学习率的自适应调整在语言模型中的应用

文　摘： 神经网络被广泛应用于智能语音、自然语言处理、图像识别等各个领域，并应用小批量梯度下降法对其进行训练。然而该训练方法对初始学习率非常敏感，如果无法通过大量实验得到较好的初始学习率，会严重影响训练结果。本文将探究在 LSTM 语言模型中加入稳定算子以自适应地调整学习率和对稳定算子进行 L2 范数优化对训练结果、收敛速度、初始学习率的敏感程度上的影响。最后得出结论，在语言模型中加入学习率的自适应算法会降低初始学习率的敏感度，加快收敛速度，加入稳定算子的 L2 范数能小幅度提升模型性能。

关键词：语言模型；自适应稳定算子；长短时间记忆循环神经网络（LSTM）；梯度下降法； L2 范数；

中图分类号： TP181；TP183

1　介绍

语言模型是一种判断一句话的流畅程度及合理性的模型，一般情况下，是在给定历史语言信息的情况下，得出符合语言规律的下一个词，它在自然语言处理、机器翻译、自动语音识别等领域被广泛地应用与研究。最初，统计学的语言模型被提出，其中加入平滑算法的 N-gram[1]语言模型在很长的一段时间内，都在语言模型领域处于统治性的地位，直到深度学习的出现。

循环神经网络（RNN）[2][3]和加入长短时间记忆单元（LSTM）[4][5]的 RNN 由于能保存历史信息并利用，能够更好地应用于具有时序性的语言模型中，因此被广泛应用于研究。

神经网络模型的训练开始用的是批梯度下降（BGD）这种最优化求解方法，后因为训练速度过慢而采用随机梯度下降法（SGD）[6]，然而该方法依然存在迭代次数过多以及搜索盲目的问题，进而现在大家都用的是（MBGD）小批量梯度下降法[7]。

在该方法中，学习率是一个重要并且敏感的参数，需要用到先验知识和进行大量实验去找到符合特定模型的最优学习率才能得到最优解。在过去的研究中，在[8]中提出了在 DNN 模型中采用学习率的自适应算法，能够降低模型对初始学习率的敏感度和提升收敛速度。在此基础上，这个算法被应用到 LSTM 循环神经网络声学模型中[9]，虽然可以一定程度上降低对初始学习率的敏感度和收敛速度，却会牺牲掉一部分性能，效果并不理想。

在本文的工作中，我们将学习率的自适应算法应用在语言模型中，后文统一称为引入稳定算子 beta，在训练中动态调整 beta 以达到自适应学习率的目的。我们在不同的语言模型网络结构中尝试加入稳定算子，探究其对训练结果、收敛速度、初始学习率的敏感程度上的影响。同时，我们参考各机器学习工具的 RNN 实现中，往往会通过对参数加入 L2 范数优化以达到避免过拟合、提升模型性能的目的，对稳定算子加入 L2 范数，研究优化后的模型性能。

后文的结构主要如下：在第二部分介绍 LSTM 语言模型、小批量随机梯度下降和学习率的数学概述；第三部分阐述本文的工作，主要分为在语言模型的各种结构中加入稳定算子的学习率自适应算法，以及对稳定算子进行 L2 范数优化；第四部分为实验和分析；最后一部分是总结和展望。

2　相关基础概述

2.1 LSTM 语言模型中学习率的自适应调整

因为句子所具有的时序性，循环神经网络语言模型（RNNLM）作为一种有隐层自环的时序性神经网络模型，能够保存有历史信息并对其加以利用，打破了传统统计学语言模型的统治地位，并在短时间内发展迅速。其中，更有长短时间记忆（LSTM）单元替代传统的 RNNLM 中的隐层单元，以达到对更长历史信息的记忆和利用。其结构中有很多的门来控制对历史信息的记忆，公式如下：

