总结

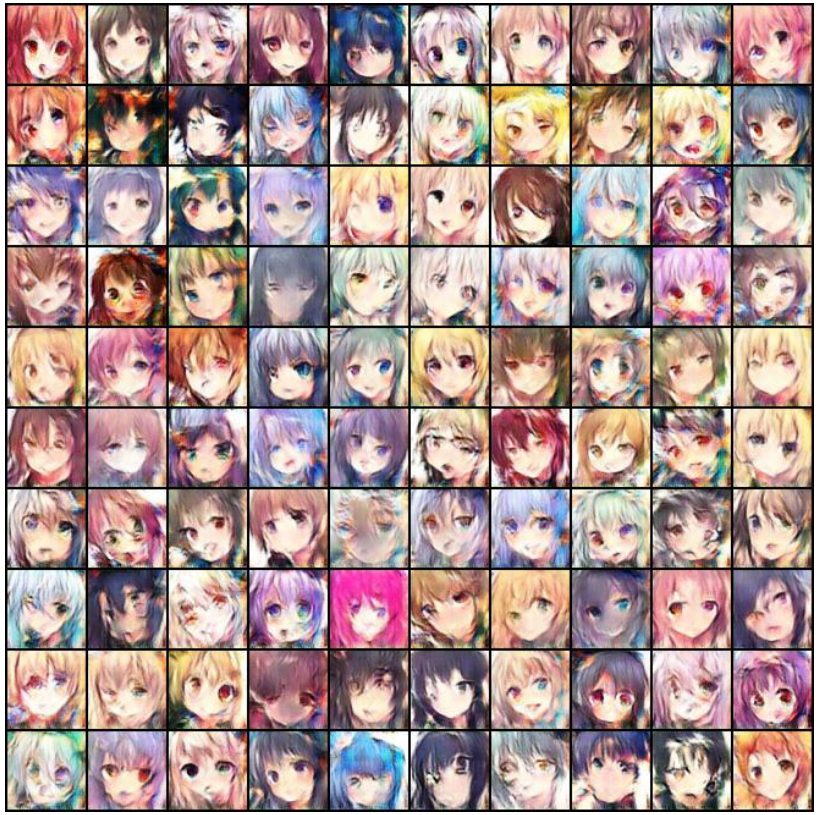
黄飞虎

2021.2.10

学习情况

看了李宏毅的视频，完成了实验GAN

```
Epoch [1/12] 1115/1115 Loss_D: 0.1006 Loss_G: 4.6194 | Save some samples to .\logs\Epoch_001.jpg.
Epoch [2/12] 1115/1115 Loss_D: 0.1960 Loss_G: 6.9085 | Save some samples to .\logs\Epoch_002.jpg.
Epoch [3/12] 1115/1115 Loss_D: 0.0187 Loss_G: 7.9961 | Save some samples to .\logs\Epoch_003.jpg.
Epoch [4/12] 1115/1115 Loss_D: 0.1304 Loss_G: 1.9668 | Save some samples to .\logs\Epoch_004.jpg.
Epoch [5/12] 1115/1115 Loss_D: 0.1750 Loss_G: 7.5980 | Save some samples to .\logs\Epoch_005.jpg.
Epoch [6/12] 1115/1115 Loss_D: 0.1964 Loss_G: 6.6192 | Save some samples to .\logs\Epoch_006.jpg.
Epoch [7/12] 1115/1115 Loss_D: 0.2668 Loss_G: 7.0876 | Save some samples to .\logs\Epoch_007.jpg.
Epoch [8/12] 1115/1115 Loss_D: 0.0927 Loss_G: 4.6872 | Save some samples to .\logs\Epoch_008.jpg.
Epoch [9/12] 1115/1115 Loss_D: 0.1267 Loss_G: 2.4959 | Save some samples to .\logs\Epoch_009.jpg.
Epoch [10/12] 1115/1115 Loss_D: 0.1018 Loss_G: 4.7500 | Save some samples to .\logs\Epoch_010.jpg.
Epoch [11/12] 1115/1115 Loss_D: 0.1888 Loss_G: 4.9839 | Save some samples to .\logs\Epoch_011.jpg.
Epoch [12/12] 1115/1115 Loss_D: 0.0423 Loss_G: 3.5435 | Save some samples to .\logs\Epoch_012.jpg.
```



