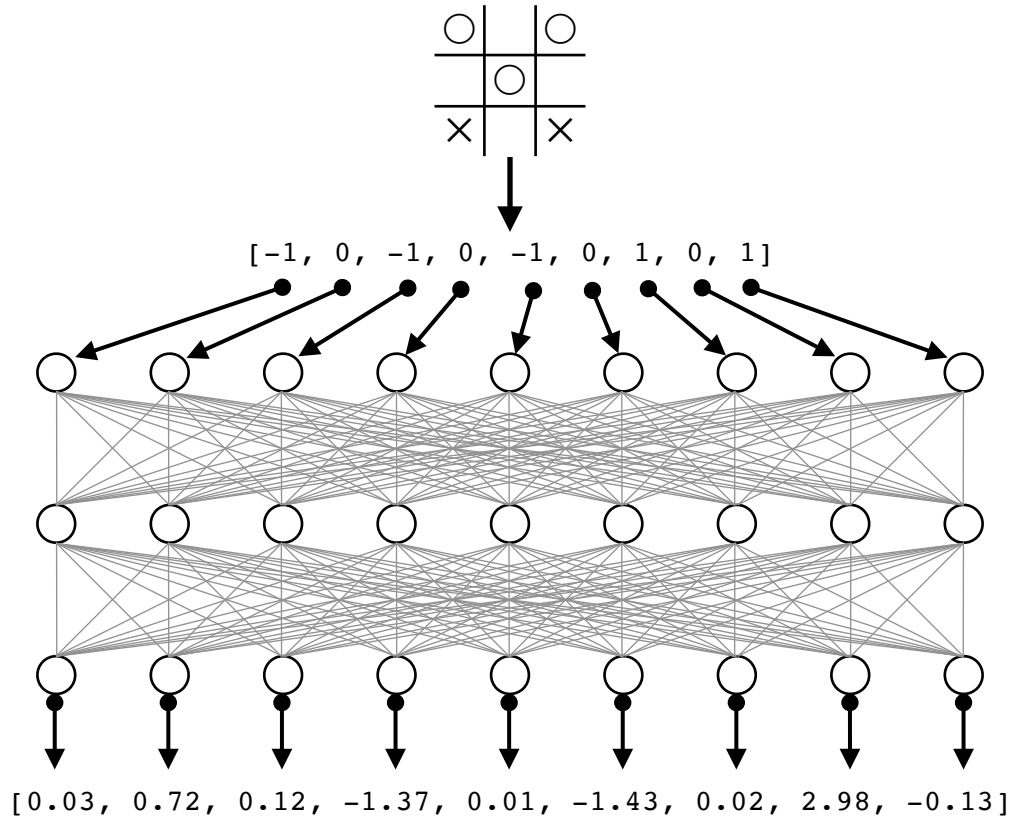
Deep Q-Network

Python 中級
鈴木 敬彦

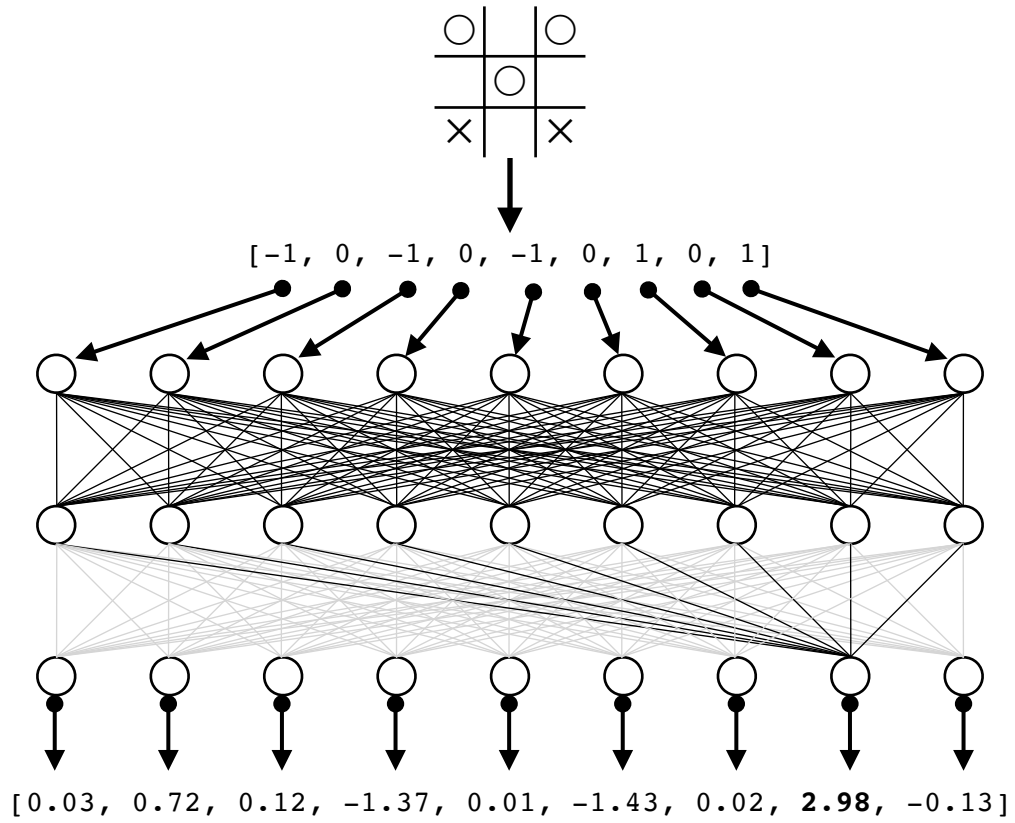
Q学習では、状態の数と行動の数が多くなると必要とする記憶容量が爆発的に増えてしまいます。そこで、ニューラルネットワークを用いて Q関数の近似値を得ようとするのが Qネットワークです。DNN の利用と共に知られたこともあり、一般に Deep Q-Network（DQN、深層 Qネットワーク）と呼ばれています。また、DQN の最適化を Deep Q-Learning（深層 Q学習）と呼びます。

DQN は、全接続型のニューラルネットワークを使用して、状態を入力とし、各行動に対する Q 値を出力します。例えば、三目並べであれば、入力層のノード数は 9、出力層のノード数も 9 となります。一回の順伝播の計算で、入力された状態に対する全ての行動の Q値を得られます。



Deep Q-Learning

損失やバックプロパゲーションに関しては、通常のニューラルネットワークと何ら変わりはありません。ただし、「正解」とされる値が若干異なってきます。なぜならば、DQN　は選択すべき行動を出力するものではなく、ある状態における全ての行動に対する Q値を出力するものだからです。Q学習エージェントが選択すべき行動は、最も大きな Q値を取る行動でした。そして、Q値の更新は、訓練データに基づいてその行動に対する Q値でした。DQN　においても、その点に変わりはなく、DQN　の最適化において個別の訓練データに基づいて最適化されるべきは、その状態に対する全ての Q値ではなく、その訓練データで選択されている行動に対する Q値となります。



Q テーブルの更新に用いたベルマン方程式を思い出してください。

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

DQN の最適化も、このベルマン方程式に基づいて行われます。以下は「Human-level control through deep reinforcement learning」[1] という深層学習ブームの発端ともなった論文に掲載されている式 (pp.6-7) です。

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') \,|\, s, a\right] \tag{1}$$

$$y = r + \gamma \max_{a'} Q^*(s', a'; \theta_i^-) \tag{2}$$

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}\left[(\mathbb{E}_{s'}[y \,|\, s, a] - Q(s, a; \theta_i))^2\right] \tag{3.1}$$

$$= \mathbb{E}_{s,a,r,s'}\left[(y - Q(s, a; \theta_i))^2\right] + \mathbb{E}_{s,a,r}[\mathbb{V}_{s'}[y]] \tag{3.2}$$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right] \tag{4}$$

[1] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).

