

龙心尘

专注机器学习与数据挖掘

目录视图

摘要视图

RSS 订阅

个人资料



龙心尘



访问：441189次
积分：3292
等级： 5
排名：第6991名

原创：38篇
转载：0篇
译文：0篇
评论：147条

个人介绍与联系方式

龙心尘

『五道口计算机学院』毕业，有几年机器学习/数据挖掘工作经验。某厂打杂，做过用户画像、智能营销策略、网络安全机器学习、NLP等项目。欢迎联系和交流。

EMAIL:
johnnygong.ml@gmail.com

QQ: 3253950332

数据科学沙龙群:
169492443（不定期在线分享相关知识经验）

机器学习交流群: 439183906(已满), 373038809(已满), 194141072

专业工作或者研究人员分享群: 472059892

文章搜索

文章分类

【公告】博客系统优化升级 【收藏】Html5 精品资源汇集 前端开发人员必须了解的七大技能图谱

深度学习与计算机视觉系列(9)_串一串神经网络之动手实现小例子

标签： 计算机视觉 深度学习 神经网络

2016-01-15 10:12 2228人阅读 评论(3) 收藏 举报

分类： 机器学习（22） 计算机视觉（10）

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

作者：寒小阳&&龙心尘

时间：2016年1月。

出处：

http://blog.csdn.net/han_xiaoyang/article/details/50521072

http://blog.csdn.net/longxinchen_ml/article/details/50521933

声明：版权所有，转载请联系作者并注明出处

1.引言

前面8小节，算从神经网络的结构、简单原理、数据准备与处理、神经元选择、损失函数选择等方面把神经网络过了一遍。这个部分我们打算把知识点串一串，动手实现一个简单的2维平面神经网络分类器，去分割平面上的不同类别样本点。为了循序渐进，我们打算先实现一个简单的线性分类器，然后再拓展到非线性的2层神经网络。我们可以看到简单的浅层神经网络，在这个例子上就能够有分割程度远高于线性分类器的效果。

2.样本数据的产生

为了凸显一下神经网络强大的空间分割能力，我们打算产生出一部分对于线性分类器不那么容易分割的样本点，比如我们生成一份螺旋状分布的样本点，如下：

```
1 N = 100 # 每个类中的样本点
2 D = 2 # 维度
3 K = 3 # 类别个数
4 X = np.zeros((N*K,D)) # 样本input
5 y = np.zeros(N*K, dtype='uint8') # 类别标签
6 for j in xrange(K):
7     ix = range(N*j,N*(j+1))
8     r = np.linspace(0, 1, N) # radius
9     t = np.linspace(j*4, (j+1)*4, N) + np.random.randn(N)*0.2 # theta
10    X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
11    y[ix] = j
12 # 可视化一下我们的样本点
13 plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
```

得到如下的样本分布：

机器学习 (23)

计算机视觉 (11)

手把手入门神经网络 (3)

自然语言处理 (5)

ML学习分享系列 (2)

DL+NLP (8)

文章存档

2016年07月 (3)

2016年06月 (5)

2016年04月 (2)

2016年03月 (2)

2016年02月 (7)

展开

阅读排行

能模仿韩寒小四写作的神

机器学习系列(9)_机器学习 (29424)

深度学习与自然语言处理 (22671)

机器学习系列(6)_从白富 (21500)

机器学习系列(7)_机器学习 (21065)

机器学习系列(8)_读《Ne (20382)

手把手入门神经网络系列 (18482)

手把手入门神经网络系列 (17682)

深度学习与计算机视觉系 (15177)

深度学习与计算机视觉系 (14566)

深度学习与计算机视觉系 (14553)

评论排行

机器学习系列(3)_逻辑回 (19)

机器学习系列(5)_从白富 (16)

机器学习系列(7)_机器学习 (15)

机器学习系列(2)_用初等 (13)

机器学习系列(6)_从白富 (10)

NLP系列(1)_从破译外星 (9)

手把手入门神经网络系列 (8)

机器学习系列(8)_读《Ne (8)

NLP系列(2)_用朴素贝叶 (6)

手把手入门神经网络系列 (5)

