18.0651 Final Project: Application of a Neural Network for Classifying Real and Fake News Headlines

Kelly Mathesius

12 May 2020

1 Introduction

Neural networks can be used for recognizing a variety of complex and unintuitive patterns in data that could otherwise go unrecognized. An interesting (and perhaps amusing) application for a neural network is recognizing patterns in news headlines, and evauluating if headlines are real or fake. Given the abundance of articles shared on social media from both reliable and questionable sources, I thought it would be interesting to train a neural network to discern between real and fake articles based only on their headlines.

Classifying news headlines as real or fake involves recognizing complex patterns in word choice, phrase structure, grammar, and more. These patterns would be difficult for a human to recognize and implement manually in a classifier, which makes this classification problem a good candidate for a neural network.

2 Neural Network Setup

This section describes the data used for training the neural network to classify news headlines and the code used for processing the data and running the neural network.

2.1 Data

This project uses news data compiled on Kaggle [1] [1]. The complied dataset came in two csv files: one containining the real news data, the other containing the fake news data. Each of the files had four columns of data: article title, article text, article subject, and publication date.

I combined both csv files into a single file, and added a 'Real' and 'Fake' classifier to each row in the file (according to which file the row originally came from). Additionally, I removed the article text, article subject, and publication data columns, since I wanted to classify the articles based only on their headlines. Some examples from the processed data are shown below in Table 1.

Table 1: Sample news headlines and associated classifiers from larger dataset.

Article Title	**Article Classifier**
Alabama to certify Democrat Jones winner of Senate election	Real
U.N. says worried by reports of forced displacement of Kurds in northern Iraq	Real
Rosie O'Donnell Thinks Martial Law is in Order to Prevent a Trump Presidency	Fake
John McCain Throws Tantrum, Decries Trump for 'Letting Assad Stay'	Fake

2.2 Code Description

This section will give a brief description of the code used to process and classify the data for this project. The code for this project was written in MATLAB, and adapted methods and examples from the MATLAB Deep Learning Toolbox and Text Analytics Toolbox [2]. The full code for this project can be seen in Appendix A.

[1]Kaggle is a subsidiary of Google which hosts an online data science community with user-assembled open-source datasets.

2.2.1 Pre-processing the Data

The first step of the code was to pre-process the data into a more suitable format for training the neural network. The csv file of data (described in the previous section) was imported into the code, and the data was split such that 80% was used for training the neural network and 20% was used for testing the neural network. The article headline strings were tokenized using the `tokenizedDocument` method, and processed to remove capitalization and punctuation. The tokenized words were then mapped to corresponding numerical indices using the `wordEncoding` method. The headlines were then sequenced into vectors of a fixed length (the vectors were truncated or padded as needed) containing their corresponding numerical indices using the `doc2string` method.

2.2.2 The Neural Network

The neural network is trained and validated using the `trainNetwork` method. The neural network uses an Adam solver, and requires several input parameters including the mini-batch size and number of epochs. These parameters are varied to investigate how they impact the validation accuracy and training time, as discussed in the following section.

3 Results

This section displays results from the trained neural network, and describes how the number of epochs and mini-batch size impacted the effectiveness of the neural network.

3.1 Number of Epochs

The number of epochs is the number of complete passes through the data made by the neural network during training. The neural network was trained using 1, 2, 3, and 10 epochs, while holding all other inputs constant (mini-batch size = 128). A summary of the results of these trainings is shown in Figure 1.

The validation accuracy remains constant at $\sim 96\%$, regardless of the number of epochs used for training the data. Because there is so much training data available for the neural network, the neural network is able to converge its classifier in only one epoch (for the selected input parameters, which are held constant). If less data was available, then one epoch might not be sufficient for the neural network to converge,

