دانشگاه تهران

پردیس دانشکده های فنی

دانشکده مهندسی برق و کامپیوتر

تکلیف شماره ۴ درس یادگیری ماشین

استاد درس :

جناب آقای دکتر نیلی

نام دانشجو :

کوروش محمودی

۸۱۰۱۹۸۰۵۰

سؤال (۱)

یک ارابه میان دو کوه گیر افتاده است و هدف آن این است که به قله ی کوه سمت راست برسد. موتور این ارابه این قدرت را ندارد که به صورت مستقیم از این تپه بالا برود و باید با انجام رفت و برگشت این کار را انجام دهد. در صورتی که ارابه با صرف انرژی کمتر به بالای تپه برسد، پاداش بیشتری دریافت خواهد کرد. همان طور که مشاهده می شود محیط این مسئله یک محیط پیوسته است و لازم است که برای انجام یادگیری ، تخمین ارزش حالت – عمل ها را با استفاده از تعمیم انجام دهیم.

قسمت الف) در این قسمت سعی می کنیم مسئله MountainCar-V0 از مجموعه محیط های OpenAI Gym برای یادگیری عامل هوشمند را به کمک روش یادگیری SARSA(λ) مبتنی بر مقدار (value based) حل کنیم. pseudo code مربوط به این الگوریتم به شکل زیر می باشد.

True online Sarsa(λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or near greedily from S using $\mathbf{w}$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose $A' \sim \pi(\cdot|S')$ or near greedily from S' using $\mathbf{w}$
    $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
    $Q \leftarrow \mathbf{w}^\top \mathbf{x}$
    $Q' \leftarrow \mathbf{w}^\top \mathbf{x}'$
    $\delta \leftarrow R + \gamma Q' - Q$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top \mathbf{x}\right)\mathbf{x}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
    $Q_{old} \leftarrow Q'$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    $A \leftarrow A'$
  until S' is terminal

در این مسئله state پیوسته و متشکل از زوج مرتب مکان-سرعت در بعد افقی می باشد. و سه action هل دادن به راست، هل دادن به چپ و هل ندادن فضای action ها را شکل می دهند. در هر قدم و بعد از انجام هر action عامل reward (-1) دریافت می کند [امتیازی برای رسیدن به قله فرض نشده است] و در نتیجه با رسیدن با قدم های هر چه کمتر تا رسیدن به قله، امتیاز بیشتری دریافت خواهد نمود.

فقط کدینگ RBF: (کد HO#4_RBF_SARSALambbda.py)

برای تعمیم در توابع ارزش حالت-عمل از کدینگ خطی RBF استفاده می کنیم یعنی تابع ویژگی (feature function) X در الگوریتم بالا متشکل از تعدادی تابع RBF هستند. مراکز این RBFها با مش بندی فضای مکان-سرعت به صورت ۵ در ۳، در ۱۵ نقطه قرار خواهند گرفت. ماتریس کوواریانس همه آنها نیز به صورت $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}$ فرض می شود. هر کدام از توابع RBF به صورت زیر تعریف می شود:

$$X = \exp\left(-0.5(s-\mu)^T\Sigma^{-1}(s-\mu)\right)$$

در الگوریتم بالا میزان وزن هر کدام از توابع RBF یعنی بردار w و بردار eligibility trace یعنی Z در هر قدم به روز رسانی می شوند.

ابتدا پارامترهای شبیه سازی را مرور می کنیم.

- سیاست نرم: سیاست $\varepsilon - \text{greedy}$ نسبت به مقادیر ارزش action ها در state ها تنظیم می شود. میزان ε در اولین اپیزود برابر 0.1 است و در هر طول اپیزود ثابت می ماند و با شروع اپیزود بعدی به صورت $\varepsilon \leftarrow \varepsilon * 0.995$ به روز می شود.
- نرخ یادگیری α: مقدار نرخ یادگیری در طول اپیزود ثابت است و برای همه actionها میزان α در اولین اپیزود برابر 0.01 است و با شروع اپیزود بعدی به صورت $\alpha \leftarrow \alpha * 0.999$ به روز می شود.
- میزان طول هر اپیزود ۲۰۰ قدم در نظر گرفته شده است.
- γ Discount factor : مقدار DF برابر $\gamma = 0.98$ در نظر گرفته شده است.
- تعداد اپیزودها: تعداد اپیزودها ۱۰۰۰ در نظر گرفته شده است.
- مقدار λ: برابر 0.8 فرض شده است.

