Chatbot Based on Deep Learning

——机器学习Proj2

小组：张晗翀 刘勉之 刘书畅

上海交通大学 软件工程 机器学习

目录

01 Background

02 Related Works

03 Approach

04 Implementation Details

05 Evaluation

06 Demo

07 Task Allocation

01

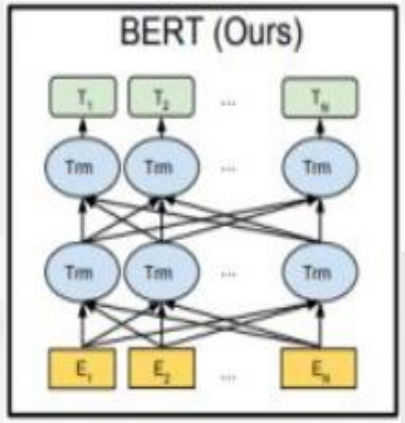
Background

项目背景

01 background

前沿工作

BERT： 全称 Bidirectional Encoder Representation from Transformers



即双向Transformer的Encoder

创新点 → pre-train方法

用了Masked LM和Next Sentence Prediction两种方法分别捕捉词语和句子级别的representation

同时读一整个单词序列
可以根据一个单词所有的前后文学习它的内容

优势：通过预训练和精调达到了惊人的效果
使用的是Transformer，也就是相对rnn更加高效、能捕捉更长距离的依赖
捕捉到的是真正意义上的bidirectional context信息

01 background

前沿工作

《Cross-Thought for Sentence Encoder Pre-training》:

BERT式模型缺点: 用无监督语料训练出的BERT做句子表示并不理想

- 如果只取CLS，这个表示是针对NSP进行优化的，表示的信息有限
- 如果取平均或最大池化，可能会把无用信息计算进来，增加噪声

改进目的:设计一个下游任务，直接优化得到的句子embedding

方法: 用周边其他句子的表示预测当前句子的token

具体的做法是: 先利用Transformer抽取句子表示，再对句子表示进行attention，选取相关的句子预测当前token

01 background

前沿工作

pQRNN: Google2019年的模型PRADO的一个改进版本

《PRADO: Projection Attention Networks for Document Classification On-Device》

作者思想： 对于文本分类这样的简单任务，很多词汇是和任务无关的，比如 a，the等

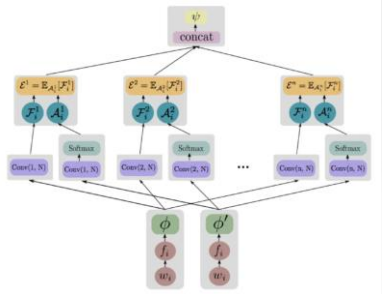
另外，也不需要embedding可以准确地表示每个词，只需要大概表示出词所属的类目就可以了。

改进： 在embedding阶段,PRADO对向量进行了如下压缩后，作者把B的维度限制在[128, 512]，d的维度限制在[32, 96]，从词表参数量上比BERT小了三四个数量级：

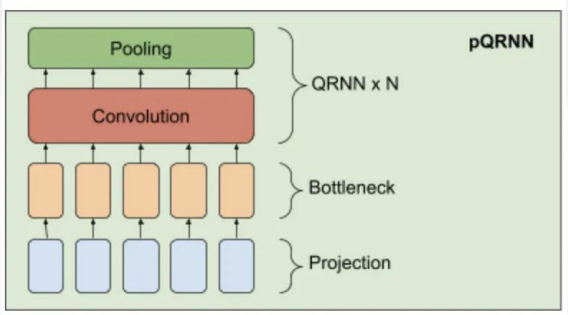
1. 对token进行哈希，得到2B bits的哈希值，比如 011001010
2. 用一个投影函数P，将每2个连续的bit映射到{-1,0,1}中
3. 得到B维的三元向量，比如[-1,-1,1,0,1]
4. 将 1xB 的三元向量和Bxd 的矩阵相乘，得到 d维embedding表示

优势： pQRNN在效果接近BERT的情况下，参数量缩小了约300倍，更轻量化

pQRNN在PRADO的基础上把encoder换成了QRNN（quasi-RNN），达到了接近BERT的效果



PRADO模型结构



pQRNN模型结构

01 background

Chatbot 个人理解

聊天机器人基于seq2seq等机器学习领域的技术，成为了当下人工智能方面的热门应用

虽然聊天机器人的基本功能并不复杂，即能够根据人们说出的自然语言，返回相应的语句作为回应，但是拥有如此功能的聊天机器人已被应用于诸如苹果的siri、小米的小爱同学等众多app

