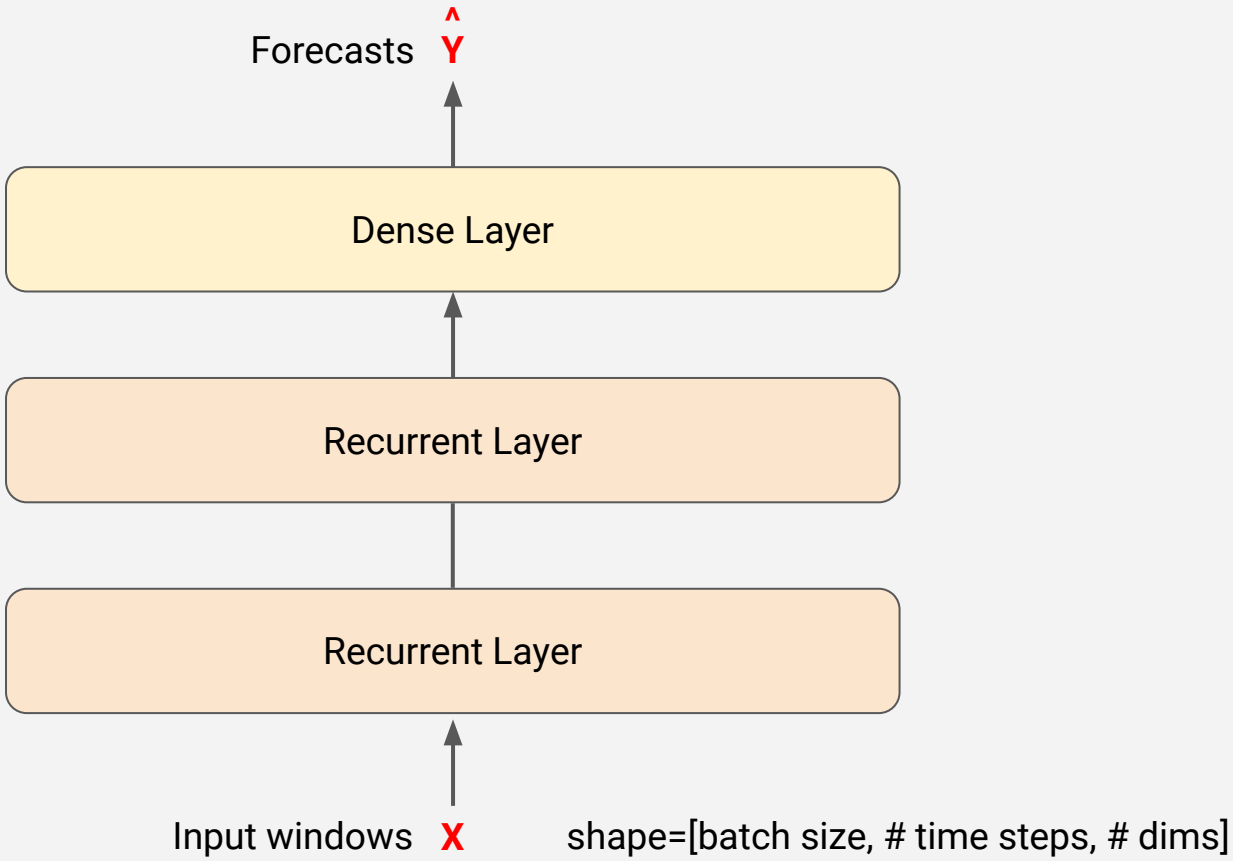
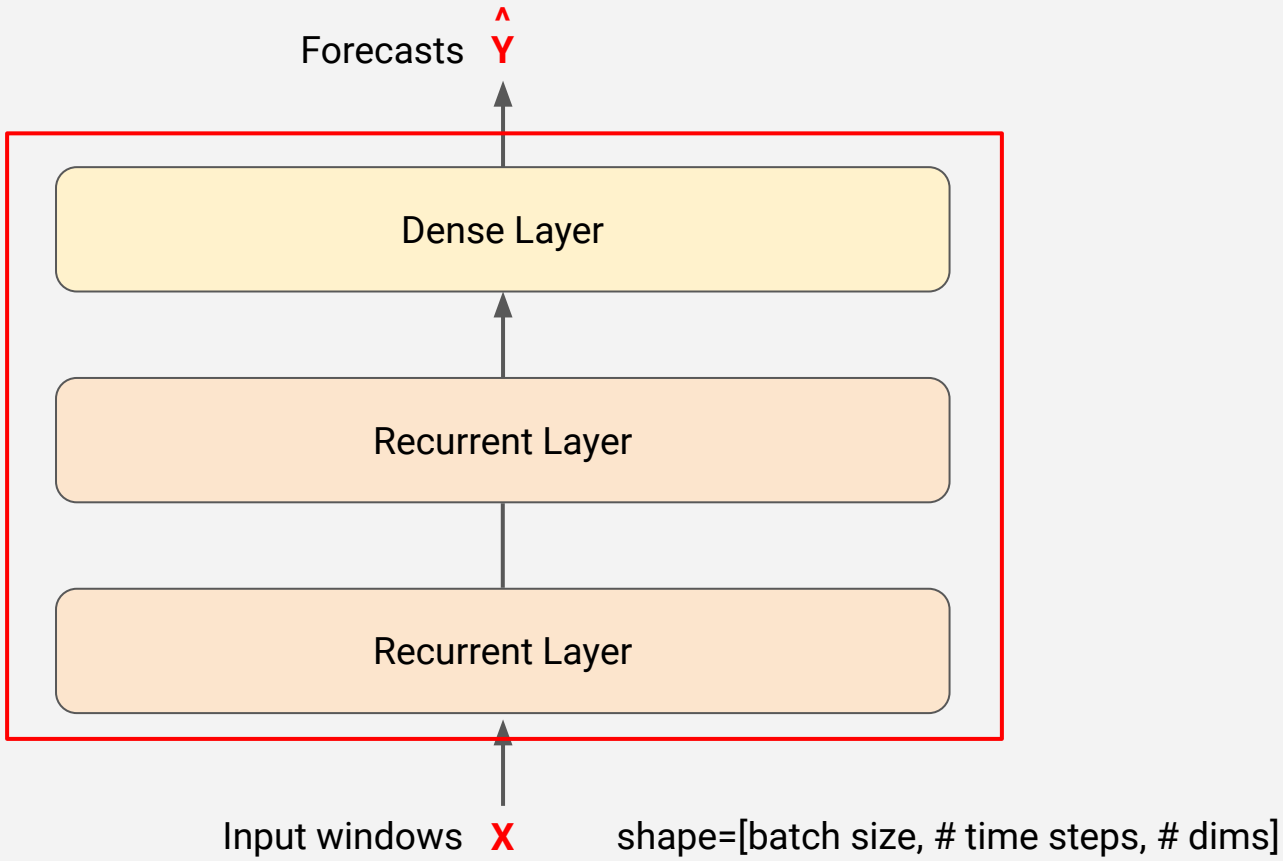
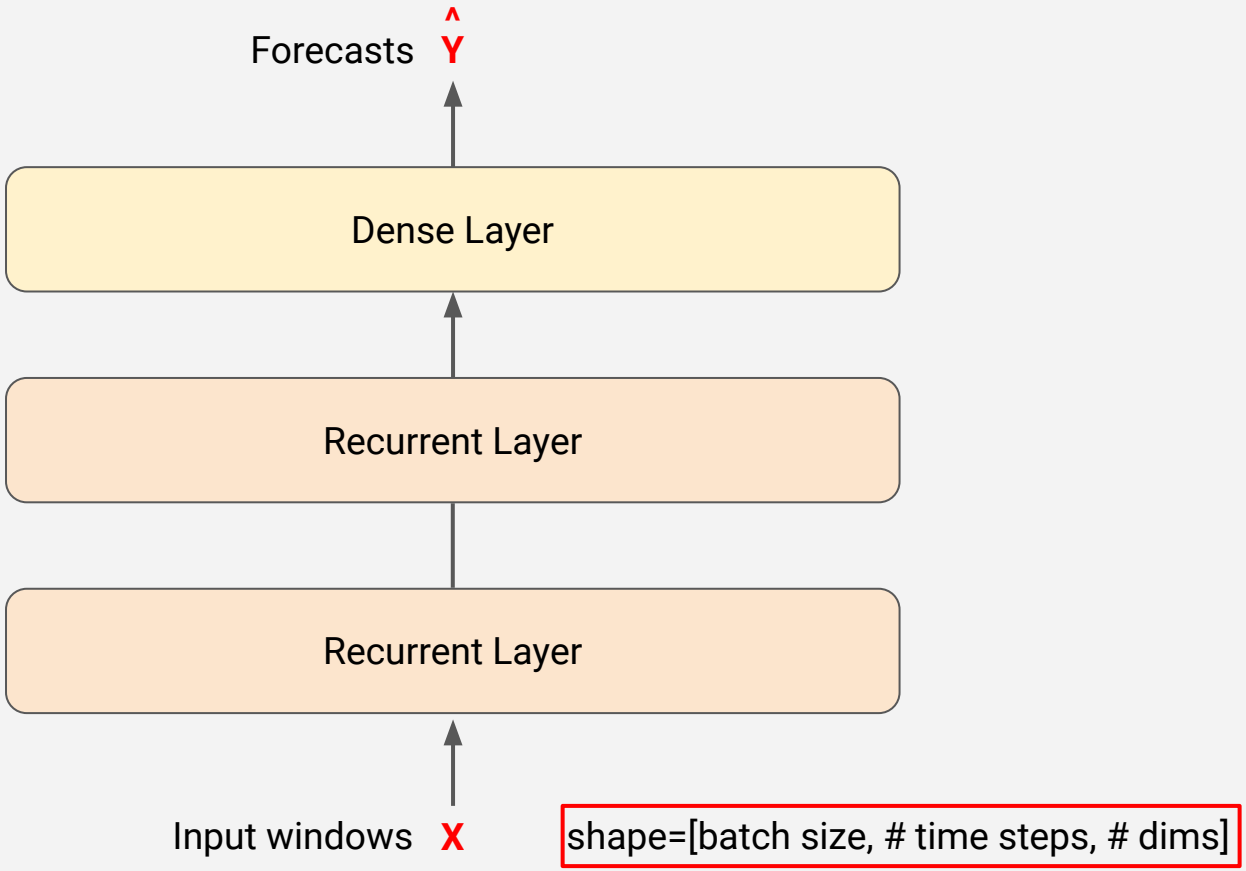
Recurrent Neural Network



