My Comprehensive Evaluation

A Comprehensive Evaluation Report
Presented to
The Statistics Faculty
Amherst College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts
in
Statistics

Jenn Halbleib

February 2018

Acknowledgements

"Todd, trust math. As in Matics, Math E. First-order predicate logic. Never fail you. Quantities and their relation. Rates of change. The vital statistics of God or equivalent. When all else fails. When the boulder's slid all the way back to the bottom. When the headless are blaming. When you do not know your way about. You can fall back and regroup around math. Whose truth is deductive truth. Independent of sense or emotionality. The syllogism. The identity. Modus Tollens. Transitivity. Heaven's theme song. The nightlight on life's dark wall, late at night. Heaven's recipe book. The hydrogen spiral. The methane, ammonia, H_2O. Nucleic acids. A and G, T and C. The creeping inevibatility. Caius is mortal. Math is not mortal. What it is is: listen: it's true." -David Foster Wallace

Before coming to Amherst, I had the privledge of learning from several incredible community college instructors who imbued me with a love for the art of mathematics. Jonathan Wherry, my statistics instuctor and honors project advisor, managed to teach me methods of probability before I understood integrals. Steve Simonds told me some of the best math jokes with great enthusiam during a year of night calculus courses. These men fortified my resolve to study math and rose to the challenge of teaching a precocious, talented student in an environment filled with all levels of learners. I also owe a great debt to my first term english instructor, Martha Henning, who pulled me into her office and demanded that I apply to liberal arts schools.

Additionally, I offer many thanks to the entire Amherst college statistics department. The program at Amherst is beyond compare, especially in forming students into independent learners and in creating statistical computing skills. I especially want to thank Professor Horton for encouraging, rallying, and supporting me during my time at Amherst. The experience has been, to say the least, transformative. Outside our department, I'd like to thank Professor Judy Frank for giving me the space and encouragement to write fiction while also studying high level mathematics. She constantly inspires me to write and live more authentically. Finally, I'd like to acknowledge David Foster Wallace for writing beautiful novels that captivated me to the point of obsession. In reading his biography, I discovered Amherst for the first time and truly, I never would have applied here or known about the magic of this place had it not been for him, however indirectly. I've been living *all* of my wildest dreams at Amherst and I am beyond grateful for this experience.

Table of Contents

List of Figures

Abstract

This paper describes the mathematics essential to neural network fitting and demonstrates how to implement a neural network on the MNIST data set in TensorFlow using the Keras package in R.

Introduction

This project seeks to lay out the mathematical and statistical underpinnings for modern neural networks. In image and video classification problems, the performance of neural networks is unparalleled. But, like other machine learning methods, neural networks can prove difficult to interpret. This paper will demystify why neural networks perform so well.

Additionally, the use of TensorFlow, an advanced, open-source machine learning library from Google will be introduced. TensorFlow offers users the ability to code neural networks from a high level while receiving the benefits of C++ run-time. This paper will demonstrate how to create a TensorFlow neural network to make predictions with the quintessential MNIST digits dataset. After working this exercise, readers should feel enabled to begin exploring the TensorFlow library for more advanced tasks.

0.1 Motivation

Approaching the comps project, I focused on the prevalence of machine learning in modern data science. I wanted to use the comps project as an opportunity to show that I can approach a novel machine learning method, understand its theoretical underpinnings, and implement the method in a productive way.

Chapter 1

Understanding Neural Networks

In this section, I will provide a detailed explanation and notation for the functioning of a *feed-forward neural network*. All neural network schemes follow this basic pattern, with some changes in iteration structures.

