Mehmet Ali Pelit
21113622

Question 2: Gradient Descent for Softmax Regression

Q2.2

```
First 10 rows of expected outputs:
[1 0 2 1 1 0 1 2 1 1]

First 10 rows of expected outputs in one-hot vector:
[[0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]]
```

Q2.3

```
Number of training inputs
3
```

Q2.4

```
0 3.7085808486476917
1000 0.1451936748083064
2000 0.1301309575504088
3000 0.12009639326384534
4000 0.11372961364786888
5000 0.11002459532472426

Final parameter:
[[ 0.41931626  6.11112089 -5.52429876]
 [-6.53054533 -0.74608616  8.33137102]
 [-5.28115784  0.25152675  6.90680425]]

 Validation accuracy:
0.9333333333333333
```

Mehmet Ali Pelit
21113622

Q2.5

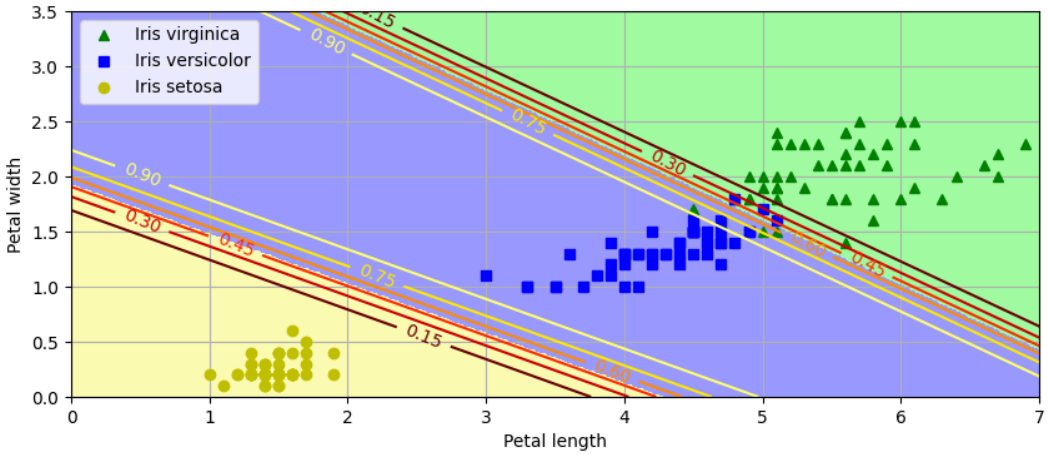


Figure 1: The results of the model

Question 4: Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks highly effective for analyzing visual imagery. In this exercise, you will work with an image dataset divided into two main folders: 'train' and 'test'. Within the 'train' dataset, there are two subfolders named 'polar' and 'not_polar', containing images of polar bears and images without polar bears, respectively. Your task will involve training a CNN to distinguish between these two categories.

Q4.1

Implement the `create_image_generators()` function to create image data generators for training and validation. Report the 'Training Generator Info' and 'Validation Generator Info' that you got when you executed `print_generator_info()` function.

```
Found 153 images belonging to 2 classes.
Found 37 images belonging to 2 classes.
Training Generator Info:
Number of images: 153
Batch size: 20
Class indices: {'test': 0, 'train': 1}
Number of classes: 2
Number of filenames loaded: 153

Validation Generator Info:
Number of images: 37
Batch size: 20
Class indices: {'test': 0, 'train': 1}
Number of classes: 2
Number of filenames loaded: 37
```

Q4.2

Build a convolutional neural network architecture using the sequential model approach mentioned in the assignment. Report the model summary and total number of parameters in the model you just built.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
flatten (Flatten)	(None, 82944)	0
dropout (Dropout)	(None, 82944)	0
dense (Dense)	(None, 512)	42,467,840
dense_1 (Dense)	(None, 1)	513

Total params: 42,487,745 (162.08 MB)
Trainable params: 42,487,745 (162.08 MB)
Non-trainable params: 0 (0.00 B)
None

Figure 1: The summary of the model

```
Total number of parameters: 42487745
```

Q4.3

To compile and train a Keras model, first, compile the model using the `compile` method, specifying the loss function as `binary_crossentropy`, the optimizer as `adam`, and tracking the `accuracy` metric. Then, train the model using the `fit` method, passing the `train_generator` for training data, setting the number of epochs to 20, and including the `validation_generator` for validation data. Plot the training and validation accuracies and losses. Does this model appear to be efficient at this stage?