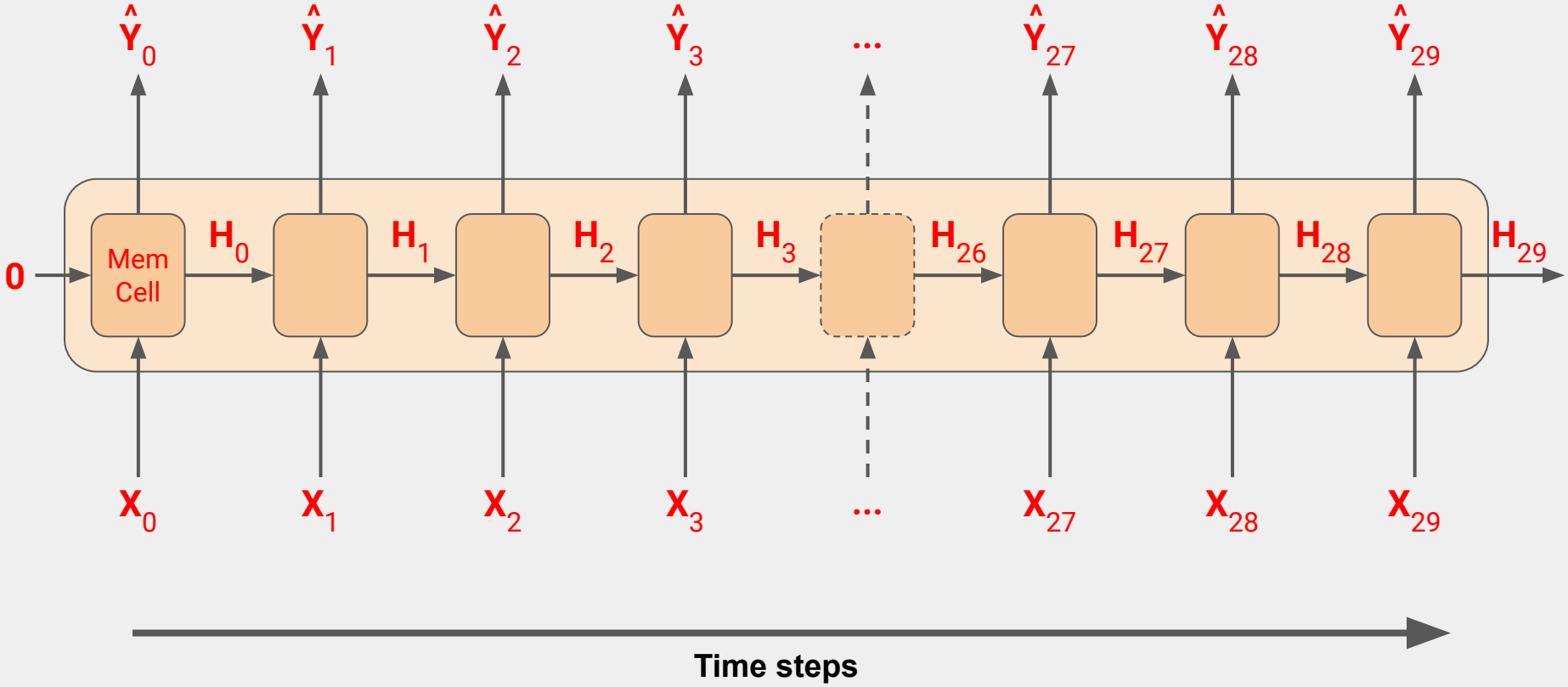
Recurrent Neural Network



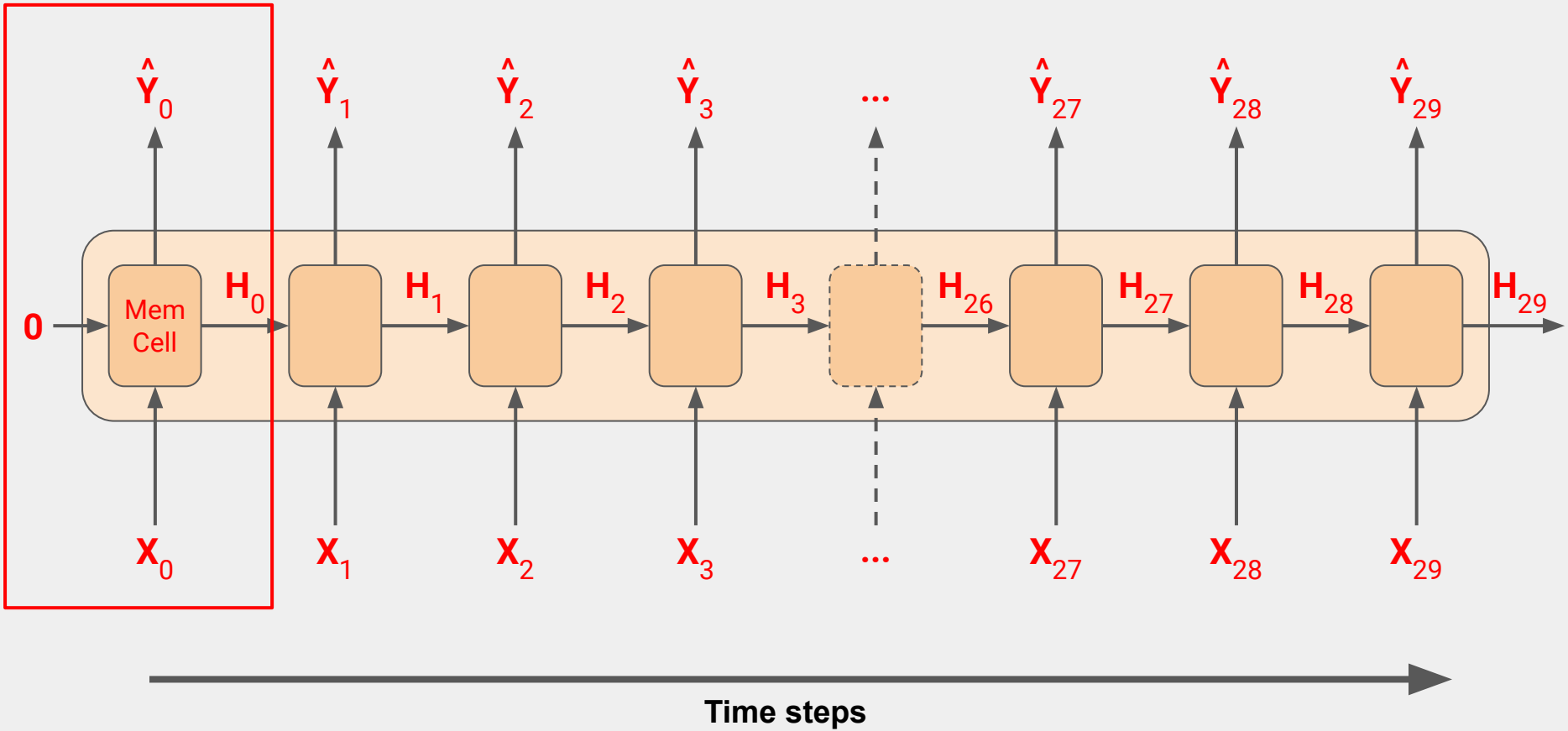
Recurrent Neural Network

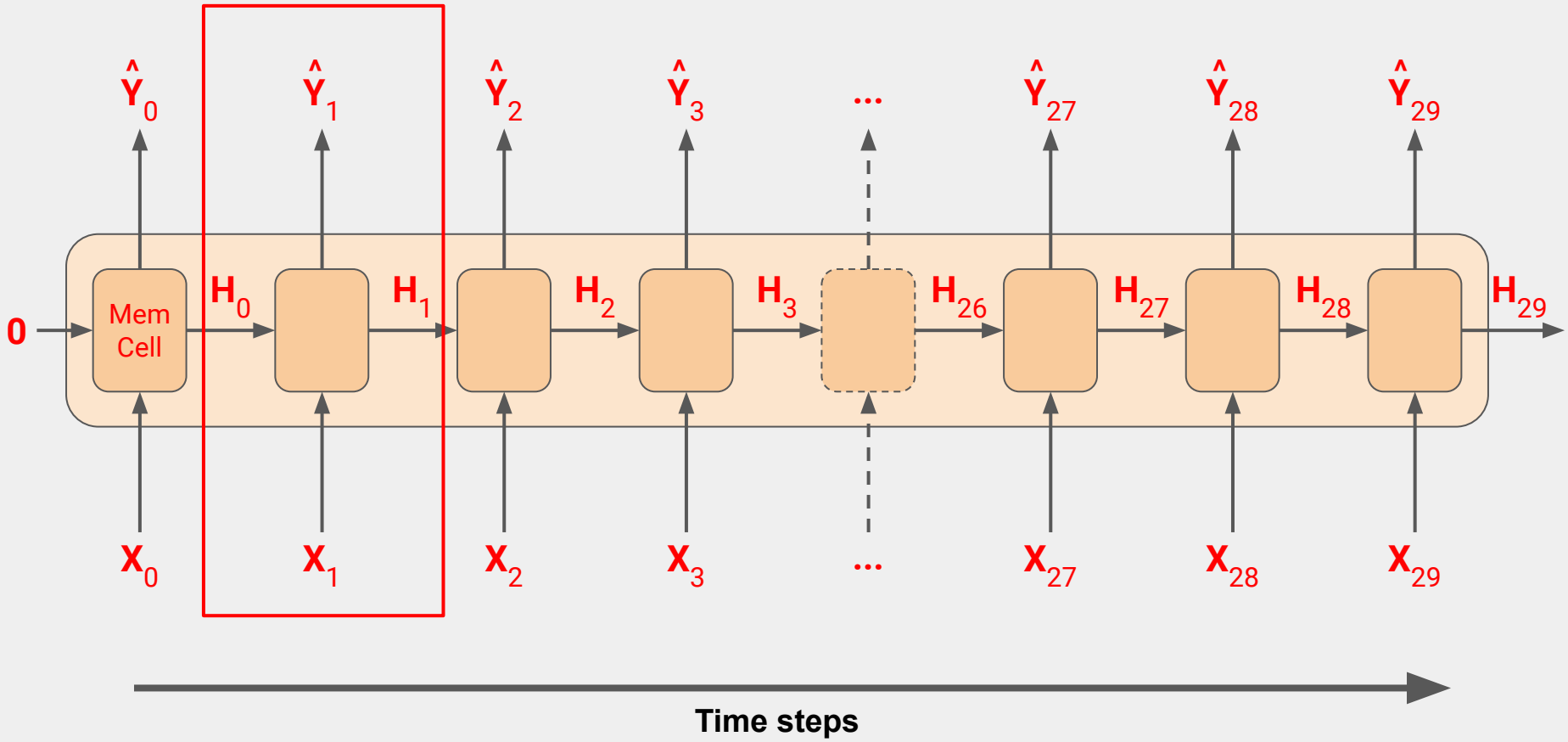


Recurrent Layer

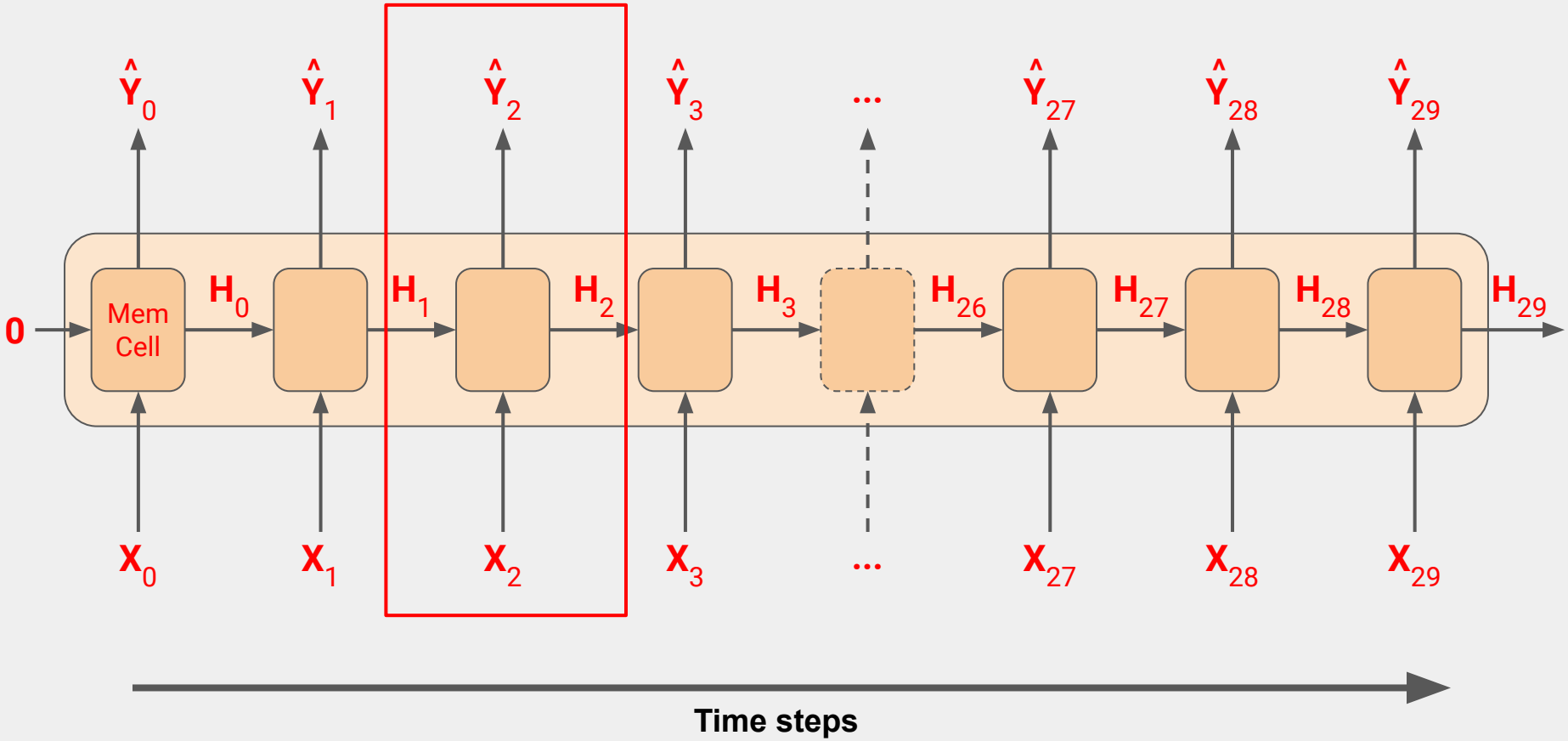


Recurrent Layer

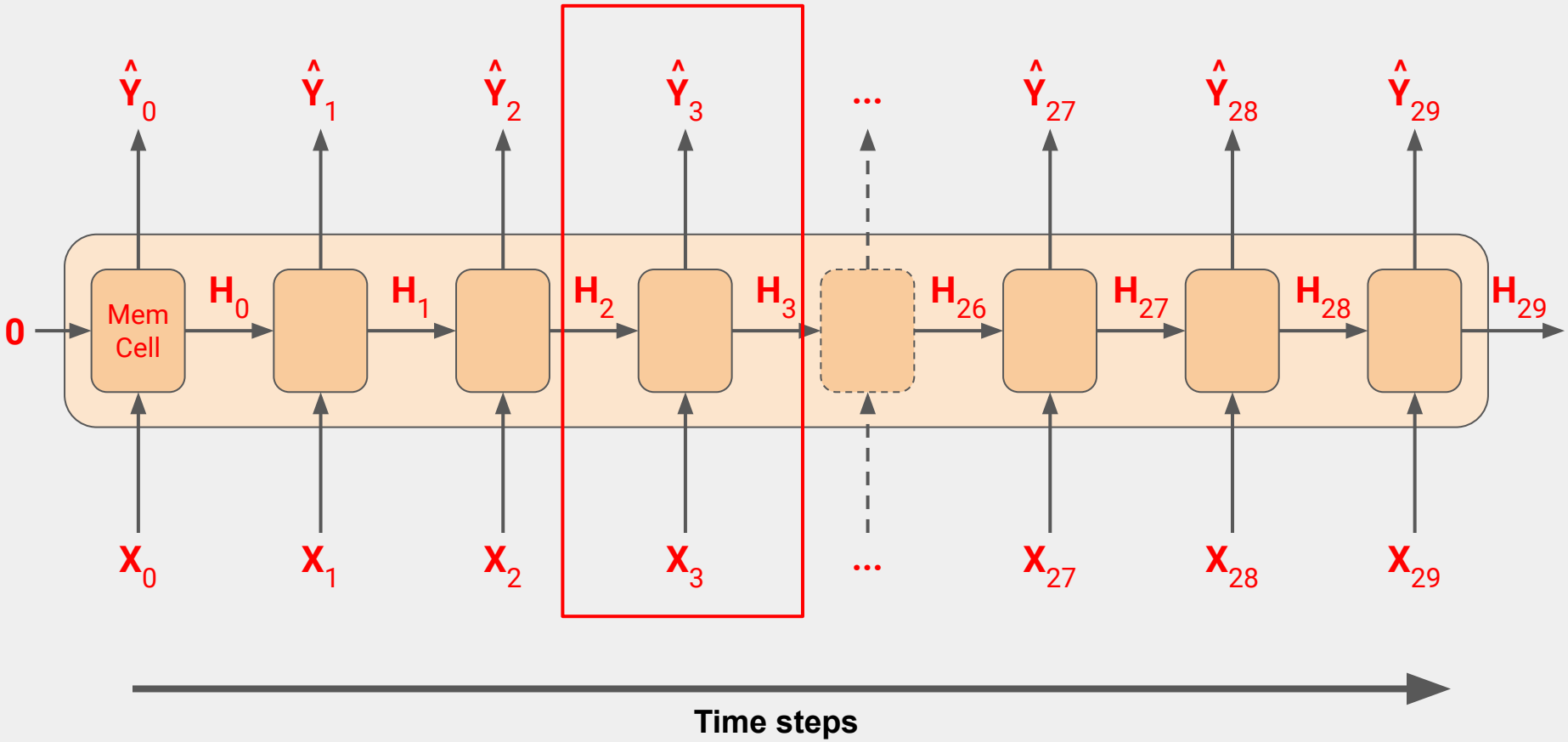



Recurrent Layer
Ŷ0
Ŷ1
Ŷ2
Ŷ3
...
Ŷ27
Ŷ28
Ŷ29
0
Mem Cell
H0
H1
H2
H3
H26
H27
H28
H29
X0
X1
X2
X3
...
X27
X28
X29
Time steps

