# 二叉树神经网络

作者：张俊东

联系方式：13016606412（手机微信），510481609@qq.com，zhangjd26@mail2.sysu.edu.cn

我目前在构思一种新的神经网络的结构，这种结构允许以一种更高效的方式来组织模型的参数，可以带来规模上和效率上的提升。经过了几个月的琢磨，我现在总算是有一些眉目，先记录一下方便与人交流。

## 参考资料

https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html

实践中，我发现KD树其实是一种高效的数据结构，它可以将空间中的多个点进行空间上的划分，并以树的形式组织起来，当从外部输入一个点并希望找到离这个点最近的元素时，它可以使用二分搜索进行高效检索。我认为该树结构非常具有启发性。

https://scikit-learn.org/stable/modules/tree.html

实际上，我是树模型的忠实拥趸，首先，树结构的可解释性非常好；其次，树结构的前向过程足够的高效迅速，对边缘计算设备非常的友好；最后，树模型可以实际上可以容纳非常的多的参数，在规模上有优势。目前，在机器学习领域，树模型主要的代表是决策树模型，但我认为这远远不够。首先决策树并不是被设计在GPU上进行训练的，这意味着它并不能很好地利用当前主流的大算力。另一方面，主流的人工神经网络虽然对于GPU设备非常友好，但我认为线性运算加上激活函数的组织方式其实是低效的，还有巨大的可优化的空间。因此，探寻一种结合树模型和神经网络的方式，我认为是有益的。

https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html

https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html

另一个角度来自于现在主流的注意力机制。注意力机制是Transformer系列模型的核心概念。对于一颗二叉树来说，实际上，它做的事情其实只是在不断地根据输入计算当前的注意力是在节点的左侧还是在节点的右侧。随着二叉树深度的增加，其实它可以建模相当大的空间的注意力。

https://zh.wikipedia.org/wiki/多层感知器

实际上，我认为二叉树模型对标的是其实是MLP，实际上它建模的是一个任意的映射关系。

## 任务

目前，我现在测试的任务其实是非常简单。我希望使用二叉树神经网络来建模一个离散映射过程，更具体来说，是二进制元素的映射关系。

对于模型的输入，我称它为一个Query，记为q，它是一个二进制变量，也就是一个布尔变量的向量。

对于模型的输出，我称它为一个Value，记为v，它也是一个二进制变量，一个布尔变量的向量。

树模型的任务就是建模这样一个确定的映射关系，q->v，对于任意的一个q，它可以学习到需要输出哪一个具体的v。

# 方法

## 对数域的运算

假设现在有一个命题A，对于该命题，我们去刻画它为真和为假的信念分别是多少。

$$P(A) = \frac{e^x}{e^x + e^y}$$
$$P(\neg A) = \frac{e^y}{e^x + e^y}$$

实际上，0和1之间是一个非常窄的区间，在这个区间内求取梯度进行机器学习往往会导致精度上出现问题。一个好的做法是将所有运算转到对数域进行，这并不复杂。

$$logP(A) = x - log(e^x + e^y)$$
$$logP(\neg A) = y - log(e^x + e^y)$$

举例来说，$(x_1 = -100, y_1 = 0)$和$(x_2 = -10000, y_2 = 0)$相比，它们在概率上其实是非常接近，但他们的x却相差了一个两个数量级。因此，通过对数操作，可以直接将x暴露出来，对于精度是有保障的。因此，此后篇幅内所有的运算都是在对数域中进行。

### 逻辑与

假设现在有一个命题A和命题B，假定这两个命题是独立的，那么命题$A \wedge B$应该如何刻画呢?

$$logP(A) = x_A - log(e^{x_A} + e^{y_A})$$
$$logP(B) = x_B - log(e^{x_B} + e^{y_B})$$

$$logP(A \wedge B) = logP(A) + logP(B)$$

### 逻辑或

$$logP(A \vee B) = logsum(logP(A \wedge B), logP(\neg A \wedge B), logP(A \wedge \neg B))$$
$$= logsum(logP(A) + logP(B), logP(\neg A) + logP(B), logP(A) + logP(\neg B))$$

logsum也是对数域的一种运算，它的运算方式如下

$$logsum(x_1, x_2, \cdots, x_n) = log(\sum_i^n e^{x_i})$$
$$= x^* + log(\sum_i^n e^{x_i - x^*})$$

