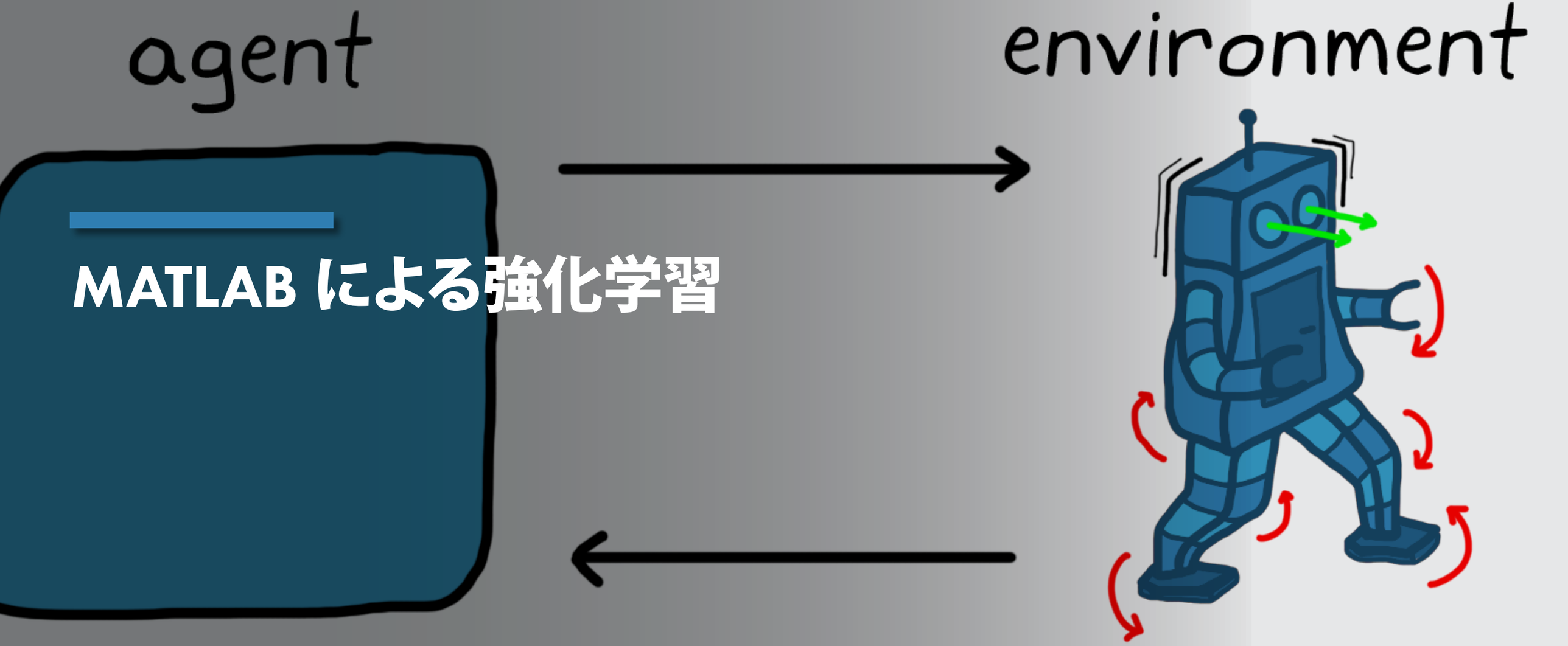
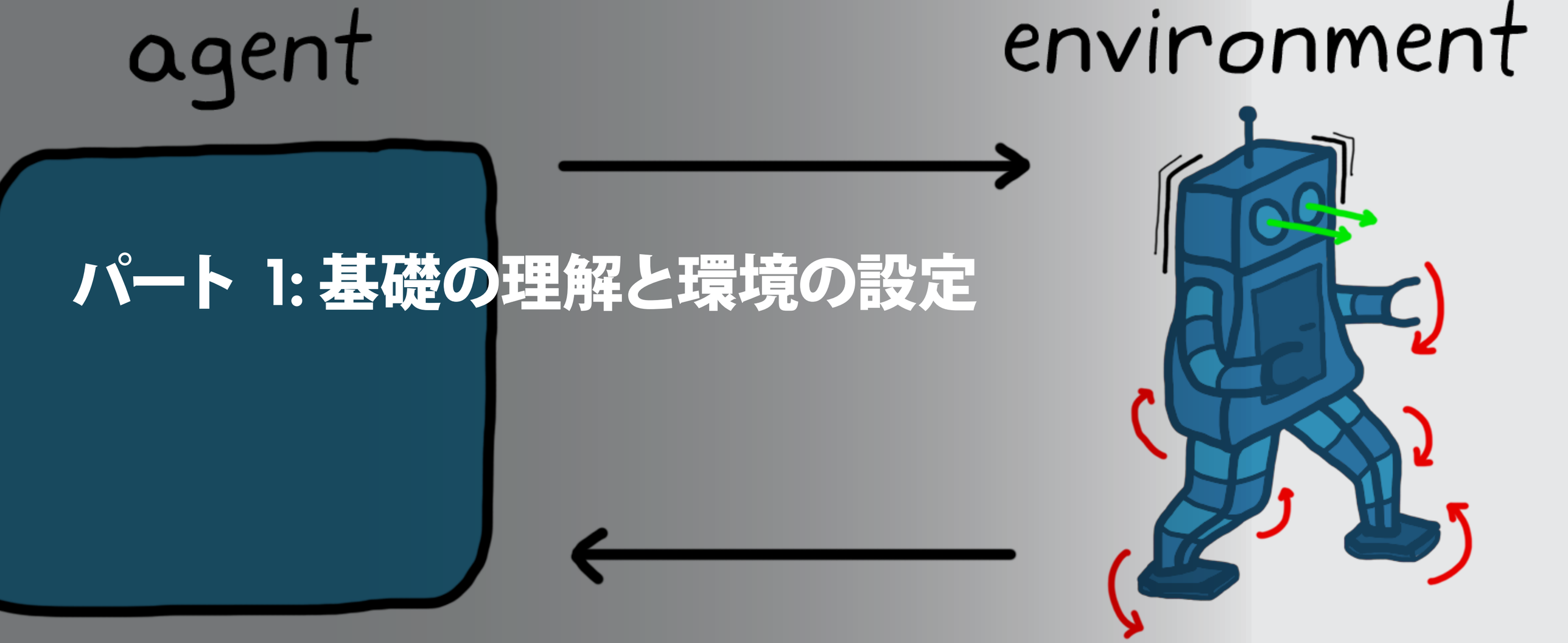


MATLAB による強化学習

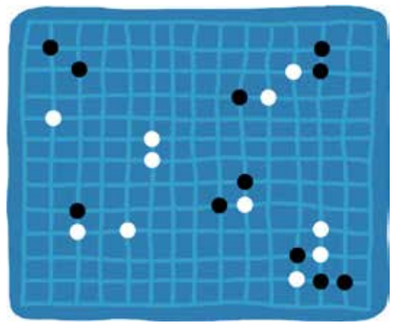
目次

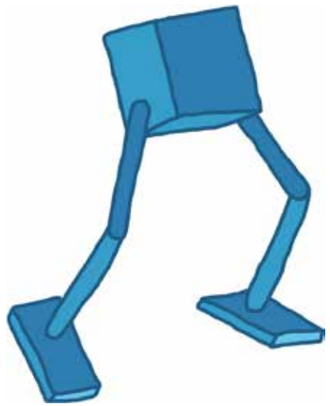


パート 1: 基礎の理解と環境の設定

強化学習とは

> *強化学習とは、数値的な報酬信号の最大化を目的として、何をすべきか、アクションに対して状況をどのように対応付けるかを学習することです。学習器に実行するアクションは指示されません。その代りに、試行を通じて最大の報酬が得られるアクションを見つけ出す必要があります。*
>
> — Sutton および Barto、*強化学習: 概要*





強化学習 (RL) は、コンピューター プログラムをトレーニングすることで、ゲームで人間の世界チャンピオンに打ち勝つレベルに到達させることができました。

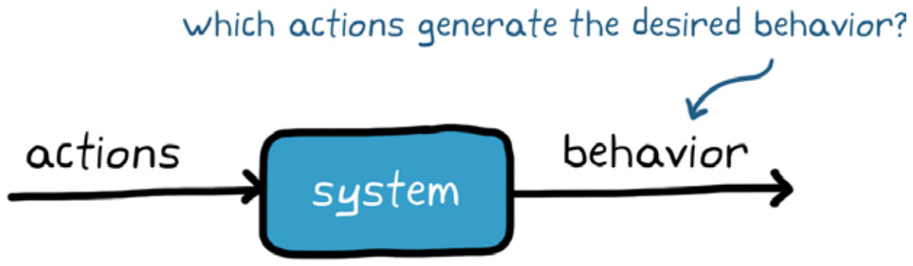
これらのプログラムは、大きな状態空間と行動空間、不完全な世界の情報が存在し、短期間のアクションが長期間の結果にどのような結果をもたらすかが不確実なゲームにおいて、最良のアクションを見つけ出します。

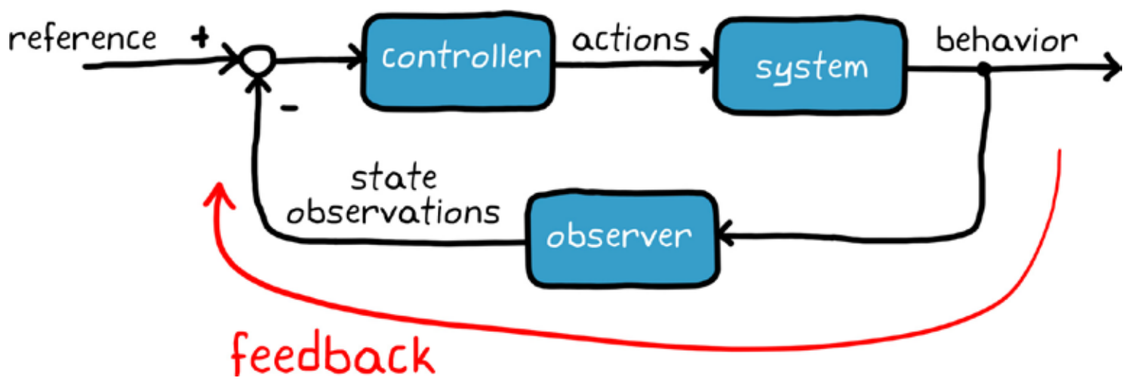
エンジニアは、実際のシステムのコントローラーを設計する際に、同様の課題に直面します。強化学習は、ロボットの歩行や、自律走行車の操縦などの複雑な制御問題の解決にも役立つでしょうか。

この eBook では、従来の制御の問題と関連づけて RL を説明してこの疑問に答え、RL 問題の設定方法と解決方法を理解できるよう支援します。

制御の目標

大まかに言って、制御システムの目標は、必要なシステム動作の生成に適した入力 (アクション) を判定することです。





フィードバック制御システムでは、コントローラーは状態観測を使用してパフォーマンスを向上し、ランダムな外乱とエラーを修正します。エンジニアは、そのフィードバックをプラントと環境のモデルとともに使用して、システム要件を満たすコントローラーを設計します。

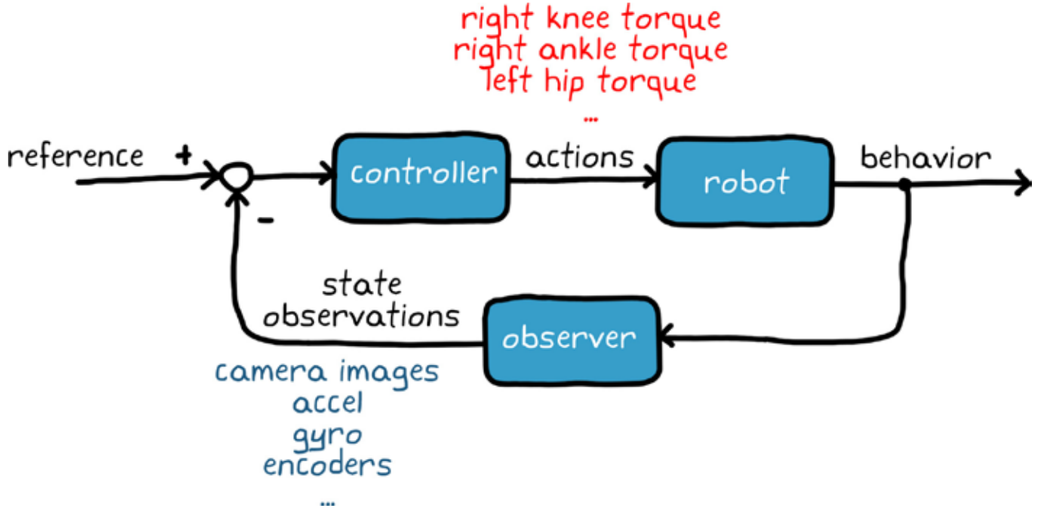
この概念は言葉にすると単純ですが、システムのモデル化が困難な場合、システムが著しく非線形である場合、または大きな状態空間と行動空間が存在する場合は実現が困難になります。

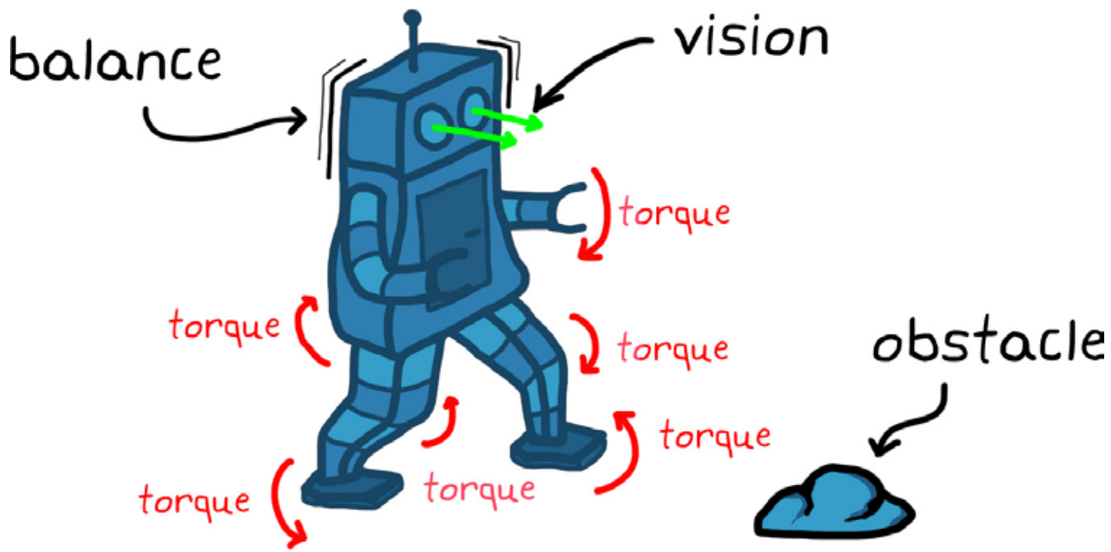
制御の問題

複雑性がいかに制御設計問題を面倒にするかを理解するには、歩行ロボットの制御システムの開発を思い浮かべてください。

ロボット (つまりシステム) を制御するには、腕や脚の各関節を動かす多数のモーターへの命令が必要になります。

各命令が実行可能な 1 つのアクションになります。状態観測は、カメラのビジョンセンサ、加速度計、ジャイロ、各モーターの符号化器などの複数のソースから発生します。





コントローラーは、次のような複数の要件を満たさなければなりません。

- ロボットの歩行およびバランスの保持の達成に適したモータートルクの組み合わせの判断
- 避けるべき障害物がランダムに存在する環境での動作
- 突風のような外乱の排除

制御システム設計では、これらに対応するだけでなく、急勾配の斜面やところどころ凍った場所を渡る際のバランス維持などの要件への対応も必要になります。

制御ソリューション

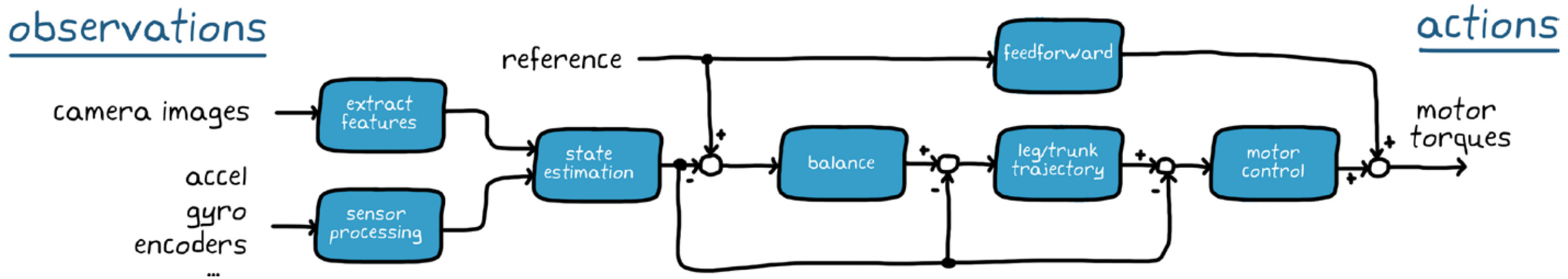
一般にこの問題へのアプローチに最適な方法は、問題を小さな個別のセクションに分割し、それらを個々に解決することです。

たとえば、カメラのイメージから特徴を抽出するプロセスを構築できます。これはたとえば、障害の場所や種類であったり、グローバルな基準フレームにおけるロボットの位置であったりします。これらの状態を、他のセンサーから得られた処理済みの観測情報と組み合わせて、全体の状態評価を完成します。

推定された状態と基準がコントローラーに供給されます。多くの場合これはネストされた複数の制御ループで構成されます。アウターループは概要レベルのロボット動作 (バランスの維持など) を管理し、インナーループは詳細レベルの動作と個々のアクチュエータを管理します。

すべての解決にはまだ至らない

ループは相互に作用するため、設計と調整を困難にします。また、これらのループの最適な構造を判断し、問題を分解することは容易ではありません。

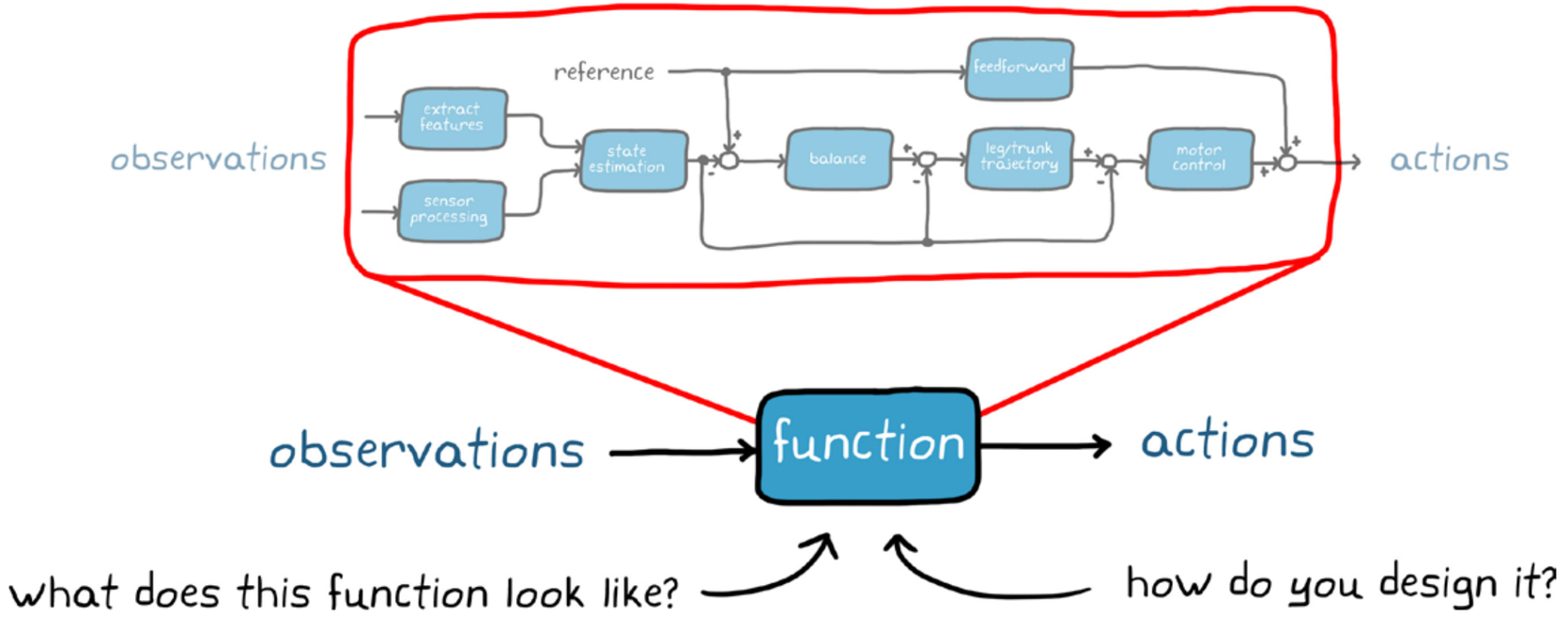


強化学習のメリット

これらのコンポーネントを個別に設計するのではなく、観測と詳細レベルのアクションの出力のすべてを直接取り込む単一の関数にあらゆるものを入力できるとしたらどうなるでしょうか。

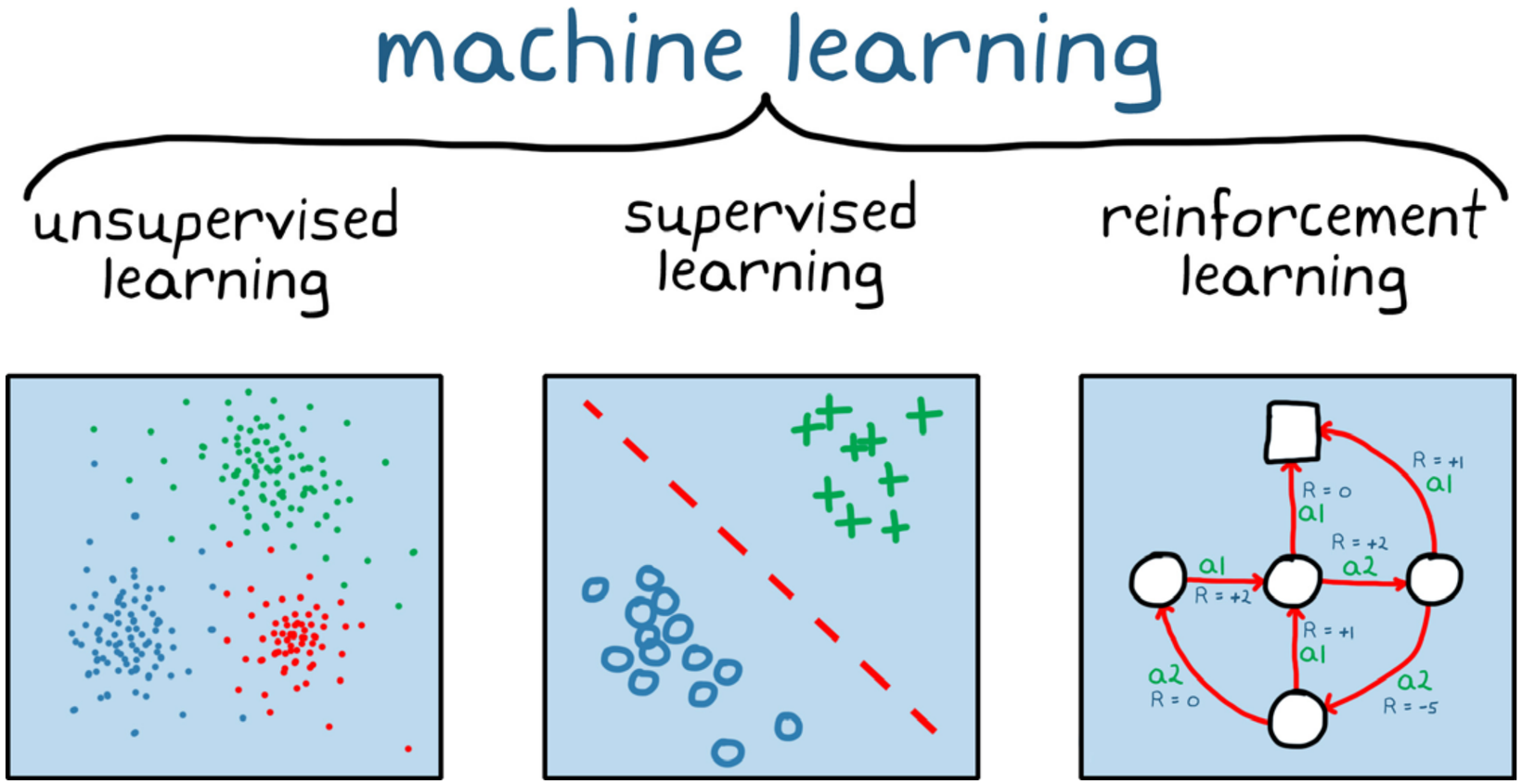
これでブロック線図は確実に簡易化されますが、この関数はどのようなものになり、どのように設計するのでしょうか。

この単一の大きな関数を作成することは、区分的なサブコンポーネントから成る制御システムの構築より困難に思われるかもしれませんが、これが強化学習が役立つところなのです。



強化学習: 機械学習のサブセット

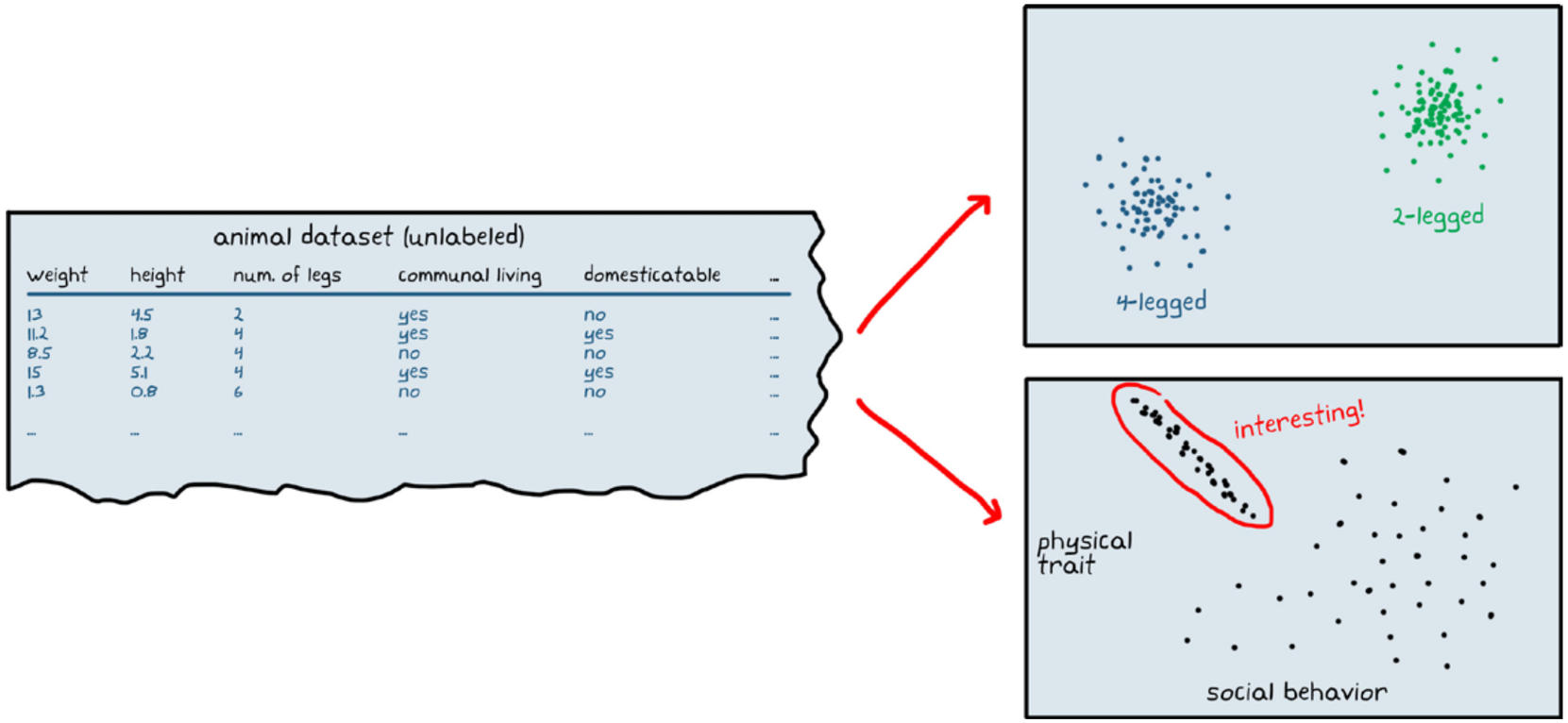
強化学習は、機械学習の主要 3 カテゴリのうちの 1 つです。この eBook では教師なし学習や教師あり学習の詳細は扱いませんが、これら 2 つと強化学習の違いを認識しておくことは重要です。



機械学習: 教師なし学習

教師なし学習は、分類やラベル付けが行われていないデータセットでパターンや隠れた構造の検出に使用されます。

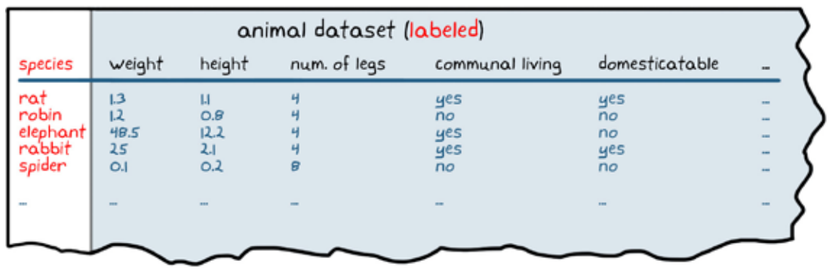
たとえば、10 万匹の動物の物理的属性と社会的傾向に関する情報があるとします。この場合、教師なし学習を使用して、動物をグループ化したり、類似した特徴でクラスタリングすることができます。これらのグループは、脚の数を基準にしたり、または事前に判明していなかった物理的特徴や社会的動作の相関などのあまりはっきりしていないパターンを基準にすることができます。

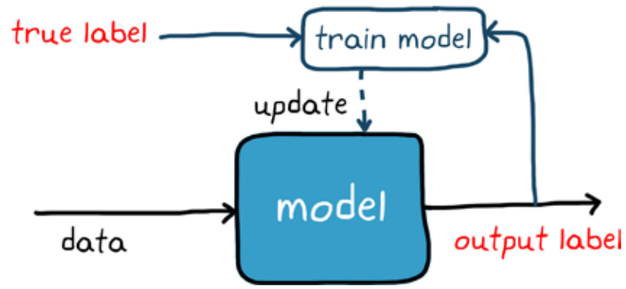


機械学習: 教師あり学習

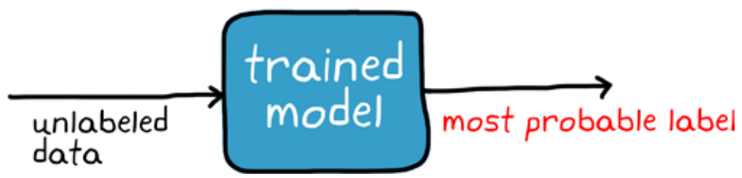
教師あり学習を使用し、特定の入力にラベルを適用するようにコンピューターに学習させます。たとえば、動物の特徴のデータセット列の 1 つが種の場合、種をラベルとして扱い、データの残りを数学的モデルへの入力として扱うことができます。

教師あり学習を使用してモデルに学習させ、データセット内の動物の特徴の各セットを正しくラベル付けすることができます。モデルは種を推測し、その後、機械学習アルゴリズムが体系的にモデルを微調整します。





信頼できるモデルの獲得に十分なトレーニングデータがあれば、新しいラベル付けされていない動物の特徴を入力すると、学習済みのモデルが最も有力なラベルをそれに付けます。

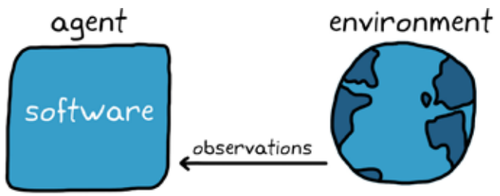


機械学習: 強化学習

強化学習はまったく異なるものです。他の 2 つの学習フレームワークは静的なデータセットを使用して機能しますが、それらとは異なり、RL は動的な環境からのデータを処理します。また、RL の目標はデータのクラスタリングやラベル付けではなく、最良の結果の生成に最適な一連のアクションを見つけることです。強化学習では、“エージェント” と呼ばれるソフトウェアを環境の探索、対話および学習に使用して、問題を解決します。

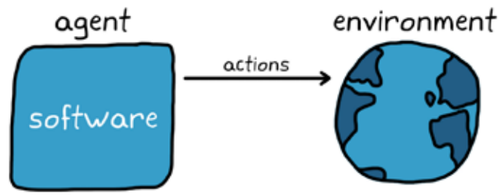
1

エージェントは、環境の現在の状態を観測できます。



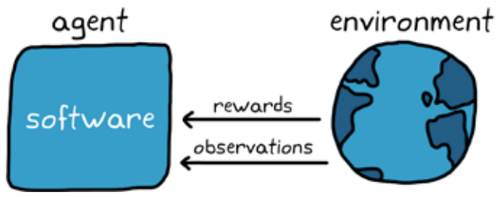
2

観測された状態から、実行アクションが決定されます。



3

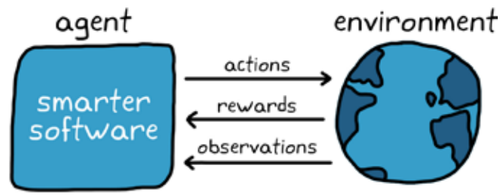
そのアクションにより環境の状態が変化し、報酬が生成されます。エージェントはその両方を受け取ります。



4

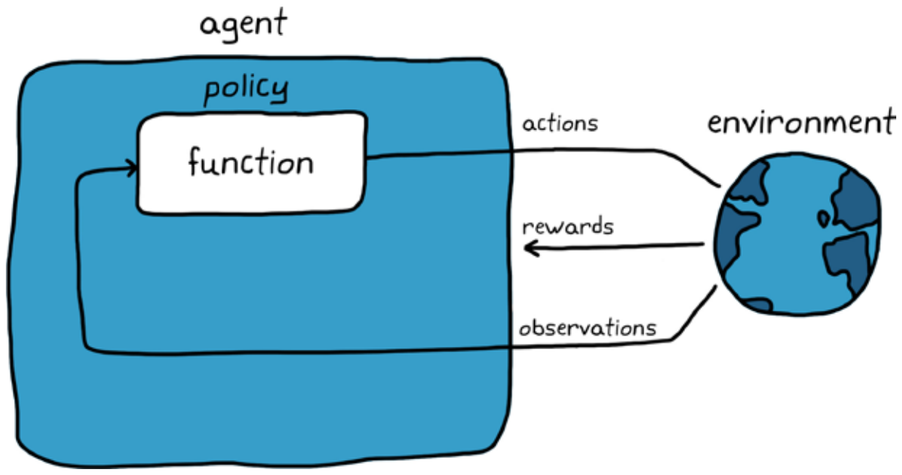
この新しい情報を使用して、エージェントはそのアクションが適切なものであり、反復すべきか、または不適切であるため避けるべきかの判断が可能になります。

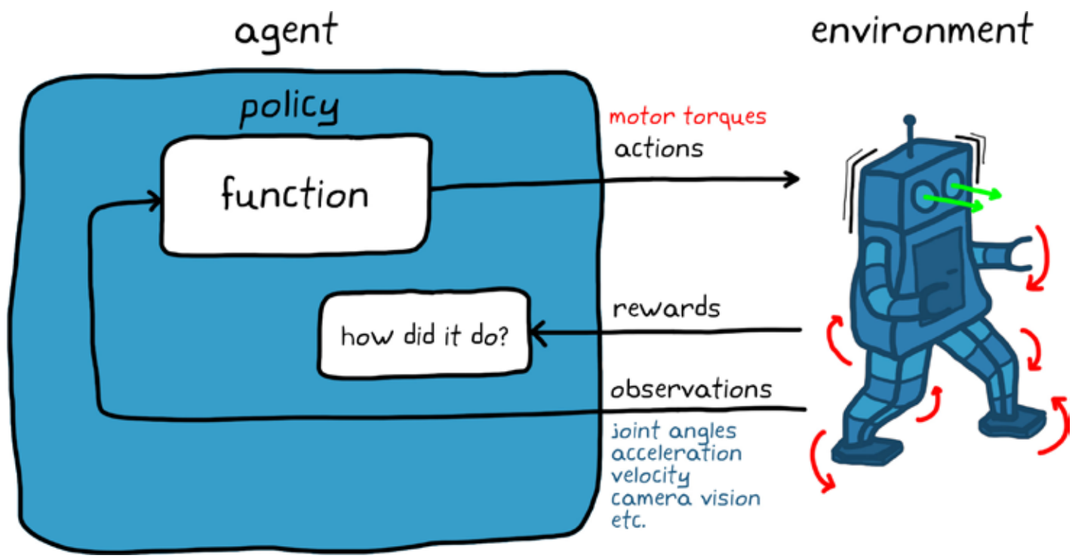
観測 - アクション - 報酬のサイクルを学習の完了まで続行します。



強化学習の構成

エージェント内に、状態の観測を取り込み (入力)、それをアクション (出力) にマッピングする関数があります。これが前述単一の関数で、制御システムの個々のサブコンポーネントのすべてを置き換わります。RL の用語では、この関数を “ポリシー” と呼びます。一連の観測から、ポリシーは実行するアクションを決定します。





