2019年度「専修学校による地域産業中核的人材養成事業」

機械学習Ⅱ教材

Society5.0実現のためのIT技術者養成モデルカリキュラム開発と実証事業

2019年度「専修学校による地域産業中核的人材養成事業」

機械学習II教材

Society5.0 実現のための IT 技術者養成モデルカリキュラム開発と実証事業

目 次

第1回：機械学習の概要

機械学習の種類とパーセプトロン

アジェンダ

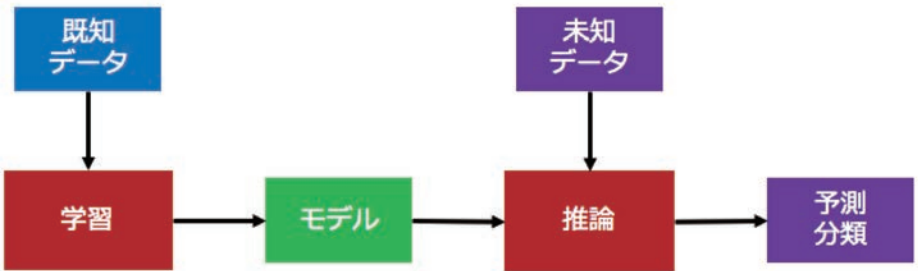
- 機械学習とは
- 機械学習の手法の分類
  - 教師あり学習
  - 教師なし学習
  - 半教師あり学習
  - 強化学習
- 代表的なアルゴリズム
- パーセプトロン
  - 歴史と発展
  - アルゴリズムの概要

全15回の講義について

- 主にニューラルネットワークの仕組み、学習アルゴリズムの理解を目標とする。
  - プログラミング言語としてはPython
  - Pythonの各種ライブラリを利用してデータ分析に必要なスキルの習得を目指す
  - 第2回以降の講義で詳細を取り扱う

機械学習とは？

既知のデータからルールを見つけ出す「学習」と、見つけたルールを用いて未知のデータに適用して結論を得る「推論」から成ります



機械学習とは？

赤い点が猫、青い点が犬を表していて、鼻の大きさと尻尾の長さのデータをプロットしているとします

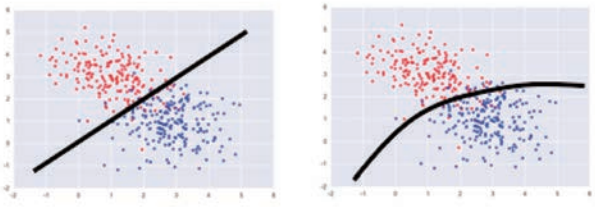
- 学習とは、猫と犬を見分けるためのルールを見つけ出すこと
- 推論とは、初めて見る動物でも、鼻の大きさと尻尾の長さから、猫なのか犬なのか判断すること

機械学習とは？

学習には「アルゴリズムを選ぶ」ことと「パラメータを決める」２つのステップが必要になります

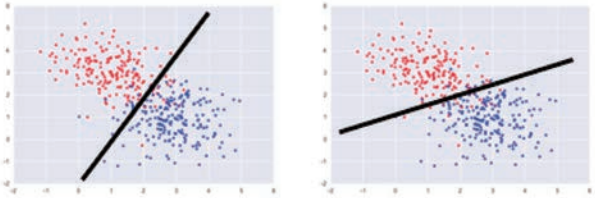
アルゴリズムを選ぶ

どのように判断するかを決める
・直線で分ける？
・曲線で分ける？
・その他の方法で分ける？



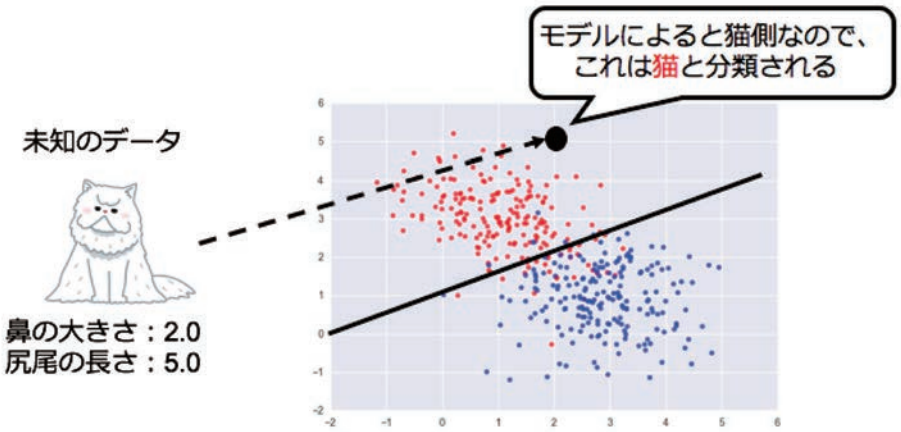
パラメータを決める

直線の場合は傾きを決める
何らかの評価基準に従って
「最も良い」傾きを決めます



機械学習とは？

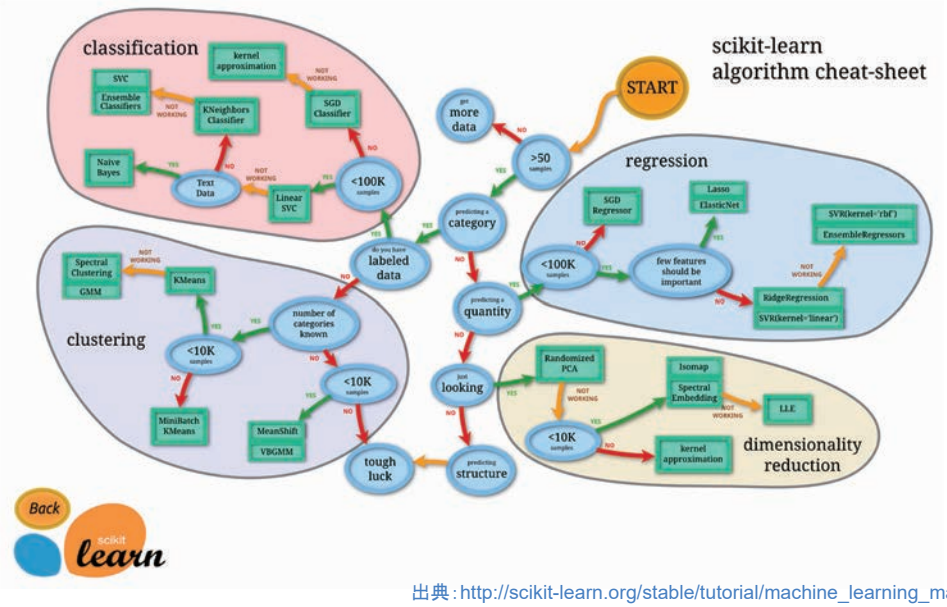
得られた学習結果（モデル）をもとに判断（猫か犬かなので分類）が行われます



機械学習の手法の分類

- 機械学習の手法にはいくつかの分類方法がありますが、代表的なものは以下の3つに分けるものです。
- 教師あり学習
  - 教師データで与えられた正解に近づくように学習し、未知のデータに対しても正しい答えを導き出せるようにする手法
- 教師なし学習
  - 教師データは与えられず、与えられたデータの構造から何らかのルールを導き出せるようにする手法
- 半教師あり学習
  - 少数の教師ありデータと大量の教師なしデータが存在する場合に、教師なしデータも活用してモデルを高度化する手法
- 強化学習
  - 試行錯誤を繰り返しながら、より良い結果にたどり着くための方法を獲得していく方法

機械学習の手法の選び方



出典：http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

（機械学習の）代表的なアルゴリズム

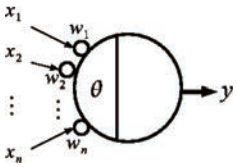
- 分類
  - ロジスティック回帰
  - 決定木
  - サポートベクターマシン
  - ニューラルネットワーク
- 回帰
  - 線形回帰
  - ニューラルネットワーク
  - 回帰木
- クラスタリング
  - K-means法
- 次元削減
  - 主成分分析

※各アルゴリズムの詳細は「機械学習III」にて取り扱います。

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

樹状突起からの入力（x_i）を重みw_iで受け付け、細胞体で電位を加算する。
加算値と閾値θの関係によって発火時の神経インパルスを出力（y）する



$$y = f\left(\sum_{k=1}^{n} w_k x_k - \theta\right)$$

出力の計算式

$$f(u) = \begin{cases} 1 & (u > \theta) \\ 0 & (u \le \theta) \end{cases}$$

$$f(u) = \begin{cases} 1 & (u \ge \theta) \\ \dfrac{u+\theta}{2\theta} & (|u| < \theta) \\ 0 & (u \le -\theta) \end{cases}$$

$$f(u) = \frac{1}{1+\exp(-\varepsilon u)}$$

活性化関数

出展：http://www.gifu-nct.ac.jp/elec/deguchi/sotsuron/oka/node5.html

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

発火の連鎖を繰り返す細胞同士の結合は強まる。

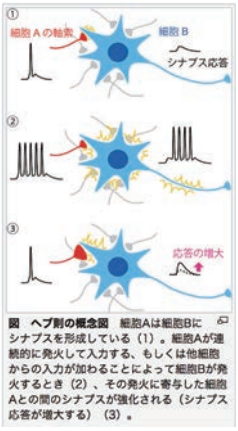


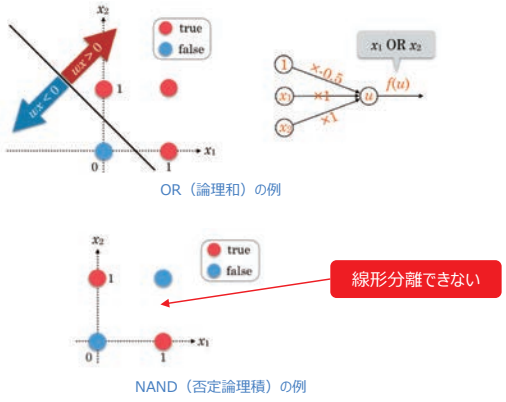
図　ヘブ則の概念図　細胞Aは細胞Bにシナプスを形成している（1）。細胞Aが連続的に発火して入力する、もしくは他細胞からの入力が加わることによって細胞Bが発火するとき（2）、その発火に寄与した細胞Aとの間のシナプスが強化される（シナプス応答が増大する）（3）。

出展：https://bsd.neuroinf.jp/wiki/ヘブ則

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

単純パーセプトロンでは線形非分離な問題が解決できない。



OR（論理和）の例

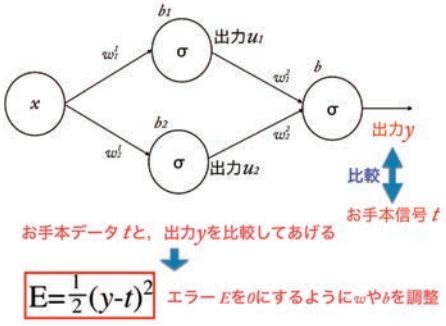
NAND（否定論理積）の例

出展： http://hokuts.com/2015/12/04/ml3-mlp/

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

バックプロパゲーション（誤差逆伝播法）を導入することにより多層パーセプトロンの学習が可能となり、線形非分離な問題が解決できるようになった。

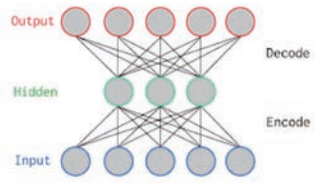


出展： https://www.yukisako.xyz/entry/backpropagation

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

オートエンコーダはニューラルネットを利用した次元圧縮アルゴリズム。Inputの情報を一旦Hiddenの次元まで圧縮する（Encode）。その後Hiddenの情報量でOutputを表現する（Decode）。
DecodeしたOutputが答えに近くなるように重みを学習する。

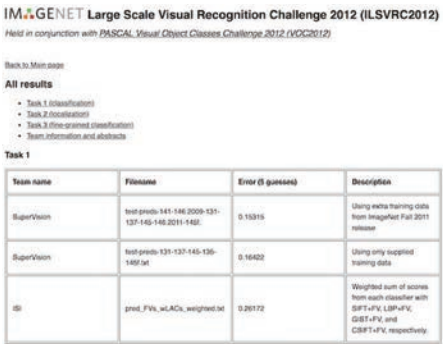


出展： https://ja.wikipedia.org/wiki/オートエンコーダ

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

画像認識コンテスト「ILSVRC（ImageNet Large Scale Visual Recognition Challenge）」で、ヒントン教授らのグループがニューラルネットを用いたSupervisionという手法で、1年前の優勝記録の誤り率25.7%から15.3%へと4割も削減し圧勝した。


IMAGENET Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)

Held in conjunction with PASCAL Visual Object Classes Challenge 2012 (VOC2012)

Back to Main page

All results

- Task 1 (classification)
- Task 2 (localization)
- Task 3 (fine-grained classification)
- Team information and abstracts

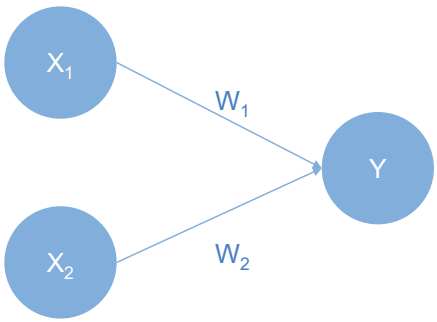
Task 1

Team name	Filename	Error (5 guesses)	Description
SuperVision	test-preds-141-146.2009-131-137-145-146.2011-14tl.	0.15315	Using extra training data from ImageNet Fall 2011 release
SuperVision	test-preds-131-137-145-136-146f.txt	0.16422	Using only supplied training data
ISI	pred_FVs_wLACs_weighted.txt	0.26172	Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.

