Lecture3-2 Deep Neural Network

1. 介绍

线性模型和神经网络的最大区别，在于神经网络的**非线性**导致大多数我们感兴趣的代价函数都变得非凸

- 线性模型可以使用线性方程求解，或者用于逻辑回归/SVM 等**凸优化算法保证全局最优解**
- 神经网络的训练通常使用**迭代的、基于梯度的优化**，使得**代价函数得到一个比较小的值**
  - 对于凸函数，从任何一种初始参数出发都会收敛
  - 对于非凸函数，没有收敛性的保证，对参数的初始值很敏感

2. 代价函数/损失函数/目标函数

深度神经网络设计中的一个重要方面是代价函数的选择

通常参数模型定义了一个分布 $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ ，简单地使用**最大似然原理**

- 这意味着我们使用训练数据和模型预测间的**交叉熵**作为代价函数
- 有时候也使用更简单的方法，仅仅预测在给定 $\boldsymbol{x}$ 的条件下 y 的某种统计量

交叉熵损失函数 Cross Entropy

与其他参数模型相似，我们通常使用**极大似然**进行训练，因此代价函数为**负对数似然**，即训练数据与模型分布之间的交叉熵

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y}|\boldsymbol{x})$$

具体对于每一个输出的预测结果 $\hat{\boldsymbol{y}}$ 来说

$$L(\hat{\boldsymbol{y}}) = \sum_{i=1}^{m} y_i \ \log \hat{y}_i$$

均方误差损失函数 MSE

$$J(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}}||\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})||^2 + \text{const}$$

3. 输出单元

一般的，如果我们定义了一个条件分布 $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$，最大似然的原则建议我们使用 $-\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ 作为代价函数

线性单元

一种简单的输出单元是基于**仿射变换**的输出单元，仿射变换**不具有非线性**，这些单元往往被直接称为**线性单元**

向线性单元给定特征 $\boldsymbol{h}$，线性输出单元层输出向量 $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$

线性输出层经常被用来产生**条件高斯分布的均值**

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y};\hat{\boldsymbol{y}},\boldsymbol{I})$$

线性模型不会饱和，它们易于采用**基于梯度的优化算法**，甚至可以使用其他多种优化算法

sigmoid 单元

许多任务需要预测**二值型变量** y 的值，二分类问题可以归结于这种形式

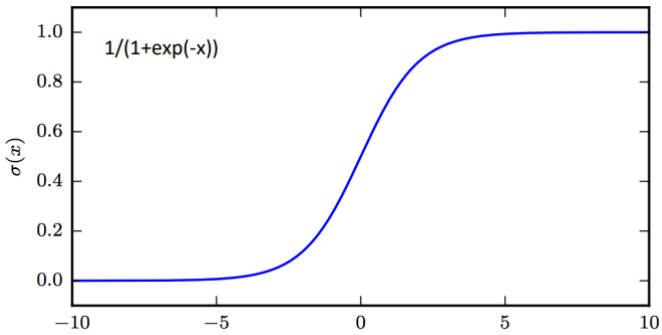
此时最大似然的方法是定义 y 在 $\boldsymbol{x}$ 条件下的 Bernoulli 分布，Bernoulli 分布仅需单个参数来定义，神经网络只需要预测 $P(y=1|\boldsymbol{x})$ 即可

sigmoid 输出单元的定义为

$$\hat{y} = \sigma(\boldsymbol{w}^T\boldsymbol{h} + b)$$

- 这里 $\sigma(z)$ 函数是 logistic sigmoid 函数
- 可以认为 sigmoid 输出单元具有两个部分
  - 它使用一个线性层来计算 $z = \boldsymbol{w}^T\boldsymbol{h} + b$
  - 然后使用一个 **sigmoid 激活函数**将 z 转换成概率

sigmoid 激活函数



$$g(z) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- 通常用来产生 Bernoulli 分布中的参数 ϕ
- 范围：$(0, 1)$
- 偏导：$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$

代价函数选择

这种在对数空间里预测概率的方法可以很自然地使用最大似然学习

当我们使用其他的损失函数，例如**均方误差**之类的，损失函数会在 $\sigma(z)$ 饱和时饱和

- sigmoid 激活函数在z 取非常小的负值时会饱和到 0，当 z 取非常大的正值时会饱和到 1
  - 这种情况一旦发生，**梯度会变得非常小以至于不能用来学习**，无论此时模型给出的是正确还是错误的答案
  - 因此，**最大似然**几乎总是训练 sigmoid 输出单元的优选方法

softmax 单元

当想要表示一个具有 n **个可能取值的离散型随机变量**的分布时，可以使用 softmax 函数

softmax 激活函数

将 sigmoid 推广到 n 个值的离散变量的情况，现在需要创造一个向量 $\hat{\boldsymbol{y}}$。使得它的每个元素 $\hat{y}_i = P(y = i|\boldsymbol{x})$，softmax 可以满足下面的要求

$$g(z) = \text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- 每个元素 $\hat{y}_i$ 的元素分布介于 0 和 1 之间
- 整个向量的元素之和为 1
- 如果 softmax 和 交叉熵一起使用的话，偏导 $\frac{\partial L(y, \hat{y})}{\partial z} = \hat{y} - y$

