

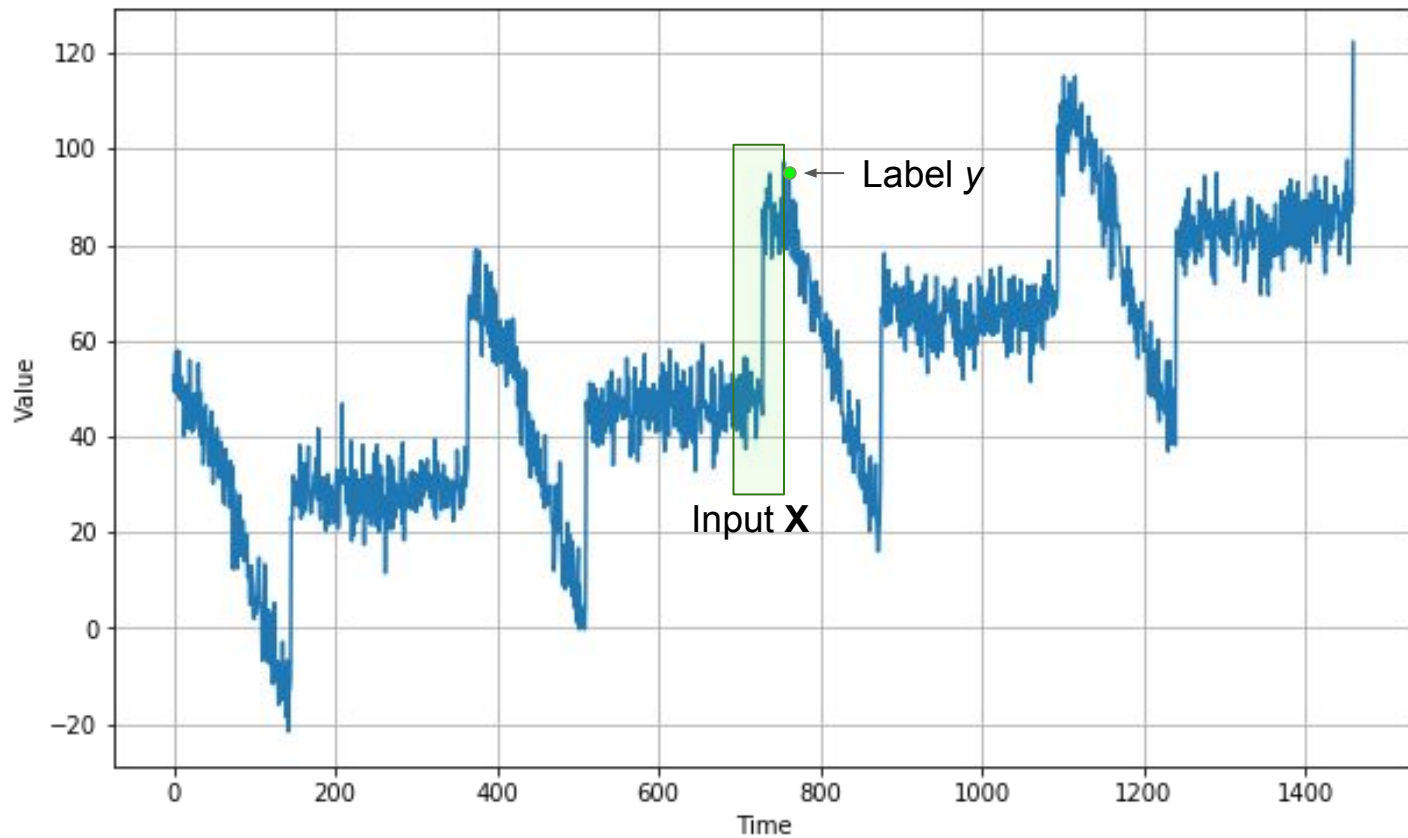
# Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

# Machine Learning on Time Windows



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

```
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())
```

