Web Lab2

PB1711585 张永停

一、实验要求

- 对于文档进行适当的预处理
- 选取合适的模型对文本进行建模，并在训练集上进行关系抽取模型训练。
- 在在线平台提交结果验证关系抽取模型的准确率。

二、算法

(一)数据处理

- 从数据集中提出句子以及关系，并将关系转为数字编码

```
    pattern_text = re.compile(r"\".*\"")
    pattern_rel = re.compile(r".*\(")
    pattern_ent1 = re.compile(r"\(.*\,")
    pattern_ent1 = re.compile(r"\,.*\)")
    info = []
    with open(train_data_path, "r") as load_f:
        info = []
        single_data = {}
        i = 1
        for line in load_f.readlines():
            if i % 2 ==1:
                single_data={}
                single_data['text'] = pattern_text.findall(line)[0][1:-1]
            else:
                single_data['rel'] = pattern_rel.findall(line)[0][:-1]
                single_data['ent1'] = pattern_ent1.findall(line)[0][1:-1]
                single_data['ent2'] = pattern_ent1.findall(line)[0][1:-1]
            i += 1
            info.append(single_data)
```

- 数据划分，从`train.txt`中按7：2.4：0.6划分train, test, develop
- 训练时数据准备以及mask选择
  - 使用bert的`tokenizer`进行`encode`
  - 同时判断长度进行`padding`

```
def load_train(data_path):
    rel2id,id2rel = map_id_rel()
    max_length=128
    tokenizer = BertTokenizer.from_pretrained(r'../../../data/Sqrti/bert-uncase/bert-base-uncased')
    train_data = {}
    train_data['label'] = []
```

```python
    train_data['mask'] = []
    train_data['text'] = []

    with open(data_path, 'r') as load_f:
        for line in load_f.readlines():
            dic = json.loads(line)
            if dic['rel'] not in rel2id:
                train_data['label'].append(0)
            else:
                train_data['label'].append(rel2id[dic['rel']])
            #sent=dic['ent1']+dic['ent2']+dic['text']
            sent = dic['text']
            indexed_tokens = tokenizer.encode(sent, add_special_tokens=True)
            avai_len = len(indexed_tokens)
            while len(indexed_tokens) <  max_length:
                indexed_tokens.append(0)  # 0 is id for [PAD]
            indexed_tokens = indexed_tokens[: max_length]
            indexed_tokens = torch.tensor(indexed_tokens).long().unsqueeze(0)  # (1, L)

            # Attention mask
            att_mask = torch.zeros(indexed_tokens.size()).long()  # (1, L)
            att_mask[0, :avai_len] = 1
            train_data['text'].append(indexed_tokens)
            train_data['mask'].append(att_mask)
    return train_data
```

(二)bert

- bert模型是一种模型迁移，将预训练模型和下游任务模型结合在一起，核心目的就是：是把下游具体NLP任务的活逐渐移到预训练产生词向量上。BERT真真意义上同时考虑了上下文。
- 用bert搭建一个实体抽取的模型, 使用Bert用于序列分类的(BertEncoder+Fc+CrossEntropy)模型`bert-base-uncased`, 加上MASK-Attention一起送进模型

```python
def train(net,dataset,num_epochs, learning_rate,  batch_size):
    net.train()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate,weight_decay=0)
    train_iter = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
    pre=0

    for epoch in range(num_epochs):
        correct = 0
        total=0
        iter = 0
        for text,mask, y in train_iter:
            iter += 1
            optimizer.zero_grad()
            if text.size(0)!=batch_size:
                break
            text=text.reshape(batch_size,-1)
            mask = mask.reshape(batch_size, -1)
            if USE_CUDA:
```

```python
                text=text.cuda()
                mask=mask.cuda()
                y = y.cuda()
            output = net(text, attention_mask=mask,labels=y)
            loss = output['loss']
            logits = output['logits']
            loss.backward()
            optimizer.step()
            _, predicted = torch.max(logits.data, 1)

            total += text.size(0)
            correct += predicted.data.eq(y.data).cpu().sum()
        loss=loss.detach().cpu()
        print("epoch ", str(epoch)," loss: ", loss.mean().numpy().tolist(),"right", correct.cpu().numpy().tolist(), "total", total, "Acc:", correct.cpu().numpy().tolist()/total)
        acc = eval(model, dev_dataset, 32)
        if acc > pre:
            pre = acc
            torch.save(model, "../model/" + str(acc)+'.pth')
    return
```