未来，聊天机器人还会得到更为广泛的发展，不仅自然语言处理方面会得到改善，甚至或许可以从人们的语言中了解人们的身份背景、情感等。

02

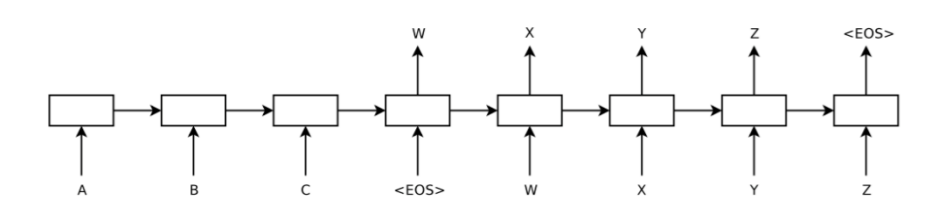
Related Works

相关工作

02 Related Works

2014

《Sequence to Sequence Learning with Neural Networks》



使用多层（原文4层）的LSTM将input sequence（不定长度的输入）映射到一个固定维度的向量。然后使用另外1个LSTM将这个向量解码输出

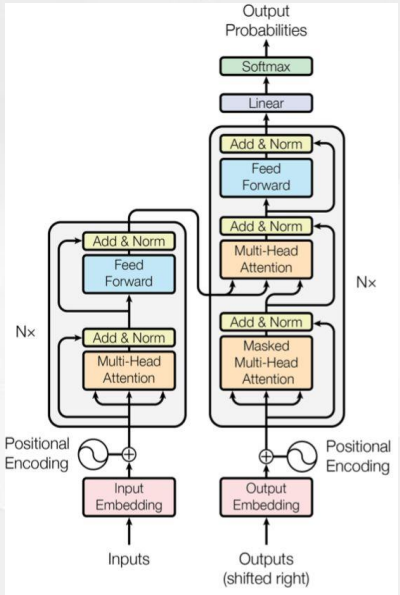
《Recurrent Models of Visual Attention》

这篇论文采用了RNN模型，并加入了Attention机制来进行图像的分类。

2017

《Attention is All You Need》

完全抛弃了RNN和CNN等网络结构，而仅仅采用Attention机制来进行机器翻译任务，并且取得了很好的效果



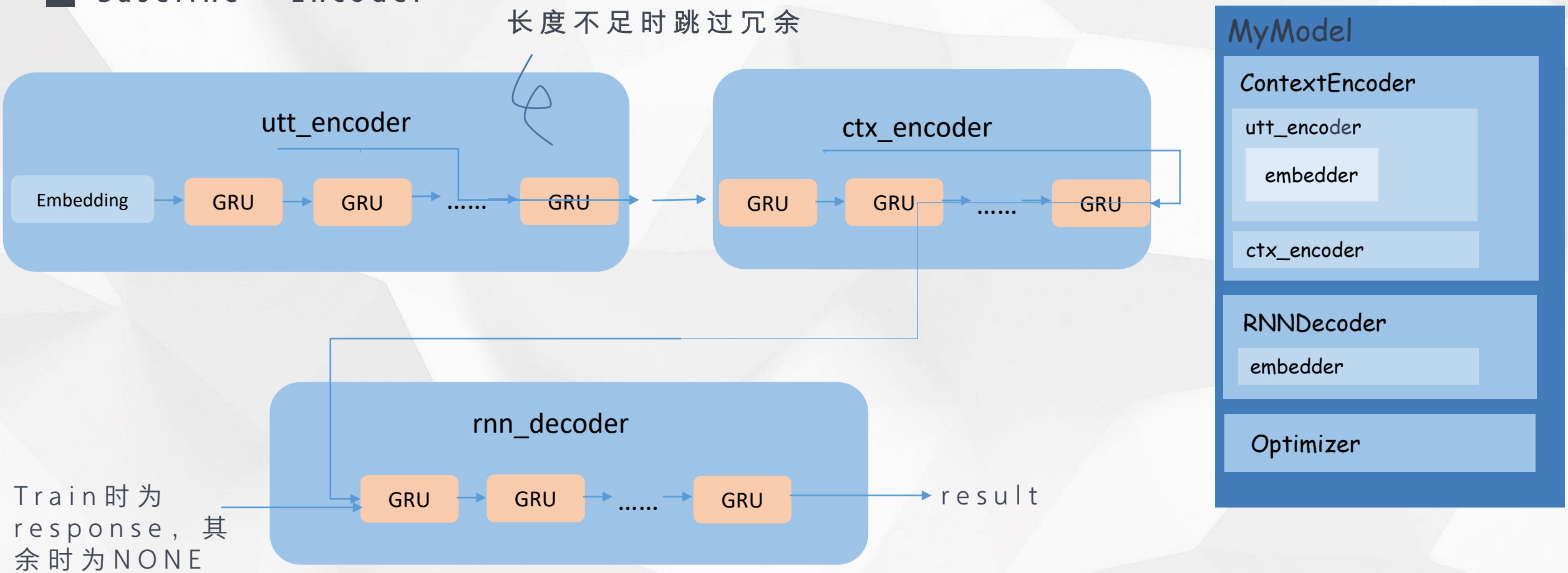
03

Approach

