Exercise 1

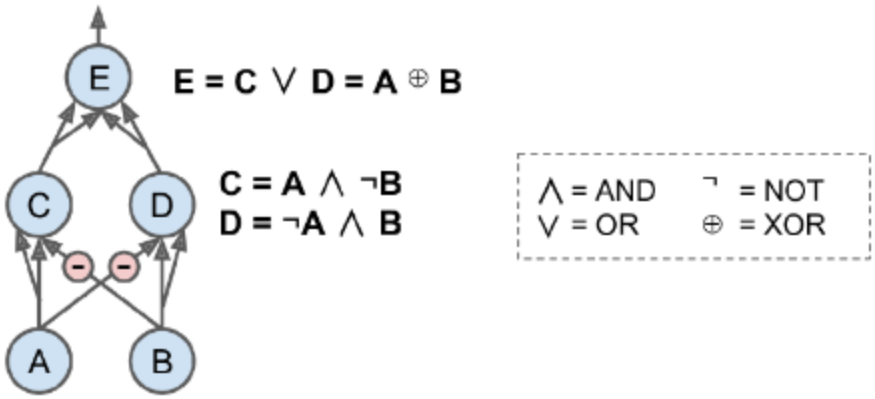
TensorFlow playground是一个方便的神经网络模拟器。在本练习中，您将只需点击几下就能训练几个二分类器，并调整模型的架构及其超参数，以获得一些关于神经网络如何工作以及它们的超参数做什么的直觉。花点时间来探索一下以下内容：

1. **通过神经网络学习到的模式**。尝试通过点击 RUN 按钮（左上角）来训练默认的神经网络。注意它如何快速为分类任务找到好的解决方案。第一隐藏层的神经元学习了简单的模式，而第二隐藏层的神经元学会了将第一隐藏层的简单模式组合成更复杂的模式。一般来说，图层越多，模式就会越复杂。
2. **激活功能**。尝试用ReLU激活功能替换tanh激活功能，然后再次训练网络。请注意，它找到解决方案的速度更快，但这一次的边界是线性的。这是由于ReLU函数的形状。
3. **局部最小值的风险**。修改网络架构，使其只有一个隐藏层和三个神经元。多次训练(重置网络权重，单击 Play 按钮旁边的 Reset 按钮)。请注意，训练的时间变化很大，有时甚至会停留在局部的最小值内。
4. **当神经网络太小时会发生什么**。移除一个神经元以保持只有两个。请注意，神经网络现在已经无法找到一个好的解决方案了，即使你尝试了很多次。该模型的参数太少，系统地无法适应训练集。
5. **当神经网络足够大时会发生什么**。将神经元的数量设置为8个，并多次训练该网络。请注意，它现在一直都是快速的，而且永远不会被卡住。这突出了神经网络理论中的一个重要发现：大型神经网络很少陷入局部最小值，即使它们做这些局部最优通常也几乎和全局最优一样好。然而，它们仍然会在很长一段时间内停留在平台期。
6. **在深度网络中梯度消失的风险**。选择螺旋数据集（“DATA”下右下角的数据集），并改变网络架构，有四个隐藏层，每层有8个神经元。请注意，训练需要更长的时间，而且经常会长时间停滞在高原上。还需要注意的是，最高层（右边）的神经元往往比最低层的神经元（左边）进化得更快。这个问题被称为消失梯度问题，可以通过更好的权重初始化和其他技术、更好的优化器（如AdaGrad或Adam）或批处理标准化（在第11章中讨论）来缓解。
7. **更进一步**。花一个小时左右的时间来研究其他参数，并了解它们在做什么，以建立对神经网络的直观理解。

Exercise 2

使用原始人工神经元绘制 ANN 计算 $A \oplus B$（其中⊕表示XOR操作）。提示：A⊕B =（A∧¬B）∨（¬A∧B）。

答案：



Exercise 3

为什么通常最好使用逻辑回归分类器而不是经典的感知器（即，使用感知器训练算法训练的一层阈值逻辑单元）？如何调整感知器，使其等同于逻辑回归分类器？

答案：

只有当数据集是线性可分离的，经典感知器才会收敛，并且它无法估计类概率。相反，即使数据集不是线性可分离的，逻辑回归分类器通常也会收敛到一个相当好的解决方案，并且会输出类概率。如果您将Perceptron的激活函数更改为sigmoid激活函数（如果有多个神经元，则为softmax激活函数），并且如果您使用梯度下降（Gradient Descent）（或最小化成本函数的其他优化算法，通常为交叉熵）对其进行训练，那么它就相当于Logistic回归分类器。

