受講にあたって

- GoogleMeetについて
  - カメラとマイクの使用を**許可**する
  - 上記許可後、**カメラ**と**マイク**は**オフ**にしてください
- 質問について
  - チャット機能を使って、質問がある旨を発言してください
  - 質問確認後、講師がお名前をお呼びしますので、マイクをオンにして質問してください
  - 質問終了後、再度マイクをオフにしてください
- 講義時間について
  - 講義時間　10：00～17：00
  - お昼休憩　12：00～13：00（予定）
  - 約１時間おきに10分休憩をとる予定です

【AI開発道場】
深層学習

目次

- 深層学習とは
- 深層学習の仕組み
- 深層学習モデルの学習テクニック
- 代表的なDNN
- まとめ

目次

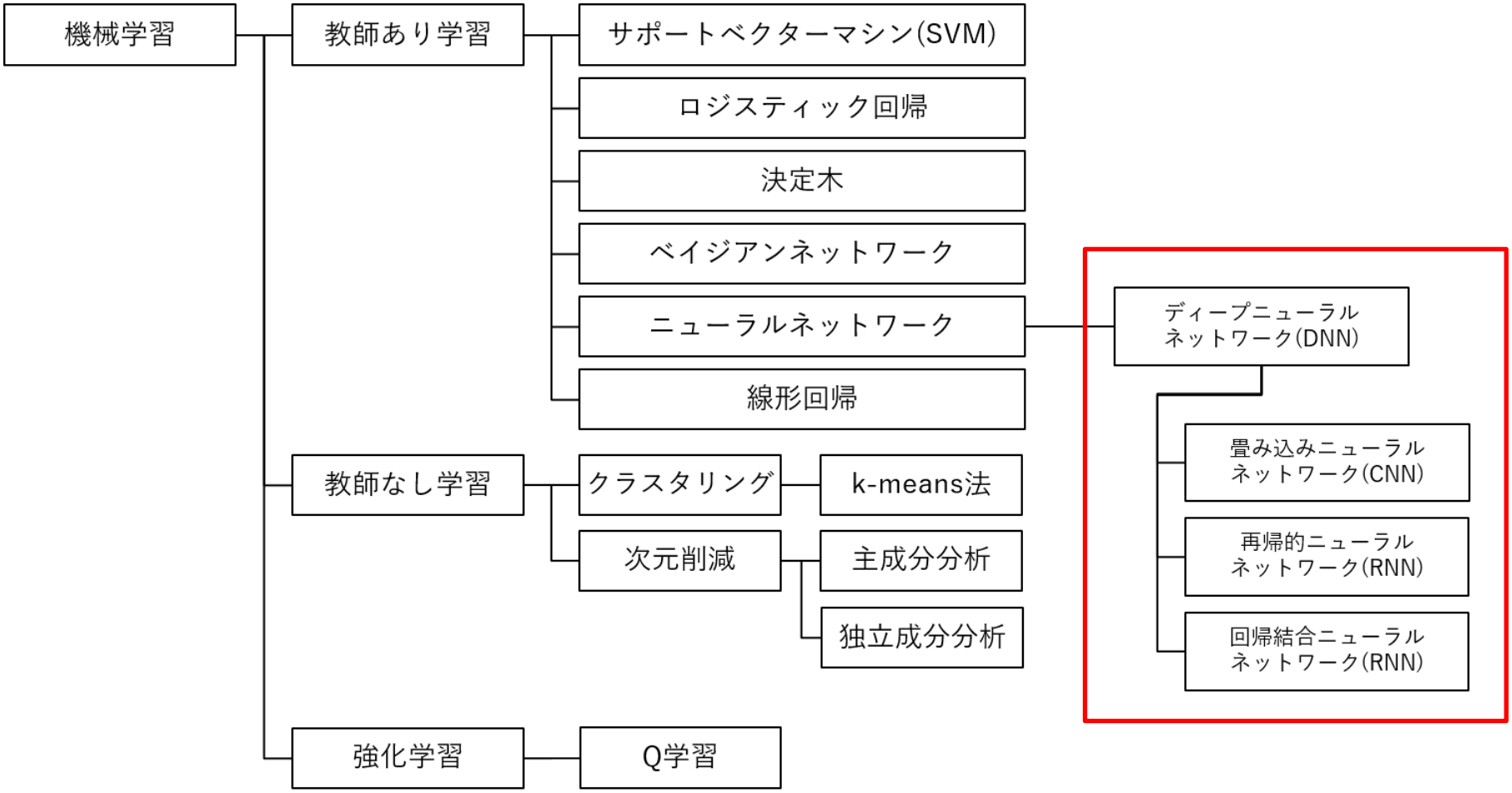
- 深層学習とは
- 深層学習の仕組み
- 深層学習モデルの学習テクニック
- 代表的なDNN
- まとめ

深層学習とは（1/2）

- 深層学習とは
  - 2012年、トロント大学のヒントン教授が発表
  - 世界的な画像認識競技会で深層学習を用いたシステムを使用し圧勝
  - その後、研究が盛んになり、次々にニュースになる
    - Googleの猫
    - AlphaGoが世界トップレベルのプロ棋士に勝利
  - 主な適用範囲
    - 画像認識
    - 音声認識
    - 自然言語処理
    - 時系列データ処理

深層学習とは（2/2）

- 深層学習の位置づけ

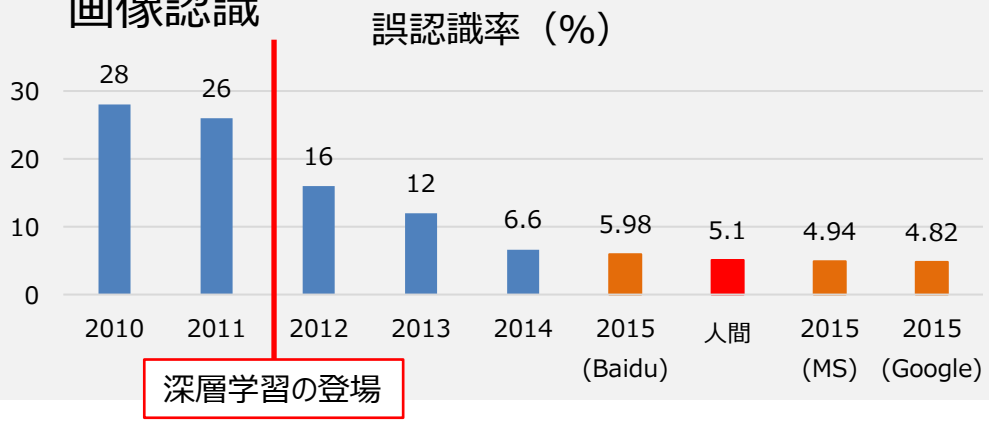


技術的背景＜適用領域の広がり（1/2）＞

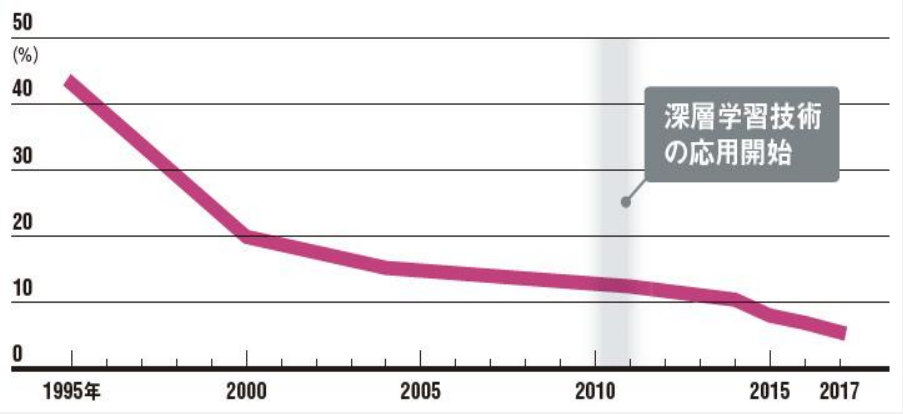
- 進化し続ける深層学習

技術 | 顧客 | DCS

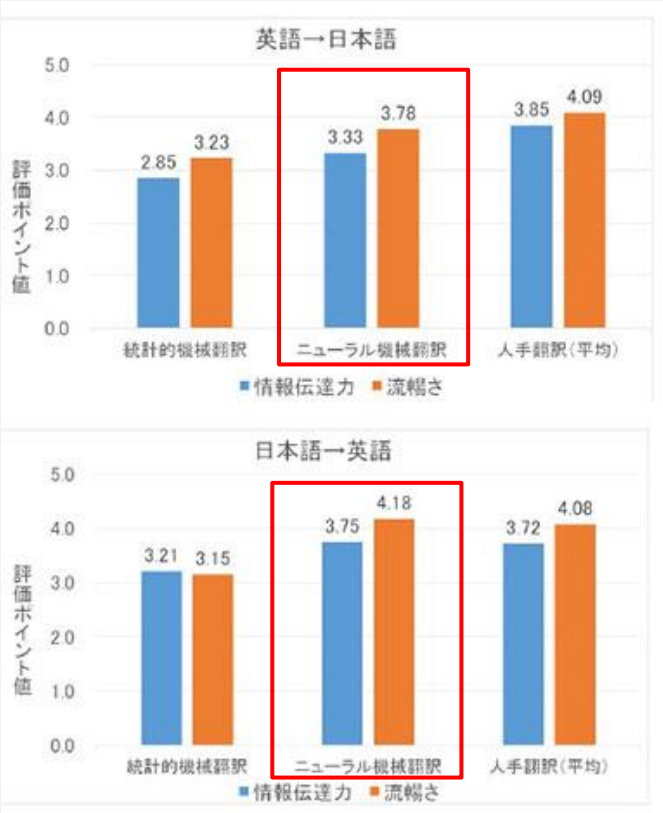
画像認識



音声認識



機械翻訳

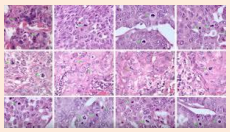


技術的背景＜適用領域の広がり（2/2）＞

- 応用領域も広がってきている

技術 | 顧客 | DCS

画像分野



がん細胞の診断

スマートグラス

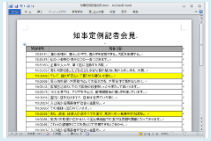
Facebookの
写真タグ付け

Tegaki

手書き文字認識

音声分野

スマートスピーカー



インタビューなどの
文字書き起こし

Google翻訳の
音声入力

IBM Watson

IBM Watson

Paid

AI自動与信審査

TESLA

テスラモーターズの
自動運転車

voicera

議事録の自動作成

How To Get Free Unlimited articoolo Articles

自動記事執筆

時系列データ分野

自然言語分野

目次

- 深層学習とは
- 深層学習の仕組み
- 深層学習モデルの学習テクニック
- 代表的なDNN
- まとめ

全体像

【事前に用意する情報】

入力： $\mathbf{x}_n = [x_{n1} .. x_{ni}]$

訓練データ： $\mathbf{d}_n = [d_{n1} .. d_{nk}]$

【多層ネットワークのパラメータ】

$\mathbf{w}^{(l)}$ { 重み： $\mathbf{W}^{(l)} = \begin{bmatrix} w_{11}^{(l)} & \cdots & w_{1I}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{J1}^{(l)} & \cdots & w_{JI}^{(l)} \end{bmatrix}$

バイアス： $\mathbf{b}^{(l)} = \left[b_1^{(l)} \ldots b_J^{(l)}\right]$

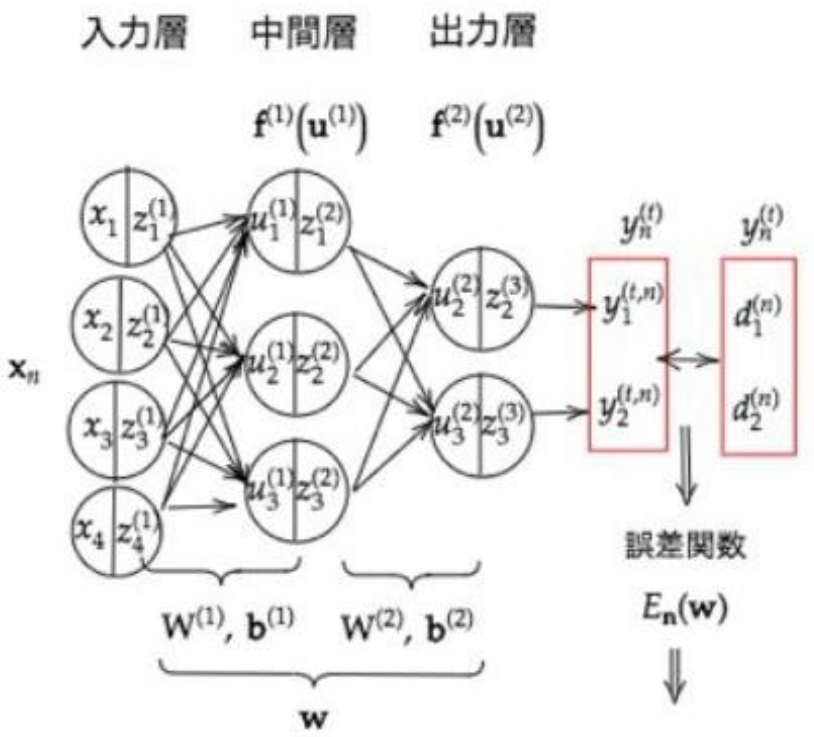
活性化関数： $\mathbf{f}^{(l)}(\mathbf{u}^{(l)}) = \left[f^{(l)}(u_1^{(l)}) .. f^{(l)}(u_J^{(l)})\right]$

中間層出力： $\mathbf{z}^{(l)} = \left[z_1^{(l)} .. z_K^{(l)}\right] = \mathbf{f}^{(l)}(\mathbf{u}^{(l)})$

総入力： $\mathbf{u}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}$

出力： $\mathbf{y}_n^{(t)} = \left[y_{n1}^{(t)} .. y_{nK}^{(t)}\right] = \mathbf{z}^{(L)}$

誤差関数： $E_n(\mathbf{w})$



$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla \mathbf{E}_n(\mathbf{w})$ ⇐ $\nabla E_n(\mathbf{w}) = \dfrac{\partial E}{\partial \mathbf{w}}$

