Artificial Intelligence Project Report

Submitted By: Md Abu Sayed (80718658)

Submission Date: 9th May, 2021

Project Name: Implement Feed Forward Neural Network (One Layer, Two Layer, and Multilayer) from scratch, train them on CIFAR-10 and MNIST dataset, and also analyze their performance on the given datasets in terms of accuracy.

Project Description: Learning form data or examples one of the exciting and challenging topics now a days. Learning form examples are all about find out the correct mapping between input and output data in order to correctly predict the new data. For that, different machine learning techniques are available to address this problem, specially find out the appropriate function or mapping that generalize the relationship of input and output data. At the same time, machine learning models are thinking somewhat black box because of their less interpretability. That’s why explainable artificial intelligence, interpretable machine learning also gained public attention.

In my project, I implement Feed Forward Neural Network with one layer(no hidden layer), Feed Forward Neural Network with two layers(one hidden layer), and Feed Forward Neural Network with multiple(L) layers(L -1 hidden layers). I also apply all these 3 models on CIFAR-10 and MNIST dataset. From chapter-18 of our course text book (Artificial Intelligence A Modern Approach), I learned different machine learning technique for learning from data such as learning decision trees, evaluating and choosing best hypothesis from model selection, loss, regularization, to feature selection, regression and classification with linear models, artificial neural network, and support vector machine. I also read the chapter-11 specially some issues in training neural networks from The Elements of Statistical Learning and chapter-5 from Pattern Recognition and Machine Learning book.

My attended online courses (<https://sites.google.com/view/abu-sayed/online-courses>) also help me to understand what happens inside machine learning model and build my coding skills to develop any machine learning model from scratch without just calling some functions from a framework.

As I have mentioned, I implemented Feed Forward Neural Network with multiple layers. Because of I implemented it from scratch and it has many hidden layers, it takes much time when I run it with whole dataset or a portion of dataset. That’s why I am decided to build some simple model such Feed Forward Neural Network with one layer (no hidden layer) and Feed Forward Neural Network with two layers (one hidden layer). This two models take less time and interestingly, sometime gives greater accuracy than Feed Forward Neural Network with multilayer model. To implement Feed Forward Neural Network with multiple layers I follow the back-propagation algorithm for learning in multilayer networks (figure-18.24, Artificial Intelligence A Modern Approach). Sudo code for Feed Forward Neural Network with multiple layers looks like this.

Feed\_Foward\_Neural\_Network\_MultiLayers(X, Y, layers\_dims, learning\_rate = 0.0075, num\_iterations = 5000):

parameters = initialize\_parameters\_multilayer(layers\_dims)

repeat until number of iterations exceed:

# Forward propagation: [LINEAR -> RELU]\*(L-1) -> LINEAR -> SIGMOID.

AL, caches = linear\_multilayer\_forward(X, parameters)

cost = compute\_cost\_cifar-10(AL, Y)

# Backward propagation: LINEAR -> SIGMOID -> [LINEAR -> RELU]\*(L-1).

grads = linear\_multilayer\_backward(AL, Y, caches)

parameters = update\_parameters(parameters, grads, learning\_rate)

return parameters

Initially, I initialize all parameter randomly using a standard normal distribution (mean 0 and variance 1). Also, divide this random number on a particular layer by the square root of the neuron number of that hidden layer.

For activation function, I used Sigmoid and Relu. At Feed Forward Neural Network with multiple layers(L), I used Relu activation function in first L-1 layers and Sigmoid activation function in Lth layer both in forward and backward propagation. At Feed Forward Neural Network with two layers, I used Relu activation function in first layer and Sigmoid activation function in second layer both in forward and backward propagation. Similarly, at Feed Forward Neural Network with one layers, I only used Sigmoid activation function both in forward and backward propagation. I also use cross entropy loss to compute cost in a particular iteration.

Equation of forward propagation : .

Cross-entropy cost : .

Equation of backward propagation : , , .

Update Parameter : ,

I tested all my implemented Feed Forward Neural Network on CIFAR-10 and MNIST dataset. CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes(airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6000 images per class. There are 50000 training images and 10000 test images. MNIST dataset contains handwritten digits(0-9) consists of a training set of 60,000 examples, and a test set of 10,000 examples. Because of running all of implemented models on whole dataset taking more than hours, I select a portion of data from both dataset. Form CIFAR-10 dataset, I take 2500 example data for training and 250 example data for testing. Form MNIST dataset , I take first 2700 example data for training and next 300 example data for testing.

In terms of evaluation performance, I use simple accuracy matrices which simply calculate the percentage of correctly classified examples form total examples. I also checked my implemented Feed Forward Neural Network with one layer(no hidden layer) and Feed Forward Neural Network with two layers(one hidden layer) result with Feed Forward Neural Network with multiple layers, just input it no hidden layers or one hidden layers.

