

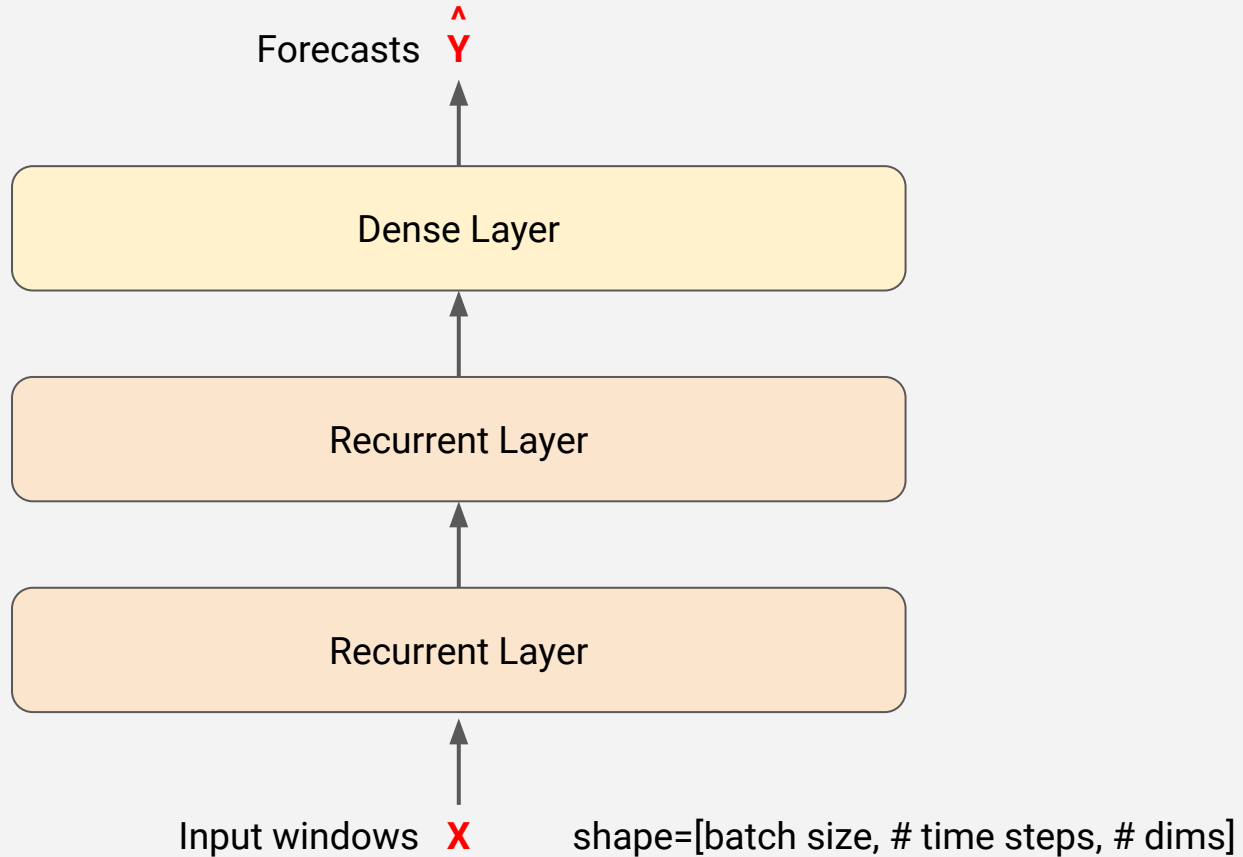
# Copyright Notice

These slides are distributed under the Creative Commons License.

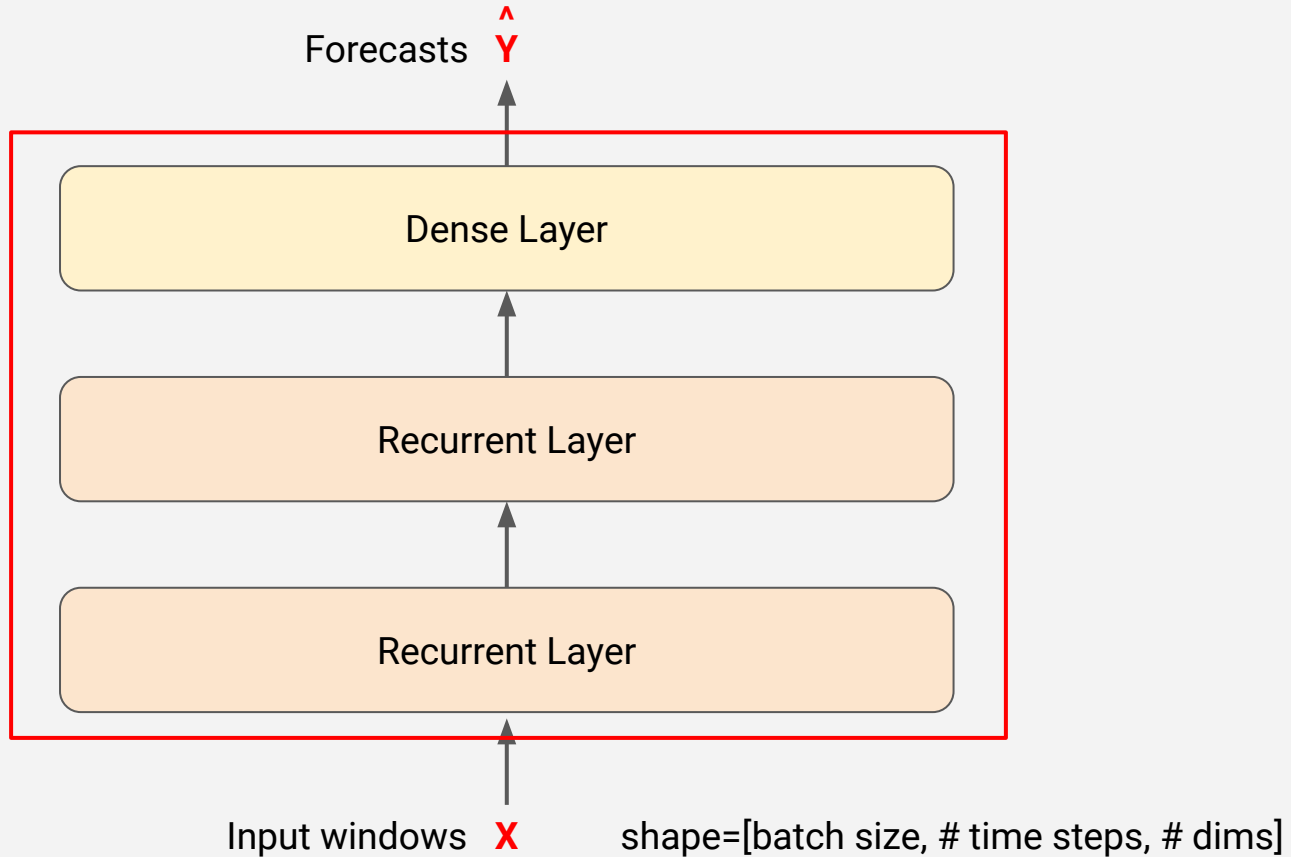
[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