4. 隐藏单元

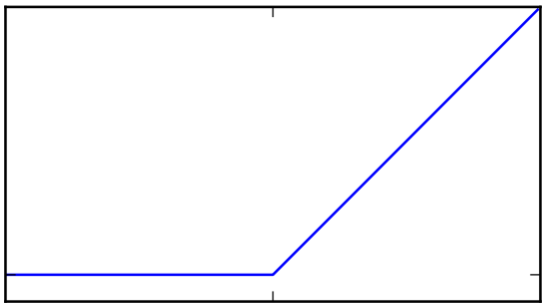
除非另有说明，大多数的隐藏单元都可以描述为接受输入向量 $\boldsymbol{x}$，计算仿射变换 $\boldsymbol{x} = \boldsymbol{W}^T\boldsymbol{x} + b$ 然后使用一个逐元素的非线性函数 $g(\boldsymbol{z})$

整流线性单元 ReLU

整流线性单元是隐藏单元极好的默认选择，它通常用于仿射变换之上，行为**更接近线性**，那么**模型更容易优化**

$$\boldsymbol{h} = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

ReLU 激活函数



$$g(z) = \max\{0, z\}$$

- 偏导：$\frac{\partial g(z)}{\partial z} = \begin{cases} 0, & \text{if } z \leq 0 \\ 1, & \text{if } z > 0 \end{cases}$

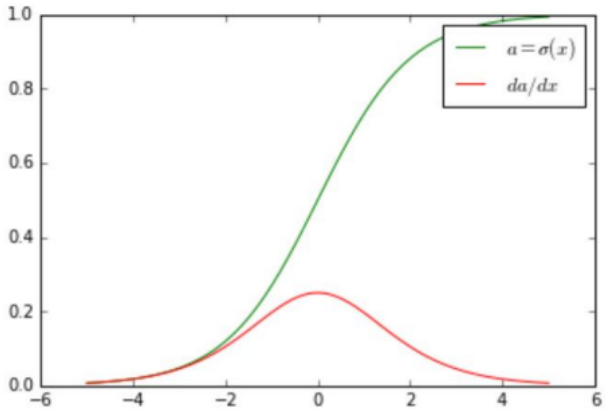
它有很多好处

- 强壮且很快速的计算梯度
- 虽然在 0 不可导，但是不是一个很大的问题
- 但是 ReLU 很容易“死亡”，我们会在梯度爆炸的时候讨论这个问题

logistic sigmoid 与双曲正切

在引入整流线性单元之前，大多数神经网络使用 logistic sigmoid 激活函数

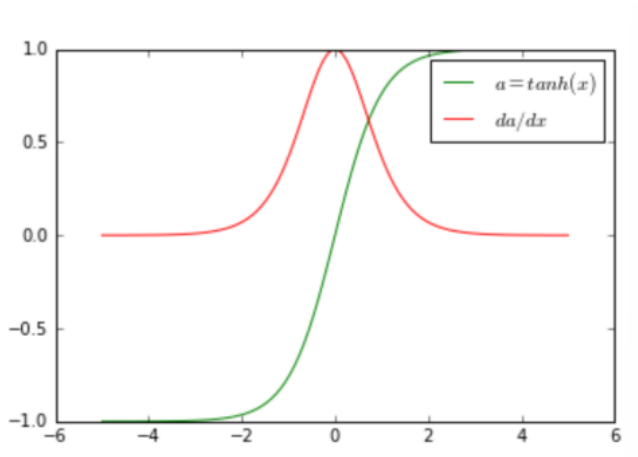
- 这两者在 ReLU 出现之前都很流行
- 但它们**很容易饱和（零梯度）**
- 此外，**小梯度**，可能会有问题，特别是当我们将几个小梯度相乘时（稍后参见链式法则…）、
- 两者之间，tanh 作为隐藏层的激活函数会更好



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

- 梯度：$\frac{\partial g(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$

或者双曲正切激活函数



$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- 梯度：$\frac{\partial g(z)}{\partial z} = 1 - \tanh^2(z)$

