图神经网络导论

循环图神经网络

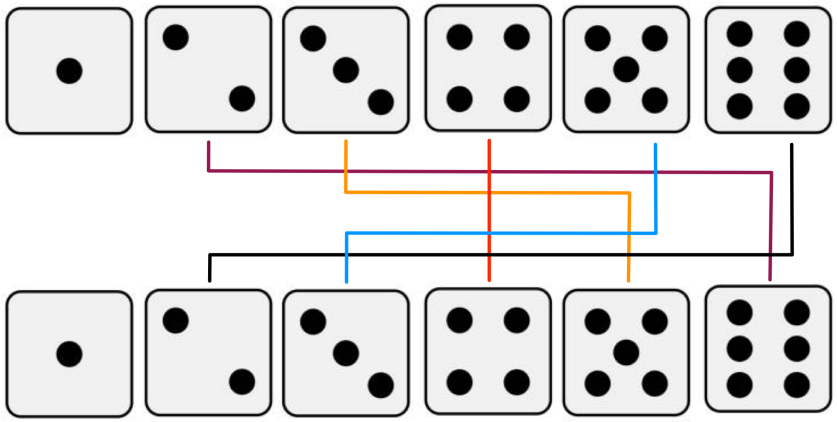
授课教师：周晟

浙江大学　软件学院

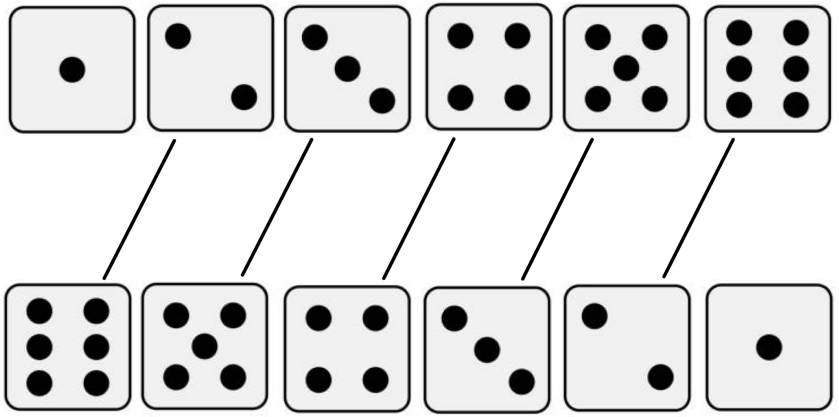
2021.11

课程内容

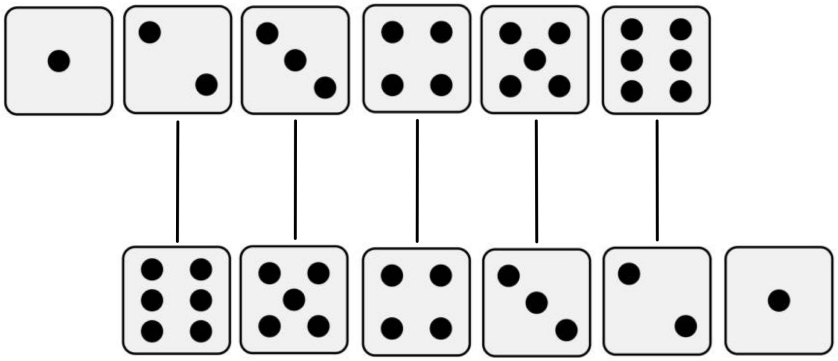
生活中的卷积



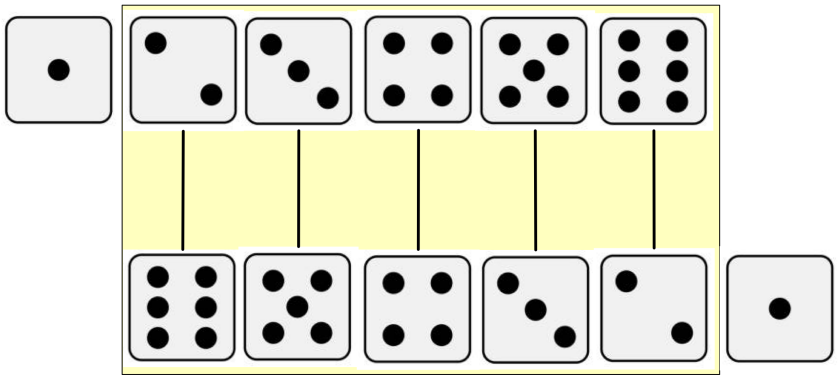
解



卷



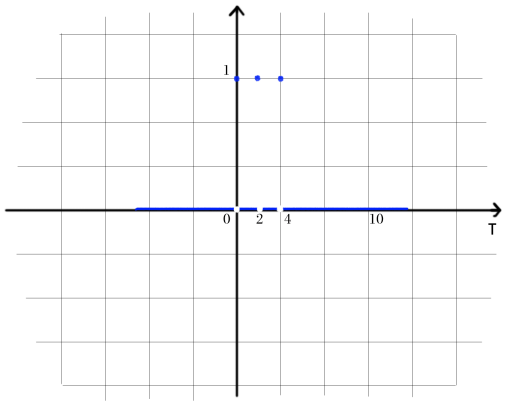
移



积

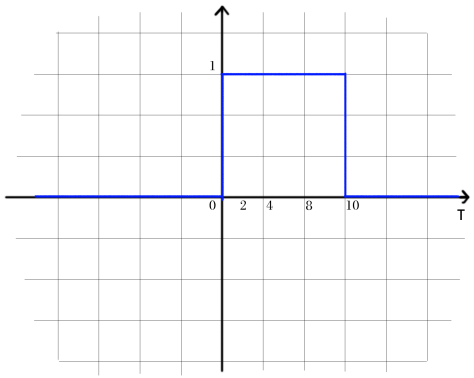
生活中的卷积

肥宅快乐问题



间隔的快乐

肥宅快乐问题



连续的快乐

数学上的卷积

卷积的定义

卷积是一种定义在两个函数 f, g 上的数学操作 $(f * g)(n)$ ：

1. 连续卷积:

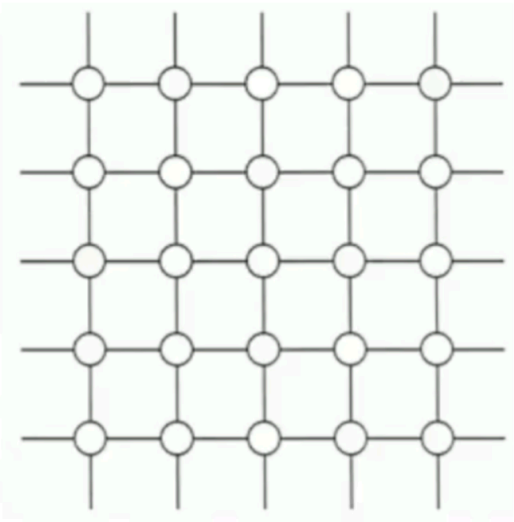
$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau) g(n - \tau) d\tau$$

2. 离散卷积:

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau) g(n - \tau)$$

卷积的三个基本操作：翻转，滑动，叠加

从图像卷积到图卷积



Grid-like network

Irregular network

图卷积的困难

1. 节点没有顺序（排列不变性）
2. 节点邻居没有固定个数（难以定义统一的特征提取函数）

图卷积

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

两个函数的卷积可以写成傅里叶变换的形式。

图上的卷积

1. 定义图上的傅立叶变换/逆变换
2. 定义图上的信号 f
3. 定义图上的操作 g（学习目标）

图上的卷积

- 卷积可以写成傅里叶变换的形式

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

- 图傅里叶变换及其逆变换：

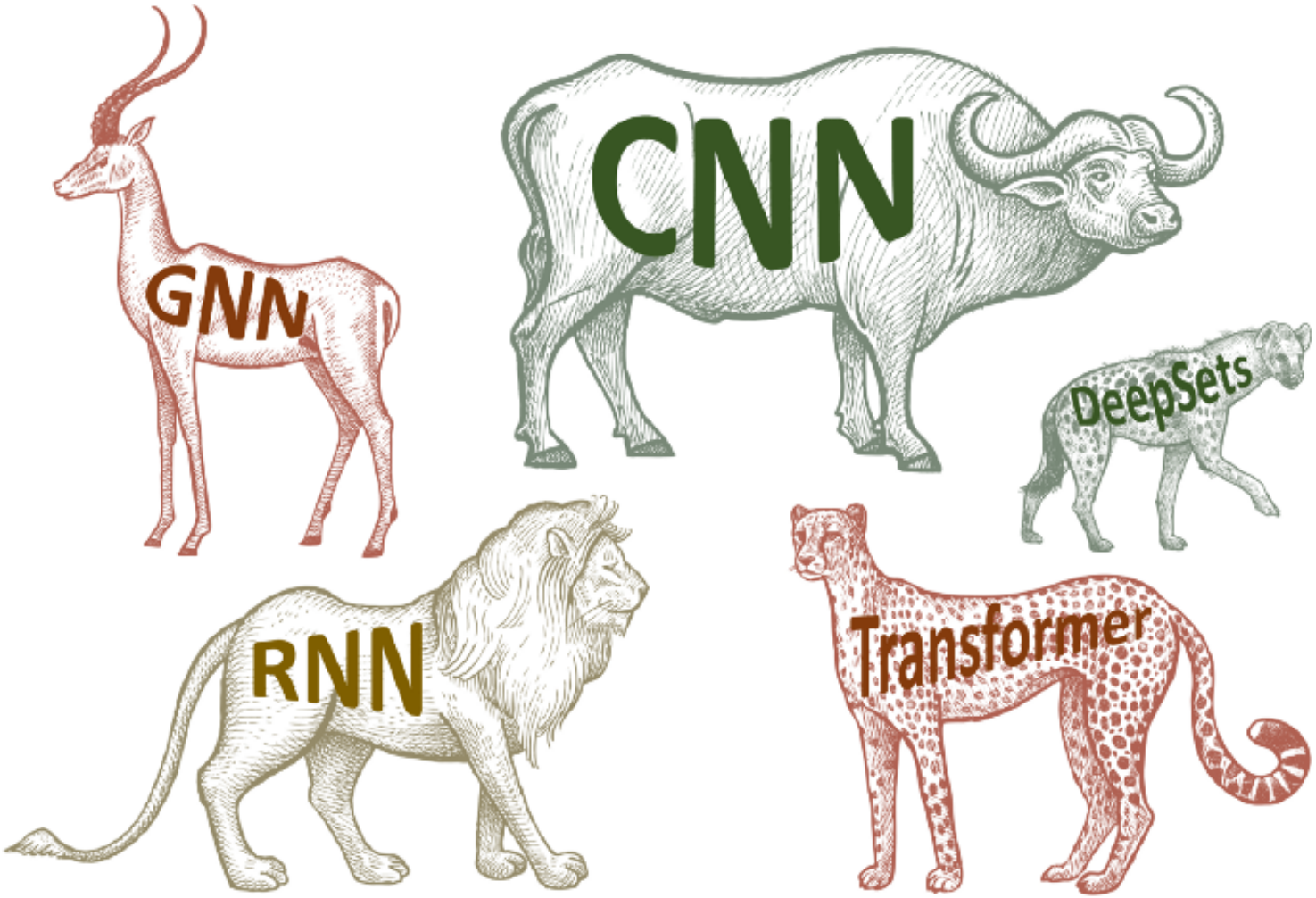
$$\mathcal{GF}\{x\} = U^T x \quad , \quad \mathcal{IGF}\{x\} = Ux$$

- 图上的卷积可以表示为：

$$x * y = U\left(\left(U^T x\right) \odot \left(U^T y\right)\right)$$

$$x * y = U g_\theta U^T x$$

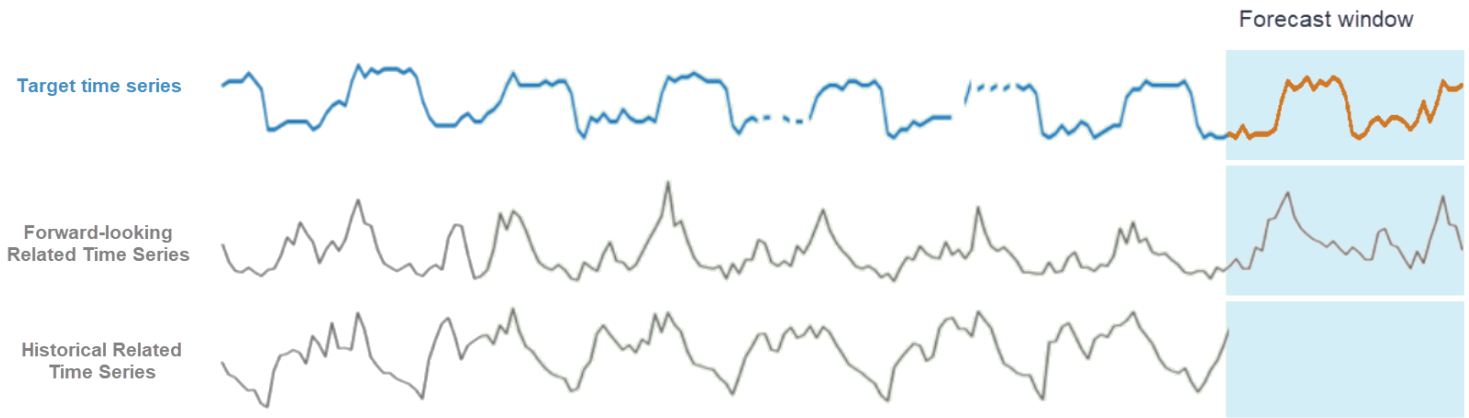
从 CNN 到 RNN



序列数据

序列数据

序列数据是一种在生产生活中常见的数据类型，它的主要特点是数据之间存在顺序，打乱顺序将改变数据整体的语义。



Hello World! This is my first line of code!

序列数据

处理序列数据的难点

1. 序列长短难以固定
2. 序列顺序信息难以保留
3. 序列影响范围难以界定

常见的序列数据学习方法：

1. RNN
2. LSTM
3. GRU

Recurrent Nerual Network

使用经典神经网络处理序列数据的问题：

- 每个样本的输出仅与当前样本有关，与其他样本无关

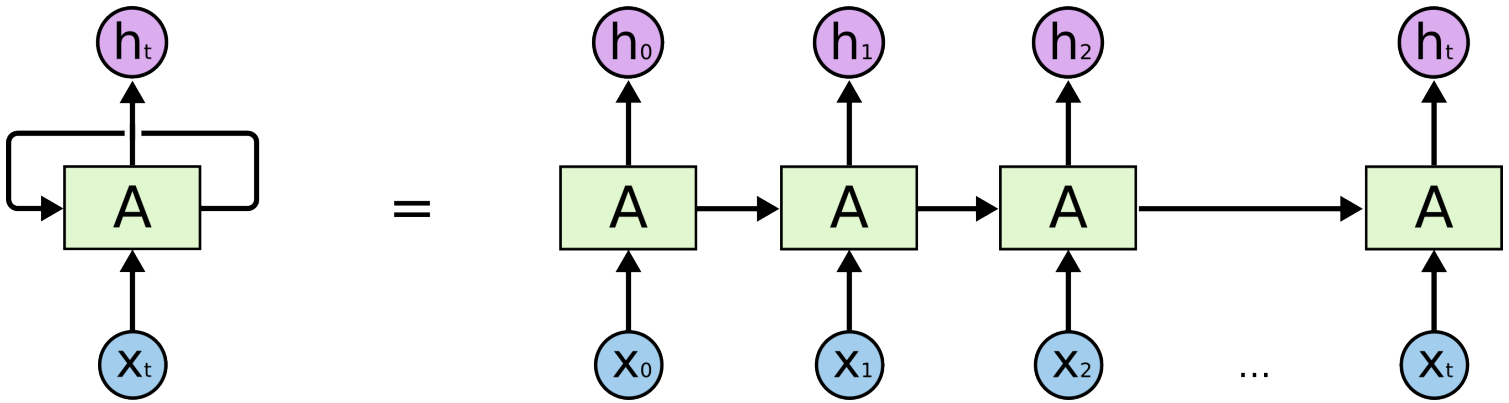
$$y = \sigma(Wx + b)$$

- 相同的样本输入将得到相同的输出
  - It's so cold that I get cold.
- 人类处理序列数据的逻辑与经典神经网络不同。人类会基于自己先前的认识来理解某一个事物，思想是具有持久性的。

RNN 通过循环网络实现信息的持久化。

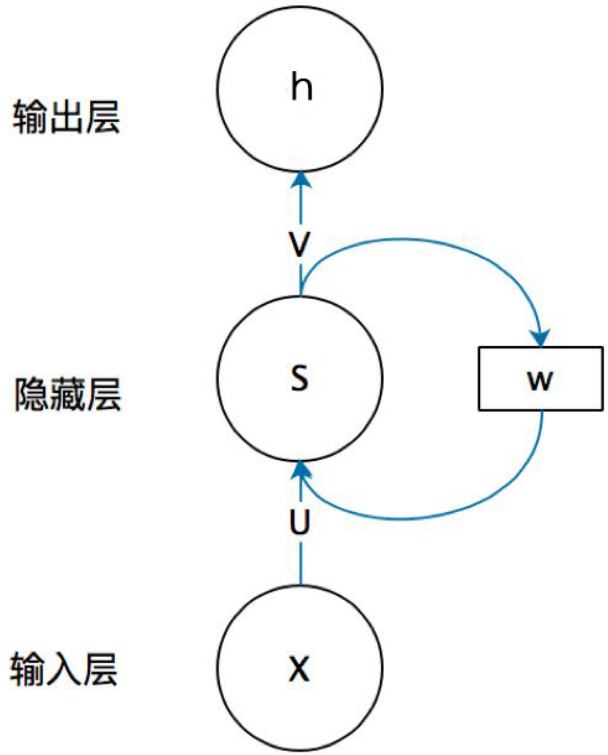
Recurrent Nerual Network

- RNN 可以被看做为同一个神经网络的多次复制，每个神经网络模块会将消息传递给下一个。

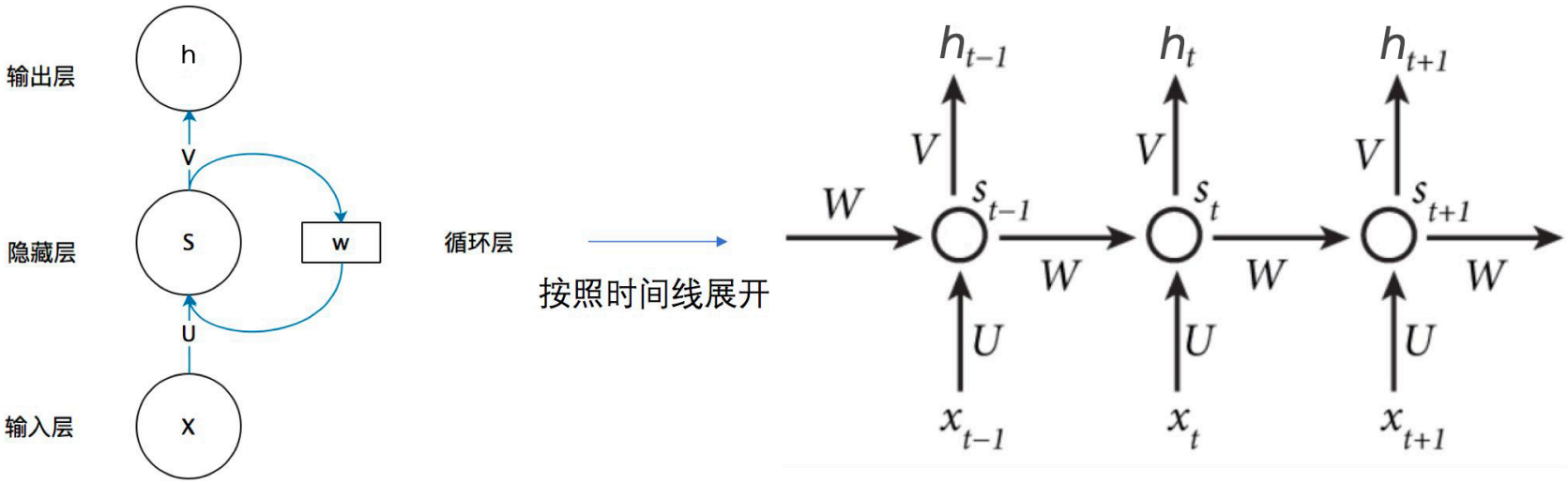


Recurrent Nerual Network

- U 和 V 分别为输入层到隐藏层和隐藏层到输出层权重矩阵。
- S 为隐藏层的值，不仅取决于输入 x，也取决于上一步中隐藏层的值。
- 权重矩阵 W 就是上一步隐藏层的值对当前步 S 的权重。