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$

$$h_t = o_t \cdot \tanh(c_t)$$

其中σ表示 sigmoid 激活函数。

语言模型的损失函数为

$$L(\theta)=-\sum_{i=1}^{T}\log\left(P\left(s=t_i\mid o_i\right)\right)$$

在训练语言模型时，求损失函数对每个参数的偏导数即可得到梯度，接着利用梯度和步长（学习率）对模型参数进行更新。

2.2 随机梯度下降法和学习率

随机梯度下降（SGD）是一种常用的最优化问题求解方法，而小批量随机梯度下降（Mini-batch SGD）能够折衷训练时间和训练效果，更是广泛地被用于各类神经网络的训练中。训练目标是更新神经网络中的参数θ，首先根据损失函数L 计算出每个参数θ_i对应的梯度

$$\Delta_i=\frac{\partial L}{\partial\theta_i}$$

根据梯度和学习率对该参数进行更新

$$\theta_i=\theta_i-\eta\Delta_i$$

其中η 是学习率，即通俗意义上的步长。

实际训练中，学习率会随着训练的进行变化，调整学习率的方法有好几种，目前用的比较多的两种是提前停止法[10]和减半法[11]。提前停止法是当某一轮监督集上的结果变差了即停止训练。减半法是当某一轮的监督集上的结果变差时将学习率减半，继续进行训练。

但无论是哪一种调整学习率的方法，都对初始学习率的值非常敏感，学习率太大会导致无法收敛，学习太小同样会导致无法收敛或者收敛速度过慢。并且不同的初始学习率可能会对总的学习轮数影响较大，不同任务的合理的初始学习率也各不相同，为了得到合理的初始学习率，需要做大量的试验。因此为了减少甚至消除初始学习率对试验结果的影响，弱化掉选取初始学习率这一过程，关于在训练算法中加入稳定算子来进行学习率的自适应的算法应运而生。

LSTM 语言模型

本文着重研究在LSTM语言模型中加入稳定算子的影响，并对其进行 L2 范数以达到优化目的。

2.3 稳定算子 Beta 及自适应。

在微软的研究中，提出引入稳定算子进行模型学习率的自适应，使得神经网络对于学习率的敏感度降低，将研究人员从调整学习率的繁琐重复工作中解放出来。具体方法是为 DNN 神经网络中每一层配备一个稳定算子 beta。普通的 DNN 隐层的矩阵计算公式为

$$y=Wx+b$$

其中 x 是输入向量，y 是输出向量，W 是现行变化参数矩阵，b 是偏移系数向量。

加入 beta 稳定算子后的矩阵公式为

$$y=e^{\beta}Wx+b$$

e 是自然对数，β 是稳定算子。

加入 beta 后的模型训练过程中，前向传播可以直接按照上述公式进行，但是在反向传播训练的过程中，需要计算 x，W，b 和β的梯度。

其中 x 和 W 的梯度进行了小幅度修改，b 的梯度没有改变，如公式所示

$$\frac{\partial L}{\partial x}=e^{\beta}W^{\mathrm{T}}\frac{\partial L}{\partial y},$$

$$\frac{\partial L}{\partial W}=e^{\beta}\frac{\partial L}{\partial y}x^{\mathrm{T}},$$

$$\frac{\partial L}{\partial b}=\frac{\partial L}{\partial y}$$

剩下的问题是如何更新 beta 呢，根据求导的链式法则

$$\frac{\partial L}{\partial x}=\frac{\partial L}{\partial y}\frac{\partial y}{\partial\beta}=e^{\beta}\frac{\partial L^{\mathrm{T}}}{\partial y}Wx$$

根据上面求 x 的梯度公式，可以得到

$$\frac{\partial L}{\partial\beta}=\frac{\partial L^{\mathrm{T}}}{\partial x}x$$

故有

$$\beta=\beta-\eta\frac{\partial L^{\mathrm{T}}}{\partial x}x$$

2.4 LSTM 语言模型引入稳定算子

LSTM 是一种特殊的 RNN 网络，将原始的 RNN 网络的隐层单元替换为 LSTM 记忆单元，它不仅保留着 RNN 的记忆历史的功能，还能在记忆门的作用下动态调整记忆长短，如公式 1 所示。