Exercise 4

为什么 sigmoid 激活函数是训练第一个 MLPs 的关键因素？

答案：

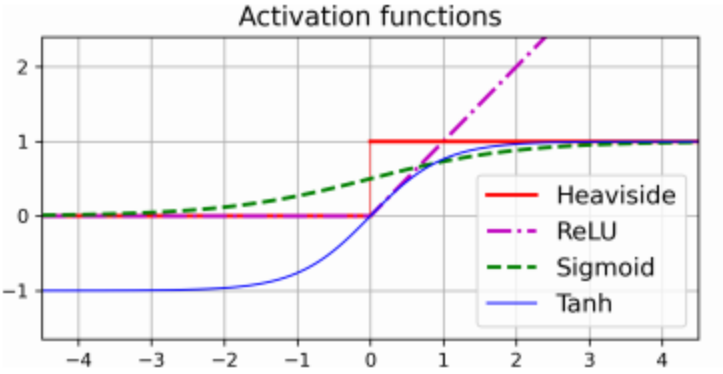
sigmoid 激活函数是训练第一个 MLPs 的关键成分，因为它的导数总是非零的，所以梯度下降总是可以沿着斜坡向下滚动。当激活函数为 step 阶梯函数时，梯度下降无法移动，因为根本没有坡度。

Exercise 5

列举三个流行的激活函数。你能画出来吗？

答案：

常用的激活函数包括 step 函数、sigmoid 函数、tanh 函数和 ReLU 函数。有关其他示例，如ELU和ReLU函数的变体，请参见第11章。



Exercise 6

假设你有一个由一个包含10个神经元输入层的MLP，然后是一个包含50个人工神经元的隐藏层，最后一个输出层有3个人工神经元。所有的人工神经元都使用ReLU激活功能。

1. What is the shape of the input matrix $\boldsymbol{X}$?
2. What are the shapes of the hidden layer's weight matrix $\boldsymbol{W}_h$ and bias vector $\boldsymbol{b}_h$?
3. What are the shapes of the output layer's weight matrix $\boldsymbol{W}_o$ and bias vector $\boldsymbol{b}_o$?
4. What is the shape of the network's output matrix $\boldsymbol{Y}$?

5. Write the equation that computes the network's output matrix $\boldsymbol{Y}$ as a function of $\boldsymbol{X}$, $\boldsymbol{W}_h$, $\boldsymbol{b}_h$, $\boldsymbol{W}_o$, and $\boldsymbol{b}_o$.

答案：

考虑到问题中描述的MLP，由一个具有10个穿透神经元的输入层组成，然后是一个具有50个人工神经元的隐藏层，最后是一个带有3个人工神经元的输出层，其中所有人工神经元都使用ReLU激活函数：

1. 输入矩阵 $\boldsymbol{X}$ 的形状是 $m \times 10$，其中 m 表示 batch 大小
2. 隐藏层权重矩阵 $\boldsymbol{W}_h$ 的形状是 10×50 ，bias 向量 $\boldsymbol{b}_h$ 的长度是50。
3. 输出层权重矩阵 $\boldsymbol{W}_o$ 的形状是 50×3，bias 向量 $\boldsymbol{b}_o$ 的长度是3。
4. 输出矩阵 $\boldsymbol{Y}$ 的形状是 $m \times 3$ 。
5. $\boldsymbol{Y} = ReLU(ReLU(\boldsymbol{X}\boldsymbol{W}_h + \boldsymbol{b}_h)\boldsymbol{W}_o + \boldsymbol{b}_o)$

Exercise 7

如果你想把电子邮件分类为垃圾右键还是普通邮件，需要多少神经元。在输出层中应该使用什么激活函数？如果你想处理MNIST，你在输出层需要多少神经元，你应该使用哪个激活函数？如何让你的网络像第二章那样预测房价呢？

答案：

要将电子邮件分类为垃圾邮件或普通邮件，你只需要在神经网络的输出层中有一个神经元，指示电子邮件是垃圾邮件的概率。在估计概率时，通常会在输出层中使用sigmoid激活函数。

如果你想处理MNIST，你需要10个神经元在输出层，你必须用softmax激活函数代替sigmoid函数，它可以处理多个类，每个类输出一个概率。

如果你想让你的神经网络像第二章那样预测房价，那么你需要一个输出神经元，在输出层中根本不使用激活函数。**注意**：当要预测的值可以变化许多数量级时，您可能希望预测目标值的对数，而不是直接预测目标值。简单地计算神经网络输出的指数将给出估计值（因为$exp(logv) = v$）。

Exercise 8

什么是反向传播，它是如何工作的?反向传播和 reverse-mode autodiff 有何区别?