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

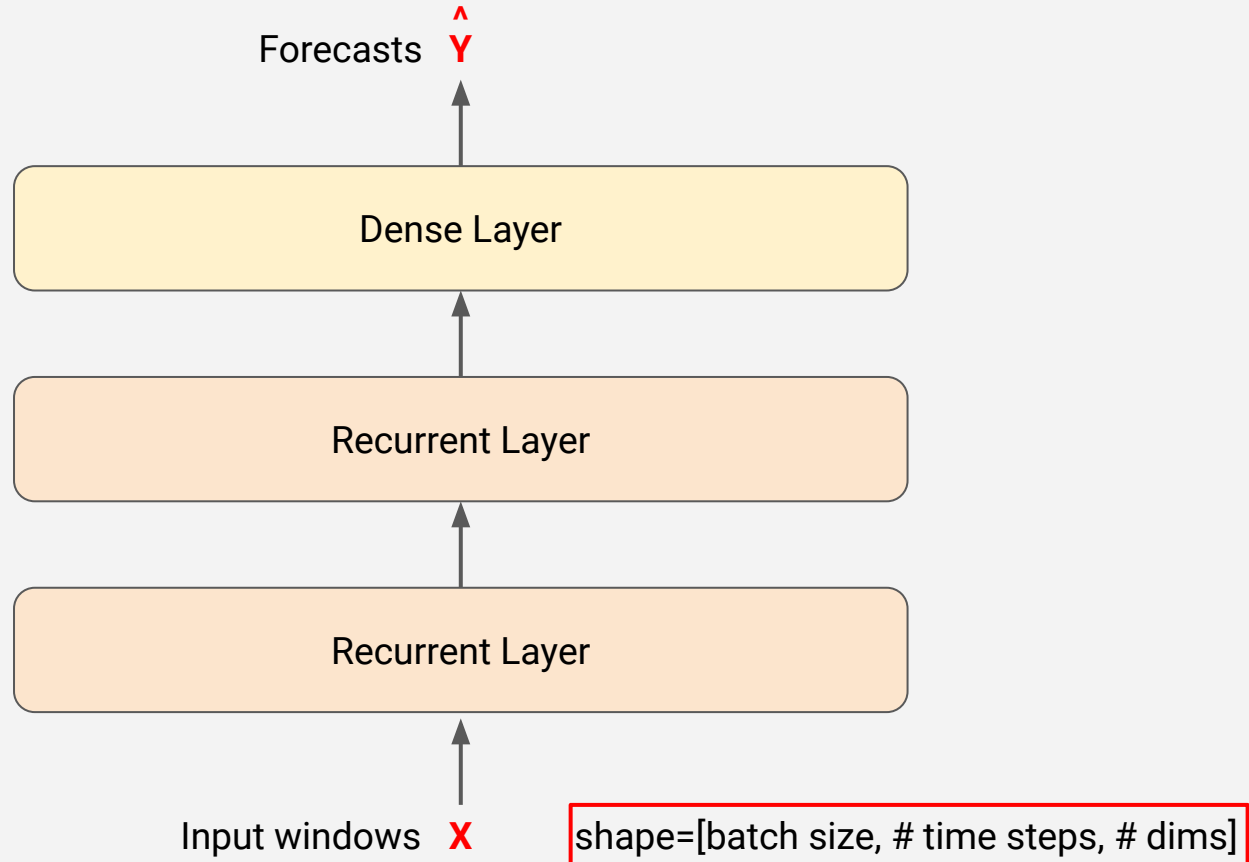
# Recurrent Neural Network



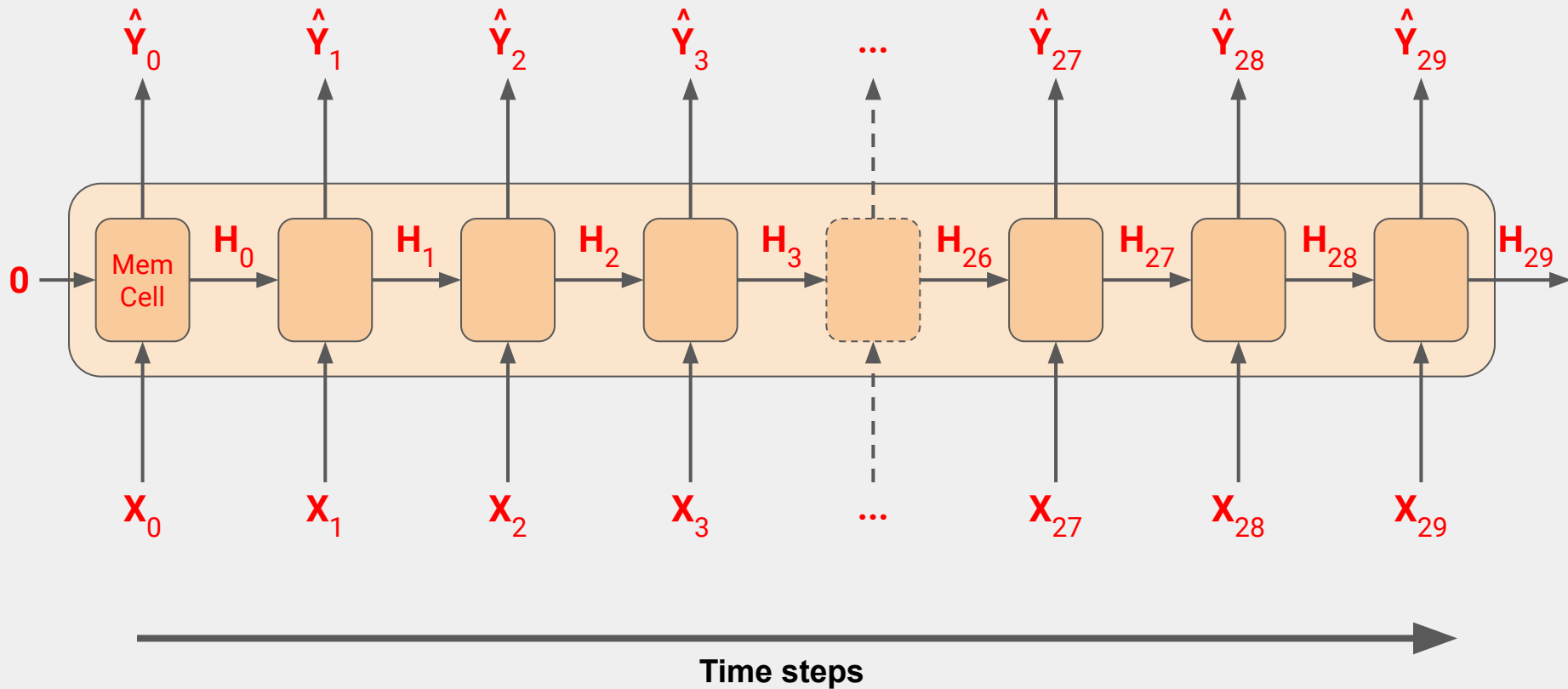
# Recurrent Neural Network



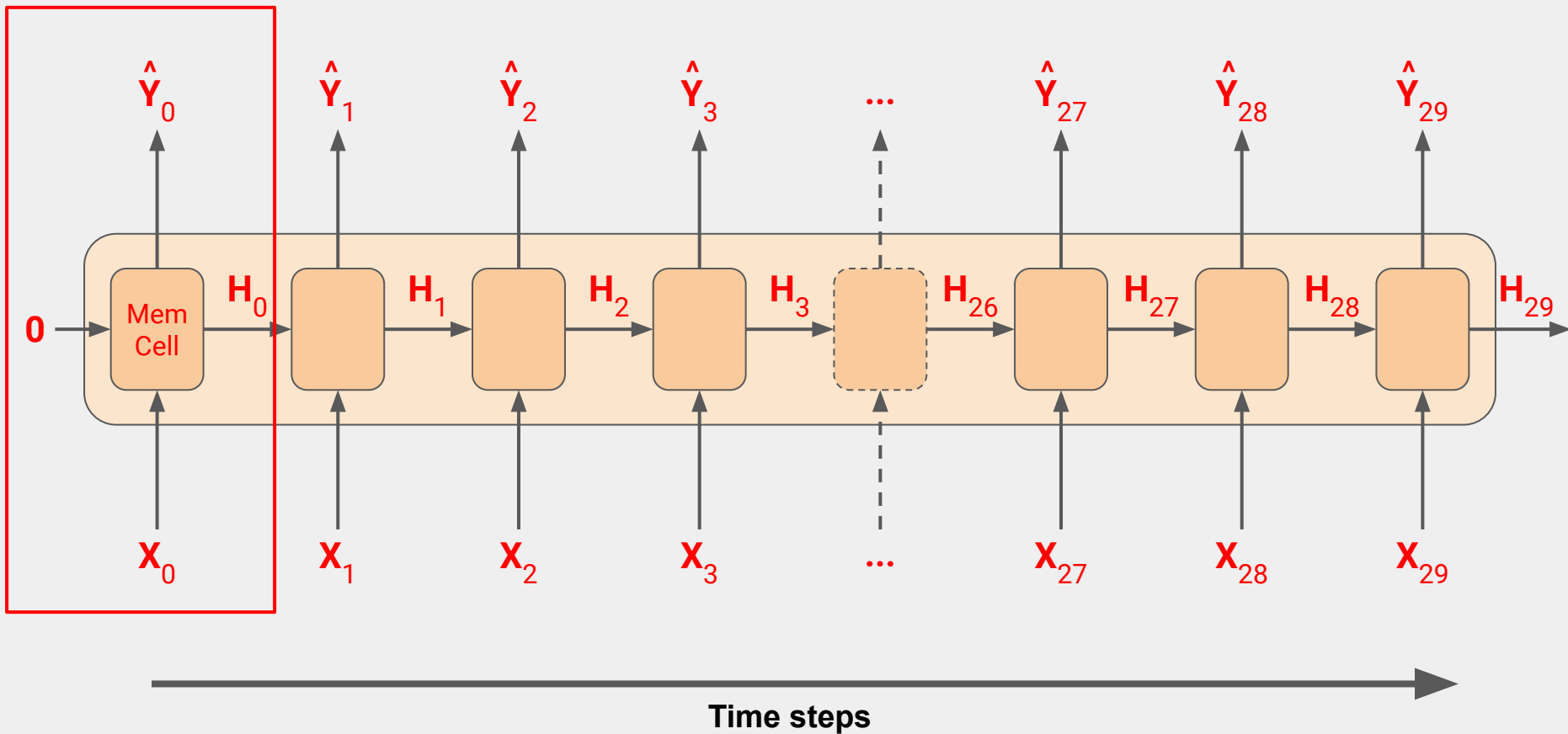
# Recurrent Neural Network



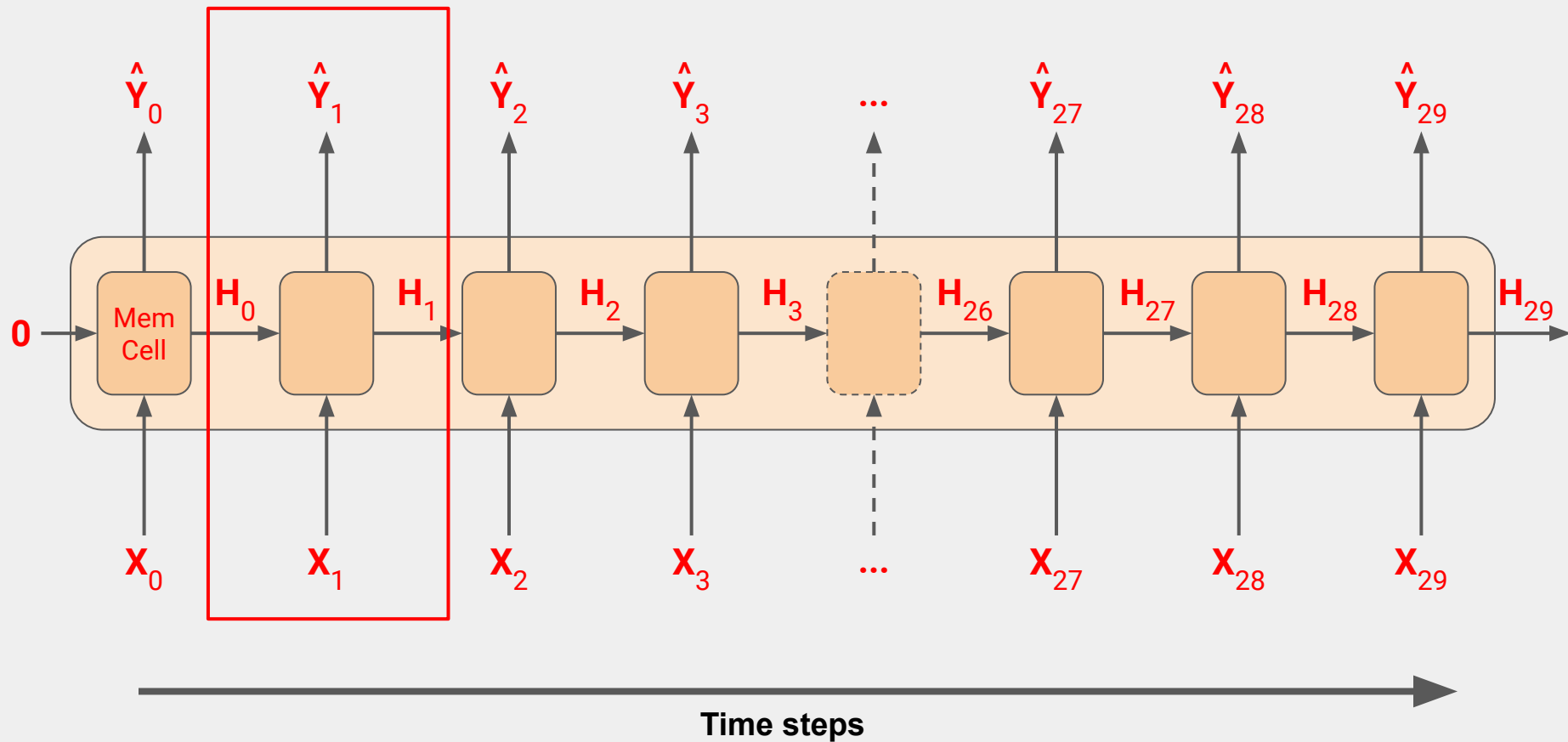
# Recurrent Layer



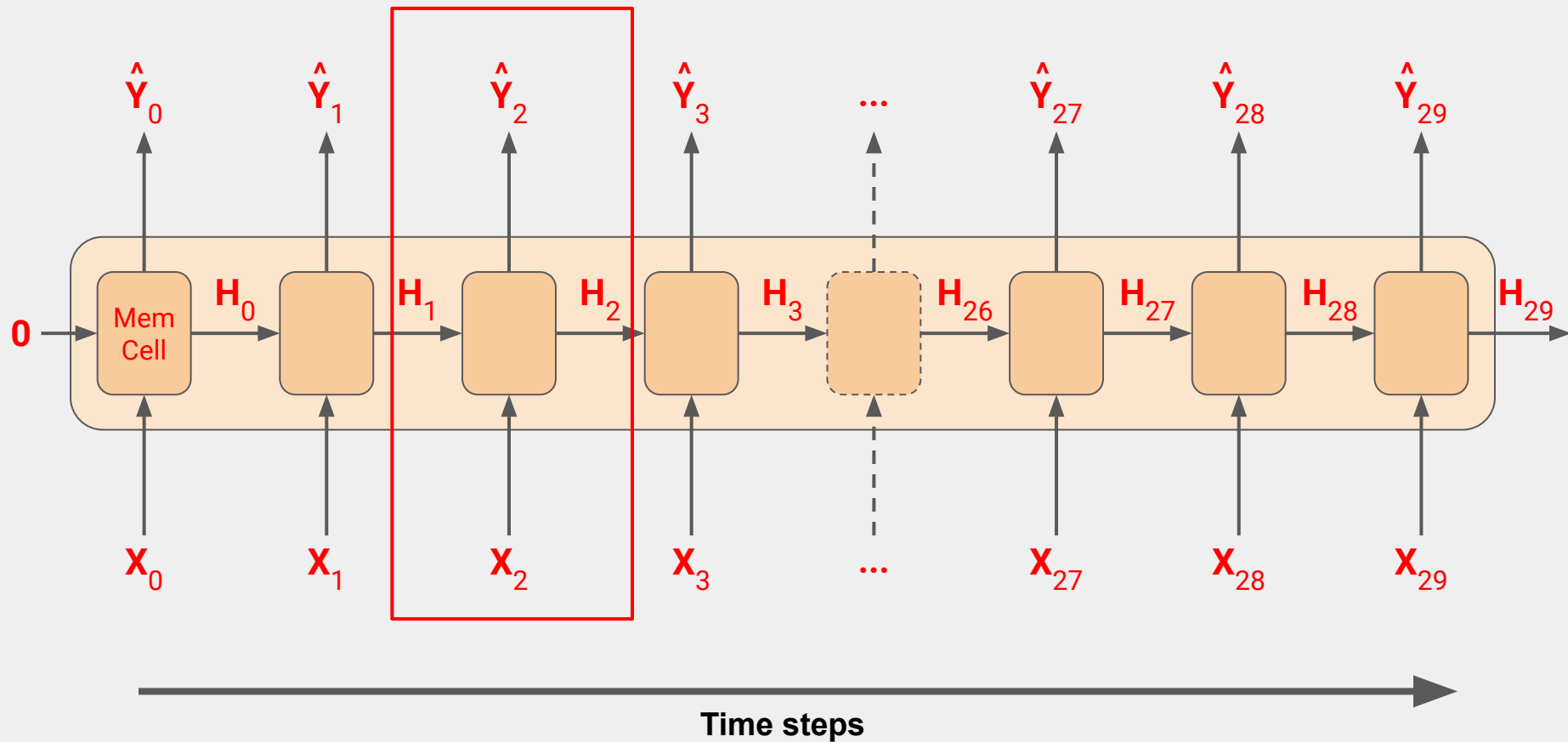
# Recurrent Layer



# Recurrent Layer

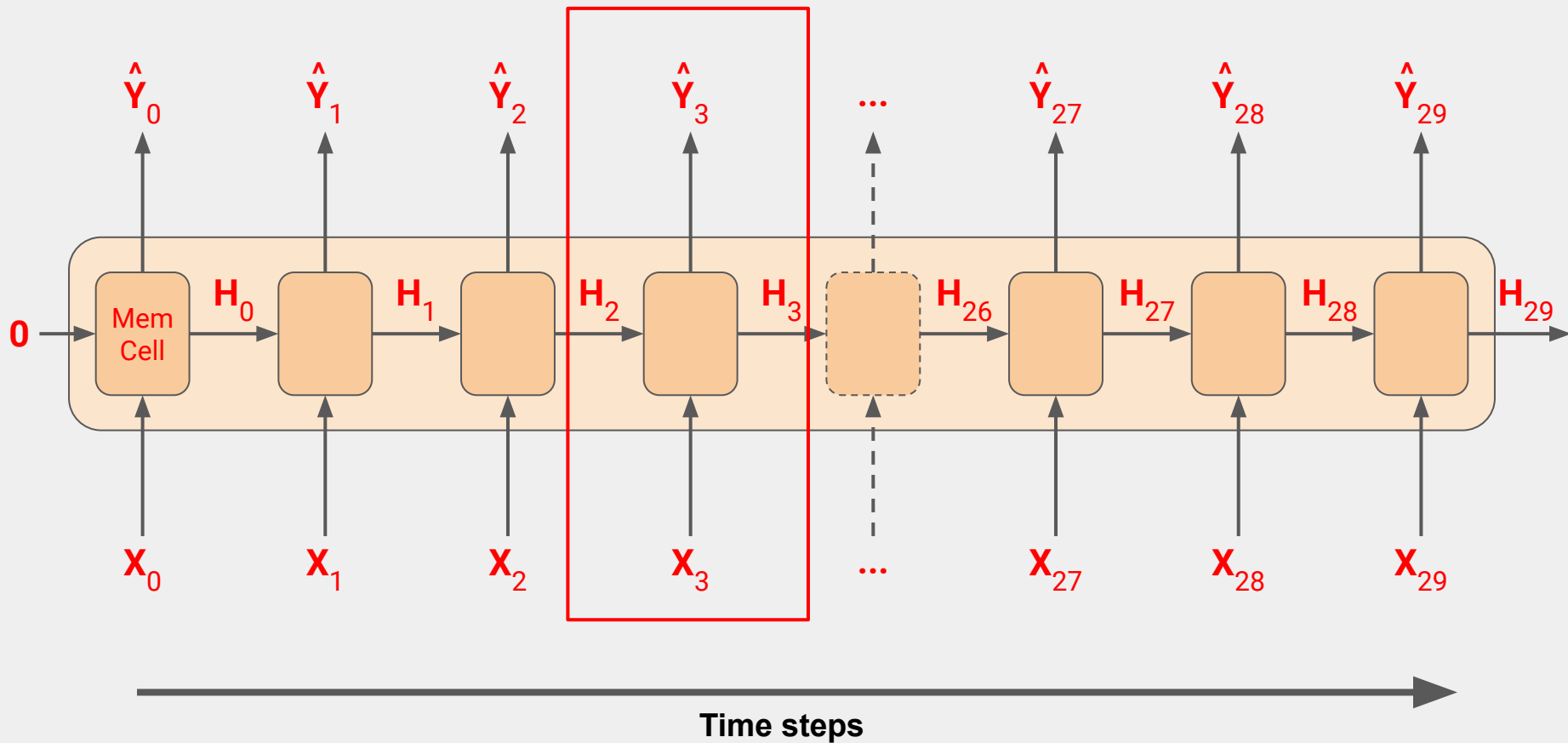


