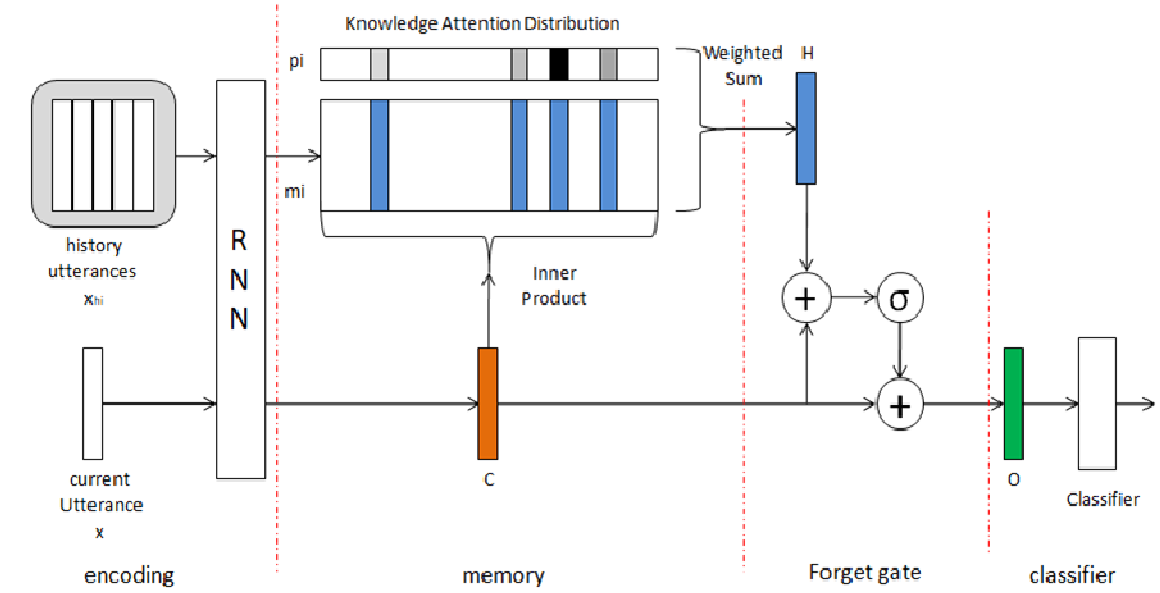
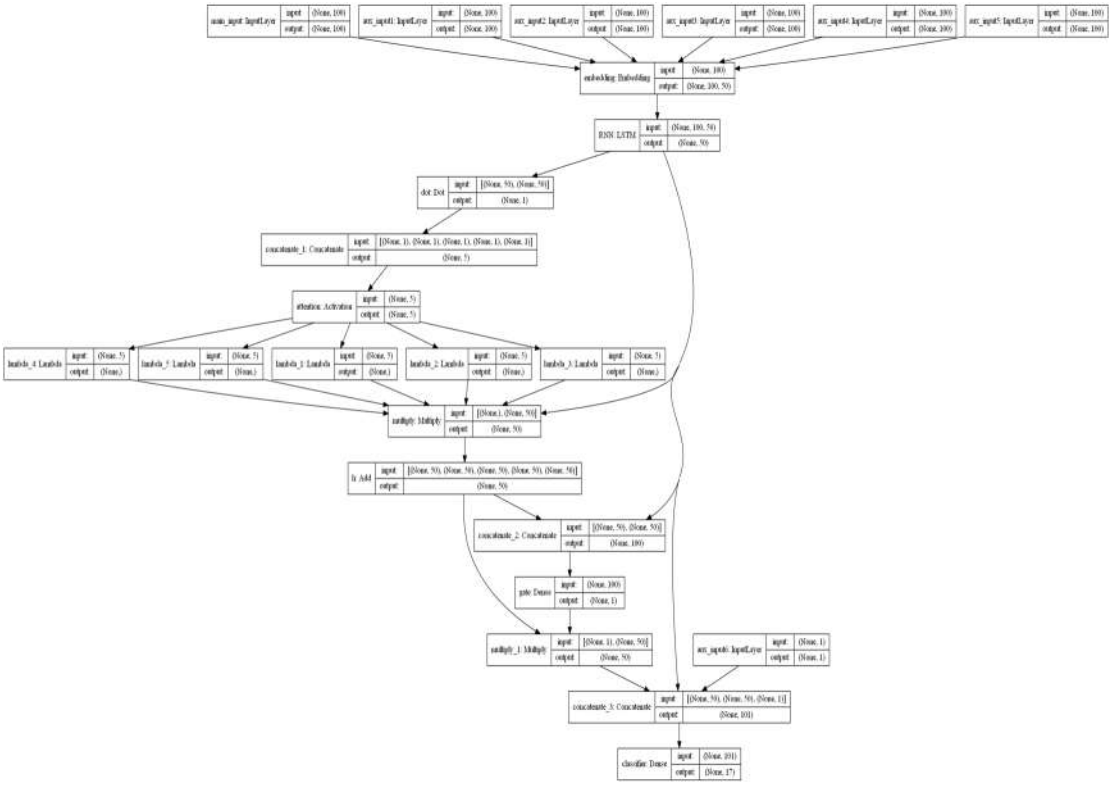
周报

