记忆力机制，时序因果推断，非线性乘算子

作者：张俊东

我们来讨论一些困扰我许久的一些问题，如无意外，这很可能是我研究的最后的东西了，因为我已经没有时间，没有机会，没有资源，我的人生没有希望了。

问题描述，部分可观测隐马尔科夫模型(HMM)

将我们的目光聚焦于这样一些特殊的问题，假设有一个确定的离散系统，但我们并不确定其内部的运行机理。在每一个时间步，它可以从外界接收一个动作a_t，系统会根据当前自身的状态s_t，根据a_t和自身运行机理来跳转到下一状态s_{t+1}。也就是说，对于固定的s_t和a_t而言，s_{t+1}是确定的。困难的是，系统的状态是无法被直接观测的，对于每一个状态s_t而言，我们只能得到一个观测o_t，对于不同的状态s_t而言，它们的观测o_t可能是相同的，因此我们无法直接利用o_t来推断s_t。我们假定初始状态s_0是固定不变的。

对于任何一个确定的时序动作序列，系统可以唯一地确定到一个状态s_t

$$s_0, a_0, a_1, a_2, \cdots, a_{t-1} \rightarrow s_t$$

而对于一个状态s_t而言，系统对于展示出的观测也是固定的

$$s_t \rightarrow o_t$$

实际上，我们希望做是，对于任意序列$s_0, a_0, a_1, a_2, \cdots, a_{t-1}$，我们是否可以利用机器学习模型准确预测出$o_t$。为了做到这一点，模型必须能对系统的状态空间进行高效表征。

请注意，我们认为被建模的系统是确定的，系统对于任意相同的输入序列都会给出相同的反应，这一点与那些基于贝叶斯的概率模型有着显著不同，我们不会考虑模型的后验概率。

对于不同的动作序列而言，系统也有可能转移到相同的状态。也就是说，对于状态空间的某个点而言，系统从初始状态到达该点的路径可能是不唯一的，允许有多条路径到达相同的状态点，实际上这一现象非常普遍。那么如何去判断两条路径到达的状态点是否是同一个状态点呢？其实也非常简单，我们只需要考察对于t时刻后任意的输入序列$a_t, a_{t+1}, a_{t+2}, \cdots$，系统返回的$o_t$是否都是一致的。如果我们总是发现这两条路径达到的点对于此后的任何输入序列，系统都有相同的反应，那我们就可以认为这两条路径到达了一个相同的状态点，这也是模型泛化的基本原理。反之，如果我们一旦发现这这两条路径对于此后的输入序列有着细微的差别，那么它们就一定是状态空间中两个不同的点，在状态空间编码中，我们就必须区分他们。

对于那些状态数有限的问题来说，上面介绍的方法是有效的。而实际上，状态空间中的元素数量往往是非常庞大的，甚至是无限的，以至于我们需要构建非常高效的编码，具有丰富语

义的编码，才能达到高效计算的目的。

我们这里介绍的隐马尔科夫模型(HMM)只能用于建模那些依赖于时序因果关系的序列问题，也就是说当前发生的所有事情仅仅依赖于之前接收到的所有输入，与以后发生的事情无关。

三个基本记忆任务

记忆力是机器学习模型做出精确逻辑推理的最重要因素，是处理长上下文任务基本原理，并且也有利于增强模型的可解释性。尽管它是如此地少被提及。也许在许多年后，人们会回忆起曾经有一场名为“Transformer”的，消耗大量金钱和能源的闹剧。实际上，它连数手指头都学不会。

Remember

我们要介绍的第一个任务叫“Remember”，在$t=0$时刻，外界会随机向系统展现一张卡片i，token“[card i]”;在$t=1,\cdots,n-1$时刻，外界会向系统输入token “[wait]”，意味着系统只需要等待不需要作任何事情；而在$t=n$时刻，外界会对系统发出提问，输入token“[ask]”，系统需要回复最开始看到的那张卡片是什么，要做到这一点，系统需要利用自身能力对先前接收的输入进行记忆；最后，外界再输入结束指令，token“[end]”。举例来说，它们动作序列和观测序列分别为:

t	0	1	2	...	n	n + 1	n + 2	...
obs	None	[wait]	[wait]	[wait]	[wait]	[card i]	[end]	[end]
act	[card i]	[wait]	[wait]	[wait]	[ask]	[end]	[end]	[end]

Count

实际上，教会机器人学会数自己的手指头是一件非常无聊的事情，世界上可能只有我这样一个愚昧又无聊的白痴在孜孜不倦地在做这样一件无聊且毫无意义的事情，否则的话我也不至于快30岁了也一事无成。可反过来说，我们每一个人类哪个不是从数自己的手指头开始学起的呢?

