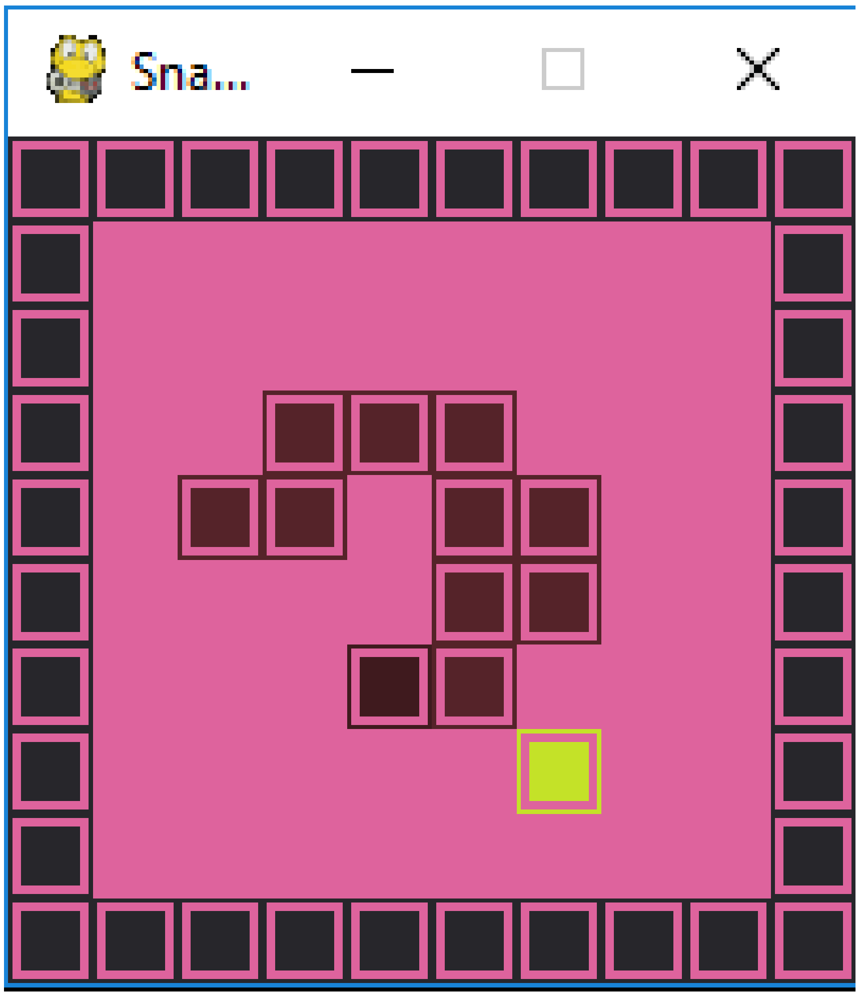
CS 256 : Topics in Artificial Intelligence

Final Project

Game Automation using DQN: Snake



Team:

Rashmeet Kaur Khanuja (012409982)

Yuvraj Singh Kanwar (012005175)

Introduction

This report is presented as a requirement for the CS 256 Topics in Artificial Intelligence course. With this project we aim to automate the game of snake by utilizing the power of Deep Q-Networks. The goal of our project is to train an AI agent to learn the snake game and perform well in the environment. Snake game was first developed as an arcade game, called Blockade, by Gremlin in 1976. It was programmed for personal computer in 1978, and since then various versions of similar games were developed. [1] Traditionally, the game is supposed to played by a human player. The Deep Q-Network has become a benchmark and building point for most of the deep reinforcement learning research. By combining reinforcement learning and convolutional neural network, a Snake game agent can be trained to play the game. [2]

Video games are challenging but easy to formalize, which is why they have been a keen area of interest for Artificial Intelligence Researchers. Game developers have tried to create AI agents for decades. Main ambition is to build the AI player that can learn the game from scratch, rather than merely following one fixed strategy, based on its gaming experience. Reinforcement learning is one of the most intriguing technology in Machine Learning, which learns the optimal action by trial and error, and shall be used for what we are trying to develop. [1]

