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

```
env = gym.make("CartPole-v1")
state = env.reset()
state
# położenie, prędkość, kąt, prędkość kątowa
```

```
array([-0.04466579,  0.00737135, -0.03683229,  0.03981359])
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from collections  import deque
import tensorflow as tf
import random
```

Funkcja **Q** aproksymowana przez **sieć neuronową** - *na wejściu* sieci **tensor o kształcie (1,4)**, *na wyjściu* sieci **tensor o kształcie (1,2)** zawierający **2 wartości Q** odpowiadające **akcjom w lewo i prawo**.

```
model = Sequential()
model.add(Dense(units = 40, input_dim=4, activation='relu'))
model.add(Dense(units = 40, activation = "relu"))
model.add(Dense(units = 2, activation = "linear"))
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.1)
#opt = keras.optimizers.SGD(learning_rate=0.001)

model.compile(loss='MSE',optimizer=opt)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 40)                200

 dense_1 (Dense)             (None, 40)                1640

 dense_2 (Dense)             (None, 2)                 82

=================================================================
Total params: 1,922
Trainable params: 1,922
Non-trainable params: 0
_________________________________________________________________
```

Parametry uczenia:

```
train_episodes = 500
epsilon = 0.3
gamma = 0.99
max_steps = 200
```

```
epsilon = 1
```

Pętla treningowa:

```
memory = deque(maxlen=100)
```

```
batch_size = 10
```

```
def train():
  state_batch, Qs_target_batch = [],[]
  minibatch = random.sample(memory, batch_size)
  for state, action, reward, next_state, done in minibatch:
    if done:
      y = reward
    else:
```

```
        y = reward + gamma*np.max(model.predict(next_state)[0])
      Q_target = model.predict(state)
      Q_target[0][action] = y
      state_batch.append(state)
      Qs_target_batch.append(Q_target)
    state_batch = np.array(state_batch).reshape(batch_size,4)
    Qs_target_batch = np.array(Qs_target_batch).reshape(batch_size,2)
    h=model.fit(state_batch,Qs_target_batch,epochs=1,verbose=0)
    loss = h.history['loss'][0]
    return loss


#TODO


Loss = []
Rewards = []

for e in range(1, train_episodes+1):
  epsilon = epsilon -(1/train_episodes)
  total_reward = 0
  t = 0

  state = env.reset()
  state = np.reshape(state, [1, 4])

  done = False
  while t < max_steps and done == False:
    Qs = model.predict(state)[0]
    if np.random.rand()<epsilon:
      action = env.action_space.sample()
    else:
      action = np.argmax(Qs)
    next_state, reward, done, _ = env.step(action)
    next_state = np.reshape(next_state, [1, 4])
    total_reward += reward

    if done:
      y = reward
    else:
      y = reward + gamma*np.max(model.predict(next_state)[0])
```

```
        Q_target = model.predict(state)
        Q_target[0][action] = y


        h = model.fit(state,Q_target,epochs=1,verbose=0)

        loss = h.history['loss'][0]

        state = next_state
        t+=1

    print(e," R=",total_reward," L=",loss)
    Rewards.append(total_reward)
    Loss.append(loss)
```

