Sequence, Time Series and Prediction

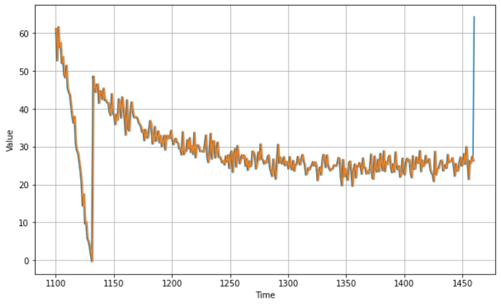
Week 1 - Sequences and Prediction

This week they talked about different approaches to model and predict for data sequences. Also, how to split the data in training, validation and test (should be sequential since the order matters) and also the properties of stationary/non-stationary series.

Some of the approaches are:

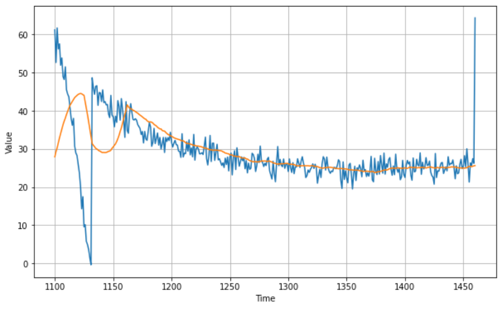
Naive Forecast

```
naive_forecast = series[split_time - 1:-1]
```



Moving Average Forecast

```
def moving_average_forecast(series, window_size):
 """Forecasts the mean of the last few values.
    If window_size=1, then this is equivalent to naive forecast"""
 forecast = []
 for time in range(len(series) - window_size):
   forecast.append(series[time:time + window_size].mean())
 return np.array(forecast)

moving_avg = moving_average_forecast(series, 30)[split_time - 30:]
```



Week 2 - Deep Neural Networks for Time Series

This week they talked about using deep neural networks for time series, preparing the data and training the model.

Below there is a snippet of the neural network code and fitting in the training dataset.

```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
   tf.keras.layers.Dense(100, input_shape=[window_size], activation="relu"),
   tf.keras.layers.Dense(10, activation="relu"),
   tf.keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=1e-6, momentum=0.9))
model.fit(dataset, epochs=100, verbose=0)
```

Week 3 - Recurrent Neural Networks for time series

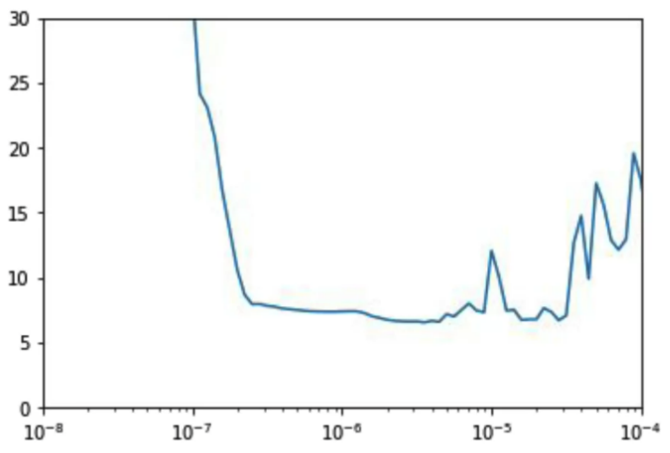
This week they talked about training recurrent neural networks, LSTMs and applying those types of models into the forecasting prediction.

Below there is a snippet of an LSTM neural network and a way of adjusting the learning rate dynamically.