式 (1) を見ると、Q テーブルの更新に用いたベルマン方程式第 2 項の括弧の中と同じであることがわかります。$Q^*(s,a)$ は、ここでは目標値であり、状態 s と行動 a の関数で、$\mathbb{E}_{s'}\left[\circ\circ \mid s,a\right]$ は状態 s で行動 a を選択して遷移した状態 s' での〇〇の期待値を意味します。r と γ は今までと同じく報酬と割引率です。$\max_{a'}Q^*(s',a')$ は、状態 s' で得られる最大の Q 値を意味します。（より正確には、状態 s' で得られる Q 値の中から最大の Q 値を取る行動 a' を選択した時の Q 値。）他の式も含めて期待値が出てきていますが、これは、ニューラルネットワークが正しく限界まで最適化された時に得られるであろう値、という程度の意味しか持ちません。実際の値は不明であるため、DNN の最適化の説明にも書いた通り、最適化実行時点でのニューラルネットワークの順伝播から得た値となります。

式 (2) は、式 (3.1) を簡潔に記述すべく直前の文中に出てくる補足で、式 (1) の括弧の中の式を y として再定義しています。セミコロン (;) の後の θ は Q 値を計算するニューラルネットワークのパラメーターを表しています。ここで θ が明示されていること、θ^- となっていることには意味があるのですが、この点については後述します。最終的には、この y が目標値となります。

式 (3) は損失関数です。目標となる Q 値と現在の Q 値の二乗誤差の期待値です。2 行目は y で書き換えたもので、第 2 項は y の分散の期待値で、現在のニューラルネットワークパラメーター θ に依存しないため、勾配の計算では無視されます。（式の変形については、バイアス-バリアンス分解を参考にしてください。）

式 (4) が最適化に必要となる損失の勾配です。あるべきはずの 2 が消えていますが、これは恐らく、学習率に吸収されてしまうのでなくても構わない…、といった意味合いだと推測されます。

このように数式から見てもわかるように、DQN と DNN は、順伝播出力から損失を計算する過程以外に違いはありません。DQN では、ニューラルネットワークの順伝播の出力全てから損失を計算するのではなく、「行動によって遷移した状態から得られるニューラルネットワークの順伝播の出力の最大値（$\max_{a'}Q^*(s',a')$）」と「ニューラルネットワークの順伝播の出力の中の選択された行動に対する値（$Q(s,a)$）」から損失を計算します。

最適化手法

DQN の最適化には、数多ある DNN の最適化の手法や強化学習の手法を利用することができます。ここでは、前出の論文で発表された、或いは利用されている手法のいくつかを紹介します。

焼き鈍し法 / Simulated Annealing

学習の進度に合わせて学習率等のハイパーパラメーターを調整する手法です。
例えば、前出の論文では ε-greedy 法を用いて、全学習期間の最初の 10% の間に ε を 1.0 から 0.1 に線形に変化させています。この様にすることで、学習の初期段階では行動価値の探索に重点を置き、その後は予測に重点を置いた学習が可能となります。

デルタクリッピング

損失が大きければ、その勾配も大きくなり、ネットワークパラメーターや損失の勾配が発散する可能性が高くなります。発散を抑えるため、損失を算出する際の差分（先ほどの例であれば式(3.2)の$y - Q(s,a;\theta_i)$）を所定の範囲（-1.0〜1.0 等）に制限します。

状態、報酬の正規化状態、報酬の正規化

状態や報酬の表現には様々な方法が考えられますが、これらを正規化することで学習が安定し、収束しやすくなります。例えば、報酬を -1, 0, 1 のみに固定したり、状態を 0, 1, 2 ではなく -1, 0, 1 で表現します。