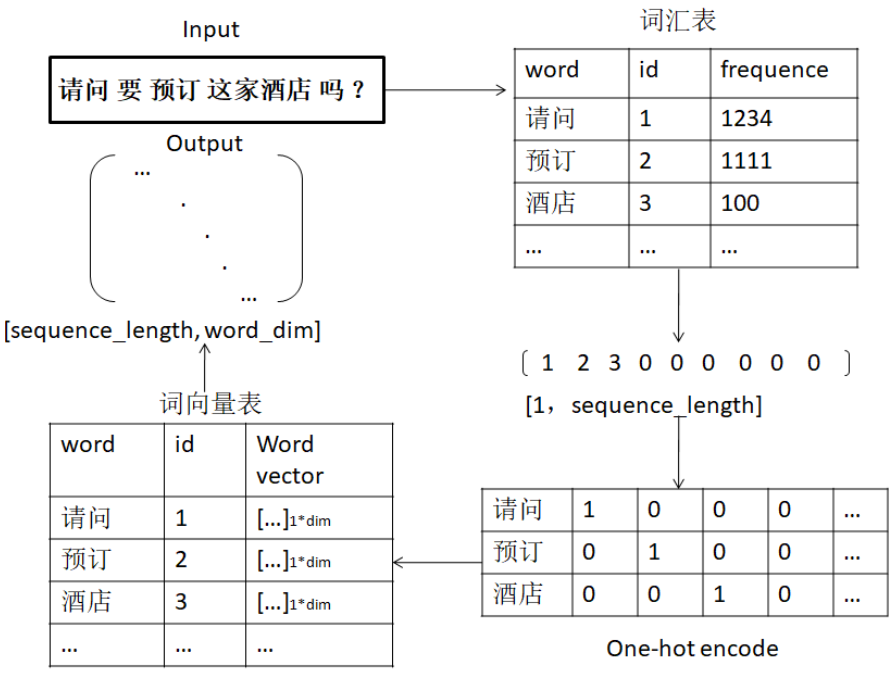
模型图：



整个模型流程图：



编码模块：



代码实现：

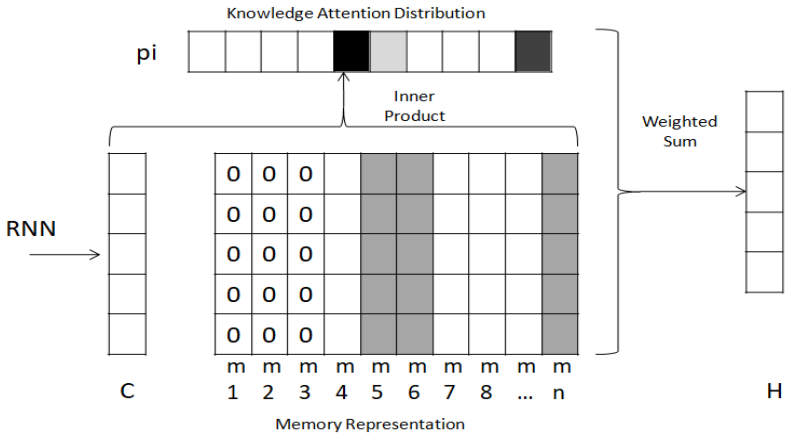
```
main_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32', name='main_input')
embedding_l = embedding_layer(main_input)
rnn_l = rnn_layer(embedding_l)

aux_input1 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32', name='aux_input1')
embedding_r1 = embedding_layer(aux_input1)
rnn_r1 = rnn_layer(embedding_r1)

aux_input2 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32', name='aux_input2')
embedding_r2 = embedding_layer(aux_input2)
rnn_r2 = rnn_layer(embedding_r2)

aux_input3 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32', name='aux_input3')
embedding_r3 = embedding_layer(aux_input3)
rnn_r3 = rnn_layer(embedding_r3)

aux_input4 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32', name='aux_input4')
embedding_r4 = embedding_layer(aux_input4)
rnn_r4 = rnn_layer(embedding_r4)

aux_input5 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32', name='aux_input5')
embedding_r5 = embedding_layer(aux_input5)
rnn_r5 = rnn_layer(embedding_r5)

aux_input6 = Input(shape=(1,), dtype='float32', name='aux_input6')
```

输入的文本处理为（NONE,MAX_SEQUENCE_LENGTH）的 2D 张量，当前轮次和历史轮次均进行 embedding_layer 和 rnn_layer 的处理。

```
embedding_layer = Embedding(output_dim=EMBEDDING_DIM,
                            input_dim=MAX_NB_WORDS + 1,