论文情况

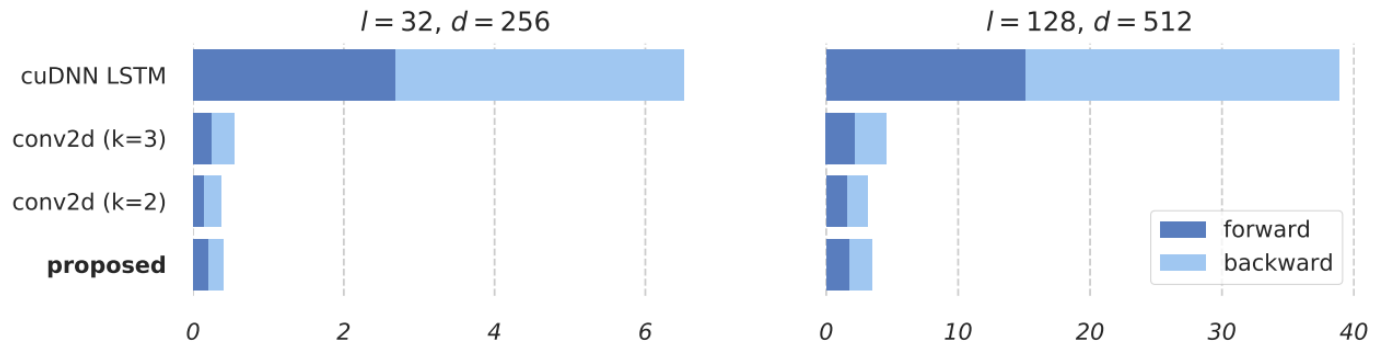
Training RNNs as Fast as CNNs

论文链接：https://arxiv.org/pdf/1709.02755v2.pdf

本文的工作

由于在并行状态计算上的内在困难，循环神经网络（RNN）的缩放很差。比如，h（t） 的前向计算被阻止直到 h（t-1） 的整个计算结束，这对并行计算来说是一个主要的瓶颈。

在本文我们介绍一种 Simple Recurrent Unit （SRU），比传统的RNN有显著的速度提升。它比起目前出现的循环实现都要快得多。循环单元简化了状态计算，从而表现出了类似 CNN、注意力模型和前馈网络的相同并行性。特别是，虽然内态 c_t 仍然利用以前的状态 c_{t-1} 更新，但是在循环步骤中，已经不再依赖于 h_{t-1} 了。结果，循环单元中所有的矩阵乘法运算可以很轻易在任何维度和步骤中并行化。



SRU的实现

比较流行的RNN结构比如LSTM 和GRU 都是通过gates的机制来控制信息流，首先是状态层更新做了些简化：

$$\begin{aligned}\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{x}_t \\ &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \tilde{\mathbf{x}}_t\end{aligned}$$

F_t和i_t分别是与lstm一致的forget gate 和 input gate，是一个是一个sigmoid gate，x_t是对输入x的处理变换。最后将内部状态c_t传给激活函数以计算输出状态。然后对输出状态h_t做额外的处理。

$$\begin{aligned}\mathbf{h}'_t &= \mathbf{r}_t \odot \mathbf{h}_t + (1 - \mathbf{r}_t) \odot \mathbf{x}_t \\ &= \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{x}_t\end{aligned}$$

速度优化

现有的 RNN 实现再循环计算中使用了前面的输出状态 h_{t-1}。例如，以往向量可以通过 $f_t=\sigma(W_f*x_t+R_f*(h_{t-1})+b_f)$ 计算得出。但是该式中的 $R*h_{t-1}$ 会破坏独立性和并行性，因为隐藏状态每一个维度都依赖于其他状态，因此 h_t 的计算必须等到整个 h_{t-1} 都完成计算。

$$
\begin{aligned}
\tilde{\mathbf{x}}_t &= \mathbf{W}\mathbf{x}_t \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{b}_f) \\
\mathbf{r}_t &= \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{b}_r) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \tilde{\mathbf{x}}_t \\
\mathbf{h}_t &= \mathbf{r}_t \odot g(\mathbf{c}_t) + (1 - \mathbf{r}_t) \odot \mathbf{x}_t
\end{aligned}
$$

实验

问答任务

不同模型在 SQuAD 上的准确匹配率和 F1 得分。我们也报告了每个 epoch 的整体处理时间、RNN 使用的时间。SRU 有更好的结果，运算速度比 cuDNN LSTM 快了 6 倍。

Model	# layers	d	Size	Dev EM	Dev F1	Time / epoch	
						RNN	Total
(Chen et al., 2017)	3	128	4.1m	69.5	78.8	-	-
Bi-LSTM	3	128	4.1m	69.6	78.7	534s	670s
Bi-LSTM	4	128	5.8m	69.6	78.9	729s	872s
Bi-SRU	3	128	2.0m	69.1	78.4	60s	179s
Bi-SRU	4	128	2.4m	69.7	79.1	74s	193s
Bi-SRU	5	128	2.8m	70.3	79.5	88s	207s

语言模型

在 PTB 语言模型数据集上的困惑度。对比的模型是使用相似的正则化与学习策略进行训练的：都使用了 cuDNN LSTM；除了（Zaremba et al., 2014), (Press and Wolf, 2017）模型，都是用了变分 dropout；除了 (Zaremba et al., 2014)，其他模型的输入和输出都附上了词嵌入；所有模型都使用了带有学习率衰减的 SGD。

Model	# layers	Size	Dev	Test	Time / epoch	
					RNN	Total
LSTM (Zaremba et al., 2014)	2	66m	82.2	78.4		
LSTM (Press and Wolf, 2017)	2	51m	75.8	73.2		
LSTM (Inan et al., 2016)	2	28m	72.5	69.0		
RHN (Zilly et al., 2017)	10	23m	67.9	65.4		
KNN (Lei et al., 2017)	4	20m	-	63.8		
NAS (Zoph and Le, 2016)	-	25m	-	64.0		
NAS (Zoph and Le, 2016)	-	54m	-	62.4		
cuDNN LSTM	2	24m	73.3	71.4	53s	73s
cuDNN LSTM	3	24m	78.8	76.2	64s	79s
SRU	3	24m	68.0	64.7	21s	44s
SRU	4	24m	65.8	62.5	23s	44s
SRU	5	24m	63.9	61.0	27s	46s
SRU	6	24m	63.4	60.3	28s	47s

Thank you