神经网络入门和提高

吴先超

2017.10.01

谢谢大家

从何处来，到何处去？

- 从我个人的想法来看，希望的是“扶上马，送一程”。目前的想法主要是：
- 深度神经网络入门和提高，
- 经典神经网络（cnn, rnn等）的实现和理解，
- 基于开源平台的分类和回归分析以及他们的应用（例如，tensorflow, theano, mxnet, caffee, chainer等），
- 面向图像/自然语言处理等具体领域的分类模型和生成模型（例如encoder-decoder; GANs）等。
- 越往后，估计越精细，到时候可以分流，让我们协会不同的专家来cover。

从入门到~~放弃~~->提高

- 万事开头难；
- 希望通过3到5次课的时间，
- 让大家彻底入门，注重实战（代码方面）和理论的结合，
- 等把一些基本概念和思路搞定之后，大家就可以自主上路放飞梦想了

预习了没?

安装了没?

源代码下载了没?

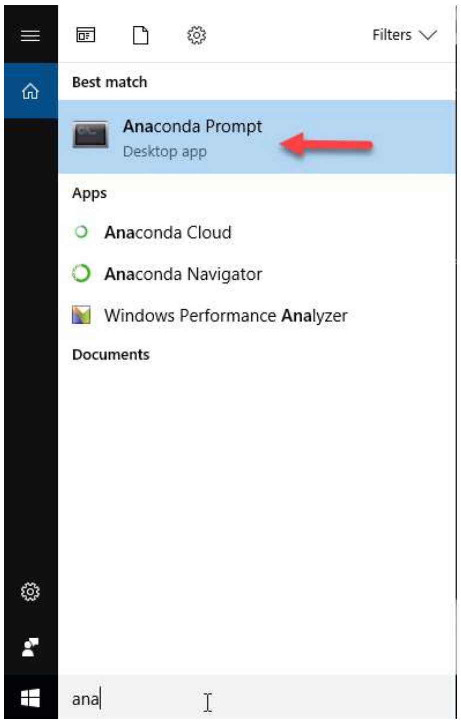
Me:各位亲，我要开始放代码了，为了push大家自己敲一遍代码，我会用截屏的方式放代码，权当一次预习吧，恳请大家批评指正

