TITLE:
Multi-grained Attention Network for Aspect-Level Sentiment Classification

GET:

论文技术核心在Multi-grained Attention

论文任务是基于aspect的情感分类

摘要

- 提出问题:

Existing approaches mostly adopt coarse-grained attention mechanism, which may bring information loss if the aspect has multiple words or larger context.

- 解决方案:

We propose a fine-grained attention mechanism, which can capture the word-level interaction between aspect and context.

- MGAN framework= coarse-grained + fine-grained

摘要

- 提出以前方法不足

unlike previous works which train each aspect with its context separately

- 对不足的修正（创新点）

design an aspect alignment loss to depict the aspect-level interactions among the aspects that have the same context.

摘要

- 数据集
- On three datasets: laptop and restaurant are from SemEval 2014, and the last one is a twitter dataset

- 模型效果
- Outperforms the state-of-the-art methods on all three datasets
- Conduct experiments :aspect alignment loss, which bring extra useful information and further improve the performance

✓ Laptops_Train.xml.seg 数据集

- I charge it at night and skip taking the T with me because of the good battery life .
- cord
- 0

- I charge it at night and skip taking the cord with me because of the good T .
- battery life
- 1

✓ Our Approach

- 3.1 任务描述
- 事先给定一句话 $s = \{w_1, w_2, \cdots, w_N\}$ 包含N个单词
- Aspect list A = $\{a_1, \cdots, a_k\}$ and 每个aspect 记作 a_i

$$a_i = \{w_{i_1}, \cdots, w_{i_M}\}$$

- 注意：每个aspect is a subsequence of sentence s

- 该论文model任务： evaluates sentiment polarity of the sentence s with respect to each aspect ai
- 该论文model架构： Multi-grained Attention Network (MGAN)

✓ Our Approach

- 3.2 Input Embedding Layer

1. 预训练词向量GloVe，获取固定维度向量 $\mathbb{L} \in \mathbb{R}^{d_v \times |V|}$
2. 将每个单词映射到高纬度向量空间
3. 比如：glove dims = 50， maps vector space dims = 300

✓ Our Approach

- 3.3 Contextual Layer

BiLSTM : capture the temporal interactions among words

$$i_t = \sigma(\overrightarrow{W}_i \cdot [\overrightarrow{h}_{t-1}, \overrightarrow{x}_t] + \overrightarrow{b}_i) \tag{1}$$

$$f_t = \sigma(\overrightarrow{W}_f \cdot [\overrightarrow{h}_{t-1}, \overrightarrow{x}_t] + \overrightarrow{b}_f) \tag{2}$$

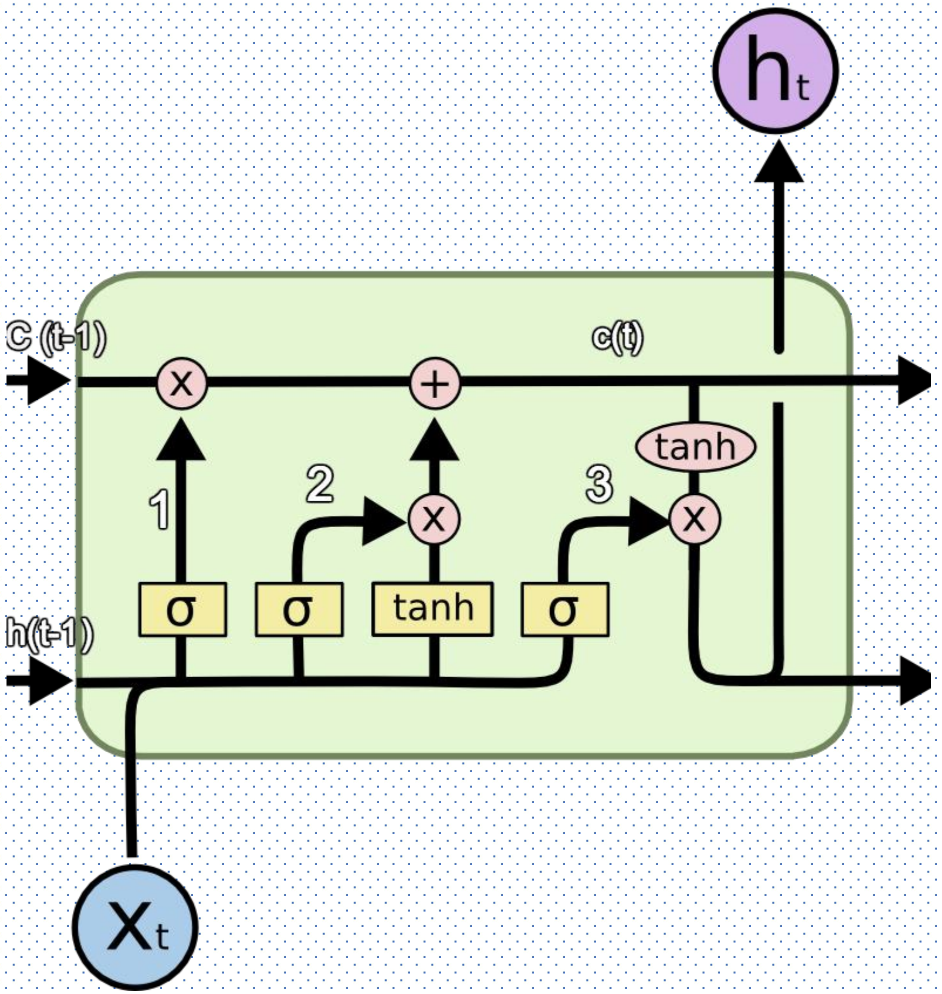
$$o_t = \sigma(\overrightarrow{W}_o \cdot [\overrightarrow{h}_{t-1}, \overrightarrow{x}_t] + \overrightarrow{b}_o) \tag{3}$$

