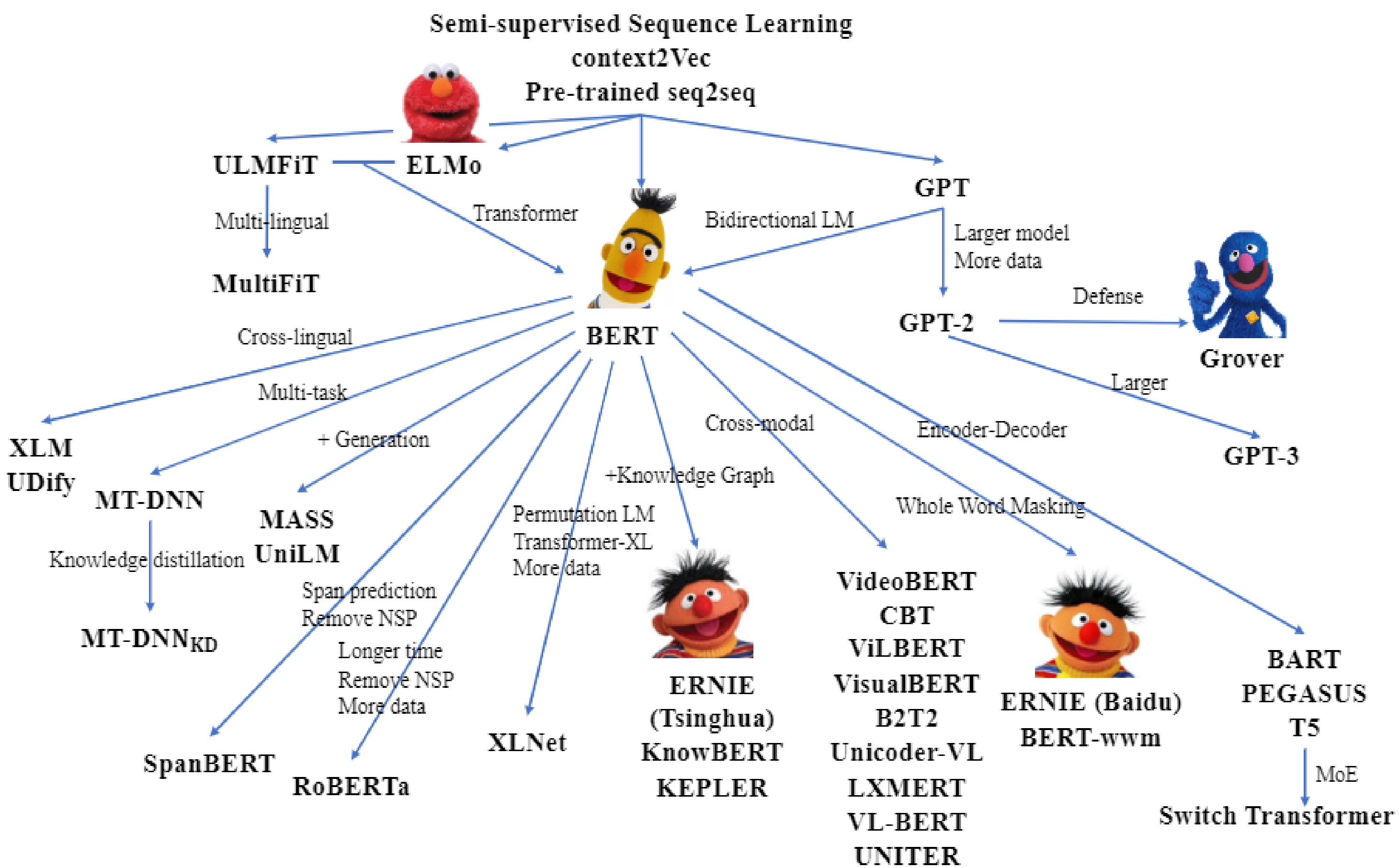
预训练模型

赵洲

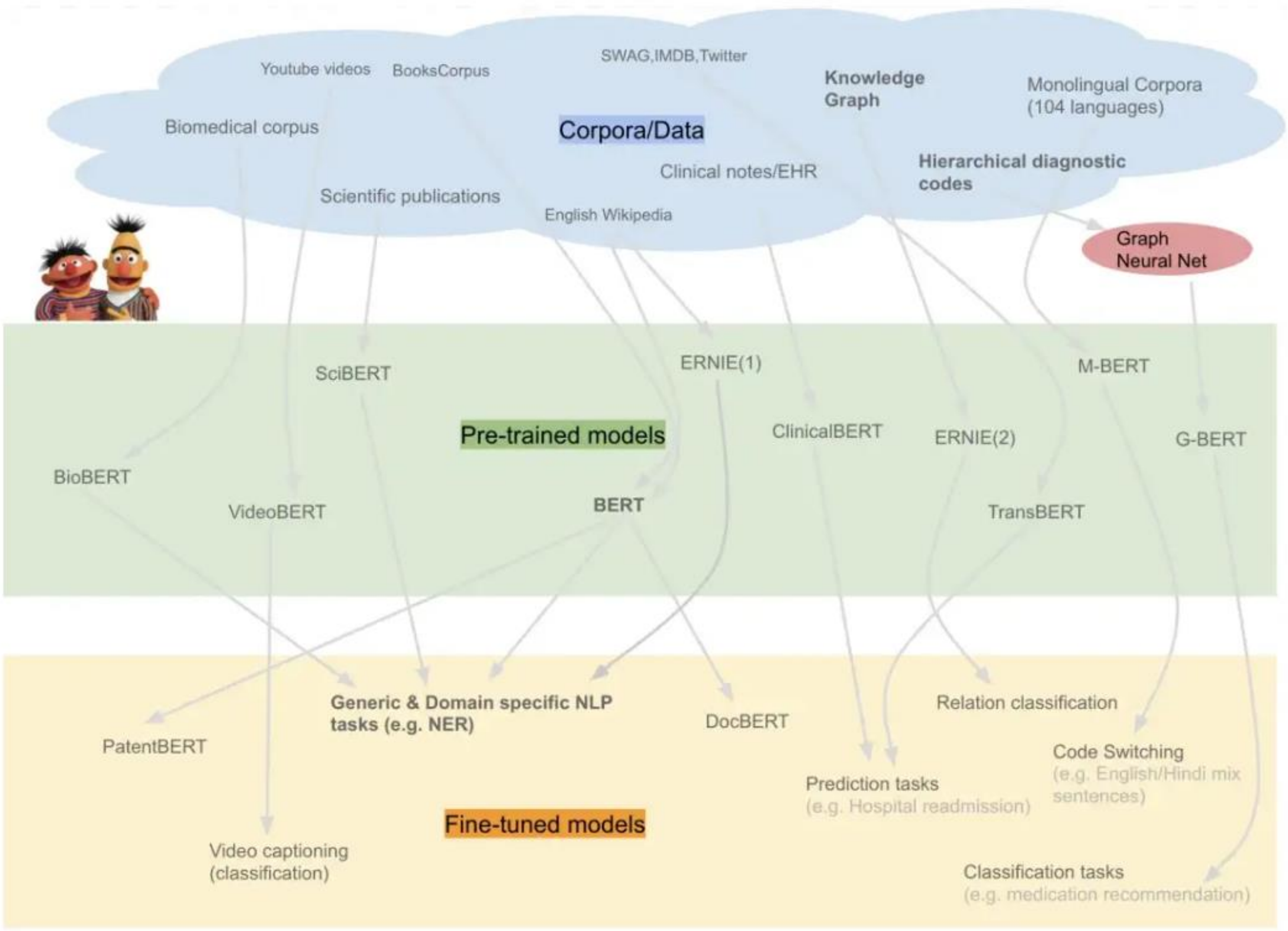
浙江大学计算机学院

什么是预训练模型？

- 预训练模型主要基于迁移学习，通过从多个源任务中获取重要的知识，再应用到目标任务中。

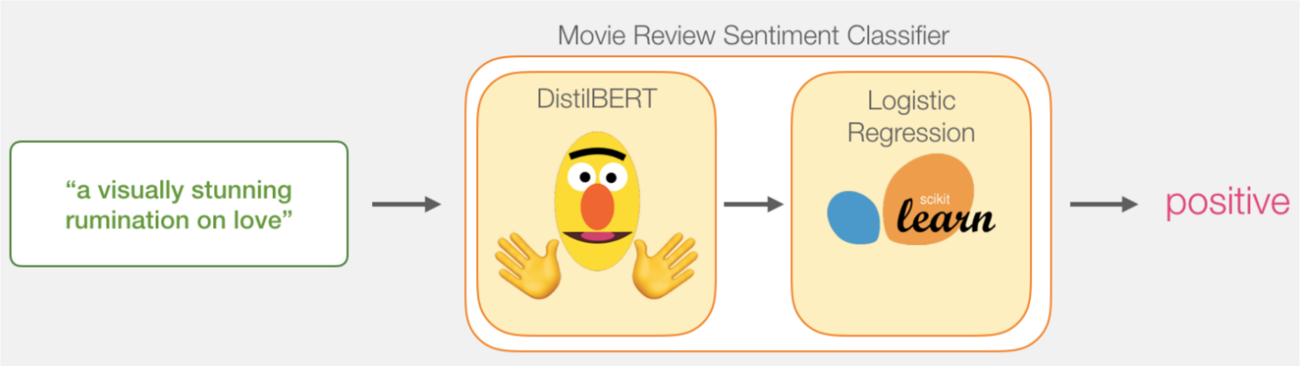


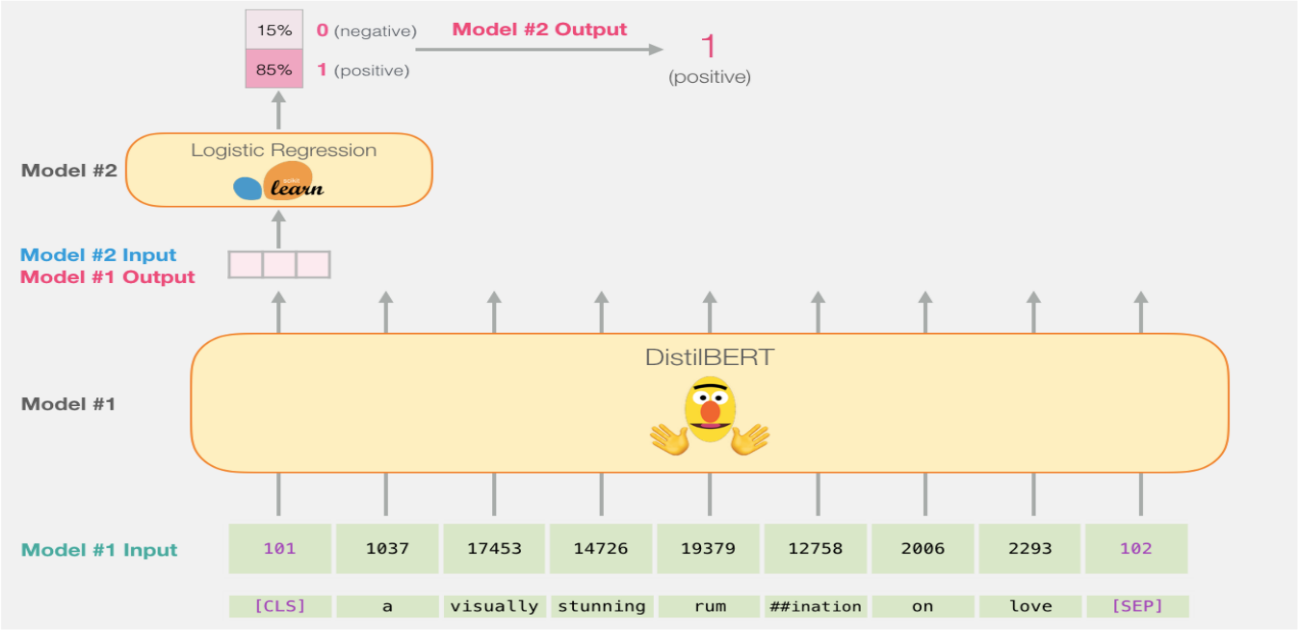
BERT的研究动机



Youtube videos
BooksCorpus
SWAG,IMDB,Twitter
Knowledge Graph
Monolingual Corpora (104 languages)
Biomedical corpus
Corpora/Data
Clinical notes/EHR
Hierarchical diagnostic codes
Scientific publications
English Wikipedia
Graph Neural Net
SciBERT
ERNIE(1)
M-BERT
Pre-trained models
ClinicalBERT
ERNIE(2)
G-BERT
BioBERT
VideoBERT
BERT
TransBERT
Generic & Domain specific NLP tasks (e.g. NER)
DocBERT
Relation classification
PatentBERT
Code Switching (e.g. English/Hindi mix sentences)
Prediction tasks (e.g. Hospital readmission)
Fine-tuned models
Video captioning (classification)
Classification tasks (e.g. medication recommendation)

