```
tf.keras.layers.Lambda(lambda x: tf.abs(x))
tf.keras.layers.Lambda(lambda x: tf.abs(x))
tf.keras.layers.Lambda(lambda x: tf.abs(x))
```

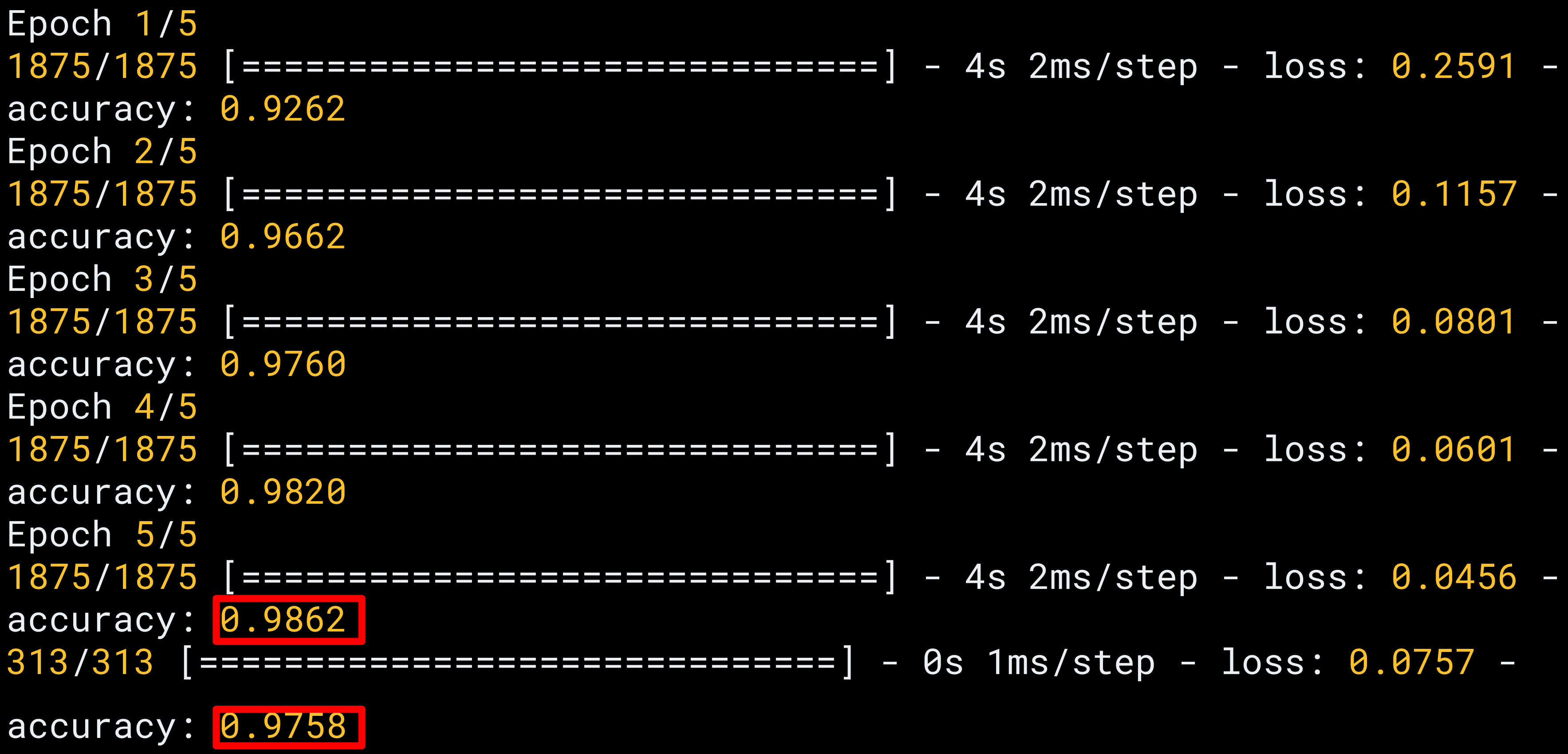
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

