Lecture Notes: Part V

Language Models, RNN, GRU and LSTM 2

CS224n 是顶级院校斯坦福出品的深度学习与自然语言处理方向专业课程，核心内容覆盖 RNN、LSTM、CNN、transformer、bert、问答、摘要、文本生成、语言模型、阅读理解等前沿内容。

笔记核心词：

Language Models. RNN. Bi-directional RNN. Deep RNN. GRU. LSTM.

课程**全部资料和信息**已整理发布，扫描下方**任意**二维码，均可获取！！

微信公众号 · 全套资料

回复 CS224n

底部菜单栏

Bilibili · 课程视频

视频简介

置顶评论

GitHub · 项目代码

阅读 ReadMe

点击超链接

1. Language Models

Notes info.	
课件/Slides	Lecture 6, P2
视频/Video	Lecture 6, 01:28
GitHub · 代码	实时在线查阅文档
Bilibili · 视频	中英字幕课程视频
Stanford University X ShowMeAI	

1.1 Introduction

语言模型计算特定序列中多个单词的出现概率。一个 m 个单词的序列 $\{w_1, ..., w_m\}$ 的概率定义为 $P(w_1, ..., w_m)$。单词 w_i 前有一定数量的单词，其特性会根据它在文档中的位置而改变，$P(w_1, ..., w_m)$ 一般只考虑前 n 个单词而不是考虑全部之前的单词。

$$P(w_1, ..., w_m) = \prod_{i=1}^{i=m} P(w_i|w_1, ..., w_{i-1}) \approx \prod_{i=1}^{i=m} P(w_m|w_{i-n}, ..., w_{i-1})$$

上面的公式在语音识别和机器翻译系统对判断一个词序列是否为一个输入句子的准确翻译起到了重要的作用。在现有的机器翻译系统中，对每个短语/句子翻译，系统生成一些候选的词序列（例如，{I have,I has,I had,me have,me had}），并对其评分以确定最可能的翻译序列。

在机器翻译中，对一个输入短语，通过评判每个候选输出词序列的得分的高低，来选出最好的词顺序。为此，模型可以在不同的单词排序或单词选择之间进行选择。它将通过一个概率函数运行所有单词序列候选项，并为每个候选项分配一个分数，从而实现这一目标。最高得分的序列就是翻译结果。例如，相比 small is the cat，翻译系统会给 the cat is small 更高的得分；相比 walking house after school，翻译系统会给 walking home after school 更高的得分。

1.2 Notes info.	
课件/Slides	Lecture 6, P7
视频/Video	Lecture 6, 03:45
GitHub · 代码	实时在线查阅文档
Bilibili · 视频	中英字幕课程视频
Stanford University X ShowMeAI	

1.2 n-gram Language Models

为了计算这些概率，每个 n-gram 的计数将与每个单词的频率进行比较，这个称为 n-gram 语言模型。例如，如果选择 bi-gram 模型，每一个 bi-gram 的频率，通过将单词与其前一个单词相结合进行计算，然后除以对应的 uni-gram 的频率。下面的两个公式展示了 bi-gram 模型和 tri-gram 模型的区别。

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

$$p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

上式 tri-gram 模型的关系主要是基于一个固定的上下文窗口(即前 n 个单词)预测下一个单词。一般 n 的取值为多大才好呢？在某些情况下，前面的连续的 n 个单词的窗口可能不足以捕获足够的上下文信息。例如，考虑句子（类似完形填空，预测下一个最可能的单词）

“Asthe proctor started the clock, the students opened their ”。如果窗口只是基于前面的三个单词“the students opened their”，那么急于这些语料计算的下划线中最有可能出现的单词就是为“books”——但是如果 n 足够大，能包括全部的上下文，那么下划线中最优可能出现的单词会是“exam”。

这就引出了 n-gram 语言模型的两个主要问题：稀疏性和存储。

1） Sparsity problems with n-gram Language models

n-gram 语言模型的问题源于两个问题。

首先，注意公式中的分子。如果 $w_1, w_2 . w_3$ 在语料中从未出现过，那么 w_3 的概率就是 0。为了解决这个问题，在每个单词计数后面加上一个很小的 δ，这就是平滑操作。

然后，考虑公式中的分母。 如果 w_1, w_2 在语料中从未出现过，那么 w_3 的概率将会无法计算。为了解决这个问题，这里可以只是单独考虑 w_2，这就是“backoff”操作。

增加 n 会让稀疏问题更加严重，所以一般 $n \leq 5$ 。

2） Storage problems with n-gram Language models

我们知道需要存储在语料库中看到的所有 n-gram 的统计数。随着 n 的增加(或语料库大小的增加)，模型的大小也会增加。

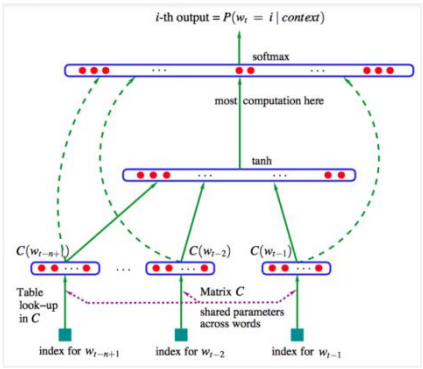
1.3 Window-based Neural Language Model

Bengio 的论文《A Neural Probabilistic Language Model》中首次解决了上面所说的“维度灾难”，这篇论文提出一个自然语言处理的大规模的深度学习模型，这个模型能够通过学习单词的分布式表示，以及用这些表示来表示单词的概率函数。右图展示了对应的神经网络结构，在这个模型中，输入向量在隐藏层和输出层中都被使用。

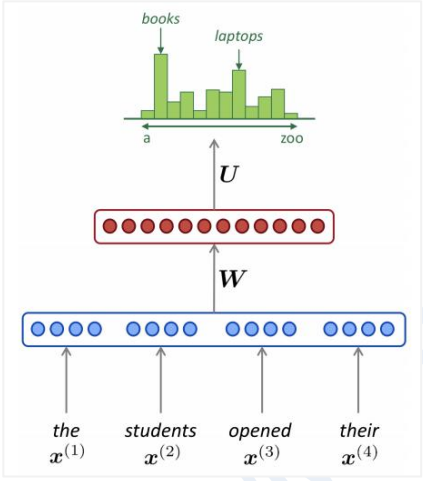
1.3 Notes info.

