Paradox

符号计算/深度学习框架　by 时间坐标

项目简介

- 地址：https://git.oschina.net/ictxiangxin/paradox
- 用python3.5+ 和 numpy1.13+ 搭建的深度学习框架。

```
model = pd.nn.Network()
model.add(pd.cnn.Convolution2DLayer((4, 5, 5), 'valid', input_shape=(None, 28, 28)))  # 使用4个5x5的卷积核。
model.add(pd.cnn.AveragePooling2DLayer((2, 2), (2, 2)))  # 2x2的均值池化。
model.add(pd.cnn.Convolution2DLayer((2, 3, 3), 'valid'))  # 使用2个3x3的卷积核。
model.add(pd.cnn.MaxPooling2DLayer((2, 2), (2, 2)))  # 2x2的max池化。
model.add(pd.nn.Dense(100))  # 接入100个神经元的全连接层。
model.add(pd.nn.Activation('tanh'))  # tanh激活。
model.add(pd.nn.Dense(50))
model.add(pd.nn.Activation('tanh'))
model.add(pd.nn.Dense(10))  # 接入到输出0~9数字的输出层。
model.loss('softmax')
```

- 用最简单的方式阐述基于符号计算的深度学习的原理。

符号计算


```python
import paradox as pd

A = pd.Constant([[1, 2], [1, 3]], name='A')
x = pd.Variable([0, 0], name='x')
b = pd.Constant([3, 4], name='b')

loss = pd.reduce_sum(0.5 * (A @ x - b) ** 2)

e = pd.Engine(loss, x)
print('loss formula =\n{}\n'.format(loss))
print('loss =\n{}\n'.format(e.value()))

x_gradient = e.gradient(x)
print('x gradient formula =\n{}\n'.format(x_gradient))
print('x gradient =\n{}\n'.format(pd.Engine(x_gradient).value()))
```

$$\begin{cases} x_1 + 2x_2 = 3 \\ x_1 + 3x_2 = 4 \end{cases}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad b = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\boldsymbol{A}x = b$$

$$loss = \frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{A}x - b)^2$$

$$\frac{\partial loss}{\partial x} = \boldsymbol{A}^T(\boldsymbol{A}x - b)$$

```
loss formula =
ReduceSum((0.5 * (((A @ x) - b) ** 2.0)), axis=None, invariant=False)

loss =
12.5

x gradient formula =
(Transpose(A, axes=None) @ ((((Broadcast(Broadcast(1.0, shape=()), shape=(2,)) * 0.5) * 2.0) * (((A @ x) - b) ** (2.0 - 1.0))) * 1.0))

x gradient =
[ -7. -18.]
```

