山东大学

<<自然语言处理>>

实验四 神经机器翻译

姓名：	霍超凡
学号：	
学院：	软件学院
教师：	孙宇清

一、实验目标

本次实验探讨了两个关键概念-子词（subword）建模和卷积网络-并将它们应用于我们在上一实验中构建的 NMT 系统。实验三 NMT 模型可以看作四个阶段：

1. 嵌入层：将原始输入文本（源语句和目标语句）转换为词向量序列。
2. 编码器：将源语句编码为编码器隐藏状态序列的 RNN。
3. 解码器：在目标语句上操作并借助编码器隐藏状态以产生解码器隐藏状态序列的 RNN。
4. 输出预测层：具有 softmax 的线性层，在每个解码器时间步上产生下一个目标字的概率分布。

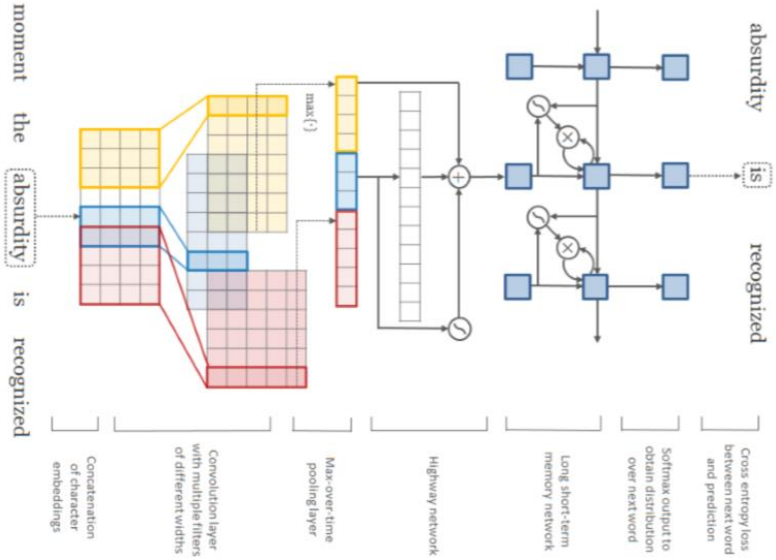
所有这四个子部分都在单词层次上模拟了 NMT 问题。在本实验的第一节中，我们将用基于字符的卷积编码器取代（1），在第二节中，我们将通过添加基于字符的 LSTM 解码器来增强（4）。这将有希望提高我们的 BLEU 测试集的性能！最后，在第三节中，我们将检查字符级编码器产生的字嵌入，并分析新的 NMT 系统的一些错误。

二、实验环境

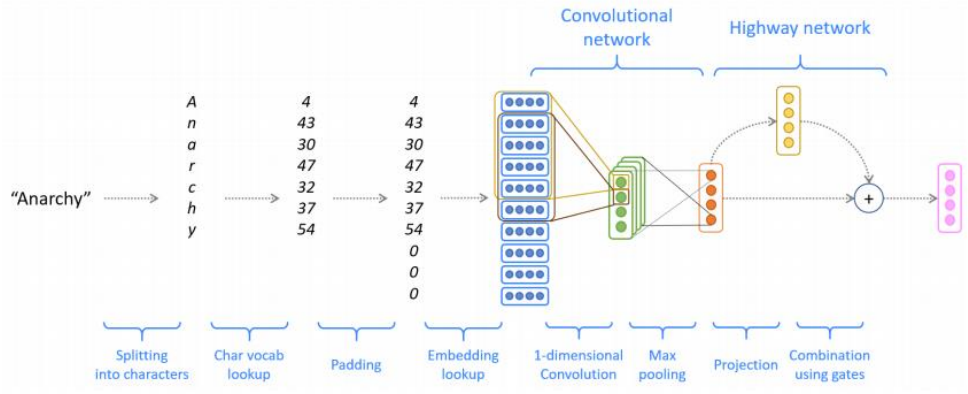
1. 开发环境
   实验使用 python 编程语言，在 python 环境中编写代码。
2. 运行环境
   实验在 Ubuntu 系统中，使用 Pytorch 框架，在 NVIDA 显卡中训练神经网络。