1.1 Visualizing a Neural Network

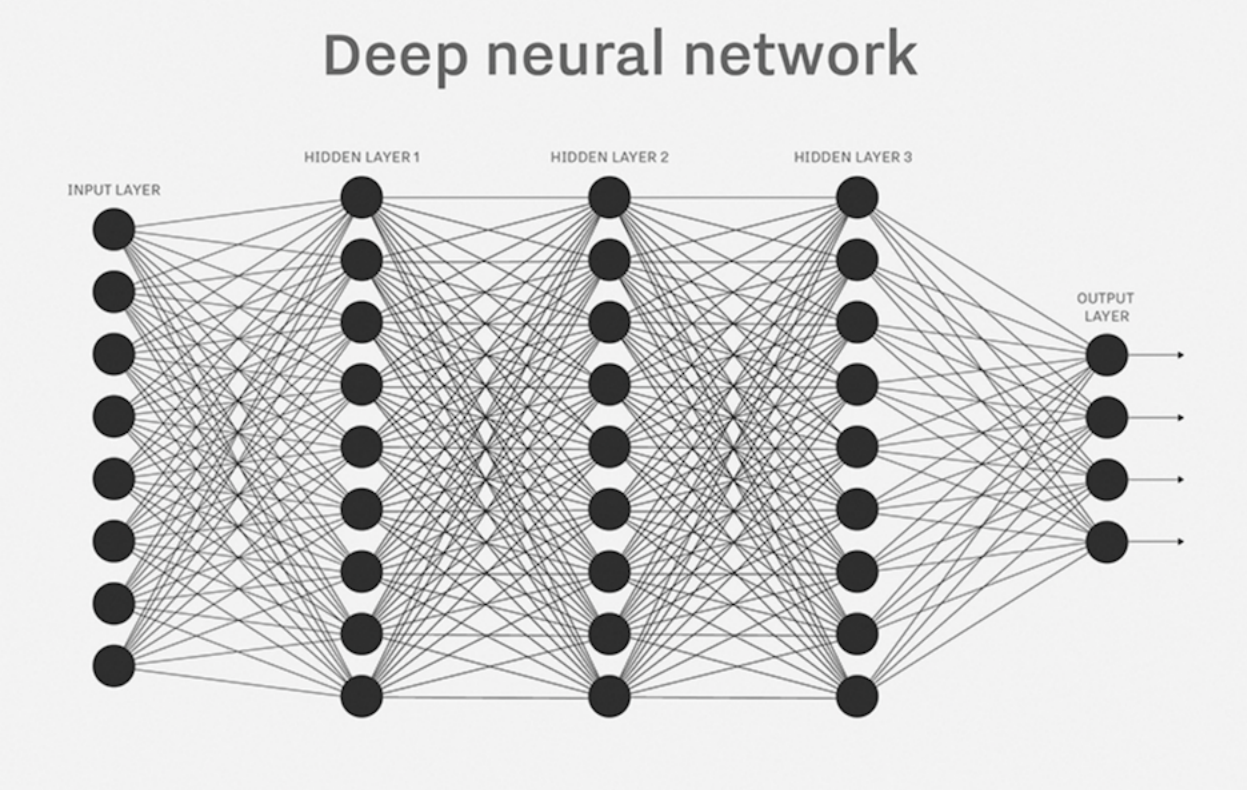


Figure 1.1: Feed Forward Neural Network with 3 Hidden Layers (Rouse, n.d.)

Looking at a neural network graph like the one displayed above for the first time, a classically trained statistician may feel a sort of vertigo. This complicated collection of graphing elements doesn't look like a typical statistical model. And yet, at its core, a neural network doesn't stray far from the tools statisticians are used to. The "Input Layer" corresponding to the first set of nodes (which are referred to as neurons in the language of neural networks) in Figure 1.1 is simply the set of $\mathbf{X} = x_1, x_2, x_3, ..., x_n$ model inputs. (Hastie and Friedman 2013) For example, if I was making a model to detect risk of default for loan applicants at a bank, I might have $x_1 = \textit{credit score}$, $x_2 = \textit{income}$, $x_3 = \textit{age}$, etc. The "Output Layer" corresponding to the final set of neurons in Figure 1.1 returns a prediction for model output $y_m|\mathbf{X}$. In the banking example, the output might be a probability score in the interval $[0, 1]$ that corresponds to the likelihood of default for a particular applicant. In the "Hidden Layer" neurons, data is fed in from the left neurons, a function is applied to the data, and the results of that function are passed into the neurons of the next layer. (352) In this way, a

neural network, in simplest form, amounts to a complex, highly-parametrized version of logistic regression, performed as a supervised learning task. (352)

