神经网络实验报告

——基于循环神经网络的语言模型

Aevilorz (1010110)　　2nd author (12345)

May 5, 2017

Abstract

这份报告介绍了如何使用循环神经网络进行语言建模，并探究了网络超参数对模型性能的影响。

我们使用了一个小的数据集进行初步实验，训练的模型都存在过拟合的现象，因此在充足数据条件下的模型性能还有待探究.

Contents

1 语言模型

语言模型（*Language Model*）是单词序列的概率分布模型[1]。假设有一个单词集合 $V = \{w_1, w_2, \cdots\}$，将任意多个单词连接起来构成单词序列 $S = w_1 w_2 \cdots w_t$, 其对应的概率为 $Prob(S)$，所有合法的序列构成语言 L，L 序列的概率分布模型 $Prob(L)$ 就称为语言模型。

$$\begin{aligned} Prob(S) &= Prob(w_1 w_2 \cdots w_t) \\ &= Prob(w_1 w_2 \cdots w_{t-1}) Prob(w_t | w_1 w_2 \cdots w_{t-1}) \end{aligned}$$

语言模型的一个最简单的功能就是是在给出序列的前文 $w_1 w_2 \cdots w_{t-1}$ 时，预测下一个单词 w_t[2,3]。由于受到计算资源和存储资源的限制，实现时主要采用 *N-Gram* 语言模型，即只根据前 *N-1* 个单词来预测下一个单词。

2 循环神经网络

与以往的神经网络不同，循环神经网络（*Recurrent Neural Network, RNN*）能够很好的处理序列数据、对序列进行建模。这主要得益于它的循环层（*Recurrent Layer*）：前一时刻隐层的状态作为后一时刻隐层的输入的一部分。这样的结构使得 *RNN* 能够充分利用序列的上下文（*Context*）进行学习。

2.1 网络结构

使用 RNN 进行语言模型建模时，通常使用三种不同类型的神经元层：词嵌入层（*Embedding Layer*）、循环层（*Recurrent Layer*）和分类层（*Softmax Layer*）。整体的网络结构如图 (1)。

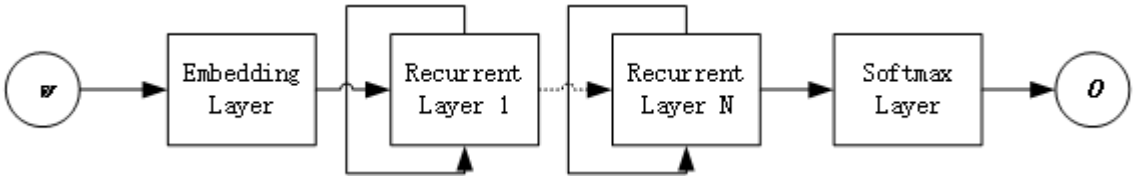

Figure 1: The structure of RNN Language Model.

2.1.1 词嵌入层

在使用神经网络进行语言建模前，需要将词表 V 中的单词转化为神经网络能够处理的向量。一种直观的转化方法是使用 one hot 向量。这种方法将 V 中的第 i 个单词 w_i 映射到一个维度为 $|V|$ 的 0/1 向量 $onehot(w_i)$，其第 i 个维度值为 1，其余为 0。但是这样的表示存在两个问题：一是在 $|V|$ 很大的时候会造成维度灾难[4]；二是在两个意思相近的词之间存在语义鸿沟，即它们对应的词向量毫无相似度可言。

[1] 漫谈 Language Model(1): 原理篇
[2] Recurrent neural network based language model, Mikolov, 2010
[3] Wikipedia: Continuous space language models
[4] A neural probabilistic language model, Bengio, 2003

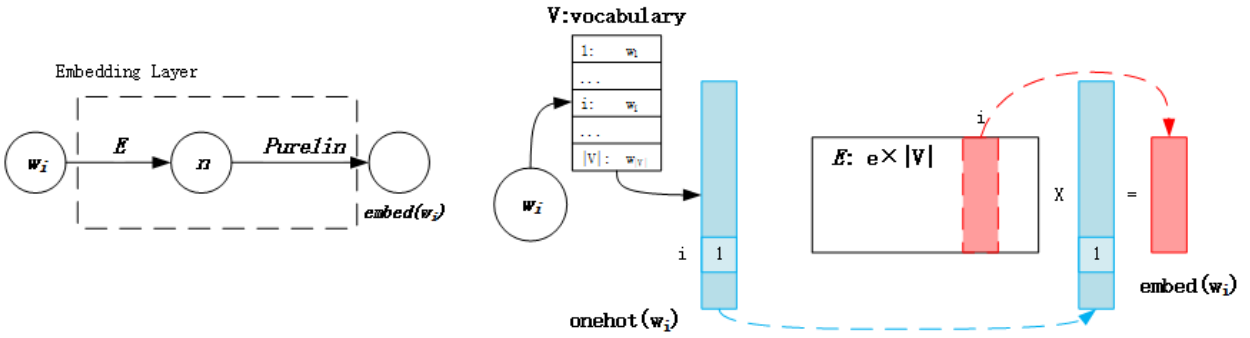

Figure 2: The structure and working principle of the word embedding layer.

目前较为流行的做法是采用词嵌入（*Word Embedding*）将单词 w_i 映射到维度 e 远小于词表大小 $|V|$ 的实值向量 $embed(w_i)$，这种表示方法能很好的解决上述问题。词嵌入层的作用正在于此，它有 $e(e \ll |V|)$ 个神经元，它有一个维度为 $e \times |V|$ 的权值矩阵 E, 没有偏置值，采用线性激活函数。它的处理过程是在接收到输入单词 w_i 后，首先将其转化为 onehot 向量 $onehot(w_i)$，然后与权值矩阵 E 相乘得到词向量 $embed(w_i)$，这个过程可以简化为输入单词 w_i 的索引 i，然后从 E 中取出第 i 列，如公式 (1)。

$$\begin{aligned} \boldsymbol{embed}(w_i) &= \boldsymbol{E} \cdot \boldsymbol{onehot}(w_i) \\ &= \boldsymbol{E}[:, i] \end{aligned} \tag{1}$$