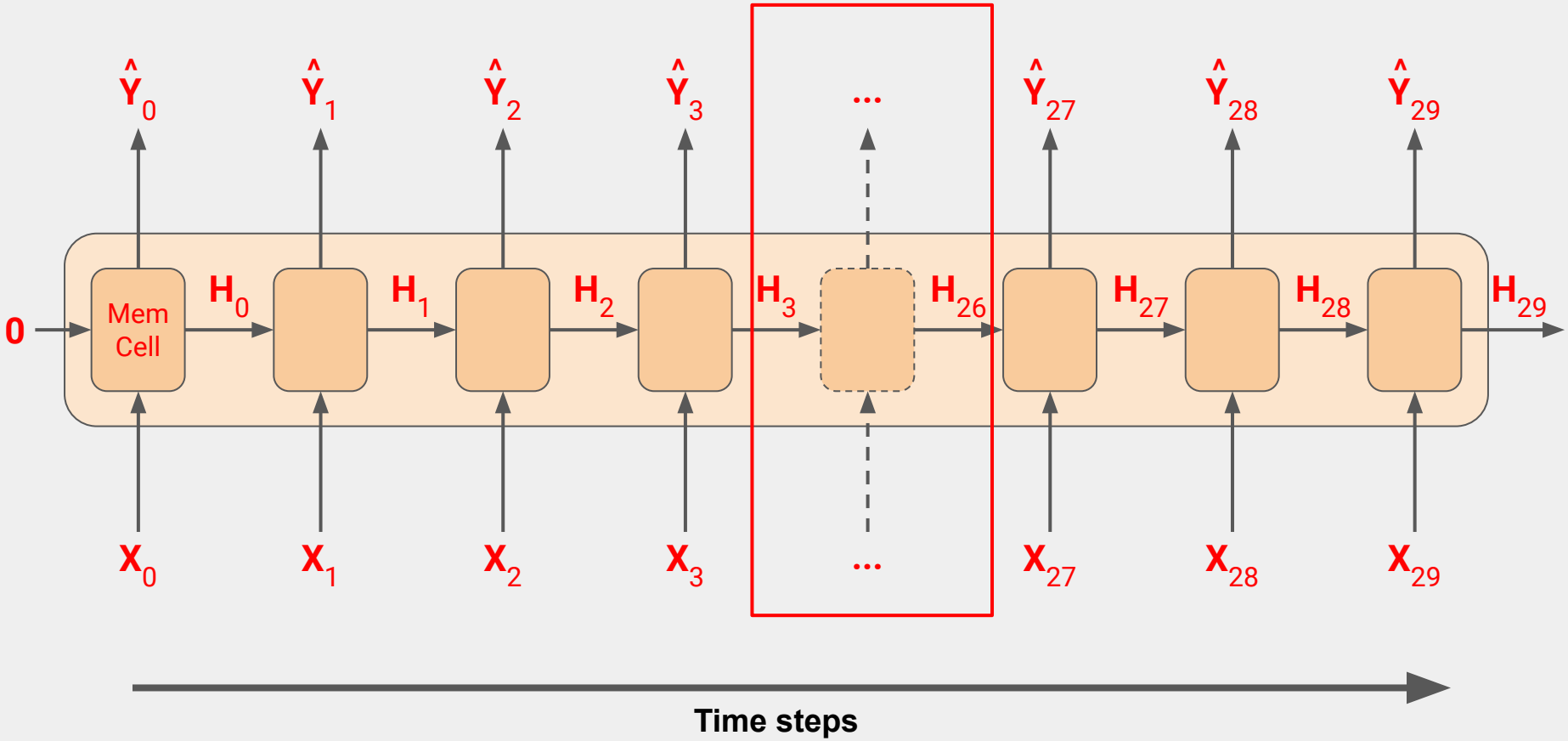
Recurrent Layer



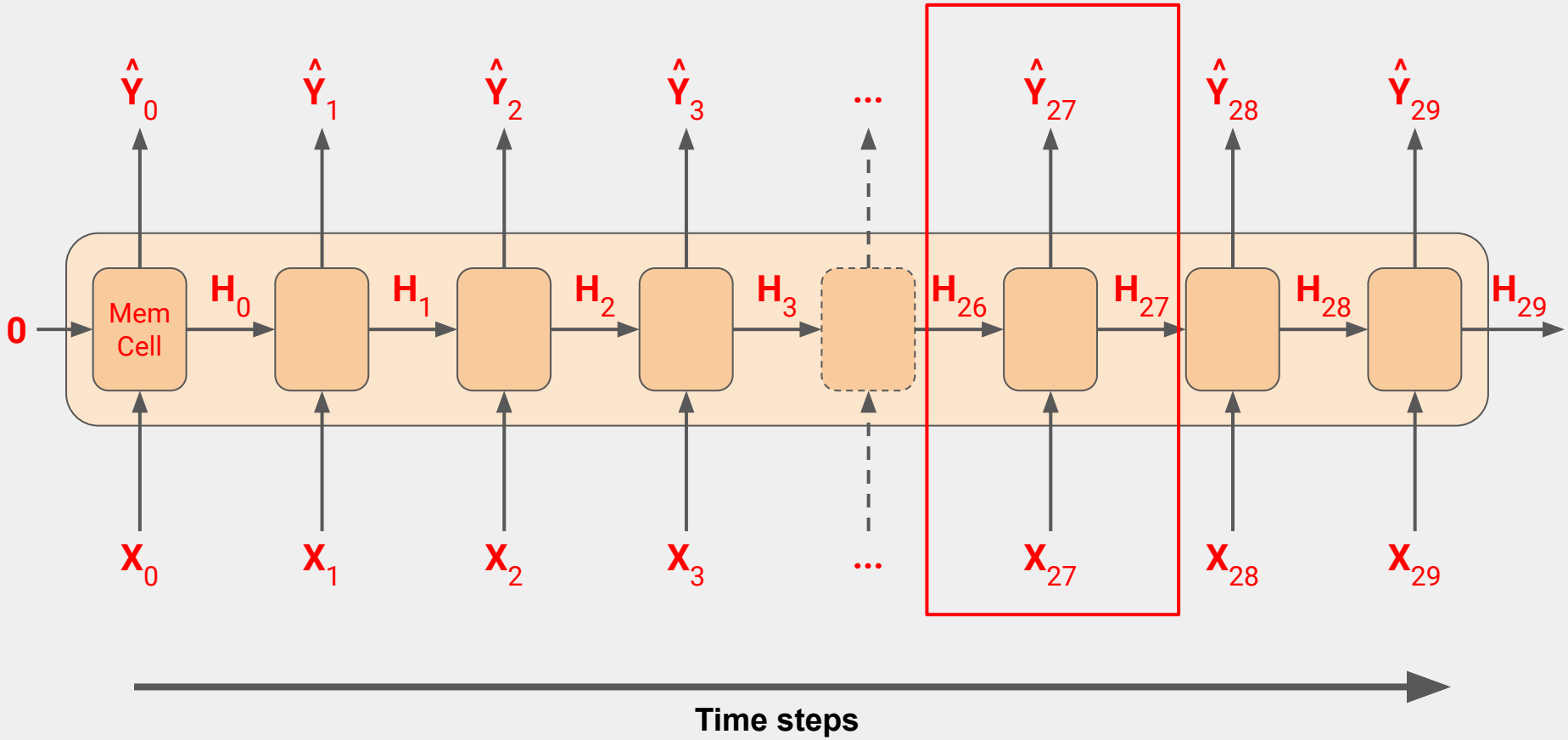
Recurrent Layer



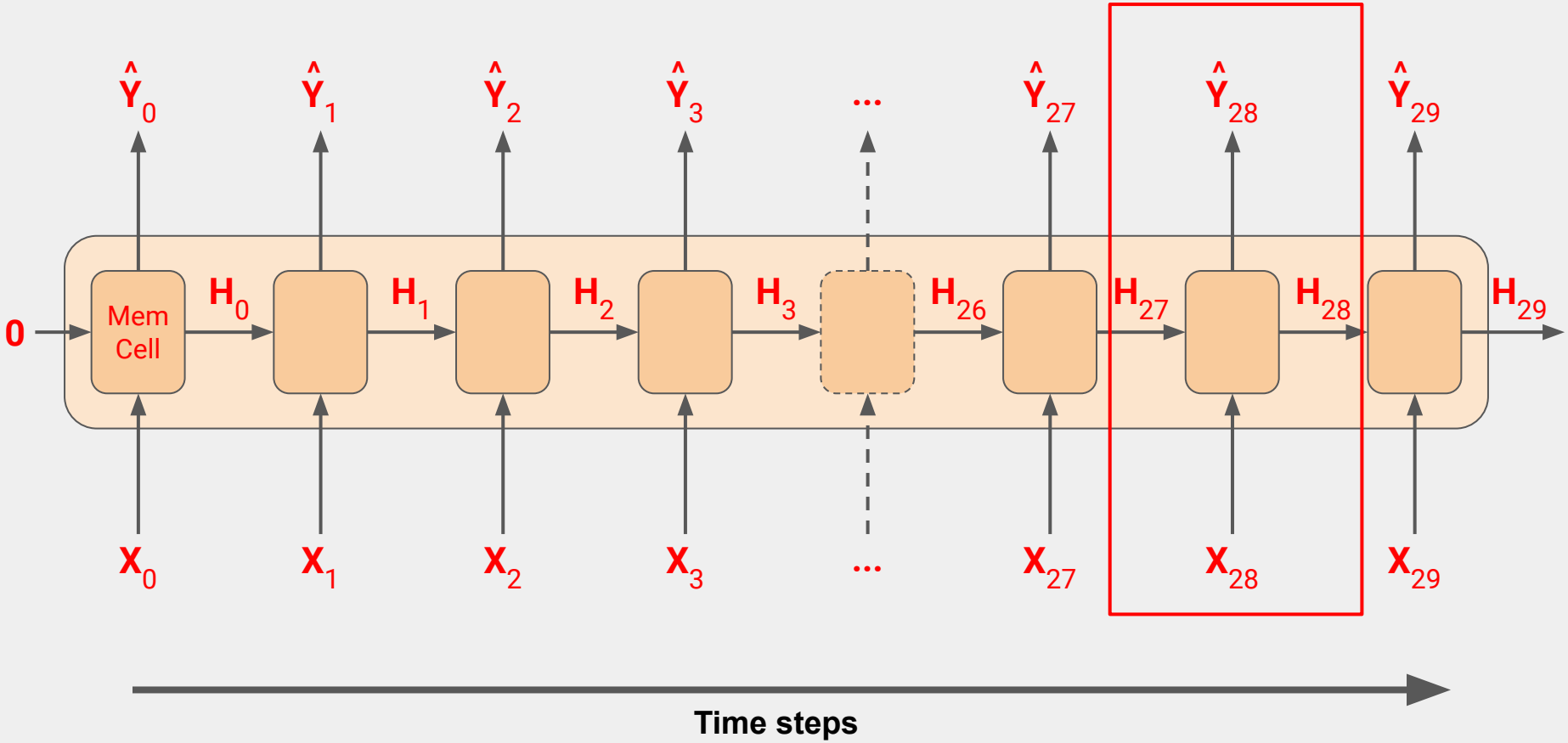
Recurrent Layer



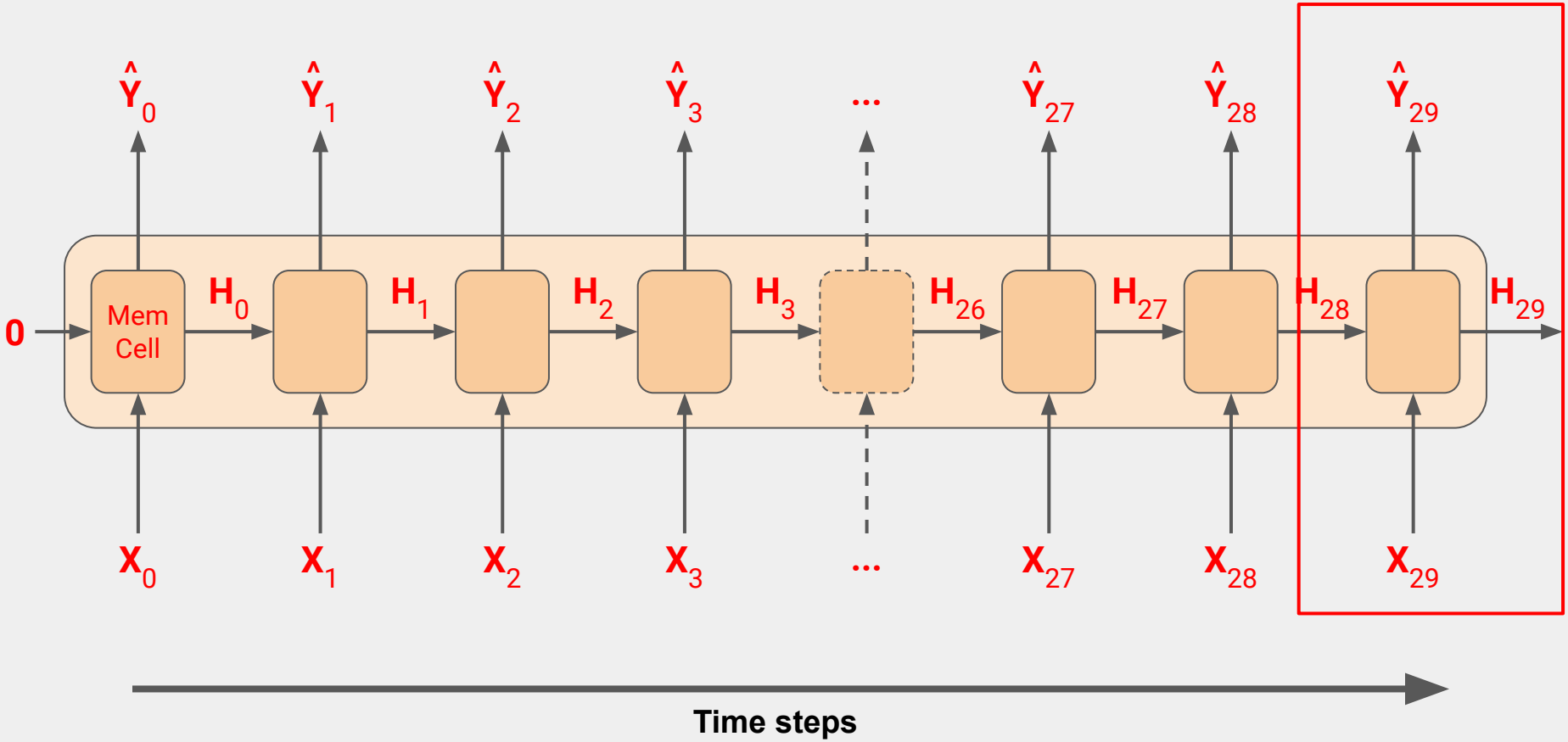
Recurrent Layer



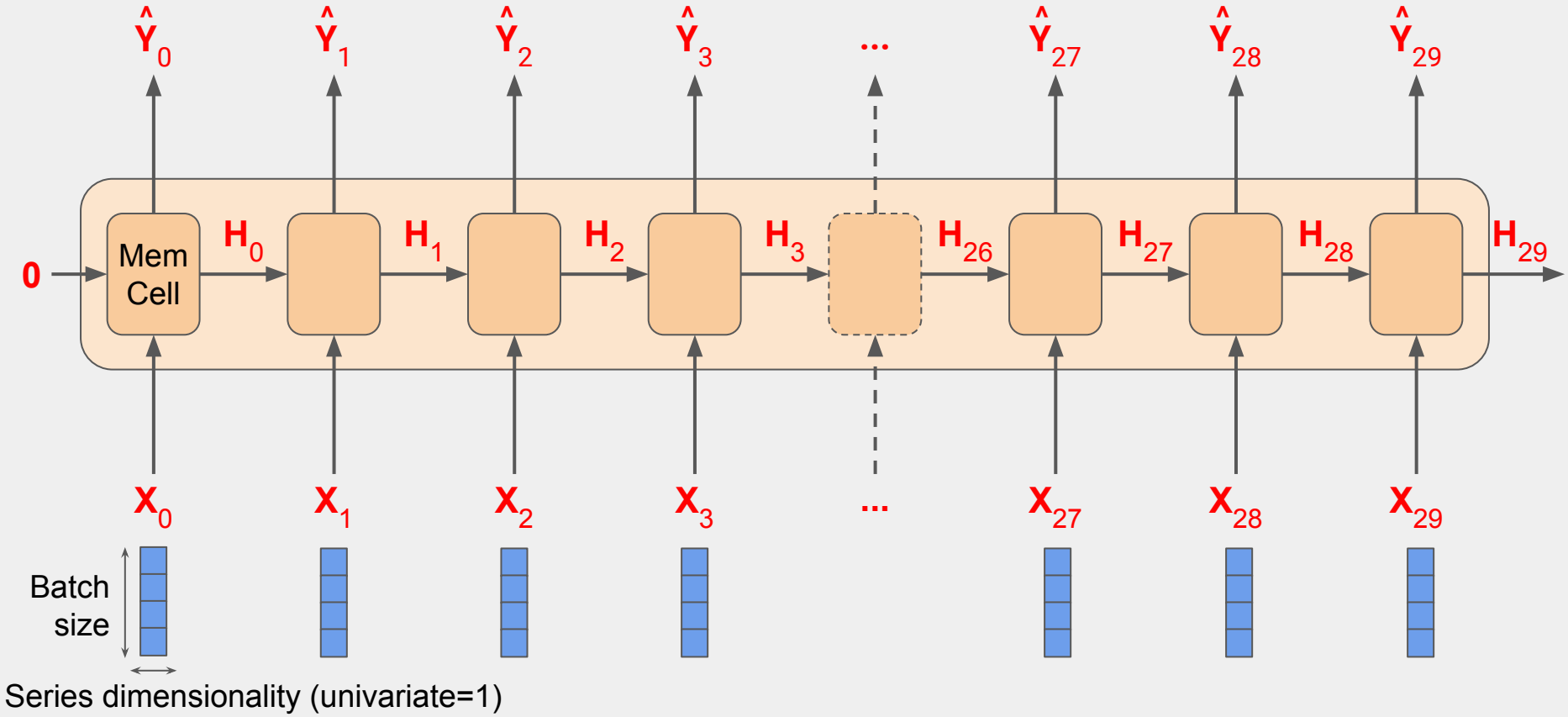
Recurrent Layer



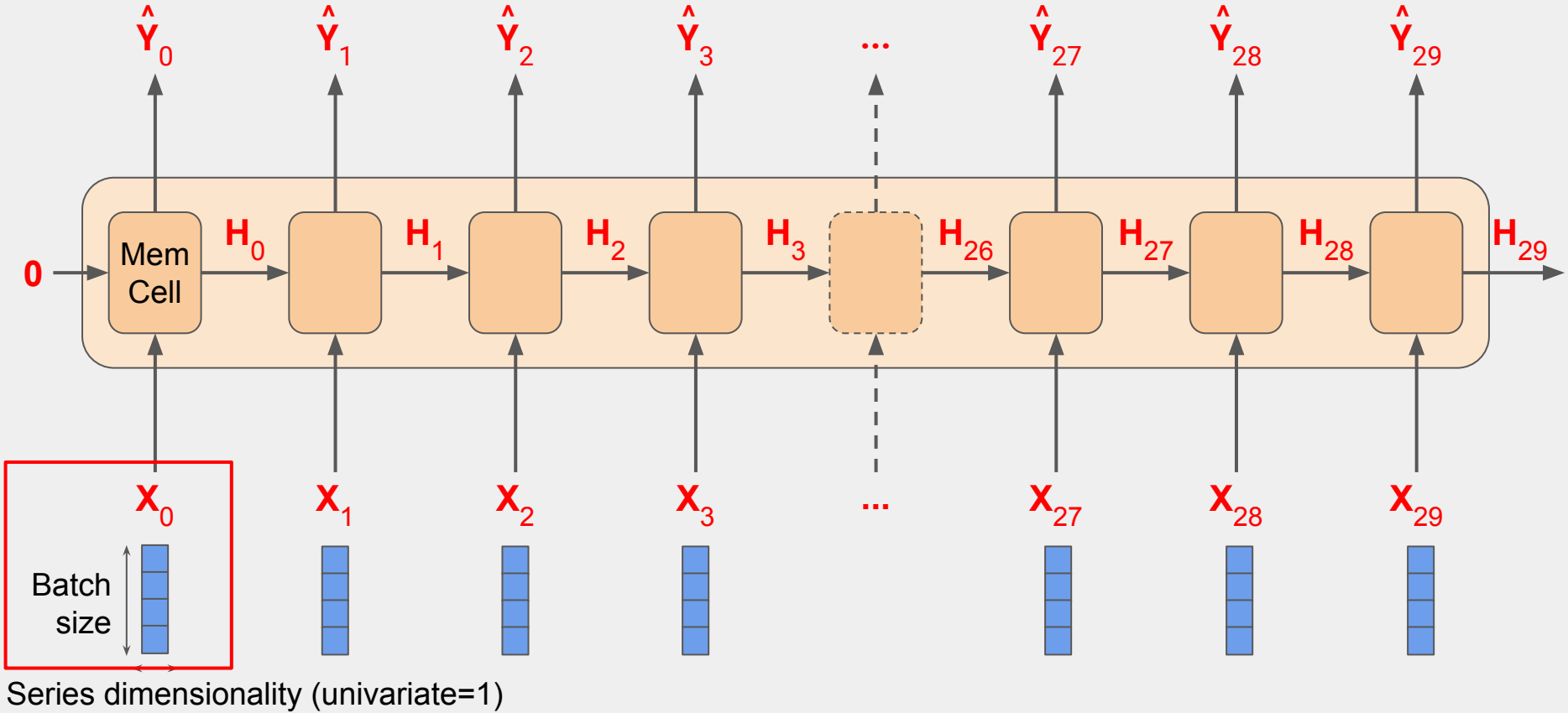
Recurrent Layer

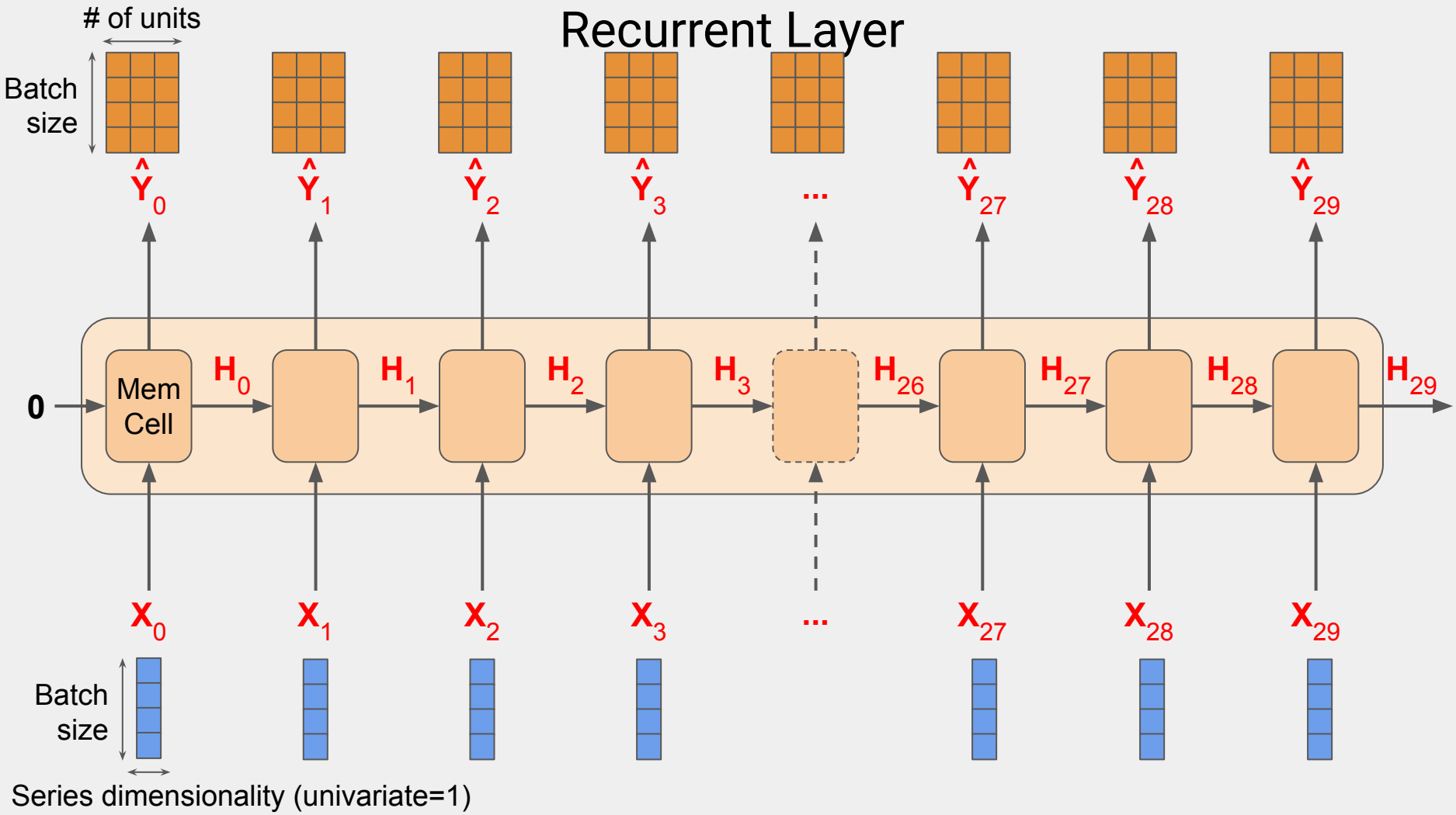


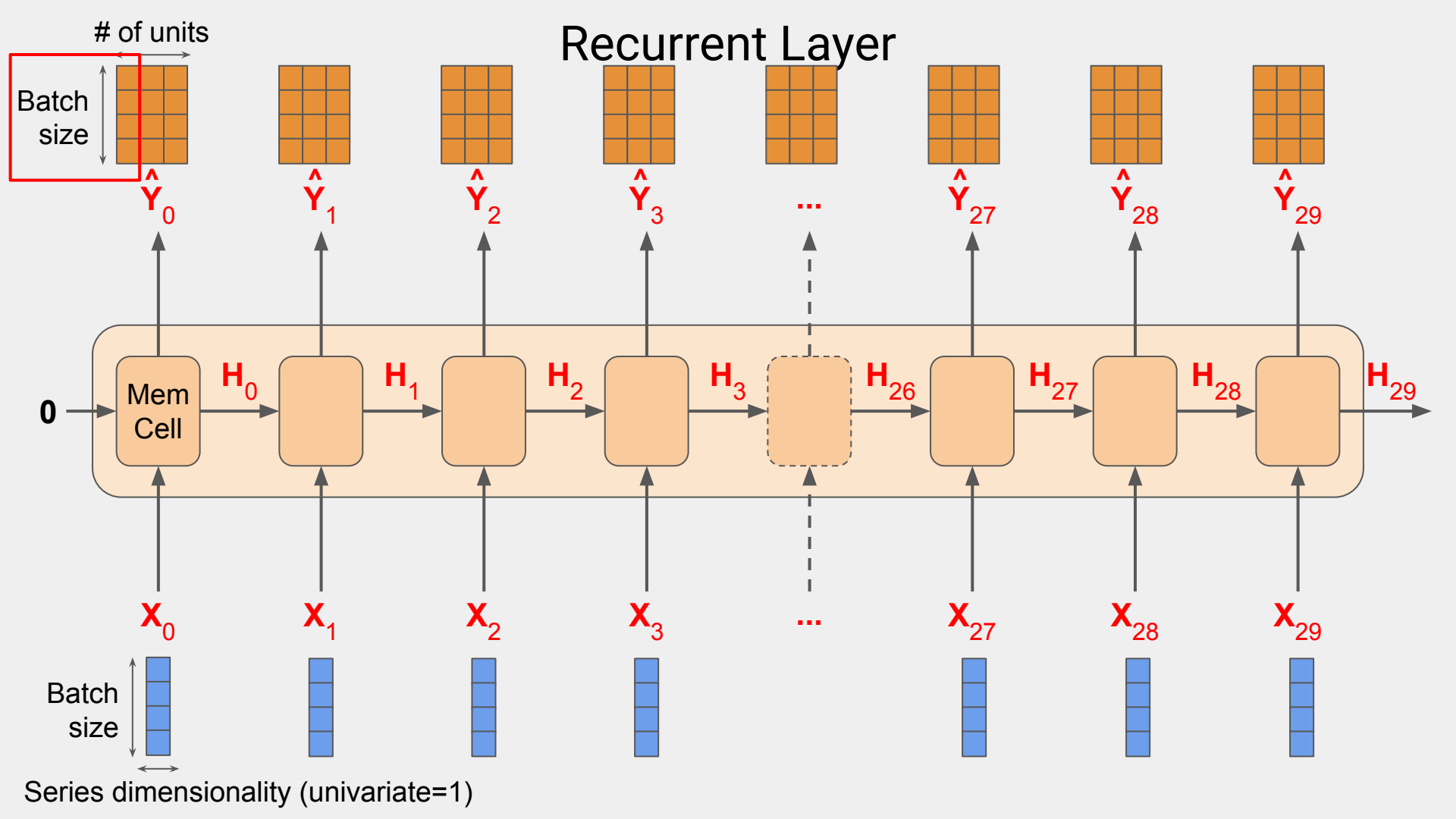
Recurrent Layer

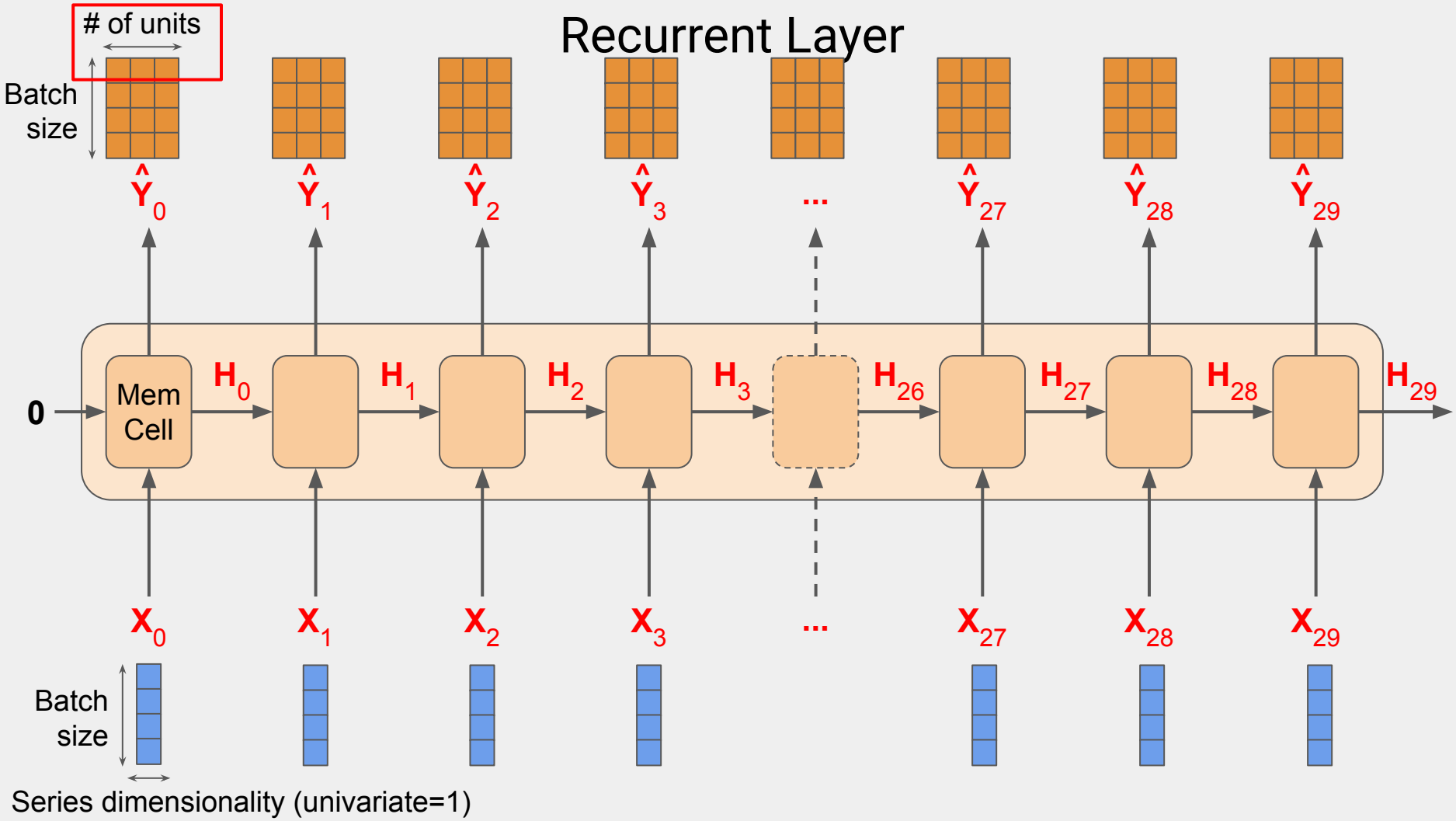


