Assignment 4 (114 pts)

Policy Gradients

Instructions

- This is an individual assignment. You are **not allowed** to discuss the problems with other students.
- Part of this assignment will be autograded by gradescope. You can use it as immediate feedback to improve your answers. You can resubmit as many times as you want.
- All your solution, code, analysis, graphs, explanations should be done in this same notebook.
- Please make sure to execute all the cells before you submit the notebook to the gradescope. You will not get points for the plots if they are not generated already.
- Please **do not** change the random seeds
- Please start early. Some of the experiments take a lot of time to run on CPU.
- If you have questions regarding the assignment, you can ask for clarifications on Piazza. You should use the corresponding tag for this assignment.
- The deadline for submitting this assignment is **10:00 PM on Sunday, November 26, 2023**

This assignment has 4 parts. The goals of these parts are:

- **Part 1**: Implementing a parameterized (neural network) policy with PyTorch
- **Part 2**: Understanding and implementing the REINFORCE algorithm
- **Part 3**: Extending the REINFORCE algorithm with a baseline
- **Part 4**: Understanding and implementing Actor-Critic

When Submitting to GradeScope: Be sure to

1. Submit a .ipynb notebook to the Assignment 4 - Code section on Gradescope.
2. Submit a pdf version of the notebook to the Assignment 4 - Report entry.

Note: You can choose to submit responses in either English or French.

Before starting the assignment, make sure that you have downloaded all the tests related for the assignment and put them in the appropriate locations. If you run the next cell, we will set this all up automatically for you in a dataset called public, which contains the test cases.

Installing Dependencies

In [1]:
```
!pip install otter-grader
!rm -rf public
!git clone https://github.com/chandar-lab/INF8250ae-assignments-2023 public
```

```
Requirement already satisfied: otter-grader in /usr/local/lib/python3.10/dist-packages
(5.2.2)
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-packages (from ott
```

```
er-grader) (0.3.7)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from o
tter-grader) (3.1.2)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from
otter-grader) (5.9.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from o
tter-grader) (1.5.3)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from o
tter-grader) (6.0.1)
Requirement already satisfied: python-on-whales in /usr/local/lib/python3.10/dist-packag
es (from otter-grader) (0.65.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
otter-grader) (2.31.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packages (from ot
ter-grader) (1.15.0)
Requirement already satisfied: jupytext in /usr/local/lib/python3.10/dist-packages (from
otter-grader) (1.15.2)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from ot
ter-grader) (8.1.7)
Requirement already satisfied: fica>=0.3.0 in /usr/local/lib/python3.10/dist-packages (f
rom otter-grader) (0.3.1)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (from
otter-grader) (7.34.0)
Requirement already satisfied: astunparse in /usr/local/lib/python3.10/dist-packages (fr
om otter-grader) (1.6.3)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (fr
om otter-grader) (7.7.1)
Requirement already satisfied: ipylab in /usr/local/lib/python3.10/dist-packages (from o
tter-grader) (1.0.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (fro
m otter-grader) (6.5.4)
Requirement already satisfied: docutils in /usr/local/lib/python3.10/dist-packages (from
fica>=0.3.0->otter-grader) (0.18.1)
Requirement already satisfied: sphinx in /usr/local/lib/python3.10/dist-packages (from f
ica>=0.3.0->otter-grader) (5.0.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-pack
ages (from astunparse->otter-grader) (0.41.2)
Requirement already satisfied: six<2.0,>=1.6.1 in /usr/local/lib/python3.10/dist-package
s (from astunparse->otter-grader) (1.16.0)
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.10/dist-packag
es (from ipywidgets->otter-grader) (6.26.0)
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.10/dist
-packages (from ipywidgets->otter-grader) (0.2.0)
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.10/dist-packag
es (from ipywidgets->otter-grader) (5.7.1)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/di
st-packages (from ipywidgets->otter-grader) (3.6.5)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/di
st-packages (from ipywidgets->otter-grader) (3.0.8)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packag
es (from ipython->otter-grader) (67.7.2)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (fr
om ipython->otter-grader) (0.19.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (fro
m ipython->otter-grader) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (f
rom ipython->otter-grader) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/loca
l/lib/python3.10/dist-packages (from ipython->otter-grader) (3.0.39)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from
ipython->otter-grader) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from
ipython->otter-grader) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packa
ges (from ipython->otter-grader) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (f
```

```
rom ipython->otter-grader) (4.8.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-package
s (from jinja2->otter-grader) (2.1.3)
Requirement already satisfied: toml in /usr/local/lib/python3.10/dist-packages (from jup
ytext->otter-grader) (0.10.2)
Requirement already satisfied: markdown-it-py>=1.0.0 in /usr/local/lib/python3.10/dist-p
ackages (from jupytext->otter-grader) (3.0.0)
Requirement already satisfied: mdit-py-plugins in /usr/local/lib/python3.10/dist-package
s (from jupytext->otter-grader) (0.4.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbc
onvert->otter-grader) (4.9.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages
(from nbconvert->otter-grader) (4.11.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from n
bconvert->otter-grader) (6.0.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (fr
om nbconvert->otter-grader) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-pack
ages (from nbconvert->otter-grader) (0.4)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert->otter-grader) (5.3.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-pac
kages (from nbconvert->otter-grader) (0.2.2)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packa
ges (from nbconvert->otter-grader) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-package
s (from nbconvert->otter-grader) (0.8.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fro
m nbconvert->otter-grader) (23.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-pa
ckages (from nbconvert->otter-grader) (1.5.0)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from
nbconvert->otter-grader) (1.2.1)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-packages
(from nbformat->otter-grader) (2.18.0)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-package
s (from nbformat->otter-grader) (4.19.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->otter-grader) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages
(from pandas->otter-grader) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages
(from pandas->otter-grader) (1.23.5)
Requirement already satisfied: pydantic!=2.0.*,<3,>=1.5 in /usr/local/lib/python3.10/dis
t-packages (from python-on-whales->otter-grader) (1.10.12)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pyt
hon-on-whales->otter-grader) (4.66.1)
Requirement already satisfied: typer>=0.4.1 in /usr/local/lib/python3.10/dist-packages
(from python-on-whales->otter-grader) (0.9.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packa
ges (from python-on-whales->otter-grader) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
t-packages (from requests->otter-grader) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
(from requests->otter-grader) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
ages (from requests->otter-grader) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
ages (from requests->otter-grader) (2023.7.22)
Requirement already satisfied: comm>=0.1.1 in /usr/local/lib/python3.10/dist-packages (f
rom ipykernel>=4.5.1->ipywidgets->otter-grader) (0.2.0)
Requirement already satisfied: debugpy>=1.6.5 in /usr/local/lib/python3.10/dist-packages
(from ipykernel>=4.5.1->ipywidgets->otter-grader) (1.6.6)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.10/dist-
packages (from ipykernel>=4.5.1->ipywidgets->otter-grader) (6.1.12)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
```

```
(from ipykernel>=4.5.1->ipywidgets->otter-grader) (1.5.7)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from i
pykernel>=4.5.1->ipywidgets->otter-grader) (5.9.5)
Requirement already satisfied: pyzmq>=20 in /usr/local/lib/python3.10/dist-packages (fro
m ipykernel>=4.5.1->ipywidgets->otter-grader) (23.2.1)
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.10/dist-packages
(from ipykernel>=4.5.1->ipywidgets->otter-grader) (6.3.2)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-pac
kages (from jedi>=0.16->ipython->otter-grader) (0.8.3)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages
(from jsonschema>=2.6->nbformat->otter-grader) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/py
thon3.10/dist-packages (from jsonschema>=2.6->nbformat->otter-grader) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-pac
kages (from jsonschema>=2.6->nbformat->otter-grader) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages
(from jsonschema>=2.6->nbformat->otter-grader) (0.10.2)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packa
ges (from jupyter-core>=4.7->nbconvert->otter-grader) (3.10.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (fr
om markdown-it-py>=1.0.0->jupytext->otter-grader) (0.1.2)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-package
s (from pexpect>4.3->ipython->otter-grader) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from
prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->otter-grader) (0.2.6)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.10/dist-package
s (from widgetsnbextension~=3.6.0->ipywidgets->otter-grader) (6.5.5)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages
(from beautifulsoup4->nbconvert->otter-grader) (2.5)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages
(from bleach->nbconvert->otter-grader) (0.5.1)
Requirement already satisfied: sphinxcontrib-applehelp in /usr/local/lib/python3.10/dist
-packages (from sphinx->fica>=0.3.0->otter-grader) (1.0.7)
Requirement already satisfied: sphinxcontrib-devhelp in /usr/local/lib/python3.10/dist-p
ackages (from sphinx->fica>=0.3.0->otter-grader) (1.0.5)
Requirement already satisfied: sphinxcontrib-jsmath in /usr/local/lib/python3.10/dist-pa
ckages (from sphinx->fica>=0.3.0->otter-grader) (1.0.1)
Requirement already satisfied: sphinxcontrib-htmlhelp>=2.0.0 in /usr/local/lib/python3.1
0/dist-packages (from sphinx->fica>=0.3.0->otter-grader) (2.0.4)
Requirement already satisfied: sphinxcontrib-serializinghtml>=1.1.5 in /usr/local/lib/py
thon3.10/dist-packages (from sphinx->fica>=0.3.0->otter-grader) (1.1.9)
Requirement already satisfied: sphinxcontrib-qthelp in /usr/local/lib/python3.10/dist-pa
ckages (from sphinx->fica>=0.3.0->otter-grader) (1.0.6)
Requirement already satisfied: snowballstemmer>=1.1 in /usr/local/lib/python3.10/dist-pa
ckages (from sphinx->fica>=0.3.0->otter-grader) (2.2.0)
Requirement already satisfied: babel>=1.3 in /usr/local/lib/python3.10/dist-packages (fr
om sphinx->fica>=0.3.0->otter-grader) (2.12.1)
Requirement already satisfied: alabaster<0.8,>=0.7 in /usr/local/lib/python3.10/dist-pac
kages (from sphinx->fica>=0.3.0->otter-grader) (0.7.13)
Requirement already satisfied: imagesize in /usr/local/lib/python3.10/dist-packages (fro
m sphinx->fica>=0.3.0->otter-grader) (1.4.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (f
rom notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->otter-grader) (23.1.0)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packa
ges (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->otter-grader) (1.8.2)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packag
es (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->otter-grader) (0.17.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packa
ges (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->otter-grader) (0.17.1)
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packag
es (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->otter-grader) (1.0.0)
Requirement already satisfied: jupyter-server>=1.8 in /usr/local/lib/python3.10/dist-pac
kages (from nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->ot
ter-grader) (1.24.0)
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-pa
ckages (from nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->o
```

```
tter-grader) (0.2.3)
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-pa
ckages (from argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets->otter-
grader) (21.2.0)
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-package
s (from jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.
0->ipywidgets->otter-grader) (3.7.1)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packag
es (from jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.
6.0->ipywidgets->otter-grader) (1.6.2)
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (f
rom argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywi
dgets->otter-grader) (1.15.1)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages
(from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnb
extension~=3.6.0->ipywidgets->otter-grader) (1.3.0)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages
(from anyio<4,>=3.1.0->jupyter-server>=1.8->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnb
extension~=3.6.0->ipywidgets->otter-grader) (1.1.3)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (fro
m cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbextension~=
3.6.0->ipywidgets->otter-grader) (2.21)
Cloning into 'public'...
remote: Enumerating objects: 124, done.
remote: Counting objects: 100% (124/124), done.
remote: Compressing objects: 100% (76/76), done.
remote: Total 124 (delta 53), reused 105 (delta 37), pack-reused 0
Receiving objects: 100% (124/124), 1.03 MiB | 6.01 MiB/s, done.
Resolving deltas: 100% (53/53), done.
```

