Developing a Generalized Intelligent Agent by Processing Information on Webpages

by

Phillip Gabriel Hege

S.B. Electrical Engineering & Computer Science,
Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by .....
Tomaso Poggio
Eugene McDermott Professor at the Department of Brain and Cognitive Sciences
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Developing a Generalized Intelligent Agent by Processing Information on Webpages

by

Phillip Gabriel Hege

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2017, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed and implemented a framework for reinforcement learning (RL) agents to interact with a web environment. With this framework, I introduce a new challenge for RL agents to learn human activity on the web. By defining a series of tasks such as using the web as a navigable resource to perform actions, I introduce an extention to natural language processing models. The framework provides an agent with rich features including element positioning, color, and size in order to process text represented in a 2D web space.

Thesis Supervisor: Tomaso Poggio
Title: Eugene McDermott Professor at the Department of Brain and Cognitive Sciences

Acknowledgments

I would like to give special thanks to Qianli Liao and to the members of CBMM for their extraordinary support on this thesis. The ideas discussed are the result of the many conversations between Qianli and me over the course of the year.

Contents

List of Figures

Chapter 1

Introduction

Since the wide adoption of the world wide web, much of human knowledge has been made accessible through it. Many people interact with this resource on a daily basis through a web browser, which translates information encoded in markup language into easily viewable content. Although online resources may range in type, from encylopedic references, to news articles, to opioninated blog posts; the information all can be viewed through the same web browser. An intelligent agent that is capable of navigating our world must also be able to navigate the web as well. This thesis primarily describes a framework that allows reinforcement learning agents to interact with a web environment through a browser. The framework allows researchers to easily create simple extensions using standard web technologies like html, css, and javascript while not having to worry about implementation details such as connecting their custom model to the environment or how to calculate model accuracy.

Humans do not memorize everything they need to know. They are resourceful and rely on reference materials that help them complete tasks. The best references are designed and formatted to optimize human efficiency. Decisions regarding image and caption arrangement, page formatting, and text coloring all convey information to a human. AI agents must be able to parse reference material as well as humans can in order to learn more quickly and excel at general knowledge tasks.

The popularization of artificial intelligence has brought with it a number of development environments and frameworks. Computationaly, there are Tensorflow and Theano, which have become standard platforms for general AI development. Additionally, OpenAI released the reinforcement learning measurement and training environment, Universe, and its companion interface, Gym. These tools allow researchers

to test models against various popular video games. The video games feature simple interfaces and allow models to make decisions in an open environment more similar to a real environment.

This thesis introduces a new challenge for reinforcement learning agents to accomplish; training a reinforcement learning agent to complete multiple offline tasks with the help of online resources. Data inputted from the online resource contains multidimensional information including text location on a 2d grid, size, and coloring. I define simple tasks such that agents can begin training. More specifically, I go on to describe how to evaluate an agent on its ability to use the information provided from a web page and define a new training workflow based on the standard reinforcement learning workflow.

1.1 Motivation for Browser Environments

The idea of a browser based environment for reinforcement learning stems from the desire to modularize machine learning research. Computer vision is a difficult problem that many higher level learning tasks are dependent. For example, solving any visual question answering (VQA) problem or training an RL agent to play a video game requires understanding information that is displayed on the screen. Convolutional neural networks are becoming the standard go-to image parsing method for reinforcement learning tasks[9]. As long as vision is an open problem, general intelligence agents can only rely on feature extraction from the current "best" computer vision models. There is no way of knowing whether the output of the vision models is even the best input for the next stage of a higher learning model. Until a solution to the problem is found, it is prudent to find a substitute method in order to standardize learning accuracy and success. If there were a way to automatically extract the important visual information without yet having a perfect model, then general intelligence research could continue unencumbered. Providing a model with visual elements in addition to raw images gives learning algorithms more features with which to train and can accelerate the search for an ideal model. Building a tool that can extract information from a graph or image will increase modularity between the research and development of computer vision models and general intelligence models.

Alternatively, natural language processing based tasks such as reading comprehension are accomplished by first serializing text input. These tasks are designed to

process text directly and forgo all means of perceptual understanding. Many NLP models cannot handle inputs that are not linear[1]. Therefore, information organized into tables or graphs cannot easily be tested, unless they are treated like an image. Leveraging web environments provides a robust language to generate multimedia content. Web technology has developed rapidly, and is how many people recieve and process information today. Web browsers display content in a meaningful way such that we can easily parse the information.

AI research is defined by the various tasks available. Each has its own unique dataset with some combination of formatted text and images. There have been attempts to standardize this data collection method, however few have used existing web technologies to generate, manipulate and store training and validation data. This has partially been because of a lack of interest in web technologies in AI research.

There also appeard to be a lack of coverage on web environments. Most of the existing RL environments extend video games or simplify web interfaces into basic components. None of them tackled the web itself as an environment for training and testing models. Because we spend so much time on the web, it should also be an area of focus.

1.2 Goals

I have identified a set of criteria necessary to measure the effectiveness of the proposed framework.

Customizable In order for a framework to be useful it must be capable serving many needs. The framework should be agnostic of model.

Simple There are a large number of frameworks that all provide different specific features and behave in slightly different ways. A framework that is complicated to understand will not be used for very long. There should be few features that are focused on providing intuitive functionality.