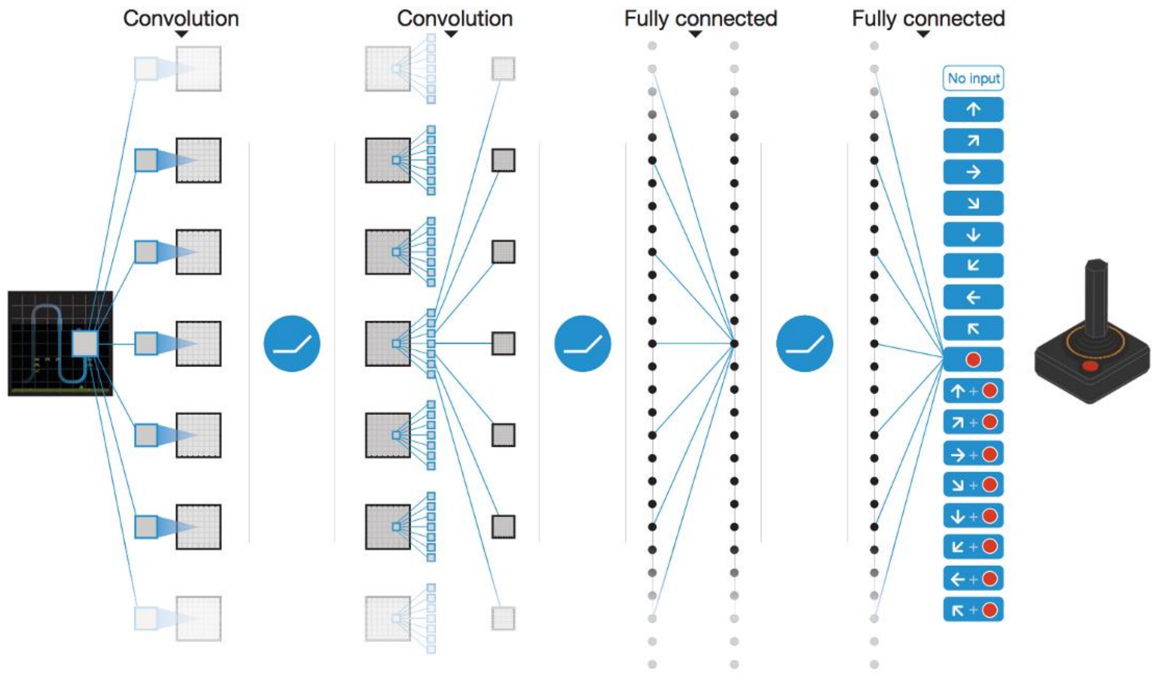
Game of Snake

Snake is a famous game all over the world, popularized with the Nokia cell phones. Traditionally, Snake is played by a human agent who controls moving direction of a snake and tries to eat items by running into them with the head of the snake, while the fruits would emerge in random place of bounded board. [2] The player plays the game using only the left and right arrow keys which will make the snake turn relative to the current direction. The snake keeps on moving forward automatically and it will increase its length and speed every time it reaches the fruit which makes it difficult to achieve a high score. The final goal of the game is to make the snake eat as many fruits as possible without running out of the board or making the snake bite itself. When this happens, the game ends and a final score is returned. [1]

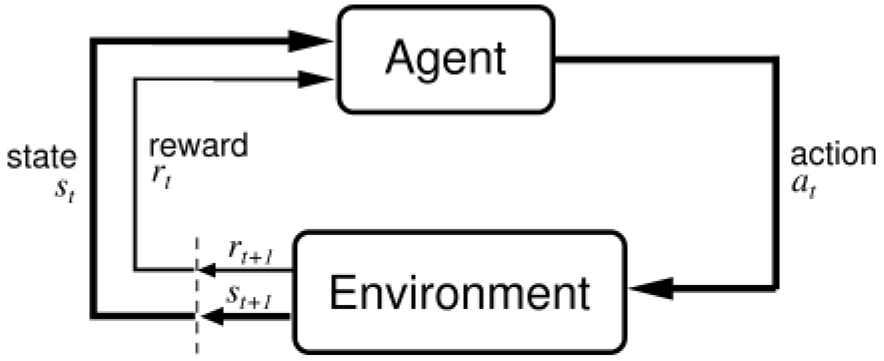
Deep Q-Networks

Deep Q-Learning can be broken into 2 parts - Deep Neural Networks and Q-Learning. We shall look at each and how they contribute towards our snake game automation.

Deep Learning refers to using Neural Networks with more than one hidden layers to maximize the learning output. Here, we have used Convolutional Neural Networks (CNN) because we are dealing with images. CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the functions that apply for neural networks still apply on CNNs. The difference being Neural Networks take as input a vector while CNNs take input a multi-channeled image (in our case, 2 channels).



Before diving into Q-Learning, we take a look at what reinforcement learning is. Reinforcement learning is about learning how to behave on some environment where our only feedback is some sparse and time delayed "labels" called rewards. They are time delayed because there are cases where the environment will only tell if your action was good or bad sometime after you actually moved. In this type of learning, we have states, actions, and rewards. Our agent performs actions to reach to the next state, if that state is a desirable one (like eating a fruit, or not dying!), it gets positive rewards that it would ultimately try to maximize. If the state is an undesirable one (like the snake hitting a wall or itself), it gets negative rewards. This process goes on and on in order to make our agent learn the best possible moves at each state in order to maximize the rewards.



Reinforcement learning solves the difficult problem of correlating immediate actions with the delayed returns they produce. Like humans, reinforcement learning algorithms sometimes have to wait a while to see the fruit of their decisions. They operate in a delayed return environment, where it can be difficult to understand which action leads to which outcome over many time steps.

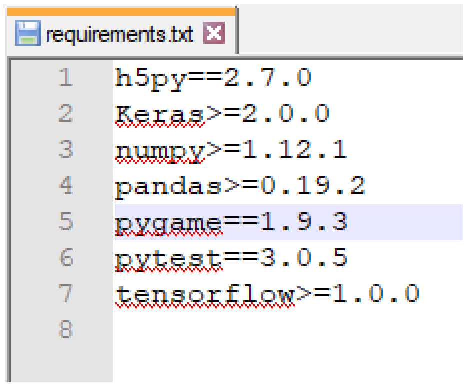
Moving to Q-Learning, it is an example of a reinforcement learning technique used to train an agent to develop an optimal strategy for solving a task, in which an agent tries to learn the optimal policy from its history of interaction with the environment, and we call the agent's knowledge base as ”Q-Factor”. This history of interactions is stored in the form of state-action-rewards such as - ⟨ s0,a0,r1,s1,a1,r2,s2,a2,r3,s3,a3,r4,s4...⟩ , which means that the agent was in state s0 and did action a0, which resulted in it receiving reward r1 and being in state s1; then it did action a1, received reward r2, and ended up in state s2; then it did action a2, received reward r3, and ended up in state s3; and so on. So in Q-learning we start by setting all the state-action values to 0 or a random value, and we go around and explore the state-action space. After the agent tries an action in a state, we evaluate the state that it has lead to. If it has lead to an undesirable outcome we reduce the Q value (or weight) of that action from that state so that other actions will have a greater value and be chosen instead the next time the agent is in that state. Similarly, if the agent is rewarded for taking a particular action, the weight of that action for that state is increased, so it's more likely to choose it again the next time it is in that state. Importantly, when we update Q, we're updating it for the previous state-action combination. The difference here being rather than assigning q-values on the basis of the end-rewards, we rather do it at each step in order to assign predictive actions based on immediate rewards. Q values are updated as follows -

