Word Embedding Part 2 Report

Yuhuan Huang, Aug 2025

Part1: the Keras Tokenizer

```
    >>> print(tk_word_counts['learning'], tk_word_counts['pow
    4 2
    >>> print(tk_word_index['powerful'], tk_word_index['deep'
    6 8
    >>> print(tx_word_docs['networks'], tx_word_docs['advance
    3 1
    >>> np.isclose(tk_word_freqs['learning'], 0.1904761904761
    np.True_
    >>> print(tk_sequences_full)
    [[3, 2, 7, 6], [8, 2, 9, 10, 4, 5], [4, 5, 11, 3, 2], [6,
    >>> print(tk_sequences_limited)
    [[3, 2, 1, 1], [1, 2, 1, 1, 1, 1], [1, 1, 1, 3, 2], [1, 1
'''
doctest.testmod()
```

```
TestResults(failed=0, attempted=6)
```

Part2: Global Average Pooling

```
#For test purposes - DO NOT modify the code below this line
import doctest
'''
  >>> print(embeddings.shape)
  (1, 5, 4)
  >>> print(manual_pooling.shape)
  (1, 4)
  >>> print(keras_pooling.shape)
  (1, 4)
  >>> np.isclose(manual_pooling-keras_pooling.numpy(), [[0, 0, 0, 0]], atol=1e-03)
  array([[ True,  True,  True,  True]])
'''
doctest.testmod()
```

```
TestResults(failed=0, attempted=4)
```

Part3: Sentiment Analyzer using Keras Embedding

```
Loading dataset...
Preprocessing data...
Creating model...
```

