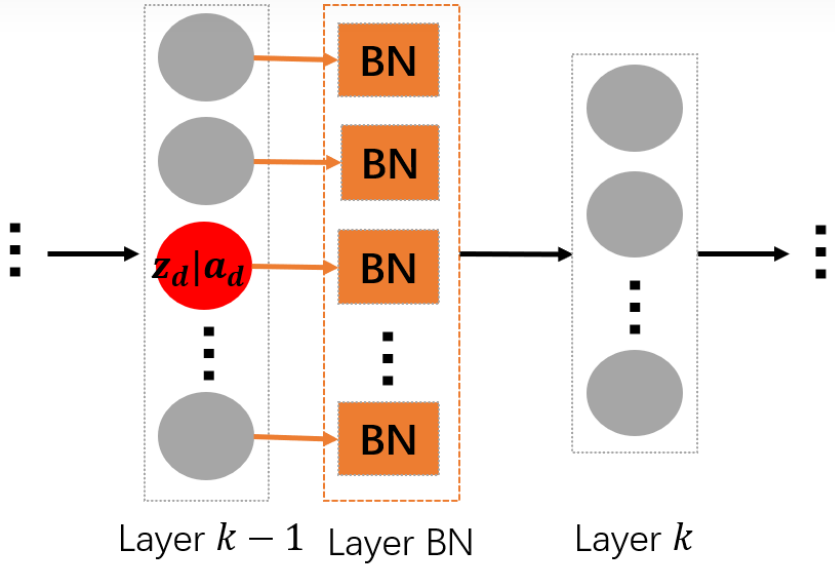
知乎 首发于 ML-DL-RL-GNN算法知识整理



神经网络高频面试题详解

傲娇的小花

综合高频面试题

0、BP的推导

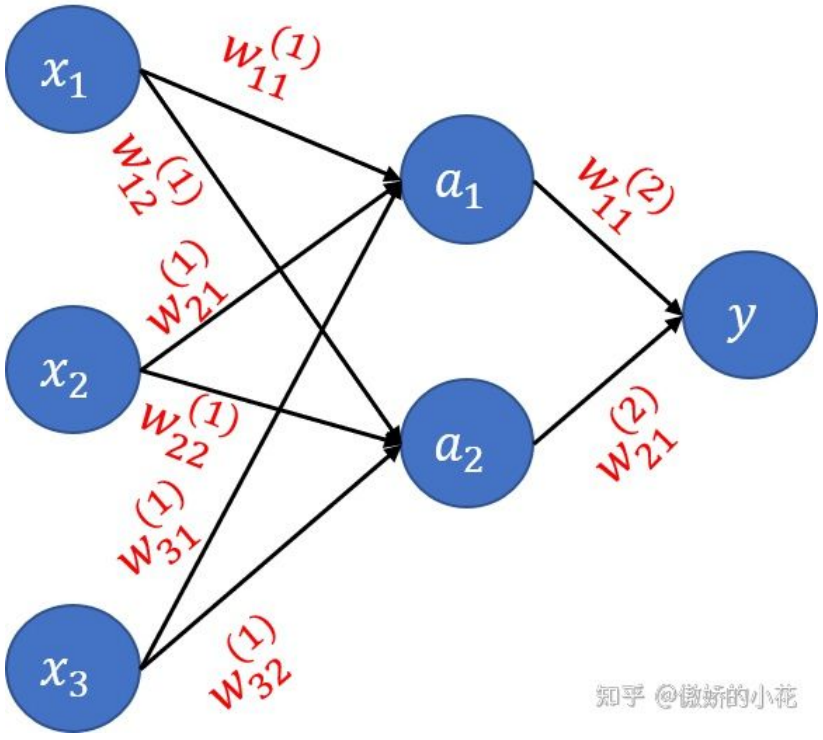


图1

以图1中的三层的神经网络为例：

- 前向传播

$$a_2 = f(z_2) = f(w_{12} * x_1 + w_{22} * x_2 + w_{32} * x_3)$$

$$y = f(o) = f(w_{11}^{(2)} * a_1 + w_{21}^{(2)} * a_2)$$

$f(\cdot)$是激活函数

- 计算损失值 $L(y,t)$，假设 t 是真实值。
- 反向传播，利用链式求导法则计算各个参数的导数，从后往前算每一层的梯度，第二层的参数梯度为：

$$\nabla w_{j1}^{(2)} = \frac{\partial L(y,t)}{\partial y} * \frac{\partial y}{\partial o} * \frac{\partial o}{\partial w_{j1}^{(2)}}$$

