現場で使えるディープラーニング基礎講座

DAY8

SkillUP AI

前回の復習

- 前回は何を学びましたか？

最終発表

最終発表

- 課題の成果を発表して頂きます。
- 1人当たりの時間は、発表3分+質疑応答2分=5分とします。
- 発表内容に入れるもの
    1. 実装の進捗状況
    2. モデルの改良点と識別精度の変遷
    3. 学習用データでの識別精度とテスト用データでの識別精度の比較
    4. Arxiv.orgなどで見つけた論文とそこから得られた知見
    5. 課題を通しての感想
- 事前に、課題をまとめたNotebookを講師にDMで渡してください。
- 発表時は、そのNotebookをプロジェクターに投影するようにします。時間が限られていますので、原則、個人PCでの発表は受け付けないようにしたいと思います。

強化学習

目次

強化学習の基礎1

強化学習とは

- 強化学習とは、試行錯誤を通じて「価値を最大化するような行動」を学習する手法。
- あらかじめ正しい答えが分かっていない（教師データが存在しないまたは取得が難しい）ような問いに対し、より良い振る舞いをさせることが得意。
- 対戦ゲームやロボットなどでの応用例が多いが、産業界での活用事例はまだ少ない。
- 強化学習の産業活用事例
  - Googleがデータセンター冷却電力を40%削減、DeepMindのAIを活用
    - https://tech.nikkeibp.co.jp/it/atcl/news/16/072102162/
  - 深層強化学習で超高層ビルの地震に備える
    - https://inforium.nttdata.com/foresight/ai-vibration-control.html
  - 自律分散型スマートグリッド上の電力取引に対する自然方策勾配法によるマルチエージェント強化学習の有効性検証
    - http://tanichu.com/wp-content/themes/tanichu/data/pdf/bunsan10.pdf

強化学習に関する参考書籍

- 『強化学習(和訳版)』(Richard S. Sutton and Andrew G. Barto、森北出版)
- 『Reinforcement Learning second edition』(Richard S. Sutton and Andrew G. Barto、The MIT press)
- 『これからの強化学習』(牧野など、森北出版)
- 『Pythonで学ぶ強化学習』(久保隆宏、講談社)
  - Githubで公開されている実装コードが非常に参考になる。
    - https://github.com/icoxfog417/baby-steps-of-rl-ja

強化学習の学習

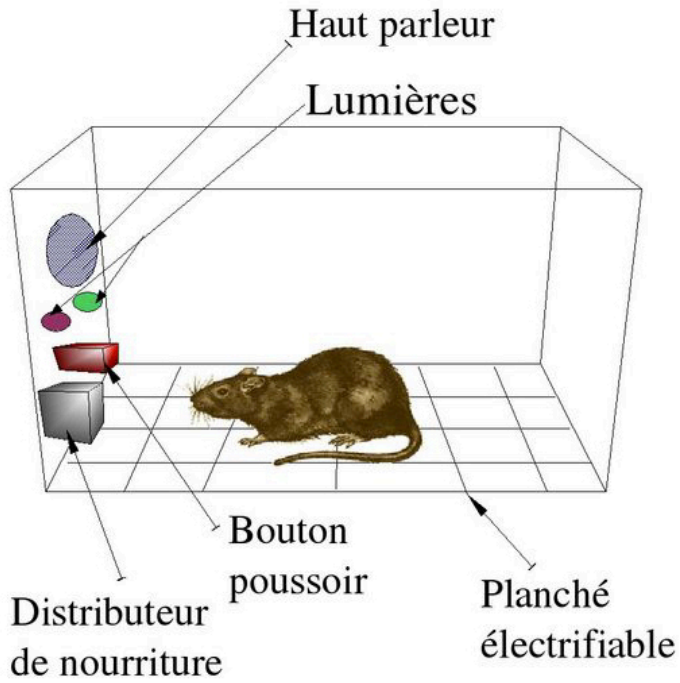
- 基本的には、教師を用意せずに、試行錯誤で学習を行う。
- 教師データを必要としない代わりに、システムを強化学習プログラムに従って動作させ、望ましい結果が得られた際には、報酬と呼ばれる信号を与える。
- 例えば、ロボットの歩行制御では歩けた距離を報酬としたり、囲碁の対戦プログラムでは勝つことによって報酬をもらえるように設計する。
- もし仮に、教師あり学習でロボットの歩行制御を学習させたいなら「脚の関節がこの角度で速度がこれくらいのときは、モータをこれくらい回せ」などのように、できるだけ多くのパターンを教える必要がある。また、モータが複数あると、正解を用意すること自体が非常に困難になる。
- 強化学習では、歩けた距離を報酬としてシステムに与えながら、何度も試行を繰り返し、報酬を大きくできるような制御ルールを覚えさせる。その結果、歩き方そのものを教えていないにも関わらず、長い距離を歩けるようになる。

深層強化学習

- 強化学習自体は、歴史のある学習手法。
- 近年、この強化学習と深層学習を組み合わせた方法が盛んに研究されており、この分野ことを深層強化学習という。
- 深層強化学習にはまだ課題が多く、利用にあたっては注意が必要。
  - 深層強化学習の主な課題
    - 試行錯誤を行うため学習に時間がかかる。(サンプル効率が悪い)
    - 報酬の設計が困難である。
    - 報酬が非常にまれにしか得られない環境では、学習が進みづらい。
    - 環境を用意する必要がある。
- もし、解決したい問題が、教師あり学習で解けるのであれば、そちらを選択する方が良い

強化学習の歴史

- 強化学習という名前は、Skinner博士の提唱した脳の学習メカニズムであるオペラント学習(ここではオペラント条件づけの強化を指す)に由来する。
- オペラント学習は、スキナー箱と呼ばれるラット実験によって、「特定の動作に対して報酬を与えると、その動作が強化される」ことを発見した(1940年頃)。この学習メカニズムがオペラント学習と呼ばれている。
- オペラント(operant)とは、オペレート(動作する, operate)からのSkinner博士による造語。
- ちなみに、パブロフの犬の実験に基づく条件づけ理論は、 1900年代初頭に提唱されている。



スキナー箱. In *Wikipedia*. Retrieved 12:51, April 30, 2018, from https://ja.wikipedia.org/w/index.php?title=%E3%82%B9%E3%82%AD%E3%83%8A%E3%83%BC%E7%AE%B1&oldid=68383226

ラットが箱の中のボタンを押すと、餌(報酬)が出てくる仕組みにしておく。ラットは、最初は偶然にボタンに触れる。すると餌が出てくる。これを何度か繰り返すうちに、ボタンと餌の関係性を覚え、ボタンを押す動作を繰り返すようになる。

強化学習の歴史

- 1990年代後半、Shultz博士らによって、脳科学の実験でオペラント学習による強化がニューロンレベルでも観測されるようになった。
- この実験によって、強化学習は脳の学習メカニズムと似ている点が示された。
- 1990年代後半から2000年代初頭には強化学習のブームが起こった。
- このブームは、2000年代後半にいったん下火になっていたが、強化学習に深層学習が使われ始め、また盛り上がってきている。
- 深層学習を用いた強化学習のことを深層強化学習(deep reinforcement learning)という。

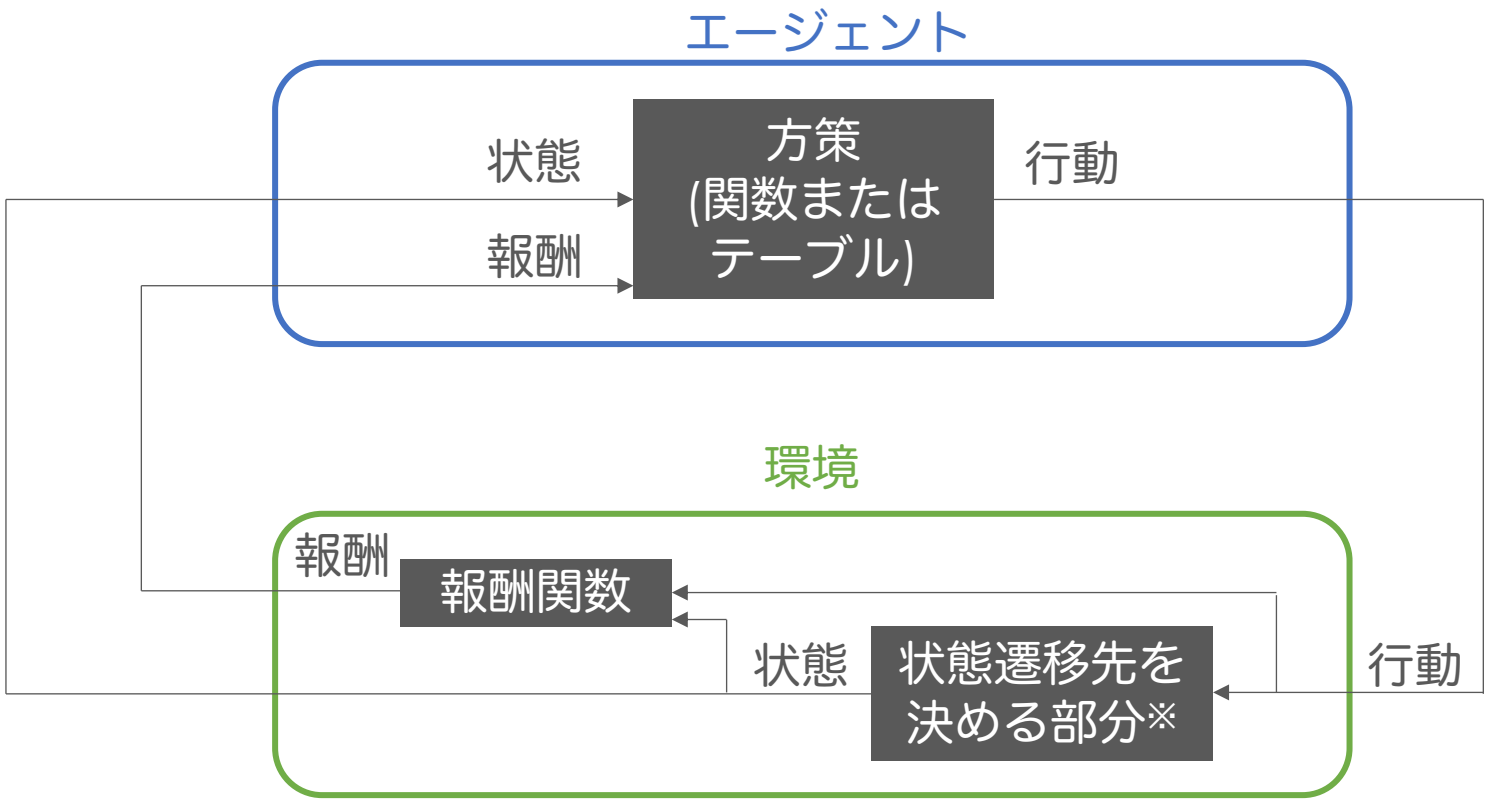
強化学習における基本概念

- 強化学習では主に以下の概念が利用される。

    1. 行動をおこなう主体のことを「**エージェント**」という
    2. 環境に対して行う働きかけの種類のことを「**行動**」という
    3. エージェントが働きかけを行う対象のことを「**環境**」という
    4. 行動を決める規則のことを「**方策（ポリシー）**」という
    5. エージェントが行動すると次の「**状態**」に遷移する
    6. ある状態の良さを「**報酬**」で評価する
    7. 一連の行動のまとまりを「**エピソード**」という

強化学習における基本概念

- 基本概念の関係を表した図を以下に示す。
- ここでは、マルコフ決定過程を仮定している。

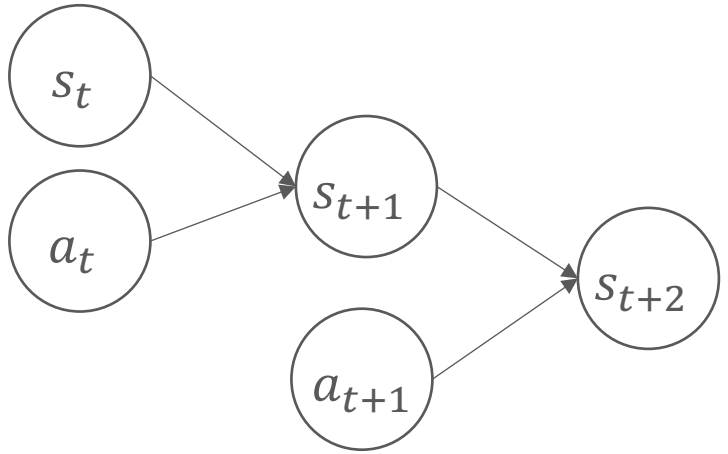


※「状態遷移先を決める部分」としては、実環境もしくはシミュレータなどの計算モデルが当てはまる。

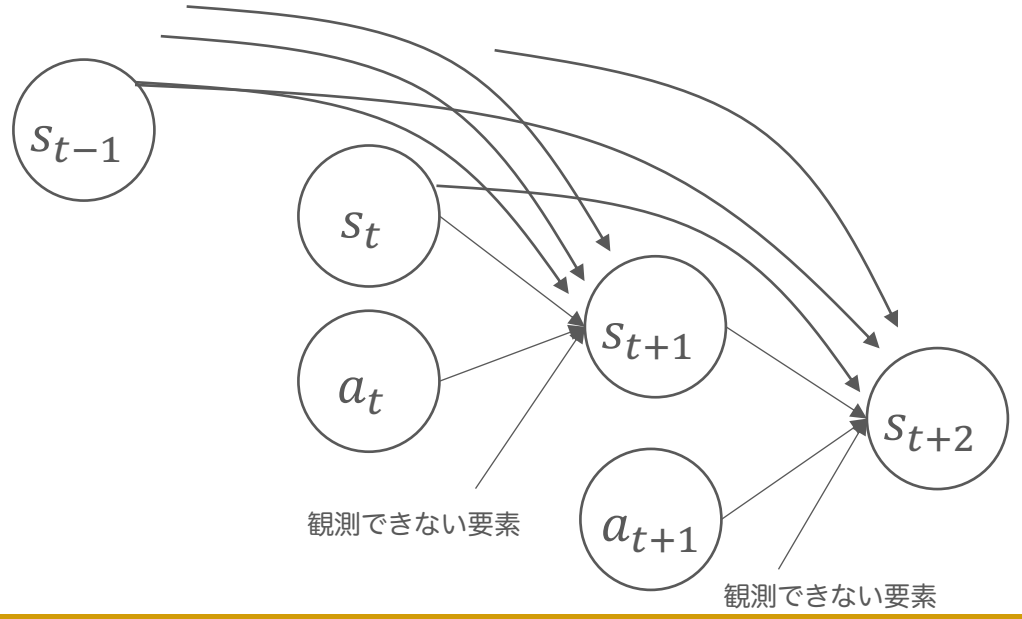
マルコフ決定過程

- マルコフ決定過程(markov decision process, MDP)とは、次の状態s_{t+1}が現在の状態s_tと採用した行動a_tで確定するシステムのこと。
- マルコフ決定過程でない場合の例としては、現在の状態s_tだけでなく、過去の状態$s_{t-1}, s_{t-2}, \ldots$や観測できない要素にも影響されて次の状態s_{t+1}が決まるようなシステムが挙げられる。
- 強化学習手法では基本的に、マルコフ決定過程を仮定している。

