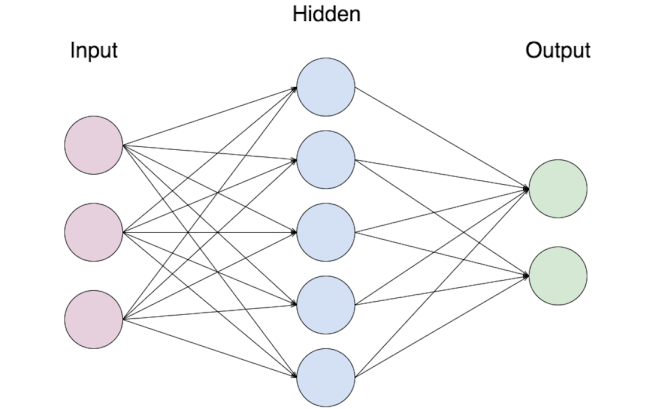
|  |  |  |  |
| --- | --- | --- | --- |
| Identification of Digits using Neural Networks | | May 22  https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/280px-Colored_neural_network.svg.png | |
| Abstract: Neural Networks are fast becoming a major part of our lives. They have found applications in fields like Artificial Intelligence, Computer Vision, Data Analytics and Machine learning among others. This project creates an Artificial Neural Network (ANN) to identify and classify numbers according to labels to train the neural network, which is then used to identify numbers from images that can be clicked in the real world. |  | |

**1. INTRODUCTION**

Artificial Neural Networks (ANN) are inspired by Animal Brains. They learn to perform tasks by considering various examples, without being programmed. An ANN has an Input Layer and an Output Layer, connected by a layer in between called the Hidden Layer. The size of the input layer is the size of the input, and size of the output layer is the size of the label to which the input has to be classified. The Hidden Layer can have any size specified by the programmer. As the size of the HL increases, the precision of the ANN increases. But this also results in an exponentially higher time for compilation, because each node of the IL is connected to each node of HL, which is in turn connected to each node of the OL. In other words, as the size of the HL increases, the Neural Network learns better. But this in turn increases the time it takes to learn.

The ANN in this project is made up of an Input layer with 784 nodes, and an Output Layer with 10 nodes. The IL has 784 nodes because every image is of size 28X28 pixels, resulting in an array of size 784. The OL has 10 nodes because each array (or image of a digit) is mapped to one of 10 digits (0-9). In each node is stored the probability of an image being one of the 10 digits, and the image is identified to be the label of the node with the highest probability. The Hidden layer, according to the program, has 100 nodes. This number can be changed as seen fit.

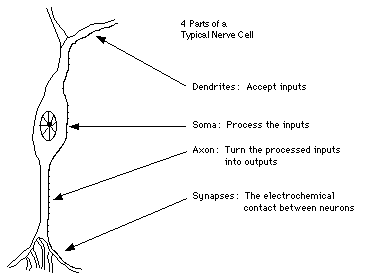


The Neural Net is created or identified as two matrices, named ‘wih’ and ‘who’. ‘wih’ is the weighted matrix between the Input layer and Hidden Layer, and ‘who’ is the weighted matrix between Hidden Layer and Output Layer. These matrices have a size of the product of the number of nodes of its layers.

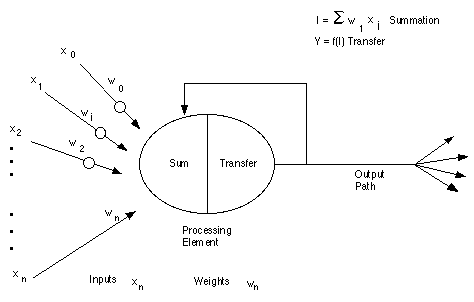
**2. BIOLOGICAL VS ARTIFICIAL NEURONS**

The most basic element of the human brain is a specific type of cell, which provides us with the abilities to remember, think, and apply previous experiences to our every action. These cells are known as neurons, each of these neurons can connect with up to 200000 other neurons. The power of brain comes from the numbers of these basic components and the multiple connections between them.

All natural neurons have four basic components, which are dendrites, soma, axon and synapses. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally non-linear operation on the result, and then output the final result. The figure below shows a simplified biological neuron and the relationship of its four components.



The basic unit of neural networks, the artificial neurons, simulates the four basic functions of natural neurons. Artificial neurons are much simpler than the biological neurons. The figure below shows the basic structure of an artificial neuron.



Artificial neural networks emerged from the studies of how brain performs. The human brain consists of millions of individual processing elements called neurons that are highly interconnected.

ANNs are made up of simplified individual models of the biological neurons that are connected together to form a network. Information is stored in the network in the form of weights or different connection strengths associated with the synapses in the artificial neuron models.

