Neural Networks and Deep Learning

Summer School
Kunag yaming Honors School
吴胜俊教授

- Textbook: Michael A. Nielsen "Neural NEtworks and Deep Learning
- Online version: 链接

Contents:

Introduction

- 解决的问题:
  - 图像识别
  - 声音识别
  - 自然语言处理
  - etc
- 课程例子:
  - 手写数字的识别:
    - 对于人类easy
    - 对于rule-based 算法: hard;
  - 解决的方法:
    - Segmentation: 分段,切片;
    - 分类;
    - Classifying individual digit:
      - 输入数据集:$[0.0, 1.0]^{28\times 28}$

        - 函数f(进行分类);
        - 输出数据集:$\{0, 1...., 9\}$ (Class labels)
    - Segmentation 问题可能的方法;
        - 试图分类
        - 用一个individual digit classfier 对上面的分类进行打分
            - 高分,低分反馈,然后继续分类;
        - 得到最高分的分类

神经网络

- 定义: 神经网络是一个**神经元**的网络。主要有：输入层,隐藏层(可以有很多很多层)，输出层;
- 学习目标:
    - 神经元的强度
    - 通过输入数据神经网络来调整神经元的强度,这就是学习/训练的过程
- Perceptrons(感知机)
    - **感知机**是一种抽象化(人造)的神经元;
    - 根据输入的X得到二进制的输出;

$$f(x) = \begin{cases} 0 & if \sum_j w_j x_j \leq \text{阈值} \\ 1 & if \sum_j w_j x_j > \text{阈值} \end{cases}$$

    - 符号规约:
        - $\sum_j w_j x_j$写成$w \cdot x$
        - 取b=-threshold

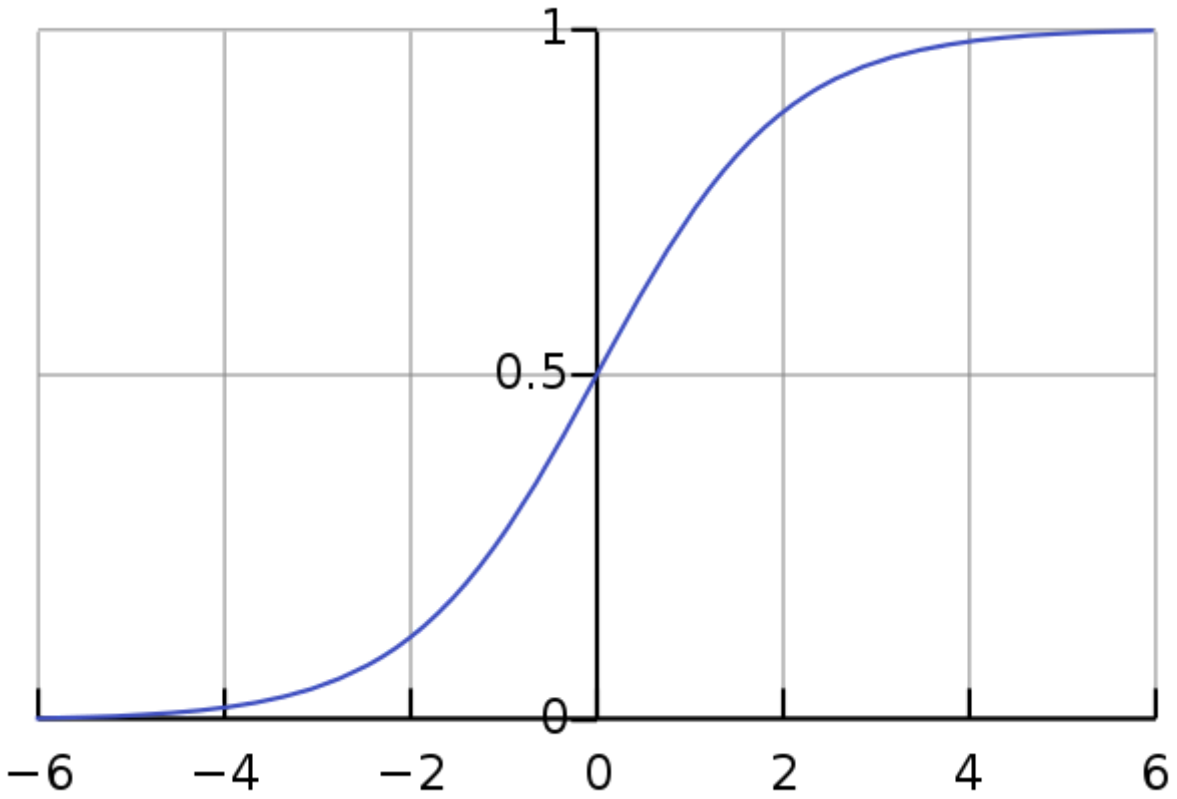
$$f(x) = \begin{cases} 0 & if \sum_j w \cdot x + b \leq 0 \\ 1 & if \sum_j w \cdot x + b > 0 \end{cases}$$

    - 实现NAND(↑)gate (非或)
        - e.g, w=$(-2, -2)^T$
        - Bias: b=3

A	B	A NAND B
0	0	1

A	B	A NAND B
0	1	1
1	0	1
1	1	0

- Sigmoid 神经元;
  - $w+\Delta w$
  - 之前的感知器不连续,会变化过于剧烈
  - Sigmoid neuron:
    output=$\sigma(w\cdot x+b)$
    sigmoid function:$\sigma(z)=\frac{1}{1+e^{-z}}$
  - sigmoid 是0-1的一个连续函数;



