DeepLearning.AI Tensorflow Developer

Sequences, Time Series and Prediction: Week 2

Time Series DNN Model

Time series prediction with DNN model requires feature and label where feature is the sequence of values in fixed window size and label is the next value of the window in the given series. Perform window slicing, shifting, shuffling, batching and prefetching using tf.data.Dataset API.

```
dataset = dataset.window(window_size+1, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(shuffle_buffer_size)
dataset = dataset.batch(batch_size)
dataset = dataset.cache().prefetch(1)
```

Train NN model on features with single Dense unit to perform simple linear regression with MSE loss function and Stochastic Gradient Descent optimization. Single NN model forecasting reduces MAE error value.

```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size,)),
  tf.keras.layers.Dense(1)])
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100)


forecast = []
for time in range(len(series) - window_size):
 forecast.append(model.predict(series[time:time + window_size][np.newaxis], verbose=0))
forecast = forecast[split_time - window_size:]
results = np.array(forecast).squeeze()
print(tf.keras.metrics.mae(x_valid, results).numpy())
```

DNN model for time series forecasting includes three Dense layers. In order to optimize the learning, we implement learning rate scheduler using callback which is called at the end of each epoch where learning rate increases with epoch number. Finally, plot the loss values in terms of learning rate and find the learning rate that minimizes the loss. Also, retrain with new learning rate to observe the loss and prediction for larger number of epochs which further minimizes the MAE error value.

```
model_tune = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size,)),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1)])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
history = model_tune.fit(dataset, epochs=100, callbacks=[lr_schedule])
```