深度学习

作者：Calvin
QQ: 179209347
Mail: 179209347@qq.com

介绍

笔记简介：

- 面向对象：深度学习初学者
- 依赖课程：**线性代数，统计概率**，优化理论，图论，离散数学，微积分，信息论

知乎专栏：

https://zhuanlan.zhihu.com/p/693738275

Github & Gitee 地址：

https://github.com/mymagicpower/AIAS/tree/main/deep_learning

https://gitee.com/mymagicpower/AIAS/tree/main/deep_learning

*** 版权声明：**

- 仅限用于个人学习
- 禁止用于任何商业用途

神经网络的训练流程

神经网络的训练是指通过反向传播算法来调整神经网络中的参数，使其能够更好地拟合训练数据，从而实现特定的任务。以下是神经网络的训练流程简介：

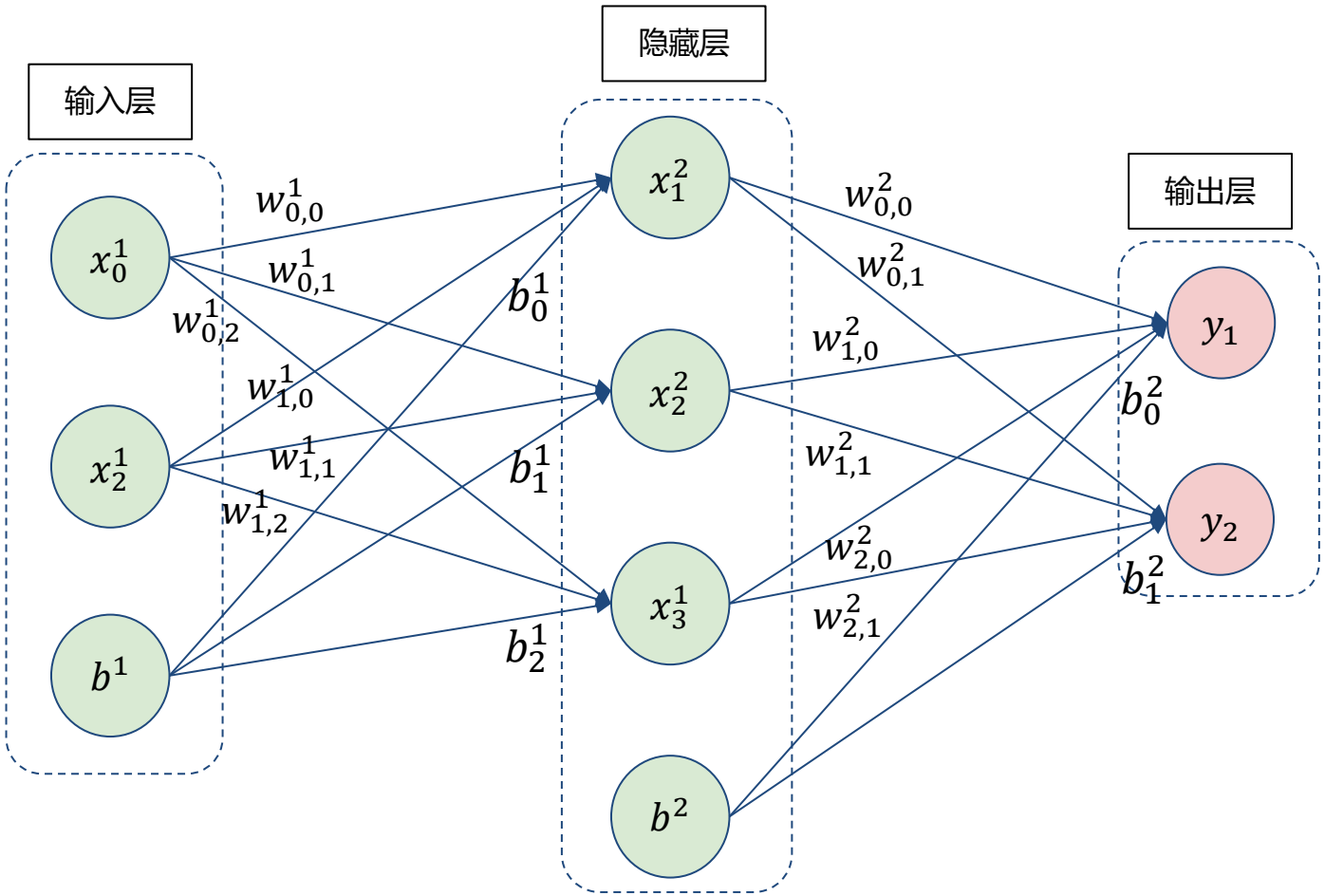
1. 参数初始化：初始化权重 w 和偏置 b。
2. 前向传播：根据给定的输入计算每一层的输出，直到得到最终的预测结果。
3. 计算损失：根据损失函数计算预测值与真实值的差距。
4. 反向传播：通过反向传播算法，计算损失函数对于每个参数的梯度。
5. 参数更新：根据计算得到的梯度，使用优化算法（如梯度下降）来更新神经网络中的参数，使损失函数尽量减小。
6. 重复步骤 2～5，直到达到迭代次数。