Recurrent Nerual Network

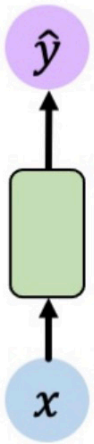


- **形式化的表示：**

$$h_t = g(V \cdot S_t)$$
$$S_t = f(U \cdot x_t + W \cdot S_{t-1})$$

其中 $f(\cdot)$ 和 $g(\cdot)$ 表示激活函数。

RNN 的结构和应用场景

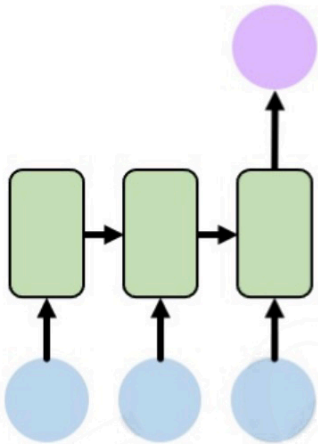


One to One

Binary Classification

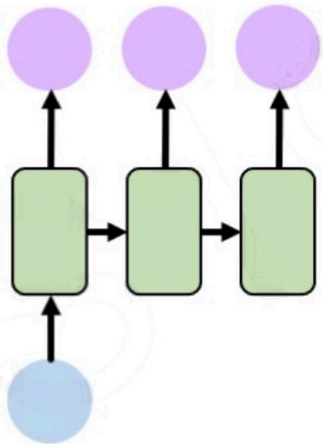
"Will I pass this class?"

Student → Pass?



Many to One

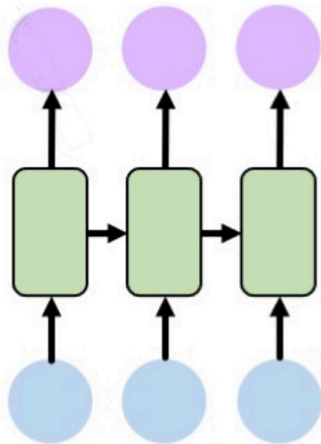
Sentiment Classification



One to Many

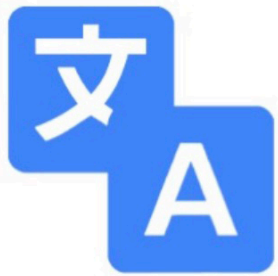
Image Captioning

"A baseball player throws a ball."



Many to Many

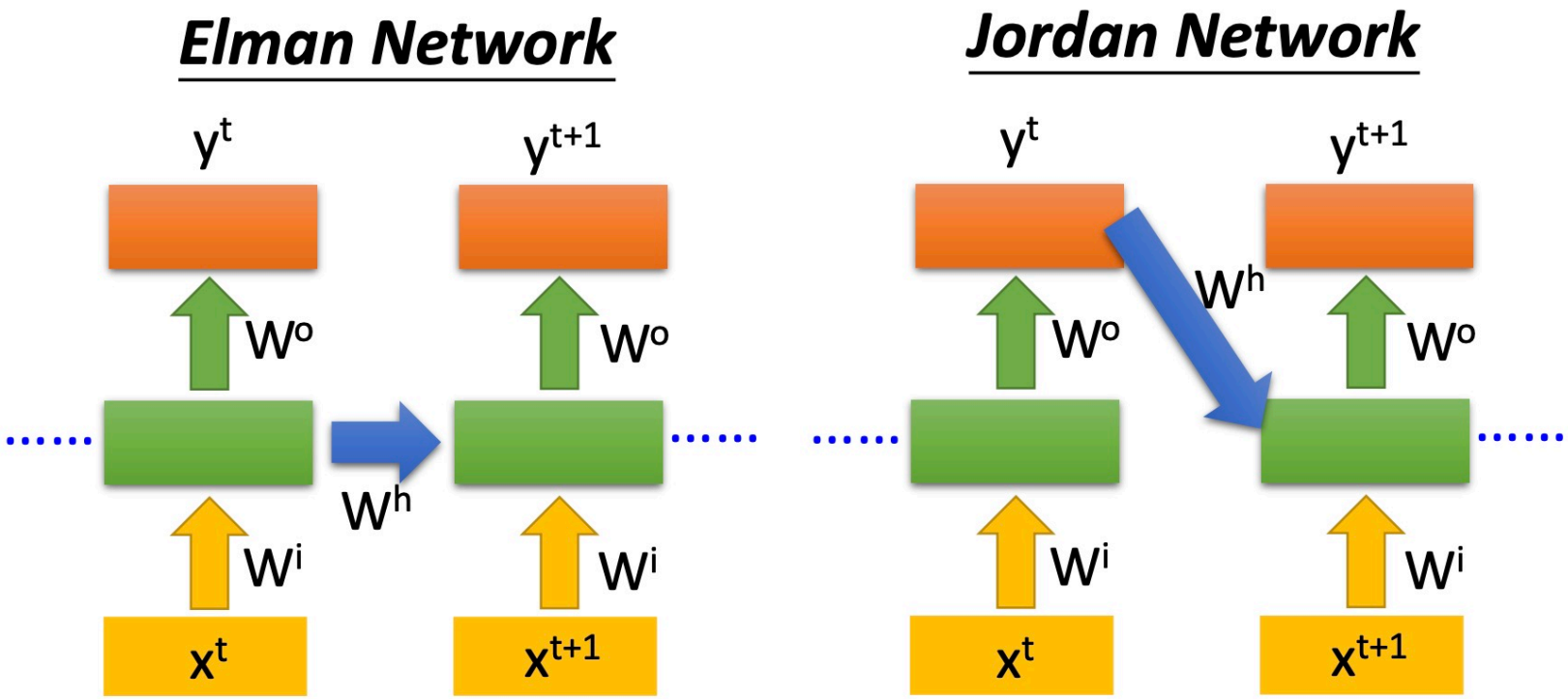
Machine Translation



Elman Network & Jordan Network

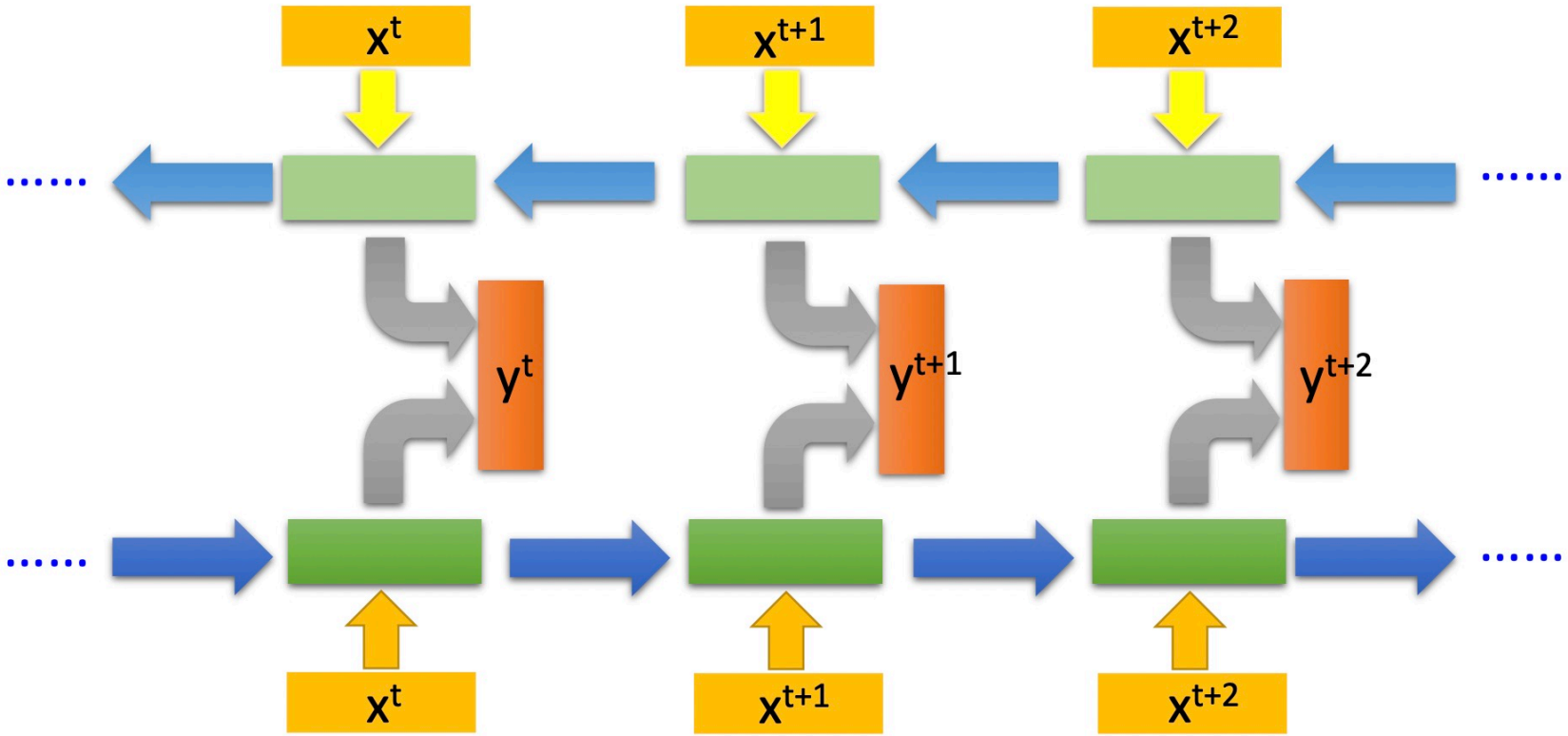
Elman Network & Jordan Network

RNN 的主要结构可以分为 Elman Network 和 Jordan Network，前者将中间状态传递给下一个单元，而 Jordan Network 将输出传递给下一个单元。



Bidirectional RNN

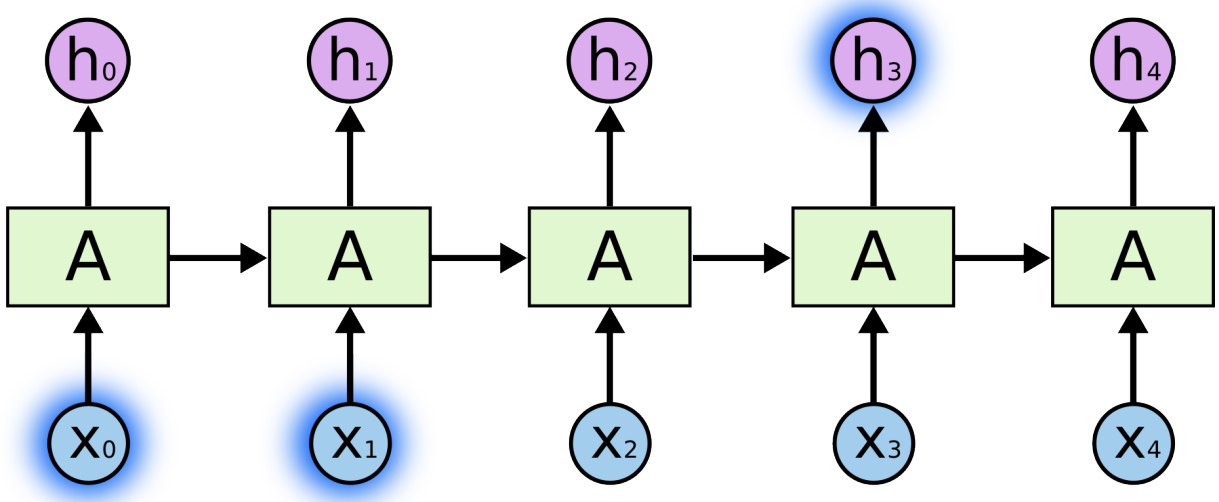
- 双向的 RNN 能够获得来自序列前后两个方向的信息，因此也会有更好的表现。



Long-Term Dependencies 长期依赖问题

- 某些情况下，仅需要知道前几步的信息就能完成预测任务。
  - “the clouds are in the sky” 中的 “sky”

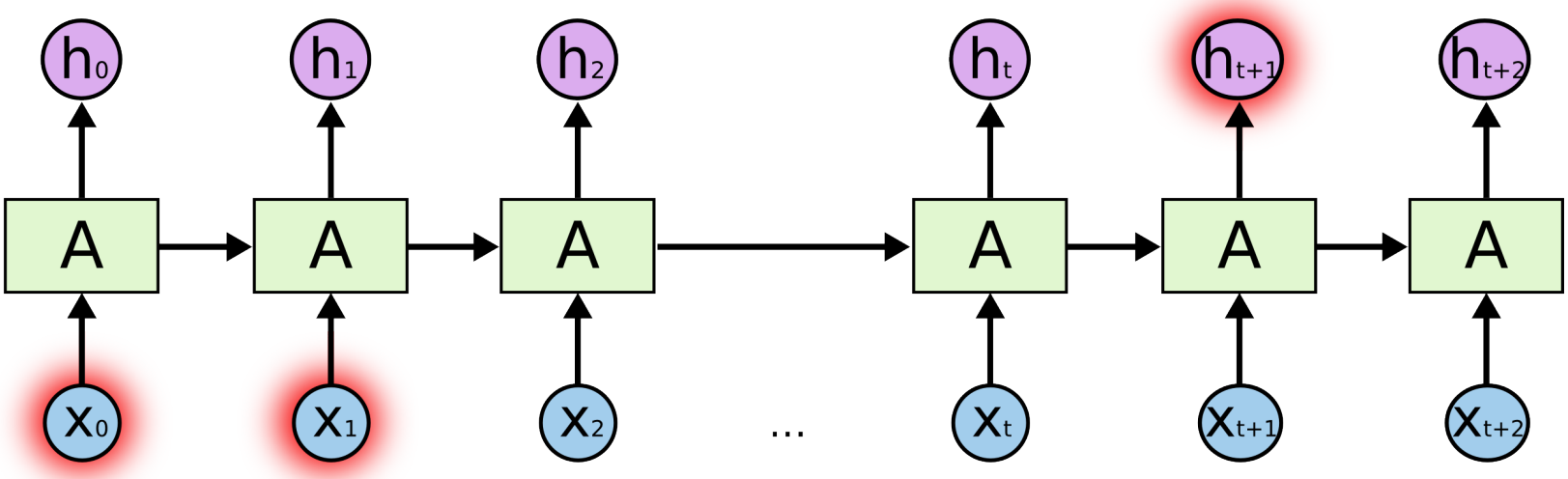
- 在这种场景中，相关信息与预测词的位置之间的间隔很小，RNN 可以轻松学会使用先前的信息。



Long-Term Dependencies 长期依赖问题

- 某些情况下，需要知道多步之前的信息才能完成预测任务。
  - “I grew up in China.....I speak fluent Chinese.” 中的 “Chinese”

