A PROJECT ON
HAND WRITTEN DIGIT RECOGNITION USING TENSORFLOW AND PYTHON

UNDER THE GUIDANCE OF PROF. MEILIU LU

BY, SHEKHAR SHIROOR
INDEX

• MOTIVATION
• INTRODUCTION
• INPUT DATA
• APPROACH
  1. WHY DNN ?
  2. VARIOUS DNN MODELS I USED
• OUTPUT AND ACCURACY
• VISUALIZATION
• FUTURE SCOPE
• CONCLUSION
• REFERENCES
MOTIVATION

• Before computers existed all the information was stored in written form, this is very inefficient form of storage as the paper information cannot be stored for very long time and can get lost or be destroyed.

• On the contrary information on computer is stored safely for long time and multiple copies of same information can be made easily.

• Thus after inventing the computers lot of money was wasted in manual labor for converting this paper information into digital information.

• Instead machine learning can be used to identify and convert this paper information into digital information without human intervention or manual labor.

• My project is just an introduction to this approach.
INTRODUCTION

• WHAT IS TENSORFLOW?
  • IT IS A MACHINE LEARNING LIBRARY INTRODUCED BY GOOGLE

• WHAT IS MNIST DATA SET?
  • IT IS A SUBSET OF NIST DATA SET WHICH CONTAINS (A-Z), (A-Z), (0-9) ALL BLACK AND WHITE IMAGES
  • IT IS DATASET OF HANDWRITTEN DIGITS (0-9)
  • IT CONSIST OF 55000 TRAINING DATA AND 10000 TEST DATA

• DATA PREPROCESSING OF MNIST
  • INITIALLY THE IMAGES WERE NORMALIZED TO 20X20 PIXELS AND CENTERED, ALSO THEIR ASPECT RATIO WAS MAINTAINED
  • AFTER THIS STEP THEY WERE RESIZED TO 28X28 PIXELS IN ORDER TO GET BETTER ACCURACY AND DISTINCTION BETWEEN THE BLACK AND WHITE PIXELS IN THE IMAGE
INPUT DATA
The input to the model is the pixel data shown and the array of values shown above.

These array values are the numbers each image represents.

The numbers represented by the array are stored in the form of 'one hot encoding' i.e. the position in the array where one is present represents the number present in the image.
APPROACH

• MY APPROACH DEEP NEURAL NETWORK (DNN) WHAT IS IT ??
  • A PERCEPTRON IS A NODE WHICH TAKES INPUT PROCESSES IT AND GIVES SINGLE OUTPUT
  • SINGLE LAYER OF PERCEPTRON IS A NEURAL NETWORK.
  • MULTIPLE LAYERS OF PERCEPTRON(>=2) IS CALLED A DEEP NEURAL NETWORK.

• WHY ??
  • GIVES BETTER ACCURACY COMPARED OTHER ALGORITHMS LIKE LINEAR REGRESSION
  • SOME OF THE ML ALGORITHMS LIKE DECISION TREES CANNOT GIVE HIGH ACCURACY ON
    MNIST DATA DUE TO IT'S LIMITATION OF PROCESSING HIGH DIMENSIONAL DATA
APPROACH (CONTINUED....)

• DNN MODELS I USED:
  • INITIALLY I USED 3-HIDDEN LAYER EACH CONTAINING 500 NODES AND RAN FOR 10 EPOCHS
  • THEN I USED THE SAME MODEL AND RAN FOR 15 EPOCHS
  • THEN I USED 4-HIDDEN LAYERS EACH CONTAINING 500,1500,1500,500 NODES RESPECTIVELY FOR 10 EPOCHS
  • THEN I USED THE SAME MODEL AND RAN FOR 20 EPOCHS
  • MY FINAL MODEL IS THE SAME 4-HIDDEN LAYER MODEL WHICH I RAN ON 15 EPOCHS
CONTINUED...

```python
# Define the neural network function
def neural_network_m(data, name='DNN'):
    hidden_layer_1 = {'weights': tf.Variable(tf.random_normal([784, nodes_hl1], name='Weights_L-1'), 'biases': tf.Variable(tf.random_normal([nodes_hl1], name='Biases_L-1'))}
    hidden_layer_2 = {'weights': tf.Variable(tf.random_normal([nodes_hl1, nodes_hl2], name='Weights_L-2'), 'biases': tf.Variable(tf.random_normal([nodes_hl2], name='Biases_L-2'))}
    hidden_layer_3 = {'weights': tf.Variable(tf.random_normal([nodes_hl2, nodes_hl3], name='Weights_L-3'), 'biases': tf.Variable(tf.random_normal([nodes_hl3], name='Biases_L-3'))}
    hidden_layer_4 = {'weights': tf.Variable(tf.random_normal([nodes_hl3, nodes_hl4], name='Weights_L-4'), 'biases': tf.Variable(tf.random_normal([nodes_hl4], name='Biases_L-4'))}
    output_layer = {'weights': tf.Variable(tf.random_normal([nodes_hl4, classes], name='Weights_to_output_L'), 'biases': tf.Variable(tf.random_normal([classes], name='Biases_to_output_L'))}
    l1 = tf.add(tf.matmul(data, hidden_layer_1['weights']), hidden_layer_1['biases'])
    l1 = tf.nn.relu(l1, name='Activation_1')
    l2 = tf.add(tf.matmul(l1, hidden_layer_2['weights']), hidden_layer_2['biases'])
    l2 = tf.nn.relu(l2, name='Activation_1')
    l3 = tf.add(tf.matmul(l2, hidden_layer_3['weights']), hidden_layer_3['biases'])
    l3 = tf.nn.relu(l3, name='Activation_1')
    l4 = tf.add(tf.matmul(l3, hidden_layer_4['weights']), hidden_layer_4['biases'])
    output = tf.matmul(l4, output_layer['weights']) + output_layer['biases']
```
CONTINUED....

