

基于Rasa_NLU的微信chatbot

2017-11-05 00:00:00

chatbot (1)

NLP (2)

重要资料

- 本项目的github地址: Rasa_wechat
- 视频讲解 (正文+编程+QA) : bilibili
- Rasa_NLU官方文档
- Rasa_Core官方文档
- wxpy文档
- 关于机器学习的数学基础和理论介绍, 我整理在机器学习Gitbook
- 如果对本项目有兴趣的话, 可以加入slack小组一起讨论

研究背景



聊天机器人系统框图

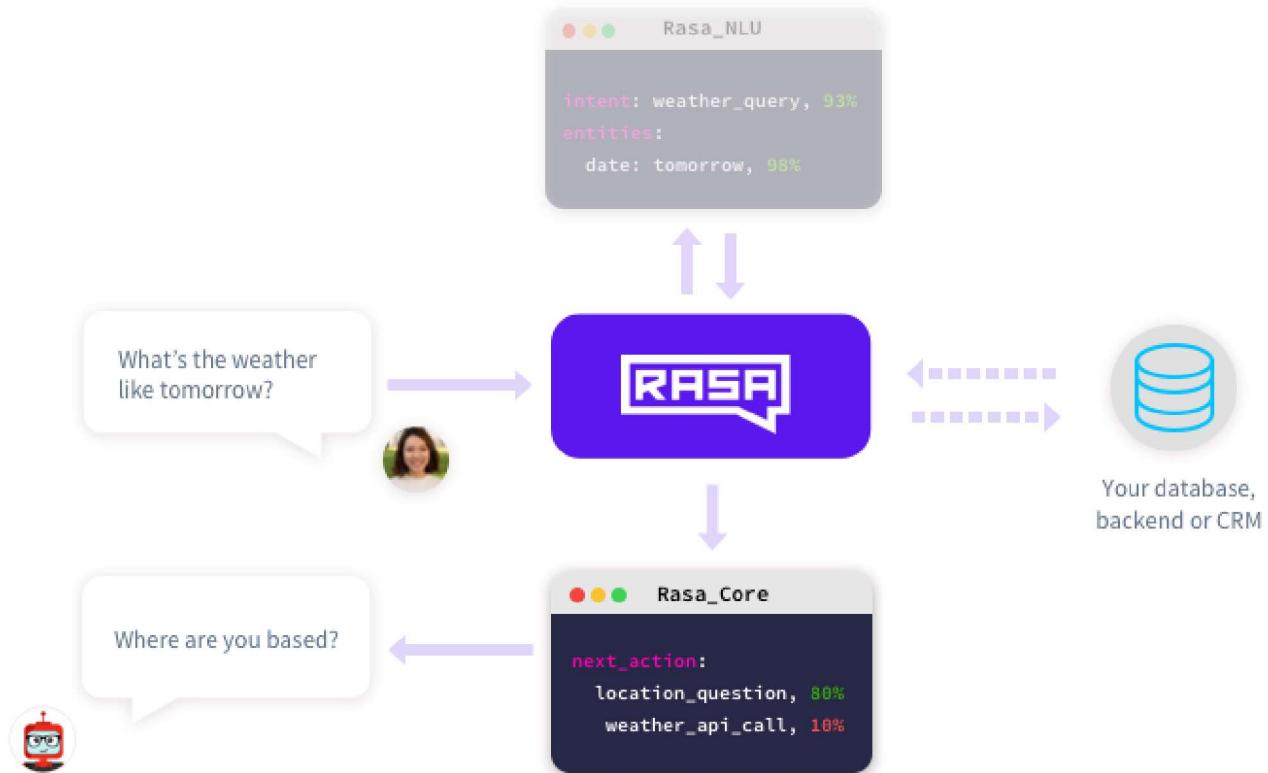
- 老式chatbot基于大量规则库, 不好维护。ps: 一个比较有意思例子是伪春菜。
- 主流架构为” NLU自然语言理解+DM对话管理+NLG自然语言生成”, 也就是上图展示的除ASR和TTS的部分。
 - NLU负责基础自然语言处理, 主要目标是意图识别与实体识别;
 - DM负责对话状态维护、数据库查询等;
 - NLG负责生成交互的自然语言。
- 机器学习尤其是深度学习在不断改进NLU、DM和NLG, 乃至颠覆整个架构:
 - 每一个模块都换成DL模型, 甚至再加入User Simulator做强化学习, 进行端到端的训练;
 - Memory Networks: 将整个知识库都encode在一个复杂的深度网络中, 然后再和encode过的问题结合起来decode生成答案, 可参考Siraj Raval的视频。这个工作最多还是应用在机器阅读理解上, 在垂直领域任务导向的chatbot上的成功应用还有待观察。

