Copyright Notice

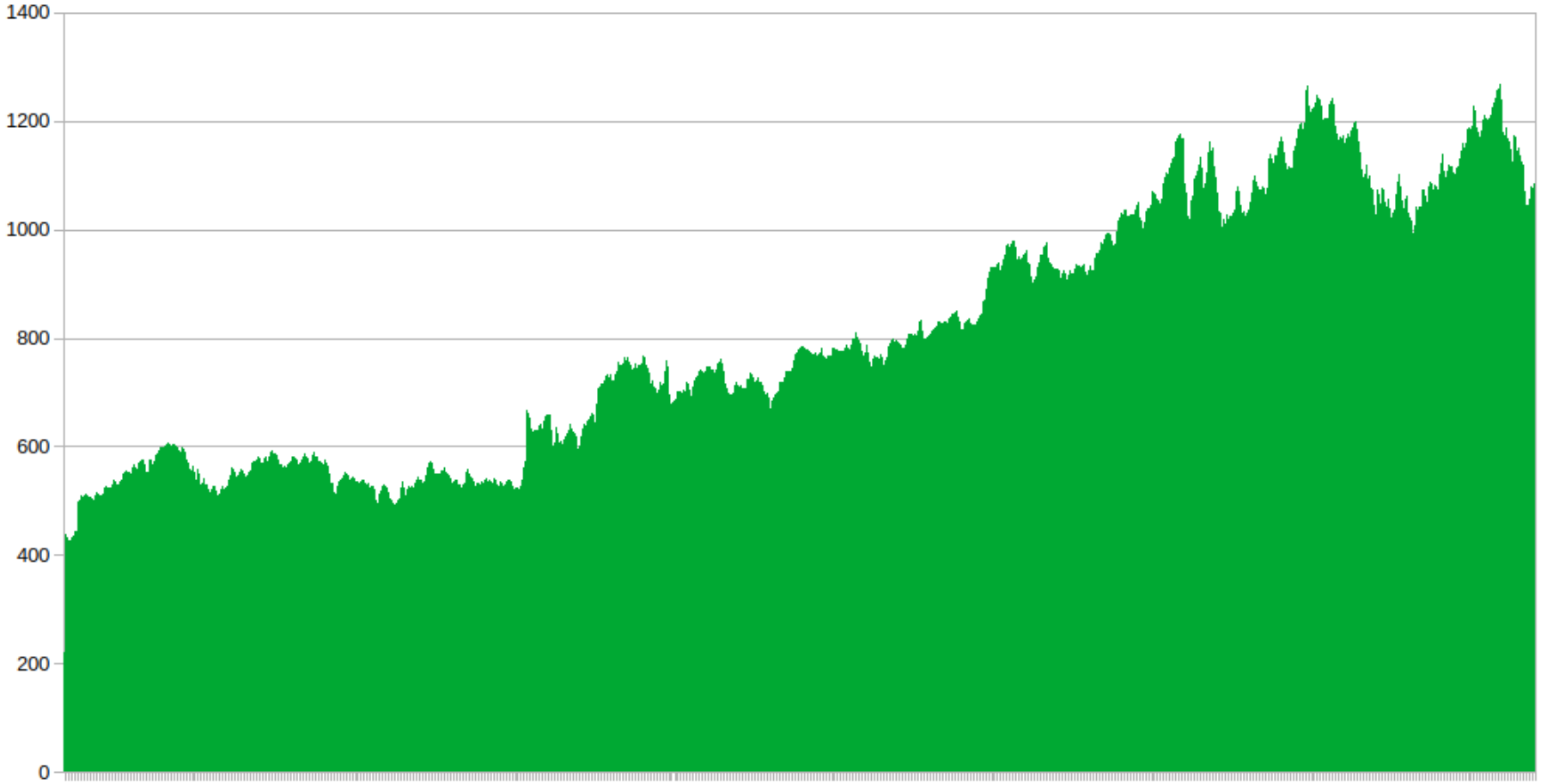


Chart Created by lmoroney@

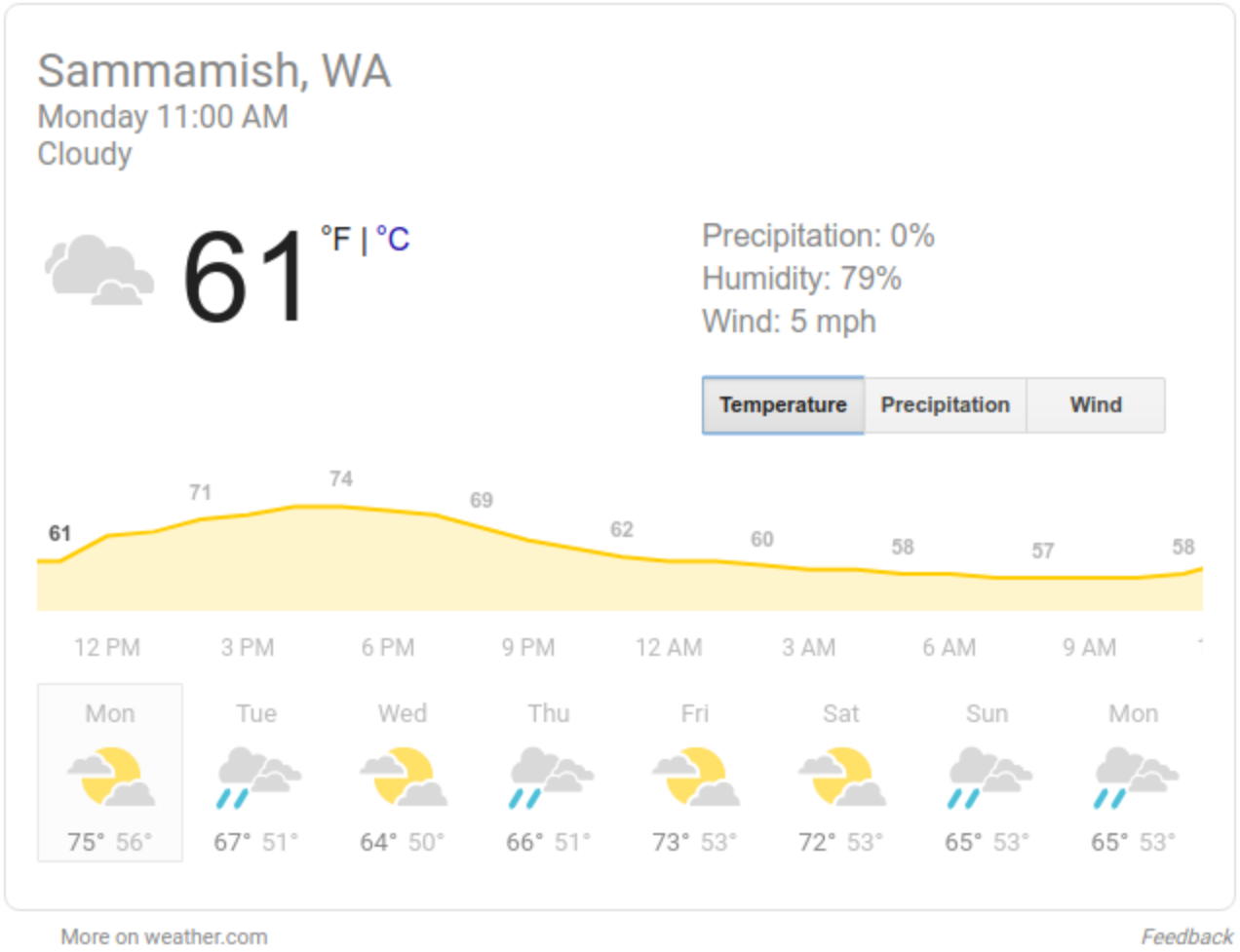


Chart From Google Search

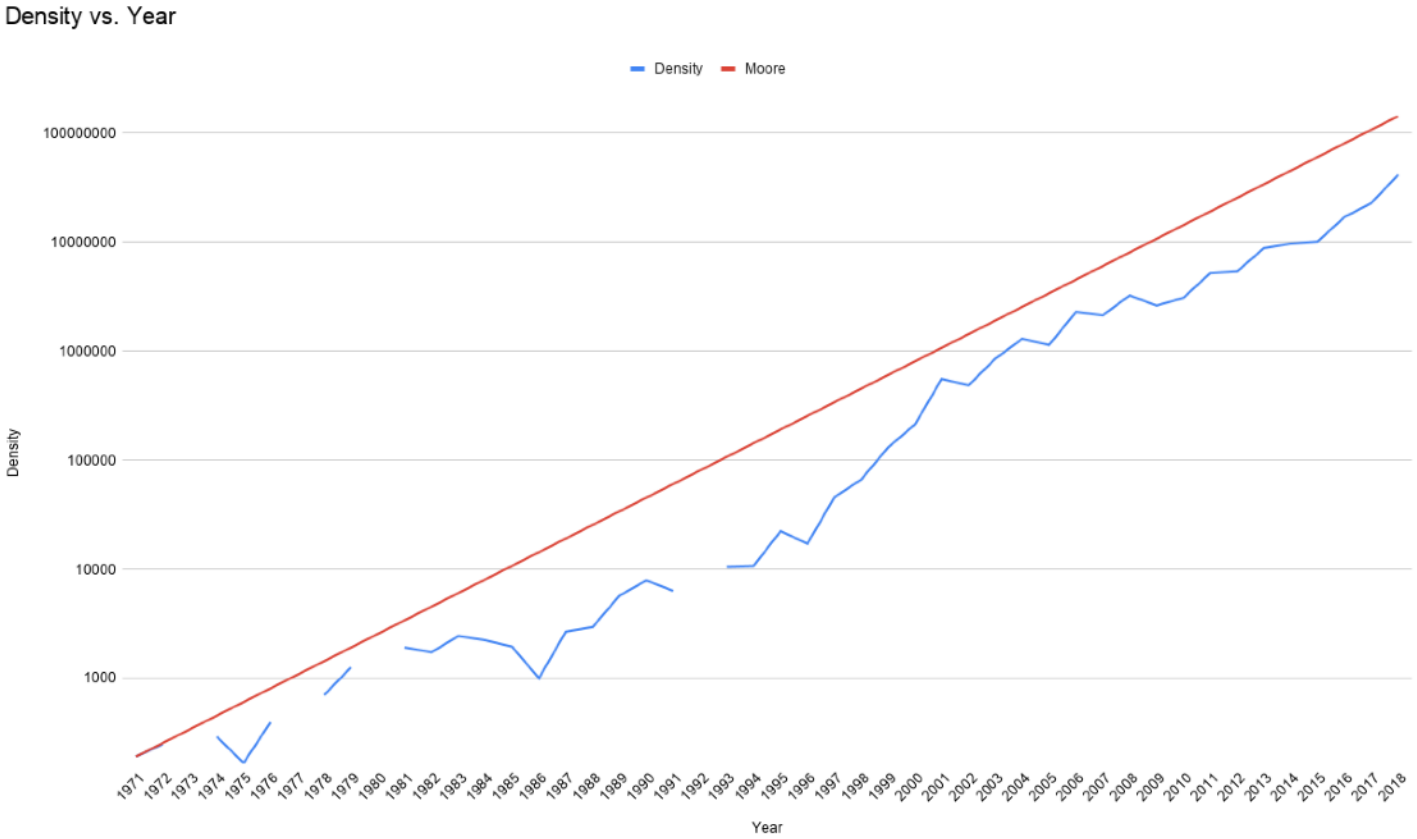
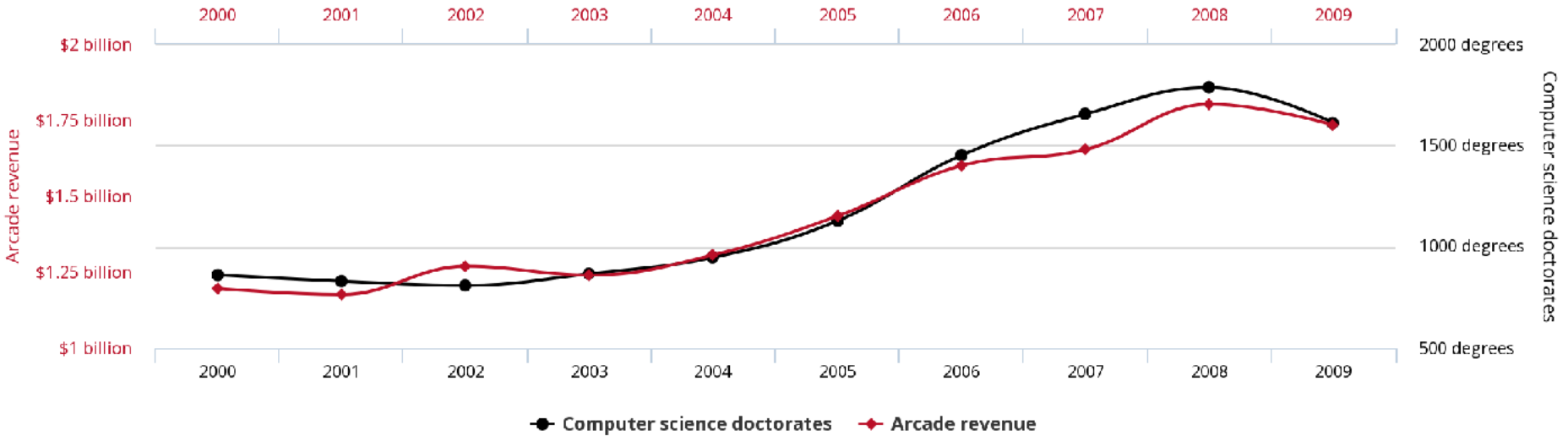


Chart Created by lmoroney@

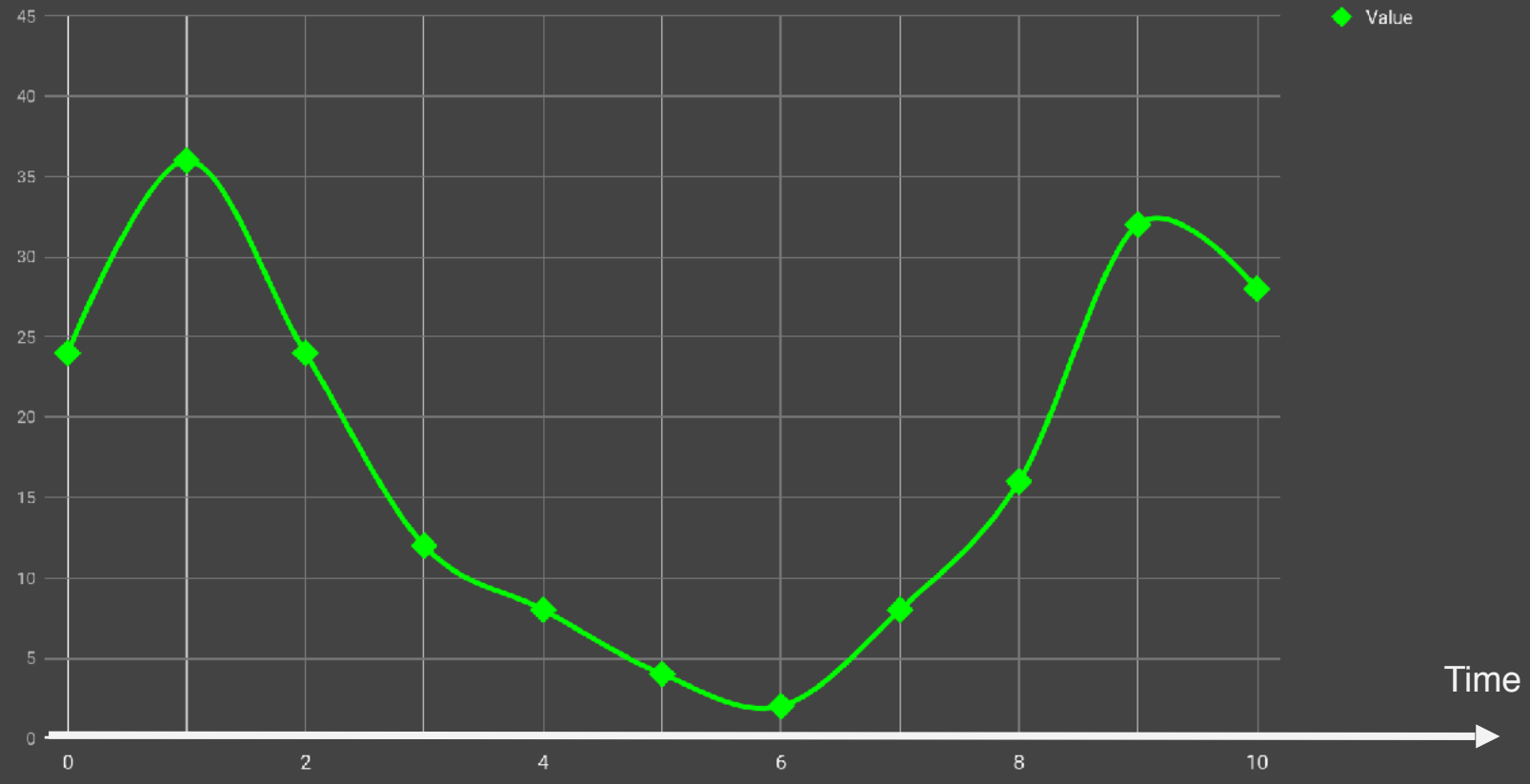
Total revenue generated by arcades

correlates with

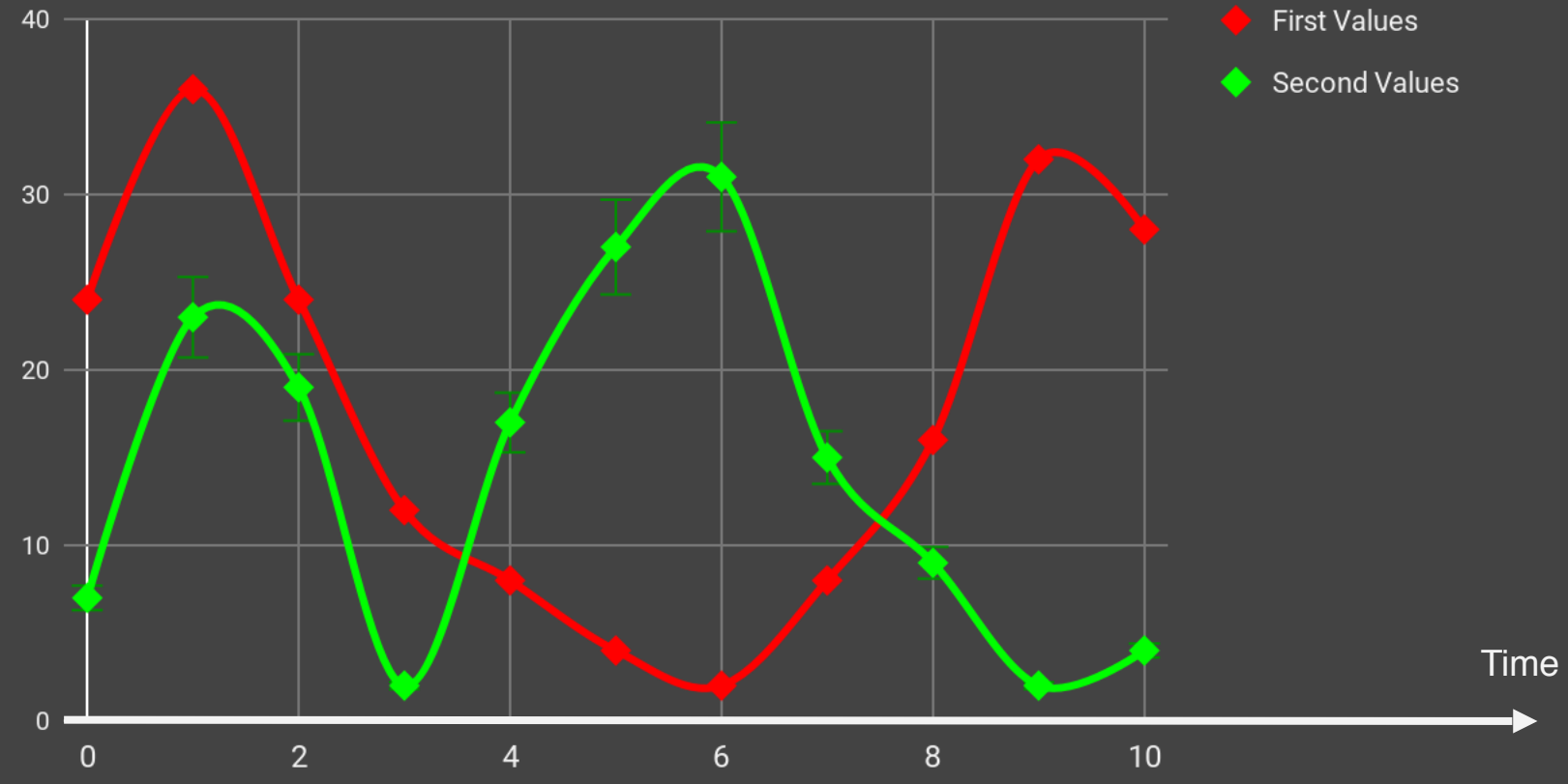
Computer science doctorates awarded in the US



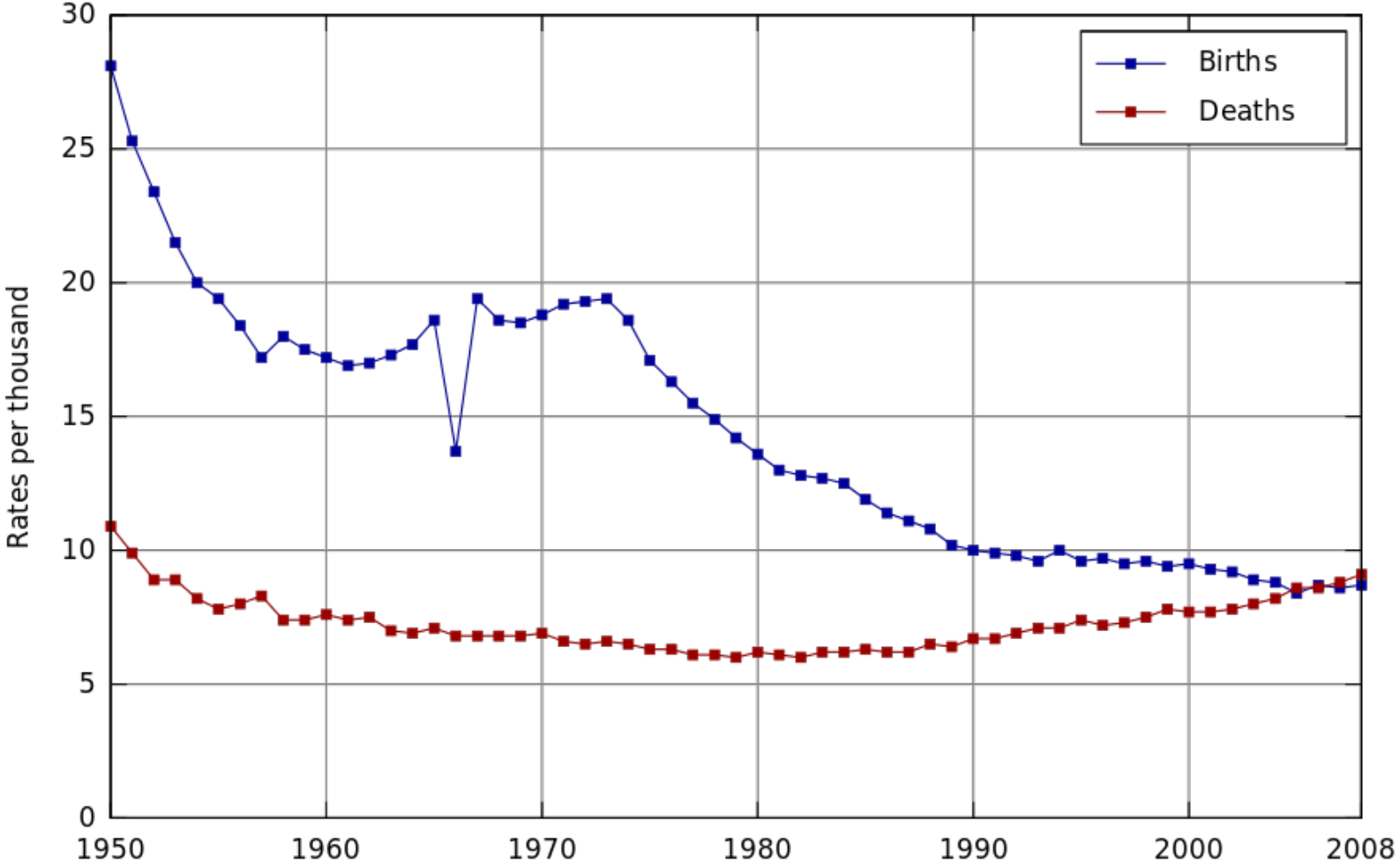
Univariate Time Series

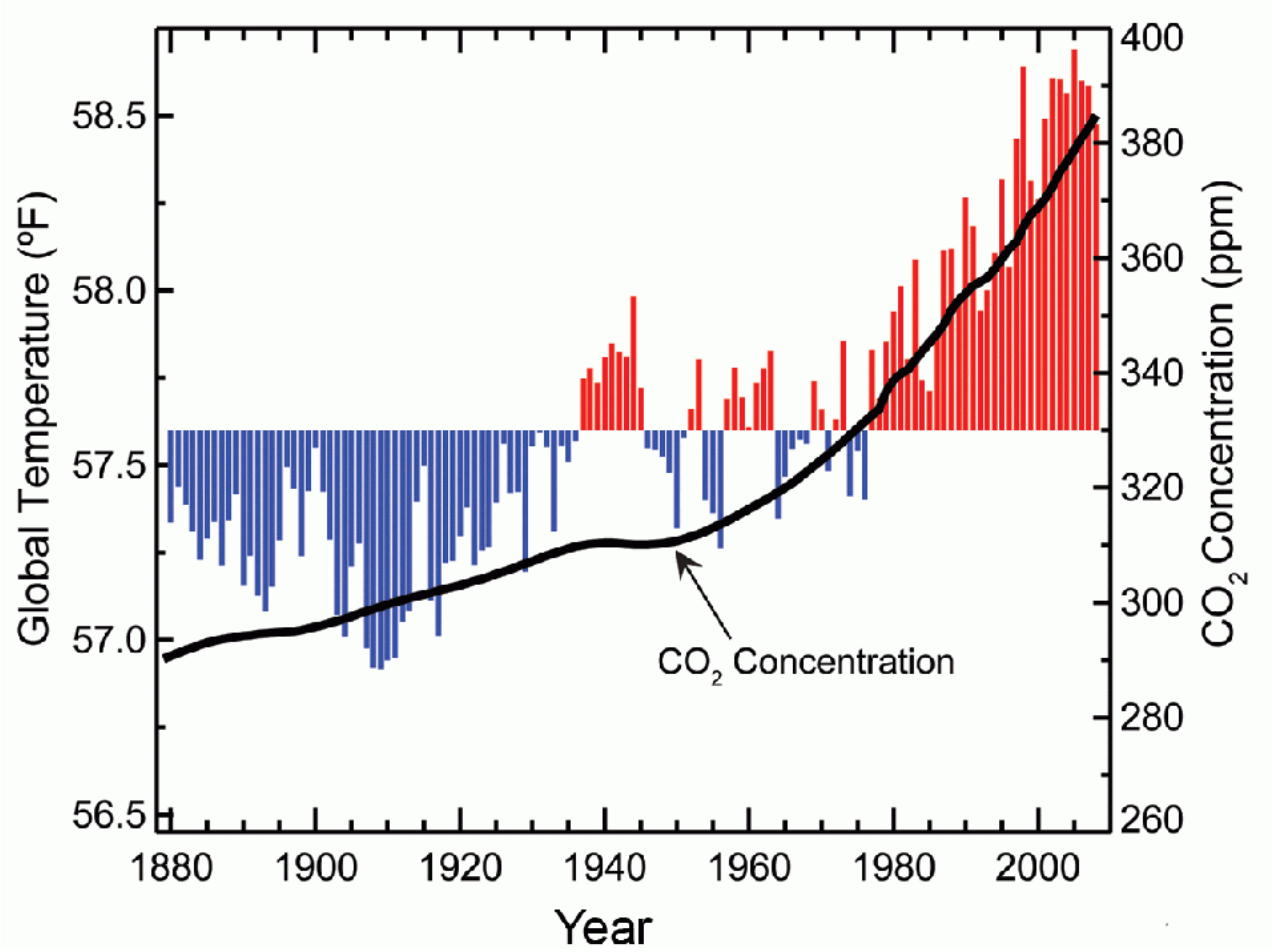


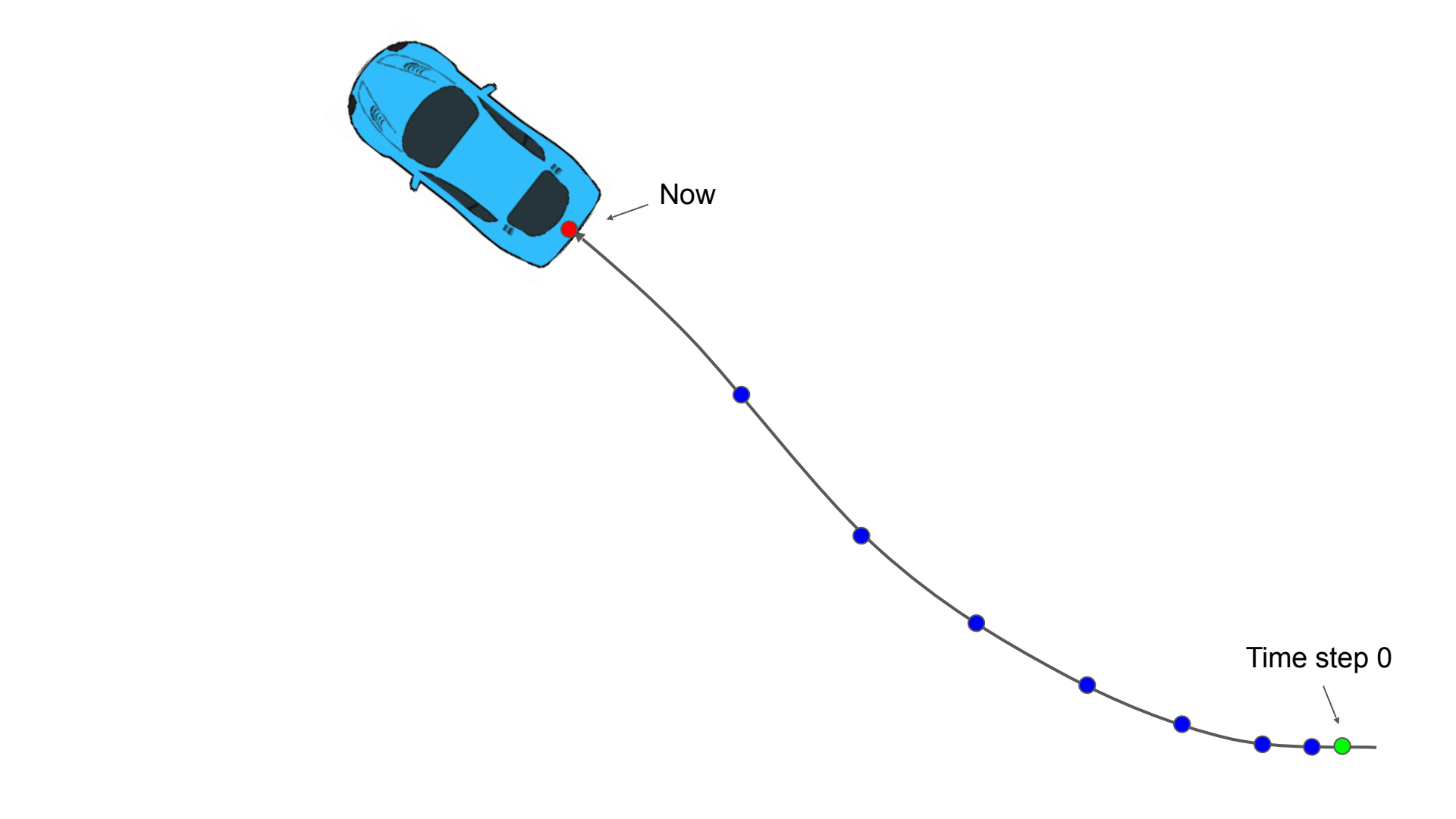
Multivariate Time Series

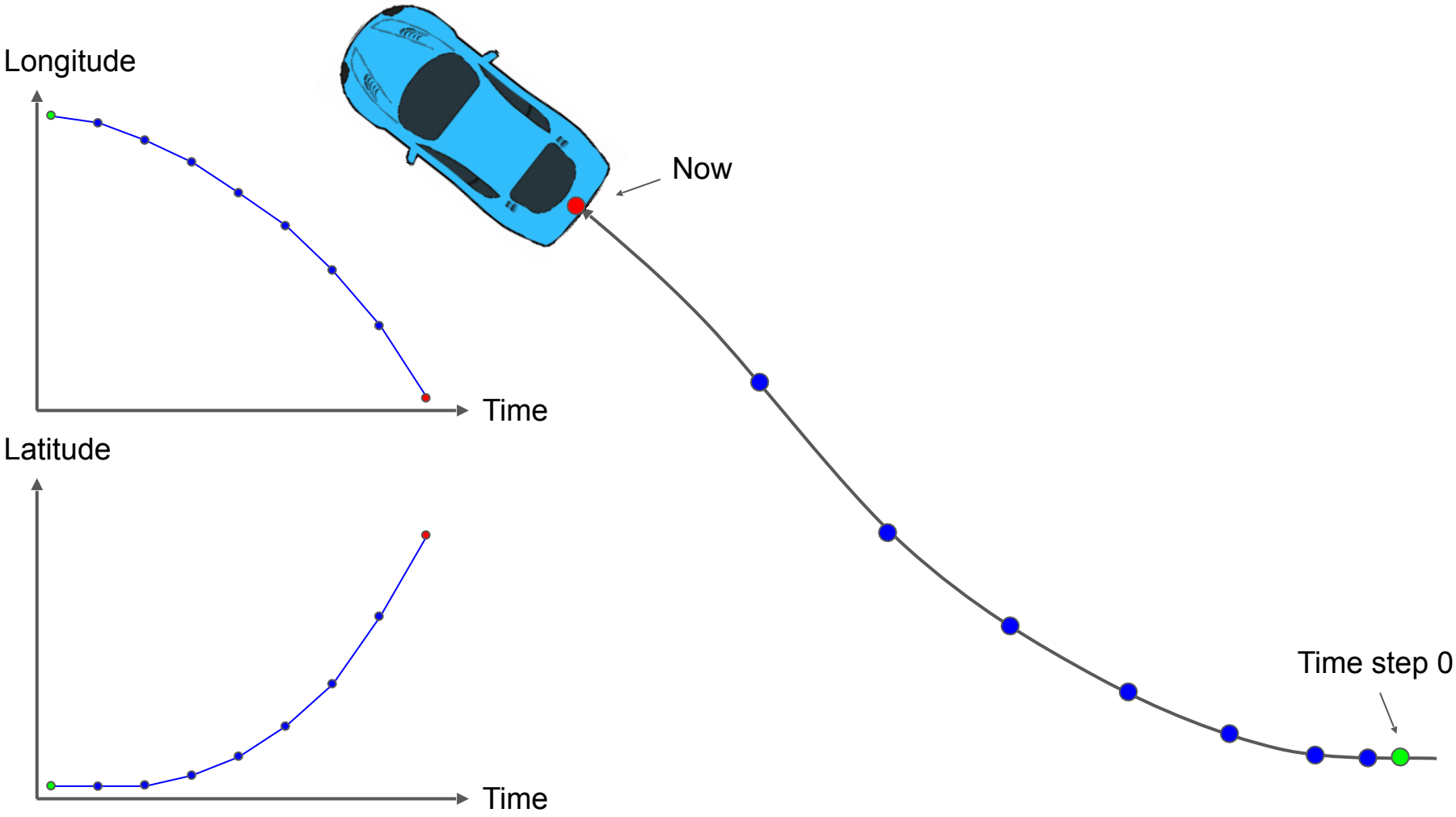


Birth and Death Rate in Japan

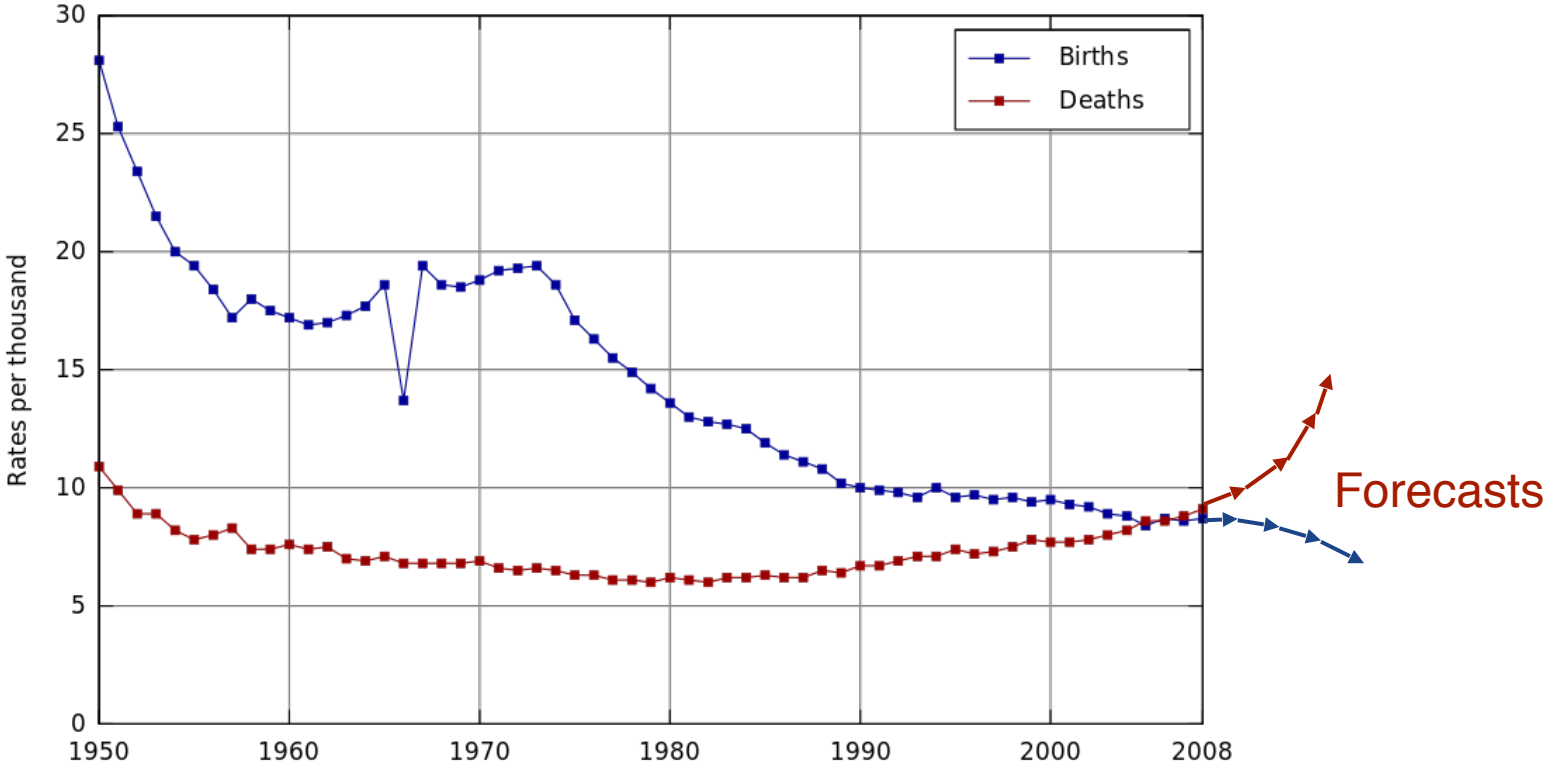



Global Temperature (ºF)
58.5
58.0
57.5
57.0
56.5
400
380
360
340
320
300
280
260
CO2 Concentration (ppm)
1880
1900
1920
1940
1960
1980
2000
Year
CO2 Concentration


Now
Time step 0


Longitude
Time
Latitude
Time
Now
Time step 0

Birth and Death Rate in Japan



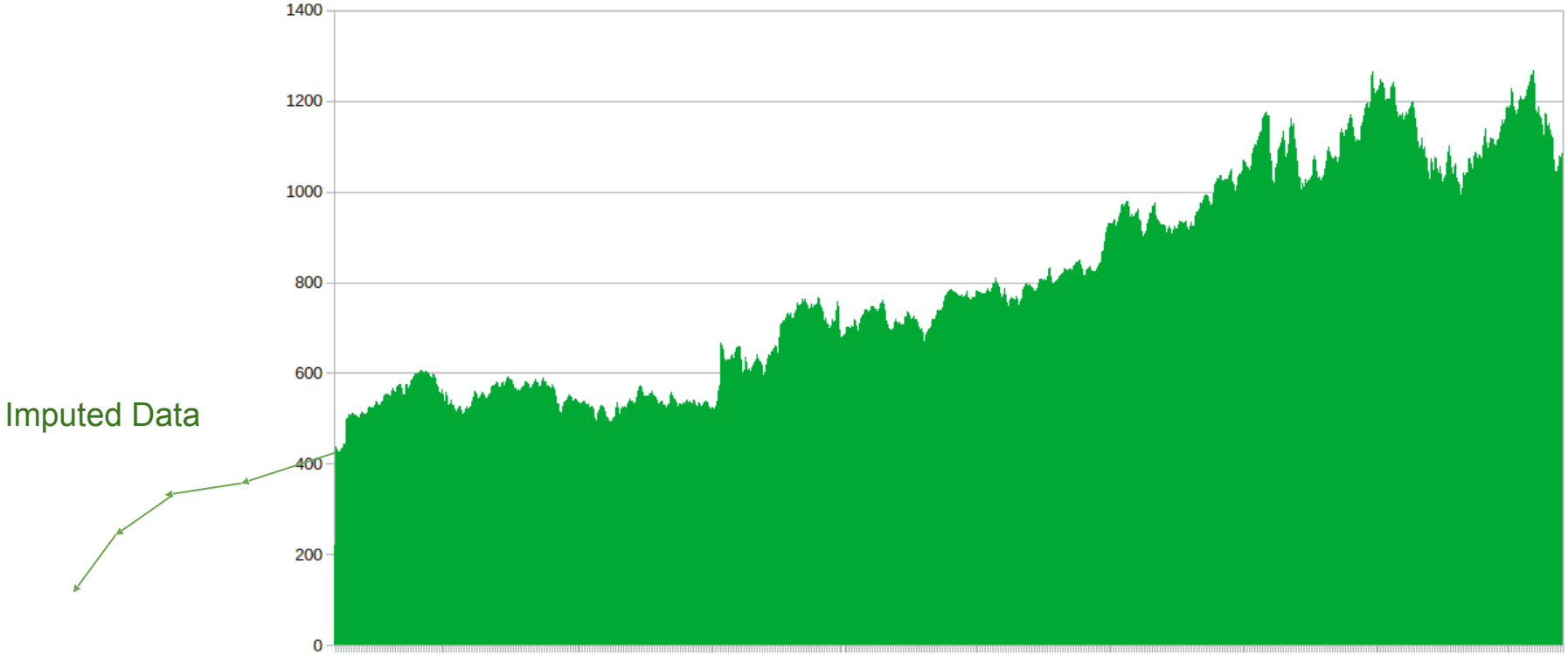


Chart Created by lmoroney@

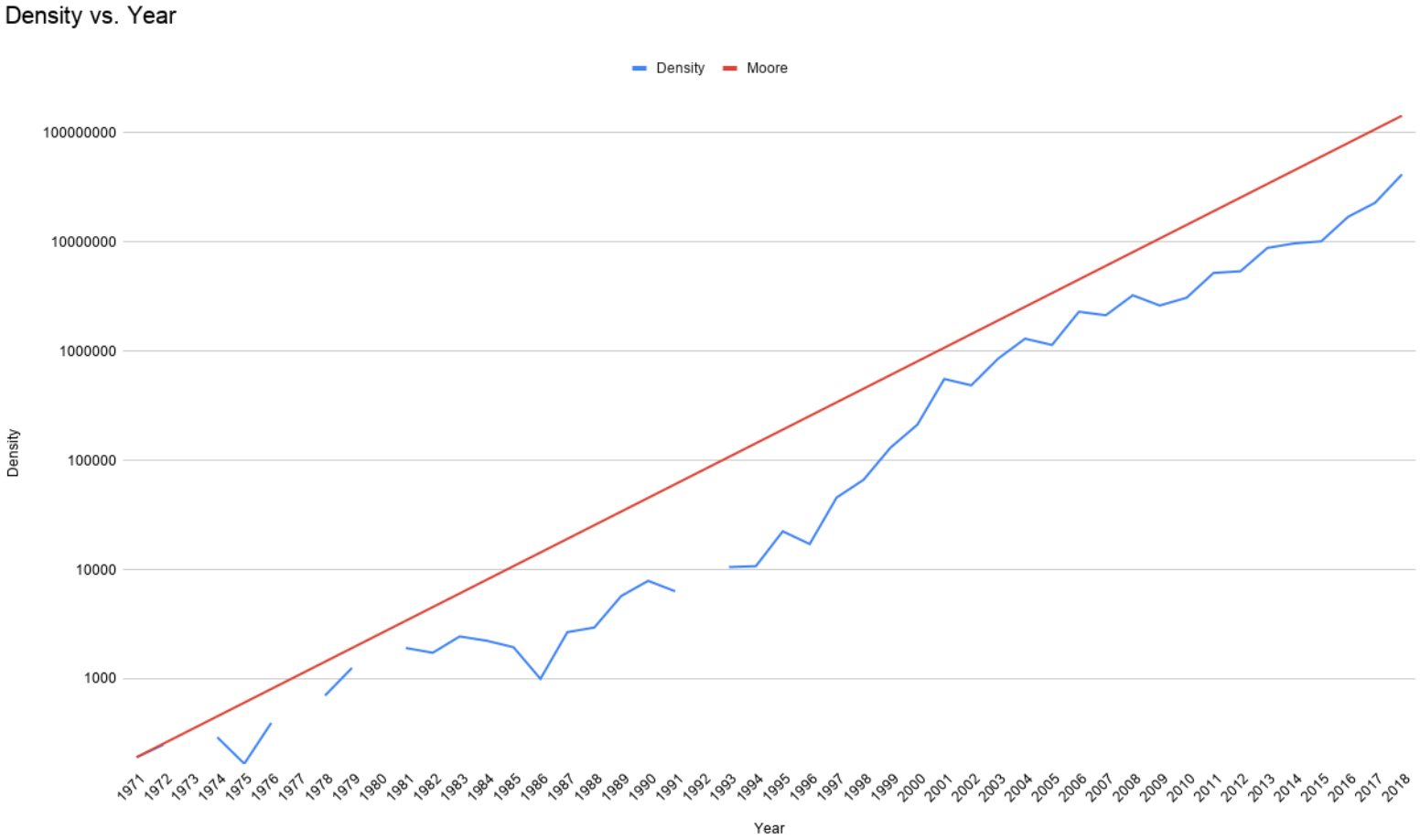


Chart Created by lmoroney@

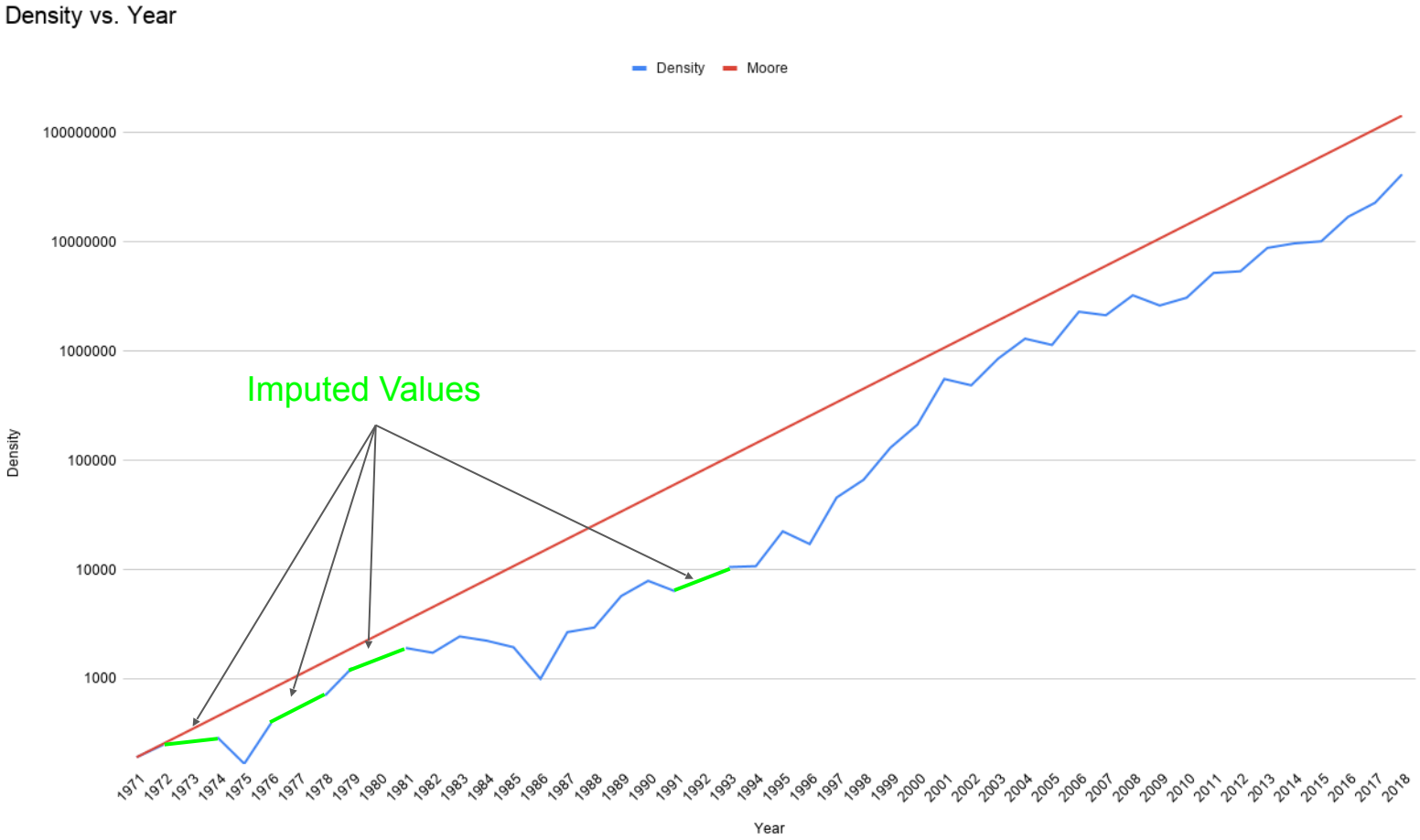
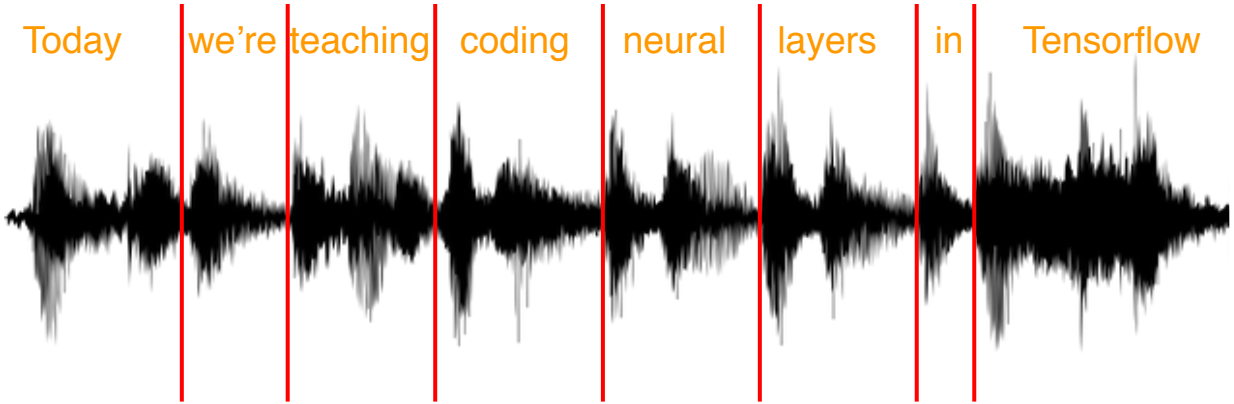


