Neural Memory Streaming Recommender Networks with Adversarial Training

Qinyong Wang Hongzhi Yin∗ Zhiting Hu

KDD ’18, August 19–23, 2018, London, United Kingdom

论文：http://net.pku.edu.cn/daim/hongzhi.yin/papers/KDD18-Qinyong.pdf

提出了一种基于具有**外部记忆**的**神经记忆网络**的**流推荐模型**，以统一的方式捕获和存储长期稳定的利益和短期的动态利益。开发了基于**生成对抗网（GAN）的自适应负采样**框架，优化了我们提出的流推荐模型，有效地克服了经典负抽样方法的局限性，提高了模型参数推理的有效性。

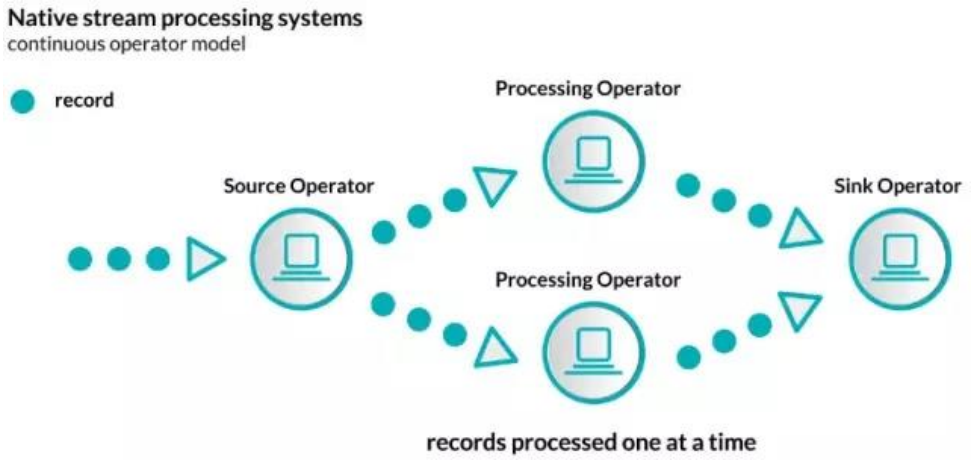
流处理与批处理：

流计算：当一条数据被处理完成后,序列化到缓存中,然后立刻通过网络传输到下一个节点,由下一个节点继续处理。
批处理：当一条数据被处理完成后,序列化到缓存中,并不会立刻通过网络传输到下一个节点,当缓存写满,就持久化到本地硬盘上,当所有数据都被处理完成后,才开始将处理后的数据通过网络传输到下一个节点。

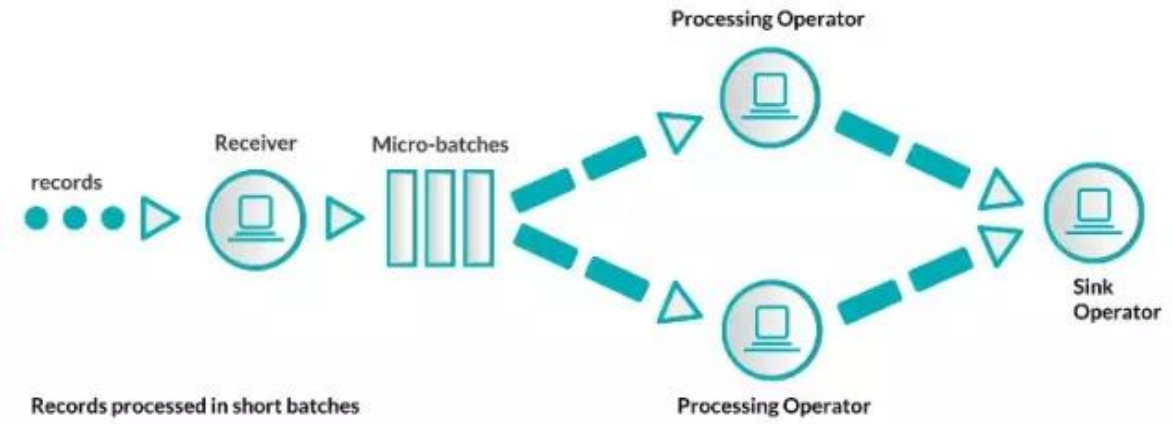
对比指标	批量计算	流式计算
数据集成方式	预先加载数据	实时加载数据实时计算
使用方式	业务逻辑可以修改，数据可重新计算	业务逻辑一旦修改，之前的数据不可重新计算(流数据易逝性)。
数据范围	对数据集中的所有或大部分数据进行查询或处理。	对滚动时间窗口内的数据或仅对最近的数据记录进行查询或处理。
数据大小	大批量数据。	单条记录或包含几条记录的微批量数据。
性能	几分钟至几小时的延迟。	只需大约几秒或几毫秒的延迟。
分析	复杂分析。	简单的响应函数、聚合和滚动指标。

流处理两种方式：

第一种是原生流处理，意味着所有输入的记录一旦到达即会一个接着一个进行处理。



第二种为微批处理。把输入的数据按照某种预先定义的时间间隔（典型的是几秒钟）分成短小的批量数据，流经流处理系统。



批处理具有的缺点：

1）通过重新运行批处理导致模型更新延迟具有**昂贵时间成本**

2）在存在流数据的情况下无法跟踪快速变化的趋势（例如，用户偏好和项目流行度）

3）显式存储所有历史数据的大量内存开销

参考资料：流处理与批处理

- https://juejin.im/post/5baef0bbe51d450e6160343d
- https://zhuanlan.zhihu.com/p/37937010
- https://zhuanlan.zhihu.com/p/37937010

记忆网络（Memory Networks)

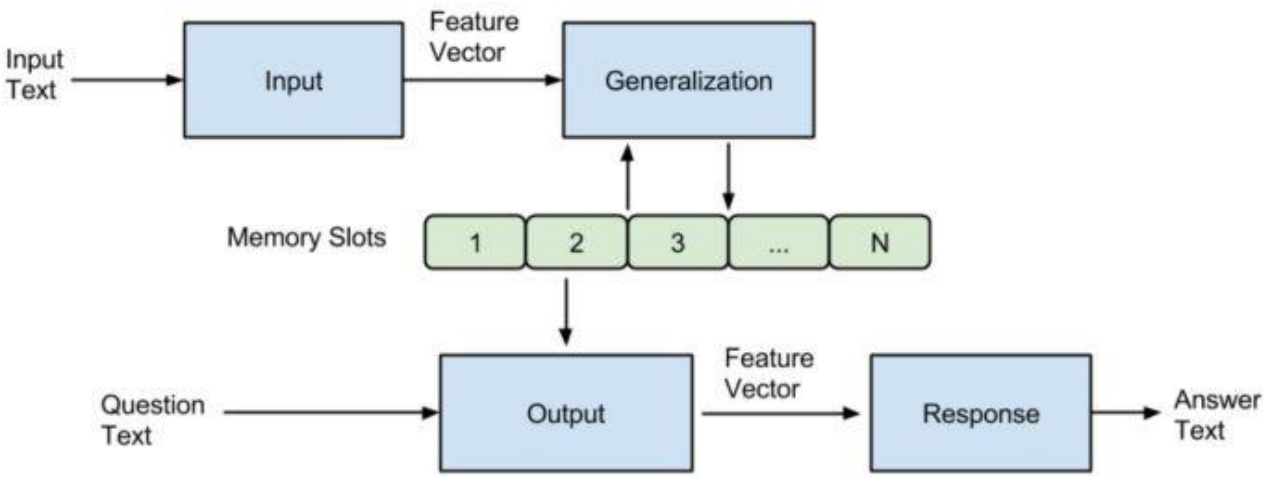


Figure 3: Memory Network architecture.

