TASK

Recurrent Neural Networks

Visit our website

Introduction

WELCOME TO THE RECURRENT NEURAL NETWORKS TASK!

In this task, you will be learning about Recurrent Neural Networks as well as how to implement them using Keras.

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.

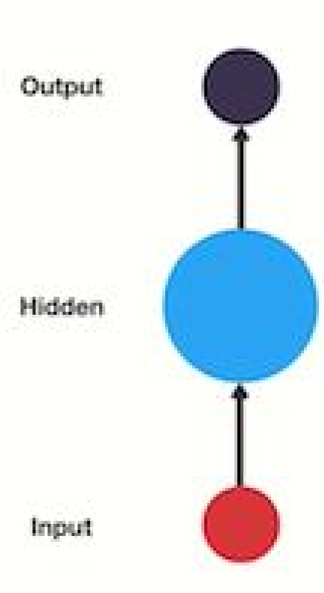
The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

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!

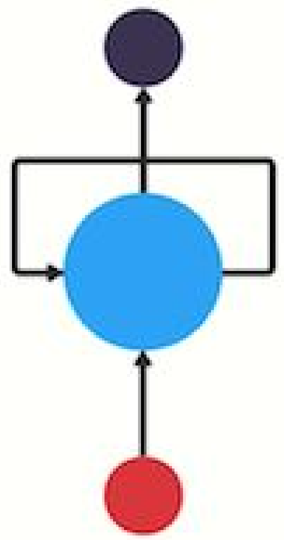
WHAT IS A RECURRENT NEURAL NETWORK?

The previous types of networks we have looked at have taken inputs of a particular fixed-size, ie. the number of features we pass to a regular feedforward network or the shape of the input images in CNNs. In real life, there are many examples where we would like to be able to make predictions where the input is a **sequence**, rather than a fixed-sized vector. Sequences can vary in length.

Examples of this are time-series data, such as natural language processing (NLP). If we consider the sentences "I had washed my car" and "I had my car washed", although we see that the same words occur in each, the meaning of the sentence is entirely different due to the order of the words. Therefore, we say the whole sequence of words gives us context for its meaning. A regular feed-forward neural network would only look at the individual words, whereas a **recurrent neural network (RNN)** would look at the entire sequence. This means that although it still processes the sentence word by word, an RNN retains a memory of the preceding elements, which is used along with the regular input to produce an output.



Feedforward Neural Network



Recurrent Neural Network

Image source: (Phi, 2018)

This means that at each layer in the network, the input to the layer is both that coming as the output from the previous layer (as in regular feed-forward networks) and the information that the layer 'remembers' from the last element in the sequence that passed through. If you consider the picture above, the additional arrow in the RNN image shows that the node is receiving a ‘memory’ of the previous elements when it passed through this node. We call this ‘memory’ the **state**, and the state is updated each time we move onto a new element in the sequence to summarise the elements it has seen before.

In an RNN, we thus advance the idea of a neuron to that of a **cell**, where we define a cell simply to be a neuron that has a state.

A popular variant of the RNN makes use of cells called **Long Short-Term Memory (LSTM)** cells. This variant of a cell has both the cell state and something called a ‘carry’. This carry means that if any context is needed much later in the sequence, it is still accessible and is not lost as its ‘memory fades’ (the gradient diminishes). This means that LSTMs have good long-term memory.

Gates allow optional additional information to adjust the cell state. LSTM cells have three different types of gate; an “input” gate, “forget” gate and an “output” gate. These gates allow the cell to retain or forget information as needed.

IMPLEMENTING A RECURRENT NEURAL NETWORK WITH KERAS

Images can also be considered to be sequences, as they are sequences of pixels. As an example, we will now use an RNN to predict MNIST. The tutorial in this section is adapted from **here**.

We start by importing the libraries we need. Note that we can import LSTM layers from Keras, along with the other layer types.

```
import tensorflow as tf
from tensorflow import keras

from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.datasets import mnist
```

As when we built our CNN to classify MNIST, we load and prepare the dataset for training as follows.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
X_train = X_train/255.0
X_test = X_test/255.0

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Instead of passing the entire image at once as we did with the CNN, or flattening the image into a one-dimensional vector as we did with the regular feed-forward network, this time we will pass in each row of the image as a sequence. Hence we will have 28 sequences of 28 elements (the images are of size 28 x 28 pixels).

We can now create our model and add our layer. This should all be straight forward. Rather than Dense or Convolutional layers, we're now just using LSTM as the layer type. The only new thing is **`return_sequences`**. This flag is used for when you're continuing on to another recurrent layer. If you are, then you want to return sequences. If you're not going to another recurrent-type of layer, then you don't set this to true. Dropout layers are also added to prevent overfitting to the training data.

```
model = Sequential()

# Recurrent layer
model.add(LSTM(128, input_shape=(X_train.shape[1:]), activation='relu',
return_sequences=True))
model.add(Dropout(0.2))

# Recurrent layer
model.add(LSTM(128, activation='relu'))
model.add(Dropout(0.1))

# Fully connected layer
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

Once the model is defined, it is then compiled and trained.

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)
```

Instructions

The exercises in these tasks require users to run Jupyter Notebooks through the Anaconda environment. Please make sure that you download the Anaconda Distribution from **https://www.anaconda.com/distribution/**.

Compulsory Task

Launch Jupyter Notebook via your Anaconda environment and upload the file named **rnn_task.ipynb** from your task folder and follow the instructions.

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 set up 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:

Phi, M. (2018). Illustrated Guide to Recurrent Neural Networks. Retrieved 28 August 2020, from **https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9**