Chart Created by lmoroney@


150
140
130
120
110
100
90
0
50
100
150


Today
we're
teaching
coding
neural
layers
in
Tensorflow

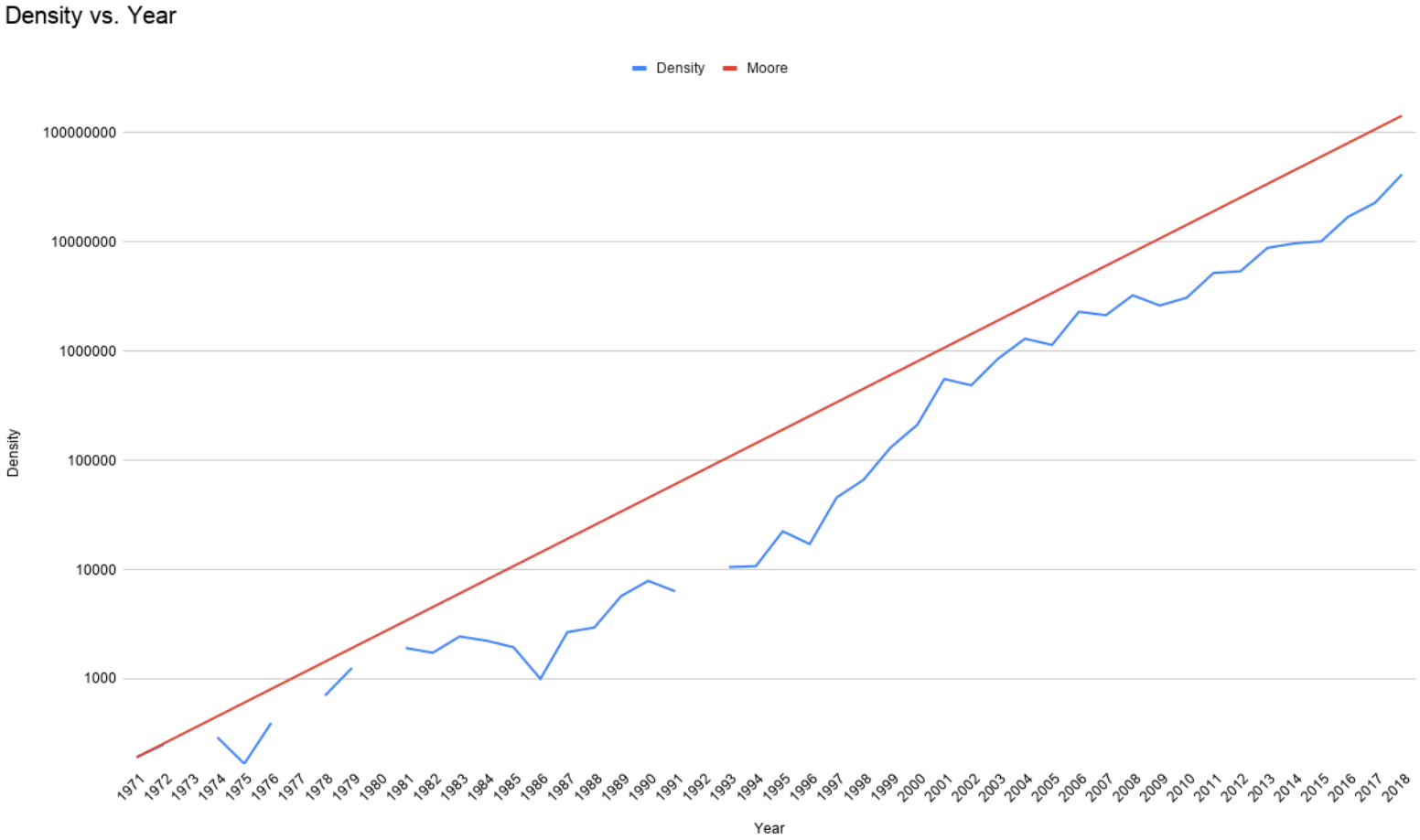
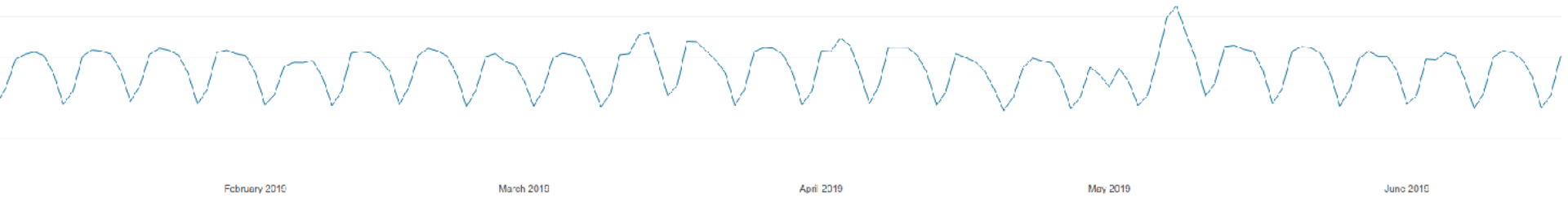
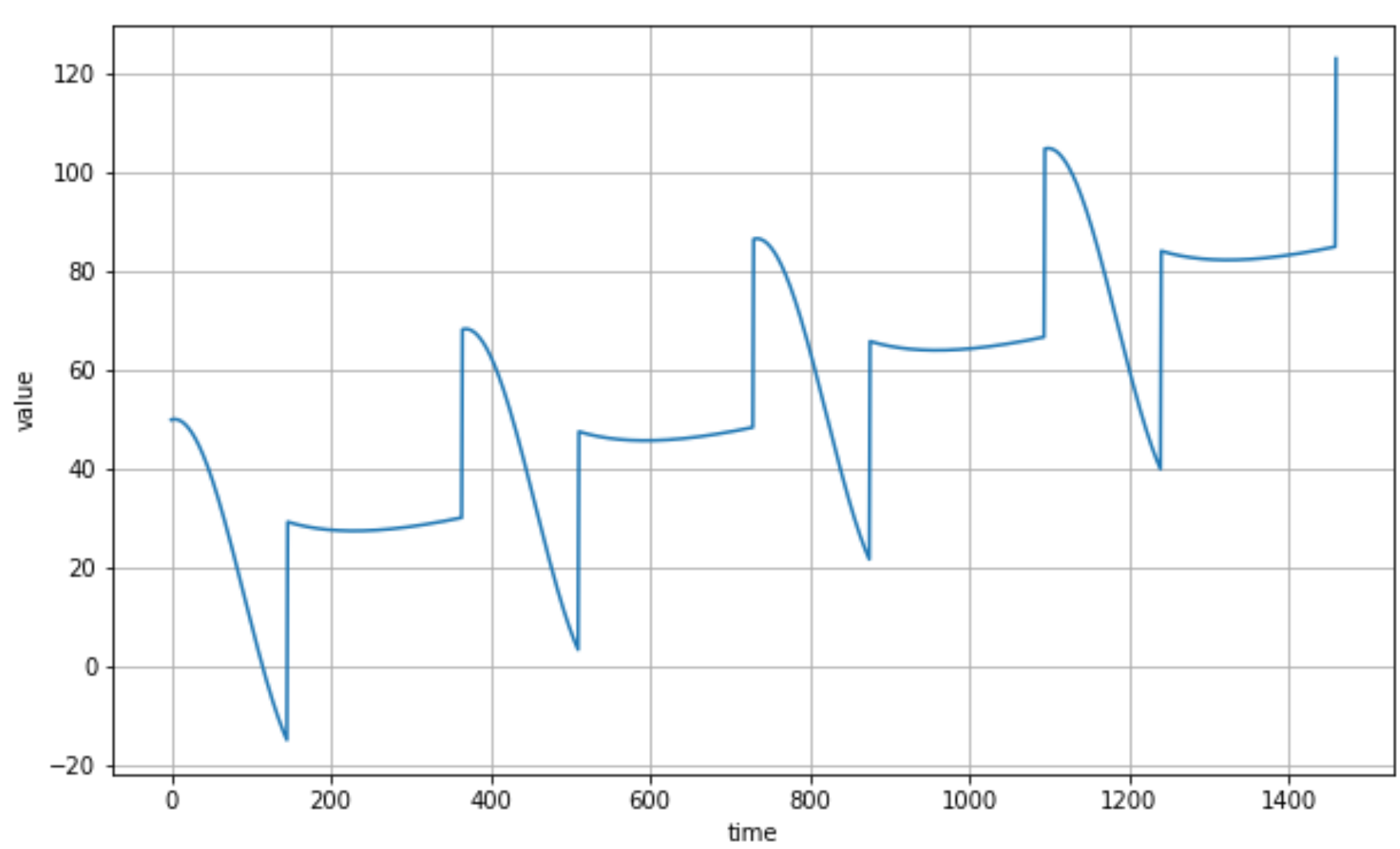
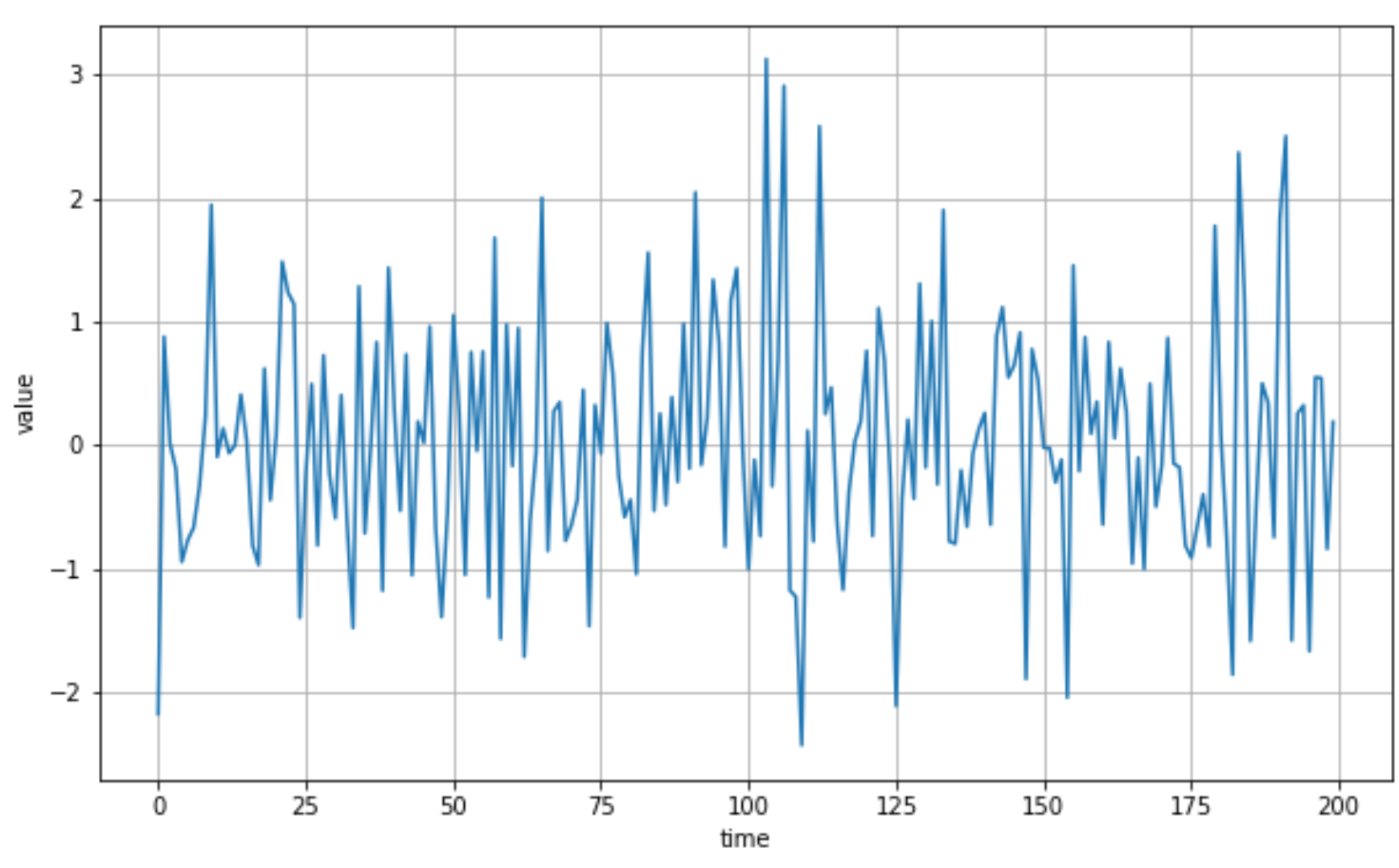


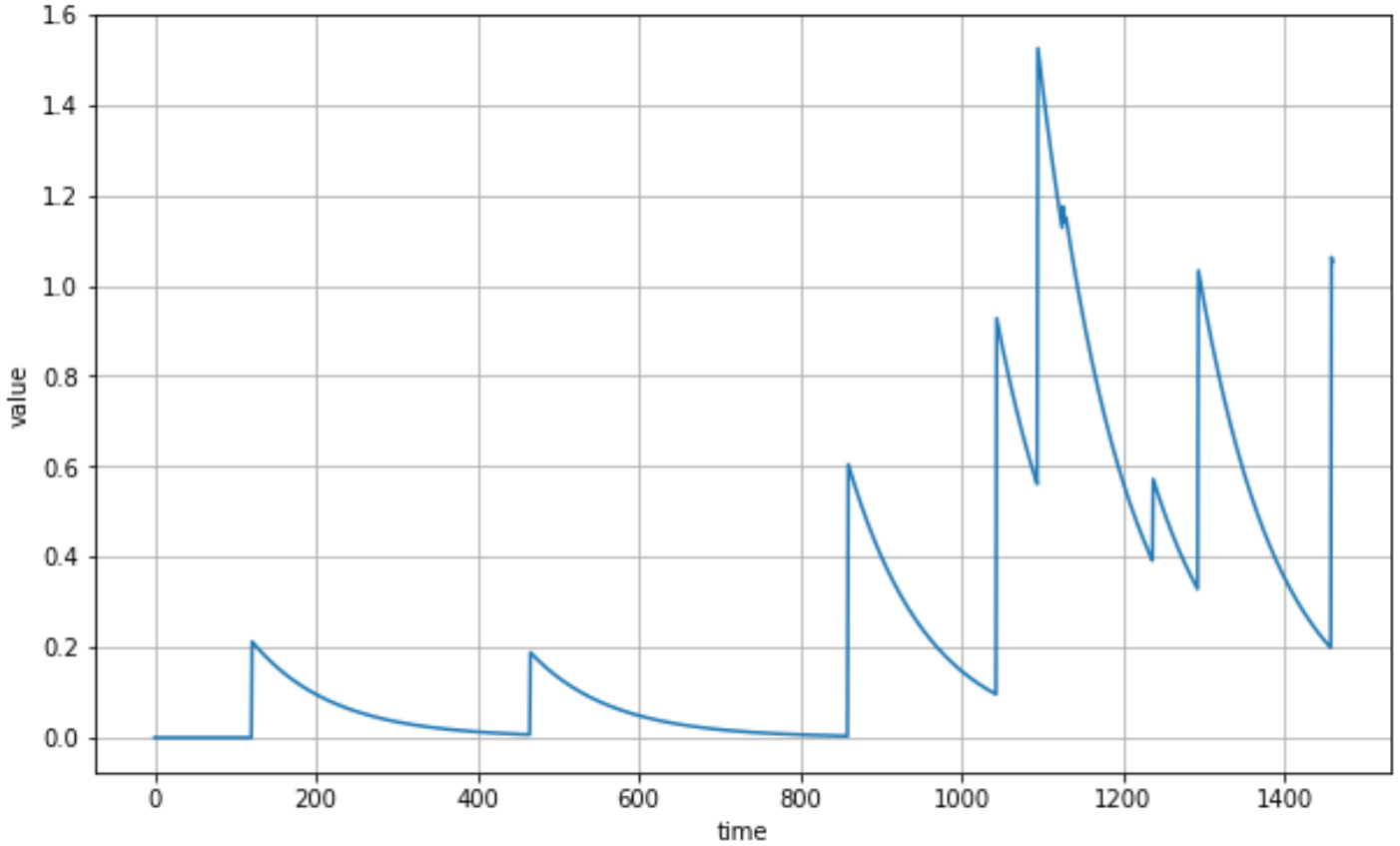
Chart Created by lmoroney@


February 2019
March 2019
April 2019
May 2019
June 2019

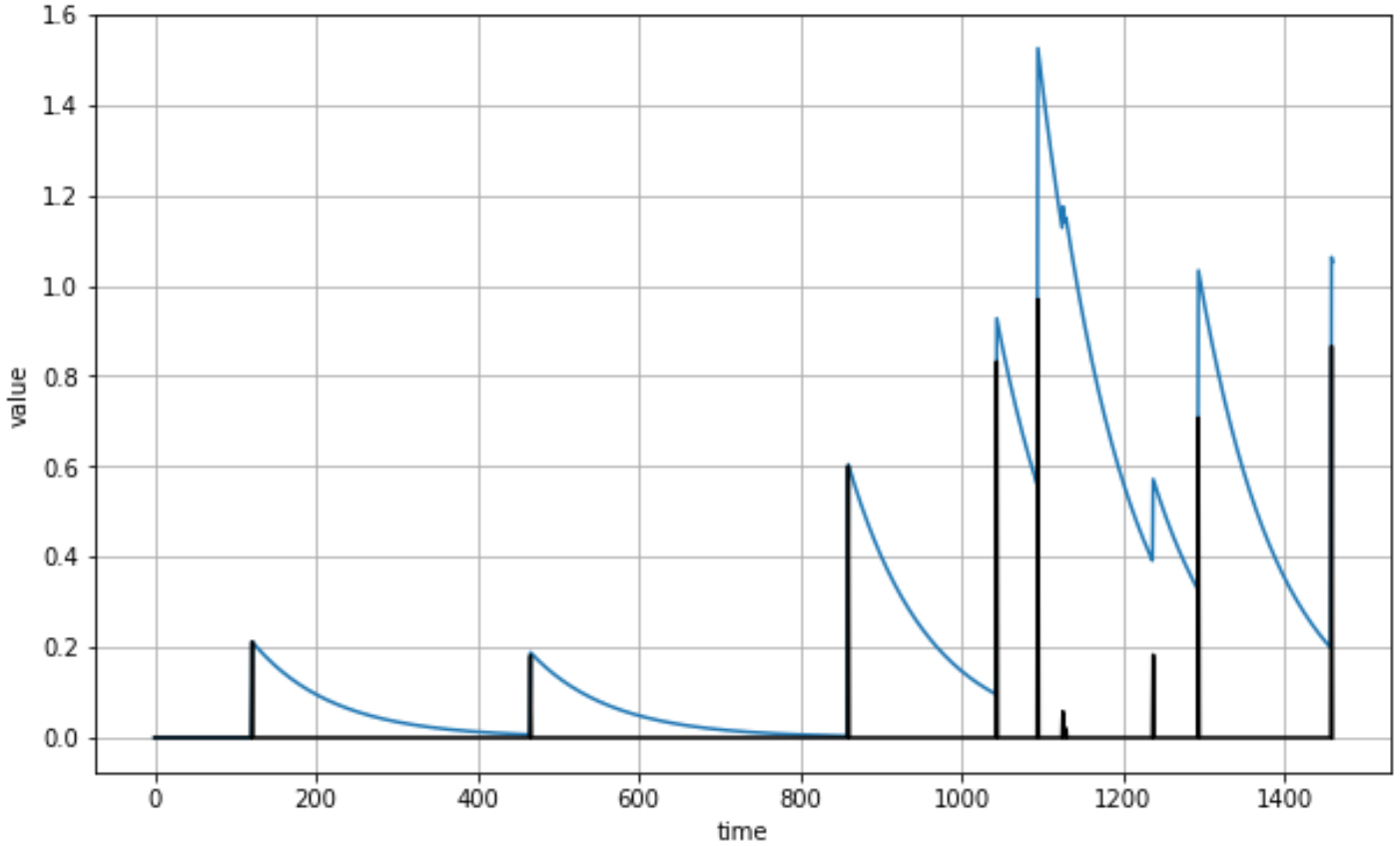

120
100
80
60
40
20
0
−20
value
0
200
400
600
800
1000
1200
1400
time


3
2
1
0
−1
−2
value
0
25
50
75
100
125
150
175
200
time

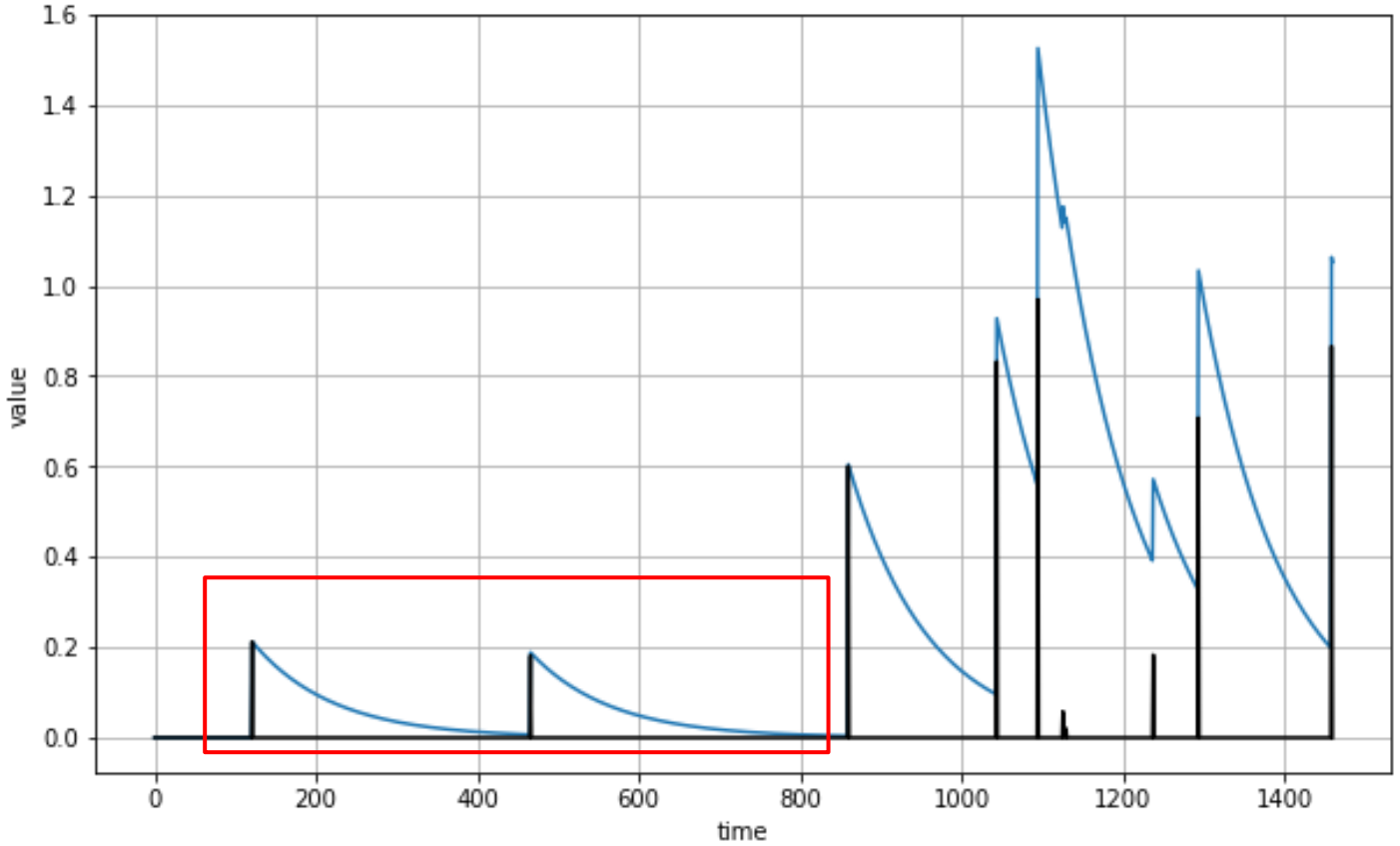
Autocorrelation



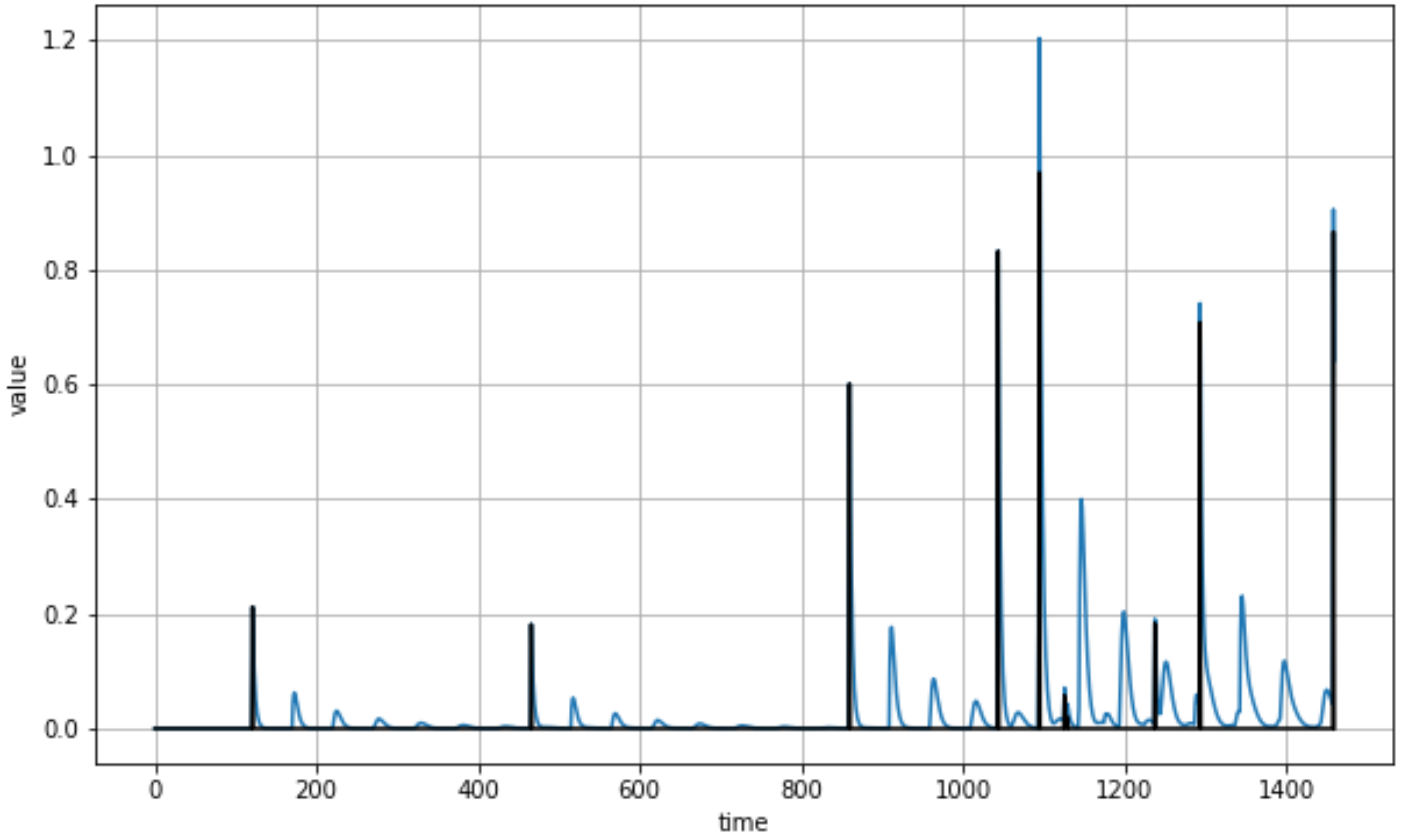
$v(t) = 0.99 \times v(t-1)$ + occasional spike



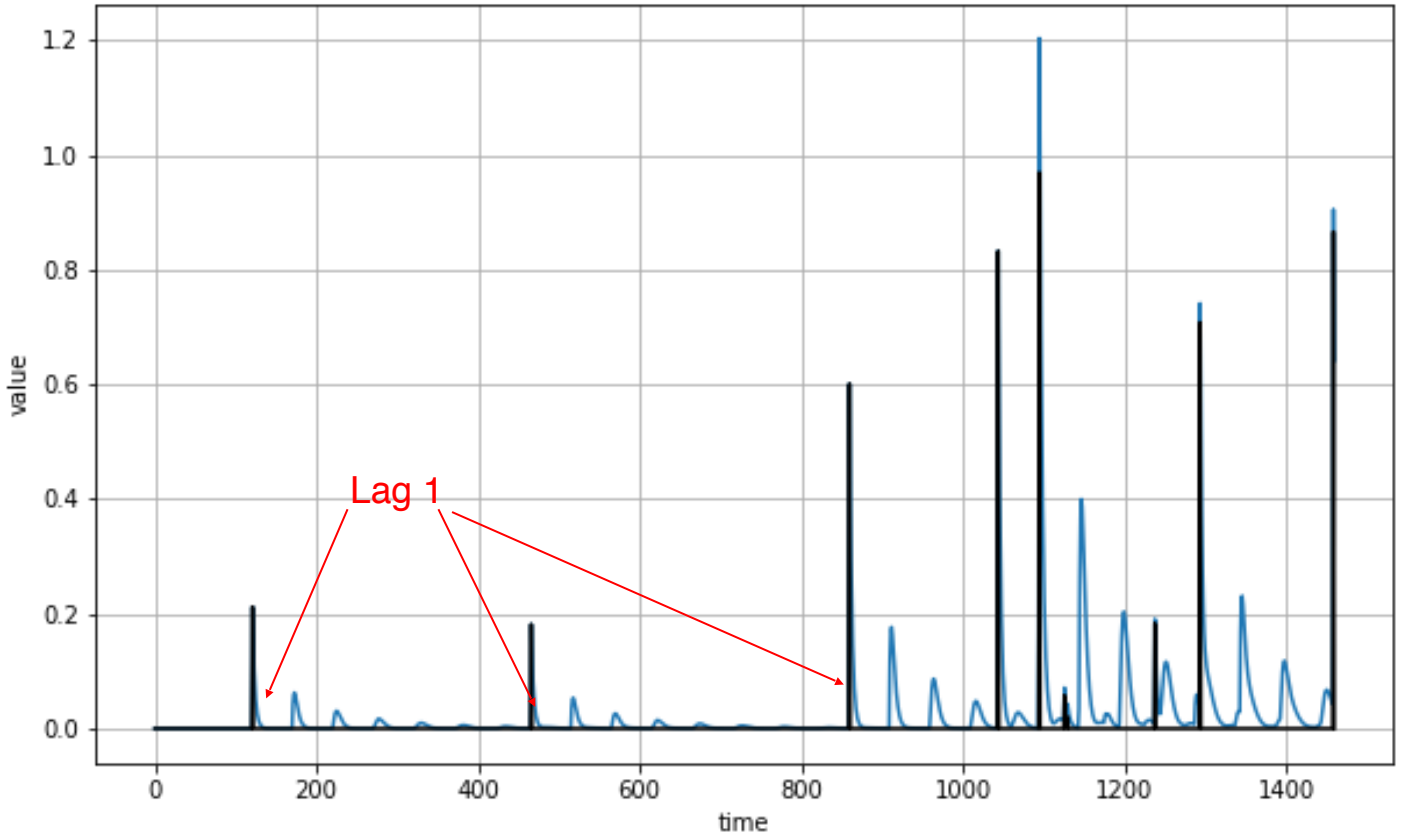
$v(t) = 0.99 \times v(t-1)$ + occasional spike



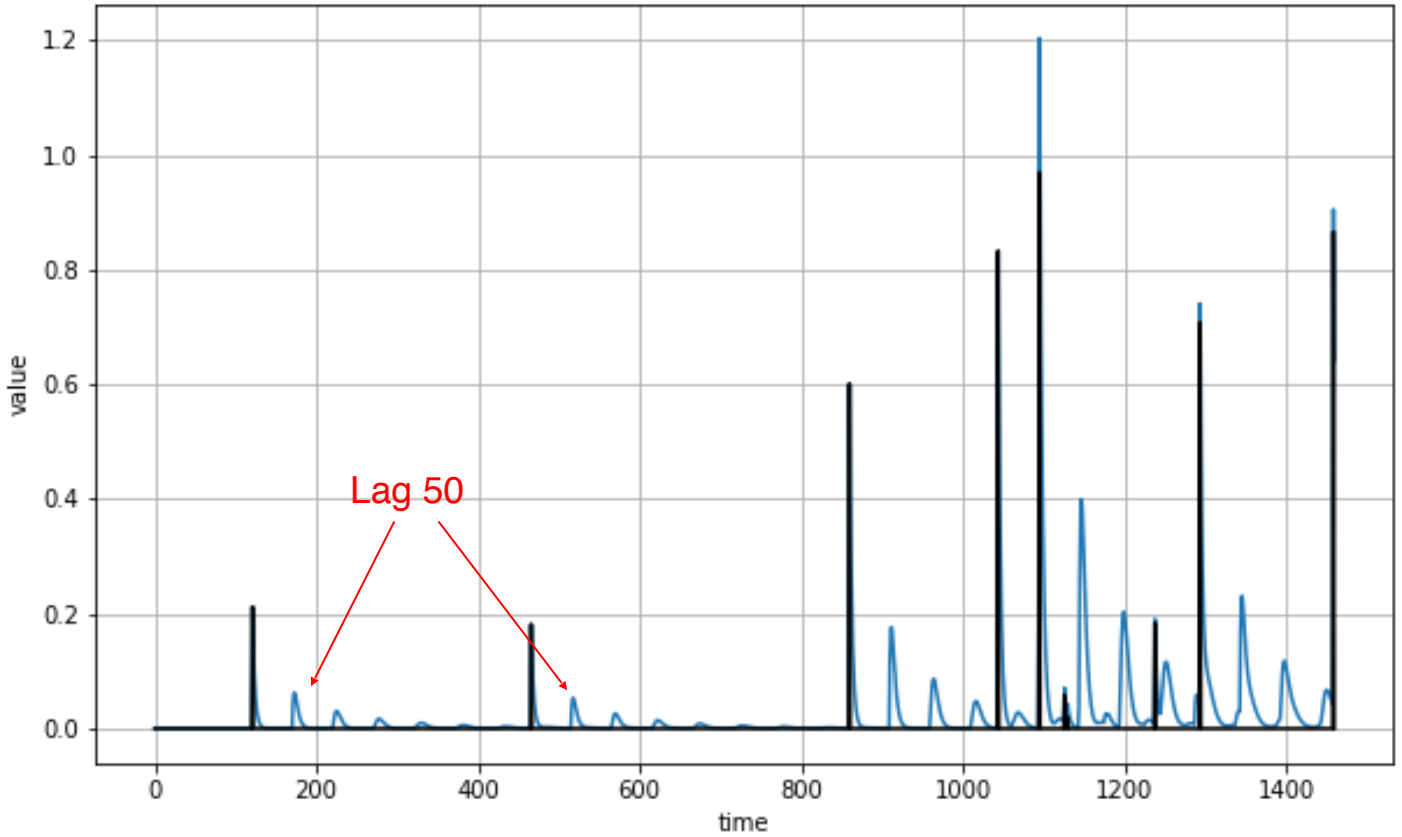
$$v(t) = 0.7 \times v(t-1) + 0.2 \times v(t-50) + \text{occasional spike}$$



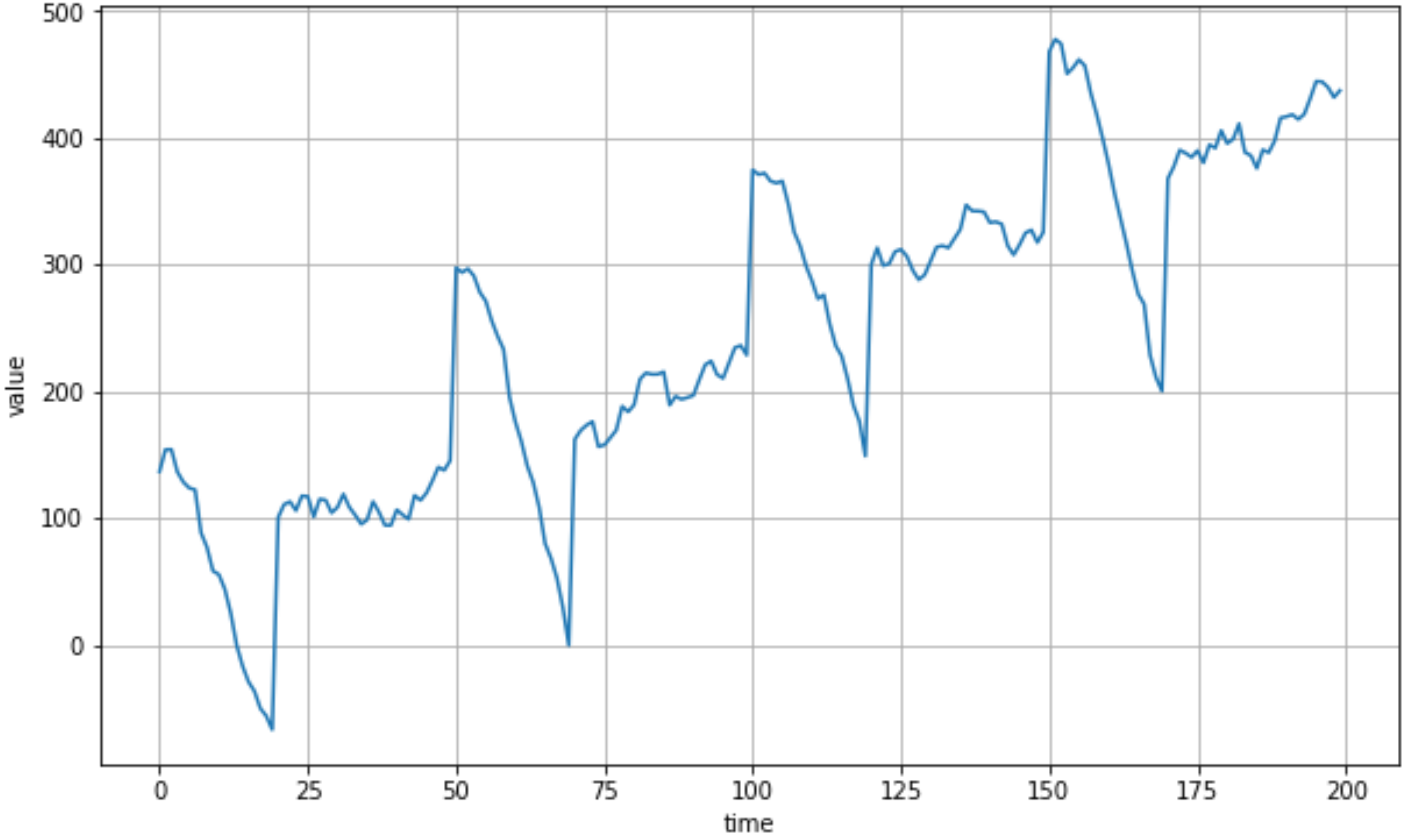
$v(t) = 0.7 \times v(t-1) + 0.2 \times v(t-50)$ + occasional spike



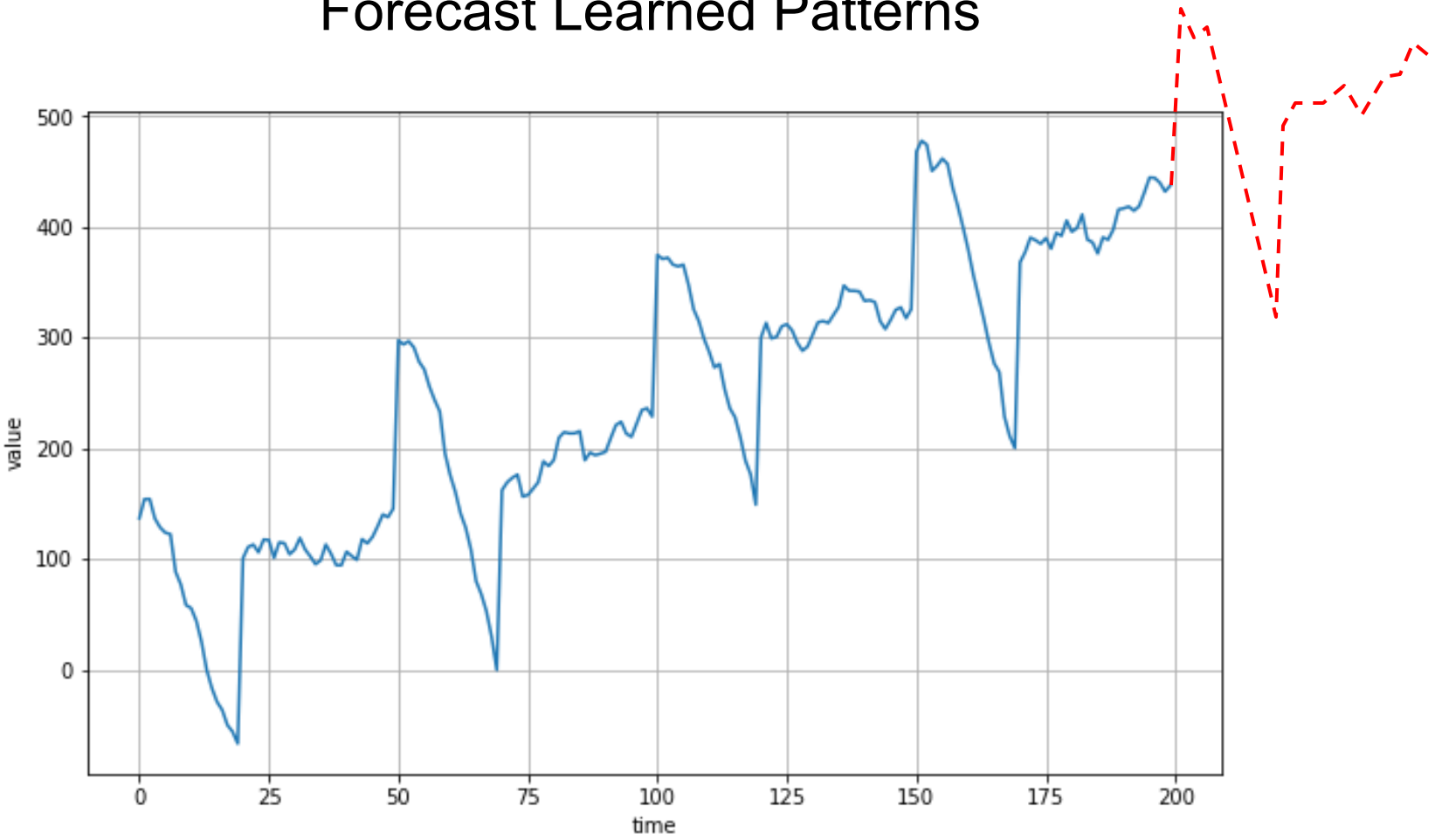
$v(t) = 0.7 \times v(t-1) + 0.2 \times v(t-50)$ + occasional spike



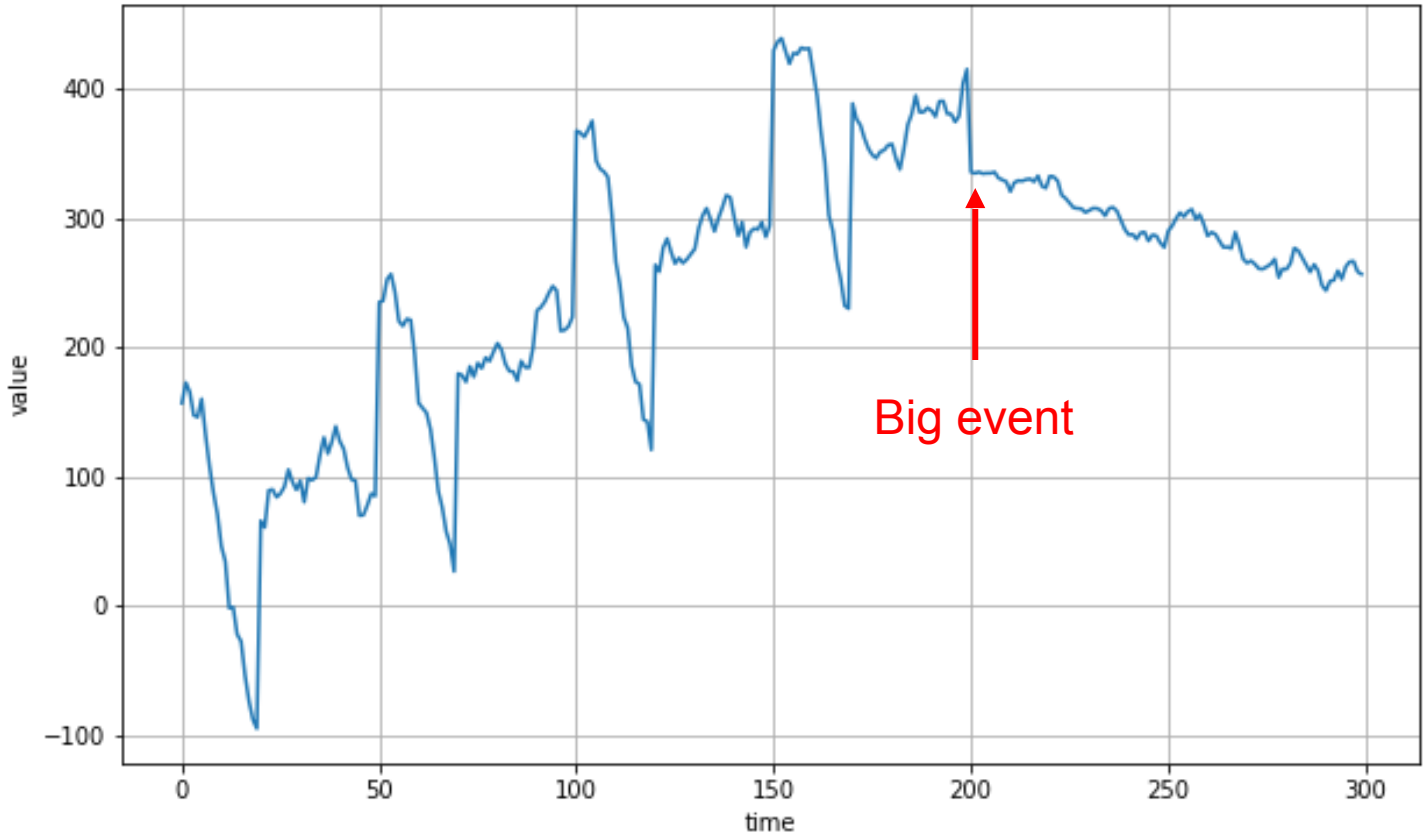
Trend + Seasonality + Autocorrelation + Noise