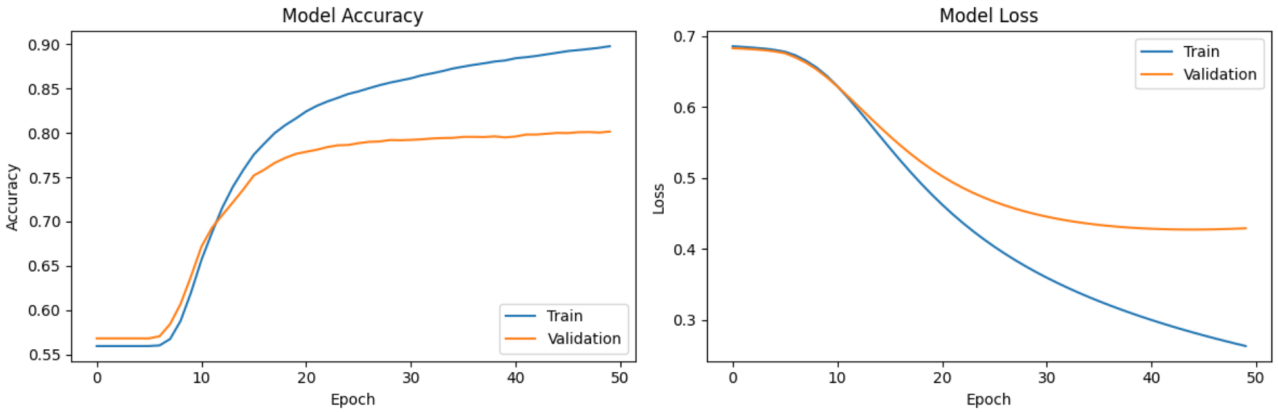
Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
global_average_pooling1d (GlobalAveragePooling1D)	?	0
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
Epoch 40/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8833 - loss: 0.3050 - val_accuracy: 0.7950 - val_loss: 0.4290
Epoch 41/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8857 - loss: 0.3000 - val_accuracy: 0.7960 - val_loss: 0.4283
Epoch 42/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8871 - loss: 0.2952 - val_accuracy: 0.7982 - val_loss: 0.4278
Epoch 43/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8891 - loss: 0.2906 - val_accuracy: 0.7982 - val_loss: 0.4274
Epoch 44/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.8905 - loss: 0.2861 - val_accuracy: 0.7991 - val_loss: 0.4272
Epoch 45/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8924 - loss: 0.2818 - val_accuracy: 0.8001 - val_loss: 0.4272
Epoch 46/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.8943 - loss: 0.2777 - val_accuracy: 0.7999 - val_loss: 0.4273
Epoch 47/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8952 - loss: 0.2737 - val_accuracy: 0.8008 - val_loss: 0.4275
Epoch 48/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8964 - loss: 0.2698 - val_accuracy: 0.8010 - val_loss: 0.4279
Epoch 49/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8977 - loss: 0.2661 - val_accuracy: 0.8004 - val_loss: 0.4284
Epoch 50/50
668/668 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.8996 - loss: 0.2624 - val_accuracy: 0.8016 - val_loss: 0.4290
```

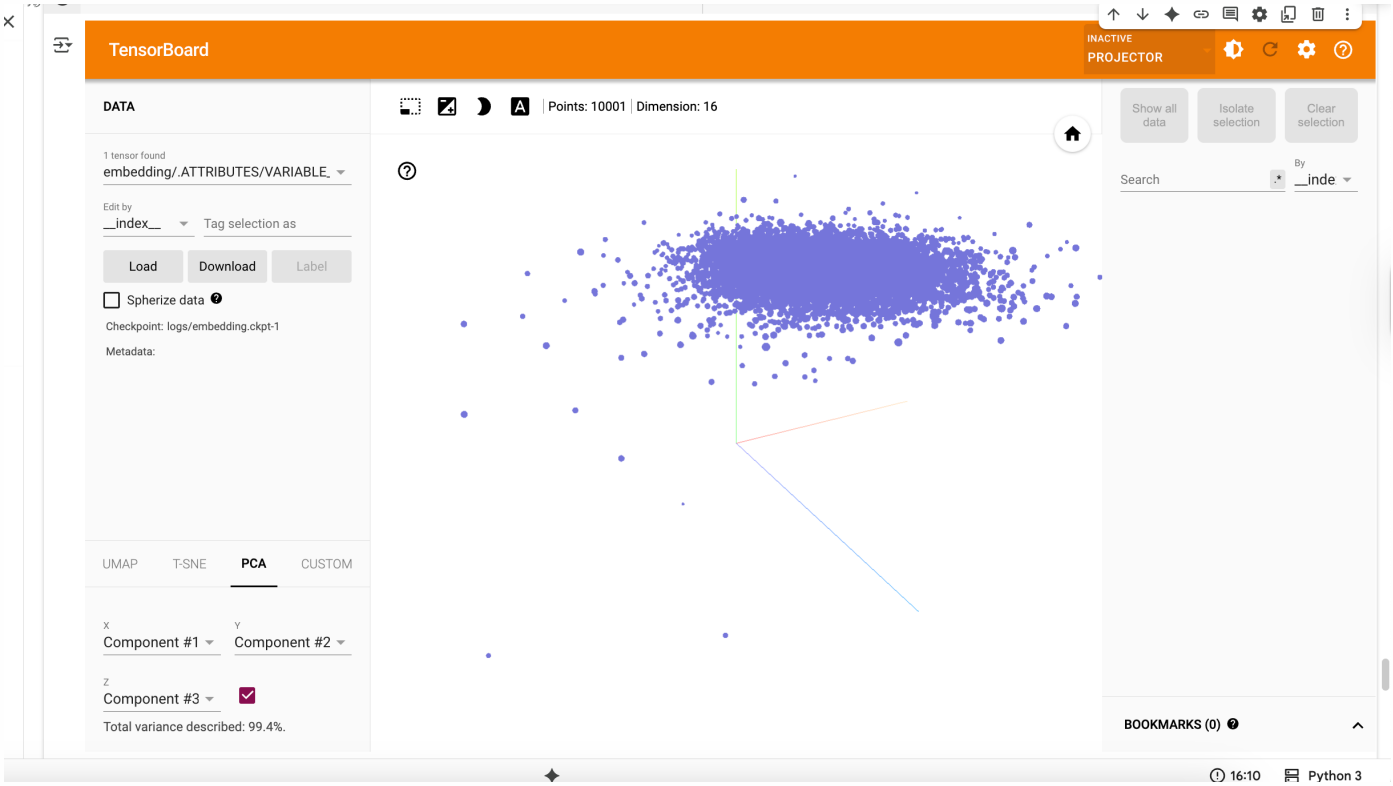
```
Evaluating model...
Validation loss: 0.4290
Validation accuracy: 0.8016