                            input_length=MAX_SEQUENCE_LENGTH,
                            weights=[embedding_matrix],
                            trainable=False,
                            name='embedding')
```

```
rnn_layer = LSTM(EMBEDDING_DIM,
                 activation='relu',
                 dropout=0.2,
                 recurrent_dropout=0.2,
                 return_sequences=False,
                 name='RNN')
```

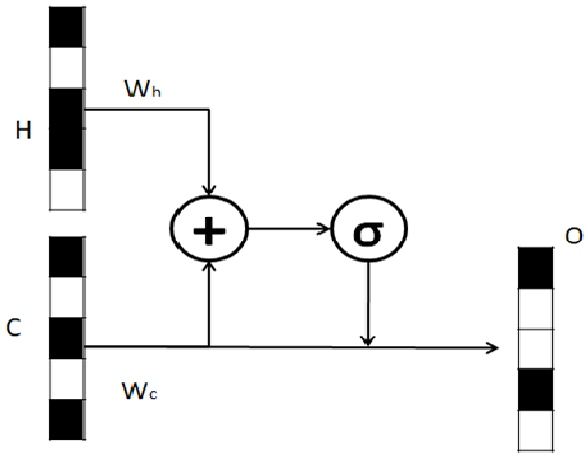
记忆模块：



代码实现：

```
dot_layer = Dot(axes=1, name='dot')
p1 = dot_layer([rnn_l, rnn_r1])
p2 = dot_layer([rnn_l, rnn_r2])
p3 = dot_layer([rnn_l, rnn_r3])
p4 = dot_layer([rnn_l, rnn_r4])
p5 = dot_layer([rnn_l, rnn_r5])

p = Concatenate(axis=1)([p1, p2, p3, p4, p5])
p = Activation(activation='softmax', name='attention')(p)
p1 = Lambda((lambda x: x[:, 0]))(p)
p2 = Lambda((lambda x: x[:, 1]))(p)
p3 = Lambda((lambda x: x[:, 2]))(p)
p4 = Lambda((lambda x: x[:, 3]))(p)
p5 = Lambda((lambda x: x[:, 4]))(p)

mul_layer = Multiply(name='multiply')

h1 = mul_layer([p1, rnn_r1])
h2 = mul_layer([p2, rnn_r2])
h3 = mul_layer([p3, rnn_r3])
h4 = mul_layer([p4, rnn_r4])
h5 = mul_layer([p5, rnn_r5])

h = Add(name='h')([h1, h2, h3, h4, h5])
```

控制门：



代码实现：

```
h = Add(name='h')([h1, h2, h3, h4, h5])

c = Concatenate(axis=1)([h, rnn_l])
a = Dense(1, activation='sigmoid', name='gate')(c)
```