I、G、O、R四个处理模块

绿色是记忆卡槽（Memory Slots）

每个模块的功能如下：

Memory Slots：内存数组。
I（输入）：将输入转换为内部特征表示。
G（概括）：在给定输入的情况下更新记忆。
O（输出）：在给定新输入（问题）和当前内在状态的情况下，生成一个新的输出。
R（响应）：将输出转换为所需要的响应格式，例如一个文本或者一个行动。

模型的流程：

将 x 转换为内部特表示 I(x)

更新存储器 M，即读写操作(只能添加和修改记忆)

在给定新输入和存储器的情况下计算输出特征：O。

解码输出特征o以给出最终响应：r = R（o）

I 模块是将原始文本转化为向量表示。向量化表示过程
G 模块是将输入的向量存储在 meomry 数组中的下个位置，即写入新的记忆（也可以更新旧记忆）
O 模块是在给定输入（问题）的情况下，寻找跟问题最相关的 top k 个记忆单元。当 k = 1 时，公式如下：

$$o_1 = O_1(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O(x, \mathbf{m}_i)$$

$$o_2 = O_2(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_i)$$

$$r = \text{argmax}_{w \in W} \; s_R([x, \mathbf{m}_{o_1}, \mathbf{m}_{o_2}], w) \tag{4}$$

where W is the set of all words in the dictionary, and s_R is a function that scores the match.

$$s(x, y) = \Phi_x(x)^\top U^\top U \Phi_y(y),$$

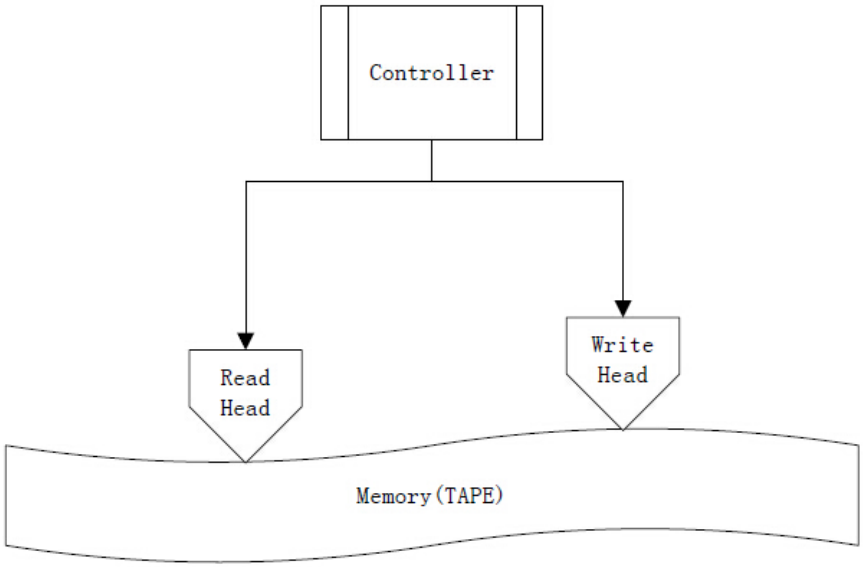
where U is a n × D matrix where D is the number of features and n is the embedding dimension.
The role of Φx and Φy is to map the original text to the D-dimensional feature space.

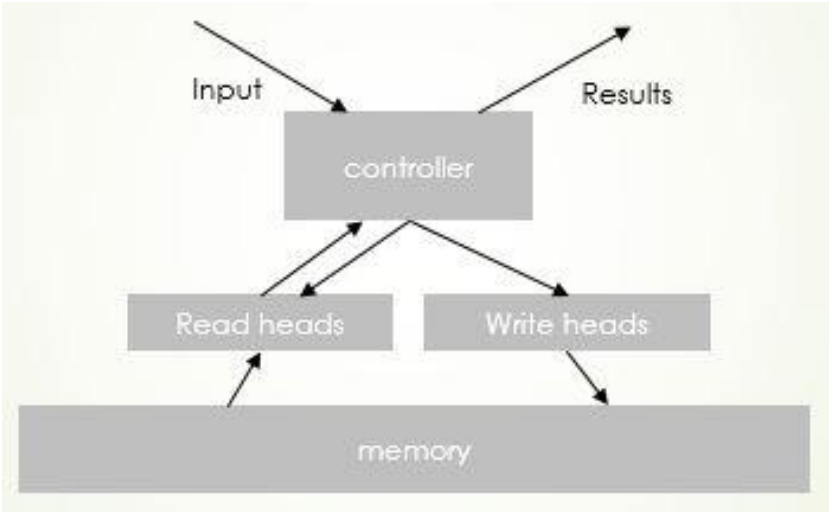
参考资料：

1. 论文:https://arxiv.org/pdf/1410.3916.pdf
2. 论文笔记：记忆网络（Memory Networks)：https://zhuanlan.zhihu.com/p/43864112
3. 记忆网络专题论文：https://zhuanlan.zhihu.com/c_129532277

