

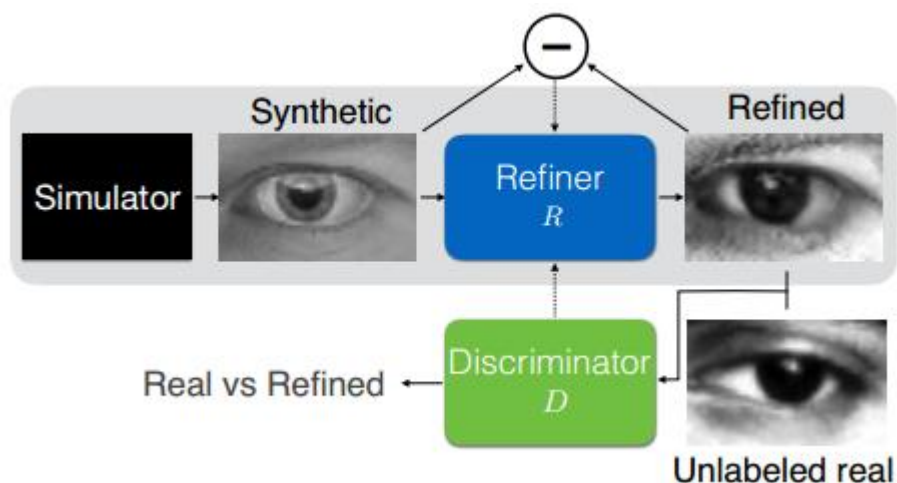
## 目录

一、 Learning from Simulated and Unsupervised Images through Adversarial Training .....	1
二、 Context Gates for Neural Machine Translation .....	3
三、 Coverage-based Neural Machine Translation .....	4
四、 Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.....	5
五、 word embedding.....	8
六、 encoder & bucketing .....	11
七、 A convolutional encoder model for neural machine translation ,Facebook .....	13
八、 attention.....	18
九、 beam search.....	21
十、 clip gradients.....	22
十一、 BLEU .....	23
十二、 平方误差和交叉熵.....	24

## 刘曼飞-20170930-阅读笔记

### 一、 Learning from Simulated and Unsupervised Images through Adversarial Training

借鉴 GAN 的 D+G 的思想设计 S+R+D 的网络。



#### 1、 Refiner

将  $R\theta$  作为一个完全卷积的神经网络，而无需 striding 和 Pooling。

实施细节：精炼网络  $R\theta$  是一个残差网络 (ResNet) 。

**(重点1)** 在像素级别上修改合成图像，而不是整体地修改图像内容。

（重点 3）设置缓冲池存储历史图像，来提高对抗训练的稳定性，防止对抗分歧。

（重点 4）学习网络不需要对真实图像进行标记。

Loss: 真实性成本+注释信息

$$\mathcal{L}_R(\theta) = \sum_i \ell_{\text{real}}(\theta; \mathbf{x}_i, \mathcal{Y}) + \lambda \ell_{\text{reg}}(\theta; \mathbf{x}_i)$$

$$\begin{aligned} \mathcal{L}_R(\theta) = & - \sum_i \log(1 - D_\phi(R_\theta(\mathbf{x}_i))) \\ & + \lambda \|\psi(R_\theta(\mathbf{x}_i)) - \psi(\mathbf{x}_i)\|_1. \end{aligned}$$

refine 的图像是合成图的概率越小，refiner 越有效。

最后一项的 $\phi$ 只是一个泛化函数，具体可以是中间的 feature map 等

## 2、 Discriminator

$D_\phi$  是 ConvNet 最后一层是输入图像为合成图像的概率。

（重点 2）D 的输入是将整张图切割成  $w \times h$  个 patch，计算每个 patch 的 loss 并且反馈给 R。

$$\mathcal{L}_D(\phi) = - \sum \log(D_\phi(\tilde{\mathbf{x}}_i)) - \sum \log(1 - D_\phi(\mathbf{y}_j))$$

合成图像判定为合成图像的概率 D 越大，loss 越小

真实图像判定为合成图像的概率越小，loss 越小

## 3、训练算法


**Algorithm 1:** Adversarial training of refiner network  $R_\theta$

**Input:** Sets of synthetic images  $\mathbf{x}_i \in \mathcal{X}$ , and real images  $\mathbf{y}_j \in \mathcal{Y}$ , max number of steps ( $T$ ), number of discriminator network updates per step ( $K_d$ ), number of generative network updates per step ( $K_g$ ).

**Output:** ConvNet model  $R_\theta$ .

```
for  $t = 1, \dots, T$  do
  for  $k = 1, \dots, K_g$  do
    1. Sample a mini-batch of synthetic images  $\mathbf{x}_i$ .
    2. Update  $\theta$  by taking a SGD step on mini-batch loss  $\mathcal{L}_R(\theta)$  in (4).
  end
  for  $k = 1, \dots, K_d$  do
    1. Sample a mini-batch of synthetic images  $\mathbf{x}_i$ , and real images  $\mathbf{y}_j$ .
    2. Compute  $\tilde{\mathbf{x}}_i = R_\theta(\mathbf{x}_i)$  with current  $\theta$ .
    3. Update  $\phi$  by taking a SGD step on mini-batch loss  $\mathcal{L}_D(\phi)$  in (2).
  end
end
```

历史refined图+当前refined图



3、 实验:

UnityEyes 上训练 ,并在 MPIIGaze 上进行测试 ,提高了 22.3%( 69.4%->87.2% )

NYU 手势数据库提高 8.8%

## 二、Context Gates for Neural Machine Translation

这篇文章主要讲了三件事:

证明source 和targetcontext 分别会影响翻译的完整度和流畅度。证明方法为控制变量法,将source context和target context分别乘以不同的权重,观察对翻译结果的影响。这个证明也是文中提出方法的motivation。

提出一种context gate的方法。其具体为分为以下两个步骤:

(1) 通过参数矩阵计算source和target的权重向量（即context gate）

$$z_i = \sigma(W_z e(y_{i-1}) + U_z t_{i-1} + C_z s_i)$$

(2) 计算赋权source与target，继续进行attention\_decoder的运算

$$t_i = f((1-z_i) \circ (W e(y_{i-1}) + U t_{i-1}) + z_i \circ C s_i)$$

证明了实验效果可以平均提高BLEU值2.+个点，效果比较显著。通过人工的双盲评价方法，证明文中方法翻译的对流畅度也有改进。

### 三、Coverage-based Neural Machine Translation

论文主要解决的问题是翻译问题中的过翻译（某些词被重复翻译）和欠翻译（某些词在翻译的过程中被忽略）的问题。

文章提出的思路是用一个coverage vector标记source中被翻译过的词语和未被翻译的词语，具体过程如下：

(1) 求得coverage vector

第一种是通过语言模型得到：具体为alignment probability的累加。alignment

probability是指source的i-th tag到target 的j-th tag的映射概率。

a、设置fertility，表示source中每个词可能会被翻译成target中的几个词

$$\Phi_j = \mathcal{N}(x_j | \mathbf{x}) = \mathcal{N}(h_j) = N \cdot \sigma(U_f h_j)$$

b、求coverage vector：

$$\beta_{i,j} = \frac{1}{\Phi_j} \sum_{k=1}^i \alpha_{k,j}$$

第二种方法是基于神经网络学习到：

$$\begin{aligned}\beta_{i,j} &= g_{update}(\beta_{i-1,j}, \alpha_{i,j}, h_j, s_{i-1}) \\ &= \tanh(U\beta_{i-1,j} + V\alpha_{i,j} + Bh_j + Ws_{i-1})\end{aligned}$$

这里的g-update函数可以是RNN神经网络。基于公式中分别是：上一个时间点的coverage vector, alignment probability、input annotation (由x经过RNN/B-RNN网络得到)、当前时间点的 hidden state (decoder中输出前某个中间层的结果)。

(2) coverage vector作用于seq2seq网络：

$$\begin{aligned}e_{i,j} &= a(s_{i-1}, h_j, \beta_{i-1,j}) \\ &= v_a^T \tanh(W_a s_{i-1} + U_a h_j + B_a \beta_{i-1,j})\end{aligned}$$

具体为将coverage vector与hidden state、input annotation 共同作为attention model 的输入，帮助attention model 决策下一个要take attention的source tag。

实验结果证明在模型较小的情况下可以达到和 state-of-art 几乎持平的效果。

## 四、Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

### 1、模型结构

参考文章 <http://www.qingpingshan.com/bc/jsp/297960.html>

#### 1) 参数说明

$\mathbf{v}, \mathbf{o}_i$  vectors

$\mathbf{U}, \mathbf{W}$  matrices

$\mathcal{V}, \mathcal{I}$  sets

$X, Y$  sequences

$x_1, x_2$  symbols in a sequence

## 2) 结构

source (input)  $X = x_1, x_2, x_3, \dots, x_M$

target  $Y = y_1, y_2, y_3, \dots, y_N$

encoder-decoder模型的想法很简单：对于RNN语言模型，在我们计算输出序列E的概率时，先另一个RNN处理源序列F，来计算语言模型的初始状态。**encoder-decoder**的含义是：通过第一个神经网络来“编码”F的信息到一个隐层状态，在使用第二个神经网络来预测E“解码”该隐层到输出序列。

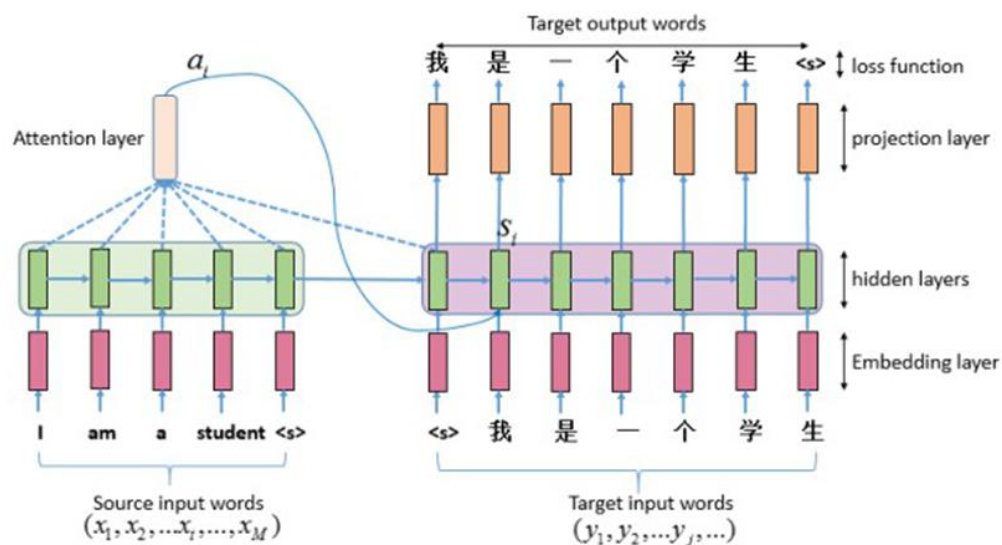


图1 encoder-decoder结构

### (1) Embedding

**embedding**: 给定单词的分类属性，模型首先查找词来源和目标嵌入以检索相应的词表征（全连接矩阵）。

### (2) Encoder

$$x_1, x_2, \dots, x_M = \text{EncoderRNN}(x_1, x_2, x_3, \dots, x_M)$$

encoder 和decoder都是RNN网络，原则上可以共享相同的权重，然而在实践中通常使用两组不同的循环神经网络参数（这些模型在拟合大型训练数据集上做得更好）。编码器 RNN 使用零向量作为它的初始状态。对于变长的输入会添加结束标识符。通过 `source_sequence_length` 告诉 `dynamic_rnn` 精确的句子长度以增加计算效率。

