Lesson 11
Neural Machine Translation

2018年7月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

文章推荐

- The 1cycle policy

  1. 循环学习率

  2. 超收敛（高学习率快速收敛）

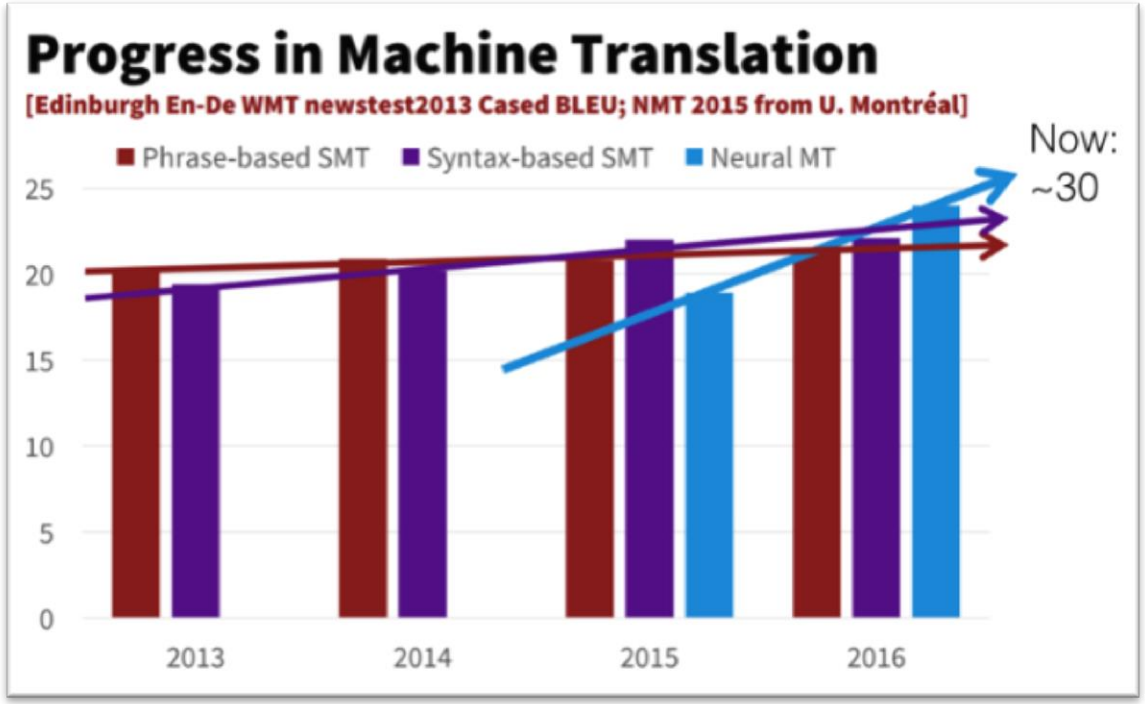
- How To Create Data Products That Are Magical Using Sequence-to-Sequence Models

  训练一个模型总结github上的问题

目录 Contents

主流机器翻译

- PBMT:基于短语的统计机器翻译
- SBMT:基于语法的统计机器翻译
- NMT:神经网络机器翻译



神经网络机器翻译(NMT)的优势

- 端到端的训练——所有参数同时优化以使得输出的loss最小化
- 构建分布式表示——更好地利用词汇和短语的相似度
- 更全面地利用上下文进行准确翻译
- 更流畅地构建文本

Four big wins of Neural MT

1. **End-to-end training**
   All parameters are simultaneously optimized to minimize a loss function on the network's output
2. **Distributed representations share strength**
   Better exploitation of word and phrase similarities
3. **Better exploitation of context**
   NMT can use a much bigger context – both source and partial target text – to translate more accurately
4. **More fluent text generation**
   Deep learning text generation is much higher quality