- 约20个epoch收敛

```
epoch  17  loss:  0.04723089933395386 right 11494 total 11648 Acc: 0.9867788461538461
Eval Result: right 707 total 768 Acc: 0.9205729166666666
epoch  18  loss:  0.052388548851013184 right 11547 total 11648 Acc: 0.9913289835164835
Eval Result: right 711 total 768 Acc: 0.92578125
epoch  19  loss:  0.02501794695854187 right 11571 total 11648 Acc: 0.9933894230769231
Eval Result: right 711 total 768 Acc: 0.92578125
epoch  20  loss:  0.028522953391075134 right 11599 total 11648 Acc: 0.9957932692307693
Eval Result: right 710 total 768 Acc: 0.9244791666666666
epoch  21  loss:  0.16468557715415955 right 11602 total 11648 Acc: 0.9960508241758241
Eval Result: right 713 total 768 Acc: 0.9283854166666666
epoch  22  loss:  0.02653355896472931 right 11607 total 11648 Acc: 0.9964800824175825
Eval Result: right 712 total 768 Acc: 0.9270833333333334
```

(三)RNN注意力模型

- 模型定义

```python
def RNN_model(input_layer, num_class):  # RNN model
    def smoothing_attention(x):
        e = K.sigmoid(x)
        s = K.sum(e, axis=-1, keepdims=True)
        return e / s
    reg = 0.0001
    dropout = 0.5
    hidden_dim = 1024
    vector = Bidirectional(CuDNNGRU(hidden_dim, return_sequences=False, kernel_regularizer=keras.regularizers.l2(reg)))(input_layer)
    lstm = Bidirectional(CuDNNGRU(hidden_dim, return_sequences=True, kernel_regularizer=keras.regularizers.l2(reg)))(input_layer)
    ee = Dot(axes=-1, normalize=True)([vector, lstm])
    weights = Lambda(smoothing_attention)(ee)
    weights = RepeatVector(2*hidden_dim)(weights)
    weights = Permute((2, 1))(weights)
    output = Multiply()([weights, lstm])
```

```python
    output = Lambda(lambda x: K.sum(x, axis=1))(output)
    output = Dense(512)(output)
    output = BatchNormalization()(output)
    output = Activation("relu")(output)
    output = Dense(256)(output)
    output = BatchNormalization()(output)
    output = Activation("relu")(output)
    output = Dropout(dropout)(output)
    output = Dense(num_class, activation='softmax')(output)
    model = Model(sequence_input, output)
    print(model.summary())
    return model
```

```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 100)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 100, 300)     5868900     input_1[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 2048)         8146944     embedding_1[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 100, 2048)    8146944     embedding_1[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                     (None, 100)          0           bidirectional_1[0][0]
                                                                 bidirectional_2[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 100)          0           dot_1[0][0]
__________________________________________________________________________________________________
repeat_vector_1 (RepeatVector)  (None, 2048, 100)    0           lambda_1[0][0]
__________________________________________________________________________________________________
permute_1 (Permute)             (None, 100, 2048)    0           repeat_vector_1[0][0]
__________________________________________________________________________________________________
multiply_1 (Multiply)           (None, 100, 2048)    0           permute_1[0][0]
                                                                 bidirectional_2[0][0]
__________________________________________________________________________________________________
lambda_2 (Lambda)               (None, 2048)         0           multiply_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          1049088     lambda_2[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 512)          2048        dense_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 512)          0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          131328      activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 256)          1024        dense_2[0][0]
__________________________________________________________________________________________________
```

