Subject: Stanford CS229 Machine Learning, Lecture 8, Neural Networks 1

Date: from December 22, 2024 to December 23, 2024

Contents

CS229 Machine Learning, Neural Networks 1, 2022, Lecture 8

YouTube:Stanford CS229 Machine Learning, Neural Networks 1, 2022, Lecture 8

Outline

Outline

$$\text{Outline}\begin{cases}1. & \text{Supervised learning with non-linear model}\\ 2. & \text{Neural networks}\begin{cases}1. & \text{How to define } h_\theta(\boldsymbol{x})?\\ 2. & \text{How to compute } \nabla J^{(i)}(0)?\end{cases}\end{cases}$$

Review

Linear and Non-linear models

Data set: $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^n, \boldsymbol{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}; \quad h_\theta : \mathbb{R}^d \to \mathbb{R}$

Linear regression

Model: $h_\theta(\boldsymbol{x}) = \theta^T \boldsymbol{x} + b$ (linear)

Cost / Loss function: $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2$

Optimize[1]**:** run Gradient Decent(GD) or Stochastic Gradient Decent(SGD) to optimize

---

Non-linear model: Kernel method

Model: $h_\theta(\boldsymbol{x}) = \theta^T \phi(\boldsymbol{x})$ (linear in parameters, non-linear in $\boldsymbol{x}$)

可以看到即使是非现象模型核方法也只是对输入 $\boldsymbol{x}$ 非线性，但是对参数 θ 仍然是线形的，那么如果希望完全是非线性的模型该怎么办呢？例如 $h_\theta(x) = \sqrt{\theta_1^3 x_2 + \sqrt{\theta_5 x_4}}$.

[1] 详见附录A

Neural network

Neural network – Introduction

例如我们考虑房价预测问题，使用房屋面积线性预测房屋价格，如图1所示.

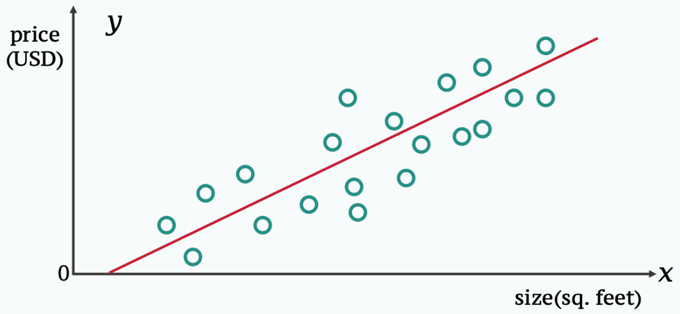

Figure 1: house price prediction

但是是时候会有两个明显的问题:

1. 房屋面积与价格可能并非简单的线性关系，而是有更加复杂的非线性关系
2. 线形模型可能导致某些面积下房屋价格为负，显然不合常理，图1就是这种情况

线性整流函数（Rectified Linear Unit, ReLU）事实上可以解决上述问题，

$$\mathrm{ReLU}(x) = \max\{0, x\}$$

因此显然ReLU 1. 是非线性函数[a]；2. 可以防止房价出现负数的情况.因此我们将输出写为

$$h_\theta(x) = \mathrm{ReLU}(\omega x + b) \tag{1}$$

在高维 $\boldsymbol{x} \in \mathbb{R}^d$ 情形下，写为:

$$h_\theta(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{\omega x} + b), \quad \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{\omega} \in \mathbb{R}^d, b \in \mathbb{R} \tag{2}$$

在神经网络(Neural networks)中，一个式(2)被称为一个神经元(neuron)，深度学习要做的是堆叠若干个、若干层这样的神经元，且上一层的输出就是下一层的输入. 此时$\boldsymbol{\omega x} + b$被称为预激活值(pre-activation)，其被激活函数作用后称为激活值(activation):

$$\text{output of activation} \rightarrow \text{input of the next neuron} \rightarrow \text{pre-activtion} \rightarrow \text{activation}$$

[a]非线形体现在"转折点"，事实上在深度学习中这就是激活函数(activation function)

Neural network – Example

以 $\boldsymbol{x} \in \mathbb{R}^4$，为例，其中 x_1: size, x_2: # bedrooms, x_3: zip code, x_4: wealth

除了这些特征外，我们可以根据经验再依据这些特征构建一些中间变量(intermediate variables)以更好进行预测，例如

1. a_1: max family size，与 size 和 # bedrooms 相关，故 $a_1 = \mathrm{ReLU}(\omega_1 x_1 + \omega_1 x_2 + b_1)$
2. a_2: walkable, 与 zip code 相关，故 $a_2 = \mathrm{ReLU}(\omega_3 x_3 + b_2)$