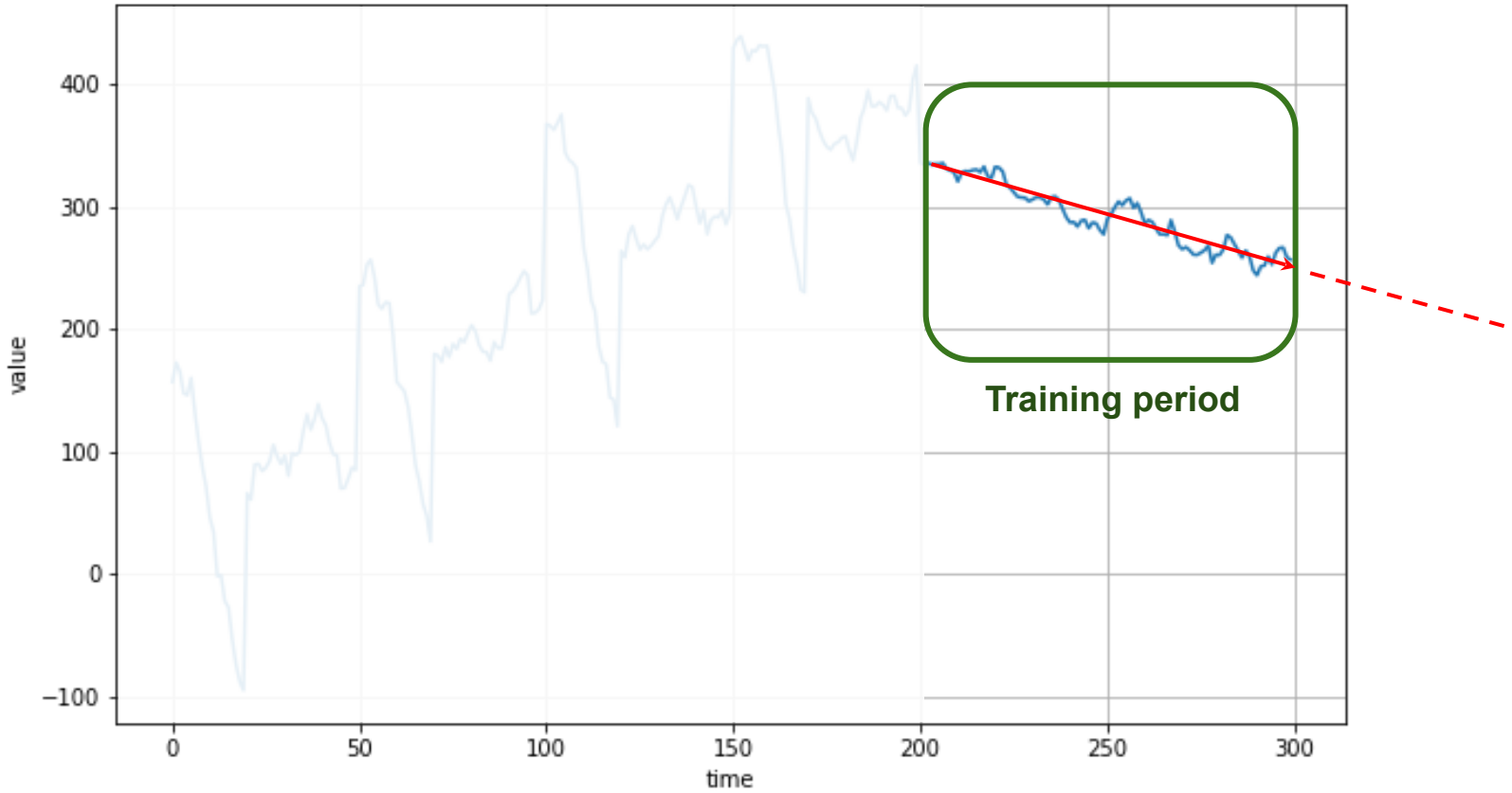
Forecast Learned Patterns



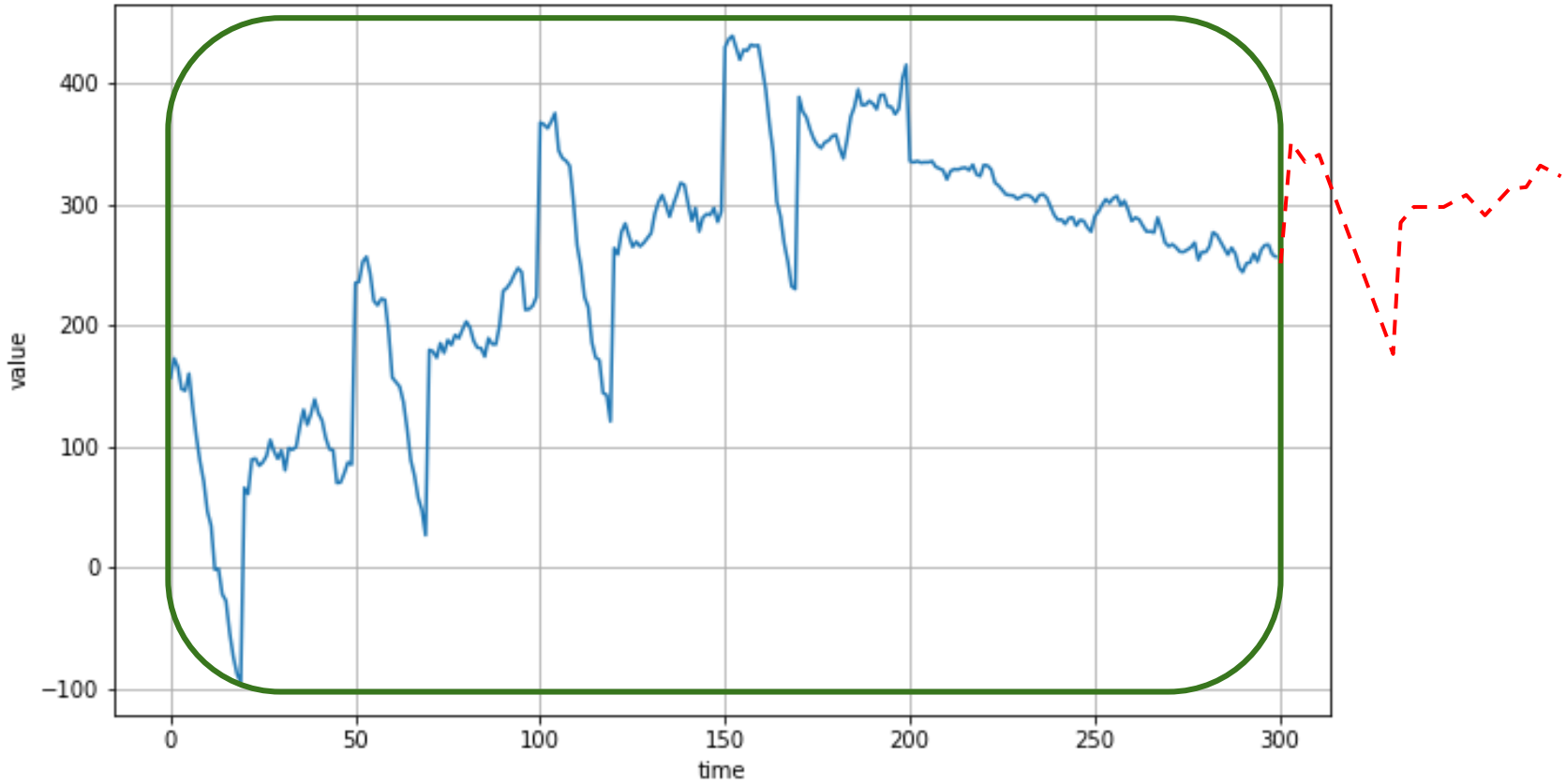
Non-Stationary Time Series



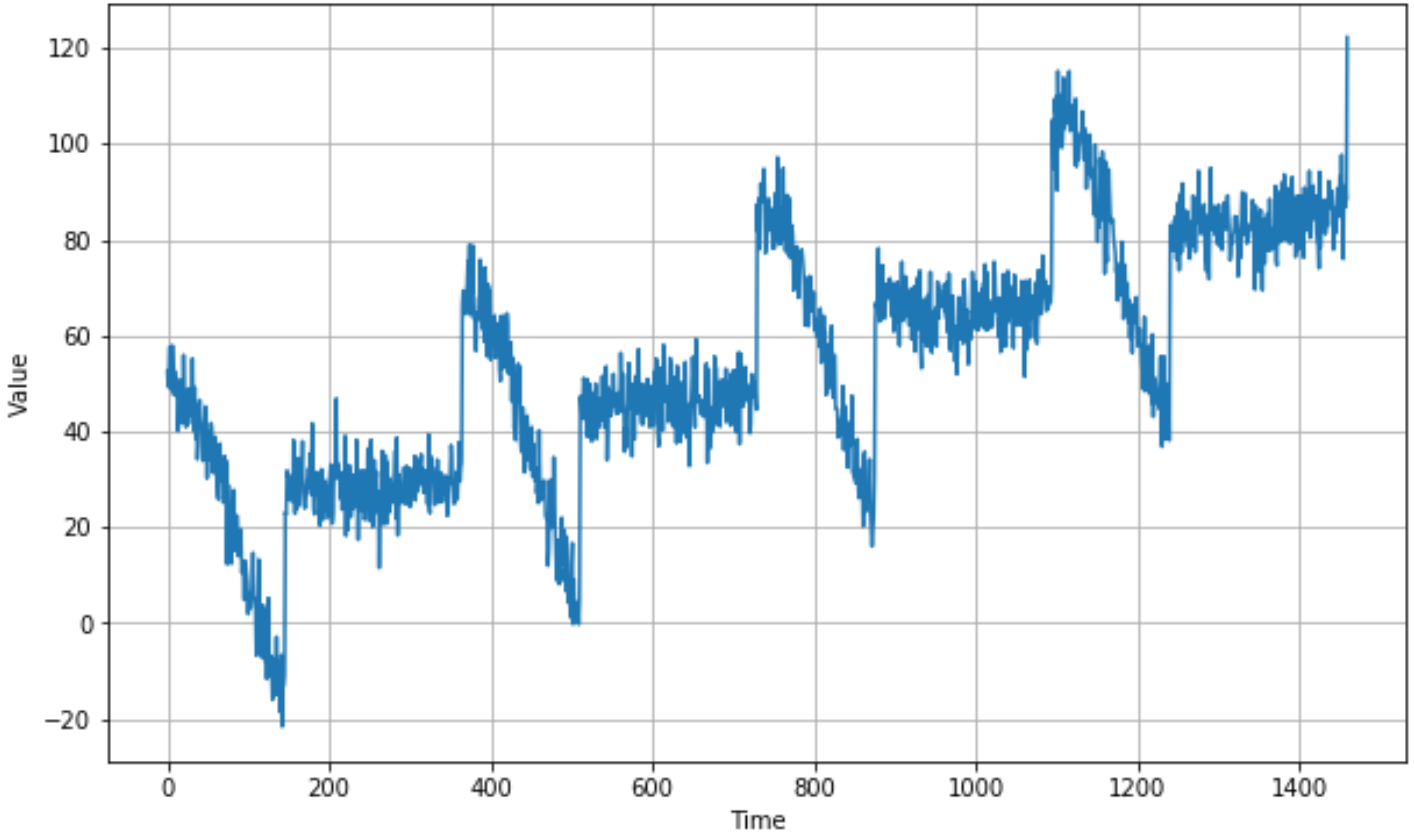
Non-Stationary Time Series



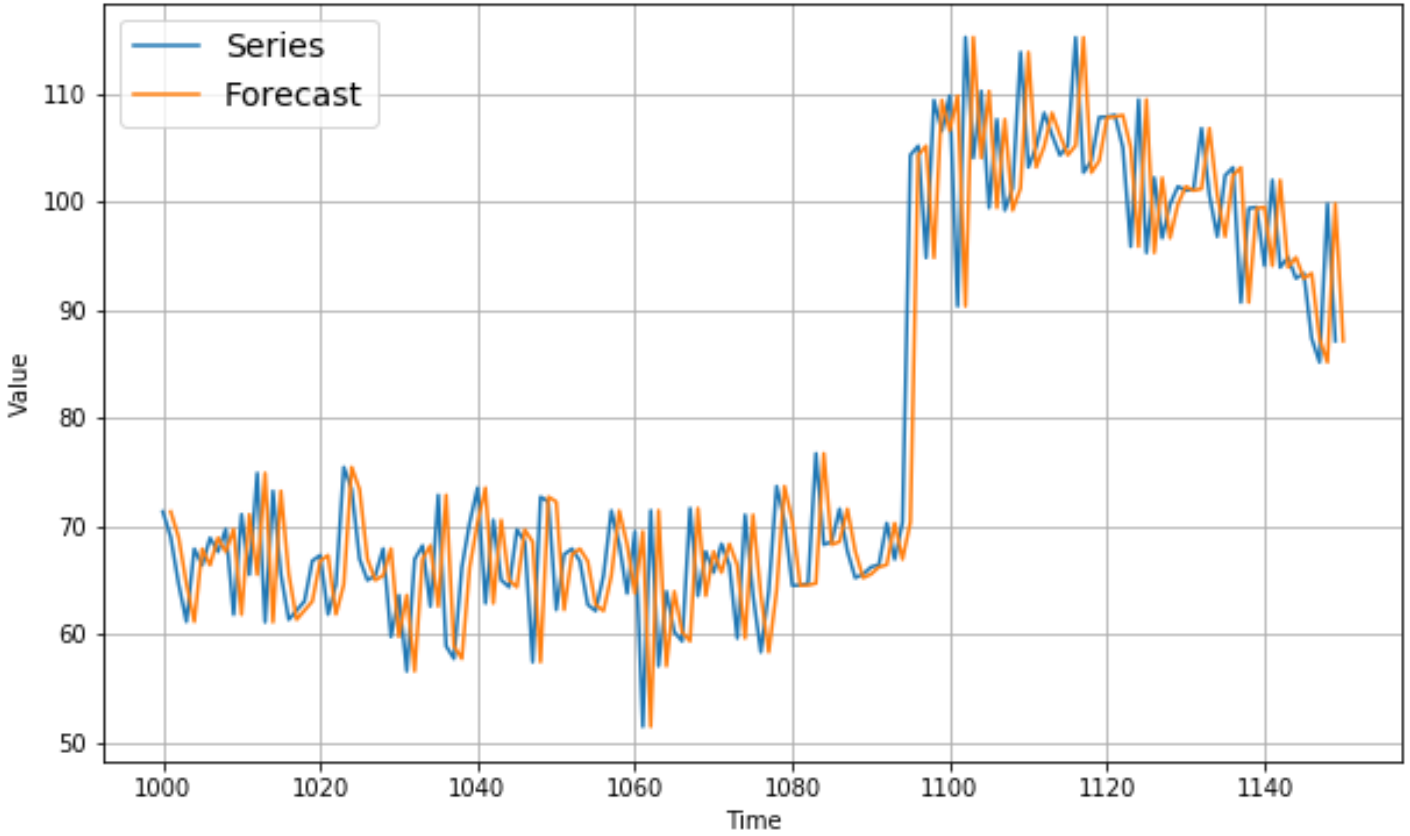
Non-Stationary Time Series



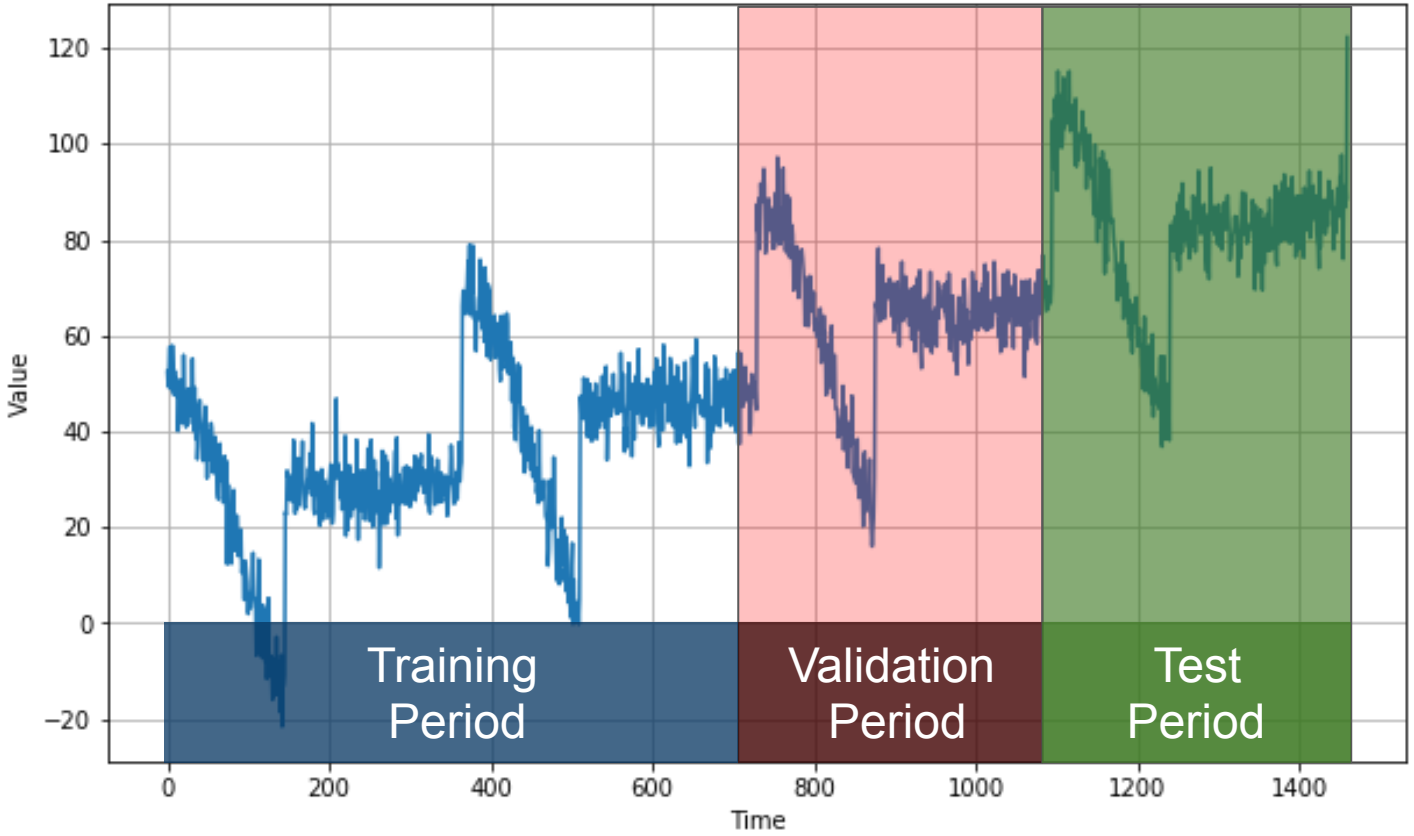
Trend + Seasonality + Noise



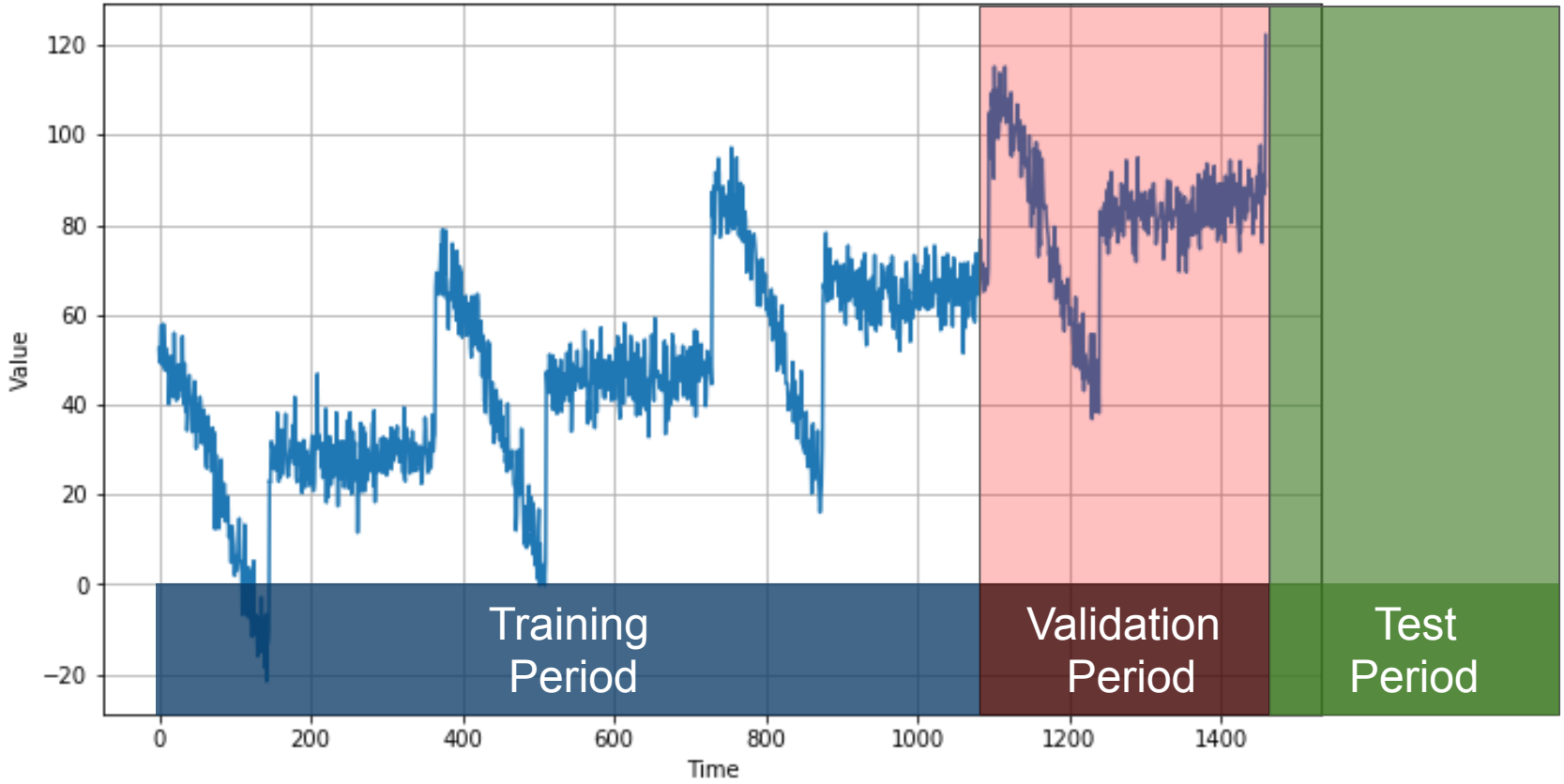
Naive Forecasting



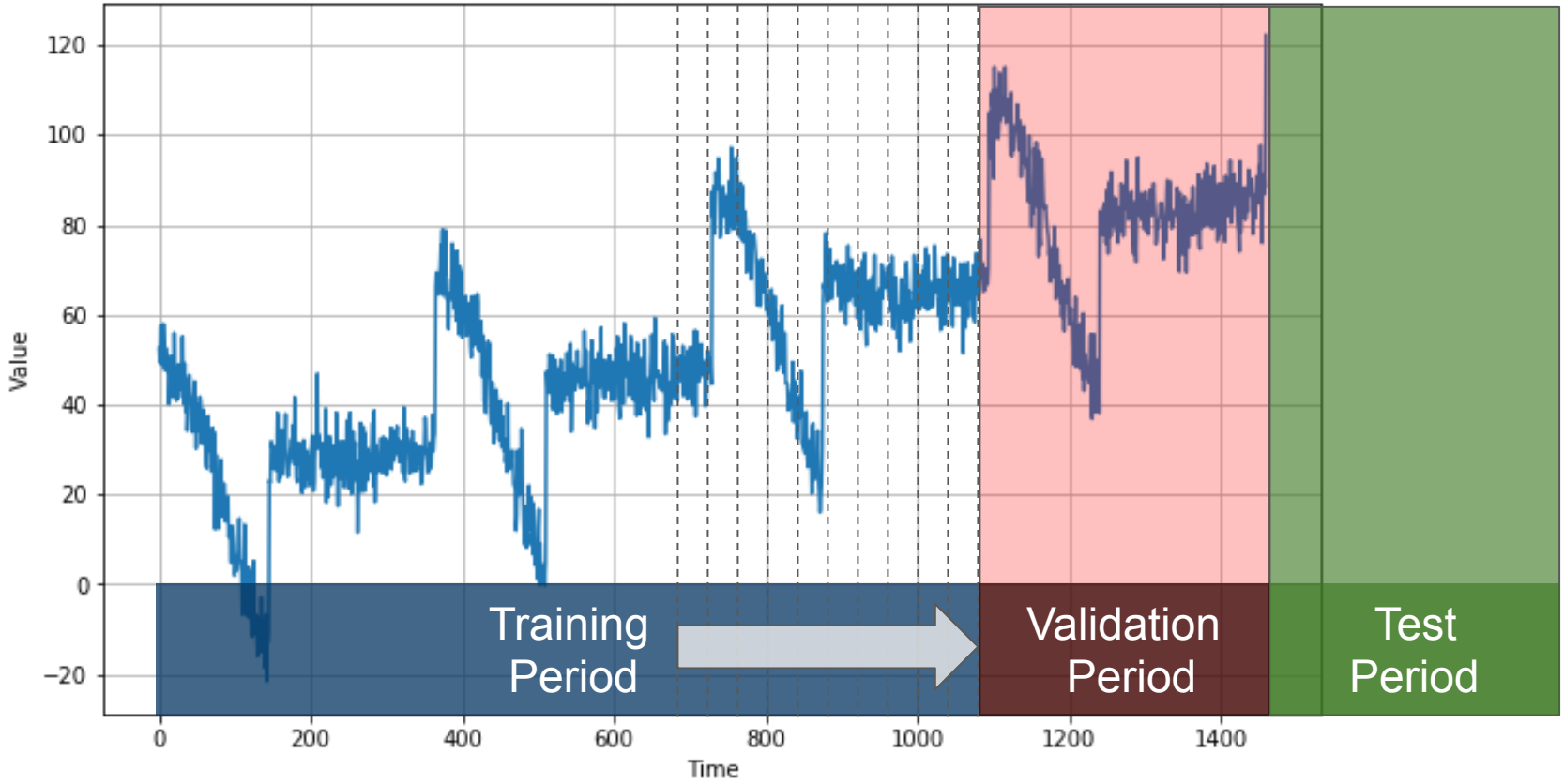
Fixed Partitioning



Fixed Partitioning



Roll-Forward Partitioning



Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

Metrics

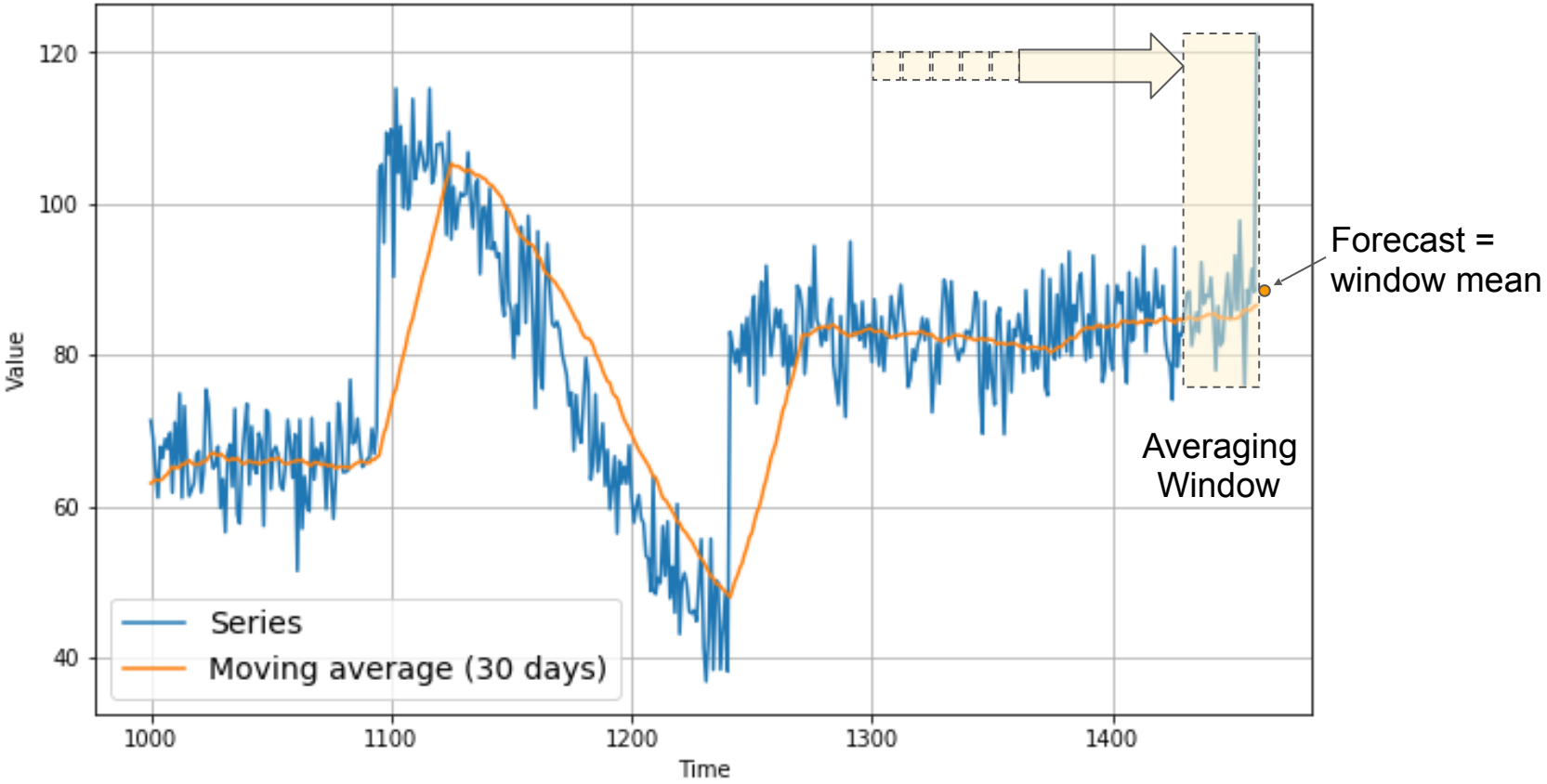
```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

Naive Forecast MAE

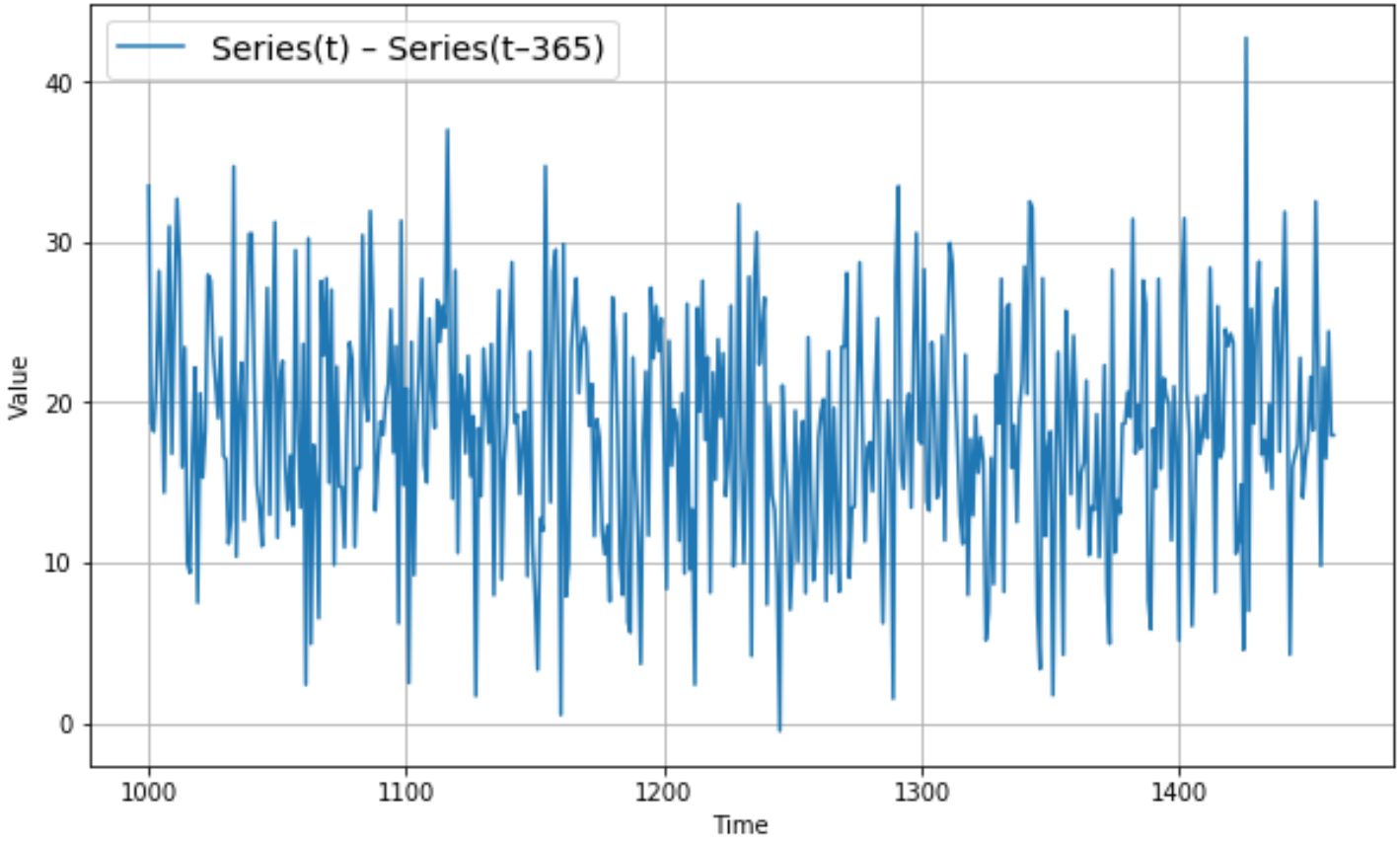
```
keras.metrics.mean_absolute_error(x_valid, naive_forecast).numpy()
```

```
5.937908515321673
```

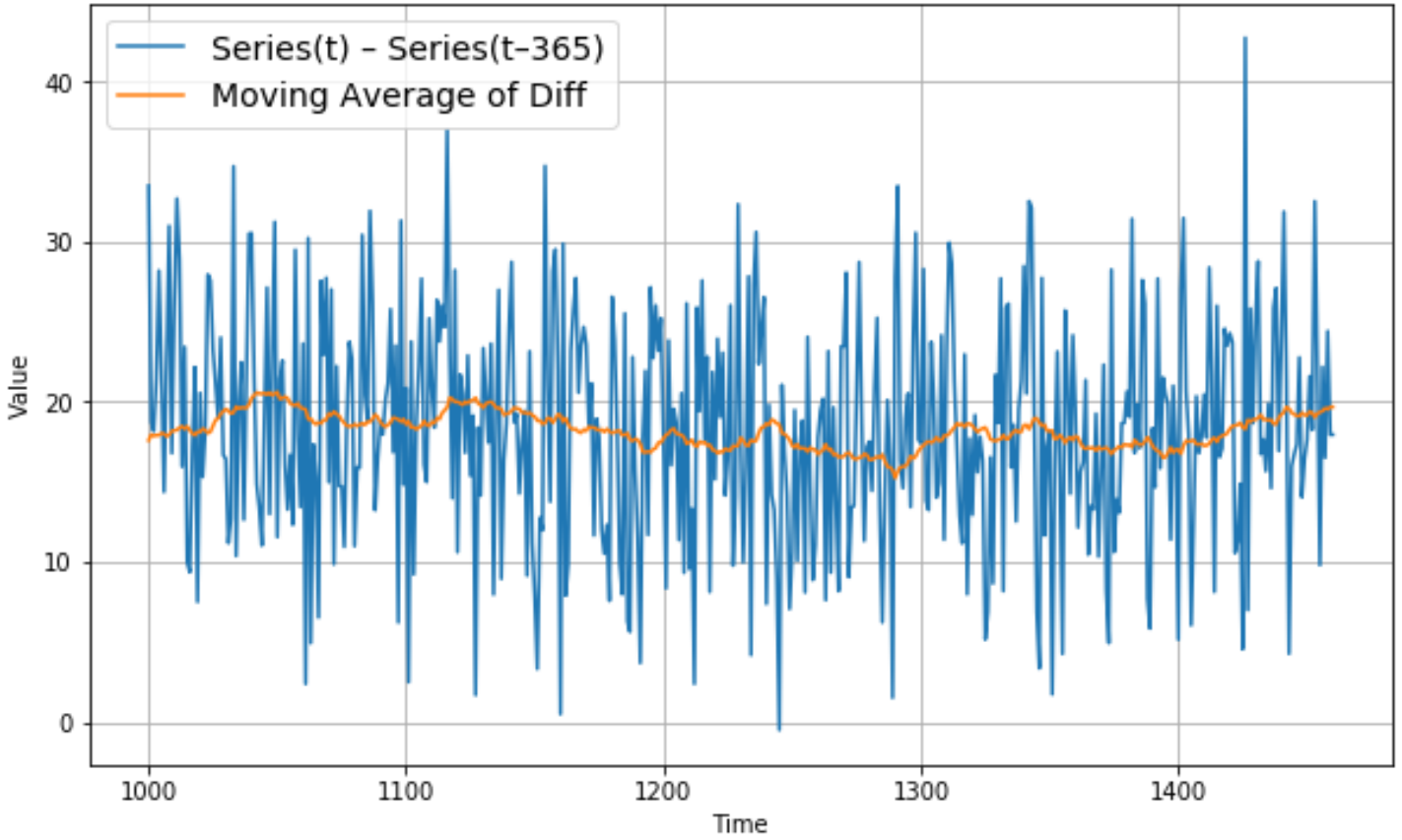
Moving Average



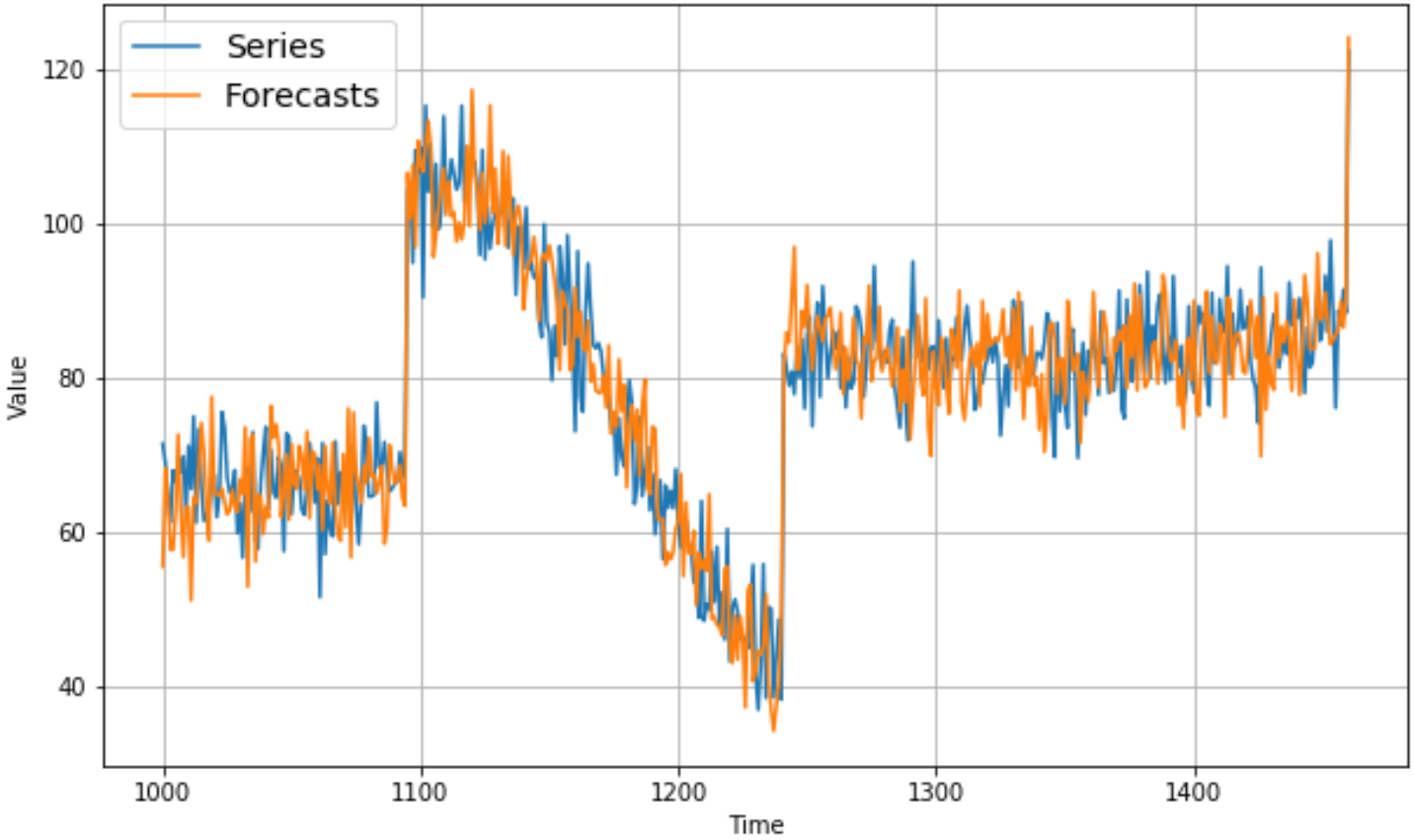
Differencing



Moving Average on Differenced Time Series

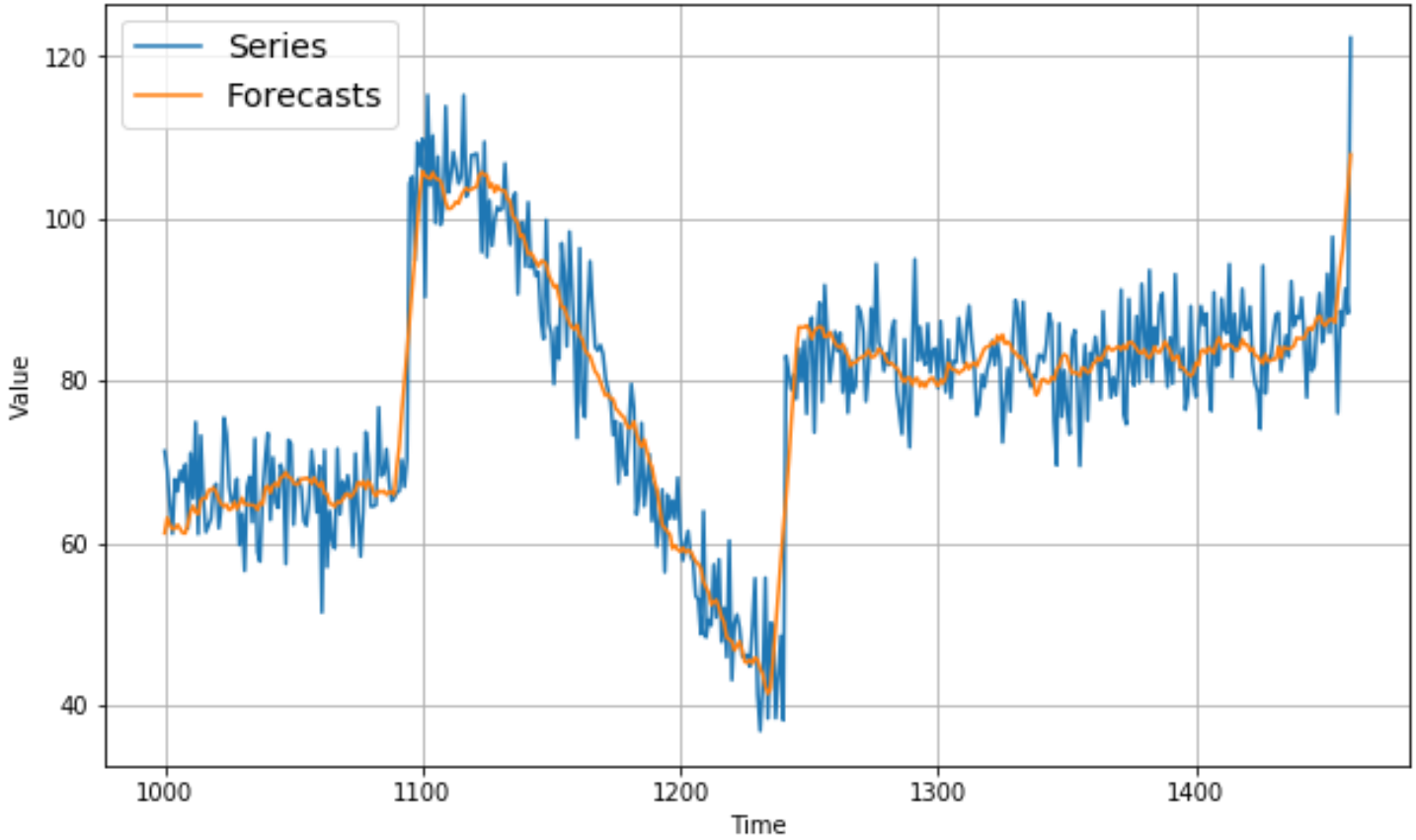


Restoring the Trend and Seasonality



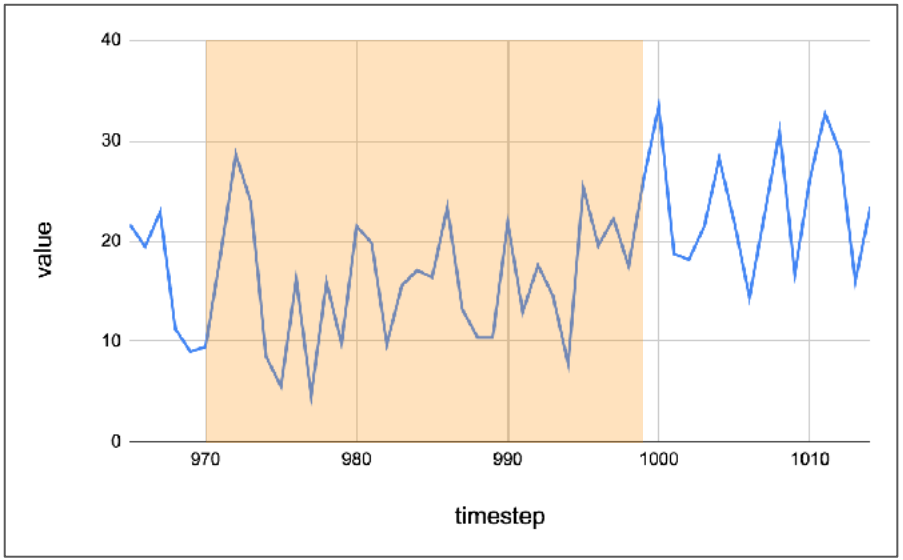
Forecasts = moving average of differenced series + series(t – 365)

Smoothing Both Past and Present Values

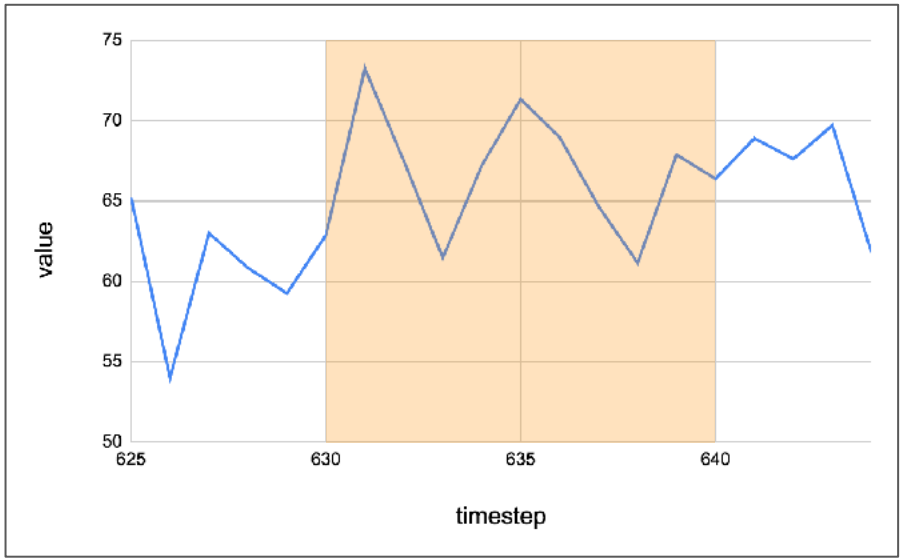


Forecasts = trailing moving average of differenced series + centered moving average of past series (t – 365)

Trailing Moving Average of Differenced Series
(zoomed at t_{1000} , window size = 30)



Centered Moving Average of Past Series (t - 365)
(zoomed at t_{635} , window size = 11)



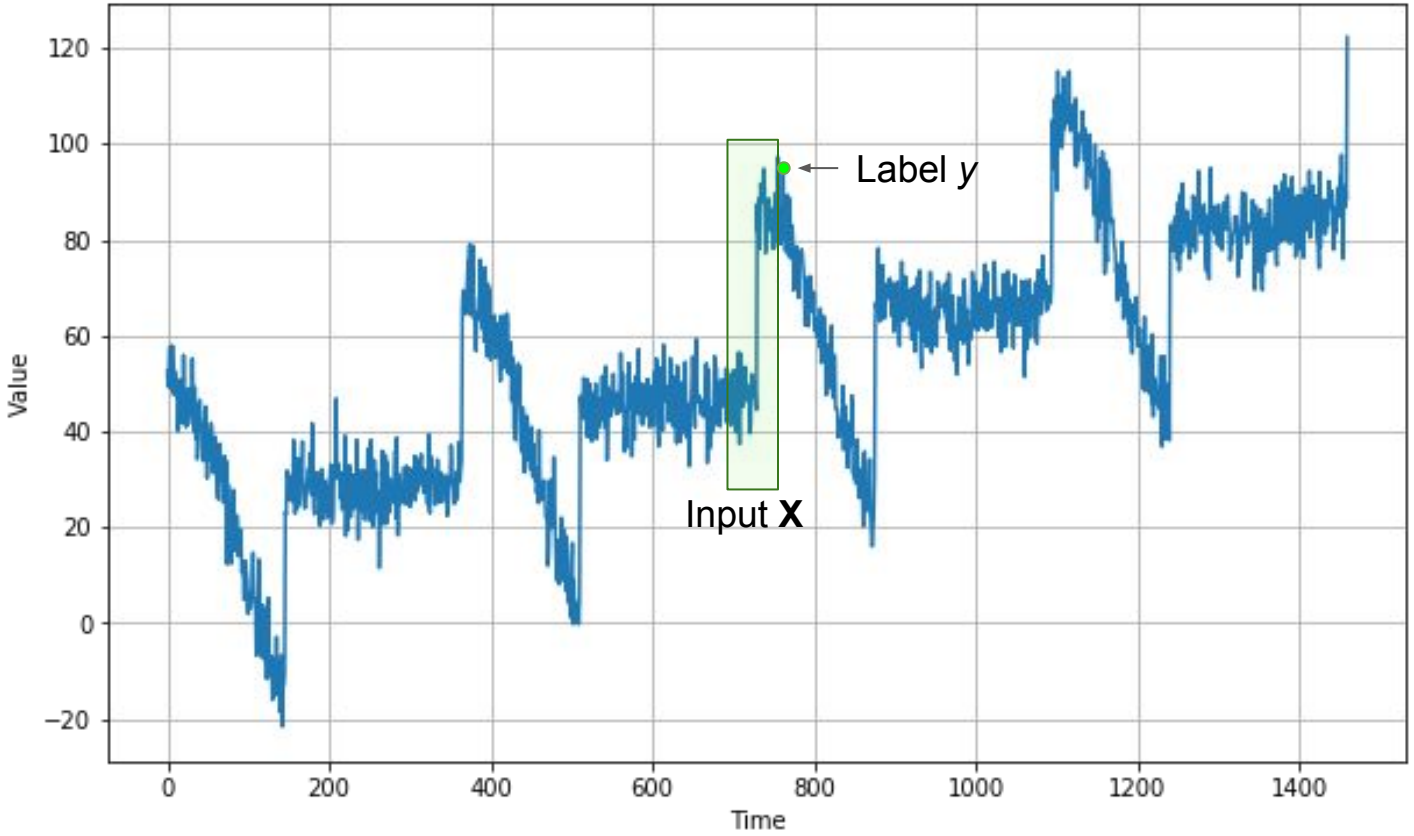
$TMA_{t1000} = (v_{t970} + v_{t971} + v_{t972} + \ldots v_{t999}) / 30$

$CMA_{t635} = (v_{t630} + v_{t631} + v_{t632} + \ldots. v_{t640}) / 11$

forecast at $t_{1000} = TMA_{t1000} + CMA_{t635}$

Copyright Notice

Machine Learning on Time Windows



```
dataset = tf.data.Dataset.range(10)
for val in dataset:
   print(val.numpy())
```

```python
dataset = tf.data.Dataset.range(10)
for val in dataset:
   print(val.numpy())
```

```
0
1
2
3
4
5
6
7
8
9
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
for window_dataset in dataset:
  for val in window_dataset:
    print(val.numpy(), end=" ")
  print()
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
5 6 7 8 9
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
for window in dataset:
  print(window.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
for window in dataset:
  print(window.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
for window in dataset:
  print(window.numpy())
```

```
[0 1 2 3 4]
[1 2 3 4 5]
[2 3 4 5 6]
[3 4 5 6 7]
[4 5 6 7 8]
[5 6 7 8 9]
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
[0 1 2 3] [4]
[1 2 3 4] [5]
[2 3 4 5] [6]
[3 4 5 6] [7]
[4 5 6 7] [8]
[5 6 7 8] [9]
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
for x,y in dataset:
  print(x.numpy(), y.numpy())
```

```
[3 4 5 6] [7]
[4 5 6 7] [8]
[1 2 3 4] [5]
[2 3 4 5] [6]
[5 6 7 8] [9]
[0 1 2 3] [4]
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x,y in dataset:
  print("x = ", x.numpy())
  print("y = ", y.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x,y in dataset:
  print("x = ", x.numpy())
  print("y = ", y.numpy())
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x,y in dataset:
  print("x = ", x.numpy())
  print("y = ", y.numpy())
```

```
x =  [[4 5 6 7] [1 2 3 4]]
y =  [[8] [5]]
x =  [[3 4 5 6] [2 3 4 5]]
y =  [[7] [6]]
x =  [[5 6 7 8] [0 1 2 3]]
y =  [[9] [4]]
```

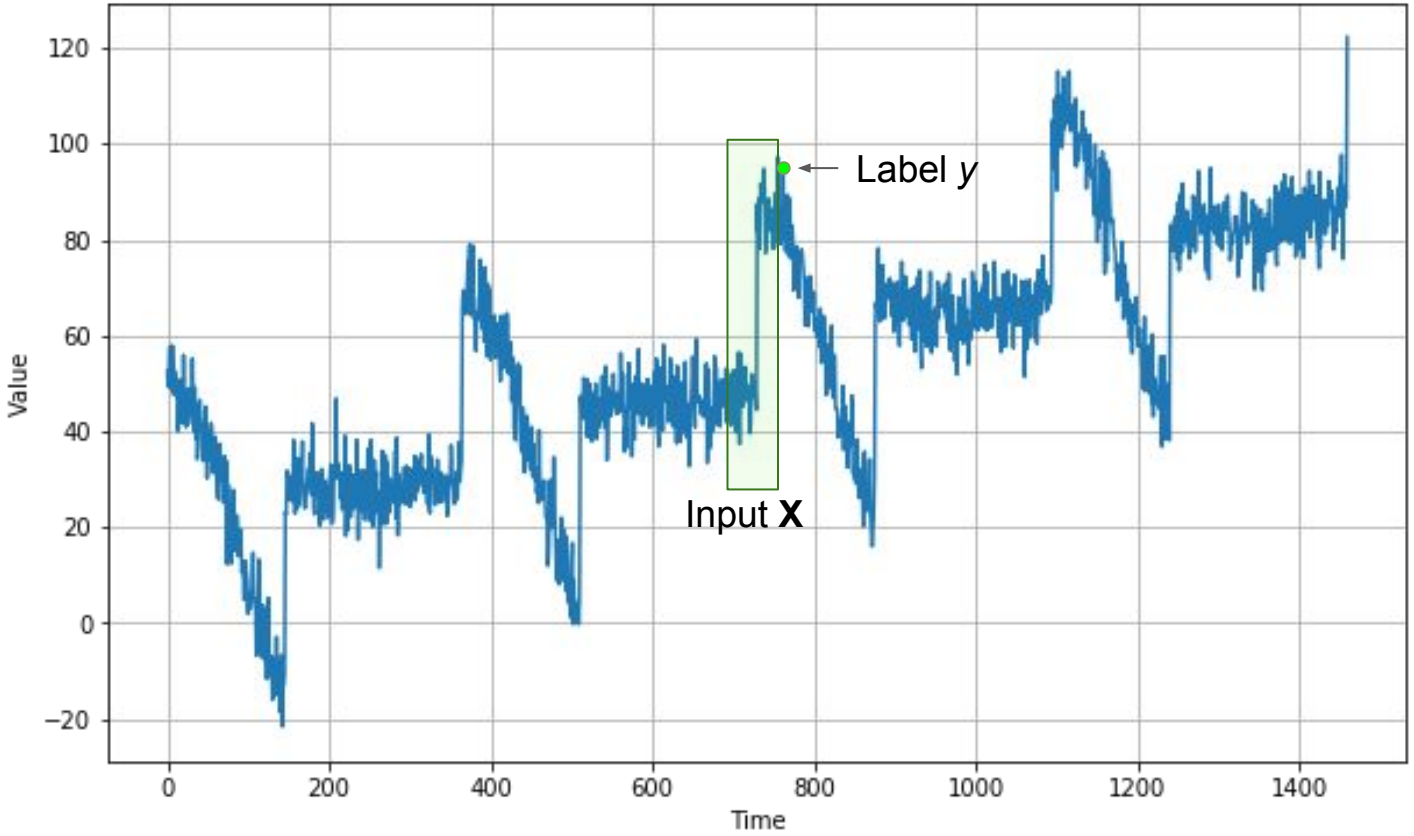
```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x,y in dataset:
  print("x = ", x.numpy())
  print("y = ", y.numpy())
```

```
x =  [[4 5 6 7] [1 2 3 4]]
y =  [[8] [5]]
x =  [[3 4 5 6] [2 3 4 5]]
y =  [[7] [6]]
x =  [[5 6 7 8] [0 1 2 3]]
y =  [[9] [4]]
```

```
dataset = tf.data.Dataset.range(10)
dataset = dataset.window(5, shift=1, drop_remainder=True)
dataset = dataset.flat_map(lambda window: window.batch(5))
dataset = dataset.map(lambda window: (window[:-1], window[-1]))
dataset = dataset.shuffle(buffer_size=10)
dataset = dataset.batch(2).prefetch(1)
for x,y in dataset:
  print("x = ", x.numpy())
  print("y = ", y.numpy())
```

```
x =  [[4 5 6 7] [1 2 3 4]]
y =  [[8] [5]]
x =  [[3 4 5 6] [2 3 4 5]]
y =  [[7] [6]]
x =  [[5 6 7 8] [0 1 2 3]]
y =  [[9] [4]]
```

Machine Learning on Time Windows



```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]
```

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]
```

```
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000

dataset = windowed_dataset(series, window_size, batch_size, shuffle_buffer_size)
l0 = tf.keras.layers.Dense(1, input_shape=[window_size])
model = tf.keras.models.Sequential([l0])
```

```
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000

dataset = windowed_dataset(series, window_size, batch_size, shuffle_buffer_size)
l0 = tf.keras.layers.Dense(1, input_shape=[window_size])
model = tf.keras.models.Sequential([l0])
```

```
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000

dataset = windowed_dataset(series, window_size, batch_size, shuffle_buffer_size)
l0 = tf.keras.layers.Dense(1, input_shape=[window_size])
model = tf.keras.models.Sequential([l0])
```

```
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000

dataset = windowed_dataset(series, window_size, batch_size, shuffle_buffer_size)
l0 = tf.keras.layers.Dense(1, input_shape=[window_size])
model = tf.keras.models.Sequential([l0])
```

```
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000

dataset = windowed_dataset(series, window_size, batch_size, shuffle_buffer_size)
l0 = tf.keras.layers.Dense(1, input_shape=[window_size])
model = tf.keras.models.Sequential([l0])
```

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

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

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

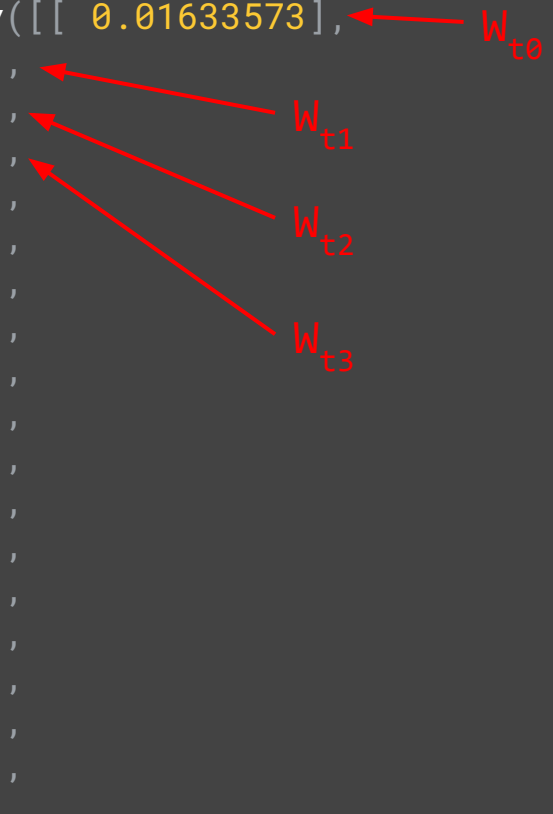
```
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

```
print("Layer weights {}".format(l0.get_weights()))
```

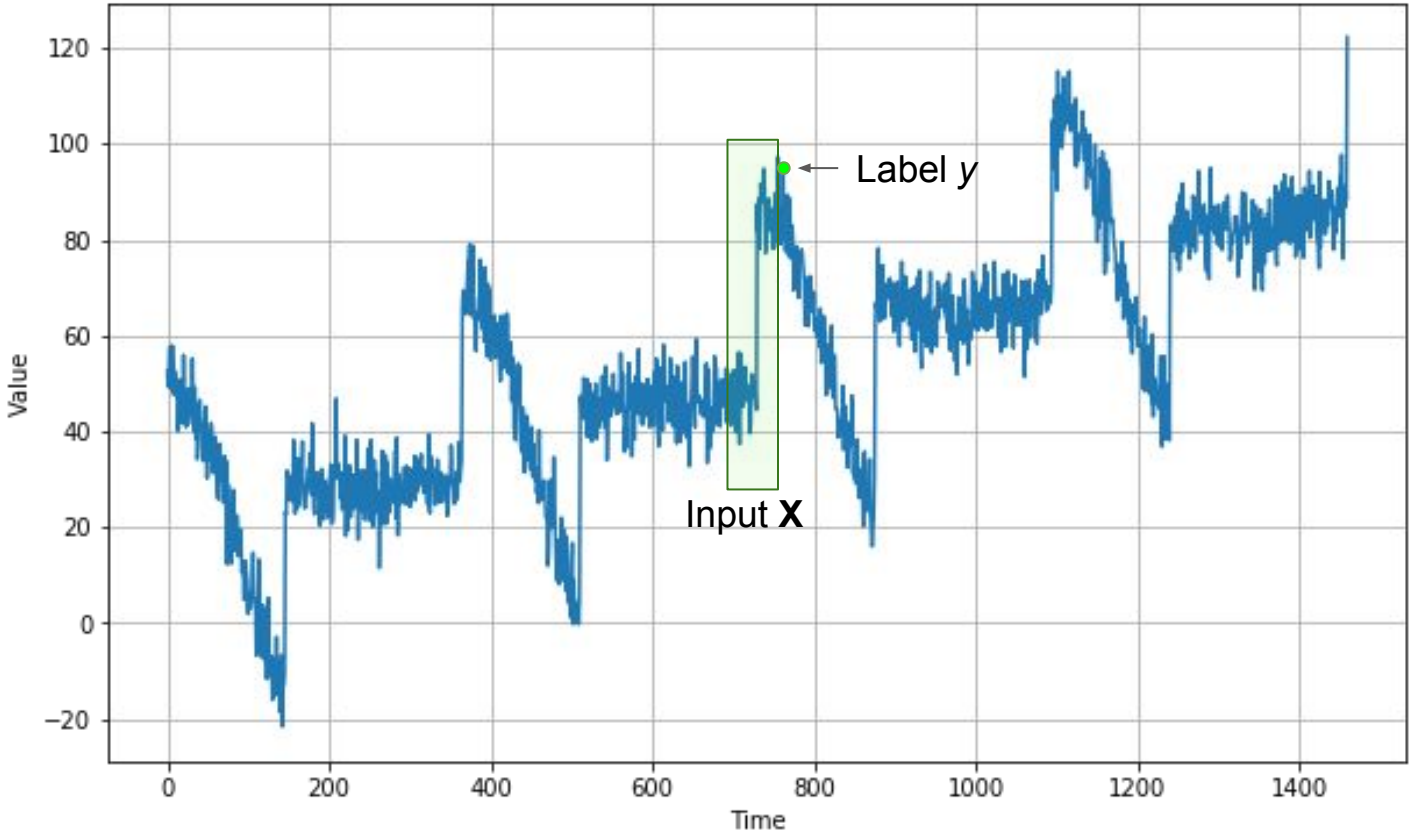
```
print("Layer weights {}".format(l0.get_weights()))
```

```
Layer weights [array([[ 0.01633573],
       [-0.02911791],
       [ 0.00845617],
       [-0.02175158],
       [ 0.04962169],
       [-0.03212642],
       [-0.02596855],
       [-0.00689476],
       [ 0.0616533 ],
       [-0.00668752],
       [-0.02735964],
       [ 0.0377918 ],
       [-0.02855931],
       [ 0.05299238],
       [-0.0121608 ],
       [ 0.00138755],
       [ 0.0905595 ],
       [ 0.19994621],
       [ 0.2556632 ],
       [ 0.41660047]], dtype=float32), array([0.01430958], dtype=float32)]
```

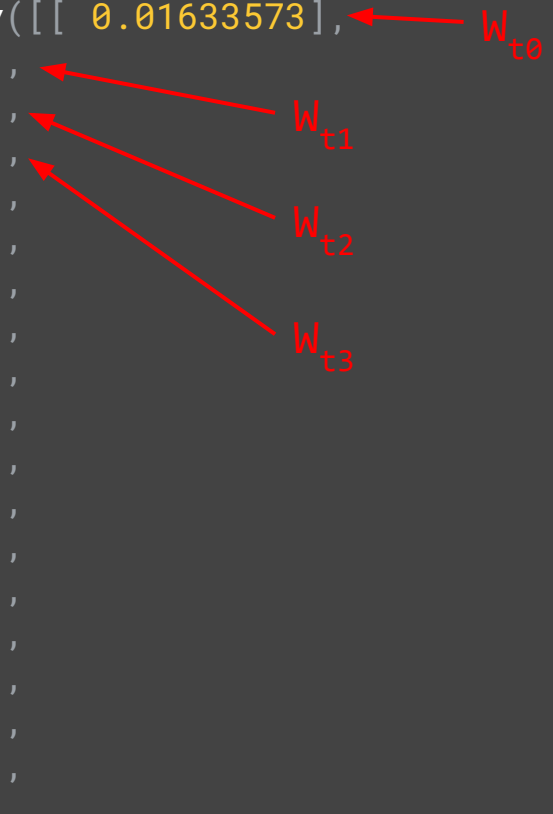
```
print("Layer weights {}".format(l0.get_weights()))