Embedding layer info:
Number of vectors in embedding layer: 10001
Embedding dimension: 16

Tokenizer info:
Vocabulary size: 29569
Num words parameter: 10000

Processed word counts:
Special tokens: 2
Regular words: 10000
Total metadata words: 10001

File verification:
Number of vector lines: 10001
Number of metadata lines: 10001
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step

Test predictions:
Headline: Scientists cure cancer, world peace achieved
Sarcasm probability: 0.2798

Headline: Local man wins lottery, plans to spend it all on cat food
Sarcasm probability: 0.9903

Headline: Breaking: Water is wet, scientists confirm
Sarcasm probability: 0.9048

Training performance: 0.8979266881942749
Validation performance: 0.8015724420547485
Predictions: [[0.2797567 ]
 [0.99030834]
 [0.90478855]]
TestResults(failed=0, attempted=3)
```

Part 4: Visualizations with TensorBoard Projector

Sorry, I don't quite understand this part. And here is the result of the TensorBoard:



Part 5: Reflections

1. Our sentiment analyzer uses an Embedding layer followed by Global Average Pooling rather than simpler techniques like Bag of Words or TF-IDF. What are the advantages of this approach for text classification, and what information about the text might be preserved or lost?

Ans: GlobalAveragePooling is a good way of reducing the dimension and preventing over-fitting. Also, if we only use the Bag of Words, the words are treated as one-hot features with no semantic connection.

2. In the code, we set random seeds in multiple places and disabled shuffling during training to ensure reproducibility. How might this affect the model's performance compared to a non-deterministic version? What are the tradeoffs between reproducibility and potential model robustness?

   Ans: If we set the random seeds, we would get the same data and result every time we run the code. It makes things easier if we want to compare different models. However, certain concerns may be raised: it might result in over-fitting of this specific set of data, or we may lack robustness evaluation. So it increases the reproducibility, but decreases the model robustness if we use the random seeds.

3. Our preprocessing includes removal of custom stopwords specific to headlines (like 'says', 'would', 'latest'). How do you think this affects the model's ability to detect sarcasm in headlines? Can you think of cases where removing these words might help or

hurt the model's performance?

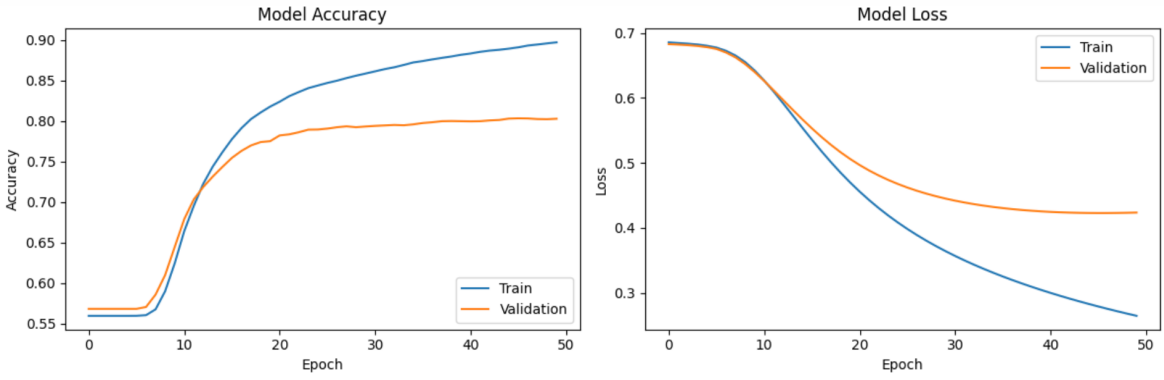
Ans: It may increase the ability if we remove the stopwords such as "says" and "latest". Since these words still have some sentimental meaning.

4. The model uses a relatively simple architecture (Embedding -> GlobalAveragePooling -> Dense -> Dense) with a small embedding dimension of 16. What considerations went into these architectural choices, and how might you modify the architecture for different types of text classification tasks?

   Ans: I think the design of the pipeline emphasizes the "simplicity". A change can be made to make the model more "complex": Instead of using the GlobalAveragePooling, we can use the RNN or Conv1D instead.

5. Experiment with the STOP_WORDS and CUSTOM_STOPS sets by removing and adding words. It is your choice which STOP_WORDS and/or CUSTOM_STOPS to modify but you must do at least 2 different modifications that show either some positive or negative effect. Retrain and evaluate your model. Compare your new validation accuracy and predictions with the original model's results. What patterns did you notice in headlines that were classified differently? How do these changes align with your understanding of how stopwords affect the signal-to-noise ratio in text classification?

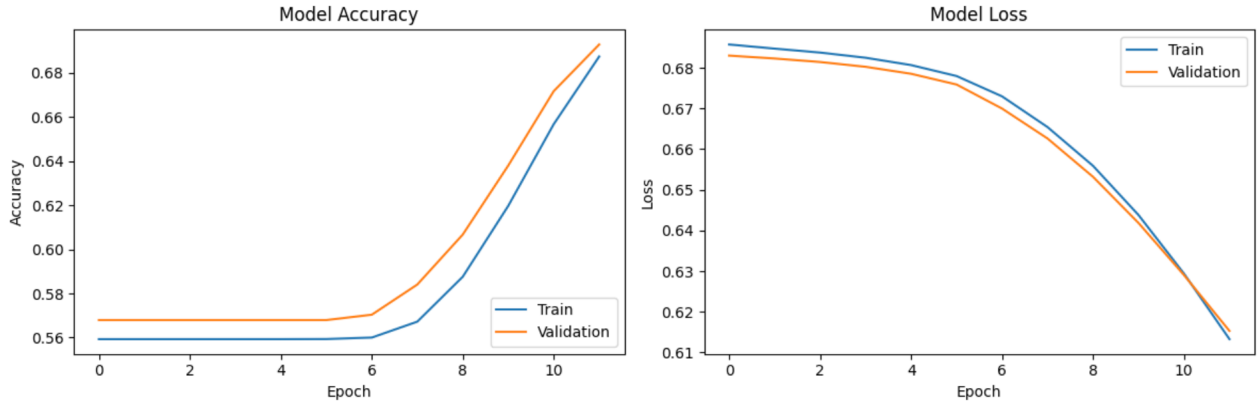
   Ans: I try to delete all other words except for "say" and "said" in the stop words, presuming it would greatly affect the model. The new functions as `preprocess_data2()` and `run_sentiment_analyzer2()`. And the result becomes:




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step

Test predictions:
Headline: Scientists cure cancer, world peace achieved
Sarcasm probability: 0.2783

Headline: Local man wins lottery, plans to spend it all on cat food
Sarcasm probability: 0.9922

Headline: Breaking: Water is wet, scientists confirm
Sarcasm probability: 0.9033

Training performance: 0.8973183035850525
Validation performance: 0.8028827905654907
Predictions: [[0.27831033]
 [0.99216896]
 [0.90325814]]
```