### ( 3 ) attention

核心思想是当我们翻译时通过「注意」相关的源内容，建立直接的短连接。attention的一个很好副产品是源语句和目标语句之间的一个易于可视化的对齐矩阵。

attention context  $a_i$ 基于以下三步求得：

$$s_t = \text{AttentionFunction}(y_{i-1}, x_t) \quad \forall t, \quad 1 \leq t \leq M$$

$$p_t = \exp(s_t) / \sum_{t=1}^M \exp(s_t) \quad \forall t, \quad 1 \leq t \leq M$$

$$a_i = \sum_{t=1}^M p_t \cdot x_t$$

其中的attention function是一个只有一层的全连接网络， $s_t$ 是attention vector，在求解下一个 $y_i$ 时候代替 $x_t$

### ( 4 ) decoder

decoder 也需要访问源信息，一种简单的方式是用编码器最后的隐藏态 `encoder_state` 对其进行初始化。在图 1中，我们将源词「student」中的隐藏态传递到了解码器。

### ( 5 ) loss function

这里采用交叉熵函数。交叉熵函数的物理意义是使用预测的分布 $q$ 来表示来自真实分布 $p$ 的编码长度。

代码中batchsize中i-th个样本对应到j-th类的loss计算如下：

$$y = labels \quad p_{ij} = \text{sigmoid}(\text{logits}_{ij}) = \frac{1}{1 + e^{-\text{logits}_{ij}}}$$

$$\text{loss}_{ij} = -[\text{pos\_weight} * y_{ij} * \ln p_{ij} + (1 - y_{ij}) \ln(1 - p_{ij})]$$

用交叉熵而非平方误差作为loss的原因是：交叉熵求导之后和sigmoid function的导数无关，解决了反向传播梯度过小，收敛过慢或者收敛不了的问题。

## **(6) 梯度计算和优化**

使用clip gradients，防止梯度爆炸。使用Adam优化方法进行反向传播。

## **(7) 测试：**

beam search 搜索最优路径。

存在问题：

- 1、语句不通顺，句法语法组合奇怪
- 2、否定词的错误使用或丢失，使得句子表达了与期望相反的内容
- 3、有重复的词语出现。
- 4、gleu是否足以作为流畅性的判别标准

思考：

- 1、GNMT中提出在loss中添加句子的reward loss (RL) 实现对通顺性的改进。但是对英->法有效，对英->德无效。
- 2、使用seqGAN做强化学习。

# **五、word embedding**

## **1) motivation**

(1) 传统自然语言处理的做法是将词表示成离散的符号，导致没有提供足够的信息来体现词语之间的某种关联。

例如将 [cat] 表示为 [Id537]，而 [dog] 表示为 [Id143]。尽管cat和dog不是同一



个词，但是却应该有着某种的联系（如都是属于动物种类）。

（2）这种一元表示法(One-hot Representation)使得词向量过于稀疏，所以往往需要大量的语料数据才能训练出一个令人满意的模型。

## 2) 背景知识：

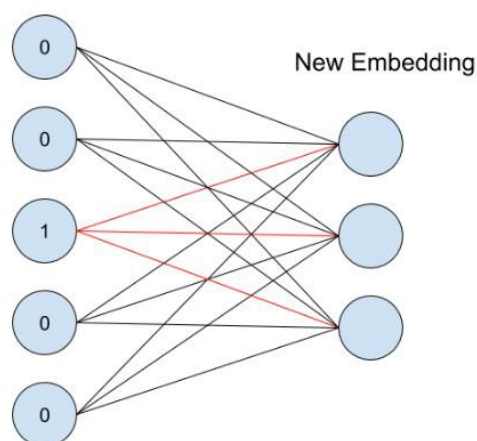
传统的向量空间模型(Vector space models, VSMs)将词语表示为一个连续的词向量，并且语义接近的词语对应的词向量在空间上也是**接近的**。

## 3) 具体实现

我们首先需要为每一种语言选定一个词汇表。通常，选定词汇表大小  $V$ ，那么频率最高的  $V$  个词将视为唯一的。而所有其他的词将转换并打上「unknown」标志，因此所有的词将有相同的嵌入。我们通常在训练期间嵌入权重，并且每种语言都有一套。