1.2 Network basics

Neural networks have three types of neurons composing the three types of layer: Input, Hidden, and Output. As mentioned previously, the Input neurons simply contain the untransformed input data for the neural net. What comes next gets notation heavy, so I'll summarize in words what we're going to set-up first. From layer to layer, a set of weights (i.e. a weight vector) is applied to the inputs from the previous layer before being passed into the next layer. (Hastie and Friedman 2013) The weight vector includes an intercept term called the *bias*. (352) In each hidden layer, a function is applied to the linear combination of the input and the weight vector. (352) The result of this function transformation passes to the next layer until the output layer is reached. (352) In the output layer, a decision function makes the final prediction or assigns the final predicted probabilities. (352)

Define the input layer.

$$(1)\ \mathbf{X} = \{1, x_1, x_2, x_3, ..., x_n\}\ \forall\ X_m$$

$$where\ n = \ number\ of\ variables,\ m = number\ of\ observations$$

.

The constant 1 in the vector is a function of the linear algebra occurring between layers. (Hastie and Friedman 2013) Most books and papers on Neural Networks avoid using this convention, since programs for implementing these methods add the 1 automatically meaning it requires no attention from the user. (Chollet and Allaire 2018)

Define the weight vector, applied "between" layers.

$$(2)\ \mathbf{W} = \{w_0, w_1, w_2, w_3, ..., w_n\}\ \forall$$

$$W_m\ where\ n = \ number\ of\ variables\ in\ x,\ m = \ number\ of\ observations\ in\ x$$

Notice the **W** index begins at 0. This intercept term, which the 1 in **X** picks up, is referred to as the *bias*. The remaining terms are weights. (Hastie and Friedman 2013)

Define a linear combination of the inputs and weights.

$$(3) g(x_m, w) = w_0 + \Sigma_{i=1}^{n} w_i x_{mi} = \mathbf{XW}$$

Notice, under this convention, $\mathbf{W}$ is a column vector, multiplied over the rows (observations) of $\mathbf{X}$. $g(x_m, w)$ occurs in the "space" between layers, transforming the inputs to each neuron by creating a weighted sum. (Hastie and Friedman 2013)

Define a neuron function.

$$(4)\ f(g) = f(\ x_m * w) = y_m$$

This function applies a transformation that generates a "score" or an indicator I for this observation, which is passed out of the neuron. (Chollet and Allaire 2018) (Note, a discussion of possible functions used for the $f(g)$ transformation is included in §1.3.)

$$(5)\ Define\ l = \ number\ of\ layers$$

To generalize these equations, define the vector of total results $\mathbf{Y}$ from layer $l-1$ equal to $\mathbf{X}$ for each layer l. In short, as the network moves forward, the results (Y) become the inputs for the next layer. The weight vector must also be referred to as W_l, since each layer receives its own set of weights. (Hastie and Friedman 2013) The function $f(g)$ may be the same or different in every layer. (355)

$$(6)\ F(g) = final\ output\ classification\ function$$

In the output nodes, a function applies a classification to each observation in the data set. (Hastie and Friedman 2013) The classification function may output a direct selection, meaning a 1 for the predicted outcome and 0 for all other outcomes or may output a probability vector which sums to 1, giving the likelihood that a particular observation falls into any of the possible categories. (355) The output layer has nodes equal to the number of possible predicted outcomes. (355)

1.3 Neuron Functions

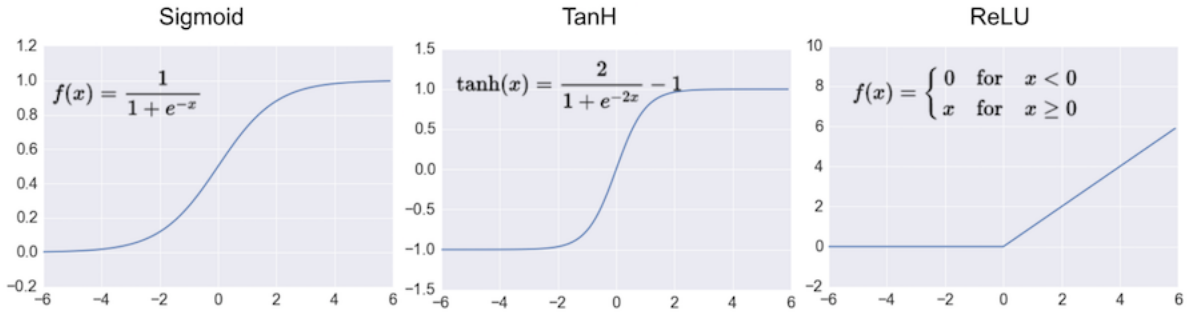


Figure 1.2: Common Neuron Functions (Moujahid 2016)