マルコフ決定過程の例



マルコフ決定過程でない例



強化学習における「状態」

- 状態とは、ある時刻の制御対象の様子を再現するために必要な情報。
- 例えば、歩行ロボットであれば、「各脚の関節の角度や速度」という情報。
- 例えば、囲碁や将棋であれば、「盤面のどの位置にどのコマがあるか」という情報。
- 深層強化学習では、盤面などの状態を画像として扱うことがある。

強化学習における「報酬」

- エージェントがとった行動の良し悪しに応じて付与されるもの。
- 最終的に受け取る総報酬を最大化することがエージェントの目的。
- 例
  - 迷路問題であれば、ゴールに到達したら報酬1を与えたりする。
  - カートポール問題であれば、立っているときだけ報酬1を与えたりする。
- 報酬の与え方(報酬関数)は確立されておらず、経験や試行錯誤で決めるのが一般的。
  - 報酬の与え方(報酬関数)を少し変えるだけで、学習の結果が大きく変わることがあり、それが強化学習の大きな課題。

強化学習における「環境」

- エージェントが関与できない部分。
- 「報酬関数」と、「状態遷移先を決める部分」で構成される。
- 「状態遷移先を決める部分」としては、実環境もしくはシミュレータなどの計算モデルが当てはまる。
    - 通常、強化学習を行う際は、実環境かシミュレータを用意する。

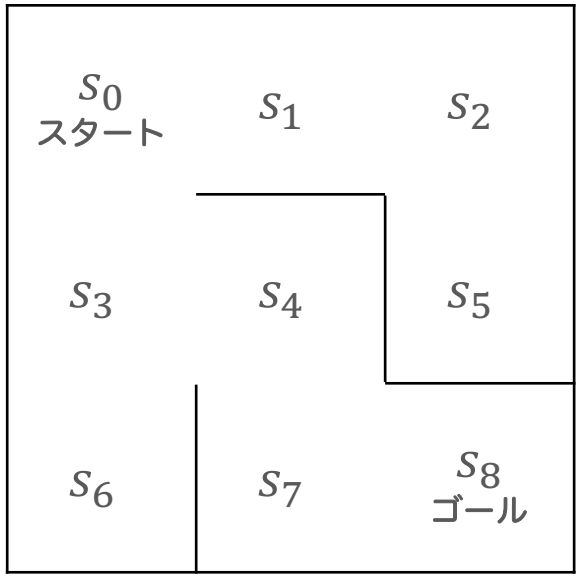
[演習] 迷路問題用のクラスを確認

- 8_1_state_action.ipynb
  - 状態を定義するためのStateクラスと行動を定義するためのActionクラスを確認しましょう
- 8_2_environment.ipynb
  - 環境を定義するためのEnvironmentクラスを確認しましょう

迷路問題とは
- 迷路問題とは、スタートからゴールまで早くたどり着ける方法を考える問題。
- 迷路上に、壁や穴を設定したりする。

強化学習を行う上での設定例
- 迷路上を移動するロボットをエージェントに設定する。
- エージェントの現在位置を状態とする。右図であればs_0からs_8の9種類。
- エージェントが実行できる行動は、a_1:上へ移動、a_2:右へ移動、a_3:下へ移動、a_4:左へ移動の4種類などとする。

迷路問題の例



[演習] マルコフ決定過程の考え方の確認

- 8_3_markov_decision_process.ipynb
  - マルコフ決定過程の考え方を迷路問題を用いて確認しましょう

強化学習の基礎2

探索と利用のトレードオフ

- 探索は、様々な行動を試し、行動・状態・報酬の関係を見い出すための情報収集行動のことを指す。
    - 探索が十分であるほど方策の最適化が十分におこなわれるが、最適な方策に従い行動する数が減ってしまい、報酬和が減ってしまう。
- 利用は、探索の結果得られた方策を運用し、より多くの報酬が得られるようにするための行動を指す。
    - 利用の数が多いほど報酬和が高くなる可能性があるが、探索が不十分で方策が正しくなければ、かえって報酬和を大きく減らしてしまう。

探索と利用のトレードオフの例

- 多腕バンディット問題
  - 当たりの出やすさが異なる複数のスロットマシンがある。
  - 当たりやすいマシンを探すには、何回か試してみる必要がある。
  - 試す回数を増やすと、当たる確率に確信が持てるようになるが、一方で、外れを経験する回数も多くなってしまう。
  - なので、ある程度探索して、良いマシンが見つかったら、それを引き続けるのが良い。
- 昼食問題
  - 会社近辺で、できるだけ美味しい定食を毎日食べたい。
  - できだけ美味しい定食は、毎日違う店に入って探さなければ見つからない。
  - しかし、毎日探し続けていると、美味しくない定食を食べるハメになる日も出てきてしまう。
  - なので、ある程度探して、良い店が見つかったら、そこに通い続けるのが良い。

探索と利用のバランシング手法例

- greedy(貪欲) 法
  - 最も単純な方法。
  - これまでの経験に基づき、報酬和の期待値が最大になる行動を選択する。
  - 最初にまとめて探索をおこない、最大の報酬和が得られる最適行動を決定。以後探索を行わず「最適な」行動を常にとる。
- ε-greedy 法
  - 通常は報酬和の期待値が最大になる行動をとるが、確率εでランダムに行動を選び、探索行動を発生させる。

[演習]ε-greedy法の考え方の確認

- 8_4_epsilon_greedy.ipynb
  - ε-greedy法の考え方をコイントス実験を用いて確認しましょう

良い方策を得るには？

- 強化学習では、良い方策を如何に得るかということが最大の関心事である。
  - つまり、方策が学習の対象
- 良い方策を得るために、収益および価値という概念を導入する。
- 価値という指標を用いることにより、方策の良さの評価、現在の状態の評価、行動の評価を行えるようになる。
  - 価値が方策を兼ねる場合もある
    - 例、Q学習

収益

- 収益(return)は、将来得られる報酬(reward)の総和のこと。
- ある時間ステップtにおける収益G_tは以下のように定義される。

$$G_t = \sum_{k=0}^{\infty} r_{t+1+k}$$

r：報酬

t：時間ステップ

- 強化学習では、収益が多く得られる方策が良いと考える。
  - 一次的な収入(報酬)よりも、長期的な収入(収益)を重視する。

割引率

- 割引率(discount rate)は、収益を算出する際に、短期的視点を大事にするか、長期的視点を大事にするか、を調整する値。
- 割引率は以下のように導入される。

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

γ：割引率

- $\gamma = 0$であれば、 $k = 0$のときの報酬が収益になるため、エージェントは一次的な収入(報酬)だけを考えて行動する。
- $\gamma = 1$に近いほど、エージェントは将来得られる報酬を考慮した行動をする。

価値

- 収益の期待値を価値(value)という。
  - 収益は、確率的に決定されるため、期待値で考えた方が方策を評価しやすい。
- 強化学習では、価値が大きくなる方策が良いと考える。
- 価値の考え方は、状態価値(または単に価値)と行動価値の2つある。
- 状態価値(または単に価値)
  - ある状態sにおいて、今後方策πに従って行動していった場合の期待収益のこと。
  - 方策πの下で、状態sであることの価値を表す。

将棋であれば、対局中のある時点での盤面の良さを表す

- 行動価値
  - ある状態sにおいて行動aをとり、その後に方策πに従って行動していった場合の期待収益のこと。
  - 方策πの下で、状態sにおいて行動aをとることの価値を表す。

将棋であれば、対局中のある時点において、ある駒の動かし方の良さを表す

状態価値

- 状態価値

$$V^{\pi}(s) = \mathbb{E}^{\pi}[G_t | s_t = s]$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

t：時間ステップ

$V^{\pi}(s)$：方策πの下での状態価値(または単に価値)

$\mathbb{E}^{\pi}[\cdot]$：期待値

G: 収益(将来に得られる報酬の和)

s：状態

r：報酬

γ：割引率$(0 \leq \gamma \leq 1)$, 将来の報酬をどれだけ割り引くかを決める値

- 状態価値を求めるための関数のことを状態価値関数(または単に価値関数)と呼ぶ。

行動価値

- 行動価値

$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}[G_t | s_t = s, a_t = a]$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k}$$

t：時間ステップ

$Q^{\pi}(s,a)$：方策πの下での行動価値

$\mathbb{E}^{\pi}[\cdot]$：期待値

G: 収益(将来に得られる報酬の和)

s：状態, a：行動

r：報酬

γ：割引率$(0 \leq \gamma \leq 1)$, 将来の報酬をどれだけ割り引くかを決める値

- 行動価値を求めるための関数のことを行動価値関数と呼ぶ。

ベルマン方程式

- 価値関数を再帰的に定義し期待値を展開すると以下のようになる。これをベルマン方程式という。ベルマン方程式が成り立つためには、マルコフ決定過程である必要がある。
- 状態価値関数に関するベルマン方程式

$$V^{\pi}(s) = \mathbb{E}^{\pi}[G_t|s_t = s] = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}|s_t = s\right] = \mathbb{E}^{\pi}\left[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+2+k}|s_t = s\right]$$

収益は再帰的に計算される

$$= \sum_{a}\pi(a|s)\sum_{s'}P(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$

ここは依然として未知なので、これを推定する方法が別途必要

状態sで行動aをとる確率

状態sで行動aをとった場合に、状態s'に遷移する確率

全行動に関する期待値

全状態に関する期待値

$\pi(a|s)$：方策関数

$P(s'|s,a)$：状態遷移関数

$r(s,a,s')$:報酬関数

方策関数は、状態sにおいて行動aをとる確率を返す関数。
状態遷移関数は、状態sから行動aをとった場合に、状態s'に遷移する確率を返す。環境によって決まる。
報酬関数は、状態sから行動aをとって状態s'に遷移した場合の報酬を返す。

ベルマン方程式

$\pi(a|s)$：方策関数

$P(s'|s,a)$：状態遷移関数

$r(s,a,s')$:報酬関数

- 行動価値関数に関するベルマン方程式

$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}[G_t|s_t = s, a_t = a] = \mathbb{E}^{\pi}\left[\sum_{k}^{\infty} \gamma^k r_{t+1+k}|s_t = s, a_t = a\right]$$

$$= \mathbb{E}^{\pi}\left[r_{t+1} + \gamma \sum_{k}^{\infty} \gamma^k r_{t+2+k}|s_t = s, a_t = a\right] = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$

