CSE 574: Introduction to Machine Learning (Fall 2018)
Instructor: Sargur N. Srihari

Project 1.1: Software 1.0 vs Software 2.0

September 17, 2018

Report By:
Siddheswar Chandrasekhar

Objective

The project is to compare two problem solving approaches to software development: the logic-based approach (Software 1.0) and the machine learning approach (Software 2.0). It is also designed to quickly gain familiarity with Python and machine learning frameworks.

Software 1.0

Software 1.0 is a simple Python program to find if a number is divisible by 3 or 5. If the number is divisible by 3 we print Fizz, if it is divisible by 5 we print Buzz, if it is divisible by both 3 and 5 we print FizzBuzz.
This simple program gives us a 100% accuracy as we write the algorithm using which the machine finds out which label / class (Fizz, Buzz, FizzBuzz or Other) a particular number belongs to.

Software 2.0

Software 2.0 is a Machine Learning problem wherein we have a training and a testing dataset. Given our training dataset, the model finds relationship between different inputs and its outputs and we then check to see how accurate predictions are made by the model on our testing dataset.

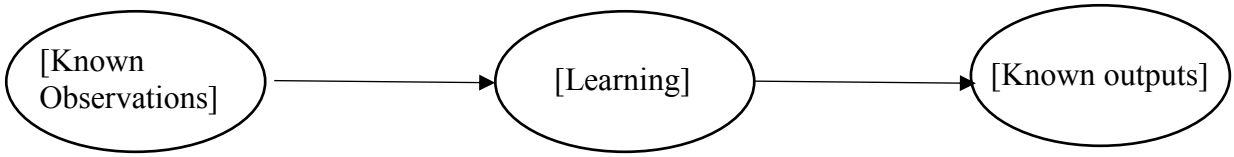
The concept of Machine Learning gives room for several questions, a few of which are answered below.

What is Machine Learning?

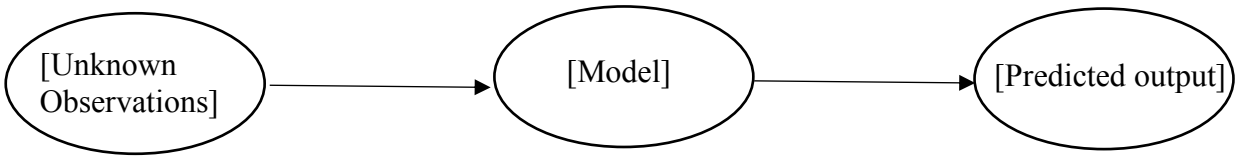
"Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions." [3]

Machine Learning is categorized into two types. Supervised Learning and Unsupervised Learning. For the scope of this project, we talk only about Supervised Learning.

The Machine Learning process is broadly categorized into two phases, namely the training phase and the testing phase.
In the training phase, we provide the 'machine' a set of inputs and its calculated outputs so that the 'machine' can infer the relationship between the inputs and the outputs.



We then provide the 'machine' with a set of inputs that haven't been seen before and based on the deductions made during the testing phase, outputs are predicted for the testing phase.

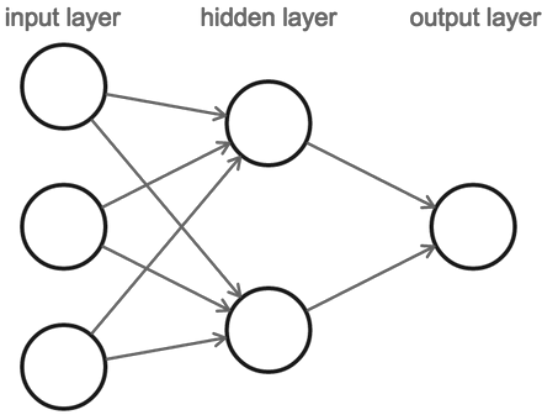


What is a Neuron?

A neuron can be thought of as a node that performs some computation. Since neurons lack the capability to perform large computations, a set of multiple neurons form the layers in a neural network.

What are Layers?

