|  |
| --- |
| Artificial Neural Networks |
| IBM's Brain-Inspired Chip Tested for Deep Learning - IEEE Spectrum |

|  |  |
| --- | --- |
| 28th July 2020 | Theoretical documentation of ANN |

Artificial Neural Networks

Theoretical documentation of ANN

# What ARE Artificial neural networks?

Neural nets are a means of doing machine learning, in which a computer learns to perform some task by analyzing training examples. Usually, the examples have been hand-labeled in advance. An object recognition system, for instance, might be fed thousands of labeled images of cars, houses, coffee cups, and so on, and it would find visual patterns in the images that consistently correlate with particular labels.

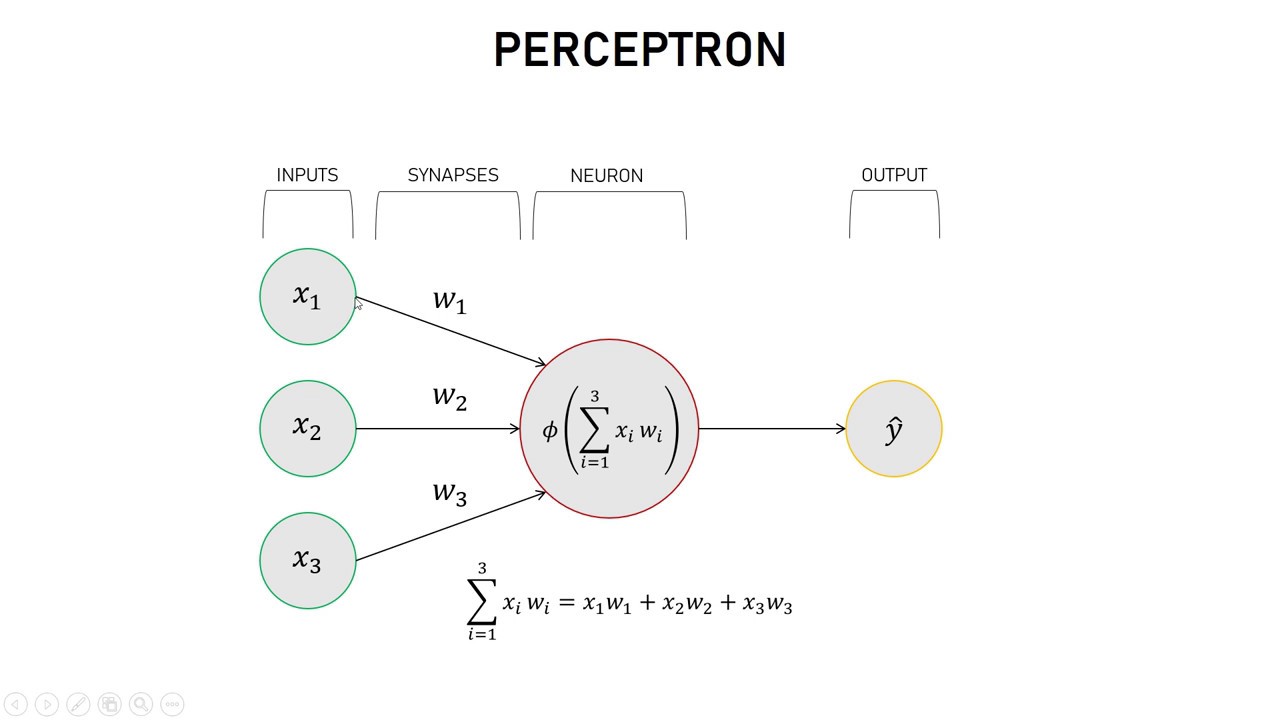
Modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today’s neural nets are organized into layers of nodes, and they’re “feed-forward,” meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

To each of its incoming connections, a node will assign a number known as a “weight.” When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node “fires,” which in today’s neural nets generally means sending the number — the sum of the weighted inputs — along all its outgoing connections.

When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the succeeding layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs.

