CONVERASTIONAL Q&A BOT, SESSION ON DOCUMENTS- DIALOG FLOW

submitted to Dept. Computer Science

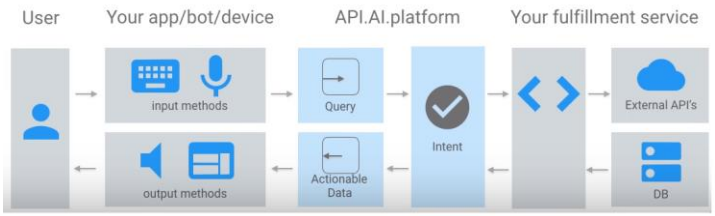
Supervisor - Dr. Hameed Alhoori

by,
Patlolla, Akhil Reddy – Z1803493

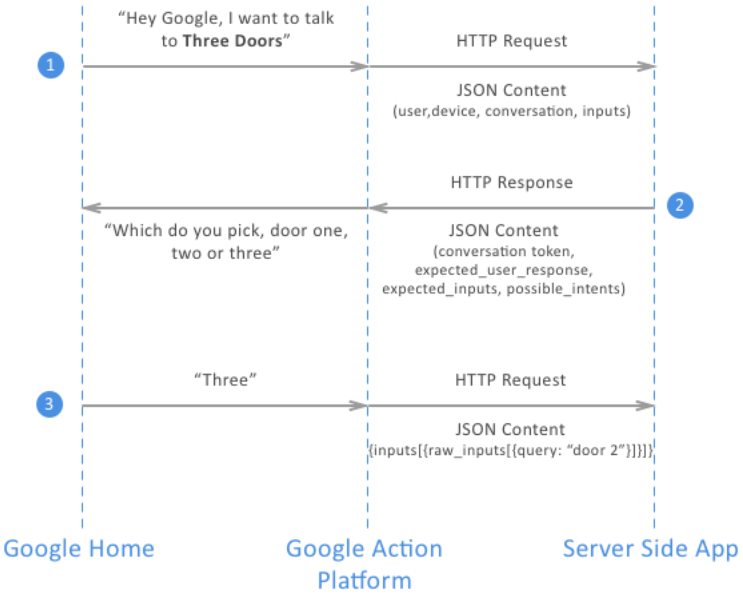
Department of Computer Science
College of Liberal Arts and Sciences
Northern Illinois University
DeKalb, Illinois-60115
Spring 2018

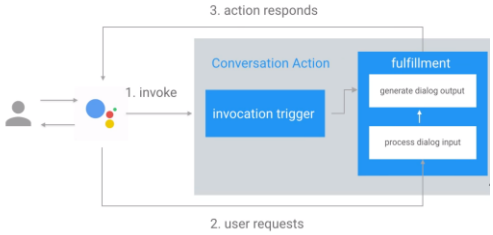
Introduction:

The Whole Idea of the project was to develop an AI Bot which can take in the input document and answer the question by the user. Intelligent Q&A System. The API.AI Python SDK can be used to integrate speech recognition with API.AI natural language processing API. API.AI allows using voice commands and integration with dialog scenarios defined for an agent in API.AI. Google Actions using Machine Learning techniques to cross-check the phrases using MAD LIB technique, a way of asking questions and fill in the blanks based on uses answers.



Conversation API





The apps or bots receives a request which is analyzed and segmented based on the Intent and fulfillment using Node JS or Python API creating a response and understand context.

This explains the dialog flow and user interactions. Invocation trigger will trigger fulfillment which involves the processed dialog input to generate the output by processing the input which is sent to actions responder to the google assistant which is again returned to the user as a response. Google Action Platform takes input from a device which is passed to Server-Side App which is custom agent handling all the requests generating results as JSON content.

Text Extraction form Pdf/ doc/docx format:

```
#!/usr/bin/python
from cStringIO import StringIO
from pdfminer.pdfinterp import PDFResourceManager, process_pdf
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams

input_file = file(sys.argv[1], 'rb')
converter = TextConverter(PDFResourceManager(), StringIO(), laparams=LAParams())
process_pdf(PDFResourceManager(), converter, input_file)
text_file = open(pdf_name.replace(".pdf",".txt"),'w')
text_file.write(StringIO().getvalue())
text_file.close()
```

Recurrent Neural Network

This is an unadulterated NumPy usage of wordage utilizing an RNN. We will have our system figure out how to anticipate the following words in a given passage. This will require an intermittent design since the system should recall a grouping of characters. The request matters. 1000 emphases and we'll have pronounceable English. The more drawn out the preparation time the better. You can sustain it any content succession (words, Python, HTML, and so on.)

What is a Recurrent Network?

