EANN 实验重构与解析

论文模型主要由 3 个模块组成——特征提取模块，分类检测模块，事件鉴别模块。

实验架构分为——数据载入，模型生成，训练、验证与测试

数据载入

数据载入是最关键也是最复杂的模块。以微博数据集为例，EANN 的数据集文件如下：

nonrumor_images：真实图片文件夹，图片名为索引；

rumor_images：虚假图片文件夹；

test_nonrumor.txt：真实新闻测试集，包括推特 id，图片名，推特内容；

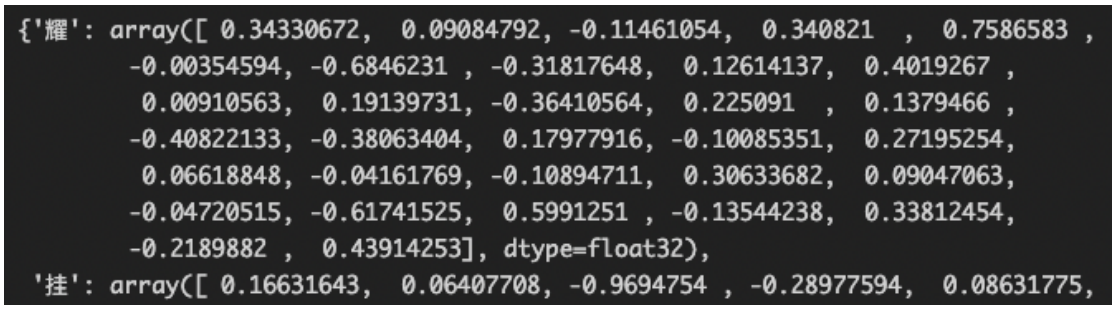
test_rumor.txt：虚假新闻测试集；

train_nonrumor.txt：真实新闻训练集；

train_rumor.txt：虚假新闻训练集；

stop_words.txt：停用词

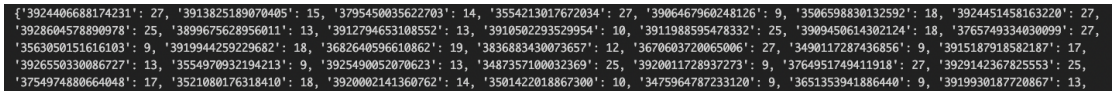
w2v.pickle：word2vec 预训练词向量字典



train_id.pickle：训练集中，以推特 id 为 key，事件种类为 value 的字典；

test_id.pickle：测试集中，以推特 id 为 key，事件种类为 value 的字典；

validate_id.pickle：验证集中，以推特 id 为 key，事件种类为 value 的字典；



1、 载入、划分、匹配、embedding

分别对两个文件夹中的图像进行 transform，从 JPG 图像格式转为 tensor。同时**构造一个 image_dict**，图片名称为 key，图像 tensor 为 value。

读入停用词，4 个 txt，pickle 事件字典。生成索引 post_id，image_id，label；对文本内容清洗、分词，生成索引 cleaned_post，seged_post；根据 pickle 事件字典按照推特 id 映射生成索引 event，基于 event 构造事件映射字典 map_id，索引 event 更新为 map_id 字典的 value，从效果上看更加有序。map_id 示例：

```
{10: 0, 5: 1, 6: 2, 7: 3, 8: 4, 11: 5}
```

利用上述索引和数据生成一个 dataframe，索引包括['post_id', 'image_id', 'cleaned_post', 'seged_post', 'label', 'event_label']。利用 post_id 作为关键值到 **image_dict** 中查找对应的首张图片，取出其 key 和 value 赋给新列表，将 dataframe 中除 image_id 的其他值取出赋给新列表，所有新列表一起生成一个字典，格式：{"post_text"， "original_post"， "image"， "social_feature"， "label"， "event_label"， "post_id"， "image_id"}。

前两段分别对训练集、验证集、测试集相同操作，生成的是三个数据子集的字典。

载入 w2v.pickle 预训练词向量字典，检查并加入陌生词向量。从 w2v.pickle 中取出词权重 W，和词索引字典 word_idx_map。将词权重 W，空字典 W2，词索引字典 word_idx_map，词频词典 vocab，最大长度 max_l 共同生成 word_embedding.pickle。

以上整体返回训练集、验证集、测试集数据子集的字典。

读入 word_embedding.pickle，复原其中的 5 个值：[W, W2, word_idx_map, vocab, max_len]。对上面返回的三个数据子集的字典推文部分 post_text 进行 word2vec 向量化。输入 post_text 和词索引字典 word_idx_map，对于一句话中每个词按照 word_idx_map 进行标记，空位补零，生成 word_embedding 以及 mask，mask 是将有值位置设为 1，空位补零。返回 word_embedding 和 mask。将 word_embedding 覆盖到 post_text 上，再给数据子集的字典生成一个新键 mask。