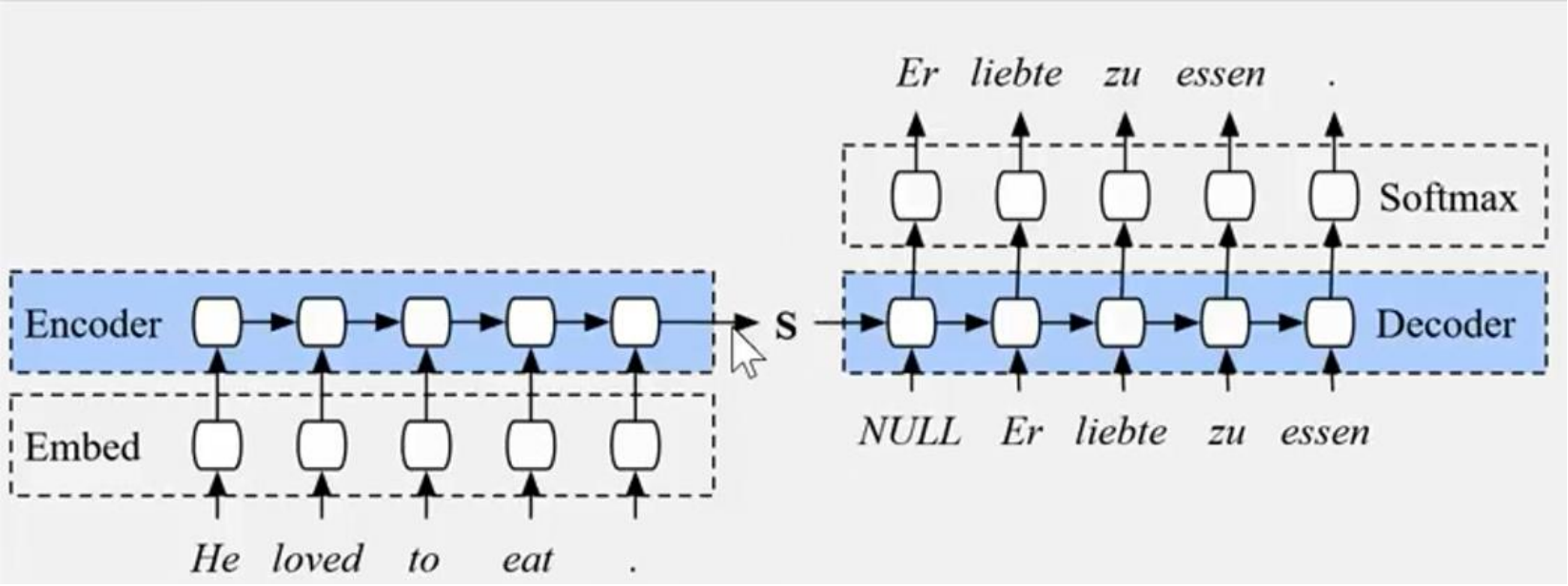
数据

- 本课程的目的是构建英语和法语的翻译网络
- 不同于IMDB中的语言模型，这里采用了平行语料库
- 平行语料库(parallel texts)：两种语言分别对应(x,y)
- 本课程采用的语料库是来自http://www.statmt.org/wmt15/translation-task.html英语-法语语料库

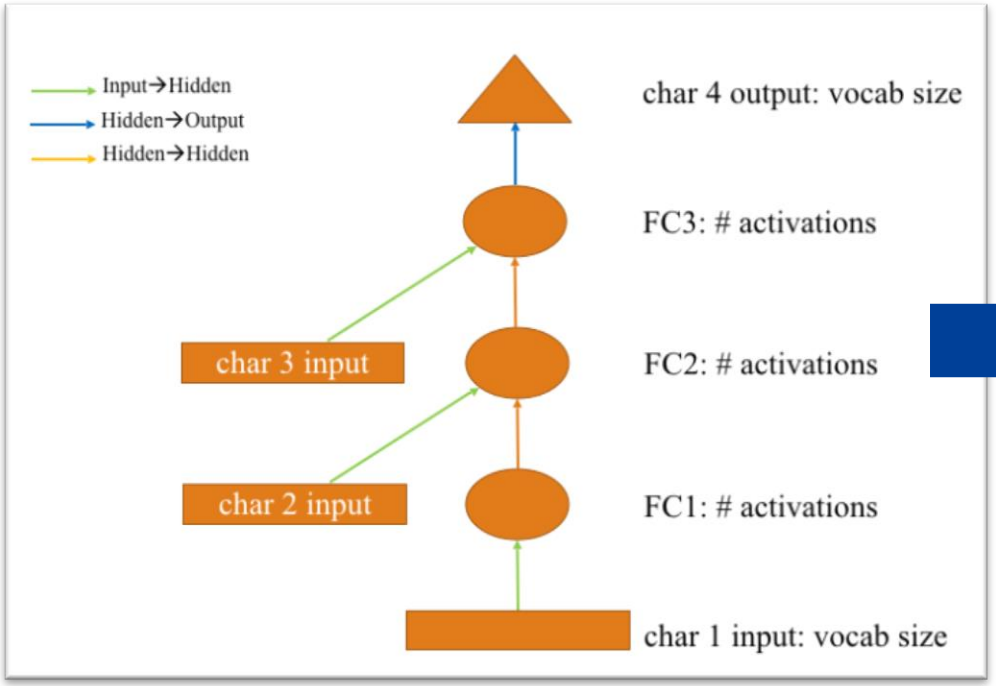
In [6]:
```
PATH = Path('data/translate')
TMP_PATH = PATH/'tmp'
TMP_PATH.mkdir(exist_ok=True)
fname='giga-fren.release2.fixed'
en_fname = PATH/f'{fname}.en'
fr_fname = PATH/f'{fname}.fr'
```

