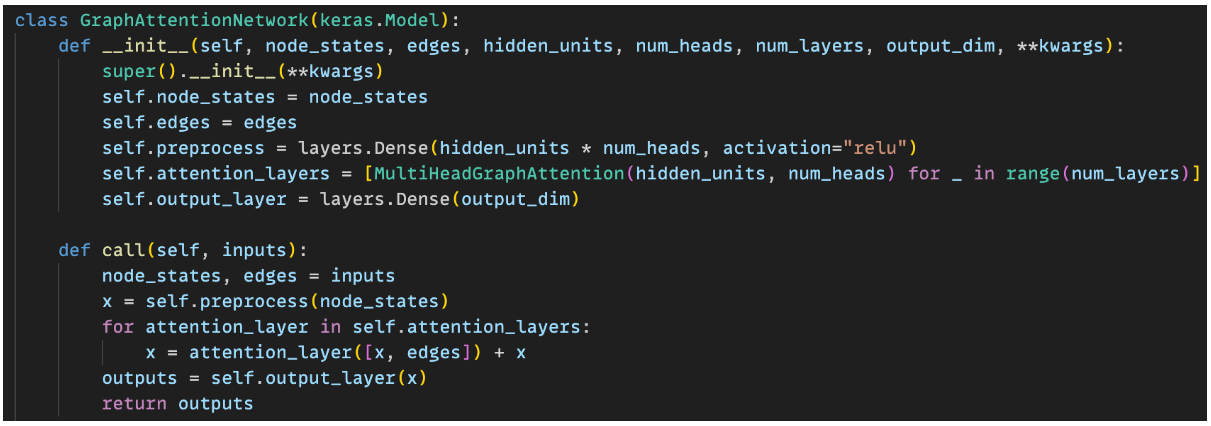
Report on Graph Attention Network for Pedestrian Trajectory Prediction

<u>**Task 1**</u>: Adjust tutorial implementation and evaluate on dedicated test set

The dataset was subjected to a 70-20-10 training-validation-testing split and normalized. Normalizing the data ensures numerical stability, more interpretable results and also better generalization across the entire dataset since it is less likely to overfit. -1 values and missing values were dropped.

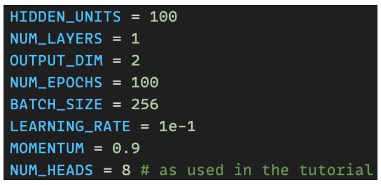
I had to combine all edges and nodes into a single dataset like the tutorial instead of treating each scene separately because through experimentation, I found that the mean Euclidean distance across the scenes can be as high as 25000, which does not make sense considering the prediction is only for a person's position after one second. A possible explanation would be that there is insufficient training data in each individual scene and the model was unable to capture the relationships and patterns of pedestrian trajectories. Hence, combing the data from multiple scenes might help give the model a more robust understanding of the trajectories.

The model architecture and hyperparameters are very similar to the tutorial.



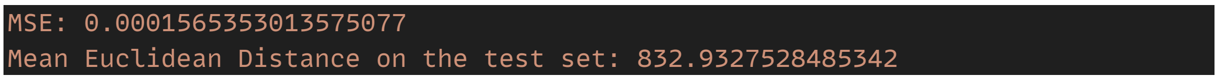
```
class GraphAttentionNetwork(keras.Model):
    def __init__(self, node_states, edges, hidden_units, num_heads, num_layers, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.node_states = node_states
        self.edges = edges
        self.preprocess = layers.Dense(hidden_units * num_heads, activation="relu")
        self.attention_layers = [MultiHeadGraphAttention(hidden_units, num_heads) for _ in range(num_layers)]
        self.output_layer = layers.Dense(output_dim)

    def call(self, inputs):
        node_states, edges = inputs
        x = self.preprocess(node_states)
        for attention_layer in self.attention_layers:
            x = attention_layer([x, edges]) + x
        outputs = self.output_layer(x)
        return outputs
```



```
HIDDEN_UNITS = 100
NUM_LAYERS = 1
OUTPUT_DIM = 2
NUM_EPOCHS = 100
BATCH_SIZE = 256
LEARNING_RATE = 1e-1
MOMENTUM = 0.9
NUM_HEADS = 8 # as used in the tutorial
```

The performance of the model was evaluated using mean Euclidean distance between the predicted and actual positions of pedestrians in the test set.



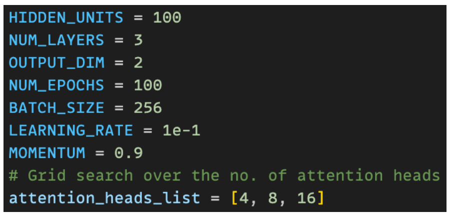
```
MSE: 0.00015653013575077
Mean Euclidean Distance on the test set: 832.9327528485342
```

With the data normalized, the MSE is small enough and the mean Euclidean distance is as shown.

