PIC16B HW5: Fake News Classification

PIC16B HW

AUTHOR
Zhe Shen

PUBLISHED
March 9, 2023

HW5: Fake News Classification

With the development of information technology, we now have more and more access to news from all over the world and in a variety of fields. But due to the excessive commercialization of the news industry and the decline of professional ethics, we are often plagued by fake news. This is undoubtedly a major detriment to freedom of information.

And in this blog, we will introduce how to create neural network to classify fake news and real news. All of our work is based on `Tensorflow`. Before we start, I recommend you working with this blog in **Google Colab**.

Part 1: Data Processing

1.1 Import libararies and data

Let's import all libraries we will use in this blog first.

```
import pandas as pd
import numpy as np
import nltk
import re
import string
from matplotlib import pyplot as plt
import plotly.express as px
import tensorflow as tf
import keras
nltk.download('stopwords')
from nltk.corpus import stopwords
from keras import layers,losses
from sklearn.decomposition import PCA
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Then, we need to import training data. This Link at below hosted our training data. We will visit and read the data as follows:

```
train_url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_train.csv?ra
df = pd.read_csv(train_url)
df
```

	Unnamed: 0	title	text	fake
0	17366	Merkel: Strong result for Austria's FPO 'big c...	German Chancellor Angela Merkel said on Monday...	0
1	5634	Trump says Pence will lead voter fraud panel	WEST PALM BEACH, Fla.President Donald Trump sa...	0
2	17487	JUST IN: SUSPECTED LEAKER and “Close Confidant...	On December 5, 2017, Circa s Sara Carter warne...	1
3	12217	Thyssenkrupp has offered help to Argentina ove...	Germany s Thyssenkrupp, has offered assistance...	0
4	5535	Trump say appeals court decision on travel ban...	President Donald Trump on Thursday called the ...	0
...	...	...	...	...
22444	10709	ALARMING: NSA Refuses to Release Clinton-Lynch...	If Clinton and Lynch just talked about grandki...	1
22445	8731	Can Pence's vow not to sling mud survive a Tru...	() - In 1990, during a close and bitter congre...	0
22446	4733	Watch Trump Campaign Try To Spin Their Way Ou...	A new ad by the Hillary Clinton SuperPac Prior...	1
22447	3993	Trump celebrates first 100 days as president, ...	HARRISBURG, Pa.U.S. President Donald Trump hit...	0
22448	12896	TRUMP SUPPORTERS REACT TO DEBATE: “Clinton New...	MELBOURNE, FL is a town with a population of 7...	1

22449 rows × 4 columns

1.2 Make a Dataset

As we can see, there are 22449 data and each one include the news’ title and text. the “fake” colum represent this news is fake(1) or real(0).

Now, for our model to be able to identify the features of each data. We need to remove some stopword, such as “the”, “and” or “but”. These words usually do not provide valid information for our machine learning model. After this step, we will construct and return a tf.data.dataset with two inputs as `(title, text)` and one output as `fake`.

```
def make_dataset(df, batchsize):

  stop = stopwords.words('english')

  # rejoin the text without stopwords
  df["title_nostop"] = df["title"].apply(lambda x: ' '.join([word for word in x.split() if wor
  df["text_nostop"] = df["text"].apply(lambda x: ' '.join([word for word in x.split() if word

  # Create dataset
  dataset = tf.data.Dataset.from_tensor_slices(({"title": df[["title_nostop"]],"text": df[["te
  return dataset.batch(batchsize)
```

Batching causes model to train on chunks of data not a individual rows. This sometimes reduce accuracy, but can also greatly increase the speed of training. Finding a balance is key. In this time we will set batche size as 100 rows.

1.3 Split Dataset as Training set and Validation set

Next step, we will call the function we create above and split 80% of data for training, the rest of 20% for validation:

```
# make a dataset of 100 batchsize
batchsize=100
dataset = make_dataset(df,batchsize)

# split into training and validation set
train_dataset = dataset.take(int(0.8 * len(dataset)))
val_dataset = dataset.skip(int(0.8 * len(dataset))).take(int(0.2 * len(dataset)))
```

1.4 Baseline rate calculation

Baseline Rate is a reference standard for evaluating accuracy before constructing the model. Let's calculate the baseline rate, which is the only value above which our model can be considered successful. Since the fake news is marked as 1 in the `fake` column, we just need to add up the numbers to get the number of fake news.

```
fake_num = 0
for _, fake in train_dataset:
  fake_num += fake["fake"].numpy().sum()     # fake=1, real=0
print("Baseline rate: %.3f" % (fake_num/len(train_dataset))+"%")
```

```
Baseline rate: 52.206%
```

The result is 52.206% which means our model should be higher than 52.206%.

1.5 Text Vectorization

Finally, since our model only accepts numbers as input, we will use `TextVectorization` to pair each word with a number. This will allow numbers to be entered into our model instead of text.

