単純パーセプトロン

作成者：辻　大貴

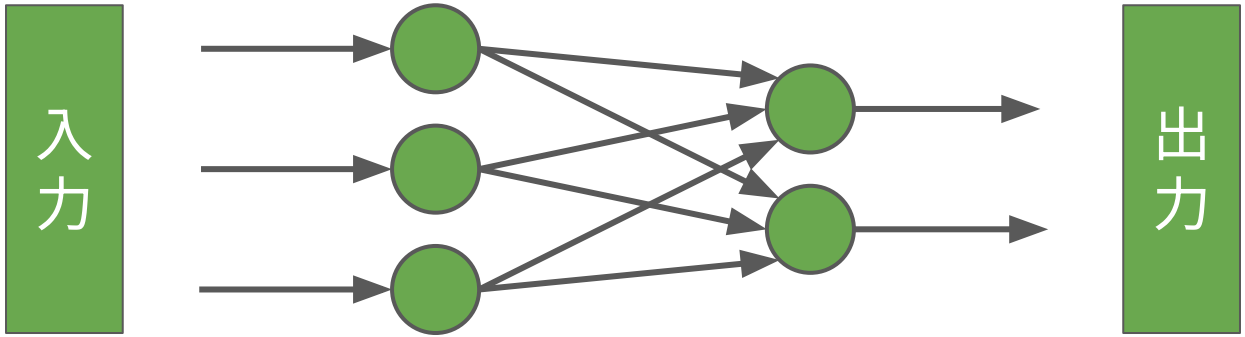
単純パーセプトロン

- **ニューラルネットワーク**

人間の**神経回路**（**ニューロン**）を模倣した数理モデル

→ニューロンは他のニューロンから信号を受け取り、

一定の信号を受け取ると、他のニューロンに信号を送る

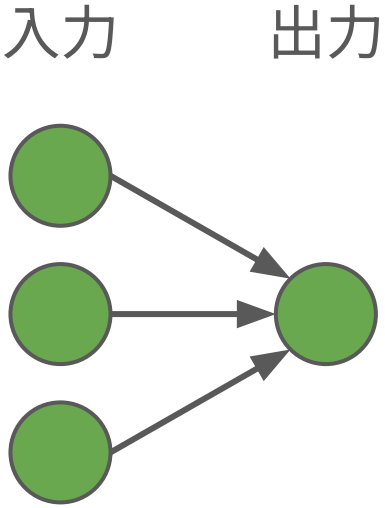


単純パーセプトロン

- **単純パーセプトロン**

  1958年に提案されたニューラルネットワーク

→**入力層**と**出力層**からなる



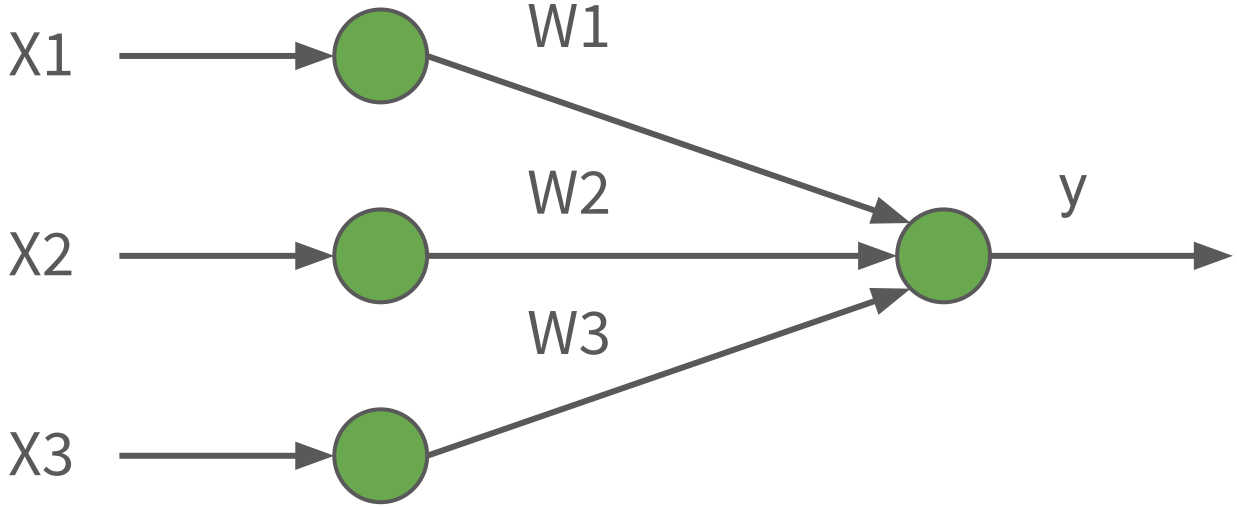
入力層：情報を受け取る層のこと

出力層：処理した値を出力する層のこと

単純パーセプトロン

入力値の重要度、貢献度を数値化したものを**重み**という

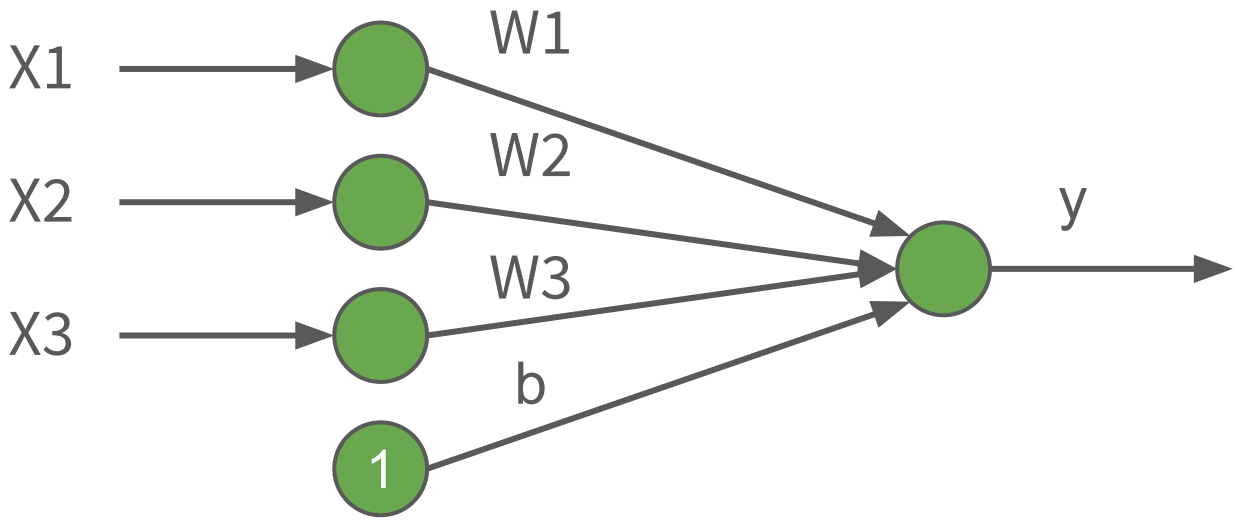
正しい予測ができるように**重み**などを調整してくことを**学習**



単純パーセプトロン

モデルの**自由度**を上げるために**バイアス**を使用する

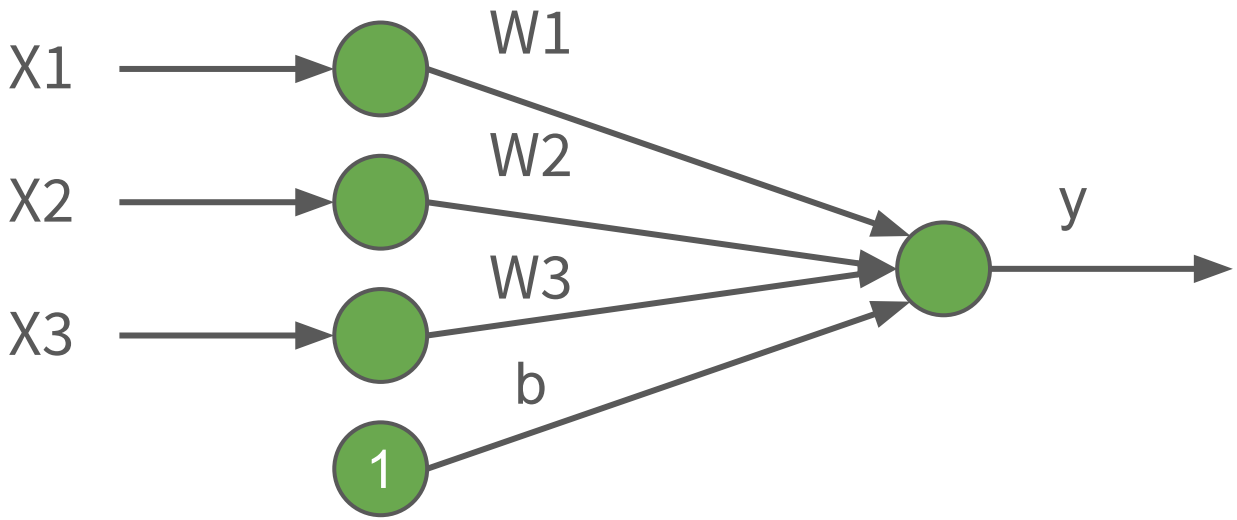
→意図的に偏りを作ることで、求める結果を出力させやすくなる



単純パーセプトロン

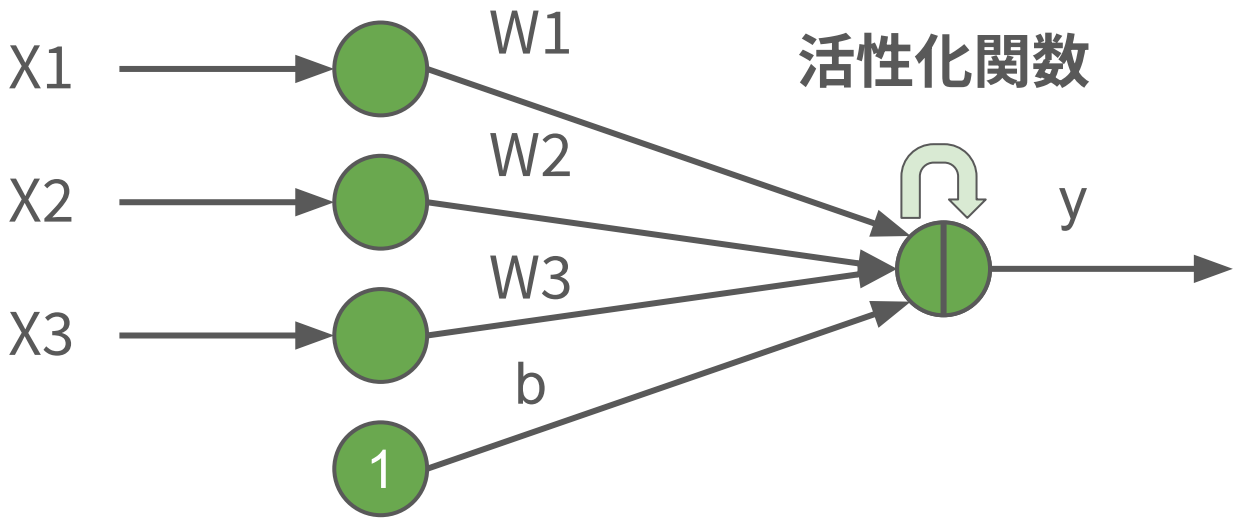
適切な出力を行うために、重みやバイアスなどの

調整するべき値のことを**パラメータ**という



単純パーセプトロン

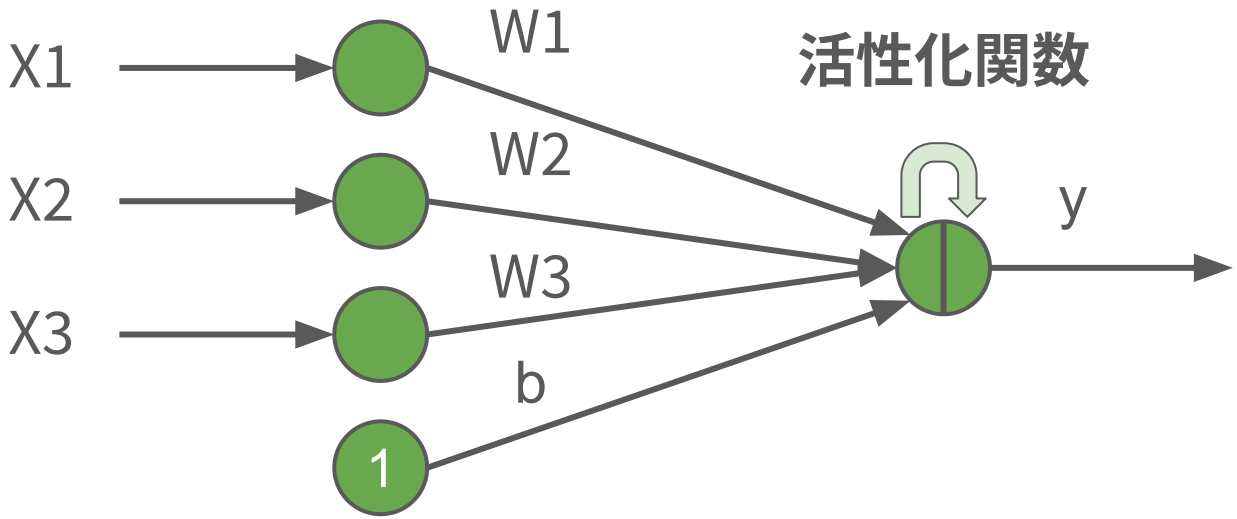
ノードから値を受け取り、次の層に値を受け渡すときに、
受け取った値を調整する関数のことを**活性化関数**という