2.1.2 循环层

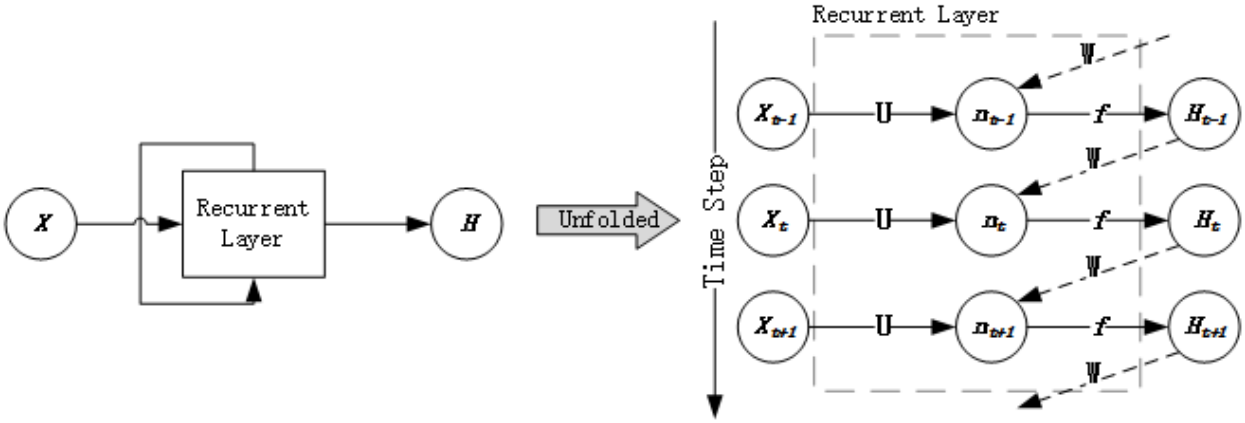

Figure 3: The structure of the recurrent layer.

循环层可简化成如图 (3) 左半部分所示。对于一个输入序列 $X_1X_2 \cdots X_tX_{t+1} \cdots$，将循环层按时间展开，如图 (3) 右半部分所示。在第 t 时刻，循环层接受输入向量 X_t 和 $t-1$ 时刻的循环层的输出 H_{t-1}，然后输出 H_t。具体过程是：X_t 和 H_{t-1} 分别与各自的权值矩阵 U 和 W 相乘后相加，得到循环层各神经元的中间输出 n_t，然后经过循环层的激活函数 f 输出当前循环层的状态 H_t，如公式 (2,3)：

$$\boldsymbol{n_t} = \boldsymbol{UX_t} + \boldsymbol{WH_{t-1}} \quad (2)$$
$$\boldsymbol{H_t} = f(\boldsymbol{n_t}) \quad (3)$$

公式 (2,3) 所描述的是最朴素的循环层结构，这种结构在对长序列进行学习时会存在梯度消失或梯度爆炸的问题。目前解决这个问题较为流行的方法是采用长短时机制，**LSTM**（*Long Short Term Memory*）和 **GRU**（*Gated Recurrent Unit*）循环层结构是长短时机制的两个典型。

2.1.3 分类层

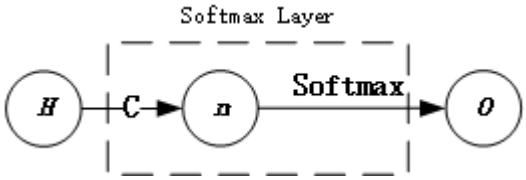

Figure 4: The structure of the softmax layer.

分类层有 $|V|$ 个神经元，其权值矩阵为 C，采用 $Softmax$ 函数作为激活函数，其作用就是将最后一个循环层的输出 H 转化为词的概率分布向量 O，如公式 (4,5,6)。最后选择 O 中概率最大的词作为该语言模型的预测结果，如公式 (7,8)。

$$Softmax(\boldsymbol{X}) = \frac{e^{\boldsymbol{X}}}{\sum e^{\boldsymbol{X}[i]}} \quad (4)$$
$$\boldsymbol{n} = \boldsymbol{CH} \quad (5)$$
$$\boldsymbol{O} = Softmax(\boldsymbol{n}) \quad (6)$$
$$i^* = \underset{i}{argmax}(\boldsymbol{O}) \quad (7)$$
$$w_p = w_{i^*} \quad (8)$$

2.2 目标函数

对于长度为 l 的训练样本序列 $S = w_1 w_2 \cdots w_l$，在第 $t(1 \le t \le l-1)$ 时刻，RNN 的输入序列为 $S[1:t] = w_1 w_2 \cdots w_t$，输出为 O_t，预测 $t+1$ 时刻的单词为 w_{p_t}，整个过程如图 (5)：

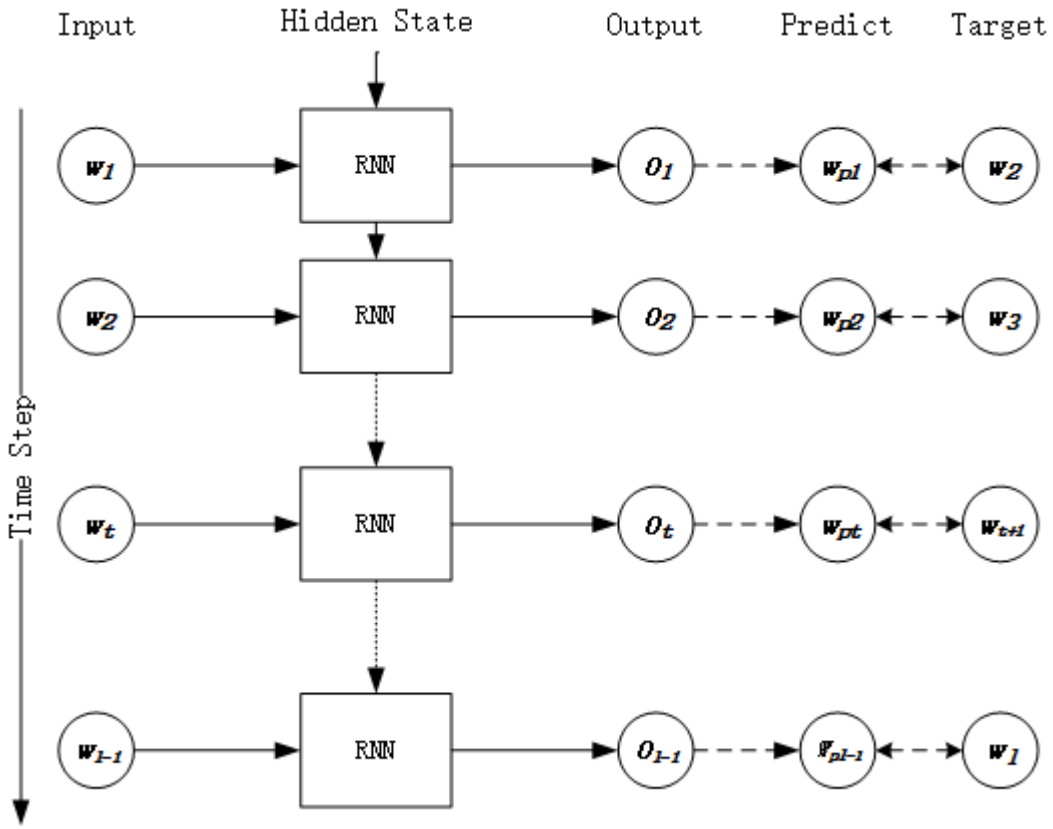

Figure 5: The rnn's prediction of a sentence instance.

