6-Load-in-pyNN

January 4, 2020

0.1 Load in pyNN

In this notebook we load some of the pretrained networks in PyNN using both NEST and SpiNNaker as backend.

Experiment List: 1. DQN-Training (How to train a conventional DQN and a spiking DQN using Surrogate Gradients (DSQN).) 2. Load-DQN (How to load a previously saved D(S)QN and how to save a replay dataset.) 3. Train-Classifier (How to train a spiking or non-spiking classifier on the saved replay data set.) 4. SNN-Conversion (How to convert a DQN and a Classifier to a SNN.) 5. Load in NEST (How to load a converted or directly trained spiking network in NEST.) 6. Conversion in pyNN with NEST or SpyNNaker (How to load spiking network in pyNN using NEST or SpyNNaker as backend.)

Attention: In our implementation, importing PyNN using NEST throws an error that nest has no attribute "l1_api". We resolved this error by deleting all lines which use the "l1_api" in the PyNN package. This did not cause any errors or problems for running the simulation

[1]:
```
import torch
import os
import sys
import random
import matplotlib.pyplot as plt
# hack to perform relative imports
sys.path.append('../../')
from Code import PyNNAgent, load_agent, FullyConnected, SQN

# set seeds
torch.manual_seed(1)
random.seed(1)
gym_seed = 1

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

# switch to the Result Directory
os.chdir('./../../Results/')
```

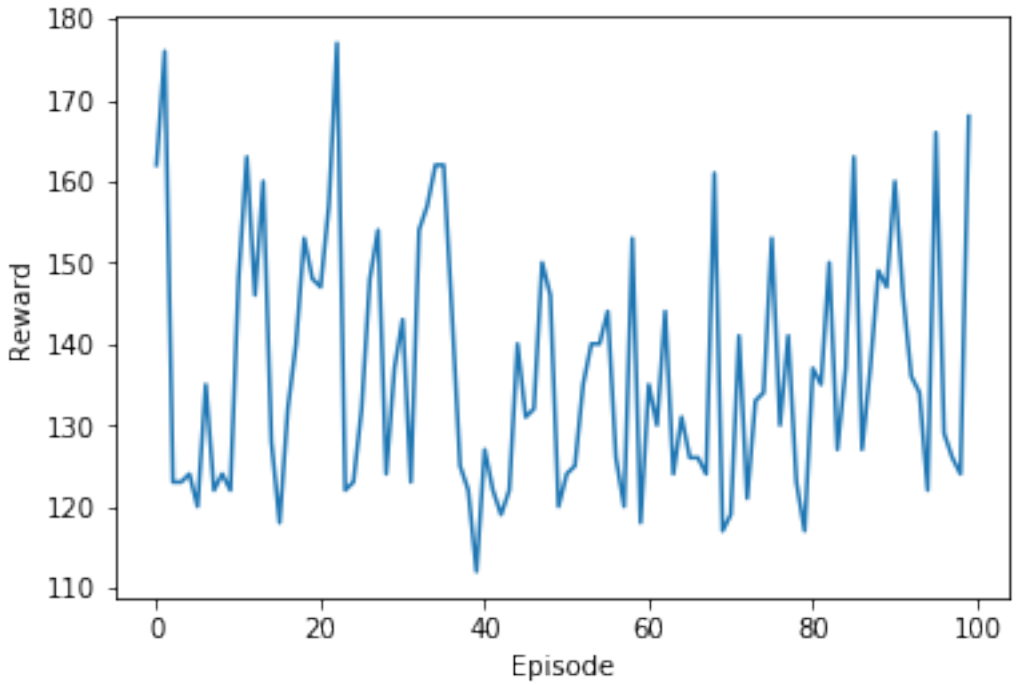
```
Detected PyNN version 0.9.5 and Neo version 0.6.1
```

To begin with, we load a network obtained by conversion from a classifier which was obtained in the previous tutorial (SNN-Conversion). In the first experiment, we use NEST as backend.