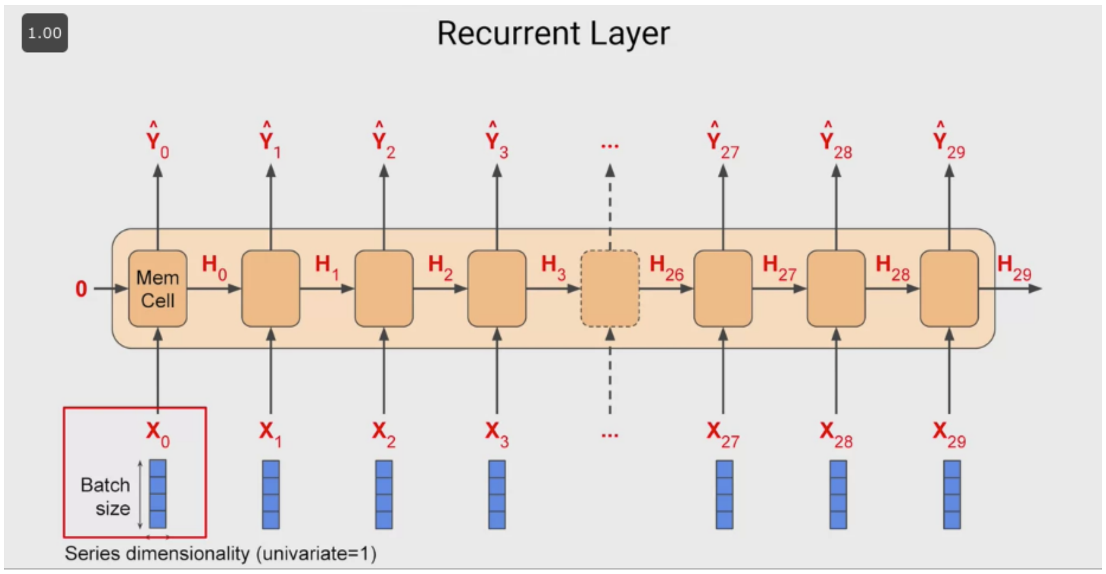
```
dataset = windowed_dataset(x_train, window_size, batch_size,
shuffle_buffer_size)

model = tf.keras.models.Sequential([
 tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                     input_shape=[None]),
                          tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,
return_sequences=True)),
 tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
 tf.keras.layers.Dense(1),
 tf.keras.layers.Lambda(lambda x: x * 10.0)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(
   lambda epoch: 1e-8 * 10**(epoch / 20))

optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)

model.compile(loss=tf.keras.losses.Huber(),
             optimizer=optimizer,
             metrics=["mae"])

history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

From that you can extract the loss and the learning rate and plot those in a graph (shown below) to identify the best learning rate for training the model.

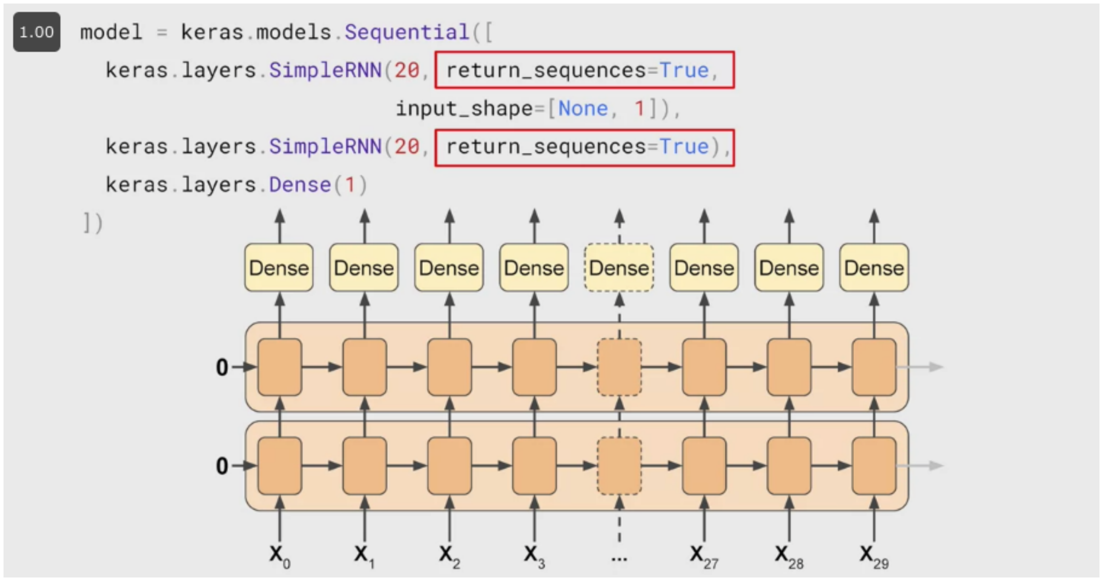
```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 30])
```



They also talked about the shape of an RNN model and provided the following illustration.

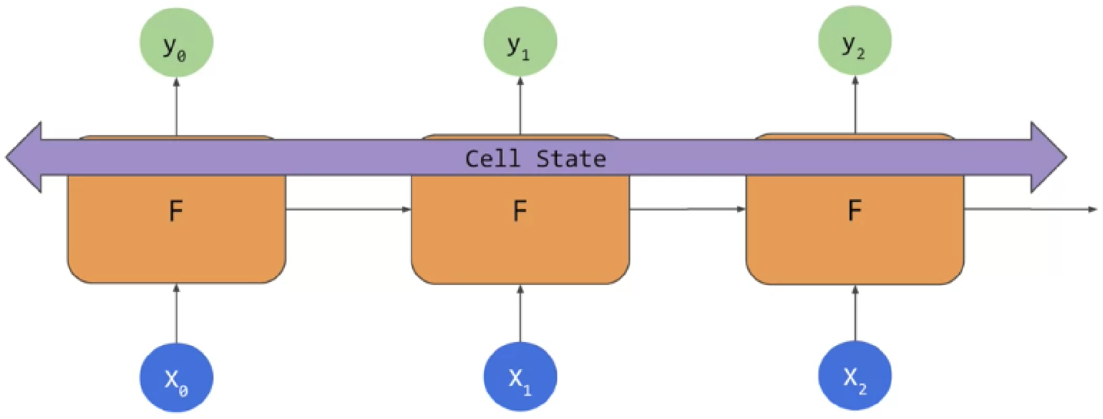

1.00
Recurrent Layer
$\hat{Y}_0$ $\hat{Y}_1$ $\hat{Y}_2$ $\hat{Y}_3$ ... $\hat{Y}_{27}$ $\hat{Y}_{28}$ $\hat{Y}_{29}$
0
Mem Cell
H_0 H_1 H_2 H_3 H_{26} H_{27} H_{28} H_{29}
X_0 X_1 X_2 X_3 ... X_{27} X_{28} X_{29}
Batch size
Series dimensionality (univariate=1)

Where Y hat and H are outputs from each memory cell, and also, if what it matters is mainly the final output (last Y hat), the former Y hats can be disconsidered for the prediction (sequence-to-vector parameter). On the other hand you might want to take into consideration all Y hats outputted by the model, in that case you would instead use a sequence-to-sequence parameter.