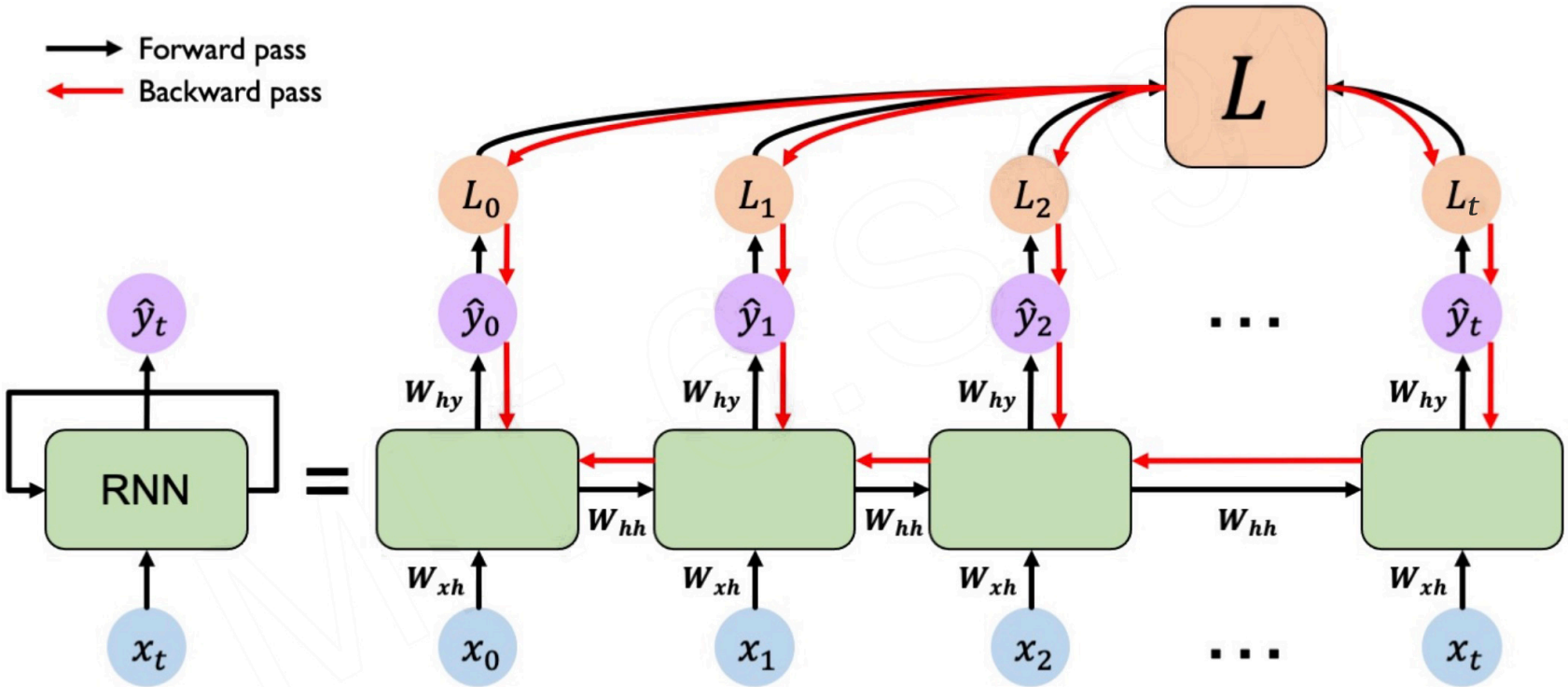
- 对于这种间隔非常大的情况，RNN 很难学习到远处的信息。



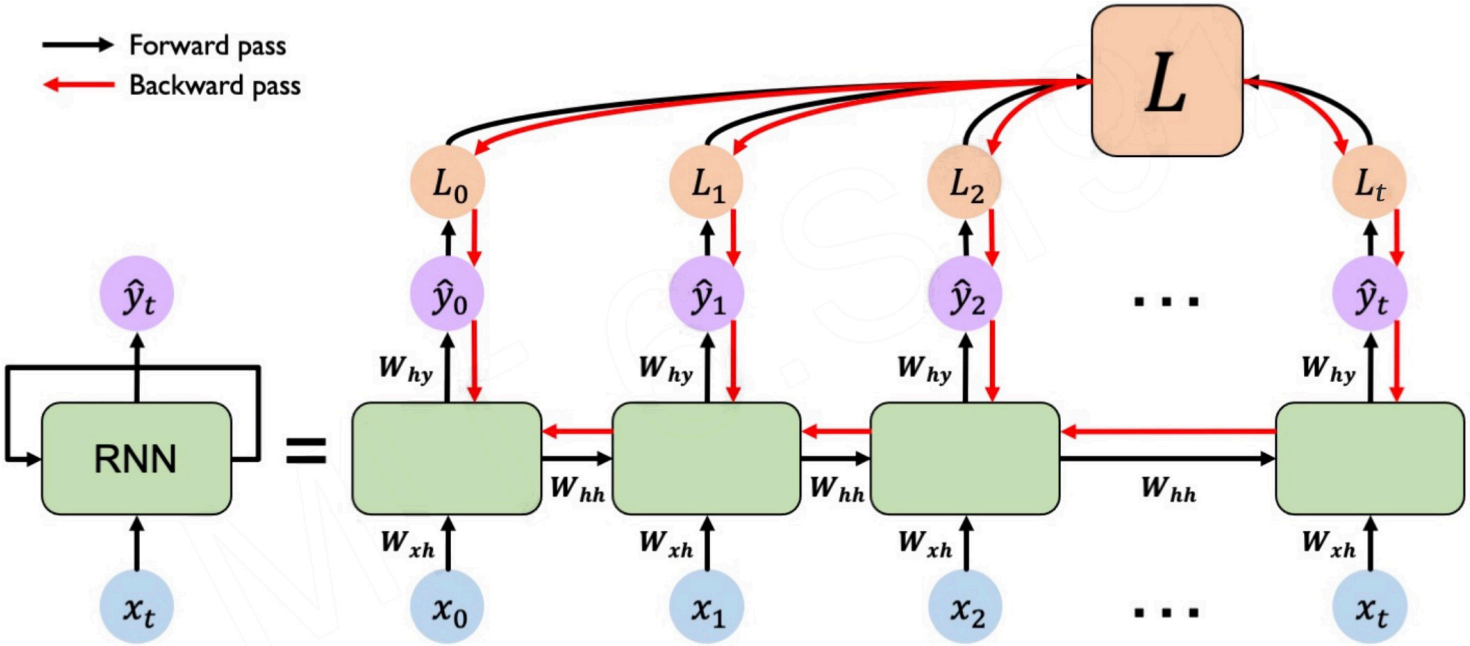
累积较远位置的信息？

Backpropagation Through Time（BPTT）

- RNN 每一层的权重 W 都是共享的。总体 Loss 由每一步的 Loss 组合而成。
- 梯度回传时也会回传到每一步，然后从每一步再依次向前回传到第 0 步（BPTT）。



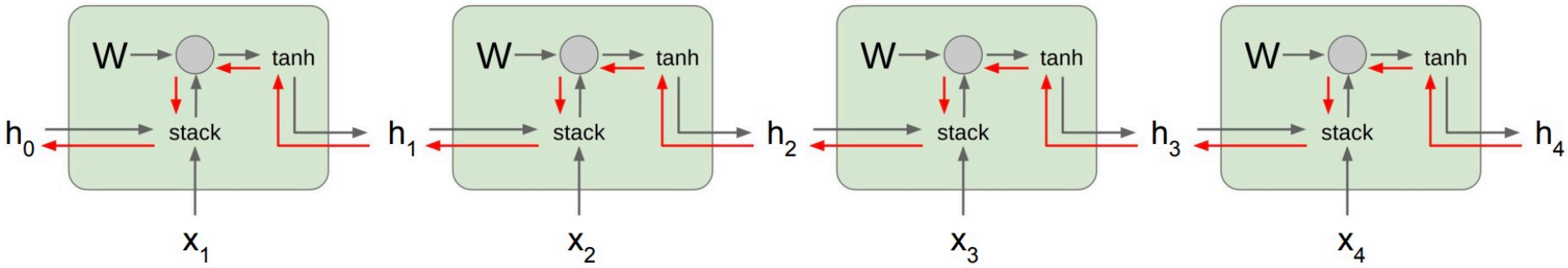
RNN 的梯度流



$$L = \sum_{t=1}^{T} L_t, \quad L_t = \text{Loss}(\hat{y}_t, y_t), \quad \hat{y}_t = W_{hy} h_t$$

$$h_t = \tanh(z_t) = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

RNN 的梯度流



- 梯度多次流经相同的权重 W，这意味着计算梯度时将包括多次关于 W 的重复计算。

RNN 的梯度流

$$h_t = \tanh(z_t) = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

- 对于一条梯度流 L_T，我们以 W_{hh} 的梯度表达式为例：

$$\frac{\partial L_T}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_T}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_T} \frac{\partial h_T}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}}$$

- 需要重点计算的是 $\frac{\partial h_T}{\partial h_t}$ 这部分：

$$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdots \frac{\partial h_{t+1}}{\partial h_t}$$

$$= \prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}} = \prod_{k=t+1}^{T} \tanh'(z_k) W_{hh}$$

RNN 的梯度流

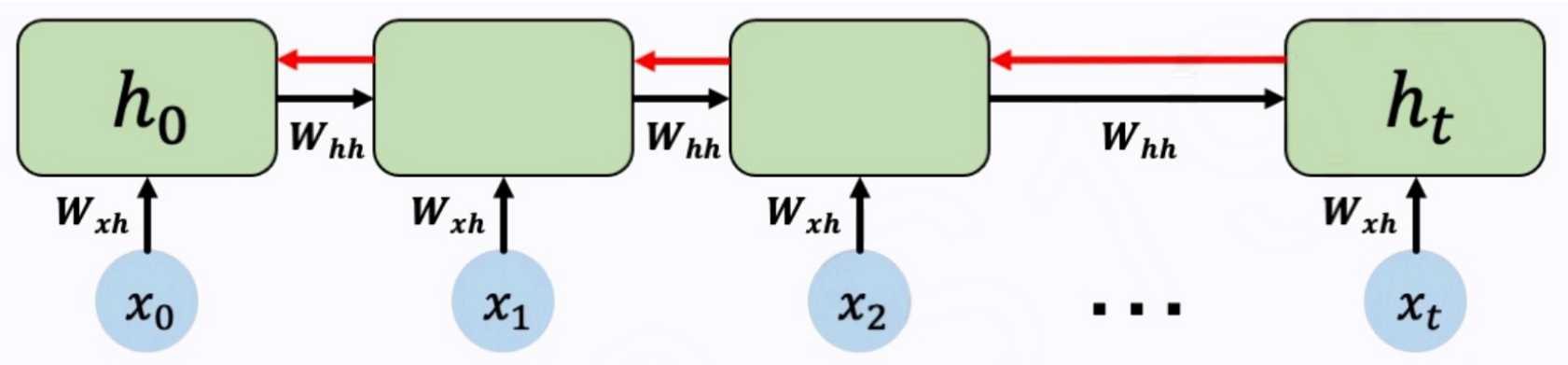
$$\frac{\partial L_T}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_T}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_T} \left(\prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}} \right) \frac{\partial h_t}{\partial W_{hh}}$$

$$= \sum_{t=1}^{T} \frac{\partial L_T}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_T} \left(\prod_{k=t+1}^{T} \tanh'(z_k) W_{hh} \right) \frac{\partial h_t}{\partial W_{hh}}$$

- 计算梯度时 W_{hh} 将会被乘上多次。简单来说，我们可以看做是许多相同数值的连乘。显然，这会带来一些问题。

RNN 的梯度流

- 如果其中有许多数值的绝对值大于 1，就会导致梯度爆炸。
- 相反，如果有许多数值的绝对值小于 1，则会导致梯度消失。

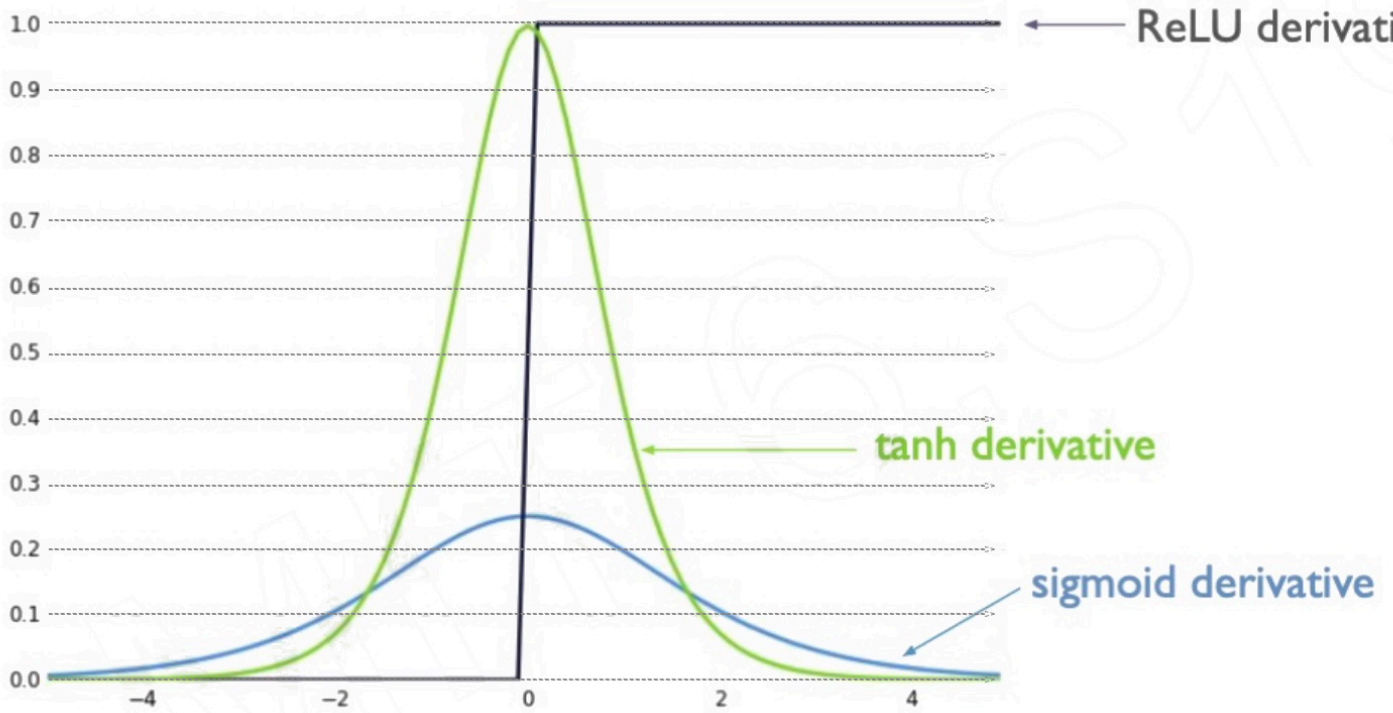


RNN 的梯度回传

- 对于梯度爆炸：
  - 梯度裁剪（Gradient clipping）：将过大的梯度裁剪到一定范围。
- 对于梯度消失：
  - 设置合适的激活函数。
  - 合适的权重初始化。
  - 设计更合理的网络结构。

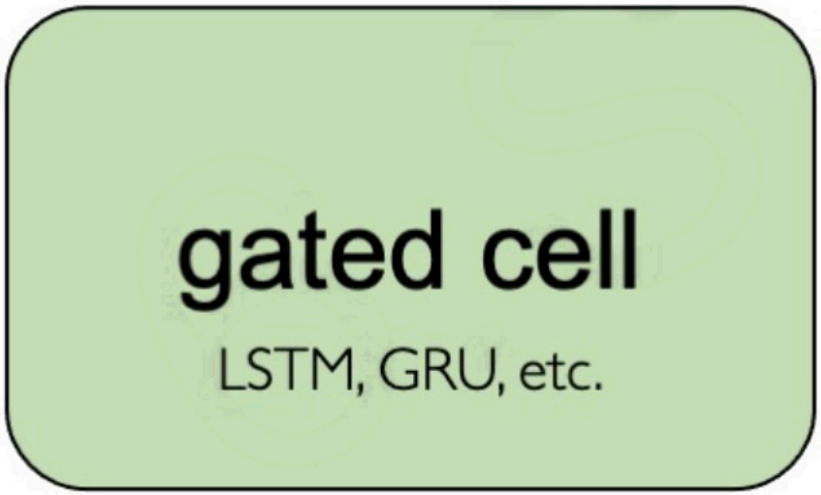
激活函数

- 使用 ReLU 作为激活函数，可以在 $x > 0$ 的时候避免梯度消失。



LSTM [Hochreiter and Schmidhuber, 1997]

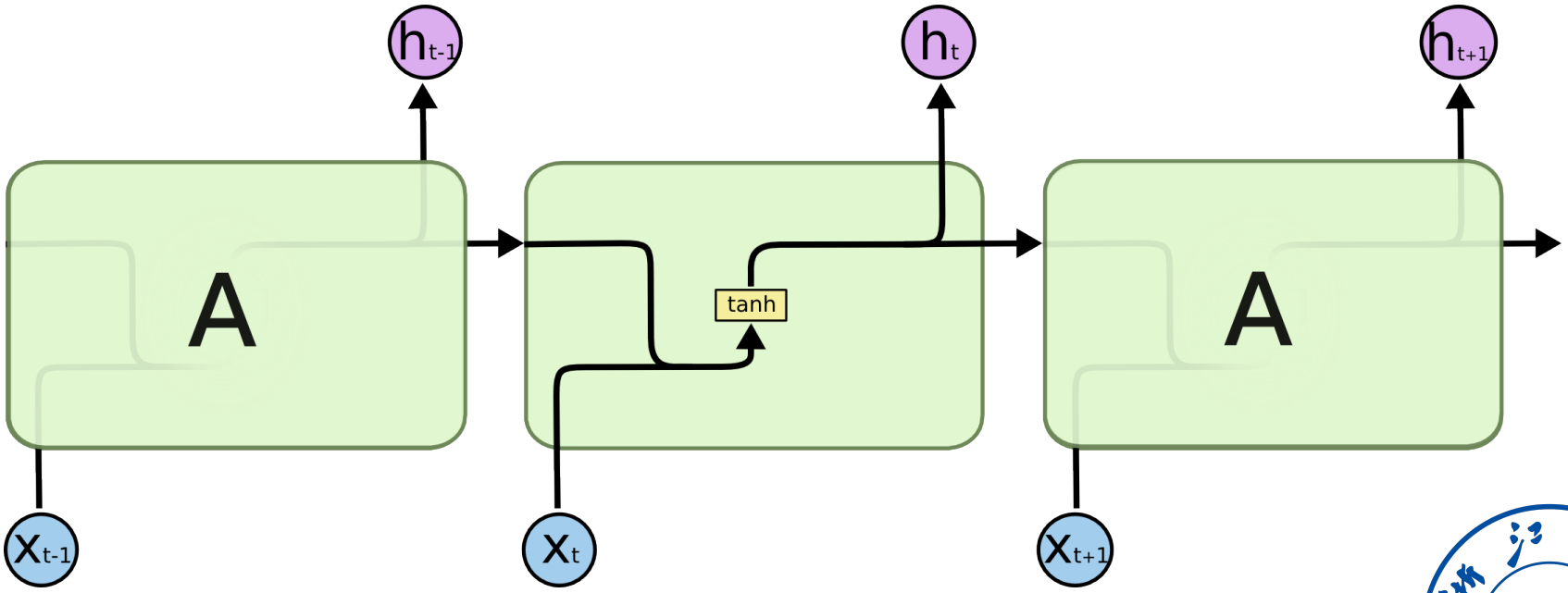
- 使用更复杂的带门控的循环单元来控制信息是否传递。



- LSTM 是一种特殊的 RNN，可以学习长期依赖信息，在许多任务上都比标准的 RNN 表现得更好——事实上，现在一般提到 RNN 指的就是 LSTM。

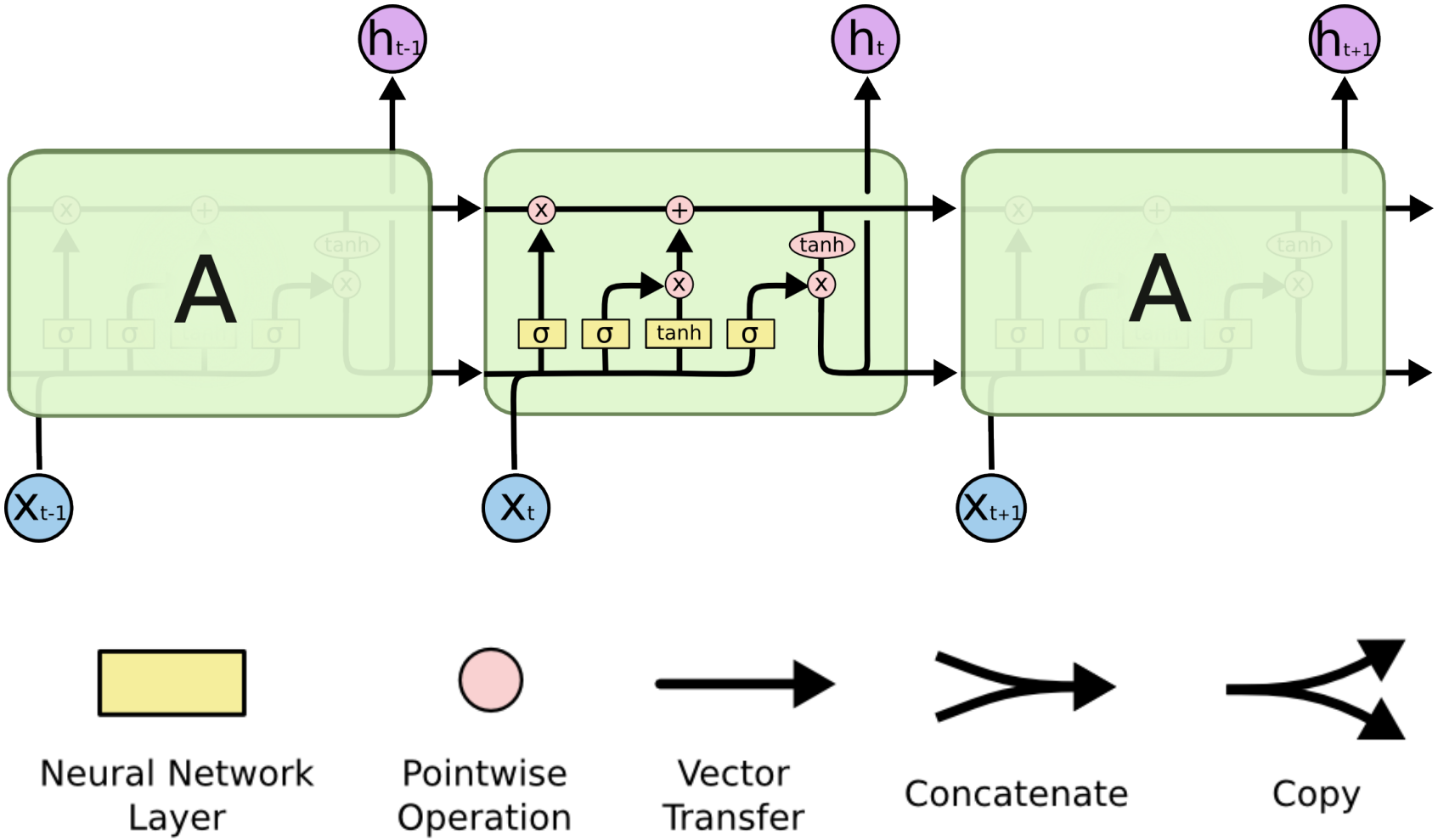
Long Short-Term Memory(LSTM)