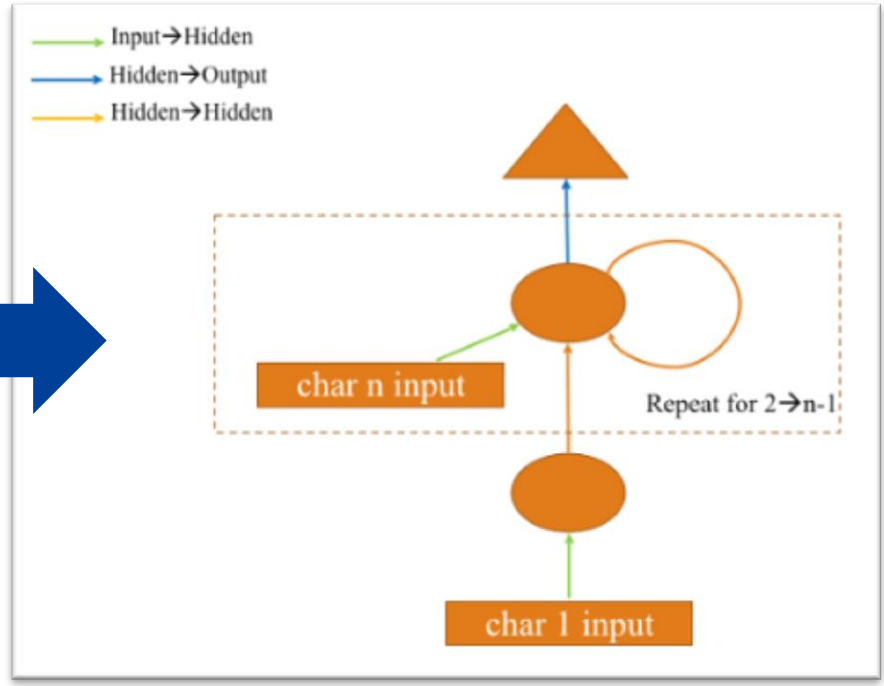
Sequence to sequence 模型

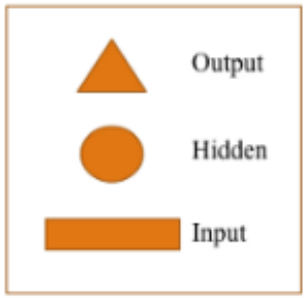
- 由编码器和解码器构成，本质上是两个RNN
- 由输入得到隐藏状态，再由隐藏状态翻译得到输出
- 翻译模型的难点：输入输出的顺序、长度都可能不同



回顾RNN：多输入模型

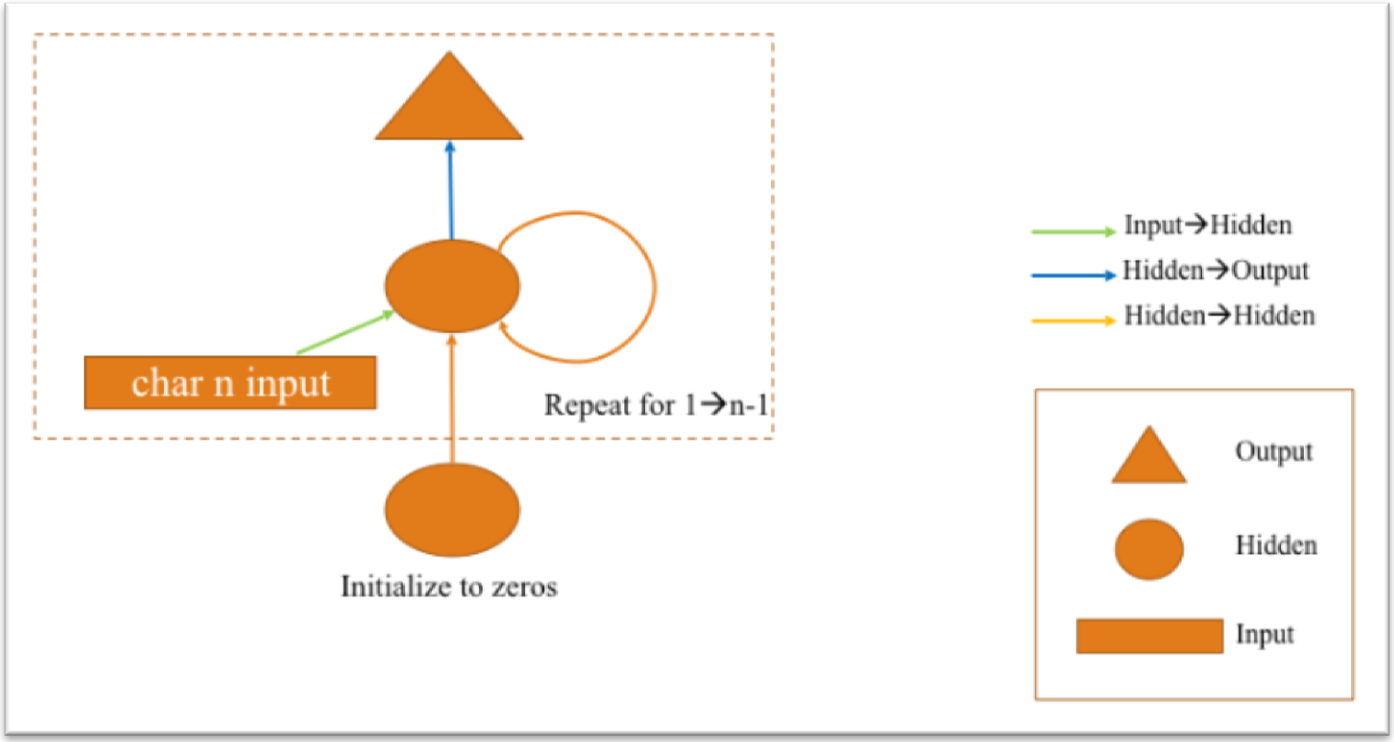

Input→Hidden
Hidden→Output
Hidden→Hidden
char 4 output: vocab size
FC3: # activations
FC2: # activations
FC1: # activations
char 3 input
char 2 input
char 1 input: vocab size


Input→Hidden
Hidden→Output
Hidden→Hidden
char n input
Repeat for 2→n-1
char 1 input