• SOME TERMS :-
  • RELU
     • RECTIFIED LINEAR UNIT : GIVES SMOOTH APPROXIMATION \[ f(x) = \ln(1 + e^x) \]

• CROSS ENTROPY LOSS :-
  • FUNCTION USED TO MEASURE ERROR AT SOFTMAX LAYER

• SOFTMAX :-
  • ACTIVATION FUNCTION/LAYER WHICH INTERPRETS OUTPUT AS PROBABILITIES

• ADAM OPTIMIZER :-
  • OPTIMIZER SIMILAR TO GRADIENT DESCENT OPTIMIZER BUT GIVES MUCH EFFICIENT RESULTS FOR AN EPOCH (FEED FORWARD AND BACKPROPAGATE )
OUTPUT AND ACCURACY

```
ese are available on your machine and could speed up CPU computations.
I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:910] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero
I tensorflow/core/common_runtime/gpu/gpu_device.cc:885] Found device 0 with properties:
name: GeForce GTX 950M
major: 5 minor: 0 memoryClockRate (GHz) 0.928
pciBusID 0000:01:00.0
Total memory: 1.96GiB
Free memory: 1.65GiB
I tensorflow/core/common_runtime/gpu/gpu_device.cc:906] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_device.cc:910] 0: Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:975] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce G
TX 950M, pci bus id: 0000:01:00.0)
('Epoch', 0, 'completed out of', 15, 'loss:', 57076235.641601562)
('Epoch', 1, 'completed out of', 15, 'loss:', 11679495.053405762)
('Epoch', 2, 'completed out of', 15, 'loss:', 5675212.272837162)
('Epoch', 3, 'completed out of', 15, 'loss:', 2969754.5679917908)
('Epoch', 4, 'completed out of', 15, 'loss:', 1959448.0925540924)
('Epoch', 5, 'completed out of', 15, 'loss:', 1307455.6754932404)
('Epoch', 6, 'completed out of', 15, 'loss:', 1770407.0635409355)
('Epoch', 7, 'completed out of', 15, 'loss:', 1201212.2745146751)
('Epoch', 8, 'completed out of', 15, 'loss:', 1186197.5414273739)
('Epoch', 9, 'completed out of', 15, 'loss:', 1145837.9236880824)
('Epoch', 10, 'completed out of', 15, 'loss:', 1286810.4259119034)
('Epoch', 11, 'completed out of', 15, 'loss:', 1065730.1571483612)
('Epoch', 12, 'completed out of', 15, 'loss:', 749443.2568757534)
('Epoch', 13, 'completed out of', 15, 'loss:', 705137.59090614319)
('Epoch', 14, 'completed out of', 15, 'loss:', 853053.65564918518)
('Accuracy', 0.96280015)
[ 0.66666667 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314 0.66784314]```
• THE ACCURACY FOR MY FINAL MODEL AS PER THE PREVIOUS PICTURE IS 96.28%

• ALSO THE OUTPUT OBTAINED FOR TEST DATA IS IN THE FORM OF 'ONE HOT ENCODING'
VISUALIZATIONS

- SCALARS
- GRAPHS
- DISTRIBUTIONS
- HISTOGRAMS
- EMBEDDING VISUALIZER (PCA AND TSNE)
FUTURE SCOPE

• EXTEND THE MODEL TO WORK ON NIST DATASET
• INCREASE THE ACCURACY FURTHER BY IMPLEMENTING MORE NUMBER OF HIDDEN LAYERS AND/OR EPOCHS
• DETECT CUSTOM HAND WRITTEN DIGITS
• USE CNN WITH LESS LAYERS TO GET BETTER ACCURACY
DISCUSSION

• RANDOM FOREST GIVES AN ACCURACY OF 0.8 APPROX FOR THE MNIST DATA SET

• KNN ALGORITHM GIVES AN ACCURACY OF 0.94 APPROX FOR THE MNIST DATA SET

• DNN(MY MODEL) GIVES AN ACCURACY OF 0.962 APPROX FOR THE MNIST DATA SET

• CNN CAN BE USED TO ACHIEVE AN ACCURACY OF 0.992 APPROX FOR THE MNIST DATA SET

• THUS NEURAL NETWORKS CAN GIVE BETTER AND MORE ACCURATE RESULTS FOR THIS PARTICULAR PROBLEM COMPARED TO OTHER MACHINE LEARNING ALGORITHMS
REFERENCES


- HTTPS://WWW.YOUTUBE.COM/WATCH?V=GJ0IYO265BC

- HTTP://YANN.LECUN.COM/EXDB/MNIST/

- HTTPS://WWW.TENSORFLOW.ORG/

- THE MNIST DATABASE OF HANDWRITTEN DIGIT IMAGES FOR MACHINE LEARNING RESEARCH [BEST OF THE WEB]

- HTTPS://EN.WIKIPEDIA.ORG/WIKI/RECTIFIER_(NEURAL_NETWORKS)
QUESTIONS ???

THANK YOU