Describe our approach using diagrams and descriptions

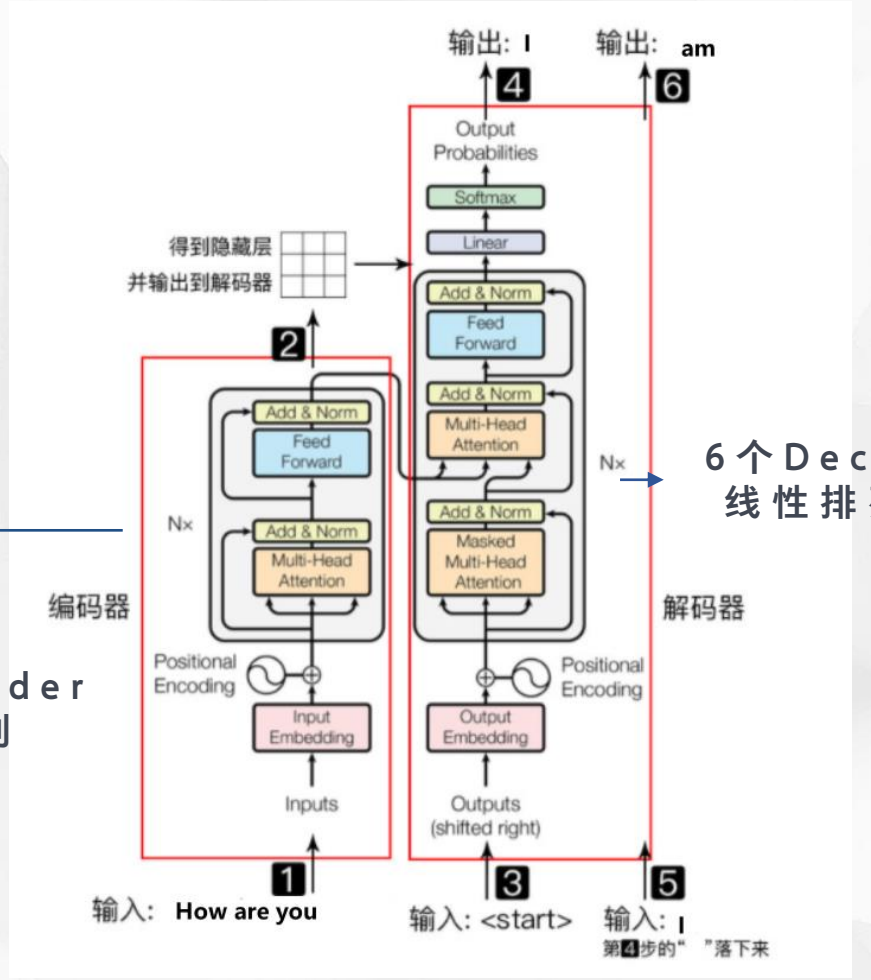
03 Approach

Baseline - Encoder



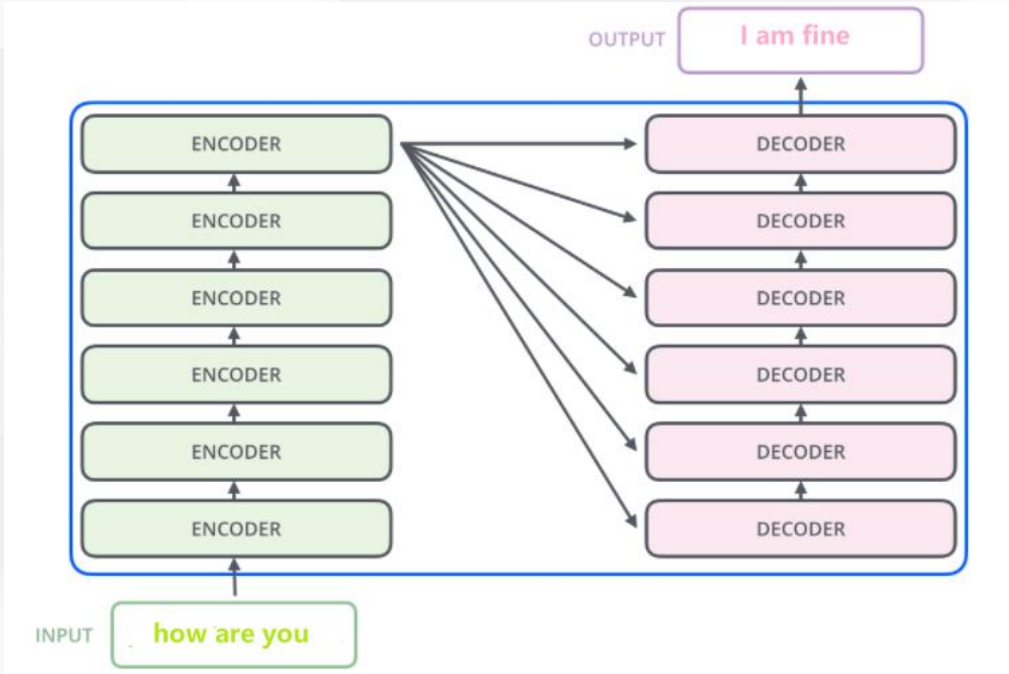
03 Approach

Transformer – 整体结构



6个Decoder
线性排列

6个Encoder
线性排列



模型组成：

Encoder 和 Decoder。Encoder 负责把输入（语言序列）映射成隐藏层然后解码器再把隐藏层映射为自然语言序列。

Transformer 和 RNN 的最大区别：

RNN 的训练是迭代的、串行的，必须要等当前字处理完，才可以处理下一个字。而 Transformer 的训练时并行的，即所有字是同时训练的，这样就大大增加了计算效率。

04

Implementation Details

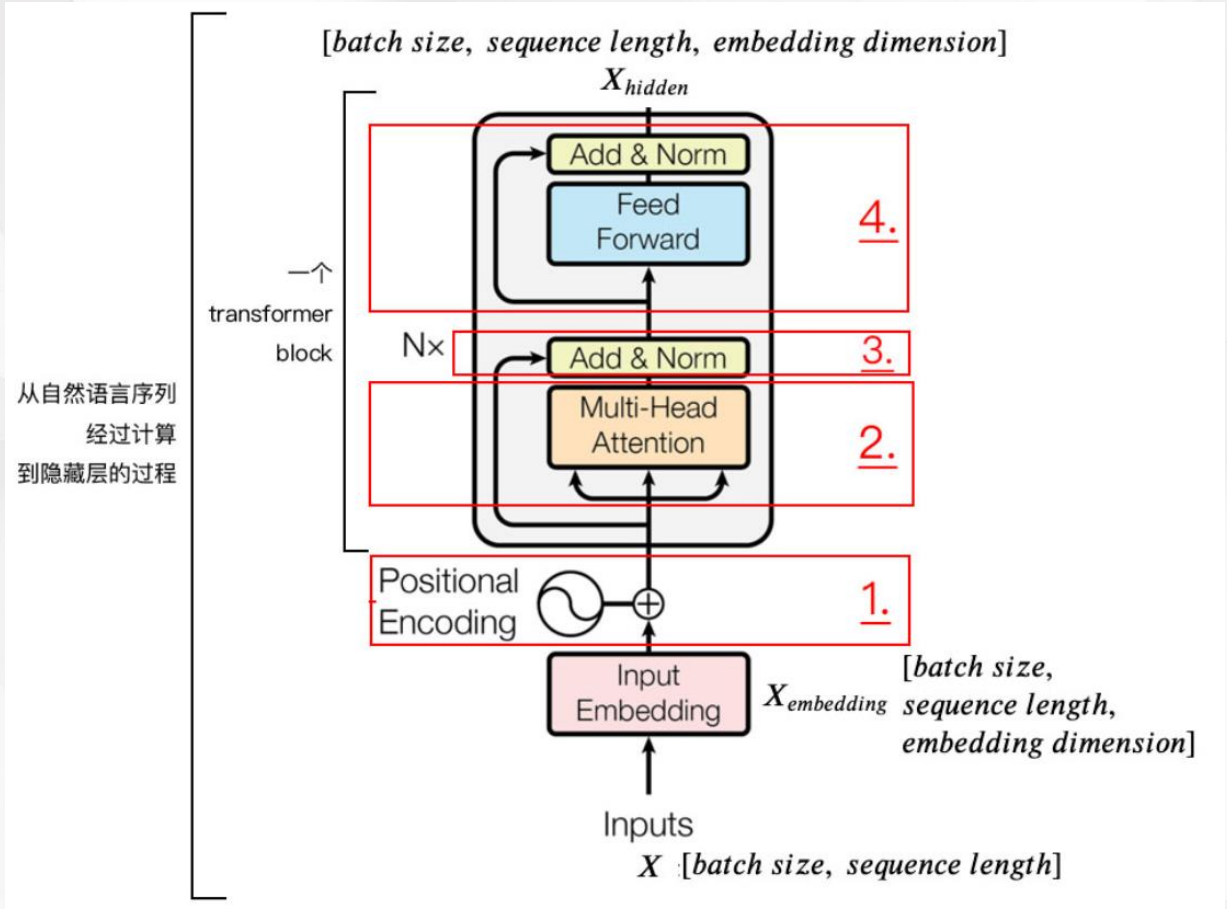
More details of key components and algorithms (e.g., encoder, decoder, etc) in your implementation.

