# MOOC北大人工智能实践：tensorflow2

地址:

MOOC官方

B站视频

Created by: Henry Huang

---

目录



# 0.函数总结

(numpy参考文档)[https://numpy.org/doc/stable/index.html]

## (1)tf.where语句

```
a=tf.constant([1,2,3,1,1])
b=tf.constant([0,1,3,4,5])
c=tf.where(tf.greater(a,b), a, b) # 若a>b，返回a对应位置的元素，否则
返回b对应位置的元素
print("c:",c)
运行结果：
c:  tf.Tensor([1 2 3 4 5], shape=(5,), dtype=int32)
```

## (2)np.random.RandomState.rand()随机数产生函数

用法np.random.RandomState.rand(维度)

返回一个[0,1)之间的随机数

```
import numpy as np
rdm=np.random.RandomState(seed=1) #seed=常数每次生成随机数相同
a=rdm.rand() # 返回一个随机标量
b=rdm.rand(2,3) # 返回维度为2行3列随机数矩阵
print("a:",a)
print("b:",b)
运行结果：
a: 0.417022004702574
b: [[7.20324493e-01 1.14374817e-04 3.02332573e-01]
[1.46755891e-01 9.23385948e-02 1.86260211e-01]]
```

# (3)np.vstack()数组垂直叠加函数

用法：np.vstack(数组1，数组2)

作用:将两个数组按垂直方向叠加

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])
c = np.vstack((a,b))
print("c:\n",c)
运行结果：
c:
[[1 2 3]
[4 5 6]]
```

# (3)np.vstack()数组垂直叠加函数

用法：np.vstack(数组1，数组2)

作用:将两个数组按垂直方向叠加

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])
c = np.vstack((a,b))
print("c:\n",c)
运行结果：
c:
[[1 2 3]
[4 5 6]]
```

# (4)np.mgrid[ ]产生规定间隔的数

用法：np.mgrid[ 起始值 : 结束值 : 步长 ， 起始值 : 结束值 : 步长 , ... ]

作用:产生规定间隔的数，如果步长是虚数如5j则包含结束值。如果步长是实数则不包含结束值。

从运行结果可以看出，如果是两组参数，那么第一组参数是往下间隔的，第二组是往右间隔的。如果是三组参数，第一组是维度按间隔的，第二组是往下间隔的，第三组是往右间隔的。如果是四组等等，除了倒数第二组是往下间隔，倒数第一组是往右间隔，其他都是维度方向间隔。

```
import numpy as np
x= np.mgrid [1:3:1,1:3:1,1:3:1]
print("x:",x)
运行结果：
x: [[[[1 1]
   [1 1]]
  [[2 2]
   [2 2]]]

 [[[1 1]
   [2 2]]
  [[1 1]
   [2 2]]]

 [[[1 2]
   [1 2]]
  [[1 2]
   [1 2]]]]
```

# (5)np对象.ravel( ) 将对象本身变为一维数组

用法：x.rave(a,order='C')

作用:将对象本身变成一维数组。默认参数是C。

C是参数，表示以行为整体的变形，代表将二维数组变成一维时，第二行接在第一行后面，第三行接在第二行后面。

也可以是F，表示以列为主的变形。

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print(np.ravel(x,order='F'))
print(np.ravel(x,order='C'))
运行结果：
[1 4 2 5 3 6]
[1 2 3 4 5 6]
```

# (6)np.c[]和np.r_[] 矩阵的拼接操作

c是column r是row，因此

np.c[]理解为两个矩阵对应行的拼接，即延长矩阵的列，拼接后矩阵的行数与原来的相同

np.r[]理解为两个矩阵对应列的拼接，即延长矩阵的行，拼接后矩阵的列数与原来的相同

用法：np.vstack(数组1，数组2)

作用:将两个数组按垂直方向叠加