$$\nabla a_j = \frac{\partial L(y,t)}{\partial y} * \frac{\partial y}{\partial o} * \frac{\partial o}{\partial a_j}$$

$$j \in \{1,2\}$$

第一层的参数的梯度为：

$$\nabla w_{ij}^{(1)} = \nabla a_j * \frac{\partial a_j}{\partial z_j} * \frac{\partial z_j}{\partial w_{ij}^{(1)}}$$

$$i \in \{1,2,3\}$$

代入具体的损失函数和激活函数就可以计算出各项的导数值了，如损失函数使用平方误差，激活函数使用sigmoid函数：

$$L(y,t) = \frac{1}{2}(y-t)^2$$

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

计算损失函数和激活函数的梯度得到：

$$\frac{\partial L(y,t)}{\partial y} = t - y$$

$$\frac{\partial f(z)}{\partial z} = f(z)(1-f(z))$$

最后求得各参数的梯度为：

$$\nabla w_{j1}^{(2)} = (t-y) * f(o)(1-f(o)) * a_j$$

$$\nabla a_j = (t-y) * f(o)(1-f(o)) * w_{j1}^{(2)}$$

$$\nabla w_{ij}^{(1)} = \nabla a_j * f(z_j)(1-f(z_j)) * x_i$$

2、神经网络的权重能初始化为0吗?

神经网络的权重不能初始化为0，以上题中的神经网络为例，损失函数是平方误差激活函数是sigmoid函数情况下，如果参数初始化为0：

前向传播过程中，所有激活神经元的输出值都是一样的，即 $a_1 = a_2 = \frac{1}{2}$，意味着网络变得跟只有一个隐层节点一样，网络失去了提取不同特征的能力。

在反向传播过程中，因为 $w_{j1}^{(2)} = w_{ij}^{(1)} = 0$，所以 $\nabla a_j = 0$，导致 $\nabla w_{ij}^{(1)} = 0$，即除了最后一层的参数能得到更新，前面的参数都无法更新。即使最后一层的参数得以更新，但是其参数更新后仍然是相等的，网络几乎学不到东西。

1. **sigmoid函数**

- 优点
  - 函数范围是[0,1]，能把输入值变换为0和1之间的连续值，用于输出层可以理解为概率值，从而使得输出结果具备可解释性。

- 缺点
  - sigmoid函数的输出值不是0均值的，$k+1$层输入k层的非0均值信号x，在计算损失函数对$k+1$层的参数w的梯度时，梯度值始终是正的，反向传播时参数要么都是往正方向更新，要么都往负方向更新，参数将呈锯齿形收敛，导致收敛缓慢。
  - 容易饱和，导数范围是[0,0.25]，用在深度神经网络中，反向传播时容易导致梯度消失问题。
  - sigmoid函数中有幂运算，计算机求解时相对来讲比较耗时。用在规模比较大的深度网络会增加训练时间。

