MineRL

Release 0.3.0

William H. Guss, Brandon Houghton

Jun 22, 2022

TUTORIALS AND GUIDES

Welcome to documentation for the MineRL project and its related repositories and components!

CHAPTER

ONE

WHAT IS MINERL

MineRL is a research project started at Carnegie Mellon University aimed at developing various aspects of artificial intelligence within Minecraft. In short MineRL consists of several major components:

- MineRL-v0 Dataset – One of the largest imitation learning datasets with over **60 million frames** of recorded human player data. The dataset includes a set of environments which highlight many of the hardest problems in modern-day Reinforcement Learning: sparse rewards and hierarchical policies.
- `minerl` – A rich python3 package for doing artificial intelligence research in Minecraft. This includes two major submodules. We develop `minerl` in our spare time, please consider supporting us on Patreon
  - `minerl.env` – A growing set of OpenAI Gym environments in Minecraft. These environments leverage a **synchronous**, **stable**, and **fast** fork of Microsoft Malmo called *MineRLEnv*.
  - `minerl.data` – The main python module for ext with the *MineRL-v0* dataset

1.1 Installation

Welcome to MineRL! This guide will get you started.

To start using the data and set of reinforcement learning environments comprising MineRL, you'll need to install the main python package, `minerl`.

1. First **make sure you have JDK 1.8** installed on your system.
   a. Windows installer – On windows go this link and follow the instructions to install JDK 8.
   b. On Mac, you can install java8 using homebrew and AdoptOpenJDK (an open source mirror, used here to get around the fact that Java8 binaries are no longer available directly from Oracle):

```
brew tap AdoptOpenJDK/openjdk
brew cask install adoptopenjdk8
```

   c. On Debian based systems (Ubuntu!) you can run the following:

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

2. Now install the `minerl` package!:

```
pip3 install --upgrade minerl
```

Note: depending on your system you may need the user flag: `pip3 install --upgrade minerl --user` to install property

3. (Optional) Download the dataset ~ 60 GB:

```
import minerl
minerl.data.download(directory="/your/local/path/")
```

Or simply download a single experiment

```
minerl.data.download('/your/local/path',experiment='MineRLObtainDiamondVectorObf-v0
↪')
```

For a complete list of published experiments, checkout the environment documentation. If you are here for the MineRL competition, checkout the competition environments.

That's it! Now you're good to go :)

Caution: Currently `minerl` only supports environment rendering in **headed environments** (machines with monitors attached).

In order to run `minerl` environments without a head use a software renderer such as `xvfb`:

```
xvfb-run python3 <your_script.py>
```

1.2 Hello World: Your First Agent

With the `minerl` package installed on your system you can now make your first agent in Minecraft!

To get started, let's first import the necessary packages

```
import gym
import minerl
```

1.2.1 Creating an environment

Now we can choose any one of the many environments included in the `minerl` package. To learn more about the environments checkout the environment documentation.

For this tutorial we'll choose the `MineRLNavigateDense-v0` environment. In this task, the agent is challenged with using a first-person perspective of a random Minecraft map and navigating to a target.

To create the environment, simply invoke `gym.make`

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

Caution: Currently `minerl` only supports environment rendering in **headed environments** (servers with monitors attached).

In order to run `minerl` environments without a head use a software renderer such as `xvfb`:

```
xvfb-run python3 <your_script.py>
```

Note: The first time you run this command to complete, it will take a while as it is recompiling Minecraft with the MineRL simulator mod!

If you're worried and want to make sure something is happening behind the scenes install a logger **before** you create the envrionment.

```
import logging
logging.basicConfig(level=logging.DEBUG)

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

1.2.2 Taking actions

As a warm up let's create a random agent.

Now we can reset this environment to its first position and get our first observation from the agent by resetting the environment.

```
obs = env.reset()
```

The `obs` variable will be a dictionary containing the following observations returned by the environment. In the case of the `MineRLNavigate-v0` environment, three observations are returned: `pov`, an RGB image of the agent's first person perspective; `compassAngle`, a float giving the angle of the agent to its (approximate) target; and `inventory`, a dictionary containing the amount of `'dirt'` blocks in the agent's inventory (this is useful for climbing steep inclines).