步入正题

亦舒的小说《喜宝》中写道: 最希望要的是爱, 很多很多爱, 如果没有爱, 钱也是好的。如果没有钱, 至少我还有健康。

现在的chatbot好像也是这样, 一种是纯聊天 (比如小黄鸡), 一种是针对特定商业应用 (比如智能客服), 这里介绍的其实更偏向后一种。

我们这里要用的Rasa_NLU以及后面介绍的Rasa_Core是rasa.ai提供的开源工具，支持Python 2和3，可以本地部署，自己针对实际需求训练和调整模型，在商业chatbot设计上有颇多考量。此外，我们加上了微信的python接口wxpy来实现微信端的交互，其流程图如下：



(图片来自Rasa_Core官网)

最上面的Rasa_NLU就是最开始讲到的负责自然语言理解的部分，而最下面的Rasa_Core则是在得到了intent和entities之后负责生成next_action以及自然语言的回复。最左边的一问一答则由wxpy来完成监听消息和发送消息的任务。

用伪代码表示的话：

```
#输入
input_string=""
#意图识别
intent_object=intent(input_string)
#回复生成
response=policy(intent_object)
#返回用户
print(response)
```

把大象塞进冰箱的第一步是？

打开冰箱门。先不要着急，首先你得明确想做一个什么样的机器人。比如我想做一个”你妈喊你回家吃饭

“，就是当我说”我饿了”，机器人能像我妈一样问我想吃啥，或者叮嘱我早睡早起少吃麻辣烫。

```
mkdir mom && cd mom
```

bash

现在我们需要两种材料，分别对应NLU和对话管理模块：

- 训练数据：nlu examples + dialogue stories
- 配置文件：nlu config + dialogue domain

```
mom/
  └── data/
    ├── stories.md          # dialogue training data
    └── nlu.md               # nlu training data
  └── domain.yml           # dialogue configuration
  └── nlu_model_config.json # nlu configuration
```

bash

下面先来介绍NLU的部分。

自然语言理解：Rasa NLU

针对用户的问题，NLU模块的任务是：

1. 意图识别（Intent）：在句子级别进行分类，明确意图；
2. 实体识别（Entity）：在词级别找出用户问题中的关键实体，进行实体槽填充（Slot Filling）。

举个例子，当我说“我想吃羊肉泡馍”，NLU模块就可以识别出用户的意图是“寻找餐馆”，而关键实体是“羊肉泡馍”。有了意图和关键实体，就方便了后面对话管理模块进行后端数据库的查询或是有缺失信息而来继续多轮对话补全其它缺失的实体槽。

已有的NLU工具，大多是以服务的方式，通过调用远程http的restful API来对目标语句进行解析完成上述两个任务，例如Google的API.ai，Microsoft的Luis.ai，Facebook的Wit.ai等。

事实上，申请到这类API的话用几行代码即可完成一个chatbot，亦可参考使用图灵机器人和api.ai相关接口。如果想从零开始动手实现NLU，推荐阅读Do-it-yourself NLP for bot developers。

遗憾的是，Rasa_NLU官方目前只支持英语和德语，因此这里参考基于中文的Rasa_NLU来训练自己的中文NLU模型，不过仍然建议认真阅读Rasa_NLU官方文档。

中文Rasa_NLU的安装：

```
$ git clone https://github.com/crownpku/rasa_nlu_chi.git
$ cd rasa_nlu_chi
$ python setup.py install
```

shell

语料获取和预处理

为了增大覆盖面，我们往往需要收集大量的词汇，比如维基百科和百度百科，同时结合自己的应用场景选择特定领域的语料（比如我为了测试加上了百度文库里的中国移动常见问题）。下载并处理中文维基语料的过程参考用Rasa_NLU构建自己的中文NLU系统这篇博文。

在获得并处理好语料以后，中文特色是先进行分词：

```
# 安装jieba分词
$ pip install jieba
# 将一个语料文件分词，以空格为分隔符
$ python -m jieba -d " " ./test > ./test_cut
```

shell

把所有分好词的语料文件放在同一个文件路径下，接下来训练MITIE模型。

训练词表示模型：MITIE的wordprep

MITIE (MIT Information Extraction) 是MIT的NLP团队发布的一个信息抽取库和工具。目前包含：

- 命名实体抽取 (Named-Entity-Recognize, NER)
- 二元关系检测功能 (Binary relation detection)

另外也提供了训练自定义抽取器和关系检测器的工具。它的主要工作在于：

- distributional word embeddings: 简单说来就是将每个词映射到向量空间，向量之间的距离代表词之间的相似度，且在语义、语法两种意义上。关于词向量最新的研究可以看Learned in translation: contextualized word vectors;
- Structural Support Vector Machines