  - 反向传播的时候很香,求导简单;
  - $$\Delta output \approx \sum \frac{\partial\, output}{\partial w_j}\Delta w_j+\frac{\partial\, output}{\partial b}\Delta b$$
  - $\sigma(z)'=\frac{e^{-z}}{(1+e^{-z})^2}$
- Feedforward neural networks
  No loops;

- Why 10 ouotput nuerons are needed?
  用十个比较离散的输出有利于:1. 比较 2.结果理解性比较强;
- 梯度下降法学习:
  - Cost function:当前的神经网络输出的结果有多好;

$$C(w,b)=\frac{1}{2n}\sum_{n}||y(x)-a||^2$$

    - w:network中的权重集合
    - b:所有的bias集合;
    - n:训练集的个数;
    - x: 输入的vector;
    - y=y(x): 输出的向量;
    - a=a(w,b,x):network这一轮训练的结果;
  - C(w,b)是一个非负的二次函数'
  - 梯度下降:
    - Minimize C(v):

$$\Delta C \approx \bigtriangledown C \cdot \Delta v$$

    - $\bigtriangledown C[\frac{\partial C}{\partial v_1}...\frac{\partial C}{\partial v_m}]$(梯度的方向)
    - $\Delta=-\eta \bigtriangledown C$
    - η是学习率;
  - 因此更新规则:
    - w $\leftarrow$ w-$\eta \bigtriangledown_w C$
    - b $\leftarrow$ w-$\eta \bigtriangledown_b C$
  - SGD:随机梯度下降;
  - 怎么算$\bigtriangledown_w$:反向传播算法(BP)
- SGD:
  - 随机从训练集里面选择一部分数据进行$\bigtriangledown C$的训练;
  - Mini-batch: 就是用m个随机出来的数据训练;
  - Epoch(训练轮次)
- 反向传播算法;
  - 定义:
    - w^l_{jk}: the weight for the connection from the k-th neuron in the l-1 th layer to the j-th nueron in th lth layer
    - b^l_j: the bias of the j-th neuron in the l-th layer
    - a^l_j: the activation of the j-th neuron in the l-th layer
    - $$a^l_j=\sigma(\sum_k w^l_{jk}a^{l-1}_k+b^l_j);\sigma:sigmoid\ function$$

- 记$z^l = w^l a^{l-1} + b^l$
- $a^l = \sigma(z^l)$

- $$C = \frac{1}{n}\sum_x C_x$$

- Derive $\bigtriangledown_w C_x$ and $\bigtriangledown_b C_x$
  - all subscripts x of C_x are omitted;
  - error:第l层神经元的错误:

$$\delta_k^l = \frac{\partial C}{\partial z^l}$$

  - 某种神奇的乘法符号;

$$(s \odot t)_j = s_j t_j$$

- BP:
  1. the error in the output layer σ^L:

$$\delta_j^l = \frac{\partial C}{\partial a_j^L}\sigma^l(z_j^L)$$

$$\delta^L = \bigtriangledown_a C \odot \sigma'(z^L)$$

$$\delta^L = (a^L - y) \odot \sigma'(Z^L)$$

  2. the error σ^L in terms of the error in the next layer σ^{l+1}:

$$\sigma^l = ((w^{l+1})^T \sigma^{l+1}) \odot \sigma'(z^l)$$

     (用来递归求解);
  3. the rate of change of the cost with respect to any bias:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C^j}{\partial b} = \sigma$$

4. the rate of change of the cost with respect to ant weight :

$$\frac{\partial C}{\partial w_{jk}^{\mathrm{l}}} = a_k^{l-1}\sigma_j^l$$

$$\frac{\partial C}{\partial w} = a_{in}\sigma_{out}$$

- BP实际操作；
  1. Input x;
     对每一个input layer　设置corresponding activation a^l;
  2. Feedforward:
     For each l=3,...,L compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$
  3. Ouyput error σ^L
     计算vector $\sigma^L \bigtriangledown_a C \odot \sigma'(z^l)$
  4. Backpropagate the error;
     For each l=L-1,...,2 compute $\sigma^l = ((w^{l+1})^T\sigma^{l+1}) \odot \sigma'(z^l)$
  5. Output:
     THe gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \sigma_j^l$

- 一种实现的方法:
- MINST
  -traning data:50,000 images
  - test set: 10,000 images
- Size of the network:
  - 784,30,10
- Hyper-parameters
  - 30 epochs
  - mini-atch size: 10
  - learning rate:$\eta = 3.0$

Neural nets to compute any function

- the universality theorem
- 感知机的调戏方式:
  1. c 越大越接近阶跃
  2. -sc来调整位置
  3. 可以用两个神经元实现一个bump函数;(s_1, s_2之间取1,其余为0);

4. 一般的函数,有多少的分段就加多少的神经元
   - 可以康康这个网站

Improving the way neural nets learn

The cross-entyopy cost function

- Reason:
  - sigmoid神经元会梯度消失;
- 解决方式:
  1. 改善cost-function:

$$C = -\frac{1}{n}\sum_{x}[ylna + (1-y)ln(1-a)]$$

  2. 推导:
  - (看不懂)