. It was important to learn when to solve the problem as well as how to solve it. Because at some points, the percentage of the error caused by a problem is too low. Then I learned how bias and variance problem emerge again because of data mismatch and how to address the data mismatch.

Transfer learning is another idea in order to develop my model. By using transfer learning, I could develop my model by transferring the data obtained from another task which was already developed. Another idea that I could use is multi-task learning which generally uses with working the data having very similar tasks. I also studied on end-to-end deep learning, but I did not practice these topics on a project.

Convolutional Neural Networks

I started convolutional neural networks with edge detection and continued extending it. First, I learned the answers of "What is padding?", "What is strided convolution?". I learned to apply the concepts (padding, stride) on a convolution with volume, also learned to create a simple CNN, pooling layers etc. To train very deep neural networks, I learned what is residual neural network how it works and how to implement it. Also I learned data augmentation and the structure of inception network which also allows to create very deep neural network(but never used, because my model was simple). Then I implemented my first convolutional neural network model in Keras.

In this project, the aim is to check if the person on the image is happy or not. After loading and normalizing the data, I created the model. First, I used 2D zero padding to pad the input. Then I applied 2D convolution, batch normalization and relu activation function respectively. The output of this process went to 2D maxpool. After converting the output of it into a vector and applying the sigmoid function, the model was ready to compile. I used Adam optimizer on the model. I trained the model with batch size = 16, epoch number = 40. The epoch number, batch size and optimization method were the most appropriate one's for my model. (I found the right hyperparameters by testing several times.) Then I tested my network. I got 95.3% test accuracy(After applying the right choices). Therefore, I created my first convolutional neural network model using Keras. I was going on another project with a deeper network, but I didn't add it to the report because it's not over yet.

These were my first studies on the neural network. So, they need to be improved in terms of efficiency, accuracy and time.