尽管 MITIE 是 C++ 写的，但它也提供了其他语言的调用 API 。先编译成动态库再用 Python 调用即可：

```
conda install libgcc
sudo apt-get install g++
pip install git+https://github.com/mit-nlp/MITIE.git
```

sql

或者

```
MITIE\mitielib>mkdir build
MITIE\mitielib>cd build
MITIE\mitielib\build>cmake ..
MITIE\mitielib\build>cmake --build . --config Release --target install
```

bash

当使用python调用时，如果报错说找不到mitie这个python包，可以手动把mitie的路径加到系统路径中：

```
import os
parent = os.path.dirname(os.path.realpath(__file__))
sys.path.append('..../MITIE/mitielib')
```

php

我们使用wordrep工具来训练所有词向量特征，后面的命名实体模型和关系模型都是建立在它的基础上，使用 `cmake` 构建一下就可直接使用：

```
\MITIE\tools\wordrep>mkdir build
\MITIE\tools\wordrep>cd build
\MITIE\tools\wordrep\build>cmake ..
\MITIE\tools\wordrep\build>cmake --build . --config Release
```

bash

之后训练模型得到 `total_word_feature_extractor.dat`：

```
$ ./wordrep -e /path/to/your/folder_of_cutted_text_files
```

shell

这个过程很漫长，也很占内存，建议在服务器上用 `nohup` 跑。项目文件中包含我基于中文维基和中国移动常见问题训练好的 `total_word_feature_extractor.dat`。

来自小伙伴的建议：pypi 镜像推荐用 <https://mirrors.tuna.tsinghua.edu.cn/> 或 ustc，支持IPV6 不走校园网流量~ nohup 不如 screen或tmux好使吧(—▽—) screen 和 tmux 确实比 nohup 较好用呢，主要厉害在窗口管理和恢复。

训练NLU模型： rasa_nlu.train

首先需要构建示例，作为意图识别和实体识别的训练数据：`mom/data/nlu.json`。举个例子：

```
{
  "text": "这附近哪里有吃麻辣烫的地方",
  "intent": "restaurant_search",
  "entities": [
    {
      "start": 7,
      "end": 10,
      "value": "麻辣烫",
      "entity": "food"
    }
  ]
}
```

- `text` (required)：待处理的句子示例。
- `intent` (optional)：这个句子应该关联的意图。
- `entities` (optional)：`start` 和 `end` 共同定义实体范围，在上面的例子中，`"text": "这附近哪里有吃麻辣烫的地方"`，那么 `text[7:10] == '麻辣烫'`。实体还可以扩展到多个词，而且 `value` 不一定要是你句子中的词，这样一来，就能将同义词、误拼映射到同一个值上，比如下面这个例子：

```
[
  {
    "text": "in the center of NYC",
    "intent": "search",
    "entities": [
      {
        "start": 17,
        "end": 20,
        "value": "New York City",
        "entity": "city"
      }
    ]
}
```

```

        ]
    },
{
    "text": "in the centre of New York City",
    "intent": "search",
    "entities": [
        {
            "start": 17,
            "end": 30,
            "value": "New York City",
            "entity": "city"
        }
    ]
}
]

```

如果嫌手动输入太麻烦，可以使用 `rasa-nlu-trainer` 进行可视化编辑：

```
$ npm i -g rasa-nlu-trainer
$ rasa-nlu-trainer -s 示例文件路径
```

shell



此处输入图片的描述

此外，rasa也支持markdown格式的训练语料，比如：

```

## intent:greeting
- hey
- hello
- moin

## intent:goodbye
- bye
- goodbye
- see you later

## intent:mood_affirm
- yes
- indeed
- correct

## intent:mood_deny
- no
- never
- no way

```

markdown

```

## intent:mood_great
- wonderful
- I am amazing
- I am going to save the world

## intent:mood_unhappy
- my day was horrible
- I am sad

```

有了示例数据，我们还需要 `mom/nlu_model_config.json` 来配置流水线（pipeline）。这里考虑中文支持，配置的Rasa NLU的工作流水线如下：

```
# MITIE+Jieba+sklearn:
[“nlp_mitie”, “tokenizer_jieba”, “ner_mitie”, “ner_synonyms”, “intent_featurizer_
```

- `nlp_mitie`：初始化MITIE，所有mitie组件都依赖于它；
- `tokenizer_jieba`：用jieba来做分词；
- `ner_mitie`：MITIE的命名实体识别；
- `ner_synonyms`：如果在示例中通过 `value` 属性定义了同义词，就将这些value映射到相同value上
- `intent_featurizer_mitie`：提取意图的特征，作为意图分类器 `intent_classifier_sklearn` 的输入；
- `intent_classifier_sklearn`：使用sklearn做意图分类。

更多pipeline内容详见[官方文档pipeline介绍](#)。根据config训练NLU模型：

```
$ python -m rasa_nlu.train -c mom/nlu_model_config.json
```

shell

更方便的做法是python调用：

```
# train_nlu.py : train NLU model
from rasa_nlu.converters import load_data
from rasa_nlu.config import RasaNLUCConfig
from rasa_nlu.model import Trainer

# 训练模型
def train():
    # 示例数据
    training_data = load_data('data/examples/rasa/demo-rasa_zh.json')
    # pipeline配置
    trainer = Trainer(RasaNLUCConfig("sample_configs/config_jieba_mitie_sklearn.js
```

python

```

    trainer.train(training_data)
    model_directory = trainer.persist('./projects/default/') # 返回nlu模型的储存位置

# 识别意图
def predict(model_directory):
    from rasa_nlu.model import Metadata, Interpreter
    # 用nlu模型初始化interpreter, `model_directory`为nlu模型位置
    interpreter = Interpreter.load(model_directory, RasaNLUCConfig("sample_configs"))
    # 使用加载的interpreter处理文本
    print (interpreter.parse(u"我想吃麻辣烫"))

