مدرس: دکتر فاطمه منصوری
تهیه کننده : فاطمه حسین زاده

هوش مصنوعی

جزوه ششم

Expectimax

اکنون دیدیم که مینی‌ماکس چگونه کار می‌کند و چگونه اجرای کامل مینی‌ماکس به ما اجازه می‌دهد تا در برابر حریف بهینه پاسخی بهینه نشان دهیم. با این حال، این الگوریتم محدودیت‌هایی هم دارد.

با اینکه مینی‌ماکس به یک حریف بهینه پاسخ می‌دهد، اغلب در شرایطی که برای یک عمل پاسخ بهینه‌ای تعریف نشده باشد، بیش از حد بدبین است. چنین موقعیت‌هایی شامل سناریوهایی با تصادفی ذاتی می‌شود؛ مانند بازی‌های ورق یا تاس یا حریفان غیرقابل پیش‌بینی که به‌طور تصادفی یا غیربهینه حرکت می‌کنند. هنگامی که در مورد فرآیندهای تصمیم‌گیری مارکوف بحث کنیم، این سناریوها را با جزئیات بیشتر بررسی خواهیم کرد.

این تصادفی بودن را می توان از طریق تعمیم مینی‌ماکس به Expectimax نشان داد. Expectimax گره‌های شانس را به درخت بازی معرفی می‌کند، که به جای در نظر گرفتن بدترین سناریو مانند گره‌های کمینه، حالت متوسط را در نظر می‌گیرد. به طور خاص، در حالی که کمینه‌سازها به سادگی حداقل سودمندی را برای فرزندانشان محاسبه می‌کنند، گره‌های شانس سود مورد انتظار یا مقدار مورد انتظار را محاسبه می‌کنند. قانون ما برای تعیین مقادیر گره‌ها با expectimax به شرح زیر است:

$$\forall\ agent-controlled\ states, V(s) = max_{s' \in successors(s)}\ V(s')$$

$$\forall\ chance\ state, V(s) = \sum_{s' \in successors(s)} p(s'|s)V(s')$$

$$\forall\ terminal\ state, V(s) = known$$

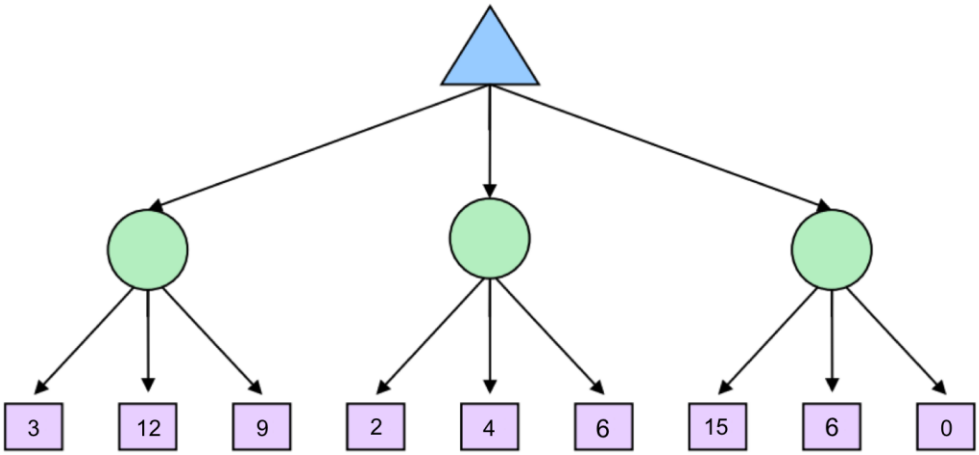
شبه کد expectimax کاملاً شبیه minimax است، اما چون ما گره‌های شانس را جایگزین گره‌های کمینه‌ساز می‌کنیم، به جای حداقل سودمندی سود مورد انتظار محاسبه می‌شود:

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the agent is MAX: return max-value(state)
    if the agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

درخت expectimax زیر را در نظر بگیرید، که در آن گره‌های شانس با گره‌های دایره‌ای به جای مثلث‌های رو به بالا/پایین برای ماکزیمایزر/مینی‌مایزر نمایش داده می‌شوند.