単純パーセプトロン

「X1 × W1 + X2 × W2 + X3 × W3 + b」の値を

活性化関数を通して次の層に伝えている（ yの値になる ）



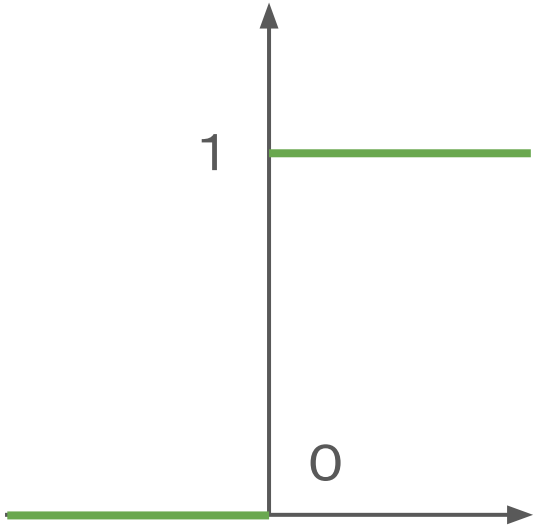
単純パーセプトロン

- **単純パーセプトロン**

活性化関数として**ステップ関数**が用いられる

→入力値が0未満なら出力値が0、

入力値が0以上なら出力値が1になる関数



多層パーセプトロン

作成者：辻　大貴

多層パーセプトロン

- **多層パーセプトロン**

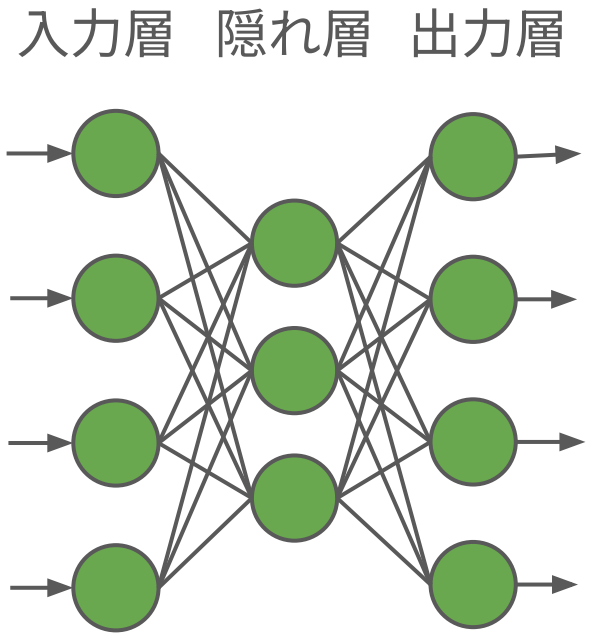
  多層化したニューラルネットワーク

→**非線形分類**が可能になった

  単純パーセプトロンは**線形分類**のみ

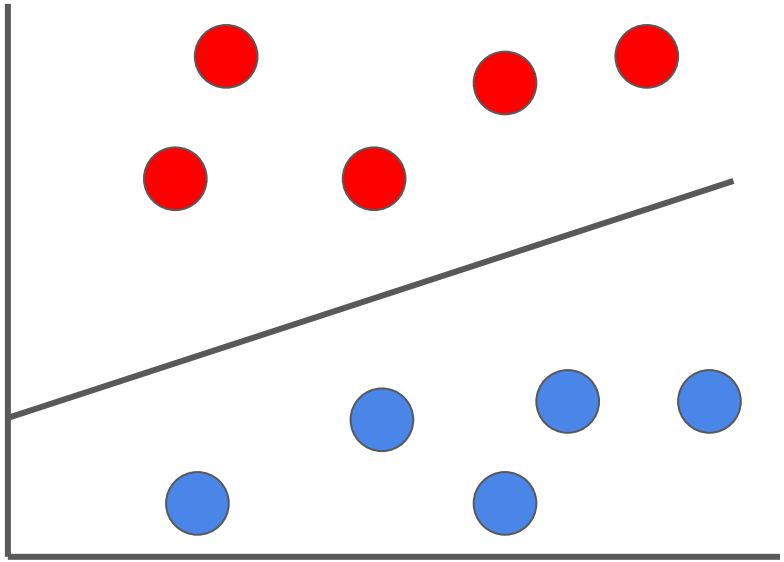
→**入力層**、**隠れ層**、**出力層**に分かれている

  隠れ層は**中間層**とも呼ばれている

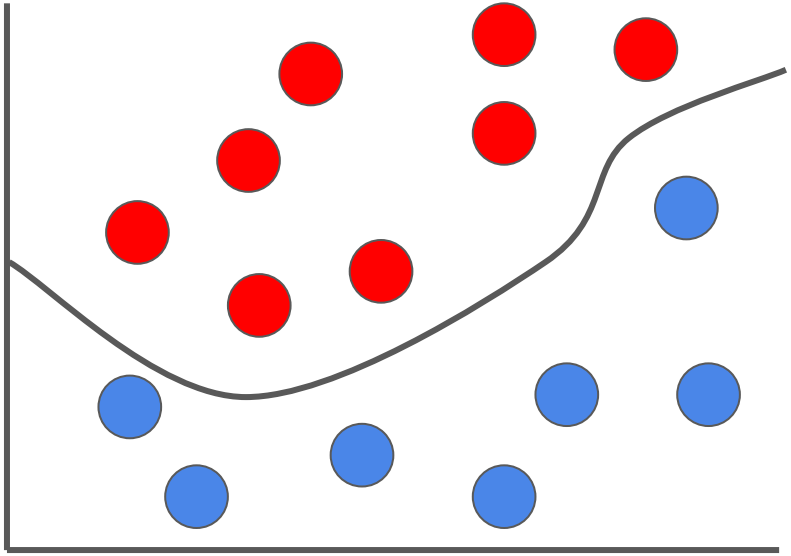


多層パーセプトロン

線形分類



非線形分類



多層パーセプトロン

・**多層パーセプトロン**

層が増えることで**重み**も増えるため従来の方法では困難

→従来は重みをランダムに決めて出力結果に応じて

値を修正していくというような方法が用いられていた

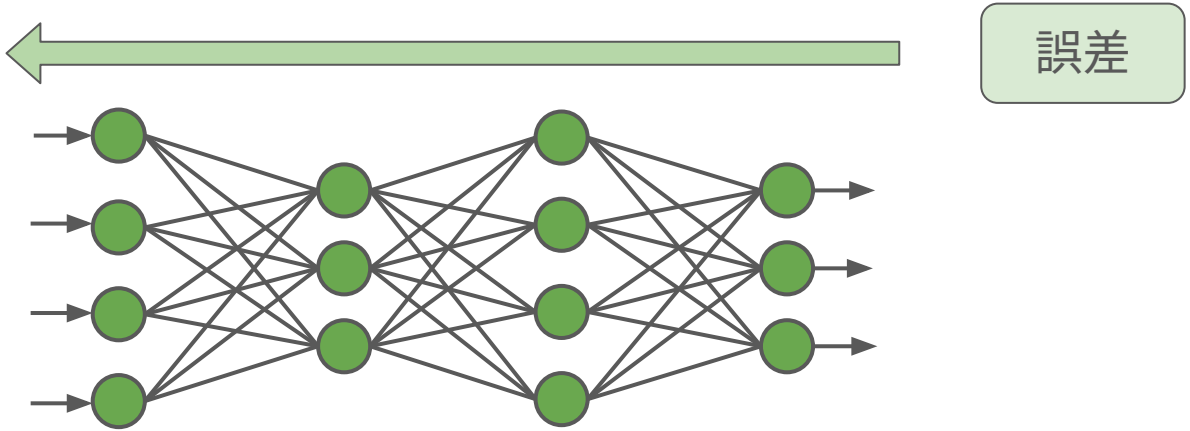
→重みが増えすぎると、どの値を修正すればいいのか分からない

多層パーセプトロン

- **多層パーセプトロン**

  出力層から入力層にかけて重み等を調整していく手法である

  誤差逆伝播法が活用されるようになってきた



多層パーセプトロン

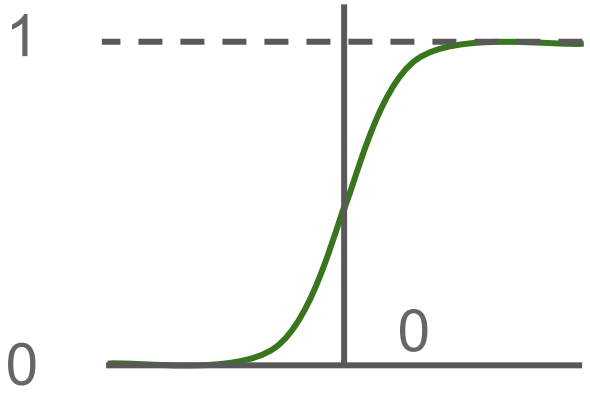
・多層パーセプトロン

誤差逆伝播法では**活性化関数**を**微分**するステップが存在する

→ステップ関数を微分した値は「0」で、誤差逆伝播法に向かない

→活性化関数として**シグモイド関数**などが
用いられるようになった

→微分した値が「0～0.25」になる



多層パーセプトロン

- **多層パーセプトロン**

  誤差逆伝播法では**合成関数の微分**や

  連鎖律（チェインルール）などの微分の公式が使用されている

→紹介したような単純パーセプトロンや多層パーセプトロンは

入力層から出力層にかけて信号が伝播するため

順伝播型ニューラルネットワークと呼ばれている

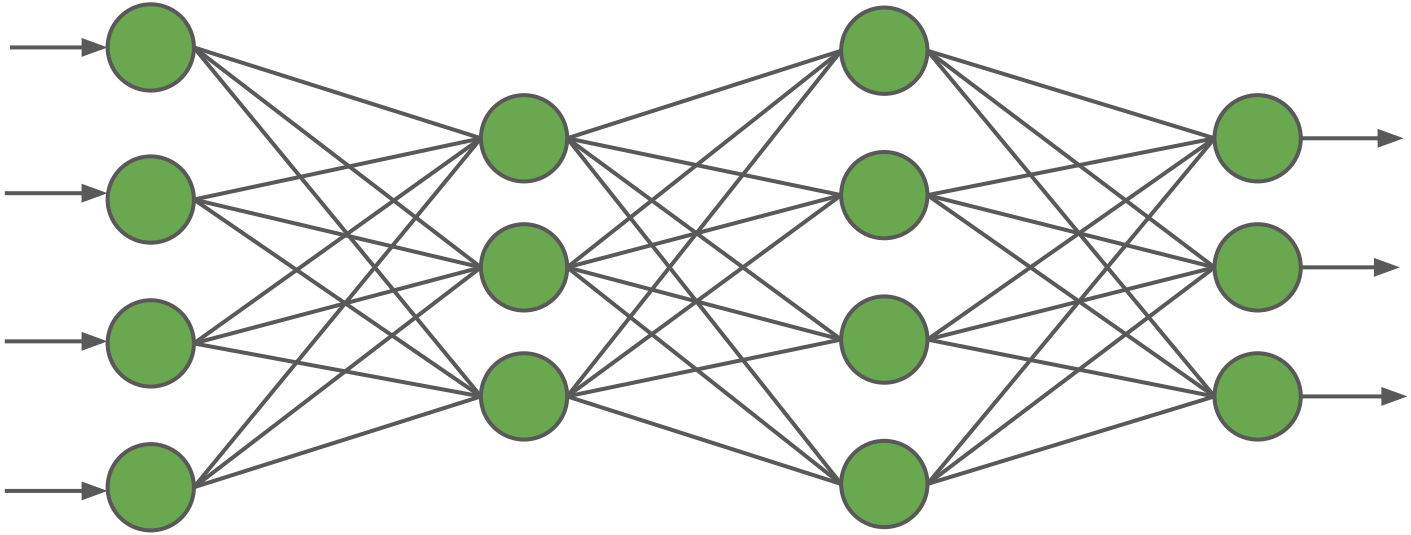
ディープラーニング

作成者：辻　大貴

ディープラーニング

- **ディープラーニング**

  多層化したニューラルネットワーク（**深層**）を使用した手法



ディープラーニング

・**ディープラーニング**

層を深くすることで解ける問題が増えると考えれたが

信用割当問題・勾配消失問題など様々な問題から

思ったような結果が得られなかった

→問題を解決するために色々な工夫がなされた

→問題が解決されたことで解ける問題の幅は広がった

ディープラーニング

- **信用割当問題**

  モデルが導き出した結果が間違っていた場合、

  どのパラメータに責任があるのか、

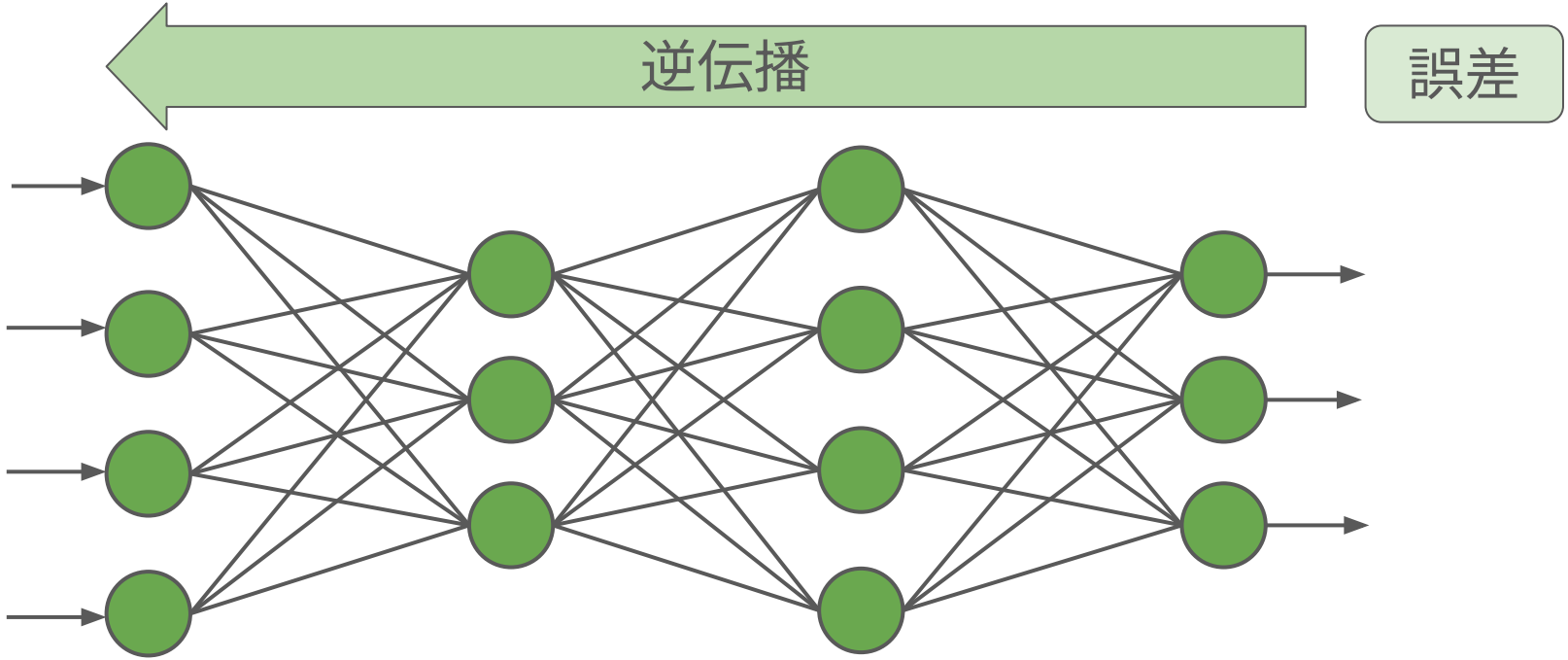
  どのパラメータを修正すればいいのか分からないという問題

→パラメータが数千や数万になると修正対象が分からない

→**誤差逆伝播法**により信用割当問題は解決されたとされている

ディープラーニング

- 信用割当問題



ディープラーニング

- **勾配消失問題**

  出力層から入力層に向けて**勾配**が緩やかになっていき、

  入力層付近で実質的に重みの調整できなくなり、

  学習が進まなくなってしまう問題のこと

→学習時に**勾配**を使用するために発生する問題

→勾配とは微分を用いて求められる関数の傾きのこと

ディープラーニング

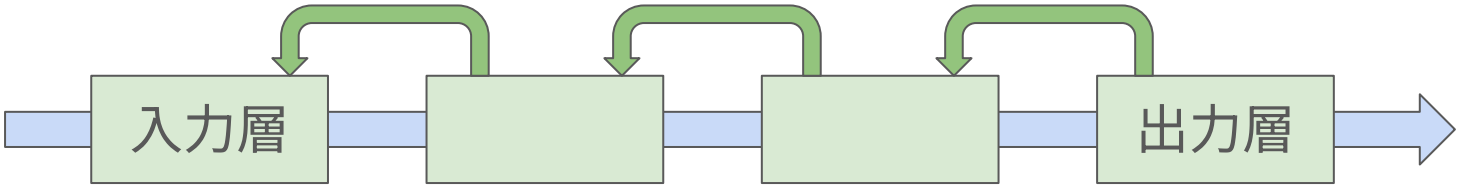
- **勾配消失問題の発生理由**

誤差逆伝播法では学習時に活性化関数の微分した値を使用する

→微分した値を掛け合わせて使用していく

→シグモイド関数の微分した値の範囲は 0〜0.25 であるため

値を掛け合わせていくうちに限りなくゼロになってしまう



ディープラーニング

- **勾配爆発問題**

  勾配が大きくなってしまい、学習が進まなくなってしまう問題

ディープラーニングを実現するには

作成者：辻　大貴

ディープラーニングを実現するには

人工知能の理論などは数多く存在したが

計算コストが高いことからサービスとして普及しづらかった

→**半導体の性能**が向上したことで**計算コスト**が低くなり

多くの人工知能が開発されるようになってきた

