TASK

Capstone Project III

Visit our website

Introduction

WELCOME TO THE THIRD CAPSTONE PROJECT TASK!

For this Capstone project, you are required to use recurrent neural networks in Keras to try and classify a movie review as either positive or negative.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

INTRODUCTION TO THE TASK!

This Capstone Project is about a basic form of **Natural Language Processing** (NLP) called **Sentiment Analysis**. For this task, you are required to use two different neural networks in Keras to try and classify a book review as either positive or negative, and report on which network type worked better.

For example, consider the review, "This book is not very good." This text ends with the words "very good" which indicates a very positive sentiment, but it is negated because it is preceded by the word "not", so the text should be classified as having a negative sentiment. We need to teach our neural network to recognise this distinction and be able to classify the review correctly.

This problem can be broken down into the following steps:

1. Get the dataset
2. Preprocessing the Data
3. Build the Model
4. Train the model
5. Test the Model
6. Predict Something

We will be working with real-world data in this task.

GET THE DATASET

For this task, we will be using a small portion of the **Multi-Domain Sentiment Dataset**, which contains product reviews from Amazon. The full dataset contains reviews for products under the categories: kitchen, books, DVDs, and electronics, but we will only be looking at reviews for the book category.

We have two files, **positive.txt** and **negative.txt**, containing the reviews. Each review is associated with a number of fields. The only field we are interested in is the “title” field, which contains the title of the review. We are going to use this title to predict the sentiment of the review. We could have used the review text itself, however, as this is a lot longer than the title, this is a much harder task.

For example, a review title might be “Horrible book”, whilst the review text might be “This book was horrible. If it was possible to rate it lower than one star I would have.”

Both the review title and the review text have the same sentiment — the title is just much more concise, which makes this task easier.

While some sentiments are easy to classify, like “don’t buy this horrible book”, others are less straightforward, like “'run don't walk to buy this book”. The latter is hard to classify because it contains the word “don’t”, which might be seen as an indication that this is a negative review, whereas actually, it is a positive one (the reviewer is suggesting that you should go as fast as you can to get the book). Thus, sentiment analysis is not always straightforward — some samples will be easy to classify, while others will not.

Some code is already included in the notebook associated with this task to start you off, which loads the relevant part of the data (the review heading) and performs some preliminary preprocessing to remove strange characters.

It also is necessary to create a vocabulary (called a text corpus) — words which our neural network will know and to “tokenise” the input. If we have a review, such as “a good book”, it is necessary to turn this into a form that a computer can understand. First, each word in the dataset is mapped to a unique number in the vocabulary. A word tokeniser will then take a sentence like this and convert it to a sequence of numbers, which map to the relevant words in the vocabulary.

Eg: “a good book” becomes an array of numbers: [1, 12, 3]. This mapping means that “a” is the first word in the vocabulary, “good” is the twelfth and “book” is the third. This mapping depends on the dataset supplied to the tokeniser.

The code for tokenisation is already included, but it is important that you understand what it does.

PREPROCESSING THE DATA

In order to feed this data into our network, all input reviews must have the same length. Since the reviews differ heavily in terms of lengths, we either need to trim or pad the reviews so that they are the same length. For this task, we will set the length of reviews to the mean length, which is around 4 words. If reviews are shorter than 4 words we will need to pad them with zeros, if they are longer than 4 words we will trim them to this length by cutting off any words after this. Keras offers a set of **preprocessing** routines that can do this for us. In order to pad our reviews, we will need to use the `pad_sequences` function.

BUILD THE MODEL

In the task today you will need to build a recurrent neural network to classify sentiment. The network will need to start with a special layer which will assist with text classification through a process called embedding.

Word embedding is a class of approaches for representing words and documents using dense vectors where a vector represents the projection of the word into a continuous vector space (Brownlee, 2017).

The position of a word within the vector space is learned from the text and is based on the words that surround the word when it is used. The position of a word in the learned vector space is referred to as its embedding.