# Recurrent Layer

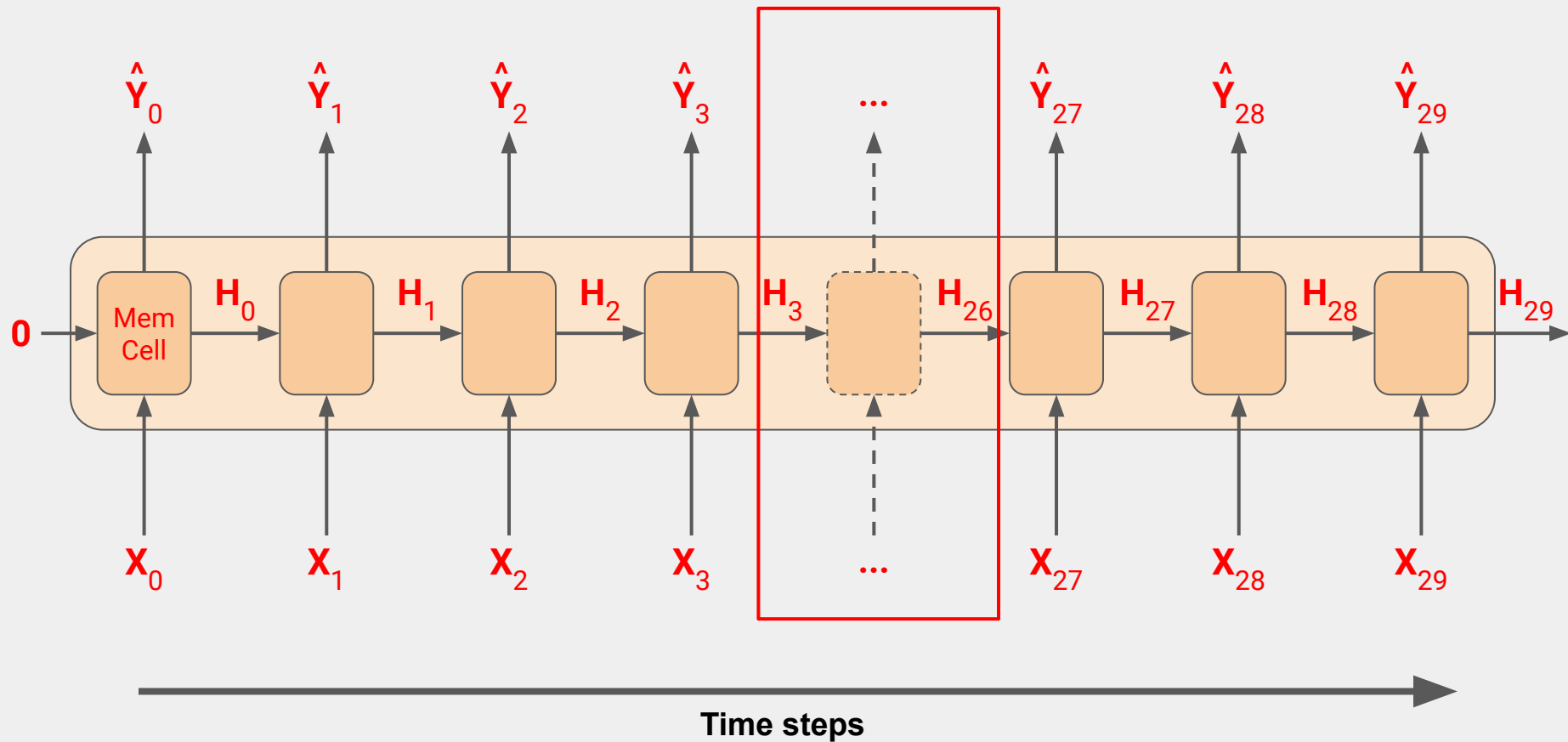




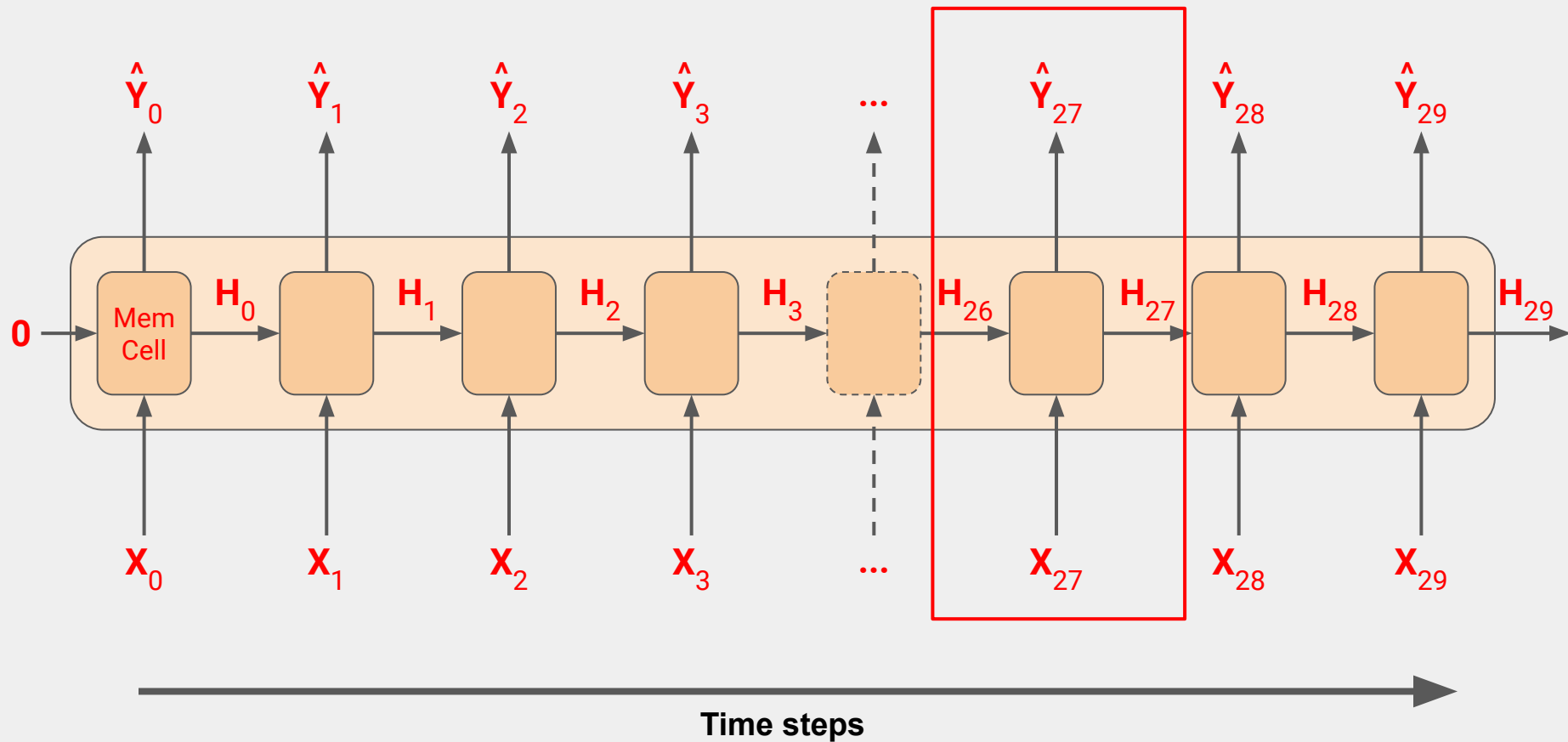
# Recurrent Layer



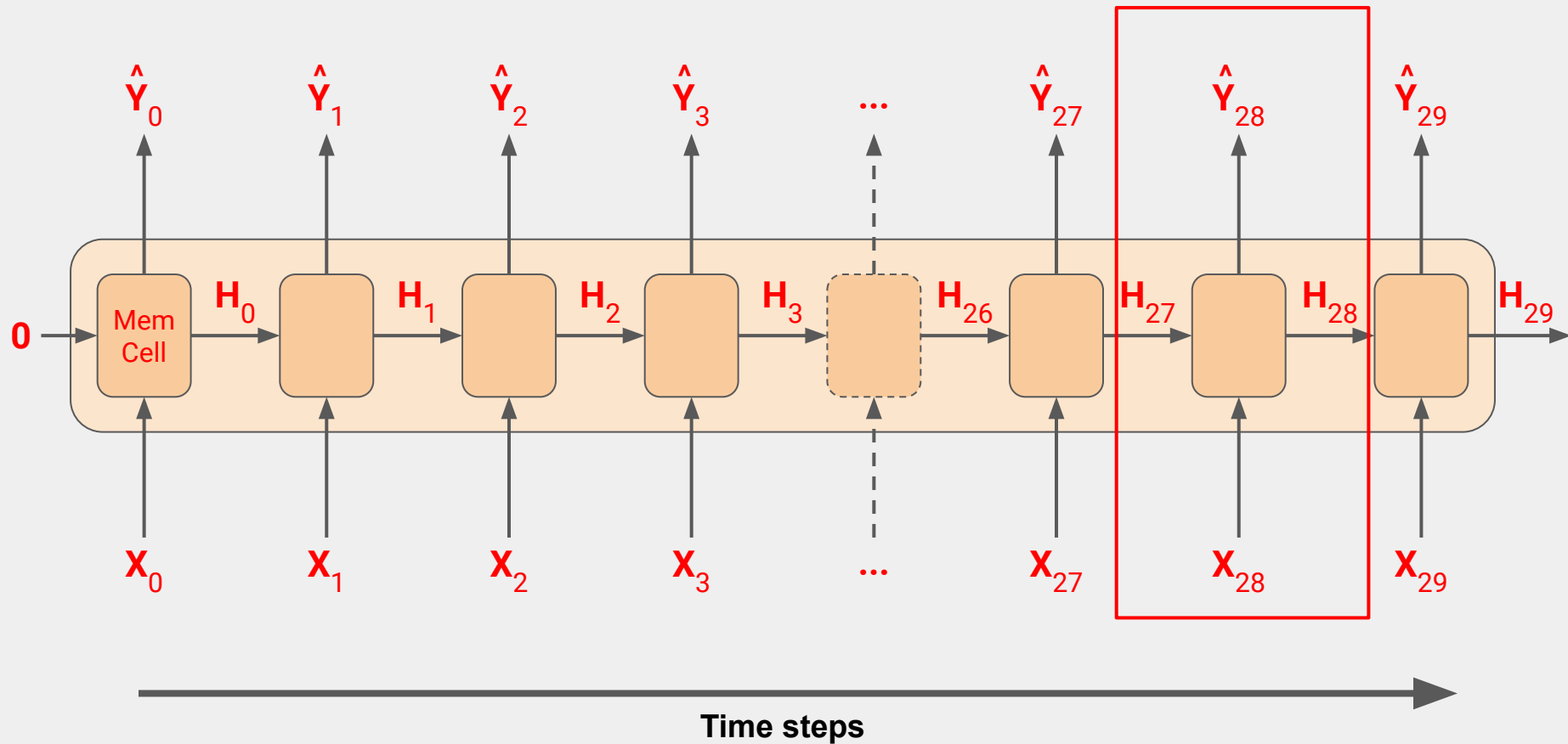
# Recurrent Layer



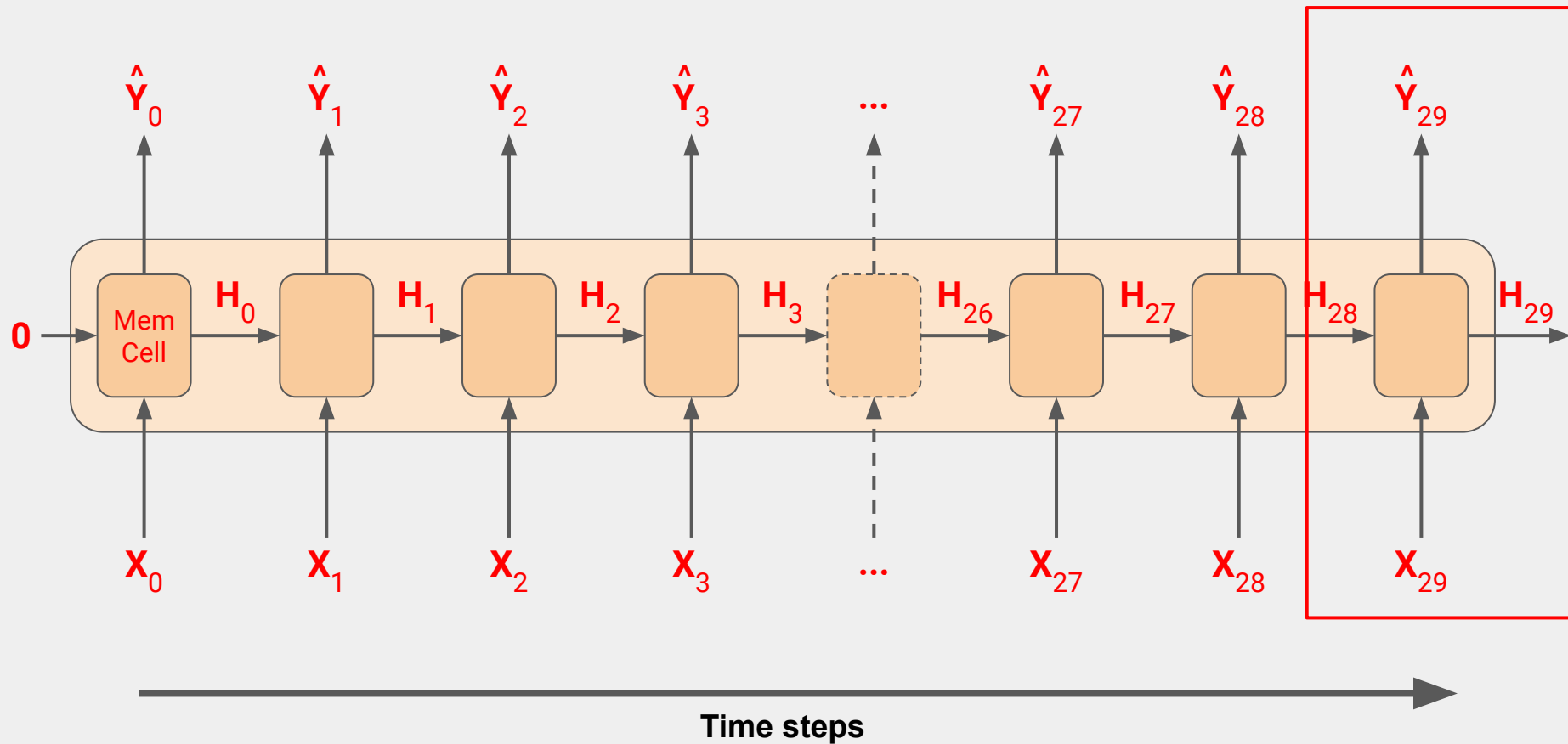
# Recurrent Layer



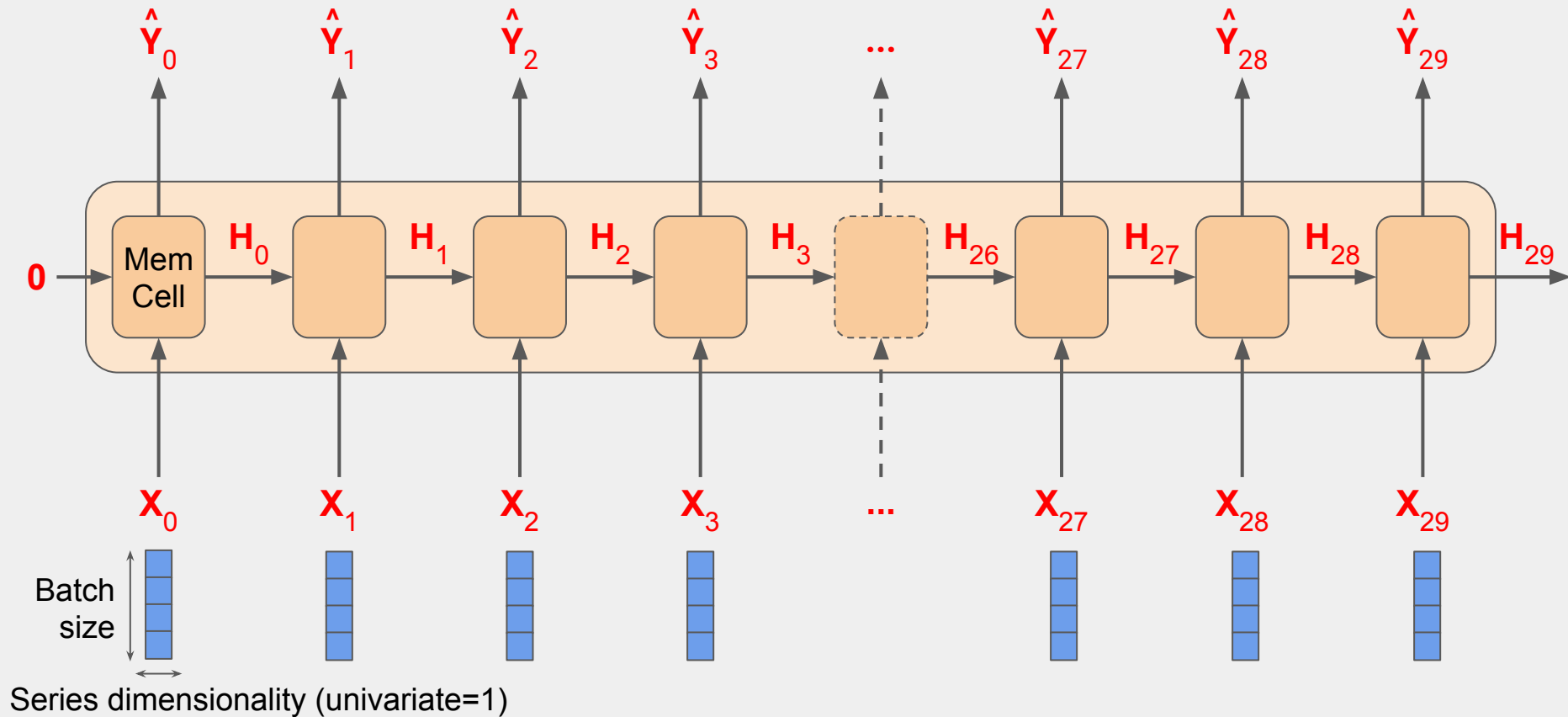
# Recurrent Layer



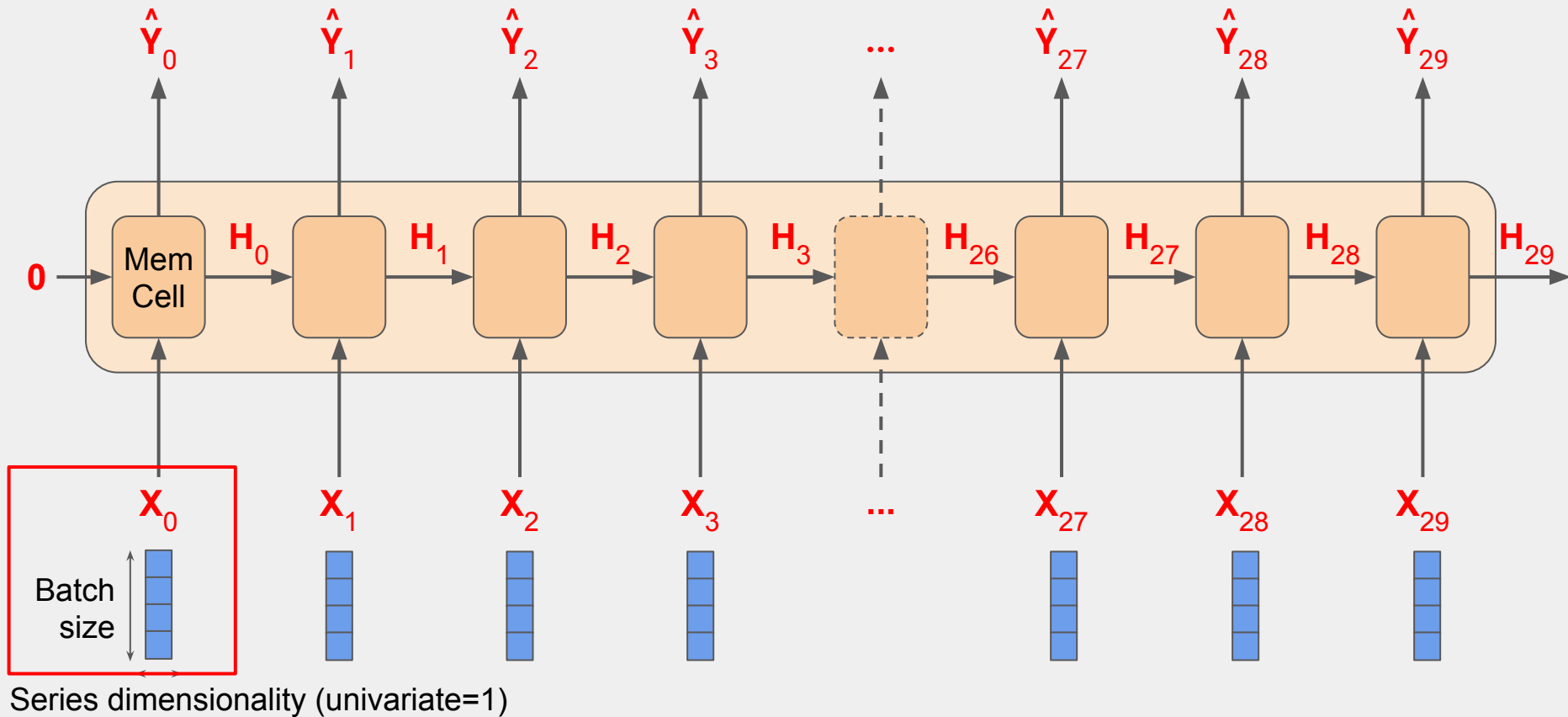