GNMT中可以看作是通过没有bias的全连接矩阵进行embedding操作。embedding的结果是连接的weight。

One-hot Embedding



实现代码细节：

encoder\_inputs [max\_encoder\_time, batch\_size]：源输入词。

decoder\_inputs [max\_decoder\_time, batch\_size]：目标输入词。

decoder\_outputs [max\_decoder\_time, batch\_size]：目标输出词，这些是

decoder\_inputs 按一个时间步向左移动，并且在右边有句子结束符。

首先需要为每一种语言选定一个词汇表。通常，选定词汇表大小  $V$ ，那么频率最高的  $V$

个词将视为唯一的。而所有其他的词将转换并打上UNK [unknown] 标志

### 3) 词向量维度影响：

词向量的维度一般要选择50维及以上。（一般根据具体任务进行实验，最后根据性能和实验需使用的时间选择合适的词向量维度）

对于分析词向量语言学特性的任务，维度越大效果越好。

对于提升自然语言处理任务而言，50维词向量通常就足够好。（随着词向量维度的增加，性能曲线先增长后趋近于平缓，甚至下降）

### 4) 是否可以预训练

理论上来说是可以进行预训练的，但是一般不这样做原因是paper作者在论文中指出：

根据词向量的损失函数选择迭代次数不合适。

条件允许的话，选择目标任务的验证集性能作为参考标准。

具体任务性能指标趋势一样，可以选简单任务的性能峰值。

如果预训练，可以用以下评价标准：

### (1) 词向量的语言学特性

词义相关性(ws): WordSim353数据集，词对语义打分。皮尔逊系数评价。

同义词检测(tfi): TOEFL数据集，80个单选题。准确率评价

单词语义类比(sem): 9000个问题。queen-king+man=women。准确率

单词句法类比(syn): 1W个问题。dancing-dance+predict=predicting。准确率

### (2) 词向量用作特征的评价

基于平均词向量的文本分类(avg): IMDB数据集，Logistic分类。准确率评价

命名实体识别(ner): CoNLL03数据集，作为现有系统的额外特征。F1值

### (3) 词向量用作神经网络模型的初始值的评价

参考文献: <http://www.cnblogs.com/robert-dlut/p/5617466.html>; 《How to Generate a Good Word Embedding?》

## 六、encoder & bucketing

### 1、实现

encoder 和decofer都是RNN网络，原则上可以共享相同的权重，然而在实践中通常使用两组不同的循环神经网络参数（这些模型在拟合大型训练数据集上做得更好）。编码器 RNN 使

用零向量作为它的初始状态。对于变长的输入会添加结束标识符。通过 `source_sequence_length` 告诉 `dynamic_rnn` 精确的句子长度以增加计算效率。

## 2、输入长度要求，为啥用bucket，有啥可替代的方法（dynamic RNN）效率和结果是否有提高，不用是否有什么区别。

1) 使用bucket 的原因是输入句子不定长的问题。对不定长的输入，rnn 的参数相同，但是计算图是不同的。

(1) 如果全部补充到最长的长度，其余的补充空白标识符（blank），填充太多空白符号会做一些没用的计算，时间代价浪费大。

(2) 补充空白到最大长度但忽略填充符的计算：在数值优化中，基本的计算，数据结构越简单越好，而这种忽略某几行的操作对于每个operator的要求都很高。

(3) 如果只是定义到几个长度区间中去，即bucket方法，就可以用长度区间中的句子补充到最大长度，但是特别长但是数量很少的长尾数据不会产生太大的副作用。节约计算资源，在开始的时候把所有的模型变量分配好，集中计算图固定下来，计算梯度的过程也可以固定下来。

bucket具体为：对于sequence先做聚类，预设几个固定的长度bucket，然后每个sequence都放到它所属的bucket里面去，然后pad到固定的长度。这

参考文献：<https://www.zhihu.com/question/42057513/answer/93421874>

2) `tf.dynamic_rnn`: 根据每个batch自动计算最好的输出，不过要更定每个example的 `sequence length`。其实minibatch中的每个长句子还是会提高batch内的时间代价。具体为：

假设RNN的输入input是`[2,20,128]`，其中2是`batch_size`,20是文本最大长度，128是

embedding\_size , dynamic返回的是两个参数：outputs,last\_states , 其中outputs是[2,20,128] , 也就是每一个迭代隐状态的输出,last\_states是由(c,h)组成的tuple , 均为[batch,128]。

两个sample , 一个长20 , 一个长13。

设置sequence\_length后 , sequence\_length长度以后的padding就不计算了 , 其last\_states将重复第13步的last\_states直至第20步 , 而outputs中超过13步的结果将会被置零。

参考文献： <http://blog.csdn.net/u010223750/article/details/71079036>

优缺点：比其他的RNN更方便和节约资源 , 但相对于bucket来说时间代价提高（由于tf采用符号式编程 , 动态数据每一次都要重新分配计算图）。实验结果影响不大。

在一个minibatch中只能用相同的长度（涉及到参数计算图的构建）（不同minibatch如果用不同的长度会使得自动梯度计算（autodiff）受到影响。）

RNN的bucketing其实是一个很好的例子：在深度学习的算法实现上，数学是一部分，工程是另一部分。好的实现，都是在数学的优雅和工程的效率之间寻找一个最优折衷的过程。

