TASK 4 REPORT

COS30018

INTELLIGENT SYSTEMS

Halim Vlahos (104015943)

CREATE MODEL SUMMARY

The Create Model function was modified from the existing model creation code from v0.1, but refactored in an adjustable way with function parameters.

```python
def create_dl_model(layer_type='LSTM', num_layers=3, units=50, dropout=0.2, input_shape=(60, 1)):
    """
    Creates a deep learning model with provided parameters

    :param layer_type: The type of layer (LSTM, GRU, RNN).
    :param num_layers: Number of layers in the model.
    :param units: Number of units/neurons per layer.
    :param dropout: Dropout rate to prevent overfitting.
    :param input_shape: Shape of the input data.
    :return: Compiled deep learning model.
    """
    model = Sequential()
    layer_map = {
        'LSTM': LSTM,
        'GRU': GRU,
        'RNN': SimpleRNN
    }

    Layer = layer_map[layer_type]

    # Add input layer
    model.add(Layer(units=units, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(dropout))

    # Add middle layer stack
    for _ in range(1, num_layers - 1):
        model.add(Layer(units=units, return_sequences=True))
        model.add(Dropout(dropout))

    # Add the final layer
    model.add(Layer(units=units))
    model.add(Dropout(dropout))

    # Add output layer
    model.add(Dense(units=1))

    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

```python
# Build the Model
## TO DO:
# 1) Check if data has been built before.
# If so, load the saved data
# If not, save the data into a directory
# 2) Change the model to increase accuracy?
#------------------------------------------------------------------------------
model = Sequential() # Basic neural network
# See: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
# for some useful examples

model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
# This is our first hidden layer which also spcifies an input layer.
# That's why we specify the input shape for this layer;
# i.e. the format of each training example
# The above would be equivalent to the following two lines of code:
# model.add(InputLayer(input_shape=(x_train.shape[1], 1)))
# model.add(LSTM(units=50, return_sequences=True))
# For som eadvances explanation of return_sequences:
# https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/
# https://www.dlology.com/blog/how-to-use-return_state-or-return_sequences-in-keras/
# As explained there, for a stacked LSTM, you must set return_sequences=True
# when stacking LSTM layers so that the next LSTM layer has a
# three-dimensional sequence input.

# Finally, units specifies the number of nodes in this layer.
# This is one of the parameters you want to play with to see what number
# of units will give you better prediction quality (for your problem)

model.add(Dropout(0.2))
# The Dropout layer randomly sets input units to 0 with a frequency of
# rate (= 0.2 above) at each step during training time, which helps
# prevent overfitting (one of the major problems of ML).

model.add(LSTM(units=50, return_sequences=True))
# More on Stacked LSTM:
# https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))

model.add(Dense(units=1))
# Prediction of the next closing value of the stock price

# We compile the model by specify the parameters for the model
# See lecture Week 6 (COS30018)
model.compile(optimizer='adam', loss='mean_squared_error')
# The optimizer and loss are two important parameters when building an
# ANN model. Choosing a different optimizer/loss can affect the prediction
# quality significantly. You should try other settings to learn; e.g.

# optimizer='rmsprop'/'sgd'/'adadelta'/...
# loss='mean_absolute_error'/'huber_loss'/'cosine_similarity'/...

# Now we are going to train this model with our training data
# (x_train, y_train)
model.fit(x_train, y_train, epochs=25, batch_size=32)
# Other parameters to consider: How many rounds(epochs) are we going to
# train our model? Typically, the more the better, but be careful about
# overfitting!
# What about batch_size? Well, again, please refer to
# Lecture Week 6 (COS30018): If you update your model for each and every
# input sample, then there are potentially 2 issues: 1. If you training
# data is very big (billions of input samples) then it will take VERY long;
# 2. Each and every input can immediately makes changes to your model
# (a souce of overfitting). Thus, we do this in batches: We'll look at
# the aggreated errors/losses from a batch of, say, 32 input samples
# and update our model based on this aggregated loss.
```

The function accepts parameter inputs for layer type/name, which comprises of LSTM, GRU, and RNN. Other parameters were the number of layers, units (aka nodes/neurons), dropout ratio, and input shape.

The function sets the model to Sequential, and maps layer name onto its respective type (from tensorflow.keras.layers). Then it proceeds to add the required layers, input layer, middle stack (in the loop), final layer, and output layer (dense). The Dropout layers are required to prevent overfitting, by setting input units to zero at the rate of the dropout ratio (default 0.2).

The model is then compiled and returned.

I used the above function to experiment with the different layer type models, and different number of layers and units.

```
#-------------------------------------------------------
#Experimenting Models
#-------------------------------------------------------
# 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.)



# Create LSTM model
model = create_dl_model(layer_type='LSTM', num_layers=3, units=50, dropout=0.2, input_shape=input_shape)
# Train model
model.fit(x_train, y_train, epochs=25, batch_size=32)
print(f"Finished training LSTM model\n")

# Create GRU model
model = create_dl_model(layer_type='GRU', num_layers=3, units=50, dropout=0.3, input_shape=input_shape)
# Train model
model.fit(x_train, y_train, epochs=25, batch_size=32)
print(f"Finished training GRU model\n")

# Create RNN model
model = create_dl_model(layer_type='RNN', num_layers=4, units=100, dropout=0.2, input_shape=input_shape)
# Train model
model.fit(x_train, y_train, epochs=25, batch_size=32)
print(f"Finished training RNN model\n")

```