Sample Code (SGD)

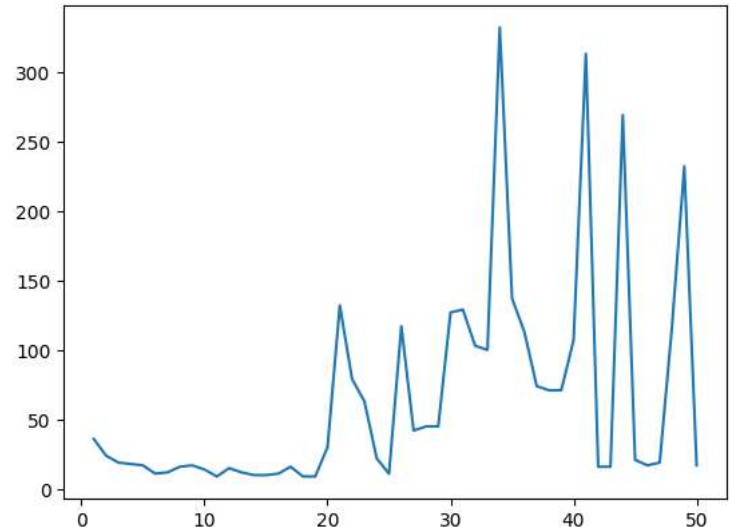
```
import gym
import numpy as np
import keras
import tensorflow as tf
```

```
import tensorflow as tf
from tensorflow import keras
from collections import deque
import numpy as np
import random

MAX_LEN =2000
BATCH_SIZE = 64
GAMMA = 0.9
EXPLORATION_DECAY = 0.9
EXPLORATION_MIN = 0.1


class Agent(object):
    def __init__(self, input_space, output_space, lr=0.001, exploration=0.9):
        self._model = keras.Sequential()
        self._model.add(keras.layers.Dense(input_shape=(input_space,), units=24, activation=tf.nn.relu))
        self._model.add(keras.layers.Dense(units=24, activation=tf.nn.relu))
        self._model.add(keras.layers.Dense(units=output_space, activation='linear'))
        self._model.compile(loss='mse', optimizer=keras.optimizers.Adam(lr))

        self._replayBuffer = deque(maxlen=MAX_LEN)
        self._exploration = exploration

    @property
    def exploration(self):
        return self._exploration

    def add_data(self, state, action, reward, state_next, done):
        self._replayBuffer.append((state, action, reward, state_next, done))

    def act(self, state):
        if np.random.uniform() <= self._exploration:
            return np.random.randint(0, 2)
        action = self._model.predict(state,verbose=0)
        return np.argmax(action[0])
    # def train_from_buffer(self):
    #     if len(self._replayBuffer) < BATCH_SIZE:
    #         return
    #     batch = random.sample(self._replayBuffer, BATCH_SIZE)
    #     states = np.array([sample[0] for sample in batch])
    #     actions = np.array([sample[1] for sample in batch])
    #     rewards = np.array([sample[2] for sample in batch])
    #     next_states = np.array([sample[3] for sample in batch])
    #     dones = np.array([sample[4] for sample in batch])

    #     # Predict Q-values for current states and next states
    #     current_q_values = self._model.predict(states, verbose=0)
    #     next_q_values = self._model.predict(next_states, verbose=0)

    #     # Calculate target Q-values
    #     target_q_values = np.copy(current_q_values)
    #     for i in range(BATCH_SIZE):
    #         if dones[i]:
    #             target_q_values[i, actions[i]] = rewards[i]
    #         else:
    #             target_q_values[i, actions[i]] = rewards[i] + GAMMA * np.amax(next_q_values[i])

    #     # Train the model using the target Q-values and states
    #     self._model.fit(states, target_q_values, verbose=0, epochs=1)

    #     # Decay exploration rate
    #     self._exploration *= EXPLORATION_DECAY
    #     self._exploration = max(EXPLORATION_MIN, self._exploration)
```

```
    def train_from_buffer(self):
        if len(self._replayBuffer) < BATCH_SIZE:
            return
        batch = random.sample(self._replayBuffer, BATCH_SIZE)
        for state, action, reward, state_next, done in batch:
            if done:
                q_update = reward
            else:
                q_update = reward + GAMMA * np.amax(self._model.predict(state_next, verbose=0))
            q_values = self._model.predict(state, verbose=0)
            q_values[0][action] = q_update
            self._model.fit(state, q_values, verbose=0)
        self._exploration *= EXPLORATION_DECAY
        self._exploration = max(EXPLORATION_MIN, self._exploration)
```

```
import gym
import numpy as np
import matplotlib.pyplot as plt

def train():
    env = gym.make("CartPole-v1")
    env.reset()
    input_space = env.observation_space.shape[0]
    output_space = env.action_space.n
    print(input_space, output_space)
    agent = Agent(input_space, output_space)
    run = 0
    x = []
    y = []
    while run < 50:
        run += 1