In [2]:

```
!pip install -U pygame --user
!pip install gymnasium
!pip install matplotlib
!pip install tqdm
```

```
Requirement already satisfied: pygame in /root/.local/lib/python3.10/site-packages (2.5.
2)
Requirement already satisfied: gymnasium in /usr/local/lib/python3.10/dist-packages (0.2
9.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages
(from gymnasium) (1.23.5)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-pack
ages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.10/dis
t-packages (from gymnasium) (4.5.0)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.10/
dist-packages (from gymnasium) (0.0.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.
7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages
(from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
ges (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (f
rom matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-package
s (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pa
```

```
ckages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.66.1)
```

In [3]:
```
!apt-get install x11-utils > /dev/null 2>&1
!pip install pyglet > /dev/null 2>&1
!apt-get install -y xvfb python-opengl > /dev/null 2>&1
!pip install pyvirtualdisplay > /dev/null 2>&1
!pip install pillow
```

```
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (9.4.0)
```

Importing Libraries

In [4]:
```
import otter
grader = otter.Notebook(colab=True, tests_dir='./public/a4/tests')
```

In [5]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import torch
import torch.distributions as torchdist
import torch.nn as nn
import torch.nn.functional as F
import warnings
import functools
import os
import matplotlib.pyplot as plt
from IPython import display as ipythondisplay

import gymnasium as gym
from tqdm import tqdm
from PIL import Image
from IPython.display import Image as IPImage, display

device = "gpu" if torch.cuda.is_available() else "cpu"
warnings.filterwarnings('ignore')
torch.manual_seed(0)
np.random.seed(0)
import os
```

In [6]:
```
GRADESCOPE_ENV_VAR = "RUNNING_IN_GRADESCOPE"

def running_in_gradescope():
    var = os.getenv(GRADESCOPE_ENV_VAR)
    if var is None:
        return False
    return var == 'yes'
```

The Environment

For this assignment, we will use `CartPole-v1` from OpenAI Gymnasium. In this environment, the goal is to balance an inverted pendulum on a cart by moving the cart laterally. The state of the agent has four components:

- The horizontal position of the cart, x
- The velocity of the cart, $\dot{x}$
- The angle of the pendulum, measured relative to the vertical axis, θ

- The angular velocity of the pendulum, $\dot{\theta}$

There are two actions in the action space:

- 0: push cart to the left
- 1: push cart to the right

The agent receives a reward of 1 at each timestep, and the episode ends when the pendulum drops too far ($|\theta|$ is more than 12^o) or when the cart goes out of bounds. Also, the environment truncates after 500 steps if it hasn't already terminated, so the greatest possible return is 200.

Part 0. Video Rendering

In [7]:

```
def render_video(env, policy = None, steps = 50):
    env.action_space.seed(0)
    obs, _ = env.reset(seed = 0)
    rewards = []
    image_list = []
    for i in range(steps):
        if policy == None:
            action = env.action_space.sample()
        else:
            action = policy.action(obs)
        obs, reward, terminated, truncated, info = env.step(action)
        rewards.append(reward)
        screen = env.render()
        image_list.append(screen)

        done = terminated or truncated
        if done:
            print("Return: ", sum(rewards))
            break
        env.close()
    pil_images = [Image.fromarray(image) for image in image_list]
    pil_images[0].save("output.gif", save_all=True, append_images=pil_images[1:], durati
    display(IPImage("output.gif"))
```

Now, let us see how a random policy performs.

In [8]:

```
env = gym.make('CartPole-v1', render_mode="rgb_array", max_episode_steps=200)
render_video(env)
```

```
error: XDG_RUNTIME_DIR not set in the environment.
Return:  18.0
<IPython.core.display.Image object>
```

Notice that the random agent cannot balance the pendulum even for 20 time steps! In this assignment you will be implementing policy gradient algorithms to learn a better policy.

Part 1. Parameterized Policy Network (17 pts total)

In this assignment, we will be studying Policy Gradient algorithms. In these algorithms, rather than using action-values to select actions, the policy itself is parameterized (in our case, by a neural network), and the policy is optimized directly via gradient ascent (although for practical purposes, we will be optimizing the negative of the objective via gradient descent).

We will use a neural network to represent the policy here. The input to the neural network is a state, and the output should encode a probability distribution over the action space. Our environment has a discrete action space, so the policy should output parameters for a *Categorical distribution*.

1a: The Policy Network (5 pts)

As a first step, fill in the `policy_init_network` function, which should return a torch neural net that will be to produce policy distributions for input states. You are free to experiment with different neural network architectures later, but for this assignment we recommend the following. Using `torch.nn.Sequential`, make a multilayer perceptron (MLP) with the following layers:

1. A linear layer of size `(state space dimension, 32)`, followed by a ReLU activation
2. A linear layer of size `(32, 32)`, followed by a ReLU activation
3. A linear layer of size `(32, number of actions)` followed by a Softmax activation, to make a probability distribution over actions.

In [9]:
```
def policy_init_network(env: gym.Env) -> nn.Module:
    # Your code here
    # --------------------------------
    return torch.nn.Sequential(
        torch.nn.Linear(env.observation_space.shape[0], 32),
        torch.nn.ReLU(),
        torch.nn.Linear(32, 32),
        torch.nn.ReLU(),
        torch.nn.Linear(32, env.action_space.n),
        torch.nn.Softmax()
    )
    # --------------------------------
```

In [10]:
```
grader.check("question 1a")
```

Out[10]:

`question 1a` passed! □

1b: The Policy Class (12 pts)

In this part, we will build a class to represent the parameterized policy. This will be done in a few steps. In the constructor of the `Policy` class, initialize the variable `opt`, which will be used to optimize the policy parameters. This variable is a `torch.optim.Optimizer`. The learning rate is 10^{-3}.

Part I: Optimizer (2 pts)

Initialize the attribute `self.opt` to Adam optimizer with learning rate 10^{-3}

Part II: The policy distribution (5 pts)

Fill in the `dist` method. This method takes as input a state and outputs a torch `Distribution` over actions.

Part III: Sampling actions (5 pts)

Fill in the `action` method. This method samples an action from the policy given a state.

In [11]:
```python
class Policy:
    def __init__(
        self, env: gym.Env, network: nn.Module, discount=0.99, name="Abstract Policy"
    ):
        self.name = name
        self.network = network
        self.discount = discount

        self.env = env
        self.obs_dim = env.observation_space.shape[0]
        self.n_actions = env.action_space.n
        # Your code here (Part I)
        # ========================
        self.opt = torch.optim.Adam(self.network.parameters(), lr=1e-3)
        # ========================

    def distribution(self, x: np.ndarray) -> torchdist.distribution.Distribution:
        """
        Get the distribution over actions for a given state

        """
        dist = None

        # Your code here (Part II)
        # ========================

        x = torch.Tensor(x).float()
        probs = self.network(x).unsqueeze(0)

        dist = torchdist.Categorical(probs)

        # ========================
        return dist

    def action(self, x: np.ndarray) -> int:
        """
        Sample an action from the policy at a given state

        Input: a state encoded as a numpy array
        Output: an action encoded as an int
        """
        action = None
        # Your code here (Part III)
        # ========================
        dist = self.distribution(x)
        action = dist.sample().item()
        # ========================
        return action

    def update(self, states, actions, rewards, dones) -> float:
        raise NotImplementedError
```

In [12]:
```python
grader.check("question 1b")
```

Out[12]:

question 1b passed! □

Generating Episode

Now, the following function rolls out an episode in the environment with the policy. The function should return `(states, actions, rewards, terminated, truncated)` where

1. `states` is a record of the states observed over the course of the episodes.
2. `actions` is a record of the actions taken.
3. `rewards` is a record of the rewards earned.
4. `terminated` is an array of `bool`s that marks the termination of the episode.
5. `truncated` is an array of `bool`s that marks the truncation of the episode.

Note that in this function, we do not append the final state.

In [13]:
```python
def generate_episode(env: gym.Env, policy: Policy):
    """
    Generates an episode given an environment and policy
    Inputs:
        env - Gymnasium environment
        policy - policy for generating episode
    Returns

    """
    # Initialize lists
    states = []
    actions = []
    rewards = []
    terminated = []
    truncated = []