Output
Hidden
Input

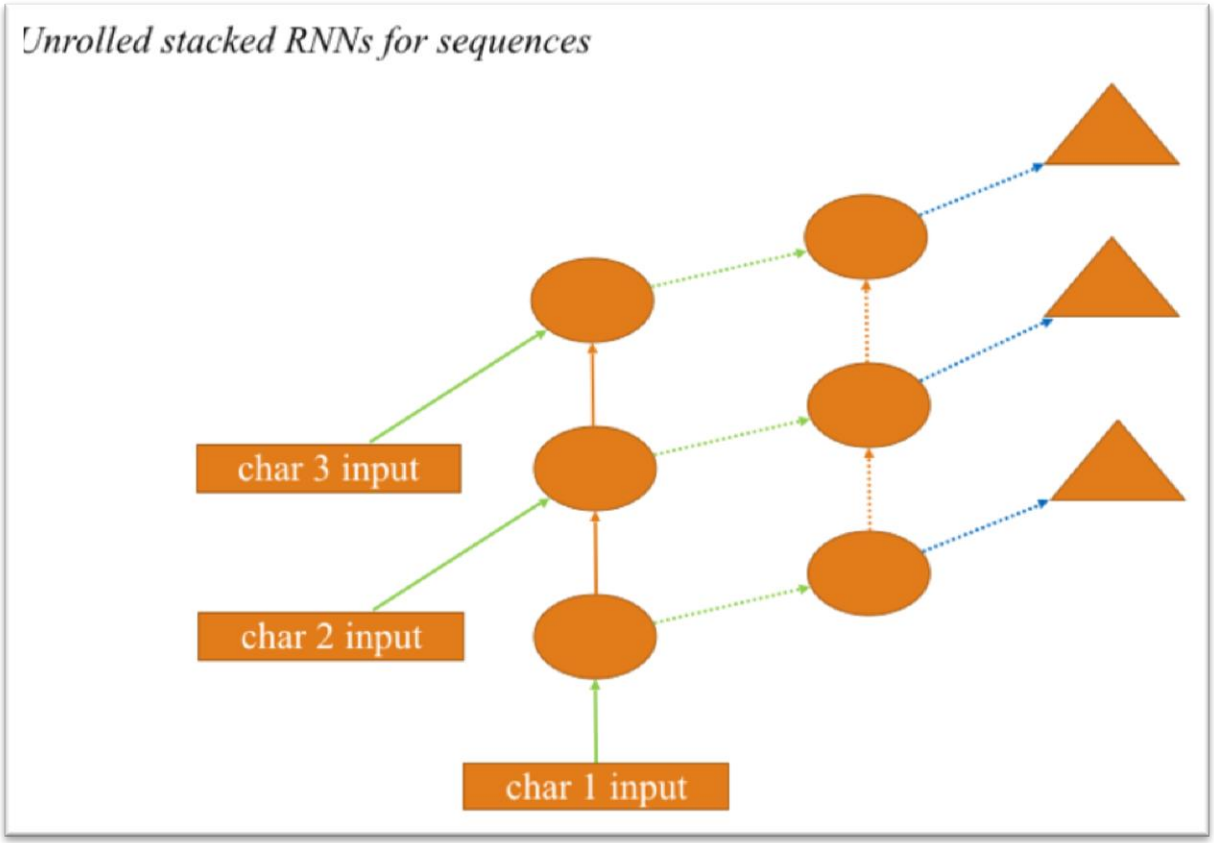
回顾RNN：多输出模型

- 每一次循环都输出一个output



回顾RNN：多层RNN

- 将前一个RNN的隐层作为后一个RNN的输入



预处理：翻译问题的简化

- 为了简化训练所占的资源，这里的RNN只采用了两层（谷歌翻译有8层）
- 所以，对所需训练的翻译问题进行相应的简化
- 只进行针对于英语问题和法语问题的翻译
- 步骤：在语料库中搜寻以“wh”开头，“？”结尾的句子
- 优点：问句长度较短，且容易关联学习

```
re_eq = re.compile('^(Wh[^?.!]+\?)')
re_fq = re.compile('^([^?.!]+\?)')

lines = ((re_eq.search(eq), re_fq.search(fq))
         for eq, fq in zip(open(en_fname, encoding='utf-8'), open(fr_fname, encoding='utf-8')))

qs = [(e.group(), f.group()) for e,f in lines if e and f]
```

预处理：将英语和法语词语进行标记

- 用spcay库对单词、符号等进行独立区分与标记
- 这里还需要用python -m spacy download fr下载spacy库的法语模型

```
en_tok = Tokenizer.proc_all_mp(partition_by_cores(en_qs))
```

```
fr_tok = Tokenizer.proc_all_mp(partition_by_cores(fr_qs), 'fr')
```

```
en_tok[0], fr_tok[0]
```

```
(['what', 'is', 'light', '?'],
 ['qu' ', 'est', '-ce', 'que', 'la', 'lumière', '?'])
```

预处理：将英语和法语词语进行标记

- 建立词汇表，取40000个频率最高的词语标记为数字，40000以外的单词记为unknown
- 对起始(_bos_)、填充(_pad_)、结束(_eos_)加入额外的标记
- 以此建立起词汇与数字的映射表

```
def toks2ids(tok,pre):
    freq = Counter(p for o in tok for p in o)
    itos = [o for o,c in freq.most_common(40000)]
    itos.insert(0, '_bos_')
    itos.insert(1, '_pad_')
    itos.insert(2, '_eos_')
    itos.insert(3, '_unk')
    stoi = collections.defaultdict(lambda: 3, {v:k for k,v in enumerate(itos)})
    ids = np.array([([stoi[o] for o in p] + [2]) for p in tok])
    np.save(TMP_PATH/f'{pre}_ids.npy', ids)
    pickle.dump(itos, open(TMP_PATH/f'{pre}_itos.pkl', 'wb'))
    return ids,itos,stoi
```

单词的嵌入向量

- 这里采用的词汇的嵌入向量为fast.text，其他可用的也有Word2vec
- 可以通过

! pip install git+https://github.com/facebookresearch/fastText.git

并下载‘bin plus text’来下载单词向量

- 将语言模型表示成一个标准的字典，当查找一个单词时，会返回其向量的维数，这里的英语和法语的向量维数都是300，向量的均值为0，标准差为0.3

```
dim_en_vec = len(en_vecd[','])
dim_fr_vec = len(fr_vecd[','])
dim_en_vec, dim_fr_vec
```

```
(300, 300)
```

```
en_vecs = np.stack(list(en_vecd.values()))
en_vecs.mean(), en_vecs.std()
```

```
(0.0075652334, 0.29283327)
```

