神经网络

激活函数

定义给定输入或输入集的那个节点的输出。

类型

- ReLU: $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Binary: $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
- Tanh: $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
- Softplus: $f(x) = \ln(1+e^{x})$
- Softmax: $f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \text{for } i = 1, \ldots, J$
- Maxout: $f(\vec{x}) = \max_i x_i$

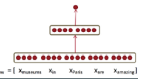
Leaky ReLu/PReLU/RReLU/ELU/SELU 等等

反向传播

是一种用于人工神经网络的方法，用于计算每个神经元在一批数据后的误差贡献，计算损失函数的梯度。在梯度下降优化算法中常用。它也被称为错误的反向传播，因为错误是在输出中计算出来的，并且通过网络层重新分布。

$s = U^T f(Wx + b) \qquad x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 1}$

$$\begin{aligned} s &= U^T a \\ a &= f(z) \\ z &= Wx + b \end{aligned}$$



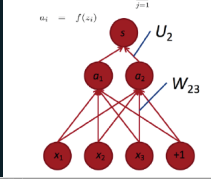
用4维向量表示的神经网络

$$J = \max(0, 1 - s + s_c) \qquad \begin{aligned} s &= U^T f(Wx + b) \\ s_c &= U^T f(Wx_c + b) \end{aligned}$$

$$\begin{aligned} \frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\ &= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\ &= \sum_{i=1}^{2} U_i \frac{\partial f(W_i x + b)}{\partial x_j} \\ &= \sum_{i=1}^{2} \underbrace{U_i f'(W_i x + b)}_{\delta_i} \frac{\partial W_i x}{\partial x_j} \\ &= \sum_{i=1}^{2} \delta_i W_{ij} \\ &= \delta^T W_{\cdot j} \end{aligned}$$

$$\frac{\partial s}{\partial x} = W^T \delta$$

$$\begin{aligned} z_i &= W_i x + b_i = \sum_{j=1}^{n} W_{ij} x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\}\,(-f(Wx + b) + f(Wx_c + b))$$

$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\}\,(-a + a_c)$$

在这种方法中，我们用了在较低层次上计算较高层次的偏导数，以提高效率。

权重初始化

但是，这是一个错误，因为如果网络中的每个神经元都计算相同的输出，那么它们也会在反向传播过程中计算相同的梯度，并进行完全相同的参数更新。换句话说，如果它们的权值被初始化为相同，那么神经元之间就没有不对称的来源。

在理想情况下，有了适当的数据归一化，我们可以合理地假设，大约一半的权重是正的，一半是负的。一个合理的想法可能是把所有的初始权值设为零，你期望这是期望中的"最佳猜测"。

全0初始化

因此，你仍然希望权值非常接近于零，但不等于零。通过这种方式，你可以随机地将这些神经元发送到非常接近于零的小数字，并将其视为对称性破断。这个想法是，神经元在开始时都是随机的和独特的，所以它们会计算出不同的更新，并将自己整合成整个网络的不同部分。

使用小的随机种子初始化

权值的实现可以简单地从正态分布和零均值和单位标准差处提取值。也可以使用从均匀分布中提取的少量数据，但这似乎对实际的最终性能影响不大。

这确保了网络中的所有神经元最初具有近似相同的输出分布，并通过经验来提高收敛速度。可以从页面中找到详细的派生，18到23张幻灯片。请注意。在推导过程中，它不考虑Relu神经元的影响。

上述建议的一个问题是，随机初始化的神经元的输出的分布有随输入数量而增加的方差。结果是，你可以将每个神经元的输出的方差标准化，通过将其权重向量按其扇形的平方根缩放。

服从某种分布的初始化

学习率

但是，如果你真的尝试这样做，每次迭代权重将会改变太多，这会导致"过度正确"，而损失实际上会增加/发散。所以在实践中，人们通常用一个叫做"学习速率"的小数值乘以每个导数，然后再减去相应的权重。

神经网络通常是由梯度下降法训练的。这意味着在每一次迭代中，我们使用反向传播来计算损失函数对每一个权重的导数，并从这个权重中减去它。

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

最简单的方法:保持它的固定，并对所有参数使用相同的方法。

技巧

- 当验证集停止提升的时降低0.5
- 最好的结果是使用自适应的学习速率

$$\alpha = \frac{\alpha_0 \tau}{\max(t, \tau)}$$

t是迭代次数,其他是参数

由于理论收敛的保证，由于超参数的存在，减少了O(1/t)

截至目前为止最好的是使用AdaGrad

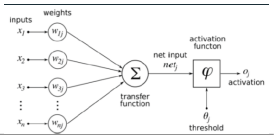
批归一化

使用小批量的样本，而不是一次一个样本，在很多方面都很有帮助。首先，小批量的损失的梯度是对训练集的梯度的估计，随着批量大小的增加，其质量也会提高。
其次，由于现代计算平台所提供的并行性，对一个批处理的计算要比单个示例的计算效率要高得多。

在SGD中，训练将逐步进行，每一步我们都考虑一个小批量的1*m的样本。小批量是用来近似于参数的损失函数的梯度。

神经元

一个单元通常是指在一个网络层中，通过有一个非线性激活函数（例如sigmoid函数）来转换输入，通常一个单元会连接一些输入和一些输出



输入层

由多个实值输入组成。每个输入必须是线性无关的。

隐藏层

指在输入层和输出层之间的神经网络层。属于深度学习的最高级别的模块，一个隐藏层就相当于一个容器，它通常接受加权之后的输入，将其通过激活函数转换为一组大多数的非线性值，作为下一层的输入