《深度学习构架》实验一 几种深度学习框架的安装及编译

一、实验目的

1. 掌握TensorFlow和PyTorch的安装与编译方法。
2. 掌握TensorFlow的基本语法和数据类型；
3. 以CSV文件为例，掌握TensorFlow的数据读取方法；
4. 掌握保存和加载TensorFlow模型的方法。

二、实验环境

1. 笔记本电脑，Windows,Linux或MacOs系统；
2. Python3.x.
3. 实验数据：pima-indians-diabetes.csv，mnist.npz

三、实验内容

1. 安装TensorFlow 2.x（或PyTorch）；
2. 安装numpy,Matplotlib等Python模块；
3. 打开Python，完成TensorFlow基本语法及数据类型的练习操作；
4. 下载pima-indians-diabetes.csv文件，读取数据并对数据进行初步了解；
5. 用TensorFlow的Keras模型完成对数据的分析；
6. 保存模型为静态文件，打开模型文件并对测试数据进行评估。

四、主要实验步骤

1. 安装TensorFlow 2.x

使用国内镜像安装TensorFlow 2.x

```
$pip install –i https://pypi.tuna.tsinghua.edu.cn/simple
tensorflow
```

2. 安装numpy,Matplotlib等Python模块

过程略。

3. 熟悉下列TensorFlow常用基本函数

加载tensorflow模块：

import tensorflow as tf

下列部分函数在TensorFlow2.0中已经被移到tensorflow.compat.v1下了，下列import代码可方便调用这些函数。

import tensorflow.compat.v1 as tf1

1）数据类型转换

- tf1.string_to_number(string, out_type=None, name=None)
  字符串转为数字
- tf1.to_double(x, name='ToDouble')
  转为64位浮点类型–float64
- tf1.to_float(x, name='ToFloat')
  转为32位浮点类型–float32
- tf1.to_int32(x, name='ToInt32')
  转为32位整型–int32
- tf1.to_int64(x, name='ToInt64')
  转为64位整型–int64
- tf.cast(x, dtype, name=None) 将x或者x.values转换为dtype

In [3]:
```
import tensorflow as tf
import tensorflow.compat.v1 as tf1

string = '12.3'
n1 = tf1.string_to_number(string, out_type=None, name=None)
print(n1)

x = 12.3
d1 = tf1.to_double(x, name='ToDouble')
print(d1)

f1 = tf1.to_float(x, name='ToFloat')
print(f1)

i1 = tf1.to_int32(x, name='ToInt32')
print(i1)

i2 = tf1.to_int64(x, name='ToInt64')
print(i2)

a = [1.8, 2.2]
i3 = tf.cast(a, tf.int32)
print(i3)
```

```
tf.Tensor(12.3, shape=(), dtype=float32)
WARNING:tensorflow:From <ipython-input-3-529769939f03>:9: to_double (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Use `tf.cast` instead.
tf.Tensor(12.300000190734863, shape=(), dtype=float64)
WARNING:tensorflow:From <ipython-input-3-529769939f03>:12: to_float (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Use `tf.cast` instead.
tf.Tensor(12.3, shape=(), dtype=float32)
WARNING:tensorflow:From <ipython-input-3-529769939f03>:15: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Use `tf.cast` instead.
tf.Tensor(12, shape=(), dtype=int32)
WARNING:tensorflow:From <ipython-input-3-529769939f03>:18: to_int64 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Use `tf.cast` instead.
tf.Tensor(12, shape=(), dtype=int64)
tf.Tensor([1 2], shape=(2,), dtype=int32)
```

- tf.shape(input, name=None) 返回数据的shape

In [7]:
```
t = [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]
tf.shape(t)
```

Out[7]:
```
<tf.Tensor: id=30, shape=(3,), dtype=int32, numpy=array([2, 2, 3])>
```

- tf.size(input, name=None) 返回数据的元素数量

In [9]:
```
t = [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]
tf.size(t)
```

Out[9]:
```
<tf.Tensor: id=32, shape=(), dtype=int32, numpy=12>
```

- tf.rank(input, name=None) 返回tensor的rank 注意：tensor的rank表示一个tensor需要的索引数目来唯一表示任何一个元素 也就是通常所说的 “order”, “degree”或"ndims"

In [14]:
```
t = [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]
tf.rank(t)
```

Out[14]:
```
<tf.Tensor: id=40, shape=(), dtype=int32, numpy=3>
```

- tf.reshape(tensor, shape, name=None) 改变tensor的形状

```
tensor ‘t’ is [1, 2, 3, 4, 5, 6, 7, 8, 9]
tensor ‘t’ has shape [9]
reshape(t, [3, 3]) ==> [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
如果shape有元素[-1],表示自动推导该维度值，-1 将自动推导得为 3
reshape(t, [3, -1]) ==> [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
```

