
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN


# Artificial Neural Networks

## HOMEWORK 1

## Cart Pole Agent Training

**NAME:** Enrique Uriel Aguilar Flores

**ID:** 2001442

**SEMESTER:** August – December 2025

**TEACHER:** Dr. José Arturo Berrones Santos

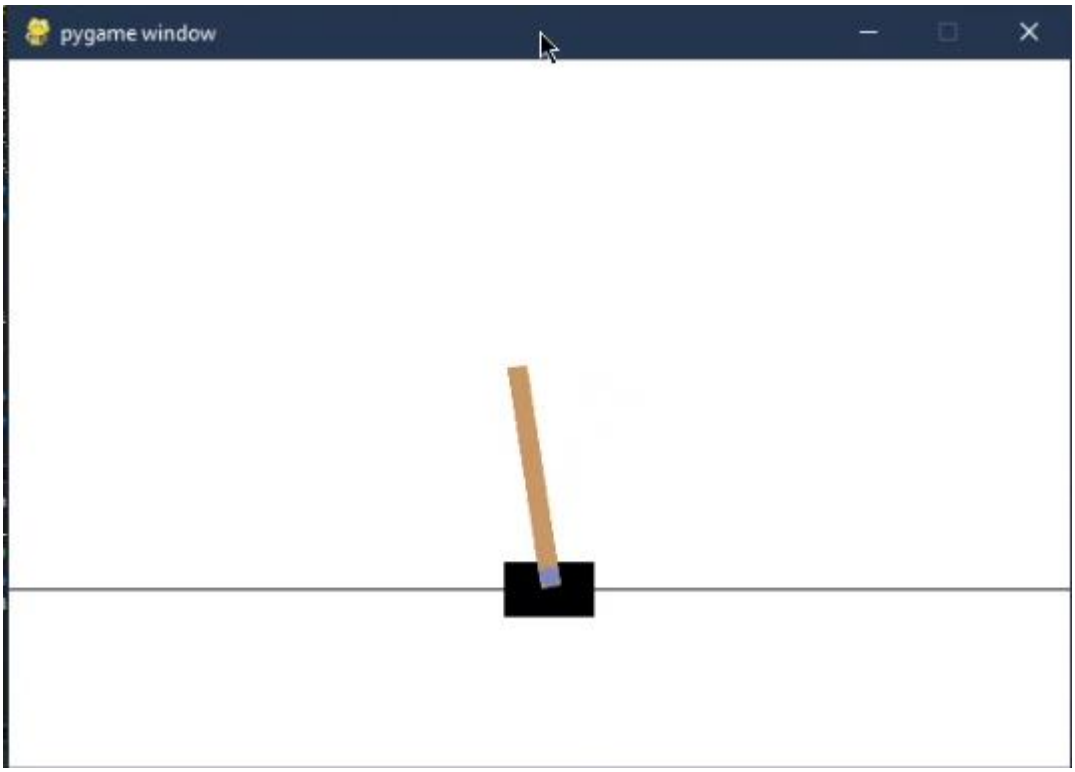
**GROUP:** 003 **HOUR:** Thursday N1-N3 **CLASSROOM:** 2101

