

# 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>

```
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)
```

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, and you learned a little bit about Convolutional Neural Networks (ConvNets). In this course you'll go deeper into using ConvNets with real-world data, and learn about techniques that you can use. [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)
```

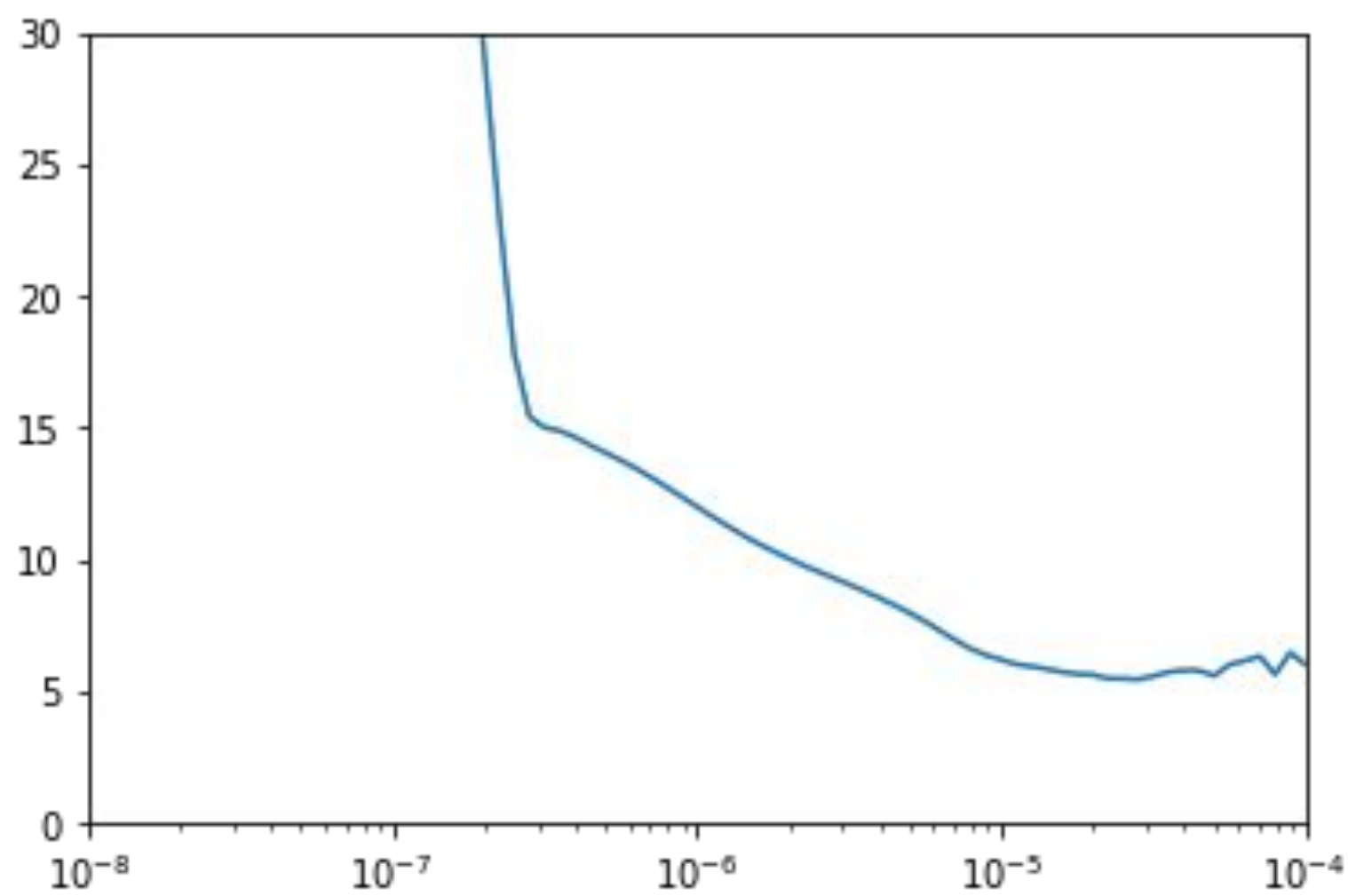
```
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
```

```
    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)
```



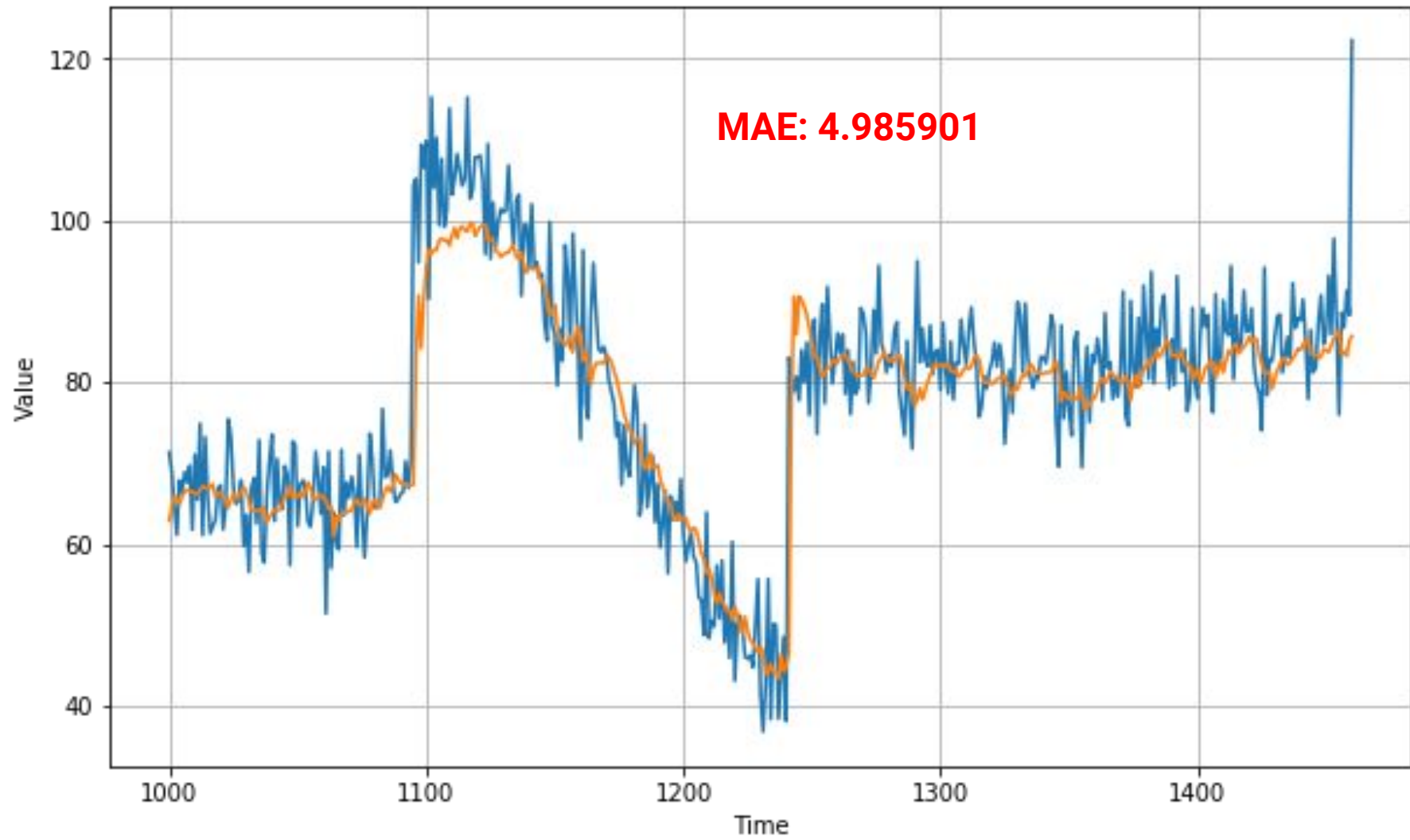
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu",
                           input_shape=[None, 1]),
    tf.keras.layers.LSTM(32, return_sequences=True),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

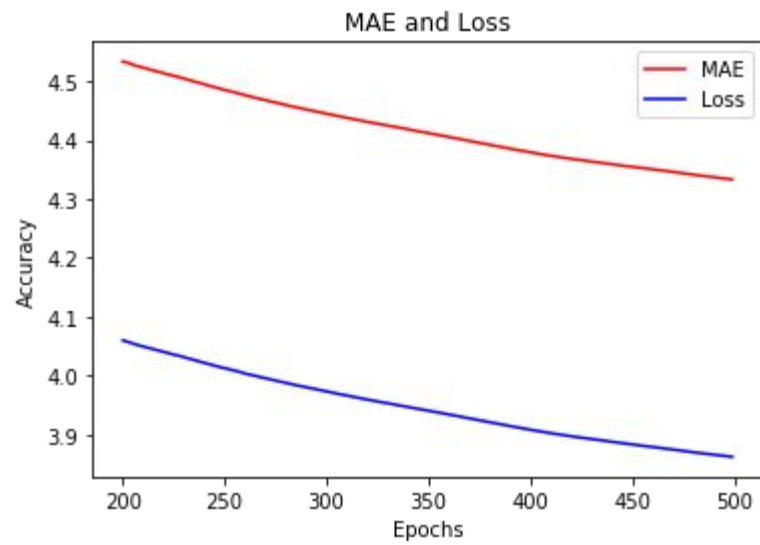
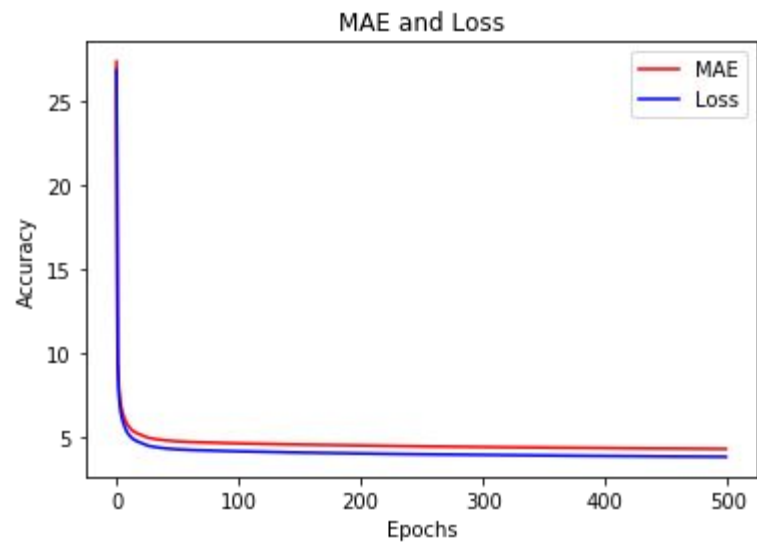
```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
```

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

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

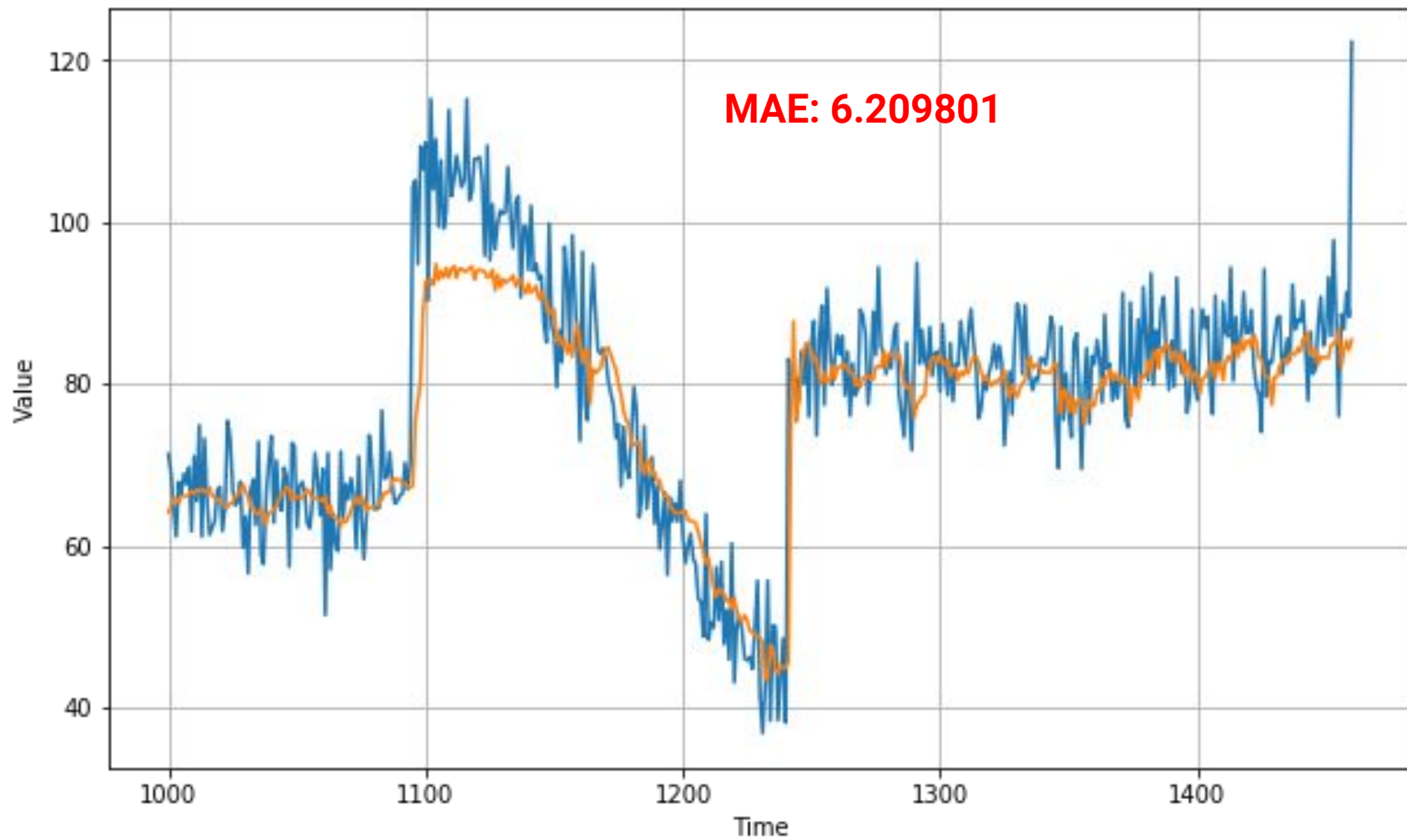




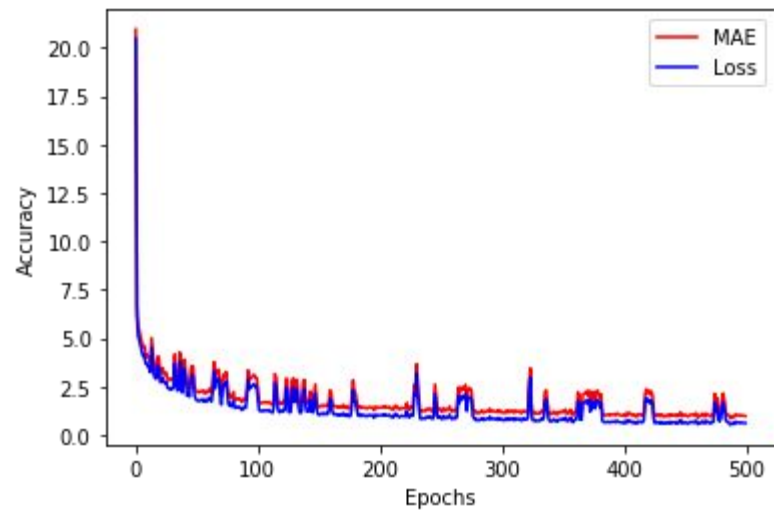