训练网络的目标是使得网络的预测序列 $S_p = w_{p_1} w_{p_2} \cdots w_{p_{l-1}}$ 尽可能地接近目标序列 $T = w_2 w_3 \cdots w_l$。这是一个多分类问题，通常采用交叉熵（*Cross Entropy*）来描述预测与目标之间的接近程度。记 *RNN* 的输入序列为 $O = O_1 O_2 \cdots O_{l-1}$ 训练网络的目标函数 J 如公式 (9)。

$$\begin{aligned} J &= XEntropy(T, \boldsymbol{O}) \\ &= \frac{1}{l-1} \sum_{t=1}^{l-1} XEntropy(T[t], \boldsymbol{O}[t]) \\ &= \frac{-1}{l-1} \sum_{t=1}^{l-1} onehot(T[t]) \cdot \log(\boldsymbol{O}[t]) \end{aligned} \tag{9}$$

2.3 学习规则

RNN 主要还是通过梯度下降（*Gradient Descent, GD*）的方法更新各层的权值矩阵，第 k 次迭代时，任意权值矩阵 M 的更新公式如 (10,11)。

$$\boldsymbol{M}(k+1) = \boldsymbol{M}(k) - \alpha \nabla_{\boldsymbol{M}} J \tag{10}$$

$$\nabla_{\boldsymbol{M}} J = \frac{\partial J}{\partial \boldsymbol{M}} \tag{11}$$

RNN 训练网络的算法称为 **BPTT**（*Back Propagration Through Time*），*BPTT* 的原理[5]和 *BP* 算法大致相同，都是先利用求导的链式法则计算灵敏度 $\delta = \frac{\partial J}{\partial n}$，再通过灵敏度计算梯度。稍有区别的是，*BPTT* 在计算 t 时刻循环层的梯度时，还需要将灵敏度沿时间反向传播至 $t-1, t-2, \cdots, 1$ 时刻，如图 (6)，再计算各时刻的梯度之和作为 t 时刻的梯度。

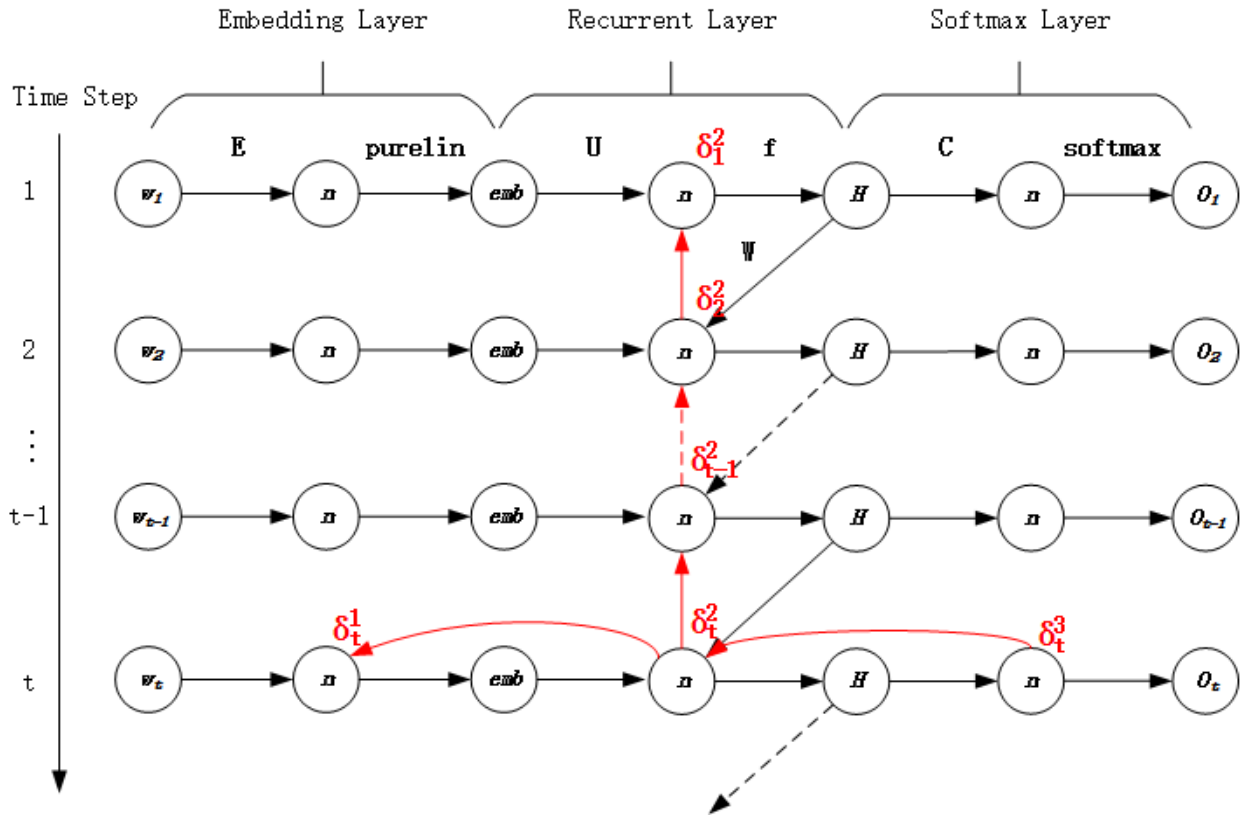

Figure 6: a demostration of BPTT.

3 实验与评估

3.1 实验环境

本实验在一台 CPU 为 *Intel Corei5-4200U*（双核四线程），内存为 8GB，操作系统为 *Windows10* 的笔记本上进行。众所周知，训练出一个深度神经网络需要巨大时间开销，为了较快速地获得实验结果，本实验使用 *Python3.5+TensorFlow* 进行编程。*TensorFlow* 是目前最受欢迎的深度学习开发平台之一，它支持多线程、异构计算等优化技术，因此能够快速地训练网络。

3.2 实验数据

本实验使用的数据为英文儿童读物《The Little Prince》。该数据集一共有 1664 个训练样本（句子），词表大小为 2230，一些统计信息如下表：

[5]RECURRENT NEURAL NETWORK TUTORAL

项目	值
数据集	the little prince
样本数	1664
词表大小	2230
样本最小长度	4
样本最大长度	103
样本平均长度	14
样本长度中位数	12