A neural network has a simple architecture that consists of an input layer, a black box which is a set of hidden layers, and an output layer. All the neurons in the network, connect with every neuron in its adjacent layer forming a mesh-like architecture.
The figure below is a good depiction of a simple neural network with three input neurons, one hidden layer of two neurons, and a single output neuron.



What is the purpose of Hidden Layer?

The hidden layers' job is to transform the input into something the output layer could use. It does so with the help of an Activation Function.

What is an Activation Function?

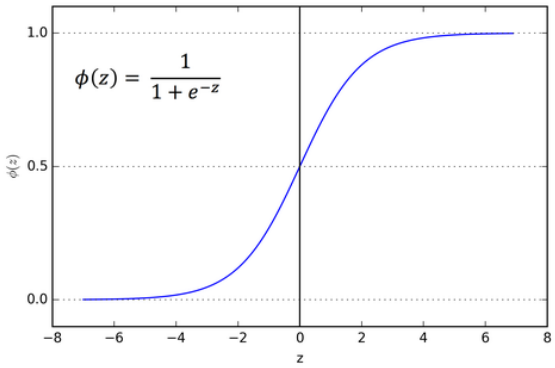
In both artificial and biological neural networks, a neuron does not just output the bare input it receives. Instead, there is one more step, called an activation function, analogous to the rate of action potential firing in the brain. [1]
The activation function takes the weighted sum input, **z=b+∑iwixi**, and then transforms it once more before finally outputting it.

The two academically most common activation functions are sigmoid and ReLU

A sigmoid is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

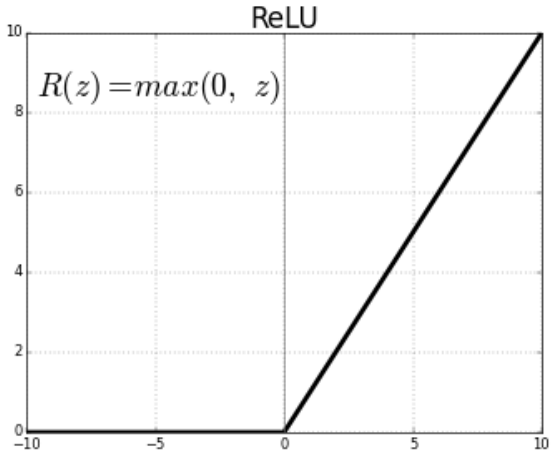
A sigmoid function has the shape:



As can be seen, a sigmoid function exists between [0 to 1] and since the probability of anything ranges between 0 to 1, sigmoid used to be a very popular choice of activation function.
However, in recent years, they have fallen out of favor. This is mainly because sigmoid makes it inefficient to train neural networks that have arbitrarily large number of hidden layers due to the vanishing gradient problem. [1]

The sigmoid function Is continuously derivable [1] and is given as:
$\sigma'(z)=\sigma(z)(1-\sigma(z))$

A ReLU on the other hand, has a shape:



As can be seen, ReLU has a range of [0 to infinity]. In simple words, ReLUs let all positive values pass through unchanged, but sets any negative value to zero. [1]

In tensorflow, the activation functions are defined in the *tf.nn* package (wrappers for neural net operations)

What is an Error Function?

An error function (more known as Loss Function in Machine Learning) is a method of evaluating how well your algorithm models the dataset. Wrong predictions output a higher loss function, whereas right ones output a lower loss function. [4]

What is Cross-entropy?

Cross-entropy is a loss function that describes the loss between two probability distributions.

What is Gradient Descent?

Gradient Descent is an algorithm used to find a good solution much faster than taking random guesses.

How does Gradient Descent work?

First, we start with a random guess at the parameters, and start there. We then figure out which direction the loss function steeps downward the most (with respect to changing the parameters), and step slightly in that direction. To put it another way, we determine the amounts to tweak all of the parameters such that the loss function goes down by the largest amount. We repeat this process over and over until we are satisfied we have found the lowest point. [1]

Why do we distribute the weights normally?