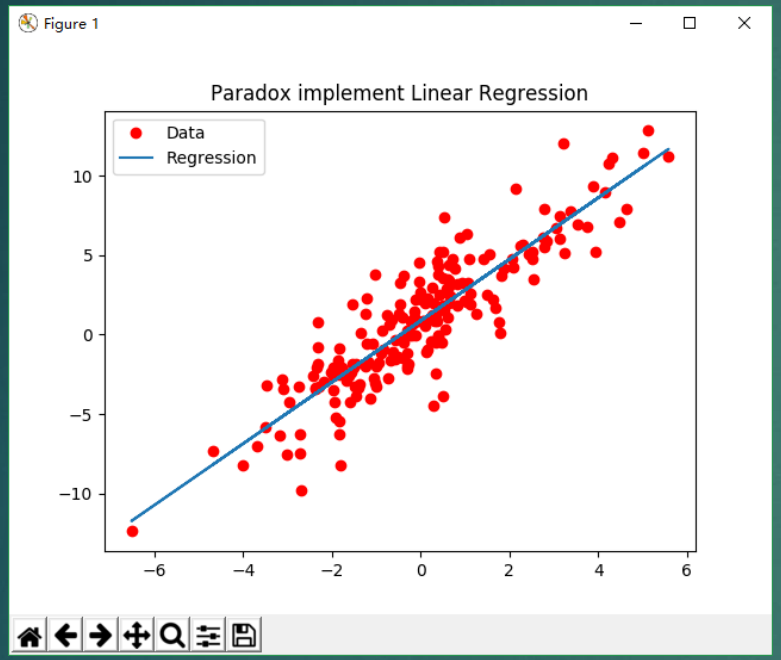
! 下一步工作：代数优化

Paradox架构

- Symbol —— 符号运算的节点（图计算）。
- Operator —— 运算符（算子），autograd的核心。
- Engine —— 驱动数值计算和梯度计算。
- Optimizer —— 优化器，最大化/最小化一个代数式。