Layer weights [array([[ 0.01633573],
       [-0.02911791],
       [ 0.00845617],
       [-0.02175158],
       [ 0.04962169],
       [-0.03212642],
       [-0.02596855],
       [-0.00689476],
       [ 0.0616533 ],
       [-0.00668752],
       [-0.02735964],
       [ 0.0377918 ],
       [-0.02855931],
       [ 0.05299238],
       [-0.0121608 ],
       [ 0.00138755],
       [ 0.0905595 ],
       [ 0.19994621],
       [ 0.2556632 ],
       [ 0.41660047]], dtype=float32), array([0.01430958], dtype=float32)]
```



Machine Learning on Time Windows



```
print("Layer weights {}".format(l0.get_weights()))

Layer weights [array([[ 0.01633573],
       [-0.02911791],
       [ 0.00845617],
       [-0.02175158],
       [ 0.04962169],
       [-0.03212642],
       [-0.02596855],
       [-0.00689476],
       [ 0.0616533 ],
       [-0.00668752],
       [-0.02735964],
       [ 0.0377918 ],
       [-0.02855931],
       [ 0.05299238],
       [-0.0121608 ],
       [ 0.00138755],
       [ 0.0905595 ],
       [ 0.19994621],
       [ 0.2556632 ],
       [ 0.41660047]], dtype=float32), array([0.01430958], dtype=float32)]
```



```
print("Layer weights {}".format(l0.get_weights()))

Layer weights [array([[ 0.01633573],
       [-0.02911791],
       [ 0.00845617],
       [-0.02175158],
       [ 0.04962169],
       [-0.03212642],
       [-0.02596855],
       [-0.00689476],
       [ 0.0616533 ],
       [-0.00668752],
       [-0.02735964],
       [ 0.0377918 ],
       [-0.02855931],
       [ 0.05299238],
       [-0.0121608 ],
       [ 0.00138755],
       [ 0.0905595 ],
       [ 0.19994621],
       [ 0.2556632 ],
       [ 0.41660047]], dtype=float32), array([0.01430958], dtype=float32)]
```

W_{t0}

W_{t1}

W_{t2}

W_{t3}

$$Y = W_{t0}X_0 + W_{t1}X_1 + W_{t2}X_2 + \ldots\ldots + W_{t19}X_{19} + b$$

b

```
print(series[1:21])
model.predict(series[1:21][np.newaxis])
```

```
print(series[1:21])
model.predict(series[1:21][np.newaxis])
```

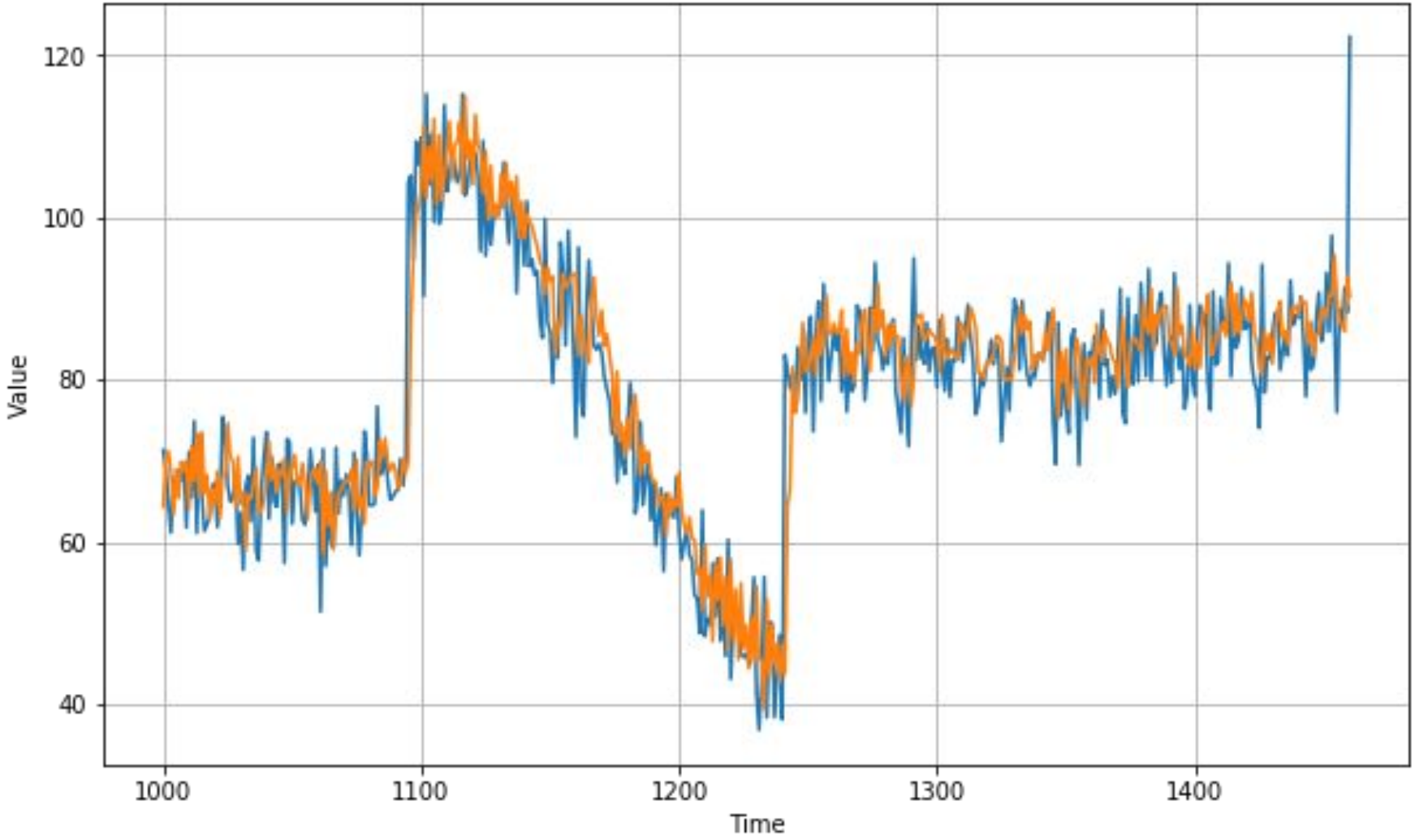
```
print(series[1:21])
model.predict(series[1:21][np.newaxis])

[49.35275  53.314735 57.711823 48.934444 48.931244 57.982895 53.897125
 47.67393  52.68371  47.591717 47.506374 50.959415 40.086178 40.919415
 46.612473 44.228207 50.720642 44.454983 41.76799  55.980938]

array([[49.08478]], dtype=float32)
```

```
forecast = []
for time in range(len(series) - window_size):
  forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]
```

```
forecast = []
for time in range(len(series) - window_size):
  forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]
```

```
forecast = []
for time in range(len(series) - window_size):
  forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]
```

```
forecast = []
for time in range(len(series) - window_size):
  forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]
```


120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
```

```
4.9526777
```

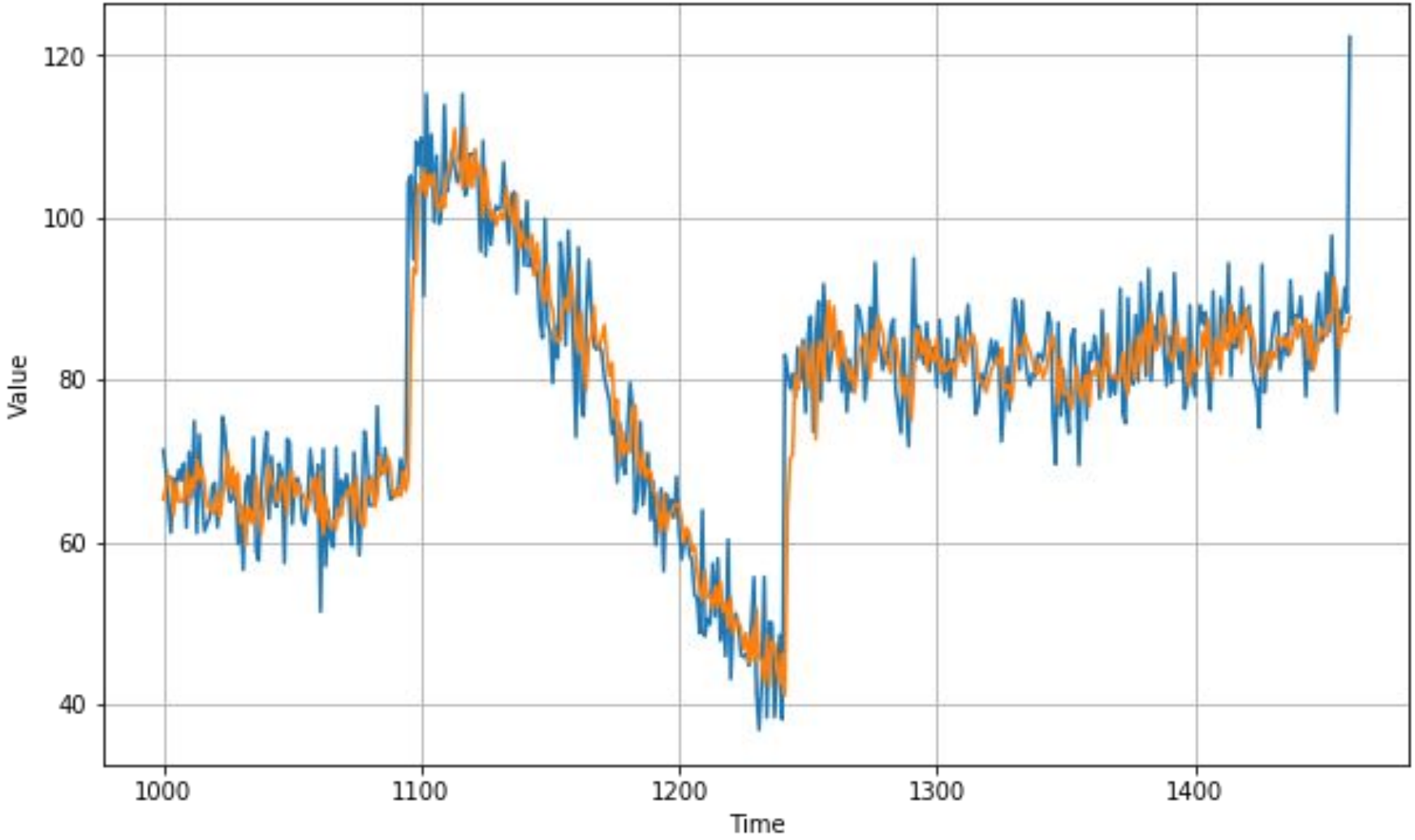
```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

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


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

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


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

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


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

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


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```


120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
```

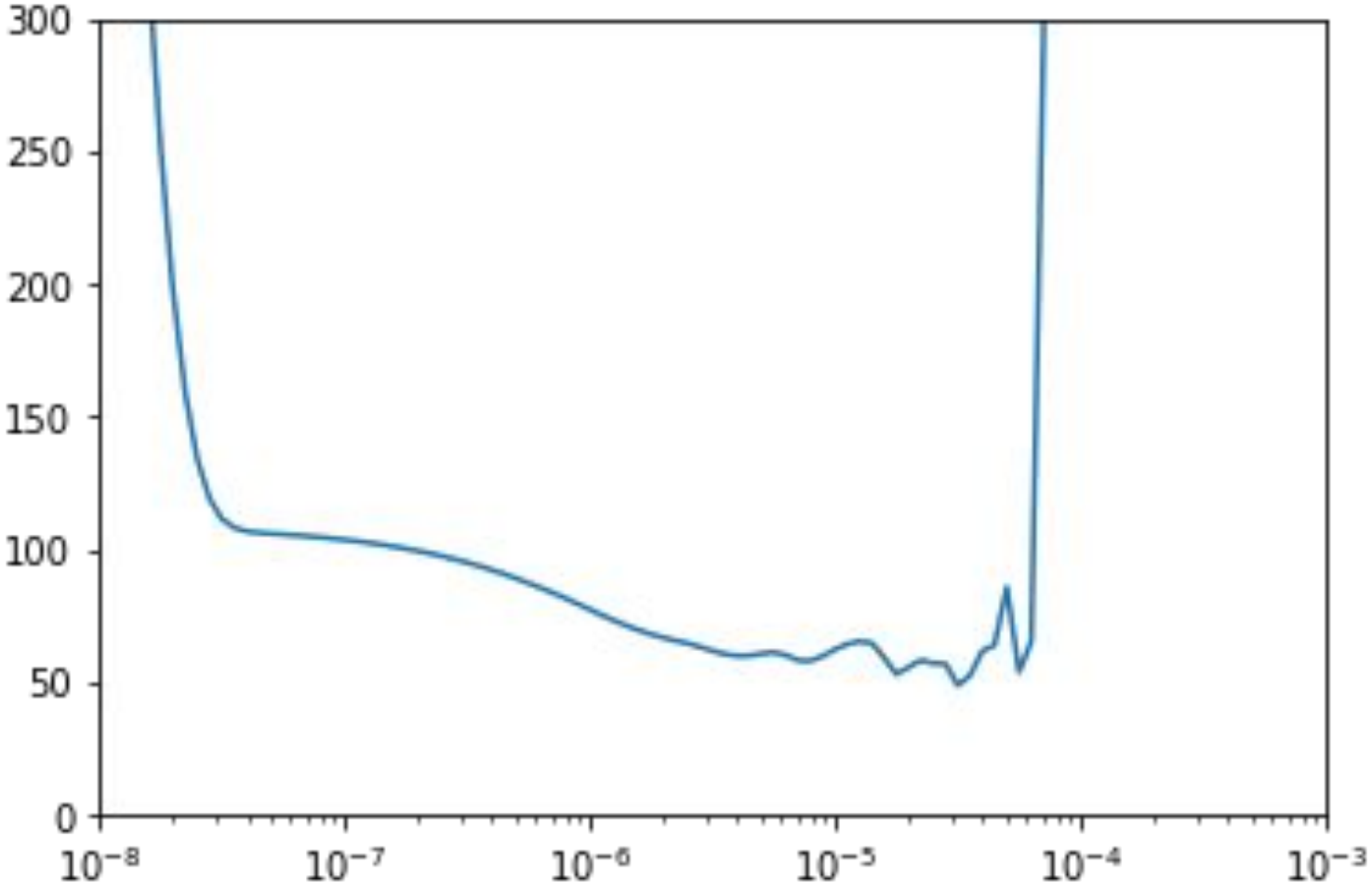
```
4.9833784
```

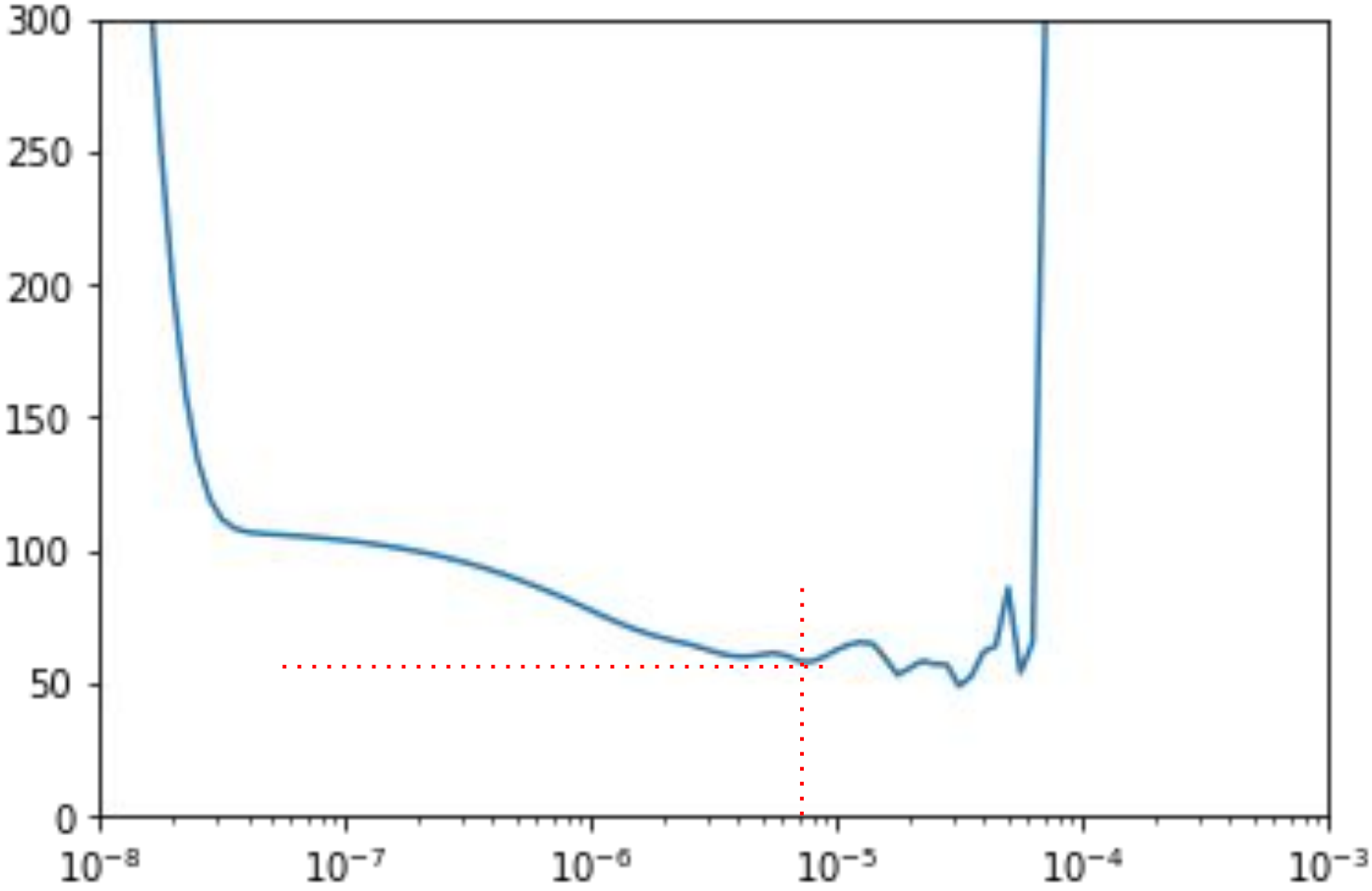
```
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, input_shape=[window_size], 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))

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

model.compile(loss="mse", optimizer=optimizer)

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

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


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, input_shape=[window_size], 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))

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

model.compile(loss="mse", optimizer=optimizer)

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

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


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, input_shape=[window_size], 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))

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

model.compile(loss="mse", optimizer=optimizer)

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

```
lrs = 1e-8 * (10 ** (np.arange(100) / 20))
plt.semilogx(lrs, history.history["loss"])
plt.axis([1e-8, 1e-3, 0, 300])
```

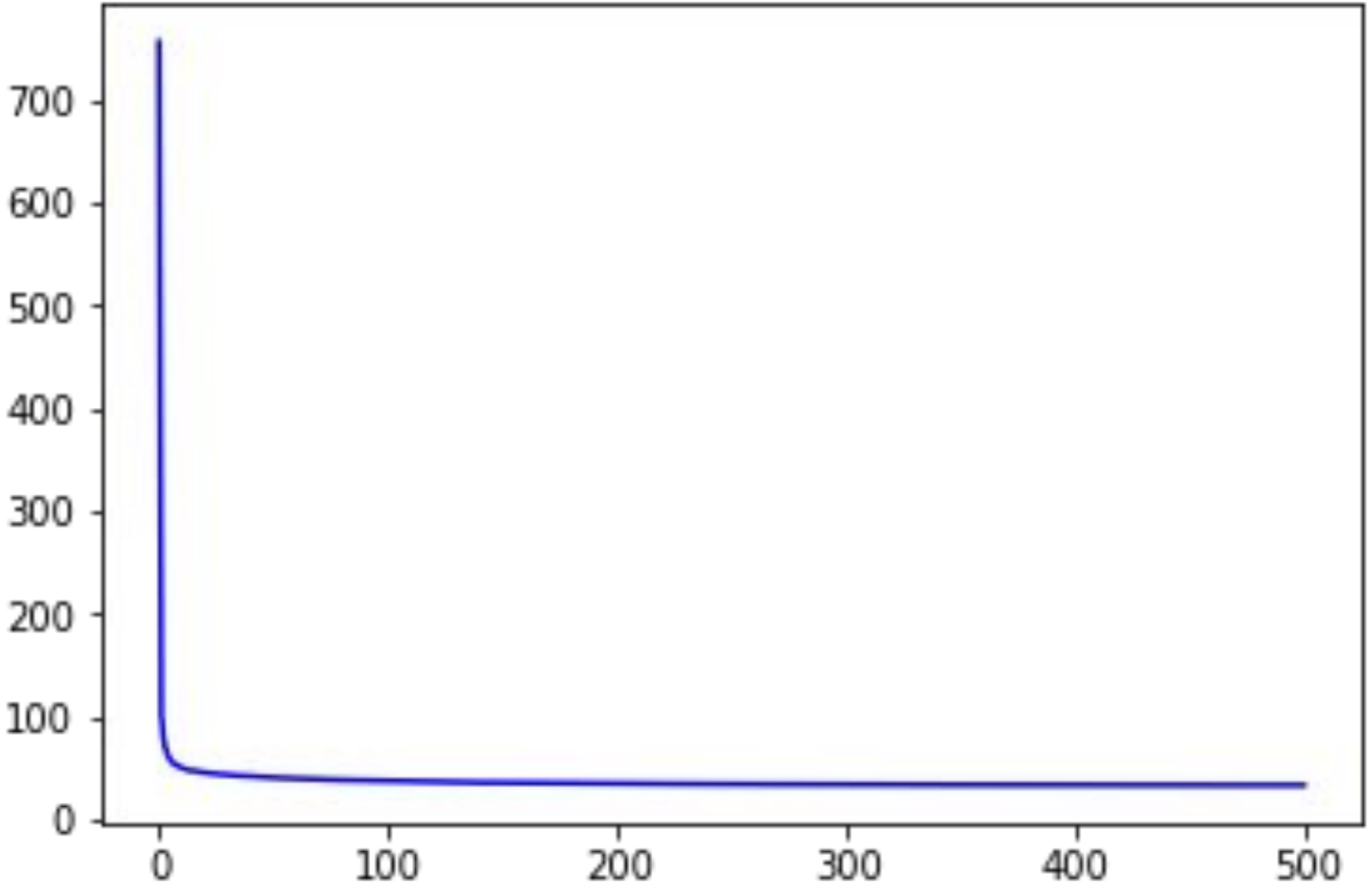

300
250
200
150
100
50
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}
10^{-3}


300
250
200
150
100
50
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}
10^{-3}

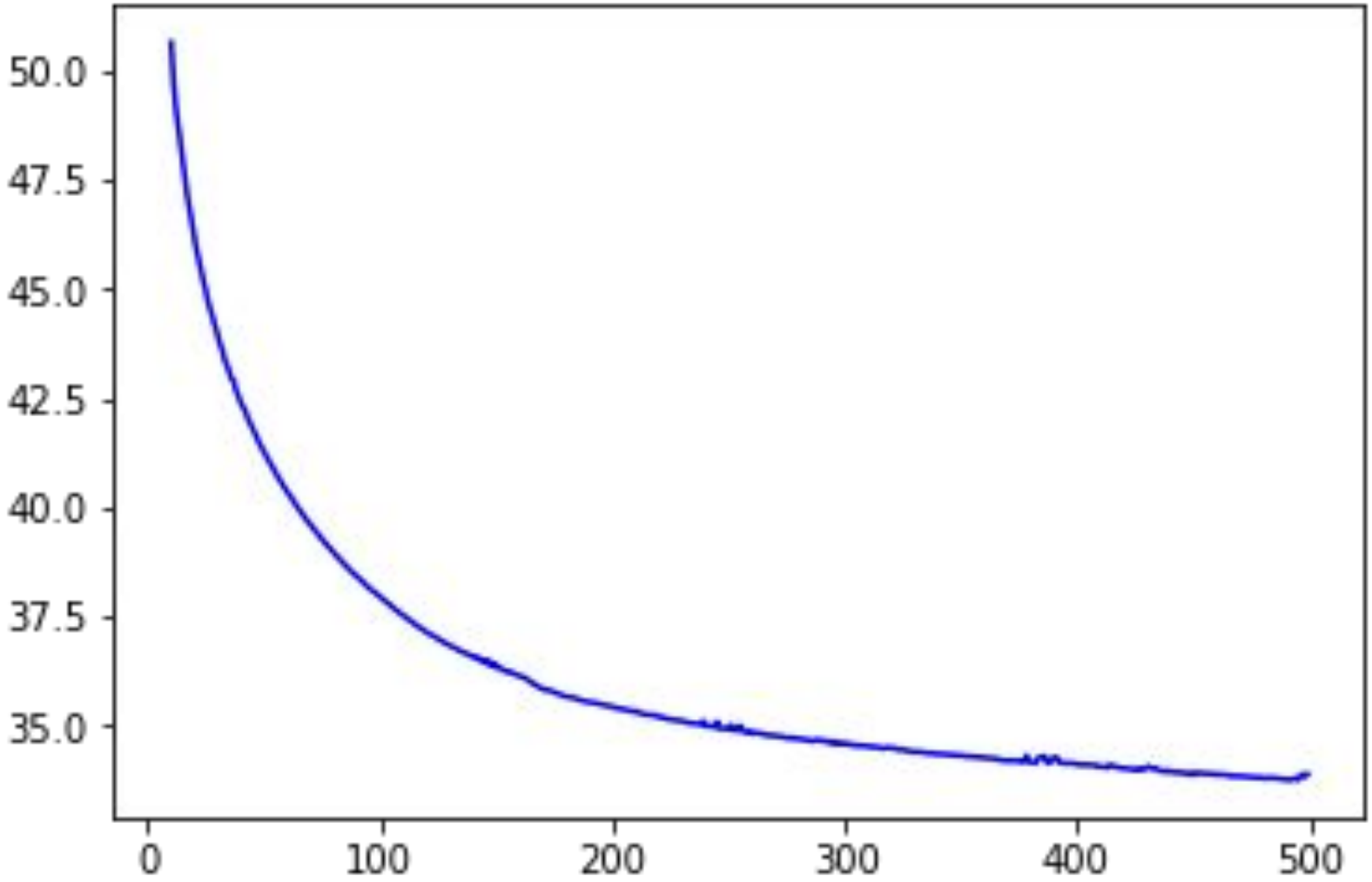
```
window_size = 30
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

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

optimizer = tf.keras.optimizers.SGD(learning_rate=7e-6, momentum=0.9)
model.compile(loss="mse", optimizer=optimizer)
history = model.fit(dataset, epochs=500)
```

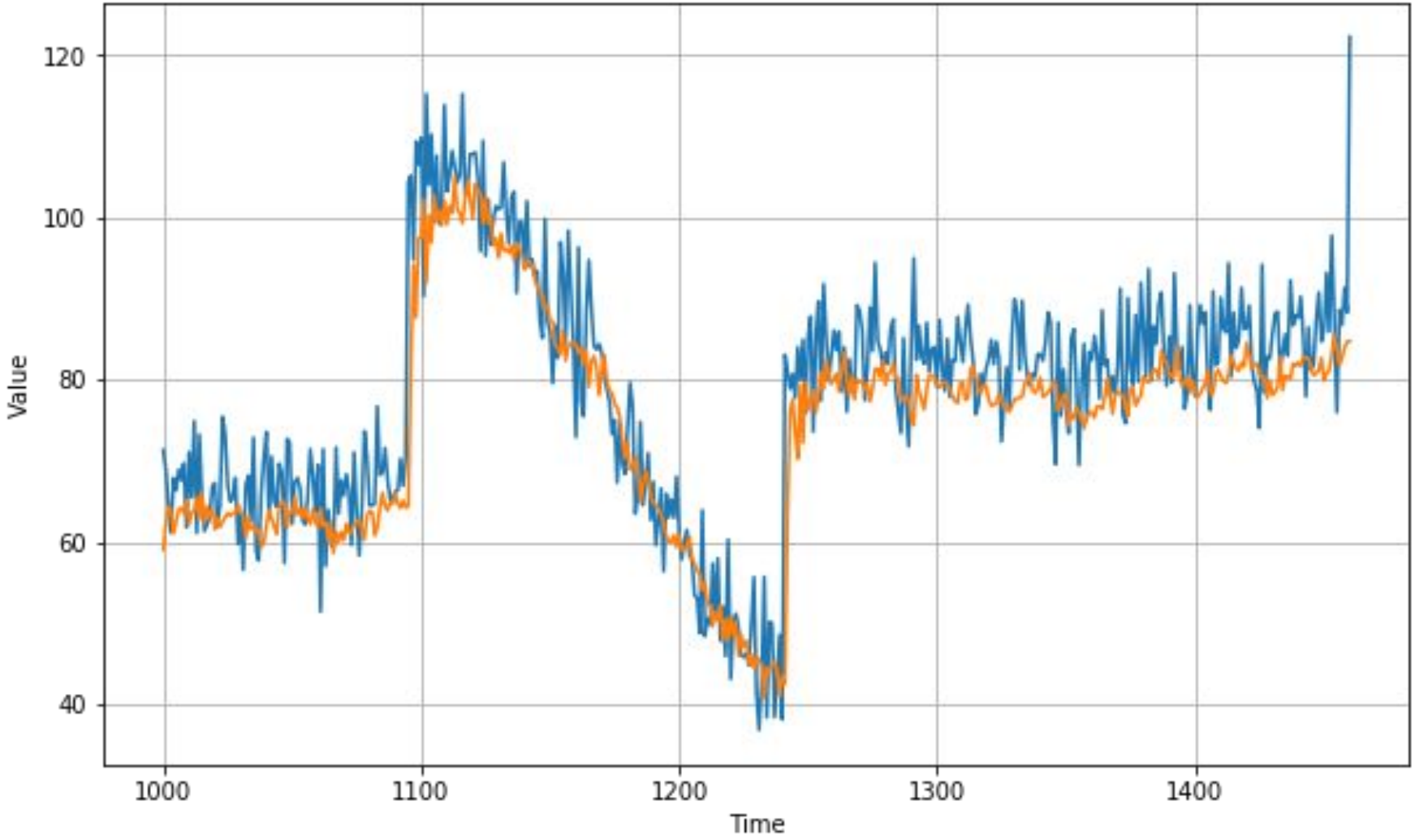
```
window_size = 30
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

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

optimizer = tf.keras.optimizers.SGD(learning_rate=7e-6, momentum=0.9)
model.compile(loss="mse", optimizer=optimizer)
history = model.fit(dataset, epochs=500)
```

```
# Plot the loss
loss = history.history['loss']
epochs = range(len(loss))
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.show()
```


700
600
500
400
300
200
100
0
0
100
200
300
400
500

```
# Plot all but the first 10
loss = history.history['loss']
epochs = range(10, len(loss))
plot_loss = loss[10:]
print(plot_loss)
plt.plot(epochs, plot_loss, 'b', label='Training Loss')
plt.show()
```


50.0
47.5
45.0
42.5
40.0
37.5
35.0
0
100
200
300
400
500

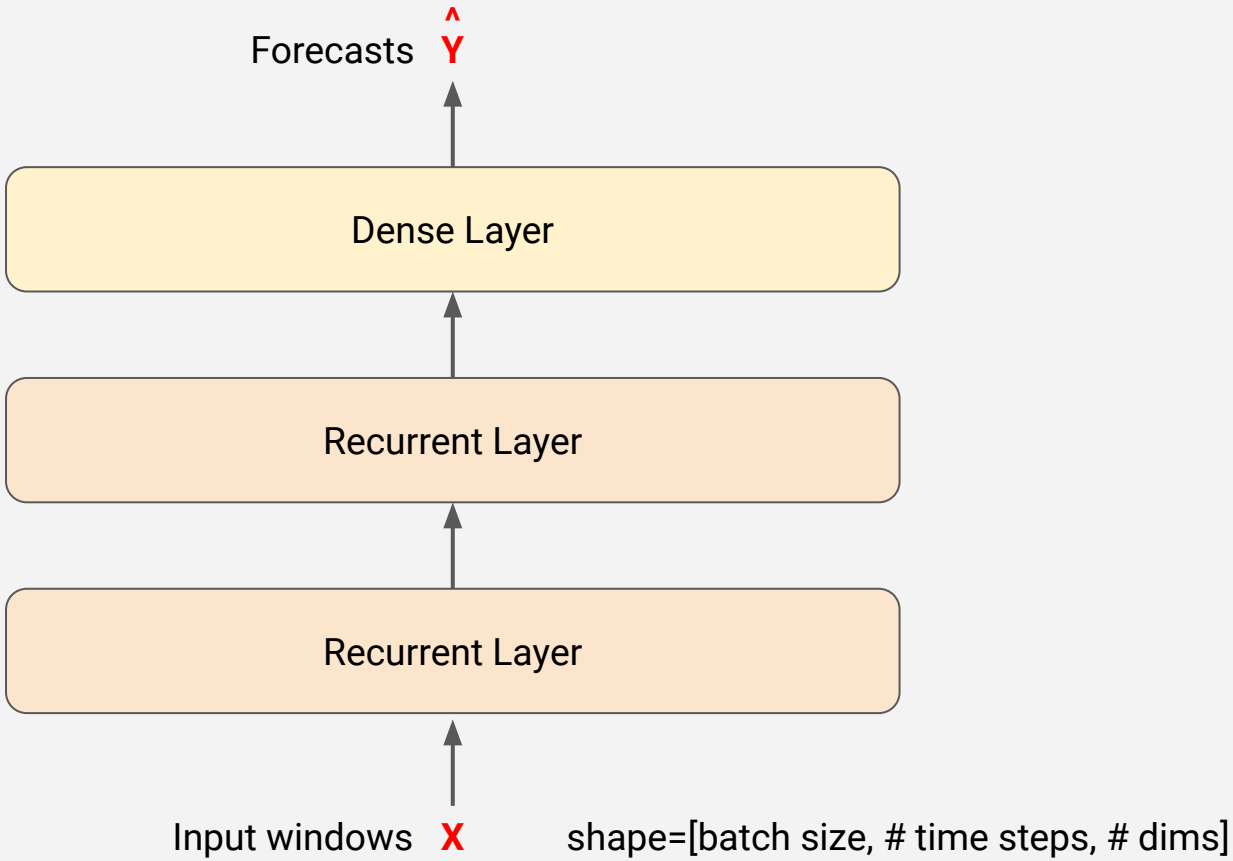

120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
```