在$t=0,\cdots,n-1$时刻，外界会随机输入一些0和1；而在$t=n$时刻，外界会对系统发出提问，输入token“[ask]”，系统需要回复之前总共输入了多少个1,加起来三多少；最后，外界再输入结束指令，token“[end]”。

t	0	1	2	...	n	n + 1	n + 2	...
obs	None	[wait]	[wait]	[wait]	[wait]	$\sum_{i=0}^{n-1} a_i$	[end]	[end]
act	0	0	1	0 or 1	[ask]	[end]	[end]	[end]

Repeat

该任务考察的是短期记忆能力，外界会不断向系统随机展示不同的卡片，，token“[card i_t]”；在$t = n$时刻，外界会对系统发出提问，输入token“[ask]”，系统需要返回$t = n - m$时刻出现了什么卡片。

t	0	1	2	...	n	n + 1	n + 2	...
obs	None	[wait]	[wait]	[wait]	[wait]	[card i_{n-m}]	[end]	[end]
act	[card i_0]	[card i_1]	[card i_2]	[card i_t]	[ask]	[end]	[end]	[end]

总结

这三个任务参考自 https://github.com/proroklab/popgym 。 不过我对其做了一些简化处理，使得我们可以更专注于核心的问题。

首先我在框架中删除了有关奖励的部分，让记忆问题成为一个纯粹的序列拟合、序列预测的问题。因此呢，我们也会把有关强化学习的内容先放到一边，尽管，它们之间存在千丝万缕的关系。在训练时，我们会直接告诉模型正确的答案，让它学习去预测未知的答案，而免去了探索的过程。

其次，我将所有的不确定性归结为外界对于系统的影响，也就是动作序列，这些动作序列可能是来自于某个外部的随机事件或由决策器（Actor）产生，但不管这些输入来源于何处，我们都统一把它放在act动作序列。而系统应对于这些不确定的因素而给出的确定性的响应，则用obs观测序列来表达。实际上，我们在这里讨论的框架与其他强化学习的理论框架有非常明显的区别。我们用动作序列和观测序列来分别描述不确定性和确定性。

这些纯粹的符号推理任务与自然语言的一些任务相比也有显著的差别，往往符号推理任务不会找到非常固定的模式来对结果进行预测，这与遵循一定语法结构的自然语言有着明显的区别。符号推理的任务注重的是因果推断的能力。

对于Remember而言，它的状态数量与卡片数量c相关。对于Count来说，状态数量与最大的求和数相关。对于Repeat来说，它的状态数是c^m。总体而言，这三个任务的状态数是依次递增的，在不同的数量级。

状态空间方程

对于一个离散线性系统而言，它的状态空间方程可以写作:

$$x_t = Ax_{t-1} + Bu_{t-1}$$
$$y_t = Cx_t$$

x其实是对于状态空间的编码，它与状态空间上的每一个点一一对应，y是对于状态x的观测，而u则是外部的控制输入。实际上，我们更加关心非线性的情况，

$$x_t = f_1(x_{t-1}, u_{t-1})$$
$$y_t = f_2(x_t)$$

布尔代数与矩阵运算

或许我们先来讨论一些布尔代数相关的事情会比较合适。我们常常将布尔代数中的“与”运算类比成实数域中的乘法运算，而将“或”运算类比成加法运算。实际上这并非不合理，因为这些算子确实有着非常相似的性质，包括交换律、分配律、结合律。除了一点，在布尔代数中有“1 + 1 = 1”，但这似乎并不影响我们去运用矩阵论的方法去处理一些布尔代数的问题。

考虑这样一个布尔代数的方程组。

$$\begin{aligned} y_1 &= (a_{11} \wedge x_1) \vee (a_{12} \wedge x_2) \vee \cdots \vee (a_{1n} \wedge x_n) \\ y_2 &= (a_{21} \wedge x_1) \vee (a_{22} \wedge x_2) \vee \cdots \vee (a_{2n} \wedge x_n) \\ &\vdots \\ y_n &= (a_{n1} \wedge x_1) \vee (a_{n2} \wedge x_2) \vee \cdots \vee (a_{nn} \wedge x_n) \end{aligned}$$