参数初始化

初始化参数：首先，需要初始化神经网络中的权重和偏置参数。通常可以使用随机初始化的方法来进行参数初始化。

常见的参数初始化方法：

- 随机初始化（Random Initialization）：
  将参数初始化为小的随机值。
- Xavier初始化（Xavier Initialization）：
  也称为Glorot初始化，它是一种常用的参数初始化方法。
- He初始化（He Initialization）：
  为了解决ReLU激活函数的参数初始化问题而提出的。
- 自适应初始化（Adaptive Initialization）：
  一些特定的神经网络结构可能需要特定的初始化方法。



前向传播算法

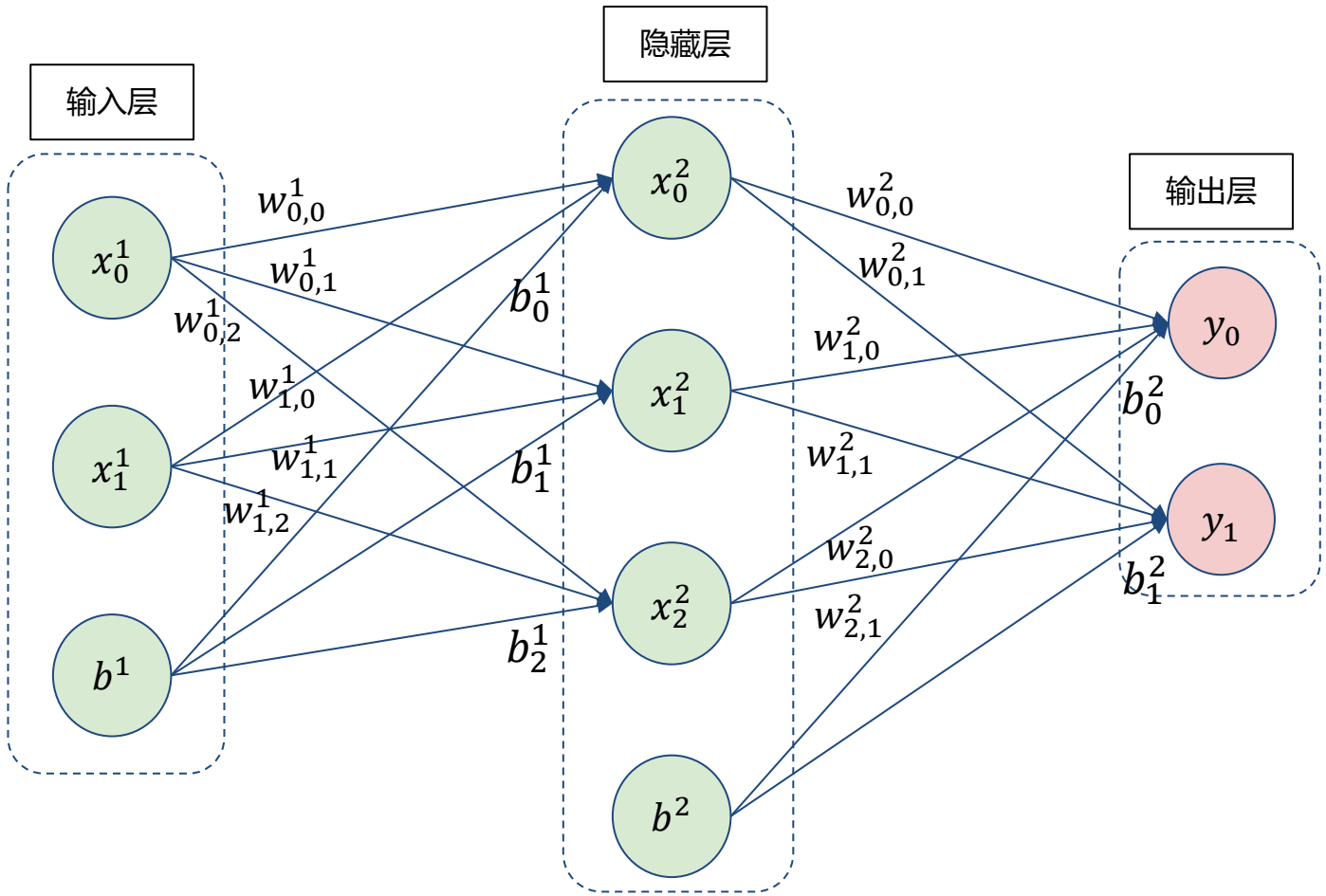
前向传播：通过将训练数据输入神经网络，计算每一层的输出，直到得到最终的预测结果。这个过程称为前向传播。