    # Reset environment
    state, _ = env.reset(seed = 0)
    done = False

    # Loop until end of episode
    while not done:
        states.append(state)
        # Get action
        action = policy.action(state)
        actions.append(action)
        # Take step
        state, reward, term, trunc,  _ = env.step(action)
        done = term or trunc
        rewards.append(reward)
        terminated.append(term)
        truncated.append(trunc)
    states = np.array(states)
    actions = np.array(actions)
    rewards = np.array(rewards)
    terminated = np.array(terminated)
    truncated = np.array(truncated)
    return (states, actions, rewards, terminated, truncated)
```

Part 2. REINFORCE (17 pts total)

In this section, you will implement the REINFORCE algorithm.

2a: Discounting Rewards (10 pts total)

This problem has 2 parts.

Recall the form of the REINFORCE policy gradient:

$$\nabla_\theta J(\theta) = \sum_{k=0}^{T} \mathbf{E}\left\{G^{\pi_\theta} \nabla_\theta \log \pi_\theta(a_k \mid s_k)\right\}$$

Here π_θ is the parameterized (neural net) policy with parameters θ, and G^{π_θ} is the random variable corresponding to the discounted return induced by following π_θ. Note that at timestep k, action a_k had no influence on rewards incurred before timestep k. For this reason, it is generally preferred to compute the following,

$$\widehat{\nabla}_\theta J(\theta) = \sum_{k=0}^{T} \mathbf{E}\left\{G_k^{\pi_\theta} \nabla_\theta \log \pi_\theta(a_k \mid s_k)\right\}$$

where

$$G_k^{\pi_\theta} = \sum_{t=k}^{T} \gamma^{t-k} r(s_k, a_k)$$

Part I (5 pts):

Question: Why do you think it is preferred to substitute $\nabla_\theta J(\theta)$ for $\widehat{\nabla}_\theta J(\theta)$ in policy gradient algorithms?

The substitution of $\nabla_\theta J(\theta)$ for $\widehat{\nabla}_\theta J(\theta)$ in policy gradient algorithms is preferred for it's lower variance its contribution to variance reduction, enhancing the convergence properties.

Part II (5 points):

Implement the function `discounted_returns`, which computes the values $(G_1^{\pi_\theta}, G_2^{\pi_\theta}, \ldots)$ for a given sequence of rewards.

The function takes three arguments:

1. `rewards`: An array of rewards, which may have been collected over several trajectories.
2. `dones`: An array of `bool`s, which mark where trajectories ended -- when either of `terminated` or `truncated` is `True`.
3. `discount`: The discount factor.

The output of the function should be a list of the same length as `rewards` containing the cumulative discounted future returns starting at each step in the reward sequence. Mathematically, for some index k, if T is the first index after k for which `dones[T] = True`, then

$$\mathtt{returns}[k] = \mathtt{rewards}[k] + \gamma \mathtt{rewards}[k+1] + \cdots + \gamma^{T-k} \mathtt{rewards}[T]$$

For example, suppose we gather data from two trajectories, which had rewards `[1,2,3]` and `[4, 2, 1]` respectively. Then:

- `rewards = [1,2,3,4,2,1]`
- `dones = [False, False, True, False, False, True]`

For `discount = 0.5`, the output should be `[2.75, 3.5, 3, 5.25, 2.5, 1]`.

NOTE: The output should be a numpy array.

In [14]:
```
def discounted_returns(
    rewards: np.ndarray, dones: np.ndarray, discount: float
) -> np.ndarray:
    """
    Compute discounted returns given rewards and terminateds
    Inputs:
        rewards - numpy array of reward values
        dones - numpy array consisting of boolean values for whether the episode has ter
        discount - discount factor
    Returns:
        returns - numpy array discounted returns
    """
    # Your code here
    # ================================
    returns = np.ndarray((len(rewards),), dtype=np.float64, buffer=np.array(rewards))
    for k in range(len(rewards) - 1, 0, -1):
        returns[k - 1] = rewards[k - 1] + discount * returns[k] * (1 - dones[k - 1])
    # ================================
    return returns
```

In [15]:
```
grader.check("question 2a")
```

Out[15]:

question 2a passed! □

2b: The REINFORCE Update (7 pts)

Finally, we'll implement REINFORCE. Fill in the `update` method for the `REINFORCEPolicy` class below. This method takes the following inputs:

1. `states`: An array of observed states.
2. `actions`: An array of actions taken at the corresponding `states`.
3. `rewards`: An array of rewards received, where `rewards[k]` is the reward for taking actions `actions[k]` at state `states[k]`.
4. `dones`: An array of `bool`s marking the end of episodes.

This method should perform the following:

- Compute the average policy gradient "loss", which is $-\sum_{n=1}^{T} G_n^{\pi_\theta} \log \pi_\theta(a_n \mid s_n)$, averaged over all trajectories
- Compute the policy gradient
- Update the policy parameters

The method should return a dictionary that contains information from the update. For now, the dictionary should only have one entry with key `'policy_loss'` that contains a scalar loss from the policy gradient computation.

In [16]:
```
class REINFORCEPolicy(Policy):
    def __init__(
        self, env: gym.Env, network: nn.Module, discount=0.99, name="Plain REINFORCE"
    ):
        super().__init__(env, network, discount=discount, name=name)

    """
    Perform a gradient update
    Inputs:
        states, actions, rewards, dones: Output from generate_episode function
    Returns:
```

```
        Dictionary with the following keys:
        - "policy_loss": float of the policy gradient loss (the quantity whose gradient
    """

    def update(self, states, actions, rewards, dones) -> dict:
        loss_dict = {}
        # Your code here"
        # ======================
        G = discounted_returns(rewards, dones, self.discount)
        G = torch.Tensor(G)
        loss = -(G * self.distribution(states).log_prob(torch.Tensor(actions))).mean()

        self.opt.zero_grad()
        loss.backward()
        self.opt.step()