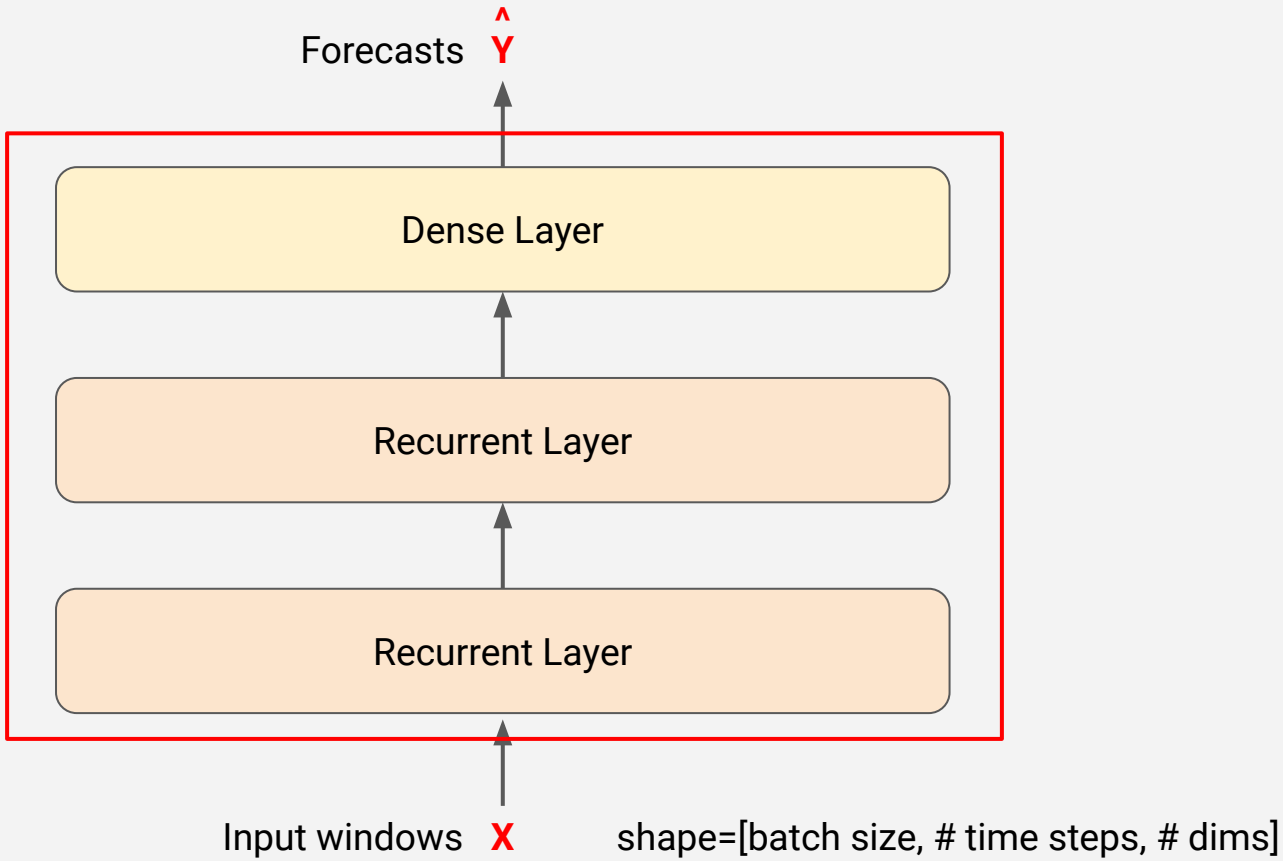
```
4.4847784
```

Copyright Notice

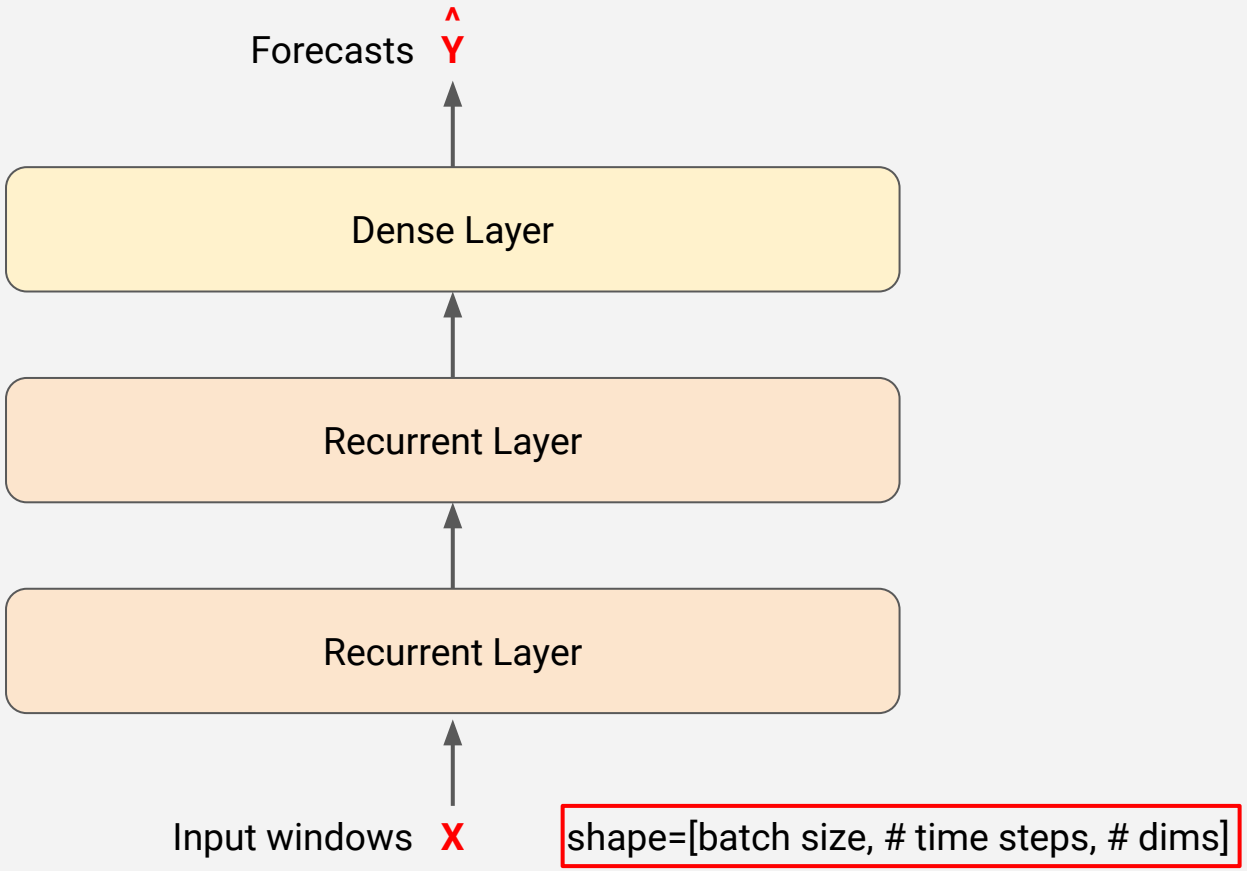
Recurrent Neural Network



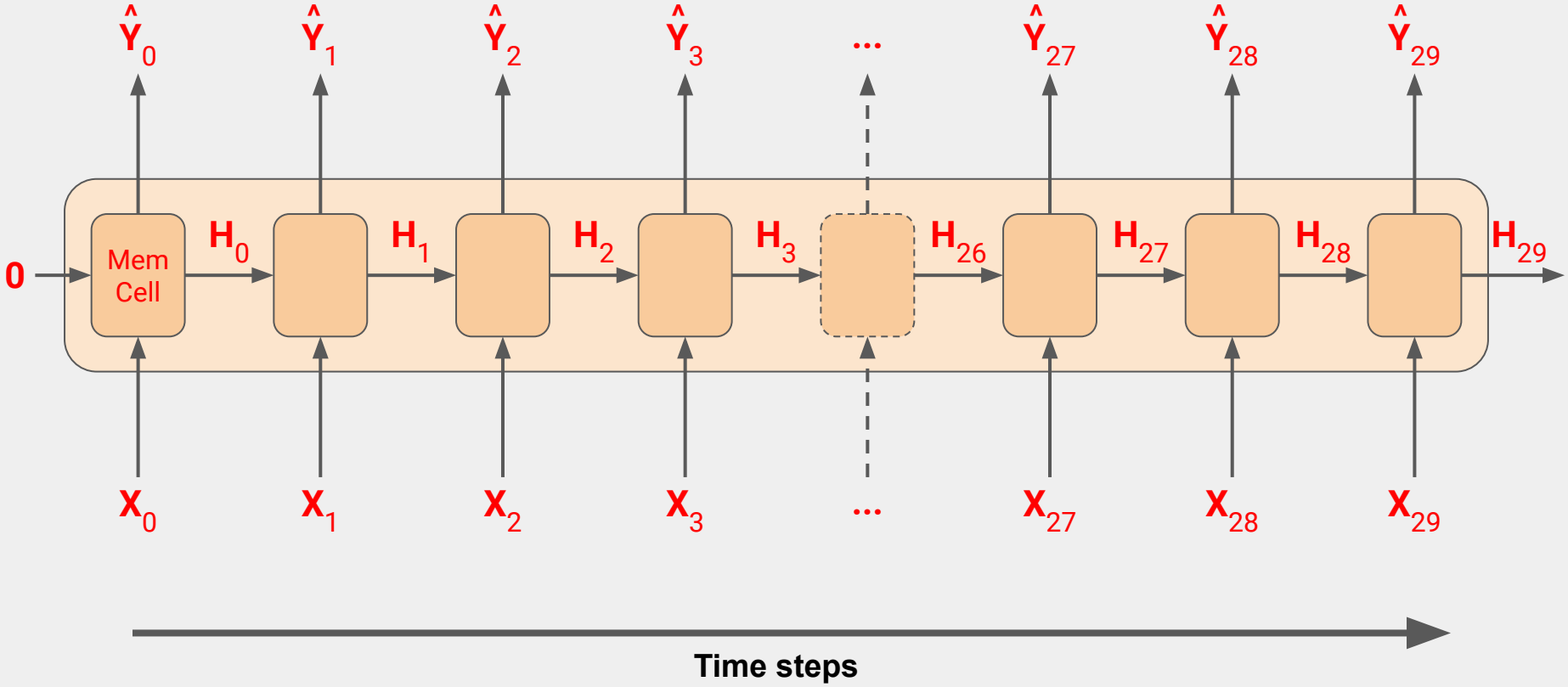
Recurrent Neural Network



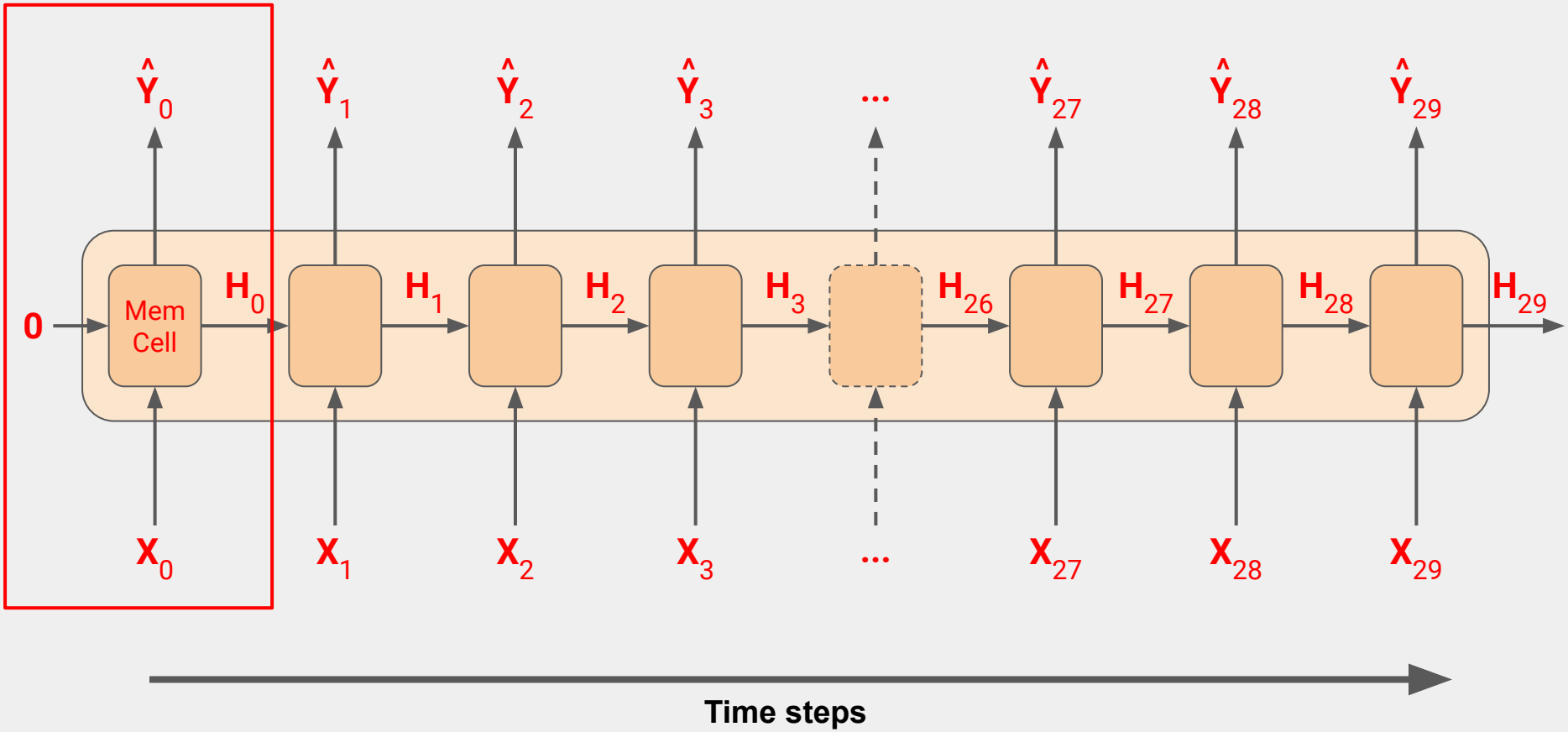
Recurrent Neural Network



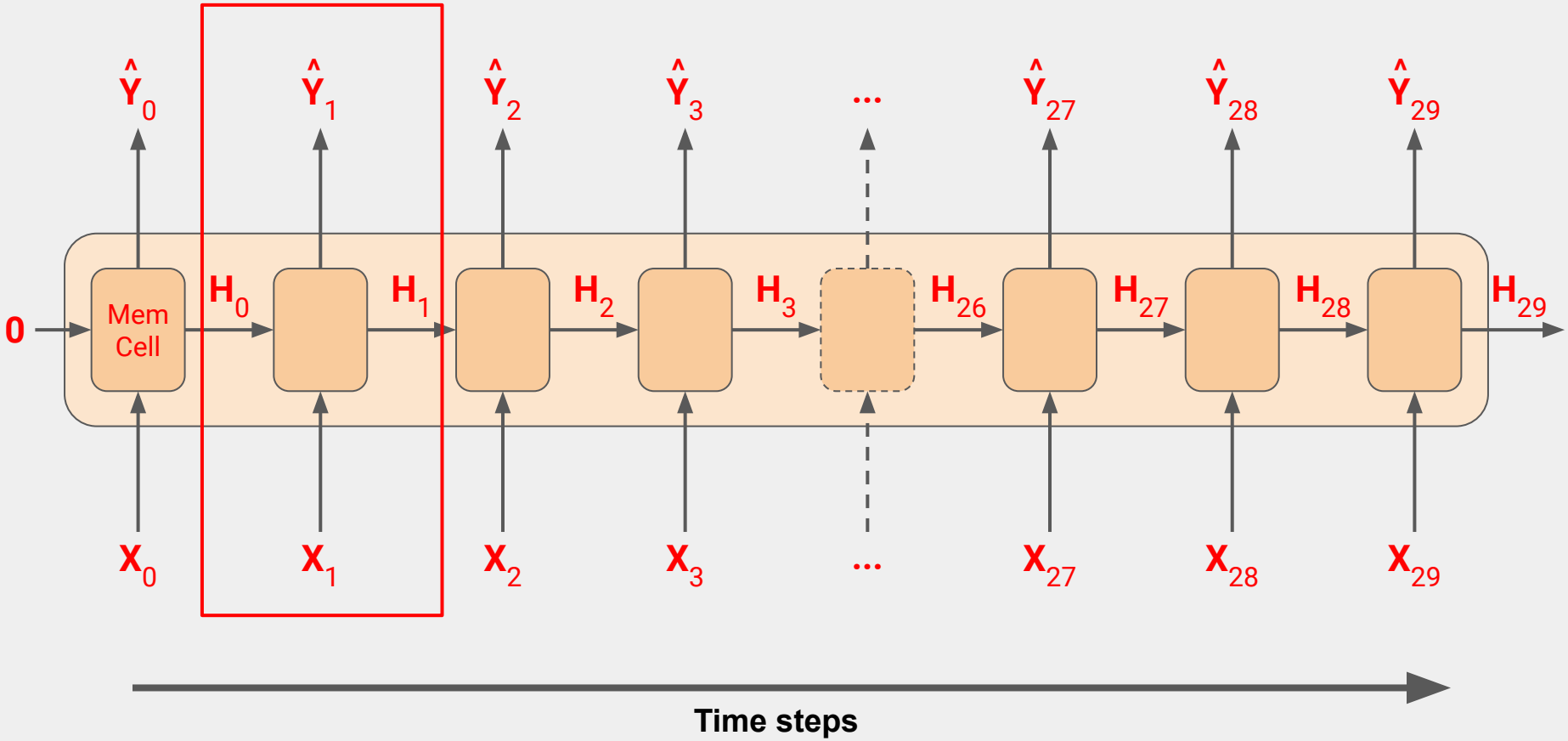
Recurrent Layer



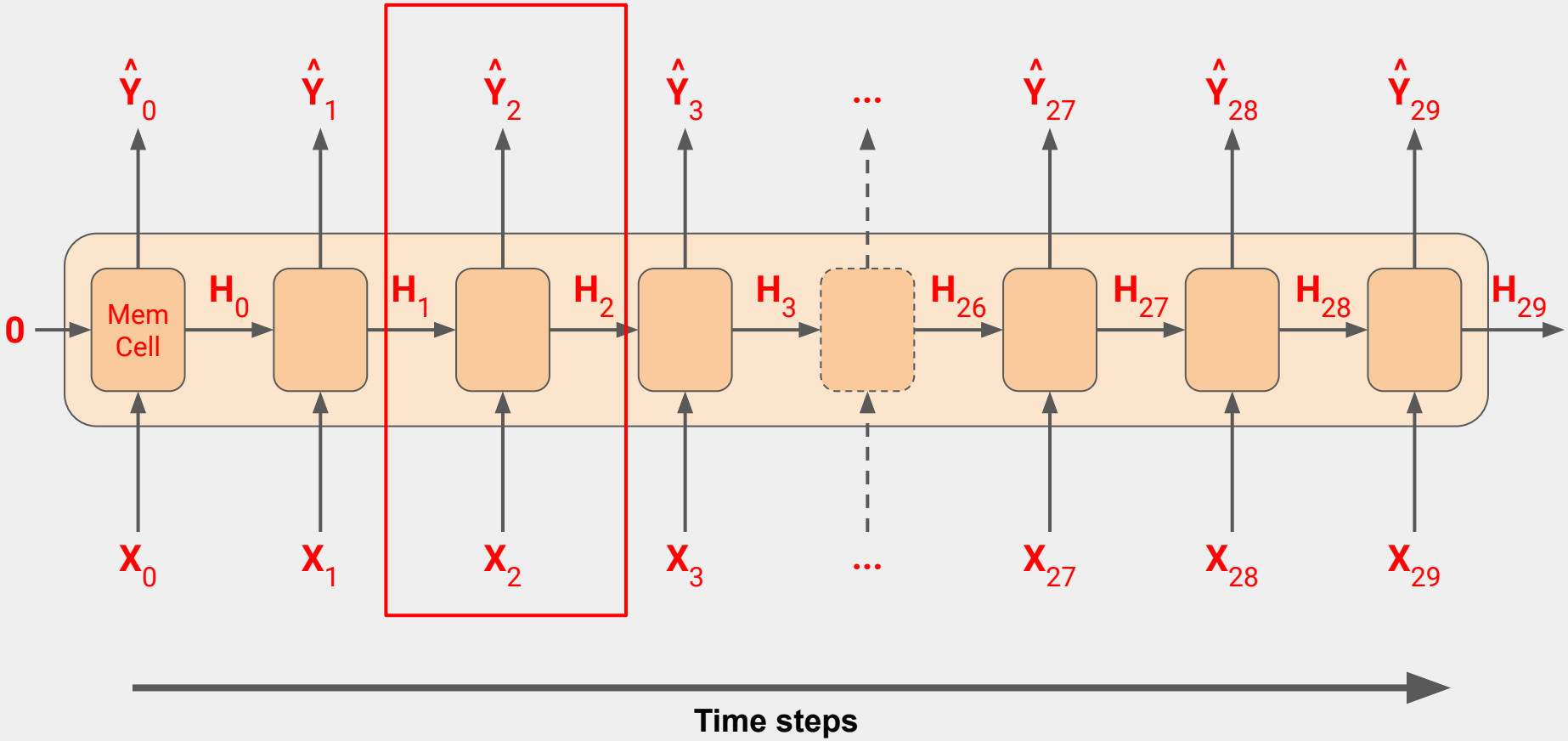
Recurrent Layer



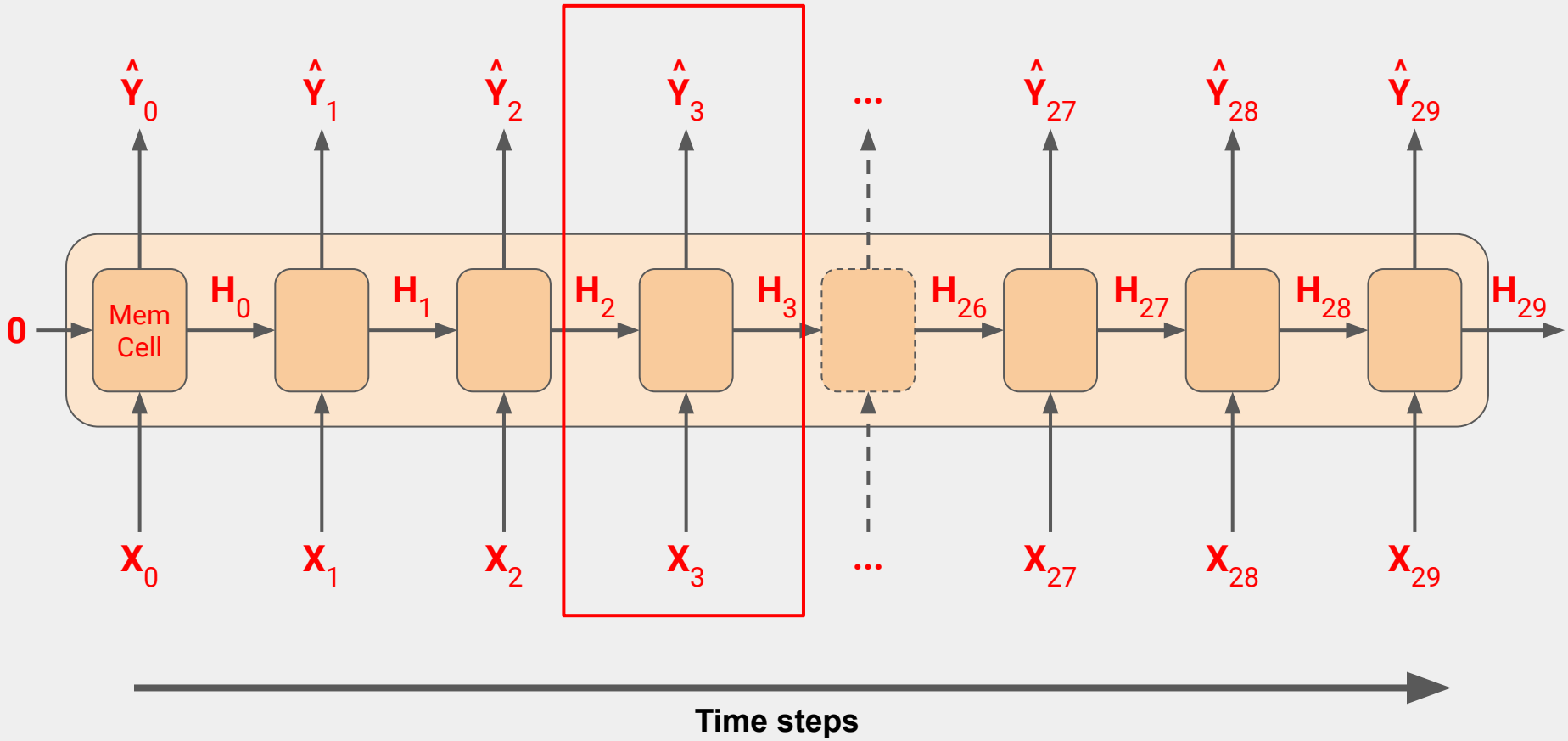
Recurrent Layer



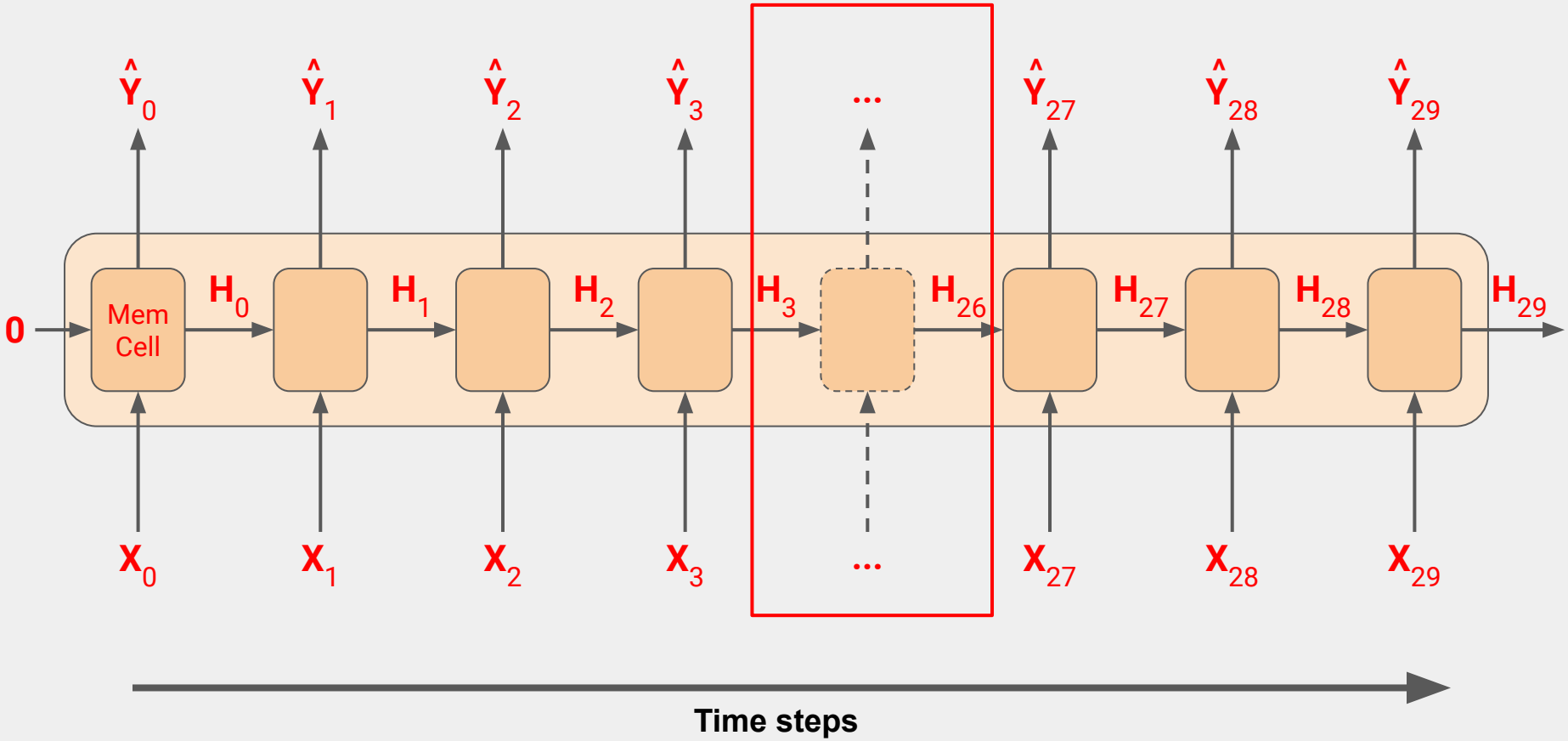
Recurrent Layer



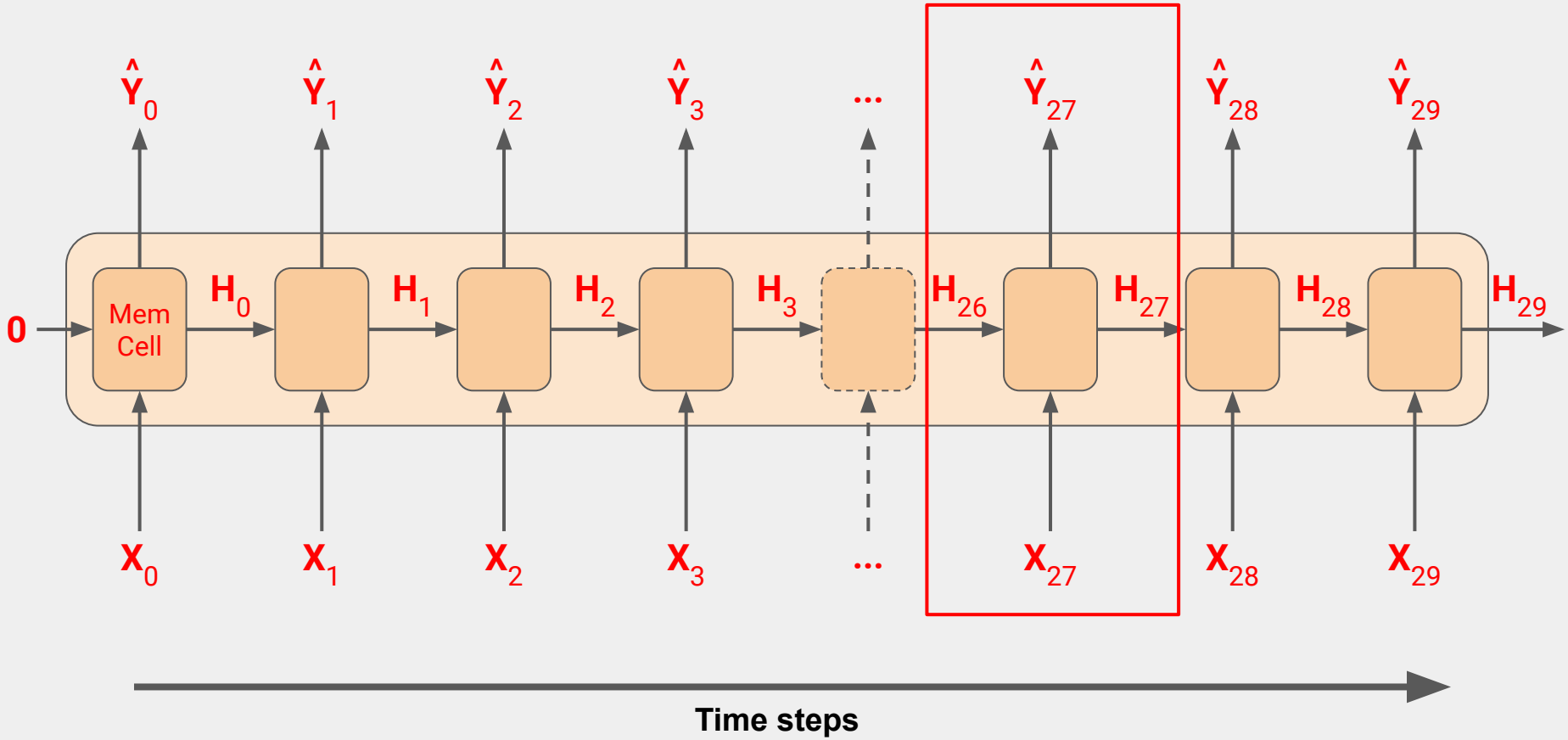
Recurrent Layer



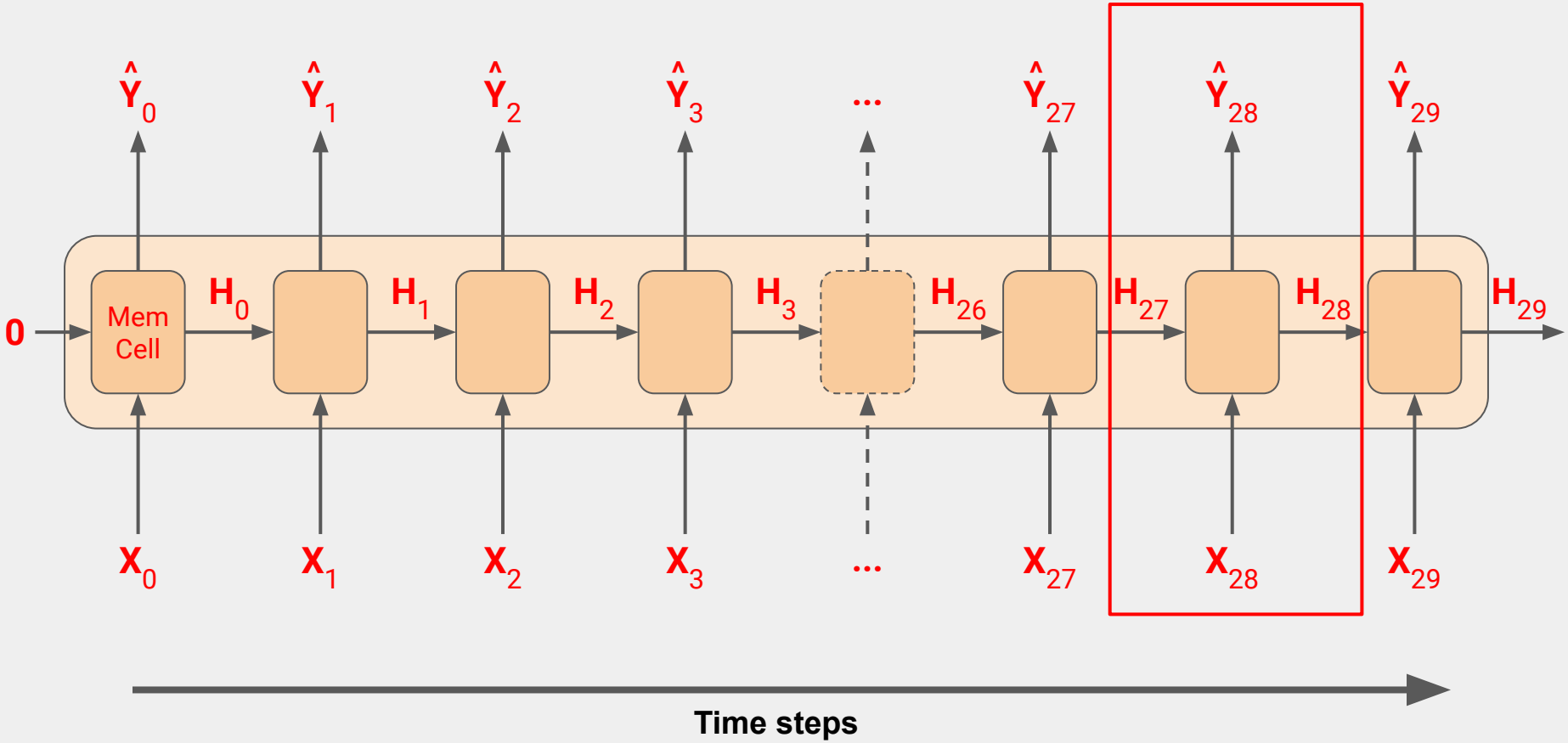
Recurrent Layer



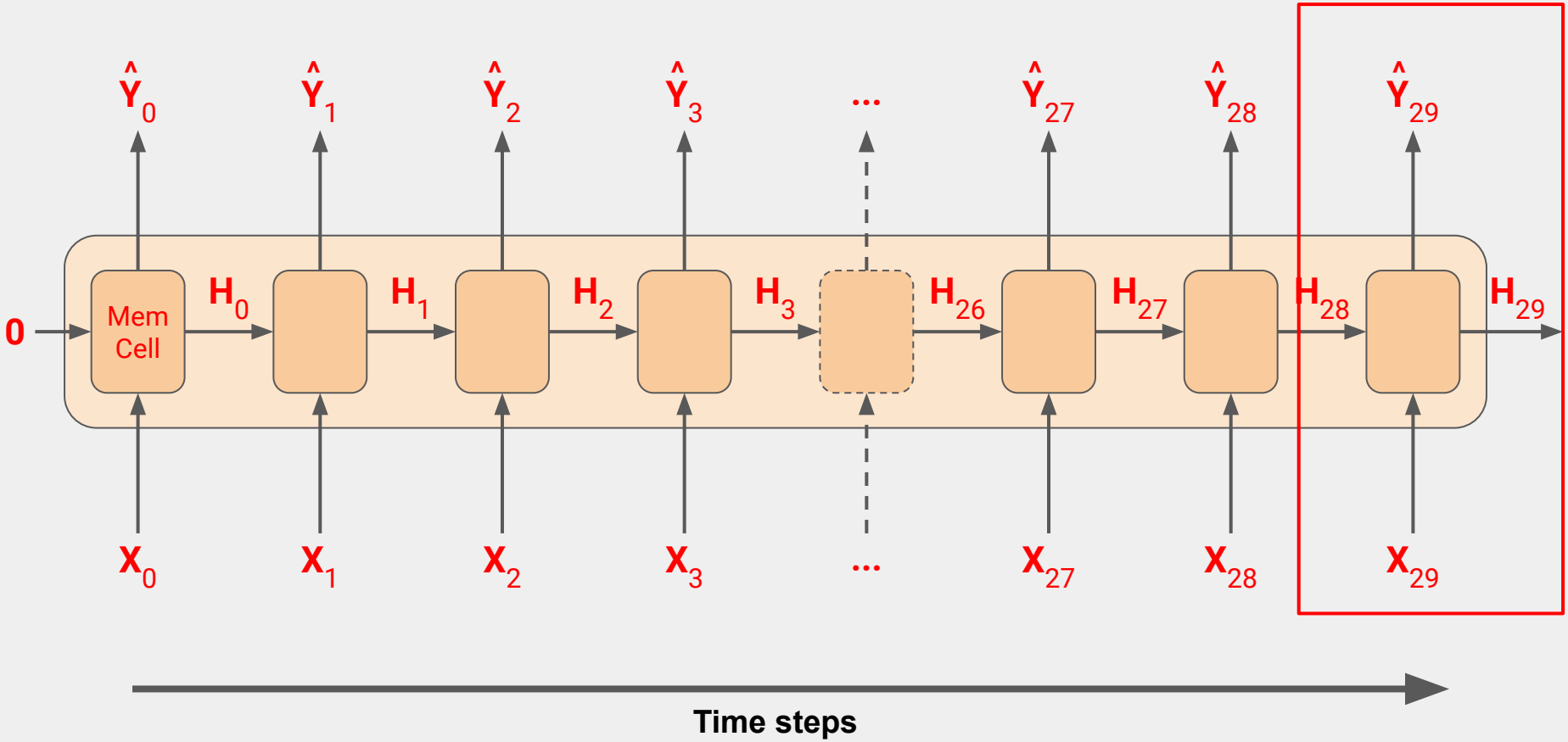
Recurrent Layer



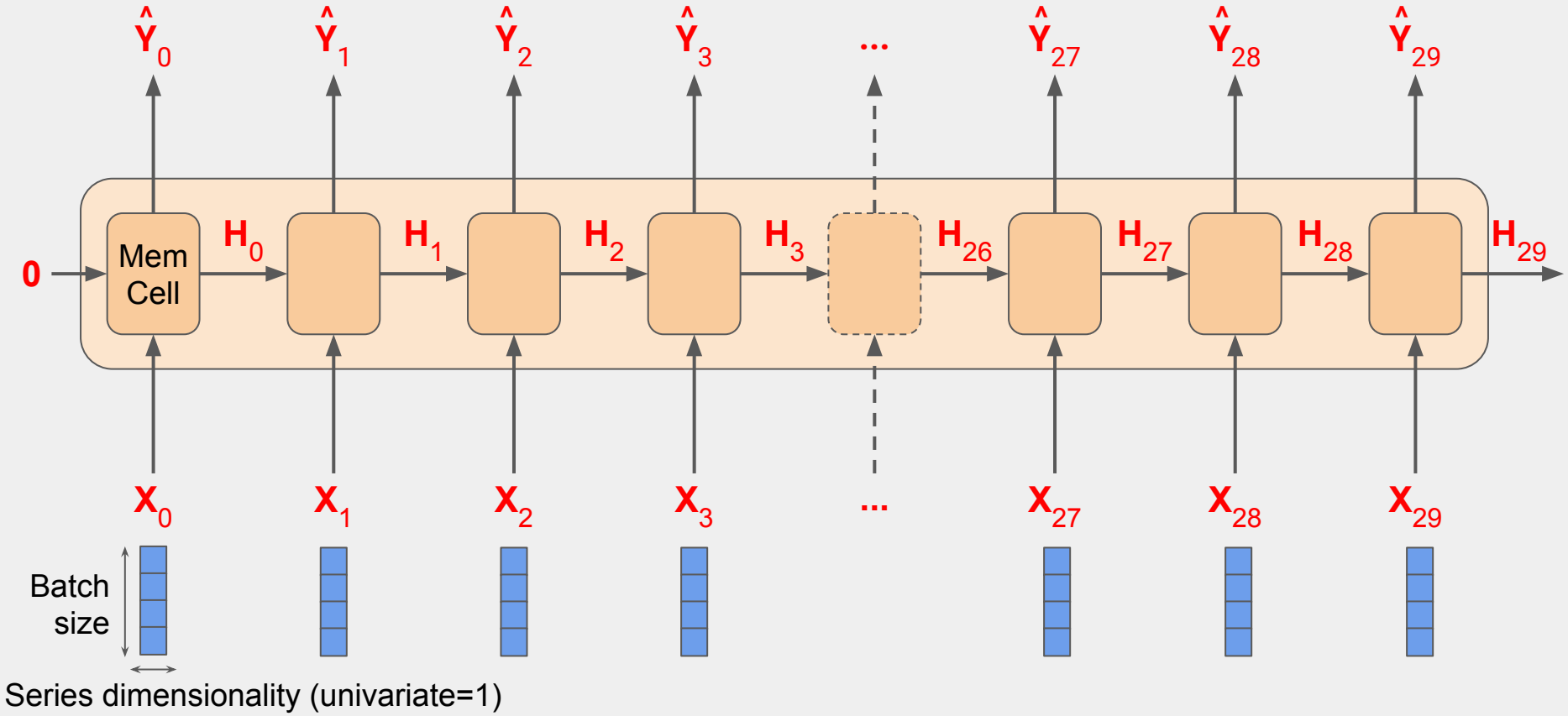
Recurrent Layer



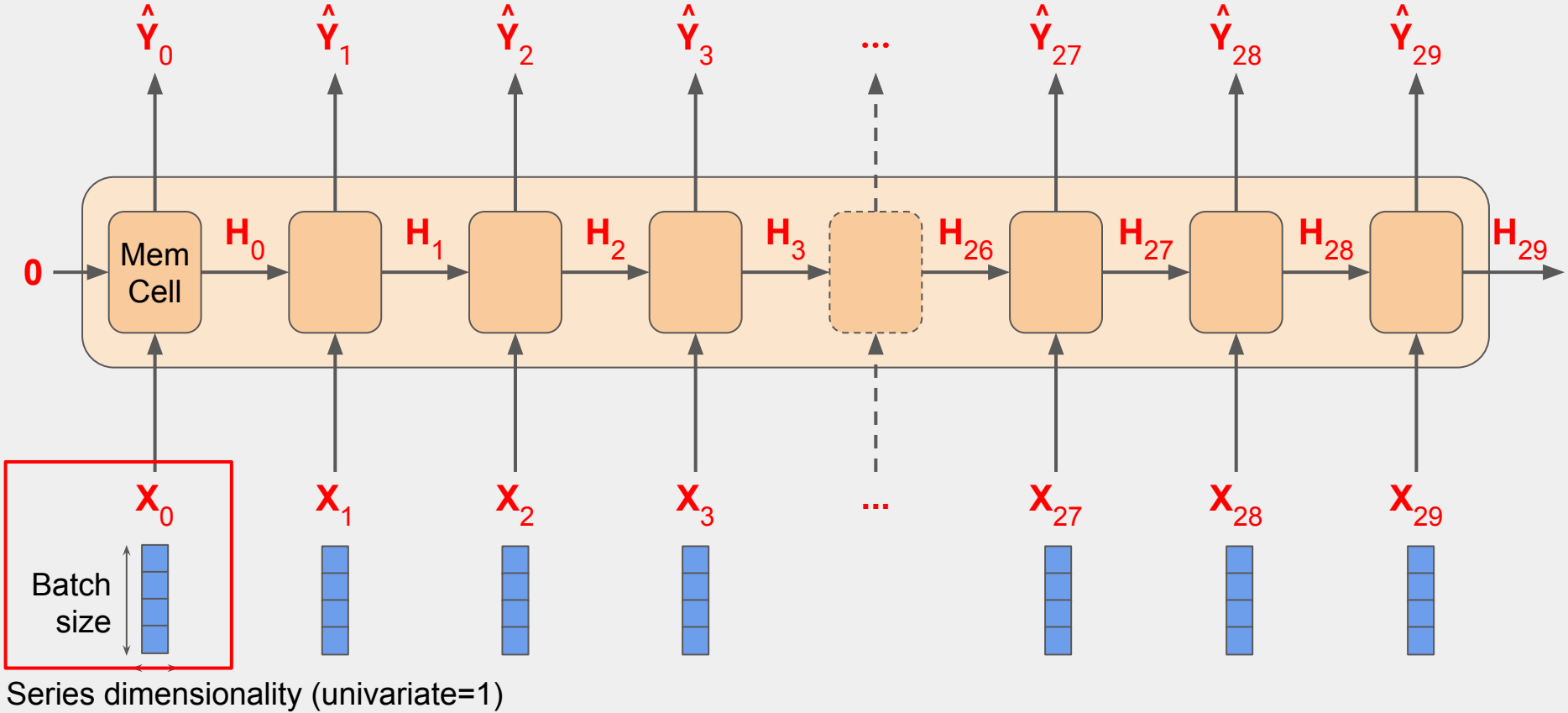
Recurrent Layer

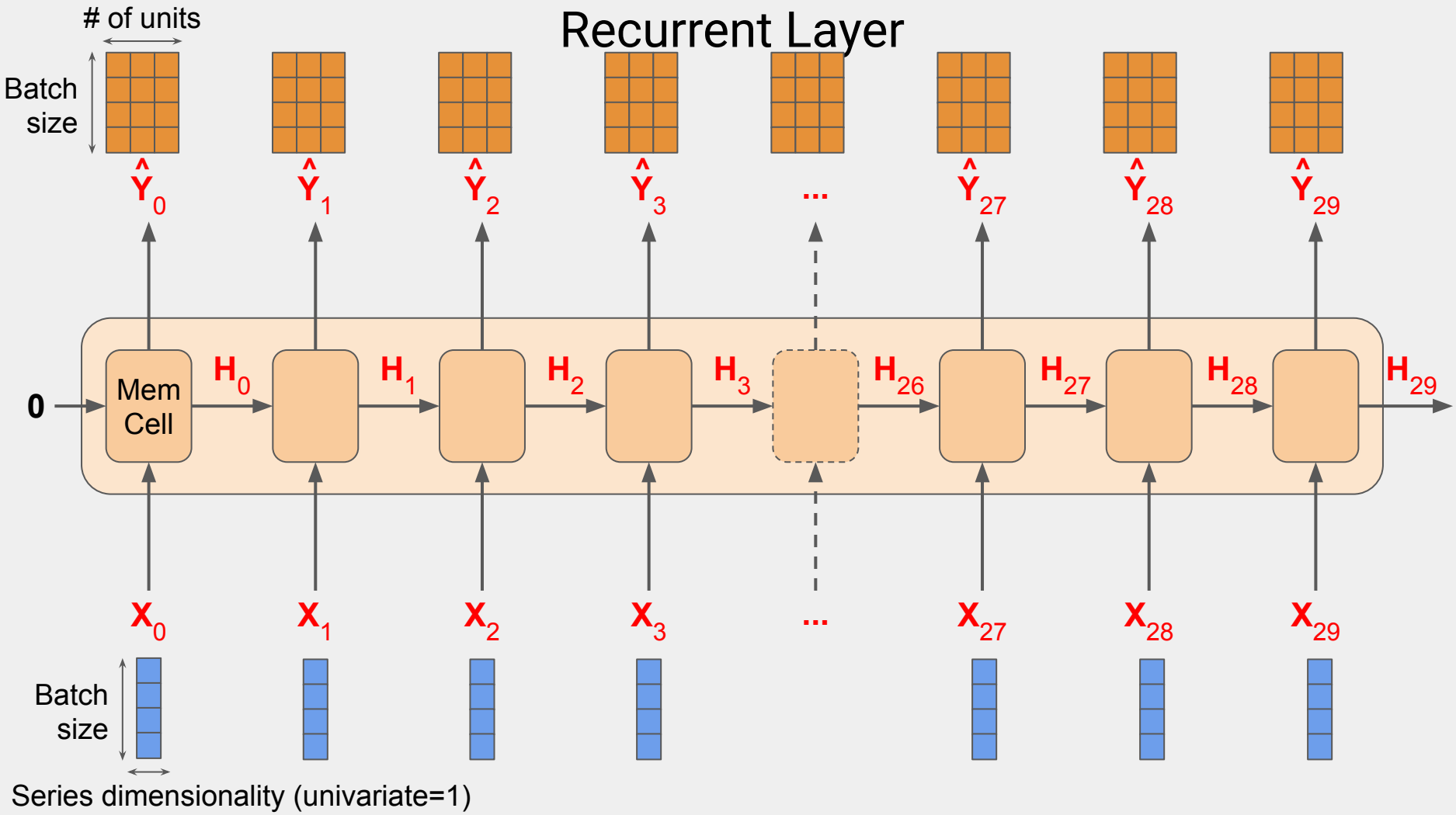


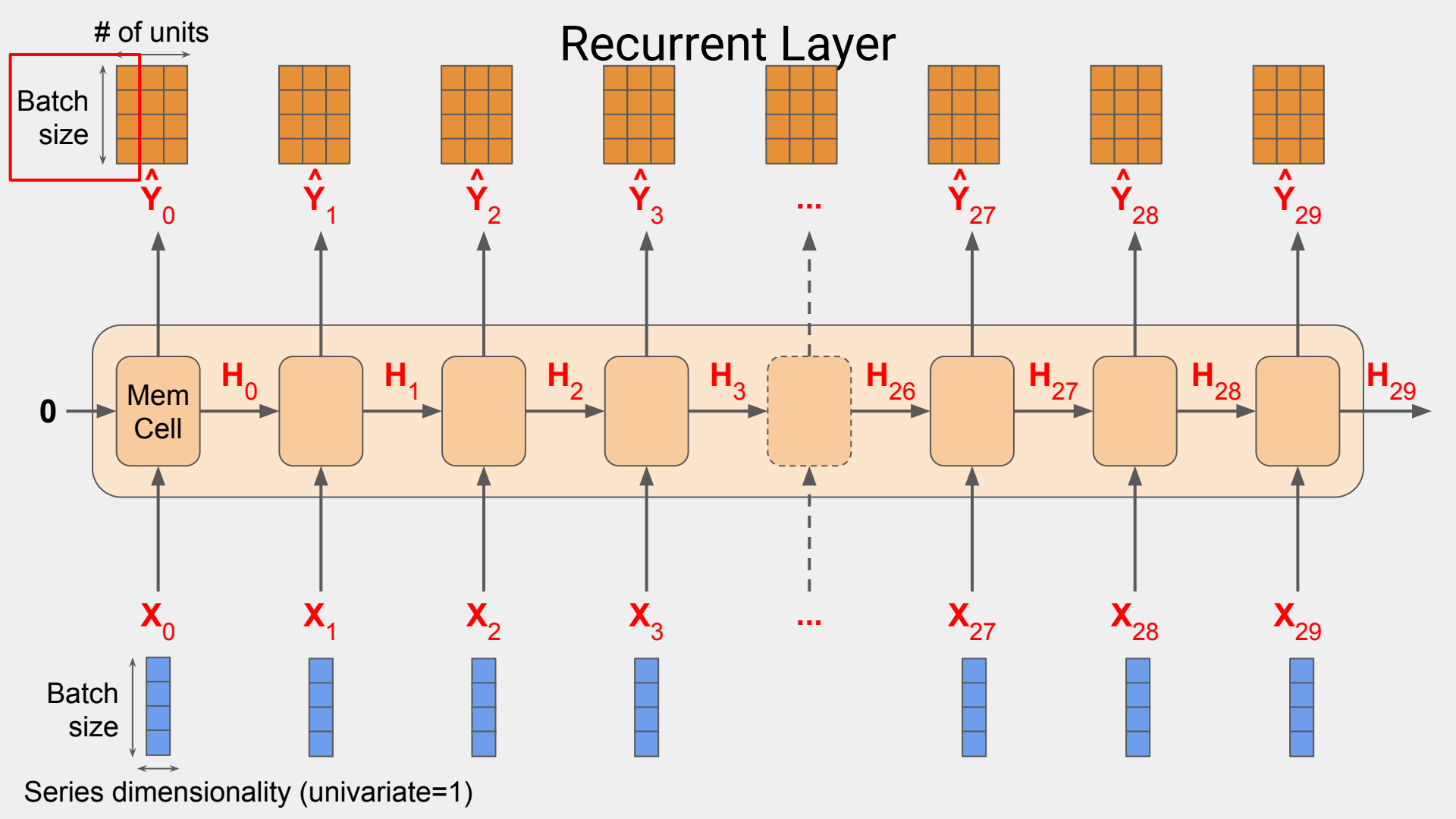
Recurrent Layer

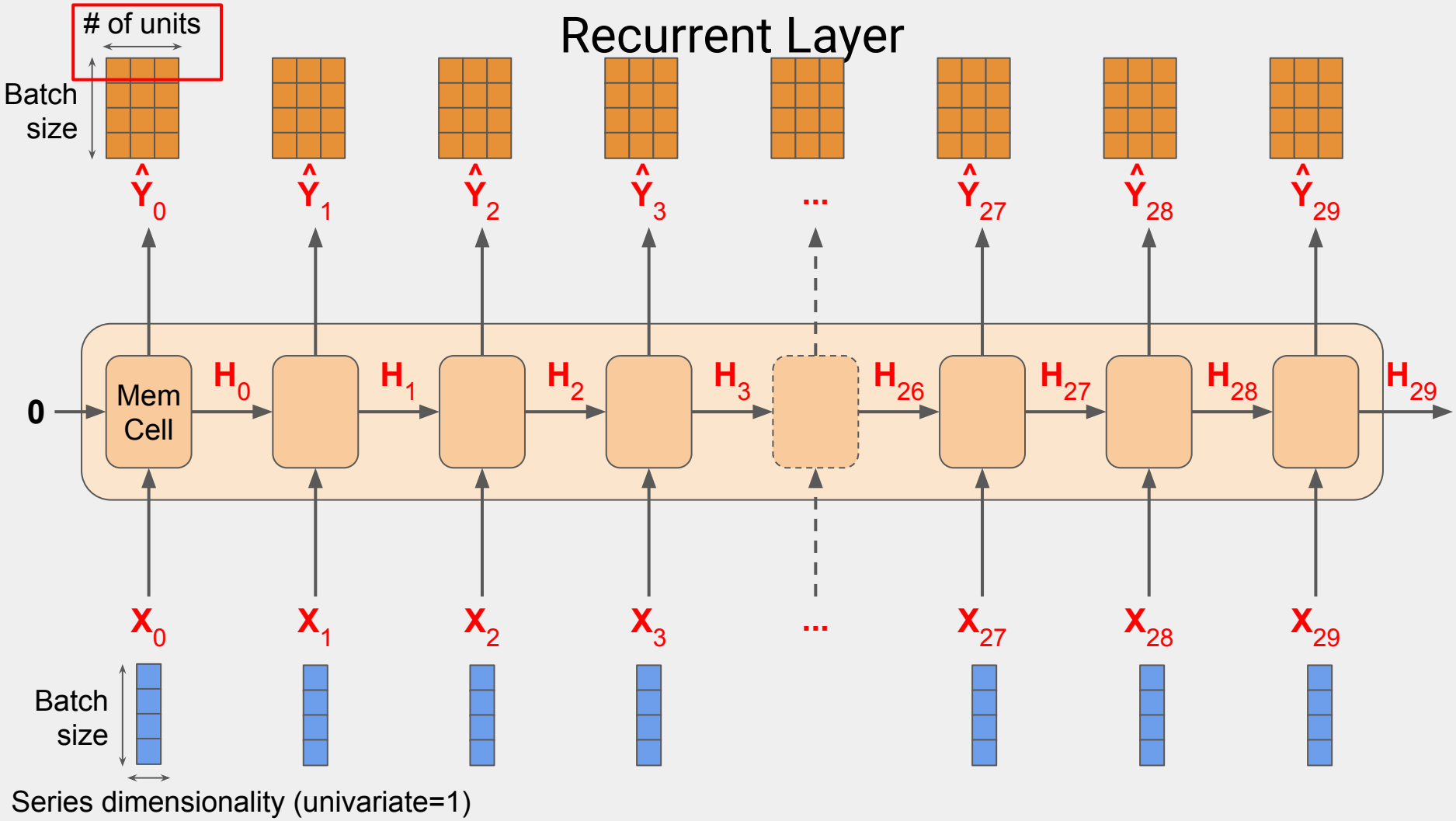


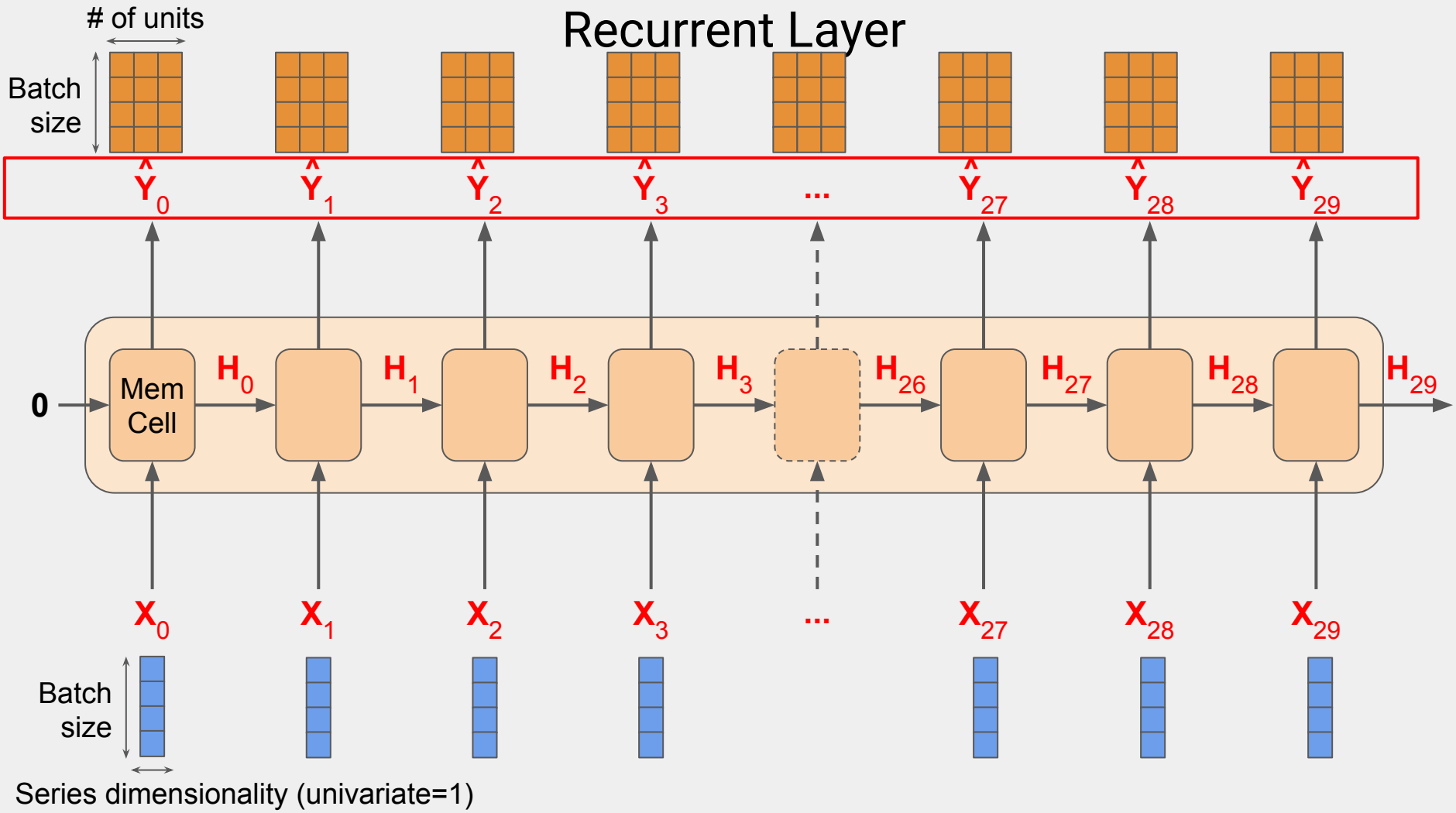
Recurrent Layer

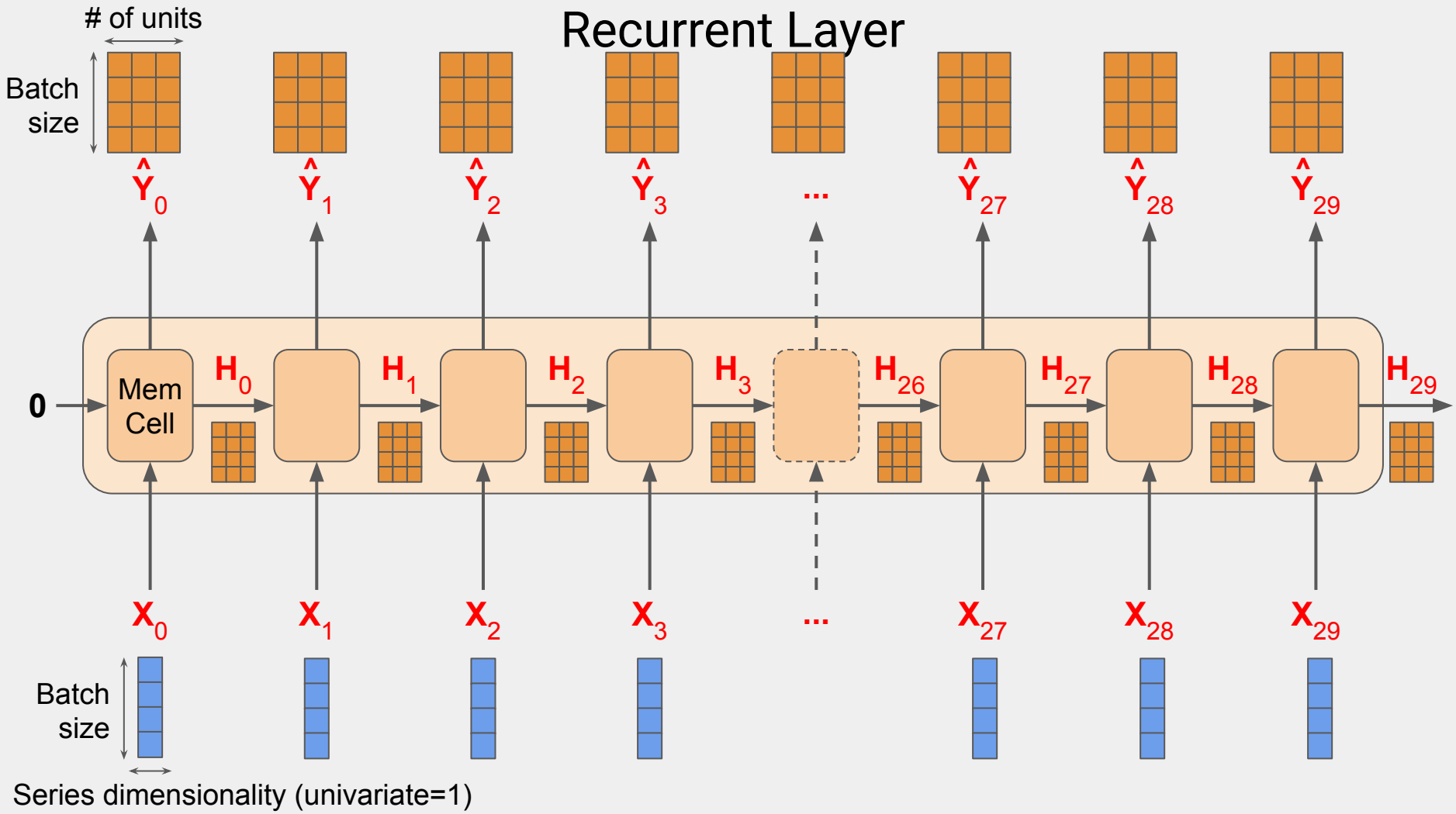


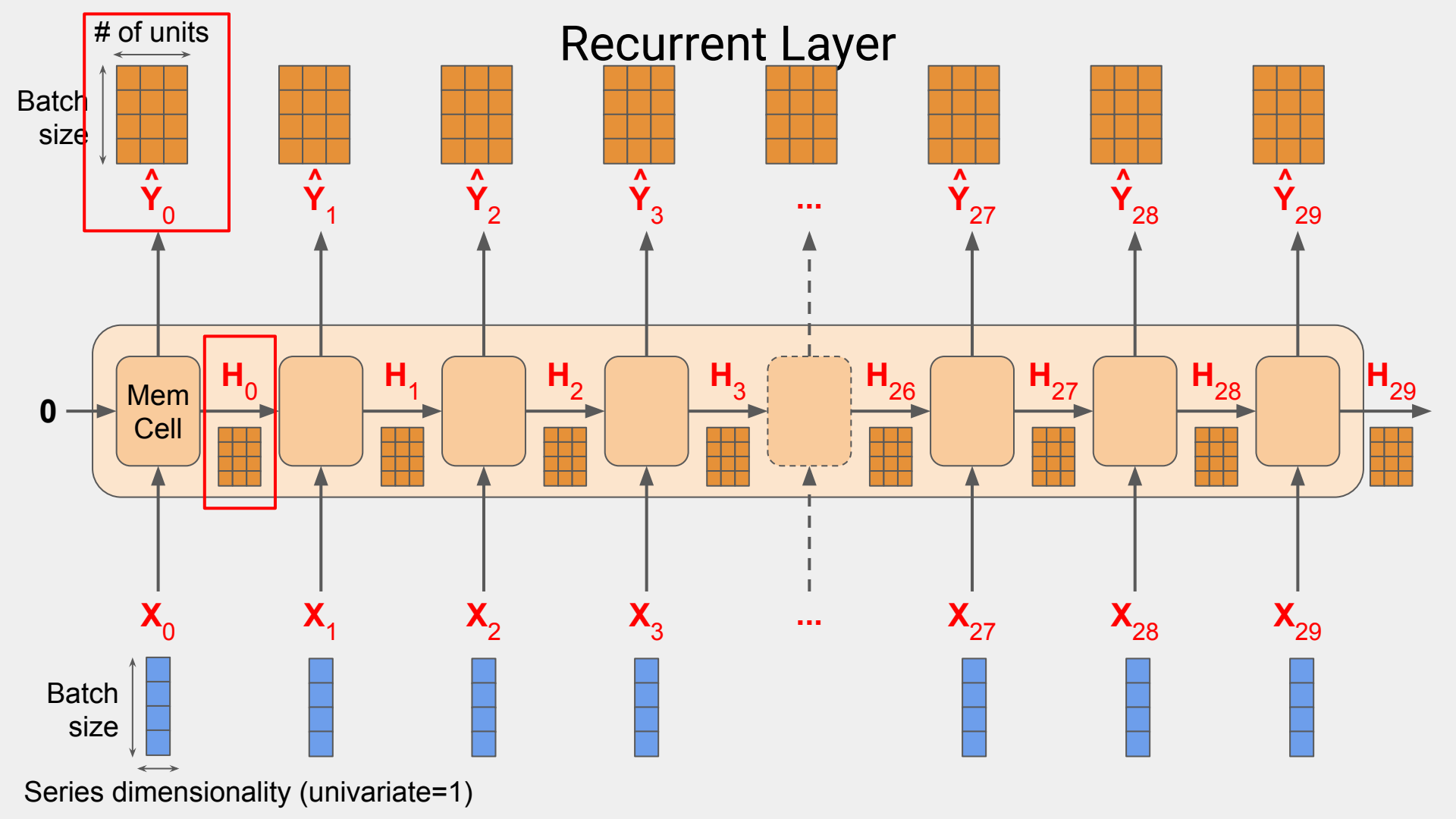

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)

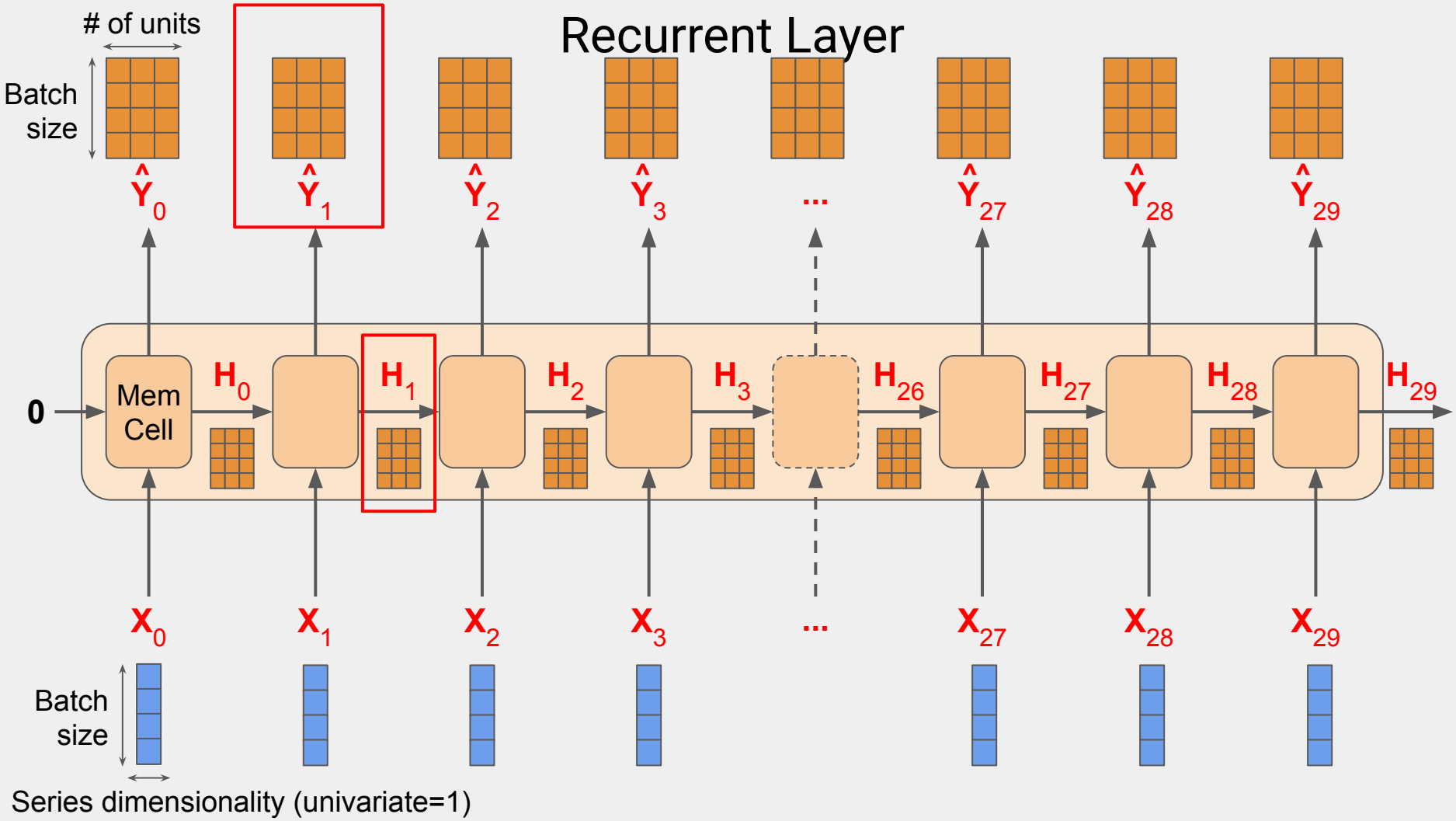

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)


Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
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)

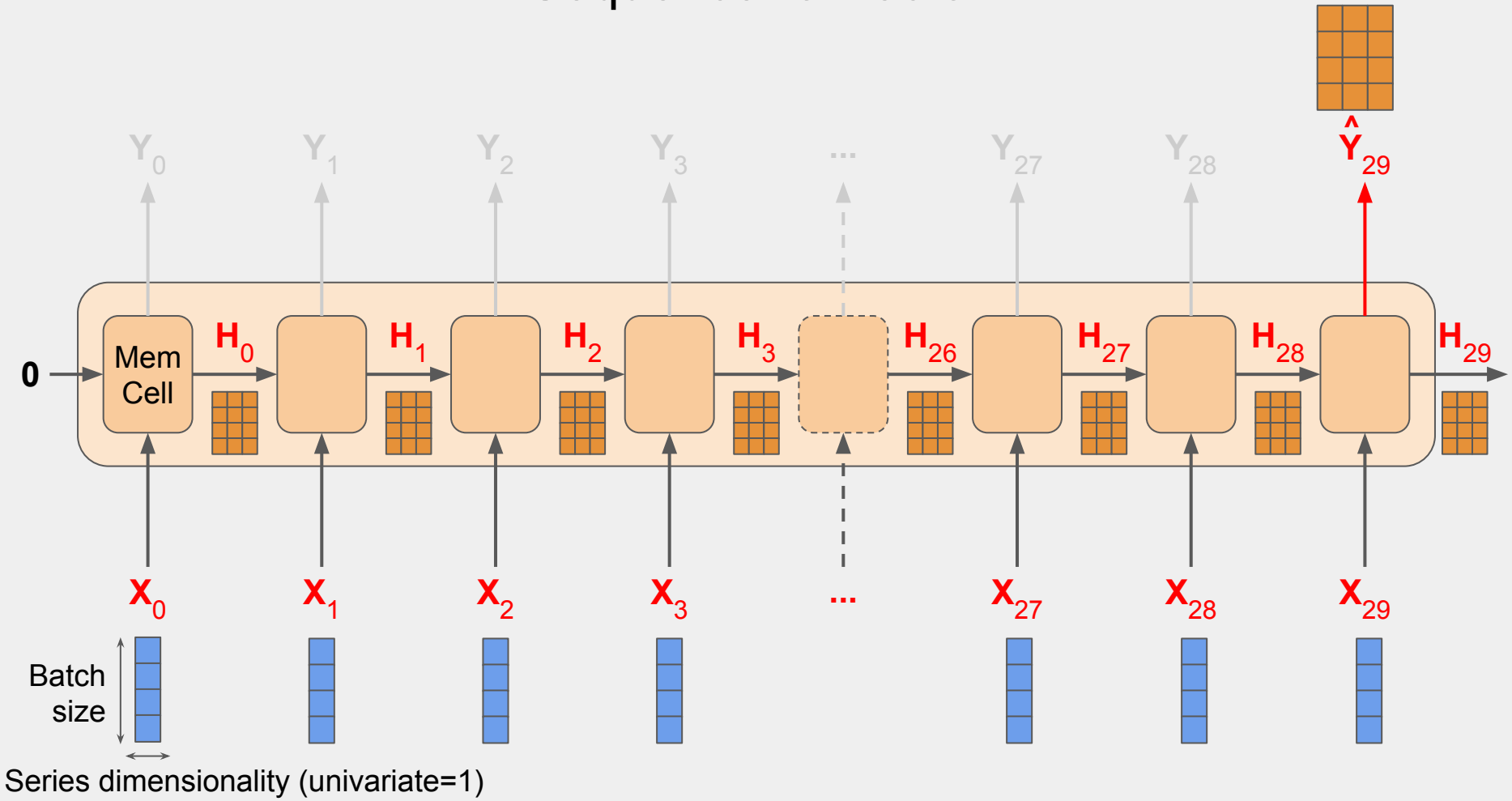

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)


Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
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)

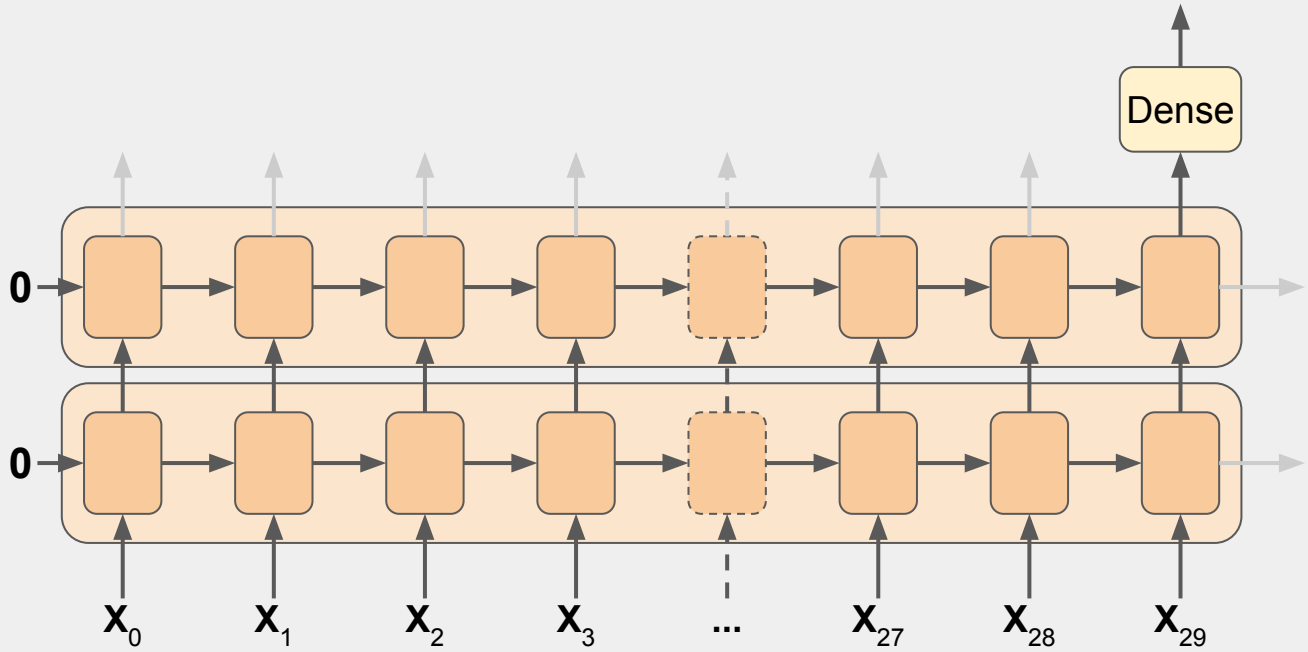

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)


Recurrent Layer
# of units
Batch size
Ŷ0
Ŷ1
Ŷ2
Ŷ3
...
Ŷ27
Ŷ28
Ŷ29
0
Mem Cell
H0
H1
H2
H3
H26
H27
H28
H29
X0
X1
X2
X3
...
X27
X28
X29
Batch size
Series dimensionality (univariate=1)

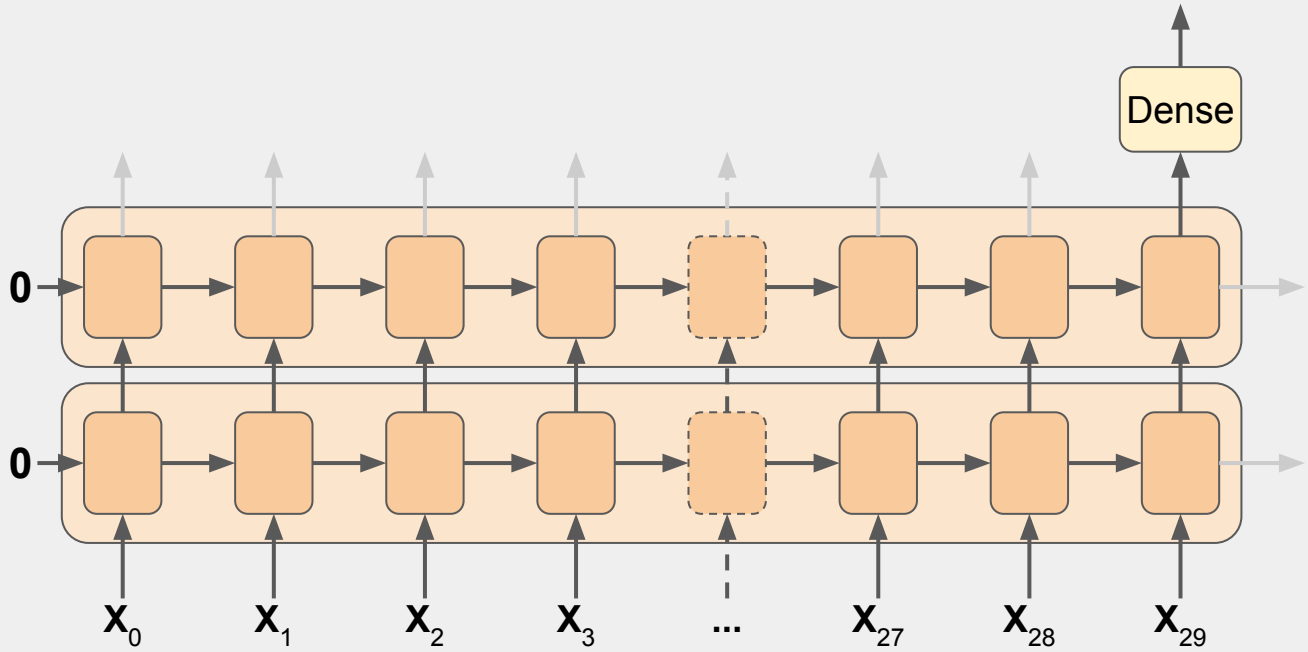
Sequence-to-Vector



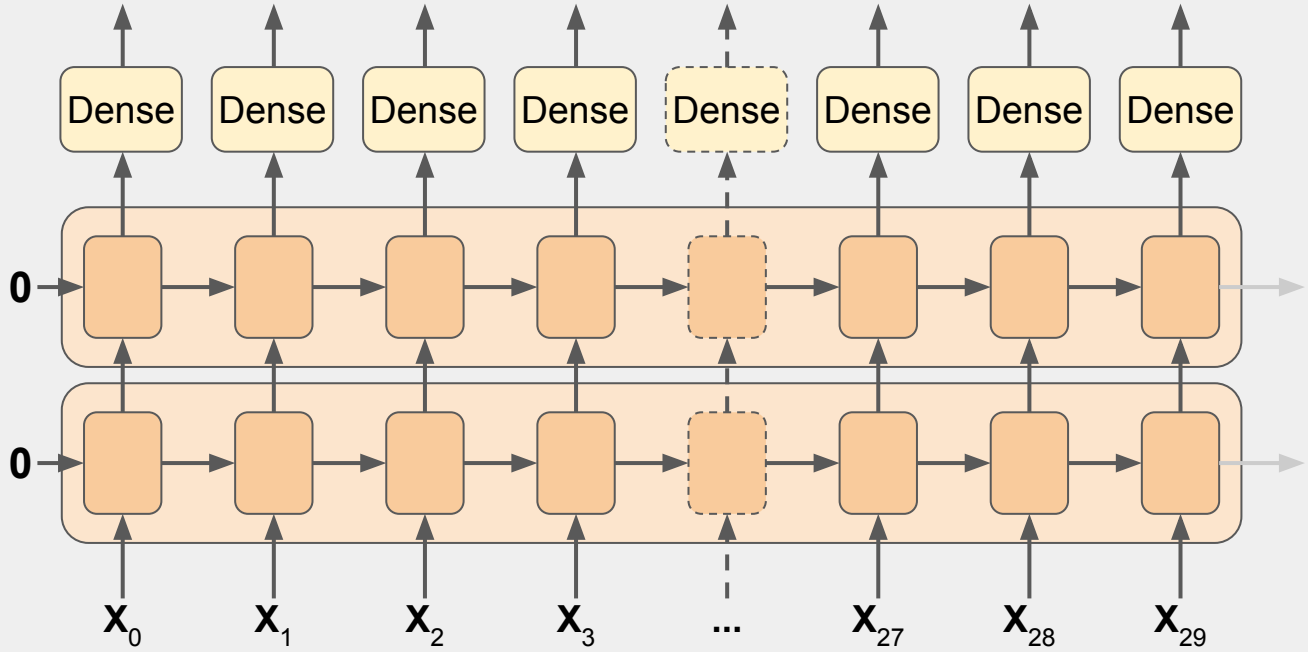
```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1)
])
```



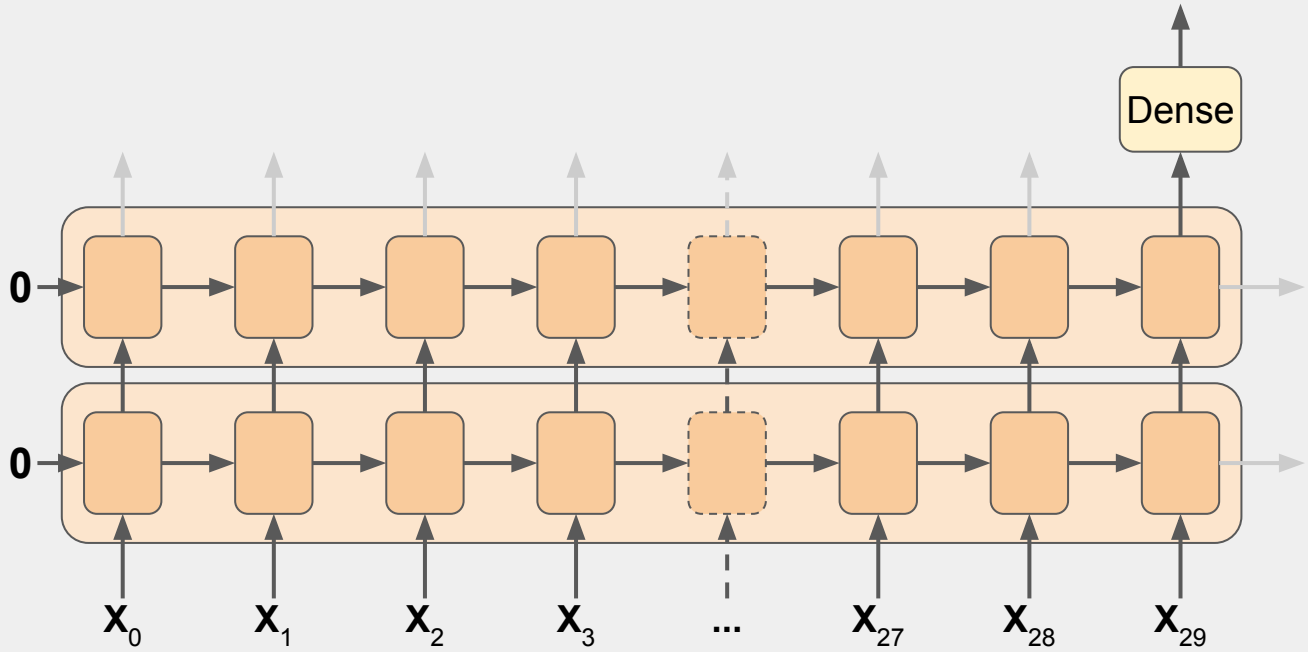
```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1)
])
```



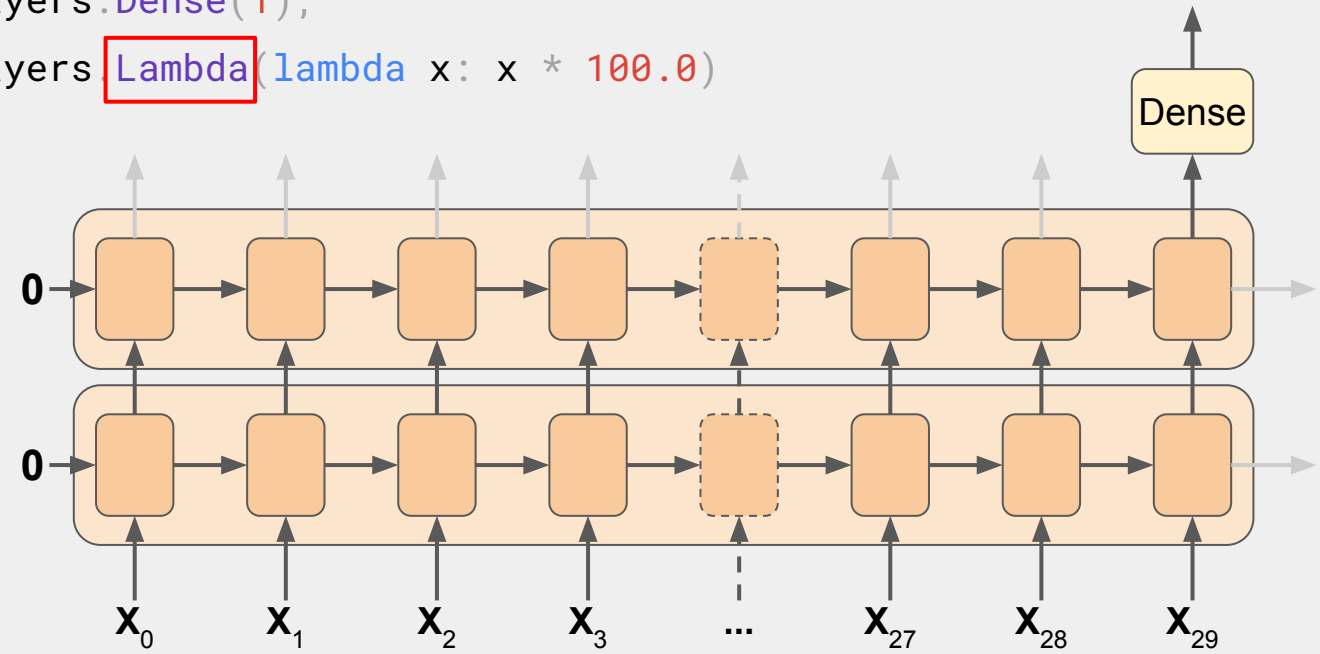
```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.Dense(1)
])
```



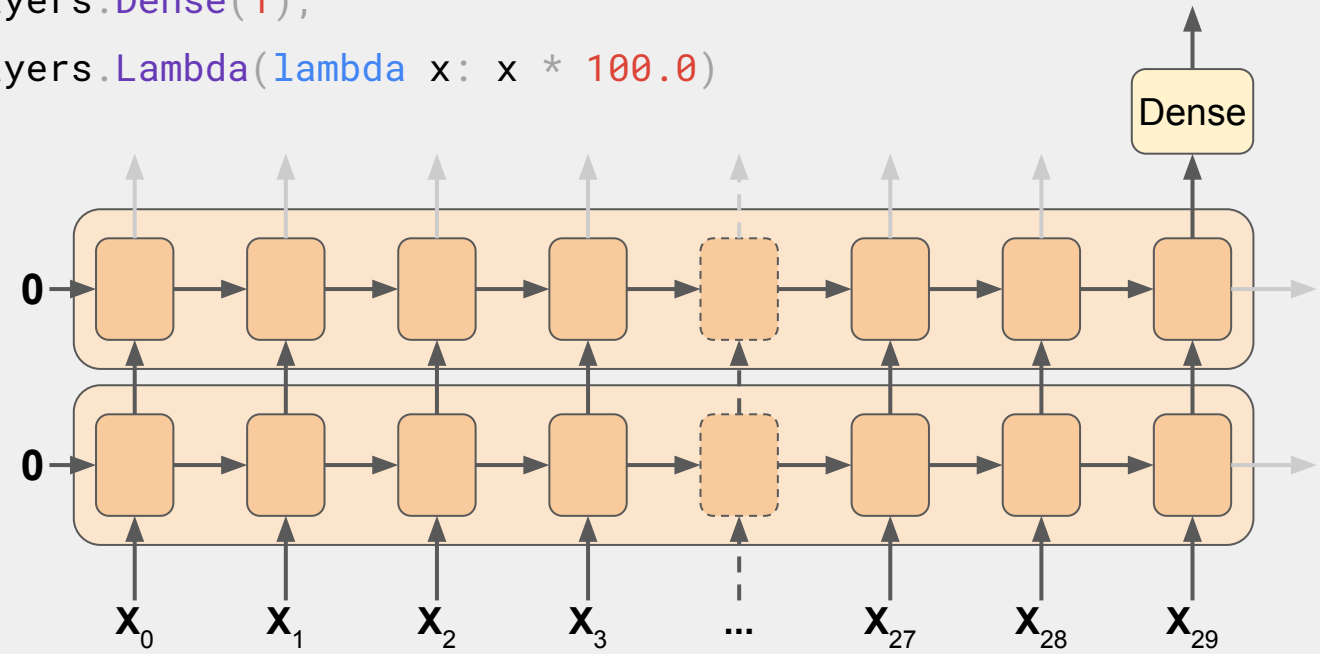
```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1)
])
```



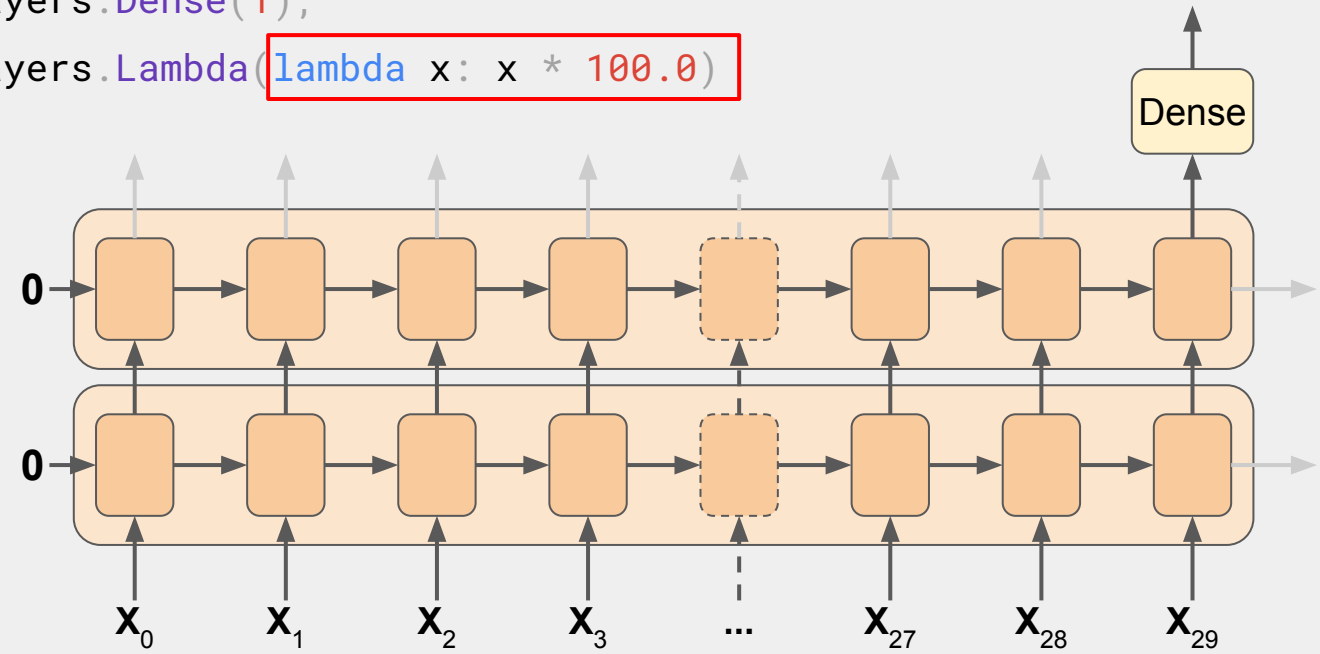
```
model = keras.models.Sequential([
  keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1),
  keras.layers.Lambda(lambda x: x * 100.0)
])
```



```
model = keras.models.Sequential([
  keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1),
  keras.layers.Lambda(lambda x: x * 100.0)
])
```



```
model = keras.models.Sequential([
  keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[None]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1),
  keras.layers.Lambda(lambda x: x * 100.0)
])
```



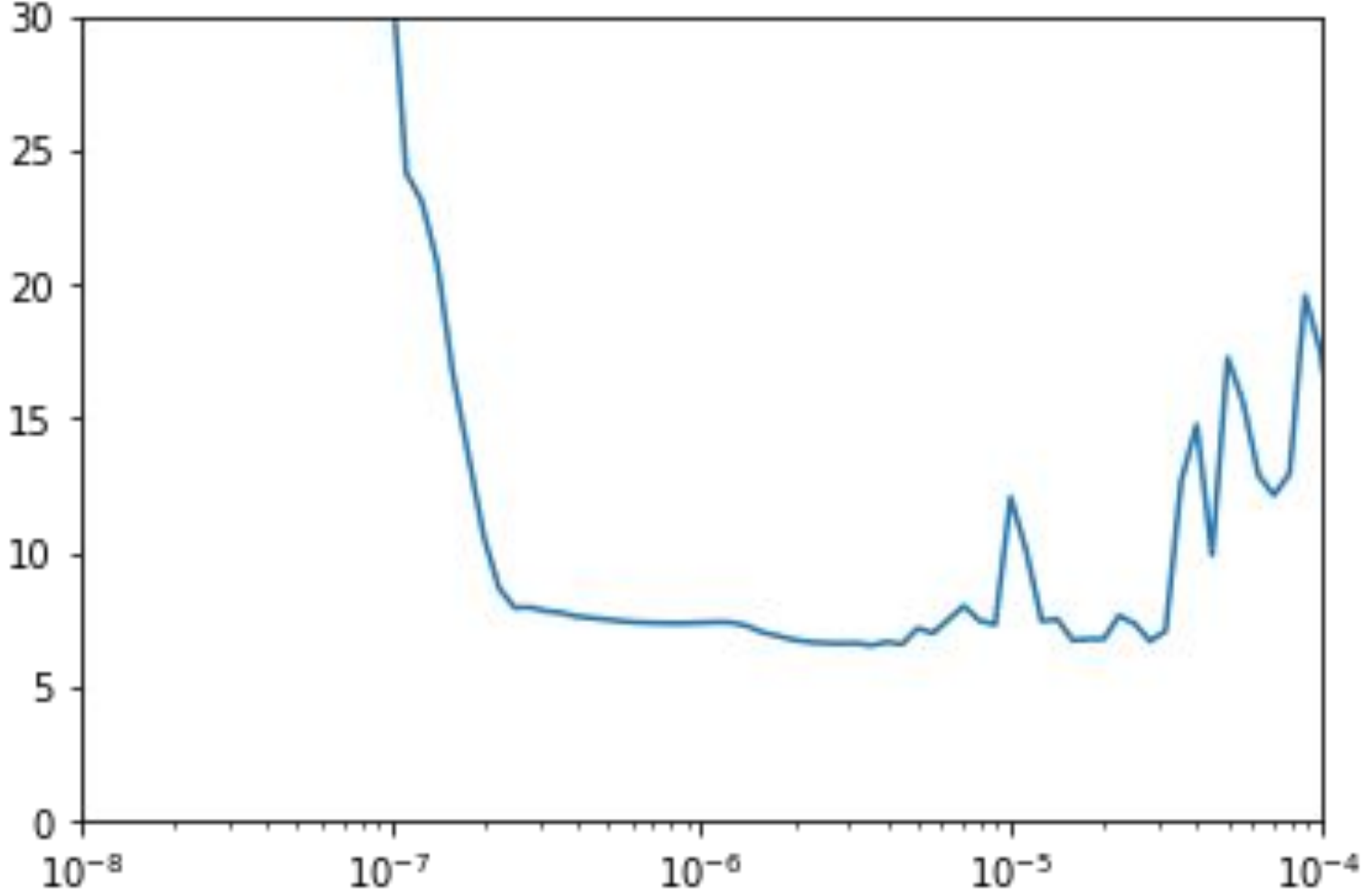
```
train_set = windowed_dataset(x_train, window_size, batch_size=128,
shuffle_buffer=shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), input_shape=[None]),
  tf.keras.layers.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

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

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

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

```
train_set = windowed_dataset(x_train, window_size, batch_size=128,
shuffle_buffer=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.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

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

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

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

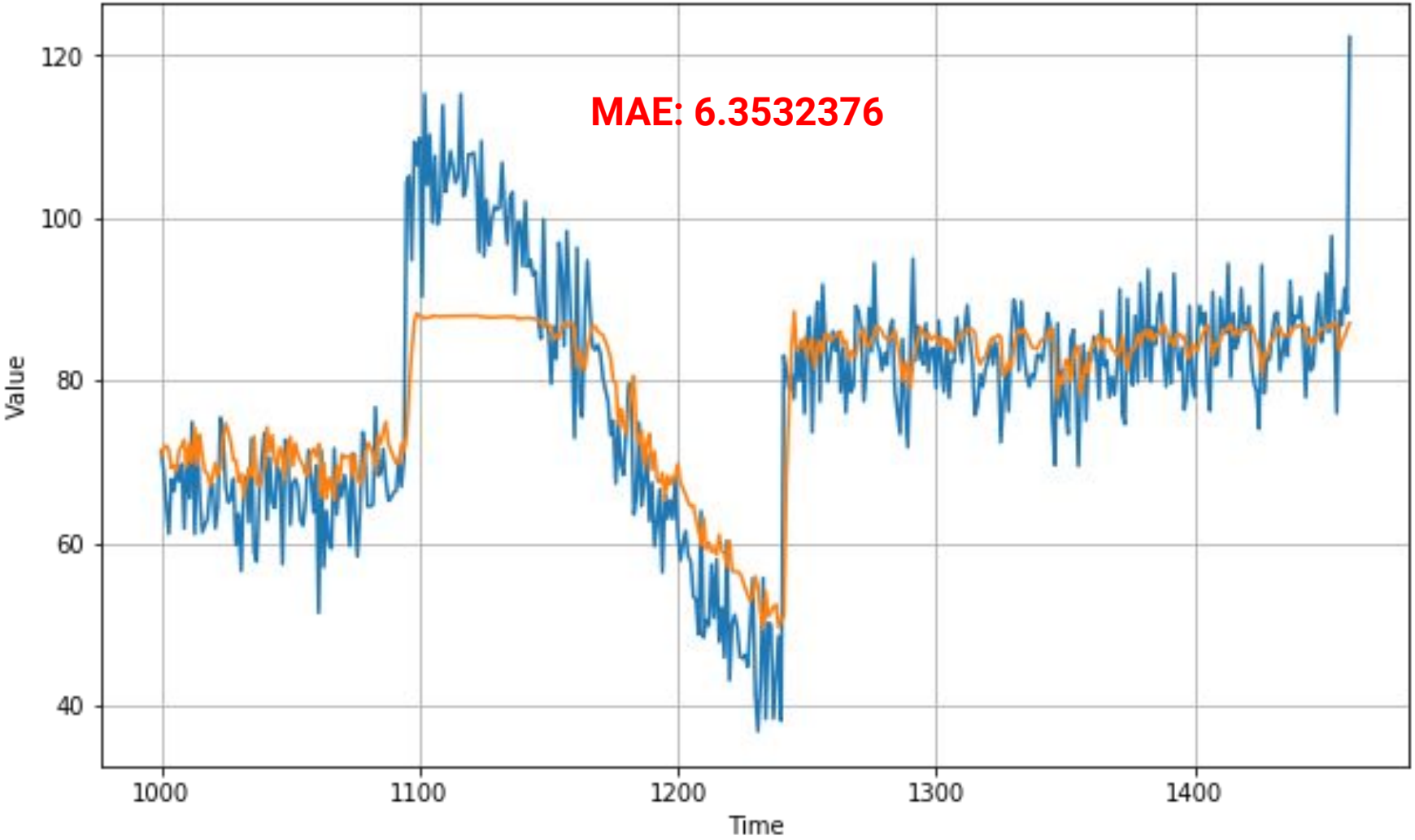
```
train_set = windowed_dataset(x_train, window_size, batch_size=128,
shuffle_buffer=shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), input_shape=[None]),
  tf.keras.layers.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

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

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

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

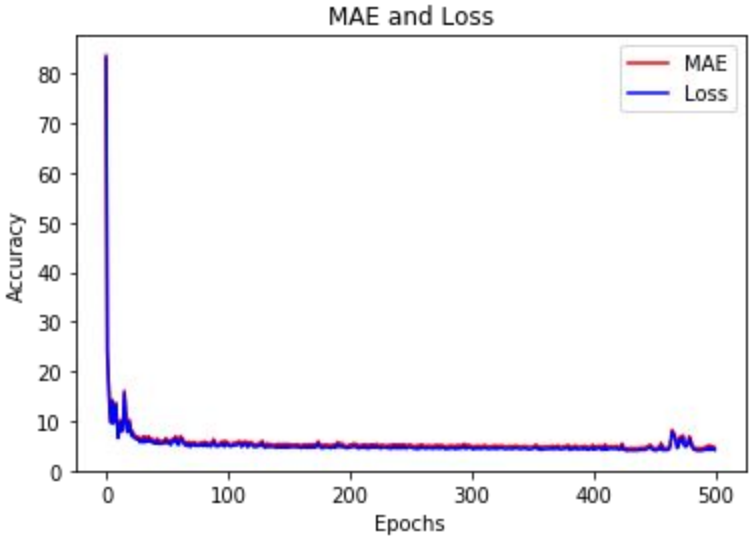
```
train_set = windowed_dataset(x_train, window_size, batch_size=128,
shuffle_buffer=shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), input_shape=[None]),
  tf.keras.layers.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

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

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

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

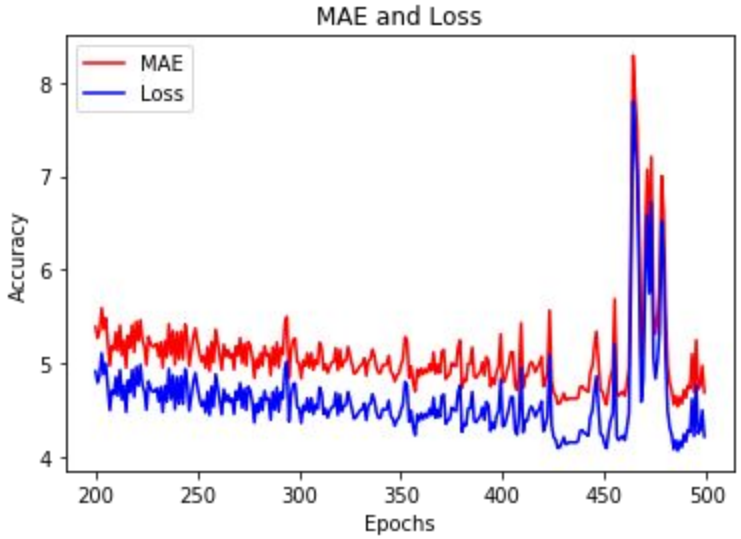
https://en.wikipedia.org/wiki/Huber_loss

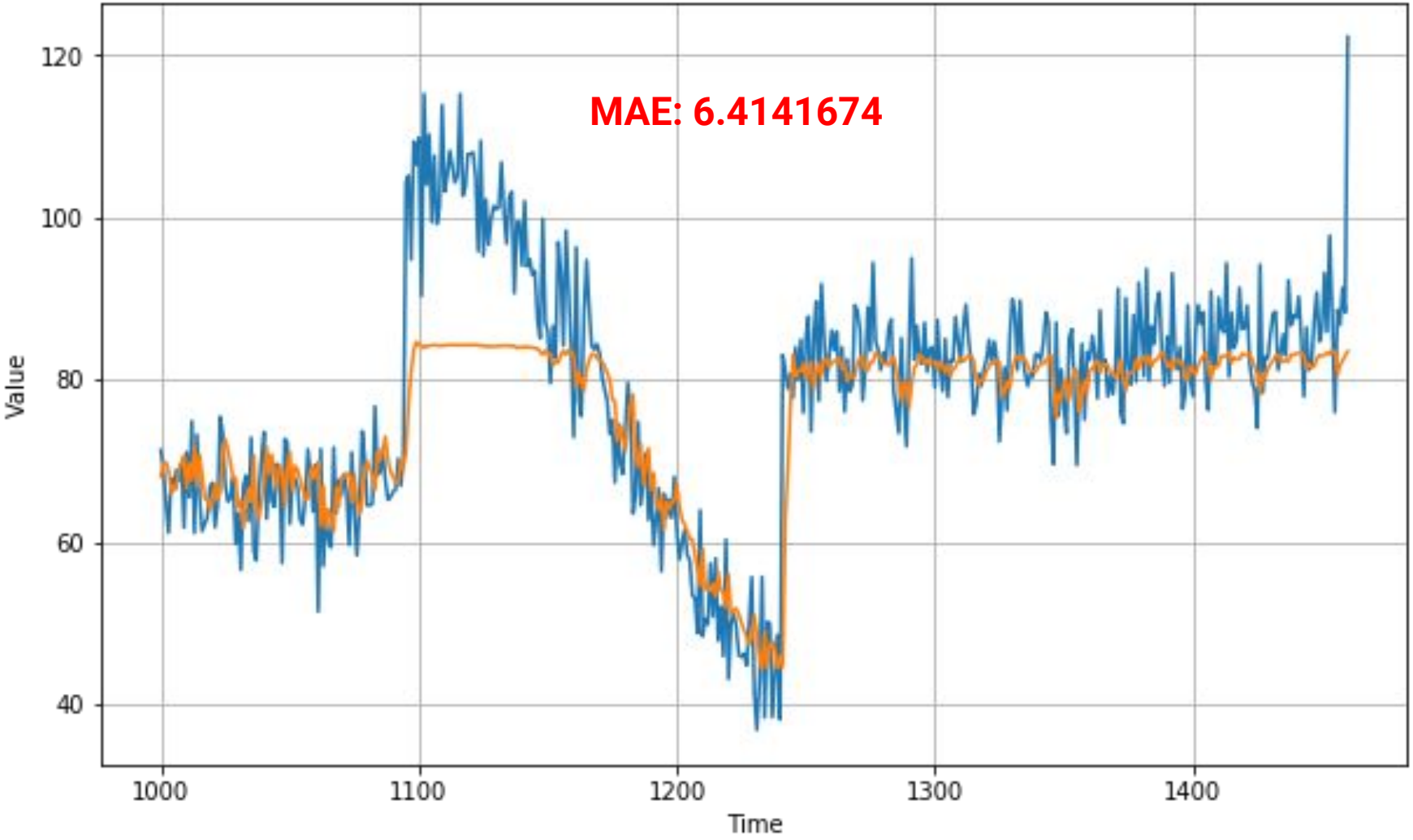

30
25
20
15
10
5
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}

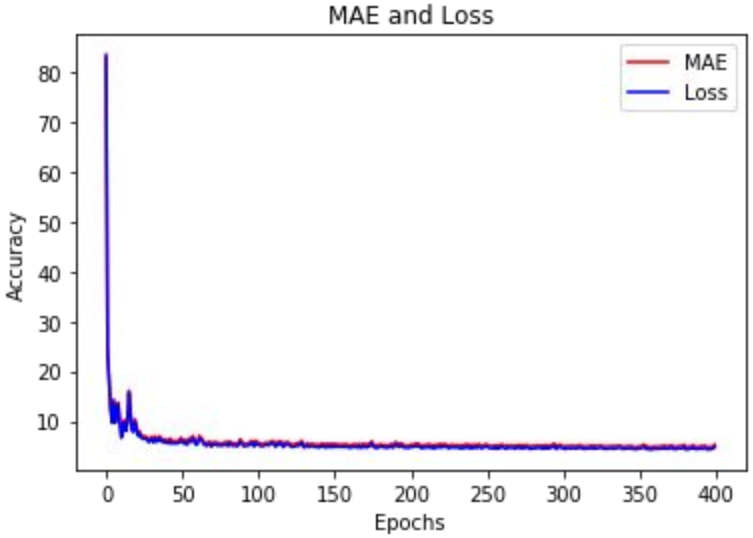
```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

dataset = windowed_dataset(x_train, window_size, batch_size=128,
shuffle_buffer=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.SimpleRNN(40, return_sequences=True),
  tf.keras.layers.SimpleRNN(40),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])

optimizer = tf.keras.optimizers.SGD(learning_rate=5e-6, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset,epochs=500)
```

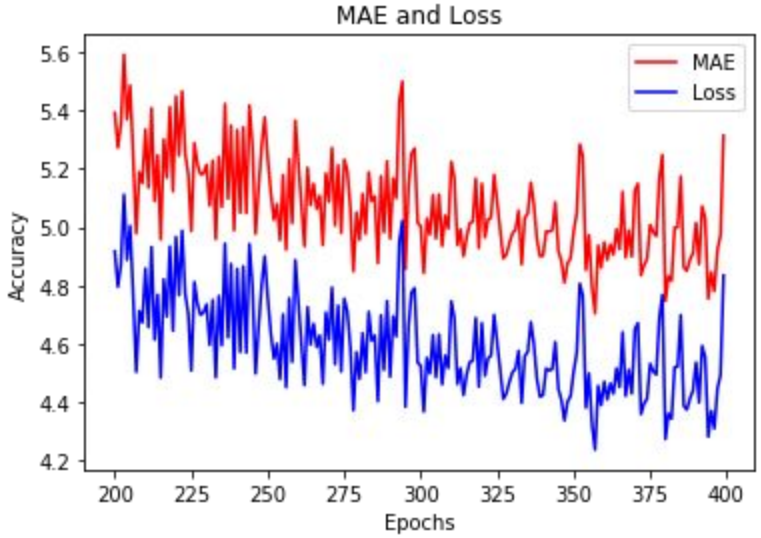

MAE: 6.3532376
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

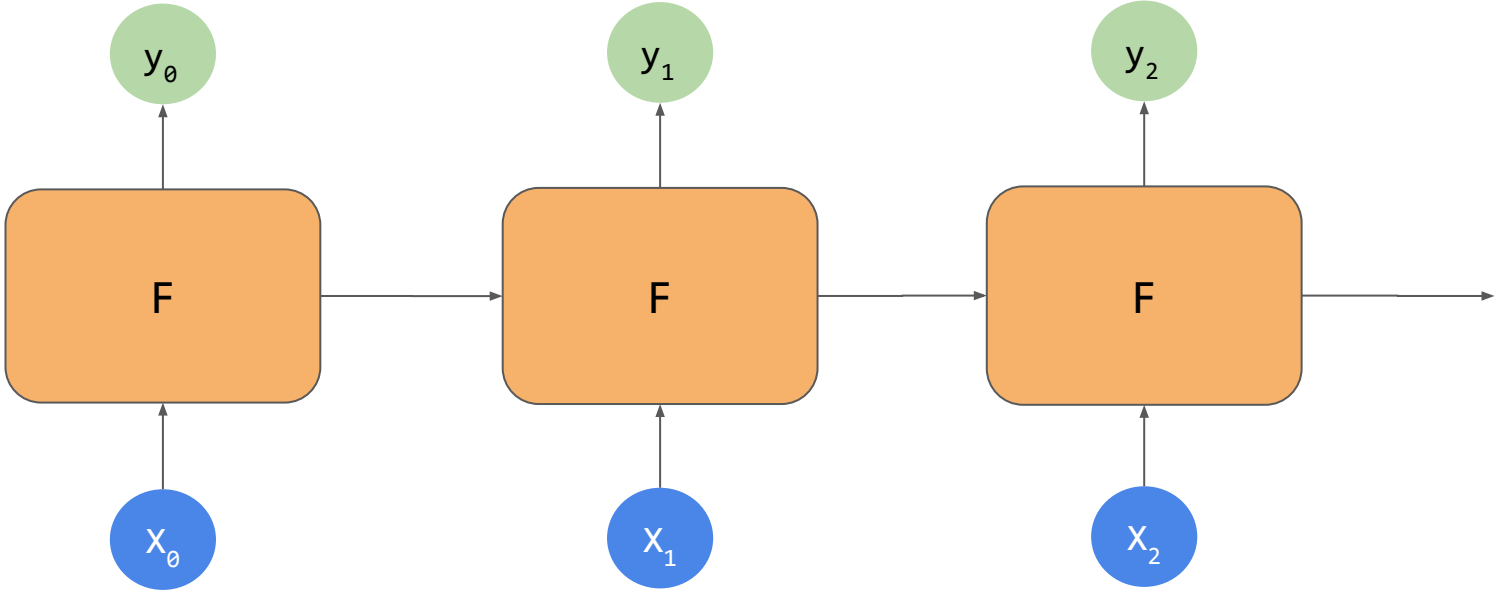

MAE and Loss
MAE
Loss
Accuracy
Epochs
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500

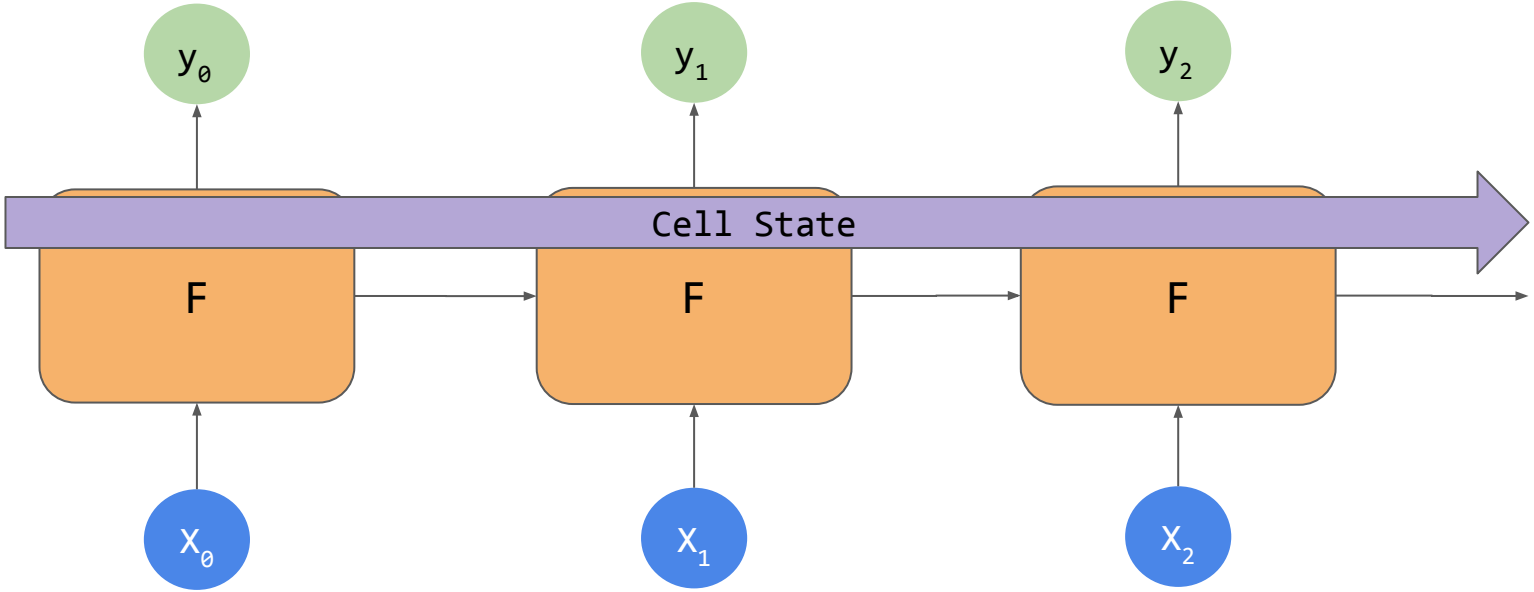

MAE and Loss
MAE
Loss
Accuracy
Epochs
4
5
6
7
8
200
250
300
350
400
450
500

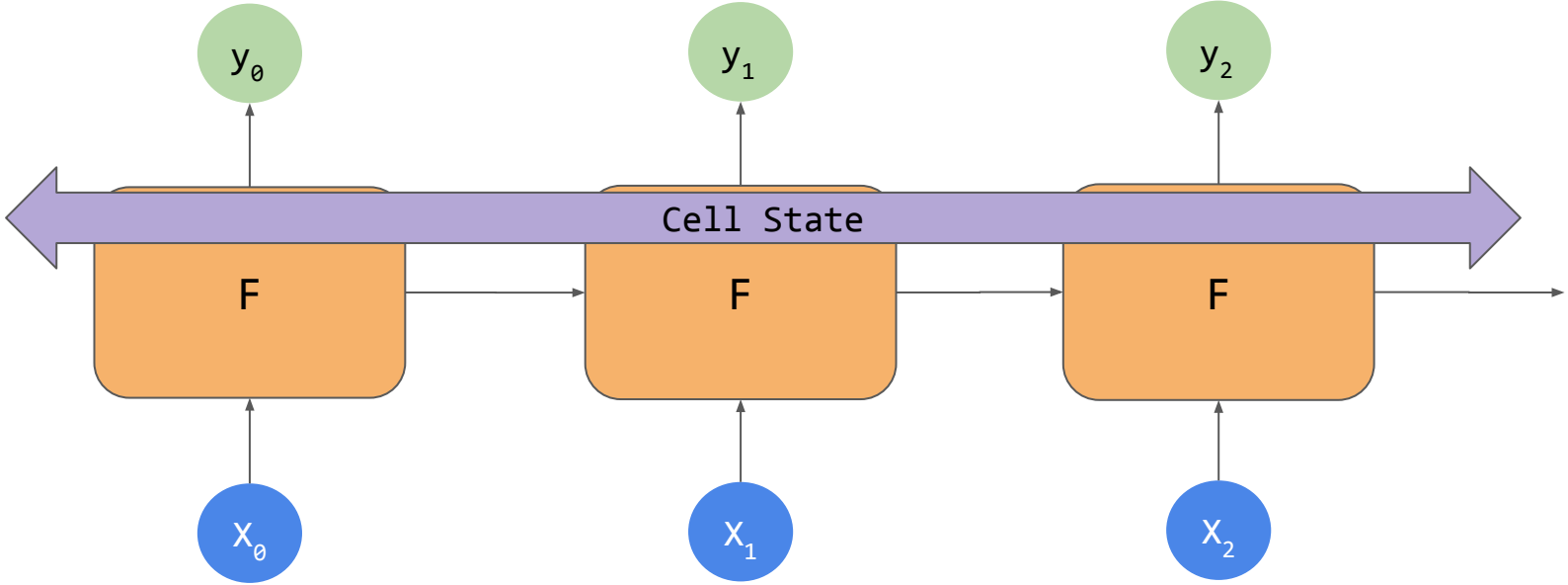

MAE: 6.4141674
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time


MAE and Loss
MAE
Loss
Accuracy
Epochs
80
70
60
50
40
30
20
10
0
50
100
150
200
250
300
350
400


MAE and Loss
MAE
Loss
Accuracy
Epochs
5.6
5.4
5.2
5.0
4.8
4.6
4.4
4.2
200
225
250
275
300
325
350
375
400

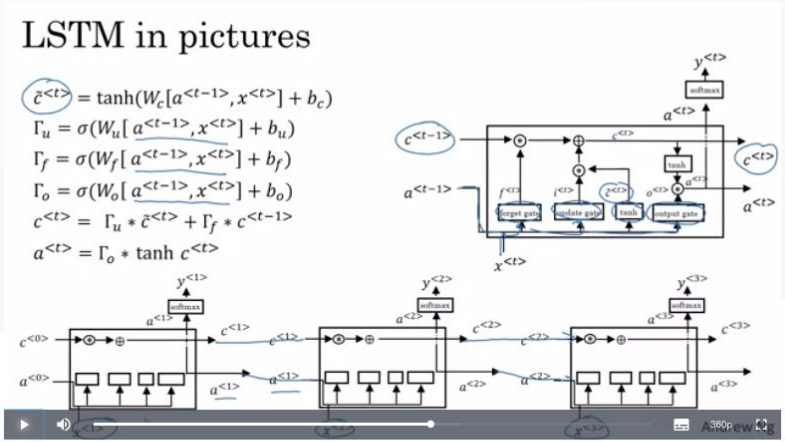

y_0
y_1
y_2
F
F
F
x_0
x_1
x_2


y0
y1
y2
Cell State
F
F
F
x0
x1
x2


y_0
y_1
y_2
Cell State
F
F
F
x_0
x_1
x_2

https://www.coursera.org/lecture/nlp-sequence-models/long-short-term-memory-lstm-KXoay

Long Short Term Memory (LSTM)



From the course by deeplearning.ai

Sequence Models

13393 ratings

Try the Course for Free

This Course | Video Transcript

deeplearning.ai

Sequence Models

13393 ratings

Course 5 of 5 in the Specialization Deep Learning

This course will teach you how to build models for natural language, audio, and other sequence data. Thanks to deep learning, sequence algorithms are working far better than just two years ago, and this is enabling numerous exciting applications in speech recognition, music synthesis, chatbots, machine translation, natural language understanding, and many others. You will: - Understand how to build and train Recurrent Neural Networks (RNNs), and commonly-

More

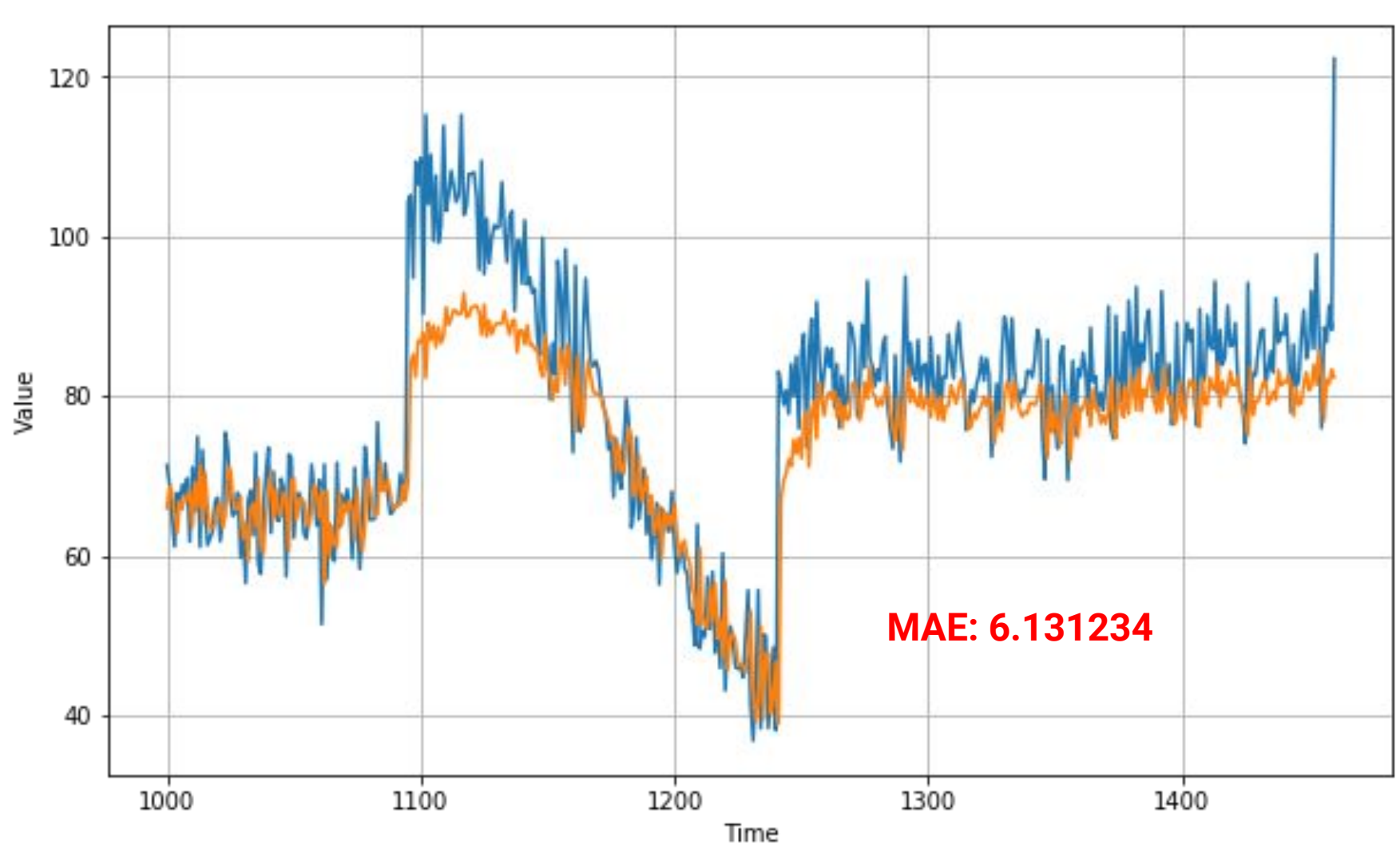
```
tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

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)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

```
tf.keras.backend.clear_session()
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)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

```
tf.keras.backend.clear_session()
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)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

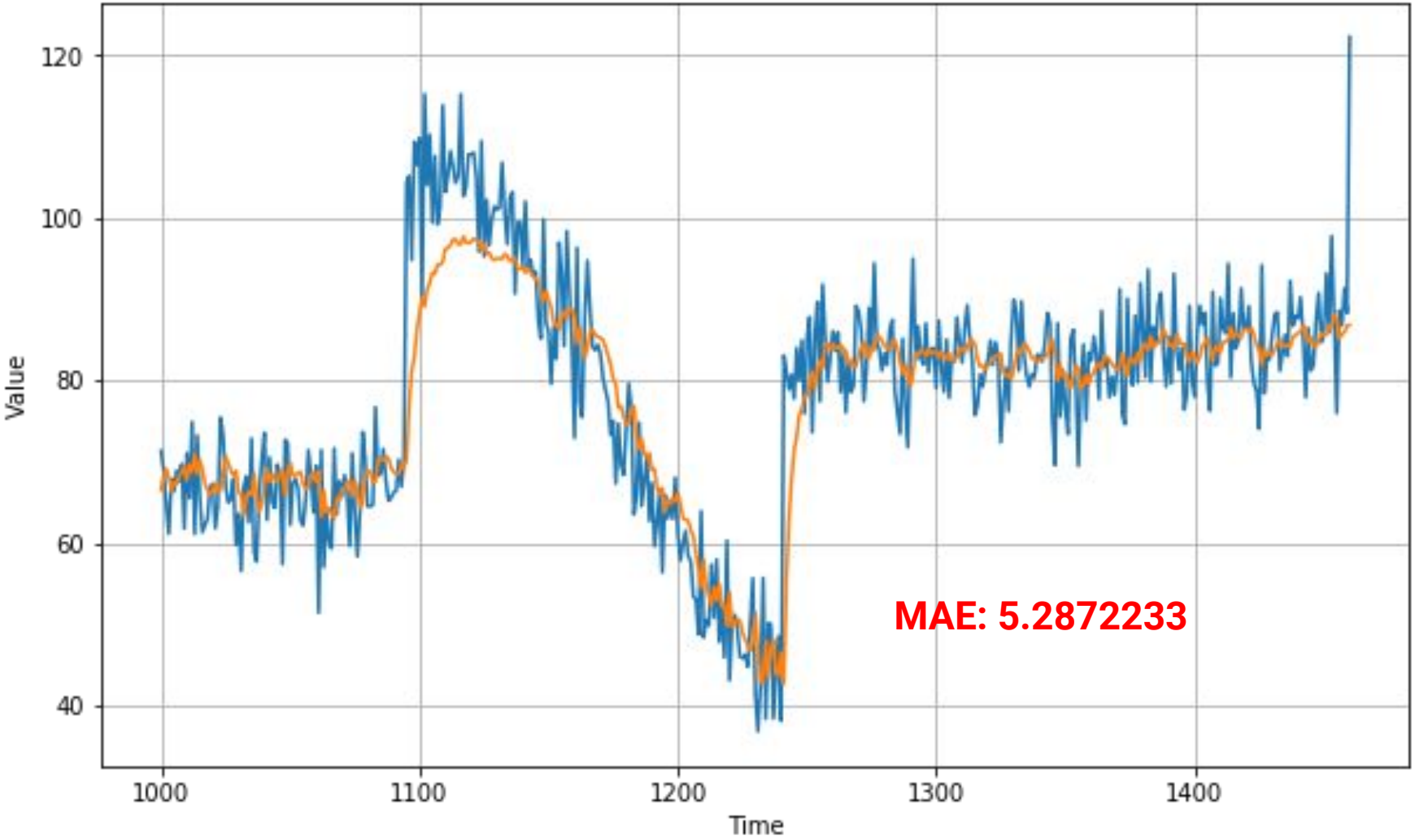
```
tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