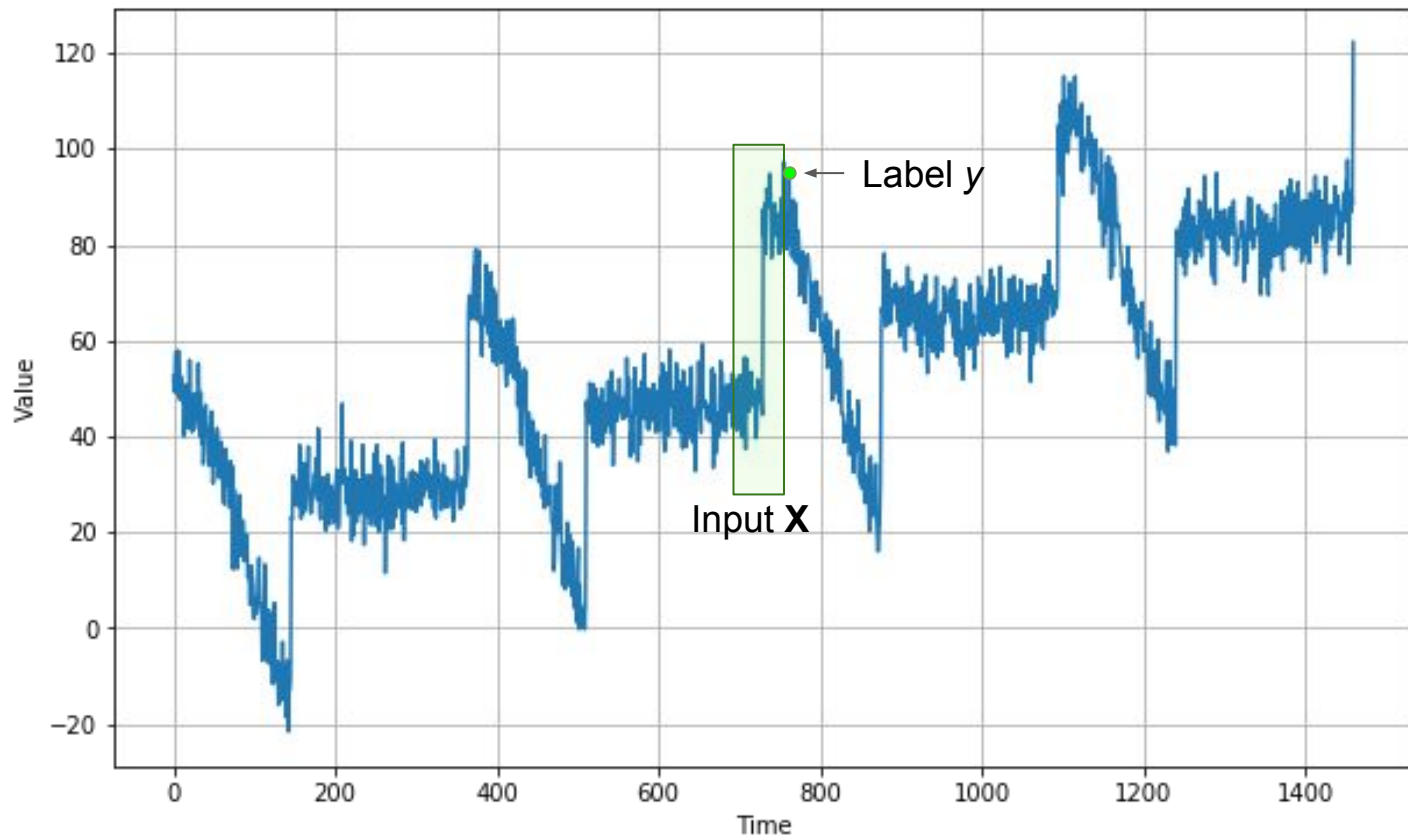
```
x = [[4 5 6 7] [1 2 3 4]]
y = [[8] [5]]
x = [[3 4 5 6] [2 3 4 5]]
y = [[7] [6]]
x = [[5 6 7 8] [0 1 2 3]]
y = [[9] [4]]
```

```
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)  
    dataset = dataset.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)]
```

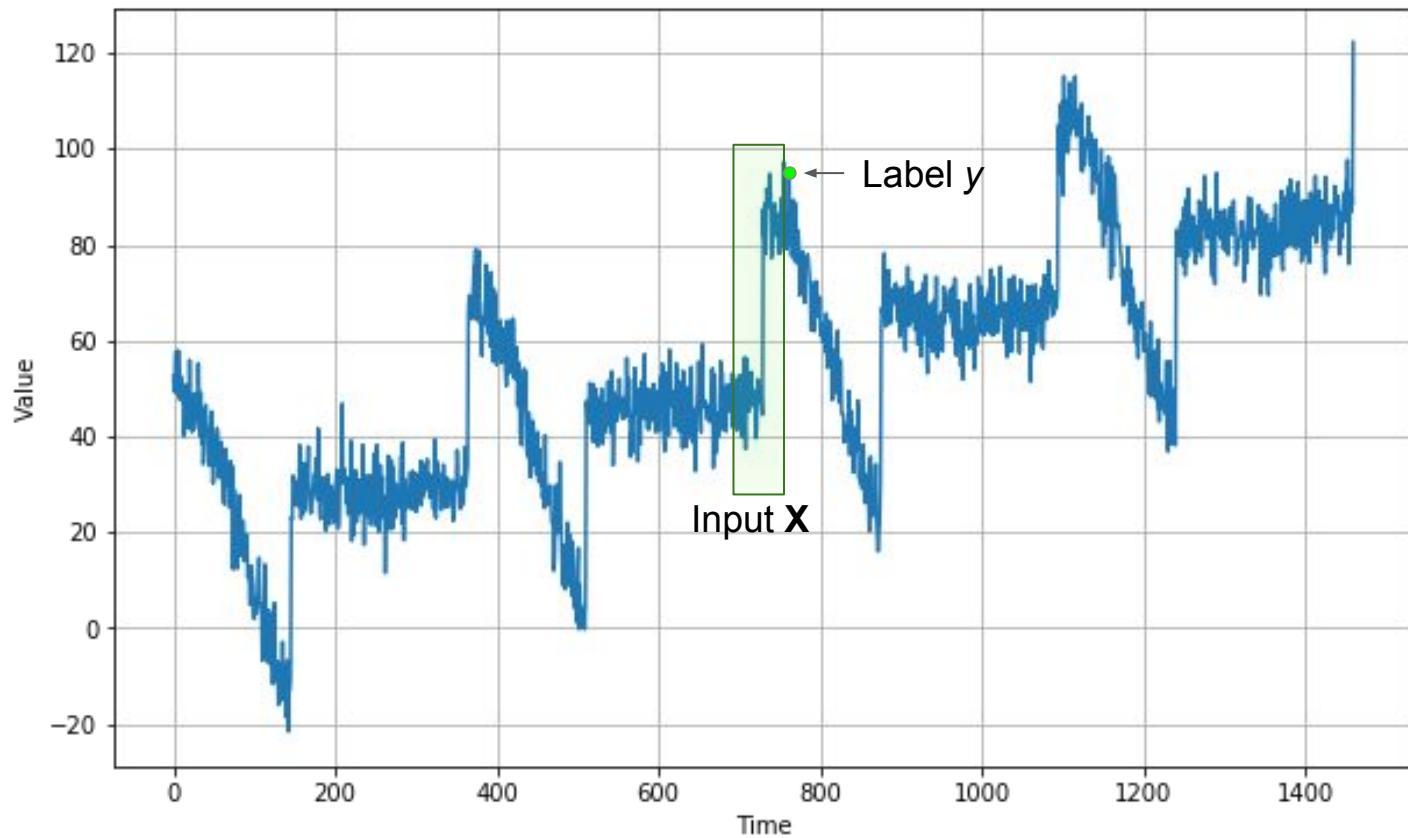
```
print("Layer weights {}".format(l0.get_weights()))
```

```
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)]
```