推荐文章

*Android RoccoFix 热修复框架

* android6.0源码分析之Camera API2.0下的初始化流程分析

*Android_GestureDetector手势滑动使用

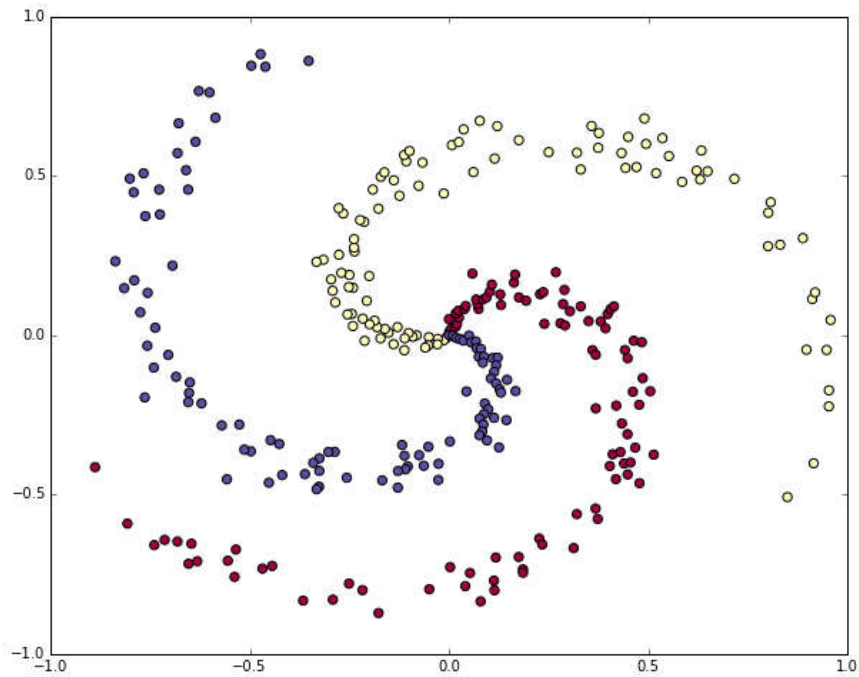
*Android MaterialList源码解析

*Android官方开发文档Training 系列课程中文版：创建自定义View之View的创建

最新评论

深度学习与自然语言处理(4)_斯坦福12期-王可欣: 我的今天 我这是什么都看不懂的节奏啊

深度学习与自然语言处理(4)_斯坦福我住在贝克街221B: 外国的大学，根本不用你私下学什么，课上认真听，课后作业做好，等大学出来，说不上全才，至少你找个工作应...



紫色，红色和黄色分布代表不同的3种类别。

一般来说，拿到数据都要做预处理，包括之前提到的去均值和方差归一化。不过我们构造的数据幅度已经在-1到1之间了，所以这里不用做这个操作。

3.使用Softmax线性分类器

3.1 初始化参数

我们先在训练集上用softmax线性分类器试试。如我们在之前的章节提到的，我们这里用的softmax分类器，使用的是一个线性的得分函数/score function，使用的损失函数是互熵损失/cross-entropy loss。包含的参数包括得分函数里面用到的权重矩阵 W 和偏移量 b ，我们先随机初始化这些参数。

```
1 #随机初始化参数
2 import numpy as np
3 #D=2表示维度，K=3表示类别数
4 W = 0.01 * np.random.randn(D, K)
5 b = np.zeros((1, K))
```

3.2 计算得分

线性的得分函数，将原始的数据映射到得分域非常简单，只是一个直接的矩阵乘法。

```
1 #使用得分函数计算得分
2 scores = np.dot(X, W) + b
```

在我们给的这个例子中，我们有2个2维点集，所以做完乘法过后，矩阵得分 `scores` 其实是一个[300*3]的矩阵，每一行都给出对应3各类别(紫，红，黄)的得分。

3.3 计算损失

然后我们就要开始使用我们的损失函数计算损失了，我们之前也提到过，损失函数计算出来的结果代表着预测结果和真实结果之间的吻合度，我们的目标是最小化这个结果。直观一点理解，我们希望对每个样本而言，对应正确类别的得分高于其他类别的得分，如果满足这个条件，那么损失函数计算的结果是一个比较低的值，如果判定的类别不是正确类别，则结果值会很高。我们之前提到了，softmax分类器里面，使用的损失函数是互熵损失。一起回忆一下，假设 f 是得分向量，那么我们的互熵损失是用如下的形式定义的：