分类器模块：

```
hh = Multiply()([a, h])
added = Concatenate(axis=1)([hh, rnn_l, aux_input6])
output2 = Dense(MAX_NB_LABELS, activation='sigmoid', name='classifier')(added)
```

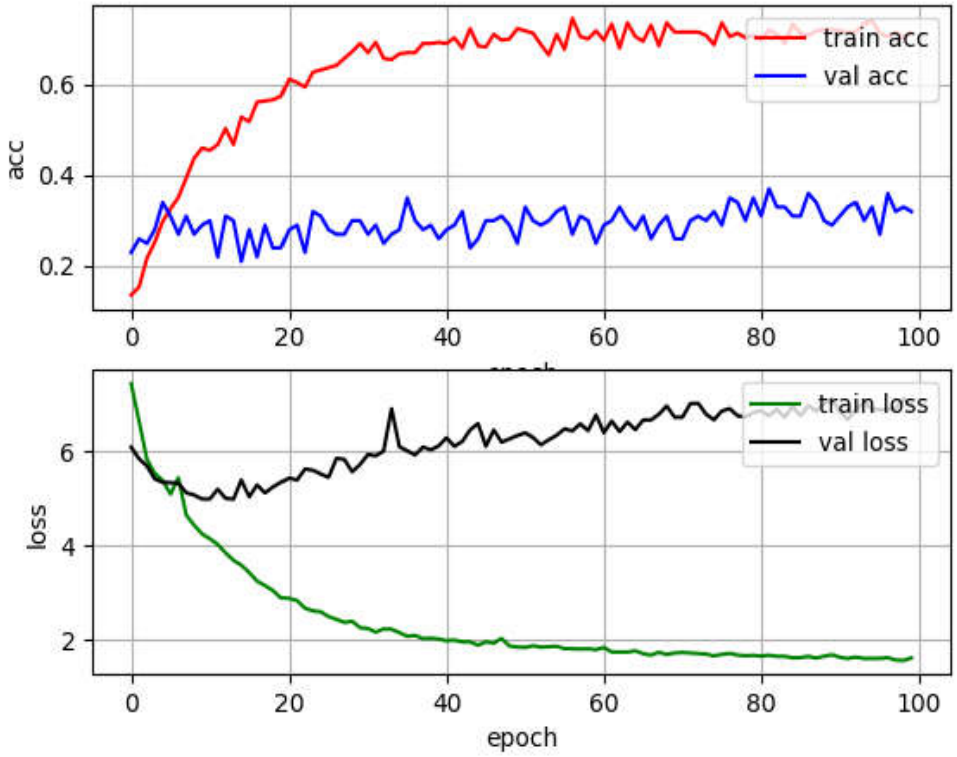
这里的激活函数为 sigmoid 函数，这个是为了配合后面的目标函数。

```
model = Model(inputs=[main_input,
                      aux_input1,
                      aux_input2,
                      aux_input3,
                      aux_input4,
                      aux_input5,
                      aux_input6
                      ], outputs=[output2])
```

上周模型训练采用的目标函数，优化函数，性能评估是配置如下：

```
model.compile(loss='categorical_crossentropy',
              optimizer=rmsprop,
              metrics=['accuracy'])
```

但是效果较差如下：



后来分析了源代码发现，TensorFlow 针对分类问题，实现了四个交叉熵函数，分别是

- tf.nn.sigmoid_cross_entropy_with_logits
- tf.nn.softmax_cross_entropy_with_logits
- tf.nn.sparse_softmax_cross_entropy_with_logits
- tf.nn.weighted_cross_entropy_with_logits

sigmoid_cross_entropy_with_logits 的公式分析

For brevity, let `x = logits, z = targets`. The logistic loss is

```
  z * -log(sigmoid(x)) + (1 - z) * -log(1 - sigmoid(x))
= z * -log(1 / (1 + exp(-x))) + (1 - z) * -log(exp(-x) / (1 + exp(-x)))
= z * log(1 + exp(-x)) + (1 - z) * (-log(exp(-x)) + log(1 + exp(-x)))
= z * log(1 + exp(-x)) + (1 - z) * (x + log(1 + exp(-x))
= (1 - z) * x + log(1 + exp(-x))
= x - x * z + log(1 + exp(-x))
```