```
size_vocabulary = 2000
```

```
def standardization(input_data):
    # lowercase the input
    lowercase = tf.strings.lower(input_data)
    # remove punctuations
    no_punctuation = tf.strings.regex_replace(lowercase,
                                  '[%s]' % re.escape(string.punctuation),'')
    return no_punctuation

# title vectorization
title_vectorize_layer = layers.TextVectorization(
    standardize=standardization,
    max_tokens=size_vocabulary,
    output_mode='int',
    output_sequence_length=500)
title_vectorize_layer.adapt(train_dataset.map(lambda x, y: x["title"]))

# text vectorization
text_vectorize_layer = layers.TextVectorization(
    standardize=standardization,
    max_tokens=size_vocabulary,
    output_mode='int',
    output_sequence_length=500)
text_vectorize_layer.adapt(train_dataset.map(lambda x, y: x["text"]))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.9/dist-
packages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83: Analyzer.lamba_check
(from tensorflow.python.autograph.pyct.static_analysis.liveness) is deprecated and will be
removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at
least in the same block. https://github.com/tensorflow/tensorflow/issues/56089
```

In this step, we first lowercase all the words and remove the punctuation. Then, we turn the title and text into a sequence of integers of length 500. As mentioned above, each integer will represent a word to be entered into the model.

Part 2: Create Model and Evaluation

Now, we will use the **Tensorflow** model to answer the following question: >When detecting fake news, is it most effective to focus on only the title of the article, the full text of the article, or both?

To answer this question, we will define three models to compare the results. And we will write our models using the Functional API instead of the Sequential API. Now, we need to specify the `inputs` to the `keras.Model` appropriately for let **Tensorflow** automatically ignore the unused inputs in the `Dataset`.

```
title_input = keras.Input(shape=(1,),name = "title", dtype = "string")
text_input = keras.Input(shape=(1, ),name = "text",dtype = "string")
```

```
#an embedding layer that will be shared by title and text inputs
embedding_layer = layers.Embedding(size_vocabulary, output_dim = 3, name="embedding")
```

Part 2.1 Model 1: news titles

Firstly, we will check how the title of the article perform in detecting fake news. So, Model 1 will only take the news titles as the inputs.

```
# creat model 1
model1_features = title_vectorize_layer(title_input)
model1_features = embedding_layer(model1_features)
model1_features = layers.Dropout(0.2)(model1_features)
model1_features = layers.GlobalAveragePooling1D()(model1_features)
model1_features = layers.Dense(64, activation="relu")(model1_features)
model1_features = layers.Dropout(0.2)(model1_features)
model1_features = layers.Dense(64, activation="relu")(model1_features)
model1_features = layers.Dropout(0.2)(model1_features)
output1 = layers.Dense(2, activation="relu", name="fake")(model1_features)

model1 = keras.Model(inputs = title_input,outputs = output1)
```

As shown above, `Model 1` has one vectorization layer, one embedding layer, one average pooling layer and two dense layers. Finally we set the output to fake so that we can compare it to our baseline rate.

In the next step, we will run 20 epochs on `Model 1` to obtain a more plausible result.

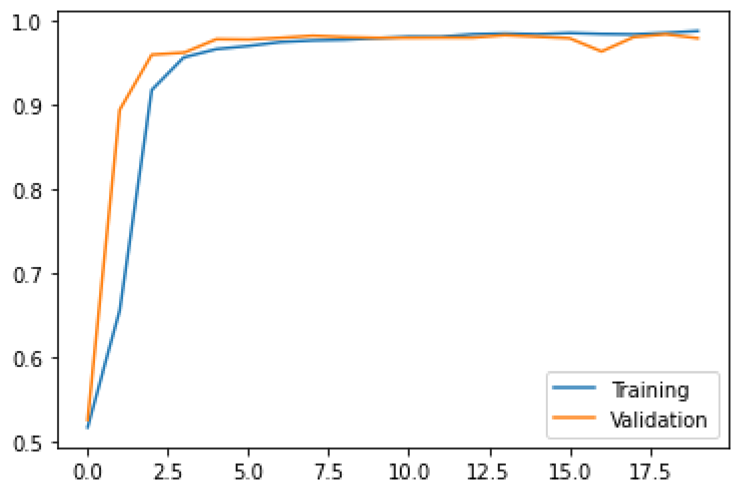
```
model1.compile(optimizer="adam",
        loss = losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=["accuracy"])

history1 = model1.fit(train_dataset,
            validation_data=val_dataset,
            epochs = 20)
```

```
Epoch 1/20

/usr/local/lib/python3.9/dist-packages/keras/engine/functional.py:638: UserWarning: Input dict
contained keys ['text'] which did not match any model input. They will be ignored by the
model.
  inputs = self._flatten_to_reference_inputs(inputs)

