Mountain Car - v0

https://gym.openai.com/envs/MountainCar-v0/ (https://gym.openai.com/envs/MountainCar-v0/)

Main Idea

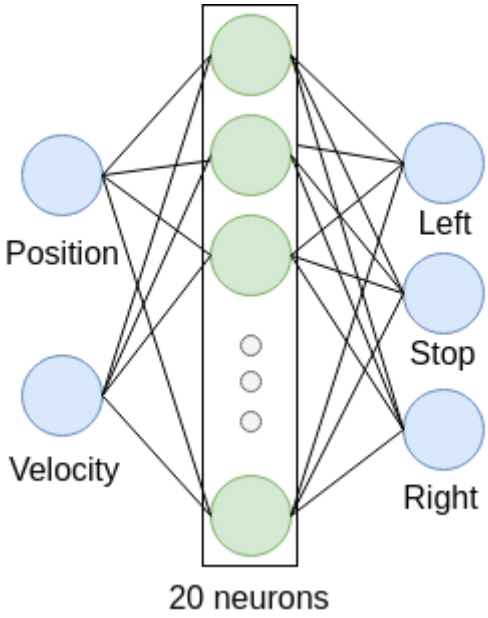
I tried to use Q-learning, Sarsa, SarsaLambda, Deep Q Network, Policy Gradient to implement this program. After the result analysis, I decide to pick **Deep Q Network** method. In the following experiment, they're based on DQN with some difference(i.g. learning rate / batch size).

If you have interest in my ohters method, you can try to run the program with different args so you can observate some interesting results about the project.

Discussion

In the report, I focus on how to **design the network in DQN** and **the reward from the environment**.

Network in DQN



Reward

- distance from the start-point (position = -0.5)
  - abs(position - (-0.5))
- phyiscal work of the car
  - (position_next - position_pres) * (velocity_next)
- special reward : weighten the reward if the car is quitely near to the destination
  - (position_next > 0.5) * (position_next - 0.2) * 10

Experiment

In [1]:

```
import numpy as np
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Read in Data

format: DQN_LearningRate_BatchSize.csv

- e.g. DQN_001_32.csv : learning rate = 0.001, batch size = 32

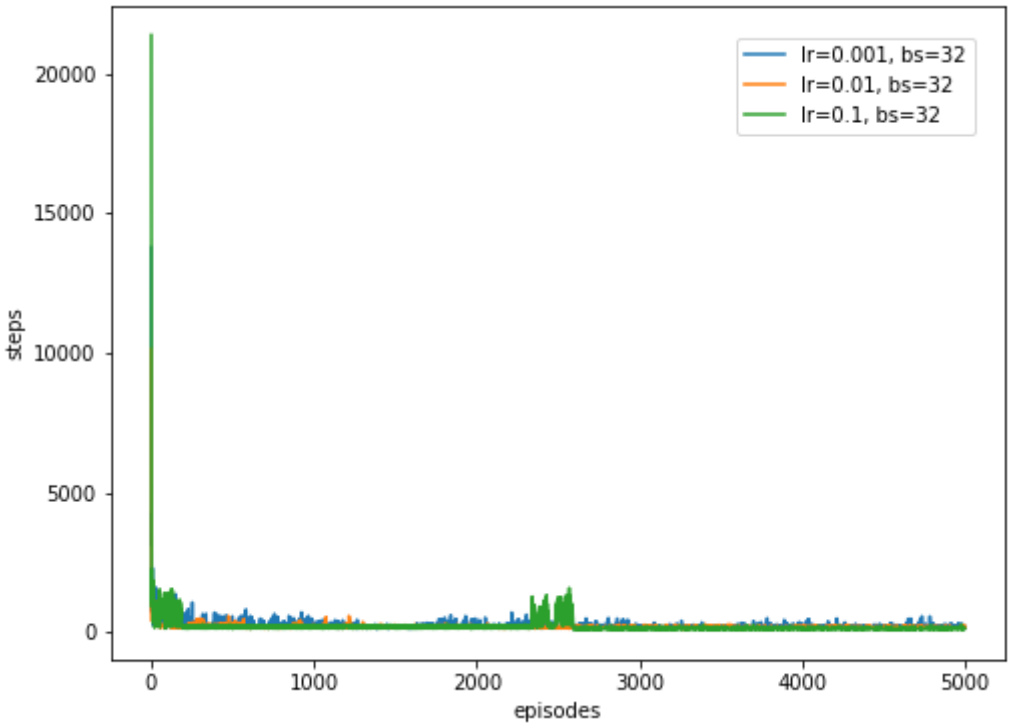
In [2]:

```
DQN_001_32 = pd.read_csv("./record/DQN_0.001_32.csv")['steps'].values
DQN_001_64 = pd.read_csv("./record/DQN_0.001_64.csv")['steps'].values