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)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

```
tf.keras.backend.clear_session()
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)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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


120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time
MAE: 6.131234

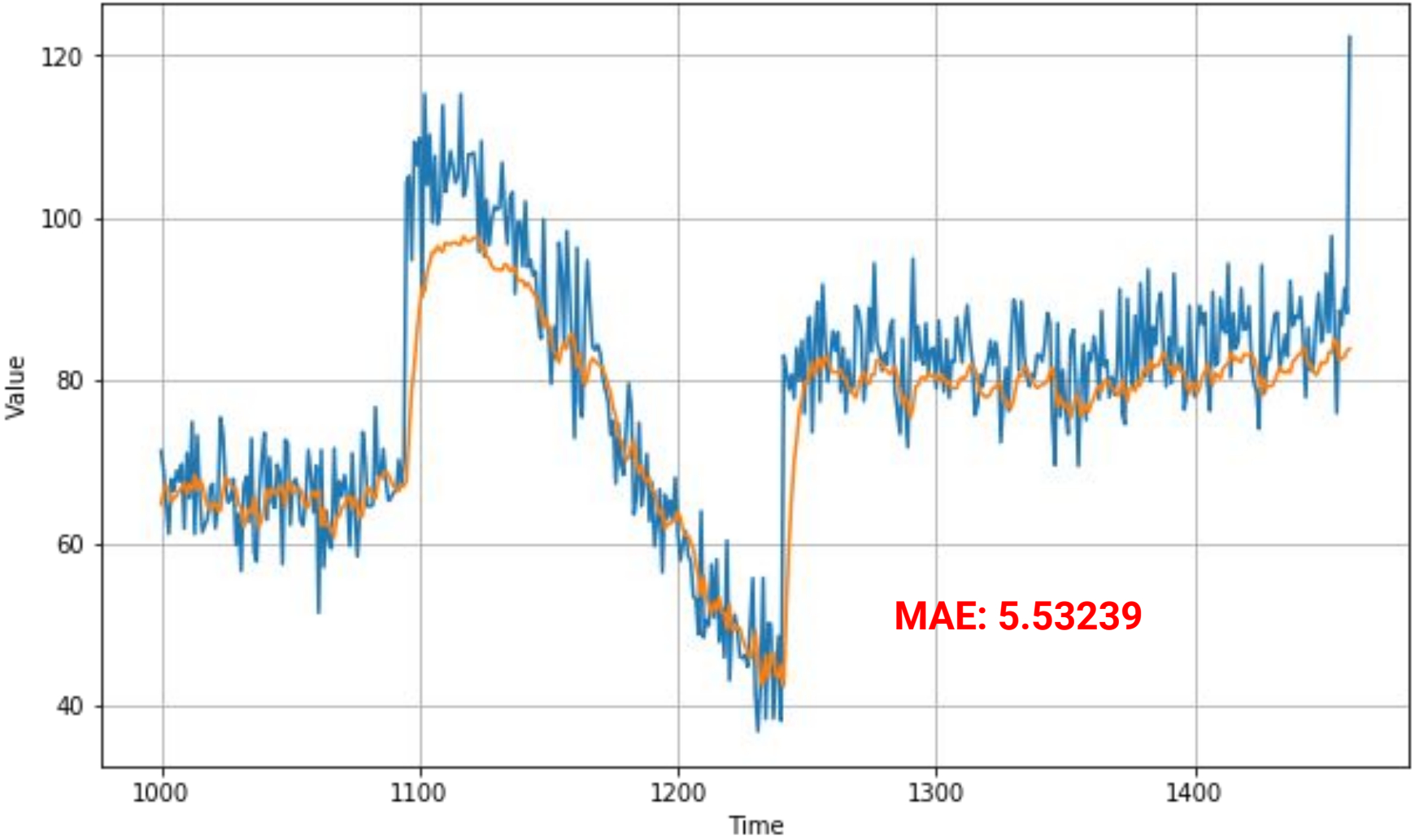
```
tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

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 * 100.0)
])


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

```
tf.keras.backend.clear_session()
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 * 100.0)
])


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


120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time
MAE: 5.2872233

```
tf.keras.backend.clear_session()
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, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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

```
tf.keras.backend.clear_session()
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, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])


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


120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time
MAE: 5.53239

Copyright Notice

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 200)
])

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)

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

model.fit(dataset,epochs=500)
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 200)
])

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)

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

model.fit(dataset,epochs=500)
```

https://www.coursera.org/learn/convolutional-neural-networks-tensorflow

deeplearning.ai **TensorFlow in Practice Specialization** Go To Course

About **Syllabus** Reviews Instructors Enrollment Options FAQ

Syllabus - What you will learn from this course

WEEK 1

4 hours to complete

Exploring a Larger Dataset

In the first course in this specialization, you had an introduction to TensorFlow, and how, with its high level APIs you could do basic image classification, an you learned a little bit about Convolutional Neural Networks (ConvNets). In this course you'll go deeper into using ConvNets will real-world data, and learn about techniques that you can u... SHOW ALL

8 videos (Total 18 min), 5 readings, 3 quizzes SEE ALL

WEEK 2

4 hours to complete

Augmentation: A technique to avoid overfitting

You've heard the term overfitting a number of times to this point. Overfitting is simply the concept of being over specialized in training -- namely that your model is very good at classifying what it is trained for, but not so good at classifying things that it hasn't seen. In order to generalize your model more effectively, you will of course need a ... SHOW ALL

7 videos (Total 14 min), 6 readings, 3 quizzes SEE ALL

WEEK 3

4 hours to complete

Transfer Learning

Building models for yourself is great, and can be very powerful. But, as you've seen, you can be limited by the data you have on hand. Not everybody has access to massive datasets or the compute power that's needed to train them effectively. Transfer learning can help solve this -- where people with models trained on large datasets train them, ... SHOW ALL

7 videos (Total 14 min), 5 readings, 3 quizzes SEE ALL

WEEK 4

4 hours to complete

Multiclass Classifications

Introduction, A conversation with Andrew Ng

Save Note Discuss Download

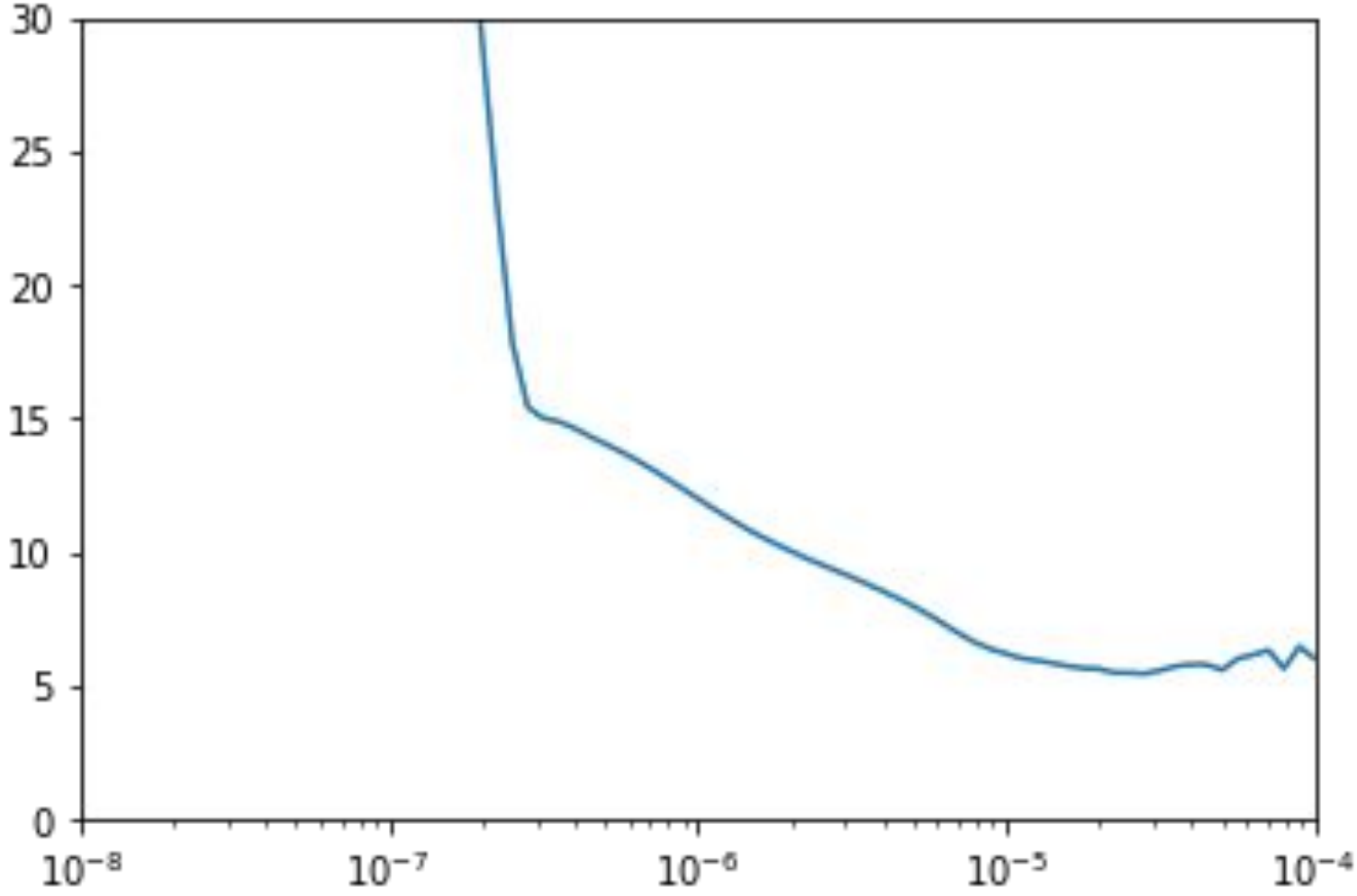
English

Help Us Translate

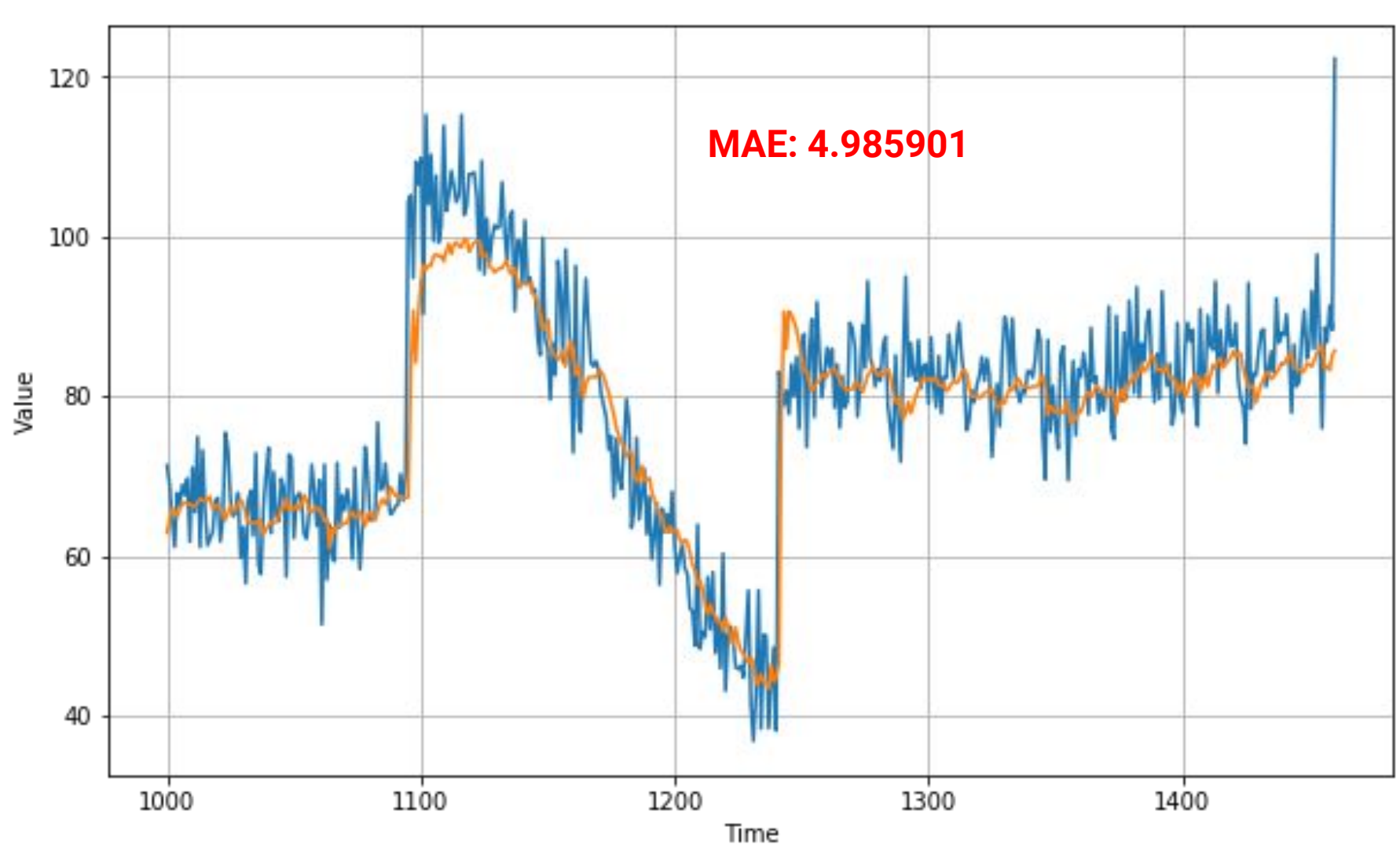
0:00 In the first course, you learned how to use TensorFlow to implement a basic neural network, going up all the way to basic Convolutional Neural Network. In this second course, you go much further. In the first week, you take the ideas you've learned, and apply them to a much bigger dataset of cats versus dogs on Kaggle. Yes so we take the full Kaggle dataset of 25,000 cats versus dogs

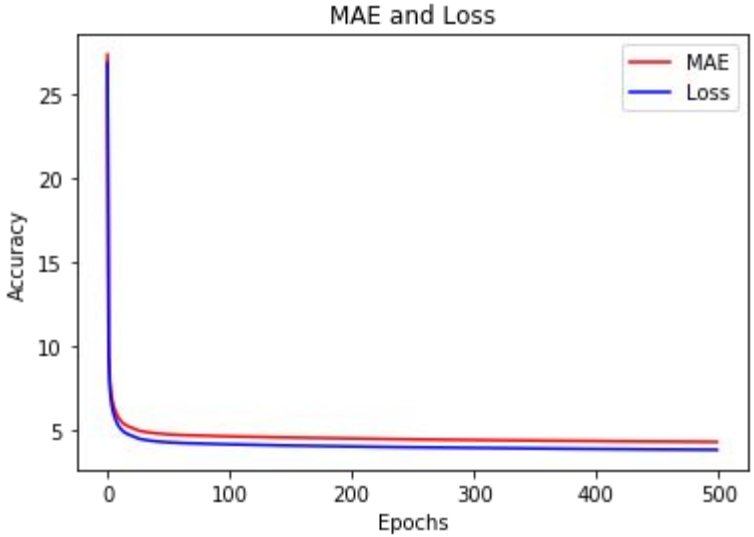
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 200)
])

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)

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

model.fit(dataset,epochs=500)
```

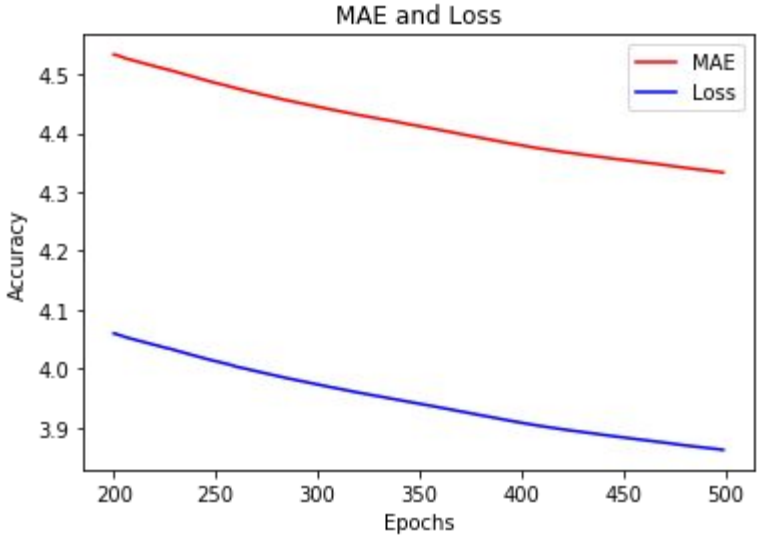
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):

    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[-1]))

    return ds.batch(batch_size).prefetch(1)
```


30
25
20
15
10
5
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 200)
])

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)

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

model.fit(dataset,epochs=500)
```

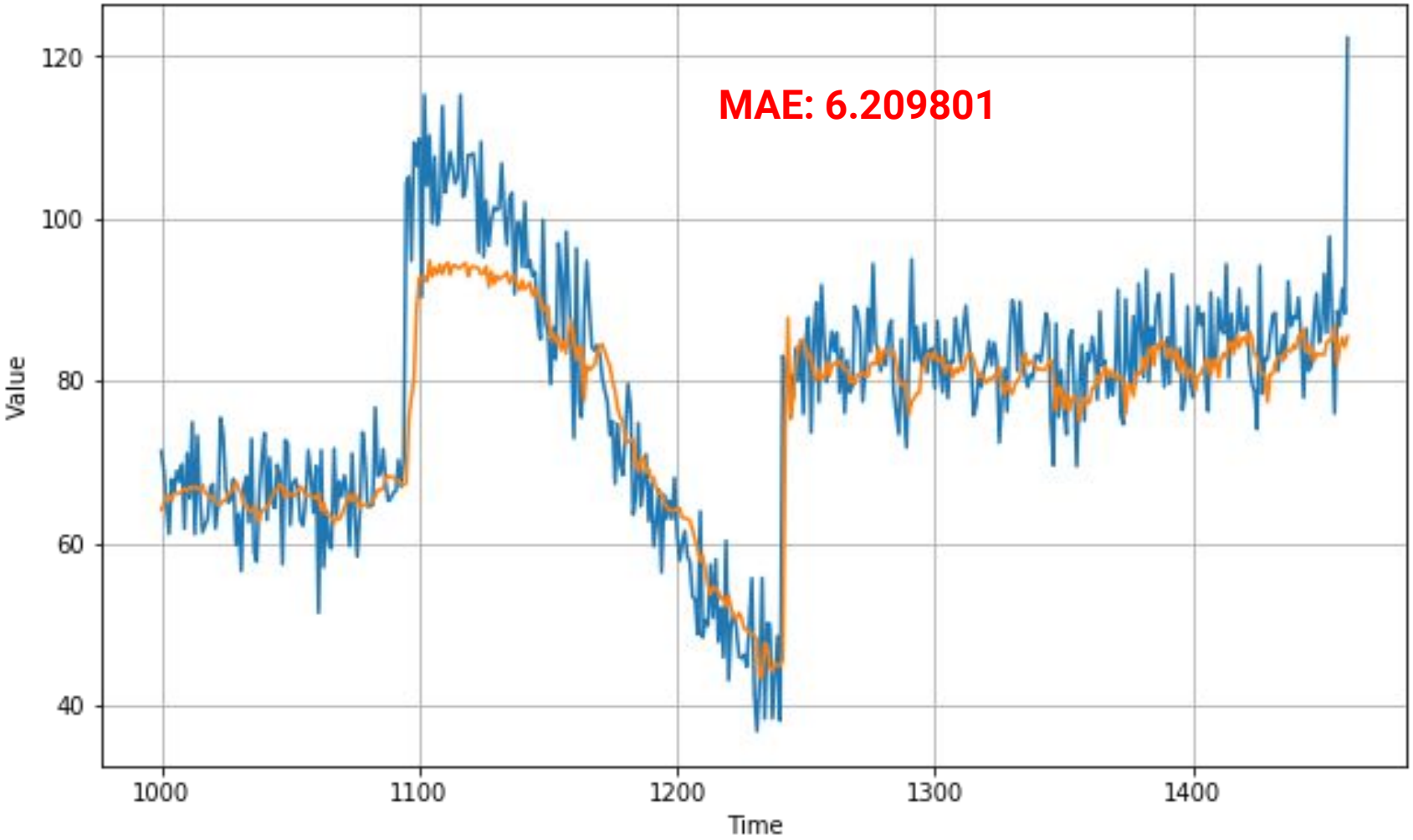

MAE: 4.985901
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

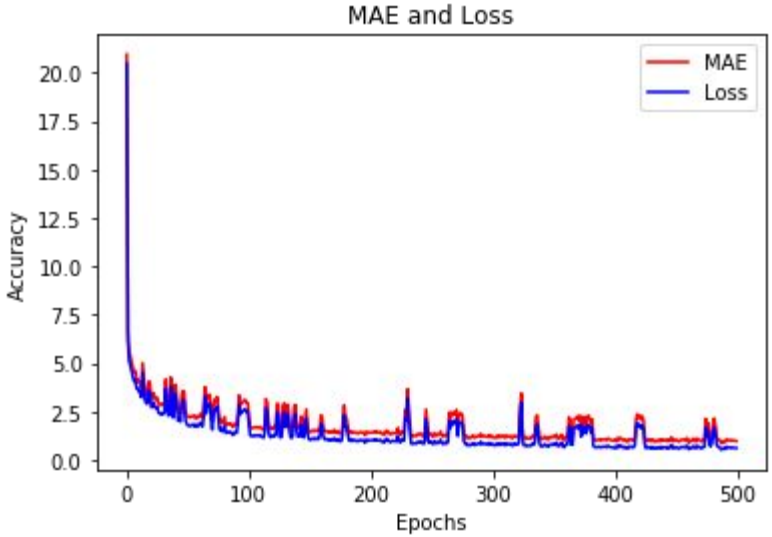

MAE and Loss
MAE
Loss
Accuracy
25
20
15
10
5
0
100
200
300
400
500
Epochs

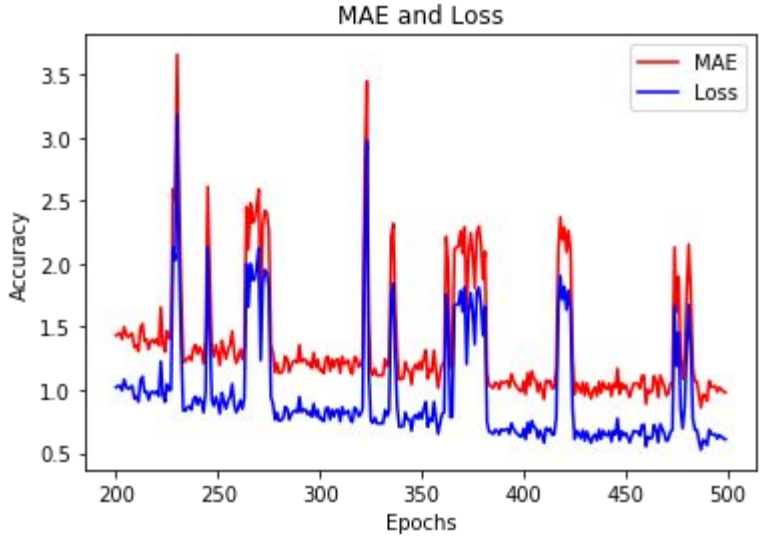

MAE and Loss
MAE
Loss
Accuracy
4.5
4.4
4.3
4.2
4.1
4.0
3.9
200
250
300
350
400
450
500
Epochs

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  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 * 200)
])
```

```
Epoch 495/500
31/31 [==============================] - 2s 58ms/step - loss: 0.6491 - mae: 1.0314
Epoch 496/500
31/31 [==============================] - 2s 60ms/step - loss: 0.6155 - mae: 0.9857
Epoch 497/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6425 - mae: 1.0207
Epoch 498/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6330 - mae: 1.0046
Epoch 499/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6155 - mae: 0.9877
Epoch 500/500
31/31 [==============================] - 2s 59ms/step - loss: 0.6111 - mae: 0.9806
```


MAE: 6.209801
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

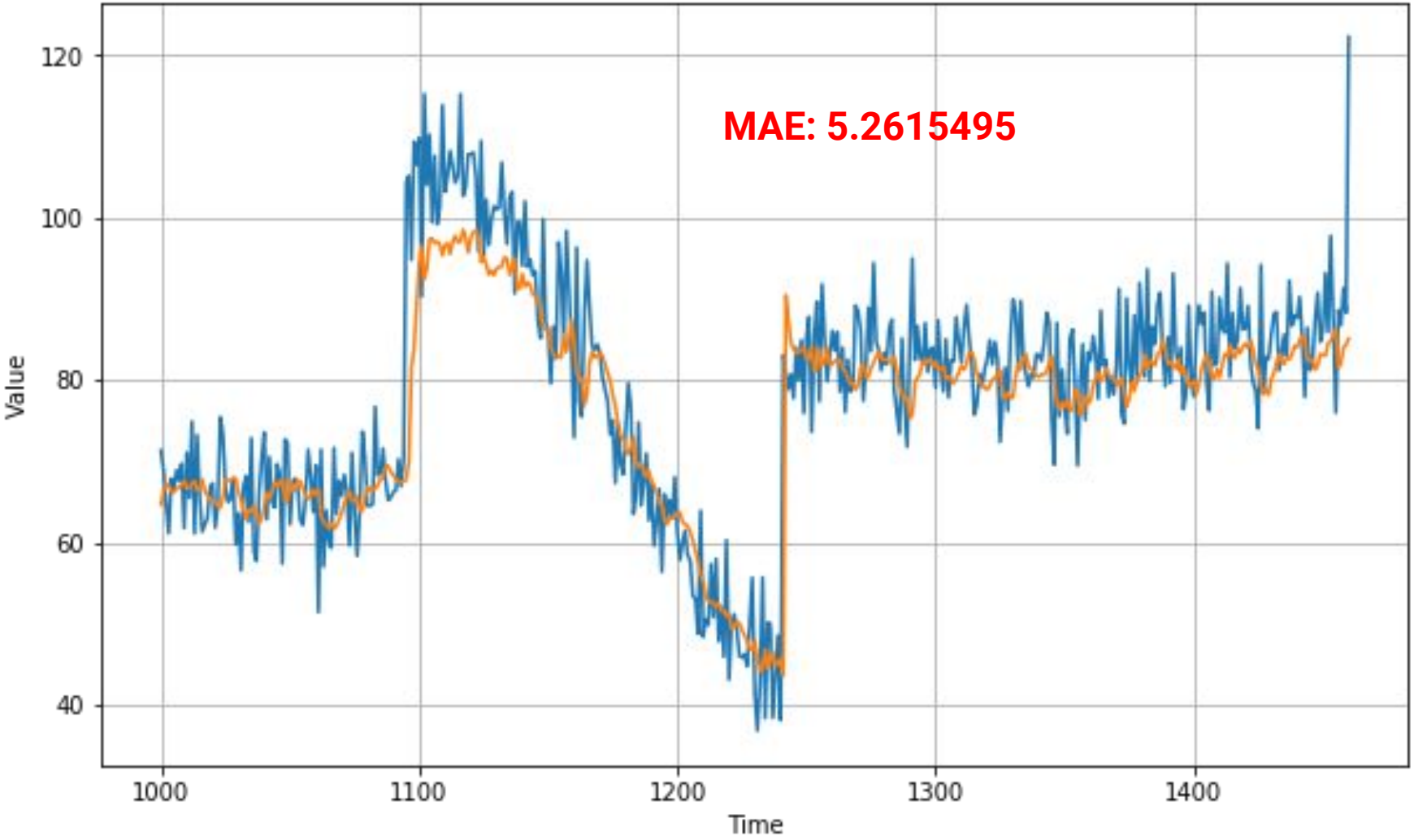

MAE and Loss
MAE
Loss
Accuracy
Epochs
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
100
200
300
400
500

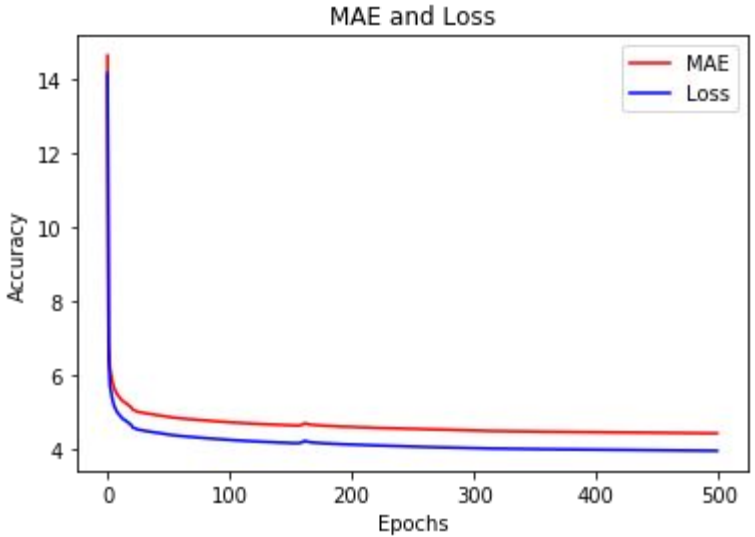

MAE and Loss
MAE
Loss
Accuracy
Epochs
3.5
3.0
2.5
2.0
1.5
1.0
0.5
200
250
300
350
400
450
500

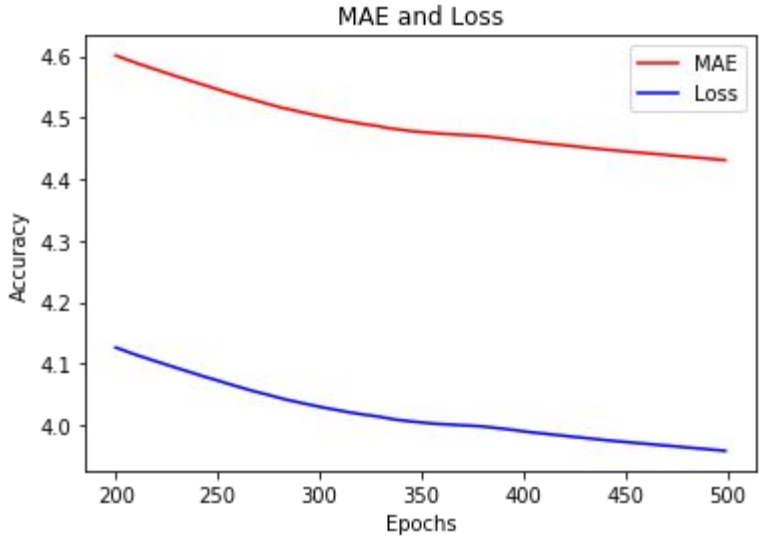
https://www.coursera.org/learn/deep-neural-network

youtube.com/watch?v=l4lSUAcvHF

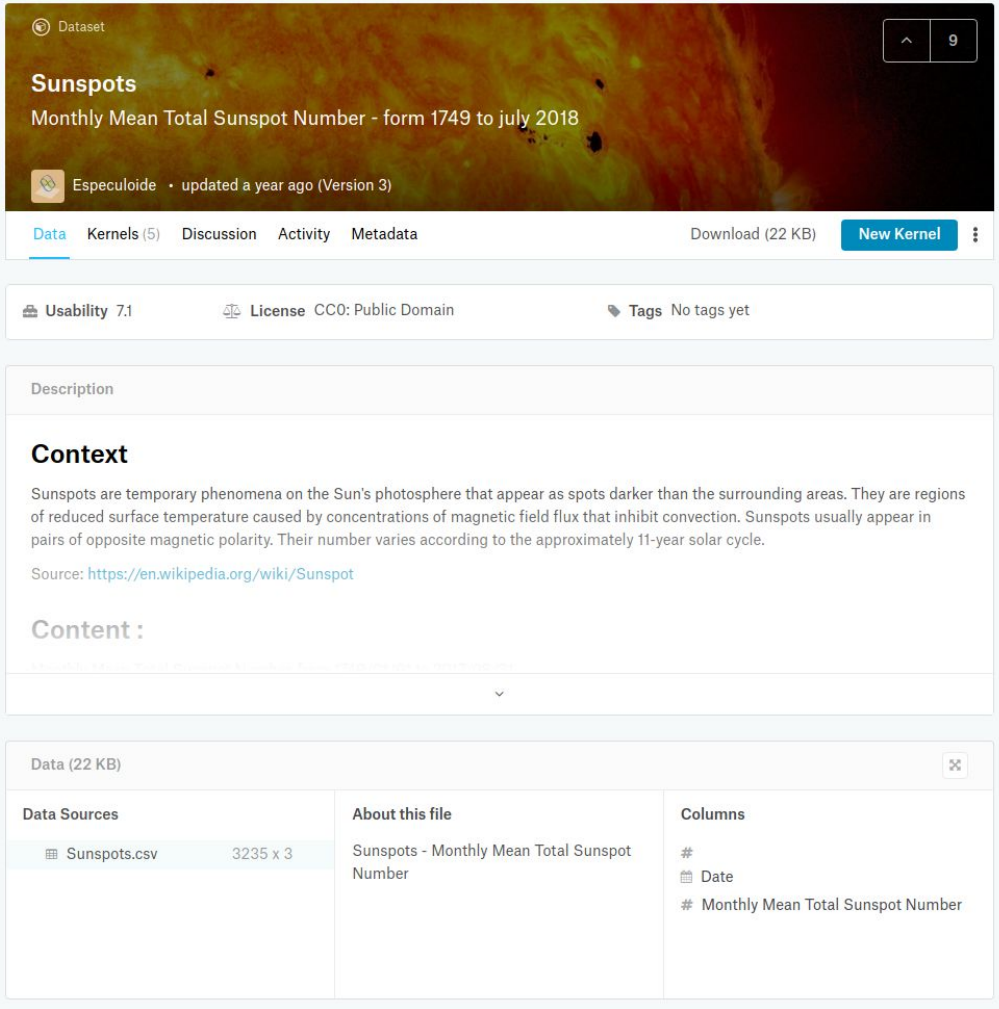
Lecture 17.3 — Large Scale Machine Learning | Mini Batch Gradient Descent — [Andrew Ng]


MAE: 5.2615495
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time


MAE and Loss
MAE
Loss
Accuracy
Epochs
14
12
10
8
6
4
0
100
200
300
400
500


MAE and Loss
MAE
Loss
Accuracy
Epochs
4.6
4.5
4.4
4.3
4.2
4.1
4.0
200
250
300
350
400
450
500

https://www.kaggle.com/robervalt/sunspots



Sunspots.csv

```
,Date,Monthly Mean Total Sunspot Number
0,1749-01-31,96.7
1,1749-02-28,104.3
2,1749-03-31,116.7
3,1749-04-30,92.8
4,1749-05-31,141.7
5,1749-06-30,139.2
6,1749-07-31,158.0
7,1749-08-31,110.5
8,1749-09-30,126.5
9,1749-10-31,125.8
10,1749-11-30,264.3
11,1749-12-31,142.0
12,1750-01-31,122.2
13,1750-02-28,126.5
14,1750-03-31,148.7
15,1750-04-30,147.2
16,1750-05-31,150.0
17,1750-06-30,166.7
```

```
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/Sunspots.csv \
    -O /tmp/sunspots.csv
```

```
import csv
time_step = []
sunspots = []

with open('/tmp/sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    sunspots.append(float(row[2]))
    time_step.append(int(row[0]))
```

```
import csv
time_step = []
sunspots = []

with open('/tmp/sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    sunspots.append(float(row[2]))
    time_step.append(int(row[0]))
```

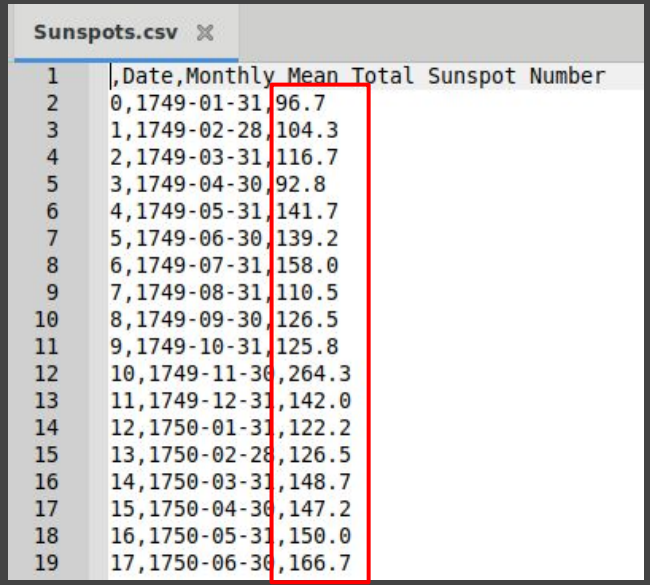
```
import csv
time_step = []
sunspots = []

with open('/tmp/sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    sunspots.append(float(row[2]))
    time_step.append(int(row[0]))
```

```
import csv
time_step = []
sunspots = []

with open('/tmp/sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    sunspots.append(float(row[2]))
    time_step.append(int(row[0]))
```

Sunspots.csv

```
1   ,Date,Monthly Mean Total Sunspot Number
2   0,1749-01-31,96.7
3   1,1749-02-28,104.3
4   2,1749-03-31,116.7
5   3,1749-04-30,92.8
6   4,1749-05-31,141.7
7   5,1749-06-30,139.2
8   6,1749-07-31,158.0
9   7,1749-08-31,110.5
10  8,1749-09-30,126.5
11  9,1749-10-31,125.8
12  10,1749-11-30,264.3
13  11,1749-12-31,142.0
14  12,1750-01-31,122.2
15  13,1750-02-28,126.5
16  14,1750-03-31,148.7
17  15,1750-04-30,147.2
18  16,1750-05-31,150.0
19  17,1750-06-30,166.7
```

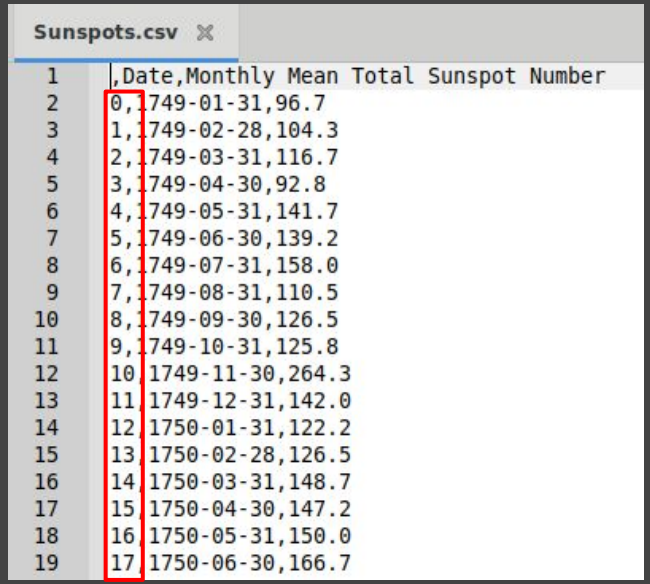
```
import csv
time_step = []
sunspots = []

with open('/tmp/sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    sunspots.append(float(row[2]))
    time_step.append(int(row[0]))
```



Sunspots.csv

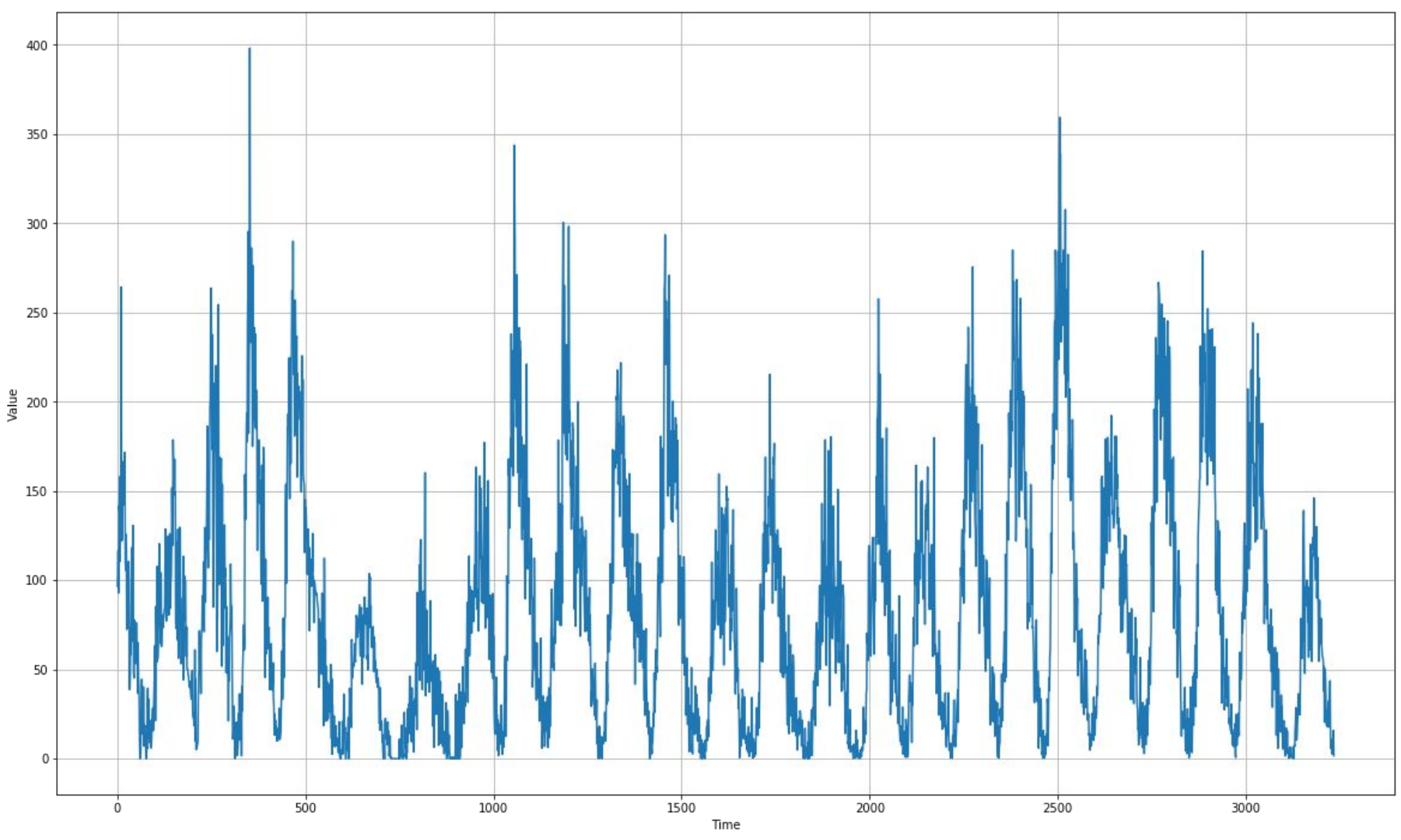
1	,Date,Monthly Mean Total Sunspot Number
2	0,1749-01-31,96.7
3	1,1749-02-28,104.3
4	2,1749-03-31,116.7
5	3,1749-04-30,92.8
6	4,1749-05-31,141.7
7	5,1749-06-30,139.2
8	6,1749-07-31,158.0
9	7,1749-08-31,110.5
10	8,1749-09-30,126.5
11	9,1749-10-31,125.8
12	10,1749-11-30,264.3
13	11,1749-12-31,142.0
14	12,1750-01-31,122.2
15	13,1750-02-28,126.5
16	14,1750-03-31,148.7
17	15,1750-04-30,147.2
18	16,1750-05-31,150.0
19	17,1750-06-30,166.7

```
import csv
time_step = []
sunspots = []

with open('/tmp/sunspots.csv') as csvfile:
  reader = csv.reader(csvfile, delimiter=',')
  next(reader)
  for row in reader:
    sunspots.append(float(row[2]))
    time_step.append(int(row[0]))
```


Sunspots.csv

1	,Date,Monthly Mean Total Sunspot Number
2	0,1749-01-31,96.7
3	1,1749-02-28,104.3
4	2,1749-03-31,116.7
5	3,1749-04-30,92.8
6	4,1749-05-31,141.7
7	5,1749-06-30,139.2
8	6,1749-07-31,158.0
9	7,1749-08-31,110.5
10	8,1749-09-30,126.5
11	9,1749-10-31,125.8
12	10,1749-11-30,264.3
13	11,1749-12-31,142.0
14	12,1750-01-31,122.2
15	13,1750-02-28,126.5
16	14,1750-03-31,148.7
17	15,1750-04-30,147.2
18	16,1750-05-31,150.0
19	17,1750-06-30,166.7

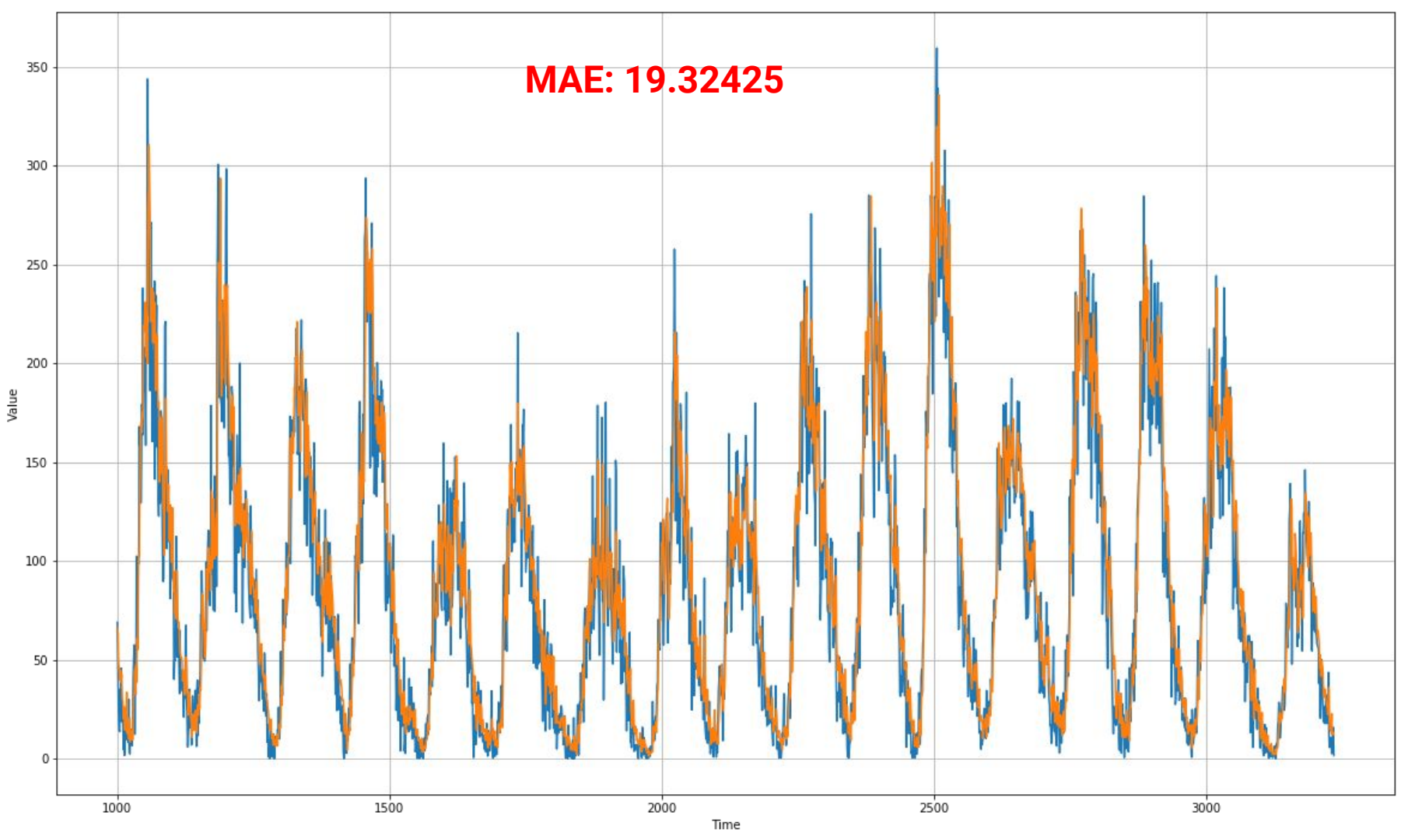
```
series = np.array(sunspots)
time = np.array(time_step)
```

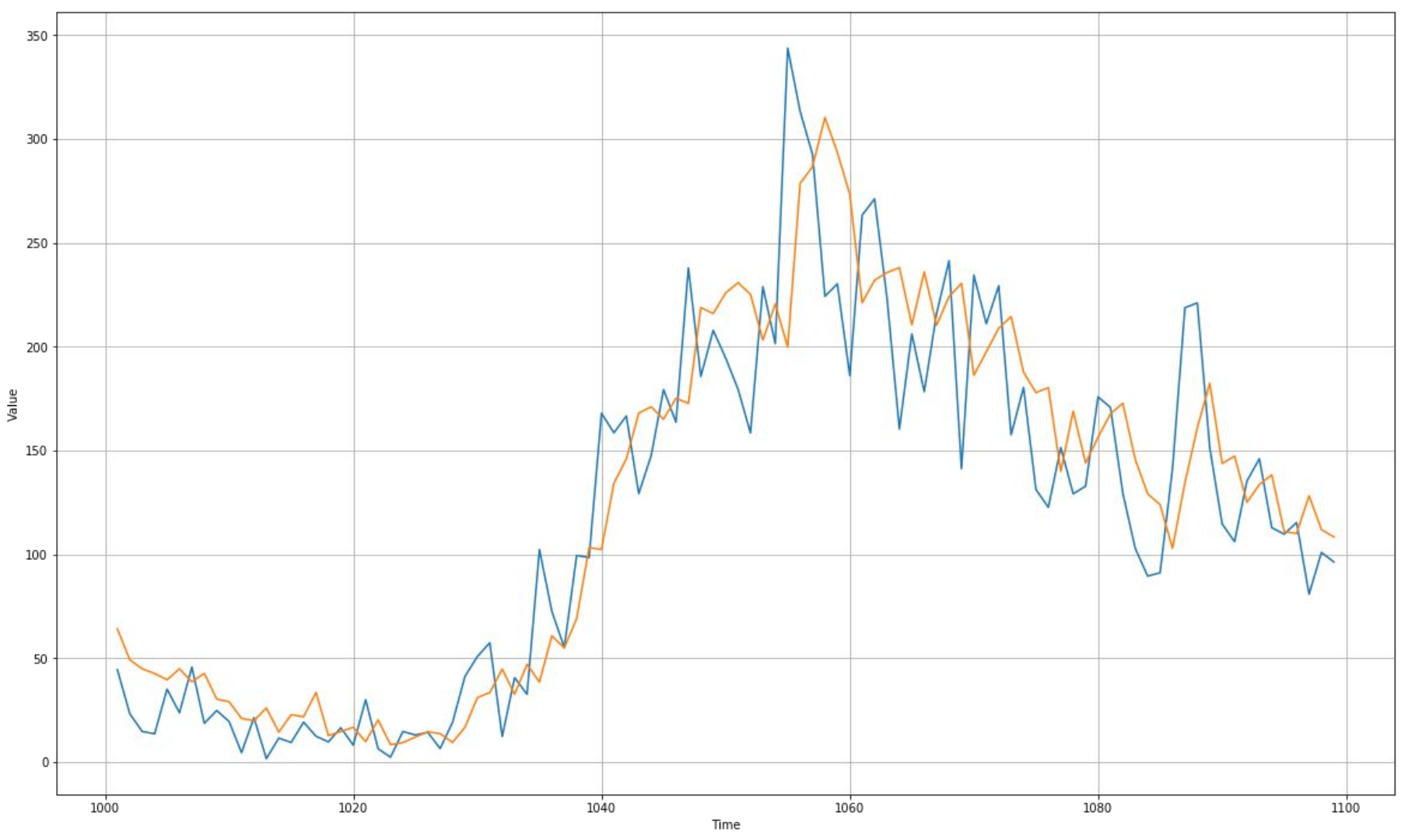

Value
Time
400
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 20
batch_size = 32
shuffle_buffer_size = 1000
```

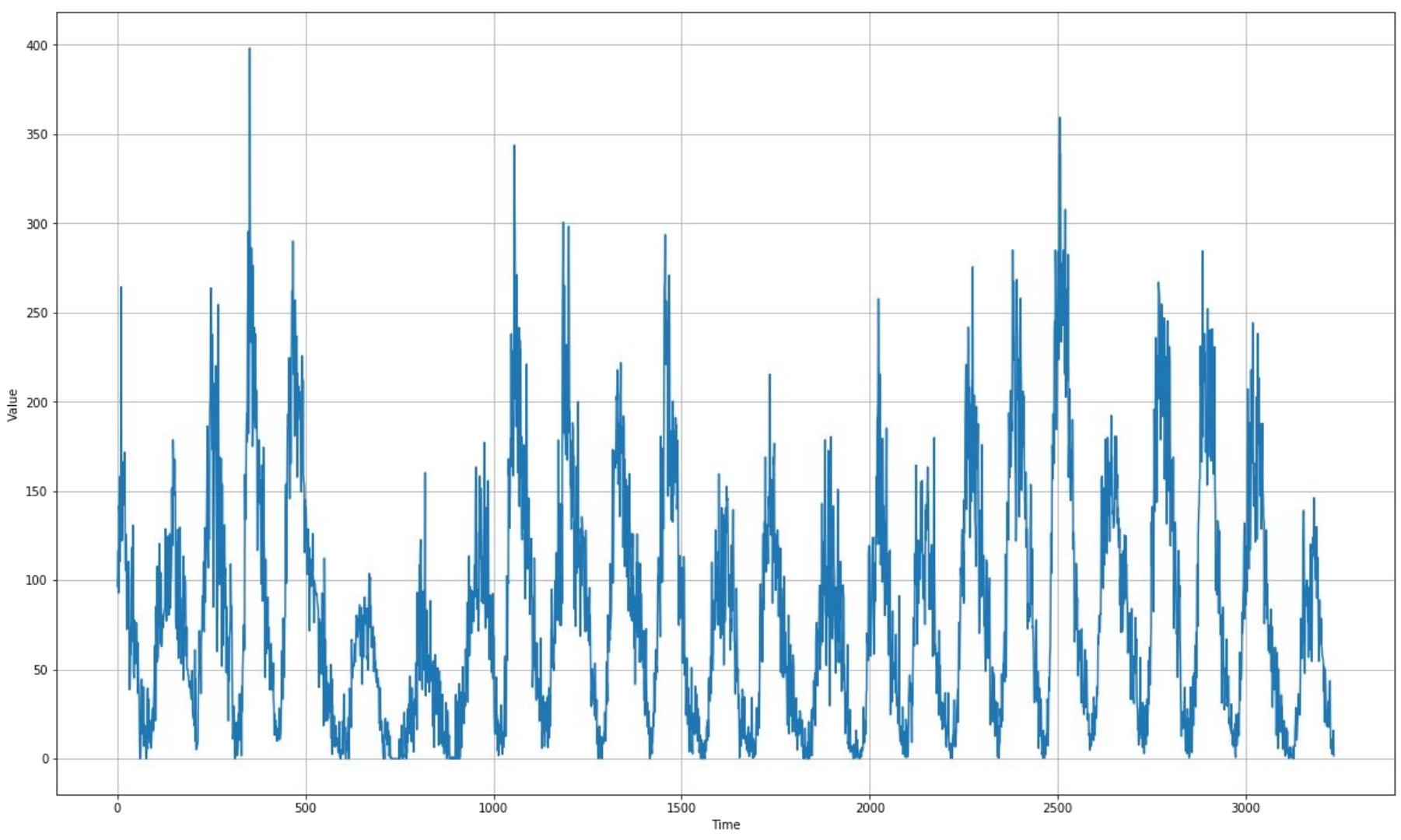
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer)
                   .map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset
```

