Università degli studi di Verona

Dipartimento di Informatica

Final project for the "fundamentals of Artificial Intelligence" course:

CartPole-v1 trained with DQN using CNN

A.A 2020/2021

Studente:

Luca Marzari VR457336

Professore:

Alessandro Farinelli

Contents

Chapter 1

Problem definition:

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.
CartPole-v1 defines "solving" as getting average reward of 195.0 over 100 consecutive trials.

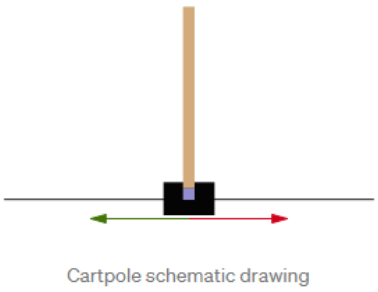

Figure 1.1: CartPole-v1 environment from OpenAI gym

This work is based on OpenAI's gym[1][2], a toolkit for developing and comparing reinforcement learning algorithms.
In this particular environment the observation space (possible state values) are four and the action space (possible actions that can be performed) are two. The table presented here below summarize the problem statement:

Observation			
Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	$-\infty$	$+\infty$
2	Pole Angle	-24°	+24°
3	Pole Angular speed	$-\infty$	$+\infty$

Action	
Num	Action
0	Push cart to the left
1	Push cart to the right

1.1 Objectives

In this work the first objective is to solve CartPole-v1 with the DQN algorithm[3], i.e as mentioned before, get an average reward of 195.0 over 100 consecutive trials, using a normal deep neural network. Here we have the observation state that could be represented with few features (4) but we have continuous states. We use Reinforcement Learning in particular DRL because we can not have a table Q(s,a) for each possible state since the variables defined in the table above has real domain.

The second objective is to solve CartPole-v1 using the DQN algorithm but this time instead of using as input the features (x, x', θ, θ') i.e. cart position, cart velocity, pole angle, and pole angle velocity, we use images, therefore the final aim of this work is training an agent that learns to balance a cart using Convolutional Neural Networks (CNN). In particular, we need and use CNN because let us suppose to have a 300x300 rgb image, and we want to use a deep neural network with images as input. In this situation, all the hidden units of the network would be connected to all pixels in input and we would have 270,000 weights for a single neuron!
If we build a model that has many fully connected units at every layer the model would be slow and prone to overfitting.

Chapter 2

Introduction to Convolutional Neural Networks

2.1 What is convolution? (briefly)

Convolutional operation in digital signal processing is also called filtering a signal with a particular kernel, which is a vector or a matrix. Generally, the kernel is also called a filter. The latter computes the local averages by averaging the values within a window. Let us see an example with an image and a filter 3x3:

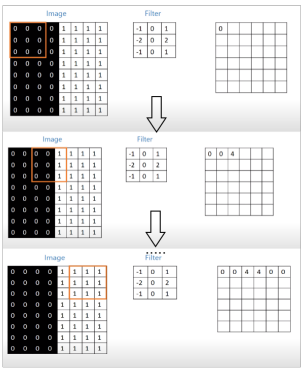

Figure 2.1: Convolution operation

In the example, we see the first part of a convolution of an 8x8 image with a 3x3 kernel or filter. The operation involves the overlay of the kernel on the image, multiply the elements, sum the products, and move to the next tile. This specific kernel used in the example is an edge detector that detects the vertical edges of images. We use CNN to have a model that learns parameters from data and discovers what kind of feature extractors would be useful to accomplish a task.
To solve this problem we can connect each neuron to only a region of the input/image, in other words, we could make a convolution of the image. A layer that consists of convolutional units like these presented before is called a convolutional layer. The parameters learned by each unit in a convolutional layer can be thought of as a filter and the output of these units are simply the filtered versions of their inputs.
One of the main advantages of having these CNNs is the abstraction level that you can have in each level, that is, each level extracts a certain type of image features, and therefore by changing the various kernels/filters you can extract various details of the image. An example is presented in the following images:

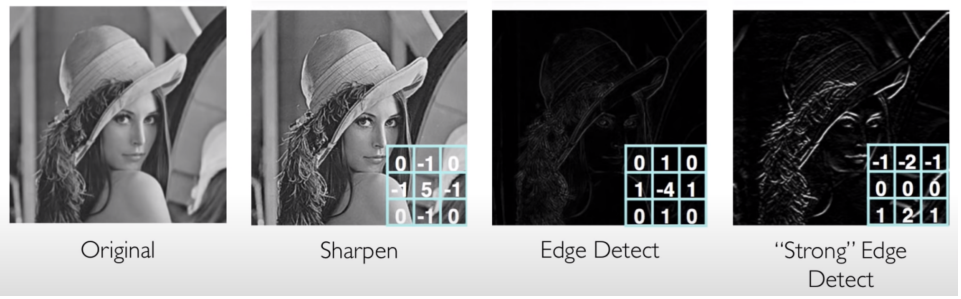

Figure 2.2: filtering operation, source: Convolutional Neural Networks | MIT 6.S191

Chapter 3

Implementation and results

In this chapter, I will present the implementation of DQN on the cart-pole scenario using CNN to process the input(i.e., the image of the cart-pole).