实际上，我们可以把它表示成为一个矩阵的形式。

$$y = Ax$$

我们可以在布尔代数中定义一种矩阵的乘法，对于两个布尔矩阵A,B而言，它们的积为：

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & & & \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & & & \\ b_{q1} & b_{q2} & \cdots & b_{qr} \end{bmatrix}$$

$$C = A \times B = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & c_{22} & \cdots & c_{2r} \\ \vdots & & & \\ c_{p1} & c_{p2} & \cdots & c_{pr} \end{bmatrix}$$

$$c_{ij} = (a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \cdots \vee (a_{iq} \wedge b_{qj})$$

这样的布尔矩阵乘法是满足分配律和结合律的。

$$A \times (B + C) = A \times B + A \times C$$
$$A \times (B \times C) = (A \times B) \times C$$

虽然布尔运算是一个“1+1=1”的非线性的运算，但其实它依然保留着实数运算中非常多好的性质。布尔矩阵乘法满足的结合律，可以启发我们设计一些好的归约算法，为算法优化提供空间。

布尔代数的状态空间方程

让我们关注一类特殊的非线性状态空间方程，该方程中的所有变量都是布尔变量，它可以使用布尔矩阵运算来进行表达。

$$x_t = A(u_{t-1})x_{t-1}$$
$$y_t = Cx_t$$
$$A(u_t) = \sum_{i=1}^{n} (u_t)_i A_i$$

u1u2u3 + ~(u1)u2~(u3)

上式中,x,y,u都是向量，$(u)_i$表示向量中第i个数，其中$A_1, A_2, \cdots A_n, C$都是该布尔状态方程的参数，它们不会随着时间而发生改变，也是我们后面想要学习到的参数。控制变量u是一个独热编码的动作变量，用来刻画在t时刻外界输入了什么动作。y也是一个独热编码变量，用来刻画系统输出的离散观测。

序列集：一些等长动作序列的集合，假定对于每一时刻有$a_t \in \{1, 2, \cdots, n\}$，n个选项，那么动作序列 “123” 可以表示 “$a_0 = 1, a_1 = 2, a_2 = 3$”的一个长度为3的序列，多个动作序列的样本可以组成一个序列集。

序列集的指示变量：对于一个序列集P而言，我们可以定义它的指示变量为

$$1(a_0, a_1, \cdots, a_\tau, P_\tau) = \begin{cases} 1 & if \quad a_0, a_1, \cdots, a_\tau \in P_\tau \\ 0 & if \quad a_0, a_1, \cdots, a_\tau \notin P_\tau \end{cases}$$

$$1(a_0, a_1, \cdots, a_\tau, P_\tau) = \sum_{p \in P_\tau} \prod_{i=0}^{\tau} 1(a_i = p_i)$$

控制变量u可以表示为

$$u_t = \begin{bmatrix} 1(a_t = 1) \\ 1(a_t = 2) \\ \vdots \\ 1(a_t = n) \end{bmatrix}$$

如果x_{t-1}可以被写成序列集的指示变量的形式，也就是说

$$x_{t-1} = \begin{bmatrix} \sum_{p \in P_{t-1,1}} \prod_{i=0}^{t-1} 1(a_i = p_i) \\ \sum_{p \in P_{t-1,2}} \prod_{i=0}^{t-1} 1(a_i = p_i) \\ \vdots \\ \sum_{p \in P_{t-1,n}} \prod_{i=0}^{t-1} 1(a_i = p_i) \end{bmatrix}$$

那么，当我们对它乘以一个常数布尔矩阵时，它依然可以表达为某些序列集的指示变量

$$Ax_{t-1} = \begin{bmatrix} \sum_{p \in P'_{t-1,1}} \prod_{i=0}^{t-1} 1(a_i = p_i) \\ \sum_{p \in P'_{t-1,2}} \prod_{i=0}^{t-1} 1(a_i = p_i) \\ \vdots \\ \sum_{p \in P'_{t-1,n}} \prod_{i=0}^{t-1} 1(a_i = p_i) \end{bmatrix}$$

其中，$P'_{t-1,i}$ 是根据A矩阵中的数值在原来的序列集 $P_{t-1,1}, P_{t-1,2}, \cdots, P_{t-1,n}$中求的并集。