关于模型数据对象

- 通常大规模语料库的末尾会有一段无用序列（如序列长度等信息），因此这里在末尾截取语料库的前90%+并省略后续段落

```
enlen_90 = int(np.percentile([len(o) for o in en_ids], 99))
frlen_90 = int(np.percentile([len(o) for o in fr_ids], 97))
enlen_90, frlen_90
```

- 针对pytorch的数据对象特点，这里定义函数以获取长度和对象参数

```
en_ids_tr = np.array([o[:enlen_90] for o in en_ids])
fr_ids_tr = np.array([o[:frlen_90] for o in fr_ids])
```

```
class Seq2SeqDataset(Dataset):
    def __init__(self, x, y): self.x,self.y = x,y
    def __getitem__(self, idx): return A(self.x[idx], self.y[idx])
    def __len__(self): return len(self.x)
```

训练集

- 采用随机数的方法创建训练集

```
np.random.seed(42)
trn_keep = np.random.rand(len(en_ids_tr))>0.1
en_trn,fr_trn = en_ids_tr[trn_keep],fr_ids_tr[trn_keep]
en_val,fr_val = en_ids_tr[~trn_keep],fr_ids_tr[~trn_keep]
len(en_trn),len(en_val)
```

```
(45219, 5041)
```

- 创建包含输入输出关系的dataset，这里是将法语翻译为英语，如果想要将英语翻译成法语，只需使其反向即可

```
trn_ds = Seq2SeqDataset(fr_trn,en_trn)
val_ds = Seq2SeqDataset(fr_val,en_val)
```

创建dataloaders

- 这里涉及的操作有转制和短句填充
- 由于要进行语句的翻译，不能像影评模型中随机设置等长的断点，因此我们要将短句进行一定的填充
- 为了节省资源，先将句子长短进行排序，使相邻句子的长度相近，这样子填充时就只需要填充一小部分
- 不同于影评模型的前置填充，这里采用后置的填充

```
bs=125
```

```
trn_samp = SortishSampler(en_trn, key=lambda x: len(en_trn[x]), bs=bs)
val_samp = SortSampler(en_val, key=lambda x: len(en_val[x]))
```

```
trn_dl = DataLoader(trn_ds, bs, transpose=True, transpose_y=True, num_workers=1,
                    pad_idx=1, pre_pad=False, sampler=trn_samp)
val_dl = DataLoader(val_ds, int(bs*1.6), transpose=True, transpose_y=True, num_workers=1,
                    pad_idx=1, pre_pad=False, sampler=val_samp)
md = ModelData(PATH, trn_dl, val_dl)
```

编码器Encoder

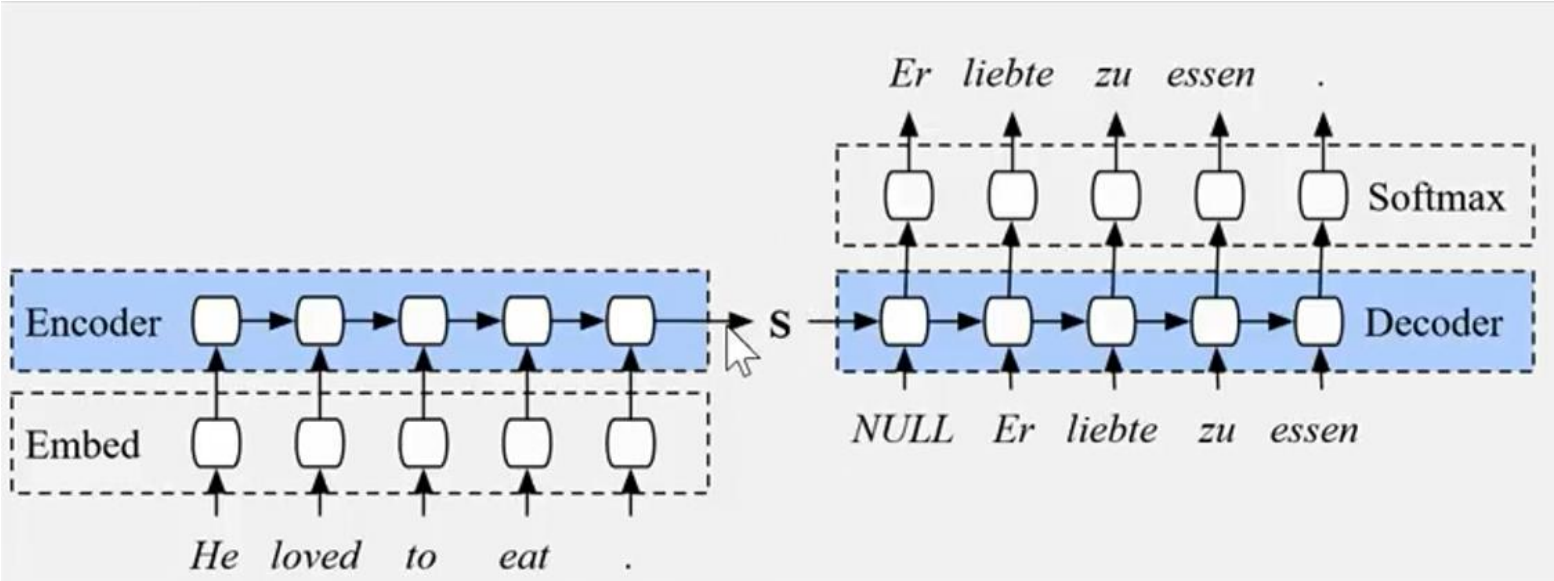
- 将文本的标记索引传入encoder，获取嵌入向量，然后传入RNN网络进行计算，得到最后的隐层参数

```
def create_emb(vecs, itos, em_sz):
    emb = nn.Embedding(len(itos), em_sz, padding_idx=1)
    wgts = emb.weight.data
    miss = []
    for i,w in enumerate(itos):
        try: wgts[i] = torch.from_numpy(vecs[w]*3)
        except: miss.append(w)
    print(len(miss),miss[5:10])
    return emb
```