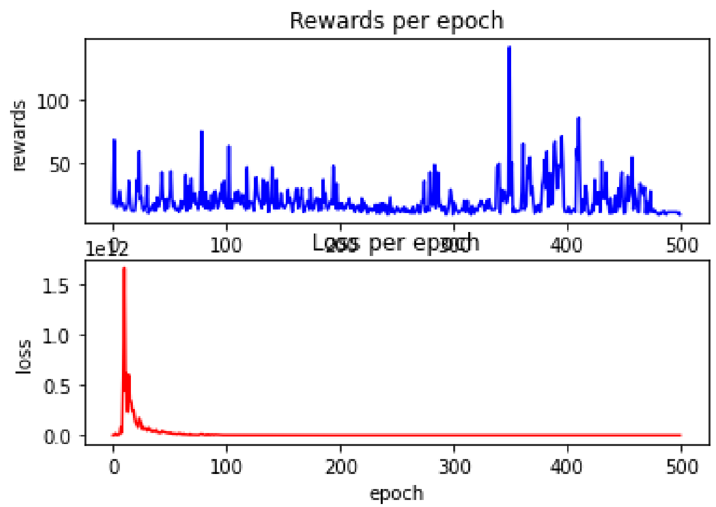
```
443  R= 23.0  L= 5.369323480408639e-05
444  R= 10.0  L= 1.1674829636376671e-07
445  R= 15.0  L= 9.391660569235682e-06
446  R= 29.0  L= 8.55073521961458e-05
447  R= 14.0  L= 4.835545155401633e-07
448  R= 35.0  L= 0.000213358347536996
449  R= 42.0  L= 182380.421875
450  R= 19.0  L= 3.953080067731207e-06
451  R= 9.0  L= 0.00011025447020074353
452  R= 10.0  L= 0.0005991252255626023
453  R= 27.0  L= 0.0011107134632766247
454  R= 39.0  L= 48316.47265625
455  R= 19.0  L= 0.0006817791145294905
456  R= 24.0  L= 24.642030715942383
457  R= 32.0  L= 5585.802734375
458  R= 54.0  L= 8181.9443359375
459  R= 12.0  L= 0.00046841567382216454
460  R= 28.0  L= 338.01898193359375
461  R= 22.0  L= 7025.31787109375
462  R= 18.0  L= 1488282.125
463  R= 10.0  L= 1423951.125
464  R= 10.0  L= 666483.3125
465  R= 33.0  L= 8084.64306640625
466  R= 31.0  L= 1357134.5
467  R= 26.0  L= 0.13452787697315216
468  R= 24.0  L= 574424.125
469  R= 30.0  L= 214408.484375
470  R= 9.0  L= 0.10796257108449936
```

```
471  R= 18.0  L= 0.12369771301746368
472  R= 9.0  L= 0.12782898545265198
473  R= 16.0  L= 0.13687244057655334
474  R= 27.0  L= 1183531.75
475  R= 11.0  L= 0.17250114679336548
476  R= 10.0  L= 0.18277494609355927
477  R= 12.0  L= 0.1959000527858734
478  R= 10.0  L= 0.20361244678497314
479  R= 10.0  L= 0.21109023690223694
480  R= 9.0  L= 0.21670474112033844
481  R= 8.0  L= 0.21742995083332062
482  R= 8.0  L= 0.21909686923027039
483  R= 10.0  L= 0.2269432097673416
484  R= 9.0  L= 0.22834771871566772
485  R= 10.0  L= 0.23135773837566376
486  R= 11.0  L= 0.23512154817581177
487  R= 9.0  L= 0.24112603068351746
488  R= 8.0  L= 0.2409575879573822
489  R= 10.0  L= 0.24631565809249878
490  R= 10.0  L= 0.25107094645500183
491  R= 10.0  L= 0.25701117515563965
492  R= 10.0  L= 0.26228341460227966
493  R= 10.0  L= 0.266231894493103
494  R= 10.0  L= 0.26818257570266724
495  R= 10.0  L= 0.27328887581825256
496  R= 10.0  L= 0.27620071172714233
497  R= 10.0  L= 0.2776587903499603
498  R= 9.0  L= 0.27853983640670776
499  R= 10.0  L= 0.27937060594558716
500  R= 8.0  L= 0.27377310395240784
```

```
plt.subplot(211)
plt.ylabel('rewards')
plt.title('Rewards per epoch')
plt.plot(range(len(Rewards)),Rewards,"b")

plt.subplot(212)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.title('Loss per epoch')
plt.plot(range(len(Loss)),Loss,"r")

plt.show()
```



✓ 0 s ukończono o 23:42