Compatible The best way to increase liklihood of use, the framework must be compatible with the system architectures researchers use to train or test their models currently.

1.3 Thesis Structure

In chapter 2, I briefly explain the concepts necessary to understand my work, including how web browsers display web pages, what is reinforcement learning, and the intuition behind convolutional neural networks.

In chapter 3, I describe related work to my thesis where I analyze the existing OpenAI framework, what is missing from it, and what additions I will make to it. I also analyze existing AI challenges such as visual question answering, and include possible areas of improvement.

In chapters 4 - 6, I describe the extension I wrote to the OpenAI framework that allows control over a web browser, and how I developed the microweb environment where a reinforcement learning agent can safely explore during training. I analyze how this framework improves current challenges such as visual question answering and how it can potentially extend AI research into new fields. Additionally, I explain how I tested the OpenAI extension and showed that it works with current models, like convolutional neural networks.

Lastly in chapter 7, I review my contributions to AI research as a whole, and make recommendations for future work in the field.

Chapter 2

Concepts and Terminology

2.1 Web Browsers

Web Browsers are commonly used programs that convert scripts written in hypertext markup language (html), cascading style sheets (css), javascript into a functional user interface. A general html page is made up of a series of tags which define the items that appear on the screen. For instance, a hyperlink is surrounded by the <a> tag, an image has the <img> tag, and a paragraph has the <p> tag. Elements can be named as well, by class or by id. Typically an id will be locally unique and no other elements will have the same id. Elements with the same class will share the same styling and attributes. The in-memory representation of a webpage is referred to as a **Document Object Model (DOM)**. CSS and javascript both interact with the DOM and is the key to making webpages interactive. CSS is used to modify the visual appearance of elements such as the color, font, and size. Lastly, javascript provides computational logic to web pages. Javascript defines how an interface will respond to actions taken by the user whether they be click, scroll, or type commands.

2.2 Reinforcement Learning

Reiinforcement Learning is the area of machine learning that studies how **agents** perform **actions** in an **environment**. An **observation** is the information that an agent receives from the environment. An agent then decides to make an action which will alter the environment in some way. Lastly, an environment has a **reward** function. Based on the agent's action and how it alters the environment, a value is calcu-

lated that scores how good or bad the agents performance was. Here is a diagram illustrating the control flow of a reinforcement learning environment.

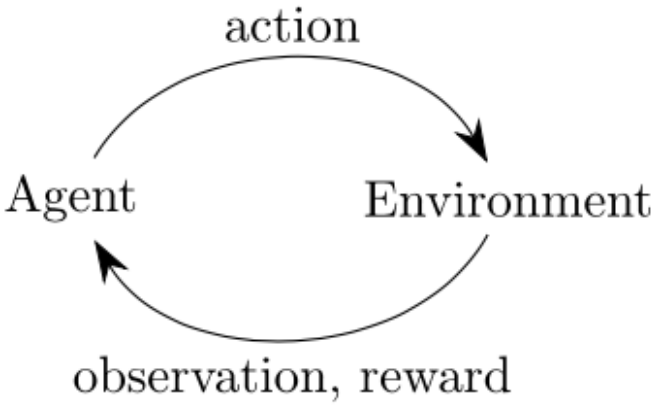

Figure 2-1: Reinforcement Learning Workflow. The agent performs an action in the environment which affects the state of the environment in some manner. A reward is calculated by the environment and returned to the agent. The change in state is returned in the form of an observation to the agent.

This is a popular field of study because it requires combining perceptual understanding with logical reasoning. It is not enough that an agent can correctly identify an object, it must also be able to reason logically, plan, and make a decision that will benefit it over the longterm. A common place to find reinforcement learning challenges is in video games because playing video games maps directly to the RL paradigm. The environment is the screen, the agent is a character, actions are button presses on a controller, and the reward is the game score.

2.3 Models

2.3.1 Convolutional Neural Network

A Convolutional Neural Network is a type of **neural network** that involves multiple **layers**. **Convolution** occurs between subsequent layers in the network. CNNs have show utility in the development of image parsing applications shown by the entries of the ImageNet competition. At a high level they extract the features from an

image by passing a filter over patches of an image. Then systematically combining and recombinging the filtered patches until a decision signal is produced. Every connection has a weight attached to it. During the training phase the weights are adjusted to maximize the networks success rate over all images in the training set. There is usually an output vector over a range of all possible answers with a probability attached to the liklihood of each possibility.[5]

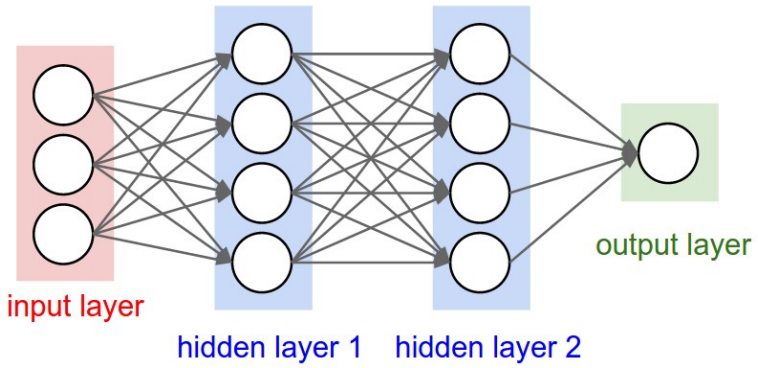

Figure 2-2: A visualization of a simple neural network arrangement from cs231n [4]. The input layer receives some value from data, and passes it through the edges of the graph to nodes in subsequent layers. Each edge has a weighted value attached to it that amplifies or diminishes the input. A node fires when it has been stimuluated by enough of its input nodes. The output layer in this case will either turn on or off if its inputs satisfy a threshold.