180/180 [==============================] - 4s 15ms/step - loss: 0.6924 - accuracy: 0.5177 -
val_loss: 0.6910 - val_accuracy: 0.5266
Epoch 2/20
180/180 [==============================] - 2s 14ms/step - loss: 0.6506 - accuracy: 0.6559 -
val_loss: 0.4917 - val_accuracy: 0.8935
Epoch 3/20
180/180 [==============================] - 2s 13ms/step - loss: 0.2919 - accuracy: 0.9170 -
val_loss: 0.1539 - val_accuracy: 0.9589
Epoch 4/20
180/180 [==============================] - 2s 13ms/step - loss: 0.1365 - accuracy: 0.9558 -
val_loss: 0.1150 - val_accuracy: 0.9613
Epoch 5/20
180/180 [==============================] - 2s 14ms/step - loss: 0.1034 - accuracy: 0.9655 -
val_loss: 0.0726 - val_accuracy: 0.9773
Epoch 6/20
180/180 [==============================] - 3s 18ms/step - loss: 0.0914 - accuracy: 0.9692 -
val_loss: 0.0724 - val_accuracy: 0.9768
```

```
Epoch 7/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0805 - accuracy: 0.9739 -
val_loss: 0.0647 - val_accuracy: 0.9789
Epoch 8/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0729 - accuracy: 0.9758 -
val_loss: 0.0560 - val_accuracy: 0.9813
Epoch 9/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0697 - accuracy: 0.9766 -
val_loss: 0.0596 - val_accuracy: 0.9798
Epoch 10/20
180/180 [==============================] - 3s 19ms/step - loss: 0.0628 - accuracy: 0.9784 -
val_loss: 0.0624 - val_accuracy: 0.9789
Epoch 11/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0583 - accuracy: 0.9802 -
val_loss: 0.0623 - val_accuracy: 0.9789
Epoch 12/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0591 - accuracy: 0.9801 -
val_loss: 0.0597 - val_accuracy: 0.9791
Epoch 13/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0525 - accuracy: 0.9831 -
val_loss: 0.0596 - val_accuracy: 0.9791
Epoch 14/20
180/180 [==============================] - 3s 16ms/step - loss: 0.0485 - accuracy: 0.9841 -
val_loss: 0.0508 - val_accuracy: 0.9820
Epoch 15/20
180/180 [==============================] - 3s 16ms/step - loss: 0.0488 - accuracy: 0.9832 -
val_loss: 0.0558 - val_accuracy: 0.9802
Epoch 16/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0480 - accuracy: 0.9846 -
val_loss: 0.0643 - val_accuracy: 0.9784
Epoch 17/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0478 - accuracy: 0.9835 -
val_loss: 0.0980 - val_accuracy: 0.9627
Epoch 18/20
180/180 [==============================] - 3s 16ms/step - loss: 0.0471 - accuracy: 0.9830 -
val_loss: 0.0602 - val_accuracy: 0.9798
Epoch 19/20
180/180 [==============================] - 2s 14ms/step - loss: 0.0455 - accuracy: 0.9847 -
val_loss: 0.0484 - val_accuracy: 0.9831
Epoch 20/20
180/180 [==============================] - 2s 13ms/step - loss: 0.0394 - accuracy: 0.9870 -
val_loss: 0.0632 - val_accuracy: 0.9784
```

It seems very good(almost 98% accuracy and val_accuracy in last epoch), so let’s visualize the training process:

```
plt.plot(history1.history["accuracy"],label="Training")
plt.plot(history1.history["val_accuracy"],label="Validation")
plt.legend()
plt.show()
```



As we can see, model 1 has 98% validation accuracy without overfitting.

Part 2.2 Model 2: news text

Secondly, we will check how the text of the article perform in detecting fake news. So, Model 2 will only take the news text as the inputs.

```
# creat model 2
model2_features = text_vectorize_layer(text_input)
model2_features = embedding_layer(model2_features)
model2_features = layers.Dropout(0.2)(model2_features)
model2_features = layers.GlobalAveragePooling1D()(model2_features)
model2_features = layers.Dense(1024, activation="relu")(model2_features)
model2_features = layers.Dropout(0.2)(model2_features)
model2_features = layers.Dense(64, activation="relu")(model2_features)
model2_features = layers.Dropout(0.2)(model2_features)
output2 = layers.Dense(2, activation="relu", name="fake")(model2_features)

model2 = keras.Model(inputs = text_input,outputs = output2)
```

Model 2 is highly similar to model 1. But because the text tends to be longer and complex, we modify the number of neurons in the first dense layer. model 1 has only 64, here we provide 1024. Let's see how model 2 perform.