ターゲットネットワーク

前出の論文で提案された手法で特に名称はありません（Double Q-Network、Double Deep Q-Network と呼ばれることも）。この手法では、予測するネットワークパラメーターと更新するネットワークパラメーターを分離させます。具体的には、各更新の前にネットワークパラメーターを複製（ターゲットネットワークと呼ぶ）し、目標値の計算にはこれを使用します。この様にすることで、訓練データ収集時と更新時の順伝播出力の差がなくなり、学習が安定します。式 (2) のθ^-は、凍結されたネットワークパラメーターであることを意味しています。

経験再生 / Experience Replay[2]

この手法では、環境から得た経験$e_t = (s, a, r, s')$を保存（$D_N = \{e_1, e_2, \cdots, e_N\}$、Replay Memory、Replay Buffer、再生メモリー等と呼ばれ、サーキュラーバッファーで実装されることが多い）して、最適化時には直近のN個の経験サンプルからランダムに選択したミニバッチを使用します。この様にすることで、環境から得た経験を効率的に利用することができ、学習の効率も向上し、安定するとのことです。経験サンプルからランダムに選択することは、経験サンプルの相関性を破壊するためです。訓練データの相関性は、学習を妨げることはあっても、有益であることはないと言われています。経験再生だけでも、次に紹介する優先度付き経験再生を含め、様々な亜種があり、経験再生が実際にどの様に学習に影響を与えているのかは現在進行形で研究されています。興味のある方には「Revisiting Fundamentals of Experience Replay」[3]が役に立つかもしれません。

優先度付き経験再生 / Prioritized Experience Replay

前述の経験再生の亜種で、蓄積された経験サンプルのうち、損失誤差の大きい経験サンプルから優先的に学習させる手法です。

---

[2] Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3-4):293—321, 1992.

[3] Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y.,Larochelle, H., Rowland, M., and Dabney, W. Revisiting Fundamentals of Experience Replay, International Conference on Machine Learning, pp. 3061—3071. PMLR, 2020

付録: 期待値と分散

期待値 (expected value) とは、確率変数を含む関数を多数回観測した時の値（実現値）の平均値のことです。これは、各実現値に各値の確率の重みを付けた加重平均と等価で、一般的に $\mathbb{E}[X]$ と表記されます。（ここで X は、確率変数です。）

例えば、1 から 6 までの値が等しい確率で出るサイコロがあったとすれば、サイコロが関数そのもので、多数回の試行で得られた数の平均が期待値に相当します。

$$\begin{aligned}\mathbb{E}[X] &= 1\times\frac{1}{6}+2\times\frac{1}{6}+3\times\frac{1}{6}+4\times\frac{1}{6}+5\times\frac{1}{6}+6\times\frac{1}{6}=\frac{1+2+3+4+5+6}{6}\\ &=\frac{21}{6}=3.5\end{aligned}$$

分散 (variance) は、値の散らばり具合を表し、一般的に $\mathbb{V}(X)$ と表記されます。分散は平均値からの偏差の2乗平均で、期待値を用いて以下のように表されます。

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}\left[(X-\mathbb{E}[X])^2\right]\\ &= \mathbb{E}[X^2]-(\mathbb{E}[X])^2\end{aligned}$$

先ほどと同様にサイコロの例であれば、以下のように計算できます。

$$\begin{aligned}\mathbb{V}[X] &= \frac{1}{6}\left[(1-3.5)^2+(2-3.5)^2+(3-3.5)^2+(4-3.5)^2+(5-3.5)^2+(6-3.5)^2\right]\\ &=\frac{1}{6}\left[6.25+2.25+0.25+0.25+2.25+6.25\right]=\frac{35}{12}=2.91666\dot{}\end{aligned}$$

展開された 2 行目の式であれば、より単純に計算可能でしょう。

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}[X^2]-(\mathbb{E}[X])^2\\ &=\frac{1+4+9+16+25+36}{6}-3.5^2\\ &=\frac{91}{6}-\frac{49}{4}=\frac{182-147}{12}=\frac{35}{12}=2.91666\dot{}\end{aligned}$$