

机器爱学习

- 专注机器学习、深度学习及其应用

博客园
新随笔
订阅

首页
联系
管理

随笔 - 66 文章 - 0 评论 - 8

昵称：AI-ML-DL
园龄：10个月
粉丝：15
关注：0
+加关注

<	2017年10月						>
日	一	二	三	四	五	六	
24	25	26	27	28	29	30	
1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	
29	30	31	1	2	3	4	

搜索

找找看

深度学习补充和总结

一、损失函数

深度学习中，常用的损失函数为均方误差和交叉熵，分别对应回归和分类问题，其实深度学习的损失函数和机器学习的损失函数差不多，是一致的，均方误差就相当于最小二乘，交叉熵其实是一种特殊的对数损失函数形式，这里不再赘述。

二、激活函数

是深度学习特有的。

关于激活函数，首先要搞清楚的问题是，激活函数是什么，有什么用？不用激活函数可不可以？答案是不可以。激活函数的主要作用是提供网络的非线性建模能力。如果没有激活函数，那么该网络仅能够表达线性映射，此时即便有再多的隐藏层，其整个网络跟单层神经网络也是等价的。因此也可以认为，只有加入了激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。那么激活函数应该具有什么样的性质呢？

非线性：当激活函数是线性的时候，一个两层的神经网络就可以逼近基本上所有的函数了。但是，如果激活函数是恒等激活函数的时候(即 $f(x)=x$)，就不满足这个性质了，而且如果MLP使用的是恒等激活函数，那么其实整个网络跟单层神经网络是等价的。

可微性：当优化方法是基于梯度的时候，这个性质是必须的。

单调性：当激活函数是单调的时候，单层网络能够保证是凸函数。

$f(x)=x$ ：当激活函数满足这个性质的时候，如果参数的初始化是random的很小的值，那么神经网络的训练将会很高效；如果不满足这个性质，那么就需要很用心的去设置初始值。

输出值的范围：当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表达受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况下，一般需要更小的learning rate。

这些性质，也正是我们使用激活函数的原因！

Activation Functions.

谷歌搜索

常用链接

[我的随笔](#)
[我的评论](#)
[我的参与](#)
[最新评论](#)
[我的标签](#)

随笔分类

[CV\(35\)](#)
[DL\(10\)](#)
[ML\(20\)](#)
[NLP\(1\)](#)

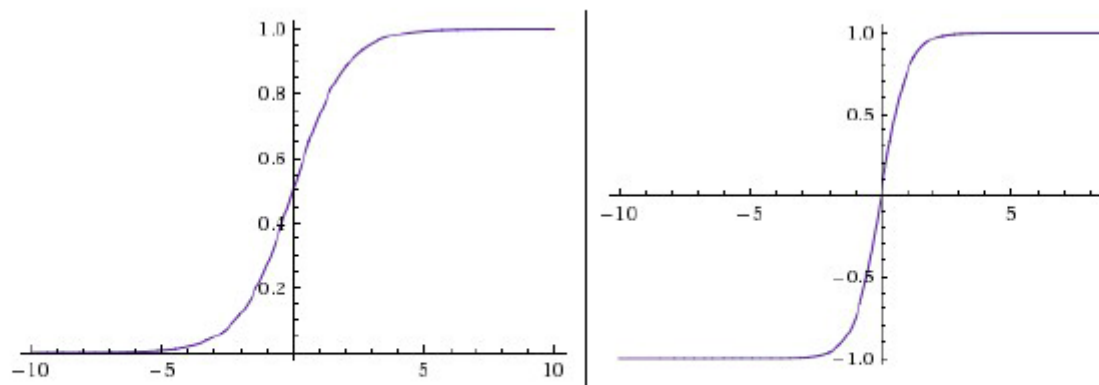
随笔档案

[2017年3月 \(5\)](#)
[2017年2月 \(19\)](#)
[2017年1月 \(8\)](#)
[2016年12月 \(23\)](#)
[2016年11月 \(11\)](#)

最新评论

1. Re:生成对抗式网络
详细
--StudyAI_com
2. Re:CV : object detection(YOLO)
@马春杰杰 可以一起交流...
--flysnow_88
3. Re:CV : object detection(YOLO)
@flysnow_88还没有呢，现在在看SSD了...
--马春杰杰
4. Re:CV : object detection(YOLO)

Sigmoid



Left: Sigmoid non-linearity squashes real numbers to range between [0,1] Right: linearity squashes real numbers to range between [-1,1].

