神经网络基础（NN）

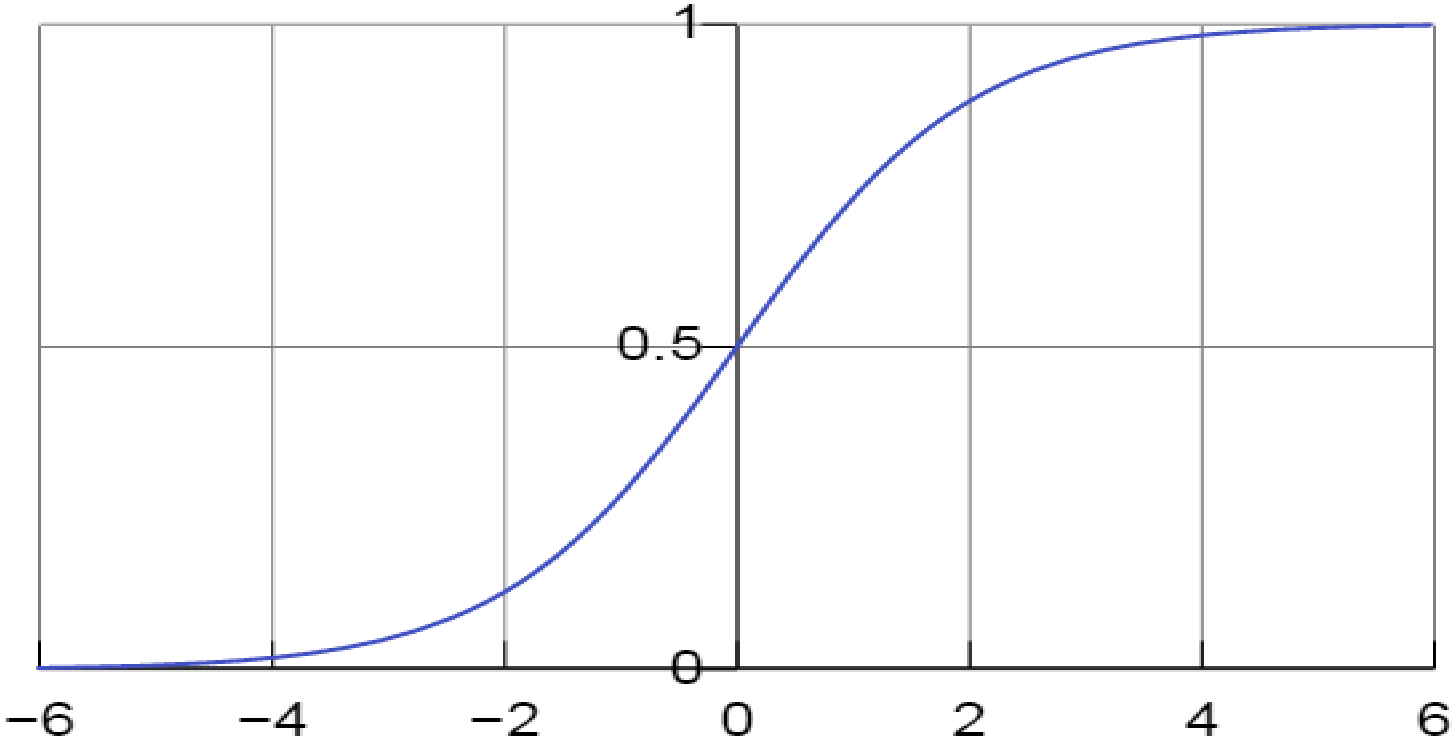
胡俊峰
北京大学信息科学技术学院
2020/11/25

hujf@pku.edu.cn

Sigmoid函数

$$g(x) = \frac{1}{1+e^{-x}}$$

将线性回归值变换到(0, 1)，将其理解为x对应的y为1的概率



分类问题转化为参数回归问题

$$\alpha(text) = \log P(text|spam) + \log P(spam) - \log P(text|nonspam) - \log P(nonspa$$