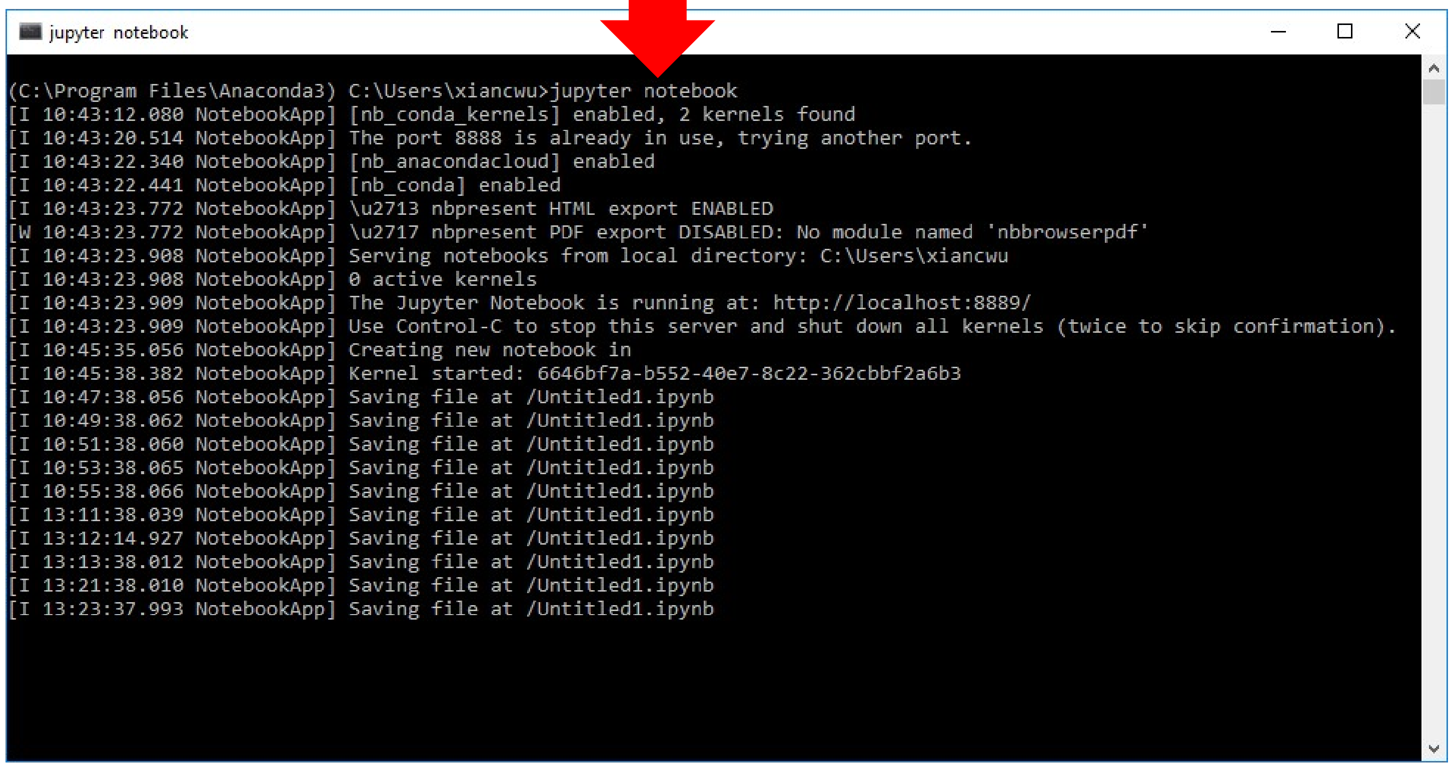
Me:三个小时的讲授时间其实非常短，因为我想让大家当堂练习（我一般不咋相信很多人在课后会练习。。。我也一样，哈哈），所以这里会提前预热一下

Me: https://www.continuum.io/downloads 安装（原则上只需要安装这一个就ok，里面封装了python和课程中我们用到的所有的库）