歩行するロボットの例では、観測情報は各関節の角度、ロボット胴体の加速度と角速度、そしてビジョンセンサーからの何千ものピクセルです。ポリシーはこれらすべての観測を取り込み、ロボットの腕と脚を動かすモーターコマンドを出力します。

次に、環境によってアクチュエータ コマンドのその特定の組み合わせの好適度をエージェントに知らせる報酬が生成されます。ロボットが直立したまま歩行を続けることができれば、ロボットが地面に倒れた場合よりも報酬が高くなります。

最適なポリシーの学習

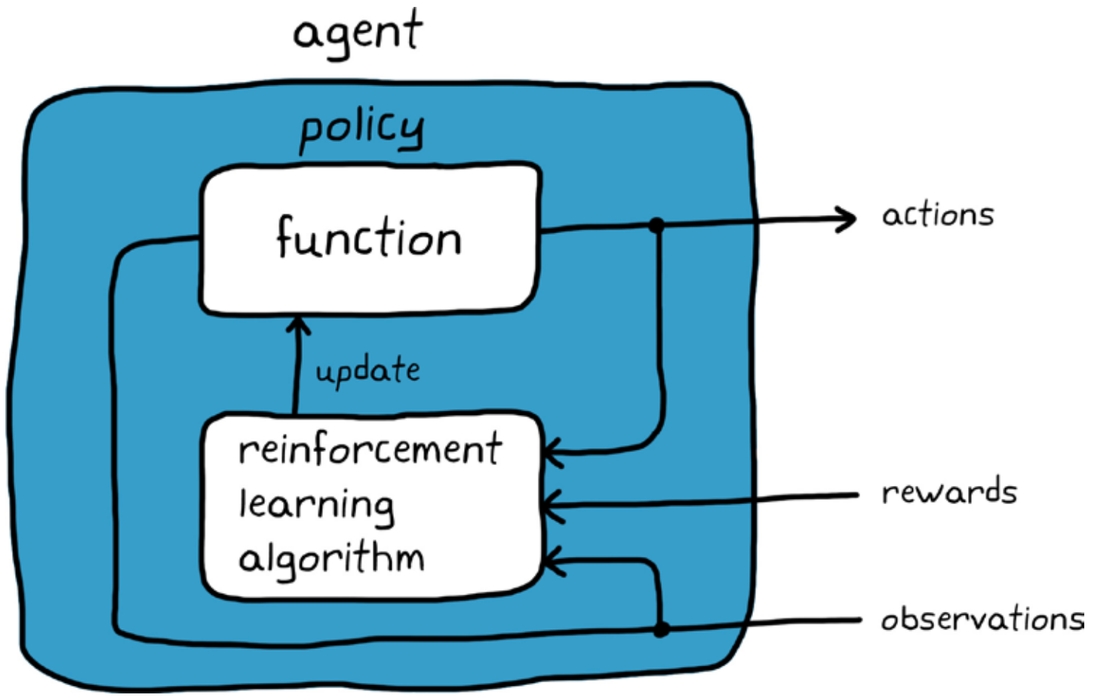
観測された各状態に応じて、正しいアクチュエータを正しく指示を行う完全なポリシーを設計できれば、あなたの仕事は終わりです。

言うまでもなく、ほとんどの状況下では困難なことです。完全なポリシーを見つけたとしても、環境は時間の経過につれて変化するため、静的なマッピングは最適なものではなくなります。

ここで登場するのが強化学習アルゴリズムです。

強化学習アルゴリズムは、実行したアクション、環境からの観測、および収集された報酬に基づいてポリシーを変更します。

エージェントの目標は、強化学習アルゴリズムを使用して最良のポリシーを学習することであり、どのような状態でも環境と対話しながら常に最適なアクションを実行できるように、つまり長期にわたって最高の報酬を生成できるようにすることです。



学習の意味

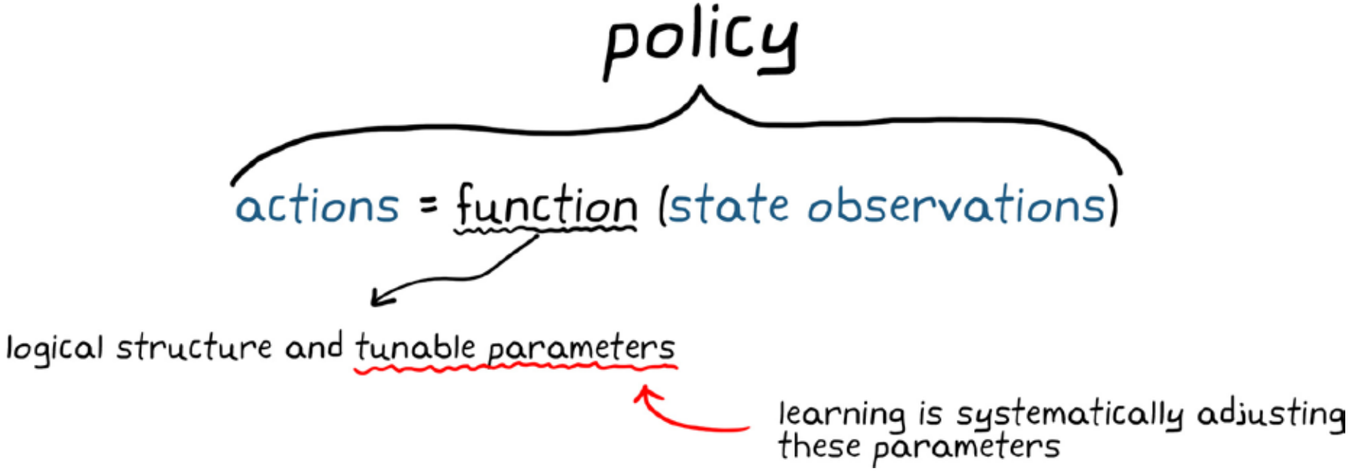
機械が学習するという意味を理解するため、ポリシーとは実際には何であるかを考えましょう。ポリシーとはロジックと調整可能なパラメーターで構成される関数です。

十分なポリシー構造 (ロジック構造) には、最適なポリシーを生成するパラメーターのセットが存在します。最適なポリシーとは、長期にわたって最大の報酬を生み出す、アクションへの状態のマッピングです。

“学習” とは、パラメーターを体系的に調整して最適なポリシーに収束させるためのプロセスを指します。

このようにすることで適正なポリシー構造の設定に専念すれば、関数を手動で調整することなく、適正なパラメーターを得ることができます。

特定のプロセスを通じて独自のパラメーターをコンピューターに学習させることができます。このプロセスについては後続部分で説明しますが、ここでは高度な試行錯誤とお考えください。



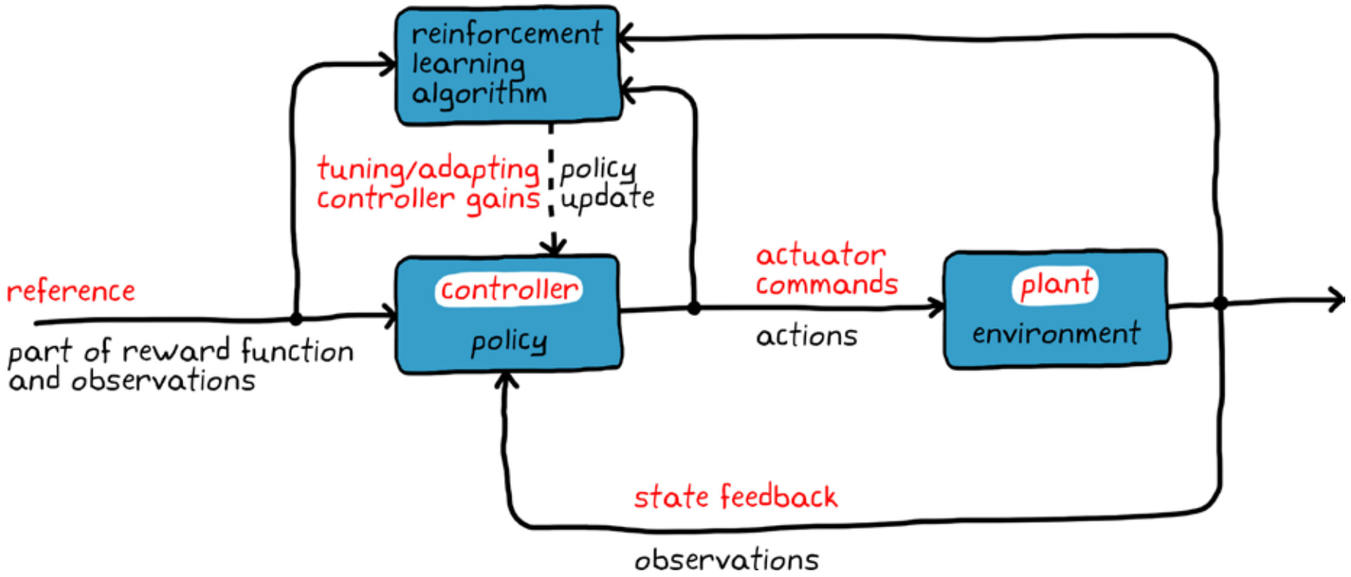
強化学習と従来の制御の類似点

強化学習の目標は、制御の問題と類似していますが、アプローチが異なり、同じ概念を表すために使用する用語が異なります。

いずれの方法でも、必要なシステム動作を生成できるようにシステムへの正しい入力を判断する必要があります。

観測した環境状態 (プラント) を最良のアクション (アクチュエータ コマンド) にマップするポリシー (コントローラー) を設計する方法を見出そうとしているのです。

状態フィードバック信号は、環境からの観測であり、基準信号は報酬関数と環境観測の両方に組み込まれます。



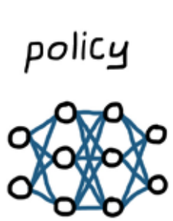
強化学習ワークフローの概要

一般に、強化学習では 5 つの異なる領域を取り扱う必要があります。この eBook では、最初の領域である、環境の設定を中心に説明します。このシリーズの他の eBook では、報酬、ポリシー、トレーニング、配布について詳しく説明します。

1 エージェントが学習できる環境が必要です。環境内への配置が必要なものを選択し、さらにそれがシミュレーションか、または物理的な設定かを選択する必要があります。

2 最後にエージェントに何をさせるかを考え、エージェントにそれをさせるための動機付けになる報酬関数を作成する必要があります。

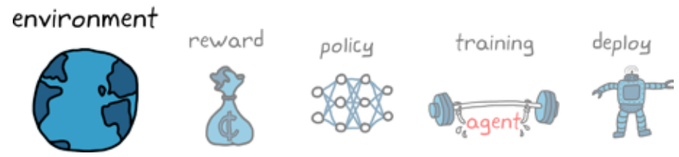
3 ポリシーを表現する方法を選択する必要があります。エージェントの意思決定部分を構成するパラメーターとロジックの構造化方法を検討します。



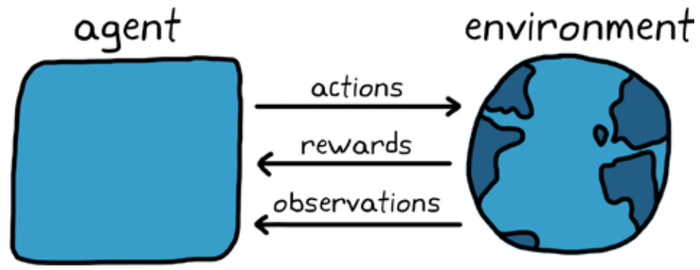
4 最適なポリシーパラメーターを見つける機能を果たすエージェントをトレーニングするアルゴリズムを選択する必要があります。

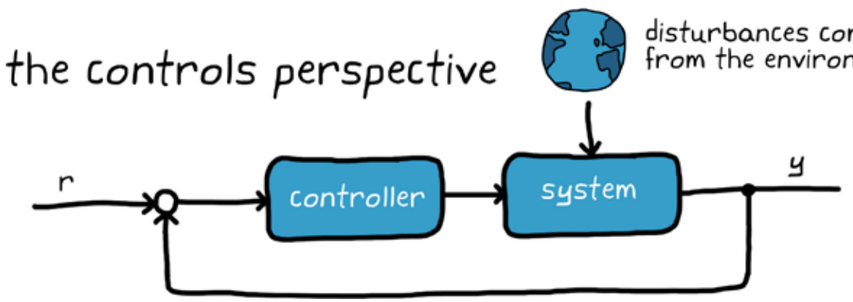
5 最終的に、ポリシーをフィールドに展開し、その結果を確認してポリシーを活用する必要があります。

環境



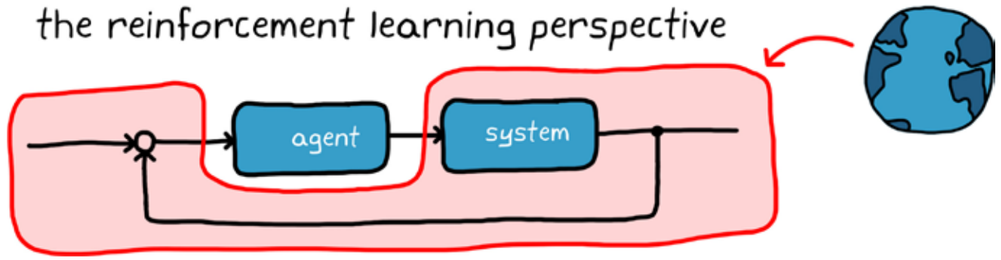
環境とは、エージェントの外側に存在するあらゆるものです。エージェントがアクションを送信する場所であり、報酬が生まれ、観測が行われる場所です。



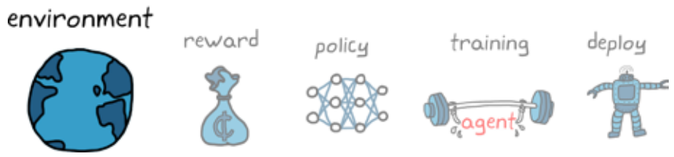


制御の観点から捉えるユーザーは、環境を、制御対象のシステムに影響を与える外乱であると考える傾向があるため、この定義は混乱を招くことがあります。

しかし、強化学習の用語では、環境はエージェント以外のすべてを意味します。これにはシステムダイナミクスも含まれます。このように、実質的にシステムの大部分は環境に含まれます。エージェントは、アクションを生成し、学習を通じてポリシーを更新する小さなソフトウェアにすぎません。



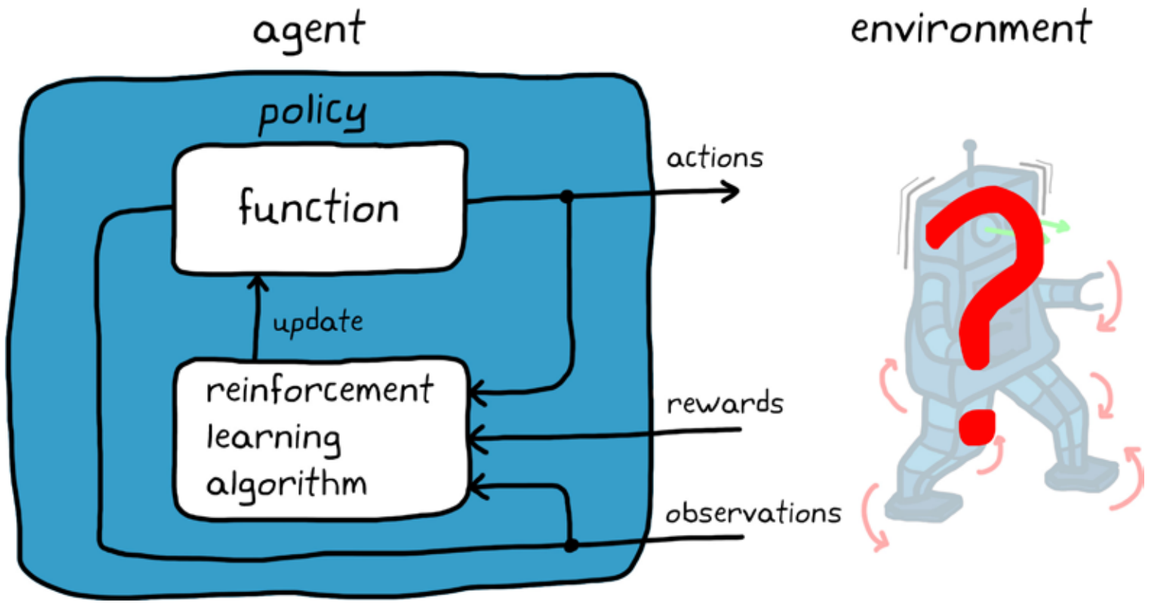
モデルに依存しない強化学習



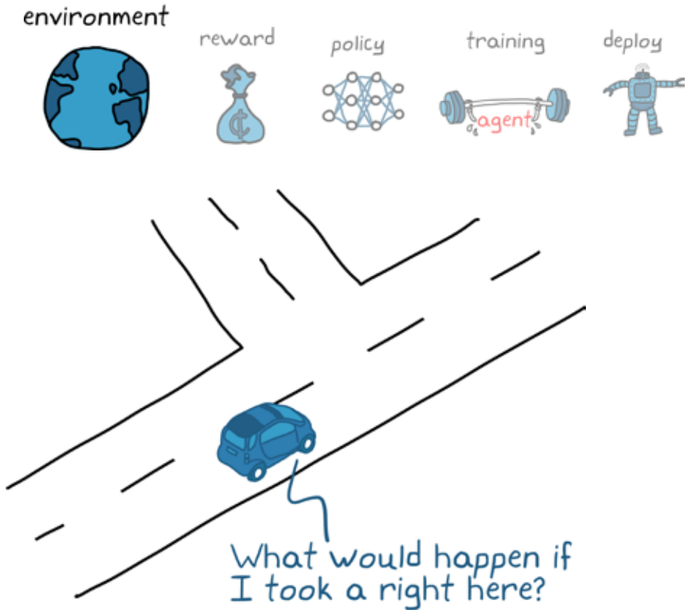
強化学習が非常に強力である理由の 1 つは、エージェントが環境について何も認識している必要がないことです。しかし、環境との対話方法の学習はできます。たとえば、エージェントは歩行ロボットのダイナミクスや運動学を認識する必要はありません。関節がどのように動くかや脚の長さを知らなくても、最大の報酬を得る方法を見つけ出すことができます。

これは、“モデルに依存しない強化学習” と呼ばれます。

モデルに依存しない RL を使用し、RL を備えたエージェントをシステムに配置すれば、エージェントは最適なポリシーを学習できます (ポリシーに観測、報酬、アクション、および十分な内部状態へのアクセスを与えていると仮定します)。

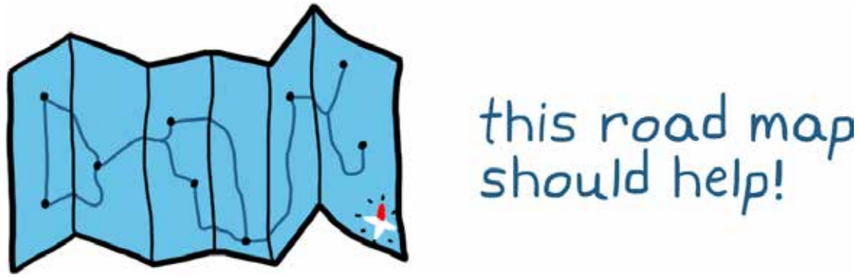


モデルベースの強化学習



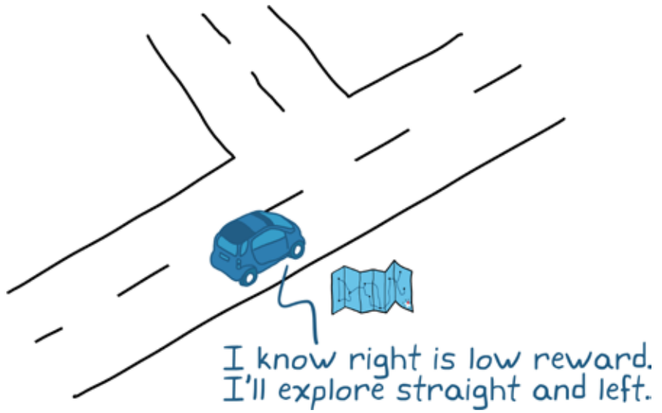
モデルに依存しない RL にも問題があります。エージェントが環境について何も知らなければ、エージェントが最大の報酬を受ける方法を見つけるために状態空間のすべての領域を調べなければなりません。

これはつまり、エージェントは学習プロセス中に、低報酬の領域の調査に時間をかけなければならないことを意味します。

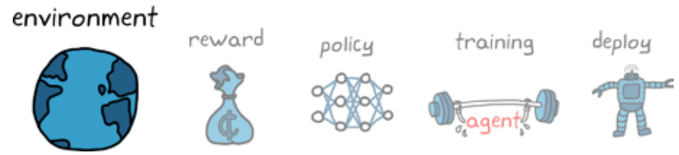


しかし、状態空間の一定の部分は、調べる価値がないことがわかっているとします。環境のモデル、または環境の部分を提供して、エージェントにこの知識を提供します。

モデルの使用により、エージェントは、実際にアクションを起こさずに、環境の一定の部分を調べることができます。不要な部分を避け、他の部分を調べられるようにすることで、モデルで学習プロセスを補完できます。



モデル非依存とモデルベース



“モデルベースの強化学習” は、モデルを使用してエージェントをガイドし、低報酬であると認識されている状態空間の領域を避けることができるため、最適なポリシーを学習する時間を短縮できます。

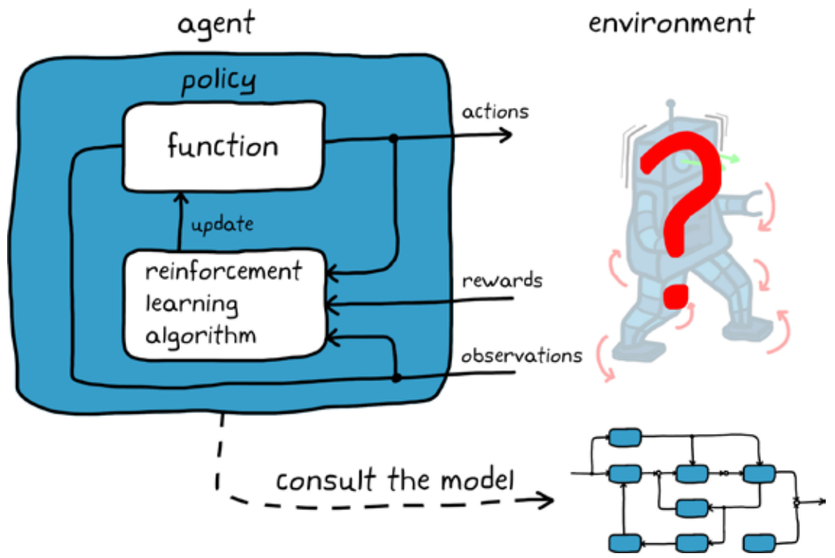
第一に、エージェントにそれらの低報酬の状態に陥らないようにするため、それらの状態でエージェントに最良のアクションを学習させる時間とる必要はありません。

モデルベースの強化学習では、環境モデル全体を知る必要はありません。既知の環境部分のみをエージェントに提供できます。

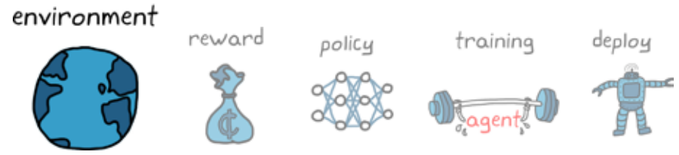
“モデルに依存しない強化学習” は、より一般的なケースであり、この eBook の後続部分で説明します。

モデルなしの強化学習の基礎を理解すると、モデルベースの RL の理解も進みます。

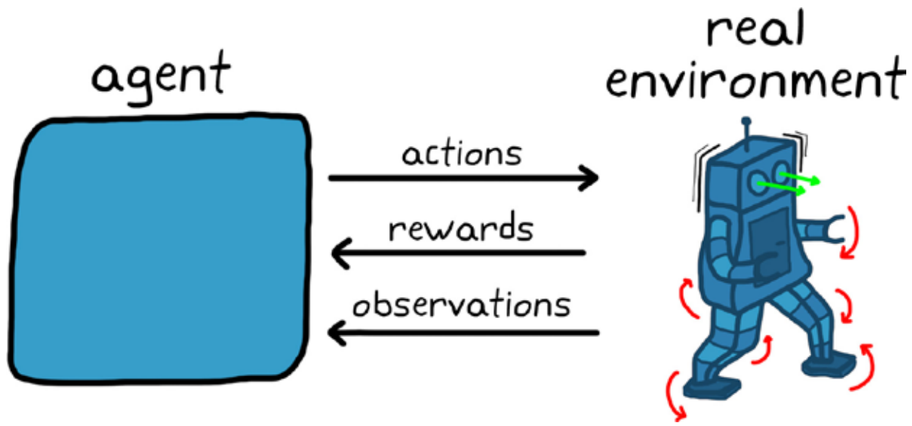
モデルに依存しない RL は現在広く使用されていますが、これは、単純ものであってもモデルの開発が困難な場合に問題の解決に使用したいと考えられているためです。一例として、ピクセル観測からの車やロボットの制御が挙げられます。ピクセルの明暗度と車やロボットのアクションの関連は、ほとんどの場合て直観的にわかるものではありません。



実際の環境とシミュレーションの環境



エージェントは環境との対話から学習するため、エージェントには実際に環境と対話する方法が必要です。これは実際の物理的環境の場合もシミュレーションの場合もあり、この 2 つのいずれを選択するかは状況によって異なります。

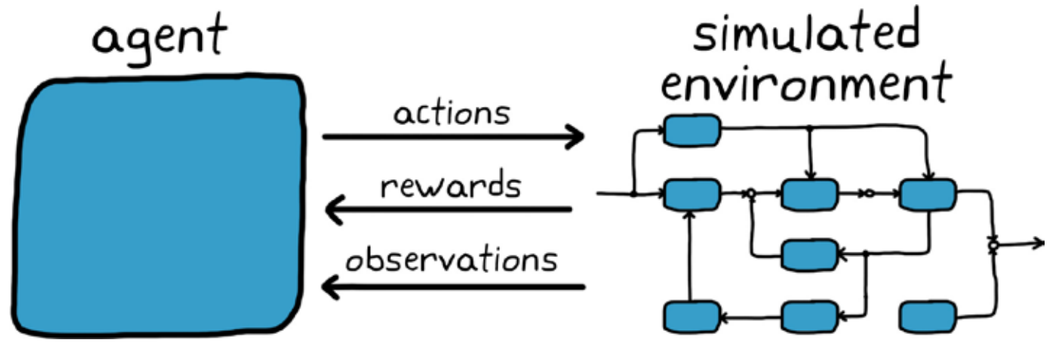


実際の環境

精度: 実際の環境ほど、環境を完全に表現できるものはありません。

簡潔性: モデルを作成したり、検証したりするために時間を費やす必要がありません。

必要性: 環境が継続的に変化する場合や、正確なモデル化が困難な場合は、実際の環境で学習させることが必要な場合もあります。



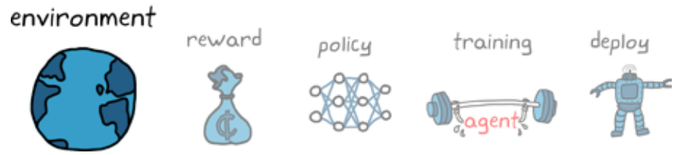
シミュレーション

スピード: シミュレーションは実際よりも高速で実行でき、また並列化も可能であり、低速な学習プロセスを高速化できます。

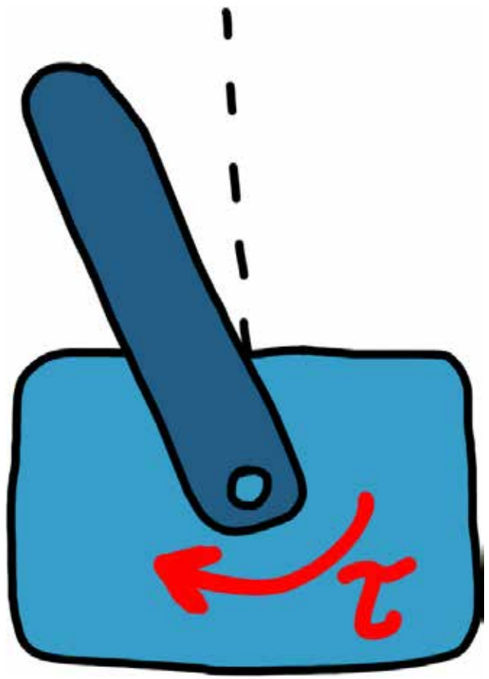
条件のシミュレーション: テストが困難な状況では、モデル化の方が容易です。

安全性: ハードウェア破損のリスクはありません。

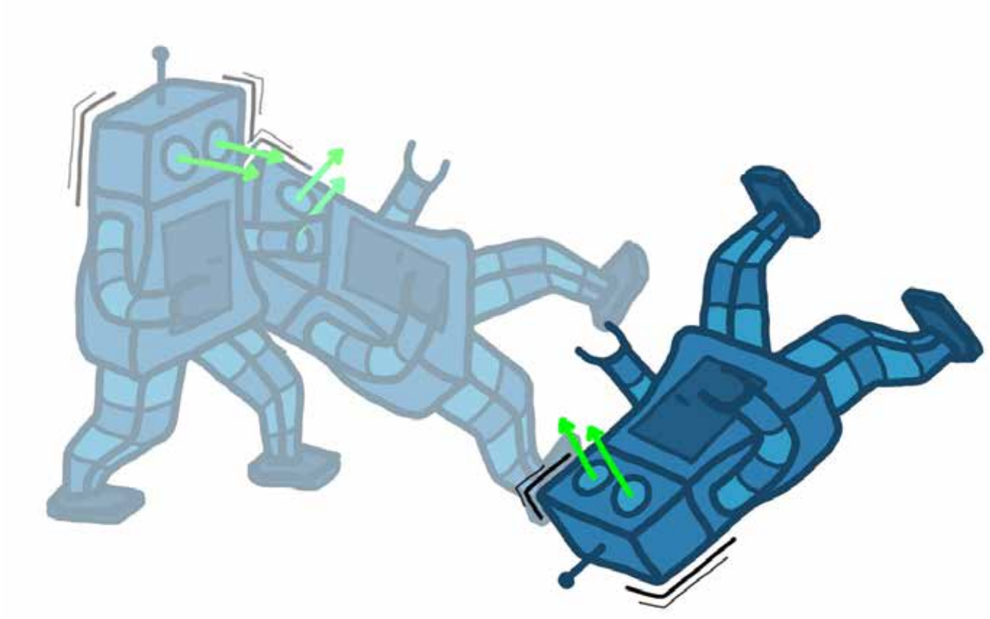
実際の環境とシミュレーションの環境



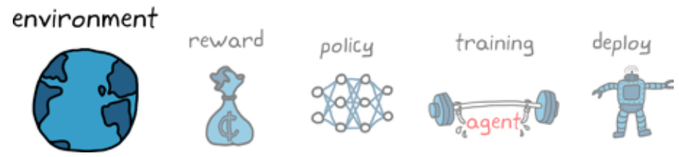
たとえば、実際の振子の設定を使用して、エージェントに倒立振子のバランスを保つ方法を学習させることができます。ハードウェアがそれ自体や他のものを破損する可能性は低いため、これは良い解決策と言えるでしょう。状態空間と行動空間が比較的小さいため、トレーニングにもそれほど時間がかからないでしょう。



しかし、歩行ロボットのケースでは、これは適したアイデアではないかもしれません。トレーニングの開始時点でポリシーが十分に最適化されていなければ、歩行するどころか、脚を動かす方法を学習するまでに、ロボットは何度も転倒したり壊れたりするでしょう。これはハードウェアを破損させるだけでなく、毎回ロボットを起こすために非常に時間がかかります。よって理想的とは言えません。



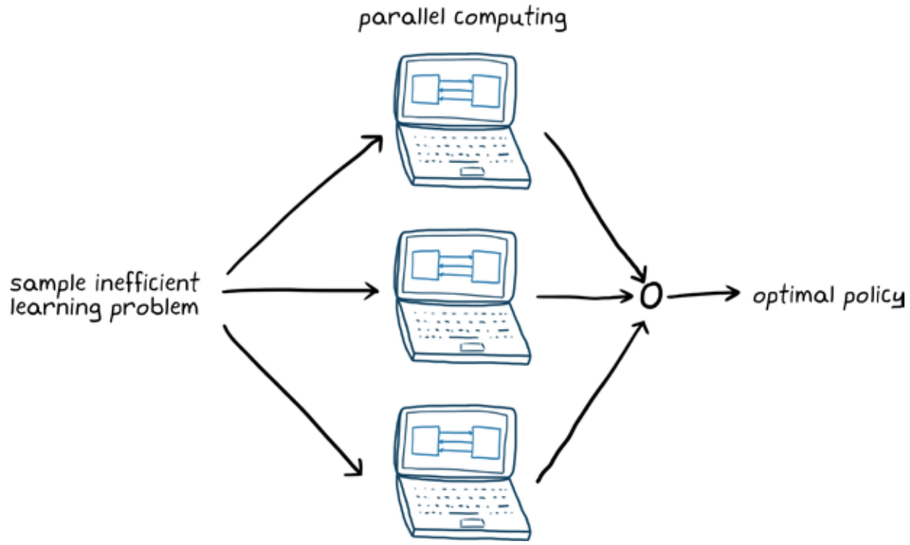
シミュレーション環境のメリット

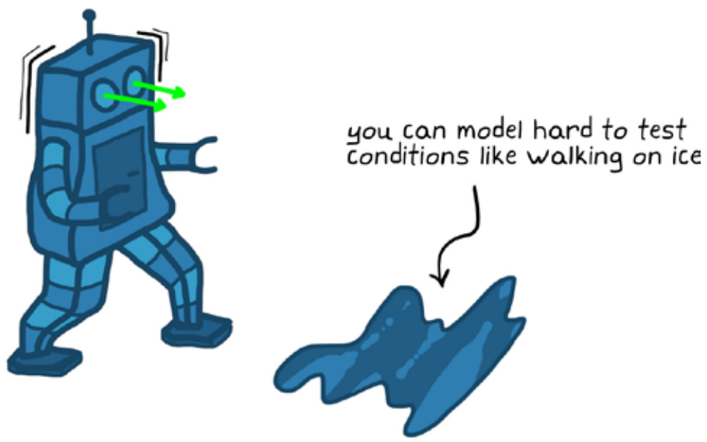


シミュレーション環境は、エージェントのトレーニングに最も一般的な方法です。制御問題に関するメリットの 1 つは、一般的に従来の制御設計ではシステムと環境のモデルが必要とされるため、通常すでに好適なモデルがあることです。MATLAB® または Simulink® で構築されたモデルがすでに存在する場合は、既存のコントローラーを強化学習エージェントに置き換え、環境に報酬関数を追加して、学習プロセスを開始できます。

学習とは、試行、エラー、修正などの多くのサンプルを必要とするプロセスです。このため、1 つの最適な解決策に収束させるまでに何千あるいは何百万もの事例が必要となる場合があるため、非常に非効率的です。

環境のモデルは実際よりも高速で実行でき、また多くのシミュレーションを並列的に実行して速度を上げることができます。この両方のアプローチにより、学習プロセスの速度を上げることができます。





シミュレーションの状況を使用すると、エージェントを実世界で使用するよりも、制御がはるかに容易になります。

たとえば、ロボットはさまざまな異なる路面を歩行しなければならないことがあります。氷のような低摩擦の路面を歩くシミュレーションは、実際の氷を使ってテストするよりもずっと簡素になります。さらに、低摩擦の環境でエージェントをトレーニングすると、あらゆる路面でのロボットの直立の維持に実際に役立ちます。シミュレーションを使用することで、トレーニング環境の創出を強化できます。