以上分别对训练集、验证集、测试集相同操作，生成 word_embedding 和 mask。

2、 变换

构造数据集的新 tensor 对象，将 post_text，mask，label，event_label 转换为 tensor，并作为属性 text，image，mask，label，event_label 赋给这个新对象自身。以上分别对训练集、验证集、测试集相同操作。

3、 Dataloader

使用 DataLoader 构造三个数据子集的可以迭代输入的新对象 train_loader，validate_loader，test_loader，设定 batch_size。

4、 生成模型

生成模型需要输入词权重 W。只关注 forward 部分，模型输入 text, image, mask。

图像部分，运用固定参数的预训练模型 VGG19，将最后一层改为 FC 加 leaky_relu 激活函数输出 32 维特征；

文本部分，利用 nn.Embedding 创建一个指定大小的词嵌入模型，将词权重字典 W 赋值到词嵌入模型的参数上，形状变为 W 的形状(4627, 32)；读入一个 batch 的文本数据的分词索引结果'post_text'，以及'mask'。从一个样本来说明，按照'post_text'中的值去索引词嵌入模型的词向量，生成 text_embeded，维度是 1*max_len*32，max_len 为 363，所以是 1*363*32；扩充 mask1*363 到与 text_embeded1*363*32 一样大小，两者相乘得到 text_masked 1*363*32。对 text_masked 需要扩充一个 channel 维度得到 1*1*363*32，随后进行 nn.Conv2d 的卷积操作得到 20*363*1，20*362*1, 20*361*1, 20*360*1 四组，设定 channel_out=20。接下来对每组进行最大池化，size(2)指每两个之间。将池化后的每组向量按行拼接，此时维度 4*20，输入 FC 层和 leaky_relu 激活，输出 32 维特征；将文本和图像的最终特征拼接。

真假分类部分，fc 层输出 2 维特征，加 softmax 输出 0/1 的二值概率分布；

域鉴别部分，手动写梯度反转层的前馈传播和反向传播，以此影响域鉴别前部分的梯度传播。

5、 main，训练、验证

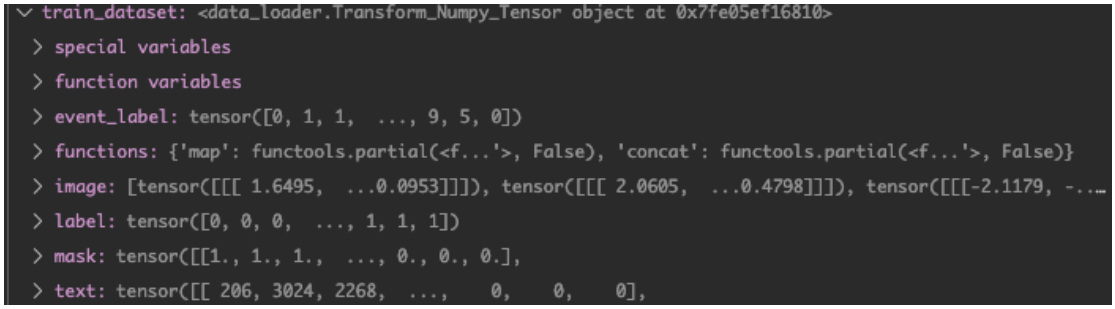
前面的准备工作做好，接下来是结合这些数据和模型进行训练和测试部分。通过 parse_arguments 函数首先定义所有 args 参数，方便全局修改和传入。使用规范如下：

```
parser = argparse.ArgumentParser()
parser = parse_arguments(parser)
args = parser.parse_args()
main(args)
```

首先创建 ArgumentParser()对象，如何构造 parse_arguments()方法来多次调用 add_argument()方法添加参数，最后使用 parse_args()解析添加的参数并存入 args 内全局调用。

调用 main 主函数，使用 load_data 方法得到划分好的训练集，验证集，测试集以及词权重 W，训练集，验证集，测试集数据格式如下，由 9 个键构成的字典，和之前的分析可以对应上。

使用 Transform_Numpy_Tensor 函数，将几个内嵌 list 都从 numpy 转为 tensor。返回(text[idx], image[idx], mask[idx]), label[idx], event_label[idx]五个值，这五个值全部都是 tensor 且一致对应一条推文数据。数据格式如下。



使用 Dataloader 封装以上数据得到 train_loader，test_loader，validate_loader，并确定 batch_size 和是否打乱。得到三个可迭代数据类，方便后续载入模型训练。将参数和词权重 W 传入模型，调用模型后将模型传入 gpu。定义损失函数、优化器。