Sigmoid 是常用的非线性的激活函数，它的数学形式如下： $f(x)=\frac{1}{1+e^{-x}}$

正如前一节提到的，它能够把输入的连续实值“压缩”到0和1之间。

特别的，如果是非常大的负数，那么输出就是0;如果是非常大的正数，输出就是1.

sigmoid 函数曾经被使用的很多，不过近年来，用它的人越来越少了。主要是因为它的一些 缺点：

Sigmoids saturate and kill gradients. (saturate 这个词怎么翻译?饱和?)sigmoid 有一个非常致命的缺点，当输入非常大或者非常小的时候(saturation)，这些神经元的梯度是接近于0的，从图中可以看出梯度的趋势。所以，你需要尤其注意参数的初始值来尽量避免saturation的情况。如果你的初始值很大的话，大部分神经元可能都会处在saturation的状态而把gradient kill掉，这会导致网络变的很难学习。

Sigmoid 的 output 不是0均值. 这是不可取的，因为这会导致后一层的神经元将得到上一层输出的非0均值的信号作为输入。

产生的一个结果就是：如果数据进入神经元的时候是正的(e.g. $x>0$ elementwise in $f=wTx+b$)，那么 w 计算出的梯度也会始终都是正的。

@马春杰杰你好：想问下，你更改了源码没？可以输出每一类的recall,AP,以及mAP了吗？我也在做这一步。...

--flysnow_88

5. Re:CV : object detection(YOLO)

@马春杰杰recall和mAP都是分类任务的指标，只是需要针对多标签任务进行一些修改，具体的，百度即可知道...

--AI-ML-DL

阅读排行榜

1. LSTM与GRU结构(8609)
2. 聚类算法 (clustering) (3629)
3. CV : object recognition(ZFNet)(3615)
4. 生成对抗式网络(2711)
5. CV : image caption(Show, Attend and Tell: Neural Image Caption Generation with Visual Attention)(1701)

评论排行榜

1. CV : object detection(YOLO)(5)
2. 时间序列分析(1)
3. 聚类算法 (clustering) (1)
4. 生成对抗式网络(1)

推荐排行榜

1. 时间序列分析(2)
2. CV : object recognition(ZFNet)(1)
3. LSTM与GRU结构(1)
4. 聚类算法 (clustering) (1)

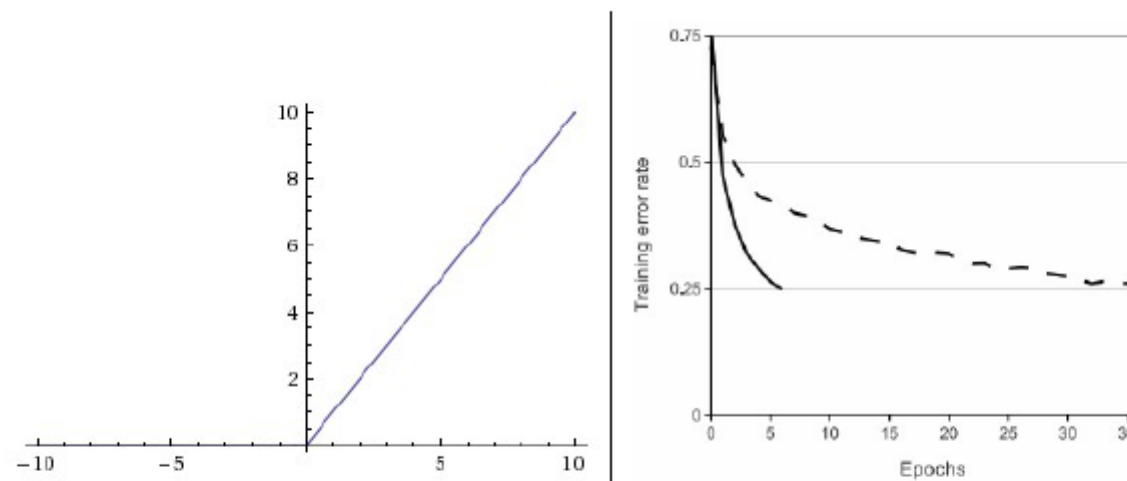
当然了，如果你是按batch去训练，那么那个batch可能得到不同的信号，所以这个问题还是可以缓解一下的。因此，非0均值这个问题虽然会产生一些不好的影响，不过跟上面提到的 kill gradients 问题相比还是要好很多的。