**DATE:** Thursday, November 6th, 2025

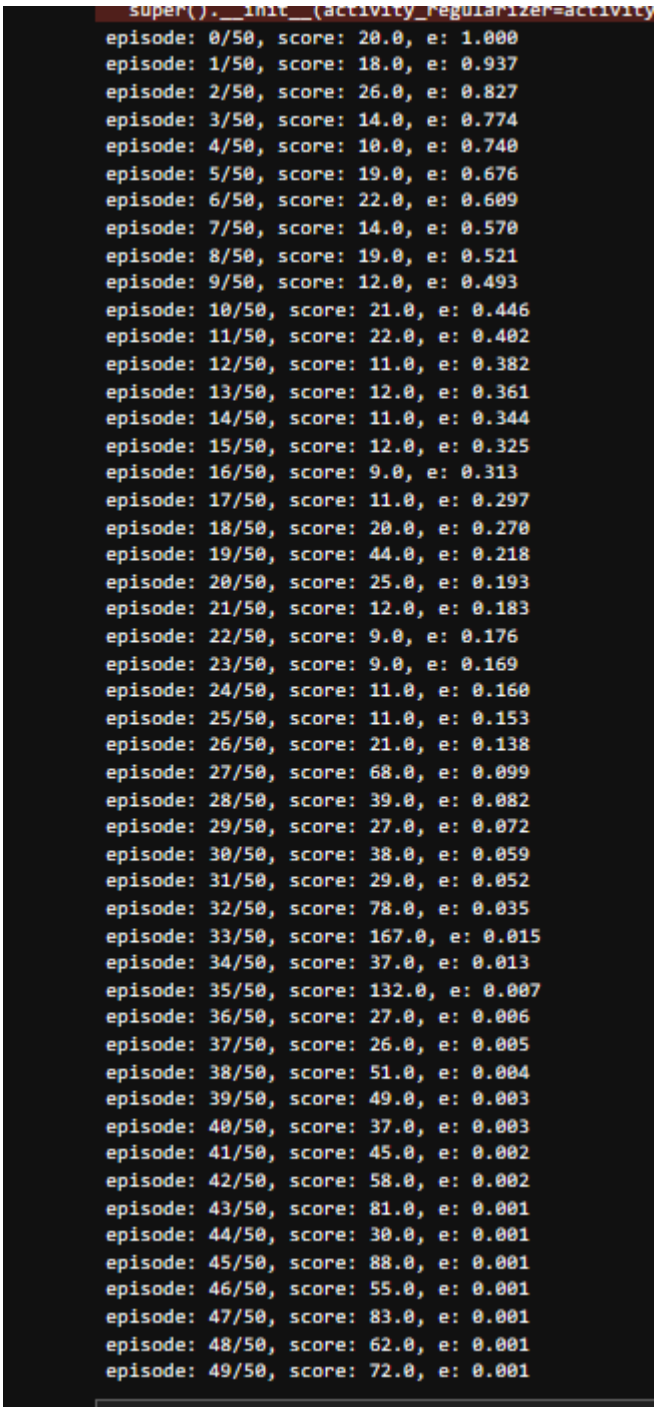
**Link to the repository:**

https://github.com/uaglr7/cartpole-agent-video

**Cart pole agent before training**

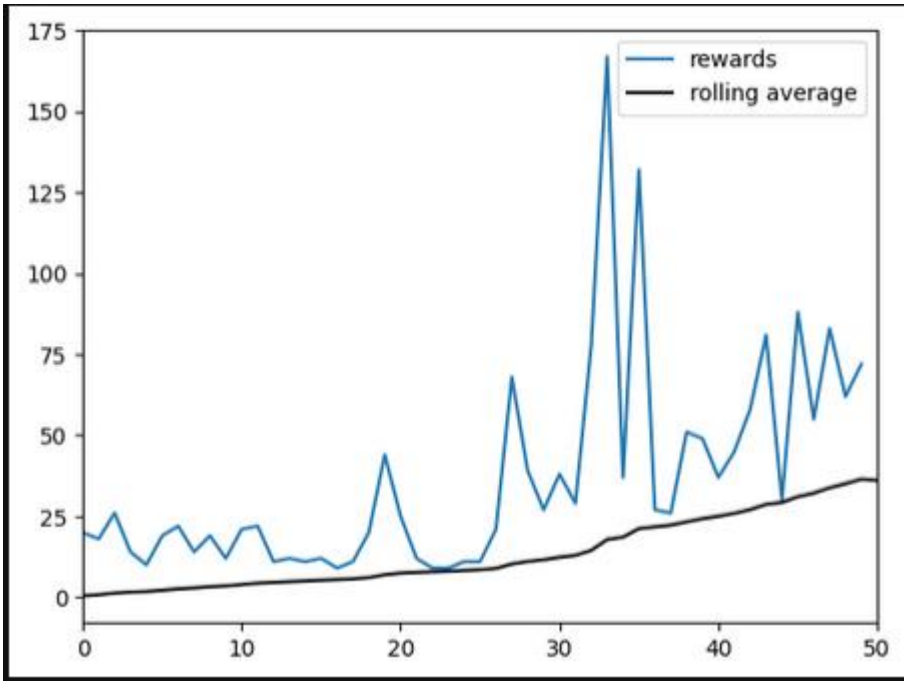


**Training logs**

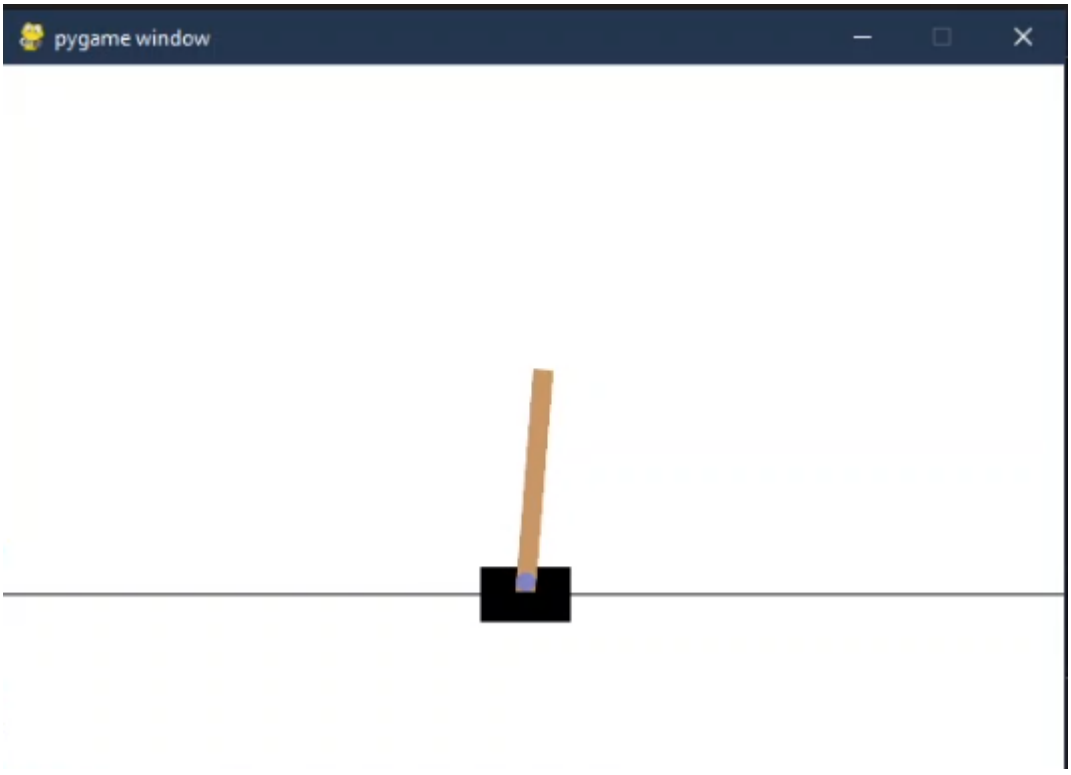


```
super().__init__(activity_regularizer=activity
episode: 0/50, score: 20.0, e: 1.000
episode: 1/50, score: 18.0, e: 0.937
episode: 2/50, score: 26.0, e: 0.827
episode: 3/50, score: 14.0, e: 0.774
episode: 4/50, score: 10.0, e: 0.740
episode: 5/50, score: 19.0, e: 0.676
episode: 6/50, score: 22.0, e: 0.609
episode: 7/50, score: 14.0, e: 0.570
episode: 8/50, score: 19.0, e: 0.521
episode: 9/50, score: 12.0, e: 0.493
episode: 10/50, score: 21.0, e: 0.446
episode: 11/50, score: 22.0, e: 0.402
episode: 12/50, score: 11.0, e: 0.382
episode: 13/50, score: 12.0, e: 0.361
episode: 14/50, score: 11.0, e: 0.344
episode: 15/50, score: 12.0, e: 0.325
episode: 16/50, score: 9.0, e: 0.313
episode: 17/50, score: 11.0, e: 0.297
episode: 18/50, score: 20.0, e: 0.270
episode: 19/50, score: 44.0, e: 0.218
episode: 20/50, score: 25.0, e: 0.193