课件/Slides	Lecture 6, P18
视频/Video	Lecture 6, 15:45
GitHub · 代码	实时在线查阅文档
Bilibili · 视频	中英字幕课程视频

Stanford University X ShowMeAI



❐ Figure 1: The first deep neural network architecture model for NLP presented by Bengio et al. 【图 1:Bengio 等人提出的 NLP 的第一个深层神经网络结构模型．】



❐ ↑ Figure 2: A simplified representation of ↑ Figure 1. 【图 2：图 1 的简化表示．】

下面公式展示了由标准 tanh 函数（即隐藏层）组成的 softmax 函数的参数以及线性函数 $W^{(3)}x + b^{(3)}$，捕获所有前面 n 个输入词向量。

$$\hat{y} = \text{softmax}\left(W^{(2)}\tanh\left(W^{(1)}x + b^{(1)}\right) + W^{(3)}x + b^{(3)}\right)$$

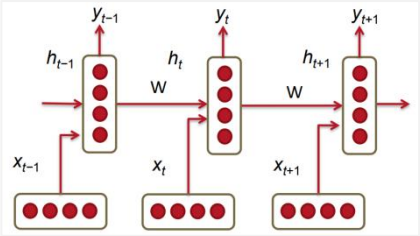
注意权重矩阵 $W^{(1)}$ 是应用在词向量上（上图中的绿色实线箭头），$W^{(2)}$ 是应用在隐藏层（也是绿色实线箭头）和 $W^{(3)}$ 是应用在词向量（绿色虚线箭头）。

这个模型的简化版本如右图所示，其中蓝色的层表示输入单词的 embedding 拼接：$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$，红色的层表示隐藏层：$\mathbf{h} = f(\mathbf{W}e + \mathbf{b}_1)$，绿色的输出分布是对词表的一个 softmax 概率分布：$\hat{y} = \text{softmax}(Uh + b_2)$。

2. Recurrent Neural Networks (RNN)

传统的翻译模型只能以有限窗口大小的前 n 个单词作为条件进行语言模型建模，循环神经网络与其不同，RNN 有能力以语料库中所有前面的单词为条件进行语言模型建模。

右图展示的 RNN 的架构，其中矩形框是在一个时间步的一个隐藏层 t。

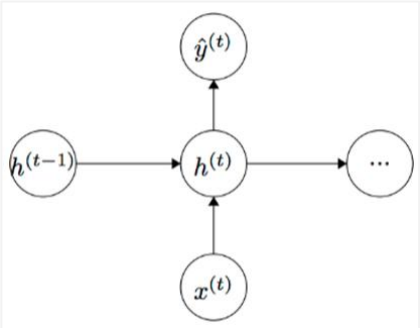


↑ Figure 3: A Recurrent Neural Network (RNN). Three time-steps are shown.【图 3：递归神经网络（RNN），显示了三个时间步骤。】

每个这样的隐藏层都有若干个神经元，每个神经元对输入向量用一个线性矩阵运算然后通过非线性变化（例如 tanh 函数）得到输出。在每一个时间步，隐藏层都有两个输入：前一个时间步的隐藏层 h_{t-1} 和当前时间步的输入 x_t，前一个时间步的隐藏层 h_{t-1} 通过和权重矩阵 $W^{(hh)}$ 相乘和当前时间步的输入 x_t 和权重矩阵 $W^{(hx)}$ 相乘得到当前时间步的隐藏层 h_t，然后再将 h_t 和权重矩阵 $W^{(S)}$ 相乘，接着对整个词表通过 softmax 计算得到下一个单词的预测结果 $\hat{y}$，如下面公式所示：

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right) \\ \hat{y} &= \text{softmax}\left(W^{(S)}h_t\right) \end{aligned}$$

每个神经元的输入和输出如右图所示：



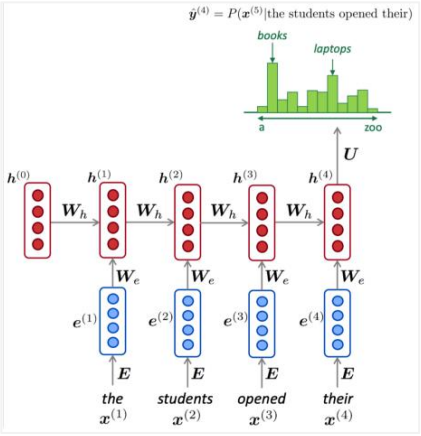
↑ Figure 4: The inputs and outputs to a neuron of a RNN.【图 4：RNN 神经元的输入和输出】

在这里一个有意思的地方是在每一个时间步使用相同的权重 $W^{(hh)}$ 和 $W^{(hx)}$。这样模型需要学习的参数就变少了，这与输入序列的长度无关——这从而解决了维度灾难。

以下是网络中每个参数相关的详细信息：

- $x_1, ..., x_{t-1}, x_t, x_{t+1}, ...x_T$：含有 T 个单词的语料库对应的词向量。
- $h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$：每个时间步 t 的隐藏层的输出特征的计算关系
- $x_t \in \mathbb{R}^d$：在时间步 t 的输入词向量。
- $W^{hx} \in \mathbb{R}^{D_h \times d}$：输入词向量 x_t 对应的权重矩阵。
- $W^{hh} \in \mathbb{R}^{D_h \times D_h}$：上一个时间步的输出 h_{t-1} 对应的权重矩阵。
- $h_{t-1} \in \mathbb{R}^{D_h}$：上一个时间步 t-1 的非线性函数输出。$h_0 \in \mathbb{R}^{D_h}$ 是在时间步 t=0 的隐藏层的一个初始化向量。
- σ：非线性函数（这里是 sigmoid 函数）。
- $\hat{y} = \text{softmax}(W^{(S)}h_t)$：在每个时间步 t 全部单词的概率分布输出。本质上，$\hat{y}$ 是给定文档上下文分数（例如 h_{t-1}）和最后观测的词向量 x_t，对一个出现单词的预测。这里，$W^{(S)} \in \mathbb{R}^{|V| \times D_h}$，$\hat{y} \in \mathbb{R}^{|V|}$，其中 $|V|$ 是词汇表的大小。