$$Q(s,a) = r + \gamma(max(Q(s',a'))$$

Where s is the current state, 'a' represents the action the agent takes from the current state. 'S' ' represents the state resulting from the action. 'r' is the reward you get for taking the action and 'γ' is the discount factor. So, the Q value for the state 'S' taking the action 'a' is the sum of the instant reward and the discounted future reward (value of the resulting state). The discount factor 'γ' determines how much importance you want to give to future rewards.

Over the years, deep reinforcement learning has gained much admiration as it has been demonstrated to perform better than previous reinforcement learning methods on domains with very large state-spaces. Earliest deep Q-Network learning papers written by Mnih et al. [2015] demonstrated a method for learning to play Atari 2600 video games, using the Arcade Learning Environment (ALE) [Bellemare et al., 2013], from performance data and image data using the same deep neural network architecture and hyper-parameters for all the games. DQN has outperformed previous reinforcement learning methods on nearly all of the games and recorded better than human performance on most. As many researchers tackle reinforcement learning problems with deep reinforcement learning methods and propose alternative algorithms, the results of the DQN paper are often used as a benchmark to show improvement. Thus, implementing the DQN algorithm is important for both replicating the results of the DQN paper for comparison and also building off the original algorithm. One of the main contributions of the deep Q-Network learning papers was finding ways to improve stability in their artificial neural networks during training. [3]

Technologies Used



```
h5py==2.7.0
Keras>=2.0.0
numpy>=1.12.1
pandas>=0.19.2
pygame==1.9.3
pytest==3.0.5
tensorflow>=1.0.0
```

1. H5py v2.7.0: The h5py package is a Pythonic interface to the HDF5 binary data format. It lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. For example, you can slice into multi-terabyte datasets stored on disk, as if they were real NumPy arrays. Thousands of datasets can be stored in a single file, categorized and tagged however you want. [4]
2. Keras v2.0.0: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Keras allows for easy and fast prototyping (through user friendliness, modularity, and extensibility). It supports convolutional networks and recurrent networks, as well as combinations of the two. [5]
3. Numpy v1.12.1: NumPy is the fundamental package for scientific computing with Python. It contains among other things:
    a. a powerful N-dimensional array object
    b. sophisticated (broadcasting) functions
    c. tools for integrating C/C++ and Fortran code
    d. useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. NumPy is licensed under the BSD license, enabling reuse with few restrictions. [6]

4. Pandas v0.19.2: Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries. [7]
5. Pygame v1.9.3: pygame (the library) is a Free and Open Source python programming language library for making multimedia applications like games built on top of the excellent SDL library. Like SDL, pygame is highly portable and runs on nearly every platform and operating system. [8]
6. Tensorflow v1.0.0: Deep reinforcement learning requires updating large numbers of gradients, and deep learning tools such as TensorFlow are extremely useful for calculating these gradients. Deep reinforcement learning also requires visual states to be represented abstractly, and for this, convolutional neural networks work best. TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support

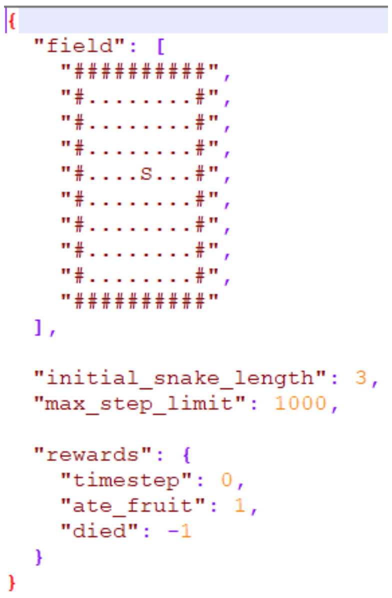
for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. [9]

Snake Game Environment

A revised version of snake game is developed using Pygame.

Levels

1. 10x10-blank : A Blank Snake game environment where snake can move freely inside a 10x10 block. Snake dies when it hits the wall or when it hits onto itself. Snake starts with initial length of 3 cell.