```



`train` 完成后就会得到一个类似 `model_20171013-153447` 的文件存在

`/models/your_project_name` 文件夹里。项目文件中提供训练好的NLU模

型: `/rasa_nlu_chi/models/rasa_nlu_test/model_20171013-153447`。使用的话注意改一下

model目录下 `metadata.json` 的 `mitie_file` 路径。

测试效果除了使用python来 `predict(model_directory)`，还可以启动rasa_nlu的HTTP服务:

```

$ curl -XPOST localhost:5000/parse -d '{"q":"我发烧了该吃什么药?", "project": "rasa_nlu_chi"}'
makefile
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload   Total   Spent    Left  Speed
100  652     0  552  100    100      157       28  0:00:03  0:00:03  ---:--  157
{
  "entities": [
    {
      "end": 3,
      "entity": "disease",
      "extractor": "ner_mitie",
      "start": 1,
      "value": "发烧"
    }
  ],
  "intent": {
    "confidence": 0.5397186422631861,
    "name": "medical"
  },
  "intent_ranking": [
    {
      "confidence": 0.5397186422631861,
      "name": "medical"
    },
    {
      "confidence": 0.16206323981749196,
      "name": "restaurant_search"
    }
  ]
}

```

```

},
{
    "confidence": 0.1212448457737397,
    "name": "affirm"
},
{
    "confidence": 0.10333600028547868,
    "name": "goodbye"
},
{
    "confidence": 0.07363727186010374,
    "name": "greet"
}
],
"text": "我发烧了该吃什么药？"
}

```



以上返回的内容结合了流水线中各个组件的结果，我们可以根据自己的需求设计流水线以及利用返回结果。

现在回到我们的chatbot目录，来讲剩下的dialogue部分的 `domain.yml` 和 `stories.md` :

```

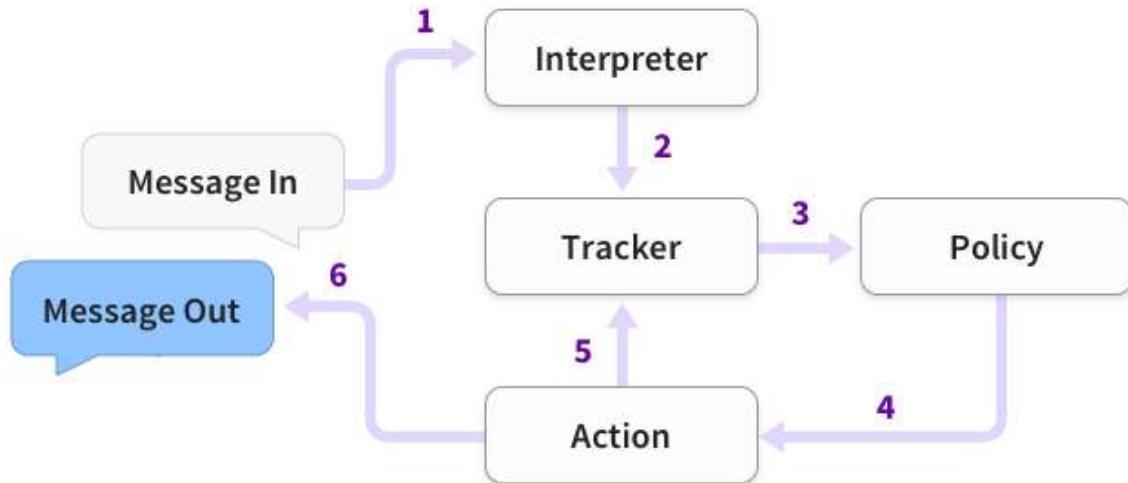
mom/
├── data/
│   ├── stories.md          # dialogue training data
│   └── nlu.md               # nlu training data
└── domain.yml             # dialogue configuration
└── nlu_model_config.json  # nlu configuration

```

bash

对话管理: Rasa Core

Rasa Core是一个用来搭建对话系统的框架，具体可查看[Rasa Core官方文档](#)（似乎要科学上网才能访问）。它的具体工作流程如下：



1. 外面来的信息首先经由interpreter转成text、intent、entities
2. tracker负责记录对话状态，接收interpreter的结果
3. policy收到当前状态信息
4. policy选择下一步action
5. 所选择的action被tracker记录
6. 输出回复

安装

```
git clone https://github.com/RasaHQ/rasa_core.git
cd rasa_core
pip install -r requirements.txt
python setup.py install
```

bash

推荐国内用户从国内镜像（比如豆瓣 PYPI 镜像源 <https://pypi.doubanio.com/simple/>）下载安装，否则 `pip install -r requirements.txt` 可能会各种连接失败。