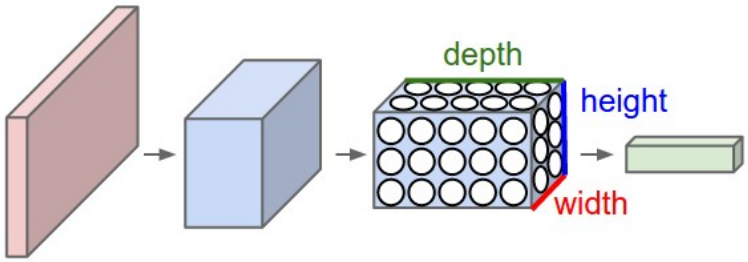

Figure 2-3: A visualization of convolution over an image with vector output from cs231n [4]. An extension of the simple network designed to work over an entire image. Notice the hidden layers have fewer nodes than the input image has pixels. Each neuron in the hidden layer is connected to a group of pixels and convolves over them, resulting in a reduced version of the starting image. Performing convolution many times after which leads to the output vector, where each location represents a possible object classification.

Initally the network begins with a random assignment of edge weights. During the training stage, as images are passed through the network an output signal is produced. The output signal is compared to the true signal and the difference between signal and ground truth propagates backward through the CNN in a process known as **backpropagation**. Backpropagation is the analog to learning where weight corrections are made. This process takes a lot of data, time and computational resources to finely tune these models.

The wonder behind CNNs is that upon inspection one can observe the filters produced within the network and see the beginnings of edge and shape detection.

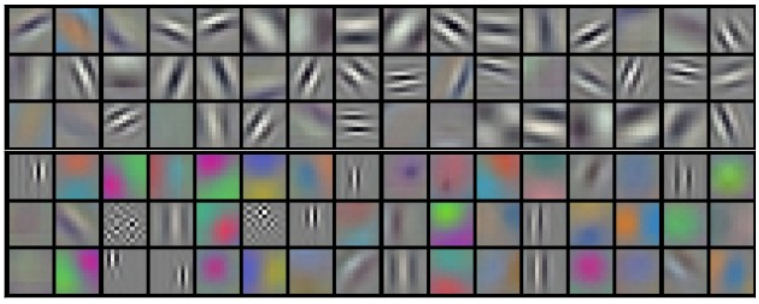

Figure 2-4: Example filters produced by Krizhevsky et al[7]. Here are 96 filters that detect various kinds of line and shade in an input image. This was created after being shown to a large set of images. These filters produce the features necessary for input into more learning networks in order to solve more tasks.

A common practice currently is leveraging pre-trained CNNs using them as a starting point to solve more complex problems. By removing the final ouptut layer and then connecting them to other AI models, the combined model can then begin to complete tasks such as VQA, drive cars, or play video games.[2]

Chapter 3

Related Work

3.1 OpenAI

Developing a framework that leverages web technologies can extend pre-existing, popular AI libraries, which currently train and validate one distinct environment at a time. OpenAI provide the libraries, "Universe" and "Gym"[3] in order to train an agent on a particular task or game (i.e. pong). The Universe architecture follows the standard reinforcement learning process, input action, output next frame in pixels and score. This happens repeatedly until either a maximum number of turns has been taken or when a completion state has been reached.

3.1.1 Gym

The OpenAI architecture is outlined as three modules: The agent, the environment and the interface in between. Typically, a researcher designs a custom agent, while OpenAI provides the interface and the environment. Most of the provided environments are games (i.e. pong). OpenAI implements the standard reinforcement learning procedure with the following adaptations:

1. The image from the original game is generated.
2. The agent makes a 'legal' move within the world of the environment based on the input image
3. The in-game score is updated
4. A new image from the game is generated

5. A pre-existing Optical Character Recgonition (OCR) model extracts the score from the image. In a game like pong, the score boxes appear on the screen above the gameplay.