収益は再帰的に計算される

全状態に関する期待値

$V^{\pi}(s) = \sum_a \pi(a|s) Q^{\pi}(s,a)$ を$V^{\pi}(s')$に代入する

V^{π}とQ^{π}の関係式

代入

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(s',a')]$$

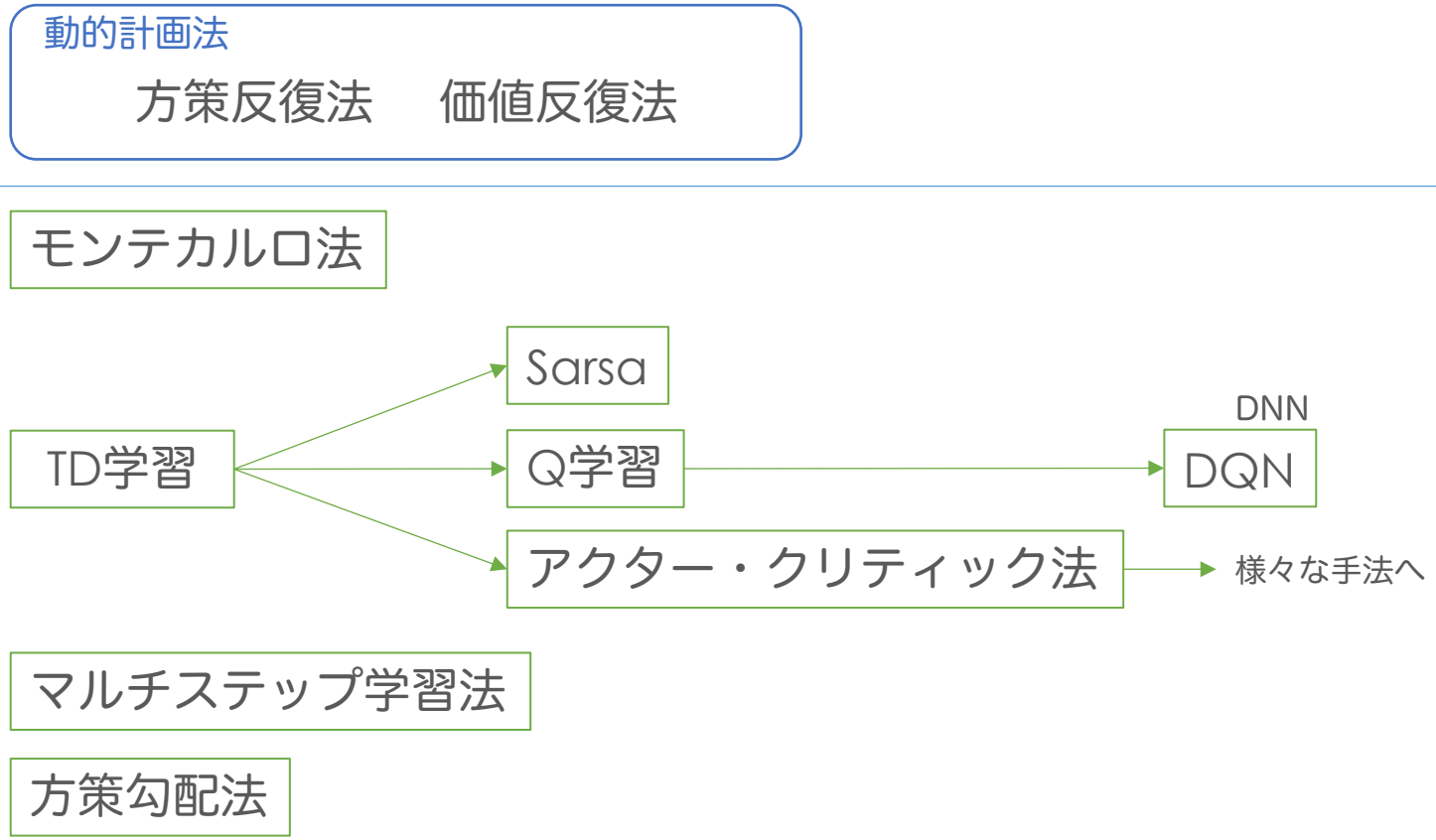
全行動に関する期待値

ここは依然として未知なので、これを推定する方法が別途必要

強化学習の各種手法

強化学習手法の一覧

- 強化学習の手法のうち代表的なものを以下に示す。



動的計画法

- 動的計画法(dynamic programming)とは、最適化手法の1つで、問題を複数の部分問題に分割し、部分問題を解きながら全体最適解を求めていく手法。
- 制御理論や強化学習の基礎となる概念。
- Richard E Bellmanの論文が起源。

強化学習における動的計画法

- 強化学習における動的計画法の代表的な手法として、方策反復法と価値反復法がある。
  - どちらも、エージェントを動かさずに、環境の情報だけから最適計画をつくるため、シミュレータなどの環境のモデルが必要。
  - どちらも、全状態を繰り返し探索するため、状態数が多い場合は、計算時間が膨大になる。
  - 方策反復法または価値反復法は、ベルマン方程式を解く※1ための1つの方法と言える。
- 方策反復法と価値反復法のどちらが良い方法なのかというのは重要ではなく、これらの方法を用いると最適解が求まるということを理解することが大事。
- 近年の強化学習において動的計画法がそのまま使われることは少ないが、他の手法を理解する際の助けとなる。

※1　ここでは、ベルマン方程式を満たし、かつ価値が最も大きくなる価値関数を推定することをベルマン方程式を解くと呼ぶことにする。

方策反復法

- 方策反復法(policy iteration)とは、方策評価と方策改善を繰り返して、よりよい方策を得ることを目指す方法。
  - 状態価値V^{π}を使って方策πを改善(方策改善)し、その方策πを用いて状態価値V^{π}を更新する(方策評価) 。これを繰り返す。
  - 動的計画法では、任意の方策に対する状態価値を計算することを方策評価と呼ぶ。

方策反復法における状態価値関数

- 方策反復法の方策評価における状態価値関数$V^{\pi}(s)$ を以下に示す。

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$

- $V^{\pi}(s)$：ある方策πの下でのある状態sの価値
- $\pi(a|s)$：状態sでとるべき行動aを表す確率
- $P(s'|s,a)$：どの状態に遷移するかを表す確率
- $\sum_{a}$：全行動に関する期待値
- $\sum_{s'}$：全状態に関する期待値

- 状態価値V^{π}は、ある状態sにおいて、ある方策πに従って行動した場合の期待収益を意味する。
- 方策πが良い方策であるほど、状態価値V^{π}は大きくなると考えられるため、状態価値V^{π} によって方策πの良さを評価できることになる。
- このように、ある方策に対する状態価値V^{π}を計算することを方策評価という。

方策反復法

- 方策反復法の計算手順を以下に示す。

1.初期化

$s \in S$に対して$V(s)$と方策$\pi(a|s)$を任意に初期化する

2.方策評価

繰り返し：

$\Delta \leftarrow 0$

各$s \in S$について：

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$ ←状態価値$V(s)$の更新

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

$\Delta < \theta$(小さな値)ならば,繰り返しを終了

3.方策改善

$\text{policy-stable} \leftarrow \text{true}$

各$s \in S$について：

$a_p \leftarrow \arg\max_a \pi(a|s)$ ←方策π基準で行動を選んだ場合

$a_v \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$ ←行動価値基準で行動を選んだ場合

もし, $a_p \neq a_v$ならば,

$\pi(a_v|s) = 1, \pi(a_v以外|s) = 0, \text{policy-stable} \leftarrow \text{false}$ ←方策の更新

もし,$\text{policy-stable} = \text{true}$ならば,終了;それ以外は,2へ.

方策π基準で選んだ行動が、行動価値基準で選んだ行動と同じになってほしい。同じになったら、その方策は、期待収益を最大化する方策であると言えるから。

価値反復法

- 価値反復法(value iteration)とは、状態価値V^{π}の更新だけを行うことで、よりよい方策を得ることを目指す方法。
- 方策反復法における方策評価と方策改善の処理を1つに結合した考え方になっている。
- 方策反復法に比べ、方策評価と方策改善の繰り返し処理がない分、計算コストが小さい。
- 状態価値V^{π}の更新が終わったら、最後に最適方策πを求める。

価値反復法における状態価値関数

- 価値反復法では、状態価値の更新式を以下のように定義し、この計算を繰り返すことによって状態価値を更新していく。

$$V_{k+1}(s) \overset{\text{def}}{=} \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_k(s')]$$

$k+1$ステップ目におけるsの価値

とりうる行動のうち値が最大になるものを返す

どの状態に遷移するかを表す確率

全状態に関する期待値

kステップ目における状態s'の価値

k： ステップ数

最終的な状態価値のイメージ
(迷路問題においてゴールで報酬100がもらえる場合, $\gamma = 0.9$)

81	90	100	G
73	81	90	100
66	73	81	90
S	66	73	81

- ここでは、「常に期待収益を最大にする行動をとること」が大前提になっており、方策を改善していくという考えはない。
- 上記の更新を続けると、状態価値の大きな状態に遷移できる状態ほど状態価値が大きくなっていく。
  - つまり、大きな報酬が得られる状態sに近いほど、状態価値が大きくなっていく。
- 計算が終わると、以下の式により、最適方策を求める。

$$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_k(s')]]$$

価値反復法

- 価値反復法の計算手順を以下に示す。

1.初期化

Vを任意の値で初期化. 例えば, $s \in S$に対して$V(s) = 0$.

2.価値更新

繰り返し：

$\Delta \leftarrow 0$

各$s \in S$について：

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$ ←状態価値$V(s)$の更新

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

$\Delta < \theta$(小さな値)ならば,繰り返しを終了

3.方策πを出力

各$s \in S$について：

$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V(s')]$

[演習] 方策反復法および価値反復法の考え方の確認

- 8_5_policy_iteration.ipynb
  - 方策反復法(policy iteration)の考え方を迷路問題を用いて確認しましょう
- 8_6_value_iteration.ipynb
  - 価値反復法(value iteration)の考え方を迷路問題を用いて確認しましょう

モンテカルロ法

<モンテカルロ法とベルマン方程式との関係>
モンテカルロ法は、ベルマン方程式を解いているわけではない。ベルマン方程式では、 状態遷移確率を用いることで価値の期待値を表した。しかし、 この状態遷移確率は、 環境が既知でないと分からない。例えば、 将棋や囲碁などは相手の手次第でこれらが変わってしまう。そこで, 実際に得られた報酬の平均をとることで各状態に対しての価値の期待値を計算することにしたのがモンテカルロ法である。

- モンテカルロ法とは、遷移のサンプルを取得し、収益を平均化することによって、価値関数を推定する方法。
  - 推定した価値関数によって方策を評価する。
- 価値関数の改善と方策の改善とを組み合わせることによって、最適方策を見つけることができる。
  - 価値関数の改善または方策の改善は、エピソード終了後に行われる。
- サンプルを沢山取得し、そこから平均を求めるという計算を行うため、「モンテカルロ」という名前がついている。
- サンプル取得においては、全ての状態行動対が無限回訪問されることを仮定する。
  - これを開始点探査(exploring starts, ES)の仮定と呼ぶ。

モンテカルロ法

- モンテカルロ法によって、行動価値関数を更新する場合の計算手順を以下に示す。
- 状態価値関数の更新も同様に可能だが、最終的に最適方策を得たいのであれば、最適行動価値関数を推定する方が良い。

全てのs, a に対して初期化
$Q(s, a) \leftarrow$ 任意の値
$\pi(s) \leftarrow$ 任意の値
$Returns(s, a) \leftarrow$ 空のリスト
繰り返し：
(a) s_0, a_0を可能な全ての組み合わせの中からランダムに選ぶ
(b) s_0, a_0を出発点とし、方策πに基づいて、エピソードを生成する
(c)エピソード中に出現する各状態s, 行動aの組み合わせについて:
$R \leftarrow s, a$の初回発生後の収益
Rを$Returns(s, a)$に追加する
$\mathrm{G} \leftarrow \mathrm{average}\big(Returns(s, a)\big)$
$Q(s, a) \leftarrow G$
(d)エピソード中の各sについて:
$\pi(s) \leftarrow \arg\max_a Q(s, a)$

※開始点探査を仮定したモンテカルロ法の計算手順

←開始点を毎回ランダムに変更することにより、開始点探査を行なっている。

←価値関数の改善または方策の改善は、エピソード終了後に行われる

[演習] モンテカルロ法の考え方の確認

- 8_7_monte_carlo.ipynb
  - モンテカルロ法の考え方を迷路問題を用いて確認しましょう

TD学習

- TD学習(temporal difference learning, TD法とも呼ばれる)とは、目標の価値と現在の価値のずれを修正していくことによって、価値関数を推定する方法。
  - 目標の価値と現在の価値のずれのことをTD誤差という。
- 動的計画法とモンテカルロ法の考え方を組み合わせたような学習方法と言える。
  - モンテカルロ法と同様、エージェントを動かしたことによって得られた経験を利用する。
  - 動的計画法と同様、エピソード終了を待たずに、価値関数や方策を更新することができる。
- TD学習の具体的な手法として、SarsaとQ学習がある。

SarsaとQ学習

- SarsaとQ学習は、TD学習の具体的な手法。
- SarsaとQ学習は、ステップ毎に行動価値関数を更新していくため、経験をすぐに次の行動に生かすことができる。
- SarsaとQ学習は、価値の更新をひたすら繰り返すという方法であり、この点は価値反復法と考え方が似ている。
- SarsaとQ学習は、行動価値関数に関するベルマン方程式を試行錯誤の経験によって解く※1ためのアルゴリズムといえる

※1　ここでは、ベルマン方程式を満たし、かつ価値が最も大きくなる価値関数を推定することをベルマン方程式を解くと呼ぶことにする。

Sarsa

- Sarsaは、経験によって行動価値関数を更新するアルゴリズムの一つ。
- Sarsaの名前の由来。
  - 現在の状態(State)、現在の行動(Action)、次の報酬(Reward)、次の状態(State)、次の行動(Action)
- Sarsaアルゴリズムにおける行動価値関数Qの更新式。

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

目標値のような役割

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

目標値との誤差と解釈できる部分。
TD(temporal difference)誤差と呼ばれる。

α: 学習率

遷移後の状態において、使用している方策（ε-greedyなど）によって選択された行動をとるときの行動価値

Sarsa

- Sarsaの計算手順を以下に示す。

全てのs, a に対して初期化
　　$Q(s, a) \leftarrow$ 任意の値
各エピソードに対して繰り返し：
　　sを初期化
　　Qから導かれる方策(例,ε-greedy)を用いて状態sでとる行動aを選択する
　　エピソード内の各ステップに対して繰り返し：
　　行動aを実行し, r, s'を観測する
　　Qから導かれる方策(例,ε-greedy)を使って,状態s'での行動a'を選択する
　　$Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma Q(s', a') - Q(s, a)\big)$
　　$s \leftarrow s'$
　　$a \leftarrow a'$

方策によって導かれた結果が行動価値関数の目標として利用される

Q学習

- Q学習(Q-learning)は、経験によって行動価値関数を更新するアルゴリズムの一つ。
- Sarsaとは異なり、行動価値関数Qの更新は行動の決定結果に依存しない。
- Sarsaよりも行動価値関数の収束が早い。
- Q学習における行動価値関数Qの更新式

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

目標値のような役割

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

α: 学習率

遷移後の状態において、行動価値関数を最大にする行動をとったときの行動価値

目標値との誤差と解釈できる部分。
TD(temporal difference)誤差と呼ばれる。

Q学習

- Q学習の計算手順を以下に示す。

全てのs, a に対して初期化
　　$Q(s, a) \leftarrow$ 任意の値
各エピソードに対して繰り返し：
　　sを初期化
　　エピソード内の各ステップに対して繰り返し：
　　　　Qから導かれる方策(例,ε-greedy)を用いて状態sでとる行動aを選択する
　　　　行動aを実行し, r, s'を観測する