出展： http://www.image-net.org/challenges/LSVRC/2012/results.html

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンは、複数の入力値に対して1つの出力値を持つ関数です。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

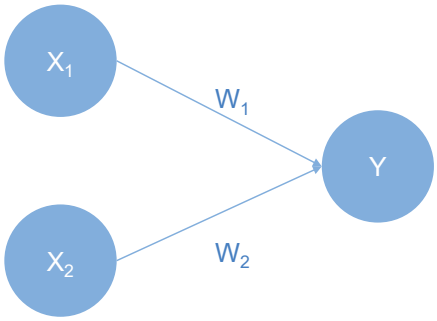
図の◯はニューロンと呼びます。記号の意味は下記の通りです。

X_1、X_2　：入力信号
Y　：出力信号
W_1、W_2　：重み

入力信号X_iに重みW_iが乗算され、その総和が一定の閾値θより大きくなった場合、値1を出力します。（それ以外は0を出力します。）
1を出力した場合「ニューロンが発火する」と表現することがあります。

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンの計算例を示します。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

各数値が

入力信号：$X_1 = X_2 = 1.0$
重み　：$W_1 = W_2 = 0.5$
閾値　：$\theta = 0.7$

の場合、出力信号Yは下記のように計算できます。

$1.0 \times 0.5 + 1.0 \times 0.5 = 1.0$
閾値0.7を超えているため、$Y = 1.0$

第2回：単純パーセプトロン

単純パーセプトロンのアルゴリズム

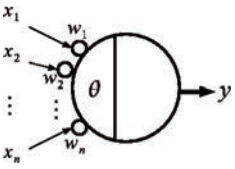
アジェンダ

- 第1回の振り返り
  - パーセプトロンの歴史と発展
  - アルゴリズムの概要
- 単純パーセプトロンのアルゴリズム
- 単純パーセプトロンの実装

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

樹状突起からの入力（x_i）を重みw_iで受け付け、細胞体で電位を加算する。
加算値と閾値θの関係によって発火時の神経インパルスを出力（y）する



$$y = f\left(\sum_{k=1}^{n} w_k x_k - \theta\right)$$

出力の計算式

$$f(u) = \begin{cases} 1 & (u > \theta) \\ 0 & (u \le \theta) \end{cases}$$

$$f(u) = \begin{cases} 1 & (u \ge \theta) \\ \dfrac{u+\theta}{2\theta} & (|u| < \theta) \\ 0 & (u \le -\theta) \end{cases}$$

$$f(u) = \frac{1}{1+\exp(-\varepsilon u)}$$

活性化関数

出展：http://www.gifu-nct.ac.jp/elec/deguchi/sotsuron/oka/node5.html

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

発火の連鎖を繰り返す細胞同士の結合は強まる。

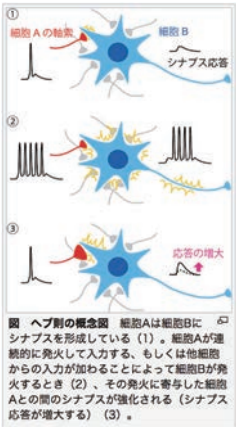


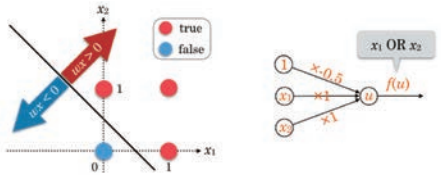
図　ヘブ則の概念図　細胞Aは細胞Bにシナプスを形成している（1）。細胞Aが連続的に発火して入力する、もしくは他細胞からの入力が加わることによって細胞Bが発火するとき（2）、その発火に寄与した細胞Aとの間のシナプスが強化される（シナプス応答が増大する）（3）。

出展：　https://bsd.neuroinf.jp/wiki/ヘブ則

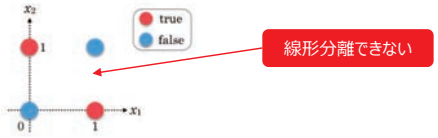
パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

単純パーセプトロンでは線形非分離な問題が解決できない。



OR（論理和）の例



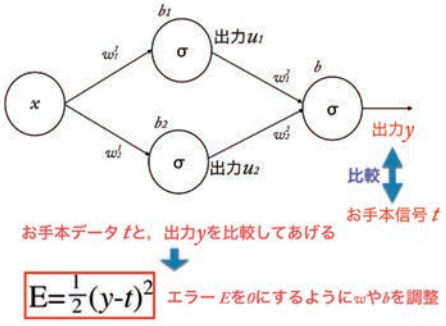
NAND（否定論理積）の例

出展： http://hokuts.com/2015/12/04/ml3-mlp/

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

バックプロパゲーション（誤差逆伝播法）を導入することにより多層パーセプトロンの学習が可能となり、線形非分離な問題が解決できるようになった。

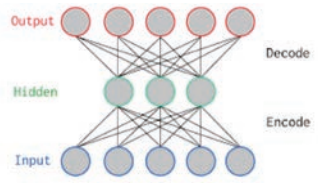


出展： https://www.yukisako.xyz/entry/backpropagation

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

オートエンコーダはニューラルネットを利用した次元圧縮アルゴリズム。Inputの情報を一旦Hiddenの次元まで圧縮する（Encode）。その後Hiddenの情報量でOutputを表現する（Decode）。
DecodeしたOutputが答えに近くなるように重みを学習する。

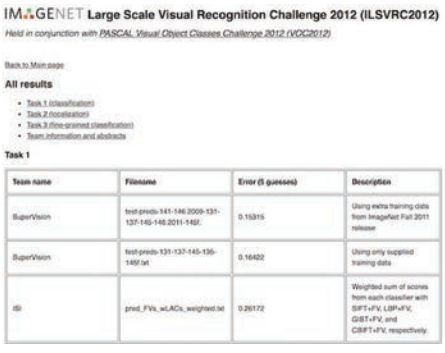


出展： https://ja.wikipedia.org/wiki/オートエンコーダ

パーセプトロン（ニューラルネットワーク）の歴史

年号	技術	人物
1943	形式ニューロンの提案	W. MCCLLOCH W.PITTS
1949	Hebb's rule提案	D.Hebb
1958	パーセプトロンの発表	F. Rosenblatt
1969	神経回路論	甘利教授
1969	コグニトロン神経回路論	福島教授
1969	パーセプトロンの限界提示	M. Minsky
1969～80	AIの冬第1期	------
1984	Hopfield model提案	J. Hopfield
1986	Boltzman Machine提案	Hinton & Sejnowski
1986	Back Propagation紹介	Rumelhart & McClelland
1987～93	AIの冬第２期	------
2006	オートエンコーダ提案	G. Hinton
2012	高精度ディープラーニング作成	G. Hinton

画像認識コンテスト「ILSVRC（ImageNet Large Scale Visual Recognition Challenge）」で、ヒントン教授らのグループがニューラルネットを用いたSupervisionという手法で、1年前の優勝記録の誤り率25.7%から15.3%へと4割も削減し圧勝した。


IMAGENET Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)

Held in conjunction with PASCAL Visual Object Classes Challenge 2012 (VOC2012)

Back to Main page

All results

- Task 1 (classification)
- Task 2 (localization)
- Task 3 (fine-grained classification)
- Team information and abstracts

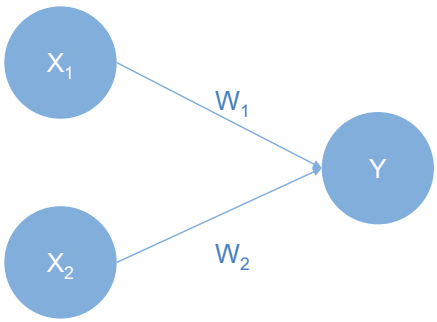
Task 1

Team name	Filename	Error (5 guesses)	Description
SuperVision	test-preds-141-146.2009-131-137-145-146.2011-14tf.	0.15315	Using extra training data from ImageNet Fall 2011 release
SuperVision	test-preds-131-137-145-136-146f.txt	0.16422	Using only supplied training data
ISI	pred_FVs_wLACs_weighted.txt	0.26172	Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.

出展： http://www.image-net.org/challenges/LSVRC/2012/results.html

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンは、複数の入力値に対して1つの出力値を持つ関数です。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

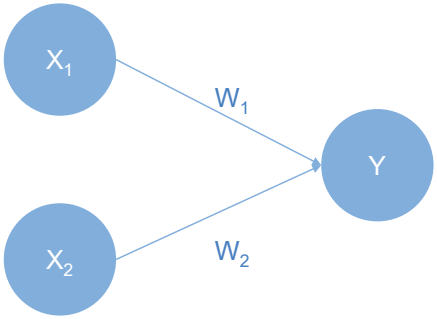
図の◯はニューロンと呼びます。記号の意味は下記の通りです。

X_1、X_2　：入力信号
Y　：出力信号
W_1、W_2　：重み

入力信号X_iに重みW_iが乗算され、その総和が一定の閾値θより大きくなった場合、値1を出力します。（それ以外は0を出力します。）
1を出力した場合「ニューロンが発火する」と表現することがあります。

演習1：単純パーセプトロンの実装

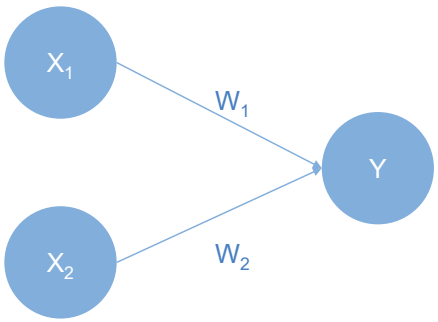
- 変数として定義すべき項目を列挙してください。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンの計算例を示します。

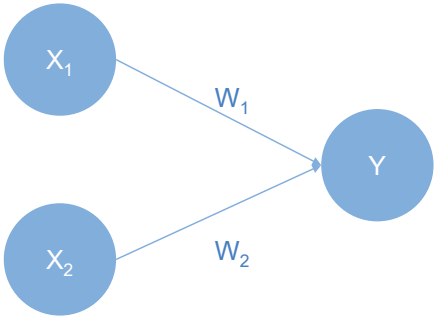


$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

各数値が
入力信号:$X_1 = X_2 = 1.0$
重み　　:$W_1 = W_2 = 0.5$
閾値　　:$\theta = 0.7$
の場合、出力信号Yは下記のように計算できます。
$1.0 \times 0.5 + 1.0 \times 0.5 = 1.0$
閾値0.7を超えているため、$Y = 1.0$

演習2：単純パーセプトロンの実装

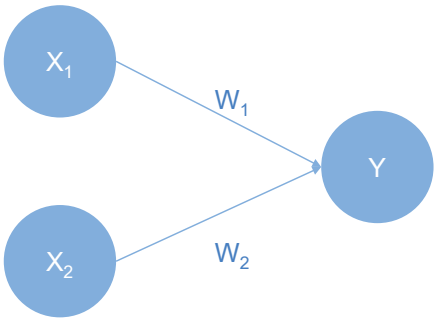
- 入力ニューロンが2つの場合の、単純パーセプトロンの計算式をpythonで実装してください。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

演習3：単純パーセプトロンの実装

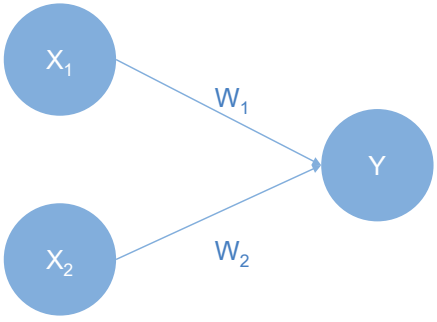
- 入力ニューロン数を柔軟に変更できるアルゴリズムを実装してください。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

演習4：単純パーセプトロンの実装

- 単純パーセプトロンの計算式を関数化してください。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

第3回：単純パーセプトロン

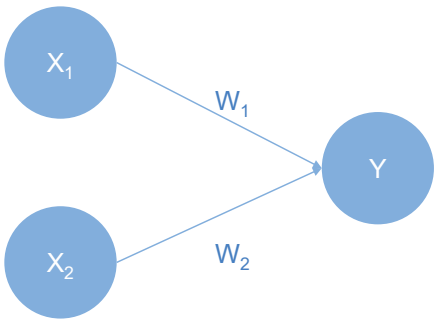
単純パーセプトロンの限界

アジェンダ

- 単純パーセプトロンのアルゴリズムの復習
- 単純パーセプトロンの限界
- 多層パーセプトロンのアルゴリズム

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンは、複数の入力値に対して1つの出力値を持つ関数です。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

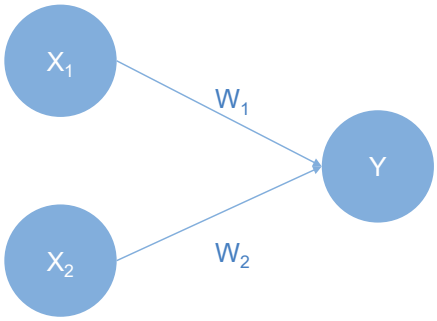
図の◯はニューロンと呼びます。記号の意味は下記の通りです。

X_1、X_2　：入力信号
Y　：出力信号
W_1、W_2　：重み

入力信号X_iに重みW_iが乗算され、その総和が一定の閾値θより大きくなった場合、値1を出力します。（それ以外は0を出力します。）
1を出力した場合「ニューロンが発火する」と表現することがあります。

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンの計算例を示します。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

各数値が

入力信号：$X_1 = X_2 = 1.0$
重み　：$W_1 = W_2 = 0.5$
閾値　：$\theta = 0.7$

の場合、出力信号Yは下記のように計算できます。

$1.0 \times 0.5 + 1.0 \times 0.5 = 1.0$
閾値0.7を超えているため、$Y = 1.0$

単純パーセプトロンの限界

- 入力層と出力層のみの2層からなる単純パーセプトロン (Simple perceptron) は線形非分離な問題を解けないことが、マービン・ミンスキーとシーモア・パパートによって1969年に指摘されました。
- 単純パーセプトロンが解決できない線形非分離な問題について、論理演算を例として次ページ以降で説明します。

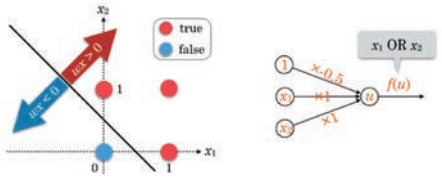
単純パーセプトロンの限界：論理演算とは

- コンピュータの演算には+、－、×、÷の四則演算のほかに論理演算があります。論理演算とは2つ以上の１または０入力値に対して、１つの演算結果（１または０）を出力する演算のことです。
- 入力値が2つの場合の論理演算のうち、論理積（AND）、論理和（OR）、否定論理積（NAND）、排他的論理和（XOR）を例に取り単純パーセプトロンの挙動を説明します。（上記以外の論理演算の種類については「https://ja.wikipedia.org/wiki/論理演算」を参照ください。）

単純パーセプトロンの限界：論理和（OR）

- 入力値X1とX2のどちらかがTRUE（=1）であれば、出力（X1 OR X2）がTRUEとなる演算を論理和と言います（左表を参照のこと）。入力値がX1とX2の2つの場合、論理和演算が対象とする組み合わせは4つです。
- 傾きが-1で切片が0.5の直線（X2 = - X1 + 0.5）を引くことができれば、4つの組み合わせの結果を直線で分割できることがわかります。

X1	X2	X1 OR X2
1	1	1
1	0	1
0	1	1
0	0	0

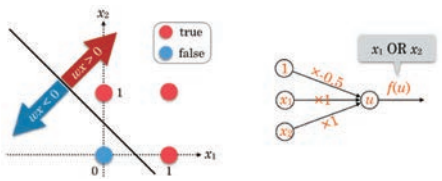


出展： http://hokuts.com/2015/12/04/ml3-mlp/

演習1：単純パーセプトロンの実装（論理和）

- 論理和が計算できるアルゴリズムを実装してください。

X1	X2	X1 OR X2
1	1	1
1	0	1
0	1	1
0	0	0

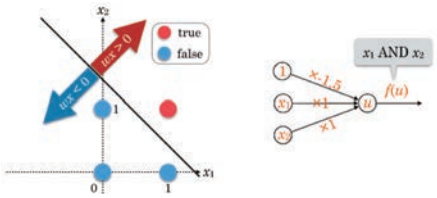


出展： http://hokuts.com/2015/12/04/ml3-mlp/

単純パーセプトロンの限界：論理積（AND）

- 入力値X1とX2のどちらもTRUE（=1）の場合のみ、出力（X1 AND X2）がTRUEとなる演算を論理積と言います（左表を参照のこと）。
- 例えば傾きが-1で切片が1.2の直線（X2 = - X1 + 1.2）を引くことができれば、4つの組み合わせの結果を直線で分割できることがわかります。

X1	X2	X1 AND X2
1	1	1
1	0	0
0	1	0
0	0	0

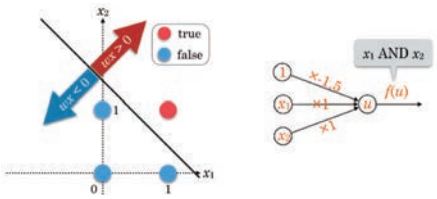


出展： http://hokuts.com/2015/12/04/ml3-mlp/

演習2：単純パーセプトロンの実装（論理積）

- 論理積が計算できるアルゴリズムを実装してください。

X1	X2	X1 AND X2
1	1	1
1	0	0
0	1	0
0	0	0

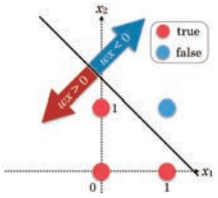


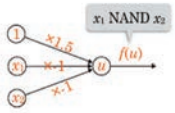
出展： http://hokuts.com/2015/12/04/ml3-mlp/