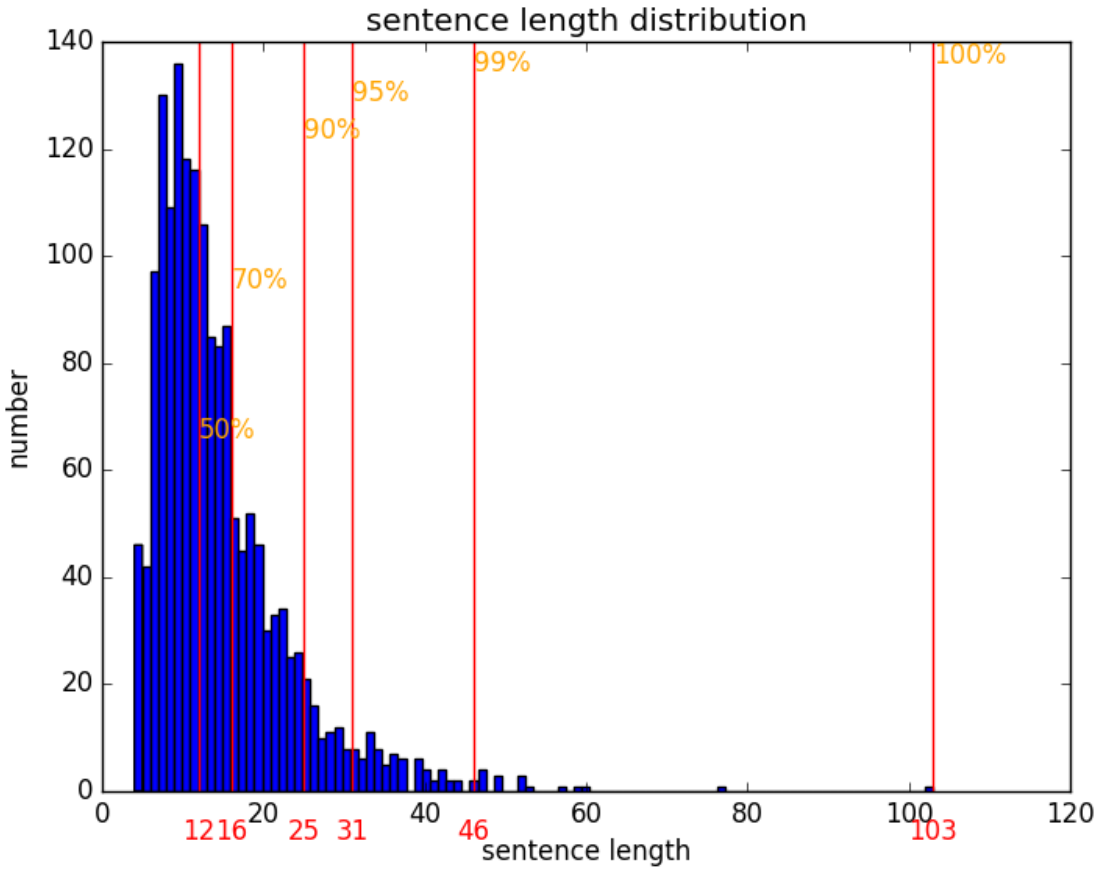

Figure 7: The distribution of the length of sentences in dataset.

该数据集的样本序列长度分布如图 (7)，99% 的序列长度在 46 以下，95% 的序列长度在 31 以下，90% 的序列在 25 以下，70% 的序列长度在 16 以下，50% 的序列长度在 12 以下。

3.3 实验方法

3.3.1 预处理

原始的数据集是一行一段的英文文本，我们需要将其转化为 *RNN* 能够接收的句子，实验中不考虑句子是否为对话内容，不考虑英文大小写。

首先，删除段落中的无效字符\u3000 和"，再将所有大写英文字符转化为小写，然后使用 **nltk** 工具包（*Natural Language Tookit*）将段落分割成句子，加入到句子集合中：

```
sentences = []
for line in f:
    paragraph_i = nltk.sent_tokenize(line.replace('\u3000', '').replace("\"","").lower())
    sentences += paragraph_i
```

然后，在每条句子的首尾分别加上开始和结束标识符 **SST** 和 **SET**，并将句子转化为单词序列：

```
sentences = ["%s %s %s" % (sentence_start_token, x, sentence_end_token) for x in sentences]
tokenized_sentences = [nltk.word_tokenize(sent) for sent in sentences]
```

最后，统计各单词的频率，创建词表并将句子中的单词映射到词表中对应的索引，得到 *RNN* 能够接收的句子集合 *idx_sentences*：

```
word_freq = nltk.FreqDist(itertools.chain.from_iterable(tokenized_sentences))
vocab = word_freq.most_common(vocabulary_size-1)
...
idx_sentences = [ [word_to_index[w] for w in sent] for sent in tokenized_sentences]
```

实验将 *idx_sentences* 划分为训练集和验证集，其中训练集占 70%，验证集占 30%。

3.3.2 训练 *RNN*

实验训练了多个 3 层的 *RNN*，其中第 1 层为词嵌入层，其神经元数量为 128；第 2 和第 3 层为 *GRU* 循环层，其神经元数量、使用的激活函数根据实验需要设定；第 4 层为分类层，其神经元数量为词表大小，即 2230。

训练采用 **Adam**（*Adaptive moment estimation*）算法作为参数（权值）更新算法，*Adam* 是基于 *GD* 的一种优化算法，它根据参数的一阶矩估计和二阶矩估计动态调整学习率[6]。同时实验采用小批量（*minibatch*）更新的方式，即一次性输入 b 条句子后更新一次参数，b 的取值也是根据实验需要设定。此外 **dropout**[7]正则化技术也将应用在训练过程中，*dropout* 通过随机保留一定比例的神经元进行训练来减小过拟合现象的产生，训练好网络之后所有的神经元都将用于预测。

在训练过程中，主要通过 *RNN* 在当前 *minibatch* 和验证集上的平均损失值（即目标函数值，*loss*）与准确率（*acc*）来衡量训练效果，实验中每 20 个 *minibatch* 计算并记录一次 *loss* 和 *acc*。

3.3.3 超参数选择

实验以 *epoch* 为 111，*minibatch* 大小为 32，*dropout* 保留率为 80%，循环层神经元数量为 100，循环层激活函数为 *elu* 的 *RNN* 训练的过程为基准，采用控制变量法，选择不同的 *minibatch* 大小、*dropout* 保留率、循环层神经元数量和激活函数，来探究这些超参数对 *RNN* 训练时间、收敛速度、收敛表现以及过拟合程度的影响，这些参数的选取如下表，激活函数定义及形状如图 (8)。

参数	取值范围
minibatch	16，32，48
dropout 保留率	1.0，0.9，0.8，0.7
神经元数量	50，100，200
激活函数	tanh，relu，elu