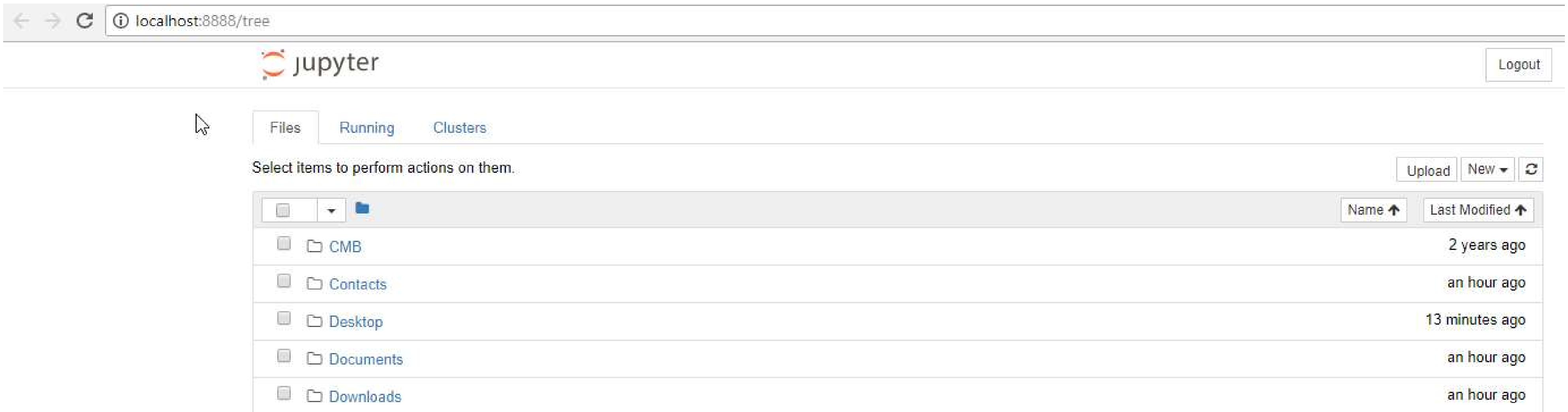
https://www.continuum.io/downloads

启动 anaconda prompt
输入 jupyter notebook



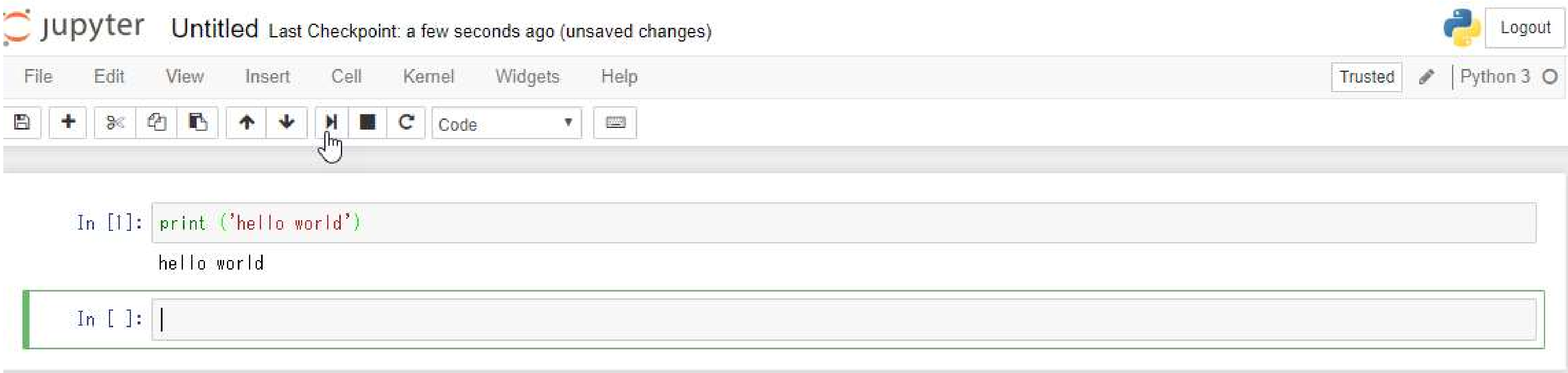


- 一个浏览器的界面会被自动启动，请选择“New”-> "Python [conda root]"意思是新创建一个小项目（one task or alike it）


localhost:8888/tree
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Name
Last Modified
CMB
2 years ago
Contacts
an hour ago
Desktop
13 minutes ago
Documents
an hour ago
Downloads
an hour ago


localhost:8888/tree
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Notebook:
Python 3
Other:
Text File
Folder
Terminals Unavailable
CMB
Contacts
Desktop
Documents