## 七、 A convolutional encoder model for neural machine translation ,Facebook

cnn最早用在文本分类上（细节），现在Facebook模拟RNN用来做NMT。

1）为什么用CNN代替RNN：

RNN无法利用未来的特征预测当前单词，就算是bi-RNN，也不过是双向重蹈覆辙而已。

rnn经常把过多注意力放到最后一个单词上。

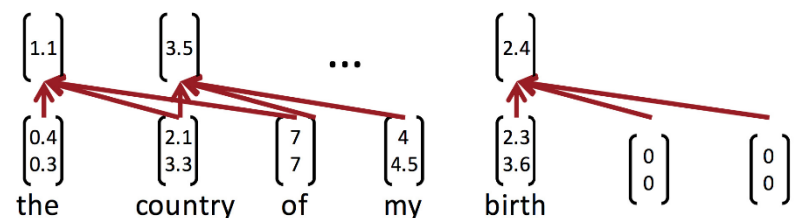
## 2) cnn用于序列问题的思想和算法

CNN的解决思路说来也很简单粗暴，那就计算相邻的ngram，不管它到底是不是真正的短语，眉毛胡子一把抓地为它们计算向量。具体为：

(1) 假设词向量 $x_i \in R^k$ ，那么nn个单词的句子可视作词向量(列向量)的按行拼接

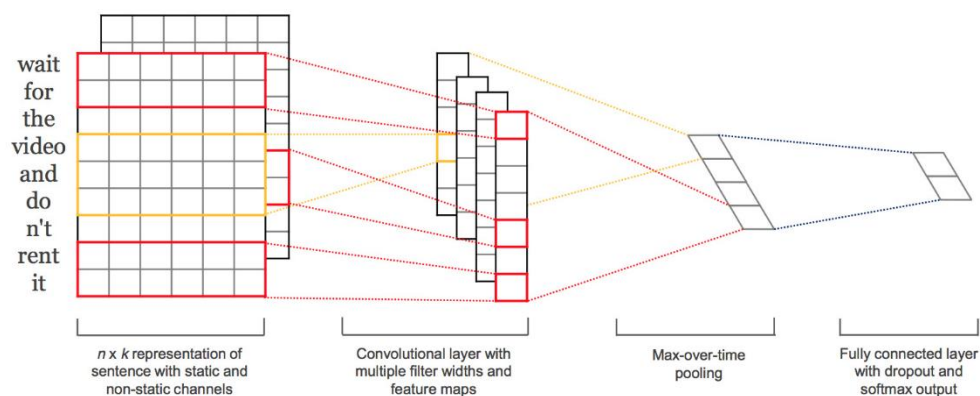
$$x_{1:n} = x_1 \oplus x_2 \dots \oplus x_n \quad x_{1:n} = x_1 \oplus x_2 \dots \oplus x_n。$$

(2) 对于h个单词上的卷积过滤器  $w \in R^{hk}$ ，比如 $k=2, n=5, h=3$ ，下图演示了NLP中的单层CNN工作过程：



为了得到多个卷积特征，简单地使用多个卷积核（不需要大小一致），然后把池化结果拼接起来。

下图为一个漂亮的图示：双通道词向量=>多个卷积核得到的feature map（红色是bigram，橙色是trigram）=>池化得到最终特征=>softmax分类输出。



参考文献：<http://www.hankcs.com/nlp/cs224n-convolutional-neural->

networks.html

优点：可以变换很多花样，比如每个feature map是变大了还是变小了，池化技巧等等

3) 用于NLP问题：

CNN：适合分类，不清楚如何做短语级别的标注，对短文本需要padding，难以用NLP的视角解释，容易GPU并行化

RNN：从左读到右，最符合认知。不是分类任务的最佳选择，比CNN慢，但可以做序列标注。

facebook 的 convolutional SeqSeq取得了超越Google翻译的成果，重要原因在于采用了很多的trick，很多工作值得借鉴：

1、Position Embedding，在输入信息中加入位置向量 $P = (p_1, p_2, \dots)$ ，把位置向量与词向量 $W = (w_1, w_2, \dots)$ 求和构成向量 $E = (w_1 + p_1, w_2 + p_2)$ ，做为网络输入，使由CNN构成的Encoder和Decoder也具备了RNN捕捉输入Sequence中词的位置信息的功能。

2、多层CNN构成了hierarchical representation表示。层叠的CNN拥有3个优点：

(1) 捕获long-distance依赖关系。底层的CNN捕捉相聚较近的词之间的依赖关系，高层CNN捕捉较远词之间的依赖关系。通过层次化的结构，实现了类似RNN (LSTM) 捕捉长度在20个词以上的Sequence的依赖关系的功能。

(2) 效率高。这点说计算量变少了。不敢苟同。

(3) 可以并行化实现。RNN对sequence的建模依赖于序列的历史信息，因此不能并行实现。相比，多层CNN正个sequence进行卷积，不依赖序列历史信息，可以并行实现，模型训练更快，特别是在工业生产，面临处理大数据量和实时要求比较高的情况下。

3、融合了Residual connection、liner mapping的多层attention。通过attention决定输入的哪些信息是重要的，并逐步往下传递。把encoder的输出和decoder的输出做点乘 ( dot products )，再归一化，再乘以encoder的输入X之后做为权重化后的结果加入到decoder中预测目标语言序列。

