Task 4: Machine Learning 1

Muhammad Ali, 103960437, COS30018, 08/09/2024, Tutor: Ru Jia.

Requirements / Deliverables –

- Implement a function that constructs deep learning (DL) models based on user-defined parameters, such as the number of layers, types of layers (e.g., LSTM, GRU), and their configurations.
- The purpose was to provide a flexible tool for experimenting with various DL architectures without manually coding each model variation.

Result –

Implementation Steps:

Function Design:

- The create_dl_model function was designed to dynamically create deep learning models using the Keras. It accepts multiple parameters like layer_types, layer_sizes, dropout_rates, return_sequences, and activation_functions, enabling the construction of custom architectures depending on user input.

Dynamic Layer Configuration:

A key feature is the loop that adds layers iteratively based on the input configuration:

```
for i in range(n_layers):
    layer_type = layer_types[i].upper()
    units = layer_sizes[i]
    dropout_rate = dropout_rates[i] if i < len(dropout_rates) else 0.0
    return_seq = return_sequences[i]
    activation = activation_functions[i]
```

- **Challenge**: Ensuring that each layer type (LSTM, GRU, Dense) was correctly configured based on the input parameters required careful validation and alignment of list lengths to avoid index errors.

Dropout layers were conditionally added to mitigate overfitting, especially important for time series models where recurrent layers can easily overfit. Deciding the optimal placement and rate for Dropout required balancing regularization without excessively hindering learning capacity.

- **Reference**: Regularization techniques were reviewed from Towards Data Science.

Error Handling:

- The function includes error handling to catch unsupported layer types and mismatched configurations:

```
• else:
      raise ValueError(f"Unsupported layer type: {layer_type}")
```

Provides clear feedback to guide the user when incorrect parameters are passed.

Layer Creation Logic:

This block dynamically configures each specified layer type, determining whether sequences should be returned (critical for stacked recurrent layers) and setting the appropriate activation functions.

```
if layer_type == 'LSTM':
    x = LSTM(units, return_sequences=return_seq)(x)
elif layer_type == 'GRU':
    x = GRU(units, return_sequences=return_seq)(x)
elif layer_type == 'RNN':
    x = SimpleRNN(units, return_sequences=return_seq)(x)
elif layer_type == 'DENSE':
    x = Dense(units, activation=activation)(x)
```

Model Compilation:

```
model.compile(optimizer=optimizer, loss=loss_function)
```

Compiling the model involves selecting suitable loss functions (e.g., 'Huber' for robustness to outliers) and optimizers ('Adam' for adaptive learning rates), both essential to the model's learning behaviour.

Summaries of Experiments with Different Configurations:

Combined LSTM, GRU, and Dense Architecture

- **Setup**: Added an LSTM layer with 150 units, followed by a GRU layer with 100 units, and a Dense layer with 50 units. Increased dropout rates to 0.3 and 0.2, using 'Huber' as the loss function.
- **Results**: Noticeable improvement in validation loss, better training stability, and reduced overfitting.
- **Insight**: The combined architecture of LSTM and GRU layers captures sequential dependencies more effectively, with dropout providing essential regularization.

Tuning Activation Functions

- **Setup**: Compared tanh activation for recurrent layers versus Relu for Dense layers.

- **Results**: Models using tanh demonstrated smoother learning curves, particularly beneficial for LSTM and GRU layers, while Relu was effective for Dense layers in final predictions.
- **Insight**: Matching activation functions to the specific characteristics of layer types enhances performance and convergence.

Testing Loss Functions and Optimizers

- **Setup**: Compared 'Adam' and 'sgd' optimizers, tested 'mse' against 'Huber' loss.
- **Results**: The 'Adam' optimizer consistently led to faster convergence and better performance metrics compared to 'sgd', and 'Huber' loss proved robust against price outliers.
- **Insight**: Using 'Adam' with 'Huber' loss is advantageous for financial data due to its volatility and noise.