Self-Driving Cars

Exercise 0 - Introduction

Kashyap Chitta

Autonomous Vision Group
MPI-IS / University of Tübingen

October 18, 2019

University of Tübingen
MPI for Intelligent Systems

Autonomous Vision Group

Outline

- Setup for Exercises
- PyTorch
- OpenAI Gym
- Exercise 0

Setup for Exercises

ILIAS

- We organize the exercises using the ILIAS system `https://ovidius.uni-tuebingen.de/ilias3`
- Exercise sheets will be available in the ILIAS system. Please be aware of the **submission deadline**.
- You are eligible to finish the homework within a group up to 2 people, but **each person must submit a solution.**
- If you have any questions, please ask at the **forum** on ILIAS.
- TA Email IDs: `eshed.ohn-bar@tue.mpg.de`, `kashyap.chitta@tue.mpg.de`, `katja.schwarz@tue.mpg.de`.

.

TCML cluster

- You are eligible to use the Training Center for Machine Learning (TCML) cluster for exercises in this lecture.
- We will create accounts for you on the cluster. Each group shares an account. **You are not supposed to apply for the account by yourself.**
- Cluster has a master node to launch jobs, and 40 compute nodes to execute jobs.
- Compute nodes are allocated based on a queuing system, so please **start the assignments early** if you want to use this resource.

TCML cluster

- Login to master:
  `ssh username@tcml-master01.uni-tuebingen.de`
- To access compute nodes: create an `.sbatch` file
- Please find more information/instructions about the cluster below:
  `https://docs.google.com/document/d/1AgtLy28VVZaPe79TwOb9jjC4F1KVzffb8y1vZoURZE8/edit?usp=sharing`.

PyTorch

PyTorch

- What is PyTorch?

  A Python-based scientific computing package for Deep Learning.
- Why PyTorch?

  Beginner friendly, well documented, good for fast development.
- How to install?

  `https://pytorch.org/get-started/locally/`
- This tutorial is for **PyTorch 1.3**

Basic Computations

Tensor

- Construct a Tensor

```python
x = torch.zeros(8, 3)
print(x)
```

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
```

```python
x = torch.rand(8, 3)
print(x)
```

```
tensor([[0.0984, 0.3671, 0.2543],
        [0.5016, 0.8017, 0.0918],
        [0.5081, 0.6020, 0.0867],
        [0.5313, 0.4571, 0.1624],
        [0.4231, 0.3993, 0.2713],
        [0.8978, 0.6039, 0.8519],
        [0.4829, 0.6648, 0.8295],
        [0.9060, 0.2132, 0.4110]])
```

Basic Computations

Operations

- Multiple syntaxes, e.g. Addition

```
y = torch.rand(8, 3)
print(x + y)
```

```
print(torch.add(x, y))
```

```
# providing an output tensor as argument
result = torch.empty(8, 3)
torch.add(x, y, out=result)
print(result)
```

```
# adds x to y
y.add_(x)
print(y)
```

- Other operations including transposing, indexing, slicing, linear algebra etc. at `https://pytorch.org/docs/stable/torch.html`

Basic Computations

Bridge to Numpy

- PyTorch $\rightarrow$ Numpy

```
a = torch.ones(5)
b = a.numpy()
```

- Numpy $\rightarrow$ PyTorch

```
a = np.ones(5)
b = torch.from_numpy(a)
```

- Tensors can only be converted to Numpy when they are on CPU

Basic Computations

Difference to Numpy

- GPU acceleration

```
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    z = x + y
    print(z)
    print(z.cpu())          # ``.to`` can change dtype
```

```
tensor([2.9218], device='cuda:0')
tensor([2.9218], dtype=torch.float64)
```

Basic Computations

Difference to Numpy

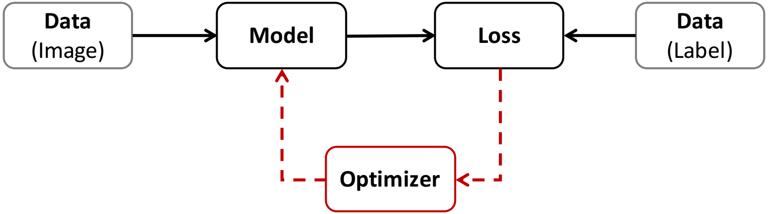
- GPU acceleration
- Automatic differentiation for all operations on Tensors