# Recurrent Layer



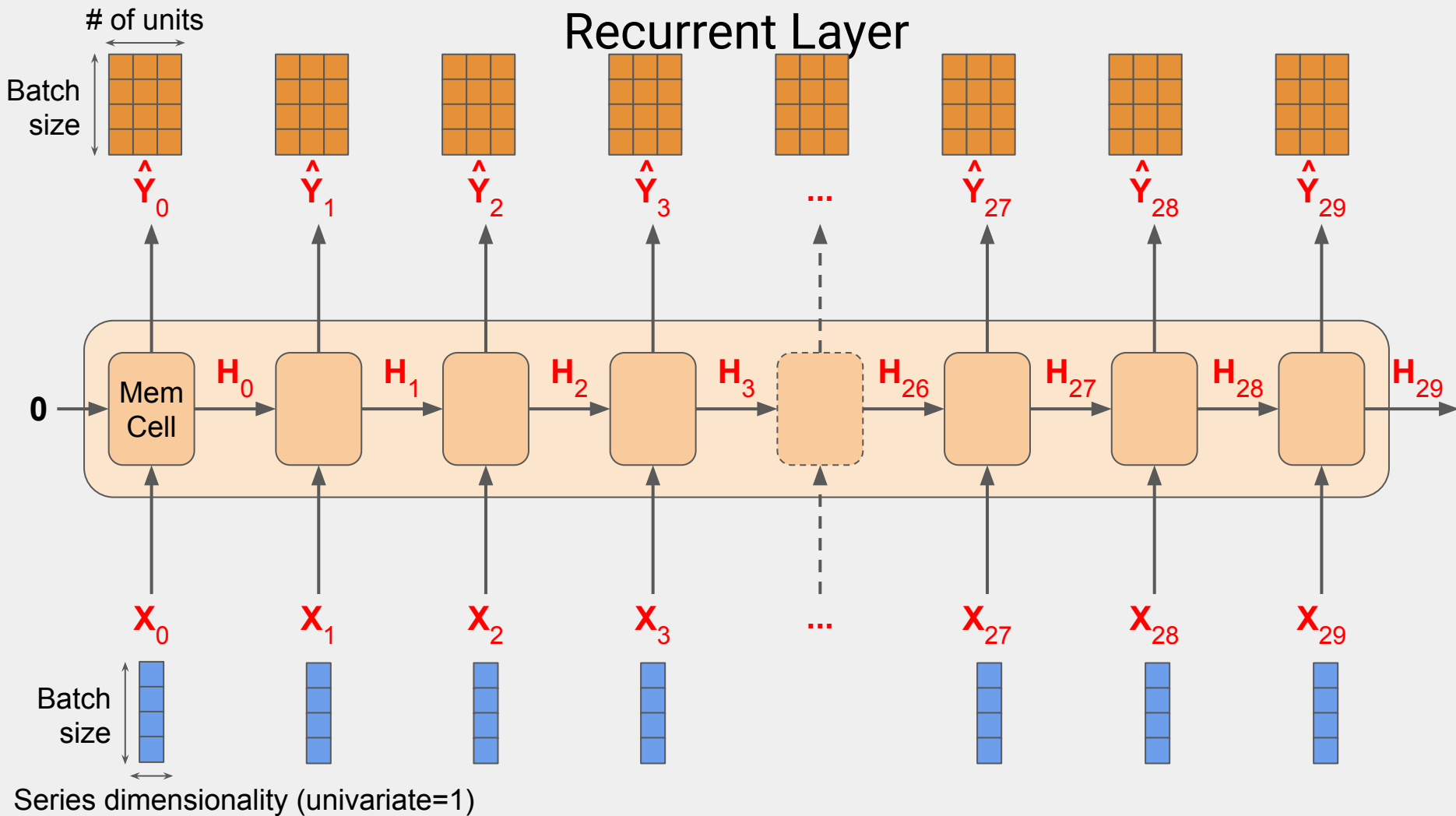
# Recurrent Layer



# Recurrent Layer

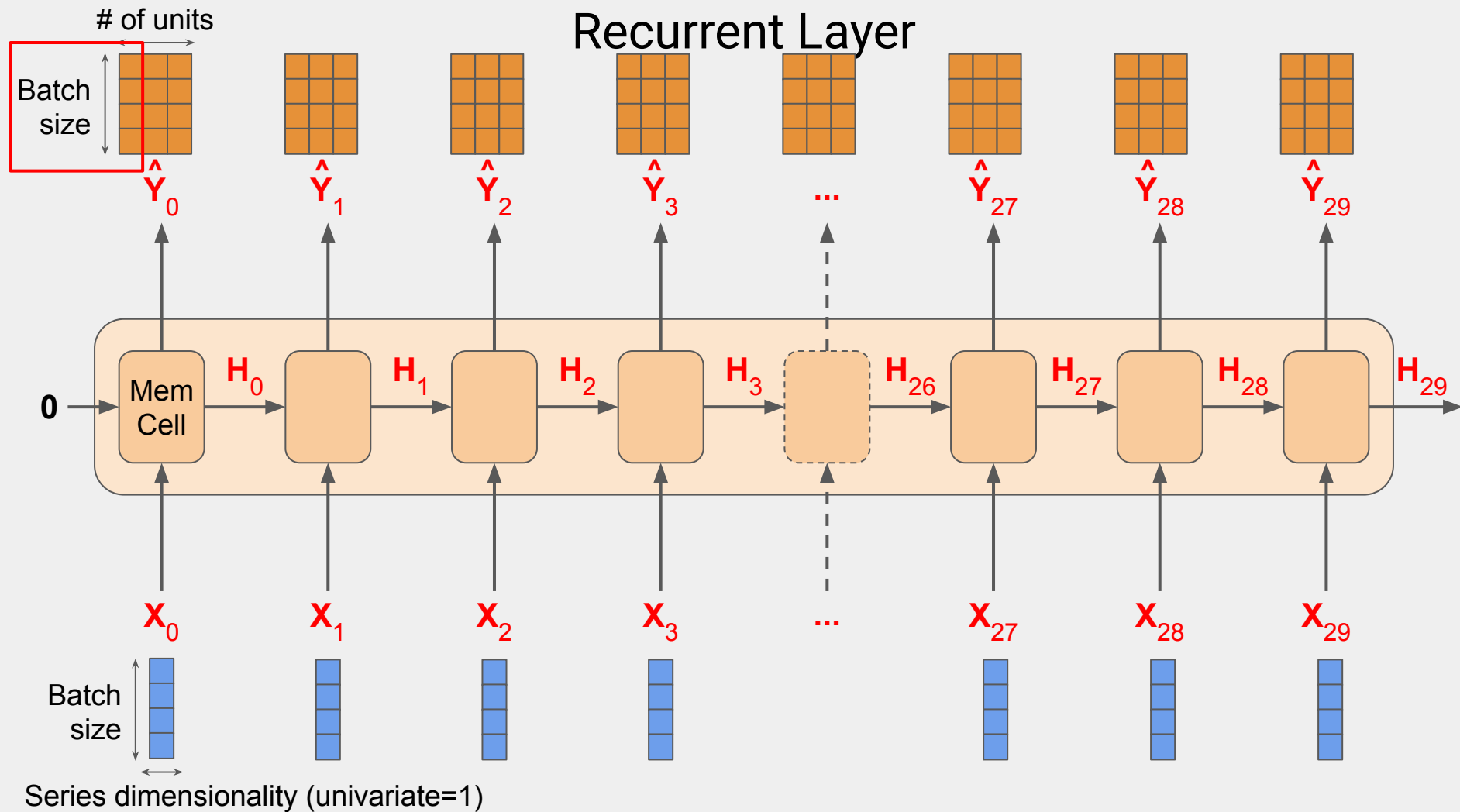


# Recurrent Layer

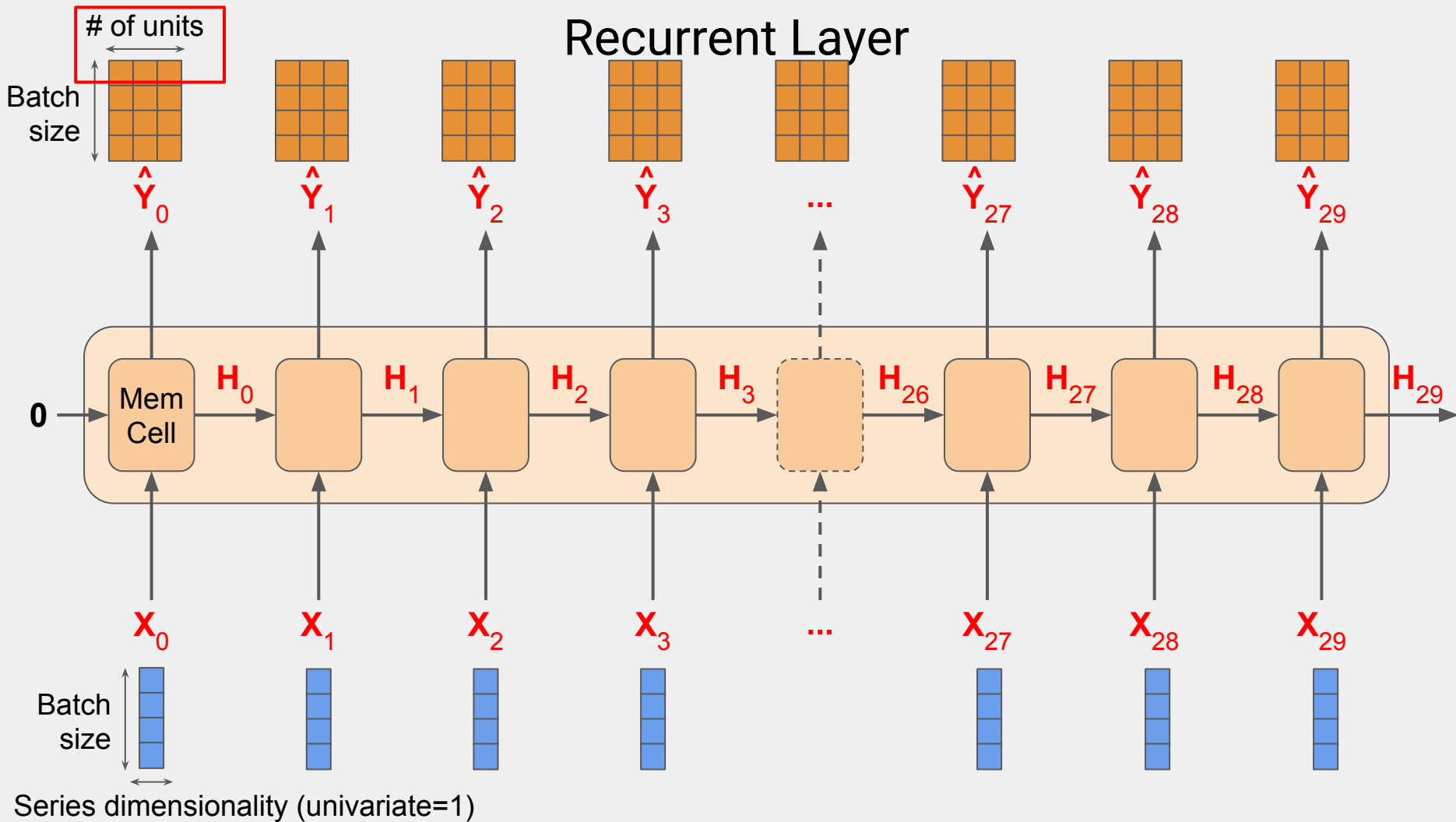




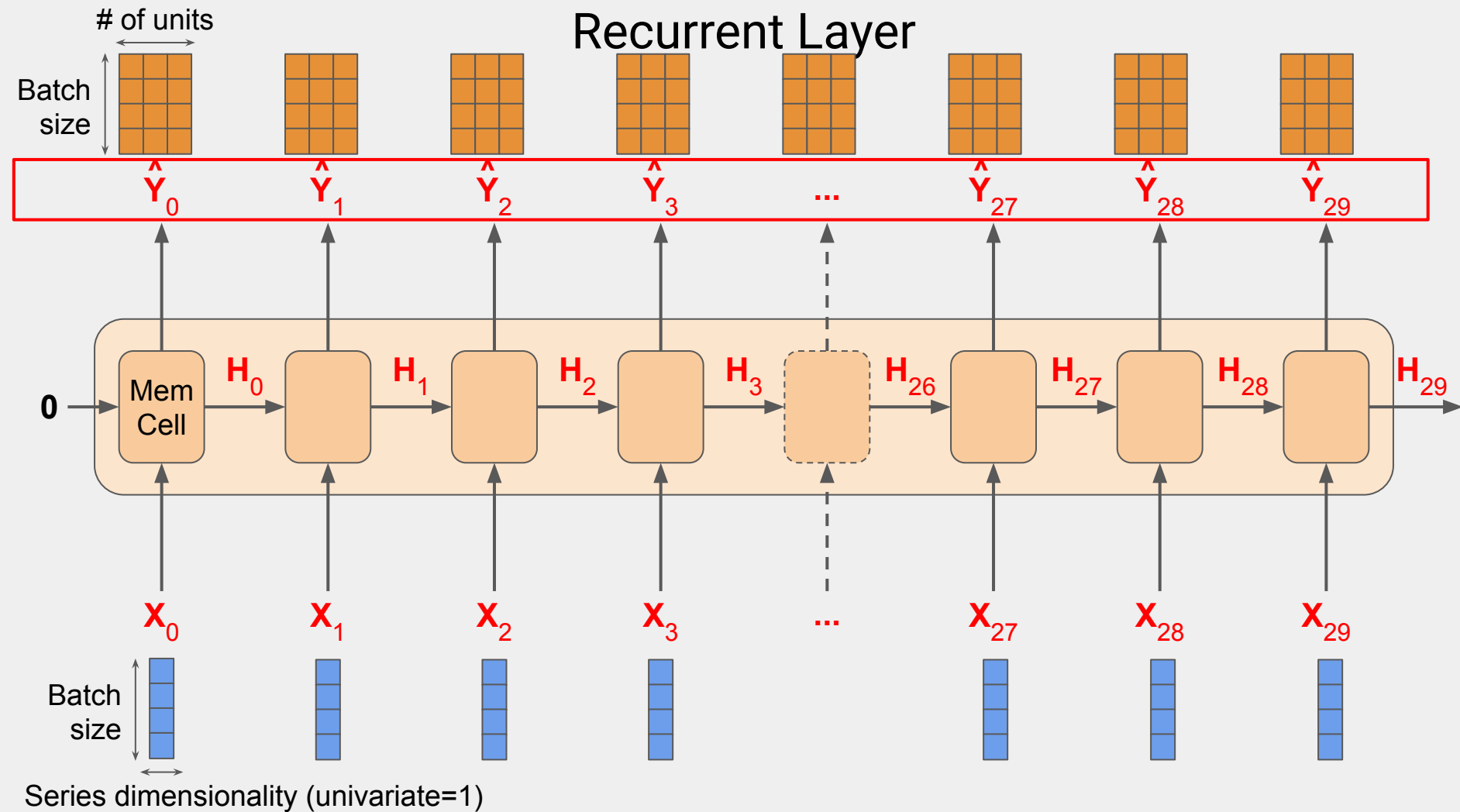
# Recurrent Layer



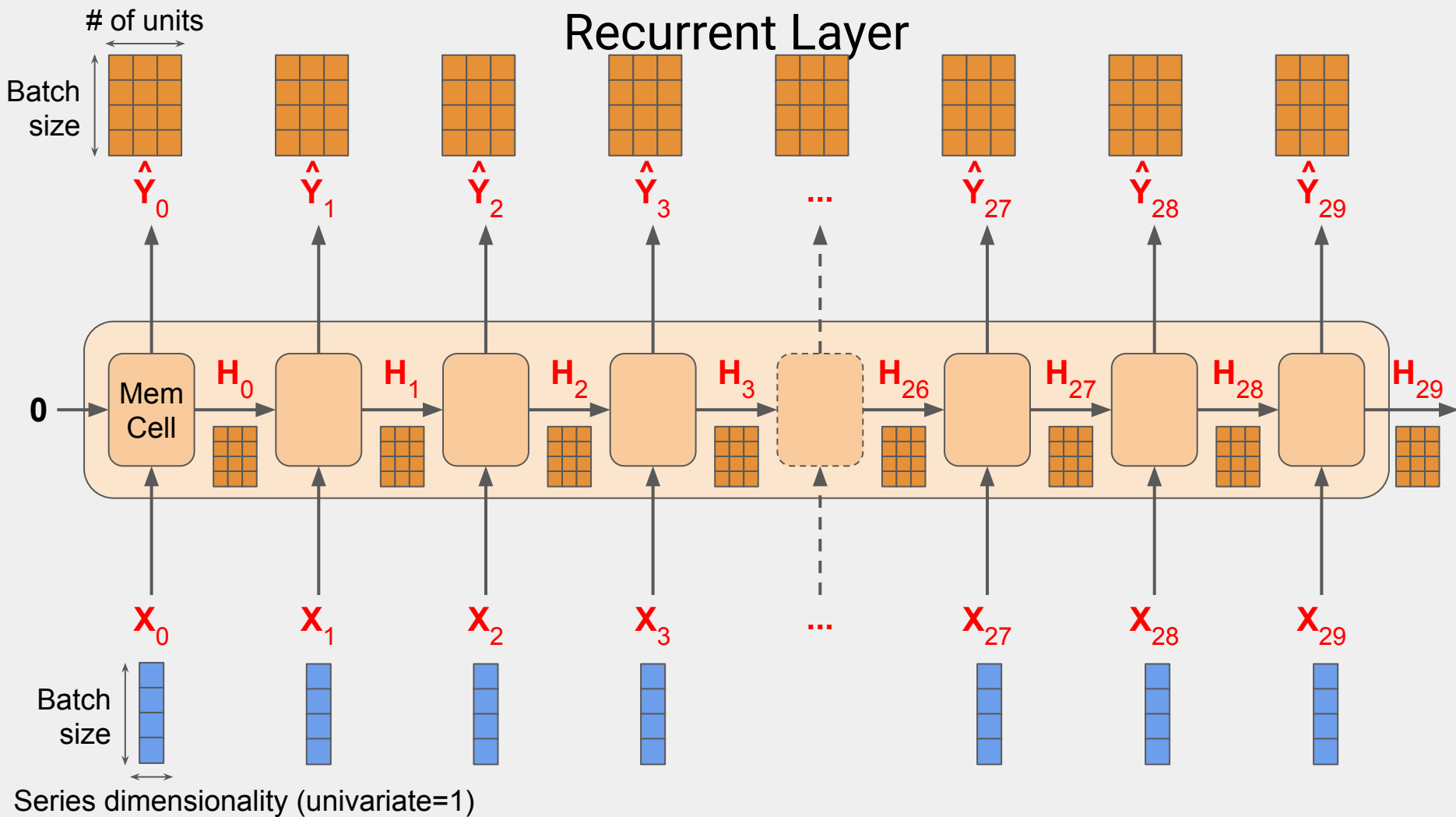
# Recurrent Layer



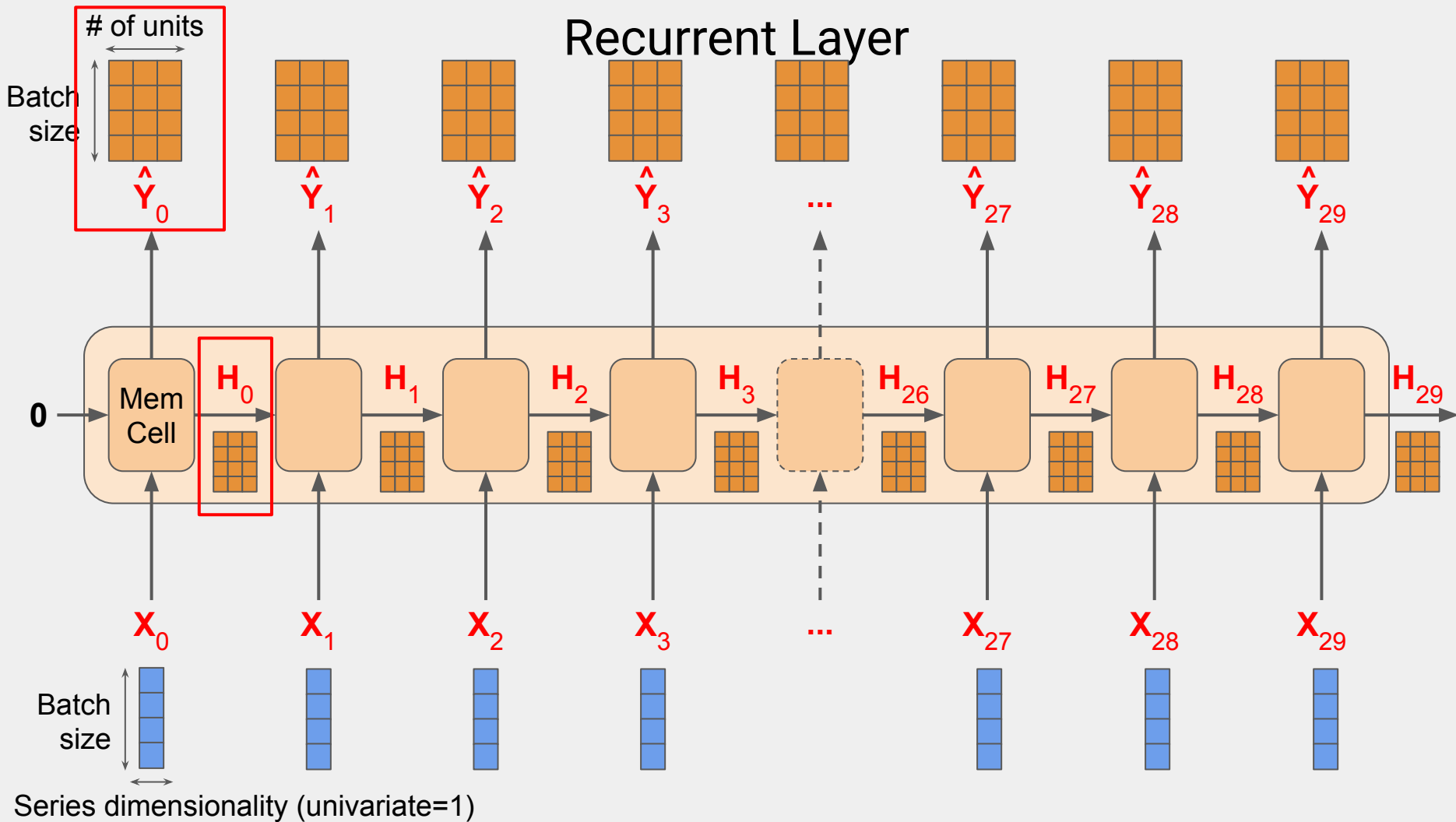
# Recurrent Layer



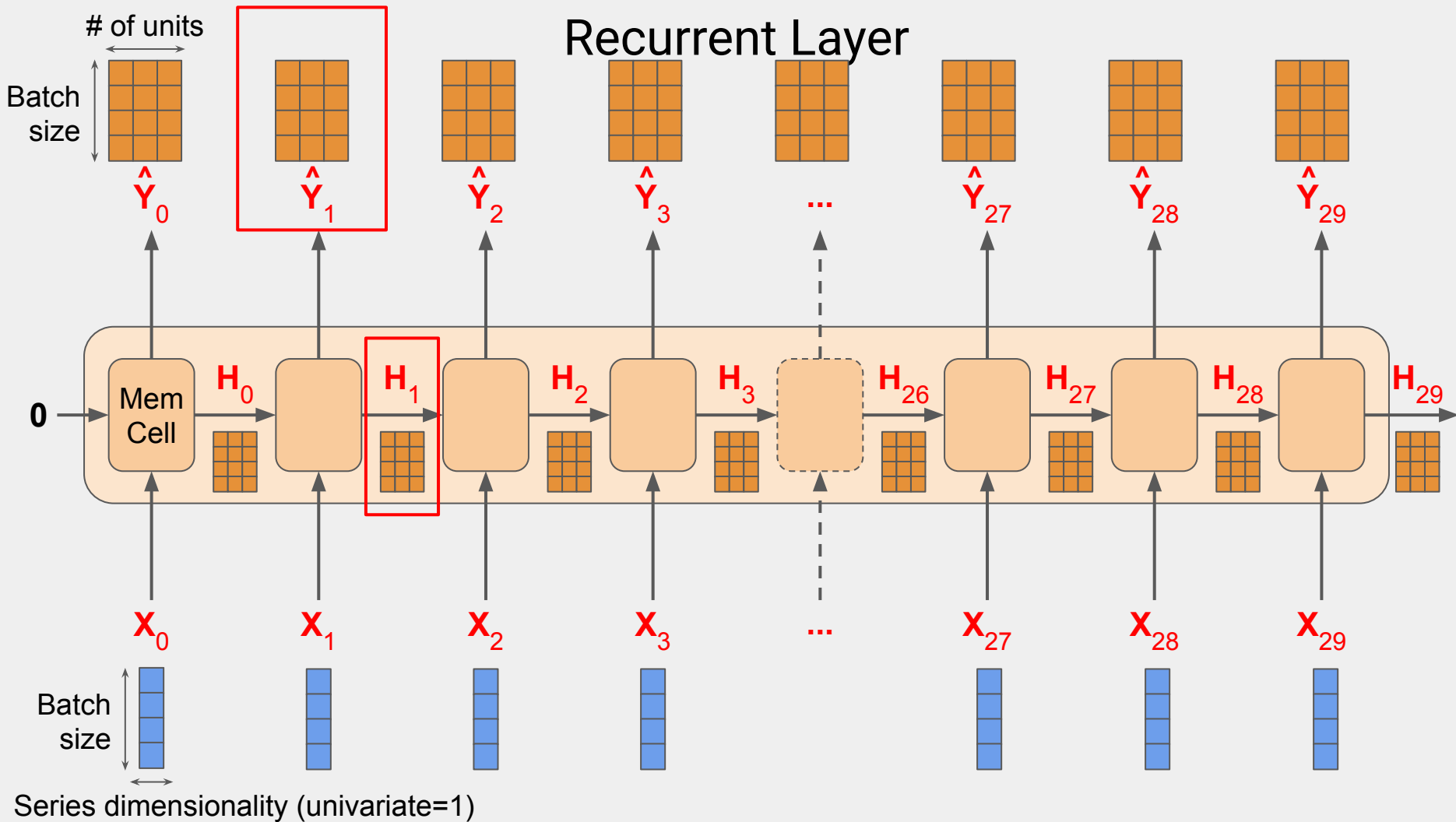