04 Implementation Details

Transformer – Implementation Details

04 Implementation Details

Transformer – Encode:自然语言序列映射为隐藏层的数学表达



1. Positional Encoding

- 由于 Transformer 模型**没有**循环神经网络的迭代操作，所以我们必须提供每个字的位置信息给 Transformer，这样它才能识别出语言中的顺序关系
- 位置嵌入的维度与词向量的维度是相同的，都是 embedding_dimension
  使用 sin 和 cos 函数的线性变换来提供给模型位置信息:

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$
$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

- 对于输入的句子 X，通过 WordEmbedding 得到该句子中每个字的字向量，同时通过 Positional Encoding 得到所有字的位置向量，将其相加（维度相同，可以直接相加），得到该字真正的向量表示

04 Implementation Details

Transformer - Encode:自然语言序列映射为隐藏层的数学表达

```python
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        '''
        x: [seq_len, batch_size, d_model]
        '''
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)
```

1. Positional Encoding

- 由于 Transformer 模型**没有**循环神经网络的迭代操作，所以我们必须提供每个字的位置信息给 Transformer，这样它才能识别出语言中的顺序关系
- 位置嵌入的维度与词向量的维度是相同的，都是 embedding_dimension
  使用 sin 和 cos 函数的线性变换来提供给模型位置信息:

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$
$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

- 对于输入的句子 X，通过 WordEmbedding 得到该句子中每个字的字向量，同时通过 Positional Encoding 得到所有字的位置向量，将其相加（维度相同，可以直接相加），得到该字真正的向量表示。第 t 个字的向量记作 xt

04 Implementation Details

Transformer – Encode:自然语言序列映射为隐藏层的数学表达

里面的参数是列表
列表里面存了 n_layers 个
Encoder Layer

```python
class Encoder(nn.Module):
    def __init__(self,config,vocab_size):
        super(Encoder, self).__init__()
        self.src_emb = nn.Embedding(vocab_size, config['d_model'])
        self.pos_emb = PositionalEncoding(config['d_model'])
        # 使用 nn.ModuleList() 里面的参数是列表，列表里面存了 n_layers 个 Encoder Layer
        self.layers = nn.ModuleList([EncoderLayer(config) for _ in range(config['n_layers'])])

    def forward(self, enc_inputs):
        '''
        enc_inputs: [batch_size, src_len]
        '''
        enc_outputs = self.src_emb(enc_inputs) # [batch_size, src_len, d_model]
        enc_outputs = self.pos_emb(enc_outputs.transpose(0, 1)).transpose(0, 1) # [batch_size, src_len, d_model]
        enc_self_attn_mask = get_attn_pad_mask(enc_inputs, enc_inputs) # [batch_size, src_len, src_len]
        enc_self_attns = []
        for layer in self.layers:
            # enc_outputs: [batch_size, src_len, d_model], enc_self_attn: [batch_size, n_heads, src_len, src_len]
            enc_outputs, enc_self_attn = layer(enc_outputs, enc_self_attn_mask)
            enc_self_attns.append(enc_self_attn)
        return enc_outputs, enc_self_attns
```

1. Positional Encoding

- 由于 Transformer 模型**没有**循环神经网络的迭代操作，所以我们必须提供每个字的位置信息给 Transformer，这样它才能识别出语言中的顺序关系
- 位置嵌入的维度与词向量的维度是相同的，都是 embedding_dimension
  使用 sin 和 cos 函数的线性变换来提供给模型位置信息:

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$
$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

- 对于输入的句子 X，通过 WordEmbedding 得到该句子中每个字的字向量，同时通过 Positional Encoding 得到所有字的位置向量，将其相加（维度相同，可以直接相加），得到该字真正的向量表示。第 t 个字的向量记作 xt

04 Implementation Details

Transformer - Encode:自然语言序列映射为隐藏层的数学表达

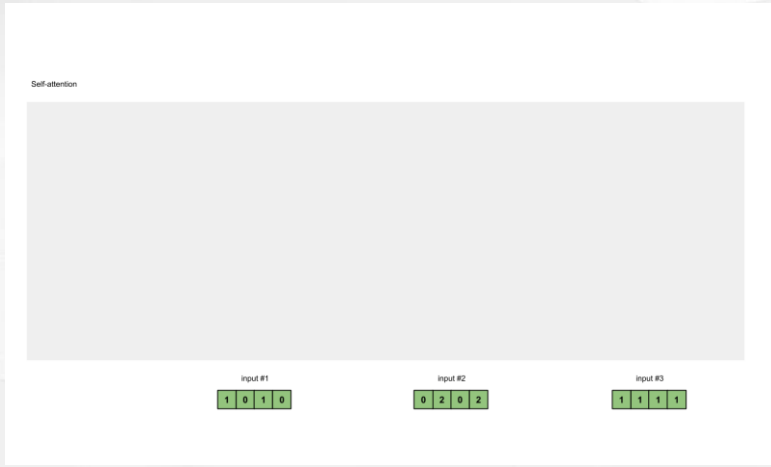
```python
class ScaledDotProductAttention(nn.Module):
    def __init__(self):
        super(ScaledDotProductAttention, self).__init__()

    def forward(self, Q, K, V, attn_mask):
        '''
        Q: [batch_size, n_heads, len_q, d_k]
        K: [batch_size, n_heads, len_k, d_k]
        V: [batch_size, n_heads, len_v(=len_k), d_v]
        attn_mask: [batch_size, n_heads, seq_len, seq_len]
        '''
        '''
        相乘之后得到的 scores 还不能立刻进行 softmax，需要和 attn_mask 相加，把一些需要屏蔽的信息屏蔽掉
        '''
        scores = torch.matmul(Q, K.transpose(-1, -2)) / np.sqrt(64) # scores : [batch_size, n_heads, len_q, len_k]
        scores.masked_fill_(attn_mask, -1e9) # Fills elements of self tensor with value where mask is True.

        attn = nn.Softmax(dim=-1)(scores)
        context = torch.matmul(attn, V) # [batch_size, n_heads, len_q, d_v]
        return context, attn
```

