

分类交叉熵

之前，我们使用平方误差的和作为网络的成本函数，但是当时我们只有单个（标量）输出值。

但是当你在使用 softmax 时，输出是向量。一个向量是输出单元的概率值。你还可以使用一种叫**独热编码(one-hot encoding)**的方法，用向量表示数据标签。

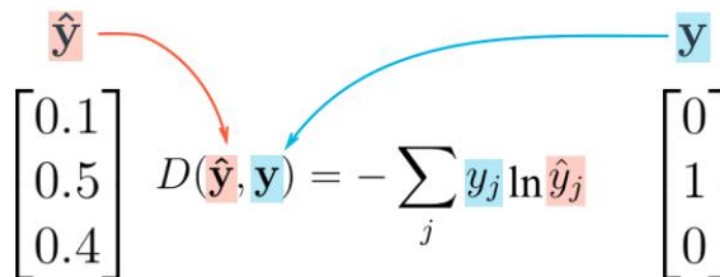
这只是表示你有一个长度为类别数量的向量，标签元素标记为 1，而其他标签设为 0。对于之前的数字分类示例，图片数字 4 的标签向量是：

$$\mathbf{y} = [0, 0, 0, 0, 1, 0, 0, 0, 0]$$

输出预测向量为：

$$\hat{\mathbf{y}} = [0.047, 0.048, 0.061, 0.07, 0.330, 0.062, 0.001, 0.213, 0.013, 0.150]$$

我们希望误差与这些向量之间的距离成比例。要计算这一距离，我们将使用**交叉熵**。我们的神经网络训练目标将为：通过尽可能地减小交叉熵使预测向量与标签向量尽量靠近。交叉熵计算公式如下所示：


$$\begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \quad D(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_j y_j \ln \hat{y}_j \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Cross entropy calculation

可以从上文中看出，交叉熵等于标签元素的和乘以预测概率的自然对数。注意，该等式并不对称！千万不能交换向量，因为标签向量里有很多 0，对 0 求对数将产生错误。

对标签向量使用独热编码的好处是除了为真的标签数值是 1 之外，其他所有的 y_j 项都为 0。因此，除了 $y_j = 1$ ，其他所有项加起来为 0，交叉熵直接变成 $D = -\ln \hat{y}$ 。例如，输入图片为数字 4 并且标为 4，那么只有与 4 对应的单元的输出，在交叉熵成本函数中才会产生影响。

练习题

如果标签向量为 $[0, 0, 0, 1, 0]$ ，预测的概率为 $[0.27, 0.11, 0.33, 0.10, 0.19]$ ，那么交叉熵是多少？

$$\log(0.1)$$

TensorFlow 中的交叉熵 (Cross Entropy)

与 softmax 一样，TensorFlow 也有一个函数可以方便地帮我们实现交叉熵。

$$D(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_j y_j \ln \hat{y}_j$$

Cross entropy loss function 交叉熵损失函数

让我们把你从视频当中学到的知识，在 TensorFlow 中来创建一个交叉熵函数。创建一个交叉熵函数，你需要用到这两个新的函数：

- `tf.reduce_sum()`
- `tf.log()`

Reduce Sum

```
x = tf.reduce_sum([1, 2, 3, 4, 5]) # 15
```

`tf.reduce_sum()` 函数输入一个序列，返回他们的和

Natural Log

```
x = tf.log(100) # 4.60517
```

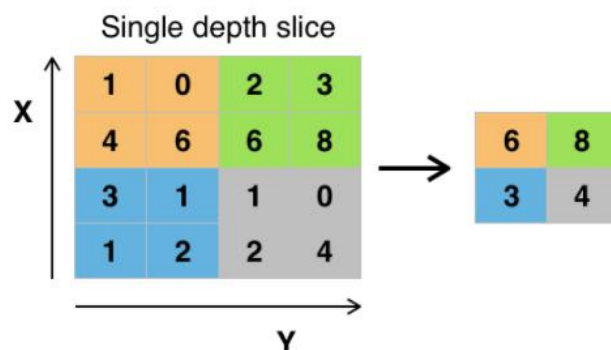
`tf.log()` 所做跟你所想的一样，它返回所输入值的自然对数。

练习

用 `softmax_data` 和 `one_hot_encoded_label` 打印交叉熵

```
quiz.py  solution.py
1 # Solution is available in the other "solution.py" tab
2 import tensorflow as tf
3
4 softmax_data = [0.7, 0.2, 0.1]
5 one_hot_data = [1.0, 0.0, 0.0]
6
7 softmax = tf.placeholder(tf.float32)
8 one_hot = tf.placeholder(tf.float32)
9
10 # TODO: Print cross entropy from session
11 cross_entropy = -tf.reduce_sum(tf.multiply(tf.log(softmax), one_hot))
12 with tf.Session() as sess:
13     output = sess.run(cross_entropy, feed_dict = {softmax:softmax_data, one_hot:one_hot_data })
14     print(output)
```