Performance (learning rate = 0.0075, iteration = 2500 ) of Feed Forward Neural Network (one, two, multilayer) on CIFAR-10 dataset given below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm name | Layers dimension | Training data shape | Teat data shape | Training Accuracy | Test Accuracy |
| Feed Forward Neural Network with one layer | [3072, 10] | 3072, 2500 | 3072, 250 | 49.64% | 35.2% |
| Feed Forward Neural Network with two layers | [3072, 15, 10] | 3072, 2500 | 3072, 250 | 48.24% | 31.6% |
| Feed Forward Neural Network with two layers | [3072, 20, 10] | 3072, 2500 | 3072, 250 | 47.6% | 35.6% |
| Feed Forward Neural Network with two layers | [3072, 7, 10] | 3072, 2500 | 3072, 250 | 36.4% | 28.4% |
| Feed Forward Neural Network with multiple layers(3 layers) | [3072, 17, 12, 10] | 3072, 2500 | 3072, 250 | 43.4% | 32.8% |
| Feed Forward Neural Network with multiple layers(4 layers) | [3072, 20, 17, 12, 10] | 3072, 2500 | 3072, 250 | 38.72% | 28.0% |
| Feed Forward Neural Network with multiple layers(5 layers) | [3072, 40, 20, 17, 12, 10] | 3072, 2500 | 3072, 250 | 34.68% | 32.4% |

Performance (learning rate = 0.0075, iteration = 2500 ) of Feed Forward Neural Network (one, two, multilayer) on MNIST dataset given below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm name | Layers dimension | Training data shape | Teat data shape | Training Accuracy | Test Accuracy |
| Feed Forward Neural Network with one layer | [784, 10] | 784, 2700 | 784, 300 | 87.66% | 85.66% |
| Feed Forward Neural Network with two layers | [784, 15, 10] | 784, 2700 | 784, 300 | 89.25% | 86.0% |
| Feed Forward Neural Network with two layers | [784, 20, 10] | 784, 2700 | 784, 300 | 89.81% | 86.33% |
| Feed Forward Neural Network with two layers | [784, 7, 10] | 784, 2700 | 784, 300 | 82.85% | 79.67% |
| Feed Forward Neural Network with multiple layers(3 layers) | [784, 17, 12, 10] | 784, 2700 | 784, 300 | 88.88% | 82.0% |
| Feed Forward Neural Network with multiple layers(4 layers) | [784, 20, 17, 12, 10] | 784, 2700 | 784, 300 | 88.30% | 83.67% |
| Feed Forward Neural Network with multiple layers(5 layers) | [784, 40, 20, 17, 12, 10] | 784, 2700 | 784, 300 | 87.11% | 85.0% |

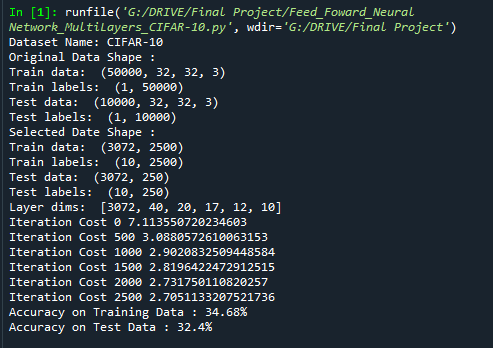
I have faced many challenges, at the time implementing Feed Forward Neural Network. For instance, I have found cost and parameters values became ‘nan’ after some iteration. The problem behind this is I mistakenly fit the output layers to one which should be the equal of number of classes. Most of the examples, I found book or online on binary classification, but my dataset needs multiclass classification. It took some days to understand the notion multiclass classification in terms of one vs all prediction. I also faced challenges to choose learning rate and number of iterations as I only implement very basic Feed Forward Neural Network model. Also, I need to read a lot to find out appropriate loss function (cross entropy) for this multiclass classification. Interestingly, I found random initialization of parameters make the parameter values very close which has also negative effect on model performance, to decimate the effect I divide initial random initialization values by of number of neuron on that layer.