単純パーセプトロンの限界：否定論理積（NAND）

- 論理積（AND）を反転させた値が否定論理積（NAND）となります（左表を参照のこと）。

X1	X2	X1 NAND X2
1	1	0
1	0	1
0	1	1
0	0	1



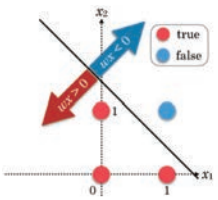


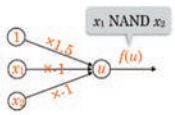
出展： http://hokuts.com/2015/12/04/ml3-mlp/

演習3：単純パーセプトロンの実装（否定論理積）

- 否定論理積が計算できるアルゴリズムを実装してください。

X1	X2	X1 NAND X2
1	1	0
1	0	1
0	1	1
0	0	1



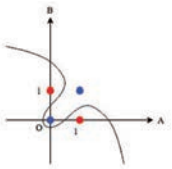


出展： http://hokuts.com/2015/12/04/ml3-mlp/

単純パーセプトロンの限界：排他的論理和（XOR）

- X1もしくはX2のいずれかがTRUEの場合のみ出力がTRUEとなり、X1とX2がともにTRUEの場合は出力がFALSEとなる値が排他的論理和（XOR）となります（左表を参照のこと）。
- 排他的論理和は線形関数で分離することが不可能です（右図を参照のこと）。この結果は、「単純パーセプトロン (Simple perceptron) は線形非分離な問題を解けない」ということを意味しています。

X1	X2	X1 XOR X2
1	1	0
1	0	1
0	1	1
0	0	0



出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

第4回：多層パーセプトロン

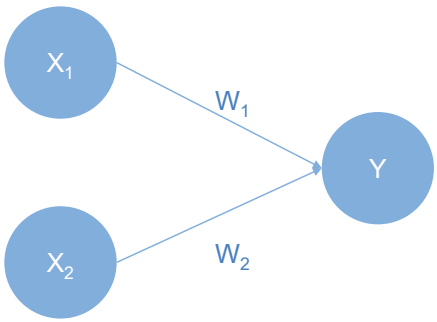
単純パーセプトロンの限界の突破

アジェンダ

- 単純パーセプトロンによる論理演算の実行と単純パーセプトロンの限界
- 多層パーセプトロンによる排他的論理和の実行

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンは、複数の入力値に対して1つの出力値を持つ関数です。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

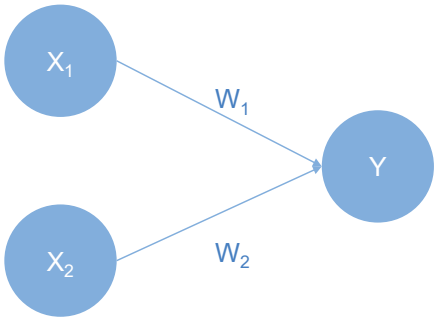
図の◯はニューロンと呼びます。記号の意味は下記の通りです。

X_1、X_2　：入力信号
Y　：出力信号
W_1、W_2　：重み

入力信号X_iに重みW_iが乗算され、その総和が一定の閾値θより大きくなった場合、値1を出力します。（それ以外は0を出力します。）
1を出力した場合「ニューロンが発火する」と表現することがあります。

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンの計算例を示します。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

各数値が

入力信号：$X_1 = X_2 = 1.0$
重み　：$W_1 = W_2 = 0.5$
閾値　：$\theta = 0.7$

の場合、出力信号Yは下記のように計算できます。

$1.0 \times 0.5 + 1.0 \times 0.5 = 1.0$
閾値0.7を超えているため、$Y = 1.0$

単純パーセプトロンの限界

- 入力層と出力層のみの2層からなる単純パーセプトロン (Simple perceptron) は線形非分離な問題を解けないことが、マービン・ミンスキーとシーモア・パパートによって1969年に指摘されました。
- 単純パーセプトロンが解決できない線形非分離な問題について、論理演算を例として次ページ以降で説明します。

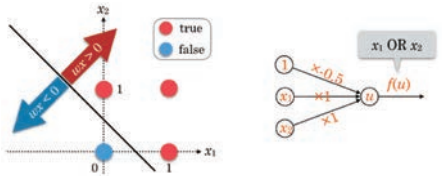
単純パーセプトロンの限界：論理演算とは

- コンピュータの演算には+、－、×、÷の四則演算のほかに論理演算があります。論理演算とは2つ以上の１または０入力値に対して、１つの演算結果（１または０）を出力する演算のことです。
- 入力値が2つの場合の論理演算のうち、論理積（AND）、論理和（OR）、否定論理積（NAND）、排他的論理和（XOR）を例に取り単純パーセプトロンの挙動を説明します。（上記以外の論理演算の種類については「https://ja.wikipedia.org/wiki/論理演算」を参照ください。）

単純パーセプトロンの限界：論理和（OR）

- 入力値X1とX2のどちらかがTRUE（=1）であれば、出力（X1 OR X2）がTRUEとなる演算を論理和と言います（左表を参照のこと）。入力値がX1とX2の2つの場合、論理和演算が対象とする組み合わせは4つです。
- 傾きが-1で切片が0.5の直線（X2 = - X1 + 0.5）を引くことができれば、4つの組み合わせの結果を直線で分割できることがわかります。

X1	X2	X1 OR X2
1	1	1
1	0	1
0	1	1
0	0	0

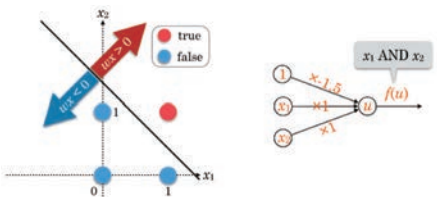


出展： http://hokuts.com/2015/12/04/ml3-mlp/

単純パーセプトロンの限界：論理積（AND）

- 入力値X1とX2のどちらもTRUE（=1）の場合のみ、出力（X1 AND X2）がTRUEとなる演算を論理積と言います（左表を参照のこと）。
- 例えば傾きが-1で切片が1.2の直線（X2 = - X1 + 1.2）を引くことができれば、4つの組み合わせの結果を直線で分割できることがわかります。

X1	X2	X1 AND X2
1	1	1
1	0	0
0	1	0
0	0	0

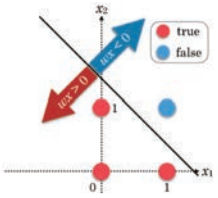


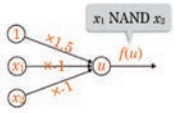
出展： http://hokuts.com/2015/12/04/ml3-mlp/

単純パーセプトロンの限界：否定論理積（NAND）

- 論理積（AND）を反転させた値が否定論理積（NAND）となります（左表を参照のこと）。

X1	X2	X1 NAND X2
1	1	0
1	0	1
0	1	1
0	0	1



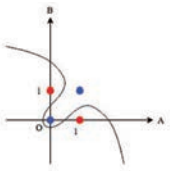


出展： http://hokuts.com/2015/12/04/ml3-mlp/

単純パーセプトロンの限界：排他的論理和（XOR）

- X1もしくはX2のいずれかがTRUEの場合のみ出力がTRUEとなり、X1とX2がともにTRUEの場合は出力がFALSEとなる値が排他的論理和（XOR）となります（左表を参照のこと）。
- 排他的論理和は線形関数で分離することが不可能です（右図を参照のこと）。この結果は、「単純パーセプトロン(Simple perceptron) は線形非分離な問題を解けない」ということを意味しています。

X1	X2	X1 XOR X2
1	1	0
1	0	1
0	1	1
0	0	0

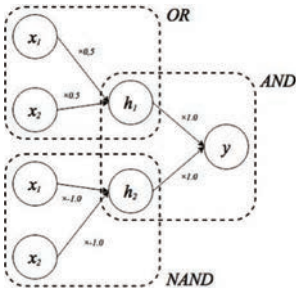


出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数を考えてみる。
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))

入力値の組み合わせ	X1	X2	X1 XOR X2
Ⅰ	1	1	0
Ⅱ	1	0	1
Ⅲ	0	1	1
Ⅳ	0	0	0

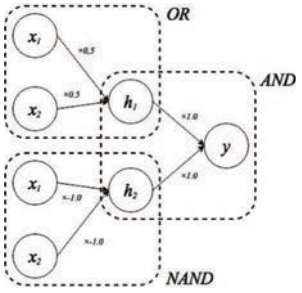


出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数を考えてみる。
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))
組み合わせⅠを考える。
OR　　：x1＝1、x2＝1なので、h1＝1
NAND　：x1＝1、x2＝1なので、h2＝0
AND　　：h1＝1、h2＝0なので、y＝0
組み合わせⅠを解決できていることが確認できる。

入力値の組み合わせ	X1	X2	X1 XOR X2
Ⅰ	1	1	0
Ⅱ	1	0	1
Ⅲ	0	1	1
Ⅳ	0	0	0

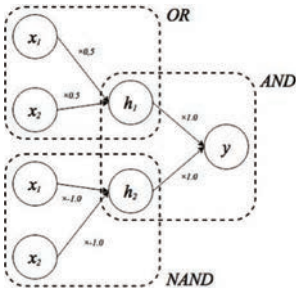


出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数を考えてみる。
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))
組み合わせⅡを考える。
OR　：x1＝1、x2＝0なので、h1＝1
NAND　：x1＝1、x2＝0なので、h2＝1
AND　：h1＝1、h2＝1なので、y＝1
組み合わせⅡを解決できていることが確認できる。

入力値の組み合わせ	X1	X2	X1 XOR X2
Ⅰ	1	1	0
Ⅱ	1	0	1
Ⅲ	0	1	1
Ⅳ	0	0	0

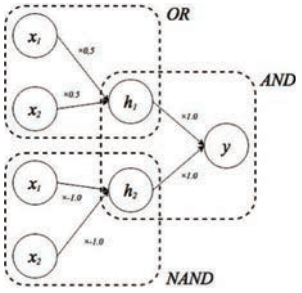


出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数を考えてみる。
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))
組み合わせⅢを考える。
OR　：x1＝0、x2＝1なので、h1＝1
NAND　：x1＝0、x2＝1なので、h2＝1
AND　：h1＝1、h2＝1なので、y＝1
組み合わせⅢを解決できていることが確認できる。

入力値の組み合わせ	X1	X2	X1 XOR X2
Ⅰ	1	1	0
Ⅱ	1	0	1
Ⅲ	0	1	1
Ⅳ	0	0	0

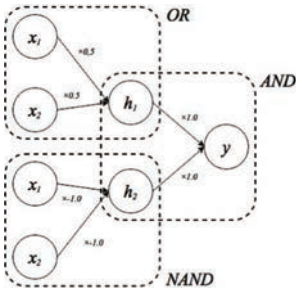


出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数を考えてみる。
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))
組み合わせⅣを考える。
OR　：x1＝0、x2＝0なので、h1＝0
NAND　：x1＝0、x2＝0なので、h2＝1
AND　：h1＝0、h2＝1なので、y＝0
組み合わせⅣを解決できていることが確認できる。

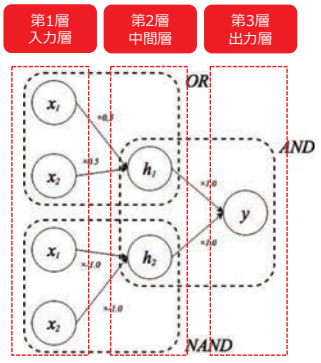
入力値の組み合わせ	X1	X2	X1 XOR X2
Ⅰ	1	1	0
Ⅱ	1	0	1
Ⅲ	0	1	1
Ⅳ	0	0	0



出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))
は、多層（この場合は3層）パーセプトロンである。
多層パーセプトロンの重みを正しく学習させることができれば、線形非分離な問題に対応可能となることが確認できる。



出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

演習1：単純パーセプトロンの復習

- 論理和（OR）、論理積（AND）、否定論理積（AND）が計算できるパーセプトロンの動きを再確認してください。

演習2：多層パーセプトロンの復習

- 論理和（OR）、論理積（AND）、否定論理積（AND）を組み合わせて排他的論理和（XOR）が計算できるパーセプトロンを実装してください。

第5回：ニューラルネットワーク

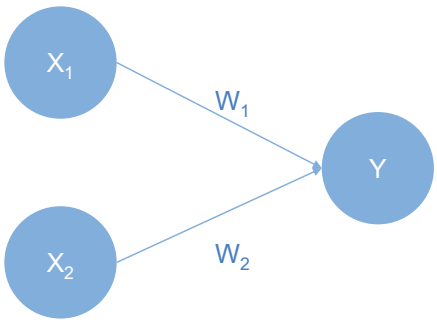
ニューラルネットワークの種類と構造

アジェンダ

- 単純パーセプトロン、3層パーセプトロンの振り返り
- 多層ニューラルネットワークとは
- 代表的なニューラルネットワークの構造
  - feed forward network（順伝播型/全結合型ニューラルネットワーク）
  - convolutional neural network（畳み込み型ニューラルネットワーク）
  - recurrent neural network（再帰型ニューラルネットワーク）
- その他のニューラルネットワーク

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンは、複数の入力値に対して1つの出力値を持つ関数です。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

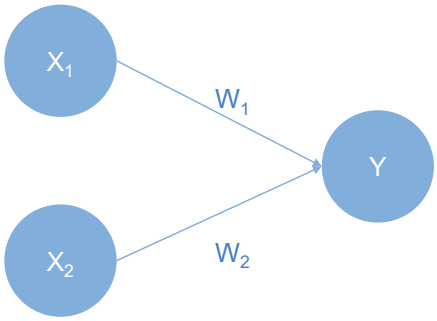
図の◯はニューロンと呼びます。記号の意味は下記の通りです。

X_1、X_2　：入力信号
Y　：出力信号
W_1、W_2　：重み

入力信号X_iに重みW_iが乗算され、その総和が一定の閾値θより大きくなった場合、値1を出力します。（それ以外は0を出力します。）
1を出力した場合「ニューロンが発火する」と表現することがあります。

パーセプトロンのアルゴリズム

- 最もシンプルなパーセプトロンの計算例を示します。



$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_i x_i \leqq \theta) \\ 1 & (\sum_{i=1}^{n} w_i x_i > \theta) \end{cases}$$

各数値が

入力信号：$X_1 = X_2 = 1.0$
重み　：$W_1 = W_2 = 0.5$
閾値　：$\theta = 0.7$

の場合、出力信号Yは下記のように計算できます。

$1.0 \times 0.5 + 1.0 \times 0.5 = 1.0$
閾値0.7を超えているため、$Y = 1.0$

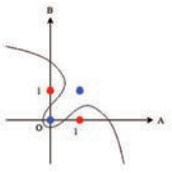
単純パーセプトロンの限界

- 入力層と出力層のみの2層からなる単純パーセプトロン (Simple perceptron) は線形非分離な問題を解けないことが、マービン・ミンスキーとシーモア・パパートによって1969年に指摘されました。
- 単純パーセプトロンが解決できない線形非分離な問題について、論理演算を例として次ページ以降で説明します。

単純パーセプトロンの限界：排他的論理和（XOR）

- X1もしくはX2のいずれかがTRUEの場合のみ出力がTRUEとなり、X1とX2がともにTRUEの場合は出力がFALSEとなる値が排他的論理和（XOR）となります（左表を参照のこと）。
- 排他的論理和は線形関数で分離することが不可能です（右図を参照のこと）。この結果は、「単純パーセプトロン (Simple perceptron) は線形非分離な問題を解けない」ということを意味しています。