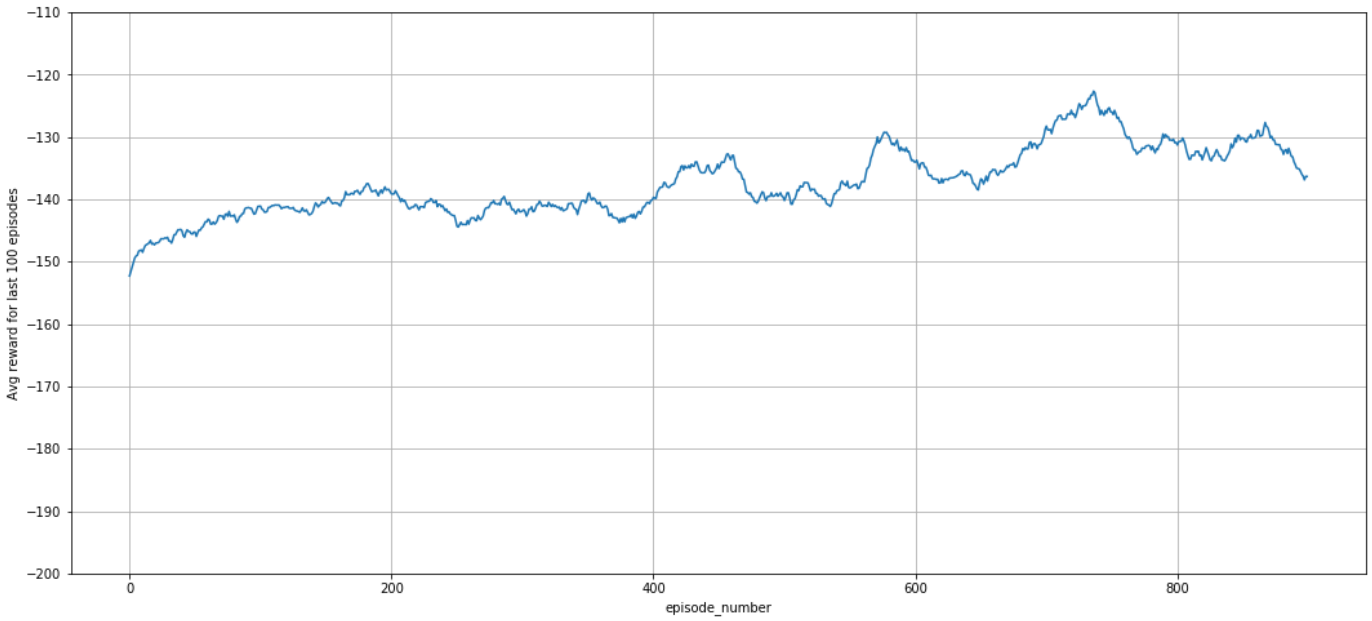
برای نمایش کارایی الگوریتم پیاده شده، مجموع ریوردهای منفی یکی که عامل در مسیر دریافت می کند را رسم می کنیم و مشاهده خواهیم کرد در طول یادگیری مقدار ریواردی که عامل در یک اپیزود ۲۰۰ قدمه جمع آوری می کند از 200- شروع شده و با یادگیری به مقادیر بالاتری خواهد رسید.

میزان مجموع امتیازها در هر اپیزود را از رابطه زیر بدست می آوریم:

$$r_e = \sum_{t=1}^{T_e} r_t \quad sum\ of\ all\ rewards\ recieved\ during\ episode\ e, \quad T_e \leq 200: length\ of\ episode\ e, \ r_t = -1$$

نتیجه:

این الگوریتم در اپیزود پانزدهم همگرا می شود و به طور متوسط با ۱۳۵ قدم به قله می رسد. نمودار زیر از متوسط گیری ۱۰۰ مقدار اخیر به دست آمده است.