其它隐藏单元

一些常见的隐藏单元包括

softplus 函数

$$g(z) = \zeta(z) = \log(1 + e^z)$$

硬双曲正切函数 hard tanh

它的形状和 tanh 以及整流线性单元类似，但是不同于后者，它是有界的

$$g(z) = \max(-1, \min(1, z))$$

4. 架构设计

架构（architecture）：网络的整体结构，它应该具有多少单元，以及这些单元应该如何连接

- 大多数神经网络被组织成称为**层**的单元组
- 大多数神经网络架构将这些层布置成**链式结构**，其中**每一层都是前一层的函数**
  - 第一层：$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$
  - 第二层：$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{x} + \boldsymbol{b}^{(2)}\right)$
- 架构主要考虑网络的**深度**和每一层的**宽度**，以及每个单元的**具体构成**

层和深度

万能近似定理 universal approximation theorem

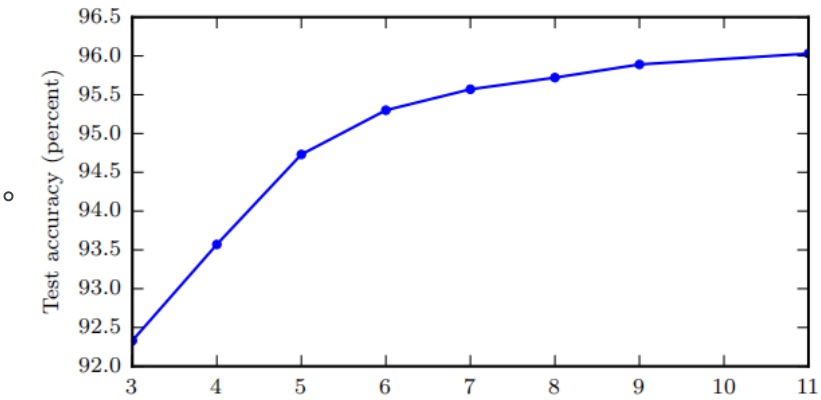
> 一个前馈神经网络如果具有**线性输出层**和**至少一层具有任何一种“挤压”性质的激活函数**（例如logistic sigmoid激活函数）的隐藏层，只要给予网络**足够数量的隐藏单元**，它可以以任意的精度来近似任何定义在 $\mathbb{R}^n$ 的有界闭集上的任意连续函数

万能近似定理意味着无论我们试图学习什么函数，一个大的 MLP 一定能够**表示**这个函数，虽然不能确定训练算法能够**学习**到这个函数

但是“没有免费的午餐定理”表示，即使 MLP 能够表示该函数，学习也可能因两个不同的原因而失败

- **局部极小值**：训练算法可能找不到用于期望函数的参数值
- **过拟合**：训练算法选择了错误的函数

层和深度的选择

- 单层的前馈网络足以表示任何函数，但是**网络层可能大得不可实现**，并且可能无法正确地学习和泛化
    - 一个大的 MLP 最坏情况下，可能需要**指数数量**的隐藏单元
- 在很多情况下，使用**更深**的模型能够减少表示期望函数所需的**单元的数量**，并且可以**减少泛化误差**
    - 

    - 测试集上的准确率随着深度的增加而不断增加（数据来自 Goodfellow *et al.* (2014d)）

连接

架构设计考虑的另外一个关键点是如何将层与层之间连接起来

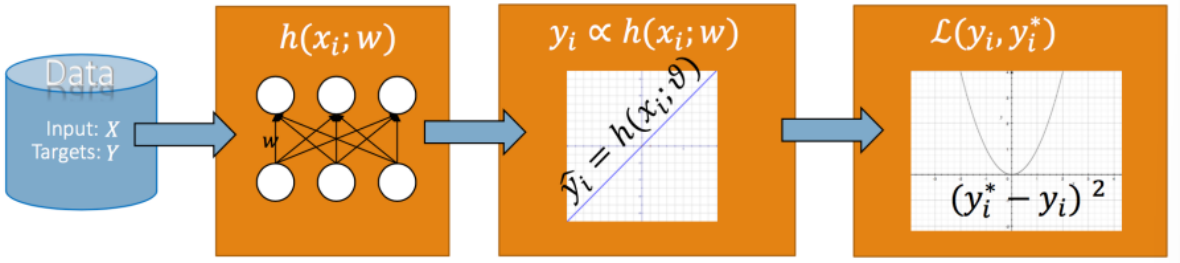
默认情况下，每个输入单元连接到每个输出单元

- 在之后章节中的许多专用网络具有较少的连接，使得输入层中的每个单元仅连接到输出层单元的一个小子集
- 这些用于减少连接数量的策略减少了参数的数量以及用于评估网络的计算量，但**通常高度依赖于问题**