3. a_3: school quality，与 zip code 和 wealth 相关，故 $a_3 = \mathrm{ReLU}(\omega_4 x_3 + \omega_5 x_4 + b_3)$

这样最后得到的输出(如图2所示)为：

$$h_\theta(\boldsymbol{x}) = \omega_6 a_1 + \omega_7 a_2 + \omega_8 a_3 + b_4, \quad \theta = \{\omega_1, \cdots, \omega_8, b_1, \cdots, b_4\} \tag{3}$$

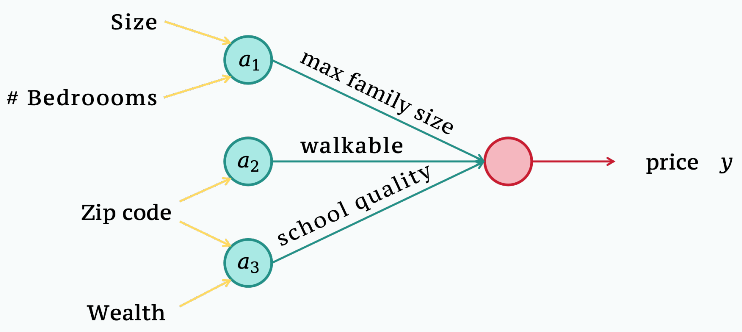

Figure 2: house price prediction example

Neural network – General case

从上面的例子可以发现，我们组建新的特征时是基于我们的先验知识的，这需要很高的代价，并且很多时候也并不全部准确. 因此我们可以考虑将层间所有的神经元都链接起来，让模型自己提取有用的特征，即如图3所示，此时称这种网络为全链接神经网络(fully connected neural network, FCNN). 图中共有两层神经元，除开输出层的都称为隐藏层.

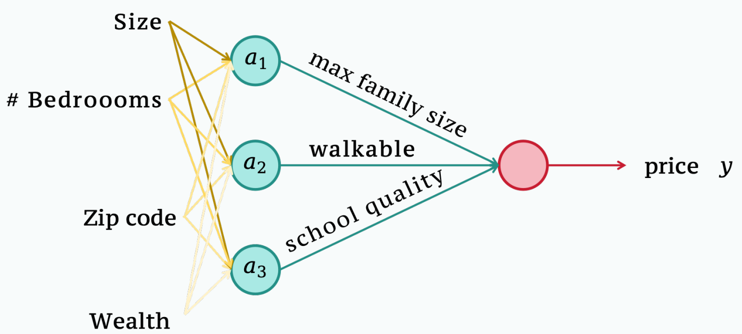

Figure 3: fuly connected neural networks

这样之后，我们就可以将此全链接神经网络数学地写为：

$$\begin{cases} a_1 = \mathrm{ReLU}(\boldsymbol{w}_1^{[1]}\boldsymbol{x} + b_1^{[1]}), & \boldsymbol{w}_1^{[1]} \in \mathbb{R}^4, \boldsymbol{x} \in \mathbb{R}^4, b_1^{[1]} \in \mathbb{R} \\ a_2 = \mathrm{ReLU}(\boldsymbol{w}_2^{[1]}\boldsymbol{x} + b_2^{[1]}), & \boldsymbol{w}_2^{[1]} \in \mathbb{R}^4, \boldsymbol{x} \in \mathbb{R}^4, b_2^{[1]} \in \mathbb{R} \\ a_3 = \mathrm{ReLU}(\boldsymbol{w}_3^{[1]}\boldsymbol{x} + b_3^{[1]}), & \boldsymbol{w}_3^{[1]} \in \mathbb{R}^4, \boldsymbol{x} \in \mathbb{R}^4, b_3^{[1]} \in \mathbb{R} \end{cases} \tag{4a}$$

$$\Rightarrow h_\theta(\boldsymbol{x}) = \boldsymbol{w}^{[2]}\boldsymbol{a} + b^{[2]}, \quad \boldsymbol{w}^{[2]} \in \mathbb{R}^3, \boldsymbol{a} \in \mathbb{R}^3, b^{[2]} \in \mathbb{R} \tag{4b}$$

Vectorization: 为得到更简洁的表示，并且帮助 GPU 并行化，需要将上述数学表示向量化. 同时我们将其推广到共r层神经元、隐藏层中每一层有$m_k, k = 1, \cdots, r-1$个神经元

