Subject: Stanford CS229 Machine Learning, Lecture 8, Neural Networks 2(Back propagation)

Date: from December 31, 2024 to January 18, 2025

Contents

CS229 Machine Learning, Neural Networks 2(backprop), 2022, Lecture 9

YouTube:Stanford CS229 Machine Learning, Neural Networks 2(backprop), 2022, Lecture 9

Outline

Outline

$$\text{Outline}\begin{cases}1. & \text{Review}\\ 2. & \text{Differential circuits / networks}\\ 3. & \text{Chain rule(Preliminary)}\\ 4. & \text{Back propagation}\end{cases}$$

Review

Loss and parameters update

在神经网络中，第 i 个样本(sample) $x^{(i)}$ 产生的 loss 为

$$J^{(i)} = \frac{1}{2}(y^{(i)} - h_\theta(x^{(i)}))^2$$

用该样本更新参数 θ 的算法为

$$\theta := \theta - \alpha\nabla J^{(i)}(\theta)$$

其中 α 为学习率。此时问题的重点就变成如何求得梯度 $\nabla J^{(i)}(\theta)$.

Differentiable circuits / networks

Differentiable circuits / networks

在介绍如何求得梯度之前，先了解一下梯度是否能计算、好计算是必要的，下面先介绍 Differentiable circuits / network，其是 second-order method 和 meta learning 的基础.

Definition 1 (Differentiable circuits / networks)**.** 称由一系列基本算符(+ - × /) 和基本函数(例如cos, sin, exp, log, *ReLU*等)组合而成的组合体为 *differentiable circuits / networks*[a].

对于此 Differentiable circuits，事实上可以说明用起计算导数是可以做到且并不困难：

Theorem 1. *设一实值函数 $f:\mathbb{R}^l \to \mathbb{R}$ 可被大小为 N 的 Differentiable circuit 计算出，那么其梯度 $\nabla f \in \mathbb{R}^l$ 可以被大小为 $O(N)$ Differentiable circuit 计算出，且计算时间复杂度为 $O(N)$.*

Note 1.
1. *Theorem 1 中隐含地假设了 $N > l$，因为要计算出所有l维度的值.*
2. *Theorem 1 表明对于一个 circuit，计算函数值 f 与计算其梯度 ∇f 所需时间差不多，因此计算梯度并不比计算 loss 本身更困难！*
3. *应用于 neural network 中，f 即 $J^{(i)}(\theta)$，l =# parameters, $N = O$(# parameters)*

Corollary 1. *在相同的设置下，$\forall v \in \mathbb{R}^l$，计算$\nabla^2 f(x)\cdot v$ 的时间复杂度为 $O(N + l)$.*

Proof. 事实上，如果先计算$\nabla^2 f(x)$就有$O(l^2)$的计算量，再做内积就有$O(l^2+l)$就算量。但是，令$g(x) := \langle \nabla f(x), v\rangle : \mathbb{R}^l \to \mathbb{R}$，由Theorem 1可知 $g(x)$的计算量为$O(N+l)$，再次使用 Theorem 1 可知 $\nabla g = \nabla \langle \nabla f, v\rangle$的计算量依然为$O(N+l)$. □

[a]更多介绍可见Differentiable Circuits And PyTorch

Preliminary – Chain rule

Chain rule

Settings: $J(\theta_1, \theta_2, \cdots, \theta_p)$为自变量为 $\theta_1, \theta_2, \cdots, \theta_p$的函数
并有中间变量 $g_1 = g_1(\theta_1, \theta_2, \cdots, \theta_p), \cdots, g_k = g_1(\theta_1, \theta_2, \cdots, \theta_p)$，因此 J 也可以写为 $J(g_1, g_2, \cdots, g_k)$

Chain rule:

$$\frac{\partial J}{\partial \theta_i} = \sum_{j=1}^{k} \frac{\partial J}{\partial g_j} \cdot \frac{\partial g_j}{\partial \theta_i}$$

若$\theta \in \mathbb{R}$，则$\frac{\partial J}{\partial \theta} \in \mathbb{R}$，若$\theta \in \mathbb{R}^d$，则$\frac{\partial J}{\partial \theta} = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_d} \end{bmatrix} \in \mathbb{R}^d$，若$A \in \mathbb{R}^{d_1 \times d_2}$，则$\frac{\partial J}{\partial A} = \left[\frac{\partial J}{\partial A_{ij}}\right] \in \mathbb{R}^{d_1 \times d_2}$

Back propagation

Chain for two layer Neural network

Settings: 我们考虑简单的两层神经网络：

$$\begin{aligned}
&z = W^{[1]}x + b^{[1]} \in \mathbb{R}^m, \quad x \in \mathbb{R}^d, W^{[1]} \in \mathbb{R}^{m \times d}, b^{[1]} \in \mathbb{R}^d \\
&a = \sigma(z) \in \mathbb{R}^m \\
&h_\theta(x) = o = W^{[2]}a + b^{[2]} \in \mathbb{R}^1 \\
&J = \frac{1}{2}(y-o)^2
\end{aligned}$$

Abstraction: 我们首先抽象地形式化看看计算梯度会发生什么，先设：

$$J = J(z), z = Wu + b, x \in \mathbb{R}^d, W \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^d$$