NTM： Neural Turing Machines

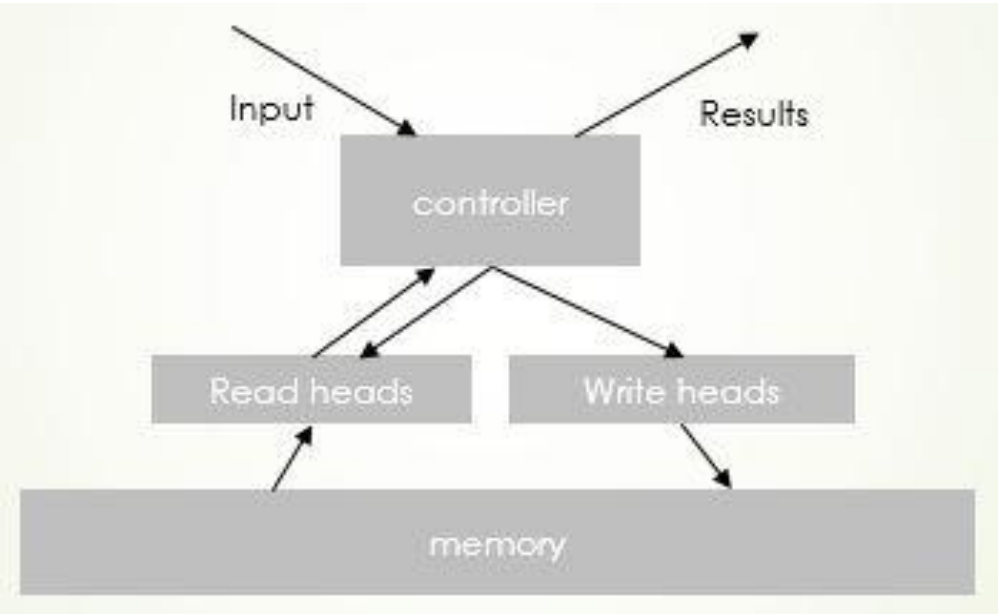
TAPE：磁带，即记忆体（Memory）
HEAD：读写头，read or write TAPE上的内容
TABLE：一套控制规则，也叫控制器（Controller），根据机器当前状态和HEAD当前所读取的内容来决定下一步的操作。在NTM中，TABLE其实模拟了大脑的工作记忆
register：状态寄存器，存储机器当前的状态。





图灵完备性的论证提供了神经网络用于实现图灵机的理论合理性：因此传统图灵机：输入纸带 - > 自动机 - > 输出纸带　变成　输入纸带 - > 神经网络 - > 输出纸带

神经网络图灵机（NTM）架构包含两个基本组件：神经网络控制器和内存池



- memory的矩阵
- 读指针
- 写指针
- 控制器

读操作：

令M_t代表时刻t的N×M内存矩阵。（N代表地址数或行数，M代表每个地址的向量大小）。令W_t在时刻t读写头在N个地址的读写比重，由于所有的权重都进行了归一化，所以W_t向量的内部元素W_t(i)满足：

$$\sum_i w_t(i) = 1, \qquad 0 \le w_t(i) \le 1, \forall i. \tag{1}$$

那么，时刻t读取到的值r_t，可以定义为每个地址的向量Mt(i)加权和：sum

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i), \tag{2}$$

写操作：

受LSTM中的forget gate和输入的启发，我们将写操作拆分成两个部分：先擦除(erase)在添加(add)。

给定t时刻的写头权重w_t，以及一个擦出向量e_t，其中M个元素均在0~1范围内，则t-1时刻的内存向量在t时刻将按下式进行调整：

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\left[\mathbf{1} - w_t(i)\mathbf{e}_t\right], \tag{3}$$

其中1是一个全部是1的行向量。当对应位置权重和擦除元素都是1时，整个内存就会被重置为零。若权重为零或者擦除向量为零，则内存保持不变。当多个写头同时存在时，多个操作可以以任意顺序相互叠加，乘法是可交换的。

添加(add)操作：而加向量a_t，是在擦除动作之后执行下面动作
：

$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\,\mathbf{a}_t. \tag{4}$$

权重是通过两种寻址机制以及一些其他补充机制的共同作用产生的

寻址机制:

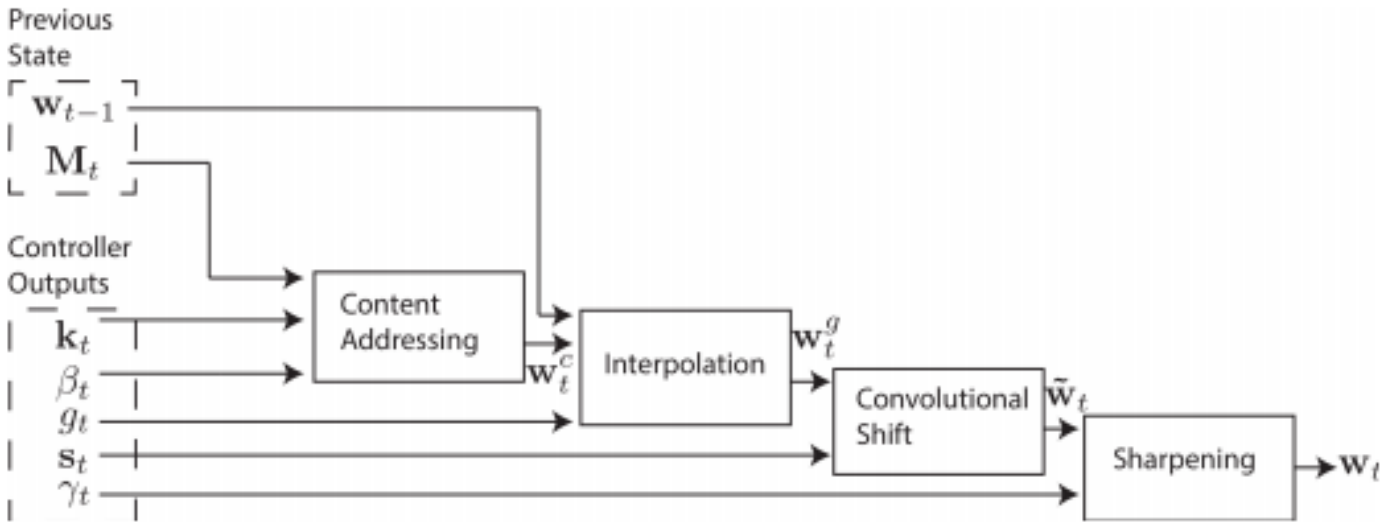


Figure 2: Flow Diagram of the Addressing Mechanism. The *key vector*, $\mathbf{k}_t$, and *key strength*, β_t, are used to perform content-based addressing of the memory matrix, $\mathbf{M}_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the *interpolation gate*, g_t. The *shift weighting*, $\mathbf{s}_t$, determines whether and by how much the weighting is rotated. Finally, depending on γ_t, the weighting is sharpened and used for memory access.