        loss_dict["policy_loss"] = loss.item()
        # ======================
        return loss_dict
```

In [17]:
```
grader.check("question 2b")
```

Out[17]:

question 2b passed! ▯

Part 3. REINFORCE with Baseline (41 pts total)

When using a baseline in REINFORCE, the policy gradient formula is modified to the following,

$$\nabla_\theta J(\theta) = \sum_{k=0}^{T} \mathbf{E}\left\{(G^{\pi_\theta} - b(s_k))\nabla_\theta \log \pi_\theta(a_k \mid s_k)\right\}$$

for some function $b : S \to \mathbf{R}$.

3a: Understanding the baseline (12 pts)

1. **(3 pts)** What purpose does the baseline serve?
2. **(3 pts)** If the baseline is a constant (that is, $b(s_1) = b(s_2)$ for any pair of states (s_1, s_2)), should we expect the performance of REINFORCE with this baseline to be any different from standard REINFORCE?
3. **(3 pts)** Why can't the baseline be a function of the action as well as the state?
4. **(3 pts)** Does the inclusion of an arbitrary baseline always help?

1. The baseline plays a crucial role in diminishing the variance of policy gradient estimates. Subtracting the baseline value from computed returns results in more stable and less noisy updates during training.

2. When the baseline remains a constant offset across all states, its impact on overall performance diminishes. In such cases, the algorithm's behavior closely resembles standard REINFORCE without a baseline.

3. Restricting the baseline to be solely a function of the state simplifies the policy gradient to $\sum_a b(s)\nabla\pi(a|s;\theta) = b(s)\sum_a \nabla\pi(a|s;\theta) = b(s) \times 0 = 0$. Thus, introducing action dependency in the baseline would alter the policy gradient, deviating from the intended behavior.

4. While a well-chosen baseline can effectively reduce variance and enhance training stability, an arbitrary or poorly selected baseline may not offer these benefits. The baseline's efficacy hinges on its capacity to capture return variance and, consequently, reduce gradient estimate variance. A baseline that fails to improve performance could manifest as a constant baseline for all states, which is suboptimal.

3b: The Value Function (5 pts)

In our experiments, we will use the value function as our baseline. It will be necessary to learn the value function from data, so our baseline will have the form

$$b(s) = V_\phi^{\pi_\theta}(s)$$

where ϕ denotes the parameters of the value function.

Fill in the code for the construction of the value function neural net in `value_init_network`. The network architecture should be **similar** to that of the policy network besides the output layer.

In [18]:
```
def value_init_network(env: gym.Env) -> nn.Module:
    # Your code here
    # ===========================================================
    return torch.nn.Sequential(
        torch.nn.Linear(env.observation_space.shape[0], 32),
        torch.nn.ReLU(),
        torch.nn.Linear(32, 32),
        torch.nn.ReLU(),
        torch.nn.Linear(32, 1),
    )
# ===========================================================
```

In [19]:
```
grader.check("question 3b")
```

Out[19]:

`question 3b` passed! □

3c: REINFORCE with Baseline (9 pts)

Fill in the constructor and the `update` method for `REINFORCEWithBaselinePolicy`.

The constructor should do set two variables:

- `self.value_network`: the value function neural network
- `self.value_opt`: the `torch.optim.Optimizer` for the value function parameters. Use Adam optimizer and a learning rate of 2×10^{-3} for the value optimizer.

This method should perform the following:

- Compute the "policy gradient loss", using the value predictions from the value function network instead of the Monte Carlo return estimates
- Compute the policy gradient, again using the value predictions from the value function network instead of the Monte Carlo return estimates
- Update the policy parameters
- Compute the "value loss", which is mean squared difference between the Monte Carlo return estimates and the value function network predictions at each state in the trajectory

- Update the value function network parameters

As with the standard REINFORCE case, the `update` method returns a dictionary with a key `'policy_loss'` reflecting the loss w.r.t. the policy gradient objective. For `REINFORCEWithBaselinePolicy`'s `update` method, however, the dictionary should also have a key `'value_loss'` reflecting the loss w.r.t. the value function error.

In [20]:
```
class REINFORCEWithBaselinePolicy(Policy):
    def __init__(
        self,
        env: gym.Env,
        policy_network: nn.Module,
        value_network: nn.Module,
        discount=0.99,
        name="REINFORCE with Baseline",
    ):
        super().__init__(env, policy_network, discount=discount, name=name)
        # Your code here
        ## Initialize value network and optimizer
        # ===========================================================
        self.value_network = value_network
        self.value_opt = torch.optim.Adam(self.value_network.parameters(), lr=2e-3)
        # ===========================================================

    """
    Perform a gradient update
    Inputs:
        states, actions, rewards, dones: Output from rollout method
    Returns:
        Dictionary with the following keys:
        - "policy_loss": float of the policy gradient loss (the quantity whose gradient
        - "value_loss": float of the squared TD error
    """

    def update(self, states, actions, rewards, dones) -> dict:
        loss_dict = {}
        # Your code here
        # ======================
        actions = torch.Tensor(actions)
        G = torch.tensor(discounted_returns(rewards, dones, self.discount))
        values = self.value_network(torch.Tensor(states)).squeeze()
        policy_loss = -(
            (G - values.detach()) * self.distribution(states).log_prob(actions)
        ).mean()

        self.opt.zero_grad()
        policy_loss.backward()
        self.opt.step()

        values = self.value_network(torch.Tensor(states)).squeeze()
        value_loss = torch.pow(G - values, 2).mean()

        self.value_opt.zero_grad()
        value_loss.backward()
        self.value_opt.step()

        loss_dict["policy_loss"] = policy_loss.item()
        loss_dict["value_loss"] = value_loss.item()

        # ======================
        return loss_dict
```

In [21]:
```
grader.check("question 3c")
```

Out[21]:

question 3c passed! □

3d: Experiments (15 pts)

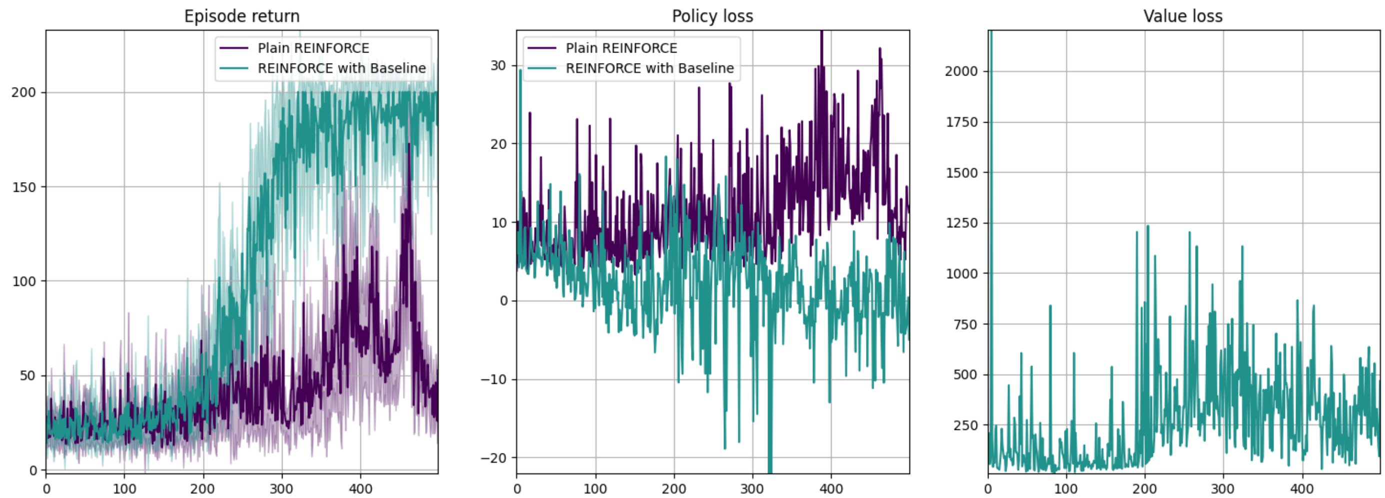
The code below will train agents with REINFORCE with and without the value function baseline. Think about how you expect the return and loss curves to behave with and without the baseline.

In [22]:

```
env = gym.make('CartPole-v1', render_mode="rgb_array", max_episode_steps=200)
agents = [
    REINFORCEPolicy(env, policy_init_network(env)),
    REINFORCEWithBaselinePolicy(env, policy_init_network(env), value_init_network(env))
]

gradient_steps = 500
scores = [np.zeros(gradient_steps) for _ in agents]
stds = [np.zeros(gradient_steps) for _ in agents]

test_runs = 5

def rollout_score(env, policy):
    _, _, rewards, _, _ = generate_episode(env, policy)
    return np.sum(rewards)

gs = list(range(gradient_steps))

cmap = plt.get_cmap('viridis')
plt.figure()
fig, (ret_ax, loss_ax, value_ax) = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

value_losses = []

if not running_in_gradescope():
    for i in range(len(agents)):
        reinforce_policy = agents[i]
        print(f"Training {reinforce_policy.name}")
        losses = []
        for g in tqdm(range(gradient_steps)):
            states, actions, rewards, terminated, truncated = generate_episode(env, rein
            dones = [term or trunc for (term, trunc) in zip(terminated, truncated)]
            loss = reinforce_policy.update(states, actions, rewards, dones)
            losses.append(loss['policy_loss'])
            if 'value_loss' in loss.keys():
                value_losses.append(loss['value_loss'])
            res = [rollout_score(env, reinforce_policy) for _ in range(test_runs)]
            scores[i][g] = np.mean(res)
            stds[i][g] = np.std(res)
        color = cmap(i / len(agents))
        ret_ax.plot(gs, scores[i], label=reinforce_policy.name, color = color)
        ret_ax.fill_between(gs, scores[i] - stds[i], scores[i] + stds[i], alpha=0.3, col
        loss_ax.plot(gs, losses, label=reinforce_policy.name, color = color)
    ret_ax.legend()
    ret_ax.grid(True)
    ret_ax.margins(0)
    ret_ax.set_title('Episode return')
    loss_ax.legend()
    loss_ax.grid(True)
    loss_ax.margins(0)
    loss_ax.set_title("Policy loss")
    value_ax.plot(gs, value_losses, color = color)
    value_ax.grid(True)
    value_ax.margins(0)
```

```
    value_ax.set_title("Value loss")
    plt.show()
plt.close('all')
```

```
Training Plain REINFORCE
100%|██████████| 500/500 [00:46<00:00, 10.85it/s]
Training REINFORCE with Baseline
100%|██████████| 500/500 [01:53<00:00,  4.41it/s]
<Figure size 640x480 with 0 Axes>
```



Visualization

REINFORCE

In [23]:
```
render_video(env, agents[0], steps = 200)
```

```
error: XDG_RUNTIME_DIR not set in the environment.
Return:  30.0
<IPython.core.display.Image object>
```

REINFORCE with Baseline

In [24]:
```
render_video(env, agents[1], steps = 200)
```

```
Return:  200.0
<IPython.core.display.Image object>
```

Analysis (5pts)

In your experiments, how did the use of the value function baseline affect your results? Explain the results you observed. Also, observe the visualizations above and qualitatively comment the nature of the policies obtained from the two agents (i.e., with and without baseline).

Initially, both agents exhibited similar episode returns, but the one with the baseline consistently surpassed its counterpart, showcasing accelerated and sustained performance improvements. Notably, the agent with the baseline also demonstrated a lower policy loss throughout training, indicative of more stable and aligned policy updates. The agent with the baseline also appeared to learn a more stable policy, as evidenced by its smoother episode return curve. In contrast, the agent without the baseline exhibited more erratic behavior.

4. Actor-Critic (39 pts total)

Finally, we will experiment with an *actor-critic* algorithm. Recall that the gradient rule for REINFORCE with the value function baseline has the following form,

$$\nabla_\theta J(\theta) = \sum_{k=0}^{T} \mathbf{E}\left\{(G^{\pi_\theta} - V_\phi^{\pi_\theta}(s_k))\nabla_\theta \log \pi_\theta(a_k \mid s_k)\right\}$$

Note that

$$\mathbf{E}\left\{G^{\pi_\theta} \mid s_0 = s\right\} = \mathbf{E}_{a\sim\pi(\cdot|s),s'\sim P(\cdot|s,a)}\left\{r(s,a) + \gamma V^{\pi_\theta}(s')\right\}$$

Because of this, actor-critic algorithms estimate G^{π_θ} by $r(s,a) + \gamma V^{\pi_\theta}(s')$. Thus, we can compute one gradient *per environment step*, since we no longer need data from the entire trajectory to estimate G^{π_θ}. The gradient rule for the policy network (actor) is

$$\nabla_\theta J_{\text{actor}}(\theta) = \mathbf{E}\left\{(r_k + \gamma V_\phi^{\pi_\theta}(s_{k+1}) - V_\phi^{\pi_\theta}(s_k))\nabla_\theta \log \pi_\theta(a_k \mid s_k)\right\}$$

for the policy parameters. The value network (critic) is trained to minimize the mean squared TD error:

$$\nabla_\phi J_{\text{critic}}(\phi) = \frac{1}{2}\left(V_\phi^{\pi_\theta}(s_k) - \texttt{stop\_gradient}\left(r_k + \gamma V_\phi^{\pi_\theta}(s_{k+1})\right)\right)^2$$

where `stop_gradient` enforces that no gradients flow through its argument.

4a: Understanding Actor-Critic (15 pts total)

This question is split into three conceptual questions.

Part I: Bias in Actor-Critic (5 pts)

It is said that actor-critic policy gradients are more biased than REINFORCE policy gradients. Explain what this means. Are actor-critic policy gradients more biased than REINFORCE policy gradients computed with the value function baseline?

Actor-critic policy gradients are considered more biased than REINFORCE policy gradients without a value function baseline. This increased bias stems from the introduction of a learned value function in actor-critic methods. The critic (value function) introduces additional approximation errors, as it estimates the state or action values, and any inaccuracies in this estimation propagate to the policy gradient computation. However, when compared to REINFORCE policy gradients computed with a value function baseline, actor-critic policy gradients might not necessarily be more biased. The use of a baseline in REINFORCE helps reduce the variance of the gradient estimates introduce a bias. The bias introduced in actor-critic methods defers in nature, as it arises from the approximation of value functions, while the bias in REINFORCE with a baseline is influenced by the choice and accuracy of the baseline.

Part II: Per-Step Updates (5 pts)

Even though actor-critic algorithms can perform one update per step, the gradients are computed based on data from only one state transition as opposed to REINFORCE gradients which are averaged over T state transitions. What is the benefit of updating once per environment step?

It brings the benefit of increased sample efficiency. With per-step updates, each transition provides an immediate learning signal, allowing the algorithm to make swift adjustments based on individual experiences, enabling the model to learn more quickly and adapt to changes in the environment with a shorter delay. Additionally, per-step updates can be computationally less demanding.

Part III: Lifelong Learning (5 pts)

Imagine a scenario where an RL agent is to be deployed on a strange planet that we do not know how to simulate. Once we drop the robot on this planet, we can never interact with it again: it just autonomously learns from environment interactions for the rest of its life. Would you prefer to employ Actor-Critic or REINFORCE with baseline for this problem? Why?

Opting for REINFORCE with a baseline over Actor-Critic is preferable due to its greater stability. REINFORCE with a baseline tends to be more robust, especially when compared to Actor-Critic, which is prone to divergence, particularly when the value function is poorly estimated. Given the constraints of no post-deployment interaction to correct potential errors in the value function, the stability of REINFORCE with a baseline becomes a critical factor.

4b: Implementing Actor-Critic (9 pts)

Fill out the `ActorCriticPolicy` class below, according to the guidelines in the code. The `policy_init_network` and `value_init_network` methods will be used to instantiate the neural nets for the actor-critic, however they are trained differently in the actor-critic algorithm.

Rather than implementing an `update` method for actor-critic, we will implement a method `train_episode` which rolls out an episode, performing updates at each step. More precisely, `train_episode` should do the following:

1. Reset the environment to a starting state
2. For each environment step:
   A. Choose an action
   B. Perform an environment step with the chosen action, observing the next state, reward, and terminal signal
   C. Update both the actor and critic networks based on this transition
3. Return a dictionary with the same entries as `REINFORCEWithBaselinePolicy`'s `update` method.

In [25]:
```
class ActorCriticPolicy(Policy):
    def __init__(
        self,
        env: gym.Env,
        policy_network: nn.Module,
        value_network: nn.Module,
        discount=0.99,
        name="Actor-Critic",
    ):
        super().__init__(env, policy_network, discount=discount, name=name)
        # Your code here
        # Initialize self.value_network and self.value_opt like before
        # ============================================================
        self.value_network = value_network
        self.value_opt = torch.optim.Adam(self.value_network.parameters(), lr=2e-3)
        # ============================================================
```