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

$s \leftarrow s'$

最も価値が高い行動が行動価値関数の目標として利用される(選択された行動aの影響を受けない)

SarsaとQ学習

- SarsaもQ学習も、経験していない「状態と行動の組み合わせ」に対する行動価値関数は更新されないことになる。
  - 探索しないと更新されない。
- Sarsaは方策オン型の手法に位置付けられる。
  - 行動を決定する方策が行動価値関数の更新にも利用される。
  - 行動価値関数を更新する際、行動価値の小さな探索結果も考慮されやすいという利点があるが、計算が不安定になりやすい。
- Q学習は方策オフ型の手法に位置付けられる。
  - 方策とは独立に、行動価値関数が更新される。
  - 行動価値関数を更新する際、探索の影響を受けにくいため計算が安定的に行えるという利点があるが、行動価値の小さな探索結果は反映されにくい。

アクター・クリティック法

<アクター・クリティック法と他の手法との関係性>
アクター・クリティック法は、方策反復法と考え方が似ている。SARSAやQ学習はクリティックのみを学習していると言える。方策勾配法はアクターのみを学習していると言える。

- アクター・クリティック法は、行動を決める部分 (actor,アクター)と方策を評価する部分(critic,クリティック)を別々にモデル化する方法の総称。
  - 方策の評価は、価値を推定することによって行う。
  - ある方策を用いた場合の価値が大きければ、その方策は良い方策であると考えることができる。
- 方策が価値関数から独立しているため、方策を任意に拡張することが可能。
  - アクターに、後述する方策勾配法を用いることも可能。
- アクター・クリティック法は、汎用的な考え方であり、近年の様々な手法で取り入れられている。
  - 例、DPG, DDPG, A2C, A3Cなど
- 一般に、アクター・クリティックにすると、学習が安定しやすくなる。
- 参考論文
  - A survey of actor-critic reinforcement learning: standard and natural policy gradients. Grondman et al.. 2012.
    https://www.researchgate.net/publication/257916782_A_Survey_of_Actor-Critic_Reinforcement_Learning_Standard_and_Natural_Policy_Gradients

マルチステップ学習

- モンテカルロ法では、1エピソード終了後に、価値関数を更新した。
- TD学習では、1ステップごとに、価値関数を更新した。
- 更新頻度をこれらの間に設定する学習方法のことをマルチステップ学習という。
  - 2ステップや3ステップがよく用いられる。
  - 価値関数を更新する際、実際に得られた報酬をより多く(ステップ数という意味で)使えるため、価値推定の精度が向上する場合がある。
- マルチステップ学習も汎用的な考え方であり、近年の様々な手法で取り入れられている。

[演習] Sarsa, Q学習, アクター・クリティック法の考え方の確認

- 8_8_sarsa.ipynb
  - Sarsaの考え方を迷路問題を用いて確認しましょう
- 8_9_q_learning.ipynb
  - Q学習の考え方を迷路問題を用いて確認しましょう
- 8_10_actor_critic.ipynb
  - アクター・クリティック法の考え方を迷路問題を用いて確認しましょう

方策のモデル化方法

- ここまでの手法は、状態と行動が対になった方策(テーブル型)を用いることを前提としていたが、パラメータをもった関数で方策をモデル化するという方法もある。
- テーブル型の方策の場合、状態や行動が連続値になると、方策をつくることが困難になる。
- 方策をパラメータをもった関数でモデル化すると、連続値の状態や行動を扱いやすくなる。

行動の種類

状態の種類	a_1 (上へ移動)	a_2 (右へ移動)	a_3 (下へ移動)	a_4 (左へ移動)
s_0	0	0.5	0.5	0
s_1	0	0.5	0	0.5
s_2	0	0	0.5	0.5
s_3	0.333	0.333	0.333	0
s_4	0	0	0.5	0.5
s_5	1	0	0	0
s_6	1	0	0	0
s_7	0.5	0.5	0	0

テーブル型方策の例

パラメータをもった方策

- 期待収益を目的関数$J(\boldsymbol{\theta})$として、これを最大化する方策π_θのパラメータ$\boldsymbol{\theta}$を求めることを考える。
- 例えば、パラメータ$\boldsymbol{\theta}$を以下のように更新していくことで、最適な方策π_θを求めたいと考える。

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$$

$\boldsymbol{\theta}$: パラメータ
$\delta\boldsymbol{\theta}$: 更新量

- このような場合、方策勾配法が使える。

方策勾配法

<方策勾配法とベルマン方程式との関係>
方策勾配法は、期待収益を最大にする方策を直接求めようとする方法であり、ベルマン方程式を解くための方法ではない。ただし、後述する方策勾配定理では、勾配を導出する過程で、行動価値に関するベルマン方程式が用いられている。

- 方策勾配法(policy gradient)は、確率的方策π_θのパラメータθを勾配法で更新していく方法。
- AlphaGoなど、最近の深層強化学習でよく用いられている。
- 方策勾配法の計算は、以下の3手順からなる。
    1. 確率的方策π_θのモデルを決める。
    2. 目的関数を決める。
    3. 更新方法を考える。

方策勾配法 - 1.確率的方策π_θのモデルを決める -

- 確率的方策π_θのモデルとしては、例えば、以下のような形が考えられる。
  - 状態と行動がともに離散の場合(ソフトマックス関数)

$$\pi_\theta(a|s) = \frac{\exp(\theta_{sa})}{\sum_b \exp(\theta_{sb})}$$

  - 状態が連続で行動が離散の場合(線形関数+ソフトマックス関数)

$$\pi_\theta(a|s) = \frac{\exp(\boldsymbol{\theta}^T \phi(s,a))}{\sum_b \exp(\boldsymbol{\theta}^T \phi(s,b))}$$

$\phi(s,a)$：状態と行動によって特徴付けられる任意の関数ϕによって定まる特徴ベクトル

  - 状態と行動がともに連続の場合(ガウス分布)

$$\pi_\theta(a|s) = \frac{1}{(2\pi)^{\frac{d_a}{2}} |\boldsymbol{C}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{a} - \boldsymbol{W}\boldsymbol{s})^{\mathrm{T}} \boldsymbol{C}^{-1} (\boldsymbol{a} - \boldsymbol{W}\boldsymbol{s})\right)$$

$\boldsymbol{s} \in \mathbb{R}^{d_s}, \boldsymbol{a} \in \mathbb{R}^{d_a}, \boldsymbol{W} \in \mathbb{R}^{d_a \times d_s}, \boldsymbol{C} \in \mathbb{R}^{d_a \times d_a}$
この時、θは$\boldsymbol{W}$と$\boldsymbol{C}$の要素を並べたベクトルとなる。

- いずれもθに関して、微分可能である点が大事。

方策勾配法 - 2.目的関数を決める -

- ある確率的方策π_θのもとで行動を行った場合の期待収益によって、その確率的方策π_θの良さを量ることにする。
- 初期ステップからの期待報酬和を最大化すべき対象と考えると、目的関数は以下のようになる。

$$J(\theta) = \mathbb{E}[G_0|s_0] = \mathbb{E}\{\sum_{t=1}^{\infty} \gamma^{t-1} r_t \,|s_0\}$$

方策勾配法 - 3.更新方法を考える -

- 確率的方策π_θのパラメータθをその勾配を用いて更新していく。

$$\theta^{t+1} = \theta^t + \Delta\theta = \theta^t + \eta\nabla_\theta J(\theta)$$

勾配を用いて更新

$J(\theta)$: 目的関数

η : 学習率

方策勾配法 - 3.更新方法を考える -

E資格対策
方策勾配定理に関する式変形は暗記してください！

- 方策勾配定理
  - 方策勾配定理を用いると、勾配を以下のような形で表現できる。
  - $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\frac{\partial \pi_\theta(a|s)}{\partial \theta}\frac{1}{\pi_\theta(a|s)}Q^\pi(s,a)\right]$

    $= \mathbb{E}_{\pi_\theta}\left[\frac{\partial \pi_\theta(a|s)}{\partial \theta}\frac{\partial \log \pi_\theta(a|s)}{\partial \pi_\theta(a|s)}Q^\pi(s,a)\right]$ 対数の微分の公式を逆に使って変形した

    $= \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta}Q^\pi(s,a)\right] = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\, Q^\pi(s,a)]$

    方策勾配定理によって導かれた勾配の形

    この勾配の期待値は解析的に求めることができないため近似的手法を使って求める。
  - 原著論文
    - Richard S. Sutton et al., Policy Gradient Methods for Reinforcement Learning with Function Approximation ,https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf

この論文のAppendixに方策勾配定理の証明が載っている

方策勾配法

エピソード的タスクとは、囲碁やゲームなどのように、開始状態へリセットされるタイミングがあるタスクのことである。
連続タスクとは、ロボットなどのように、明確なリセットタイミングがなく、連続的に動作し続けるタスクのことである。

- 前頁の勾配は、解析的に求めることができないため、π_θに基づいて経験したサンプルを用いて近似する場合がある。これをモンテカルロ近似という。

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\, Q^\pi(s,a)] \approx \frac{1}{N}\sum_{n=1}^{N}\frac{1}{T}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^n|s_t^n)\, Q^\pi(s_t^n, a_t^n)$$

この勾配は求めることができる。π_θがガウス分布であれば、ガウス分布の対数を微分すれば良い。

未知

N:エピソード数，T:ステップ数

1エピソード毎にパラメータを更新する場合は、$N = 1$とすればよい。
1タイムステップ毎にパラメータを更新する場合は、$N = 1, T = 1$とすればよい 。

- しかし依然として、$Q^\pi(s_t^n, a_t^n)$が未知である。 REINFORCEアルゴリズムでは、$Q^\pi(s_t^n, a_t^n)$を以下のように扱う。
  - エピソード的タスクの場合：$Q^\pi(s_t^n, a_t^n) = \sum_{k=t}^{T} r_k^n$
    累積報酬和
  - 連続タスクの場合($N = 1, T = \infty$)：$Q^\pi(s_t, a_t) \approx r_t$
    $N = 1$なので、nは省略　　タイムステップtで得られた報酬で近似
  - REINFORCEアルゴリズムの原論文
    - Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. https://link.springer.com/article/10.1007/BF00992696

Deep Q-Network

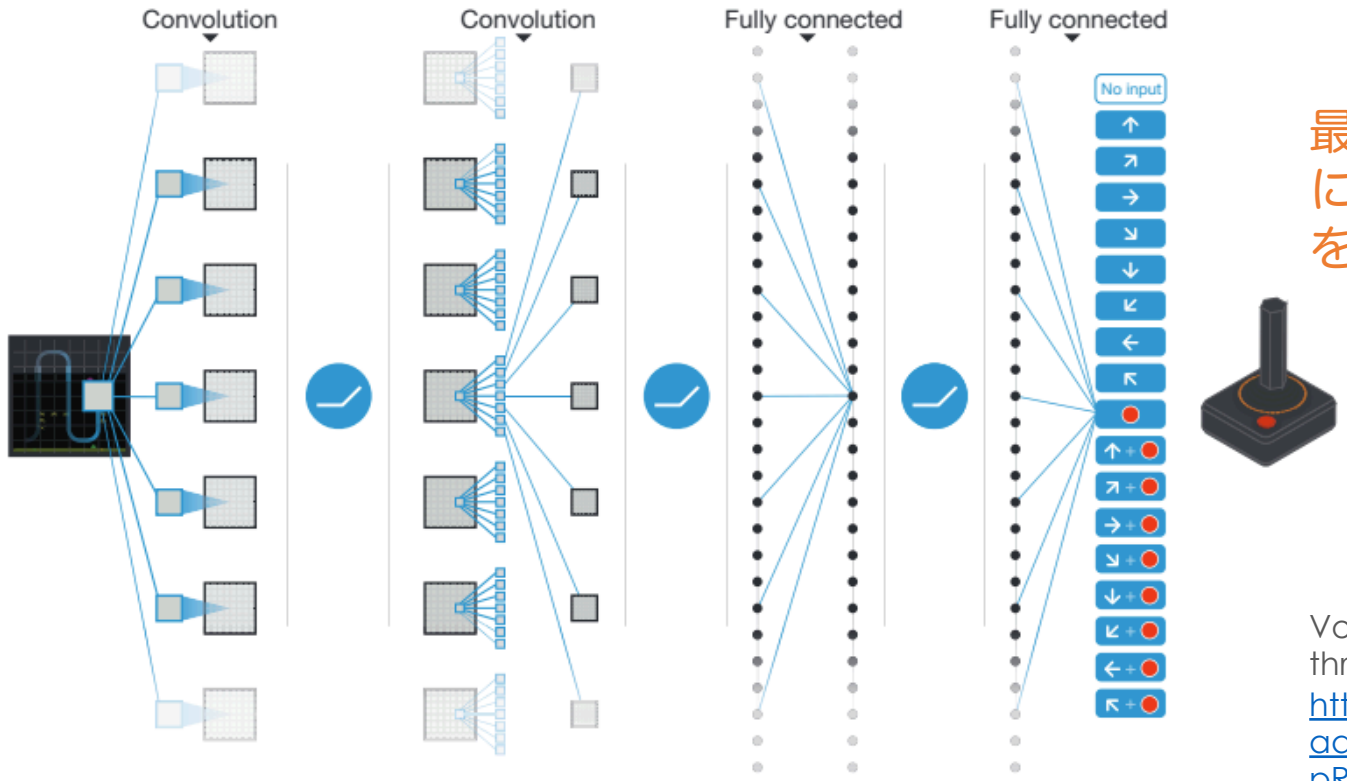
Deep Q-Network

- Deep Q-Networks (DQN) [Mnih et al. 2013; Mnih et al. 2015] は、強化学習に深層学習を適用し成功した一つの例である。
- DQNは、Atari2600の様々なゲームに対して、チューニングをせずに学習しても高い性能を示した。
- DQNでは、ゲームの画面自体を入力とするCNNモデルを行動価値関数(Q関数)として利用している。
- CNNでは、入力として状態情報（ある時点におけるゲーム画面の画像）を受け取り、その時とりうる各行動の価値を出力する。
- 原著論文
  - Volodymyr Mnih et al., Human-level control through deep reinforcement learning, https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
  - Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. Playing Atari with Deep Reinforcement Learning.https://arxiv.org/pdf/1312.5602.pdf

