基于知识共享与对抗的双向机器翻译及其翻译知识蒸馏方法

**前言：**

机器翻译技术实现从一种自然语言到另一种自然语言的自动转换。无论是统计机器翻译还是神经机器翻译，典型的翻译方法都是从左往右逐词生成目标语言句子。但我们发现，有些语言（例如汉语到英语的翻译）采用从左到右的方式更优，而有些语言（例如英语到日语的翻译）采用从右到左的方式更优；而且即使对于同一种语言翻译方向，有的句子适合从左到右的译文生成，有的句子则相反。因此，我们希望设计一种新颖的翻译模型，既可以从左往右生成也可以从右往左生成，并且能从两个模型中进行翻译知识蒸馏从而进一步提升译文质量。

1. **机器翻译模型的数学表示**

机器翻译任务中，我们一般采用下面的式子建模目标语言文本与源语言文本之间的关系：

这就是典型的从左往右的机器翻译方法，其中表示翻译模型参数。当然，也可以采用从右往左的译文生成范式：

其中，表示从右往左翻译方法的模型参数。通常情况下，基于编码器-解码器的神经机器翻译方法需要分别采用两套模型（即两套参数和）才能实现从左往右和从右往左的译文生成。

1. **基于知识共享与对抗的双向机器翻译模型**

如果采用两套翻译模型参数，一方面浪费存储空间，另一方面没有充分挖掘两套模型之间的共享知识。我们提出一种基于知识（模型参数）共享的神经机器翻译方法，也即一套参数实现两个方向的译文生成。

我们的想法可以由图1简要说明。不同于传统的从左到右和从右到左解码模型分别需要优化编码器-解码器神经网络参数和，我们的方法采用同一套参数既可以实现从左到右的翻译，也可以实现从右到左的翻译。特别地，我们在解码的初始时刻利用特殊符号和分别指示当前采用的是从左到右翻译还是从右到左翻译。测试时，我们可以采用从左往右的解码方法，也可以选择从右往左的解码方法，并且也可以从两种结果中选择更优的作为最终译文结果。

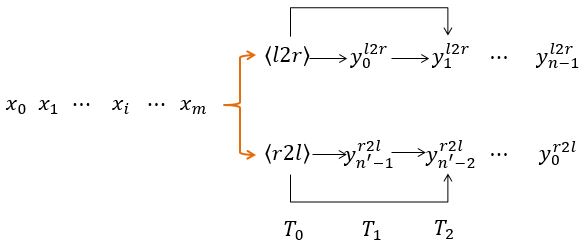


图1：基于知识共享与对抗的双向翻译简单示意图

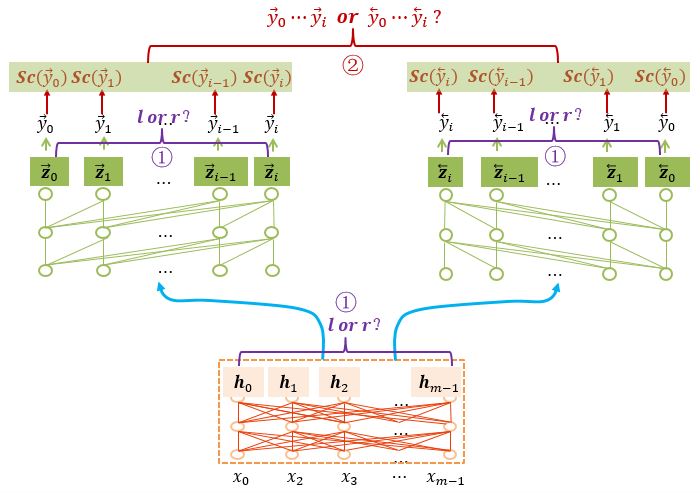


图2：基于知识共享与对抗的双向翻译模型架构

下面，我们通过图2详细介绍知识共享与对抗以及动态译文选择的模型架构。

1. 知识共享模型

在知识共享模型中，不同于传统的从左往右和从右往左的编码器-解码器神经机器翻译采用两套编码和解码参数和，我们采用同一套编码和解码参数（图2中相同的颜色表示采用相同的参数，如橙色的编码网络和绿色的解码网络）。形式化地，传统从左往右解码方法计算第个词语的概率采用，传统从右往左解码方法计算第个词语的概率采用。和之间没有独立训练，没有任何关系，然而由于仅仅是翻译方向不同，源语言编码和目标语言解码绝大部分知识是可以共享的。因此，我们采用一套编码-解码参数，计算和。