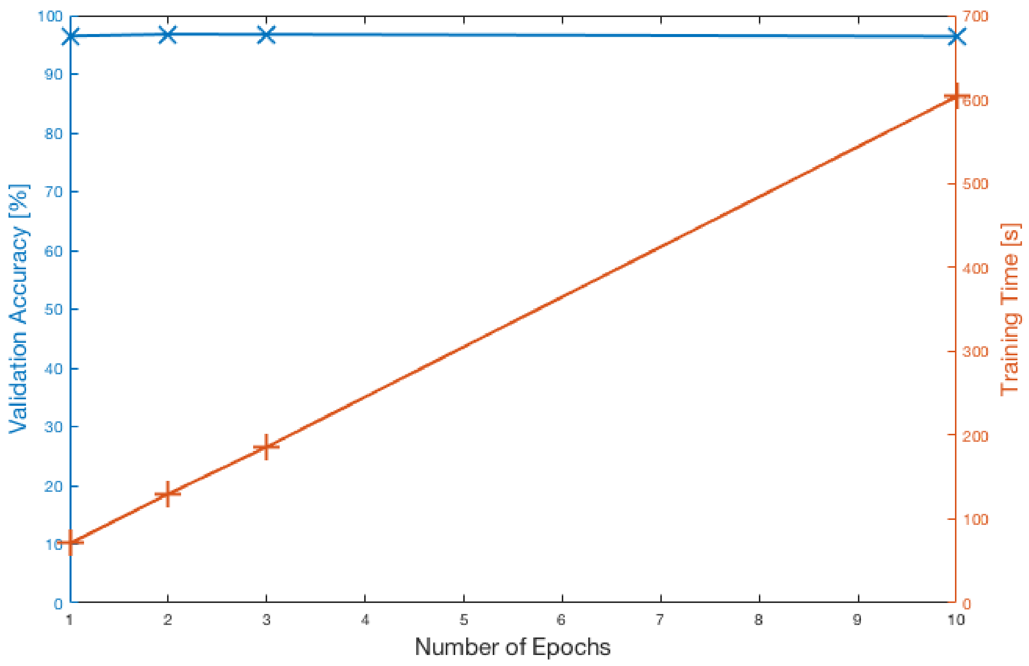

Figure 1: Summary of results for varying number of epochs.

and we would expect validation accuracy to increase with number of epochs (until it converged somewhere else). The time is takes to train the data increases linearly with the number of epochs. This makes sense, as the amount of time it takes to train the data should scale with the number of times the neural network passes through the data.

A plot of the validation accuracy of the neural network for 10 epochs is shown in Figure 2. As can be seen, the neural network converges after only one epoch. The training during the additional epochs runs the risk of overfitting the classifier to the training data, and therefor negatively impacting the performance. This doesn't appear to happen for 10 epochs (as demonstrated by the constant validation accuracy plotted in Figure 1). However, it's possible that for larger numbers of epochs, the validation accuracy could actually decrease. For this set of parameters, 1 or 2 epochs is a good choice to ensure that the classifier is fully converged, but that it is not overfit to the training data.

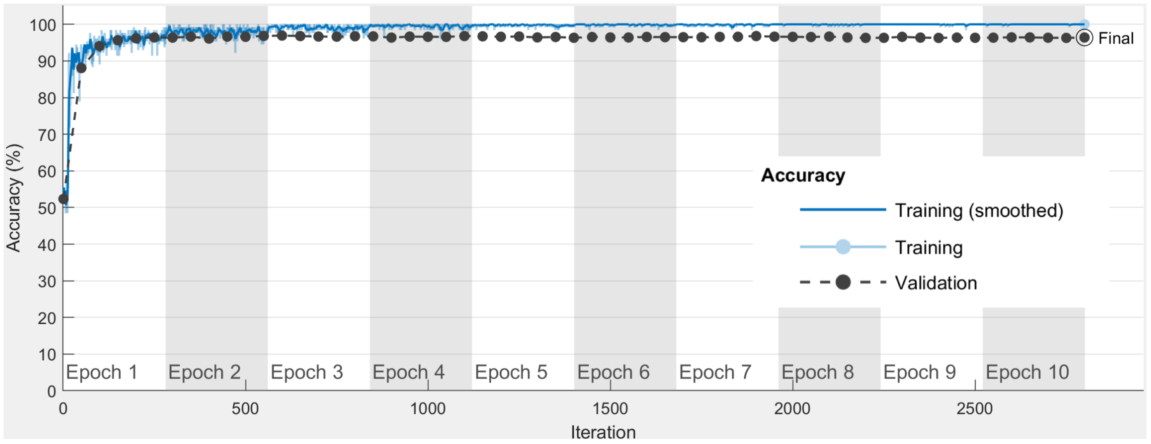

Figure 2: Training and validation accuracy for 10 epochs. The training and validation accuracy converge well before 10 epochs; for this set of input parameters, 1 or 2 epochs is probably a more reaonsable choice.