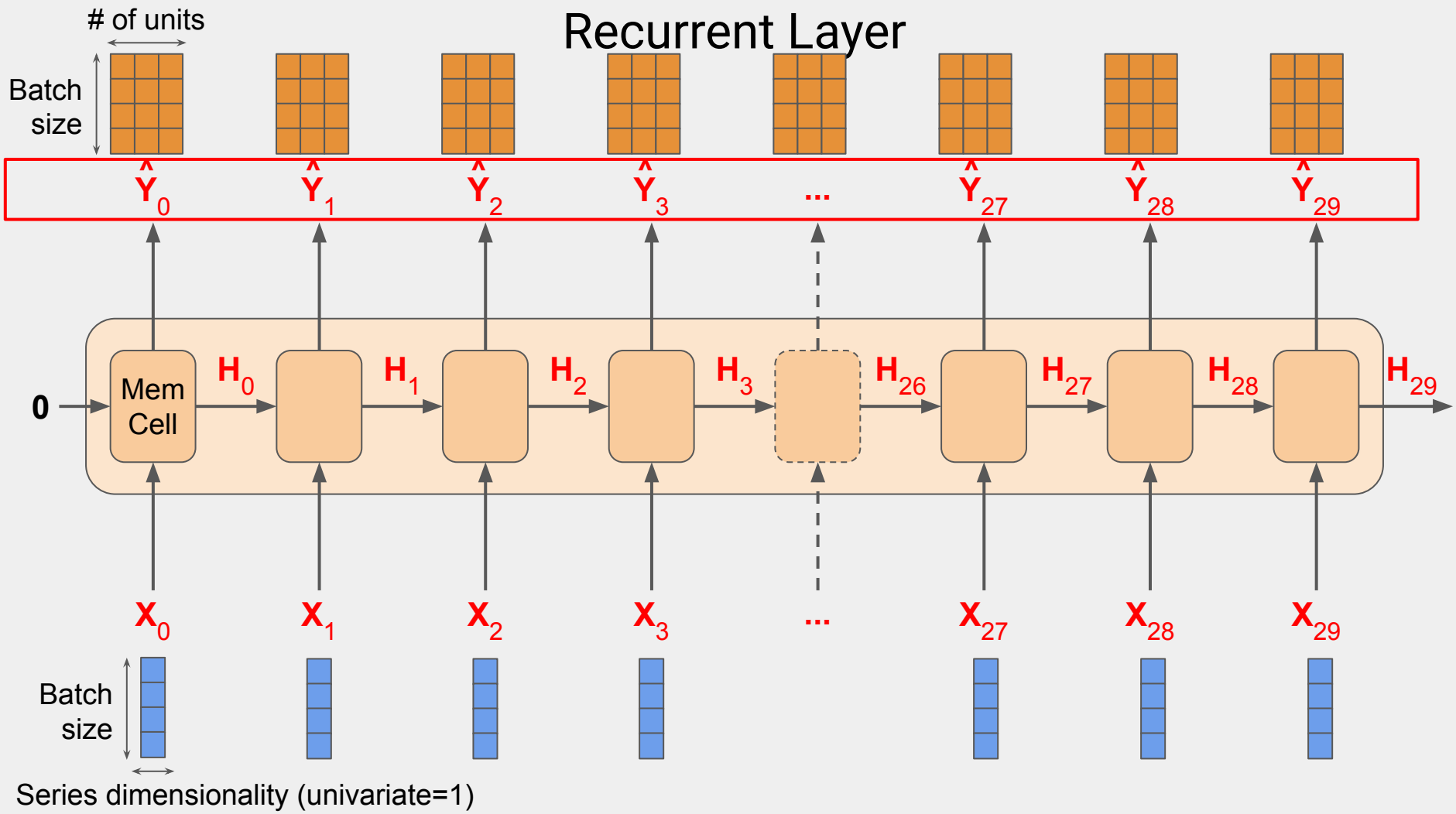
Recurrent Layer

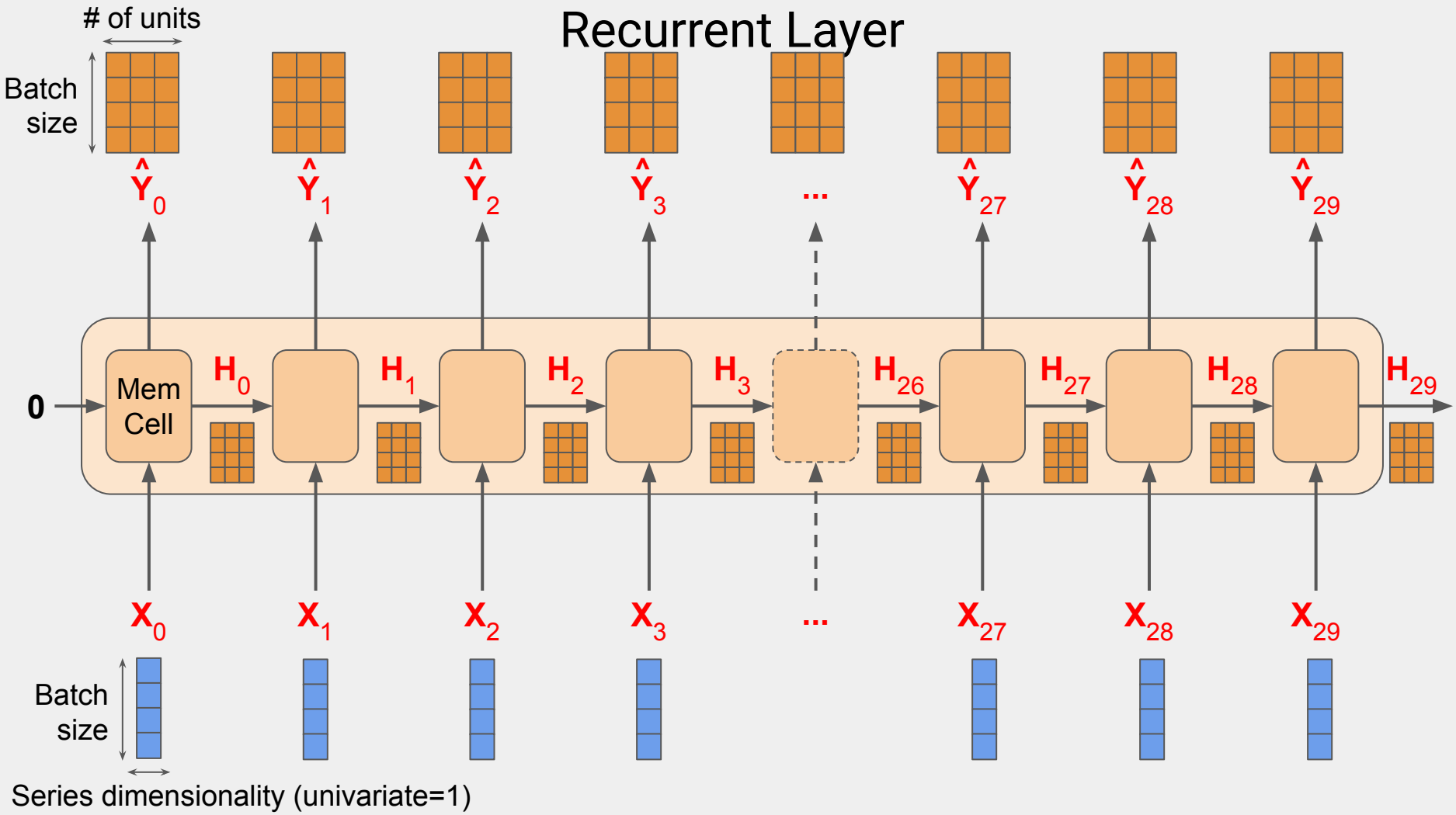


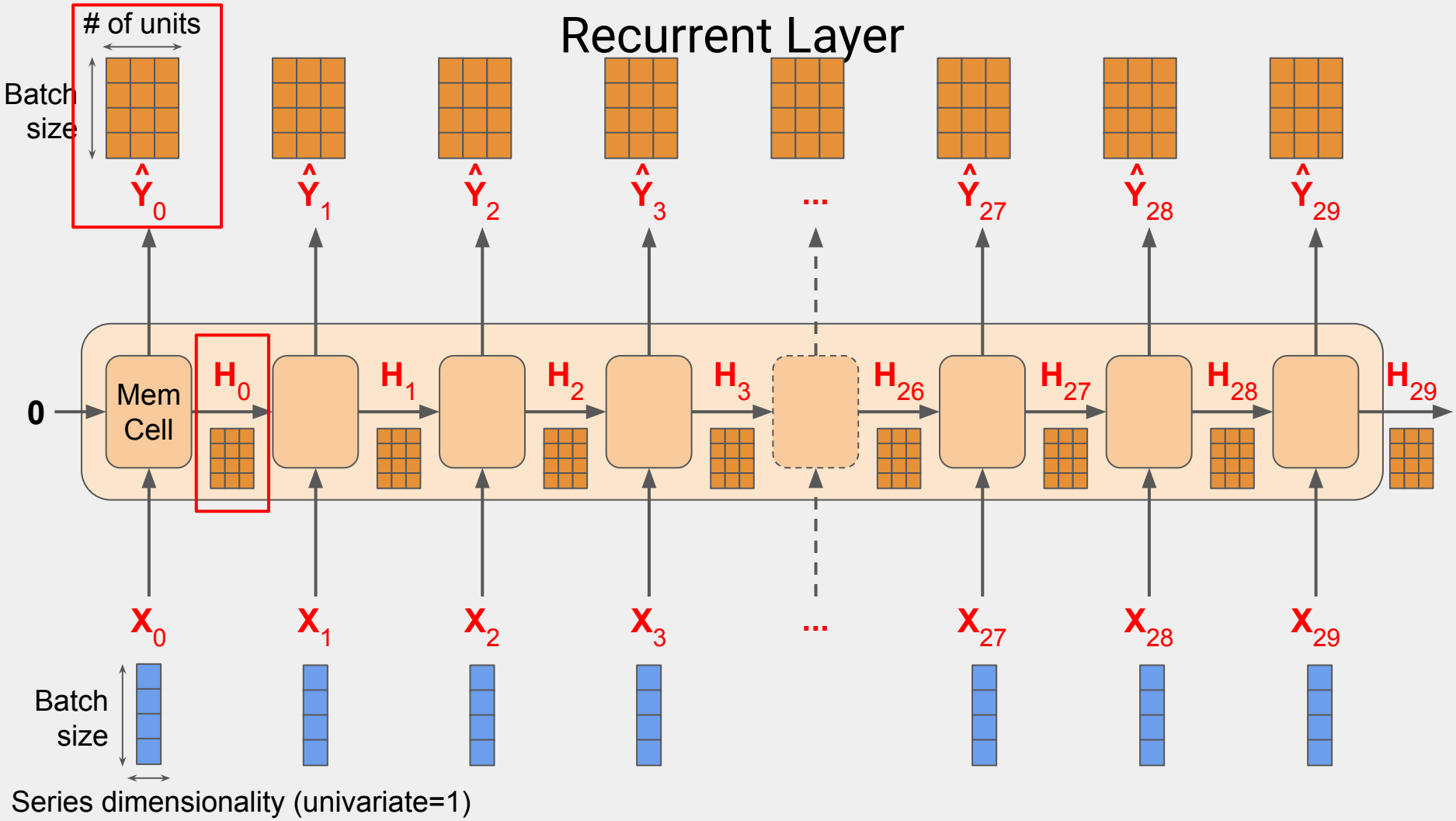

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
Mem Cell
0
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)

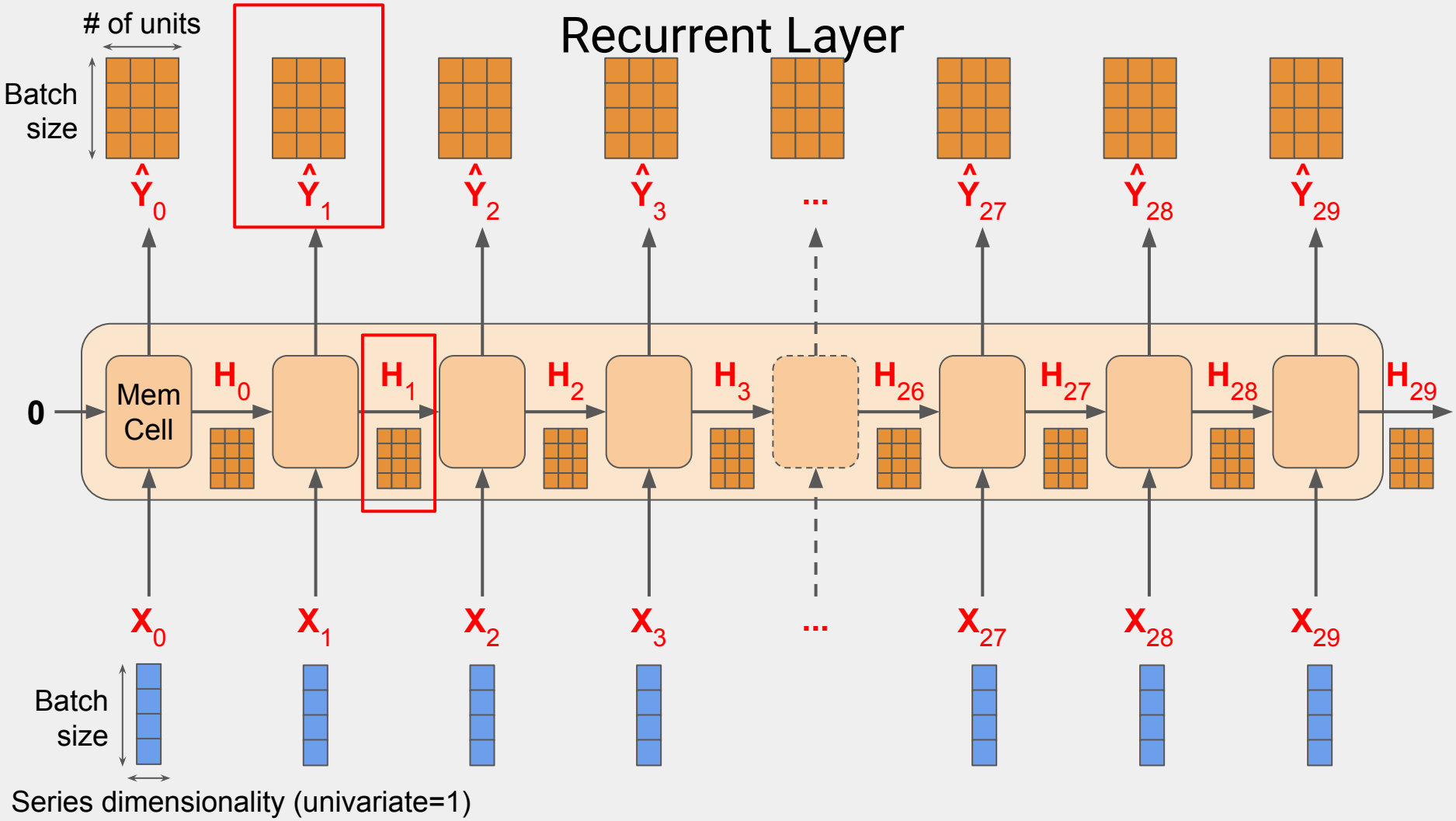

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)


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)

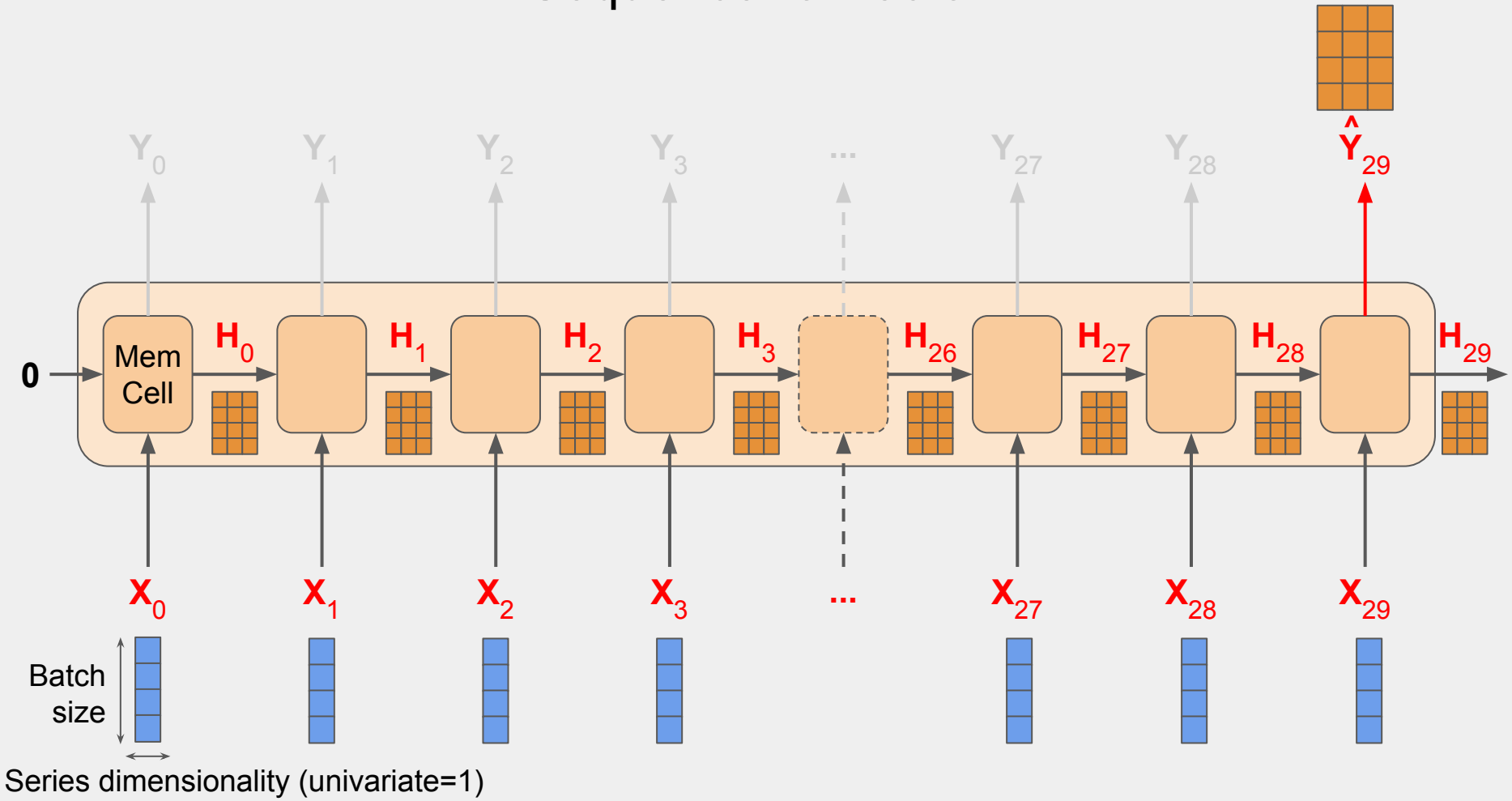

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)


Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
Mem Cell
0
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)

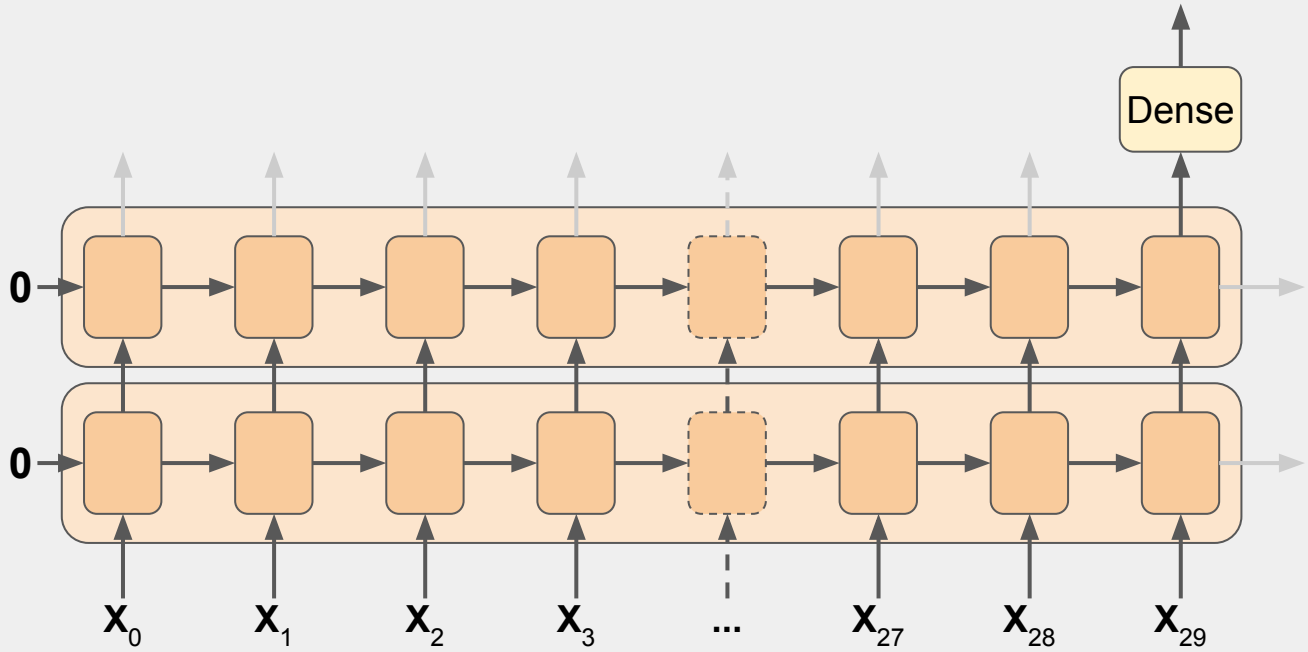

Recurrent Layer
# of units
Batch size
$\hat{Y}_0$
$\hat{Y}_1$
$\hat{Y}_2$
$\hat{Y}_3$
...
$\hat{Y}_{27}$
$\hat{Y}_{28}$
$\hat{Y}_{29}$
0
Mem Cell
H_0
H_1
H_2
H_3
H_{26}
H_{27}
H_{28}
H_{29}
X_0
X_1
X_2
X_3
...
X_{27}
X_{28}
X_{29}
Batch size
Series dimensionality (univariate=1)


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)