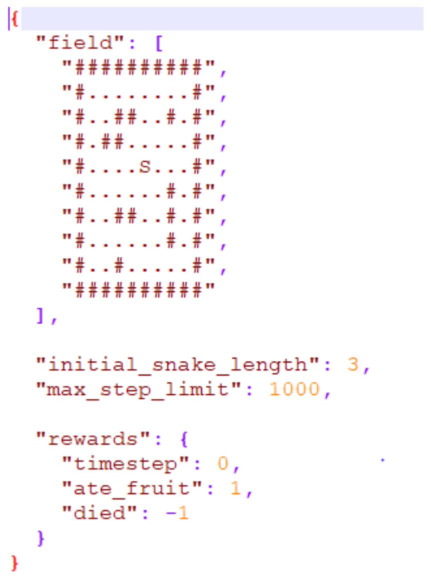


```
{
  "field": [
    "##########",
    "#........#",
    "#........#",
    "#........#",
    "#....S...#",
    "#........#",
    "#........#",
    "#........#",
    "#........#",
    "##########"
  ],

  "initial_snake_length": 3,
  "max_step_limit": 1000,

  "rewards": {
    "timestep": 0,
    "ate_fruit": 1,
    "died": -1
  }
}
```

2. 10x10-obstacle: A Snake game environment where snake has to dodge the obstacles inside a 10x10 block. Snake dies when it hits the wall, hits the obstacle or when it hits onto itself. Snake starts with initial length of 3 cell.



```
{
  "field": [
    "##########",
    "#........#",
    "#..##..#.#",
    "#.##.....#",
    "#....S...#",
    "#......#.#",
    "#..##..#.#",
    "#......#.#",
    "#..#.....#",
    "##########"
  ],

  "initial_snake_length": 3,
  "max_step_limit": 1000,

  "rewards": {
    "timestep": 0,
    "ate_fruit": 1,
    "died": -1
  }
}
```

Entities of the Environment of Snake Game:

1. Point: Class that represent 2D point with named axes
2. CellType: Define types of cells like empty, fruit, snake head, snake body, wall etc. that can be found in the game
3. SnakeDirection: Defines possible directions(i.e. North, South, east or West) the snake can take and the corresponding offsets
4. SnakeAction: Maintain Direction, turn left or turn right.
5. Snake: Snake that has a position, can move, and change directions
6. Field: Playing field for the game

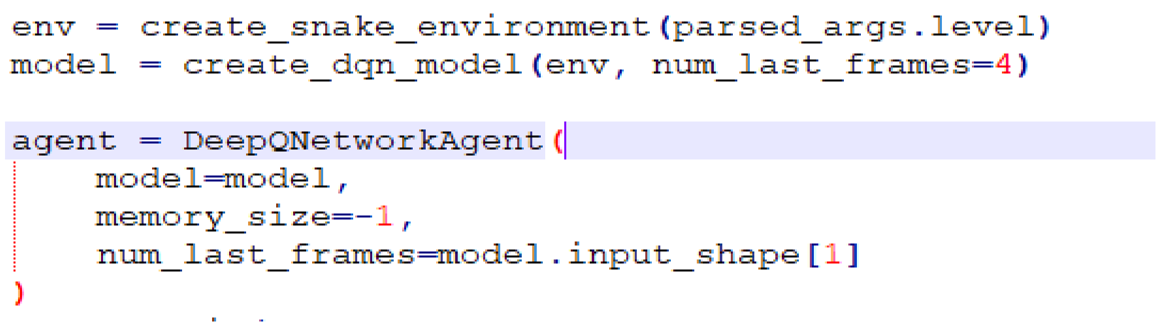
```
class Environment(object):
    def __init__(self, config, verbose=1):

        self.field = Field(level_map=config['field'])
        self.snake = None
        self.fruit = None
        self.initial_snake_length = config['initial_snake_length']
        self.rewards = config['rewards']
        self.max_step_limit = config.get('max_step_limit', 1000)
        self.is_game_over = False

        self.timestep_index = 0
        self.current_action = None
        self.stats = EpisodeStatistics()
        self.verbose = verbose
        self.debug_file = None
        self.stats_file = None
```

Agents

1. DeepQNetworkAgent: Snake agent that is powered by DQN with experience replay. Need to train the agent to make the snake perform well in the snake environment and define functions for DQN agent to act upon. Some arguments required for training the DQN agent
    a. Env: an instance of Snake environment.
    b. num_episodes (int): the number of episodes to run during the training.
    c. batch_size (int): the size of the learning sample for experience replay.
    d. discount_factor (float): discount factor (gamma) for computing the value function.
    e. checkpoint_freq (int): the number of episodes after which a new model checkpoint will be created.
    f. exploration_range (tuple): A (max, min) range specifying how the exploration rate should decay over time.
    g. exploration_phase_size (float): the percentage of the training process at which the exploration rate should reach its minimum.



```
env = create_snake_environment(parsed_args.level)
model = create_dqn_model(env, num_last_frames=4)

agent = DeepQNetworkAgent(
    model=model,
    memory_size=-1,
    num_last_frames=model.input_shape[1]
)
```

2. RandomActionAgent: A Snake agent that takes a random action at every step to implement Reinforcement Learning.
3. Human: An agent that is operated by a human

Classes:

1. PyGameGUI: Contains Environment and GUI for the Snake Game. Some vital methods defined in this class are:
    a. Load_environment
    b. Load_agent
    c. Render_Cell
    d. Map_key_to_snake_action
    e. Run_Episode
2. Stopwatch: Class created to measure time elapsed since last checkpoint. Defines functions to fetch time since the last checkpoint or reset a new checkpoint.
3. Colors: Contains definitions for screen background and cell types like wall, snake head, snake body, fruit, etc.
4. Environment: Represents the RL environment for the Snake game that implements the game logic, provides rewards for the agent and keeps track of game statistics.Some vital methods defined in this class are:
    a. Seed: Initialize the random state of the environment to make results reproducible.
    b. Observation_shape: Get the shape of the state observed at each timestep.