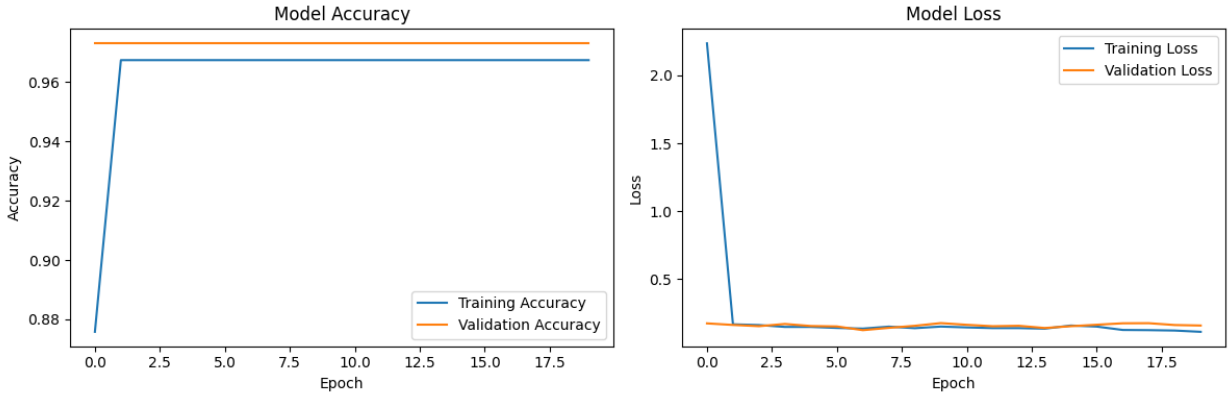


Figure 2: Training and validation accuracies and losses of the model

Does this model appear to be efficient at this stage?

Q4.4

There are just six images in the test folder. Load them one by one and infer by seeing what the model classifies. Report the predictions of the model. Which of the test images were

classified wrong and why did that happen and how can we address that?