The diagram illustrates the mapping of weights  $W_t$  to the elements of the layer weights array. Red arrows point from the labels  $W_{t0}$ ,  $W_{t1}$ ,  $W_{t2}$ ,  $W_{t3}$ , and  $b$  to the corresponding elements in the array. The array is a 2D array of shape (20, 1) with dtype float32, followed by a bias term  $b$  of shape (1, 1) with dtype float32.

Label	Value
$W_{t0}$	0.01633573
$W_{t1}$	-0.02911791
$W_{t2}$	0.00845617
$W_{t3}$	-0.02175158
$b$	0.01430958

# 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)]
```

Diagram illustrating the layer weights and bias vector  $b$  for a layer. The weights are represented by the first array, and the bias is represented by the second array.

Weights ( $W_t$ ):

- $W_{t0}$  points to 0.01633573
- $W_{t1}$  points to -0.02911791
- $W_{t2}$  points to 0.00845617
- $W_{t3}$  points to 0.04962169

Bias ( $b$ ):

- $b$  points to 0.01430958



```
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)]
```



Diagram illustrating the layer weights and bias vector. The weights are represented as a list of 20 values, and the bias vector is a single value. Arrows point from the labels  $W_{t0}$ ,  $W_{t1}$ ,  $W_{t2}$ , and  $W_{t3}$  to their corresponding values in the array.

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

$b$



```
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]
```

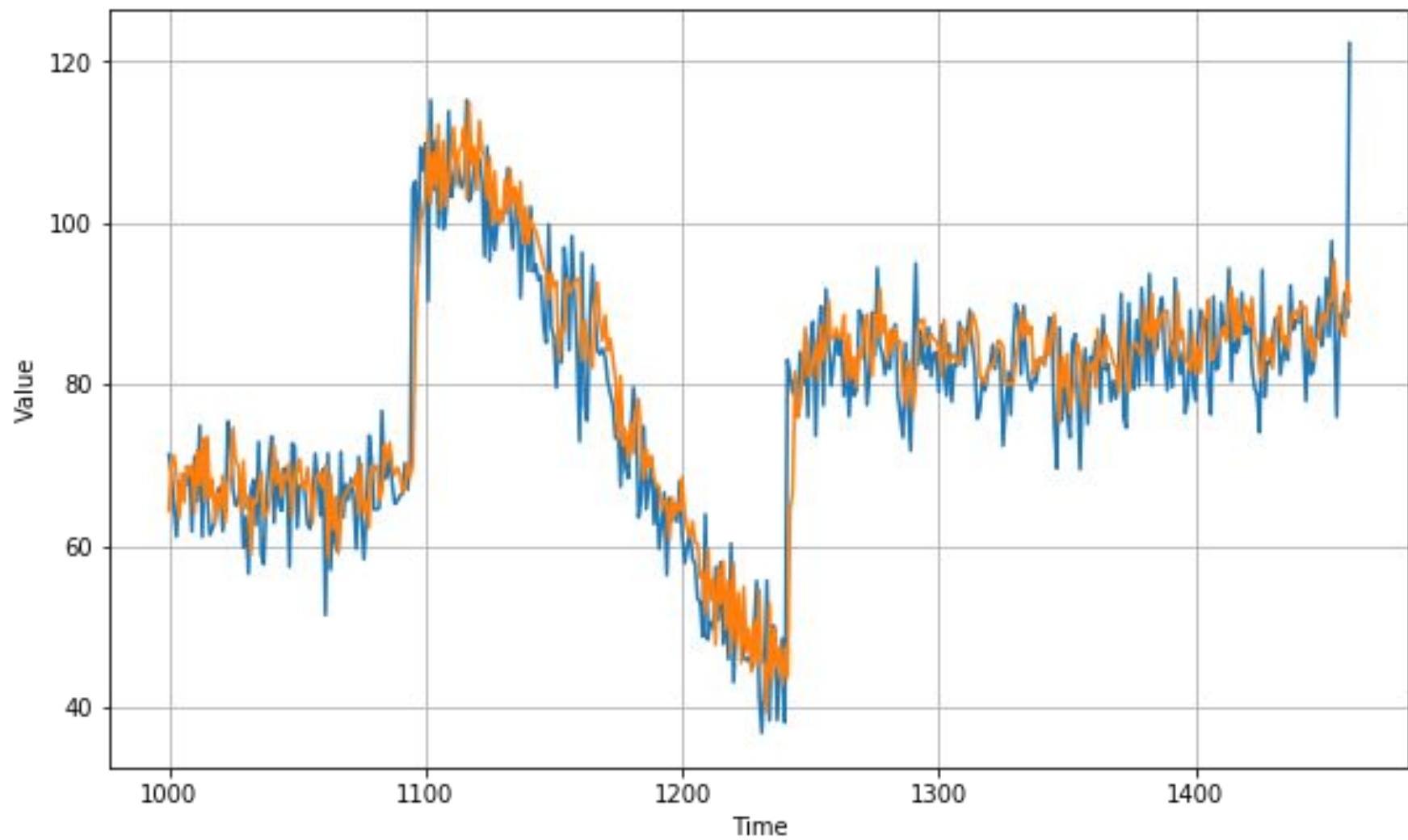
```
forecast = []  
for time in range(len(series) - window_size):  
    forecast.append(model.predict(series[time:time + window_size][np.newaxis]))  
  