- 在普通的 RNN 中，重复的神经网络模块 A 中只有一个非常简单的结构，例如一个 tanh 层。



Long Short-Term Memory(LSTM)

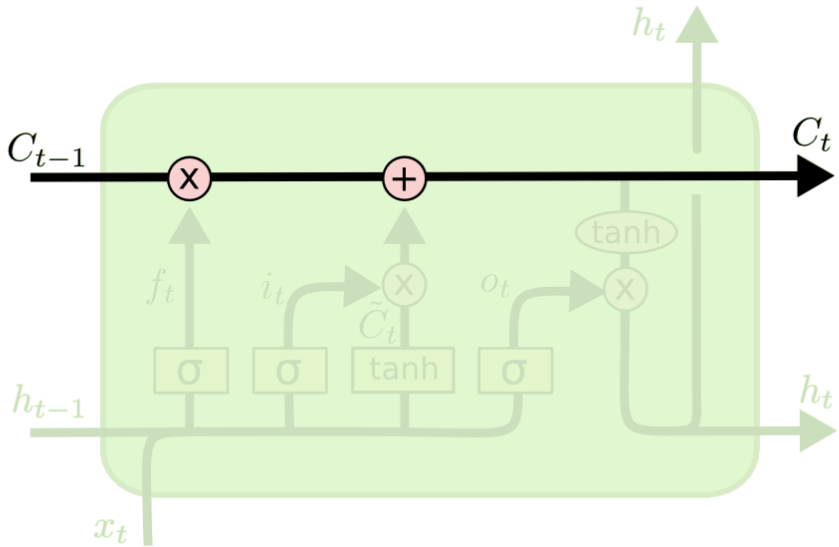
- LSTM 也是同样的结构，其改进之处主要在于重复模块的结构。主要就是三个门（Gate）的操作。



Long Short-Term Memory(LSTM)

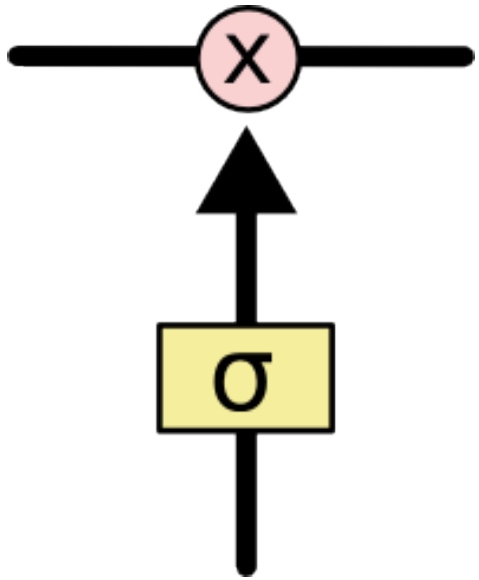
- RNN 的核心部分就是传递的数据，在 LSTM 中的形式为细胞（Cell）状态，表示为图上方的水平线。

- Cell 状态类似于传送带，只有一些少量的线性交互，因此信息可以在上面稳定地传递。



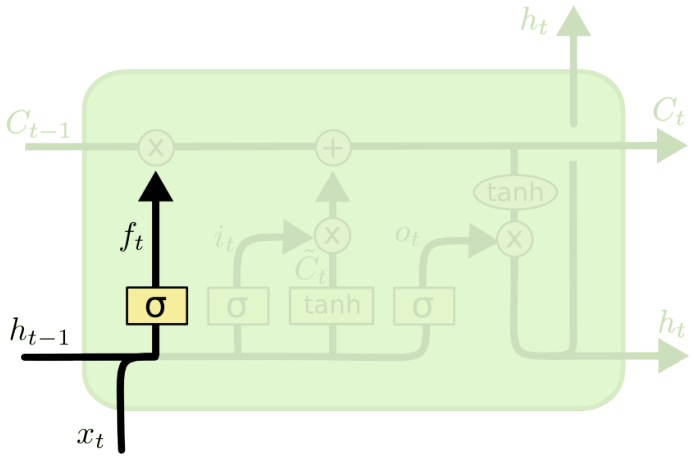
Long Short-Term Memory(LSTM)

- 门（Gate）是 LSTM 的核心部分，它是一个精心设计的结构，可以去除或增加信息到“细胞状态”。
- Gate 包含一个 sigmoid 神经网络层和一个按位的乘法操作。
- sigmoid 层输出 0 到 1 之间的数值，控制每一位有多少量可以通过。0 表示“不许任何量通过”，1 表示“允许所有量通过”。



Long Short-Term Memory(LSTM)

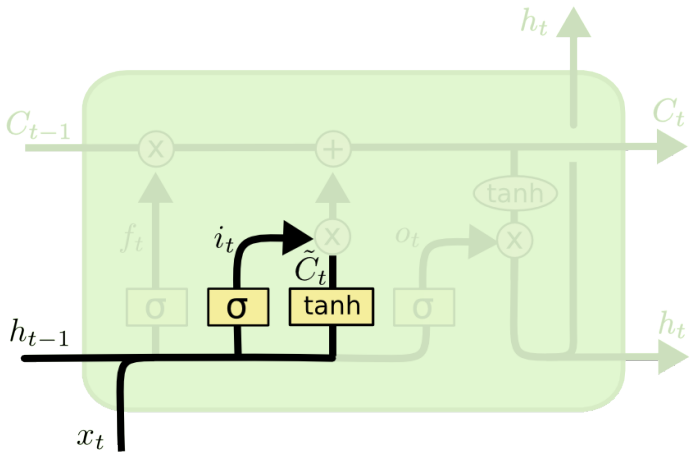
- 遗忘门（forget gate）：决定在上一步的 Cell 状态中需要丢弃哪些信息。
- 输入 h_{t-1} 和 x_t，经过网络层和 sigmoid 后，为每个 C_{t-1} 中的数字输出一个相对应的 0 到 1 之间的数值，表示其保留下来的比例。



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Long Short-Term Memory(LSTM)

- 输入门（input gate）：决定在当前步的 Cell 状态中加入哪些新的信息。
- sigmoid 层就是 input gate，生成输入比例信号，而新的信息 $\hat{C}_t$ 则是由 tanh 层来构建。为了更统一的理解，这个 tanh 层有时候也被强行叫做 gate gate，$g_t = \hat{C}_t$。

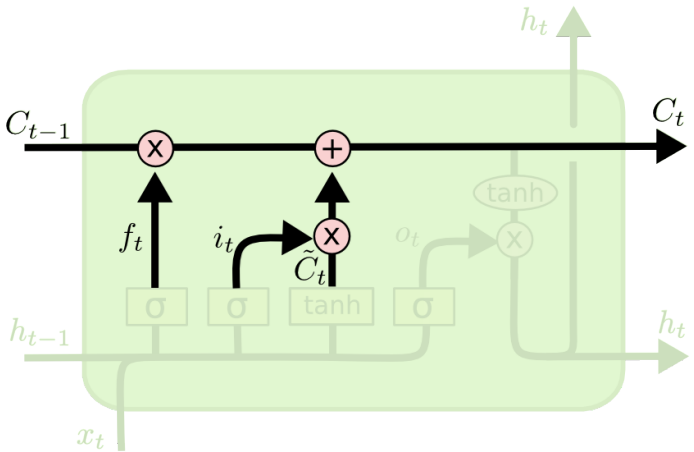


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Long Short-Term Memory(LSTM)

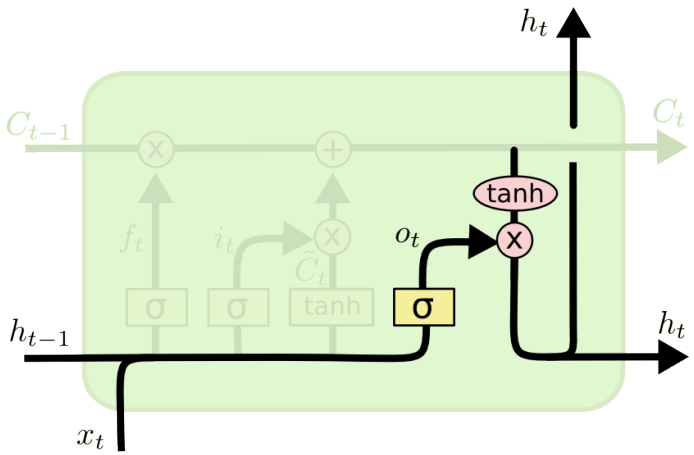
- 有了 forget gate 和 input gate 的信息，我们就可以更新 Cell 状态了。
- 首先，将旧的状态 C_{t-1} 与 forget gate 的信号 f_t 按位相乘，丢弃掉不需要的信息。而需要加入的信息由 input gate 的信号 i_t 与新信息 $\hat{C}_t$ 按位相乘获得，再将其加入到新的 Cell 状态中。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long Short-Term Memory(LSTM)

- 输出门（output gate）：决定需要输出哪些 Cell 状态。
- sigmoid 层为 output gate，生成输出比例信号，而输出的 Cell 状态需要先经过 tanh 处理（获得-1 到 1 之间的值），再与输出信号按位相乘。

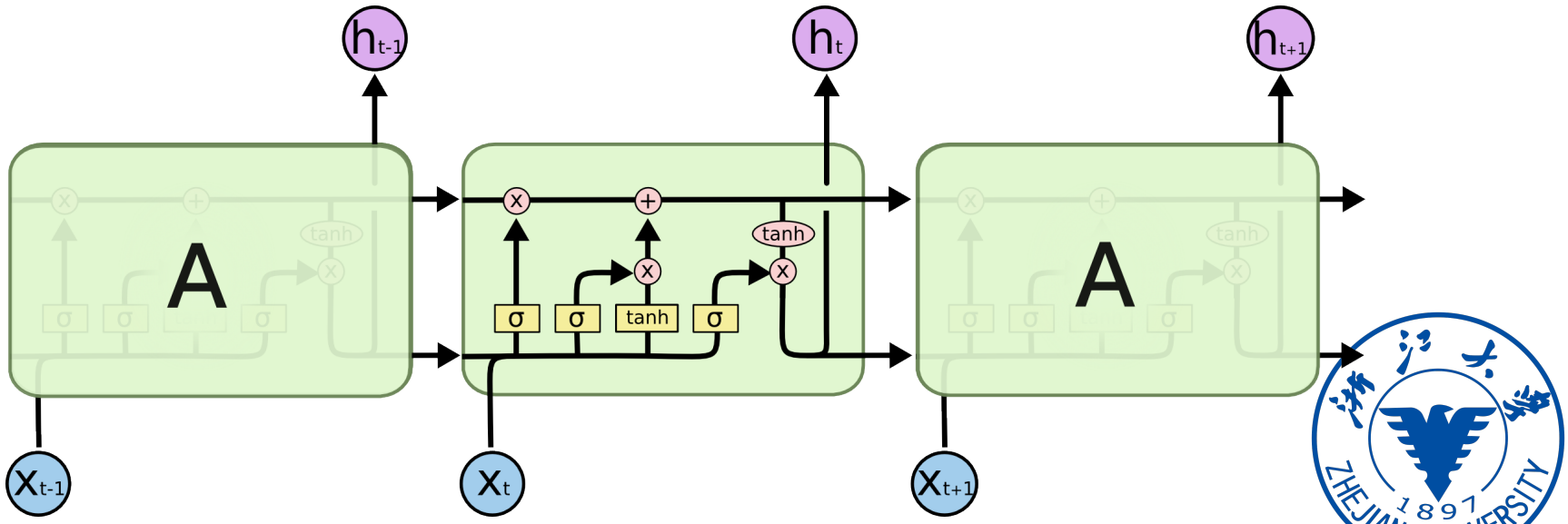


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

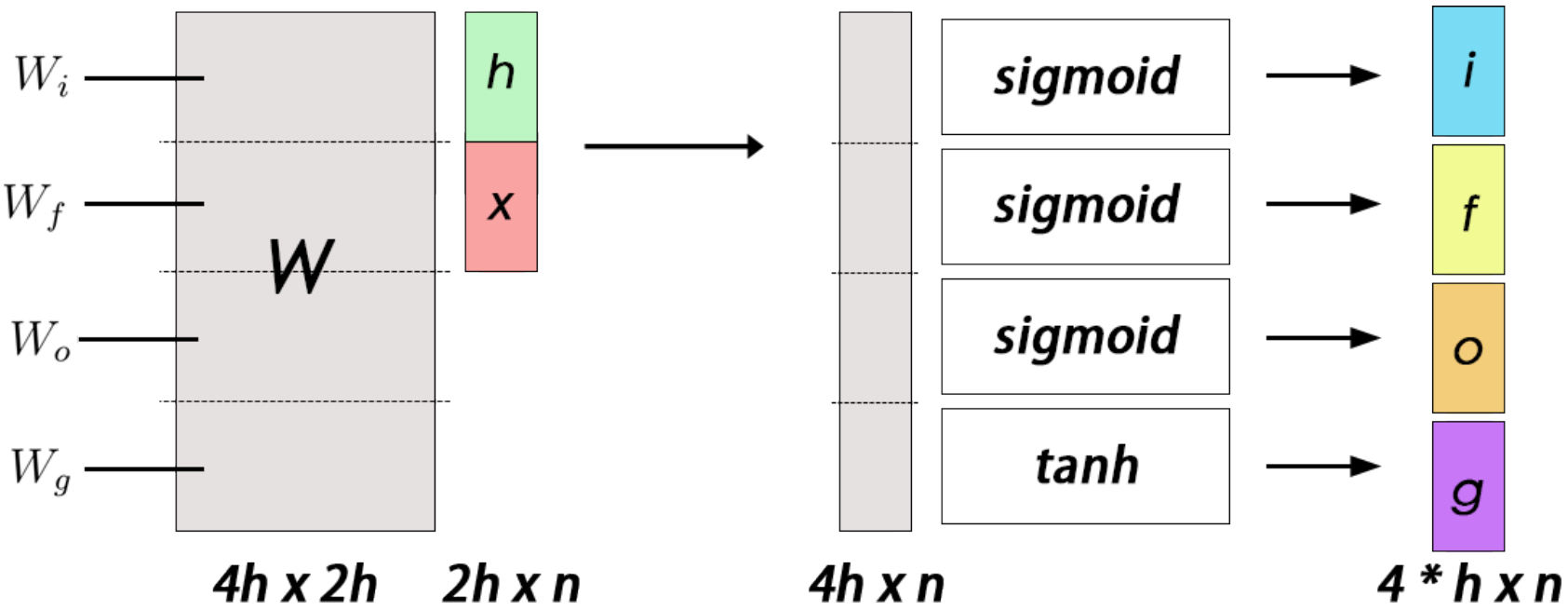
$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short-Term Memory(LSTM)

- 由此，我们逐步介绍了 LSTM，现在再来看一下整体的框架。
- LSTM 每一步的输入为上一步输出 h_{t-1} 与当前输入 x_t 的拼接 $[h_{t-1}, x_t]$，将其复制四份分别输入到 forget gate、input gate、gate gate 和 output gate 中，得到相应的门信号和新消息。

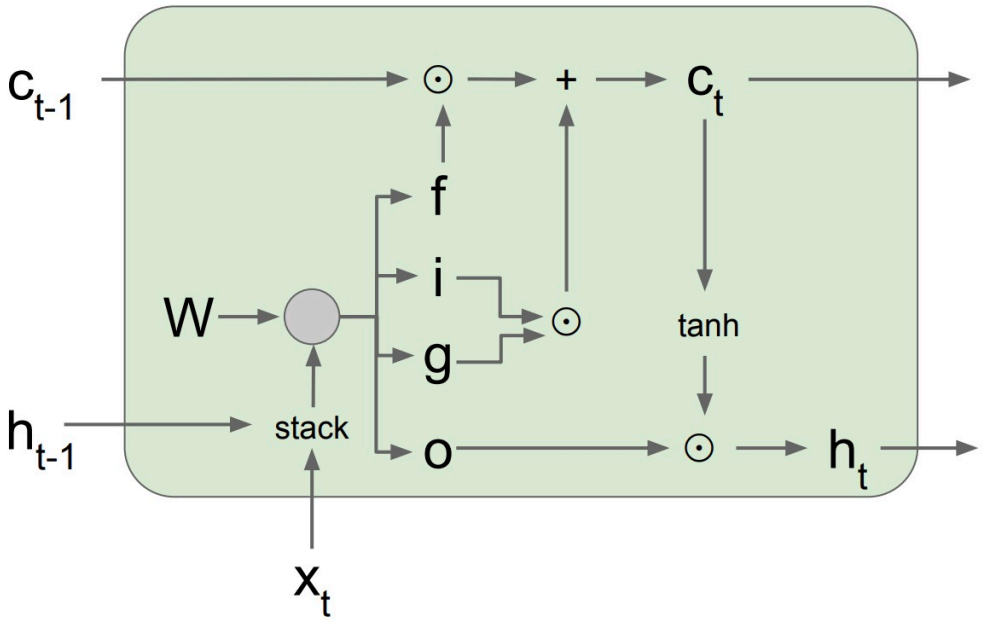


Long Short-Term Memory(LSTM)