3.2 Mini-batch Size

The mini-batch is the number of data samples from the larger training data set evaluated by the neural network before updating its weights. Varying the mini-batch size has a significant effect on the validation accuracy and training time for the neural network (assuming all other parameters are fixed), as shown in Figure 3.

The validation accuracy holds constant at $\sim 96\%$ for mini-batch sizes less $< 1,000$. For larger mini-batch sizes, the validation accuracy drops off very quickly, until it plateaus at $\sim 53\%$ for mini-batch sizes $> 8,000$. This validation accuracy for mini-batch sizes $> 8,000$ indicates that at these mini-batch sizes the neural network essentially learns nothing - at the end of the training, the classifier is blindly guessing whether the news headline is real or fake, and is correct approximately half of the time. This trend is reasonable - for smaller mini-batch sizes, the neural network is allowed to update its internal weights many times before reaching the end of the training (2 epochs for these tests). For mini-batch sizes $< 1,000$, the neural network is able to converge almost completely. However, with larger mini-batch sizes, the nummber of times the neural network can update its weights is insufficient for convergence. Eventually, the neural network is updated so few times during the training that it must blindly guess at classifying the headlines.

The training time drops off very quickly with increasing mini-batch size, until it converges at ~ 60s for mini-batch sizes $> 1,000$. As the mini-batch size increases, the optimizer solves for internal weights fewer times before the end of the neural

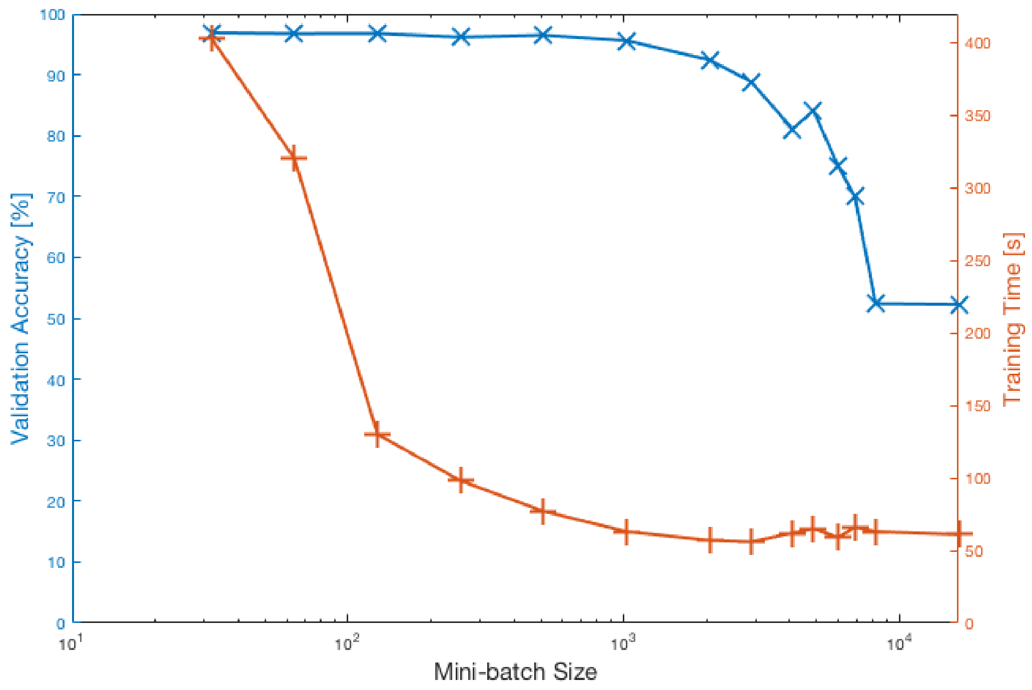

Figure 3: Summary of results for varying mini-batch size.

network training. As the mini-batch size increases further, the time spent calculating internal weights becomes minimal (since very few weights are calculated), and the time it takes for the neural network to simply read the input data probably becomes dominant. Therefore, it makes sense that training time would decrease with increasing mini-batch size, and then plateau at some mini-batch size.