的一般情形：

$$\boldsymbol{W}^{[1]}=\begin{bmatrix} -- & (\boldsymbol{w}_1^{[1]})^T & -- \\ -- & (\boldsymbol{w}_2^{[1]})^T & -- \\ -- & \cdots & -- \\ -- & (\boldsymbol{w}_{m_1}^{[1]})^T & -- \end{bmatrix}\in\mathbb{R}^{m_1\times d},\quad \boldsymbol{b}^{[1]}=\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{m_1}^{[1]} \end{bmatrix},\in\mathbb{R}^{m_1\times 1}\quad \boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}\in\mathbb{R}^{d\times 1} \tag{5}$$

其中上标$^{[1]}$表示第1层，这样得到第一层的预激活值(pre-activation)为

$$\boldsymbol{z}^{[1]}=W^{[1]}\boldsymbol{x}+\boldsymbol{b}^{[1]},\quad \underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_{m_1}^{[1]} \end{bmatrix}}_{\boldsymbol{z}^{[1]}\in\mathbb{R}^{m_1\times 1}} = \underbrace{\begin{bmatrix} -- & w_1^{[1]\top} & -- \\ -- & w_2^{[1]\top} & -- \\ & \vdots & \\ -- & w_{m_1}^{[1]\top} & -- \end{bmatrix}}_{W^{[1]}\in\mathbb{R}^{m_1\times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{\boldsymbol{x}\in\mathbb{R}^{d\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{m_1}^{[1]} \end{bmatrix}}_{\boldsymbol{b}^{[1]}\in\mathbb{R}^{m_1\times 1}} \tag{6}$$

第一层的激活值(activation)为

$$\boldsymbol{a}^{[1]}=\begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_{m_1}^{[1]} \end{bmatrix}=\begin{bmatrix} \mathrm{ReLU}(z_1^{[1]}) \\ \mathrm{ReLU}(z_2^{[1]}) \\ \vdots \\ \mathrm{ReLU}(z_{m_1}^{[1]}) \end{bmatrix}\triangleq\mathrm{ReLU}(\boldsymbol{z}^{[1]}) \tag{7}$$

第k层的预激活值和激活值分别为$\boldsymbol{z}^{[k]}=W^{[k]}\boldsymbol{a}^{[k-1]}+\boldsymbol{b}^{[k]}$和$\boldsymbol{a}^{[k]}=\mathrm{ReLU}(\boldsymbol{z}^{[k]})$：

$$\underbrace{\begin{bmatrix} z_1^{[k]} \\ \vdots \\ \vdots \\ z_{m_k}^{[k]} \end{bmatrix}}_{\boldsymbol{z}^{[k]}\in\mathbb{R}^{m_k\times k}} = \underbrace{\begin{bmatrix} -- & w_1^{[k]\top} & -- \\ -- & w_2^{[k]\top} & -- \\ & \vdots & \\ -- & w_{m_k}^{[k]\top} & -- \end{bmatrix}}_{W^{[k]}\in\mathbb{R}^{m_k\times m_{k-1}}} \underbrace{\begin{bmatrix} a_1^{[k-1]} \\ a_2^{[k-1]} \\ \vdots \\ a_{k-1}^{[k-1]} \end{bmatrix}}_{\boldsymbol{a}^{[k-1]}\in\mathbb{R}^{m_{k-1}\times 1}} + \underbrace{\begin{bmatrix} b_1^{[k]} \\ b_2^{[k]} \\ \vdots \\ b_{m_k}^{[k]} \end{bmatrix}}_{\boldsymbol{b}^{[k]}\in\mathbb{R}^{m_k\times 1}} \tag{8}$$

$$\boldsymbol{a}^{[k]}=\begin{bmatrix} a_1^{[k]} \\ a_2^{[k]} \\ \vdots \\ a_{m_k}^{[k]} \end{bmatrix}=\begin{bmatrix} \mathrm{ReLU}(z_1^{[k]}) \\ \mathrm{ReLU}(z_2^{[k]}) \\ \vdots \\ \mathrm{ReLU}(z_{m_k}^{[k]}) \end{bmatrix}\triangleq\mathrm{ReLU}(\boldsymbol{z}^{[k]}) \tag{9}$$

最后全部层的向前传播公式为：

$$\begin{aligned} a^{[1]} &= \mathrm{ReLU}(W^{[1]}x+b^{[1]}) \\ a^{[2]} &= \mathrm{ReLU}(W^{[2]}a^{[1]}+b^{[2]}) \\ &\vdots \\ a^{[r-1]} &= \mathrm{ReLU}(W^{[r-1]}a^{[r-2]}+b^{[r-1]}) \\ h_\theta(x) &= W^{[r]}a^{[r-1]}+b^{[r]} \end{aligned} \tag{10}$$

Why do we need activation function?

