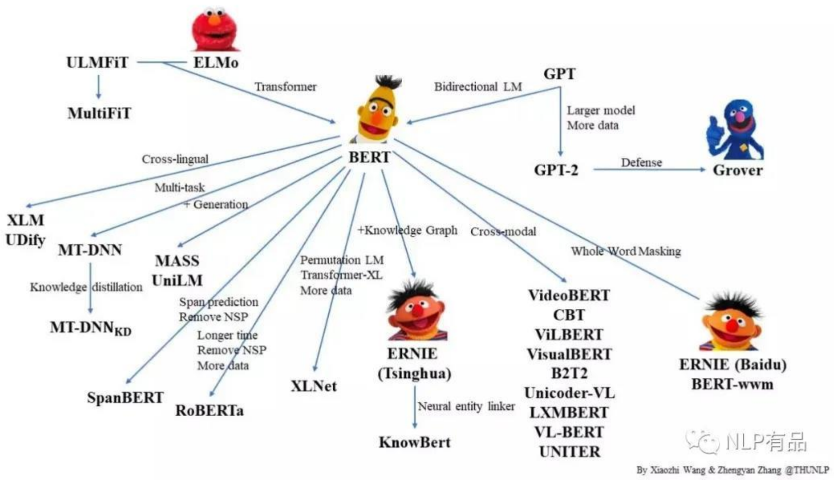
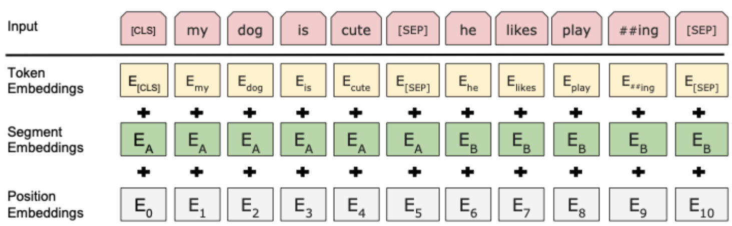
Table of Contents

BERT原理及实现



1 输入序列与嵌入表示



输入序列

$$input = ([CLS], s_1, s_2, \cdots, s_m, [SEP], p_1, p_2, \cdots, p_n, [SEP])$$

其中，$s_i, p_j \in \mathbb{N}$为输入符号表中的序号；子序列$(s_1, \cdots, s_m)$为句子对中前序句子；子序列$(p_1, \cdots, p_n)$为句子对中后续句子；输入序列首标记$[CLS]$用作分类任务表示；特殊标记$[SEP]$用作区分句子对各子句。

符号嵌入$Embedding_{tok}(input) \in \mathbb{R}^{N \times d_{model}}$，其中，$N$为输入符号个数，$d_{model}$为符号嵌入维度。

句子分割嵌入$Embedding_{seg}(input) \in \mathbb{R}^{N \times d_{model}}$，其中，$N$为输入符号个数，$d_{model}$为句子嵌入维度。对于单句输入，仅使用句子A嵌入。

符号位置嵌入$Embedding_{pos}(input) \in \mathbb{R}^{N \times d_{model}}$，其中，$N$为输入符号个数，$d_{model}$为符号位置嵌入维度。

2 单词级和句子级的多任务学习

2.1 遮蔽语言模型（MLM)训练任务

遮蔽语言模型可描述为给定单词上下文序列后，当前单词出现的条件概率的乘积：

$$P\left(w_1^T\right) = \prod_{t=1}^{T} P\left(w_t | w_1^{t-1}, w_{t+1}^T\right)$$

其中，w_t是第t个单词，$w_i^j = \left(w_i, w_{i+1}, \ldots, w_{j-1}, w_j\right)$是从第$i$个单词到第$j$个单词的子序列。

具体的，训练数据集中选择15%的遮蔽单词w_t，并以特殊标记$[MASK]$进行替换。为减少$[MASK]$标记对微调的影响，数据生成器将执行以下操作，而不是始终用$[MASK]$替换所选单词:

- 80%的时间：将单词替换为$[MASK]$；
- 10%的时间：用随机单词替换单词;
- 10%的时间：保持单词不变。这样做的目的是使表示偏向实际观察到的单词。