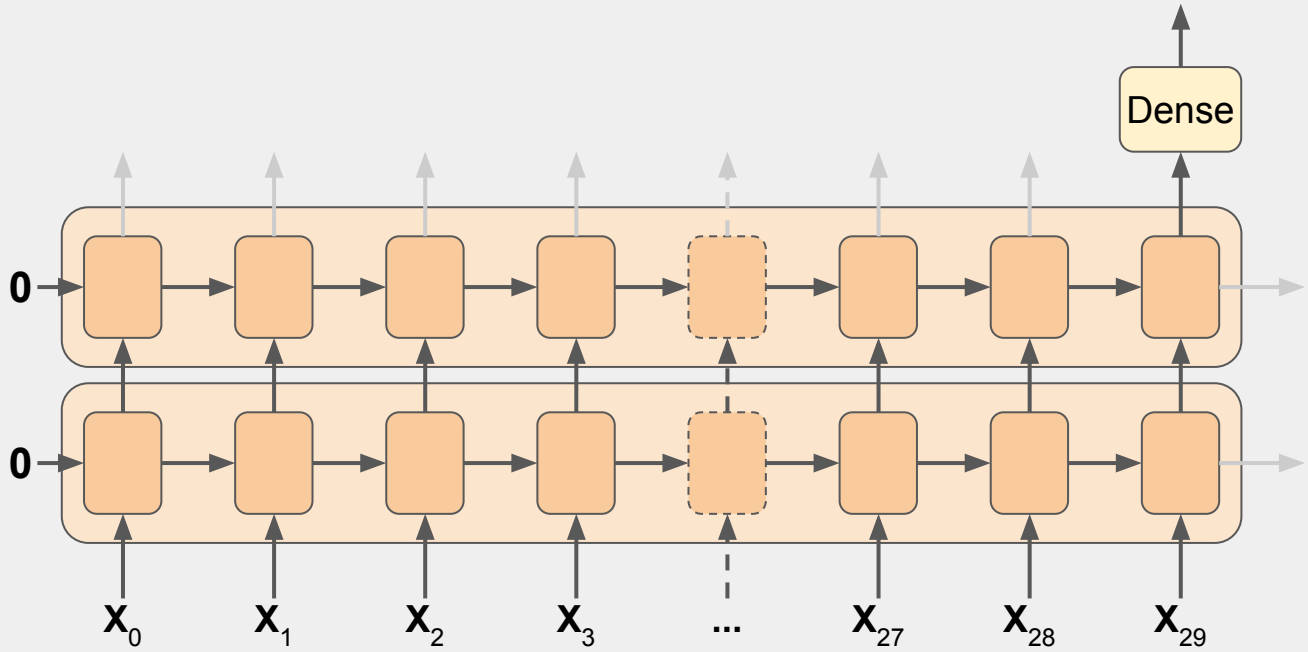
Sequence-to-Vector



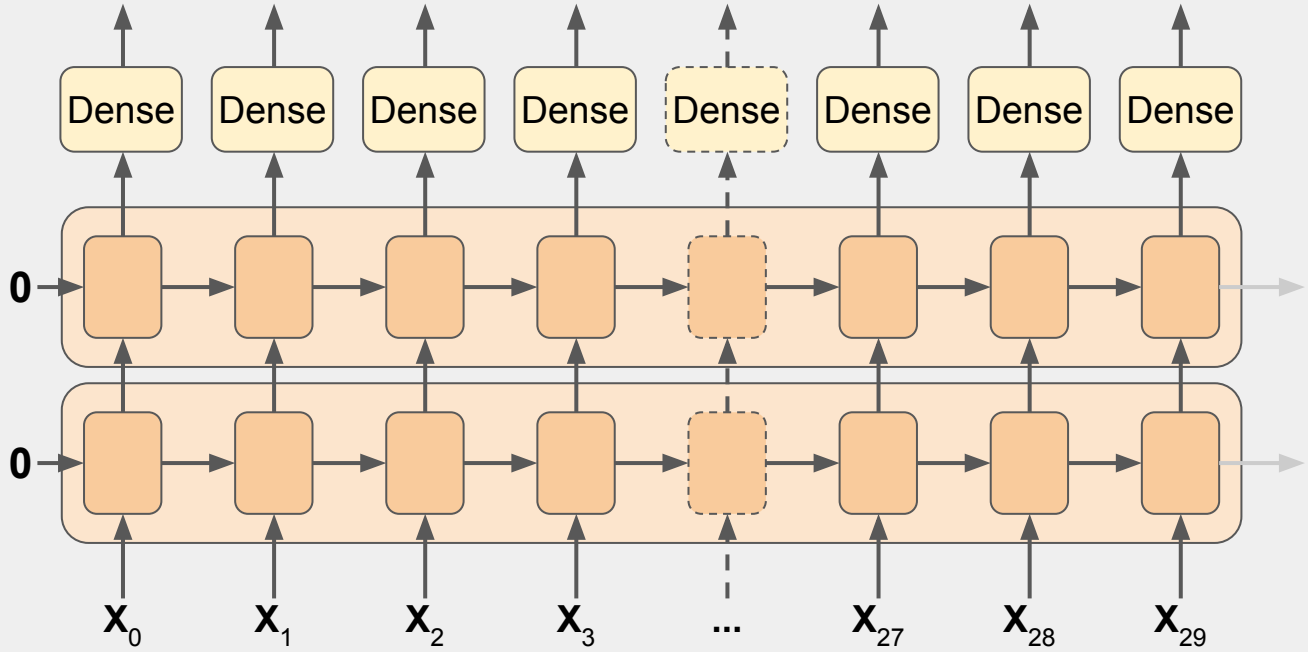
```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size, 1)),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.SimpleRNN(20),
  tf.keras.layers.Dense(1)
])
```



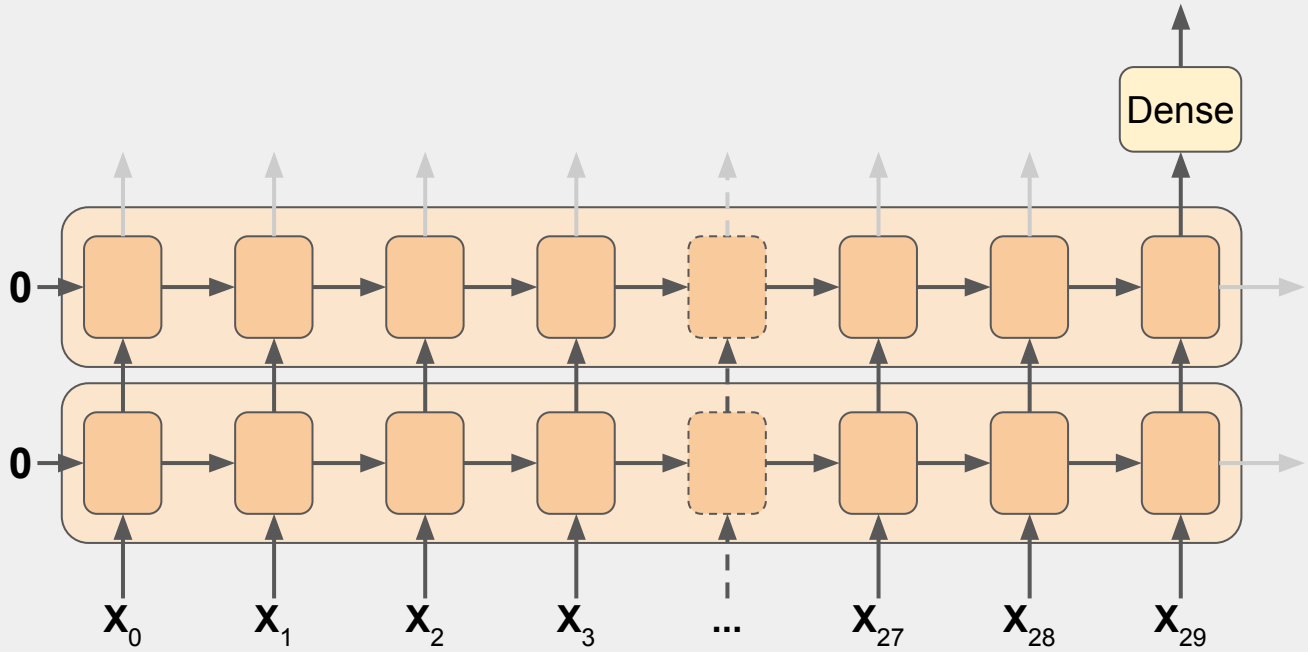
```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size, 1)),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.SimpleRNN(20),
  tf.keras.layers.Dense(1)
])
```



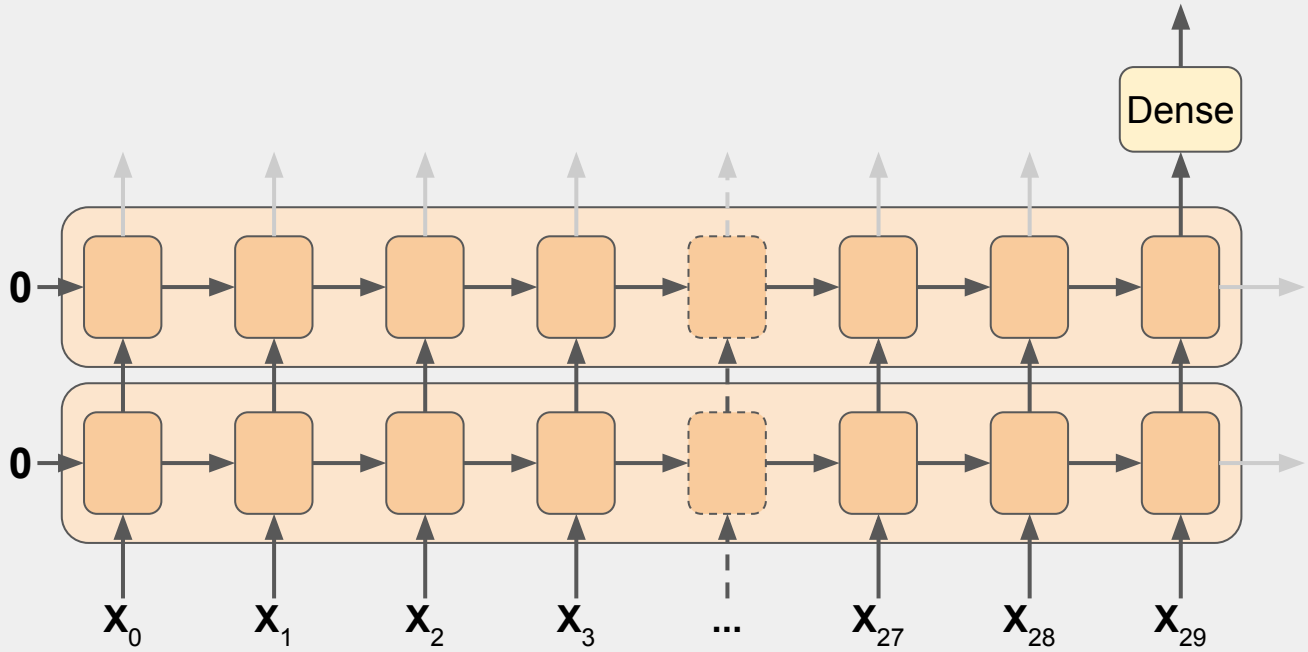
```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size, 1)),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.Dense(1)
])
```



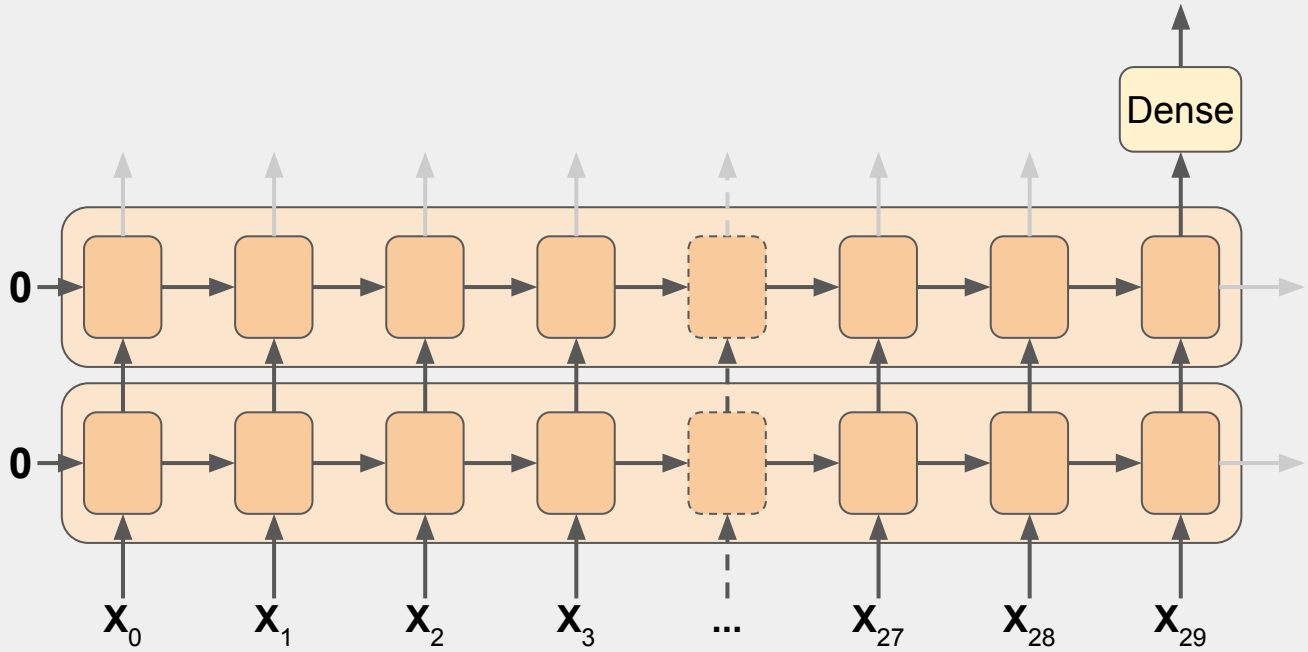
```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size, 1)),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.SimpleRNN(20),
  tf.keras.layers.Dense(1)
])
```



```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size, 1)),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.SimpleRNN(20),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])
```



```
model = tf.keras.models.Sequential([
  tf.keras.Input(shape=(window_size, 1)),
  tf.keras.layers.SimpleRNN(20, return_sequences=True),
  tf.keras.layers.SimpleRNN(20),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])
```



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

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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(dataset, 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.Input(shape=(window_size, 1)),
    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(dataset, epochs=100, callbacks=[lr_schedule])
```

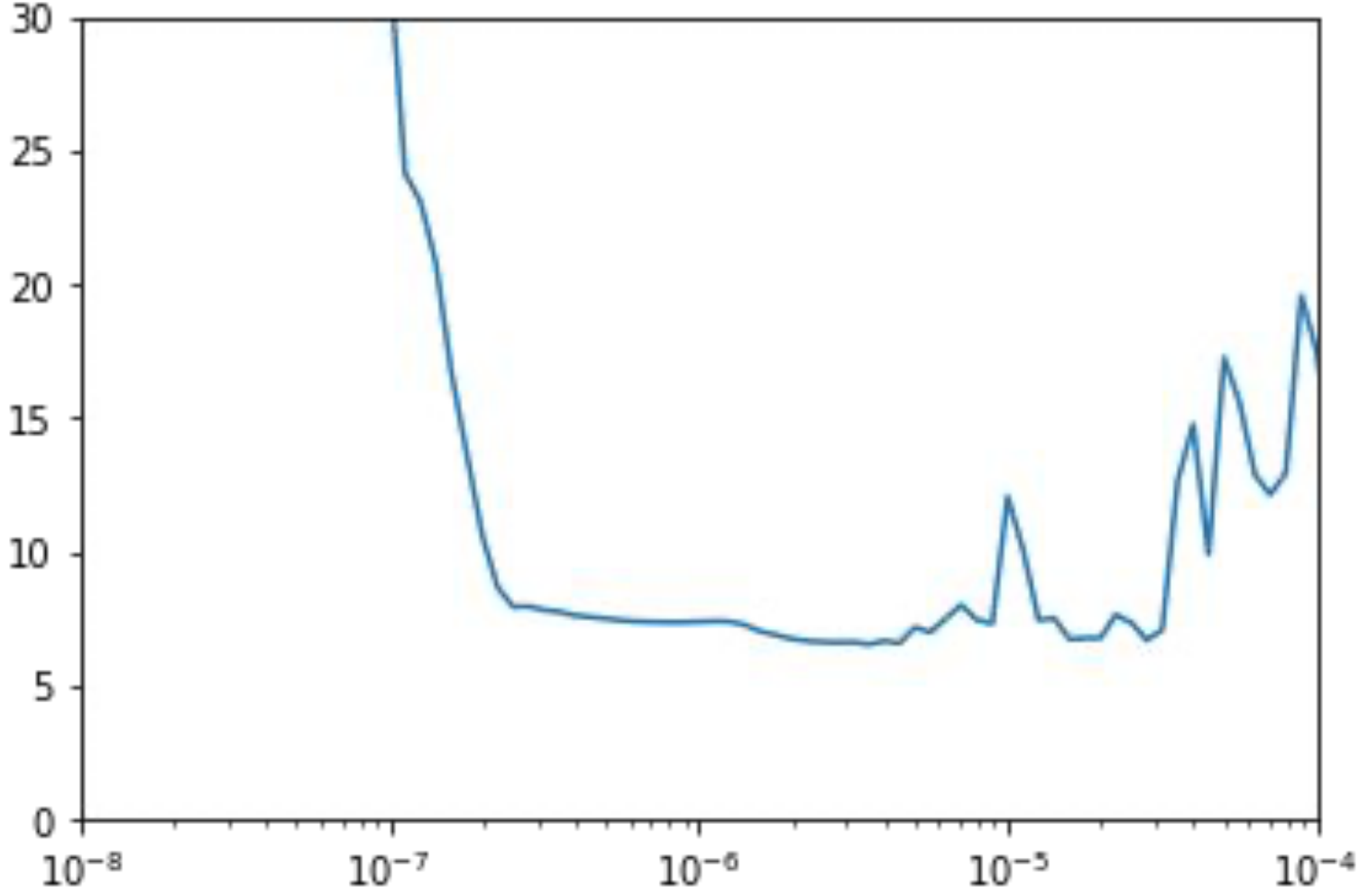
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size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

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

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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(dataset, epochs=100, callbacks=[lr_schedule])
```