6. The score and the gameplay image is passed to the RL agent in order to generate a new action.

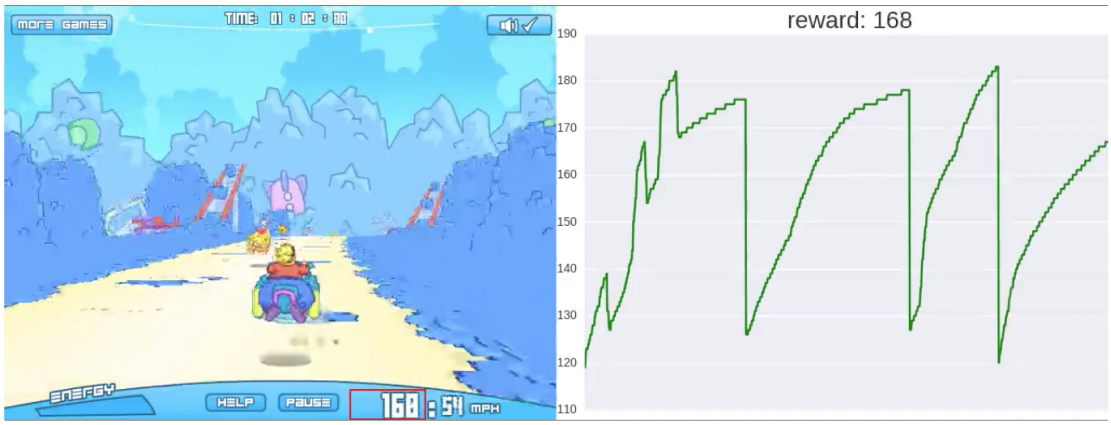

Figure 3-1: OpenAI's implementation of OCR

Although the workflow is straightforward, to add a new environment type requires additional work to extract the score from the game. This inherently limits the speed at which new games can be added, and the type of games that can be added, thus limiting the project's scope.

3.1.2 Analysis

The tasks provided by OpenAI Gym are primarily arcade style video games, with simple game mechanics[10]. This is a mechanism that allows for agents to explore indepth environments while still having full developmental control. Training agents to play games develops a visual processor in a more realistic way but existing in a video game is simulation, not realism.

3.2 Tasks

This section discusses the current state of AI and their drawbacks as well as describe a new set of tasks that can push research in perception and intelligence. AI research

tasks are currently divided into various problem areas including Natural Language Processing (NLP), Percption, and Reasoning. There have also been developments made in bridging the gap between these areas through tasks and endeavors like Visual Question Answering (VQA) and OpenAI Universe. VQA requires agents to parse images, understand its contents enough to answer logical questions about the image. OpenAI Universe provides access to various video games which require agents to parse an image and perform an action in order to optimize a reward function.

3.2.1 VQA

The VQA challenge is a dataset of questions paired with questions. The challenge provides an image (byte array) separate from a question (string). One popular way to tackle this challenge is to parse the string with the state of art NLP model and parse the image with the state of art computer vision model. Although this modularizes the development of each model, it limits the amount of interconnection possible. The appropriate mental model when researching this problem is to imagine you open your eyes with to a scene, and you hear a question spoken to you. That appropriately conveys the idea that the image is received from a separate input mechanism than the question.[6] Given this setup, it is time consuming to create new data. New images are either realistic photographs collected over time or computer generated images generated very quickly but not realistic. By the very nature of VQA, and tasks like it, require models to be too passive. There is no interaction between agent and image. VQA is good method of testing an intelligent visual processor but not training one.

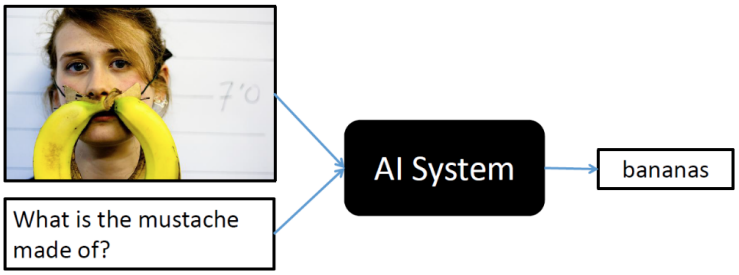

Figure 3-2: Image from the VQA challenge shows an example of how the VQA challenge is meant to be performed. The AI model that can answer this question must have already learned what a mustache is and then be able to visually identify that a banana is and understand how it is being used.

3.2.2 Analysis

VQA is a good first step into developing the multimedia models necessary to exist in our sensor rich world. However, the VQA task as it is currently defined side-steps what intelligence really is. A person who has never seen a mustache or banana would not be able to answer the above example question yet they would still be deemed intelligent. An agent shows more intelligence if when it does not know something can search for a solution.

Chapter 4

Framework Uses

4.1 Extending Reinforcement Learning

A new approach to reinforcement learning changes the workflow slightly in order to produce a more robust method to train an RL model. Instead of completing one task in one environment, there is the option to complete a large number of tasks using a local web environment. As an example I will show the workflow as it applies to VQA. The work flow proceeds with the following steps:

1. The agent is provided with a question and image, "What is the mustache made out of?"
2. The image and image content from the browser is generated.
3. An agent takes a 'legal' action (i.e. mouse click, or button press) based on the input image
4. The environment tracks movement and counts number of UI actions taken
5. A new image from the web is generated
6. An agent produces an answer, **if** incorrect: continue search. **else** go to next question
7. The score and the next browser image is passed to the RL agent in order to generate a new action.

To accomplish this, there are essentially two environments present. One environment, the resource environment, maintains the browser state and navigation score, while the

other environment, the task environment, maintains the agent's performance when completing an offline task. Some tasks include "Write a function in python that can print input text", "What is the price of an apple?", or "How long does it take to fly from Boston to Seattle?". One can imagine these questions requiring varying level of skills, yet all could be solved farily easily when provided with the appropriate resource. In this design, the resource environment points to a particular webpage that has supporting information pertaining to the task environment. There is a score for each environemnt being maintained. The resource score is calculated the same way regardless of what the information task is, while the task score will changed depending on the specific task. This two environment model lends itself to general knowledge based problems and can quickly grow in complexity.