tanh

tanh 是上图中的右图，可以看出，tanh 跟sigmoid还是很像的，实际上，tanh 是sigmoid的变形：

$$\tanh(x) = 2\text{sigmoid}(2x) - 1$$

与 sigmoid 不同的是，tanh 是0均值的。因此，实际应用中，tanh 会比 sigmoid 更好(毕竟去粗取精了嘛)。



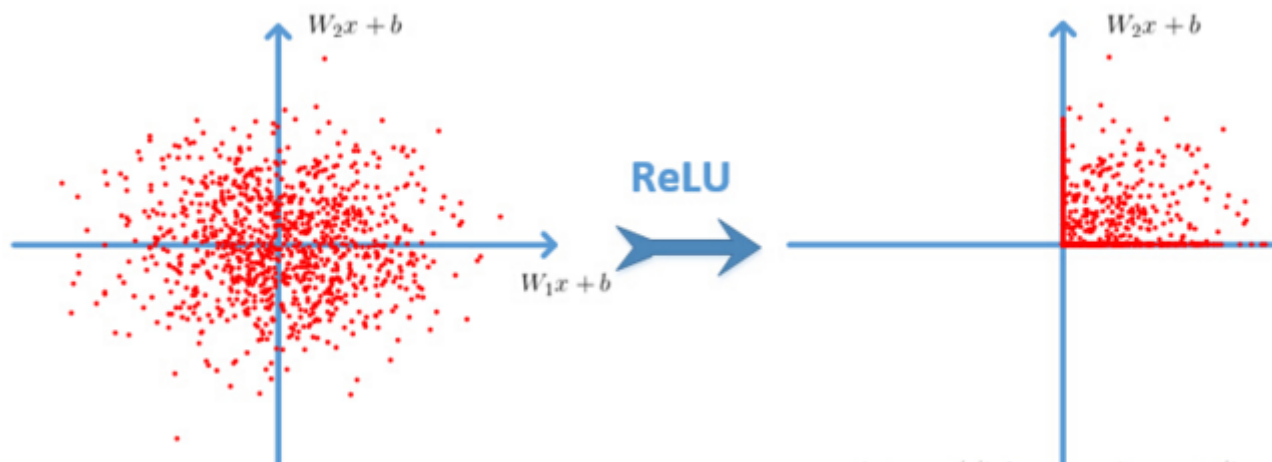
Left: Rectified Linear Unit (ReLU) activation function, which is zero when $x < 0$ and has slope 1 when $x > 0$. **Right:** A plot from Krizhevsky et al. (pdf) paper indicating the 6x in convergence with the ReLU unit compared to the tanh unit.

ReLU

近年来，ReLU 变的越来越受欢迎。它的数学表达式如下：

$$f(x) = \max(0, x)$$

很显然，从图左可以看出，输入信号 <0 时，输出都是0， >0 的情况下，输出等于输入。 w 是二维的情况下，使用ReLU之后的效果如下：



ReLU 的优点：

Krizhevsky et al. 发现使用 ReLU 得到的SGD的收敛速度会比 sigmoid/tanh 快很多(看右图)。有人说这是因为它是 linear，而且 non-saturating