[6]Adam, Kingma, 2015
[7]Dropout, Zaremba, 2015

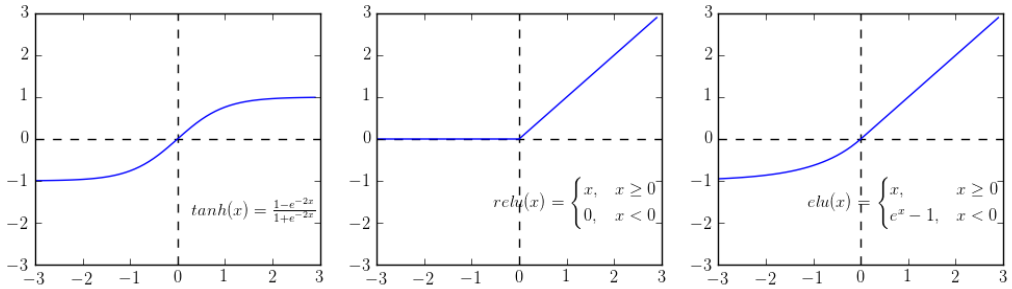

Figure 8: The shape of tanh, relu and elu.

3.3.4 其他

除此之外，为了更直观的展示 *RNN* 对语言模型的学习效果，训练过程中每 40 个 *minibatch* 从整个训练集中抽取 10 条句子进行预测，并打印预测值与真实值的对比结果；每 60 个 *minibatch* 打印 5 条由 *RNN* 自动生成的句子。为了能够使用训练好的模型以及能够间断性训练，每 80 个 *minibatch* 保存一次模型，保存的模型将根据训练时的准确率来命名，因此实验还能够展示在不同的训练阶段模型的表现。

3.4 实验结果与分析

3.4.1 基准模型的表现

在训练过程中，基准模型的损失值和准确率与迭代次数的关系曲线如图 (9) 所示。

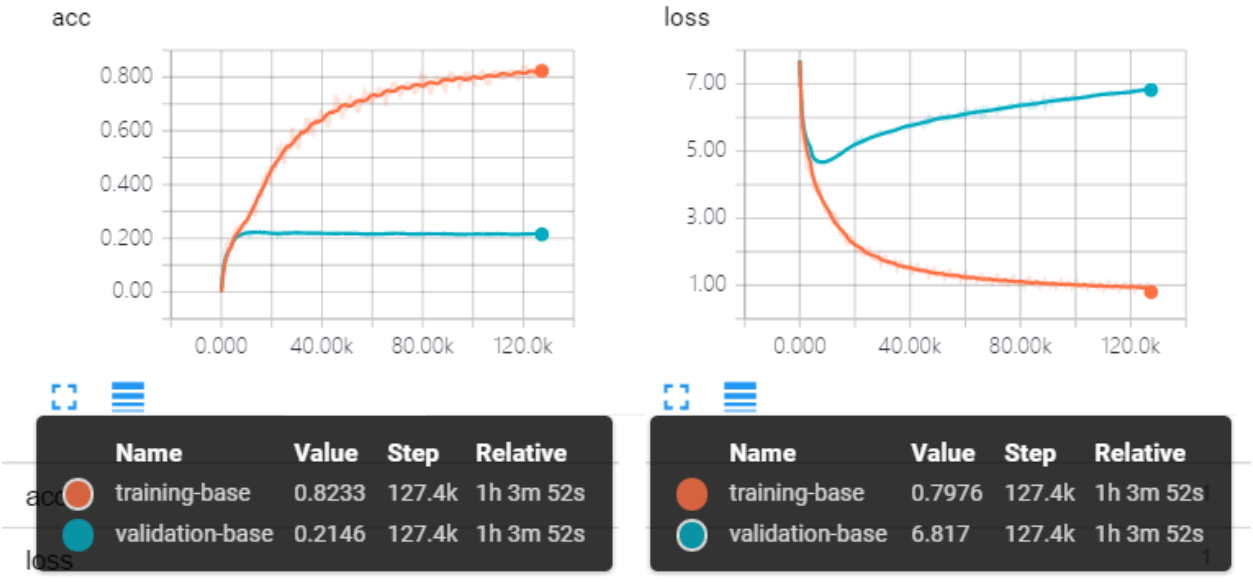

Figure 9: The performance of the basic model VS step.

从图 (9) 可以观察到：首先，在训练集（*minibatch*）上的损失值曲线呈先下降后稳定的的趋势，最终稳定在 0.8 附近，而在验证集上的损失值曲线呈先下降后上升的趋势；其次，准确率曲线呈先上升后稳定的趋势，训练集

上的曲线稳定在 82% 附近，验证集上的曲线稳定在 21% 附近。由训练集上的损失值曲线和准确率曲线可知，对模型的训练时收敛的，而且在训练集上的学习效果还不错；但结合验证集上曲线走势可知，实际上在训练模型的过程产生了过拟合（*overfitting*）现象。

Figure 10: The reasons of overfitting.

产生过拟合的原因可能有来自模型或数据。如果神经网络复杂度高，则网络拥有强大的学习能力，能够学习训练集上细微的特征，从而导致模型在验证集上的表现差；如果数据不够多，不足以代表整个语言模型，则网络也只能学习到训练集上语言的分布情况，从而在验证集上表现差。解决过拟合的方法通常为使用正则化技术（*regularization*），实验曾尝试使用 **dropout**[8]正则化技术和降低网络复杂度，但仍不能解决过拟合问题。另外，本实验所用的数据集规模要远小于一般自然语言处理任务的数据集规模，因此，我推断本实验中过拟合现象是由于数据造成的。

3.4.2 预测效果与自动生成句子

在训练模型的收敛阶段，模型的预测值与实际值对比如图 (11)，对一个句子，上方显示的是实际值，下方显示的是预测值。从图中可以观察到：**1**. 对于训练集中的句子，模型的预测大部分都是正确的，但是句子开头的几个词很少能够预测正确；**2**. 对于验证集中的句子，模型的预测大部分都是错误的，但是一些简单的、词频较高的词（如 **i**、**said**、**that**、**am** 和 **.** 等）能够被正确预测。这一方面验证了前面所提到的过拟合现象，另一方面也说明，在没有足够上下文信息的情况下，一个好的模型（相对于训练集）也不能够进行准确的预测。