三、原理和模型论述

实验使用文章《Character-Aware Neural Language Models》所提出的基于字符感知的神经网络机器翻译模型，该模型分成两个部分，第一部分是字符级卷积编码器，第二部分是字符级 LSTM 解码器，其网络框架如下图所示，



字符卷积编码部分将单词转化为词向量，使用了字符级别的卷积，其具体结构如下：



给定单词x，通过查找每个字符的整数索引，可以将长度为 1 的单词x表示为整数向量

$$\mathrm{x} = [\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_\mathrm{l}] \in \mathbb{Z}^\mathrm{l}$$

其中每个c_l是字符表中的整数索引，使用一个特殊的<PAD>字符，将每一个单词填充或截断，使其具有长度$\mathrm{m}_{\mathrm{word}}$，

$$\mathrm{x}_{\mathrm{padded}} = [c_1, c_2, \ldots, c_{m_{word}}] \in \mathbb{Z}^{m_{word}}$$

对于每一个字符c_i，我们查找一个字符嵌入（具有形状$\mathrm{e}_{\mathrm{char}}$），这将产生一个张量$\mathrm{x}_{\mathrm{emb}}$

$$\mathrm{x}_{\mathrm{emb}} = CharEmbedding(x_{padded}) \in \mathbb{R}^{\mathrm{m}_{\mathrm{word}} \mathrm{x} \mathrm{e}_{\mathrm{char}}}$$

在进入卷积网络之前，我们将对$\mathrm{x}_{\mathrm{emb}}$进行形状重塑以获得$\mathrm{x}_{\mathrm{reshape}} \in \mathrm{R}^{e_{char} \times m_{word}}$。为了计算第i个输出特征（其中$\mathrm{i} \in \{1, \ldots, \mathrm{f}\}$），对于输入的第 t 个窗口，在输入窗口$(\mathrm{x}_{\mathrm{reshaped}})_{[:,\mathrm{t}:\mathrm{t}+\mathrm{k}-1]} \in \mathrm{R}^{\mathrm{e}_{\mathrm{char}} \times \mathrm{k}}$以及权重矩阵$\mathrm{W}_{[\mathrm{i},:,:]} \in \mathrm{R}^{\mathrm{e}_{\mathrm{char}} \times \mathrm{k}}$之间执行卷积操作，并加上偏置项$\mathrm{b}_{\mathrm{i}} \in \mathrm{R}$：

$$(\mathbf{x}_{\mathrm{conv}})_{i,t} = \mathrm{sum}\left(\mathbf{W}_{[i,:,:]} \odot (\mathbf{x}_{\mathrm{reshaped}})_{[:,\, t:t+k-1]}\right) + \mathbf{b}_i \quad \in \mathbb{R}$$

其中⊙是两个形状相同的矩阵的元素相乘，sum 是矩阵中所有元素的和。对每个特征i和每个窗口t执行此操作，其中$\mathrm{t} \in \{1, \ldots, \mathrm{m}_{\mathrm{word}} - \mathrm{k} + 1\}$。这些步骤会综合产生输出$\mathrm{x}_{\mathrm{conv}}$：

$$\mathbf{x}_{\mathrm{conv}} = \mathrm{Conv1D}(\mathbf{x}_{\mathrm{reshaped}}) \in \mathbb{R}^{f \times (m_{\mathrm{word}} - k + 1)}$$

对于我们的应用程序，我们将f设置为等于$\mathrm{e}_{\mathrm{word}}$，即单词x的最终的嵌入大小（图 2 中最右边的向量）。因此，

$$\mathbf{x}_{\mathrm{conv}} \in \mathbb{R}^{e_{\mathrm{word}} \times (m_{\mathrm{word}} - k + 1)}$$