Keras offers an **embedding layer**, used for neural networks on text data, and requires that the input data be integer encoded, so each word is represented by a unique integer (Brownlee, 2017). We have already achieved this format through tokenisation.

The embedding layer is trained as a part of the neural network and will learn to map words with similar semantic meanings to similar embedding-vectors. It is initialised with random weights and will learn an embedding for all of the words in the training dataset (Brownlee, 2017).

The Embedding layer is defined as the first hidden layer of a network. It must have three arguments (Keras Team, 2020):

- `input_dim:` This is the size of the vocabulary in the text data. For example, if your data is integer encoded to values between 0-5000, then the size of the vocabulary would be 5001 words.

- `output_dim`: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger. This is a hyper-parameter that needs to be tuned — test different values for your problem.

- `input_length`: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 4 words, this would be 4. This is the length which we padded/trimmed the inputs to during pre-processing.

Build a neural network, as outlined below.

- A recurrent neural network. This type of network is commonly used in NLP. This network should have the following architecture:

Embedding layer
SpatialDropout1D(0.2)
BatchNormalization()
LSTM(32)
Dense(2, activation='softmax')

TRAIN AND TUNE THE MODEL

You are now ready to train your model. Remember to compile your model by specifying the loss function and optimizer we want to use while training, as well as any evaluation metrics we'd like to measure — set the optimizer to 'adam' and the loss function to 'binary_crossentropy'.

Once compiled, you can start the training process. Note that there are two important training parameters that we have to specify, namely batch size and the number of training epochs. Together with our model architecture, these parameters determine the total training time.

For the network:

- Set the number of epochs to train for to 5 and batch size to 10.
- Tune the **output_dim** hyper-parameter of the embedding layer. Try values: 10, 25, 50 and 100. Report on the performance metrics for each value.
- Select the **output_dim** which gives the best performance on the test set and plot a graph of both the accuracy and loss of the model while training. Use these graphs to determine the point at which the model starts to overfit or if it has not yet converged. Identify a more optimal number of epochs to train for.
- You can also try tuning other metrics — such as batch size — to get the best possible performance.
- Report on the performance metrics of the final model.

MAKING PREDICTIONS

Finally, we would like to be able to use our model to predict something. To do this, we need to translate the sentence into the relevant integers and pad as necessary. This will allow us to put it into our model and see whether it predicts if we will like or dislike the book. A small selection of samples has been provided to get you going — you are welcome to add to this.

Compulsory Task

A notebook is associated with this task, which contains some useful functions/code to get you started.

Follow these steps:

- Use the provided files **positive.txt** and **negative.txt** and follow the above steps to train a recurrent neural network. Your goal is to classify the sentiment of book reviews as positive or negative.
- Note: Some blocks of code are labelled do not modify — make sure that the code in these blocks is left alone, or else you will encounter issues. Do read through it and see if you can follow what it does.
- When complete, push this project to GitHub. Include a detailed README file in which you describe your project in detail.

Optional Task

- Repeat the task for a different type of neural network: a fully connected network (starting with an embedding layer) of the form:

Embedding layer
Dropout(0.2)
BatchNormalization()
Flatten()
Dense(32, activation='relu')
Dense(2, activation='softmax')

Completed the task(s)?

Ask your mentor to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask your mentor for alternative instructions.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

References:

Brownlee, J. (2017). How to Use Word Embedding Layers for Deep Learning with Keras. Retrieved 24 August 2020, from **https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/**

Ebermann, T. (2018). Sentiment detection with Keras, word embeddings and LSTM deep learning networks · Blog · Liip. Retrieved 24 August 2020, from **https://www.liip.ch/en/blog/sentiment-detection-with-keras-word-embeddings-and-lstm-deep-learning-networks**

Keras Team (2020). Keras documentation: Embedding layer. Retrieved 24 August 2020, from **https://keras.io/api/layers/core_layers/embedding/#embedding**