Install keras and use a network to control an agent

In this lab we are going to install the keras library. Then we are going to create a neural network and try to use it to control a game player. At this point, we are not going to try and train the network. We'll just see how it is possible to use a network to control an agent in a game.

Objectives

At the end of this lab, you will be able to:

- Install the keras library
- Create a simple neural network
- Use an untrained neural network to select actions for a game playing agent

Install keras

We will assume that you have set up gymnasium and that you have been able to create a random game playing agent, as per the instructions in a previous worksheet. If you are using a coursera lab environment, keras should already be installed.

Inside your Python virtual environment, let's install keras:

```
pip install tensorflow # on the regular command line
!pip install tensorflow # if you are in ipython
```

Then:

```
import keras
print(keras.__version__)
```

I see some messages about tensorflow and cuda, then '3.3.3' for the version. It should be ok if you have a different version.

Create a network

Now let's create a simple neural network:

```
import tensorflow as tf

inputs = tf.keras.Input(shape=(1,))
layer1 = tf.keras.layers.Dense(512, activation="relu")(inputs)
outputs = tf.keras.layers.Dense(3, activation="linear")(layer1)
nn_model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

This one has 1 input, one hidden layer of 512, densely connected units then 3 outputs. You can print a description of the network with:

```
nn_model.summary()
```

I see this:

```
Model: "model_1"
+---------------------------------+------------------------+-------------+
| Layer (type)                    | Output Shape           |    Param #  |
+---------------------------------+------------------------+-------------+
| input_layer_3 (InputLayer)      | (None, 1)              |           0 |
+---------------------------------+------------------------+-------------+
| dense_5 (Dense)                 | (None, 512)            |       1,024 |
+---------------------------------+------------------------+-------------+
| dense_6 (Dense)                 | (None, 3)              |       1,539 |
+---------------------------------+------------------------+-------------+
 Total params: 2,563 (10.01 KB)
 Trainable params: 2,563 (10.01 KB)
 Non-trainable params: 0 (0.00 B)
```

Feed some data into the network

Now we have a network, we can try to feed some data into it. The input layer receives 1 value, but this does not work:

```
output = nn_model(0.5)
```

That throws an error because we need to convert the input into a tensor.

```
input = tf.constant([[3.0]])
output = nn_model(input)
print(output)
```

I see this:

```
tf.Tensor([[-0.27445188  0.14845954 -0.03802669]], shape=(1, 3), dtype=float32)
```

You might see different values because the network starts with random values for its 2,563 parameters. Can you feed the network a sequence of values between 0 and 1? Here is some handy code to help with that:

```
import numpy as np
inputs = np.arange(0, 1.1, 0.1)
```

Use the network to select actions

Now we have created a simple network and played around with it, we will try to create a network that can control an agent in a gymnasium.

As a reference point, the DQN agent contains a neural network. The neural network takes as its input the state of the gym environment and produces at its output the expected future reward of all available actions. We will see DQN in much more detail later, but for now we can at least make a network with the correct inputs and outputs.

We can start with a simple gym environment which has a simple observation/ state space:

```
import gymnasium as gym
e = gym.make('MountainCarContinuous-v0', render_mode='human')
e.reset()
e.render()
```

You should see something like this:

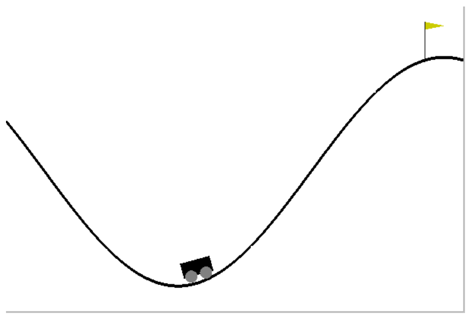

Figure 1: Mountain car gym

According to the documentation in the source code this environment here:

https://gymnasium.farama.org/environments/classic_control/mountain_car/

The observation space, AKA state is defined as follows:

```
Observation:
    Type: Box(2)
    Num    Observation               Min            Max
    0      Car Position              -1.2           0.6
    1      Car Velocity              -0.07          0.07
```

We can examine a state like this:

```
import gymnasium as gym
e = gym.make('MountainCarContinuous-v0', render_mode='human')
s1 = e.reset()
print(s1)
```

The output (a state) on my machine here looks like this:

```
(array([-0.5497219,  0.       ], dtype=float32), {})
```

Look carefully - what is the state exactly?

```
print(type(s))
```

This:

```
tuple
```

So if we are feeding that into a network, we need two inputs. But what about the action space? Again from the docs:

```
Actions:
    Type: Box(1)
    Num    Action                    Min            Max
    0      the power coef            -1.0           1.0
    Note: actual driving force is calculated by multiplying the power coef by power (0.0015)
```

and an example action:

```
e.action_space.sample()
array([-0.18952793], dtype=float32)
```

So the action is a float value in the range -1 .. 1

So we need two inputs for the state and one output for the action.

```
inputs = tf.keras.Input(shape=(2,))
layer1 = tf.keras.layers.Dense(512, activation="relu")(inputs)
outputs = tf.keras.layers.Dense(1, activation="linear")(layer1)
nn_model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

Control te agent with the network

Ok so let's try and control the agent using our un-trained neural network. We do not expect this to be a successful agent but at least we will be using a real neural network to process the state and produce actions.

```
# assuming you have nn_model and e set up as above
s = e.reset() # remember this is a tuple
input = tf.Variable([s[0]]) # tf.Variable lets us wrap our input in a tensor
a = nn_model(input)
s,r,d,t,i = e.step(a)
print(s)
```

Now - can you put the 'select action and step' part into a loop and add a call to e.render() after each step? Does the car drive up the hill? Probably not! But we are controlling it with a neural network now. We'll cover how to train the neural network using DQN in a later lab.