最后，我们将ReLU函数应用于$\mathrm{x}_{\mathrm{conv}}$，然后使用最大池化(max-pooling)将其转换为单个向量$\mathrm{x}_{\mathrm{conv_out}} \in \mathrm{R}^{e_{\mathrm{word}}}$,这就是卷积网络的最终输出：

$$\mathbf{x}_{\mathrm{conv\_out}} = \mathrm{MaxPool}(\mathrm{ReLU}(\mathbf{x}_{\mathrm{conv}})) \in \mathbb{R}^{e_{\mathrm{word}}}$$

在这里，MaxPool取第二个维度上的最大值。给定矩阵$\mathrm{M} \in \mathrm{R}^{\mathrm{a} \times \mathrm{b}}$，有$\mathrm{MaxPool(M)} \in R^a$。对于$\mathrm{i} \in \{1, \ldots, \mathrm{a}\}, \mathrm{MaxPool(M)_i} = \max_{1 \le \mathrm{j} \le \mathrm{b}} \mathrm{M_{ij}}$。Highway 网络有一个由动态门控制的跳跃连接（skip-connection）。给定输入$\mathrm{x}_{\mathrm{conv_out}} \in \mathrm{R}^{e_{\mathrm{word}}}$，我们计算：

$$\mathbf{x}_{\mathrm{proj}} = \mathrm{ReLU}(\mathbf{W}_{\mathrm{proj}} \mathbf{x}_{\mathrm{conv\_out}} + \mathbf{b}_{\mathrm{proj}}) \quad \in \mathbb{R}^{e_{\mathrm{word}}}$$

$$\mathbf{x}_{\mathrm{gate}} = \sigma(\mathbf{W}_{\mathrm{gate}} \mathbf{x}_{\mathrm{conv\_out}} + \mathbf{b}_{\mathrm{gate}}) \quad \in \mathbb{R}^{e_{\mathrm{word}}}$$

其中权重矩阵W_{proj}，$W_{gate} \in R^{e_{word} \times e_{word}}$，偏差向量$b_{proj}$，$b_{gate} \in R^{e_{word}}$，σ 是sigmoid函数。接下来，我们使用门将一般投影和跳跃连接结合起来，获得输出$x_{highway}$

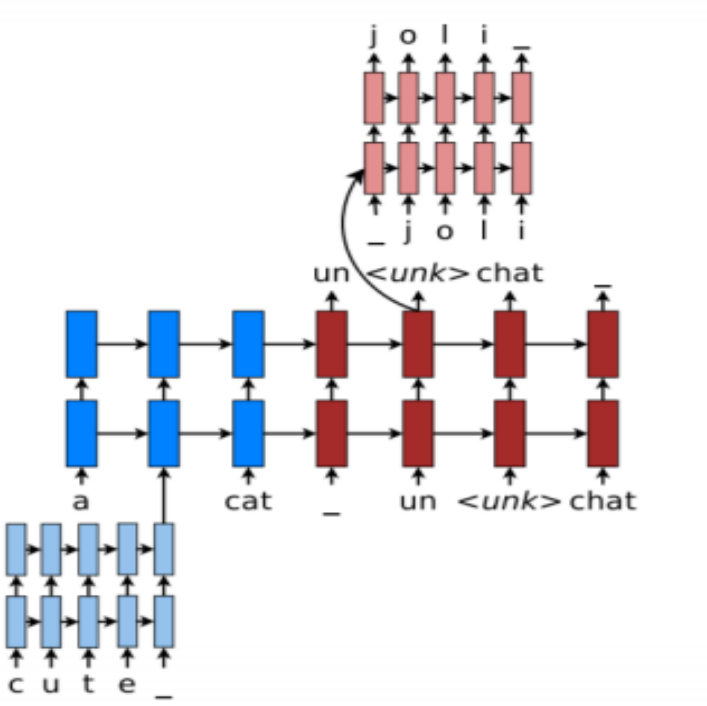
$$\mathbf{x}_{\text{highway}} = \mathbf{x}_{\text{gate}} \odot \mathbf{x}_{\text{proj}} + (1 - \mathbf{x}_{\text{gate}}) \odot \mathbf{x}_{\text{conv\_out}} \in \mathbb{R}^{e_{\text{word}}}$$