```
model2.compile(optimizer="adam",
        loss = losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=["accuracy"])
history2 = model2.fit(train_dataset,
            validation_data=val_dataset,
            epochs = 20)
```

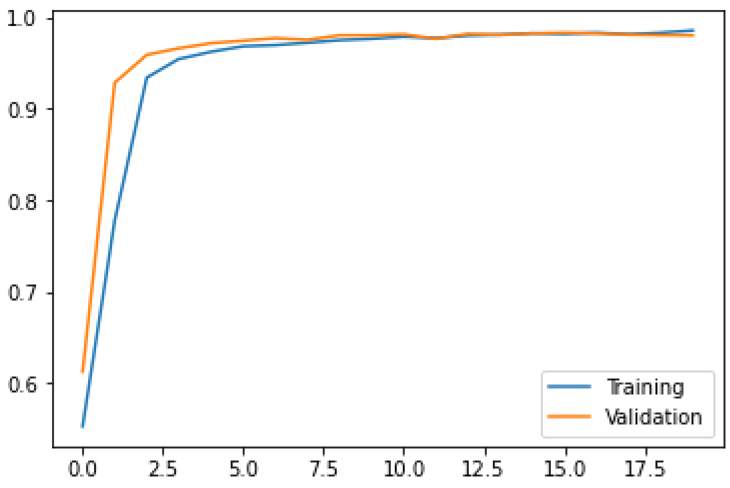
```
Epoch 1/20

/usr/local/lib/python3.9/dist-packages/keras/engine/functional.py:638: UserWarning: Input dict
contained keys ['title'] which did not match any model input. They will be ignored by the
model.
  inputs = self._flatten_to_reference_inputs(inputs)

180/180 [==============================] - 8s 31ms/step - loss: 0.6816 - accuracy: 0.5528 -
val_loss: 0.6485 - val_accuracy: 0.6127
```

```
Epoch 2/20
180/180 [==============================] - 5s 28ms/step - loss: 0.4568 - accuracy: 0.7774 -
val_loss: 0.2102 - val_accuracy: 0.9283
Epoch 3/20
180/180 [==============================] - 7s 39ms/step - loss: 0.1792 - accuracy: 0.9339 -
val_loss: 0.1216 - val_accuracy: 0.9589
Epoch 4/20
180/180 [==============================] - 5s 30ms/step - loss: 0.1302 - accuracy: 0.9544 -
val_loss: 0.0983 - val_accuracy: 0.9661
Epoch 5/20
180/180 [==============================] - 4s 25ms/step - loss: 0.1063 - accuracy: 0.9623 -
val_loss: 0.0868 - val_accuracy: 0.9717
Epoch 6/20
180/180 [==============================] - 6s 32ms/step - loss: 0.0925 - accuracy: 0.9682 -
val_loss: 0.0764 - val_accuracy: 0.9744
Epoch 7/20
180/180 [==============================] - 6s 31ms/step - loss: 0.0863 - accuracy: 0.9695 -
val_loss: 0.0687 - val_accuracy: 0.9771
Epoch 8/20
180/180 [==============================] - 4s 25ms/step - loss: 0.0791 - accuracy: 0.9721 -
val_loss: 0.0730 - val_accuracy: 0.9753
Epoch 9/20
180/180 [==============================] - 4s 25ms/step - loss: 0.0720 - accuracy: 0.9749 -
val_loss: 0.0611 - val_accuracy: 0.9802
Epoch 10/20
180/180 [==============================] - 5s 30ms/step - loss: 0.0718 - accuracy: 0.9764 -
val_loss: 0.0612 - val_accuracy: 0.9802
Epoch 11/20
180/180 [==============================] - 4s 25ms/step - loss: 0.0634 - accuracy: 0.9787 -
val_loss: 0.0569 - val_accuracy: 0.9813
Epoch 12/20
180/180 [==============================] - 5s 29ms/step - loss: 0.0647 - accuracy: 0.9771 -
val_loss: 0.0659 - val_accuracy: 0.9764
Epoch 13/20
180/180 [==============================] - 4s 25ms/step - loss: 0.0590 - accuracy: 0.9799 -
val_loss: 0.0562 - val_accuracy: 0.9818
Epoch 14/20
180/180 [==============================] - 6s 31ms/step - loss: 0.0553 - accuracy: 0.9809 -
val_loss: 0.0604 - val_accuracy: 0.9807
Epoch 15/20
180/180 [==============================] - 4s 24ms/step - loss: 0.0519 - accuracy: 0.9825 -
val_loss: 0.0539 - val_accuracy: 0.9822
Epoch 16/20
180/180 [==============================] - 5s 30ms/step - loss: 0.0522 - accuracy: 0.9818 -
val_loss: 0.0534 - val_accuracy: 0.9831
Epoch 17/20
180/180 [==============================] - 5s 26ms/step - loss: 0.0497 - accuracy: 0.9831 -
val_loss: 0.0549 - val_accuracy: 0.9822
Epoch 18/20
180/180 [==============================] - 4s 25ms/step - loss: 0.0513 - accuracy: 0.9814 -
val_loss: 0.0537 - val_accuracy: 0.9809
Epoch 19/20
180/180 [==============================] - 6s 31ms/step - loss: 0.0471 - accuracy: 0.9830 -
val_loss: 0.0570 - val_accuracy: 0.9804
```