更改pip源速度快了好多~

运行helloworld示例来检验是否正常安装成功：

```
python examples/hello_world/run.py
Bot loaded. Type hello and press enter :
hello
hey there!
```

coffeescript

定义意图和动作： domain. yml

Rasa基于 `intents` 、 `entities` 和当前对话的内部状态，从 `actions` 中选择下一步采取的行动。

比如机器人选择 `utter_greet` 的话，就会从 `templates` 中找出相应的句子作为回复（可以填充变量）。而这些都需要我们在domain. yml中预先定义好。

`ActionListen` 是一个特殊的action，意味着等待用户，直到说别的内容。当然你还可以定义你自己的action，作为python类来调用。

举个例子：

```
#创建文件common_domain.yml, 复制如下代码 makefile
intents:#定义意图
- greet
- goodbye
- chat

entities:
- action

templates:#表示对于相应的意图采取什么样的回复
utter_greet:
- "hello!"
utter_goodbye:
- "byebye :("
utter_chat:
- "It seems like funny!"

actions:#定义bot可以采取的动作
- utter_greet
- utter_goodbye
- utter_chat
```

- `intents` : NLU模型识别的意图；
- `entities` : NLU模型识别的实体；
- `actions` : 机器人说和做的内容，比如问候对方、调用某个API、查询数据库等；
- `slots` : 用来跟踪上下文和多轮对话的关键信息，具体参考slot types；
- `templates` : 说话模板。

定义解释器（interpreter）：

interpreter用来处理消息，包括执行NLU和把消息转为格式化信息。其实就是上一个part所讲的训练NLU model的过程。这里假定我们已经有了NLU model。

训练你的对话模型

我们通过 `domain.yml` 定义好了 `action`、教会了小宝宝牙牙学语（各种模板），还要告诉ta什么时候说什么话（基于会话历史训练概率模型，来预测需要采取的 `action`）。有两种”教学方法”（训练方式）：

1. 念故事书：如果你有标记好的对话数据，可以直接做有监督学习（`supervised learning`）；
2. 面对面聊天：但大多数人没有监督学习条件，此时推荐从无到有的交互式学习（`interactive learning`）。在交互学习模式下，你的chatbot每做出一个决策，你都要即时给出反馈，这有点强化学习的意思，但强化学习是在对话结束后给反馈。

第二种交互式学习中，当你的chatbot答错时，你需要示范给它正确答案，此时模型会立即自我更新，这样就很难“重蹈覆辙”了。当你结束交互训练时，对话数据会被记录并添加到你的训练数据中。这里主要介绍第一种，我们将故事写在 `stories.md` 中：

```
## happy path          <!-- 故事名 - just for debugging -->
* _greet
  - utter_greet
* _mood_great         <!-- user utterance, in format _intent[entities] -->
  - utter_happy

## sad path 1          <!-- this is already the start of the next story -->
* _greet
  - utter_greet        <!-- action of the bot to execute -->
* _mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
* _mood_affirm
  - utter_happy

## sad path 2
* _greet
  - utter_greet
* _mood_unhappy
  - utter_cheer_up
  - utter_did_that_help
* _mood_DENY
  - utter_goodbye
```

```
## say goodbye
* __goodbye
  - utter_goodbye
```

Wizard-of-Oz (W0z)：自己 (wizard) 扮演chatbot，wizard能通过搜索接口访问数据库，接收用户的输入并决定下一步说什么。W0z收集到的对话数据可用于对话系统各个组件的研究，从NLU到NLG，或者不同领域端到端的对话系统，从而降低为每个领域人工设计新系统的成本。