$$g_t = tanh(\overrightarrow{W}_g \cdot [\overrightarrow{h}_{t-1}, \overrightarrow{x}_t] + \overrightarrow{b}_g) \tag{4}$$

$$\overrightarrow{c}_t = f_t * \overrightarrow{c}_{t-1} + i_t * g_t \tag{5}$$

$$\overrightarrow{h}_t = o_t * tanh(\overrightarrow{c}_t) \tag{6}$$

补充：LSTM 神经网络结构图



✓ Our Approach

- 3.3 Contextual Layer
- 论文又提出个小创新点，其建立在这个假设上面：

context words with closer distance to an aspect may have higher influence to the aspect

- 解决方案：

utilize the position encoding mechanism to simulate the observation

$$w_t = 1 - \frac{l}{N - M + 1} \tag{7}$$

✓ Our Approach

- 3.4 Multi-grained Attention Layer
- 首先提出之前广泛使用的attention的不足:

Adopt coarse-grained attentions, which simply use the averaged aspect/context vector as the guide to learn the attention weights on context/aspect

本论文解决方案:

We propose the fine-grained attention mechanism, which is responsible for linking and fusing information from the aspect and context words.

final representation = concat (coarse-grained + fine-grained)

✓ Our Approach

- 3.4 Multi-grained Attention Layer

- 论文创新点：引入alignment loss机制

实验观察，aspects之间的关系能带来额外有价值信息，所以提出该机制，用来强化关注相同上下文但情感极性不同的aspects的区别。

✓ Our Approach

- 3.4 Multi-grained Attention Layer
- C-Aspect2Context 计算过程

Sca : score function,计算权重weight 表示context Word对aspect极性的重要程度。

Wca : 表示attention weight matrix

$$s_{ca}(Q_{avg}, H_i) = Q_{avg} * W_{ca} * H_i \qquad (8)$$

$$a_i^{ca} = \frac{exp(s_{ca}(Q_{avg}, H_i))}{\sum_{k=1}^{N} exp(s_{ca}(Q_{avg}, H_k))} \qquad (9)$$

$$m^{ca} = \sum_{i=1}^{N} a_i^{ca} \cdot H_i \qquad (10)$$

✓ Our Approach

- 3.4 Multi-grained Attention Layer
- C-Context2Aspect 计算过程

$$s_{cc}(H_{avg}, Q_i) = H_{avg} * W_{cc} * Q_i \tag{11}$$

$$a_i^{cc} = \frac{exp(s_{cc}(H_{avg}, Q_i))}{\sum_{k=1}^{M} exp(s_{cc}(H_{avg}, Q_k))} \tag{12}$$

$$m^{cc} = \sum_{i=1}^{M} a_i^{cc} \cdot Q_i \tag{13}$$

✓ Our Approach

- 3.4 Multi-grained Attention Layer
- Fine-grained Attention 介绍
- alignment matrix == similarity matrix

$$U_{ij} = W_u([H_i; Q_j; H_i * Q_j]) \qquad (14)$$

- Where Wu ∈R[1* 6d维度] is the weight matrix, [;] is the vector concatenation across row, * is the elementwise multiplication.