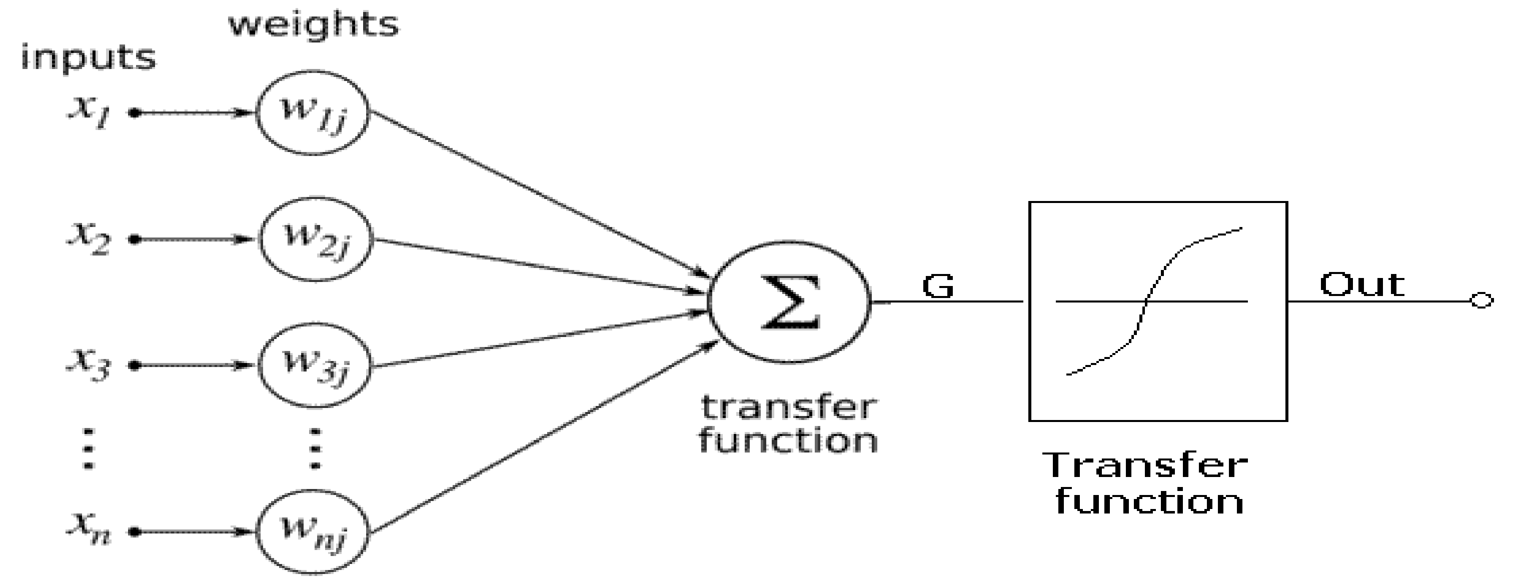
$$= x_1 \log P(w_1|spam) + \ldots + x_{|V|} \log P(w_{|V|}|spam) + \log P(spam)$$

$-\ldots$(对应的nonspam的项)

(x_i表示词典中第i个词在text中出现的次数)

$$= k_0 + x_1 k_1 + x_2 k_2 + \ldots x_{|V|} k_{|V|}$$

基于特定的损失函数回归一组系数



Install PYTORCH

Get Started.

Select your preferences, then run the PyTorch install command.

Please ensure that you are on the latest pip and numpy packages.
Anaconda is our recommended package manager

OS	Linux	OSX	
Package Manager	conda	pip	Source
Python	2.7	3.5	3.6
CUDA	7.5	8.0	None

Run this command:

```
pip3 install http://download.pytorch.org/whl/torch-0.2.0.post3-cp36-cp36m-macosx_10_7_x86_64.whl
pip3 install torchvision
# OSX Binaries dont support CUDA, install from source if CUDA is needed
```

http://pytorch.org

Tensor的声明与初始化：

```python
import torch
x = torch.zeros(5, 3, dtype=torch.long)
print(x)
```

```
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

```python
x = torch.tensor([5.5, 3])  # use list ini
print(x)
```

```
tensor([5.5000, 3.0000])
```

Tensor的声明与初始化（续）：

```python
x = torch.empty(5, 3)
print(x)
y = torch.empty(5, 3)
y = torch.randn_like(x, dtype=torch.float)
print(y)
print(id(x))
```

