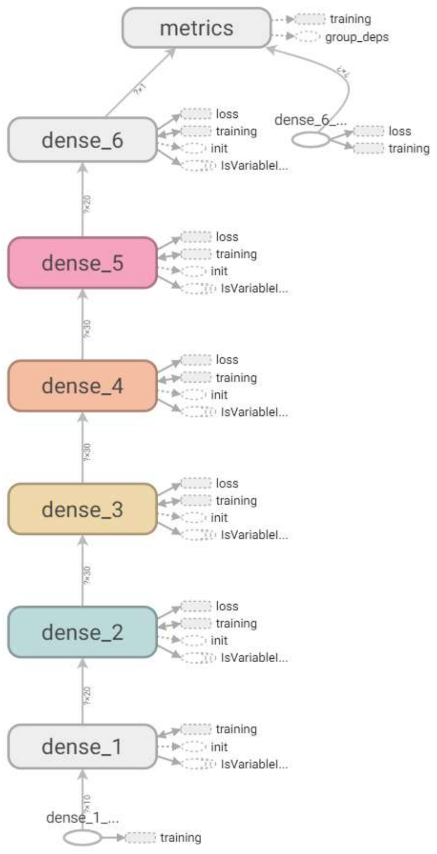
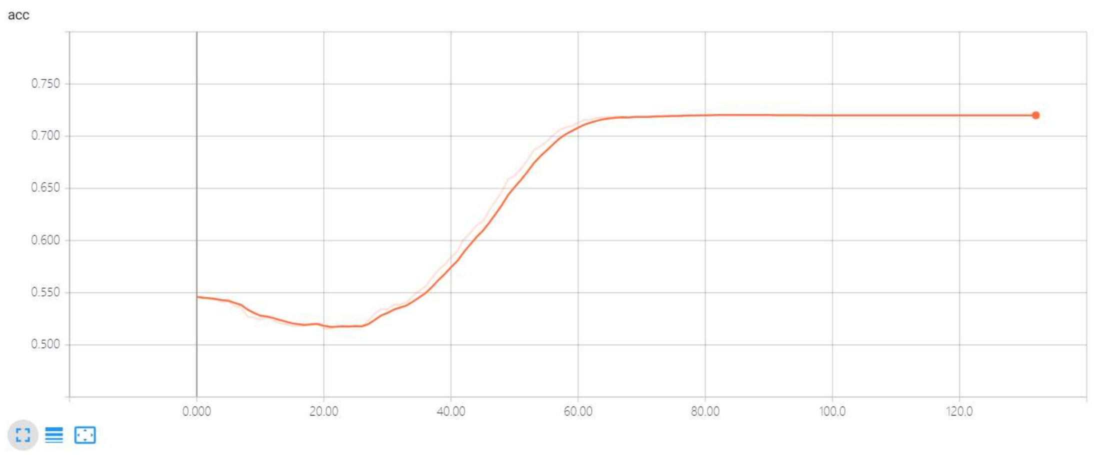
2018 年 12 月 19 日
实验三任务说明：

实验内容如下：对给定数据集进行分类，并建立相应的分类器。分析分类结果指标，改变参数并比较不同的实验结果，给出你认为的最佳模型，并说明理由。

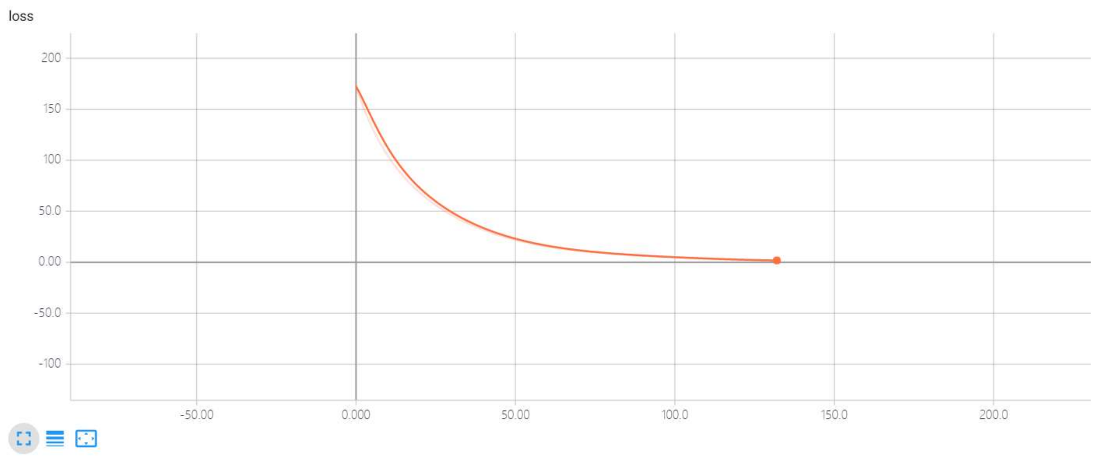
本次实验使用 keras 搭建二分类神经网络分类器.
建立了 6 层神经网络
如下图所示:



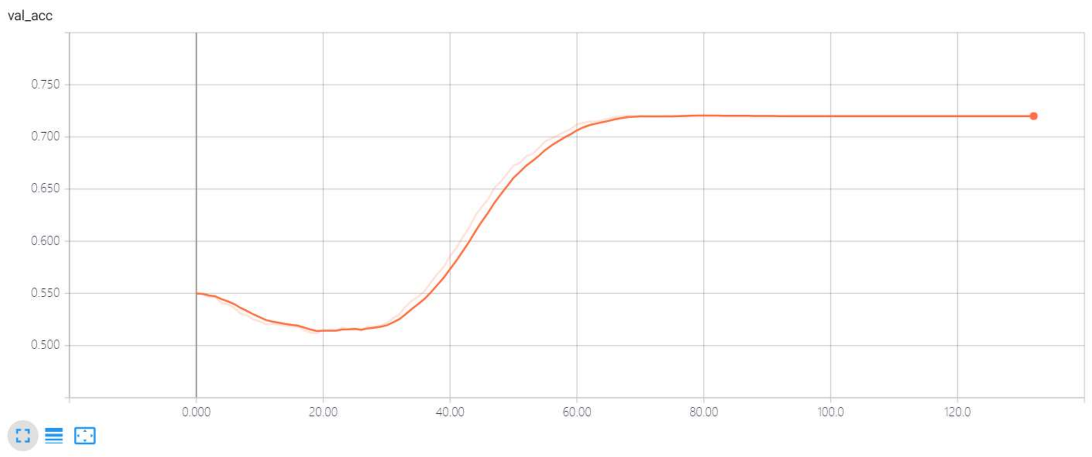
实验在训练集准确率:



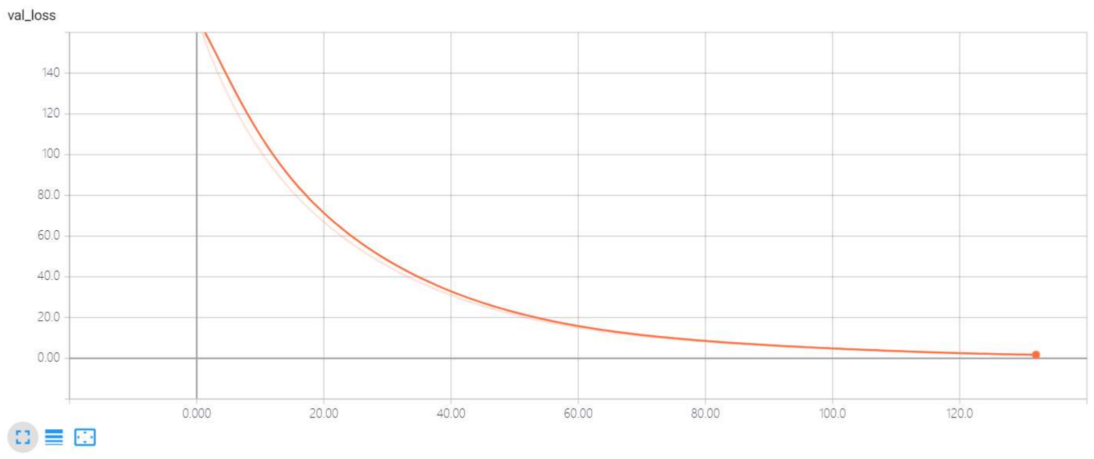
实验在训练集损失值:



实验在验证集准确率:



实验在验证集上损失值:



为了避免过拟合,我们通过观察验证集的损失函数作为我们训练的依据.我们的目标就是使损失函数最小.

我们将神经网络输出与类别进行交叉熵计算,作为损失函数,函数形式如下所示,

$$C = -\frac{1}{n}\sum_{x}[y\ln a + (1-y)\ln(1-a)]$$

其中 a 为标签,y 为神经网络输出值,如上图所示,在迭代 120 次后,验证集的损失值达到最小为 1.6. 同时在验证集的准确率为 72%并且不再发生变化.

通过改变神经网络每一层的神经元数量,以及神经网络的层数,发现层数越高准确率越高,但是存在瓶颈. 神经元的数量并不太大影响.

通过给每一层增加正则化方法提高了准确率.

为了避免过拟合,使用了 earlyStopping 方法,在验证集分类准确度下降时,终止在训练集上训练.
实验代码如下:
神经网络代码:

```
from keras import models
from keras.layers import Dense
from keras.optimizers import Adam
from keras.regularizers import l2, l1


class Model():
    def __init__(self,input_dim=10):
        self.input_dim=input_dim
        self.model=self.model()
        # Now compile the network.
        optimizer = Adam(lr=1e-5, decay=1e-6)
        metrics = ['accuracy']
        self.model.compile(loss='binary_crossentropy',
optimizer=optimizer, metrics=metrics)

    def model(self):
        model = models.Sequential()
        model.add(Dense(20, activation='relu',
input_dim=self.input_dim))
        model.add(Dense(30, activation='relu',
                      kernel_regularizer=l2(0.01),
                      activity_regularizer=l1(0.01),
                      bias_regularizer=l2(0.001)))
        model.add(Dense(30, activation='relu',
                      kernel_regularizer=l2(0.01),
                      activity_regularizer=l1(0.01),
                      bias_regularizer=l2(0.001)))
        model.add(Dense(30, activation='relu',
                      kernel_regularizer=l2(0.01),
                      activity_regularizer=l1(0.01),
                      bias_regularizer=l2(0.001)))
        model.add(Dense(20, activation='relu',
                      kernel_regularizer=l2(0.01),
                      activity_regularizer=l1(0.01),
                      bias_regularizer=l2(0.001)))
        model.add(Dense(1, activation='sigmoid'))
        return model
```

数据获取代码(在实验开始时,将数据保存为 csv 格式):

```
import csv
from time import sleep
import numpy as np
import random
import pandas as pd
import threading
class threadsafe_iterator:
    def __init__(self, iterator):
        self.iterator = iterator
        self.lock = threading.Lock()
    def __iter__(self):
        return self
    def __next__(self):
        with self.lock:
            return next(self.iterator)
def threadsafe_generator(func):
    """Decorator"""
    def gen(*a, **kw):
        return threadsafe_iterator(func(*a, **kw))
    return gen

class DataSet():
    def get_memory(self,train_test):
        data_set = pd.read_csv(f"data/{train_test}_data.csv").values# 加
载数据集
        X = data_set[:, 0:10].astype(np.int16)  # 分割为 10 个输入变量
        Y = data_set[:, 10].astype(np.int16)
        return X,Y
    @threadsafe_generator
    def generator(self, batch_size, train_test):
        data_set = pd.read_csv(f"data/{train_test}_data.csv").values# 加
载数据集
        X = data_set[:, 0:10].astype(np.int16)  # 分割为 10 个输入变量
         Y = data_set[:, 10].astype(np.int16)
        print("Creating %s generator with %d samples." %
(train_test, len(data_set)))

        while 1:
            X, y = [], []
            # Generate batch_size samples.
            for _ in range(batch_size):
                sample = random.choice(data_set)
                X.append(sample[0:10].astype(np.int16))
                y.append(sample[10].astype(np.int16))
            yield np.array(X), np.array(y)
if __name__ == '__main__':
    ds=DataSet()
    for x,y in ds.generator(32,'train'):
        print(x)
        print(y)
        sleep(5)
```

程序入口如下所示:

```
from keras.callbacks import TensorBoard, ModelCheckpoint,
EarlyStopping, CSVLogger
from models import Model
from data import DataSet
import time
import os.path
import os

def train(saved_model=None,
          load_to_memory=False,
          batch_size=32,
          nb_epoch=100):
    # Helper: Save the model.
    if not os.path.exists(os.path.join('data', 'checkpoints')):
        os.makedirs(os.path.join('data', 'checkpoints'))
    if not os.path.exists(os.path.join('data', 'logs')):
        os.makedirs(os.path.join('data', 'logs'))
    if not os.path.exists(os.path.join('data', 'checkpoints')):
        os.makedirs(os.path.join('data', 'checkpoints'))
    check_pointer = ModelCheckpoint(
        filepath=os.path.join('data', 'checkpoints','val_loss-
{val_loss:.3f}_val_acc-{val_acc:3f}.hdf5'),
        verbose=1,
        save_best_only=True)
    # Helper: TensorBoard
    tb = TensorBoard(log_dir=os.path.join('data', 'logs'))
    # Helper: Stop when we stop learning.
    early_stopper = EarlyStopping(patience=2)
    # Helper: Save results.
    timestamp = time.time()
    csv_logger = CSVLogger(os.path.join('data', 'logs','training-' +
str(timestamp) + '.log'))

    # Get the data and process it.

    data = DataSet()
    # X,Y,X_test,Y_test,generator,val_generator=None
    if load_to_memory:
        # Get data.
        X, Y = data.get_memory('train')
        X_test, Y_test = data.get_memory('test')
    else:
        # Get generators.
        generator = data.generator(batch_size, 'train')
        val_generator = data.generator(batch_size, 'test')
    # Get the model.
    model=Model(10).model
    # Fit!
    if load_to_memory:
        # Use standard fit.
        model.fit(
            X,
            Y,
            batch_size=batch_size,
            validation_data=(X_test, Y_test),
            verbose=1,
            callbacks=[tb, early_stopper, csv_logger, check_pointer],
            epochs=nb_epoch)
    else:
        # Use fit generator.
        steps_per_epoch = 3500 // batch_size
        model.fit_generator(
            generator=generator,
            steps_per_epoch=steps_per_epoch,
            verbose=1,
            callbacks=[tb, early_stopper, csv_logger, check_pointer],
            validation_data=val_generator,
            validation_steps=40,
            workers=10,
            epochs=nb_epoch)
```

```
def main():
    saved_model = None
    load_to_memory = True  # pre-load the sequences into memory
    batch_size = 32
    nb_epoch = 1000  # 一般使用 early_stopper =
EarlyStopping(patience=10)提前终止运行
    train(saved_model=saved_model,
          load_to_memory=load_to_memory,
          batch_size=batch_size,
          nb_epoch=nb_epoch)
if __name__ == '__main__':
    main()
```