深度学习与自然语言处理(1)_斯坦福cs224d: 在这一步, 将这些向量取平均 $v^a = (Vc - m + Vc - m + 1 + \dots + Vc + m) / 2m$ 应该是 $v^a = (Vc - \dots$

斯坦福cs224d (深度学习在自然Lucky7: 两个好基友啊! 加油

斯坦福cs224d (深度学习在自然邱承佳: 博主加油

斯坦福cs224d (深度学习在自然邱承佳: 博主加油

机器学习系列(5)_从白富美相亲helloR123: 是的

机器学习系列(5)_从白富美相亲helloR123: 看了博主写的几篇文章,真的觉得写的好好,受益匪浅~有个疑问:这篇文章说的“我们期望P(z)的概率越...

NLP系列(3)_用朴素贝叶斯进行FFNNNNNNNN: 那中文的拼写纠错怎么办呢? 计算编辑距离不太管用把, 因为中文的话, 词长大概都是2~3个的, 编辑距离为1...

深度学习与自然语言处理(1)_斯坦福ming_road: 非常感谢分享...能否留微信,方便后续交流请教.我的微信:85791418. 谢谢

$$L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$$

直观地理解一下上述形式, 就是Softmax分类器把类别得分向量 f 中每个值都看成对应三个类别的log似然概率。因此我们在求每个类别对应概率的时候, 使用指数函数还原它, 然后归一化。从上面形式里面大家也可以看得出来, 得到的值总是在0到1之间的, 因此从某种程度上说我们可以把它理解成概率。如果判定类别是错误类别, 那么上述公式的结果就会趋于无穷, 也就是说损失相当相当大, 相反, 如果判定类别正确, 那么损失就接近 $\log(1) = 0$ 。这和我们直观理解上要最小化 损失 是完全吻合的。

当然, 当然, 别忘了, 完整的损失函数定义, 一定会加上正则化项, 也就是说, 完整的损失 L 应该有如下的形式:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\frac{1}{2} \lambda \sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}}$$

好, 我们实现以下, 根据上面计算得到的得分 `scores`, 我们计算以下各个类别上的概率:

```
1 # 用指数函数还原
2 exp_scores = np.exp(scores)
3 # 归一化
4 probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
```

在我们的例子中, 我们最后得到了一个[300*3]的概率矩阵 `prob`, 其中每一行都包含属于3个类别的概率。然后我们就可以计算完整的互熵损失了:

```
1 #计算log概率和互熵损失
2 correct_logprobs = -np.log(probs[range(num_examples), y])
3 data_loss = np.sum(correct_logprobs)/num_examples
4 #加上正则化项
5 reg_loss = 0.5*reg*np.sum(W*W)
6 loss = data_loss + reg_loss
```

正则化强度 λ 在上述代码中是`reg`, 最开始的时候我们可能会得到 `loss=1.1`, 是通过 `np.log(1.0/3)` 得到的(假定初始的时候属于3个类别的概率一样), 我们现在想最小化损失 `loss`

3.4 计算梯度与梯度回传

我们能够用损失函数评估预测值与真实值之间的差距, 下一步要做的事情自然是最小化这个值。我们用传统的梯度下降来解决这个问题。多解释一句, 梯度下降的过程是: 我们先选取一组随机参数作为初始值, 然后计算损失函数在这组参数上的梯度(负梯度的方向表明了损失函数减小的方向), 接着我们朝着负梯度的方向迭代和更新参数, 不断重复这个过程直至损失函数最小化。为了清楚一点说明这个问题, 我们引入一个中间变量 p , 它是归一化后的概率向量, 如下:

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \quad L_i = -\log(p_{y_i})$$