- Create_emb:根据fast.text来创建嵌入层的大小（300）
- Nn.embedding:创建随机的嵌入层参数，均值0，标准差1
- 若对应词语在fast.text中已有嵌入向量的值，则将其乘以3来替换随机值（乘以3的原因是要使其参数的标准差一致）

解码器decoder

- 将编码器输出的隐层参数输入解码器
- 由上一个翻译得到的词语作为嵌入层的输入（因此第一个的输入为null）
- 输出为维度是所有独立词语大小的张量，每个参数对应翻译为每一个词语的概率（如词汇表中共5000个词，则输出5000*1的tensor，每个对应概率）



```
class Seq2SeqRNN(nn.Module):
    def __init__(self, vecs_enc, itos_enc, em_sz_enc, vecs_dec, itos_dec, em_sz_dec, nh, out_sl, nl=2):
        super().__init__()
        self.nl,self.nh,self.out_sl = nl,nh,out_sl
        self.emb_enc = create_emb(vecs_enc, itos_enc, em_sz_enc)
        self.emb_enc_drop = nn.Dropout(0.15)
        self.gru_enc = nn.GRU(em_sz_enc, nh, num_layers=nl, dropout=0.25)
        self.out_enc = nn.Linear(nh, em_sz_dec, bias=False)

        self.emb_dec = create_emb(vecs_dec, itos_dec, em_sz_dec)
        self.gru_dec = nn.GRU(em_sz_dec, em_sz_dec, num_layers=nl, dropout=0.1)
        self.out_drop = nn.Dropout(0.35)
        self.out = nn.Linear(em_sz_dec, len(itos_dec))
        self.out.weight.data = self.emb_dec.weight.data

    def forward(self, inp):
        sl,bs = inp.size()
        h = self.initHidden(bs)
        emb = self.emb_enc_drop(self.emb_enc(inp))
        enc_out, h = self.gru_enc(emb, h)
        h = self.out_enc(h)

        dec_inp = V(torch.zeros(bs).long())
        res = []
        for i in range(self.out_sl):
            emb = self.emb_dec(dec_inp).unsqueeze(0)
            outp, h = self.gru_dec(emb, h)
            outp = self.out(self.out_drop(outp[0]))
            res.append(outp)
            dec_inp = V(outp.data.max(1)[1])
            if (dec_inp==1).all(): break
        return torch.stack(res)

    def initHidden(self, bs): return V(torch.zeros(self.nl, bs, self.nh))
```

定义encoder

定义decoder

1. 循环大小：最大英语句子长度
2. Dec_inp:单词索引参数，为1时(padding)即代表结束
3. Outp：大小为所有单词数的tensor，表示概率
4. Outp.data.max：取最大概率位置的索引

Loss function

- Loss function本质是分类交叉熵损失
- 这里做了一些小的调整

1. 如果生成的序列长度小于目标的序列长度，需要添加一些填充。
2. F.cross_entropy期望一个2阶张量，但是我们有按批次大小排列的序列长度，所以要把它拉平。这就是view(-1，…)的作用。

```
def seq2seq_loss(input, target):
    sl,bs = target.size()
    sl_in,bs_in,nc = input.size()
    if sl>sl_in: input = F.pad(input, (0,0,0,0,0,sl-sl_in))
    input = input[:sl]
    return F.cross_entropy(input.view(-1,nc), target.view(-1))#, ignore_index=1)
```

- 之后调用fit，lr_find等函数进行训练与拟合，得到trn_loss2.87 & val_loss3.58

Test

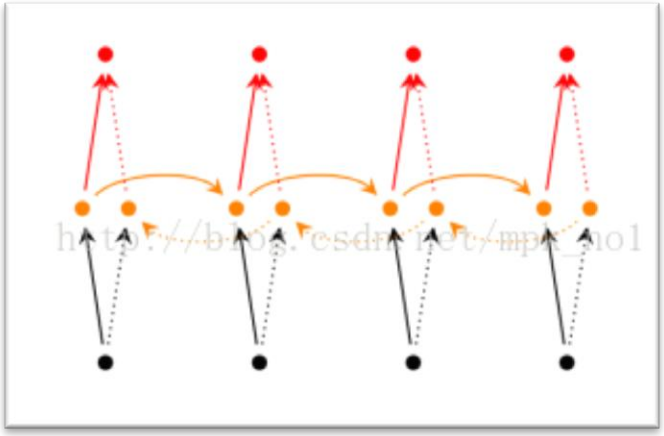
```
x,y = next(iter(val_dl))
probs = learn.model(V(x))
preds = to_np(probs.max(2)[1])

for i in range(180, 190):
    print(' '.join([fr_itos[o] for o in x[:,i] if o != 1]))
    print(' '.join([en_itos[o] for o in y[:,i] if o != 1]))
    print(' '.join([en_itos[o] for o in preds[:,i] if o!=1]))
    print()
```

```
quels facteurs pourraient influer sur le choix de leur emplacement ? _eos_
what factors influencetheir location ? _eos_
what factors might might influence on the their ? ? _eos_

qu' est -ce qui ne peut pas changer ? _eos_
what can not change ? _eos_
what not change change ? _eos_

que faites - vous ? _eos_
what do you do ? _eos_
what do you do ? _eos_

qui réglemente les pylônes d' antennes ? _eos_
who regulates antenna towers ? _eos_
who regulates the doors doors ? _eos_

où sont - ils situés ? _eos_
where are they located ? _eos_
where are the located ? _eos_
```

