Managers Report:

Dear Bill,

This program is a Learning machine, simulating a neural network that is found in the brain. It consists of inputs, a hidden layer, and outputs. Neurons are found both in the hidden layer and the output layer. It learns through self training, where it gradually modifies itself to produce the correct answer. Below I have described how this is done.

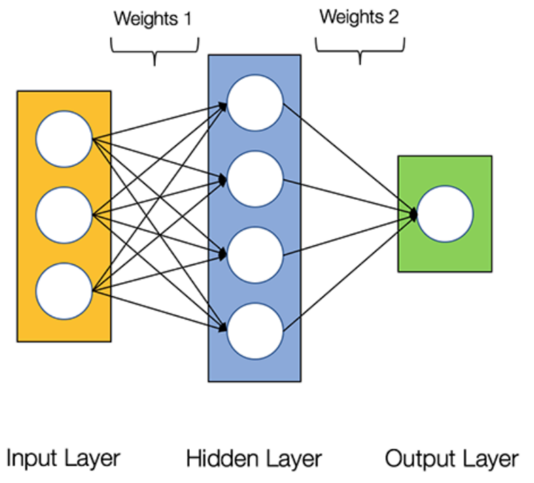


Image showing Neural Network

This Neural Network consists of two phases, the learning , or training phase, and the deployment phase. In the learning phase the machine modifies itself based on training data, until it can correctly calculate outputs based on different inputs.

During the training stage, each output will be calculated using the following formula: $y = \sigma(bias + x0 \cdot w0 + x10 \cdot w1 + ...xn \cdot wn)$, where σ is the activation function described as $\sigma(z) = 1/(1 + e^{-z})$. If the $\sigma(z)$ is high, then the neuron will activate, meaning that the output will be 1, if it is low then the output is 0. The output is then compared to the expected output, the difference being the error.

This is done using the following process:
The machine runs InitNet, in here it fills the neurons biases and weights with random numbers ranging from -1 to 1. This is done for all neurons in the hidden and output layer. I found that as the number of hidden layers increased, the time taken to successfully train decreased, up until a certain point. This point was around 32 hidden layers, so this is the number that I have chosen for this particular Neural Network.

Next, LoadTrainingSet is called. In this method, training data gets read from a file and divided into two categories, input data, and target output values. Input values are put into x_t array, sorted by row, and target data is put into the t_t array. In this case the default training dataset consists of 750 lines, defined by nPatts, each of which has 64 input values, and 8 outputs. These input values correspond to pixels in an 8x8 image, while the output values correspond to the image number that the input values represent.

Next, the program is trained, using the Train method. In here the program selects a random row from 0 to nPatts. This row then copies its input and target output values into a seperate array (x and t). Then BackProp is called, this firstly calculates the output of the neuron, and the difference in the target and its calculated output. Then it calculates the derivative of the output for the biases and weights. After this, it returns back to train, which uses the information that backProp provided to make one step in decreasing the error, it does this using DerStep. This process is repeated 3000 times, and the error should be minimised. Once this is completed the method CheckTrainSet calculates the success rate of detecting the correct image. If this is less than 95%, the program is retrained.