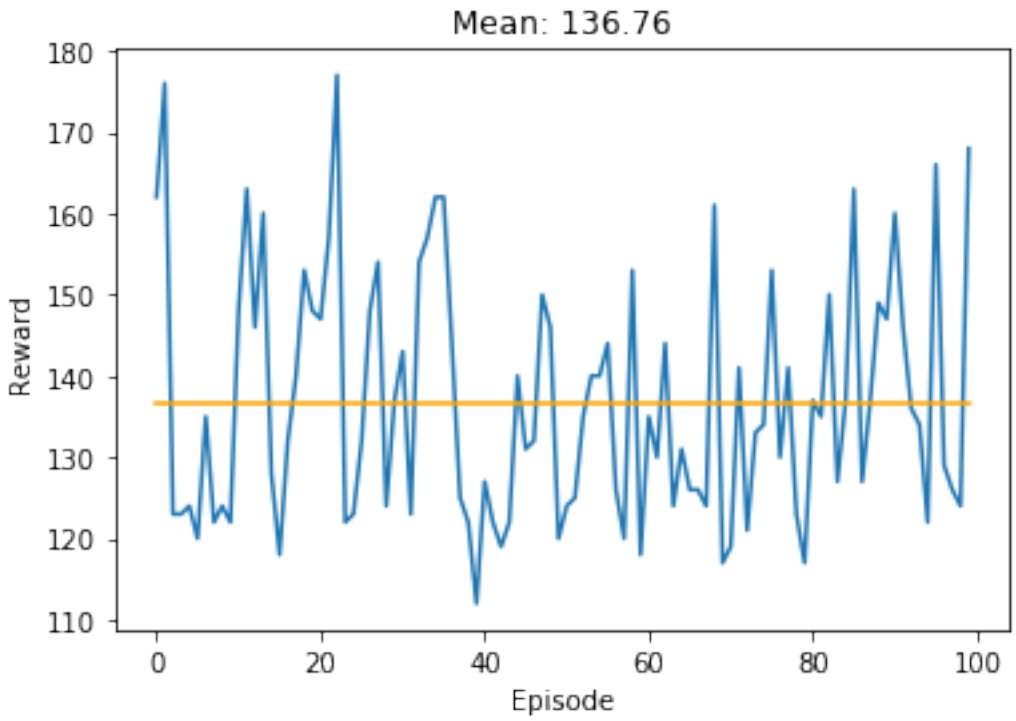
[2]:
```
# specify the file, where the network is saved
file = './CartPole-A/Classifier-Converted/model.pt'

# set hyperparameters of NEST:
# encoding/decoding methods are limited to constant input currents and potential␣
 ↪outputs.
# set correct architecture
architecture = [4,16,16,2]
# set simulation time in ms, changing the resolution is not supported in our code
simulation_time = 100
# the backend can be set to 'nest' or 'spinnaker'. With some modifications the␣
 ↪code can also be adjusted to use
# native pynn models, the backend would then be specified as 'pynn'.
backend = 'nest'

# set up network in PyNN with NEST as backend
pynn_agent = PyNNAgent(architecture, file, simulation_time, backend=backend,␣
 ↪add_bias_as_observation=False)
```

[3]:
```
# load the agent and compare against the original Classifier
classifier = FullyConnected(architecture).to(device)
classifier.load_state_dict(torch.load('./CartPole-A/Classifier/trained/model.
 ↪pt'))

# run the PyNN agent and compare against the original classifier
env = 'CartPole-v0'
```

[7]:
```
load_agent(env,pynn_agent,device,epsilon=0,gym_seed=gym_seed,save_replay=False,
           max_steps=500,num_episodes=100, render=True,␣
 ↪compare_against=classifier)
```




```
Similarity (Conversion Accuracy) after 13676 iterations: 77.93945598128107%
Complete
Mean:  136.76
Std:  15.11980439187243
Similarity (Conversion Accuracy) after 13676 iterations: 77.93945598128107%
```



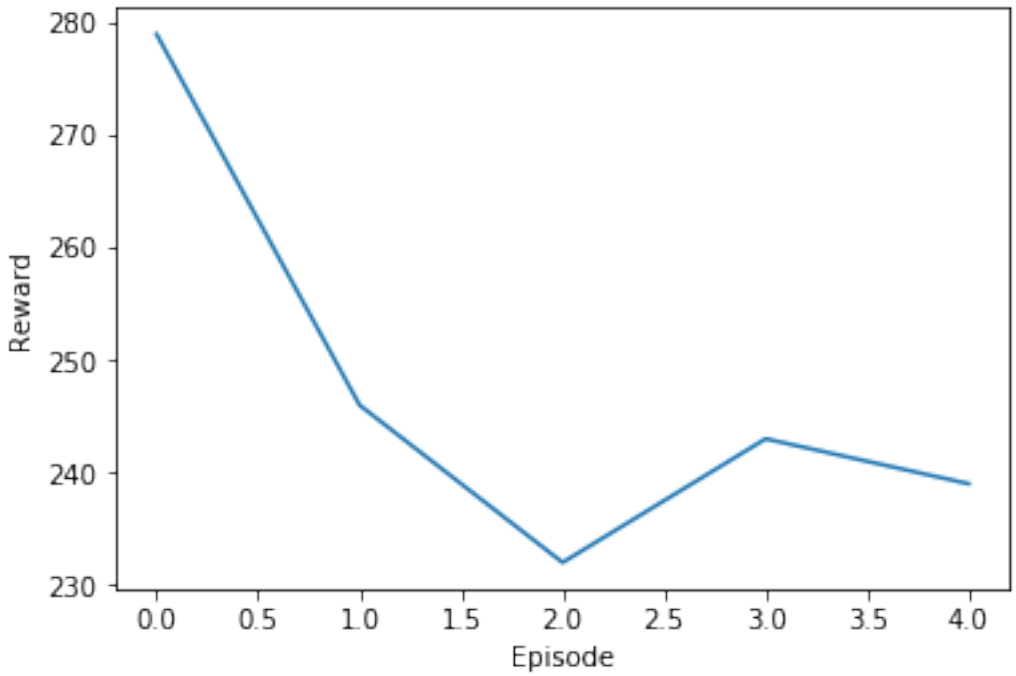
Info: The results when loading in PyNN using NEST as backend are not equivalent to loading in NEST. This is probably due to simulation hyperparameters being set differently by default. Further investigation will be necessary to reproduce the results from loading in NEST.