```
    """
    Run a training episode
    Inputs:
        seed: Seed of the environment (default: 0)
    Returns:
        Dictionary with the following keys:
        - "policy_loss": float of the policy gradient loss (the quantity whose gradient
                         averaged over the episode
        - "value_loss": float of the squared TD error averaged over the episode
    """

    def train_episode(self, seed=0) -> float:
        loss_dict = {}
        state, _ = self.env.reset(seed=seed)
        # Your code here
        # ======================
        done = False
        policy_losses = []
        value_losses = []
        while not done:
            action = self.action(state)
            next_state, reward, terminated, truncated, _ = self.env.step(action)
            done = terminated or truncated

            # Policy Update
            next_state_value = (
                self.value_network(torch.Tensor(next_state)).squeeze().detach()
            )
            current_state_value = (
                self.value_network(torch.Tensor(state)).squeeze().detach()
            )
            pi_a_s = self.distribution(state).log_prob(torch.Tensor([action])).squeeze()
            policy_loss = (
                -pi_a_s
                * (reward + self.discount * next_state_value - current_state_value)
            )

            self.opt.zero_grad()
            policy_loss.backward()
            self.opt.step()

            # Value Update
            current_state_value = self.value_network(torch.Tensor(state)).squeeze()
            G = torch.tensor(reward + self.discount * next_state_value).detach()
            value_loss = 0.5 * torch.pow(current_state_value - G, 2)

            self.value_opt.zero_grad()
            value_loss.backward()
            self.value_opt.step()

            policy_losses.append(policy_loss.item())
            value_losses.append(value_loss.item())
            state = next_state
        loss_dict["policy_loss"] = np.mean(policy_losses)
        loss_dict["value_loss"] = np.mean(value_losses)
        # ======================
        return loss_dict
```

In [26]:
```
grader.check("question 4b")
```

Out[26]:

question 4b passed! 🙌

4c: Experiments (15 pts)

In the following experiments, we test the following agents:

- REINFORCE with one trajectory per gradient update
- REINFORCE with the value function baseline, one trajectory per gradient update
- Actor-Critic

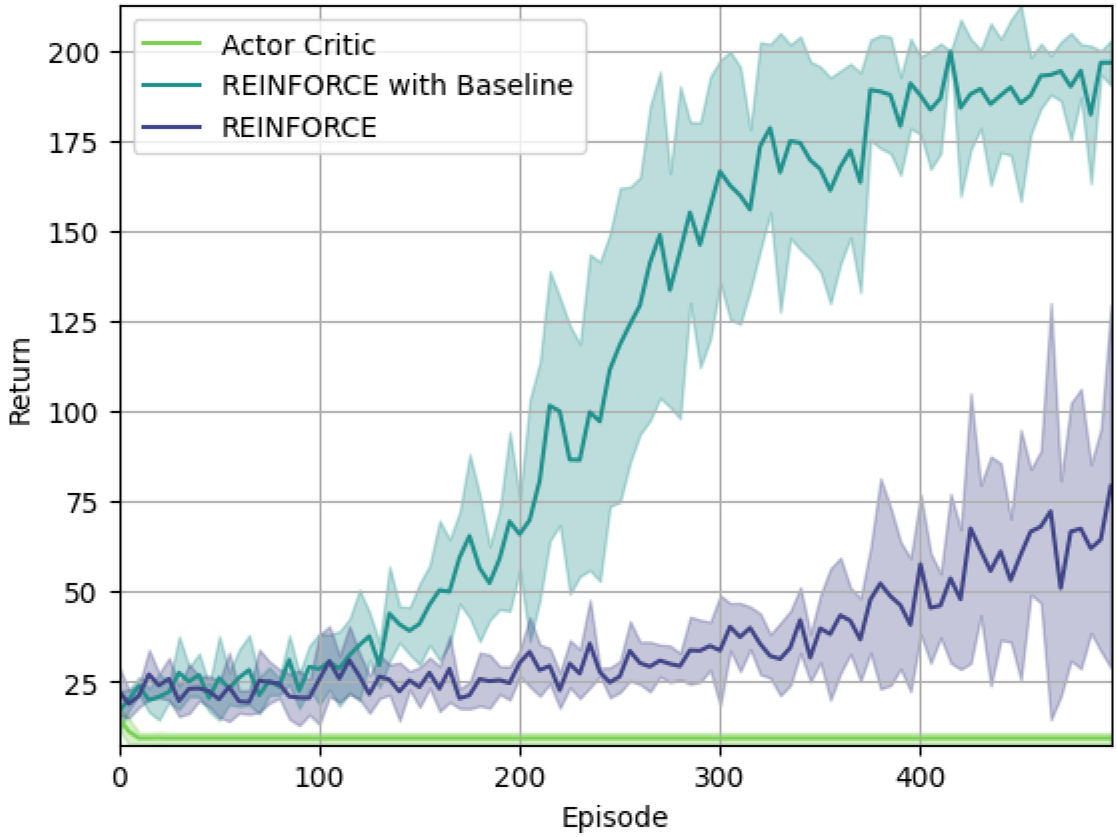
Each agent is trained for 400 episodes, and the experiment is repeated 6 times with different random seeds. The plot displays the mean and variance of the return across the seeds for each agent.

In [30]:
```
env = gym.make('CartPole-v1', render_mode="rgb_array", max_episode_steps=200)
import itertools
SEEDS = [4, 8, 16, 23, 42]