episode: 21/50, score: 12.0, e: 0.183
episode: 22/50, score: 9.0, e: 0.176
episode: 23/50, score: 9.0, e: 0.169
episode: 24/50, score: 11.0, e: 0.160
episode: 25/50, score: 11.0, e: 0.153
episode: 26/50, score: 21.0, e: 0.138
episode: 27/50, score: 68.0, e: 0.099
episode: 28/50, score: 39.0, e: 0.082
episode: 29/50, score: 27.0, e: 0.072
episode: 30/50, score: 38.0, e: 0.059
episode: 31/50, score: 29.0, e: 0.052
episode: 32/50, score: 78.0, e: 0.035
episode: 33/50, score: 167.0, e: 0.015
episode: 34/50, score: 37.0, e: 0.013
episode: 35/50, score: 132.0, e: 0.007
episode: 36/50, score: 27.0, e: 0.006
episode: 37/50, score: 26.0, e: 0.005
episode: 38/50, score: 51.0, e: 0.004
episode: 39/50, score: 49.0, e: 0.003
episode: 40/50, score: 37.0, e: 0.003
episode: 41/50, score: 45.0, e: 0.002
episode: 42/50, score: 58.0, e: 0.002
episode: 43/50, score: 81.0, e: 0.001
episode: 44/50, score: 30.0, e: 0.001
episode: 45/50, score: 88.0, e: 0.001
episode: 46/50, score: 55.0, e: 0.001
episode: 47/50, score: 83.0, e: 0.001
episode: 48/50, score: 62.0, e: 0.001
episode: 49/50, score: 72.0, e: 0.001
```