# Recurrent Layer



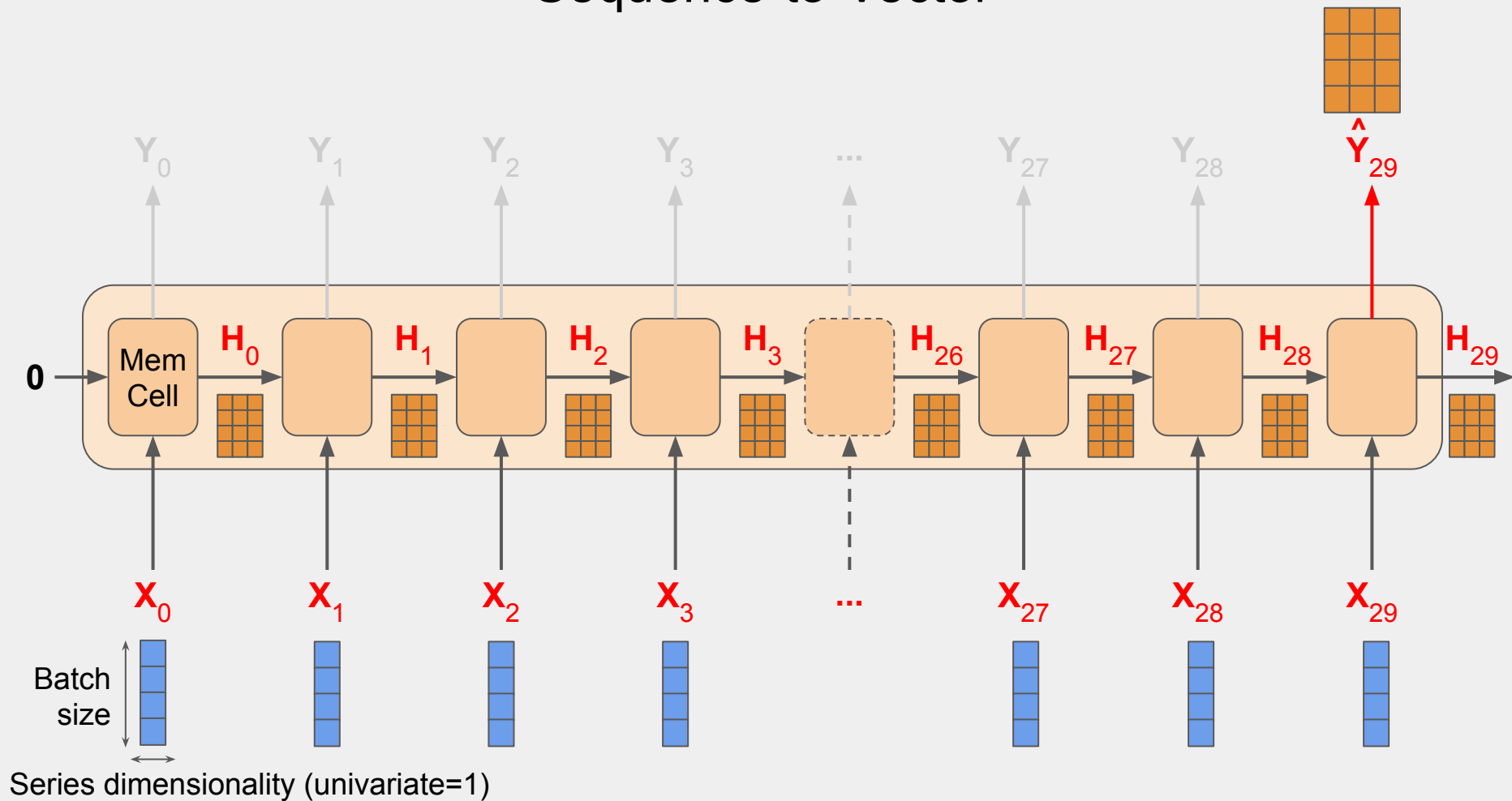
# Recurrent Layer



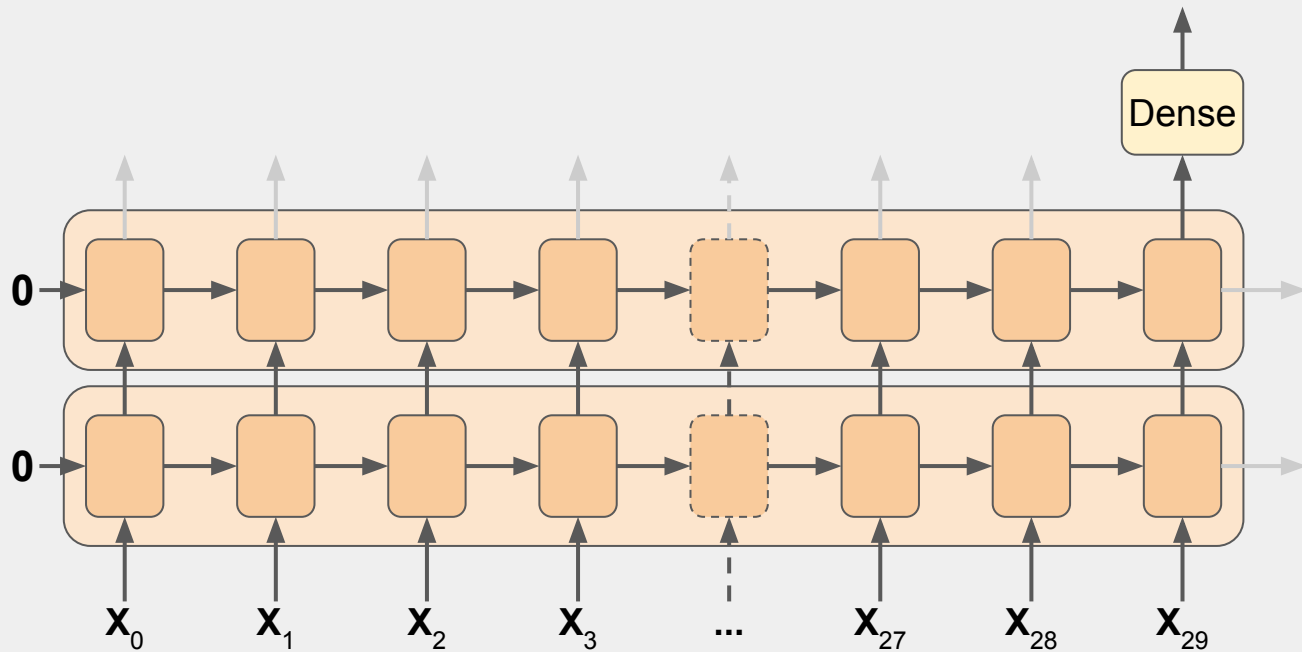
# Recurrent Layer



# Sequence-to-Vector

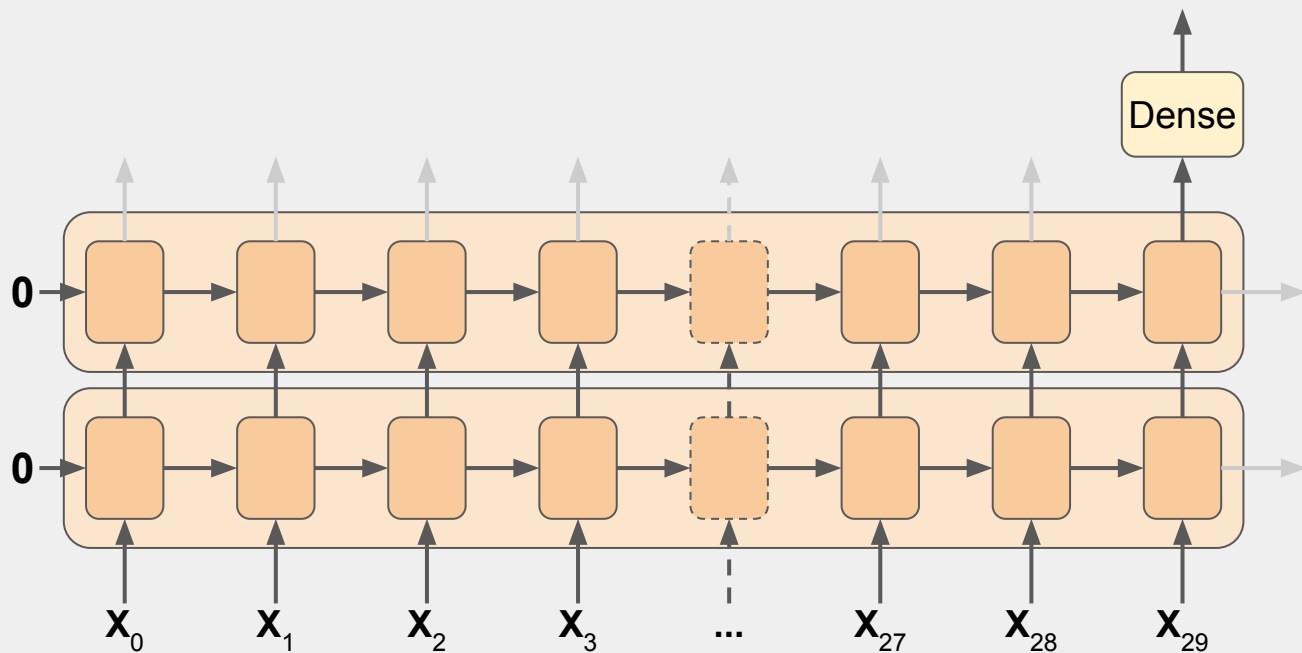


```
model = keras.models.Sequential([  
    keras.layers.SimpleRNN(20, return_sequences=True,  
                             input_shape=[None, 1]),  
    keras.layers.SimpleRNN(20),  
    keras.layers.Dense(1)  
])
```