```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                            strides=1, padding="causal",
                            activation="relu",
                            input_shape=[None, 1]),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,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

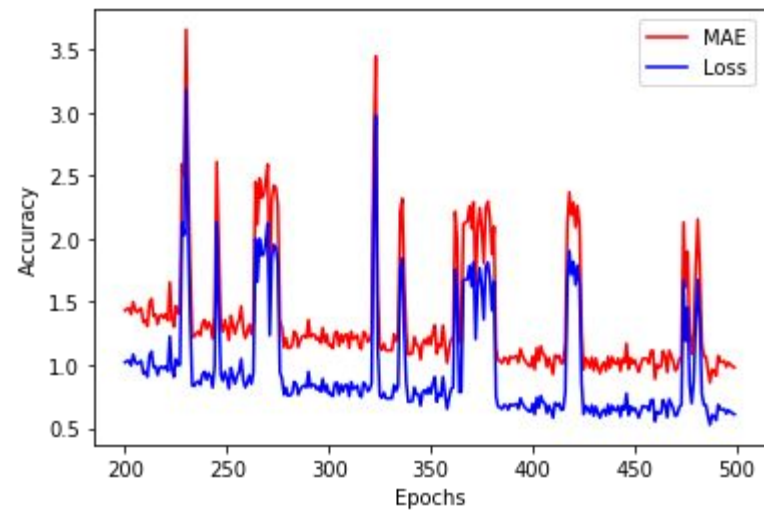
Epoch 495/500  
31/31 [=====] - 2s 58ms/step - loss: 0.6491 - mae: 1.0314  
Epoch 496/500  
31/31 [=====] - 2s 60ms/step - loss: 0.6155 - mae: 0.9857  
Epoch 497/500  
31/31 [=====] - 2s 59ms/step - loss: 0.6425 - mae: 1.0207  
Epoch 498/500  
31/31 [=====] - 2s 59ms/step - loss: 0.6330 - mae: 1.0046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



MAE and Loss



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

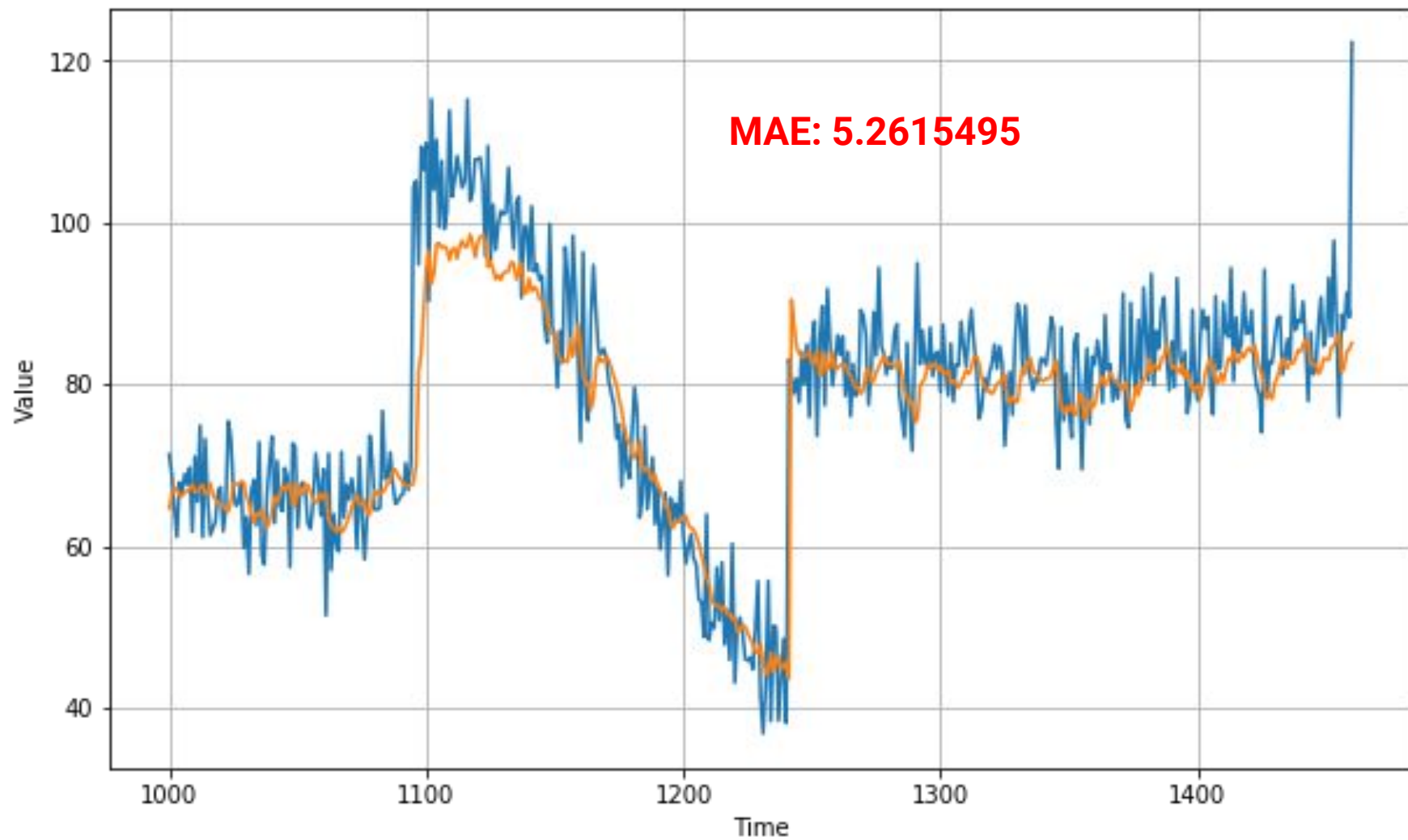
[youtube.com/watch?v=l4ISUAcvHF](https://www.youtube.com/watch?v=l4ISUAcvHF)



Machine Learning

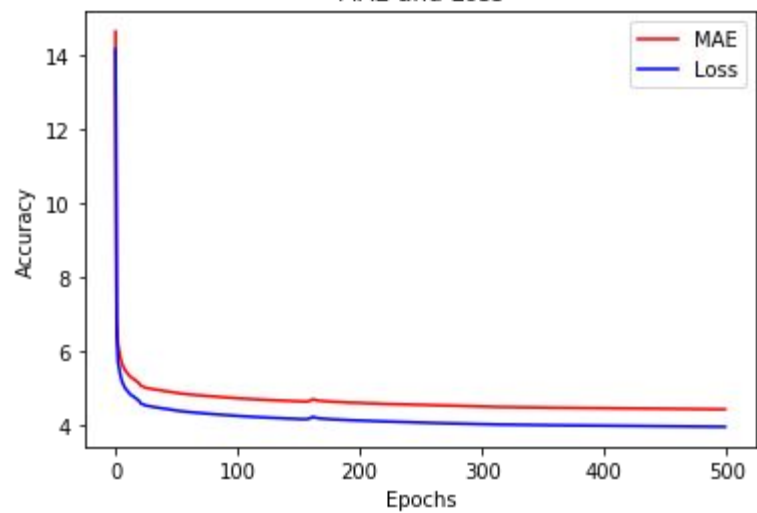


Lecture 17.3 – Large Scale Machine Learning | Mini Batch Gradient Descent – [ Andrew Ng ]