Deep Q-Network

- DQNのネットワーク構成を以下に示す。

直近4フーレム分の画面画像を入力する。この画面が状態に相当する。



最大18種類の行動に対する行動価値を出力する。

Volodymyr Mnih et al., Human-level control through deep reinforcement learning, https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

Deep Q-Network

- DQN は、NNで出力される行動価値が真の行動価値に近づくように学習していく。
  - Q学習アルゴリズムに基づいている。
- 価値関数をDNNにより近似する手法はDQNが登場するよりも古くから提案されていたかが、学習が不安定であった。それらの課題に取り組んだ点が、DQNの重要な貢献である。
- 学習を安定化するために、DQNで導入された4つの工夫
  - 体験再生(experience replay)
  - Target Q-Networkの固定
  - 報酬のクリッピング
  - Huber損失

体験再生

- DQNでは、「入力データが時系列データであり、入力データ間に独立性がない」という問題に対して、体験再生(experience replay)を適用した。
- [対象としている問題点]
  - 時系列データは、近い時間のデータが似通ってしまうため、相関性の強い系列となる。
  - 強い相関性をもつ入力系列に対して学習を行うと、直近の入力に引きずられてパラメータが修正されるため、過去の入力に対する推定が悪化し、収束性が悪くなる。
- [体験再生の仕組み]
  - エージェントが経験した状態・行動・報酬・遷移先は一旦メモリに保存される。
  - 学習を行う際は、メモリに保存したデータからランダムにサンプリングしたデータを用いる。

Q学習における損失の定義

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right)^2\right]$$

目標値(教師信号)　NNの出力

体験再生を適用した場合の損失の定義

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta)\right)^2\right]$$

目標値(教師信号)　NNの出力

ランダムサンプリングされたデータを用いて学習する

Target Q-Networkの固定

- DQNでは、 「価値関数が小さく更新されただけでも選ばれる行動が大きく変わってしまう」という問題に対して、Target Q-Networkを固定した。
- [対象としている問題点]
  - 学習の目標値(教師信号)算出に用いるネットワークと、行動価値Qの推定に用いるネットワークが同じ場合、行動価値関数を更新すると目標値(教師信号)も変化してしまい、学習が不安定になる。
- [Target Q-Network固定の仕組み]
  - 一定期間の学習の間、目標値の計算に用いる行動価値関数ネットワーク(Target Q-Network)のパラメータを固定し、一定周期でこれを更新することで学習を安定させる。
  - Target Q-Networkを固定した場合の損失の定義

目標値(教師信号)　　NNの出力

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta)\right)^2\right]$$

過去のある時点のパラメータθ^{-}を使って行動価値関数Qが最大になる行動aを求める

報酬のクリッピング

- DQNでは、 「報酬のスケールが与えられたタスクによって大きく異なる」という問題に対して、報酬のクリッピングという方法を導入した。
- [対象としている問題点]
  - ゲームの種類によって得点の範囲が異なるため、報酬のスケールがゲームによって異なる。このままだと、学習が不安定になる。
- [報酬クリッピングの仕組み]
  - 報酬の値を [-1, 0, 1] の3通りに制限することで学習を安定させる。
  - 報酬の大小を区別できなくなるデメリットはあるものの、学習が安定するメリットの方が大きい。

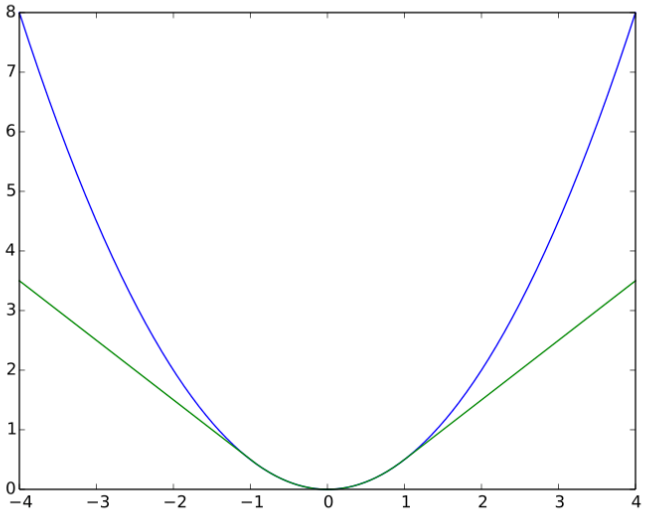
Huber損失

- DQNでは、損失算出にHuber損失を用いている。
- [対象としている問題点]
  - 誤差が大きい場合に二乗誤差を使用すると誤差関数の出力が大きくなりすぎて学習が安定しづらい。
- [DQNで使用されているHuber損失]

$$L(\theta) = \frac{1}{2}a^2, \qquad |a| \leq 1$$

$$L(\theta) = |a| - \frac{1}{2}, \qquad \text{上記以外}$$

a：誤差

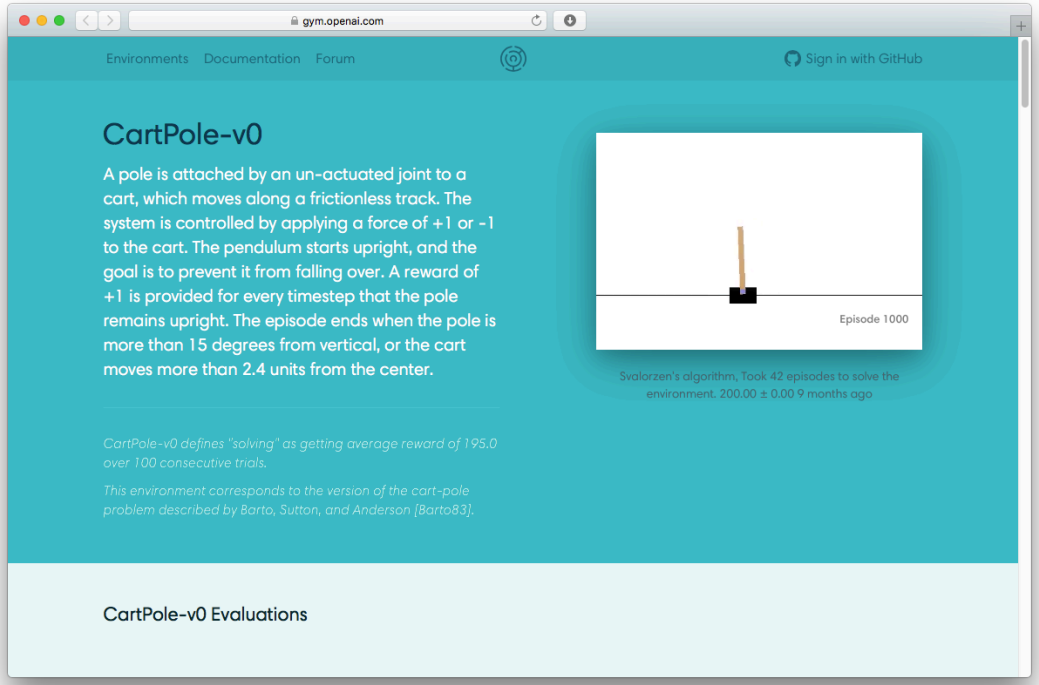


ー Huber損失
ー 二乗誤差損失

カートポール問題

カートポール問題

- ここでは、強化学習の例題としてよく用いられるカートポール問題を紹介する。
- 参考：https://gym.openai.com/envs/CartPole-v1/



強化学習を行うにはシミュレータが必要

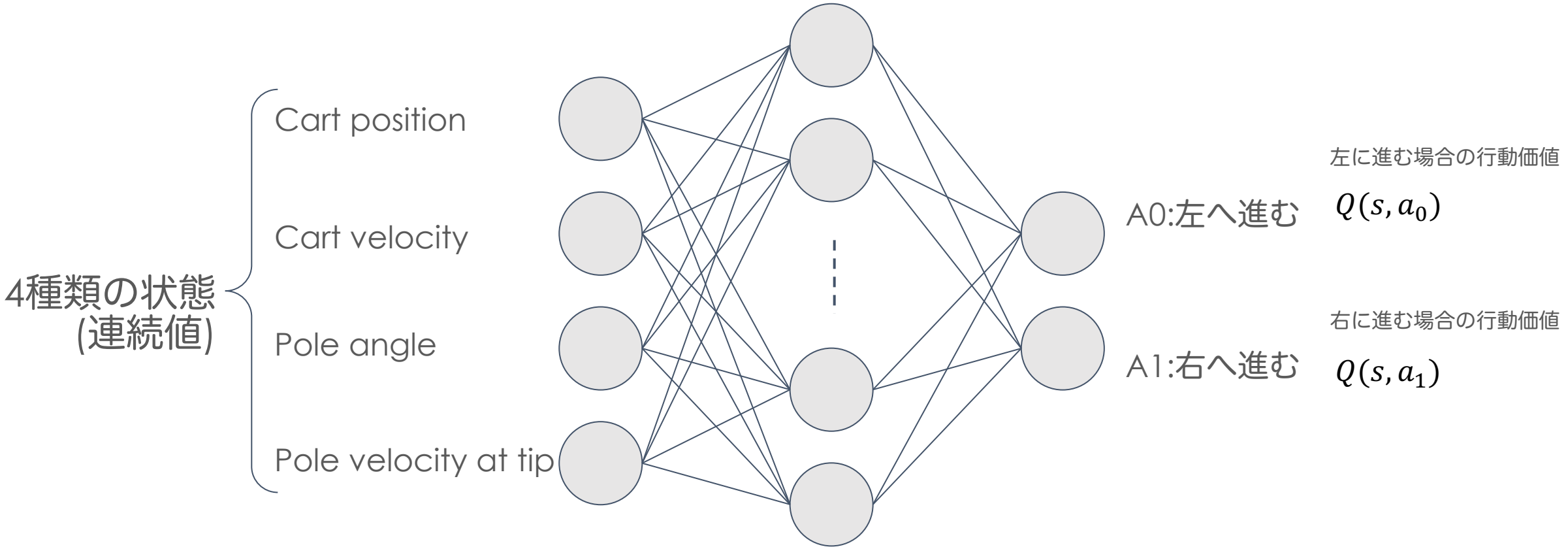
- 強化学習を行うには、制御対象となる実機もしくは制御対象を模したシミュレータが必要。
- シミュレータとしては、OpenAI Gymなどがある。
- OpenAI Gym
  - 古典的な問題からクラシックなゲーム、3Dゲームに至るまで、様々なケースで強化学習アルゴリズムを試せるよう共通のインターフェイスを整備して提供している、オープンソースの開発ツールキット。
  - https://gym.openai.com/
  - https://github.com/openai/gym

OpenAI Gymのカートポールシミュレータにおける問題設定

- カートの上のポールが倒れてしまわないよう（ここでは15度以上傾かないよう）、傾き始めた時にカートを左右へ動かすことで、ポールのポジションを補正する。
- ポールが一定以上傾く、カートが一定距離以上初期位置から動く、200タイムステップ一定以内の傾きを維持する、のいずれかで、1 エピソードが終了する。
- 傾きを維持している間、報酬が +1 されていき、最大200 タイムステップで +200 の収益を得られる。

行動価値関数Qを近似するためのニューラルネットワーク例

- カートポール問題において、行動価値関数(Q関数)を学習させるニューラルネットワークの構成例



カートポール問題の状態と行動

- 状態と行動の種類と意味

状態

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

行動

Actions

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

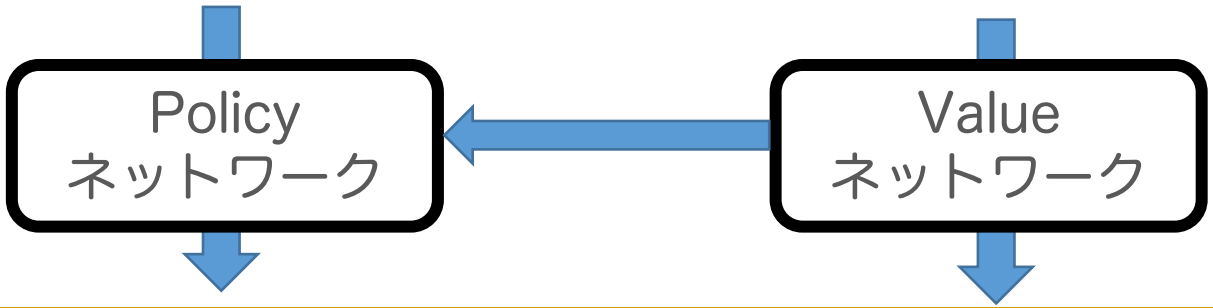
kerasを使った計算例

- 本講義では、時間の都合上カートポール問題のNotebook演習を行わないが、インターネットで検索すれば、kerasを使ってカートポール問題を解いている例が多数ヒットするので、興味のある方はそちらを参照されたい。
- kerasを使った実装例
  - https://github.com/keras-rl/keras-rl
  - https://github.com/keras-rl/keras-rl/blob/master/examples/dqn_cartpole.py

AlphaGO

AlphaGo

- GoogleのDeepMind社が開発した囲碁AI。
- 碁石を画素としたCNNベースのネットワーク。
- 盤面から優劣を推定するValueネットワークと、次の手を生成するPolicyネットワークで構成される。
- いずれのネットワークもはじめにプロの棋譜で教師あり学習、その後強化学習という転移学習ベースの方策。
  - Policyネットワークの強化学習では、方策勾配法が用いられている。
- 論文
  - David Silver et al. Mastering the game of Go with deep neural networks and tree search. https://www.nature.com/articles/nature16961