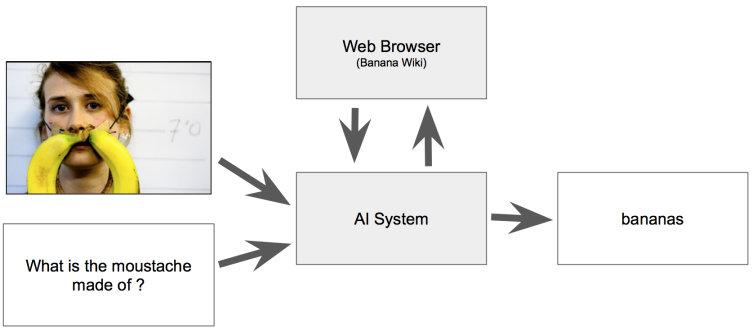

Figure 4-1: With an example of a slight modification to the VQA challenge now trains a model on its ability to learn a new task instead of measuring how much it knows.

4.2 Flexibility of Reinforcement Learning on the Browser

Placing the content on a series of webpage instead of as raw images and text turns the VQA task into another reinforcement learning task. The web browser can either produce an image or raw text and image using the <img> and <p> tags.

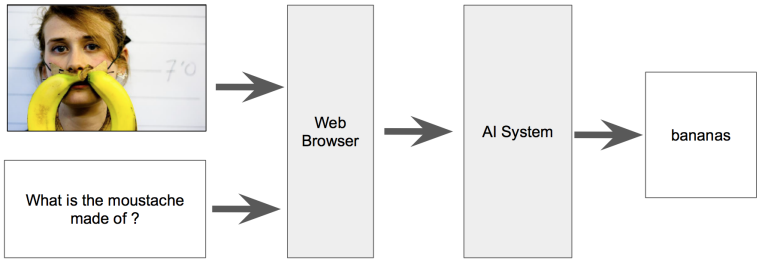

Figure 4-2: This diagram shows how integrating browser framework won't alter the reinforcement learning workflow.

4.3 Supporting New Tasks

With the additional control provided by the web browser, multimedia content can be easily controlled. This also introduces possibility of new areas of research. A new task, existing between computer vision and nlp parallels a more realistic scenario for machines in the real world. Whereas the model for nlp is to be listening to words and being able to parse sounds into letters and the inputting those characters into your mental model, this new tasks extends visual reading tasks. When we are presented with text, visually we see the entire page. We notice if it's a paragraph, a list or a caption. Each visual detail tells us important information which we can use to help decipher a document. Existing AI tasks including VQA do not effective test for this.

Web pages are especially designed for this sort of problem because web browser must know exactly where each element shows up in the page and its relative positioning to other elements. It must be able to distinguish an image caption from a button label. Training AI models should leverage the robust markup languages to create the plethora of information availble on the web.

Chapter 5

Environment

The resource environemnt is the part of the framework that integrates with existing OpenAI architecture. This module is in charge of breaking down the content provided on a webpage to modular compontents that can be fed directly into an AI agent. An example of a resource could be a wikipedia page, a cnn article, or python reference manual. As long as the content is represented using DOM objects instead of images or flash content, then the information can be parsed and provided to agents.

5.1 Implementation

The resource environment is a custom extension to OpenAI Gym. It can be instantiated following the standard syntax as defined in their API

```
env = ResourceEnvironment()
```

5.1.1 Chrome-CLI

To enable interactions with chrome, I first wrote a wrapper for the chrome command line interface in python, Chrome-CLI-PY [8]. The chrome-cli provides window and tab support but limited javascript console support. I provided handlers to support click and page scroll. ResourceEnvironment makes calls to Chrome-CLI-PY to interface AI models and the Google Chrome web browser. With these controls, the ResourceEnvironment can track the relative position and organization of all elements present on the page.

The key variables and methods in the Controller object are defined in the following way

VARIABLES

action_record Stores all the action values it receives including, click, scroll, and type

screen_dimensions Dimensions of the screen (monitor)

window_dimensions Dimensions of the window on the screen

element Stores the id of the last selected item

METHODS

click(x,y) Calls a click on the browser window at location **x,y**

scrollTo(x,y) Scrolls the browser window to location **x,y**

type(value) When an input element has been selected, it will fill the value

open(url) Opens a new tab at **url**

close() Closes the currently active tab

getScreenshot() Takes a screenshot of the browser and saves to the file screenshot.png

getSrc() Fetches the html src code from the currently active page

getItems(tag) Fetches all of the instances of a particular html tag type such as <a> or <p>

5.1.2 ChromeNavigator

The ChromeNavigator Environment is the object that follows the OpenAI specification and is the direct interface between a model and the chrome controller. To support the OpenAI spec, I first created the ChromeNavigator Environment by adding the resource to the Gym env directory. This allows for creation of the new environment by including gym and running

```
env = gym.make('ChromeNavigator-v0')
```

Within this env, the step, act, and reset functions were overridden to support the chrome actions.