Feedforward networks are great for learning a pattern between a set of inputs and outputs.

- temperature & location
- height & weight
- car speed and brand

But what if the ordering of the data matters?

Letters in order, Lyrics of a tune. These are put away utilizing Conditional Memory. You can just access a component on the off chance that you approach the past components (like a LinkedList). We bolster the concealed state from the past time advance once again into the system at whenever step. So rather than the information stream activity happening this way

input -> hidden -> output

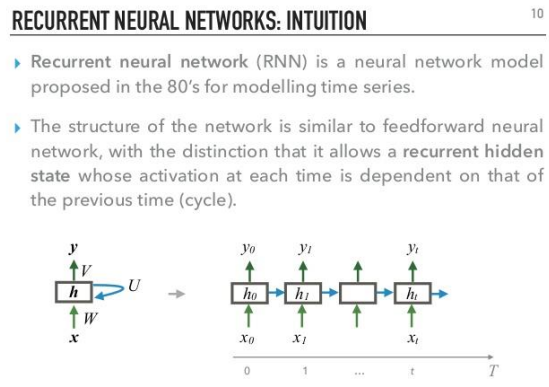
it happens like this

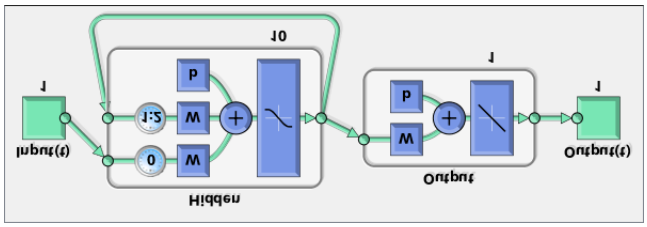
(input + prev_hidden) -> hidden -> output

wait. Why not this?

(input + prev_input) -> hidden -> output

Hidden recurrence learns what to remember whereas input recurrence is hard-wired to just remember the immediately previous data point.





RNN Formula

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta),$$

It essentially says the currently concealed state h(t) is a capacity f of the past shrouded state h(t-1) and the present info x(t). The theta is the

parameters of the capacity f. The system normally figures out how to utilize h(t) as a sort of lossy outline of the assignment significant parts of the past success of contributions up to t.

Loss function

$$L(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}) = \sum_t L^{(t)} = \sum_t -\log \hat{y}^{(t)}_{y^{(t)}}. \qquad (10.7)$$

The aggregate misfortune for a given succession of x esteems combined with a grouping of y esteems would then be only the whole of the misfortunes over all the time steps. For instance, if L(t) is the negative log-probability of y (t) given x (1), . . . , x (t), then total them up you get the misfortune for the grouping

Steps

- Initialize weights randomly
- Give the model a char pair (input char & target char. The target char is the char the network should guess, its the next char in our sequence)
- Forward pass (We calculate the probability for every possible next char according to the state of the model, using the parameters)
- Measure error (the distance between the previous probability and the target char)
- We calculate gradients for each of our parameters to see the impact they have on the loss (backpropagation through time)
- update all parameters in the direction via gradients that help to minimize the loss
- Repeat! Until our loss is small AF

What are some use cases?

- Time series prediction (weather forecasting, stock prices, traffic volume, etc.)
- Sequential data generation (music, video, audio, etc.)

The code contains 4 parts

- Load the training data
  - encode char into vectors
- Define the Recurrent Network
- Define a loss function
  - Forward pass
  - Loss
  - Backward pass
- Define a function to create sentences from the model
- Train the network
  - Feed the network
  - Calculate gradient and update the model parameters
  - Output a text to see the progress of the training

The **loss** is a key concept in all neural networks training. It is a value that describes how good is our model.

The smaller the loss, the better our model is.
(A good model is a model where the predicted output is close to the training output)

During the training phase, we want to minimize the loss.

The misfortune work figures the misfortune yet, in addition, the inclinations (see in the reverse pass): •It plays out a forward pass: figure the following roast given a burn from the preparation set.
•It figures the misfortune by contrasting the anticipated roast with the objective burn. (The objective roast is the information following burn in the tanning set)
•It figures the regressive go to ascertain the inclinations This function takes as input:

- a list of input char
- a list of target char
- and the previously hidden state

This function outputs:

- the loss
- the gradient for each parameter between layers
- The last hidden state

Advanced dynamic seq2seq with TensorFlow

The encoder is bidirectional at this point. Decoder is actualized utilizing tf.nn.raw_rnn. It sustains already produced tokens amid preparing as contributions, rather than target succession.

consistent seq2seq Rectangles are encoder and decoder's intermittent layers. Encoder gets [A, B, C] grouping as data sources. We couldn't care less about encoder yields, just about the concealed state it amasses while perusing the arrangement. After information succession closes, encoder passes its last state to decoder, which gets [<EOS>, W, X, Y, Z] and is prepared to yield [W, X, Y, Z, <EOS>]. <EOS> token is an exceptional word in vocabulary that signs to the decoder the start of interpretation.