```
Epoch 20/20
180/180 [==============================] - 5s 26ms/step - loss: 0.0421 - accuracy: 0.9855 -
val_loss: 0.0661 - val_accuracy: 0.9802
```

It seems very good(almost 98% accuracy and val_accuracy in last epoch), so let's visualize the training process:

```
plt.plot(history2.history["accuracy"],label="Training")
plt.plot(history2.history["val_accuracy"],label="Validation")
plt.legend()
plt.show()
```



As we can see model 2 has 98% validation accuracy without overfitting which very like model 1.

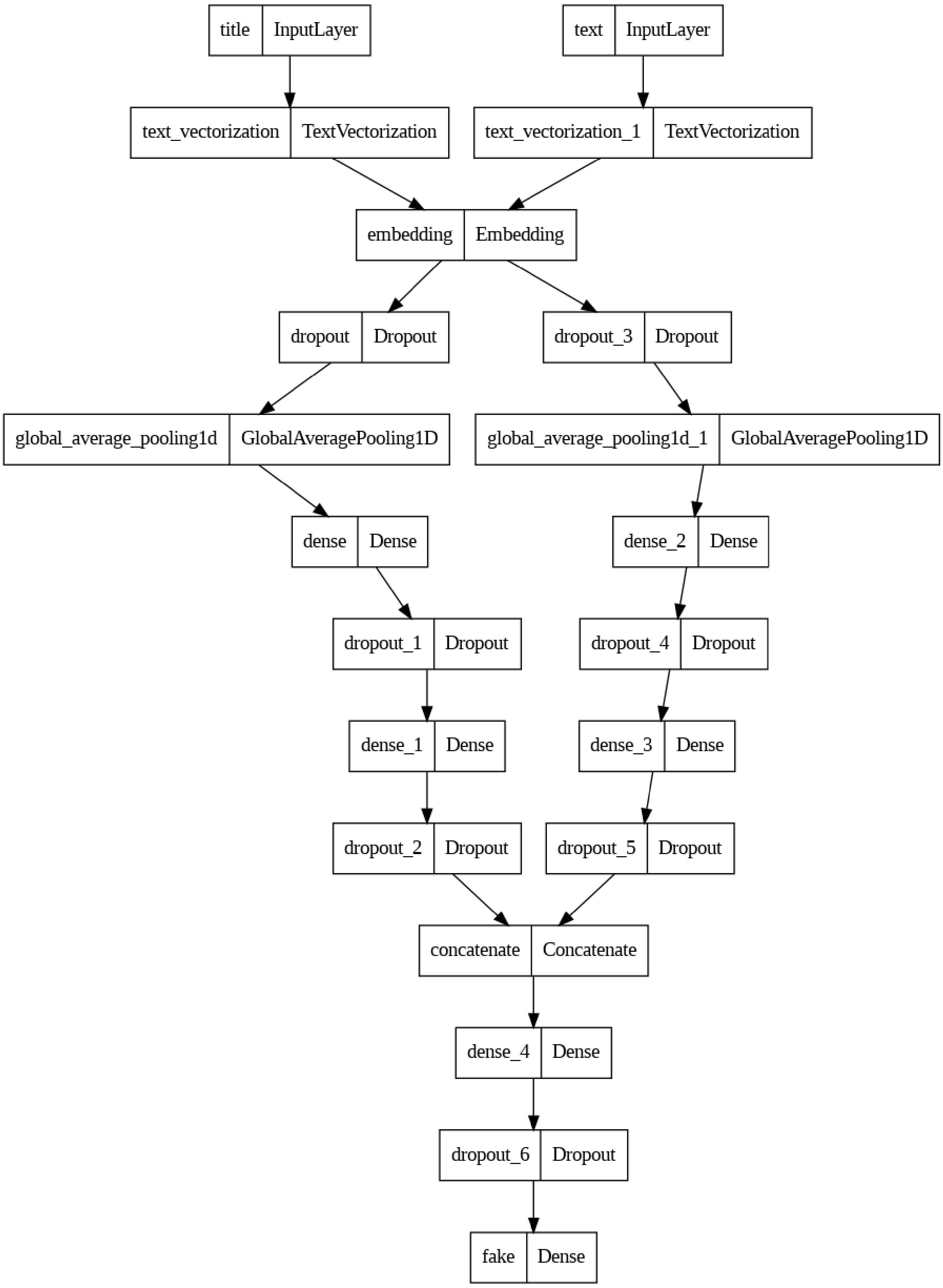
Part 2.3: Model 3: both of news' title and text

In this section, we will try to enter both the title and the text. Since we have already done the title input and text input separately in the previous section. So for the construction of model3 I will try to connect the 2 first models. The way to do this is to use `layers.concatenate`.