关于故事格式可以参考官方文档的stories部分。

使用Python API **Agent** 我们可以方便地训练、加载、使用模型：

```
# script/train_dm.py                                         python

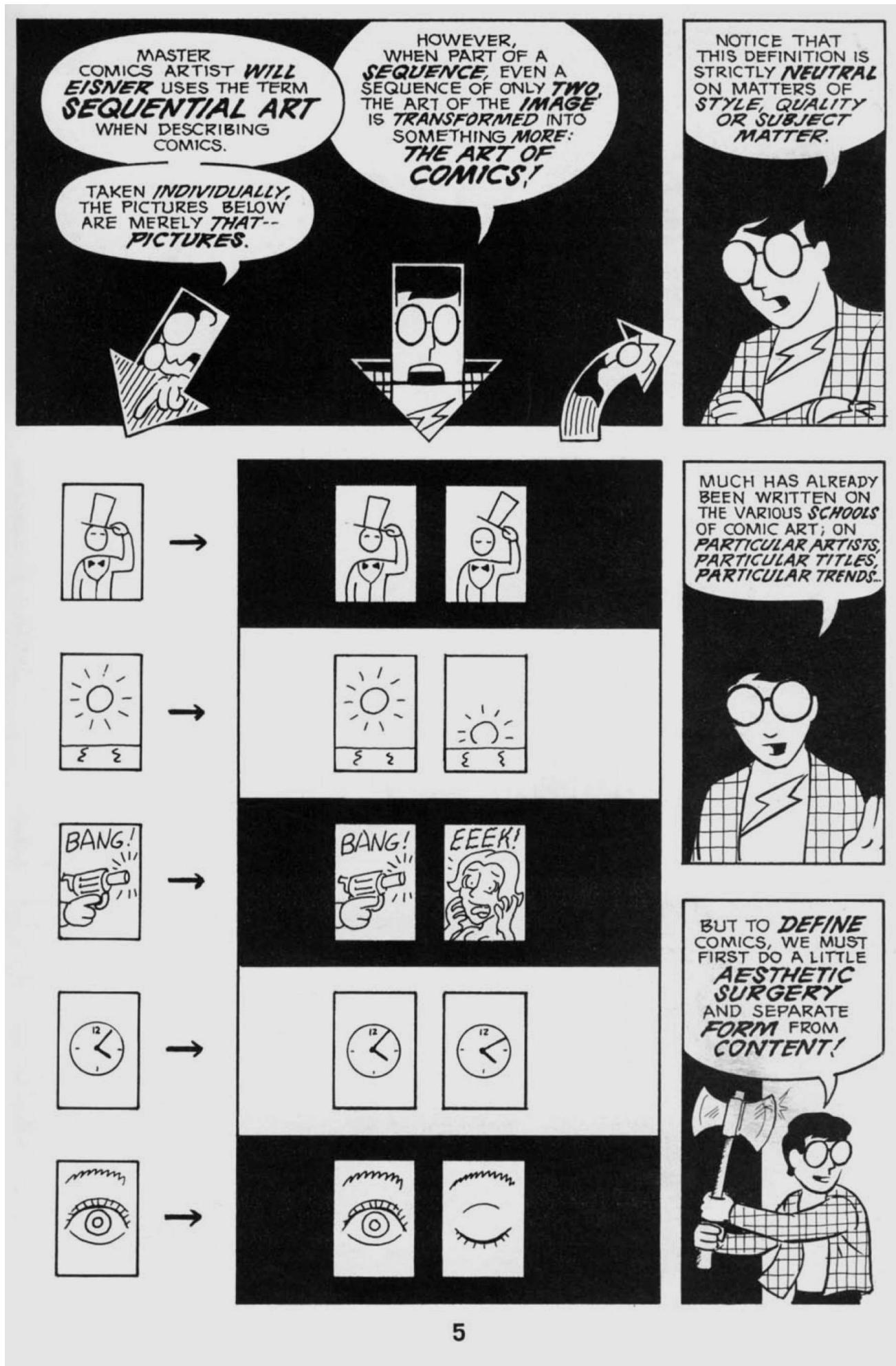
from rasa_core.agent import Agent
from rasa_core.policies.memoization import MemoizationPolicy

# 训练policy模型
def train_mom_dm():
    agent = Agent("../mom/domain.yml",
                  policies=[MemoizationPolicy()])

    agent.train(
        training_data_file,
        max_history=3,
        epochs=100,
        batch_size=50,
        augmentation_factor=50,
        validation_split=0.2
    )

    agent.persist(model_path)
```

看到这里想必你已经非常疲惫了吧，来看个漫画轻松一下：



这是《Understanding Comics》里解释漫画原理的一页，这里借它来说明训练 policy模型的原理，先扫

一眼训练模型的代码：

```
python
class MomPolicy(KerasPolicy):
    def _build_model(self, num_features, num_actions, max_history_len):
        """Build a keras model and return a compiled model.
        :param max_history_len: The maximum number of historical turns used to
                               decide on next action"""
        from keras.layers import LSTM, Activation, Masking, Dense
        from keras.models import Sequential

        n_hidden = 32 # size of hidden layer in LSTM
        # Build Model
        batch_shape = (None, max_history_len, num_features)

        model = Sequential()
        model.add(Masking(-1, batch_input_shape=batch_shape))
        model.add(LSTM(n_hidden, batch_input_shape=batch_shape))
        model.add(Dense(input_dim=n_hidden, output_dim=num_actions))
        model.add(Activation('softmax'))

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

        logger.debug(model.summary())
        return model
```

这里的LSTM其实就是处理漫画这样的sequence数据的关键道具，想象一下，我们的对话模型每句话说完就是一帧图像（当前对话状态），这样一帧一帧输入到LSTM中就是对每个时刻的对话状态进行学习，最终的目标是训练出policy，这样下次再看到“睁眼”，就能猜到下一秒应该是“闭眼”了。

现在我们有了NLU模型和policy模型，只需要再加上微信接口，即 `输入` 和 `返回用户` 的过程。

微信后台

python操作微信号：wxpy

wxpy可用来实现各种微信个人号的自动化操作。具体的使用参见[wxpy文档](#)，应用实例可参考[connectorwechat-bot](#)。这里其实只用了收发消息的接口，“智能”的部分交由后文的Rasa_Core完成。

```
from wxpy import *
# 初始化机器人，扫码登陆
bot = Bot(console_qr=True, cache_path=True)
```

python

由于代码部署在服务器上，不能通过终端打开图片，参数 `console_qr=True` 表示在终端内显示二维码；`cache_path=True` 启用缓存，来保存自己的登录状态。