[8]Dropout, Zaremba, 2015

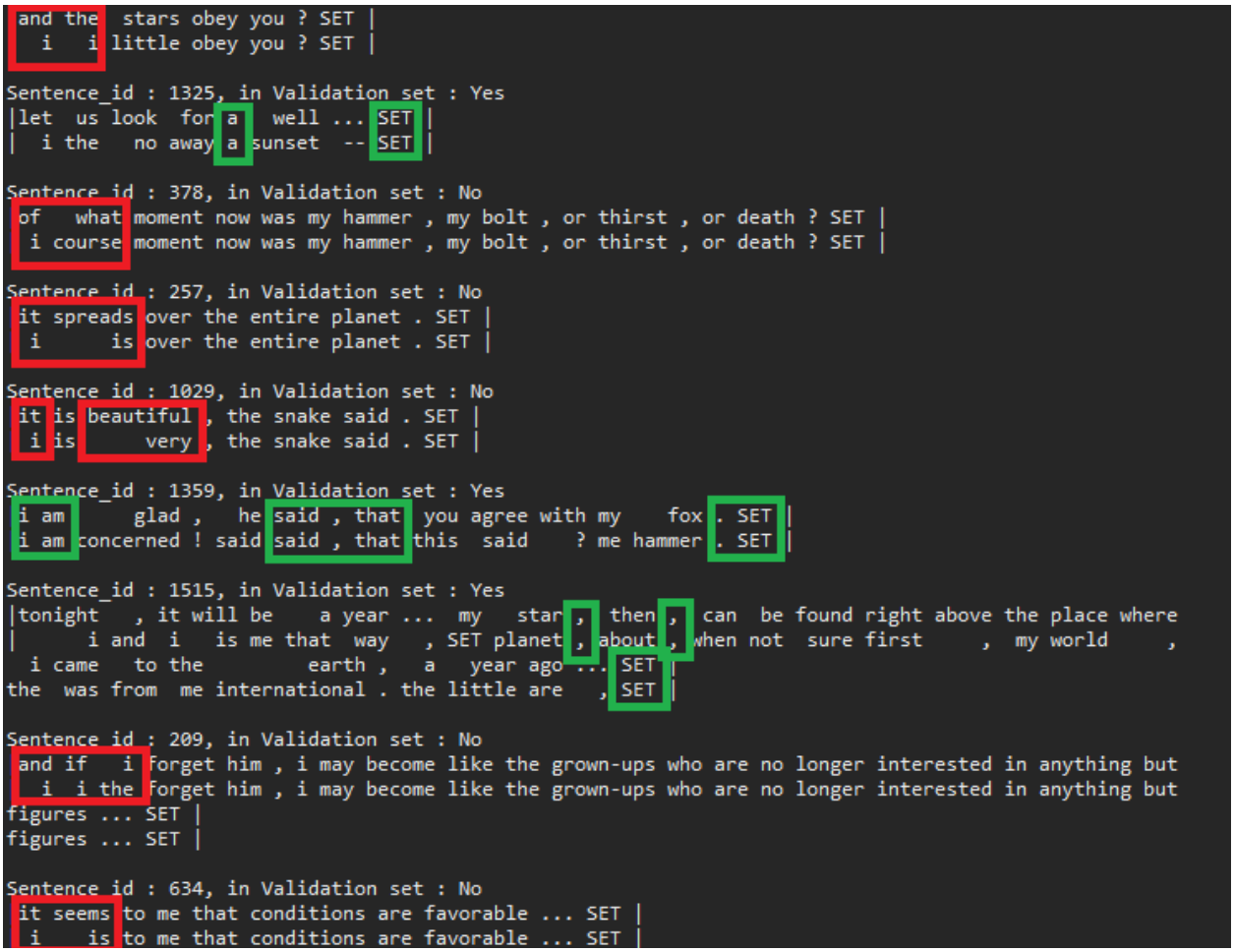

Figure 11: Some predictions from trained rnn language model.

在训练模型的收敛阶段，通过将 ***SST*** 输入模型，模型预测的词作为其下一个输入，直到输出 ***SET*** 或达到设定的最大句子长度为止的方法，使模型自动生成句子，其效果如图 (12) 所示。从图中可以看出，模型生成的句子基本通顺且有一定的含义。

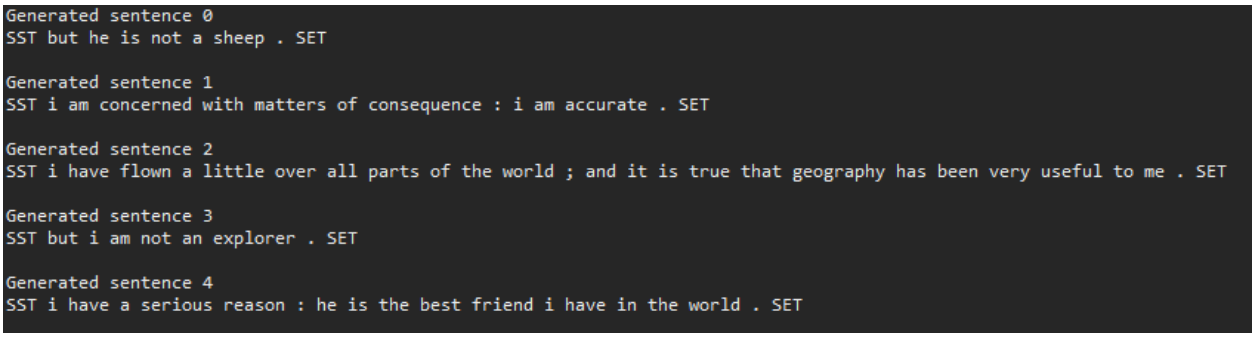

Figure 12: Some sentences generated by trained rnn language model.