由微软提出的 DNN 网络中的学习率自适应方法适用于普通深度神经网络中的现行矩阵变换，然而无法直接应用于在 LSTM 网络中，因为在每个 LSTM 单元中，分别有三个门和一个仿射

变换操作。因此在刘奇的研究中，主要比较了三种不同的方法：每层共享一个 beta，相同种类的门共享一个 beta，以及每个门的 beta 都各不相同。其中第三种方式中，每个 beta 能够单独地分别调整每个参数矩阵，因而表现最好。该方法公式的修改如下：

$$i_t = \sigma\left(e^{\beta_{xi}} W_{xi} x_t + e^{\beta_{hi}} W_{hi} h_{t-1} + e^{\beta_{ci}} W_{ci} c_{t-1} + b_i\right)$$

$$f_t = \sigma\left(e^{\beta_{xf}} W_{xf} x_t + e^{\beta_{hf}} W_{hf} h_{t-1} + e^{\beta_{cf}} W_{cf}\ c_{t-1} + b_f\right)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh\left(e^{\beta_{xc}} W_{xc} x_t + e^{\beta_{hc}} W_{hc} h_{t-1} + b_c\right)$$

$$o_t = \sigma\left(e^{\beta_{xo}} W_{xo} x_t + e^{\beta_{ho}} W_{ho} h_{t-1} + e^{\beta_{co}} W_{co} c_t + b_o\right)$$

$$h_t = o_t \cdot \tanh\left(c_t\right)$$

这也是本文采用的方法。

在加入稳定算子进行自适应的LSTM语言模型的训练过程中，别的参数的梯度求解方法和前面类似，包括 beta 在内的所有参数的更新方法，都是按照新的公式求解梯度，结合稳定算子和学习率进行更新。

2.5 稳定算子 beta 的 L2norm 范数

在语言模型中，原始的损失函数为

$$L(\theta) = -\sum_{i=1}^{T} \log\left(P\left(s = t_i \mid o_i\right)\right)$$

目前在各机器学习框架的实现过程中，会对所有参数矩阵 W 加入 L2 正则项优化，防止过拟合，使训练结果更好，加入 L2 优化后的损失函数为

$$L(\theta) = -\sum_{i=1}^{T} \log\left(P\left(s = t_i \mid o_i\right)\right) + \frac{\lambda}{2} \| W \|_2$$

如前文所述，在前人工作中，稳定算子 beta 作为参数被加入到 LSTM 语言模型的训练中，并不改变损失函数。只是在更新 beta 的时候，把 beta 当做和其它参数一样求偏导数得到梯度，在学习率的控制下进行更新。本文在损失函数中为 beta 加上 L2 范数项，以达到优化的目的。加入稳定算子 beta 的 L2 正则项后的损失函数为

$$L_2(\theta) = L(\theta) + \frac{\lambda_1}{2} \| W \|_2 + \frac{\lambda_2}{2} \| e^{\beta} \|_2$$

3 实验

3.1 实验设计

实验总共分为三个部分：

首先，我们设置不同的初始学习率，验证在语言模型中引入稳定算子是否同其它任务一样，会使任务对初始学习率不敏感，使得在初始学习率不优的情况下几乎不影响最后的结果。

其次，我们采用已经调优的初始学习率作为不加稳定算子的对比实验，首先我们在普通语言模型和多视角语言模型的基础上，分别尝试用前述三种方式中的两种方式：每个隐层 LSTM 单元的每个门拥有其各自的不同的稳定算子；每个隐层的 LSTM 单元中相同的门公用同一个稳定算子。其次，我们在不同隐层规模、不同深度的模型上尝试加入稳定算子 beta，并观察混淆度（PPL），学习率（lrate）和各算子的变化趋势，并比较结果。

最后，我们根据观察稳定算子的变化趋势对结果的影响，在稳定算子中加入 L2 范数约束，观察最后结果。

本次实验的数据均为 PTB 数据，该数据集为语言模型中通用的英文数据集，总共含有 887521 个词，词表大小为 10000，最后的判断准则为语言模型的混淆度（PPL）。