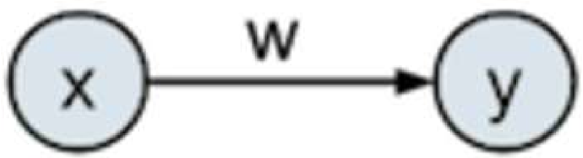
开始 – 课堂练习



- Shift + Enter -> 运行

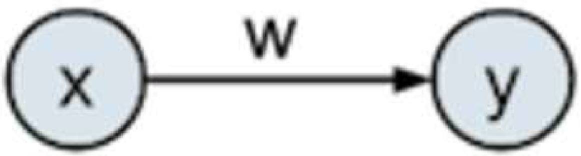
线性回归 linear regression

一个函数 = 一个网络



- x 一个输入变量
- w 一个参数
- y 一个输出变量
- 例如：y = 2 * x

一个值预测函数 = 一个回归网络



- x 一个输入变量
- w 一个参数
- y 一个预测输出变量
- t 一个实际参考答案
- 例如：y = 2 * x

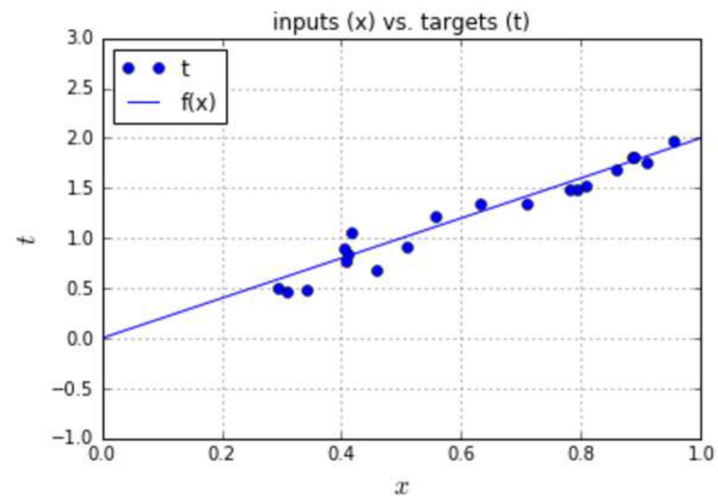
最小化损失函数
Squared error loss function

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \|t_i - y_i\|^2 .$$

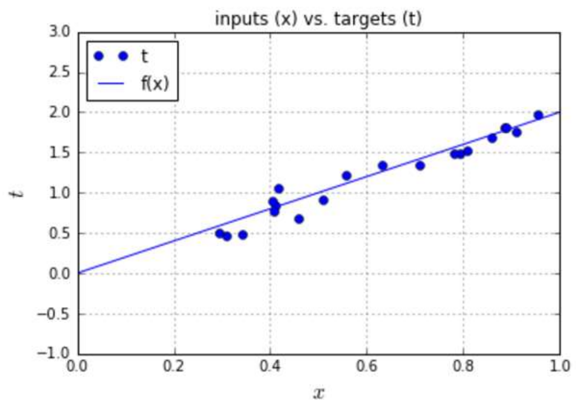
$$\mathbf{y} = \mathbf{x} * w,$$

构造20个点，加高斯分布噪音（0, 0.2）

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.random.uniform(0, 1, 20)

def f(x):
    return x * 2

noise_variance = 0.2

noise = np.random.randn(x.shape[0]) * noise_variance

t = f(x) + noise

plt.plot(x, t, 'o', label = 't')
plt.plot([0, 1], [f(0), f(1)], 'b-', label='f(x)')
plt.xlabel('$x$', fontsize=15)
plt.ylabel('$t$', fontsize=15)
plt.ylim([-1.0, 3])
plt.title('inputs (x) vs. targets (t)')
plt.grid()
plt.legend(loc=2)
plt.show()
```



课堂练习

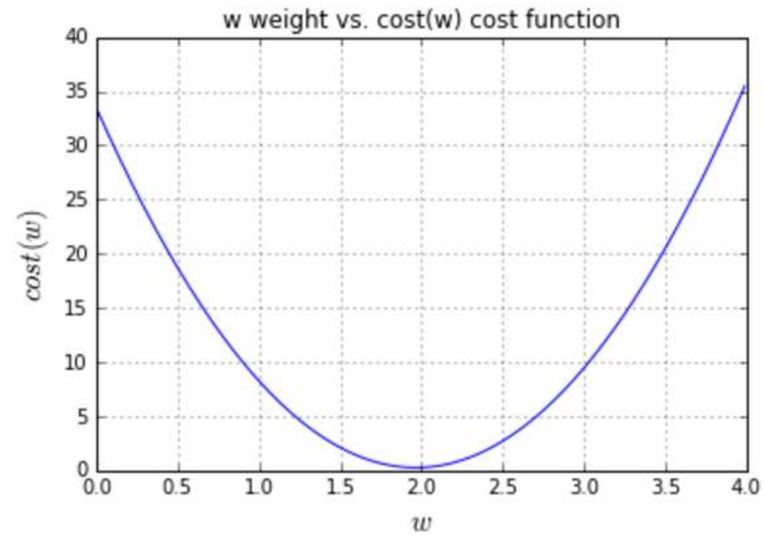



```python
x = np.random.uniform(0, 1, 20)
def f(x):
    return x * 2

noise = np.random.randn(x.shape[0]) * 0.2
t = f(x) + noise

plt.plot(x, t, 'o', label = 't')
plt.plot([0, 1], [f(0), f(1)], 'b-', label='f(x)')
plt.xlabel('$x$')
plt.ylabel('$t$')
plt.ylim([-1.0, 3])
plt.legend(loc=2)
plt.show()
```

参数和损失函数之间的关系