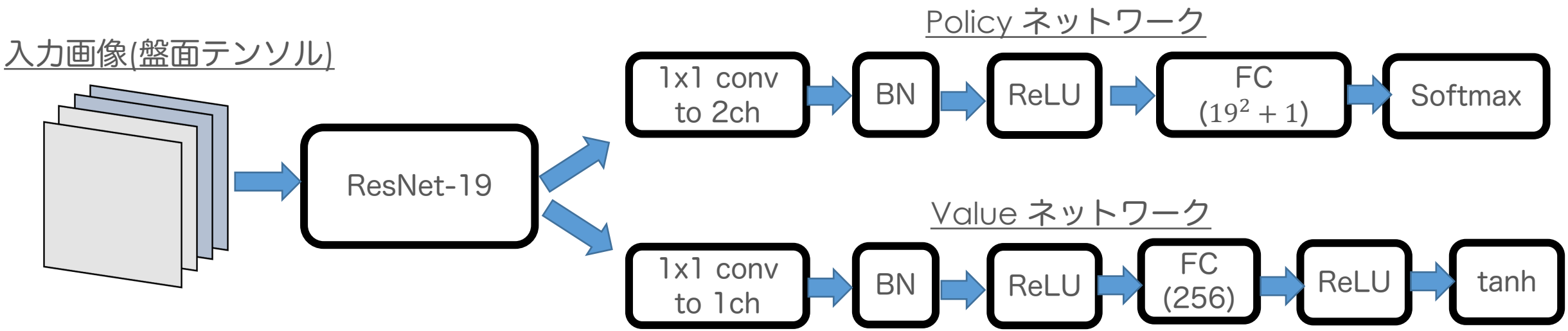
AlphaGoZero

- AlphaGoの後継 AlphaGoZero 。
- 世界チャンピオンを打倒した旧AlphaGoに100戦100勝0敗。
- AlphaGoはプロ棋士の打った手を学習していたが、Zeroは使わない。
  - 教師なしの囲碁AI
- 指し手を決めるPolicyネットワークと盤面を評価するValueネットワークの一部を共通化することで学習の効率をアップ。
- 論文
  - David Silver et al. Mastering the game of Go without human knowledge. (Nature 550) Oct. 2017.

- https://www.nature.com/articles/nature24270.epdf?author_access_token=VJXbVjaSHxFoctQQ4p2k4tRgN0jAjWel9jnR3ZoTv0PVW4gB86EEpGqTRDtpIz-2rmo8-KG06gqVobU5NSCFeHILHcVFUeMsbvwS-lxjqQGg98faovwjxeTUgZAUMnRQ
- https://deepmind.com/blog/article/alphago-zero-starting-scratch

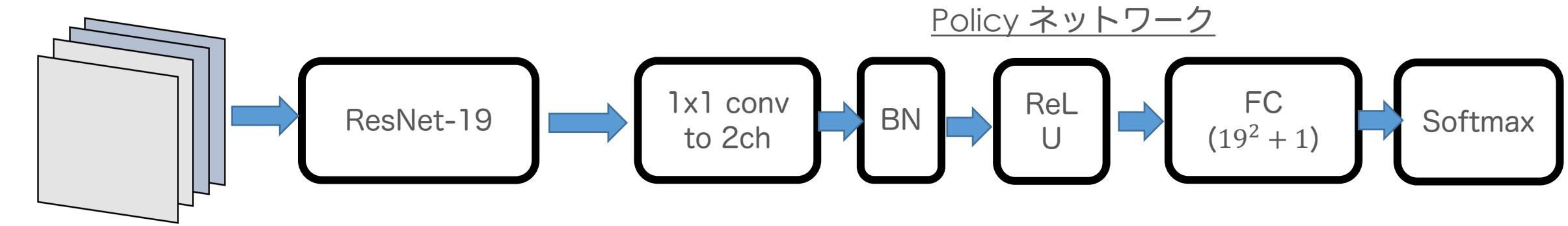
AlphaGoZero

- 盤面の状態を19x19の2値画像で表す。
- 入力画像のチャンネル数は17 = (白/黒)*(8手分の履歴) + (今どちらの手番かの真っ白 or 真っ黒画像) 。
- 256チャンネルのbottleneck-Residual BlockからなるResNet-19(19層のResNet)で盤面を処理。
- ResNetの中間表現からPolicy側とValue側にネットワークを分岐。



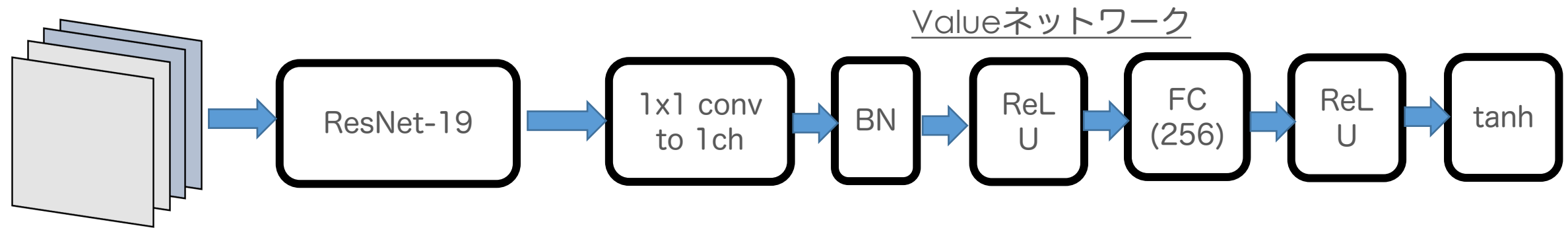
Policyネットワーク

- 次にどの手を指すか、を決定するネットワーク。
- 出力は(19^2+1)次元の確率ベクトル$\boldsymbol{p}$で、どこに石を置けば良いかを表す。
- 基本的に石が置ける場所の中で$\arg\max_i p_i$を選択する。



Valueネットワーク

- 今の盤面がどの程度優勢か、を推定するネットワーク。
- ResNetの出力を全結合層に通して自分が勝つ見込みを[-1,1]で出力。
- AlphaGoZero全体は $(\boldsymbol{p}, v) = f(\boldsymbol{s}; \Theta)$とかける。
  - $\boldsymbol{p}$:ポリシー , v: 勝率(value), $\boldsymbol{s}$: 盤面, Θ: モデルのパラメータ



AlphaGoZeroの学習(概観)

探索木とは？
現在の状態から行動の選択を繰り返した時の状態遷移を表現するもの。状態がノード、行動がエッジに対応する。ボードゲームなどで手を読む際に使う。

- 2人のプレイヤーを戦わせて強くしていく(自己対戦) 。
- 対戦する際は、モンテカルロ木探索 (Monte-Carlo Tree Search, MCTS)を用いて、次の手を決めていく。
  - MCTSでは、探索木を用いて、取りうる行動群に対する行動確率を算出する。
  - 行動確率が求まれば、次の手を決めることができる。
  - MCTSでやっていることは、人間における「先の手を読む」行為に相当する。
- MCTSを行う際に、 PolicyネットワークとValueネットワークを用いる。
  - Policyネットワークの出力結果を用いてノードを追加していく。
  - Valueネットワークの出力結果を用いて、あるノードの勝率を推定する。あるノードの勝率が高いと、そのノードに至る行動の行動確率が高まる。
- n回対戦したら、 PolicyネットワークとValueネットワークの重みを更新する。

AlphaGoZeroの学習(概観)

- Policyネットワークの学習
  - 入力：局面の状態$\boldsymbol{s}$
  - 出力：次に選ぶ手の確率$\boldsymbol{p}(\boldsymbol{s})$
  - 正解：勝ったプレイヤーがとった手
  - 勝ったプレイヤーがとった手と同じ手を出力できるようにパラメータを更新していく

(ここだけ見ると教師あり学習)

- Valueネットワークの学習
  - 入力：局面の状態$\boldsymbol{s}$
  - 出力：最終局面において勝つ見込み[-1,1] $v(\boldsymbol{s})$
  - 正解：勝者側を1、敗者側を-1
  - 平均2乗誤差を最小にするようにパラメータを更新していく

(ここだけ見ると教師あり学習)

深層強化学習の実用面での課題

深層強化学習の実用面での課題

- 深層強化学習は、ゲームAIなどでいくつかも成果が発表され、近年非常に注目されているが、解決すべき課題はまだ多い。
- 例えば、
  - 試行錯誤を行うため学習に時間がかかる。(サンプル効率が悪い)
  - 報酬の設計が困難である。
  - 報酬が非常にまれにしか得られない環境では、学習が進みづらい。
  - 環境(状態遷移を決める部分) を用意する必要がある。
- 深層強化学習の課題に関しては、以下のブログも参考になる。
  - Alex Irpan, Deep Reinforcement Learning Doesn't Work Yet, https://www.alexirpan.com/2018/02/14/rl-hard.html

逆強化学習

強化学習の課題　報酬定義

- 強化学習では、分析者が報酬を定めて、方策を学習させる。
- しかし実際のところ報酬を適切に定めるのはかなり難しい。
  - ゲームの場合、スコア or 勝ち負けなど指針がたくさんあるが…
  - 歩行ロボットの場合、「膝と腰に負担をかけない歩き方」に対する報酬をどのように定めるべきか？

逆強化学習

- 逆強化学習(Inverse Reinforcement Learning; IRL)は、模倣学習の方法論の一つ。
  - 報酬の与え方(報酬関数)を獲得することが目的。
  - 獲得した報酬の与え方(報酬関数)は、似たようなタスクに転用できるかもしれない。
    - 例、自転車に乗るタスクで報酬関数を獲得し、それを一輪車に乗るタスクで利用する。
- 模倣学習(imitation learning)とは、エキスパートの行動を模倣するよう学習することの総称。
  - 模倣学習にGANを利用した例：GAIL。
    - 識別器は、生成器が生成した方策がエキスパートのものかを識別する。
    - Jonathan Ho, Stefano Ermon . Generative Adversarial Imitation Learning.(NIPS 2016) Jun. 2016

逆強化学習

- 一般的な逆強化学習の手順
  1. エキスパートの行動を取得し、方策を定義する。
  2. 報酬関数の初期化を行う。
  3. 報酬関数を利用し方策を学習する。
  4. 学習した方策が、エキスパートの方策(手順1)と近しくなるように報酬関数を更新する。
  5. 手順3に戻る

転移学習

目次

転移学習

転移学習

- 転移学習(transfer learning)とは、ある問題設定で学んだことを別の問題設定の汎化性能向上に役立てること。
- 異なる問題設定間において、共通の特徴量を仮定できるとき、転移学習は有効である。
- 参考：『深層学習, Ian Goodfellow, KADOKAWA, p.392』
- 転移学習の極端な例
  - ワンショット学習
  - ゼロショット学習

<参考>
転移学習に似た言葉として再学習(fine tuning)という言葉がある。再学習という言葉の意味は、明確に定められていないが、以下のようなことを指すことが多い。
- 学習済みモデルのパラメータのうち入力層から出力層の数層手前までのパラメータをそのまま利用し、これに新しい層を結合して、その新しい層のパラメータを更新すること。
- 教師なし学習により事前に学習されたパラメータを初期値にして、教師あり学習によってそのパラメータを更新すること。

ワンショット学習

- 転移学習の極端な形式として、ワンショット学習(one shot learning)がある。
- 転移先のタスクにおいてはラベル付き事例が1つだけ与えられる。
- ワンショット学習の事例
  - Siamese Neural Networks for One-shot Image Recognition, Gregory Koch, Richard Zemel , Ruslan Salakhutdinov,
  - https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

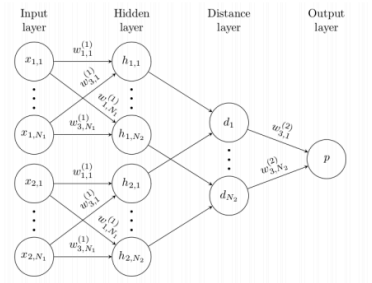


Figure 3. A simple 2 hidden layer siamese network for binary classification with logistic prediction p. The structure of the network is replicated across the top and bottom sections to form twin networks, with shared weight matrices at each layer.

<この論文で提案された手法の概要>
- 2つの画像が同じクラスであるかを予測するモデルを構築しておく。
- これに、新しいクラスの画像(下記の例ではバスケットボール)を1枚だけ学習させる。
- すると、次からバスケットボール画像のクラス分類ができるようになる。

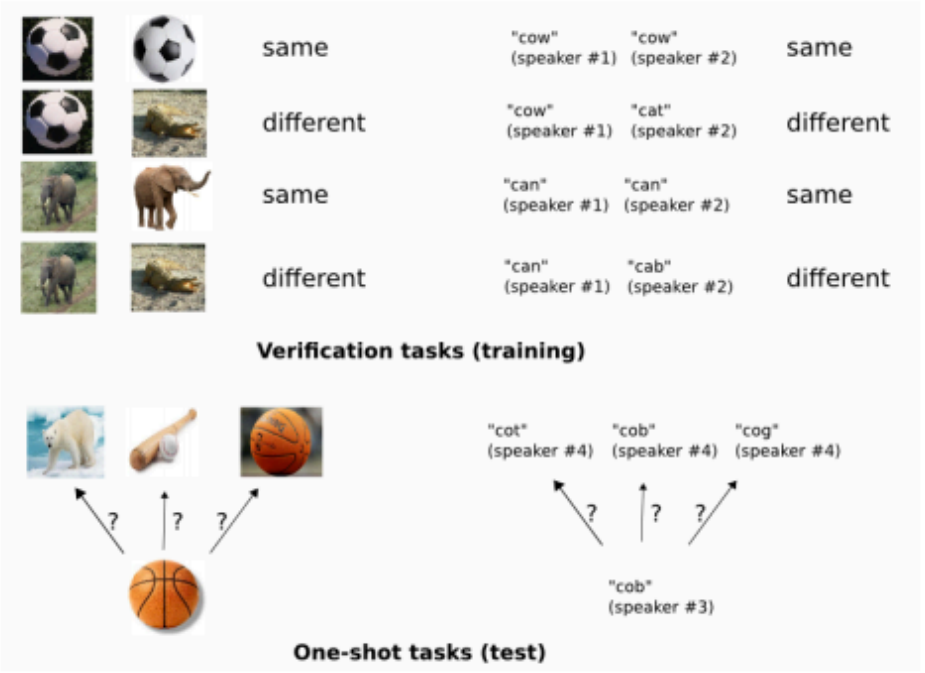


Figure 2. Our general strategy. 1) Train a model to discriminate between a collection of same/different pairs. 2) Generalize to evaluate new categories based on learned feature mappings for verification.