$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

Long Short-Term Memory(LSTM)



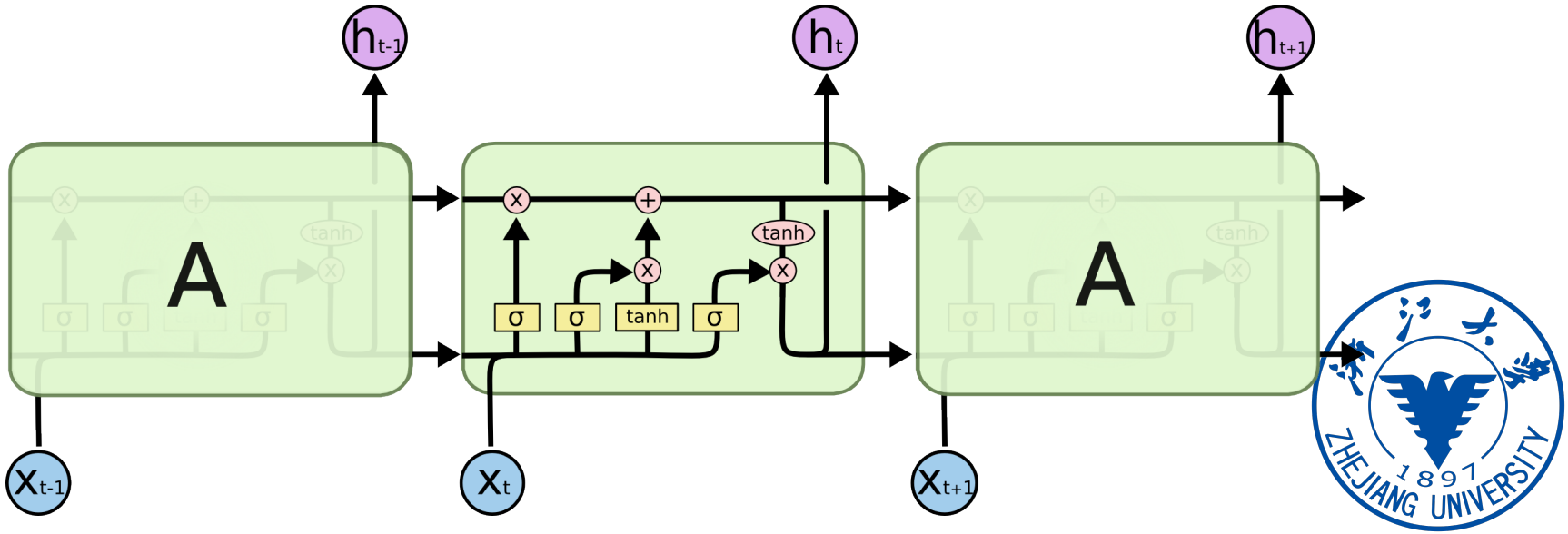
$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(C_t)$$

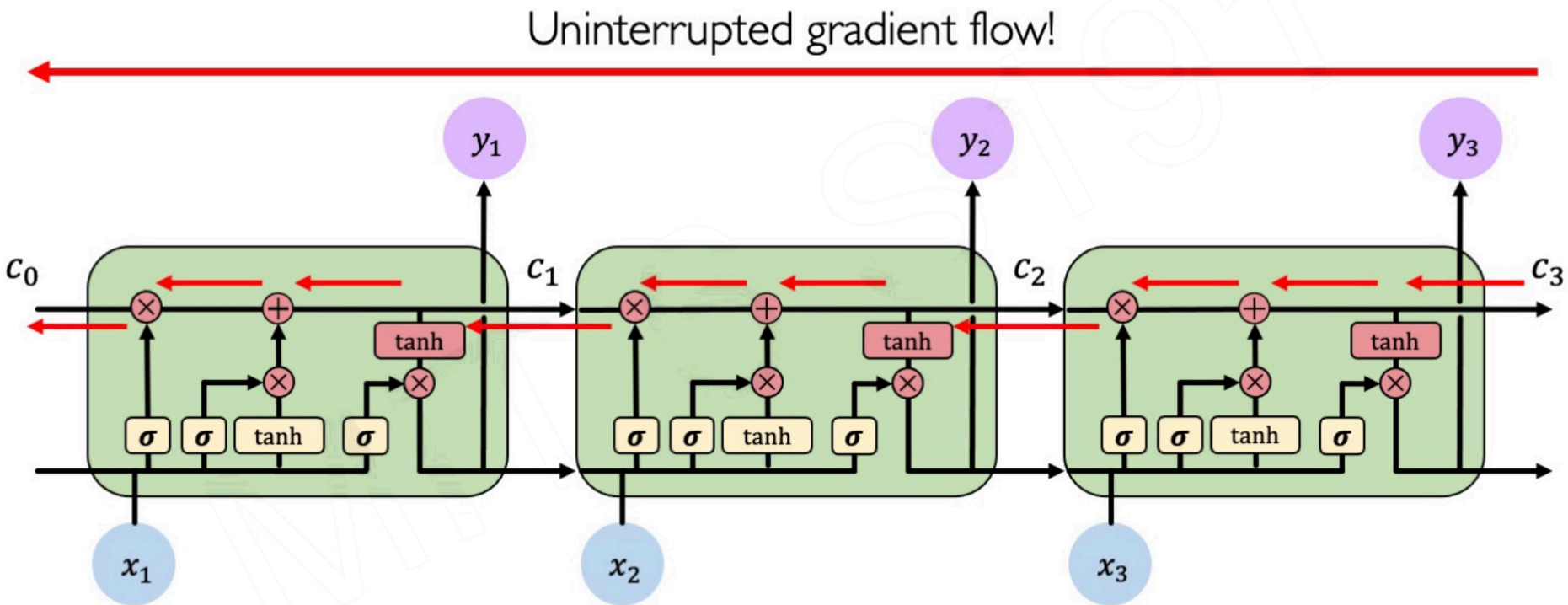
Long Short-Term Memory(LSTM)

- sigmoid 和 tanh 的选择：
- sigmoid：输出 0 到 1 之间的数值，结合按位相乘操作可以起到筛选的功能。
- tanh：输出-1 到 1 之间的数值，可以用来对数据进行标准化，防止过大或者过小。



LSTM 的梯度流

- 在 LSTM 中，存在着一条从 C_t 到 C_{t-1} 再一直到 C_0 的不被影响的梯度流，因此可以缓解梯度消失的问题。之所以说是“缓解”，是因为只有在这一条路上能解决，而其他路上还是存在梯度消失的问题。



LSTM 的梯度流

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(C_t)$$

- 与 RNN 类似，我们现在重点关注 $\frac{\partial C_t}{\partial C_{t-1}}$：

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t + \frac{\partial f_t}{\partial C_{t-1}} \odot C_{t-1} + \frac{\partial i_t}{\partial C_{t-1}} \odot g_t + \frac{\partial g_t}{\partial C_{t-1}} \odot i_t$$

RNN 的梯度流

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t + \frac{\partial f_t}{\partial C_{t-1}} \odot C_{t-1} + \frac{\partial i_t}{\partial C_{t-1}} \odot g_t + \frac{\partial g_t}{\partial C_{t-1}} \odot i_t$$

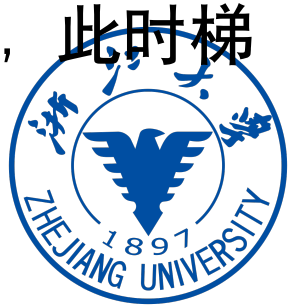
由此可以得到:

$$\begin{aligned}
\frac{\partial C_T}{\partial C_t} &= \frac{\partial C_T}{\partial C_{T-1}} \frac{\partial C_{T-1}}{\partial C_{T-2}} \cdots \frac{\partial C_{t+1}}{\partial C_t} \\
&= \prod_{k=t+1}^{T} \frac{\partial C_k}{\partial C_{k-1}} \\
&= (f_k f_{k+1} \cdots f_T) + \text{others}
\end{aligned}$$

RNN 的梯度流

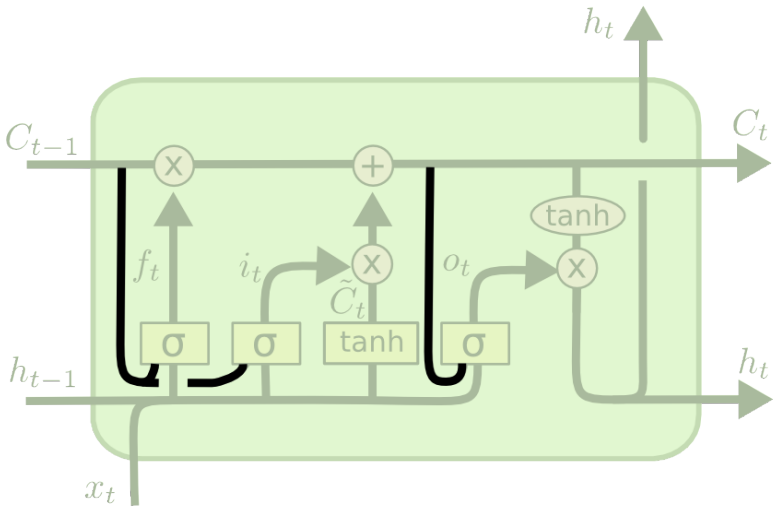
$$\frac{\partial C_T}{\partial C_t} = (f_k f_{k+1} \cdots f_T) + \text{others}$$

- 在 LSTM 中，多次连乘的 f_k 不再是固定的，而是可以学习的参数。因此就解决了 RNN 中梯度消失的问题。
- 当 $f_k = 1$ 时，表示完全保留旧的 Cell 状态，此时梯度可以正常传递。相反，当 $f_k = 0$ 时，表示完全舍弃旧状态，此时梯度流也被切断，无需再向前传播。



Long Short-Term Memory(LSTM)——变体

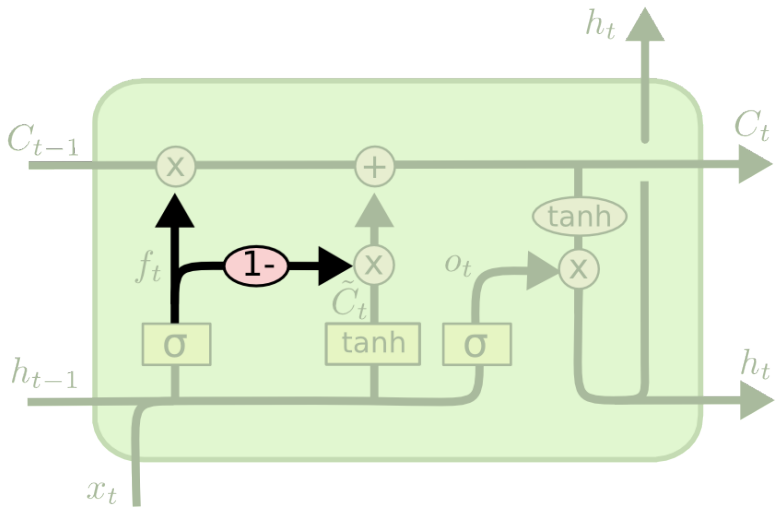
- peephole connection：使 gate 也接受上一步 Cell 状态的输入。



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Long Short-Term Memory(LSTM)——变体

- coupled forget and input gates：**同时确定需要忘记和需要添加的信息。**

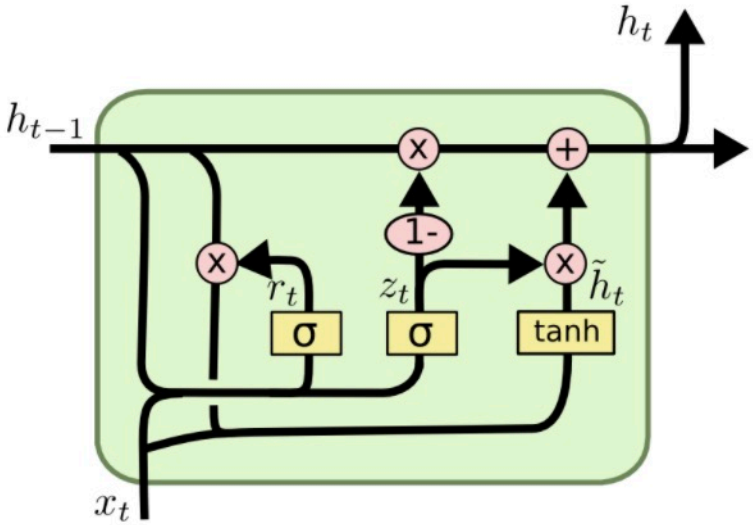


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Gated Recurrent Unit(GRU) [Cho et al., 2014]

GRU 的定义

1. 重置门（reset gate）：控制旧的记忆如何与新的知识相结合。
2. 更新门（update gate）：控制当前记忆如何更新。



$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W_g \cdot [r_t \odot h_{t-1}, x_t])$$
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

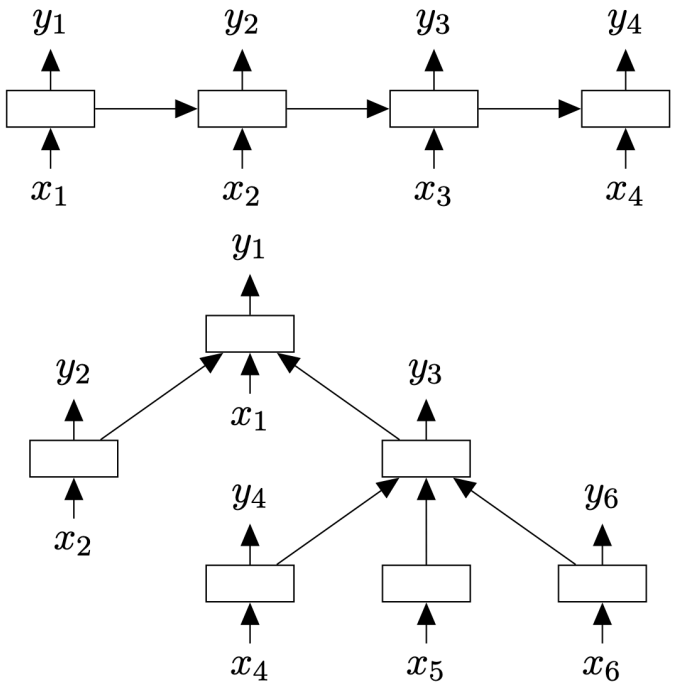
GRU 与 LSTM

- GRU 和 LSTM 的核心思想都是通过 gate 来控制记忆的传递以及与当前输入的结合。
- 两者在很多情况下效果相差无几，但相比之下 GRU 更容易进行训练，能够很大程度上提高训练效率，因此很多时候会更倾向于使用 GRU。

Tree-LSTM [Tai et al., 2015]

研究动机

传统的 LSTM 只能建模链式结构（chain-structured）的依赖关系，不能建模复杂结构的依赖关系。

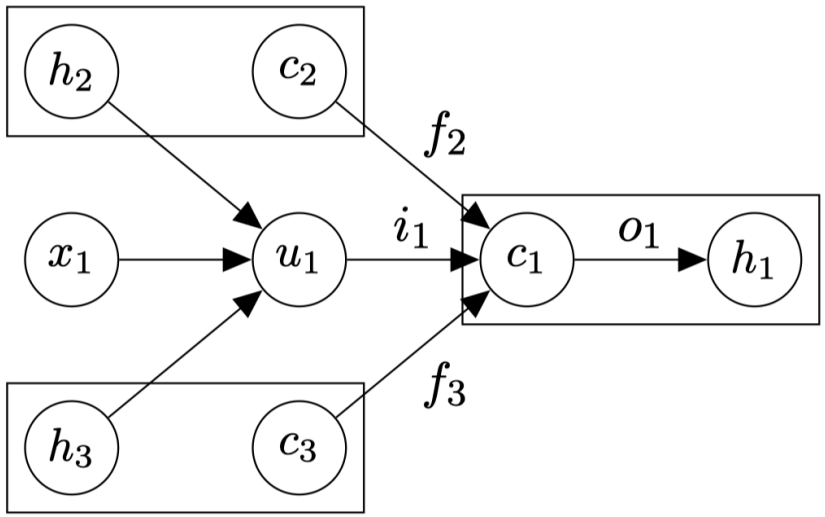


真实世界很多语料存在树形结构

Tree-LSTM

Tree-LSTM 与传统 LSTM 的区别

Tree-LSTM 的 Gating Vectors 和 Memory Cell 需要基于所有的子节点。



Tree-LSTM

Tree-LSTM 模型结构

$$\tilde{h}_j = \sum_{k \in C(j)} h_k$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right)$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right)$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right)$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right)$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k$$

$$h_j = o_j \odot \tanh\left(c_j\right)$$

Tree-LSTM

Tree-LSTM 视角的 GNN

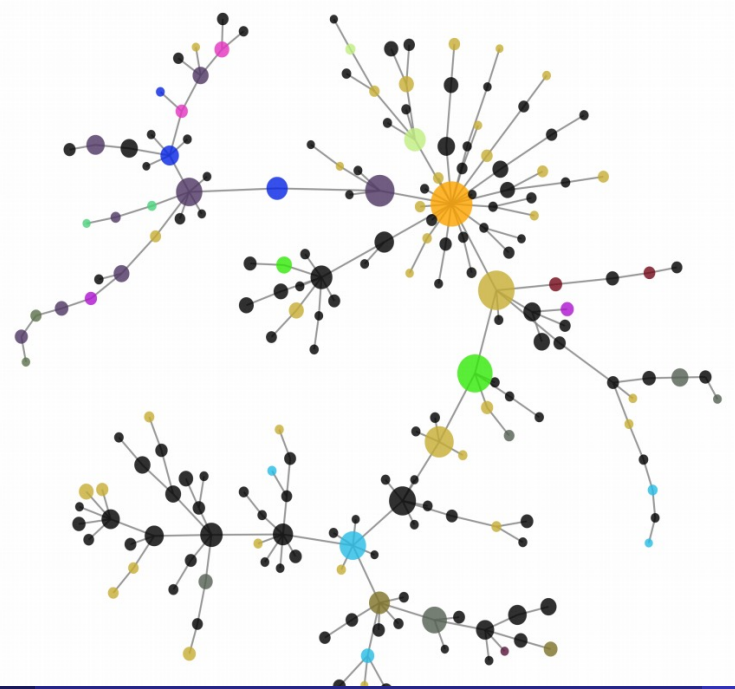
Tree-LSTM 通过融合节点自身特性以及 children 节点的信息更新自身表示，其基本思想与 GNN 一致。

两者的区别是什么？

Graph-Structured LSTM [Zayats and Ostendorf, 2018]