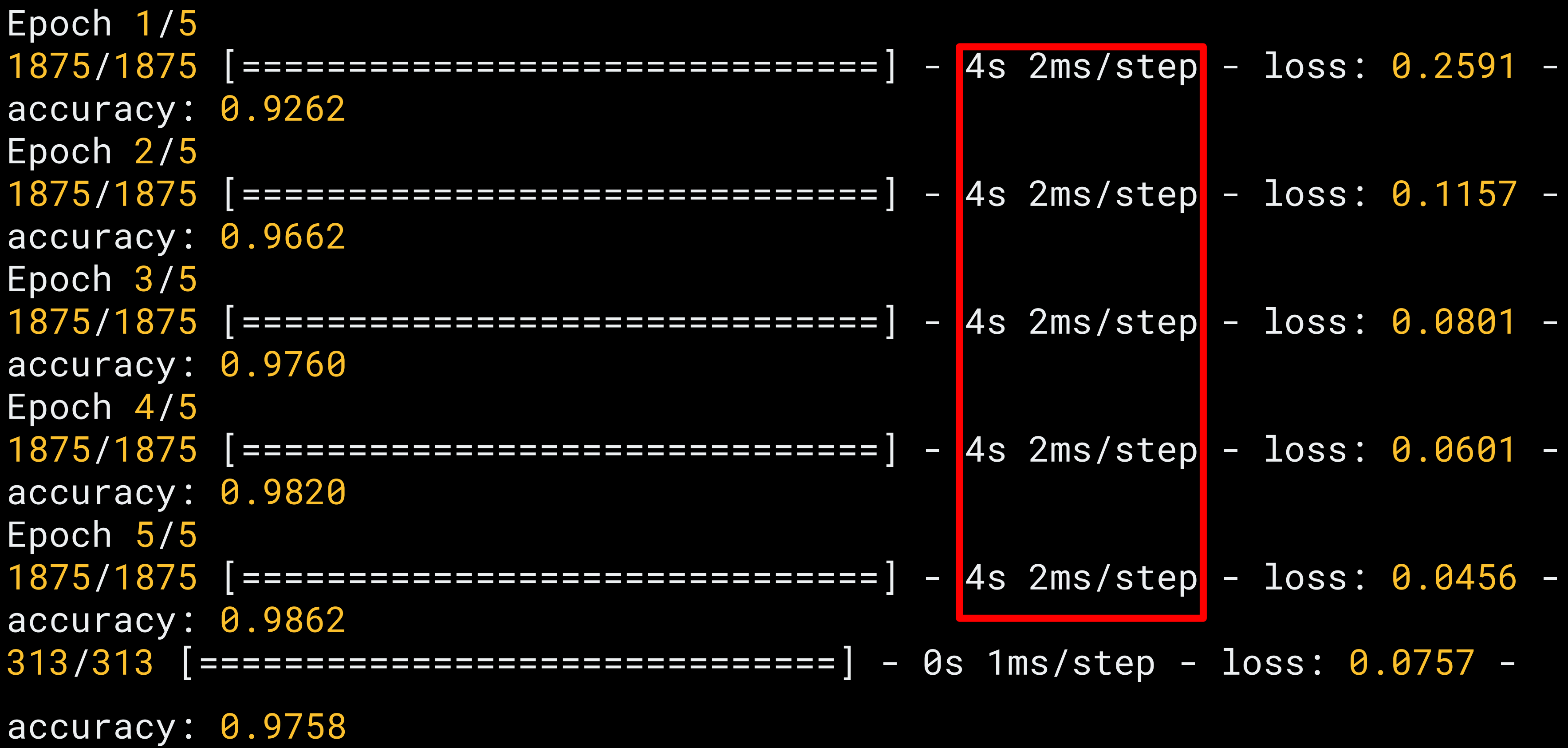
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
if(x>0):
    return x
else:
    return 0
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2591 -
accuracy: 0.9262
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1157 -
accuracy: 0.9662
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0801 -
accuracy: 0.9760
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0601 -
accuracy: 0.9820
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0456 -
accuracy: 0.9862
313/313 [==============================] - 0s 1ms/step - loss: 0.0757 -