3.2 实验结果及分析

在第一部分试验中，我们在单层语言模型网络中进行试验，控制网络结构和除学习率以外的所有参数相同，通过变化学习率比较最后训练结果，实验结果如表 1，通过实验结果发现加入稳定算子确实可以使语言模型的训练免于对初始学习率不优的干扰。

表 1　稳定算子对不同的初始学习率的影响

初始学习率	是否加入稳定算子	PPL
0.015	否	138.8
0.08	否	124.0
0.15	否	103.3
0.8	否	117.1
1.5	否	129.4
0.015	是	108.4
0.08	是	108.0
0.15	是	107.3
0.8	是	109.4
1.5	是	110.1

第二部分实验中，我们主要是通过实验探究稳定算子加入到语言模型中是否会提升语言模型的性能；以何种方式加入稳定算子会取的更好的效果；稳定算子在不同结构、不同参数以及不同深度的语言模型中的表现各是怎样的。实验结果如表 2 所示，其网络类型中的 LM 为普通语言模型，Multitask 为多任务模型，在本文中是用的语言模型任务加词性标注任务；隐层分为单层网络和多层网络，单层网络皆为 LSTM 网络，多层的分为全为 LSTM 网络和半 LSTM 半 DNN 网络，

其中在 Multitask 任务中，前面部分为两个任务公用隐层，后面的部分为每个任务各自独立的隐层。本文所有实验均在 nerv 上实现，采用的隐层大小为 300，训练总轮数固定 25，batch 大小为 20，chunk 大小为 15，不加分类训练及 dropout。Beta 类型为空即为不加入稳定算子。

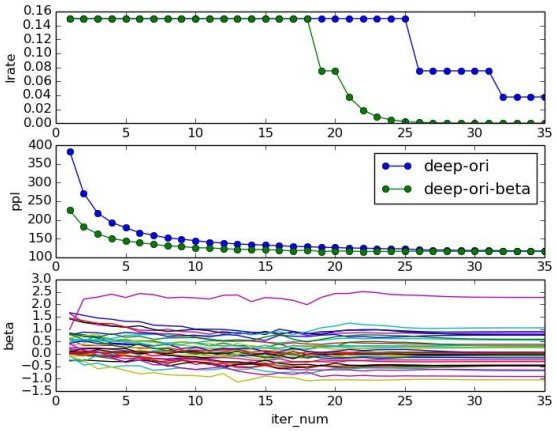
表 2　稳定算子在不同模型中的 PPL

实验编号	网络类型	Beta 类型	隐层参数	PPL
1	LM	-	1 lstm	103.3
2	LM	*Same*	1 lstm	107.3
3	LM	Diff	1 lstm	106.9
4	Multitask	-	1 lstm	102.2
5	Multitask	*Same*	1 lstm	106.1
6	Multitask	Diff	1 lstm	105.6
7	LM	-	1 lstm+2 dnn	111.8
8	LM	*Same*	1 lstm+2 dnn	111.2
9	LM	Diff	1 lstm+2 dnn	111.0
10	LM	Diff	3 lstm	139.0
11	Multitask	-	1 lstm+2 dnn	110.6
12	Multitask	*Same*	1 lstm+2 dnn	110.1
13	Multitask	Diff	1 lstm+2 dnn	109.8
14	Multitask	*Diff*	1 lstm+2lstm	131.9