Values in a Gaussian Distribution are closely bound around the mean. Also, three standard deviations around the mean in Gaussian Distribution, gives us 99% of the set. Hence, initializing the weights normally, give us a better accuracy.

Observations

We now take a couple of hyperparameters in our program, and see how changing these affect the accuracy and efficiency of the model.

	Epochs	Batch Size	Avg Iteration Time	Avg Accuracy (%)
1	5000	128	01:50	96
2	2500	128	00:52	86
3	2500	64	01:08	91
4	10000	32	06:15	97
5	1000	32	00:38	91
6	1000	128	00:21	55
7	5000	64	02:16	93
8	5000	48	02:29	91
9	10000	64	04:56	91
10	10000	128	03:46	95
11	5000	256	01:40	93
12	5000	512	01:25	55

NOTE: For the above calculations, we have number of neurons in our hidden layer as 100, Learning Rate as 0.05 and the Activation Function used is ReLU.

We can make the following deductions from the above observation table:

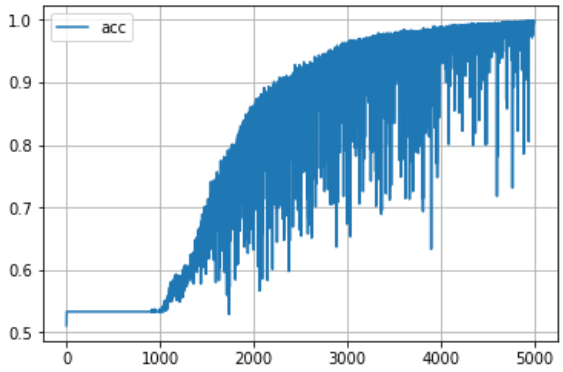
- Higher the batch size, higher the iteration rate per second
- Higher the number of epochs, implies higher the number of iterations to be made, which generally leads to a better accuracy
- Higher the number of neurons in the network, lower the iteration rate per second
- Increasing the number of neurons in the hidden layer increases the accuracy at the cost of lower iterations per second

We now take a few of these observations, make deductions and find the best set of hyperparameters leading to the best accuracy and efficiency.

Observation #1

- With our number of epochs as 5,000 and batch size of 128, we get a weighted average accuracy of about 95 – 96%.

- We get a nice learning curve that peaks at accuracy 1 just inside 5000 epochs



- As can be inferred from the above plot of accuracy to number of epochs, we see significant drops in accuracy throughout the iterations.
  If we want to find the exact accuracy of our model at a particular epoch, we can do so using the *df.loc* function of *pandas.DataFrame* (as our accuracy is stored in a dataframe).
  For example, we find that at epoch 3700, the accuracy of our model drops to about 0.79

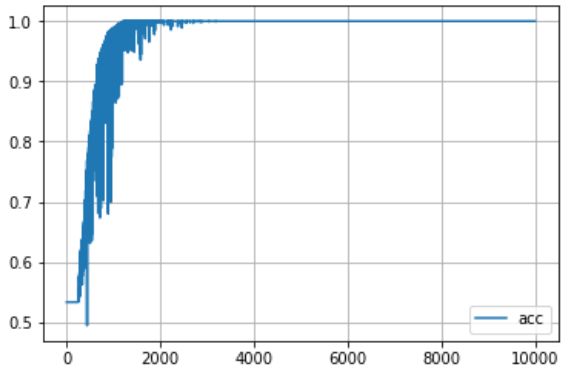
- If we change the activation function to sigmoid, we see that the accuracy drops significantly. Hence we stick with ReLU for now.

- Deduction: This is a good set of hyperparameters as this leads to a model wherein we reach an accuracy of near 1.0 within the number of set epochs

Observation #4

- With our number of epochs as 10,000 and batch size of 32, we get a weighted average accuracy of about 97 – 98%

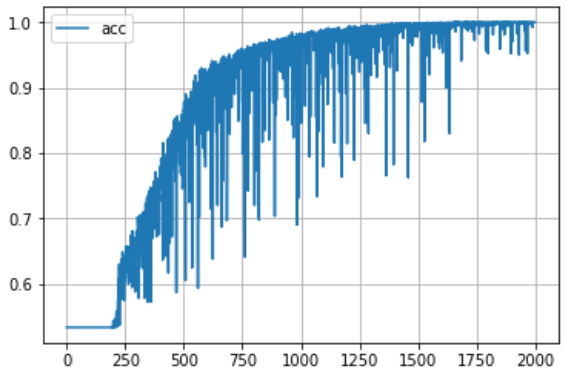
- We get a learning curve that shoots up high early on in the iterations and reaches its constant high at around 2000 epochs.



