My first Neural Network: Classify Hand-written number of 0/1

用 tensorflow + Keras 实现

引入库

In [ ]:
```
import numpy as np
import tensorflow as tf
import tensorrt as trt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
from autils import *

import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
tf.autograph.set_verbosity(0)
```

解释所用的一些库：

- numpy 线性代数计算库
- tensorflow Google 开发的机器学习模型库。这憨批东西整死我了，它据说 pip 上直接弄下来的暂时还不支持 CUDA 12，真难绷
- keras 用 Python 编写的高级神经网络API。用 Python 编程主要是为了快。用 Keras 写更加是为了快速验证想法。这段时间跟 Prof.Zhang 的交流更加意识到了 C++ 的重要性。以后真正的开发还是要用到 C++
- matplotlib 是一个 Visualization 的工具
- autils 是一个该课程作业已经编写好的库，可以直接调用
- logging 是一个生成日志用的库。

读入数据

读入数据是真的蛋疼。我一直觉得 Python 那个 `print` 函数它真的是有好也有坏。大部分时间都不太好用。Python 现在从一个数据文件中(可能是 csv, txt, image) 读入有很多方法。最近学习 Python 不太习惯的就是它原生的东西其实不多。这种脚本语言主要靠调用库，然而库实在是它丰富了，一时间真的不知道该选择什么。看 Kaggle 的一些教程在使用一个叫做 Panda 的库。当然这个课程作业已经帮大家编写好了读入数据的函数。

接下来要读入的是一个 20×20 的图像文件，可以视作一个矩阵。矩阵的每个元素为浮点型，表示灰阶。存储数据的时候，直接把它拆成一个模为 400 的列向量。我们的样本大小为 1000，那么最后读入进来存储的数据就是一个 400×1000 的矩阵。

$$X = \begin{pmatrix} --- (x^{(1)}) --- \\ --- (x^{(2)}) --- \\ \vdots \\ --- (x^{(m)}) --- \end{pmatrix}$$

In [ ]:
```
x_data, y_data = load_data()
```

构建神经网络

首先我们要把我们的数据构建一个输入层导入神经网络 (Input layer)

我们每一个数据都是一个固定模为 400 的列向量，其它参数任意。在这里我们选择 `tf.keras.Input()` 来构建输入层

`tf.keras.Input(shape=(400,))` 这行代码中 `shape=(400,)` 参数意义如下：400 描述了一个一维的 shape，其余参数均为 None,它是可变的。这意味着您可以在训练时批量处理不同数量的样本。

之后，我们的模型是第一层有 25 个神经元，第二层 15 个神经元，第三层 (Output layer) 是二元分类的输出层。

神经网络中参数的维度通常如下确定：

1. 权重矩阵（W）：
    - 如果一个神经网络层当前层有 `sin` 个单元（神经元），下一层有 `sout` 个单元，那么连接这两个层的权重矩阵 `W` 的维度将是 `sin x sout`。
    - 权重矩阵的每一行对应于连接当前层中一个单元到下一层中所有单元的权重。
    - 权重矩阵 `W` 编码了相邻层神经元之间的连接强度。
2. 偏置向量（b）：
    - 如果一个神经网络层下一层有 `sout` 个单元，那么该层的偏置向量 `b` 将有 `sout` 个元素。
    - 偏置向量的每个元素对应于下一层中的一个特定单元。
    - 偏置向量 `b` 允许网络在前向传播过程中为下一层的每个神经元引入额外的偏移或位移。

总之，神经网络中的权重矩阵（W）和偏置向量（b）是可学习的参数，它们的维度取决于当前层和下一层的神经元数量。权重矩阵（W）确定了连接的强度，而偏置向量（b）在网络前向传播过程中引入了神经元激活的偏差或位移。这些参数对于网络在训练过程中学习和适应数据非常重要。

In [ ]:
```
model = Sequential(
    [
        tf.keras.Input(shape=(400, )),
        tf.keras.layers.Dense(25, activation="relu"),
        tf.keras.layers.Dense(15, activation="sigmoid"),
        tf.keras.layers.Dense(1, activation="sigmoid")
    ], name = "zeroStar"
)
```

```
# model.summary()

model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(0.001),
)

model.fit(
    x_data, y_data,
    epochs = 20
)
```

```
Epoch 1/20
32/32 [==============================] - 0s 793us/step - loss: 0.5379
Epoch 2/20
32/32 [==============================] - 0s 767us/step - loss: 0.2455
Epoch 3/20
32/32 [==============================] - 0s 791us/step - loss: 0.1346
Epoch 4/20
32/32 [==============================] - 0s 772us/step - loss: 0.0946
Epoch 5/20
32/32 [==============================] - 0s 758us/step - loss: 0.0744
Epoch 6/20
32/32 [==============================] - 0s 873us/step - loss: 0.0615
Epoch 7/20
32/32 [==============================] - 0s 765us/step - loss: 0.0524
Epoch 8/20
32/32 [==============================] - 0s 1ms/step - loss: 0.0458
Epoch 9/20
32/32 [==============================] - 0s 1ms/step - loss: 0.0406
Epoch 10/20
32/32 [==============================] - 0s 965us/step - loss: 0.0365
Epoch 11/20
32/32 [==============================] - 0s 969us/step - loss: 0.0331
Epoch 12/20
32/32 [==============================] - 0s 854us/step - loss: 0.0302
Epoch 13/20
32/32 [==============================] - 0s 1ms/step - loss: 0.0278
Epoch 14/20
32/32 [==============================] - 0s 802us/step - loss: 0.0257
Epoch 15/20
32/32 [==============================] - 0s 680us/step - loss: 0.0238
Epoch 16/20
32/32 [==============================] - 0s 713us/step - loss: 0.0221
Epoch 17/20
32/32 [==============================] - 0s 639us/step - loss: 0.0201
Epoch 18/20
32/32 [==============================] - 0s 629us/step - loss: 0.0184
Epoch 19/20
32/32 [==============================] - 0s 659us/step - loss: 0.0167
Epoch 20/20
32/32 [==============================] - 0s 768us/step - loss: 0.0151
```

Out[ ]: <keras.src.callbacks.History at 0x7f453c3d6fb0>

测试数据

我们先来测试一下第一个和最后一个的图像的标签是多少

In [ ]:
```
firstResult = y_data[0]
lastResult = y_data[-1]

print(f"test result for the first is {firstResult}, for the last is {last
```

```
test result for the first is [0], for the last is [1]
```

现在来看一下神经网络跑出来的结果。

In [ ]:
```
prediction = model.predict(x_data[0].reshape(1, 400))
print(f" predicting a zero: {prediction}")
prediction = model.predict(x_data[-1].reshape(1, 400))
print(f" predicting a one: {prediction}")
```

```
1/1 [==============================] - 0s 32ms/step
 predicting a zero: [[0.01691746]]
1/1 [==============================] - 0s 13ms/step
 predicting a one: [[0.99072397]]
```

现在写一个控制输出的阈值的函数。

In [ ]:
```
def threshold(prediction):
    if prediction >= 0.5:
        return 1
    else:
        return 0
```

现在来对所有数据进行训练后的预测

In [ ]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

n, m = x_data.shape

fig, axes = plt.subplots(8, 8, figsize=(8, 8))
fig.tight_layout(pad=0.1,rect=[0, 0.03, 1, 0.92]) #[left, bottom, right,

for i, ax in enumerate(axes.flat):
    randomIndex = np.random.randint(n)
    xRandomReshaped = x_data[randomIndex].reshape((20, 20)).T
    ax.imshow(xRandomReshaped, cmap='gray')
    prediction = model.predict(x_data[randomIndex].reshape(1, 400))
    yHat = threshold(prediction)
    ax.set_title(f"{y_data[randomIndex, 0]}, {yHat}")
    ax.set_axis_off()
fig.suptitle("Label, yHat", fontsize = 16)
plt.show()
```

```
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 12ms/step
```

```
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 14ms/step
```