5. 反向传播算法

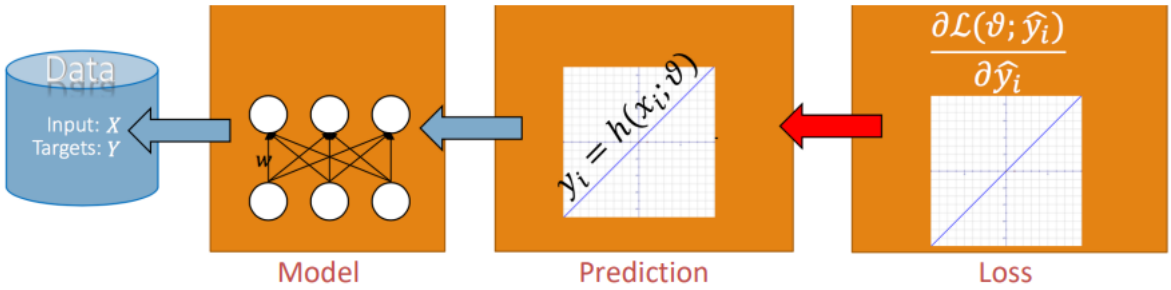
概念

前向传播（forward propagation）



- 输入 x 提供初始信息
- 然后传到每一层的隐藏单元
- 产生输出 y
- 持续向前直到它产生一个标量的代价函数 $J(\theta)$

反向传播（back propagation）



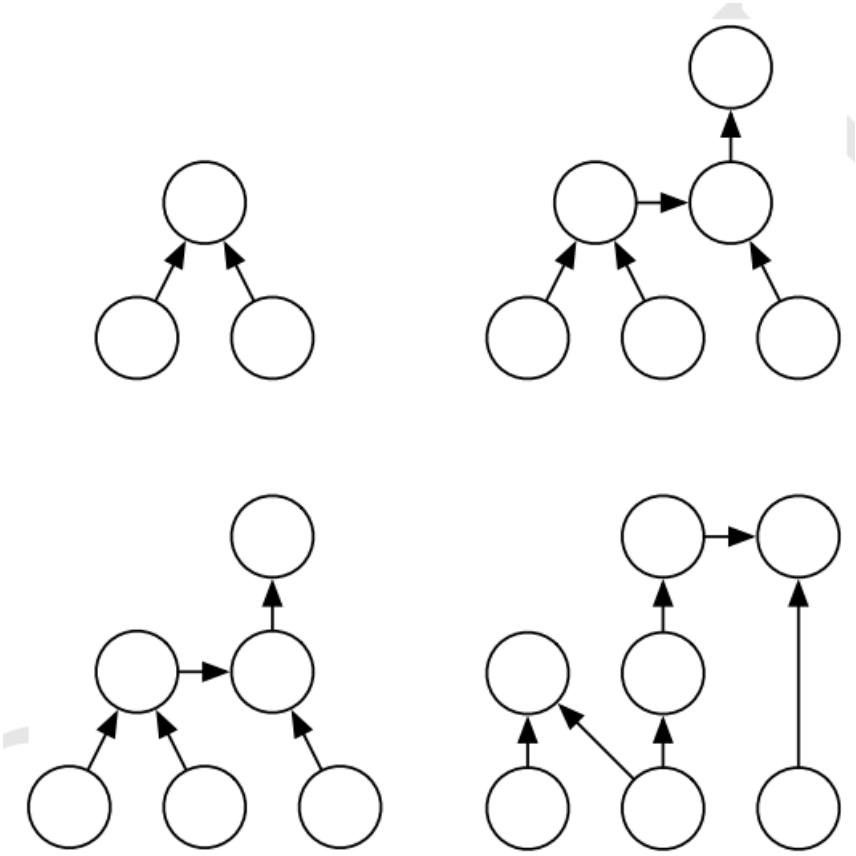
- 计算代价函数的梯度
- 传播梯度信息并在每个单元上计算梯度（链式法则）
- 使用计算梯度和梯度下降规则更新模型参数

关于反向传播的误解

- 反向传播被误解为用于多层神经网络的整个学习算法
  - 事实上，反向传播**仅指用于计算梯度的方法**
  - **随机梯度下降**使用该梯度来进行学习
- 反向传播被误解为仅适用于多层神经网络
  - 其实它可以计算任何函数的导数