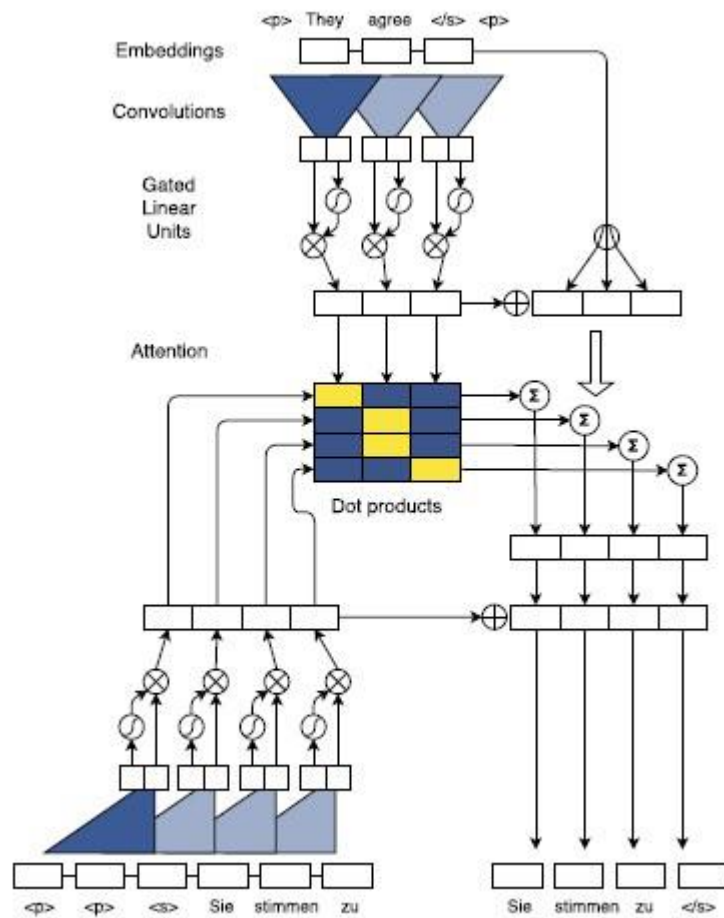
4、采用GLU做为gate mechanism。GLU单元激活方式如下公式所示：

$$h_l(\mathbf{X}) = (\mathbf{X} * \mathbf{W} + \mathbf{b}) \otimes \sigma(\mathbf{X} * \mathbf{V} + \mathbf{c}) \quad (1)$$

每一层的输出都是一个线性映射 $\mathbf{X} * \mathbf{W} + \mathbf{b}$ ，被一个门gate:  $\sigma(\mathbf{X} * \mathbf{V} + \mathbf{c})$  控制，通过做乘法来控制信息向下层流动的力， $\sigma$ 采用双曲正切S型激活函数。这个机制类似LSTM中的gate mechanism，对于语言建模非常有效，使模型可以选择那些词或特征对于预测下一个词是真的有效的。

5、进行了梯度裁剪和精细的权重初始化，加速模型训练和收敛。





上左encoder部分：通过层叠的卷积抽取输入源语言（英语）sequence的特征，图中直进行了一层卷积。卷积之后经过GLU激活做为encoder输出。

下左decoder部分：采用层叠卷积抽取输出目标语言（德语）sequence的特征，经过GLU激活做为decoder输出。

中左attention部分：把decoder和encoder的输出做点乘，做为输入源语言（英语）sequence中每个词权重。

中右Residualconnection：把attention计算的权重与输入序列相乘，加入到decoder

的输出中输出输出序列。

最后实验结论：在多个公开数据集上获得了新的state-of-the-art的成绩。

总结：个人感觉本文采用了很多简单且非常有效的trick，达到了基于LSTM的NMT方法更好的效果，正因为如此，并不能说，基于CNN seq2seq模型就一定比基于LSTM的Seq2Seq一定好。采用CNN的Seq2Seq最大的优点在于速度快，效率高，缺点就是需要调整的参数太多。上升到CNN和RNN用于NLP问题时，CNN也是可行的，且网络结构搭建更加灵活，效率高，特别是在大数据集上，往往能取得比RNN更好的结果。

参考文献：<https://zhuanlan.zhihu.com/p/26884494>

## 八、attention

### 1、motivation

传统的Encoder-Decoder缺点是，无论之前的输入语句( $x_1, x_2, \dots, x_i$ )有多长，包含多少信息量，最终都要被压缩成一个几百维的语义C vector，然后根据这个C得到翻译的文本( $y_1, y_2, \dots, y_j$ )，文本中每一个单词都是利用了同一个语义C，也就是说输入文本的每一个单词 $x_i$ 都对输出文本的单词 $y_j$ 的贡献是一样的。这意味着context越大，最终的state vector会丢失越多的信息。

事实上，因为context在输入时已知，一个模型完全可以在decode的过程中利用context的全部信息,或者部分信息而不仅仅是最后一个state。所以attention model的作用就在于能够根据序列的变化一直更新最关心的部分。

### 2、attention的具体实现

- (1)使用一层全连接网络 $a(.)$ 将目标语言和源语言的每个词联系了起来。用来计算 $s_{j-1}, h_i$ 这两者的关系分数，如果分数大则说明关注度较高，注意力分布会更加集中在这个输入单词上。
- (2)soft函数将其归一化得到一个概率分布， $a_{ij}$ 就是attention矩阵的每一个元素。
- (3)将attention矩阵乘以源输入，相当于给每个source的word加权，得到的结果就是attention context。
- (4)用上一个时间点的hidden state、输出和attention context解码出当前输出的hidden state  $s_i$ 。
- (5)当前的hidden state通过一层全连接网络解码出真正的输出 $y_i$ 。

参考文献：《Neural Machine Translation by Jointly Learning to Align and Translate》

<https://yzhihao.github.io/machine%20learning/2017/05/06/Encoder-Decoder%E6%A1%86%E6%9E%B6.html#attention-based-rnn-in-nlp>

[http://blog.csdn.net/qq\\_21190081/article/details/53083516](http://blog.csdn.net/qq_21190081/article/details/53083516)

3、为什么Google可以只用attention，为什么这么做。

模型中也并非只有attention。一般做法是encoder+seq2seq+decoder，，其中encoder和decoder部分都是用cnn或者rnn做。这篇文章中没有专门的attention层，而是encoder部分和decoder部分都用了attention的思想和类似算法，但是除了

attention外还有全连接层，relu层， normalization， dropout 的使用，严格来说不能说只有attention完成了整个模型的工作。

对于rnn和cnn所能捕捉到的位置信息，利用position embedding的集成完成。

这里的点乘attention可以并行化，利于加速。

作者在文中深入讨论了为什么选择使用self attention这一结构。主要从三个方面来谈。

第一是每层的总计算复杂性，其次是能并行计算的数量，这点论文用所要求的序列操作的最小值来量化。第三点是针对网络长距离依赖的路径长度。为了提升在长序列上的计算性能，self-attention应该改被限制（restrict）在一个size大小为r 的邻域内，这样可以将最大路径长度增加到 $O(n/r)$ ，至于具体如何去做作者说在未来工作中会研究。