Content-base(基于内容的寻址)：

产生一个待查询的值k_t，将该与M_t中的所有N个地址的值进行比较，最相近的那个M_t(i)即为待查询的值。

首先，需要进行寻址操作的Head(Read or Write)生成一个长度为M的key vector:k_t,然后将k_t与每个M_t(i)进行相似度比较（相似度计算函数为K[u,v]），最后将生成一个归一化的权重向量,计算公式如下：

$$w_t^c(i) \leftarrow \frac{\exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(i)\right]\right)}{\sum_j \exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(j)\right]\right)}.$$

相似性度量是余弦相似度：

$$K\left[\mathbf{u}, \mathbf{v}\right] = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||}.$$

其中，β_t满足$\beta_t > 0$是一个调节因子，用以调节寻址焦点的范围。β_t越大，函数的曲线变得越发陡峭，焦点的范围也就越小。

Location-base(基于位置的寻址)：

直接使用内存地址进行寻址，跟传统的计算机系统类似，controller给出要访问的内存地址，Head直接定位到该地址所对应的内存位置。

基于地址的寻址方式可以同时提升简单顺序访问和随机地址访问的效率。我们通过对移位权值进行旋转操作来实现优化。例如，当前权值聚焦在一个位置，旋转操作1将把焦点移向下一个位置，而一个负的旋转操作将会把焦点移向上一个位置。

在旋转操作之前，将进行一个插入修改的操作(interpolation),每个head将会输出一个修改因子g_t∈[0,1], g值被用作混合前一时刻中读写头产生的w_{t-1}和当前时刻中有内容系统产生的权重列表w_t^c

$$\mathbf{w}_t^g \longleftarrow g_t \mathbf{w}_t^c + (1 - g_t)\mathbf{w}_{t-1}. \tag{7}$$

如果gate是0，那么整个内容权重就被完全忽略，而来自前一个时刻的权重列表就被直接使用。相反，如果gate值是1，那么就完全采用内容寻址的结果。

在插值(interpolation)之后，每个读写头都会给出一个位移权重S_t，用于定义一个在允许的整数值位移上的归一化分布

如果内存地址为0到N-1，使用s_t来旋转w_t^g，可以使用下面的循环卷积来表示：

$$\tilde{w}_t(i) \longleftarrow \sum_{j=0}^{N-1} w_t^g(j)\, s_t(i-j) \tag{8}$$

S_t写成矩阵的形式如下：　$\widetilde{w_t} = s_t w_t^g$

$$\mathbf{S}_t = \begin{bmatrix} s_t(0) & s_t(N-1) & \cdots & s_t(2) & s_t(1) \\ s_t(1) & s_t(0) & s_t(N-1) & \cdots & s_t(2) \\ \vdots & s_t(1) & s_t(0) & \ddots & s_t(2) \\ s_t(N-2) & & \ddots & \ddots & s_t(N-1) \\ s_t(N-1) & s_t(N-2) & \cdots & s_t(1) & s_t(0) \end{bmatrix}$$

由于卷积操作会使权值的分布趋于均匀化，这将导致本来集中于单个位置的焦点出现发散现象。为了解决这个问题，还需要对结果进行锐化操作。具体做法是Head产生一个因子$\gamma_t \geq 1$，并通过如下操作来进行锐化：

$$w_t(i) \longleftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}}$$

我们通过一个简单的例子来说明：

假设N=5，当前焦点为1，三个位置-1,0,1对应的权值为0.1,0.8,0.1，$\mathbf{w}_t^g = \begin{bmatrix} 0.06 \\ 0.1 \\ 0.65 \\ 0.15 \\ 0.04 \end{bmatrix}$ 则

$$\mathbf{S}_t = \begin{bmatrix} s_t(0) & s_t(4) & s_t(3) & s_t(2) & s_t(1) \\ s_t(1) & s_t(0) & s_t(4) & s_t(3) & s_t(2) \\ s_t(2) & s_t(1) & s_t(0) & s_t(4) & s_t(3) \\ s_t(3) & s_t(2) & s_t(1) & s_t(0) & s_t(4) \\ s_t(4) & s_t(3) & s_t(2) & s_t(1) & s_t(0) \end{bmatrix} = \begin{bmatrix} 0.1 & 0 & 0 & 0.1 & 0.8 \\ 0.8 & 0.1 & 0 & 0 & 0.1 \\ 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0.1 & 0.8 & 0.1 \end{bmatrix}$$

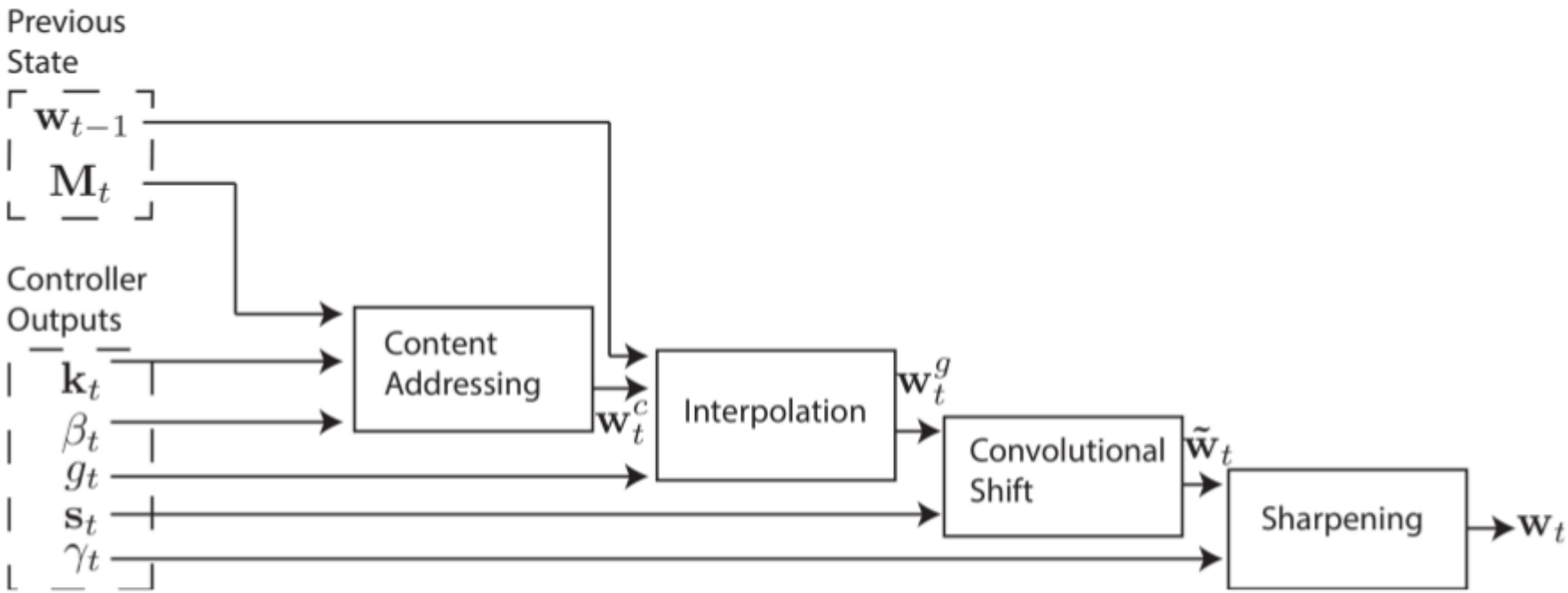
所以有：

$$\widetilde{\mathbf{w}}_t = \mathbf{S}_t \mathbf{w}_t^g = \begin{bmatrix} 0.1 & 0 & 0 & 0.1 & 0.8 \\ 0.8 & 0.1 & 0 & 0 & 0.1 \\ 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0.1 & 0.8 & 0.1 \end{bmatrix} \times \begin{bmatrix} 0.06 \\ 0.1 \\ 0.65 \\ 0.15 \\ 0.04 \end{bmatrix} = \begin{bmatrix} 0.053 \\ 0.062 \\ 0.151 \\ 0.545 \\ 0.189 \end{bmatrix}$$