我们现在希望知道朝着哪个方向调整权重能够减小损失, 也就是说, 我们需要计算梯度 $\partial L_i / \partial f_k$ 。损失 L_i 从 p 计算而来, 再退一步, 依赖于 f 。于是我们又要做高数题, 使用链式求导法则了, 不过梯度的结果倒是非常简单:

$$\frac{\partial L_i}{\partial f_k} = p_k - 1(y_i = k)$$

解释一下, 公式的最后一个部分表示 $y_i = k$ 的时候, 取值为1。整个公式其实非常的优雅和简单。假设我们计算的概率 $p = [0.2, 0.3, 0.5]$, 而中间类别才是真实的结果类别。根据梯度求解公式, 我们得到梯度 $d_f = [0.2, -0.7, 0.5]$ 。我们想想梯度的含义, 其实这个结果是可解释性非常高的: 大家都知道, 梯度是最快上升方向, 我们减掉它乘以步长才会让损失函数值减小。第1项和第3项(其实就是不正确的类别项)梯度为正, 表明增加它们只会让最后的损失/loss增大, 而我们的目标是减小loss; 中间的梯度项-0.7其实再告诉我们, 增加这一项, 能减小损失 L_i , 达到我们最终的目的。

我们依旧记 `probs` 为所有样本属于各个类别的概率, 记 `dscores` 为得分上的梯度, 我们可以有以下的代码:

```

1 dscores = probs
2 dscores[range(num_examples), y] -= 1
3 dscores /= num_examples

```

我们计算的得分 $\text{scores} = \text{np.dot}(X, W) + b$ ，因为上面已经算好了 scores 的梯度 dscores ，我们现在可以回传梯度计算 W 和 b 了：

```

1 dW = np.dot(X.T, dscores)
2 db = np.sum(dscores, axis=0, keepdims=True)
3 #得记着正则化梯度哈
4 dW += reg*W

```

我们通过矩阵的乘法得到梯度部分，权重 W 的部分加上了正则化项的梯度。因为我们在设定正则化项的时候用了系数 $0.5 \left(\frac{d}{dw} \left(\frac{1}{2} \lambda w^2 \right) = \lambda w \right)$ ，因此直接用 $\text{reg} * W$ 就可以表示出正则化的梯度部分。

3.5 参数迭代与更新

在得到所需的所有部分之后，我们就可以进行参数更新了：

```

1 #参数迭代更新
2 W += -step_size * dW
3 b += -step_size * db

```

3.6 大杂合：训练SoftMax分类器

```

1 #代码部分组一起，训练线性分类器
2
3 #随机初始化参数
4 W = 0.01 * np.random.randn(D, K)
5 b = np.zeros((1, K))
6
7 #需要自己敲定的步长和正则化系数
8 step_size = 1e-0
9 reg = 1e-3 #正则化系数
10
11 #梯度下降迭代循环
12 num_examples = X.shape[0]
13 for i in xrange(200):
14
15     # 计算类别得分，结果矩阵为[N x K]
16     scores = np.dot(X, W) + b
17
18     # 计算类别概率
19     exp_scores = np.exp(scores)
20     probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]
21
22     # 计算损失loss(包括互熵损失和正则化部分)
23     correct_logprobs = -np.log(probs[range(num_examples), y])
24     data_loss = np.sum(correct_logprobs) / num_examples
25     reg_loss = 0.5 * reg * np.sum(W * W)
26     loss = data_loss + reg_loss
27     if i % 10 == 0:
28         print "iteration %d: loss %f" % (i, loss)
29
30     # 计算得分上的梯度
31     dscores = probs
32     dscores[range(num_examples), y] -= 1
33     dscores /= num_examples
34
35     # 计算和回传梯度
36     dW = np.dot(X.T, dscores)
37     db = np.sum(dscores, axis=0, keepdims=True)
38
39     dW += reg * W # 正则化梯度
40
41     #参数更新
42     W += -step_size * dW
43     b += -step_size * db