```
x = torch.ones(2, 2, requires_grad=True)
y = x + 2
z = y * y * 3
out = z.mean()

out.backward()

print(x.grad)
```

```
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
```

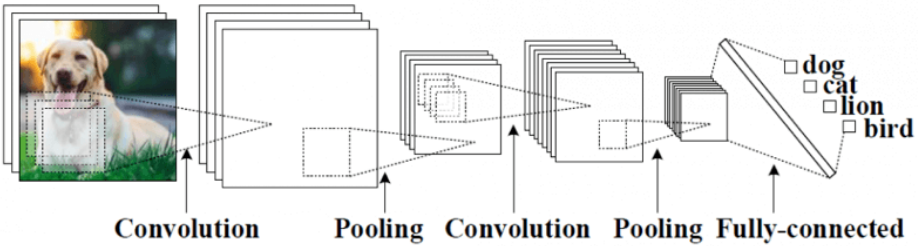
Neural Networks

- Model → `torch.nn.Module`
- Loss → `torch.nn.Module.Loss`
- Optimizer → `torch.optim`
- Data → `torch.utils.data`



Neural Networks

Model



Neural Networks

Model

- Define a model as a class that inherits from `torch.nn.Module`

```python
class Net(nn.Module):
```

- Define layers in the `__init__()` method

```python
    def __init__(self):
        ...
```

- Define computation flow given an input `x` in the `forward()` method

```python
    def forward(self, x):
        ...
```

- `backward()` is automatically defined

Neural Networks

Model

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        return x
```

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html#define-the-network

Neural Networks

Model

- PyTorch contains a branch of standard layers which are also subclasses of `torch.nn.Module`:
  - Convolution layers — `nn.Conv2d(`C_{in}`,` C_{out}`,` K`)`
  - Pooling layers — `nn.MaxPool2d(`K`)`
  - Non-linear activations — `nn.ReLU()`
  - Normalization layers — `nn.BatchNorm2d(`N`)`
  - Linear layers — `nn.Linear(`C_{in}`,`C_{out}`)`
  - ... — ...
- It is also easy implement custom layers: `https://pytorch.org/docs/stable/notes/extending.html#extending-torch-nn`

Loss

- Loss function returns a non-negative value J measuring the distance between network estimation and the ground truth
- PyTorch contains a branch of loss functions which are also subclasses of `torch.nn.Module`:
  - L1Loss
  - MSELoss
  - CrossEntroyLoss
  - NLLLoss
  - SmoothL1Loss
  - ...

Neural Networks

Loss

- Example of using a loss function

```
loss = nn.CrossEntropyLoss()
input = torch.randn(3, 5, requires_grad=True)
target = torch.empty(3, dtype=torch.long).random_(5)
output = loss(input, target)
output.backward()
```

Neural Networks

Optimizer

- Optimizer decides how to update the parameters in the model, e.g.
  $\theta = \theta - \eta \nabla J(\theta)$
- PyTorch implements a set of optimization algorithms in `torch.optim`:
  - SGD
  - SGD + Momentum
  - Adam
  - ...

Neural Networks

Optimizer

- 1. Construct an Optimizer

```
optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)
```

- 2. Take an optimization step for every batch/sample

```
for input, target in dataset:
    # clear saved gradients before computing gradient for the new batch
    optimizer.zero_grad()

    output = model(input)
    loss = loss_fn(output, target)

    loss.backward()

    # update parameters in model
    optimizer.step()
```

Data

- PyTorch provides `Dataset`,`DataLoader` in `torch.utils.data` that allows batching data, shuffling data and load data with multiple processes. Good tutorial at `https://pytorch.org/tutorials/beginner/data_loading_tutorial.html`
- For small scale of dataset it is fine to implement your own data loader.

Neural Networks

Saving Models

- Save/Load `state_dict` (Recommended)

```
# save
torch.save(model.state_dict(), PATH)
# load
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

- Save/Load entire model