一个 RNN 语言模型的例子如右图所示。右图中的符号有一些的不同：W_h 等同于 $W^{(hh)}$，W_e 等同于 $W^{(hx)}$，U 等同于 $W^{(S)}$。E 表示单词输入 $x^{(t)}$ 转化为 $e^{(t)}$。



↑ Figure 5: An RNN Language Model.【图 5:RNN 语言模型。】

在 RNN 中常用的损失函数是在之前介绍过的交叉熵误差。下面的公式是这个函数在时间步 t 全部单词的求和。最后计算词表中的 softmax 计算结果展示了基于前面所有的单词对输出单词 $x^{(5)}$ 的不同选择的概率分布。这时的输入可以比 4 到 5 个单词更长。

2.1 RNN Loss and Perplexity

RNN 的损失函数一般是交叉熵误差。

$$J^{(t)}(\theta) = \sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

在大小为 T 的语料库上的交叉熵误差的计算如下：

$$J = -\frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

2.1 Notes info.

课件/Slides	Lecture 6, P24
视频/Video	Lecture 6, 26:13
GitHub • 代码	实时在线查阅文档
Bilibili • 视频	中英字幕课程视频

Stanford University X ShowMeAI

2.2 Advantages, Disadvantages and Applications of RNNs

RNN 有以下优点:

1. 它可以处理任意长度的序列
2. 对更长的输入序列不会增加模型的参数大小
3. 对时间步 t 的计算理论上可以利用前面很多时间步的信息
4. 对输入的每个时间步都应用相同的权重，因此在处理输入时具有对称性

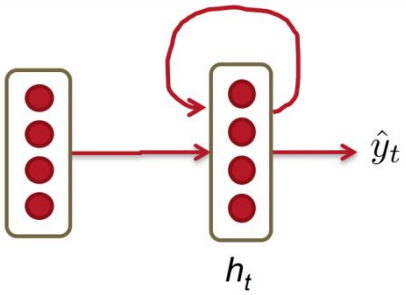
但是 RNN 也有以下缺点:

1. 计算速度很慢——因为它每一个时间步需要依赖上一个时间步，所以不能并行化
2. 在实际中因为梯度消失和梯度爆炸，很难利用到前面时间步的信息。

运行一层 RNN 所需的内存量与语料库中的单词数成正比。例如，我们把一个句子是为一个 mini batch，那么一个有 k 个单词的句子在内存中就会占用 k 个词向量的存储空间。同时，RNN 必须维持两对 W 和 b 矩阵。然而 W 的可能是非常大的，它的大小不会随着语料库的大小而变化（与传统的语言模型不一样）。对于具有 1000 个循环层的 RNN，矩阵 W 的大小为 1000×1000 而与语料库大小无关。

RNN 可以应用在很多任务，例如标注任务（词性标注、命名实体识别），句子分类（情感分类），编码模块（问答任务，机器翻译和其他很多任务）。在后面的两个任务，我们希望得到对句子的表示，这时可以通过采用该句子中时间步长的所有隐藏状态的 $element-wise$ 的最大值或平均值来获得。

右图是一些出版物中对 RNN 模型的另外一种表示。它将 RNN 的每个隐层用一个环来表示。



↑ Figure 6: The illustration of a RNN as a loop over time-steps.【图 6:RNN 随时间步长循环的图示】

2.3 Vanishing Gradient & Gradient Explosion Problems

RNN 从一个时间步传播权值矩阵到下一个时间步。回想一下，RNN 实现的目标是通过长距离的时间步来传播上下文信息。例如，考虑以下两个句子:

2.3 Notes info.

课件/Slides	Lecture 7, P6
视频/Video	Lecture 7, 04:37
GitHub · 代码	实时在线查阅文档
Bilibili · 视频	中英字幕课程视频

Stanford University X ShowMeAI

Sentence 1

"Jane walked into the room. John walked in too. Jane said hi to ___"

Sentence 2

"Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to ___"

对上面的两个句子，根据上下文，都可以知道空白处的答案是“John”,第二个在两个句子的上下文中均提及了好几次的人。迄今为止我们对 RNN 的了解，在理想情况下，RNN 也是能够计算得到正确的答案。然而，在实际中，RNN 预测句子中的空白处答案正确可能性，第一句要比第二句高。这是因为在反向传播的阶段的过程中，从前面时间步中回传过来的梯度值会逐渐消失。因此，对于长句子，预测到“John”是空

白处的答案的概率会随着上下文信息增大而减少。下面，我们讨论梯度消失问题背后的数学原因。

考虑公式在时间步 t，计算 RNN 误差 $\frac{dE}{dW}$，然后我们把每个时间步的误差都加起来。也就是说，计算并累积每个时间步长 t 的 $\frac{dE_t}{dW}$。

$$\frac{\partial \mathrm{E}}{\partial \mathrm{W}}=\sum_{\mathrm{i}=1}^{\mathrm{T}} \frac{\partial \mathrm{E}_{\mathrm{t}}}{\partial \mathrm{W}}$$