    c. Num_actions: Get the number of actions the agent can take
    d. New_episode: Reset the environment and begin a new episode
    e. Record_timestep_stats: Record environment statistics according to the verbosity level
    f. Get_observation : Observe the state of the environment
    g. Choose_action: Choose the action that will be taken at the next timestep
    h. Timestep: Execute the timestep and return the new observable state
    i. Generate_fruit: Generate a new Fruit
    j. Has_hit_wall
    k. Has_hit_own_body
    l. is_alive
5. TimestepResult: Object of this class represents the information provided to the agent after each timestep
6. EpisodeStatistics: Object of this class represents the summary of the agent's performance during the episode

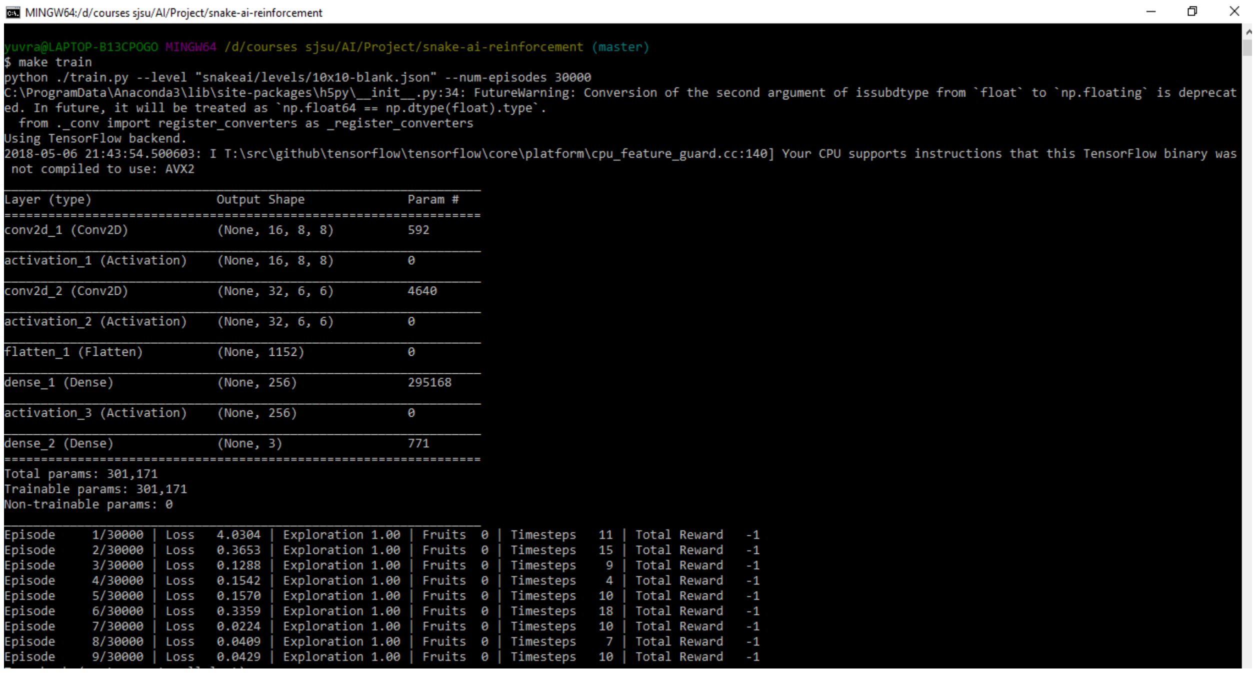
Training Data

Training of data is done by creating a DQN model using Neural Networks Library of Tensorflow. Training model makes use of Convolutions. We train the model from scratch by dividing our game into episodic tasks and letting the model learn over each episode. Training model is added with arbitrary filters that can mix channels together using Conv2d. Conv2d computes a 2-D convolution given 4-D input and filter tensors.

Activation Functions: The activation ops provide different types of nonlinearities for use in neural networks. These include smooth nonlinearities (sigmoid, tanh, elu, selu, softplus, and softsign), continuous but not everywhere differentiable functions (relu, relu6, crelu and relu_x), and random regularization (dropout). For Snake game, which has a function which is continuous but not differentiable everywhere, we have used Rectified Linear Unit or reLu activation function that helps bring sparsity and a reduced likelihood of vanishing gradient.