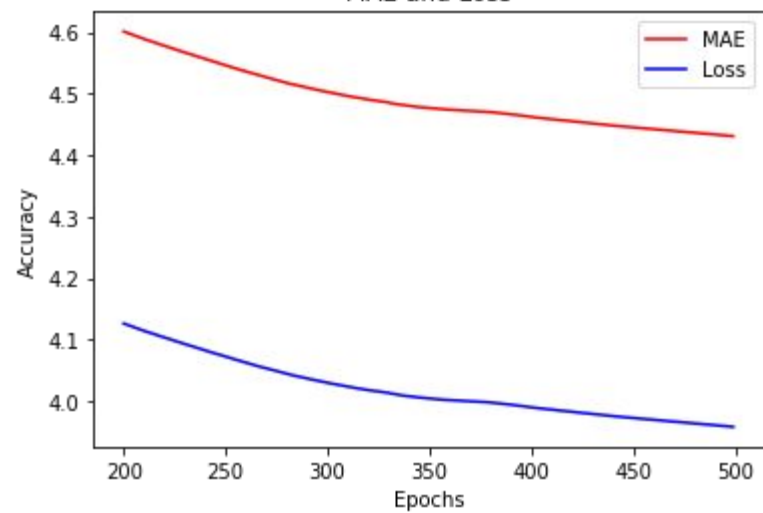




MAE and Loss



MAE and Loss





Dataset

Sunspots

Monthly Mean Total Sunspot Number - form 1749 to july 2018



Especuloide · updated a year ago (Version 3)

Data

Kernels (5)

Discussion

Activity

Metadata

Download (22 KB)

New Kernel

 Usability 7.1

 License CC0: Public Domain

 Tags No tags yet

Description


Context

Sunspots are temporary phenomena on the Sun's photosphere that appear as spots darker than the surrounding areas. They are regions of reduced surface temperature caused by concentrations of magnetic field flux that inhibit convection. Sunspots usually appear in pairs of opposite magnetic polarity. Their number varies according to the approximately 11-year solar cycle.

Source: <https://en.wikipedia.org/wiki/Sunspot>

Content :

1749-01-01 1749-01-01 1749-01-01 1749-01-01 1749-01-01 1749-01-01 1749-01-01 1749-01-01 1749-01-01 1749-01-01

Data (22 KB)			
Data Sources	About this file	Columns	
<div><div></div><div>Sunspots.csv</div><div>3235 x 3</div></div>	Sunspots - Monthly Mean Total Sunspot Number	<div>#</div> <div>Date</div> <div># Monthly Mean Total Sunspot Number</div>	

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

```

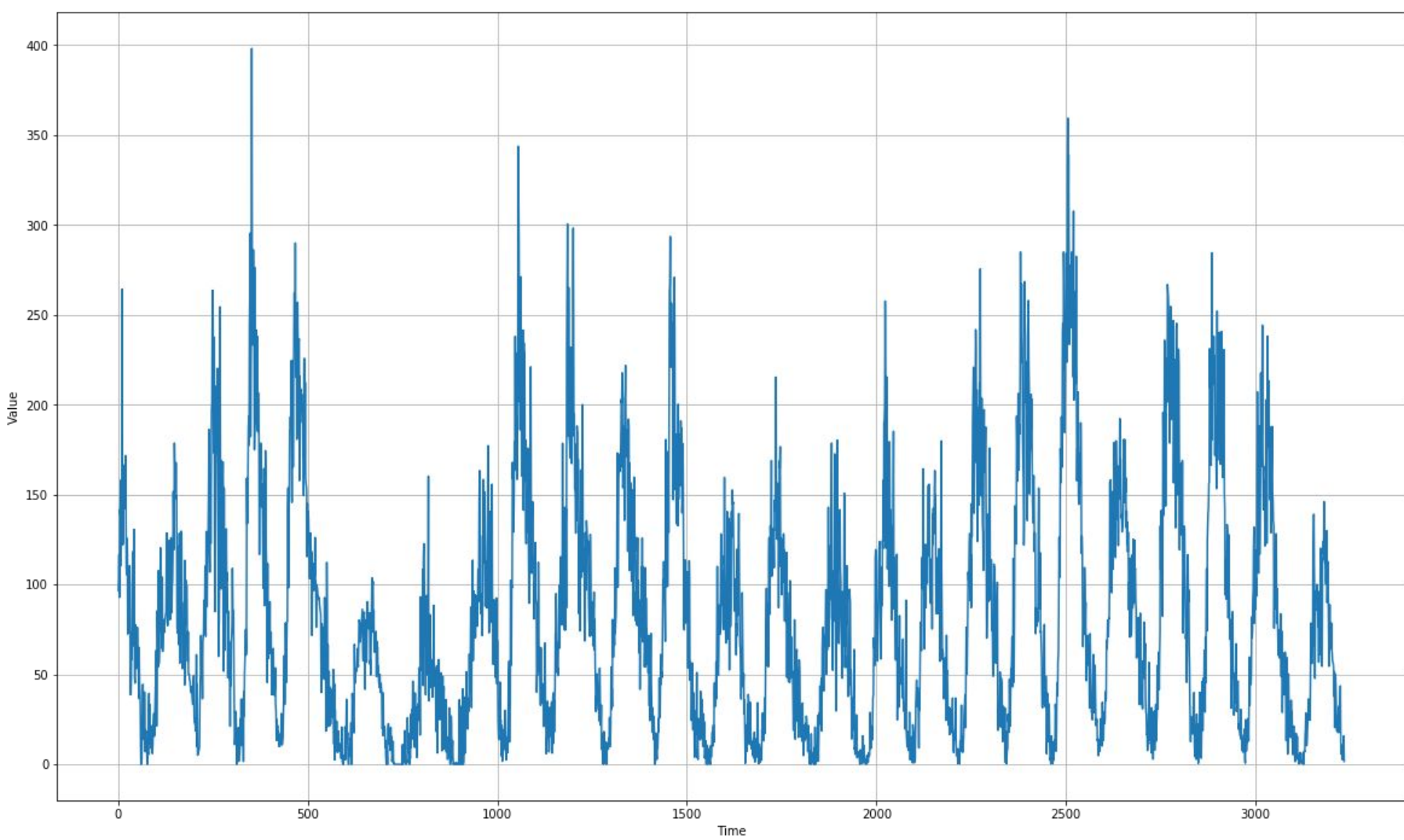
import csv
time_step = []
sunspots = []

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

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



```
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
```

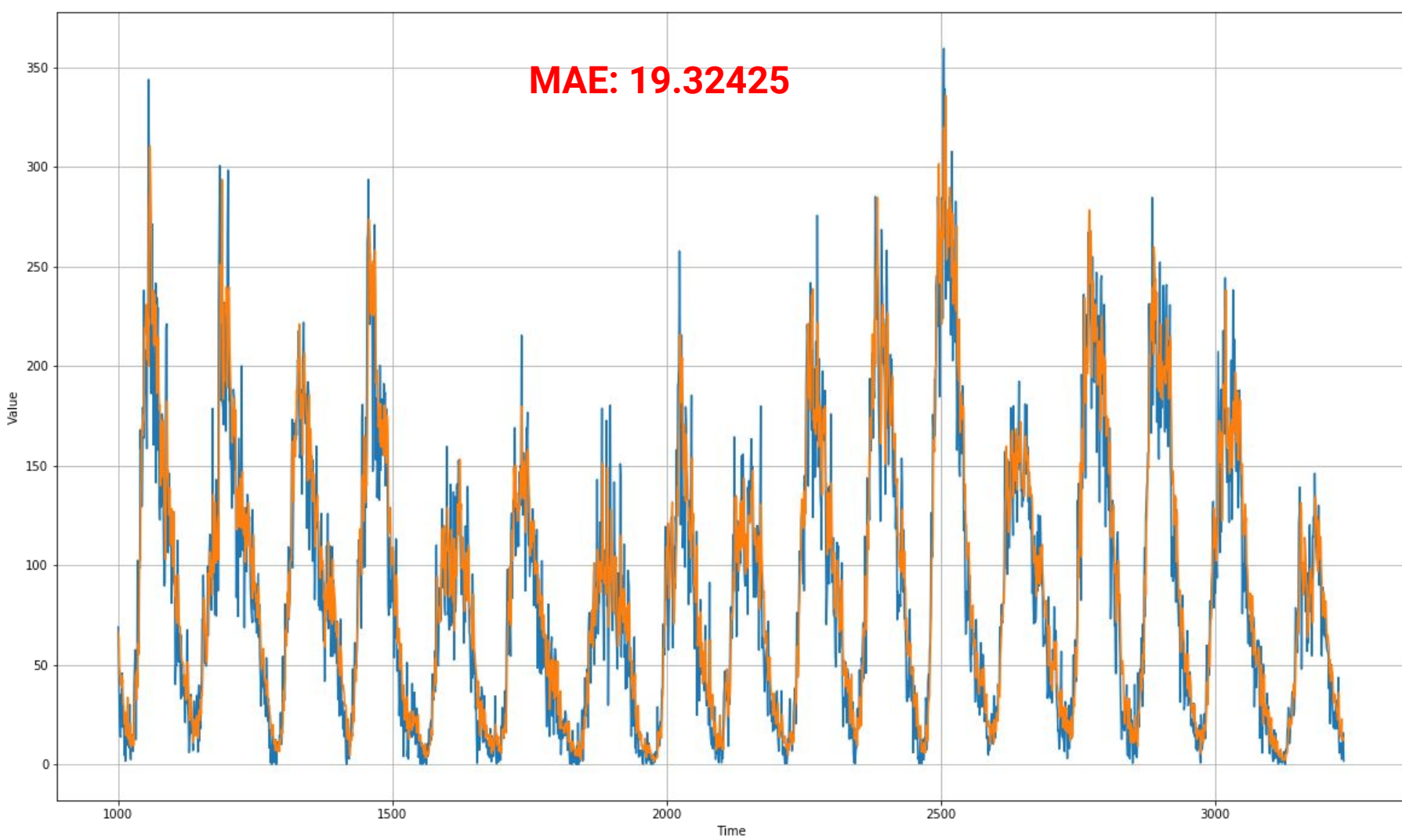
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):  
    dataset = tf.data.Dataset.from_tensor_slices(series)  
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)  
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))  
    dataset = 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)
```