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

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

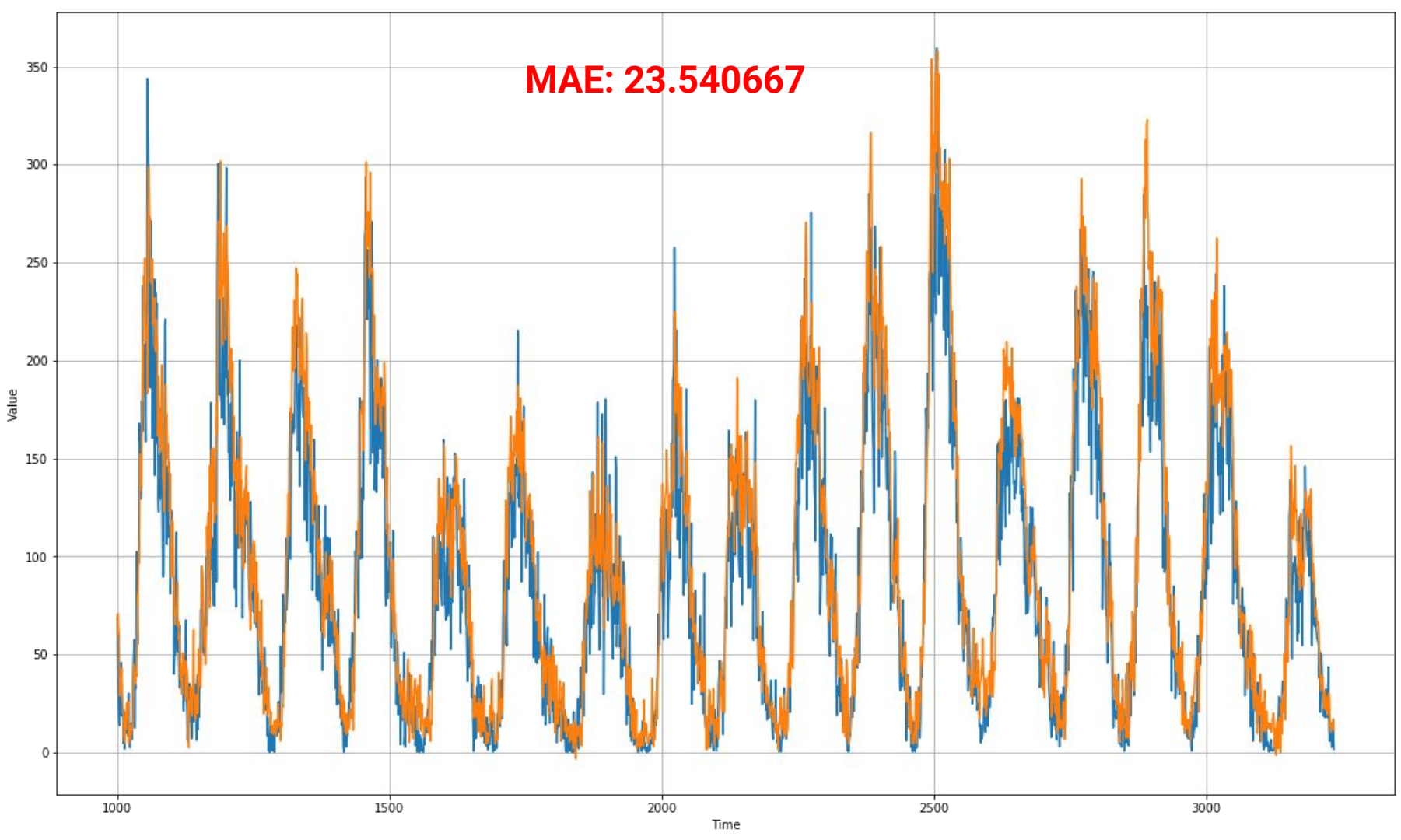

MAE: 19.32425
Value
Time
0
50
100
150
200
250
300
350
1000
1500
2000
2500
3000

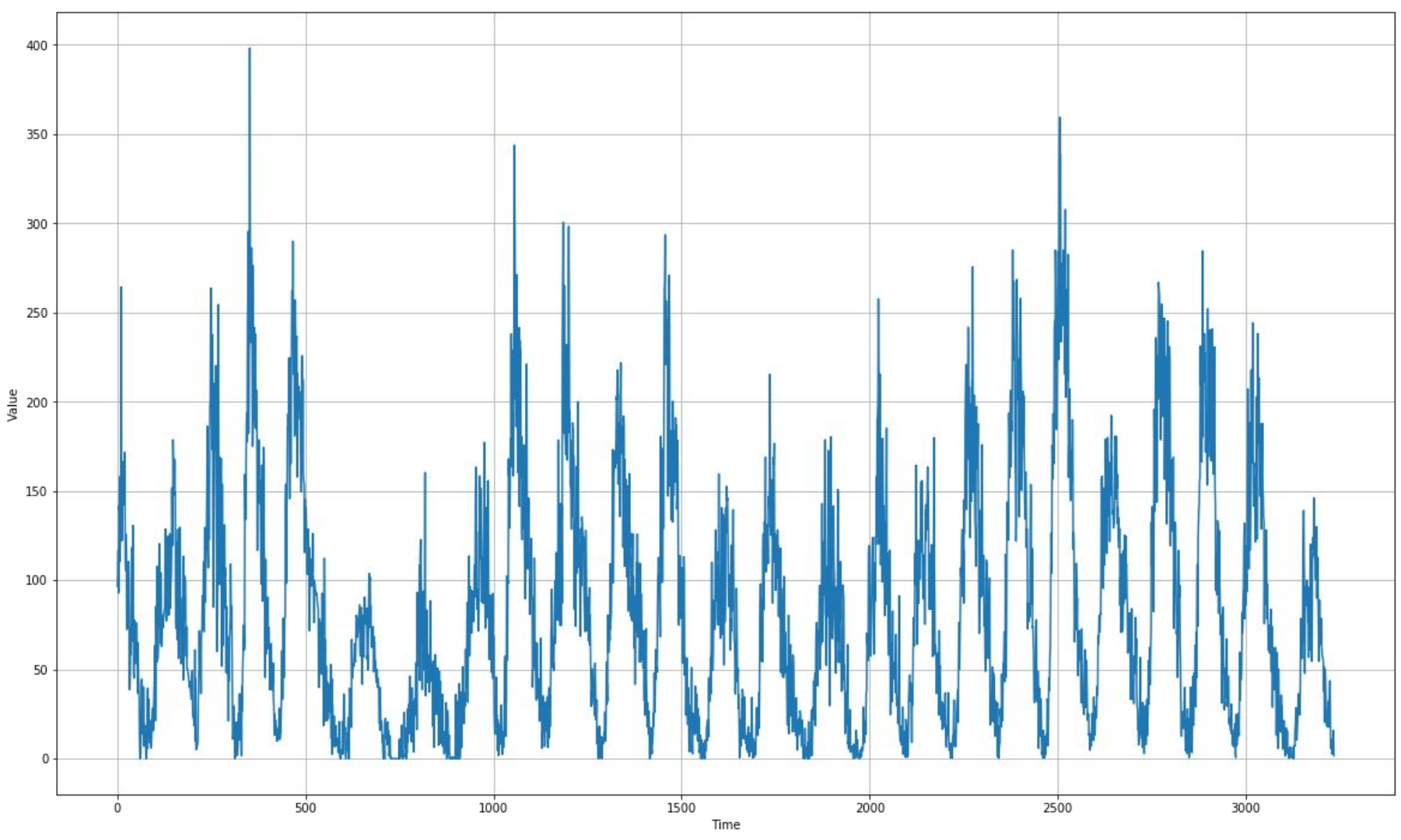

350
300
250
200
150
100
50
0
Value
1000
1020
1040
1060
1080
1100
Time

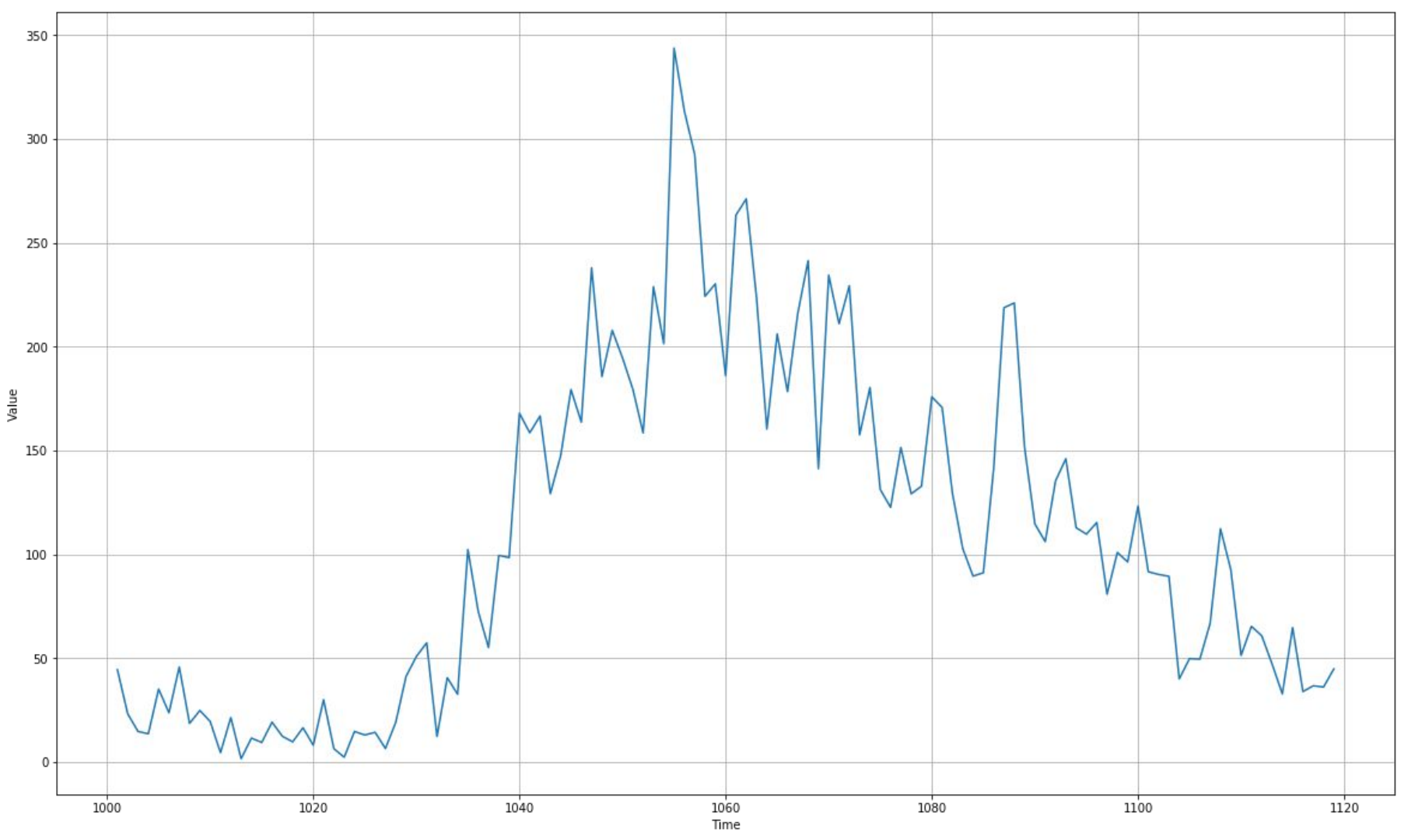
```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 20
batch_size = 32
shuffle_buffer_size = 1000
```


Value
Time
400
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 132
batch_size = 32
shuffle_buffer_size = 1000
```


MAE: 23.540667
350
300
250
200
150
100
50
0
Value
1000
1500
2000
2500
3000
Time

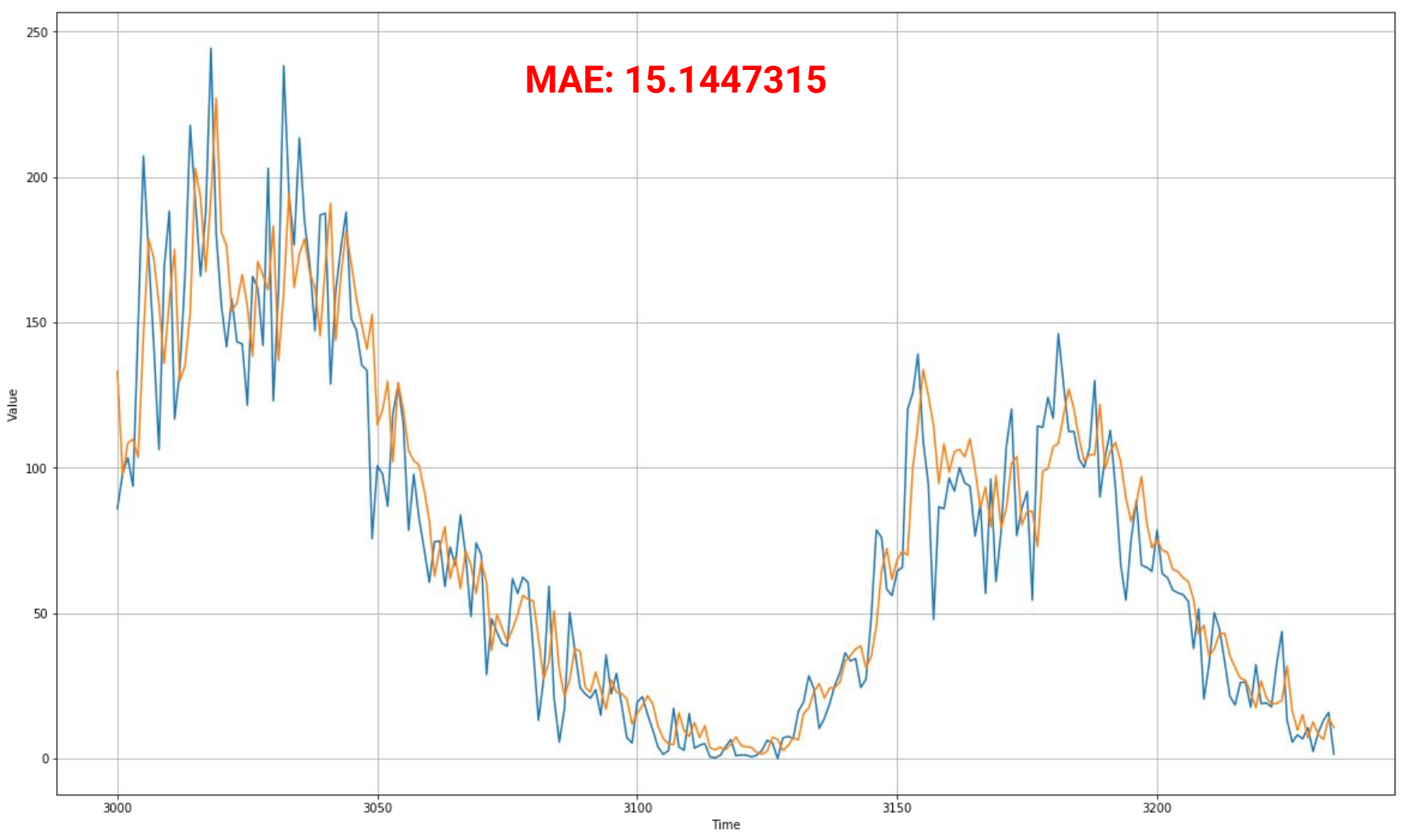

Value
Time
400
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000


350
300
250
200
150
100
50
0
Value
1000
1020
1040
1060
1080
1100
1120
Time

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

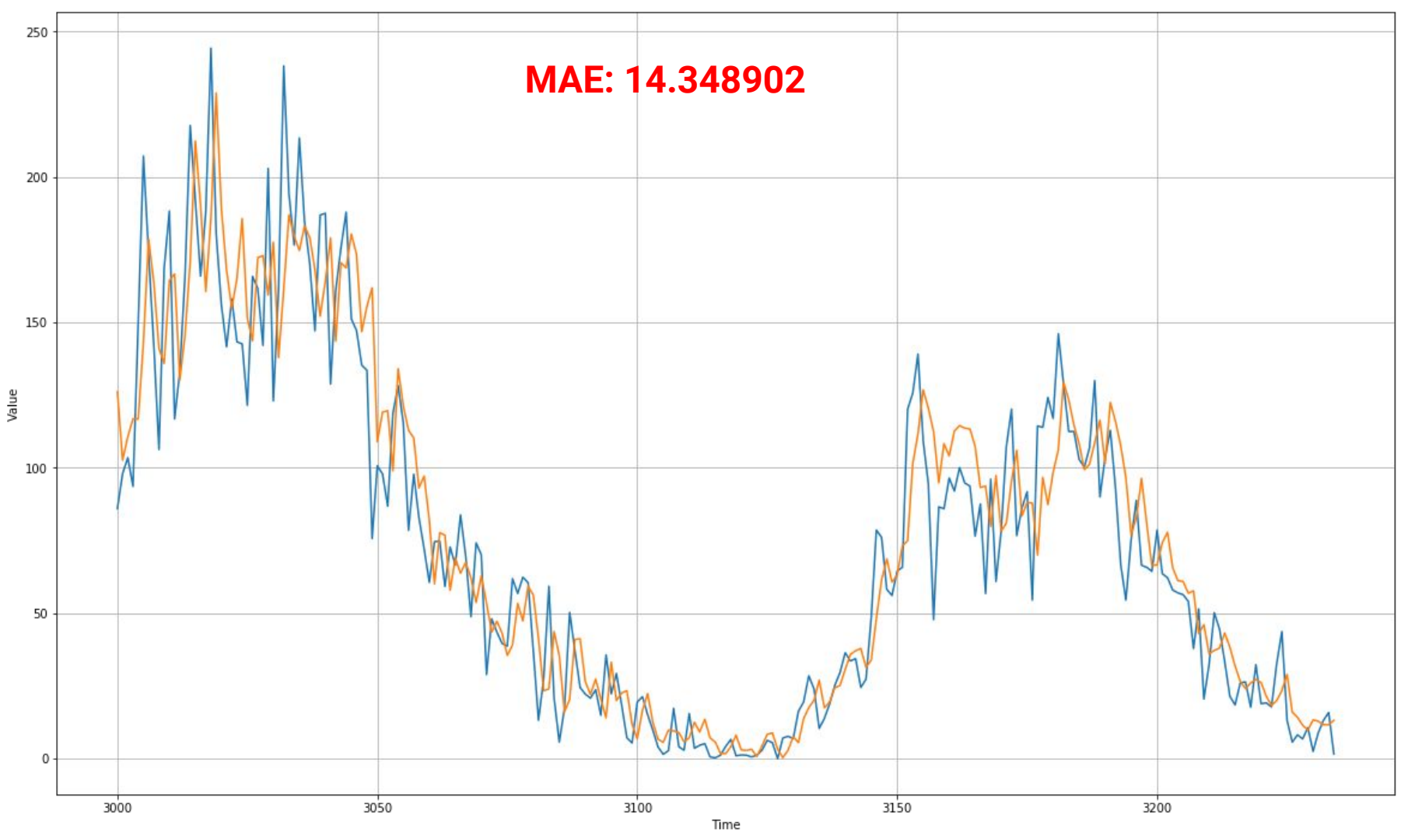
```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

```
split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```


MAE: 15.1447315
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

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

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

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

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(30, input_shape=[window_size], activation="relu"),
    tf.keras.layers.Dense(15, activation="relu"),
    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,verbose=0)
```


MAE: 14.348902
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

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

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```

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

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, 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(learning_rate=1e-7,
momentum=0.9))
model.fit(dataset,epochs=100,verbose=0)
```


MAE: 14.122419
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

```
model.predict(series[3205:3235][np.newaxis])
```

```
model.predict(series[3205:3235][np.newaxis])
```

```
7.0773993
```

https://www.sws.bom.gov.au/Solar/1/6

(last updated 01 Jun 2019 09:42 UT)

OBSERVED MONTHLY SUNSPOT NUMBERS

2001	142.6	121.5	165.8	161.7	142.1	202.9	123.0	161.5	238.2	194.1	176.6	213.4
2002	184.6	170.2	147.1	186.9	187.5	128.8	161.0	175.6	187.9	151.2	147.2	135.3
2003	133.5	75.7	100.7	97.9	86.8	118.7	128.3	115.4	78.5	97.8	82.9	72.2
2004	60.6	74.6	74.8	59.2	72.8	66.5	83.8	69.7	48.8	74.2	70.1	28.9
2005	48.1	43.5	39.6	38.7	61.9	56.8	62.4	60.5	37.2	13.2	27.5	59.3
2006	20.9	5.7	17.3	50.3	37.2	24.5	22.2	20.8	23.7	14.9	35.7	22.3
2007	29.3	18.4	7.2	5.4	19.5	21.3	15.1	9.8	4.0	1.5	2.8	17.3
2008	4.1	2.9	15.5	3.6	4.6	5.2	0.6	0.3	1.2	4.2	6.6	1.0
2009	1.3	1.2	0.6	1.2	2.9	6.3	5.5	0.0	7.1	7.7	6.9	16.3
2010	19.5	28.5	24.0	10.4	13.9	18.8	25.2	29.6	36.4	33.6	34.4	24.5
2011	27.3	48.3	78.6	76.1	58.2	56.1	64.5	65.8	120.1	125.7	139.1	109.3
2012	94.4	47.8	86.6	85.9	96.5	92.0	100.1	94.8	93.7	76.5	87.6	56.8
2013	96.1	60.9	78.3	107.3	120.2	76.7	86.2	91.8	54.5	114.4	113.9	124.2
2014	117.0	146.1	128.7	112.5	112.5	102.9	100.2	106.9	130.0	90.0	103.6	112.9
2015	93.0	66.7	54.5	75.3	88.8	66.5	65.8	64.4	78.6	63.6	62.2	58.0
2016	57.0	56.4	54.1	37.9	51.5	20.5	32.4	50.2	44.6	33.4	21.4	18.5
2017	26.1	26.4	17.7	32.3	18.9	19.2	17.8	32.6	43.7	13.2	5.7	8.2
2018	6.8	10.7	2.5	8.9	13.1	15.6	1.6	8.7	3.3	4.9	4.9	3.1
2019	7.8	0.8	9.5	9.1	10.1							

```
split_time = 3000
window_size = 60

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(20, 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-7, momentum=0.9))
```

```
window_size = 60
batch_size = 64
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  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)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

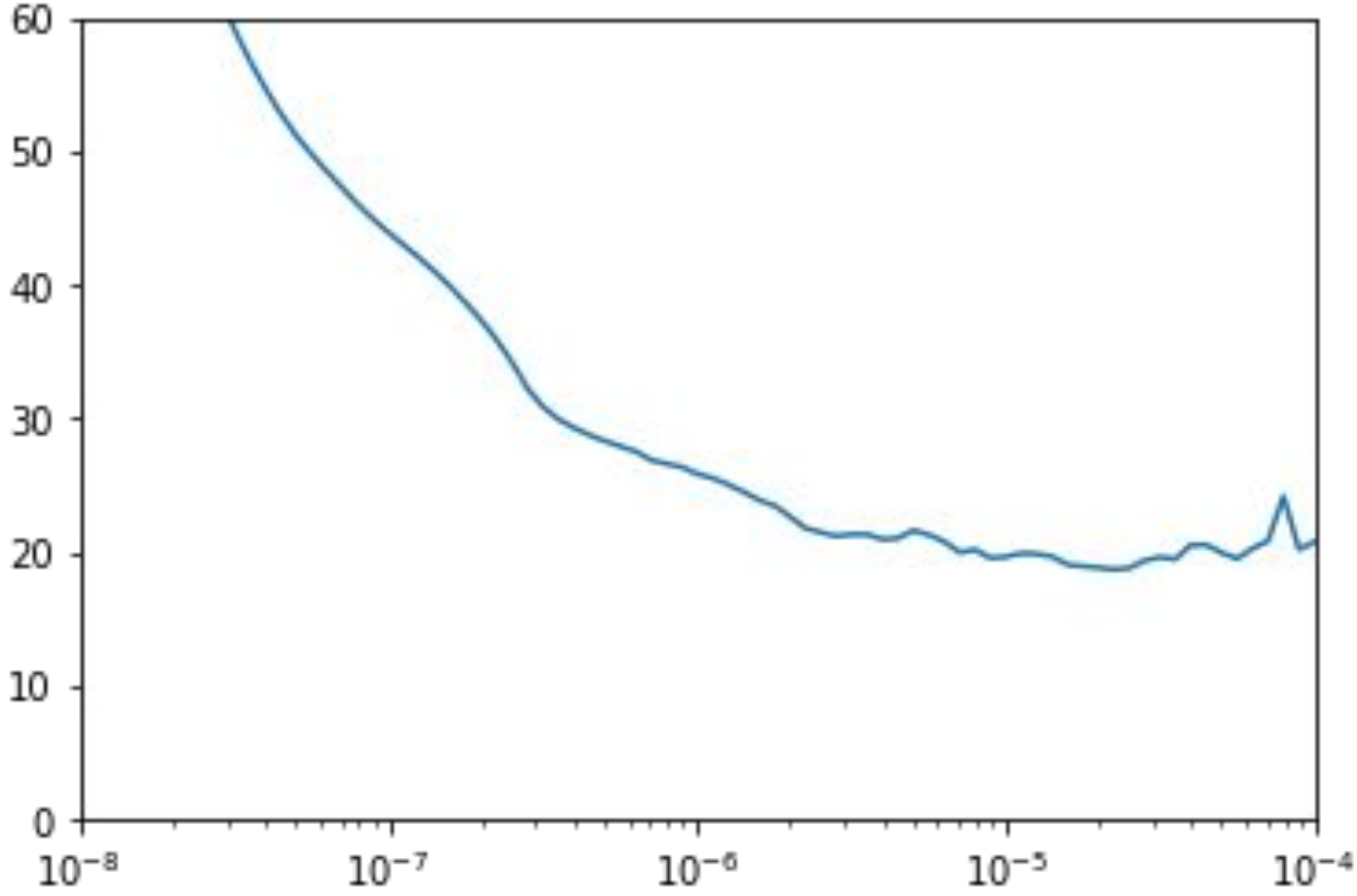
```
window_size = 60
batch_size = 64
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  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)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

```
window_size = 60
batch_size = 64
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  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)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

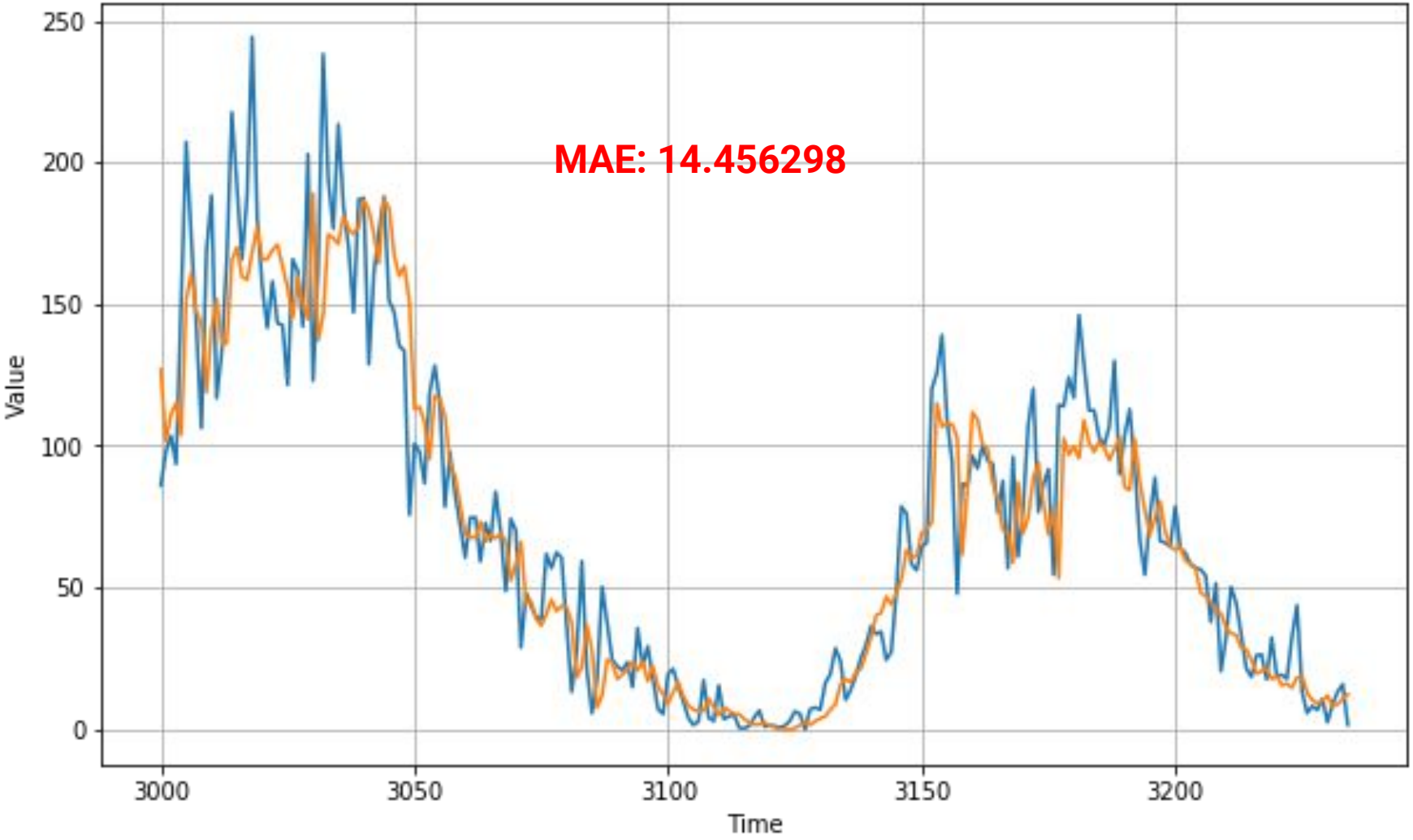
```
window_size = 60
batch_size = 64
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  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)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

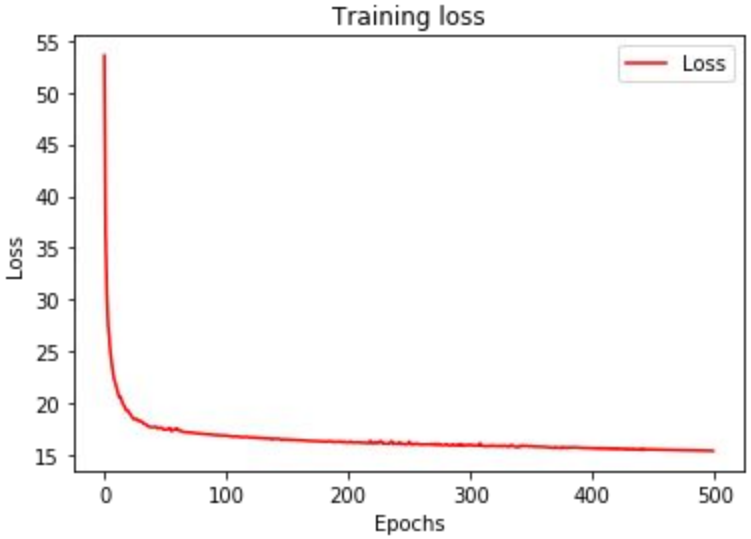
```
window_size = 60
batch_size = 64
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  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)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

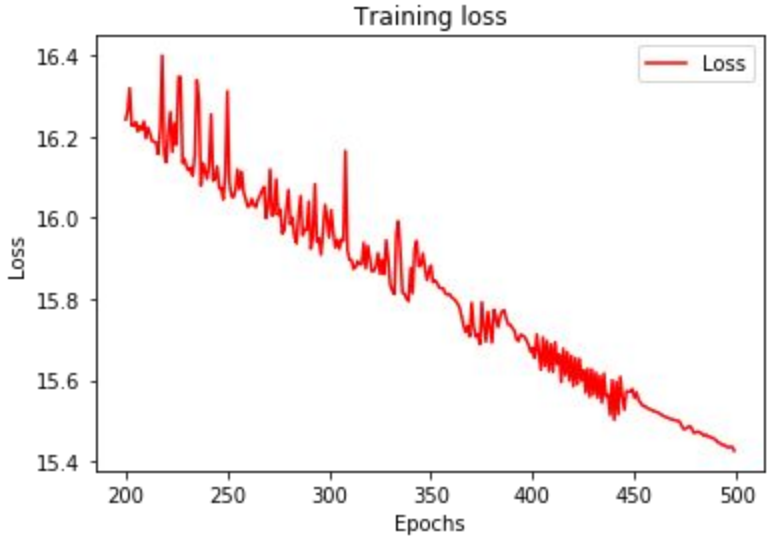
```
window_size = 60
batch_size = 64
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  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)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```


60
50
40
30
20
10
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}

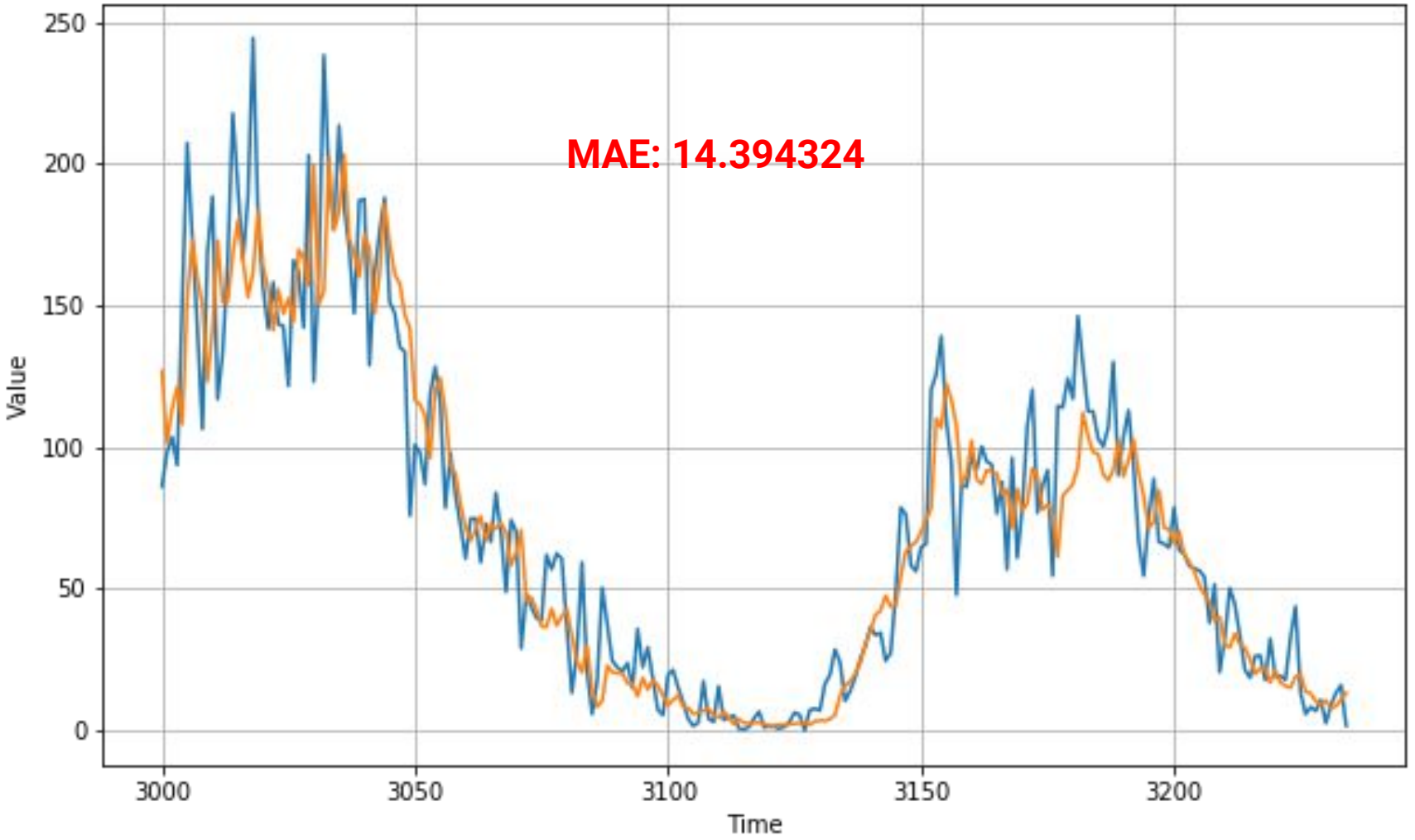
```
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=500)
```

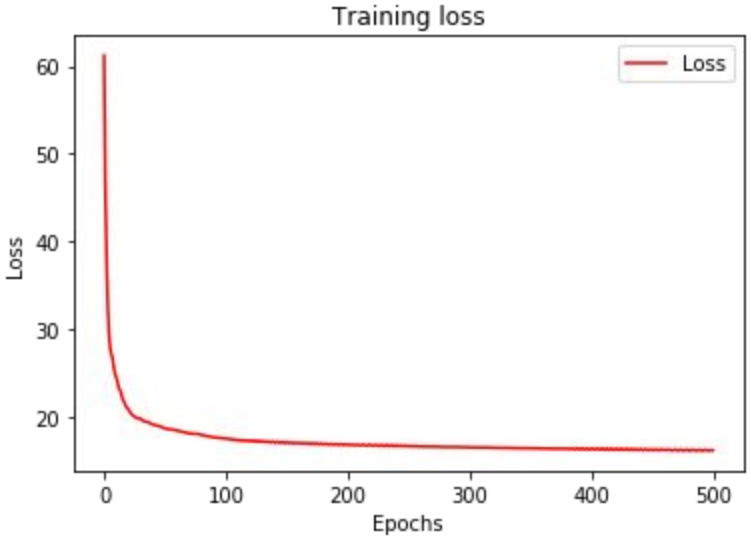

MAE: 14.456298
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

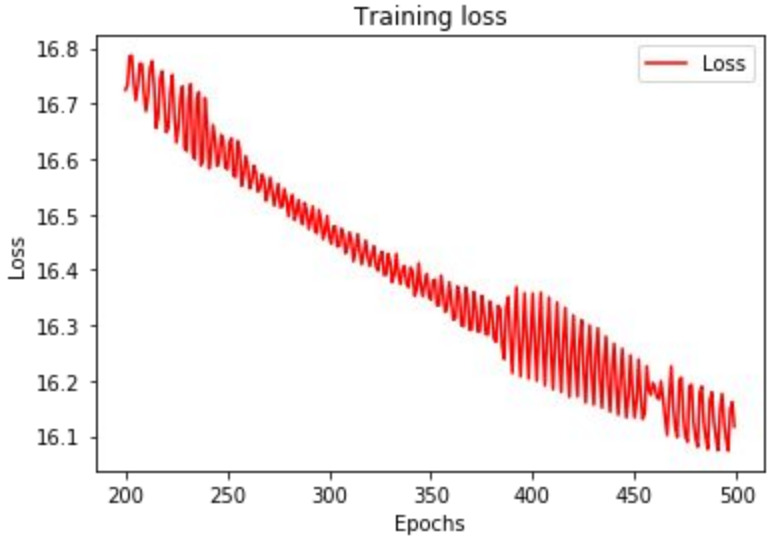

Training loss
Loss
Epochs
55
50
45
40
35
30
25
20
15
0
100
200
300
400
500


Training loss
Loss
Epochs
16.4
16.2
16.0
15.8
15.6
15.4
200
250
300
350
400
450
500

```
train_set = windowed_dataset(x_train, window_size, batch_size=256, shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(32, return_sequences=True),
  tf.keras.layers.LSTM(32),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=500)
```


MAE: 14.394324
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time

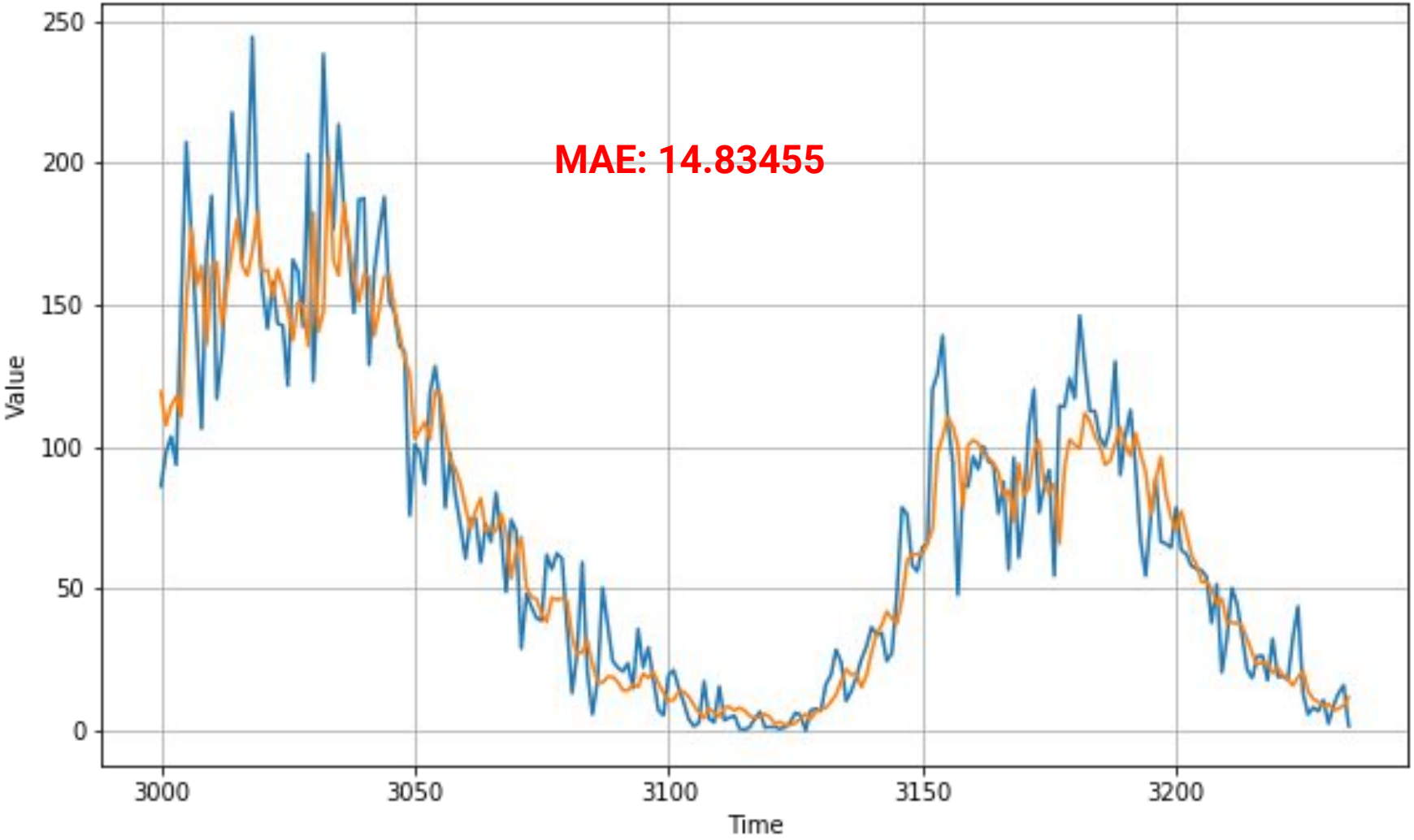

Training loss
Loss
60
50
40
30
20
Loss
0
100
200
300
400
500
Epochs

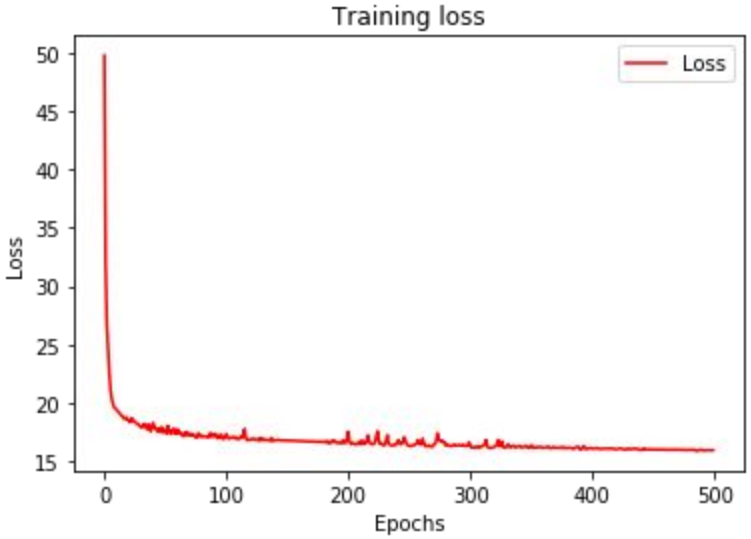

Training loss
Loss
16.8
16.7
16.6
16.5
16.4
16.3
16.2
16.1
Loss
200
250
300
350
400
450
500
Epochs

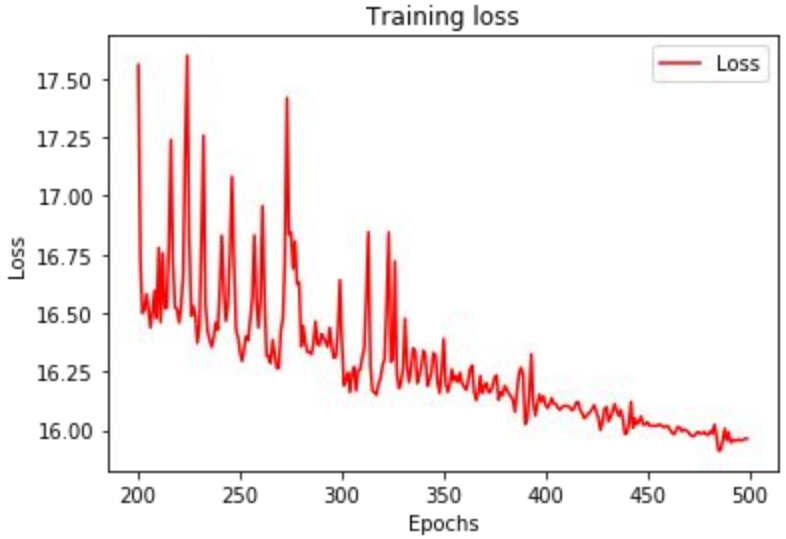
```
train_set = windowed_dataset(x_train, window_size=60, batch_size=250, shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60),
  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(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=500)
```

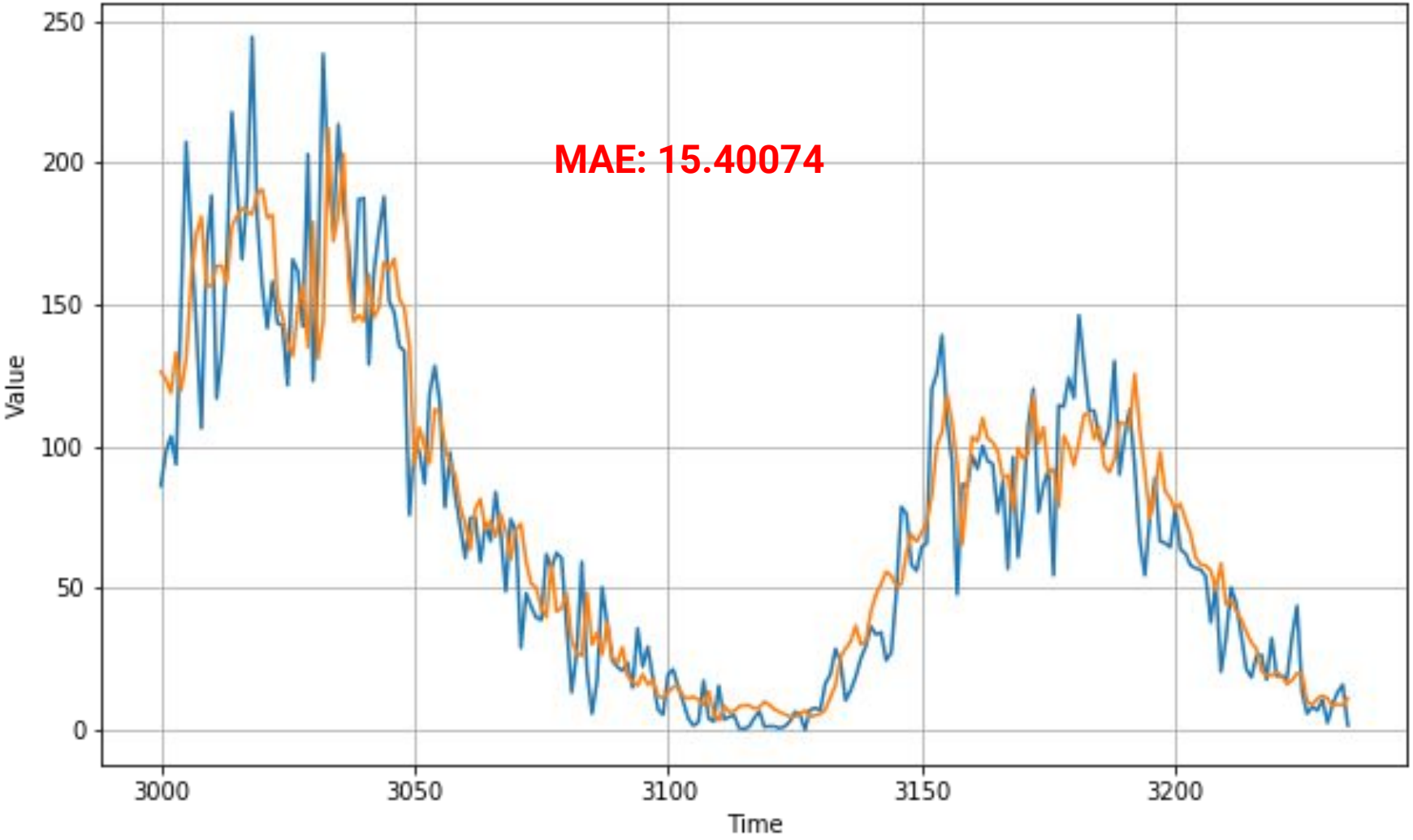
```
train_set = windowed_dataset(x_train, window_size=60, batch_size=250, shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60),
  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(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=500)
```

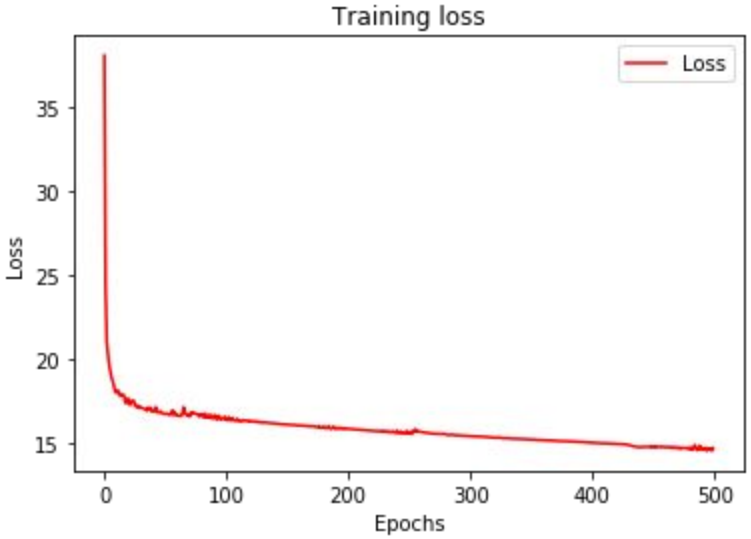
```
train_set = windowed_dataset(x_train, window_size=60, batch_size=250, shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu",
                         input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60),
  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(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=500)
```

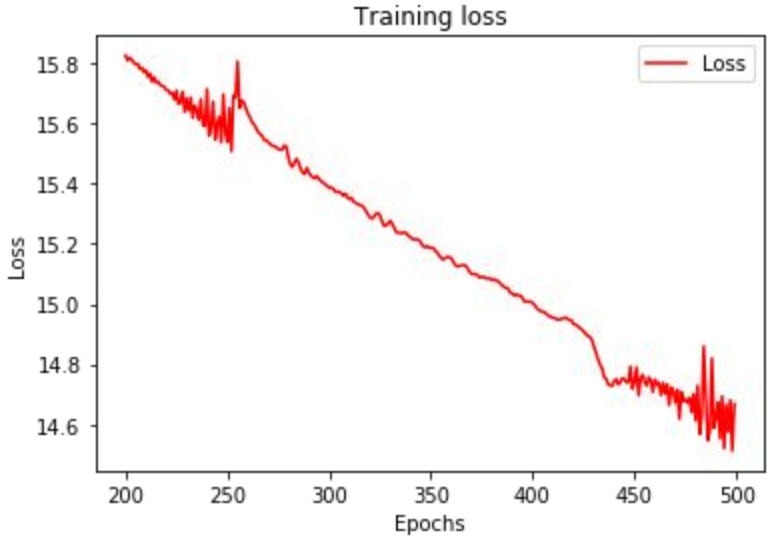

MAE: 14.83455
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time


Training loss
Loss
Loss
Epochs
50
45
40
35
30
25
20
15
0
100
200
300
400
500


Training loss
Loss
Loss
Epochs
17.50
17.25
17.00
16.75
16.50
16.25
16.00
200
250
300
350
400
450
500


MAE: 15.40074
250
200
150
100
50
0
Value
3000
3050
3100
3150
3200
Time


Training loss
Loss
35
30
25
20
15
Loss
0
100
200
300
400
500
Epochs


Training loss
Loss
15.8
15.6
15.4
15.2
15.0
14.8
14.6
Loss
200
250
300
350
400
450
500
Epochs