・**ムーアの法則（ゴードン・ムーア**が提唱）

半導体の集積密度は18ヶ月で2倍になるという経験則

ディープラーニングを実現するには

- **CPU（Central Processing Unit）**

  パソコンの頭脳と言われており、汎用的な演算処理を行う装置

→基本的に逐次的に演算処理を行う

- **GPU（Graphics Processing Unit）**

  並列的な演算処理が得意で、

  単純な演算処理が必要な画像処理などが向いている装置

ディープラーニングを実現するには

- **CPU（Central Processing Unit）**

  演算処理を行うコアの数は数個程度である

→コア数が多いほど並列処理が得意になる

→CPUはGPUよりも複雑な命令の処理に適している

- **GPU（Graphics Processing Unit）**

  演算処理を行うコアの数は数千個程度である

ディープラーニングを実現するには

ディープラーニングでは同じような計算が大量になされている

→GPUは並列的に処理をするのが得意であるため

GPUとディープラーニングは相性が良い

→画像処理が得意なGPUを、画像処理以外の処理でも

活用できるように改良した演算処理装置が**GPGPU**である

ディープラーニングを実現するには

- **GPGPU**

  General-Purpose computing on Graphics Processing Units

→**GPUによる汎用計算**

→アメリカにある**NVIDIA社**などが様々なGPUを開発している

　汎用並列コンピューティングプラットフォーム（**CUDA**）を提供

→GPUを利用した開発プラットフォームのこと

ディープラーニングを実現するには

- **TPU（Tensor Processing Unit）**

  Googleが開発した機械学習に特化した演算処理装置

ディープラーニングを実現するには

・**ディープラーニングのデータ量**

学習に必要なデータ量は明確に決まっていない

→経験則としてモデルのパラメータ数の10倍は必要と言われている（**バーニーおじさんのルール**）

→パラメータ数が1,000万ならば、データ量は1億が必要

現実的に難しいためパラメータ数を減らす等工夫が行われている

学習データの投入方法

作成者：辻　大貴

学習データの投入方法

- **学習データの投入方法（学習方法）**

  データの投入方法（学習方法）として、**バッチ学習**、**オンライン学習**、**ミニバッチ学習**があり、それぞれメリットとデメリットがある

→ニューラネルネットワークでは**ミニバッチ学習**が頻繁に使用される学習方法である

学習データの投入方法

- **バッチ学習**

訓練データ全体を投入してモデルを学習させる手法

→学習結果は安定しやすいが、全ての訓練データを使用するため、
計算負担が大きいという特徴がある

→訓練データ全体を投入するため、訓練データ内に
異常なデータが少し混じっていてもモデルへの影響は少ない

学習データの投入方法

- **バッチ学習**

  訓練データが増えると、1から計算をやり直す必要があるため、

  リアルタイムでモデルを更新することは難しい

→随時更新するような株価などの予測には向かない

学習データの投入方法

- **オンライン学習**

  ランダムに訓練データを1件ずつ投入して

  モデルを学習させていく手法のこと

→1件ずつデータを投入して学習していくため、

  学習が安定しにくいが、リアルタイムでモデルを更新できる

→データを1件ずつ使用するためメモリの使用量が少ない

学習データの投入方法

- **オンライン学習**

  1件ずつデータを投入して学習させていくため、

  異常なデータが少し混じっていると、

  モデルの精度が落ちやすくなってしまうという特徴がある

→異常なデータを取り除く仕組みが必要になる

→1件異常なデータが入っていると誤った学習を行ってしまう

学習データの投入方法

- **ミニバッチ学習**

訓練データをいくつかのグループに分けて、

各グループごとにモデルを学習させていく手法のこと

→バッチ学習とオンライン学習の中間的な学習手法

ニューラルネットワークでよく使われる学習手法になる

→1,000件のデータがある場合、100件ずつ学習させていく

学習データの投入方法

・**ミニバッチ学習**

全体のデータから分割したデータ数を**バッチサイズ**という

→1,000件のデータがあり、100件ずつ学習させていく場合

バッチサイズは100になり、少なくとも10回学習させていく

→100 × 10で1,000件のデータを学習させていく

学習データの投入方法

- ミニバッチ学習

	バッチ学習	ミニバッチ学習	オンライン学習
メモリ使用量	多い	中間	少ない
1データあたりの計算速度	速い	中間	遅い
異常データから受ける影響	小さい	中間	大きい
学習の安定度	高い	中間	低い