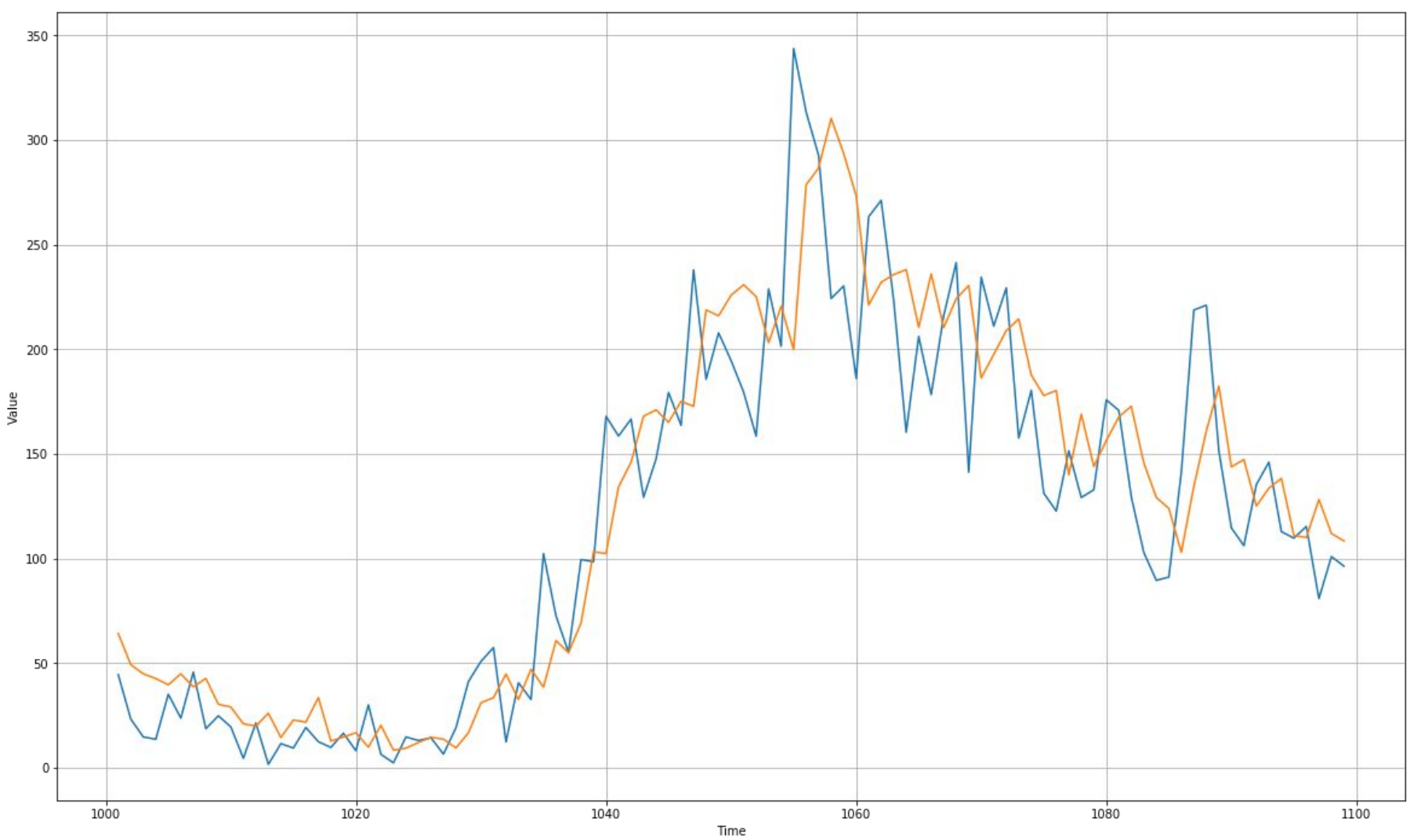
```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Dense(10, input_shape=[window_size], activation="relu"),  
    tf.keras.layers.Dense(10, activation="relu"),  
    tf.keras.layers.Dense(1)  
])
```

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

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

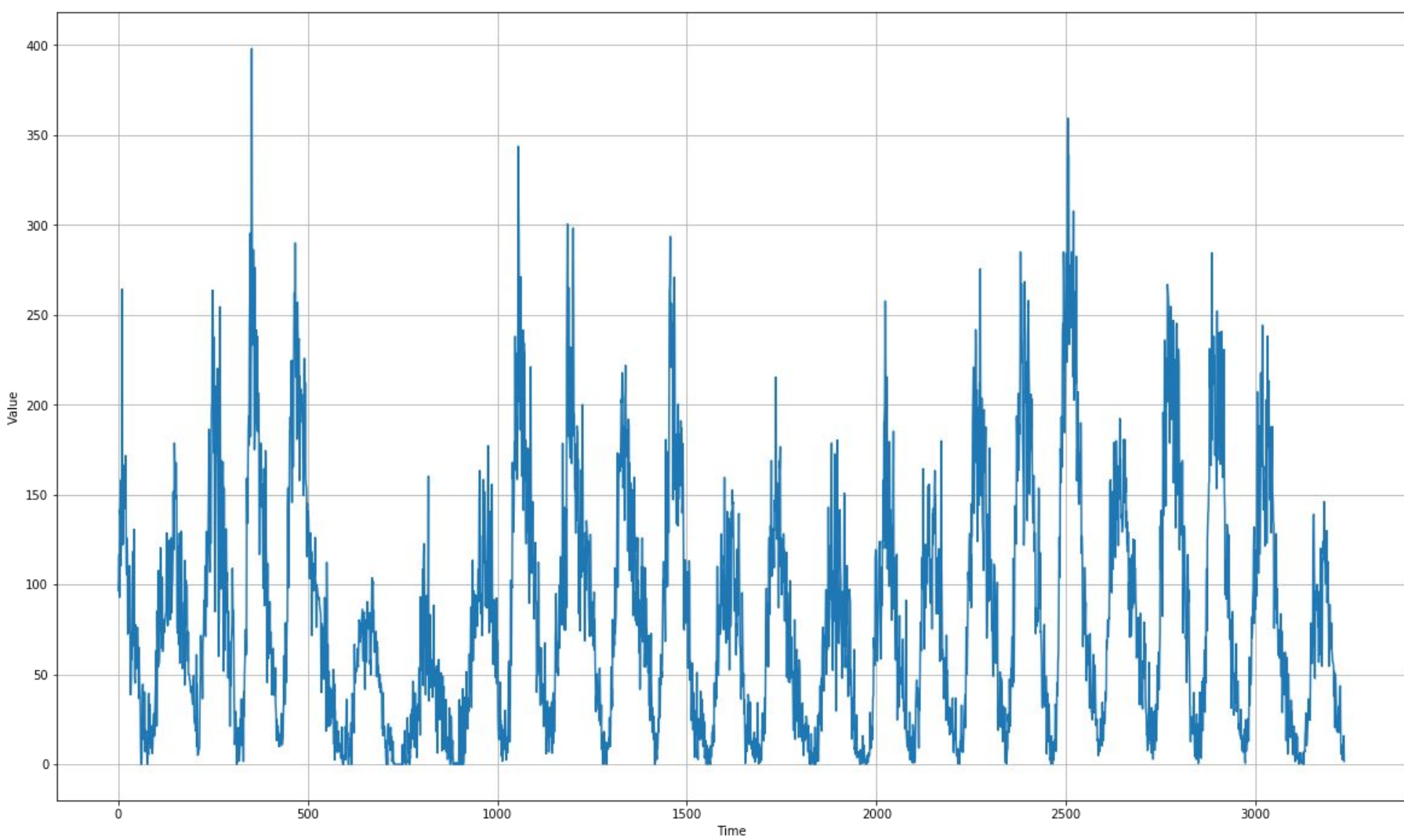






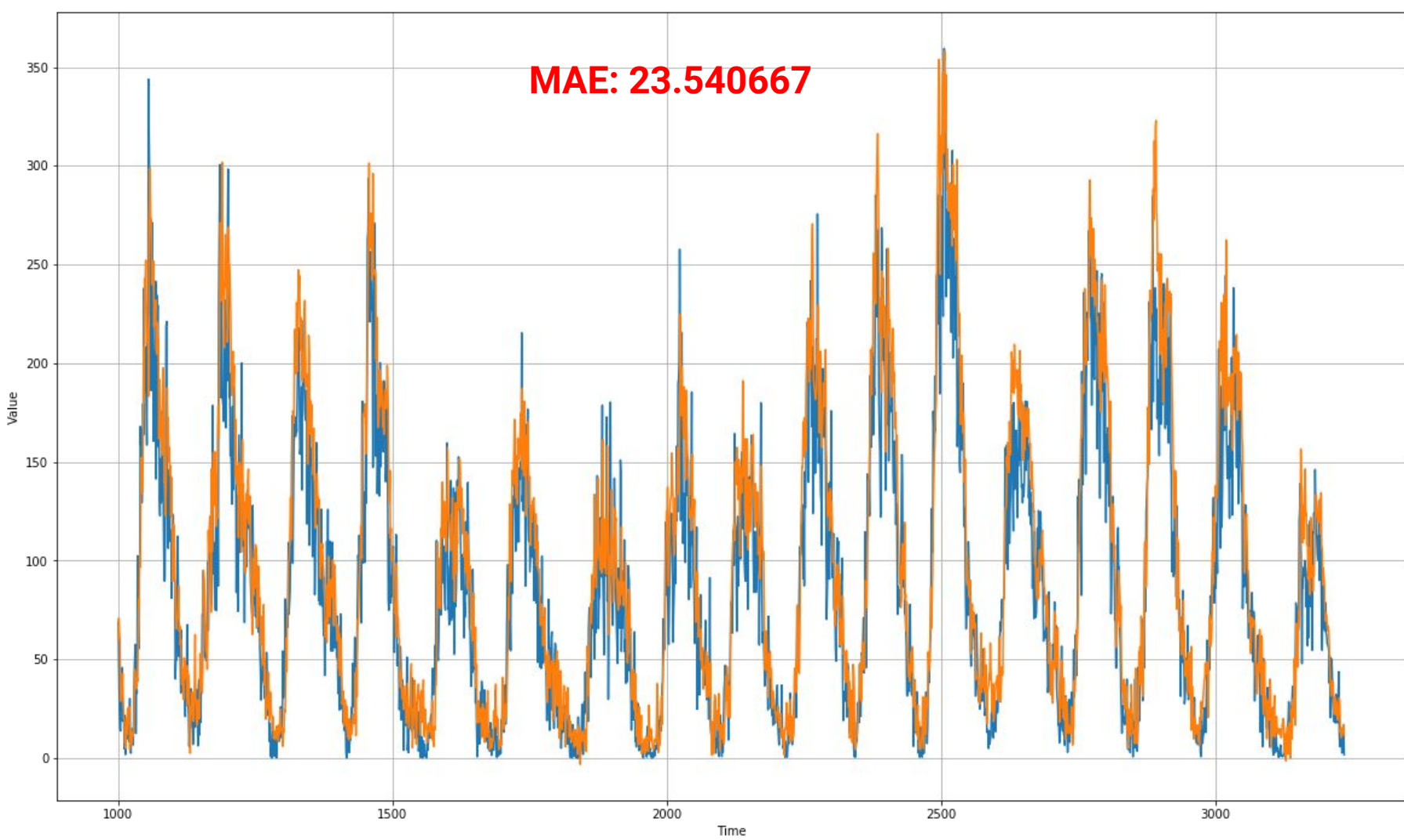
```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]
```