```
{
    'pov': array([[[ 63,  63,  68],
        [ 63,  63,  68],
        [ 63,  63,  68],
        ...,
        [ 92,  92, 100],
        [ 92,  92, 100],
        [ 92,  92, 100]],,


        ...,


        [[ 95, 118, 176],
        [ 95, 119, 177],
        [ 96, 119, 178],
        ...,
        [ 93, 116, 172],
        [ 93, 115, 171],
        [ 92, 115, 170]]], dtype=uint8),
    'compassAngle': -63.48639,
    'inventory': {'dirt': 0}
}
```

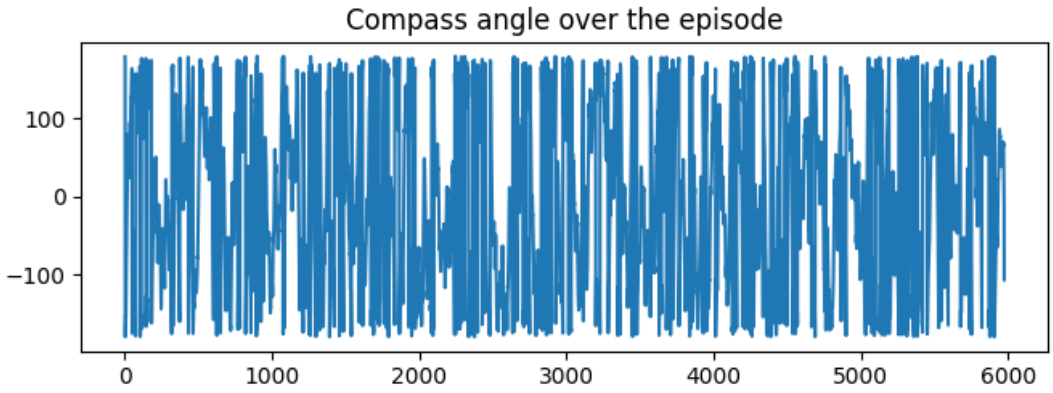
Note: To see the exact format of observations returned from and the exact action format expected by `env.step` for any environment refer to the environment reference documentation!

Now let's take actions through the environment until time runs out or the agent dies. To do this, we will use the normal OpenAI Gym `env.step` method.

```
done = False

while not done:
    action = env.action_space.sample()
    obs, reward, done, _ = env.step(action)
```

After running this code you should see your agent move sporadically until the `done` flag is set to true. To confirm that our agent is at least qualitatively acting randomly, on the right is a plot of the compass angle over the course of the experiment.



1.2.3 No-op actions and a better policy

Now let's make a hard-coded agent that actually runs towards the target.

To do this at every step of the environment we will take the *noop* action with a few modifications; in particular, we will only move forward, jump, attack, and change the agent's direction to minimize the angle between the agent's movement direction and it's target, `compassAngle`.

```
import minerl
import gym
env = gym.make('MineRLNavigateDense-v0')


obs  = env.reset()
done = False
net_reward = 0

while not done:
    action = env.action_space.noop()