自己加自己好友方便以后做测试：

```
# 在 Web 微信中把自己加为好友
bot.self.add()
bot.self.accept()

# 发送消息给自己
bot.self.send('能收到吗？')
```

php

wxpy提供了注册消息的方法，可以将各种类型的消息注册并自定义处理方式。我们就是利用它来监听任何发送给自己的消息：

```
from rasa_core.agent import Agent
from rasa_core.interpreter import RasaNLUIInterpreter

@bot.register(bot.self, except_self=False)
def reply_self(msg):
    ans = agent.handle_message(msg.text)
    # 自己训练好的NLU模型路径
    nlu_model_path = '/home/luoling/rasa_nlu_chi/models/rasa_nlu_test/model_20171'
    # agent = 自己训练好的policy模型 + NLU模型作为interpreter
    agent = Agent.load("../babi/models/policy/current", interpreter=RasaNLUIInterp
```

python

重头戏分别是负责语言理解的 `nlu_model` 和负责对话管理的 `agent`，将在下面两部分展开介绍。

最后别忘了让程序保持运行：

```
# 进入 Python 命令行、让程序保持运行
embed()
```

shell

python搭建微信公众号后台

目前还在测试阶段，大部分时间无法正常使用，抱歉~

我参考微信公众号搭chatbot加上了公众号的支持，使用的原材料如下：

- 内存足够（tensorflow跑起来占400M左右）的VPS（我选的bandwagon 20G那个，一年\$49.9，有更便宜的请务必告诉我）
- 微信订阅号

最后的效果欢迎关注公众号测试：



整体来看

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import logging

import numpy as np

from rasa_core import utils
```

python

```

from rasa_core.actions.action import ACTION_LISTEN_NAME
from rasa_core.agent import Agent
from rasa_core.channels.console import ConsoleInputChannel
from rasa_core.domain import TemplateDomain
from rasa_core.interpreter import NaturalLanguageInterpreter
from rasa_core.policies import Policy
from rasa_core.tracker_store import InMemoryTrackerStore

logger = logging.getLogger(__name__)

class CommonPolicy(Policy):
    def predict_action_probabilities(self, tracker, domain):
        # type: (DialogueStateTracker, Domain) -> List[float]
#将对应的意图与动作绑定
        responses = {
            "greet": 2,
            "goodbye": 3,
            "chat": 4,
        }

        if tracker.latest_action_name == ACTION_LISTEN_NAME:
            key = tracker.latest_message.intent["name"]
            action = responses[key] if key in responses else 4
            return utils.one_hot(action, domain.num_actions)
        else:
            return np.zeros(domain.num_actions)

class HelloInterpreter(NaturalLanguageInterpreter):
    def parse(self, message):
        # intent = "greet" if 'hello' in message else "default"
        global intent
#进行意图识别
        if 'hello' in message:
            intent='greet'
        elif 'goodbye' in message:
            intent='goodbye'
        else:
            intent='chat'
        return {
            "text": message,
            "intent": {"name": intent, "confidence": 1.0},
            "entities": []
        }

def run_hello_world(serve_forever=True):
    default_domain = TemplateDomain.load("./common_domain.yml")#加载多个domain怎么办
    agent = Agent(default_domain,
                  policies=[CommonPolicy()],
                  interpreter=HelloInterpreter(),
                  tracker_store=InMemoryTrackerStore(default_domain))

```

```

if serve_forever:
    # Attach the commandline input to the controller to handle all
    # incoming messages from that channel
    agent.handle_channel(ConsoleInputChannel())