计算图

为了精确的描述反向传播算法，使用精确的**计算图（computational graph）**语言



- **节点（node）**：表示每一个**变量**（变量可以是标量、向量、矩阵、张量等）
- **操作（operation）**：一个或者多个变量的简单函数
  - 定义一个操作仅返回单个输出变量
- **边（edge）**：如果变量 y 是变量 x 通过一个操作的到的，那可以画一条从 x 到 y 的有向边

微积分中的链式法则

实数的链式法则

假设 x 是实数，f 和 g 是从实数映射到实数的函数

- 假设 $y = g(x), z = f(g(x)) = f(y)$，那么链式法则有

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

向量的链式法则

假设 $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$，$g$ 是从 $\mathbb{R}^m$ 到 $\mathbb{R}^n$ 的映射，f 是从 $\mathbb{R}^n$ 到 $\mathbb{R}$ 的映射

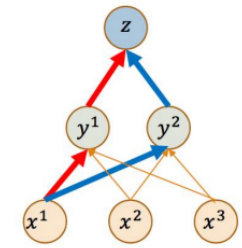
- 假设 $\boldsymbol{y} = g(\boldsymbol{x})$，并且 $z = f(\boldsymbol{y})$，那么链式法则有

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

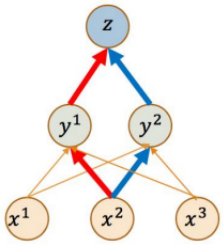
使用向量记法，可以等价写成

$$\nabla_{x^z} = (\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}})^T \nabla_{y^z}$$

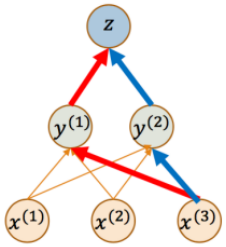
- 这里 $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ 是 $n \times m$ 的 Jacobian 矩阵
- 反向传播算法由图中每一个这样的 Jacobian 梯度的乘积操作所组成



$$\frac{dz}{dx^1} = \frac{dz}{dy^1}\frac{dy^1}{dx^1} + \frac{dz}{dy^2}\frac{dy^2}{dx^1}$$



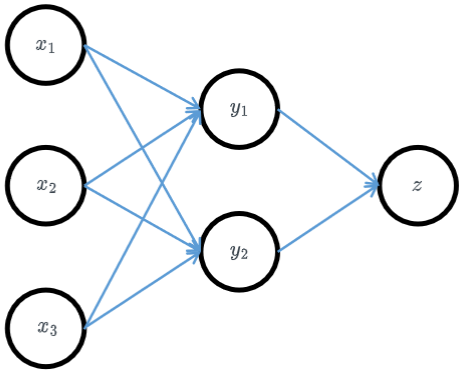
$$\frac{dz}{dx^2} = \frac{dz}{dy^1}\frac{dy^1}{dx^2} + \frac{dz}{dy^2}\frac{dy^2}{dx^2}$$



$$\frac{dz}{dx^3} = \frac{dz}{dy^1}\frac{dy^1}{dx^3} + \frac{dz}{dy^2}\frac{dy^2}{dx^3}$$

- 我们计算所有可能路径的梯度
- 如果用向量来表示的话，我们在计算：$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z$

例如：如下图所示，可以得到



$$\frac{\partial z}{\partial x_1} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_1} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x_1}$$

$$\begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \\ \frac{\partial z}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \frac{\partial y_1}{\partial x_3} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_2}{\partial x_3} \end{bmatrix}^T \begin{bmatrix} \frac{\partial z_1}{\partial y_1} \\ \frac{\partial z_2}{\partial yz} \end{bmatrix}$$

$$\frac{\partial z}{\partial \boldsymbol{x}} = \left[\underbrace{\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}}_{(d^y \times d^x \text{Jacobian})}\right]^T \frac{\partial z}{\partial \boldsymbol{y}}$$

张量的链式法则

为了表示值 z 关于张量 $\mathbf{X}$ 的梯度，记为 $\nabla_{\mathbf{X}^z}$，$\mathbf{X}$ 就像是向量一样，索引现在有多个坐标

如果 $\mathbf{Y} = g(\mathbf{X})$，并且 $z = f(\mathbf{Y})$，那么

$$\nabla_{\mathbf{X}^z} = \sum_j (\nabla_{\mathbf{X}^{Y_j}}) \frac{\partial z}{\partial \mathbf{Y}_j}$$

全连接 MLP 中的前向传播和反向传播

前向传播

典型深度神经网络中的前向传播和代价函数的计算

- $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$：损失函数，取决于输出 $\hat{\boldsymbol{y}}$ 和目标 $\boldsymbol{y}$
- J：总代价
- $\Omega(\theta)$：正则化项
- θ：θ 包含所有参数（权重和偏置）