3.1 Model

Resolving CartPole using images involves using the $env.render(mode = \text{'rgb\_array'})$ function to have a rendered frame image for each step of the episode. Obviously, retrieving information from every single frame is computationally heavy and time-consuming, which is one of the reasons why CNNs take longer to solve a task than to use the (x, x', θ, θ') as input for a deep neural network. The Cartpole gym environment outputs 600x400 RGB arrays(600x400x3). There were too many pixels giving no information about the cart position (for example the set of white pixels all on the right and left), more than necessary, so in order to simplify the task I converted the output to grayscale and resized the image to 240x160.
The result is presented here below:

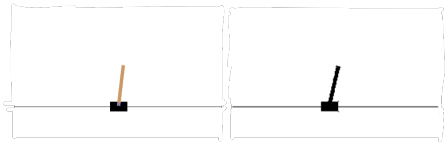

Figure 3.1: Rendered image of CartPole environment (left); the image downsized to 240x160 and converted to grayscale that I used as input to the algorithm (right)

In order to derive both the direction, velocity, and acceleration of the moving object, I kept the last 4 frames for each step, so my input for the CNN was a numpy array of (240x160x4). For building the model I used Tensorflow and in particular Keras, an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

3.2 Architecture

The input to the neural network consists of 160x240x4 images. After the input layer we have three 2D convolutional layers with kernel that started from 5x5 and became 3x3 in the last layer of Conv2D with stride 3x3, 2x2 and 1x1 in the last 2D convolutional layer.
The Stride basically told me how much shift the window during convolution. As mentioned above one of the main advantages of having these CNNs is the level of abstraction that you can have in each level, that is, each level extracts a certain type of image feature.
We use ReLu activation functions for each layer both Conv2D and Dense. The final hidden layer was fully connected "Flatten" units and I obtained as output of this Flatten layer 52992 nodes which were the inputs for the following four Dense levels of the neural network.

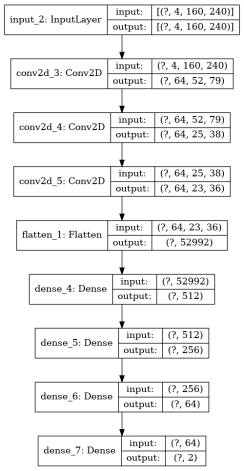

Figure 3.2: Model architecture

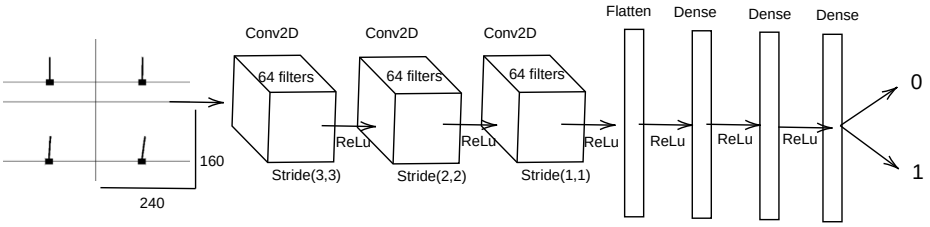

Figure 3.3: Model architecture

As we can notice from the previous two images of model architecture, the output of the network consists of two neurons. These neurons represent the possible action that the agent could perform in the environment, in particular, if the output equals 0 means that the agent will move the cart to the left otherwise on the right. The algorithm used is the classic DQN[3]:

Algorithm 2 DQN

Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ randomly in $[-1, 1]$
Initialize s {observe current state}
loop
  Select and execute action a
  Observe new state s' receive immediate reward r
  Add (s, a, s', r) to experience buffer
  Sample mini-batch MB of experiences from buffer
  for $(\hat{s}, \hat{a}, \hat{s}', \hat{r}) \in MB$ **do**
    $\frac{\partial Err(\boldsymbol{w})}{\partial \boldsymbol{w}} = (Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_{\overline{\boldsymbol{w}}}(\hat{s}', \hat{a}'))\frac{\partial Q_{\boldsymbol{w}}(\hat{s}, \hat{a})}{\partial \boldsymbol{w}}$
    update weights $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial Err(\boldsymbol{w})}{\partial \boldsymbol{w}}$
  end for
  update state $s \leftarrow s'$
  every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$
end loop

The results are presented in the next page.

3.3 Results:

All the plot shown below are the average reward over 100 consecutive trials. The first result presented is the solution of CartPole-v1 with DNN using DQN algorithm. Here the inputs to the network, as mentioned before, are the features cart position, cart velocity, pole angle, and pole angle velocity provided directly by the environment. In this implementation we have a DNN composed by an input layer of 4 values connected to two hidden layers each with a hidden layer size of 32 nodes with a ReLu activation function. The output layer is composed of two nodes that are as mentioned before the possible actions that the agent could perform. In particular, for the implementation of DQN using DNN for CartPole, I didn't use a target network but only a single training network with these parameters: epsilon = 1, gamma = .95, batch_size = 32, epsilon_min =.01, epsilon_decay = .995, learning_rate = 0.001.