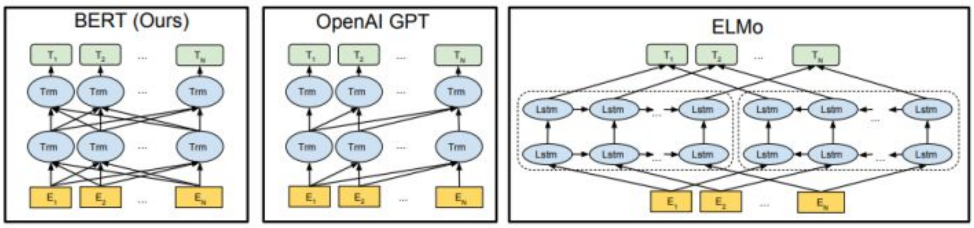
2.2 下一句预测任务

从语料库中生成二值化的下一句句子预测任务。

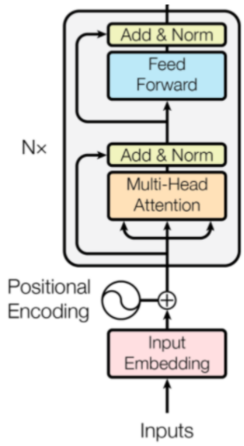
具体的，当为每个预训练选择句子A和B时，B的50%的时间是跟随A的实际下一个句子，而50%的时间是来自语料库的随机句子。

- input=[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP] label = IsNext
- input=[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP] label = NotNext

3 模型架构



3.1 编码器结构



编码器结构：

$$e_0 = Embedding_{tok}(inputs) + Embedding_{seg}(inputs) + Embedding_{pos}(inputs)$$
$$e_l = EncoderLayer(e_{l-1}), l \in [1, n]$$

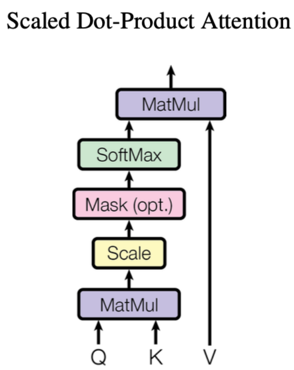
其中，$e_0 \in \mathbb{R}^{N \times d_{model}}$为编码器输入，$EncoderLayer(\cdot)$为编码器层，$n$为层数，$e_l \in \mathbb{R}^{N \times d_{model}}$为第$l$层编码器层输出。

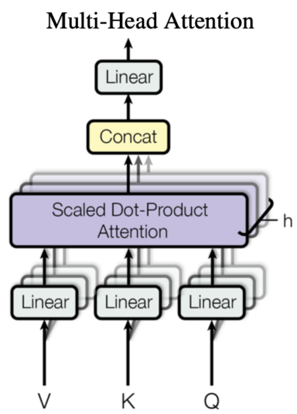
编码器层EncoderLayer：

$$e_{mid} = LayerNorm(e_{in} + MultiHeadAttention(e_{in}))$$
$$e_{out} = LayerNorm(e_{mid} + FFN(e_{mid}))$$

其中，$e_{in} \in \mathbb{R}^{N \times d_{model}}$为编码器层输入，$e_{out} \in \mathbb{R}^{N \times d_{model}}$为编码器层输出，$MultiHeadAttention(\cdot)$为多头注意力机制，$FFN(\cdot)$为前馈神经网络，$LayerNorm(\cdot)$为层归一化。

3.2 多头注意力机制与缩放点积





输入向量序列$e_{in} = (e_{in1}, e_{in2}, \cdots, e_{inN}) \in \mathbb{R}^{N \times d_{model}}$，分别得到查询向量序列$Q = e_{in}$，键向量序列$K = e_{in}$，值向量序列$V = e_{in}$。

多头注意力机制

$$MultiHeadAttention(e_{in}) = MultiHead(Q, K, V) = Concat(head_1, \cdots, head_h)W^O$$

其中，多头输出$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$，可学习的参数矩阵
$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$

使用缩放点积作为打分函数的自注意力机制

$$Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) = softmax\left(\frac{QW_i^Q\left(KW_i^K\right)^\top}{\sqrt{d_k}}\right)VW_i^V$$

3.3 前馈神经网络

$$FFN(e_{mid}) = GELU(e_{mid}W_1 + b_1)W_2 + b_2$$

其中，参数矩阵$W_1 \in \mathbb{R}^{dmodel \times dff}$, $W_2 \in \mathbb{R}^{dff \times dmodel}$; 偏置$b_1 \in \mathbb{R}^{dff}$, $b_2 \in \mathbb{R}^{dmodel}$。