我们可以使用数学归纳法，如果x_{t-1}可以被写成长度为t-1序列集的指示变量的形式，那么x_t就可以被写成长度为t序列集的指示变量的形式。

$$x_t = \sum_i 1(a_t = i) A_i x_{t-1}$$

x_0 可以表示长度为0序列集的全集或者空集，证得。

也就是说在这个布尔状态空间方程中，x的每一个分量时刻都在表征当前的序列是否存在于某个集合中，该方程具有很强的表达能力。而这种表达能力与x的长度相关。

可微分的近似布尔算子

尽管布尔运算保留着分配律和结合律等优秀的数学性质，但由于它作用于的是布尔变量，是无法求导数的，因此也难以使用现存的一些基于导数的技术框架。我们希望在实数空间找到一些近似的可以求导数的非线性算子，来拟合布尔运算。

方案1

假设有两个在0到1开区间的实数变量x,y，可以定义它们的与算子和非算子为

$$x \wedge y = \min(x, y)$$
$$\neg x = 1 - x$$
$$x \vee y = \neg(\neg x \wedge \neg y) = 1 - \min(1 - x, 1 - y) = \max(x, y)$$

方案2

实数变量x,y在0到1开区间

$$x \wedge y = xy$$
$$\neg x = 1 - x$$
$$x \vee y = \neg(\neg x \wedge \neg y) = 1 - (1-x)(1-y) = x + y - xy$$

方案3

实数变量x,y是任意实数

$$x \wedge y = \min(x, y)$$
$$\neg x = -x$$
$$x \vee y = \neg(\neg x \wedge \neg y) = -\min(-x, -y) = \max(x, y)$$

方案4

实数变量a,b在0到1开区间

$$a = \frac{1}{1+e^{-x}}$$
$$b = \frac{1}{1+e^{-y}}$$
$$c = \frac{1}{1+e^{-x}+e^{-y}}$$
$$a \wedge b = c$$
$$\neg a = 1 - a$$
$$a \vee b = \neg(\neg a \wedge \neg b) = 1 - \frac{1}{1+e^{x}+e^{y}} = \frac{1}{1+\frac{1}{e^{x}+e^{y}}}$$

方案5

实数变量x,y是任意实数

$$x \wedge y = -\log(e^{-x} + e^{-y})$$
$$\neg x = -x$$
$$x \vee y = \neg(\neg x \wedge \neg y) = \log(e^{x} + e^{y})$$

实际上，方案5的数学性质是最好的，它是方案4在对数域的形式，只要对方案5采取sigmod操作就会回到方案4。对于任意近似布尔算子，如果是在0到1开区间，那么就必须满足

$$1 \wedge 0 = 0$$

显然，对于方案4,一定有 “$c < a, c < b$”, 在方案4中，

$$a \wedge b = \frac{1}{1+e^{-z}} \Rightarrow z = -\log(e^{-x} + e^{-y})$$

$$a \vee b = \frac{1}{1+e^{-z}} \Rightarrow z = \log(e^x + e^y)$$

$$\neg a = \frac{1}{1+e^{-z}} \Rightarrow z = -x$$

方案4要比方案2要好得多，因为方案4的数值要比方案2更加贴近方案1。实际上，方案1完全继承了布尔运算的数学性质，包括分配律、结合律。实际上其他基于连续函数的拟合都会破坏结合律。但是方案1并非一个连续函数，因此它的效果也并不好。虽然使用连续函数来拟合布尔运算会破坏结合律，但数值在两端取极限时，结合律又是成立的，也就是说，当模型收敛后，结合律会重新成立。

方案5对于方案4的改进在于避免了数值精度的消失问题。那么采取方案5来重新定义我们的布尔矩阵运算，它将会是非常的有意思。

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & & & \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & & & \\ b_{q1} & b_{q2} & \cdots & b_{qr} \end{bmatrix}$$

ai1b1j + ai2b2j....

$$C = A \times B = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & c_{22} & \cdots & c_{2r} \\ \vdots & & & \\ c_{p1} & c_{p2} & \cdots & c_{pr} \end{bmatrix}$$

$$c_{ij} = \log\left(\frac{1}{e^{-a_{i1}} + e^{-b_{1j}}} + \frac{1}{e^{-a_{i2}} + e^{-b_{2j}}} + \cdots + \frac{1}{e^{-a_{iq}} + e^{-b_{qj}}}\right)$$

我们将会紧紧围绕这一非线性的乘法算子来构建我们的神经网络，而这在这个网络中，我们会剔除掉所有的激活函数。

矩阵归约乘法，可并行计算的循环神经网络

我参考了一些将RNN应用于大语言模型的工作，发现他们都有类似的归约计算的思路，如Flash Attention, RWKV, Mamba等。我的模型与这些工作的主要区别是采用了我设计的类布尔运算的乘法算子，来实现非线性的效果，没有采用激活函数，也没有采用层层堆叠的结构。