2. Self Attention Mechanism

这里要做的是，通过 Q 和 K 计算出 scores，然后将 scores 和 V 相乘，得到每个单词的 context vector

相乘之后得到的 scores 还不能立刻进行 softmax，需要和 attn_mask 相加，把一些需要屏蔽的信息屏蔽掉，attn_mask 是一个仅由 True 和 False 组成的 tensor，并且一定会保证 attn_mask 和 scores 的维度四个值相同（不然无法做对应位置相加）

mask 完了之后，就可以对 scores 进行 softmax 了
使得它们的和为 1
最后再与 V 相乘，得到 context



04 Implementation Details

Transformer – Encode:自然语言序列映射为隐藏层的数学表达

```python
class MultiHeadAttention(nn.Module):
    def __init__(self,config):
        super(MultiHeadAttention, self).__init__()
        self.W_Q = nn.Linear(config['d_model'], config['d_k'] *config['n_heads'], bias=False)
        self.W_K = nn.Linear(config['d_model'], config['d_k'] * config['n_heads'], bias=False)
        self.W_V = nn.Linear(config['d_model'], config['d_v'] * config['n_heads'], bias=False)
        self.fc = nn.Linear(config['n_heads'] * config['d_v'], config['d_model'], bias=False)
    def forward(self,config, input_Q, input_K, input_V, attn_mask):
        '''
        input_Q: [batch_size, len_q, d_model]
        input_K: [batch_size, len_k, d_model]
        input_V: [batch_size, len_v(=len_k), d_model]
        attn_mask: [batch_size, seq_len, seq_len]
        '''
        residual, batch_size = input_Q, input_Q.size(0)
        # (B, S, D) -proj-> (B, S, D_new) -split-> (B, S, H, W) -trans-> (B, H, S, W)
        Q = self.W_Q(input_Q).view(batch_size, -1, config['n_heads'], config['d_k']).transpose(1,2)  # Q: [batch_size, n_heads, len_q, d_k]
        K = self.W_K(input_K).view(batch_size, -1, config['n_heads'], config['d_k']).transpose(1,2)  # K: [batch_size, n_heads, len_k, d_k]
        V = self.W_V(input_V).view(batch_size, -1, config['n_heads'], config['d_v']).transpose(1,2)  # V: [batch_size, n_heads, len_v(=len_k), d_v]

        attn_mask = attn_mask.unsqueeze(1).repeat(1, config['n_heads'], 1, 1) # attn_mask : [batch_size, n_heads, seq_len, seq_len]

        # context: [batch_size, n_heads, len_q, d_v], attn: [batch_size, n_heads, len_q, len_k]
        context, attn = ScaledDotProductAttention()(Q, K, V, attn_mask)
        context = context.transpose(1, 2).reshape(batch_size, -1, config['n_heads'] *config[' d_v']) # context: [batch_size, len_q, n_heads * d_v]
        output = self.fc(context) # [batch_size, len_q, d_model]
        return nn.LayerNorm(config['d_model']).cuda()(output + residual), attn
```

2. Self Attention Mechanism

- Transformer 为什么需要进行 Multi-head Attention

原论文中说到进行 Multi-head Attention 的原因是将模型分为多个头，形成多个子空间，可以让模型去关注不同方面的信息，最后再将各个方面的信息综合起来。

其实直观上也可以想到，如果自己设计这样的一个模型，必然也不会只做一次 attention，多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用多个卷积核的作用

直观上讲，多头的注意力有助于网络捕捉到更丰富的特征 / 信息

04 Implementation Details

Transformer - Encode:自然语言序列映射为隐藏层的数学表达

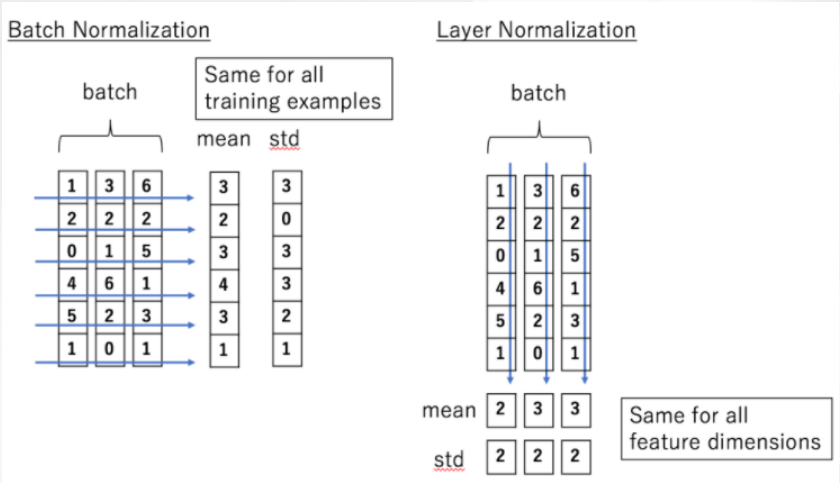
```python
class PoswiseFeedForwardNet(nn.Module):
    def __init__(self,config):
        super(PoswiseFeedForwardNet, self).__init__()
        '''
            做两次线性变换，残差连接后再跟一个 Layer Norm
        '''
        self.fc = nn.Sequential(

            nn.Linear(config['d_model'], config['d_ff'], bias=False),
            nn.ReLU(),
            nn.Linear(config['d_ff'], config['d_model'], bias=False)
        )
    def forward(self, config,inputs):
        '''
        inputs: [batch_size, seq_len, d_model]
        '''
        residual = inputs
        output = self.fc(inputs)
        return nn.LayerNorm(config['d_model']).cuda()(output + residual) # [batch_size, seq_len, d_model]
```