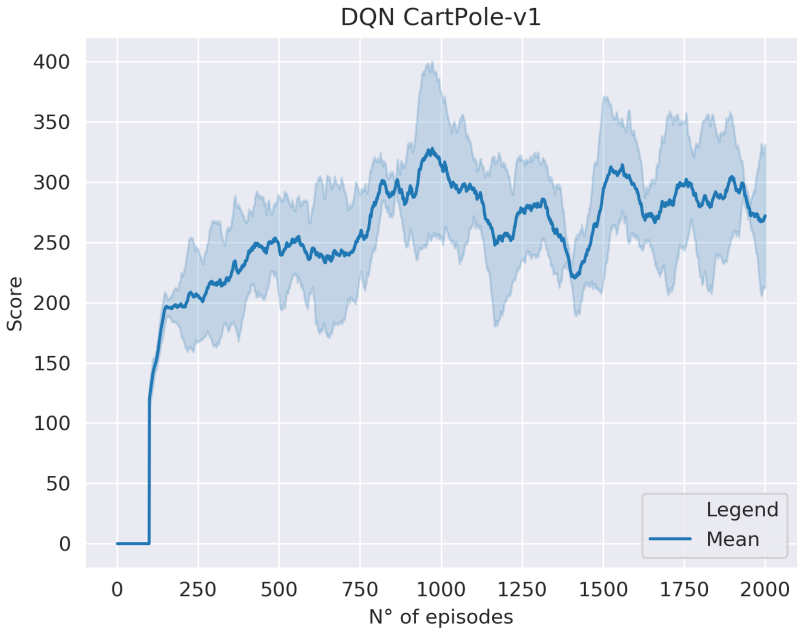

Figure 3.4: CartPole-v1 using DQN algortithm. The dark blue line represents the average reward over 100 consecutive trials, the light blue shadow represents the variance of the average over three randomly seeded runs.

As we can see from the plot starting from episode 150 more or less the CartPole-v1 is "solved" since it had an average reward of 195.0 or greater over 100 consecutive trials.

The second result presented is the solution of CartPole-v1 with CNN using standard DQN algorithm (with one training network and one target network) with these parameters epsilon = 1, epsilon_min = 0.01, epsilon_decay =0.9995, self.batch_size = 32, TAU = 0.1 (target network soft update hyperparameter):

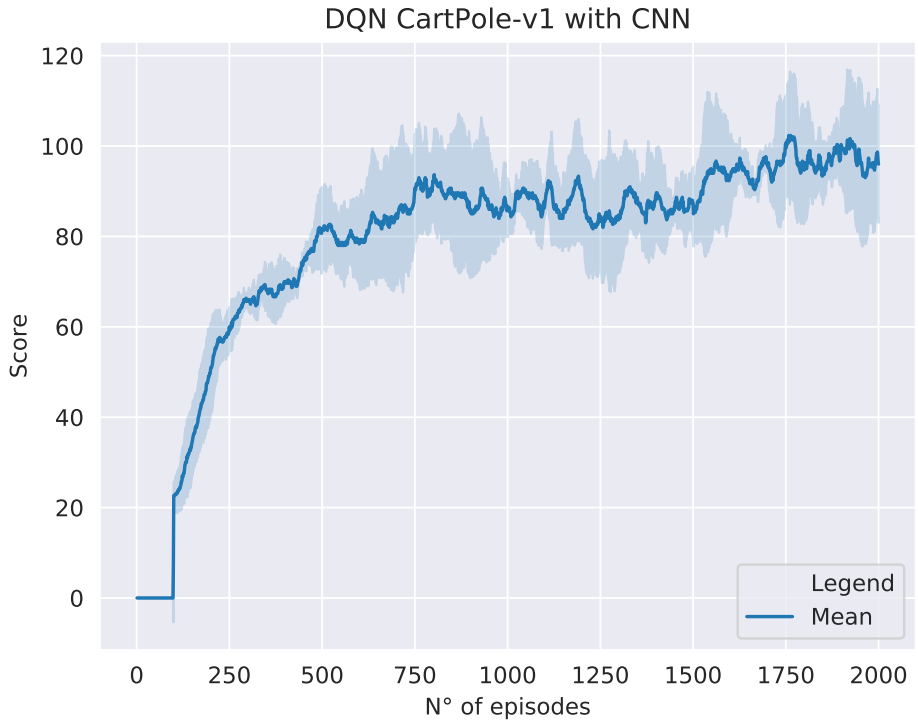

Figure 3.5: CartPole-v1 using CNN. The dark blue line represents the average reward over 100 consecutive trials, the light blue shadow represents the variance of the average over three randomly seeded runs.

Making a comparison between the two plots, with the same number of episodes completed, we note that the use of images and in particular CNN to solve the task requires more time to consider the CartPole task solved. However, we can see how the average is constantly increasing, so I speculate that with less than 5000 episodes the task can be solved also with CNN.

Bibliography

[1] https://gym.openai.com/docs/

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," arXiv preprint arXiv:1606.01540, 2016

[3] Human-level control through deep reinforcement learning (V. Mnih et al., Nature 2015)