MNIST Hand Digit Recognition: Using Keras: 3 and 5 layer FFNN

Satya krishna Saiteja.Pillarisetti

AITS INTERSHIP 2019 - BATCH NO-4

AI-Tech Systems

pksaiteja1@gmail.com

www.ai-techsystems.com

1. Abstract

Using MNIST hand written digit data we identifying the number's pattern recognition in this paper we see how we train and predict in test data using hand written number pattern recognition data using deep learning neural networks

2. Introduction to neural networks

What is a Neural Network?

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

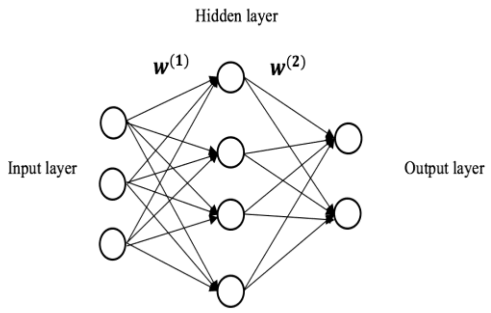


Fig 2.1 Structure of simple neural network or single neural network

3. Introduction to Deep learning or Deep neural network

What does Deep Neural Network mean?

A deep neural network is a neural network with a certain level of complexity, a neural network with more than two layers. Deep neural networks use sophisticated mathematical modeling to process data in complex ways.

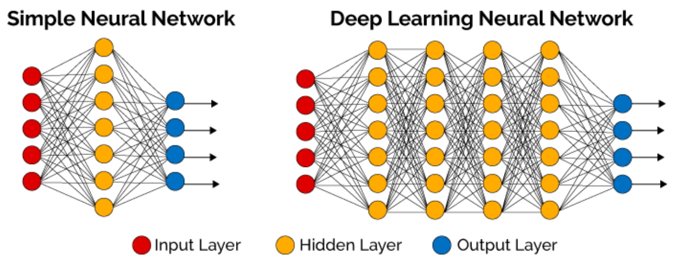


Fig 3.1 Deep neural network structure

4. Activation function

What are Activation Functions?

Activation functions are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable. They introduce non-linear properties to our Network. Their main purpose is to convert a input signal of a node in a A-NN to an output signal. That output signal now is used as an input in the next layer in the stack.

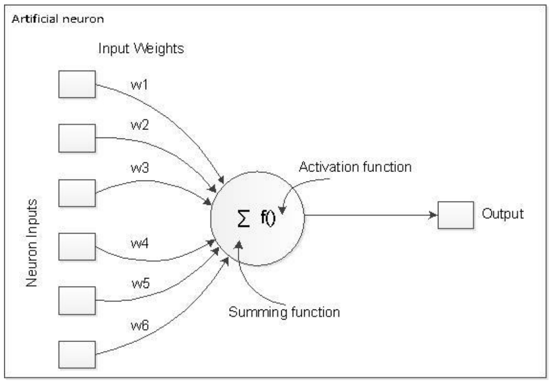


Fig 4.1 states what's the activity function

There are different activity functions to use for different type of data consider below figure