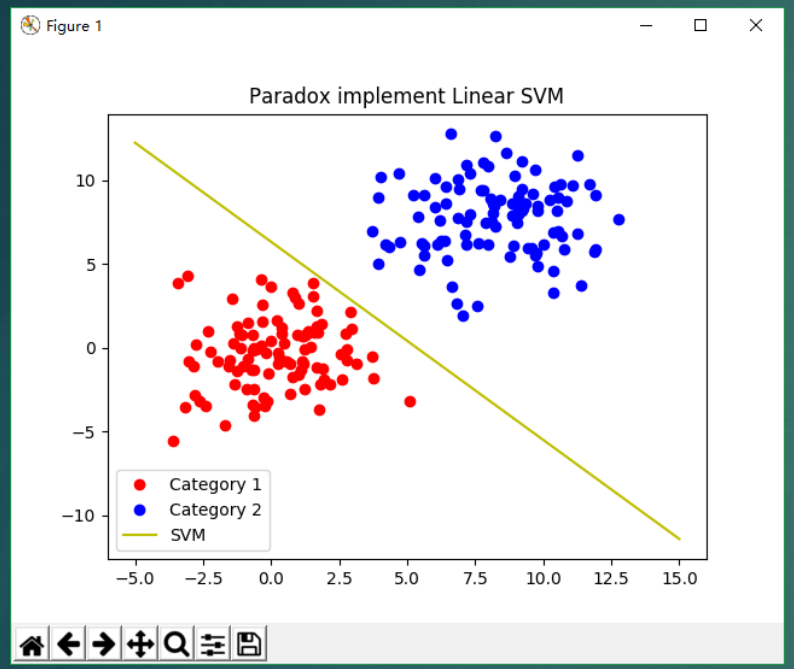
- Network —— High-level API，用于快速构建网络。

示例



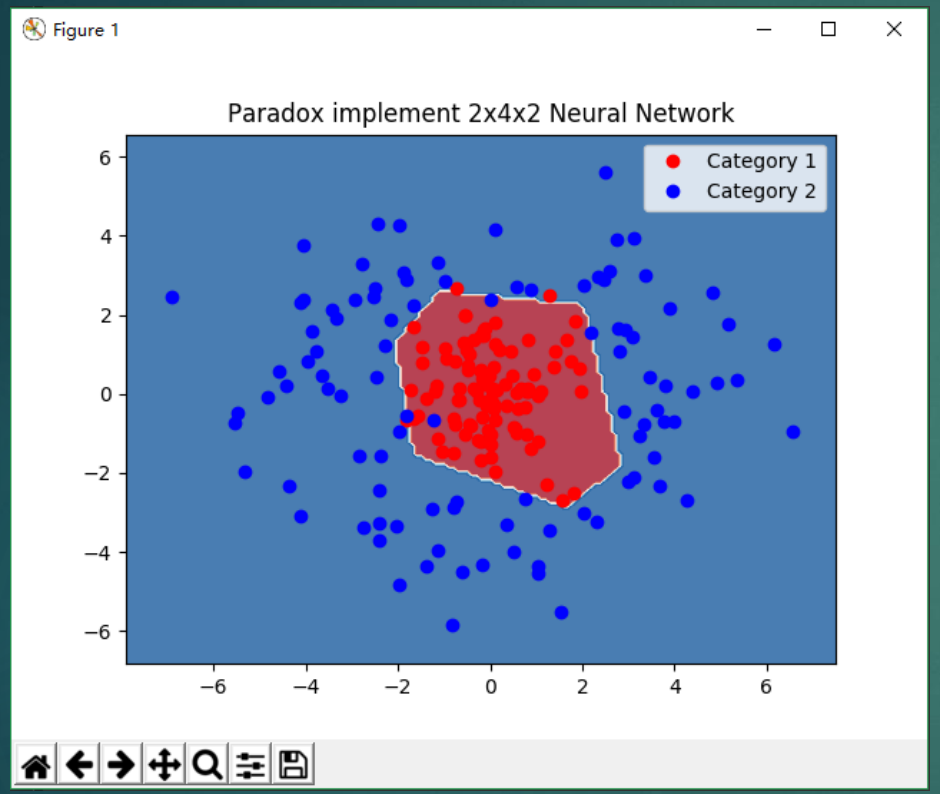

```python
import numpy as np
import matplotlib.pyplot as plt
import paradox as pd

# 随机生成点的个数。
points_sum = 200

x_data = []
y_data = []

# 生成y = 2 * x + 1直线附近的随机点。
for _ in range(points_sum):
    x = np.random.normal(0, 2)
    y = 2 * x + 1 + np.random.normal(0, 2)
    x_data.append(x)
    y_data.append(y)
x_np = np.array(x_data)
y_np = np.array(y_data)

# 定义符号。
X = pd.Constant(x_np, name='x')
Y = pd.Constant(y_np, name='y')
w = pd.Variable(0, name='w')
b = pd.Variable(0, name='b')

# 使用最小二乘误差。
loss = pd.reduce_mean((w * X + b - Y) ** 2)

# 创建loss计算引擎，申明变量为w和b。
loss_engine = pd.Engine(loss, [w, b])

# 梯度下降optimizer。
optimizer = pd.GradientDescentOptimizer(0.00005)

# 迭代100次最小化loss。
for epoch in range(1000):
    optimizer.minimize(loss_engine)
    loss_value = loss_engine.value()
    print('loss = {:.8f}'.format(loss_value))

# 获取w和b的训练值。
w_value = pd.Engine(w).value()
b_value = pd.Engine(b).value()

# 绘制图像。
plt.title('Paradox implement Linear Regression')
plt.plot(x_data, y_data, 'ro', label='Data')
plt.plot(x_data, w_value * x_data + b_value, label='Regression')
plt.legend()
plt.show()
```