1. 翻译对抗模型

在基于同一套编码-解码参数的双向翻译模型中，翻译知识是完全共享的，即使解码时给定一个从左往右或从右往左的标签和作为解码方向的提示，仍然存在一个风险：由于参数共享，编码器和解码器不知道当前的解码方向，可能导致无法完美区分从左往右和从右往左的范式。

针对上述问题，我们提出一种翻译对抗模型，如图2中编码和解码网络中的部分。基本思想是：基于参数共享的编码器和解码器本质上是不用区分从左往右和从右往左的解码方向，但实际上我们是希望编码器和解码器在实际翻译过程中对翻译方向是敏感的，这样能够生成更好的译文。为此，我们设计一种生成对抗模型，即在编码器和解码器的最上层加入判别函数，使得每个最上层的隐层表示能够具有翻译方向的敏感性：

其中表示编码器或解码器的顶层隐层表示，编码器时，解码器时，是一个卷积神经网络，将顶层表示进行一个变换得到，然后利用预测翻译方向（或）。该模型的目标是如果当前执行从左往右解码，则希望，如果执行从右往左解码，则希望。

模型参数训练时，给定包含个平行句对的训练数据集，基于知识完全共享的模型旨在学习一套参数以最大化训练数据的对数似然：

其中，=，即在原始目标语言句子的开始位置加入一个特殊符号；是的逆序表示，并在开始位置加入另一个特殊符号。

除了上述目标函数外，我们加入翻译对抗损失函数：

其中是示性函数，当解码方向是从左往右时为1否则为0。上述目标函数表示如果当前训练是从左往右解码，则最大化概率，否则最大化概率。

1. 动态译文选择模型

基于知识共享和对抗模型，在翻译时我们可以同时获得从左往右的译文和从右往左的译文。虽然在训练时，模型充分融合了从左往右和从右往左的翻译知识，但是测试时两个译文中必定有一个是更好的。解码结束时，该选择哪个翻译方向的译文应该取决于不同的源语言句子和翻译过程。

针对上述问题，我们提出一种动态译文打分和全局译文选择的方法。如图2中最上面的②所示，在译文生成过程中，无论从左往右还是从右往左，我们动态地为每个时刻的译文计算一个得分，根据所有时刻的译文得分和翻译方向的对抗得分我们设计一个全局打分函数。动态局部打分函数和全局译文打分函数设计如下：

其中表示目标端除了的剩余部分，表示基于源语言句子和目标译文端除的剩余部分利用自注意力机制对当前时刻的译文进行打分。表示根据每个时刻的译文得分以及翻译方向的对抗得分利用前馈神经网络计算译文的全局得分。解码结束后，如果，则我们选择从左往右的翻译结果，否则我们选择从右往左的翻译结果。

上述局部打分函数和全局打分函数的参数训练目标如下：

上述目标函数是局部打分的目标，即给定双语句对，我们希望每个时刻利用自我注意力机制预测的得分都是最高的。

对于全局打分函数，给定双语平行数据，我们训练知识共享和对抗的模型，然后利用训练的模型对平行数据的源语言解码，分别得到左往右的译文结果和从右往左的译文结果。然后，利用和与标准答案进行对比打分，分别得到和的匹配得分和。若，我们则希望优化参数使得，否则优化参数使得。

我们提出知识共享和对抗模型以及译文选择方法有三点显著的优势：1，与传统从左往右的神经翻译方法相比，通过知识共享和翻译对抗实现从左往右和从右往左的解码；2，基于神经网络参数（也即翻译知识）共享机制可以增强从左往右和从右往左的模型能力；3，依据我们设计的局部与全局打分函数，我们可以动态地选择从左往右或从右往左的结果中更优的译文。