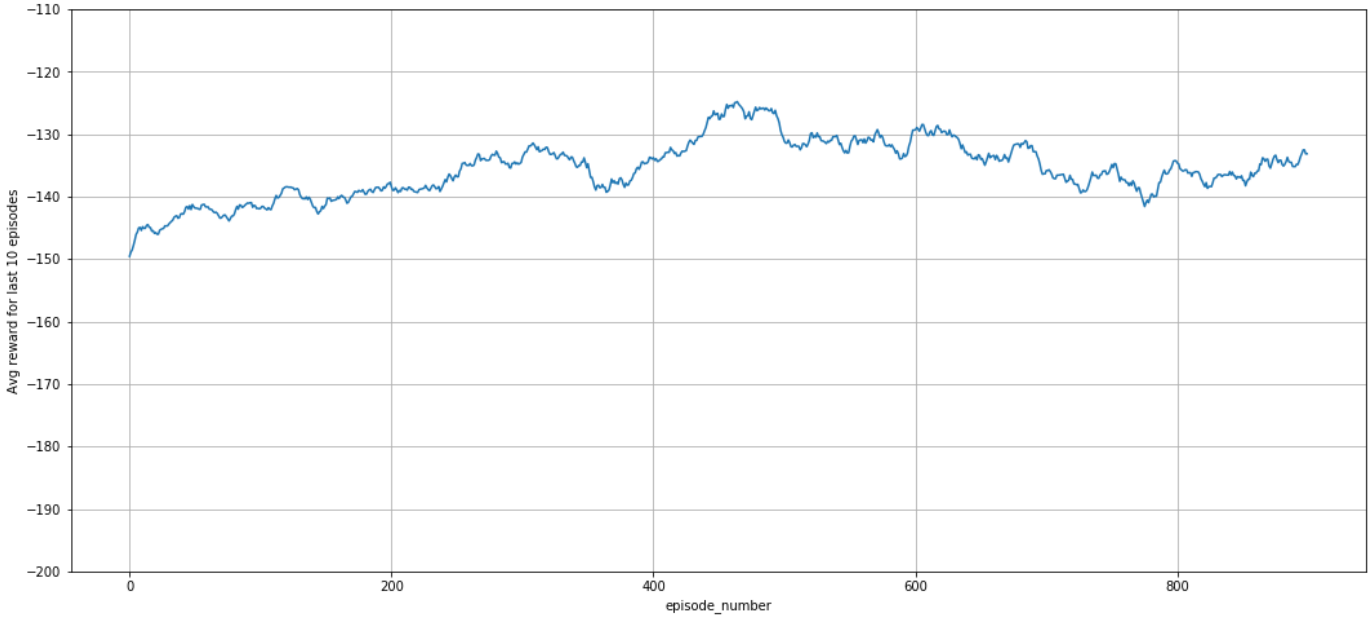


همچنین برای نمونه چند فیلم از اپیزودهای ابتدایی، میانی و پایانی یادگیری در پوشه RBF موجود می باشد.

ترکیب کدینگ RBF و خطی: (کد HO#4_RBFLinear_SARSALambbda.py)

حال یک بعد دیگر به کدینگ حالت-عمل اضافه می کنیم. این بعد جدید شامل یک ترکیب خطی از مکان و سرعت به صورت $y = 0.5x + 10v$ خواهد بود. در نتیجه هم اکنون ۱۶ بعد برای کدینگ خود خواهیم داشت. همه پارامترها مانند قبل تنظیم شده اند.

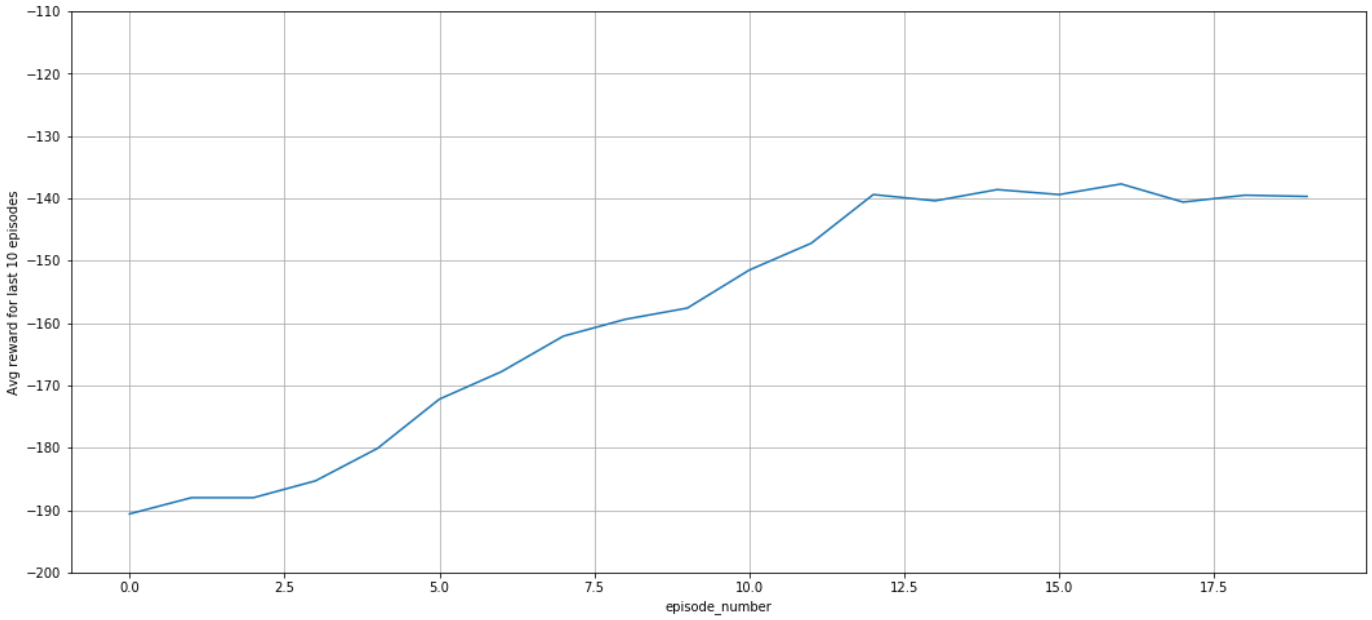
نتیجه:



همچنین برای نمونه چند فیلم از اپیزودهای ابتدایی، میانی و پایانی یادگیری در پوشه RBF_Linear موجود می باشد.

مقایسه دو روش کدینگ: همانطور که از شکل پیداست، کمی میانگین مجموع ریواردهای عامل در طول اپیزود نسبت به زمانی که فقط از توابع RBF استفاده می شد بالاتر آمده که به این معنی است که عامل به طور متوسط زودتر به قله رسیده است و پشیمانی کمتری دارد. این بدان دلیل است که دانش خود از مسئله مبنی بر ارتباط خطی میان مکان و سرعت را وارد کدینگ خود کرده ایم.

در هر دوی حالت فقط RBF و ترکیب RBF و خطی، الگوریتم طی ۱۲ اپیزود به مقدار 140- رسیده است (شکل زیر) و همچنین در زیر ده اپیزود عامل توانسته برای اولین بار به قله برسد.



قسمت ب) در این قسمت سعی می شود به کمک روش Fuzzy مسئله MountainCarContinuous-v0 را یادگیری کنیم. در اینجا علاوه بر فضای حالات، فضای اعمال نیز پیوسته است.

بر اساس مدل Takegi-Sugeno اعمال ابتدایی را 0، 1 و 1- در نظر می گیریم و اعمال نهایی در بازه [1,1-] از ترکیب قوانین fuzzy به دست می آیند. لیبل های مکان را به صورت [1,-0.5,0,0.3-] و لیبل های سرعت را به صورت [0.05,-0.02,0,0.02,0.05-] فرض می کنیم. تابع عضویت به صورت $\exp\left(-(X-C)^2/State\right)$ در نظر گرفته شده است. firing rate بر اساس حداقل مقدار عضویت ابعاد state در نظر گرفته شده است.

الگوریتم استفاده شده برای آپدیت مقادیر، الگوریتم SARSA در نظر گرفته شده است.

ابتدا پارامترهای شبیه سازی را مرور می کنیم.

- نرخ های یادگیری α: مقدار نرخ یادگیری ثابت است و برابر 0.0035 است.
- سیاست نرم: سیاست $\varepsilon - \text{greedy}$ نسبت به مقادیر ارزش action ها در state ها تنظیم می شود. میزان ε در اولین اپیزود برابر 1 است و در هر طول اپیزود ثابت می ماند و با شروع اپیزود بعدی به صورت $\varepsilon \leftarrow \varepsilon * 0.9999$ به روز می شود.
- میزان طول هر اپیزود را آزاد در نظر گرفتیم تا زمانی که به انتها برسد.
- Discount factor γ : مقدار DF برابر $\gamma = 0.9$ در نظر گرفته شده است.
- تعداد اپیزودها: تعداد اپیزودها ۳۰ در نظر گرفته شده است.
- مقدار λ_W و λ_θ: برابر 0.9 فرض شده اند.

برای این قسمت با تغییر مقادیر پارامترهای شبیه سازی جواب مناسبی گرفته نشد. اما کد آن پیاده سازی شده است. HO#4_Fuzzy.py

قسمت ج) در این قسمت سعی می کنیم مسئله MountainCar-v0 را به کمک روش یادگیری Policy Gradient و SARSA(λ) حل کنیم. pseudo code مربوط به این الگوریتم به شکل زیر می باشد.