- 训练
  - 使用斯坦福Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB)作为词向量预训练模型
  - 仿照bert的形式，对实体前后做标注`xxxxxxxxxe1xxxxxxxxx`与`sssssssse1ssssssss`

```python
    embeddings_index = {}
    with open(os.path.join(GLOVE_DIR, 'glove.42B.300d.txt'), encoding='utf-8') as f:  # read pre-trained word embedding
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    X_train = []
    Y_train = []
    label_to_y = dict()
    with open(os.path.join(GLOVE_DIR,"train.txt")) as f:
        for idx, l in enumerate(f):
            l = l.strip()
            if idx % 4 == 0:
```

```
                ID, sentence = l.split("\t")
                sentence = sentence[1:-1]
                sentence = sentence.replace('<e1>', 'xxxxxxxxxe1xxxxxxxxx ')
                sentence = sentence.replace('<e2>', 'xxxxxxxxxe2xxxxxxxxx ')
                sentence = sentence.replace('</e1>', ' ssssssssse1sssssssss')
                sentence = sentence.replace('</e2>', ' ssssssssse2sssssssss')
                X_train.append(sentence)
            elif idx % 4 == 1:
                label = l
                if label not in label_to_y:
                    label_to_y[label] = len(label_to_y)
                Y_train.append(label_to_y[label])
            else:
                pass
    y_to_label = {j:i for i, j in label_to_y.items()}
    Y_train = np.array(Y_train, dtype=int)
    num_class = max(Y_train) + 1
    Y_train = to_categorical(Y_train)
    tokenizer = Tokenizer(oov_token="UNK")
    tokenizer.fit_on_texts(X_train)
    sequences = tokenizer.texts_to_sequences(X_train)
    X_train = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
    if VALIDATION_SPLIT > 0:  # generate validation set
        indices = np.arange(len(Y_train))
        np.random.shuffle(indices)
        val_index = int(VALIDATION_SPLIT * len(Y_train))
        X_val = X_train[indices[:val_index]]
        Y_val = Y_train[indices[:val_index]]
        X_train = X_train[indices[val_index:]]
        Y_train = Y_train[indices[val_index:]]
    X_test = []
    ID_test = []
    with open(os.path.join(GLOVE_DIR,"test.txt")) as f:
        for l in f:
            ID, sentence = l.strip().split("\t")
            sentence = sentence[1:-1]
            sentence = sentence.replace('<e1>', 'xxxxxxxxxe1xxxxxxxxx ')
            sentence = sentence.replace('<e2>', 'xxxxxxxxxe2xxxxxxxxx ')
            sentence = sentence.replace('</e1>', ' ssssssssse1sssssssss')
            sentence = sentence.replace('</e2>', ' ssssssssse2sssssssss')
            ID_test.append(ID)
            X_test.append(sentence)
    sequences = tokenizer.texts_to_sequences(X_test)
    X_test = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
    word_index = tokenizer.word_index  # word dictionary <word, index>
    num_words = len(word_index) + 1
    embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))  # create word embedding matrix
    for word, i in word_index.items():
        embedding_vector = embeddings_index.get(word)
```

```
            if embedding_vector is not None:
                embedding_matrix[i] = embedding_vector
        embedding_layer = Embedding(num_words,  # inital word embedding weights
                                    EMBEDDING_DIM,
                                    weights=[embedding_matrix],
                                    input_length=MAX_SEQUENCE_LENGTH,
                                    trainable=True)
        sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH, ), dtype='int32')  # input layer
        embedded_sequences = embedding_layer(sequence_input)  # word embedding
        model = RNN_model(embedded_sequences, num_class)
        model.compile(loss='categorical_crossentropy',
                      optimizer=keras.optimizers.adam(lr= 0.001, amsgrad=True, clipvalue=15),
                      metrics=['accuracy'])
        early_stop = EarlyStopping(monitor='val_loss' if VALIDATION_SPLIT > 0 else "loss", patience=15, mode='min')
        model.fit(X_train, Y_train,
                  batch_size=128,
                  epochs=50,
                  callbacks=[early_stop],
                  validation_data=(X_val, Y_val) if VALIDATION_SPLIT > 0 else None)
        Y_pre = model.predict(X_test)
        Y_pre = np.argmax(Y_pre, axis=1)
        Y_pre = [y_to_label[i] for i in Y_pre]
        with open("predict.txt", 'w') as f:
            for ID, label in zip(ID_test, Y_pre):
                f.write(ID + "\t" + label + "\n")

```

- 该方法收敛极快，约10个epoch就收敛了

```
Epoch 3/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.8462 - acc: 0.7564 - val_loss: 1.1482 - val_acc: 0.6630
Epoch 4/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.5611 - acc: 0.8450 - val_loss: 1.9637 - val_acc: 0.5104
Epoch 5/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.3579 - acc: 0.9193 - val_loss: 1.0919 - val_acc: 0.6861
Epoch 6/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.2266 - acc: 0.9576 - val_loss: 1.2955 - val_acc: 0.6740
Epoch 7/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.1909 - acc: 0.9687 - val_loss: 1.1721 - val_acc: 0.7135
Epoch 8/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.1329 - acc: 0.9842 - val_loss: 1.1935 - val_acc: 0.7179
Epoch 9/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.1004 - acc: 0.9918 - val_loss: 1.2665 - val_acc: 0.7256
Epoch 10/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.0794 - acc: 0.9959 - val_loss: 1.3510 - val_acc: 0.7212
Epoch 11/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.1161 - acc: 0.9851 - val_loss: 1.3609 - val_acc: 0.6959
Epoch 12/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.0759 - acc: 0.9965 - val_loss: 1.2854 - val_acc: 0.7420
Epoch 13/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.0670 - acc: 0.9974 - val_loss: 1.3495 - val_acc: 0.7014
Epoch 14/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.0571 - acc: 0.9989 - val_loss: 1.2409 - val_acc: 0.7398
Epoch 15/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.0609 - acc: 0.9973 - val_loss: 1.3256 - val_acc: 0.7387
Epoch 16/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.1222 - acc: 0.9816 - val_loss: 1.8178 - val_acc: 0.6103
Epoch 17/50
8206/8206 [==============================] - 15s 2ms/step - loss: 0.0939 - acc: 0.9896 - val_loss: 1.6436 - val_acc: 0.6926
Epoch 18/50
```