3.4 pad掩码

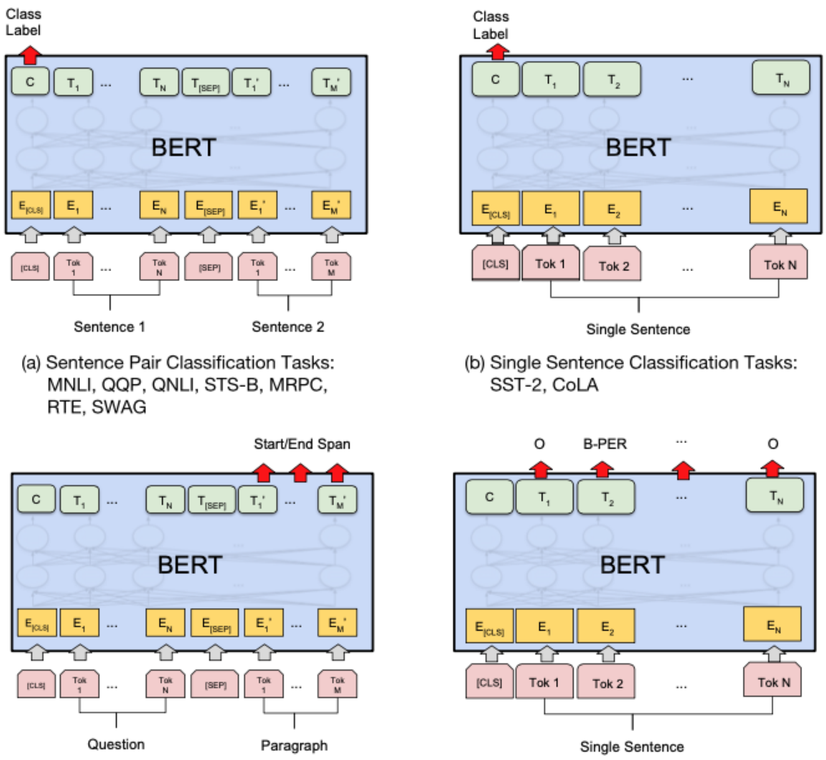
$$enc\_pad\_mask_j = (e_{j1}, e_{j2}, \cdots, e_{jp}, \cdots, e_{jN})$$

其中，

$$e_{jp} = \begin{cases} True, & i_p = 0 \\ False, & i_p \neq 0 \end{cases} \quad j = 1, 2, \cdots, N$$

$enc\_pad\_mask \in \mathbb{R}^{N \times N}$，$i_p$为输入序列$inputs$对应位置序号。

3.5 模型任务



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

4 PyTorch实现

4.1 导包及参数设置

In [2]:

```python
import math
import re
from random import *
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable

maxlen = 30
batch_size = 6
max_pred = 5
n_layers = 6
n_heads = 12
d_model = 768
d_ff = 768 * 4
d_k = d_v = 64
n_segments = 2
```

executed in 1.52s, finished 11:39:41 2020-06-28

4.2 数据处理

In [3]:

```python
text = (
    'Hello, how are you? I am Romeo.\n'
    'Hello, Romeo. My name is Juliet. Nice to meet you.\n'
    'Nice to meet you too. How are you today?\n'
    'Great. My baseball team won the competition.\n'
    'Oh Congratulations, Juliet\n'
    'Thanks you Romeo'
)

sentences = re.sub("[.,!?\\-]", '', text.lower()).split('\n')
word_list = list(set(" ".join(sentences).split()))
word_dict = {'[PAD]' : 0, '[CLS]' : 1, '[SEP]' : 2, '[MASK]' : 3}
for i, w in enumerate(word_list):
    word_dict[w] = i + 4
number_dict = {i : w for i, w in enumerate(word_dict)}
vocab_size = len(word_dict)

token_list = list()
for sentence in sentences:
    arr = [word_dict[s] for s in sentence.split()]
    token_list.append(arr)

def make_batch():
    batch = []
    positive = negative = 0
    while positive != batch_size / 2 or negative != batch_size /2:
        tokens_a_index, tokens_b_index = randrange(len(sentences)), randrange(len(s
        tokens_a, tokens_b = token_list[tokens_a_index], token_list[tokens_b_index]
        input_ids = [word_dict['[CLS]']] + tokens_a + [word_dict['[SEP]']] + tokens
        segment_ids = [0] * (1 + len(tokens_a) + 1) + [1] * (len(tokens_b) + 1)

        n_pred = min(max_pred, max(1, int(round(len(input_ids) * 0.15))))
        cand_maked_pos = [i for i, token in enumerate(input_ids)
                          if token != word_dict['[CLS]'] and token != word_dict['[S
        shuffle(cand_maked_pos)
        masked_tokens, masked_pos = [], []
        for pos in cand_maked_pos[: n_pred]:
            masked_pos.append(pos)
            masked_tokens.append(input_ids[pos])
            if random() < 0.8:
                input_ids[pos] = word_dict['[MASK]']
            elif random() < 0.5:
                index = randint(0, vocab_size - 1)
                input_ids[pos] = word_dict[number_dict[index]]

        n_pad = maxlen - len(input_ids)
        input_ids.extend([0] * n_pad)
        segment_ids.extend([0] * n_pad)

        if max_pred > n_pred:
            n_pad = max_pred - n_pred
            masked_tokens.extend([0] * n_pad)
            masked_pos.extend([0] * n_pad)

        if tokens_a_index + 1 == tokens_b_index and positive < batch_size / 2:
            batch.append([input_ids, segment_ids, masked_tokens, masked_pos, True])
            positive += 1
        elif tokens_a_index + 1 != tokens_b_index and negative < batch_size / 2:
            batch.append([input_ids, segment_ids, masked_tokens, masked_pos, False]
            negative += 1

    return batch
```