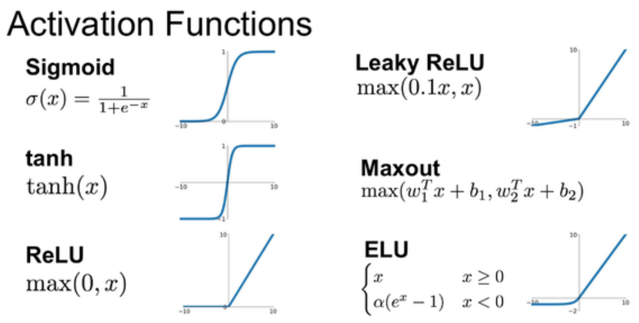


Fig 4.2 Different Activation functions

5. **Weight initializers in Keras**

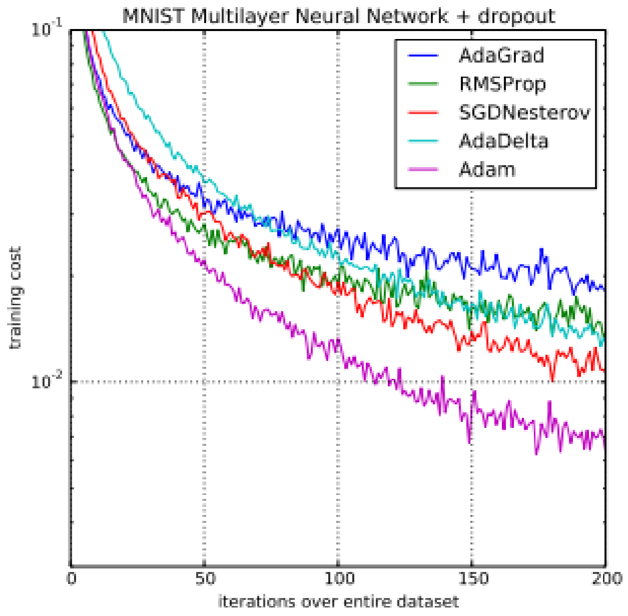
The aim of weight initialization is to prevent layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. If either occurs, loss gradients will either be too large or too small to flow backwards beneficially, and the network will take longer to converge, if it is even able to do so at all, There are different weight initializations depends upon data we use different weight initializations Refer this link and read the documentation for different weight initializers

Refer: https://keras.io/initializers/

6. **optimizers**

Deep Learning, to a large extent, is really about solving massive nasty optimization problems. A Neural Network is merely a very complicated function, consisting of millions of parameters that represents a mathematical solution to a problem, by training neural networks, we essentially mean we are minimizing a loss function. The value of this loss function gives us a measure how far from perfect is the performance of our network on a given dataset, there are many optimizers which is updated from one by one every optimizers have some drawbacks so one of the most used optimizer is **ADAM** which avoids all the drawbacks by other optimizers like **SGD**, **SGD-Momentum**, **RMS-Prop**, **AdaGrad, Ada-delta**.

Below we see that out of all optimizers **ADAM** will gives **low-loss** compare to **ADAGRAD**, **RMSPROP**, **SGD** and **ADADELTA**.



` Fig 6.1 Different optimizer's performance

7. **Batch-Normalization**

Batch normalization is a technique for improving the performance and stability of neural networks, and also makes more sophisticated deep learning architectures,

The idea is to normalize the inputs of each layer in such a way that they have a mean output activation of zero and standard deviation of one. This is analogous to how the inputs to networks are standardized,

How does this help? We know that normalizing the inputs to a network helps it learn. But a network is just a series of layers, where the output of one layer becomes the input to the next. That means we can think of any layer in a neural network as the first layer of a smaller subsequent network.

#code for batch normalization

```
from keras.layers.normalization import
BatchNormalization
model = Sequential()
# think of this as the input layer
model.add(Dense(64, input_dim=16, init='uniform'))
model.add(BatchNormalization())--→ B.N
model.add(Activation('tanh'))
```

Note: Suppose if we miss batch normalization it leads to variance and mean shift for every iteration it gives bad performance of model