入力層ノードのインデックス： $i\,(=1 \ldots I)$

中間層ノードのインデックス： $j\,(=1 \ldots J)$

出力層ノードのインデックス： $k\,(=1 \ldots K)$

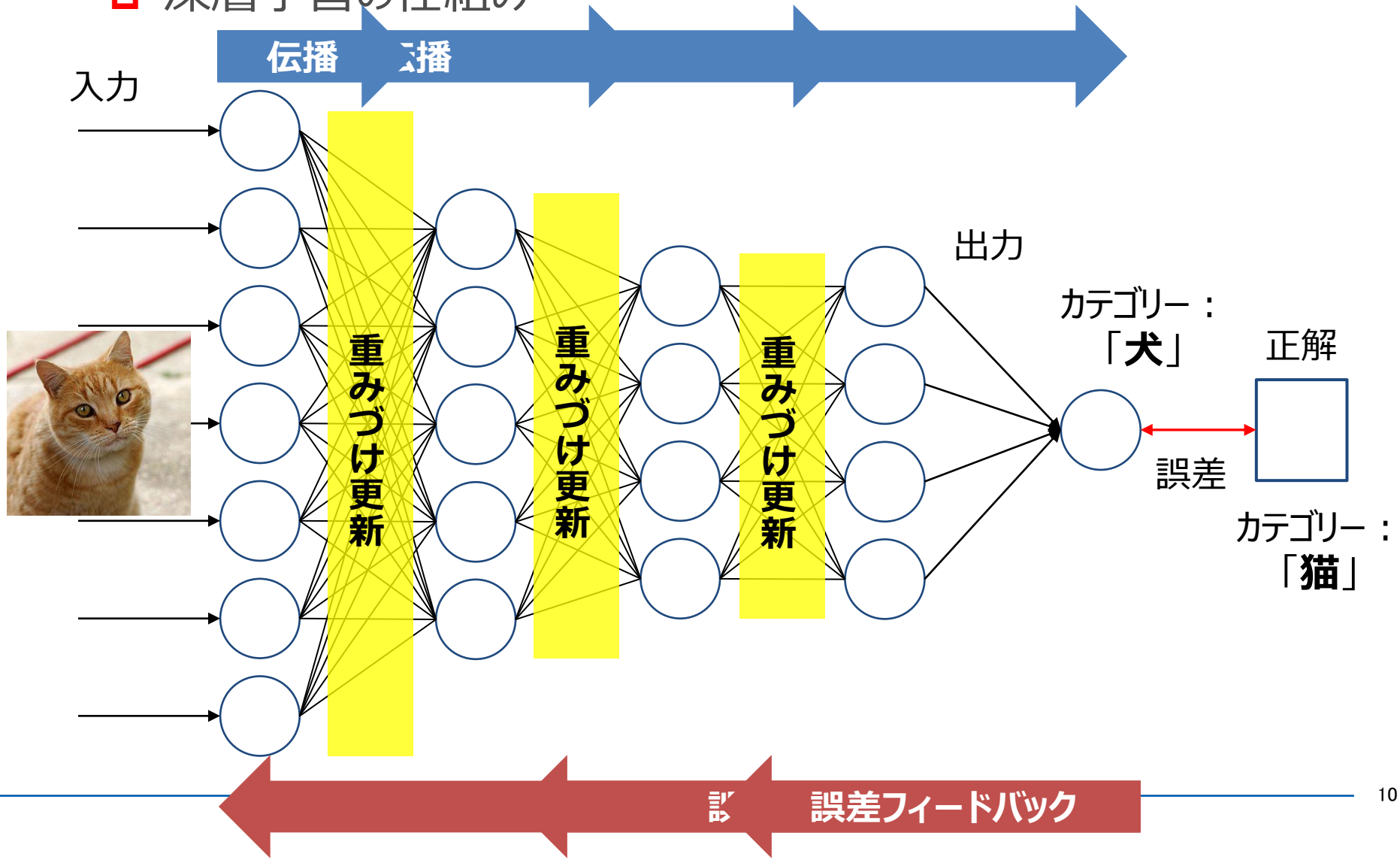
層のインデックス： $l\,(=1 \ldots L)$

訓練データのインデックス： $n\,(=1 \ldots N)$

試行回数のインデックス： $t\,(=1 \ldots T)$

学習の仕組み

- 深層学習の仕組み



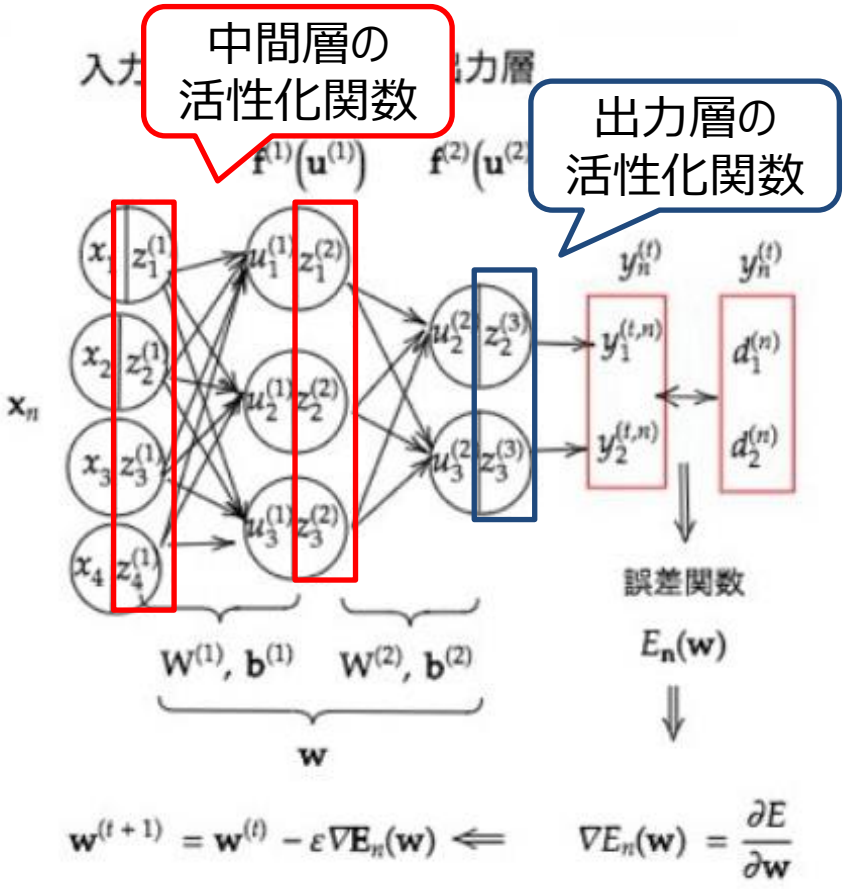
活性化関数

- 活性化関数とは
  - NNにおいて、次の層への出力の大きさを決める非線形の関数
  - 入力の値によって、次の層への信号のON/OFFや強弱を定める働きを持つ

- 出力層と中間層
  - 値の強弱
    - 中間層：閾値の前後で信号の強弱を調整
    - 出力層：信号の大きさ（比率）はそのままに変換
  - 確率出力
    - 分類問題の場合、出力層の出力値域は0～1、総和1

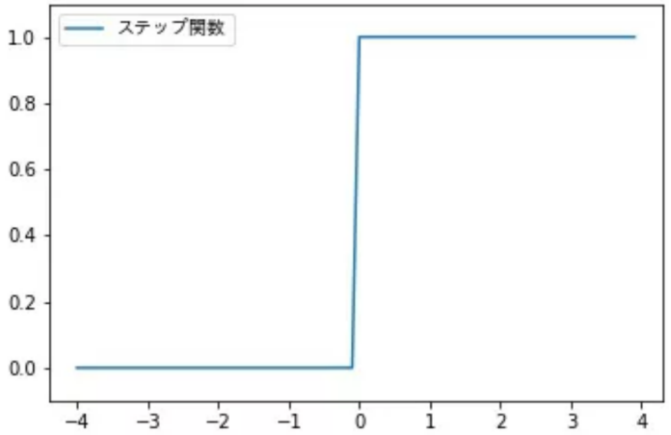
活性化関数

- 活性化関数の種類
  - 中間層
    - ReLU関数
    - シグモイド関数
    - ステップ関数
  - 出力層
    - ソフトマックス関数
    - 恒等写像
    - シグモイド関数



活性化関数

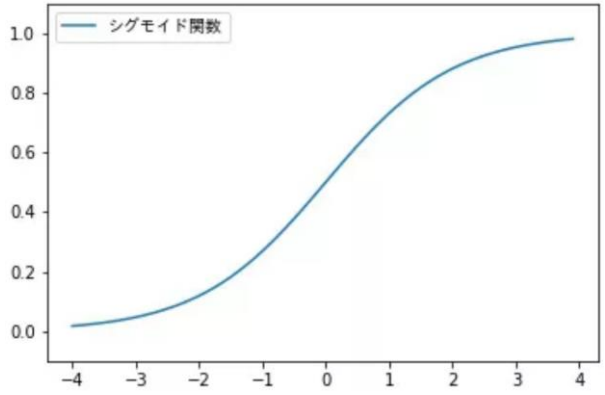
- ステップ関数
  - 図
  - 数式
    - $f(x) = \begin{cases} 1\ (x \geq 0) \\ 0\ (x < 0) \end{cases}$



  - 閾値を超えたら発火する関数
  - 出力は常に0,1
  - パーセプトロンで用いられた
  - 課題
    - 0-1間の間を表現できず、線形分離可能なものしか学習できない

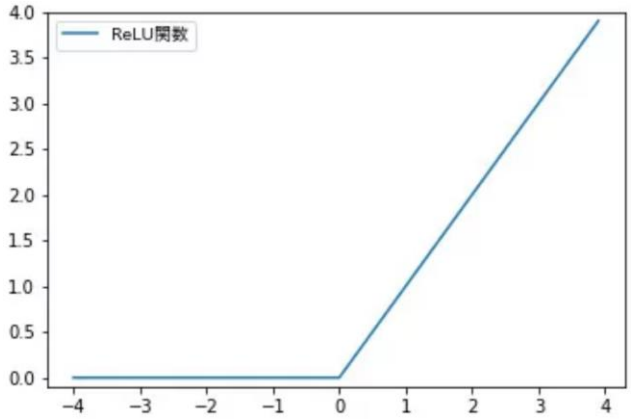
活性化関数

- シグモイド関数
  - 図
  - 数式
    - $f(u) = \frac{1}{1+e^{-u}}$
  - 0~1を緩やかに変化する関数
  - 信号の強弱を伝えられるようになり、NN普及のきっかけとなった
  - 出力層では、一般的に2値分類問題で利用
  - 課題
    - 大きな値では出力の変化が微小なため、勾配消失問題を引き起こすことがあった



活性化関数

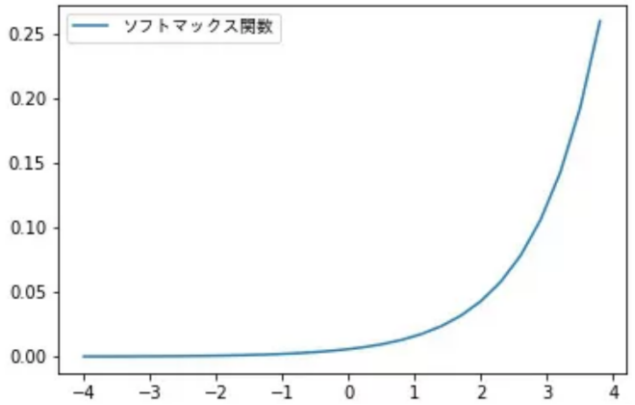
- ReLU関数
  - 図
  - 数式
    - $f(x) = \begin{cases} x\ (x\ > 0) \\ 0\ (x\ \leq 0) \end{cases}$



  - 今もっとも使われている活性化関数
  - 勾配消失問題の回避と、スパース化に貢献

活性化関数

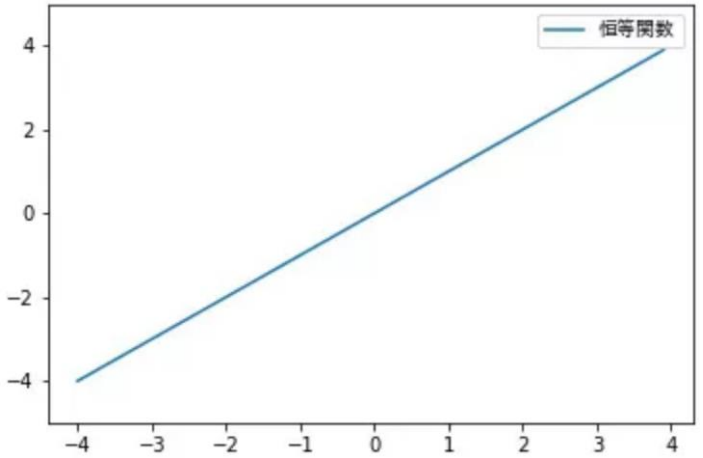
- ソフトマックス関数
  - 図
  - 数式
    - $f(\boldsymbol{i}, \boldsymbol{u}) = \frac{e^{u\_i}}{\sum_{k=1}^{K} e^{u_k}}$