2. **tanh函数**

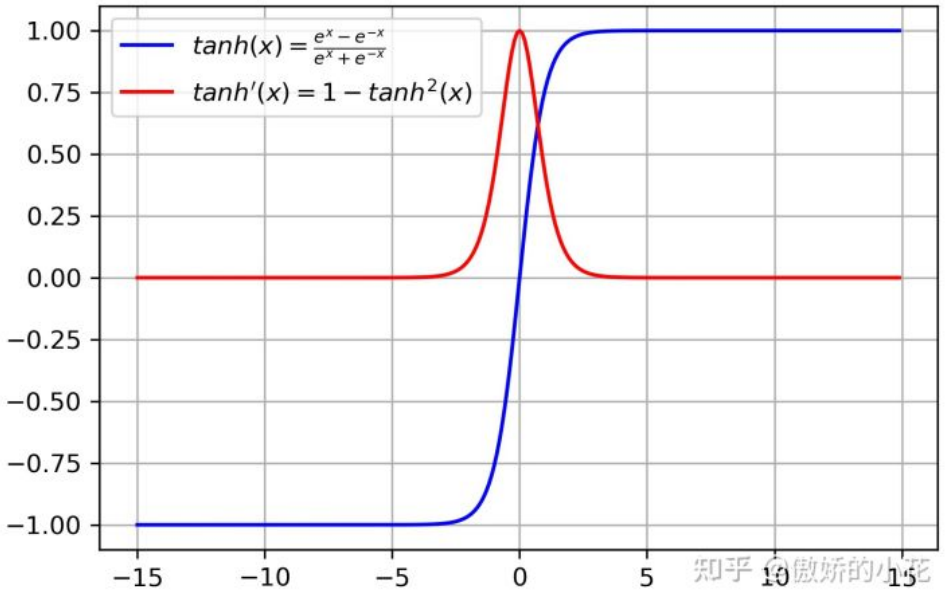


图3：tanh函数

- 优点
  - tanh函数范围是[-1,1]，因此能得到0均值信号输出，解决了sigmoid的非0均值问题
  - 导数范围[0,1]，相比于sigmoid函数的导数范围要大，一定程度缓解了梯度消失问题