forecast = forecast[split_time-window_size:]  
results = np.array(forecast)[: , 0, 0]
```

```
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]
```





```
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))
```

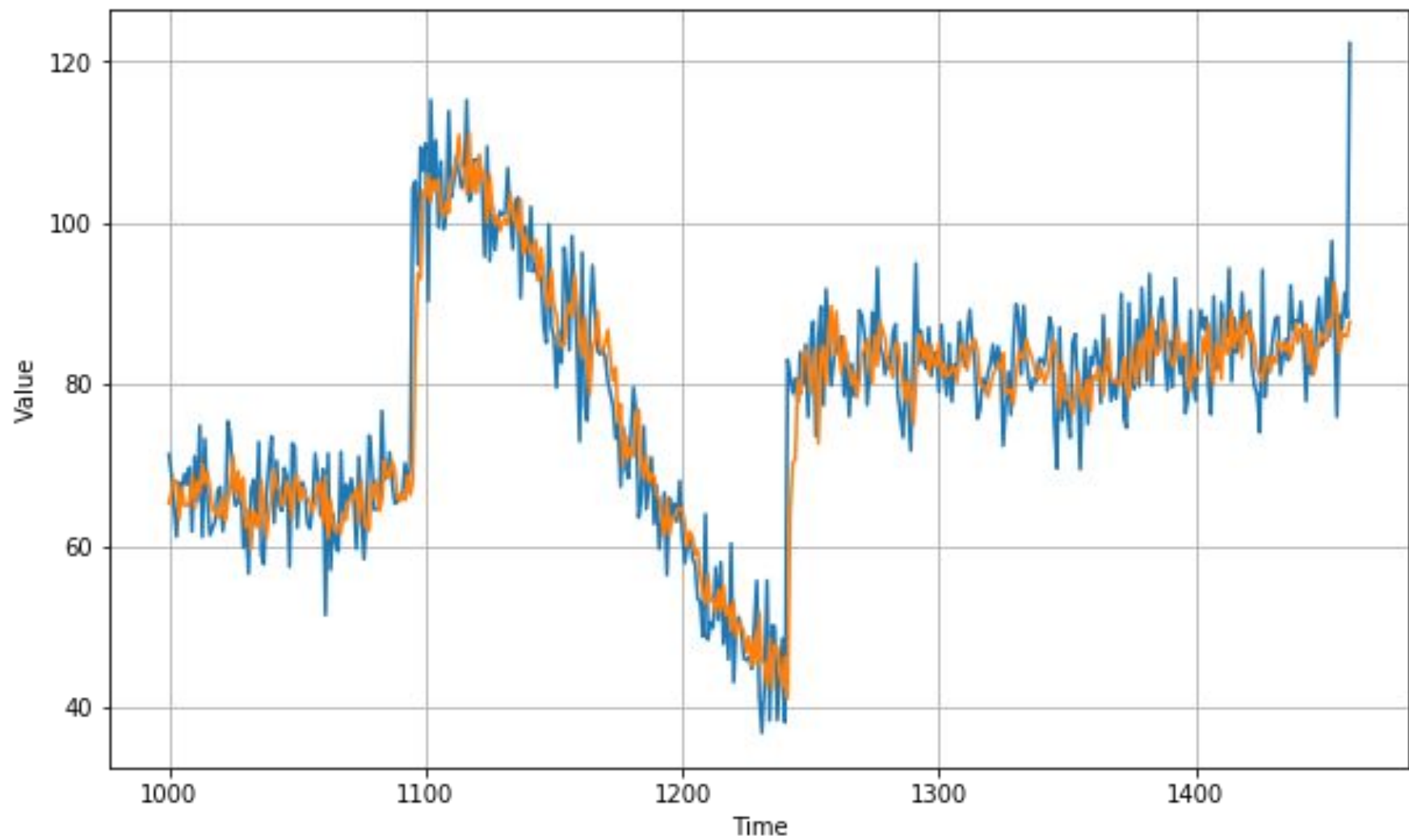
```
model.fit(dataset, epochs=100, verbose=0)
```