  - 出力が(0, 1)かつ全体の総和が1となるように変換する
  - K=2の時がシグモイド関数
  - 一般的に多値分類で利用

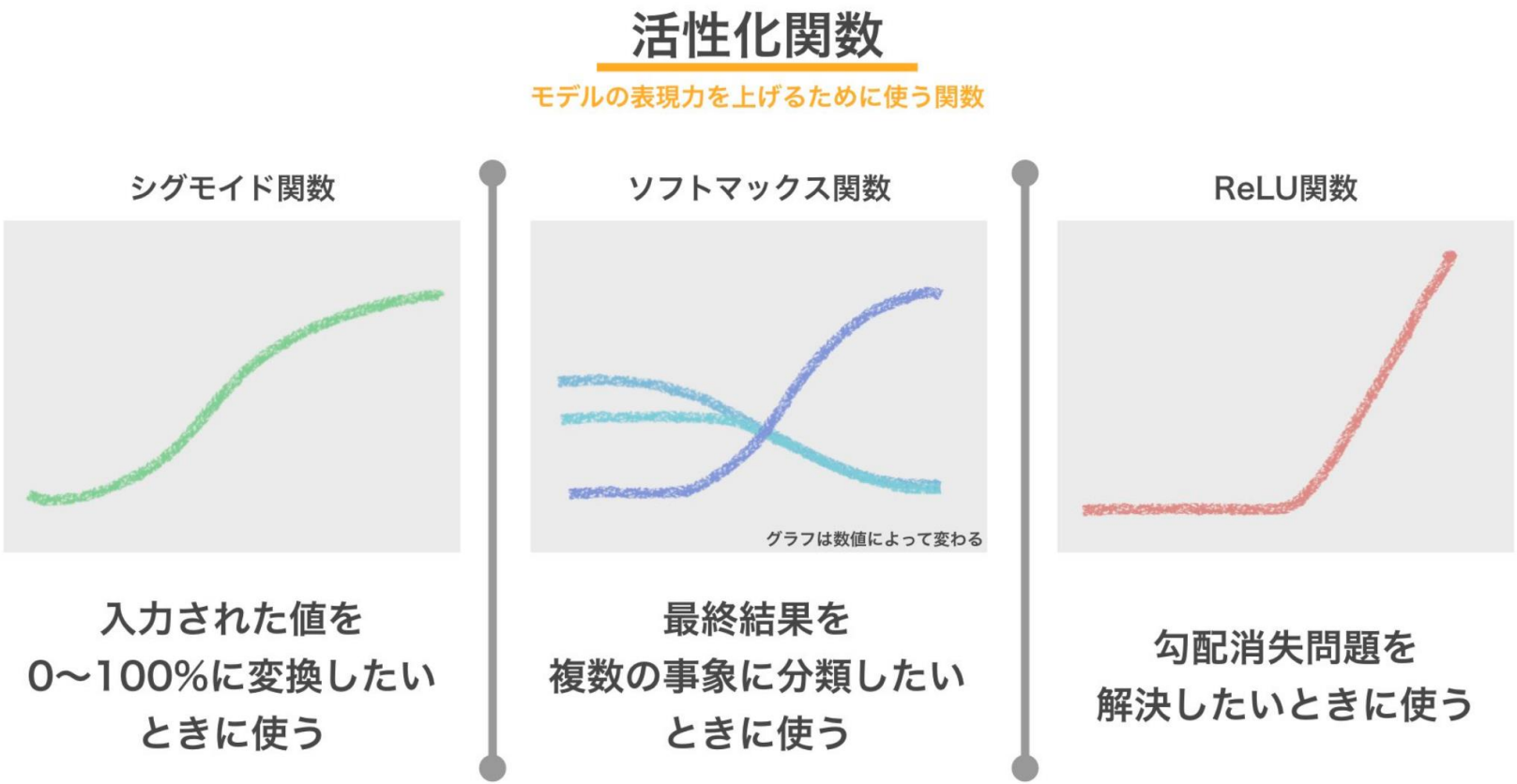
活性化関数

- 恒等写像
  - 図
  - 数式
    - $f(x) = x$
  - 入力と出力が全く同じ関数
  - 一般的に回帰問題で使用



活性化関数

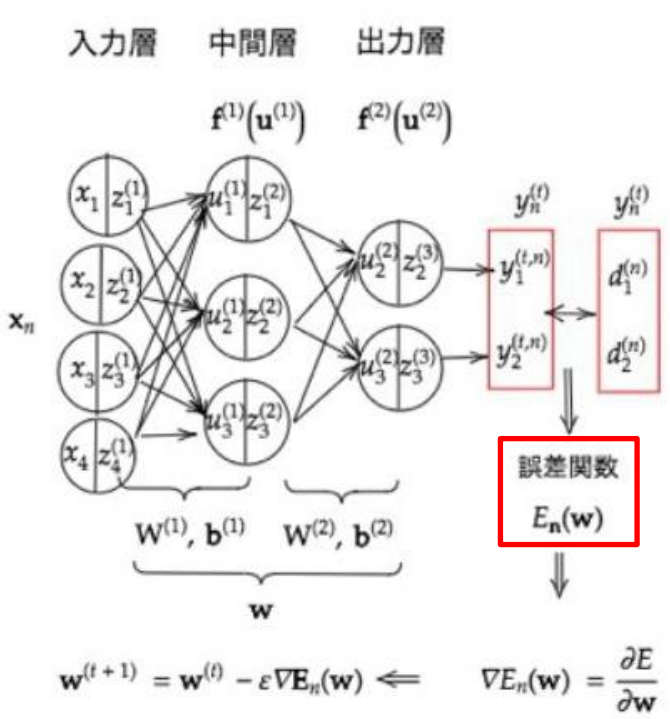
- 活性化関数比較



損失関数

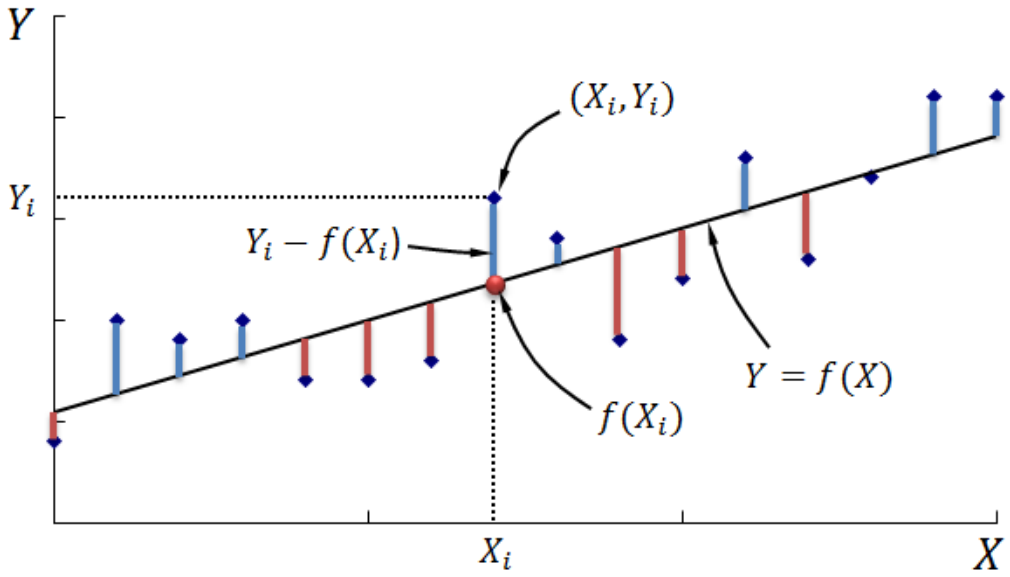
- 損失関数とは
  - 予測データと正解データの差を評価する関数
  - どれくらい正解から離れているかの尺度を決める

- 損失関数の種類
  - 平均二乗誤差
  - クロスエントロピー誤差



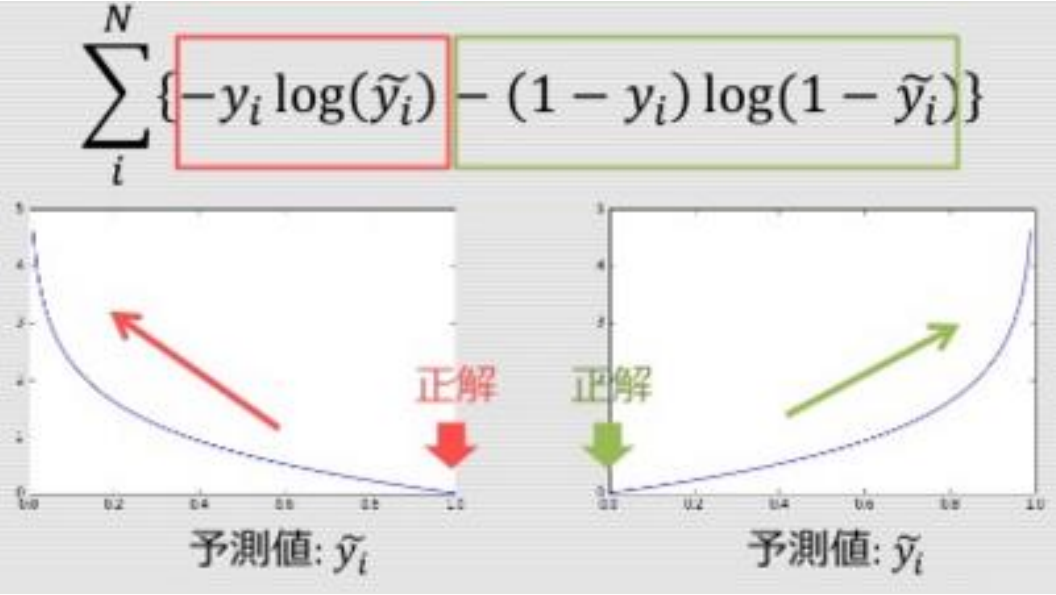
損失関数

- 平均二乗誤差
  - 数式
    - $E = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - 一般的に回帰問題で使用



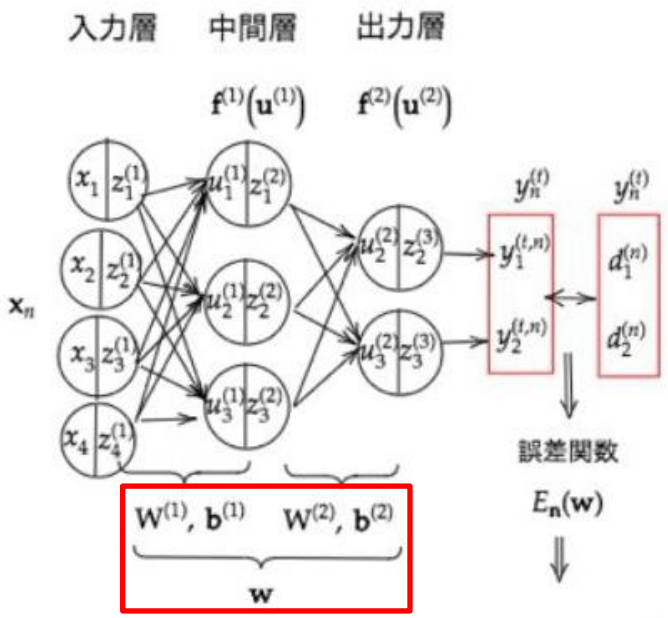
損失関数

- クロスエントロピー誤差
  - 数式
    - $E = -\sum_k t_k \log y_k$
  - 一般的に分類問題で使用
  - 2つの確率分布(正解、不正解)がどれくらい「離れているか」
  - 二乗誤差と比較して誤差が大きいときの学習速度が早い



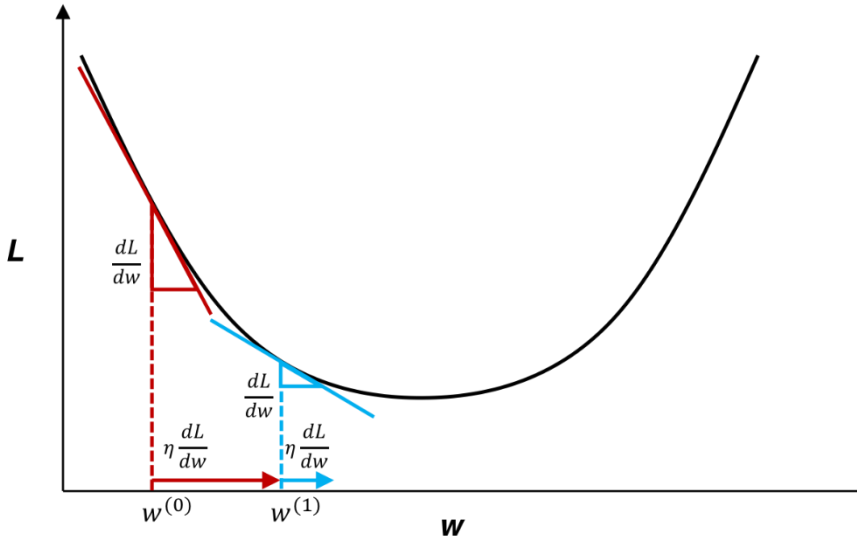
重み更新

- 深層学習の目的
  - 学習を通して誤差を最小にするネットワークを作成すること
    →誤差を最小化するパラメータを見つけること
  - 勾配降下法を利用してパラメータを最適化

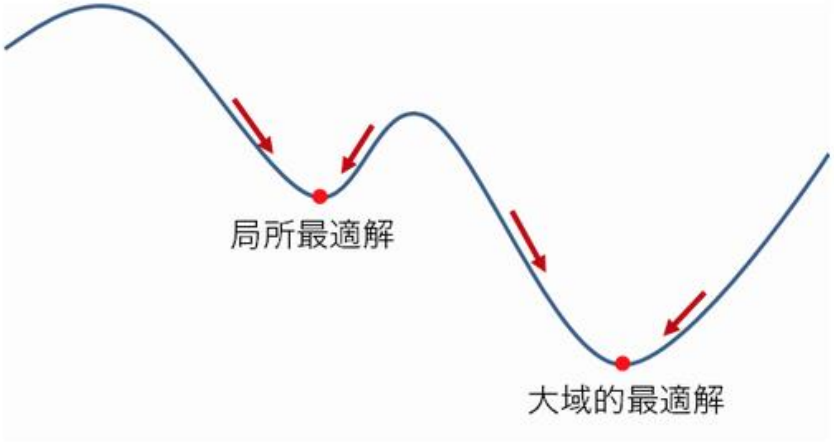


重み更新

- 勾配降下法
  - 数式
    - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \varepsilon \nabla \boldsymbol{E}$

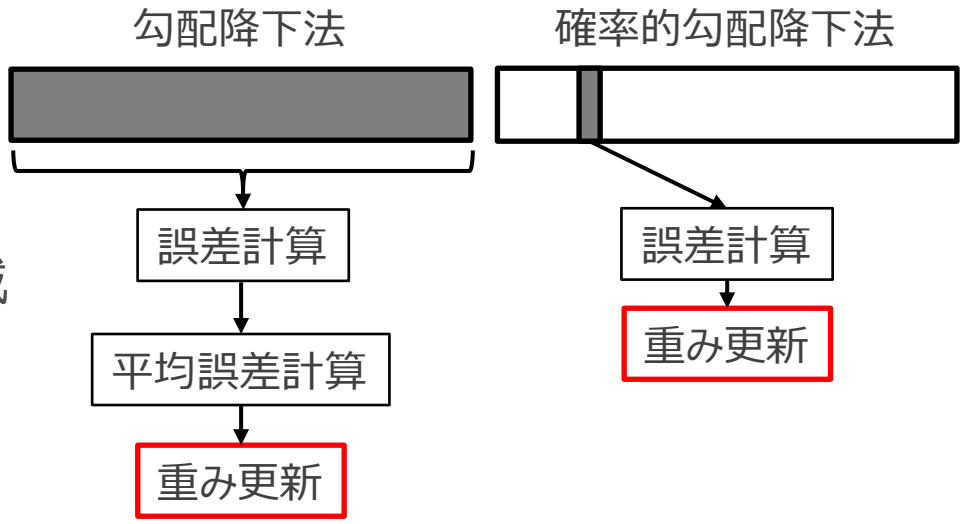


  - 学習率
    - 学習率が大きすぎる→発散
    - 学習率が小さすぎる→収束時間がかかる、局所解に陥りやすい
    - 学習率最適化で詳細説明



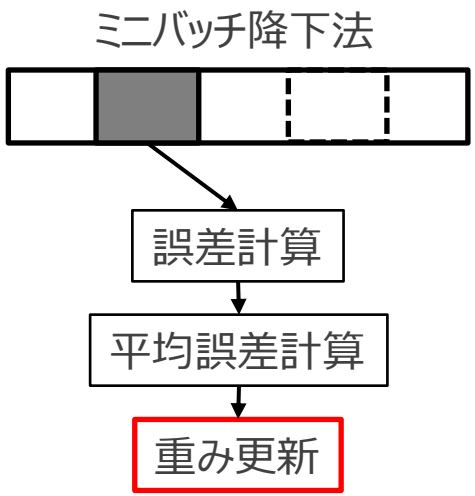
重み更新

- 確率的勾配降下法
  - ランダムに抽出したサンプルの誤差を用いて重み更新
    （勾配降下法は全サンプルの平均誤差）
  - 数式
    - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \varepsilon \nabla E_n$
  - メリット
    - 計算コストの軽減
    - 局所最適解のリスク軽減
    - オンライン学習ができる