#### 4、已有Seq2Seq模型优缺点分析：

传统的基于CNN或RNN的Seq2Seq模型都存在一定的不足：CNN不能直接用于处理变长的序列样本，RNN不能并行计算，效率低。虽然完全基于CNN的Seq2Seq模型可以并行实现，但非常占内存，很多的trick，大数据量上参数调整并不容易。传统的Seq2Seq模型不适用于长的句子，Seq2Seq+attention虽然提升了处理长句子的能力，但encoder解码得到隐变量Z时，任然对观测序列X的计算添加了约束。

基于Attention Mechanism + LSTM的Seq2Seq模型的优点：自适应地计算一个权值矩阵W，权重矩阵W长度与X的词数目一致，每个权重衡量输入序列X中每个词对输入序列Y的重要程度，不需要考虑输入序列X与输出序列Y中，词与词之间的距离关系。缺点：attention mechanism通常是和RNN结合使用，但RNN依赖t-1的历史信息来计算t

时刻的信息，因此不能并行实现，计算效率比较低，特别是训练样本量非常大的时候。

基于CNN的Seq2Seq+attention的优点：基于CNN的Seq2Seq模型具有基于RNN的Seq2Seq模型捕捉long distance dependency的能力，此外，最大的优点是可以并行化实现，效率比基于RNN的Seq2Seq模型高。缺点：计算量与观测序列X和输出序列Y的长度成正比。操作的步骤随着位置距离的增加而增加，这使得对长距离的依赖比较难处理。

## 九、beam search

beam search

search只在test的时候需要。训练的时候知道正确答案，并不需要再进行这个搜索。

test的时候，假设词表大小为3，内容为a，b，c。beam size是2

decoder解码的时候：

1：生成第1个词的时候，选择概率最大的2个词，假设为a,c,那么当前序列就是a,c

2：生成第2个词的时候，我们将当前序列a和c，分别与词表中的所有词进行组合，得到新的6个序列aa ab ac ca cb cc,然后从其中选择2个得分最高的，作为当前序列，假如为aa cb

3：后面会不断重复这个过程，直到遇到结束符为止。最终输出2个得分最高的序列。

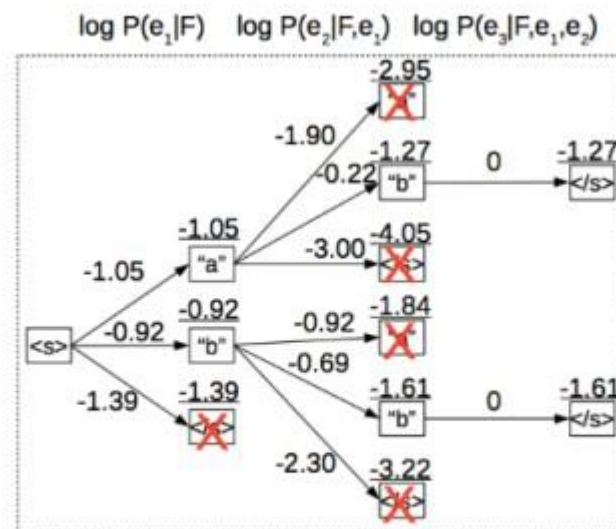


Figure 23: An example of beam search with  $b = 2$ . Numbers next to arrows are log probabilities for a single word  $\log P(e_t | F, e_1^{t-1})$ , while numbers above nodes are log probabilities for the entire hypothesis up until this point.

## 十、clip gradients

### 1. 梯度爆炸的影响

(1) 在一个只有一个隐藏节点的网络中，损失函数和权值 $w$ 偏置 $b$ 构成error surface，其中有一堵墙（梯度很大的一小段）。

损失函数每次迭代都是每次一小步，但是当遇到这堵墙时，在墙上的某点计算梯度，梯度会瞬间增大，指向某处不理想的位置。如果我们使用缩放，可以把误导控制在可接受范围内。

(2) 每一层使用接近的学习速率有利于维持网络的稳定和较好的学习方向。唯一让所有层都接近相同的学习速度的方式是所有这些项的乘积都能得到一种平衡。

### 2. 解决梯度爆炸问题的方法

clip gradients 算法：将权重控制在一定范围之内。

算法：

a. 设置一个梯度阈值：clip\_gradient

b. 求在后向传播中求出各参数的梯度的l2范数

c. 比较梯度的l2范数||g||与clip\_gradient的大小

d. 如果前者大，求缩放因子clip\_gradient/||g||，由缩放因子可以看出梯度越大，则缩放因子越小，这样便很好地控制了梯度的范围

e. 将梯度乘上缩放因子便得到最后所需的梯度

## 十一、BLEU

基于准确率，分母是预测结果总数。

Bilingual Evaluation understudy，先计算出一个n元词在一个句子中最大可能出现的次数MaxRefCount(n-gram)，然后跟候选译文中的这个n元词出现的次数作比较，取它们之间最小值作为这个n元词的最终匹配个数。

$$Count_{clip}(n-gram) = \min\{Count(n-gram), MaxRefCount(n-gram)\}$$

Count(n-gram): 某个n元词在候选译文中的出现次数

MaxRefCount(n-gram): 该n元词在参考译文中出现的最大次数

精度：

$$P_n = \frac{\sum_{C \in candidates} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C \in candidates} \sum_{n-gram \in C} Count(n-gram)}$$