其中，$x^* = \max_i^n x_i$。通过这种拆解技巧，可以有效保持参数在对数域上的精度。另外，logsum运算支持规约操作，在GPU上有非常高效的实现。

$$logsum(x_1,x_2,x_3,x_4)=logsum(logsum(x_1,x_2),logsum(x_3,x_4))$$

### 非二值命题

一个命题可能会有多个取值，如0、1、2、3...等，而不仅仅是True和False。对于非二值命题应该如何刻画呢？一种经典的做法是使用softmax，不过这里介绍一种更有技巧的方法。假设一个命题可能会有8种不同的取值，那么我们其实可以把这个命题拆解成3个二值命题，因为$2^3=8$。这要怎么做呢？

$$\begin{gathered}logP(A=0)=logP(\neg A_0)+logP(\neg A_1)+logP(\neg A_2)\\logP(A=1)=logP(\neg A_0)+logP(\neg A_1)+logP(A_2)\\logP(A=2)=logP(\neg A_0)+logP(A_1)+logP(\neg A_2)\\logP(A=3)=logP(\neg A_0)+logP(A_1)+logP(A_2)\\logP(A=4)=logP(A_0)+logP(\neg A_1)+logP(\neg A_2)\\logP(A=5)=logP(A_0)+logP(\neg A_1)+logP(A_2)\\logP(A=6)=logP(A_0)+logP(A_1)+logP(\neg A_2)\\logP(A=7)=logP(A_0)+logP(A_1)+logP(A_2)\end{gathered}$$

### 可学习命题

来考察一个最简单的一个机器学习问题，模型的输入是一个二值命题A，模型的输出也是一个二值命题B。那么就只会有两种情况$B=A$ 或者 $B=\neg A$。到底是哪一种情况呢？可以设置一个可学习的命题C来进行刻画，

$$B=\begin{cases}A & if \quad \neg C\\ \neg A & if \quad C\end{cases}$$

那么B的 logP 就表示为

$$\begin{gathered}logP(B)=logsum(logP(A)+logP(\neg C),logP(\neg A)+logP(C))\\logP(\neg B)=logsum(logP(A)+logP(C),logP(\neg A)+logP(\neg C))\end{gathered}$$

此时命题C就可以通过梯度下降来进行学习。通过设置各种各样的可学习命题还可以实现各种各样有趣的功能，比如我们可以对于特征向量学习一个重要性的排序；也可以在排序后特征空间中学习一个边界，当特征向量在边界的左边时，走左子树，而在右边时走右子树。

### 小结

这里介绍的对数域的逻辑运算，是一种逻辑自洽的智算理论，运用它可以构建许多推理过程。

## 二叉树

### 非叶节点

我的二叉树参考了KD树的运行方式，每进入一个节点，会对q中的特征进行重排，取非，边界比较三个步骤，然后会计算出左子树和右子树的对数概率。下面的子树会接收这些对数概率，并根据自己计算的左子树和右子树的对数概率，将接收到的对数概率分裂成两部分再传给自己的下一层。

假设有非叶节点N, $N(q)=0$表示节点N把q分配到左子树， $N(q)=1$表示节点N把q分配到右子树。

假设有三个非叶节点$N_1, N_2, N_3$，其中$N_2, N_3$为$N_1$的左右子节点，选择$N_1$的对数概率记为$logP(N_1)$。那么有

$$logP(N_2) = logP(N_1) + logP(N_1(q) = 0)$$
$$logP(N_3) = logP(N_1) + logP(N_1(q) = 1)$$

沿着二叉树从根节点一路计算下去，就可以得到q相对于每一个叶节点的注意力的对数概率。这里的二叉树，只考虑完全二叉树的情况。

## 叶节点

假设树有叶节点$N_1, N_2, N_3, \cdots N_m$，我们给每一个叶节点分配一个可学习的$v_i$，$v_i$可以是二值变量也可能是非二值变量。把模型的输出记为$Tree(q)$，那么

$$logP(Tree(q) = j) = logsum(logP(N_1) + logP(v_1 = j), logP(N_2) + logP(v_2 = j), \cdots, logP(N_m) + logP(v_m = j))$$

在叶节点这里其实它是根据上层递推过来的注意力来分配或者说检索相对应的v。