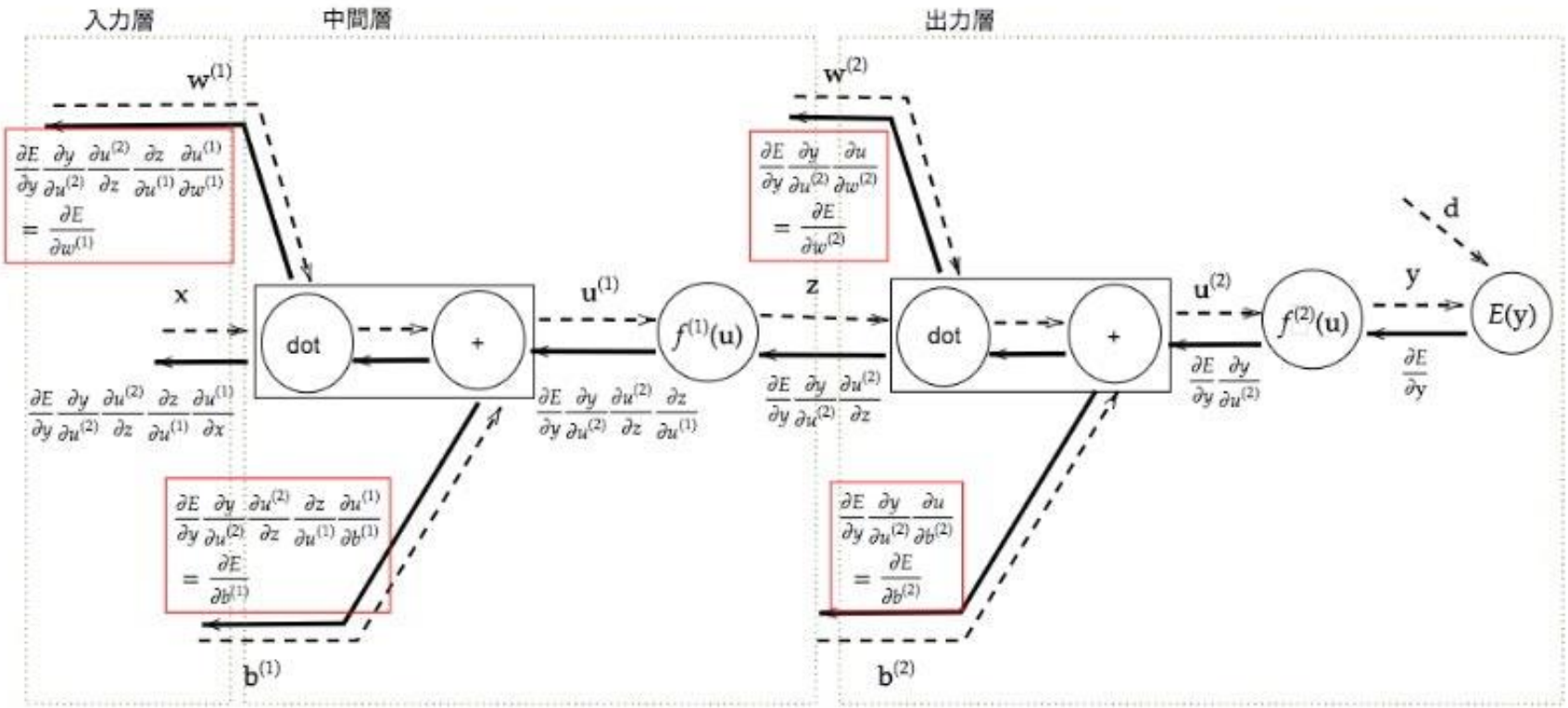
重み更新

- ミニバッチ降下法
  - ランダムに分割したデータの集合（ミニバッチ）に属するサンプルの平均誤差を用いて重み更新
  - 数式
    - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \varepsilon \nabla E_t$
    - $E_t = \frac{1}{N_t} \sum_{n \in D_t} E_n$
    - $N_t = |D_t|$
  - メリット
    - 確率的勾配降下法のメリットを損なわず、計算機の資源を有効活用可能
      →並列計算ができるので、CPUのスレッド並列化やGPU利用の恩恵を得られる

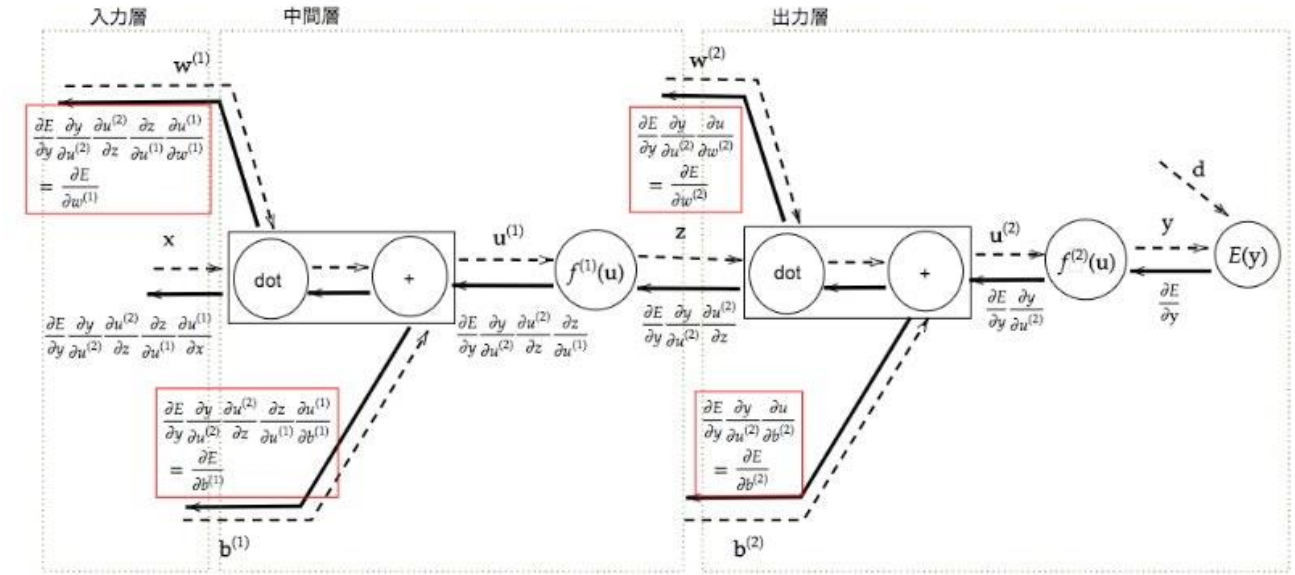


重み更新

- 誤差逆伝播法
  - 勾配計算（微分計算）を高速に計算する手法



重み更新



- メリット
  - 説明変数のデータ項目が多い場合に有効
- 注意事項
  - 隠れ層が深いと重みが変化しなくなり学習が進まなくなる
    - 勾配消失問題

全体像

【事前に用意する情報】

入力： $\mathbf{x}_n = [x_{n1} .. x_{ni}]$

訓練データ： $\mathbf{d}_n = [d_{n1} .. d_{nk}]$

【多層ネットワークのパラメータ】

$\mathbf{w}^{(l)}$ { 重み： $\mathbf{W}^{(l)} = \begin{bmatrix} w_{11}^{(l)} & \cdots & w_{1I}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{J1}^{(l)} & \cdots & w_{JI}^{(l)} \end{bmatrix}$

バイアス： $\mathbf{b}^{(l)} = [b_1^{(l)} \ldots b_J^{(l)}]$

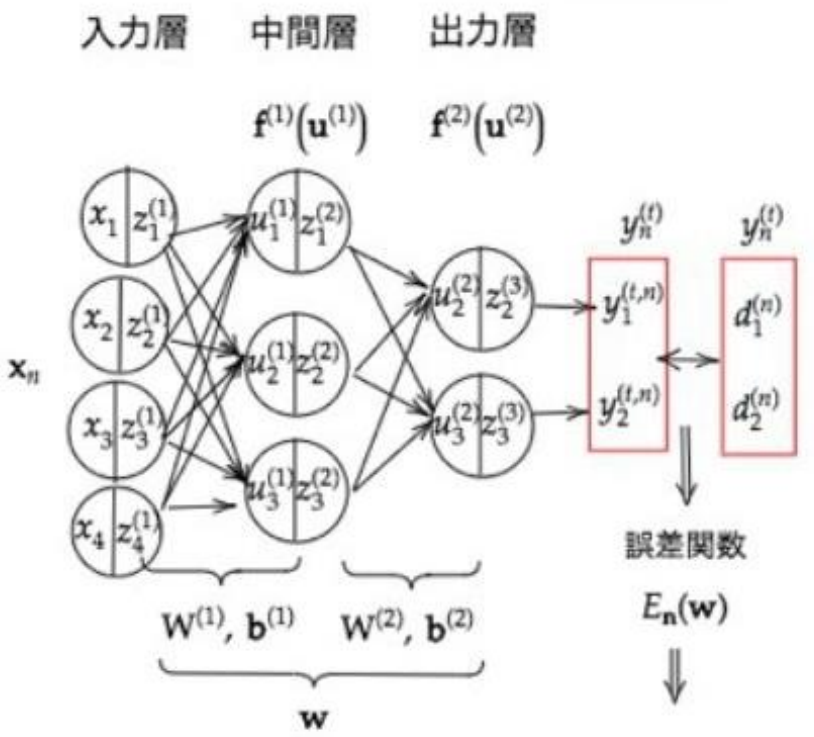
活性化関数： $\mathbf{f}^{(l)}(\mathbf{u}^{(l)}) = [f^{(l)}(u_1^{(l)}) .. f^{(l)}(u_J^{(l)})]$

中間層出力： $\mathbf{z}^{(l)} = [z_1^{(l)} .. z_K^{(l)}] = \mathbf{f}^{(l)}(\mathbf{u}^{(l)})$

総入力： $\mathbf{u}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}$

出力： $\mathbf{y}_n^{(t)} = [y_{n1}^{(t)} .. y_{nK}^{(t)}] = \mathbf{z}^{(L)}$

誤差関数： $E_n(\mathbf{w})$



$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \varepsilon \nabla \mathbf{E}_n(\mathbf{w})$ ⇐ $\nabla E_n(\mathbf{w}) = \dfrac{\partial E}{\partial \mathbf{w}}$

入力層ノードのインデックス： $i\,(=1 \ldots I)$

中間層ノードのインデックス： $j\,(=1 \ldots J)$

出力層ノードのインデックス： $k\,(=1 \ldots K)$

層のインデックス： $l\,(=1 \ldots L)$

訓練データのインデックス： $n\,(=1 \ldots N)$

試行回数のインデックス： $t\,(=1 \ldots T)$

目次

- 深層学習とは
- 深層学習の仕組み
- 深層学習モデルの学習テクニック
- 代表的なDNN
- まとめ

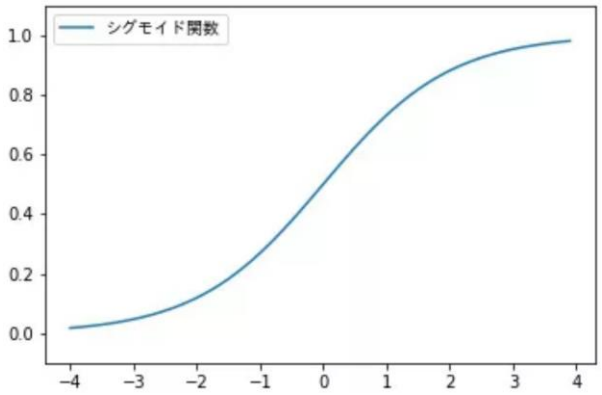
勾配消失問題

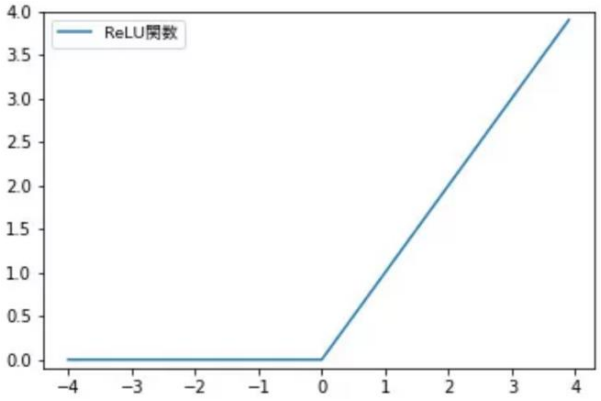
- 勾配消失問題とは
  - 誤差逆伝播法は階層に進むにつれて、勾配がどんどん緩やかになる
    →階層のパラメータはほとんど変わらず、学習が収束しなくなる

- 解決法
  - 活性化関数の選択
  - 重みの初期値設定
  - バッチ正規化

勾配消失問題

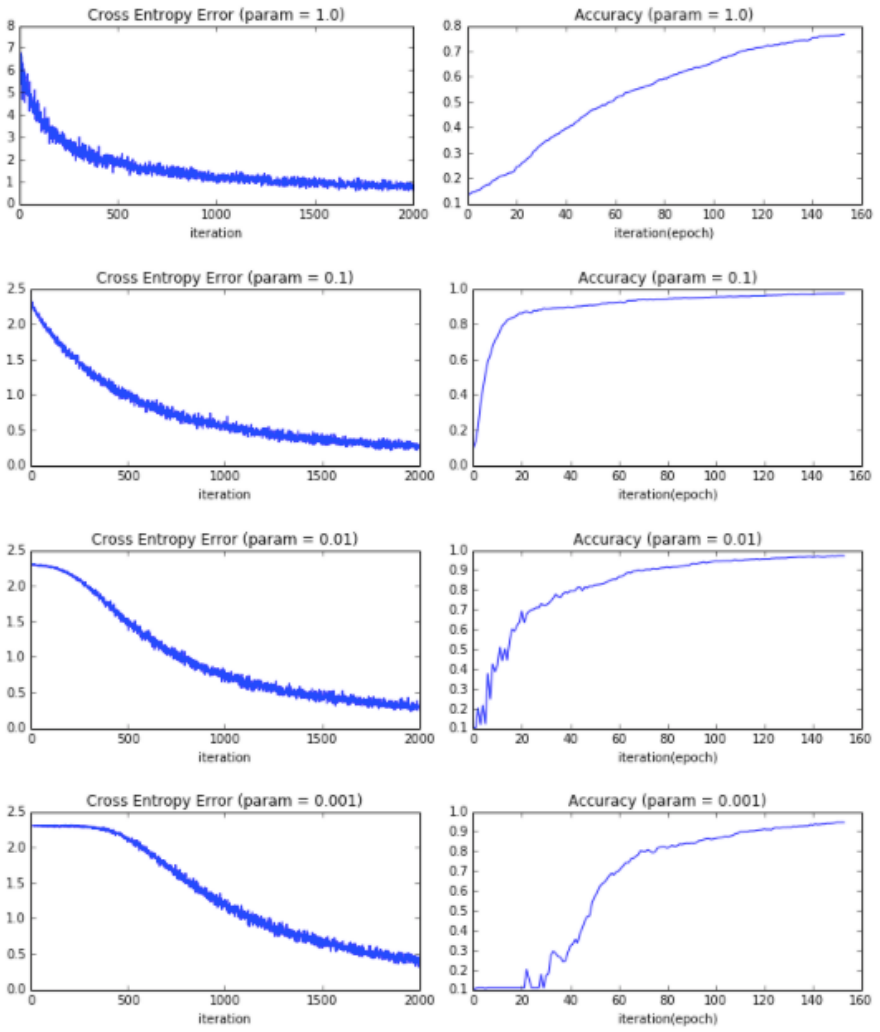
- 活性化関数の選択
  - アプローチ
    - 損失関数の値が小さいと勾配消失
      →損失関数の値がある程度大きくなるような活性化関数を使う
  - ReLU関数
    - シグモイド関数と違い、大きな値ほど出力の変化が大きいため、勾配消失問題が起こりにくい
    - 最も使われている活性化関数