```
# Concatenate two previous models for model 3
model3_features = layers.concatenate([model1_features, model2_features], axis = 1)
model3_features = layers.Dense(64, activation="relu")(model3_features)
model3_features = layers.Dropout(0.2)(model3_features)
output3 = layers.Dense(2, activation="relu", name="fake")(model3_features)

model3 = keras.Model(inputs = [title_input, text_input],outputs = output3)
```

According to my judgment, model 3 will be a very well-developed model and the final performance will be excellent. So before we test it, let's see what the model looks like:

```
keras.utils.plot_model(model3)
```



Now let’s see how the model 3 performs:

```
model3.compile(optimizer="adam",
            loss = losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=["accuracy"])
```

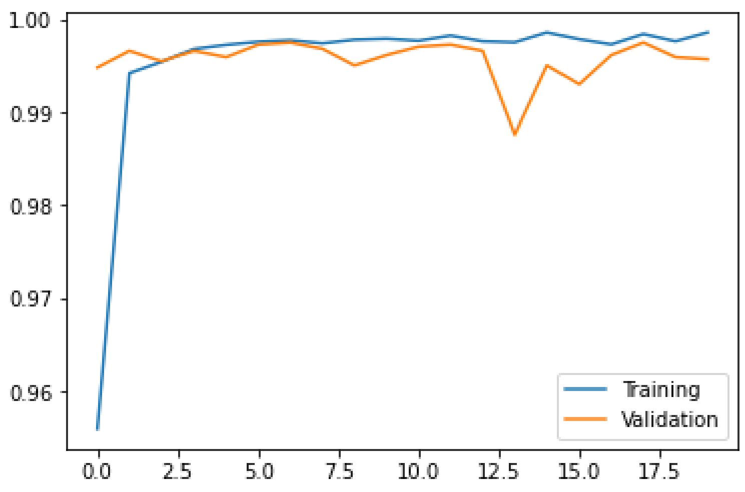
```
history3 = model3.fit(train_dataset, validation_data=val_dataset,
            epochs = 20)
```

```
Epoch 1/20
180/180 [==============================] - 8s 33ms/step - loss: 0.1326 - accuracy: 0.9559 -
val_loss: 0.0147 - val_accuracy: 0.9948
Epoch 2/20
180/180 [==============================] - 6s 32ms/step - loss: 0.0180 - accuracy: 0.9942 -
val_loss: 0.0111 - val_accuracy: 0.9966
Epoch 3/20
180/180 [==============================] - 6s 36ms/step - loss: 0.0135 - accuracy: 0.9954 -
val_loss: 0.0147 - val_accuracy: 0.9955
Epoch 4/20
180/180 [==============================] - 6s 36ms/step - loss: 0.0096 - accuracy: 0.9968 -
val_loss: 0.0109 - val_accuracy: 0.9966
Epoch 5/20
180/180 [==============================] - 6s 35ms/step - loss: 0.0078 - accuracy: 0.9973 -
val_loss: 0.0145 - val_accuracy: 0.9960
Epoch 6/20
180/180 [==============================] - 6s 34ms/step - loss: 0.0086 - accuracy: 0.9976 -
val_loss: 0.0084 - val_accuracy: 0.9973
Epoch 7/20
180/180 [==============================] - 6s 31ms/step - loss: 0.0071 - accuracy: 0.9978 -
val_loss: 0.0102 - val_accuracy: 0.9975
Epoch 8/20
180/180 [==============================] - 7s 39ms/step - loss: 0.0074 - accuracy: 0.9974 -
val_loss: 0.0083 - val_accuracy: 0.9969
Epoch 9/20
180/180 [==============================] - 7s 40ms/step - loss: 0.0060 - accuracy: 0.9978 -
val_loss: 0.0155 - val_accuracy: 0.9951
Epoch 10/20
180/180 [==============================] - 7s 39ms/step - loss: 0.0055 - accuracy: 0.9979 -
val_loss: 0.0113 - val_accuracy: 0.9962
Epoch 11/20
180/180 [==============================] - 7s 38ms/step - loss: 0.0065 - accuracy: 0.9977 -
val_loss: 0.0083 - val_accuracy: 0.9971
Epoch 12/20
180/180 [==============================] - 6s 32ms/step - loss: 0.0056 - accuracy: 0.9983 -
val_loss: 0.0069 - val_accuracy: 0.9973
Epoch 13/20
180/180 [==============================] - 7s 39ms/step - loss: 0.0072 - accuracy: 0.9977 -
val_loss: 0.0085 - val_accuracy: 0.9966
Epoch 14/20
180/180 [==============================] - 6s 32ms/step - loss: 0.0065 - accuracy: 0.9976 -
val_loss: 0.0437 - val_accuracy: 0.9876
Epoch 15/20
180/180 [==============================] - 7s 39ms/step - loss: 0.0048 - accuracy: 0.9986 -
val_loss: 0.0164 - val_accuracy: 0.9951
Epoch 16/20
180/180 [==============================] - 6s 31ms/step - loss: 0.0064 - accuracy: 0.9979 -
val_loss: 0.0260 - val_accuracy: 0.9930
Epoch 17/20
180/180 [==============================] - 6s 32ms/step - loss: 0.0077 - accuracy: 0.9973 -
val_loss: 0.0099 - val_accuracy: 0.9962
```

