Neural Networks

# Introduction

Usage:

* Image recognition
* Speech / word recognition (Alexa)

Two main types of Neuron Nets:

* Classifier - used when you aim to produce classes.
* Regression – used when you aim to produce numbers.

One more thing about classifiers: there is no relation between classes. Class dog and cat are two different classes. It means, that there is no ‘middle’ value between those classes. Numbers may be used as names. If you use numbers as names (dog=0 and cat=1, mouse=2) numbers 0.5 or 1.5 mean nothing. The meaning has only 0, 1 and 2.

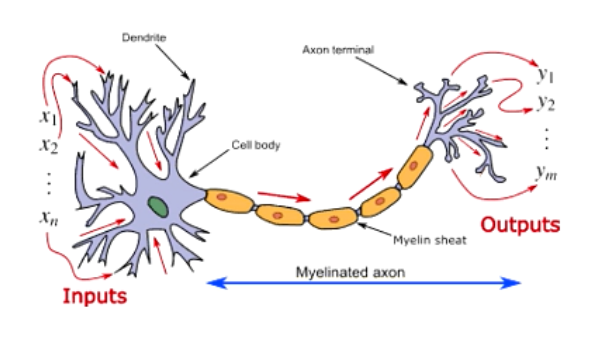


Figure Single brain neuron

How Neural Network works?

The concept is taken form biology. Brain is composed of neurons. Single neuron is connected on both sides with many other neurons. It receives many inputs (x1, x2, …, xn) and produces many outputs (y1, y2, …, ym). Scientist figured out that a single input, even the strong one, is not enough to activate neuron to produce output. Some inputs are more important than others. Single neuron must receive many inputs of certain strength to fire out outputs. Outputs are inputs for the next neurons.

# Single Neuron in mathematical concept

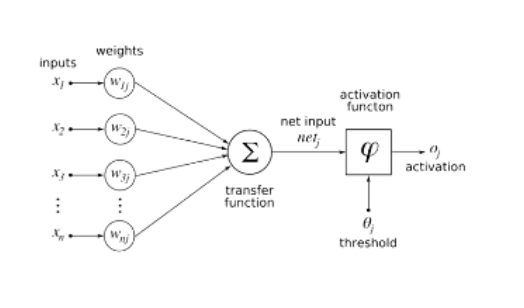


Figure Single AI Neuron

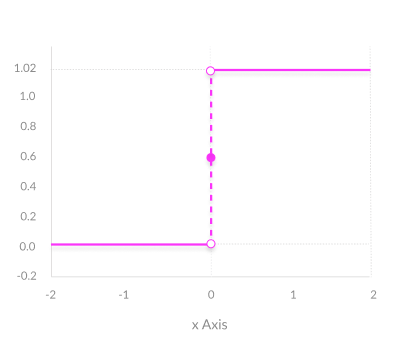
Mathematicians modelled it mathematically by applying:

* **Inputs** (x) – inputs take data from previous output.
* **weights** (w) to describe that some inputs are more important than others. Weights are numbers assigned to each input. Weights must differ from each other.
* **transfer function**, called also **Net input function** - it calculates a **net input** which is usually equal to the sum of **weighted inputs** and **biases** (b)**:**
  + **weighted inputs** are sum of the inputs multiplied by given weights.
  + **biases** – extra, disproportionate weight in favour of something.
* **net input** is passed to **activation function**. Activation function uses **threshold**. If the output produced by activation function is above threshold, it is ‘activated’ – gives output.

Examples of activation function types:

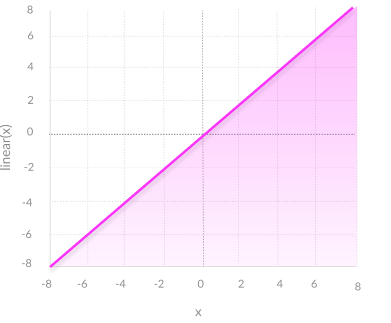
* **Binary Step Function**

If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.



* **Linear Activation Function**

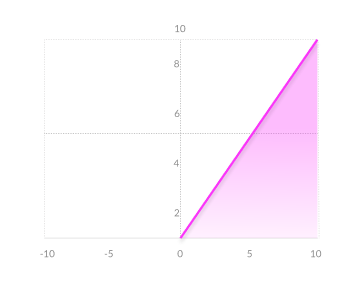
Creates an output signal proportional to the input. In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no



Non-Linear Activation Functions: Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network’s inputs and outputs, which are essential for learning and modelling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality.