通过将微分链式法则应用于公式 (6) 和 (5) 来计算每个时间步长的误差；公式 (11) 展示对应的微分计算。注意 $\frac{dh_t}{dh_k}$ 是 h_t 对之前所有的 k 个时间步的偏导数。

$$\frac{\partial \mathrm{E}_{\mathrm{t}}}{\partial \mathrm{W}}=\sum_{\mathrm{k}=1}^{\mathrm{T}} \frac{\partial \mathrm{E}_{\mathrm{t}}}{\partial \mathrm{y}_{\mathrm{t}}} \frac{\partial \mathrm{y}_{\mathrm{t}}}{\partial h_{\mathrm{t}}} \frac{\partial h_{\mathrm{t}}}{\partial h_{\mathrm{k}}} \frac{\partial h_{\mathrm{k}}}{\partial \mathrm{W}}$$

下式 展示了计算每个 $\frac{dh_t}{dh_k}$ 的关系；这是在时间间隔 $[k,t]$ 内对所有的隐藏层的应用一个简单的微分链式法则。

$$\frac{\partial h_{\mathrm{t}}}{\partial h_{\mathrm{k}}}=\prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{t}} \frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1}}=\prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{t}} \mathrm{W}^{\mathrm{T}} \times \operatorname{diag}\left[\mathrm{f}^{\prime}\left(\mathrm{j}_{\mathrm{j}-1}\right)\right]$$

因为 $h \in \mathbb{R}^{\mathrm{D_n}}$，每个 $\frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1}}$ 是 h 的 Jacobian 矩阵的元素：

$$\frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1}}=\left[\frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1,1}} \cdots \frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1,\mathrm{D_n}}}\right]=\begin{bmatrix} \frac{\partial h_{\mathrm{j},1}}{\partial h_{\mathrm{j}-1,1}} & \cdots & \frac{\partial h_{\mathrm{j},1}}{\partial h_{\mathrm{j}-1,\mathrm{D_n}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{\mathrm{j},\mathrm{D_n}}}{\partial h_{\mathrm{j}-1,1}} & \cdots & \frac{\partial h_{\mathrm{j},\mathrm{D_n}}}{\partial h_{\mathrm{j}-1,\mathrm{D_n}}} \end{bmatrix}$$

将公式合起来，我们有以下关系。

$$\frac{\partial \mathrm{E}}{\partial \mathrm{W}}=\sum_{\mathrm{t}=1}^{\mathrm{T}}\sum_{\mathrm{k}=1}^{\mathrm{t}}\frac{\partial \mathrm{E}_{\mathrm{t}}}{\partial \mathrm{y}_{\mathrm{t}}}\frac{\partial \mathrm{y}_{\mathrm{t}}}{\partial h_{\mathrm{t}}}\left(\prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{t}}\frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1}}\right)\frac{\partial h_{\mathrm{k}}}{\partial \mathrm{W}}$$

下式展示了 Jacobian 矩阵的范数。这里的 β_{W} 和 β_h 是这两个矩阵范数的上界值。因此通过公式所示的关系计算在每个时间步 t 的部分梯度范数。

$$\left\|\frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1}}\right\| \leq \left\|\mathrm{W}^{\mathrm{T}}\right\|\left\|\mathrm{diag}\left[\mathrm{f}'\left(h_{\mathrm{j}-1}\right)\right]\right\| \leq \beta_{\mathrm{W}}\beta_h$$

计算这两个矩阵的 L2 范数。在给定的非线性函数 sigmoid 下，$\mathrm{f}'\left(h_{\mathrm{j}-1}\right)$ 的范数只能等于 1。

$\left\|\frac{\partial h_{\mathrm{t}}}{\partial h_{\mathrm{k}}}\right\| = \left\|\prod_{\mathrm{j}=\mathrm{k}+1}^{\mathrm{t}}\frac{\partial h_{\mathrm{j}}}{\partial h_{\mathrm{j}-1}}\right\| \leq (\beta_{\mathrm{W}}\beta_h)^{\mathrm{t}-\mathrm{k}}$ 当 $t-k$ 足够大和 $\beta_W\beta_h$ 远远小于 1 或者远远大于 1，指数项 $(\beta_W\beta_h)^{t-k}$ 的值就很容易变得非常小或者非常大。由于单词之间的距离过大，用一个很大的 $t-k$ 评估交叉熵误差可能会出现问题。在反向传播的早期就出现梯度消失，那么远处单词对在时间步长 t 预测下一个单词中，所起到的作用就会变得很小。

在实验的过程中，一旦梯度的值变得非常大，会导致在运行过程中容易检测到其引起的溢出（即 NaN）；这样的问题称为梯度爆炸问题。然而，当梯度接近为 0 的时候，梯度近乎不再存在，同时降低模型对语料库中的远距离的单词的学习质量；这样的问题称为梯度消失问题。如果相对梯度消失问题的有更直观的了解，你可以访问这个样例网站。

样例网站：
https://link.zhihu.com/?target=http%3A//cs224d.stanford.edu/notebooks/vanishing_grad_example.html

2.4 Solution to the Exploding & Vanishing Gradients

现在我们知道了梯度消失问题的本质以及它在深度神经网络中如何表现出来，让我们使用一些简单实用的启发式方法来解决这些问题。

为了解决梯度爆炸的问题，Thomas Mikolov 等人首先提出了一个简单的启发式解决方案，每当梯度大于一个阈值的时候，将其截断为一个很小的值，具体如下面算法中的伪代码所示。

2.4 Notes info.

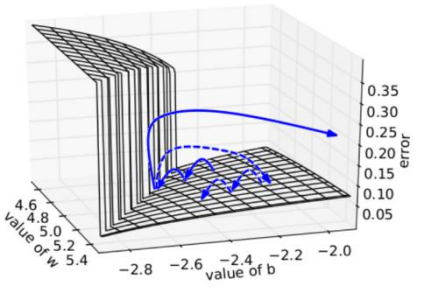
课件/Slides	Lecture 7, P19
视频/Video	Lecture 7, 22:10
GitHub · 代码	实时在线查阅文档
Bilibili · 视频	中英字幕课程视频