Input

- l：网络深度
- $\boldsymbol{W}^{(i)}, i \in \{1, \dots, l\}$：模型的权重矩阵
- $\boldsymbol{b}^{(i)}, i \in \{1, \dots, l\}$：模型的偏置参数
- $\boldsymbol{x}$：程序的输入
- $\boldsymbol{y}$：目标输出

Algorithm

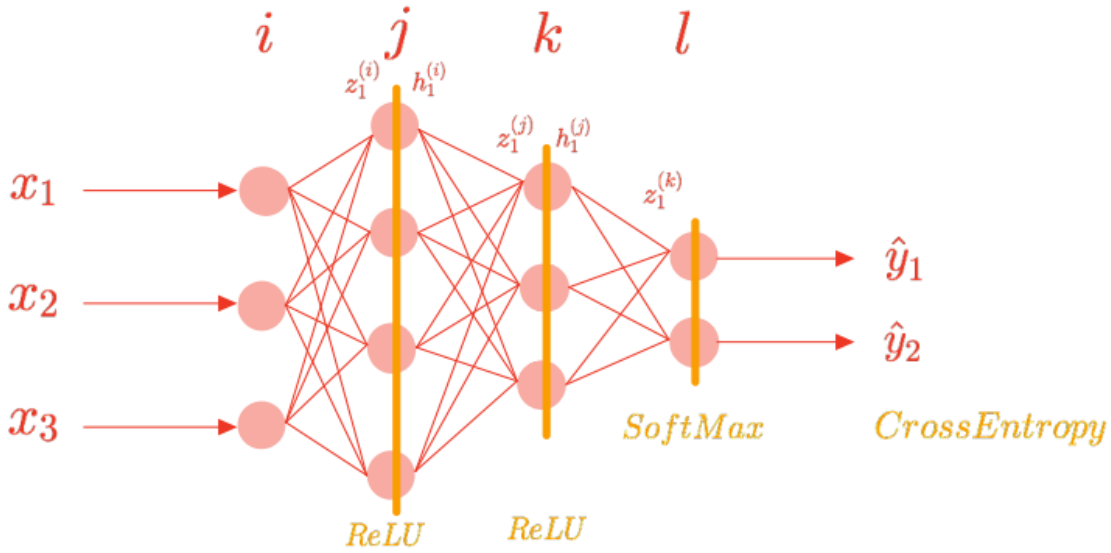
1. $\boldsymbol{h}^{(0)} = \boldsymbol{x}$
2. $\textbf{for } k = 1, \dots, l \textbf{ do}$
3. $\quad \boldsymbol{z}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
4. $\quad \boldsymbol{h}^{(k)} = g(\boldsymbol{z}^{(k)})$
5. $\textbf{end for}$
6. $\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
7. $J = L(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \lambda \Omega(\theta)$

反向传播

Algorithm

1. 在前向传播完成后，计算顶层的梯度
2. $\boldsymbol{g} \leftarrow \boldsymbol{\nabla}_{\hat{\boldsymbol{y}}} J = \boldsymbol{\nabla}_{\hat{\boldsymbol{y}}} L(\boldsymbol{y}, \hat{\boldsymbol{y}})$
3. $\textbf{for } k = l, l-1, \ldots, 1 \textbf{ do}$
4. 　// 将关于层输出的梯度转换为非线性激活输入前的梯度（如果 g 是逐元素的，则逐元素地相乘）
5. 　$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{z}^{(k)}} J = \boldsymbol{g} \odot g'\left(\boldsymbol{z}^{(k)}\right)$
6. 　// 计算关于权重和偏置的梯度（如果需要的话，还要包括正则项）：
7. 　$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
8. 　$\nabla_{\boldsymbol{W}^{(k)} J} = \boldsymbol{g}\boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
9. 　// 关于下一更低层的隐藏层传播梯度
10. 　$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)T} \boldsymbol{g}$
11. $\textbf{end for}$

MLP 示例



- 如图所示为一个有四层的 MLP，其中隐藏单元使用 ReLU 激活函数，输出单元使用 SoftMax 激活函数，代价函数使用交叉熵
- 已知的参数如下

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\boldsymbol{W}^{(i)} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \\ w_{14} & w_{24} & w_{34} \end{bmatrix} \qquad \boldsymbol{b}^{(i)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$\boldsymbol{W}^{(j)} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix} \qquad \boldsymbol{b}^{(j)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\boldsymbol{W}^{(k)} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{bmatrix} \qquad \boldsymbol{b}^{(k)} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

