神経情報システム論

第11回　強化学習 II

今回の予定

- Model freeの強化学習システム
- Monte Carlo 法
- 探索のためのアルゴリズム
- TD学習
- eligibility trace とTD(λ)学習
- 価値関数の表現方法

復習：行動決定と価値関数

- 強化学習の目的
  - 受け取る報酬和を最大化する行動決定則を求める．
- 価値関数
  - 報酬和の期待値を表す個体内部の量．
  - 価値関数が最大になる行動を選択する（貪欲法）．
- 価値関数をいかにして求めるか？
  - 環境モデルがある場合 (model-based)：DP法
  - 環境モデルがない場合 (model-free)：

Model-free の強化学習システム

- 環境に対する知識を仮定しない．
- モデルを用いずに価値関数を直接学習する．
  - 環境のモデルを作れない場合に有用．
- action-value function $Q(s, a)$ を使って行動 a を定める．
  - モデルがないので，V からQを求めることは不可．
  - Q を学習して， Qが最大の行動を選択する．

モンテカルロ法 (Monte Carlo method)

- episode中に経験する各状態 s に着目する．
- その状態から始めてepisodeが終わるまでの報酬和を状態ごとに求める．
- 同じ状態を2度経験する場合は，初めてその状態に達したときに着目する．
- 多数のepisodeを繰り返して報酬和の平均値を求める．

アルゴリズム

1. 適当なpolicy $\pi(s, a)$を定める．
2. πを使って行動を決めてepisodeを一回経験する．
3. そのepisodeで経験したすべての状態 s について，始めて状態に到達した時点からの報酬和 R を求める．
4. 得られた R の値を既存の $R\mathrm{old}$ と平均して，新しい $R\mathrm{new}$を求め，$Q(s, a)$を更新する．
5. 4.での変化量が一定値以下なら終わり．
6. $Q(s, a)$を使ってpolicy $\pi(s, a)$を更新し，2へもどる．

例題：Grid World

- スタートからゴールまでできるだけ早く到達する．
- 状態 s：格子上のどこにいるかが状態としてわかる．
- 行動 a：格子の上下左右に動く．
- 報酬 r：各ステップで -1 の報酬（+1の罰）．
  - 割引率 γ が 1であれば，報酬和の絶対値はゴール到達に要したステップ数に等しい．

興味がある人は実装してみよ．

価値関数学習のための行動選択

- すべての状態，行動の組について価値関数の学習が一様に進むように工夫する必要がある．
- exploring starts：総あたり，しらみつぶし
  - episodeの初期状態としてすべての組を設定．
- on-policy Monte Carlo：ランダム性のある行動選択
  - 一定の割合で行動をランダムに決定．
    1. ε -greedy法：ε の割合だけランダムに決定．
    2. Boltzmann法(softmax法）：
       価値関数の大きさに応じて選択確率を決定．

ε-greedy法とBoltzmann選択法

- greedy法（貪欲法）
  - 常に，価値関数が最大な行動を選択する．
  - 「あそび」がなく，新しいことが学べない．
- ε-greedy法
  - 価値関数最大の行動を選ぶ割合を100%にしない．
  - 一定の割合 (ε) でランダムに行動を選ぶ．
    ランダムに選んだ行動により新しい知識を得る．
  - ε の値に応じて行動のランダム性が変化する．

Boltzmann選択法

- 以下の確率に従って行動を選ぶ．

$$P(a; s) = \frac{\exp\left(Q(s, a) / T\right)}{\Sigma_{a'} \exp\left(Q(s, a') / T\right)}$$

→ $Q(s, a)$が大きな a を選ぶ確率が高くなる．

- ランダム性は T によって調整する．
  - 温度パラメタ(temperature parameter)と呼ばれる．
  - 温度が高いほどランダム性が高い．

TD学習 (Temporal Difference Learning)

- 1990年代に注目されたアルゴリズム
  - 時間的に局所的なアルゴリズム
  - 脳のモデルとしての期待
- Dynamic programmingと Monte Carlo法の中間的方法
  - model-free RL
    環境のモデルをもたず，価値関数を直接学習する．
  - 一度の状態遷移に着目する．
    Episode終了まで待たずに，一度の行動での経験に基づいて価値関数を修正する．

価値関数の修正法

- 価値関数の更新規則

$$\Delta V(s_t) = \alpha \left(R_t - V(s_t) \right)$$

$$= \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right)$$

観測した報酬　次の状態の価値関数

- 学習性能
  - 収束性が保証されている．($\alpha \to 0$の極限で)
  - 一般に，モンテカルロ法に比べてよい成績を示す．

TD学習のアルゴリズム 1

- Sarsa: On-line TD learning
  - 最も自然なアルゴリズム．

$$\Delta Q(s_t, a_t) = \alpha \left(R_t - Q(s_t, a_t) \right)$$

$$= \alpha \{ r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \}$$

報酬　次の行動の価値関数

- 次の時刻で実際に選んだ行動に対するQ値を用いる．
- 乱数で選んだ価値関数が影響を及ぼすことになる．

TD学習のアルゴリズム 2

- Q-Learning: Off-line TD learning (Watkins, 1989)
  - 現代最もよく使われている方法

$$\Delta Q(s_t, a_t) =$$

$$\alpha \{ r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \}$$

報酬　次の状態での最大価値関数

- 次の状態で最大のQ値を与える行動を用いる．
- 数学的に解析しやすいが，オンラインでの性能はSarsaに劣ることがある．

TD学習のアルゴリズム 3

- Actor-Critic model：
  - 脳の学習モデルとしてよく使われる方法
- 行動選択系(actor)と，価値関数生成系 (critic)を明確に分離した構造をもつ．
  - critic：価値関数の誤差（TD誤差）を計算する．
    TD誤差に基づいて価値関数を更新．
  - actor：TD誤差に基づいて行動決定ルールを修正．
- state-value function $V(s)$を用いる．