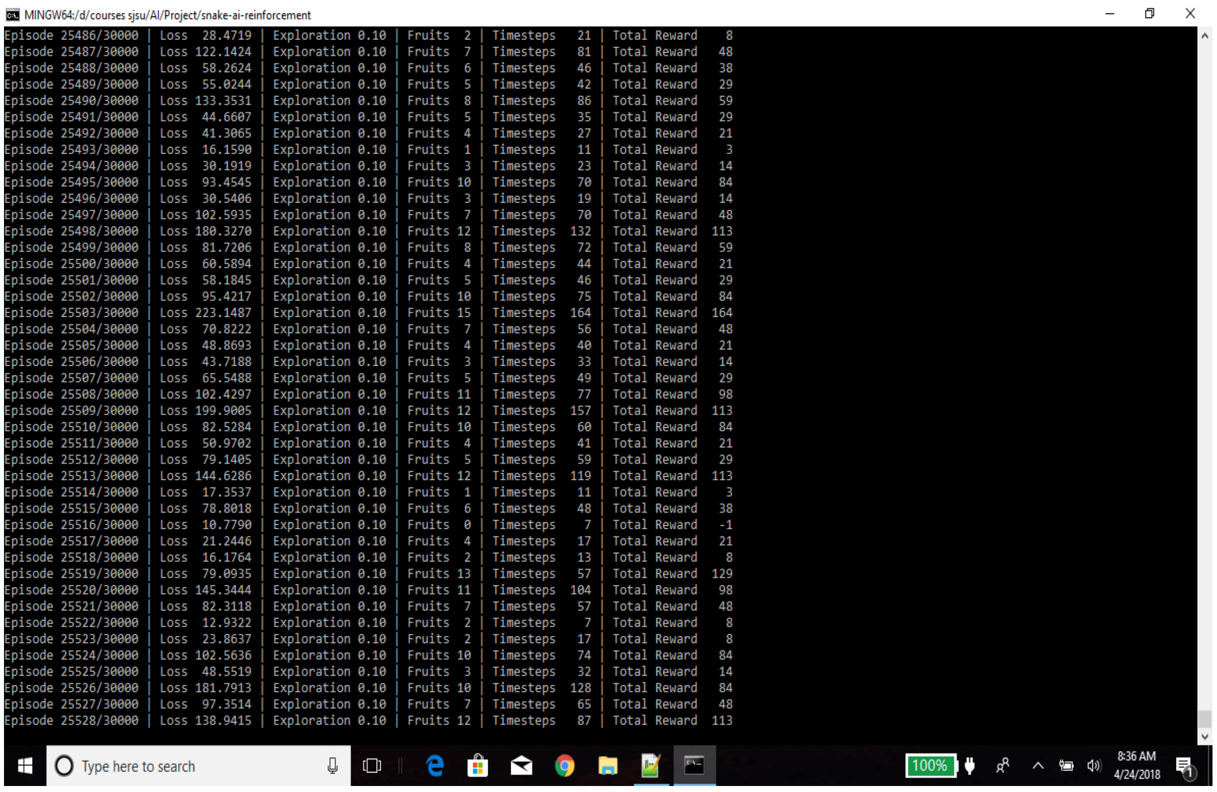
The definition of a ReLU is h=max(0,a)h=max(0,a) where a=Wx+ba=Wx+b. One major benefit is the reduced likelihood of the gradient to vanish. This arises when a>0a>0. In this regime the gradient has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLUs results in faster learning. The other benefit of ReLUs is sparsity. Sparsity arises when a≤0a≤0. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.

No. of Episodes used for training the data was 30000 for blank levels and obstacle levels each. It took around 25 hours to train the data.



```
yuvra@LAPTOP-B13CPOGO MINGW64 /d/courses sjsu/AI/Project/snake-ai-reinforcement (master)
$ make train
python ./train.py --level "snakeai/levels/10x10-blank.json" --num-episodes 30000
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecat
ed. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2018-05-06 21:43:54.500603: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was
 not compiled to use: AVX2
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 16, 8, 8)          592
_________________________________________________________________
activation_1 (Activation)    (None, 16, 8, 8)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 6, 6)          4640
_________________________________________________________________
activation_2 (Activation)    (None, 32, 6, 6)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               295168
_________________________________________________________________
activation_3 (Activation)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771
=================================================================
Total params: 301,171
Trainable params: 301,171
Non-trainable params: 0
_________________________________________________________________
Episode     1/30000 | Loss   4.0304 | Exploration 1.00 | Fruits  0 | Timesteps   11 | Total Reward   -1
Episode     2/30000 | Loss   0.3653 | Exploration 1.00 | Fruits  0 | Timesteps   15 | Total Reward   -1
Episode     3/30000 | Loss   0.1288 | Exploration 1.00 | Fruits  0 | Timesteps    9 | Total Reward   -1
Episode     4/30000 | Loss   0.1542 | Exploration 1.00 | Fruits  0 | Timesteps    4 | Total Reward   -1
Episode     5/30000 | Loss   0.1570 | Exploration 1.00 | Fruits  0 | Timesteps   10 | Total Reward   -1
Episode     6/30000 | Loss   0.3359 | Exploration 1.00 | Fruits  0 | Timesteps   18 | Total Reward   -1
Episode     7/30000 | Loss   0.0224 | Exploration 1.00 | Fruits  0 | Timesteps   10 | Total Reward   -1
Episode     8/30000 | Loss   0.0409 | Exploration 1.00 | Fruits  0 | Timesteps    7 | Total Reward   -1
Episode     9/30000 | Loss   0.0429 | Exploration 1.00 | Fruits  0 | Timesteps   10 | Total Reward   -1
```

```python
def create_dqn_model(env, num_last_frames):
    #Build a new DQN model to be used for training
    #Args:
    #    env: an instance of Snake environment
    #    num_last_frames: the number of last frames the agent considers as state
    #Returns: A compiled DQN model
    model = Sequential()
    # Convolutions.
    model.add(Conv2D(
        16,
        kernel_size=(3, 3),
        strides=(1, 1),
        data_format='channels_first',
        input_shape=(num_last_frames, ) + env.observation_shape
    ))
    model.add(Activation('relu'))
    model.add(Conv2D(
        32,
        kernel_size=(3, 3),
        strides=(1, 1),
        data_format='channels_first'
    ))
    model.add(Activation('relu'))
    # Dense layers.
    model.add(Flatten())
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(Dense(env.num_actions))
    model.summary()
    model.compile(RMSprop(), 'MSE')
    return model
```