accuracy: 0.9758
```



```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2591 -
accuracy: 0.9262
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1157 -
accuracy: 0.9662
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0801 -
accuracy: 0.9760
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0601 -
accuracy: 0.9820
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0456 -
accuracy: 0.9862
313/313 [==============================] - 0s 1ms/step - loss: 0.0757 -

accuracy: 0.9758
```



```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2591 -
accuracy: 0.9262
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1157 -
accuracy: 0.9662
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0801 -
accuracy: 0.9760
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0601 -
accuracy: 0.9820
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0456 -
accuracy: 0.9862
313/313 [==============================] - 0s 1ms/step - loss: 0.0757 -

accuracy: 0.9758
```

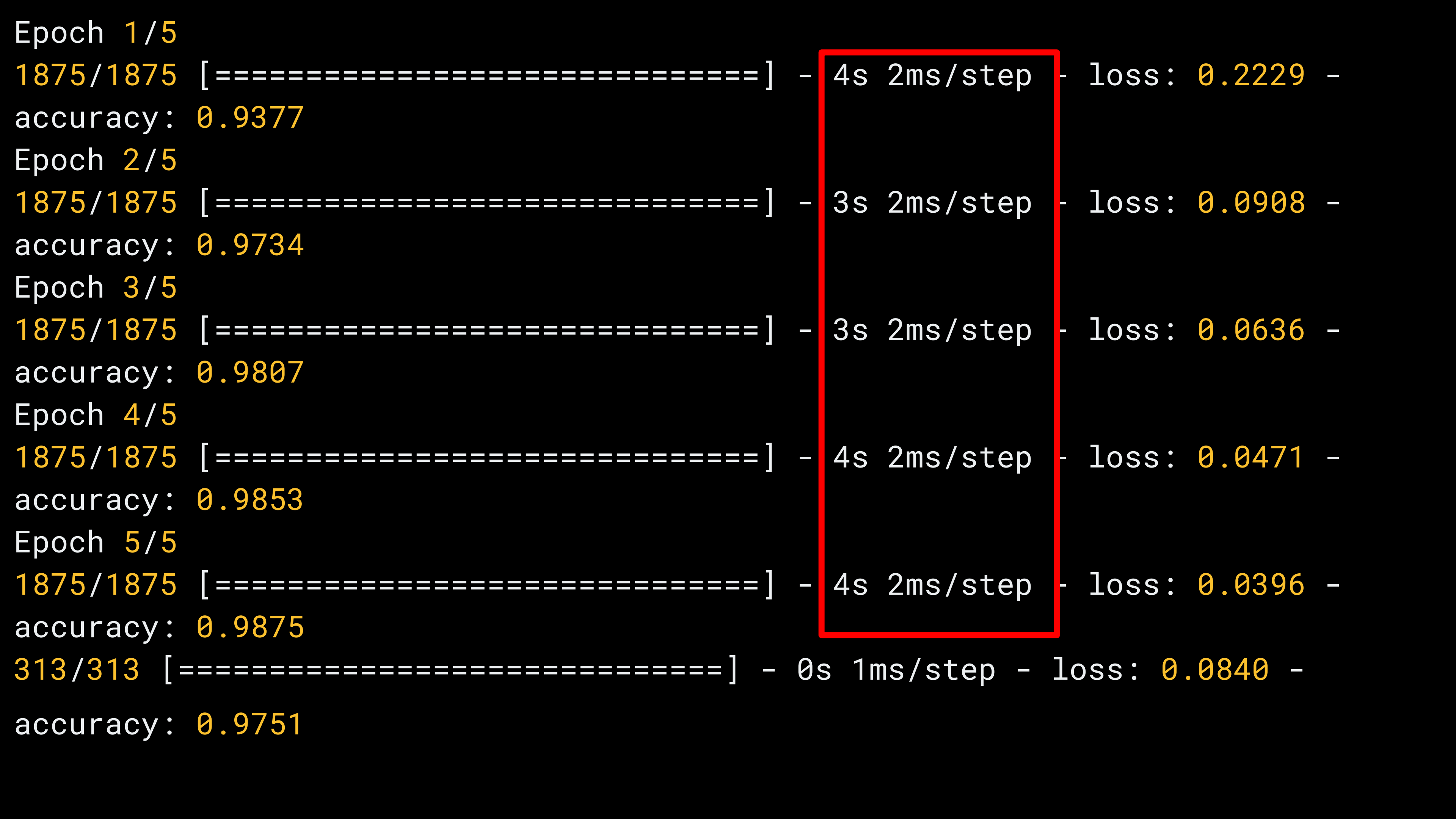
```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3568 -
accuracy: 0.8984
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2988 -
accuracy: 0.9170
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2880 -
accuracy: 0.9192
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2821 -
accuracy: 0.9205
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2763 -
accuracy: 0.9227
313/313 [==============================] - 0s 1ms/step - loss: 0.3031 -
accuracy: 0.9154
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3568 -
accuracy: 0.8984
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2988 -
accuracy: 0.9170
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2880 -
accuracy: 0.9192
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2821 -
accuracy: 0.9205
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2763 -
accuracy: 0.9227
313/313 [==============================] - 0s 1ms/step - loss: 0.3031 -
accuracy: 0.9154
```

