CS534 - HW 5

Keith Chester

Due date: July 19th 2022

Problem 1

In this problem, we will discuss key concepts in reinformcenet learning as we've covered in lectures and reading.

Part 1

In this part we will discuss the differences between Model-based Reinforcement Learning (MBRL) and Model-free Reinforcement Learning (MFRL). First, let's clarify key differences between the two approaches.

- Model-based reinforcement learning learns a model of its environment over time and thus has an idea of the probability of the reward for certain actions prior to taking it. A chess playing agent, for instance, understands the probability of success of a given move prior to taking it. This is because the agent learns an understanding and representation of its environment during training.
- By contrast, Model-free reinforcement learning explores more, lives "more in the moment" by considering its actions at each step. While this mean that you may end up creating an agent the avoids optimal performance (chasing highest probability moves of a high reward from the start) it is more robust to interruptions. For example - a (poorly made) robot with reinforcement learning driving its behaviour; a MBRL agent would take maximum reward paths, which may mean in a dynamic environment running into a pedestrian. By contrast the MFRL agent would consider each individual moment/state, and could more easily deal with interruptions such as pesky humans.
- MBRL agents tend to learn quicker and result in better performance - they tend to isolate and train upon the key signals in an environment in which to improve their performance and resulting rewards. MFRL agents do train slower, but are better suited for chaotic dynamic environments or tasks that require ample exploration.

Let's look at some real world examples of each.

For MBRL, we can look at the very famous AlphaGo, an AI that trained to super-human capability at the game of Go and beat the world's reiging champion handidly. This agent combined Monte-Carlo Tree Search and a model based reinforcement learning agent where it learned to explore moves. Outside the opening database and tree search utility, it was free to learn the rules of the game itself. Eventually it built a super model that would determine moves with the highest probability of success. (Link: `https://www.deepmind.com/research/highlighted-research/alphago`)

For MFRL, we look towards more modern games - specifically success with Atari games. With advancements in computer vision and deep learning thanks to the introduction of the convolutional neural network, we've begun to see agents that can take visual input and directly derive actions from a more literal "obsevation" of the environment than some reduced representation of the environment. These agents can directly see all the pixels that a human player would see, and be fed directly a win/loss or video game score based on its performance. Here the agent must explore and react to its environment at the moment in time. While it is unable to fully predict what might be thrown at it, it will react to it with skill. These agents have been used to demonstrate that a singular model architecture and training approach can learn many games. (Link: `https://www.endtoend.ai/envs/gym/atari/`)

Part 2

Now we'll discuss the differences between Passive Reinforcement Learning (PRL) and Active Reinforcement Learning (ARL). We'll again begin by clarifying some key differences between the two approaches.

- In PRL, the agent is told what to do by its policy - hence *passive*. The agent here is ultimately aiming to calculate the expected utility reward for a given state s.
- In contrast, an active agent is aiming to learn the optimal policy, and thus must decide what to do at any given state s. To do this, it must balance approaching policies for exploitation (greedy) or exploration (often random, but could lead to improvement)
- Changing the approach of a passive agent to include possible exploration can change a passive agent to an active one.

Active reinforcement learning is seen in Q-Learning - as for an example tihs repository of work I've completed in the past: `https://github.com/hlfshell/deep_q_network` - this project utilized a generic deep q architecture to solve numerous game-like environments, including direct image input through a convolutional neural network (CNN).

For passive learning, an example would be an reinforcement learning powered vehicle. If the vehicle gets into a position where its optimal policy says to do a certain task (albeit with extra validation for safety to avoid incidents such as ramming pedestrians), it would be unsafe for an actual car in a public environment to decide to instead ignore the best decision and actively explore another option. This would result in vehicles occasionally veering into traffic, taking wrong turns, or other equally disasterous events.

Problem 2

Given the following simple documents:

- Document 1 - "The cat chased a rat"
- Document 2 - "A big rat chased the big dog"

...generate the two document vectors by using several methods.

Bag of words

Document	the	cat	chased	a	rat	big	dog
Document 1	1	1	1	1	1	0	0
Document 2	1	0	1	1	1	1	1

Bag of 2-grams

Document	the cat	cat chased	chased a	a rat	a big	big rat	rat chased	chased the	the big	big dog
Document 1	1	1	1	1	0	0	0	0	0	0
Document 2	0	0	0	0	1	1	1	1	1	1

Bag of 3-grams

Doc	the cat chased	cat chased a	chased a rat	a big rat	big rat chased	chased the big	the big dog
Doc 1	1	1	1	0	0	0	0
Doc 2	0	0	0	1	1	1	1

TF-IDF

...where we will be using the following equation:

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{df_i} \tag{1}$$

...where tfi, j is the number of occurences of i in j, df_i is the number of documents containing i, and N is the total number of documents.