Next, we load the same classifier using SpiNNaker as backend. To run this your computer needs to be connected to a SpiNNaker neuromorphic hardware chip which should be automatically detected if it is connected correctly. As simulation on SpiNNaker is much slower compared to our other experiments, we run the agent for only 5 episodes.

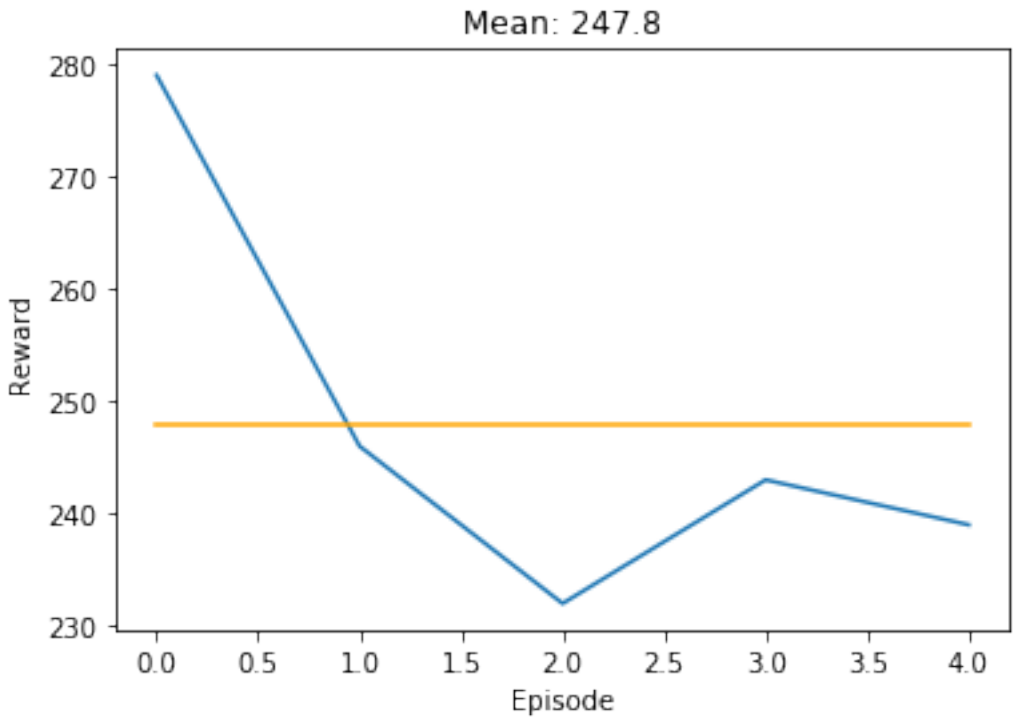
Attention: If you want to run the following cell twice, the Kernel needs to be restarted to avoid getting the error: "ConfigurationException: Illegal call after simulation is shutdown"

Info: Communication with the SpiNNaker chip produces large amount of terminal output which we cannot be easily suppressed. For the sake of understanding this experiment, it can be ignored.

[4]:
```
# set backend
backend='spinnaker'

# set up network in PyNN as before
pynn_agent = PyNNAgent(architecture, file, simulation_time, backend=backend,␣
 ↪add_bias_as_observation=False)
# run the PyNN agent and compare against the original classifier
load_agent(env,pynn_agent,device,epsilon=0,gym_seed=gym_seed,save_replay=False,
           max_steps=500,num_episodes=5, render=True, compare_against=classifier)
```




```
Similarity (Conversion Accuracy) after 1239 iterations: 84.50363196125907%
Complete
Mean:  247.8
Std:  18.212632978237934
Similarity (Conversion Accuracy) after 1239 iterations: 84.50363196125907%
```



Info: The results when loading in PyNN using SpiNNaker as backend, are different again from both the results using PyNN with NEST as backend and using NEST directly. This is propably due to differences in the implementation of the neuron models (we use 'iaf_psc_delta' in NEST and 'IFCurDelta' in SpiNNaker). Additionally different simulation hyperparameters could be responsible for discrepancies. This needs further investigation. Furthermore, simulating on SpiNNaker is extremely slow compared to running the experiments in NEST. Whether this is due to the hardware itsel being slow or the communication between the conventional machine and the SpiNNaker chip needs to be investigated to run larger experiments in the future.