学習データの投入方法

- **イテレーションとエポック**

  学習とは最適なパラメータ（重み）を見つけ出すために
  何度もパラメータ（重み）を更新していくことである

→パラメータ（重み）を更新した回数を**イテレーション数**
  パラメータ（重み）を20回更新した場合、
  イテレーション数は20になる

学習データの投入方法

- **イテレーションとエポック**

  学習のために訓練データを繰り返し使った回数を

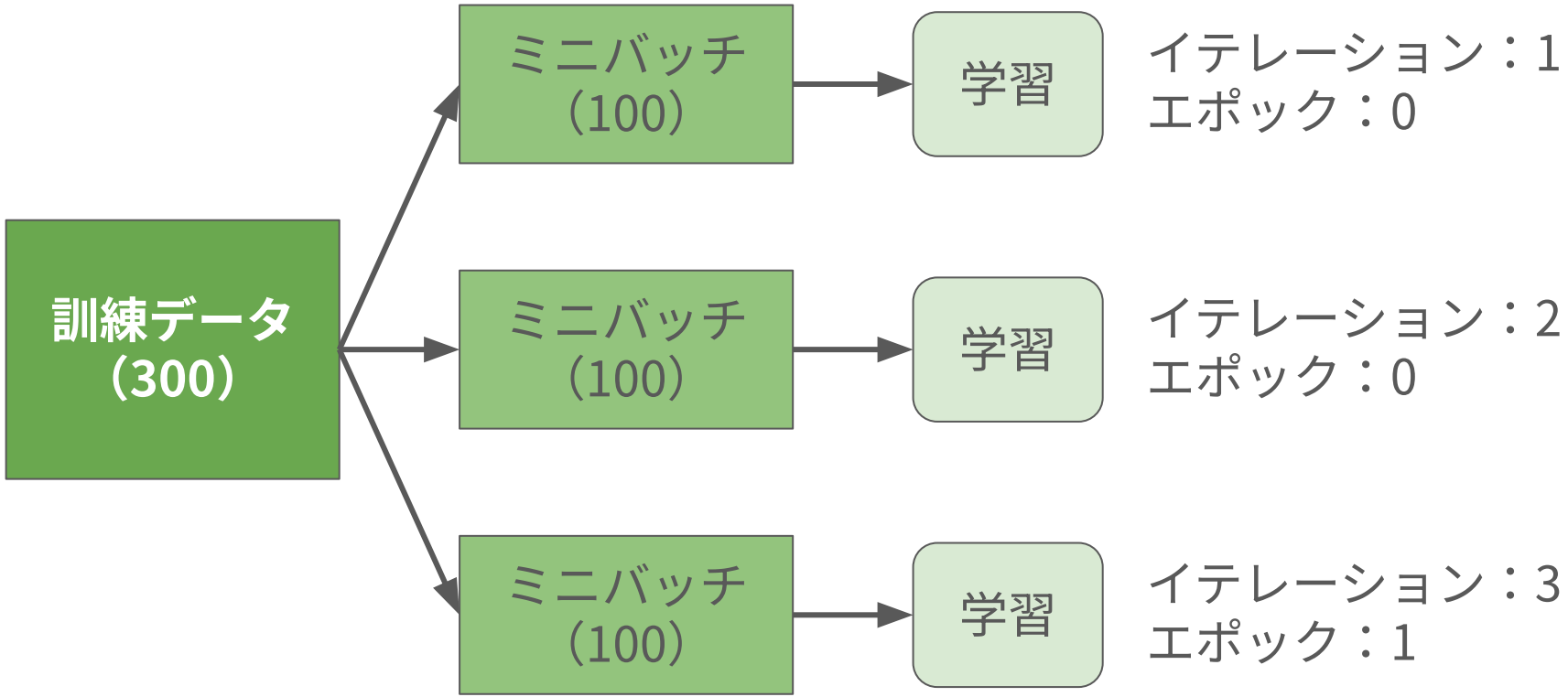
  エポック数という

→訓練データを5回使用した場合はエポック数は5になる

→**イテレーション**はパラメータ（重み）を更新した回数

エポックは訓練データを繰り返し使った回数

学習データの投入方法



誤差関数

作成者：辻　大貴

誤差関数

- **学習**

  ニューラルネットワークでは、予測値と実際の値の**誤差**を

  近くするように学習をしていく

→重みやバイアスなどのパラメータを更新して**最適化**していく

→誤差は**誤差関数（損失関数）**を使用して求めていく

平均二乗誤差関数、**交差エントロピー誤差関数**などがある

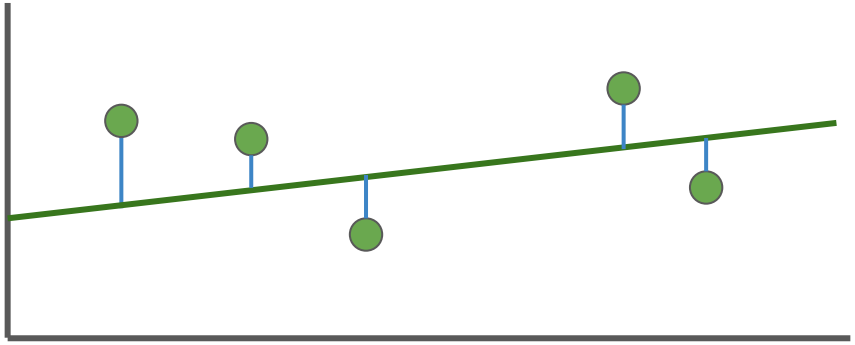
誤差関数

- **平均二乗誤差関数**

主に回帰問題で使用される誤差関数である

→予測値と実際の値の差の2乗和をデータ数で割った値を出力

予測値と実際の値の差の2乗の平均の値を出力している



誤差関数

- **交差エントロピー誤差関数**

  主に分類問題で使用される誤差関数である

→**真の確率分布**と**推定した確率分布**を使用して誤差を求める

→画像に写っている動物を分類する（犬, 猫, キツネ）

犬が写っている場合、真の確率分布は（1, 0, 0）

推定した確率分布が（0.7, 0.1, 0.2）、これらを使って誤差計算

誤差関数

- **誤差関数**

　回帰問題や分類問題に当てはまらない問題も存在する

→顔認識、画像検索、異常検知などが挙げられる

→**データ間の距離（類似度）**を学習し、これを使用して問題を解く

　Aさんの写真と類似度の高い写真を探す　など

→データ間の距離（類似性）を学習することを**距離学習**という

誤差関数

- 誤差関数

データA 

データB 

データC 

データA = [0, 8, 7, 10]

データB = [2, 5, 8, 12]

データC = [10, 15, 2, 3]

誤差関数

- **誤差関数**

  類似度の高い画像は距離が近くなるように、

  類似度の低い画像は距離が遠くなるように学習していく

→ディープラーニングに用いて、モデル自体がデータから

  距離を学習することを深層距離学習という

誤差関数

- **誤差関数**

  深層距離学習には**Siamese Network**や**Triplet Network**がある

  - **Siamese Network**

    ２種類のデータを使用する方法である

  →類似する画像ペアの距離は小さく、

    異なる画像ペアの距離は大きくするように学習していく

誤差関数

- **誤差関数**
  - **Triplet Network**

    基準データ(アンカーデータ)、類似データ(ポジティブデータ)、
    異なるデータ(ネガティブデータ)を使用する方法

→基準データと類似データの距離は小さく、
基準データと異なるデータの距離は大きくするように学習

誤差関数

- **誤差関数**

Siamese Networkでは誤差関数として**Contrastive Loss**が使用

→２つの画像を使用して誤差を求めている

Triplet Networkでは誤差関数として**Triplet Loss**が使用

→３つの画像を使用して誤差を求めている

勾配降下法

作成者：辻　大貴

勾配降下法

- **学習**

　関数の最小値（誤差の最小値）を見つけるために**微分**を使用

→変数（パラメータ）が1つならば、**微分した値が0**のとき

　関数の最小値（誤差の最小値）になる

→誤差関数には多数の変数（パラメータ）が含まれており、

　計算の難易度が一気に上がる（簡単に求めることができない）

勾配降下法

- **学習**

1つの変数に注目して微分をすることが**偏微分**という

→ $y = 3a^3 + 8b^3 - 3c^2 + d^4$ のとき、

d で微分をすると $\dfrac{\Delta y}{\Delta d} = 4d^3$ になる

→各変数で微分すると以下のようになる

$$\frac{\Delta y}{\Delta a} = 9a^2 \ , \ \frac{\Delta y}{\Delta b} = 24b^2 \ , \ \frac{\Delta y}{\Delta c} = -6c \ , \ \frac{\Delta y}{\Delta d} = 4d^3$$

