Extracted from:

Machine Learning in Elixir

Learning to Learn with Nx and Axon

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Your Elixir Source

Machine Learning in Elixir

Learning to Learn with Nx and Axon

Sean Moriarity

Edited by Tammy Coron

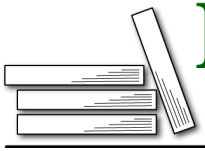
Machine Learning in Elixir

Learning to Learn with Nx and Axon

Sean Moriarity

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 979-8-88865-034-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—July 19, 2023

Breaking Down a Neural Network

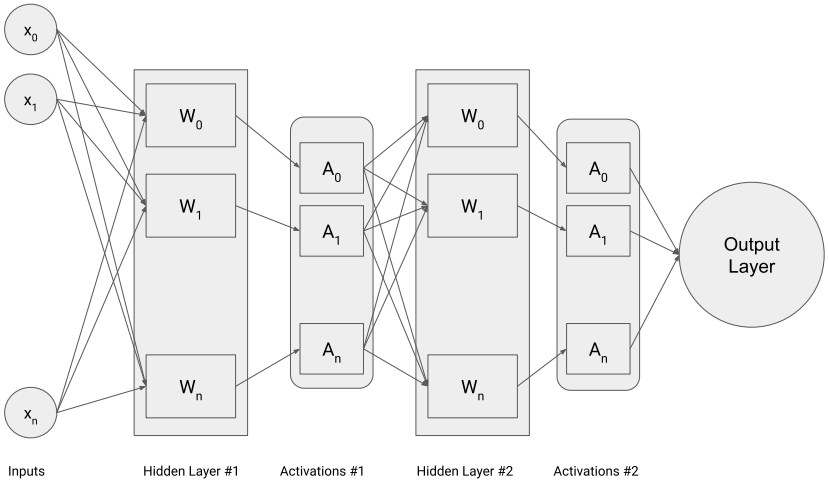
With an understanding of why neural networks are necessary, and why they're so powerful, it's time to dive into the question of what is a neural network. In this section, you'll break down the vocabulary surrounding deep learning, the anatomy of neural networks, and what a neural network actually looks like in Elixir. Understanding the building blocks of neural networks will help build your intuition as you start to use Axon.

Getting the Terminology Right

Deep models, neural networks, artificial neural networks (ANNs), multi-layer perceptrons (MLPs)—if you've spent some time reading about deep learning, you likely have encountered all of these terms used almost interchangeably. The terminology of deep learning can be more daunting than implementing the models themselves.

Deep learning refers to a subset of machine learning algorithms that make use of deep models, or artificial neural networks. These models are considered "deep" as opposed to other models with respect to their layers of successive computation. If you roll out the computation graph of operations that take place in a deep model, the computation graph would appear deep (e.g., lots of operations). You can also consider models deep with respect to the number of intermediate layers—with each successive layer increasing the depth of the model.

Artificial neural networks (ANNs) are one term for deep models. The term artificial neural network probably invokes the thought of images similar to this:



ANNs are named for their brain-inspired design. The transformation of inputs in an ANN is meant to simulate the firing of neurons passing information around the brain. The usage of the term ANN is probably a bit of a misnomer, as there is little evidence to suggest the brain works in the same way that neural networks do.

Multi-layer perceptrons (MLPs) are a class of deep learning models that make use of *fully connected layers* or *densely connected layers*. You'll see what this means in The Anatomy of a Neural Network, on page 5. All you need to know is that MLPs are a specific class or *architecture* of neural networks. You might also see them referred to as *feedforward networks* because information flows from previous layers forward toward output layers. There are many other architecture types you'll implement in this book, including Convolutional Neural Networks (CNNs) in Chapter 7, Learn to See, on page ?, Recurrent Neural Networks (RNNs) in *the (as yet) unwritten Chapter 9, Understand Text,* , and Generative Adversarial Networks (GANs) in *the (as yet) unwritten Chapter 12, Learn without Supervision,* .

The Rebranding

Before deep learning had its watershed moment in 2012, the field was led by a relatively small number of researchers. Most top machine learning conferences would accept only one or two deep learning papers per year—if they accepted any. At the time, researchers working on deep learning were referred to as "connectionists," owing to the connections between layers when visualizing

The Rebranding

deep models. Deep learning came about as a strategic rebranding by connectionists in an attempt to overcome the bias against neural networks at the time.