Stanford University X ShowMeAI

Algorithm：Pseudo-code for norm clipping in the gradients whenever they explode【算法：伪代码的范数剪辑在梯度每当他们爆炸】

$$\hat{g} \leftarrow \frac{\partial E}{\partial W}$$
if $\| \hat{g} \| \geq threshold$ **then**
$$\hat{g} \leftarrow \frac{threshold}{\| \hat{g} \|} \hat{g}$$
end if

右图可视化了梯度截断的效果。它展示了一个权值矩阵为 W 和偏置项为 b 的很小的 RNN 神经网络的决策界面。该模型由一个单一单元的循环神经网络组成，在少量的时间步长上运行；实心箭头阐述了在每个梯度下降步骤的训练过程。当在梯度下降的过程中，模型碰到目标函数中的高误差壁时，梯度被推到决策面上的一个遥远的位置。截断模型生成了虚线，在那里它将误差梯度拉回到靠近原始梯度的地方。



❐ ↑ Figure 7: Gradient explosion clipping visualization【图 7：渐变爆炸剪裁可视化】

为了解决梯度消失问题，我们提出两个技术。第一个技术是不去随机初始化 $W^{(hh)}$，而是初始化为单位矩阵。

第二个技术是使用 Rectified Linear（ReLU）单元代替 sigmoid 函数。ReLU 的导数是 0 或者 1。这样梯度传回神经元的导数是 1，而不会在反向传播了一定的时间步后梯度变小。

2.5 Deep Bidirectional RNNs

到目前为止，我们已经讨论了用 RNN 如何使用过去的词来预测序列中的下一个单词。同理，可以通过令 RNN 模型向反向读取语料库，根据未来单词进行预测。Irsoy 等人展示了一个双向深度神经网络；在每个时间步 t，这个网络维持两个隐藏层，一个是从左到右传播而另外一个是从右到左传播。为了在任何时候维持两个隐藏层，该网络要消耗的两倍存储空间来存储权值和偏置参数。最后的分类结果 $\hat{y}$，是结合由两个 RNN 隐藏层生成的结果得分产生。右图展示了双向 RNN 的网络结构。

2.5 Notes info.

课件/Slides	Lecture 7, P36
视频/Video	Lecture 7, 59:00
GitHub • 代码	实时在线查阅文档
Bilibili • 视频	中英字幕课程视频

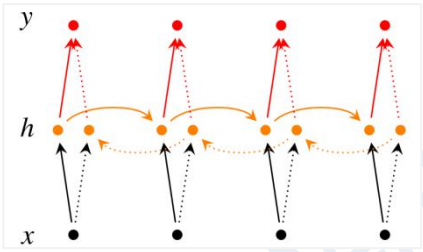
Stanford University X ShowMeAI

而下式展示了给出了建立双向 RNN 隐层的数学公式。两个公式之间唯一的区别是递归读取语料库的方向不同。最后一行展示了通过总结过去和将来的单词表示，显示用于预测下一个单词的分类关系。

$$\vec{h}_t = f\left(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b}\right)$$

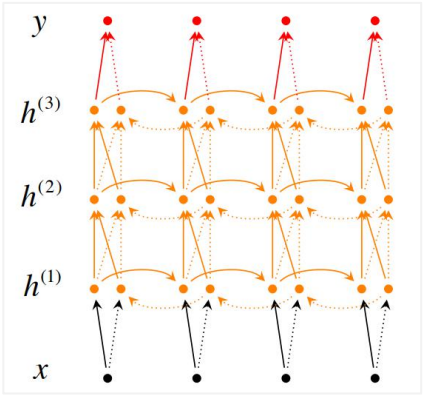
$$\overleftarrow{h}_t = f\left(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t-1} + \overleftarrow{b}\right)$$

$$\hat{y}_t = g(Uh_t + c) = g\left(U\left[\vec{h}_t ; \overleftarrow{h}_t\right] + c\right)$$



❐ ↑ Figure 8: A bi-directional RNN model【图 8：双向 RNN 模型】

RNN 也可以是多层的。右图展示一个多层的双向 RNN，其中下面的隐藏层传播到下一层。如图所示，在该网络架构中，在时间步 t，每个中间神经元从前一个时间步（在相同的 RNN 层）接收一组参数和前一个 RNN 隐藏层的两组参数；这两组参数一组是从左到右的 RNN 输入，另外一组是从右到左的 RNN 输入。



❐ ↑ Figure 9: A deep bi-directional RNN with three RNN layers.【图 9：具有三个 RNN 层的深双向 RNN。】

为了构建一个 L 层的深度 RNN，上述的关系要修改为在公式中的关系，其中在第 i 层的每个中间神经元的输入是在相同时间步 t 的 RNN 第 $i-1$ 层的输出。最后的输出 $\hat{y}$，每个时间步都是输入参数通过所有隐层传播的结果。

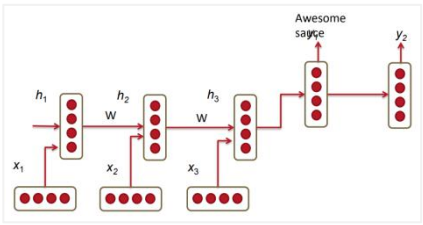
$$\overrightarrow{h}_t^{(i)} = f\left(\overrightarrow{W}^{(i)} x_t + \overrightarrow{V}\,\overrightarrow{h}_{t-1} + \overrightarrow{b}\right)$$

$$\overleftarrow{h}_t = f\left(\overleftarrow{W} x_t + \overleftarrow{V}\,\overleftarrow{h}_{t-1} + \overleftarrow{b}\right)$$

$$\hat{y}_t = g(Uh_t + c) = g\left(U\left[\overrightarrow{h}_t ; \overleftarrow{h}_t\right] + c\right)$$

2.6 Application: RNN Translation Model

传统的翻译模型是非常复杂的：它们包含很多应用在语言翻译流程的不同阶段的机器学习算法。在本节中，我们讨论采用 RNN 作为传统翻译模型的替代方法的潜力。考虑右图中展示的 RNN 模型；其中德语短语 Echt dicke Kiste 翻译为 Awesome sauce。