```
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)
```





```
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)  
])
```

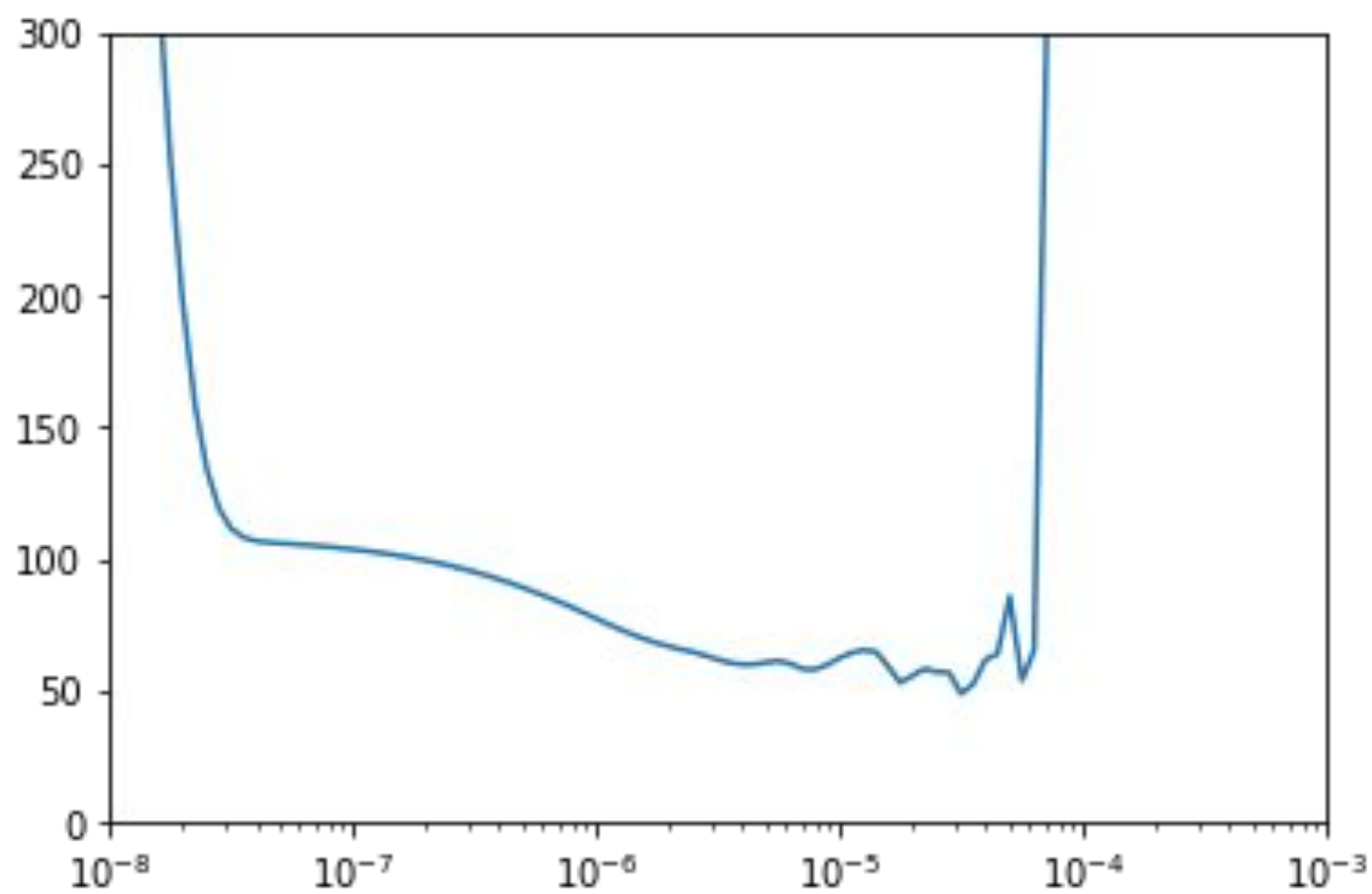
```