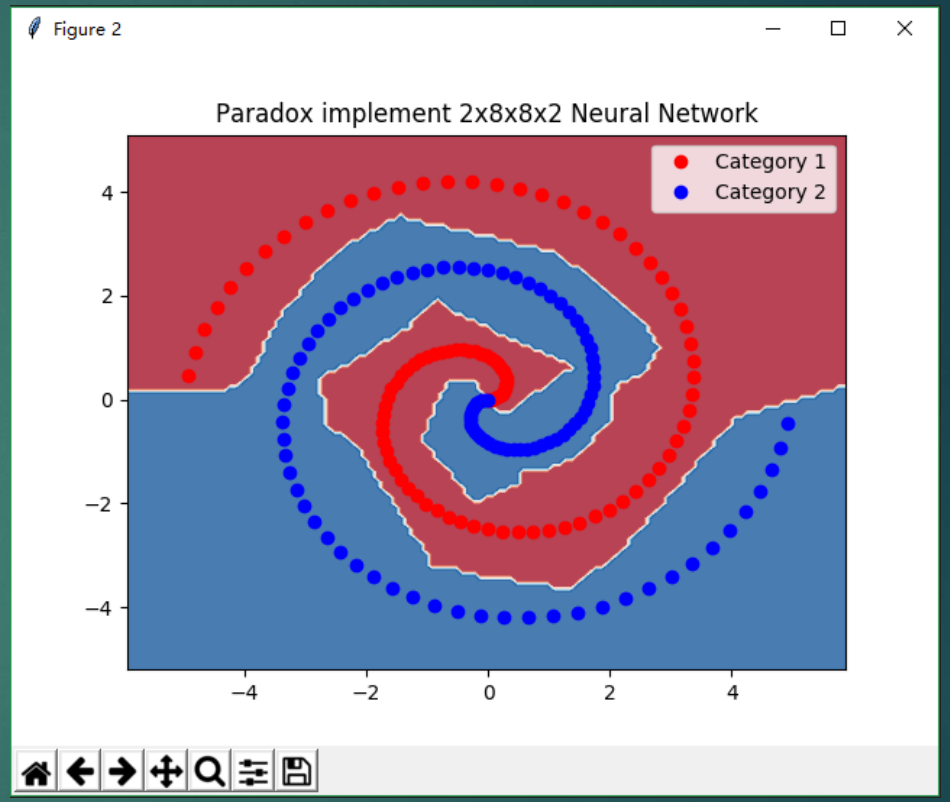
示例



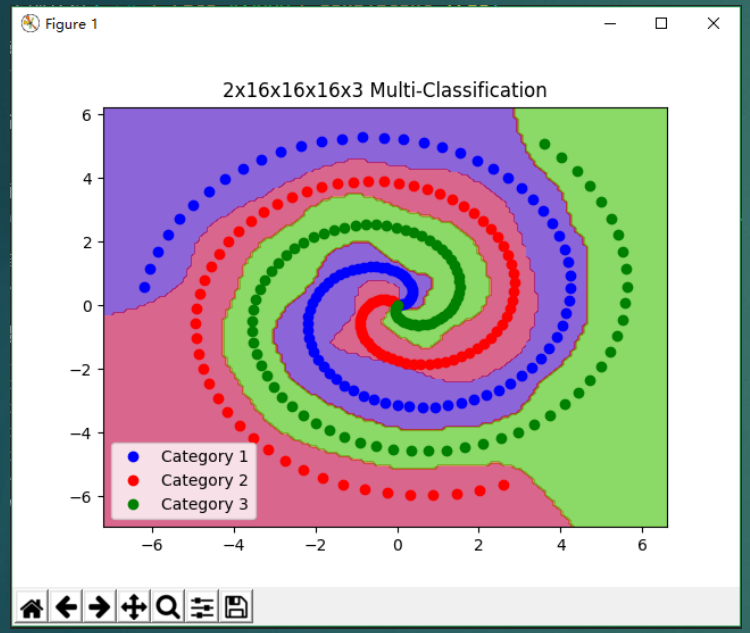

```python
import numpy as np
import matplotlib.pyplot as plt
import paradox as pd

# 每类随机生成点的个数。
points_sum = 100

c1_x = []
c1_y = []
c2_x = []
c2_y = []

# 分别在(0, 0)点附近和(8, 8)点附近生成2类随机数据。
for _ in range(points_sum):
    c1_x.append(np.random.normal(0, 2))
    c1_y.append(np.random.normal(0, 2))
    c2_x.append(np.random.normal(8, 2))
    c2_y.append(np.random.normal(8, 2))

# 定义符号。
c1 = pd.Constant([c1_x, c1_y], name='c1')
c2 = pd.Constant([c2_x, c2_y], name='c2')
W = pd.Variable([[1, 1], [1, 1]], name='w')
B = pd.Variable([[1], [1]], name='b')

# 定义SVM loss函数。
loss = pd.reduce_mean(pd.maximum(0, [[1, -1]] @ (W @ c1 + B) + 1) + pd.maximum(0, [[-1, 1]] @ (W @ c2 + B) + 1))

# 创建loss计算引擎，申明变量为W和B。
loss_engine = pd.Engine(loss, [W, B])

# 创建梯度下降optimizer。
optimizer = pd.GradientDescentOptimizer(0.01)

# 迭代至多1000次最小化loss。
for epoch in range(1000):
    optimizer.minimize(loss_engine)
    loss_value = loss_engine.value()
    print('loss = {:.8f}'.format(loss_value))
    if loss_value < 0.001:  # loss阈值。
        break

# 获取W和B的训练结果。
w_data = pd.Engine(W).value()
b_data = pd.Engine(B).value()

# 计算分类直线的斜率和截距。
k = (w_data[1, 0] - w_data[0, 0]) / (w_data[0, 1] - w_data[1, 1])
b = (b_data[1, 0] - b_data[0, 0]) / (w_data[0, 1] - w_data[1, 1])

# 绘制图像。
plt.title('Paradox implement Linear SVM')
plt.plot(c1_x, c1_y, 'ro', label='Category 1')
plt.plot(c2_x, c2_y, 'bo', label='Category 2')
plt.plot([-5, 15], k * np.array([-5, 15]) + b, 'y', label='SVM')
plt.legend()
plt.show()
```