前向传播

$$\begin{aligned} \boldsymbol{z}^{(i)} &= \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)} & \boldsymbol{h}^{(i)} &= g_{ReLU}\left(\boldsymbol{z}^{(i)}\right) \\ \boldsymbol{z}^{(j)} &= \boldsymbol{W}^{(j)}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(j)} & \boldsymbol{h}^{(j)} &= g_{ReLU}\left(\boldsymbol{z}^{(j)}\right) \\ \boldsymbol{z}^{(k)} &= \boldsymbol{W}^{(k)}\boldsymbol{h}^{(j)} + \boldsymbol{b}^{(k)} & \hat{\boldsymbol{y}} &= g_{SoftMax}\left(\boldsymbol{z}^{(k)}\right) \end{aligned}$$

$$J(\boldsymbol{\theta}) = L_{CE}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) + \lambda\Omega(\boldsymbol{\theta})$$

反向传播

第 k 层梯度更新

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{W}^{(k)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \widehat{\boldsymbol{y}}} \frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{W}^{(k)}} = (\widehat{\boldsymbol{y}} - \boldsymbol{y}) \cdot \boldsymbol{h}^{(j)T}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{b}^{(k)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \widehat{\boldsymbol{y}}} \frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{b}^{(k)}} = (\widehat{\boldsymbol{y}} - \boldsymbol{y})$$

第 j 层梯度更新

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{W}^{(j)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \widehat{\boldsymbol{y}}} \frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{h}^{(j)}} \frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{z}^{(j)}} \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{W}^{(j)}} = \boldsymbol{W}^{(k)T} \cdot (\widehat{\boldsymbol{y}} - \boldsymbol{y}) \odot g^{(j)'} \cdot \boldsymbol{h}^{(i)T}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{b}^{(j)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \widehat{\boldsymbol{y}}} \frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{h}^{(j)}} \frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{z}^{(j)}} \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{b}^{(j)}} = \boldsymbol{W}^{(k)T} \cdot (\widehat{\boldsymbol{y}} - \boldsymbol{y}) \odot g^{(j)'}$$

第 i 层梯度更新

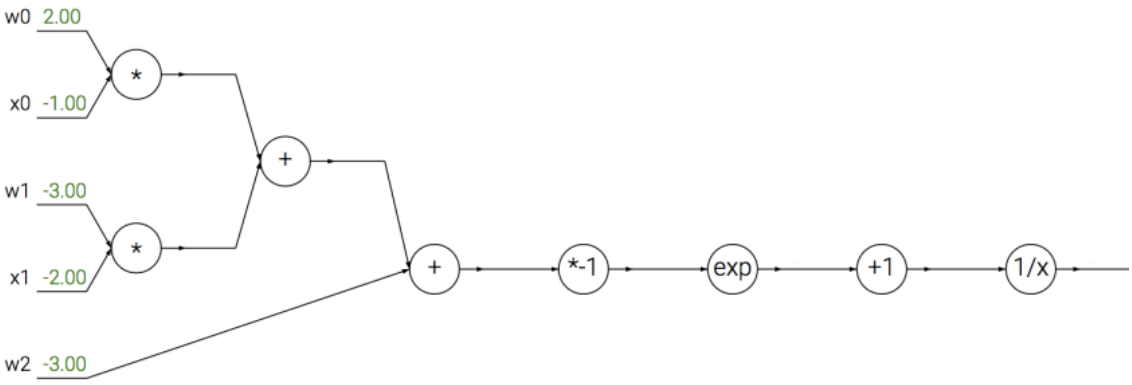
$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{W}^{(i)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \widehat{\boldsymbol{y}}} \frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{h}^{(j)}} \frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{z}^{(j)}} \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{h}^{(i)}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{z}^{(i)}} \frac{\partial \boldsymbol{z}^{(i)}}{\partial \boldsymbol{W}^{(i)}} = \boldsymbol{W}^{(j)T}[\boldsymbol{W}^{(k)T} \cdot (\widehat{\boldsymbol{y}} - \boldsymbol{y}) \odot g^{(j)'}] \odot g^{(i)'} \cdot \boldsymbol{x}^T$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{b}^{(i)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \widehat{\boldsymbol{y}}} \frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{h}^{(j)}} \frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{z}^{(j)}} \frac{\partial \boldsymbol{z}^{(j)}}{\partial \boldsymbol{h}^{(i)}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{z}^{(i)}} \frac{\partial \boldsymbol{z}^{(i)}}{\partial \boldsymbol{b}^{(i)}} = \boldsymbol{W}^{(j)T} \cdot [\boldsymbol{W}^{(k)T} \cdot (\widehat{\boldsymbol{y}} - \boldsymbol{y}) \odot g^{(j)'}] \odot g^{(i)'}$$