lr_schedule = tf.keras.callbacks.LearningRateScheduler(  
    lambda epoch: 1e-8 * 10**(epoch / 20))
```

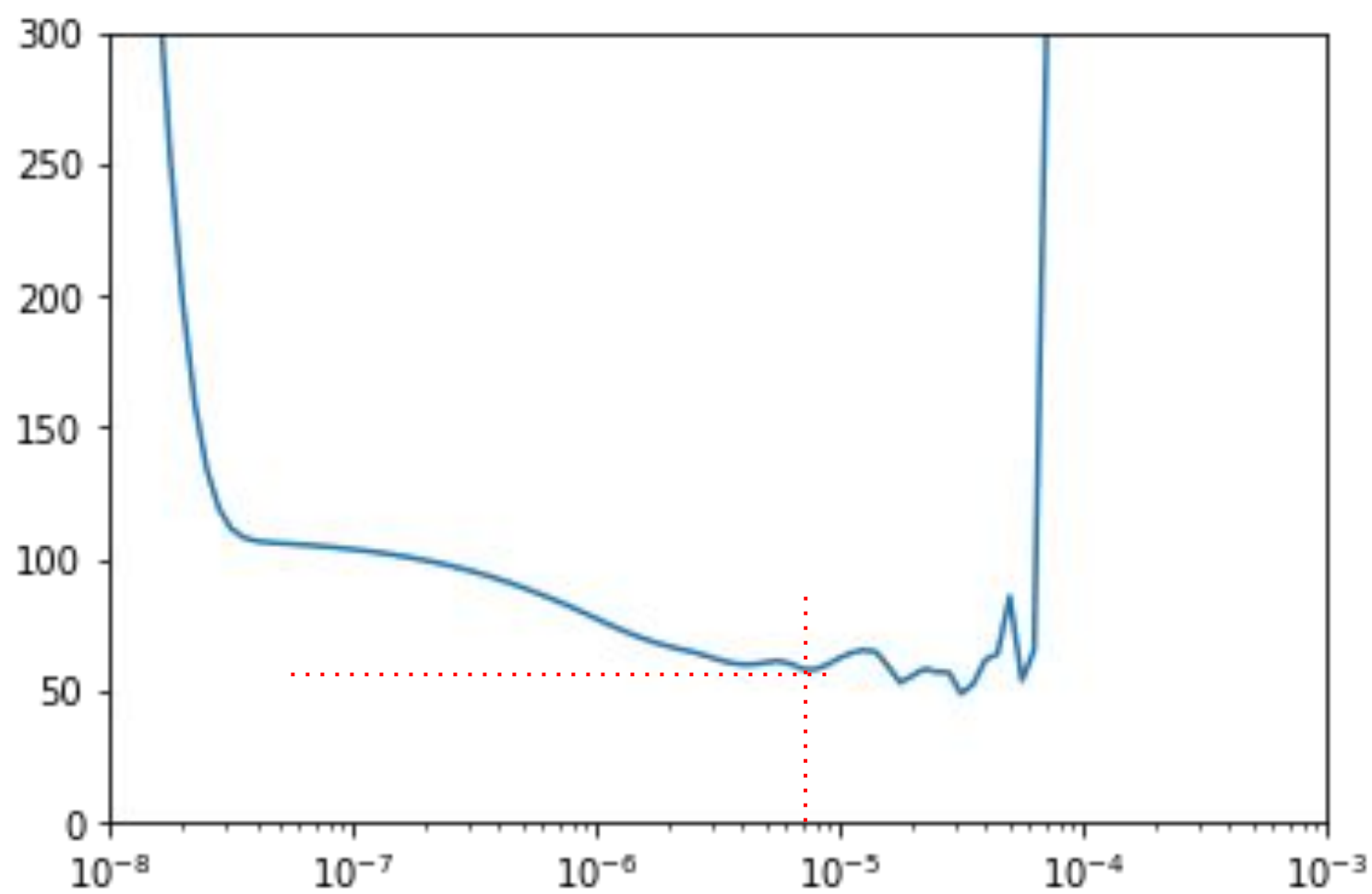
```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-8, momentum=0.9)
```