```
import numpy as np
x,y = np.mgrid[0:2:1,4:6:1]
print(x)
```

```
print(y)
print(np.c_[x,y])
print(np.r_[x,y])
运行结果：
[[0 0]
 [1 1]]
[[4 5]
 [4 5]]
[[0 0 4 5]
 [1 1 4 5]]
[[0 0]
 [1 1]
 [4 5]
 [4 5]]
```

# (7)tf.losses.categorical_crossentropy()计算交叉熵

用法：tf.losses.categorical_crossentropy(概率分布矩阵A,概率分布矩阵B)

作用:计算两个概率分布的交叉熵

```
loss_ce1=tf.losses.categorical_crossentropy([1,0],[0.6,0.4])
loss_ce2=tf.losses.categorical_crossentropy([1,0],[0.8,0.2])
print("loss_ce1:", loss_ce1)
print("loss_ce2:", loss_ce2)

运行结果：
loss_ce1: tf.Tensor(0.5108256, shape=(), dtype=float32)
loss_ce2: tf.Tensor(0.22314353, shape=(), dtype=float32)
```

# (8)tf.nn.softmax_cross_entropy_with_logits()直接softmax+交叉熵

用法：tf.nn.softmax_cross_entropy_with_logits(y_，y)

作用:y_是输出值,y是真实值 我们直接将两个都变成概率分布，然后计算他们的交叉熵

```
y_ = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 0], [0, 1, 0]])
y = np.array([[12, 3, 2], [3, 10, 1], [1, 2, 5], [4, 6.5, 1.2], [3, 6, 1]])
y_pro = tf.nn.softmax(y)
loss_ce1 = tf.losses.categorical_crossentropy(y_, y_pro)
loss_ce2 = tf.nn.softmax_cross_entropy_with_logits(y_, y)
print('分步计算的结果:\n', loss_ce1)
print('结合计算的结果:\n', loss_ce2)

运行结果：
分步计算的结果：
tf.Tensor(
[1.68795487e-04 1.03475622e-03 6.58839038e-02 2.58349207e+00
5.49852354e-02], shape=(5,), dtype=float64)
结合计算的结果：
tf.Tensor(
[1.68795487e-04 1.03475622e-03 6.58839038e-02 2.58349207e+00
5.49852354e-02], shape=(5,), dtype=float64)
```

## (3)np.vstack()数组垂直叠加函数

用法：np.vstack(数组1，数组2)

作用:将两个数组按垂直方向叠加

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])
c = np.vstack((a,b))
print("c:\n",c)
运行结果：
c:
[[1 2 3]
[4 5 6]]
```

# 1.神经网络的复杂度

- 空间复杂度：
  - 层数 = 隐藏层的层数 + 1个输出层（空间复杂度只统计具有运算能力的层，而输入层不具有运算能力）
  - 总参数 = 总w + 总b
- 时间复杂度：
  -进行乘加运算的次数（神经网络图中，一个权重线就代表了一次乘加运算）

# 2 学习率的设置

可以有指数衰减学习率和分段常数衰减学习率。

TensorFlow API:见老师给出的API

指数学习率=$初始学习率 * 学习率衰减率^{(当前轮数/多少轮衰减一次)}$

其中的超参数为：初始学习率、学习率衰减率、多少轮衰减一次

而当前轮数为变量

指数衰减学习率是先使用较大的学习率来快速得到一个较优的解，然后随着迭代的继续,逐步减小学习率，使得模型在训练后期 更加稳定。指数型学习率衰减法是最常用的衰减方法，在大量模型中都广泛使用。

# 3.激活函数

**优秀的激活函数应满足：**

- 非线性： 激活函数非线性时，多层神经网络可逼近所有函数
- 可微性： 优化器大多用梯度下降更新参数
- 单调性： 当激活函数是单调的，能保证单层网络的损失函数是凸函数
- 近似恒等性： . 当参数初始化为随机小值时，神经网络更稳定