X1	X2	X1 XOR X2
1	1	0
1	0	1
0	1	1
0	0	0

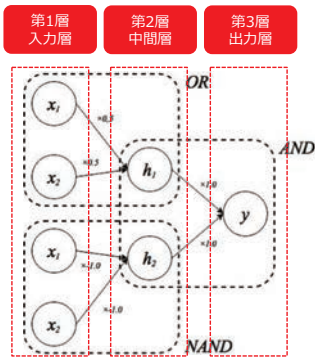


出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

排他的論理和（XOR）をパーセプトロンで解決するには

論理和（OR）、否定論理積（NAND）、論理積（AND）を組み合わせた下記の関数
XOR(X1, X2) = AND(OR(X1, X2), NAND(X1, X2))
は、多層（この場合は3層）パーセプトロンである。
多層パーセプトロンの重みを正しく学習させることができれば、線形非分離な問題に対応可能となることが確認できる。

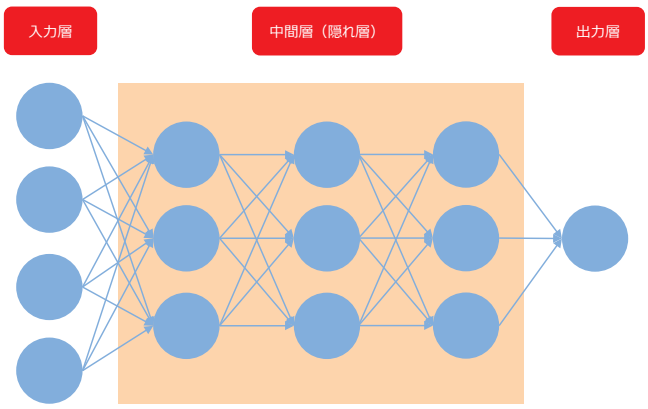
入力値の組み合わせ	X1	X2	X1 XOR X2
Ⅰ	1	1	0
Ⅱ	1	0	1
Ⅲ	0	1	1
Ⅳ	0	0	0



出展：
http://tuz.hatenablog.com/entry/2017/06/29/194801

多層ニューラルネットワークとは

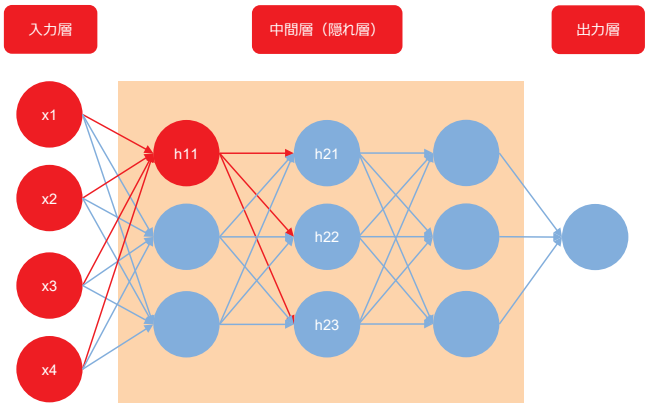
- 中間層が1層の多層パーセプトロンでは排他的論理和（XOR）が解決できることを学んできました。中間層の数を増やすことによって、ニューラルネットワークの表現力は向上していきます。
- 一般的に、中間層2層以上のニューラルネットワークを多層ニューラルネットワーク（ディープラーニング）と言います。



feed forward network
（順伝播型/全結合型ニューラルネットワーク）

feed forward network
（順伝播型/全結合型ニューラルネットワーク）

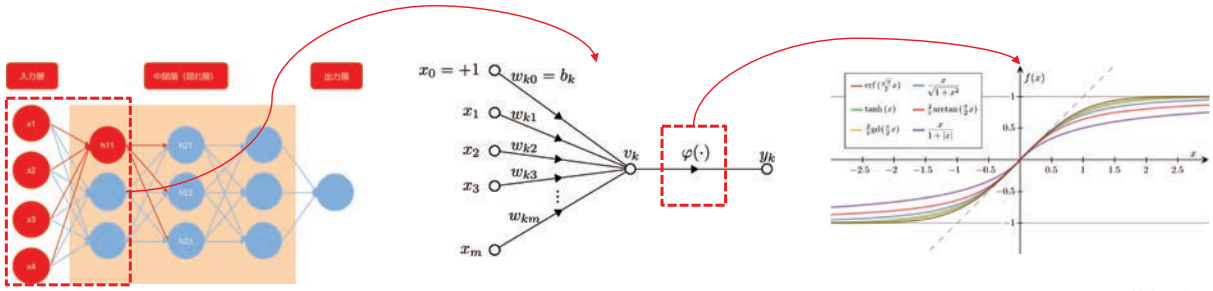
- 中間層のノードh_iは、1つ前の層の全ノードx_iと重みw_iで結合しています。
- x_iからの入力を受けたh_{11}は、h_{2i}へ値を出力します。



feed forward network
（順伝播型/全結合型ニューラルネットワーク）

h_{11}がh_{2i}に出力する値は、下記のようにして算出します。

1. $x_i \times w_i$を足し合わせたu_kを計算します（真ん中の図）。
2. u_kを活性化関数Φに入力しy_kを算出します（右の図）。

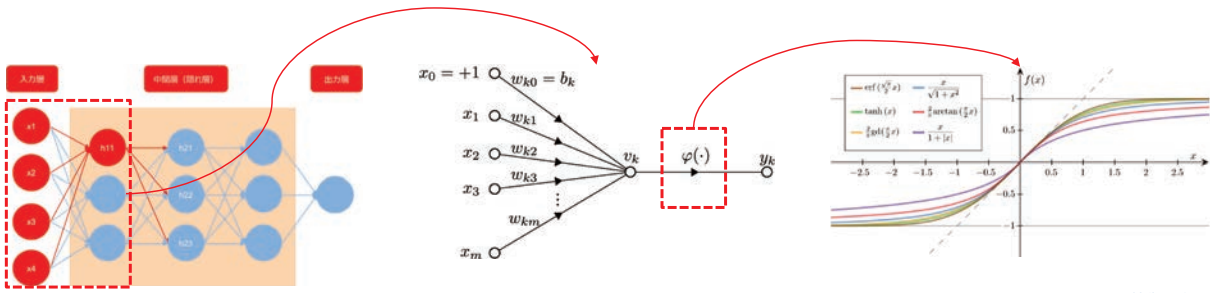


出展： https://ja.wikipedia.org/wiki/活性化関数

feed forward network
（順伝播型/全結合型ニューラルネットワーク）

質問

順伝播型ニューラルネットワークを構築するには、どの様なパラメータを設定する必要があるのか列挙してください。

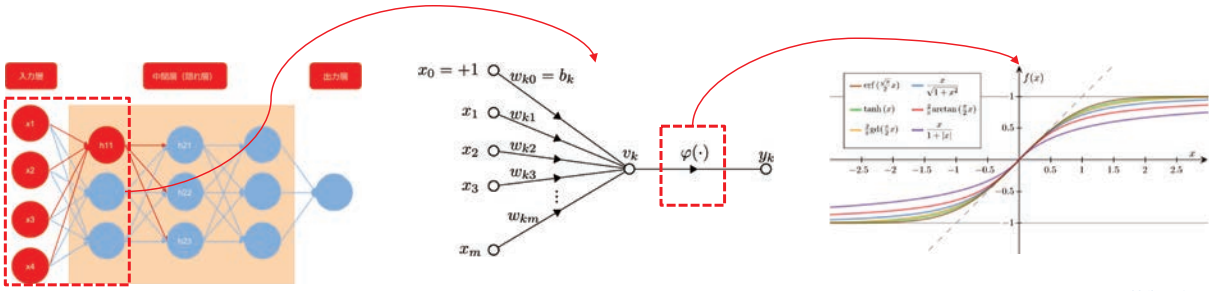


出展： https://ja.wikipedia.org/wiki/活性化関数

feed forward network
（順伝播型/全結合型ニューラルネットワーク）

順伝播型ニューラルネットワークには下記の様なパラメータがあります。

- 入力層、出力層のノードの数
- 中間層の層数、各中間層のノードの数
- 入力層-中間層、中間層-中間層、中間層-出力層間の結合重み、バイアス
- 中間層、出力層の活性化関数の種類



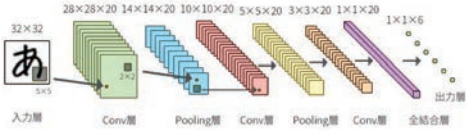
出展： https://ja.wikipedia.org/wiki/活性化関数

convolutional neural network
（畳み込み型ニューラルネットワーク）

convolutional neural network
（畳み込み型ニューラルネットワーク）

ロボティクスの分野において「コンピュータビジョン（環境を認識するための視覚をはじめとする各種センサーの開発とセンサー情報の処理手法」という分野が存在し自動運転技術、スマートフォン上の顔認識ソフトウェア、画像認識アプリケーションに至るまで、様々な会社や組織においてコンピュータービジョンは活躍しています。

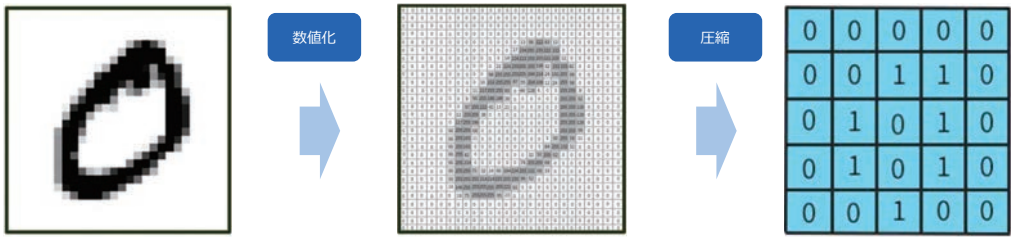
コンピュータビジョンの分野におけるコンペティションImageNetにおいて、Alex Krizhevskyのチームが2012年に高精度を達成して圧勝した手法が畳み込みニューラルネットワークです。



出展： https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

convolutional neural network
（畳み込み型ニューラルネットワーク）

- 畳み込み型ニューラルネットワークをはじめ、機械学習器で画像を扱うためには、画像を数値化する必要があります。
- 最もシンプルなグレースケールで数値化すると色が最も濃い場所を「255」、何も記載されていない場所は「0」と表現できます（真ん中の図）。
- マスの数が多すぎる場合には計算量が大きくなるため、情報量を圧縮することもあります（右図）。

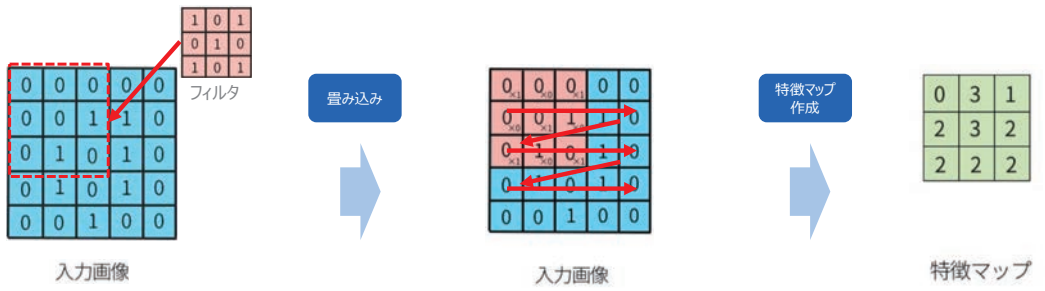


出展： https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

convolutional neural network （畳み込み型ニューラルネットワーク）

機械学習器の性能を上げるためには、学習データから「特徴」を抽出/作成する必要があります。
畳み込み型ニューラルネットワークでは、「畳み込み」という手法で特徴を作成します。

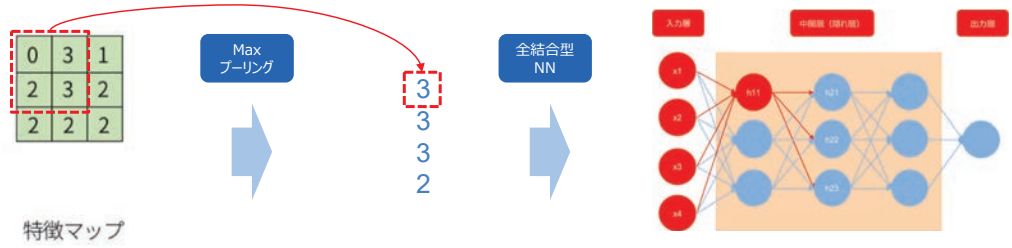
1. 入力画像の左上にフィルタをかけ合わせ、数値を算出します。下図の例では0が計算されます。
2. 右に1マスずらし、同様の計算を実行します。下図の例では3が算出されます。
3. 同様の計算を繰り返し、特徴マップを作成します。



出展： https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

convolutional neural network （畳み込み型ニューラルネットワーク）

- 畳み込みによる特徴マップの作成後、プーリングによるデータを圧縮します。下図の場合、小領域内の最大値を取るマックスプーリングを実施しています。
- プーリング実施後、全結合型ニューラルネットワークによる教師あり学習を実施します。

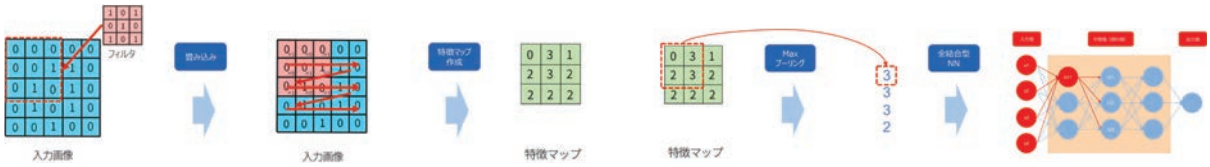


出展： https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

convolutional neural network
（畳み込み型ニューラルネットワーク）

質問

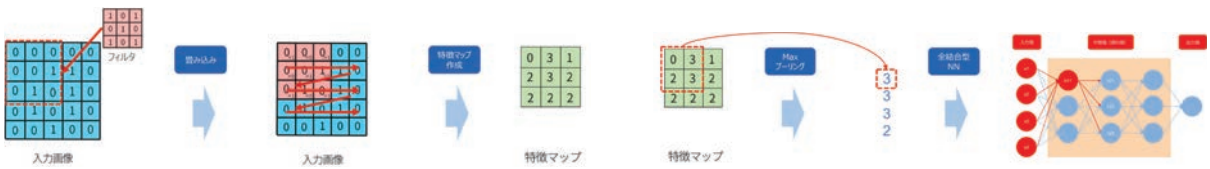
畳み込み型ニューラルネットワークを構築するには、どの様なパラメータを設定する必要があるのか列挙してください。



convolutional neural network
（畳み込み型ニューラルネットワーク）

畳み込み型ニューラルネットワークには下記の様なパラメータがあります。

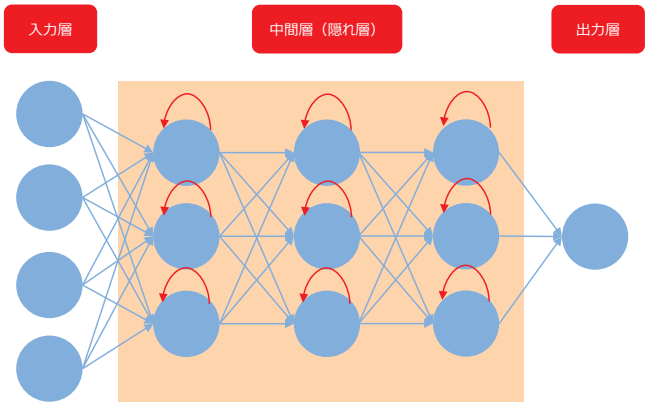
- 畳み込み、プーリングの回数（層の数）
- 畳み込み、プーリングフィルタのサイズと関数
- 全結合部の層数、各中間層のノードの数
- 全結合部の中間層-中間層、中間層-出力層間の結合重み、バイアス
- 全結合部の中間層、出力層の活性化関数の種類



recurrent neural network
（再帰型ニューラルネットワーク）

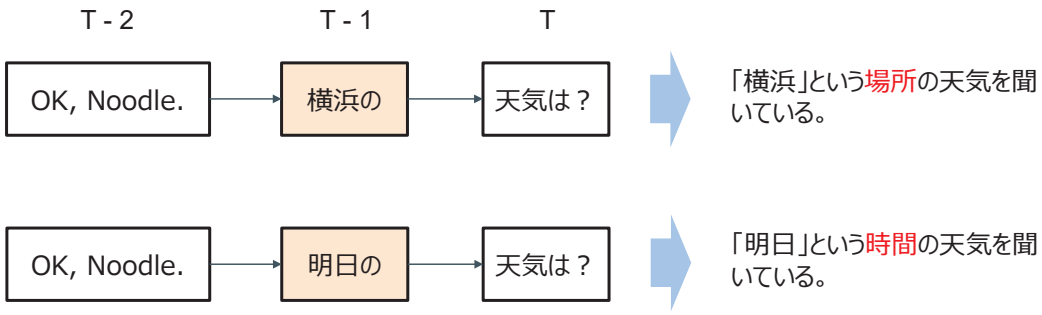
recurrent neural network
（再帰型ニューラルネットワーク）