    action['camera'] = [0, 0.03*obs["compassAngle"]]
    action['back'] = 0
```

(continues on next page)

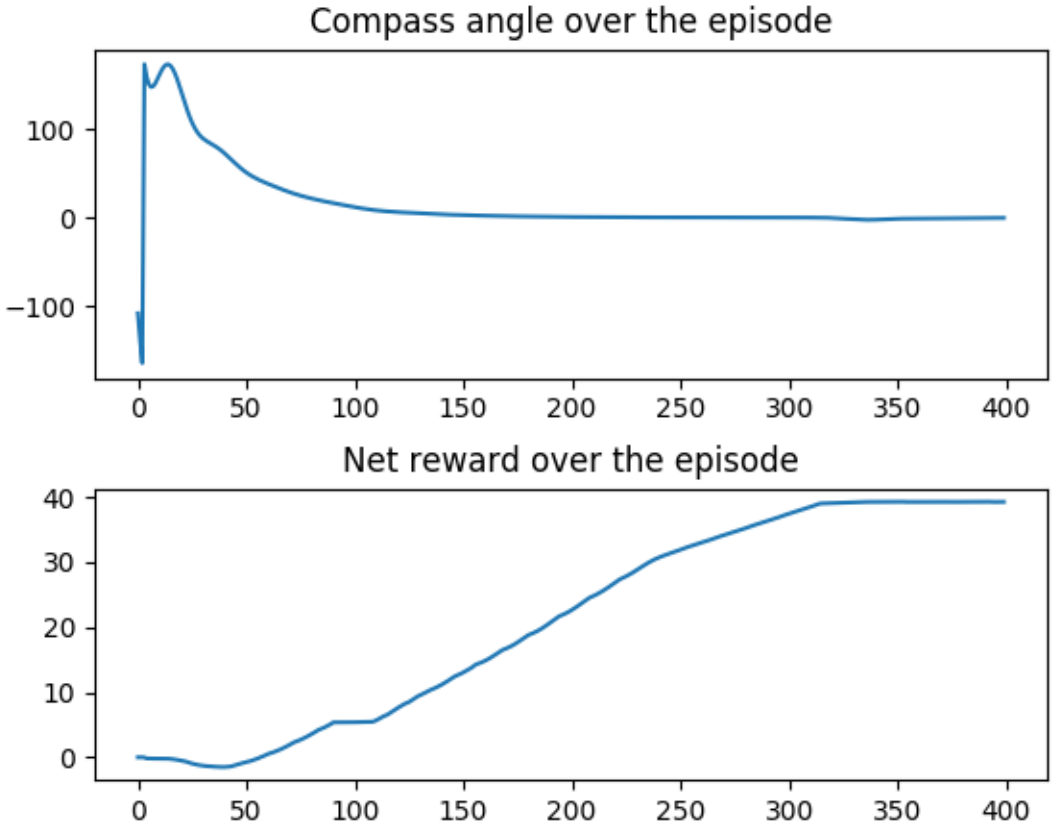
(continued from previous page)

```
    action['forward'] = 1
    action['jump'] = 1
    action['attack'] = 1

    obs, reward, done, info = env.step(
        action)

    net_reward += reward
    print("Total reward: ", net_reward)
```

After running this agent, you should notice marekedly less sporadic behaviour. Plotting both the `compassAngle` and the net reward over the episode confirm that this policy performs better than our random policy.



Congratulations! You've just made your first agent using the `minerl` framework!

1.3 Sampling The Dataset

Now that your agent can act in the environment, we should show it how to leverage human demonstrations.

To get started, let's ensure the data has been downloaded.

```
# Unix, Linux
$MINERL_DATA_ROOT="your/local/path" python3 -m minerl.data.download
```

(continues on next page)

(continued from previous page)

```
# Windows
$env:MINERL_DATA_ROOT="your/local/path"; python3 -m minerl.data.download
```

Or we can simply download a single experiment

```
# Unix, Linux
$MINERL_DATA_ROOT="your/local/path" python3 -m minerl.data.download "MineRLObtainDiamond-
↪v0"

# Windows
$env:MINERL_DATA_ROOT="your/local/path"; python3 -m minerl.data.download
↪"MineRLObtainDiamond-v0"
```

For a complete list of published experiments, checkout the environment documentation. You can also download the data in your python scripts

Now we can build the datast for `MineRLObtainDiamond-v0`

```
data = minerl.data.make(
    'MineRLObtainDiamond-v0')


for current_state, action, reward, next_state, done \
    in data.batch_iter(
        batch_size=1, num_epochs=1, seq_len=32):