相比于 sigmoid/tanh，ReLU 只需要一个阈值就可以得到激活值，而不用去算一大堆复杂的运算。

ReLU 的缺点：当然 ReLU 也有缺点，就是训练的时候很“脆弱”，很容易就“die”了。什么意思呢？

举个例子：一个非常大的梯度流过一个 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了。

如果这个情况发生了，那么这个神经元的梯度就永远都会是0。

实际操作中，如果你的learning rate 很大，那么很有可能你网络中的40%的神经元都“dead”了。

当然，如果你设置了一个合适的较小的learning rate，这个问题发生的情况其实也不会太频繁。

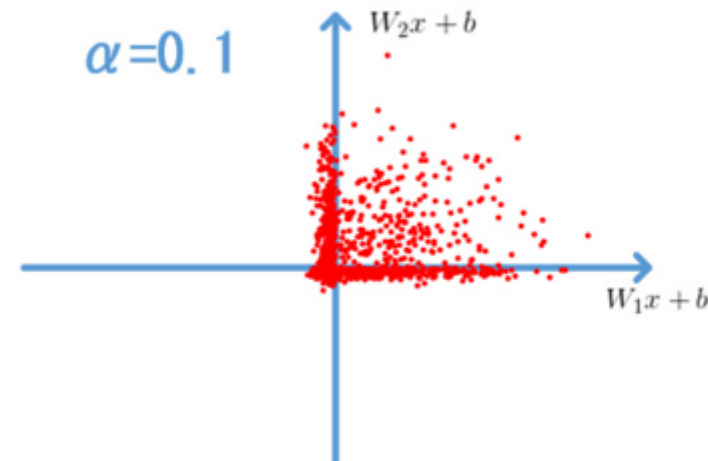
Leaky-ReLU、P-ReLU、R-ReLU

Leaky ReLUs：就是用来解决这个“dying ReLU”的问题的。与 ReLU 不同的是：

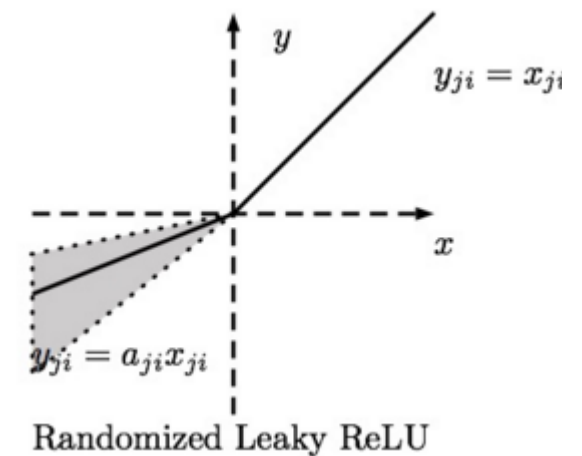
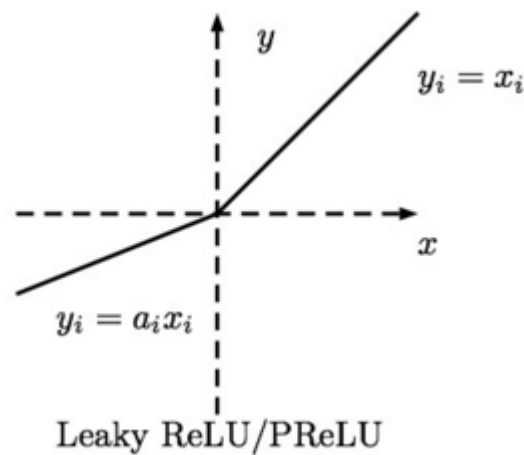
$$f(x)=\alpha x, (x<0)$$

$$f(x)=x, (x \geq 0)$$

这里的 α 是一个很小的常数。这样，即修正了数据分布，又保留了一些负轴的值，使得负轴信息不会全部丢失。



关于Leaky ReLU 的效果，众说纷纭，没有清晰的定论。有些人做了实验发现 Leaky ReLU 表现的很好;有些实验则证明并不是这样。



Parametric ReLU：对于 Leaky ReLU 中的 α ，通常都是通过先验知识人工赋值的。

然而可以观察到，损失函数对 α 的导数我们是可以求得的，可不可以将它作为一个参数进行训练呢？

Kaiming He的论文《Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification》指出，不仅可以训练，而且效果更好。

公式非常简单，反向传播至未激活前的神经元的公式就不写了，很容易就能得到。对 α 的导数如下：

$$\delta y_i \frac{\partial \alpha}{\partial \alpha} = 0, \text{ (if } y_i > 0), \text{ else } = y_i$$

原文说使用了Parametric ReLU后，最终效果比不用提高了1.03%.

$$y_{ji} = \begin{cases} x_{ji} & \text{if } x_{ji} \geq 0 \\ a_{ji} x_{ji} & \text{if } x_{ji} < 0, \end{cases}$$

where

$$a_{ji} \sim U(l, u), l < u \text{ and } l, u \in [0, 1)$$

Randomized ReLU：

Randomized Leaky ReLU 是 leaky ReLU 的random 版本 (α 是random的).

它首次试在 kaggle 的NDSB 比赛中被提出的。

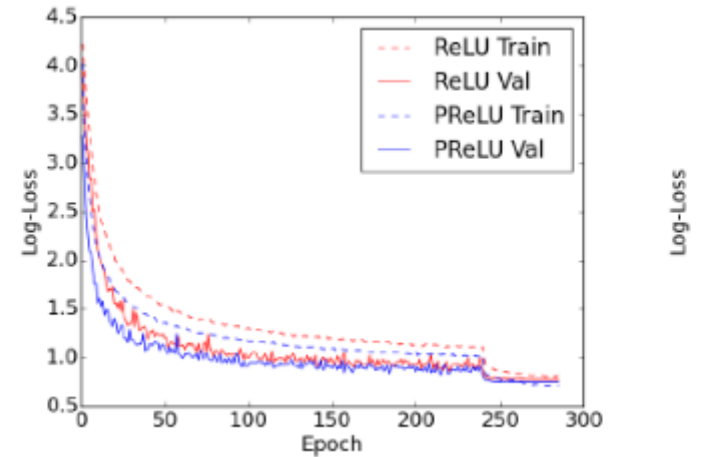
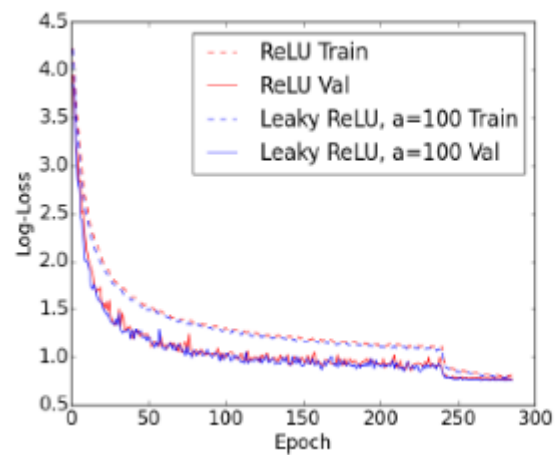
核心思想就是，在训练过程中， α 是从一个高斯分布 $U(l, u)$ 中 随机出来的，然后再测试过程中进行修正(有点像 dropout的用法)。