表现一般

技巧1：采用双向RNN

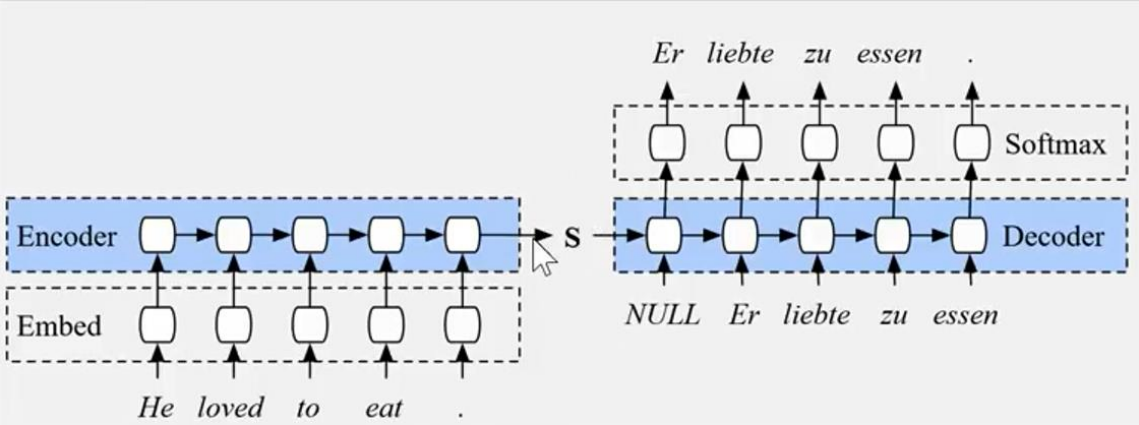
- 双向RNN是由两个RNN上下叠加在一起组成的。输出由这两个RNN的状态共同决定。即：根据前文与后文同时预测当前词语



- 这里采用bidirectional=True在encoder中添加双向rnn的方式
- 为什么不在decoder中添加双向rnn？——难以实现，cheating。
- 训练结果的交叉熵损失相比单向RNN有微小的降低，可以达到3.51

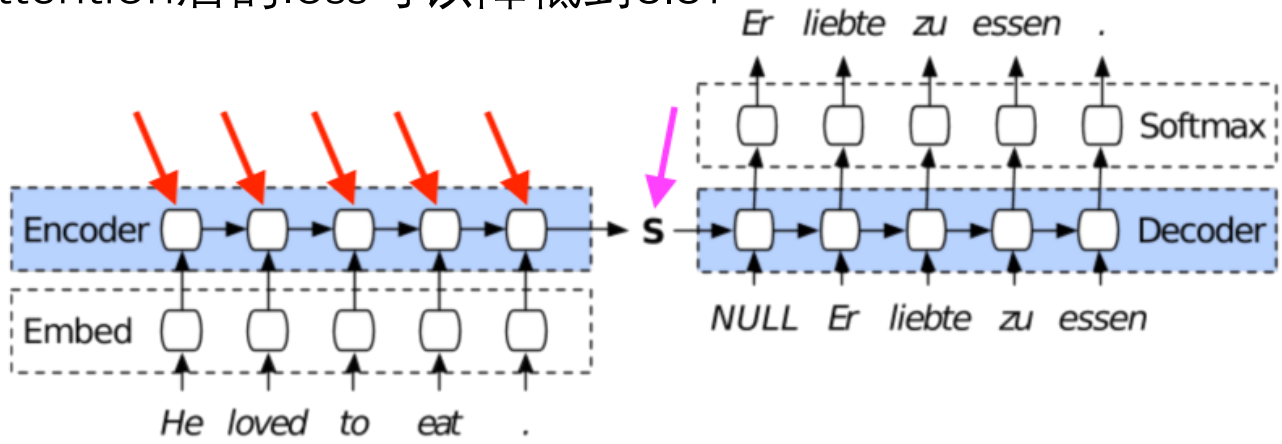
技巧2：teacher forcing

- 训练开始时网络几乎没有预测能力，decoder中前一个预测值作为输入进行下一个词语的预测的过程几乎没有用处
- 用一个随着训练进行而变化的概率pr_force，来决定是否使用理想的预测结果来作为下一步预测的输入
- 可以设置pr_force在开始时很大，即以很大概率用一个理想的输出来作为下一步的输入；而随着训练的进行，pr_force逐步减小，直至降为0
- 使用teacher forcing可以将loss减小到3.49



技巧3：Attentional model

- RNN不仅仅在最后有一个隐藏状态，它前面每输入一个单词也会对应一个隐藏状态
- 利用encoder中多个隐藏状态的加权平均来预测输出，这里的权重某种意义上可以表示句子的某些部分对于翻译来说是最重要的
- 方法：利用decoder的前一隐藏状态为输入，建立简单神经网络来进行权重的预测，利用softmax保证和为1且适当放大其中的某个权值
- 加入attention后的loss可以降低到3.37