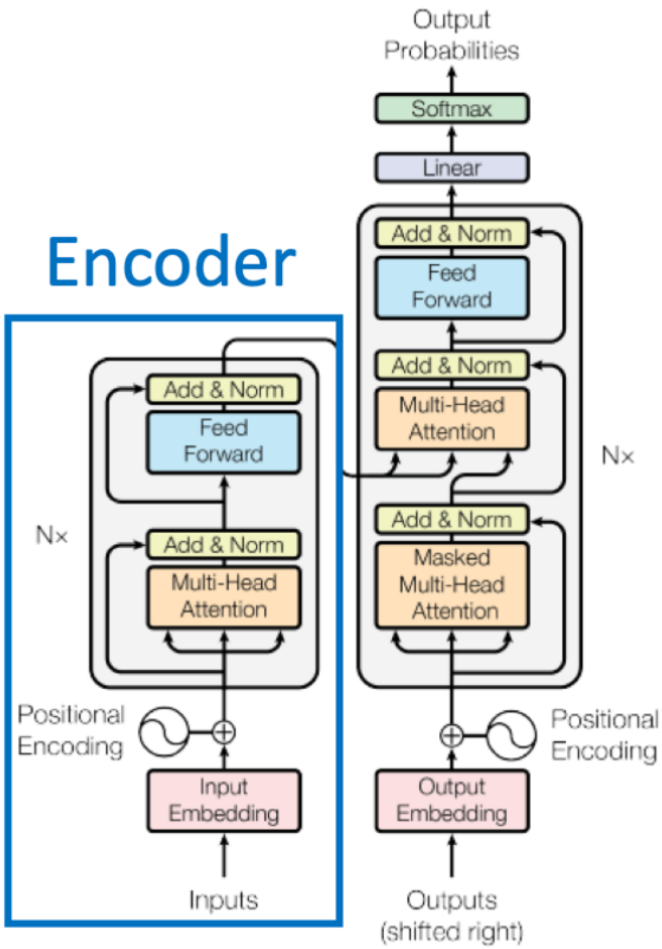
Pre-trained + Fine-Tuning



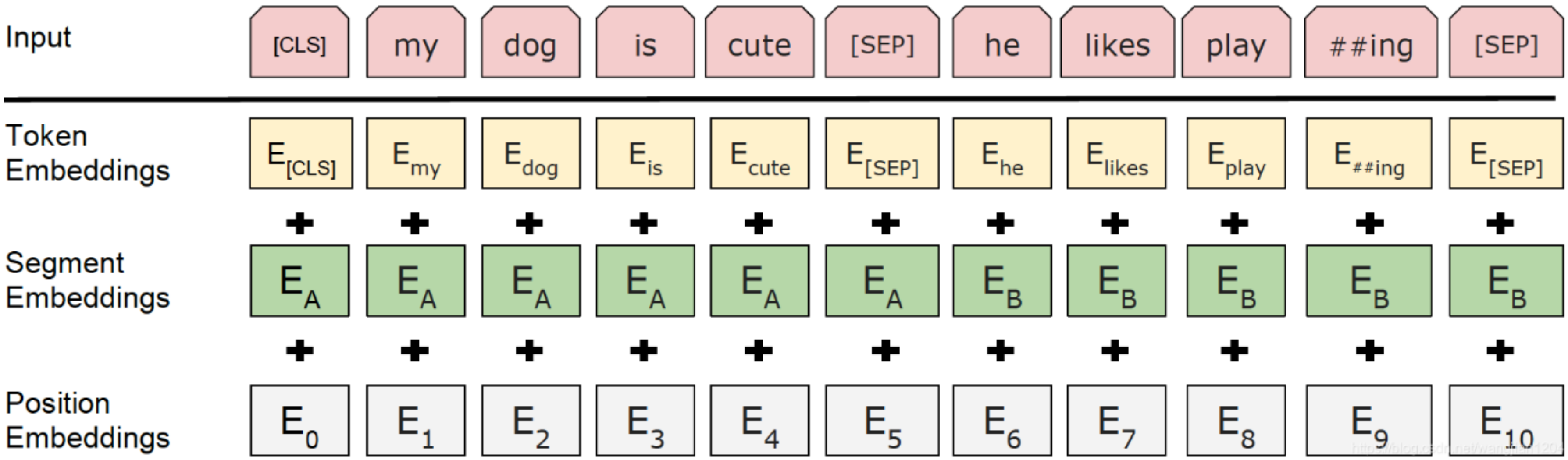


BERT网络结构

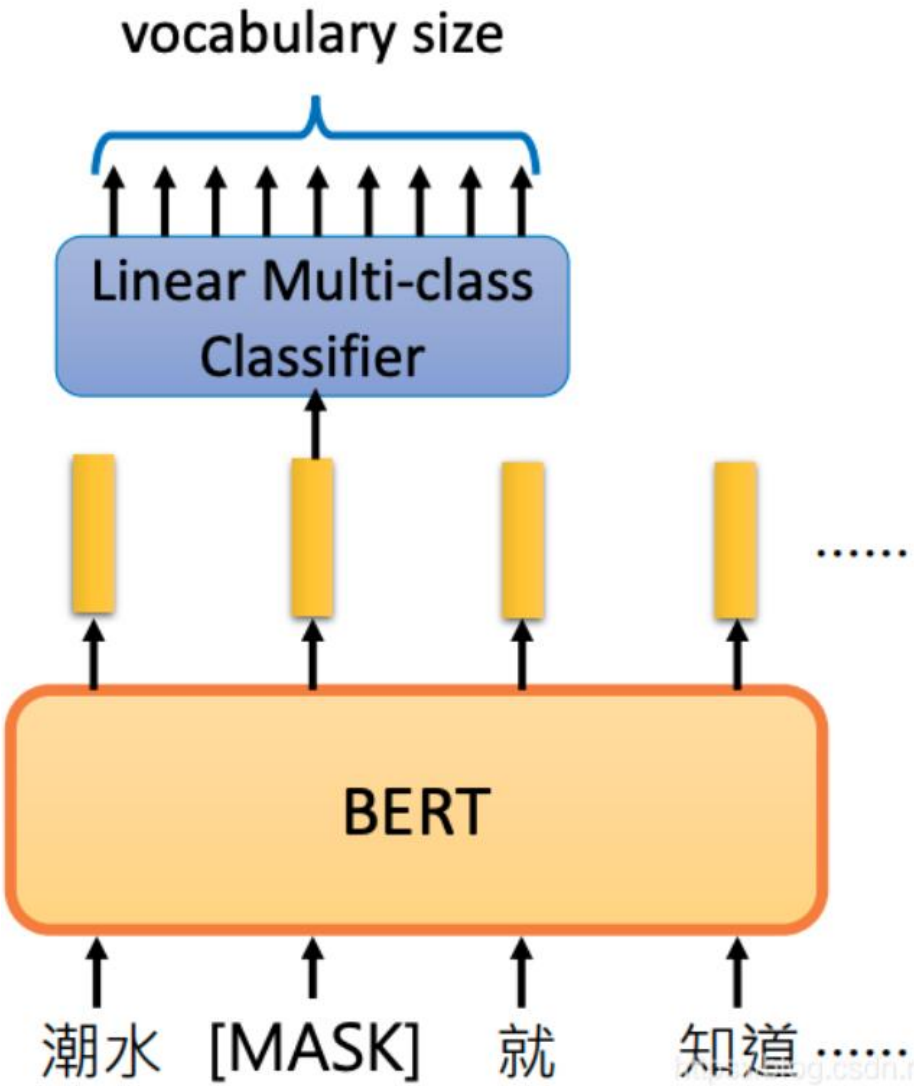
- BERT = 双向Transformer的Encoder，通过给定的语料生成每个词对应的Embedding向量。



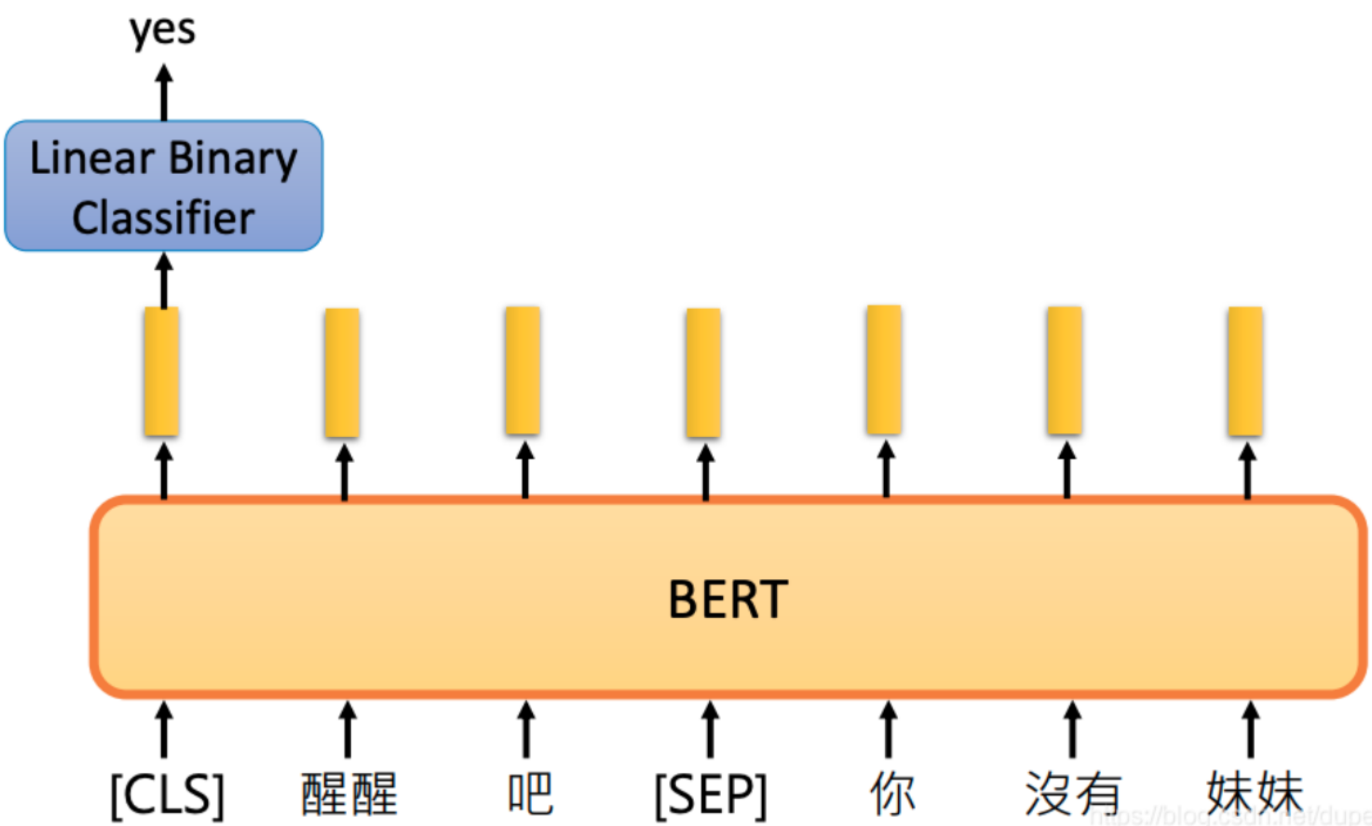
模型输入


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$
E_{my}
E_{dog}
E_{is}
E_{cute}
$E_{[SEP]}$
E_{he}
E_{likes}
E_{play}
$E_{\#\#ing}$
$E_{[SEP]}$
Segment Embeddings
E_A
E_A
E_A
E_A
E_A
E_A
E_B
E_B
E_B
E_B
E_B
Position Embeddings
E_0
E_1
E_2
E_3
E_4
E_5
E_6
E_7
E_8
E_9
E_{10}