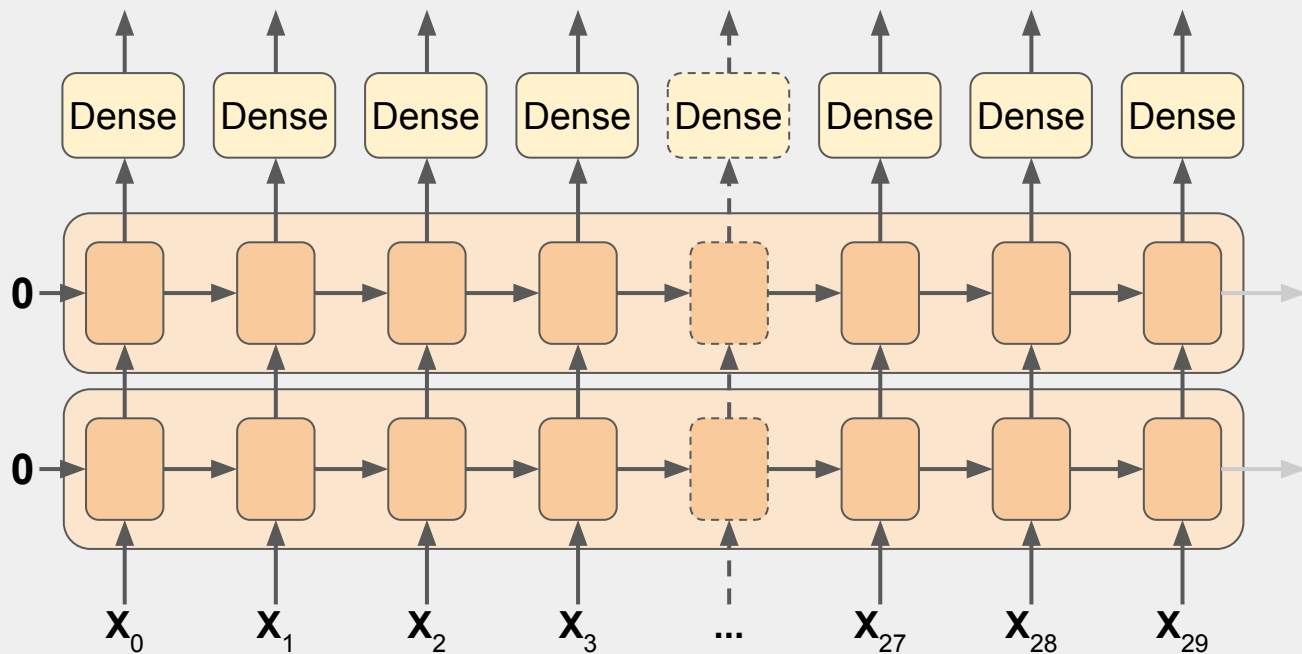




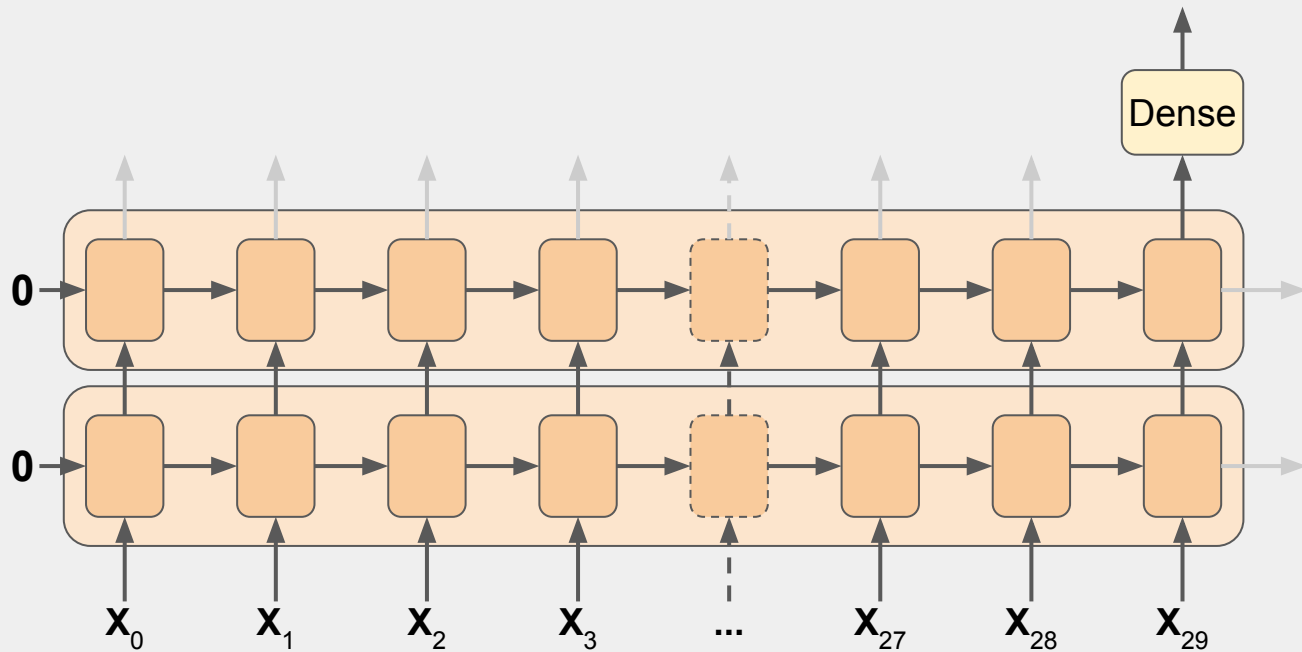
```
model = keras.models.Sequential([  
    keras.layers.SimpleRNN(20, return_sequences=True,  
                             input_shape=[None, 1]),  
    keras.layers.SimpleRNN(20),  
    keras.layers.Dense(1)  
])
```



```
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)  
])
```



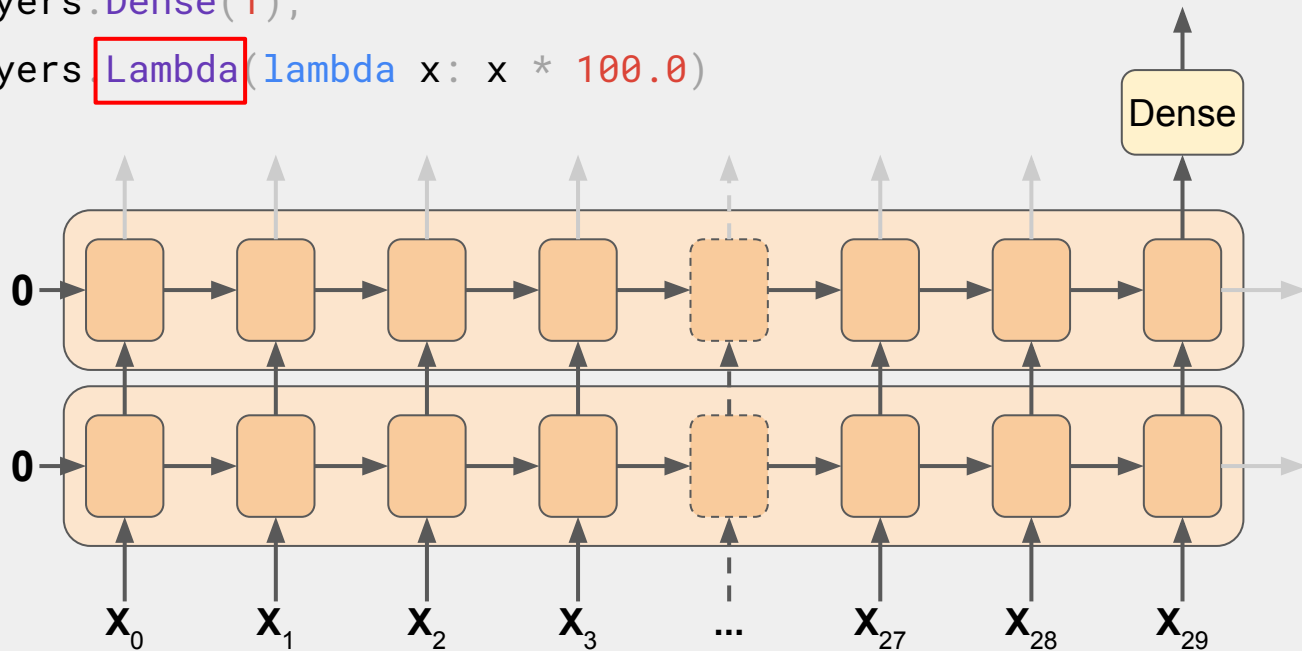
```
model = keras.models.Sequential([  
    keras.layers.SimpleRNN(20, return_sequences=True,  
                             input_shape=[None, 1]),  
    keras.layers.SimpleRNN(20),  
    keras.layers.Dense(1)  
])
```