答案：

反向传播是一种用于训练人工神经网络的技术。它首先计算每个模型参数（所有权重和偏差）的成本函数梯度，然后使用这些梯度执行梯度下降步骤。该反向传播步骤通常使用许多训练批次执行数千或数百万次，直到模型参数收敛到（希望）最小化成本函数的值。

为了计算梯度，反向传播使用reverse-mode autodiff（虽然在反向传播被发明时并没有这样称呼，而且它已经被重新发明了好几次）。reverse-mode autodiff通过计算图执行正向传递，计算当前训练批的每个节点的值，然后执行反向传递，一次计算所有梯度。

那么有什么区别呢？反向传播是指使用多个反向传播步骤训练人工神经网络的整个过程，每个步骤计算梯度并使用它们执行梯度下降步骤。相反，反向模式autodiff只是一种有效计算梯度的技术，它恰好被反向传播所使用。

Exercise 9

您能列出您可以在一个基本的MLP中调整的所有超参数吗？如果MLP过度拟合训练数据，你如何调整这些超参数来解决问题？

答案：

这里列出了基本MLP中可以调整的所有超参数：隐藏层的数量、每个隐藏层中的神经元数量以及每个隐藏层和输出层中使用的激活函数。通常，ReLU激活函数是隐藏层的良好默认。对于输出层，通常情况下，您将需要用于二分类的sigmoid激活函数，用于多分类的softmax激活函数，或用于回归的无激活函数。

如果MLP过拟合训练数据，可以尝试减少隐藏层的数量，并减少每个隐藏层的神经元数量。

Exercise 10

在MNIST数据集上训练一个深度MLP(您可以使用tf.keras.datasets.mnist.load_data()加载它).看看你是否可以通过人工调整超参数获得超过98%的精度。尝试使用本章中提出的方法来寻找最佳的学习率（即，通过指数级增长学习率，绘制损失，并找到损失上升的点）。接下来，尝试使用Keras Tuner来调整超参数--保存检查点、使用早期停止和使用TensorBoard绘制学习曲线。

答案：

In [36]:
```
import tensorflow as tf
from matplotlib import pyplot as plt
import numpy as np
from pathlib import Path
from time import strftime
import tensorboard
```

In [2]:
```
# load the dataset

(X_train_full, y_train_full), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.
npz
11490434/11490434 [==============================] - 2s 0us/step
```

In [3]:
```
# the MNIST training set contains 60,000 grayscale images
# each 28x28 pixels

X_train_full.shape
```

Out[3]:
```
(60000, 28, 28)
```

In [4]:
```
# Each pixel intensity is also represented as a byte (0 to 255):

X_train_full.dtype
```

Out[4]:
```
dtype('uint8')
```

In [5]:
```
# split the full training set into a validation set and a training set
# scale the pixel intensities down to the 0-1 range and convert them to floats

X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

In [15]:
```
# plot an image using Matplotlib's imshow() function
# with a 'binary' color map