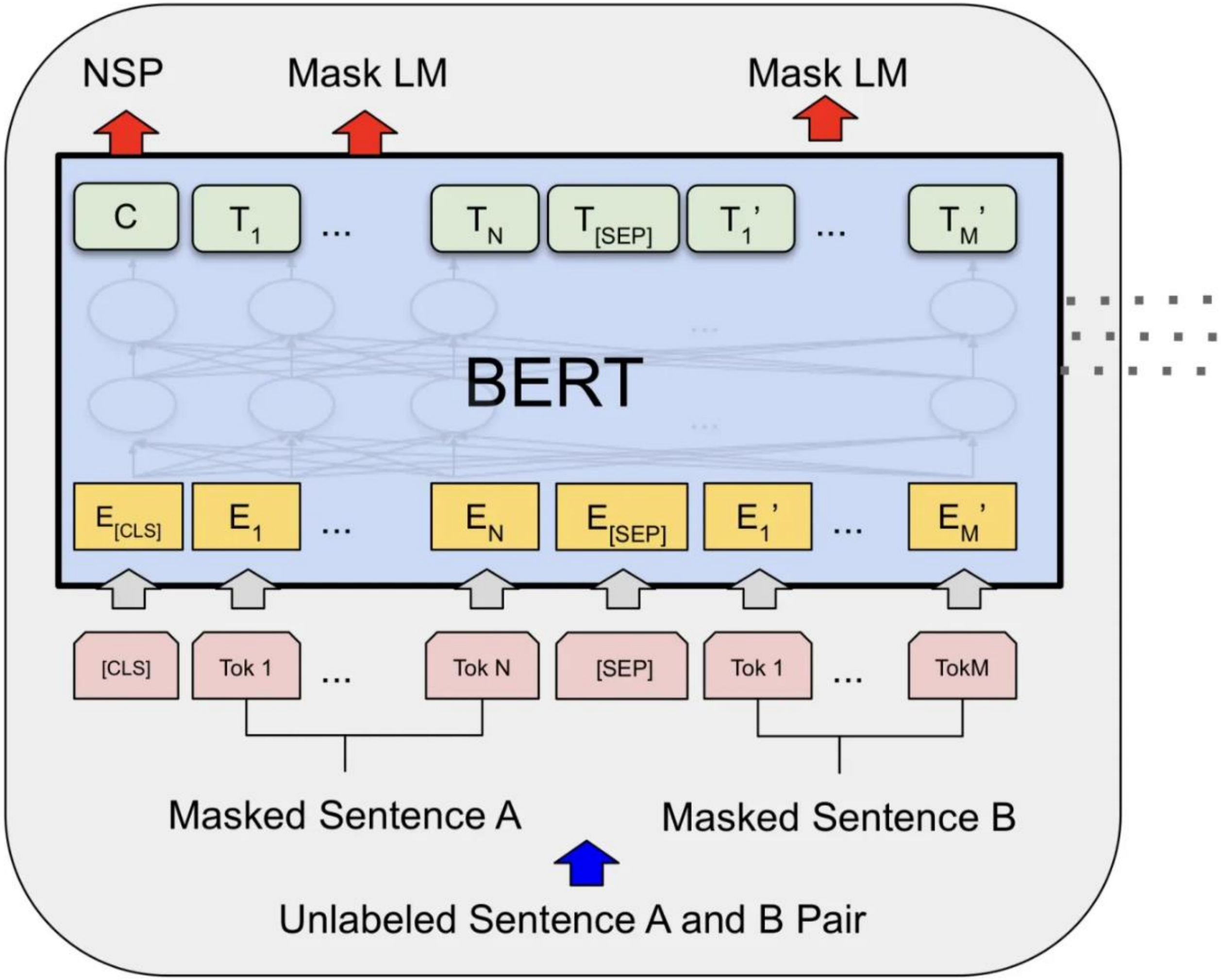
替代词预测


vocabulary size
Linear Multi-class Classifier
……
BERT
潮水 [MASK] 就 知道……

相临句预测



训练BERT



Pre-training

从Pre-training到Fine-Tuning

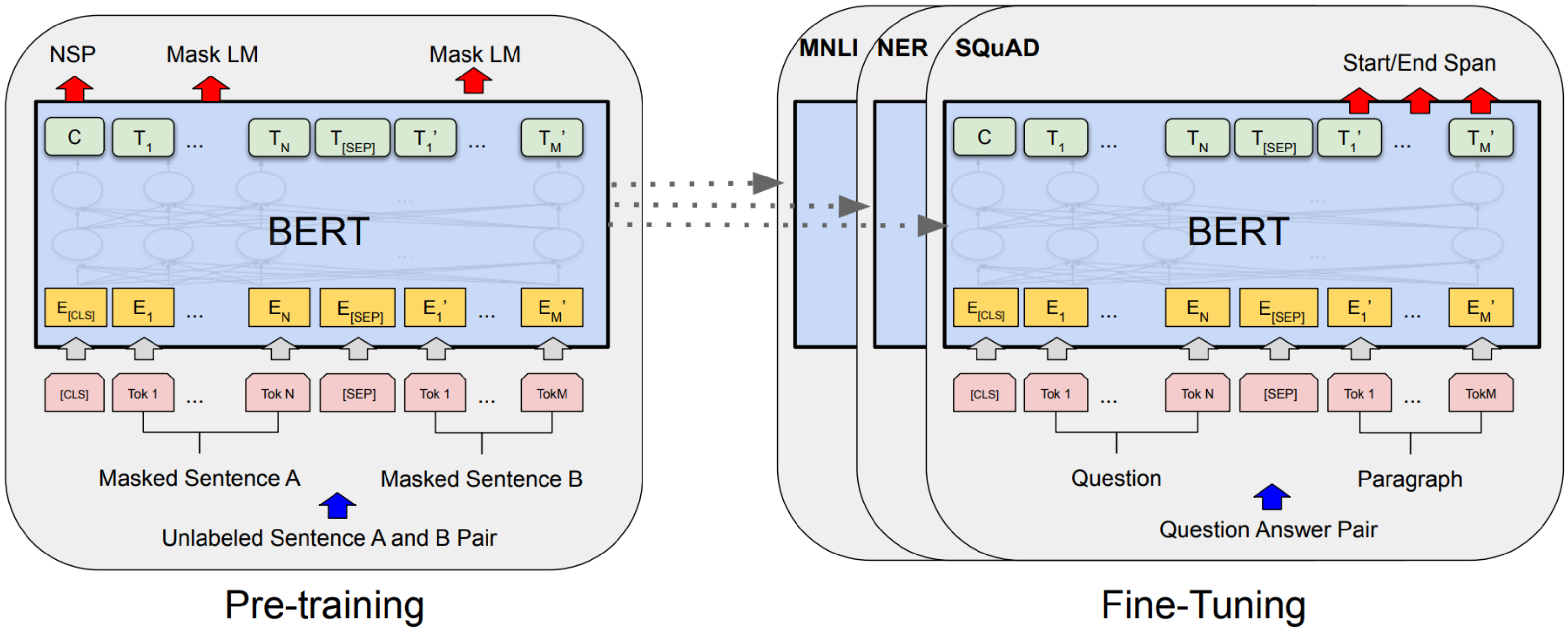
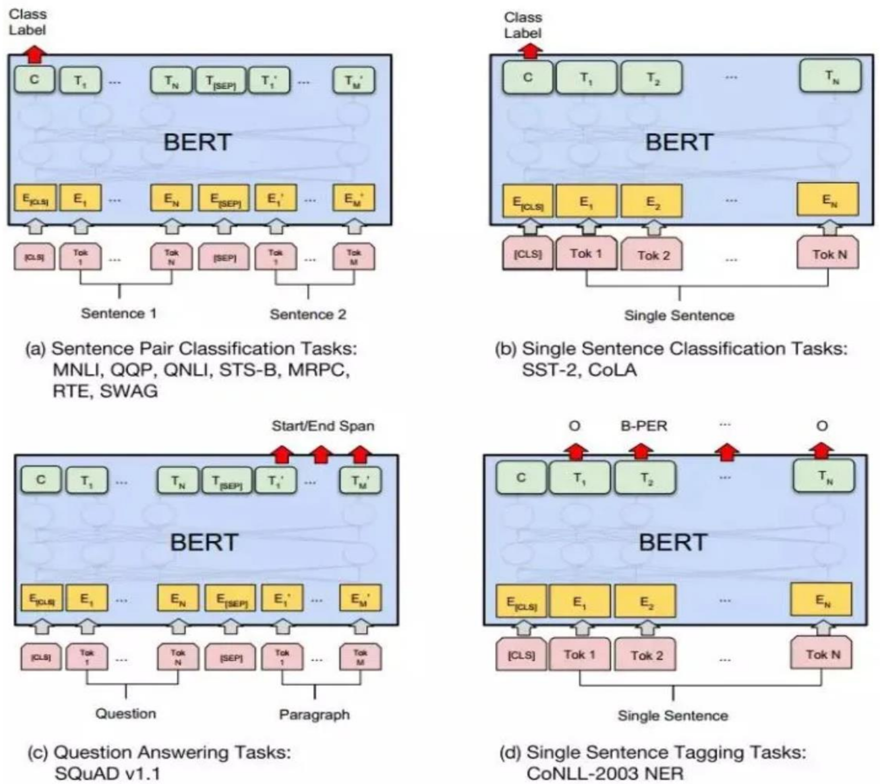