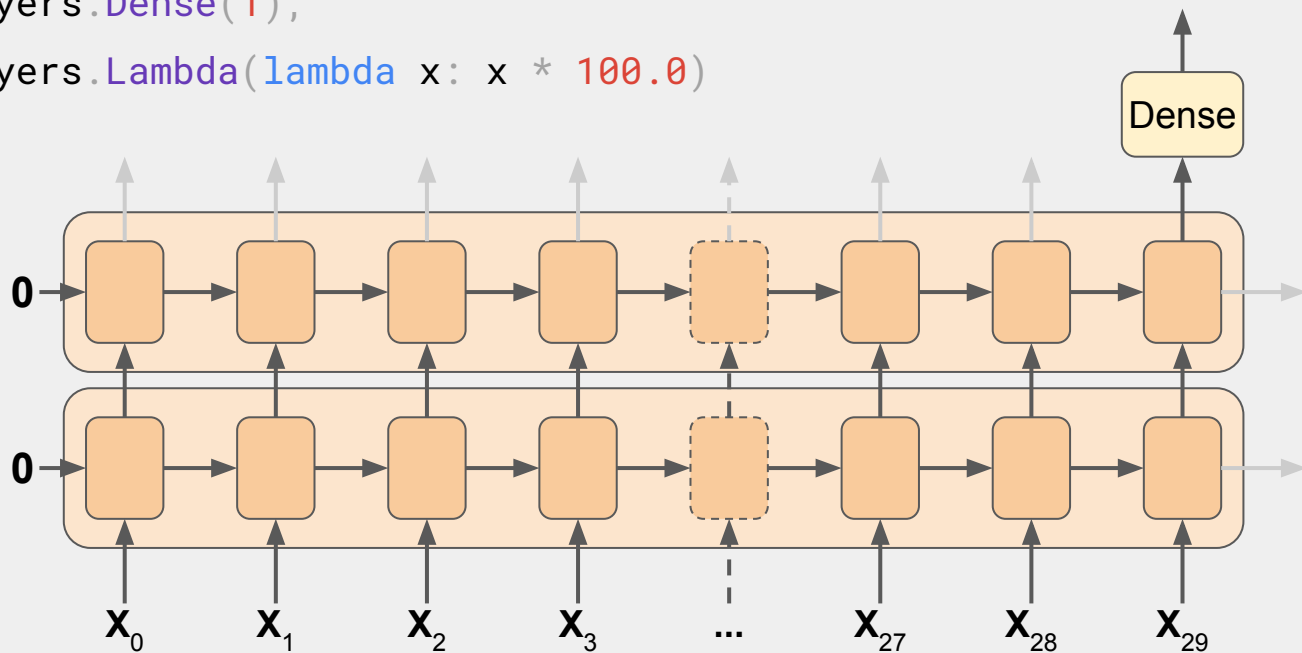
```

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)
])

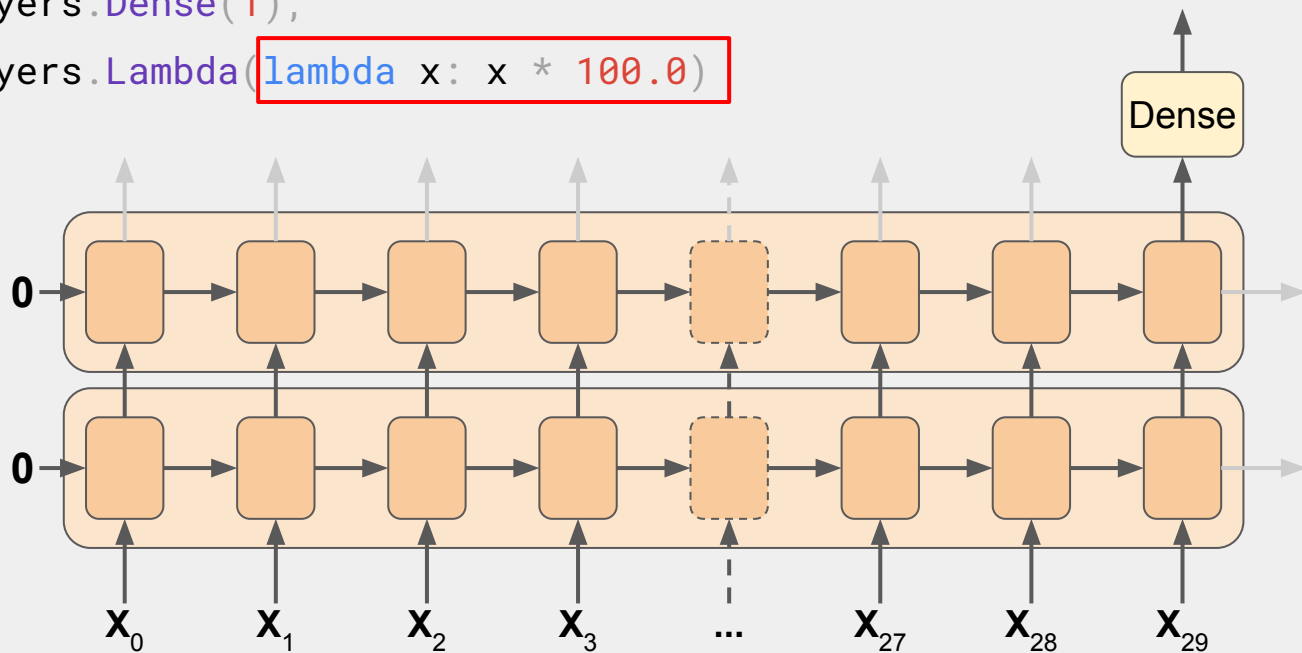
```



```
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)
])

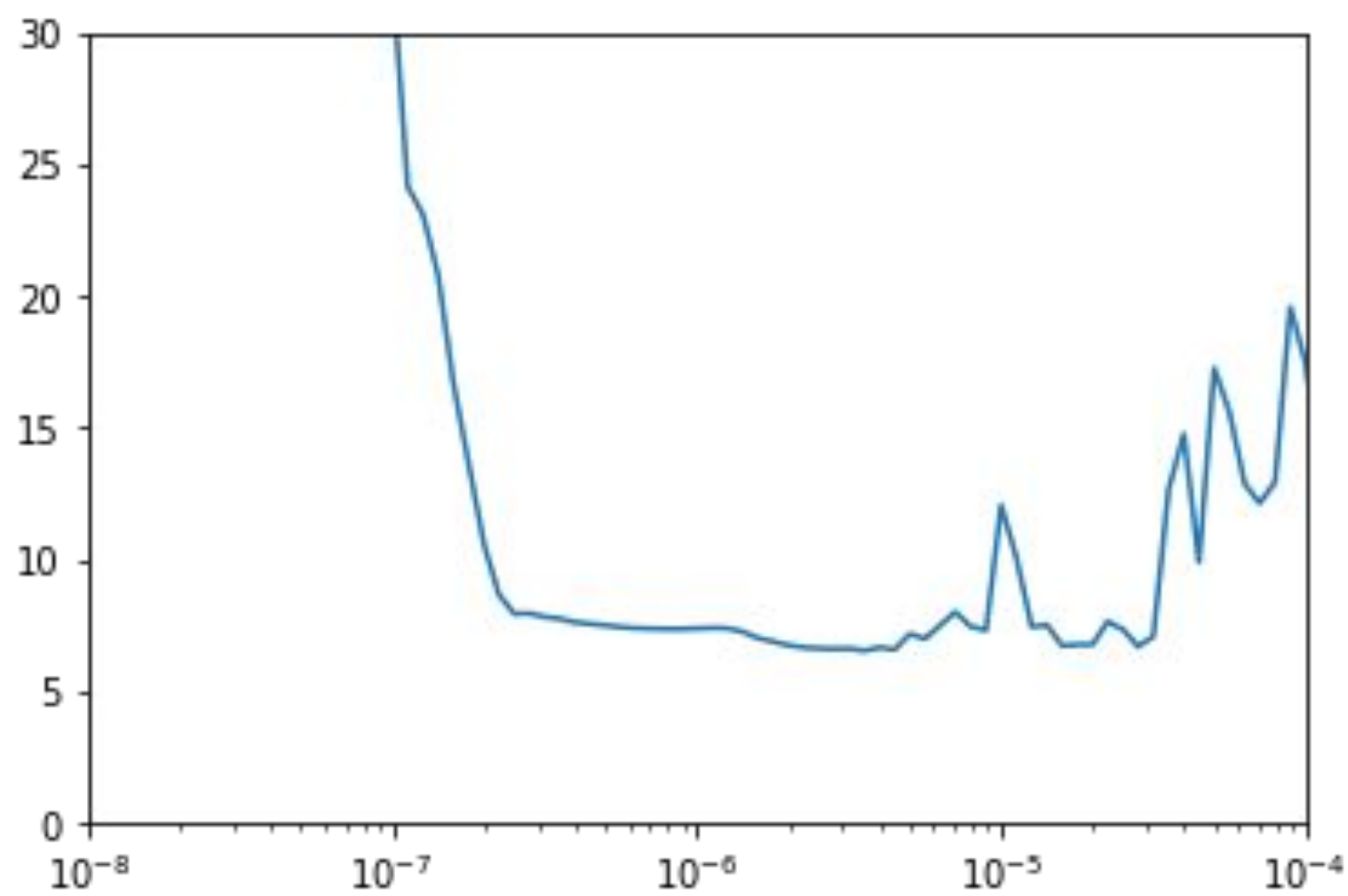
lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-8 * 10**(epoch / 20))

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](https://en.wikipedia.org/wiki/Huber_loss)

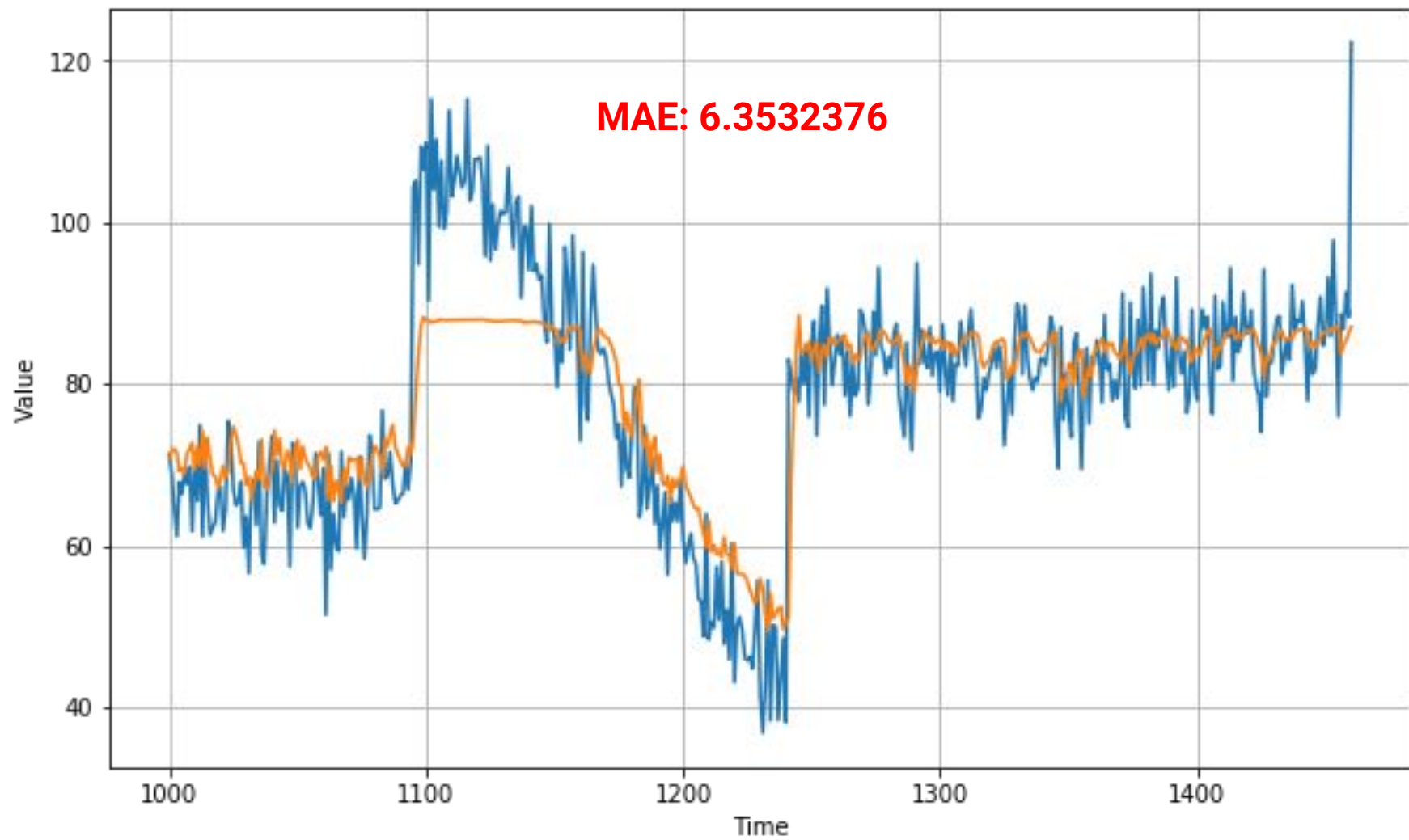


```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
```

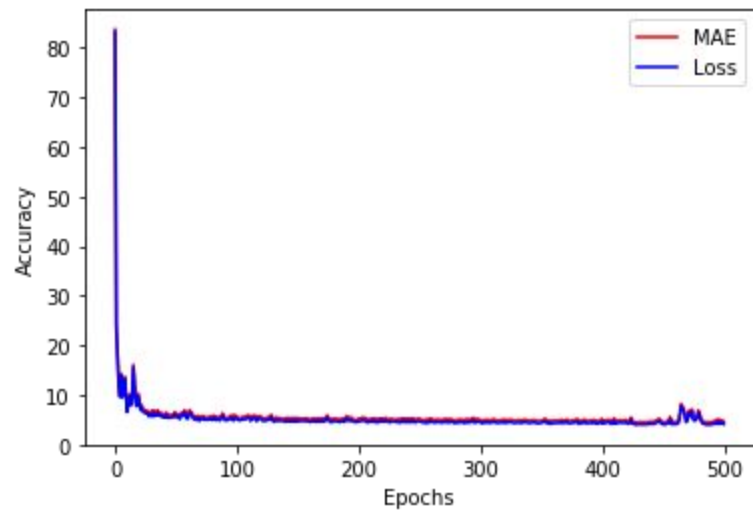
```
dataset = windowed_dataset(x_train, window_size, batch_size=128,
shuffle_buffer=shuffle_buffer_size)
```

