Neural Networks: Structure

Estimated Time: 7 minutes

If you recall from the Feature Crosses unit (/machine-learning/crash-course/feature-crosses/video-lecture), the following classification problem is nonlinear:

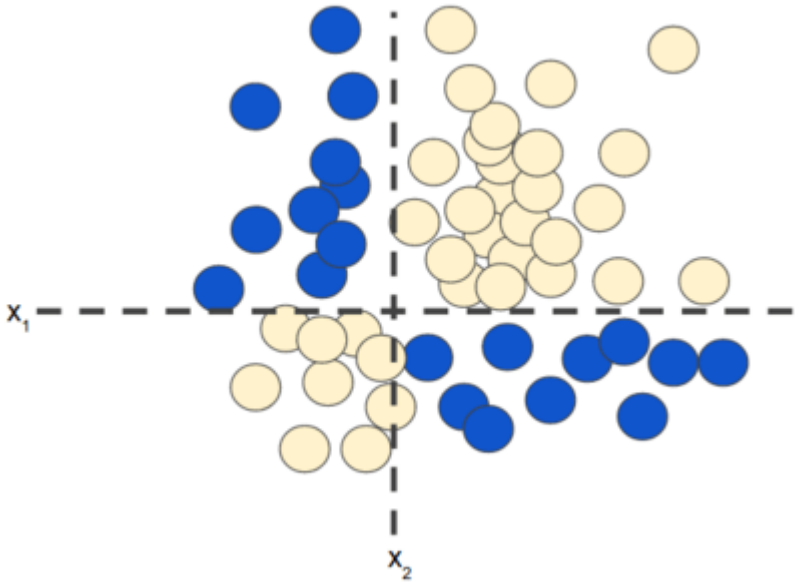

Figure 1. Nonlinear classification problem.

"Nonlinear" means that you can't accurately predict a label with a model of the form $b + w_1 x_1 + w_2 x_2$ In other words, the "decision surface" is not a line. Previously, we looked at feature crosses (/machine-learning/crash-course/feature-crosses/video-lecture) as one possible approach to modeling nonlinear problems.

Now consider the following data set:

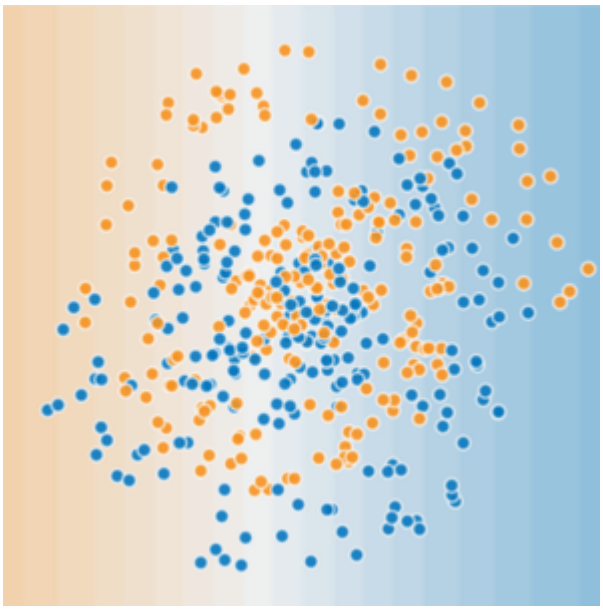

Figure 2. A more difficult nonlinear classification problem.

The data set shown in Figure 2 can't be solved with a linear model.

To see how neural networks might help with nonlinear problems, let's start by representing a linear model as a graph:

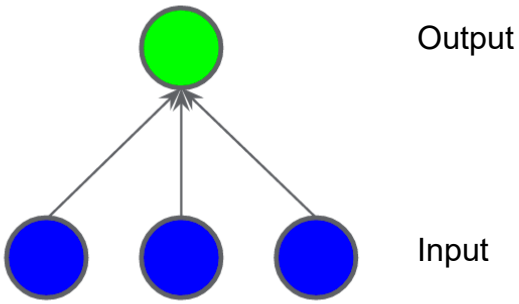

Figure 3. Linear model as graph.

Each blue circle represents an input feature, and the green circle represents the weighted sum of the inputs.

How can we alter this model to improve its ability to deal with nonlinear problems?

Hidden Layers

In the model represented by the following graph, we've added a "hidden layer" of intermediary values. Each yellow node in the hidden layer is a weighted sum of the blue

input node values. The output is a weighted sum of the yellow nodes.