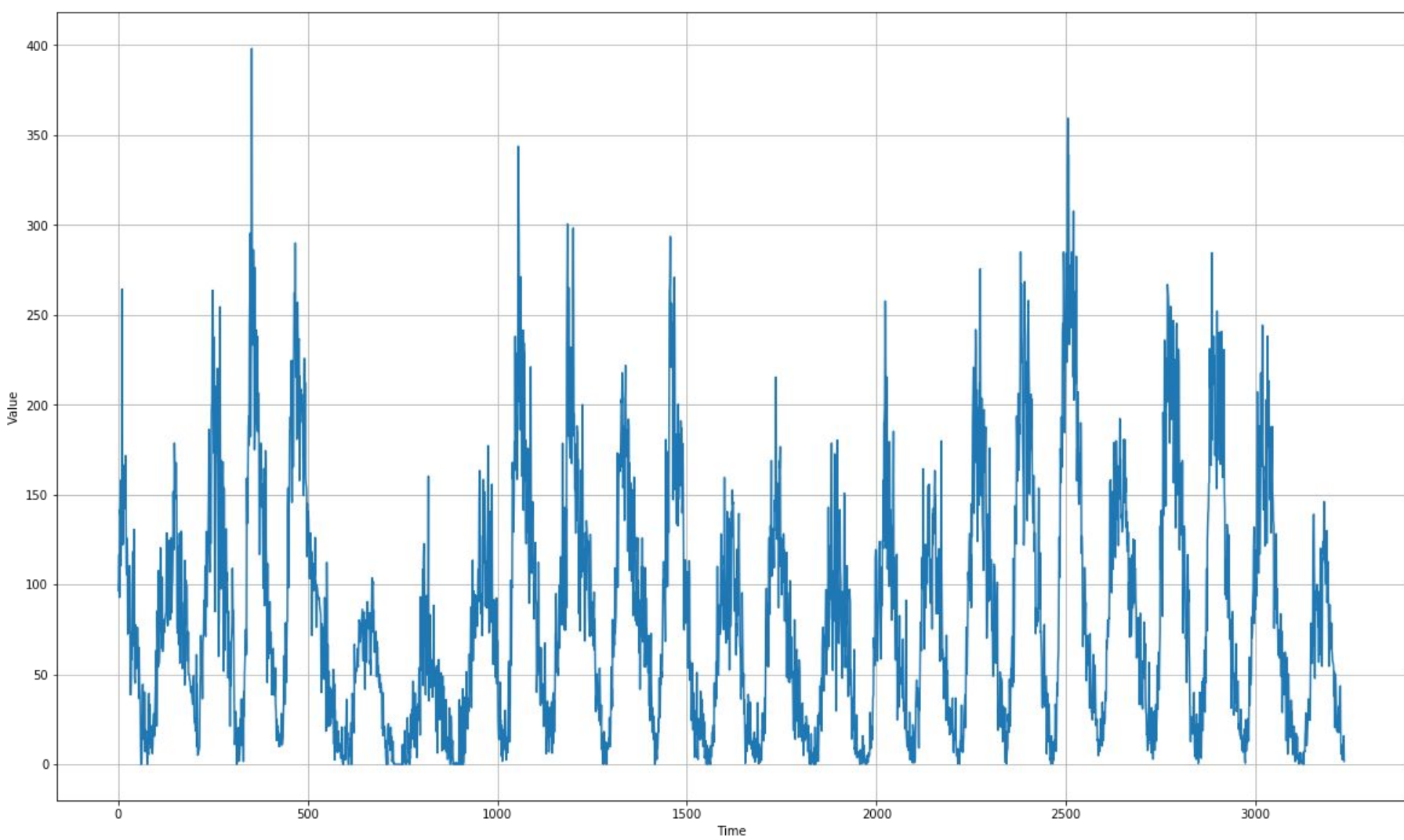
```
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000
```

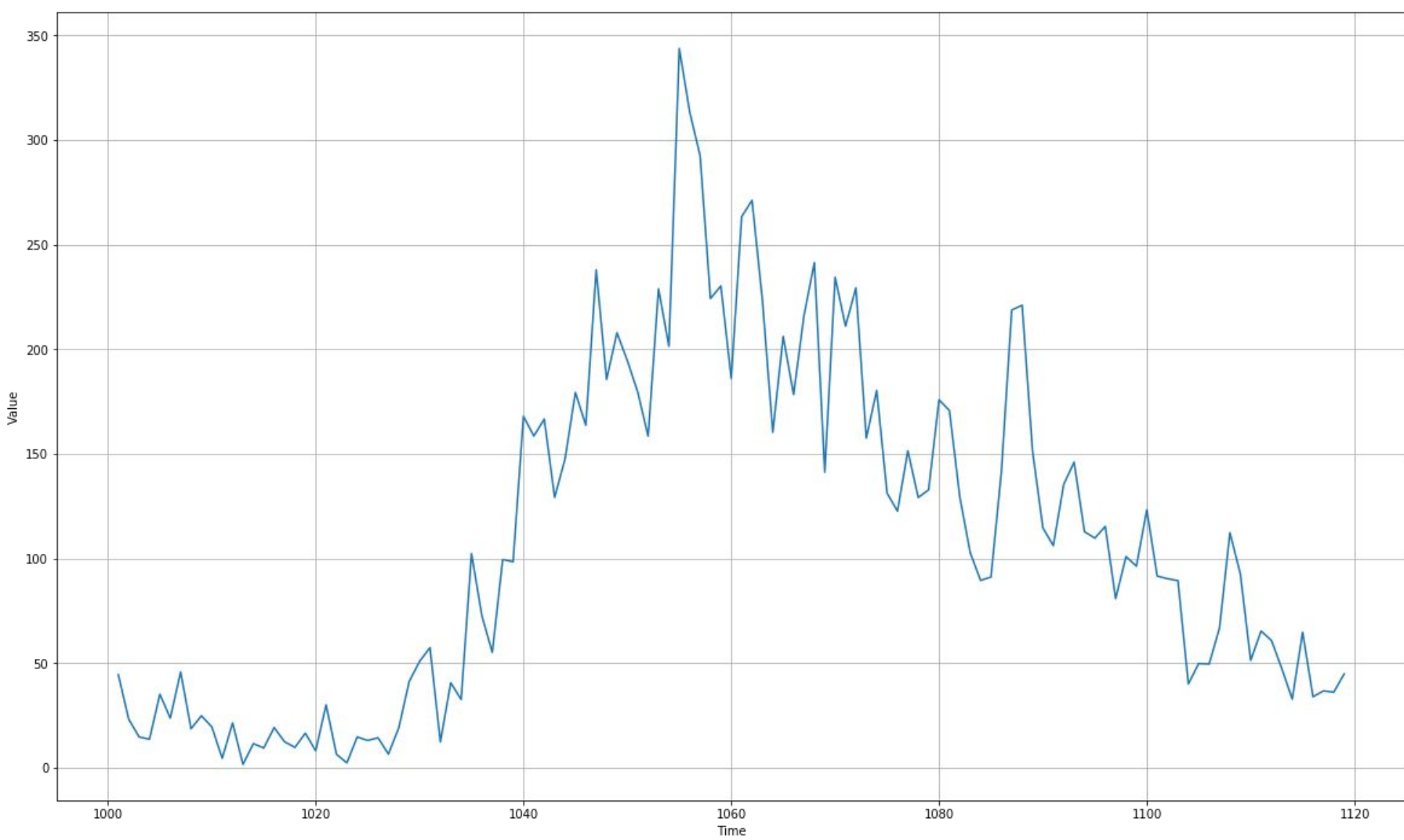


```
split_time = 1000  
time_train = time[:split_time]  
x_train = series[:split_time]  
time_valid = time[split_time:]  
x_valid = series[split_time:]
```

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







```
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
```

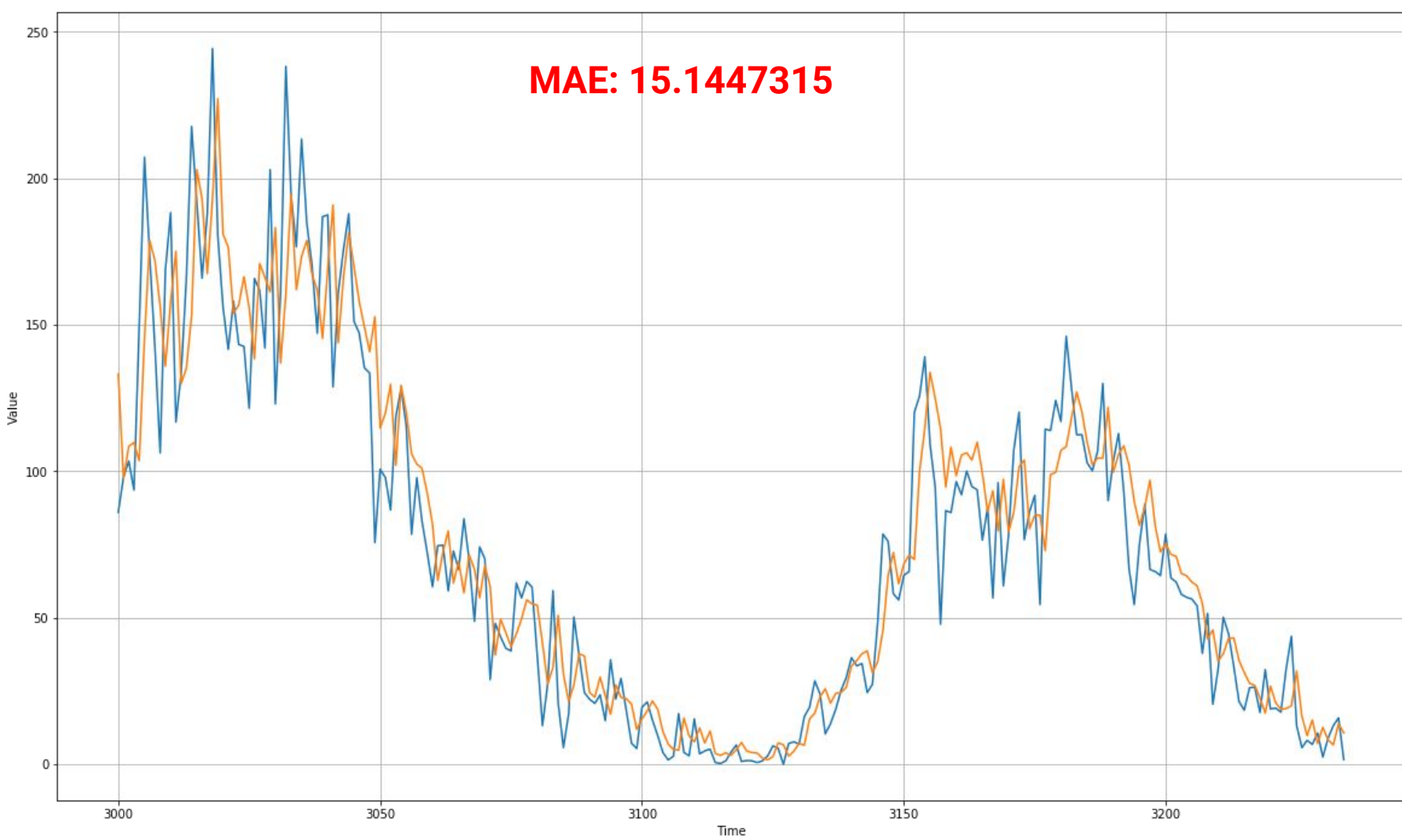


```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

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
```



```
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))
```