- 缺点
  - 仍然面临梯度消失问题
  - 与sigmoid一样含有幂运算，在大规模网络中会增加训练时间。

3. **relu**

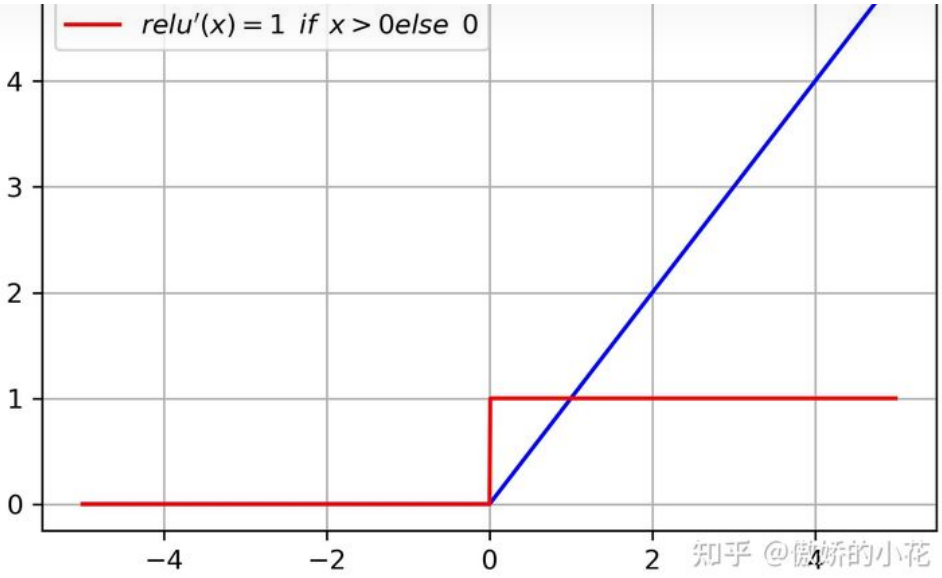


图4：relu函数

- 优点
  - 仅是max运算，计算简单，高效
  - 在区正间始终存在梯度值，解决了梯度消失问题
  - 收敛速度比sigmoid和tanh快

- 缺点
  - 输出不是0均值
  - 在小于0的区间梯度值为0，导致有些神经元无法被更新，该问题也被称为**dead relu problem**

4. **leaky_relu**

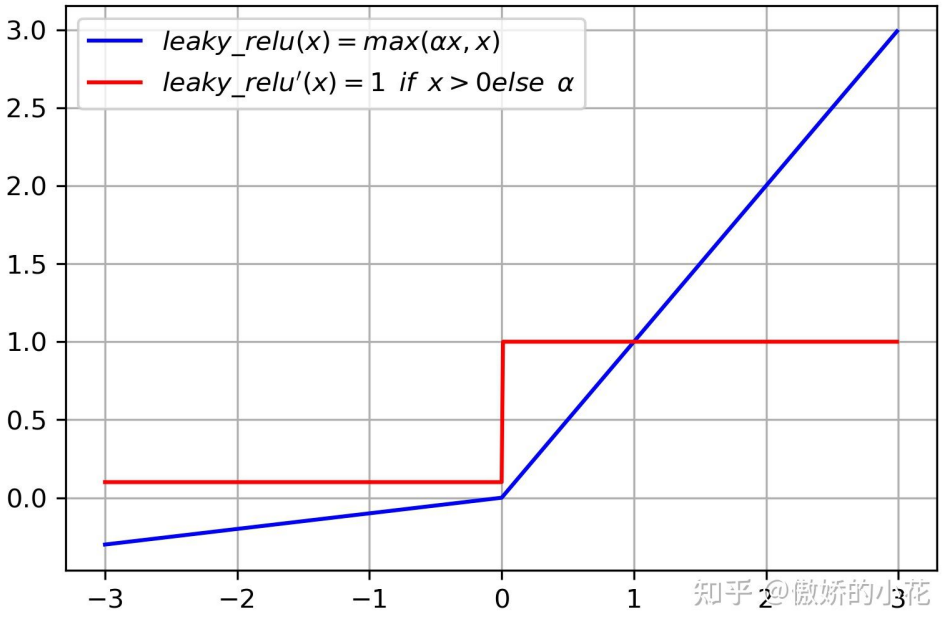


图5：leaky_relu

- 优点
  - relu的优点它都有
  - 在小于0的区间仍然存在小的梯度值，解决了dead relu problem。

3、梯度消失、梯度爆炸的现象、原因、解决方法

	梯度消失	梯度爆炸
现象	反向传播过程梯度值变为0，导致参数无法更新	反向传播过程梯度值过大，甚至溢出
原因	1、激活函数不合适，如sigmoid、tanh 2、网络过深，如果梯度值小于1 ，反向传播过程梯度更新信息将以指数形式衰减 3、初始值太小，反向传播过程使得梯度值原来越小	1、网络过深，如果梯度大于1，梯度更新信息将以指数级增长 2、初始值过大，反向传播过程使得梯度值原来越大
解决方法	1、使用relu或者leaky_relu等激活函数 2、加BN层，保证网络稳定性 3、使用残差连接 4、如果是RNN网络则换成LSTM	1、梯度裁剪，使得梯度值有上限 2、使用权重正则化方法，更新过程中不断降低权重