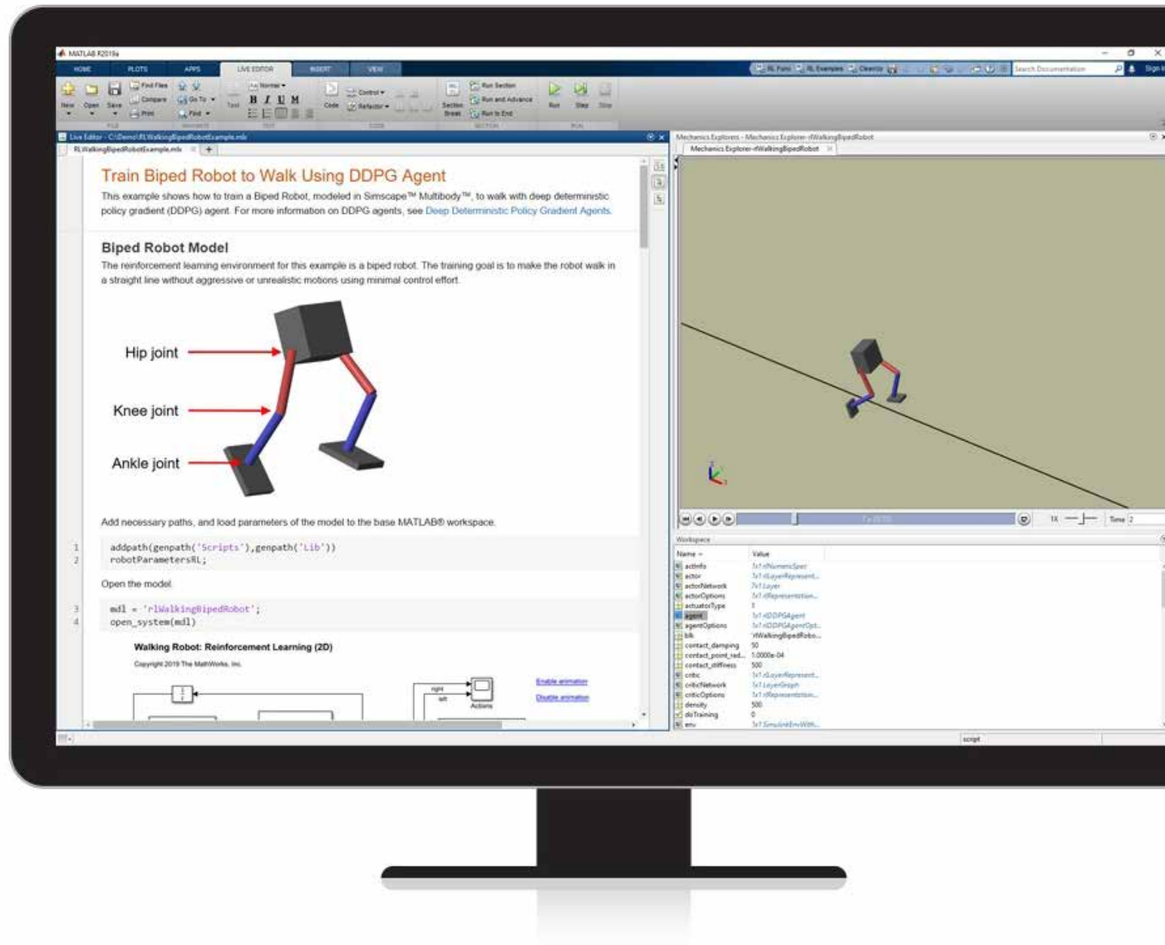
MATLAB および Simulink を使用した強化学習

Reinforcement Learning Toolbox には、強化学習アルゴリズムを使用したポリシーのトレーニングのための関数とブロックがあります。これらのポリシーを使用して、ロボットや自律システムなどの複雑なシステムのためのコントローラーと意思決定アルゴリズムを実装できます。このツールボックスを使用すると、MATLAB で表現された環境との対話と可能にして、または Simulink モデルを使用して、ポリシーに学習させることができます。

たとえば、MATLAB で強化学習環境を定義するには、提供されたテンプレート スクリプトとクラスを使用し、環境ダイナミクス、報酬、観測、アクションを用途の必要に応じて変更します。

Simulink では、制御と強化学習の問題の解決に必要となる頻度が高い、多様なタイプの環境をモデル化できます。たとえば、車両運動や飛行力学、Simscape™ を使用した各種の物理システム、System Identification Toolbox™ によって測定データから近似化されたダイナミクス、レーダー、ライダー、IMU のようなのセンサーなどをモデル化できます。

mathworks.com/products/reinforcement-learning



関連情報

agent

見る

What Is Reinforcement Learning? (強化学習とは) (14 分 05 秒)

Understanding the Environment and Rewards (環境と報酬の理解) (13 分 27 秒)

歩行ロボットのモデリングおよびシミュレーション (21:19)

検索

Getting Started with Reinforcement Learning Toolbox (Reinforcement Learning Toolbox 入門)

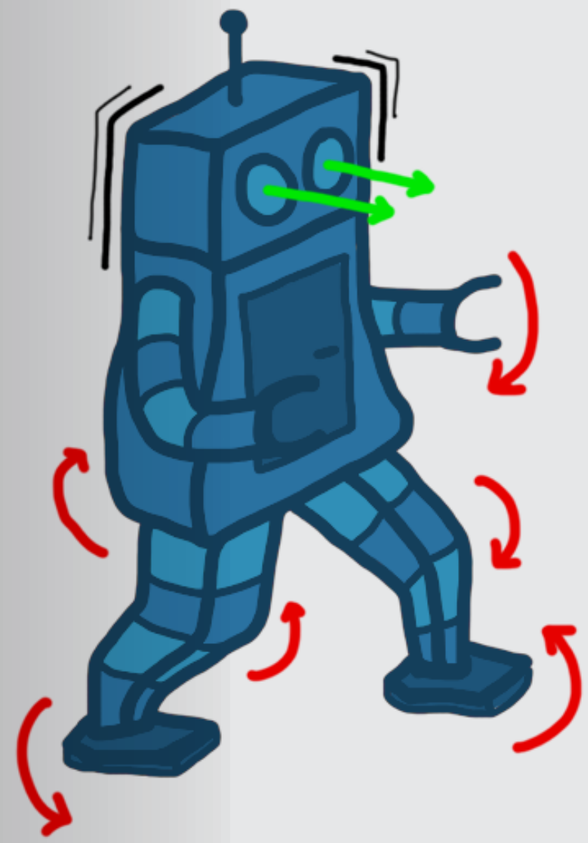
CREATING ENVIRONMENTS IN MATLAB (MATLAB での環境作成)

Creating Environments in Simulink (Simulink での環境作成)

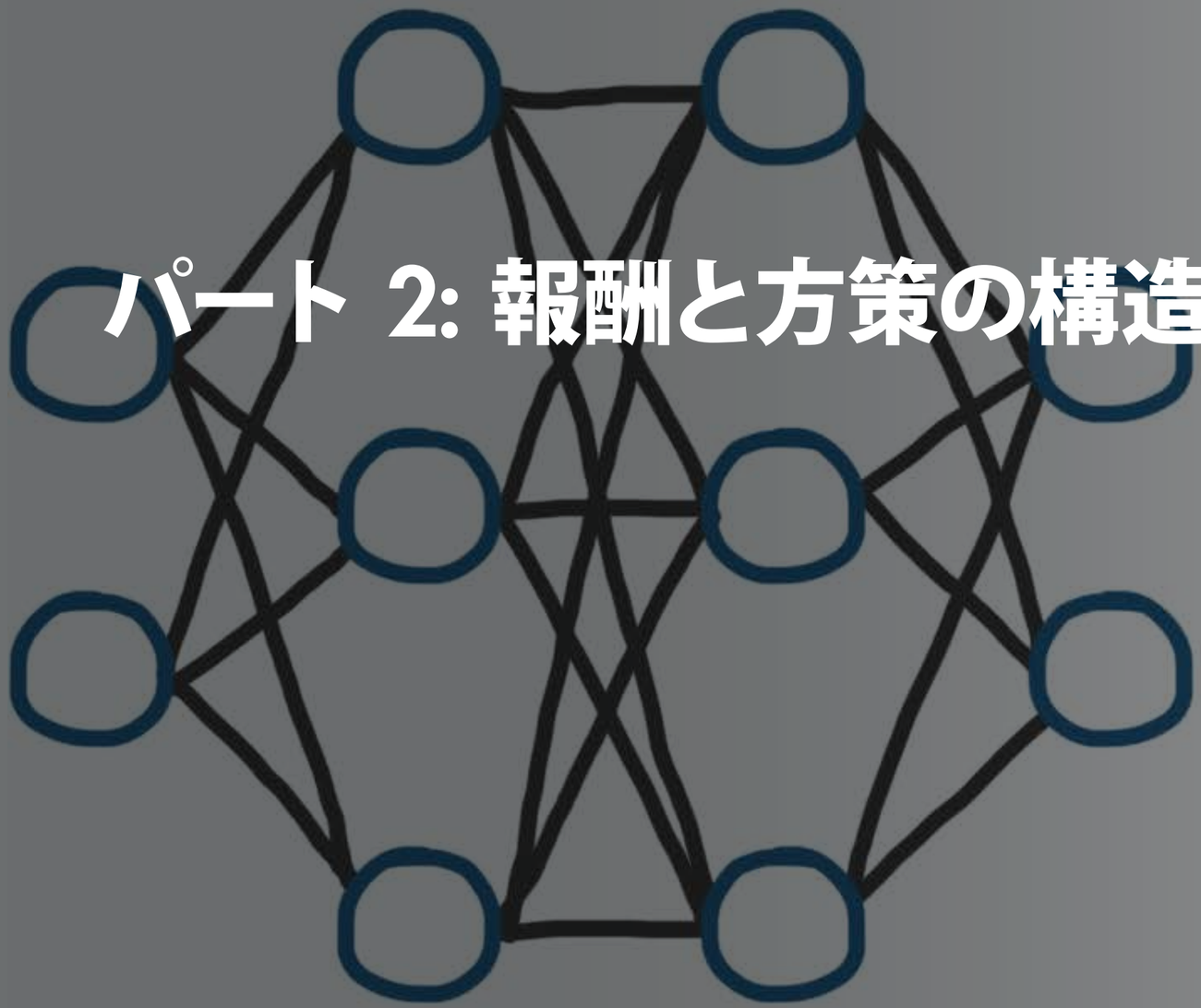
Modeling Flight Dynamics in Simulink (Simulink での飛行力学のモデリング)

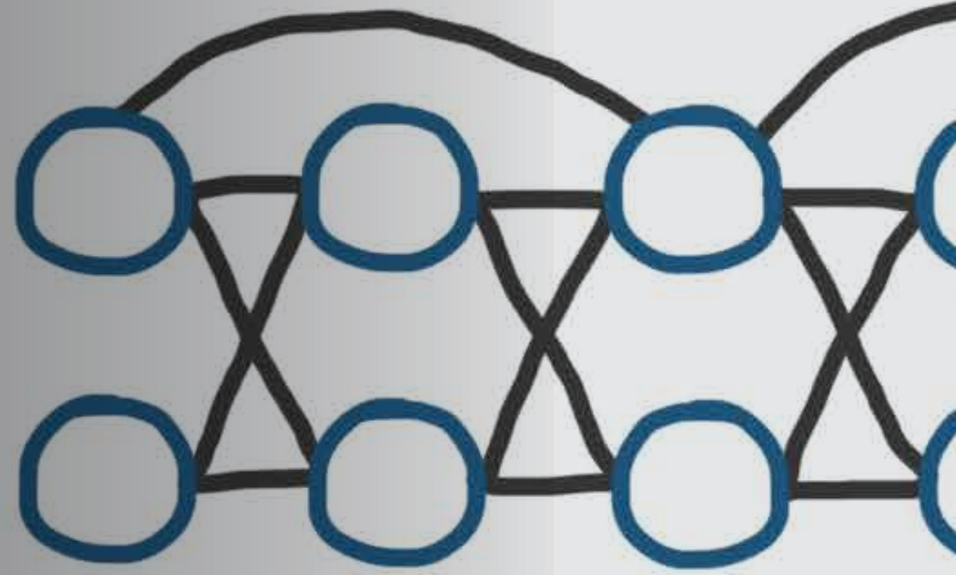
Simulating Full Vehicle Dynamics in Simulink (Simulink での完全車両運動のモデリング)

environment



パート 2: 報酬と方策の構造の理解

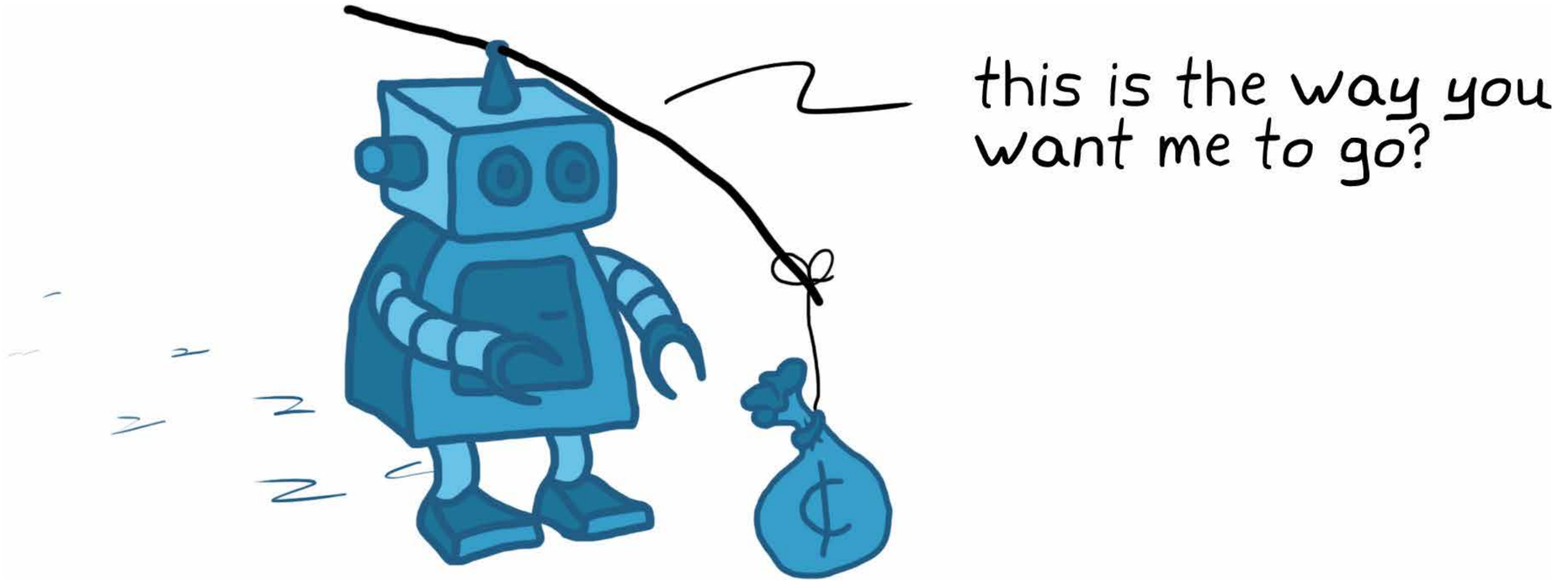




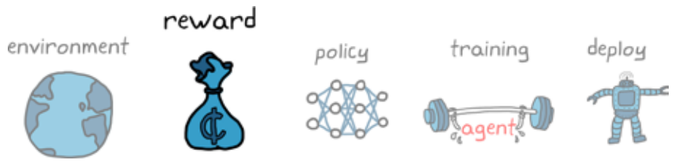
MathWorks®

報酬

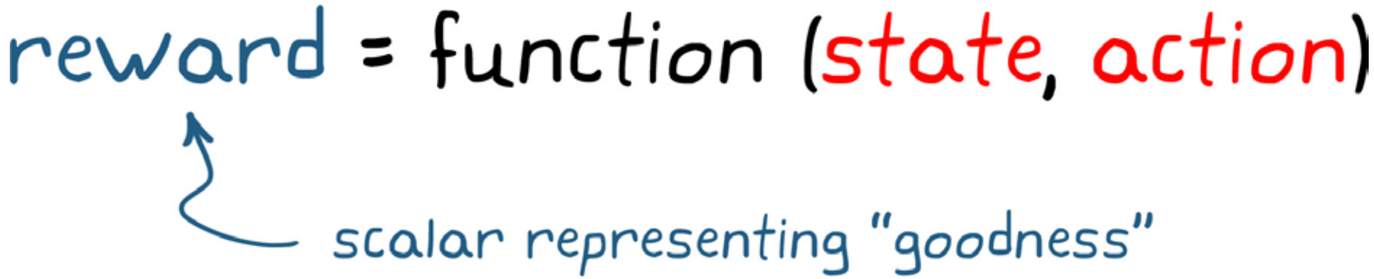
環境の設定を行ったら、次に、エージェントに行わせたいこと、および、望み通りの動作をした際にどのように報酬を与えるかを検討します。この手順では、学習アルゴリズムが、方策が改善されている際にそれを「理解」し、最終的に望む結果に収束するよう報酬関数を設計することが要求されます。



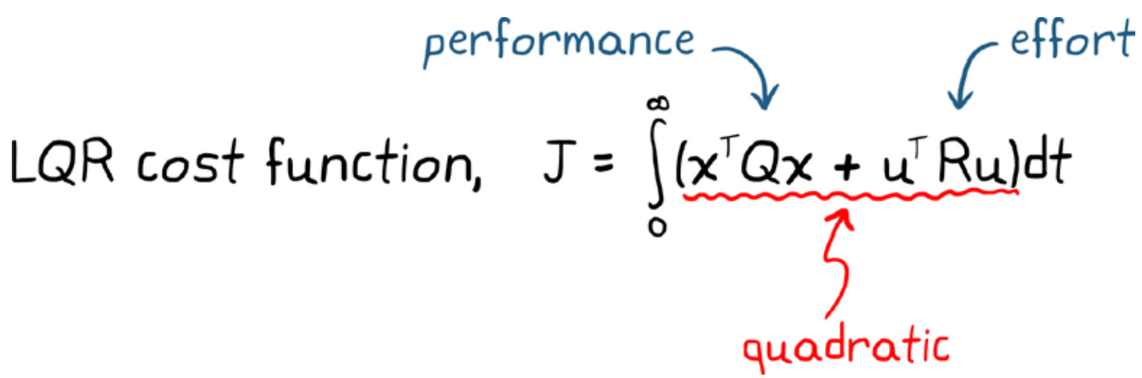
報酬とは



*報酬*とは、エージェントが特定の状態を維持し、特定の行動を行う上での「良さ」を表現するスカラー数を生み出す関数のことです。

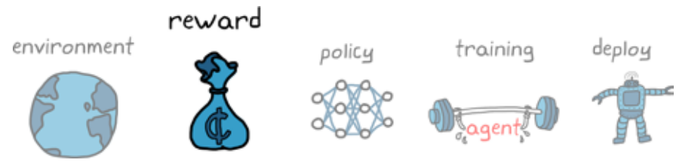


この概念は、システム性能の劣化とアクチュエーター負荷の増加にペナルティを与える LQR のコスト関数と似たものです。もちろん、コスト関数が値を最少化しようとする試みであることに対し、報酬関数が値を最大化しようとする試みであるという違いはあります。しかし、報酬は負のコストと考えることができるため、同じ問題の解決につながります。

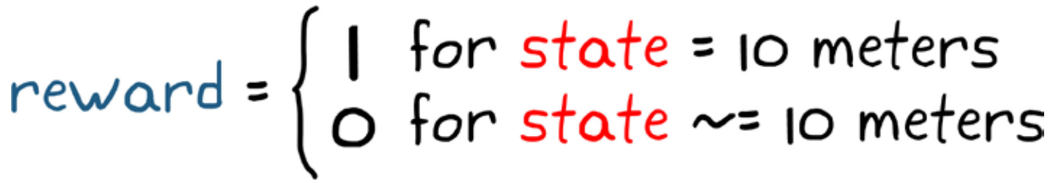


主な違いは、コスト関数が二次である LQR とは異なり、強化学習 (RL) では報酬関数の作成に制限が全く無いことです。スパースな報酬、タイムステップごとの報酬、または長い期間の後のエピソードの最後にだけ与えられる報酬を作成することができます。報酬は非線形関数からの計算、または数千のパラメーターを用いた計算が可能です。これはエージェントが効果的に学習するために必要なものによって異なります。

スパースな報酬

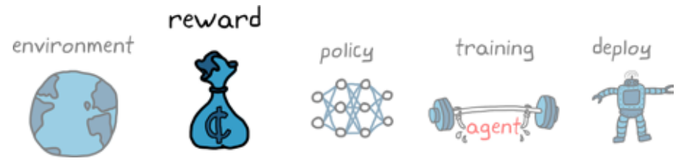


報酬関数の作成に制限が無いため、報酬がスパースになる状況が生じる場合があります。つまり、動機付けしたい目標が、長い一連の行動の後に起こるということです。これは、歩行ロボットが 10 メートルの歩行に成功した後にのみエージェントが報酬を受け取るような報酬関数を設定した場合に起こります。これは最終的にロボットに何をさせたいかということであるため、このような報酬を設定することは合理的です。



一方でスパースな報酬に生じる問題とは、エージェントが長い期間にわたり失敗し続け、異なる行動を試したり、多くの異なる状態になったりしても報酬を得ることがないため、プロセスで何も学ばない可能性があるということです。エージェントが、ランダムに、スパースな報酬を得る一連の行動を偶然発見する可能性はほとんどないといえます。例えば、ロボットを直立させ、床を這い回らせずに 10 メートル歩行させるための正確で完全なモーター コマンドをはじめから生成することと同じくらい難しいことです。

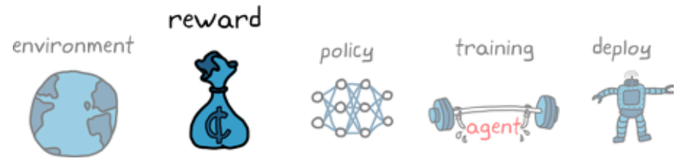
報酬のシェイピング



スパースな報酬は、エージェントを正しい方向に導く小さい中間報酬を与えるという、報酬のシェイピングによって改善することができます。

ただし、報酬のシェイピングにはいくつか問題もあります。最適化アルゴリズムにショートカットを与えると、アルゴリズムはそれを利用します。そして、ショートカットは報酬関数内に隠れます。シェイピングを開始すると、その傾向が高まります。また、報酬関数のシェイピングが不適切な場合、エージェントに対するほとんどの報酬がその解により生じている状況にも関わらず、エージェントが理想的ではない解に収束することがあります。中間報酬によりロボットに 10 メートルの目標まで歩行させることができるだろうと考えがちですが、この最初の報酬に向かって歩行することが実は最適な解ではないという可能性もあります。逆に、歩行の途中で不格好に倒れながらも報酬を集めることにより、この行動が強化される場合もあります。また、ロボットは地面を尺取り虫のように這って残りの報酬を集めることに収束するかもしれません。エージェントにとって、これは非常に合理的な、報酬の高い解ですが、設計者にとっては、明らかに望まない結果といえます。

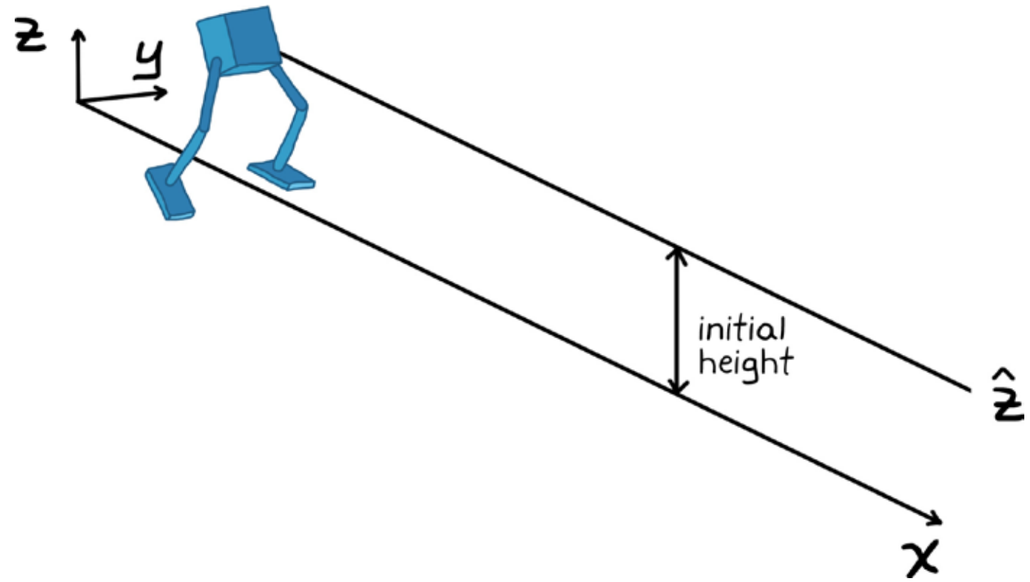
ドメイン固有の知識



報酬のシェイピングは、スパースな報酬を補うという目的以外にも利用されます。エンジニアは、報酬のシェイピングにより、ドメイン固有の知識をエージェントに注入することができます。たとえば、地面を這うことなく歩行させるなど、ロボットに対する要求が明確な場合、ロボット胴体が歩行時の高さに維持されていることに対してエージェントに報酬を与えることができます。さらに、アクチュエーターの負荷が低いこと、長く立っていられること、設定された道筋を外れないでいることに対しても報酬を与えることができます。

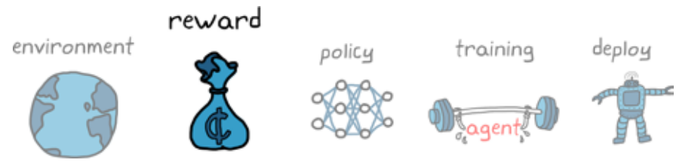
$$r_t = v_x - 3y^2 - 50\hat{z}^2 + 25\frac{T_s}{T_f} - 0.02\sum_i u_{t-1}^{i\,2}$$

- v_x: forward velocity
- $3y^2$: don't stray from path
- $50\hat{z}^2$: keep trunk high
- $25\frac{T_s}{T_f}$: walk as long as possible
- $0.02\sum_i u_{t-1}^{i\,2}$: minimize actuator effort

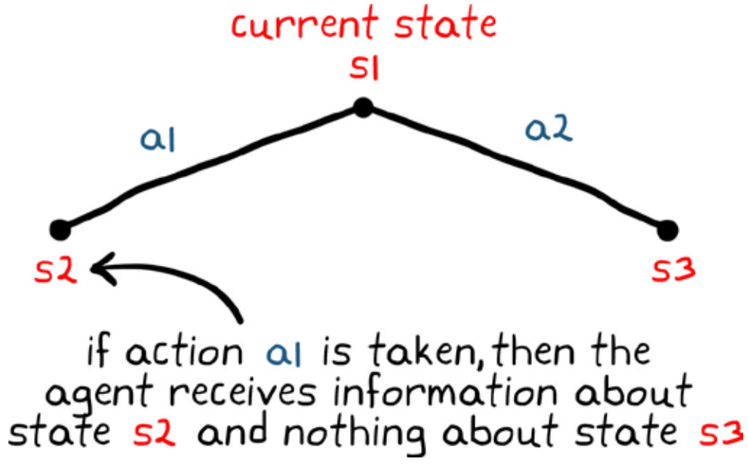


これまでの内容は少し難しく感じられるかもしれません。報酬関数の設計を正しく行うことは、強化学習でも難しいタスクの一つです。たとえば、エージェントの学習に長い時間をかけた後に、望んだ結果が出ないことで、初めて、報酬関数の設計が不適切だったということが分かるというようなことがあります。ただ、全体的な視点で見れば、ここまでの説明により少なくとも注意が必要な部分が分かり、それによって報酬関数の設計が少し楽になるので、その分は進歩したといえるでしょう。

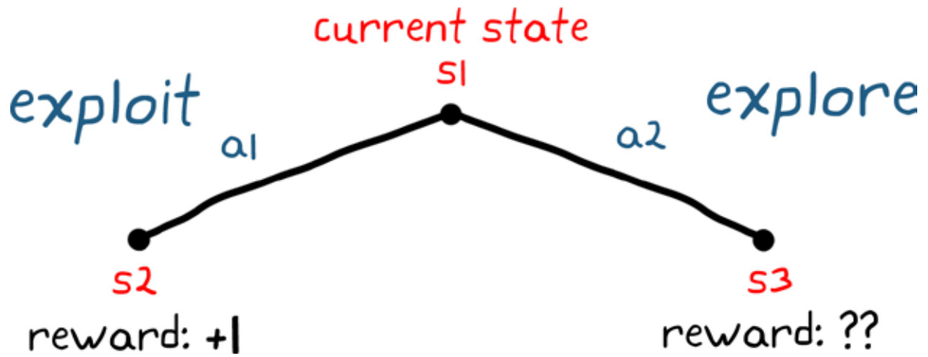
探索と活用



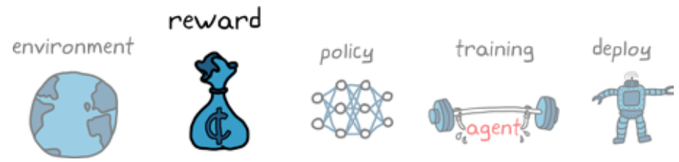
強化学習の重要な側面は、エージェントが環境と対話する際の、探索と活用のトレードオフです。この決定が強化学習で問題になるのは、学習がオンラインで行われるためです。静的なデータセットを扱うのではなく、エージェントの行動によって、環境から返されるデータが決定されます。エージェントの選択によって受け取る情報が決まり、学習に使用できる情報もそれにより決まります。



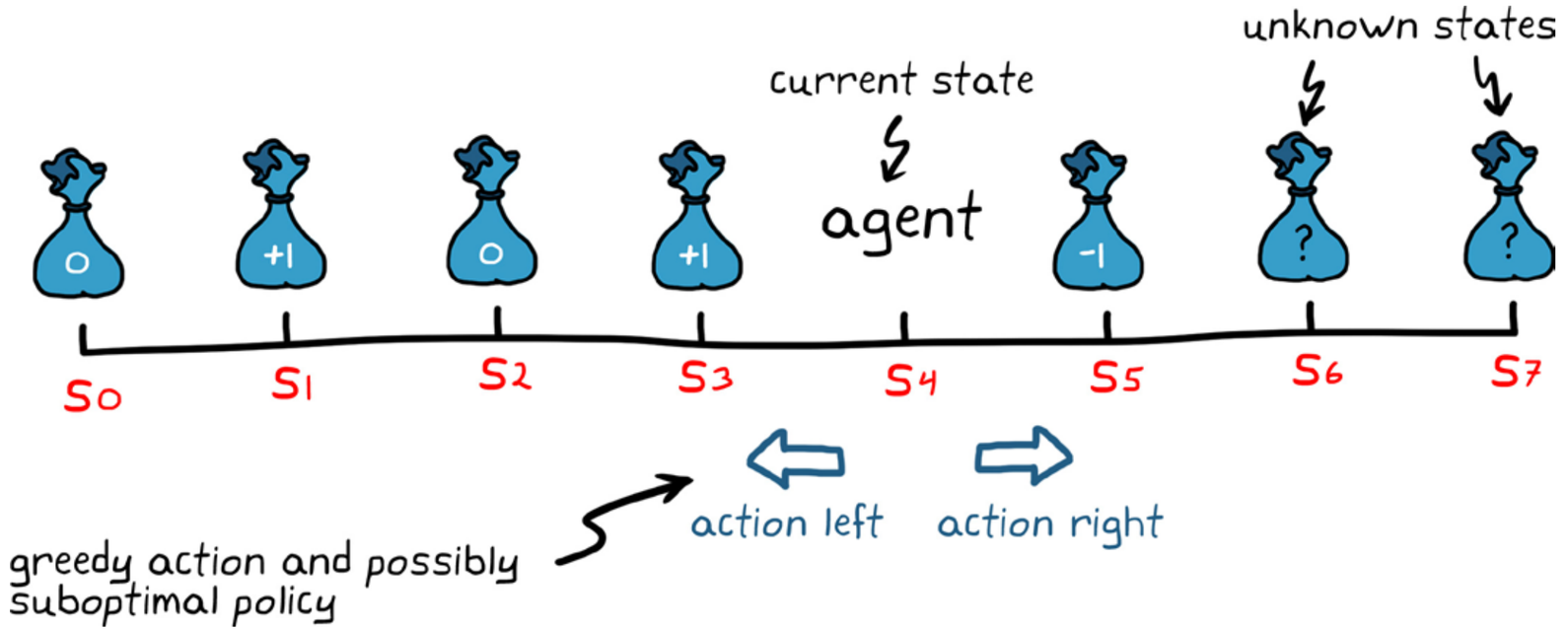
エージェントは、報酬を最大にする既知の行動を選選択して環境を活用すべきか、または環境の一部を探索する未知の行動を選択するべきかのいずれかを選ぶことになります。



純粋な活用での問題



たとえば、エージェント特定の状態で、右に行くか左に行くか、2 つの行動のうち 1 つを選ぶことができるとします。エージェントは左に行くと報酬が 1 プラスされ、右に行くと報酬が 1 マイナスされることを知っています。しかし、最初の報酬が低い状態にある右側の環境について、それよりも先の報酬など、他のことは何も知りません。もし、エージェントが環境を常に活用する貪欲な方法を選択する場合、エージェントは左に行って、その時点で知る限りの最大の報酬を集め、他の状態を完全に無視します。



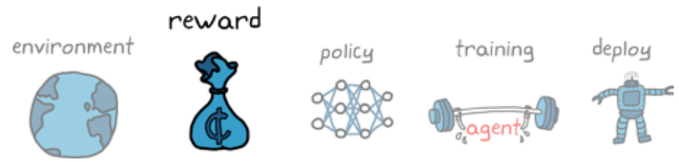
つまり、エージェントが特定の時点で最善だと考える行動を常に活用する場合、エージェントは低い報酬の行動の後の状態について、追加的な情報を全く取得しない可能性があることが分かります。この純粋な活用では、最適な方策を見つけるためにかかる時間が長くなることがあり、また状態空間のすべてが探索されていないため、学習アルゴリズムが最適でない方策に収束することもあります。

純粋な探索での問題

一方、報酬が少なくなるリスクを冒して、時々エージェントに探索をさせた場合、エージェントは方策を新しい状態にまで広げることができます。これにより、未知のより高い報酬を発見する可能性が広がり、大域解に収束する機会が増えます。
しかし、このアプローチにも欠点があるため、エージェントの探索を大幅に増やすことは理想的ではありません。たとえば、エージェントが損害を引き起こす行動を探索するリスクがあるため、物理ハードウェアで学習する場合には、純粋な探索は良いアプローチとはいえません。例えば、高速道路でランダムにステアリング ホイールの入力を探索する場合などには、自律走行車によって大きな損害が生じることが予想されます。

しかし、損害が問題とならないシミュレーション環境でも、エージェントは状態空間の大部分を網羅するのに時間がかかるため、純粋な探索は学習の効率的な方法ではありません。これは大域解の発見には有益ですが、過度の探索が学習率を遅くするため、妥当な量の学習時間では十分な解が得られない場合があるからです。そのため、最善の学習アルゴリズムとするには、環境の探索と活用においてバランスを取らなければなりません。