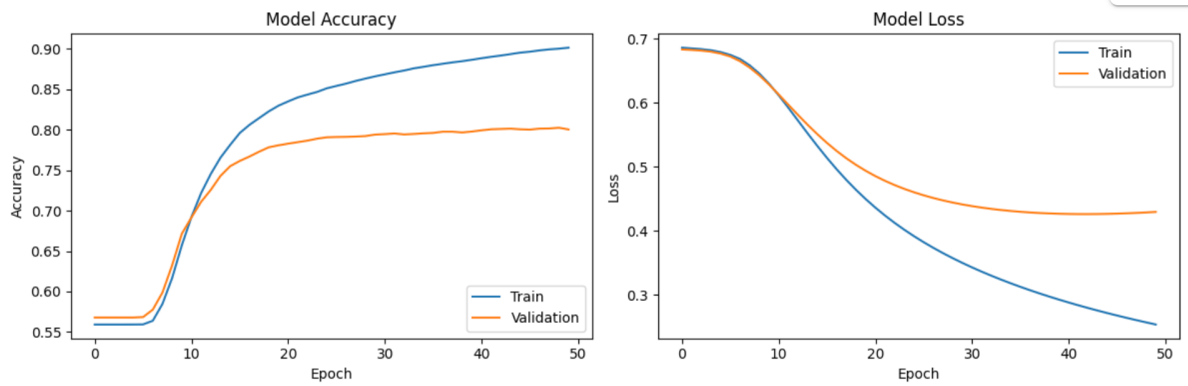
6. You should have (hopefully) noticed that the model tends to exhibit overfitting after about 20 epochs (the validation performance tends to level off while the training performance continues to improve). Experiment with different configuration parameters (MAX_LENGTH, VOCAB_SIZE, EMBEDDING_DIM, EPOCHS, BATCH_SIZE) and observe the effects after retraining and evaluating your model. It is your choice which parameters to modify, but you must do at least 2 different modifications that show either some positive or negative effect. Which parameter modifications had the most significant impact on reducing the gap between training and validation accuracy? Analyze why these parameters affect model generalization, supporting your answer with specific examples from your experiments.