4、神经网络不收敛的现象、原因、解决方法

- 不收敛的现象：训练过程中的损失值无明显下降趋势或者甚至上升
- 原因：

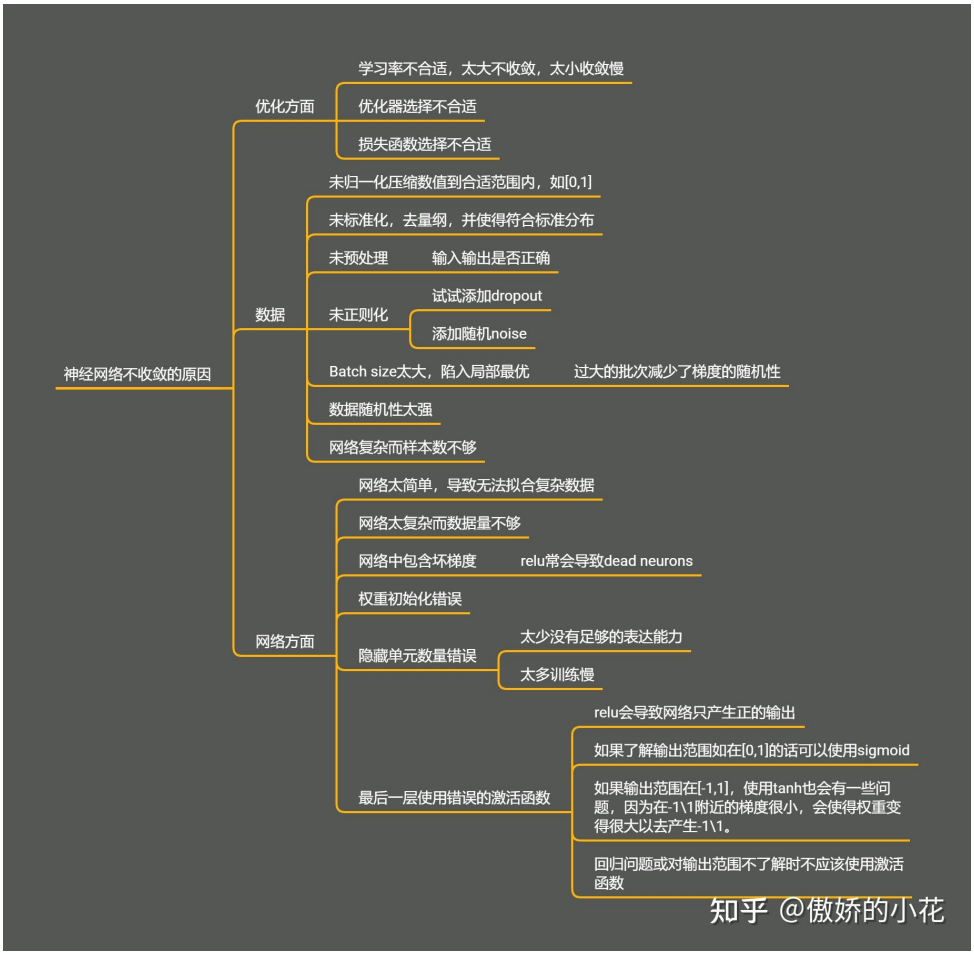


图6

当检查出不收敛的原因之后，对应的解决方法也就知道了，比如数据少的话，就通过增加数据量、数据增强或者减少网络规模等方法解决。

5、BN层的作用

- BN有正则化作用，可以无需额外使用dropout来避免过拟合，从而提高泛化能力。
- BN对不同的初始化机制和学习率更鲁棒，即降低了对初始化的要求。

什么是**internal covariate shift**问题？具体表现是什么呢？

在训练过程中，由于参数的变化，每层的输入数据的均值和协方差一直在变化。

比如 $A^{[k]} = f(Z^{[k]}) = f(W^{[k]}A^{[k-1]} + b^{[k]})$，随着参数 $W^{[k]}$ 和 $b^{[k]}$ 的不断更新，输入 $Z^{[k]}$ 的分布发生变化，作为 $k+1$ 层输入的 $A^{[k]}$ 的分布也随之发生变化。这意味着后一层需要调整自己以适应输入数据分布的变化，从而导致神经网络的训练速度降低。同时，下层输入的变化可能趋于变大或变小，导致落入激活函数的饱和区。

6、BN是怎么做的？公式是什么？训练和测试的时候怎么处理？训练时需要记住每个批次的均值和方差吗？

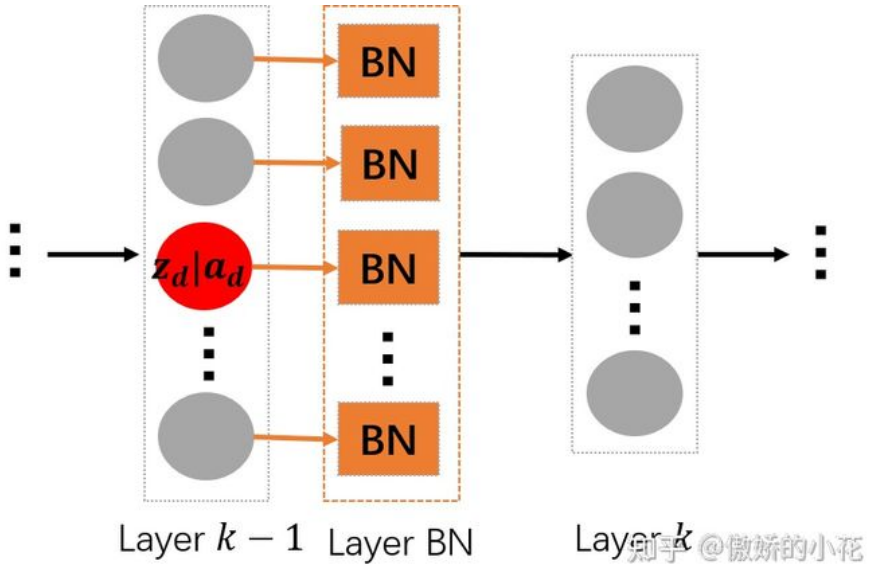


图7：BN

BN层一般加在隐层之间，基本思想是对 $k-1$ 层的输出进行规范化，固定 k 层输入的均值和协方差。因为神经网络一般使用**批次梯度下降法**更新参数，所以训练过程中无法使用全局信息，而是使用每个批次的信息进行正则化。