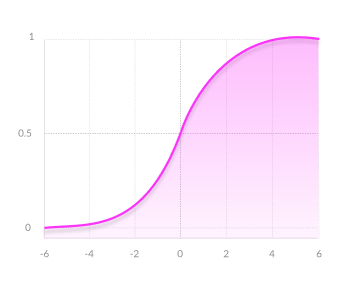
* **ReLU (Rectified Linear Unit)**

ReLU is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance (computationally efficient).



* Sigmoid / Logistic

The Sigmoid Function curve looks like a S-shape. The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

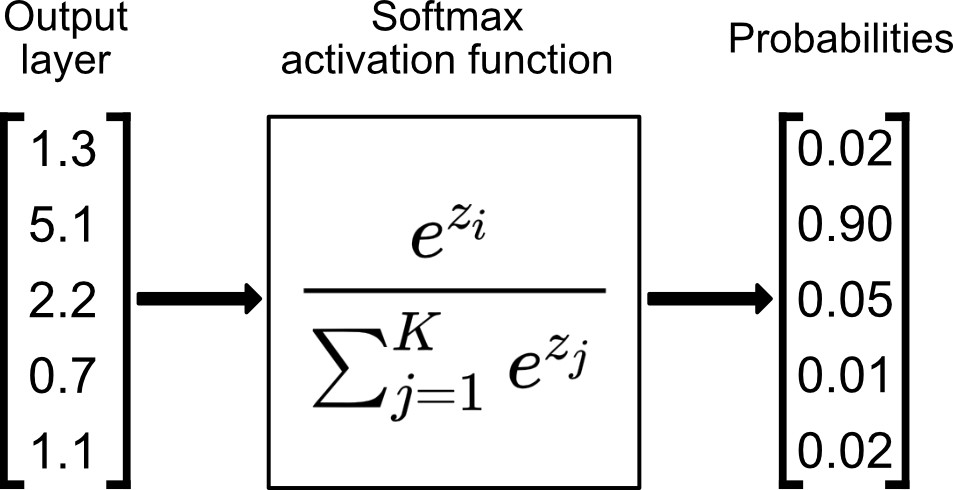


* **Softmax**, also known as SoftArgMax or Normalized Exponential Function

Sofmax function takes vectors of real numbers as inputs and normalizes them into a probability distribution proportional to the exponentials of the input numbers. Before applying, some input data could be negative or greater than 1. Also, they might not sum up to 1. After applying Softmax, each element will be in the range of 0 to 1, and the elements will add up to 1. This way, they can be interpreted as a probability distribution. For more clarification, the larger the input number, the larger the probabilities will be.

Simply saying, softmax function takes inputs and returns an array of probabilities.

Typically, Softmax is used only for the **output layer**, for neural networks that need to classify inputs into multiple categories.



# Neuron Nest

Neuron Networks are composed of multiple **layers**. Each layer is composed of multiple single neurons. A node layer is a row of neurons. The basic idea is that every single neuron from one layer is attached to every single neuron in immediate neighbour layer. Such Neuron Net it **densely connected**.

