Assignment 2-3

Files

config.py: Define configuration parameters.

data/

data_utils.py: Helper functions or classes used in data processing.

process.py: Process a raw dataset into a sample file and partition it into train, dev, test sets.

transform_images.py: Preprocess images.

model/

dataset.py: Define the format of samples used in the model.

evaluate.py: Evaluate the loss in the dev set.

model.py: Define the model.

predict.py: Generate a summary.

test_vocab.py: Testing vocab.

train.py: Train the model.

utils.py: Helper functions or classes used for the model.

vocal.py: Define the vocabulary object.

saved_model/

Save the trained model objects here.

files/

Place data files here.

runs/

Save logs here for TensorboardX.

TO-DO list:

必备资料

为了完成以下任务，我们需要逐步熟悉、掌握Pytorch框架，所以请大家在完成每一个模块时先查阅一下Pytorch的API文档，弄清楚要使用的模块是做什么的以及如何使用。此外，此次作业设计多模态PGN的实现，需要参见论文 Aspect-Aware Multimodal Summarization for Chinese E-Commerce Products。

此外，这次作业我们将使用有3w文本对+商品图片的京东智能营销文本数据集。

模块1: 图片预处理

为了引入图片信息，我们用ResNet将图片encode为向量，这部分我们可以先对图片进行批量预处理，将向量保存下来，后面读取数据时只需要根据图片名称读取预先生成的图片向量即可。

data/transform_images.py:

任务1: 完成这个文件。

将数据集中的所有图片转换为ResNet的输出向量，保存在文件中，每一行对应一张图片，每一行的格式为：图片名称\t图片向量，其中，图片向量的每一维用空格隔开。示例：`112Ehz23oz.jpg -0.82819766 -0.5631548 ... 0.8781789 0.8810012`

文件保存路径为files/，文件名为img_vecs.txt。

ResNet的使用可参考文档https://pytorch.org/docs/stable/torchvision/models.html?highlight=resnet101#torchvision.models.resnet101.

模块2: 数据预处理

先跑data/process.py得到训练文件。

model/dataset.py:

任务1: 完成SampleDataset类中的get_img_vecs函数。

这一函数的作用是从模块1生成的img_vecs.txt中读取图片对应的向量，并保存到字典中。

任务2: 完成SampleDataset类中的get_sample和build_samples函数。

这两个函数的作用是将数据集中的文本对和图片等信息读取到SampleDataset保存到一个字典sample(keys: 'src', 'tgt', 'cate', 'img_vec'，分别对应source, target, 类别和图片向量)中。其中source和target的处理方式与之前一致，区别在于：

1. 这次的source是已拼接过的。
2. 一个source可能对应多个target。

注意图片信息请利用get_img_vecs()转换为图片向量。

模块3: 完成多模态PGN

model/model.py:

任务1: 完成ReduceState。

实现论文"Aspect-Aware Multimodal Summarization for Chinese E-Commerce Products"中定义的MMPG+DecInit模型并对比三者效果，即使用图片向量来初始化decoder的初始状态。详见公式(11)(12)。

任务2 (optional):

尝试实现论文中提到的另外两种多模态实现方式：MMPG+EncInit和MMPG+MMAtt，局部图片特征的提取可参考以下代码。

```
def hierarchical_attention(self, h_t, src_encoding, src_encoding_att_linear,
img_fc, img_conv, img_conv_att_linear, dec_init_vec):
        """
        :param h_t: (batch_size, hidden_size)
        :param src_encoding: (batch_size, src_sent_len, hidden_size * 2)
        :param src_encoding_att_linear: (batch_size, src_sent_len,
hidden_size)
        :param img_fc: (batch_size, 4096)
        :param img_conv: (batch_size, 49, 512)
        """

        # (batch_size, 1, hidden_size) + (batch_size, src_sent_len,
hidden_size) =>
        # (batch_size, src_sent_len, hidden_size)
        att_hidden_text =
F.tanh(self.att_ht_linear(h_t).unsqueeze(1).expand_as(src_encoding_att_linear)
+ src_encoding_att_linear)

        # (batch_size, src_sent_len)
        att_weights_text = F.softmax(tensor_transform(self.att_v_linear,
att_hidden_text).squeeze(2))

        # (batch_size, hidden_size * 2)
        ctx_vec_text = torch.bmm(src_encoding.permute(0, 2, 1),
att_weights_text.unsqueeze(2)).squeeze(2)

        # (batch_size, 1, hidden_size) + (batch_size, patch_size, hidden_size)
=>
        # (batch_size, patch_size, hidden_size)
        att_hidden_img =
F.tanh(self.att_hv_linear(h_t).unsqueeze(1).expand_as(img_conv_att_linear) +
img_conv_att_linear)

        # (batch_size, patch_size)
        att_weights_img = F.softmax(tensor_transform(self.att_i_linear,
att_hidden_img).squeeze(2))
```

```
        # (batch_size, hidden_size)
        ctx_vec_img = torch.bmm(img_conv.permute(0, 2, 1),
att_weights_img.unsqueeze(2)).squeeze(2)


        beta_text = F.sigmoid(self.att_ua_linear(h_t) +
self.att_wa_linear(ctx_vec_text))
        beta_img = F.sigmoid(self.att_ub_linear(h_t) +
self.att_wb_linear(ctx_vec_img))
        beta_all = beta_text + beta_img
        beta_text = beta_text / beta_all
        beta_img = beta_img / beta_all

        img_fc_linear = self.att_wf_linear(img_fc)
        Ic = F.sigmoid(self.att_init_linear(dec_init_vec)  + img_fc_linear +
self.att_ws_linear(h_t))
        ctx_vec_img = Ic * ctx_vec_img
        ctx_vec = beta_text * self.ctx_vec_text_linear(ctx_vec_text) +
beta_img * self.ctx_vec_img_linear(ctx_vec_img)

        return ctx_vec
```