University of Virginia

Research Computing

TENSORFLOW – TURNING THE KNOBS

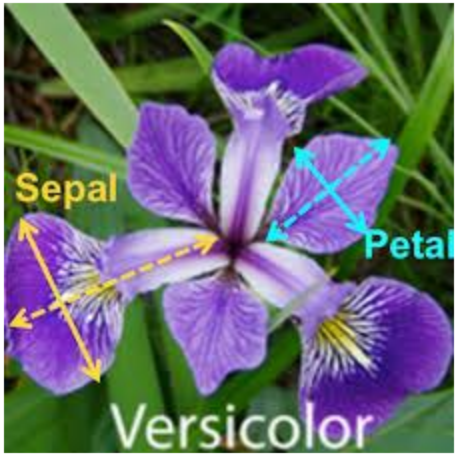
Jacalyn Huband

29 Nov. 2020

NEURAL NETWORKS FOR CLASSIFICATION

The Problem

Suppose we have a set of objects where we have measurements on several features and a label for what the object is.

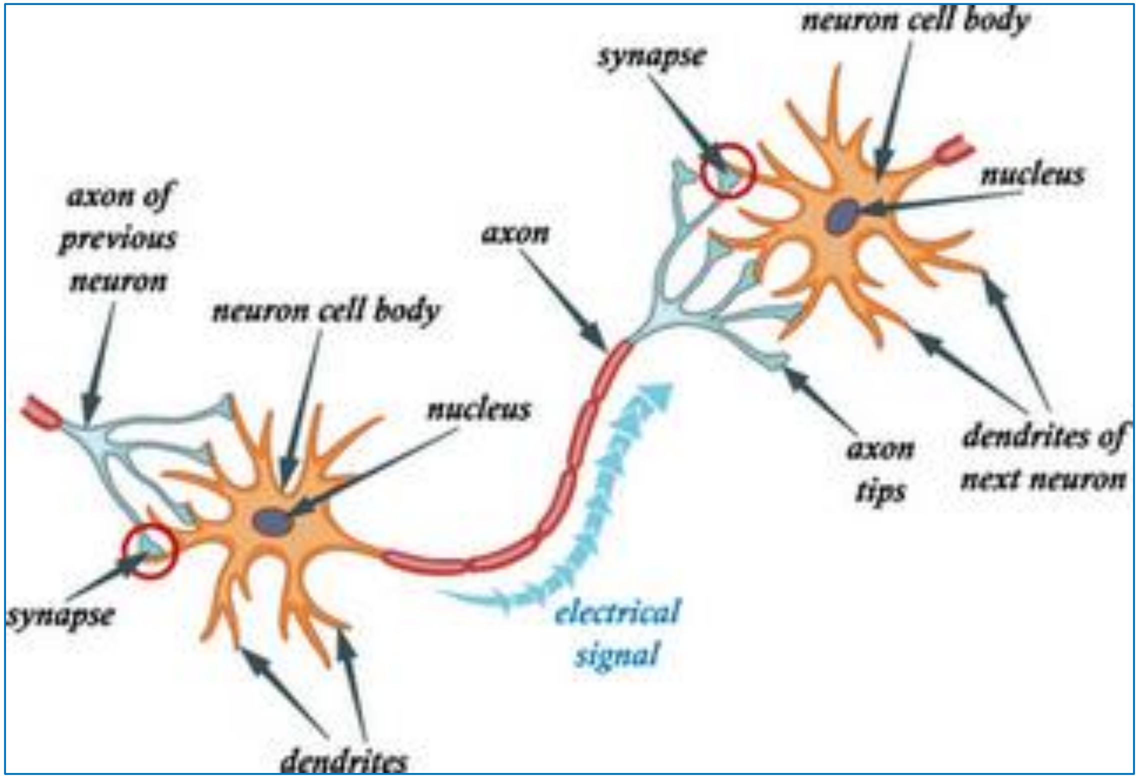


Can a computer "learn" from that data and predict the label for an unknown object?

Neural Network

Computational techniques that model the biology of the human brain for identifying objects or determining the curves that fit the data the best.