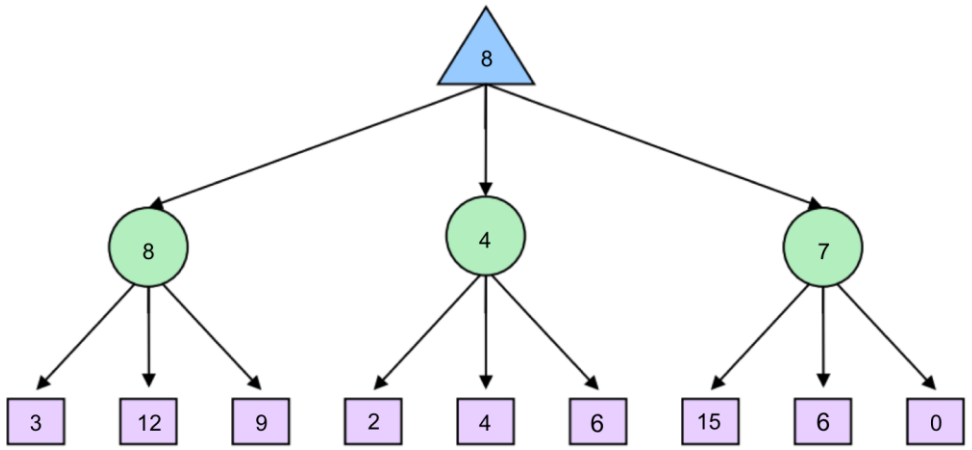


فرض کنید که همه فرزندان گره‌های شانس احتمال وقوع $\frac{۱}{۳}$ را دارند. از این رو، با استفاده از فرمول expectimax از چپ به راست ۳ گره شانس مقادیر

$$\frac{۱}{۳} \times ۳ + \frac{۱}{۳} \times ۱۲ + \frac{۱}{۳} \times ۹ = ۸, \frac{۱}{۳} \times ۲ + \frac{۱}{۳} \times ۴ + \frac{۱}{۳} \times ۶ = ۴ \text{ و } \frac{۱}{۳} \times ۱۵ + \frac{۱}{۳} \times ۶ + \frac{۱}{۳} \times ۰ = ۷$$

را می گیرند.

و در آخر مقدار بیشینه برابر ۸ را انتخاب می‌کند و یک درخت بازی پر شده را به صورت زیر به دست می‌آورد:



به عنوان آخرین نکته در مورد expectimax، مهم است که بدانیم، به طور کلی، لازم است به همه گره‌های شانس نگاه کنیم – نمی‌توانیم هرس کنیم– بر خلاف محاسبه مینیمم‌ها یا ماکزیمم‌ها در minimax، یک مقدار می تواند مقدار مورد انتظار محاسبه شده توسط expectimax را به طور دلخواه زیاد یا کم کند. با این حال، هرس زمانی می تواند امکان پذیر باشد که محدودیت‌های مقادیر احتمالی گره بدانیم.

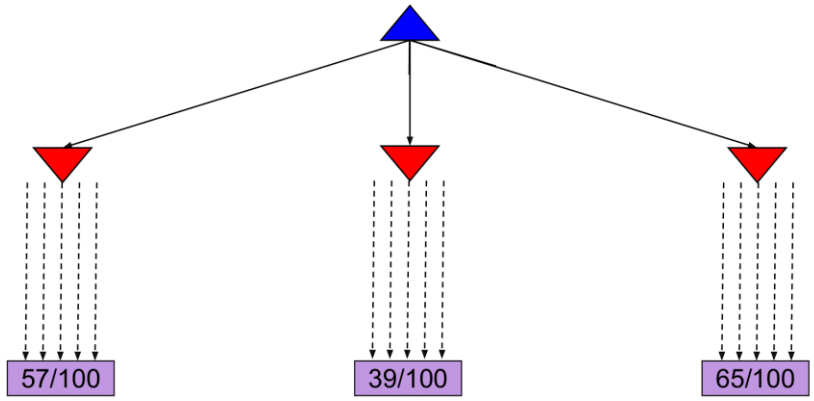
درخت جستجو مونت کارلو

برای برنامه هایی با ضریب انشعاب زیاد، مانند بازی Go، دیگر نمی‌توان از minimax استفاده کرد. برای چنین برنامه‌هایی از الگوریتم جستجوی درخت مونت کارلو استفاده می‌کنیم. MCTS[1] بر دو ایده استوار است:

- ارزیابی بر اساس عرضه: از حالت s بارها با استفاده از یک خط مشی (مثلاً تصادفی) بازی می‌کنند و برد/باخت‌ها را شمارش می‌کنند.
- جستجوی انتخابی: جستجو بخش‌هایی از درخت، بدون محدودیت، که تصمیم‌گیری را در ریشه بهبود می‌بخشد.

در مثال Go، از یک وضعیت معین، چندین بار طبق یک خط مشی تا پایان بازی بازی می کنیم.