✓ Our Approach

- Fine-grained Attention
- F-Aspect2Context 意义和计算过程
- 意义： estimates which context word has the closest similarity to one of the aspect word and are hence critical for determining the sentiment
- U： $\mathbb{R}^{N*M}$ a^{fa} ：N维

$$s_i^{fa} = max(U_{i,:}) \quad (15)$$

$$a_i^{fa} = \frac{exp(s_i^{fa})}{\sum_{k=1}^{N} exp(s_k^{fa})} \quad (16)$$

$$m^{fa} = \sum_{i=1}^{N} a_i^{fa} \cdot H_i \quad (17)$$

✓ Our Approach

- Fine-grained Attention
- F-Context2Aspect 意义和计算过程
- 意义： measures which aspect words are most relevant to each context word

$$a_{ij}^{fc} = \frac{exp(U_{ij})}{\sum_{k=1}^{M} exp(U_{ik})} \qquad (18)$$

$$q_i^{fc} = \sum_{j=1}^{M} a_{ij}^{fc} \cdot Q_j \qquad (19)$$

$$m^{fc} = Pooling([q_1^{fc}, \cdots, q_N^{fc}]) \qquad (20)$$

✓ Our Approach

- Fine-grained Attention
- 暂无补充 pass

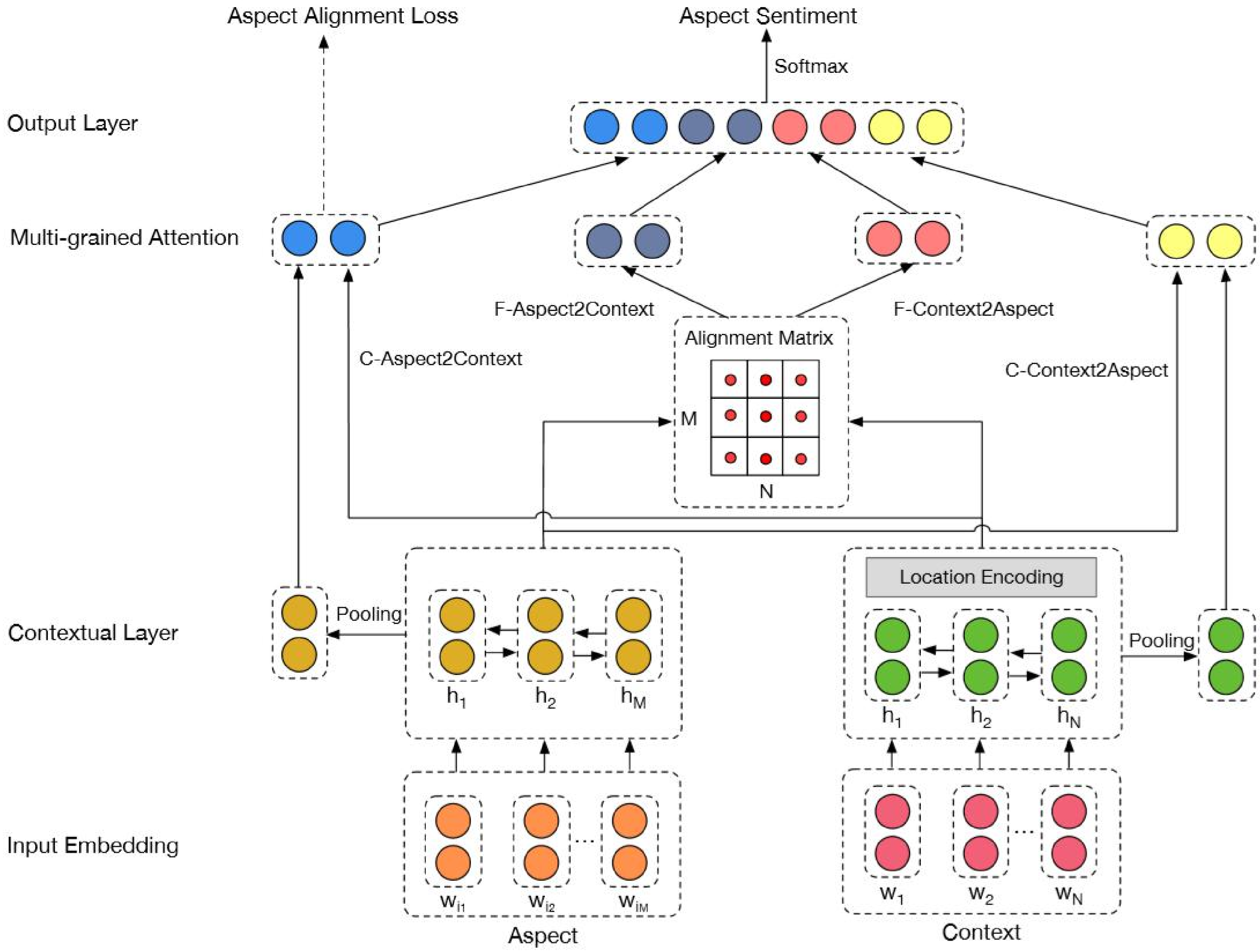


Figure 1: The architecture of the proposed multi-grained attention network.

✓ Our Approach

- 3.5 Output Layer
- final representation： concatenate both the coarse-grained and fine-grained attention vectors
- fed to a soft-max layer for determining the aspect sentiment polarity

$$m = [m^{ca}; m^{cc}; m^{fa}; m^{fc}] \tag{21}$$

$$p = softmax(W_p * m + b_p) \tag{22}$$

- Here we set C = 3, which is the number of aspect sentiment classes

✓ Our Approach

- 3.6 Model Training
- Aspect Alignment Loss

With the constraint of aspect alignment loss, each aspect will pay more attention on the important words through the comparisons with other related aspects.

$$d_{ij} = \sigma(W_d([Q_i; Q_j; Q_i * Q_j])) \quad (23)$$