数学表示如下：

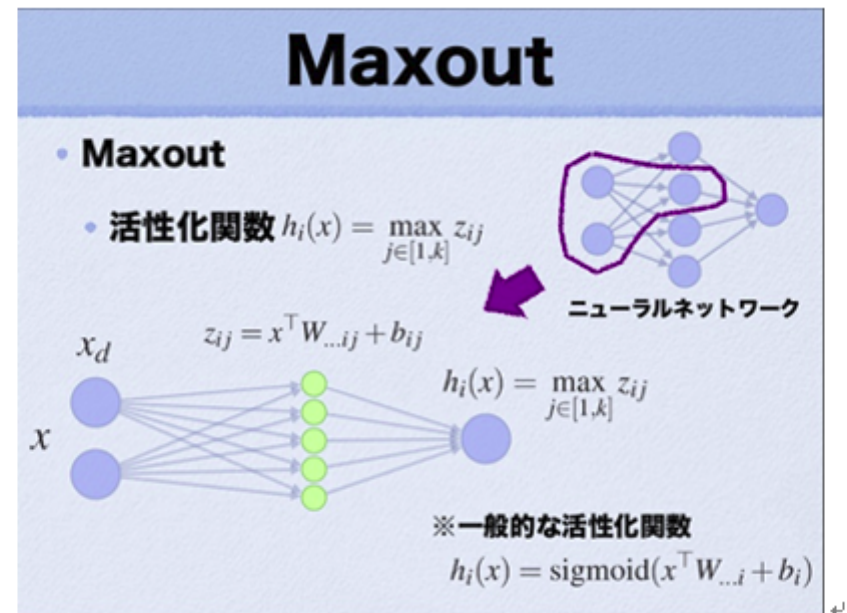
在测试阶段，把训练过程中所有的 a_{ij} 取个平均值。NDSB 冠军的 α 是从 $U(3, 8)$ 中随机出来的。那么，在测试阶段，激活函数就是就是：

$$y_{ij} = x_{ij} + u^2$$

看看 cifar-100 中的实验结果：



Maxout



Maxout出现在ICML2013上，作者Goodfellow将maxout和dropout结合后，号称在MNIST, CIFAR-10, CIFAR-100, SVHN这4个数据上都取得了start-of-art的识别率。

Maxout 公式如下：

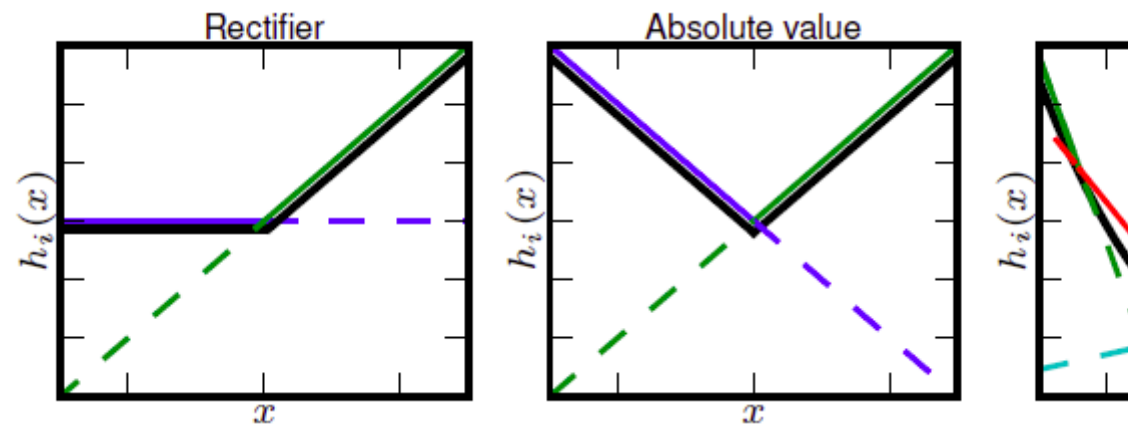
$$f_i(x) = \max_{j \in [1, k]} z_{ij}$$

假设 w 是2维，那么有：

$$f(x) = \max(wT_1x + b_1, wT_2x + b_2)$$

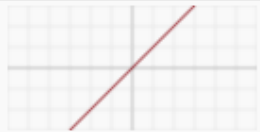

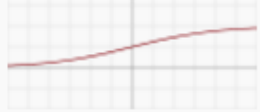
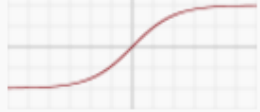


