二维神经网络实现

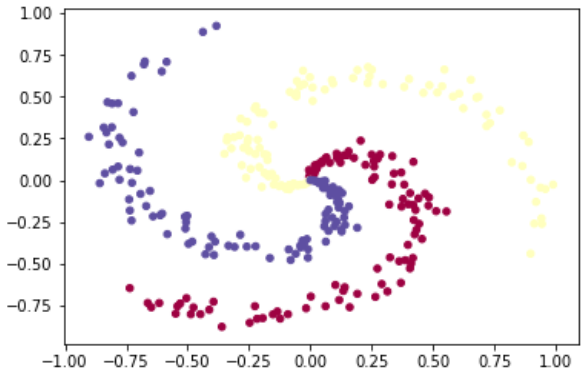
刘闯 来凯建 物理科学与技术学院 武汉大学

摘要

首先，实现一个简单的分类器，之后将代码扩展到一个两层的神经网络中

1. 生成一些数据

生成一个螺旋型的数据集，有红，黄，蓝三类，并非线性分割。而且数据值均在[-1,1] 范围内，如图。



2. 线性分类器简介

一种图像分类的方法，这种方法主要两部分组成：一个是**评分函数(Score function)**，它是原始图像数据到类别分值的映射。另一个是**损失函数（loss function）**，它是用来量化预测分类标签的得分与真实标签之间一致性的。该方法可转化为一个最优化问题，在最优化过程中，将通过更新评分函数的参数来最小化损失函数。

评分函数，这个函数将图像的像素值映射为各个分类类别的得分，得分高低代表图像属于该类别的可能性高低。

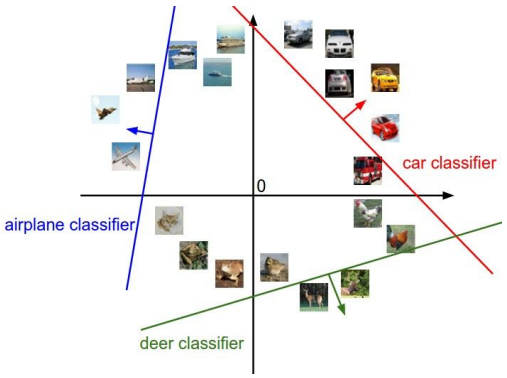
简单的线性函数 $f(x_i, W, b) = Wx_i + b$

假设每个图像数据都被拉长为一个长度为D的列向量，大小为[D x 1]。其中大小为[K x D]的矩阵**W**和大小为[K x 1]列向量**b**为该函数的**参数（parameters)**参数W被称为权重(weights)

b被称为偏差向量 (bias vector)

线性分类器计算图像中3个颜色通道中所有像素的值与权重的矩阵乘，从而得到分类分值。根据我们对权重设置的值，对于图像中的某些位置的某些颜色，函数表现出喜好或者厌恶（根据每个权重的符号而定）

将图像看做高维度的点：既然图像被伸展成为了一个高维度的列向量，那么我们可以把图像看做这个高维度空间中的一个点。 定义每个分类类别的分值是权重和图像的矩阵乘，那么



每个分类类别的分数就是这个空间中的一个线性函数的函数值。

3. SoftMax 线性分类器

SoftMax 线性分类器的损失函数（Loss function)为 cross-entropy loss

$$L_i = -log\frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}} = -s_{y_i} + log\sum_{j=1}^{C} e^{s_j}$$

使用 S 来表示分类评分向量中的第 i 个元素。和之前一样，整个数据集的损失值是数据集中所有样本数据的损失值的均值与正则化损失之和。

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

softmax 函数：其输入值是一个向量，向量中元素为任意实数的评分值，函数对其进行压缩，输出一个向量，其中每个元素值在0到1之间，且所有元素之和为1。

从**概率论的角度来理解**，就是在最小化正确分类的负对数概率。Softmax分类器将输出向量中的评分值解释为没有归一化的对数概率。那么以这些数值做指数函数的幂就得到了没有归一化的概率，而除法操作则对数据进行了归一化处理，使得这些概率的和为1。

4. 训练 SoftMax 线性分类器

Softmax分类器计算每个类别的概率，其损失函数反应的是真实样本标签label的预测概率，概率越接近 1 ，则loss越接近0。由于引入正则项，超参数λ越大，则对权重W的惩罚越大，使得W更小，分布趋于均匀，造成不同类别之间的概率分布也趋于均匀。

SoftMax分类器计算过程

1） 随机初始化参数：权重和偏置
2） 计算数据的score
3） 计算数据的损失函数
4） **反向传播算法**计算梯度
5） 进行参数更新

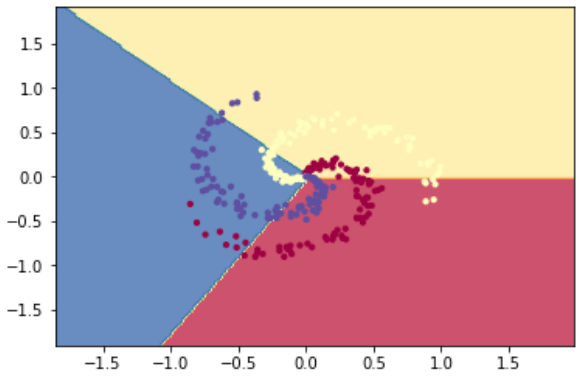
$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\log\left(p_{y_i}\right)$$

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

5. SoftMax 线性分类器训练结果

```
iteration 80: loss 0.773316
iteration 90: loss 0.771806
iteration 100: loss 0.770784
iteration 110: loss 0.770081
iteration 120: loss 0.769591
iteration 130: loss 0.769247
iteration 140: loss 0.769003
iteration 150: loss 0.768828
iteration 160: loss 0.768703
iteration 170: loss 0.768612
iteration 180: loss 0.768546
iteration 190: loss 0.768499
training accuracy: 0.52
```

正确率只有52%，仅仅比随机概率33% 高一些，分类结果不是很理想。



通过图片可以明显看出，螺旋末尾的数据都没有正确的分类。

之后**调整步长和超参数λ，发现结果没有明显的改善**

6. 训练 Neutral Network

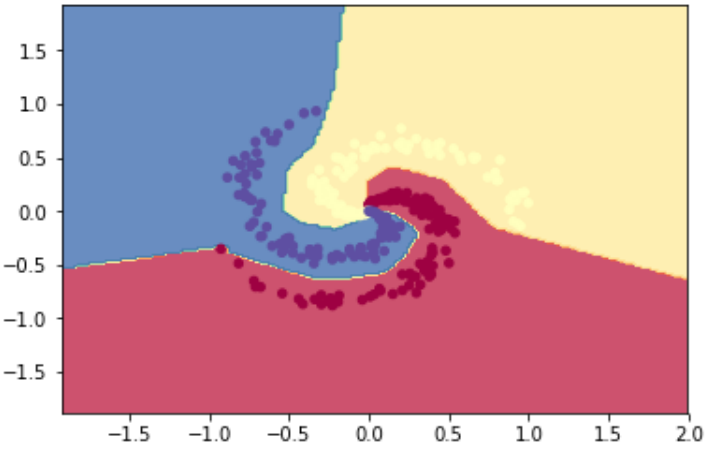
训练一个两层的神经网络：

1. 主要算法 类似于线性分类器
2. 需要两层网络，有两套权重和偏置
3. 前向计算，反向传播
4. 更新参数
5. 记录正确率
6. 数据可视化处理

7. Neutral Network训练结果

```
In [23]: runfile('/Users/liuch
liuchuang/.spyder-py3')
iteration 0: loss 1.098528
iteration 1000: loss 0.293471
iteration 2000: loss 0.265619
iteration 3000: loss 0.253780
iteration 4000: loss 0.249988
iteration 5000: loss 0.247220
iteration 6000: loss 0.245409
iteration 7000: loss 0.244855
iteration 8000: loss 0.244437
iteration 9000: loss 0.244244
training accuracy: 0.99
```

训练结果正确率是99%
下图所示，神经网络进行了较好的分类



8. 问题思考

SoftMax 分类算法中，如何进行参数的最优化。

神经网络的层数是否会影响该分类算法的正确率

9. 参考文献

1. Neural Networks and Deep Learning By Michael Nielse
2. Numpy 官方文档
3. Cs231 课堂笔记

LiuChuang0059 / ML_Project

Branch: master ▾ ML_Project / HM / Neural_Network.py Find file Copy pat

LiuChuang0059 Add files via upload

1 contributor

128 lines (79 sloc) 3.66 KB

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 23 10:36:43 2018

@author: liuchuang
"""

import numpy as np
import matplotlib.pyplot as plt


N = 100 # number of points per class  每一类的数据数目
D = 2 # dimensionality   数据纬度
K = 3 # number of classes  数据类别数目
X = np.zeros((N*K,D)) # data matrix (each row = single example) 生成一个N*K行 D列的全0矩阵
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius  每一类N个点之间的距离--均匀
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta 点的角度 在一定的范围内均匀基础上 随机取点
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]  #  确定点的坐标
  y[ix] = j # 0-N是 全是0 ;  N-2N是1 ;  2N-3N 是2;
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap=plt.cm.Spectral) # 画 散点图,  X的第一列是横坐标，第二列是纵坐
# 注意 y不是yellow  而是一个列表
plt.show()


## 训练一个线性分类器
#Train a Linear Classifier
# initialize parameters randomly 初始化权重和 偏置
W = 0.01 * np.random.randn(D,K)  # 权重是 2x3 的矩阵
b = np.zeros((1,K))    # 偏置是 1 X k 的矩阵

# some hyperparameters
step_size = 1e-1   #  步长
reg = 1e-3 # regularization strength 正则化参数

# gradient descent loop
num_examples = X.shape[0]  ## 表示 X的行数  即是 N*K 所有样本点数
for i in range(200):  # 循环200次

  # evaluate class scores, [N x K]
  scores = np.dot(X, W) + b   #  矩阵乘后 N*K行 K列 , 加b 的时候 用到numpy  的广播性质

  # compute the class probabilities
```

```
    exp_scores = np.exp(scores)  # 未进行归一化的 概率
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K, k]
    # 概率进行归一化 [300 ,3]的规模  每一行代表了 一个点对应三种分类的可能性

    # compute the loss: average cross-entropy loss and regularization
    #  按照公式 计算样本损失 和正则化损失
    correct_logprobs = -np.log(probs[range(num_examples),y])  # 遍历 所有的概率值
    data_loss = np.sum(correct_logprobs)/num_examples
    reg_loss = 0.5*reg*np.sum(W*W)
    loss = data_loss + reg_loss

    if i % 10 == 0:    # 每10次 打印一下 loss 报告
      print("iteration %d: loss %f" %(i, loss))

    # compute the gradient on scores
    dscores = probs
    dscores[range(num_examples),y] -= 1  #  下降梯度 就是 对应的概率 -1
    dscores /= num_examples  #  计算平均概率值

    # backpropate the gradient to the parameters (W,b)
    dW = np.dot(X.T, dscores)
    db = np.sum(dscores, axis=0, keepdims=True)

    dW += reg*W # regularization gradient

    # perform a parameter update
    W += -step_size * dW   #  梯度更新
    b += -step_size * db


# evaluate training set accuracy
# 计算分类的正确率
scores = np.dot(X, W) + b
predicted_class = np.argmax(scores, axis=1)
print('training accuracy: %.2f' % (np.mean(predicted_class == y)))