plt.imshow(X_train[1], cmap="binary")
plt.axis('off')
plt.show()
```

In [14]:
```
y_train
```

Out[14]:
```
array([7, 3, 4, ..., 5, 6, 8], dtype=uint8)
```

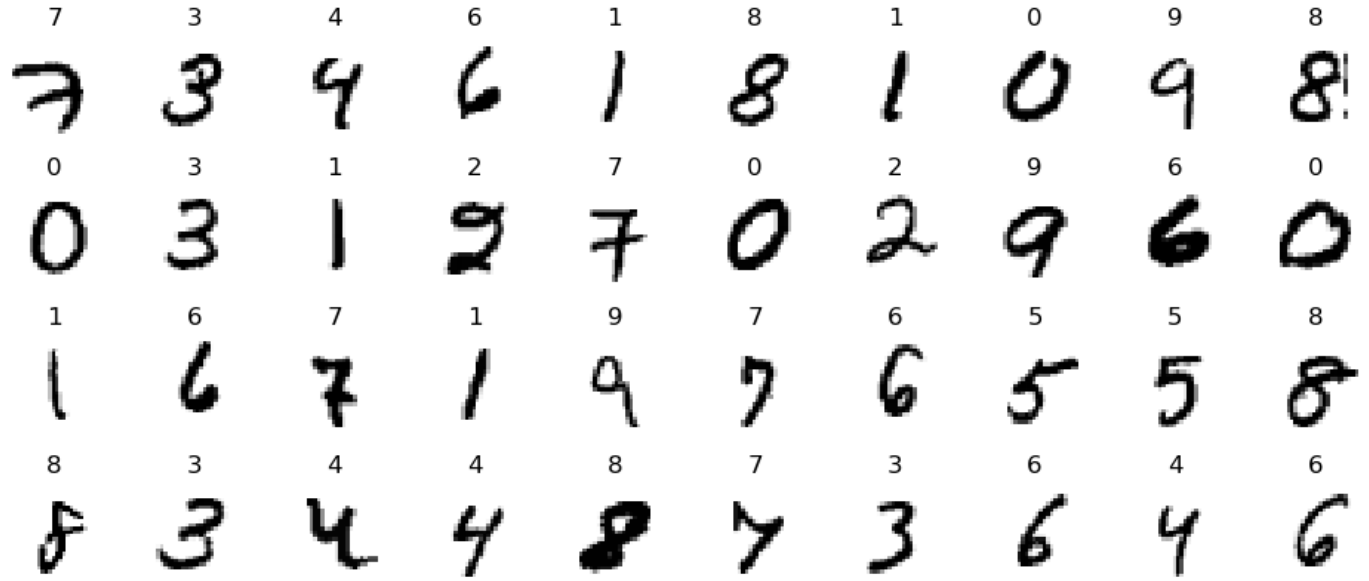
In [16]:
```
# the validation set contains 5,000 images
# the test set contains 10,000 images

X_valid.shape
```

Out[16]:
```
(5000, 28, 28)
```

In [17]:
```
X_test.shape
```

Out[17]:
```
(10000, 28, 28)
```

In [18]:
```
# Let's take a look at a sample of the images in the dataset

n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))

for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(y_train[index])


plt.subplots_adjust(wspace=0.2, hspace=0.5)
plt.show()
```



In [19]:
```
# build a simple dense network and find the optimal learning rate
# need a callback to grow the learning rate at each iteration

K = tf.keras.backend

class ExponentialLearningRate(tf.keras.callbacks.Callback):
    def __init__(self, factor):
        self.factor = factor
        self.rates = []
        self.losses = []

    def on_batch_end(self, batch, logs):
        self.rates.append(K.get_value(self.model.optimizer.learning_rate))
        self.losses.append(logs["loss"])
        K.set_value(self.model.optimizer.learning_rate, self.model.optimizer.learning_ra
```

In [22]:
```
tf.keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