Actor–Critic with Eligibility Traces (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
  $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

کدینگ تابع ارزش حالت و تابع سیاست:

در این روش بایستی هم تابع ارزش حالت و هم تابع سیاست را کد کرد. تابع ارزش حالت به صورت خطی فرض شده و کدینگ آن همانند قسمت الف) به صورت ترکیب چند RBF و یک تابع خطی از فضا و سرعت فرض می شود و پارامتر آن W است. همچنین برای تابع سیاست کدینگ برای فضای state و action توأماً انجام می پذیرد؛ به نحوی که کدینگ فضا مانند کدینگ تابع ارزش حالت بوده و فقط یک بعد action به آن اضافه می گردد. پارامتر تابع سیاست را با θ نشان می دهیم.

برای تعمیم در توابع ارزش حالت از کدینگ خطی شامل توابع RBF و تابع خطی (ترکیب خطی از مکان و سرعت به صورت $y = 0.5x + 10v$) استفاده می کنیم یعنی تابع ویژگی (feature function) X در الگوریتم بالا متشکل از تعدادی تابع RBF هستند. مراکز این RBFها با مش بندی فضای مکان-سرعت به صورت ۵ در ۳، در ۱۵ نقطه قرار خواهند گرفت. در نتیجه هم اکنون ۱۶ بعد برای کدینگ خود خواهیم داشت.

ماتریس کوواریانس همه آنها نیز به صورت $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ فرض می شود. برای تعمیم در تابع سیاست از کدینگ خطی شامل توابع RBF استفاده می کنیم یعنی تابع ویژگی (feature function) X در الگوریتم بالا متشکل از تعدادی تابع RBF هستند. مراکز این RBFها با مش بندی فضای مکان-سرعت-عمل

به صورت ۵ در ۳ در ۲، در ۳۰ نقطه قرار خواهند گرفت. برای action ها دو نقطه 0.5 و 1.5 فرض شده اند. ماتریس کوواریانس همه آنها نیز به صورت $\Sigma = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ فرض می شود. هر کدام از توابع RBF به صورت زیر تعریف می شود:

$$X = \exp\left(-0.5(s-\mu)^T\Sigma^{-1}(s-\mu)\right)$$

در الگوریتم بالا میزان وزن هر کدام از توابع RBF یعنی بردار W و θ و بردار eligibility trace هر کدام یعنی $Z - W$ و Z_θ به ازای در هر قدم به روز رسانی می شوند.

ابتدا پارامترهای شبیه سازی را مرور می کنیم.

- نرخ های یادگیری α_W و α_θ: مقدار نرخ یادگیری در طول اپیزود ثابت است و برای همه actionها میزان α ها در اولین اپیزود برابر 0.01 است و با شروع اپیزود بعدی به صورت $\alpha \leftarrow \alpha * 0.999$ به روز می شود.
- میزان طول هر اپیزود را آزاد در نظر گرفتیم تا زمانی که به انتها برسد.
- Discount factor γ : مقدار DF برابر $\gamma = 0.98$ در نظر گرفته شده است.
- تعداد اپیزودها: تعداد اپیزودها ۳۰ در نظر گرفته شده است.
- مقدار λ_W و λ_θ: برابر 0.9 فرض شده اند.

میزان مجموع امتیازها در هر اپیزود را از رابطه زیر بدست می آوریم:

برای نمایش کارایی الگوریتم پیاده شده، مجموع ریوردهای منفی یکی که عامل در مسیر دریافت می کند را رسم می کنیم و مشاهده خواهیم کرد در طول یادگیری مقدار ریواردی که عامل در یک اپیزود ۲۰۰ قدمه جمع آوری می کند از 200- شروع شده و با یادگیری به مقادیر بالاتری خواهد رسید.

$$r_e = \sum_{t=1}^{T_e} r_t \quad sum\ of\ all\ rewards\ recieved\ during\ episode\ e, \quad T_e \leq 200: length\ of\ episode\ e,\ r_t = -1$$

نتیجه:

این الگوریتم در اپیزود پانزدهم همگرا می شود و به طور متوسط با ۱۳۵ قدم به قله می رسد. نمودار زیر از متوسط گیری ۱۰۰ مقدار اخیر به دست آمده است.

برای این قسمت نیز پاسخ مناسبی گرفته نشد اما کد آن پیاده سازی شد. HW#4_PolicyGradient.py