DQN_01_32 = pd.read_csv("./record/DQN_0.01_32.csv")['steps'].values
DQN_01_64 = pd.read_csv("./record/DQN_0.01_64.csv")['steps'].values

DQN_1_32 = pd.read_csv("./record/DQN_0.1_32.csv")['steps'].values
DQN_1_64 = pd.read_csv("./record/DQN_0.1_64.csv")['steps'].values
DQN_1_128 = pd.read_csv("./record/DQN_0.1_128.csv")['steps'].values
DQN_1_256 = pd.read_csv("./record/DQN_0.1_256.csv")['steps'].values
```

Different Learning Rate

In [3]:

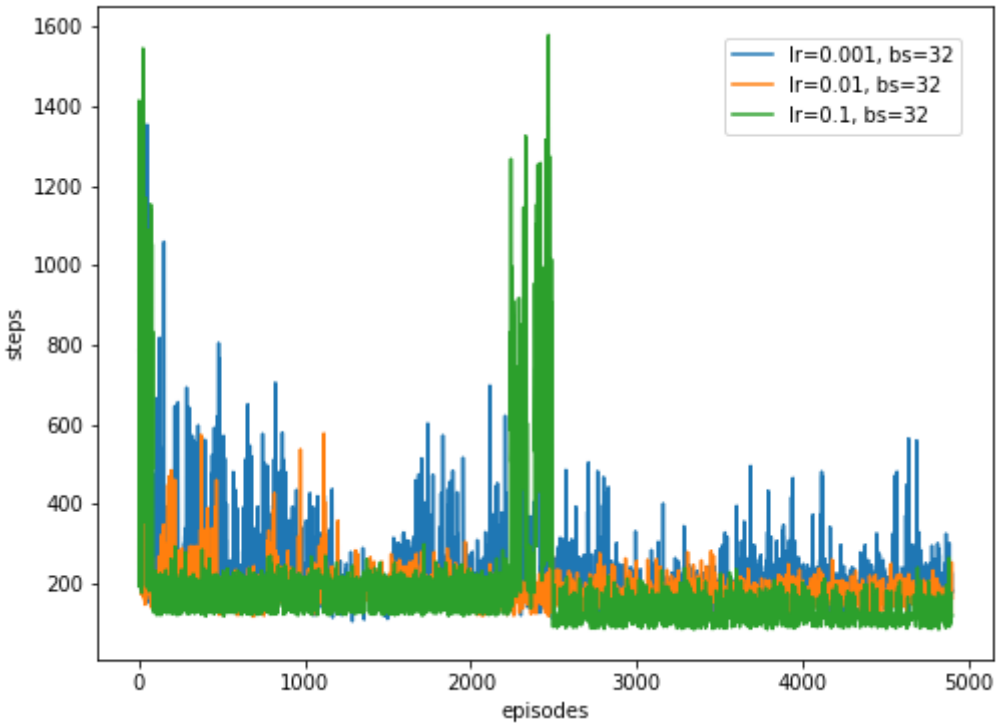
```
plt.figure(figsize=(8,6))
plt.plot(DQN_001_32, label='lr=0.001, bs=32')
plt.plot(DQN_01_32, label='lr=0.01, bs=32')
plt.plot(DQN_1_32, label='lr=0.1, bs=32')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



the number of step are quite large in the head of the records, so I choose the curve from the 100'th episode to show the relation more clearly