```
# save
torch.save(model, PATH)
# load
# Model class must be defined somewhere
model = torch.load(PATH)
model.eval()
```

References

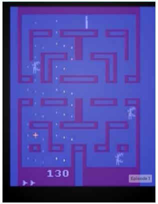
- PyTorch official tutorials: `https://pytorch.org/tutorials/`
- Stanford Course on Deep Learning for Computer Vision: `http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture08.pdf`
- NTU Machine Learning Course: `https://www.slideshare.net/lymanblueLin/pytorch-tutorial-for-ntu-machine-learing-course-2017`
- PyTorch tutorial with code examples: `https://github.com/MorvanZhou/PyTorch-Tutorial`

OpenAI Gym

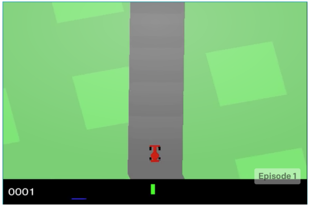
OpenAI Gym

- What is OpenAI Gym?
  A python based toolkit for developing and comparing RL algorithms.
- Why OpenAI Gym?
  Standardization of environments/benchmarks for RL algorithms.
- How to install?
  We provide a custom package with instructions.

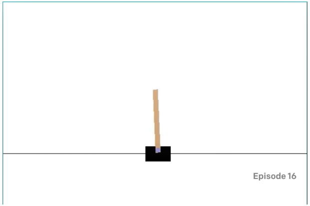
Examples



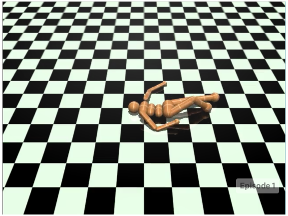
(a) Atari



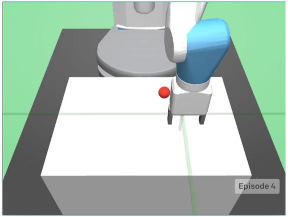
(b) Box2D



(c) Control



(d) MuJoCo

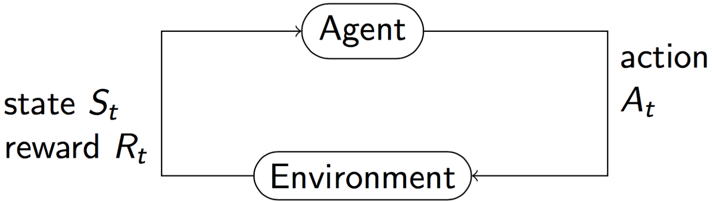


(e) Robotics

(f) Grid Worlds

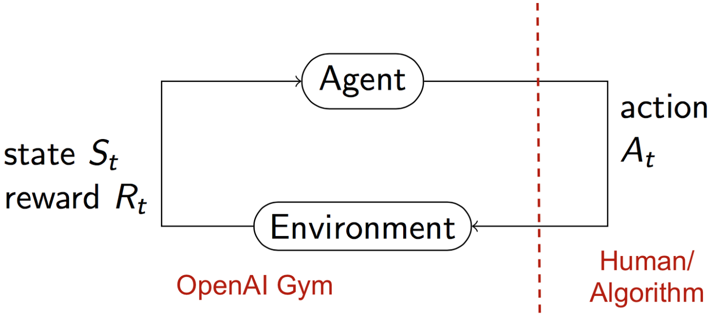
Reinforcement Learning

- 1. Create an Environment and initialize the State
- 2. Iteratively take an Action and observe the State, with the goal to maximize the Reward



Reinforcement Learning

- 1. Create an Environment and initialize the State
- 2. Iteratively take an Action and observe the State, with the goal to maximize the Reward



Work flow

```python
import gym
env = gym.make('CarRacing-v0')
env.reset()
for _ in range(1000):

    env.render()

    # take a random action
    env.step(env.action_space.sample())
```

Work flow

Create an Environment

- Each gym environment has a unique name
- To create an environment from the name use the

```
env = gym.make(env_name)
```

- For example, to create a CarRacing environment:

```
env = gym.make('CarRacing-v0')
```

Work flow

Initialize State

- Used to reinitialize a new episode
- Returns the initial state

```
init_state = env.reset()
```

Work flow

Take an action

- Performs the specified action and returns the resulting state
- The main method your agent interacts with

```
step(action) -> (next_state,
        reward,
        is_terminal,
        debug_info)
```

Work flow

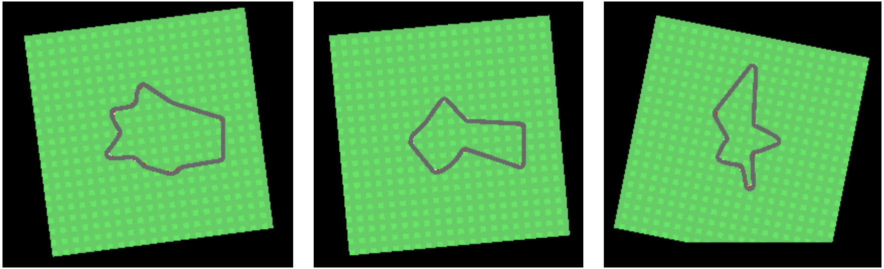
Take an observation: Render

- Optional method
- Used to display the state of your environment
- Useful for debugging and qualitatively comparing different agent policies