勾配降下法

- **学習**

　微分した値を**勾配**といい、勾配が最小になるパラメータを探す

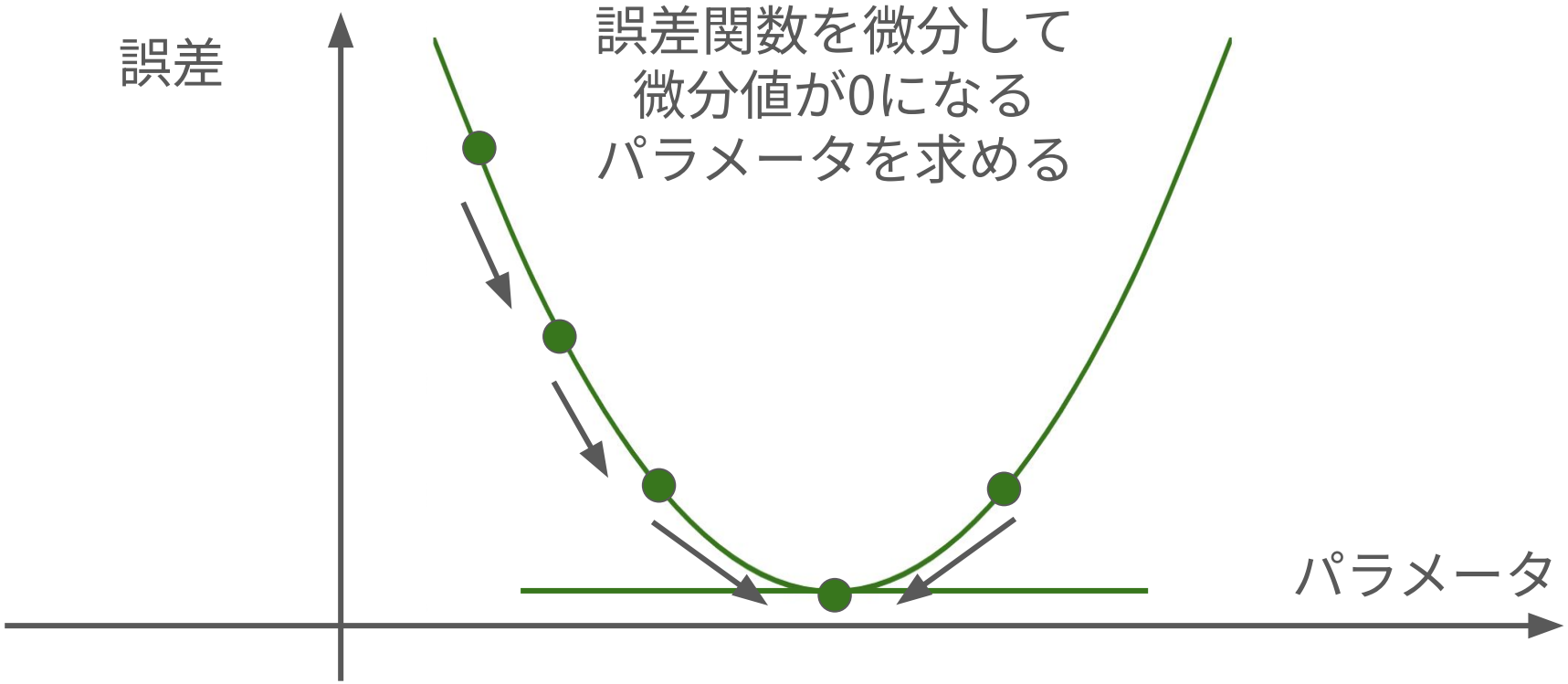
→変数が多いためアルゴリズムを使用してパラメータを探す

　使用するアルゴリズムが**勾配降下法**である

→各パラメータに対して**勾配降下法**を行い、最適解を見つける

　最適解が見つかるまで何度も計算するため時間がかかる

勾配降下法



勾配降下法

- **学習率**

  パラメータを調整する値のことで**専門家**が決める

  どのくらいパラメータの値を動かすかを決めるときに使用

→**学習率**は大きすぎても小さすぎても良くない

→**学習率**が大きすぎたり、小さすぎたりしたときに

発生する問題については次回以降詳しく解説

勾配降下法

- 勾配降下法の種類

  １.最急降下法

  ２.確率的勾配降下法（SGD）

  ３.ミニバッチ勾配降下法

勾配降下法

１.最急降下法

学習データの誤差合計からパラメータを更新する方法

→学習データが多いと計算が多くなり時間がかかる

学習データが増えるたび再計算する必要がある

→パラメータを更新するタイミングでいうと

最急降下法は**バッチ学習**に相当する

勾配降下法

２.確率的勾配降下法（SGD）

学習データをランダムに選び出し、

誤差を計算してパラーメータを更新する手法

→学習データが増えた場合は新しい学習データのみを学習させる

→パラメータを更新するタイミングでいうと

SGDは**オンライン学習**に相当する

勾配降下法

３.ミニバッチ勾配降下法

　最急降下法と確率的勾配降下法の中間的な手法

→学習データを複数個ランダムに選び出し

　誤差を計算してパラーメータを更新する手法

→パラメータを更新するタイミングでいうと

　ミニバッチ勾配降下法は**ミニバッチ学習**に相当する

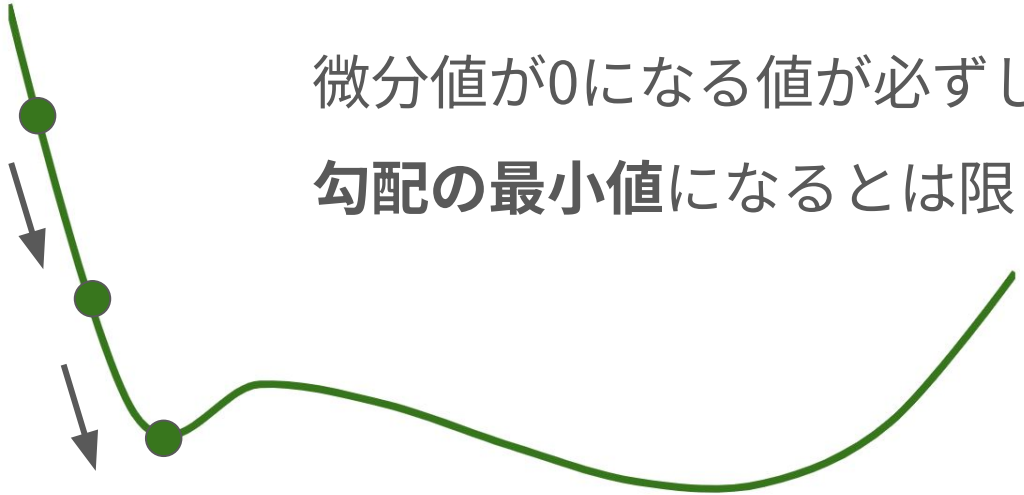
勾配降下法の問題点

作成者：辻　大貴

勾配降下法の問題点

- **勾配降下法の問題点**

  勾配降下法で勾配の最小値が見つからない場合も存在する



微分値が0になる値が必ずしも

勾配の最小値になるとは限らない

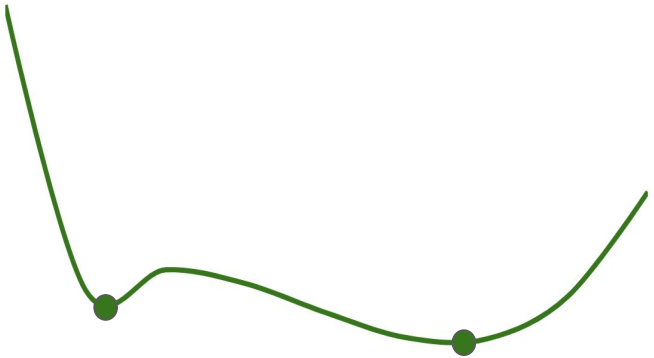
勾配降下法の問題点

- **勾配降下法の問題点**
  - **局所最適解**

    限られた範囲内における最適な解のこと

  - **大域最適解**

    範囲全体で見たときに最適な解のこと



勾配降下法の問題点

- **局所最適解と大域最適解**

局所最適解を防ぐ方法として、**学習率**を大きくする

→小さな山を飛び越えることができるため

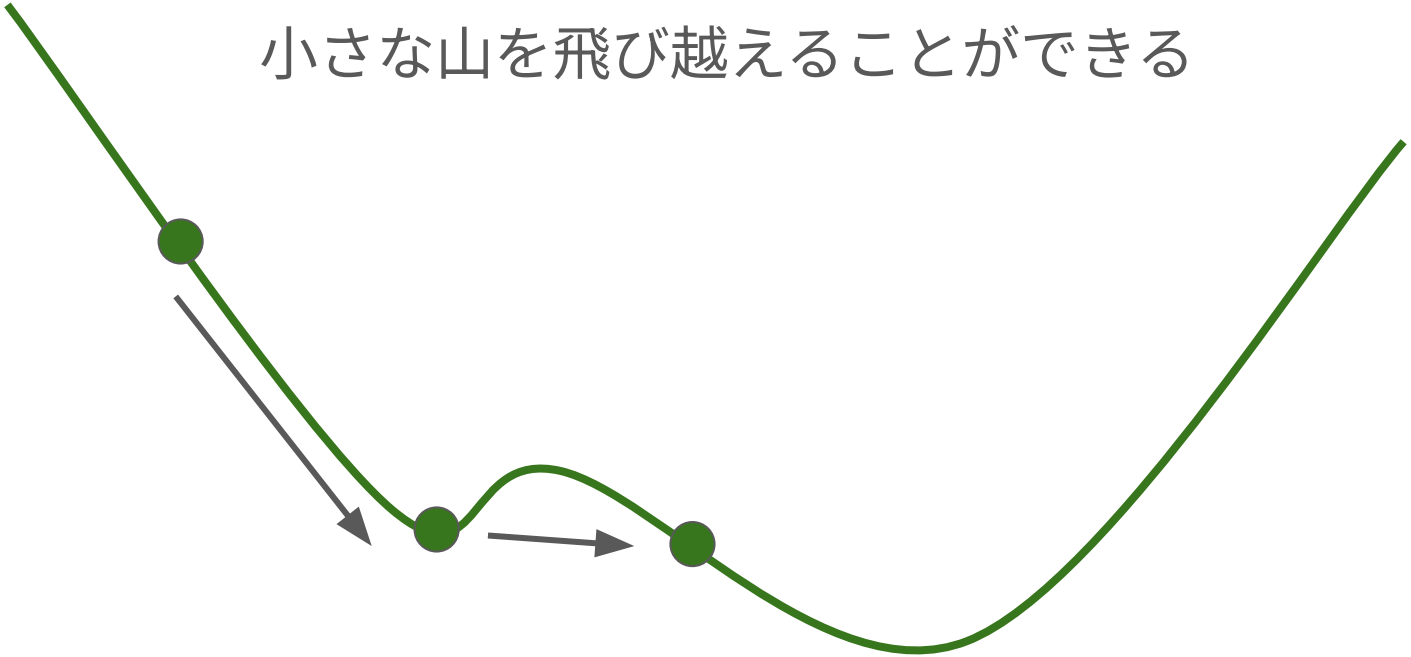
大域最適解を見つけられる可能性が高くなる

→学習率が大きすぎると**大域最適解**を飛び越えることも

最適なタイミングで**学習率**を小さくする

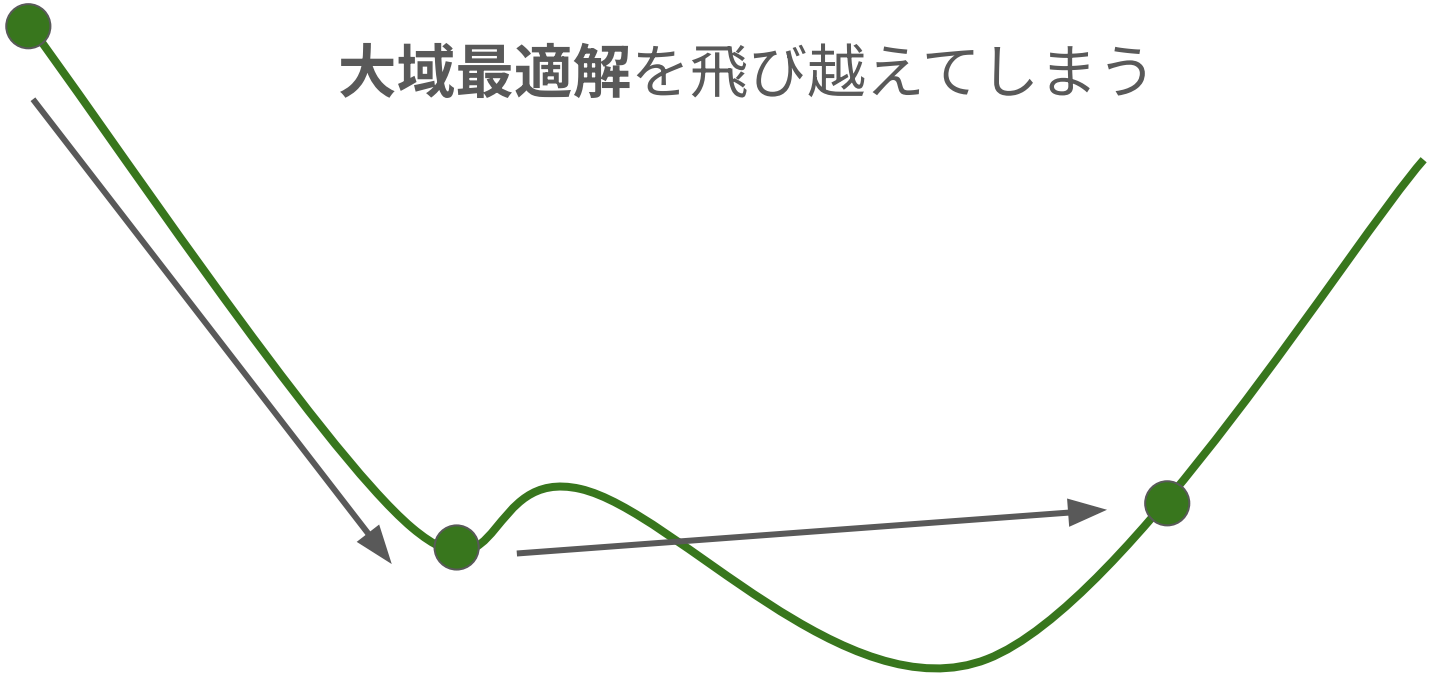
勾配降下法の問題点

- 勾配降下法の問題点



勾配降下法の問題点

- 勾配降下法の問題点

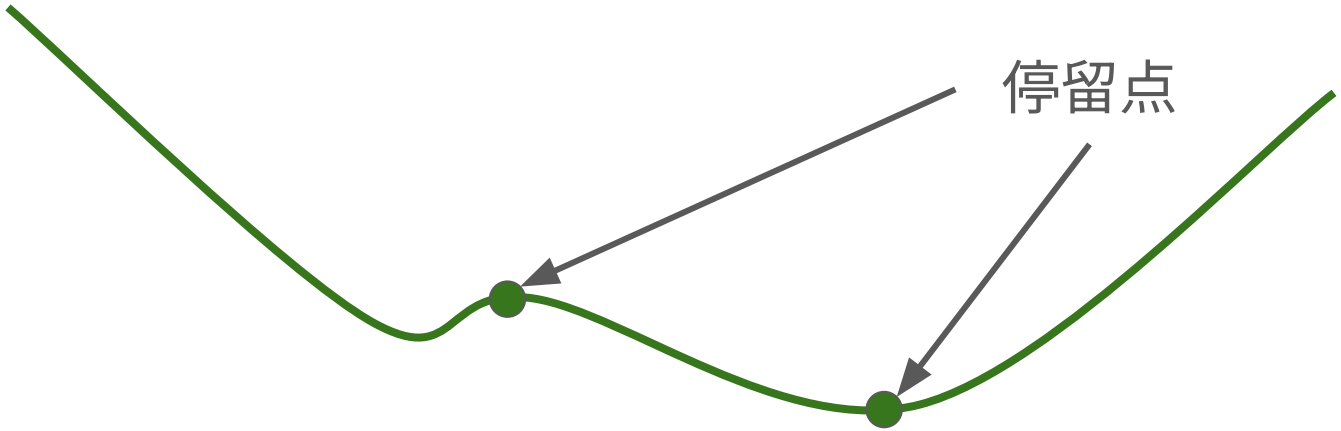


勾配降下法の問題点

- **局所最適解と大域最適解**

  停留点の影響で学習が進みにくくなることがある

→勾配が0になる点のこと、極大点、極小点も停留点の1つ



勾配降下法の問題点

- **鞍点**

  ある次元では極小、別次元から見たときは極大になっている点

→**鞍点**の近くは平坦になっていることが多く