3.残差连接和 Layer Normalization

- 我们在上一步得到了经过 self-attention 加权之后输出，也就是 Attention(Q, K, V)，然后把他们加起来做残差连接)

$$X_{embedding} + Self\ Attention(Q,\ K,\ V)$$

- Layer Normalization 的作用是把神经网络中隐藏层归一为标准正态分布，也就是 i.i.d 独立同分布，以起到加快训练速度、加速收敛的作用



04 Implementation Details

Transformer - Decode

```python
class DecoderLayer(nn.Module):
    def __init__(self,config):
        super(DecoderLayer, self).__init__()
        self.dec_self_attn = MultiHeadAttention(config)
        self.dec_enc_attn = MultiHeadAttention(config)
        self.pos_ffn = PoswiseFeedForwardNet(config)

    def forward(self, dec_inputs, enc_outputs, dec_self_attn_mask, dec_enc_attn_mask):
        '''
        dec_inputs: [batch_size, tgt_len, d_model]
        enc_outputs: [batch_size, src_len, d_model]
        dec_self_attn_mask: [batch_size, tgt_len, tgt_len]
        dec_enc_attn_mask: [batch_size, tgt_len, src_len]
        '''
        # dec_outputs: [batch_size, tgt_len, d_model], dec_self_attn: [batch_size, n_heads, tgt_len, tgt_len]
        dec_outputs, dec_self_attn = self.dec_self_attn(dec_inputs, dec_inputs, dec_inputs, dec_self_attn_mask)
        # dec_outputs: [batch_size, tgt_len, d_model], dec_enc_attn: [batch_size, h_heads, tgt_len, src_len]
        dec_outputs, dec_enc_attn = self.dec_enc_attn(dec_outputs, enc_outputs, enc_outputs, dec_enc_attn_mask)
        dec_outputs = self.pos_ffn(dec_outputs) # [batch_size, tgt_len, d_model]
        return dec_outputs, dec_self_attn, dec_enc_attn
```

Decoder Layer

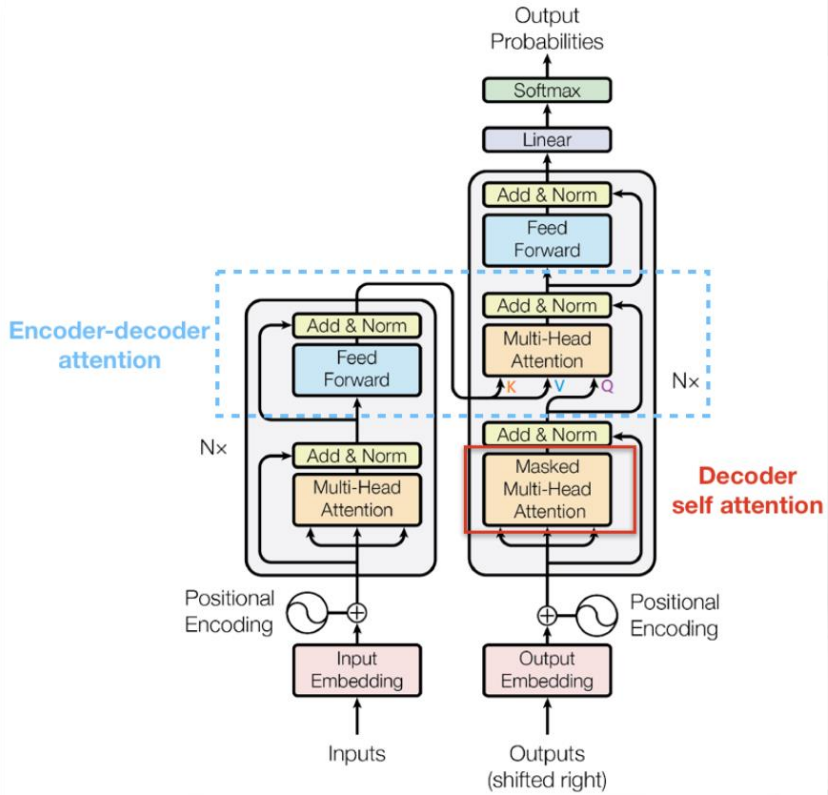
在 Decoder Layer 中会调用两次 MultiHeadAttention

第一次是计算 Decoder Input 的 self-attention，得到输出 dec_outputs
然后将 dec_outputs 作为生成 Q 的元素，enc_outputs 作为生成 K 和 V 的元素，再调用一次 MultiHeadAttention，得到的是 Encoder 和 Decoder Layer 之间的 context vector

最后将 dec_outputs 做一次维度变换

04 Implementation Details

Transformer - Decode



Masked Self-Attention

传统 Seq2Seq 中 Decoder 使用的是 RNN 模型，因此在训练过程中输入 t 时刻的词，模型无论如何也看不到未来时刻的词，因为循环神经网络是时间驱动的，只有当 t 时刻运算结束了，才能看到 t+1 时刻的词。

而 Transformer Decoder 抛弃了 RNN，改为 Self-Attention，由此就产生了一个问题，在训练过程中，整个 ground truth 都暴露在 Decoder 中，这显然是不对的，我们需要对 Decoder 的输入进行一些处理，该处理被称为 Mask

举个例子，当我们输入 "I" 时，模型目前仅知道包括 "I" 在内之前所有字的信息，即 "<start>" 和 "I" 的信息，不应该让其知道 "I" 之后词的信息。道理很简单，我们做预测的时候是按照顺序一个字一个字的预测，怎么能这个字都没预测完，就已经知道后面字的信息了呢？Mask 非常简单，首先生成一个下三角全 0，上三角全为负无穷的矩阵，然后将其与 Scaled Scores 相加即可

Mask ：首先生成一个下三角全 0，上三角全为负无穷的矩阵，然后将其与 Scaled Scores 相加即可

```
dec_self_attn_pad_mask = get_attn_pad_mask(dec_inputs, dec_inputs).cuda() # [batch_size, tgt_len, tgt_len]
dec_self_attn_subsequence_mask = get_attn_subsequence_mask(dec_inputs).cuda() # [batch_size, tgt_len, tgt_len]
dec_self_attn_mask = torch.gt((dec_self_attn_pad_mask + dec_self_attn_subsequence_mask), 0).cuda() # [batch_size, tgt_len, tgt_len]
```