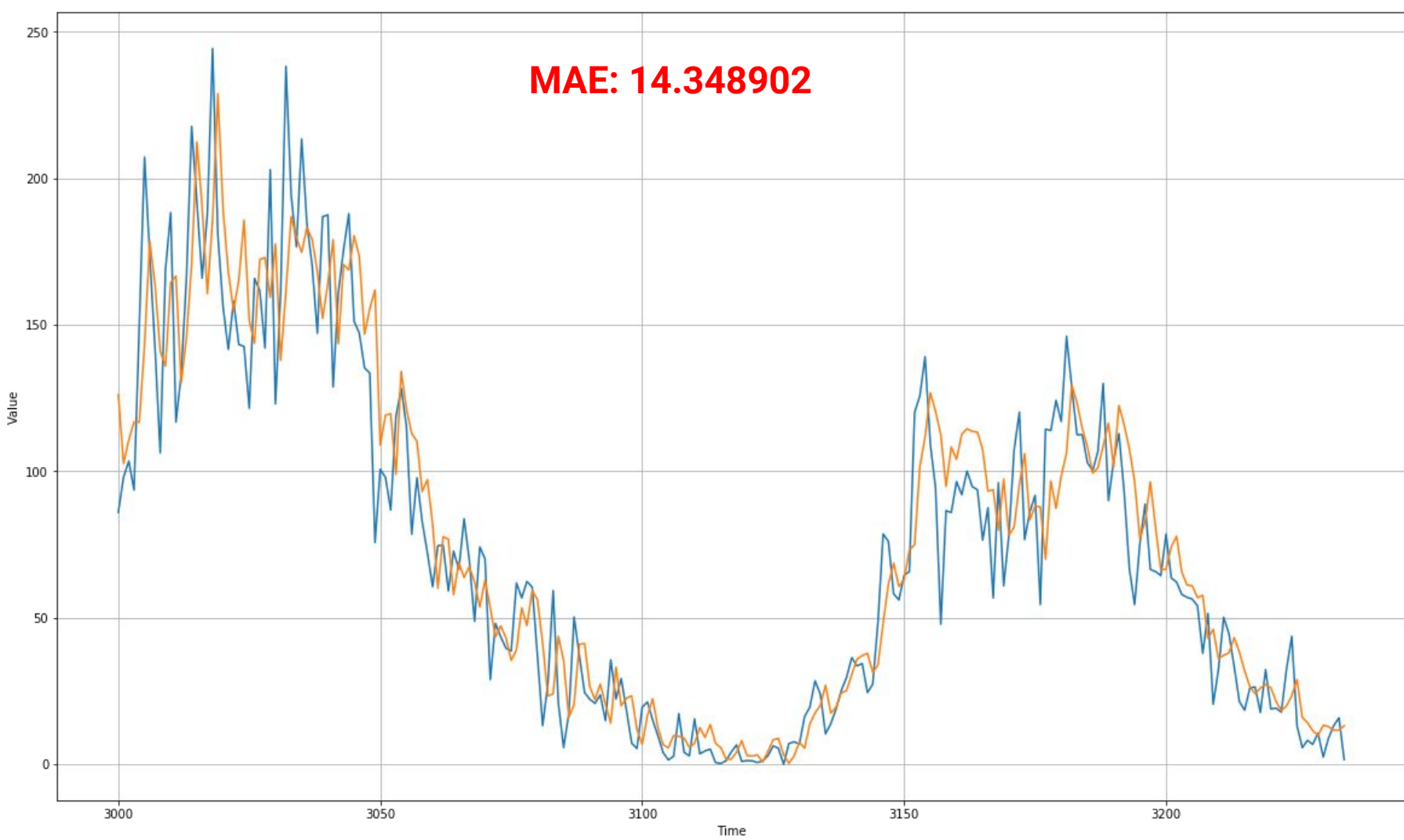
```
model.fit(dataset, epochs=100, verbose=0)
```

```
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)
```



```
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)  
])
```

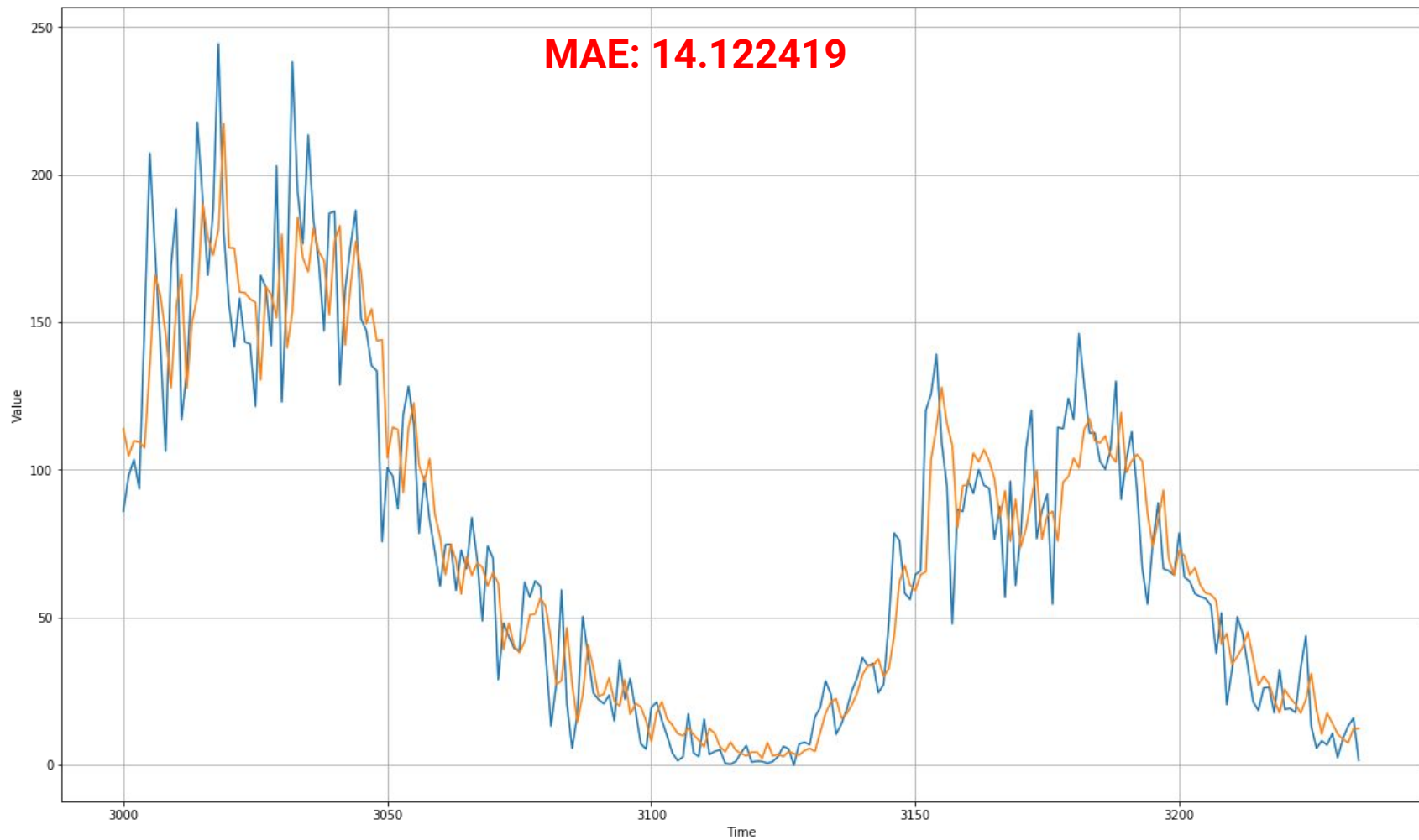
```
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6,  
momentum=0.9))  
model.fit(dataset, epochs=100, verbose=0)
```

```
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)
```





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

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

7.0773993

(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)

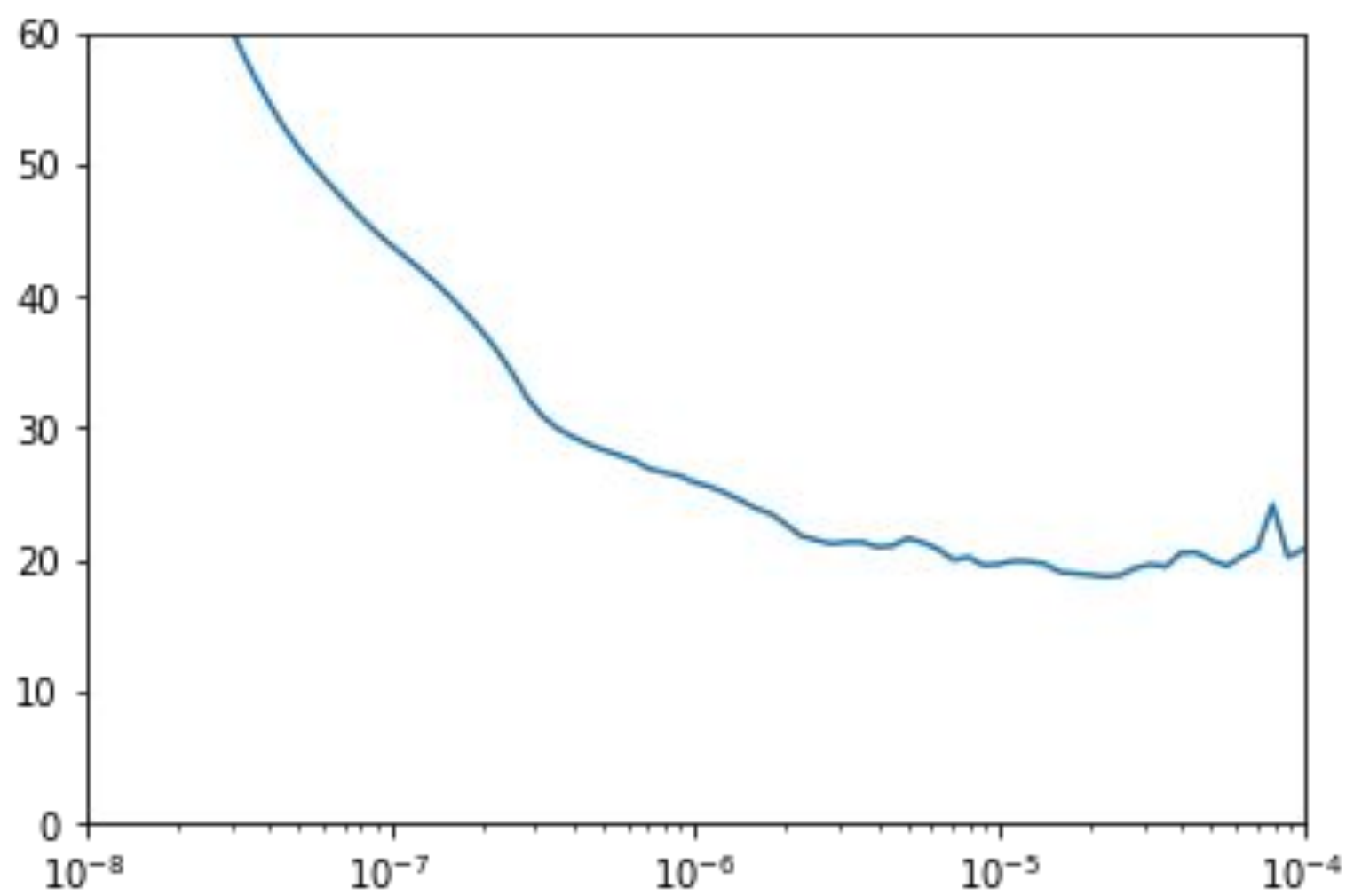
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                           input_shape=[None, 1]),
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, 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)