标准的 Cross Entropy 算法实现，对 W * X 得到的值进行 sigmoid 激活，保证取值在 0 到 1 之间，然后放在交叉熵的函数中计算 Loss。这个函数的输入是 logits 和 targets，logits 就是神经网络模型中的 W * X 矩阵，注意不需要经过 sigmoid，而 targets 的 shape 和 logits 相同，就是正确的 label 值，分类之间是独立的、不要求是互斥。

softmax_cross_entropy_with_logits 的公式分析

Softmax 本身的算法很简单，就是把所有值用 e 的 n 次方计算出来，求和后算每个值占的比率，保证总和为 1，一般我们可以认为 Softmax 出来的就是 confidence 也就是概率，算法实现如下。

For each batch `i` and class `j` we have

```
softmax = exp(logits) / reduce_sum(exp(logits), dim)
```

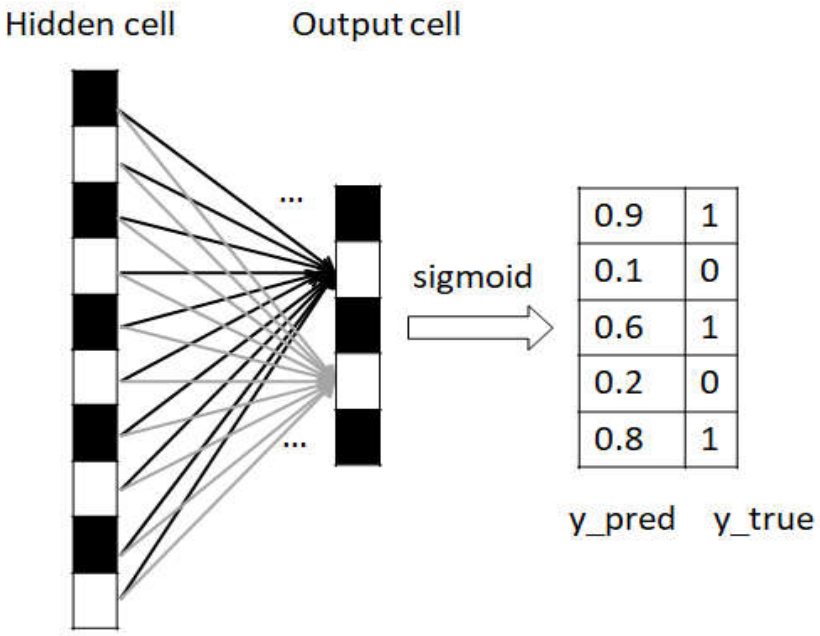
函数实现会在内部更高效得使用 softmax，对于任意的输入经过 softmax 都会变成和为 1 的概率预测值，这个值就可以代入变形的 Cross Entroy 算法- y * ln(a) - (1 - y) * ln(1 - a)算法中，得到有意义的 Loss 值了。如果是多标签问题，经过 softmax 就不会得到多个和为 1 的概率，而且 label 有多个 1 也无法计算交叉熵，因此这个函数只适合单目标的二分类或者多分类问题。

所以最后经过确认后，目标函数，优化函数，性能评估是配置如下：

```
model.compile(loss='binary_crossentropy',
              optimizer=rmsprop,
              metrics=['binary_accuracy'])
```

我的理解是，输出层的每一个节点都类似于二分类分类器，只需要判断对应标签

的预测值是靠近 0 还是靠近 1。每个标签分别对应一个节点，相当于为每个标签训练一个二分类分类器。



y_true：真实的数据标签，Theano/TensorFlow 张量

y_pred：预测值，与 y_true 相同 shape 的 Theano/TensorFlow 张量

由于激活函数选择的是 sigmoid 函数, 所以阈值设置为 0.5, 大于 0.5 的预测为 1, 小于 0.5 的预测为 0。

准确率 = TP / (TP + FP + 0.1)

召回率 = TP / (TP + FN + 0.1)