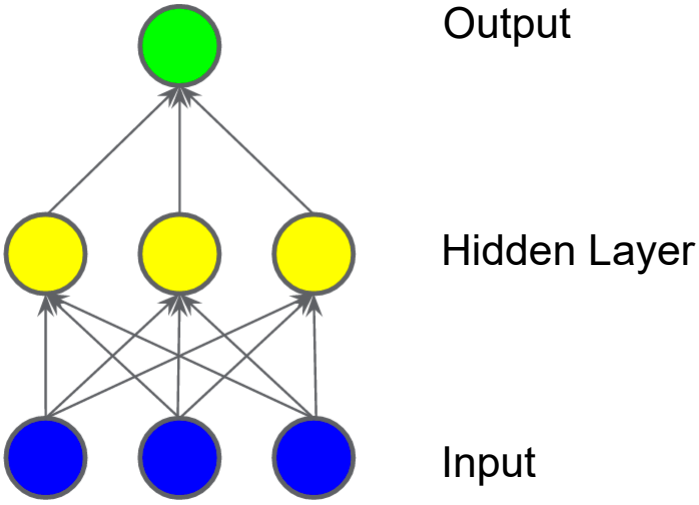

Figure 4. Graph of two-layer model.

Is this model linear? Yes—its output is still a linear combination of its inputs.

In the model represented by the following graph, we've added a second hidden layer of weighted sums.

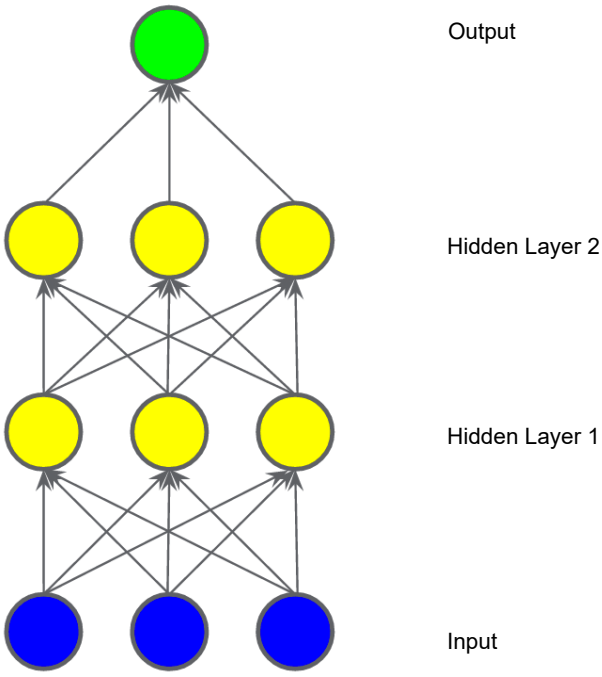

Figure 5. Graph of three-layer model.

Is this model still linear? Yes, it is. When you express the output as a function of the input and simplify, you get just another weighted sum of the inputs. This sum won't effectively

model the nonlinear problem in Figure 2.

Activation Functions

To model a nonlinear problem, we can directly introduce a nonlinearity. We can pipe each hidden layer node through a nonlinear function.

In the model represented by the following graph, the value of each node in Hidden Layer 1 is transformed by a nonlinear function before being passed on to the weighted sums of the next layer. This nonlinear function is called the activation function.

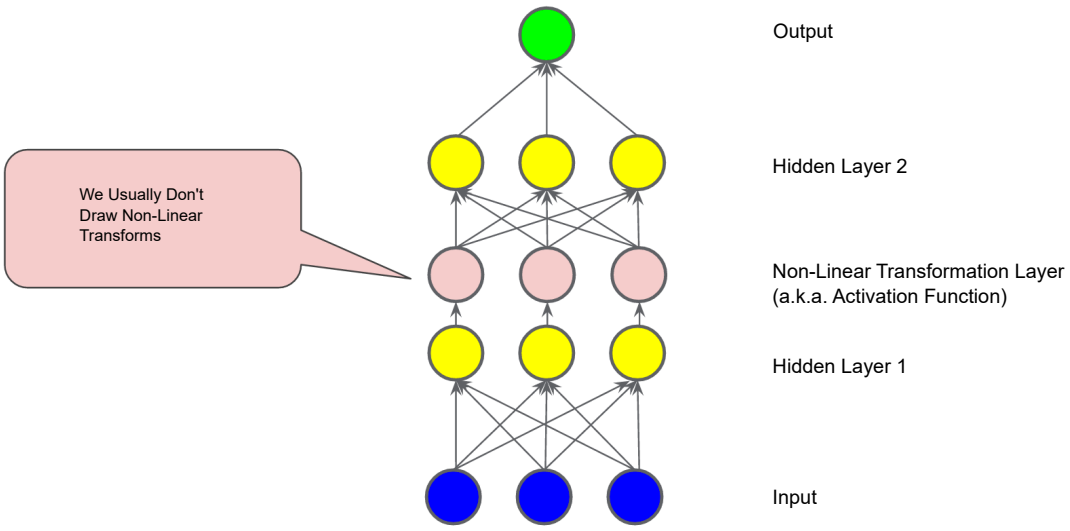

Figure 6. Graph of three-layer model with activation function.