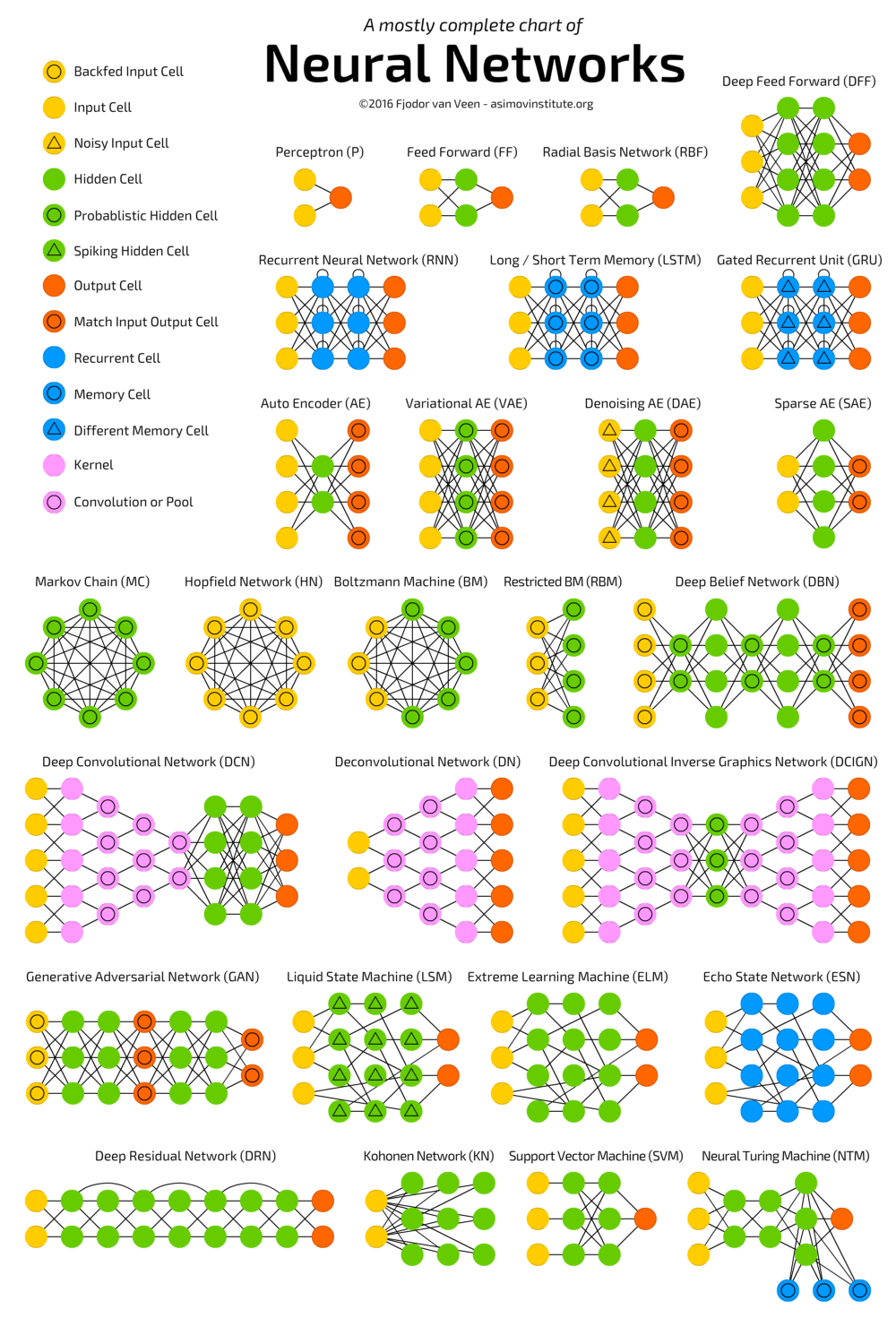
Many different types of neural networks are available and multilayered neural network are the most popular which are extremely successful in pattern reorganization problems. An artificial neuron is shown in the figure. Each neuron input is weighted by wi. Changing the weights of an element will alter the behavior of the whole network. The output y is obtained summing the weighted inputs and passing the result through a non-linear activation function.

**3. TYPES OF NEURAL NETWORKS**

* **Recurrent Neural Network:** An RNN is a type of Artificial Neural Network in which the output of a particular layer is saved and fed back into the input. This helps predict the outcome of the layer.
* **Convolutional Neural Network:** A CNN consists of one or more convolutional layers. These layers can be completely interconnected or pooled. Before passing to the before passing to the next layer, the convolutional layer uses a convolutional operation on the input.
* **Modular Neural Network:** A modular neural network has a number of different networks that function independently and perform sub-tasks. The different networks do not really interact with or signal each other during the computation process. They work independently towards achieving the output.
* **Sequence-to-sequence Neural Network:** An SSNN model consists of two RNNs. There’s an encoder that processes the input and a decoder that processes the output. The encoder and decoder can either use the same or different parameters.