Figure 3: The prediction of test_1.jpg: [[0.98639035]] Polar Bear

Figure 4: The prediction of test_2.jpg: [[0.7867882]] Polar Bear

Figure 5: The prediction of test_3.jpg: [[0.9510427]] Polar Bear

Figure 6: The prediction of test_4.jpg: [[0.82749814]] Polar Bear

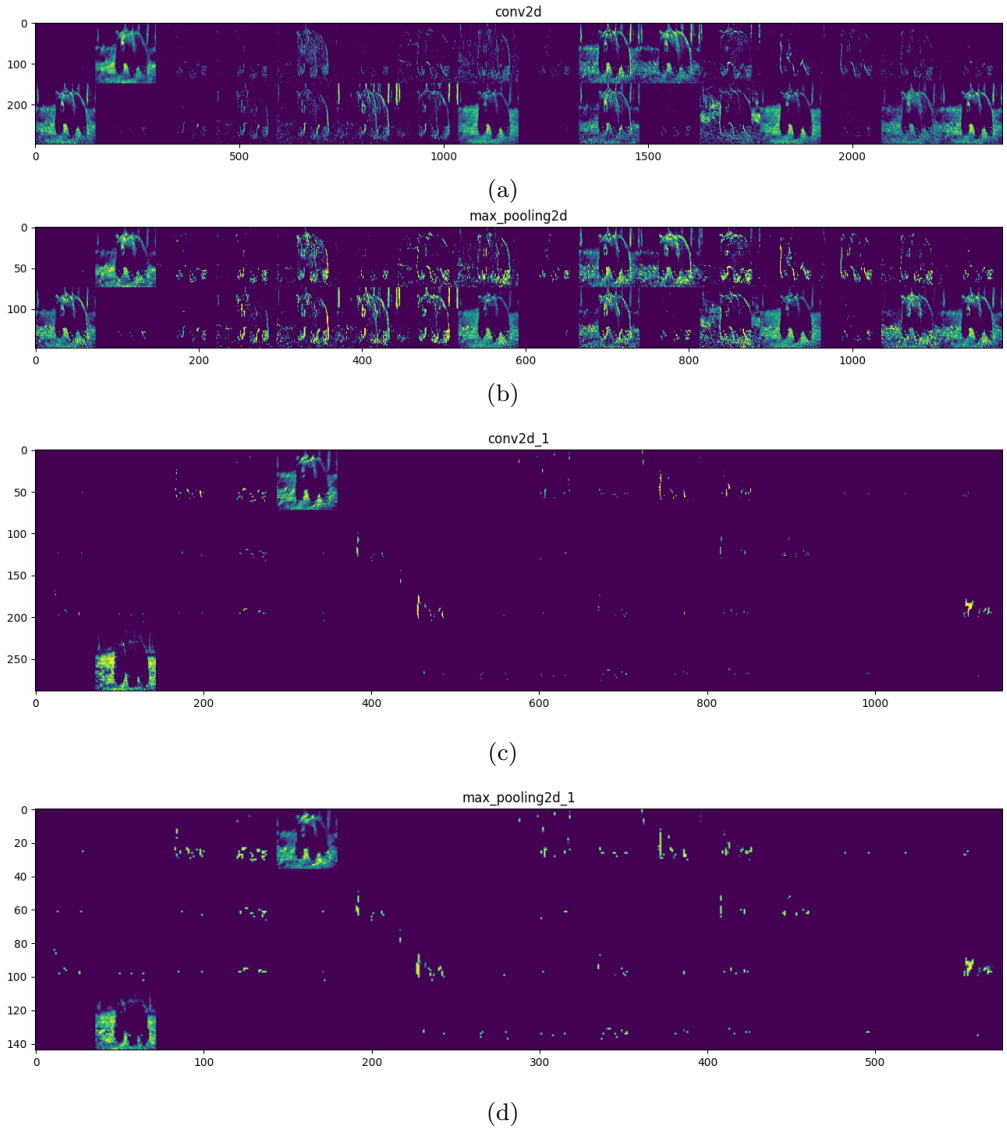
Figure 7: The prediction of test_5.jpg: [[0.8932706]] Polar Bear

Figure 8: The prediction of test_6.jpg: [[0.9828844]] Polar Bear

Which of the test images were classified wrong and why did that happen and how can we address that?

Q4.5

Define the get_layer_outputs function, begin by specifying its signature, which includes two parameters: 'model', representing the Keras model, and 'img_tensor', the input tensor for the model. Within the function, extract the output tensors for all layers up to the last MaxPooling layer (index 3) in the model. Next, create a new model named 'activation_model' using the 'Model' class from Keras, which takes the original model's input and the tensors extracted in the previous step as output. Finally, utilize the 'predict' method of the 'activation_model' to predict the activations for the input 'img_tensor', and return these activations as a list of numpy arrays, where each array corresponds to the activations of one layer. These activations are then used by display_layer_activations function. Upload the visualizations of the activations for test_4.png and comment on increasing abstractness as we go deeper into the model.



(a)

(b)

(c)

(d)

Figure 9: Visualizations of the activations for test_4.png

comment on increasing abstractness as we go deeper into the model.

Mehmet Ali Pelit
21113622

Question 5: word2vec

Create Python script for creating Word2vec from scratch by training a Continuous Bag of Words (CBOW) model using TensorFlow and Keras. Do not use the Gensim library. Set seeds to ensure reproducibility in your code. Use the same seed value (25) for all random number generators in your environment, including libraries like NumPy, TensorFlow, and Python's built-in random module.

Q5.1

Complete the `preprocess()` function by converting the text in lowercase and splitting it into words. What is the total number of words?

```
Number of words: 280000
```

Q5.2

Complete the `build_and_prepare_data()` function. What is the vocab size, number of contexts and number of targets?

```
Vocabulary size: 28
Length of contexts array: 279996
Length of targets array: 279996
```

Q5.3

Complete the `build_cbow_model()` function that constructs a Continuous Bag of Words (CBOW) model using TensorFlow and Keras libraries. Your function should return the constructed model. Print the model summary.