3.4.3 *minibatch* 大小对模型的影响

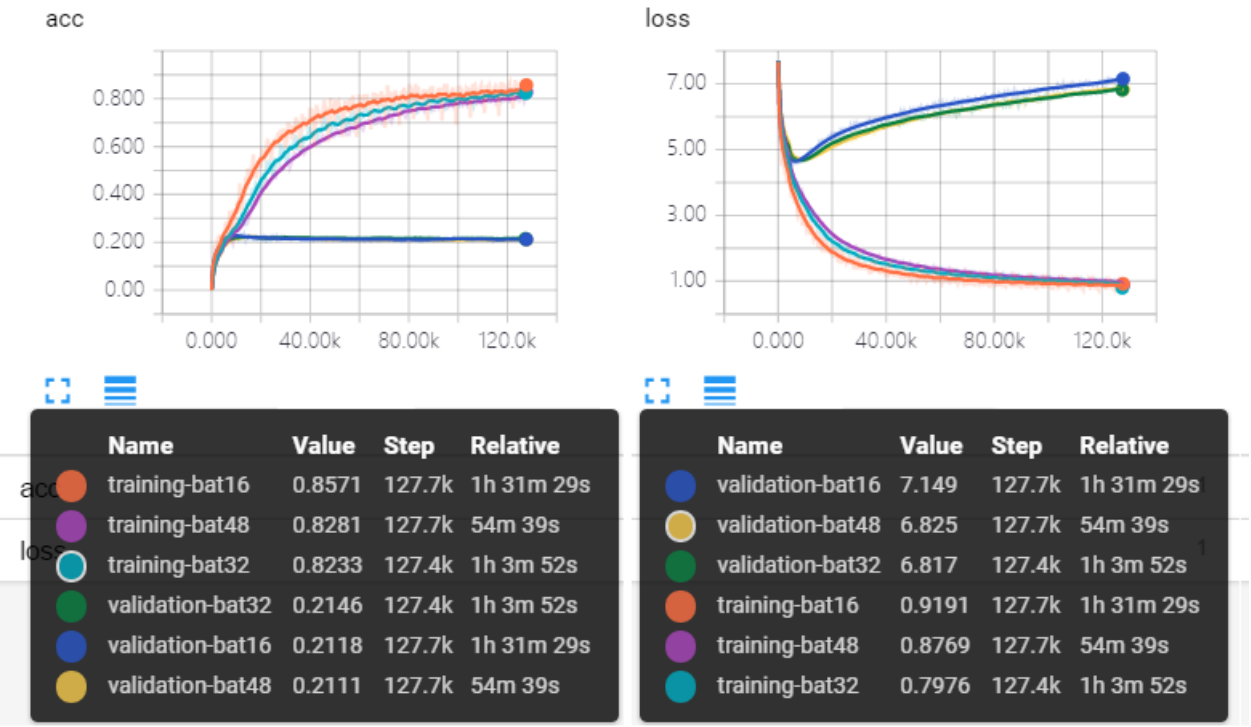

Figure 13: The learning curves with different batch sizes.

使用不同大小的 *minibatch* 时，模型准确率与损失值曲线如图 (13)。首先，在学习样本数（*Step*）相同的情况下，花费的训练时间随着 *minibatch* 的增大而减小：使用大小为 16、32 和 48 的 *minibatch* 学习 127.4k 个样本时分别需要 1 小时 31 分、1 小时 4 分和 54 分；其次，*minibatch* 大小对收敛速度影响不明显：三者几乎都在学习 120k 个样本后收敛，而 *minibatch* 小的收敛得更快一些；最后，*minibatch* 大小对过拟合现象的影响不明显：尽管 *minibatch* 大小不同，三者损失值曲线都很接近。

在学习相同数量训练样本的情况下，*minibatch* 越小迭代更新网络参数的次数越多，因此能够较快的收敛，但是更新参数需要计算梯度，其中包含了许多矩阵运算，因此小的 *minibatch* 花费的时间也就越多。总的来看，*minibatch* 大小主要还是影响模型的训练时间，过小的 *minibatch* 将导致训练过程十分漫长。

3.4.4 *dropout* 保留率对模型的影响

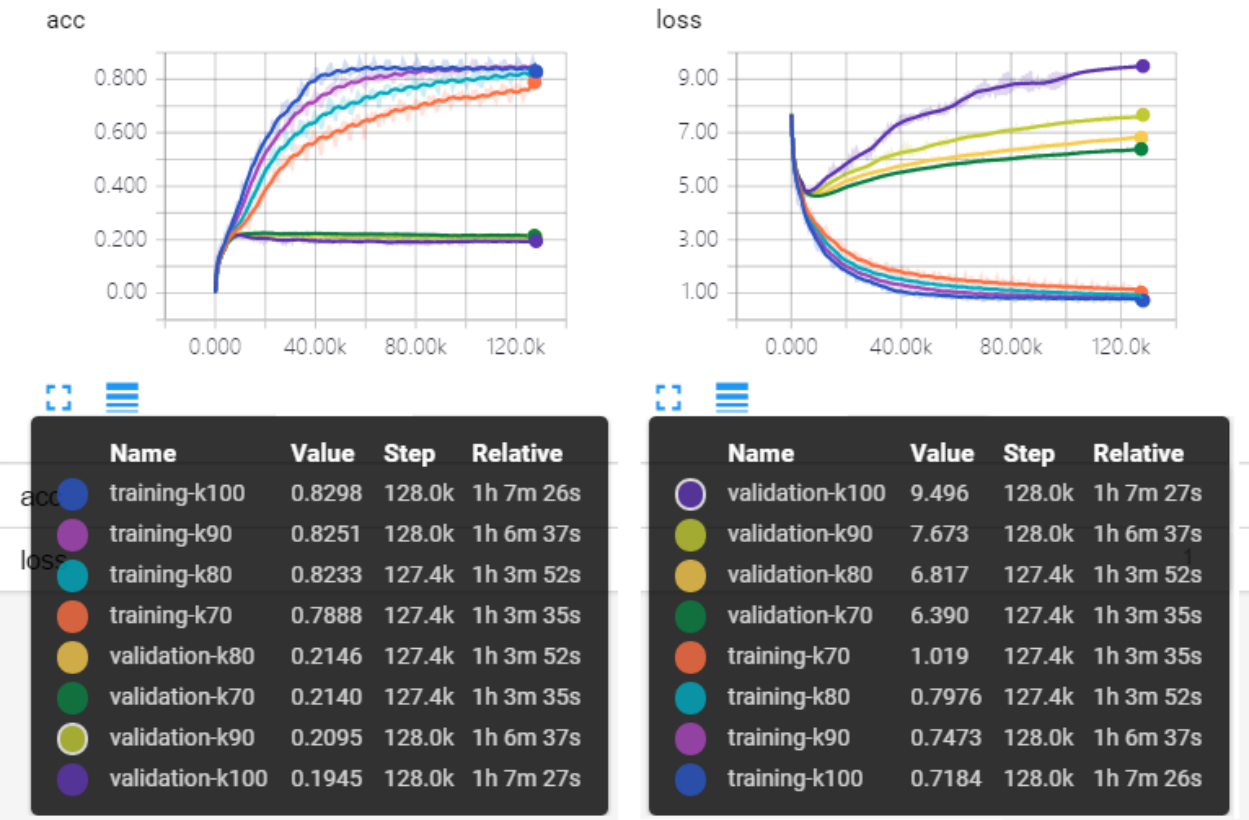

Figure 14: The learning curves with different keep probabilities of dropout.

在不同 *dropout* 保留率下，模型准确率与损失值曲线如图 (14)。首先，不同的保留率对训练时间没有影响：学习样本数量（*Step*）相同（$120k$）时，花费的时间（*Relative*）基本都在 1 小时左右；其次，在处理相同数量样本的情况下，保留率越低，收敛速度越慢：保留率为 100%、90% 和 80% 的情况下，模型分别在 *Step* 为 $40k$、$50k$ 和 $120k$ 的时候收敛，此时模型的在训练集上的准确率大约为 82%，损失值大约为 0.75，保留率为 70% 的情况下，*Step* 达到 $120k$ 时仍未收敛；最后，过拟合程度随保留率的降低而减小：在验证集上的损失曲线从 *Step* 约为 $10k$ 时开始上升，曲线斜率随保留率的降低而减小，其减小程度在保留率从 100% 下降到 90% 时最为显著，从 80% 下降到 70% 时最不显著。

总的来说，虽然在实验中 *dropout* 没有能够消除过拟合现象，但是它还是一个行之有效的正则化技术，保留率越低，正则化的效果越明显。但正则化效果可能存在一个下界，过低的保留率不但不会提升正则化的效果，而且还会增加模型收敛所用的迭代次数。

3.4.5 循环层神经元数量对模型的影响

Figure 15: The learning curves with different sizes of the recurrent layer.

