Project Title: Doom DQN using CNN
Author: Daniel Newman

Aim:

The aim of this experiment is to explore how a convolutional neural network can be applied to process images and identify points of interest that inform and train our model to complete an in-game challenge in Doom utilising a Deep Q-Learning model.

Experiment

Requirements

Python 3.6

Python Libraries

- Numpy
- SciPy
- Reinforcement Learning Library
  - Tensorflow
  - PyTorch
  - Theano
  - etc
- OpenCV
- OpenAI Gym

Python related IDE

- Spyder
- Jupyter
- Pycharm

Experiment Specific Requirements

Vizdoom: http://vizdoom.cs.put.edu.pl/tutorial
Vizdoom Github: https://github.com/mwydmuch/ViZDoom

Steps to Reproduce

Our main function of the AI is the Neural Network, where we develop our layers of our where we start with our convolutional layers for image processing, these layers have the rectifier activation method applied to them. These layers are then flattened and processed through our regular neural network layers for to determine the best value action from the processed frames.

```
class Net(nn.Module):
    def __init__(self, available_actions_count):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 8, kernel_size=6, stride=3)
        self.conv2 = nn.Conv2d(8, 8, kernel_size=3, stride=2)
        self.fc1 = nn.Linear(192, 128)
        self.fc2 = nn.Linear(128, available_actions_count)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = x.view(-1, 192)
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

These actions are then stored in a local memory which is re-sampled over time to reinforce the neural networks connections. We then move onto the training process to train up our network with the goal of the Neural network reaching a particular average test value, which in this case we rate at 1000

```
class ReplayMemory:
    def __init__(self, capacity):
        channels = 1
        state_shape = (capacity, channels, resolution[0], resolution[1])
        self.s1 = np.zeros(state_shape, dtype=np.float32)
        self.s2 = np.zeros(state_shape, dtype=np.float32)
        self.a = np.zeros(capacity, dtype=np.int32)
        self.r = np.zeros(capacity, dtype=np.float32)
        self.isterminal = np.zeros(capacity, dtype=np.float32)

        self.capacity = capacity
        self.size = 0
        self.pos = 0

    def add_transition(self, s1, action, s2, isterminal, reward):
        self.s1[self.pos, 0, :, :] = s1
        self.a[self.pos] = action
        if not isterminal:
            self.s2[self.pos, 0, :, :] = s2
        self.isterminal[self.pos] = isterminal
        self.r[self.pos] = reward

        self.pos = (self.pos + 1) % self.capacity
        self.size = min(self.size + 1, self.capacity)

    def get_sample(self, sample_size):
        i = sample(range(0, self.size), sample_size)
        return self.s1[i], self.a[i], self.s2[i], self.isterminal[i], se
```

The training process is batched into sequences of 2000 to reduce the demands required on the hardware. Since we didn't have CUDA enabled GPU at our disposal, we had to resort to processing our neural network via the CPU.

Observations:

Through the training process we observed that the model wasn't learning optimally and would get stuck in a local minimum which it struggled to get out of on its way to producing an optimal output value. This caused the AI to get stuck in inefficient behaviours that even with changes to the learning rate and discount factor we couldn't reach our target goal of 1000 and only reached an average of 300, with a rare maximum value of 1000. We were also limited by our hardware in terms of the processing length, where we reached upwards of 200 epochs without much of an increase in performance over time.

Discussion:

The results for the AI weren't as promising as we hoped for, however it does illustrate how the use of accompanying optimizations can help facilitate a working as intended AI from a poor AI. If we implemented our Deep Q-Learning neural network with Stochastic Gradient Descent and Eligibility Trace I believe that we could produce better results, but we may still be limited by our hardware as to what our average value could accomplish. Investigation in different Reinforcement Learning Models can also be useful as a comparison between each model to determine the benefits and limitations of each model.

Conclusion

In conclusion we found that our DQN was not the most efficient model to use in this situation, and with our new knowledge we can try to apply a new model to our situation for contrast and comparison.