Evelina Ramoskaite

Kaggle Username: Evelina Ramoskaite

Data Preparation and Exploratory Data Analysis

To prepare the data, I first had to convert the “date_time” variable into a proper date format. After that, I made sure there were no missing values in the dataset. I decided to remove the categorical variables because it made running the models much simpler. Once the unnecessary columns were removed, I split the data into a training, validation, and test set. These were then normalized like the TensorFlow tutorial sets.

I also made univariate plots of the most interesting features, like holidays, as well as some time plots. These showed a clear repeating pattern in traffic volumes. I also looked at the general distribution and 5-number summaries for each numerical variable.

Models and Development

I chose to experiment with different model architectures first, and then tuned the hyperparameters of the model that performed the best. Each model was trained for 20 epochs using the Adam optimizer with a default learning rate of 0.001. My best model was a CNN.

Baseline Model – Dense (7-1-3)

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_23 (Dense)            (None, 7, 7)              42

 dense_24 (Dense)            (None, 7, 1)              8

 dense_25 (Dense)            (None, 7, 3)              6

=================================================================
Total params: 56
Trainable params: 56
Non-trainable params: 0
_________________________________________________________________
```

For my first model, I had a dense neural network with three layers. I kept the structure minimal, and only used a few nodes per layer. This was more complicated than the baseline given in the TensorFlow tutorial, but minimal relative to my other models. After 20 epochs, the MAE for the validation set was 0.8004.

Long Short Term Memory RNN Model

Another type of neural network I experimented with was an LSTM model. I used 1 layer initially, and then attempted a more complicated model. More layers performed better, which I expected. Here is my best LSTM model:

```
] lstm_model = tf.keras.models.Sequential([
      # Shape [batch, time, features] => [batch, time, lstm_units]
      tf.keras.layers.LSTM(160, return_sequences=True),
      tf.keras.layers.LSTM(62, return_sequences=True),
      tf.keras.layers.LSTM(32, return_sequences=True),
      # Shape => [batch, time, features]
      tf.keras.layers.Dense(units=1)
  ])
```

It had a MAE of 0.4491 on the validation set with three layers. Since it underperformed relative to the convolutional neural network, I chose to focus more on tuning the CNN for the Kaggle competition.

Convolutional Neural Network Model

The convolutional model had great performance without any tuning relative to the LSTM and baseline model . I experimented with different numbers of layers, numbers of nodes per layer, optimizers, and activation functions.

Overview of Tuning Process

Structure # layers : (filters,layer1 units, layer 2 units, etc..)	Activation	MAE (Validation)
4: (32,32,16,1)	Relu	0.2388
4: (32,32,16,1)	LeakyReLu	0.2284
4: (32,32,16,1)	Tanh	0.2371
4: (32,32,16,1)	Softmax	0.2900
4: (126,63,32,1)	Relu	0.2303
5: (126,63,32,32,1)	Relu	0.2359
4: (126,63,32,1)	Tanh	0.2318
5: (126,64,16,8,1)	LeakyReLu	0.2269
5: (126,64,16,8,1)	Tanh	0.2277
5: (64,64,16,8,1)	LeakyReLu	0.2329
4: (126,63,32,1)	Tanh, relu, LeakyRelu	0.2212

For my first attempt, I had a total of 4 layers and used the rectified linear unit activation function.

```
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=32,
                           kernel_size=(CONV_WIDTH,),
                           activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=16, activation='relu'),
    tf.keras.layers.Dense(units=1),
])
```

The MAE of this model on the validation set was 0.2386.

After building this model, I tuned the activation functions in each layer.

One version had the same structure, with LeakyReLu(MAE=0.2282). Another had Tanh(MAE=0.2371) as the activation function. I used softmax(MAE=0.249) as well, but it had the worst results.

I found through trial and error that using a combination of the LeakyReLu and Tanh activation function resulted in better performance than having the same activation function in every layer.

After determining what the best activation functions were, I tweaked the number of nodes in each layer, as well as the amount of filters again. I found that having more nodes in the initial layer resulted in the best performance.

```
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=126,
                           kernel_size=(CONV_WIDTH,),
                           activation='relu'),
    tf.keras.layers.Dense(units=63, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
])
```

The MAE of this model on the validation set was 0.2303.

I then compared the performance between a network with 4 layers and a network with 5 layers. I believed that a model with more layers would perform better, but the additional complexity did not yield any meaningful improvements. This model was not the best on the test set, and likely had some overfitting.

```
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=126,
                           kernel_size=(CONV_WIDTH,),
                           activation='tanh'),
    tf.keras.layers.Dense(units=64, activation='tanh'),
    tf.keras.layers.Dense(units=16, activation='LeakyReLU'),
    tf.keras.layers.Dense(units=8, activation='LeakyReLU'),
    tf.keras.layers.Dense(units=1),
])
```

Kaggle Best Submission

I attempted improving the CNN model several more times by modifying the activation functions and number of layers. The best model had 4 layers, with 126 filters in the first layer.

```
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=126,
                           kernel_size=(CONV_WIDTH,),
                           activation='tanh'),
    tf.keras.layers.Dense(units=63, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='LeakyReLU'),
    tf.keras.layers.Dense(units=1),
])
```

MAE (validation) :0.2212

Kaggle MAE score: 406.87

There are different activation functions in each layer. I think that this allowed for different types of non-linearities to be captured at each layer and compensated for any shortcomings an individual activation function may have.