勾配消失問題

- 重みの初期値設定
  - アプローチ
    - 重みの初期値設定によって収束したり発散したりする
      →最適な重みを設定すれば勾配消失は起こらない



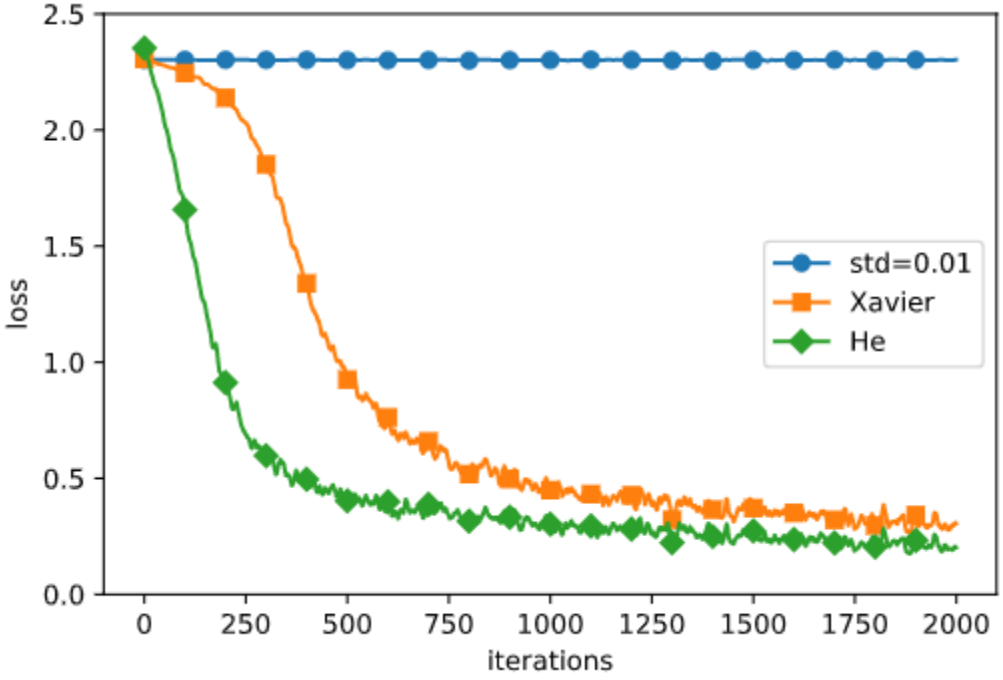
初期値による学習推移の変化

勾配消失問題

- 重みの初期値設定
  - 重みの初期値を決める手法
    - Xavier
      - 平均0、標準偏差$\frac{1}{\sqrt{n}}$である正規分布から重み生成
      - 活性化関数がシグモイド関数やtanh関数の時によく使う
    - He
      - 平均0、標準偏差$\sqrt{\frac{2}{n}}$である正規分布から重み生成
      - 活性化関数がReLUの時によく使う

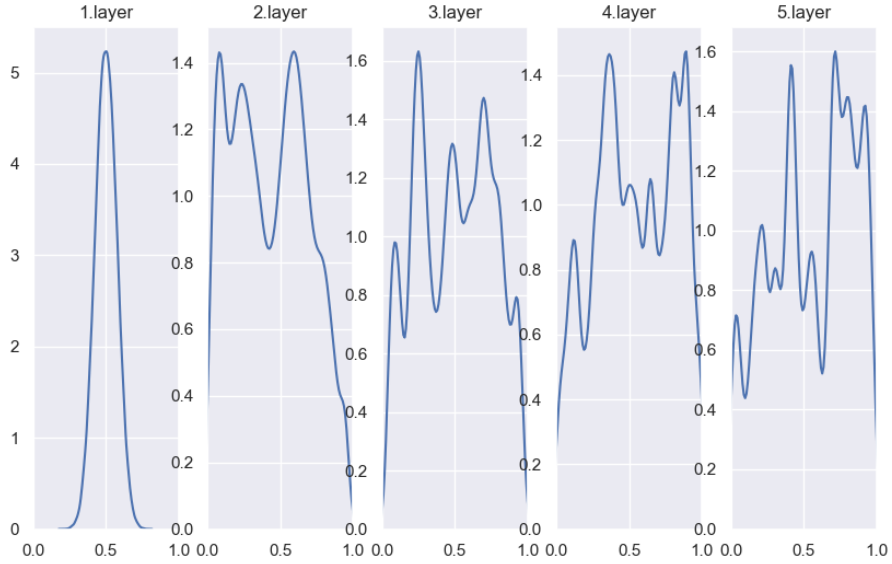
勾配消失問題

- 重みの初期値設定
  - 例：MNIST
    - 5層のネットワーク
    - 重みの初期値を以下のように設定
      1. 標準偏差0.01
      2. Xavier
      3. He

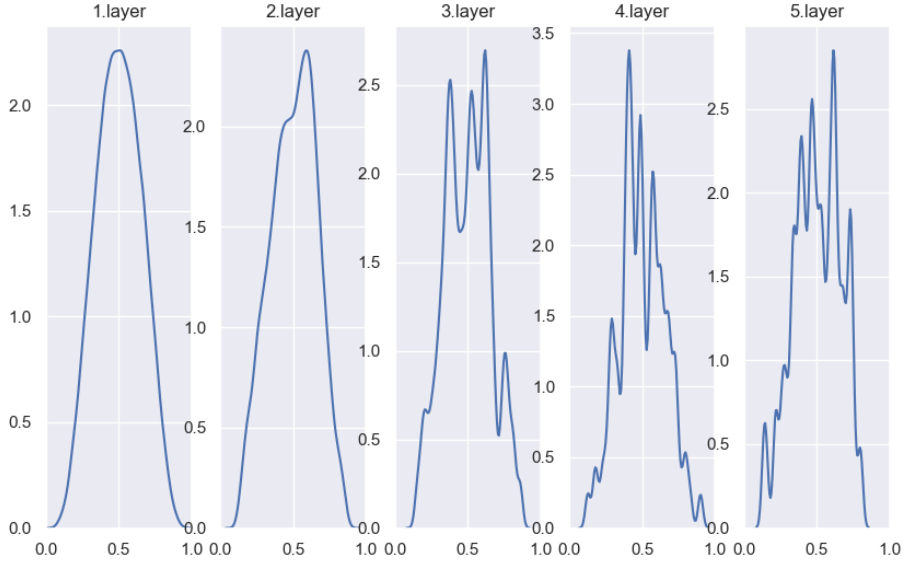


勾配消失問題

- バッチ正規化
  - アプローチ
    - ミニバッチ単位での入力値のデータの偏りを抑制
  - 入力値を平均0、分散1に正規化する
  - 精度向上や収束を早める効果がある



バッチ正規化前の各層の出力分布



バッチ正規化後の各層の出力分布

学習率最適化

- 学習率の更新
  - 学習率が大きい
    - 最適にいつまでもたどり着かず発散（振動）
  - 学習率が小さい
    - 収束までに時間がかかる
    - 局所解に陥りやすい

- 最適化手法
  - 上記問題を解決するために、様々な更新手法が存在
    - モメンタム
    - AdaGrad
    - RMSProp
    - Adam

学習率最適化

- モメンタム
  - アプローチ
    - 現在の勾配の向きにたいして、今まで動いていた方向の慣性を加算
  - 数式
    - $V_t = \mu V_{t-1} - \epsilon \nabla E$
    - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + V_t$
    - 慣性：μ
  - メリット
    - 大域最適解となりやすい
    - 谷間についてから最も低い位置に行くまでの時間が早い

学習率最適化

- AdaGrad
  - アプローチ
    - 各次元ごとに学習率を調整する
  - 数式
    - $h_0 = \theta$
    - $h_t = h_{t-1} + (\nabla E)^2$
    - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t}} \nabla E$
  - メリット
    - 勾配の緩やな斜面に対して、最適解に近づける
  - 課題
    - 学習率が徐々に小さくなるので、鞍点問題を引き起こすことがある

学習率最適化

- RMSProp
  - アプローチ
    - 勾配の大きさに応じて学習率を調整する
  - 数式
    - $h_t = \alpha h_{t-1} + (1-\alpha)(\nabla E)^2$
    - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \epsilon \frac{1}{\sqrt{h_t}+\theta} \nabla E$
  - メリット
    - 大域最適解になりやすい
    - ハイパーパラメータの調整が必要な場合が少ない

学習率最適化

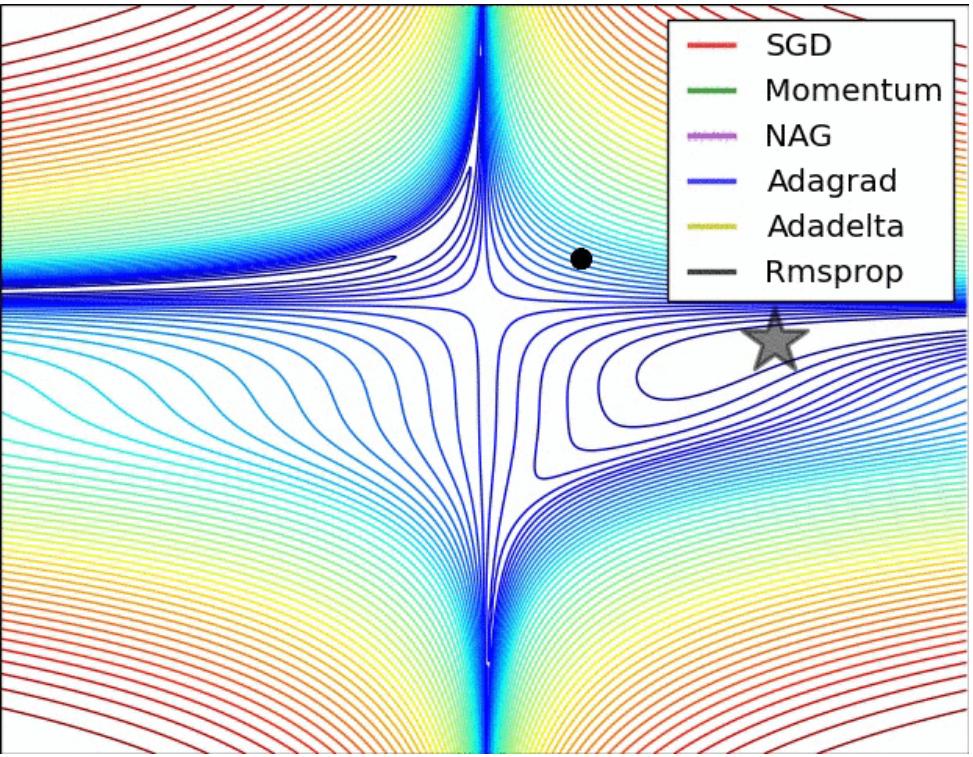
- Adam
  - アプローチ
    - モメンタムとRMSPropのいいとこどり
  - 数式

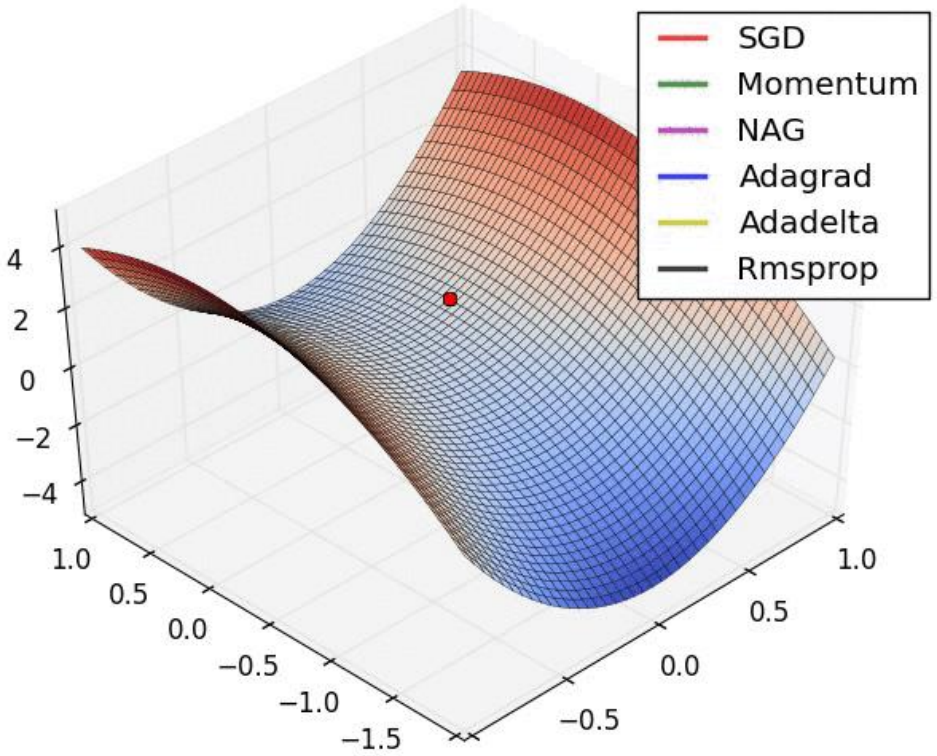
$$\nu_t = \beta_1 \nu_{t-1} + (1 - \beta_1) G$$
$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) G^2$$
$$w_t = w_{t-1} - \alpha \frac{\nu_t}{\sqrt{s_t + \epsilon}} G$$

  - メリット
    - モメンタムとRMSPropのメリットをどちらも含む
  - デファクトスタンダード

学習率最適化

- 最適化手法の比較
  - GIF



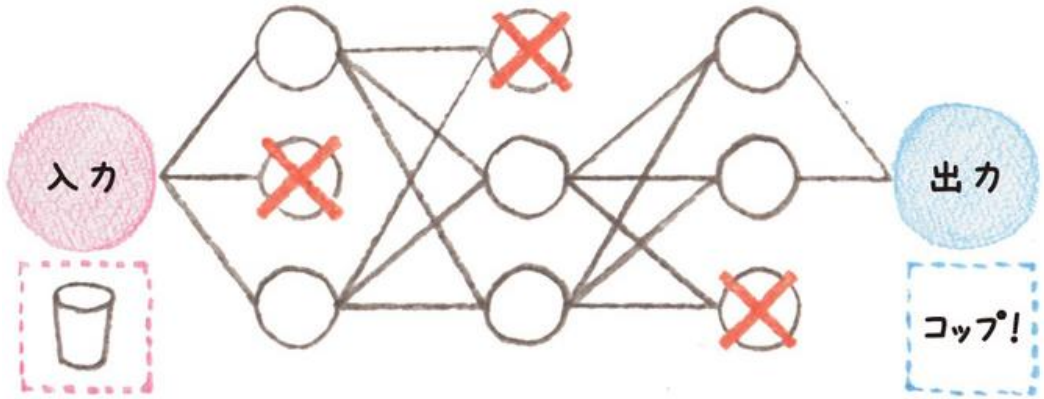


過学習

- 過学習復習
  - テスト誤差と訓練誤差とで学習曲線が乖離すること
    - 特定の訓練サンプルに特化して学習してしまう
- 深層学習における過学習
  - 重みが大きい値をとることで、過学習が発生
    - L1正則化、L2正則化
      →機械学習概論を参考
  - ノード数が多いと、表現自由度が高く、過学習がおきやすい
    - ドロップアウト法

過学習

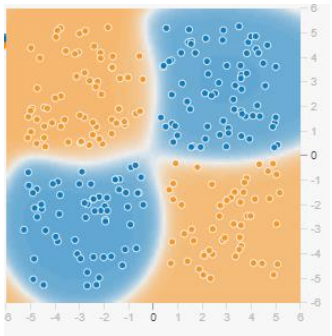
- ドロップアウト
  - ある更新で、ランダムにノードを無効化して学習させる手法
  - 更新ごとに無効化するノードを変える
    →汎化性能を上げる