1. 计算隐藏层：
对隐藏层的输入使用激活函数f()进行处理.

$$x_j^2 = f\left(\sum_{i=0}^{1} w_{i,j}^1 x_i^1 + b_j^1\right)$$

2. 计算输出层：
对输出层的输入使用激活函数f()进行处理.

$$y_k = f\left(\sum_{j=0}^{2} w_{j,k}^2 x_j^2 + b_k^2\right)$$



损失函数

计算损失：将神经网络的预测结果与真实标签进行比较，计算出预测结果与真实标签之间的差距，这个差距通常使用损失函数来表示。损失函数常用的有均方误差和交叉熵误差，也可以自定义损失函数。

均方误差

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - y_i')^2$$

交叉熵误差

$$\mathrm{CEE} = -\sum_{i=0}^{n-1}[y_i \ln(y_i')]$$

- y_i 为第 i 个样本的真实值
- y_i' 为第 i 个样本的预测值
- n 为样本量

矩阵微积分

矩阵微积分是微积分与线性代数的结合，它涉及到矩阵和向量的微分和积分运算。在矩阵微积分中，我们考虑的不再是单变量或多变量函数，而是矩阵值函数或向量值函数。

标量关于向量的偏导数：

$$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x_1}, \cdots \frac{\partial y}{\partial x_p}\right]^{\mathrm{T}},$$

向量关于向量的偏导数：

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_q}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_p} & \cdots & \frac{\partial y_q}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}$$

链式法则

链式法则（Chain Rule）是微积分中的一个重要概念，用于求解复合函数的导数。当一个函数由另外两个函数复合而成时，链式法则描述了如何计算这个复合函数的导数。
设有函数 y = f(u) 和 u = g(x)，则复合函数 y = f(g(x)) 的导数可以表示为：

$$\frac{\partial \boldsymbol{y}}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial \boldsymbol{x}}$$