$$\mathcal{L}_{align} = -\sum_{i=1}^{M-1} \sum_{j=i+1, y_i \neq y_j}^{M} \sum_{k=1}^{N} d_{ij} \cdot (a_{ik}^{ca} - a_{jk}^{ca})^2 \quad (24)$$

$$\mathcal{L} = -\sum_{i=1}^{C} y_i log(p_i) + \beta \mathcal{L}_{align} + \lambda \|\Theta\|^2 \quad (25)$$

✓ Our Approach

- 3.6 Model Training
- Aspect Alignment Loss

- In terms of the previous example, the aspect "speaker quality" should pay more attention on "lacking" and less attention on "like", compared with aspect "Mac OS" due to their different sentiment polarities.
- Example "I like coming back to Mac OS but this laptop is lacking in speaker quality compared to my $400 old HP laptop"

✓ Our Approach

- 3.6 Model Training
- 损失函数：The final loss function is consisting of the cross-entropy loss, aspect alignment loss and regularization item as follows:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i log(p_i) + \beta \mathcal{L}_{align} + \lambda \|\Theta\|^2 \quad (25)$$

- We employ the stochastic gradient descent (SGD) optimizer to compute and update the training parameters. Inaddition,we utilize dropout strategy to **avoid overfitting**

✓ Experiments

Method	Laptop		Restaurant		Twitter	
	Acc	Macro-F1	Acc	Macro-F1	Acc	Macro-F1
Majority	0.5350	0.3333	0.6500	0.3333	0.5000	0.3333
Feature-SVM	0.7049	-	0.8016	-	0.6340	0.6330
ATAE-LSTM	0.6870	-	0.7720	-	-	-
TD-LSTM	0.7183	0.6843	0.7800	0.6673	0.6662	0.6401
IAN	0.7210	-	0.7860	-	-	-
MemNet	0.7237	-	0.8032	-	0.6850	0.6691
BILSTM-ATT-G	0.7312	0.6980	0.7973	0.6925	0.7038	0.6837
RAM	0.7449	0.7135	0.8023	0.7080	0.6936	0.6730
MGAN	**0.7539**	**0.7247**	**0.8125**	**0.7194**	**0.7254**	**0.7081**