In [4]:

```
plt.figure(figsize=(8,6))
plt.plot(DQN_001_32[100:], label='lr=0.001, bs=32')
plt.plot(DQN_01_32[100:], label='lr=0.01, bs=32')
plt.plot(DQN_1_32[100:], label='lr=0.1, bs=32')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```

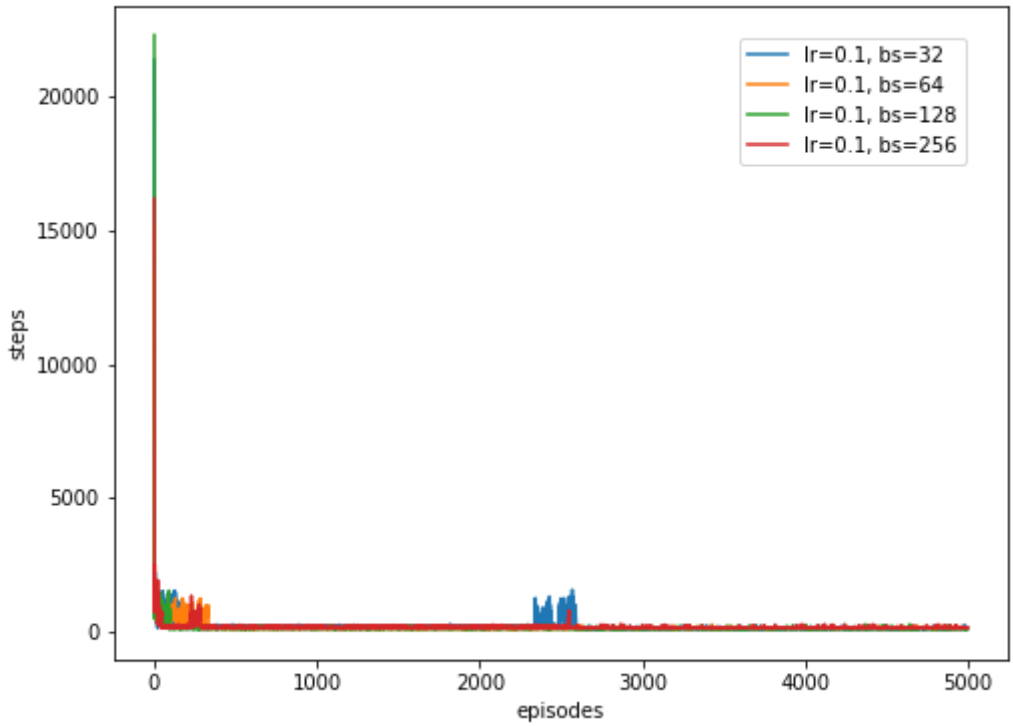


Different Batch Size

Learning Rate = 0.1

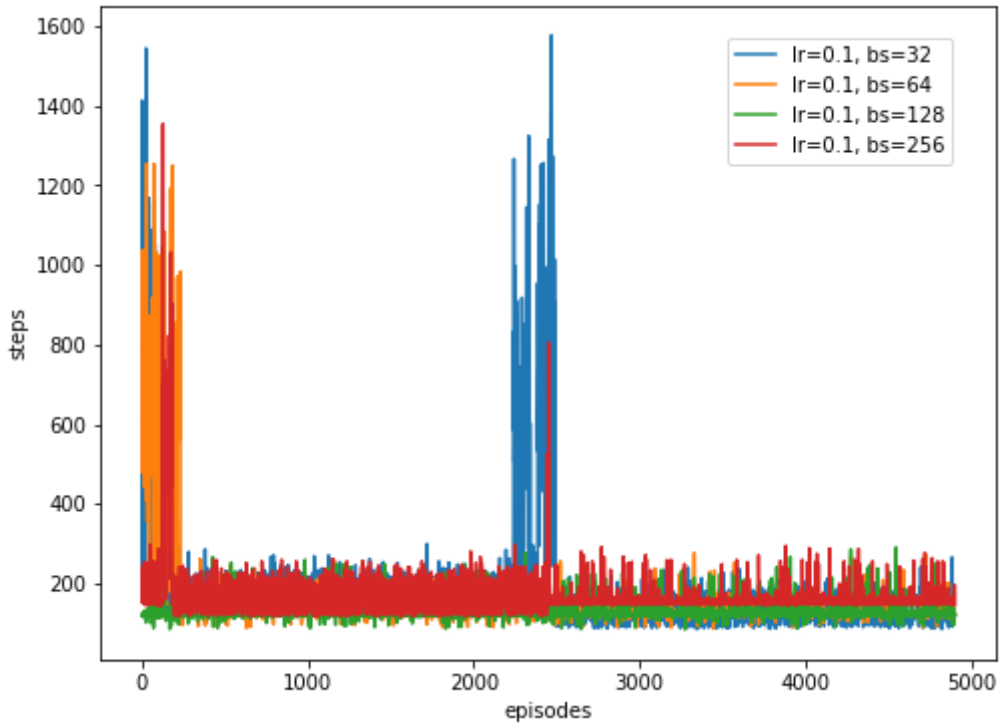
In [5]:

```
plt.figure(figsize=(8,6))
plt.plot(DQN_1_32, label='lr=0.1, bs=32')
plt.plot(DQN_1_64, label='lr=0.1, bs=64')
plt.plot(DQN_1_128, label='lr=0.1, bs=128')
plt.plot(DQN_1_256, label='lr=0.1, bs=256')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



In [6]:

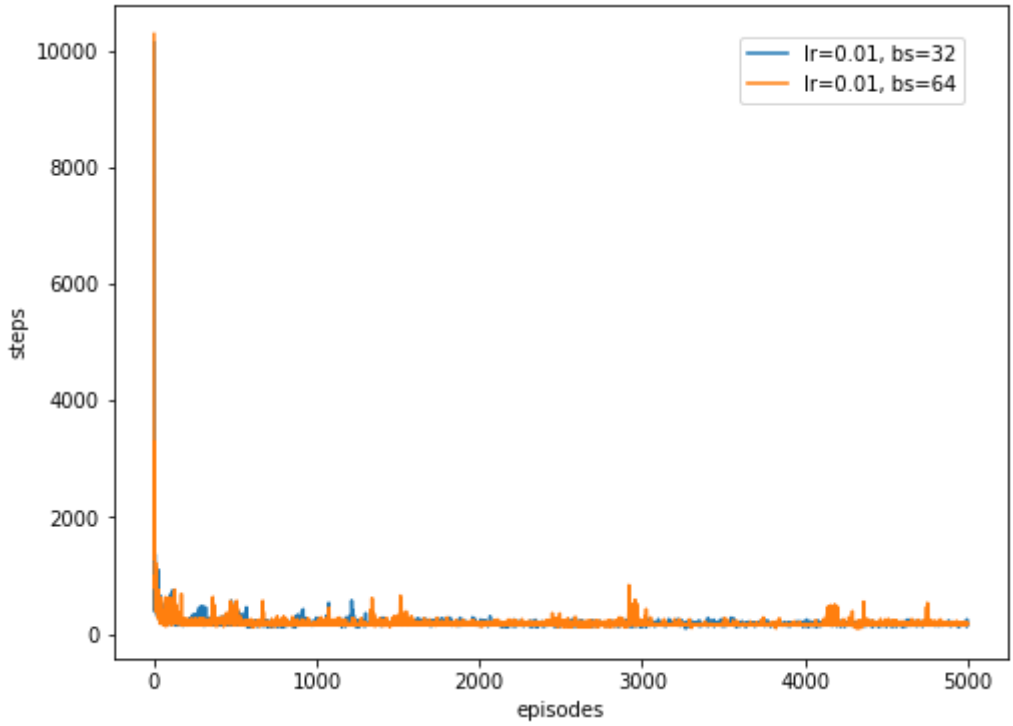
```
plt.figure(figsize=(8,6))
plt.plot(DQN_1_32[100:], label='lr=0.1, bs=32')
plt.plot(DQN_1_64[100:], label='lr=0.1, bs=64')
plt.plot(DQN_1_128[100:], label='lr=0.1, bs=128')
plt.plot(DQN_1_256[100:], label='lr=0.1, bs=256')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



Learning Rate = 0.01

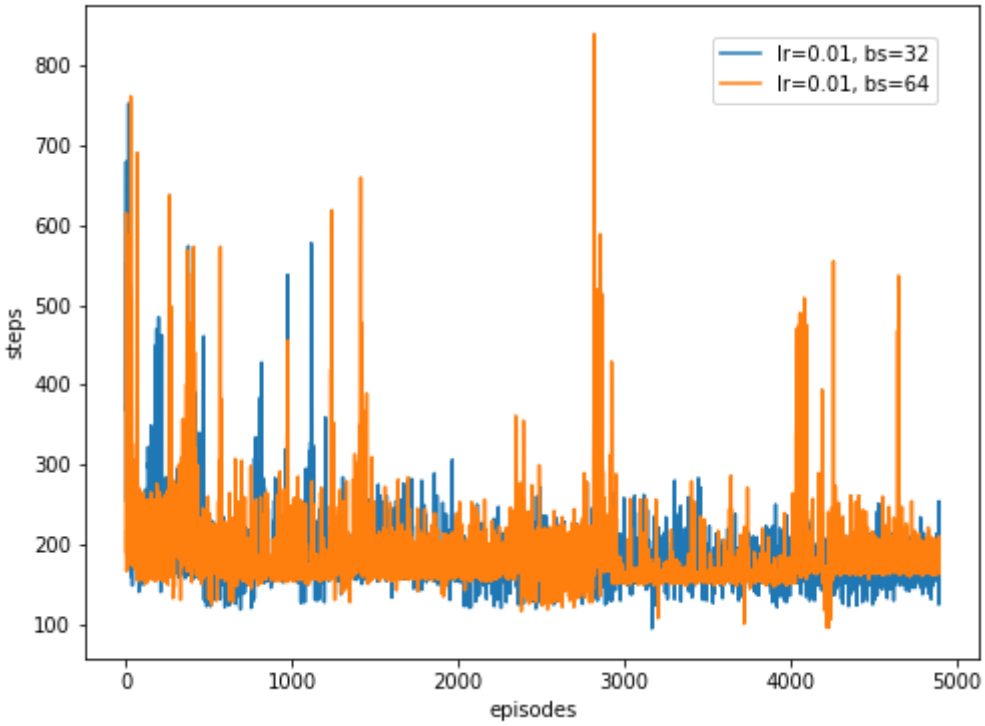
In [7]:

```
plt.figure(figsize=(8,6))
plt.plot(DQN_01_32, label='lr=0.01, bs=32')
plt.plot(DQN_01_64, label='lr=0.01, bs=64')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



In [8]:

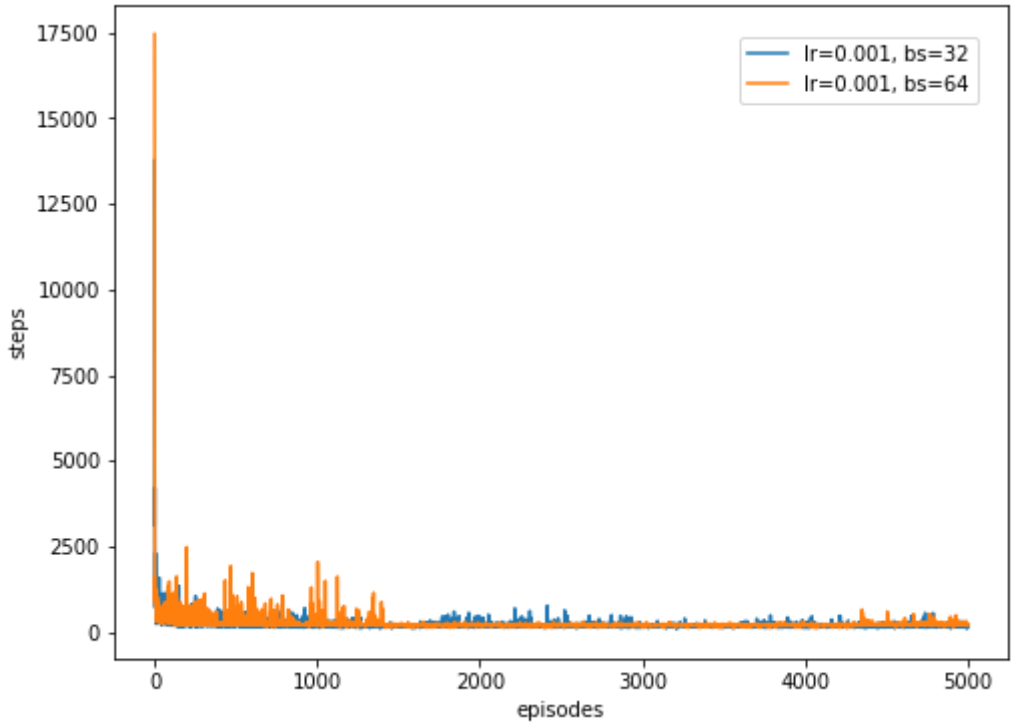
```
plt.figure(figsize=(8,6))
plt.plot(DQN_01_32[100:], label='lr=0.01, bs=32')
plt.plot(DQN_01_64[100:], label='lr=0.01, bs=64')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



Learning Rate = 0.001

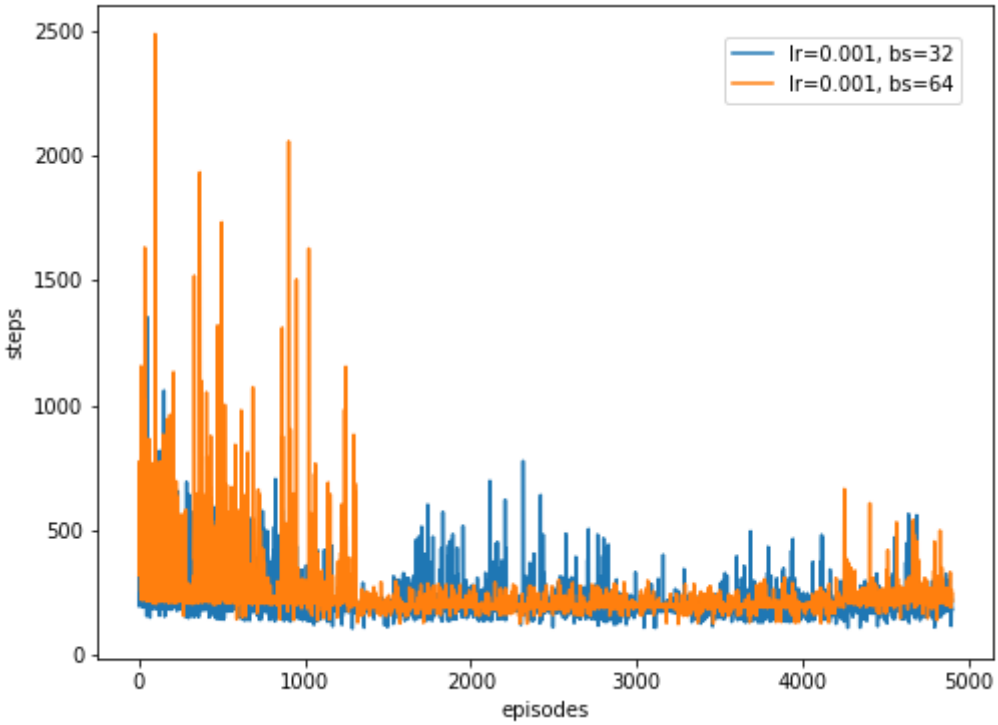
In [9]:

```
plt.figure(figsize=(8,6))
plt.plot(DQN_001_32, label='lr=0.001, bs=32')
plt.plot(DQN_001_64, label='lr=0.001, bs=64')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



In [10]:

```
plt.figure(figsize=(8,6))
plt.plot(DQN_001_32[100:], label='lr=0.001, bs=32')
plt.plot(DQN_001_64[100:], label='lr=0.001, bs=64')
plt.legend(bbox_to_anchor=(0.7, 0.95), loc=2, borderaxespad=0.)
plt.xlabel("episodes")
plt.ylabel("steps")
plt.show()
```



Conclusion

In [11]:

```
print("DQN lr:0.001, batch_size:32, Mean: {}, Min: {}".format(DQN_001_32.mean(),
 DQN_001_32.min()))
print("DQN lr:0.001, batch_size:64, Mean: {}, Min: {}".format(DQN_001_64.mean(),
 DQN_001_64.min()))
print("------------------")
print("DQN lr:0.01, batch_size:32, Mean: {}, Min: {}".format(DQN_01_32.mean(), D
QN_01_32.min()))
print("DQN lr:0.01, batch_size:64, Mean: {}, Min: {}".format(DQN_01_64.mean(), D
QN_01_64.min()))
print("------------------")
print("DQN lr:0.1, batch_size:32, Mean: {}, Min: {}".format(DQN_1_32.mean(), DQN
_1_32.min()))
print("DQN lr:0.1, batch_size:64, Mean: {}, Min: {}".format(DQN_1_64.mean(), DQN
_1_64.min()))
print("DQN lr:0.1, batch_size:128, Mean: {}, Min: {}".format(DQN_1_128.mean(), D
QN_1_128.min()))
print("DQN lr:0.1, batch_size:256, Mean: {}, Min: {}".format(DQN_1_256.mean(), D
QN_1_256.min()))
```

```
DQN lr:0.001, batch_size:32, Mean: 215.6162, Min: 106
DQN lr:0.001, batch_size:64, Mean: 248.5838, Min: 124
------------------
DQN lr:0.01, batch_size:32, Mean: 183.5924, Min: 94
DQN lr:0.01, batch_size:64, Mean: 190.2742, Min: 95
------------------
DQN lr:0.1, batch_size:32, Mean: 198.3772, Min: 85
DQN lr:0.1, batch_size:64, Mean: 163.4776, Min: 86
DQN lr:0.1, batch_size:128, Mean: 142.8926, Min: 84
DQN lr:0.1, batch_size:256, Mean: 178.148, Min: 116
```

The best of the hyper-parameter is **lr=0.1, batch_size=128**. The following is the learing curve in this hyper-parameter setting.

Execute 50 round, and each performs 3000 episodes. Use the mean steps of the nearest 10 episodes to prevent outlier

In [12]:

```
df1 = pd.read_csv("./record/DQN_0.1_128_1.csv")
df2 = pd.read_csv("./record/DQN_0.1_128_2.csv")
df = pd.concat([df1, df2], axis=1)
result = df.rolling(window=10).mean().dropna()
result['mean'] = result.mean(axis=1)
result['std'] = result.std(axis=1)
result.head()
```