链式法则的应用非常广泛，特别是在求解复杂函数的导数时非常有用。通过逐步求导并应用链式法则，可以简化复杂函数的导数计算过程。

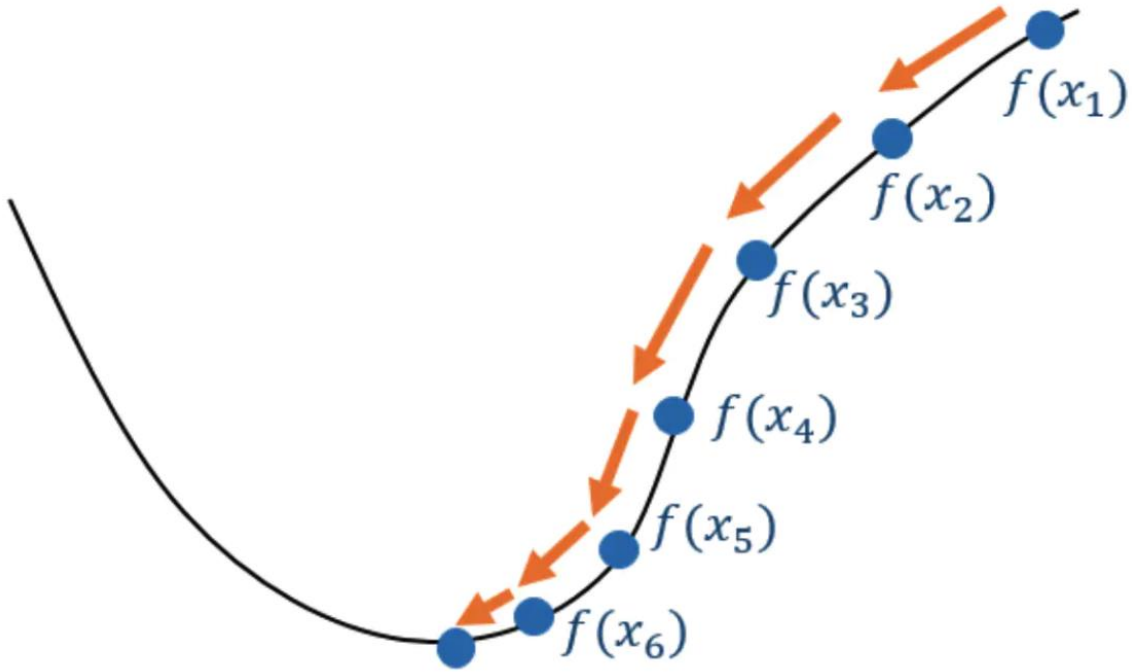
梯度下降算法

梯度下降算法是一种常用的优化算法，用于最小化一个函数的数值，通常用于机器学习和深度学习中的模型训练过程中。其基本思想是通过迭代的方式沿着函数梯度的反方向更新参数，从而逐渐接近函数的局部最小值或全局最小值。

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial \text{Loss}(w, b)}{\partial w}$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial \text{Loss}(w, b)}{\partial b}$$