学習が進みにくくなってしまう（**プラトー**という現象）

→鞍点問題に対応するために考えられた手法が**モーメンタム法**

勾配降下法の問題点

- **モーメンタム法**

慣性の考え方を適応し、学習の停滞を防ぐとされる

→前回の更新量を、現在の更新量に反映させる

モーメンタム法を改良した手法として**NAG**がある

→モーメンタム法よりも効率的なアルゴリズムが考えられている

勾配降下法の問題点

- **最適化アルゴリズムの一例**
  - **AdaGrad** ：SGDの改良した手法、学習率を自動で調整
    過去の勾配を蓄積し学習率を小さくしていく
  - **RMSprop**：AdaGradを改良した手法
    新しいパラメータ更新の影響を大きくし、
    学習率が意図しない形で小さくなることを防ぐ

勾配降下法の問題点

- **最適化アルゴリズムの一例**
  - **AdaDelta**：AdaGradを改良した手法
    過去の勾配を蓄積する範囲を制限する
  - **Adam**　　：RMSpropを改良した手法
    RMSpropにモーメンタム法を取り入れた手法

勾配降下法の問題点

- **最適化アルゴリズムの一例**
  - **AdaBound**　：Adamを改良した手法

    最初はAdamを使用して、後半はSGDを使用する

  - **AMSBound**　：AMSGradを改良した手法

    最初はAMSGradを使用して、後半はSGDを使用する

勾配降下法の問題点

・**ハイパーパラメータ**

　機械学習モデルにおいて人間があらかじめ設定するパラメータ

→学習率、ニューラルネットワークの層の数　など

→予測の精度に大きな影響を与えている

　最適な値を自動的に設定しようと考えられている

→**グリッドサーチ**、**ランダムサーチ**、**ベイズ最適化**などがある

勾配降下法の問題点

- **グリッドサーチ**

  あらかじめ用意した全てのパラメータの組み合わせで

  学習を行い、最適な組み合わせを採用する手法

→全てのパターンを調べるため**時間がかかる**

→全てのパターンを試すため、あらかじめ用意した

  パラメータの組み合わせで最適な組み合わせの最適解を発見

勾配降下法の問題点

- **ランダムサーチ**

  指定された分布に従ってランダムにパラメータを抽出し

  学習を行って、最適なパラメータを探す方法

→**指定した範囲内**のハイパーパラメータを

  分布に従ってランダムに**指定回数**分だけ抽出していく

→0～5の範囲内で分布Aに従ってランダムに10回抽出する　など

勾配降下法の問題点

- **ランダムサーチ**

  全てのパターンを調べるわけではないので、

  探索するための時間がグリッドサーチよりも短い

→**最適解**を見つけることができるとは限らない

→時間がある場合はグリッドサーチを使用し、

  時間がない場合はランダムサーチが使用されることが多い

勾配降下法の問題点

- **ベイズ最適化を使用する方法**

  過去の試行結果をもとにして、パフォーマンスの高い

  ハイパーパラメータの値を中心に探索を行う手法のこと

→パラメータAの値が10、パラメータBの値が15で、

  パフォーマンスが高いとき、周辺の値の組み合わせを探索

→探索が偏りすぎないように、探索が少ない箇所も適度に探索

勾配降下法の問題点

- **遺伝的アルゴリズムを使用する方法**

  生物における**遺伝的な変化**（DNAの変化）を模倣した
  アルゴリズム（遺伝的アルゴリズ）を使用して、
  ハイパーパラメータの最適な値を見つけ出していく方法も存在