VARIABLES

starting_url The url the controller will reset to

goal_url In the Navigator example, the target reference material

reward_terms A list of words and phrases that will increase the environment score when they appear

controller An instance of the chrome controller pointed at the starting_url

METHODS

Action is a tuple generated by the model. Like the name suggests, it represents the current action the model would like to make in the current browser. These actions are ("click",(x,y)), ("scroll",(x,y)), or ("type",(value)).

Step takes an action tuple as input and outputs a tuple containing (observation, reward, done, info). These are open ended variables that can be redefined as necessary. For the ChromeNavigator environment, observation is a dictionary which includes an image array representing the current page view in chrome, and the html source for the page. The current iteration of the framework parses the source for hyperlink tags. This parsing can be applied to any tag type. Reward is a float value representing how well the agent is using the interface. The primarily goal of a web browser is to train the agent to use the web properly which means that the agent must be scored by how well it finds a particular bit of information. The agent is scored by how few actions it takes and how similar the current url is to the destination url. In pseudocode the equation is $\alpha \frac{distance-between-strings(current-url,target-url)}{number-of-actions-taken}$ where α is a tunable parameter. Essentially, the sooner the model selects an accurate link to a webpage, the higher it scores. The done variable is a boolean that returns true when the browser has reached the target url, otherwise returns false. Info is a variable contains debugging information on the state of the environment useful for the researcher to know what is going on even if they cannot see the state of the webpage.

Reset is a function that is meant to return the environment back to its initial starting state. For the ChromeNavigator it returns the browser to the defined starting page, clearing the action list allowing the model to start with a clean slate.

5.2 Models

In order to test the framework, I implemented some common models used on reinforcement learning tasks.

5.2.1 Testing

To test the ChromeNavigator, I first passed an image of the browser window through a convolutional neural network. The model produced two outputs, one discrete variable that selected the action either click, scroll, or type; and one continuous variable that corresponded with a location on the screen if click or scroll were selected or from a list of letters. This effectively tested whether the agent could interact with the controller. Essentially the model would perform one action per turn and an action could either be a clicking a point on the window, scroll to a location, or typing a character from a character list.

5.2.2 Results

The selected model is likely not the ideal model for intelligence however it proved the framework was functional. Even a simple model could perform actions in the browser environment.

5.3 Analysis

5.3.1 Benefits

The ChromeNavigator is defined open-ended enough such that new metrics can be modified as necessary. The resource environment fits in well with the existing OpenAI architecture making it a part of a larger ecosystem of research and development. The framework leverages web technologies to make it extendable and easier to pick up. These reasons could facilitate the creation of new research tasks such as:

Navigable VQA Asking a question that requires the model to interact with the image in order to find the answer. Just like with VQA, in navigable VQA, a question is posed and an image is provided. The answer may or may not be present on the current image, but when the agent interacts with the image in a particular manner, produces a new image with the answer on it.

Common Web Tasks Comparative shop for an item across multiple websites, or search for the U.S. city with the largest population. Most of the popular website look and behave similarlly. This is a rich area for researchers to design agents that can interact with user interfaces.

Custom Research Tasks Design new interfaces to test an agent's proficiency at one tasks. Such as the current VQA task could be loaded as a series of webpages that a model could be trained on.

Automatically Generate Tasks Web language relies on ids and class tags, which can make simple tasks such as "click on the header" instantly scale from tens of webpages to thousands of webpages. Instead of having to define tasks by hand, adding a new taks type is a matter of defining the question by searching for a DOM element id which can then be applied across the entire micro-web.

5.3.2 Limitations

This environment is open ended but because it is new it is limited in scope. As more webapp modules are added to the micro-web the framework because more useful. Additionally, adding functionality such as video support would allow for a larger range of tasks that can be accomplished by this framework. The next step in AI research is to train agents to solve a large set of problems in a multimedia environment. In preparation of that, we need multimedia frameworks and datasets that can support the activities researchers will want to investigate.

Chapter 6

Micro-Web

This chapter explains the micro-web system and how it was created. Any agent that uses this framework should interact with the micro-web instead of training on live websites. In this way, an agent can be tested in sandbox which can produce reproducible test results. Furthermore any agent that works on the entire web, should first be able to successfully navigate the micro-web.

6.1 Implementation

There are two key directories included in the micro-web, the webpage modules, and the webapp modules.

6.1.1 Webpage Modules

In order to ensure reproducible results, the browser connects to a micro-web instead of actual web pages. A set of static, hyperlinked .html files from websites including wikipedia, mit news, and, open courseware are included. There are also a set of simple web app interfaces defined simply in html and javascript including a message board and search engine. These webpage samples were scraped from live websites using the Httrack command line tool. Approximately 20 250 mb website samples were downloaded. Within each domain the links are active and will allow an agent to navigate between webpages. If a link points to a resource outside of the domain, a file not found error will return.

6.1.2 Webapp Modules

There are two webapps provided in this version. The first is a search app which contains an index over all the webpage modules provided in the modules directory. The index can be recreated by running

```
python index_websites.py
```

(See Appendix A). The directory search-index contains the index-ing files. The second is a message board that allows agents to write and read messages from other users. There are no user accounts, however messags are stored in 'data.csv'. Both the search and message board applications are supported by server.py (See Appendix A).