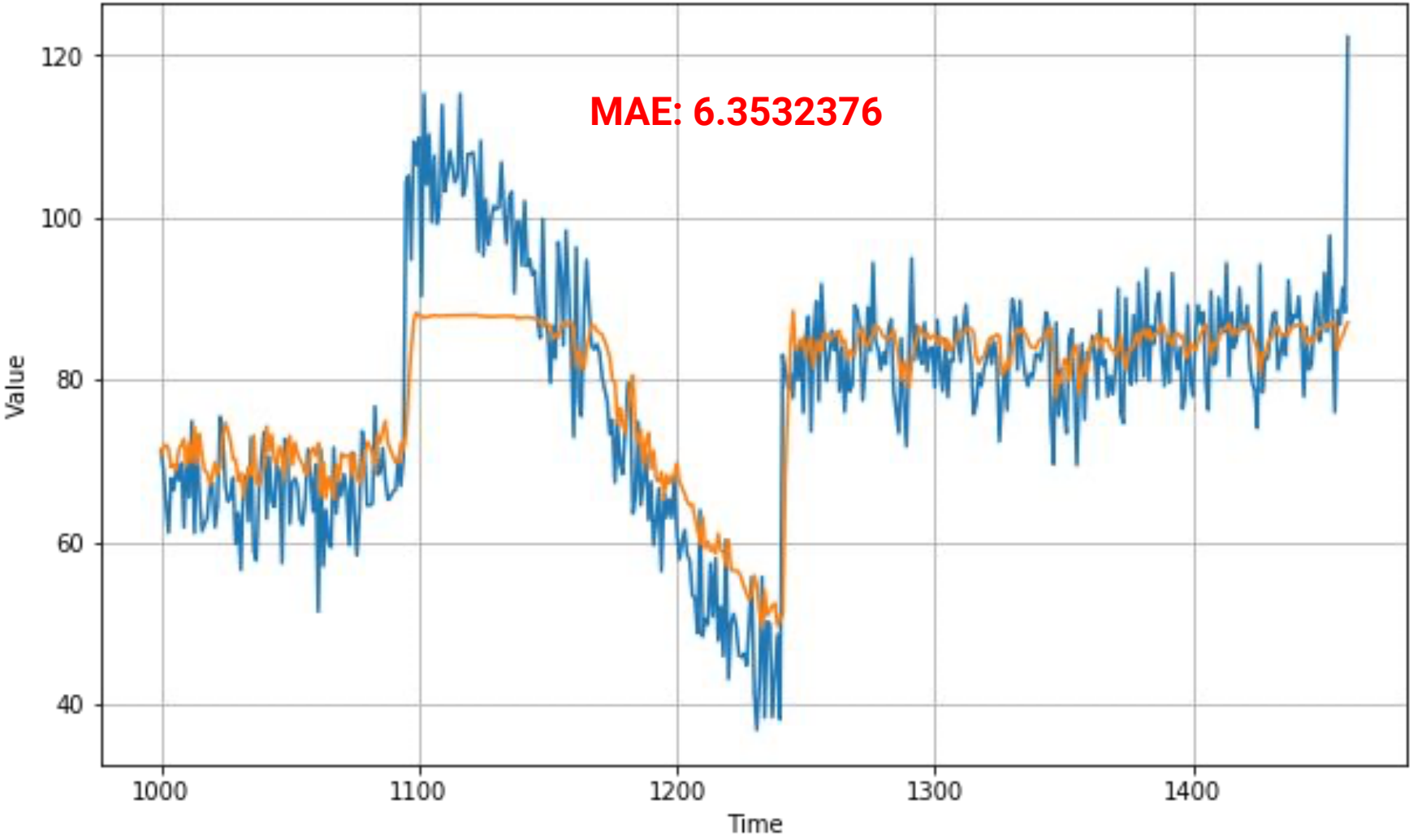
```
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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(dataset, epochs=100, callbacks=[lr_schedule])
```

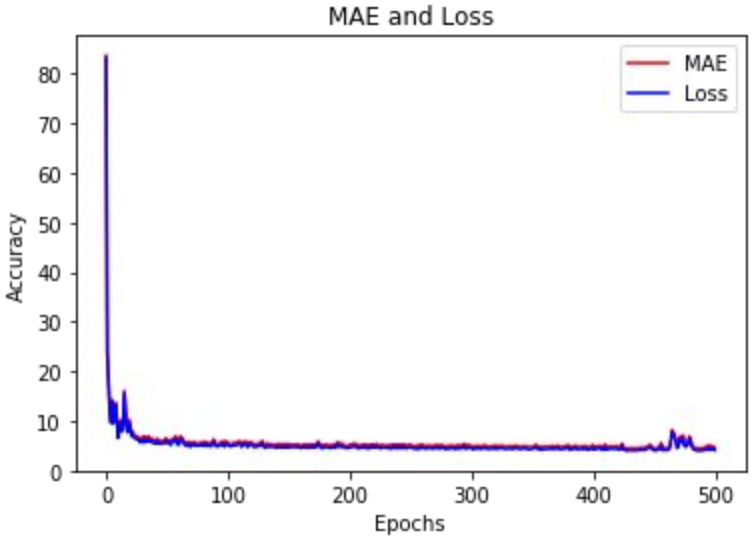
```
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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(dataset, epochs=100, callbacks=[lr_schedule])
```

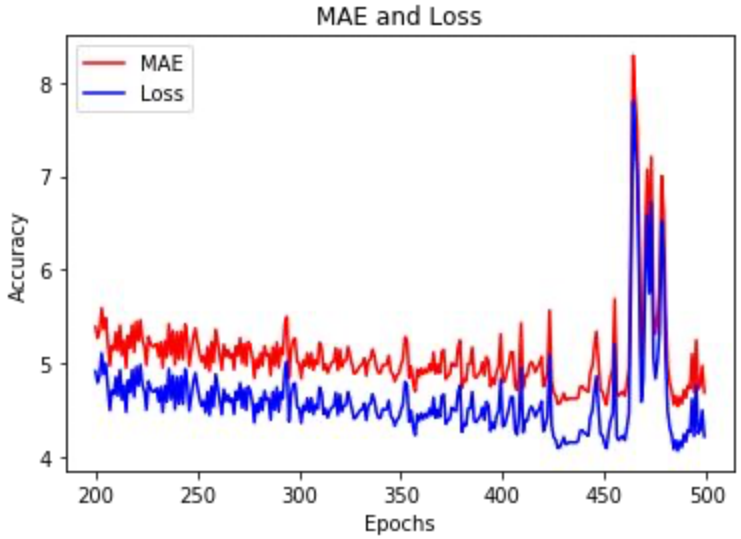
https://en.wikipedia.org/wiki/Huber_loss

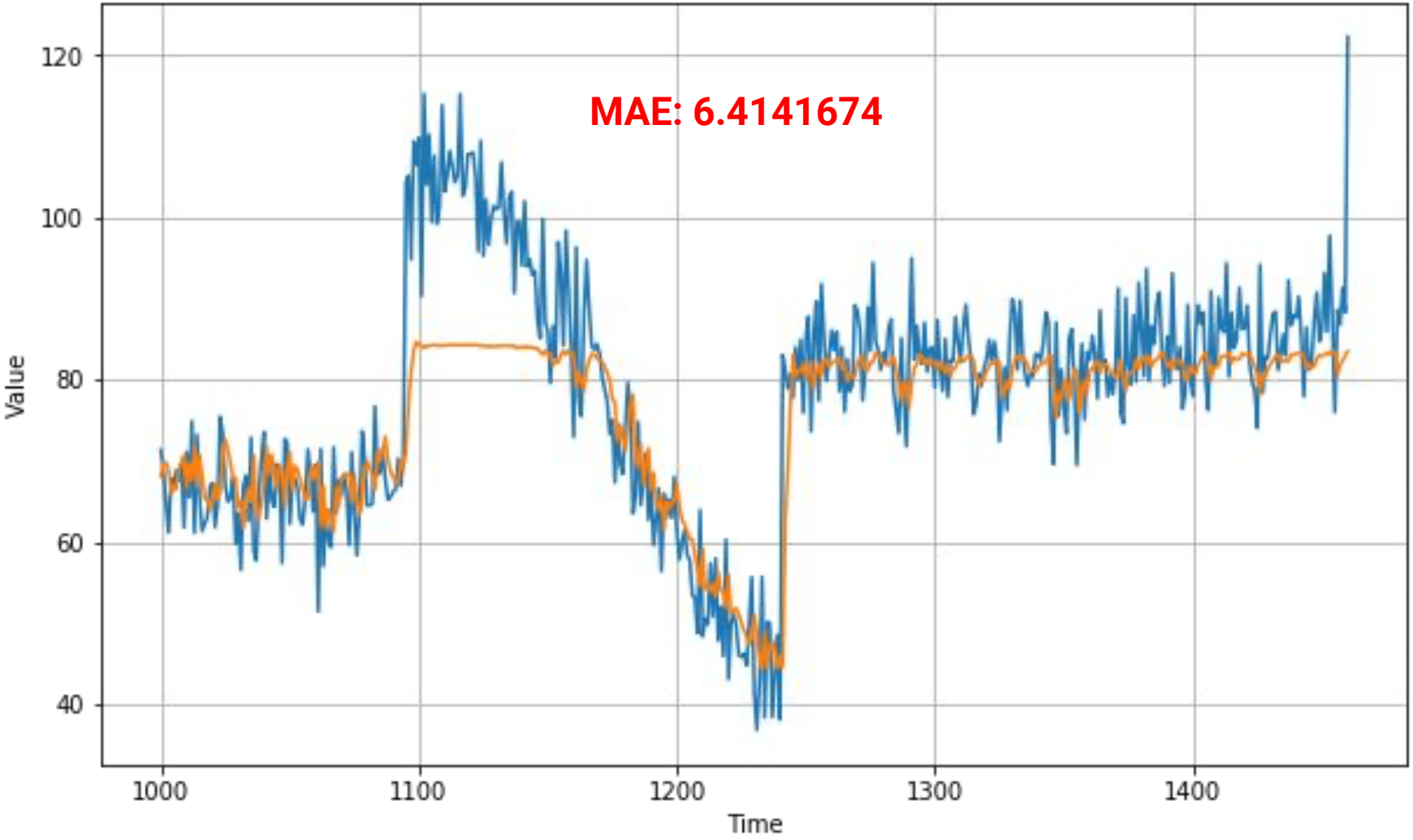

30
25
20
15
10
5
0
10^{-8}
10^{-7}
10^{-6}
10^{-5}
10^{-4}

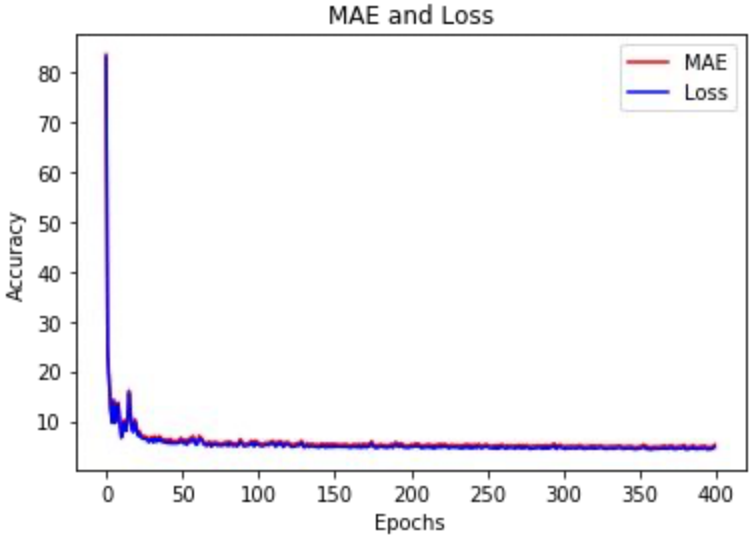
```
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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)
```

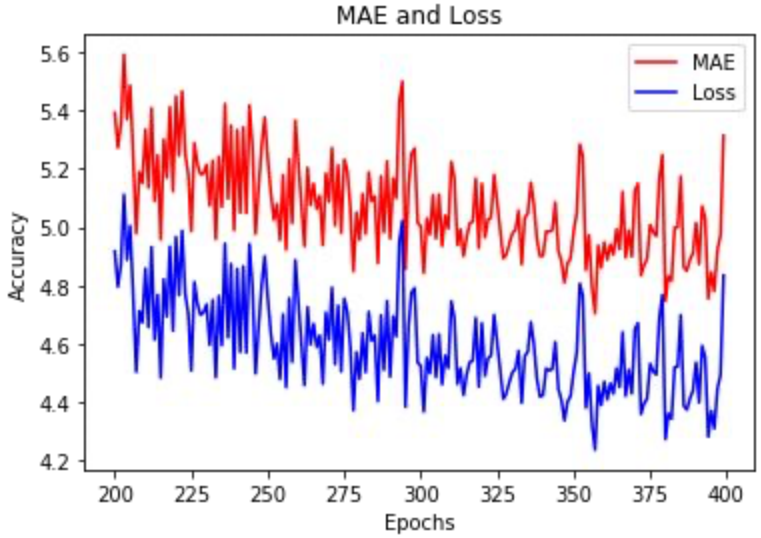

MAE: 6.3532376
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time

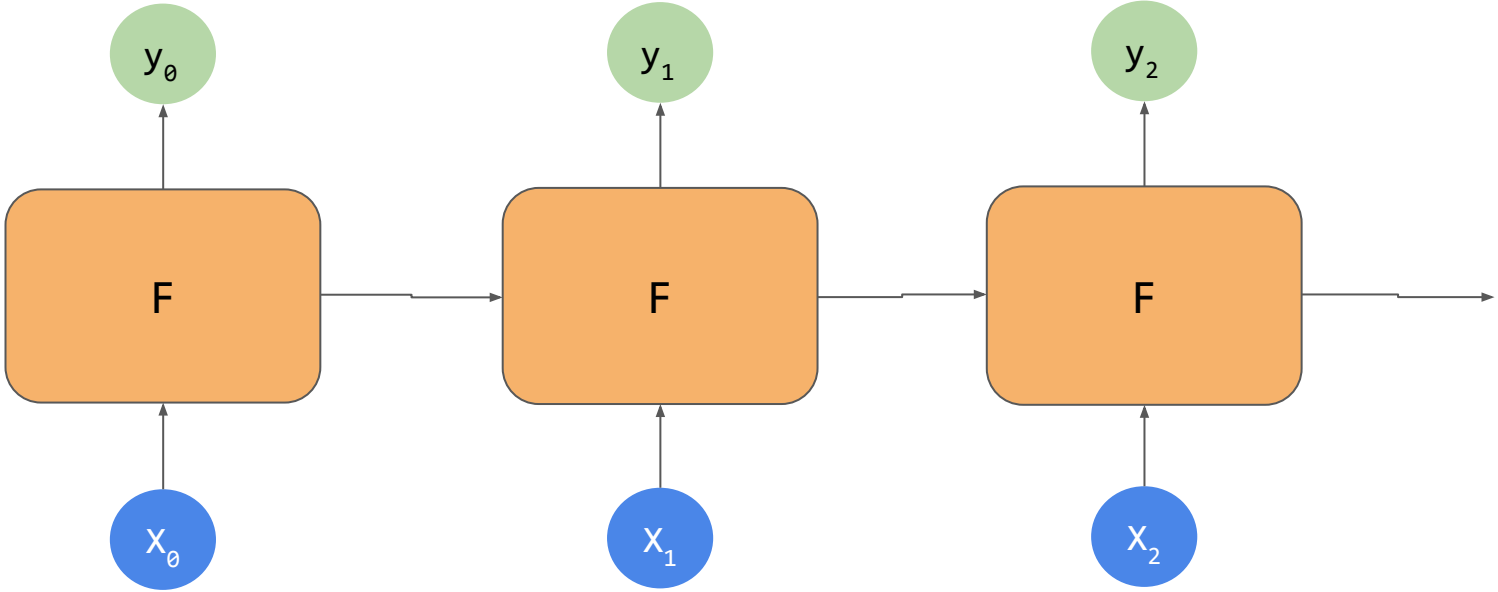

MAE and Loss
MAE
Loss
Accuracy
Epochs
0
10
20
30
40
50
60
70
80
0
100
200
300
400
500