→ハイパーパラメータの値を調整していくことを
ハイパーパラメータチューニングという

過学習に対するテクニック

作成者：辻　大貴

過学習に対するテクニック

- **過学習**

  訓練データを学習しすぎた結果、

  未知のデータに対する精度が悪くなってしまう現象のこと

→過学習対策手法として**ドロップアウト**、**早期終了**、

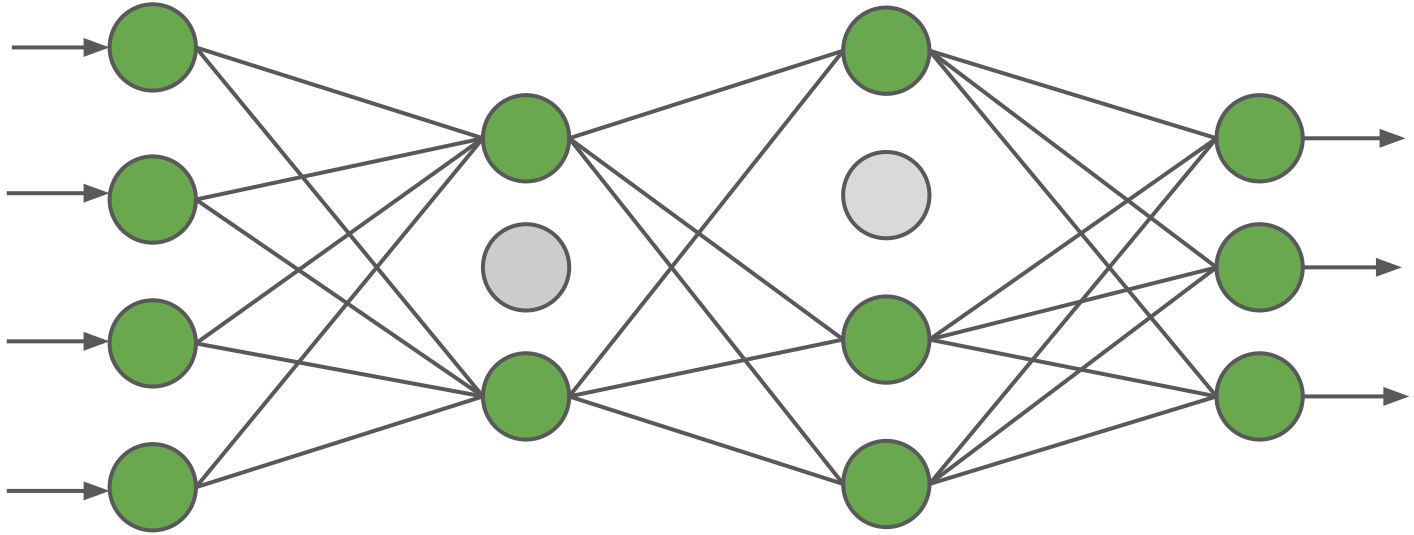
アンサンブル学習、**正則化**などがある

→今回は**ドロップアウト**、**早期終了**について解説していく

過学習に対するテクニック

- **ドロップアウト**

  ミニバッチごとにランダムにユニットを**無効化**する手法のこと



過学習に対するテクニック

- **早期終了**

  過学習が起きる前に学習を終了する方法（汎用性が高い）

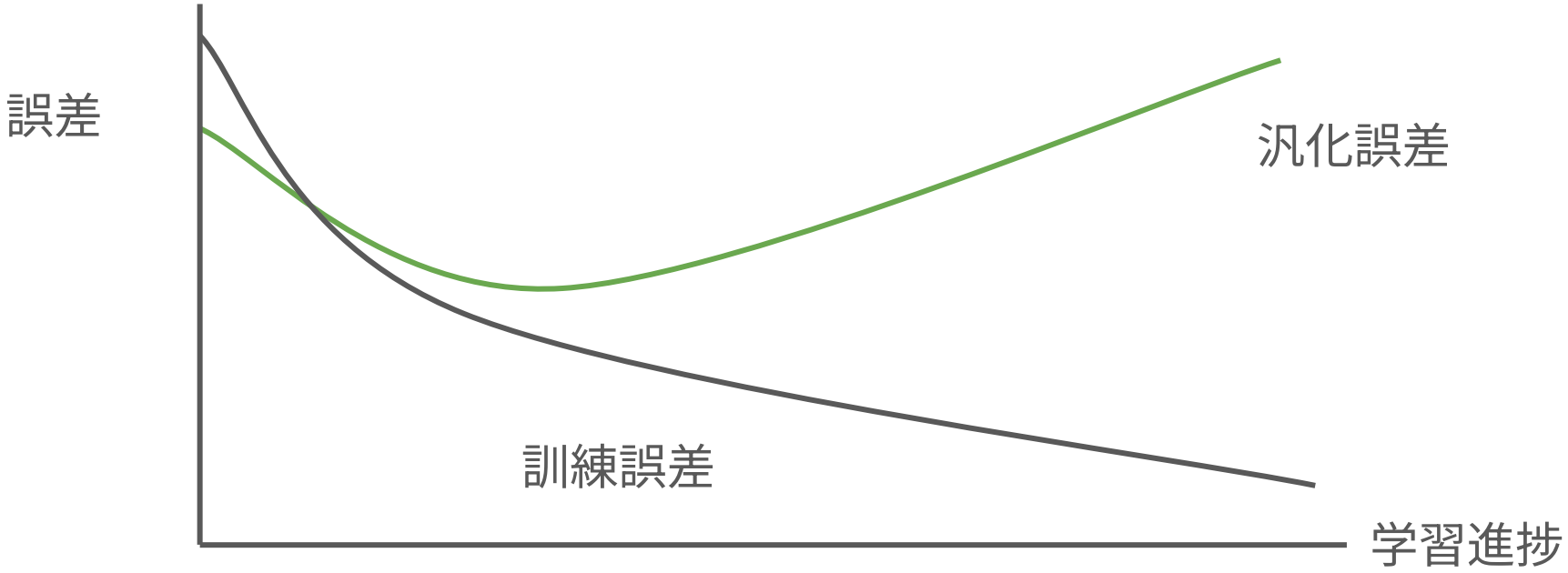
→**汎化誤差**が増加し始めたら学習をやめていく

訓練誤差：訓練データに対する予測と正解の誤差

汎化誤差：未知データに対する予測と正解の誤差

過学習に対するテクニック

- 早期終了



過学習に対するテクニック

- **二重降下現象**

  一定期間は汎化誤差が増加（モデル精度の悪化）するが

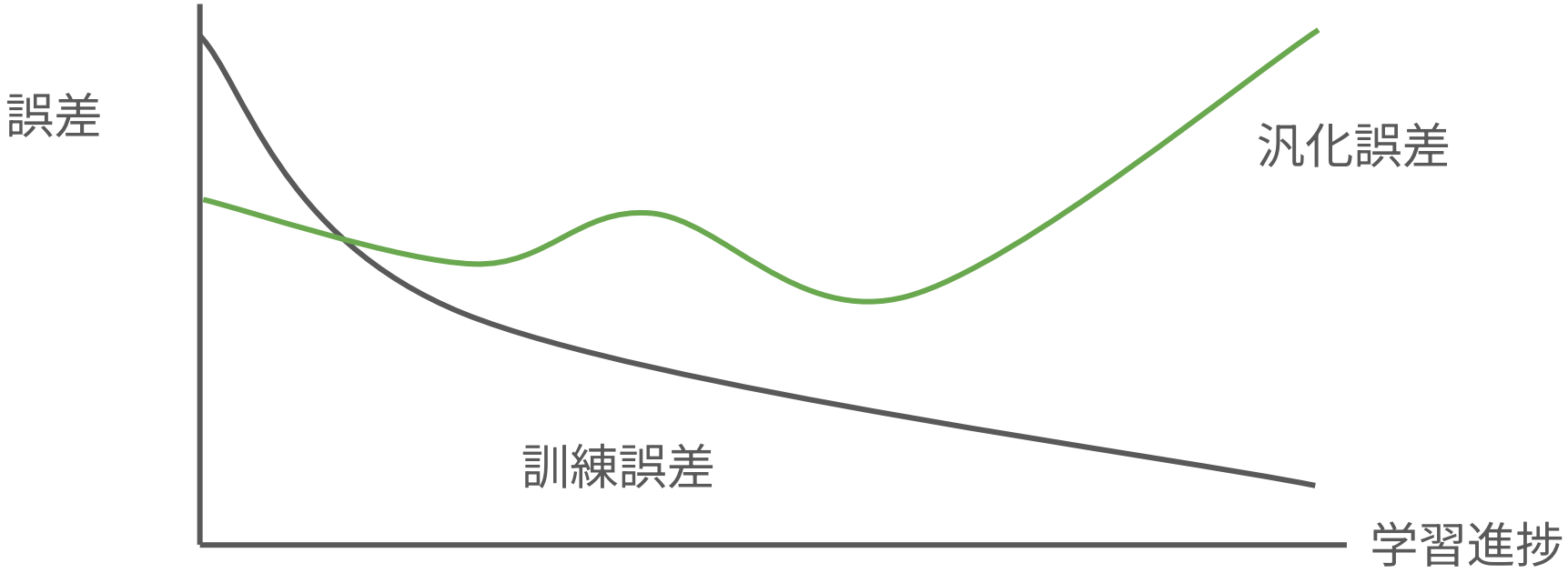
  その後汎化誤差が減少（モデル精度の向上）していく現象

→どのタイミングで学習をやめるのかは難しい問題

→汎化誤差が最小になる学習タイミングは分からないため

過学習に対するテクニック

- 早期終了



過学習に対するテクニック

- **ノーフリーランチ定理**

  あらゆる問題を効率的に解く汎用的な方法はないということ

→**ジェフリー・ヒントン**は

早期終了を「**Beautiful FREE LUNCH**」と表現している

→**早期終了**は過学習を抑えるテクニックとしては優秀な手法

活性化関数

作成者：辻　大貴

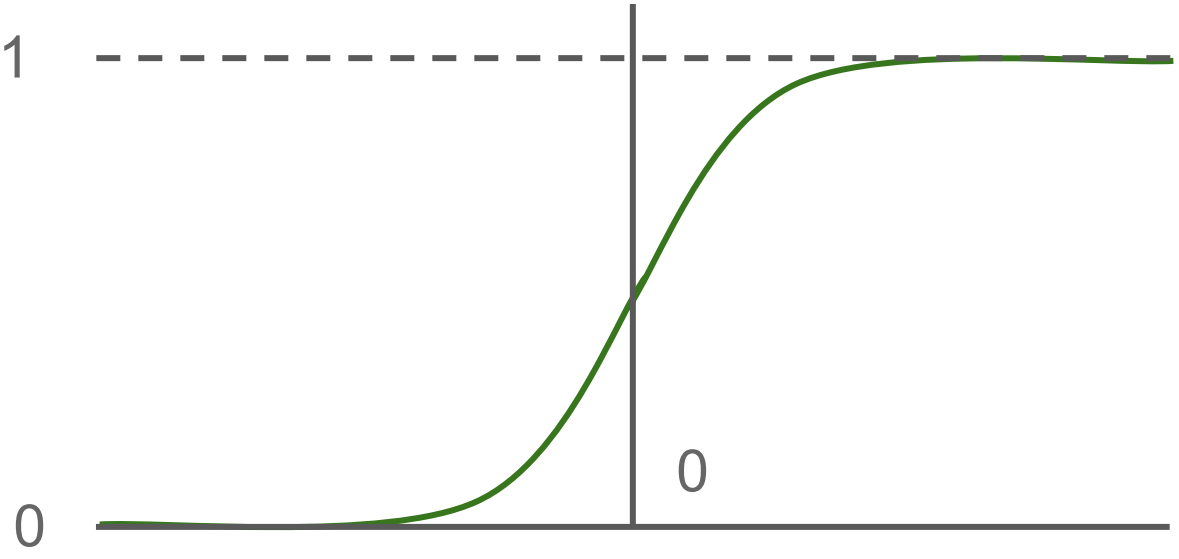
活性化関数

- **代表的な活性化関数**
  - シグモイド関数
  - ソフトマックス関数
  - 恒等関数
  - tanh関数
  - ReLU関数
  - Leaky ReLU関数　　など