取$\gamma_t = 2$，

$$\mathbf{w}_t = \frac{\widetilde{\mathbf{w}}_t^{\gamma_t}}{\sum_j \widetilde{w}_t(j)^{\gamma_t}} = \begin{bmatrix} 0.0078 \\ 0.0106 \\ 0.0630 \\ 0.8201 \\ 0.0986 \end{bmatrix}$$

可以看出来，经过锐化处理后$\mathbf{w}_t$不同元素直接的差异变得更明显了（即变得“尖锐”了），内存操作焦点将更加突出。


Previous
State
$\mathbf{w}_{t-1}$
$\mathbf{M}_t$
Controller
Outputs
$\mathbf{k}_t$
β_t
g_t
$\mathbf{s}_t$
γ_t
Content
Addressing
$\mathbf{w}_t^c$
Interpolation
$\mathbf{w}_t^g$
Convolutional
Shift
$\tilde{\mathbf{w}}_t$
Sharpening
$\mathbf{w}_t$

参考文献：

1. 论文:https://arxiv.org/abs/1410.5401
2. 论文 _ 更新控制器：http://arxiv.org/abs/1409.0473
3. 神经网络图灵机：https://www.zhihu.com/question/42029751
4. 论文中文解读：https://cloud.tencent.com/developer/article/1160561
5. https://blog.csdn.net/rtygbwwwerr/article/details/50548311

Neural Memory Streaming Recommender Networks with Adversarial Training

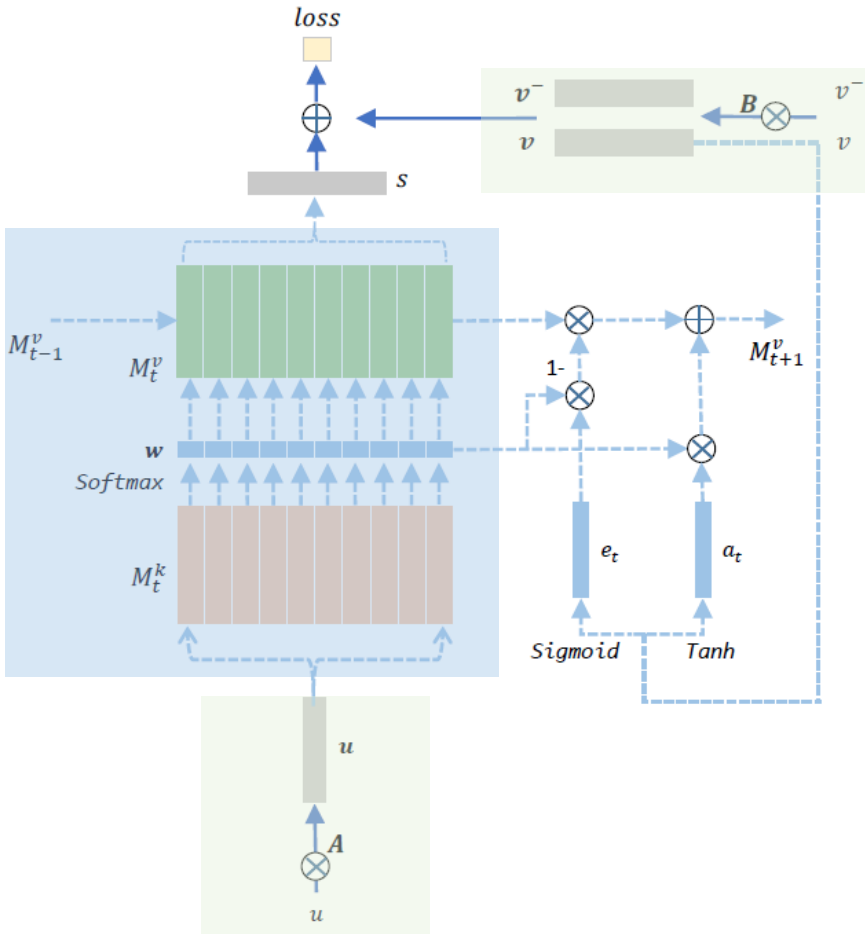


Figure 1: the Architecture of the Propose Model

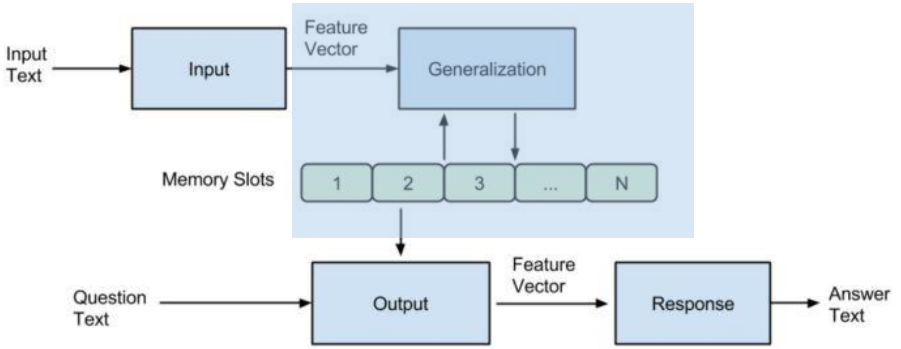
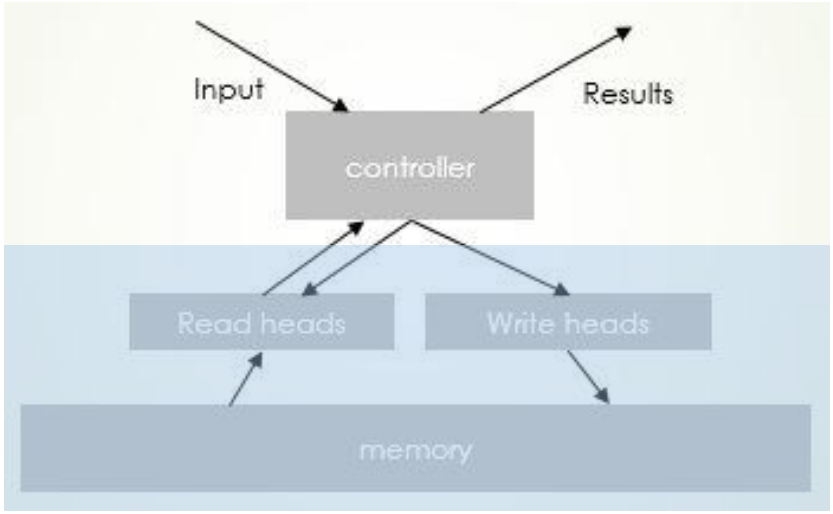


Figure 3: Memory Network architecture.