The functions employed for $f(g)$ in the hidden layers apply non-linearities.(Hastie and Friedman 2013) Figure 1.2 shows some of the most common functions. (Note, in the figure, x has been used, instead of g, as in the notation from §1.2.) Statisticians will recognize the sigmoid function from logistic regression. The overall purpose of these functions is to synthesize the weighted inputs into a single value. In practice, the choice of non-linearity is often the result of trial and error, with the sigmoid function serving as the most common default.(Chollet and Allaire 2018) However, all of these functions have one commonality: they are differentiable, which is required for optimization of the neural network. (41)

1.4 Fitting the Neural Net: Optimization Through Stochastic Gradient

As with all supervised learning methods, neural networks optimize over a loss function. Loss function choices vary, with Log Loss and Root Mean Squared Error serving as typical stand-by choices.

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L[g(x, w), y] \right\}$$

In this equation, L is the loss function chosen for the network, with inputs g, the network outputs, and y, the training set values. (Hastie and Friedman 2013)

To initialize the network, all weights are set randomly. (Chollet and Allaire 2018) Then, a gradient descent algorithm is iterated to optimize over the weights. (41) Gradient descent uses partial derivatives of L with respect to (x, w) to move the

weights in the direction of local minima of L. (43) By iterating this process over the network repeatedly, the weight vector is tuned in the direction of the global minimum of L. (43)

In practice, neural networks can have significant computational costs. (Chollet and Allaire 2018) For this reason, a stochastic version of gradient descent is used. (44) From each iteration, a small batch of samples is drawn and the gradient is calculated for the sample. (44) The results are used to adjust all the weights in the network. (44)

More complicated versions of stochastic gradient descent involve weighting or normalizing the current gradient calculation using the most recent or several of the most recent gradients. (Chollet and Allaire 2018) These methods can help control for problems that arise with local minima in the loss function, which can cause the network to "stick" at a non-optimal set of weights. (45)

Chapter 2

Hello World

“You can think of ‘solving’ MNIST as the”hello world" of deep learning–it’s what you do to verify that your algorithms are working as expected."
-Francois Chollet

Figure 2.1: Digits from MNIST (LeCun, Cortes, and Burges, n.d.)

In this chapter, we’ll learn how to implement a neural net to classify handwritten digits from the MNIST data set in R using a TensorFlow neural net constructed using the Keras package. The MNIST data set contains thousands of handwritten digits, divided into a testing and a training set. (Yann LeCun, Cortes, and Burges, n.d.) Figure 2.1 displays some of the MNIST digits.

2.1 Keras and TensorFlow

TensorFlow is an opensource computational library created by Google researchers specifically for machine learning that can be executed as a flow graph. In this way, TensorFlow is optimized for building neural nets, even complex ones.