活性化関数

- **シグモイド関数**

  2値分類で活用される（迷惑メールかどうか　など）

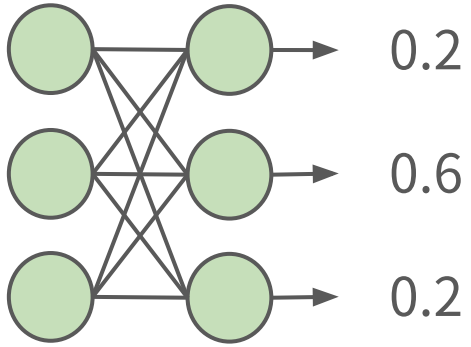


活性化関数

- **ソフトマックス関数**

  多クラス分類のときに使われる関数

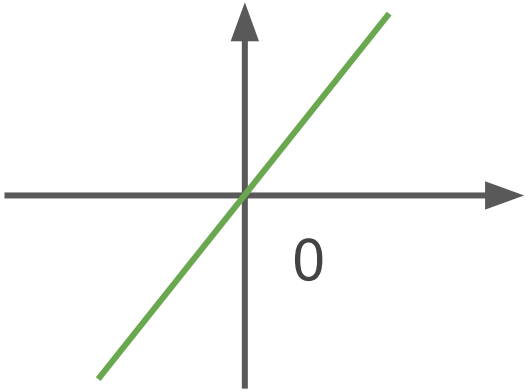
→モデルの出力の総和は1になる



- **恒等関数**

  回帰問題のときに使われる関数

→入力した値と同じ値を返す



活性化関数

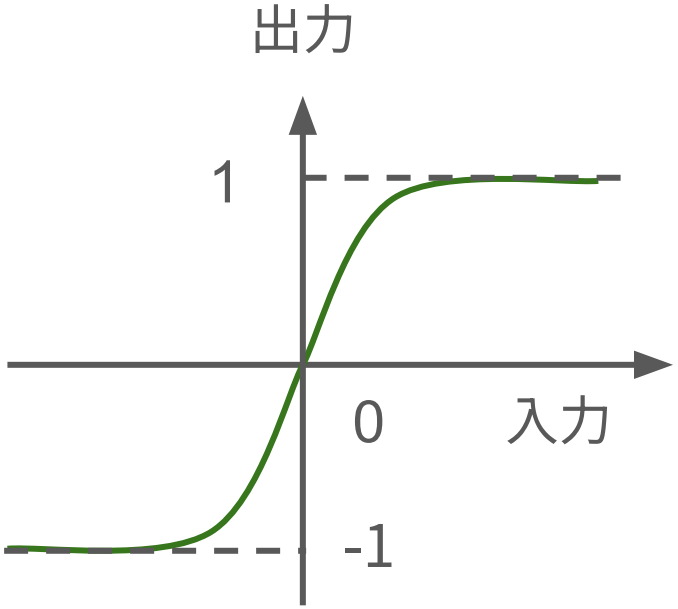
- **tanh関数（ハイパボリックタンジェント関数）**

  -1から1の値を出力する関数

→隠れ層の関数をシグモイド関数から

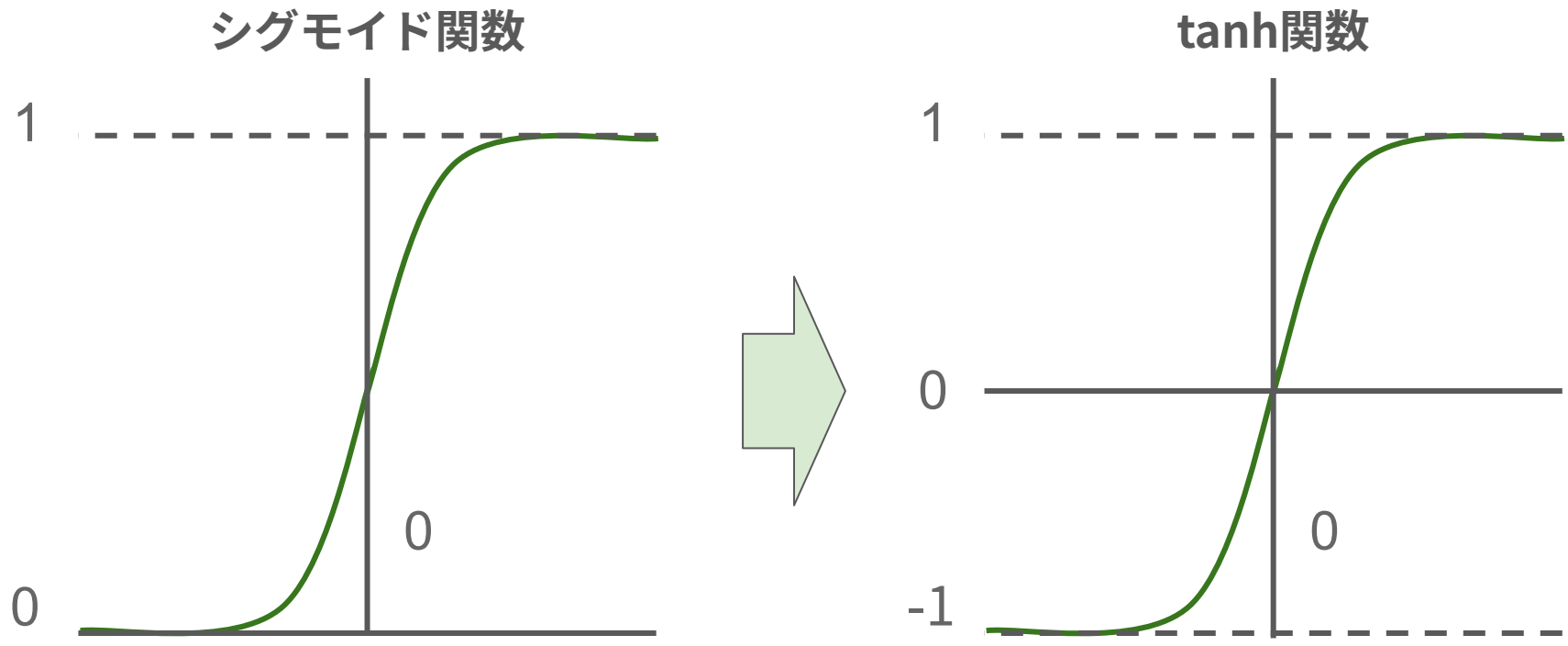
tanh関数に変更することで、

勾配消失問題を緩和することができる



→シグモイド関数を**線形変換**した関数

活性化関数



活性化関数

- **tanh関数（ハイパボリックタンジェント関数）**

誤差逆伝播法では活性化関数の微分を活用する

→シグモイド関数の微分した値の範囲は**0〜0.25**と小さいため

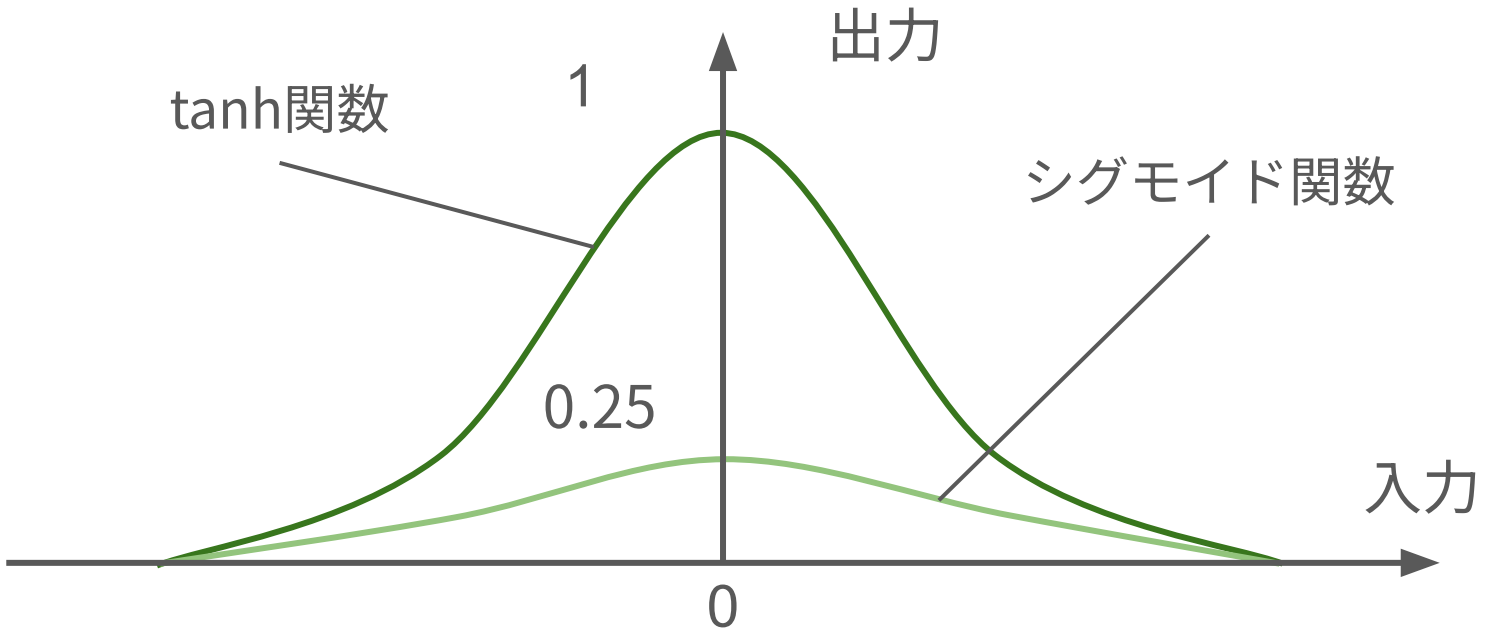
層が深くなってしまうと学習が進まなくなってしまう

→微分した値の最大値は1であるため**勾配が消失しにくい**

勾配消失問題を緩和することができる

活性化関数

- tanh関数（ハイパボリックタンジェント関数）

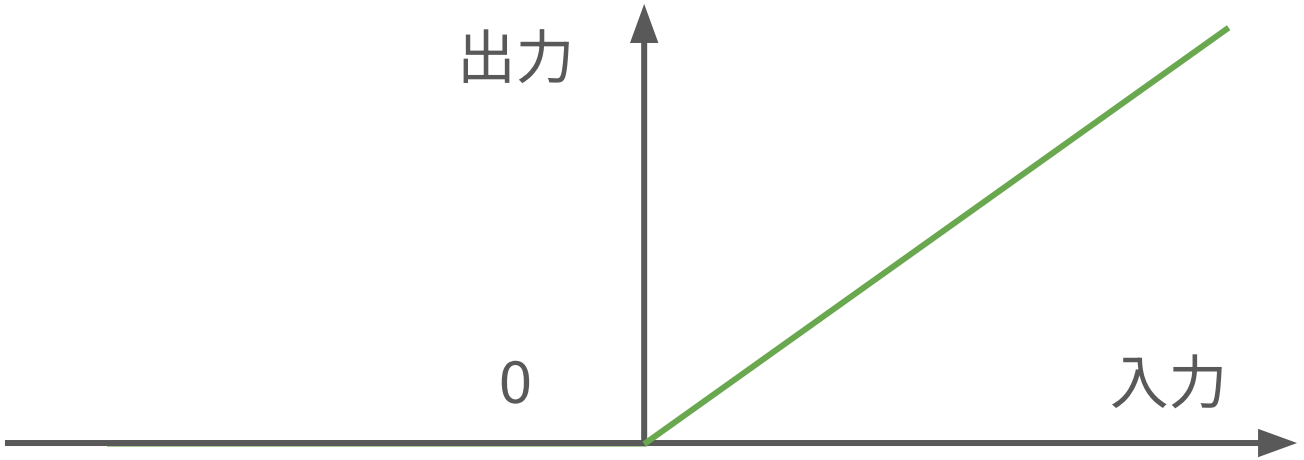


活性化関数

- **ReLU関数（Rectified Linear Unit関数）**

  入力値が0以下のときは0、入力値が0を超えるときは

  入力値をそのまま出力する関数

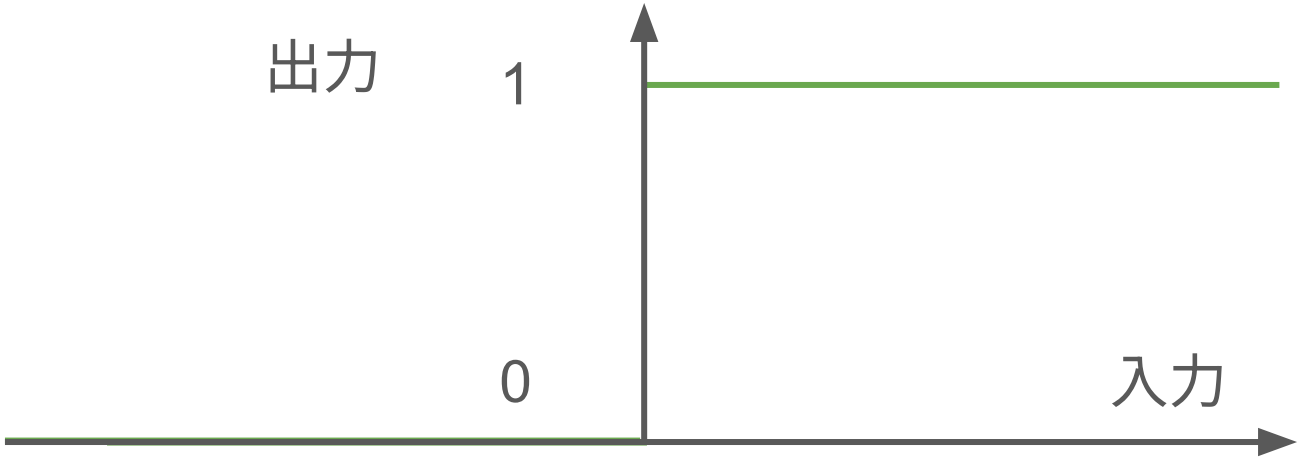


活性化関数

- **ReLU関数（Rectified Linear Unit関数）**

  微分した値は、入力が0よりも小さい場合は0

  入力が0以上の場合は1になる



活性化関数

- **ReLU関数（Rectified Linear Unit関数）**

  活性化関数に**ReLU関数**を使用することで

  tanh関数よりも勾配消失が起きにくくなる

→入力値が0以上なら微分した値は1なので勾配消失が起きにくい

→入力値が0未満なら微分した値は0になってしまうので、

データによってはReLU関数と相性が合わないこともある

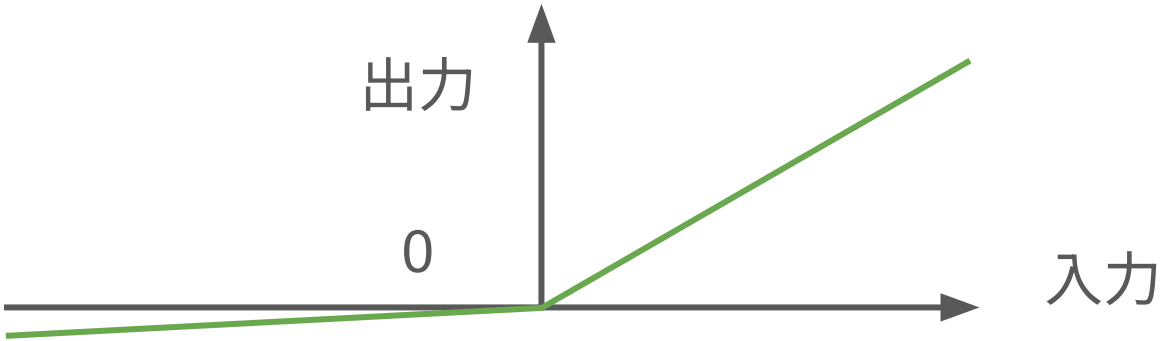
活性化関数

- **Leaky ReLU関数**

  ReLU関数を改良した関数の1つ

→入力値が0より小さい場合、入力値をα倍した値を出力、

  0以上の場合には入力値と同じ値を出力する関数



活性化関数

- **Leaky ReLU関数**

  微分した値が0にならないため勾配消失が起きにくい

→他にReLU関数から派生した関数も存在する

  どの関数を使用するかはケースバイケースである

  - Parametric ReLU関数　：αは学習によって決定
  - Randomized ReLU関数：αは範囲内の値からランダムに選択

活性化関数

- **活性化関数について**

　使用する活性化関数によってモデルの精度は大きく変わる

→目的にあった活性化関数を選択することが大切

→同一モデル内で異なる活性化関数が使用されることもある

データの正規化と重みの初期値

作成者：辻　大貴

データの正規化と重みの初期値

- **重みの初期値と正規化**

  モデルの精度を高めていくために、データを加工したり、

  重みの初期値を工夫したりする方法がある

→データの偏りを減らすことでモデルの精度を高める効果

→重みなどのパラメータをランダムな値にするよりも

  効率的に学習ができるような重みを設定する方が良い

データの正規化と重みの初期値

- **正規化**

効率的に学習が行えるようにデータを調整すること

→各データの最大値や最小値が大きく異なってしまうと

パラメータに偏りが生じてしまい学習効率が落ちてしまう

→データの**スケール（範囲）**を調整していく

特徴量の範囲を処理することを**スケーリング**という

データの正規化と重みの初期値

- **スケールの調整方法（年齢と給与）**

  それぞれの特徴量を最大値で割って範囲を 0〜1へ変換

  - 年齢：最大値が80、データAの年齢が40の場合
    「40 ÷ 80」から0.5に変換
  - 給与：最大値が20,000,000、データAの給与が5,000,000の場合、「5,000,000 ÷ 20,000,000」から0.25に変換

データの正規化と重みの初期値

- **標準化**

  各特徴量の平均を０、分散が１になるように変換すること

→データのスケールを合わせるよりも効果が高い

- **白色化**

  各特徴量を**無相関化**し、**標準化**すること

→計算コストが大きくなるので**標準化**を使うケースが多い

データの正規化と重みの初期値

- **データの偏りと対処法**
  - **オーバーサンプリング（アップサンプリング）**

    データが少ないカテゴリに対して水増しすること

  →オーバーサンプリングの代表的な手法にSMOTEがある

  - **アンダーサンプリング（ダウンサンプリング）**

    データが多いカテゴリのデータを減らすこと

データの正規化と重みの初期値

データを正規化や標準化しても重みに偏りがあると

層を伝播していくうちに**データの分布**が崩れてしまう

→**勾配消失問題**などが発生しやすくなってしまう

データの分布に影響が少ない重みの設定が大切

→シグモイド関数やtanh関数を用いる場合は**Xavierの初期値**

ReLU関数を用いる場合は**Heの初期値**が良いとされる

データの正規化と重みの初期値

- **Xavierの初期値**

  平均 0、標準偏差 $\frac{1}{\sqrt{n}}$ の正規分布から生成された初期値のこと

  （ n：前の層のユニット数）

- **Heの初期値**

  平均 0、標準偏差 $\sqrt{\frac{2}{n}}$ の正規分布から生成された初期値のこと

  （ n：前の層のユニット数）

データの正規化と重みの初期値

- **内部共変量シフト**

  重みの初期値などを工夫しても何層も伝播していくうちに

  データの分布に偏りが生じてしまうこと

→層に伝わるデータを**正規化**することで

  データの分布に偏りが生じるのを防ごうとする

→隠れ層に入力する値を正規化する手法が取られる