1. **基于知识蒸馏的翻译模型增强**

假设基于知识共享的双向翻译模型在双语平行数据上进行有监督学习和参数优化，得到。然后，我们采用解码训练数据中的每一个源语言句子，从而得到从左往右的译文结果和从右往左的译文结果。每一个源语言句子将对应三种可能的译文结果、和，形成三组候选、、，从而设计基于知识蒸馏的方法优化翻译模型。例如，可以通过下述的方法优化从左往右的翻译模型：

其中，为示性函数，当时，否则。第一项表示传统在标准数据上从左往右翻译方法训练的优化目标。表示双向翻译模型中从左往右模型预测词汇的概率，其中的式子表示知识蒸馏，希望概率分布与已经训练的双向翻译模型生成时具有一致的词汇预测概率。最后一项表示从右往左翻译模型的知识蒸馏。通过知识蒸馏，从左往右的翻译模型可以充分利用标准训练数据以及双向翻译模型的知识。

在实际使用中，可以简化上述的优化公式，即假设和。

1. **实验结果**

我们以Transformer为机器翻译的基线系统，在汉语到英语的翻译任务上进行了测试。汉语-英语采用约200万的双语训练数据，开发集为NIST02，测试集为NIST03-06。我们采用BLEU值对比不同系统的优劣。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | DEV | NIST03 | NIST04 | NIST05 | NIST06 | AVG |
| Transformer | 48.12 | 47.63 | 48.32 | 47.51 | 45.31 | 47.19 |
| Bi-Transformer(l2r) | 48.53 | 49.78 | 48.56 | 48.06 | 46.18 | 48.15 |
| KD-l2r | 50.19 | 50.24 | 50.04 | 49.76 | 47.88 | 49.48 |
| KD-l2r-r2l | **50.98** | **50.82** | **50.81** | **50.36** | **47.99** | **50.00** |
| SBNMT | 50.99 | 51.87 | 51.50 | 51.23 | 48.83 | 51.11 |

从上表中的实验结果可以看出，基于知识共享的双向机器翻译方法Bi-Transformer(l2r)训练后采用从左往右的解码方式能够显著提升译文质量，相比于标准基线系统提升约1个BLEU值（48.15 vs. 47.19）。如果再加上知识蒸馏策略，性能能够进一步显著提升。如果只采用双向模型中从左往右的模型KD-l2r进行知识蒸馏，平均性能提升2.29个BLEU值（49.48 vs. 47.19）。若进一步加入从右往左的模型KD-l2r-r2l进行知识蒸馏，译文质量进一步提升了2.81.个BLEU值（50.00 vs. 47.19）。并且，可以发现从汉语到英语的翻译性能提升方面我们的模型稍微弱于SBNMT。

值得说明的是，我们的方法无需修改网络结构，因此与基线系统拥有相同的解码效率，比SBNMT高效很多，训练和测试效率对比可见下表：

|  |  |  |
| --- | --- | --- |
| Model | Train (batch/s) | Test (sent/s) |
| Transformer | 2.07 | 20 |
| SBNMT | 1.26 | 17.8 |
| KD | 2.07 | 20 |

其中，训练中我们以每秒钟完成多少个batch（每个batch取4096个词语）的参数更新为度量标准，可以发现，我们的模型与基线Transformer系统是一样高效的，每秒钟能够更新2.07个batch。相比而言，SBNMT的训练过程要慢很多。测试中，我们以每秒钟翻译多少个句子为度量标准，同样发现，我们模型的解码速度与基线Transformer一样，而SBNMT的解码速度要明显低于我们的KD模型。

为了验证Bi-Transformer+KD模型在不同语言翻译上的性能，我们又在标准的WMT评测数据英语到德语翻译上进行了测试。训练数据为450万句对，测试集为WMT2014的英德测试集，详细的对比结果如下：

|  |  |
| --- | --- |
| Model | Test |
| Transformer | 27.72 |
| SBNMT | 29.21 |
| KD | **29.29** |