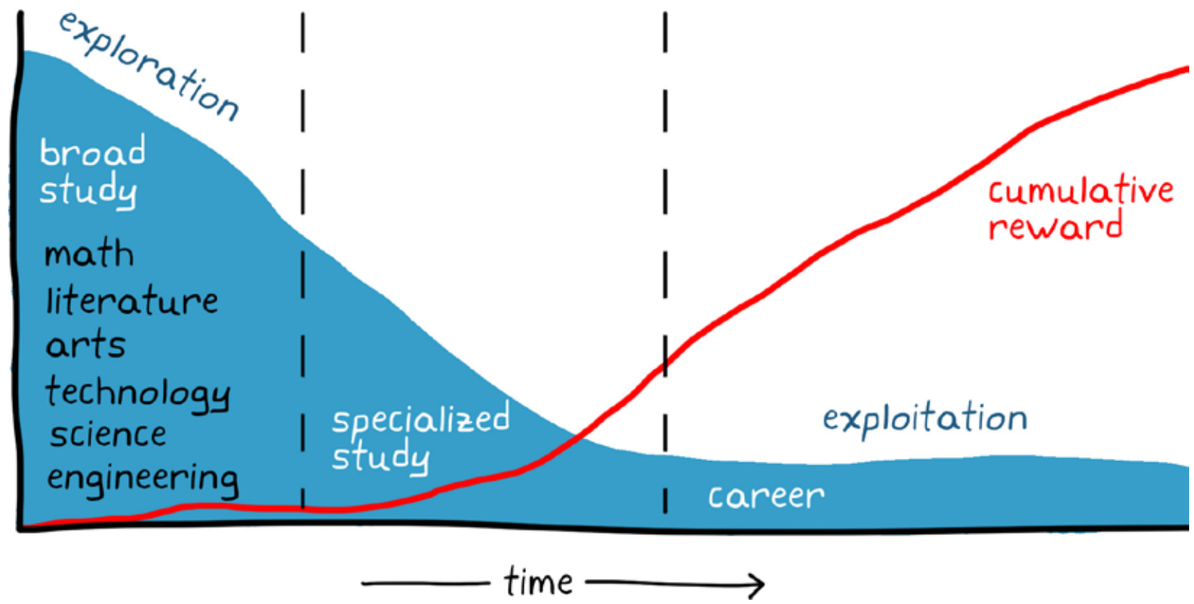
探索と活用のバランス



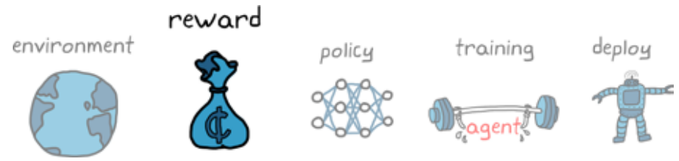
学生がキャリア パスを選ぶアプローチを考えてみましょう。学生が若い場合、彼らは異なる教科やクラスについて色々調べることに前向きで、一般的に新しい経験にも抵抗がないといえます。学生の大半は、しばらく探索した後、専門科目について詳しく学習することにし、最終的には、経済的利益と職業的満足の両方 (報酬) が最大限得られそうなキャリアを選ぶことでしょう。

可能性のあるすべてのキャリアの選択肢を探索するには、1 度の人生では足りないといえます。そのため、学生はそれまでに探索した選択肢の中から、最適なキャリア パスを決定しなければなりません。知識の活用を先送りにして新しいキャリアの選択肢を探索し続ければ、努力した結果としての収益を回収する時間があまり残らないことになります。

強化学習アルゴリズムは、探索と活用のバランスを取る単純な方法を提供しますが、学習プロセス全体でそのバランスをどこに置けばよいのかが明確でない場合があり、エージェントが学習のために割り当てられた時間内で十分な方策を決められない可能性があります。ただし、一般的には、エージェントも学生のように、学習の開始時により多くの探索を行い、終了時までに徐々に活用を増やしていきます。



価値の値

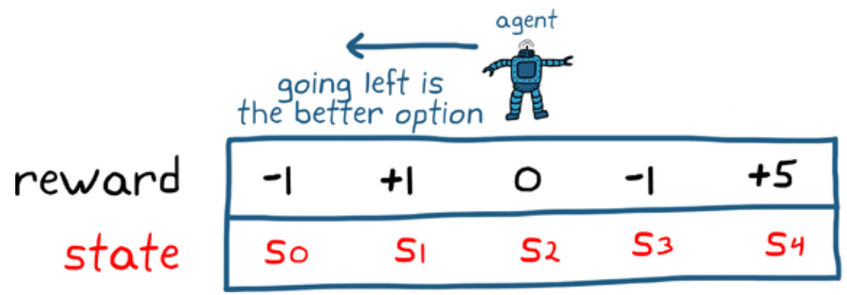


強化学習で重要な 2 つ目の側面は、*価値*の概念です。報酬ではなく、状態または行動の価値を評価することで、エージェントは短期的な利益でなく長期的に最も多くの報酬を集める行動を選択することができるようになります。

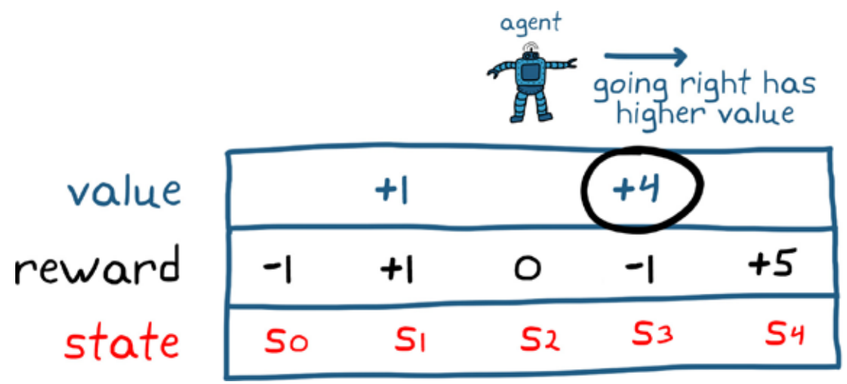
報酬: ある状態の中で、ある行動をすることの即時的な利益

価値: エージェントがある状態から未来までに期待する報酬の総量

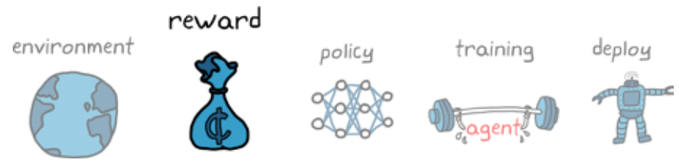
たとえば、エージェントが 2 つの行動で最大の報酬を収集しようとしている場合を考えてみましょう。エージェントが 1 つの行動による報酬のみを検討する場合、エージェントは、右より報酬が多いという理由で、まず左に行くでしょう。次にエージェントは、再び、最大の報酬であるという理由で右に戻り、最終的に合計 +1 の報酬を取得します。



しかし、エージェントが状態の価値を算出することができれば、報酬は低いものの、右に行く方が左に行くよりも値が高くなるということを、エージェントが理解できるようになります。価値を基準として行動すると、エージェントは最終的に +4 の合計報酬を得ることができます。

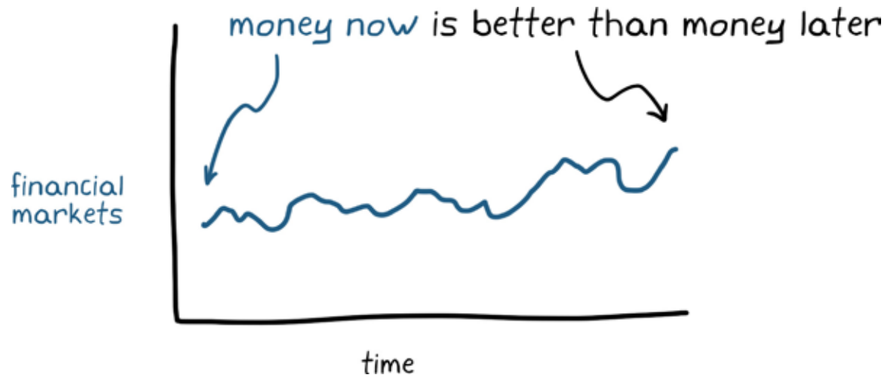


近視眼的であることの利点

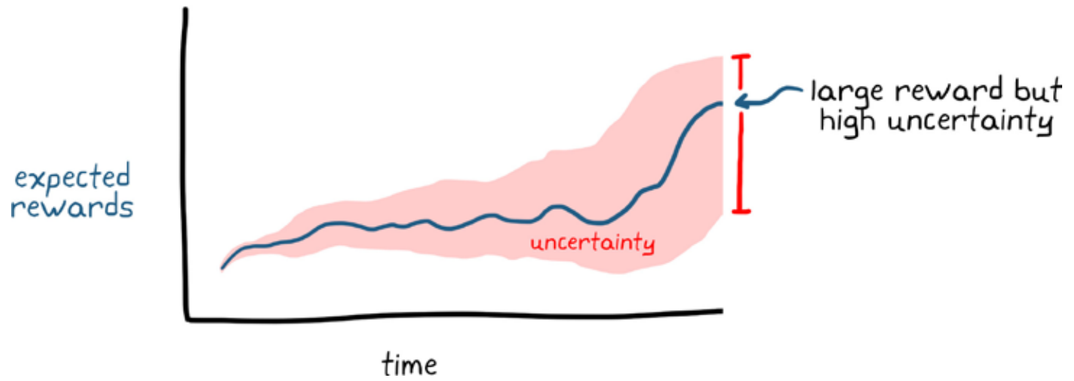


多くの連続した行動を取れば高い報酬を受け取れると約束されている場合でも、このことは最初の行動が常に最善であるということを意味していません。これには少なくとも 2 つの理由があります。

1 つ目の理由として、金融市場のように、現在持っている現金の価値が高く、1 年後に金額は増えているのに価値は下がっている、というようなことが起こりうるからです。



2 つ目の理由として、遠い未来に受け取る報酬についての予想は、信頼度が高いとはいえないからです。高い報酬はエージェントがそこに至るまでに無くなっている可能性があります。

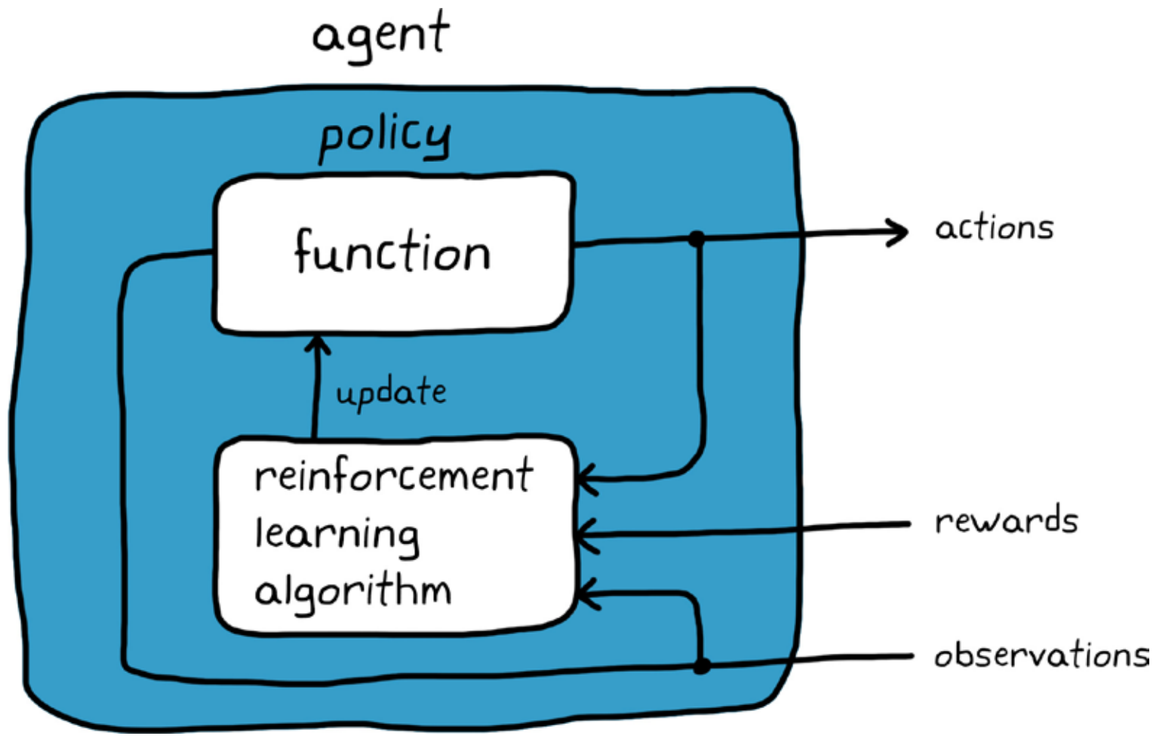


これらのケースでは、価値を算出する際に、より近視眼的であることが有利になります。強化学習では、時間的に遠くなればなるほど報酬を大きく割り引くことで、エージェントの近視眼の程度を設定することができます。これは、割引率のガンマを、0 から 1 の間で設定して行います。

$$\text{total discounted reward} = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots = \sum_{i=1}^{T} \gamma^{i-1} r_i$$

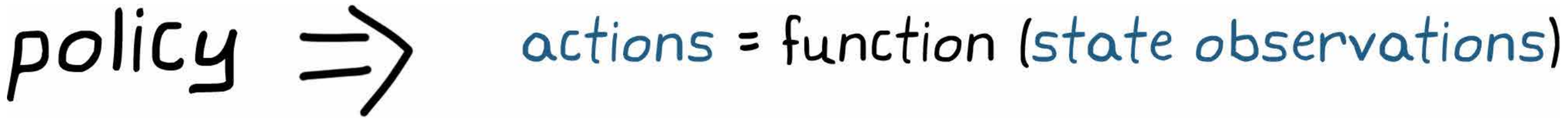
方策とは

状態と報酬を提供する環境の役割について理解が深まったところで、次は、エージェントについてご説明します。エージェントは方策と学習アルゴリズムから構成されています。方策は観測を行動にマッピングする関数であり、学習アルゴリズムは最適な方策の発見に用いられる最適化手法です。



方策の表現

方策は、最も基本的なレベルでは、状態の観測を入力とし、行動を出力する関数です。そのため、この入力と出力を持つ関数はすべて、方策を表現する方法として機能します。



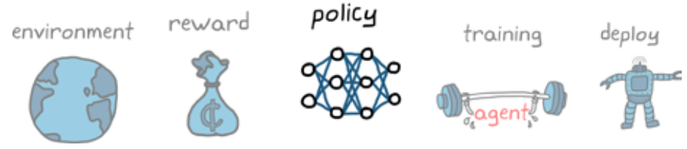
一般的には、方策関数を構築するには 2 つのアプローチがあります。

- 直接的なアプローチ: 状態の観測と行動の間に特定のマッピングを行います。
- 間接的なアプローチ: 価値などの他の指標を見て最適なマッピングを推論します。*

次の数ページで、方策の表現に使用できる異なる種類の数学的構造を、価値ベースの手法を用いてハイライトする方法について説明します。これらの構造は、方策ベースの関数にも適用される可能性があることに注意してください。

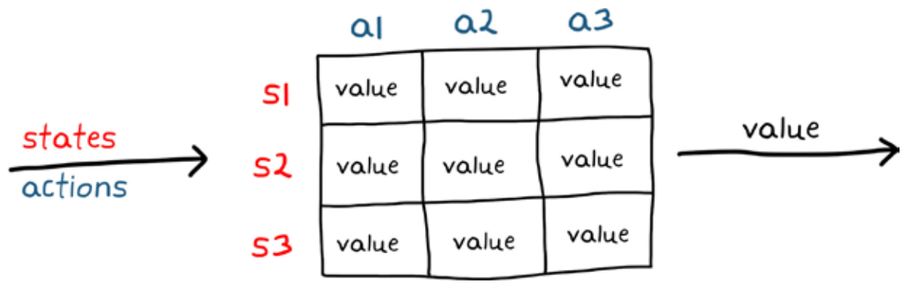
** 第 3 のアプローチとして、直接方策のマッピングを行い且つ価値ベースのマッピングも行うという両方の良さを組み合わせた Actor-Critic と呼ばれる手法があります。これについては後述します。*

方策をテーブルで表現

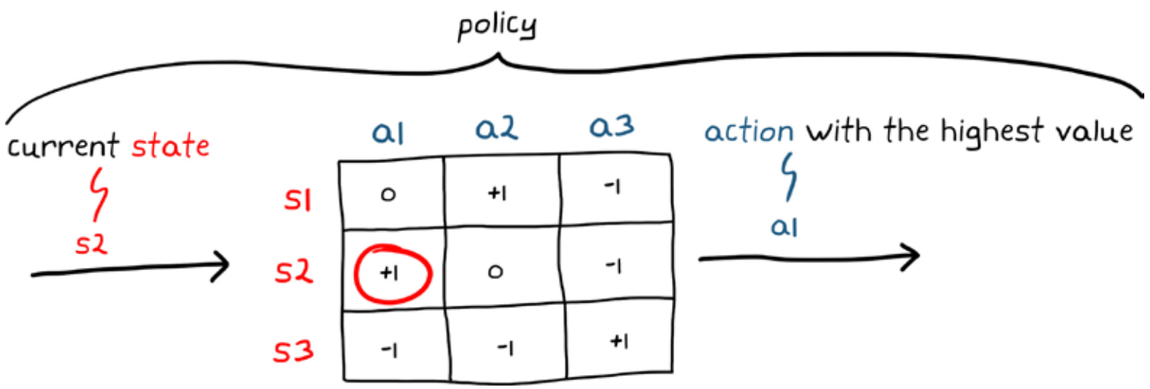


環境の状態空間と行動空間が離散で、かつそれらの数が少なければ、方策の表現に単純なテーブルを使用することができます。

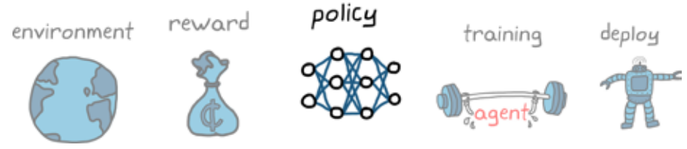
テーブルはよくある形式のもので、入力が検索アドレスとして働き、出力はテーブル内の対応する数値になります。テーブル形式の関数の 1 つは Q テーブルで、Q テーブルは状態と行動を価値に関連付けます。



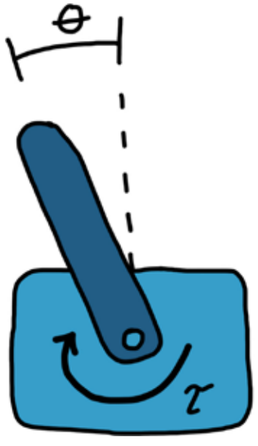
Q テーブルでは、方策は、現在の状態における可能な行動の価値をすべて確認し、最も高い価値を持つ行動を選択することになります。Q テーブルでのエージェントの学習は、テーブルでのそれぞれの状態/行動のペアに対する正しい価値の判断で構成されます。すべて正確な価値でテーブルを埋めることができれば、長期的に最も高い収益を生み出す行動の選択はとても簡単です。

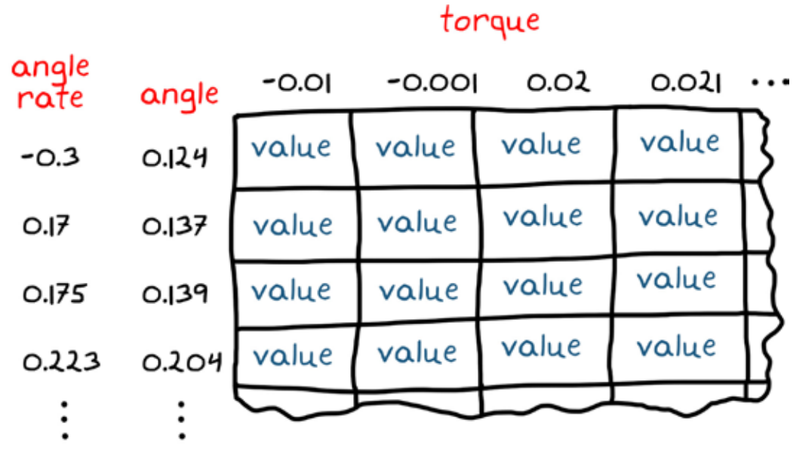


連続した状態/行動空間



状態/行動のペアが膨大または無限である場合、方策 パラメーターのテーブルでの表現は不可能です。これを理解するために、倒立振子を制御する方策について考えてみましょう。振子の状態は、-π から π までのどの角度、どの角速度にもなる可能性があります。また行動空間は、負の限界から正の限界までのすべてのモータートルクです。すべての状態と行動のすべての組み合わせをテーブルで記録することは不可能です。



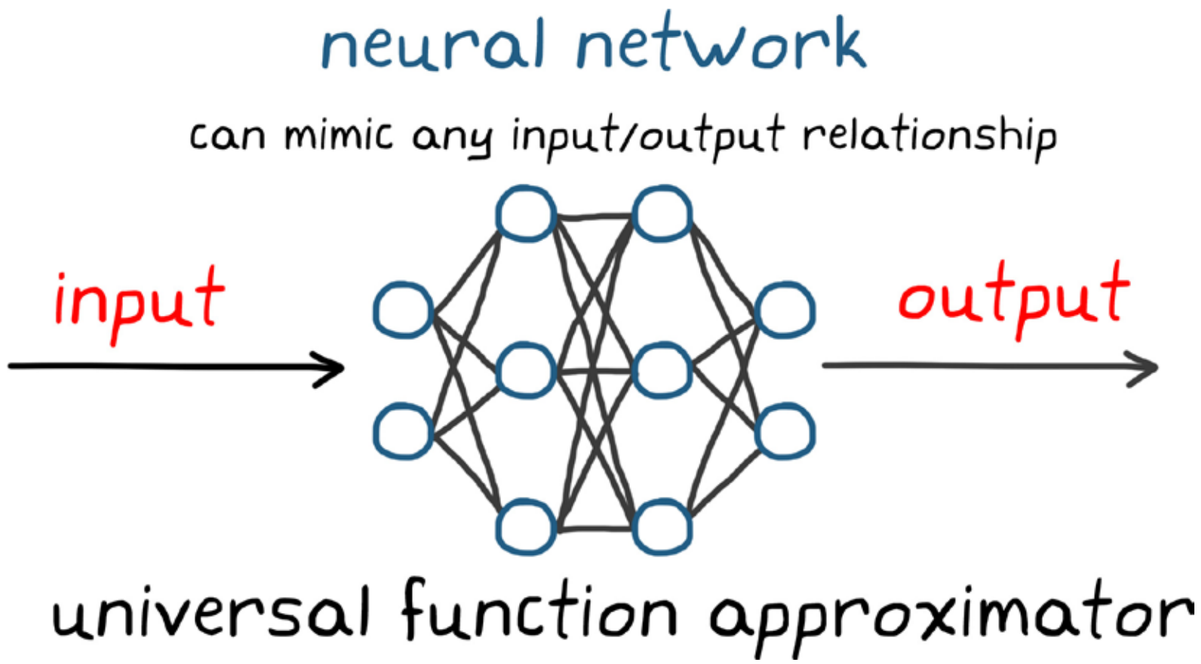


倒立振子の連続した性質は、状態を入力とし、行動を出力するある連続関数で表現することができます。しかし、この関数で正しいパラメーターを学習する前に、論理構造を定義する必要があります。自由度の高いシステムや非線形システムでは、これが難しくなる場合があります。

そのため、連続した状態と行動を扱える関数を表現でき、さらにあらゆる環境に対しても論理構造の構築が比較的簡単な方法が求められます。ここで、ニューラルネットワークが登場します。

普遍的な関数近似器

*ニューラルネットワーク*はノードの集まり、つまり人工ニューロンであり、普遍的な関数近似器として機能させる方法として関連付けされます。すなわち適切なノードと接続の組み合わせが与えられれば、入力と出力の関係を模擬するネットワークを構築することができます。ニューラル ネットワークの普遍性により、関数が極端に複雑であっても、それを実現できる何らかのニューラル ネットワークが存在するはずです。

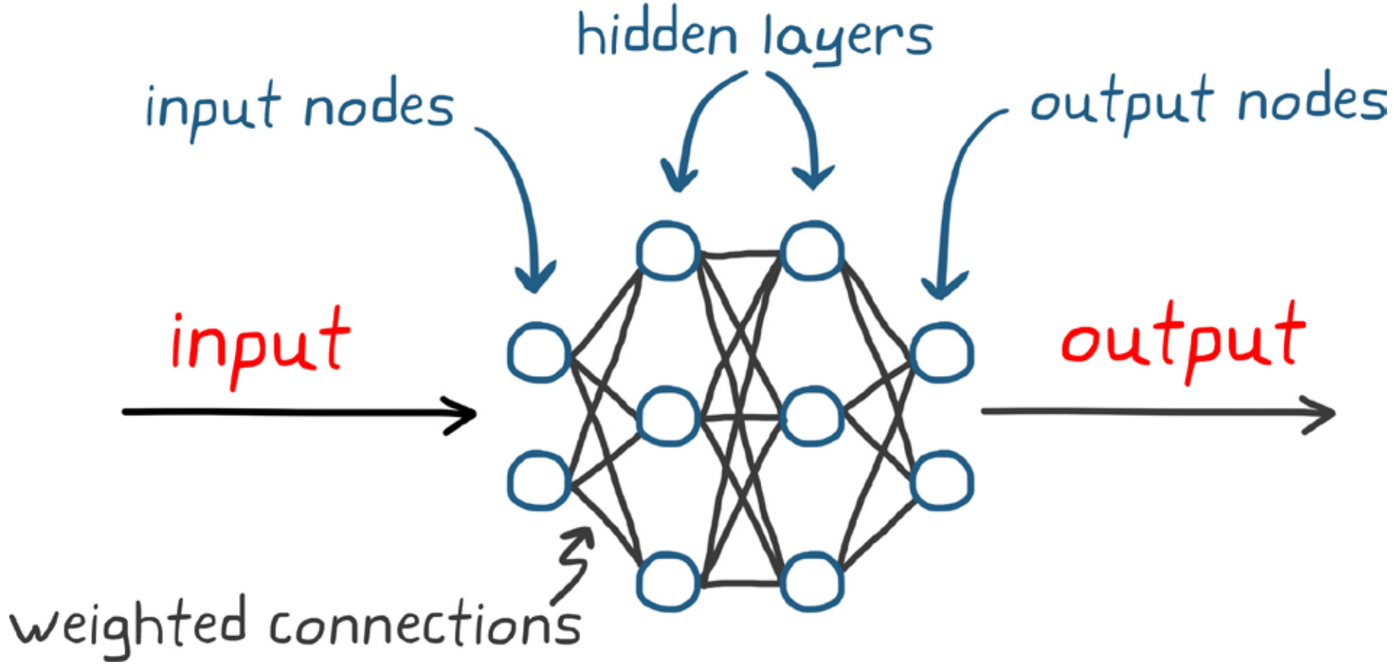


そのためニューラル ネットワークを用いれば、特定の環境で機能する完璧な非線形関数構造を見つける必要はなく、ノードと接続の同じ組み合わせを多くの異なる環境で使用することができます。唯一の違いは、パラメーター自体です。学習の過程では、最適な入力/出力の関係を見つけるための体系的なパラメーター調整によって構成されます。

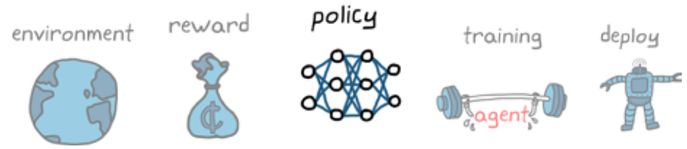
ニューラル ネットワークとは

ニューラル ネットワークの計算については、ここでは詳しく取り上げませんが、後述する、どのように方策を設定するかを理解するのに役立つ事項をいくつかご説明します。

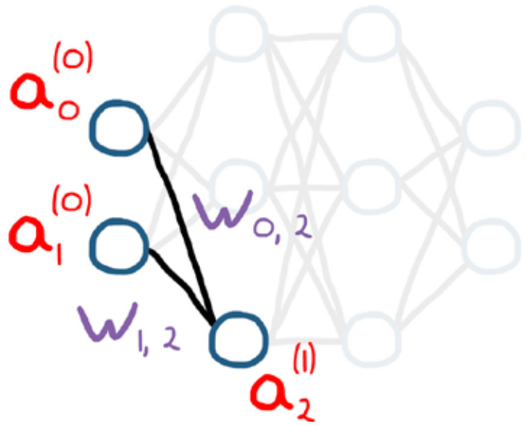
左側が入力ノードで、関数への各入力に対し 1 つの入力ノードが対応し、右側が出力ノードを表します。これらの間にあるのが、隠れ層と呼ばれるノードの列です。このネットワークには、2 つの入力、2 つの出力、およびそれぞれ 3 つのノードを持った 2 つの隠れ層があります。完全に接続されたネットワークでは、各入力ノードから次の層の各ノードへの重み付けされた接続があり、その後は、そのノードからその後の層への接続というように、出力ノードに達するまでこれが繰り返されます。



ネットワーク内で行われている計算



あるノードの値は、そこへの入力となるノードにそれぞれの重み係数を掛けたものの和に、バイアスを足した値と等しくなります。



$$\text{value of } a_2^{(1)} = w_{0,2} \cdot a_0^{(0)} + w_{1,2} \cdot a_1^{(0)} + b_2^{(1)}$$

(layer: (1), node position: 2)

層のすべてのノードでこの計算を実行しますが、これを線形方程式系として簡潔な行列形式で書くことができます。このような行列操作を基本として、ある層にあるノードの数値を次の層にあるノードの値へと変換します。

transform from layer 0 to layer 1

$$a^{(1)} = W_0 a^{(0)} + b^{(1)}$$

matrices

transform from layer 1 to layer 2

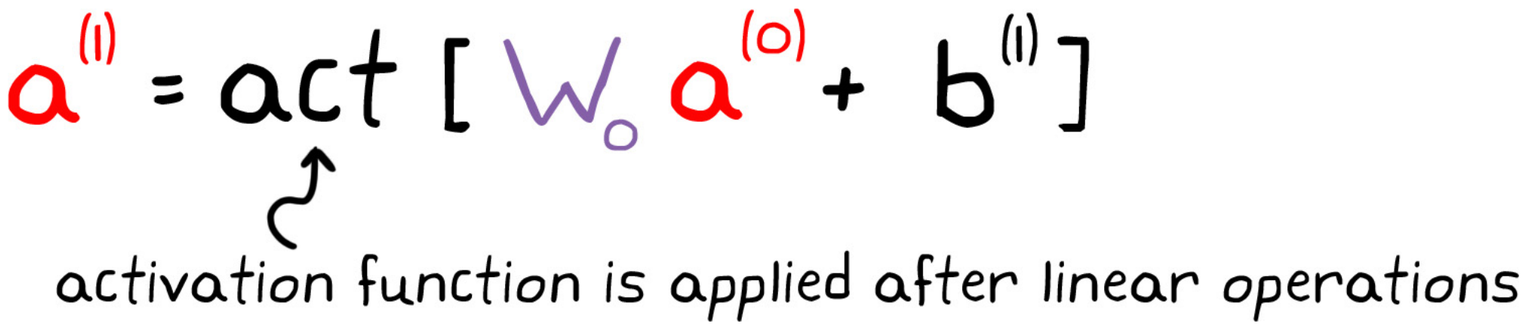
$$a^{(2)} = W_1 a^{(1)} + b^{(2)}$$

transform from layer 2 to layer 3

$$a^{(3)} = W_2 a^{(2)} + b^{(3)}$$

不可欠な手順

次々に伝搬する膨大な線形方程式がどのように普遍的な関数近似器として機能するのでしょうか。特に、それらはどのように非線形関数を表現するのでしょうか。人工のニューラル ネットワークの最も重要な特徴の 1 つとなっている手順が存在します。ノードの値が計算された後で、次の層への入力として与えられる前にノードの値を変更する活性化関数が適用されます。

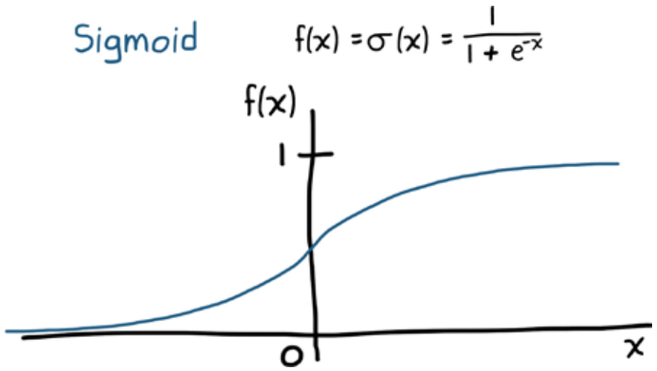


活性化関数には様々なものが存在します。それらすべてに共通することは非線形であることで、任意の関数に対して近似できるネットワークを作る上で非常に重要です。それはなぜでしょうか。これについては、多くの非線形関数は、活性化関数の出力の重み付けされた組み合わせに分解できるからです。

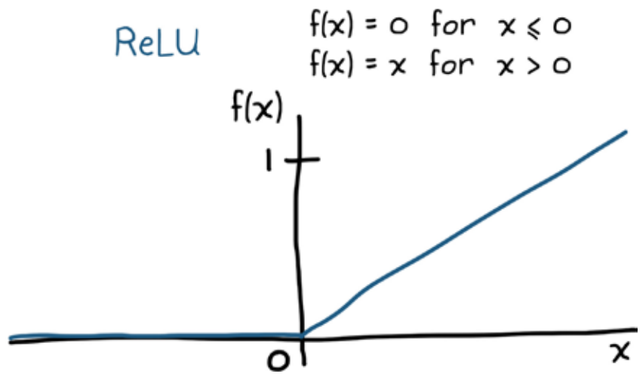
詳細については、*「普遍性定理の可視化」*をご覧ください。

ReLU とシグモイド活性化関数

シグモイド活性化関数は、負の無限大から正の無限大までのどの入力も、0 から 1 までの間に圧縮する滑らかなカーブを生成します。



正規化線形ユニット (ReLU) 関数ではすべての負のノード値を 0 にして、正の値はそのままにします。

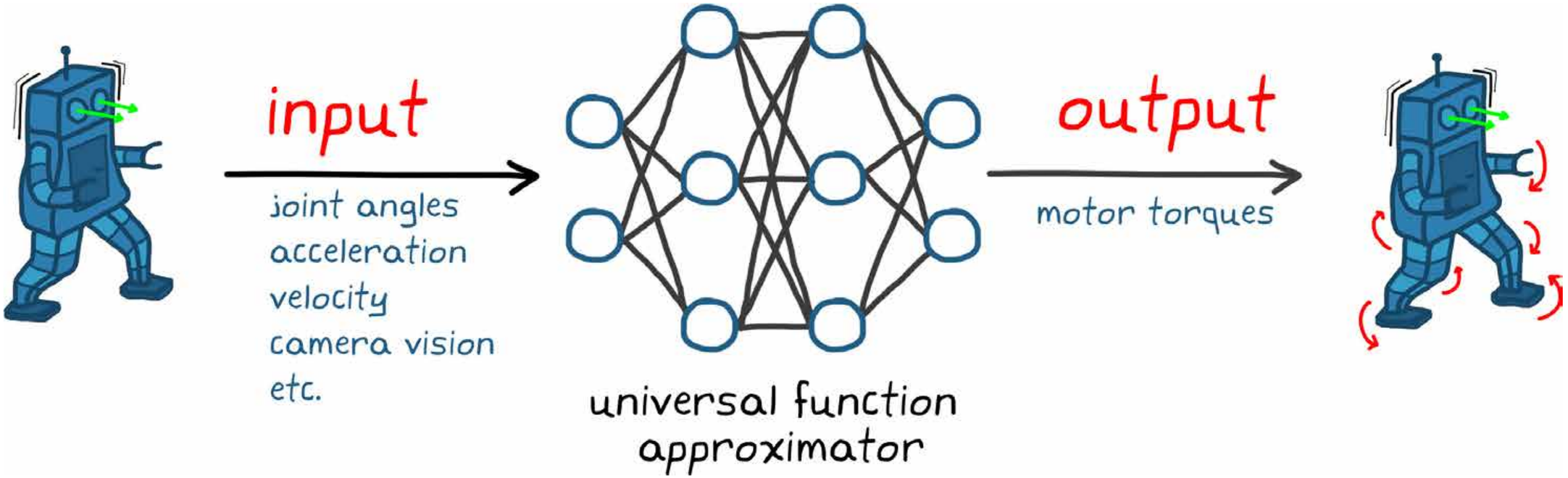


たとえば、 活性化前のノード値の -2 は、シグモイド活性化では 0.12 になり、ReLU 活性化では 0 になります。

preactivation node value	postactivation sigmoid	postactivation ReLU
-2	0.12	0
-1	0.27	0
1	0.73	1
2	0.88	2