ゼロショット学習

- 転移学習の極端な形式として、ゼロショット学習(zero shot learning)がある。
- 転移先のタスクにおいてはラベル付き事例が与えられない。
- ゼロショット学習の事例
  - Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation, Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat
  - 参考ブログ：https://ai.googleblog.com/2016/11/zero-shot-translation-with-googles.html

Google Neural Machine Translation (GNMT)を用いた事例

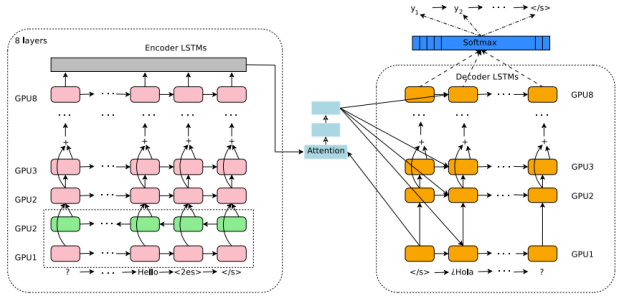
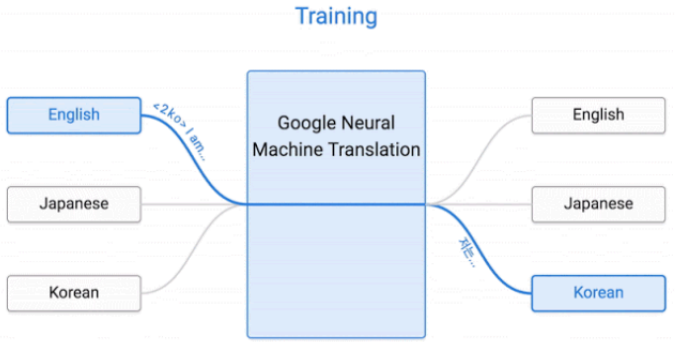


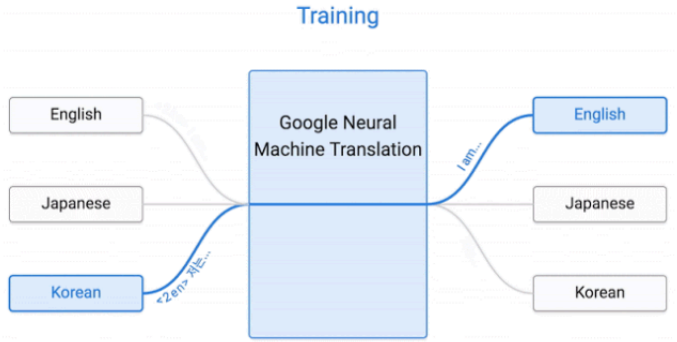
Figure 1: The model architecture of the Multilingual GNMT system. In addition to what is described in [24], our input has an artificial token to indicate the required target language. In this example, the token “<2es>” indicates that the target sentence is in Spanish, and the source sentence is reversed as a processing step. For most of our experiments we also used direct connections between the encoder and decoder although we later found out that the effect of these connections is negligible (however, once you train with those they have to be present for inference as well). The rest of the model architecture is the same as in [24].

ゼロショット学習

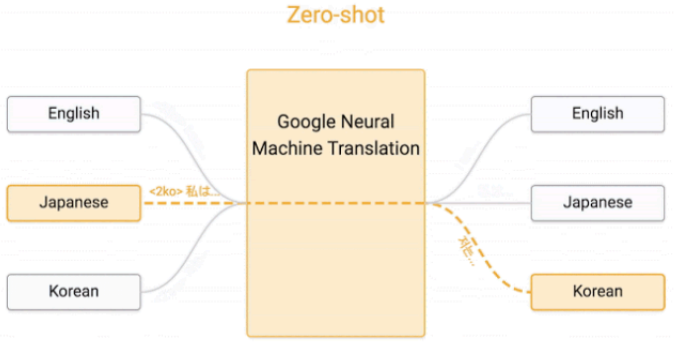
英語->韓国語を学習



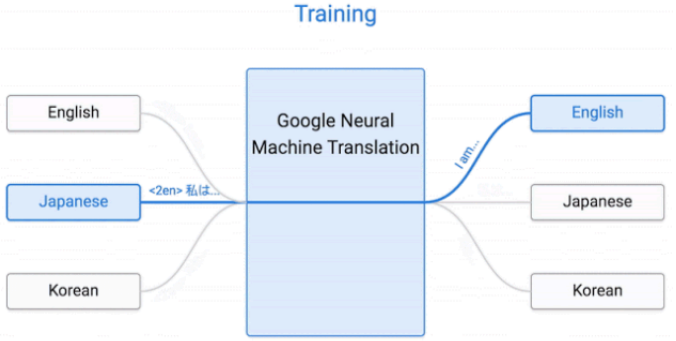
韓国語->英語を学習



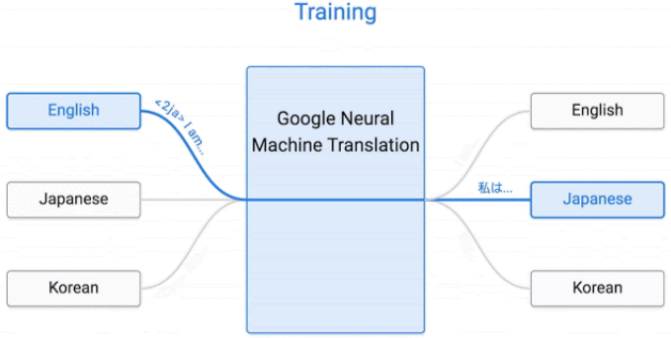
学習したことがない
日本語->韓国語も翻訳できる
ようになった



日本語->英語を学習



英語->日本語を学習



原著論文では、中間言語ようなものを獲得できたと表現されている

Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation, Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat

軽量化技術

目次

軽量化技術

- 近年のDNNのモデルは、パラメータ数が多くなる傾向にある。
- パラメータ数が多くなると、その分、計算時間やメモリ使用量が増加することになる。
- ここでは、モデルを軽量にする技術として、蒸留、プルーニング、量子化を紹介する。
- 軽量化技術は、ハードウェアへ組み込む場合に活用できる技術として注目されている。

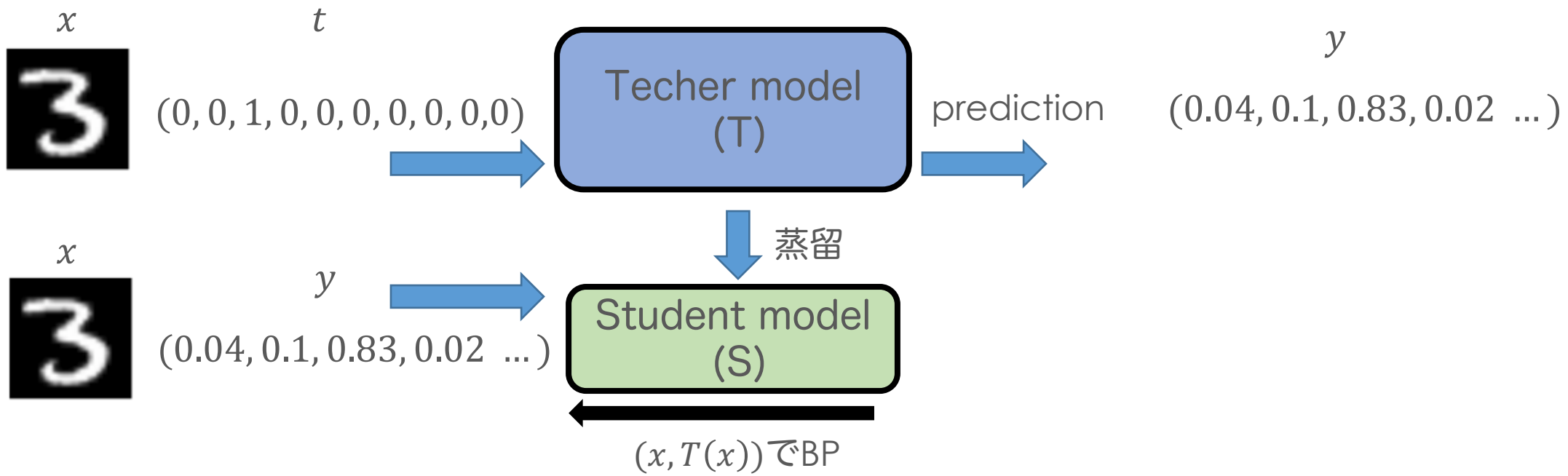
蒸留

蒸留

- 蒸留(distillation)とは、DNNのモデルを小さくして、省メモリ化や高速化を図りたいときに使う方法。
- 基本方針は「大きなモデルの動作を真似させる」という非常にシンプルなアプローチ。
  - 学習済みの大きなDNNの持っている知識を小さなDNNに移す。
  - これを「知識の蒸留(knowledge distillation)」と呼んでいる。
- 原著論文
  - Geoffrey Hinton, Oriol Vinyals, Jeff Dean ,Distilling the Knowledge in a Neural Network,(NIPS 2014) Oct. 2014
- 派生手法もたくさんあり、モデル圧縮の基本となる考え方のひとつ。

蒸留の基本的考え方

- 学習済みの大きなモデルをTeacher(T), 蒸留先の小さなモデルをStudent(S)とする。
- Studentを学習させるとき、本当の教師ラベルを正解にするのではなく、Teacherの出力を正解にする。



ソフトラベリング

- ワンホットな教師ラベルはハードラベリングと呼ばれる。
  - ラベル間の関連情報を持っていない。
- これに対し、十分学習されたTeacherのソフトマックス出力はソフトラベリング。
  - モデルにとって「どれとどれが近いか」の情報があるため、ハードラベリングよりも上手く学習できると考えられている。
    - 例えば、十分学習されたTeacherのソフトマックス出力では、ネコの値とトラの値は、イヌの値とトラの値よりも近くなるはずである。

蒸留の悪用

- 蒸留という方法を用いると、隠蔽されたモデルを複製できることがあり、学習済みモデルの権利保護という観点で、対応策が課題になっている。
- 例、
  - ある事業者が学習済みモデルを用いた機械学習APIを提供している。
  - 学習済みモデルはサーバー上にあるため、API利用者は直接学習済みモデルにアクセスできない。
  - 学習済みモデルを不正に複製したいと思ったAPI利用者は、何度もAPIを利用し、入力と出力(ソフトラベリング)のデータセットを作り、手元のモデルで学習を行ってみた。
  - すると、元の機械学習APIと同等精度のモデルが出来上がってしまった。

プルーニング

プルーニング

- プルーニング(pruning, 剪定)とは、精度に大きく影響を与えないノードや重みを刈りとることによって、データ量と計算負荷を削減する方法。
- プルーニングの例
  - 値が0に近い重みを消していく手法
    - https://arxiv.org/pdf/1506.02626.pdf
  - 各ユニットの出力に重みをかけるというモデルに対し、その各ユニットの重みをL1正則化付き学習で潰す方法
    - https://arxiv.org/pdf/1708.06519.pdf
  - 各ユニットの出力にバイナリ値をかけるというモデルに対し、バイナリ値を進化戦略で最適化する方法
    - https://arxiv.org/pdf/1907.06341.pdf

プルーニングの事例

- 論文
  - Song Han, Jeff Pool, John Tran, William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. https://arxiv.org/pdf/1506.02626.pdf

プルーニングの手順
(値が0に近い重みを削除していく)

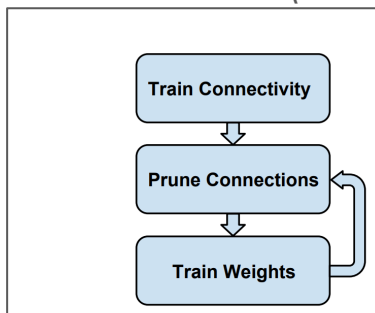


Figure 2: Three-Step Training Pipeline.

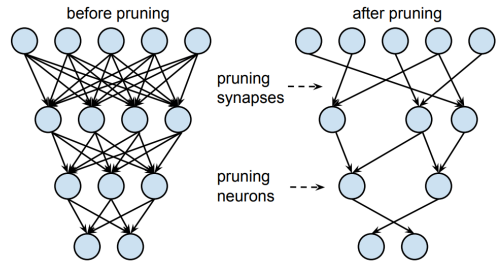


Figure 3: Synapses and neurons before and after pruning.