        # Print the POV @ the first step of the sequence
        print(current_state['pov'][0])

        # Print the final reward pf the sequence!
        print(reward[-1])

        # Check if final (next_state) is terminal.
        print(done[-1])

        # ... do something with the data.
        print("At the end of trajectories the length"
              "can be < max_sequence_len", len(reward))
```

Warning: The `minerl` package uses environment variables to locate the data directory. For portability, plese define `MINERL_DATA_ROOT` as `/your/local/path/` in your system environment variables.

1.3.1 Moderate Human Demonstrations

MineRL-v0 uses community driven demonstrations to help researchers develop sample efficient techniques. Some of these demonstrations are less than optimal, however others could feacture bugs with the client, server errors, or adversarial behavior.

Using the MineRL viewer, you can help curate this dataset by viewing these demonstrations manually and reporting bad streams by submitting an issue to github with the following information:

1. The stream name of the stream in question
2. The reason the stream or segment needs to be modified
3. The sample / frame number(s) (shown at the bottom of the viewer)

1.4 K-means exploration

With the 2020 MineRL competition, we introduced vectorized obfuscated environments which abstract non-visual state information as well as the action space of the agent to be continuous vector spaces. See MineRLObtainDiamondVectorObf-v0 for documentation on the evaluation environment for that competition.

To use techniques in the MineRL competition that require discrete actions, we can use k-means to quantize the human demonstrations and give our agent n discrete actions representative of actions taken by humans when solving the environment.

To get started, let's download the MineRLTreechopVectorObf-v0 environment.

```
python -m minerl.data.download 'MineRLTreechopVectorObf-v0'
```

Note: If you are unable to download the data ensure you have the `MINERL_DATA_ROOT` env variable set as demonstrated in data sampling.

Now we load the dataset for MineRLTreechopVectorObf-v0 and find 32 clusters using sklearn learn

```
from sklearn.cluster import KMeans

dat = minerl.data.make('MineRLTreechopVectorObf-v0')

# Load the dataset storing 1000 batches of actions
act_vectors = []
for _, act, _, _,_ in tqdm.tqdm(dat.batch_iter(16, 32, 2, preload_buffer_size=20)):
    act_vectors.append(act['vector'])
    if len(act_vectors) > 1000:
        break

# Reshape these the action batches
acts = np.concatenate(act_vectors).reshape(-1, 64)
kmeans_acts = acts[:100000]

# Use sklearn to cluster the demonstrated actions
kmeans = KMeans(n_clusters=32, random_state=0).fit(kmeans_acts)
```

Now we have 32 actions that represent reasonable actions for our agent to take. Let's take these and improve our random hello world agent from before.

```
i, net_reward, done, env = 0, 0, False, gym.make('MineRLTreechopVectorObf-v0')
obs = env.reset()

while not done:
    # Let's use a frame skip of 4 (could you do better than a hard-coded frame skip?)
    if i % 4 == 0:
        action = {
            'vector': kmeans.cluster_centers_[np.random.choice(NUM_CLUSTERS)]
        }

    obs, reward, done, info = env.step(action)
    env.render()

    if reward > 0:
        print("+{} reward!".format(reward))
    net_reward += reward
    i += 1

print("Total reward: ", net_reward)
```

Putting this all together we get:

Full snippet

```
import gym
import tqdm
import minerl
import numpy as np

from sklearn.cluster import KMeans

dat = minerl.data.make('MineRLTreechopVectorObf-v0')

act_vectors = []
NUM_CLUSTERS = 30

# Load the dataset storing 1000 batches of actions
for _, act, _, _, _ in tqdm.tqdm(dat.batch_iter(16, 32, 2, preload_buffer_size=20)):
    act_vectors.append(act['vector'])
    if len(act_vectors) > 1000:
        break

# Reshape these the action batches
acts = np.concatenate(act_vectors).reshape(-1, 64)
kmeans_acts = acts[:100000]

# Use sklearn to cluster the demonstrated actions
kmeans = KMeans(n_clusters=NUM_CLUSTERS, random_state=0).fit(kmeans_acts)

i, net_reward, done, env = 0, 0, False, gym.make('MineRLTreechopVectorObf-v0')
obs = env.reset()
```

(continues on next page)

(continued from previous page)