其中⊙表示按元素进行的乘法运算。最后，我们向$x_{highway}$应用 dropout：

$$\mathbf{x}_{\text{word\_emb}} = \text{Dropout}(\mathbf{x}_{\text{highway}}) \in \mathbb{R}^{e_{\text{word}}}$$

x_{word_emb}是我们将用于表示单词x的嵌入——这将替换我们在实验三中使用的基于查找的单词嵌入。

字符级编码器结构如下图所示，



其主要思想是，当我们的词级解码器生成一个<UNK>标记时，我们运行字符级解码器（您可以将其视为字符级条件语言模型）来每次生成目标单词一个字符，如图 3 所示。这将有助于我们产生生僻的和词汇外(oov)的目标词。

给定一个整数序列$x_1, \ldots, x_n \in Z$代表一个字符序列，我们查找他们的字符嵌入$\mathbf{x}_1, \ldots, \mathbf{x}_n \in R^{e_{char}}$,然后将其作为单向 LSTM 的输入。获得隐层状态$\mathbf{h}_1, \ldots, \mathbf{h}_n$以及单元状态$\mathbf{c}_1, \ldots, \mathbf{c}_n$:

$$\mathbf{h}_t, \mathbf{c}_t = \text{CharDecoderLSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \text{ where } \mathbf{h}_t, \mathbf{c}_t \in \mathbb{R}^h$$

其中h是 CharDecoderLSTM 隐层状态的大小。对于词级 NMT 解码器的当前时间步，初始的隐层状态$\mathbf{h}_0$和细胞状态$\mathbf{c}_0$，都被设置入组合输出向量(combined output vector)。对于每个时间步$t \in \{1, \ldots, n\}$我们计算分数（也称为 logits）$s_t \in R^{V_{char}}$:

$$\mathbf{s}_t = \mathbf{W}_{\text{dec}}\mathbf{h}_t + \mathbf{b}_{\text{dec}} \in \mathbb{R}^{V_{char}}$$

其中权重矩阵$W_{dec} \in R^{V_{char} \times h}$和偏置向量$b_{dec} \in R^{V_{char}}$。如果我们使$s_t$经过一个softmax函数，我们将得到序列中下一个字符的概率分布。

当我们训练 NMT 系统时，我们用目标句子中每个单词训练字符解码器（不仅仅是由<UNK>

表示的单词）。例如，在主 NMT 解码器的特定步骤上，如果目标单词是 music，那么 CharDecoderLSTM 的输入序列是 $[x_1,\ldots,x_n]=[<\text{START}>,\text{m},\text{u},\text{s},\text{i},\text{c}]$，CharDecoderLSTM 的目标序列是 $[x_2,\ldots,x_{n+1}]=[\text{m},\text{u},\text{s},\text{i},\text{c},<\text{END}>]$。我们将输入序列 $x_1,\ldots,x_n$（以及从组合输出向量获得的初始状态 $\mathbf{h}_0$ 和 $\mathbf{c}_0$）输入 CharDecoderLSTM，从而获得分数 $s_1,\ldots,s_n$，我们将其与目标序列 $x_2,\ldots,x_{n+1}$ 进行比较。我们根据交叉熵损失之和进行优化：

$$\mathbf{p}_t=\text{softmax}(\mathbf{s}_t)\in\mathbb{R}^{V_{\text{char}}} \qquad \forall t\in\{1,\ldots,n\}$$

$$\text{loss}_{\text{char\_dec}}=-\sum_{t=1}^{n}\log\mathbf{p}_t(x_{t+1})\in\mathbb{R}$$