**Plot (rewards and rolling average)**




**Cart pole agent after training**



## Code

The original code was divided into cells to simplify the recording:

### Cell 1 - Setup

```
[10]:
import numpy as np, random, imageio
import matplotlib.pyplot as plt
from collections import deque
import tensorflow as tf
from tensorflow import keras

try:
    import gymnasium as gym
except Exception:
    import gym

def reset_compat(env, seed=None):
    try:
        return env.reset(seed=seed)  # gymnasium: (obs, info)
    except TypeError:
        if seed is not None and hasattr(env, "seed"):
            env.seed(seed)
        obs = env.reset()
        info = {}
        return obs, info

def step_compat(env, action):
    out = env.step(action)
    if len(out) == 5:
        obs, reward, terminated, truncated, info = out
        done = terminated or truncated
    else:
        obs, reward, done, info = out
    return obs, reward, done, info

def make_env_human():
    try:
        return gym.make("CartPole-v1", render_mode="human")
    except TypeError:
        return gym.make("CartPole-v1")

def make_env_silent():
    try:
        return gym.make("CartPole-v1")
    except TypeError:
        return gym.make("CartPole-v1")

EPISODES = 50
TRAIN_END = 0

def discount_rate(): return 0.95
def learning_rate(): return 0.001
def batch_size():    return 24
```