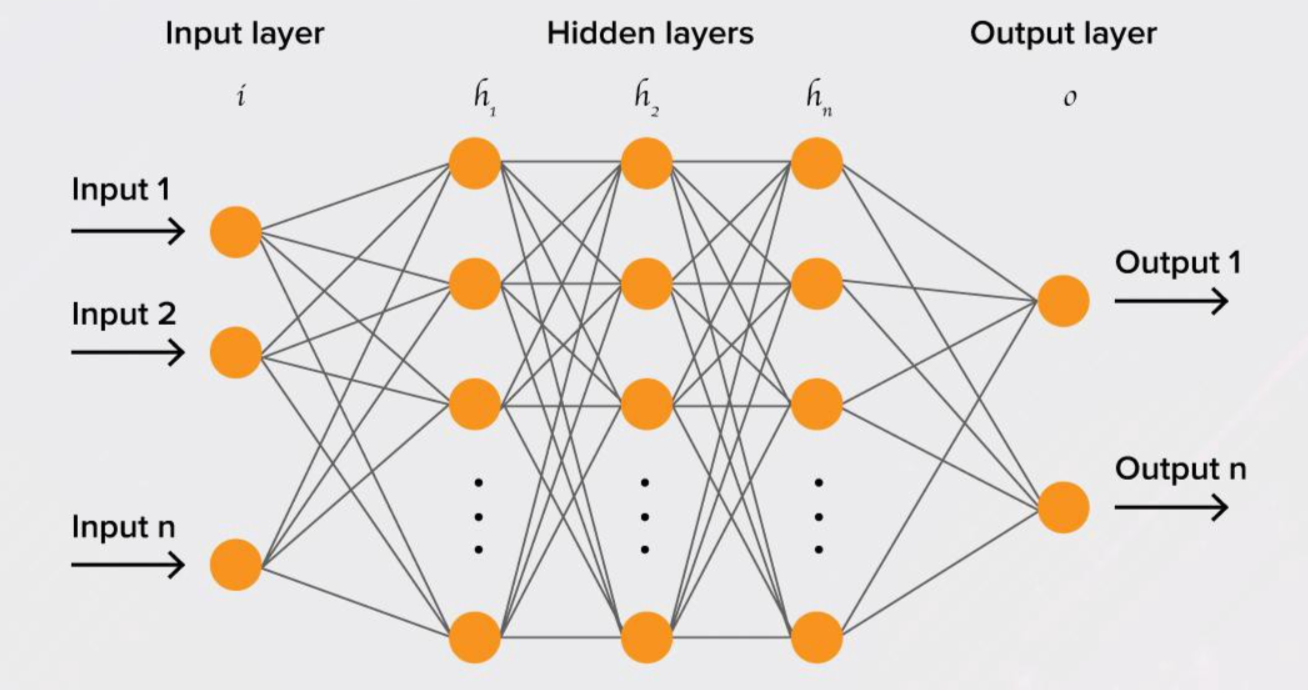
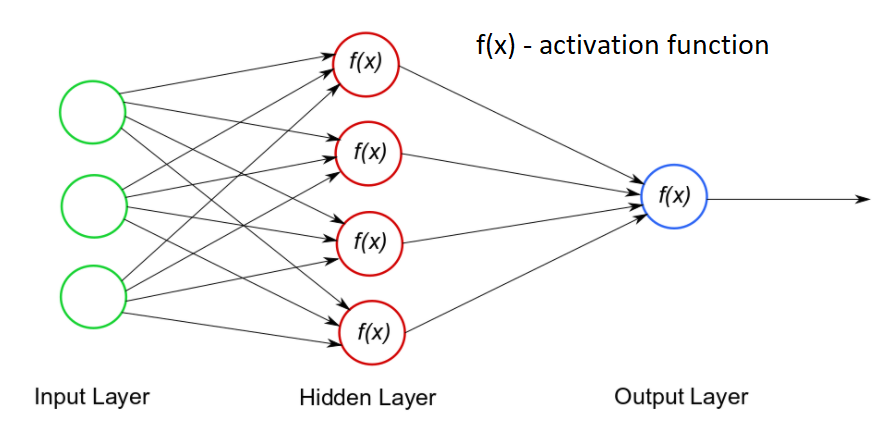


Figure Neuron Networks – input layer, hidden layer, output layer

There are three types of Neuron Network layers:

* **input layer** – it is the first layer of NN. The number of inputs in input layer is equal to the number of features. Each feature goes to separate input. Input layer does not have activation function.
* **hidden layer** – the number of hidden layers may vary, as well the number units (single neurons) in each hidden layer. Unites determine the number of outputs from the NN layer. The problem is to choose the correct number of hidden layers and number of units in each hidden layer. One hidden layer is sufficient for most problems. There are some empirically derived rules-of-thumb that states that ‘the optimal size of the hidden layer is usually between the size of the input and size of the output layers.’.
* **output layer** – it is the last layer of NN. In classification Neuron Networks, the number of outputs must be equal the number of classes.

Every layer can have different activation function. Activation functions reside within neurons, but not all neurons. Hidden and output layer neurons possess activation functions but input layer neurons do not.



Neuron Networks are usually implemented in **Feed Forward** way (**Feed Forward Neuron Net; FFNN)**. It means that they work ‘left to right’ – take input, calculate it, pass output forward. It is much easier to implement such Neuron Networks. The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards. Another implementation may forward-back.