return agent

if __name__ == '__main__':
    run_hello_world()

```

Todo List

Future Work

rasa扩展

- 目前对中文entity的处理还有问题（认“salad”但不认“麻辣烫”）
- 实现微信上的交互式训练，包括向提问人学习如何提问（能主动发起会话）；
- 读取历史聊天信息作为训练数据，充分发挥聊天app的优势，难点在于非结构数据的结构化；
- 增量式学习，在增加新语料时不用从头再来；
- 基于google搜索结果和QA模型做chatbot的百事通功能。

微信扩展：

- 表情包推荐功能
 - 预测gif情感
- 结合语音判断情绪，作为辅助信息
- 个奇怪的问题：机器不会主动来“关心”你，这套体系更偏向商业应用，即解决用户的问题

Things to learn

- NLG：自然语言生成
- intent定义：现在还是很生硬的人工编辑，有没有可能自动地从对话中去提炼intent和action
- 目前的conversation representation是binary vector，是否能在语义空间做改进？类似encoder decoder那种
- 对抗式训练，做两个chatbot相互交流，不过得有一个objective function。忍不住想到一个世纪难题：把一个健身房教练和理发店造型师放一间房里，到底谁会先让谁办卡……

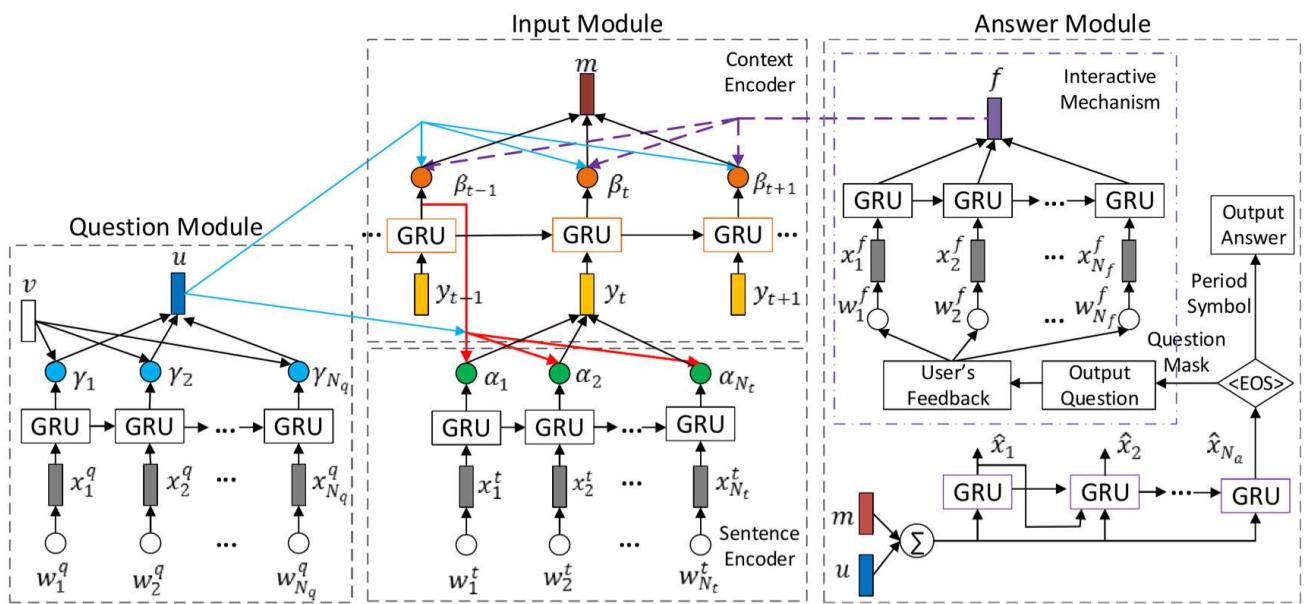
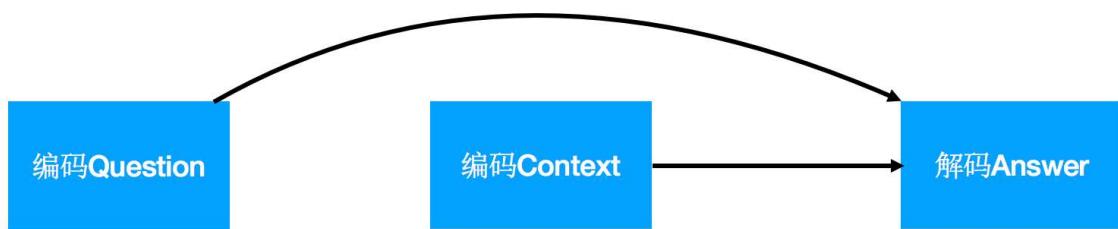
Reference

1. 浅谈垂直领域的chatbot
2. Rasa Blog: A New Approach to Conversational Software

补充：QA系统

QA (Question Answering) 即问答系统，目前研究集中在自然语言问答和视觉问答，分别基于给定的文字和图像回答特定问题。在Chatbot（聊天机器人）和智能客服中应用广泛。

技术特点在于对问题（Question）和给定内容（Context）进行编码（encode），并从中解码（decode）出答案，事实上，相似模型也可应用在基于样例的语音关键词检测（Query-by-Example Spoken Term Detection）中，即对搜索内容（Query）和搜索对象（Context）进行编码，从中解码出关键词（Query）出现与否以及关键词的出现位置。



本文由 [Row11ng](#) 创作，采用 [知识共享署名4.0 国际许可协议](#) 进行许可

本站文章除注明转载/出处外，均为本站原创或翻译，转载前请务必署名

本文会经常更新，最后编辑时间为:2017-11-05 00:00:00

点赞 • 1 个赞

发布issue

尚无评论，快来抢沙发吧~

[写评论](#)[预览](#)[用 GitHub 帐号 登录](#)

(在这里写评论吧)

支持 Markdown

[评论](#)

由 [Gitment](#) 驱动

ROWLING

Theme is Jekyll rowling by Rowling

Powered by Jekyll

© 2018 Rowling blog RSS

本站 UV 1401 本站 PV 3343