神经网络与tf.keras

2.1 图片基础与tf.keras介绍

2.1.1 图像基本知识

思考：如何将图片文件转换成机器学习算法能够处理的数据？

2.1.1.1 图片三要素

高、宽度，通道数（黑白：单通道， 彩色图片三通道）

[height, width, channel]

- 单个图片：[height, width, channel]
- 多个图片：[batch,height, width, channel]，batch表示一个批次的张量数量

[200, 200, 3], [N, 200, 200, 3]

2.1.2 tf.keras介绍

Keras 遵循减少认知困难的最佳实践: 它提供一致且简单的 API

Keras 被工业界和学术界广泛采用

Keras 拥有强大的多 GPU 和分布式训练支持

2.1.3 tf.keras与keras API

2.1.4 图片读取处理

- **要使用该模块需要下载图片读取库**

```
pip install Pillow
```

图片特征值处理-图片大小

- 统一到一个大小， 神经网络模型要求图片的大小必须固定。300 * 300
- 减少训练的计算开销

每个像素点，都是0~255之间

load_img(path=filepath, target_size)

img_to_array(img, data_format=None, dtype=None)

2.1.4.2 NHWC与NCHW

设置为 "NHWC" 时，排列顺序为 [batch, height, width, channels]；

设置为 "NCHW" 时，排列顺序为 [batch, channels, height, width]。

Tensorflow默认的[height, width, channel]

- tf.reshape不能对于一维中按照RGB排列方式做channel_last变换，智能做channel_first转换
  - 否则转换结果错误
- 怎么做：
  - 1、首先reshape:[channel, height, width]
  - 2、进行tf.transpose(depth_major, [1, 2, 0]).eval()，维度位置的替换,[height, width, channel]

2.1.5 tf.keras 数据集

2.2 神经网络基础

2.2.1 神经网络

- 组成：
- **输入层，输出层以及隐藏层**
- 每层的圆圈代表神经元
- 神经网络的特点
  - 每个连接都有个权值
  - 同一层神经元之间没有连接
  - 最后的输出结果对应的层也称之为**全连接层**

2.2.1.1 感知机(PLA: Perceptron Learning Algorithm))

2.2.2 playground使用

但是这种结构的线性的二分类器，但不能对非线性的数据并不能进行有效的分类

- 能够很好去解决与、或等问题，但是并不能很好的解决异或等问题
- 怎么取解决
  - **其实我们多增加几个感知机即可解决**

tf.kerasSequential构建模型使用

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
])
```

构建模型推荐使用tf.keras.Sequential

1.4 神经网络原理

1.4.1 softmax回归

- 假设输出结果为：2.3, 4.1, 5.6 softmax的计算输出结果为： y1_p = e^2.3/(e^2.3+e^4.1+e^5.6) y1_p = e^4.1/(e^2.3+e^4.1+e^5.6) y1_p = e^5.6/(e^2.3+e^4.1+e^5.6)
- 得到概率相加为1

1.4.2 交叉熵损失

- **神经网络预测的概率分布和真实答案的概率分布之间的距离**
- y_i':真实值， y_i代表预测概率值，

```
-
(0log(0.10)+0log(0.05)+0log(0.15)+0log(0.10)+0log(0.05)+0log(0.20)+1log(
0.10)+0log(0.05)+0log(0.10)+0log(0.10))=- 1log(0.10)
```

- 要想这个损失越小，那么对应全连接层神经元位置输出概率越来越大

1.4.4 网络原理总结

1.3 Tensorflow实现神经网络

1.3.1 tf.keras构建模型步骤API介绍

- 获取数据集
- keras.layers：构建网络的每一层

```
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(64, activation=tf.nn.relu),
    Dense(128, activation=tf.nn.relu),
    Dense(10, activation=tf.nn.softmax)
])
```

- 优化算法：from tensorflow.python.keras.optimizers import SGD

```
from tensorflow.python.keras.losses import binary_crossentropy
```

  - from tensorflow.python.keras.losses import categorical_crossentropy
  - from tensorflow.python.keras.losses import sparse_categorical_crossentropy
- model.compile()
  - 优化器，损失计算，准确率
  - sparse_categorical_crossentropy:对于目标值是整型的进行交叉熵损失计算
  - categorical_crossentropy:对于两个output tensor and a target tensor进行交叉熵损失计算
- model.fit(x=None,y=None, batch_size=None,epochs=1,callbacks=None)
  - epochs=1：训练迭代次数

2.4.2 案例：实现多层神经网络进行时装分类

- 案例步骤：
  - 读取数据集
  - datasets
  - 建立神经网络模型
    - model = keras.Sequential([keras.layers.Flatten(input_shape=(28, 28)), keras.layers.Dense(128, activation=tf.nn.relu), keras.layers.Dense(10, activation=tf.nn.softmax)])
  - 编译模型优化器、损失、准确率
    - SingleNN.model.compile(optimizer=keras.optimizers.SGD(lr=0.01), loss=tf.keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
  - 进行fit训练
    - SingleNN.model.fit(self.train, self.train_label, epochs=5, batch_size=32)
  - 评估模型测试效果
    - SingleNN.model.evaluate(self.test, self.test_label)
- 参数：关于迭代次数与每次训练样本数
  - 在每迭代一次训练时，60000，每次选择batch_size=32个样本训练，
  - 在第二次迭代次训练时，60000，每次选择batch_size=32个样本训练，
  - 在第二次迭代次训练时

2.4.2.4 手动保存和恢复模型

- 保存成ckpt形式
  - SingleNN.model.load_weights("./ckpt/SingleNN")
- 保存成h5文件
  - SingleNN.model.save_weights("./ckpt/SingleNN.h5")

2.4.3 fit的callbacks详解

- 定制化保存模型
    - ModelCheckpoint('./ckpt/singlenn_{epoch:02d}-{val_acc:.2f}.h5', monitor='val_acc', save_best_only=True, save_weights_only=True, mode='auto', period=1)
    - fit_generator()
- 保存events文件
    - 添加tensoboard观察

  tensorboard = keras.callbacks.TensorBoard(log_dir='./graph', histogram_freq=1, write_graph=True, write_images=True)

2.5 深层神经网络与优化算法

随着神经网络的深度加深，模型能学习到更加复杂的问题，功能也更加强大

2.5.1 深层神经网络表示

2.5.2 四层网络的前向传播与反向传播

理解过程

2.5.3 参数与超参数

- **参数**即是我们在过程中想要模型学习到的信息（**模型自己能计算出来的**），例如 W[l]W[l]，b[l]b[l]

2.5.4 优化遇到的问题

2.5.4.1 梯度消失

现象：计算梯度接过指数爆炸， -inf

2.5.4.2 局部最优

鞍点（saddle）**是函数上的导数为零，但不是轴上局部极值的点。通常梯度为零的点是上图所示的鞍点，而非局部最小值。减少损失的难度也来自误差曲面中的鞍点，而不是局部最低点

- 原因：鞍点问题

如何解决上述问题

- 初始化参数策略（第一部分第四节提到）
- Mini梯度下降法
- 梯度下降算法的优化
- 学习率衰减

2.2.2 参数初始化策略

- 随机初始化权重
- w和b参数的初始化，以标准正太分布去初始化

2.2.3 批梯度下降算法（Batch Gradient Descent）

- **定义：批梯度下降法(btach)，即同时处理整个训练集。
- epoch:1, 60000一起输入计算训练，batch_size=所有训练样本数

2.2.3.1 Mini-Batch Gradient Descent

- **定义：Mini-Batch 梯度下降法（小批量梯度下降法）每次同时处理固定大小的数据集。**
- mini-batch 的大小为 1，即是**随机梯度下降法（stochastic gradient descent）**

batch 梯度下降法：不采用，随机梯度下降也不采用

Mini-batch 梯度下降

Minit-bath？个数？取值=2^m，根据计算机二进制

2.2.4 指数加权平均

可以理解成梯度下降的过程，加上移动平均计算，达到梯度平滑的过程

2.2.5 动量梯度下降法 Momentum

- 鞍点附近：小球在向下运动过程中，导致越来越快，由于β的存在使得不会一直加速运行。
- 鞍点附近：由于增加了一个动量，在C、D方向上能够有一个动量冲过。
- 给定一个冲量，冲过鞍点位置

2.2.6 RMSProp 算法

最终RMSProp 有助于减少抵达最小值路径上的摆动，并允许使用一个更大的学习率 α，从而加快算法学习速

2.2.7 Adam算法

Adam 优化算法（Adaptive Moment Estimation，自适应矩估计）将 Momentum 和 RMSProp 算法结合在一起。

- beta1, beta2

2.2.9 学习率衰减

- 算法中自带：指数衰减学习率

2.2.10 其它非算法优化的方式-标准化输入

2.2.10 标准化输入

- 损失函数变得碗状结构

两种：算法层面，数据输入初始化角度