04 Implementation Details

Transformer – Demo Details

04 Implementation Details

```
device = next(model.parameters()).device
context_it = np.zeros(shape=[1, conf['diaglen'], conf['maxlen'] - 1], dtype=np.int64)
utt_lens_it = np.zeros(shape=[1, conf['diaglen']], dtype=np.int64)
for i in range(conf['diaglen']):
    context_it[0][i][0] = EOS_ID
    utt_lens_it[0][i] = 1
i = 0
while 1:
    if i >= conf['diaglen']:
        for j in range(conf['diaglen'] - 1):
            context_it[0][j] = context_it[0][j + 1]
            utt_lens_it[0][j] = utt_lens_it[0][j + 1]
        i = conf['diaglen'] - 1
    text = input()
    indexes = sent2indexes(text, vocab, conf['maxlen'] - 1)
    indexes[0][indexes[1][0]] = EOS_ID
    context_it[0][i] = indexes[0]
    utt_lens_it[0][i] = indexes[1][0] + 1
    context = torch.tensor(context_it)
    context_lens = torch.tensor([i + 1])
    utt_lens = torch.tensor(utt_lens_it)
    context, context_lens, utt_lens = [tensor.to(device).long() for tensor in [context, context_lens, utt_lens]]
    utt_lens[utt_lens <= 0] = 1
    with torch.no_grad():
        sample_words, sample_lens = model.sample(context, context_lens, utt_lens, repeat)
    pred_sents, _ = indexes2sent(sample_words, vocab)
    print(pred_sents[0])
    i += 1
```

初始化context窗口迭代器和长度迭代器

语句数量达到窗口大小时，移除最早语句

输入文本转化为indexes

模型进行预测

预测indexes转化为文本

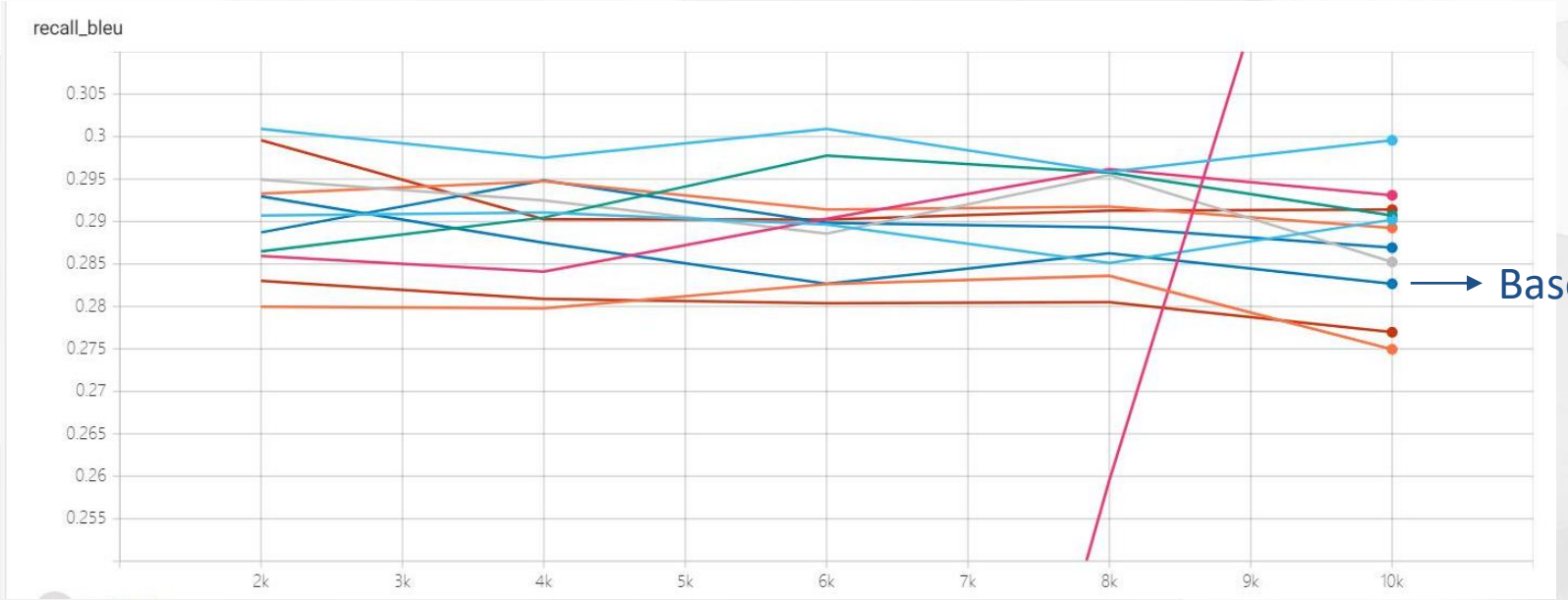
05

Evaluation

Show the curves, comparison , descriptions

05 Implementation Details

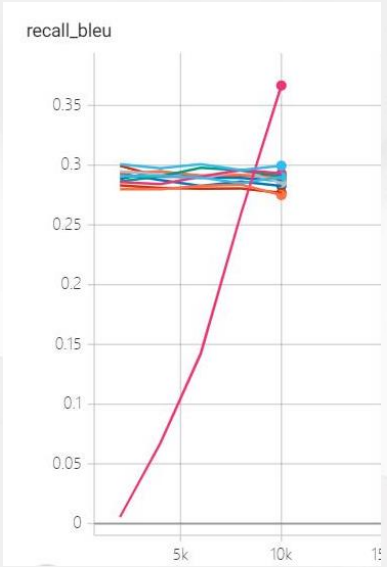
Name	Smoothed	Value	Step	Time	Relative
1	0.275	0.275	10k	Sun Dec 20, 15:26:05	11m 7s
10	0.2996	0.2996	10k	Sat Dec 26, 21:49:06	39m 23s
11	0.3667	0.3667	10k	Sun Dec 27, 01:34:42	38m 56s
2	0.277	0.277	10k	Sun Dec 20, 20:34:14	10m 58s
3	0.2902	0.2902	10k	Sun Dec 20, 22:49:28	17m 55s
4	0.2931	0.2931	10k	Mon Dec 21, 00:24:43	29m 41s
5	0.2907	0.2907	10k	Mon Dec 21, 11:12:15	11m 30s
6	0.2852	0.2852	10k	Mon Dec 21, 14:34:02	11m 51s
7	0.2892	0.2892	10k	Mon Dec 21, 15:13:23	18m 55s
8	0.2869	0.2869	10k	Mon Dec 21, 18:59:34	9m 43s
9	0.2914	0.2914	10k	Mon Dec 21, 20:07:04	17m 12s
baseline	0.2827	0.2827	10k	Sun Dec 20, 16:31:53	11m 9s