```python
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(lambda x: tf.abs(x)),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(lambda x: tf.abs(x)),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2229 -
accuracy: 0.9377
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0908 -
accuracy: 0.9734
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0636 -
accuracy: 0.9807
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0471 -
accuracy: 0.9853
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0396 -
accuracy: 0.9875
313/313 [==============================] - 0s 1ms/step - loss: 0.0840 -
accuracy: 0.9751
```



```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2229 -
accuracy: 0.9377
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0908 -
accuracy: 0.9734
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0636 -
accuracy: 0.9807
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0471 -
accuracy: 0.9853
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0396 -
accuracy: 0.9875
313/313 [==============================] - 0s 1ms/step - loss: 0.0840 -

accuracy: 0.9751
```

```
def my_relu(x):
  return K.maximum(0.0, x)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```python
def my_relu(x):
  return K.maximum(0.0, x)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```python
def my_relu(x):
  return K.maximum(0.0, x)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
def my_relu(x):
  return K.maximum(0.0, x)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
def my_relu(x):
  return K.maximum(0.5, x)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Some commonly used layers

Convolutional

Conv1D/Conv2D/Conv3D

SeparableConv2D

DepthwiseConv2D

Recurrent

LSTM

GRU

Pooling

MaxPooling2D

AveragePooling2D

GlobalAveragePooling2D

Merge

Add

Subtract

Multiply

Activations (Advanced)

LeakyReLU

PReLU

ELU

Core

Activation

Lambda

Input

Dense

Dropout

BatchNormalization

Some commonly used layers

Convolutional

Conv1D/Conv2D/Conv3D

SeparableConv2D

DepthwiseConv2D

Recurrent

LSTM

GRU

Pooling

MaxPooling2D

AveragePooling2D

GlobalAveragePooling2D

Merge

Add

Subtract

Multiply

Activations (Advanced)

LeakyReLU

PReLU

ELU

Core

Activation

Lambda

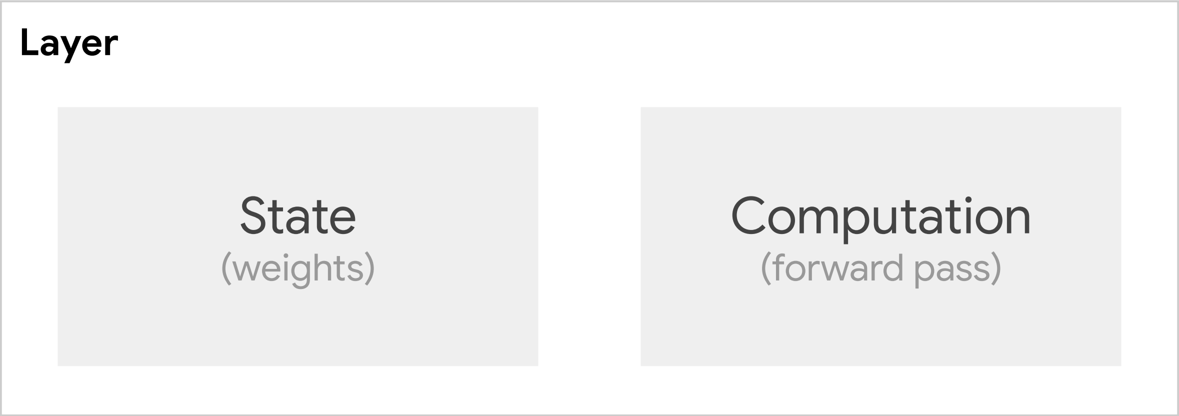
Input

Dense

Dropout

BatchNormalization

What is a Layer?