✓ Experiments

Method	Laptop		Restaurant		Twitter	
	Acc	Macro-F1	Acc	Macro-F1	Acc	Macro-F1
MGAN-C	0.7273	0.6933	0.8054	0.7099	0.7153	0.6952
MGAN-F	0.7398	0.7082	0.8000	0.7092	0.7110	0.6918
MGAN-CF	0.7445	0.7121	0.8089	0.7135	0.7254*	0.7081*
MGAN	**0.7539**	**0.7247**	**0.8125**	**0.7194**	**0.7254**	**0.7081**

✓ Experiments

✓ code

```
# Hyper Parameters
parser = argparse.ArgumentParser()
parser.add_argument('--model_name', default='mgan', type=str)
parser.add_argument('--dataset', default='laptop', type=str, help='twitter, restaurant, laptop')
parser.add_argument('--optimizer', default='adam', type=str)
parser.add_argument('--initializer', default='xavier_uniform_', type=str)
parser.add_argument('--learning_rate', default=2e-5, type=float, help='try 5e-5, 2e-5 for BERT, 1e-3 for others')
parser.add_argument('--dropout', default=0.1, type=float)
parser.add_argument('--l2reg', default=0.01, type=float)
parser.add_argument('--num_epoch', default=100, type=int, help='try larger number for non-BERT models')
parser.add_argument('--batch_size', default=64, type=int, help='try 16, 32, 64 for BERT models')
parser.add_argument('--log_step', default=5, type=int)
parser.add_argument('--embed_dim', default=50, type=int)
parser.add_argument('--hidden_dim', default=300, type=int)
parser.add_argument('--bert_dim', default=768, type=int)
parser.add_argument('--pretrained_bert_name', default='bert-base-uncased', type=str)
parser.add_argument('--max_seq_len', default=80, type=int)
parser.add_argument('--polarities_dim', default=3, type=int)
parser.add_argument('--hops', default=3, type=int)
parser.add_argument('--device', default=None, type=str, help='e.g. cuda:0')
parser.add_argument('--seed', default=None, type=int, help='set seed for reproducibility')
parser.add_argument('--valset_ratio', default=0, type=float, help='set ratio between 0 and 1 for validation support')
opt = parser.parse_args()
```

```python
class LocationEncoding(nn.Module):
    def __init__(self, opt):
        super(LocationEncoding, self).__init__()
        self.opt = opt

    def forward(self, x, pos_inx):
        batch_size, seq_len = x.size()[0], x.size()[1]
        weight = self.weight_matrix(pos_inx, batch_size, seq_len).to(self.opt.device)
        x = weight.unsqueeze(2) * x
        return x

    def weight_matrix(self, pos_inx, batch_size, seq_len):
        pos_inx = pos_inx.cpu().numpy()
        weight = [[] for i in range(batch_size)]
        for i in range(batch_size):
            for j in range(pos_inx[i][0]):
                relative_pos = pos_inx[i][0] - j
                aspect_len = pos_inx[i][1] - pos_inx[i][0] + 1
                sentence_len = seq_len - aspect_len
                weight[i].append(1 - relative_pos / sentence_len)
            for j in range(pos_inx[i][0], pos_inx[i][1] + 1):
                weight[i].append(0)
            for j in range(pos_inx[i][1] + 1, seq_len):
                relative_pos = j - pos_inx[i][1]
                aspect_len = pos_inx[i][1] - pos_inx[i][0] + 1
                sentence_len = seq_len - aspect_len
                weight[i].append(1 - relative_pos / sentence_len)
        weight = torch.tensor(weight)
        return weight
```

```python
class AlignmentMatrix(nn.Module):
    def __init__(self, opt):
        super(AlignmentMatrix, self).__init__()
        self.opt = opt
        self.w_u = nn.Parameter(torch.Tensor(6*opt.hidden_dim, 1))

    def forward(self, batch_size, ctx, asp):
        ctx_len = ctx.size(1)
        asp_len = asp.size(1)
        alignment_mat = torch.zeros(batch_size, ctx_len, asp_len).to(self.opt.device)