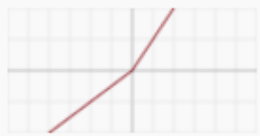


可以注意到，ReLU 和 Leaky ReLU 都是它的一个变形(比如， $w_1, b_1=0$ 的时候，就是 ReLU)。



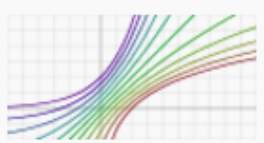
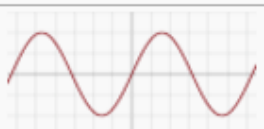
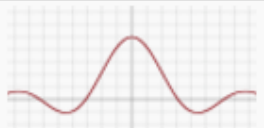
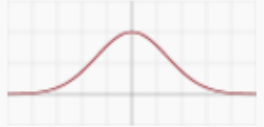
Maxout的拟合能力是非常强的，它可以拟合任意的凸函数。作者从数学的角度上也证明了这个结论，即只需2个maxout节点就可以拟合任意的凸函数了(相减)，前提是“隐层”节点的个数可以任意多。



所以，Maxout 具有 ReLU 的优点(如：计算简单，不会 saturation)，同时又没有 ReLU 的一些缺点(如：容易 go die)。不过呢，还是有一些缺点的嘛：就是把参数double了。

还有其他一些激活函数，请看下表：

Name	Plot	Equation	
Identity		$f(x) = x$	$f'(x)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x)$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x)$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x)$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x)$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x)$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x)$

			
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x)$
SoftExponential		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x)$
Sinusoid		$f(x) = \sin(x)$	$f'(x)$
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x)$
Gaussian		$f(x) = e^{-x^2}$	$f'(x)$

How to choose a activation function?

怎么选择激活函数呢？

我觉得这种问题不可能有定论的吧，只能说是个人建议。

如果你使用 ReLU，那么一定要小心设置 learning rate，而且要注意不要让你的网络出现很多“dead”神经元，如果这个问题不好解决，那么可以试试 Leaky ReLU、PReLU 或者 Maxout.

三、优化算法

深度学习的优化算法跟机器学习也大体一样，但出现一些专门用于神经网络的优化算法，现在一一介绍。

主要是一阶的梯度法，包括SGD, Momentum, Nesterov Momentum, AdaGrad, RMSProp, Adam。其中SGD, Momentum, Nesterov Momentum是手动指定学习速率的,而后面的AdaGrad, RMSProp, Adam,就能够自动调

节学习速率。
二阶的方法目前还不太常用。

BGD

即batch gradient descent. 在训练中,每一步迭代都使用训练集的所有内容. 也就是说,利用现有参数对训练集中的**每一个输入**生成一个估计输出 y_i^{\wedge} ,然后跟实际输出 y_i 比较,统计所有误差,求平均以后得到平均误差,以此来作为更新参数的依据.

具体实现:

需要:学习速率 ϵ , 初始参数 θ

每步迭代过程:

1. 提取训练集中的所有内容 $\{x_1, \dots, x_n\}$,以及相关的输出 y_i
2. 计算梯度和误差并更新参数:

$$\hat{g} \leftarrow + \frac{1}{n} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i)$$
$$\theta \leftarrow \theta - \epsilon \hat{g}$$

优点:

由于每一步都利用了训练集中的所有数据,因此当损失函数达到最小值以后,能够保证此时计算出的梯度为0,换句话说,就是能够收敛.因此,使用BGD时不需要逐渐减小学习速率 ϵ_k

缺点:

由于每一步都要使用所有数据,因此随着数据集的增大,运行速度会越来越慢.

SGD

SGD全名 stochastic gradient descent , 即随机梯度下降。不过这里的SGD其实跟MBGD(minibatch gradient descent)是一个意思,即随机抽取一批样本,以此为根据来更新参数.

具体实现:

需要:学习速率 ϵ , 初始参数 θ

每步迭代过程:

1. 从训练集中的随机抽取一批容量为m的样本 $\{x_1, \dots, x_m\}$,以及相关的输出 y_i
2. 计算梯度和误差并更新参数:

$$\hat{g} \leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i)$$

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

优点:

训练速度快,对于很大的数据集,也能够以较快的速度收敛.

缺点:

由于是抽取,因此不可避免的,得到的梯度肯定有误差.因此学习速率需要逐渐减小.否则模型无法收敛

因为误差,所以每一次迭代的梯度受抽样的影响比较大,也就是说梯度含有比较大的噪声,不能很好的反映真实梯度.