```
while not done:
    # Let's use a frame skip of 4 (could you do better than a hard-coded frame skip?)
    if i % 4 == 0:
        action = {
            'vector': kmeans.cluster_centers_[np.random.choice(NUM_CLUSTERS)]
        }

    obs, reward, done, info = env.step(action)
    env.render()

    if reward > 0:
        print("+{} reward!".format(reward))
    net_reward += reward
    i += 1

print("Total reward: ", net_reward)
```

Try comparing this k-means random agent with a random agent using `env.action_space.sample()`! You should see the human actions are a much more reasonable way to explore the environment!

1.5 Visualizing The Data `minerl.viewer`

To help you get familiar with the MineRL dataset, the `minerl` python package also provides a data trajectory viewer called `minerl.viewer`:

The `minerl.viewer` program lets you step through individual trajectories, showing the observation seen the player, the action they took (including camera, movement, and any action described by an MineRL environment's action space), and the reward they received.

> **Error:** Unable to execute python code at minerl_tools.rst:25:
>
> No module named 'minerl'

Try it out on a random trajectory by running:

```
# Make sure your MINERL_DATA_ROOT is set!
export MINERL_DATA_ROOT='/your/local/path'

# Visualizes a random trajectory of MineRLObtainDiamondDense-v0
python3 -m minerl.viewer MineRLObtainDiamondDense-v0
```

Try it out on a specific trajectory by running:

> **Error:** Unable to execute python code at minerl_tools.rst:51:
>
> No module named 'minerl'

1.6 Interactive Mode `minerl.interactor`

Once you have started training agents, the next step is getting them to interact with human players. To help achieve this, the `minerl` python package provides a interactive Minecraft client called `minerl.interactor`:

The `minerl.interactor` allows you to connect a human-controlled Minecraft client to the Minecraft world that your agent(s) is using and interact with the agent in real time.

Note: For observation-only mode hit the `t` key and type `/gamemode sp` to enter spectator mode and become invisible to your agen``t(s).

Error: Unable to execute python code at minerl_tools.rst:92:

No module named 'minerl'

1.7 General Information

The `minerl` package includes several environments as follows. This page describes each of the included environments, provides usage samples, and describes the exact action and observation space provided by each environment!

Caution: In the MineRL Competition, many environments are provided for training, however competition agents will only be evaluated in `MineRLObtainDiamondVectorObf-v0` which has **sparse** rewards. See **`MineRLObtainDiamondVectorObf-v0`_**.

Note: All environments offer a default no-op action via `env.action_space.no_op()` and a random action via `env.action_space.sample()`.

1.8 Environment Handlers

Minecraft is an extremely complex environment which provides players with visual, auditory, and informational observation of many complex data types. Furthermore, players interact with Minecraft using more than just embodied actions: players can craft, build, destroy, smelt, enchant, manage their inventory, and even communicate with other players via a text chat.

To provide a unified interface with which agents can obtain and perform similar observations and actions as players, we have provided first-class for support for this multi-modality in the environment: **the observation and action spaces of environments are `gym.spaces.Dict` spaces.** These observation and action dictionaries are comprised of individual fields we call *handlers*.

Note: In the documentation of every environment we provide a listing of the exact `gym.space` of the observations returned by and actions expected by the environment's step function. We are slowly building documentation for these handlers, and **you can click those highlighted with blue** for more information!

1.8.1 Environment Handlers

Minecraft is an extremely complex environment which provides players with visual, auditory, and informational observation of many complex data types. Furthermore, players interact with Minecraft using more than just embodied actions: players can craft, build, destroy, smelt, enchant, manage their inventory, and even communicate with other players via a text chat.

To provide a unified interface with which agents can obtain and perform similar observations and actions as players, we have provided first-class for support for this multi-modality in the environment: **the observation and action spaces of environments are `gym.spaces.Dict` spaces.** These observation and action dictionaries are comprised of individual fields we call *handlers*.

Note: In the documentation of every environment we provide a listing of the exact `gym.space` of the observations returned by and actions expected by the environment's step function. We are slowly building documentation for these handlers, and **you can click those highlighted with blue** for more information!

1.8.2 Observations

Visual Observations - `pov`, `third-persion`

`pov : Box(width, height, nchannels)`

An RGB image observation of the agent's first-person perspective.

Type
`np.uint8`

`third-person : Box(width, height, nchannels)`

An RGB image observation of the agent's third-person perspective.

> **Warning:** This observation is not yet supported by any environment.

Type
`np.uint8`

`compass-observation : Box(1)`

The current position of the *minecraft:compass* object from 0 (behind agent left) to 0.5 in front of agent to 1 (behind agent right)

Note: This observation uses the default Minecraft game logic which includes compass needle momentum. As such it may change even when the agent has stoped moving!

1.8.3 Actions

Camera Control - `camera`

`camera : Box(2) [delta_pitch, delta_yaw]`

This action changes the orientation of the agent's head by the corresponding number of degrees. When the `pov` observation is available, the camera changes its orientation pitch by the first component and its yaw by the second component. Both `delta_pitch` and `delta_yaw` are limited to [-180, 180] inclusive

Type
`np.float32`

Shape
[2]

`attack : Discrete(1) [attack]`

This action causes the agent to attack.

Type
`np.float32`

Shape
[1]

Tool Control - `camera`

`equip : Enum('none', 'air', 'wooden_axe', 'wooden_pickaxe', 'stone_axe', 'stone_pickaxe', 'iron_axe', 'iron_pickaxe'))`

This action equips the first instance of the specified item from the agents inventory to the main hand if the specified item is present, otherwise does nothing. `none` is never a valid item and functions as a no-op action. `air` matches any empty slot in an agent's inventory and functions as an un-equip action.

Type
`np.int64`

Shape
[1]

Note: Agents may un-equip items by performing the `equip air` action.

1.9 Basic Environments

Warning: The following basic environments are NOT part of the 2020 MineRL Competition! Feel free to use them for exploration but agents may only be trained on MineRL competition environments!

Error: Unable to execute python code at index.rst:67:

No module named 'minerl'

1.10 Competition Environments

Error: Unable to execute python code at index.rst:170:

No module named 'minerl'

1.11 Windows FAQ

This note serves as a collection of fixes for errors which may occur on the Windows platform.

1.11.1 The `freeze_support` error (multiprocessing)