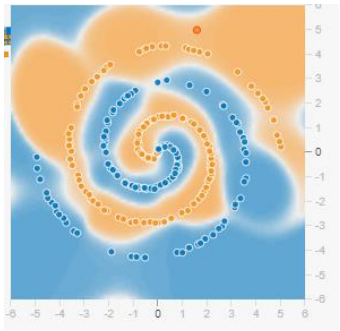


ドロップアウトのイメージ

演習①

- 深層学習体験
  - 以下のサイトにアクセスし、ネットワークの構築をやってみる
    - https://deepinsider.github.io/playground/
  - XORと螺旋のデータセットについて、うまく分類できるパラメータを探す
    - 例：

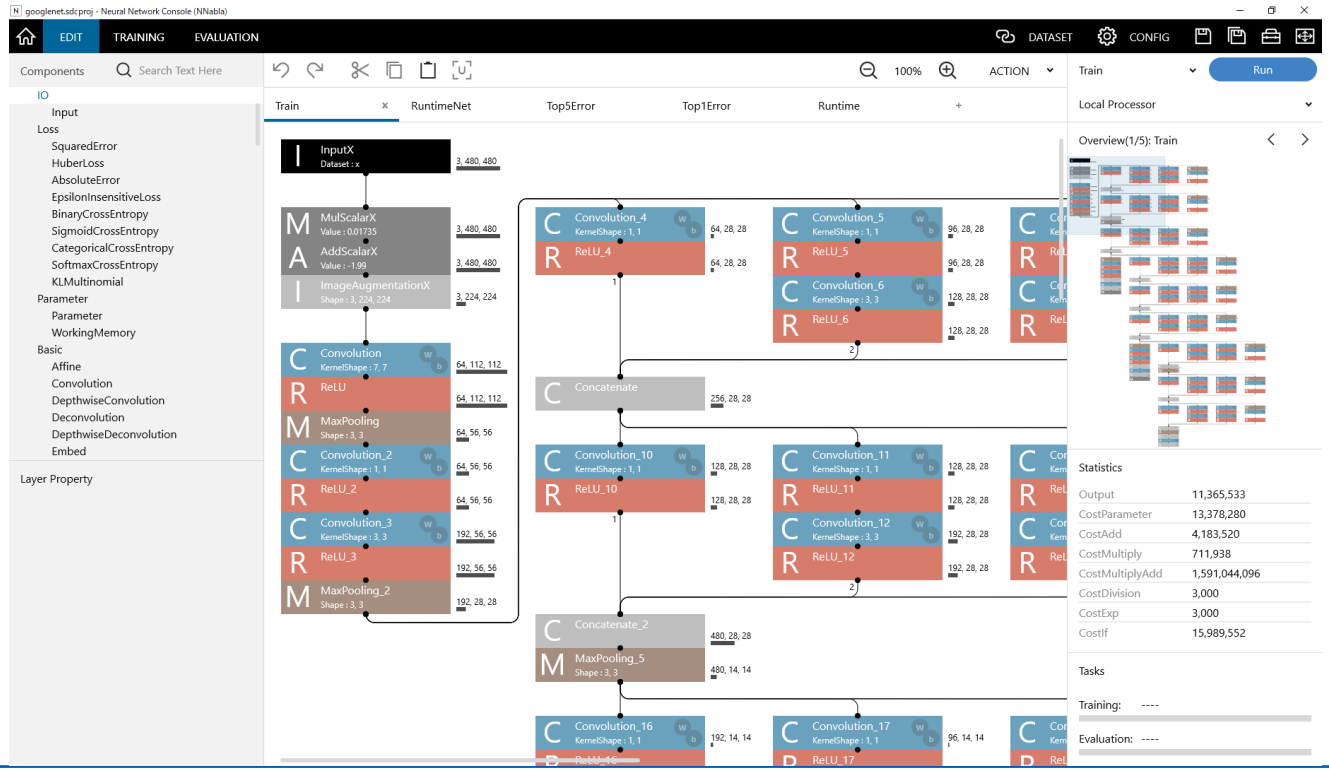




  - うまく分類できたパラメータを記録し、URLを張り付けて提出
  - 提出先：
    - ファイル ＞ AI概論_提出用フォルダ ＞ 深層学習 ＞ 演習①
    - ファイル名：深層学習①_名前.拡張子

参考

- Sony Newral Network Console
  - Sonyが提供しているソフトウェア
  - GUIベースでネットワークの構築が可能
  - 無料利用可能



目次

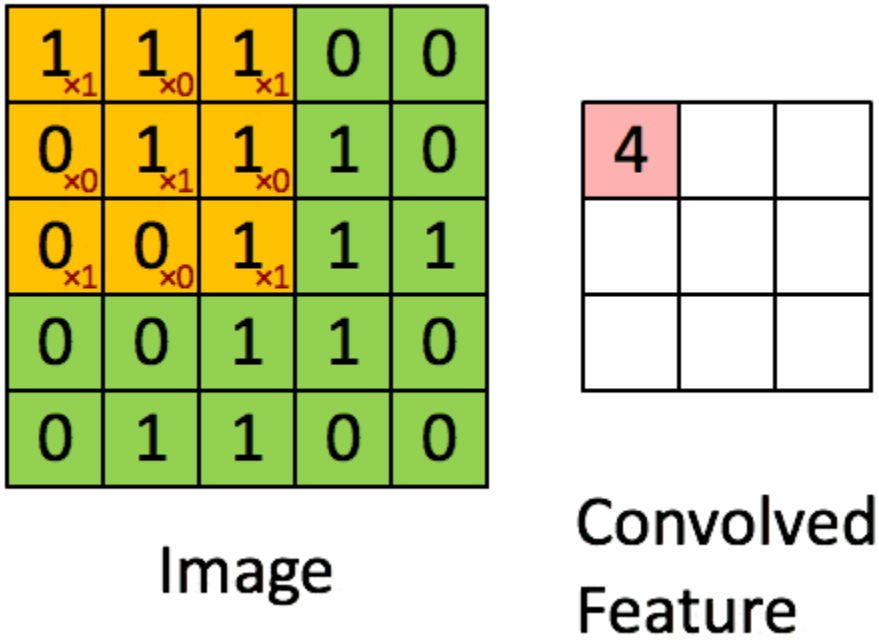
- 深層学習とは
- 深層学習の仕組み
- 深層学習モデルの学習テクニック
- 代表的なDNN
- まとめ

畳み込みニューラルネットワーク

- 畳み込みニューラルネットワークとは
  - 畳み込み層とプーリング持つニューラルネットワーク
  - 畳み込み層
    - 画像の畳み込み層演算を行うレイヤー
  - プーリング層
    - プーリング処理を行うレイヤー
- 主な応用先
  - 物体検知
  - セグメンテーション

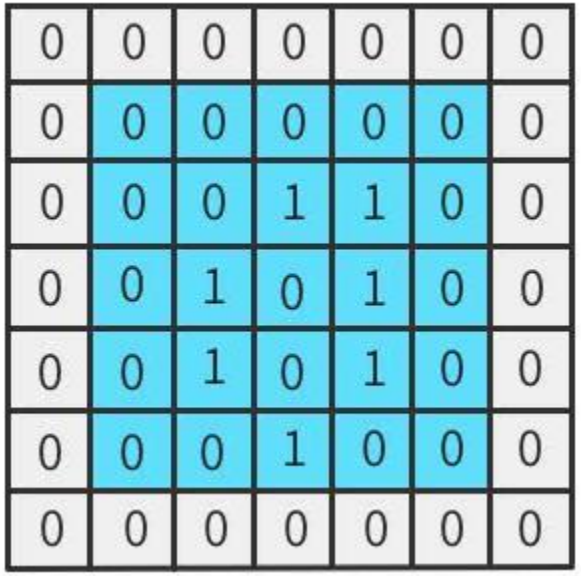
畳み込みニューラルネットワーク

- 畳み込み層
  - 畳み込み層の演算
    - 入力画像に畳み込みフィルターを乗算し、バイアスを足す



畳み込みニューラルネットワーク

- パディング
  - 入力画像の外側に固定データを埋める処理
  - メリット
    - 端のデータに対する畳み込み回数が増えるので端の特徴も考慮されるようになる
    - 畳み込み演算の回数が増えるのでパラメーターの更新が多く実行される
    - カーネルのサイズや、層の数を調整できる

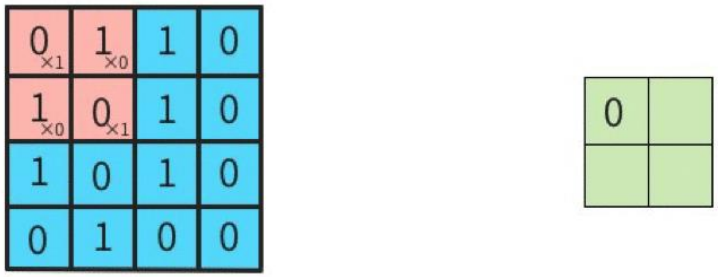


畳み込みニューラルネットワーク

- ストライド
  - フィルターを適用する際の移動間隔
  - 出力される特徴マップのサイズが変わる

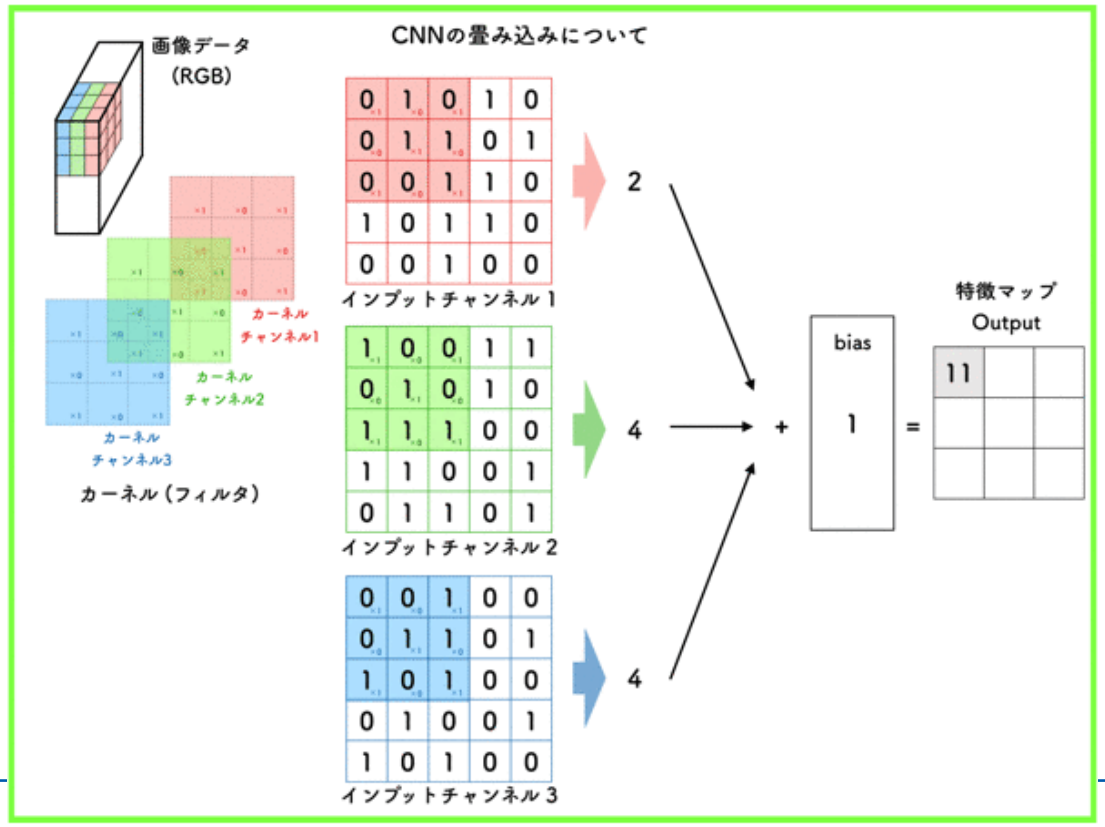
ストライド1

ストライド2



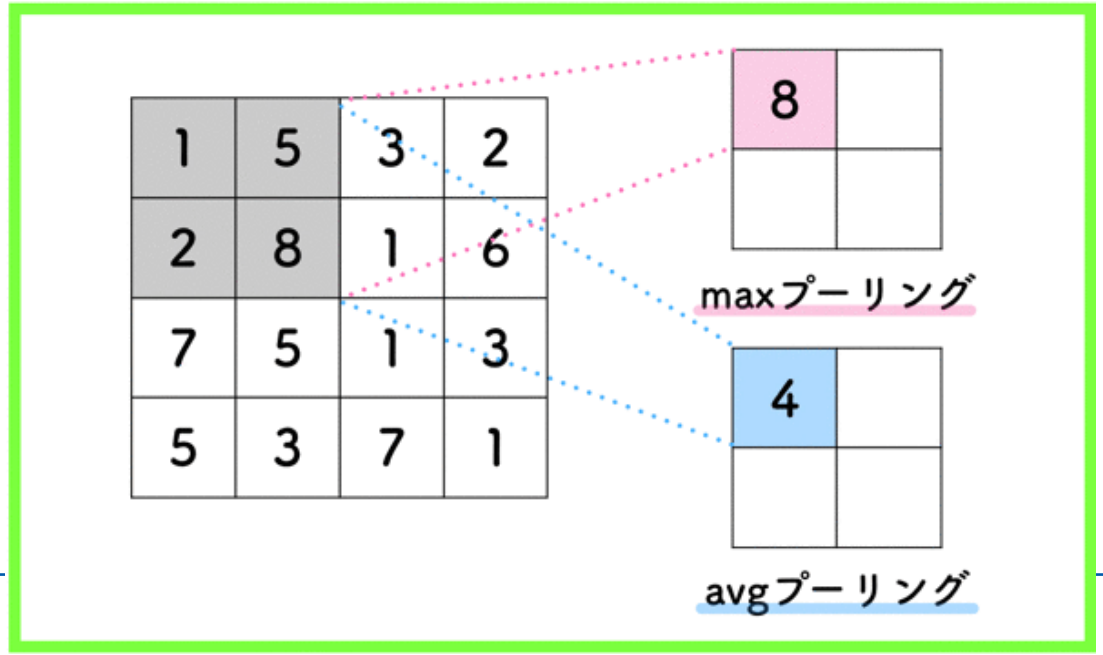
畳み込みニューラルネットワーク

- チャンネル
  - 画像データは主にRGBの3チャンネルを持つ
  - チャンネルを奥行として、３次元での畳み込みを行う



畳み込みニューラルネットワーク

- プーリング層
  - 縦・横方向の空間を小さくする演算
- 演算概念
  - 対象領域のMax値または平均値を取得
  - 4*4の画像に3*3のフィルタを適用し、2*2の出力を得る