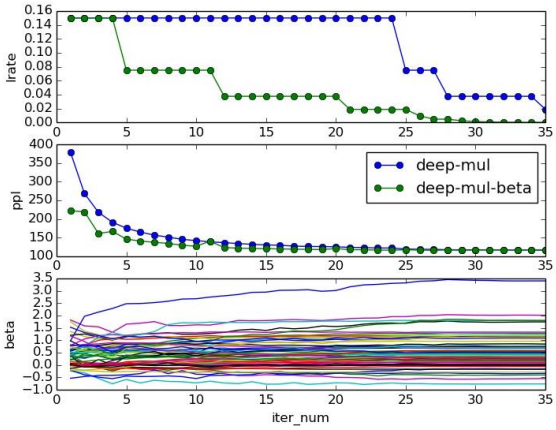
注：隐层参数的前部分为公共隐层，后部分为每个 task 各自的隐层。

通过对比实验 2,3 和对比实验 5,6 发现第二种稳定算子引入算法稍微优于第一种，即每个隐层的每个 LSTM 单元的每个门分别拥有各自独立的稳定算子 beta。对比实验 1,2 和 4,5 发现普通的稳定算子 beta 的引入算法对于单层 LSTM 语言模型和 Multitask 语言模型的提升没有帮助，反而有阻碍作用；然而对比实验 7,9 和实验 11,13 发现当网络结构加深的情况下，稳定算子开始起了作用，而本实验无法继续加深网络，因为 PTB 训练数据小，使得参数量大的网络无法训练至收敛，这也是 3 层的网络结构比单层网络结构差的原因。比较实验 10 和 9 以及实验 13 和 14 可以发现，因为 LSTM 参数量远大于 DNN 网络，使得 PTB 无法有效训练深层的 LSTM 语言模型。

为了进行更深入的分析，观察为何加入稳定算子的 PPL 没有进一步减小，以及针对性的进行后面的工作，我们将实验 7,9 和实验 11,13 这两组分别进行对比，比较他们的学习率（lrate）、混淆度（ppl）和每个门的 beta 数值在 35 轮中的变化情况，分别如图和图所示。



（a）实验 7、9：3 层普通 LM



（b）实验 11、13：3 层 Multitask

图 1　学习率、ppl 和 beta 随着轮数的变化曲线图

通过这两组对比实验，我们可以更直观的发现，加入稳定算子 beta 的模型，ppl 会更快速地收敛，以至于学习率下降得更快，到了一定范围内学习率几乎接近于 0 以至于后面的训练几乎不起作用，从而被不加稳定算子的模型接近甚至超越。另外，通过横向对比我们发现，多任务学习的 PPL 和学习率的收敛速度比单任务语言模型快，并且收敛后的 beta 的绝对值要更大。

为了探究加入稳定算子的 L2 范数是否能够提升语言模型，我们设计第三组实验，同样是在 PTB 数据上，我们控制对照组的模型结构和参数相同，一组加入原始的 beta，另一组在第一组的基础上加上 beta 的 L2 范数。实验结果如表 3。

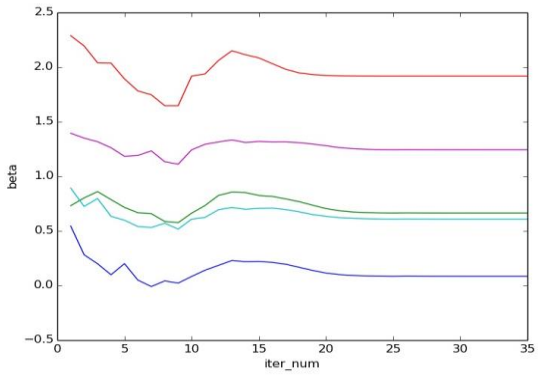
表 3　加入 beta-L2norm 的比较

网络类型	隐层参数	是否加入 beta 是否加入 L2norm	PPL
LM	1lstm	None	103.346
LM	1lstm	Only beta	107.316
LM	1lstm	Beta+L2norm	104.755
Multitask	1lstm	None	102.207

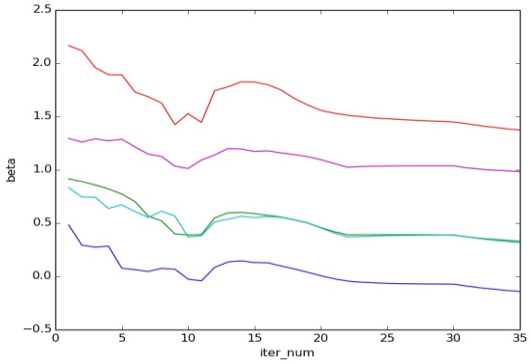
网络类型	隐层参数	是否加入 beta 是否加入 L2norm	PPL
Multitask	1lstm	Only beta	106.101
Multitask	1lstm	Beta+L2norm	103.115
LM	1 lstm+2 dnn	None	111.8
LM	1 lstm+2 dnn	Only beta	111.2
LM	1 lstm+2 dnn	Beta+L2norm	110.1
Multitask	1 lstm+2 dnn	None	110.6
Multitask	1 lstm+2 dnn	Only beta	110.1
Multitask	1 lstm+2 dnn	Beta+L2norm	109.0