Reddit 受欢迎程度预测

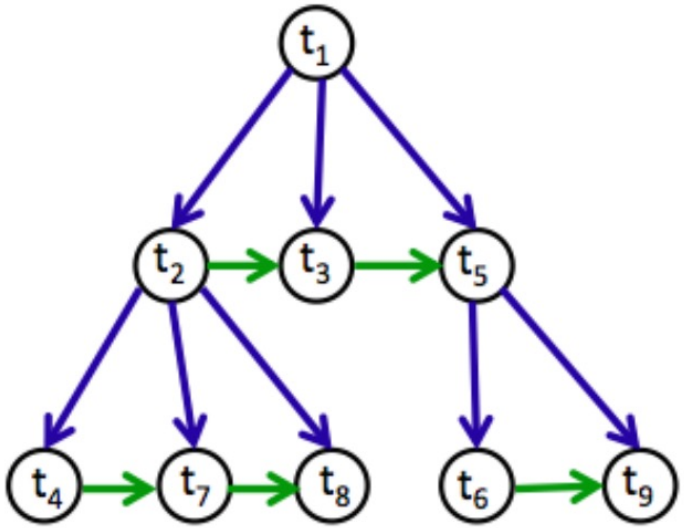
社交网络中的评论是一个天然的树状结构：原始帖子是根节点，而对其的评论相当于是为其添加了一个子节点。同样，对评论的回复也是在添加子节点。



Graph-Structured LSTM

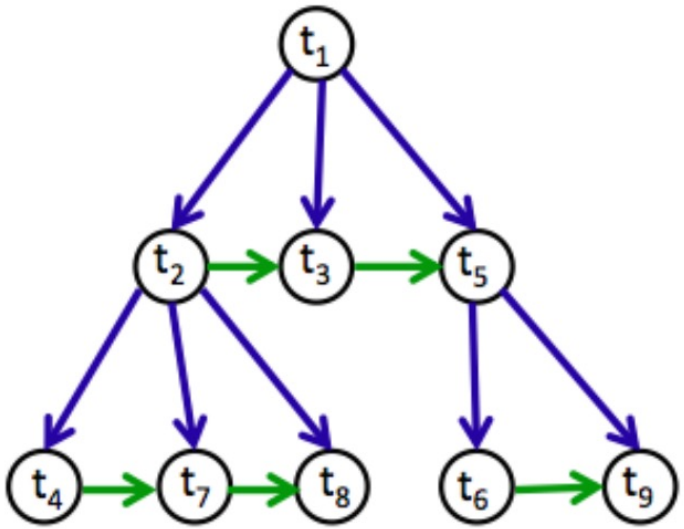
- 评论不仅仅是树状结构，同时，其先后顺序也包含着重要的时序信息。

- 由此，我们可以根据其层次结构和时间结构进行建模。下图中节点的下标表示时间顺序，比如评论 t_2 就是在 t_3 之前发布的。蓝线表示层次传播，绿线表示时间传播。



Graph-Structured LSTM

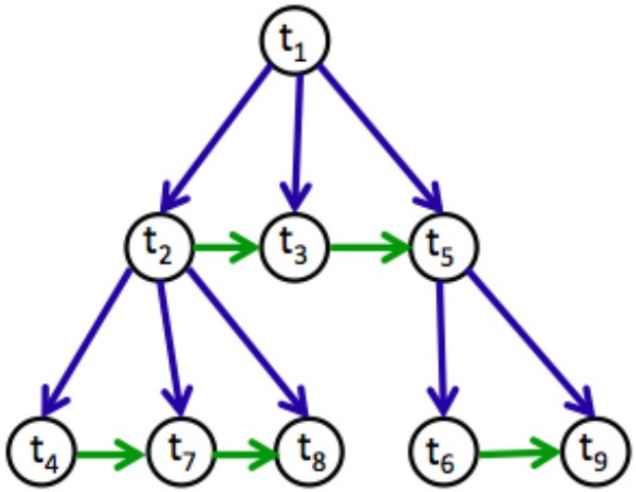
- 每个节点最多收到来自两个入边的信息：
  - 来自父节点的通过层次传播的信息；
  - 来自同级兄节点的通过时间传播的信息。



Graph-Structured LSTM

在开始具体介绍 LSTM 单元前，我们先来了解几个定义：

- $\pi(t)$：t 的父节点。
- $k(t)$：t 的第一个子节点。
- $p(t)$：t 的同级兄节点。
- $s(t)$：t 的同级弟节点。
- 例如：
  - $\pi(t_2) = t_1$，$k(t_2) = t_4$，$p(t_2) = \emptyset$，$s(t_2) = t_3$
  - $\pi(t_3) = t_1$，$k(t_3) = \emptyset$，$p(t_3) = t_2$，$s(t_3) = t_5$



Graph-Structured LSTM

这里的 LSTM 单元包含了四个 gate：

- i：输入门（input gate）
- f：时间遗忘门（temporal forget gate）
- g：层次遗忘门（hierarchichal forget gate）
- o：输出门（output gate）

Graph-Structured LSTM

$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{p(t)} + V_i h_{\pi(t)} + b_i) \\
f_t &= \sigma(W_f x_t + U_f h_{p(t)} + V_f h_{\pi(t)} + b_f) \\
g_t &= \sigma(W_g x_t + U_g h_{p(t)} + V_g h_{\pi(t)} + b_g) \\
o_t &= \sigma(W_o x_t + U_o h_{p(t)} + V_o h_{\pi(t)} + b_o)
\end{aligned}
$$

- 与常规 LSTM 非常相似，区别在于，它是将父节点和同级兄节点的信息作为“记忆”来学习的当前节点信息。
- 当 $p(t) = \emptyset$ 时，$h_{p(t)}$ 将用初始的状态值来替换，后面的 $c_{p(t)}$ 也是相同处理。

Graph-Structured LSTM

- Cell 状态 c_t 的更新以及输出 h_t 的计算如下：

$$\tilde{c}_t = W_c x_t + U_c h_{p(t)} + V_c h_{\pi(t)} + b_c$$
$$c_t = f_t \odot c_{p(t)} + g_t \odot c_{\pi(t)} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot \tanh(c_t)$$

- 我们在前面介绍过双向的 RNN，它可以获取位于序列后方的信息。在这里，双向的 LSTM 也能够提高模型的效果。
- 对于 Tree 结构而言，自底向上的回传可以带来许多有用的信息，比如子节点的数量、子节点的评论受欢迎度等等。这些对于我们预测父节点的受欢迎程度是非常有帮助的。

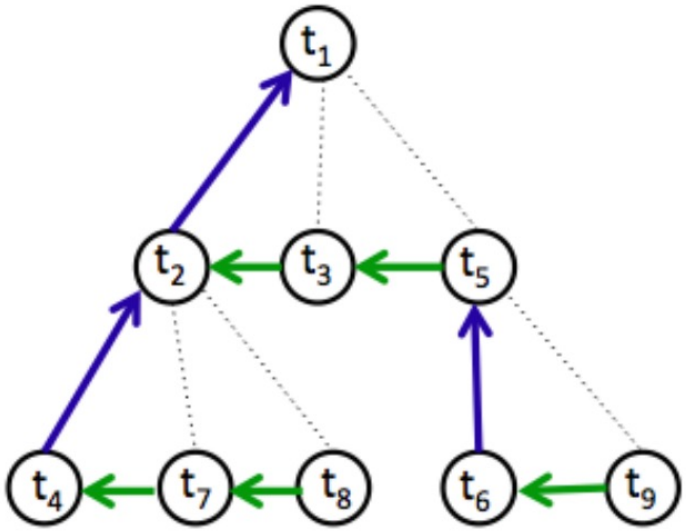
Graph-Structured LSTM

- 我们分别用 $+$ 和 $-$ 来表示正向和反向的 LSTM。因此正向的 LSTM 过程可以写作：

$$
\begin{aligned}
i_t^+ &= \sigma(W_i^+ x_t + U_i^+ h_{p(t)}^+ + V_i^+ h_{\pi(t)}^+ + b_i^+) \\
f_t^+ &= \sigma(W_f^+ x_t + U_f^+ h_{p(t)}^+ + V_f^+ h_{\pi(t)}^+ + b_f^+) \\
g_t^+ &= \sigma(W_g^+ x_t + U_g^+ h_{p(t)}^+ + V_g^+ h_{\pi(t)}^+ + b_g^+) \\
o_t^+ &= \sigma(W_o^+ x_t + U_o^+ h_{p(t)}^+ + V_o^+ h_{\pi(t)}^+ + b_o^+) \\
\tilde{c}_t^+ &= W_c^+ x_t + U_c^+ h_{p(t)}^+ + V_c^+ h_{\pi(t)}^+ + b_c^+ \\
c_t^+ &= f_t^+ \odot c_{p(t)}^+ + g_t^+ \odot c_{\pi(t)}^+ + i_t^+ \odot \tilde{c}_t^+ \\
h_t^+ &= o_t^+ \odot \tanh(c_t^+)
\end{aligned}
$$

Graph-Structured LSTM

- 对于反向的 LSTM 而言，信息传播也是相反的：
  - 层次信息由第一个子节点 $k(t)$ 传播；
  - 时间信息由同级弟节点 $s(t)$ 传播。



Graph-Structured LSTM

- 因此，反向 LSTM 的过程就是将正向中的 $\pi(t)$ 替换为 $k(t)$，$p(t)$ 替换为 $s(t)$，并且学习一套不同的权重矩阵和偏置向量。

$$
\begin{aligned}
i_t^- &= \sigma(W_i^- x_t + U_i^- h_{s(t)}^- + V_i^- h_{k(t)}^- + b_i^-) \\
f_t^- &= \sigma(W_f^- x_t + U_f^- h_{s(t)}^- + V_f^- h_{k(t)}^- + b_f^-) \\
g_t^- &= \sigma(W_g^- x_t + U_g^- h_{s(t)}^- + V_g^- h_{k(t)}^- + b_g^-) \\
o_t^- &= \sigma(W_o^- x_t + U_o^- h_{s(t)}^- + V_o^- h_{k(t)}^- + b_o^-) \\
\tilde{c}_t^- &= W_c^- x_t + U_c^- h_{s(t)}^- + V_c^- h_{k(t)}^- + b_c^- \\
c_t^- &= f_t^- \odot c_{s(t)}^- + g_t^- \odot c_{k(t)}^- + i_t^- \odot \tilde{c}_t^- \\
h_t^- &= o_t^- \odot \tanh(c_t^-)
\end{aligned}
$$

Graph-Structured LSTM

- 在 LSTM 单元的顶端，我们将正向 LSTM 和反向 LSTM 的状态向量连接，通过一个 MLP+softmax 后预测其受欢迎程度（共被分为八个等级。）

$$P(y_t = j|x, h) = \frac{\exp(W_s^j[h_t^+, h_t^-])}{\sum_{k=1}^{8} \exp(W_s^k[h_t^+, h_t^-])}$$

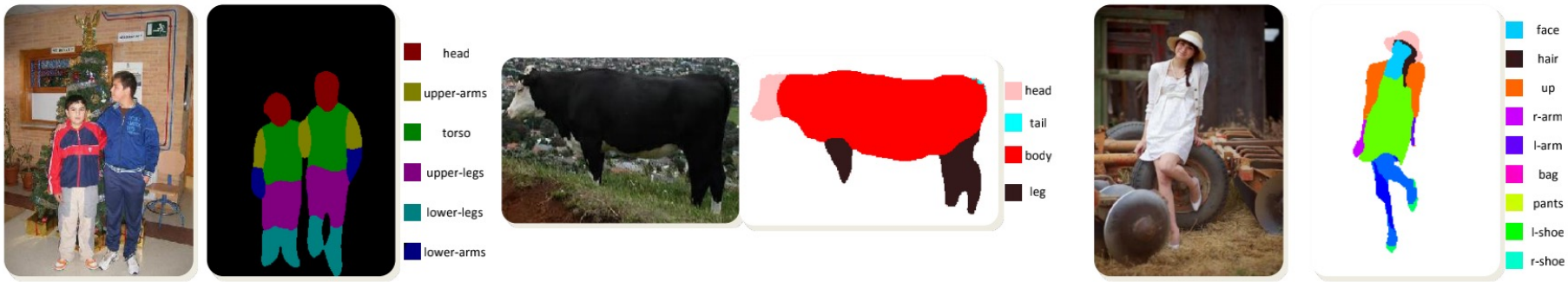
- 目标为最小化与真实 label 之间的 cross-entropy。

Graph LSTM [Liang et al., 2016]

- 以上的方法还是借助了 Tree 结构所带来的层次性，对于一般的无向图，我们该如何构造序列呢？
- Graph LSTM 将传统的 LSTM 模型从顺序和多维数据扩展到一般的图结构数据，并在图像的语义信息解析任务中展示了其优越性。

Graph LSTM

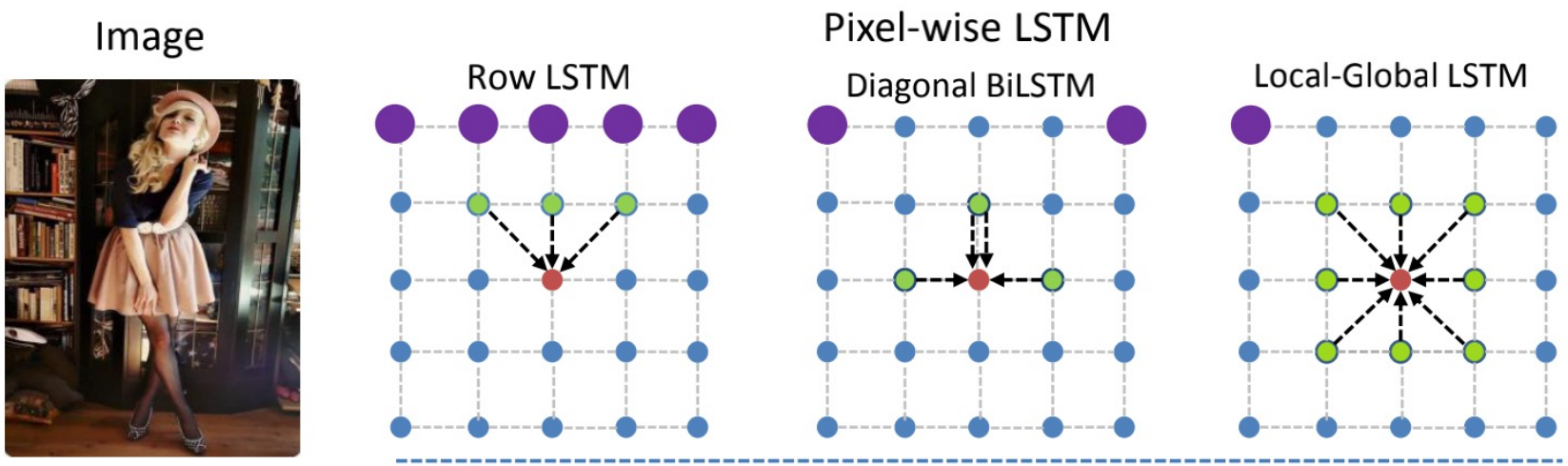
- 语义对象解析：将图像中的对象分割成更细粒度的语义的多个部分，并提供对图像内容的全面理解。



图：语义对象解析

Graph LSTM

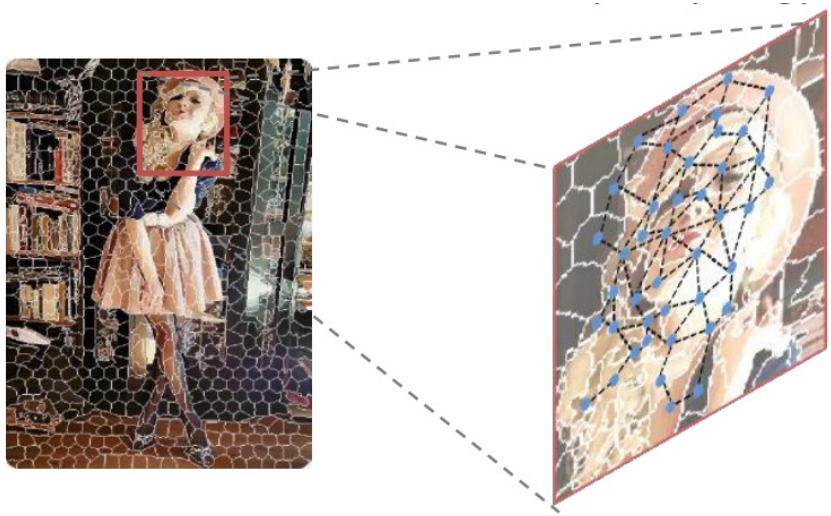
- 对于 image，以往的 LSTM 算法将其均匀地、固定地划分为像素或块，按照预定义的拓扑结构和更新序列将信息广播到固定的局部领域。

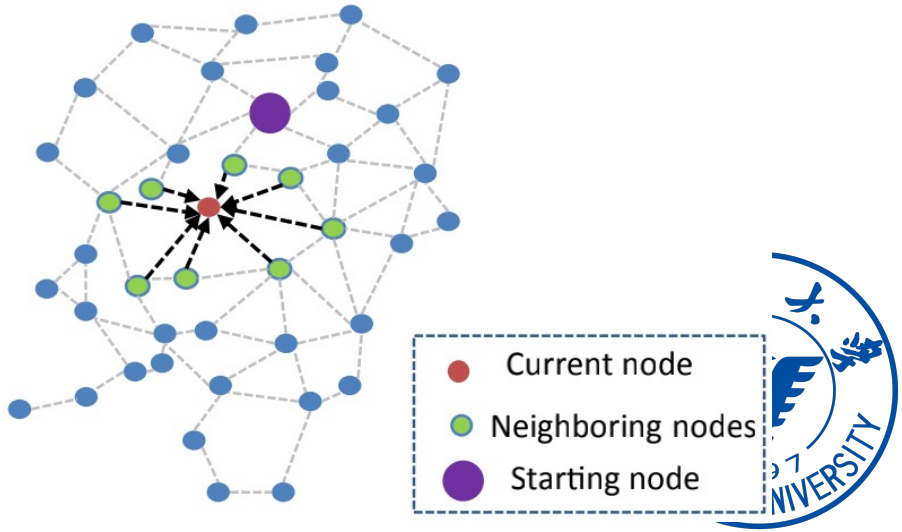


Graph LSTM

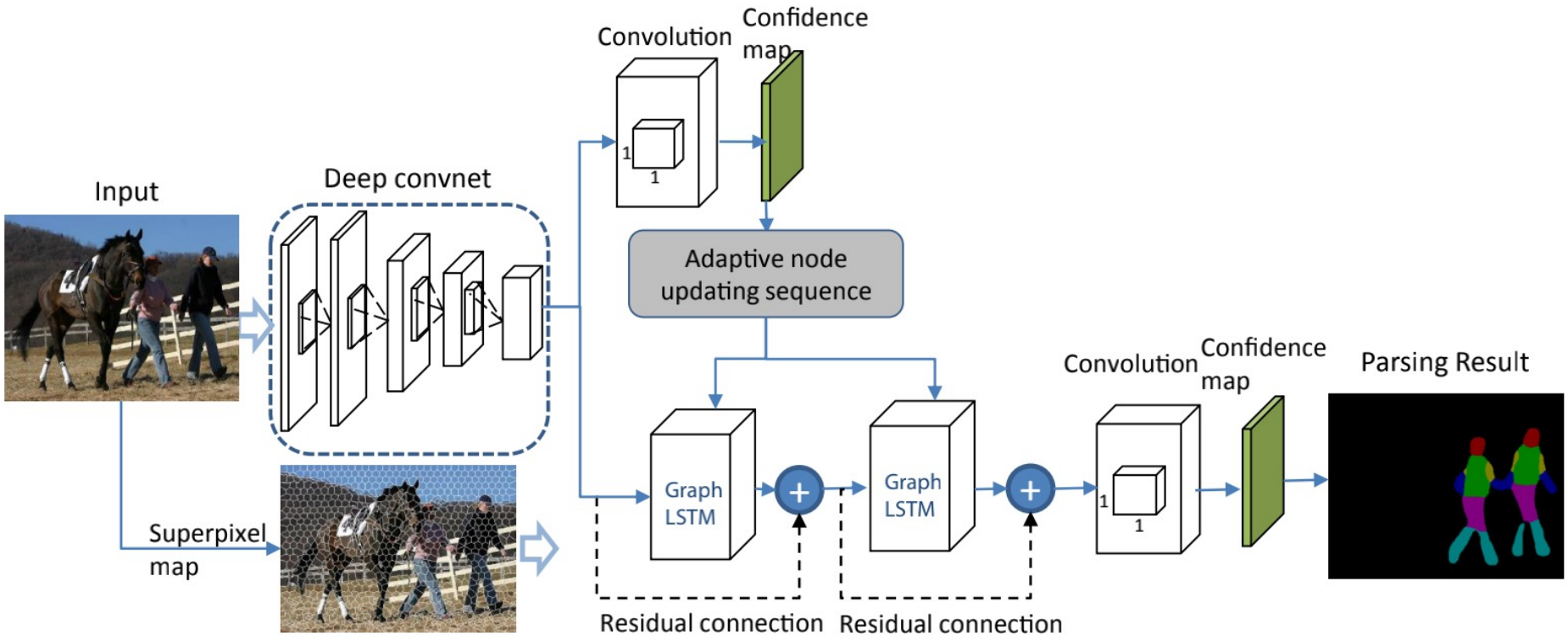
- 而在 Graph LSTM 中，是将每个任意形状的超像素作为语义一致的节点，而它们的空间邻域关系自然地用于构建无向图的边。