```
tensor([[9.2755e-39, 9.1837e-39, 9.3674e-39],
        [1.0745e-38, 1.0653e-38, 9.5510e-39],
        [1.0561e-38, 1.0194e-38, 1.1112e-38],
        [1.0561e-38, 9.9184e-39, 1.0653e-38],
        [4.1327e-39, 1.0194e-38, 1.0469e-38]])
tensor([[-0.5130, -0.5909,  2.9082],
        [ 0.1144,  1.7929,  0.4377],
        [ 0.8158, -1.3595,  0.0530],
        [-1.1909,  1.1853, -0.2734],
        [-0.6974,  0.0031,  0.8806]])
2160412460040
```

```python
x = torch.rand(5, 3) # 会重新生成一个新的tensor对象
print(x)
print(id(x))
```

Tensor的运算：

In [12]:

```
y = torch.ones(5, 3, dtype=torch.int)
print(x + y)
# print(y.add_(x)) #result type Float can't be cast to the desired
print(x.add_(y))
```

```
tensor([[0.2586, 2.1962, 3.9954],
        [2.1969, 1.5400, 2.5352],
        [2.2736, 1.9837, 1.0773],
        [0.3881, 1.4794, 1.2122],
        [2.3094, 2.5322, 2.9014]])
tensor([[0.2586, 2.1962, 3.9954],
        [2.1969, 1.5400, 2.5352],
        [2.2736, 1.9837, 1.0773],
        [0.3881, 1.4794, 1.2122],
        [2.3094, 2.5322, 2.9014]])
```

切片与广播：

```
x[2:] = torch.tensor([1,2,3])    # 切片与广播
print(x)
```

```
tensor([[0.5116, 0.8722, 0.3706],
        [0.8005, 0.3415, 0.5838],
        [1.0000, 2.0000, 3.0000],
        [1.0000, 2.0000, 3.0000],
        [1.0000, 2.0000, 3.0000]])
```

```
y = torch.tensor([0,1,2])  # 切片与广播
print(x*y)
```

```
tensor([[0.0000, 0.8722, 0.7413],
        [0.0000, 0.3415, 1.1677],
        [0.0000, 2.0000, 6.0000],
        [0.0000, 2.0000, 6.0000],
        [0.0000, 2.0000, 6.0000]])
```

Torch tensor与Numpy ndarray：

```python
# Converting a Torch Tensor to a NumPy
a = torch.ones(5,dtype=torch.int)
print(a)
b = a.numpy()
print(b)
b += 1                                  # 与a共享数据定义, b += 1.0 会有type cast error
print(a)
a[3]= 6
print(b)
```

```
tensor([1, 1, 1, 1, 1], dtype=torch.int32)
[1 1 1 1 1]
tensor([2, 2, 2, 2, 2], dtype=torch.int32)
[2 2 2 6 2]
```

```python
b = b + 1.0                             # 赋值语句生成新的对象, 类型转换自动升级
print(b)
print(a)
```

```
[3. 3. 3. 7. 3.]
tensor([2, 2, 2, 6, 2], dtype=torch.int32)
```

在python中，对向量直接进行计算会比for循环逐元素计算快得多

```python
c = torch.zeros(n)
timer = Timer()
for i in range(n):
    c[i] = a[i] + b[i]
f'{timer.stop():.5f} sec'
```

0.13640 sec

```python
timer.start()
d = a + b
f'{timer.stop():.5f} sec'
```

0.00029 sec

使用GPU加速

```python
# let us run this cell only if CUDA is available
# We will use ``torch.device`` objects to move tensors in and out of GPU
if torch.cuda.is_available():
    device = torch.device("cuda")          # a CUDA device object
    y = torch.ones_like(x, device=device)  # directly create a tensor on GPU
    x = x.to(device)                       # or just use strings ``.to("cuda")``
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # ``.to`` can also change dtype together!
else:
    print("No cuda available.")
```

```
tensor([[1.2586, 3.1962, 4.9954],
        [3.1969, 2.5400, 3.5352],
        [3.2736, 2.9837, 2.0773],
        [1.3881, 2.4794, 2.2122],
        [3.3094, 3.5322, 3.9014]], device='cuda:0')
tensor([[1.2586, 3.1962, 4.9954],
        [3.1969, 2.5400, 3.5352],
        [3.2736, 2.9837, 2.0773],
        [1.3881, 2.4794, 2.2122],
        [3.3094, 3.5322, 3.9014]], dtype=torch.float64)
```

Autograd，NN模型回归机制