- 全結合型ニューラルネットワークと似た形をしていますが、中間層に過去の自身の出力が再び入力されている点が異なります。
- ある時点の出力は、過去の入出力の遷移に依存するという時系列データを取り扱うことができます。



recurrent neural network （再帰型ニューラルネットワーク）

- 時点Tにおいて、質問の意図が場所に関するものなのか、時間に関するものなのかを判断するためには、前の時点T - 1、T - 2、、の内容を加味しなければならない。



recurrent neural network （再帰型ニューラルネットワーク）

質問

「自然言語処理における質問の分類問題」以外に、再帰型ニューラルネットワークが適用できそうな事例を挙げてください。

recurrent neural network （再帰型ニューラルネットワーク）

「自然言語処理における質問の分類問題」以外に、再帰型ニューラルネットワークが適用されているのは下記の様な分野です。

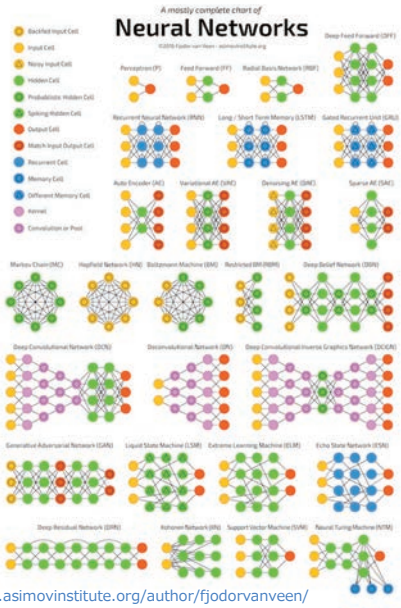
- 機械翻訳
- 音声認識、音声合成
- 文章生成
- 短期為替変動予測、トレンド予測

※いずれの分野にも共通するのは、ある時点の予測を行うためには、過去の遷移を加味しなければならないという時系列データを扱っているという点です。

その他のニューラルネットワーク

その他のニューラルネットワーク

- 本資料では代表的なニューラルネットワークである順伝播型（全結合型）、畳込み型、再帰型のニューラルネットワークを取り扱いました。
- その他にも主に事前学習で使用するオートエンコーダやボルツマンマシン、生成モデルであるGANなど、2018年末時点で多種多様なニューラルネットが考案されています。
- 機械学習の分野は日進月歩で新しいアイディアが生み出されており、更に踏み込んだ学習をしたい方は論文やwebサイトの解説記事を検索してみてください。



出展： https://www.asimovinstitute.org/author/fjodorvanveen/

第6回：活性化関数

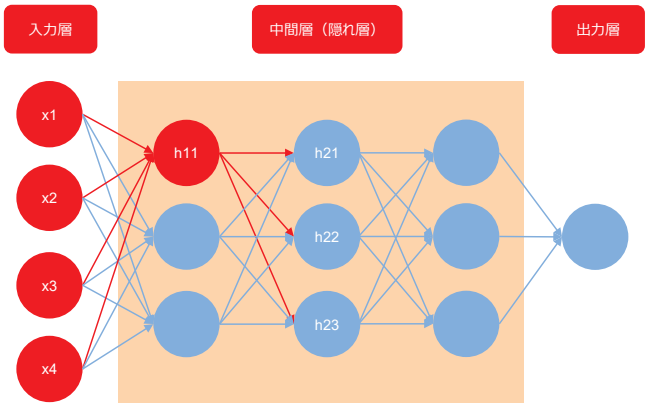
活性化関数の種類と挙動

アジェンダ

- feed forward network（順伝播型/全結合型ニューラルネットワーク）の振り返り
- 活性化関数とは
- 代表的な活性化関数とその挙動

feed forward network
（順伝播型/全結合型ニューラルネットワーク）

- 中間層のノードh_iは、1つ前の層の全ノードx_iと重みw_iで結合しています。
- x_iからの入力を受けたh_{11}は、h_{2i}へ値を出力します。

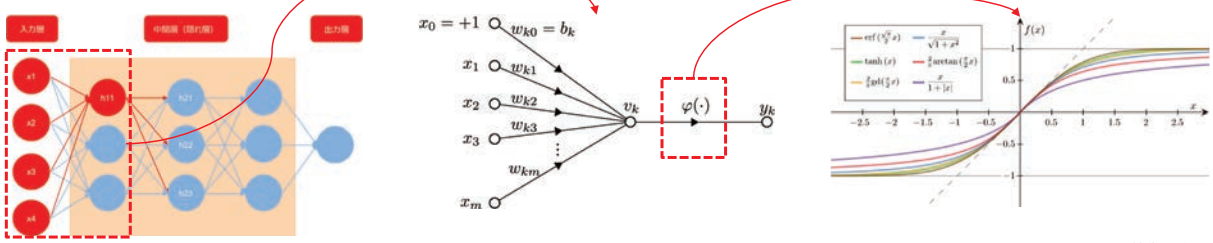


feed forward network
（順伝播型/全結合型ニューラルネットワーク）

h_{11}がh_{2i}に出力する値は、下記のようにして算出します。

1. $x_i \times w_i$を足し合わせたu_kを計算します（真ん中の図）。
2. u_kを活性化関数Φに入力しy_kを算出します（右の図）。

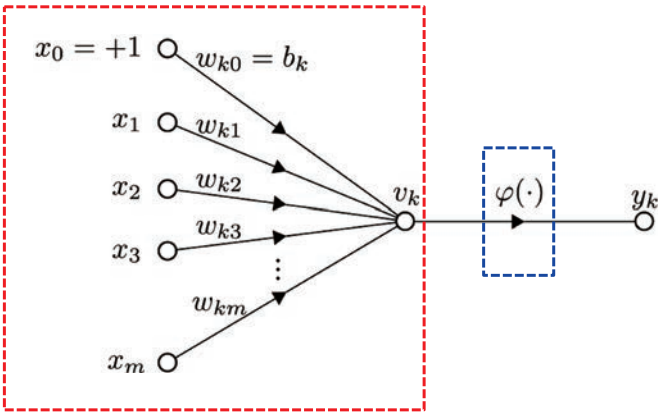
本資料の学習対象



出展： https://ja.wikipedia.org/wiki/活性化関数

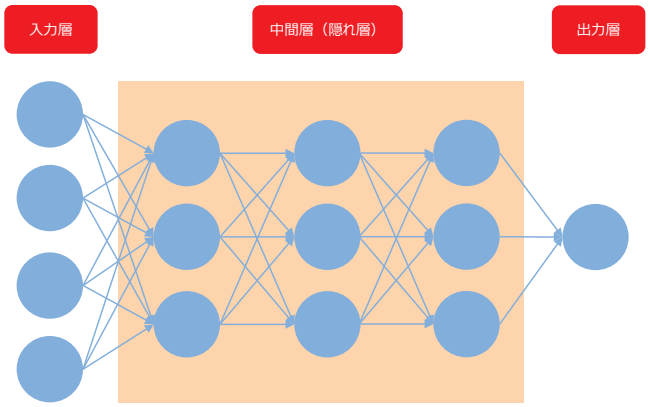
活性化関数とは

- 活性化関数Φは、入力信号の総和u_kがどのように活性化するか（次の層に渡す値y_k）を決定します。



活性化関数とは

- 主に中間層で使用される活性化関数としてステップ関数、シグモイド関数、ReLU関数などがあります。
- 出力層で使用される活性化関数として恒等写像関数、ソフトマックス関数があります。

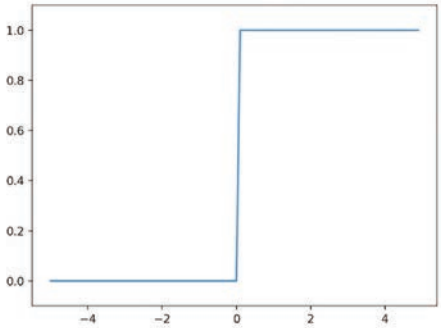


中間層で使用される活性化関数

ステップ関数

- 閾値を境にして出力が切り替わる関数で、「階段関数」とも呼ばれます。
- 一般的に単純パーセプトロンで使われる関数です。

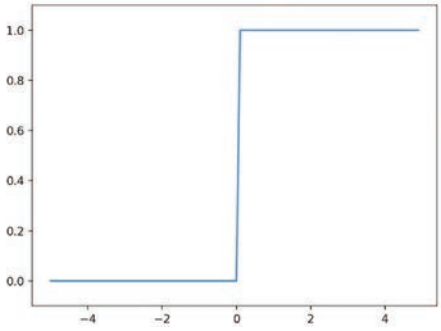
$$y = H(u-b) = \begin{cases} 1 & \text{if } u \geq b \\ 0 & \text{if } u < b \end{cases}$$



演習1：ステップ関数の実装

- ステップ関数を実装し、入力値に対する挙動を確認してください。

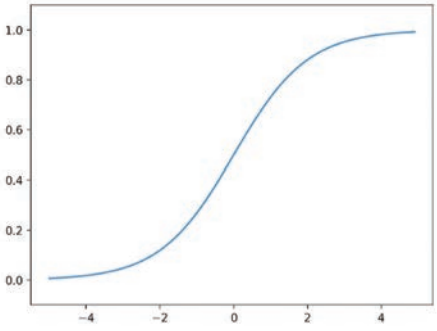
$$y = H(u-b) = \begin{cases} 1 & \text{if } u \geq b \\ 0 & \text{if } u < b \end{cases}$$



シグモイド関数

- シグモイド関数は微分の計算が容易であることから、バックプロパゲーションを伴うニューラルネットワークで使われています。ネットワークを数学的に扱うことが容易になるため、シミュレーションの計算負荷を減らしたいと思っていた初期の研究者がシグモイド関数をこぞって採用しました。

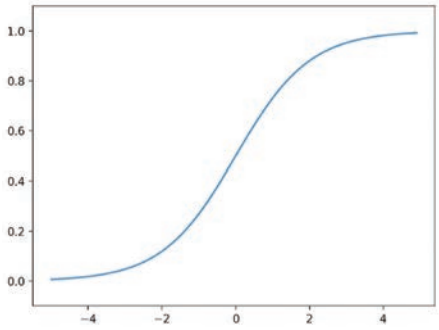
$$\varphi(x) = \varsigma_1(x) = \frac{1}{1+e^{-x}} = \frac{\tanh(x/2)+1}{2}$$



演習2：シグモイド関数の実装

- シグモイド関数を実装し、入力値に対する挙動を確認してください。

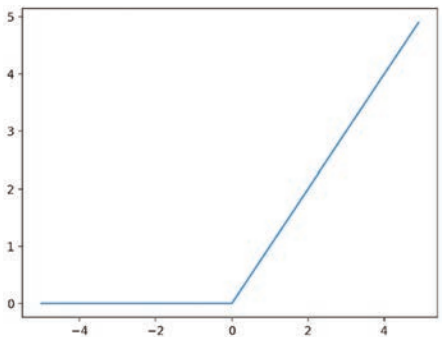
$$\varphi(x) = \varsigma_1(x) = \frac{1}{1+e^{-x}} = \frac{\tanh(x/2)+1}{2}$$



ReLU関数

- ReLUとはRectified Linear Unit（正規化線形関数）の略称です。
- 入力した値が0以下のとき0になり、1より大きいとき入力をそのまま出力します。

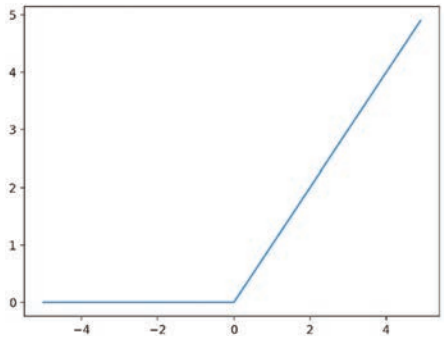
$$\varphi(x) = x_+ = \max(0, x)$$



演習3：ReLU関数の実装

- ReLU関数を実装し、入力値に対する挙動を確認してください。

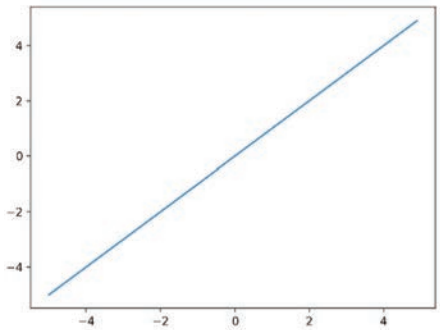
$$\varphi(x) = x_+ = \max(0, x)$$



出力層で使用される活性化関数

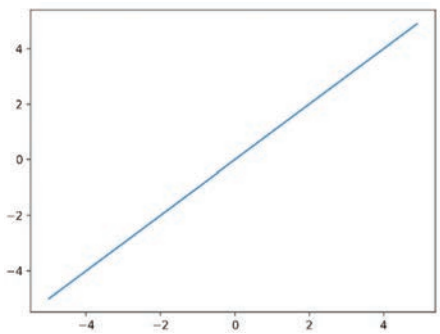
恒等写像関数

- 入力値と同じ値をそのまま出力する関数です。



演習4：恒等写像関数の実装

- 恒等写像関数を実装し、入力値に対する挙動を確認してください。



補足：恒等写像関数を中間層で使用しない理由

- 中間層の活性化関数に恒等写像関数（線形関数）を用いると、中間層の数を増やしてニューラルネットワークを深くする意味がなくなります。
- 線形関数を用いた中間層を多層重ねても、それと同じ計算が実行できる「中間層のないネットワーク」が存在するからです。
- 例えば、線形関数h(x)=cxを活性化関数として3層重ねるとy=h(h(h(x))と表現できますが、hは線形関数なので係数cを括りだせば、y=c*c*c*x(y=ax)というように中間層のないネットワークで表現できることになります。

ソフトマックス関数

- K 個のクラス分類で使用します。出力は K 個で、総和は 1 であり、そのクラスに所属する確率と解釈できます。

$$\varphi(u_k) = \frac{e^{u_k}}{\sum_{i=1}^{K} e^{u_i}}$$

演習5：ソフトマックス関数の実装

- ソフトマックス関数を実装し、入力値に対する挙動を確認してください。

$$\varphi(u_k) = \frac{e^{u_k}}{\sum_{i=1}^{K} e^{u_i}}$$

第7回：勾配降下法

勾配降下法の目的と種類

アジェンダ

- モデルの評価をどのようにおこなうか
  - 誤差関数の導出
  - モデルの予測値と実測値の誤差を最小化する方法

モデルの評価をどのようにおこなうか

モデルの評価を行うためには指標が必要です。その指標を誤差関数（コスト関数）と言います。
下図のように青い点を赤い直線で近似することを考えます。

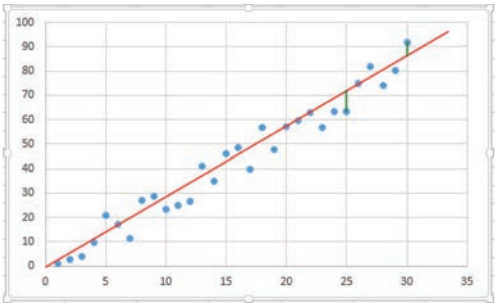
まず考えられるのが実測値（青い点）と予測線（赤い線）の差を取ることです。

$$J = y - f(x)$$

x=25ではJは-10くらいになります。
x=30ではJは+5くらいになります。
誤差を評価するのに符号は邪魔なので、2乗します。2乗すると、

$$J = (y - f(x))^2$$

x=25ではJは100くらいになります。
x=30ではJは25くらいになります。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