注意，当我们计算一批单词的 $\text{loss}_{\text{char\_dec}}$ 时，我们取整批单词的和（不是平均值）。在每次训练迭代中，我们将 $\text{loss}_{\text{char\_dec}}$ 加入到基于词的解码器的损失中，从而同时训练基于词的模型和基于字符的解码器。

在测试时首先我们以通常的方式从我们的基于词的 NMT 系统产生翻译（例如，像 beam search 这样的解码算法）。如果翻译包含任何<UNK>标记，那么对于每个那样的位置，我们使用基于单词的解码器的组合输出向量来初始化 CharDecoderLSTM 的初始 $\mathbf{h}_0$ 和 $\mathbf{c}_0$，然后使用 CharDecoderLSTM 来生成字符序列。为了生成字符序列，我们使用贪婪解码算法，该算法反复选择最可能的下一个字符，直到生成<END>标记或达到预定的最大长度。算法如下所示，仅针对一个样本（不成批）。

四、实验方案

4.1 数据集介绍及预处理

实验数据集使用实验提供的数据集，该数据集包含三个部分 train,dev 和 test，train 和 dev 用于训练模型、验证模型，test 用于测试模型，在预处理时，要建立翻译语言和目标语言的语料库，建立好语料库后，根据语料库单词索引将输入神经网络的字符串句子转换成 tensor，基于字符的输入，要建立字符索引，并基于字符索引转化成 tensor。

4.2 超惨设置

参数均采用默认设置，词向量大小为 256，lstm 隐藏层的大小为 256，设置 batch_size 大小为 32，dropout 概率设成 0.3，学习率为 0.001。

4.3 优化函数及评价指标

优化函数使用交叉熵计算，将输入序列 $x_1,\ldots,x_n$（以及从组合输出向量获得的初始状态 h_0 和 c_0）输入 CharDecoderLSTM，从而获得分数 $s_1,\ldots,s_n$，我们将其与目标序列 $x_2,\ldots,x_{n+1}$ 进行比较

$$\mathbf{p}_t=\text{softmax}(\mathbf{s}_t)\in\mathbb{R}^{V_{\text{char}}} \qquad \forall t\in\{1,\ldots,n\}$$

$$\text{loss}_{\text{char\_dec}}=-\sum_{t=1}^{n}\log\mathbf{p}_t(x_{t+1})\in\mathbb{R}$$

4.5 其它探索

以上探讨的词向量 embedding 过程中没有考虑到词的上下文环境，引入上下文环境的词向量应该可以进一步提高性能，在实验中，尝试使用 CNN 和 LSTM 在词向量级别对词向量重新计算，综合上下文环境，我们在上面的 embedding 过程中得到 $x_{\text{word\_emb}}$，在此基础上我们引入 CNN 对词进行 1D 卷积得到，综合上下文环境的词向量

$$x_{\text{contex}}=CNN(x_{word\_emb})$$

引入门结构使原先的词向量和词的上下文环境综合，得到

$$\mathrm{x}_{\mathrm{gate}} = \sigma(W_{gate}[x_{contex}, x_{word\_emb}] + b_{gate})$$

$$\mathrm{x}_{\mathrm{contex\_word\_emb}} = x_{gate} \odot x_{contex} + (1 - \mathrm{x}_{\mathrm{gate}}) \odot x_{word\_emb}$$

还有一种综合上下文环境的方法是使用 LSTM，从而前面$\mathrm{x}_{\mathrm{contex}}$变成

$$\mathrm{x}_{\mathrm{contex}} = LSTM(x_{word\_emb})$$

另外，还观察到上面所提及的模型只是用了一层 LSTM，实际上语言的特征也是分层次的，底层表示词本身的含义，高层表示词的词性、在句子中修饰成分等高层语义，使用多层次 LSTM 来对句子进行编码，在第 1 层 LSTM 的结构为