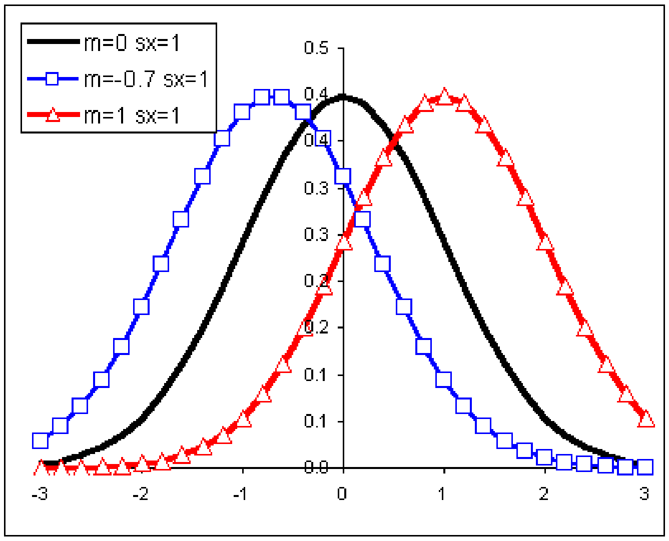


Fig 7.1 Different mean and variance shifting

8. Dropout

The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By "ignoring", I mean these units are not considered during a particular forward or backward pass.

More technically, at each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

Below is the figure for understanding the structure of dropout in deep neural networks

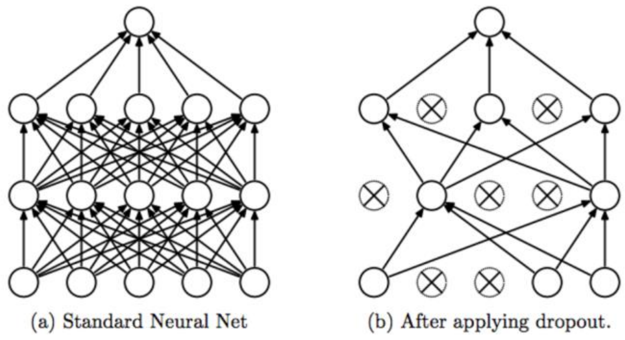


Fig 8.1 Dropout

9. BatchNormalization and Dropout Performance

Below we see Train loss in MNIST data by using Batch normalization and dropout

With Batch Normalization + Dropout

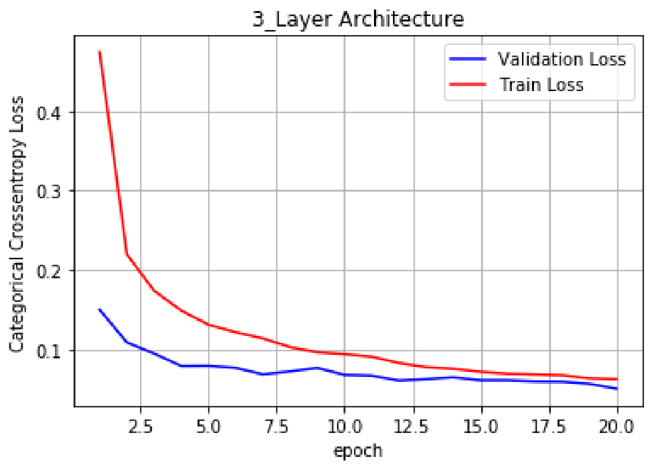
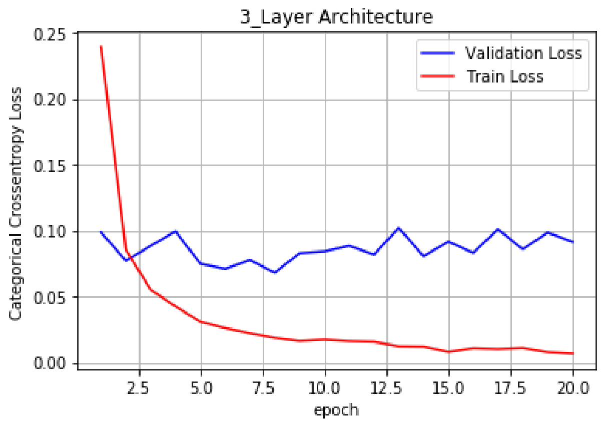


Fig 9.1 we see above Train loss is good by using batch normalization and dropout rate

We also see without using batch normalization and dropout

With-out Batch normalization + Dropout



In above fig 9.2 we see that with-out using batch normalization and dropout we got more test loss than train it seems bad performance so using batch normalization and dropout at every layer gives good model performance

10. Loss Functions

Loss function is an important part in artificial neural networks, which is used to measure the inconsistency between predicted value (^y) and actual label (y). It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function. Loss function is the hard core of empirical risk function as well as a significant component of structural risk function, so there are many loss functions for our data we use multi loss because we have multi class for that we use Cross entropy.