综合考虑，调整 11 以下参数为：

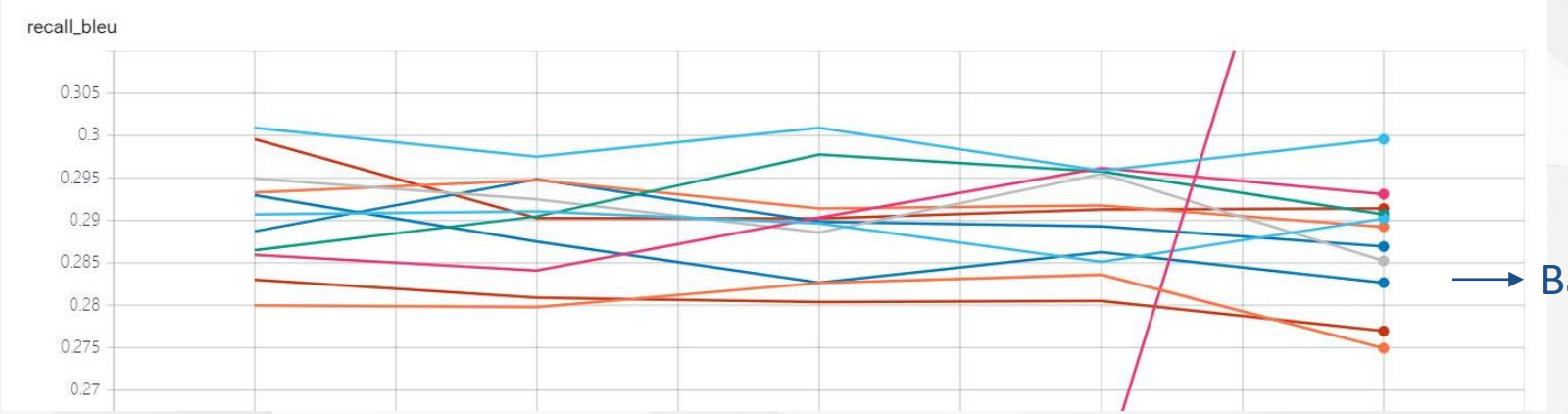
"diaglen": 20
"emb_size": 512
"rnn_hid_utt": 1024
"rnn_hid_ctx": 1024
"rnn_hid_dec": 1024
"n_layers": 3
"dropout": 0.7
"lr": 2e-07

经过一定数量的样本学习后，效果突出

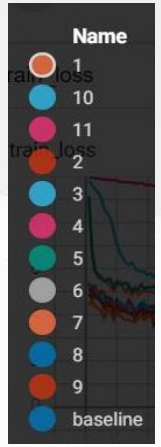


标号	参数	备注
baseline		
2	dropout=0	不考虑过拟合，效果更差
3	n_layers=3	增加深度，效果变好
5	lr=2e-05	学习率降低，效果变好
6	emb_size=300	增加词向量长度，效果变好
9	rnn_hid_*=1024	效果变好

05 Implementation Details



→ Baseline



	maxlen	diaglen	emb_size	rnn_hid_*	n_layers	dropout	teach_force	batch_size	epochs	lr	beta1	init_w	clip
1	40	10	200	512	1	0.5	0.8	64	10	0.002	0.9	0.05	5.0
2	40	10	200	512	1	0	0.8	64	10	0.0002	0.9	0.05	5.0
3	40	10	200	512	3	0.5	0.8	64	10	0.0002	0.9	0.05	5.0
4	40	10	200	512	6	0.5	0.8	64	10	0.0002	0.9	0.05	5.0
5	40	10	200	512	1	0.5	0.8	64	10	2e-05	0.9	0.05	5.0
6	40	10	300	512	1	0.5	0.8	64	10	0.0002	0.9	0.05	5.0
7	40	10	200	512	3	0.5	0.8	64	10	2e-05	0.9	0.05	5.0
8	40	1	200	512	1	0.5	0.8	64	10	0.0002	0.9	0.05	5.0
9	40	10	200	1024	1	0.5	0.8	64	10	0.0002	0.9	0.05	5.0
10	40	20	512	1024	3	0.7	0.8	64	10	2e-06	0.9	0.05	5.0
11	40	20	512	1024	3	0.7	0.8	64	10	2e-07	0.9	0.05	5.0

06

Demo

Show some concrete examples (selected context and the generated response) yielded by both yours and the baseline model.

06 Demo

```
Me:       hello .
Chatbot: which makes high in christmas evening ? how did you use ? </s>
Me:       i have to do my homework
Chatbot: exactly what ? </s>
Me:       it is you .
```

我们选取了某次训练效果较好的模型，截取了一段与Chatbot的对话。可以看到Chatbot在经过训练后，可以根据用户的话语，做出语法通顺、逻辑合理的回应。

07

Task Allocation

Show the contribution of each member.

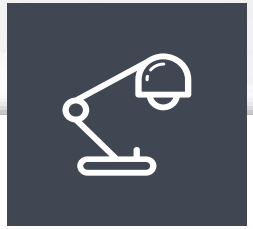
07 Task Allocation

张晗翀

- 理解参数含义
- 调参
- 实现Demo
- PPT演讲

刘勉之

- 理解baseline
- 为baseline加注释
- PPT资料查询



刘书畅

- 理解transformer
- 搭建transformer
- PPT制作

欢迎老师批评指正

——机器学习

小组：张晗翀 刘勉之 刘书畅

上海交通大学 软件工程 机器学习