```
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])
```





```
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)
```

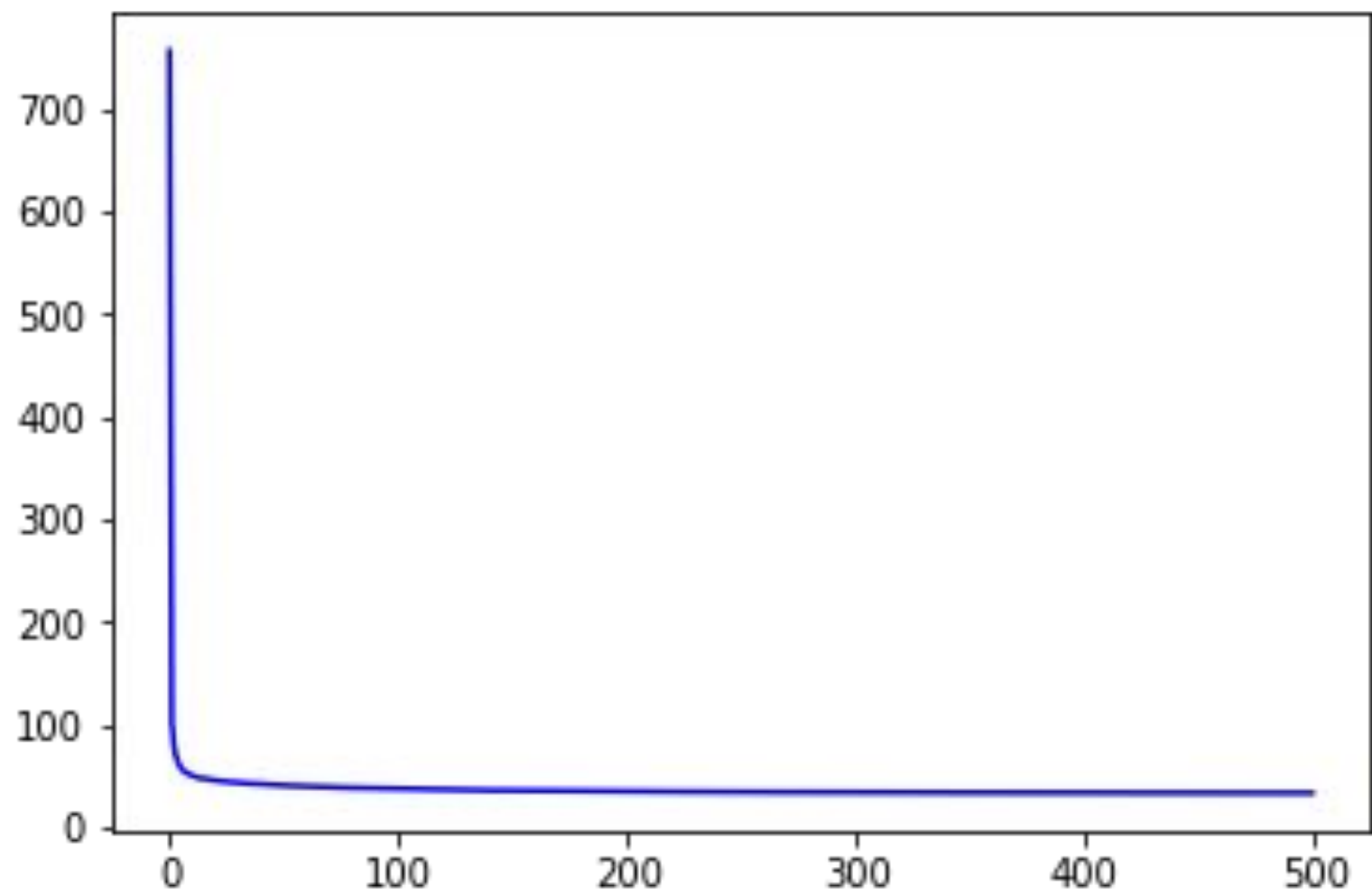


```
window_size = 30
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

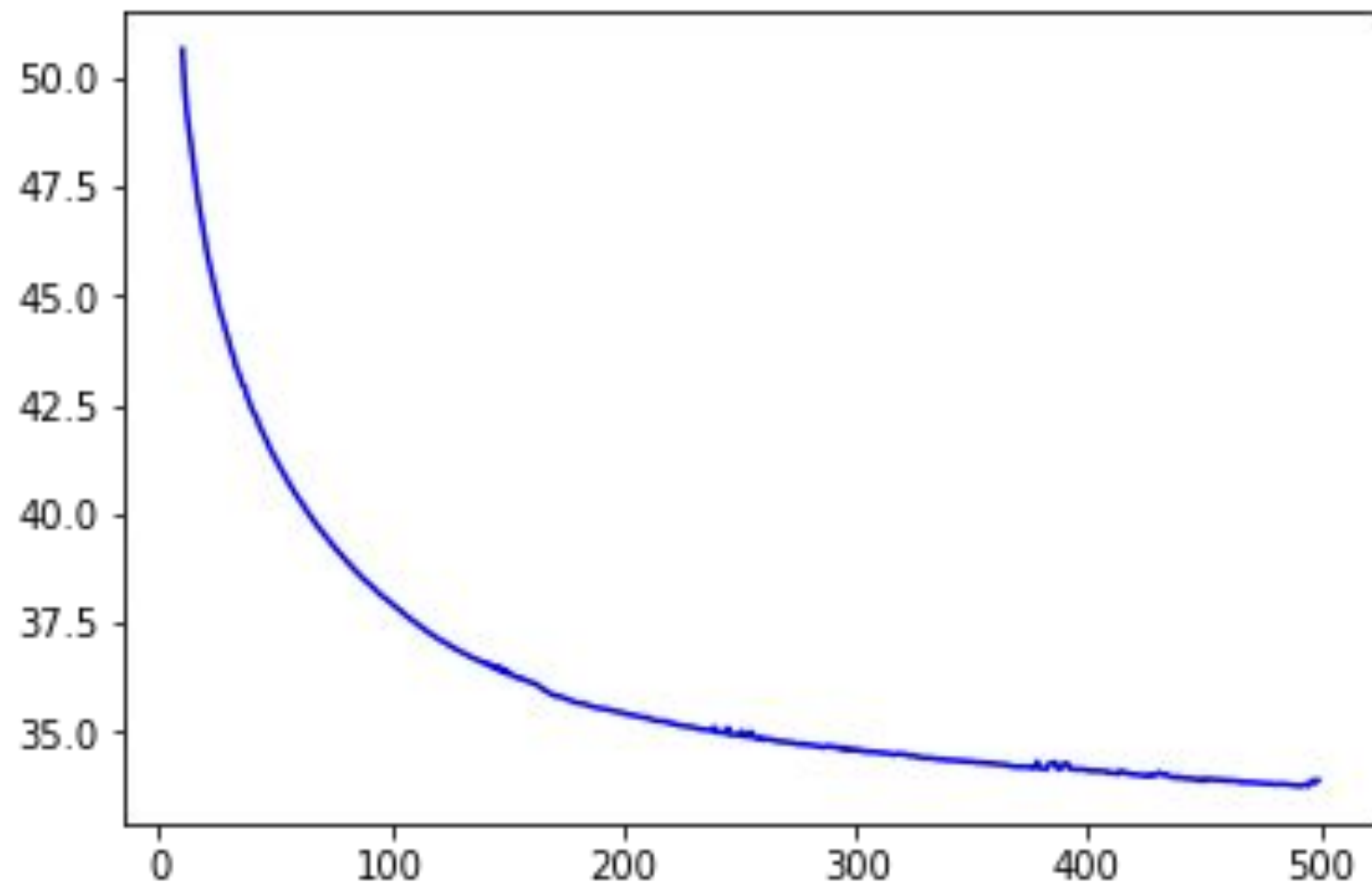
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, activation="relu", input_shape=[window_size]),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])