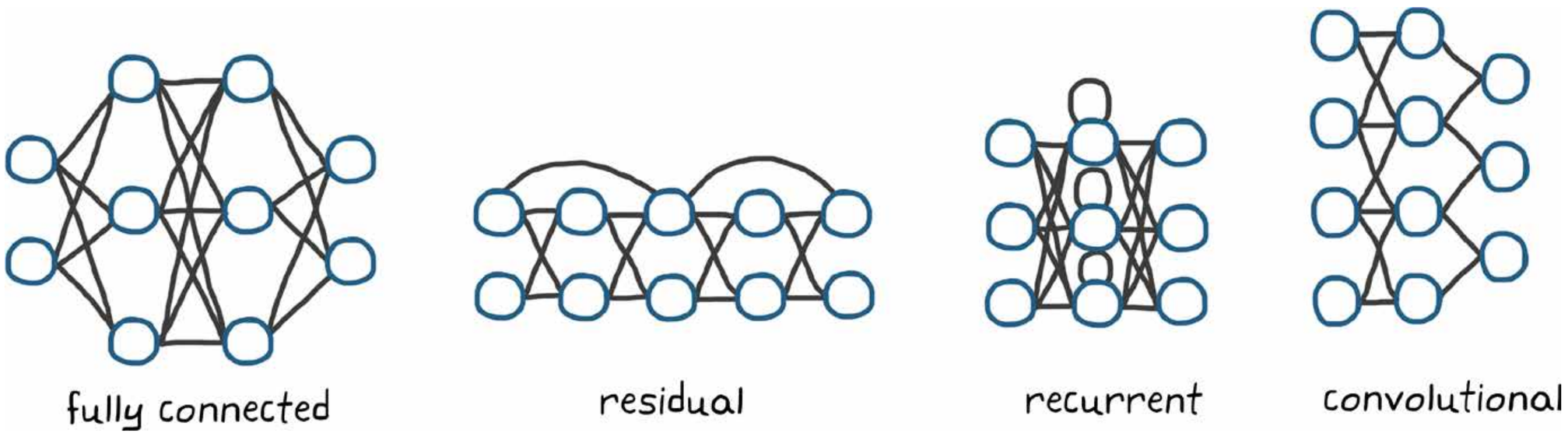
方策をニューラル ネットワークで表現

次に進む前にこれまでの内容を復習してみましょう。まず、多数の観測を取り込み、それらを非線形環境を制御する一連の行動に変換する関数を発見することが目標でした。この関数の構造は直接解くには複雑過ぎることがよくあるため、時間をかけて関数を学習するニューラル ネットワークで近似しようと考えています。そして、任意のニューラルネットワークを当てはめ、強化学習アルゴリズムに重みとバイアスの正しい組み合わせを発見させようと考えがちですが、残念ながら、これがうまくいきません。



ニューラル ネットワークの構造

求める関数に近似できる程度に複雑であって、しかし学習が不可能または不可能といえるほど重たくない程度であるようにするため、前もってニューラル ネットワークについていくつかの選択をする必要があります。たとえば、ここまで見てきたように、活性化関数、隠れ層の数、および各層のニューロン数を選択する必要があります。また、ネットワークの内部構造も管理しなければなりません。最初のネットワークのように完全に接続されたネットワークにするべきか、または残差ニューラル ネットワークのように層の接続をスキップすべきか。再帰型ニューラルネットワークを使ってノード自身にループバックして内部に記録を残すべきか。畳み込みニューラル ネットワークによってニューロンのグループを共同で動作させるべきか。

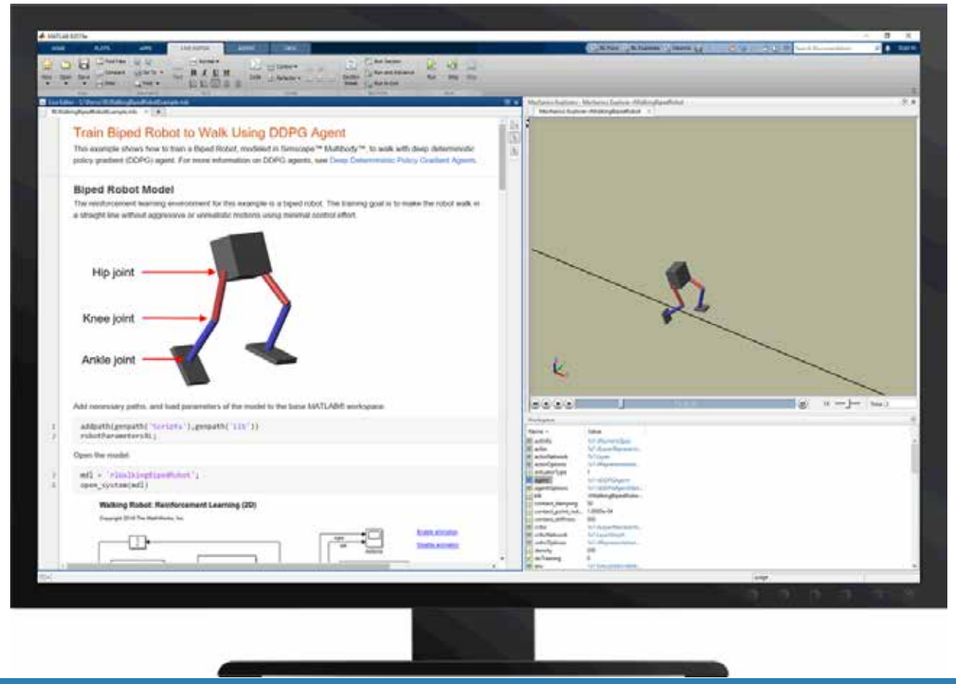


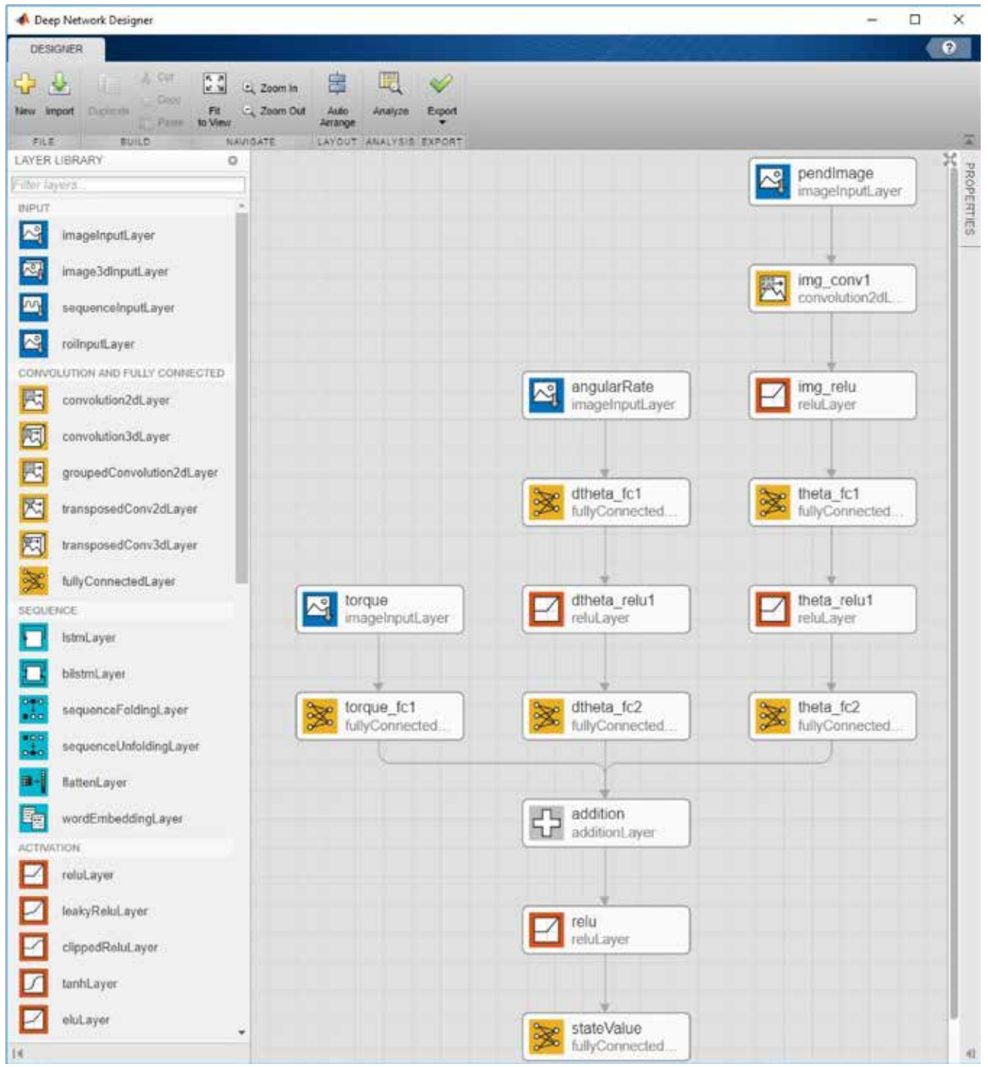
他の管理手法と同様、ニューラル ネットワークの構造を決めるための正しいアプローチというものはありません。多くの場合、解こうとしている問題と同じような問題で既に適用されているネットワーク構造から始め、それを少しずつ変えていくという方法に帰着します。

MATLAB による強化学習

Reinforcement Learning Toolbox™ には、強化学習アルゴリズムを使用した方策の学習のための関数とブロックが用意されています。これらの方策を使用して、ロボットや自律システムなどの複雑なシステムのためのコントローラーと意思決定アルゴリズムを実装できます。

ツールボックスでは、ディープ ニューラル ネットワーク、多項式、またはルックアップテーブルを使用して方策を実装できます。MATLAB® または Simulink® モデルで表現された環境との対話を可能にすることで、方策を学習させることができます。

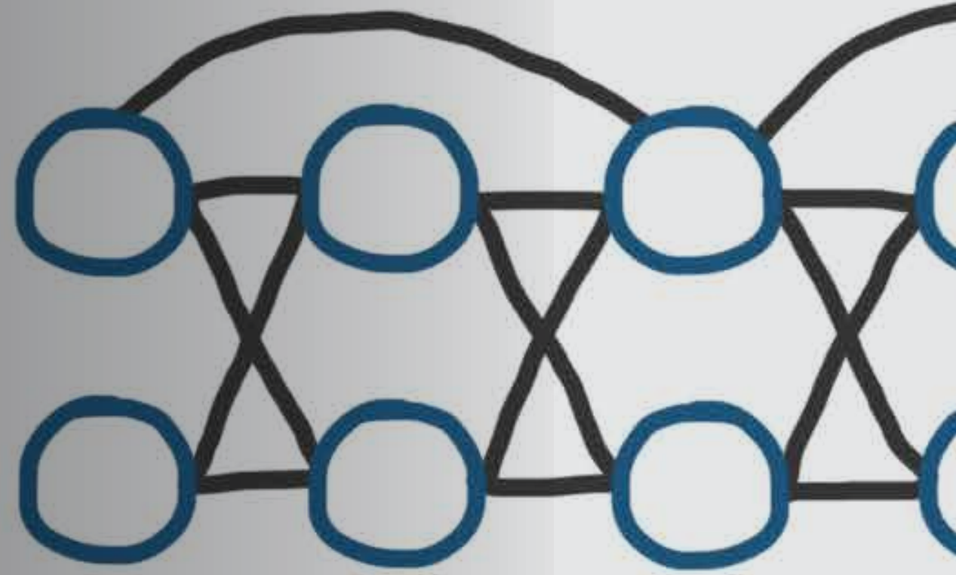




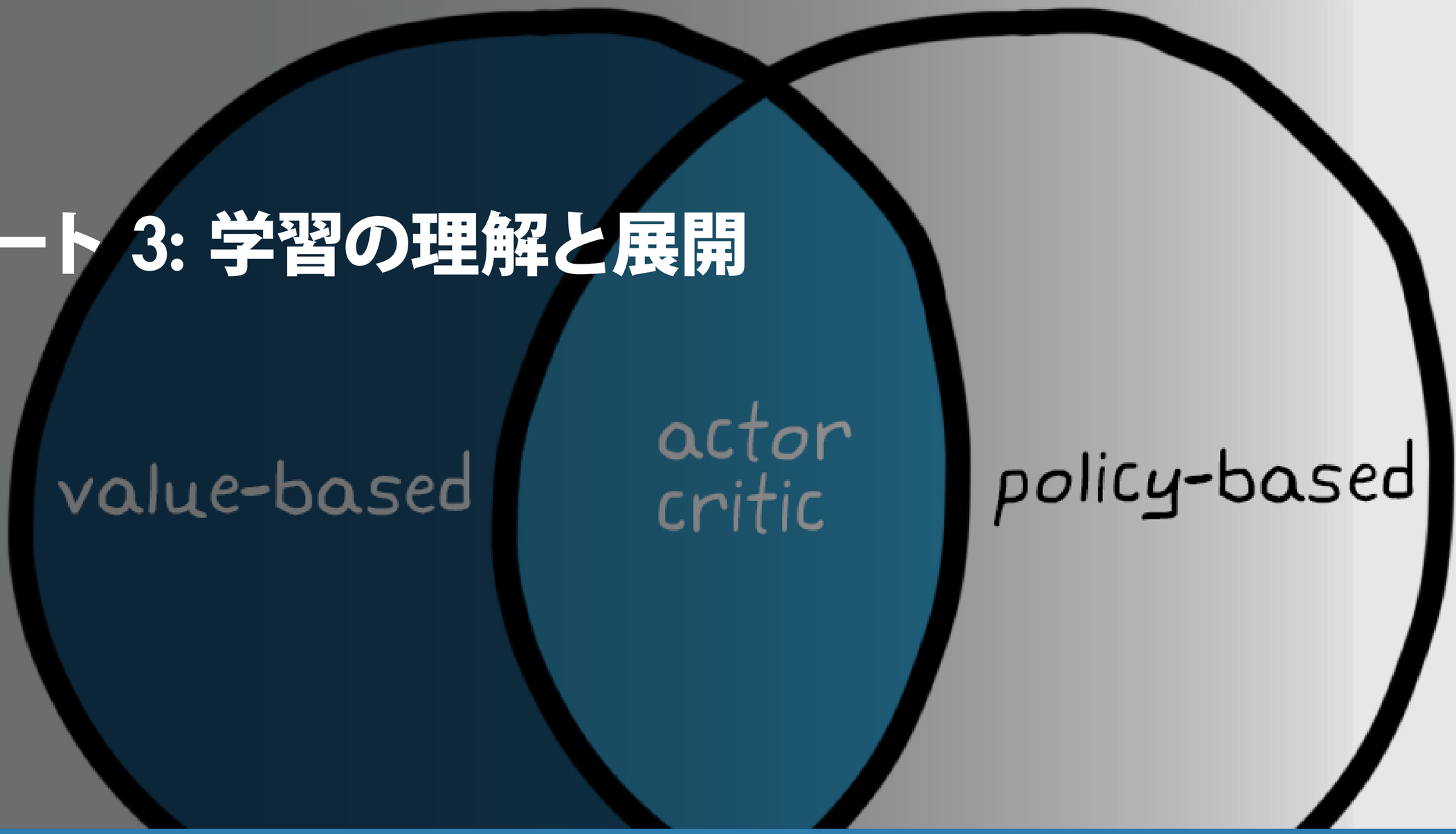
ディープ ネットワーク デザイナー・アプリで作成された Deep Q-learning network (DQN) エージェント

関連情報

Reinforcement Learning Toolbox - 概要
方策と学習アルゴリズムの理解 (17:50) - ビデオ
MATLAB および Simulink での報酬信号の定義 - ドキュメンテーション
方策と価値関数の表現 - ドキュメンテーション
入門者向け参照例 - 例

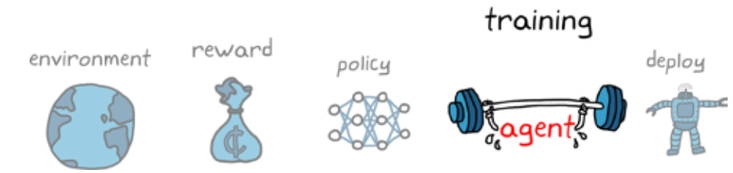


パート 3: 学習の理解と展開

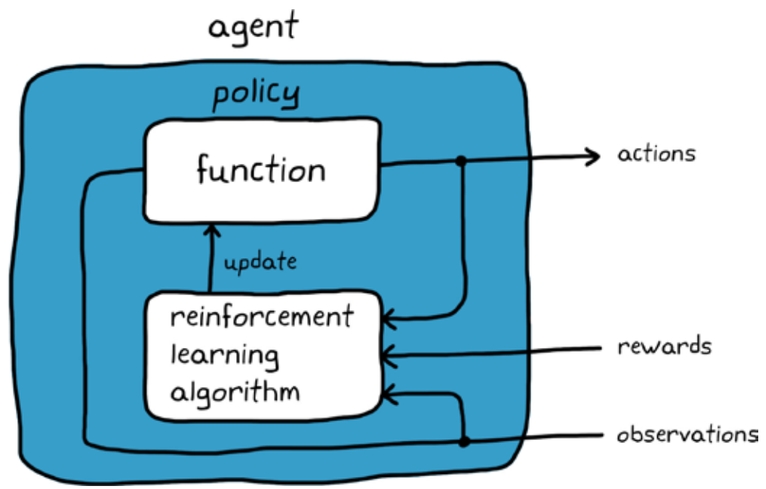


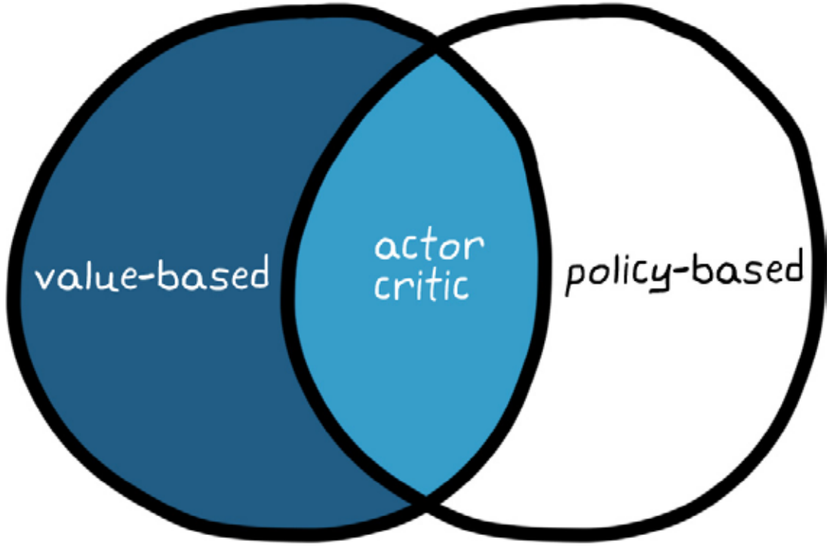
方策の構造

強化学習 (RL) アルゴリズムでは、ニューラル ネットワークがエージェントの方策を表現します。方策の構造と強化学習アルゴリズムは密接に関係しています。強化学習アルゴリズムを選択することなく方策の構造を決めることはできません。



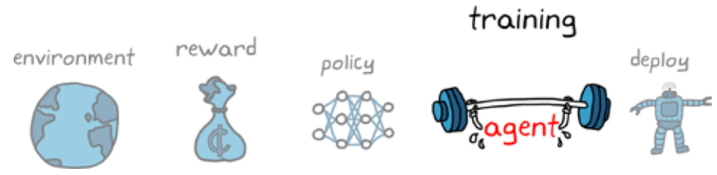
the policy structure and the RL algorithm are intimately intertwined





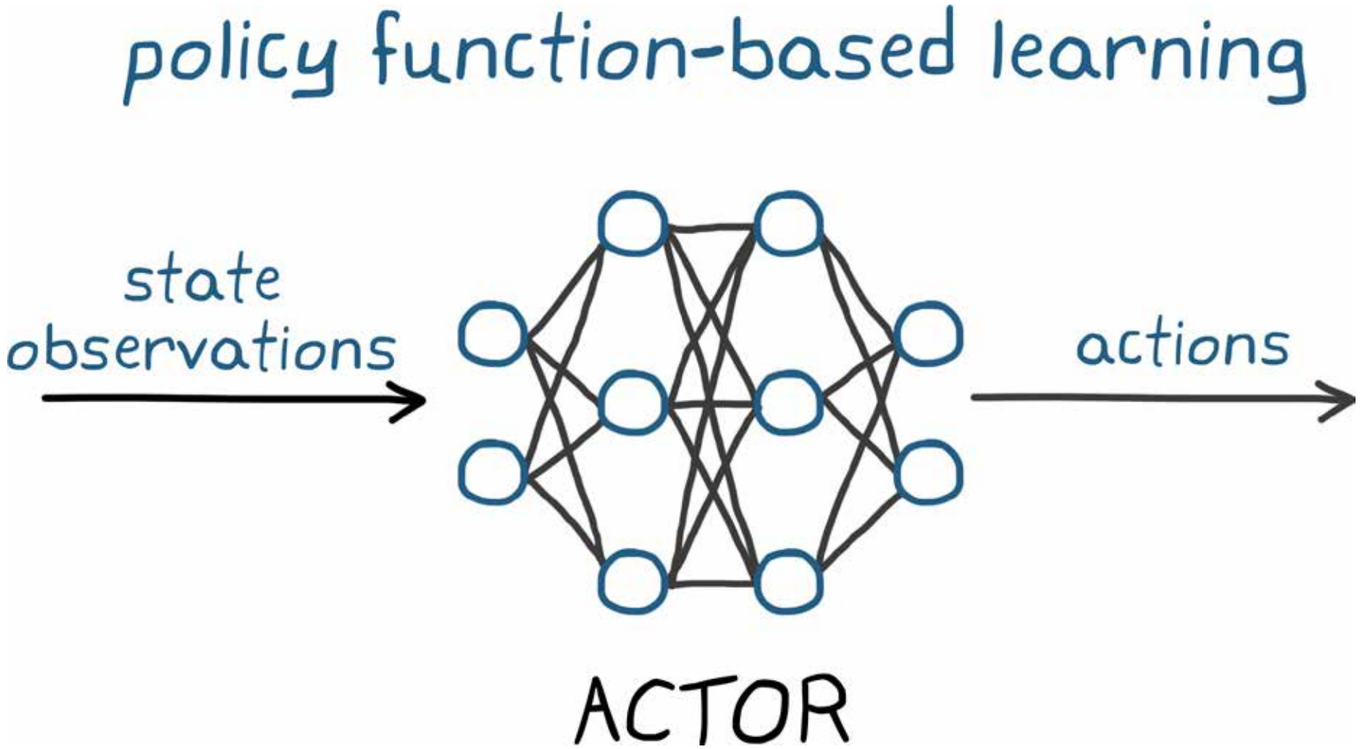
次の数ページで、方策関数ベース、価値関数ベース、およびactor-criticの強化学習へのアプローチについて解説しながら、方策構造の違いを明確にしていきます。説明を単純にしすぎていると感じられるかもしれませんが、方策の構造化の手法について基本を理解するために、単純なところから始めてみましょう。

方策関数ベースの学習

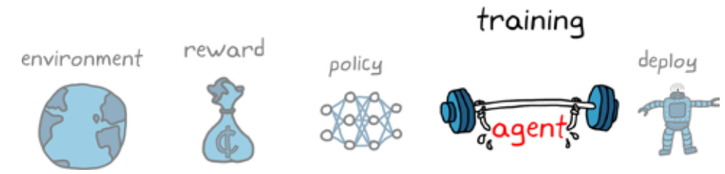


方策関数ベースの学習アルゴリズムは、状態の観測を入力とし、行動を出力するニューラル ネットワークを学習させます。ニューラル ネットワークが方策全体となるため、*方策関数ベースのアルゴリズム*と呼ばれます。ニューラル ネットワークは、エージェントに直接取るべき行動を伝えるため、actorと呼ばれます。

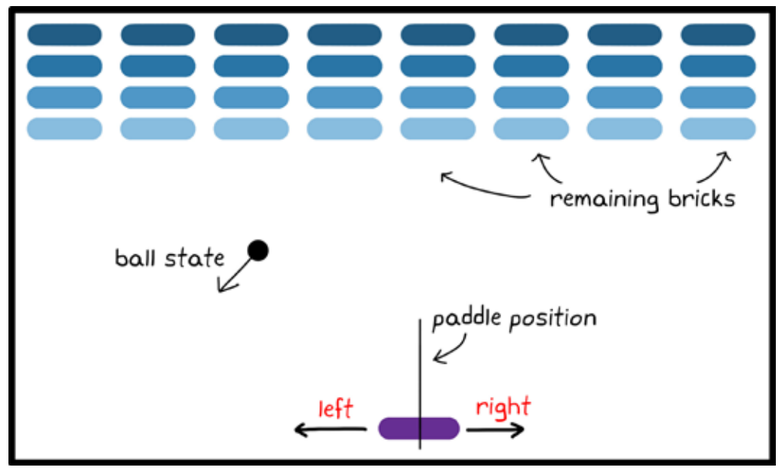
ここでの疑問は、このニューラル ネットワークの学習にどうアプローチすればよいかということです。これがどのように行われるのかを実感するため、Atari ゲームの Breakout を見てみましょう。

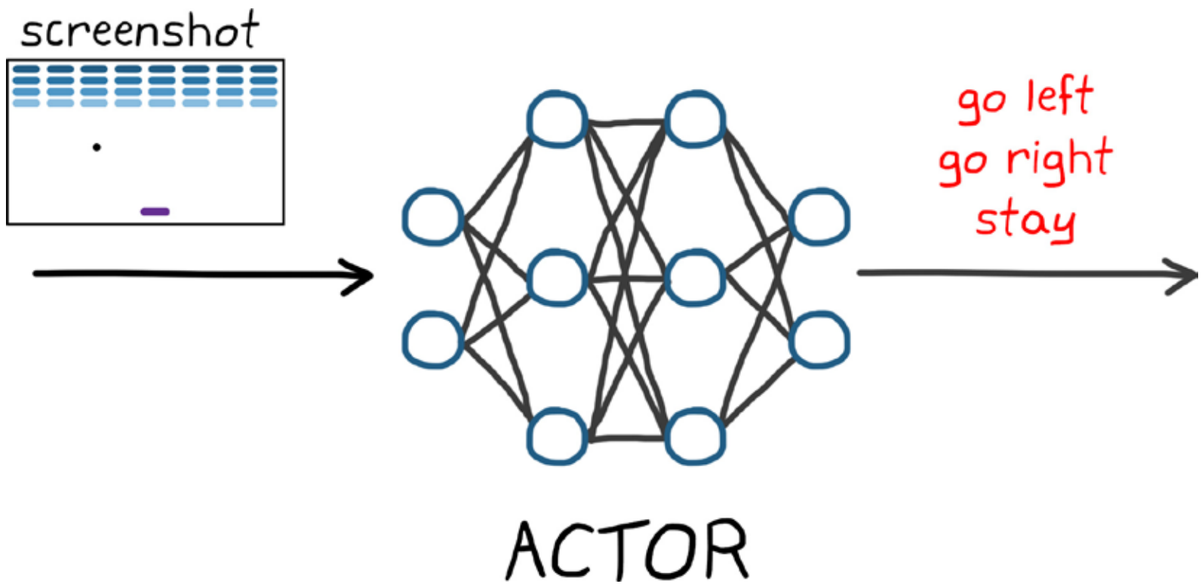


Breakout の学習における方策でのアプローチ



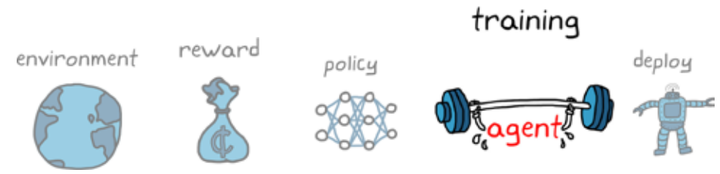
Breakout はラケットを使ってバウンドするボールを跳ね返し、ブロックを消していくゲームです。このゲームには、ラケットを左に動かす、右に動かす、または動かさないの 3 つの行動があります。またラケットの位置、ボールの位置と速度、そして残りのブロックの場所など、連続に近い状態空間があります。



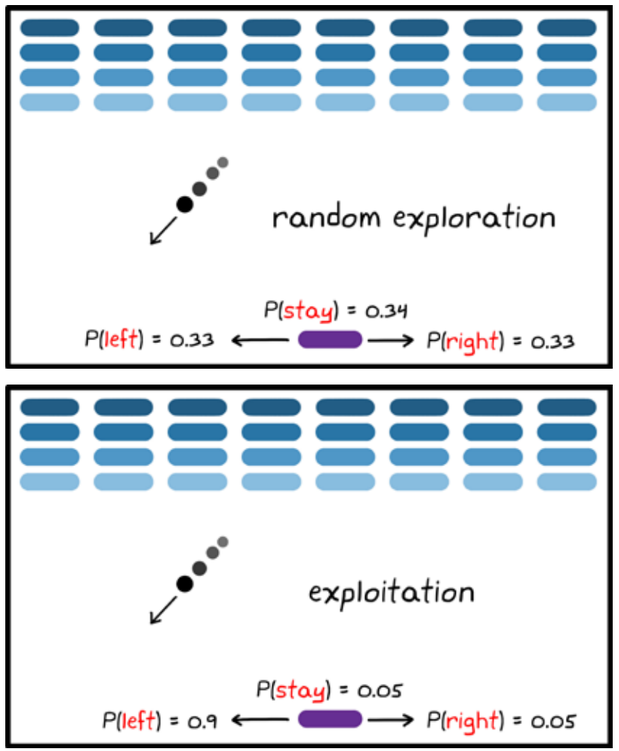


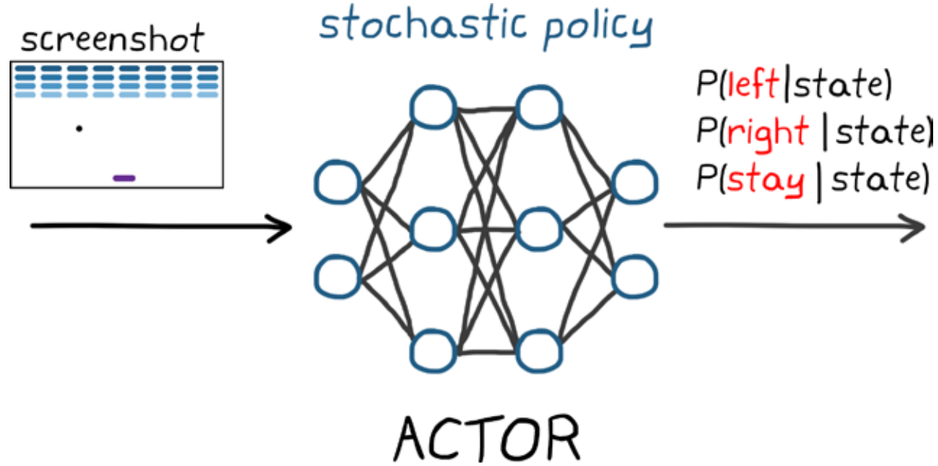
この例では、actorネットワークへの入力は、ラケット、ボール、およびブロックの状態です。出力は、左、右、および静止という、行動を表現するノードです。手作業で状態を計算してネットワークに入力するかわりに、ゲームのスクリーン ショットを入力すれば、ネットワークにその画像中のどの特徴量が出力を決定するにあたり最も重要かを学習させることができます。actorは数千のピクセルの輝度を 3 つの出力にマッピングします。

確率的方策



ネットワークの準備ができたら、次はそれを学習させるアプローチについて見ていきましょう。方策勾配法は、それ自体に多くのバリエーションがある幅広いアプローチの 1 つです。方策勾配は確率的方策で動作させることができます。そのため、確定的な「左に動く」のような行動を生成するかわりに、方策は左に動く確率を出力します。確率は 3 つの出力ノードと直接関連しています。



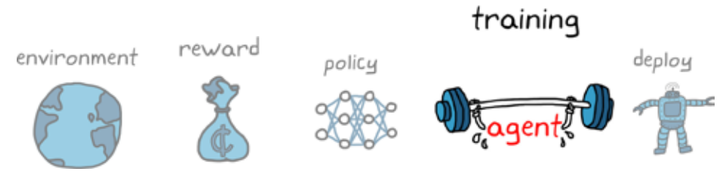


エージェントは、既に分かっている最大の報酬を与える行動を選択することで環境を活用すべきか、または未知の環境の一部を探索する行動を選択するべきでしょうか?。

確率的方策は、探索を確率に組み入れることにより、このトレードオフに対処します。こうすると、エージェントは学習時、その確率を更新するだけでよくなります。左に動くことは右に動くことより良い選択肢かどうかを考えます。これが正しい場合、この状態で左に動く確率を高くします。

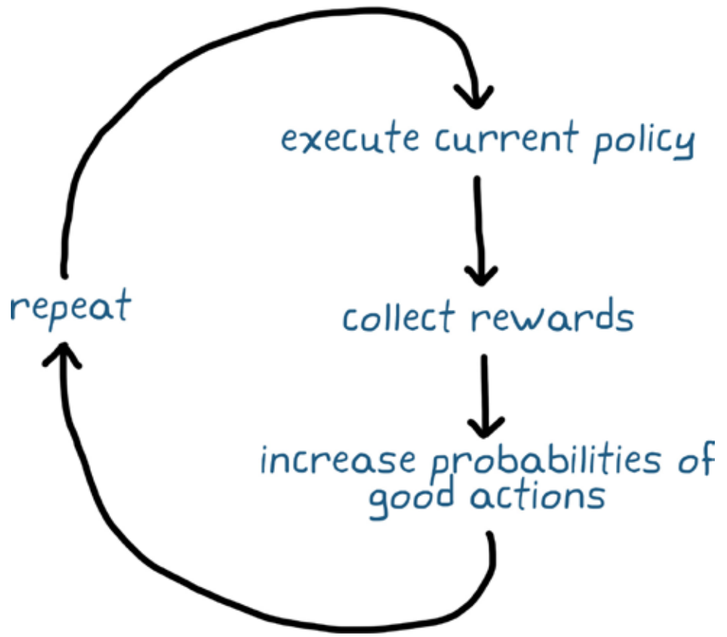
時間と共に、エージェントはこれらの確率を報酬が最も高くなる方向へ少しずつ動かしていきます。最終的に、すべての状態において最も有利な行動に非常に高い確率が付けられるので、エージェントは常にその行動をとるようになります。

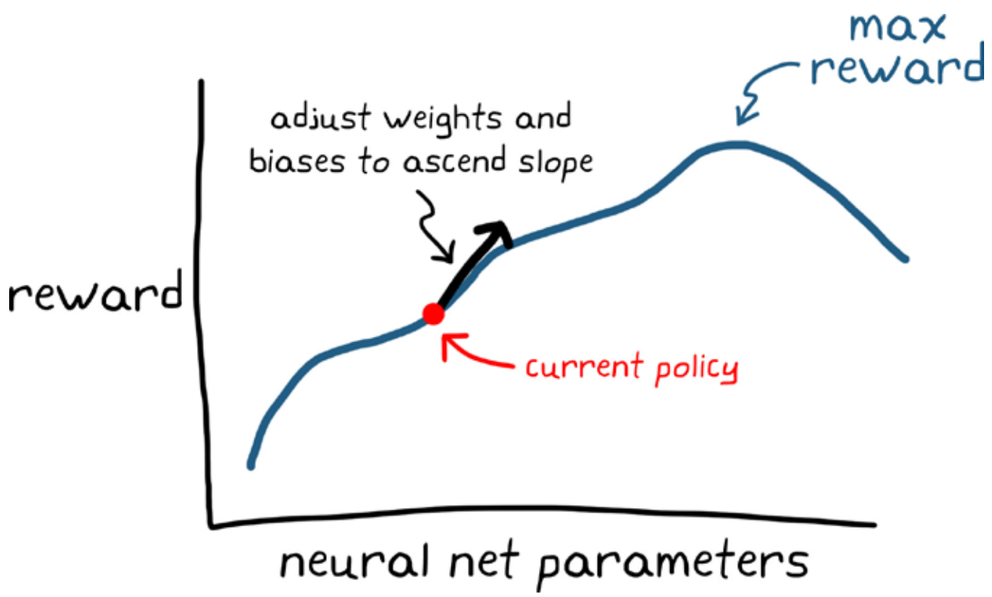
方策勾配法



それでは、エージェントはどのように行動が良いかまたは悪いかを理解するのでしょうか。次のように考えていきます。現在の方策を実行し、途中で報酬を集め、そしてネットワークを更新してより高い報酬につながる行動の確率を高くします。

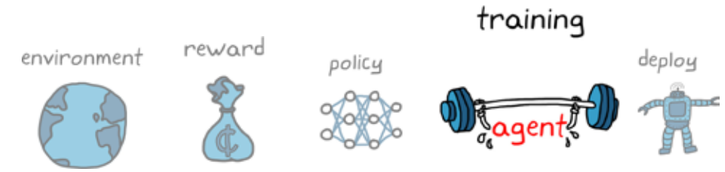
ラケットが左に動けば、ボールが取れず負の報酬になるので、ニューラル ネットワークを更新して、エージェントが次にこの状態なった時にラケットを右に動かす確率を高めます。