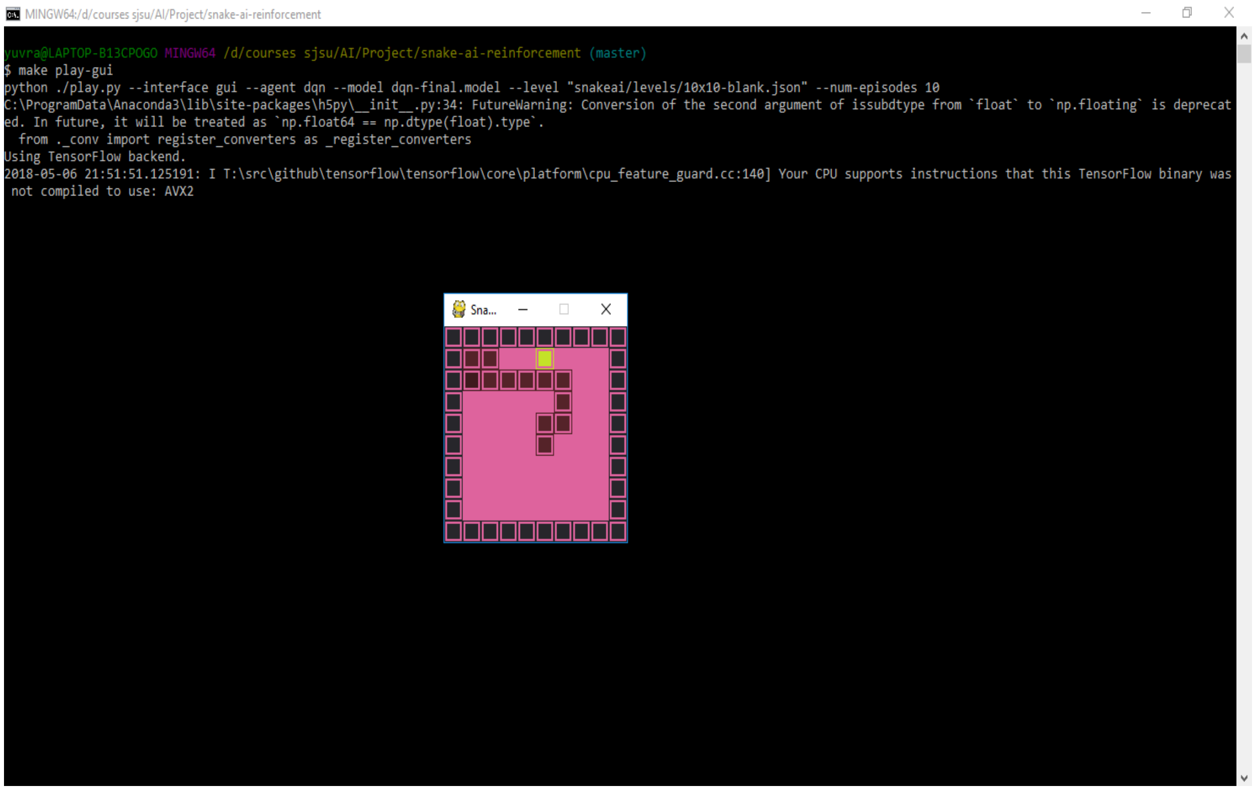


```
Episode 25486/30000 | Loss  28.4719 | Exploration 0.10 | Fruits  2 | Timesteps   21 | Total Reward    8
Episode 25487/30000 | Loss 122.1424 | Exploration 0.10 | Fruits  7 | Timesteps   81 | Total Reward   48
Episode 25488/30000 | Loss  58.2624 | Exploration 0.10 | Fruits  6 | Timesteps   46 | Total Reward   38
Episode 25489/30000 | Loss  55.0244 | Exploration 0.10 | Fruits  5 | Timesteps   42 | Total Reward   29
Episode 25490/30000 | Loss 133.3531 | Exploration 0.10 | Fruits  8 | Timesteps   86 | Total Reward   59
Episode 25491/30000 | Loss  44.6607 | Exploration 0.10 | Fruits  5 | Timesteps   35 | Total Reward   29
Episode 25492/30000 | Loss  41.3065 | Exploration 0.10 | Fruits  4 | Timesteps   27 | Total Reward   21
Episode 25493/30000 | Loss  16.1590 | Exploration 0.10 | Fruits  1 | Timesteps   11 | Total Reward    3
Episode 25494/30000 | Loss  30.1919 | Exploration 0.10 | Fruits  3 | Timesteps   23 | Total Reward   14
Episode 25495/30000 | Loss  93.4545 | Exploration 0.10 | Fruits 10 | Timesteps   70 | Total Reward   84
Episode 25496/30000 | Loss  30.5406 | Exploration 0.10 | Fruits  3 | Timesteps   19 | Total Reward   14
Episode 25497/30000 | Loss 102.5935 | Exploration 0.10 | Fruits  7 | Timesteps   70 | Total Reward   48
Episode 25498/30000 | Loss 180.3270 | Exploration 0.10 | Fruits 12 | Timesteps  132 | Total Reward  113
Episode 25499/30000 | Loss  81.7206 | Exploration 0.10 | Fruits  8 | Timesteps   72 | Total Reward   59
Episode 25500/30000 | Loss  60.5894 | Exploration 0.10 | Fruits  4 | Timesteps   44 | Total Reward   21
Episode 25501/30000 | Loss  58.1845 | Exploration 0.10 | Fruits  5 | Timesteps   46 | Total Reward   29
Episode 25502/30000 | Loss  95.4217 | Exploration 0.10 | Fruits 10 | Timesteps   75 | Total Reward   84
Episode 25503/30000 | Loss 223.1487 | Exploration 0.10 | Fruits 15 | Timesteps  164 | Total Reward  164
Episode 25504/30000 | Loss  70.8222 | Exploration 0.10 | Fruits  7 | Timesteps   56 | Total Reward   48
Episode 25505/30000 | Loss  48.8693 | Exploration 0.10 | Fruits  4 | Timesteps   40 | Total Reward   21
Episode 25506/30000 | Loss  43.7188 | Exploration 0.10 | Fruits  3 | Timesteps   33 | Total Reward   14
Episode 25507/30000 | Loss  65.5488 | Exploration 0.10 | Fruits  5 | Timesteps   49 | Total Reward   29
Episode 25508/30000 | Loss 102.4297 | Exploration 0.10 | Fruits 11 | Timesteps   77 | Total Reward   98
Episode 25509/30000 | Loss 199.9005 | Exploration 0.10 | Fruits 12 | Timesteps  157 | Total Reward  113
Episode 25510/30000 | Loss  82.5284 | Exploration 0.10 | Fruits 10 | Timesteps   60 | Total Reward   84
Episode 25511/30000 | Loss  50.9702 | Exploration 0.10 | Fruits  4 | Timesteps   41 | Total Reward   21
Episode 25512/30000 | Loss  79.1405 | Exploration 0.10 | Fruits  5 | Timesteps   59 | Total Reward   29
Episode 25513/30000 | Loss 144.6286 | Exploration 0.10 | Fruits 12 | Timesteps  119 | Total Reward  113
Episode 25514/30000 | Loss  17.3537 | Exploration 0.10 | Fruits  1 | Timesteps   11 | Total Reward    3
Episode 25515/30000 | Loss  78.8018 | Exploration 0.10 | Fruits  6 | Timesteps   48 | Total Reward   38
Episode 25516/30000 | Loss  10.7790 | Exploration 0.10 | Fruits  0 | Timesteps    7 | Total Reward   -1
Episode 25517/30000 | Loss  21.2446 | Exploration 0.10 | Fruits  4 | Timesteps   17 | Total Reward   21
Episode 25518/30000 | Loss  16.1764 | Exploration 0.10 | Fruits  2 | Timesteps   13 | Total Reward    8
Episode 25519/30000 | Loss  79.0935 | Exploration 0.10 | Fruits 13 | Timesteps   57 | Total Reward  129
Episode 25520/30000 | Loss 145.3444 | Exploration 0.10 | Fruits 11 | Timesteps  104 | Total Reward   98
Episode 25521/30000 | Loss  82.3118 | Exploration 0.10 | Fruits  7 | Timesteps   57 | Total Reward   48
Episode 25522/30000 | Loss  12.9322 | Exploration 0.10 | Fruits  2 | Timesteps    7 | Total Reward    8
Episode 25523/30000 | Loss  23.8637 | Exploration 0.10 | Fruits  2 | Timesteps   17 | Total Reward    8
Episode 25524/30000 | Loss 102.5636 | Exploration 0.10 | Fruits 10 | Timesteps   74 | Total Reward   84
Episode 25525/30000 | Loss  48.5519 | Exploration 0.10 | Fruits  3 | Timesteps   32 | Total Reward   14
Episode 25526/30000 | Loss 181.7913 | Exploration 0.10 | Fruits 10 | Timesteps  128 | Total Reward   84
Episode 25527/30000 | Loss  97.3514 | Exploration 0.10 | Fruits  7 | Timesteps   65 | Total Reward   48
Episode 25528/30000 | Loss 138.9415 | Exploration 0.10 | Fruits 12 | Timesteps   87 | Total Reward  113
```