Index_websites.py is implemented using the whoosh python library. Whoosh provides easy to use local file indexing. It is commonly used for creating indexes to documentation for large software projects and is capable of handling a corpus of html documents. Indexing files with whoosh requires defining a schema. For the first index used in this system, I defined three fields, FileName, FilePath, and Content. These values will then become searchable.

```
schema = Schema(FileName=TEXT(stored=True),
                FilePath=TEXT(stored=True),
                Content=TEXT(stored=True))
```

Server.py is implemented using python's web.py library. The script creates a local server at that accepts requests at 'http://0.0.0.0:8080/'. Searching is accomplished by placing a GET request with the id tag with a search term (i.e. http://0.0.0.0:8080/?id=MIT). The server opens the local database and finds documents containing the word MIT and returns them in a json encoded object. The message board can be interacted with the at the 'msg' route. To do this, send a POST request with a content and date tage (i.e. http://0.0.0.0:8080/msg?content=hello&date=5/10/2017). To fetch the list of messages saved in a local file, perform a GET request with no tags (i.e. http://0.0.0.0:8080/msg). A json object will be returned with the complete set of messages.

These implementations are small and non-scalable by design. With the scope of creating a realistic online environment, a given agent should only ever experience a relatively small amount of data.

6.1.3 Example Views

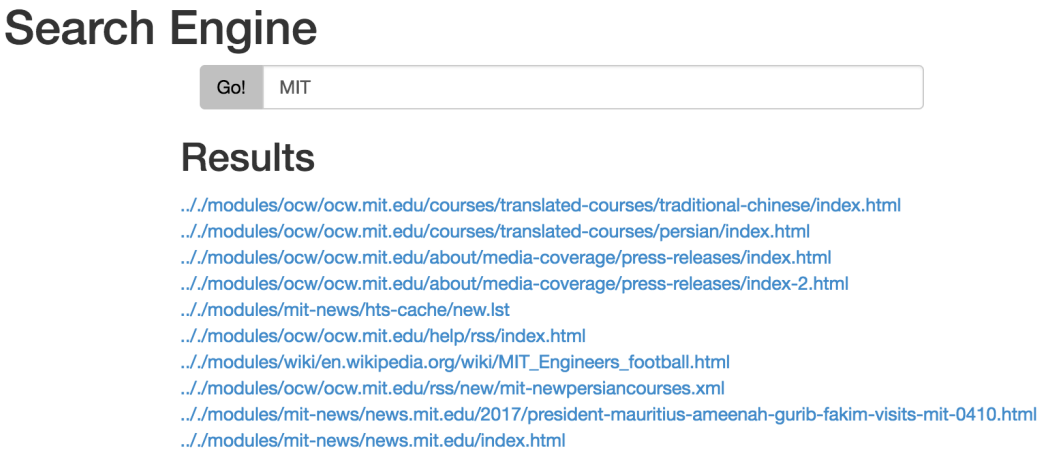

Figure 6-1: Basic search results showing accessible webpages. The links point to files in the local directory and do not require internet access in order to function