```
env.render()
```

Car Racing

- Randomly generated tracks



Car Racing

- `action_space`: three continous values, including steer, gas, brake

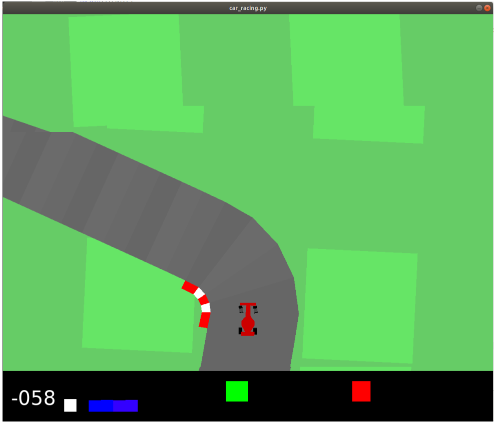
```
self.action_space = spaces.Box( low=np.array([-1,0,0]),
                                high=np.array([+1,+1,+1]),
                                dtype=np.float32)
```

- `observation_space`: color image

```
self.observation_space = spaces.Box(low=0, high=255,
                                    shape=(STATE_H, STATE_W, 3),
                                    dtype=np.uint8)
```

- `reward`: $R = N_{visited\_tile} * \frac{1000}{N_{all\_tile}} - N_{frame} * 0.1$

Car Racing



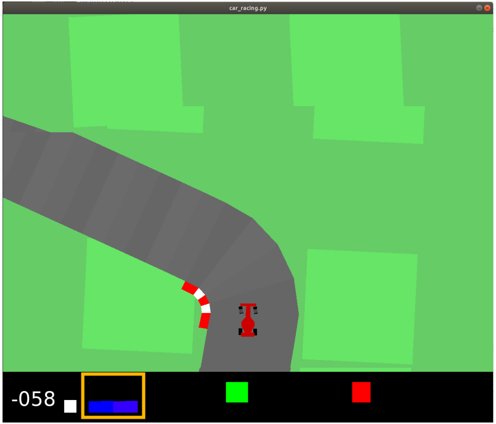
Car Racing

- Reward

Car Racing

- Reward
- Car speed

Car Racing



- Reward
- Car speed
- Wheel speed

Car Racing

- Reward
- Car speed
- Wheel speed
- Joint angle

Car Racing

- Reward
- Car speed
- Wheel speed
- Joint angle
- Angular Velocity

Car Racing

- We will create a leaderboard for the exercises
- Evaluation metrics:
  - $R = N_{visited\_tile} * \frac{1000}{N_{all\_tile}}$ given a fixed N_{frame}
  - $R = R - 100$ if the car get too far away from the track
- Submit your code and we will evaluate it on our server

References

- Official document: `https://gym.openai.com/docs`
- Source code: `https://github.com/openai/gym`
- `https://katefvision.github.io/10703_openai_gym_recitation.pdf`

Exercise 0

Exercise 0

Install PyTorch locally

- Recommended to install with Anaconda

PyTorch Build	Stable	Preview			
Your OS	Linux	Mac	Windows		
Package	Conda	Pip	LibTorch	Source	
Language	Python 2.7	Python 3.5	Python 3.6	Python 3.7	C++
CUDA	8.0	9.0	9.2	None	

Run this Command:

```
conda install pytorch torchvision -c pytorch
```

Exercise 0

Install OpenAI Gym locally

- Python 3.5+
- Download `sdc_gym.zip` from **ILIAS**, unzip and enter the folder
- Install the box2d package

```
pip3 install -e '.[box2d]'
```

- Please use the code we provided as there are some modifications compared to the official version.

Exercise 0

Work on Cluster

- Download the Singularity image and copy it to your home directories on the cluster: `https://owncloud.tuebingen.mpg.de/index.php/s/CbGdQrCfcpP4EFA/download`
  It is an environment that contains everything you need for the exercises of this lecture such as PyTorch, OpenAI Gym.
- Run the code under the Singularity environment:

```
singularity exec ~/sdc_gym.simg python your_python_file.py
```

Questions?