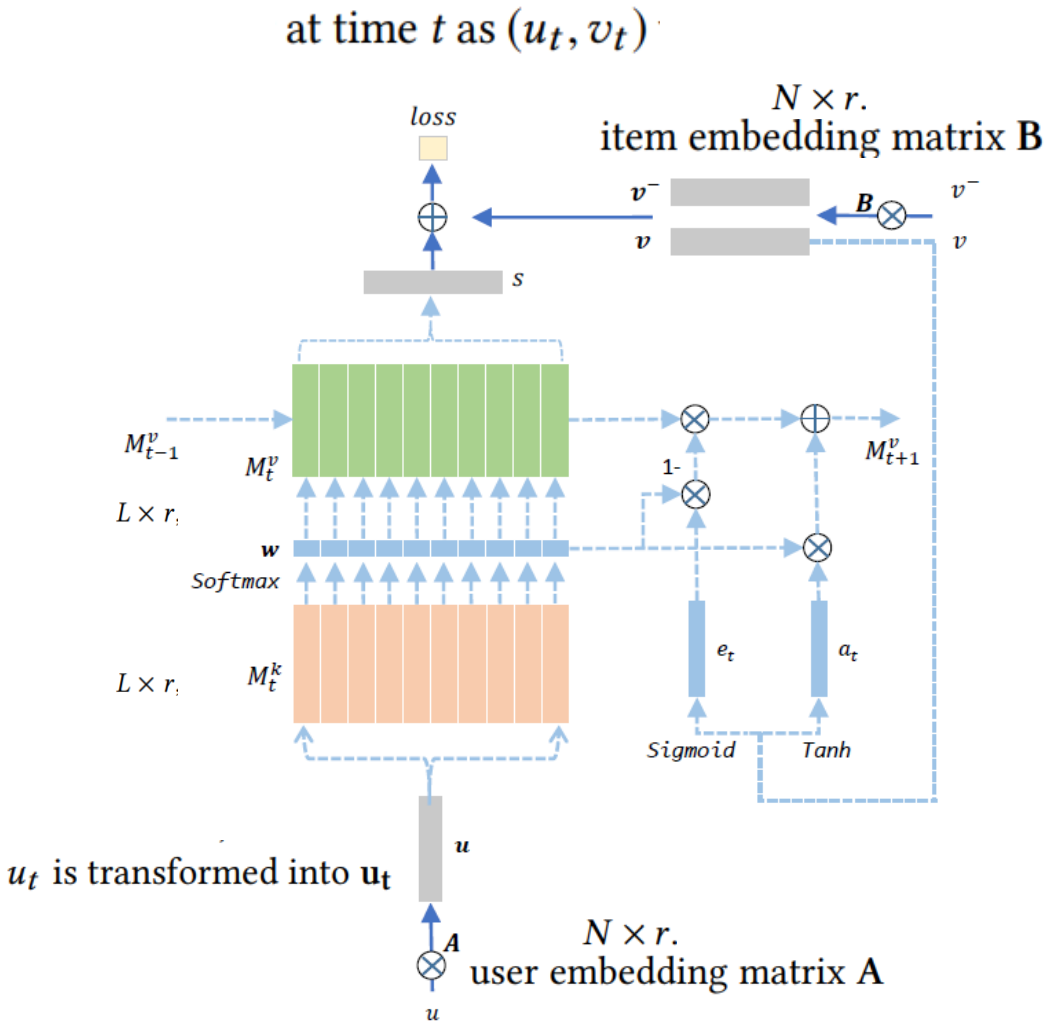
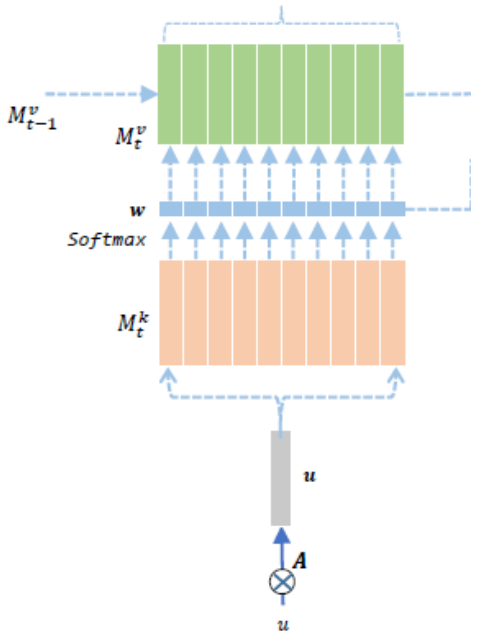


Figure 1: the Architecture of the Propose Model

The total numbers of users and items are denoted as N and M. user-item interaction pair at time t as (u_t, v_t) while v_t^- is a sampled negative item for a specific user at time t.

u_t is transformed into $\mathbf{u_t}$ by multiplied with user embedding matrix A while v_t and v_t^- are transformed into $\mathbf{v_t}$ and $\mathbf{v_t^-}$ with item embedding matrix B, where $\mathbf{u_t}$, $\mathbf{v_t}$ and $\mathbf{v_t^-}$ have real-valued vectors with r dimensions, and the size of A and B is $N \times r$. $\mathbf{M_t^k}$ and $\mathbf{M_t^v}$ respectively represent key memory matrix and value memory matrix at time t respectively, both with size of $L \times r$, where L and r stand for the number of memory slots and the dimension of each memory slot respectively. $\mathbf{w_t}$ is an attention weight vector with size L that is generated from $\mathbf{M_t^k}$ activated by a specific user u at time t. Thus, the weight vector is both user- and time-specific. Note that we leave out the subscript t of u_t, v_t, $\mathbf{u_t}$, $\mathbf{v_t}$, $\mathbf{v_t^-}$ and $\mathbf{w_t}$ to make the expressions more compact in the following sections and figures where no confusion occurs.

