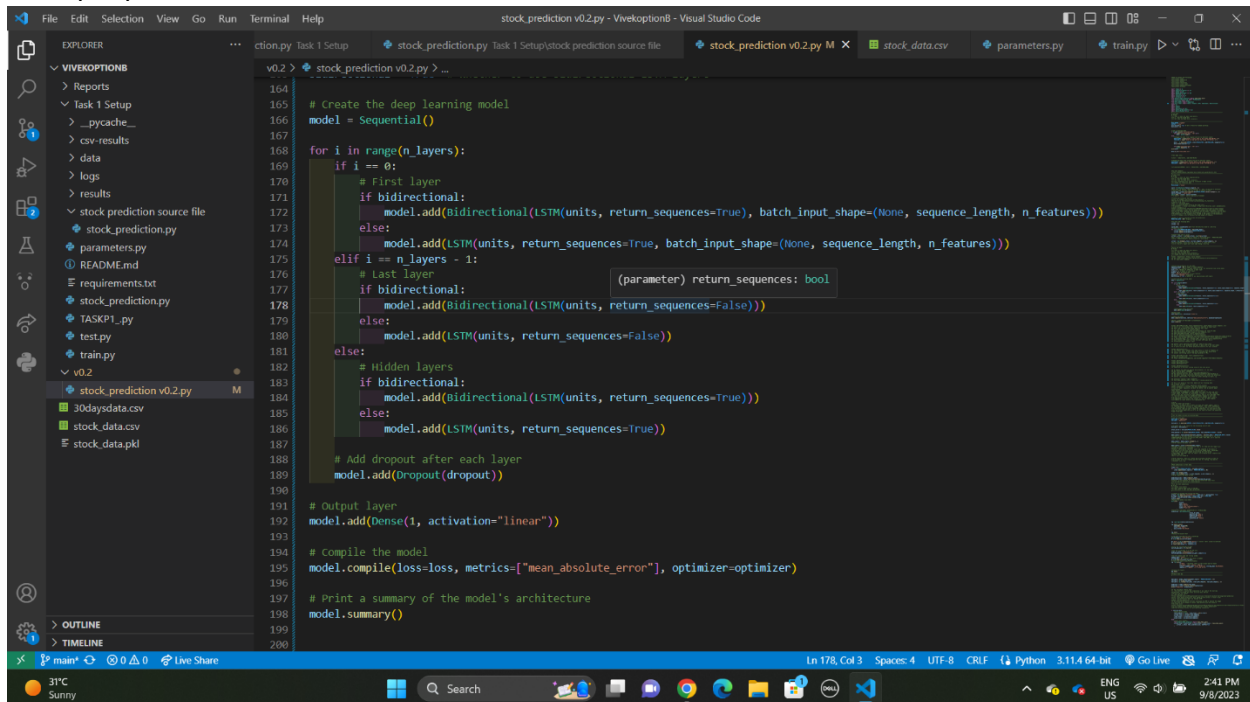


COS30018 - Option B - Task 4: Machine Learning 1

Alright lets start again with this project. I will explain what I have changed in this project to make the new version that is v0.3. I will try to keep it short, to the point and crisp.

Write a function that takes as input several parameters including the number of layers, the size of each layer, the layer name and return a Deep Learning model. Again, our reference project (P1) will give you an example of how this can be done. You can reuse this example, extend it, and most importantly, explain in detail all the code in your program.

For the first step , I have changed the existing model with the one I have created by using the example present in P1.



```
164
165 # Create the deep learning model
166 model = Sequential()
167
168 for i in range(n_layers):
169     if i == 0:
170         # First layer
171         if bidirectional:
172             model.add(Bidirectional(LSTM(units, return_sequences=True), batch_input_shape=(None, sequence_length, n_features)))
173         else:
174             model.add(LSTM(units, return_sequences=True, batch_input_shape=(None, sequence_length, n_features)))
175     elif i == n_layers - 1:
176         # Last layer
177         if bidirectional:
178             model.add(Bidirectional(LSTM(units, return_sequences=False)))
179         else:
180             model.add(LSTM(units, return_sequences=False))
181     else:
182         # Hidden layers
183         if bidirectional:
184             model.add(Bidirectional(LSTM(units, return_sequences=True)))
185         else:
186             model.add(LSTM(units, return_sequences=True))
187
188 # Add dropout after each layer
189 model.add(Dropout(dropout))
190
191 # Output layer
192 model.add(Dense(1, activation="linear"))
193
194 # Compile the model
195 model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)
196
197 # Print a summary of the model's architecture
198 model.summary()
199
200
```

This LSTM model is built based on user-specified parameters, such as sequence length, number of input features, number of LSTM units in each layer, number of LSTM layers, dropout rate, loss function, optimizer, and whether to use bidirectional LSTM layers.

- The `sequence_length` determines the number of previous days' stock prices to consider as input for predicting the next day's price. This is a hyperparameter that you can adjust.
- The `n_features` specifies the number of input features. In this case, it's set to 1 since the input is the closing price of the stock.
- The `units` parameter specifies the number of LSTM units (neurons) in each layer.
- The `n_layers` parameter specifies how many LSTM layers are stacked in the model.
- The dropout rate is a regularization technique to prevent overfitting by randomly dropping a fraction of neurons during training.
- The loss function is used to measure the model's error during training, and "mean_absolute_error" is chosen as the loss function for this code.
- The optimizer is the optimization algorithm used during training. "rmsprop" is selected as the optimizer.
- The `bidirectional` parameter determines whether to use bidirectional LSTM layers, which can capture information from both past and future time steps.