   Ans: From the graphs, I think the "better" epochs should be around 12. Then I change the number of epochs from 50 to 12:




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step

Test predictions:
Headline: Scientists cure cancer, world peace achieved
Sarcasm probability: 0.4202

Headline: Local man wins lottery, plans to spend it all on cat food
Sarcasm probability: 0.6140

Headline: Breaking: Water is wet, scientists confirm
Sarcasm probability: 0.4605

Training performance: 0.6873683929443359
Validation performance: 0.6928116679191589
Predictions: [[0.42015898]
 [0.6140223 ]
 [0.46045303]]
```

I also try to change EMBEDDING_DIM from 16 to 20:

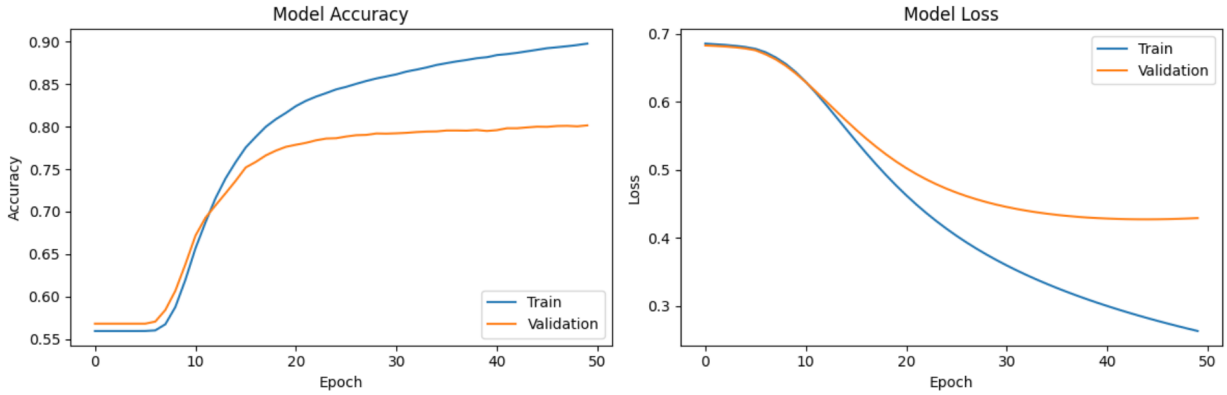



```
WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x78b55a41cc20> trig
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step

Test predictions:
Headline: Scientists cure cancer, world peace achieved
Sarcasm probability: 0.2897

Headline: Local man wins lottery, plans to spend it all on cat food
Sarcasm probability: 0.9915

Headline: Breaking: Water is wet, scientists confirm
Sarcasm probability: 0.9079

Training performance: 0.9014835953712463
Validation performance: 0.8004492521286011
Predictions: [[0.28966558]
 [0.99152935]
 [0.90793043]]
```

7. Experiment with the word frequency threshold (min_freq parameter in save_embeddings_for_projector function), which controls the minimum frequency threshold for words to be included in the embedding visualization. Observe the effects after retraining and evaluating your model. How did changing this parameter affect both the visualization insights and model performance? What does this tell us about the relationship between word frequency, embedding quality, and model understanding of sentiment/sarcasm? Support your analysis with specific examples from your experiments.

   Ans: I changed the min_freq from 5 to 2.