Now that we've added an activation function, adding layers has more impact. Stacking nonlinearities on nonlinearities lets us model very complicated relationships between the inputs and the predicted outputs. In brief, each layer is effectively learning a more complex, higher-level function over the raw inputs. If you'd like to develop more intuition on how this works, see Chris Olah's excellent blog post (http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/).

Common Activation Functions

The following **sigmoid** activation function converts the weighted sum to a value between 0 and 1.

$$F(x) = \frac{1}{1 + e^{-x}}$$

Here's a plot:

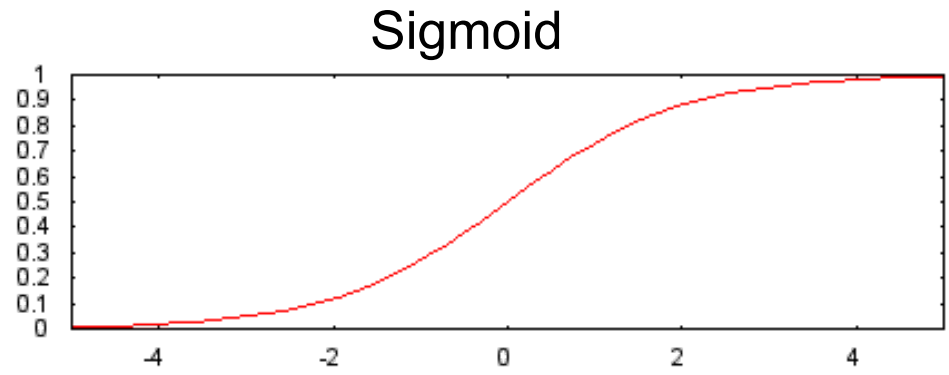

Figure 7. Sigmoid activation function.

The following **rectified linear unit** activation function (or **ReLU**, for short) often works a little better than a smooth function like the sigmoid, while also being significantly easier to compute.

$$F(x) = max(0, x)$$

The superiority of ReLU is based on empirical findings, probably driven by ReLU having a more useful range of responsiveness. A sigmoid's responsiveness falls off relatively quickly on both sides.

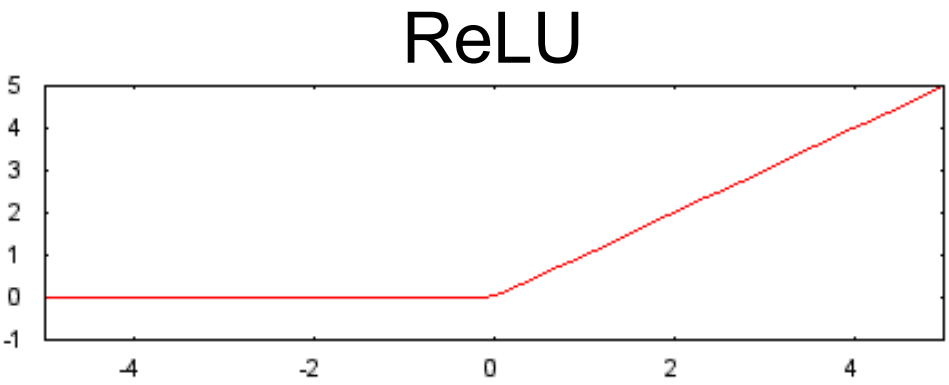

Figure 8. ReLU activation function.

In fact, any mathematical function can serve as an activation function. Suppose that σ represents our activation function (Relu, Sigmoid, or whatever). Consequently, the value of a node in the network is given by the following formula:

$$\sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

TensorFlow provides out-of-the-box support for many activation functions. You can find these activation functions within TensorFlow's list of wrappers for primitive neural network operations (https://www.tensorflow.org/api_docs/python/tf/nn). That said, we still recommend starting with ReLU.

Summary

Now our model has all the standard components of what people usually mean when they say "neural network":

- A set of nodes, analogous to neurons, organized in layers.
- A set of weights representing the connections between each neural network layer and the layer beneath it. The layer beneath may be another neural network layer, or some other kind of layer.
- A set of biases, one for each node.
- An activation function that transforms the output of each node in a layer. Different layers may have different activation functions.

A caveat: neural networks aren't necessarily always better than feature crosses, but neural networks do offer a flexible alternative that works well in many cases.

Key Terms

- activation function (/machine-learning/glossary#activation_function)
- hidden layer (/machine-learning/glossary#hidden_layer)
- neural network (/machine-learning/glossary#neural_network)
- neuron (/machine-learning/glossary#neuron)
- rectified linear unit (ReLU) (/machine-learning/glossary#ReLU)
- sigmoid function (/machine-learning/glossary#sigmoid_function)

Help Center (https://support.google.com/machinelearningeducation)

Previous

← Video Lecture
(/machine-learning/crash-course/introduction-to-neural-networks/video-lecture)

Next
Playground Exercises →
(/machine-learning/crash-course/introduction-to-neural-networks/playground-exercises)