In [20]:
```
t = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(tf.shape(t))
t2 = tf.reshape(t, [3, 3])
print(tf.shape(t2))
t3 = tf.reshape(t, [3, -1])
print(t3)
```

```
tf.Tensor([9], shape=(1,), dtype=int32)
tf.Tensor([3 3], shape=(2,), dtype=int32)
tf.Tensor(
[[1 2 3]
 [4 5 6]
 [7 8 9]], shape=(3, 3), dtype=int32)
```

- tf.expand_dims(input, dim, name=None) 插入维度1进入一个tensor中

```
't' is a tensor of shape [2]
shape(expand_dims(t, 0)) ==> [1, 2]
shape(expand_dims(t, 1)) ==> [2, 1]
shape(expand_dims(t, -1)) ==> [2, 1]
't2' is a tensor of shape [2, 3, 5]
shape(expand_dims(t2, 0)) ==> [1, 2, 3, 5]
shape(expand_dims(t2, 2)) ==> [2, 3, 1, 5]
shape(expand_dims(t2, 3)) ==> [2, 3, 5, 1]
```

In [26]:
```
t = [2,3]
t1 = tf.shape(tf.expand_dims(t, 0))
t2 = tf.shape(tf.expand_dims(t, 1))
t3 = tf.shape(tf.expand_dims(t, -1))
print(t1)
print(t2)
print(t3)
```

```
tf.Tensor([1 2], shape=(2,), dtype=int32)
tf.Tensor([2 1], shape=(2,), dtype=int32)
tf.Tensor([2 1], shape=(2,), dtype=int32)
```

In [27]:
```
t4 = tf.ones([2,3,5])
t5 = tf.shape(tf.expand_dims(t4, 0))
t6 = tf.shape(tf.expand_dims(t4, 2))
t7 = tf.shape(tf.expand_dims(t4, 3))

print(t4)
print(t5)
print(t6)
print(t7)
```

```
tf.Tensor(
[[[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]

 [[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]], shape=(2, 3, 5), dtype=float32)
tf.Tensor([1 2 3 5], shape=(4,), dtype=int32)
tf.Tensor([2 3 1 5], shape=(4,), dtype=int32)
tf.Tensor([2 3 5 1], shape=(4,), dtype=int32)
```

- tf.slice(input_, begin, size, name=None) 对tensor进行切片操作

其中size[i] = input.dim_size(i) - begin[i]

该操作要求 0 <= begin[i] <= begin[i] + size[i] <= Di for i in [0, n]

In [28]:
```
t = [[[1, 1, 1], [2, 2, 2]],[[3, 3, 3], [4, 4, 4]],[[5, 5, 5], [6, 6, 6]]]
t1 = tf.slice(t, [1, 0, 0], [1, 1, 3])
t2 = tf.slice(t, [1, 0, 0], [1, 2, 3])
t3 = tf.slice(t, [1, 0, 0], [2, 1, 3])

print(t1)
print(t2)
print(t3)
```

```
tf.Tensor([[[3 3 3]]], shape=(1, 1, 3), dtype=int32)
tf.Tensor(
[[[3 3 3]
  [4 4 4]]], shape=(1, 2, 3), dtype=int32)
tf.Tensor(
[[[3 3 3]]

 [[5 5 5]]], shape=(2, 1, 3), dtype=int32)
```

- tf.split(split_dim, num_split, value, name='split') 沿着某一维度将tensor分离为num_split tensors

In [32]:
```
t = tf.ones([5,30])
t1, t2, t3 = tf.split(t,3,1)
print(tf.shape(t1))
print(tf.shape(t2))
print(tf.shape(t3))
```

```
tf.Tensor([ 5 10], shape=(2,), dtype=int32)
tf.Tensor([ 5 10], shape=(2,), dtype=int32)
tf.Tensor([ 5 10], shape=(2,), dtype=int32)
```

- tf.concat(concat_dim, values, name='concat') 沿着某一维度连结tensor

In [36]:
```
t1 = [[1, 2, 3], [4, 5, 6]]
t2 = [[7, 8, 9], [10, 11, 12]]
t3 = tf.concat([t1, t2], 0)
t4 = tf.concat([t1, t2], 1)
print(t3)
print(t4)
```

```
tf.Tensor(
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]], shape=(4, 3), dtype=int32)
tf.Tensor(
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]], shape=(2, 6), dtype=int32)
```

In [35]:
```
t1 = [[[1, 2], [2, 3]], [[4, 4], [5, 3]]]
t2 = [[[7, 4], [8, 4]], [[2, 10], [15, 11]]]
tf.concat([t1, t2], -1)
```

Out[35]:
```
<tf.Tensor: id=196, shape=(2, 2, 4), dtype=int32, numpy=
array([[[ 1,  2,  7,  4],
        [ 2,  3,  8,  4]],

       [[ 4,  4,  2, 10],
        [ 5,  3, 15, 11]]])>
```

- stack(values, axis=0, name='stack') Stacks a list of rank-`R` tensors into one rank-`(R+1)` tensor.