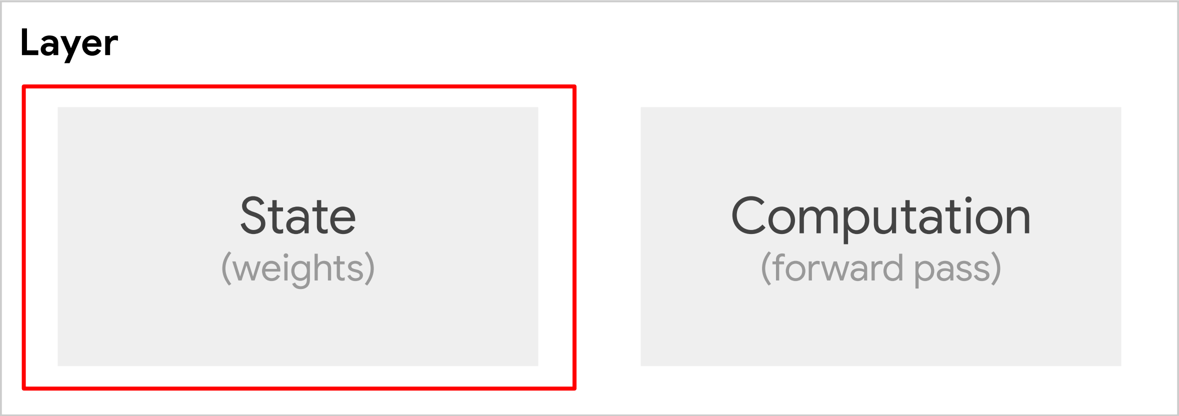


Layer

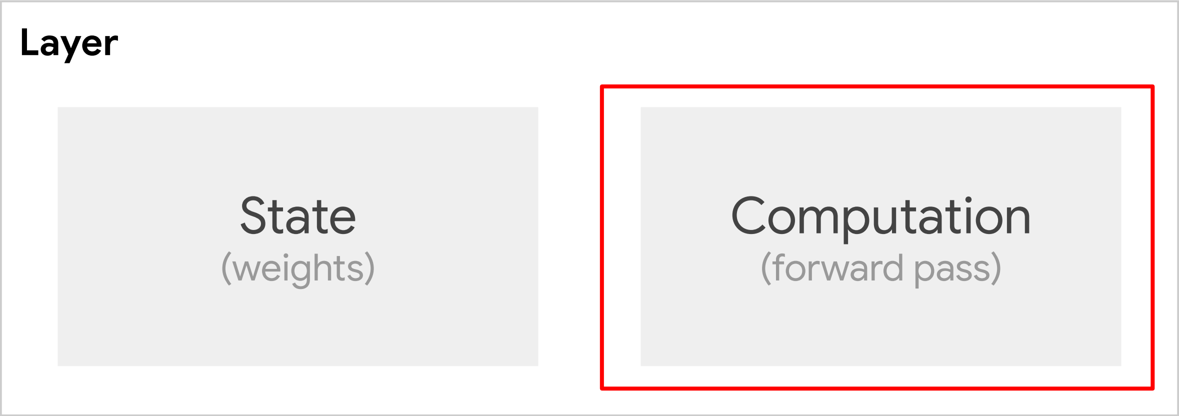
State
(weights)

Computation
(forward pass)

What is a Layer?



What is a Layer?