Memory Addressing



L latent factors for interests

Euclidean distance:

$$sim_i = \left\| \mathbf{u} - \mathbf{M}_\mathbf{t}^\mathbf{k}(i) \right\| \tag{2}$$

Then, an attention weight vector w with size L are obtained by applying the Softmax function, i.e., $Softmax(z_i) = e^{z_i} / \Sigma_j e^{(z_j)}$, to the negative similarities so that the most similar memory slot produces the largest weight:

$$w(i) = Softmax(-sim_i) \tag{3}$$

Content-base(基于内容的寻址)：

产生一个待查询的值k_t，将该与M_t中的所有N个地址的值进行比较，最相近的那个M_t(i)即为待查询的值。

首先，需要进行寻址操作的Head(Read or Write)生成一个长度为M的key vector:k_t,然后将k_t与每个M_t(i)进行相似度比较（相似度计算函数为K[u,v]），最后将生成一个归一化的权重向量,计算公式如下：

$$w_t^c(i) \leftarrow \frac{\exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(i)\right]\right)}{\sum_j \exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(j)\right]\right)}.$$

相似性度量是余弦相似度：

$$K\left[\mathbf{u}, \mathbf{v}\right] = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||}.$$

其中，β_t满足$\beta_t > 0$是一个调节因子，用以调节寻址焦点的范围。β_t越大，函数的曲线变得越发陡峭，焦点的范围也就越小。

在插值(interpolation)之后，每个读写头都会给出一个位移权重S_t，用于定义一个在允许的整数值位移上的归一化分布

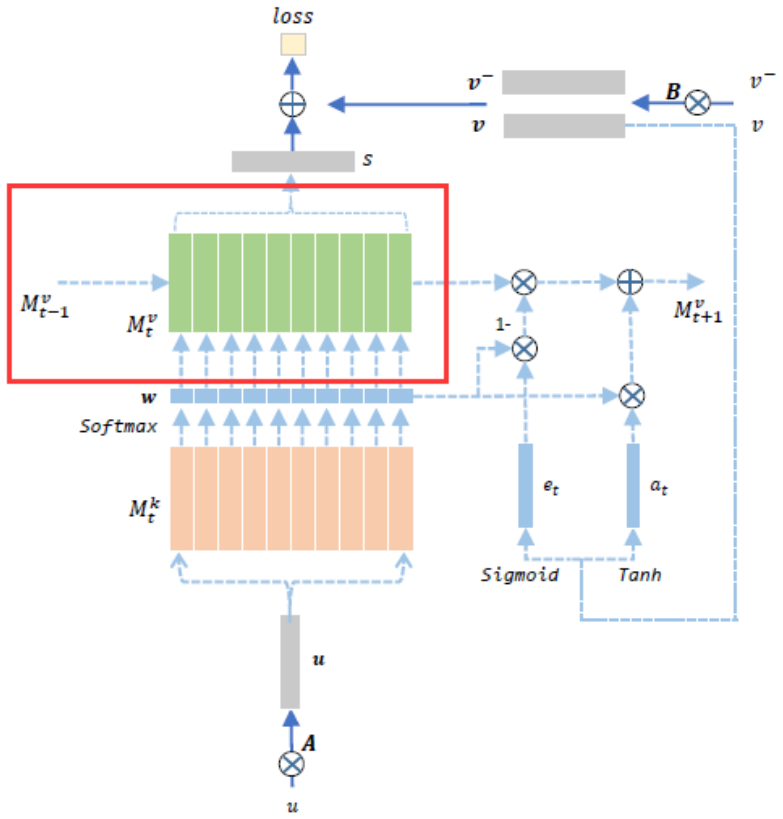
如果内存地址为0到N-1，使用s_t来旋转w_t^g，可以使用下面的循环卷积来表示：

$$\tilde{w}_t(i) \longleftarrow \sum_{j=0}^{N-1} w_t^g(j)\, s_t(i-j) \tag{8}$$

S_t写成矩阵的形式如下：　$\widetilde{w_t} = s_t w_t^g$

$$\mathbf{S}_t = \begin{bmatrix} s_t(0) & s_t(N-1) & \cdots & s_t(2) & s_t(1) \\ s_t(1) & s_t(0) & s_t(N-1) & \cdots & s_t(2) \\ \vdots & s_t(1) & s_t(0) & \ddots & s_t(2) \\ s_t(N-2) & & \ddots & \ddots & s_t(N-1) \\ s_t(N-1) & s_t(N-2) & \cdots & s_t(1) & s_t(0) \end{bmatrix}$$

Memory Reading



proxy preference vector:

$$\mathbf{s} = \sum_{i=1}^{L} w(i)\mathbf{M}_t^{\mathbf{v}}(i) \tag{4}$$

Memory Writing

Once a new user-item pair arrives at the system, the item's embedding will be written to the value memory matrix using the same attention weights w generated in the memory reading stage. The relevant memories are first erased and then the new item information such as the item's popularity is added to update the value memory.

$$\mathbf{e}_t = Sigmoid(\mathbf{W}_e\mathbf{v} + \mathbf{b}_e) \tag{5}$$

$$\tilde{\mathbf{M}}_{t+1}^{\mathbf{v}}(i) = \mathbf{M}_t^{\mathbf{v}}(i) \circ [\mathbf{I} - w(i)\mathbf{e}_t] \tag{6}$$

$$\mathbf{a}_t = Tanh(\mathbf{W}_a\mathbf{v} + \mathbf{b}_a)$$

$$\mathbf{M}_{t+1}^{\mathbf{v}}(i) = \tilde{\mathbf{M}}_{t+1}^{\mathbf{v}}(i) + w(i)\mathbf{a}_t$$

读操作：

令M_t代表时刻t的N×M内存矩阵。（N代表地址数或行数，M代表每个地址的向量大小）。令W_t在时刻t读写头在N个地址的读写比重，由于所有的权重都进行了归一化，所以W_t向量的内部元素W_t(i)满足：

$$\sum_i w_t(i) = 1, \qquad 0 \le w_t(i) \le 1, \forall i. \tag{1}$$

那么，时刻t读取到的值r_t，可以定义为每个地址的向量Mt(i)加权和：sum

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i), \tag{2}$$

写操作：

受LSTM中的forget gate和输入的启发，我们将写操作拆分成两个部分：先擦除(erase)在添加(add)。

给定t时刻的写头权重w_t，以及一个擦出向量e_t，其中M个元素均在0~1范围内，则t-1时刻的内存向量在t时刻将按下式进行调整：

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\left[\mathbf{1} - w_t(i)\mathbf{e}_t\right], \tag{3}$$

其中1是一个全部是1的行向量。当对应位置权重和擦除元素都是1时，整个内存就会被重置为零。若权重为零或者擦除向量为零，则内存保持不变。当多个写头同时存在时，多个操作可以以任意顺序相互叠加，乘法是可交换的。