```
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)
])
```

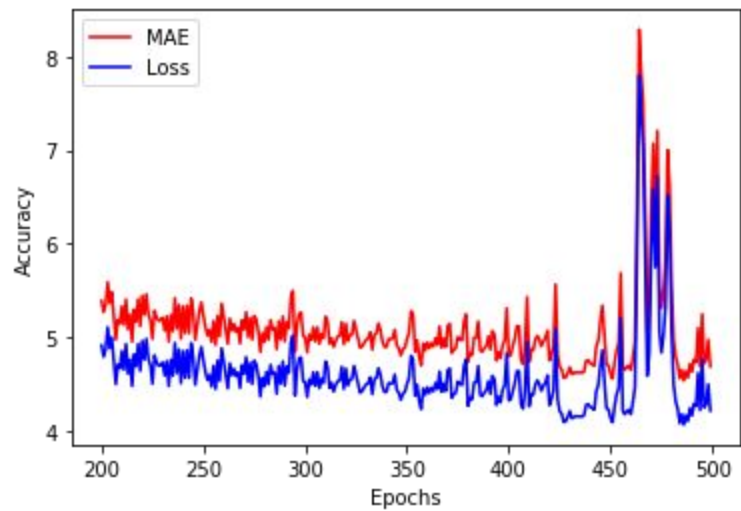
```
optimizer = tf.keras.optimizers.SGD(learning_rate=5e-6, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer, metrics=["mae"])
history = model.fit(dataset, epochs=500)
```

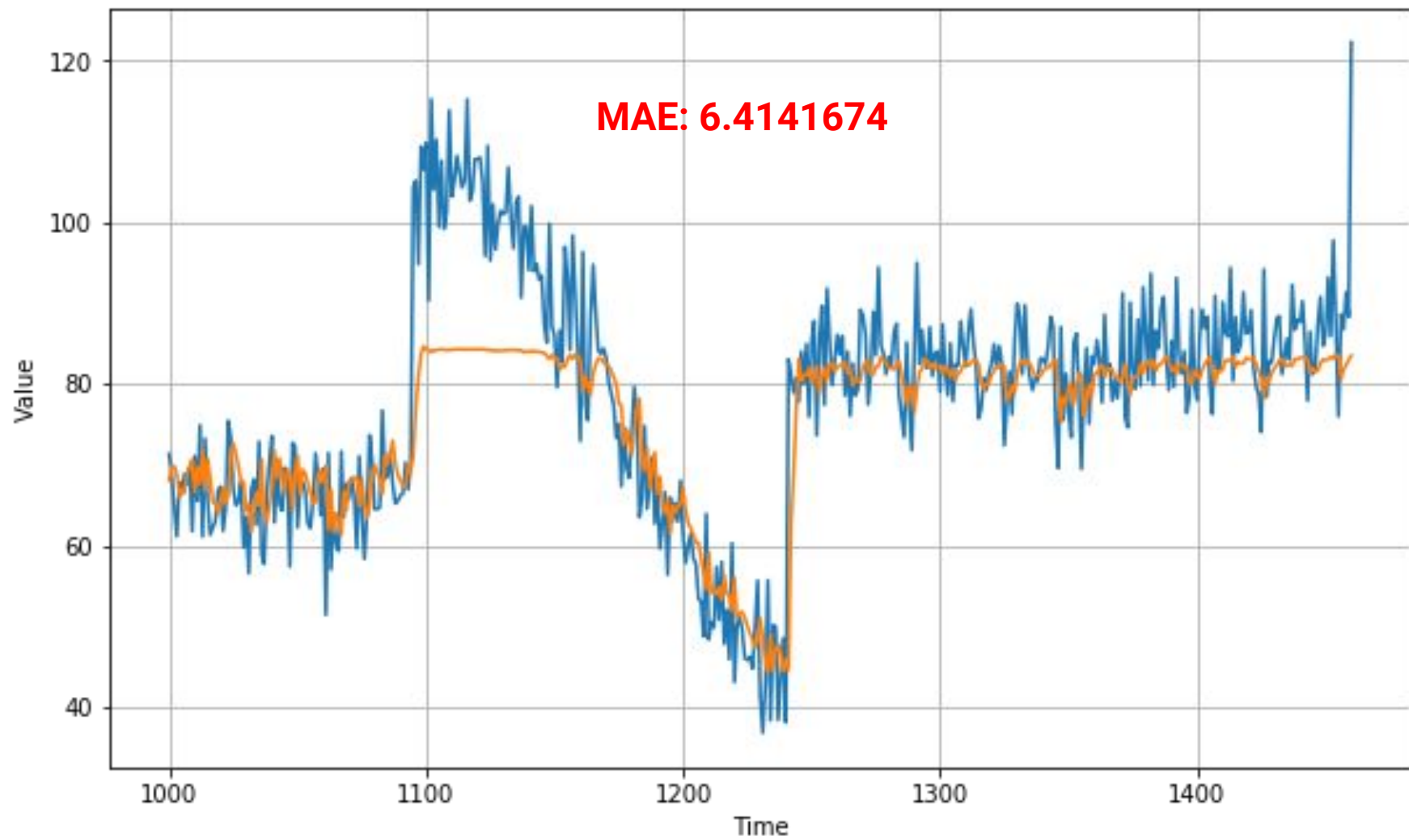


MAE and Loss

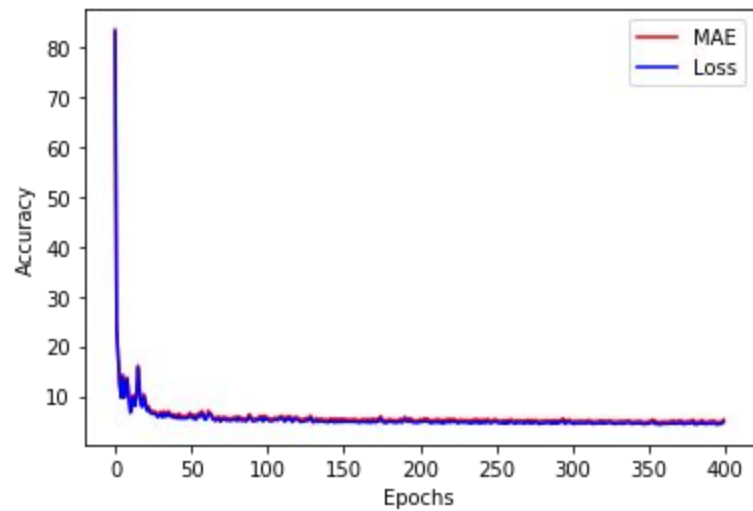


MAE and Loss

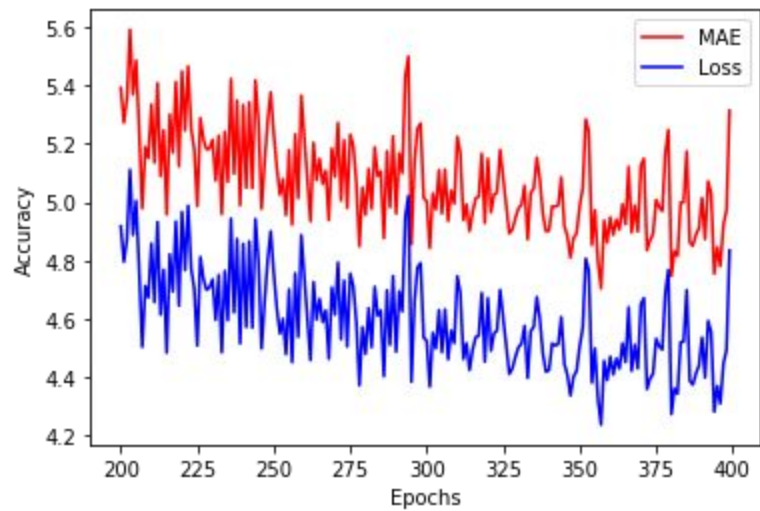




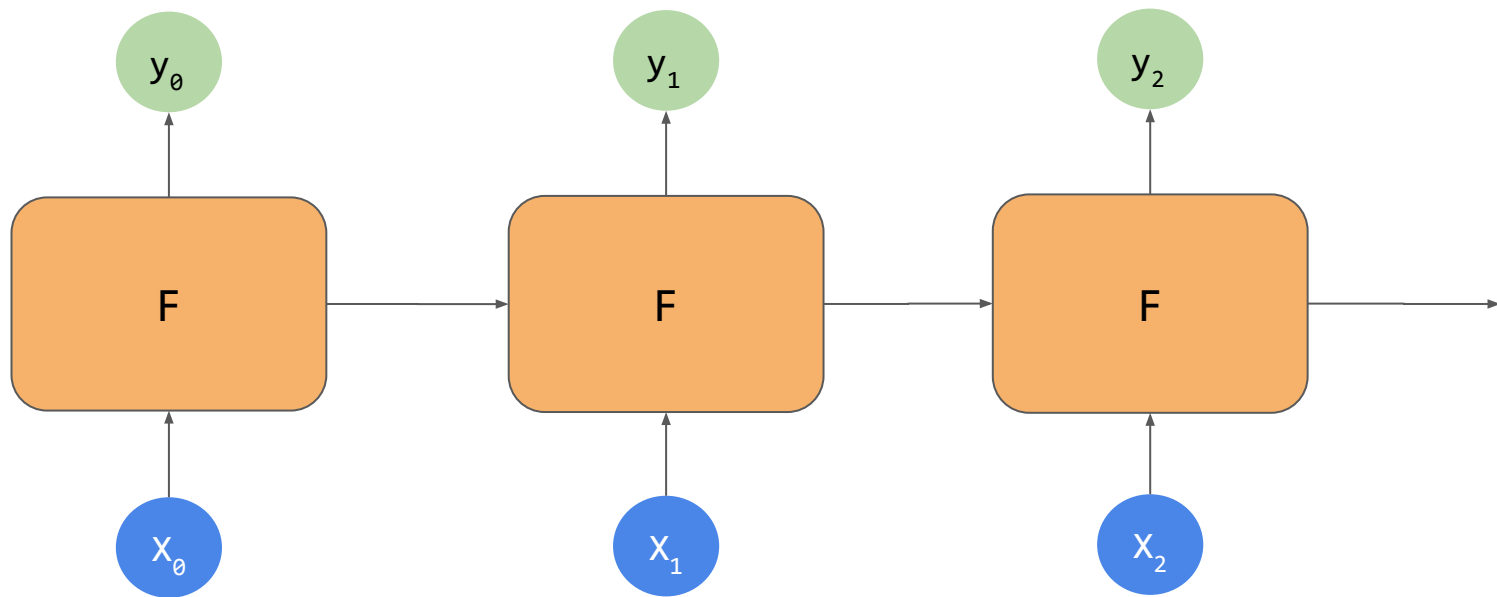
MAE and Loss

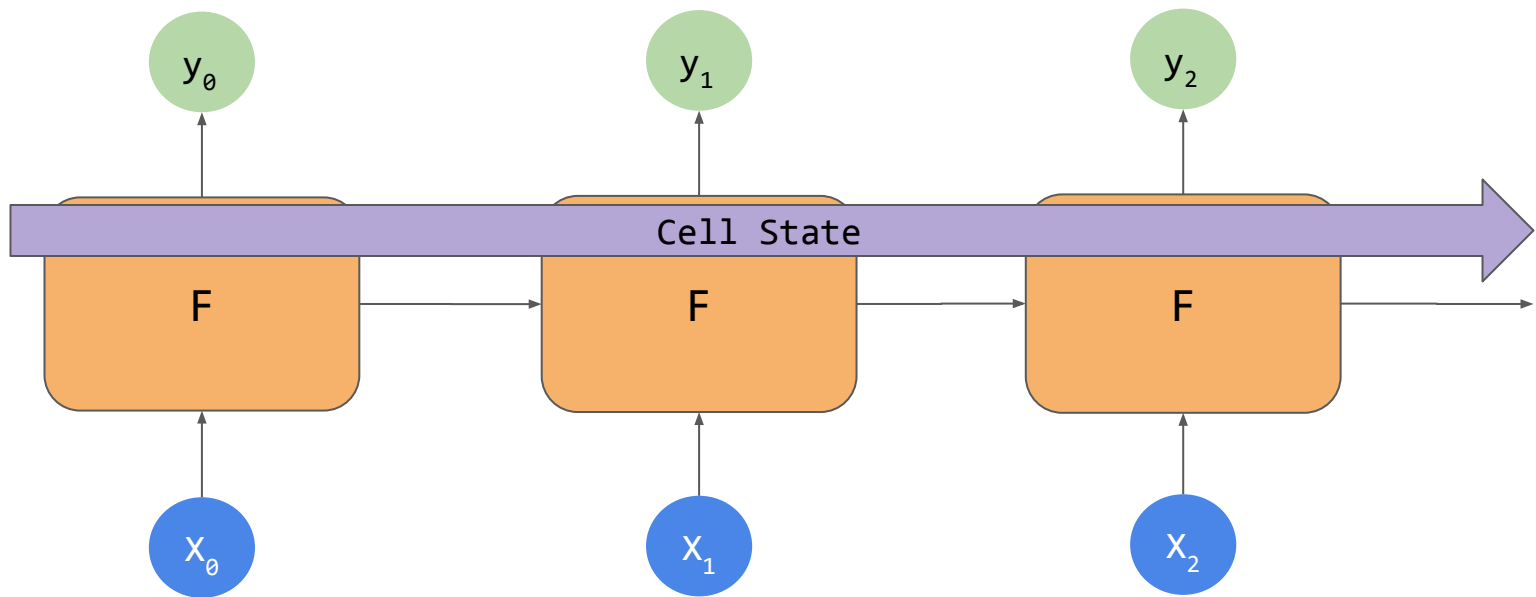


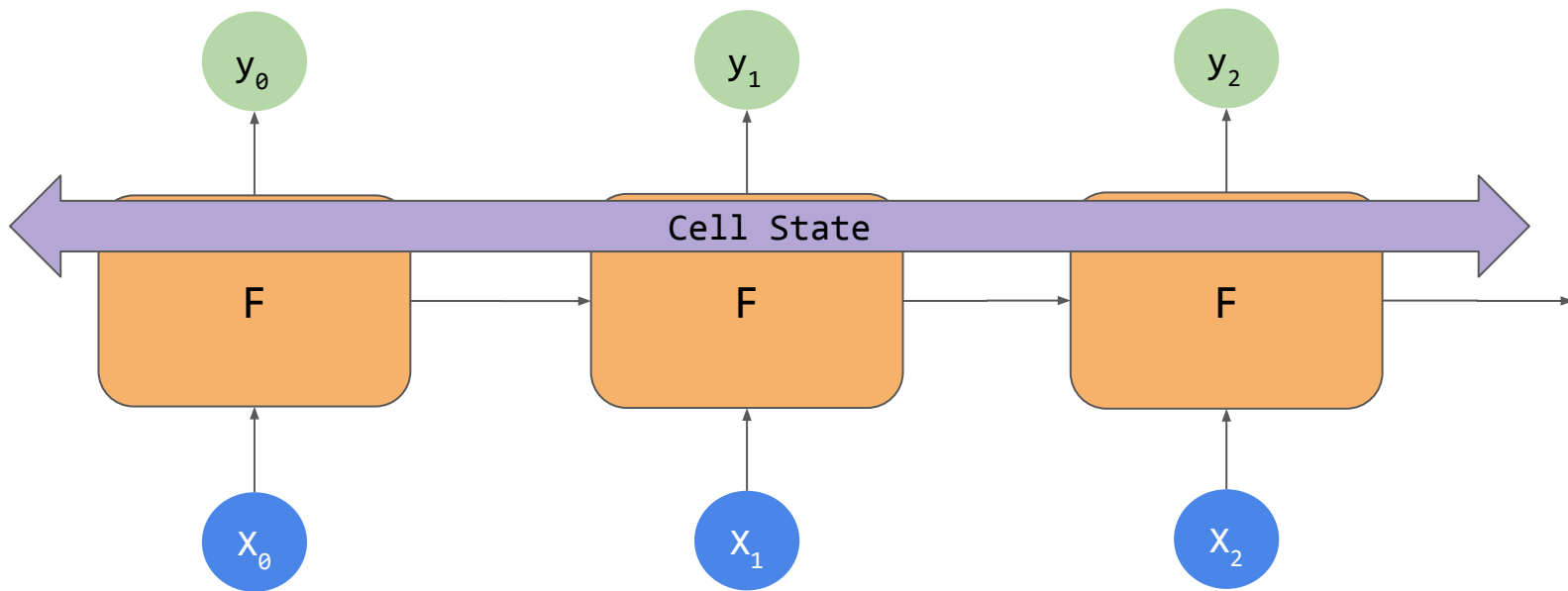
MAE and Loss







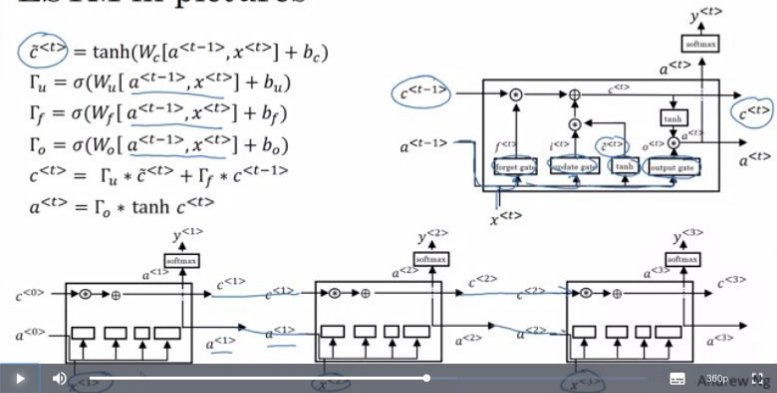




Long Short Term Memory (LSTM)

LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$



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)
])

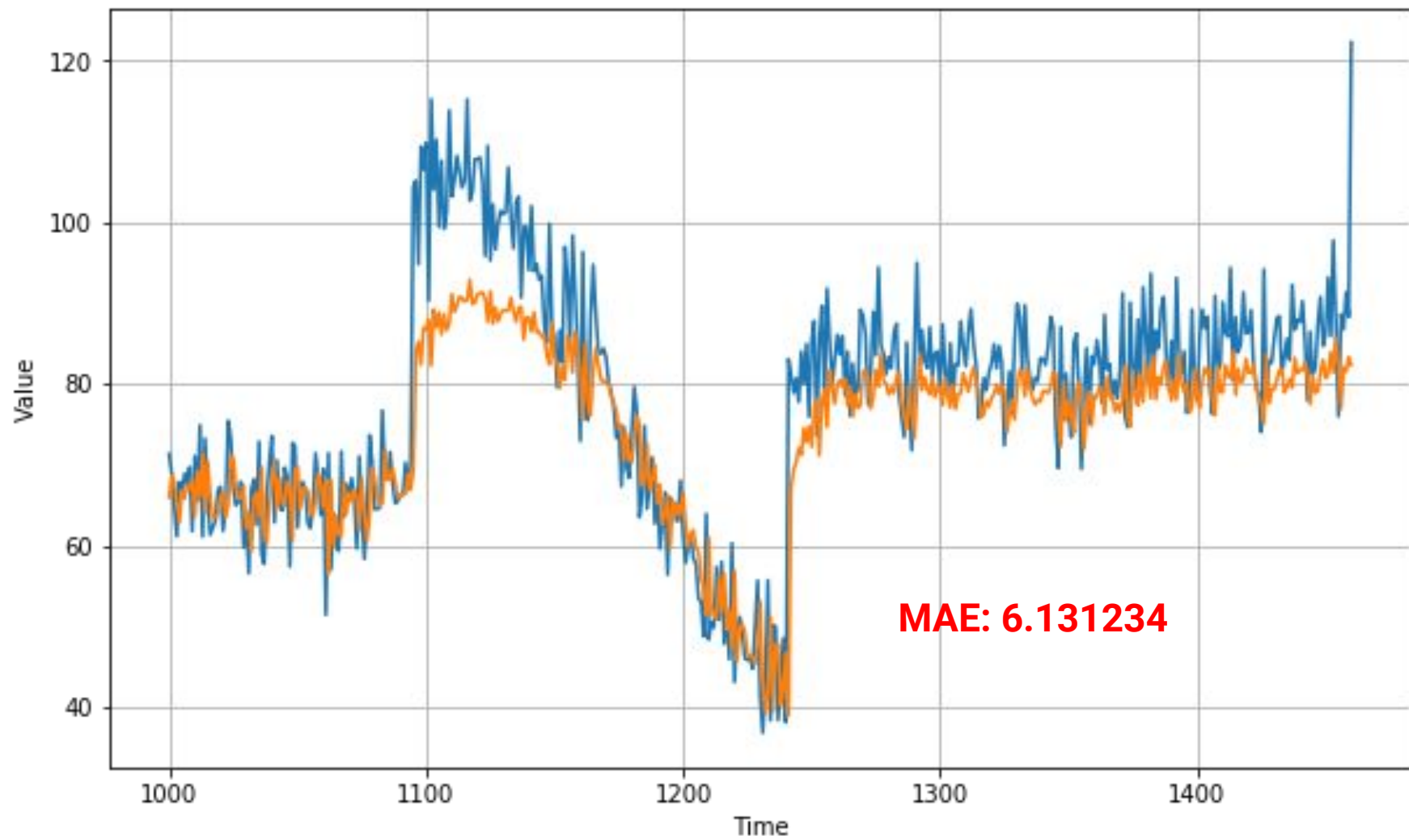
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6,
momentum=0.9))
model.fit(dataset, epochs=100, verbose=0)
```



```
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)
```



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

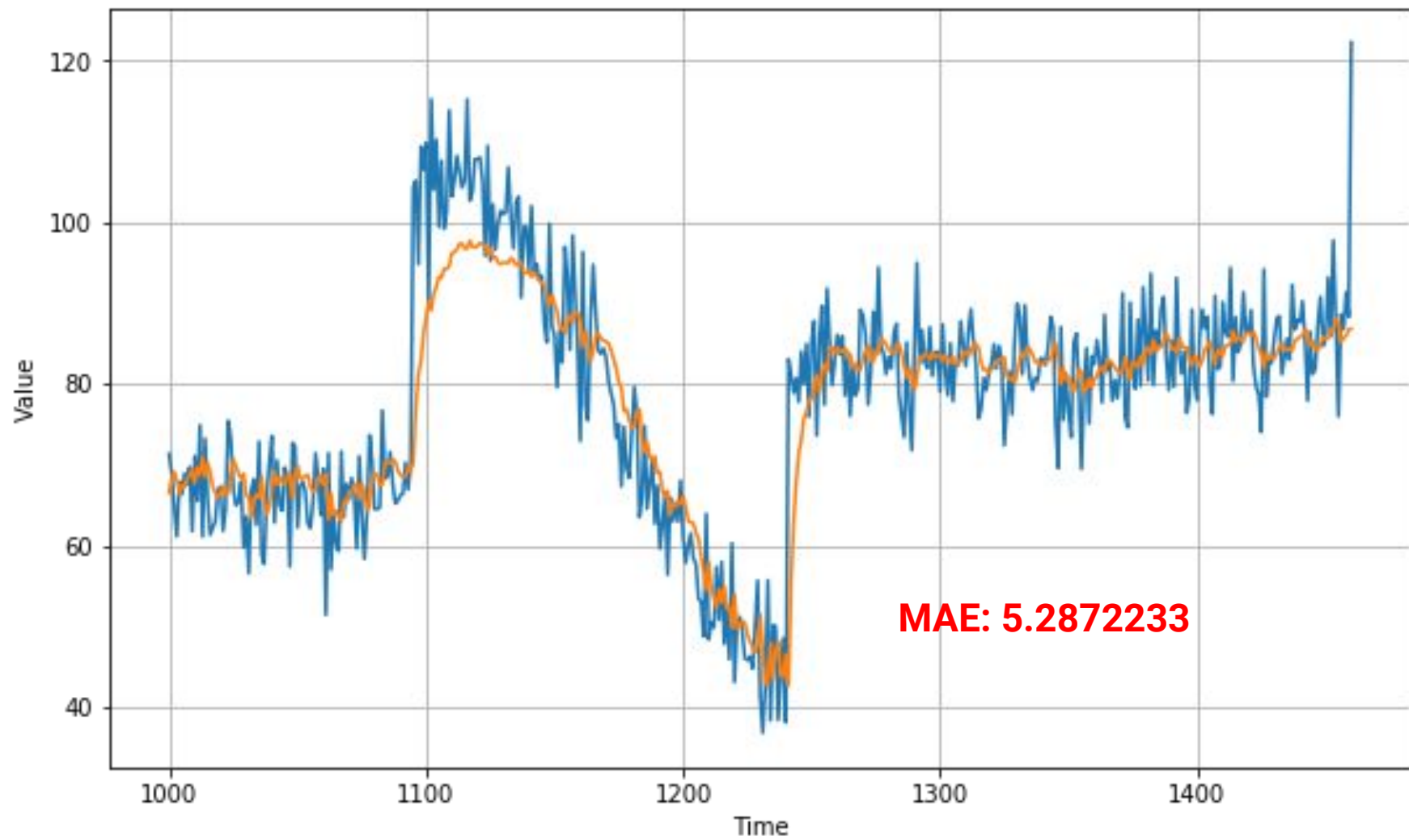
model = tf.keras.models.Sequential([
    tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                           input_shape=[None]),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(1),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



```
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)
```