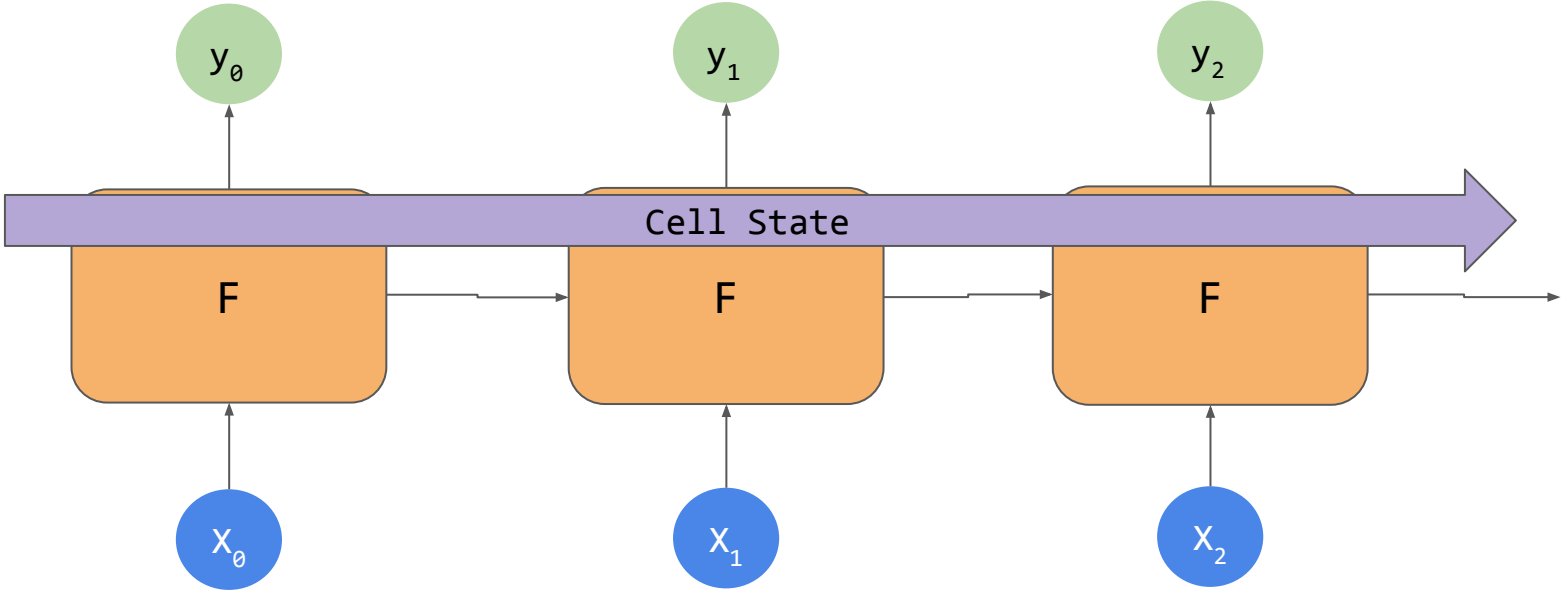

MAE and Loss
MAE
Loss
Accuracy
Epochs
4
5
6
7
8
200
250
300
350
400
450
500

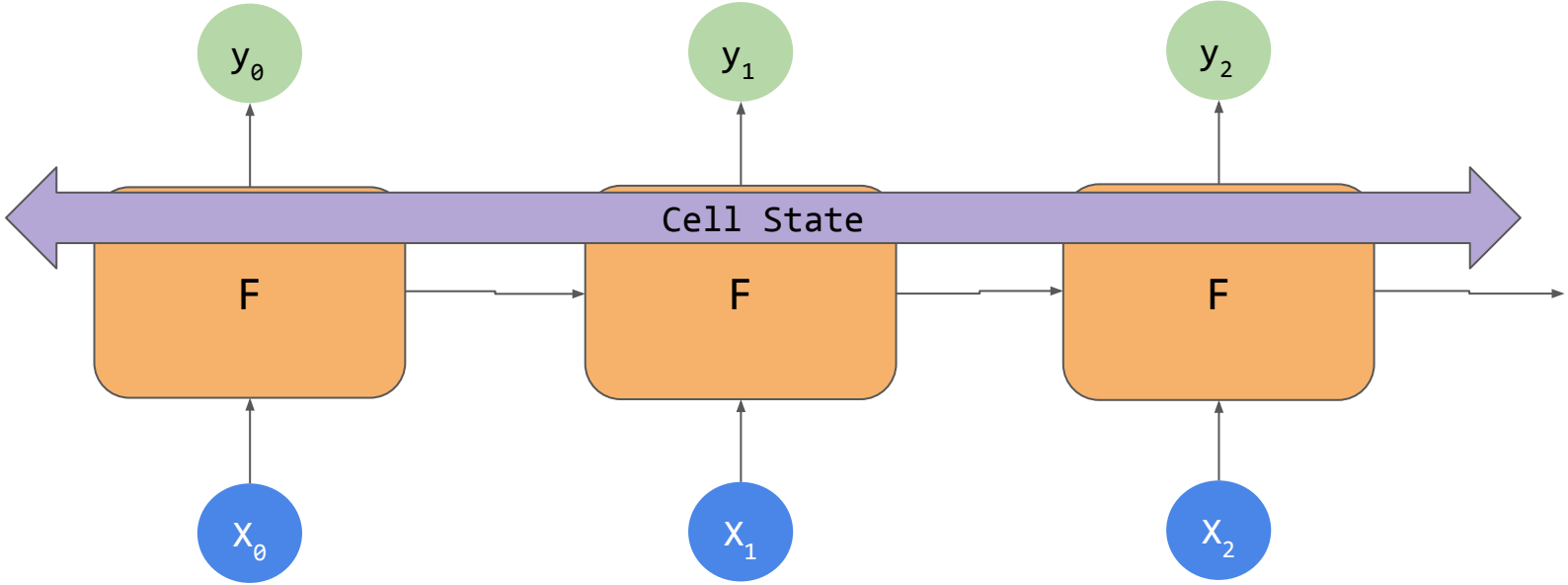

MAE: 6.4141674
120
100
80
60
40
Value
1000
1100
1200
1300
1400
Time


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

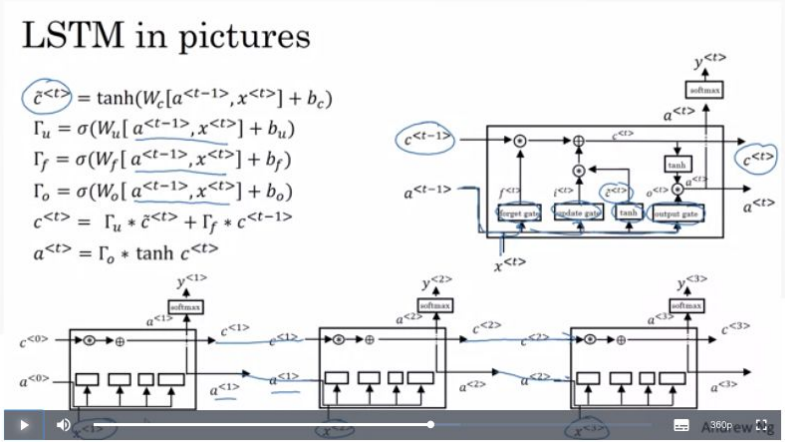

y0
y1
y2
F
F
F
x0
x1
x2


y_0
y_1
y_2
Cell State
F
F
F
x_0
x_1
x_2


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

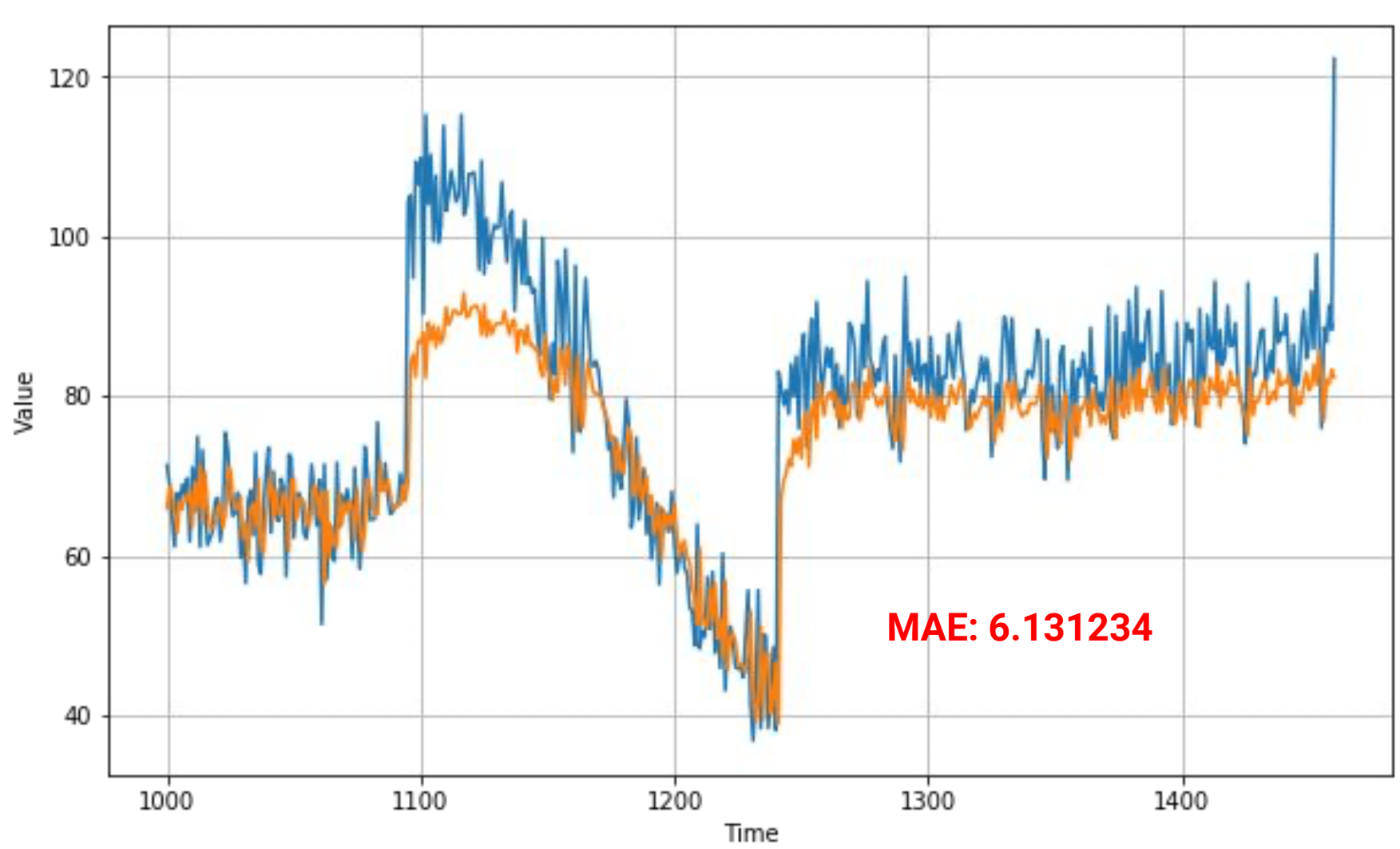
```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```

```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```

```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```

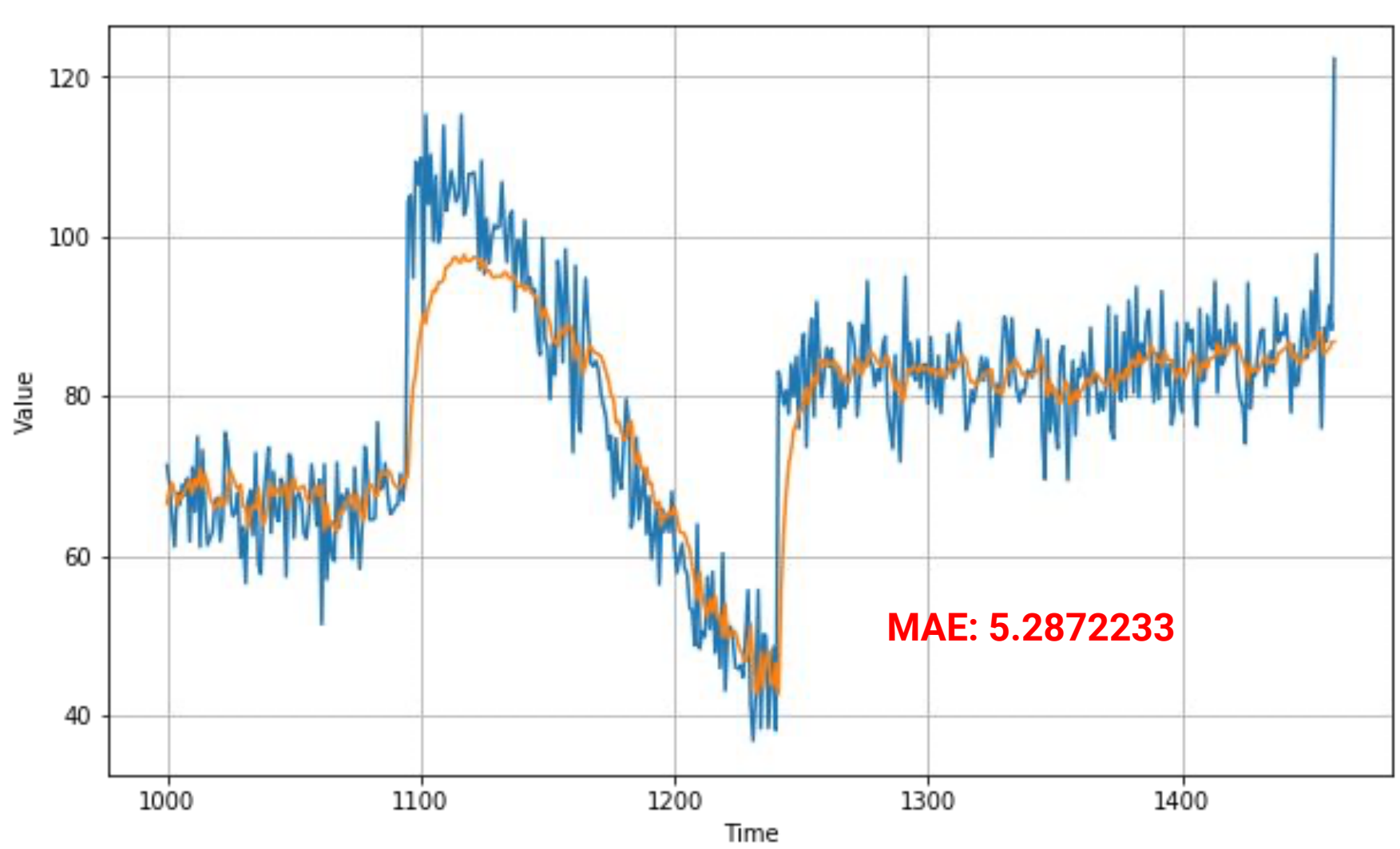
```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```

```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```


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

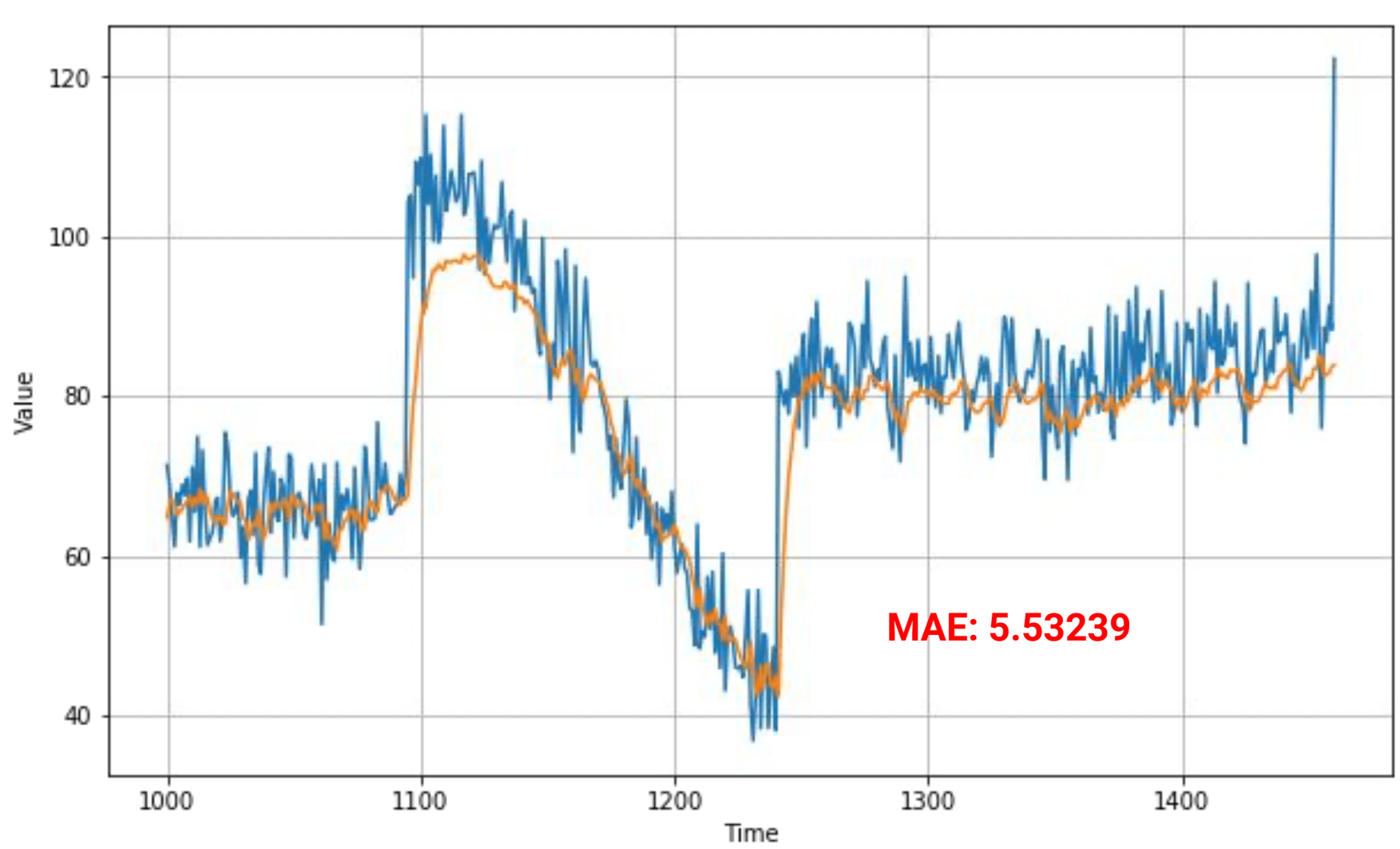
```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 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 * 100.0)
])


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

history = model.fit(dataset, epochs=100)
```

```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 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 * 100.0)
])


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

history = model.fit(dataset, epochs=100)
```


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

```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```

```
tf.keras.backend.clear_session()

train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size, 1)),
    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))

history = model.fit(dataset, epochs=100)
```


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