根据之前的布尔状态空间方程，我们可以写出x_t关于每个时刻输入的形式。

$$x_t = A(u_{t-1})A(u_{t-2})A(u_{t-3})\cdots A(u_0)x_0$$

可以看到展开后它是不同时刻的矩阵连乘的结果，因此如果矩阵的乘法满足结合律，就会有非常高效的计算方法。

```
def matrix_cum_mul(self, m: Tensor):
    # m是需要被连乘的矩阵序列
    B, L, D, D = m.shape
    step = 1
    while step < L:
        m1 = m[:, 0:L-step, :, :].clone()
        m2 = m[:, step:L, :, :].clone()
        m3 = m[:, :step, :, :]
        m4 = self.matrix_mul(m1.view(-1, D, D), m2.view(-1, D, D))
        m4 = m4.view(B, -1, D, D)
        m = torch.cat([m3, m4], dim=1)
        step *= 2
    return m
```

我们也讨论过，其实对于近似的布尔矩阵乘法算子，它是不满足结合律的，但它在取极限的情况下，在模型收敛的情况下，是满足结合律的。现在摆在眼前的问题是如何保证模型的收敛性。

在这个循环神经网络中，我们的要学习的参数是之前布尔状态空间方程中所介绍的参数，不过它的数域范围不是0到1，而是任意实数。

参数的随机分布，添加噪声跳出局部最优解

我们可以将学习的参数看作服从某个标准差固定的正态分布的随机变量。假定有学习参数x

$$x \sim N(\mu, \sigma)$$

σ为预设置好的超参数，当μ接近0时，$\frac{1}{1+e^{-x}}$会在一个比较大的范围内波动；而当μ比较接近正无穷或者负无穷时，此时$\frac{1}{1+e^{-x}}$会变得非常稳定。因此，模型的每一个参数，如果模型能收敛的话，它会趋向于正无穷或者负无穷，因为在这些区域，sigmod值才会维持在一个稳定的水平，才会让模型的损失函数持续保持在一个低谷。

损失函数，训练方法

我们采用与gpt类似的训练方法，就是不断地预测下一个词，或者说预测下一个观察。我们回顾一下观察变量的计算方法为：

$$y_t = Cx_t$$

在状态x_t的基础上乘以一个学习的C布尔矩阵。我们可以在y变量上应用softmax和交叉熵函数。

$$L = -\sum_{i} 1(o_t = i)((y_t)_i - \log(\sum_{j} e^{(y_t)_j}))$$

作为一个时序因果推断的模型，我们只需要学习第一个预测错误的单词，这是因为如果无法保证之前预测的结果完全正确，那么其实后面的推理也是全部错误的，此时那些后面单词的学习信号就会成为一种干扰。模型的学习过程也是从短序列一直学习到长序列的过程。

总结

我将一个系统分为不确定性和确定性两部分，其中系统的不确定性完全来自于外界，而系统的确定性体现为对于不确定性的有规律的反应。

系统的反应取决于系统当前所在的状态。

我主要通过三个基本的记忆任务来考察模型的记忆能力。目前，我的模型能够顺利通过这些测试。

我在布尔代数的基础上提出了一个布尔变量的状态空间方程，它有非常不错的表达能力，它是用矩阵的形式来进行表达的。

将布尔运算与矩阵运算结合起来非常有用。

我使用一些连续函数来拟合布尔运算，使得可以利用导数的信息来进行参数搜索。

我使用添加噪声的方法迫使模型跳出局部最优解，这些噪声是不影响收敛的。

放弃科研

我在中山大学这几年，总是花大量的时间来研究各种问题，可换来的却是无尽的羞辱。因此，我想是时候放弃科研了，我遭遇的荒唐至极的事情已经够多了，我已经不可能毕业了，我已经没有钱吃饭了，我已经不可能找到工作了，我没有一技之长，因为我把时间都拿去研究这样一些荒诞至极的东西，教机器人走路，教机器人数手指，没有任何意义。

放弃强国情节，享受躺平人生，每天提醒自己，祖国发展不发展，科技进步还是不进步，根本和我没有任何关系，我不建设有的是人建设。有时候，我也希望自己的生命能够短暂一点，这样我就不用吃这么多苦了。