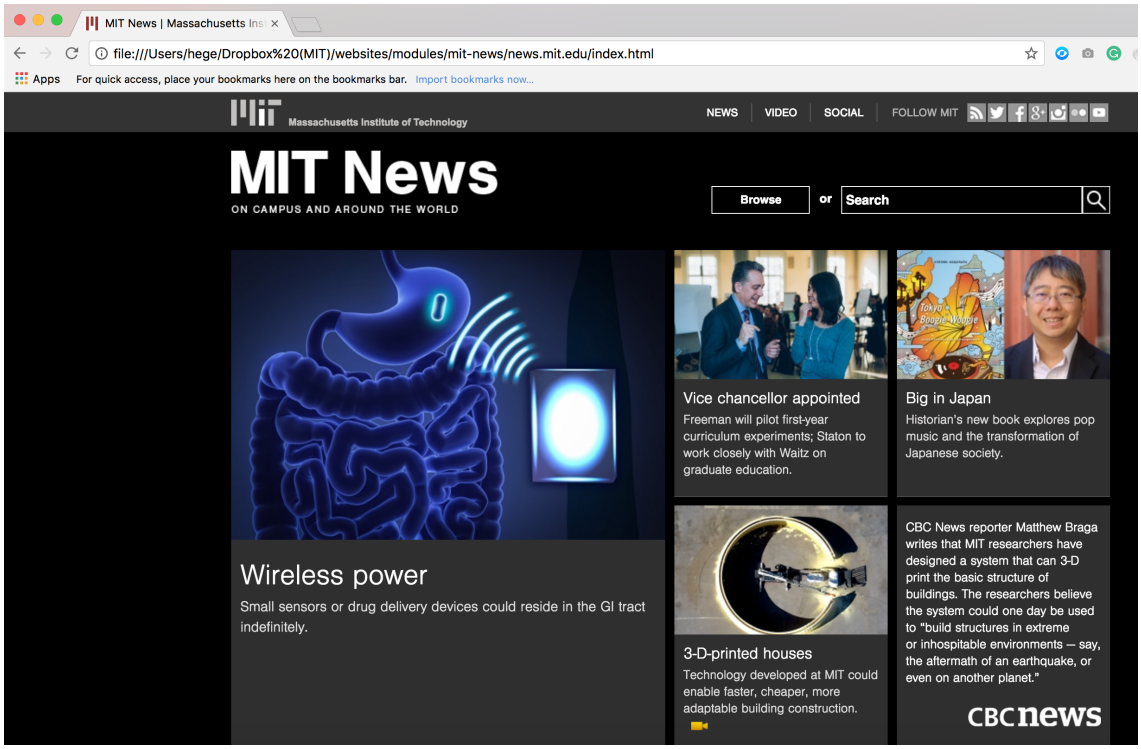

Figure 6-2: Selecting link allows for local navigation. An example of a webpage displayed locally.

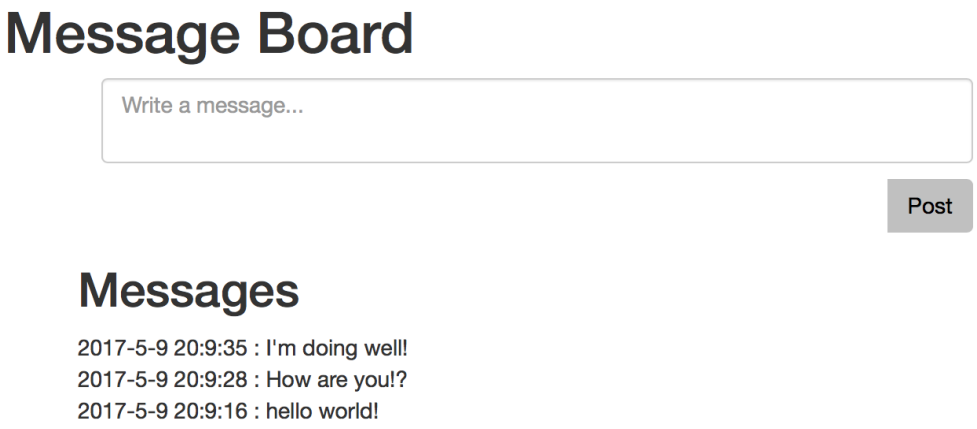

Figure 6-3: A snapshot of simple message written onto a messageboard. This includes basic information such as timestamp and message.

6.2 Analysis

The micro-web is a catalog of webpages and webapps in order for AI researchers to train and validate models on web tasks without having to create quality web pages by hand. Each module is 250 mb each which is 5gb for 20 websites. The index that allows the search engine to run over the microweb is also 5gb. This limit was set arbitrarily and may vary depending on what kind of tasks somoene wishes to perform.

6.2.1 Benefits

The catalaog of webpages are simple text-based webapges filled with rich information displayed in the same format that would be viewed online. While most AI tasks and frameworks must sacrifice realism for magnitude, this repository does not. The webpages are the same as how a human would view them.

6.2.2 Limitations

Because the content is scraped from websites, dynamic content beyond local moving gif files is not yet supported. Any content that would be supported by a server must be built in explicitly which takes additional time.

Chapter 7

Conclusion

7.1 Recommendations for Future Work

The framework requires many modules in order to appropriately simulate the world wide web. With a larger suite of applications, more skills can be learned before it is used in a live, online setting. Additionally the framework lacks video support. Although images on a web browser will be contained in the screenshot of the browser the framework doesn't refresh quick enough to support a video feed. Reading and writing the latest video frame to the file system would be particularly slow. One possible solution to this is to open a channel between chrome and the agent so it will see the video directly.

Currently most AI models are trained and validated on large server clusters and must be able to work on those server setups in order to be widely adopted.

7.2 Contribution

For this M.Eng thesis, I extended the popular OpenAI framework to allow reinforcement learning agents access to train and control a web browser as a resource to complete offline tasks. To support this, I designed a micro-web that provides offline access to a suite of webpages and webapps for training and validation. This environment is easily extendable through a set of webpage modules. I tested the framework with a convolutional neural network that converted a raw image input into actions that explored the browser environment. Lastly, I define new AI tasks and describe how they can lead to more intelligent machines.

Bibliography

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: visual question answering. *CoRR*, abs/1505.00468, 2015.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. http://arxiv.org/abs/1604.07316.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[4] cs231n. Cs231n: Convolutional neural networks for visual recognition., 2075. http://cs231n.github.io/convolutional-networks/.

[5] Adit Deshpande. A beginner's guide to understanding convolutional neural networks, 2017. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner

[6] Yash Goyal, Akrit Mohapatra, Devi Parikh, and Dhruv Batra. Interpreting visual question answering models. *CoRR*, abs/1608.08974, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Peter Rasmussen. Chrome-cli, 2015. https://github.com/prasmussen/chrome-cli.

[9] Tanmay Shankar, Santosha K. Dwivedy, and Prithwijit Guha. Reinforcement learning via recurrent convolutional neural networks. *CoRR*, abs/1701.02392, 2017.

[10] Ilya Sutskever Vicki Cheung, Jonas Schneider and Greg Brockman. Infrastructure for deep learning, 2017. https://blog.openai.com/infrastructure-for-deep-learning/.