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 4)	0
embedding (Embedding)	(None, 4, 2)	56
lambda (Lambda)	(None, 2)	0
dense (Dense)	(None, 28)	84

Total params: 422 (1.65 KB)
Trainable params: 140 (560.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 282 (1.11 KB)

Figure 1: The summary of the model

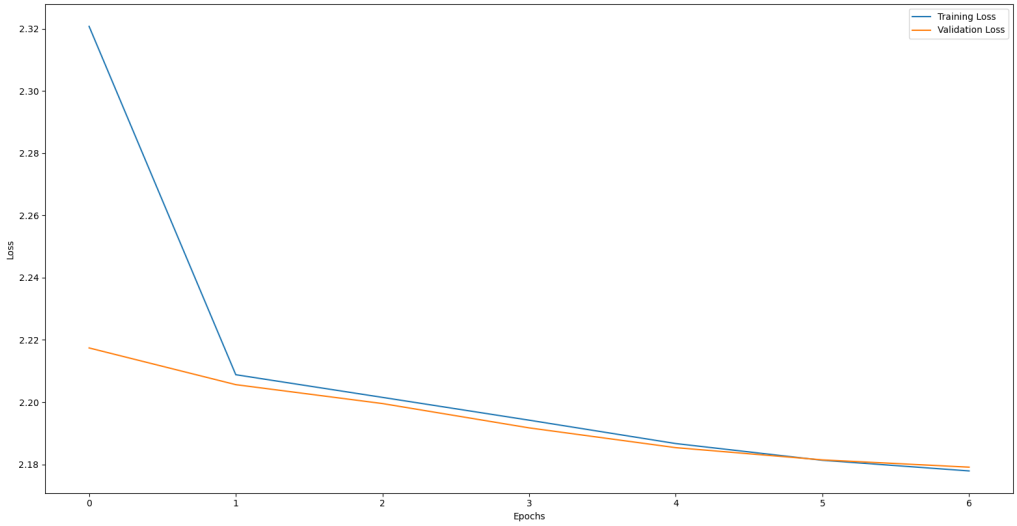


Figure 2: Training and validation loss of the model

Q5.4

Extract the embeddings from the embedding layer. Since the size the embedding size is 2, you should plot the embeddings and visualize them. Provide the plot in your submission.

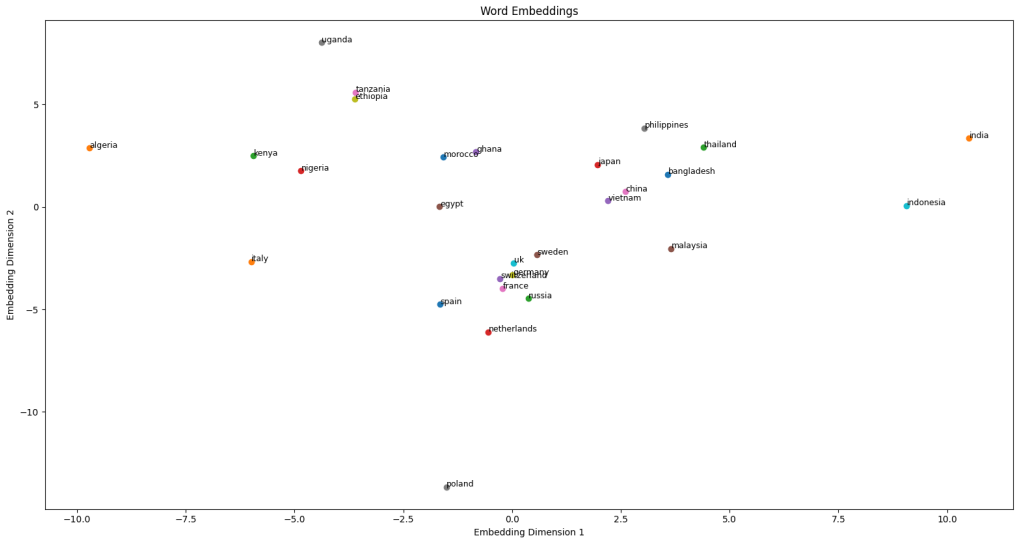


Figure 3:

Q5.5

Complete the two functions, `cosine_similarity()` and `find_similar_words()`, to analyze word similarities using vector embeddings. The `cosine_similarity()` function should measure how similar two vectors are based on their orientation. The `find_similar_words()` function should identify and return the most similar words to a given `query_word` from a set of word embeddings, ranking them by their similarity. Use these functions to find and return the top 3 similar countries for each of the following: Poland, Thailand, and Morocco.

```
Words most similar to 'sweden':
russia: 0.9877
uk: 0.9745
germany: 0.9719

Words most similar to 'thailand':
bangladesh: 0.9861
japan: 0.9745
india: 0.9620

Words most similar to 'uganda':
tanzania: 0.9974
morocco: 0.9973
ethiopia: 0.9947
```

Q5.6

Consider a small window size, e.g., 2 or 3. In this scenario, is there a possibility that antonyms (opposite words) might end up with similar embeddings? Explain your views.

Yes. The antonyms might be used in similar senteces. For someone who doesn't know the meaning of these words, like a simple model, might not realize the difference of these antonyms solely by looking at via small window. For example, "That person is good." and "That person is bad". If the model checks the words 'good' and 'bad' via a narrow window, it would conclude that these two words are similar.

Question 6: Next Word Prediction

"Alice in Wonderland" is a whimsical tale by Lewis Carroll about a young girl named Alice who falls through a rabbit hole into a fantastical world full of peculiar creatures and surreal adventures. Your task is to build a next-word prediction model, trained only on the Alice in Wonderland textual data.

Q6.1

For this task, you'll be working with a text file containing data that needs to be preprocessed for further analysis. Start by accessing the file from its location on your drive. Once opened, read its contents into a string, ensuring that the text is handled in a case-insensitive manner by converting it to lowercase. To remove punctuations in the text, apply a regular expression that filters out all characters that are not letters, digits, underscores, or whitespace (not \w and s in regex). This preprocessing step simplifies the text, making it uniform and easier to analyze in subsequent tasks. Print the length of the final processed text obtained.

```
Size of the text BEFORE preprocessing: 148574
Size of the text AFTER preprocessing: 140269
```

Q6.2

Initiate the process of text tokenization which is vital for preparing data for natural language processing models. Utilize the Tokenizer from the TensorFlow Keras library used for preparing text data for deep learning models to analyze the text and identify unique words. By fitting the tokenizer to the text, it constructs a comprehensive dictionary of these unique words. Subsequently, calculate the total number of unique words, which is essential for configuring various model parameters, such as input dimensions in neural networks. This total also includes an additional count to accommodate the tokenizer's indexing method. Print the total number of words.

```
Total number of unique words: 2750
Total number of words: 26410
```

Q6.3

In this task, you'll prepare input sequences for training by first splitting the preprocessed text on newline character and converting each line into a list of tokenized words. For each line, generate n-gram sequences of increasing length to create a comprehensive set of training samples. These n-grams, which consist of consecutive tokens, help the model learn contextual relationships within the data. After constructing these sequences, identify the maximum sequence length and standardize all sequences to this length using padding. This padding, typically added to the beginning of sequences, ensures that all input data fed into the model maintains a consistent format, crucial for effective training of sequence-based neural networks like LSTMs or RNNs. For the following steps print the number of input sequences finally created for the actual given text.

```
Count of input sequences: 26410
First 10 input sequences:
[[274], [274, 465], [274, 465, 11], [274, 465, 11, 682], [274], [274, 465],
 [274, 465, 11], [274, 465, 11, 682], [1472], [1472, 1473]]
First 10 padded sequences:
[[   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0  274]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0  274  465]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  274  465   11]
 [   0    0    0    0    0    0    0    0    0    0    0    0  274  465   11  682]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0  274]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0  274  465]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  274  465   11]
 [   0    0    0    0    0    0    0    0    0    0    0    0  274  465   11  682]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0 1472]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0 1472 1473]]
```