BN是对每个神经元都进行规范化，我们以图7中红色神经元为例，来说明BN的基本步骤，

- 假设batch_size= m，计算输入到第 d 个神经元的数据的均值和协方差

$$\mu_d = \frac{1}{m}\sum_{i=1}^{m} z_i^d$$

$$\sigma_d^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i^d - \mu_d)^2$$

- 正则化

$$\hat{z}_i^d = \frac{z_i^d - \mu_d}{\sqrt{\sigma_d^2 + \epsilon}}$$

> ϵ 是为了避免除0操作。经过第二步的正则化操作后，第 d 个神经元的输入分布 $\hat{z}^d \sim \mathcal{N}(0, 1)$，变为均值为0方差为1的标准正态分布。其它神经元经过正

- 线性变换

$$\tilde{z}_i^d = \gamma_d \hat{z}_i^d + \beta_d$$

γ_d, β_d 都是训练过程需要学习的参数，经过第三步之后，数据分布变为均值为 β_d，方差为 γ_d 的分布，相当于对标准正太分布进行了平移和缩放。

为什么需要第三步线性变换呢?

因为第二步得到均值为0方差为1的输入在经过sigmoid激活函数或者tanh激活函数时，容易陷入线性区域，从而失去非线性的特征表达能力。第三步线性变换是将标椎正太分布进行缩放和平移，从而让输入能更多落在非线性区域，从而获得非线性的表达能力。

那么BN在训练时测试时都是怎么处理的呢?

测试时一般是针对单个样本或者少数几个样本，如果针对这几个样本计算 $< \mu_d, \sigma_d^2 >$ 的话，会得到有偏估计。**因此训练时，神经网络需要记住每个批次算得的** $< \mu_{batch,d}, \sigma_{batch,d}^2 >$，然后统计在整个训练数据上的统计量用于对测试数据进行标准化:

$$\mu_{test,d} = \mathbb{E}[\mu_{batch,d}]$$
$$\sigma_{test,d}^2 = \frac{m}{m-1}\mathbb{E}[\sigma_{batch,d}^2]$$

6、BN、LN的区别

BN中以mini-batch的统计量作为整体统计量的近似估计，这就使得BN受mini-batch大小的约束。如果batch较小的话，会使得数据分布差别很大，增加模型训练复杂度。一般需要mini-batch的size较大，让批次间的数据分布较接近，并且训练前要做好充分的shuffle。并且因为要统计 μ 和 σ，不适用动态网络结构和RNN网络。

LN（Layer Normalization）在计算均值和协方差时，通过统计每层中所有神经元的数据来计算，不受mini-batch大小的约束，适合RNN结构的网络。

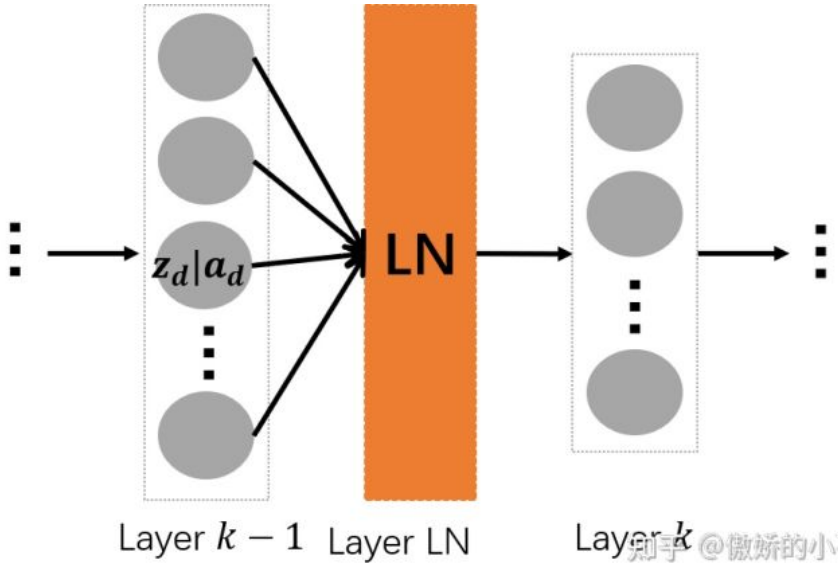


图8：LN

假设 $k-1$ 层有 H 个神经元，

- LN的均值和协方差计算如下:

$$\sigma^2 = \frac{1}{H}\sum_{d=1}^{H}(z_d - \mu)^2$$

- 对每个神经进行正则化

$$\hat{z}_d = \frac{z_d - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

- 平移变换

$$\tilde{z}_d = \gamma \hat{z}_d + \beta$$

可以看到，在LN中，所有神经元的μ和σ都是一样的，不受到批次的约束。并且训练和测试的处理方式是一样的，不用记住训练时的统计量，节省存储空间。

不过LN中所有神经元的输入都在同一区间范围内，一定程度上降低了模型的表达能力。

7、dropout的原理

使用小数据集训练较大规模的神经网络容易出现过拟合风险。dropout是一种正则化方法，相等于使用相同的数据同时训练不同的模型，能够有效避免过拟合。

dropout可以用在除了输出层外的各个层中，在训练时，以一定的概率将神经元的输出设置为0，被dropped out的神经元不参与本次迭代的前向传播过程，也不参与反向传播参数更新过程。

网络的权值会因为dropout而大于正常的权值。因此，在最终确定网络之前，每个权重w首先根据所选的dropout概率p进行缩放，即$w = pw$。然后，该网络就可以正常地进行预测。所以dropout仅用在训练时，不用于测试时。

dropout每次迭代都会尝试一个新的结构，降低了神经元间的互适应关系，迫使神经元学习更为鲁棒的特征。

8、神经网络的优化器

1. Batch gradient descent（BGD）

$$w = w - \eta_w \cdot \nabla J(w)$$

- 基本思想：使用整个数据集来计算梯度
- 优点：
  - 梯度计算准确；
  - 下降过程稳定；
  - 对凸损失函数能保证收敛到全局最小值，非凸损失函数能保证收敛到局部最小值。
- 缺点：
  - 如果数据集含有许多相似或者重复样本，则梯度计算冗余
  - 计算耗时；
  - 收敛慢；
  - 如果整个数据集较大放不进内存则无法计算梯度；
  - 不可用于在线学习算法。

2.Stochastic gradient descent（SGD）

- 基本思想：随机选择一个样本 (x_i, y_i) 计算负梯度，沿着负梯度方向更新参数，步长随着迭代次数慢慢变小。
- 优点
  - 避免了BGD梯度计算的冗余问题
  - 梯度计算更快
  - 可用于在线学习算法，来了新样本就可以计算梯度更新模型
  - 随机性可以帮助跳出局部最小值，使用模拟退火方法在迭代过程中不断减小步长可以慢慢逼近最小值

- 缺点
  - 梯度方差大，容易上下震荡，到达最值点的路径比较曲折

3. Mini-batch gradient descent（MBGD）

$$w = w - \eta \cdot \nabla_w J(w; \{x_i\}_{i=1}^m, \{y_i\}_{i=1}^m)$$

- 基本思想：每次更新参数时，使用小批次的数据来计算梯度。
- 优点
  - 减少梯度的方差，收敛过程比SGD更稳定
  - 梯度计算比BGD效率更高

- 缺点
  - 不能保证好的收敛性
  - 选择合适的学习率 η 困难，太大收敛过程容易震荡，或者收敛不到最小值，太小收敛慢
  - 收敛过程中学习率的变化需要事前确定并适应于收敛过程
  - 所有的参数使用一样的学习率，不同特征的量纲不同，对结果的影响也不同，所有参数都使用一样的学习率进行相同幅度的更新容易导致陷入局部最优。
  - 后三个关于学习率的缺点**BGD\SGD**也有