技巧3：Attentional model

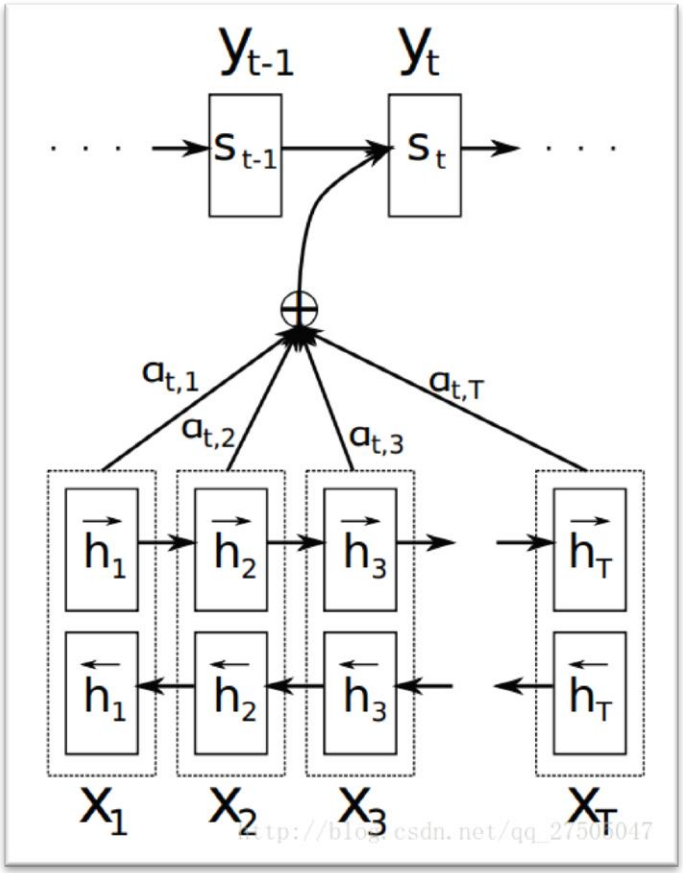
- 以中文to英文为例：

技巧3：Attentional model

- 无attention：输出由【前一个输出的结果】【前一隐层（可追述至encoder最后一个的隐藏状态）】共同决定
- 有atttention：输出由【前一个输出的结果】【前一隐层】【encoder各隐藏状态加权平均】共同决定

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$



目录 Contents

Paper：A Deep Visual-Semantic Embedding Model

- 工作：实现一个deep visual-semantic embedding model同时使用标注的图像数据和文本的语义信息来进行目标识别
- 贡献：使用文本数据学习不同label之间的语义关联，将图像映射到一个语义空间中，能够对未知的category预测label
- 方法：先分别预训练了两个图像和文本的模型，然后再映射到其自己的embedding model中进行统一度量
- 例子：

-图像分类中，将一只比格犬误识为喷气飞机或柯基犬，这是相同程度的错误

-但在单词向量的语义中，柯基和比格犬的词向量十分接近，但喷气飞机的词向量和这两种狗差别很大

将ImageNet分类映射到单词向量

- 数据来源：ImageNet的1000个类别和wordnet的82000个词语
- 由于ImageNet的类别是采用wordnet中词语命名，比较方便映射
- 遍历wordnet中词语，找到其在fast.text中的对应向量，共有49469个词语拥有对应的单词向量，同时将ImageNet中的类别名也与单词向量相对应
- 目标：训练一个输入为图像，输出为对应图像分类的单词向量的网络

```
tfms = tfms_from_model(arch, 224, transforms_side_on, max_zoom=1.1)
md = ImageClassifierData.from_names_and_array(PATH, images, img_vecs, val_idxs,
        classes=None, tfms=tfms, continuous=True, bs=256)
```

- 这里的continuous=True表明不要将输出处理为one-hot编码，而是将其作为连续的值进行回归，即词向量的方向

建立网络结构

- f：网络模型，这里使用resnet50
- C：所需要的输出，这里的数目是fast.text单词向量的大小，即300
- Is_multi=False：不是多分类也不是分类问题
- is_reg=True：这里是回归问题

```
models = ConvnetBuilder(arch, md.c, is_multi=False, is_reg=True, xtra_fc=[1024], ps=[0.2,0.2])

learn = ConvLearner(md, models, precompute=True)
learn.opt_fn = partial(optim.Adam, betas=(0.9,0.99))
```

- 注意：这里不需要softmax等函数

Loss function

- 这里为了判断多维空间向量（300维）的近似程度，loss function采用的是余弦相似度

```
def cos_loss(inp,targ): return 1 - F.cosine_similarity(inp,targ).mean()
learn.crit = cos_loss
```

```
learn.lr_find(start_lr=1e-4, end_lr=1e15)
```

```
learn.sched.plot()
```

设向量 A = (A1,A2,...,An)，B = (B1,B2,...,Bn) 。推广到多维:

$$\cos\theta = \frac{\sum_1^n (A_i \times B_i)}{\sqrt{\sum_1^n A_i^2} \times \sqrt{\sum_1^n B_i^2}}$$

- 进一步计算拟合以训练好完整的网络

测试

- 使用nmslib来对单词向量查找其最近邻，在ImageNet和wordNet上测试结果一致
- Text-image：取ImageNet中不存在的分类“boat”的单词向量，搜索其在ImageNet对应词向量中的最近邻，获取到船的图片；取“boat”和“engine”两个词向量的均值，搜索最近邻也能获取相应的图片
- Image-image：从一张已有图片获取其词向量，进行最近邻搜索，获取到的图片与原有图片十分近似



谢谢！