<u>**Task 2**</u>: Hyperparameter tuning and Deeper Embedding of node features

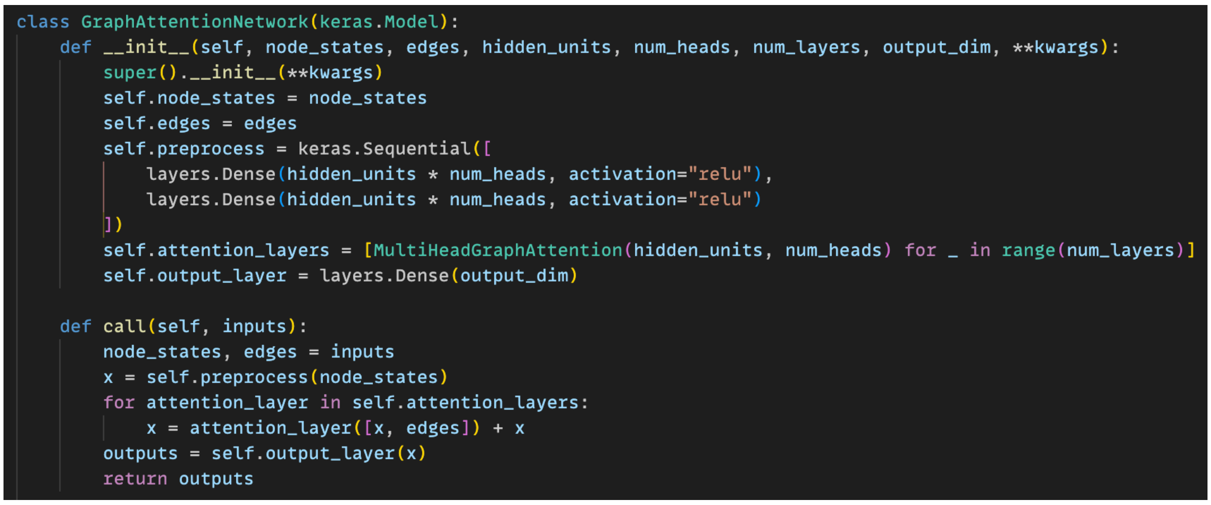
The dataset and data preparation follows Task 1.

Two additional settings for the number of attention heads were evaluated: [4, 8, 16]. The optimal number of attention heads were determined based on the validation mean squared error (MSE).



```
HIDDEN_UNITS = 100
NUM_LAYERS = 3
OUTPUT_DIM = 2
NUM_EPOCHS = 100
BATCH_SIZE = 256
LEARNING_RATE = 1e-1
MOMENTUM = 0.9
# Grid search over the no. of attention heads
attention_heads_list = [4, 8, 16]
```

The node features were embedded using a deeper network where the number of layers (num_layers) was increased 3 from 1. Also, the node features were passed through two fully connected layers with added ReLU activations at the final layer.

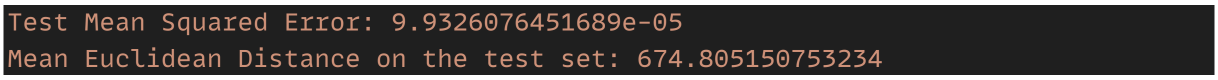


```
class GraphAttentionNetwork(keras.Model):
    def __init__(self, node_states, edges, hidden_units, num_heads, num_layers, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.node_states = node_states
        self.edges = edges
        self.preprocess = keras.Sequential([
            layers.Dense(hidden_units * num_heads, activation="relu"),
            layers.Dense(hidden_units * num_heads, activation="relu")
        ])
        self.attention_layers = [MultiHeadGraphAttention(hidden_units, num_heads) for _ in range(num_layers)]
        self.output_layer = layers.Dense(output_dim)

    def call(self, inputs):
        node_states, edges = inputs
        x = self.preprocess(node_states)
        for attention_layer in self.attention_layers:
            x = attention_layer([x, edges]) + x
        outputs = self.output_layer(x)
        return outputs
```

Gradient clipping is used to avoid the problem of vanishing gradients, which did occur when training with 16 attention heads.

Hyperparameter tuning found 4 attention heads were the most optimal with the lowest MSE - 0.0001375. Testing this model with 4 attention heads yields the result:

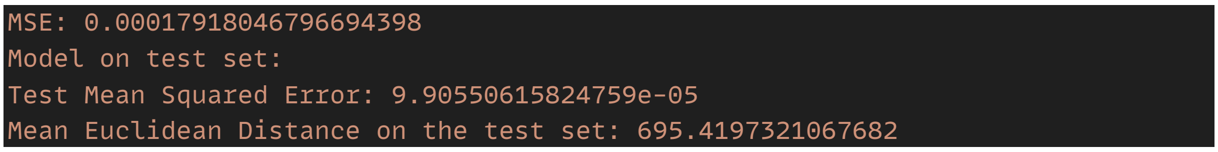


```
Test Mean Squared Error: 9.93260764451689e-05
Mean Euclidean Distance on the test set: 674.805150753234
```

The mean Euclidean distance is 19% lower than in Task 1, indicating better model performance. Predicted pedestrian positions are closer to actual positions, reflecting higher accuracies and lower variance in predictions.

Task 3: Cosine similarity based attention network

The dataset and data preparation follows Task 1. The hyperparameters I used is the same as Task 2, but the number of attention heads used is 4; as found by the hyperparameter tuning. Attention mechanism is now uses cosine similarity.



```
MSE: 0.00017918046796694398
Model on test set:
Test Mean Squared Error: 9.905506158247592e-05
Mean Euclidean Distance on the test set: 695.4197321067682
```

Both mechanisms in Task 2 and 3 were very similar.