$$\mathrm{x}_{\mathrm{hidden}}^{(l+1)} = LSTM\left(x_{hidden}^{(l)}\right)$$

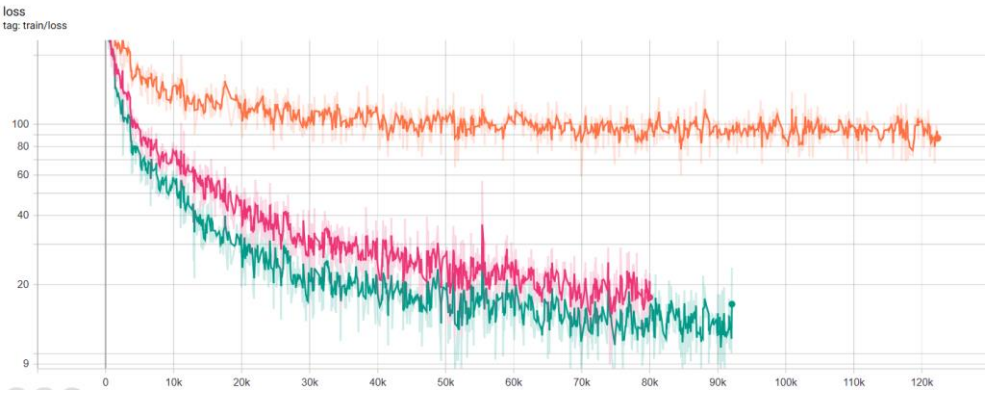
计算 LSTM 的输入时，引入带有门结构的跳跃连接，那么输入第 l+1 层 LSTM 的向量通过下式计算，

$$\mathrm{x}_{\mathrm{gate}}^{(l)} = \sigma\left(W_{gate}^{(l)}\left[x_{hidden}^{(l-1)}, x_{hidden}^{(l)}\right] + b_{gate}^{(l)}\right)$$

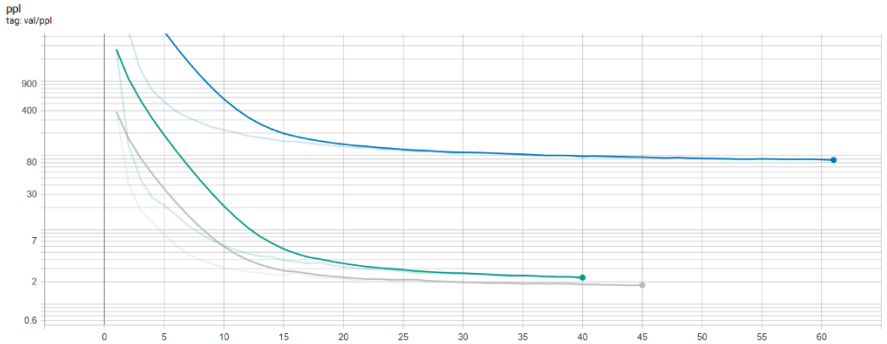
$$\mathrm{x}_{\mathrm{hidden\_in}}^{(l)} = \mathrm{x}_{\mathrm{gate}}^{(l)} \odot x_{hidden}^{l} + (1 - x_{gate}^{(l)}) \odot x_{hidden\_in}^{(l-1)}$$

五、实验结果

在实验过程中，发现引入上下文环境的词向量，能够使损失下降的很快，模型收敛到更小损失值状态，如下图所示，



上图的横坐标是训练计数，纵坐标为交叉熵损失，注意到纵坐标做了对数处理，橙色为没有改进的损失曲线，粉色使用了 LSTM 综合上下文，绿色使用 CNN 综合上下文。可以很明显感受到引入上下文环境使模型的损失降到更低的水平。模型在验证集上的 ppl 指标也是如此，