TP: 将正类预测为正类数

FN: 将正类预测为负类数

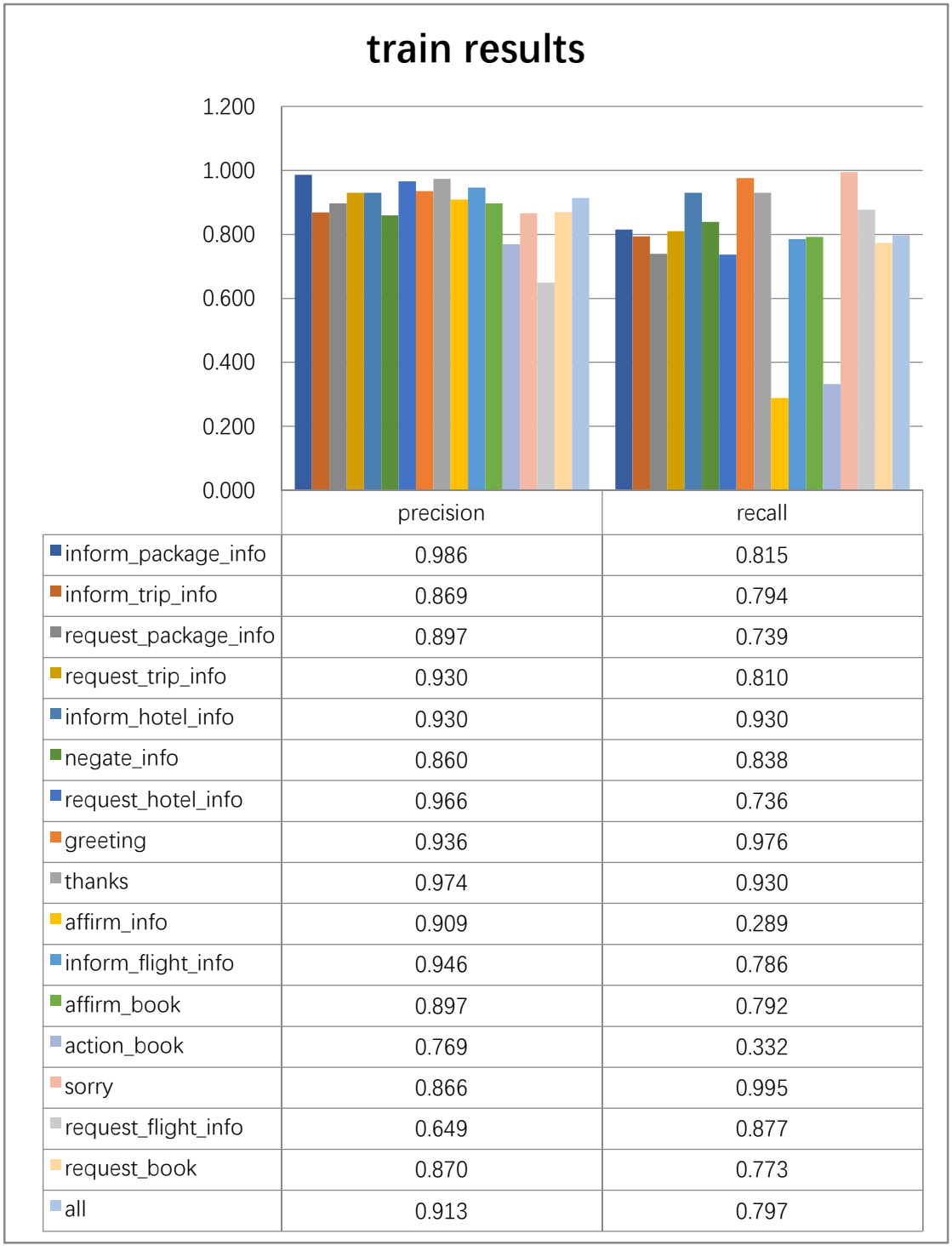
FP: 将负类预测为正类数

TN: 将负类预测为负类数

训练数据结果：

准确率	0.913
召回率	0.797

具体结果如下：

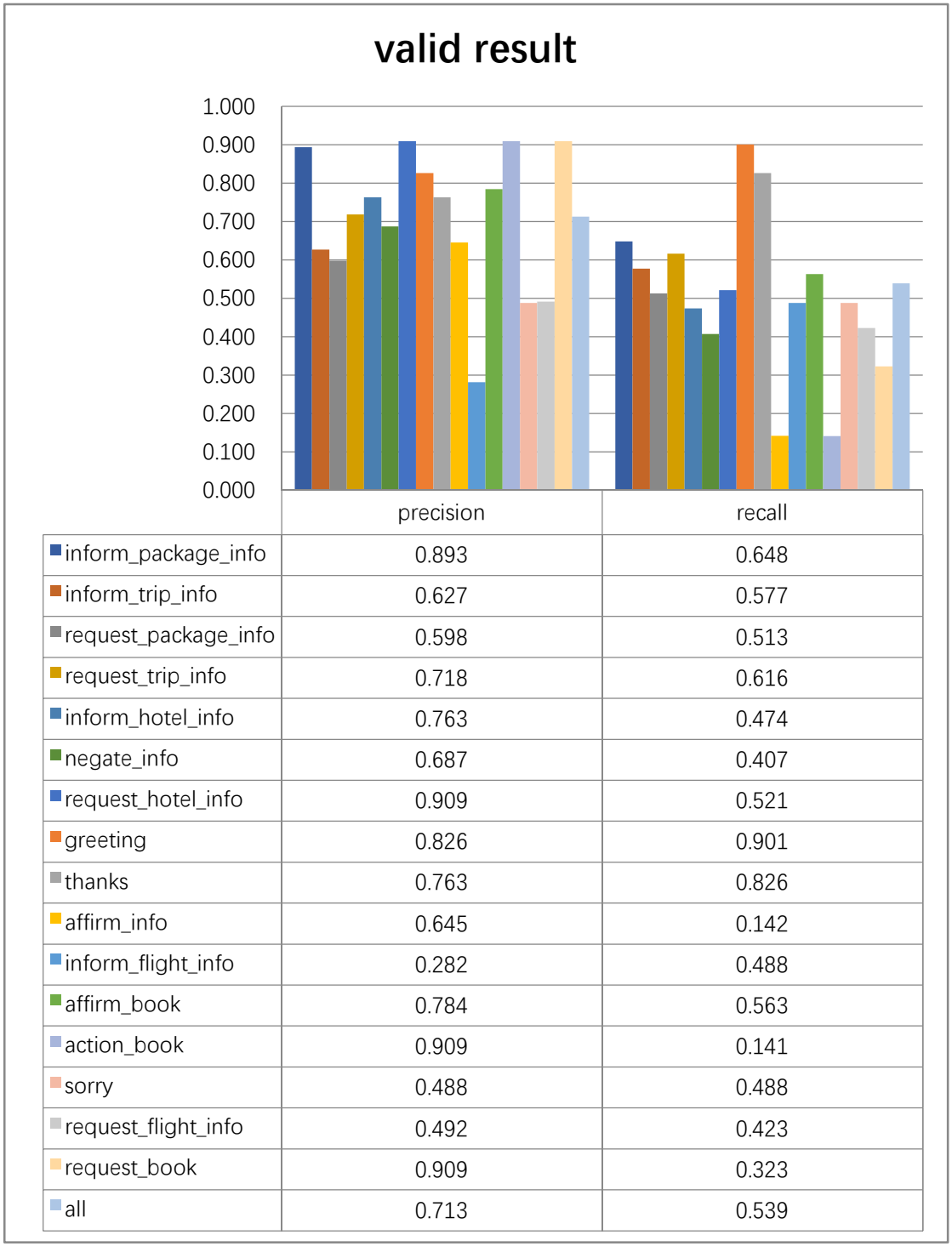


	precision	recall
inform_package_info	0.986	0.815
inform_trip_info	0.869	0.794
request_package_info	0.897	0.739
request_trip_info	0.930	0.810
inform_hotel_info	0.930	0.930
negate_info	0.860	0.838
request_hotel_info	0.966	0.736
greeting	0.936	0.976
thanks	0.974	0.930
affirm_info	0.909	0.289
inform_flight_info	0.946	0.786
affirm_book	0.897	0.792
action_book	0.769	0.332
sorry	0.866	0.995
request_flight_info	0.649	0.877
request_book	0.870	0.773
all	0.913	0.797

