Project B (OpenAI Gym agent) Third Report

---- Deep Reinforcement Learning Agent using Policy Gradients

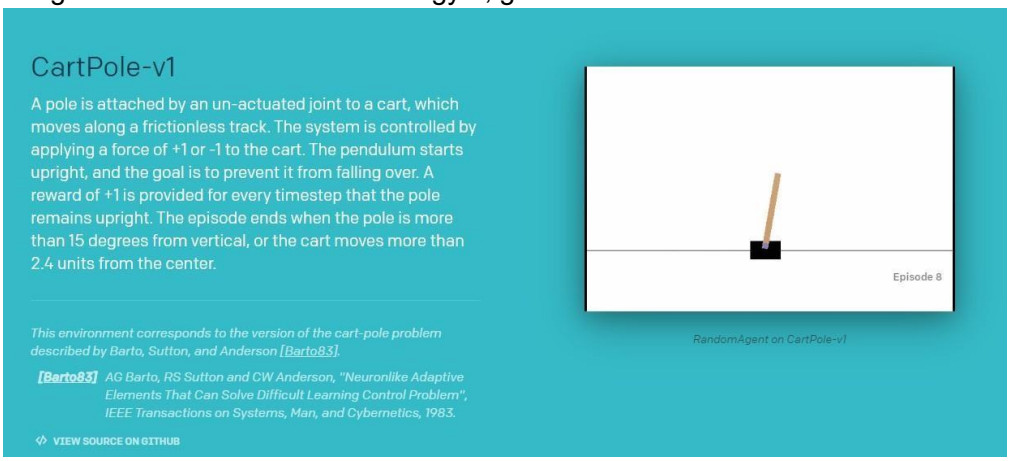
2017201980

李平山

12/13

Introductions:

Using 'CartPole-v1' environment in gym, game introduction:



[Retrieved from https://gym.openai.com/envs/CartPole-v1/]

Test Environment:

- windows 10,python 3, Tensorflow 2.0.0

Brief introduction to Policy Gradients:

Policy gradients is a policy-based reinforcement learning technique.
In policy-based methods, instead of learning a value function(as it is in DQN) that tells us what is the expected sum of rewards given a state and an action, we learn directly the policy function that maps state to action, that is to say, we select actions without using a value function.

$$\pi_\theta(a|s) = P[a|s]$$

In policy search, we have our policy π that has a parameter θ. This π outputs a probability distribution of actions.

$$J(\theta) = E_{\pi\theta}[\sum \gamma r]$$

Secondly, we build our Policy Score function in the model. Remember that policy can be seen as an optimization problem. We must find the best parameters (θ) to maximize a score function, J(θ). The main idea here is that J(θ) will tell us how good our π is. Policy gradient ascent will help us to find the best policy parameters to maximize the sample of good actions.

$$Policy: \pi_\theta$$
$$Objective\ function: J(\theta)$$
$$Gradient: \nabla_\theta J(\theta)$$
$$Update: \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

And set the Policy gradient ascent. In normal cases, policy gradient ascent functioned as above.

Experimental Procedure：

1） Set the model using Keras Sequential

```
In [104]: model = Sequential()
          model.add(Dense(64, input_dim=state_d, activation='relu'))
          model.add(Activation('relu'))
          model.add(Dropout(dropout_rate))
          model.add(Dense(64))
          model.add(Activation('relu'))
          model.add(Dropout(dropout_rate))
          model.add(Dense(action_d))
          model.add(Activation('softmax'))
          model.compile(loss = 'mse', optimizer=optimizers.Adam(0.001))
```

We set two dense levels in the network, using 'relu' as activation function.

Note that we use Adam optimizer, as defined in keras reference:

Adam [source]

```
keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
```

Adam optimizer.

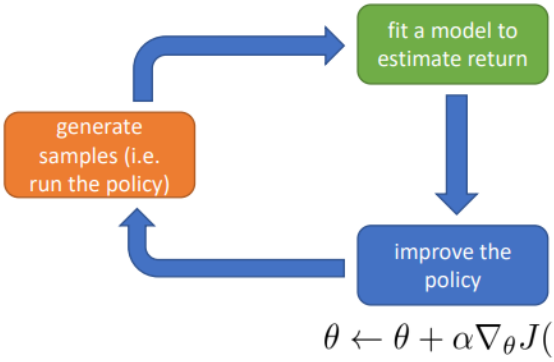
Default parameters follow those provided in the original paper.