## Cell 2 – DQN Agent

[11]:

```
class DeepQNetwork():
    def __init__(self, states, actions, alpha, gamma, epsilon, epsilon_min, epsilon_decay):
        self.nS = states
        self.nA = actions
        self.memory = deque([], maxlen=2500)
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_min = epsilon_min
        self.epsilon_decay = epsilon_decay
        self.model = self.build_model()
        self.loss = []

    def build_model(self):
        model = keras.Sequential()
        model.add(keras.layers.Dense(24, input_dim=self.nS, activation='relu'))
        model.add(keras.layers.Dense(24, activation='relu'))
        model.add(keras.layers.Dense(self.nA, activation='linear'))
        model.compile(loss='mean_squared_error',
                      optimizer=keras.optimizers.Adam(learning_rate=self.alpha))
        return model

    def action(self, state):
        if np.random.rand() <= self.epsilon:
            return random.randrange(self.nA)
        action_vals = self.model.predict(state, verbose=0)
        return int(np.argmax(action_vals[0]))

    def test_action(self, state):
        action_vals = self.model.predict(state, verbose=0)
        return int(np.argmax(action_vals[0]))

    def store(self, state, action, reward, nstate, done):
        self.memory.append((state, action, reward, nstate, done))

    def experience_replay(self, batch_size):
        minibatch = random.sample(self.memory, batch_size)
        x, y = [], []

        np_array = np.array(minibatch, dtype=object)
        st  = np.vstack(np_array[:,0].reshape(-1))
        nst = np.vstack(np_array[:,3].reshape(-1))

        st_predict  = self.model.predict(st,  verbose=0)
        nst_predict = self.model.predict(nst, verbose=0)

        for idx, (state, action, reward, nstate, done) in enumerate(minibatch):
            target = reward if done else reward + self.gamma * np.amax(nst_predict[idx])
            target_f = st_predict[idx]
            target_f[action] = target
            x.append(state.reshape(self.nS,))
            y.append(target_f)

        x_reshape = np.array(x).reshape(len(x), self.nS)
        y_reshape = np.array(y)
        hist = self.model.fit(x_reshape, y_reshape, epochs=1, verbose=0)
        self.loss.append(hist.history['loss'][0])

        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

## Cell 3 – Cart pole ‘before’ window

[14]:

```
env = make_env_human()
obs, info = reset_compat(env, seed=50)
for _ in range(400):
    action = np.random.randint(0, env.action_space.n)
    obs, reward, done, info = step_compat(env, action)
    if done:
        obs, info = reset_compat(env)
env.close()
```

## Cell 4 – Training

```
[15]: envCartPole = make_env_silent()
      obs, info = reset_compat(envCartPole, seed=50)

      nS = envCartPole.observation_space.shape[0]
      nA = envCartPole.action_space.n
      dqn = DeepQNetwork(nS, nA, learning_rate(), discount_rate(), 1.0, 0.001, 0.995)

      rewards, epsilons = [], []
      TEST_Episodes = 0
      BATCH = batch_size()

      for e in range(EPISODES):
          obs, info = reset_compat(envCartPole)
          state = np.reshape(obs, (1, nS))
          tot_rewards = 0
          for t in range(210):
              action = dqn.action(state)
              nobs, reward, done, _ = step_compat(envCartPole, action)
              nstate = np.reshape(nobs, (1, nS))
              tot_rewards += reward
              dqn.store(state, action, reward, nstate, done)
              state = nstate

              if done or t == 209:
                  rewards.append(tot_rewards)
                  epsilons.append(dqn.epsilon)
                  print(f"episode: {e}/{EPISODES}, score: {tot_rewards}, e: {dqn.epsilon:.3f}")
                  break

              if len(dqn.memory) > BATCH:
                  dqn.experience_replay(BATCH)

          if len(rewards) > 5 and np.average(rewards[-5:]) > 195:
              TEST_Episodes = EPISODES - e
              TRAIN_END = e
              break

      envCartPole.close()
```

## Cell 5 – Plot

```
[17]:
      rolling = np.convolve(rewards, np.ones( min(100, len(rewards)) ) / min(100, len(rewards)))
      plt.figure()
      plt.plot(rewards, label="rewards")
      plt.plot(rolling, color='black', label="rolling average")
      plt.xlim((0, len(rewards)))
      plt.legend()
      plt.show()
```

## Cell 6 – Cart pole ‘after’ window

```
[18]: env = make_env_human()
      obs, info = reset_compat(env)

      for _ in range(600):  # ~20 s
          action = dqn.test_action(np.reshape(obs, (1, env.observation_space.shape[0])))
          obs, reward, done, info = step_compat(env, int(action))
          if done:
              obs, info = reset_compat(env)

      env.close()
```