使用不同大小的循环层时，模型准确率和损失值曲线如图 (15)。首先，在学习样本数（*Step*）相同的情况下，花费的训练时间随着循环层神经元数量的增多而增大：采用神经元数量为 50、100 和 200 的循环层，学习 127.4k 个样本时分别需要 1 小时 1 分、1 小时 4 分和 1 小时 20 分；其次，循环层神经元数量越多收敛速度越快且准确率越高：在训练集上，循环层神经元数量为 200 时，模型在学习约 40k 个训练样本后就收敛到了 84% 准确率，循环层神经元数量为 100 时，模型在学习了约 120k 个样本时，达到了 82% 的准确率，而循环层神经元数量为 50 时，模型学习了约 120k 个样本后，只达到了 62% 的准确率；最后，过拟合程度随循环层神经元数量的增多而增强。

神经元数量越多，模型的复杂度越高，意味着学习的能力也就越强，同时在训练时需要更新的参数也就越多，学习相同数量的样本所需的时间开销也就越大。另一方面，过强的学习能力能够学习到训练集中的细微的特征，因此可能会产生过拟合现象。这部分实验使用减少循环层神经元的方法降低模型复杂度，但不仅没能够消除过拟合现象，还使得模型在训练集上的表现下降。根据这一情况，我推断实验中过拟合的原因来自于数据的质量：数据集规模过小，从数据集中随机抽取的训练集不足以代表整个数据集，因此训练出的模型在训练集上效果好，在验证集上效果差。

3.4.6 循环层激活函数对模型的影响

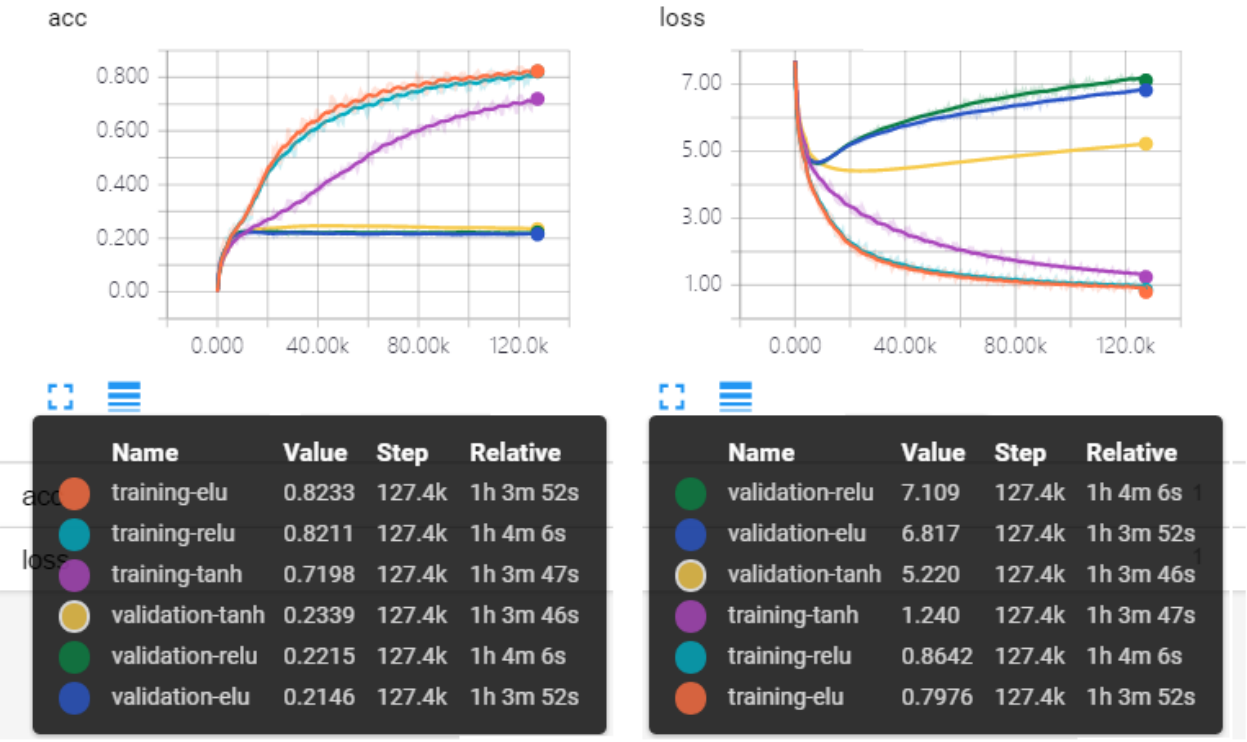

Figure 16: The learning curves using different activation function of the recurrent layer.

使用不同激活函数的循环层时，模型准确率和损失值曲线如图 (16)。首先，使用 *tanh*、*relu* 和 *elu* 作为循环层的激活函数对训练的时间没有影响：学习 127.4k 的样本，用时均为 1 小时 4 分左右；其次，*elu* 和 *relu* 有着几乎相同的收敛速度，前者稍快，在训练集上 *elu* 在 127.4k 时准确率达到了 82.3%，*relu* 在 127.4k 时准确率达到了 82.1%，*tanh* 的收敛速度则要比前两者慢，在 127.4k 时准确率只有 71.9%；最后，过拟合程度从小到大依次是 *tanh*、*elu* 和 *relu*，其中 *tanh* 的过拟合程度要明显小于后两者，后两者的过拟合程度差别不是很大。

4 总结与展望

通过本次实验，首先，我掌握了 *RNN* 的基本结构、工作原理以及对 *RNN* 学习规则的推导，同时也巩固了高等数学知识；其次，我学习并了解了自然语言处理的一些基本知识，如词嵌入、语言模型等；然后，我初步学会了如何在 TensorFlow 平台上搭建，训练和使用神经网络，并将其运用到自然语言处理任务中；最后，通过对实验结果的分析，我对配置（超参数）与神经网络表现的影响关系有了一定的认识，并对机器学习中数据的重要性有更切身的感受。

本次实验受于时间和实验环境的限制，没有选择较大规模的数据集，在未来的工作中若条件允许我将继续探索。如今网络中存在的丰富的数据流，尝试使用 *RNN* 对网络流行为建模也将是我未来工作的一个方向。