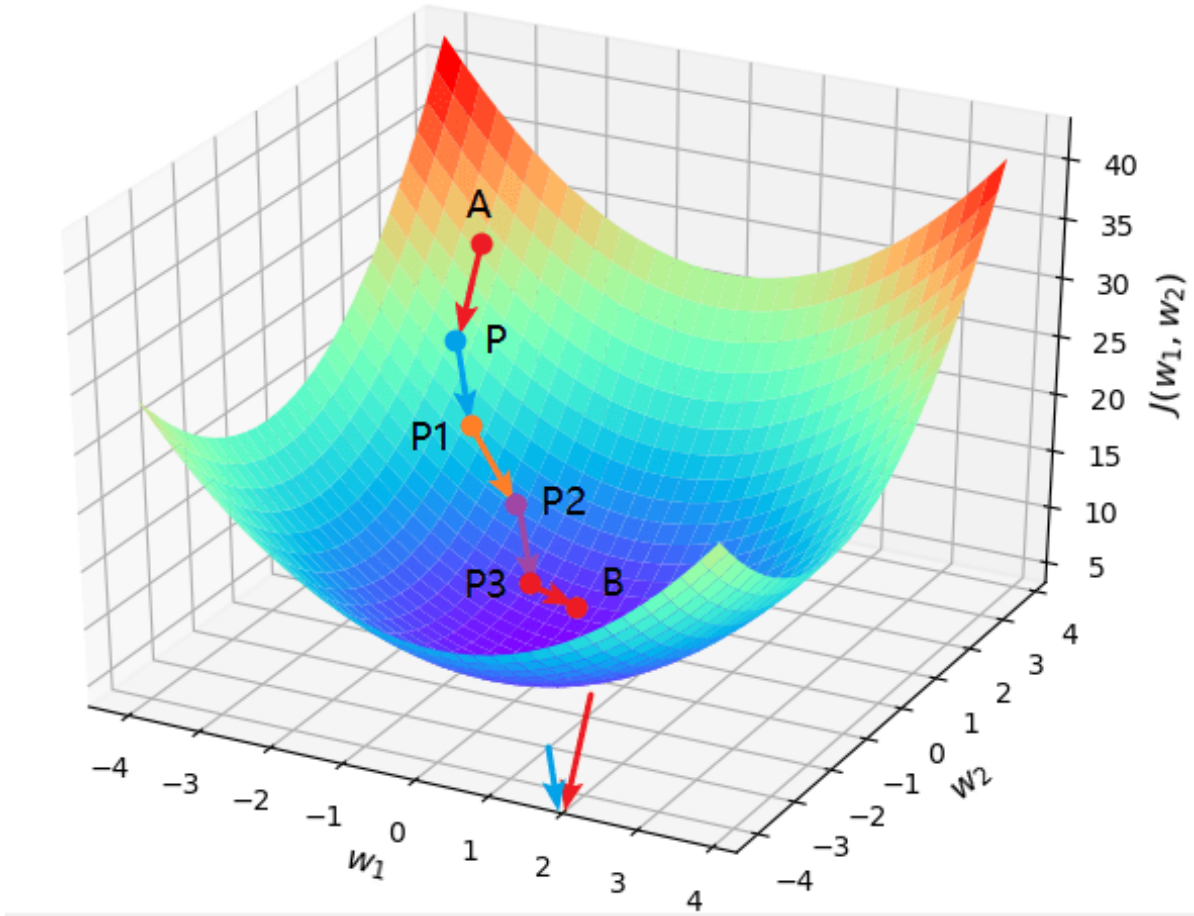
- w 为权重
- b 为偏置
- Loss()为损失函数
- η 为学习率



梯度下降算法

在实际应用中，梯度下降算法有多种变体，如：

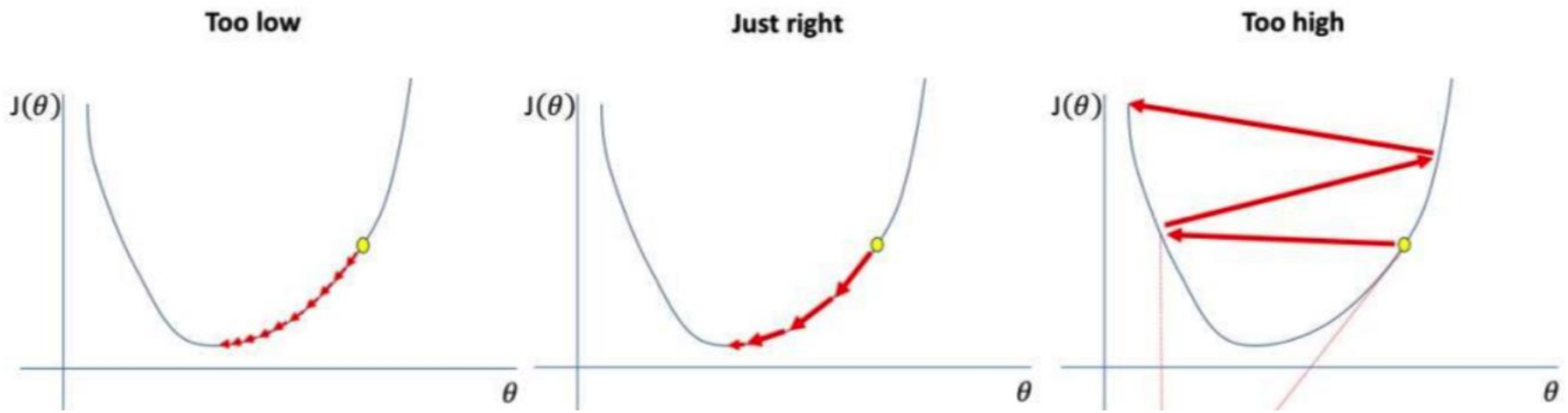
- 批量梯度下降（Batch Gradient Descent）
- 随机梯度下降（Stochastic Gradient Descent）
- 小批量梯度下降（Mini-batch Gradient Descent）