```
Epoch 18/20
180/180 [==============================] - 6s 32ms/step - loss: 0.0053 - accuracy: 0.9984 -
val_loss: 0.0074 - val_accuracy: 0.9975
Epoch 19/20
180/180 [==============================] - 6s 34ms/step - loss: 0.0063 - accuracy: 0.9977 -
val_loss: 0.0118 - val_accuracy: 0.9960
Epoch 20/20
180/180 [==============================] - 6s 35ms/step - loss: 0.0043 - accuracy: 0.9986 -
val_loss: 0.0164 - val_accuracy: 0.9957
```

It seems perfect(almost 100% accuracy and val_accuracy in last epoch), so let's visualize the training process:

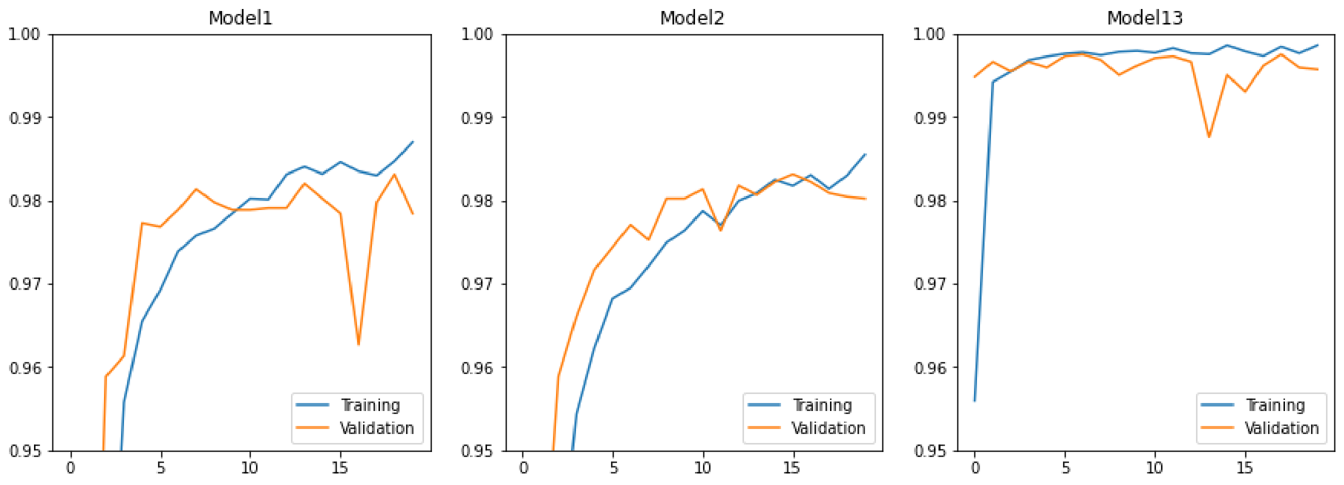
```
plt.plot(history3.history["accuracy"],label="Training")
plt.plot(history3.history["val_accuracy"],label="Validation")
plt.legend()
plt.show()
```



Part 2.4: Model Evaluation

To facilitate the analysis, we visualize the training history of these 3 models as follows:

```
# visualize model 1's training history
plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
plt.plot(history1.history["accuracy"],label="Training")
plt.plot(history1.history["val_accuracy"],label="Validation")
plt.ylim([0.95,1])
plt.title("Model1")
plt.legend(loc="lower right")

# visualize model 2's training history
plt.subplot(1,3,2)
plt.plot(history2.history["accuracy"],label="Training")
plt.plot(history2.history["val_accuracy"],label="Validation")
plt.ylim([0.95,1])
plt.title("Model2")
plt.legend(loc="lower right")

# visualize model 3's training history
plt.subplot(1,3,3)
plt.plot(history3.history["accuracy"],label="Training")
plt.plot(history3.history["val_accuracy"],label="Validation")
plt.ylim([0.95,1])
plt.title("Model13")
plt.legend(loc="lower right")
plt.show()
```



We can see that model 3 is very close to the best accuracy almost at the first epoch. While the other two models are need about 2 to 4 epochs to get close to the final accuracy. And the accuracy of model 3 is always kept at a very high level. So, now let's try using model3 to run on the new fake news dataset

```
# loading test set.
url = "https://github.com/PhilChodrow/PIC16b/blob/master/datasets/fake_news_test.csv?raw=true"
test_df = pd.read_csv(url)
test_set = make_dataset(df,BATCHSIZE)
# test  accuracy
test_loss, test_acc = model3.evaluate(test_set)
print(test_acc)
```