注：

其中第 2,3 个实验的 beta 变化曲线图如图所示



（a）1 层 LSTM-LM 的 beta 变化曲线图



（b）1 层 LSTM-LM beta 加 L2notm 的 beta 变化曲线图

图 2　beta 变化曲线图

如表 3 所示，在代码实现过程中加入 beta 的 L2 范数优化后，语言模型的 PPL 结果有微小提升，大概 1%，符合实验预期。将每个门的 beta 参数的根据每轮训练的变化打印如图 2，可以发现加入 L2 范数的 beta 参数更小，符合上面 beta 越小会使 PPL 结果更好的猜想。

4　总结

在语言模型中引入稳定算子 beta 进行初始学习率的自适应能够很大幅度忽略初始学习率对结果的影响；同时，稳定算子能够较大幅度提升语言模型的收敛速度，有效减少不必要的训练时间；在多种 LSTM 语言模型引入稳定算子的算法中，每个隐层的参数分别拥有自己独立的稳定算子会使结果更优。

稳定算子的引入对于普通语言模型和多任务语言模型来看在语言模型的性能上没有明显区别，并不像实验前预期的那样或许能智能调整不同任务的学习速度从而提升最后的训练结果，根据分析发现，这是因为多任务模型中为不同的任务分配权重已经可以达到不同任务学习速度不同的要求。另一方面，在参数量合理、没有超出训练数据的收敛极限的情况下，越深的网络，引入稳定算子的结果越好。

引入稳定算子的同时，在训练过程中对其加入 L2 正则优化会略微提升语言模型的性能，提升幅度大概 1%，使 beta 收敛在一个比较小的值。

今后的工作中，会围绕稳定算子如何优化以达到提升性能的目的做下一步研究。

参 考 文 献

[1] Slava Katz, “Estimation of probabilities from sparse data for the language model component of a speech recognizer,” IEEE transactions on acoustics, speech, and signal processing, vol. 35, no. 3, pp. 400–401, 1987.

[2] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur, “Recurrent neural network based language model.,” in Interspeech, 2010, vol. 2, p. 3.

[3] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur, “Extensions of recurrent neural network language model,” in 2011 IEEE International Conference on Acoustics, Speech and Signal.

[4] Sepp Hochreiter and Jrgen Schmidhuber, “Long shortterm memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] Martin Sundermeyer, Ralf Schl¨uter, and Hermann Ney,“Lstm neural networks for language modeling.” in Interspeech,2012, pp. 194–197.

[6] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, “Efficient backprop,” in Neural Networks: Tricks of the Trade, pp. 9–50, Springer Berlin Heidelberg, 1998.

[7] Konecny J, Liu J, Richtarik P, et al. Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting[J]. IEEE Journal of Selected Topics in Signal Processing, 2014, 10(2):242-255.

[8] P. Ghahremani and J. Droppo, “Self-stabilized deep neural network,” in IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 5450–5454, 2016.

[9] Qi Liu, Tian Tan and Kai Yu. An Investigation on Deep Learning with Beta Stabilizer. 13th IEEE International Conference on Signal Processing (ICSP), Chengdu, China, 2016: 557-561.

[10] Y. Yao, L. Rosasco, and A. Caponnetto, “On early stopping in gradient descent learning,” Constructive Approximation, vol. 26, no. 2, pp. 289–315, 2007.

[11] I. Goodfellow, Y. Bengio, and A. Courville, “Deep learning.” Book in preparation for MIT Press, 2016.