Patel-Small-Fully-Connected

January 9, 2020

0.1 Training of the small fully connected network from Patel et al.

In this notebook we train a fully connected network with one hidden layer of 1000 neurons on the Breakout problem as in the paper by Patel et al. [1]. For a detailed description of how to train DQNs with our code, see the tutorial in the CartPole experiments chapter 1-DQN-Training. In this notebook, we only outline the relevant changes to the CartPole problem in detail.

[ ]:
```
import torch
import torch.optim as optim
import os
import sys
import random
import numpy as np
import matplotlib.pyplot as plt
# hack to perform relative imports
sys.path.append('../../')
from Code import train_agent, SQN, FullyConnected
from skimage.transform import resize
```

Attention: If the directory with the specified name already exists, the next cell will throw an error. You need to specify a different name or delete the old directory. If this happens, you should restart the kernel, as the directory is a relative path which changes everytime this cell is run.

[ ]:
```
# switch to the Result Directory
os.chdir('./../../Results/')
# choose the name of the result directory
result_directory = 'Breakout-Experiment-DQN-Training-Patel'
# create the result directory (throws an error if the directory already exists)
os.makedirs(result_directory)
os.chdir(result_directory)
# for the first experiment we create an additinonal sub folder
os.makedirs('DQN')
os.chdir('DQN')
```

[ ]:
```
# set seeds for reproducibility (these are not the same as used to create the␣
 ↪graphic in our thesis).
torch_seed = 12345678
torch.manual_seed(torch_seed)
random_seed = 12345678
random.seed(random_seed)
```

```
gym_seed = 12345678
```

Next, we define the hyperparameters of the neural network. There are some differences to the CartPole problem: 1. Most importantly, the agent does not receive the raw pixels as input, but some preprocessing happens. We use the grayscale preprocessing analog to Patel et al. This preprocessing is specified in a function and then passed to the agent. 2. BreakOut uses some additional hyperparameters. Update Frequency determines after how many steps in the environment the networks are updated. Frame skip describes how often an action the agent selected is repeated where the agent does not see any of the in between states. No-op-max describes the maximum number of "Do-nothing" operations executed in the beginning of each episode. No-op-range is a tuple (x,y) where y is the no-op-max and x is the no-op-min which is equivalent to the history length which is 4. We use an initial replay memory size where before reaching this size, no training happens. 3. The target update in BreakOut happens every x iterations instead of every x episodes. 4. There is no notion of solving the BreakOut environment. 5. Patel et al. do not define a maximum number of steps per episode, so we simply set it to infinity. However this can only be done if the final random action probability is sufficiently high, such that the agent cannot get stuck, because it never uses "fire" to get a new ball into play. 6. We use the optimizer RMS-Prop as as Patel et al. use the same hyperparameters as Mnih et al. [2] who use RMS-Prop. However, as PyTorchs RMS-Prop optimizer has slightly different hyperparameters compared to tensorflows RMS-Prop used in Mnih et al., the optimizers might still be slightly different.

[ ]:
```
#CartPole
env = 'BreakoutDeterministic-v4'

# define the preprocessing function


#hyperparameters, set analog to Patel et al.
BATCH_SIZE = 32
DISCOUNT_FACTOR = 0.99
EPSILON_START = 1.0
EPSILON_END = 0.1
EPSILON_DECAY = 0.9999977 # 0.9999977 makes epsilon reach 0.1 after␣
 ↪approximately 1000000 frames
TARGET_UPDATE_FREQUENCY = 10000
# variable that tells the agent whether to update the target net every x␣
 ↪episodes or every x iterations
TARGET_UPDATE_MODE = 'iterations'
LEARNING_RATE = 0.00025
REPLAY_MEMORY_SIZE = 2*10**5
# additional BreakOut hyperparameters
UPDATE_FREQUENCY = 4
FRAMESKIP = 4
NO_OP = 0 # the action which is "do nothing"
NO_OP_RANGE = (4,30)
OBSERVATION_HISTORY_LENGTH = 4
# minimum size of the replay memory before the training starts
INITIAL_REPLAY_SIZE = 50000
```

```
# there is no notion of a gym standard for Breakout
GYM_TARGET_AVG = None
GYM_TARGET_DURATION = None
# Patel et al. do not specify a limit on the maximum number of steps
MAX_STEPS = np.inf
# number of episodes to train the agent as specified by Patel et al.
NUM_EPISODES = 30000
# whether to use Double Q Learning and Gradient Clipping
DOUBLE_Q = False
GRADIENT_CLIPPING = True
# whether to render the environment
RENDER = True

# device: automatically runs on GPU, if a GPU is detected, else uses CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# input is 80x80, one hidden layer with 1000 neurons
architecture = [6400, 1000, 4]
policy_net = FullyConnected(architecture).to(device)

target_net = FullyConnected(architecture).to(device)
target_net.load_state_dict(policy_net.state_dict())

# initialize optimizer
optimizer = optim.RMSprop(policy_net.parameters(), lr=LEARNING_RATE,eps=0.
 ↪001,alpha=0.95)

# define input preprocessing: grayscale processing according to Patel
def input_preprocessing(observation_history):
    def rgb2gray(rgb):
        """converts a coloured image to gray"""
        # extract the luminance
        image = rgb[:, :, 0] * 0.2126 + rgb[:, :, 1] * 0.7152 + rgb[:, :, 2] * 0.
 ↪0722
        # crop the image
        image = image[25::, :]
        # downsize to 80x80
        image_resized = resize(image, (80, 80))
        # return the processed image and the new prev frame
        return image_resized
    preprocessed = rgb2gray(observation_history[0]) + 0.75 *␣
 ↪rgb2gray(observation_history[1]) + 0.5 * rgb2gray(observation_history[2]) + 0.
 ↪25 * rgb2gray(observation_history[3])
    # rescale such that maximum value is 255 for conversion to uint8
    preprocessed[preprocessed>255] = 255
    preprocessed = torch.tensor(preprocessed,device=device,dtype=torch.uint8)
    # cast to tensor of shape (6400,) to be processed correctly by the SQN
    preprocessed = preprocessed.reshape(6400,)
    return preprocessed
```