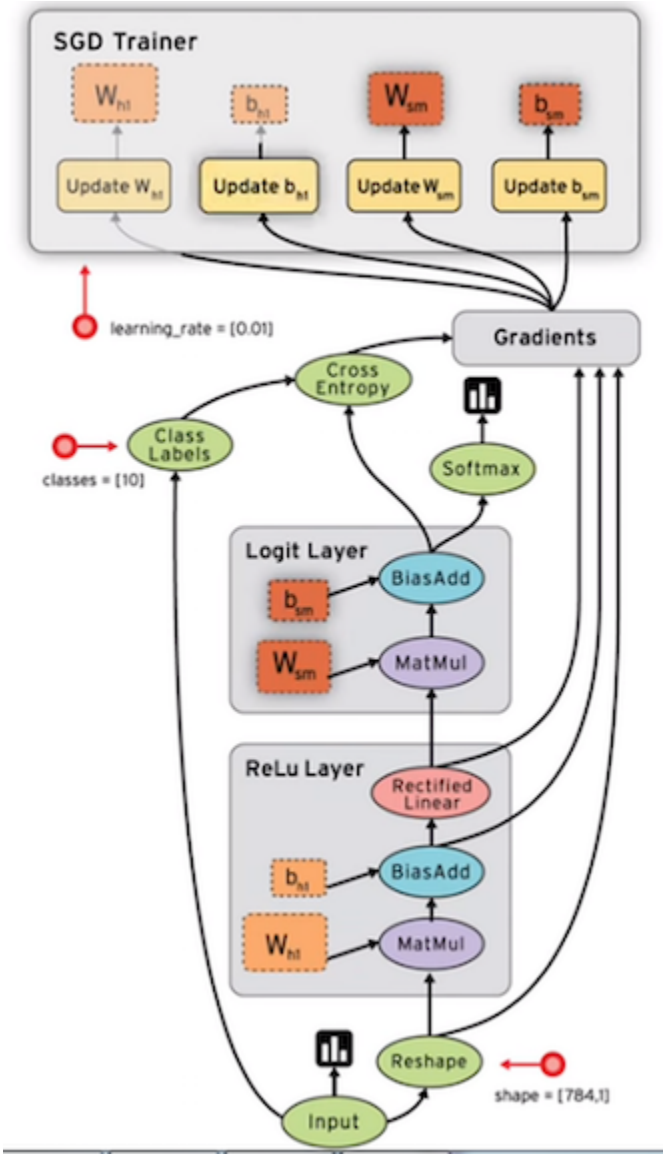


Figure 2.2: Tensor Flow Graph (Chollet and others 2015)

In TensorFlow, we write program functions into layers and then set our data up to flow through those layers. (Chollet and Allaire 2018)
Keras is a package that provides an RStudio interface to TensorFlow, with impressive levels of functionality. Far beyond serving as a neural net package, Keras gives R users access to the full suite of TensorFlow computational abilities, executed in C++

run time. Further, since TensorFlow is hardware independent, users can choose to run computations on their CPU, GPU (if in possession of a machine with a NVIDIA graphics card), or to connect to Google Cloud architecture. For these reasons, Keras may well prove over the next few years to be the best way to implement machine learning applications of all types in RStudio. (Chollet and others 2015)

2.2 Tensors

Just as base R relies on vectors and matrices to store data, and as dplyr relies on data frames, TensorFlow relies on a base data type called a tensor.(Chollet and Allaire 2018) A tensor can be any size:

- Scalar = 0D Tensor
- Vector = 1D Tensor
- Matrix = 2D Tensor
- A matrix of matrices (can visualize as a cube) = 3D Tensor
- Etc.

When discussing the size of a tensor, we can say *two-dimensional tensor*, *2-D tensor*, or *a tensor of rank 2*. (29) The dimension refers to the number of axes the tensor has, so the convention is slightly different than our typical R convention of referring to dimension as the number of rows or columns in a data frame. (29)

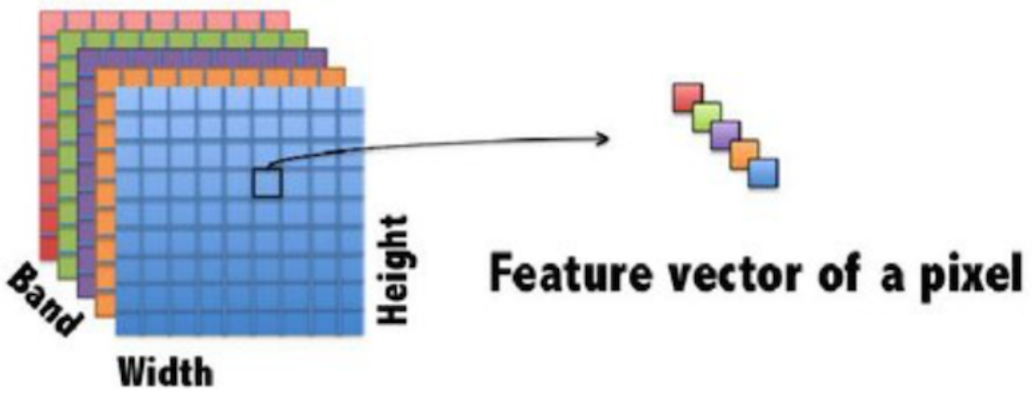


Figure 2.3: Image Tensor (Fan and Messinger 2016)