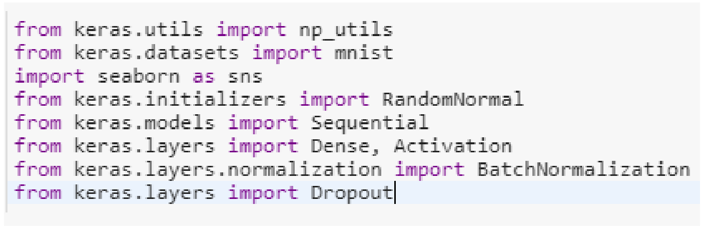
Cross Entropy is commonly-used in binary classification (labels are assumed to take values 0 or 1) as a loss function (For multi-classification, use Multi-class Cross Entropy), which is computed by below equation

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Cross entropy measures the divergence between two probability distribution, if the cross entropy is large, which means that the difference between two distribution is large, while if the cross entropy is small, Which means that two distribution is similar to each other.

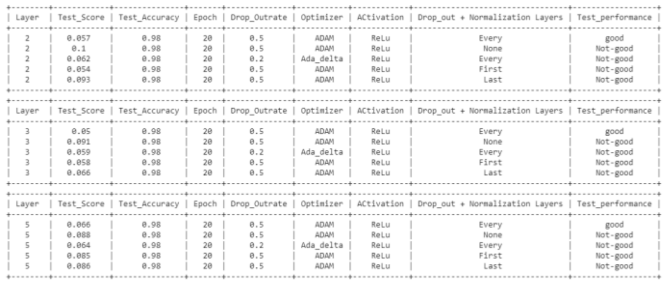
11. Libraries used in Keras

Below are the some libraries we used in Keras for our project



```python
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
```

12. Test Evaluation



```
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
| Layer | Test_Score | Test_Accuracy | Epoch | Drop_Outrate | Optimizer | ACtivation | Drop_out + Normalization Layers | Test_performance |
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
|   2   |   0.057    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |               Every               |       good       |
|   2   |    0.1     |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |                None               |     Not-good     |
|   2   |   0.062    |      0.98     |   20  |     0.2      | Ada_delta |    ReLu    |               Every               |     Not-good     |
|   2   |   0.054    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |               First               |     Not-good     |
|   2   |   0.093    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |                Last               |     Not-good     |
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
| Layer | Test_Score | Test_Accuracy | Epoch | Drop_Outrate | Optimizer | ACtivation | Drop_out + Normalization Layers | Test_performance |
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
|   3   |    0.05    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |               Every               |       good       |
|   3   |   0.091    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |                None               |     Not-good     |
|   3   |   0.059    |      0.98     |   20  |     0.2      | Ada_delta |    ReLu    |               Every               |     Not-good     |
|   3   |   0.058    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |               First               |     Not-good     |
|   3   |   0.066    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |                Last               |     Not-good     |
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
| Layer | Test_Score | Test_Accuracy | Epoch | Drop_Outrate | Optimizer | ACtivation | Drop_out + Normalization Layers | Test_performance |
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
|   5   |   0.066    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |               Every               |       good       |
|   5   |   0.088    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |                None               |     Not-good     |
|   5   |   0.064    |      0.98     |   20  |     0.2      | Ada_delta |    ReLu    |               Every               |     Not-good     |
|   5   |   0.085    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |               First               |     Not-good     |
|   5   |   0.086    |      0.98     |   20  |     0.5      |    ADAM   |    ReLu    |                Last               |     Not-good     |
+-------+------------+---------------+-------+--------------+-----------+------------+-----------------------------------+------------------+
```

Above is test evaluation results for all 2, 3, 5 layer architecture models with different type of models.

13. References

https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78

https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3

https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5