The Anatomy of a Neural Network

Most neural networks can be simplified down to a few key components. The most common abstraction for a unit of a computation or work in a neural network is a layer. Typically, a layer represents a transformation of the input which is to be forwarded to the next layer. The number of layers in the model is typically referred to as the *depth* of the model. Generally, increasing the depth of the model also increases the capacity of the model. However, at a certain point, making a model too deep can hinder the learning process.

There are many different types of layers you'll use throughout the rest of this book. In a neural network, you'll generally have three classes of layers: *input layers*, *hidden layers*, and *output layers*.

Input Layers

Input layers are really just placeholders for model inputs. Certain operations on a neural network require a known input shape. You can refer back to Chapter 2, Get Comfortable with Nx, on page ? to get a better idea of how certain real-world data maps to tensor inputs.

Hidden Layers

Hidden layers are where the magic happens in a neural network. Hidden layers are intermediate layers of computation which transform the input into a useful representation for the output layer. They are the additional conductors in a very large orchestra, which make high-dimensional inputs manageable for the output layer.

The most common hidden layer is the densely connected, *fully connected*, or simply *dense layer*. The dense layer is named for the dense connections it creates between two layers—in other words, every input to a dense layer maps to an output in the dense layer. Dense layers have a number of output *units*, which represents the dimensionality of the dense layer. If you like the analogy of neural networks to the brain, you can think of an individual unit as a *neuron*. A dense layer with 128 units has 128 neurons.

The number of units in a dense layer is referred to as the *width* of the layer. Wider dense layers have more representational capacity. However, there is

also a point of diminishing returns. It's common to use hidden widths that are multiples of two. This is because of how memory layouts work on modern accelerators.

Mathematically, dense layers are just matrix multiplications or linear transformations. Dense layers learn to project inputs in such a way that extracts a useful representation for successive layers.

Activations

Hidden layers often times have an *activation* function that applies a nonlinear function to the output. The introduction of nonlinearities into the neural network are what makes it a universal approximator. It's common to use activation functions that scale or squeeze inputs into some useful output range. For example, the sigmoid function is often used as an activation because it squeezes outputs between 0 and 1. Because neural networks are trained with gradient descent, it's important that activation functions be differentiable.

You can think of activation functions as a means of signaling certain input features. For example, your neural network might learn to only have certain neurons firing on certain input features. A neuron's activation can be interpreted as its importance to the final output. Some neurons that are not important will be entirely "turned off."

There are a number of activation functions you can use in a neural network. Finding better activation functions is a popular area of research. The activation functions you should be familiar with are ReLU, sigmoid, and softmax.

ReLU

The Rectified Linear Unit (ReLU) activation function is a very popular intermediate activation that computes the function:

```
defn relu(x) do
  Nx.max(0, x)
end
```

ReLU takes all negative inputs to 0, and maps positive inputs to the same value.

Sigmoid

The sigmoid activation function is a popular output activation because it squeezes outputs to the range 0-1. It computes the logistic sigmoid function:

```
defn sigmoid(x) do
  1 / (1 + Nx.exp(-x))
```

```
end
```

The sigmoid function is especially useful when you're trying to compute an output probability between 0 and 1.

Softmax

The softmax function is a popular output activation for multi-class classification problems. It outputs a categorical probability distribution.

Output Layers

From an implementation perspective, output layers are no different than input layers. Output layers are the final result of your neural network. After transforming your inputs into useful representations with hidden layers, output layers transform those representations into something you can meaningfully use or interpret, such as a probability.

For classification problems, it's common to use a dense layer with a sigmoid or softmax activation as the final output layer. For binary classification problems, the final layer will usually be a dense layer with one output unit and sigmoid activation. For multi-class classification problems, the final layer will usually be a dense layer with `N` output units and softmax activation, where `N` is the number of possible classes.

For scalar regression problems, it's common to use a dense layer with one output unit and no activation—so the output neuron just maps to a scalar.

The form of an output layer is very problem dependent. You'll see lots of different output layers throughout the rest of this book.