In [23]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

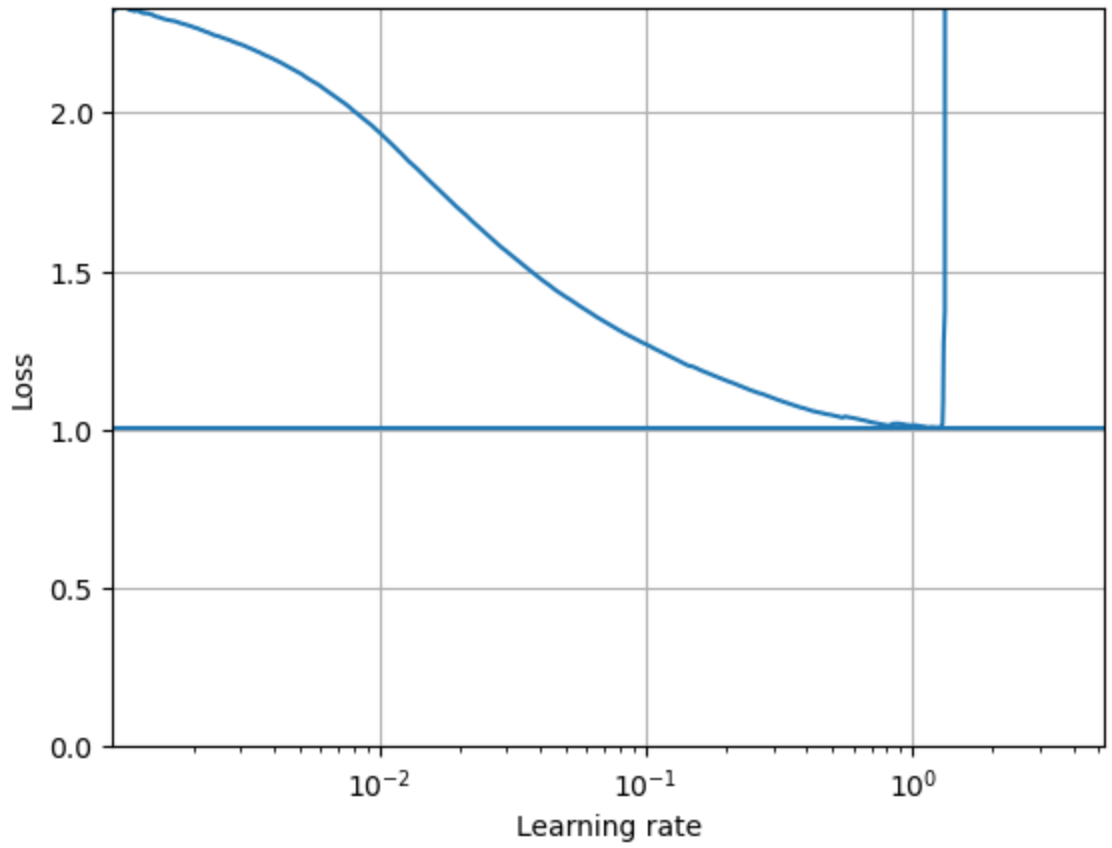
In [24]:
```
# start with a small learning rate of 1e-3, and grow it by 0.5% at each iteration

optimizer = tf.keras.optimizers.SGD(learning_rate=1e-3)

model.compile(loss="sparse_categorical_crossentropy",
              optimizer=optimizer,
              metrics=["accuracy"])

expon_lr = ExponentialLearningRate(factor=1.005)
```

In [25]:
```
# Now let's train the model for just 1 epoch

history = model.fit(X_train, y_train, epochs=1,
                    validation_data=(X_valid, y_valid),
                    callbacks=[expon_lr])
```

```
1719/1719 [==============================] - 4s 2ms/step - loss: nan - accuracy: 0.6135
- val_loss: nan - val_accuracy: 0.0958
```

In [26]:
```
# We can now plot the loss as a functionof the learning rate

plt.plot(expon_lr.rates, expon_lr.losses)

# 设置缩放x轴
plt.gca().set_xscale('log')

# 画出水平线
# plt.hlines必须要指定xmin,xmax; xmin:最小的横坐标,开始横坐标 xmax:最大的横坐标,结束横坐标
plt.hlines(min(expon_lr.losses), min(expon_lr.rates), max(expon_lr.rates))

# 标出x轴范围，y轴范围
plt.axis([min(expon_lr.rates), max(expon_lr.rates), 0, expon_lr.losses[0]])

plt.grid()
plt.xlabel("Learning rate")
plt.ylabel("Loss")
```

Out[26]:
```
Text(0, 0.5, 'Loss')
```



In [27]:
```
# the loss starts shooting back up violently when the learning rate goes over 6e-1
# let's try using half of that, at 3e-1

tf.keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

In [28]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

In [29]:
```
optimizer = tf.keras.optimizers.SGD(learning_rate=3e-1)
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=optimizer,
              metrics=["accuracy"])
```

In [32]:
```
run_index = 1 # increment this at every run
run_logdir = Path() / "my_mnist_logs" / "run_{:03d}".format(run_index)
run_logdir
```

Out[32]:
```
WindowsPath('my_mnist_logs/run_001')
```

In [33]:
```
early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=20)
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("my_mnist_model", save_best_only=True
tensorboard_cb = tf.keras.callbacks.TensorBoard(run_logdir)

history = model.fit(X_train, y_train, epochs=100,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, early_stopping_cb, tensorboard_cb])
```