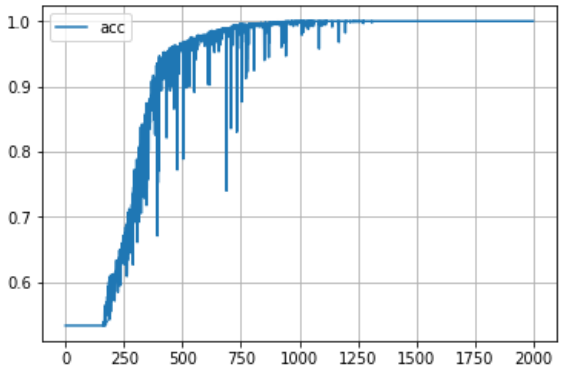
- If we change the activation function to sigmoid, we see that the accuracy drops to about 0.55 Hence we conclude that for this model, we get a better accuracy using ReLU and hence we stick with it.

- Even though the high number of epochs add to the overall iteration time of the model, we get a fairly good accuracy curve that peaks out early on. More importantly, as can be seen in the plot, the accuracy barely ever drops after 2000 epochs. In other words, we get a constant high throughout.

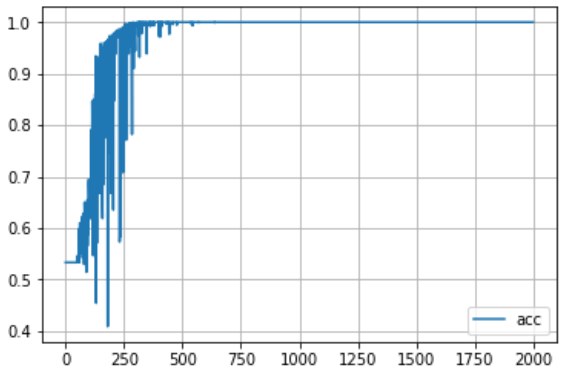
- This essentially also means, that if we change the number of epochs to 2000, whilst keeping all the other hyperparameters unchanged, we would get a similar accuracy of near 1.0 but with far fewer iterations and hence better efficiency. This is shown in the plot:



- We can try and tweak a few more hyperparameters hoping to achieve an even better model. If we increase the number of neurons in the hidden layer to 500, we get a similar accuracy of near 1.0 in even fewer epochs. This is shown in the plot:



- If we now try increasing the Learning Rate of our model to 0.15 we find that we reach accuracy 1.0 in even fewer epochs. This is shown in the plot:



- Deduction: After several more tweaks, including using leaky ReLU and tanh activation functions, we find that this is the best set of hyperparameters that gives us a highly efficient (that reaches near accuracy of 1.0 in about 500 epochs) and up to 100% accurate model.

Conclusion

Our final model gives us up to 100% accuracy and has the following set of hyperparameters:

- Number of neurons in hidden layer = 500
- Learning Rate = 0.15
- Activation Function = ReLU
- Number of Epochs = 750 (a little over 500)
- Batch Size = 32

References:

1. Andreas Refsgaard, Francis Tseng, Gene Kogan (n.d.). Machine Learning for Artists "Neural Networks"
   https://ml4a.github.io/ml4a/neural_networks/

2. Sagar Sharma (2017). Activation Functions: Neural Networks
   https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

3. Daniel Faggella (2018). What is Machine Learning?
   https://www.techemergence.com/what-is-machine-learning/

4. Justin Gage (2018). Introduction to Loss Functions
   https://blog.algorithmia.com/introduction-to-loss-functions/