调用训练函数，进入一个 epoch，衰减学习率并给到优化器。遍历 train_loader 一组 batch 的 5 个值，将一个 batch_size 的数据一起取出。

```
    for i, (train_data, train_labels, event_labels) in enumerate(train_loader):
        train_text = Transform_Tensor_Variable(train_data[0])
        train_image = Transform_Tensor_Variable(train_data[1])
        train_mask = Transform_Tensor_Variable(train_data[2])
        train_labels = Transform_Tensor_Variable(train_labels)
        event_labels = Transform_Tensor_Variable(event_labels)
```

```
> train_text.shape: torch.Size([20, 363])
> train_image.shape: torch.Size([20, 3, 224, 224])
> train_mask.shape: torch.Size([20, 363])
> train_labels.shape: torch.Size([20])
> event_labels.shape: torch.Size([20])
```

将优化器梯度清零，将 train_text, train_image, train_mask 三值输入模型，根据模型返回 class_outputs, domain_outputs，其中 class_outputs 是二分类（真/假），domain_outputs 是事件分类（10 类事件），class_outputs, domain_outputs 维度：

```
> class_outputs.shape: torch.Size([20, 2])
> domain_outputs.shape: torch.Size([20, 10])
```

利用 nn.CrossEntropyLoss()分别计算分类损失和鉴别损失，nn.CrossEntropyLoss()输入参数两个值：每一类的概率 weight 以及真实标签，因此可以反过来推断 class_outputs, domain_outputs 在模型中最后一层肯定是 softmax 函数。将两类损失相加成为总损失，再将这个损失反向传播，优化器开始优化。

利用 torch.max(input, dim)函数将最大值的位置信息和值一并取出，位置信息即对应类别。将一个 batch 的预测类别和真实训练标签进行对比，生成准确率 accuracy。

将以上得到的 class_loss，domain_loss，总 loss，accuracy 四个值 append 进一个数组。

以上过程在一个 epoch 内部进行循环，一个 epoch 训练完得到 batch 个数长度的四个数组，长度为 271。在一个 epoch 训练完后进行验证集测试，数据读入方式、计算模型输出、计算损失（验证只计算分类损失，不计算鉴别损失），计算验证准确率，将验证集准确率 append 进一个数组 validate_acc_vector_temp，长度为 43。

在一个 epoch 的验证集测试完后，得到该 epoch 下的验证集的平均准确率，将该准确率 append 进一个数组 valid_acc_vector，并按照 Epoch, Loss, Class Loss, domain loss, Train_Acc, Validate_Acc 打印该 epoch 训练和验证信息。

```
print('Epoch [%d/%d], Loss: %.4f, Class Loss: %.4f, domain loss: %.4f, Train_Acc: %.4f, Validate_Acc:
      % (  (variable) epoch: int
          epoch + 1, args.num_epochs,
          np.mean(cost_vector),
          np.mean(class_cost_vector),
          np.mean(domain_cost_vector),
          np.mean(acc_vector),
          validate_acc)
      )
```

保留最佳验证准确率，保存其模型参数，名字为当前 epoch。

6、测试

训练结束后，得到最佳模型参数，测试时初始化模型，载入最佳模型参数。调整 model.eval 模式，载入 test_loader，将一组 batch 的 test_text, test_image, test_mask 输入模型，输出 test_outputs 即分类结果，维度[20, 2]，torch.max 取出 test_argmax 信息，维度[20]。接下来将 test_outputs，test_argmax，test_labels 从 tensor 转为三个数组 test_score，test_pred，test_true，维度保持不变。从第二个 batch 开始将三个值直接 concat 进原来的数组，维度不变，增加数量。最终几个数组维度为：

```
> test_score.shape: (1465, 2)
> test_pred.shape: (1465,)
> test_true.shape: (1465,)
```

第一维度即测试集样本数量。得到测试数据后，计算指标。分别有 test_f1，test_precision，test_recall，test_score_convert，test_aucroc，test_confusion_matrix。分析以上 6 个指标：

test_f1 = metrics.f1_score(test_true, test_pred, average='macro')利用所有真实值和预测值计算 F1-score。用来衡量二分类（或多任务二分类）模型精确度的一种指标。它同时兼顾了分类模型的准确率和召回率，可以看作是模型准确率和召回率的一种加权平均，数值越大越好。

test_precision = metrics.precision_score(test_true, test_pred, average='macro') 利用所有真实值和预测值计算 precision

test_recall = metrics.recall_score(test_true, test_pred, average='macro') 利用所有真实值和预测值计算 recall

以上两个 precision 和 recall 可以用 F1-score 直接表示，实验中用 metrics.classification_report(test_true, test_pred)可以直接得到以上结果。课题内论文一般指采用 Acc，Pre，Recall，F1 作为指标，其中 F1 是最关键的。

