Artificial Neural Networks
人工神经网络

权小军教授
中山大学计算机学院

quanxj3@mail.sysu.edu.cn

2023 年 6 月 6 日

Final-term Project: Chinese-to-English Machine Translation

数据集简介

- 数据集说明：压缩包中共有 3 个文件夹，分别对应着训练集、评估集和测试集，它们的大小分别是 10000、444、444。每个文件夹中都含有 src data (中文) 和 target data (英文) 构成平行语料对。模型的性能以测试集的结果为最终标准。
- 数据下载地址：课程百度网盘

数据预处理–构建词表

- 数据清洗：非法字符，脏词过滤；过长过短句子的过滤。
- 分词：

```
from nltk.tokenize import word_tokenize as en_tokenizer  # english words seg
from transformers import BertTokenizer  # chinese words seg
ch_tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')

en_words = en_tokenizer("To be or not to be, it's a question.")
ch_words = ch_tokenizer.tokenize("自然语言处理是人工智能皇冠上的明珠。")
```

- 构建词典：

```
print('initialize source vocabulary ..')
src = VocabEntry.from_corpus(src_sents, vocab_size, freq_cutoff)

print('initialize target vocabulary ..')
tgt = VocabEntry.from_corpus(tgt_sents, vocab_size, freq_cutoff)

return Vocab(src, tgt)
```

数据加载

- 自定义加载

```
for src_sents, tgt_sents in batch_iter(train_data, batch_size=train_batch_size, shuffle=True):
    train_iter += 1
```

```
def to_input_tensor(self, sents: List[List[str]], device: torch.device, max_len: int=None) -> torch.Tensor:
    word_ids = self.words2indices(sents)
    sents_t = input_transpose(word_ids, self['<pad>'], max_len)
    sents_var = torch.tensor(sents_t, dtype=torch.long, device=device)
    return sents_var
```

- torch.utils.data.DataLoader (pytorch 封装好的 pipeline)
  - RandomSampler or SequentialSampler + Collator

```
train_dial_sampler = RandomSampler(train_dial_dataset)
train_dial_dataloader = DataLoader(
    train_dial_dataset,
    sampler=train_dial_sampler,
    batch_size=opt.per_gpu_batch_size,
    drop_last=True,
    num_workers=10,
    collate_fn=dial_collator
)
```

模型构建–单向双层 Lstm+LuongAttention+DotAlign

- 模型初始化

```
# initialize embedding layers..
self.src_embed = nn.Embedding(len(vocab.src), embed_size, padding_idx=vocab.src['<pad>'])
self.tgt_embed = nn.Embedding(len(vocab.tgt), embed_size, padding_idx=vocab.tgt['<pad>'])
# initialize encoder and decoder
self.encoder_lstm = nn.LSTM(embed_size, hidden_size, bidirectional=True,
                            num_layers=enoder_layer_num)
decoder_lstm_input = embed_size + hidden_size if self.input_feed else embed_size
self.decoder_lstm_input_layer = nn.LSTMCell(decoder_lstm_input, hidden_size)
self.decoder_lstm_hidden_layers = nn.ModuleList([nn.LSTMCell(hidden_size, hidden_size)
                                                 for i in range(decoder_layer_num-1)])
# initialize attention layer: dot product attention
# project source encoding to decoder rnn's state space
self.att_src_linear = nn.Linear(hidden_size * 2, hidden_size, bias=False)
# transformation of decoder hidden states and context vectors before reading out target words
# this produces the `attentional vector` in (Luong et al., 2015)
self.att_vec_linear = nn.Linear(hidden_size * 2 + hidden_size, hidden_size, bias=False)
# prediction layer of the target vocabulary
self.readout = nn.Linear(hidden_size, len(vocab.tgt), bias=False)

# dropout layer
```

模型构建–单向双层 Lstm+LuongAttention+DotAlign

- Multi-layer Lstm Decoder

```
def step(self, x: torch.Tensor,
         h_tm1: Tuple[torch.Tensor, torch.Tensor],
         src_encodings: torch.Tensor, src_encoding_att_linear: torch.Tensor,
         src_sent_masks: torch.Tensor):
    # h_t: (batch_size, hidden_size)
    h_tm1_new = []
    c_tm1_new = []
    h_t, cell_t = self.decoder_lstm_input_layer(x, (h_tm1[0][0], h_tm1[1][0]))
    h_tm1_new.append(h_t)
    c_tm1_new.append(cell_t)

    h_t, cell_t = self.dropout(h_t), self.dropout(cell_t)
    for i in range(self.decoder_layer_num-1):
        h_t, cell_t = self.decoder_lstm_hidden_layers[i](h_t, (h_tm1[0][i+1], h_tm1[1][i+1]))
        h_t, cell_t = self.dropout(h_t), self.dropout(cell_t)
        h_tm1_new.append(h_t)
        c_tm1_new.append(cell_t)

    h_tm1_new = torch.cat(h_tm1_new, 0)
    c_tm1_new = torch.cat(c_tm1_new, 0)
    h_tm1 = (h_tm1_new, c_tm1_new)
    ctx_t, alpha_t = self.dot_prod_attention(h_t, src_encodings,
                                             src_encoding_att_linear, src_sent_masks)
    att_t = torch.tanh(self.att_vec_linear(torch.cat([h_t, ctx_t], 1)))  # E.q. (5)
    att_t = self.dropout(att_t)

    return h_tm1, att_t, alpha_t
```

模型构建–单向双层 Lstm+LuongAttention+DotAlign

- LuongAttention+DotAlign

```
def dot_prod_attention(self, h_t: torch.Tensor,
                       src_encoding: torch.Tensor,
                       src_encoding_att_linear: torch.Tensor,
                       mask: torch.Tensor=None) -> \
        Tuple[torch.Tensor, torch.Tensor]:
    # (batch_size, src_sent_len)
    att_weight = torch.bmm(src_encoding_att_linear, h_t.unsqueeze(2)).squeeze(2)
    # dot product

    if mask is not None:
        att_weight.data.masked_fill_(mask.bool(), -float('inf'))  # encoder attention mask

    softmaxed_att_weight = F.softmax(att_weight, dim=-1)

    att_view = (att_weight.size(0), 1, att_weight.size(1))
    # (batch_size, hidden_size)
    ctx_vec = torch.bmm(softmaxed_att_weight.view(*att_view), src_encoding).squeeze(1)

    return ctx_vec, softmaxed_att_weight
```

模型训练

- 优化器：SGD、Adam
- batch size: 通常为 2 的幂
- 学习率：SGD 学习率较大，Adam 学习率较小；batch size 越大，学习率越大
- 防止过拟合：early stop (≤ 5)

评估指标

- 原则
  - 验证集调参，测试集评估
- BLEU (1-4)
  - nltk.translate.bleu_score

```
bleu_score = corpus_bleu([[ref] for ref in references],
                         [hyp.value for hyp in hypotheses])
```

Thank you!