モデルの評価をどのようにおこなうか

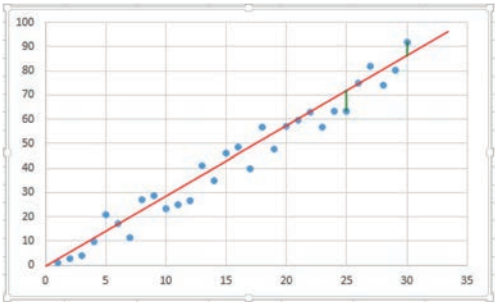
すべての点について誤差を足し合わせます。

$$J = \sum_{i=0}^{n} (y_n - f(x_n))^2$$

このままでも良いのですが、後ほど微分の話がでてくるため係数1/2を付与して、

$$J = \frac{1}{2} \sum_{i=0}^{n} (y_n - f(x_n))^2$$

とします。これを2乗誤差といいます。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

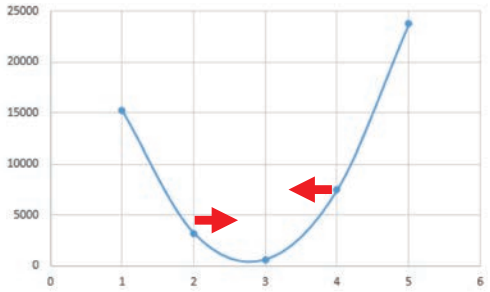
モデルの評価をどのようにおこなうか

誤差を最小にするためには、誤差関数の微分が0（傾きが0）になる点を探す必要があります。
予測線（前ページの赤い線）は原点を通る線なので、

$$f(x) = wx$$

で表すことができます。今回の例だとwが3より少し小さくなるようにチューニングすると、モデルの精度が良くなることがわかります。

チューニング中のある計算時点のとき、wが4だとすると、
wを小さくする方がモデルの精度が良くなり、
wが2だとすると、wを大きくする方がモデルの精度が
良くなります。



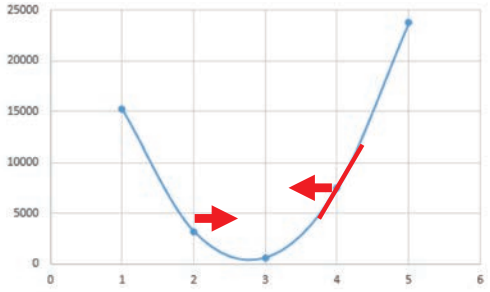
出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

モデルの評価をどのようにおこなうか

チューニング時の計算において、誤差関数の微分が正の値のときはwを小さくしたいので、

$$w = w - \rho \frac{\partial J}{\partial w}$$

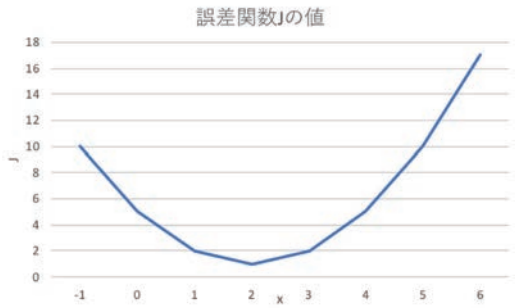
という式を考えます。このときρは学習率といい、更新が大きくなり過ぎることを防ぐ係数です。
このように誤差関数を微分して勾配を小さく（降下）する方法を
勾配降下法といいます。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

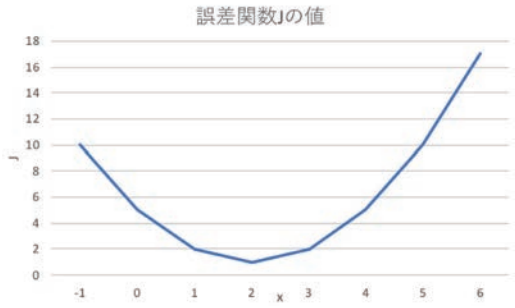
演習1：勾配降下法の実装

- 誤差関数J = x^2 - 4x + 5が最小となるxを求めるプログラムを実装してください。



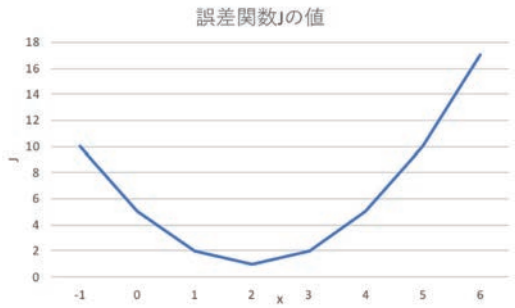
演習2：勾配降下法の実装

- 最適値よりも小さな値から次第に最適値に向けて収束する様子を観測してください。



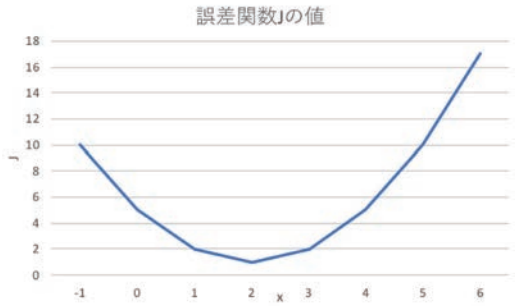
演習3：勾配降下法の実装

- 学習率を大きくし、収束計算が粗い間隔で実行されていることを観測してください。



演習4：勾配降下法の実装

- 学習率を更に大きくし、収束計算が発散することを観測してください。

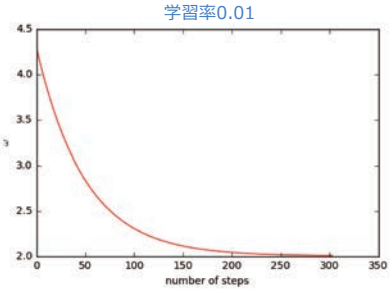


勾配降下法（学習）のテクニック

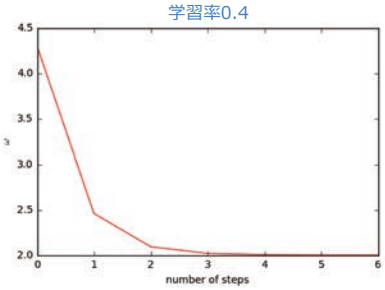
学習率の設定

- 学習率を大きくすることによって、少ない計算回数で最適値へ収束させることができます。
- しかし、学習率を大きくしすぎると、何度計算しても最適値に収束しない現象が起きてしまいます。
- まずは大きめな学習率からスタートし、徐々に学習率を小さくしていく方が、試行時間の短縮につながります。

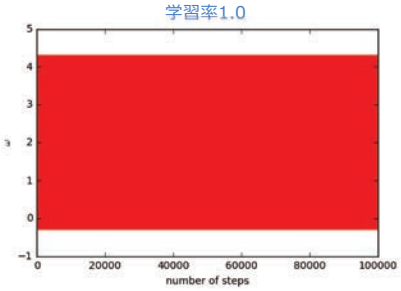
学習率0.01



学習率0.4



学習率1.0



第8回：確率的勾配降下法

勾配降下法の種類と確率的勾配降下法

アジェンダ

- 前回学習（勾配降下法）の復習
- 勾配降下法の種類
  - 最急降下法
  - 確率的勾配降下法
- 確率的勾配降下法

モデルの評価をどのようにおこなうか

モデルの評価を行うためには指標が必要です。その指標を誤差関数（コスト関数）と言います。
下図のように青い点を赤い直線で近似することを考えます。

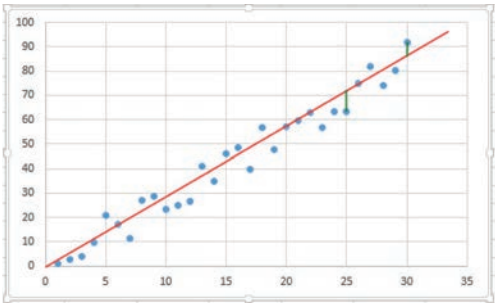
まず考えられるのが実測値（青い点）と予測線（赤い線）の差を取ることです。

$$J = y - f(x)$$

x=25ではJは-10くらいになります。
x=30ではJは+5くらいになります。
誤差を評価するのに符号は邪魔なので、2乗します。2乗すると、

$$J = (y - f(x))^2$$

x=25ではJは100くらいになります。
x=30ではJは25くらいになります。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

モデルの評価をどのようにおこなうか

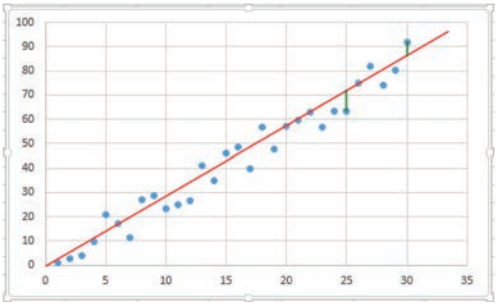
すべての点について誤差を足し合わせます。

$$J = \sum_{i=0}^{n} (y_n - f(x_n))^2$$

このままでも良いのですが、後ほど微分の話がでてくるため係数1/2を付与して、

$$J = \frac{1}{2} \sum_{i=0}^{n} (y_n - f(x_n))^2$$

とします。これを2乗誤差といいます。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

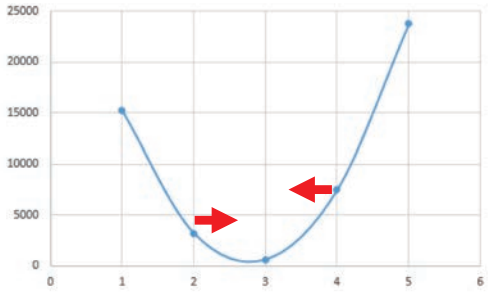
モデルの評価をどのようにおこなうか

誤差を最小にするためには、誤差関数の微分が0（傾きが0）になる点を探す必要があります。
予測線（前ページの赤い線）は原点を通る線なので、

$$f(x) = wx$$

で表すことができます。今回の例だとwが3より少し小さくなるようにチューニングすると、モデルの精度が良くなることがわかります。

チューニング中のある計算時点のとき、wが4だとすると、
wを小さくする方がモデルの精度が良くなり、
wが2だとすると、wを大きくする方がモデルの精度が
良くなります。



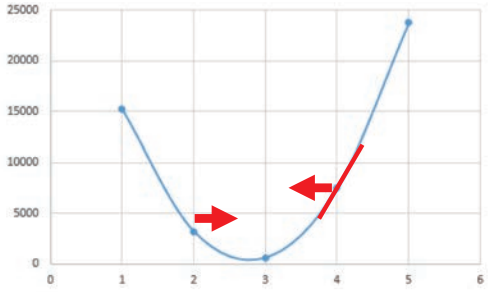
出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

モデルの評価をどのようにおこなうか

チューニング時の計算において、誤差関数の微分が正の値のときはwを小さくしたいので、

$$w = w - \rho \frac{\partial J}{\partial w}$$

という式を考えます。このときρは学習率といい、更新が大きくなり過ぎることを防ぐ係数です。
このように誤差関数を微分して勾配を小さく（降下）する方法を
勾配降下法といいます。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

演習1：勾配降下法の確認

- 前回学習した「勾配降下法」の挙動を確認してください。

勾配降下法の種類

最急降下法

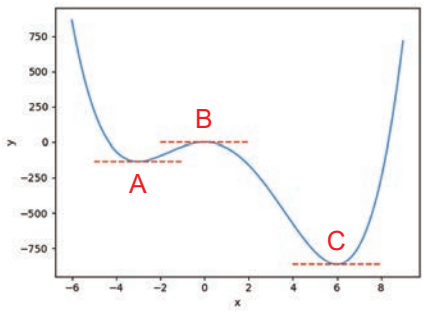
- すべての誤差の合計を計算した後、パラメータを更新します。
- すべての誤差の合計から計算するため、少量の外れ値の影響をあまり受けることなくパラメータを更新することができます。
- すべての誤差の合計から計算するため、計算量が大きくなってしまうというデメリットがあります。

確率的勾配降下法

- 学習データ群からランダムに1つを取り出して誤差を計算し、パラメータを更新します。
- 1回の計算に使用するデータ量が少ないので、計算が早くなります。
- 局所最適化に陥らず、全体最適化に辿り着く可能性が高くなります。
- 学習時にランダムに抽出したデータに外れ値が含まれていた場合、パラメータが外れ値に大きく引きずられてしまうというデメリットがあります。

局所最適化とは

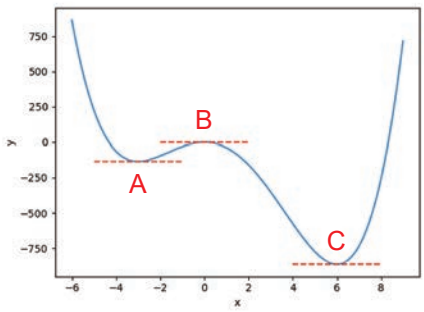
- 勾配降下法では、誤差関数の微分が0になる（傾きが0になる）ようにパラメータを更新します。
- X = -6から計算がスタートした場合、Aに計算が落ち着くことがあります。グラフ全体で見ればCが最も誤差が小さいのですが、勾配降下法のアルゴリズムではAが最適値だとみなしてしまいます。
- Aで計算が落ち着くことを局所最適、Cで計算が落ち着くことを大域的（全体）最適といいます。



出展： https://qiita.com/koshian2/items/028c457880c0ec576e27

演習2：局所最適化の確認

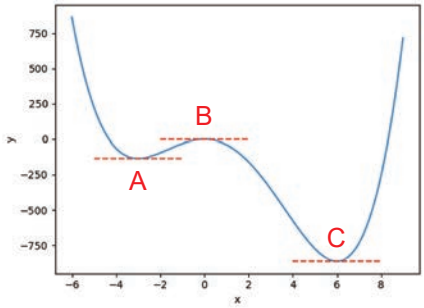
- X = -6から計算をスタートし、Aに収束する（局所最適に陥る）ことを確認してください。



出展： https://qiita.com/koshian2/items/028c457880c0ec576e27

演習3：局所最適化の確認

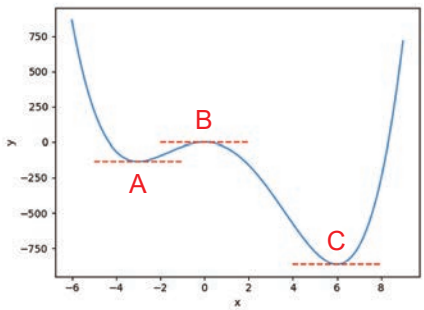
- 学習データをランダムに抽出する確率的勾配降下法を実装してください。
- X = -6から計算をスタートし、Aに収束する（局所最適に陥る）ことを確認してください。
- Xの初期値を2からスタートし、Cに収束することを確認してください。