三、优化

(一)bert

- 优化器如`sgd`，`adam`，`lr_scheduler`等的使用

- 发现使用adam无法收敛，初步怀疑是自己的参数设置的不好
- `lr_scheduler`可以有效的减少`sgd`的反向跳动

- 最终模型准确率在训练集可以达到99%， 在验证集可以达到90%， 但在在线系统提交后(已经使用优化器)却只有58%， 初步猜测数据分布不均衡造成过拟合。

  经过统计，发现训练集合类别分布不均匀，`other`类别数目过多

  ```
  Distribution:  [1890. 1355. 1415. 1344.  984. 1057. 1088. 1050.  798.  696.]
  ```

  这同时可以从模型的预测发现，模型更倾向于预测0类别

  ```
  tensor([1, 6, 0, 3, 1, 6, 8, 2, 1, 8, 4, 0, 0, 0, 2, 2, 2, 3, 7, 4, 4, 1, 4, 6,
          3, 3, 7, 4, 4, 9, 7, 1], device='cuda:0')
  tensor([2, 0, 0, 0, 0, 6, 2, 2, 0, 2, 2, 0, 0, 0, 6, 2, 0, 0, 0, 2, 0, 2, 0, 2,
          0, 0, 0, 0, 0, 2, 2, 2], device='cuda:0')
  ```

  上一行显示为label, 下一行为预测值

  - 对交叉熵按分布加权

    ```
    ce_loss = F.cross_entropy(output, target, weight=self.weight, reduction='none')
    ```

    其中`weight`第一次取值为`[1, 1.39824732, 1.3019039, 1.36501901 ,1.90703851, 1.79724656,1.67953216 ,1.82002535 ,2.36963696 ,2.65925926]`

    经过这样处理后模型的预测就比较均匀，最后正确率为`52%`(我试了好几种分布，均没有原来的高QWQ)

  - 使用**Focal Loss**

    Focal Loss 是用于解决`hard sample`的一种loss, 它也是对`loss`加权，按照`pred`之间的比例来进行加权

    ```
    class FocalLoss(nn.Module):
        def __init__(self, gamma=0, weight=None, reduction='mean'):
            super(FocalLoss, self).__init__()
            self.gamma = gamma
            self.weight = weight
            self.reduction = reduction

        def forward(self, output, target):
            # convert output to pseudo probability
            out_target = torch.stack([output[i, t] for i, t in enumerate(target)])
            probs = torch.sigmoid(out_target)
            focal_weight = torch.pow(1 - probs, self.gamma)

            # add focal weight to cross entropy
            ce_loss = F.cross_entropy(output, target, weight=self.weight, reduction='none')
            focal_loss = focal_weight * ce_loss

            if self.reduction == 'mean':
                focal_loss = (focal_loss / focal_weight.sum()).sum()
            elif self.reduction == 'sum':
                focal_loss = focal_loss.sum()

            return focal_loss
    ```

使用Focal Loss的训练部分仅需要更改loss获得形式即可

```
loss_fn = FocalLoss()
loss_fn  = FocalLoss()
loss = loss_fn(log, y)
loss.backward()
```

(二)RNN模型

该部分见算法Part 3

四、结果

ACC

(一)bert

- bert+att： 0.58
- bert+att+weighted loss: 0.52
- bert+att+focal loss: 0.59
- bert+att+focal: 0.61

Filename	ACC-Relation	ACC-NER
张永停-PB17111585-7-1.txt	0.613125	0.0

(二)RNN+ATT

- 0.62

Filename	ACC-Relation	ACC-NER
张永停-PB17111585-10.txt	0.6225	0.0

无论哪种方法，均在测试集表现不佳，模型过拟合严重。

当使用bert时， 验证集准确率可以达到0.9, 但RNN+ATT只有70%附近。私以为， bert的学习能力要比RNN强(当然这个几乎公认了)。

其中bert在验证集的表现最令我疑惑，为什么验证集这么高但测试集却只有0.6?