```
def nn(x, w):
    return x * w

def cost(y, t):
    return ((t - y) ** 2).sum()

def gradient(w, x, t):
    return 2 * x * (nn(x, w) - t)

def delta_w(w_k, x, t, learning_rate):
    return learning_rate * gradient(w_k, x, t).sum()

ww = np.arange(0, 4, 0.01)
costw = [cost(nn(x, ww1), t) for ww1 in ww]
plt.plot(ww, costw)
plt.xlabel('$w$', fontsize = 15)
plt.ylabel('$cost(w)$', fontsize = 15)
plt.title('w weight vs. cost(w) cost function')
plt.grid()
plt.show()
```

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \|t_i - y_i\|^2 .$$



y = w * x，求的是w

- 梯度下降法

$$w(k+1) = w(k) - \Delta w(k)$$

y = w * x，求的是w

- 梯度下降法 $w(k+1) = w(k) - \Delta w(k)$

最小化 损失函数
Squared error loss function

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \|t_i - y_i\|^2 .$$

$$\boxed{\text{loss function } \xi = \sum_{i=1}^{N} \|t_i - y_i\|^2}$$

$$\Delta w = \mu \frac{\partial \xi}{\partial w}$$

/ˈksaɪ/

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \|t_i - y_i\|^2 .$$

/ˈksaɪ/

$$\xi = \sum_{i=1}^{N} \|t_i - y_i\|^2$$

loss function for all N points

$$\frac{\partial \xi_i}{\partial y_i} = \frac{\partial (t_i - y_i)^2}{\partial y_i} = -2(t_i - y_i) = 2(y_i - t_i)$$

$$\mathbf{y} = \mathbf{x} * w,$$

loss function for i-th point

$$\boxed{\frac{\partial \xi_i}{\partial w} = \frac{\partial \xi_i}{\partial y_i} * \frac{\partial y_i}{\partial w}}$$

$$\frac{\partial y_i}{\partial w} = \frac{\partial (x_i * w)}{\partial w} = x_i$$

y 一个预测输出变量

t 一个实际参考答案

loss function for i-th point

$$\Delta w = \mu * \frac{\partial \xi_i}{\partial w} = \mu * 2x_i(y_i - t_i)$$

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \|t_i - y_i\|^2 . \qquad \xi = \sum_{i=1}^{N} \|t_i - y_i\|^2$$

$$\frac{\partial \xi_i}{\partial y_i} = \frac{\partial (t_i - y_i)^2}{\partial y_i} = -2(t_i - y_i) = 2(y_i - t_i)$$