```

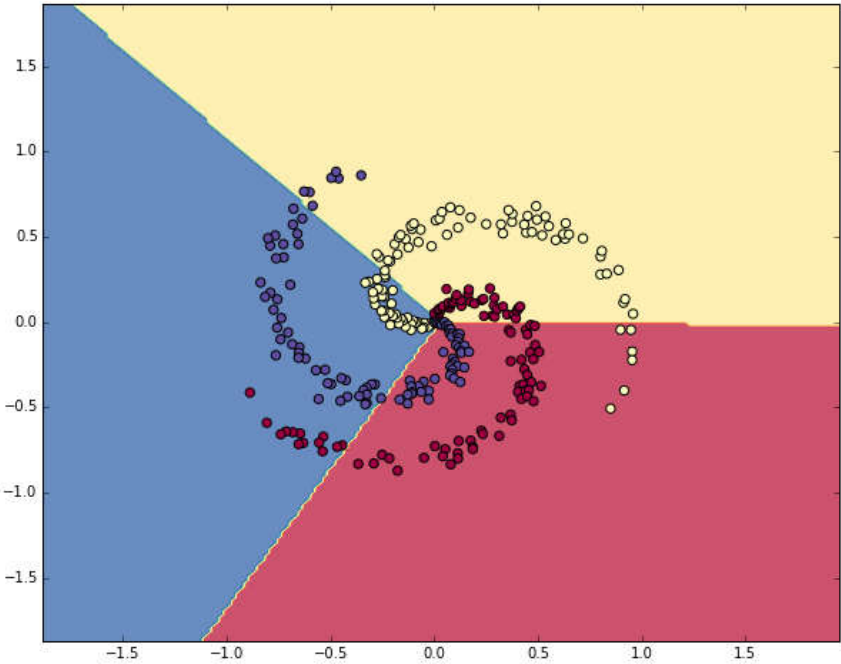
得到结果：

```
1 iteration 0: loss 1.096956
2 iteration 10: loss 0.917265
3 iteration 20: loss 0.851503
4 iteration 30: loss 0.822336
5 iteration 40: loss 0.807586
6 iteration 50: loss 0.799448
7 iteration 60: loss 0.794681
8 iteration 70: loss 0.791764
9 iteration 80: loss 0.789920
10 iteration 90: loss 0.788726
11 iteration 100: loss 0.787938
12 iteration 110: loss 0.787409
13 iteration 120: loss 0.787049
14 iteration 130: loss 0.786803
15 iteration 140: loss 0.786633
16 iteration 150: loss 0.786514
17 iteration 160: loss 0.786431
18 iteration 170: loss 0.786373
19 iteration 180: loss 0.786331
20 iteration 190: loss 0.786302
```

190次循环之后，结果大致收敛了。我们评估一下准确度：

```
1 #评估准确度
2 scores = np.dot(X, W) + b
3 predicted_class = np.argmax(scores, axis=1)
4 print 'training accuracy: %.2f' % (np.mean(predicted_class == y))
```

输出结果为49%。不太好，对吧？实际上也是可理解的，你想想，一份螺旋形的数据，你偏执地要用一个线性分类器去分割，不管怎么调整这个线性分类器，都非常非常困难。我们可视化一下数据看看决策边界(decision boundaries)：



4.使用神经网络分类

从刚才的例子中可以看出，一个线性分类器，在现在的数据集上效果并不好。我们知道神经网络可以做非线性的分割，那我们就试试神经网络，看看会不会有更好的效果。对于这样一个简单问题，我们用单隐藏层的神经网络就可

以了，这样一个神经网络我们需要2层的权重和偏移量：

```
1 # 初始化参数
2 h = 100 # 隐层大小(神经元个数)
3 W = 0.01 * np.random.randn(D, h)
4 b = np.zeros((1, h))
5 W2 = 0.01 * np.random.randn(h, K)
6 b2 = np.zeros((1, K))
```

然后前向计算的过程也稍有一些变化：

```
1 #2层神经网络的前向计算
2 hidden_layer = np.maximum(0, np.dot(X, W) + b) # 用的 ReLU单元
3 scores = np.dot(hidden_layer, W2) + b2
```

注意到这里，和之前线性分类器中的得分计算相比，多了一行代码计算，我们首先计算第一层神经网络结果，然后作为第二层的输入，计算最后的结果。哦，对了，代码里大家也看的出来，我们这里使用的是ReLU神经单元。

其他的东西都没太大变化。我们依旧按照之前的方式去计算loss，然后计算梯度 `dscores`。不过反向回传梯度的过程形式上也有些小小的变化。我们看下面的代码，可能觉得和Softmax分类器里面看到的基本一样，但注意到我们用 `hidden_layer` 替换掉了之前的 `X`：

```
1 # 梯度回传与反向传播
2 # 对W2和b2的第一次计算
3 dW2 = np.dot(hidden_layer.T, dscores)
4 db2 = np.sum(dscores, axis=0, keepdims=True)
```