可以发现，在英语到德语的翻译方向上，我们的模型相比于基线Transformer，性能提升了1.57个BLEU值，甚至还要稍稍由于SBNMT模型。可见，我们的模型适用于多个数据集。

1. **总结**

我们提出了一种基于知识共享的双向机器翻译模型，采用一套编解码神经网络实现从左往右和从右往左的两种译文生成方式，提供更多的解码方式，并提升译文质量。以双向机器翻译模型为基础，通过知识蒸馏可以非常显著地提升机器翻译的译文质量，同时不影响译文生成过程中的解码效率。

基于知识共享的双向机器翻译及其翻译知识蒸馏方法

代码使用说明

1. 基本概括，代码文件包含4个子文件夹：

casia-nmt/casia-nmt-share：用于基于知识共享的双向机器翻译模型的（1）训练和（2）双向译文的产生。

casia-nmt/casia-nmt-distillation：用于基于知识蒸馏的翻译模型增强的（1）训练和（2）测试。

casia-nmt/data-share: 用于基于知识共享的双向机器翻译模型的数据管理。

casia-nmt/ data-distillation: 用于基于知识蒸馏的翻译模型增强的数据管理。

2. 使用步骤：

2.1 训练基于知识共享的双向机器翻译模型。

（1）语料预处理，根据翻译语言对不同而不同，比如中文包括分词，BPE亚词切分等操作，英文包括token, 大写转小写等操作。

（2）词表生成，进入casia-nmt/data-share，使用get\_vocab\_share.py生成词表文件，可参考运行脚本./create\_share\_vocab.sh。

（3）将数据转成tensorflow格式，具体地，进入casia-nmt/casia-nmt-share运行脚本datagen.sh。

python run.py \

--generate\_data=True \

--data\_dir=./data \

--tmp\_dir=./data-tmp \

--train\_src\_name=2m.bpe.zh \

--train\_tgt\_name=2m.bpe.en \

--vocab\_src\_size=30720 \

--vocab\_tgt\_size=30720 \

--vocab\_src\_name=vocab.zh \

--vocab\_tgt\_name=vocab.en \

--num\_shards=25

其中generate\_data保持True,

data\_dir为生成tensorflow格式数据的文件夹名，

tmp\_dir为原始数据的文件夹名，

train\_src\_name为源端数据，

train\_tgt\_name为目标端数据，

vocab\_src\_size为源端词表大小，

vocab\_src\_name为源端词表，

vocab\_tgt\_name为目标端词表，

num\_shards为数据切分成n份保存，一般取1到100，根据语料大写自己定。

（4）训练基于知识共享的双向机器翻译模型，运行train.sh

export CUDA\_VISIBLE\_DEVICES=3

nohup python run.py \

--worker\_gpu=1 \

--gpu\_mem\_fraction=0.9 \

--hparams='batch\_size=4096, shared\_source\_embedding\_and\_softmax\_weights=0 ' \