验证数据结果：

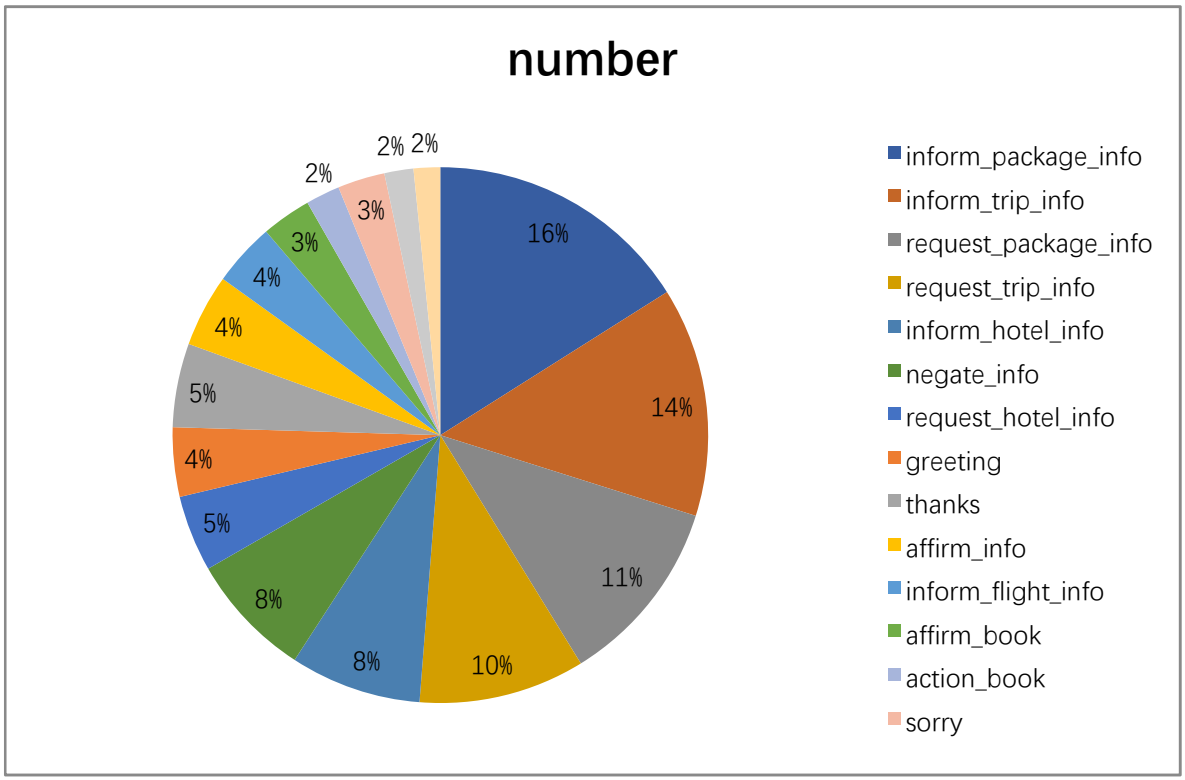
准确率	0.713
召回率	0.539

具体结果如下：



	precision	recall
inform_package_info	0.893	0.648
inform_trip_info	0.627	0.577
request_package_info	0.598	0.513
request_trip_info	0.718	0.616
inform_hotel_info	0.763	0.474
negate_info	0.687	0.407
request_hotel_info	0.909	0.521
greeting	0.826	0.901
thanks	0.763	0.826
affirm_info	0.645	0.142
inform_flight_info	0.282	0.488
affirm_book	0.784	0.563
action_book	0.909	0.141
sorry	0.488	0.488
request_flight_info	0.492	0.423
request_book	0.909	0.323
all	0.713	0.539

如上，标签共有 16 个，标签的分布如下：



Attention 值的分析：

目前实验中，记忆模型的空间定为 5 轮，查看其 attention 发现，在预测结果正确的情况下，通过 softmax 函数计算之后，历史轮次和当前轮次如果意图相同，则 attention 较靠近 1。历史轮次为空时，attention 均为 0.2。

控制门

目前的控制门，感觉还是存在一些问题，gate 值被限制在（0,1）之间，历史意图与当前意图无关时，gate 值靠近 0，历史意图和当前意图相关时，gate 值靠近 1，而查看 gate 值发现 90%的 gate 值都>0.9。