Before we can train a neural net to recognize digits, we need to form the MNIST data into tensors of the correct size.(Chollet and Allaire 2018) For images, TensorFlow expects to receive grids of normalized pixels (dimensions 1 and 2), with a third axis to represent the color channels (labeled band in Figure 2.3), and a fourth axis representing each individual sample. (34) In this case, our color channel has a length of one, since the digits data set is in gray-scale (labeled "Feature vector of a pixel" in Figure 2.3). (Fan and Messinger 2016) All of this means image data is interpreted as a *tensor of rank four*. Finally, we need to convert the categorical response variable $y \in [0, 9]$ to a one-hot encoding.(Chollet and others 2015)

```
# forming tensors for input
# note the images in MNIST are of size 28*28 pixels
# array_reshape operates row-wise
# array_reshape is the appropriate function
# here for it's C-style functionality
x_train <- array_reshape(x = x_train, dim = c(nrow(x_train),28*28))
x_test <- array_reshape(x = x_test, dim = c(nrow(x_test), 28*28))

# normalizing the pixels
x_train <- x_train / 255
x_test <- x_test / 255

# One-hot encoding y
# to_categorical is a KERAS function that
#"converts a class vector to binary class matrix"
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

2.3 Setting Up the Neural Net

We will start by setting up a network with a single hidden layer. The call keras_model_sequential initializes a feed-forward network. (Allaire, n.d.) Calls to layer_dense adds layers to the network which are fully connected to the previous layer, meaning each neuron sends inputs to every neuron of the next layer. (Allaire, n.d.) Notice, the input layer does not need to be initialized in the model. Inside layer_dense, we specify the number of neurons in the layer (units), and the neuron

function $g(x, w)$ (activation). (Chollet and Allaire 2018) In the first hidden layer, we also specify the size of the input from the tensor x_train. (26)