❐ ↑ Figure 10: A RNN-based translation model. The first three RNN hidden layers belong to the source language model encoder, and the last two belong to the destination language model decoder【图 10：基于 RNN 的翻译模型，前三个 RNN 隐藏层属于源语言模型编码器，后两个属于目标语言模型解码器】

首先，前三个时间步的隐藏层 编码 德语单词为一些语言的单词特征（ h_3 ）。后面两个时间步解码 h_3 为英语单词输出。下式分别展示了编码阶段和解码阶段(后两行)。

$$h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$$

$$h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$$

$$y_t = \text{softmax}\left(W^{(s)} h_t\right)$$

一般可以认为使用交叉熵函数的 RNN 模型可以生成高精度的翻译结果。实际上，在模型中增加一些扩展方法可以提升翻译的准确度表现。

$$\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p_\theta\left(y^{(n)} | x^{(n)}\right)$$

扩展 1：在训练 RNN 的编码和解码阶段时，使用不同的权值。这使两个单元解耦，让两个 RNN 模块中的每一个进行更精确的预测。这意味着在公式 (23) 和 (24) 在 $\phi()$ 函数中是使用不同的 $W^{(hh)}$ 矩阵。

扩展 2：使用三个不同的输入计算解码器中的每个隐藏状态

- 前一个隐藏状态 h_{t-1}（标准的）
- 编码阶段的最后一个隐藏层（上图中的 $c = h_T$）
- 前一个预测的输出单词 $\hat{y}_{t-1}$

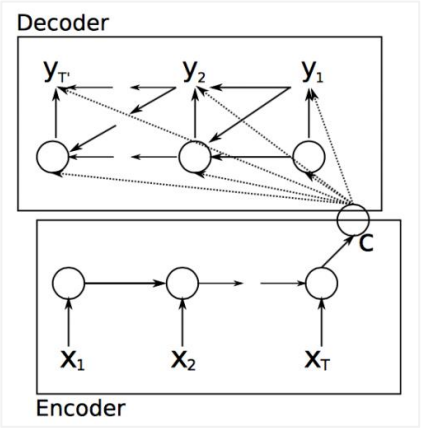
将上述的三个输入结合将之前公式的解码函数中的 ϕ 函数转换为下式的 ϕ 函数。上图展示了这个模型。

$$h_t = \phi(h_{t-1}, c, y_{t-1})$$

扩展 3：使用多个 RNN 层来训练深度循环神经网络。神经网络的层越深，模型的就具有更强的学习能力从而能提升预测的准确度。当然，这也意味着需要使用大规模的语料库来训练这个模型。

扩展 4：训练双向编码器，提高准确度。

扩展 5：给定一个德语词序列 A B C，它的英语翻译是 X Y。在训练 RNN 时不使用 A B C → X Y，而是使用 C B A → X Y。这么处理的原因是 A 更有可能被翻译成 X。因此对前面讨论的梯度消失问题，**反转输入句子的顺序有助于降低输出短语的错误率**。



↑ Figure 11: Language model with three inputs to each decoder neuron: (h_{t-1}, c, y_{t-1})【图 11：每个解码器有三个输入的语言模型 (h_{t-1}, c, y_{t-1})】

3. Gated Recurrent Units

除了迄今为止讨论的扩展方法之外，我们已经发现 RNN 通过使用更复杂的激活单元来获得表现更好。到目前为止，我们已经讨论了从隐藏状态 h_{t-1} 向 h_t 转换的方法，使用了一个仿射转换和 $point-wise$ 的非线性转换。在这里，我们讨论门激活函数的使用并修改 RNN 结构。虽然理论上 RNN 能捕获长距离信息，但实际上很难训练网络做到这一点。门控制单元可以让 RNN 具有更多的持久性内存，从而更容易捕获长距离信息。让我们从数学角度上讨论 GRU 如何使用 h_{t-1} 和 x_t 来生成下一个隐藏状态 h_t。然后我们将深入了解 GRU 架构。

3 Notes info.

课件/Slides	Lecture 7, P28
视频/Video	Lecture 7, 46:30
GitHub • 代码	实时在线查阅文档
Bilibili • 视频	中英字幕课程视频

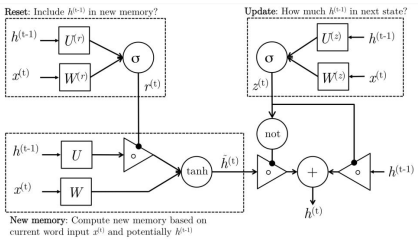
Stanford University X ShowMeAI

$$
\begin{aligned}
z_t &= \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right) && \text{(Update gate)}\\
r_t &= \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right) && \text{(Reset gate)}\\
\tilde{h}_t &= \tanh(r_t \circ Uh_{t-1} + Wx_t) && \text{(New memory)}\\
h_t &= (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} && \text{(Hidden state)}
\end{aligned}
$$

上述的共识可以认为是 GRU 的四个基本操作阶段，下面对这些公式作出更直观的解释，右图展示了 GRU 的基本结构和计算流程：

1. **New memory generation**：一个新的记忆 $\tilde{h}_t$ 是由一个新的输入单词 x_t 和过去的隐藏状态 h_{t-1} 共同计算所得。这个阶段是将新输入的单词与过去的隐藏状态 h_{t-1} 相结合，根据过去的上下文来总结得到向量 $\tilde{h}_t$ 。
2. **Reset Gate**：复位信号 r_t 是负责确定 h_{t-1} 对总结 $\tilde{h}_t$ 的重要程度。如果确定 $\tilde{h}_t$ 与新的记忆的计算无关，则复位门能够完全消除过去的隐藏状态（即忽略之前隐藏的信息）。
3. **Update Gate**：更新信号 z_t 负责确定有多少 h_{t-1} 可以向前传递到下一个状态。例如，如果 $z_t \approx 1$ ，然后 h_{t-1} 几乎是完全向前传递到下一个隐藏状态。反过来，如果 $z_t \approx 0$ ，然后大部分的新的记忆 $\tilde{h}_t$ 向前传递到下一个隐藏状态。
4. **Hidden state**：利用更新门的建议，使用过去的隐藏输入 h_{t-1} 和新生成的记忆 $\tilde{h}_t$ 生成隐藏状态 h_t 。