```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.Dense(1)
])
```

Now, when it comes to LSTMs, the main difference between those and RNNs is the fact that the impact of each cell may diminish over time, whereas in LSTMs the data from earlier in the window can have a greater impact on the overall projection than in the case of RNNs. This cell state in the LSTMs can be bidirectional and can move the information longer (illustration below).



Week 4 - Real-world time series data

This week they talked about using a mix of LSTM / RNN, DNN and Convolutional NN for tackling the forecasting problem using the sunspots dataset.

Also, during the videos they would be tweaking the parameters to find the best performance for the model.

Below you can find the snippet of the neural network mix mentioned above.

```
train_set = windowed_dataset(
   x_train, window_size=60, batch_size=100,
   shuffle_buffer=shuffle_buffer_size)

model = tf.keras.models.Sequential([
 tf.keras.layers.Conv1D(filters=60, kernel_size=5,
                     strides=1, padding="causal",
                     activation="relu",
                     input_shape=[None, 1]),
 tf.keras.layers.LSTM(60, return_sequences=True),
 tf.keras.layers.LSTM(60, return_sequences=True),
 tf.keras.layers.Dense(30, activation="relu"),
 tf.keras.layers.Dense(10, activation="relu"),
 tf.keras.layers.Dense(1),
 tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
             optimizer=optimizer,
             metrics=["mae"])
history = model.fit(train_set,epochs=150)
```