```
def gradient(w, x, t):
    return 2 * x * (nn(x, w) - t)
```

$$\Delta w = \mu * \frac{\partial \xi_i}{\partial w} = \mu * 2x_i(y_i - t_i)$$

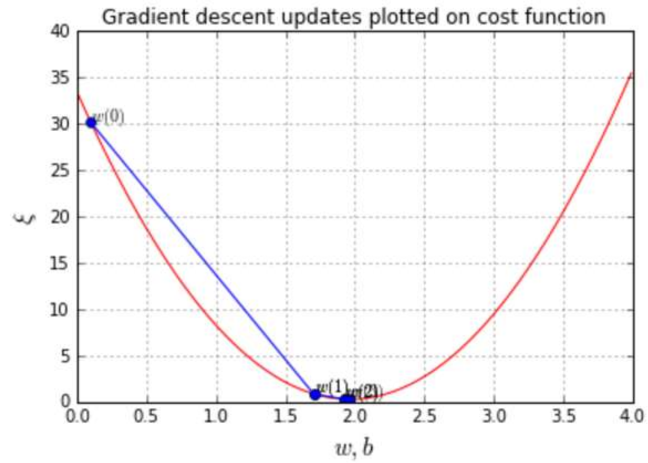
Learning rate 学习率

1. y 一个预测输出变量
2. t 一个实际参考答案
3. 寥寥几行代码。。。
4. w在代码中是怎么被更新的？（和公式一样吗？）

学习曲线-梯度下降直观展示

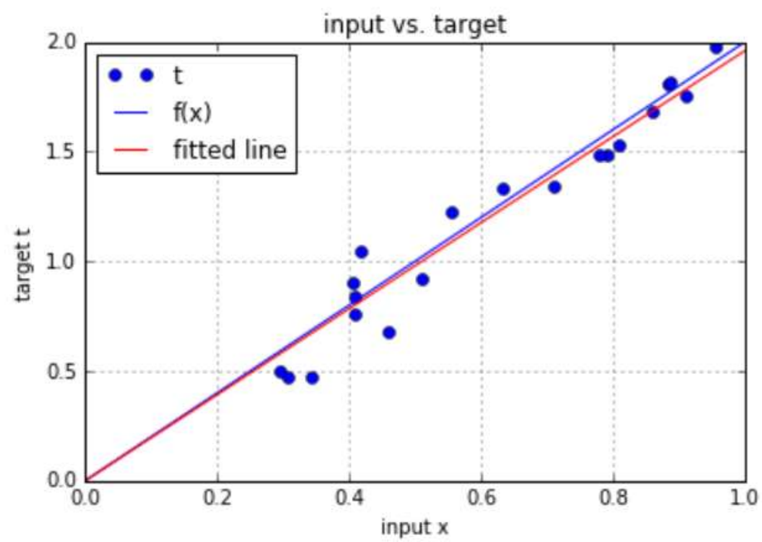
```python
# initial value of the weight w, a real value
learning_rate = 0.05

w = 0.1
nb_of_iterations = 3 # you can try 10, 20, 50, and 100 for example
w_cost = [(w, cost(nn(x, w), t))]

for i in range(nb_of_iterations):
    dw = delta_w(w, x, t, learning_rate)
    w = w - dw
    w_cost.append((w, cost(nn(x, w), t)))

for i in range(0, len(w_cost)):
    print('lr={} \t w({}): {:.6f} \t cost: {:.6f}'.
          format(learning_rate, i, w_cost[i][0], w_cost[i][1]))

costw = [cost(nn(x, ww1), t) for ww1 in ww]
plt.plot(ww, costw, 'r-')

for i in range(0, len(w_cost)-1):
    w1, c1 = w_cost[i]
    w2, c2 = w_cost[i+1]
    plt.plot(w1, c1, 'bo')
    plt.plot(w2, c2, 'bo')
    plt.plot([w1, w2], [c1, c2], 'b-')
    plt.text(w1, c1+0.3, '$w({})$'.format(i))
    plt.text(w2, c2+0.3, '$w({})$'.format(i+1))

plt.xlabel('$w, b$', fontsize=15)
plt.ylabel('$\\xi$', fontsize=15)
plt.title('Gradient descent updates plotted on cost function')
plt.grid()
plt.show()
```

```
lr=0.05      w(0): 0.100000      cost: 30.162236
lr=0.05      w(1): 1.703317      cost: 0.840337
lr=0.05      w(2): 1.928828      cost: 0.260257
lr=0.05      w(3): 1.960546      cost: 0.248782
```



Predicted line vs. golden/reference line

```
# for a relatively small loss
plt.plot(x, t, 'o', label='t')
plt.plot([0, 1], [f(0), f(1)], 'b-', label='f(x)')
plt.plot([0, 1], [0*w, 1*w], 'r-', label='fitted line')
plt.xlabel('input x')
plt.ylabel('target t')
plt.ylim([0, 2])
plt.title('input vs. target')
plt.grid()
plt.legend(loc=2)
plt.show()
```



小结：搞定了最简单的一个损失函数
squared error loss function

- 损失函数最小化 $$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} \|t_i - y_i\|^2 .$$
- 参数的梯度
- 使用梯度下降法，更新参数 $$\Delta w = \mu * \frac{\partial \xi_i}{\partial w} = \mu * 2x_i(y_i - t_i)$$

公式很复杂
代码很简单

回顾一下！

- 代码中
  - x, f(x), t以及w的维度(哪些是数组/向量，哪些是标量)？
  - sum()都出现在了哪里？是干啥的？
  - 看多少个点之后，更新w？为啥？