恩，并没有完事啊，因为 `hidden_layer` 本身是一个包含其他参数和数据的函数，我们得计算一下它的梯度：

```
1 dhidden = np.dot(dscores, W2.T)
```

现在我们有隐层输出的梯度了，下一步我们要反向传播到ReLU神经元了。不过这个计算非常简单，因为 $r = \max(0, x)$ ，同时我们又有 $\frac{dr}{dx} = 1(x > 0)$ 。用链式法则串起来后，我们可以看到，回传的梯度大于0的时候，经过ReLU之后，保持原样；如果小于0，那本次回传就到此结束了。因此，我们这一部分非常简单：

```
1 #梯度回传经过ReLU
2 dhidden[hidden_layer <= 0] = 0
```

终于，翻山越岭，回到第一层，拿到总的权重和偏移量的梯度：

```
1 dW = np.dot(X.T, dhidden)
2 db = np.sum(dhidden, axis=0, keepdims=True)
```

来，来，来。组一组，我们把整个神经网络的过程串起来：

```
1 # 随机初始化参数
2 h = 100 # 隐层大小
3 W = 0.01 * np.random.randn(D, h)
4 b = np.zeros((1, h))
5 W2 = 0.01 * np.random.randn(h, K)
6 b2 = np.zeros((1, K))
7
8 # 手动敲定的几个参数
9 step_size = 1e-0
10 reg = 1e-3 # 正则化参数
11
12 # 梯度迭代与循环
13 num_examples = X.shape[0]
14 for i in xrange(10000):
15
16     hidden_layer = np.maximum(0, np.dot(X, W) + b) #使用的ReLU神经元
17     scores = np.dot(hidden_layer, W2) + b2
18
19     # 计算类别概率
20     exp_scores = np.exp(scores)
21     probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]
```

```
22
23     # 计算熵损失与正则化项
24     corect_logprobs = -np.log(probs[range(num_examples), y])
25     data_loss = np.sum(corect_logprobs)/num_examples
26     reg_loss = 0.5*reg*np.sum(W*W) + 0.5*reg*np.sum(W2*W2)
27     loss = data_loss + reg_loss
28     if i % 1000 == 0:
29         print "iteration %d: loss %f" % (i, loss)
30
31     # 计算梯度
32     dscores = probs
33     dscores[range(num_examples), y] -= 1
34     dscores /= num_examples
35
36     # 梯度回传
37     dW2 = np.dot(hidden_layer.T, dscores)
38     db2 = np.sum(dscores, axis=0, keepdims=True)
39
40     dhidden = np.dot(dscores, W2.T)
41
42     dhidden[hidden_layer <= 0] = 0
43     # 拿到最后W, b上的梯度
44     dW = np.dot(X.T, dhidden)
45     db = np.sum(dhidden, axis=0, keepdims=True)
46
47     # 加上正则化梯度部分
48     dW2 += reg * W2
49     dW += reg * W
50
51     # 参数迭代与更新
52     W += -step_size * dW
53     b += -step_size * db
54     W2 += -step_size * dW2
55     b2 += -step_size * db2
```