示例



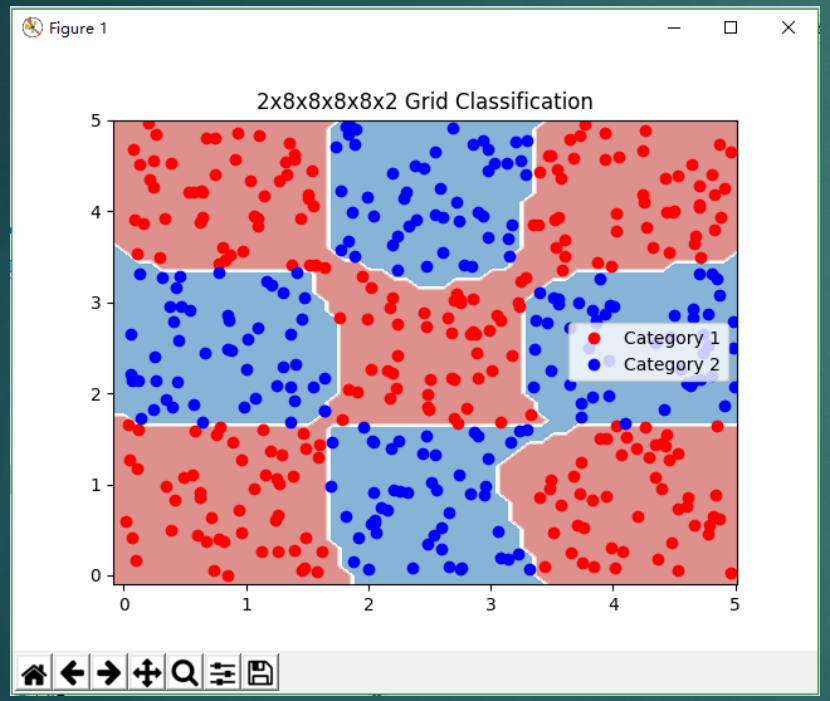


示例




```python
import numpy as np
import matplotlib.pyplot as plt
import paradox as pd

# 每类随机生成点的个数。
points_sum = 100

# 调用paradox的数据生成器生成三螺旋的3类数据。
data = pd.data.helical_2d(points_sum, 3, max_radius=2*np.pi)

# 组合数据。
c_x = data[0][0] + data[1][0] + data[2][0]
c_y = data[0][1] + data[1][1] + data[2][1]

# 定义每个点的分类类别。
classification = pd.utils.generate_label_matrix([0] * len(data[0][0]) + [1] * len(data[1][0]) + [2] * len(data[2][0]))[0]

# 调用高层API生成2x16x16x16x3的网络
model = pd.nn.Network()
model.add(pd.nn.Dense(16, input_dimension=2))  # 2维输入8维输出的全连接层。
model.add(pd.nn.Activation('tanh'))  # 使用tanh激活函数。
model.add(pd.nn.Dense(16))
model.add(pd.nn.Activation('tanh'))
model.add(pd.nn.Dense(16))
model.add(pd.nn.Activation('tanh'))
model.add(pd.nn.Dense(3))
model.add(pd.nn.Activation('tanh'))
model.loss('softmax')  # 使用softmax loss。

# 使用梯度下降优化器，使用一致性update大幅提升性能。
model.optimizer('gd', rate=0.003, consistent=True)

# 执行训练。
model.train(np.array([c_x, c_y]).transpose(), classification, epochs=10000)

# 设置网格密度为0.1。
h = 0.1

# 生成预测采样点网格。
x, y = np.meshgrid(np.arange(np.min(c_x) - 1, np.max(c_x) + 1, h), np.arange(np.min(c_y) - 1, np.max(c_y) + 1, h))

# 生成采样点预测值。
z = model.predict(np.array([x.ravel(), y.ravel()]).transpose()).argmax(axis=1).reshape(x.shape)

# 绘制图像。
plt.title('2x16x16x16x3 Multi-Classification')
plt.plot(data[0][0], data[0][1], 'bo', label='Category 1')
plt.plot(data[1][0], data[1][1], 'ro', label='Category 2')
plt.plot(data[2][0], data[2][1], 'go', label='Category 3')
plt.contourf(x, y, z, 3, cmap='brg', alpha=.6)