畳み込みニューラルネットワーク

- プーリング層の特徴
  - 学習するパラメータがない
  - チャンネル数は変化しない
  - 微小な位置変化に対してロバスト（頑強）

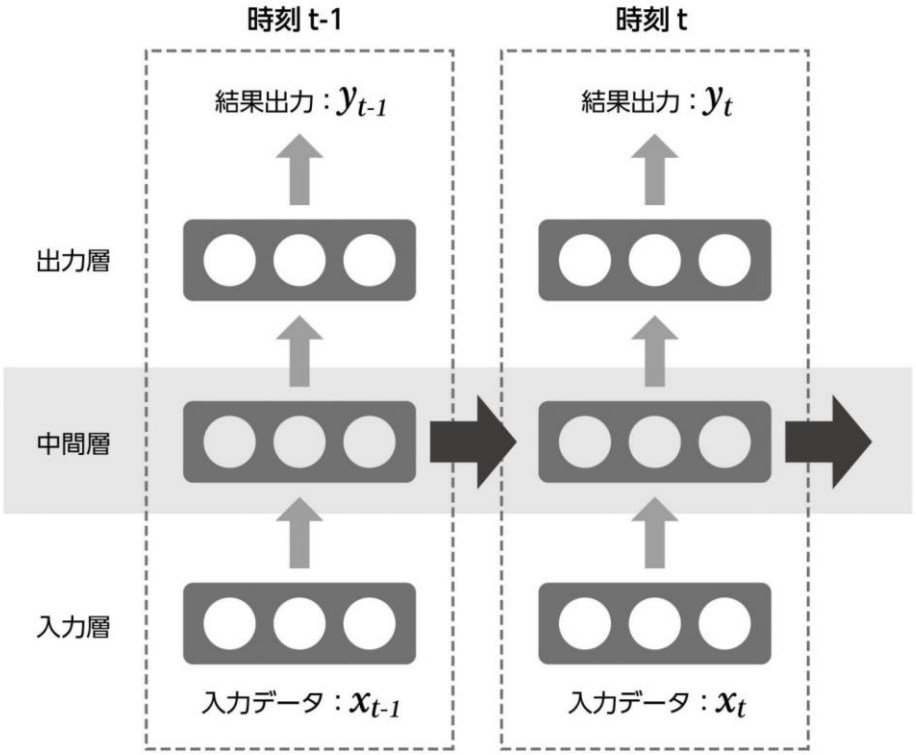
畳み込みニューラルネットワーク

- 代表的なモデル
  - AlexNet(2012)
  - VGG(2014)
  - ResNet(2015)
  - SSD(2016)
  - YOLO(2016)
- 画像処理編で説明

再帰型ニューラルネットワーク

- 再起型ニューラルネットワークとは
  - 全時刻からのディレイ入力を加えることで、時系列データに適応したネットワーク

- 主な応用先
  - 機械翻訳
  - 音声認識
  - 時系列予測

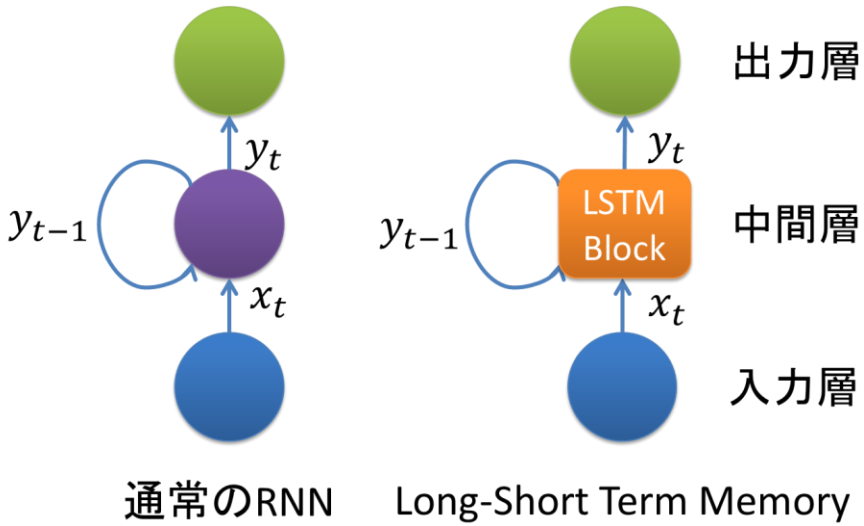


再帰型ニューラルネットワーク

- 代表的なモデル
  - LSTM
  - GRU
  - 双方向RNN
  - RNN　Encoder-Decoder
- 自然言語処理編で説明

再帰型ニューラルネットワーク

- LSTM（Long Short Term Memory）
  - 中間ユニット（メモリユニット）を持つネットワーク
  - 中間層が以下のモジュール群に置き換わったもの

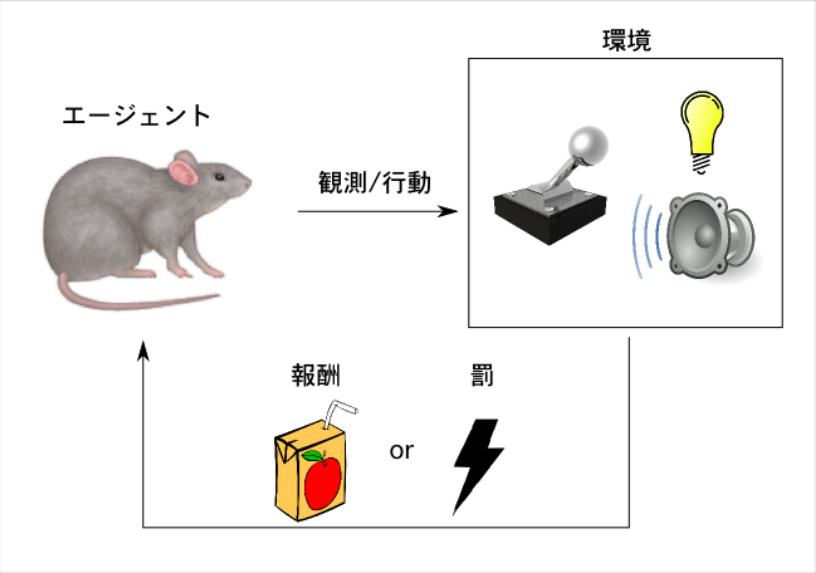


  - 参考
    - https://qiita.com/KojiOhki/items/89cd7b69a8a6239d67ca

深層強化学習

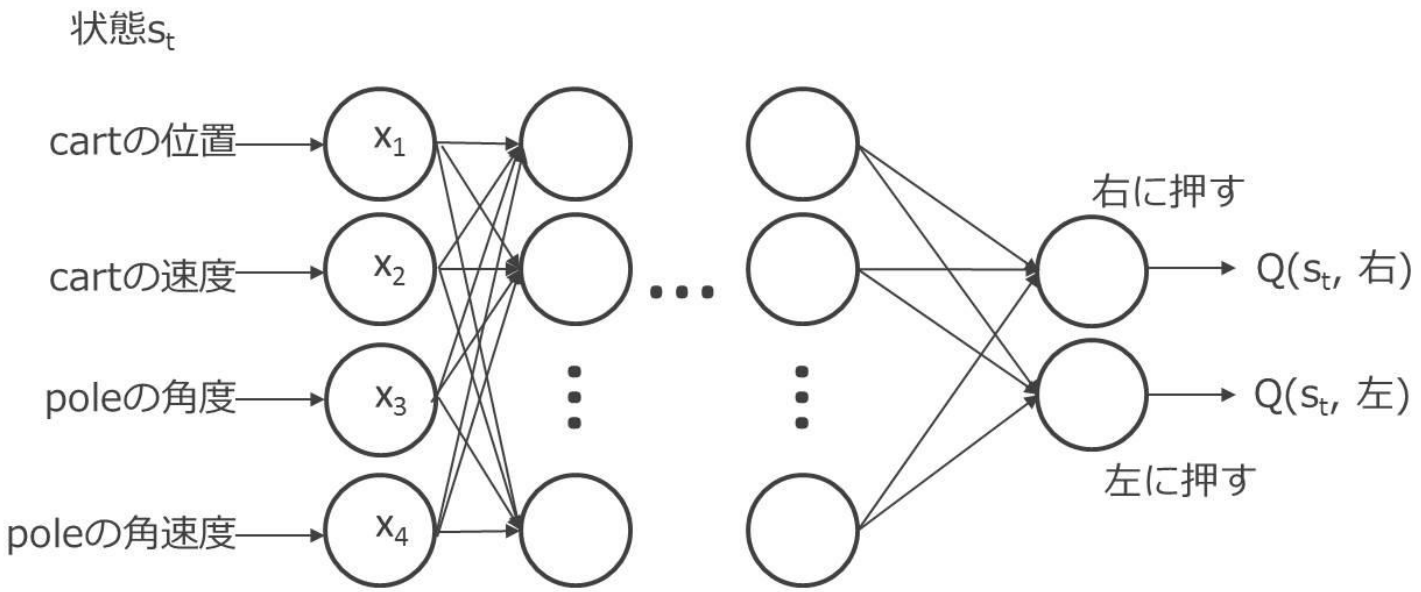
- 深層強化学習とは
  - 深層学習＋強化学習

- 代表的な手法
  - 価値ベース
    - DQN
    - R2D2
  - 方策ベース
    - A3C
    - UNREAL
    - PPO



深層強化学習

- DQN（Deep Q Network）
  - Q学習＋DNN
  - 最適行動価値関数をNNを使った近似関数で求める



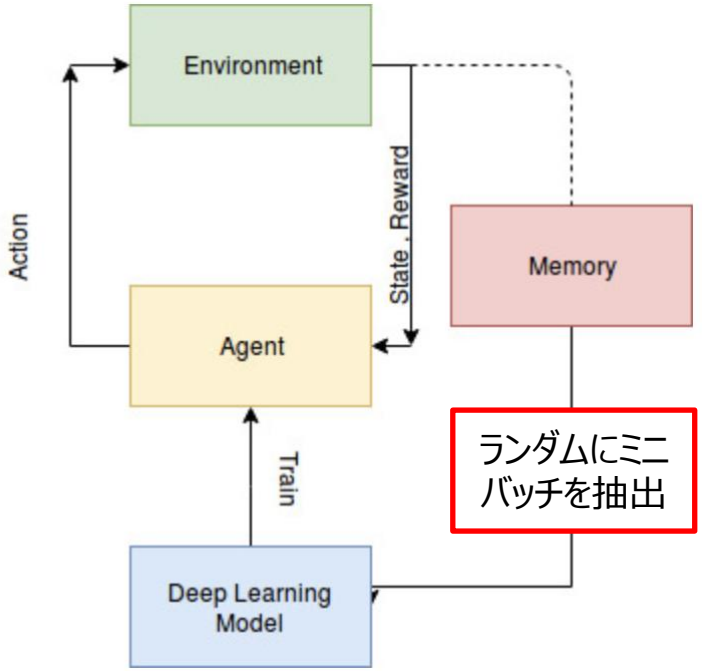
例：自動運転におけるQNDのネットワーク

深層強化学習

- DQN （Deep Q Network）
  - 最適行動価値関数のNNを使って近似しただけでは、DQNの学習は不安定であり良い方策が得られない
  - 安定した学習を実現させるための４つの工夫
    - Experience Replay
    - Fixed Target Q-Network
    - 報酬のclipping
    - Huber関数

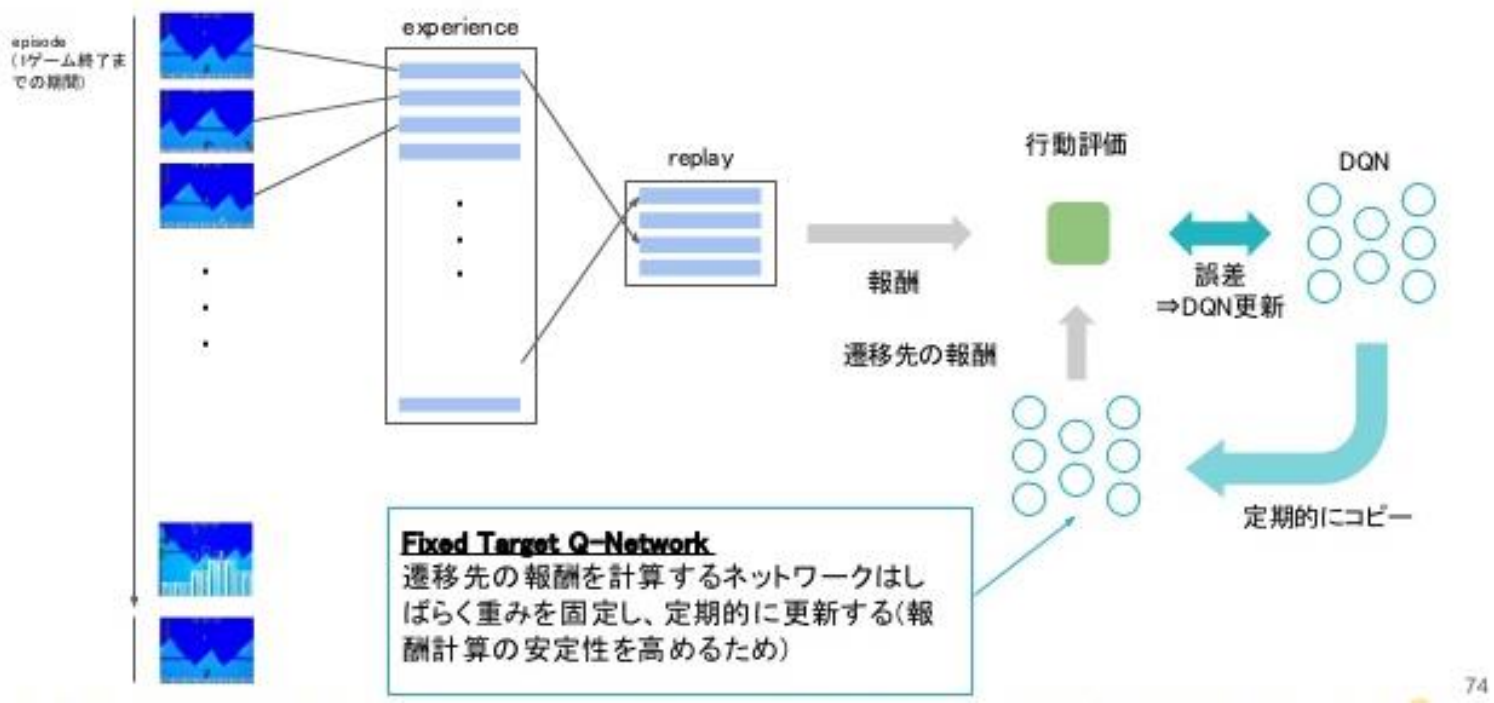
深層強化学習

- Experience Replay
  - 各ステップごとに学習すると、時間的に相関の高い内容を学習するため、安定しづらい
  - １ステップごとに学習するのではなく、メモリに各ステップの情報を保存しておき、ランダムに取り出して学習させる