```
#Initialize the model

#first call to layer_dense() processes the images
#second call to layer_dense() returns an array of
#10 probability scores which sum to one that
#denote probability image is digit
#specified by that neuron
mod_simple <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu",
              input_shape = c(28*28)) %>%
  layer_dense(units = 10, activation = "softmax")
```

Next, we compile the network, specifying the type of stochastic gradient to use in optimization (optimizer), the loss function to optimize over (loss), and metrics for the network to produce while training. (Chollet and Allaire 2018) The Keras documentation website can help with choosing the most appropriate optimization and loss methods for various prediction types. (`https://keras.io`) In this case, Rmsprop is a stochastic gradient method that uses normalized results instead of the raw gradient to control the speed of learning. Categorical Cross-entropy is a log loss function for use with categorical variables.

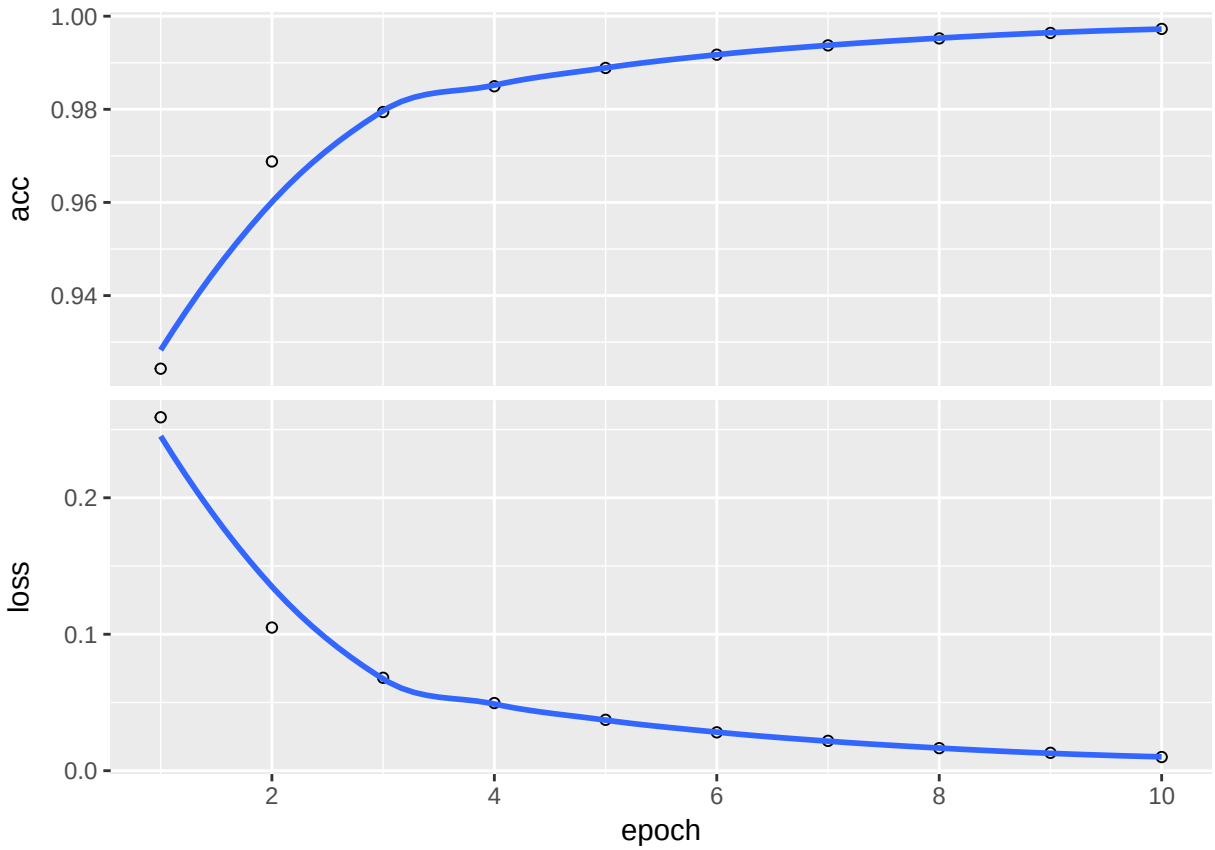
```
#compile the model.
#notice, modifies in place
mod_simple %>% compile(optimizer = "rmsprop",
                       loss = "categorical_crossentropy",
                       metrics = c("accuracy"))
```

To train the model, we call fit, specifying the number of times to iterate over the entire data set (epochs) and the size of samples for the gradient (batch_size, default = 32). (Chollet and Allaire 2018) Saving this fit object allows for generation of a plot showing the training history. (66)

```
#train the model
#Keras will modify in place

mod_training_data <- mod_simple %>%
  fit(x_train, y_train, epochs = 10, batch_size = 128)

#Plot fit history
plot(mod_training_data)
```



After training, evaluate() will provide loss data for the test set and predict_classes will return predictions. (Chollet and Allaire 2018)

```
#Feeding trained model the test sets and checking for accuracy
mod_simple_pred_results <- mod_simple %>% evaluate(x_test, y_test)
mod_simple_pred_results
```

```
$loss
[1] 0.06704257
```

```
$acc
[1] 0.9819
```

```
#Can also get predictions
test_preds <- mod_simple %>% predict_classes(x_test[27:39,])
#Predicted digits
test_preds
```