**激活函数输出值的范围：**

- 激活函数输出为有限值时，基于梯度的优化方法更稳定
- 激活函数输出为无限值时，建议调小学习率

常见的激活函数有：sigmoid，tanh，ReLU，Leaky ReLU，PReLU，RReLU，ELU（Exponential Linear Units），softplus，softsign，softmax等。

激活函数的详细见老师笔记。

对于初学者的建议：

1. 首选ReLU激活函数；
2. 学习率设置较小值；
3. 输入特征标准化，即让输入特征满足以0为均值，1为标准差的正态分布；
4. 初始化问题：初始参数中心化，即让随机生成的参数满足以0为均值，$\sqrt{\frac{2}{\text{当前层输入特征个数为标准差的}}}$ 为标准差的正态分布。

# 4.损失函数

可以使用mse、交叉熵等。

详情见老师笔记

# 5. 欠拟合过拟合与正则化

## 5.1 欠拟合与过拟合

欠拟合：与数据集拟合的不好
**解决方法:**

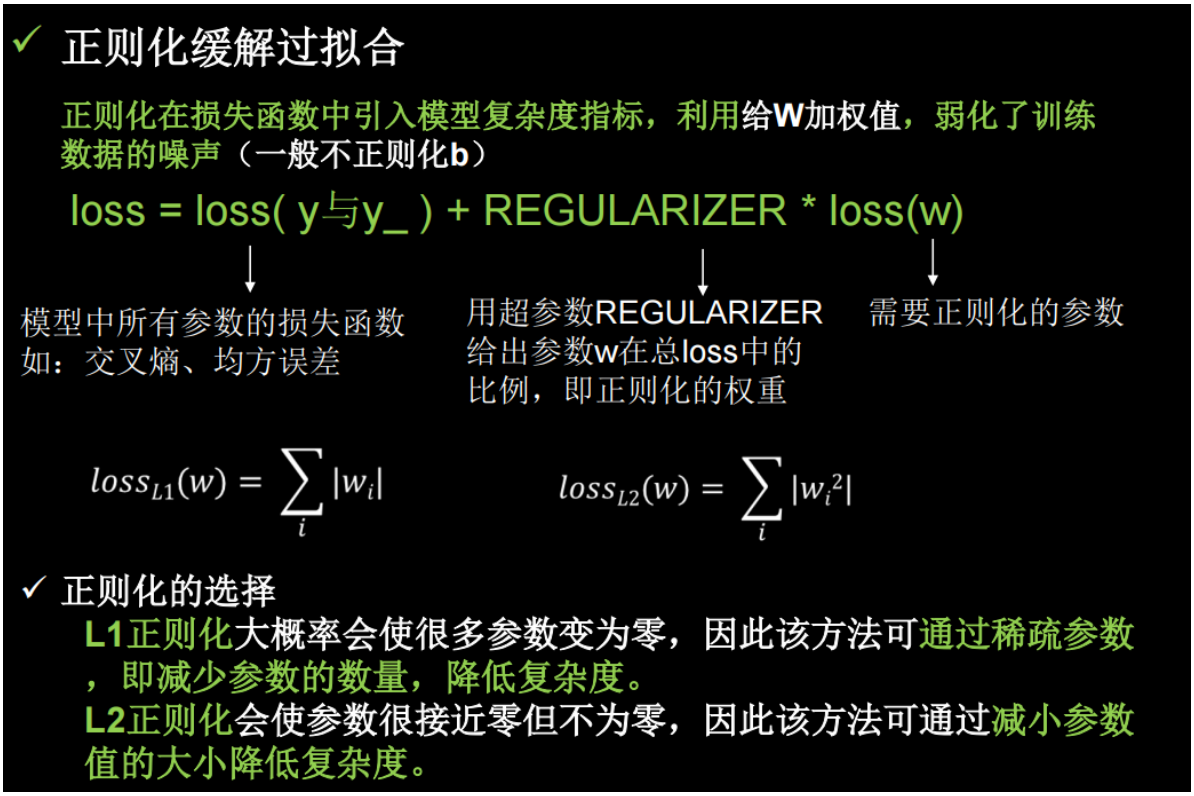
- 增加输入特征项
- 增加网络参数
- 减少正则化参数

过拟合：与数据集拟合的太好，在新的数据集可能不会这么好
**解决方法:**

- 数据清洗
- 增大训练集
- 采用正则化
- 增大正则化参数

## 5.2 L1与L2正则化

对于正则化到底是什么意思。知乎上有个15年的答案 [机器学习中常常提到的正则化到底是什么意思？ - 陶轻松的回答 - 知乎]




建立网络后，因为参数过多，可能导致网络对数据的过拟合。

因此我们在最终的Loss函数中加入所谓的正则化项（可以是参数的0\1\2阶范数(对应L0,L1,L2正则化项)），然后在最优化阶段起到对参数的约束作用。

Loss加入0阶范数之后，整个优化目标就变为了，在参数的个数最少的情况下将神经网络的Loss降到最小。(0阶范数是向量中非0向量的个数)

那么采用一阶(即$\sum_i |参数_i|$)而非0阶是因为0阶难运算，那最小化一阶其实就相当于让参数的数值和个数都同时的小。使用一阶的后果就是会让大多数参数都接近0.

而采取二阶范数，进行正则化，目的就是:让每个参数都接近于0，但是又不都为0。

直接引用知乎

> 1范数和0范数可以实现稀疏，1因具有比L0更好的优化求解特性而被广泛应用。然后L2范数，是下面这么理解的，我就直接查别人给的解释好了，反正简单，就不自己动脑子解释了：L2范数是指向量各元素的平方和然后求平方根。我们让L2范数的正则项||W||2最小，可以使得W的每个元素都很小，都接近于0，但与L1范数不同，它不会让它等于0，而是接近于0，这里是有很大的区别的哦；所以大家比起1范数，更钟爱2范数。