--data\_dir=../data \

--vocab\_src\_size=30720 \

--vocab\_tgt\_size=30720 \

--vocab\_src\_name=vocab.zh \

--vocab\_tgt\_name=vocab.en \

--hparams\_set=transformer\_params\_base \

--train\_steps=200000 \

--keep\_checkpoint\_max=2 \

--output\_dir=../train-v21-base > log.train &

worker\_gpu为使用GPU的个数，

gpu\_mem\_fraction=0.9 为占显存比例，

hparams为一些超参数设置，比如batch\_size=4096，

data\_dir为datagen.sh步骤中生成tensorflow格式数据的文件夹

hparams\_set为模型参数设置，一般为transformer\_params\_base或者transformer\_params\_big

train\_steps为训练步数，

keep\_checkpoint\_max为最大保存模型数，

output\_dir为模型输出路径。

2.2 使用训练好的双向机器翻译模型进行解码。

（1）解码训练集源语言生成L2R译文，运行test.sh

export CUDA\_VISIBLE\_DEVICES=3

python run.py \

--gpu\_mem\_fraction=0.6 \

--hparams=' shared\_source\_embedding\_and\_softmax\_weights=0 ' \

--data\_dir=../data \

--hparams\_set=transformer\_params\_base \

--output\_dir=../train \

--vocab\_src\_size=30720 \

--vocab\_tgt\_size=30720 \

--vocab\_src\_name=vocab.zh \

--vocab\_tgt\_name=vocab.en \

--train\_steps=0 \

--decode\_beam\_size=4 \

--decode\_alpha=0.6 \

--decode\_batch\_size=50 \

--is\_decode\_l2r=True \

--decode\_from\_file=../data/03.seg.bpe \

--decode\_to\_file=../output/dev..tmp.out

train\_steps设置为0 表示当前为解码，

decode\_beam\_size为解码的beam size 大小

decode\_alpha为长度惩罚

decode\_batch\_size为解码的batch大小

**is\_l2r\_decoding为True表示译文生成为从左至右；为False表示译文生成为从右至左。**

decode\_from\_file为解码的源文件

decode\_to\_file为解码的输出文件

其他参数同上。

（2）解码训练集源语言生成R2L译文，将test.sh中的is\_l2r\_decoding置为False，再次运行./test.sh

2.3 训练基于知识蒸馏的翻译模型增强

（1）将训练集，词表，和生成的L2R，R2L译文复制到文件夹casia-nmt/data-distillation下，合成增强的语料，运行cat.sh

##这里假设src，tgt分别为源语言和目标语言，l2r.out和r2l.out分别为步骤2.2中生成的正向和反向翻译结果。

python reverse\_remove.py r2l.out r2l.out\_std

python remove.py l2r.out l2r.out\_std

cat src src src > train.dis.src

cat tgt l2r.out\_std r2l.out\_std > train.dis.tgt

（2）将数据转成tensorflow格式，具体地，进入casia-nmt/casia-nmt-distillation文件夹，运行脚本datagen.sh

python run.py \

--generate\_data=True \

--data\_dir=./data \

--tmp\_dir=./data-tmp \

--train\_src\_name=2m.bpe.zh \

--train\_tgt\_name=2m.bpe.en \

--vocab\_src\_size=30720 \

--vocab\_tgt\_size=30720 \

--vocab\_src\_name=vocab.zh \

--vocab\_tgt\_name=vocab.en \

--num\_shards=25

参数含义同上，

这里的train\_src\_name 和train\_tgt\_name对应增强的训练集。

（3）训练基于知识蒸馏的翻译模型增强，运行./train.sh

export CUDA\_VISIBLE\_DEVICES=3

nohup python run.py \

--worker\_gpu=1 \

--gpu\_mem\_fraction=0.9 \

--hparams='batch\_size=4096, shared\_source\_embedding\_and\_softmax\_weights=0' \

--data\_dir=../data-dis \

--vocab\_src\_size=30720 \

--vocab\_tgt\_size=30720 \

--vocab\_src\_name=vocab.zh \

--vocab\_tgt\_name=vocab.en \

--hparams\_set=transformer\_params\_base \

--train\_steps=200000 \

--keep\_checkpoint\_max=2 \

--output\_dir=../train-dis> log.train-dis &

（4）使用训练好的基于知识蒸馏的翻译模型进行解码，运行./test.sh

export CUDA\_VISIBLE\_DEVICES=3

python run.py \

--gpu\_mem\_fraction=0.6 \

--hparams=' shared\_source\_embedding\_and\_softmax\_weights=0' \

--data\_dir=../data \

--hparams\_set=transformer\_params\_base \

--output\_dir=../train \

--vocab\_src\_size=30720 \

--vocab\_tgt\_size=30720 \

--vocab\_src\_name=vocab.zh \

--vocab\_tgt\_name=vocab.en \

--train\_steps=0 \

--decode\_beam\_size=4 \

--decode\_alpha=0.6 \

--decode\_batch\_size=50 \

--decode\_from\_file=../data/03.seg.bpe \

--decode\_to\_file=../output/dev..tmp.out

参数含义同上