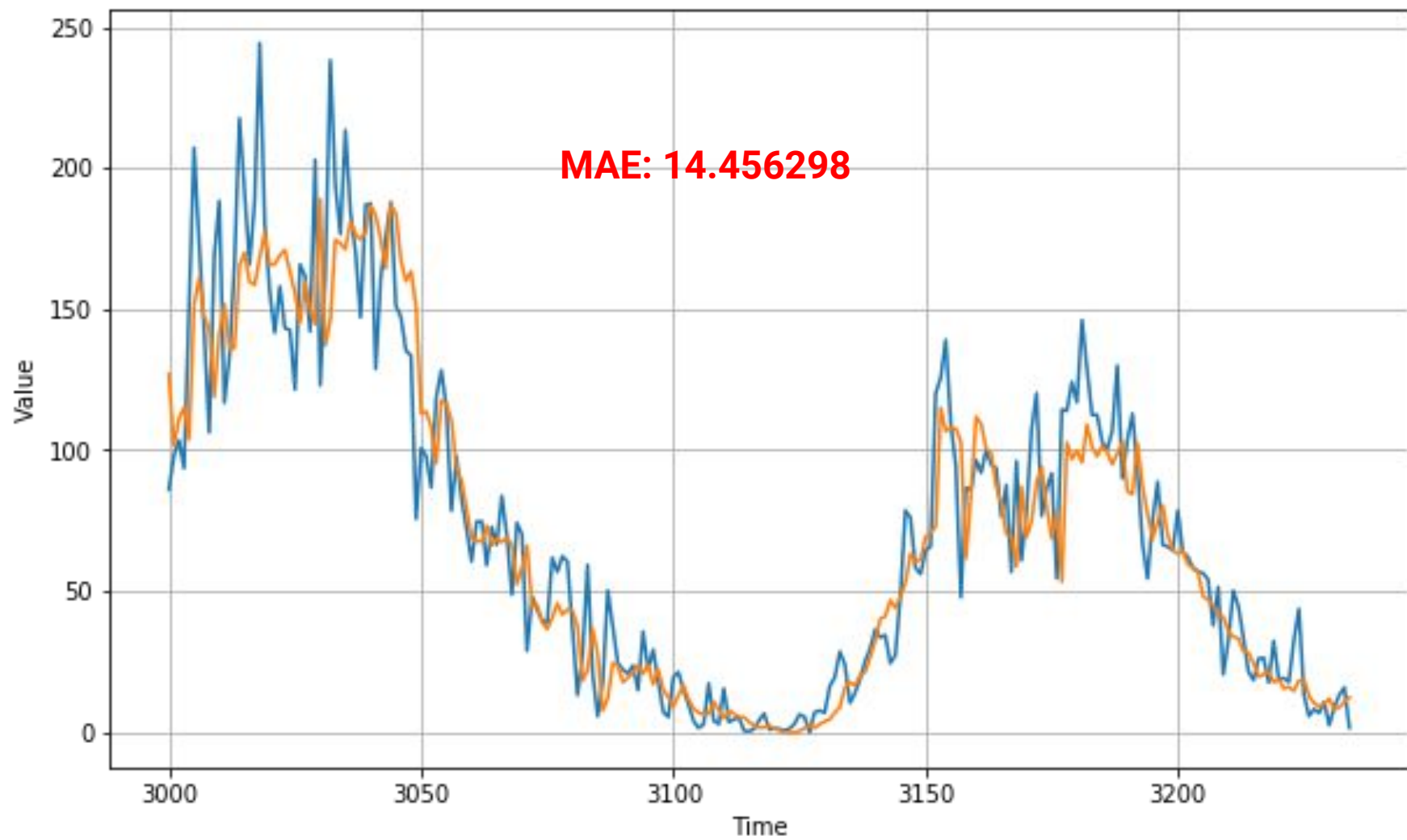
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal", activation="relu",
                           input_shape=[None, 1]),
    tf.keras.layers.LSTM(32, return_sequences=True),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dense(30, 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])
```

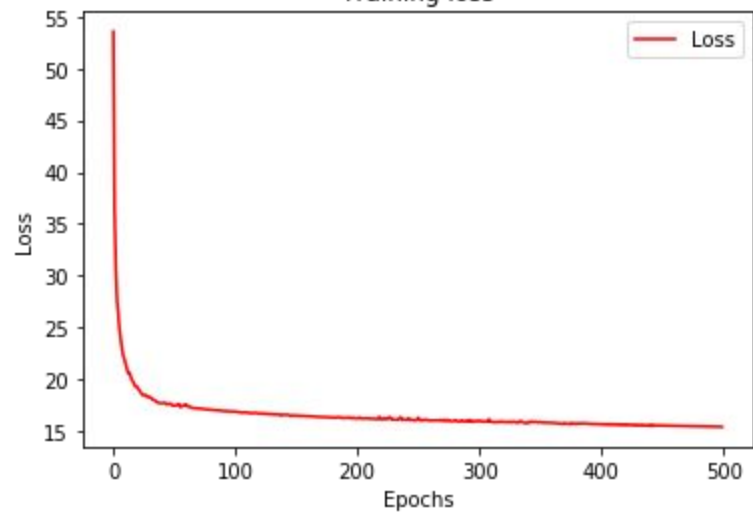


```
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                           strides=1, padding="causal",
                           activation="relu",
                           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)
])
```

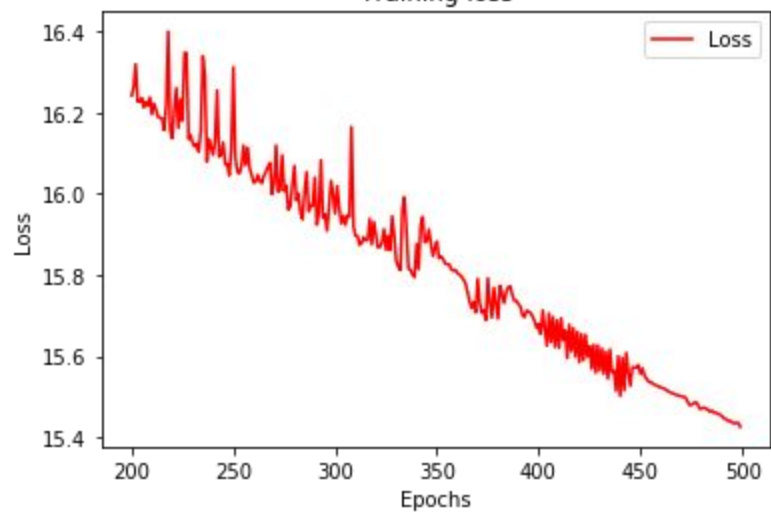
```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=500)
```



Training loss



Training loss



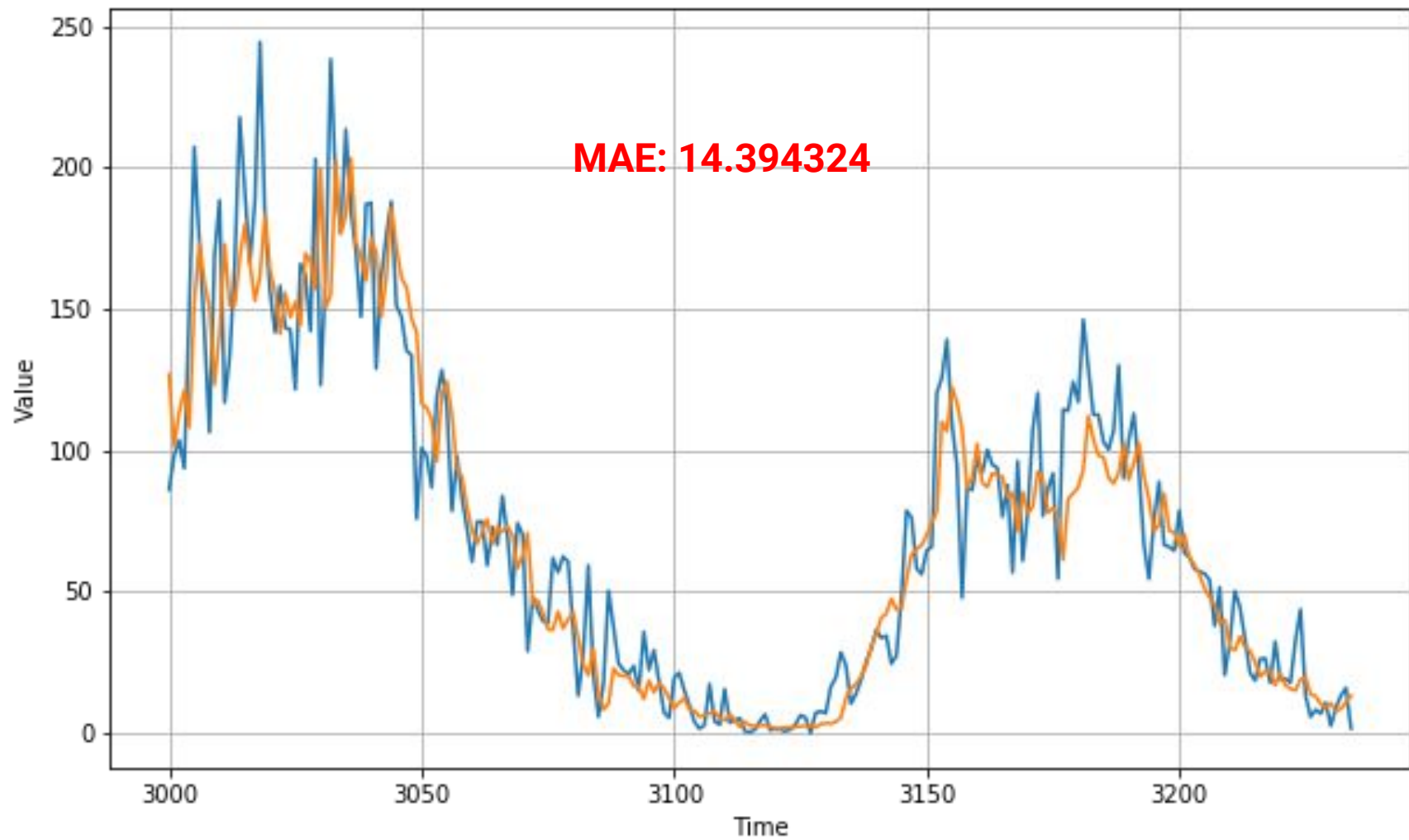
```

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)
])

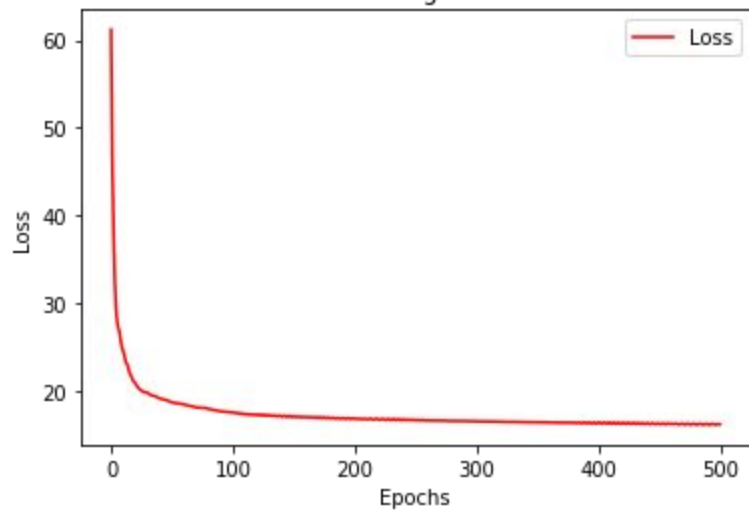
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=500)

```

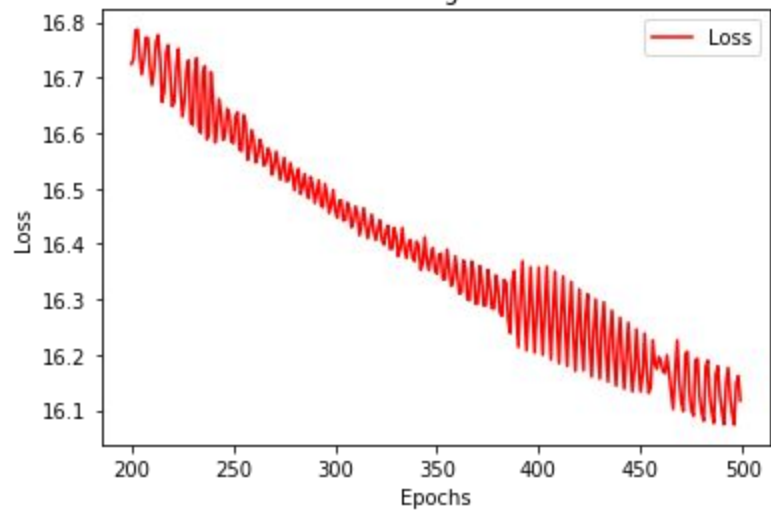




Training loss



Training loss