At the time of implementing Feed Forward Neural Network I have learned many things. At first stage mistakenly, I put one neuron at my output layer (which should be the number of classes for multiclass prediction). Then I found after 5-10 iteration cost and value of parameter became ‘nan’. I think it’s the robustness of the neural network architecture that signals that my implemented model has major logical error. In our textbook, back-propagation algorithm for learning in multilayer networks (figure-18.24) uses L2 loss which mainly works for regression model. As I am working developing model for classification problem, I found cross entropy loss need to use on that scenario. My implemented Feed Forward Neural Network gives better performance on MNIST dataset compare to CIFAR-10. I think the actual reason behind this is simple dimension of MNIST image (28,28) than CIFAR-10 image (32,32,3) which(MNIST) is easily identifiable with my basic Feed Forward Neural Network model. I also test my Feed Forward Neural Network model with multiple layers by one , two layers in layer dimension. I have found that it gives the same result like my Feed Forward Neural Network model with one and two layers. As a result, that validates my neural network implementation. Additionally, my implemented Feed Forward Neural Network with one layer gives good accuracy which I implement only from the curiosity to see the performance.

Find out correct approximation function for a particular dataset is always an open problem and now-a-days expressivity or interpretability of a model also facilitate model applicability. Result of my Feed Forward Neural Network on MNIST dataset is quite good than CIFAR-10 dataset, even though it’s a very simple model. As I have mentioned, due to timing issue I am unable run my algorithm on whole dataset. After reading our textbook, I think stochastic gradient descent will solve this issue. As my implemented Feed Forward Neural Network is very basic model which can be improved by some techniques such as regularization, scaling the input, changing number of hidden units and layers and so on.

**How to run code:** I implemented Feed Forward Neural Network model for one, two and multiple layers. Also run these on CIFAR-10 and MNIST dataset. So, in total 6 python files( ‘Feed\_Foward\_Neural Network\_MultiLayers\_CIFAR-10.py’,’Feed\_Foward\_NeuralNetwork\_MultiLayers\_MNIST.py’,’ Feed\_Foward\_NeuralNetwork\_OneLayers\_CIFAR10’,’Feed\_Foward\_NeuralNetwork\_OneLayers\_MNIST’,’Feed\_Foward\_NeuralNetwork\_TwoLayers\_CIFAR10’,’Feed\_Foward\_NeuralNetwork\_TwoLayers\_MNIST’). Any of these files can be easily run by any python IDE (I use Spyder). Before running number of iterations and learning rates can be changed. Also, number of training and test data can be changed form main function.

For example, after running ‘Feed\_Foward\_Neural Network\_MultiLayers\_CIFAR-10.py’, I get following output.



Writeup on the Research Paper

Neural Additive Models: Interpretable Machine Learning with Neural Nets:

Deep neural networks(DNN) provides powerful approximation of function that have achieved impressive results on a variety of tasks, such as computer vision natural language processing and reinforcement learning. But it is very difficult to understand what happens inside any DNN, specially how do they make any predictions. That’s why they are often considered as black-box model which limits their applicability to high-stakes domains such as healthcare and criminal justice.

In this paper, author present Neural Additive Models (NAMs), a new class of models which combines the intelligibility of Generalized Additive Models(GAMs) with the expressivity of DNNs. In NAMs architecture, each feature mapped to a neural network and it learns from the combination of neural networks. That’s actually augment the expressivity of neural nets as easy to deduce which input feature contribute to prediction. Their experiments on regression and classification datasets show that NAMs are more accurate than logistic regression and shallow decision trees. NAM has some interesting properties than GAM. For example, NAM can be combined with other deep learning methods which augment the intelligibility of deep learning models. At the same time, the graph learned by NAM not only just explanation, but also an exact description of how NAMs make a prediction. Additionally, because of having flexible structure, NAM can be extended to any flexible settings. Furthermore, NAMs can be trained on GPU as they are scalable. Finally, GAMs require millions of decision trees to fit each shape function, whereas NAMs only use a small ensemble (10 - 100) of neural nets.

Interpreting NAM is easy as the impact of a feature on the prediction does not rely on the other features. But that opens up a question what happens if the impact of a feature has dependency on the other features. I think in future, it’s an open question to explore which also enrich the interpretable machine learning domain.

Hierarchical Interpretations for Neural Network Predictions:

Deep neural networks (DNNs) have recently demonstrated impressive predictive performance due to their ability to learn complex, non-linear, relationships between variables. However, the inability to effectively visualize these relationships has led DNNs to be characterized as black boxes.

In this work, they introduce agglomerative contextual decomposition (ACD), a novel hierarchical interpretation algorithm. ACD is the first method to use a hierarchy to interpret individual neural network predictions. Doing so enables ACD to automatically detect and display non-linear contributions to individual DNN predictions, something prior interpretation methods are unable to do. The benefits of capturing the non-linearities inherent in DNNs are demonstrated through human experiments and examples of diagnosing incorrect predictions and dataset bias. Importantly, they also demonstrate that ACD’s hierarchy is robust to adversarial perturbations in CNNs, implying that it captures fundamental aspects of the input and ignores spurious noise.

This paper also opens up the opportunities to see the interpretable machine learning in terms of hierarchy.