Q6.4

In this phase of preparing your data for machine learning models, you'll separate the previously formatted input sequences into predictors (features) and labels (targets). By slicing the sequences, the last token of each sequence becomes the label, while the preceding tokens form the predictors. Convert the label tokens into one-hot encoded vectors using TensorFlow's utility function, facilitating effective categorical output prediction. Subsequently, divide your dataset into training and validation subsets using a 20% split for validation. Print the size of the train and validation subsets for the features and targets.

```
Shape of Training Features: (21128, 15)
Shape of Training Labels: (21128, 2751)
Shape of Validation Features: (5282, 15)
Shape of Validation Labels: (5282, 2751)
```

Q6.5

Create a simple LSTM-based model by defining a sequential architecture. Begin with an embedding layer that uses the total number of words as the input dimension, and an output dimension of 100. Follow this with an LSTM layer containing 150 units. Then, add a dense layer with the total number of words as the output dimension, using the softmax activation function. Compile the model with categorical cross-entropy as the loss function, the Adam optimizer, and accuracy as the metric. After defining the model, print its summary. Build and train the model for 20 epochs. After training, visualize the performance by plotting the training and validation accuracy and loss over the epochs. Is the model overfitting? Explain your observation. Create one more model of your choice (you may explore Bidirectional, LayerNormalization, Dropout, Attention and GRU etc) that improves upon the previous model in terms of overfitting. Print this new model summary, train for 20 epochs, and then plot the training and validation accuracy and loss.

Mehmet Ali Pelit
21113622

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 15, 100)	275,100
lstm (LSTM)	(None, 150)	150,600
dense (Dense)	(None, 2751)	415,401

Total params: 2,523,305 (9.63 MB)
Trainable params: 841,101 (3.21 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,682,204 (6.42 MB)

Figure 1: The summary of the model

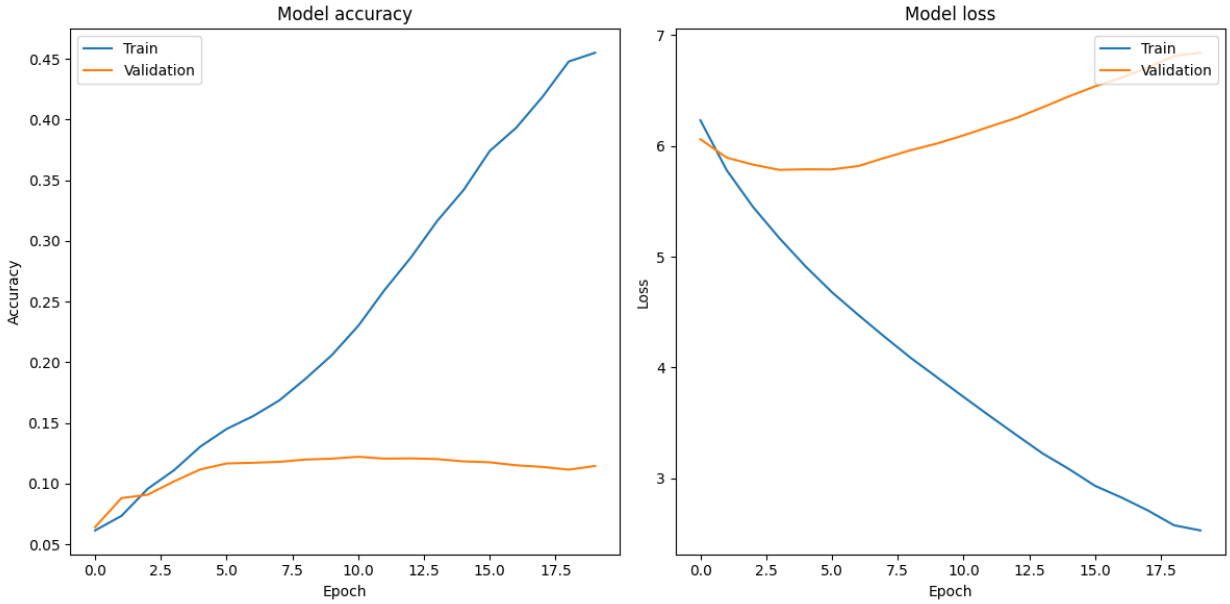


Figure 2: Performance of the model

Comment: Is the model overfitting? Yes. The validation accuracy is deviated from the training curve significantly in the model accuracy plot. That is, the model is overfitted. This was expected since the model is trained by a single text which has almost no trend between its words.