        state = env.reset()
        state = np.reshape(state, [1, -1])
        step = 0
        while True:

            step += 1
            action = agent.act(state)
            state_next, reward, done, _ = env.step(action)
            reward = reward if not done else -reward
            state_next = np.reshape(state_next, [1, -1])
            agent.add_data(state, action, reward, state_next, done)
            state = state_next
            if done:
                print("Run: " + str(run) + ", exploration: " +
                      str(agent.exploration) + ", score:" + str(step))
                x.append(run)
                y.append(step)
                break
        agent.train_from_buffer()
    agent._model.save("trained_model.h5")

    # Load the saved model and display the summary


    plt.plot(x, y)
    plt.show()


if __name__ == "__main__":
    train()
```

```
/usr/local/lib/python3.10/dist-packages/gym/core.py:317: DeprecationWarning: WARN: Initi
  deprecation(
/usr/local/lib/python3.10/dist-packages/gym/wrappers/step_api_compatibility.py:39: Depre
  deprecation(
4 2
Run: 1, exploration: 0.9, score:36
Run: 2, exploration: 0.9, score:24
Run: 3, exploration: 0.9, score:19
Run: 4, exploration: 0.81, score:18
Run: 5, exploration: 0.7290000000000001, score:17
Run: 6, exploration: 0.6561000000000001, score:11
Run: 7, exploration: 0.5904900000000002, score:12
Run: 8, exploration: 0.5314410000000002, score:16
Run: 9, exploration: 0.47829690000000014, score:17
Run: 10, exploration: 0.43046721000000016, score:14
Run: 11, exploration: 0.38742048900000015, score:9
Run: 12, exploration: 0.34867844010000015, score:15
Run: 13, exploration: 0.31381059609000017, score:12
Run: 14, exploration: 0.28242953648100017, score:10
Run: 15, exploration: 0.25418658283290013, score:10
Run: 16, exploration: 0.22876792454961012, score:11
Run: 17, exploration: 0.2058911320946491, score:16
Run: 18, exploration: 0.1853020188851842, score:9
Run: 19, exploration: 0.16677181699666577, score:9
Run: 20, exploration: 0.1500946352969992, score:30
Run: 21, exploration: 0.13508517176729928, score:132
Run: 22, exploration: 0.12157665459056936, score:79
Run: 23, exploration: 0.10941898913151243, score:63
Run: 24, exploration: 0.1, score:22
Run: 25, exploration: 0.1, score:11
Run: 26, exploration: 0.1, score:117
Run: 27, exploration: 0.1, score:42
Run: 28, exploration: 0.1, score:45
Run: 29, exploration: 0.1, score:45
Run: 30, exploration: 0.1, score:127
Run: 31, exploration: 0.1, score:129
Run: 32, exploration: 0.1, score:103
Run: 33, exploration: 0.1, score:100
Run: 34, exploration: 0.1, score:332
Run: 35, exploration: 0.1, score:137
Run: 36, exploration: 0.1, score:113
Run: 37, exploration: 0.1, score:74
Run: 38, exploration: 0.1, score:71
Run: 39, exploration: 0.1, score:71
Run: 40, exploration: 0.1, score:107
Run: 41, exploration: 0.1, score:313
Run: 42, exploration: 0.1, score:16
Run: 43, exploration: 0.1, score:16
Run: 44, exploration: 0.1, score:269
Run: 45, exploration: 0.1, score:21
Run: 46, exploration: 0.1, score:17
Run: 47, exploration: 0.1, score:19
Run: 48, exploration: 0.1, score:117
Run: 49, exploration: 0.1, score:232
Run: 50, exploration: 0.1, score:17
```




```
from tensorflow.keras.models import load_model

if __name__ == "__main__":

loaded_model = load_model("trained_model.h5")
```

```
loaded_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 24)                120

 dense_7 (Dense)             (None, 24)                600

 dense_8 (Dense)             (None, 2)                 50

=================================================================
Total params: 770
Trainable params: 770
Non-trainable params: 0
_________________________________________________________________
```