添加(add)操作：而加向量a_t，是在擦除动作之后执行下面动作
：

$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\,\mathbf{a}_t. \tag{4}$$

权重是通过两种寻址机制以及一些其他补充机制的共同作用产生的

The pairwise loss function is defined as follows:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{S}, v^- \sim \mathcal{V}_u^-} w_{u,v} * [m + d(u,v) - d(u,v^-)]_+ \qquad (9)$$

where $[z]_+ = max(z;0)$ denotes the standard Hinge-loss, $w_{u,v}$ is the ranking loss weight (described later) and $m > 0$ is the safety margin size; $\mathcal{S}$ is the user activities data currently available for training, $\mathcal{V}_u^-$ is a set of items that u has never interacted with, currently, we simply assume that each negative item v^- is uniformly drawn from $\mathcal{V}_u^-$, but we will introduce a better-designed adaptive negative sampling method based on GAN in Section 3; $d(u,v)/d(u,v^-)$ is the distance between u's proxy preference vector $\mathbf{s}$ and the positive/negative item vector $\mathbf{v}/\mathbf{v}^-$.

$$w_{u,v} = \log(rank_{u,v} + 1) \tag{10}$$

This scheme penalizes a positive item at a lower rank much more heavily than at the top. However, it is computationally inefficient to rank all items for each user. To avoid ranking all items, we adopt the following approximate method to obtain $rank_{u,v}$: for each positive item v of user u, it needs to draw $N_{u,v}$ negative items until a v^- satisfying $d(u,v) - d(u,v^-) + m > 0$, i.e., an impostor, is found. Then the approximate rank is calculated as:

$$rank_{u,v} \approx \lfloor \frac{N-1}{N_{u,v}} \rfloor \tag{11}$$

Recall that N is the total number of items.

Hinge loss

Hinge loss 的叫法来源于其损失函数的图形，为一个折线，通用的函数表达式为：

$$L(m_i) = max(0, 1 - m_i(w))$$

表示如果被正确分类，损失是0，否则损失就是 $1 - m_i(w)$ 。

3.2.2 Generator. The objective of the generator maximizes the expectation of $-d_D(u, v^-)$ using the generated items so as to encourage the generator to generate plausible items to confuse the discriminator:

$$\mathcal{L}_G = \sum_{(u,v)\in\mathcal{S}} \mathbb{E}[-d_D(u, v^-)] \tag{13}$$
$$(u, v^-) \sim P_G(u, v^-|u, v)$$

The distribution $P_G(u, v^-|u, v)$ is modeled as:

$$P_G(u, v^-|u, v) = \frac{exp(-d_G(u, v^-))}{\sum_{\bar{v}\in\mathcal{V}_u^-} exp(-d_G(u, \bar{v}))} \tag{14}$$

where d_G is the Euclidean distance between user u and item v measured by the generator though a neural network. We embed u with

where, analogously, (u, v) is the *state*, $P_G(u, v^-|u, v)$ is the *policy*, (u, v^-) is the *action* and $-d_D(u, v^-)$ is the *reward*.

where, analogously, (u, v) is the *state*, $P_G(u, v^-|u, v)$ is the *policy*, (u, v^-) is the *action* and $-d_D(u, v^-)$ is the *reward*.

$$\mathcal{L}_G = \sum_{(u,v)\in\mathcal{S}} \mathbb{E}[-d_D(u, v^-)] \tag{13}$$
$$(u, v^-) \sim P_G(u, v^-|u, v)$$

The distribution $P_G(u, v^-|u, v)$ is modeled as:

$$P_G(u, v^-|u, v) = \frac{exp(-d_G(u, v^-))}{\sum_{\bar{v}\in\mathcal{V}_u^-} exp(-d_G(u, \bar{v}))} \tag{14}$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta E[R(S, A)|\pi_\theta] \\
&= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a|s) R(s, a) \\
&= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(a|s) R(s, a) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} R(s, a) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \nabla_\theta log\pi_\theta(a|s) R(s, a) \\
&= E[\nabla_\theta log\pi_\theta(a|s) R(s, a)]
\end{aligned}$$

$$\begin{aligned}
\nabla_{\theta_G} \mathcal{L}_G &= \sum_{(u,v)\in\mathcal{S}} \mathbb{E}_{(u,v^-)\sim P_G} [-d_D(u, v^-) \nabla_{\theta_G} logP_G(u, v^-|u, v)] \\
&\simeq \sum_{(u,v)\in\mathcal{S}} \frac{1}{T} \sum_{(u_i, v_i^-)\sim P_G, i=1}^{T} [-d_D(u_i, v_i^-) \nabla_{\theta_G} logP_G(u_i, v_i^-|u, v)]
\end{aligned} \tag{15}$$

强化学习公式推导：

1. **Policy Gradient 理论推导**:https://zhuanlan.zhihu.com/p/31278940
2. **强化学习：**https://zhuanlan.zhihu.com/p/51476952

Algorithm 1: The Adversarial Training Algorithm

INPUT: *user-item* interaction history $(u, v) \in \mathcal{S}$, margin m and learning rates λ_D and λ_G;
OUTPUT: the discriminator D, the generator G and their distance measurement function d_D and d_G;
Initialize the parameters θ_D and θ_G for D and G;
$b = 0$; //initiate the baseline
Pre-train D and G w.r.t. to θ_D and θ_G;
while *not convergent* **do**
    Sample a mini-batch $\mathcal{S}_{batch} \in \mathcal{S}$; $\Delta_G = 0$; $\Delta_D = 0$; $R_{batch} = 0$;//zero the gradients of D and G and rewards
    for $(u, v) \in \mathcal{S}_{batch}$ **do**
        Compute its ranking weight $w_{(u,v)}$;
        Uniformly randomly sample T negative items: $\mathcal{V}_u^-$;
        Measure the sampling probability distribution: $P_G(u, v^-|u, v) = \frac{exp(-d_G(u,v^-))}{\sum_{\bar{v} \in \mathcal{V}_u^-} exp(-d_G(u,\bar{v}))}$;
        Sample a negative item v^- forming (u, v^-) by the distribution $P_G(u, v^-|u, v)$;
        Freeze the weights of G;
        $\Delta_D = \Delta_D + \nabla_{\theta_D}\{w_{u,v} * [m + d_D(u, v) - d_D(u, v^-]_+\}$; // accumulate the example gradient to Δ_D
        $R_{batch} = R_{batch} + (-\mathcal{D}(u, v^-))$;
        Freeze the weights of D;
        $\Delta_G = \Delta_G + (R_{batch} - b)\nabla_{\theta_G} \log p_G$; // accumulate the example gradient to Δ_G
    end
    $\theta_D = \theta_D - \lambda_D \Delta_D$;$\theta_G = \theta_G + \lambda_G \Delta_G$; //minimize the loss of D while maximize the likelihood of G
    $b = \frac{R_{batch}}{|\mathcal{S}_{batch}|}$; //update the baseline
end