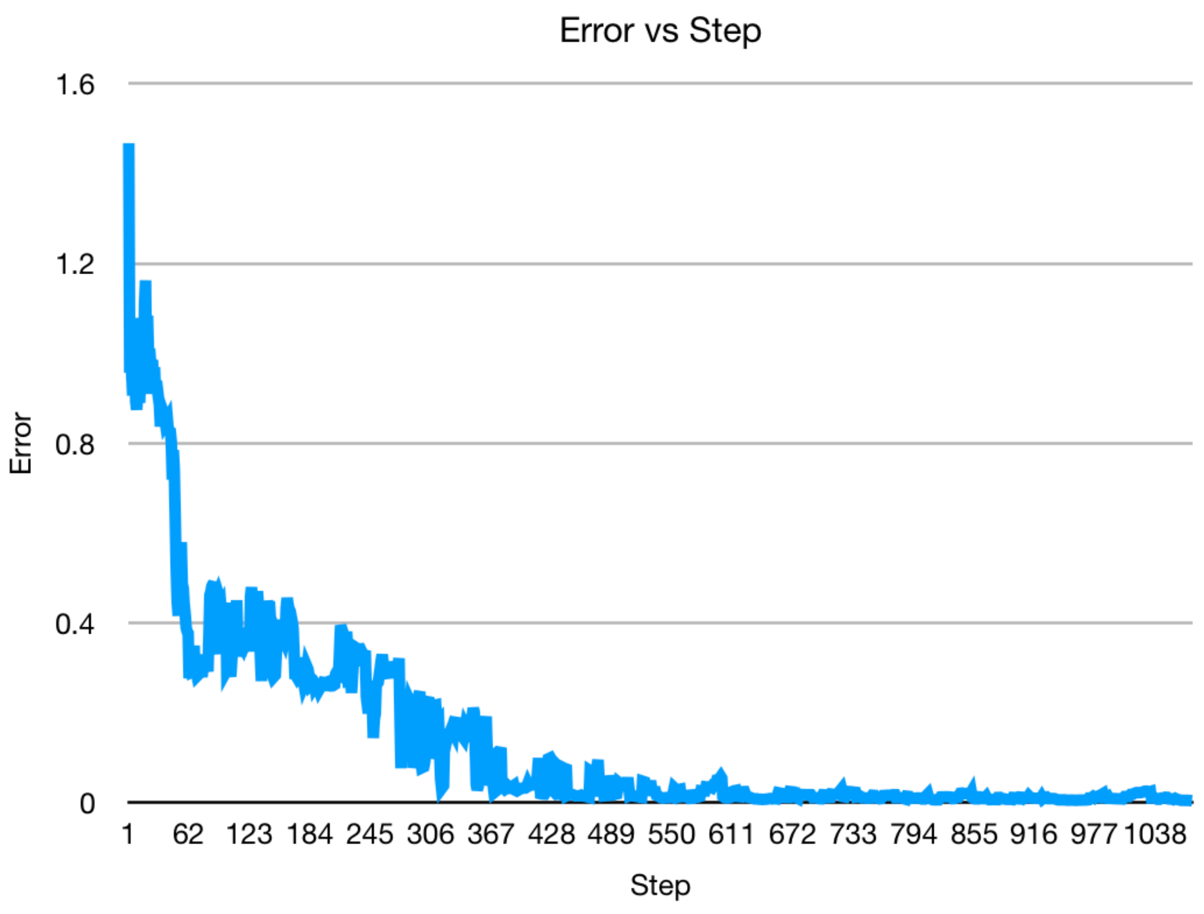
Once the program has been successfully trained, it saves the coefficients of the neurons into a file, so that it does not have to be retrained every time the program is run. This is saved into "save.txt" and is loaded at startup if the user does not wish to retrain.

Once that is complete, the program is ready to be deployed, it asks the user for a file of working data, and calls LoadWorkingSet. This is very similar to load training set, but only input data is available, and copied into x_w.

The last thing the program does, is processWorkingSet, in here it writes the calculated answers to a file called "answers.txt", this is done by copying each line of x_w into x, and calls forwardProp, which calculates the output. Using OutputMax, the highest output in the output array is calculated, returned, and saved to the file along with the image it is currently processing.

In conclusion, this program is ready to be shipped to the customer, the program is easy to use, due to the simple question and answers required. Furthermore it is easy to load a previously trained version of the program, meaning that the customer does not have to retrain with every use. To summarise, this neural net consists of 32 Neurons in the hidden layer, nu is set to 1 minX is at -1, while maxX is at 1.

The graph below shows a typical training cycle, which shows the error gradually converge. The spikes are due to the fact that I have chosen an on-line training method, rather than batch training, this means that the error fluctuates, as only one line is being trained at a time.



Reference:
https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6

Software Manual:

Thank you for buying our software.
This is a machine learning software, this means that it modifies itself until it produces the correct output. This is done by working in two stages, the training stage, and the working stage. During the training stage it is provided with a large amount of data, or inputs, as well as answers, or expected outputs. During this stage the system will tune itself until it receives the right answer, which matches the one you provided. This is done by modifying the weights and biases in the following equation: $y = \sigma(bias + x0 \cdot w0 + x10 \cdot w1 + ...xn \cdot wn)$, where σ is the activation function described as $\sigma(z) = 1/(1 + e^{-z})$. If the $\sigma(z)$ is high, then the neuron will activate, meaning that the output will be 1, if it is low then the output is 0. The output is then compared to the expected output, the difference being the error. If the error is high, the coefficients (bias, weights) will be changed, until the error is at a minimum.

After this training stage is complete, you are able to give it another set of data, with only inputs, and it will calculate the correct answer. The key to having a successfully trained program which provides reliable results is the quality of the training dataset, not only should it be of a large quantity, but the data should be similar to the working dataset.

There are several types of Machine learning softwares, this particular set is of type Neural Network. It is made up of numerous neurones which are interconnected, starting with the inputs, being processed by neurons in a hidden layer, passed to neurons in the outer layer, which calculate the final output. This is described visually in the image below. After calculating, the output of a given neuron can either be 0 or 1, which will determine if the neuron fires or not.