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

Fine-Tunning

- 取BERT模型中第一个CLS token的最终隐藏状态C, 加入新参数权重W, 下游任务可以被构建为：

$$P = softmax(CW^T)$$



实验结果

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

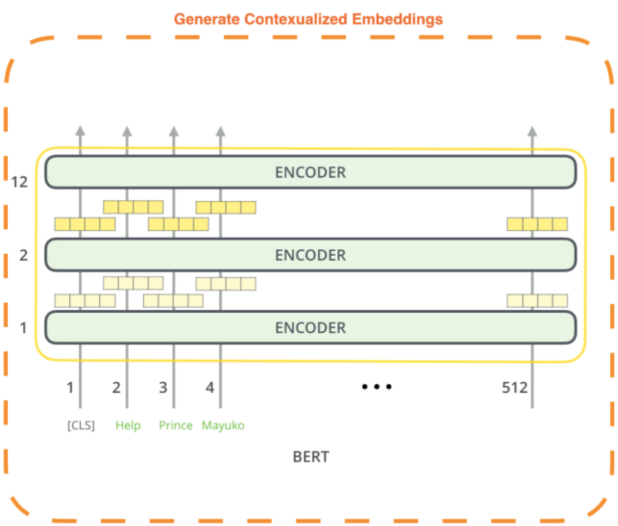
Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

实验结果

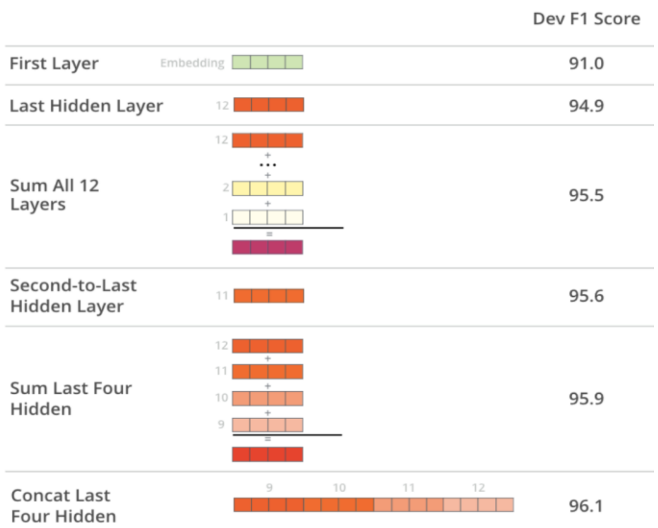
System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