Let's design a modified model that includes Bidirectional LSTM and Dropout layers to potentially enhance performance and reduce overfitting compared to the previous model.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 15, 100)	275,100
bidirectional (Bidirectional)	(None, 300)	301,200
dense_1 (Dense)	(None, 2751)	828,051

Total params: 4,213,055 (16.07 MB)
Trainable params: 1,404,351 (5.36 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2,808,704 (10.71 MB)

Figure 3: The summary of the new model

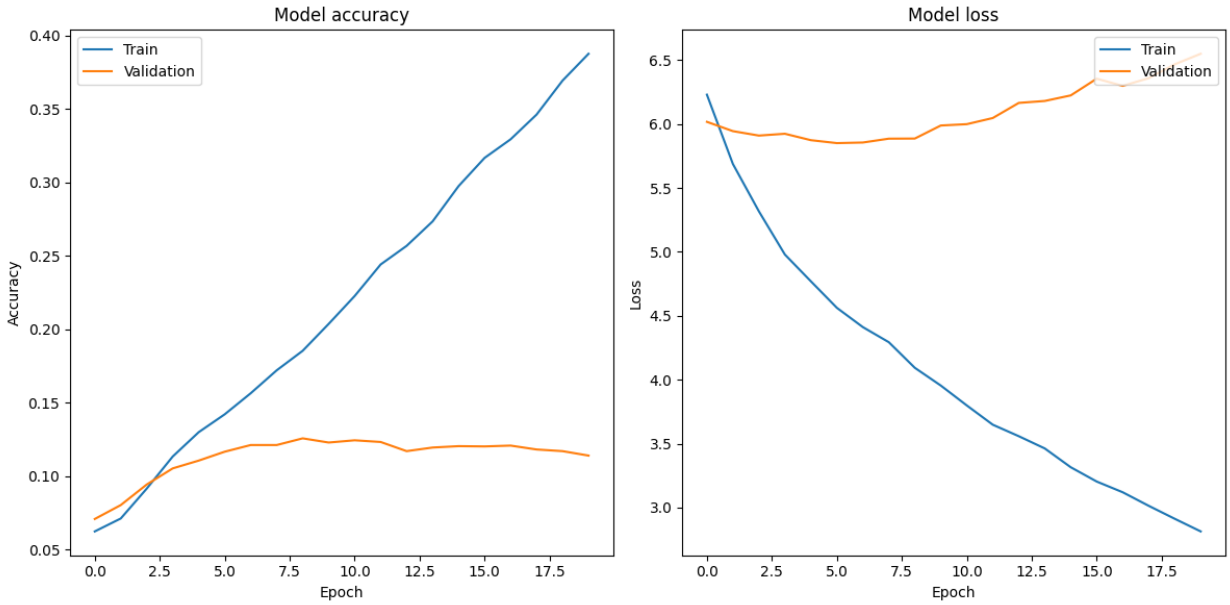


Figure 4: Performance of the new model

Q6.6

Define the `generate_text()` function that takes a starting text, the desired number of additional words, a predictive model, the maximum sequence length, and a temperature parameter as inputs. Demonstrate the function by generating text with temperature values 0.05 and 1.5 using the previously created model with less overfitting.

```
Generated text with temperature 0.05:
Forest is birthday fanning herself feeling a gave mystery the curtseying pray

Generated text with temperature 1.5:
Forest is brightened give squeaking twenty exactly treading slipped hoarsely
cautiously expression
```

Q6.7

In the preprocessing step for NLP, removing stop words is often considered important. We did not perform stop word removal in our text generation task. Should we have done that? Explain reasons to support your answer.

In generation tasks, removing the stopwords may cause loss of main context and structure of the text which is not wanted for our case since we need information as much as possible. However, it would be beneficial for classification tasks or keyword extraction practices because the stopwords might not contribute significantly to the meaning of the text.