```
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)
])
```

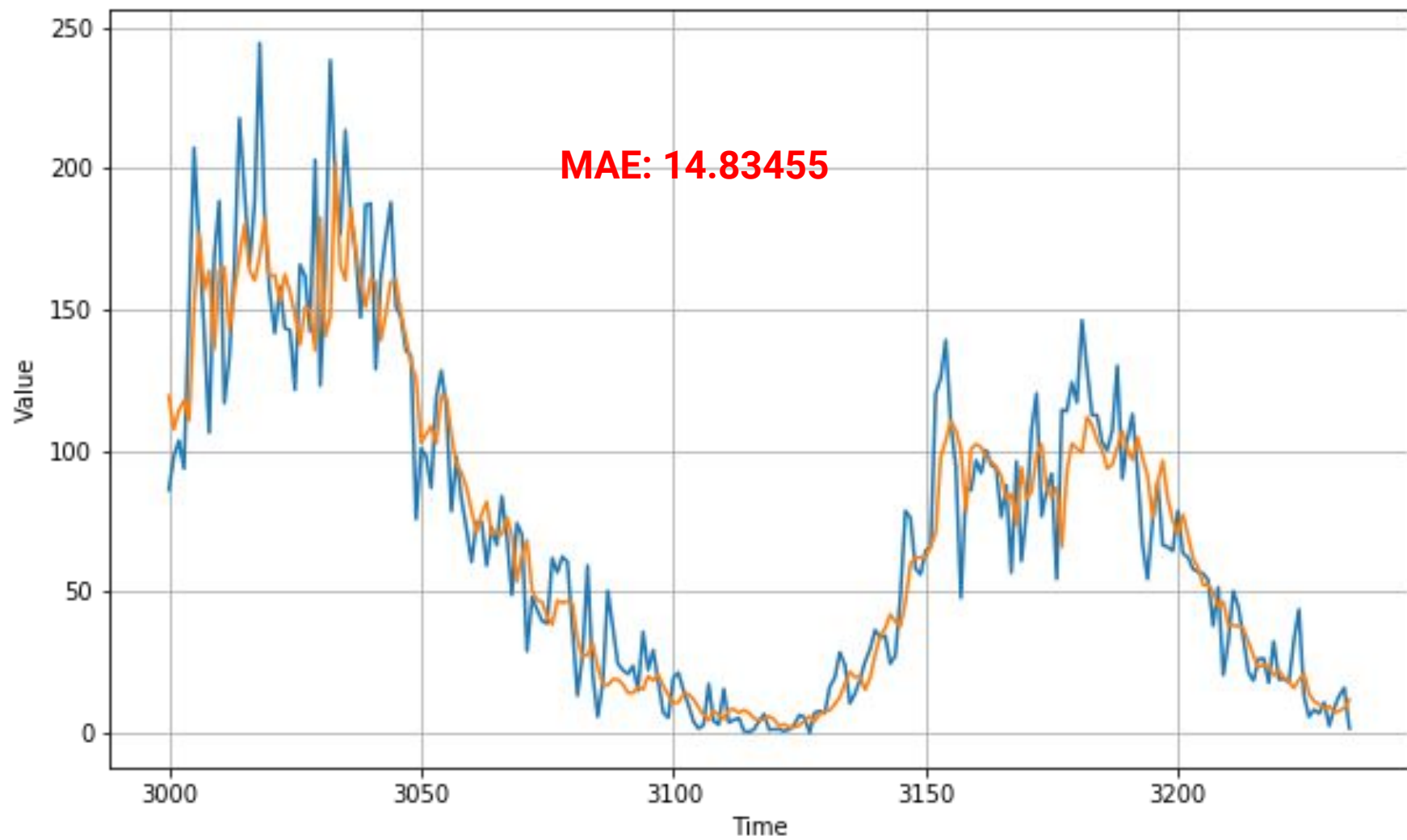
```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=500)
```

```
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)
])
```

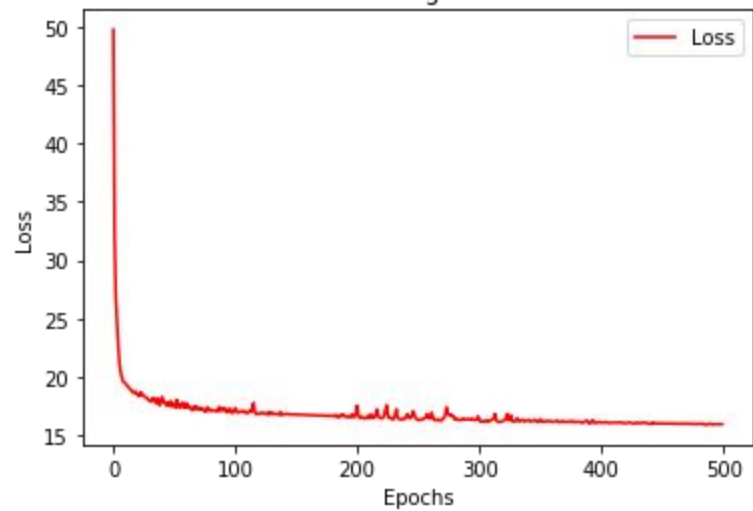
```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=500)
```

```
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)
])
```

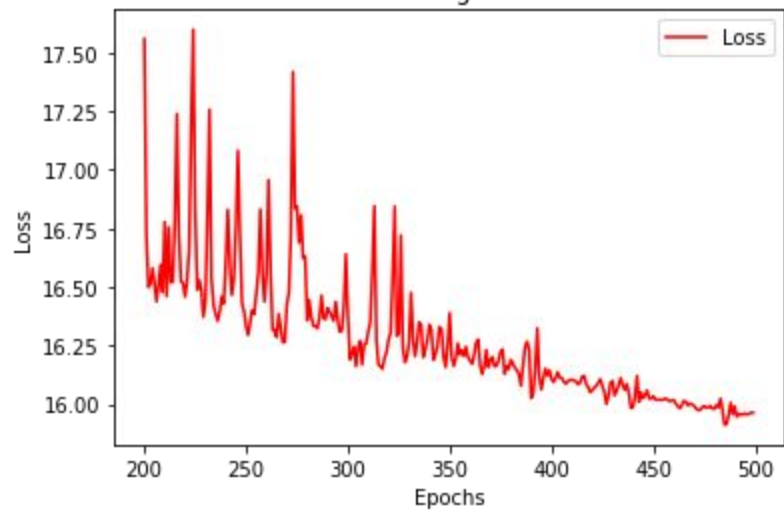
```
optimizer = tf.keras.optimizers.SGD(learning_rate=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=500)
```

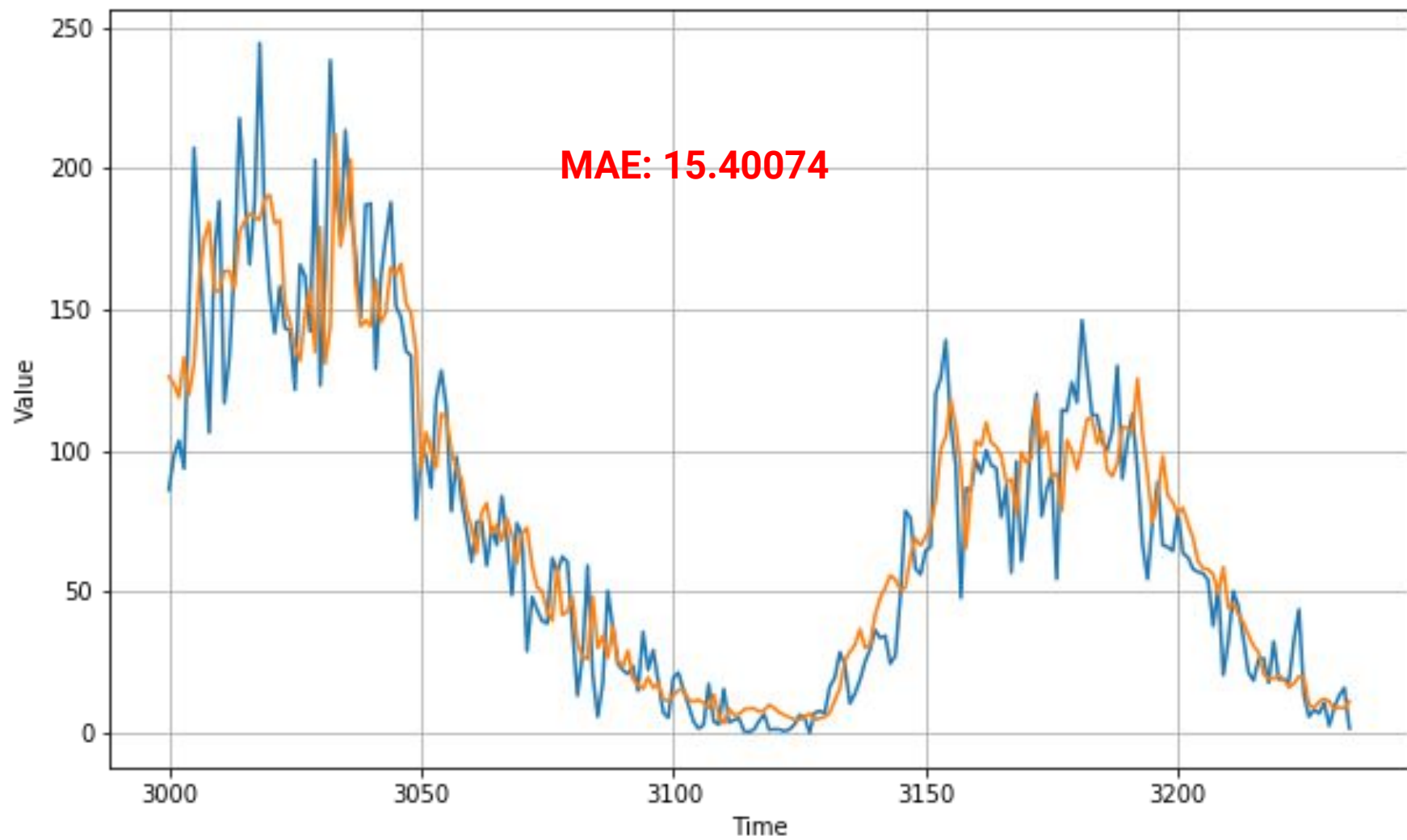


Training loss



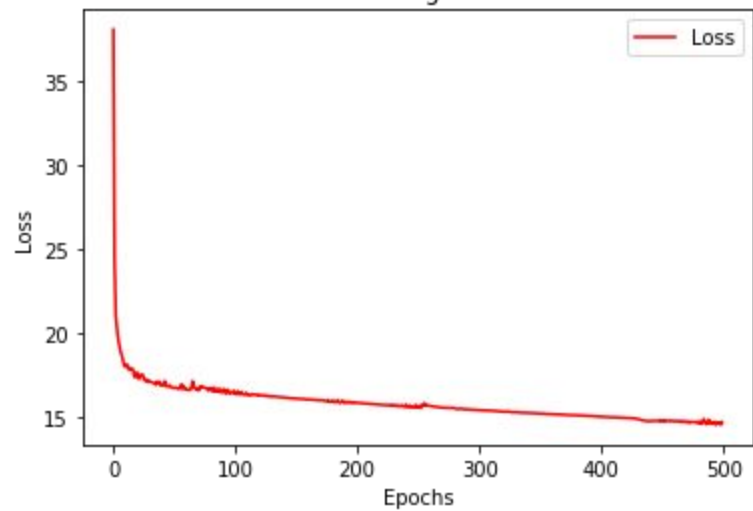
Training loss







Training loss



Training loss