- 这样做不仅可以捕获更多图像的自然属性（比如局部的边界和语义上一致的像素组），而且还可以有效地减少冗余的计算开销。





Graph LSTM



Graph LSTM 方法框架

Graph LSTM

Confidence Driven Sequence

与 Graph-Structured LSTM 相同，Graph 中的每个节点都会对应一个 LSTM 单元。区别在于前者的邻居节点和更新顺序都是事先规定好，但 Graph LSTM 的邻居是通过构图生成的，而更新顺序则是由 confidence-driven 策略构造的。

- 给定卷积特征，1×1 卷积滤波器可以生成关于每个语义标签的初始置信度图。
- 对于每个超像素，对其包含的像素的置信度求均值来计算每个标签的置信度，并且将具有最高置信度的标签分配给该超像素。
- 节点更新的顺序可以通过对超像素的置信度进行排序来确定。

Graph LSTM

- 为简便计算，我们将邻居节点的 hidden 状态合成一个平均的值。在我们采用的更新策略中，当 Graph LSTM 层操作到一个具体节点时，它的邻居节点中可能有一些已经更新过了，而还有一些没有。

- 因此，我们需要一个 flag q_j 来标识邻居节点 v_j 是否已经更新。在此情况下，v_i 的邻居平均 hidden 状态 $\bar{h}_{i,t}$ 为：

$$\bar{h}_{i,t} = \frac{\sum_{j \in \mathcal{N}_{\mathcal{G}}(i)} (\mathbf{1}(q_j = 1) h_{j,t+1} + \mathbf{1}(q_j = 0) h_{j,t})}{|\mathcal{N}_{\mathcal{G}}(i)|}$$

Graph LSTM

- 与传统的 LSTM 不同，Graph LSTM 还为每个邻居节点设置了自适应的 forget gate。
- 由此，不同邻居节点会对当前节点的状态产生不同的影响。
- 是不是很像之前讲过的 GAT ？

Graph LSTM

Graph LSTM 单元包含五个 gate：

- g^u：输入门（input gate）
- g^f：遗忘门（forget gate）
- $\bar{g}^f$：自适应遗忘门 (adaptive forget gate)
- g^c：输入状态（gate gate）
- g^o：输出门（output gate）

Graph LSTM

- 以上五个 gate 的公式如下：

$$
\begin{aligned}
g_i^u &= \sigma(W^u f_{i,t+1} + U^u h_{i,t} + U^{un} \bar{h}_{i,t} + b^u) \\
\bar{g}_{ij}^f &= \sigma(W^f f_{i,t+1} + W^{fn} h_{j,t} + b^f) \\
g_i^f &= \sigma(W^f f_{i,t+1} + U^f h_{i,t} + b^f) \\
g_i^o &= \sigma(W^o f_{i,t+1} + U^o h_{i,t} + b^o) \\
g_i^c &= \tanh(W^c f_{i,t+1} + U^c h_{i,t} + U^{cn} \bar{h}_{i,t} + b^c)
\end{aligned}
$$

Graph LSTM

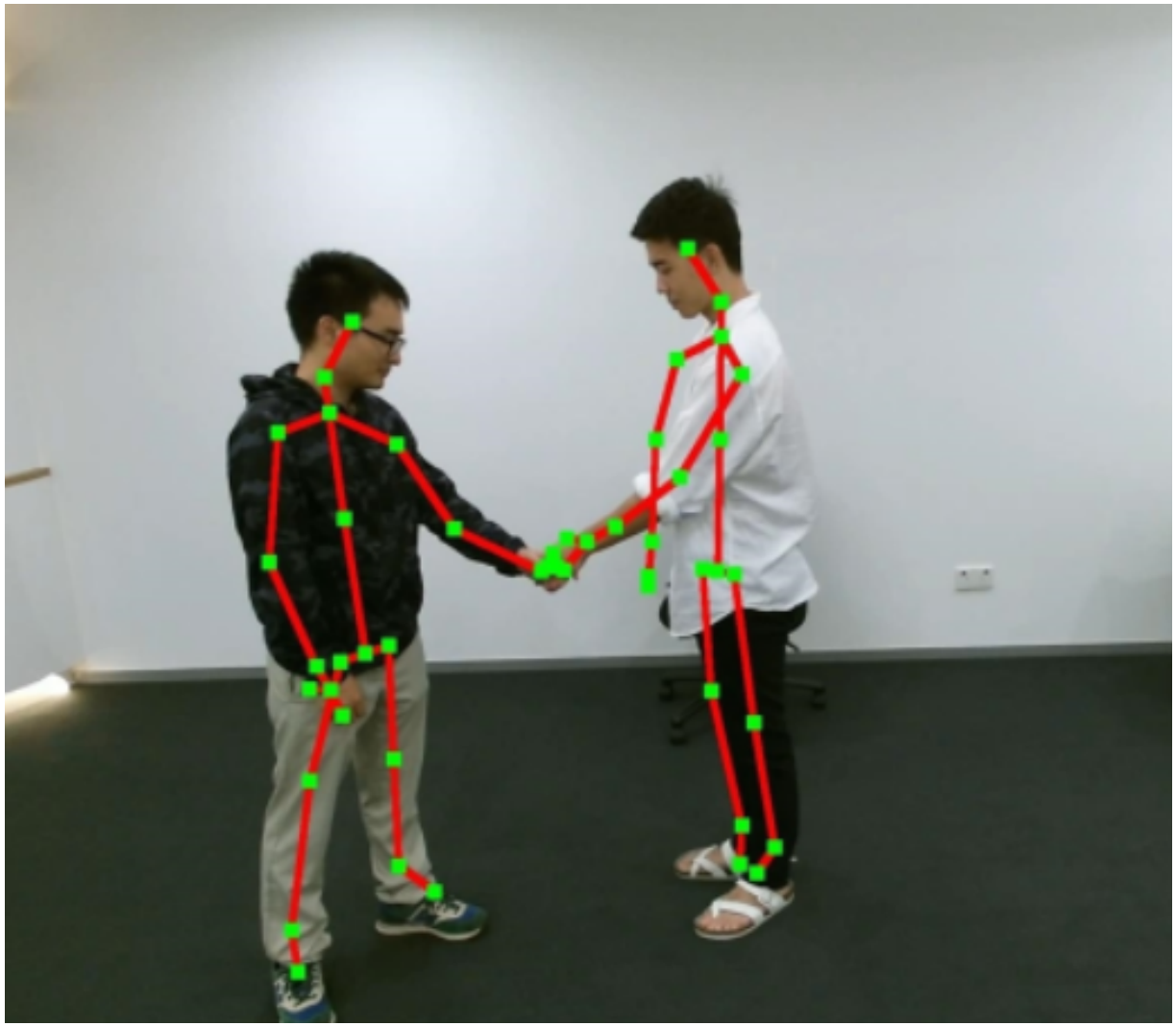
- 最终，memory 状态和 hidden 状态的更新如下：

$$m_{i,t+1} = \frac{\sum_{j \in \mathcal{N}_{\mathcal{G}}(i)} (\mathbf{1}(q_j = 1) \bar{g}_{ij}^f \odot m_{j,t+1} + \mathbf{1}(q_j = 0) \bar{g}_{ij}^f \odot m_{j,t})}{|\mathcal{N}_{\mathcal{G}}(i)|} + g_i^f \odot m_{i,t} + g_i^u \odot g_i^c$$

$$h_{i,t+1} = \tanh(g_i^o \odot m_{i,t+1})$$

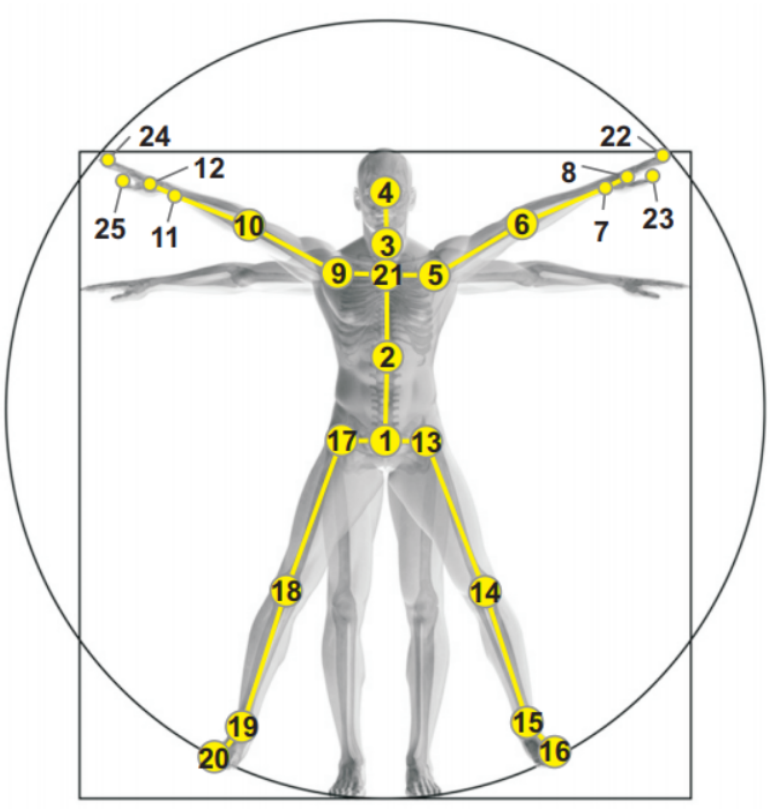
Part-Aware LSTM [Shahroudy et al., 2016]