梯度下降算法 – 学习率

学习率是一个通过调整损失函数梯度来改变网络权重的超参数。学习率决定了损失函数能否收敛到局部最小值，以及以多快的速度收敛到最小值。

- 学习率越低，损失函数的变化速度越慢；
- 学习率越高，损失函数的变化速度就越快。



但当学习率过低时，神经网络的损失函数下降速度会很缓慢，不能快速收敛，优化的效率较低。

合适的学习率使损失函数在相对合适的时间内收敛到局部最小值。

当学习率设置得过高，参数更新的幅度就会很大，这样会导致网络快速收敛到局部最优点或者产生振荡，进而可能越过最优点。

反向传播算法

反向传播是指从输出层向输入层反向传播误差信号，调整网络中每个参数以减小损失函数。直到网络输出的误差减小到可以接受的程度。

具体步骤如下：

- 计算输出层的误差：通过损失函数计算出输出层的误差梯度。
- 反向传播误差：将输出层的误差梯度向前传播到隐藏层，计算隐藏层的误差梯度。
- 更新参数：根据误差梯度和学习率等参数，更新网络中的权重和偏置，使损失函数逐渐减小，模型逐渐收敛。

反向传播算法的本质是链式求导法则。

利用链式求导法则，将误差一步步由神经网络输出层向输入层进行传递，再利用梯度下降算法计算每个神经元的权重参数对损失的影响并调整参数的大小。