惩罚因子:

$$BP = \begin{cases} 1 & \text{if } c > r \\ \text{EXP}(1 - r/c) & \text{if } c \leq r \end{cases}$$

优化Pn在语句较短时表现更好 的问题

$$BLEU = BP * \exp \left[ \left( \sum_{n=1}^N w_n \log p_n \right) \right]$$

Wn: n元词的权重,常取1/n

## 十二、平方误差和交叉熵

参考地址:<http://blog.csdn.net/yimingsilence/article/details/52740638>

### 0、一些性质

方差代价函数与叉熵代价函数同样有两个性质:

- 非负性。(所以我们的目标就是最小化代价函数)
- 当真实输出a与期望输出y接近的时候,代价函数接近于0.(比如y=0, a~0; y=1, a~1时,代价函数都接近0)。

### 1、平方误差

$$C = \frac{(y - a)^2}{2};$$

y是我们期望的输出, a为神经元的实际输出【  $a = \sigma(z)$ , where  $z = wx + b$  】。



在训练神经网络过程中，我们通过梯度下降算法来更新w和b，因此需要计算代价函数对

w和b的导数：

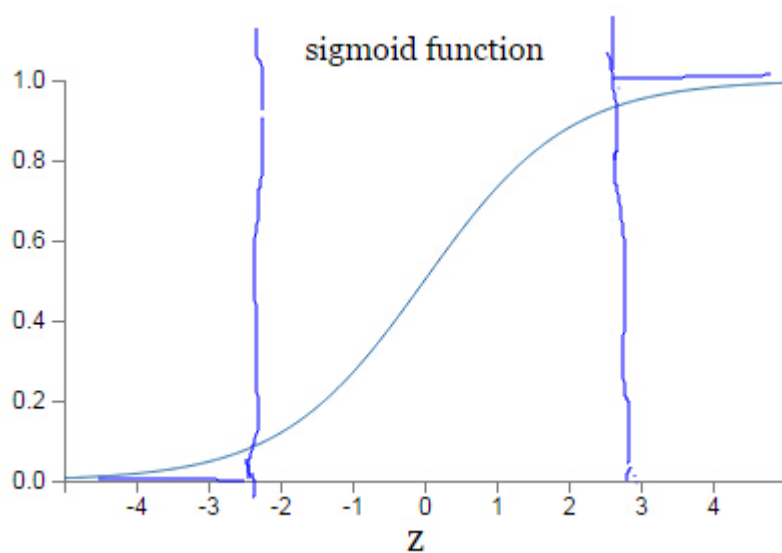
$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z) = a\sigma'(z),$$

然后更新w、b：

$$w := w - \eta * \frac{\partial C}{\partial w} = w - \eta * a * \sigma'(z)$$

$$b := b - \eta * \frac{\partial C}{\partial b} = b - \eta * a * \sigma'(z)$$

因为sigmoid函数的性质，导致 $\sigma'(z)$ 在z取大部分值时会很小（如下图标出来的两端，几近于平坦），这样会使得w和b更新非常慢（因为 $\eta * a * \sigma'(z)$ 这一项接近于0）。



## 2、交叉熵

假设：p为真实分布，q非真实分布。

熵：本质是香农信息量或编码长度的期望。

$$H(p) = \sum_i p(i) * \log \frac{1}{p(i)}$$

交叉熵：表示使用错误分布q来表示来自真实分布p的平均编码长度。

交叉熵越小，两个分布越相似。本质上也是最大似然法则。

$$H(p,q) = \sum_i p(i) * \log \frac{1}{q(i)}$$

相对熵：由q得到的平均编码长度比由p得到的平均编码长度多出的bit数

$$D(p||q) = H(p,q) - H(p) = \sum_i p(i) * \log \frac{p(i)}{q(i)}$$

交叉熵具体应用公式：

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)]$$

导数计算公式：

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y).$$

导数中没有 $\sigma'(z)$ 这一项，权重的更新是受 $\sigma(z)-y$ 这一项影响，即受误差的影响。所以当误差大的时候，权重更新就快，当误差小的时候，权重的更新就慢。这是一个很好的性质。

**实现1：**

参考地址：[http://blog.csdn.net/qw\\_sunny/article/details/72885403](http://blog.csdn.net/qw_sunny/article/details/72885403)

GNMT中代码实现：

```
tf.nn.weighted_cross_entropy_with_logits(labels, logits, pos_weight,
name=None)
```

计算具有权重的sigmoid交叉熵sigmoid\_cross\_entropy\_with\_logits ( )

argument:

\_sentinel:本质上是不要的参数，不用填

logits:一个数据类型 ( type ) 是float32或float64;

shape:[batch\_size,num\_classes],单样本是[num\_classes]

labels:和logits具有相同的type(float)和shape的张量(tensor) ,

pos\_weight:正样本的一个系数

name:操作的名字 , 可填可不填

output:

loss , shape:[batch\_size,num\_classes]

计算公式:

$$y = labels \quad p_{ij} = \text{sigmoid}(\text{logits}_{ij}) = \frac{1}{1 + e^{-\text{logits}_{ij}}}$$

$$\text{loss}_{ij} = -[\text{pos\_weight} * y_{ij} * \ln p_{ij} + (1 - y_{ij}) \ln(1 - p_{ij})]$$