学习速率该如何调整:

那么这样一来, ϵ 如何衰减就成了问题.如果要保证SGD收敛,应该满足如下两个要求:

$$\left. \begin{aligned} \sum_{k=1}^{\infty} \epsilon_k &= \infty \\ \sum_{k=1}^{\infty} \epsilon_k^2 &< \infty \end{aligned} \right|$$

而在实际操作中,一般是进行线性衰减:

$$\epsilon_k = (1 - \alpha) \epsilon_0 + \alpha \epsilon_{\tau}$$

$$\alpha = \frac{k}{\tau}$$

其中 ϵ_0 是初始学习率, ϵ_{τ} 是最后一次迭代的学习率. τ 自然代表迭代次数.一般来说, ϵ_{τ} 设为 ϵ_0 的1%比较合适.而 τ 一般设为让训练集中的每个数据都输入模型上百次比较合适.那么初始学习率 ϵ_0 怎么设置呢?书上说,你先用固定的学习速率迭代100次,找出效果最好的学习速率,然后 ϵ_0 设为比它大一点就可以了.

Momentum

上面的SGD有个问题,就是每次迭代计算的梯度含有比较大的噪音. 而Momentum方法可以比较好的缓解这个问题,尤其是在面对小而连续的梯度但是含有很多噪声的时候,可以很好的加速学习.Momentum借用了物理中的动量概念,即前几次的梯度也会参与运算.为了表示动量,引入了一个新的变量 v (velocity). v 是之前的梯度的累加,但是每回合都有一定的衰减.

具体实现:

需要:学习速率 ϵ , 初始参数 θ , 初始速率 v , 动量衰减参数 α

每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本 $\{x_1, \dots, x_m\}$,以及相关的输出 y_i
2. 计算梯度和误差,并更新速度 v 和参数 θ :

$$\begin{aligned}\hat{g} &\leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\ v &\leftarrow \alpha v - \epsilon \hat{g} \\ \theta &\leftarrow \theta + v\end{aligned}$$

其中参数 α 表示每回合速率 v 的衰减程度.同时也可以推断得到,如果每次迭代得到的梯度都是 g ,那么最后得到的 v 的稳定值为

$$\frac{\epsilon \|g\|}{1 - \alpha}$$

也就是说,Momentum最好情况下能够将学习速率加速 $1/(1-\alpha)$ 倍.一般 α 的取值有0.5,0.9,0.99这几种.当然,也可以让 α 的值随着时间而变化,一开始小点,后来再加大.不过这样一来,又会引进新的参数.

特点:

前后梯度方向一致时,能够加速学习

前后梯度方向不一致时,能够抑制震荡

Nesterov Momentum

这是对之前的Momentum的一种改进,大概思路就是,先对参数进行估计,然后使用估计后的参数来计算误差

具体实现:

需要:学习速率 ϵ , 初始参数 θ , 初始速率 v , 动量衰减参数 α

每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本 $\{x_1, \dots, x_m\}$, 以及相关的输出 y_i
2. 计算梯度和误差, 并更新速度 v 和参数 θ :

$$\begin{aligned}\hat{g} &\leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta + \alpha v), y_i) \\ v &\leftarrow \alpha v - \epsilon \hat{g} \\ \theta &\leftarrow \theta + v\end{aligned}$$

注意在估算 \hat{g} 的时候, 参数变成了 $\theta + \alpha v$ 而不是之前的 θ

AdaGrad

AdaGrad可以自动变更学习速率,只是需要设定一个全局的学习速率 ϵ ,但是这并非实际学习速率,实际的速率是与以往参数的模之和的开方成反比的.也许说起来有点绕口,不过用公式来表示就直白的多:

$$\epsilon_n = \frac{\epsilon}{\delta + \sqrt{\sum_{i=1}^{n-1} g_i \odot g_i}}$$

其中 δ 是一个很小的常亮,大概在 10^{-7} ,防止出现除以0的情况.

具体实现:

需要:全局学习速率 ϵ , 初始参数 θ , 数值稳定量 δ

中间变量: 梯度累计量 r (初始化为0)

每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本 $\{x_1, \dots, x_m\}$,以及相关的输出 y_i
2. 计算梯度和误差,更新 r ,再根据 r 和梯度计算参数更新量

$$\left. \begin{aligned} \hat{g} &\leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\ r &\leftarrow r + \hat{g} \odot \hat{g} \\ \Delta\theta &= -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g} \\ \theta &\leftarrow \theta + \Delta\theta \end{aligned} \right|$$

优点:

能够实现学习率的自动更改。如果这次梯度大,那么学习速率衰减的就快一些;如果这次梯度小,那么学习速率衰减的就慢一些。

缺点:

任然要设置一个变量 ϵ

经验表明,在普通算法中也许效果不错,但在深度学习中,深度过深时会造成训练提前结束。

RMSProp

RMSProp通过引入一个衰减系数,让 r 每回合都衰减一定比例,类似于Momentum中的做法。

具体实现:

需要:全局学习速率 ϵ , 初始参数 θ , 数值稳定量 δ , 衰减速率 ρ

中间变量: 梯度累计量 r (初始化为0)

每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本 $\{x_1, \dots, x_m\}$,以及相关的输出 y_i
2. 计算梯度和误差,更新 r ,再根据 r 和梯度计算参数更新量

$$\begin{aligned}
 \hat{g} &\leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\
 r &\leftarrow \rho r + (1 - \rho) \hat{g} \odot \hat{g} \\
 \Delta \theta &= - \frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g} \\
 \theta &\leftarrow \theta + \Delta \theta
 \end{aligned}$$

优点：

相比于AdaGrad,这种方法很好的解决了深度学习中过早结束的问题
 适合处理非平稳目标，对于RNN效果很好

缺点：

又引入了新的超参，衰减系数 ρ
 依然依赖于全局学习速率

RMSProp with Nesterov Momentum

当然，也有将RMSProp和Nesterov Momentum结合起来的

具体实现：

需要:全局学习速率 ϵ , 初始参数 θ , 初始速率 v , 动量衰减系数 α , 梯度累计量衰减速率 ρ

中间变量: 梯度累计量 r (初始化为0)

每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本 $\{x_1, \dots, x_m\}$, 以及相关的输出 y_i
2. 计算梯度和误差, 更新 r , 再根据 r 和梯度计算参数更新量

$$\begin{aligned}
 \tilde{\theta} &\leftarrow \theta + \alpha v \\
 \hat{g} &\leftarrow + \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x_i; \tilde{\theta}), y_i) \\
 r &\leftarrow \rho r + (1 - \rho) \hat{g} \odot \hat{g} \\
 v &\leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot \hat{g} \\
 \theta &\leftarrow \theta + v
 \end{aligned}$$

Adam

Adam(Adaptive Moment Estimation)本质上是带有动量项的RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

具体实现:

需要:步进值 ϵ , 初始参数 θ , 数值稳定量 δ , 一阶动量衰减系数 ρ_1 , 二阶动量衰减系数 ρ_2

其中几个取值一般为: $\delta=10^{-8}$, $\rho_1=0.9$, $\rho_2=0.999$

中间变量: 一阶动量 s , 二阶动量 r , 都初始化为0

每步迭代过程:

1. 从训练集中的随机抽取一批容量为 m 的样本 $\{x_1, \dots, x_m\}$, 以及相关的输出 y_i
2. 计算梯度和误差, 更新 r 和 s , 再根据 r 和 s 以及梯度计算参数更新量

$$\begin{aligned}
 g &\leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\
 s &\leftarrow \rho_1 s + (1 - \rho_1) g \\
 r &\leftarrow \rho_2 r + (1 - \rho_2) g \odot g \\
 \hat{s} &\leftarrow \frac{s}{1 - \rho_1} \\
 \hat{r} &\leftarrow \frac{r}{1 - \rho_2} \\
 \Delta \theta &= -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta} \\
 \theta &\leftarrow \theta + \Delta \theta
 \end{aligned}$$

四、模型

就模型分类来说，深度学习和机器学习并没有很大区别，只是深度学习更在意神经网络的结构与特定应用领域的契合度，衍生出很多模型框架，这里不再赘述，本博客其他文章都有讨论。

分类: [DL](#)

好文要顶

关注我

收藏该文



AI-ML-DL

关注 - 0

粉丝 - 15

[+加关注](#)

0

0

« 上一篇: [机器学习的补充和总结](#)

» 下一篇: [CV: HOG](#)

posted @ 2017-02-09 16:59 AI-ML-DL 阅读(975) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】报表开发有捷径：快速设计轻松集成，数据可视化和交互



最新IT新闻:

- 全球数据库排名：MySQL三连跌，PostgreSQL最稳
 - 谷歌也将推出7吋屏智能音箱：产品代号“曼哈顿”
 - 贾跃亭向美法院申请的临时禁令威力如何
 - 阿里抄袭事件：被抄袭者回应称阿里行为是诈骗
 - 中国快递分拣有多牛：画面太逆天我不敢看
- » 更多新闻...



最新知识库文章:

- 实用VPC虚拟私有云设计原则
 - 如何阅读计算机科学类的书
 - Google 及其云智慧
 - 做到这一点，你也可以成为优秀的程序员
 - 写给立志做码农的大学生
- » 更多知识库文章...