# COmponents of a neural networks

1. Neuron: -An artificial neuron is a mathematical function conceived as a model of biological neurons, a neural network. Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs (representing excitatory postsynaptic potentials and inhibitory postsynaptic potentials at neural dendrites) and sums them to produce an output (or activation, representing a neuron's action potential which is transmitted along its axon). Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function or transfer function
2. Activation Function: - In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input.
3. Input layer: - Takes in the features of a dataset as the input
4. Output layer: - Provides the output after all the features are processed
5. Hidden Layers: - There can be n number of hidden layers. Each hidden layer computes more and more complex features which helps us in form better and more accurate predictions

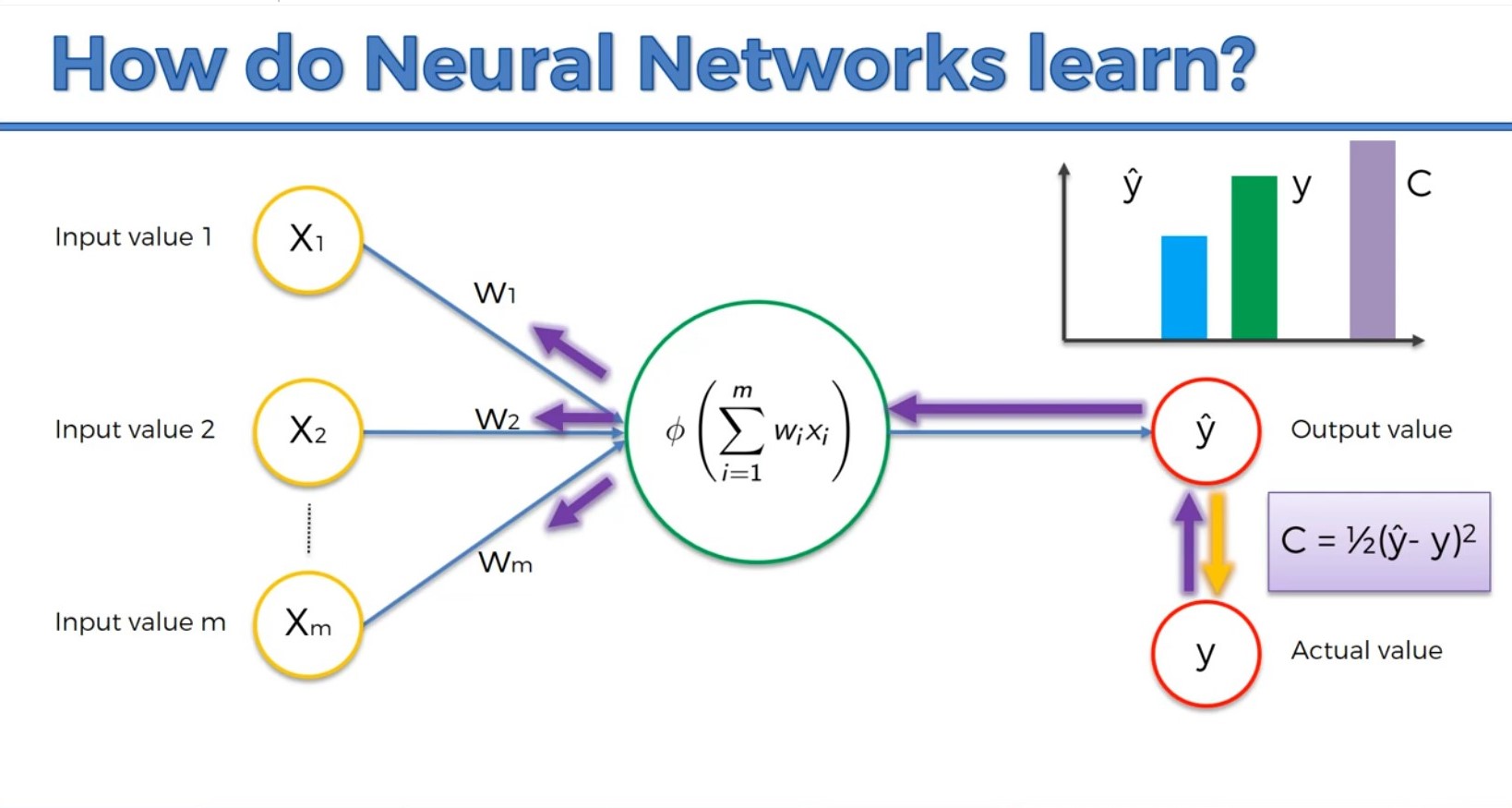


# How does a neural network learn?

Fundamentally, each neuron in a neural network is just a mathematical function. Each neuron computes a weighted sum of its inputs—the larger an input's weight, the more that input affects the neuron's output. This weighted sum is then fed into a non-linear function called an activation function—a step that enables neural networks to model complex non-linear phenomena.

The power of Rosenblatt's early perceptron experiments—and of neural networks more generally—comes from their capacity to "learn" from examples. A neural network is trained by adjusting neuron input weights based on the network's performance on example inputs. If the network classifies an image correctly, weights contributing to the correct answer are increased, while other weights are decreased. If the network misclassifies an image, the weights are adjusted in the opposite direction.

This procedure allowed early neural networks to "learn" in a way that superficially resembled the behavior of the human nervous system. The weigh associated with each of the ‘synapses’ is determined by the cost function. Cost function is an essential part of supervised learning as it acts like the reward/punishment for the model to make the right or the wrong decision. Our main focus for an ideal model is to minimize the cost

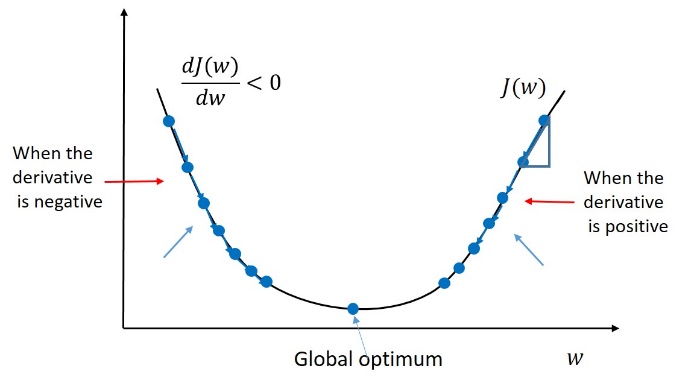