For this set of input parameters, a mini-batch size of $\sim 1,000$ is a good choice, since trained neural network gets the benefits of full convergence of the classifier and a reasonably small number of weights to compute, minimizing the training time.

Analyzing the behavior of the validation accuracy versus training progress for different mini-batch sizes is also interesting. Validation accuracies for mini-batch sizes of 32, 1,024, and 16,384 are shown in Figure 4.

For a mini-batch size of 32 (top subplot in Figure 4), the neural network converges very quickly during the training. For a mini-batch size of 1,024 (middle subplot in Figure 4), the neural network converges more slowly, just barely converging by the end of the training. For a mini-batch size of 16,384 (bottom subplot in Figure 4), the

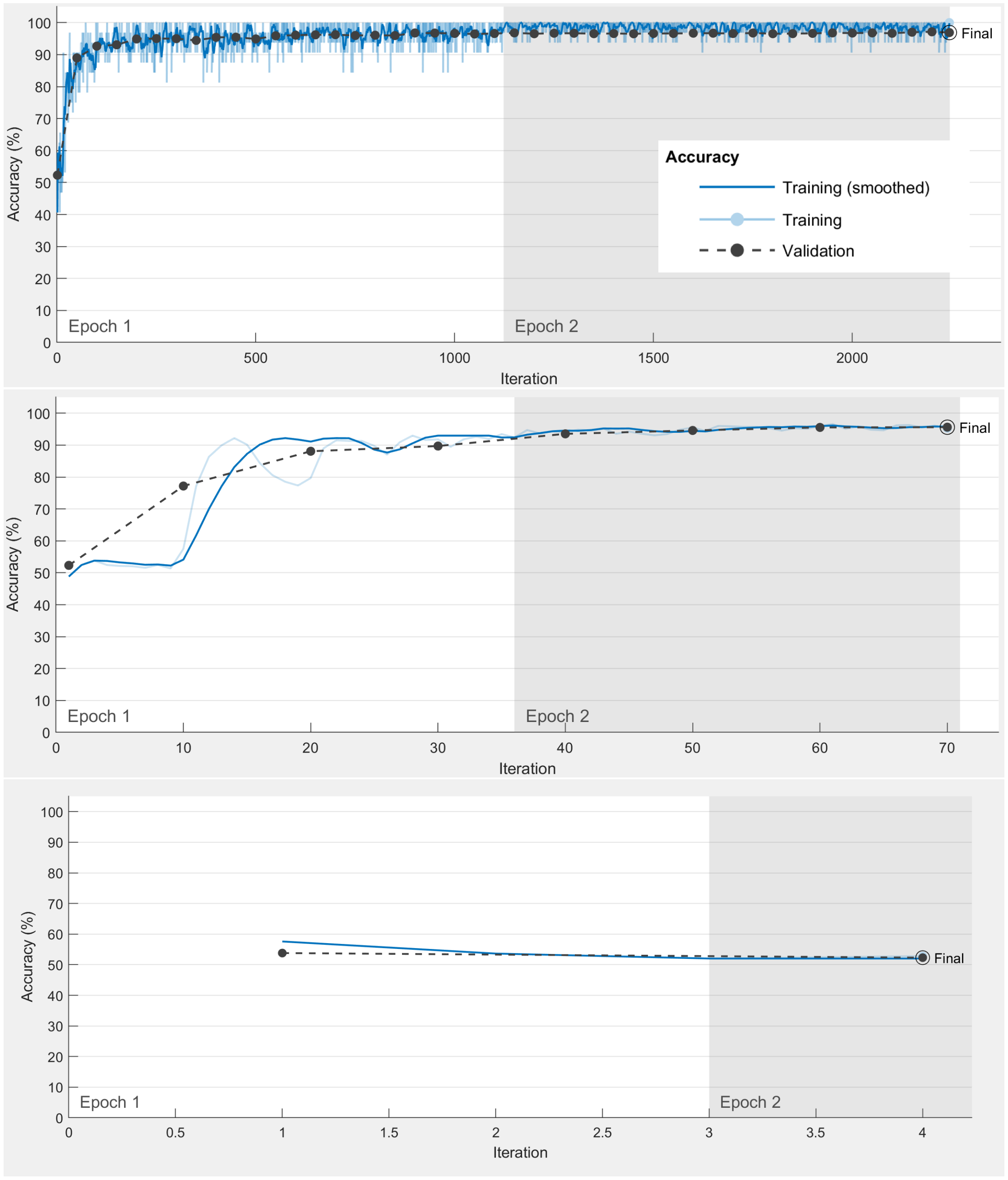

Figure 4: Comparison of validation data accuracy for mini-batch sizes of 32 (top), 1,024 (middle), and 16,384 (bottom).

neural network has not converged at all, and the validation accuracy holds constant at $\sim 50\%$. For larger mini-batch sizes, the neural network would likley be able to converge if it was allowed more epochs. The network would have more opportunities to update its weights (like the neural networks with smaller mini-batch sizes), thus producing a better result.

4 Conclusion and Future Work

A neural network can be used to accurately classify news headlines as real or fake. If the neural network is built and trained correctly, it can classify headlines with up to 96% accuracy. This has a variety of applications for detecting false articles shared on social media or elsewhere.

The performance of the neural network at classifying news headlines is dependent on the number of epochs and the mini-batch size (among other things). For a sufficiently small mini-batch size, the accuracy of the neural network does not depond on the number of epochs (assuming the neural network is not overfit to the training data), and the training time increases with the number epochs. When varying the mini-batch size, both the validation accuracy and training time decrease with increasing mini-batch size, before plateauing at some value. With these insights in mind, values for number of epochs and mini-batch size must be chosen carefully – it is important to make sure the neural network has fully converged to provide high classification accuracy (without overfitting), however a smaller training time is desirable to reduce computational time and effort.