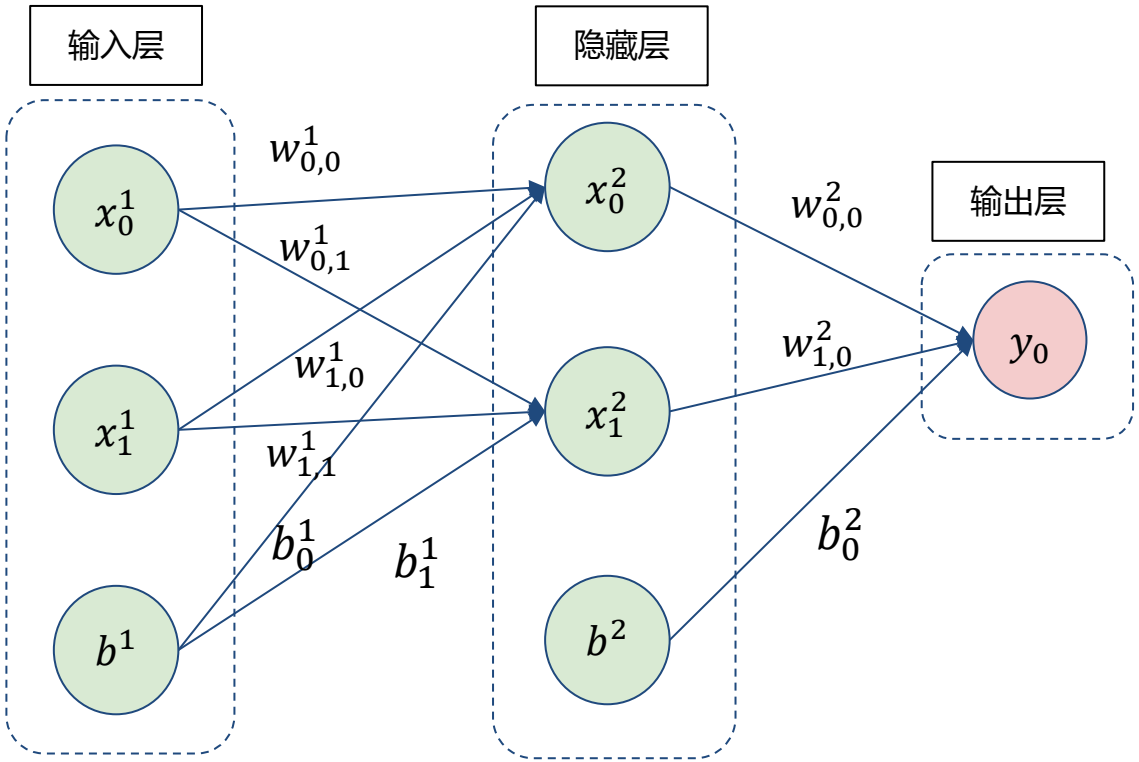
反向传播算法

1. 前向传播：通过将训练数据输入神经网络，计算每一层的输出，直到得到最终的预测结果。

$$u_0 = x_0^1 w_{0,0}^1 + x_1^1 w_{1,0}^1 + b_0^1$$
$$x_0^2 = f(u_0)$$

$$u_1 = x_0^1 w_{0,1}^1 + x_1^1 w_{1,1}^1 + b_1^1$$
$$x_1^2 = f(u_1)$$

$$u_2 = x_0^2 w_{0,0}^2 + x_1^2 w_{1,0}^2 + b_0^2$$
$$y_0 = g(u_2)$$



计算输出层的误差：通过损失函数计算出输出层的误差梯度。

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - y_i')^2$$

f()和g()分别为隐藏层和输出层的激活函数。

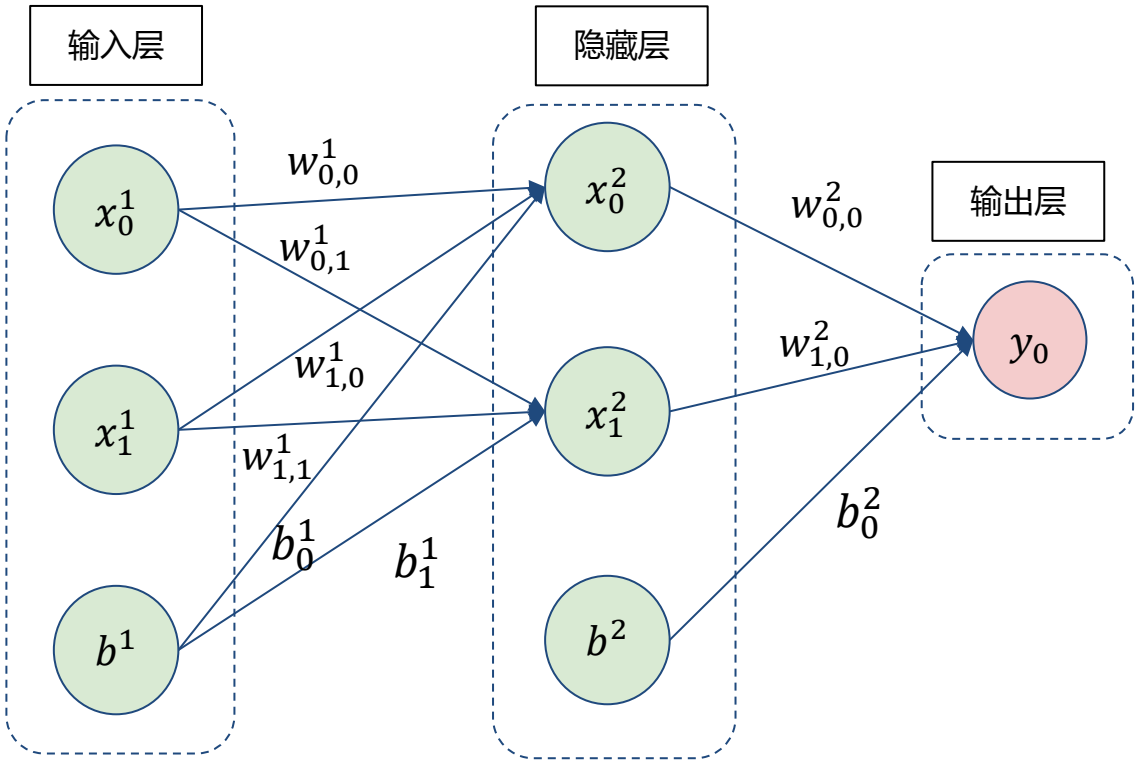
反向传播算法

2. 反向传播：在得到实际输出值 y_0 后，使用损失函数 $\text{Loss}(y_0, y)$ 求得实际输出与真实值之间的误差值。从输出层向输入层逐层使用链式求导法则，计算各个权重和偏置的梯度值。