executed in 21ms, finished 11:40:49 2020-06-28

4.3 嵌入表示

In [9]:

```
class Embedding(nn.Module):
    def __init__(self):
        super(Embedding, self).__init__()
        self.tok_embed = nn.Embedding(vocab_size, d_model)
        self.pos_embed = nn.Embedding(maxlen, d_model)
        self.seg_embed = nn.Embedding(n_segments, d_model)
        self.norm = nn.LayerNorm(d_model)

    def forward(self, x, seg):
        seq_len = x.size(1)
        pos = torch.arange(seq_len, dtype=torch.long)
        pos = pos.unsqueeze(0).expand_as(x)
        embedding = self.tok_embed(x) + self.pos_embed(pos) + self.seg_embed(seg)
        return self.norm(embedding)
```

executed in 6ms, finished 11:43:43 2020-06-28

4.4 编码器定义

In [4]:

```
class EncoderLayer(nn.Module):
    def __init__(self):
        super(EncoderLayer, self).__init__()
        self.enc_self_attn = MultiHeadAttention()
        self.pos_ffn = PoswiseFeedForwardNet()

    def forward(self, enc_inputs, enc_self_attn_mask):
        enc_outputs, attn = self.enc_self_attn(enc_inputs, enc_inputs, enc_inputs,
        enc_outputs = self.pos_ffn(enc_outputs)
        return enc_outputs, attn
```

executed in 5ms, finished 11:41:59 2020-06-28

4.5 缩放点积、多头注意力机制与前馈神经网络

In [5]:

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

    def forward(self, Q, K, V, attn_mask):
        scores = torch.matmul(Q, K.transpose(-1, -2)) / np.sqrt(d_k)
        scores.masked_fill_(attn_mask, -1e9)
        attn = nn.Softmax(dim=-1)(scores)
        context = torch.matmul(attn, V)
        return context, attn

class MultiHeadAttention(nn.Module):
    def __init__(self):
        super(MultiHeadAttention, self).__init__()
        self.W_Q = nn.Linear(d_model, d_k * n_heads)
        self.W_K = nn.Linear(d_model, d_k * n_heads)
        self.W_V = nn.Linear(d_model, d_v * n_heads)

    def forward(self, Q, K, V, attn_mask):
        residual, batch_size = Q, Q.size(0)
        q_s = self.W_Q(Q).view(batch_size, -1, n_heads, d_k).transpose(1, 2)
        k_s = self.W_K(K).view(batch_size, -1, n_heads, d_k).transpose(1, 2)
        v_s = self.W_V(V).view(batch_size, -1, n_heads, d_v).transpose(1, 2)

        attn_mask = attn_mask.unsqueeze(1).repeat(1, n_heads, 1, 1)

        context, attn = ScaledDotProductAttention()(q_s, k_s, v_s, attn_mask)
        context = context.transpose(1, 2).contiguous().view(batch_size, -1, n_heads
        output = nn.Linear(n_heads * d_v, d_model)(context)
        return nn.LayerNorm(d_model)(output + residual), attn

class PoswiseFeedForwardNet(nn.Module):
    def __init__(self):
        super(PoswiseFeedForwardNet, self).__init__()
        self.fc1 = nn.Linear(d_model, d_ff)
        self.fc2 = nn.Linear(d_ff, d_model)

    def forward(self, x):
        return self.fc2(gelu(self.fc1(x)))

def gelu(x):
    return x * 0.5 * (1.0 + torch.erf(x / math.sqrt(2.0)))
```

executed in 20ms, finished 11:42:02 2020-06-28

4.6 模型定义

In [7]:

```python
class BERT(nn.Module):
    def __init__(self):
        super(BERT, self).__init__()
        self.embedding = Embedding()
        self.layers = nn.ModuleList([EncoderLayer() for _ in range(n_layers)])
        self.fc = nn.Linear(d_model, d_model)
        self.activ1 = nn.Tanh()
        self.linear = nn.Linear(d_model, d_model)
        self.activ2 = gelu
        self.norm = nn.LayerNorm(d_model)
        self.classifier = nn.Linear(d_model, 2)

        embed_weight = self.embedding.tok_embed.weight
        n_vocab, n_dim = embed_weight.size()
        self.decoder = nn.Linear(n_dim, n_vocab, bias=False)
        self.decoder.weight = embed_weight
        self.decoder_bias = nn.Parameter(torch.zeros(n_vocab))

    def forward(self, input_ids, segmetn_ids, masked_pos):
        output = self.embedding(input_ids, segmetn_ids)
        enc_self_attn_mask = get_attn_pad_mask(input_ids, input_ids)
        for layer in self.layers:
            output, enc_self_attn = layer(output, enc_self_attn_mask)
        h_pooled = self.activ1(self.fc(output[:, 0]))
        logits_clsf = self.classifier(h_pooled)

        masked_pos = masked_pos[:, :, None].expand(-1, -1, output.size(-1))
        h_masked = torch.gather(output, 1, masked_pos)
        h_masked = self.norm(self.activ2(self.linear(h_masked)))
        logits_lm = self.decoder(h_masked) + self.decoder_bias

        return logits_lm, logits_clsf

def get_attn_pad_mask(seq_q, seq_k):
    batch_size, len_q = seq_q.size()
    batch_size, len_k = seq_k.size()

    pad_attn_mask = seq_k.data.eq(0).unsqueeze(1)
    return pad_attn_mask.expand(batch_size, len_q, len_k)
```

executed in 12ms, finished 11:43:33 2020-06-28

4.7 模型训练及测试

In [12]:

```python
model = BERT()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

batch = make_batch()
input_ids, segment_ids, masked_tokens, masked_pos, isNext = zip(*batch)
input_ids, segment_ids, masked_tokens, masked_pos, isNext = \
    torch.LongTensor(input_ids),  torch.LongTensor(segment_ids), torch.LongTensor(m
    torch.LongTensor(masked_pos), torch.LongTensor(isNext)

for epoch in range(100):
    optimizer.zero_grad()
    logits_lm, logits_clsf = model(input_ids, segment_ids, masked_pos)
    loss_lm = criterion(logits_lm.transpose(1, 2), masked_tokens) # for masked LM
    loss_lm = (loss_lm.float()).mean()
    loss_clsf = criterion(logits_clsf, isNext) # for sentence classification
    loss = loss_lm + loss_clsf
    if (epoch + 1) % 10 == 0:
        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.6f}'.format(loss))
    loss.backward()
    optimizer.step()

# Predict mask tokens ans isNext
input_ids, segment_ids, masked_tokens, masked_pos, isNext = batch[0]
print(text)
print([number_dict[w] for w in input_ids if number_dict[w] != '[PAD]'])

logits_lm, logits_clsf = model(torch.LongTensor([input_ids]), \
                               torch.LongTensor([segment_ids]), torch.LongTensor([m
logits_lm = logits_lm.data.max(2)[1][0].data.numpy()
print('masked tokens list : ',[pos for pos in masked_tokens if pos != 0])
print('predict masked tokens list : ',[pos for pos in logits_lm if pos != 0])

logits_clsf = logits_clsf.data.max(1)[1].data.numpy()[0]
print('isNext : ', True if isNext else False)
print('predict isNext : ',True if logits_clsf else False)
```

executed in 1m 41.2s, finished 11:55:23 2020-06-28

```
Epoch: 0010 cost = 23.786695
Epoch: 0020 cost = 16.356310
Epoch: 0030 cost = 15.282166
Epoch: 0040 cost = 11.731420
Epoch: 0050 cost = 4.075703
Epoch: 0060 cost = 5.408222
Epoch: 0070 cost = 7.387890
Epoch: 0080 cost = 6.051405
Epoch: 0090 cost = 7.315189
Epoch: 0100 cost = 4.481243
Hello, how are you? I am Romeo.
Hello, Romeo My name is Juliet. Nice to meet you.
Nice meet you too. How are you today?
Great. My baseball team won the competition.
Oh Congratulations, Juliet
Thanks you Romeo
['[CLS]', 'great', 'my', 'baseball', 'team', 'won', 'the', 'competition', '[SEP]', 'g
reat', '[MASK]', 'baseball', '[MASK]', 'won', 'romeo', 'competition', '[SEP]']
masked tokens list :  [23, 28, 27]
predict masked tokens list :  [23, 23, 23]
isNext :  False
predict isNext :  True
```

In [ ]:

```python

```