```
225/225 [==============================] - 3s 12ms/step - loss: 0.0039 - accuracy: 0.9989
0.9989309310913086
```

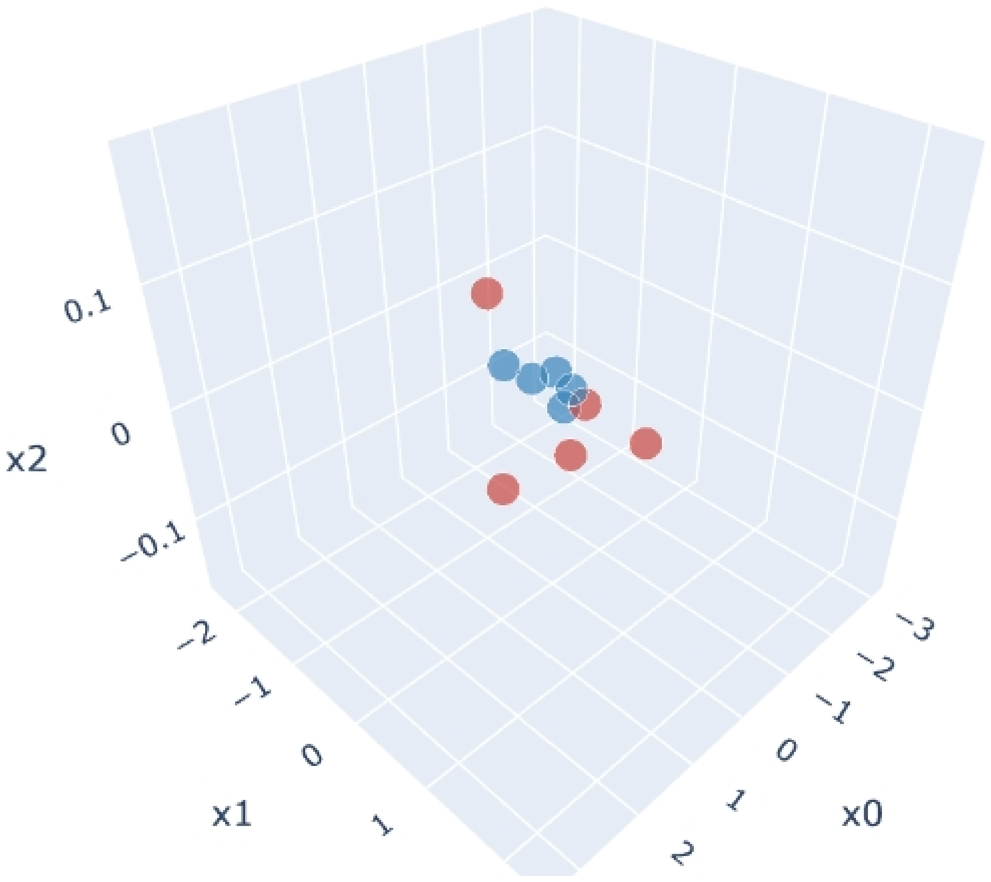
The results are very good, and we obtained an accuracy of 99.89% here. It is perfect.

Part 3: Embedding Visualization

Based on the success of the above work, we can try to explore how certain specific words are related between true and false news. Here I will try to use 3D embedding to show the use of positive words (Like "interest", "development", "progress", "young" and "better") and negative words (Like "died", "shot", "gun", "drug" and "kill") in true and false news.

```
negative_w = ["died", "shot", "gun", "drug", "kill"]
positive_w = ["interest", "development", "progress", "young", "better"]

weights = model3.get_layer('embedding').get_weights()[0]
vocab = text_vectorize_layer.get_vocabulary()
pca = PCA(n_components=3)
weights = pca.fit_transform(weights)
```

```python
embedding_df = pd.DataFrame({'word':vocab,'x0':weights[:,0],'x1':weights[:,1],'x2':weights[:,2

def mapper(x):
    if x in negative_w:return 1 # Red
    elif x in positive_w:return 3 # Blue
    else:return 0
embedding_df["highlight"] = embedding_df["word"].apply(mapper)
embedding_df["size"] = np.array(200*(embedding_df["highlight"] > 0))

fig = px.scatter_3d(embedding_df, x = "x0", y = "x1", z = "x2",color = "highlight",
                 size = list(embedding_df["size"]),color_continuous_scale = "rdbu",
                 range_color=(0.5,3.5),hover_name = "word")
fig.show()
```



20230310153752.jpg

PS: this visualization can not show on quarto, so I just upload my screenshot here.

As shown in the image, we see that positive words (show as Blue points) tend to cluster in a certain area, while negative words (show as Red points) tend to be scattered. This indicates that our model has a more obvious judgment of the positivity and negativity of words. This also means that all positive words (“interest”, “development”, “progress”, “young” and “better”) express similar good meanings sience their coordinates are more concentrated.. For negative words (“died”, “shot”, “gun”, “drug” and “kill”), the scattered points indicate that the presence of larger values in the 3D coordinates means that the word itself has a more negative meaning for our model. And this is done by providing different weights to the words via adding embedding layer.(check carefully at the 3D coordinates).