**4. FEATURES OF ANN’s**

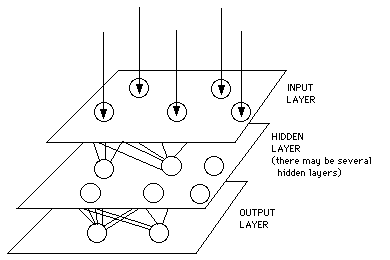
* Their ability to represent non-linear relations makes them well suited for non-linear modeling in control systems.
* Adaptation and learning in uncertain system through off line and on line weight adaptation.
* Parallel processing architecture allows fast processing for large-scale dynamic system.
* Neural network can handle large number of inputs and can have many outputs.
* ANNs can store knowledge in a distributed fashion and consequently have a high fault tolerance.



**5. DESIGNING ANNs**

Designing of a neural network consists of:

* Arranging neurons in various layers.
* Deciding the type of connection among neurons of different layers , as well as among the neurons within a layer.
* Deciding the way neurons receive input and produces output.
* Determining the strength of connection that exists within the network by allowing the neurons learn the appropriate values of connection weights by using a training data set.



**6. WORKING**

The project is split into 3 programs to make the execution faster and more efficient; mnist\_pickler.py, neural\_net\_trainer.py and Number Identifier.py. These programs complete specific functions necessary to finish the project. They are interlinked with each other by the use of pickled files.

**6.1 mnist\_pickled.py**

This program uses the following import files; pickle, numpy, scipy and matplotlib.

‘mnist\_train.csv’ and ‘mnist\_test.csv’ are the two datasets used for training and testing the datasets. However, since they are significantly large ‘.csv’ format files, they take a long time to load. Hence after the dataset is loaded using ‘numpy’ and operations performed on it, it is converted into pickle format and stored into a ‘.pkl’ file. This is done using functions from the ‘pickle’ library.

Loading data from the ‘.csv’ file is fairly easy. This is done using a function in ‘numpy’ library, which converts the data into a Numpy array. This data is basically images stored as rows of RGB valued pixels. Each row is of length 784, because they represent 28X28 pixels. All operations are done in greyscale, hence it is important to convert the RGB pixel values into Greyscale values. This is done by multiplying each value with (0.99/255) and adding 0.01 to avoid 0’s. Also, the first element of each row is the label of the image, i.e., the number that the image signifies. This is taken separately from the array and stored in new arrays. Their ‘one-hot’ values are also calculated and stored in different arrays. All these new arrays are then pickled into ‘pickled\_mnist.pkl’ file, which is stored in the same location as the programs.

**6.2 neural\_net\_trainer.py**

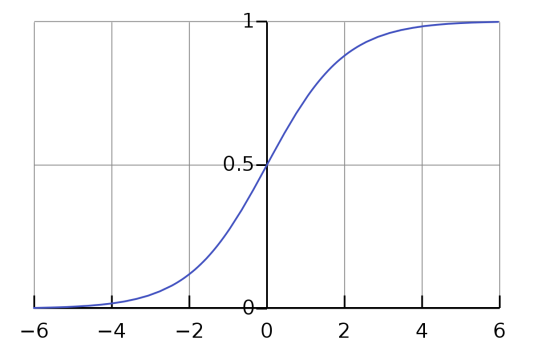
This program uses the following import libraries; numpy, matplotlib, time, tensorflow, pickle and scipy.

First of all, the MNIST dataset is accessed from ‘pickled\_mnist.pkl’. This was created by the previous program (mnist\_pickled.py), and contains the MNIST dataset after processing. Because the dataset doesn’t change very often, the same pickle can be re-used. Hence it is not necessary to run the previous program each time. This is also why the program was separated from this one.

As explained earlier, our neural net has 784 nodes in the Input Layer (which is square of the image size), 10 nodes in the Output Layer (representing each digit from 0 to 9) and 100 nodes in the Hidden Layer (just because). The Learning rate is 0.2. These two values can be tweaked to get best results from the Neural Net. The Learning Rate defines the size of the corrective steps that the model takes to adjust for errors in each observation. A higher Learning rate would allow for faster execution of the program, but with a reduced accuracy and vice versa.

The first step is to create the two matrices ‘wih’ and ‘who’. Conventionally, this is done by assigning random values to the Matrix, which is then trained to near perfection. This is what is done in case of this ANN. After this function, ‘wih’ is a matrix of size 784X100 filled with random values and ‘who’ is a matrix of size 100X10 filled with random values. Of course starting with random values has its weaknesses, but when the Net is trained with enough data (in this case, 10000 images), the Net becomes accurate enough to function.