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
Baseline Caffemodel [26]	42.78%	19.73%	61.0M	1×
Data-free pruning [28]	44.40%	-	39.6M	1.5×
Fastfood-32-AD [29]	41.93%	-	32.8M	2×
Fastfood-16-AD [29]	42.90%	-	16.4M	3.7×
Collins & Kohli [30]	44.40%	-	15.2M	4×
Naive Cut	47.18%	23.23%	13.8M	4.4×
SVD [12]	44.02%	20.56%	11.9M	5×
Network Pruning	**42.77%**	**19.67%**	**6.7M**	**9×**

この論文では、AlexNetを使って、ImageNetのタスク(写真を1000カテゴリから分類する)を剪定しながら学習させたところ、標準モデル(baseline)と同等の精度を実現できたと報告している。容量は1/9。
TOP-5 Errorとは、予測時の確率が大きかった上位5つの中に、正解が無い割合。

量子化

量子化

- 量子化(quantization)とは、通常、32bitもしくは16bit精度の浮動小数点で表現されるニューラルネットワーク内のデータを、それより少ないビット数で表現することにより、計算の高速化や省メモリ化を図ること。
- 1bitで表現することを特に2値化(binarization)という。
- 通常、量子化の対象となるデータは、パラメータ、アクティベーション(中間層の出力)、勾配である。
- 学習済みモデルを何らかのハードウェアに組み込む場合は、そのハードウェアの仕様を基に削減後のビット数を決めたりする。

量子化の事例

- 論文
  - Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. https://arxiv.org/pdf/1511.00363.pdf
  - 順伝播計算と逆伝播計算において、パラメータを2値化することによって軽量化を図っている。
  - 精度を維持したまま、計算時間が1/3、メモリ容量が1／16。

Method	MNIST	CIFAR-10	SVHN
No regularizer	1.30 ± 0.04%	10.64%	2.44%
BinaryConnect (det.)	1.29 ± 0.08%	9.90%	2.30%
BinaryConnect (stoch.)	1.18 ± 0.04%	**8.27%**	2.15%
50% Dropout	1.01 ± 0.04%		
Maxout Networks [29]	0.94%	11.68%	2.47%
Deep L2-SVM [30]	**0.87%**		
Network in Network [31]		10.41%	2.35%
DropConnect [21]			1.94%
Deeply-Supervised Nets [32]		9.78%	**1.92%**

BinaryConnect(det.):確定的に2値化する方法
BinaryConnect(stoch.):確率的に2値化する方法

高速化技術

目次

GPU

GPU

- GPU(graphics processing unit)は、計算機における画像処理を主に担当するプロセッサ。
- 特に、3次元画像の描画を高速化する形で発展してきた。
- 3次元画像の描画は並列に行える計算が多いため、GPUは並列に計算を行うことが得意。
  - 特に、行列演算が得意。
- GPUの高い性能を計算用途に利用しようとする動きが2000年代に出てきた。
  - この技術は、GPGPU(general purpose computing on graphics processing units)と呼ばれる。
- 処理の高速化を支援するハードウェアを総称してアクセラレータと呼ぶことから、GPUアクセラレータと呼ばれることがある。
- 深層学習では、最適化のための膨大な時間が必要になるため、GPUのような効率の良い計算手段が求められるようになってきた。

GPU

- CPUが利用するメモリのことをメインメモリ(Random Access Memory, RAM)と呼ぶのに対し、GPUが利用するメモリのことをビデオメモリ(Video RAM, VRAM)と呼ぶ。
- 深層学習において、GPUを利用する際、GPUのビデオメモリが利用可能メモリの上限として気をつけなければならない。
  - NVIDIA社のTesla P100というGPUのビデオメモリは、16GB または 12GB。
- 大量のコアに対して、計算対象のデータを一度に大量に供給する必要があるため、メモリの読み込みが高速であり、CPUよりもメモリ帯域幅が広い。

GPU

- GPUは、CPUに比べはるかに多くのコアを備えており、大量のコアを使って、一度に大量の演算ができる設計になっている。
  - CPUでは、多くても数個から数十個のコア数。
  - GPUでは、数百から数千のコア数。
- ただし、各コアの単体性能は、CPUのそれよりも低いことに注意。
  - GPUの各コアは、動作周波数が低く、CPUが備えている多くの高速化機能を持たない。
- GPUのコアは、それぞれ独立に制御できず、複数のコアに対して、同じ命令を実行する必要がある。
  - 単一の命令に対して、異なるデータを処理させる形式のことをSIMD(single instruction multiple data)と呼ぶ。
  - このため、なるべく同じ命令を実行するように、プログラムを組む必要がある。
  - 深層学習では、フレームワークがやってくれているので、ユーザーは意識する必要はない。

GPU

- GPUでは、PCI Expressなどの拡張デバイスを利用して、 CPUとデータのやり取りを行う。
- 通常、 PCI Expressを用いた通信は、GPU-ビデオメモリ間の通信よりも遅くなる。
  - NVIDIA社のTesla P100というGPUでは、PCI Expressデータ転送帯域は23GB/secであり、ビデオメモリ帯域は730GB/secである。
- このため、複数のGPUを使う場合、通信速度が計算効率のボトルネックになる。

GPU

- CPU、GPUに関わらず、深層学習のような数値計算では浮動小数点による計算を行う。
  - 16ビット(半精度浮動小数点数という)
  - 32ビット(単精度浮動小数点数という)
  - 64ビット(倍精度浮動小数点数という)
- GPUでは、単精度の計算しか行えない製品が多いが、半精度を利用する製品もある。
  - 精度を落とす場合、オーバーフローが起こりやすくなることに注意する必要がある。

GPUに関する参考文献

- 『深層学習による自然言語処理, 坪井,海野,鈴木,講談社, 7章』

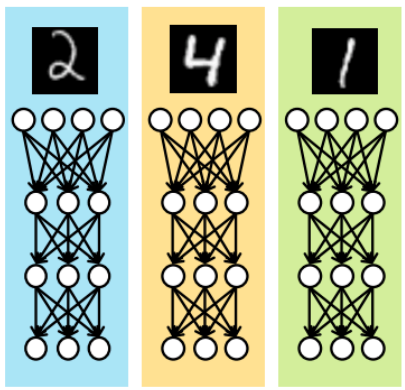
分散深層学習

分散深層学習

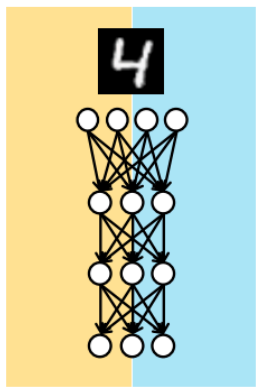
- 深層学習では、単一の計算機上で利用できる計算資源では不十分ということがある。
- その場合、多数の計算機を利用し、計算負荷を分散したくなる。
- 入力データを分割して、複数の計算機で分担する方法をデータ並列処理(data parallelism)という。
  - 学習済みモデルを用いて、予測(推論ともいう)する際は、この考え方により、簡単に分散処理が実現できる。
- 複数の計算機が1つのデータに対して共同で動作し、それぞれがモデルの異なる部分を実行することをモデル並列処理(model parallelism)という。
  - 訓練時、予測時(推論時)の両方で実現可能。
  - 高解像度画像を入力とするCNNなど、1プロセス上に載りきらないサイズのモデルを訓練したい場合などに用いられる。
- 参考文献
  - Ian Goodfellow, 深層学習, 12章

分散深層学習

- 訓練時におけるデータ並列処理には、同期型と非同期型がある。
- ChainerMNでは、同期型でデータ並列処理を行う。
  - Takuya Akiba, Keisuke Fukuda, Shuji Suzuki. ChainerMN: Scalable Distributed Deep Learning Framework.
    http://learningsys.org/nips17/assets/papers/paper_25.pdf



(a) Data parallelism.



(b) Model parallelism.

Figure 1: Data parallelism and model parallelism.

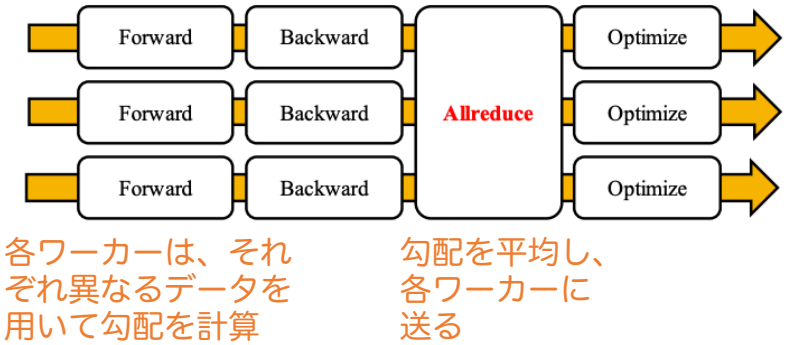


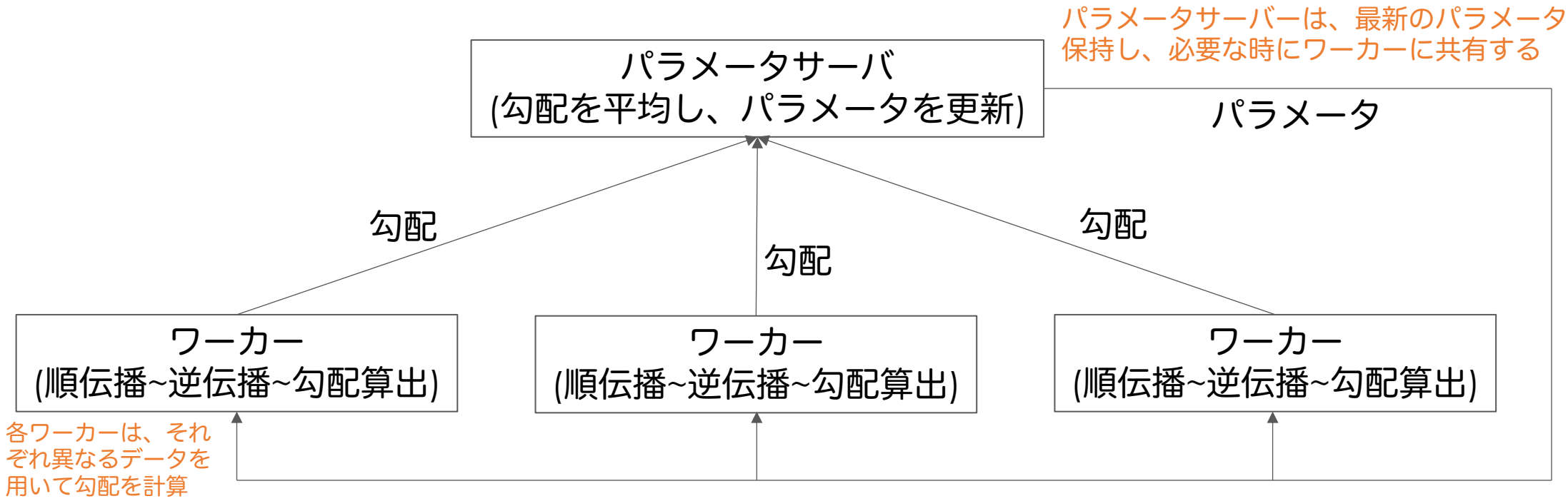
Figure 2: The four steps that constitute an iteration of synchronous data parallelism.

各ワーカーが求めた勾配の平均値(平均勾配)を求め、それを用いて各ワーカーのモデルパラメータを更新する。これにより、常に最新の平均勾配を用いて訓練を行うことができる。
全ワーカーは常に同じパラメータを持っている。

分散深層学習

同期を取らずに平均勾配が更新されるため、パラメータサーバー上に古い劣悪な勾配が残っていると、パラメータを悪化させてしまう事がある。 このような古い勾配を「陳腐化した勾配（stale gradient）」と呼ぶ。陳腐化した勾配の影響を緩和する方法としては、学習率を下げる、陳腐化した勾配を定期的に捨てる、ミニバッチサイズを調整する、などが考えられる。

- TensorFlowでは、分散学習のライブラリがいくつか用意されており、その中のParameterServerStrategyを用いると、非同期にデータ並列処理を実行できる。
- https://www.tensorflow.org/guide/distribute_strategy#parameterserverstrategy



分散深層学習

- 同期型の良さ
  - 陳腐化した勾配（stale gradient）が発生しない。
  - 一般に、収束性が良いと言われている。
  - 以下の論文では、同期更新の方が非同期更新よりも収束速度や正解精度が良いと報告されている。
    - Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, Rafal Jozefowicz. Revisiting Distributed Synchronous SGD. https://arxiv.org/pdf/1604.00981.pdf
- 非同期型の良さ
  - 1つのワーカーが落ちても計算を継続できる。
  - 他のワーカーを待たなくて良いので、待ち時間が発生しない。

Any Questions?

[グループワーク] 強化学習の基本的概念

- 以下の強化学習タスクに対し、7つの概念に対応するものを設定してみましょう。
  - 迷路問題、カートポール問題、ロボット掃除機、車の自動運転、その他(自由に)
- 2~3名のグループに分かれて、上記課題に取り組みましょう。(20分)
- 最後に、代表の1グループに発表していただきます。 (15分)

将棋AIの例

概念	将棋AI
行動	例、「歩」を前へ移動、「角」を右斜め前へ3つ分移動し裏返す
エージェント	棋士
方策(ポリシー)	駒の並び方に応じて最適な駒の動かし方を決める部分
環境(状態遷移先を決める部分)	盤面+相手(人間もしくはコンピュータ)
報酬	例、勝ったら1点
状態	駒の並び
エピソード	1局

[グループワーク] 強化学習手法の比較

- Notebook演習で確認したSarsa、Q学習、アクター・クリティック法の3つの手法を比較し、それぞれの特徴をつかみましょう。
- Notebookにおいて、grid条件、エージェントの初期位置、epsilon、alpha、報酬の与え方、などを変更した計算を行い、結果を比較し、その結果になる理由について考察を行いましょう。
- 2~3名のグループに分かれて、上記課題に取り組みましょう。(20分)
- 最後に、グループごとに発表していただきます。 (15分)

対面講義でこのグループワークを円滑に行うため、
予習の段階で必ずNotebook演習を行なって来てください。

講座の時間が余ったら

- 今回の復習をします。