Screenshot: Training of data

Playing the Snake Game

Trained Data is stored is stored in CSV files and DQN model file created by Tensorflow. To play the snake game, there are 2 ways - the human version using the make play-human command for a user to play the game or over GUI by using the DQN agent, we first need to load the snake environment and then load the pre-trained snake model with Keras. In order to execute the automated GUI version, we give the make play-gui command to see our agent play the game.



```
yuvra@LAPTOP-B13CPOGO MINGW64 /d/courses sjsu/AI/Project/snake-ai-reinforcement (master)
$ make play-gui
python ./play.py --interface gui --agent dqn --model dqn-final.model --level "snakeai/levels/10x10-blank.json" --num-episodes 10
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecat
ed. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2018-05-06 21:51:51.125191: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was
 not compiled to use: AVX2
```

References

[1] Shu Kong, 80888472, skong2@uci.edu Joan Aguilar Mayans, 87286425, joana1@uci.edu, Automated Snake Game Solvers via AI Search Algorithms

[2] Bowei Ma, Meng Tang, Jun Zhang, Exploration of Reinforcement Learning to SNAKE

[3] Melrose Roderick, James MacGlashan, Stefanie Tellex, Implementing the Deep Q-Network (Submitted on 20 Nov 2017)

[4] HDF5 for Python, https://www.h5py.org/

[5] Keras: The Python Deep Learning library, https://keras.io/

[6] NumPy © 2018 NumPy developers. Created using Sphinx 1.6.6, http://www.numpy.org/

[7] pandas 0.19.2 documentation, pandas: powerful Python data analysis toolkit, http://pandas.pydata.org/pandas-docs/version/0.19/

[8] About Pygame, https://www.pygame.org/wiki/about

[9] About TensorFlow, https://www.tensorflow.org/