激活函数(activation function) 在神经网络中是非常重要的，其作用是引入非线性关系，从而增强模型的表达能力，常见的激活函数有：

Comparison of Common Activation Functions

Function	Formula	Range	Derivative Characteristics
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$	Vanishing gradient for large $\lvert x\rvert$
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1, 1)$	Zero-centered, vanishing gradient for large $\lvert x\rvert$
ReLU	$\mathrm{ReLU}(x) = \max(0, x)$	$[0, \infty)$	Non-saturating, gradient is 0 for $x < 0$
Leaky ReLU	$\mathrm{LReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$	$(-\infty, \infty)$	Non-zero gradient for $x < 0$ (controlled by α)
Softmax	$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	$(0, 1)$	Used for multi-class classification, sum of outputs = 1

[1] ReLU: Rectified Linear Unit; α in Leaky ReLU is typically a small positive constant (e.g., 0.01).
[2] Sigmoid and Tanh can suffer from the vanishing gradient problem, especially in deep networks.

Table 1: Comparison of Common Activation Functions

如果没有激活函数作用，那么以两层神经网络为例：

$$a^{[1]} = W^{[1]}x + b^{[1]}, \quad h_\theta(x) = W^{[2]}a^{[1]} + b^{[2]} = W^{[2]}W^{[1]}x + W^{[2]}b^{[1]} + b^{[2]} \tag{11}$$

其仍然是一个线形的，并且等价于一层线性层作用.

在 kernel methods 中：

$$a = \phi(x), \quad h(x) = Wa + b = W\phi(x) + b \tag{12}$$

我们使用 feature map $\phi(x)$ 自主选择了需要的特征(features)，但是在神经网络中，我们实际上是通过学习得到的 $\phi(\cdot)$，因此在隐藏层也称为 features / representations learning，中间得到的 $a^{[k]}$ 也被称为 features / representations.

Questions

Questions: kernel and Neural networks

如果在神经网络中每一层再加上一部 kernel 的作用会怎么样呢？即：

$$\begin{aligned} &a^{[r-1]} = \phi_\beta(a^{[r-2]}), \quad \beta = \{W^{[1]}, \cdots, W^{[r-1]}, b^{[1]}, \cdots, b^{[r-1]}\} \\ &h(x) = W^{[r]}a^{[r-1]} + b^{[r]} \end{aligned} \tag{13}$$

1. 这样会导致神经网络表达能力更强吗?
2. 还是说理论上可以证明即使加了 kernel methods 二者的表达能力相同?
3. 即使理论上表达能力相同，实际效果是否有差异?
4. 如果理论上就强那可以弥补带来的计算量上升吗?

A Different optimize methods

Different optimize methods

Algorithm 1 Full Gradient Descent

Require: Dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, loss function $J(\boldsymbol{\theta})$, learning rate η

1: **Initialize** parameters $\boldsymbol{\theta}$
2: **repeat**
3: Compute gradient: $\boldsymbol{g} = \frac{1}{N}\sum_{i=1}^N \nabla J_i(\boldsymbol{\theta})$
4: Update parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \boldsymbol{g}$
5: **until** convergence

Algorithm 2 Stochastic Gradient Descent (SGD)

Require: Dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, loss function $J(\boldsymbol{\theta})$, learning rate η

1: **Initialize** parameters $\boldsymbol{\theta}$
2: **repeat**
3: Randomly sample a data point $(\boldsymbol{x}_i, y_i)$
4: Compute gradient: $\boldsymbol{g} = \nabla J_i(\boldsymbol{\theta})$
5: Update parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \boldsymbol{g}$
6: **until** convergence

Algorithm 3 Mini-batch Gradient Descent

Require: Dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, loss function $J(\boldsymbol{\theta})$, learning rate η, batch size m

1: **Initialize** parameters $\boldsymbol{\theta}$
2: **repeat**
3: Randomly sample a mini-batch $\mathcal{B}$ of m data points
4: Compute gradient: $\boldsymbol{g} = \frac{1}{m}\sum_{i\in\mathcal{B}} \nabla J_i(\boldsymbol{\theta})$
5: Update parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \boldsymbol{g}$
6: **until** convergence

Last updated: December 8, 2024

References