As with all machine learning algorithms, neural networks are not perfect and will often underperform using a basic implementation. When a neural network model does not meet performance expectations, it is usually due to one of two causes: inadequate or inappropriate model design for a given dataset, or insufficient or ineffective training data. Although collecting more training/test data is almost always beneficial, it may be impossible due to budget or logistical limitations. Therefore, the most straightforward means of improving neural network performance is tweaking the model design and parameters.

When it comes to tweaking a neural network model, a little can go a long way. If we tweak too many design aspects and parameters at once, we can cause a model to become less effective without a means of understanding why. To avoid trapping ourselves in endless optimization iterations, we can use characteristics of our input data to determine what parameters should be changed.

There are a few means of optimizing a neural network:

* Check out your input dataset.
* Add more neurons to a hidden layer.
* Add additional hidden layers.
* Use a different activation function for the hidden layers.
* Add additional epochs to the training regimen.
* **Check Out Your Input Dataset**
* Before you start down the endless journey of model optimization, it is always a good idea to check the input data and ensure that there are no variables or set of outliers that are causing the model to be confused. Although neural networks are tolerant of noisy characteristics in a dataset, neural networks can learn bad habits (like the brain does). Even if we standardize and scale our numerical variables, too many outliers in a single variable can lead to performance issues.

**Add More Neurons and Hidden Layers**

**REWIND**

Previously, we explored how to optimize a neural network by adding neurons to the hidden layer. Adding neurons to a hidden layer has diminishing returns—more neurons means more data as well as a risk to overfitting the model.

Instead of adding more neurons, we could change the structure of the model by adding additional hidden layers, which allows neurons to train on activated input values, instead of looking at new training data. Therefore, a neural network with multiple layers can identify nonlinear characteristics of the input data without requiring more input data.

**IMPORTANT**

This concept of a multiple-layered neural network is known as a **deep learning neural network**. We'll be exploring deep learning neural networks in greater detail later in the module.

**Use a Different Activation Function**

Another strategy to increase performance of a neural network is to change the activation function used across hidden layers. Depending on the shape and dimensionality of the input data, one activation function may focus on specific characteristics of the input values, while another activation function may focus on others.

It is important to use an activation function that matches the complexity of the input data. If we wanted to rank the four most-used activation functions by data complexity and ideal use case, the order would be as follows:

1. The **sigmoid function** values are normalized to a probability between 0 and 1, which is ideal for binary classification.
2. The **tanh function** can be used for classification or regression, and it expands the range between -1 and 1.
3. The **ReLU** **function** is ideal for looking at positive nonlinear input data for classification or regression.
4. The **Leaky ReLU function** is a good alternative for nonlinear input data with many negative inputs.

**NOTE**

By default, the Keras Dense layer will implement the **linear** activation function, which means that the net sum value is not transformed. In other words:

α ( x ) = x

The linear activation function limits the neural network model to only perform a linear regression. Therefore, the linear activation function is only appropriate for an output layer.

To experiment and optimize using an activation function, try selecting from activation functions that are slightly more complex than your current activation function. For example, if you were trying to build a regression neural network model using a wide input dataset, you might start with a tanh activation function. To optimize the regression model, try training with the ReLU activation function, or even the Leaky ReLU activation function. In most cases, it is better to try optimizing using a higher complexity activation function rather than a lower complexity activation function. Using a higher complexity activation function will assess the input data differently without any risk of censoring or ignoring lower complexity features.

**Add Additional Epochs to Training Regimen**

If your model still requires optimizations and tweaking to meet desired performance, you can increase the number of epochs, or training iterations. As the number of epochs increases, so does the amount of information provided to each neuron. By providing each neuron more information from the input data, the neurons are more likely to apply more effective weight coefficients. Adding more epochs to the training parameters is not a perfect solution—if the model produces weight coefficients that are too effective, there is an increased risk of model overfitting. Therefore, models should be tested and evaluated each time the number of epochs are increased to reduce the risk of overfitting.

As with all machine learning models, creating an ideal classification or regression model is part mathematics and part art. As we design more and more models, optimizing and fine-tuning becomes less trial and error and more pattern recognition.