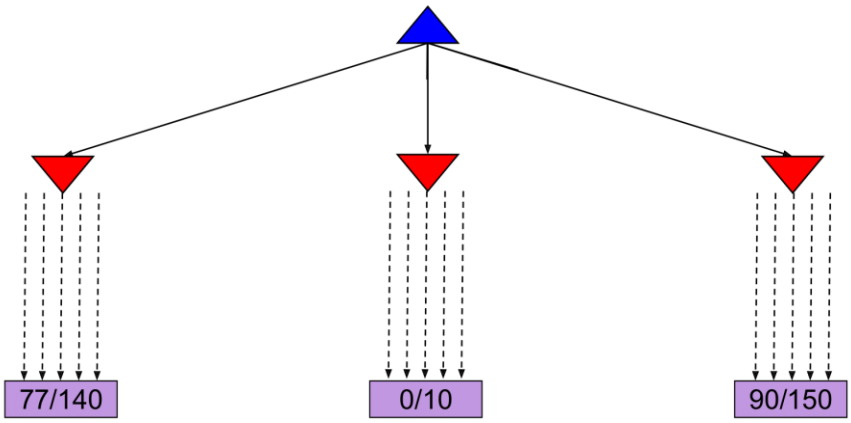
[1] Monte Carlo Tree Search



در حالت فعلی ما سه عمل مختلف داریم (چپ، وسط و راست). هر اقدام را ۱۰۰ بار انجام می‌دهیم و درصد برد را برای هر یک ثبت می‌کنیم. پس از شبیه‌سازی‌ها، ما نسبتاً مطمئن هستیم که بهترین عمل کدام است.

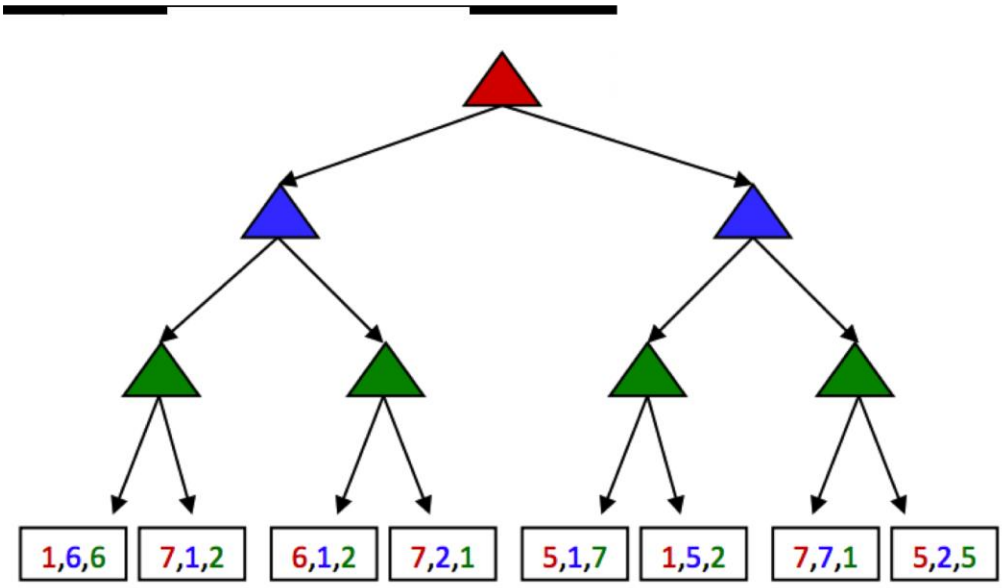
اما در مثال بعد، ما همان مقدار شبیه‌سازی را به هر عمل جایگزین اختصاص دادیم. با این حال، ممکن است پس از چند شبیه‌سازی مشخص شود که یک عمل خاص، بردهای زیادی را بر نمی‌گرداند و بنابراین ممکن است این تلاش محاسباتی را در انجام شبیه‌سازی‌های بیشتر برای سایر اقدامات اختصاص دهیم.

در این مثال تصمیم گرفتیم ۹۰ شبیه سازی باقی مانده را از گره میانی به گره‌های چپ و راست اختصاص دهیم.



بازی‌ها

همه بازی‌ها zero-sum نیستند. در واقع، عامل‌های مختلف ممکن است وظایف مجزایی در یک بازی داشته باشند که مستقیماً مستلزم رقابت شدید با یکدیگر نباشد. چنین بازی‌هایی را می‌توان با درختانی تنظیم کرد که با سودمندی چند عامل مشخص می‌شوند. چنین سودمندی‌ای، به جای اینکه یک مقدار واحد داشته باشد که عامل‌ها سعی کنند آن را به حداقل یا حداکثر برسانند، به صورت چندتایی با مقادیر مختلف در داخل یک تاپل نشان داده می‌شود. سپس هر عامل تلاش می‌کند تا سود خود را در هر گره‌ای که کنترل می‌کند، بدون توجه به ابزارهای دیگر عوامل به حداکثر برساند. درخت زیر را در نظر بگیرید:



گره‌های قرمز، سبز و آبی مربوط به سه عامل مجزا هستند که به ترتیب سودمندی گره‌های قرمز، سبز و آبی را از گزینه‌های ممکن در هر لایه به حداکثر می‌رسانند. در مثال بالا ابتدا باید مقادیر سبز باهم مقایسه شوند، پس بین تاپل‌های (۱,۶,۶) و (۷,۱,۲) فقط ۶ و ۲ باهم مقایسه می‌شوند و چون درخت مقدار بیشینه را می‌خواهد پس (۱,۶,۶) انتخاب می‌شود. اگر به همین ترتیب ادامه دهیم در نهایت تاپل (۵,۲,۵) در بالای درخت انتخاب می‌شود.