训练截图：

```
Epoch [78/100], Loss: 1.9364, Class Loss: 0.3257, domain loss: 1.6107, Train_Acc: 0.9872,  Validate_Acc: 0.7295.
Epoch [79/100], Loss: 1.9285, Class Loss: 0.3239, domain loss: 1.6046, Train_Acc: 0.9896,  Validate_Acc: 0.7260.
Epoch [80/100], Loss: 1.9391, Class Loss: 0.3249, domain loss: 1.6142, Train_Acc: 0.9886,  Validate_Acc: 0.7120.
Epoch [81/100], Loss: 1.9306, Class Loss: 0.3250, domain loss: 1.6056, Train_Acc: 0.9884,  Validate_Acc: 0.7097.
Epoch [82/100], Loss: 1.9465, Class Loss: 0.3336, domain loss: 1.6129, Train_Acc: 0.9790,  Validate_Acc: 0.7097.
Epoch [83/100], Loss: 1.9353, Class Loss: 0.3259, domain loss: 1.6094, Train_Acc: 0.9873,  Validate_Acc: 0.6981.
Epoch [84/100], Loss: 1.9352, Class Loss: 0.3261, domain loss: 1.6091, Train_Acc: 0.9871,  Validate_Acc: 0.6992.
Epoch [85/100], Loss: 1.9358, Class Loss: 0.3248, domain loss: 1.6110, Train_Acc: 0.9882,  Validate_Acc: 0.6911.
Epoch [86/100], Loss: 1.9389, Class Loss: 0.3250, domain loss: 1.6139, Train_Acc: 0.9884,  Validate_Acc: 0.6992.
Epoch [87/100], Loss: 1.9310, Class Loss: 0.3250, domain loss: 1.6060, Train_Acc: 0.9880,  Validate_Acc: 0.6946.
Epoch [88/100], Loss: 1.9371, Class Loss: 0.3262, domain loss: 1.6109, Train_Acc: 0.9869,  Validate_Acc: 0.7062.
Epoch [89/100], Loss: 1.9277, Class Loss: 0.3244, domain loss: 1.6033, Train_Acc: 0.9891,  Validate_Acc: 0.7027.
Epoch [90/100], Loss: 1.9412, Class Loss: 0.3251, domain loss: 1.6162, Train_Acc: 0.9880,  Validate_Acc: 0.6888.
Epoch [91/100], Loss: 1.9440, Class Loss: 0.3258, domain loss: 1.6182, Train_Acc: 0.9873,  Validate_Acc: 0.6888.
Epoch [92/100], Loss: 1.9268, Class Loss: 0.3238, domain loss: 1.6030, Train_Acc: 0.9895,  Validate_Acc: 0.6957.
Epoch [93/100], Loss: 1.9351, Class Loss: 0.3238, domain loss: 1.6113, Train_Acc: 0.9895,  Validate_Acc: 0.6946.
Epoch [94/100], Loss: 1.9400, Class Loss: 0.3260, domain loss: 1.6140, Train_Acc: 0.9872,  Validate_Acc: 0.6934.
Epoch [95/100], Loss: 1.9327, Class Loss: 0.3283, domain loss: 1.6044, Train_Acc: 0.9845,  Validate_Acc: 0.7085.
Epoch [96/100], Loss: 1.9307, Class Loss: 0.3254, domain loss: 1.6053, Train_Acc: 0.9878,  Validate_Acc: 0.7004.
Epoch [97/100], Loss: 1.9374, Class Loss: 0.3250, domain loss: 1.6125, Train_Acc: 0.9884,  Validate_Acc: 0.6957.
Epoch [98/100], Loss: 1.9306, Class Loss: 0.3232, domain loss: 1.6074, Train_Acc: 0.9902,  Validate_Acc: 0.7027.
Epoch [99/100], Loss: 1.9297, Class Loss: 0.3242, domain loss: 1.6055, Train_Acc: 0.9891,  Validate_Acc: 0.7062.
Epoch [100/100], Loss: 1.9439, Class Loss: 0.3276, domain loss: 1.6163, Train_Acc: 0.9851,  Validate_Acc: 0.7085.
```

测试截图：

```
---------------------------------------------- 开始训练 ----------------------------------------------
testing model
Classification Acc: 0.7863, AUC-ROC: 0.8793
Classification report:
              precision    recall  f1-score   support

           0       0.80      0.75      0.77       709
           1       0.78      0.82      0.80       756

    accuracy                           0.79      1465
   macro avg       0.79      0.79      0.79      1465
weighted avg       0.79      0.79      0.79      1465


Classification confusion matrix:
[[531 178]
 [135 621]]
```