Thus the ‘wih’ and ‘who’ matrices are created and assigned values. These two matrices are the foundation of the Neural Net that we build. They are then used to train the Net using the data. To train the data, the function also needs the image (which after the previous program, is a 1D array of size 784 filled with pixel values of the greyscale image), the label corresponding to the image (a float value of the number in the image) and the learning rate. To train the Neural Net, first the Web between Input and Hidden Layer is calculated by taking Dot product of the existing web (‘wih’) and the input vector (the given image). This is Sigmoided to maintain the Greyscale consistency. Then the Web between the Hidden and Output layer is calculated by taking dot product of the existing web (‘who’) with the previously calculated new Web between input and Hidden Layer, which is also Sigmoided. To train the Neural net, the error between the calculated and output and targeted output (the original value of the image as taken from the label) is calculated, and added to the existing web between Hidden and Output Layer, after also taking into account the Learning Rate. The same is done to calculate the new Web between Input and Hidden Layer. These two values are passed as the new Webs, to be trained by the next image. Thus the Neural Net is trained for every image in the MNIST training dataset, eventually creating the webs ‘wih’ and ‘who’ ready to take on any new images.



At some instances in the above function, some arrays were sigmoided. Sigmoid is a function that converts any value between -∞ and ∞, and assigns them a value between 0 and 1. Thus the pixels values in RGB format, which is between 0 and 255, can be given values between 0 and 1 when passed through a sigmoid function.

Once the Neural Net has been trained with the dataset, we can proceed to test it with the MNIST Test dataset. But in this project, testing is done with pictures clicked in the real world. This is done in the next program.

Before being able to test the Neural net in the next program, it is necessary to allow that program to access the Neural Net that was trained in this program. In case you are wondering why it was necessary to separate the two programs, understand that training a Neural net is a time consuming process. This Neural net was trained using only 10000 Images. In the real world, 10000 is not nearly enough, and hence it is necessary to save the net to be accessed later, whether that is to test or to train the Net. An Artificial Neural Net (ANN) is essentially just the webs between each layer, in our case, ‘wih’ and ‘who’. Hence the two arrays will be pickled into the file ‘neural\_net.pkl’.

**6.3 Number Identifier.py**

Remember how the Neural Net we created has a specific number of Input nodes? This number was generated as the size of the image. This was specified by the MNIST dataset. This however means that to test any image with the trained Neural net, our images have to be converted to a specific format similar to the images in MNIST dataset. So the first major function in this program is used to convert a given ‘.jpg’ image into a format that can be tested using our Neural Net. This means that the image has to be converted into a 1D array of size 784, and the values should be the pixel values of the image in Greyscale.

This is exactly what the function does in this program. The function, after first opening the image, converts it to greyscale, resizes it to a 20X20 size image and pastes it on the center of a 28X28 size, just as MNIST should be. This is however in a 2D array, which is then flattened to make it 1D. Next, the following formula is applied to every value in the array:

*pic\_array[i] = ( (255 - pic\_array[i]) \* 0.99/255 ) + 0.01*

Additionally, if any value is less than 0.4, it is changes to 0.01. This is a basic Thresholding operation. Now the picture has been converted to a format that can be passed to the identifying function.

To identify an image, the image is first multiplied with the web between Input Layer and Hidden layer, i.e., dot product of ‘wih’ and ‘input\_vector’. This is sigmoided and the multiplied with the web between Hidden Layer and Output Vector, i.e., another dot Multiplication. This is again sigmoided and returned. The argument of the Maximum valued node is the identified number, and the value of the Maximum valued node is the accuracy of that identification.

**7. CONCLUSION**

When tested with the MNIST testing Dataset for Handwritten digits, The trained Neural Net has an accuracy of 94.3%. While some might argue that this net shows signs of Overfitting, the Neural net has shown some considerable results when tested with real pictures.

* The Highest Accuracy of a right match is 99.98%.
* The Highest Accuracy for a wrong match is 99.37%.
* The Lowest Accuracy for a Right match is 5.32%.
* The Lowest Accuracy for a Wrong match is 4.86%.

When tested with 9 pictures of digits from 1 to 9, the Neural net recognized 77.77% of the images correctly. Interestingly, the Highest accuracy for the Wrong matches was only 39.86%, while the Lowest accuracy for the Right matches was 13.50%.

**8. REFERENCES**

1. [*https://www.youtube.com/watch?v=h3l4qz76JhQ*](https://www.youtube.com/watch?v=h3l4qz76JhQ)
2. [*https://www.python-course.eu/neural\_network\_mnist.php*](https://www.python-course.eu/neural_network_mnist.php)
3. [*https://docs.python.org/3/library/pickle.html*](https://docs.python.org/3/library/pickle.html)
4. [*https://numpy.org/*](https://numpy.org/)
5. [*https://www.tensorflow.org/*](https://www.tensorflow.org/)