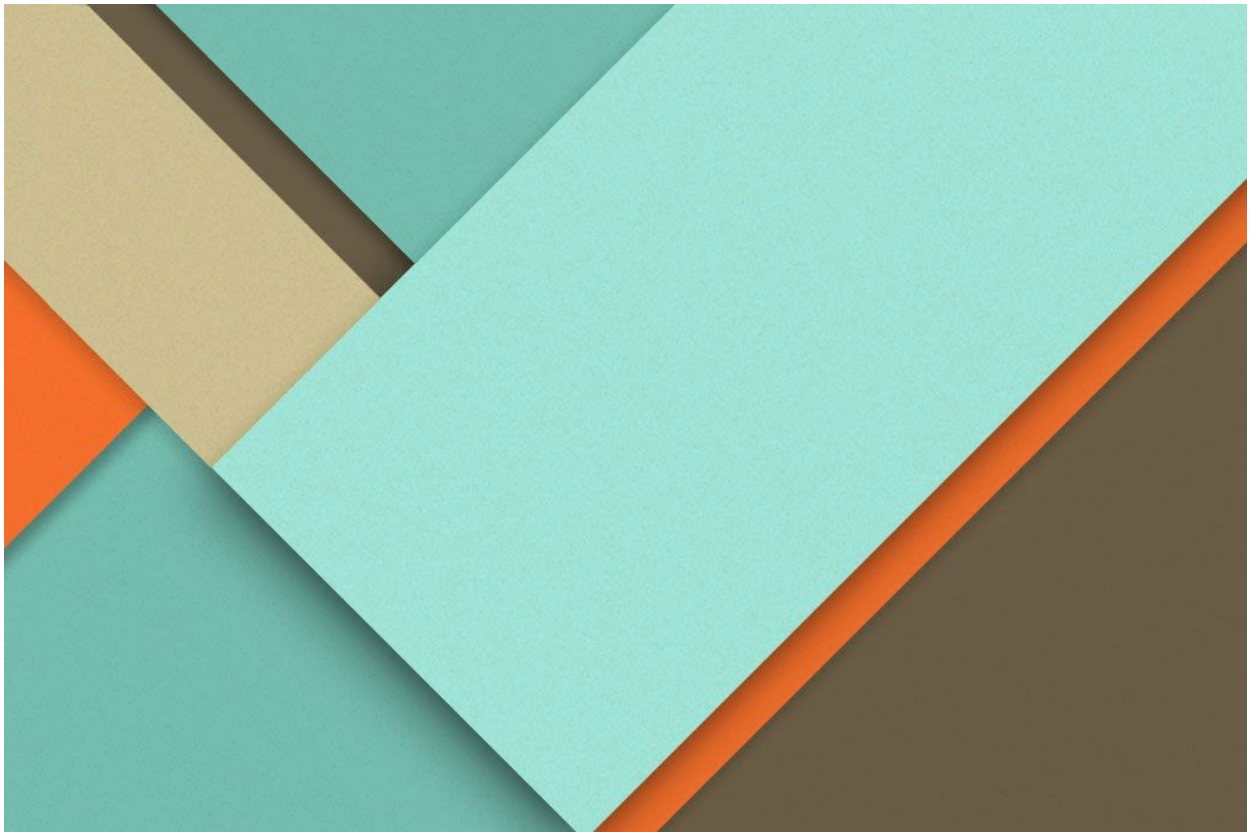

FizzBuzz

(using a feed-forward neural network)

Bazil Ahmed
Mtech AI
Sr.No: 16677
Deep Learning

Overview

In this project, I have trained a feed-forward neural network on Integers [101,1000] to classify them in 4 classes namely: (1). fizz (2). buzz (3) fizzbuzz (4) remaining if divisible by 3,5,15(both 3 and 5), else respectively.

Goals

1. Choose suitable hyperparameters for NN such that it replicates the probability distribution of the training data by an estimated model(minimizing the loss function).
2. The model should not overfit the training data and should be generalizable within a suitable accuracy measure on the test data.

Specifications of the final model (Software 2.0)

Number of hidden layers: 1

Number of neuron units: 100

Loss function (objective function): cross-entropy

Optimization technique: Adam

Learning rate: 0.005

Batch size for optimization step: 128

Activation function: RELU

Input layer width: 10

Input feature: 10-bit binary features (binary representation of the input)

Output layer width: 4

Output bias weights: [0.2,2,1,0.6]

Output mapping(argMax): 0->None, 1->fizzbuzz, 2->buzz, 3->fizz

Epochs: 5000

Accuracy on test_data: 94%

Precautions used:

1. Training data were randomly permuted to prevent it from any biasing because of batch-wise optimization.
2. While training the model, test data was completely kept aside and only focus was the minimization of the loss on training data.
3. With the same hyperparameters, different accuracy was observed because of the different shuffling of the input data. Hence several runs were given for the training of the model to get an average understanding of the accuracy.
4. Due to the uneven number of training inputs for individual classes, biasing weights were attached in the loss function for different weightage to the activation value of the output.

Libraries used in main.py:

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pickle
import getopt, sys
from torch.autograd import Variable
```

Experiments and conclusions:

Effect of increasing learning rate:

On increasing the learning rate, the reduction of the objective function per epoch was better up to a certain limit. After that limit(around 0.04) is reached there was no significant change in the loss value from the initial point. And hence resulted in bad accuracy in the training as well as the test set.

Effect of batch size for optimization:

Reducing the batch size resulted in high fluctuations of the loss value and most of the time not converging. However, increasing the batch size was time-consuming for even small convergence.

Effect of the number of neuron units in the hidden layer:

Increasing the number of neuron units certainly increased the accuracy on the test data. Initially, I used a single layer of 6 units which resulted in 76% accuracy which got increased to 94% with 100 neuron units.

Effect of the number of hidden layers:

I trained 2 hidden layers each of size 12 units the resulting accuracy was bad (80%). Then I shifted to a single hidden layer with increased units, which in turn increased the accuracy.

What to be expected:

Run the below command in the main.py containing directory.

python main.py --test-data <test file>

It will generate two files namely **Software1.txt** and **Software2.txt**. Where software1.txt is the result of the logic-based program (100% accuracy) and software2.txt is the result of the neural net implementation of the program. Software2 named NN model gets picked up from the Model directory automatically by main.py.

NN_create.py has the code for the training of the model.