  ‘x’ is [1, 4], ‘y’ is [2, 5], ‘z’ is [3, 6]
  tf.stack([x, y, z]) => [[1, 4], [2, 5], [3, 6]]
  沿着第一维pack
  tf.stack([x, y, z], axis=1) => [[1, 2, 3], [4, 5, 6]]
  等价于tf.stack([x, y, z]) = np.asarray([x, y, z])

In [7]:
```
x = [1, 4]
y = [2, 5]
z = [3, 6]
t1 =tf.stack([x, y, z])
print(t1)
#沿着第一维stack
t2 = tf.stack([x, y, z], axis=1)
print(t2)

#等价于
t3 = tf.stack([x, y, z])
print(t3)
import numpy as np
t4 = np.asarray([x, y, z])
print(t4)
```

```
tf.Tensor(
[[1 4]
 [2 5]
 [3 6]], shape=(3, 2), dtype=int32)
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[1 4]
 [2 5]
 [3 6]], shape=(3, 2), dtype=int32)
[[1 4]
 [2 5]
 [3 6]]
```

- tf.reverse(tensor, dims, name=None) 沿着某维度进行序列反转

其中dim为列表，元素为bool型，size等于rank(tensor)

In [40]:
```
t = [[[[ 0,  1,  2,  3],
[ 4,  5,  6,  7],
[ 8,  9, 10, 11]],
[[12, 13, 14, 15],
[16, 17, 18, 19],
[20, 21, 22, 23]]]]
# tensor 't' shape is [1, 2, 3, 4]

dims = [3]# or 'dims = [-1]''
t1 = tf.reverse(t, dims)
print(t1)

dims = [1] #or 'dims = [-3]'
t2 = tf.reverse(t, dims)

dims = [2] #or 'dims = [-2]'
t3 = tf.reverse(t, dims)
print(t1)
print(t2)
print(t3)
```

```
tf.Tensor(
[[[[ 3  2  1  0]
   [ 7  6  5  4]
   [11 10  9  8]]

  [[15 14 13 12]
   [19 18 17 16]
   [23 22 21 20]]]], shape=(1, 2, 3, 4), dtype=int32)
tf.Tensor(
[[[[ 3  2  1  0]
   [ 7  6  5  4]
   [11 10  9  8]]

  [[15 14 13 12]
   [19 18 17 16]
   [23 22 21 20]]]], shape=(1, 2, 3, 4), dtype=int32)
tf.Tensor(
[[[[12 13 14 15]
   [16 17 18 19]
   [20 21 22 23]]

  [[ 0  1  2  3]
   [ 4  5  6  7]
   [ 8  9 10 11]]]], shape=(1, 2, 3, 4), dtype=int32)
tf.Tensor(
[[[[ 8  9 10 11]
   [ 4  5  6  7]
   [ 0  1  2  3]]

  [[20 21 22 23]
   [16 17 18 19]
   [12 13 14 15]]]], shape=(1, 2, 3, 4), dtype=int32)
```

- tf.transpose(a, perm=None, name='transpose') 调换tensor的维度顺序 按照列表perm的维度排列调换tensor顺序，如未定义，则perm为(n-1...0)

In [42]:
```
t = [[1, 2, 3],[4, 5, 6]]
t1 = tf.transpose(t)
t2 = tf.transpose(t, perm=[1, 0])
print(t1)
print(t2)
```

```
tf.Tensor(
[[1 4]
 [2 5]
 [3 6]], shape=(3, 2), dtype=int32)
tf.Tensor(
[[1 4]
 [2 5]
 [3 6]], shape=(3, 2), dtype=int32)
```

- tf.gather(params, indices, validate_indices=None, name=None) 合并索引indices所指示params中的切片

- tf.one_hot(indices,depth,on_value=None,off_value=None,axis=None,dtype=None,na

In [43]:
```
indices = [0, 1, 2]
depth = 3
t1 = tf.one_hot(indices, depth)


indices = [0, 2, -1, 1]
depth = 3
t2 = tf.one_hot(indices, depth,on_value=5.0, off_value=0.0,axis=-1)

indices = [[0, 2], [1, -1]]
depth = 3
t3 = tf.one_hot(indices, depth,on_value=1.0, off_value=0.0,axis=-1)

print(t1)
print(t2)
print(t3)
```

```
tf.Tensor(
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]], shape=(3, 3), dtype=float32)
tf.Tensor(
[[5. 0. 0.]
 [0. 0. 5.]
 [0. 0. 0.]
 [0. 5. 0.]], shape=(4, 3), dtype=float32)
tf.Tensor(
[[[1. 0. 0.]
  [0. 0. 1.]]

 [[0. 1. 0.]
  [0. 0. 0.]]], shape=(2, 2, 3), dtype=float32)
```