出展： https://qiita.com/koshian2/items/028c457880c0ec576e27

演習4：学習データのランダムサンプリング

- 学習データの初期値をランダムに設定し、全体最適に収束する挙動を確認してください。

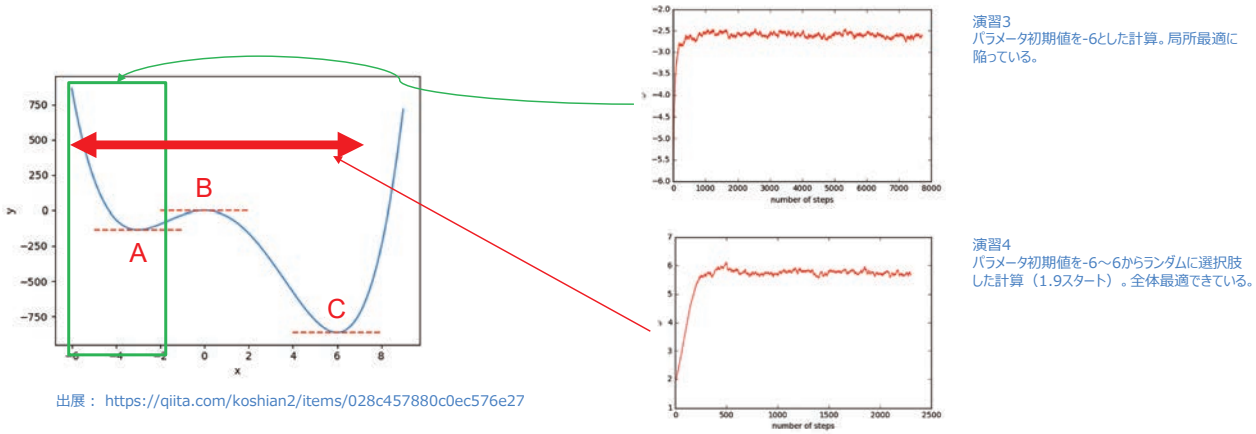


出展： https://qiita.com/koshian2/items/028c457880c0ec576e27

確率的勾配降下法（学習）のテクニック

学習データの抽出方法

- 学習データの全範囲からランダムにデータを抽出し学習することによって、局所最適を回避できる可能性が上がります（演習4）。このようにランダムに抽出したいくつかのデータを一塊として学習させる手法をミニバッチ学習と言います。



出展： https://qiita.com/koshian2/items/028c457880c0ec576e27

演習3
パラメータ初期値を-6とした計算。局所最適に陥っている。

演習4
パラメータ初期値を-6～6からランダムに選択肢した計算（1.9スタート）。全体最適できている。

第9回：誤差逆伝播法

多層ニューラルネットワークにおけるパラメータの最適化

アジェンダ

- 誤差関数の復習
- 誤差逆伝播法とは
  - ニューラルネットワークのパラメータ

誤差関数の復習

モデルの評価をどのようにおこなうか

モデルの評価を行うためには指標が必要です。その指標を誤差関数（コスト関数）と言います。
下図のように青い点を赤い直線で近似することを考えます。

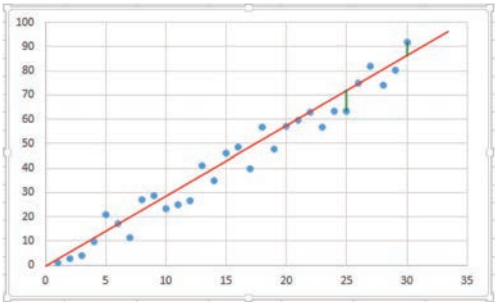
まず考えられるのが実測値（青い点）と予測線（赤い線）の差を取ることです。

$$J = y - f(x)$$

x=25ではJは-10くらいになります。
x=30ではJは+5くらいになります。
誤差を評価するのに符号は邪魔なので、2乗します。2乗すると、

$$J = (y - f(x))^2$$

x=25ではJは100くらいになります。
x=30ではJは25くらいになります。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

モデルの評価をどのようにおこなうか

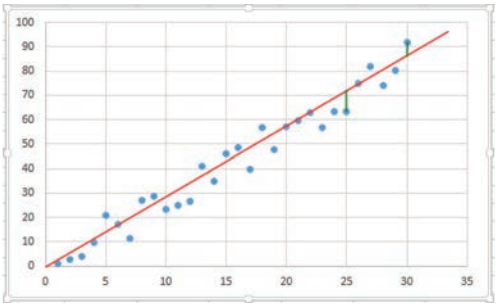
すべての点について誤差を足し合わせます。

$$J = \sum_{i=0}^{n} (y_n - f(x_n))^2$$

このままでも良いのですが、後ほど微分の話がでてくるため係数1/2を付与して、

$$J = \frac{1}{2} \sum_{i=0}^{n} (y_n - f(x_n))^2$$

とします。これを2乗誤差といいます。



出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

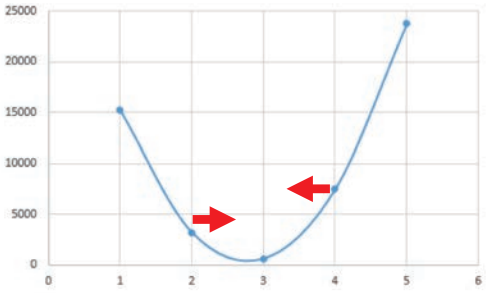
モデルの評価をどのようにおこなうか

誤差を最小にするためには、誤差関数の微分が0（傾きが0）になる点を探す必要があります。
予測線（前ページの赤い線）は原点を通る線なので、

$$f(x) = wx$$

で表すことができます。今回の例だとwが3より少し小さくなるようにチューニングすると、モデルの精度が良くなることがわかります。

チューニング中のある計算時点のとき、wが4だとすると、
wを小さくする方がモデルの精度が良くなり、
wが2だとすると、wを大きくする方がモデルの精度が
良くなります。



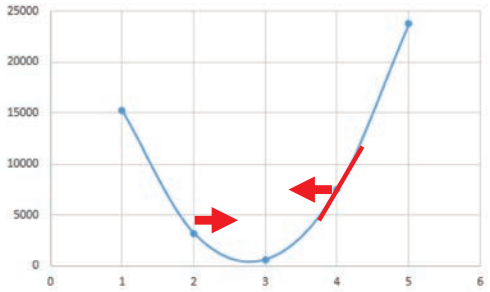
出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

モデルの評価をどのようにおこなうか

チューニング時の計算において、誤差関数の微分が正の値のときはwを小さくしたいので、

$$w = w - \rho\frac{\partial J}{\partial w}$$

という式を考えます。このときρは学習率といい、更新が大きくなり過ぎることを防ぐ係数です。
このように誤差関数を微分して勾配を小さく（降下）する方法を
勾配降下法といいます。

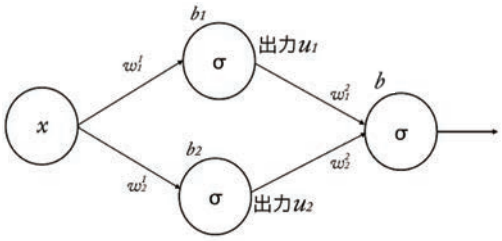


出展： https://shironeko.hateblo.jp/entry/2016/10/29/173634

誤差逆伝播法

パラメータの種類

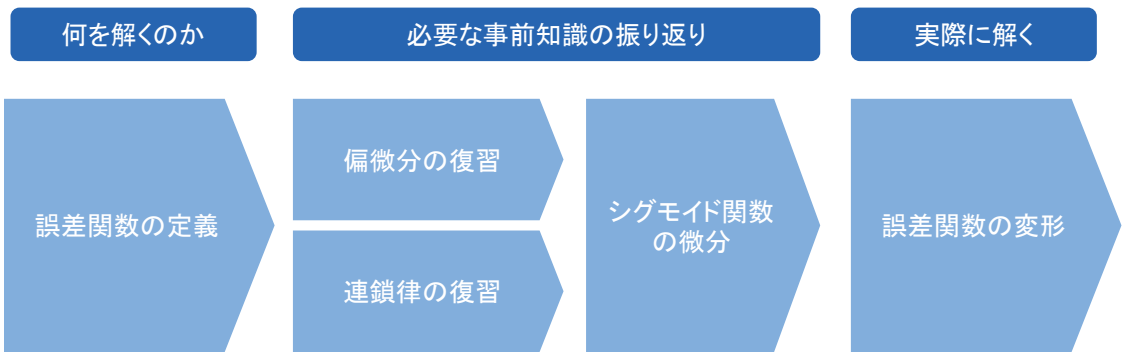
- バックプロパゲーションの基本的な仕組みを学習するため、中間層1層のシンプルなニューラルネットワークを考えます。
- 活性化関数は、計算のしやすさからシグモイド関数を考えます（その理由は、シグモイド関数を微分するときれいな形に変形でき、誤差逆伝播法の学習に都合が良いからです。詳細は後述します。）。
- 最適化したいパラメータは下記の通りです。
  入力層-中間層間：結合重みw^1_kとバイアスb_1、b_2
  中間層-出力層間：結合重みw^2_kとバイアスb



出展： https://www.yukisako.xyz/entry/backpropagation

誤差逆伝播法の理解までの手順

- 以下の手順で学習していきます。



誤差関数の更新

- 最適化したいパラメータは下記の通りです。
  入力層-中間層間：結合重みw^1_kとバイアスb_1、b_2
  中間層-出力層間：結合重みw^2_kとバイアスb

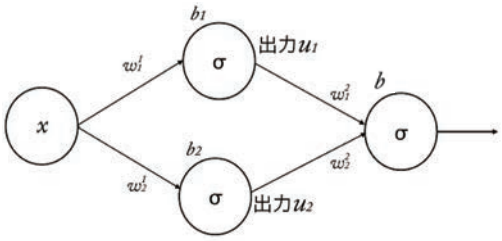
- 誤差関数J = E(w^1_k, b_1, b_2, w^2_k, b)だと定義すると、パラメータ更新は以下の式を更新することになります。

w_k^1を$w_k^1 - \alpha\frac{\partial E}{\partial w_k^1}$ に更新

w_k^2を$w_k^2 - \alpha\frac{\partial E}{\partial w_k^2}$ に更新

b_kを$b_k - \alpha\frac{\partial E}{\partial b_k}$ に更新

bを$b - \alpha\frac{\partial E}{\partial b}$ に更新



出展： https://www.yukisako.xyz/entry/backpropagation

偏微分とは

- 変数が2つある関数 z(x, y) = x^2+y^2 の偏微分は下記のようになります。

$\frac{\partial z}{\partial x} = 2x$

$\frac{\partial z}{\partial y} = 2y$

- 1つ目の式は、「xが変化するとzがどれくらい変動するか？」を意味しています。
- 2つ目の式は、「yが変化するとzがどれくらい変動するか？」を意味しています。

- このように、複数の変数のうち1つの変数だけで微分することを偏微分といいます。

連鎖律とは

- 合成関数の微分では，例えばyがuの関数で、かつuがxの関数であるとき、yをxで微分するには以下のようにします。

$$\frac{dy}{dx}=\frac{dy}{du}\frac{du}{dx}$$

- 同じようにzがuの関数で、かつuが(x, y)の関数である時，zをxもしくはzをyで偏微分するには以下のようにします。

$$\frac{\partial z}{\partial x}=\frac{\partial z}{\partial u}\frac{\partial u}{\partial x}$$
$$\frac{\partial z}{\partial y}=\frac{\partial z}{\partial u}\frac{\partial u}{\partial y}$$

シグモイド関数の微分

- 本教材では活性化関数にシグモイド関数を使用すると記載しました。その理由は、シグモイド関数を微分するときれいな形に変形でき、誤差逆伝播法の学習に都合が良いからです。
- シグモイド関数は下記の形をしています。

$$\frac{1}{1+e^{-x}}$$

- シグモイド関数を微分すると、下記のようになります。

$$\sigma(x)'=\left(\frac{1}{1+e^{-x}}\right)'=\frac{-(1+e^{-x})'}{(1+e^{-x})^2}=\frac{e^{-x}}{(1+e^{-x})^2}=\frac{e^{-x}}{(1+e^{-x})}\frac{1}{(1+e^{-x})}=\frac{1+e^{-x}-1}{(1+e^{-x})}\frac{1}{(1+e^{-x})}$$

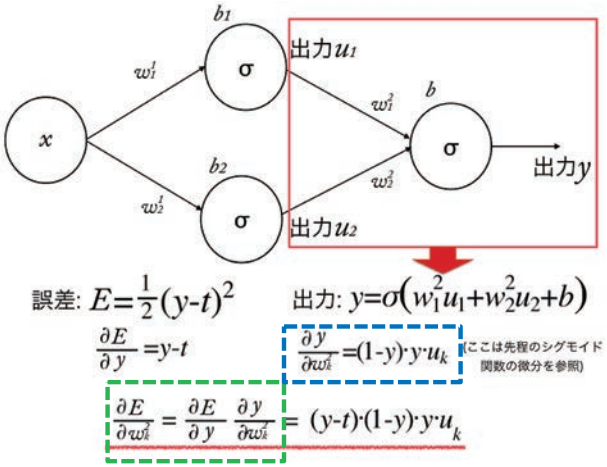
$$=\left(1-\frac{1}{(1+e^{-x})}\right)\frac{1}{(1+e^{-x})}=(1-\sigma(x))\sigma(x)$$

中間層-出力層間の重みw^2_kの更新

- W^2_kの誤差を勾配降下法で解くためには、下記の更新を実施する必要があります。

$$w_k^2 \text{を} w_k^2 - \alpha\frac{\partial E}{\partial w_k^2} \text{に更新}$$

- 誤差関数Eを重みw^2_kで微分している箇所は、連鎖律の知識を使って実行します。
- 次に、シグモイド関数の微分の知識を使って出力yを重みw^2_kで偏微分します。



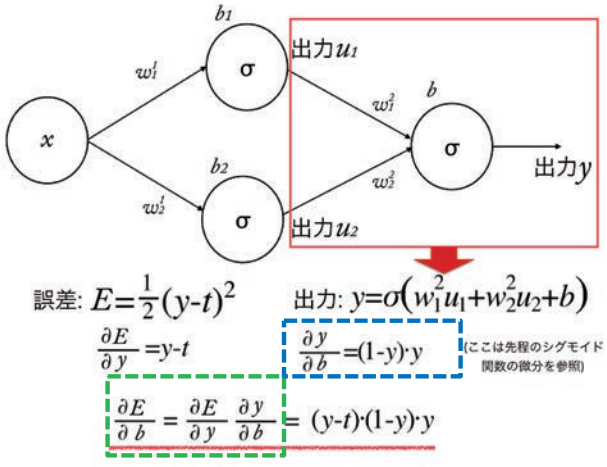
出展： https://www.yukisako.xyz/entry/backpropagation

中間層-出力層間のバイアスbの更新

- bの誤差を勾配降下法で解くためには、下記の更新を実施する必要があります。

$$b \text{を} b - \alpha\frac{\partial E}{\partial b} \text{に更新}$$

- 誤差関数Eをバイアスbで微分している箇所は、連鎖律の知識を使って実行します。
- 次に、シグモイド関数の微分の知識を使って出力yをバイアスbで偏微分します。



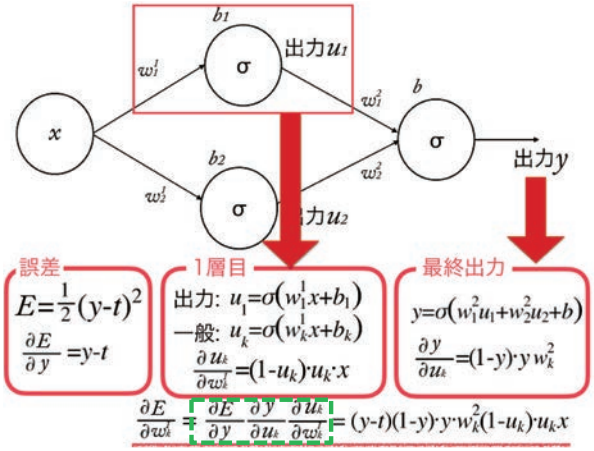
出展： https://www.yukisako.xyz/entry/backpropagation

