CartPole-v1 with TensorFlow2

Problem gym.openai.com/envs/CartPole-v1/

Setup

```
import numpy as np
import os
import pandas as pd
import random
from google.colab import drive
```

```
try:
    import gym
except:
    !pip install gym
import gym
print(gym.__version__)
```

```
0.10.11
```

```
try:
    import tensorflow as tf
except:
    !pip install tensorflow==2.0.0-beta1
    import tensorflow as tf
if tf.__version__[0] == "1":
    !pip install tensorflow==2.0.0-beta1
    import tensorflow as tf
print(tf.__version__)
```

```
2.0.0-beta1
```

```
drive.mount('/content/gdrive', force_remount=False)
```

```
Drive already mounted at /content/gdrive; to attempt to forcib
```

Visualize gym environment - random action

```
import gym
env = gym.make("CartPole-v1")
no_actions = env.action_space.n
```

```
no_observations = env.observation_space.shape[0]
print(no_actions)
print(no_observations)
```

```
2
4
```

Action and observation interpretations: github.com/openai/gym/blob/master/gym/envs/cl

```
observation = env.reset()
for ep in range(10):
    state = env.reset()
    for t in range(100):
#         env.render()
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

```
Episode finished after 32 timesteps
Episode finished after 12 timesteps
Episode finished after 25 timesteps
Episode finished after 13 timesteps
Episode finished after 12 timesteps
Episode finished after 14 timesteps
Episode finished after 33 timesteps
Episode finished after 16 timesteps
Episode finished after 11 timesteps
Episode finished after 22 timesteps
```

Train with TF2, Keras

```
def generate_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(
        10,
        activation="relu",
        input_shape=(no_observations,)
        )
    )
    model.add(tf.keras.layers.Dense(10, activation="relu"))
    model.add(tf.keras.layers.Dense(no_actions, activation="linear"))
    model.compile(loss="mse", optimizer=tf.keras.optimizers.Adam(lr=0.001))
    return model
```

```
# ALPHA = 1
# GAMMA = 0.95
# model = generate_model()
# def fit_model(state, action, reward, next_state, done):
```

```
#     q = model.predict(np.array([state]))
#     q_next = model.predict(np.array([next_state]))
#     q_update = reward
#     if not done:
#         q_update = ((1-ALPHA)*q[0, action] +
#                     ALPHA*(reward + GAMMA * q_next.max(axis=1)))
#     q[0, action] = q_update
#     model.fit(np.array([state]), q, epochs=1, batch_size=1)
```

```
ALPHA = 1
GAMMA = 0.95
state_history = []
action_history = []
reward_history = []
next_state_history = []
model = generate_model()
def fit_model(state, action, reward, next_state, done):
    state_history = []
    action_history = []
    next_state_history = []
    reward_history = []
    state_history.append(state)
    action_history.append(action)
    next_state_history.append(next_state)
    reward_history.append(reward)
    q = model.predict(np.array(state_history))
    # print("q")
    # print(q)
    q_next = model.predict(np.array(next_state_history))
    # print("q next")
    # print(q_next)
    q_update = reward_history
    # print("q update")
    # print(q_update)
    if not done:
        q_update = ((1-ALPHA)*q[np.arange(len(q)), action_history] +
                    ALPHA*(reward_history + GAMMA * q_next.max(axis=1)))
    q[np.arange(len(q)), action_history] = q_update
    # print(q_update)
    # print(next_best_actions)
    # print(q)
    model.fit(np.array(state_history), q, epochs=1, batch_size=32)
```

```
%%capture
for ep in range(1500):
    state = env.reset()
    for t in range(1000):
#         env.render()
        if random.random() < 0.7:
            action = env.action_space.sample()
        else:
            action = model.predict(np.array([state])).argmax(axis=1)[0]
        next_state, reward, done, info = env.step(action)
        fit_model(state, action, reward, next_state, done)
        fit_model(state, action, reward, next_state, done)
        state = next_state
        if done:
            print(f"Episode {ep} finished after {t} timesteps")
            break
```

Test with TF2, Keras

```
for ep in range(20):
    state = env.reset()
#     env.render()
    for t in range(1000):
        action = model.predict(np.array([state])).argmax(axis=1)[0]
        next_state, reward, done, info = env.step(action)
        if done:
            print(f"Episode {ep} finished after {t} timesteps")
            break
        state = next_state
env.close()
```

```
Episode 0 finished after 250 timesteps
Episode 1 finished after 292 timesteps
Episode 2 finished after 435 timesteps
Episode 3 finished after 255 timesteps
Episode 4 finished after 268 timesteps
Episode 5 finished after 239 timesteps
Episode 6 finished after 323 timesteps
Episode 7 finished after 227 timesteps
Episode 8 finished after 224 timesteps
Episode 9 finished after 245 timesteps
Episode 10 finished after 276 timesteps
Episode 11 finished after 262 timesteps
Episode 12 finished after 499 timesteps
Episode 13 finished after 228 timesteps
Episode 14 finished after 277 timesteps
Episode 15 finished after 294 timesteps
Episode 16 finished after 402 timesteps
Episode 17 finished after 240 timesteps
Episode 18 finished after 294 timesteps
Episode 19 finished after 499 timesteps
```

```
loc = "/content/gdrive/My Drive/Colab Notebooks/gym-openai-cartpole/model.h
model.save(loc)
loaded_model = tf.keras.models.load_model(loc)
loaded_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 10)                50
_________________________________________________________________
dense_4 (Dense)              (None, 10)                110
```

```
     _
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 22
=================================================================
Total params: 182
Trainable params: 182
Non-trainable params: 0
_________________________________________________________________
```