Neurons in the Brain

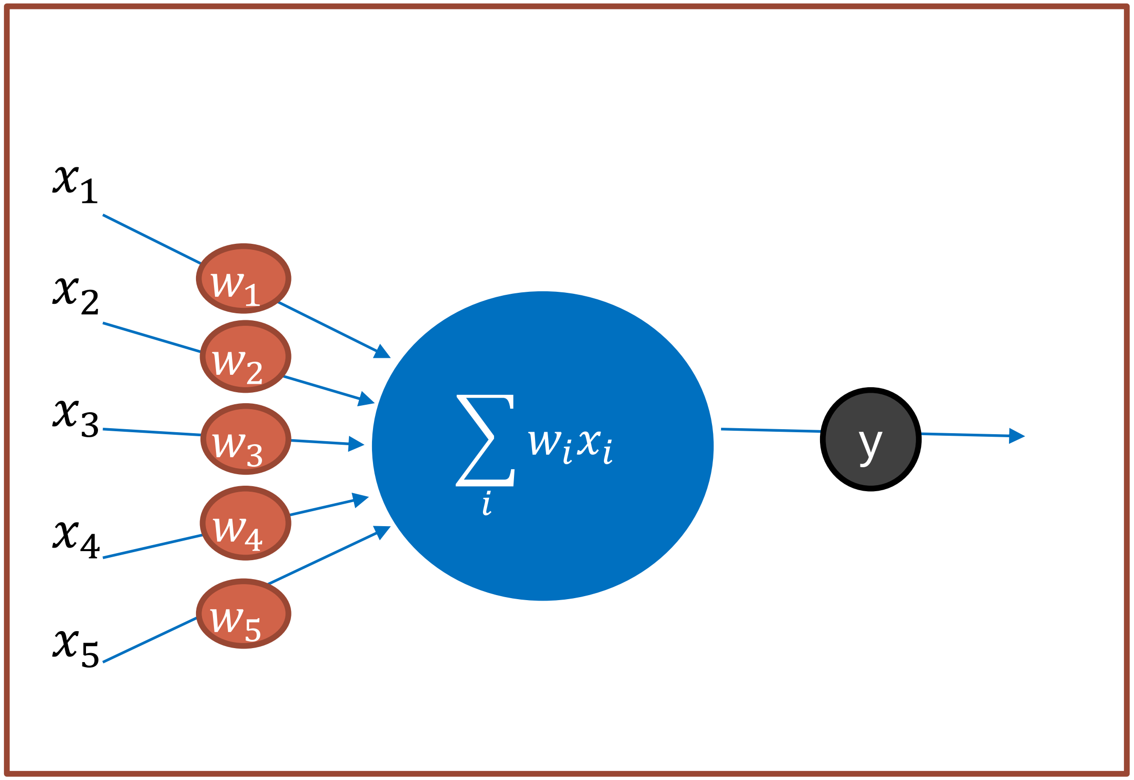


Neurons continuously receive signals, process the information, and fire out another signal.

The human brain has about 86 billion neurons, according to Dr. Suzana Herculano-Houzel

Diagram borrowed from
http://study.com/academy/lesson/synaptic-cleft-definition-function.html