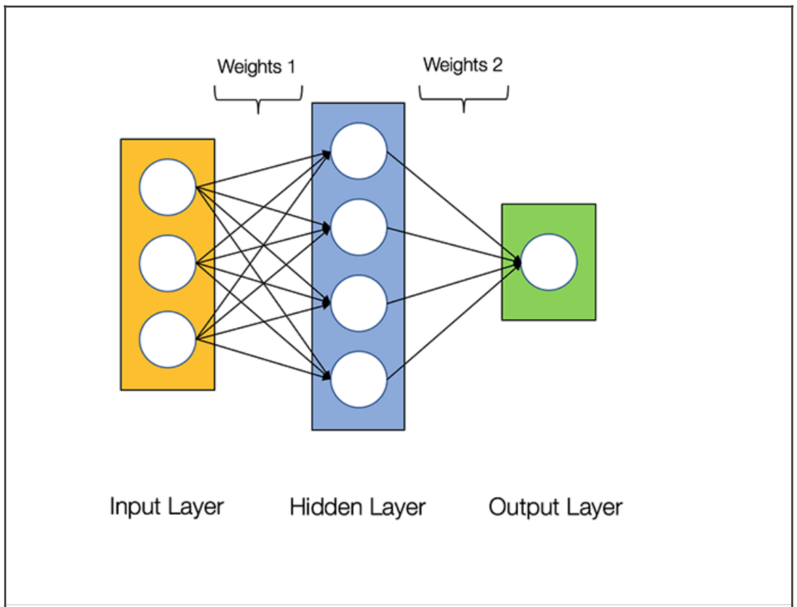


Image showing Neural Network

This piece of software processes images. With the current setup it is given 8*8 pixel images consisting of black or white, making up a road intersection. This could be modified to fit other images, the how to of which will be described later on.

Without modifying the program at all, the input data should be 64 numbers, either -10.0 (a black pixel), or 10.0 (a white pixel). The outputs will then be a number from 0-7, corresponding to the images below.

The training data is set to 750 images by default. The size of the working dataset does not matter.

We provide training and working datasets for your Neural Network. A dataset is a set of low-resolution 8x8 black-and-white images.
Each image can be one out of 8 possible types.

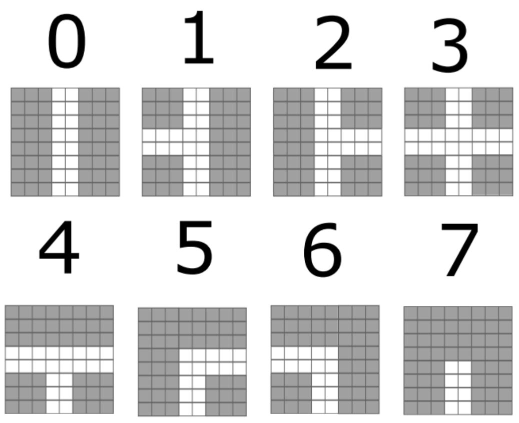


Figure 1: Images and their classifications

<u>How to run the software:</u>
To run the software, it needs to be trained first. This only needs to be done once per training dataset. To do this type in the name of the file when prompted. After that it will train on its own. Once it is done it will automatically save the required data to a file. This means that next time you open the program, if you are using similar working data, you can simply say no when it asks if you wish to train, and it will load this data into the program.

After the program has been trained, it will ask for a working data file. Again, enter the name of the file when prompted. This will create a file called "answers.txt" which will have the outputs for the corresponding inputs listed.

<u>How to modify the Software:</u>
The Function called "public MLReport()" sets up the system, so most modifications are able to be done there, there is no need to modify other methods.
To change the number of inputs:
'nPatts' is the number of inputs in the training dataset, it is set to 750 by default, as this is the amount of images provided in the "train.txt" data file. Modify this for the amount of images present in your training set.
To change the number of outputs:
'nOut' is the number of outputs. As seen in the image above, by default there are 8 images, however, if you have more or less, this is where you would change it.

NOTE: The training dataset and working dataset should be similar.
<u>How to achieve good results using a Neural Network:</u>
If the program has been modified, it is possible that it takes a very long time to train, or does not train successfully at all. If this is the case then the difference in the machines output and target answer is large, also known as the error. In this case a few modifications can be made:

The number of hidden layers: This is currently set to 4x the number of outputs, which means that it achieves convergence quickly, if the number of outputs is changed, then this should be changed as well.

Nu. Nu is the search constant, it is currently set to 1, this could be increased so that it takes bigger search steps, but the error found will not be as precise.

minX and maxX. These are currently at -1 and 1 respectively, if none of the other changes work, then making the minimum lower and maximum larger could achieve convergence. These have to be changed in all places where InitNet() is called.