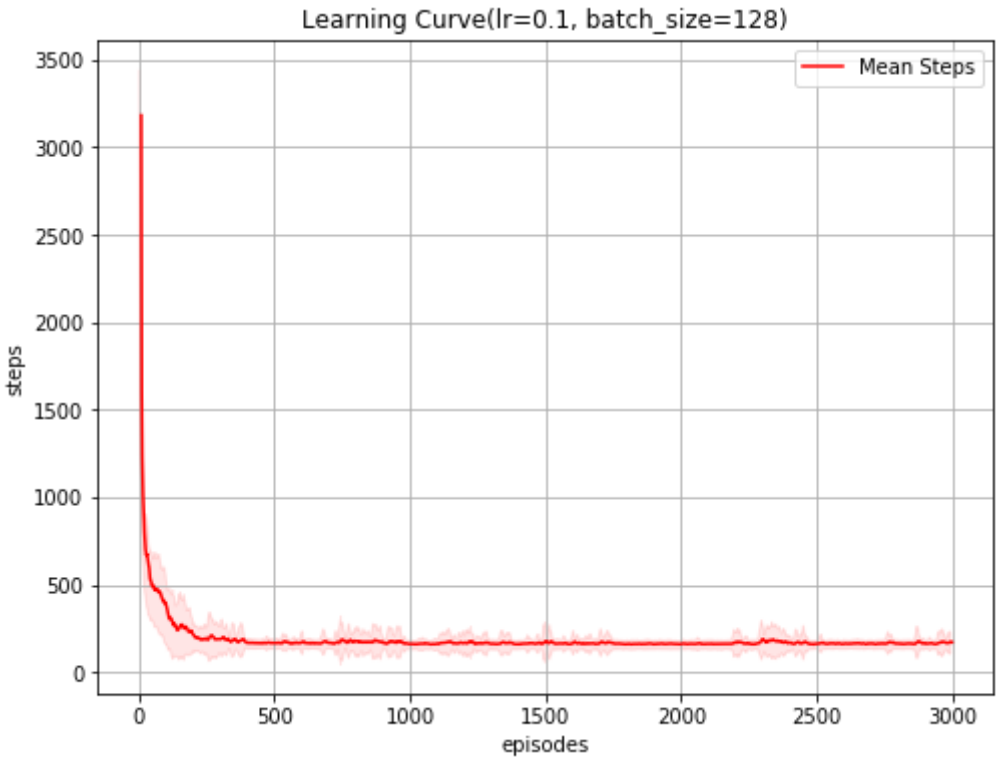
Out[12]:

	0	1	2	3	4	5	6	7	8	9	...	
9	3567.9	3068.2	3349.0	3326.0	2931.5	3099.7	3042.6	3358.2	3233.5	3353.7	...	3
10	1516.2	1802.3	2171.9	2620.8	2160.9	1348.8	1734.3	1763.9	1336.3	1794.2	...	1
11	1443.1	1673.2	1501.9	1227.9	1814.2	1179.5	1249.8	1634.7	1321.0	1552.6	...	1
12	1326.4	1548.5	1359.4	1063.4	1062.0	1164.4	1155.9	1343.6	1303.5	1498.4	...	1
13	1243.2	1529.0	1333.5	962.2	1045.1	982.3	1080.4	1143.6	1357.3	1589.6	...	1

5 rows × 52 columns

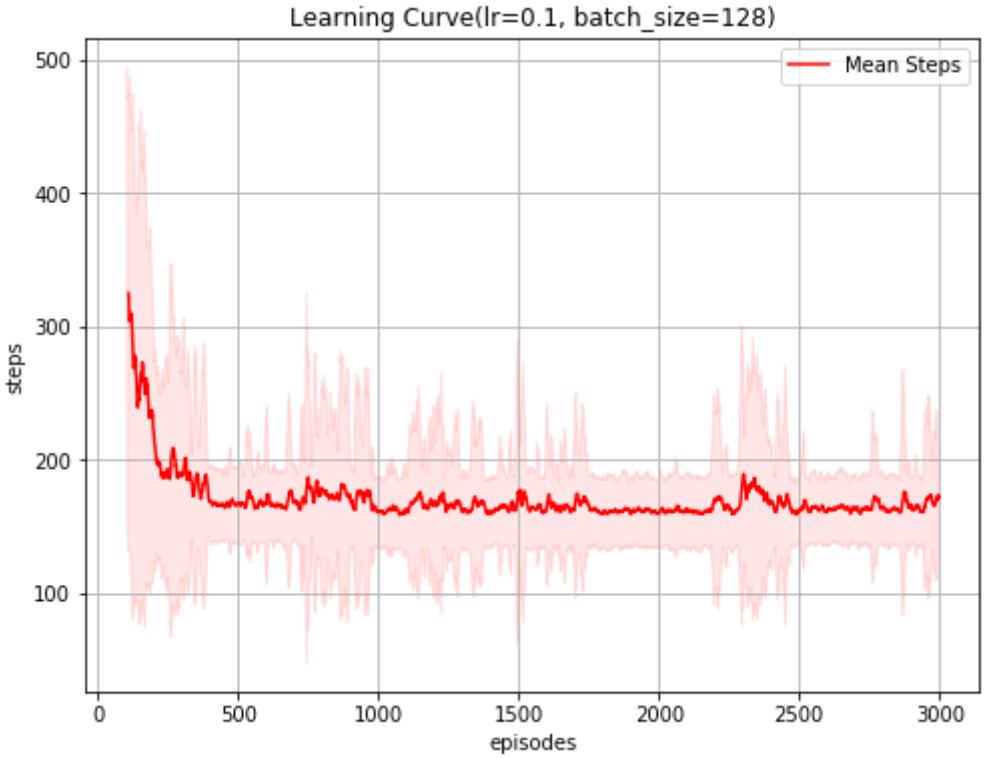
In [13]:

```
plt.figure(figsize=(8, 6))
plt.title("Learning Curve(lr=0.1, batch_size=128)")
plt.xlabel("episodes")
plt.ylabel("steps")

plt.grid()
plt.fill_between(np.arange(2991), result['mean'] - result['std'], result['mean']
 + result['std'], alpha=0.1, color="r")
plt.plot(result['mean'], '-', color="r", label="Mean Steps")
plt.legend(loc="best")
plt.show()
```