Arguments

- **learning_rate**: float >= 0. Learning rate.
- **beta_1**: float, 0 < beta < 1. Generally close to 1.
- **beta_2**: float, 0 < beta < 1. Generally close to 1.
- **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

2) Set gradient ascent algorithm

Gradient ascent is the inverse of gradient descent. Remember that gradient always points to the steepest change. In gradient descent, we take the direction of the steepest decrease in the function. In gradient ascent we take the direction of the steepest increase of the function.



In this particular case of CartPole-v1, we can compare gradient ascent to loss function, using discount_value to calculate for simplicity (as later proved to be of good performance)

```
model.fit(states, probs, sample_weight=discount_value, verbose=0)
```

Note that discount_value is used to calculate gradient ascent after normalization.

```
def discount_rewards(rewards, gamma=0.975):
    prior = 0
    out = np.zeros_like(rewards)
    for i in reversed(range(len(rewards))):
        prior = prior * gamma + rewards[i]
        out[i] = prior
    return out / np.std(out - np.mean(out))
```

Using a decrease rate as gamma of 0.975.

And that we calculate probs in one-hot coding to describe the actions.

eg: P(0.1,0.9) -> probs.add(0,1)

Also note that we use sample_weight in tensorflow.keras to cal loss function as defined below (from reference): Optional Numpy array of weights for the training samples, used for weighting the loss function (during training only). You can either pass a flat (1D) Numpy array with the same length as the input samples

3) Train the model

Before training, we can find that in CartPole, reward is forever 1 in every possible timestep,to accelerate the process of training the model, we update the reward in the following rules:

```
def cal_score(score):
        if(score > 400): return 1 + 1
        elif (score > 200): return 1 + 0.3
        else: return 1
```

Because it is pre-known that the winning condition in the CartPole-v0 and CartPole-v1 is 200 timesteps and 500 timesteps in one episode, we set the parameters in the function, 200 and 400 and the bonus rewards based on the rules and the actual performance.

Set the max_episode to be 2000 and start the training :

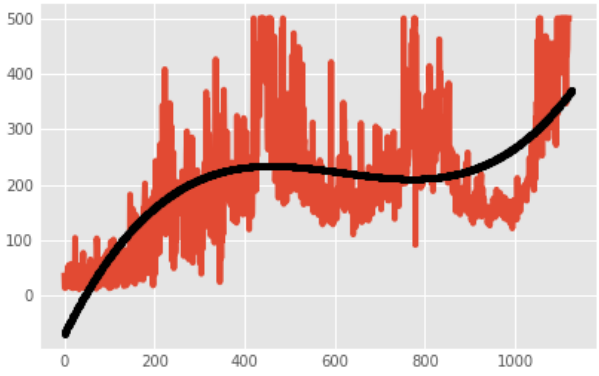
```
for i in range(max_episodes):
    s = env.reset()
    score = 0
    replay_records = []
    while True:
        a = act(s)
        next_s, r, done, _ = env.step(a)
        r = cal_score(score)
        replay_records.append((s, a, r))
        score += r
        s = next_s
        if done:
```

```
            train(replay_records)
            score_list.append(score)
            break
```

4) Performance and test results

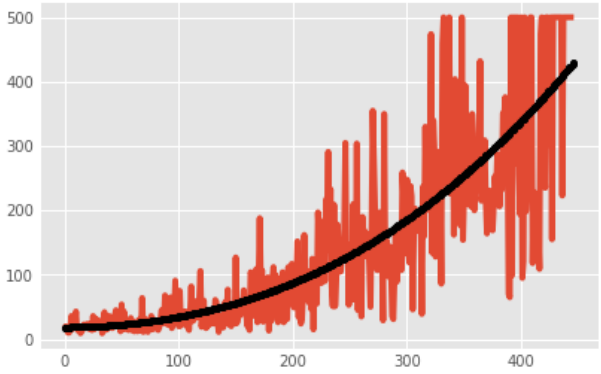
The performance of the model relay much on the parameters mentioned in sections above.

For example, a model using 'tahh' acceleration function without using optimized-reward algorithm and dropout strategy may look like this as performance:



It may encounter overfitting issues and use more episodes to solve the problem.

After adding dropout strategy, setting better parameters and optimize the functions, we can get a better performed model as below(in average)



Check the model after reset the environment:

```
CartPole-v1 test starts:
  test result using trained policy gradient model: 500.0
  test result using trained policy gradient model: 500.0
  test result using trained policy gradient model: 500.0
  test result using trained policy gradient model: 500.0
  test result using trained policy gradient model: 500.0
```

We can see from the episodes that all reached best score!