plt.legend()
plt.show()
```

示例




```python
import numpy as np
import matplotlib.pyplot as plt
import paradox as pd

# 每类随机生成点的个数。
points_sum = 50

# 调用paradox的数据生成器生成3x3网格状数据。
data = pd.data.grid_2d(points_sum, raw=3, column=3)

# 组合数据。
c_x = data[0][0] + data[1][0]
c_y = data[0][1] + data[1][1]

# 定义每个点的分类类别。
classification = pd.utils.generate_label_matrix([0] * len(data[0][0]) + [1] * len(data[1][0]))[0]

# 调用高层API生成2x8x8x8x8x2的网络，5层网络。
model = pd.nn.Network()
model.add(pd.nn.Dense(8, input_dimension=2))  # 2维输入8维输出的全连接层。
model.add(pd.nn.Activation('tanh'))  # 使用tanh激活函数。
model.add(pd.nn.Dense(8))
model.add(pd.nn.Activation('tanh'))
model.add(pd.nn.Dense(8))
model.add(pd.nn.Activation('tanh'))
model.add(pd.nn.Dense(8))
model.add(pd.nn.Activation('tanh'))
model.add(pd.nn.Dense(2))
model.add(pd.nn.Activation('tanh'))
model.loss('softmax')  # 使用softmax loss。

# 使用梯度下降优化器。
model.optimizer('gd', rate=0.0003)

# 执行训练。
model.train(np.array([c_x, c_y]).transpose(), classification, epochs=30000)

# 设置网格密度为0.1。
h = 0.1

# 生成预测采样点网格。
x, y = np.meshgrid(np.arange(np.min(c_x) - .1, np.max(c_x) + .1, h), np.arange(np.min(c_y) - .1, np.max(c_y) + .1, h))

# 生成采样点预测值。
z = model.predict(np.array([x.ravel(), y.ravel()]).transpose()).argmax(axis=1).reshape(x.shape)

# 绘制图像。
plt.title('2x8x8x8x8x2 Grid Classification')
plt.plot(data[0][0], data[0][1], 'ro', label='Category 1')
plt.plot(data[1][0], data[1][1], 'bo', label='Category 2')
plt.contourf(x, y, z, 2, cmap='RdBu', alpha=.6)
plt.legend()
plt.show()
```

Paradox by ictxiangxin

Thank you!