```python
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.random.uniform(0, 1, 20)
print ('x.shape={}, x={}'.format(x.shape, x))
def f(x):
    return x * 2

noise = np.random.randn(x.shape[0]) * 0.2
t = f(x) + noise
print ('f(x).shape={}, f(x)={}'.format(f(x).shape, f(x)))
print ('t.shape={}, t={}'.format(t.shape, t))

plt.plot(x, t, 'o', label = 't')
plt.plot([0, 1], [f(0), f(1)], 'b-', label='f(x)')
plt.xlabel('$x$')
plt.ylabel('$t$')
plt.ylim([-1.0, 3])
plt.legend(loc=2)
plt.show()
```

```
x.shape=(20,), x=[ 0.9160266   0.30956275  0.7194621   0.150296    0.95249883  0.91033576
  0.72598144  0.77837547  0.19979594  0.18999839  0.08115148  0.43221855
  0.52144169  0.55600951  0.12194787  0.51412905  0.23823495  0.27481218
  0.47571463  0.93043548]
f(x).shape=(20,), f(x)=[ 1.8320532   0.61912549  1.43892419  0.300592    1.90499765  1.82067151
  1.45196289  1.55675093  0.39959187  0.37999677  0.16230296  0.8644371
  1.04288338  1.11201901  0.24389573  1.02825809  0.47646991  0.54962436
  0.95142926  1.86087097]
t.shape=(20,), t=[ 1.76491729  0.6196392   1.44330872  0.40922835  1.59389195  1.61768216
  1.58493342  1.40237558  0.31827665  0.27312904  0.32600485  0.68206676
  1.05023831  1.30006583  0.29169434  0.89299819  0.64788436 -0.00610575
  0.83399293  1.80547422]
```

x, f(x), t以及w的维度(哪些是数组/向量，哪些是标量)？

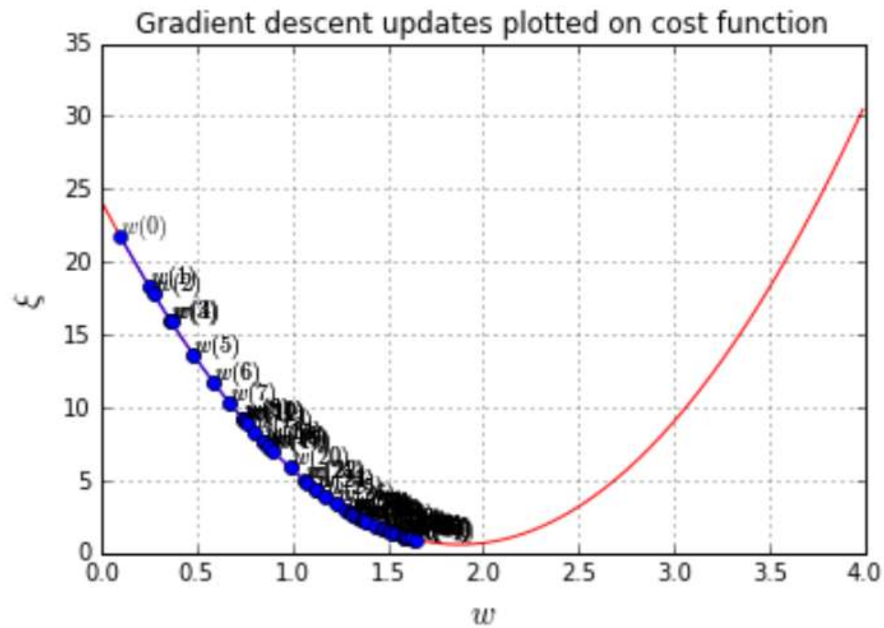
sum()都出现在了哪里？是干啥的？