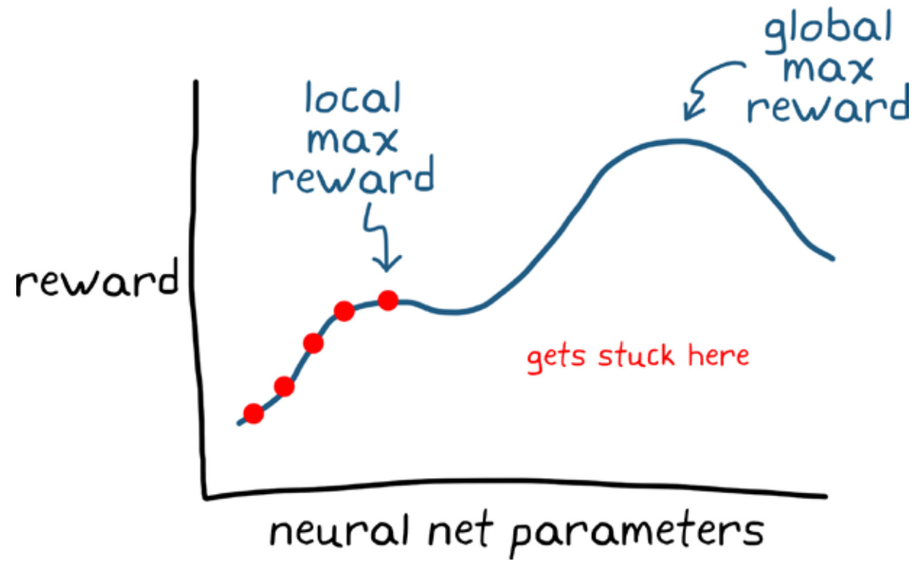


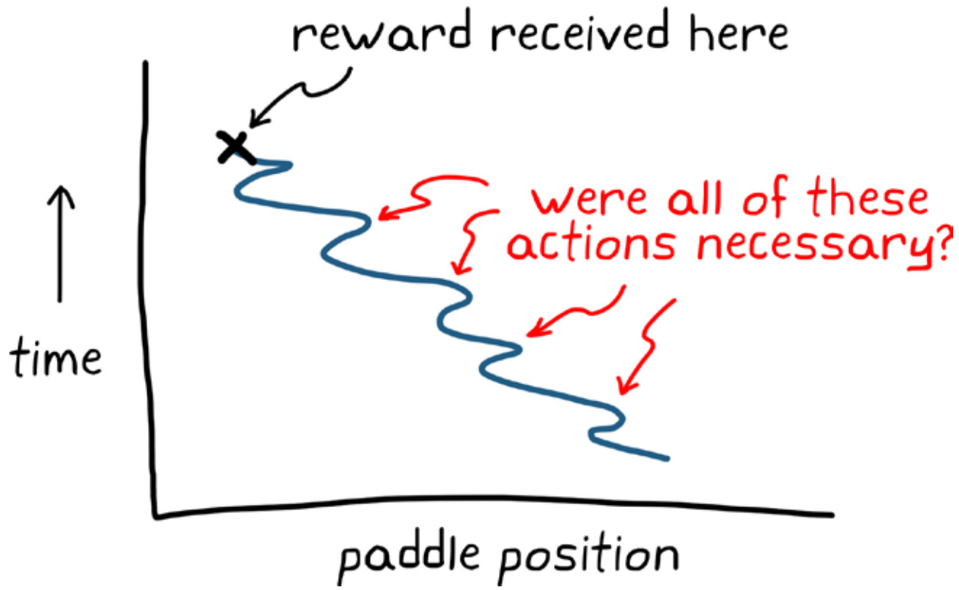
ネットワーク内の重みとバイアスの報酬に関する微分係数を求め、それらを正の報酬が増加する方向へ調整します。この方法で、学習アルゴリズムはネットワーク内の重みとバイアスを動かして、報酬の傾斜を登ります。これが名称に勾配という言葉が使われている理由です。

方策勾配法の欠点



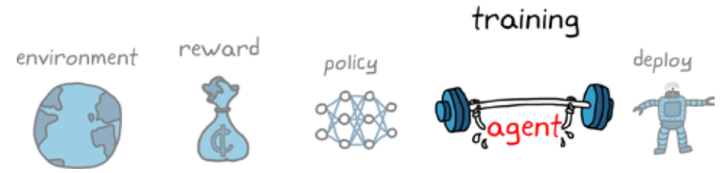
方策勾配法の欠点の 1 つは、最急上昇をする方向へ動くだけの単純なアプローチは、大域的ではなく局所的最大値に収束する可能性があることです。方策勾配法は、ノイズの多い測定に敏感なため、収束が遅くなることもあります。たとえば、多くの連続した行動を取った後に一つの報酬を得ると、結果として得られる報酬和の分散がエピソードの間で大きくなります。



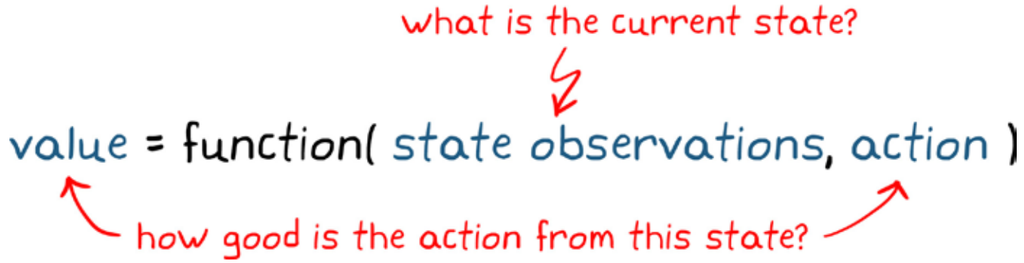


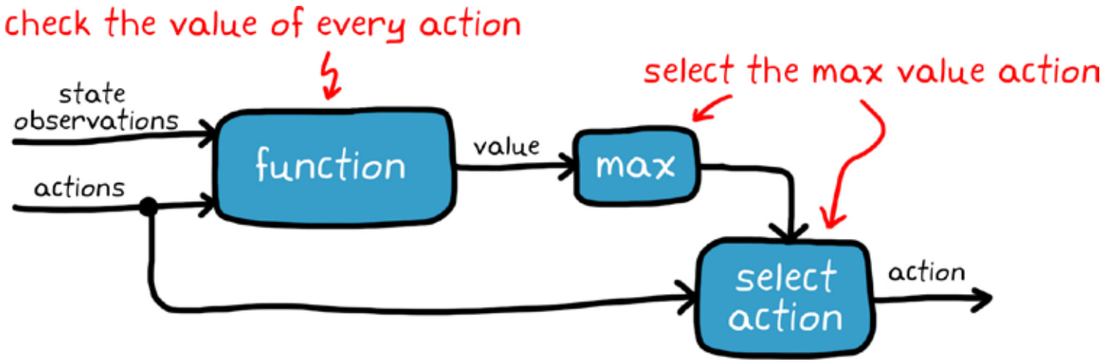
例えばBreakout では、エージェントがラケットを左右に素早く何度動かそうとも、結局のところラケットはフィールドを上手く動き、ボールを打ち返し、報酬を得ることができます。エージェントは、これらの一つ一つの左右への動きが報酬を得るために実際に必要であったのかどうかを知ることはできません。そのため、方策勾配ではそれぞれの行動が必要であるかのように扱い、それに応じて確率を調整します。

価値関数ベースの学習



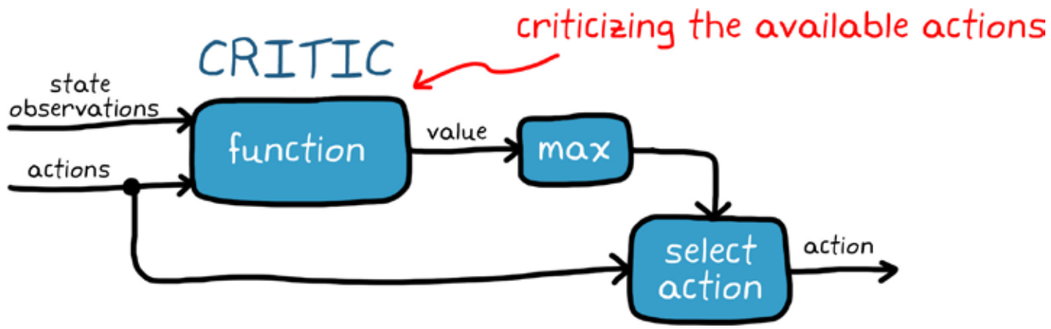
価値関数ベースのエージェントでは、関数は状態とその状態から可能な行動を入力とし、その行動の価値を出力します。



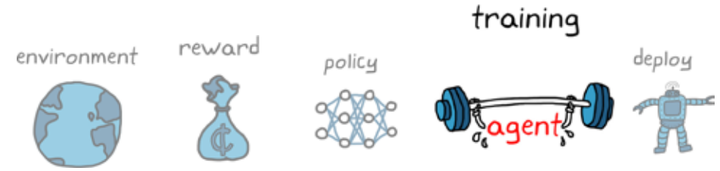


この関数は価値を出力しますが、方策は行動を出力する必要があるため、この関数だけでは方策を表現するのに十分ではありません。そのため、方策はこの関数を使い、ある状態で選択可能なすべての**行動**の**価値**を確認し、最も高い価値を持つ行動を選択します。

この関数は選択可能な行動を見てエージェントの選択を批評しているため、critic と考えることができます。



価値関数とグリッドワールド

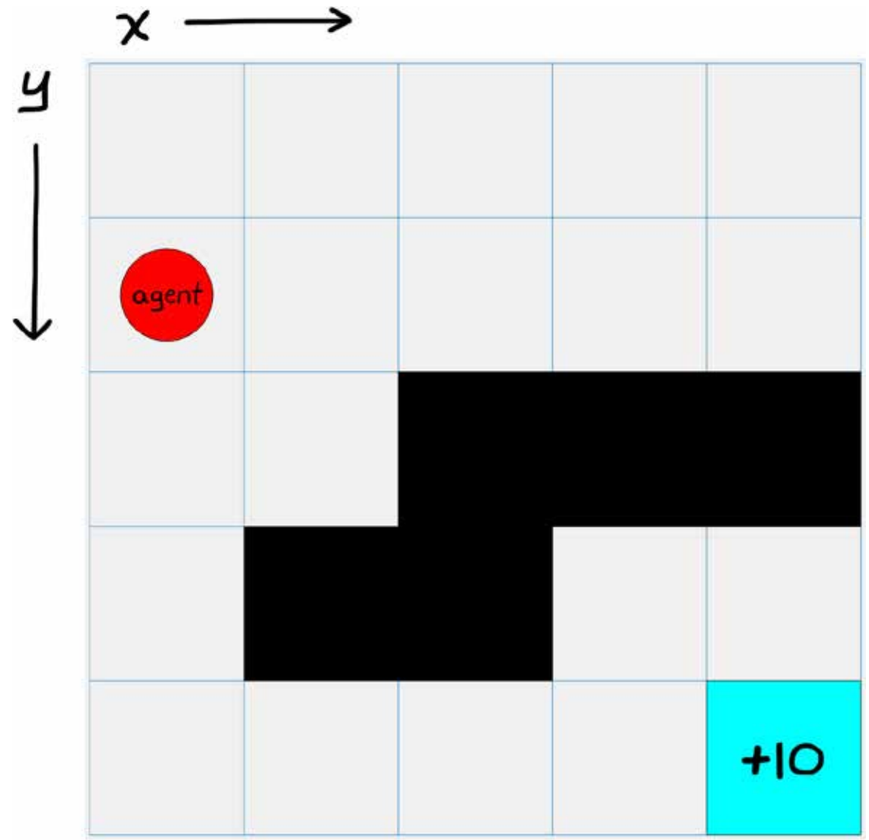


価値関数が実際にどう機能するかを確認するため、グリッドワールドを用いた例を考えてみます。

この環境では、2 つの離散状態変数である X グリッド位置 と Y グリッド位置があります。エージェントは 1 回に 1 マスのみ上、下、左、または右に進むことができ、エージェントがとるそれぞれの行動は -1 の報酬になります。

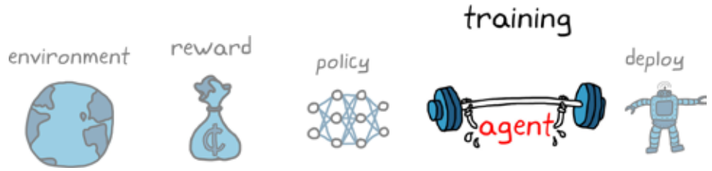
もしエージェントがマス目の外に出ようしたり、黒い障害物にぶつかって行こうとする場合、エージェントは新しい状態へは動きませんが、-1 の報酬は受け取ります。この方法でエージェントは、本質的に壁にぶつかって行ったことに対しペナルティが与えられ、その労力に見合うだけの物理的な前進は全くありません。

+10 の報酬を生み出す 1 つの状態があります。報酬を最大にするには、エージェントは可能な限り最も少ない移動で +10 に着く方策を学習する必要があるという考え方になります。



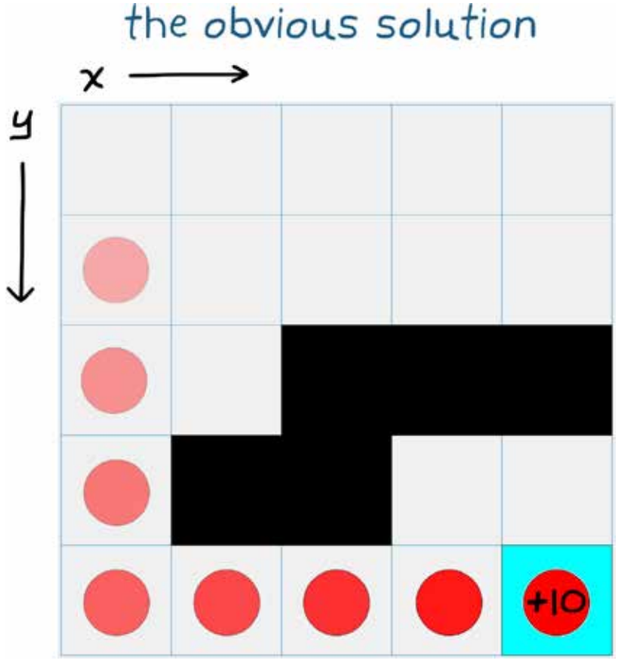
» *MATLAB でグリッドワールドを解く方法について見る*

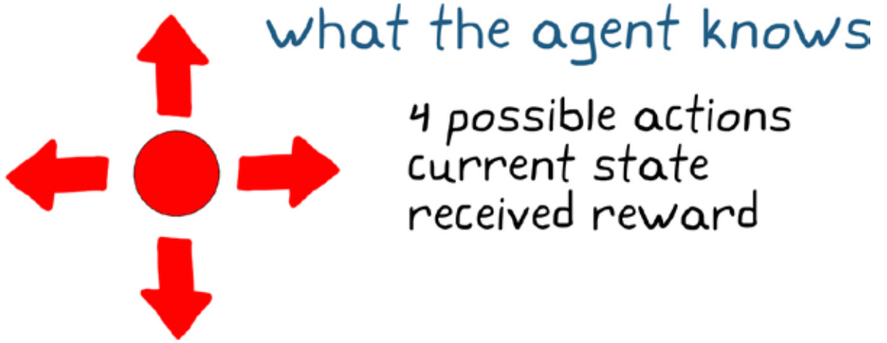
価値関数とグリッドワールド *続き*



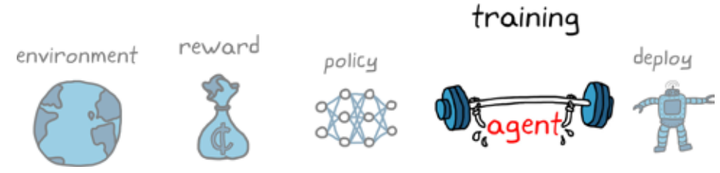
報酬を獲得するルートを決めることは簡単に思えるかもしれません。

しかし、モデルの無い強化学習では、エージェントは環境について何も知らないという点に注意する必要があります。エージェントは +10 に到着しようとしていることを知りません。4 つのうちの 1 つの行動を取ることができること、そして行動の後で環境から位置と報酬を受け取ることのみを知っています。



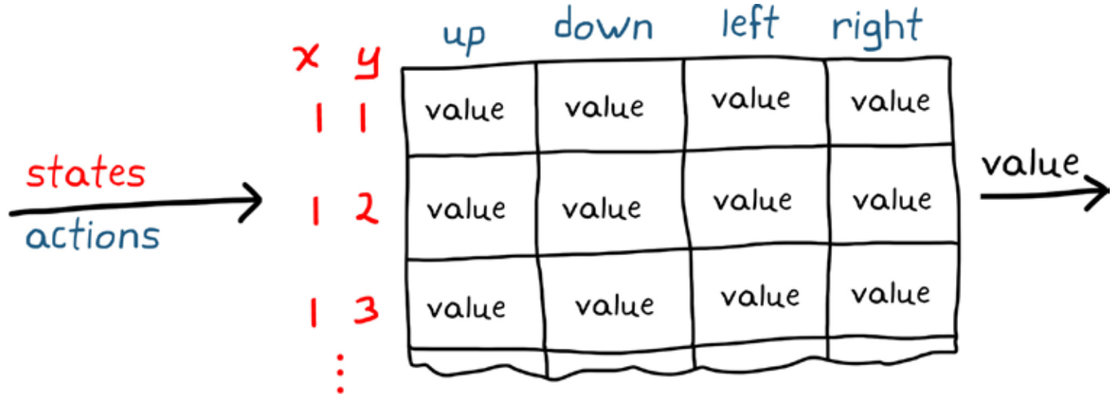


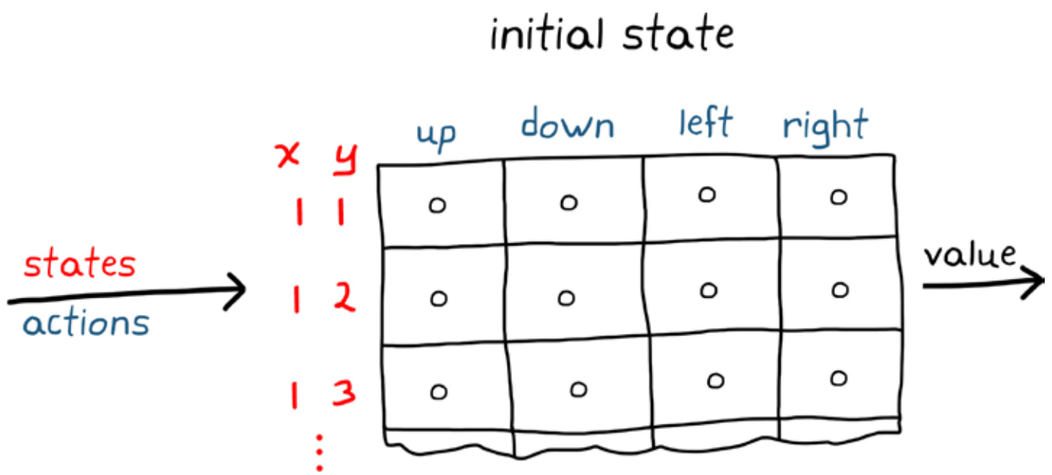
Q テーブルでグリッドワールドを解く



エージェントが環境の知識を得る方法は、行動をとり、受け取った報酬からその状態/行動の価値を学習することです。グリッドワールドでは状態と行動の数が有限であるため、Q テーブルを用いてこれらに価値をマッピングできます。

それでは、エージェントはどのようにしてこれらの価値を学習するのでしょうか。Q ラーニングというプロセスを通して学習します。

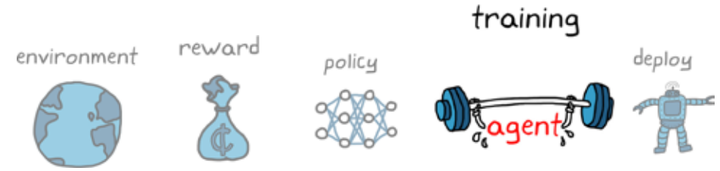




Q ラーニングでは、最初にテーブルを 0 で初期化できるので、すべての行動はエージェントにとって同じにように見えます。エージェントがランダムな行動をとった後、新しい状態になり、環境から報酬を集めます。

エージェントはその報酬を新しい情報として使い、有名なベルマン方程式を使用して前の状態と行動の価値を更新します。

ベルマン方程式



$$\text{new } Q(s, a) = Q(s, a) + \alpha\left[R(s, a) + \gamma \cdot \max Q'(s', a') - Q(s, a)\right]$$

ベルマン方程式により、エージェントは、問題の全体を複数の単純な手順に分解して、一定時間間隔でQ テーブルを解いていきます。1 つの手順で状態/行動のペアの真値を解くかわりに、エージェントは動的モデルにより、状態/行動のペアが更新される度に価値を更新します。ベルマン方程式は、DQN のような学習アルゴリズムだけでなく、Q ラーニングにおいても重要です。方程式の各条件について、詳しく見ていきましょう。

エージェントは状態 (**s**) から行動 (**a**) をとった後、報酬を受け取ります。

$$\text{new } Q(s, a) = Q(s, a) + \alpha\left[R(s, a) + \gamma \cdot \max Q'(s', a') - Q(s, a)\right]$$

reward for taking action, a, from state, s

価値は、行動からの即時の収益というだけでなく、将来にわたっての最大期待値です。そのため、状態/行動ペアの価値はエージェントが今受け取った報酬に、将来エージェントが集めるであろう報酬を加えたものです。

$$\text{new } Q(s, a) = Q(s, a) + \alpha\left[R(s, a) + \gamma \cdot \max Q'(s', a') - Q(s, a)\right]$$

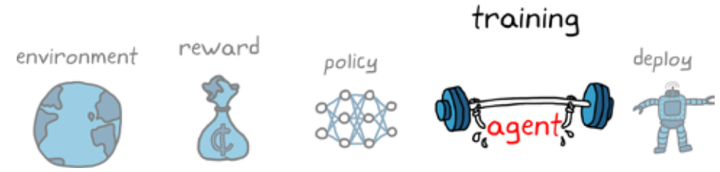
maximum expected value from state, s'

エージェントが遠い将来の報酬を信頼し過ぎないよう、将来の報酬をガンマで割り引くことができます。ガンマは 0 (将来の報酬を考慮せずに評価) と 1 (無限に遠い将来の報酬まで考慮して評価) の間の数値です。

$$\text{new } Q(s, a) = Q(s, a) + \alpha\left[R(s, a) + \gamma \cdot \max Q'(s', a') - Q(s, a)\right]$$

discount future rewards

ベルマン方程式 *続き*



これで合計は状態と行動のペアの新しい価値 (**s**、**a**) になり、前の推定と比較して誤りを見つけます。

$$\text{new } Q(s,a) = Q(s,a) + \alpha\left[\underbrace{R(s,a) + \gamma \cdot \max Q'(s',a')}_{\text{new best estimate of value}} - \overbrace{Q(s,a)}^{\text{previous estimate of value}}\right]$$

誤りを学習率で乗じます。学習率により、古い価値の見積もりを新しい値で置き換えるか(アルファ = 1)、古い価値を新しい値に少し動かすか(アルファ < 1)制御できるようになります。

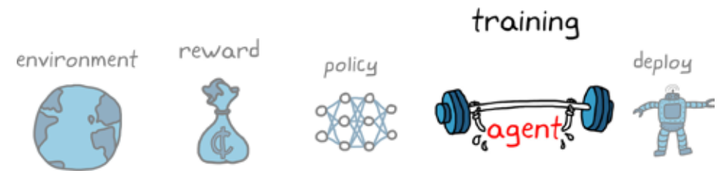
$$\text{new } Q(s,a) = Q(s,a) + \underbrace{\alpha}_{\text{error is multiplied by learning rate}}\left[R(s,a) + \gamma \cdot \max Q'(s',a') - Q(s,a)\right]$$

最後に、結果となるデルタ値が古い推定に加えられ、Q テーブルが更新されます。

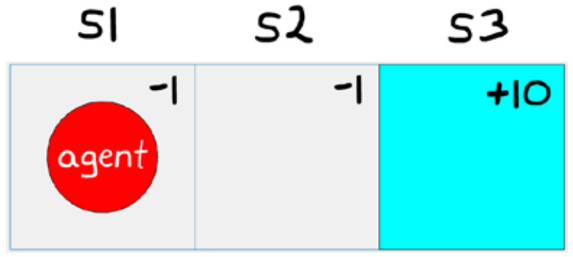
$$\text{new } Q(s,a) = \underbrace{Q(s,a)}_{\text{delta value is added to old estimate}} + \alpha\left[R(s,a) + \gamma \cdot \max Q'(s',a') - Q(s,a)\right]$$

ベルマン方程式は、強化学習と従来の制御理論とのもう 1 つのつながりです。最適制御理論をご存知なら、この方程式がハミルトン-ヤコビ-ベルマン方程式の離散したバージョンだと気が付くかもしれません。これは状態空間全体で解く場合には、最適性の必要十分条件となります。

ベルマン方程式 *続き*



簡単なグリッドワールドの例で最初のいくつかのステップの動作を確認し、ベルマン方程式が動くところを見てみましょう。この例では、アルファは 1 に、ガンマは 0.9 に設定されています。両方の行動が同じ価値となる場合、エージェントはランダムな行動をとります。そうでなければ、エージェントは最も高い価値を持つ行動を選択します。



エピソード	手順	状態	現在の Q(s, a)	行動	R(s, a)	新しい Q(s, a)
1	1	S1		右 (ランダム)	-1	

	S1	S2	S3
左	0	0	0
右	0	0	0

	S1	S2	S3
左	0	0	0
右	-1	0	0

Bellman equation: $0 + 1 \cdot [-1 + 0.9 \cdot 0 - 0] = -1$

エピソード	手順	状態	現在の Q(s, a)	行動	R(s, a)	新しい Q(s, a)
1	2	S2		右 (ランダム)	+10	

	S1	S2	S3
左	0	0	0
右	0	0	0

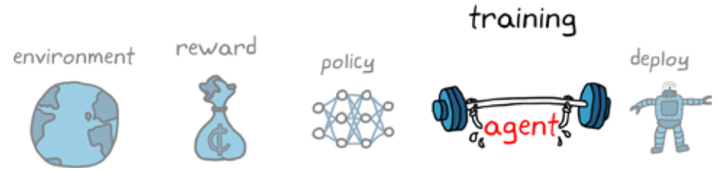
	S1	S2	S3
左	0	0	0
右	-1	10	0

Bellman equation: $0 + 1 \cdot [10 + 0.9 \cdot 0 - 0] = +10$

エピソードの終わり

エージェントが **S3** の終了状態に到達したら、エピソードが終わり、エージェントは最初の状態の **S1** に再初期化されます。
Q テーブルの価値は保持され、学習は次のページで書かれている、次のエピソードへと続きます。

ベルマン方程式 *続き*



エピソード	手順	状態	現在の Q(s, a)	行動	R(s, a)	新しい Q(s, a)
2	1	S1	S1: 左 0, 右 -1; S2: 左 0, 右 10; S3: 左 0, 右 0	左 (貪欲)	-1	S1: 左 -1, 右 -1; S2: 左 0, 右 10; S3: 左 0, 右 0
2	2	S1	S1: 左 -1, 右 -1; S2: 左 0, 右 10; S3: 左 0, 右 0	右 (ランダム)	-1	S1: 左 -1, 右 8; S2: 左 0, 右 10; S3: 左 0, 右 0

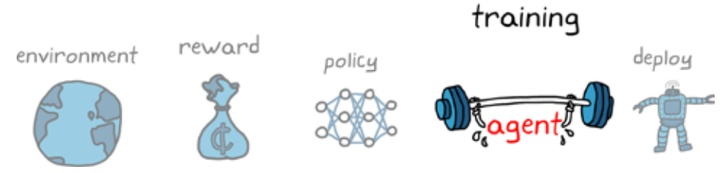
Bellman equation: $0 + 1 \cdot [-1 + 0.9 \cdot 0 - 0] = -1$

Bellman equation: $-1 + 1 \cdot [-1 + 0.9 \cdot 10 - (-1)] = +8$

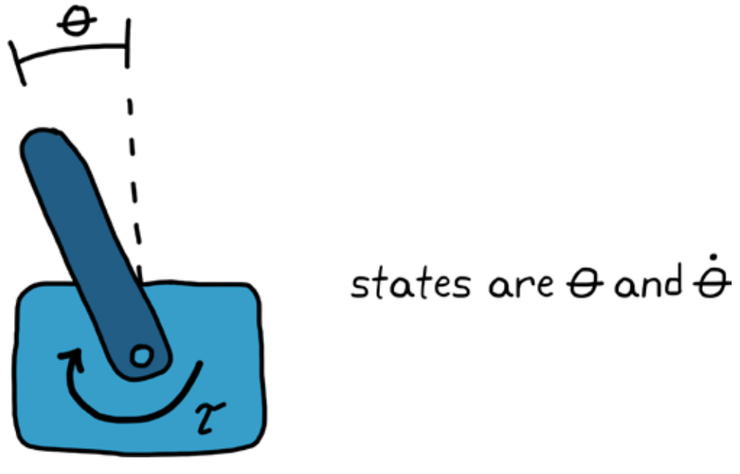
エピソードの終わり

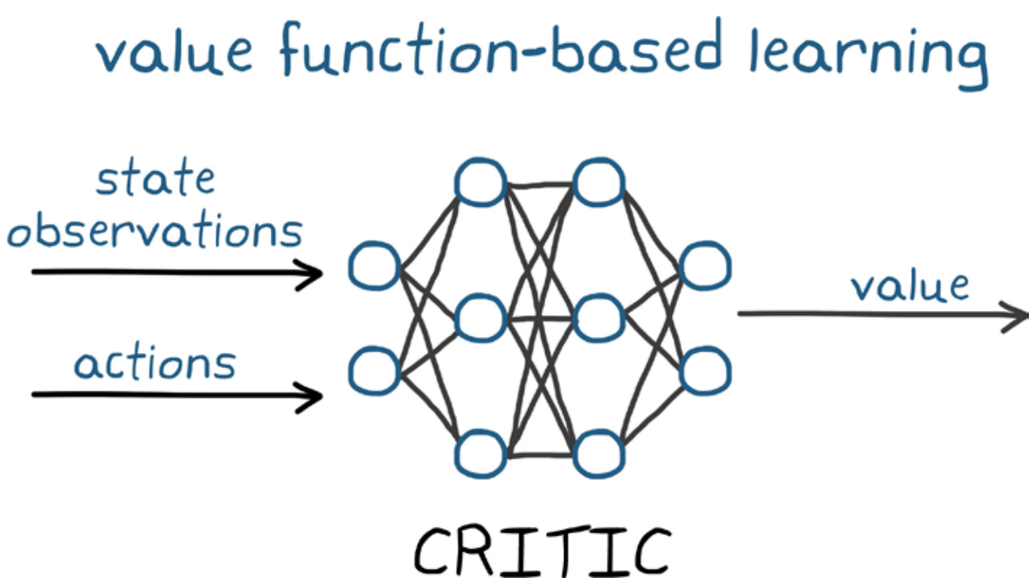
わずか 4 つの行動で、エージェントは最適な方策を生み出す Q テーブルに落ち着きました。**S1** の状態では、価値 8 は -1 より高いためエージェントは右に行き、**S2** の状態では、価値 10 は 0 より高いためまた右に行きます。この結果で興味深いことは、Q テーブルは各状態/行動ペアの真の価値になっていないというところです。学習を続ければ、価値は実際の価値の方向へと動き続けます。しかし、最適な方策を生み出すのに真の価値を発見する必要はありません。最も高い数値になる最適な行動の価値が分かれば十分です。

ニューラル ネットワークとしてのCritic



この考え方を倒立振子に広げてみます。グリッドワールドのように、角度と角速度の 2 つの状態がありますが、こちらの状態は連続状態です。

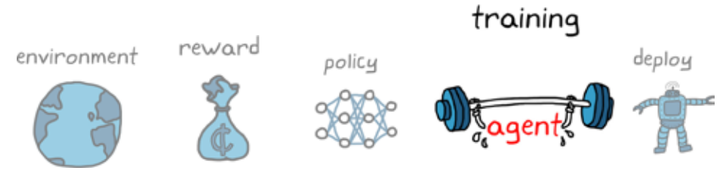




価値関数 (critic) がニューラル ネットワークで表現されます。考え方はテーブルと同じです。状態の観測と行動を入力すると、ニューラル ネットワークは状態/行動のペアの価値と、最も高い価値の行動を選択する方策を返します。

時間と共に、ネットワークは連続した状態空間のどの場所のどの行動でも正しい価値を返す関数にゆっくりと収束していきます。

価値ベース 方策の欠点



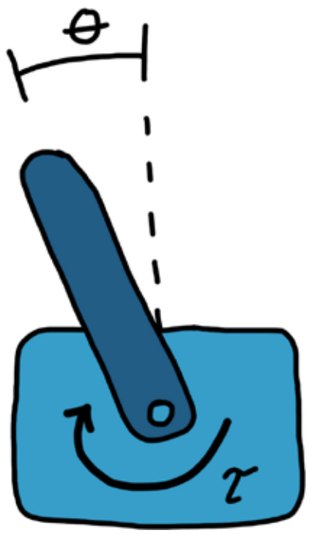
ニューラル ネットワークを使用して、連続した状態空間の価値関数を定義できます。倒立振子が離散した行動空間を持つ場合、離散した行動を 1 つずつcriticのネットワークに入力することができます。

価値関数ベースの方策は、連続した行動空間ではうまく機能しません。これは無限にある行動の価値を 1 つずつ計算して、最大の価値を見つける方法が無いためです。大規模な (しかし無限ではない) 行動空間でも、これにより演算が膨大になります。これは残念です。なぜなら、連続したトルクの範囲を倒立振子の問題に適用する場合など、制御の問題ではたびたび行動空間が連続となるためです。

それではどうすべきでしょうか。

方策関数ベースのアルゴリズムのセクションで説明されている、Vanilla Policy Gradient 法を実装できます。これらのアルゴリズムは連続した行動空間を処理できますが、報酬の差が大きく勾配にノイズが多いときには、収束が難しくなります。また、actor-criticと呼ばれるアルゴリズムのクラスに 2 つの学習手法を統合できます。

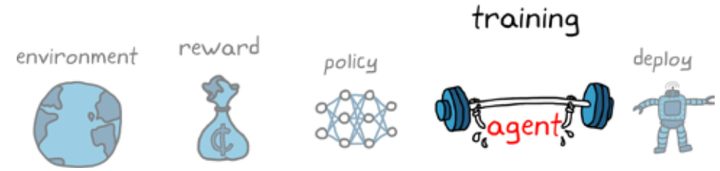
» *MATLAB でactor-criticのエージェントを学習させて、倒立振子のバランスを取る方法について見る*