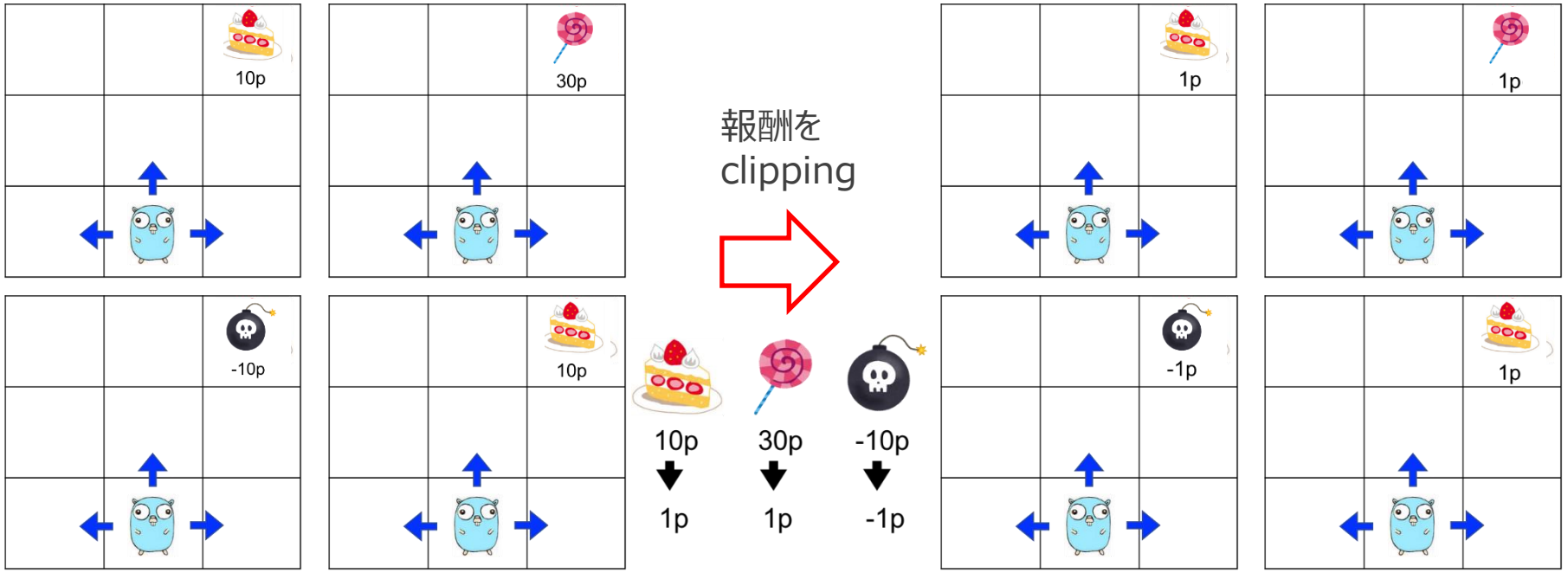
深層強化学習

- Fixed Target Q-Network
  - NNの学習にNNの出力を使用する学習が安定しづらい
  - 主たるNNとは別に、最適行動価値関数を求めるネットワークを準備する



深層強化学習

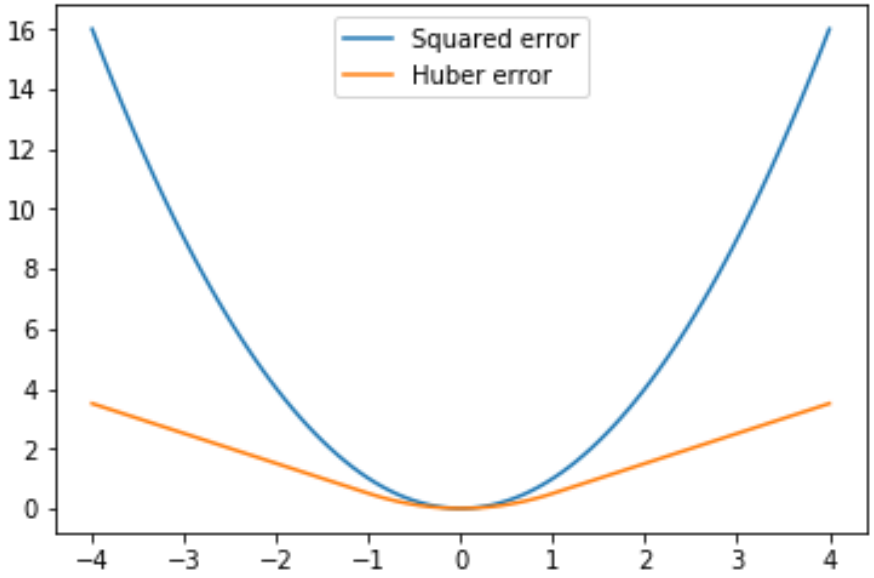
- 報酬のclipping
  - ゲームの内容によって報酬が異なると、同じハイパーパラメータでのDNNの実行が困難
  - 各ステップで得られる報酬を固定する



ゲームごとに報酬がバラバラ

深層強化学習

- Huber関数
  - 誤差の出力が大きいと、学習が安定しづらい
  - 二乗誤差では損失が大きくなると勾配が発散してしまう
  - Huber関数を使うことで、勾配を一定値に固定することが可能



GAN（生成モデル）

- GANとは
  - 敵対的生成ネットワーク
  - データから特徴を学習することで、実在しないデータを生成したり、存在するデータの特徴に沿って変換できる
- 例

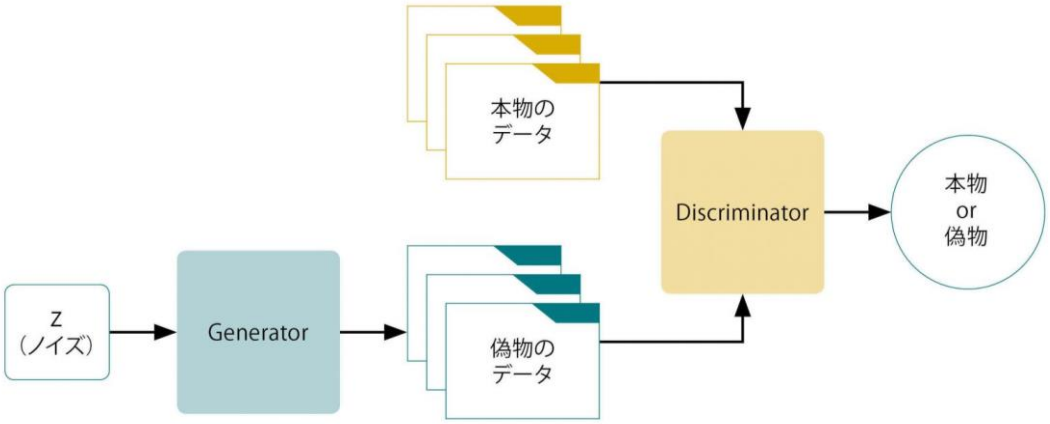
存在しないベッドルームの画像合成

線画の着色

画風の変換

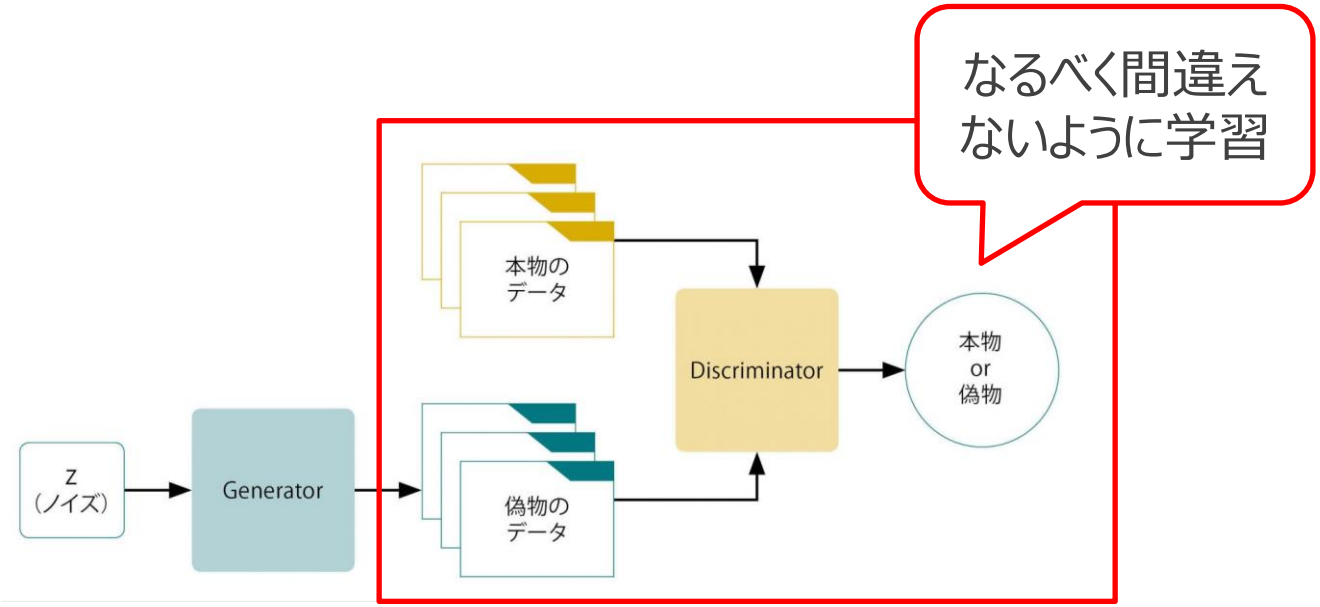
GAN（生成モデル）

- GANの仕組み
  - 2つのニューラルネットワークで構成
    - Generator
      - データを生成
    - Discriminator
      - Generatorが生成した偽物のデータと本物のデータの真偽判定を行う
  - アーキテクチャ



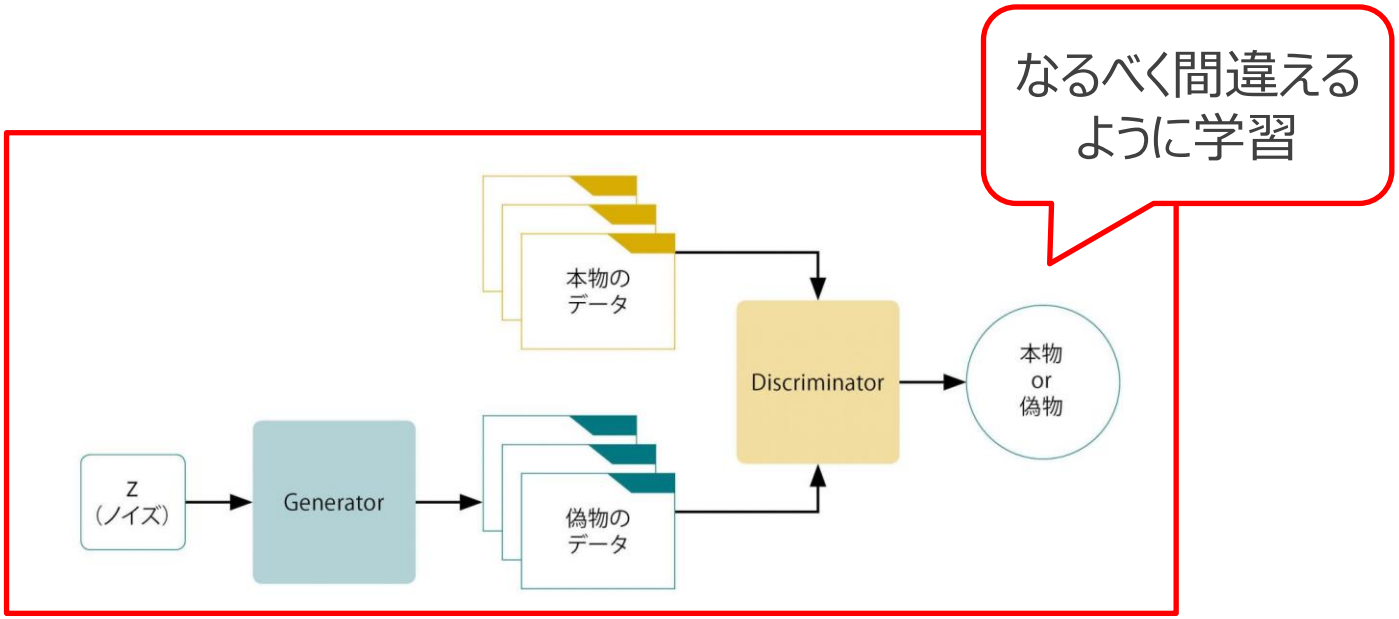
GAN（生成モデル）

- Discriminator
  - Discriminatorの分類誤差が最小になるように
    パラメータを更新



GAN（生成モデル）

- Generator
  - Discriminatorの分類誤差が最大になるように学習

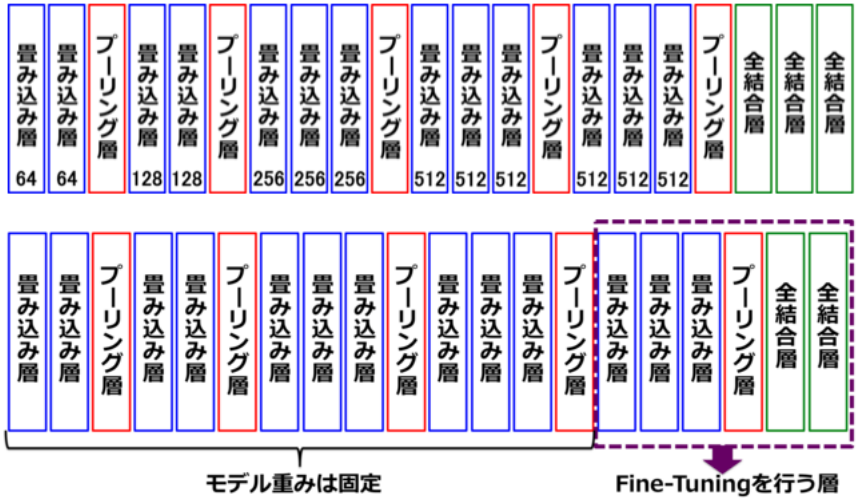


GAN（生成モデル）

- 応用例
  - Conditional GAN
  - SRGAN
  - pix2pix
  - Cycle GAN

転移学習とファインチューニング

- 転移学習とファインチューニングの違い
  - 転移学習
    - 既存の学習済モデルを、重みデータは変更せずに特徴量抽出器として利用する
  - ファインチューニング
    - 既存の学習済モデルを、重みデータの一部を再学習させて利用する
  - イメージ図



VGG-16の学習済モデルの構造と重みを使って転移学習

VGG-16の学習済モデルの前半はそのまま、出力層は変更し、ファインチューニング

転移学習とファインチューニング

- 応用例
  - BERT
    - 事前言語処理のタスクにおける汎用言語表現モデル
  - VGG16
    - 画像処理タスクにおける汎用画像表現モデル

演習②

- 事例調査
  - CNN、RNN、それぞれのネットワークについて、応用事例を調査する
  - 以下内容をまとめて提出すること
    - 実現できる機能
    - ネットワークの名前（あれば）
    - ネットワーク図
    - 入出力データ
    - ネットワークの概要（各層のノード数、レイヤー、活性化関数、誤差関数など）
    - ネットワークの特徴
    - その他特筆事項
  - フォーマットは自由
  - 時間が余った方は、GANや深層強化学習についても調査してください
  - 提出先：
    - ファイル ＞ AI概論_提出用フォルダ ＞ 深層学習 ＞ 演習②
    - ファイル名：深層学習②_名前.拡張子

目次

- 深層学習とは
- 深層学習の仕組み
- 深層学習モデルの学習テクニック
- 代表的なDNN
- まとめ

まとめ

- 2012年に登場して以来、深層学習の研究が盛ん
- 深層学習はニューラルネットワークの発展形であり、ネットワーク構造や学習方式を工夫することで、様々な分野に応用可能
- 近年では、転移学習やファインチューニングといった学習方式も登場し、少ないリソースでも高精度のモデル作成が可能になった

参考資料

- 深層学習とTensorFlow入門
- ゼロから始める深層強化学習
- GANのまとめ (基礎)