1. Neural Network

A neural network is a particularly inventive AI model, which lends its name and basic design of connected neurons to the brain's network of neurons itself. It is designed to recognize (or predict) patterns and solve problems by learning from large sets of data in the given domain.

A neural network "learns" by making (at first random) predictions on a set of inputs and slowly adjusting the parameters of each neuron according to a function of some "loss" metric. Loss is a measure of the error the network makes while outputting predictions. In short, the more the network predicts, the more the parameters of the network are tweaked against the errors it produces (or rather functions based on said errors).

2. Neuron

A singular neuron is made up of a set of weights and biases, whose values are in the range of [0, 1], and a so-called "activation function". The network part of neural [b]network[/b] comes from the fact that these neurons take some numerical input, multiply it by the weights, add the bias and pass this output into neurons of the next layer. In this way the neurons are connected throughout and depend on each other’s inputs/outputs.

To summarize, neurons are arranged in layers, take an input and transform it, and then pass it on to the next layer's neurons. The very last layer is then interpreted as a result of the prediction. More on this in the next sections.

3. Input

The input into a network refers to the initial data fed into the network at the first layer, the input layer, which is then processed by the network to produce an output. It's typically structured as a one-dimensional array of decimal values in the range [0, 1], which may have been transformed from some initial arbitrary sample.

This network features the "MNIST" dataset, a set of tens of thousands of handwritten digits compiled into images of 28x28 pixels. For each image sample, the pixels are transformed and fed into the system as a 28 x 28 = 784 long array of single byte values (0 .. 255, later divided to fit the range [0, 1]).

4. Activation Functions

What was omitted in the "Neurons" section is the fact that the output is finally driven by the activation function of the neuron. Before the neuron outputs a value, the weighted inputs are first passed through the activation function, which controls the intensity and frequency of outputs.

Activation functions could be simple linear functions or more advanced ones like sigmoid or tanh functions, which ensure a smooth range of output values. Depending on the domain, some functions may perform better than others. In this application, every neuron (except for input neurosn that don't use any) is assigned a tanh function.

5. Output

Similar to the input, the output is an array of decimal values. The difference to the input is that the output is shaped by the network designer and can therefore be arbitrarily interpreted.

In this application, the output is an array of 10 values, one for each digit, that represent the confidence that the current input corresponds with said digit. For example, if the 4th value of the array is 0.85, the network predicted that the current input image shows a handwritten 5 with a certainty of 85%. (the digits are zero-indexed. the first value is the digit 0 and so on).

6. Layers

A neural network is organized into layers of neurons, where neurons of the same layer only interact with neurons of the immediately following layer. There are three fundamental types of layers: one input layer, whose inputs are the raw, unweighted samples of the dataset, an arbitrary (chosen) number of so-called hidden layers and one output layer.

The input layer simply acts as an interface for the input to enter the network and therefore has no weights, biases or activation functions. As such, it is ignored in training. It's dimensions (number of neurons) directly correspond with the feature dimensions of the dataset.

The hidden layers are called hidden as they lie between the input and the output layer, which, in contrast, are easily interpretable, where the hidden layers can be described as a computational blackbox. Together with the output layer, they contain the neurons whose weights and biases are tweaked during training.

The final layer of the network is the output layer, which contain the neurons carrying the result of a prediction. As stated in the "Output" section, the output values are interpreted as confidences of a prediction. This layer is designed by a user for the precise sake of interpretability of a network's workings.

7. Forward Propagation

Forward propagation is a fundamental algorithm in a neural network that drives both the prediction and training processes. In prediction, a single forward pass determines the output of a single input sample, whereas for training, the forward pass is followed by backward propagation to adjust the weights and biases of each neuron. In both scenarios, the forward propagation steps are the same.

Forward propagation starts at the input layer that passes in the features of the current sample as an array of values. These values are then passed to (in the case of a fully connected network) each and every neuron of the next layer (the first hidden layer). These neurons then sum and weigh these inputs, calculate a result via their activation functions, and output a new value that is passed on to the next layer. This process is repeated until the output layer is reached, and a result can be read.

8. Backward Propagation

The backward propagation algorithm does not start immediately after the forward pass, as it requires a set of errors calculated from the current prediction. As mentioned previously, a

prediction contains of a set of confidences, i.e. values in the range of [0, 1] for each output neuron. These values are used to calculate the differences towards the correct target, which is known beforehand (which makes NNs a model of the supervised learning school).

These errors, also called loss, are the first input into the backward pass. Each backward pass, happening one layer at a time, calculates the derivates of a neuron's error based on its weights and bias, in order to adjust these weights and biases in the negative direction. This process is called "gradient descent", as it gradually corrects the weights and biases over a large number of iterations until the error is minimized. The derivative of the current layer acts as the input into the next backward pass, until the final corrective step happens at the first hidden layer. After this, a single training iteration is complete and a new sample can be predicted, starting again at the forward pass.