```
def save_embeddings_for_projector(model, tokenizer, log_dir='logs', min_freq=2):
    """Save embeddings for visualization in TensorBoard Projector"""
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step

Test predictions:
Headline: Scientists cure cancer, world peace achieved
Sarcasm probability: 0.2798

Headline: Local man wins lottery, plans to spend it all on cat food
Sarcasm probability: 0.9903

Headline: Breaking: Water is wet, scientists confirm
Sarcasm probability: 0.9048

Training performance: 0.8979266881942749
Validation performance: 0.8015724420547485
Predictions: [[0.2797567 ]
 [0.99030834]
 [0.90478855]]
TestResults(failed=0, attempted=3)
```

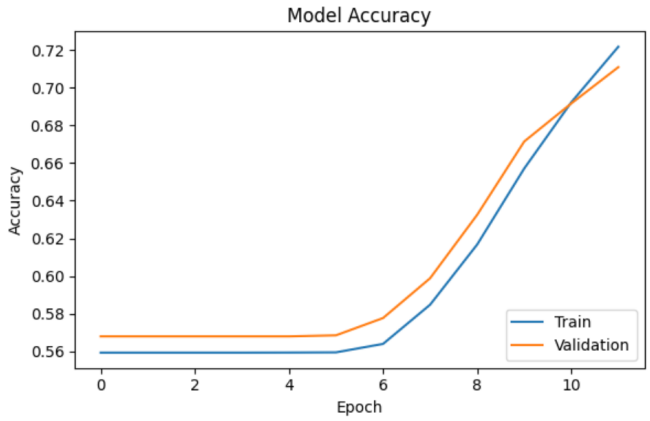
It seems like the result doesn't change much!

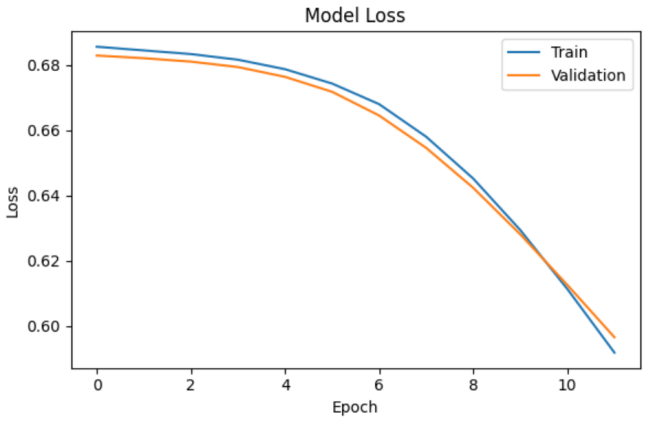
8. Finally, with your best model that you found after experimenting with the parameters in the previous questions, add a few (at least 3) new test headlines (search for "*test_-headlines*") to test your sarcasm detector.

Ans:

```
test_headlines = [
    "Scientists cure cancer, world peace achieved",
    "Local man wins lottery, plans to spend it all on cat food",
    "Breaking: Water is wet, scientists confirm",
    "Good news: the new administration has passed the new act.",
    "Bark or park: a discussion on building pet-friendly parks",
    "A car accident happened in New York City this Friday"
]
```

```
# Do NOT change these line
MAX_LENGTH = 100
VOCAB_SIZE = 10000
EMBEDDING_DIM =20
EPOCHS = 12
BATCH_SIZE = 32
```






```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
```

```
Test predictions:
Headline: Scientists cure cancer, world peace achieved
Sarcasm probability: 0.4140

Headline: Local man wins lottery, plans to spend it all on cat food
Sarcasm probability: 0.6597

Headline: Breaking: Water is wet, scientists confirm
Sarcasm probability: 0.4720

Headline: Good news: the new administration has passed the new act.
Sarcasm probability: 0.4513

Headline: Bark or park: a discussion on building pet-friendly parks
Sarcasm probability: 0.4745

Headline: A car accident happened in New York City this Friday
Sarcasm probability: 0.4306

Training performance: 0.7218140363693237
Validation performance: 0.7109696865081787
Predictions: [[0.41400567]
 [0.65969104]
 [0.472036  ]
 [0.45127556]
 [0.47447917]
 [0.43062624]]
```