```
 [1] 7 4 0 1 3 1 3 4 7 2 7 1 2
```

```
#Actual digits
y_test_to_compare
```

```
 [1] 7 4 0 1 3 1 3 4 7 2 7 1 2
```

As our accuracy results indicate, training the MNIST set is a trivial task for a neural net.

2.4 Common Uses for Neural Networks

As of the writing of this paper, neural networks in the "real world" have been relegated almost exclusively to the world of computer vision and image classification. Recent increases in computational power have made the task of image classification trivial through the implementation of neural networks. Additionally, neural networks have shown promise in classification problems in many areas of research, including cancer and genomic work, and in time series applications. Moving forward, finding ways to apply the power of neural nets in predictive problems stands as an open research area for statisticians. (Chollet and Allaire 2018)

Conclusion

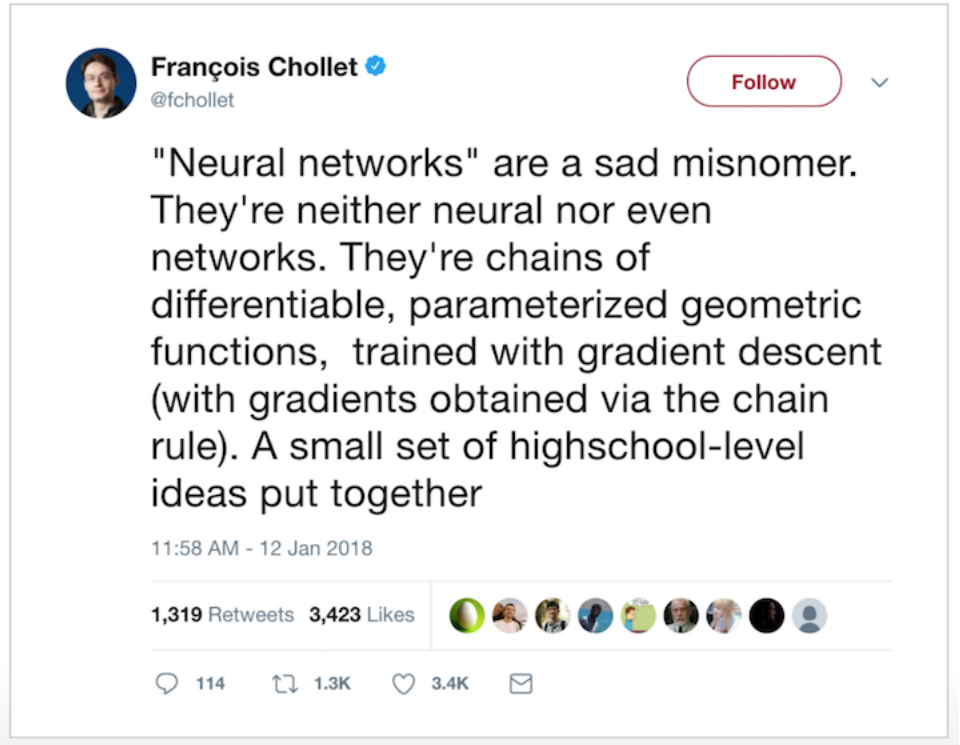


Figure 4.4: Tweet From the Author of the Keras Package

Now that we have made our way through the theory underlying neural networks and implemented one, we can enjoy the humor of this tweet. Google's TensorFlow library and the Keras R Interface to TensorFlow make the use of neural networks simple and computationally reasonable for many datasets. And, the mathematics underlying neural networks has proven to be relatively simple. Using typical statistical tools; like the Sigmoid function, linear combinations of data with weights, and measures of loss; combined with a clever use of gradient descent; has given modern statisticians a powerful tool set for classification problems that is ripe with possibilities for continued research.

References

Allaire, J.J. n.d. *TensorFlow for R*. https://tensorflow.rstudio.com/keras/.

Chollet, François, and J.J. Allaire. 2018. *Deep Learning with R*. Shelter Island, New York: Manning.

Chollet, François, and others. 2015. "Keras." https://github.com/keras-team/keras; GitHub.

Fan, Lei, and David W. Messinger. 2016. "Tensor Subspace Analysis for Spatial-Spectral Classification of Hyperspectral Data." *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXII*. doi:10.1117/12.2223815.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Hastie, Robert; Trevor; Tibshirani, and Jerome Friedman. 2013. *Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.

LeCun, Y., Y. Bengio, and P. Huffier. 1998. "Gradient-Based Learning Applied to Document Recognition." In *Proceedings of the Ieee*, 11th ed. Vol. 86. 2278-2324.

LeCun, Yann, Corinna Cortes, and Christopher J.C. Burges. n.d. "THE Mnist Database." *MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges*. http://yann.lecun.com/exdb/mnist/.

Moujahid, Adil. 2016. "A Practical Introduction to Deep Learning with Caffe and Python." *A Practical Introduction to Deep Learning with Caffe and Python*. http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/.

Rouse, Margaret. n.d. "What Is Neural Network? - Definition from Whatis.com."

SearchNetworking. `http://searchnetworking.techtarget.com/definition/neural-network`.