需要注意的是，为了训练 GRU，我们需要学习所有不同的参数：$W, U, W^{(r)}, U^{(r)}, W^{(z)}, U^{(z)}$ 。这些参数同样是通过反向传播算法学习所得。



❐ ↑ Figure 12: The detailed internals of a GRU【图 12:GRU 的详细内部结构，大图见本节最后一页】

4. Long-Short-Term-Memories

Long-Short-Term-Memories 是和 GRU 有一点不同的另外一种类型的复杂激活神经元。它的作用与 GRU 类似，但是神经元的结构有一点区别。我们首先来看看 LSTM 神经元的数学公式，然后再深入了解这个神经元的设计架构：

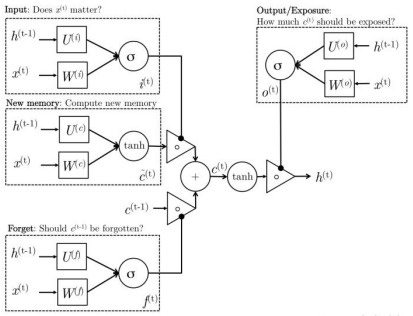
$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right) && \text{(Input gate)}\\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right) && \text{(Forget gate)}\\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right) && \text{(Output/Exposure gate)}\\
\tilde{c}_t &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right) && \text{(New memory cell)}\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)}\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

4 Notes info.

课件/Slides	Lecture 7, P22
视频/Video	Lecture 7, 29:40
GitHub · 代码	实时在线查阅文档
Bilibili · 视频	中英字幕课程视频