```
def nn(x, w):
    return x * w

def cost(y, t):
    return ((t - y) ** 2).sum()

def gradient(w, x, t):
    return 2 * x * (nn(x, w) - t)

def delta_w(w_k, x, t, learning_rate):
    return learning_rate * gradient(w_k, x, t).sum()
```

看多少个点之后，更新w？为啥？(mini batch vs. perceptron)

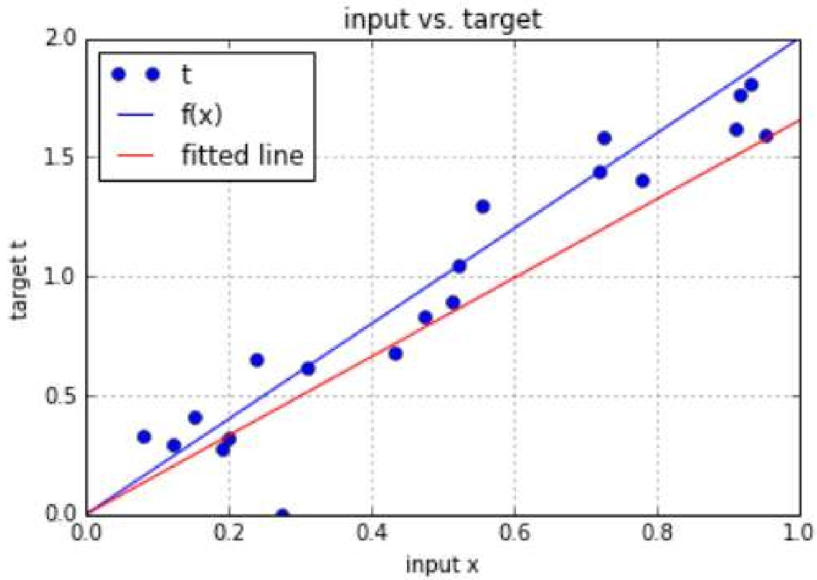
如果想修改w的更新的时机？

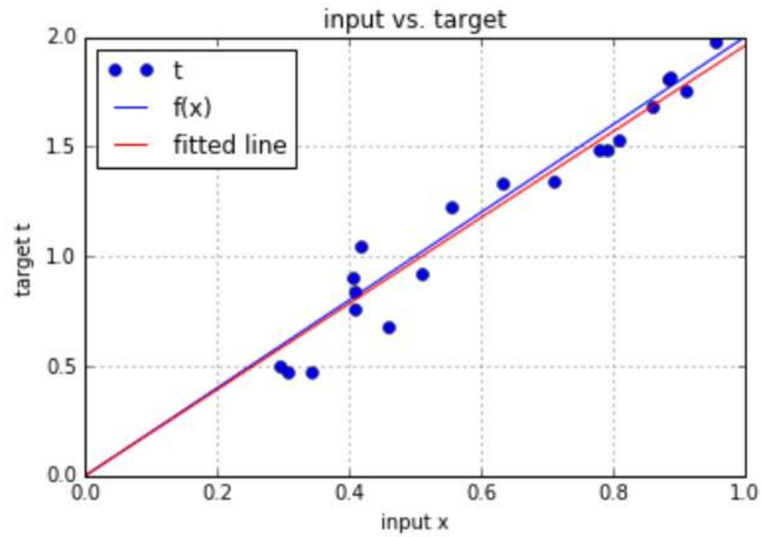
```python
#for i in range(nb_of_iterations):
#    dw = delta_w(w, x, t, learning_rate)
#    w = w - dw
#    w_cost.append((w, cost(nn(x, w), t)))

for i in range(nb_of_iterations):
    for j in range(x.shape[0]):
        xi = x[j]
        ti = t[j]
        dw = delta_w(w, xi, ti, learning_rate)
        w = w - dw
        w_cost.append((w, cost(nn(x, w), t)))
```



```python
# for a relatively small loss
plt.plot(x, t, 'o', label='t')
plt.plot([0, 1], [f(0), f(1)], 'b-', label='f(x)')
plt.plot([0, 1], [0*w, 1*w], 'r-', label='fitted line')
plt.xlabel('input x')
plt.ylabel('target t')
plt.ylim([0, 2])
plt.title('input vs. target')
plt.grid()
plt.legend(loc=2)
plt.show()
```

结果变差了，为啥?





- 待续。。。