- 人体动作可用空间时序图表示。

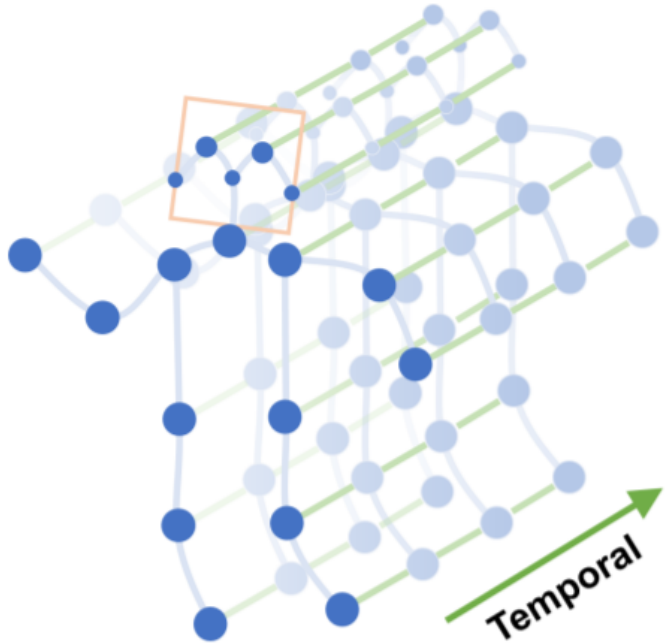


Part-Aware LSTM

- LSTM 能否捕捉动作中的时序信息？



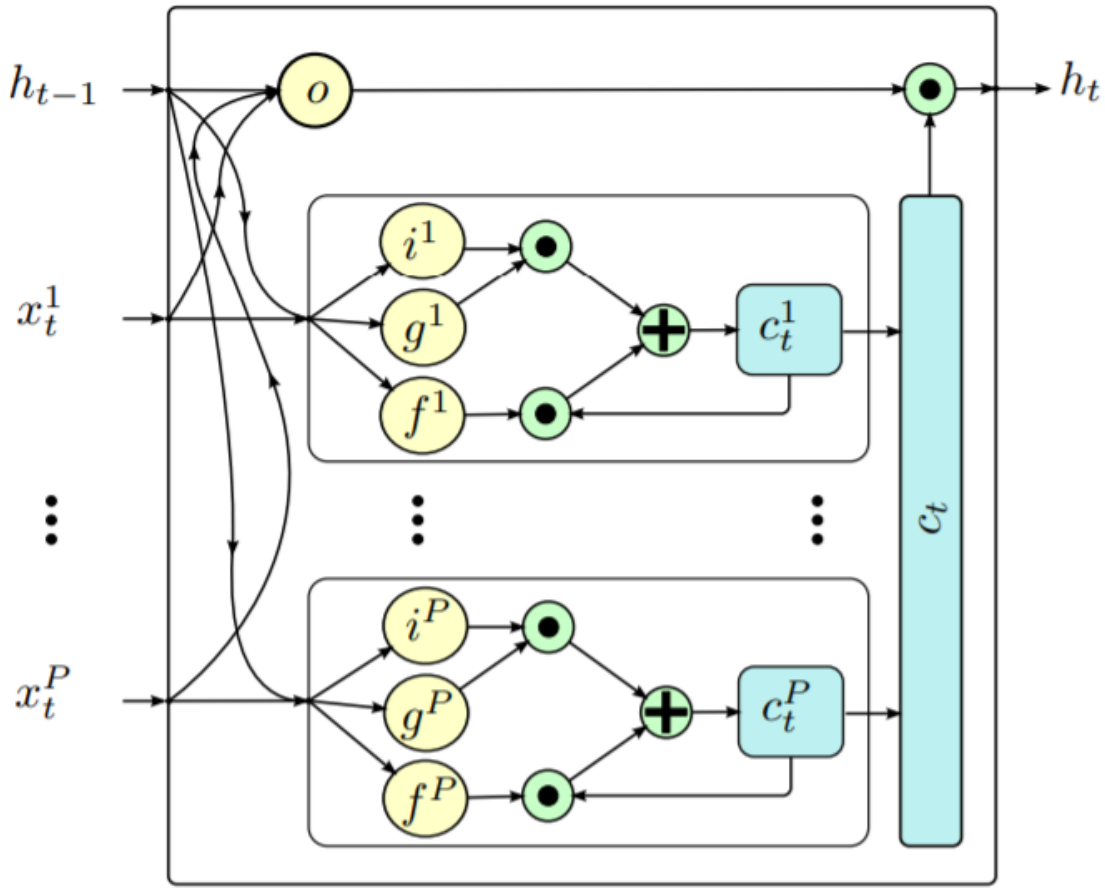
关节点划分



时序骨架图

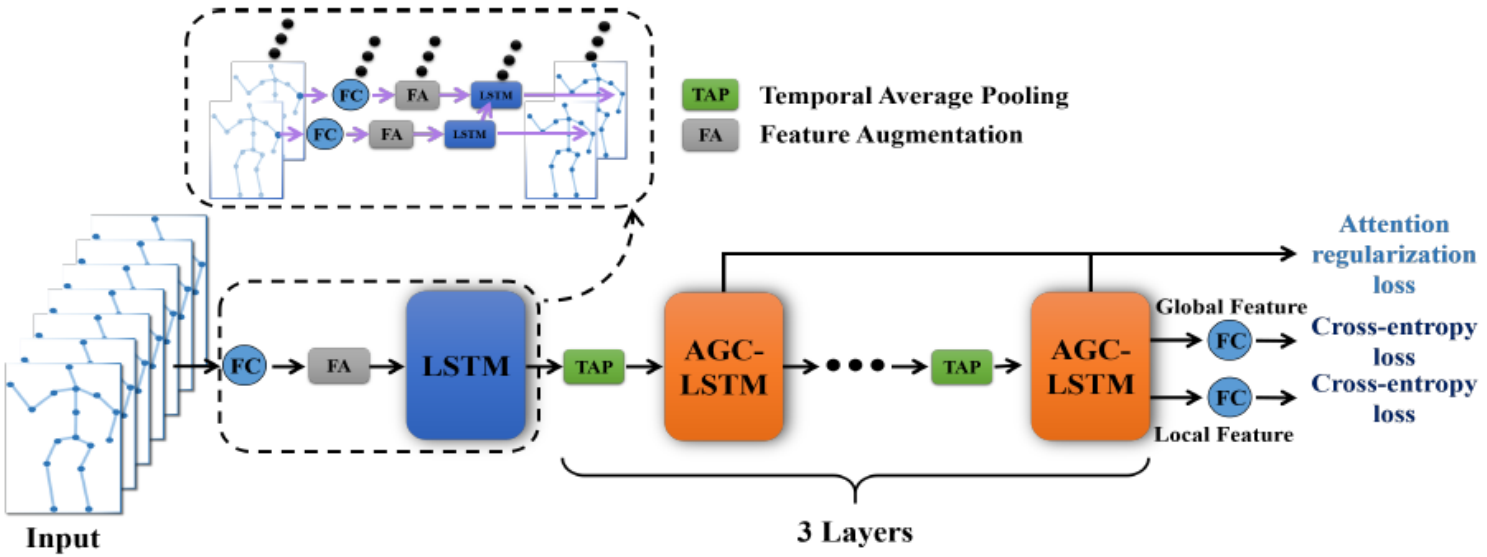
Part-Aware LSTM

- 基于身体部位的多门控单元
- 基于身体部位的多记忆单元



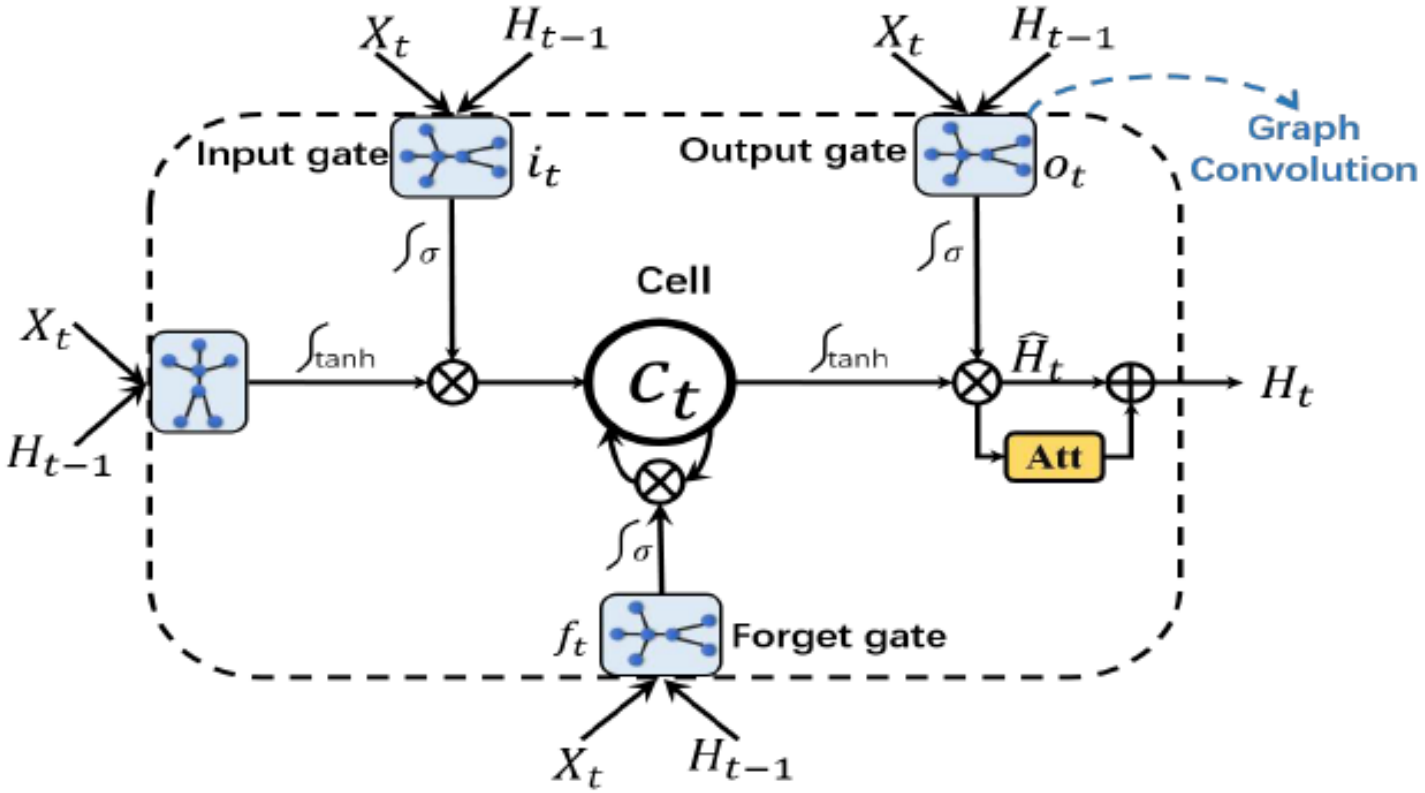
AGC-LSTM [Si et al., 2019]

- 探索人体动作在空间和时间域的共现关系，关注不同关节点在不同时间戳的相关性。



AGC-LSTM

- 核心模块：AGC-LSTM，融合人体动作时空特征。



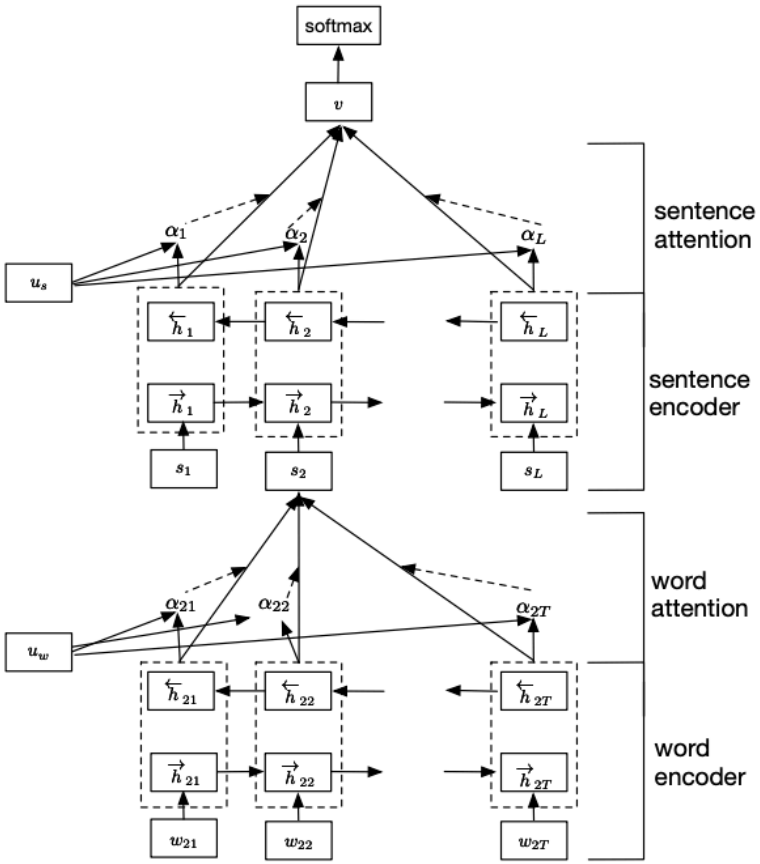
- 提取人体关节空间特征：GCN
- 挖掘人体动作时序关系：LSTM

AGC-LSTM

- $\mathcal{G}$ 表示图卷积操作算子，f_{att} 表示注意力机制。

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}_{xi} * \mathcal{G}\mathbf{X}_t + \mathbf{W}_{hi} * \mathcal{G}\mathbf{H}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}_{xf} * \mathcal{G}\mathbf{X}_t + \mathbf{W}_{hf} * \mathcal{G}\mathbf{H}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_{xo} * \mathcal{G}\mathbf{X}_t + \mathbf{W}_{ho} * \mathcal{G}\mathbf{H}_{t-1} + \mathbf{b}_o\right) \\
\mathbf{u}_t &= \tanh\left(\mathbf{W}_{xc} * \mathcal{G}\mathbf{X}_t + \mathbf{W}_{hc} * \mathcal{G}\mathbf{H}_{t-1} + \mathbf{b}_c\right) \\
\mathbf{C}_t &= \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\widehat{\mathbf{H}}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{C}_t\right) \\
\mathbf{H}_t &= f_{\text{att}}\left(\widehat{\mathbf{H}}_t\right) + \widehat{\mathbf{H}}_t
\end{aligned}
\tag{1}
$$

HAN [Yang et al., 2016]



$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right)$$

$$\tilde{h}_t = \tanh\left(W_h x_t + r_t \odot \left(U_h h_{t-1}\right) + b_h\right)$$

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right)$$

HAN

Word Encoder:

$$
\begin{aligned}
x_{it} &= W_e w_{it}, t \in [1, T] \\
\vec{h}_{it} &= \overrightarrow{\mathrm{GRU}}\left(x_{it}\right), t \in [1, T] \\
\overleftarrow{h}_{it} &= \overleftarrow{\mathrm{GRU}}\left(x_{it}\right), t \in [T, 1]
\end{aligned}
$$

Word Attention:

$$
\begin{aligned}
u_{it} &= \tanh\left(W_w h_{it} + b_w\right) \\
\alpha_{it} &= \frac{\exp\left(u_{it}^{\top} u_w\right)}{\sum_t \exp\left(u_{it}^{\top} u_w\right)} \\
s_i &= \sum_t \alpha_{it} h_{it}
\end{aligned}
$$

HAN

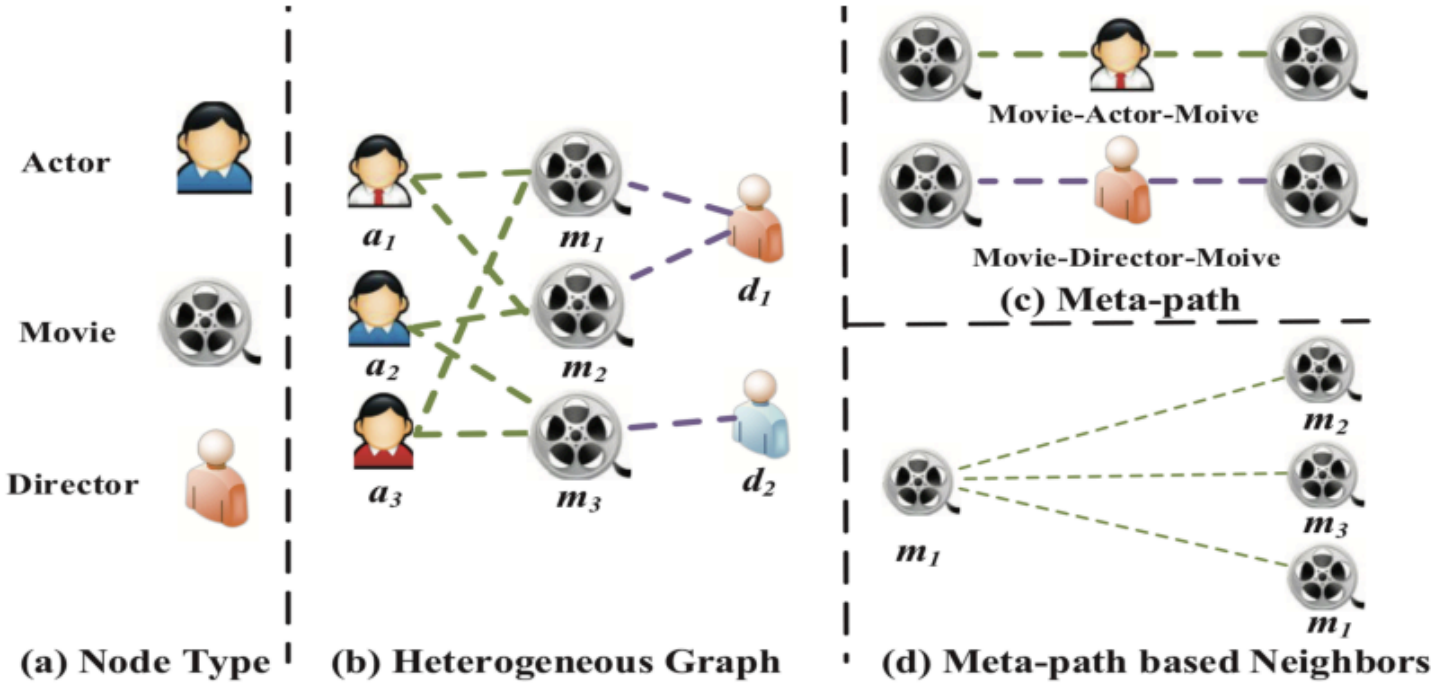
Sentence Encoder:

$$\vec{h}_i = \overrightarrow{\mathrm{GRU}}\left(s_i\right), i \in [1, L]$$
$$\overleftarrow{h}_i = \overleftarrow{\mathrm{GRU}}\left(s_i\right), t \in [L, 1]$$

Sentence Attention:

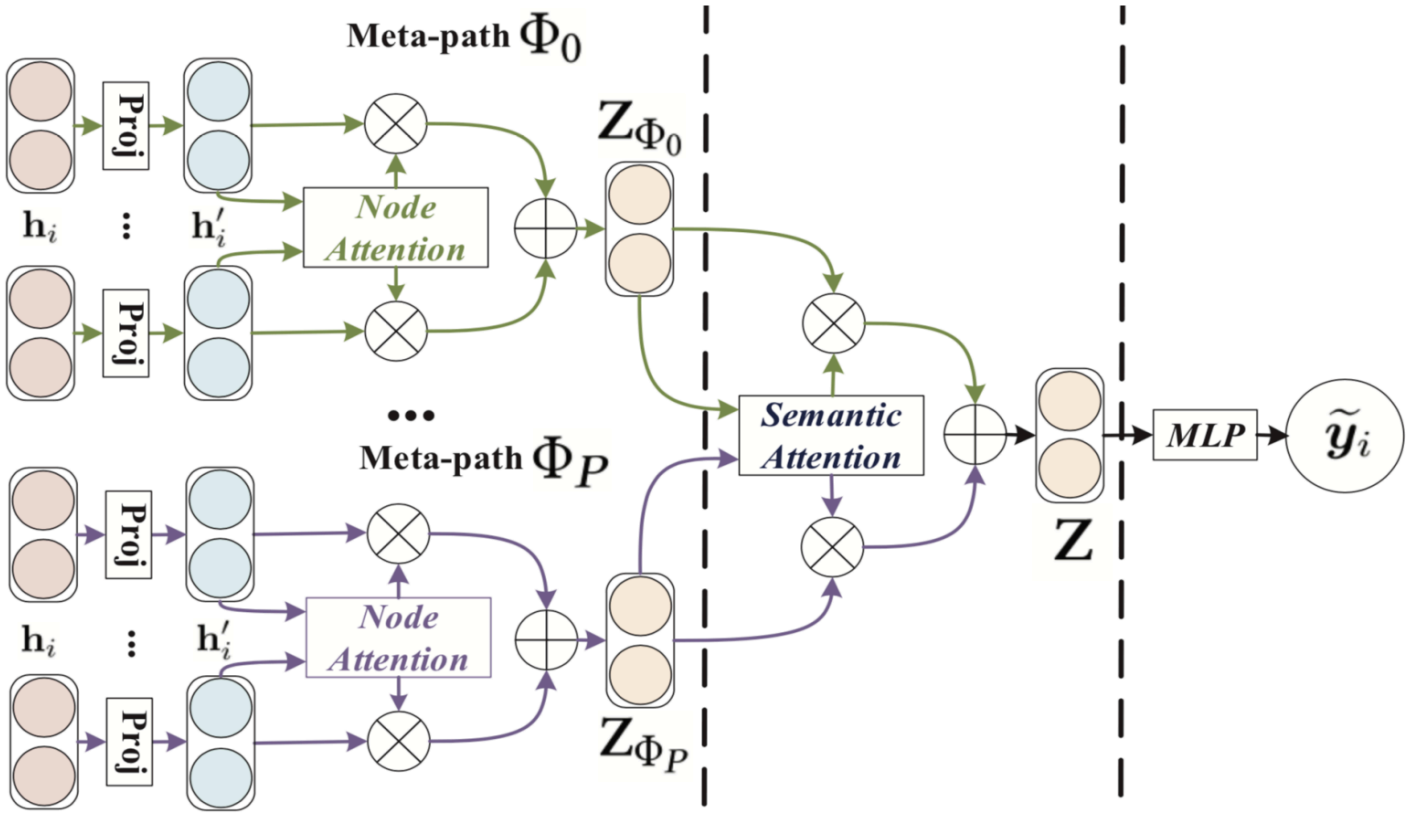
$$u_i = \tanh\left(W_s h_i + b_s\right)$$
$$\alpha_i = \frac{\exp\left(u_i^\top u_s\right)}{\sum_i \exp\left(u_i^\top u_s\right)}$$
$$v = \sum_i \alpha_i h_i$$

HAN


Actor
Movie
Director
a_1
a_2
a_3
m_1
m_2
m_3
d_1
d_2
Movie-Actor-Moive
Movie-Director-Moive
(c) Meta-path
m_2
m_3
m_1
m_1
(a) Node Type
(b) Heterogeneous Graph
(d) Meta-path based Neighbors

异构信息网络

HAN


Meta-path Φ_0
Proj
Proj
$\mathbf{h}_i$
$\mathbf{h}'_i$
Node Attention
Z_{Φ_0}
Meta-path Φ_P
Proj
Proj
$\mathbf{h}_i$
$\mathbf{h}'_i$
Node Attention
Z_{Φ_P}
Semantic Attention
Z
MLP
$\tilde{y}_i$
(a) Node-Level Attention
(b) Semantic-Level Attention
(c) Prediction

HAN


ZHEJIANG UNIVERSITY
1897

HAN

Node Encoder:

$$\mathbf{z}_i^{\Phi} = \sigma\left(\sum_{j \in \mathcal{N}_i^{\Phi}} \alpha_{ij}^{\Phi} \cdot \mathbf{h}_j'\right)$$

$$\mathbf{z}_i^{\Phi} = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i^{\Phi}} \alpha_{ij}^{\Phi} \cdot \mathbf{h}_j'\right)$$

Node Attention:

$$e_{ij}^{\Phi} = att_{\text{node}}\left(\mathbf{h}_i', \mathbf{h}_j'; \Phi\right)$$

$$\alpha_{ij}^{\Phi} = \text{softmax}_j\left(e_{ij}^{\Phi}\right) = \frac{\exp\left(\sigma\left(\mathbf{a}_{\Phi}^{\mathrm{T}} \cdot \left[\mathbf{h}_i' \| \mathbf{h}_j'\right]\right)\right)}{\sum_{k \in \mathcal{N}_i^{\Phi}} \exp\left(\sigma\left(\mathbf{a}_{\Phi}^{\mathrm{T}} \cdot \left[\mathbf{h}_i' \| \mathbf{h}_k'\right]\right)\right)}$$

HAN

Semantic Attention:

$$w_{\Phi_p} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \mathbf{q}^{\mathrm{T}} \cdot \tanh\left(\mathbf{W} \cdot \mathbf{z}_i^{\Phi_p} + \mathbf{b}\right)$$

$$\beta_{\Phi_p} = \frac{\exp\left(w_{\Phi_p}\right)}{\sum_{p=1}^{P} \exp\left(w_{\Phi_p}\right)}$$

$$\mathbf{Z} = \sum_{p=1}^{P} \beta_{\Phi_p} \cdot \mathbf{Z}_{\Phi_p}$$

References I

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014).
Learning phrase representations using rnn encoder-decoder for statistical machine translation.
arXiv preprint arXiv:1406.1078.

Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural computation, 9(8):1735–1780.

Liang, X., Shen, X., Feng, J., Lin, L., and Yan, S. (2016).
Semantic object parsing with graph lstm.
In *European Conference on Computer Vision*, pages 125–143. Springer.

References II


Shahroudy, A., Liu, J., Ng, T.-T., and Wang, G. (2016).
Ntu rgb+ d: A large scale dataset for 3d human activity analysis.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1010–1019.


Si, C., Chen, W., Wang, W., Wang, L., and Tan, T. (2019).
An attention enhanced graph convolutional lstm network for skeleton-based action recognition.
In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1227–1236.


Tai, K. S., Socher, R., and Manning, C. D. (2015).
Improved semantic representations from tree-structured long short-term memory networks.
arXiv preprint arXiv:1503.00075.

References III

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016).
Hierarchical attention networks for document classification.
In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Zayats, V. and Ostendorf, M. (2018).
Conversation modeling on reddit using a graph-structured lstm.
Transactions of the Association for Computational Linguistics, 6:121–132.