那么根据链式法则就有：

$$\underbrace{\frac{\partial J}{\partial W}}_{\in \mathbb{R}^{m \times d}} = \underbrace{\frac{\partial J}{\partial z}}_{\in \mathbb{R}^{m \times 1}} \cdot \underbrace{u^T}_{\in \mathbb{R}^{1 \times d}}, \quad \frac{\partial J}{\partial b} = \frac{\partial J}{\partial z}$$

$$\begin{aligned}
\frac{\partial J}{\partial W_{ij}} &= \sum_{k=1}^{m} \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial W_{ij}} = \sum_{k=1}^{m} \frac{\partial J}{\partial z_k} \cdot \frac{\partial (W_{k1}u_1 + W_{k2}u_2 + \cdots + W_{kd}u_d + b_k)}{\partial W_{ij}} \\
&= \frac{\partial J}{\partial z_k} \cdot \frac{\partial (W_{i1}u_1 + W_{i2}u_2 + \cdots + W_{id}u_d + b_k)}{\partial W_{ij}} = \frac{\partial J}{\partial z_k} \cdot u_j
\end{aligned}$$

Application to 2-layers NN: 我们想要计算损失函数关于神经网络参数的梯度，根据上面的结果就有：

$$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial z} \cdot x^T, \quad \frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial z}, \quad \frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial o} \cdot a^T, \quad \frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial o}$$

由于 $a = \sigma(z) \in \mathbb{R}^m, J = J(a)$，因此有：

$$\underbrace{\frac{\partial J}{\partial z}}_{\in\mathbb{R}^m} = \underbrace{\frac{\partial J}{\partial a}}_{\in\mathbb{R}^m} \odot \underbrace{\sigma'(z)}_{\in\mathbb{R}^m}$$

其中 $\odot$ 指entry-wise，因为激活函数 σ 的作用也是 entry-wise 的.

对于 $\frac{\partial J}{\partial a}$，有：

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial a} = w^{[2]} \cdot \frac{\partial J}{\partial o}$$

对于 $\frac{\partial J}{\partial o}$，有：

$$\frac{\partial J}{\partial o} = -(y - o)$$

Chain rule for deep neural networks

Settings: 考虑一个 r 层的深度神经网络：

$$\begin{aligned} z^{[1]} &= w^{[1]}x + b^{[1]} \in \mathbb{R}^m \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= w^{[2]}a^{[1]} + b^{[2]} \\ &\cdots \\ a^{[r-1]} &= \sigma(z^{[r-1]}) \\ z^{[r]} &= w^{[r]}a^{[r-1]} + b^{[r]} \\ J &= \frac{1}{2}(y - z^{[r]})^2 \end{aligned}$$

Chain rule: 我们要计算损失 J 对第 k层参数 $W^{[k]}, b^{[k]}$ 的梯度。由于

$$\begin{aligned} z^{[k]} &= W^{[k]} \cdot a^{[k-1]} + b^{[k]} \\ J &= J\left(z^{[k]}\right) \end{aligned}$$

因此：

$$\frac{\partial J}{\partial W^{[k]}} = \frac{\partial J}{\partial z^{[k]}} \cdot \left(a^{[k-1]}\right)^T$$

由于

$$\begin{aligned} a^{[k]} &= \sigma(z^{[k]}) \\ J &= J(a^{[k]}) \end{aligned}$$

因此：

$$\frac{\partial J}{\partial z^{[k]}} = \frac{\partial J}{\partial a^{[k]}} \odot \sigma'(z^{[k]})$$

由于

$$z^{[k+1]} = W^{[k+1]} \cdot a^{[k]} + b^{[k+1]}$$

$$J = J\left(z^{[k+1]}\right)$$

因此:

$$\frac{\partial J}{\partial a^{[k]}} = \left(W^{[k+1]}\right)^T \frac{\partial J}{\partial a^{[k+1]}}$$

...

$$\ldots$$

Summary

在神经网络向前传播中，如图1计算顺序是:

$$\text{forward pass: } z^{[1]}, a^{[1]} \to z^{[2]}, a^{[2]}, \to \cdots \to z^{[r]}, a^{[r]}$$

$$x \longrightarrow \boxed{MM} \longrightarrow z^{[1]} \longrightarrow \boxed{\textit{activation}} \longrightarrow a^{[1]} \longrightarrow \boxed{MM} \longrightarrow z^{[2]} \longrightarrow \cdots \longrightarrow z^{[r]} \longrightarrow J$$

Figure 1: Forward pass

其中 MM 是一个模块(module)，指矩阵乘法(matrix multiplication).
但是在计算梯度时，计算的顺序是:

$$\text{backward pass: } \frac{\partial J}{\partial z^{[r]}} = -(y - z^{[r]}) \to \frac{\partial J}{\partial a^{[r-1]}} = (W^{[r]})^T \frac{\partial J}{\partial z^{[r]}}, \frac{\partial J}{\partial z^{[r-1]}} = \frac{\partial J}{\partial a^{[r-1]}} \odot \sigma'(z^{[r-1]})$$
$$\to \cdots \to \frac{\partial J}{\partial a^{[k]}}, \frac{\partial J}{\partial z^{[k]}} \to \cdots$$

因此计算梯度时，MM-module 原本的输出变成了输入，输入变成了输出，这也是反向传播名称的由来.

Last updated: January 18, 2025

References