        ctx_chunks = ctx.chunk(ctx_len, dim=1)
        asp_chunks = asp.chunk(asp_len, dim=1)
        for i, ctx_chunk in enumerate(ctx_chunks):
            for j, asp_chunk in enumerate(asp_chunks):
                feat = torch.cat([ctx_chunk, asp_chunk, ctx_chunk*asp_chunk], dim=2) # batch_size x 1 x 6*hidden_dim
                alignment_mat[:, i, j] = feat.matmul(self.w_u.expand(batch_size, -1, -1)).squeeze(-1).squeeze(-1)
        return alignment_mat
```

```
class MGAN(nn.Module):
    def __init__(self, embedding_matrix, opt):  # embedding_matrix: (3600, 50)
        super(MGAN, self).__init__()
        self.opt = opt
        self.embed = nn.Embedding.from_pretrained(torch.tensor(embedding_matrix, dtype=torch.float), freeze=False)
        self.ctx_lstm = DynamicLSTM(opt.embed_dim, opt.hidden_dim, num_layers=1, batch_first=True, bidirectional=True)
        self.asp_lstm = DynamicLSTM(opt.embed_dim, opt.hidden_dim, num_layers=1, batch_first=True, bidirectional=True)
        self.location = LocationEncoding(opt)
        self.w_a2c = nn.Parameter(torch.Tensor(2*opt.hidden_dim, 2*opt.hidden_dim))
        self.w_c2a = nn.Parameter(torch.Tensor(2*opt.hidden_dim, 2*opt.hidden_dim))
        self.alignment = AlignmentMatrix(opt)
        self.dense = nn.Linear(8*opt.hidden_dim, opt.polarities_dim)

    def forward(self, inputs):
        text_raw_indices = inputs[0] # batch_size x seq_len
        aspect_indices = inputs[1]
        text_left_indices= inputs[2]
        batch_size = text_raw_indices.size(0)
        ctx_len = torch.sum(text_raw_indices != 0, dim=1)
        asp_len = torch.sum(aspect_indices != 0, dim=1)
        left_len = torch.sum(text_left_indices != 0, dim=-1)
        aspect_in_text = torch.cat([left_len.unsqueeze(-1), (left_len+asp_len-1).unsqueeze(-1)], dim=-1)
        # 下面相当于lookup过程
        ctx = self.embed(text_raw_indices) # batch_size x seq_len x embed_dim
        asp = self.embed(aspect_indices) # batch_size x seq_len x embed_dim

        ctx_out, (_, _) = self.ctx_lstm(ctx, ctx_len)
        ctx_out = self.location(ctx_out, aspect_in_text) # batch_size x (ctx)seq_len x 2*hidden_dim
        ctx_pool = torch.sum(ctx_out, dim=1)
        ctx_pool = torch.div(ctx_pool, ctx_len.float().unsqueeze(-1)).unsqueeze(-1) # batch_size x 2*hidden_dim x 1

        asp_out, (_, _) = self.asp_lstm(asp, asp_len) # batch_size x (asp)seq_len x 2*hidden_dim
        asp_pool = torch.sum(asp_out, dim=1)
        asp_pool = torch.div(asp_pool, asp_len.float().unsqueeze(-1)).unsqueeze(-1) # batch_size x 2*hidden_dim x 1

        alignment_mat = self.alignment(batch_size, ctx_out, asp_out) # batch_size x (ctx)seq_len x (asp)seq_len
        # batch_size x 2*hidden_dim  torch.Size([64, 600, 57]) torch.Size([64, 57, 1]) torch.Size([64, 600])
        f_asp2ctx = torch.matmul(ctx_out.transpose(1, 2), F.softmax(alignment_mat.max(2, keepdim=True)[0], dim=1)).squeeze(-1)
        f_ctx2asp = torch.matmul(F.softmax(alignment_mat.max(1, keepdim=True)[0], dim=2), asp_out).transpose(1, 2).squeeze(-1)

        c_asp2ctx_alpha = F.softmax(ctx_out.matmul(self.w_a2c.expand(batch_size, -1, -1)).matmul(asp_pool), dim=1)
        c_asp2ctx = torch.matmul(ctx_out.transpose(1, 2), c_asp2ctx_alpha).squeeze(-1)
        c_ctx2asp_alpha = F.softmax(asp_out.matmul(self.w_c2a.expand(batch_size, -1, -1)).matmul(ctx_pool), dim=1)
        c_ctx2asp = torch.matmul(asp_out.transpose(1, 2), c_ctx2asp_alpha).squeeze(-1)

        feat = torch.cat([c_asp2ctx, f_asp2ctx, f_ctx2asp, c_ctx2asp], dim=1)
        out = self.dense(feat) # bathc_size x polarity_dim

        return out
```