In [14]:

```
plt.figure(figsize=(8, 6))
plt.title("Learning Curve(lr=0.1, batch_size=128)")
plt.xlabel("episodes")
plt.ylabel("steps")

plt.grid()
plt.fill_between(np.arange(2891)+100, result['mean'][100:] - result['std'][100
:], result['mean'][100:] + result['std'][100:], alpha=0.1, color="r")
plt.plot(result['mean'][100:], '-', color="r", label="Mean Steps")
plt.legend(loc="best")
plt.show()
```



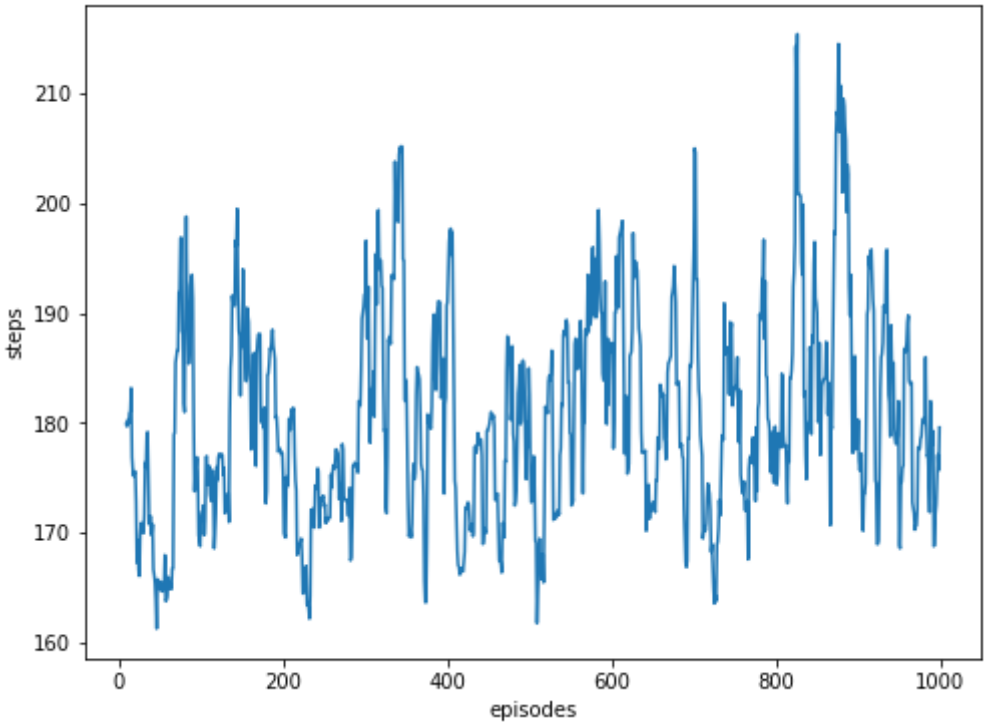
According to the learning curve, I select **lr=0.1, bs=128, episode=2000** as my final model in Mountain Car project.

Testing

use the saved model to test the Mountain Car - v0

In [16]:

```
test = pd.read_csv("./record/test.csv")
plt.figure(figsize=(8, 6))
plt.xlabel("episodes")
plt.ylabel("steps")

plt.plot(test.rolling(window=10).mean().dropna())
plt.show()
```



In [21]:

```
print("DQN Testing: lr:0.1, batch_size:128\n - Mean: {}, Min: {}".format(test.va
lues.mean(), test.values.min()))
```

```
DQN Testing: lr:0.1, batch_size:128
 - Mean: 180.66, Min: 135
```

Question

1. What kind of RL algorithms did you use?

Value-based

Deep Q Network is based on Q-learning. According to the current reward from the last action, we can update the network. Thus, it's **value-based** algorithm.

2. The algorithm is off-policy or on-policy?

Off-policy

Deep Q Network gets the next action by the neural network. We update the parameters in NN with state and reward in each round. It’s off-policy because we can learn from the past experience which stored in the experience pool. So it doesn't calculate the value with the currnect policy only.

3. How does your algorithm solve the correlation problem in the same MDP?

Because DQN has an experience pool. The record is random sampled from that pool to train the neural network. So the correlation problem doesn't influence DQN much. With the concept of the memory pool and experience replay, DQN is more efficient to get the best results.