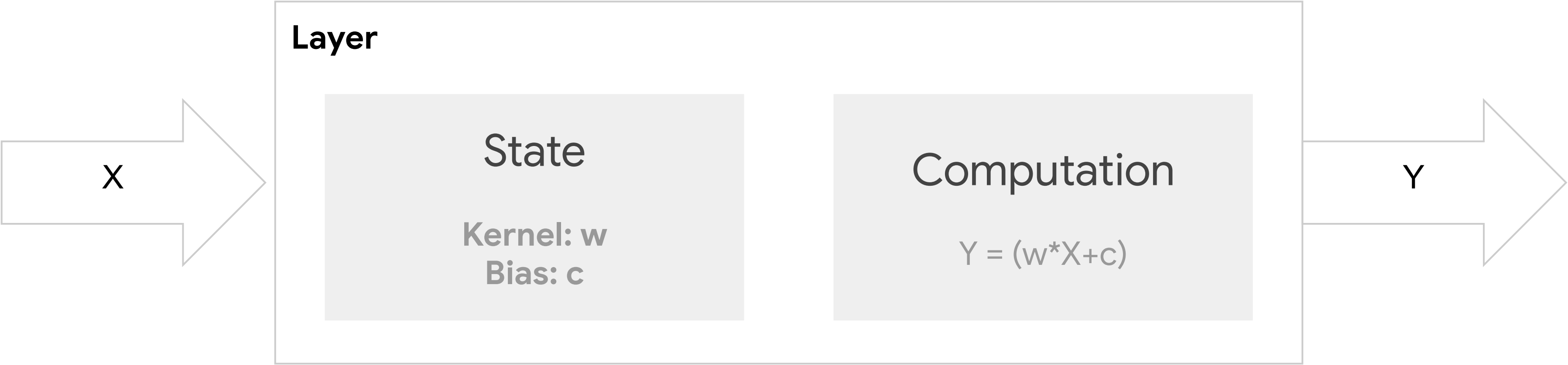


Layer

State
(weights)

Computation
(forward pass)

Simple Dense Layer



X

Layer

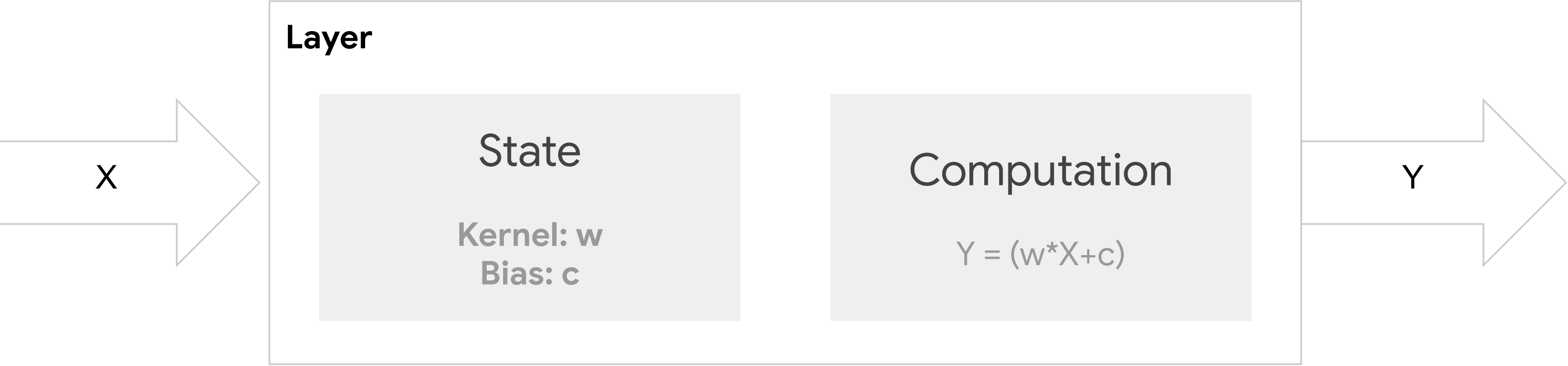
State

Kernel: w
Bias: c

Computation

Y = (w*X+c)

Y


Layer
X
State
Kernel: w
Bias: c
Computation
Y = (w*X+c)
Y

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```python
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```
[<tf.Variable 'simple_dense_7/kernel:0' shape=(1, 1)
dtype=float32, numpy=array([[0.03688493]], dtype=float32)>,

<tf.Variable 'simple_dense_7/bias:0' shape=(1,)
dtype=float32, numpy=array([0.], dtype=float32)>]
```