```
RuntimeError:
       An attempt has been made to start a new process before the
       current process has finished its bootstrapping phase.

   This probably means that you are not using fork to start your
   child processes and you have forgotten to use the proper idiom
   in the main module:

       if __name__ == '__main__':
           freeze_support()
           ...

   The "freeze_support()" line can be omitted if the program
   is not going to be frozen to produce an executable.
```

The implementation of `multiprocessing` is different on Windows, which uses `spawn` instead of `fork`. So we have to wrap the code with an if-clause to protect the code from executing multiple times. Refactor your code into the following structure.

```
import minerl
import gym

def main()
    # do your main minerl code
    env = gym.make('MineRLTreechop-v0')

if __name__ == '__main__':
    main()
```

1.12 minerl.env

The `minerl.env` package provides an optimized python software layer over *MineRLEnv*, a fork of the popular Minecraft simulator Malmo which enables **synchronous**, **stable**, and **fast** samples from the Minecraft environment.

1.12.1 MineRLEnv

1.12.2 InstanceManager

1.13 minerl.data

The `minerl.data` package provides a unified interface for sampling data from the *MineRL-v0 Dataset*. Data is accessed by making a dataset from one of the minerl environments and iterating over it using one of the iterators provided by the `minerl.data.DataPipeline`

The following is a description of the various methods included within the package as well as some basic usage examples. To see more detailed descriptions and tutorials on how to use the data API, please take a look at our numerous getting started manuals.

1.13.1 MineRLv0

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search