Usage subtle elements

TensorFlow has its own usage of seq2seq. As of late it was moved from center cases to TensorFlow/models repo and utilizes censured seq2seq execution. Belittling happened claiming it utilizes static unrolling.

Static unrolling includes the development of calculation diagram with a settled grouping of the time step. Such a chart can just deal with successions of particular lengths. One answer for taking care of groupings of fluctuating lengths is to make numerous diagrams with various time lengths and separate the dataset into this containers.

Dynamic unrolling rather utilizes control stream operations to process arrangement well ordered. In TF this should more space effective and similarly as quick. This is presently a prescribed method to actualize RNNs.

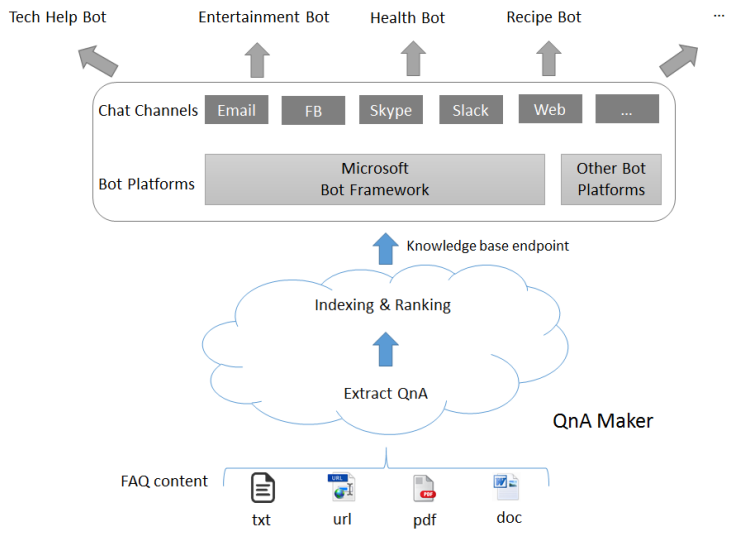
we give encoder input grouping like 'hi how are you', we take the last concealed state and nourish to the decoder and it will produce a decoded esteem. we contrast that with target esteem if interpretation would be 'bonjour cava' and limit the distinction by streamlining a misfortune work for this situation we simply need to encode and translate the information effectively bidirectional encoder We will educate our model to retain and replicate input arrangement. Arrangements will be arbitrary, with differing length. Since arbitrary arrangements don't contain any structure, the model won't have the capacity to misuse any examples in information. It will basically encode grouping in an idea vector, at that point interpret from it. this isn't about expectation (ultimate objective), it's tied in with understanding this engineering this is an encoder-decoder design. The encoder is bidirectional, so It nourishes beforehand produced tokens amid preparing as contributions, rather than target succession.

Dialog Flow:

It's a natural language procession tool and was called API.AI. They are just decision trees and machine learning to better understand the input. There are examples what user give input and understand in future to get an edge on prediction. It is flexible and able to learn and understand the given resources.

Microsoft QnA:

One of the fundamental prerequisites in composing our own particular Bot benefit is to seed it with inquiries and answers. As a rule, the inquiries and answers as of now exist in content like FAQ URLs/archives, item manuals, and so on. With QnA Maker, clients can inquiry your application in a characteristic, conversational way. QnA Maker utilizes machine figuring out how to remove applicable inquiry answer sets from your substance. It additionally utilizes capable coordinating and positioning calculations to give the ideal match between the client inquiry and the inquiries. The simple to-utilize graphical UI empowers you to make, oversee, prepare and utilize your administration with no designer encounter.



The knowledge base is generated and can be used as a source for any further questioning.

Extraction: Structured inquiry answer information is separated from semi-organized information sources, for example, FAQs and item manuals. This extraction is done while making the learning base. See here to figure out how to make your insight base.

Coordinating: Once your insight base has been prepared and tried, you distribute it. This empowers an endpoint to your QnA Maker information base, which you would then be able to use your bot or application. This endpoint acknowledges a client question and reacts with the best inquiry/answer coordinate in the information base, alongside a certainty score for the match.

This QnA stack comprises of the accompanying parts:

QnA administration administrations (control plane): The administration encounter for a QnA Maker information base, which incorporates creation, refresh, preparing, and distributing. These exercises should be possible through the gateway or the administration APIs. The administration administrations converse with the runtime part beneath.