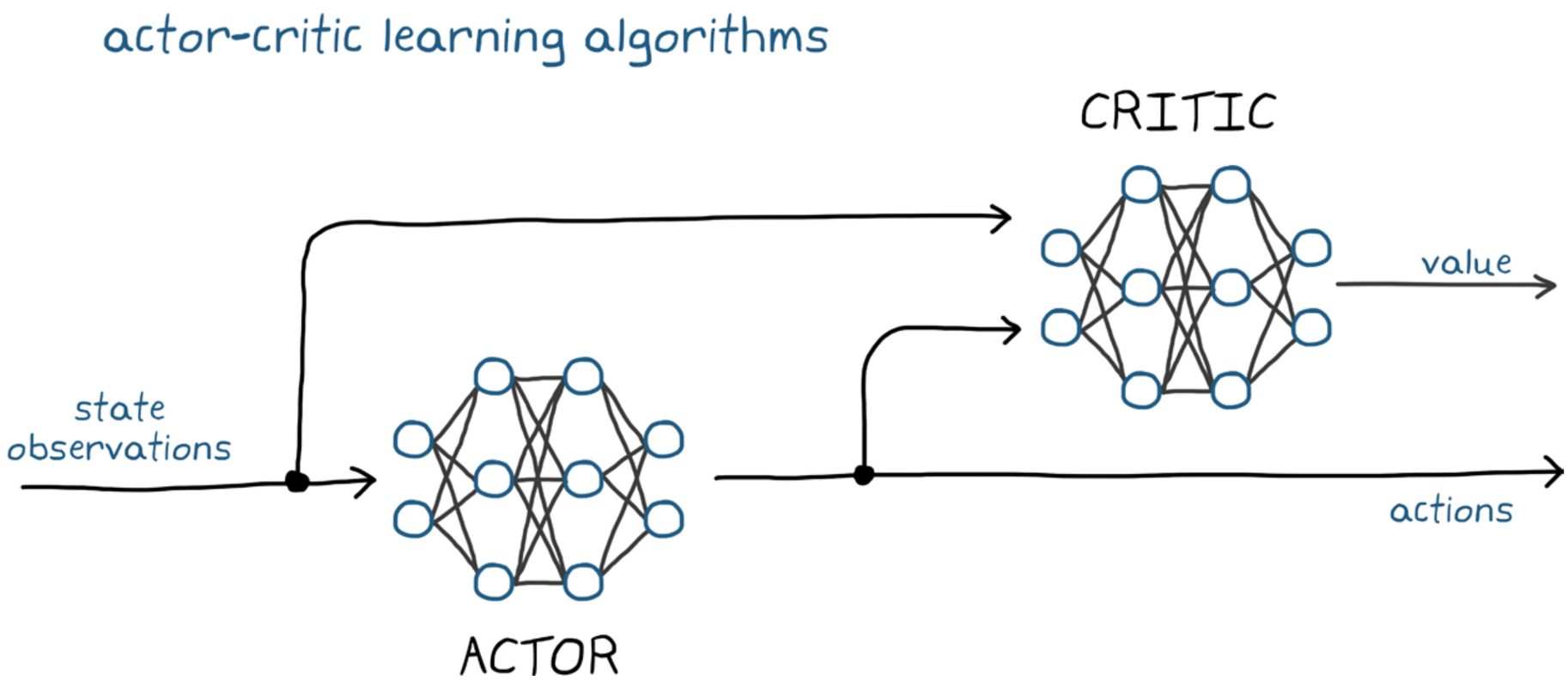
continuous states: θ and $\dot{\theta}$

discrete action space: τ = [-2, -1, 0, 1, 2] Nm

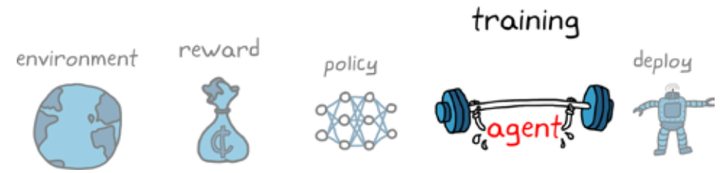
Actor-Critic 手法



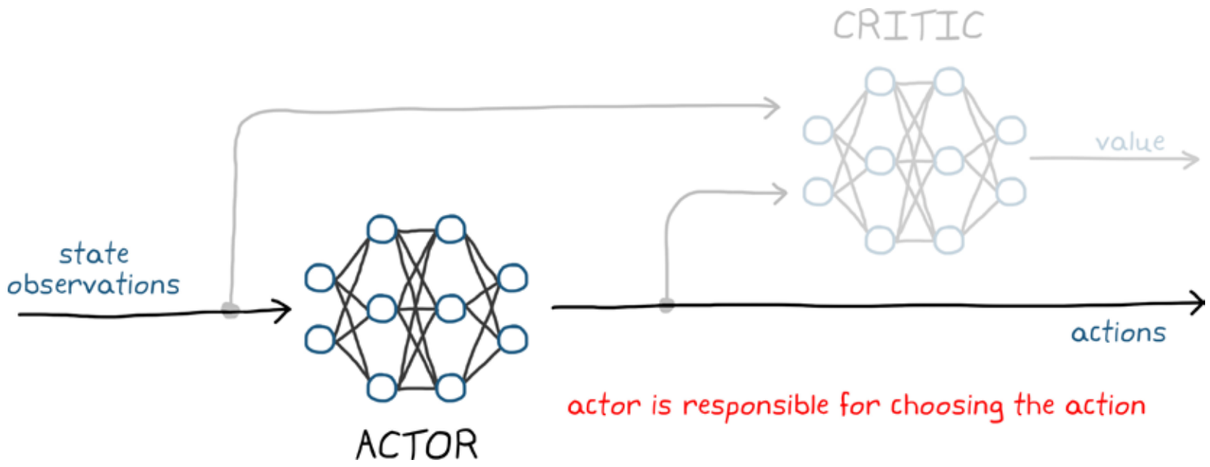
actor は、方策関数の手法で見たように、現在の状態で最善だと考える行動をとるネットワークのことです。*critic* は、価値関数の手法で見たように、actorがとる行動の価値を推定しようとする 2 つ目のネットワークのことです。このアプローチでは、criticはactorがとる行動の 1 つを見るだけでいいので、すべてを評価して最善の行動を見つける必要が無く、連続した行動空間でもうまく機能します。

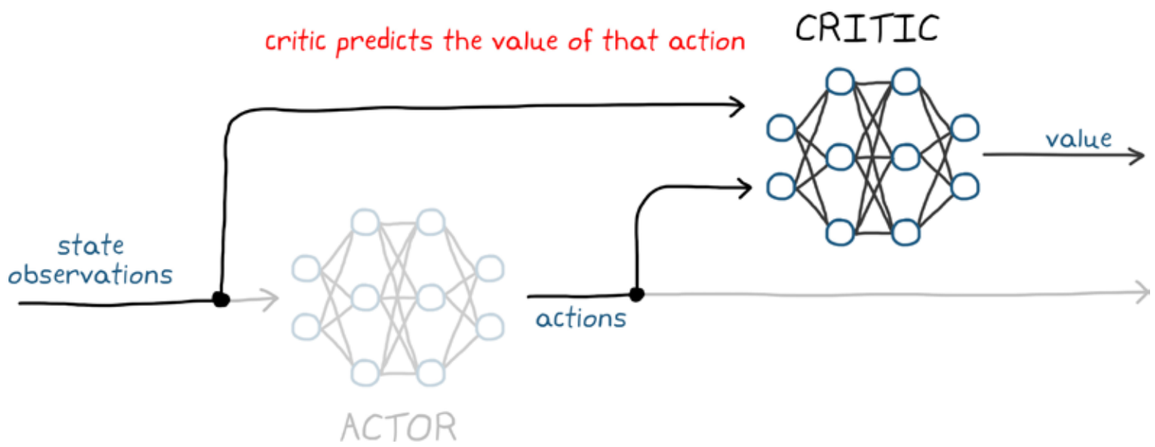


Actor-Critic の学習サイクル



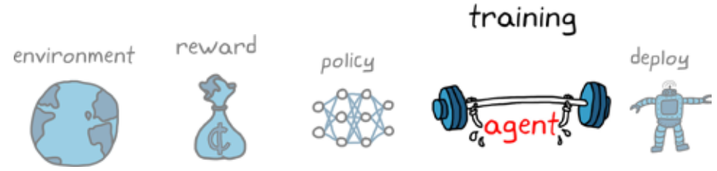
actorは、方策関数アルゴリズムと同じように行動を選択し、それが環境に適用されます。



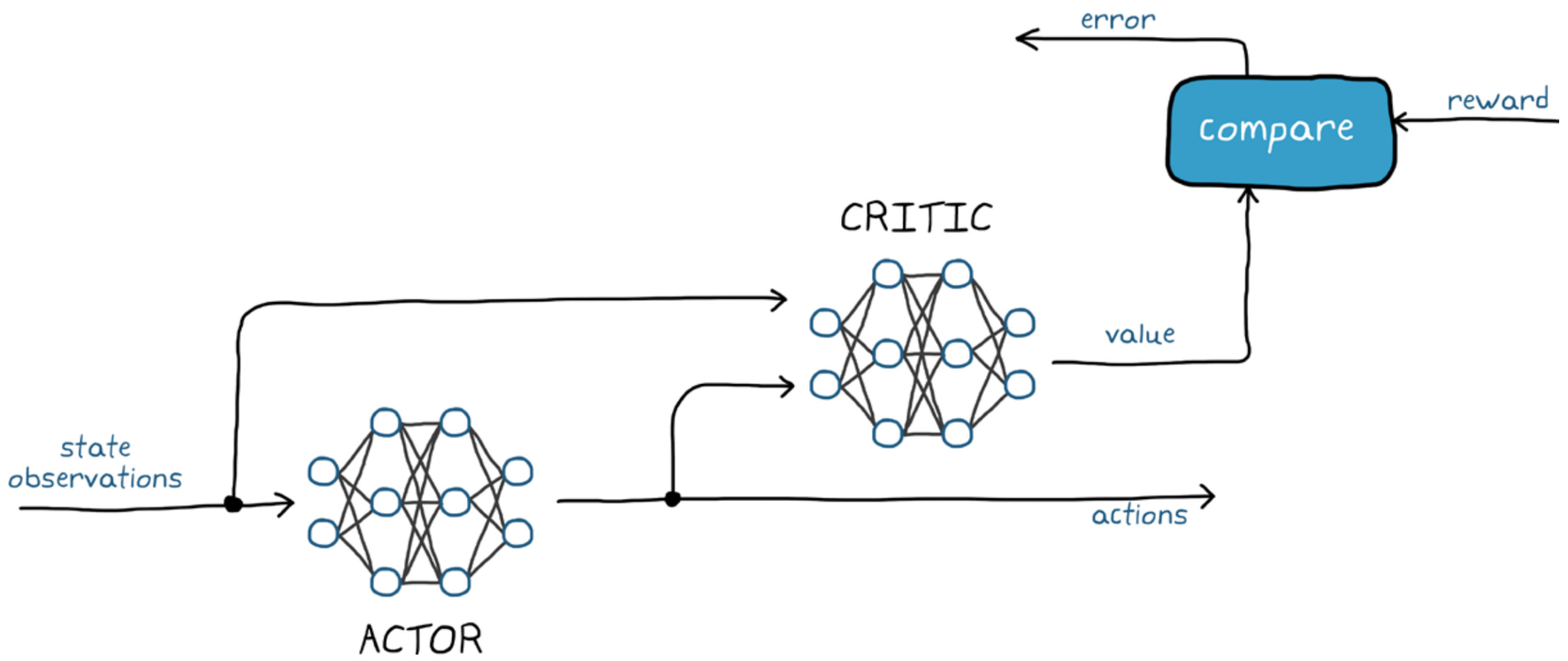


criticは、現在の状態/行動のペアでの行動の価値を予測します。

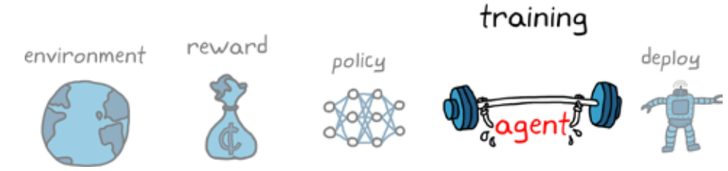
Actor-Critic の学習サイクル *続き*



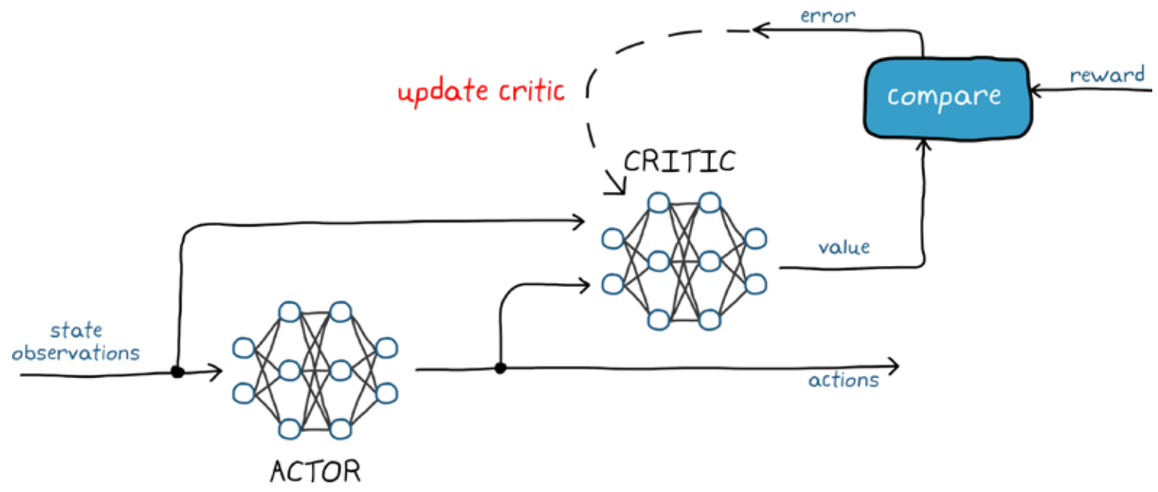
次にcriticは環境からの報酬を用いて、価値予測の精度を判定します。誤差は、critic ネットワークでの前の状態の新しい推定価値と、前の状態の古い価値の差です。新しい推定価値は、受け取った報酬と現在の状態の割り引きされた価値に基づきます。誤差により、criticは予想より良いか悪いかについて知ることができます。

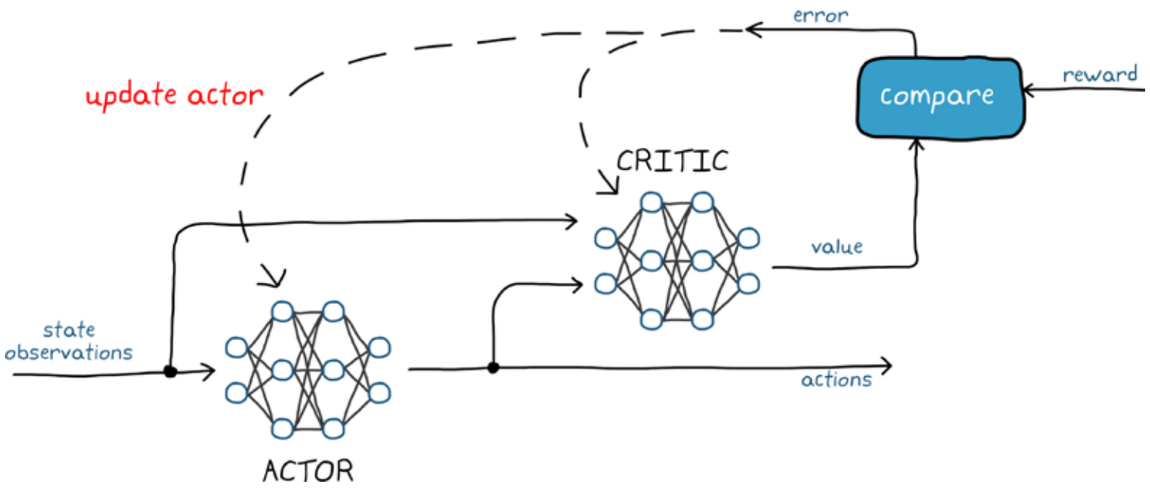


Actor-Critic の学習サイクル *続き*



criticは誤差を使用して、価値関数と同じように自らを更新し、この状態における次の予測を改善します。

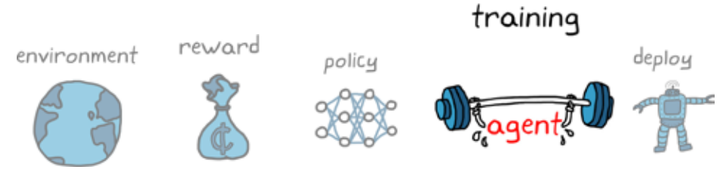




actorもcriticからの応答に応じて自らを更新し、将来にまた同じ行動をとる確率を調整します。

この方法で方策は、報酬を直接用いるかわりに、critic による疑似的な報酬を使って改善できるようになります。

2 つの相補的ネットワーク

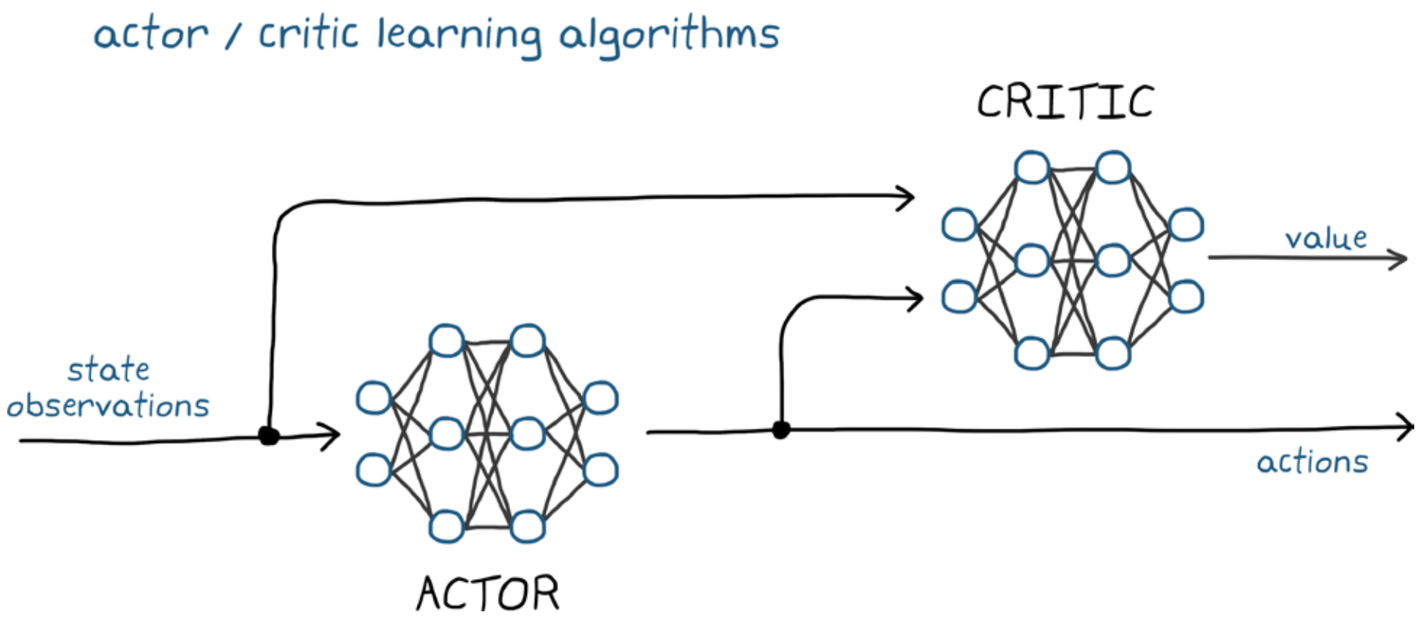


多くの異なる種類の学習アルゴリズムがactor-critic方策を使用しています。この ebook ではこれらの概念を一般化して、特定のアルゴリズムに依存しないものにしています。

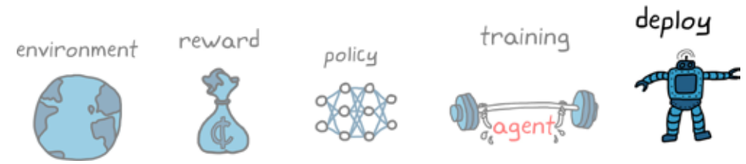
actorとcriticは最適な挙動を学習するニューラル ネットワークです。actorは、criticからのフィードバックを用いて正しい行動を学習し、どの行動が良いか、悪いかを知ります。そしてcriticは受け取った報酬から価値関数を学習し、actorが行った行動を適切に批評できるようにします。

actor-critic手法では、エージェントは、方策関数と価値関数アルゴリズムの双方の利点を活用できます。方策 criticは連続した状態空間および行動空間の両方を処理でき、返される報酬に差が大きい場合も高速に学習できます。

エージェントの作成時に 2 つのニューラル ネットワークをセットアップしなければならない理由がこれで明確になったと思います。それぞれが非常に明確な役割を担っています。

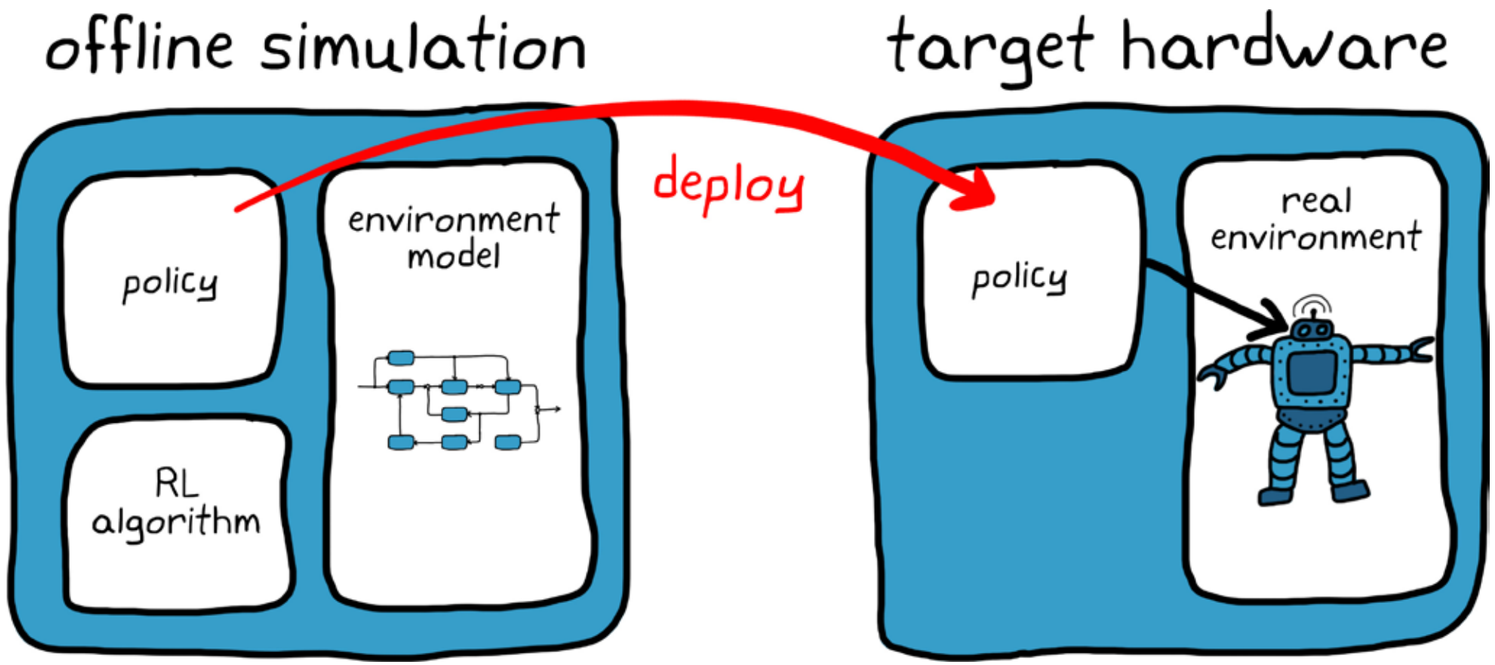


方策の展開

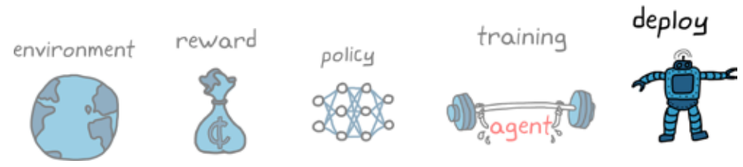


強化学習のワークフローにおける最後の手順が方策の展開です。

現実の環境で物理的なエージェントにより学習が行われていれば、既にエージェントには学習した方策があり、これを活用できます。このebook では、シミュレーションされた環境と対話することで、エージェントがオフラインで学習を行っていることを前提とします。方策が十分に最適化されれば、従来の開発された制御則の展開と同じように、学習プロセスを終えて、静的な方策を任意の数のターゲットに展開できます。



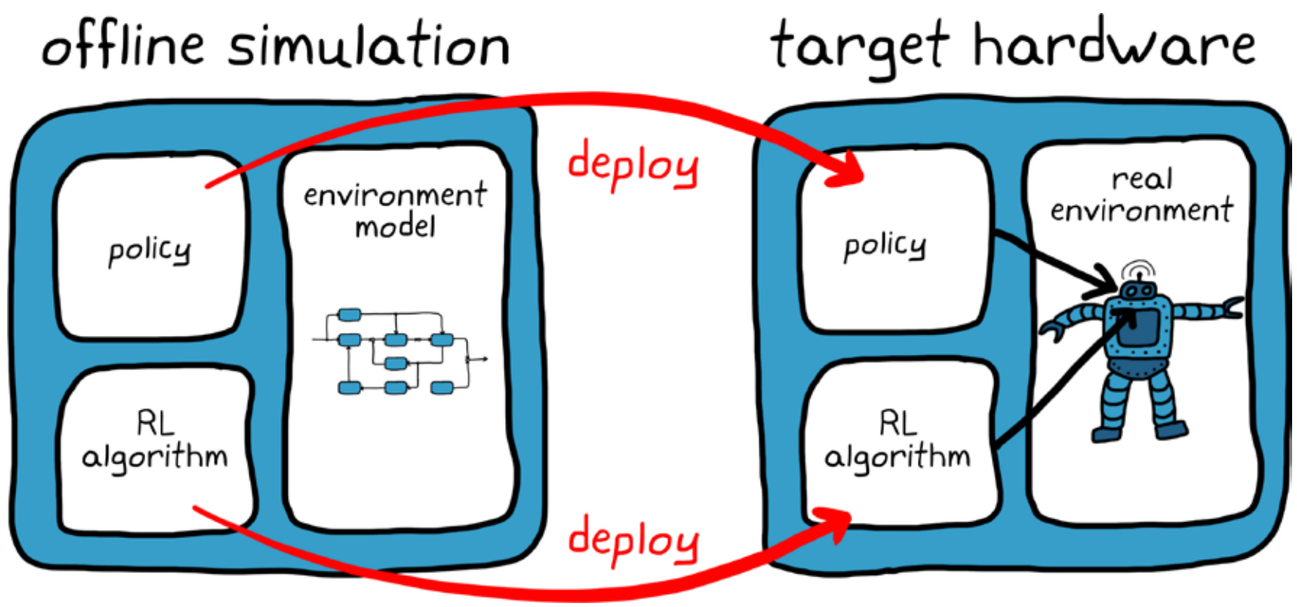
学習アルゴリズムの展開



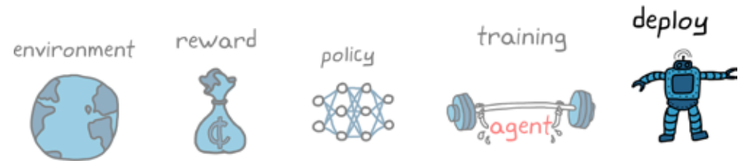
学習の大部分がシミュレーション環境でオフラインで行われた場合でも、実際の物理ハードウェアで展開後に学習を続ける必要があることがあります。

これは正確にモデル化することが難しい環境がいくつかあるためです。そのため、モデルで最適な方策も、実際の環境では最適ではない可能性があります。また、他に考えられる理由としては、環境が時間の経過に伴ってゆっくりと変化するため、エージェントはこれらの変化に合わせて調整できるよう、時々学習を継続する必要があるのかもしれません。

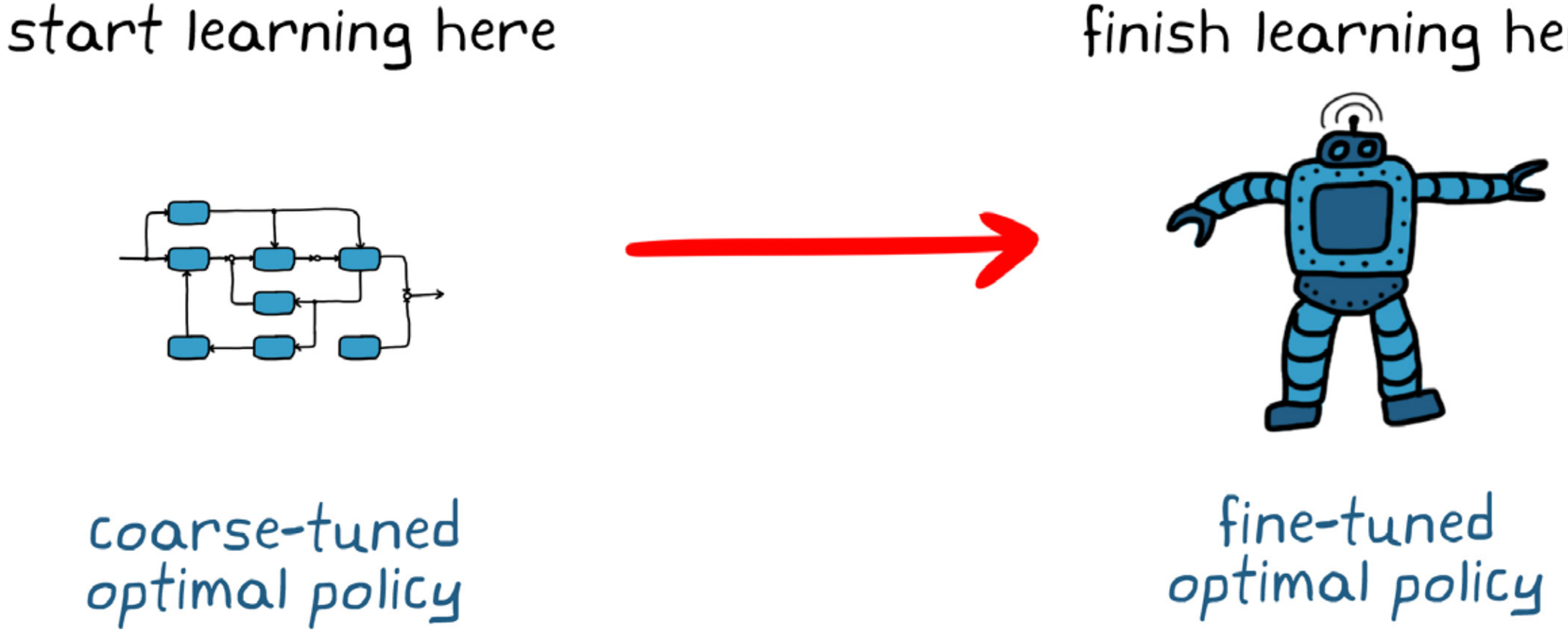
これらの理由により、静的な方策と学習アルゴリズムの両方をターゲットに展開します。このセットアップでは、静的な方策を実行する(学習を無効にする) か、方策の更新を続ける (学習を有効にする) かの選択肢があります。



補足的関係



シミュレーション環境での学習と現実環境での学習の間には補足的関係があります。シミュレーションでは、安全かつ比較的早く、十分に最適な方策を学習できます。その方策はハードウェアを安全に保ち、完璧ではなくても望まれる挙動に近いものになります。次に物理ハードウェアとオンラインでの学習を用いて方策を微調整し、環境にぴったりあったものを作り上げます。

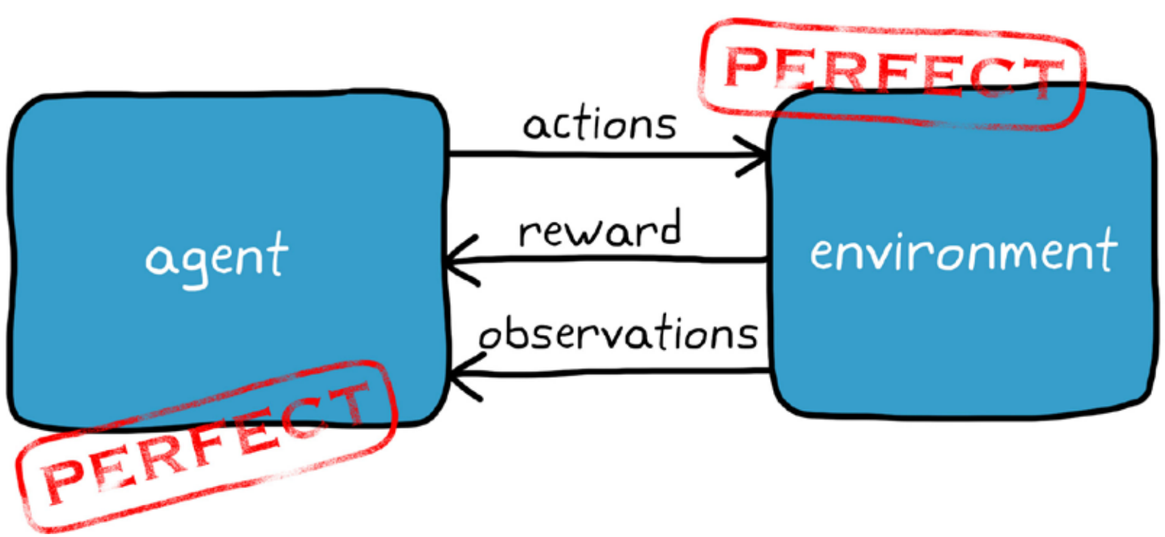


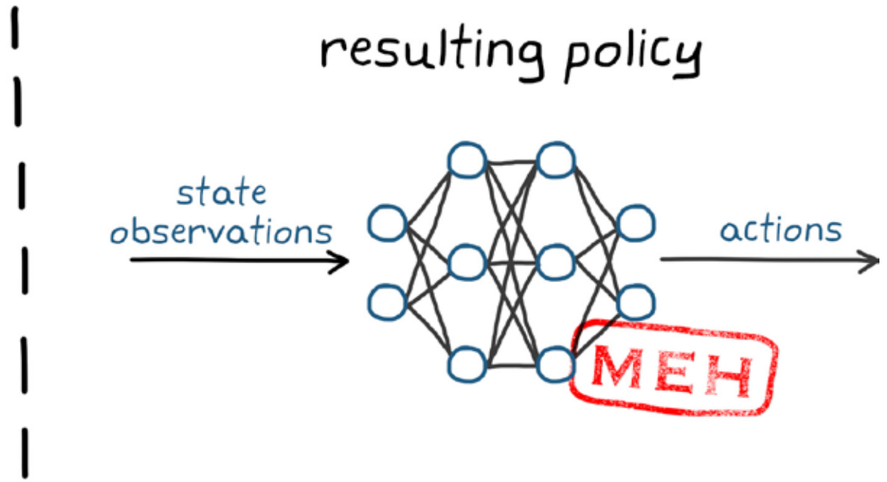
強化学習の難点

この時点で、環境をセットアップして、強化学習のエージェントをそこに設置し、どこかに行ってコーヒーを飲んでいる間に、コンピューターに問題を解決させることができるだろうと考えるかもしれません。残念ながら、完璧なエージェントと完璧な環境をセットアップし、学習アルゴリズムが解に収束したとしても、この手法にはまだ課題があります。

これらの課題は次の 2 つの疑問に要約されます。

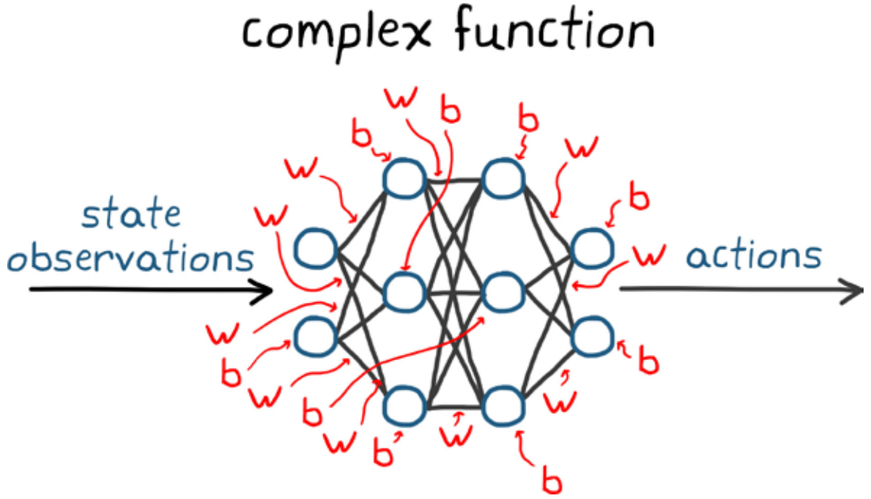
- その解が機能するとどうすれば分かるのか。
- それが完璧でない場合、手動で調整する方法はあるか。

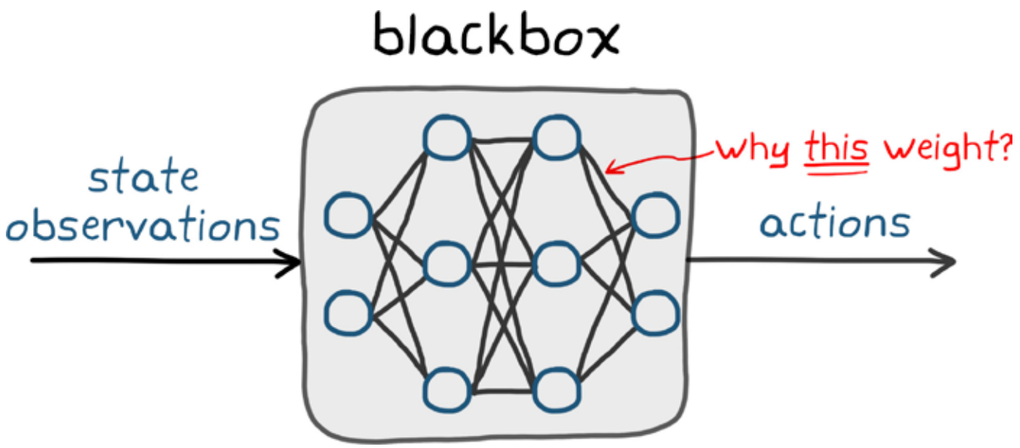




説明できないニューラル ネットワーク

数学的には、方策はおよそ数百から数千の重みとバイアス、および非線形活性化関数を持つニューラル ネットワークで構成されます。ネットワークでのこれらの構造と値の組み合わせが、高レベルの観測と低レベルの行動をマッピングする複雑な関数を作り出します。