# plot the resulting classifier
# 绘制 分类结果
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))  # 从坐标向量返回坐标矩阵
Z = np.dot(np.c_[xx.ravel(), yy.ravel()], W) + b
Z = np.argmax(Z, axis=1)
Z = Z.reshape(xx.shape)
fig = plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8) # 绘制背分类边界
plt.scatter(X[:, 0], X[:, 1], c=y, s=10, cmap=plt.cm.Spectral) # 绘制散点图
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
```

LiuChuang0059 / ML_Project

Branch: master ▾ ML_Project / HM / Neural_Network2.py Find file Copy pa

LiuChuang0059 Add files via upload

1 contributor

116 lines (86 sloc) 3.27 KB

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 23 16:55:23 2018

@author: liuchuang
"""

import numpy as np
import matplotlib.pyplot as plt

N = 100 # number of points per class  每一类的数据数目
D = 2 # dimensionality   数据纬度
K = 3 # number of classes  数据类别数目
X = np.zeros((N*K,D)) # data matrix (each row = single example) 生成一个N*K行 D列的全0矩阵
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius  每一类N个点之间的距离--均匀
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta 点的角度 在一定的范围内均匀基础上 随机取点
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]  #  确定点的坐标
  y[ix] = j # 0-N是 全是0 ;  N-2N是1 ;  2N-3N 是2;

# initialize parameters randomly
h = 100 # size of hidden layer
W = 0.01 * np.random.randn(D,h)
b = np.zeros((1,h))
W2 = 0.01 * np.random.randn(h,K)
b2 = np.zeros((1,K))

# some hyperparameters
step_size = 1e-0
reg = 1e-3 # regularization strength

# gradient descent loop
num_examples = X.shape[0]
for i in range(10000):

  # evaluate class scores, [N x K]
  hidden_layer = np.maximum(0, np.dot(X, W) + b) # note, ReLU activation
  scores = np.dot(hidden_layer, W2) + b2

  # compute the class probabilities
  exp_scores = np.exp(scores)
  probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]

  # compute the loss: average cross-entropy loss and regularization
```

```
    correct_logprobs = -np.log(probs[range(num_examples),y])
    data_loss = np.sum(correct_logprobs)/num_examples
    reg_loss = 0.5*reg*np.sum(W*W) + 0.5*reg*np.sum(W2*W2)
    loss = data_loss + reg_loss
    if i % 1000 == 0:
      print("iteration %d: loss %f" % (i, loss))

    # compute the gradient on scores
    dscores = probs
    dscores[range(num_examples),y] -= 1
    dscores /= num_examples

    # backpropate the gradient to the parameters
    # first backprop into parameters W2 and b2
    dW2 = np.dot(hidden_layer.T, dscores)
    db2 = np.sum(dscores, axis=0, keepdims=True)
    # next backprop into hidden layer
    dhidden = np.dot(dscores, W2.T)
    # backprop the ReLU non-linearity
    dhidden[hidden_layer <= 0] = 0
    # finally into W,b
    dW = np.dot(X.T, dhidden)
    db = np.sum(dhidden, axis=0, keepdims=True)

    # add regularization gradient contribution
    dW2 += reg * W2
    dW += reg * W

    # perform a parameter update
    W += -step_size * dW
    b += -step_size * db
    W2 += -step_size * dW2
    b2 += -step_size * db2


# evaluate training set accuracy
hidden_layer = np.maximum(0, np.dot(X, W) + b)
scores = np.dot(hidden_layer, W2) + b2
predicted_class = np.argmax(scores, axis=1)
print('training accuracy: %.2f' % (np.mean(predicted_class == y)))

# 可视化处理
h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = np.dot(np.maximum(0, np.dot(np.c_[xx.ravel(), yy.ravel()], W) + b), W2) + b2
Z = np.argmax(Z, axis=1)
Z = Z.reshape(xx.shape)
fig = plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap=plt.cm.Spectral)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
```