- tf.unique(x, out_idx=tf.dtypes.int32,name=None)

In [47]:
```
t = [1, 1, 2, 4, 4, 4, 7, 8, 8]
y, idx = tf.unique(t)
print(y)
print(idx)
```

```
tf.Tensor([1 2 4 7 8], shape=(5,), dtype=int32)
tf.Tensor([0 0 1 2 2 2 3 4 4], shape=(9,), dtype=int32)
```

- tf.zeros(shape,dtype=tf.dtypes.float32,name=None)
- tf.diag(diagonal, name=None)
- tf.trace(x, name=None)
- tf.matrix_determinant(input, name=None)
- tf.matrix_inverse(input, adjoint=None, name=None)
- tf.transpose(a, perm=None, name='transpose')

In [49]:
```
t = [[1, 2, 3],[4, 5, 6]]
t1 = tf.transpose(t)
t2 = tf.transpose(t, perm=[1, 0])
print(t1)
print(t2)
```

```
tf.Tensor(
[[1 4]
 [2 5]
 [3 6]], shape=(3, 2), dtype=int32)
tf.Tensor(
[[1 4]
 [2 5]
 [3 6]], shape=(3, 2), dtype=int32)
```

2）矩阵操作

- tf.matmul(a,b,
  transpose_a=False,transpose_b=False,a_is_sparse=False,b_is_sparse=False,name=
  矩阵相乘
- tf.complex(real, imag, name=None) 将两实数转换为复数形式
- tf.complex_abs(x, name=None) 计算复数的绝对值，即长度。
- tf.conj(input, name=None) 计算共轭复数
- tf.imag(input, name=None)
- tf.real(input, name=None)
- tf.fft(input, name=None)
- tf.eye(num_rows,num_columns=None,batch_shape=None,dtype=tf.dtypes.float32,n
- tf.fill(dims,value,name=None)
- tf.ones(shape,dtype=tf.dtypes.float32,name=None)

3）生成随机张量

- tf.random_normal()、
- tf.truncated_normal() 产生截断正态分布随机数，取值范围为 [mean - 2 *stddev, mean + 2* stddev]
- tf.random_uniform()
- tf.random_shuffle() 随机地将张量沿其第一维度打乱

4. 练习1

1）读取数据并查看数据结构 下载pima-indians-diabetes.csv文件并在python中加载。

```
import tensorflow as tf
import numpy

# 加载数据
dataset = numpy.loadtxt("pima-indians-diabetes.csv",
delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
```

2）设计神经网络并训练模型。

```
# step 1. define the network
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(12, input_dim=8,
activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
# step 2. compile the network
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])
# step 3. fit the network
history = model.fit(X, Y, epochs=100, batch_size=10)
# step 4. evaluate the network
loss, accuracy = model.evaluate(X, Y)
print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy*100))
# step 5. make predictions
probabilities = model.predict(X)
predictions = [float(numpy.round(x)) for x in probabilities]
accuracy = numpy.mean(predictions == Y)
print("Prediction Accuracy: %.2f%%" % (accuracy*100))
```

3）保存和加载模型 TensorFlow有两种保存模型的方法，一种是只保存模型的权重和偏置，另一种是保存整个模型。基本用法如下：

- 只保存模型的权重和偏置

  ```
  # step 6 保存模型的权重和偏置
  model.save_weights('./save_weights/my_save_weights')
  # step 7 恢复权重
  model.load_weights('./save_weights/my_save_weights')
  ```

- 保存整个模型

  ```
  model.save('my_model.h5')
  restored_model = tf.keras.models.load_model('my_model.h5')
  ```

5. 练习2

完成MNIST数据读取、模型搭建、训练、测试等过程，并尝试修改模型以提高测试效果。

```
import tensorflow as tf
import numpy as np

def load_data(path):
    with np.load(path) as f:
        x_train, y_train = f['x_train'], f['y_train']
        x_test, y_test = f['x_test'], f['y_test']
        return (x_train, y_train), (x_test, y_test)


((trainX, trainY), (testX, testY)) = load_data("mnist.npz")
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0

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, activation='softmax')
])

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

model.fit(trainX, trainY, epochs=5)

model.evaluate(testX,  testY, verbose=2)
```

五、实验要求

1. 按顺序完成上述操作要求，掌握TensorFlow的基本用法；
2. 记录实验过程及结果；
3. 完成实验报告，双面打印。
4. 每个人独立完成实验报告，内容如有雷同则所有雷同报告将都会被判为不及格。