Simulation of a Neuron

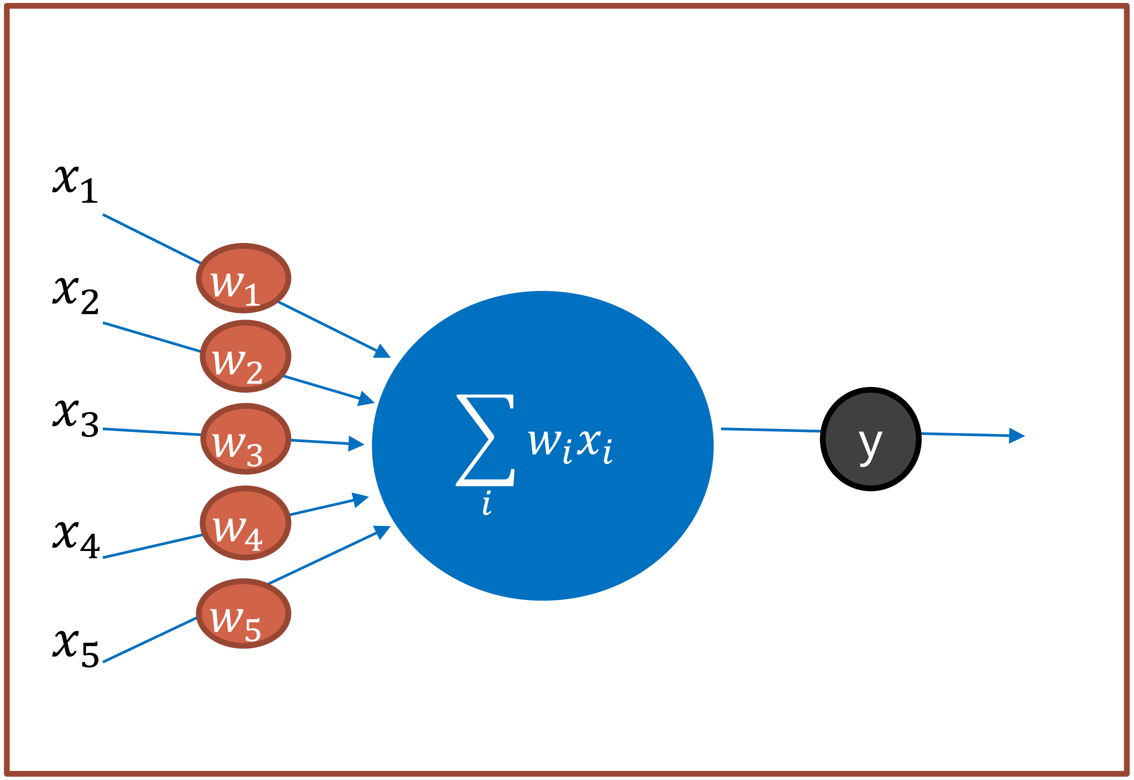


The "incoming signals" could be values from a data set(s).

A simple computation (like a weighted sum) is performed by the "nucleus".

The result, y, is "fired out".

Simulation of a Neuron



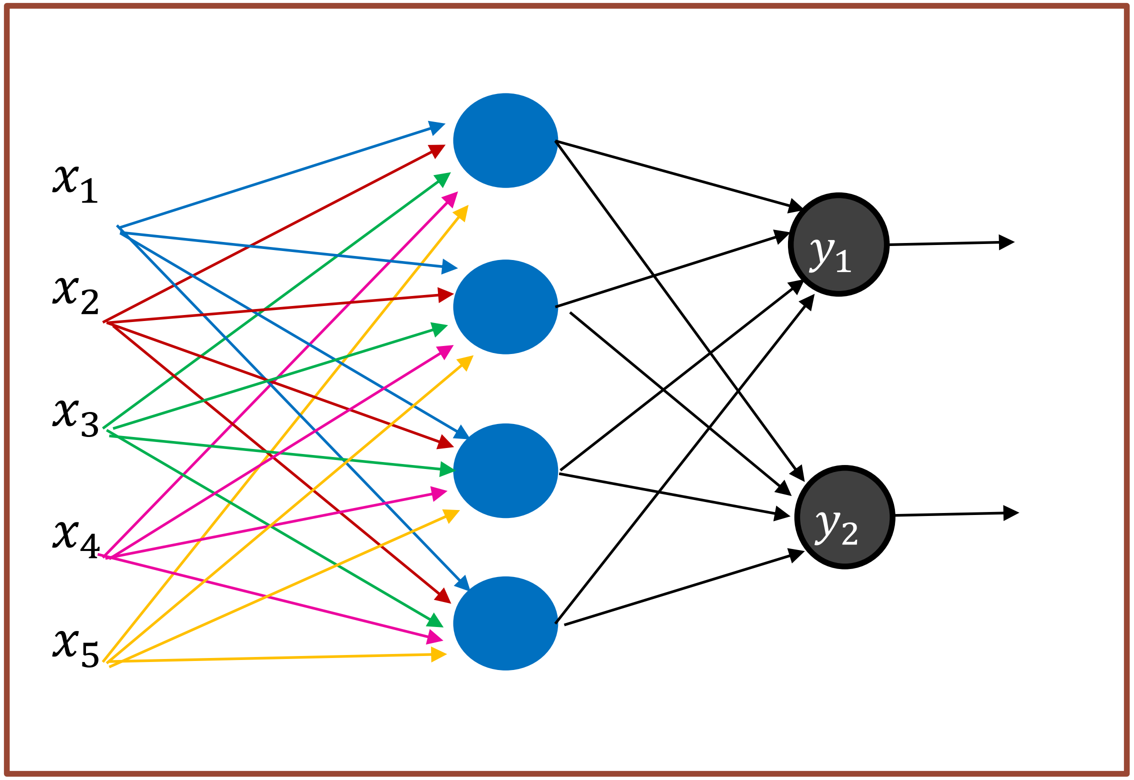
The weights, w_i, are not known.