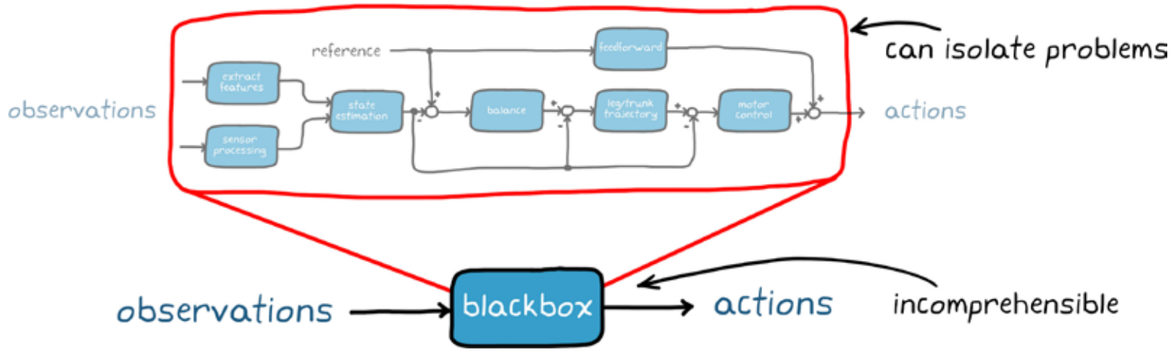
この関数は設計者にとってブラックボックスです。この関数が動作する方法や、ネットワークが識別した隠れた特徴について直感的に分かっていたとしても、ある重みやバイアスの値の背後にある理由については分かりません。そのため、方策が仕様を満たさない場合や、動作環境が変わった場合も、その問題に対処するためどのように方策を調整すればいいかは分かりません。

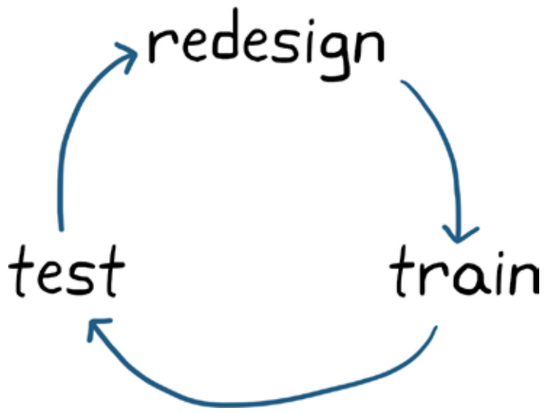
*説明可能な人工知能*の概念を押し広げる活発な研究がなされています。これは人間がネットワークを構築できるのだから、それを簡単に理解して監査することもできるという考え方です。この時点では、強化学習により生成される方策の大部分はブラックボックスとして分類されているため、解決が必要な課題となっています。

問題を正確に特定

制御問題を解くことを簡単にした方法そのものに課題があります。難解なロジックを単一のブラックボックス関数に要約したことで、最終的な解が理解不能になりました。これを従来の方法で設計された制御システムと比較すると、従来型は、一般的には階層化されたループとカスケードコントローラーがあり、それぞれがシステムの特定の動的な品質を制御するよう設計されています。付属品の長さやモーター定数などの物理的なプロパティからゲインが算出される方法、また物理的なシステムが変更された場合にこれらのゲインを変更する簡単さについて考えてみてください。

また、システムの挙動が期待どおりでない場合も、従来の設計では特定のコントローラーやループでの問題を正確に特定して、分析に焦点を絞ることができます。コントローラーを分離し、それをテストして変更し、特定の条件で実行されていること確認した後に、全体のシステムにそれを戻すことができます。



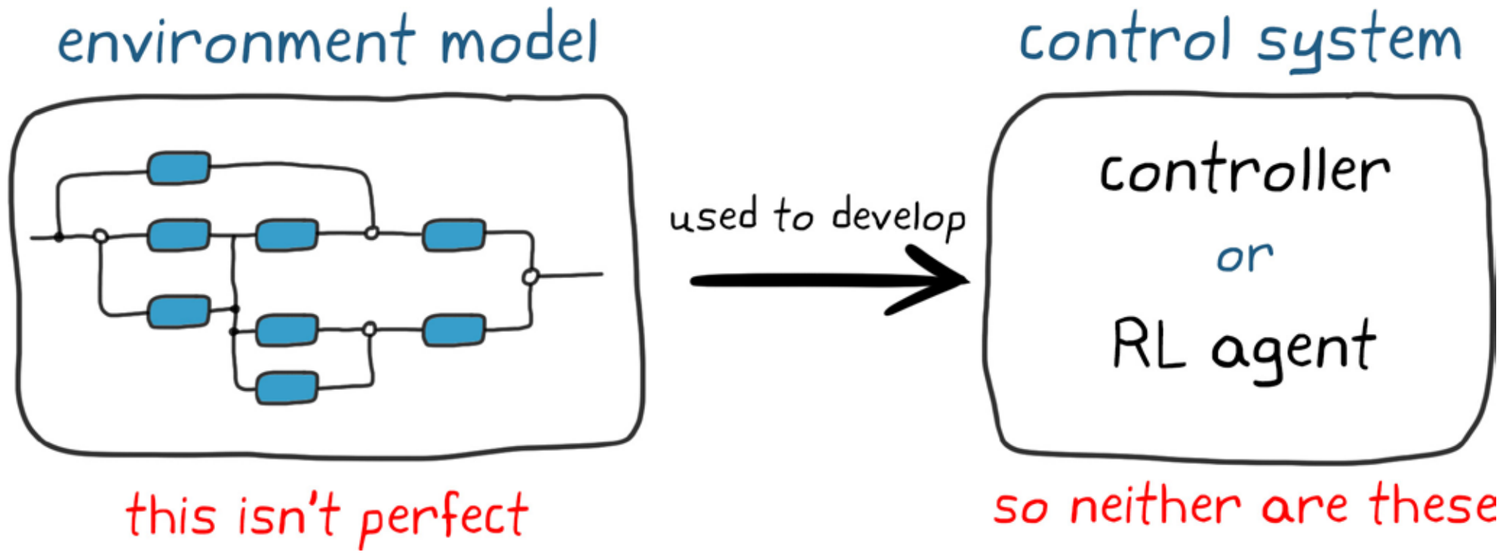


解がニューロンと重みおよびバイアスのモノシリックな集合である場合、問題の分離は困難です。そのため、方策がまったく正しくない結果になった場合、方策の欠陥のある部分を修正するかわりに、エージェントまたは環境モデルを再設計して再度学習させなければなりません。この再設計、学習、テストのサイクルには多くの時間がかかります。

より大きな問題

ここにはエージェントの学習にかかる時間よりも大きな問題が立ちはだかっていています。そしてこれは環境モデルの精度の点に行き着きます。

外乱やノイズの他に重要なシステム ダイナミクスをすべて考慮した、十分に現実的なモデルを開発するのは困難です。ある点で現実を完璧には反映しないため、そのモデルで開発した制御システムも完璧にはなりません。これがモデルですべてを検証するだけではなく、物理テストも行う必要がある理由です。



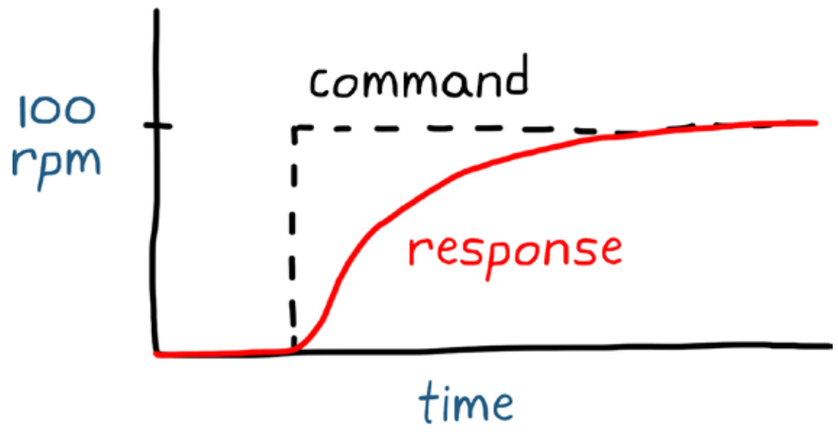
モデルを使用して従来の制御システムを設計する際には、関数を理解してコントローラーを調整できるため、この点はそれほど問題になりません。しかし、ニューラル ネットワークの方策では、このような贅沢はできません。絶対的に現実的なモデルは構築できないため、モデルで学習したエージェントは、どれも少しだけ間違っています。これを修正する唯一の選択肢はエージェントの学習を物理ハードウェアで終わらせることですが、これはその性質上、困難な場合があります。

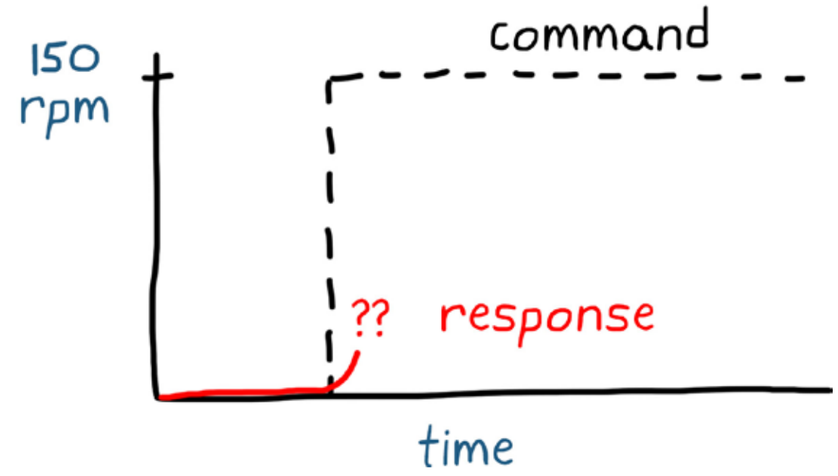
学習した方策の検証

方策が仕様を満たしているかの検証も、ニューラル ネットワークでは困難です。理由の 1 つとして、学習した方策では、他の状態での挙動からある状態でのシステムの挙動を予測するのが難しいということがあります。たとえば、0 から 100 RPM までのステップ入力に従うようにして、電気モーターの速度を制御するようエージェントを学習させた場合、テストなしでは、0 から 150 RPM までという類似のステップ入力に同じ方策が従うかどうかを確信できません。これはモーターが線形で挙動するにも関わらず真実です。

少しの変化が、全く異なる一連のニューロンを活性化させ、望まない結果を生み出すことがあります。テストするまでその結果を知ることができません。より多くの条件をテストすることでリスクは減りますが、すべての入力の組み合わせをテストできない限り、方策が 100% 正しいと保証することはできません。

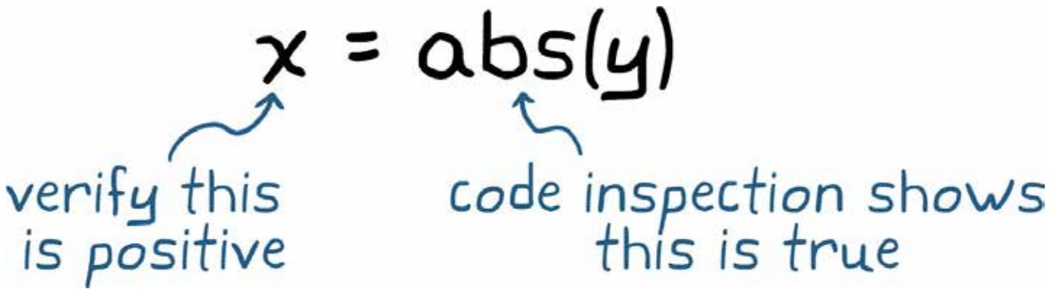
少しのテストを追加で行う必要があることは、それほど大きな問題でないように思えるかもしれません。しかしディープ ニューラル ネットワークの利点の 1 つが、非常に大規模な入力空間を持つカメラ画像など、高機能なセンサーからのデータを処理できることであるという点に注意が必要です。それぞれに 0 から 255 までの値を設定できる数千のピクセルを考えてみてください。このシナリオですべての組み合わせをテストするのは不可能です。

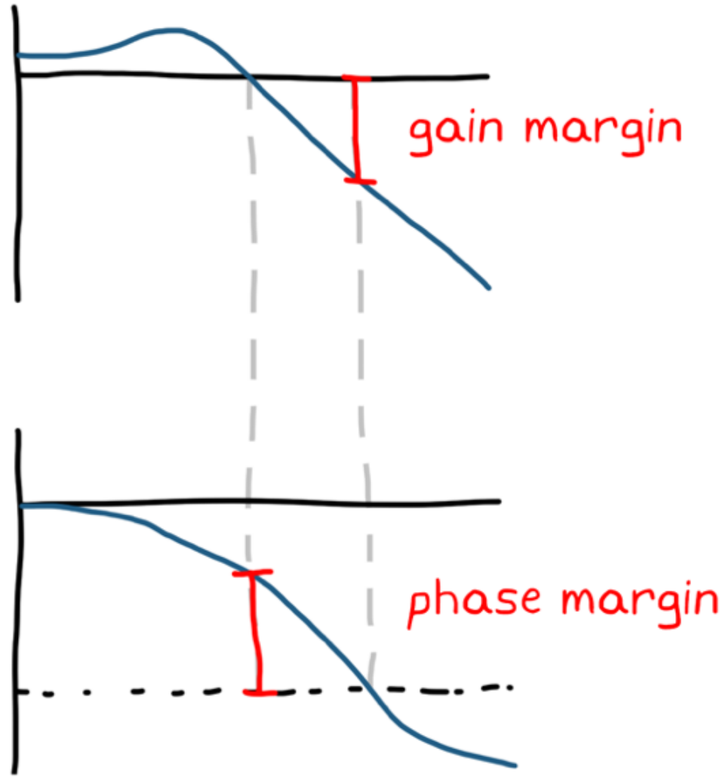




形式的検証手法

学習したニューラル ネットワークも、形式的検証を困難にします。これらの手法は、テストを使用するのではなく、形式的実証を提供することによって、ある条件が満たされることを保証します。たとえば、ソフトウェアで信号の絶対値の操作が実行されていれば、信号が常に正の値になってるかどうかをテストして確認する必要はありません。単にコードを検査して条件が常に満たされることを表示することにより、これを検証できます。他の種類の形式的検証には、ゲイン余裕や位相余裕などロバスト性や安定性の値の計算があります。



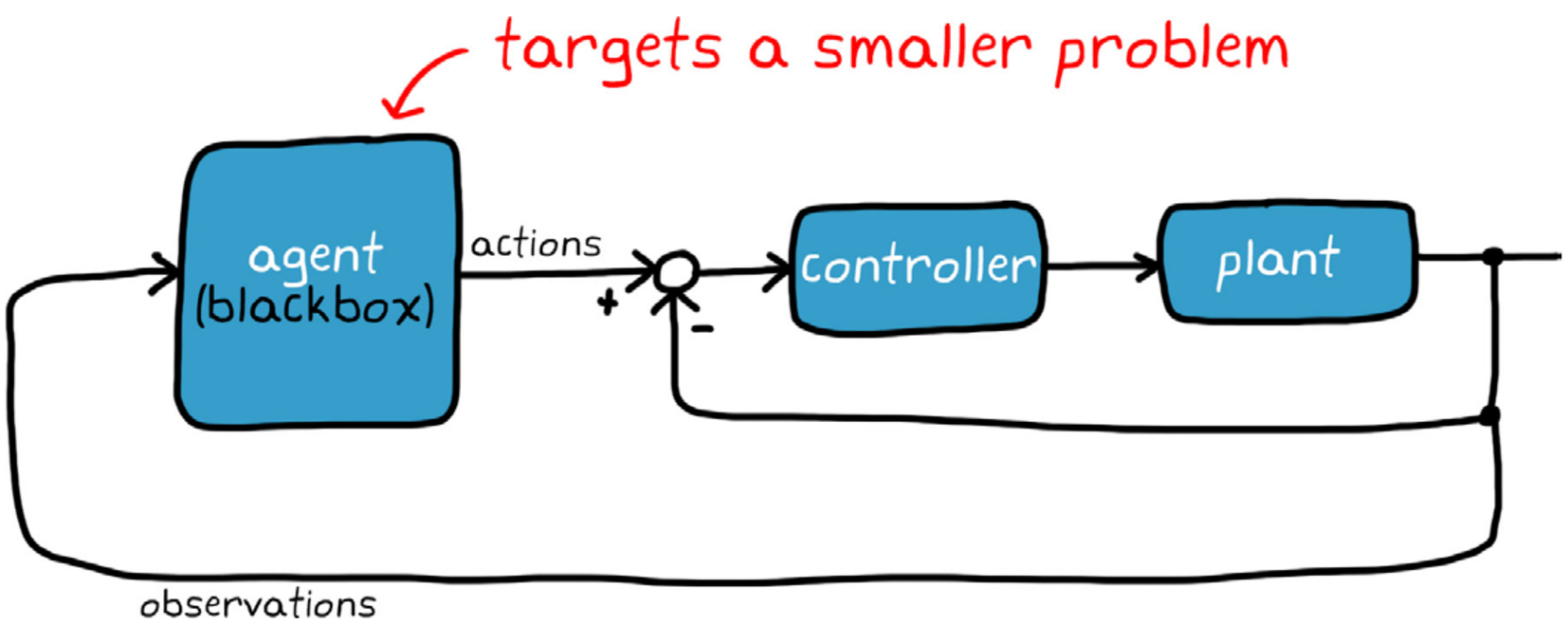


ニューラル ネットワークでは、この種類の形式的検証はより難しくなります。ここまでで議論したように、コードを検査してそれがどう挙動するかを保証することは困難です。またロバスト性や安定性を判定する手法はありません。これはすべて、関数の内部で何が起きているかを説明できないという事実に還元されます。

問題の縮小

これらの問題の規模を小さくする良い方法は、強化学習エージェントのスコープを狭めることです。最も高いレベルの観測を取り込み、最も低いレベルの行動を命令する方策を学習するのではなく、従来のコントローラーを強化学習エージェントでラップして、高度に専門化された問題のみを解決するようにします。強化学習エージェントがターゲットにする問題を小さくすることで、説明できないブラックボックスを、従来の手法では難しく解決できないシステムの一部分へと縮小できます。

方策を小さくすると焦点も絞られるため、何が起きているかも理解しやすくなります。また、システム全体への影響も限定され、学習の時間も短縮されます。ただし、方策を縮小することによっては、問題は解決しません。複雑さを緩和するだけです。不確実性に対してロバストであるか、安定性の保証はあるか、またシステムが仕様を満たすか検証できるかは、まだ分かりません。



これらの問題の回避

ロバスト性、安定性、および安全性を定量化することはできませんが、設計における回避策でこれらの問題に対処できます。

ロバスト性および安定性については、重要な環境パラメーターがシミュレーション実行の度に毎回調整される環境において、エージェントを学習させることができます。

たとえば、ロボット歩行に関する各エピソードの開始時に異なる最大のトルク値を選択した場合、方策は最終的に製造公差に対しロバストなものに収束します。このようにすべての重要なパラメーターを微調整することが、結果として全体をロバストな設計にすることに役立ちます。特定のゲイン余裕や位相余裕を主張することはできないかもしれませんが、その結果が幅広い範囲の動作状態空間を処理することができるという信頼性が向上します。

episode	torque	length	delay	reference	...
1	2 Nm	1 cm	10 ms	step	.
2	2.5 Nm	1.3 cm	8 ms	ramp	.
3	2.1 Nm	1.7 cm	14 ms	impulse	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

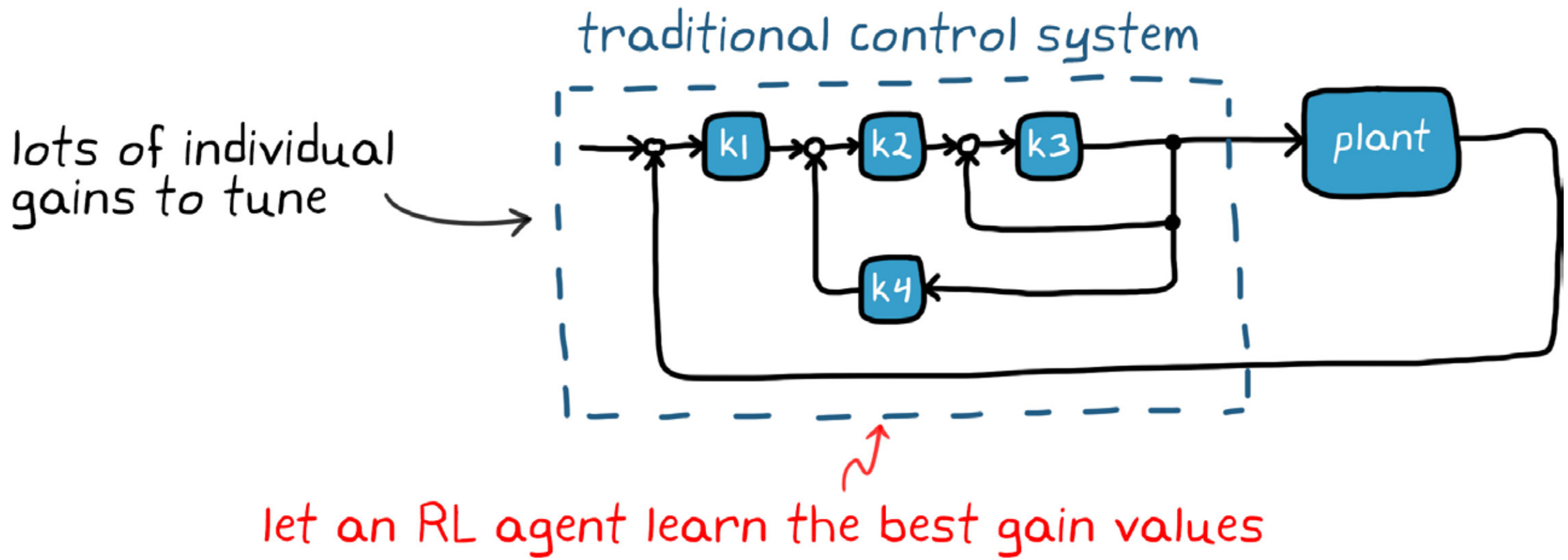
```
% software monitor
if abs(body_angle) > 45;    % monitor for falling
    mode = "safe";          % set safe mode
    extend_arms();          % prepare for impact
end
```

また、システムに確実に回避させたい状況を決め、その状況を監視するソフトウェアを方策の外側で構築することで安全性を向上させることができます。そしてこのモニターがトリガーされたら、損害を引き起こす可能性が生じる前に、システムを拘束するか引き継いで、ある種のセーフ モードに置くことができます。

これによって危険な方策を展開してしまうことは無くなりませんが、システムは守られ、それが失敗する方法を学んで、報酬と学習環境を調整し、その障害に対処することができます。

他の問題の解決

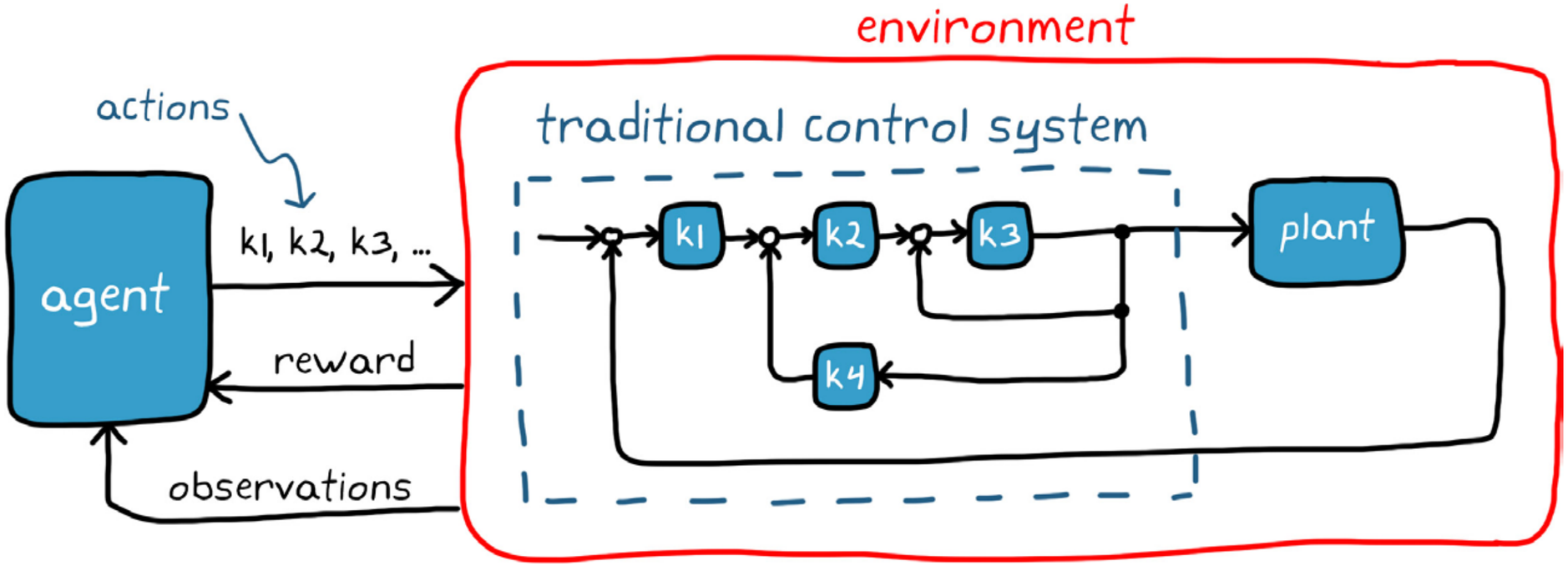
回避策も良い案ですが、他の全く異なった問題を解決することでこの課題に対処することもできます。強化学習は、従来の方法で構築された制御システムの コントローラー ゲインを最適化するツールとして使用することができます。それぞれにゲインがいくつかある、数十の入れ子になったループとコントローラーを持つアーキテクチャの設計を想像してみましょう。結果として、100 以上の個別のゲイン値を調整しなければいけない状況になることもあります。これらの各ゲインを手作業で調整するのではなく、強化学習エージェントをセットアップして、これらすべてに対する最も良い値を一度で学習することができます。



従来の手法を補足する強化学習

制御システムとプラントにより構成される環境を想像してみてください。報酬は、システムの性能の良さとその性能を出すための負荷になり、行動はシステムのゲインになるでしょう。各エピソードの後に、学習アルゴリズムはニューラル ネットワークを微調整して、報酬を増やす方向にゲインを更新します (性能を向上させ負荷を減らす)。

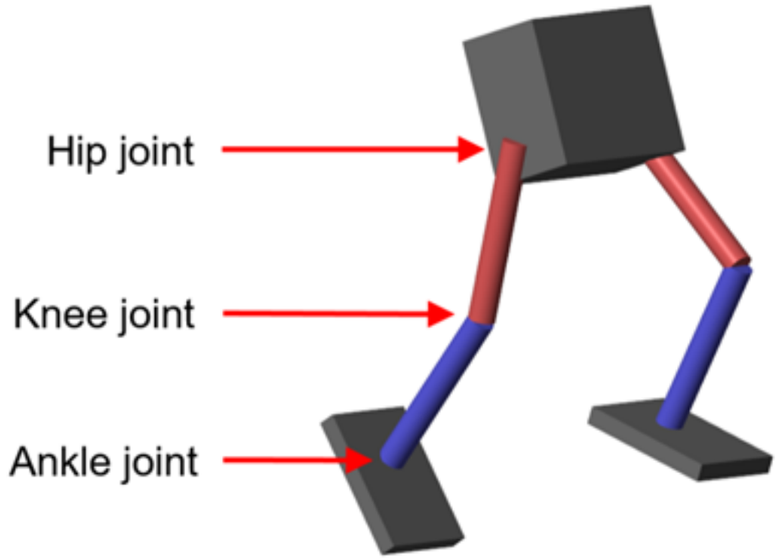
この方法で、両方の長所を生かすことができます。ニューラル ネットワークの展開および検証や、その変更の必要はありません。コントローラーに対する最終的な静的ゲイン値をコーディングするだけです。この方法ではまだ、ハードウェアで検証して手作業で調整できる、従来の方法で構築されたシステムです。しかし、強化学習を用いて選択された最適なゲイン値が入力されています。



強化学習の将来

強化学習は難題を解く強力なツールです。解を理解してそれが機能することを検証する点に課題はありますが、ここまで説明したように、現在、これらの課題を回避するいくつかの方法があります。強化学習はまだ可能性を最大限に発揮していませんが、すべての複雑な制御システムで最初に選ばれる設計手法となるのは、それほど遠い未来ではないかもしれません。

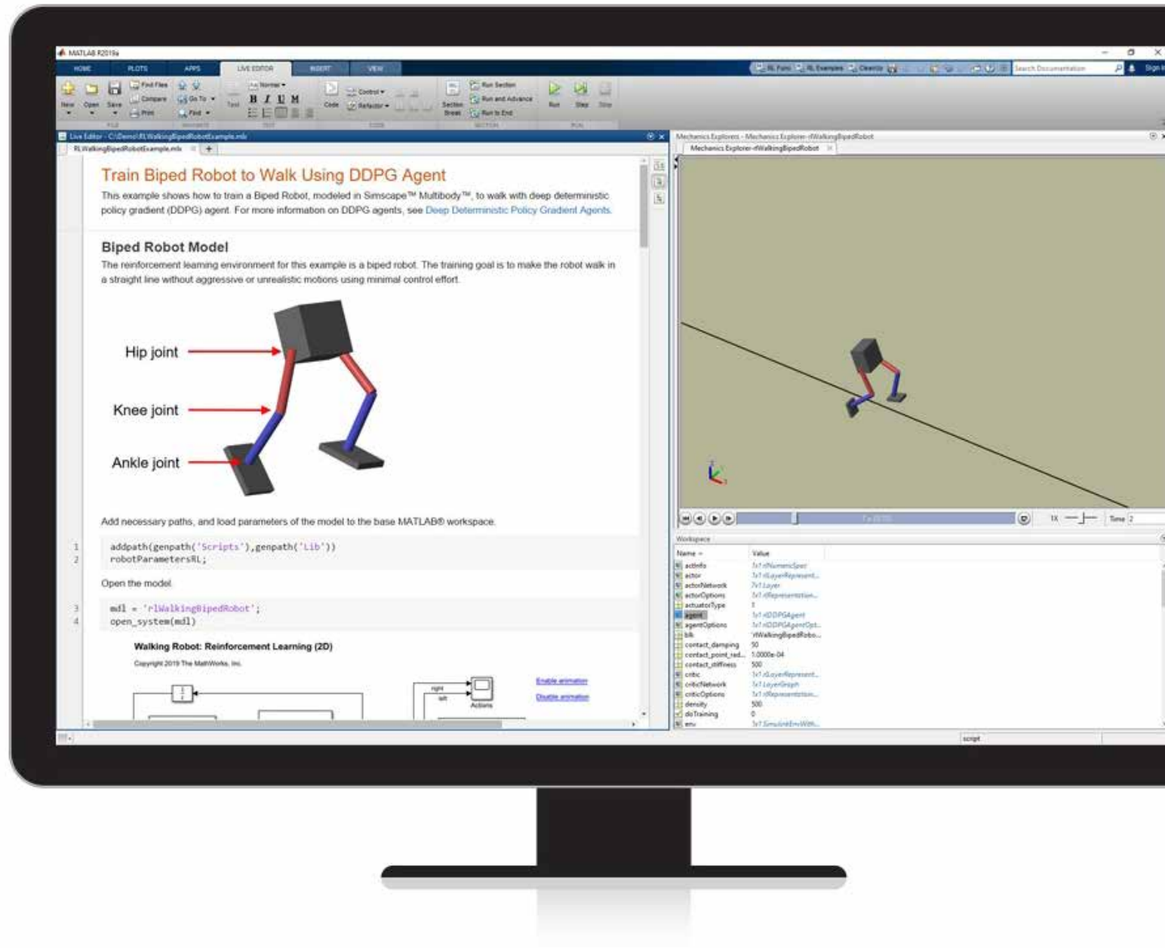
学習アルゴリズム、MATLAB および Reinforcement Learning Toolbox™ のような強化学習設計ツール、そして検証手法は常に進歩しています。



MATLAB による強化学習

Reinforcement Learning Toolbox では、強化学習アルゴリズムを使用して方策を学習する関数およびブロックが用意されています。これらの方策を使用して、ロボットや自律システムなどの複雑なシステムのためのコントローラーと意思決定アルゴリズムを実装できます。

ツールボックスでは、ディープ ニューラル ネットワーク、多項式、またはルックアップテーブルを使用して方策を実装できます。MATLAB® または Simulink® モデルで表現された環境との対話を可能にすることで、方策を学習させることができます。



関連情報

基本的なグリッドワールドで強化学習エージェントを学習 - ドキュメンテーション

Actor-Critic エージェントの学習によるカートポールのバランス制御

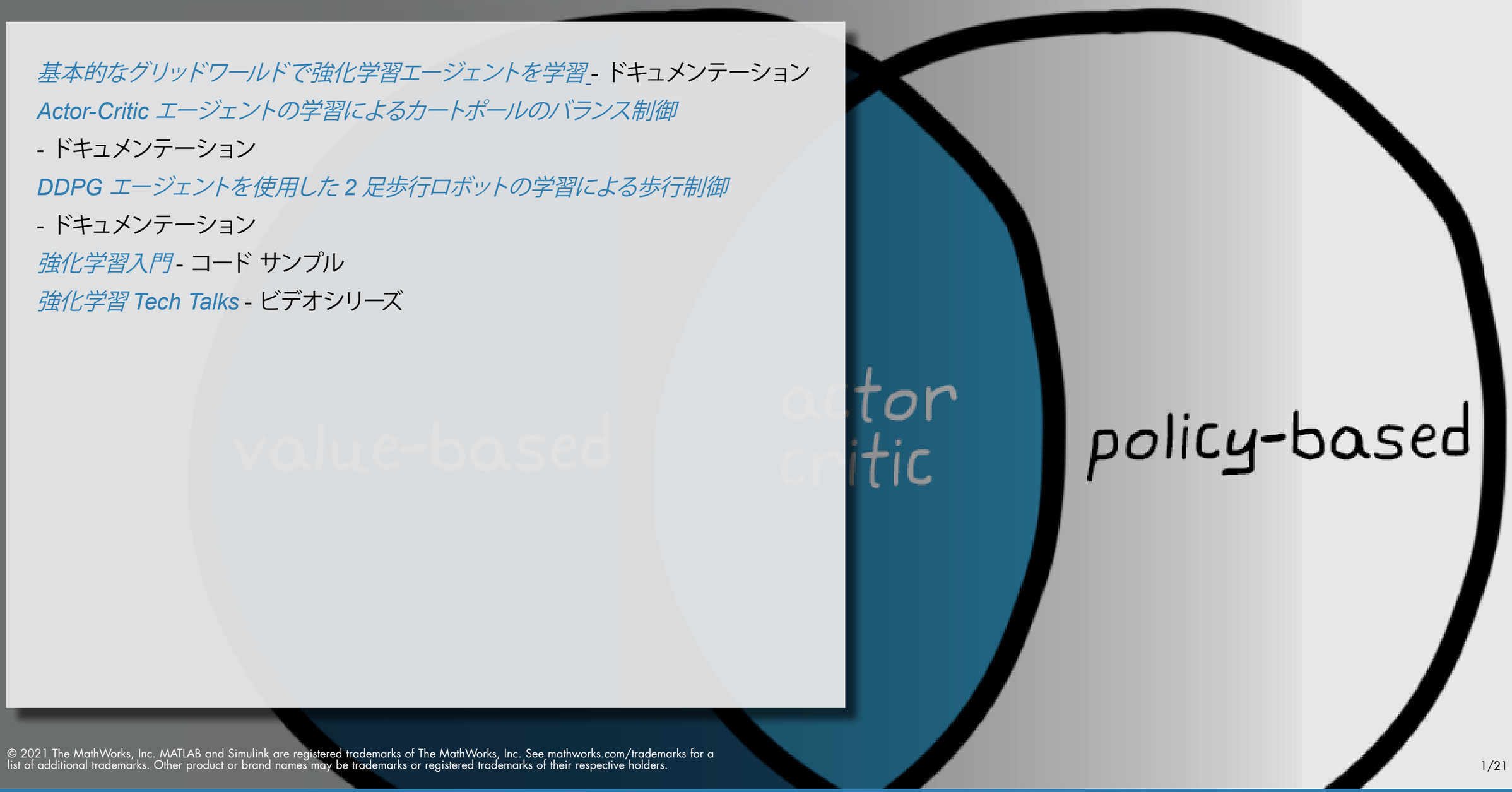
- ドキュメンテーション

DDPG エージェントを使用した 2 足歩行ロボットの学習による歩行制御

- ドキュメンテーション

強化学習入門 - コード サンプル

強化学習 Tech Talks - ビデオシリーズ



MathWorks