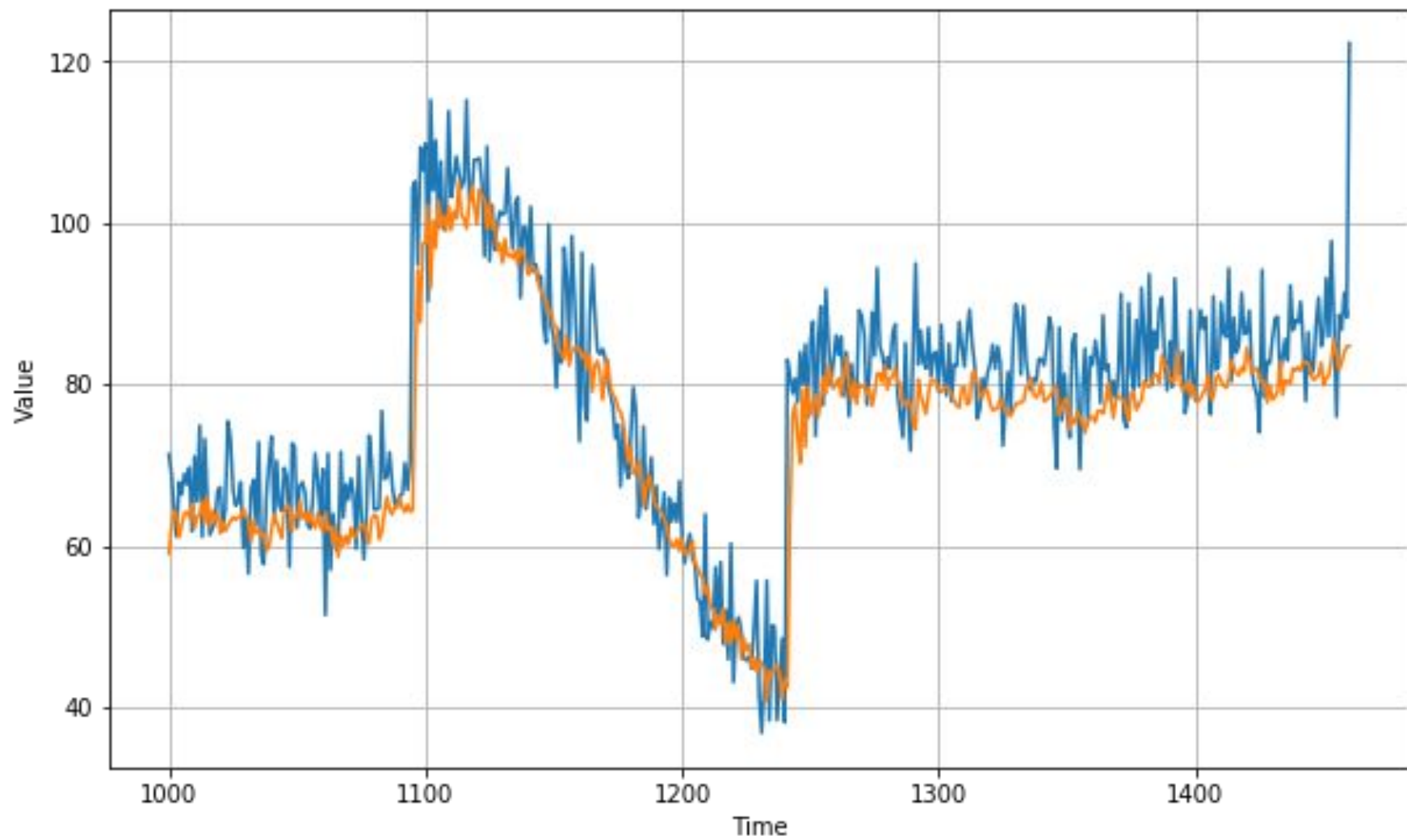
optimizer = tf.keras.optimizers.SGD(learning_rate=7e-6, momentum=0.9)
model.compile(loss="mse", optimizer=optimizer)
history = model.fit(dataset, epochs=500)
```

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



```
# 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()
```





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

4.4847784