首先分别求输出层权重 $w_{0,0}^2$ 和 $w_{1,0}^2$，以及偏置 b_0^2 的梯度值。例如， $w_{0,0}^2$ 的梯度计算公式如下：

$$u_2 = x_0^2 w_{0,0}^2 + x_1^2 w_{1,0}^2 + b_0^2$$
$$y_0 = g(u_2)$$

$$\frac{\partial \text{Loss}(y_0, y)}{\partial w_{0,0}^2} = \frac{\partial \text{Loss}(y_0, y)}{\partial g(u_2)} \frac{\partial g(u_2)}{\partial u_2} \frac{\partial u_2}{\partial w_{0,0}^2}$$



f()和g()分别为隐藏层和输出层的激活函数。

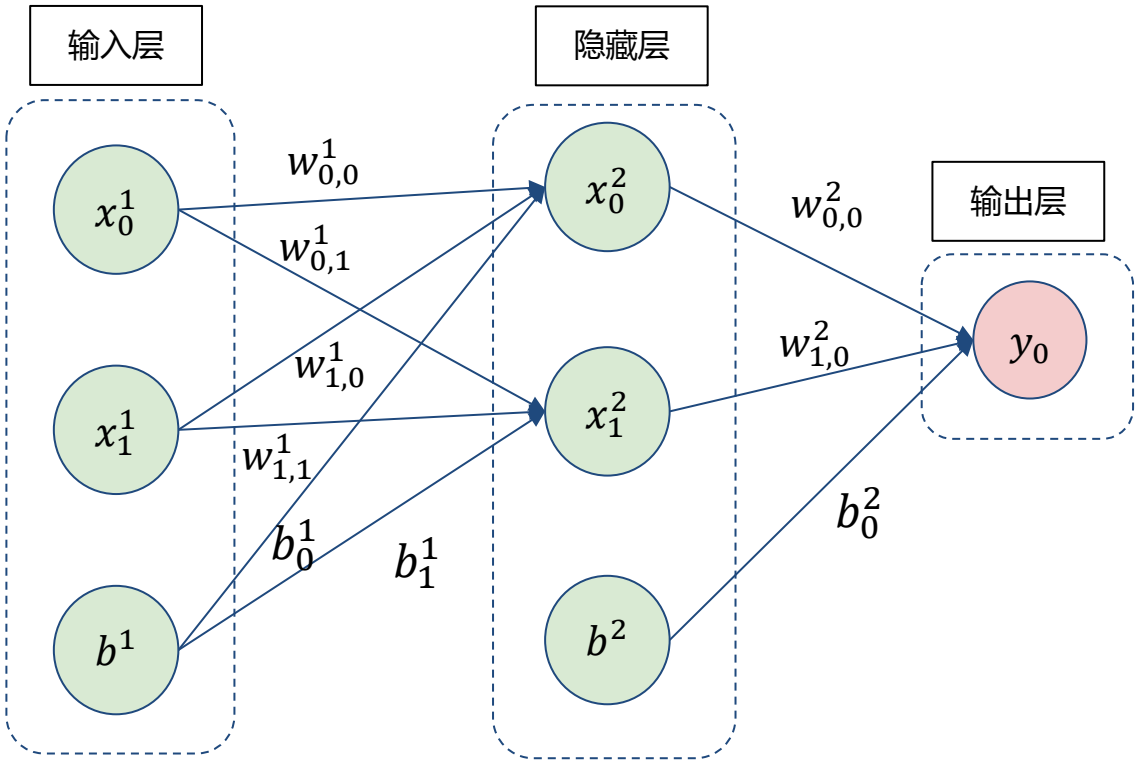
反向传播算法

然后分别求隐藏层权重$w_{0,0}^1$，$w_{0,1}^1$，$w_{1,0}^1$，$w_{1,1}^1$以及偏置b_0^1，b_1^1的梯度值。例如，$w_{0,0}^1$的梯度计算公式如下：

$$u_0 = x_0^1 w_{0,0}^1 + x_1^1 w_{1,0}^1 + b_0^1$$
$$x_0^2 = f(u_0)$$

$$u_2 = x_0^2 w_{0,0}^2 + x_1^2 w_{1,0}^2 + b_0^2$$
$$y_0 = g(u_2)$$

$$\frac{\partial \mathrm{Loss}(y_0, y)}{\partial w_{0,0}^1} = \frac{\partial \mathrm{Loss}(y_0, y)}{\partial g(u_2)} \frac{\partial g(u_2)}{\partial u_2} \frac{\partial u_2}{\partial f(u_0)} \frac{\partial f(u_0)}{\partial u_0} \frac{\partial u_0}{\partial w_{0,0}^1}$$



f()和g()分别为隐藏层和输出层的激活函数。


Thank
You