```python
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

```
   [<tf.Variable 'simple_dense_7/kernel:0' shape=(1, 1)
   dtype=float32, numpy=array([[0.03688493]], dtype=float32)>,

   <tf.Variable 'simple_dense_7/bias:0' shape=(1,)
   dtype=float32, numpy=array([0.], dtype=float32)>]
```

```
my_dense = SimpleDense(units=1)
x = tf.ones((1, 1))
y = my_dense(x)
print(my_dense.variables)
```

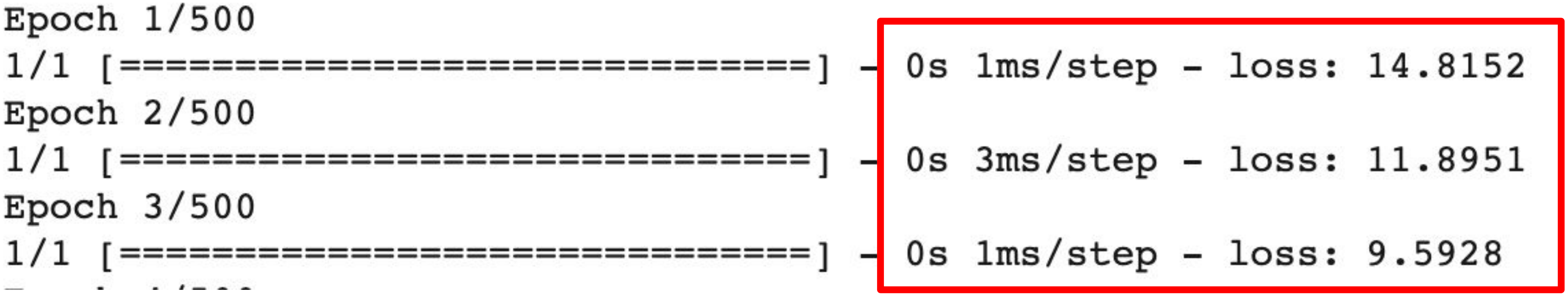
```
[<tf.Variable 'simple_dense_7/kernel:0' shape=(1, 1)
dtype=float32, numpy=array([[0.03688493]], dtype=float32)>,

<tf.Variable 'simple_dense_7/bias:0' shape=(1,)
dtype=float32, numpy=array([0.], dtype=float32)>]
```

```
import numpy as np

xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model = tf.keras.Sequential([SimpleDense(units=1)])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(xs, ys, epochs=500,verbose=0)
print(model.predict([10.0]))
```

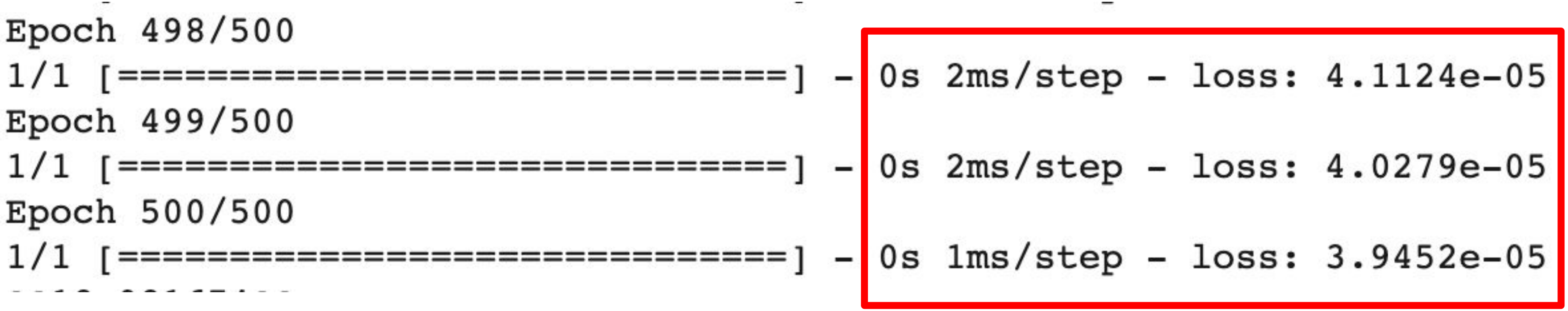
Expected Answer: 19 (y=2x-1)