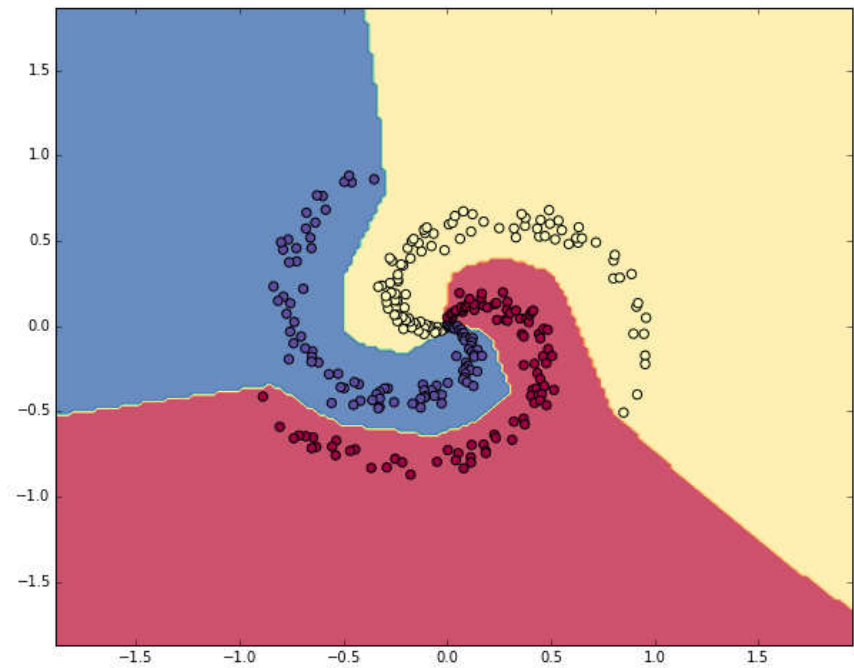
输出结果：

```
1 iteration 0: loss 1.098744
2 iteration 1000: loss 0.294946
3 iteration 2000: loss 0.259301
4 iteration 3000: loss 0.248310
5 iteration 4000: loss 0.246170
6 iteration 5000: loss 0.245649
7 iteration 6000: loss 0.245491
8 iteration 7000: loss 0.245400
9 iteration 8000: loss 0.245335
10 iteration 9000: loss 0.245292
```

现在的训练准确度为：

```
1 #计算分类准确度
2 hidden_layer = np.maximum(0, np.dot(X, W) + b)
3 scores = np.dot(hidden_layer, W2) + b2
4 predicted_class = np.argmax(scores, axis=1)
5 print 'training accuracy: %.2f' % (np.mean(predicted_class == y))
```

你猜怎么着，准确度为98%，我们可视化一下数据和现在的决策边界：



看起来效果比之前好多了，这充分地印证了我们在手把手入门神经网络系列(1)_从初等数学的角度初探神经网络中提到的，神经网络对于空间强大的分割能力，对于非线性的不规则图形，能有很强的划分区域和区分类别能力。

参考资料与原文

cs231n 神经网络小例子

顶

4

踩

0

上一篇

深度学习与计算机视觉系列(8)_神经网络训练与注意点

下一篇

NLP系列(1)_从破译外星人文字浅谈自然语言处理基础

我的同类文章

机器学习（22） 计算机视觉（10）

• 能模仿韩寒小四写作的神奇...

2016-04-26

阅读 29422

• 机器学习系列(8)_读《Natur...

2016-03-16

阅读 18479

• NLP系列(4)_朴素贝叶斯实战..

2016-02-03

阅读 10572

• 深度学习与计算机视觉系列(...

2016-01-19

阅读 5070

• 机器学习系列(6)_从白富美相..

2016-01-10

阅读 21064

• 机器学习系列(9)_机器学习算..

2016-04-19

阅读 22671

• 机器学习系列(7)_机器学习路..

2016-02-28

阅读 20377

• NLP系列(3)_用朴素贝叶斯进..

2016-02-03

阅读 9847

• 深度学习与计算机视觉系列(...

2016-01-15

阅读 2268

• 机器学习系列(5)_从白富美相..

2016-01-06

阅读 14009

更多文章

猜你在找

- MySQL数据库管理
- DB2大型数据库设计与开发入门
- 深度学习与计算机视觉系列10_细说卷积神经网络
- 深度学习与计算机视觉系列10_细说卷积神经网络

经典JDBC+MyBatis学习视频

深度学习与计算机视觉系列10_细说卷积神经网络

数据库简明教程

深度学习与计算机视觉系列8_神经网络训练与注意点


C++语言基础

深度学习与计算机视觉系列7_神经网络数据预处理正则



查看评论

2楼 包包_安安 2016-02-24 15:52发表



这个神经元网络例子有完整源码吗？

1楼 howard327 2016-01-15 15:24发表



想问问作者有没有什么推荐的python开发环境，刚接触python，不太懂这些。

Re: 龙心尘 2016-01-15 16:20发表



回复howard327：python还是很简单的。有一些不错的ide与扩展包吧。一些典型的方法我们在群内有共享，如果你感兴趣也可以进来。机器学习QQ群: 439183906

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

aptech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服

杂志客服

微博客服

webmaster@csdn.net

400-600-2320

北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved