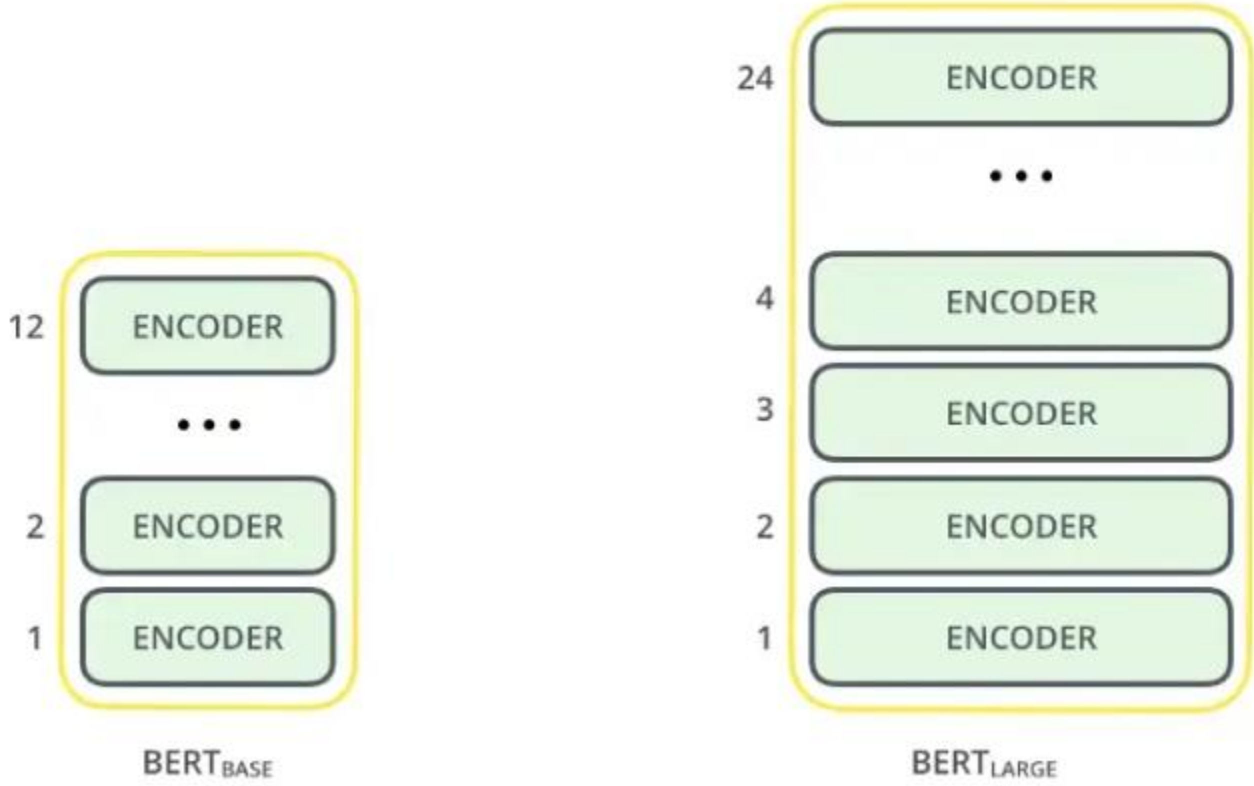
特征抽取


Generate Contexualized Embeddings
ENCODER
ENCODER
ENCODER
12
2
1
1
2
3
4
512
[CLS]
Help
Prince
Mayuko
BERT

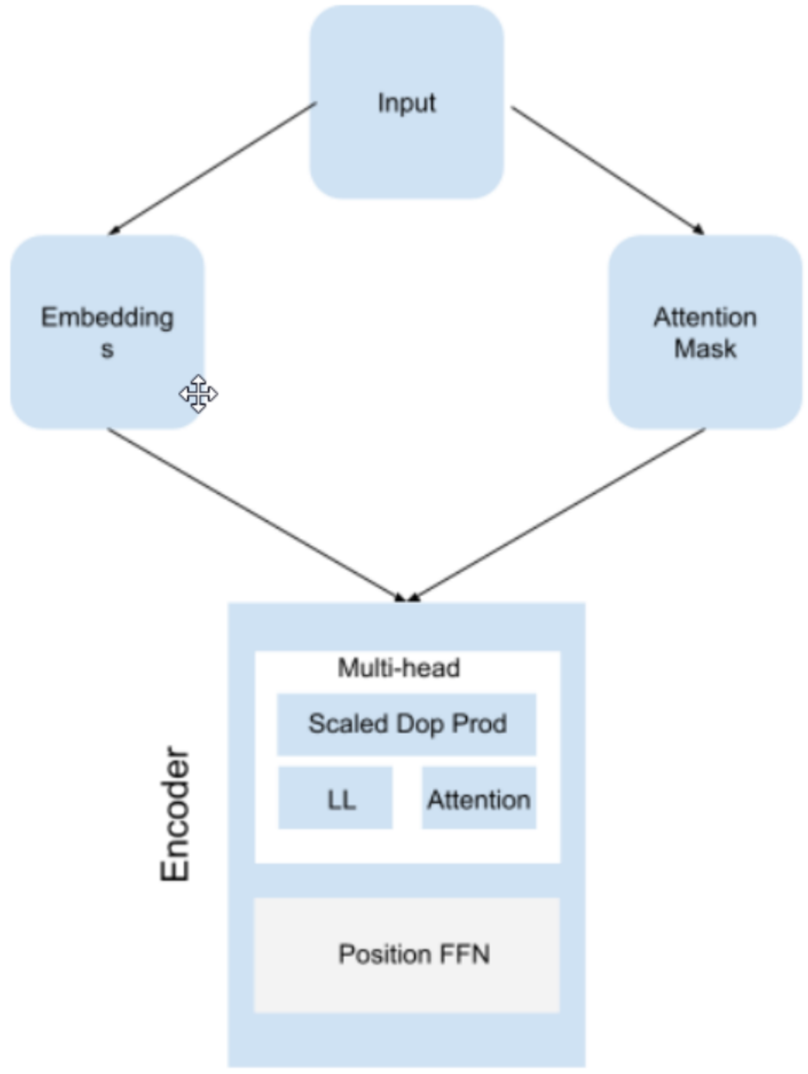

The output of each encoder layer along each token's path can be used as a feature representing that token.
Help
Prince
Mayuko
Help
But which one should we use?


Dev F1 Score
First Layer
Embedding
91.0
Last Hidden Layer
94.9
Sum All 12 Layers
95.5
Second-to-Last Hidden Layer
95.6
Sum Last Four Hidden
95.9
Concat Last Four Hidden
96.1

BERT版本


12
ENCODER
• • •
2
ENCODER
1
ENCODER
BERTBASE
24
ENCODER
• • •
4
ENCODER
3
ENCODER
2
ENCODER
1
ENCODER
BERTLARGE

代码结构



Input
Embedding
s
Attention
Mask
Encoder
Multi-head
Scaled Dop Prod
LL
Attention
Position FFN

BERT参数设置

```
maxlen = 30 # 句子的最大长度 cover住95% 不要看平均数 或者
batch_size = 6 # 每一组有多少个句子一起送进去模型
max_pred = 5  # max tokens of prediction
n_layers = 6 # number of Encoder of Encoder Layer
n_heads = 12 # number of heads in Multi-Head Attention
d_model = 768 # Embedding Size
d_ff = 3072  # 4*d_model, FeedForward dimension
d_k = d_v = 64  # dimension of K(=Q), V
n_segments = 2
```

预处理

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

```python
  sentences = re.sub("[.,!?\\-]", '', text.lower()).split('\n')  # filter '.',
   word_list = list(set(" ".join(sentences).split()))
```

```python
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)
```

掩码序列构建

```
# MASK LM
        n_pred =  min(max_pred, max(1, int(round(len(input_ids) * 0.15)))) # n_pred=3; 整
        cand_maked_pos = [i for i, token in enumerate(input_ids)
                          if token != word_dict['[CLS]'] and token != word_dict['[SEP]']]
        shuffle(cand_maked_pos)## 打乱顺序: cand_maked_pos=[6, 5, 17, 3, 1, 13, 16, 10, 1
        masked_tokens, masked_pos = [], []
        for pos in cand_maked_pos[:n_pred]:## 取其中的三个; masked_pos=[6, 5, 17] 注意这里
            masked_pos.append(pos)
            masked_tokens.append(input_ids[pos])
            if random() < 0.8:  # 80%
                input_ids[pos] = word_dict['[MASK]'] # make mask
            elif random() < 0.5:  # 10%
                index = randint(0, vocab_size - 1) # random index in vocabulary
                input_ids[pos] = word_dict[number_dict[index]] # replace
```

Embedding构建

```
class Embedding(nn.Module):
    def __init__(self):
        super(Embedding, self).__init__()
        self.tok_embed = nn.Embedding(vocab_size, d_model)  # token embedding
        self.pos_embed = nn.Embedding(maxlen, d_model)  # position embedding
        self.seg_embed = nn.Embedding(n_segments, d_model)  # segment(token type)
        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)  # (seq_len,) -> (batch_size, seq_len
        embedding = self.tok_embed(x) + self.pos_embed(pos) + self.seg_embed(seg)
        return self.norm(embedding)
```

聚焦掩码构建

```python
def get_attn_pad_mask(seq_q, seq_k):
    batch_size, len_q = seq_q.size()
    batch_size, len_k = seq_k.size()
    # eq(zero) is PAD token
    pad_attn_mask = seq_k.data.eq(0).unsqueeze(1)  # batch_
    return pad_attn_mask.expand(batch_size, len_q, len_k)
```

```
Output:
(tensor([False, False, False, False, False, False, False, False, False, False,
         False, False, False,  True,  True,  True,  True,  True,  True,  True,
          True,  True,  True,  True,  True,  True,  True,  True,  True,  True])
 tensor([ 1,  3, 26, 21, 14, 16, 12,  4,  2, 27,  3, 22,  2,  0,  0,  0,  0,  0
          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0]))
```

编码层实现

```
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_self_attn_mask) # enc_inputs to same Q,K,V
        enc_outputs = self.pos_ffn(enc_outputs) # enc_outputs: [batch_size x len_q x d_model]
        return enc_outputs, attn
```

多头聚焦机制

```
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):
        # q: [batch_size x len_q x d_model], k: [batch_size x len_k x d_model],
        residual, batch_size = Q, Q.size(0)
        # (B, S, D) -proj-> (B, S, D) -split-> (B, S, H, W) -trans-> (B, H, S,
        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) # attn_mask

        # context: [batch_size x n_heads x len_q x d_v], attn: [batch_size x n_
        context, attn = ScaledDotProductAttention()(q_s, k_s, v_s, attn_mask)
        context = context.transpose(1, 2).contiguous().view(batch_size, -1, n_h
        output = nn.Linear(n_heads * d_v, d_model)(context)


return nn.LayerNorm(d_model)(output + residual), attn # output: [batch_size x
```

聚焦机制

```python
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) # Fills elements of sel
        attn = nn.Softmax(dim=-1)(scores)
        context = torch.matmul(attn, V)
        return score, context, attn
```

BERT模型

```
class BERT(nn.Module):
    def __init__(self):
        super(BERT, self).__init__()
        self.embedding = Embedding() ## 词向量层，构建词表矩阵
        self.layers = nn.ModuleList([EncoderLayer() for _ in range(n_layers)]) ## 把N个encoder堆叠起来，具体encoder实
        self.fc = nn.Linear(d_model, d_model) ## 前馈神经网络-cls
        self.activ1 = nn.Tanh() ## 激活函数-cls
        self.linear = nn.Linear(d_model, d_model)#-mlm
        self.activ2 = gelu ## 激活函数--mlm
        self.norm = nn.LayerNorm(d_model)
        self.classifier = nn.Linear(d_model, 2)## cls 这是一个分类层，维度是从d_model到2，对应我们架构图中就是这种：
        # decoder is shared with embedding layer
        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, segment_ids, masked_pos):
        output = self.embedding(input_ids, segment_ids)## 生成input_ids对应的embdding；和segment_ids对应的embedding
        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)
        # output : [batch_size, len, d_model], attn : [batch_size, n_heads, d_mode, d_model]
        # it will be decided by first token(CLS)
        h_pooled = self.activ1(self.fc(output[:, 0])) # [batch_size, d_model]
        logits_clsf = self.classifier(h_pooled) # [batch_size, 2]

        masked_pos = masked_pos[:, :, None].expand(-1, -1, output.size(-1)) # [batch_size, max_pred, d_model]  其中
        # get masked position from final output of transformer.
        h_masked = torch.gather(output, 1, masked_pos) # masking position [batch_size, max_pred, d_model]
        h_masked = self.norm(self.activ2(self.linear(h_masked)))
        logits_lm = self.decoder(h_masked) + self.decoder_bias # [batch_size, max_pred, n_vocab]

        return logits_lm, logits_clsf
```

BERT模型训练

```
for epoch in range(100):
    optimizer.zero_grad()
    logits_lm, logits_clsf = model(input_ids, segment_ids, masked_pos)## logits_lm 【6, 5, 29】 bs*max_
    loss_lm = criterion(logits_lm.transpose(1, 2), masked_tokens) # for masked LM ;masked_tokens [6,5]
    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()
```

基于BERT的分类

```
import transformers

class BERTClassification(nn.Module):
    def __init__ (self):
        super(BERTClassification, self).__init__()
        self.bert = transformers.BertModel.from_pretrained('bert-base-cased')
        self.bert_drop = nn.Dropout(0.4)
        self.out = nn.Linear(768, 1)

    def forward(self, ids, mask, token_type_ids):
        _, pooledOut = self.bert(ids, attention_mask = mask,
                                token_type_ids=token_type_ids)
        bertOut = self.bert_drop(pooledOut)
        output = self.out(bertOut)

        return output
```