# 5.3代码书写

✓ 正则化缓解过拟合

```
with tf.GradientTape() as tape:  # 记录梯度信息

    h1 = tf.matmul(x_train, w1) + b1  # 记录神经网络乘加运算
    h1 = tf.nn.relu(h1)
    y = tf.matmul(h1, w2) + b2

    # 采用均方误差损失函数mse = mean(sum(y-out)^2)
    loss_mse = tf.reduce_mean(tf.square(y_train - y))
    # 添加l2正则化
    loss_regularization = []
    loss_regularization.append(tf.nn.l2_loss(w1))
    loss_regularization.append(tf.nn.l2_loss(w2))
    loss_regularization = tf.reduce_sum(loss_regularization)
    loss = loss_mse + 0.03 * loss_regularization #REGULARIZER = 0.03

# 计算loss对各个参数的梯度
variables = [w1, b1, w2, b2]
grads = tape.gradient(loss, variables)
```

# 6. 优化器

## 神经网络参数优化器

待优化参数$w$，损失函数$loss$，学习率$lr$，每次迭代一个batch，t表示当前batch迭代的总次数：

1. 计算t时刻损失函数关于当前参数的梯度$g_t = \nabla loss = \frac{\partial loss}{\partial(w_t)}$
2. 计算t时刻一阶动量$m_t$和二阶动量$V_t$
3. 计算t时刻下降梯度：$\eta_t = lr \cdot m_t / \sqrt{V_t}$
4. 计算t+1时刻参数：$w_{t+1} = w_t - \eta_t = w_t - lr \cdot m_t / \sqrt{V_t}$

一阶动量：与梯度相关的函数

二阶动量：与梯度平方相关的函数

老师提到，不同的优化器其实就是设置不同的一二阶动量。

使用不同的优化器，其实就是参照不同的动量公式，在tf中写出加入动量的参数更新公式就可以了。