Future work in characterizing the effects of other parameters on the neural network could provide a more complete picture to future neural network designers. This can help to ensure the best possible classification accuracy.

References

[1] Clement Bisaillon. *Fake and real news dataset*. `https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset`. [Accessed: 2020-04-20]. 2020.

[2] *Classify Text Data Using Deep Learning*. `https://www.mathworks.com/help/textanalytics/ug/classify-text-data-using-deep-learning.html`. [Accessed: 2020-04-20]. 2020.

Appendix A MATLAB Code

```
close all
clear all
clc

%import data
filename = 'Data/combined_data_abridged.csv';
data = readtable(filename,'TextType','string');
head(data)

data.status = categorical(data.status);

%divide data into training and validation data
cvp = cvpartition(data.status,'Holdout',0.2);
dataTrain = data(training(cvp),:);
dataValidation = data(test(cvp),:);

%define which part of the data is for training and
   validation
textDataTrain = dataTrain.title;
textDataValidation = dataValidation.title;
YTrain = dataTrain.status;
YValidation = dataValidation.status;

%pre-process data for the neural network
documentsTrain = preprocessText(textDataTrain);
documentsValidation = preprocessText(textDataValidation);

enc = wordEncoding(documentsTrain);

%convert headlines to vectors of numerical indices
sequenceLength = 25;
XTrain = doc2sequence(enc,documentsTrain,'Length',
   sequenceLength);
XValidation = doc2sequence(enc,documentsValidation,'Length
   ',sequenceLength);
```

```
%specifiy neural network options/parameters
inputSize = 1;
embeddingDimension = 50;
numHiddenUnits = 80;

numWords = enc.NumWords;
numClasses = numel(categories(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension,numWords)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]

options = trainingOptions('adam', ...
    'MiniBatchSize',1024, ...
    'GradientThreshold',Inf, ...
    'MaxEpochs',2, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{XValidation,YValidation}, ...
    'Plots','training-progress', ...
    'ValidationFrequency', 10, ...
    'Verbose',false);

%train the neural network
net = trainNetwork(XTrain,YTrain,layers,options);
```