TD誤差とCriticの動作

- TD 誤差 (TD error):

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

- criticは，TD errorを用いて状態価値関数を更新する．

$$\Delta V(s_t) = \alpha \delta_t$$

Actorの動作

- actorは，関数 $q(s, a)$を用いて，Boltzmann選択法により行動を選択する．

$$P(a; s) = \exp(q(s, a) / T) \ / \Sigma_{a'} \exp(q(s, a') / T)$$

  - $q(s, a)$ は行動価値関数ではないことに注意．

- TD errorを用いて$q(s, a)$を更新し，policyを変更する．
  - TD errorが正 → 行動a_t を選択したのはよかった．
    負 → 行動a_t を選択したのは誤り．

$$\Delta q(s_t, a_t) = \beta \delta_t \text{ または，} \beta \delta_t (1 - \pi_{\text{old}}(s_t, a_t))$$

Monte Carlo法とTD学習を結ぶ考え方

- Monte Carlo法：
  - episode全体の状態遷移を使って学習する．
  - 遠い将来に得られる報酬が反映される．
- TD learning:
  - 一回の状態遷移だけを使って学習する．
  - 1時刻先に得られる報酬だけを使う．
- 両者の違いはどこにあるか？
  - 何時刻先まで考慮して報酬期待値を計算するか？

将来の報酬の計算方法

- n ステップ先までの報酬を使った報酬和：

$$R_t(n) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots + \gamma^n V(s_{t+n})$$

- Monte Carlo とTD 学習はそれぞれ R(1), R(∞)を教師信号に使って学習を行なう．

$$R_t(1) = r_{t+1} + \gamma V(s_{t+1})$$

TD(λ)学習 1

- 種々の n に関する$R_t(n)$の効果を一緒に考える．
  - $R_t(n)$ の重みを λ^{n-1} として$R_t(n)$ の荷重和を考える．

$$R_t^{\lambda} = (1-\lambda)\{ R_t(1) + \lambda R_t(2) + \lambda^2 R_t(3) + \ldots + \lambda^{n-1} R_t(n) + \ldots \}$$

- この式を展開してみると中身が理解できる.

TD(λ)学習 2

- R_t^λ を報酬と考えて，TD学習を行なう．
- したがって，TD誤差は，

$$\delta_t = R_t^\lambda - V_{\text{old}}(s_t)$$

あるいは，

$$\delta_t = R_t^\lambda - Q_{\text{old}}(s_t, a_t)$$

となるので，これに従って学習を行なう．

TD(λ)の実装方法 1

- ある時刻で学習を行なう際に，その時刻だけでなく，過去に経験した状態に関する価値関数も修正する．
- Sarsaの場合を例にとると，以下のようになる．

1. 時刻 t での状態と行動を記録する（履歴を残す）．

$$e(s_t, a_t) := e(s_t, a_t) + 1$$

その状態を経由したことを示す証拠を残す．

$e(s_t, a_t)$ を eligibility trace と呼ぶ．

TD(λ)の実装方法 2

2. 履歴情報を使って価値関数を修正する．

- すべての状態・行動組に対して次の更新を行なう．

$$Q(s, a) := Q(s, a) + \alpha\, \delta\, e(s, a)$$

3. 履歴情報を減衰させる．

- 現在時刻より離れた時刻の重みを減少させる．

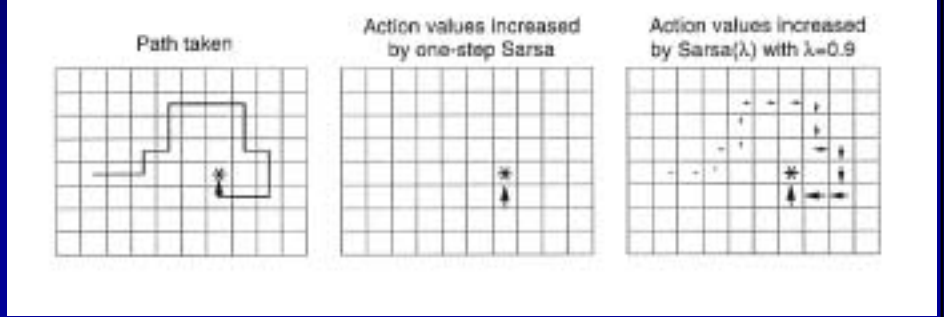
$$e(s, a) := \gamma\, \lambda\, e(s, a)$$

γ : 割引率

λ :過去の重みづけの大きさ

TD(λ)の有効性の例

- 通常のSarsa では，1ステップ分のみ学習する．
- eligibility traceを考慮すると，経路上の全ステップに対して学習が進む．



価値関数の実装の問題

- 状態や行動の数が多い場合，$Q(s, a)$ や $V(s)$ を表形式で保持するのは非現実的である．

  例：ロボットの制御（状態が連続値をとる）．

- 価値関数の表現するため，一般的な関数近似の手法を用いる．
  - 線形近似，多層パーセプトロン，RBF, …
  - この問題は他の問題と共通する大きな問題．