The LSTM model is built layer by layer within a loop, where each layer consists of LSTM units followed by dropout layers. The last layer is a dense layer with a linear activation function, responsible for the final prediction

For the information related to the terms I have used info from this website:

<https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/>

Now I have used the example and did some tweaks to match my code from the one that is given in P1. I have first used hardcoded values to fill in the hyperparameters and check that whether my code is working or not. After successfully checking the code was working. I changed the code lines so that the parameters takes the user inputs. The errors I faced was with the inputs and had to parse the data into integers and float as required.

Then after executing the code the summary would look like this:

```
Enter the dropout rate (for example 0.6):
0.6
2023-09-08 15:17:07.750634: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use
ns in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in other operations, rebuild Tensor
e compiler flags.
Model: "sequential"

Layer (type)                 Output Shape                 Param #
=====
bidirectional (Bidirection   (None, 60, 128)             33792
al)

dropout (Dropout)            (None, 60, 128)             0
bidirectional_1 (Bidirecti   (None, 128)                 98816
onal)

dropout_1 (Dropout)          (None, 128)                 0
dense (Dense)                (None, 1)                   129
=====
Total params: 132737 (518.50 KB)
Trainable params: 132737 (518.50 KB)
Non-trainable params: 0 (0.00 Byte)

2/2 [=====] - 1s 5ms/step
1/1 [=====] - 0s 27ms/step
Prediction: [[30.25699]]
PS C:\Semester 3\Intelligent systems\Gitrepo\VivekoptionB> |
```

I have also added comments in respectively to tell what line of code does what like how many layers are there and what does it accomplish.

By creating this model, it gives me a good grasp on how these deep learning models actually work and how we can manipulate the input data to make it different. For example , I changed the loss method and fed in different values and experimented. I took the reference to understand the concepts from here : <https://builtin.com/data-science/loss-functions-deep-learning-python>

Moving on to the Second task:

Use the above function to experiment with different DL networks (e.g., LSTM, RNN, GRU, etc.) and with different hyperparameter configurations (e.g. different numbers of layers and layer sizes, number of epochs, batch sizes, etc.

I was quite unsure about this task, so to make this one I made another function which would add GRU layers and add it to the learning model to get the output. It was really similar to the model that I made , instead of writing LSTM all had to do was replace it with GRU, import the

GRU library from keras library. But after this I had some errors in the code due to bad indentation and to make it right I had to give it some time and rewrite that portion where I had to define it first and also did a couple of tweaks here and there. I used the same methods to implement the addition of layers similar to the previous model.

```
169 select = int(input("Press 1 for LSTM Model and press 2 for GRU Model:"))
170
171 def create_gru_model(sequence_length, n_features, units=64, n_layers=2, dropout=0.3,
172                       loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False):
173     model = Sequential()
174
175     for i in range(n_layers):
176         if i == 0:
177             # First layer
178             if bidirectional:
179                 model.add(Bidirectional(GRU(units, return_sequences=True), batch_input_shape=(None, sequence_length, n_features)))
180             else:
181                 model.add(GRU(units, return_sequences=True, batch_input_shape=(None, sequence_length, n_features)))
182         elif i == n_layers - 1:
183             # Last layer
184             if bidirectional:
185                 model.add(Bidirectional(GRU(units, return_sequences=False)))
186             else:
187                 model.add(GRU(units, return_sequences=False))
188         else:
189             # Hidden layers
190             if bidirectional:
191                 model.add(Bidirectional(GRU(units, return_sequences=True)))
192             else:
193                 model.add(GRU(units, return_sequences=True))
194
195     # Add dropout after each layer
196     model.add(Dropout(dropout))
197
198     # Output layer
199     model.add(Dense(1, activation="linear"))
200
201     # Compile the model
202     model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)
203
204     return model
205
```

This the GRU model function.

At last, to make it work we have already input the values of hyperparameters . I put up an if else condition to select which DL model we can use.

```
39     # Print a summary of the model's architecture
40     model.summary()
41 elif select==2 :
42     model = create_gru_model(sequence_length, n_features, units=64, n_layers=2, dropout=0.3,
43                             loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False)
44 else:
45     print("Invalid selection. Please choose 1 for LSTM or 2 for GRU.")
46     exit()
47
48
49
```

After selecting the options it executes and gives us the summary as well.

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 60, 64)	12864
dropout (Dropout)	(None, 60, 64)	0
gru_1 (GRU)	(None, 64)	24960
dropout_1 (Dropout)	(None, 64)	0
dense (Dense)	(None, 1)	65

```

=====
Total params: 37889 (148.00 KB)
Trainable params: 37889 (148.00 KB)
Non-trainable params: 0 (0.00 Byte)
=====
16/16 [=====] - 0s 5ms/step
1/1 [=====] - 0s 18ms/step
Prediction: [[84.75643]]
PS C:\Semester 3\Intelligent systems\Gitrepo\VivekoptionB>

```

I hope this is all required for now that satisfies creating a new deep learning model , testing out with another DL Network and taking the parameters that can be changed by the user via input function.

In conclusion I have learnt about the working of different neural network how we can change the layers and also add different layers to our learning model.

NOTE: I also tried making a separate model with different types of layers like LSTM,GRU, ANN, DROPUTS and tried to merge it into a single one but it was getting way too complex and not working at all so I had to scrap that idea. So in the end , I think this satisfies the requirements and will wait for the feedback :)