Document	the	cat	chased	a	rat	big	dog
Document 1	0.0	0.060	0.0	0.0	0.0	0.0	0.0
Document 2	0.0	0.0	0.0	0.0	0.0	0.086	0.043

...with the only non-zero calculations being for cat, big, and dog, respectively:

$$cat \rightarrow \frac{1}{5} \times \log \frac{2}{1} = 0.060 \tag{2}$$

$$big \rightarrow \frac{2}{7} \times \log \frac{2}{1} = 0.086 \tag{3}$$

$$dog \rightarrow \frac{1}{7} \times \log \frac{2}{1} = 0.043 \tag{4}$$

Part A

Now we aim to compute the cosine of similarity using each of the above text vectorization methods. To do this, we are solving the equation of:

$$similarity(doc_1, doc_2) = \cos(\theta) = \frac{doc_1 doc_2}{|doc_1||doc_2|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{5}$$

First we shall look at bag of words:

$$A \cdot B = \sum_{i=1}^{n} A_i B_i = (1*1) + (1*0) + (1*1) + (1*1) + (1*1) + (0*1) + (0*1) = 4 \tag{6}$$

$$\sqrt{\sum_{i=1}^{n} A_i^2} = \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 0^2} = \sqrt{5} = 2.24 \tag{7}$$

$$\sqrt{\sum_{i=1}^{n} B_i^2} = \sqrt{1^2 + 0 + 1^2 + 1^2 + 1^2 + 2^2 + 1^2} = \sqrt{9} = 3 \tag{8}$$

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} = \frac{4}{2.24 \cdot 3} = 0.60 \tag{9}$$

Then we move onto the cosine of similarity for our 2-gram approach.

$$A \cdot B = \sum_{i=1}^{n} A_i B_i = (1*0) + (1*0) + (1*0) + (1*0) + (1*0) + (0*1) + (0*1) + (0*1) + (0*1) + (0*1) = 0 \tag{10}$$

Since we know the numerator will be 0, we can stop here - the similarity will be 0. Moving onto our 3-gram:

$$A \cdot B = \sum_{i=1}^{n} A_i B_i = (1*0) + (1*0) + (1*0) + (0*1) + (0*1) + (0*1) + (0*1) = 0 \tag{11}$$

Once again we see no similarity due to lack of overlap from the 3-gram approach.
Finally we shall look at cosine of similarity for our TF-IDF vectorization:

$$A \cdot B = \sum_{i=1}^{n} A_i B_i = (0.0*0.0) + (0.060*0.0) + (0.0*0.0) + (0.0*0.0) + (0.0*0.0) + (0.0*0.086) + (0.0*0.043) = 0.0 \tag{12}$$

...and again, with a 0 numerator, we are going to have a cosine of similarity of 0.

Part B

Now, utilizing what we learn from above, we're going to use SKLearn to create a Python program to analyse the following documents:

- "Mr Jeremy put on a macintosh, and a pair of shiny shoes; he took his fishing rod and basket, and set off with enormous hops to the place where he kept his boat. The boat was round and green, and very like the other lily-leaves. It was tied to a water-plant in the middle of the pond."
- "Peter never stopped running or looked behind him till he got home to the big fir-tree. He was so tired that he flopped down upon the nice soft sand on the floor of the rabbit-hole and shut his eyes. His mother was busy cooking; she wondered what he had done with his clothes. It was the second little jacket and pair of shoes that Peter had lost in a week!"

With these documents, we will:

- Pre-process the documents and then display the cleaned documents respectively on the output
- Generate the two TF-IDF vectors respectively and then display the two TF-IDF vectors on the output
- Compute their cosine similarity and display the value on the output

...and finally discuss whether or not these documents are similar from those findings.

To see the code, please reference **problem_2.py**. There we use Scikit Learn to calculate the TF-IDF vectors and cosine of similarity. With a cosine of similarity of 0.44, these documents are somewhat similar.

Problem 3

In Problem 3, we attempted to utilize PyTorch to create an RNN and LSTM neural network trained on a Kaggle provided dataset of Amazon reviews to find positive and negative sentiments.

For this, we created *build_vocab.py*, which goes through the reviews and builds an embedded vocab of all the words utilized throughout the reviews. These embeddings are then utilized as index referencing a specific word.

We then established the longest length of a review to act as our tensor size - any review that was shorter would be padded with 0's (no word) to be of equivalent size.

From this we built out an RNN and LSTM model in problem_3_RNN.py and problem_3_LSTM.py, respectively.

Unfortunately, an issue plagued me for hours - no matter what I did, my linear layers would always convert the network's output into the same values, resulting in a network that would never train. As such, I am submitting the work I have due to time constraints.