QnA runtime (information plane): The information and runtime are sent in the client's Azure membership in an area of their picking. Client question/answer content is put away in Azure Search, and the runtime is sent as App benefit. Alternatively, you can likewise send an Application bit of knowledge asset for examination.

BOT UI / Angular JS:

It's an HTTP post request to dialog flow to implement webhooks and read the response to create a clean chat interface to work within an Angular JS.

```
import { Injectable } from '@angular/core';
import { environment } from '../../environments/environment';
import { ApiAiClient } from 'api-ai-javascript';
import { Observable } from 'rxjs/Observable';
import { BehaviorSubject } from 'rxjs/BehaviorSubject';
export class Message {
  constructor(public content: string, public sentBy: string) {}
}
@Injectable()
export class ChatService {
  readonly token = environment.dialogflow.angularBot;
  readonly client = new ApiAiClient({ accessToken: this.token });
  conversation = new BehaviorSubject<Message[]>([]);
  constructor() {}
  // Sends and receives messages via DialogFlow
  converse(msg: string) {
    const userMessage = new Message(msg, 'user');
    this.update(userMessage);

    return this.client.textRequest(msg)
               .then(res => {
                  const speech = res.result.fulfillment.speech;
                  const botMessage = new Message(speech, 'bot');
                  this.update(botMessage);
               });
  }
```

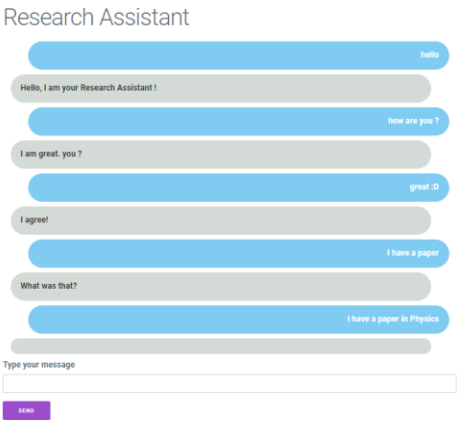
The API client call is made and access token is passed to open a session. The text from the chat context is made as a request and response are generated on the dialog flow.

Behavior subject is defined as an array of messages and another method add other users message into the behaviors array and then take action on the API Call which in turn updates the bot's response in the same array.

To have an observable array which is in turn updated with new messages thereby generating the value.

```
import { Component, OnInit } from '@angular/core';
import { ChatService, Message } from '../chat.service';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/scan';


@Component({
  selector: 'chat-dialog',
  templateUrl: './chat-dialog.component.html',
  styleUrls: ['./chat-dialog.component.scss']
})
export class ChatDialogComponent implements OnInit {

  messages: Observable<Message[]>;
  formValue: string;

  constructor(public chat: ChatService) { }

  ngOnInit() {
    this.messages = this.chat.conversation.asObservable()
        .scan((acc, val) => acc.concat(val) );
  }

  sendMessage() {
    this.chat.converse(this.formValue);
    this.formValue = '';
  }
```

That's just an ng class for each message which is, in turn, applied conditionally to the class whatever it's determined. Simple Miligram CSS is used.



References:

1. Lloret E, Plaza L, Aker A. The challenging task of summary evaluation: an overview. Language Resources & Evaluation [serial on the Internet]. (2018, Mar), [cited May 8, 2018]; 52(1): 101-148.
2. Wang W, Li Z, Wang J, Zheng Z. How far can we go with extractive text summarization? Heuristic methods to obtain near upper bounds. Expert Systems With Applications [serial on the Internet]. (2017, Dec 30), [cited May 8, 2018]; 90439-463.
3. Rada Mihalcea. 2004. Graph-based ranking algorithms for sentence extraction applied to text summarization. In Proceedings of the ACL 2004 on Interactive poster and demonstration sessions (ACLdemo '04). Association for Computational Linguistics, Stroudsburg, PA, USA, Article 20. DOI=http://dx.doi.org/10.3115/1219044.1219064
4. John M. Conroy and Dianne P. O'Leary. 2001. Text summarization via hidden Markov models. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '01). ACM, New York, NY, USA, 406-407. DOI:https://doi.org/10.1145/383952.384042
5. Yihong Gong and Xin Liu. 2001. Generic text summarization using relevance measure and latent semantic analysis. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '01). ACM, New York, NY, USA, 19-25. DOI: https://doi.org/10.1145/383952.383955
6. Dialog flow, https://cloud.google.com/dialogflow-enterprise/docs
7. Microsoft QnA, https://www.qnamaker.ai
8. Akash Kandpal, Recurring Neural Networks, https://codeburst.io/recurrent-neural-network-4ca9fd4f242