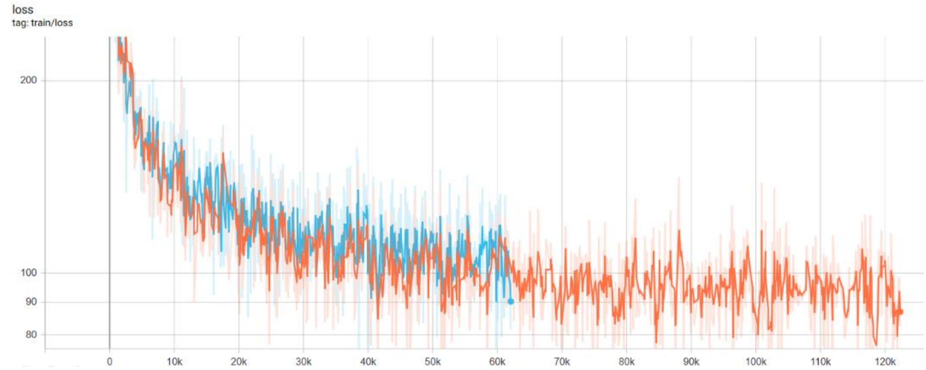
横坐标为验证迭代计数次数，纵坐标使 ppl 指标，图中的曲线从上到下依次为没有改进的、引入 LSTM 综合上下文环境、引入 CNN 综合上下文环境。经过修改的模型在训练过程的效果虽然是好的，但是在测试集上测试的 BLEU 分数为 0！这是另我困惑的地方，明明在训练集上的效果很好，却在测试集上的效果如此差，模型把句子同一翻译成：

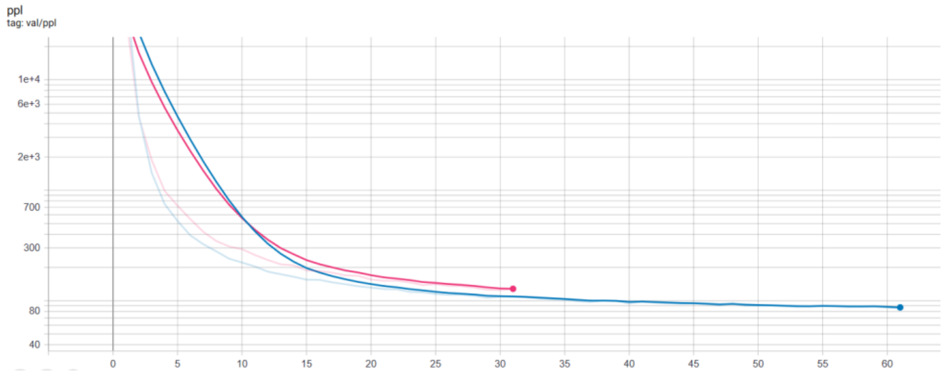
If us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us
us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us us
It

a a's you y a's you y a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a's you why a

诸如此类无意义的句子，令人奇怪的是，为什么这样的模型在训练过程中损失很小，却翻译的结果很差。

下图给出了单层 LSTM 编码器和多层带有超链接的 LSTM 编码器的训练过程和验证结果，





多层 LSTM 没有在原来基础上改进多少，反而效果不如改进前的模型，最终我们测试的未改进前的 BLEU 分数为 35.49448，多层 LSTM 的 BLEU 分数为 32.92713，其它引入上下文环境的词向量 embedding 模型的 BLEU 分数为 0。

六、总结和未来工作

本次实验训练了一个神经网络模型，并尝试对其进行改进，我们发现经过上面所述方法改进之后的模型不如未改经的模型，引入上下文环境的词向量 embedding 模型可以是模型在训练过程中的损失值很低，但是翻译效果很差，多层 LSTM 效果不如单层 LSTM。本次实验所探讨的模型是句子和句子之间的直接映射，模型的感知域为单个句子，没有把句子放到段落片段中，也没有引入外界知识，而实际上，句子的上下文环境和领域知识对翻译是至关重要的，未来的工作是如何将段落环境和领域知识融入到模型中，从而进一步提高翻译效果。

七、参考文献

[1] Kim, Yoon & Jernite, Yacine & Sontag, David & Rush, Alexander. (2015). Character-Aware Neural Language Models.