Stanford University X ShowMeAI

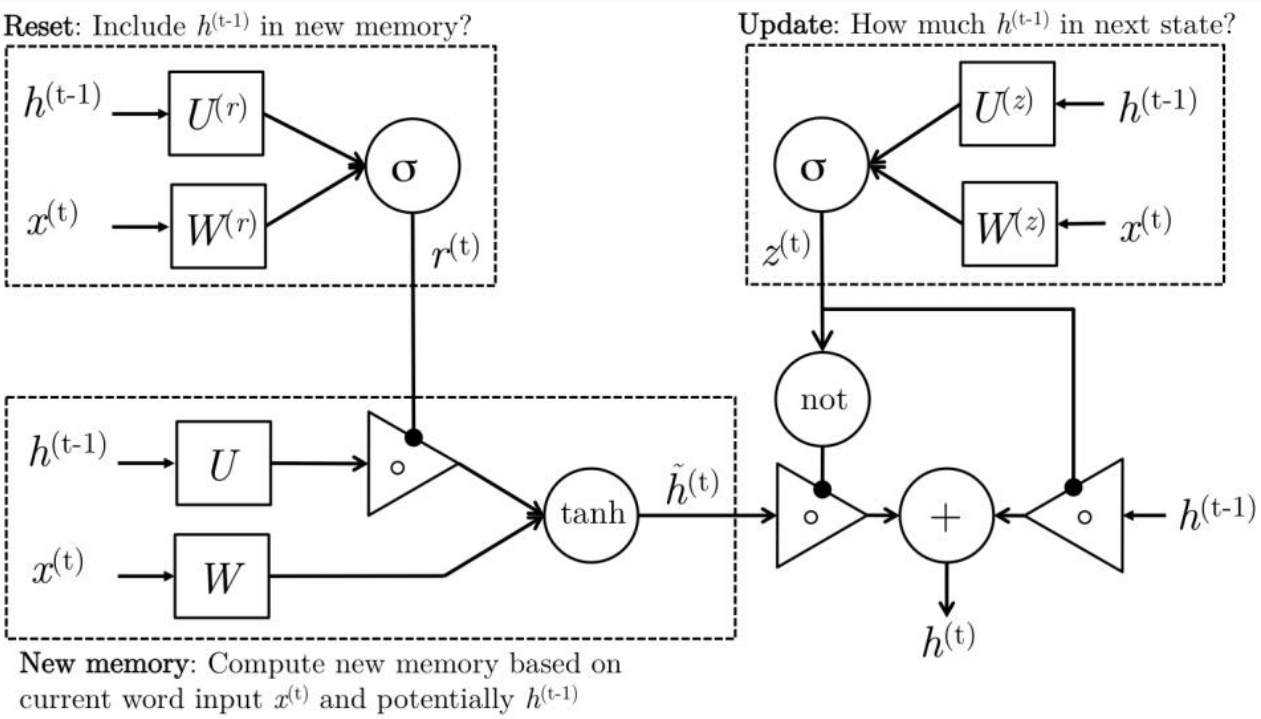
右图是 LSTM 的计算图示



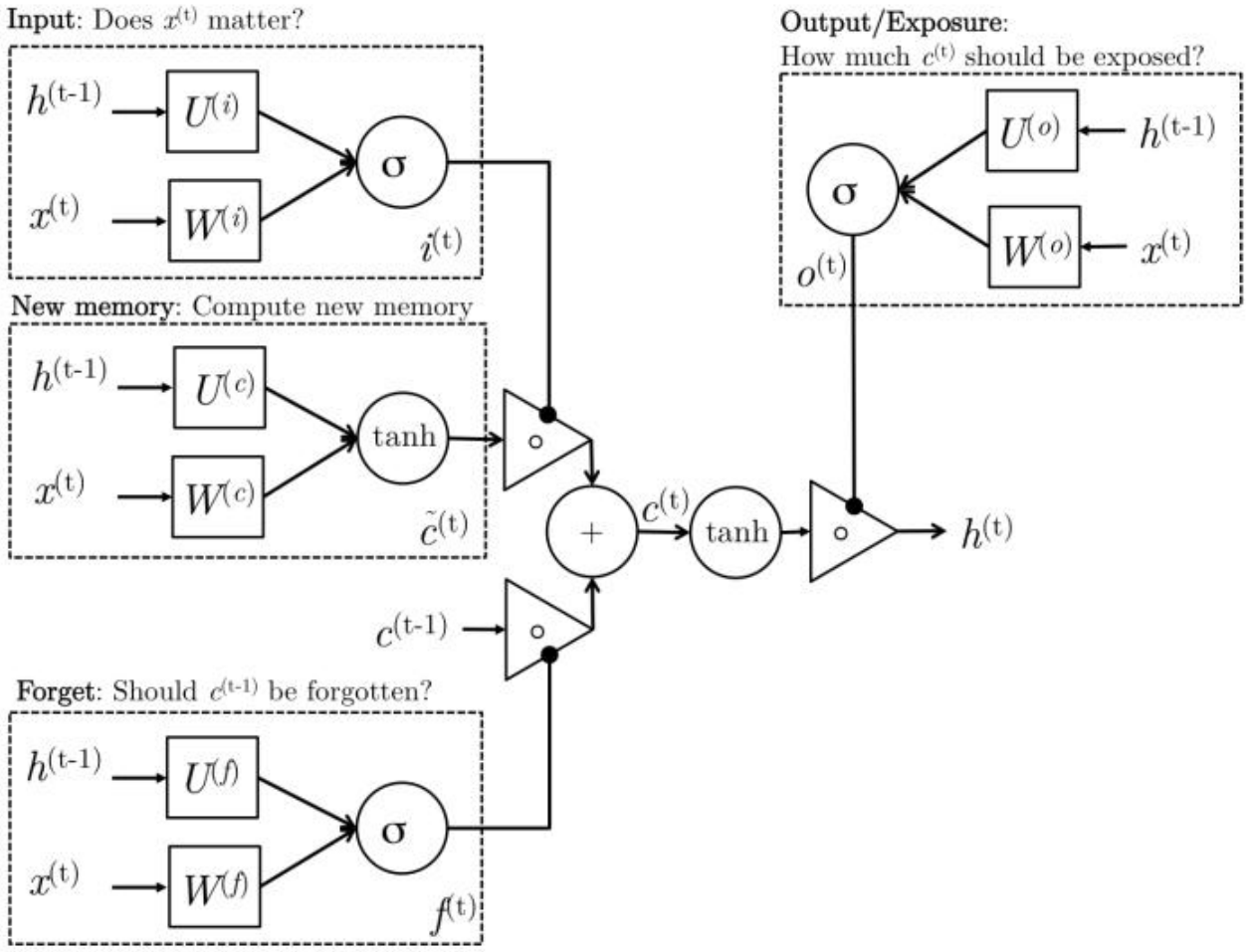
↑ Figure 13: The detailed internals of a LSTM 【图 13:LSTM 的详细内部结构，大图见本节最后一页】

我们可以通过以下步骤了解 LSTM 的架构以及这个架构背后的意义：

1. **New memory generation**：这个阶段是类似于 GRU 生成新的记忆的阶段。我们基本上是用输入单词 x_t 和过去的隐藏状态来生成一个包括新单词 $x^{(t)}$ 的新的记忆 $\tilde{c}_t$ 。
2. **Input Gate**：我们看到在生成新的记忆之前，新的记忆的生成阶段不会检查新单词是否重要——这需要输入门函数来做这个判断。输入门使用输入词和过去的隐藏状态来决定输入值是否值得保存，从而用来进入新内存。因此，它产生它作为这个信息的指示器。
3. **Forget Gate**：这个门与输入门类似，只是它不确定输入单词的有用性——而是评估过去的记忆是否对当前记忆的计算有用。因此，遗忘门查看输入单词和过去的隐藏状态，并生成 f_t。
4. **Final memory generation**：这个阶段首先根据忘记门 f_t 的判断，相应地忘记过去的记忆 c_{t-1} 。类似地，根据输入门 i_t 的判断，相应地输入新的记忆 $\tilde{c}_t$ 。然后将上面的两个结果相加生成最终的记忆 c_t 。
5. **Output/Exposure Gate**：这是 GRU 中没有明确存在的门。这个门的目的是从隐藏状态中分离最终的记忆。最终的记忆 c_t 包含很多不需要存储在隐藏状态的信息。隐藏状态用于 LSTM 的每个单个门，因此，该门是要评估关于记忆单元 c_t 的哪些部分需要显露在隐藏状态 h_t 中。用于评估的信号是 o_t ，然后与 c_t 通过 $o_t \circ \tanh(c_t)$ 运算得到最终的 h_t 。



❒ ↑ Figure 12: The detailed internals of a GRU【图 12:GRU 的详细内部结构】



❒ ↑ Figure 13: The detailed internals of a LSTM【图 13:LSTM 的详细内部结构】

机器学习	深度学习	自然语言处理	计算机视觉	知识图谱
Machine Learning	Deep Learning	Natural Language Processing	Computer Vision	Knowledge Graphs
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n	Stanford · CS520

 # 系列内容 Awesome AI Courses Notes Cheatsheets

图机器学习	深度强化学习	自动驾驶
Machine Learning with Graphs	Deep Reinforcement Learning	Deep Learning for Self-Driving Cars
Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094
...	...	...

是 ShowMeAI 资料库的分支系列，覆盖最具知名度的 TOP20+门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

斯坦福大学(Stanford University) Natural Language Processing with Deep Learning (CS224n) 课程，是本系列的第三门产出。

课程版本为 2019 Winter，核心深度内容(transformer、bert、问答、摘要、文本生成等)在当前(2021 年)工业界和研究界依旧是前沿的方法。最新版课程的笔记生产已在规划中，也敬请期待。

笔记内容经由深度加工整合，以 5 个部分构建起完整的“CS224n 内容世界”，并依托 GitHub 创建了汇总页。快扫描二维码，跳转进入吧！有任何建议和反馈，也欢迎通过下方渠道和我们联络 (*￣3￣)~