Actual Answer: 0.36

(W = 0.036, B=0 => Y=.036 * 10 + 0 = 0.36)



```
Epoch 1/500
1/1 [==============================] - 0s 1ms/step - loss: 14.8152
Epoch 2/500
1/1 [==============================] - 0s 3ms/step - loss: 11.8951
Epoch 3/500
1/1 [==============================] - 0s 1ms/step - loss: 9.5928
```



```
Epoch 498/500
1/1 [==============================] - 0s 2ms/step - loss: 4.1124e-05
Epoch 499/500
1/1 [==============================] - 0s 2ms/step - loss: 4.0279e-05
Epoch 500/500
1/1 [==============================] - 0s 1ms/step - loss: 3.9452e-05
```

```
import numpy as np

xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model = tf.keras.Sequential([SimpleDense(units=1)])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(xs, ys, epochs=500,verbose=0)
print(model.predict([10.0]))
```

```
import numpy as np

xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model = tf.keras.Sequential([SimpleDense(units=1)])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(xs, ys, epochs=500,verbose=0)
print(model.predict([10.0]))
```

```
[[18.981468]]
```

```
[<tf.Variable
'sequential_15/simple_dense_19/kernel:0' shape=(1, 1)
dtype=float32, numpy=array([[1.9972587]],
dtype=float32)>, <tf.Variable
'sequential_15/simple_dense_19/bias:0' shape=(1,)
dtype=float32, numpy=array([-0.991501],
dtype=float32)>]
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  SimpleDense(128),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  SimpleDense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```python
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  SimpleDense(128),
  tf.keras.layers.Lambda(my_relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

 def __init__(self, units=32):
   super(SimpleDense, self).__init__()
   self.units = units

 def build(self, input_shape):  # Create the state of the layer (weights)
  w_init = tf.random_normal_initializer()
  self.w = tf.Variable(name="kernel",
    initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
    trainable=True)

  b_init = tf.zeros_initializer()
  self.b = tf.Variable(name="bias",
    initial_value=b_init(shape=(self.units,), dtype='float32'),
    trainable=True)

 def call(self, inputs):  # Defines the computation from inputs to outputs
   return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32):
      super(SimpleDense, self).__init__()
      self.units = units

  def build(self, input_shape):  # Create the state of the layer (weights)
    w_init = tf.random_normal_initializer()
    self.w = tf.Variable(name="kernel",
        initial_value=w_init(shape=(input_shape[-1], self.units), dtype='float32'),
        trainable=True)

    b_init = tf.zeros_initializer()
    self.b = tf.Variable(name="bias",
        initial_value=b_init(shape=(self.units,), dtype='float32'),
        trainable=True)

  def call(self, inputs):  # Defines the computation from inputs to outputs
      return tf.matmul(inputs, self.w) + self.b
```

```
class SimpleDense(Layer):

  def __init__(self, units=32, activation=None):
      super(SimpleDense, self).__init__()
      self.units = units
      self.activation = tf.keras.activations.get(activation)


  def call(self, inputs):
      return self.activation(tf.matmul(inputs, self.w) + self.b)
```

```
class SimpleDense(Layer):

  def __init__(self, units=32, activation=None):
      super(SimpleDense, self).__init__()
      self.units = units
      self.activation = tf.keras.activations.get(activation)


  def call(self, inputs):
      return self.activation(tf.matmul(inputs, self.w) + self.b)
```

```
class SimpleDense(Layer):

  def __init__(self, units=32, activation=None):
      super(SimpleDense, self).__init__()
      self.units = units
      self.activation = tf.keras.activations.get(activation)


  def call(self, inputs):
      return self.activation(tf.matmul(inputs, self.w) + self.b)
```

```python
class SimpleDense(Layer):

  def __init__(self, units=32, activation=None):
      super(SimpleDense, self).__init__()
      self.units = units
      self.activation = tf.keras.activations.get(activation)


  def call(self, inputs):
      return self.activation(tf.matmul(inputs, self.w) + self.b)
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  SimpleDense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  SimpleDense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```