TensorFlow 最大池化



由 Aphex34 (自己的作品 [CC BY-SA 4.0](#)), 通过 Wikimedia Commons 共享

这是一个最大池化的例子 **max pooling** 用了 2×2 的滤波器 stride 为 2。四个 2×2 的颜色代表滤波器移动每个步长所产生的最大值。

例如 $[[1, 0], [4, 6]]$ 生成 6，因为 6 是这 4 个数字中最大的。同理 $[[2, 3], [6, 8]]$ 生成 8。理论上，最大池化操作的好处是减小输入大小，使得神经网络能够专注于最重要的元素。最大池化只取覆盖区域中的最大值，其它的值都丢弃。

TensorFlow 提供了 `tf.nn.max_pool()` 函数，用于对卷积层实现 **最大池化**。

```
...
conv_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')
conv_layer = tf.nn.bias_add(conv_layer, bias)
conv_layer = tf.nn.relu(conv_layer)
# Apply Max Pooling
conv_layer = tf.nn.max_pool(
    conv_layer,
    ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1],
    padding='SAME')
```

`tf.nn.max_pool()` 函数实现最大池化时，`ksize` 参数是滤波器大小，`strides` 参数是步长。 2×2 的滤波器配合 2×2 的步长是常用设定。

`ksize` 和 `strides` 参数也被构建为四个元素的列表，每个元素对应 input tensor 的一个维度 (`[batch, height, width, channels]`)，对 `ksize` 和 `strides` 来说，batch 和 channel 通常都设置成 1。

TensorFlow Softmax

The softmax 函数可以把它的输入，通常被称为 **logits** 或者 **logit scores**，处理成 0 到 1 之间，并且能够把输出归一化到和为 1。这意味着 softmax 函数与分类的概率分布等价。它是一个网络预测多类问题的最佳输出激活函数。



softmax 函数的实际应用示例

TensorFlow Softmax

当我们用 TensorFlow 来构建一个神经网络时，相应地，它有一个计算 softmax 的函数。

```
x = tf.nn.softmax([2.0, 1.0, 0.2])
```

就是这么简单，**tf.nn.softmax()** 直接为你实现了 softmax 函数，它输入 logits，返回 softmax 激活函数。

练习

在下面使用 softmax 函数返回 logits 的 softmax。

quiz.py

solution.py

```
1 # Solution is available in the other "solution.py" tab
2 import tensorflow as tf
3
4
5 def run():
6     output = None
7     logit_data = [2.0, 1.0, 0.1]
8     logits = tf.placeholder(tf.float32)
9
10    # TODO: Calculate the softmax of the logits
11    softmax = tf.nn.softmax(logit_data)
12
13    with tf.Session() as sess:
14
15        # TODO: Feed in the logit data
16        output = sess.run(softmax, feed_dict={logits: logit_data} )
17
18    return output
19
```

TensorFlow 卷积层

让我们看下如何在 TensorFlow 里面实现 CNN。

TensorFlow 提供了 `tf.nn.conv2d()` 和 `tf.nn.bias_add()` 函数来创建你自己的卷积层。

```
# Output depth
k_output = 64

# Image Properties
image_width = 10
image_height = 10
color_channels = 3

# Convolution filter
filter_size_width = 5
filter_size_height = 5

# Input/Image
input = tf.placeholder(
    tf.float32,
    shape=[None, image_height, image_width, color_channels])

# Weight and bias
weight = tf.Variable(tf.truncated_normal(
    [filter_size_height, filter_size_width, color_channels, k_output]))
bias = tf.Variable(tf.zeros(k_output))

# Apply Convolution
conv_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')
# Add bias
conv_layer = tf.nn.bias_add(conv_layer, bias)
# Apply activation function
conv_layer = tf.nn.relu(conv_layer)
```

上述代码用了 `tf.nn.conv2d()` 函数来计算卷积, `weights` 作为滤波器, `[1, 2, 2, 1]` 作为 `strides`。TensorFlow 对每一个 `input` 维度使用一个单独的 `stride` 参数, `[batch, input_height, input_width, input_channels]`。我们通常把 `batch` 和 `input_channels` (`strides` 序列中的第一个第四个) 的 `stride` 设为 1。

你可以专注于修改 `input_height` 和 `input_width`, `batch` 和 `input_channels` 都设置成 1。`input_height` 和 `input_width` `strides` 表示滤波器在 `input` 上移动的步长。上述例子中, 在 `input` 之后, 设置了一个 5x5, `stride` 为 2 的滤波器。

`tf.nn.bias_add()` 函数对矩阵的最后一维加了偏置项。