episodes = 500
eval_runs = 5
eval_every = 5
epochs = list(range(0, episodes, eval_every))

cmap = plt.get_cmap('viridis')
plt.grid(True)
plt.margins(0)
plt.xlabel("Episode")
plt.ylabel("Return")

pg_constructor = lambda: REINFORCEPolicy(env, policy_init_network(env))
pg_baseline_constructor = lambda: REINFORCEWithBaselinePolicy(env, policy_init_network(e

ac_agent_constructor = lambda: ActorCriticPolicy(env, policy_init_network(env), value_in

if not running_in_gradescope():
    ### ACTOR CRITIC
    ac_agents = {}
    ac_score_traces = {}
    print(f"Training Actor-Critic")
    for (seed, ep) in tqdm(itertools.product(SEEDS, np.arange(episodes))):
        if seed not in ac_agents.keys():
            np.random.seed(seed)
            torch.manual_seed(seed)
            ac_agents[seed] = ac_agent_constructor()
            ac_score_traces[seed] = []
            ac_agents[seed].env.action_space.seed(seed)
        agent = ac_agents[seed]
        loss = agent.train_episode()
        if (ep + 1) % eval_every == 0:
            res = [rollout_score(env, agent) for _ in range(eval_runs)]
            ac_score_traces[seed].append(np.mean(res))

    ac_data = np.vstack([ac_score_traces[seed] for seed in SEEDS])
    ac_score_mean = np.mean(ac_data, axis=0)
    ac_score_std = np.std(ac_data, axis=0)

    plt.plot(epochs, ac_score_mean, color=cmap(0.8), label='Actor Critic')
    plt.fill_between(
        epochs,
        ac_score_mean - ac_score_std,
        ac_score_mean + ac_score_std,
        color=cmap(0.8),
        alpha=0.3
    )

    ### REINFORCE WITH BASELINE
    pg_baseline_agents = {}
    pg_baseline_score_traces = {}
```

```python
    print(f"Training REINFORCE with Baseline")
    for (seed, ep) in tqdm(itertools.product(SEEDS, np.arange(episodes))):
        if seed not in pg_baseline_agents.keys():
            np.random.seed(seed)
            torch.manual_seed(seed)
            pg_baseline_agents[seed] = pg_baseline_constructor()
            pg_baseline_score_traces[seed] = []
            env.env.action_space.seed(seed)
        agent = pg_baseline_agents[seed]
        states, actions, rewards, terminated, truncated = generate_episode(env, agent)
        dones = [term or trunc for (term, trunc) in zip(terminated, truncated)]
        loss = agent.update(states, actions, rewards, dones)
        if (ep + 1) % eval_every == 0:
            res = [rollout_score(env, agent) for _ in range(eval_runs)]
            pg_baseline_score_traces[seed].append(np.mean(res))

    pg_baseline_data = np.vstack([pg_baseline_score_traces[seed] for seed in SEEDS])
    pg_baseline_score_mean = np.mean(pg_baseline_data, axis=0)
    pg_baseline_score_std = np.std(pg_baseline_data, axis=0)

    plt.plot(epochs, pg_baseline_score_mean, color=cmap(0.5), label='REINFORCE with Base
    plt.fill_between(
        epochs,
        pg_baseline_score_mean - pg_baseline_score_std,
        pg_baseline_score_mean + pg_baseline_score_std,
        color=cmap(0.5),
        alpha=0.3
    )

    ### REINFORCE
    pg_agents = {}
    pg_score_traces = {}
    print(f"Training REINFORCE with Baseline")
    for (seed, ep) in tqdm(itertools.product(SEEDS, np.arange(episodes))):
        if seed not in pg_agents.keys():
            np.random.seed(seed)
            torch.manual_seed(seed)
            pg_agents[seed] = pg_constructor()
            pg_score_traces[seed] = []
            env.env.action_space.seed(seed)
        agent = pg_agents[seed]
        states, actions, rewards, terminated, truncated = generate_episode(env, agent)
        dones = [term or trunc for (term, trunc) in zip(terminated, truncated)]
        loss = agent.update(states, actions, rewards, dones)
        if (ep + 1) % eval_every == 0:
            res = [rollout_score(env, agent) for _ in range(eval_runs)]
            pg_score_traces[seed].append(np.mean(res))

    pg_data = np.vstack([pg_score_traces[seed] for seed in SEEDS])
    pg_score_mean = np.mean(pg_data, axis=0)
    pg_score_std = np.std(pg_data, axis=0)

    plt.plot(epochs, pg_score_mean, color=cmap(0.2), label='REINFORCE')
    plt.fill_between(
        epochs,
        pg_score_mean - pg_score_std,
        pg_score_mean + pg_score_std,
        color=cmap(0.2),
        alpha=0.3
    )

    plt.legend()
```

```
Training Actor-Critic
2500it [01:18, 31.83it/s]
```

```
Training REINFORCE with Baseline
2500it [03:12, 13.00it/s]
Training REINFORCE with Baseline
2500it [01:06, 37.34it/s]
```



Visualizing AC policy

In [31]:
```
env = gym.make('CartPole-v1', render_mode="rgb_array", max_episode_steps=200)
render_video(env, policy = ac_agents[4], steps = 200)
```

```
error: XDG_RUNTIME_DIR not set in the environment.
Return:  8.0
<IPython.core.display.Image object>
```

In [32]:
```
plt.close('all')
```

Analysis (5 pts)

Based on your experiments, does actor-critic perform favorably to REINFORCE (with and/or without baseline)? Explain your observations based on the learning curves and visualization.

The performance of actor-critic did not exhibit a favorable trend compared to REINFORCE, both with and without a baseline. The learning curves revealed a collapse in the actor-critic's performance, suggesting challenges in stability during training. In contrast, REINFORCE, particularly when equipped with a baseline, demonstrated more stable and consistent learning throughout the training process. The learning curves for REINFORCE exhibited smoother convergence, indicating a more reliable adaptation to the environment. The use of a baseline in REINFORCE contributed to reduced variance in the policy gradient estimates, enhancing training stability.