As we can see from the above diagram, the purple lines going backwards represents the backpropagation algorithm. In backpropagation we calculate the collective cost of the model and then backpropagate through the network to adjust the weights. This process is repeated till the total cost of the model reduces to minimum. In order to reduce the cost function we use different algorithms

# Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

Gradient descent is used to reduce the cost function of the model.



Standard Gradient descent updates the parameters only after each epoch i.e. after calculating the derivatives for all the observations it updates the parameters. This phenomenon may lead to the following caveats.

* It can be very slow for very large datasets because only one-time update for each epoch so large number of epochs is required to have a substantial number of updates.
* For large datasets, the vectorization of data doesn’t fit into memory.
* For non-convex surfaces, it may only find the local minimums.

# Stochastic gradient descent

Stochastic gradient descent updates the parameters for each observation which leads to a greater number of updates. So, it is a faster approach which helps in quicker decision making.

Here, lots of oscillations take place which causes the updates with higher variance i.e. noisy updates. These noisy updates help in finding new and better local minima.

Disadvantages of SGD

* Because of the greedy approach, it only approximates (stochastics) the gradient.
* Due to frequent fluctuations, it will keep overshooting near to the desired exact minima

# Mini Batch Gradient Descent

Another variant of GD to address the problems of SGD, it lies in between GD and SGD. Mini-batch Gradient descent updates the parameters for a finite number of observations. These observations together are referred to a batch with some fixed size. Batch size is chosen as a multiple of 64 e.g. 64, 128, 256, etc. Many more updates take place in one epoch through Mini-batch GD.

Advantages of Mini-batch GD

* Updates are less noisy compared to SGD which leads to better convergence.
* A high number of updates in a single epoch compared to GD so a smaller number of epochs are required for large datasets.
* Fits very well to the processor memory which makes computing faster.