```
#Create a tensor and set requires_grad=True to track computation with it:

x = torch.ones(2, 2, requires_grad=True)
print(x)
y = x + 1
print(y)
z = y * 2
print(z)
```

```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
tensor([[2., 2.],
        [2., 2.]], grad_fn=<AddBackward0>)
tensor([[4., 4.],
        [4., 4.]], grad_fn=<MulBackward0>)
```

```
out = z.mean()
print(out)
out.backward()
print(x.grad)
#print(y.grad)
```

```
tensor(4., grad_fn=<MeanBackward0>)
tensor([[0.5000, 0.5000],
        [0.5000, 0.5000]])
```

Dive into Deep Learning

Chapter 2 Preliminaries by

2.1 Data Manipulation

NumPy (array)

- Operations

  Element-wise（对每个元素）: +, -, *, /, **, exp()

  reshape(), sum()

- Broadcasting

  以复制数据的方式扩展array，使得它们的shape相同

- Indexing and slicing

  e.g. x[1:3], x[-1], x[1, 2], x[0:2, :]

- Saving Memory

  X = X + Y（赋值后X被分配新的内存空间）

  -> X[:] = X + Y 或 X += Y （X仍使用原来的内存空间）

2.2 Data Preprocessing

Pandas

- 读取csv文件
  - pd.read_csv()
  - 数据切片： iloc[: , 0 : 2]
- 处理缺失数据 NaN
  - Impulation: 用替代值填补， fillna()， get_dummies()
- 转换成tensor
  - np.array()

	NumRooms	Alley	Price
0	NaN	Pave	127500
1	2.0	NaN	106000
2	4.0	NaN	178100
3	NaN	NaN	140000

	NumRooms	Alley
0	3.0	Pave
1	2.0	NaN
2	4.0	NaN
3	3.0	NaN

	NumRooms	Alley_Pave	Alley_nan
0	3.0	1	0
1	2.0	0	1
2	4.0	0	1
3	3.0	0	1

- 查看数据： shape()， head()， info()
- 去重： duplicated()
- PyTorch数据加载与预处理：参考官方文档 https://pytorch.org/docs/stable/data.html#module-torch.utils.data

2.3 Linear Algebra

Scalars， vectors， matrices， tensor

- Length
- Dimension：对于vector或者axis，指长度；对于tensor，指axis的数量
- Shape：tuple，各个axis上的长度

- 运算
  - Hadamard积：相同位置相乘（element-wise）
  - 与标量运算：每一个元素都与标量进行运算
  - sum()：可以指定坐标轴
  - 向量点积：两个向量做element-wise的乘法再求和
  - 范数：度量一个向量的大小
    - 满足正齐次性，三角不等式，非负性，正定性
    - L1范数：绝对值之和，L2范数：欧式距离，F范数：平方和开根号
    - Lp范数公式：

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$

2.4 Calculus

- 函数求导，复合函数求导规则
- 求偏导，梯度

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^\top$$

- 常用公式：
  - For all $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\nabla_{\mathbf{x}} \mathbf{A}\mathbf{x} = \mathbf{A}^\top$,
  - For all $\mathbf{A} \in \mathbb{R}^{n\times m}$, $\nabla_{\mathbf{x}} \mathbf{x}^\top \mathbf{A} = \mathbf{A}$,
  - For all $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\nabla_{\mathbf{x}} \mathbf{x}^\top \mathbf{A}\mathbf{x} = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$,
  - $\nabla_{\mathbf{x}} \|\mathbf{x}\|^2 = \nabla_{\mathbf{x}} \mathbf{x}^\top \mathbf{x} = 2\mathbf{x}$.
- 链式法则：

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

$$\frac{dy}{dx_i} = \frac{dy}{du_1}\frac{du_1}{dx_i} + \frac{dy}{du_2}\frac{du_2}{dx_i} + \cdots + \frac{dy}{du_m}\frac{du_m}{dx_i}$$

待续...