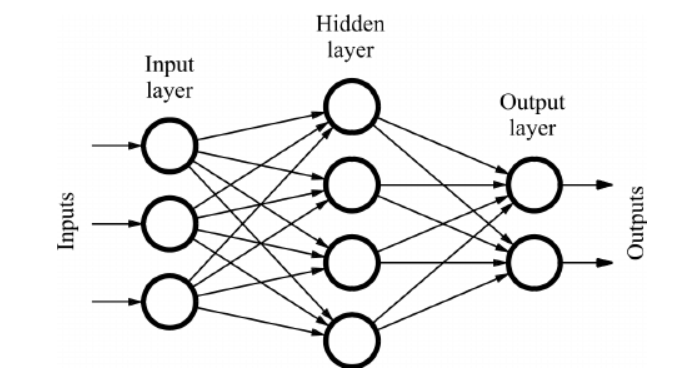
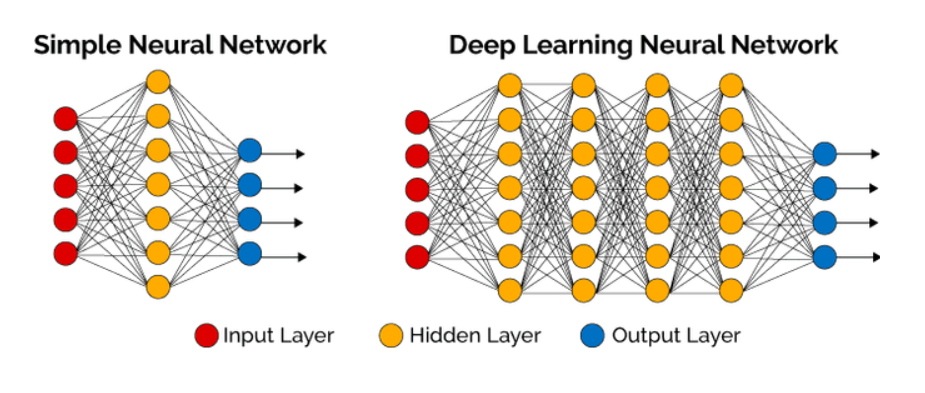


Figure Feed Forward Neuron Network

**Deep-learning networks** are distinguished from the more commonplace single-hidden-layer neural networks by their depth; that is, the number of node layers through which data must pass in a multistep process of pattern recognition.



Earlier versions of neural networks such as the first perceptrons were shallow, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as “deep” learning.

# Training Neuron Nets

Neuron Networks are trained iteratively. During training process new set of weights and biases are tested. Based on this set the trained model predicts output. The predicted output is compared to the test target data and the **error** is calculated. The fully trained model contains the optimal set of weights and biases, for which the error is the smallest.

We cannot calculate the perfect weights for a neural network; there are too many unknowns and too many options. Instead, the problem of learning is cast as a **search or optimization problem** and an algorithm is used to navigate the space of possible sets of weights the model may use in order to make good or good enough predictions. Typically, with neural networks, we seek to minimize the error. That requires a **loss function** to calculate the **error**.

**Gradient descent** is an iterative optimization algorithm for finding a local minimum of a differentiable function. The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error. The problem is solved once the minimum is found (the gradient is zero). There may be multiple local minimum points.

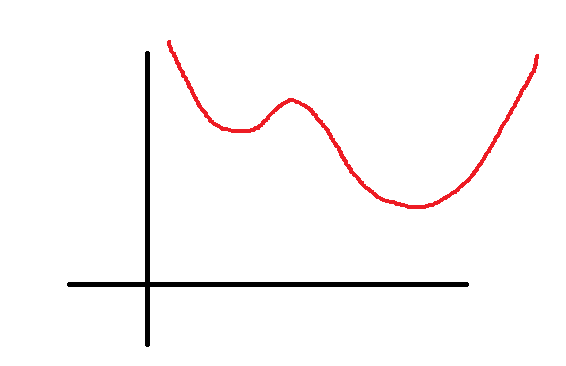
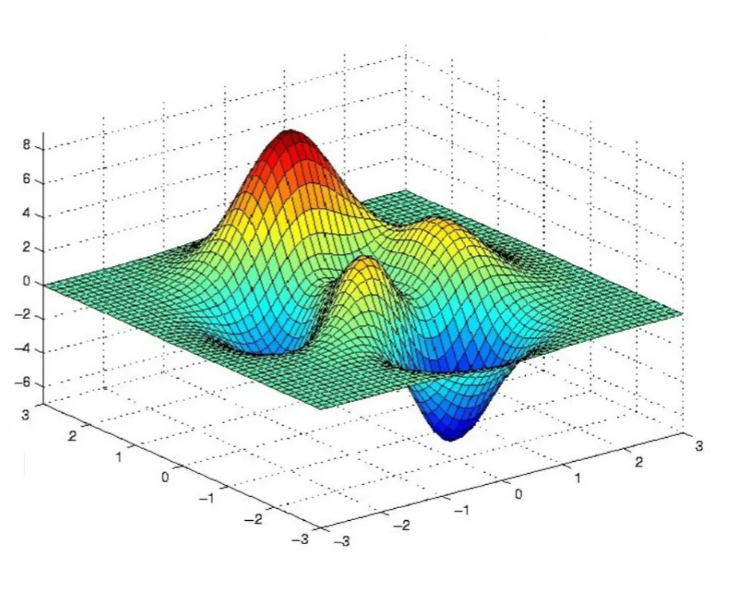