```
Epoch 1/100
1693/1719 [============================>.] - ETA: 0s - loss: 0.2355 - accuracy: 0.9272IN
FO:tensorflow:Assets written to: my_mnist_model\assets
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2341 - accuracy: 0.92
76 - val_loss: 0.1050 - val_accuracy: 0.9686
Epoch 2/100
1711/1719 [============================>.] - ETA: 0s - loss: 0.0927 - accuracy: 0.9719IN
FO:tensorflow:Assets written to: my_mnist_model\assets
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0928 - accuracy: 0.97
19 - val_loss: 0.1020 - val_accuracy: 0.9740
Epoch 3/100
1696/1719 [============================>.] - ETA: 0s - loss: 0.0647 - accuracy: 0.9795IN
FO:tensorflow:Assets written to: my_mnist_model\assets
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0649 - accuracy: 0.97
95 - val_loss: 0.0755 - val_accuracy: 0.9776
Epoch 4/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0483 - accuracy: 0.98
47 - val_loss: 0.0797 - val_accuracy: 0.9772
Epoch 5/100
1715/1719 [============================>.] - ETA: 0s - loss: 0.0374 - accuracy: 0.9880IN
FO:tensorflow:Assets written to: my_mnist_model\assets
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0373 - accuracy: 0.98
80 - val_loss: 0.0755 - val_accuracy: 0.9800
Epoch 6/100
1709/1719 [============================>.] - ETA: 0s - loss: 0.0308 - accuracy: 0.9900IN
FO:tensorflow:Assets written to: my_mnist_model\assets
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0309 - accuracy: 0.99
00 - val_loss: 0.0653 - val_accuracy: 0.9818
Epoch 7/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0230 - accuracy: 0.99
21 - val_loss: 0.0868 - val_accuracy: 0.9812
Epoch 8/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0179 - accuracy: 0.99
44 - val_loss: 0.0789 - val_accuracy: 0.9820
Epoch 9/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0153 - accuracy: 0.99
52 - val_loss: 0.0824 - val_accuracy: 0.9824
Epoch 10/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0116 - accuracy: 0.99
63 - val_loss: 0.0855 - val_accuracy: 0.9800
Epoch 11/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0109 - accuracy: 0.99
65 - val_loss: 0.0909 - val_accuracy: 0.9784
Epoch 12/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0094 - accuracy: 0.99
69 - val_loss: 0.0855 - val_accuracy: 0.9822
```

```
Epoch 13/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0063 - accuracy: 0.99
79 - val_loss: 0.0909 - val_accuracy: 0.9800
Epoch 14/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0046 - accuracy: 0.99
86 - val_loss: 0.0826 - val_accuracy: 0.9830
Epoch 15/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0061 - accuracy: 0.99
81 - val_loss: 0.1046 - val_accuracy: 0.9782
Epoch 16/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0087 - accuracy: 0.99
72 - val_loss: 0.0967 - val_accuracy: 0.9812
Epoch 17/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0103 - accuracy: 0.99
65 - val_loss: 0.0907 - val_accuracy: 0.9816
Epoch 18/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0116 - accuracy: 0.99
64 - val_loss: 0.0989 - val_accuracy: 0.9824
Epoch 19/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0081 - accuracy: 0.99
77 - val_loss: 0.0873 - val_accuracy: 0.9830
Epoch 20/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0040 - accuracy: 0.99
88 - val_loss: 0.0883 - val_accuracy: 0.9848
Epoch 21/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0033 - accuracy: 0.99
92 - val_loss: 0.0837 - val_accuracy: 0.9856
Epoch 22/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0034 - accuracy: 0.99
90 - val_loss: 0.1138 - val_accuracy: 0.9812
Epoch 23/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0016 - accuracy: 0.99
97 - val_loss: 0.0868 - val_accuracy: 0.9864
Epoch 24/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0015 - accuracy: 0.99
96 - val_loss: 0.1181 - val_accuracy: 0.9814
Epoch 25/100
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0011 - accuracy: 0.99
97 - val_loss: 0.0890 - val_accuracy: 0.9864
Epoch 26/100
1719/1719 [==============================] - 3s 2ms/step - loss: 9.0939e-05 - accuracy:
1.0000 - val_loss: 0.0904 - val_accuracy: 0.9864
```

In [34]:
```
model = tf.keras.models.load_model("my_mnist_model") # rollback to best model

model.evaluate(X_test, y_test)
```

Out[34]:
```
313/313 [==============================] - 0s 1ms/step - loss: 0.0668 - accuracy: 0.9799
[0.06679337471723557, 0.9799000024795532]
```

In [38]:
```
# We got over 98% accuracy
%load_ext tensorboard

%tensorboard --logdir=./my_mnist_logs
```