Discrete action spaces: Breakout

The next thing we will look at is how to select actions from discrete action spaces. In retro-style arcade games, we normally select from a set of actions such as up, down, left, right, fire. How can we configure the network to tell us which of those actions to take?

The answer is to have one output for each possible action, then to select the output with the highest activation value.

Let's create an arcade style gym with a discrete action space:

```
import gymnasium as gym
# info here: https://gymnasium.farama.org/environments/atari/breakout/
env_name = "BreakoutNoFrameskip-v4"
e = gym.make(env_name)
s = e.reset()
print(e.action_space)
```

I see: Discrete(4). Let's try to generate random sample from the action space:

```
print(e.action_space.sample())
print("Observation/ state shape:", s[0].shape)
```

I see:

```
0
Observation/ state shape: (210, 160, 3)
```

But oh no! the state is really complex now. No problem, we just need a more complex input layer. So here is a network that can take that state as an input and which gives us 4 outputs:

```
import tensorflow as tf
inputs = tf.keras.layers.Input(shape=(210, 160, 3,))
layer1 = tf.keras.layers.Flatten()(inputs)
layer2 = tf.keras.layers.Dense(512, activation="relu")(layer1)
outputs = tf.keras.layers.Dense(4, activation="linear")(layer2)
nn_model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

You can try swapping out your model for that code. Or alternatively, here is a complete block of code you can run:

```
import gymnasium as gym
import tensorflow as tf

# make the gym
env_name = "BreakoutNoFrameskip-v4"
e = gym.make(env_name, render_mode='human')
s = e.reset()

# make the nn model
inputs = tf.keras.layers.Input(shape=(210, 160, 3,))
layer1 = tf.keras.layers.Flatten()(inputs)
layer2 = tf.keras.layers.Dense(512, activation="relu")(layer1)
outputs = tf.keras.layers.Dense(4, activation="linear")(layer2)
nn_model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

```
# get the starter state by resetting
state = e.reset()

# flatten the state and push it through the model to get actions
action_values = nn_model(tf.expand_dims(state[0], axis=0))
print("Action values:", action_values.numpy())
```

That prints out:

```
Action values: [[-156.33253 -153.69717  107.56254  146.16824]]
```

In the MDP model seen the lectures, where the network approximates the Q-function, the outputs of the network are the expected future values of each of the four actions. The higher the value, the the more reward the network thinks that move will give us in the future. Remember it is a random network now, not trained, so it is most likely completely wrong. But ... how do we select an action from action_values? Simple:

```
action = tf.argmax(action_values[0]).numpy()
print("Network chooses action:", action)
```

(argmax tells us the index of the highest value in an array.)

With breakout, the first screen is blank (state is all zeroes) so we need to step it to get some decent state values flowing through:

```
e.reset()
state,r,d,t,i = e.step(1)
for i in range(100):
    state_tensor = tf.expand_dims(state, axis=0)
    action_values = nn_model(state_tensor)
    action = tf.argmax(action_values[0]).numpy()
    print(action_values, action)
    state,r,d,t,i = e.step(1)
    e.render()
```

I used the action 1 which according to the docs here:https://gymnasium.farama.org/environments/atari/breakout/ is the fire button. I noticed when testing this code that the game will not start until the player presses the fire button. The player needs to press fire after the ball goes off the screen also, otherwise the game waits without a ball.

I see some output like this:

```
tf.Tensor([[ 0.29468542  0.54243755 -0.20937413 -0.1774773 ]], shape=(1, 4), dtype=float32)
tf.Tensor([[ 0.30317318  0.5552192  -0.21397775 -0.16606629]], shape=(1, 4), dtype=float32)
tf.Tensor([[ 0.29633278  0.5732359  -0.20236504 -0.19220275]], shape=(1, 4), dtype=float32)
```

Remember those outputs are the expected rewards for each of the actions. You can see that the neural network output is changing as the state input changes. To be clear - we are not training the network yet - we are just passing different values in, so we get different values out.

In my case it always selects the same action - index 2. That is move right, so it always ends up on the right of the screen! Again - that is where the training comes in. More on that later.

Wrap up

We have now created a few different neural networks and wired them into agents for different gyms. So you have now seen how the data flow works in DQN. In later labs and videos you will see how we integrate training into the DQN workflow.

Objectives

Let's review the objectives for the lab. How did you do?

- Install the keras library
- Create a simple neural network
- Use a untrained neural network to select actions for a game playing agent

Complete script to create breakout, create a network and run a few frames:

```
import gymnasium as gym
import tensorflow as tf

# make the gym
env_name = "BreakoutNoFrameskip-v4"
e = gym.make(env_name, render_mode='human')
s = e.reset()

# make the nn model
inputs = tf.keras.layers.Input(shape=(210, 160, 3,))
layer1 = tf.keras.layers.Flatten()(inputs)
layer2 = tf.keras.layers.Dense(512, activation="relu")(layer1)
outputs = tf.keras.layers.Dense(4, activation="linear")(layer2)
nn_model = tf.keras.Model(inputs=inputs, outputs=outputs)

# run a few actions
e.reset()
state,r,d,t,i = e.step(1)
for i in range(100):
    state_tensor = tf.expand_dims(state, axis=0)
    action_values = nn_model(state_tensor)
    action = tf.argmax(action_values[0]).numpy()
    print(action_values, action)
    state,r,d,t,i = e.step(1)
    e.render()
```