Figure Example of error function

Above is an example of an error function, that takes 1 parameter (1 weight) as x and returns an error as y. Errors may go up or down for different weight values. But here is a local minimum. There is also a global minimum.



The diagram above represents all errors for 2 weights. There are multiple options and its only for the size of 2 weights. Impossible to calculate all of them. There are multiple **local minimums**. The task is not to stack in the local minimum but try to find the **global minimum**.

Typically, a neural network model is trained using the **stochastic gradient descent optimization algorithm**. It is implemented in a way that tries to avoid local minimum.

**Loss function** or **cost function** is a function that minimizes error. It calculates error by comparing predicted output to target data. Main types of loss functions:

* Mean square error.
* Mean absolute error – it is similar to mean square error, but instead of square it uses absolute value.

# Neuron Networks Python libraries

**Keras** is a Python neural network library. <https://keras.io/>

Keras is built on top of **TensorFlow**. It has much simpler API that TensorFlow but is not as much flexible. Keras uses Tensorflow as its default backend, computational engine. Another backend used by Keras is Theano.

Keras was developed with a focus on enabling **fast experimentation**. Being able to go from idea to result with the least possible delay is key to doing good research.

**Sequential model** is used to build a plain stack of layers. It is good for building a standard Feed Forward NN.

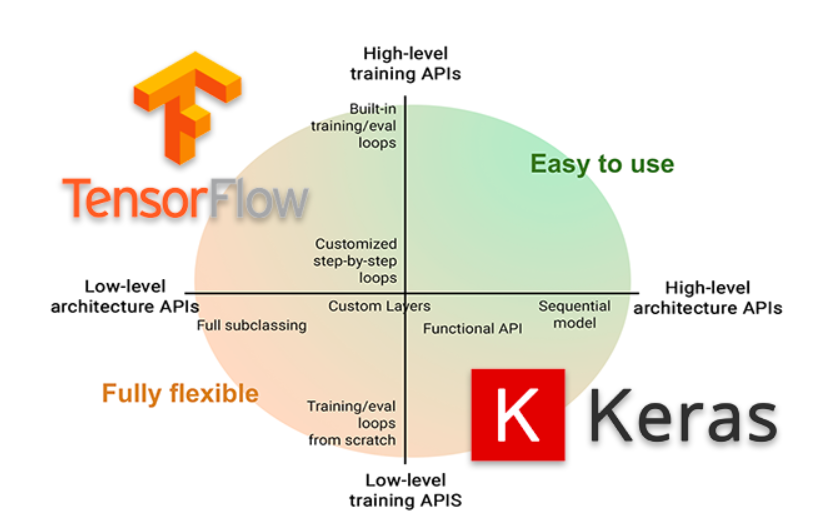
<https://keras.io/guides/sequential_model/>

**Dense layer** is s regular fully connected (dense) network layer.

Here you can find more about Dense layer:

* Dense layer: <https://medium.com/@hunterheidenreich/understanding-keras-dense-layers-2abadff9b990>
* Keras Dense Layer Video: <https://www.youtube.com/watch?v=ohgONsuoxVs>
* Keras Dense Layer: <https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/#1_Units>

**TensorFlow** has a particular focus on training and inference of deep neural networks <https://www.tensorflow.org/>



The diagram shows the relation between Keras and Tensorflow.

**Keras and TensorFlow**

Keras started as a separate project and is still available as a separate package.

In 2019 Keras was integrated into TensorFlow version 2 as a module. It is recommended to use Keras via TensorFlow.

<https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/>

I used Keras package in an example.