反向传播计算图示例

数值计算

我们以如下的计算图为示例

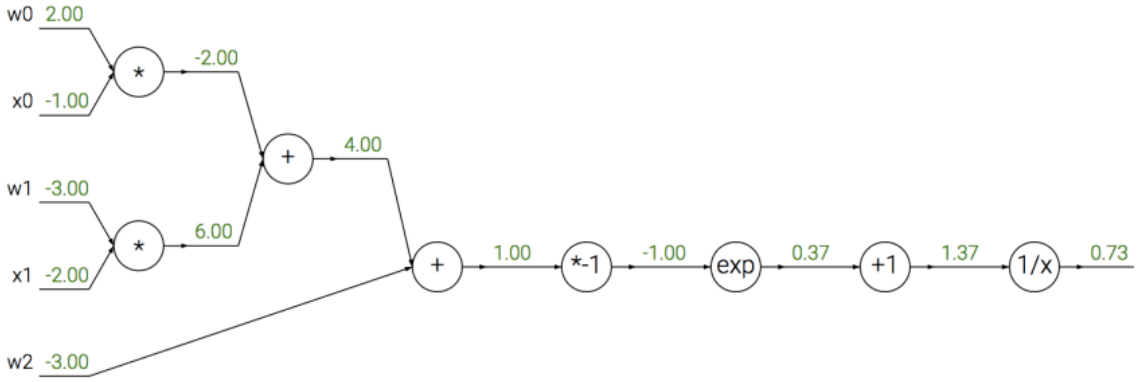


$$f(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

给输入赋值为

$$w_0 = 2, x_0 = -1, w_1 = -3, x_1 = -2, w_2 = -3$$

前向传播经过计算后输出如下



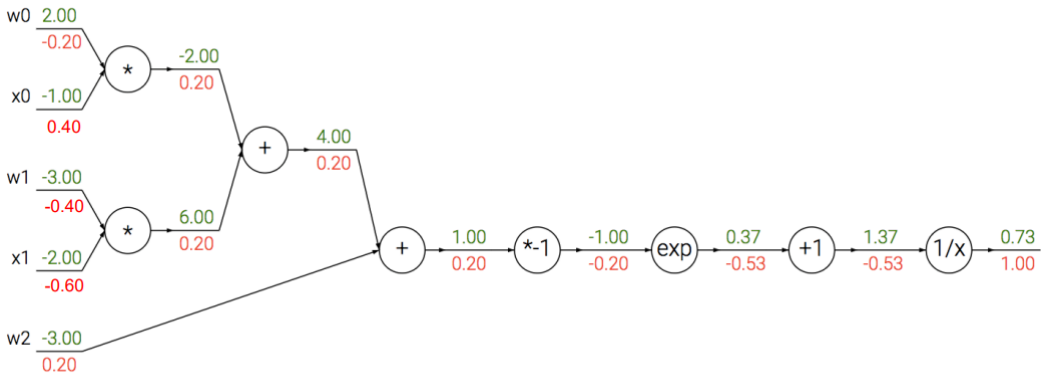
那么其实这个函数可以表示成很多复合函数的嵌套

函数	导数/偏导
$f_1(x,w)=wx$	$\frac{\partial f}{\partial x}=w$ $\frac{\partial f}{\partial w}=x$
$f_2(x,y)=x+y$	$\frac{\partial f}{\partial x}=1$ $\frac{\partial f}{\partial y}=1$
$f_3(x)=x+1$	$\frac{df}{dx}=1$
$f_4(x)=-x$	$\frac{df}{dx}=-1$
$f_5(x)=e^x$	$\frac{df}{dx}=e^x$
$f_6(x)=\frac{1}{x}$	$\frac{df}{dx}=\frac{-1}{x^2}$

那么整个函数就被表示成了

$$f_6(f_3(f_5(f_4(f_3(f_2(f_2(f_1(x_0,w_0),f_1(x_1,w_1)),w_2))))))$$

现在逐层反向求导


w0
2.00
-0.20
x0
-1.00
0.40
*
-2.00
0.20
w1
-3.00
-0.40
x1
-2.00
-0.60
*
6.00
0.20
+
4.00
0.20
w2
-3.00
0.20
+
1.00
0.20
*-1
-1.00
-0.20
exp
0.37
-0.53
+1
1.37
-0.53
1/x
0.73
1.00