During training, the "best" set of weights are determined that will generate a value close to y given a collection of inputs x_i.

Simulation of a Neuron

A single neuron does not provide much information (often times, a 0/1 value)

A Network of Neurons



Different computations with different weights can be performed to produce different outputs.

This is called a feedforward network because all values progress from the input to the output.

What is TensorFlow?

- A neural network that has many layers; an example of deep learning

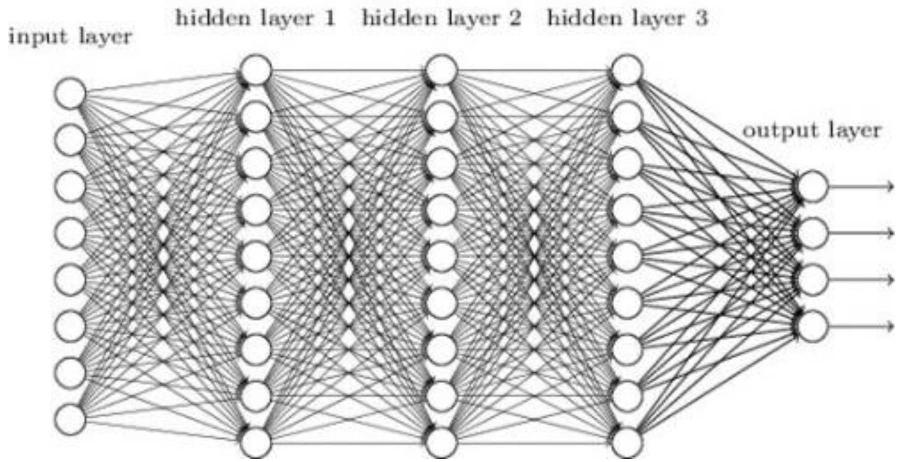


Image borrowed from:
http://www.kdnuggets.com/2017/05/deep-learning-big-deal.html

- A software library, developed by the Google Brain Team

This is great, but . . .

If we have a data set, how do we determine the right neural network (e.g., number of hidden layers, number of nodes, weights)?

The Art of Neural Networks . . .

- There are no set rules for designing your Tensorflow network.
- Often, we need to use trial-and-error to find a good design.
- Knowing what we can tweak in the design will help with the process.

The General Workflow

- The main step is training the network. Let's look at the big picture and drill down.

1. Start will a data set that has known labels/classifications.
2. Choose a model design (i.e., the number of hidden layers and hidden nodes that the network will have).
3. Run the inputs (i.e., the measurements) through the network to get the outputs. This is a ***forward feed*** step.
4. Compare the outputs with the known labels/classifications
5. Use the error in the outputs to work back through the network and tweak the weights. This is a ***back propagation*** step.
6. Repeat Steps 3 – 5 until we have an acceptable result.