4. Momentum

$$v_t = \beta v_{t-1} + \eta \cdot \nabla_w J(w)$$
$$w = w - v_t$$

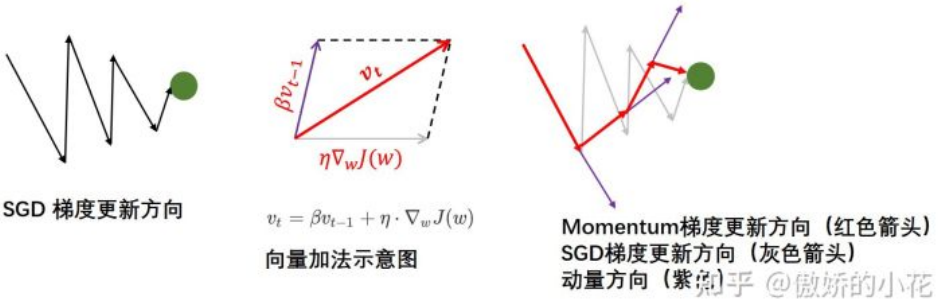


图9：SGD和Momentum的更新过程

SGD算法连续两次的梯度更新方向没有相关性，所以在更新过程中容易震荡，造成收敛慢。Momentum则使得每次参数更新方向不仅取决于负梯度还受到上一次更新方向的影响。如图9，从Momentum的更新过程可以看到，如果当前负梯度方向与上次更新方向偏离太大，那么加上动量向量后，就可以将本次的更新方向拉回来。反之，如果负梯度方向与上次的更新方向相似，那么加上动量向量后，本次的更新方向会更大。动量在迭代过程中不断积累能量，从而加速收敛。

- 基本思想：对原始梯度做平滑，加速收敛。
- 优点：收敛更快
- 缺点：关于学习率的三个缺点未解决

$$g_{t,i} = \nabla_w J(w_{t,i})$$
$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{\sum_{j=1}^{t}((g_i)_j)^2 + \epsilon}} \cdot g_{t,i}$$

- 基本思想：根据前几轮迭代的历史梯度值动态地调整学习率，且参数的每个分量都有自己的学习率
- 优点
  - 无需手动调优学习率
  - 对于简单的二次问题，表现良好

- 缺点
  - 随着时间的积累，分母越来越大，学习率迅速减少趋于0，导致参数无法更新
  - 学习率迅速减少会降低学习速度，难以收敛到全局最优

6. RMSprop

$$g_{t,i} = \nabla_w J(w_{t,i})$$
$$c_i = \gamma c_{i-1} + (1-\gamma) g_{t,i}^2$$
$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{c_i + \epsilon}} \cdot g_{t,i}$$

- 基本思想：仅积累最近迭代的梯度来修正，加了一个衰减因子，解决了Adagrad学习率迅速减少的问题。

7. Adam

其实是Momentum+RMSProp，前两个公式类似于Momentum的动量积累公式，分别积累迭代过程中的一阶梯度和二阶梯度。m_t 和 v_t 初始化为0向量时，如果 β_1 和 β_2 接近1，在迭代的初始阶段 m_t 和 v_t 会偏向于0，所以分别使用第三个公式和第四个公式做修正。最后的梯度更新公式就类似于RMSProp，每个参数都有自己的学习率。

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

- 优点
  - 自适应调整学习率
  - 在实践中表现良好，并优于其他自适应学习方法算法。

其它面试题整理：

知乎 首发于 ML-DL-RL-GNN算法知识整理

发布于 2021-02-06 19:34

神经网络 BP算法 激活函数

文章被以下专栏收录

ML-DL-RL-GNN算法知识整理
整理ML-DL-RL-GNN相关的算法理论和面试题

推荐阅读

深度学习中常用的激活函数详解

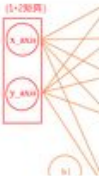
什么是激活函数？如图1，在神经元中，输入的 inputs 通过加权，求和后，还被作用了一个函数，这个函数就是激活函数。引入激活函数是为了增加神经网络模型的非线性。如果不用激活函数，每一...

小糊糊 发表于每天学习一...

深度学习领域最常用的10个激活函数，一文详解数学原理及...

本文转自机器之心 作者：Sukanya Bag激活函数是神经网络模型重要的组成部分，本文作者Sukanya Bag从激活函数的数学原理出发，详解了十种激活函数的优缺点。 激活函数（Activation Function...

极市平台 发表于极市平台



神经网络向传播到

Mr.看海

2 条评论 切换为时间排序

写下你的评论...

早睡早起 2021-10-14
太强了
赞

知乎用户 2021-02-06
总结得真好
赞