Warning: The code below is not fully tested, because the computer this was tested on has not enough computational power to complete the training in a reasonable amount of time. If you run into any problems, feel free to contact me via chris.hahn@tum.de. From our thesis it should be expected that the agent achieves a 100-episode reward of around 10 after a couple of thousand episodes and then pretty much stays there for the raining eppisodes.

[ ]:
```
# call train agent with the additional options for Breakout
train_agent(env,policy_net,target_net,BATCH_SIZE,DISCOUNT_FACTOR,EPSILON_START,
           ␣
 ↪EPSILON_END,EPSILON_DECAY,TARGET_UPDATE_FREQUENCY,optimizer,LEARNING_RATE,
           ␣
 ↪REPLAY_MEMORY_SIZE,device,GYM_TARGET_AVG,GYM_TARGET_DURATION,num_episodes=NUM_EPISODES,
           ␣
 ↪max_steps=MAX_STEPS,render=RENDER,double_q_learning=DOUBLE_Q,gradient_clipping=GRADIENT_CLIPP
            initial_replay_size=INITIAL_REPLAY_SIZE,gym_seed=gym_seed,␣
 ↪torch_seed=torch_seed, random_seed=random_seed,
            input_preprocessing=input_preprocessing,␣
 ↪no_op_range=NO_OP_RANGE,no_op=NO_OP,
           ␣
 ↪update_frequency=UPDATE_FREQUENCY,observation_history_length=OBSERVATION_HISTORY_LENGTH,
           target_update_mode=TARGET_UPDATE_MODE, frameskip=FRAMESKIP)
```

The blue line shows the single episode rewards, thr orange line indicates 100-episode average rewards.

Next, we train a DSQN.

Attention: If the directory with the specified name already exists, the next cell will throw an error. You need to specify a different name or delete the old directory. If this happens, you should restart the kernel, as the directory is a relative path which changes everytime this cell is run.

Warning: Again, the code is not fully tested. You can have a look at our thesis to see what kind of results can be expected.

[ ]:
```
os.makedirs('./../DSQN')
os.chdir('./../DSQN')
```

[ ]:
```
# set up the network
SIMULATION_TIME = 20
architecture = [6400,1000,4]
policy_net = SQN(architecture, device, alpha=0, beta=1,␣
 ↪simulation_time=SIMULATION_TIME,
                 add_bias_as_observation=True)
target_net = SQN(architecture, device, alpha=0, beta=1,␣
 ↪simulation_time=SIMULATION_TIME,
                 add_bias_as_observation=True)
target_net.load_state_dict(policy_net.state_dict())
```

```
# initialize optimizer
optimizer = optim.Adam(policy_net.parameters(), lr=0.001)

# call train agent with the additional options for Breakout
train_agent(env,policy_net,target_net,BATCH_SIZE,DISCOUNT_FACTOR,EPSILON_START,
          ␣
 ↪EPSILON_END,EPSILON_DECAY,TARGET_UPDATE_FREQUENCY,optimizer,LEARNING_RATE,
          ␣
 ↪REPLAY_MEMORY_SIZE,device,GYM_TARGET_AVG,GYM_TARGET_DURATION,num_episodes=NUM_EPISODES,
          ␣
 ↪max_steps=MAX_STEPS,render=RENDER,double_q_learning=DOUBLE_Q,gradient_clipping=GRADIENT_CLIPP
            initial_replay_size=0,gym_seed=gym_seed, torch_seed=torch_seed,␣
 ↪random_seed=random_seed,
            input_preprocessing=input_preprocessing,␣
 ↪no_op_range=NO_OP_RANGE,no_op=NO_OP,
          ␣
 ↪update_frequency=UPDATE_FREQUENCY,observation_history_length=OBSERVATION_HISTORY_LENGTH,
           target_update_mode=TARGET_UPDATE_MODE, frameskip=FRAMESKIP)
```

[1] Devdhar Patel, Hananel Hazan, Daniel J. Saunders, Hava T. Siegelmann, and Robert Kozma. Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game. Neural Networks, 2019.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, Volume 518, 26 February 2015.