The Model Design

- All neural network models will have three basic layers:
  - the input layer,
  - a hidden layer, and
  - an output layer.
- For small data sets, these three layers will be sufficient.
  - The input layer should have the same number of nodes as the number of features in the data.
  - The output layer should have the same number of nodes as the number of labels or categories.
  - That leaves the number of hidden layers and hidden nodes.

The Model Design: Hidden Layer

- For the number of hidden layer, start with just one hidden layer and observe the results. Add more layers if
  - The results are not accurate; and
  - The data set is relatively large – lots of observations and features.

- For the number of nodes in a hidden layer, you can
  - Choose a number between the number of nodes in the input layer and the number of nodes in the output layer; or
  - Use the formula:

$$N_h = \frac{N_s}{\alpha(N_i + N_o)}$$

where N_s is the size of the training sample, N_i is the number of nodes in the input layer, N_o is the number of nodes in the output layer, and α is a scaling factor between 2 and 10.

The Model Design Notes

- If there are too few hidden nodes, the model may *underfit* the data, causing large differences between the predicted classifications and the actual classifications.

- If there are too many hidden nodes, the model basically memorizes the training data.
  - This will give you very good results for the training data set.
  - But, the accuracy for the testing data set will not be good.

The Model Computation

- For the first forward feed, the model will need initial values for the weights (w_i). Although we can specify how we want the weights initialized, it may be better to let the algorithm randomly select the values.

- It determines whether the node should be activated (i.e., if the neuron should "fire" or not). We need a way for the algorithm to decide if a neuron should "fire" or not. This is achieved through an ***activation function***.

- Finally, for the training, we want to ensure that a small number of nodes are not dominating the entire model.
  - This is done by randomly dropping nodes to see how weights are adjusted.
  - Because the selection of nodes is random, we will only control the number of nodes to be dropped through a ***drop-out rate***. The value can range from 0.001 up to 100.

Model Computation: Activation Function

- Three commonly-used activation functions are
  - RELU (rectified linear unit activation) function. Basically, this function allows values greater than zero to pass through.
  - Sigmoid function. The sigmoid function looks like an S-shaped curve. It transforms small (or negative) values to values close to zero and large values to values close to 1.
  - Softmax function. This function is used in the output layer to convert the neural network result to probabilities for the given classifications.
- These functions are a small sampling of what is available. More activation functions are available at https://www.tensorflow.org/api_docs/python/tf/keras/activations

Model Training

- We need a function to determine the amount of error in predicted output of the model. This is done with a ***loss function***.

- We also need a function that will provide a technique for determining how to make corrections. This is done with an ***optimizer function***. However, we do not want large changes to be made to the values
  - Large changes could cause the model to oscillate between values being too big and being too small.
  - Instead, we want the algorithm to make small changes to the values.
  - We control this with a ***learning rate***.

Model Training: Loss Function

- As with the activation function, we have several options for the loss function. For classification problems, we, most often, will use the one of the following:
  - `binary_crossentropy` for classification models where the each outcome is one of two categories;
  - `categorical_crossentropy` for a classification models where the each outcome is one of three or more categories.
  - More loss functions are available at https://www.tensorflow.org/api_docs/python/tf/keras/losses

Model Training: Optimizer Function

- Again, there are several optimizer functions available. Two popular ones are Adam and Stochastic Gradient Descent.

- Adam -- an adaptive method that adjusts the learning rate based on the momentums associated with the descent.
  - You can use the default parameters. For example `keras.optimizers.Adam()`; or
  - You can provide your own values. For example: `keras.optimizers.Adam(lr=0.01, beta_1=0.95, beta_2=0.975)`

- Stochastic Gradient Descent -- updates the parameters in batches. There is a lot of fluctuation in the results, causing the algorithm to take longer to achieve a good result.
  - You can use the default parameters. For example: `keras.optimizers.SGD();` or
  - You can provide your own values. For example: `keras.optimizers.SGD(lr=0.01,momentum=0.09, decay=0.01,nesterov=True)`

- More loss functions are available at
  - https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

CODING A TENSOR FLOW

We will switch to a Jupyter Notebook.