入力層-中間層間の重みw^1_kの更新

- W^1_kの誤差を勾配降下法で解くためには、下記の更新を実施する必要があります。

$$w_k^1 \text{を} w_k^1 - \alpha \frac{\partial E}{\partial w_k^1} \text{に更新}$$

- 誤差関数Eを重みw^1_kで微分している箇所は、連鎖律の知識を使って実行します。
- 誤差関数Eを重みw^1_kで微分するためには、中間層-出力層側のw^2_kを使用する必要があります。まずw^2_kを計算してからw^1_kを計算する必要があるため、誤差の**逆伝播**という表現を使います。



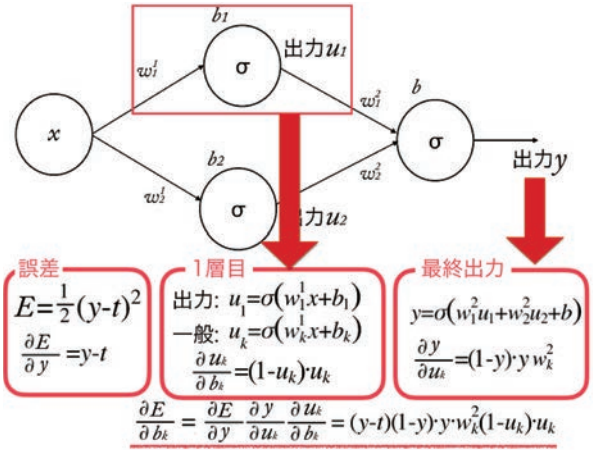
出展： https://www.yukisako.xyz/entry/backpropagation

入力層-中間層間のバイアスb_kの更新

- b_kの誤差を勾配降下法で解くためには、下記の更新を実施する必要があります。

$$b_k \text{を} b_k - \alpha \frac{\partial E}{\partial b_k} \text{に更新}$$

- 誤差関数Eをバイアスb_kで微分している箇所は、連鎖律の知識を使って実行します。
- バイアスの計算でも、同様に中間層-出力層側のw^2_kを使用する必要があります。



出展： https://www.yukisako.xyz/entry/backpropagation

誤差逆伝播法のまとめ

- ニューラルネットワークにおける学習とは、重みとバイアスを調整し、誤差関数J = E(w^1_k, b_1, b_2, w^2_k, b)を小さくしていくことです。
- 誤差関数を小さくするためには、下記の計算を繰り返し実施する必要があります。

w^1_kを$w^1_k - \alpha\frac{\partial E}{\partial w^1_k}$に更新　　$\frac{\partial E}{\partial w^1_k} = (y-t)(1-y)\cdot y\cdot w^2_k(1-u_k)\cdot u_k\cdot x$

w^2_kを$w^2_k - \alpha\frac{\partial E}{\partial w^2_k}$に更新　　$\frac{\partial E}{\partial w^2_k} = (y-t)\cdot(1-y)\cdot y\cdot u_k$

b_kを$b_k - \alpha\frac{\partial E}{\partial b_k}$に更新　　$\frac{\partial E}{\partial b_k} = (y-t)(1-y)\cdot y\cdot w^2_k(1-u_k)\cdot u_k$

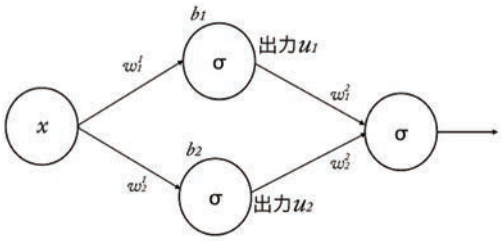
bを$b - \alpha\frac{\partial E}{\partial b}$に更新　　$\frac{\partial E}{\partial b} = (y-t)\cdot(1-y)\cdot y$

誤差逆伝播法のさらなる学習のために

- 本資料では簡略化のため、中間層1層のシンプルなニューラルネットワークを考えました。また、活性化関数は、計算のしやすさからシグモイド関数を考えました。
- 多層のより一般化したニューラルネットワークにおける誤差逆伝播法の学習について、様々な書籍やwebサイトが存在していますので、興味のある方は目を通してみてください。
- 2019年現在、TensorFlowやそのラッパーのkerasなど、誤差逆伝播法を簡単に実装できるライブラリが充実しています。読者の皆さん自身で独自に実装できなくても、これらのライブラリを駆使することで高度なニューラルネットワークを構築することができます。

演習1：誤差逆伝播法の実装

- 中間層1層、ユニット2つのシンプルなニューラルネットを元に、重みを更新する誤差逆伝播法を実装してください。



出展： https://www.yukisako.xyz/entry/backpropagation

第10回：TensorFlowとKeras

ライブラリを活用したニューラルネットワークの開発環境

アジェンダ

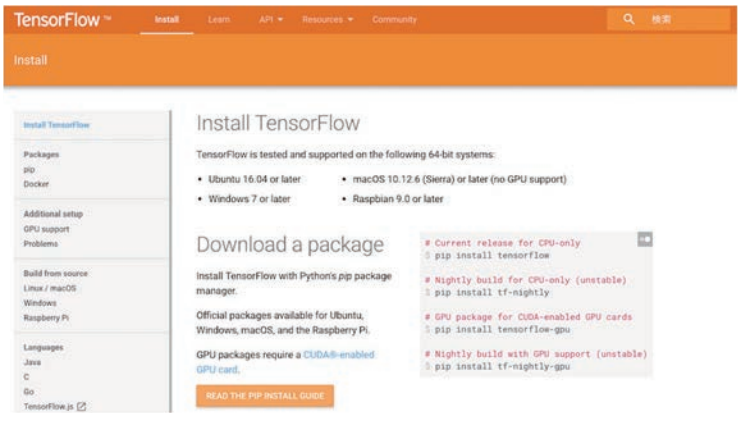
- TensorFlowとは
- Kerasとは
- 開発環境構築手順

TensorFlowとは

- Googleが開発しオープンソースで公開している、機械学習に用いるためのソフトウェアライブラリのことです（https://github.com/tensorflow/tensorflow）。
- TensorFlowは元々、Google内部での使用のためにGoogle Brainチームによって開発されました。開発された目的は、「人間が用いる学習や論理的思考と似たように、パターンや相関を検出し解釈するニューラルネットワークを構築、訓練することができるシステムのための要求を満たすため」とされています。

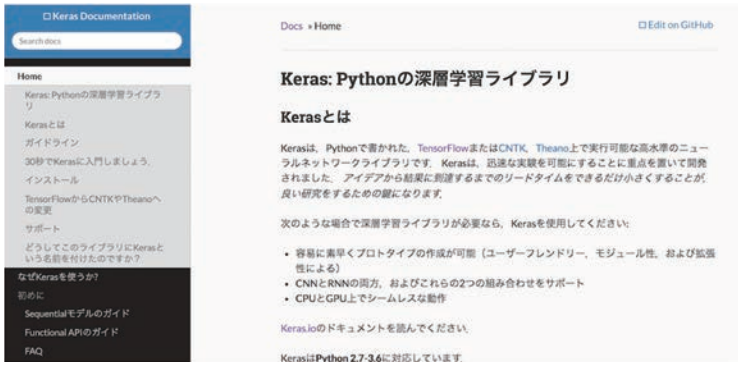
TensorFlowの使い方について

- 公式HP （https://www.tensorflow.org/）にインストールの仕方、チュートリアルなどが公開されています。



Kerasとは

- Kerasは、Pythonで書かれたオープンソースニューラルネットワークライブラリです（https://keras.io/）。MXNet、Deeplearning4j、TensorFlow、CNTK、Theanoの上部で動作します。ディープニューラルネットワークを用いた迅速な実験を可能にするよう設計され、最小限、モジュール式、拡張可能であることに重点が置かれています。中心的な開発者はGoogleエンジニアのFrançois Cholletです。



TensorFlowとKerasの組み合わせ

- TensorFlowとKerasを組み合わせることによって、複雑な多層ニューラルネットワークを比較的簡易に構築することができます。

演習1：TensorFlowとKerasの開発環境構築

- 下記コマンドを実行し、開発環境を構築してください（ついでに機械学習IIIで利用する画像処理ライブラリのpillowもインストールします）。
- モジュールを有効化するために、jupyterの再起動が必要です。

```
$ pip(conda) install tensorflow keras pillow pydot h5py
```

演習2

- TensorFlowとKerasが正常にインストールされていることを確認してください。

第11回：Kerasによる多層ニューラルネットワークの実装

ライブラリを活用したニューラルネットワークの実装

アジェンダ

- 層の追加方法
- 損失関数の設定方法
- 学習、評価の実行方法

層の追加方法

- コンストラクタもしくはadd()関数で層と追加することができます。

コンストラクタによる層の追加例

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

add関数による層の追加例

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

層の追加方法

- 入力データの次元数（ユニットの数）と中間層の次元数を指定します。

コンストラクタによる層の追加例

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

784次元の入力データを、32次元の中間層に接続する。

出力層の次元は10。

add関数による層の追加例

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

層の追加方法

- 活性化関数を指定します。

コンストラクタによる層の追加例

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

中間層の活性化関数はReLU関数。

出力層の活性化関数はソフトマックス関数。

add関数による層の追加例

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

損失関数の設定方法

- 解決したい問題ごとに損失関数（評価関数）を設定することが可能です。
- 独自の損失関数を定義することも可能です。

```
# マルチクラス分類問題の場合
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 2値分類問題の場合
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# 平均二乗誤差を最小化する回帰問題の場合
model.compile(optimizer='rmsprop',
              loss='mse')

# 独自定義の評価関数を定義
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

学習、評価の実行方法

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout

# 疑似データの生成
x_train = np.random.random((1000, 20))
y_train = np.random.randint(2, size=(1000, 1))
x_test = np.random.random((100, 20))
y_test = np.random.randint(2, size=(100, 1))

model = Sequential()
model.add(Dense(64, input_dim=20, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

ネットワークの定義。
入力データの次元は20。
中間層1の次元は64で、活性化関数はReLU関数。
中間層2の次元も64で、活性化関数はReLU関数。
出力層の次元は1で、活性化関数はシグモイド関数。

損失関数は2値分類問題用の関数を設定している。

学習の実行。

評価の実行。

演習1：多層ニューラルネットワークの実装

- 中間層が2層以上のニューラルネットワークを構築し、学習/評価を実行してみてください。

演習2：データを変えての実行

- Irisデータ（アヤメ）データを使用し、多層ニューラルネットワークを実行してください。
- Irisデータ（https://en.wikipedia.org/wiki/Iris_flower_data_set）とは機械学習のデモでよく用いられるデータセットのことです。

演習3：中間層の数を変えての実行

- 中間層の数を減らして実行してください。中間層を減らした方が精度が向上する（多いと逆に精度が低下する）現象を確認できます。

第12回：学習アルゴリズム

学習のテクニック

アジェンダ

- 説明変数と目的変数の分離
- 学習データとテストデータの分離
- 目的変数のone hot vector化
- ミニバッチ学習とエポック数

データの加工：説明変数と目的変数の分離

- 通常、データセットは説明変数と目的変数が一体となっています。
- 機械学習モデルを作成する前に、どの列を目的変数として使用し、どの列を説明変数として使用するのかを決めます。

アヤメデータ

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

花弁とがく片の長さ、幅のデータ。
あやめの種類を判定するための説明変数として使用する。

あやめの種類。
目的変数として使用する。

演習1：説明変数と目的変数の分離

- アヤメデータを説明変数と目的変数に分離してください。

アヤメデータ

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

花弁とがく片の長さ、幅のデータ。
あやめの種類を判定するための説明変数として使用する。

あやめの種類。
目的変数として使用する。

データの加工：学習データとテストデータの分離

- モデルの作成と評価を行うためには、データセットを4分割する必要があります。
- 学習データでモデルを作成し、テストデータでモデルの精度を評価します。

演習2：学習データとテストデータの分離

- アヤメデータセットを学習データとテストデータに分離してください。

データの加工：目的変数のone hot vector化

- アヤメデータの目的変数は、下記のように品種名（setone/versicolor/virginica）の羅列となっています。

```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
```

- 1次元の目的変数を、下記のように3次元のone hot vectorに変換し、機械学習で扱えるようにします。

setosa	versicolor	virginica
1	0	0
0	1	0
1	0	0
0	0	1

演習3：目的変数のone hot vector化

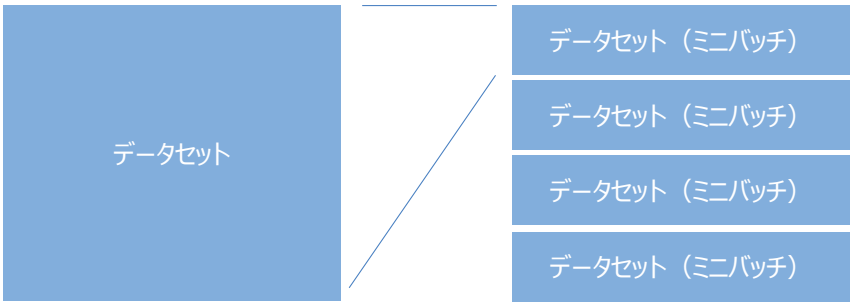
- アヤメデータの目的変数をone hot vectorに変換し、機械学習で扱えるようにしてください。

```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
```

setosa	versicolor	virginica
1	0	0
0	1	0
1	0	0
0	0	1

ミニバッチ学習とエポック数

- 確率的勾配降下法において、学習データをランダムに抽出するということを学習してきました。
- データセット全体からランダムに抽出したデータセットをミニバッチといい、確率的勾配降下法による1回の学習を1ミニバッチで実施します。
- データセットから抽出した全ミニバッチで学習が終わると、1エポックの学習が終了したと言います。
- 1エポックの学習が終了すると、再度ミニバッチの抽出から実施して2エポック目の学習へと続いていきます。



演習4：ミニバッチ数の変動

- ミニバッチ数を変えて学習を実行し、精度が異なることを確認してください。

第13回：過学習の回避

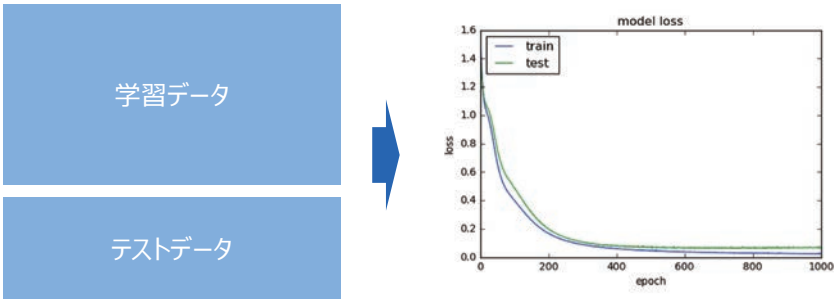
損失のモニタリング

アジェンダ

- 学習時とテスト時の損失
- 過学習が起きているときの損失の遷移
- 過学習の回避

学習時とテスト時の損失

- モデルの学習時の損失関数の出力の遷移が、右図の「train」の曲線です。
- 各エポックごとにテストデータをモデルに投入して計算した損失関数の出力が、右図の「test」の曲線です。



Kerasにおける実装

- モデルを学習させるfit関数にvalidation_dataという名前でテストデータを設定します。

```
history = model.fit(train_X, train_y_ohe,
          epochs=1000,
          batch_size=10,
          validation_data=(test_X, test_y_ohe))
```

- 1エポックの学習ごとにテストデータにおける損失を計算させることができます。

```
Epoch 776/1000
75/75 [==============================] - 0s 448us/step - loss: 0.0291 - acc: 1.0000 - val_loss: 0.0692 - val_acc: 0.9733
```

Kerasにおける実装

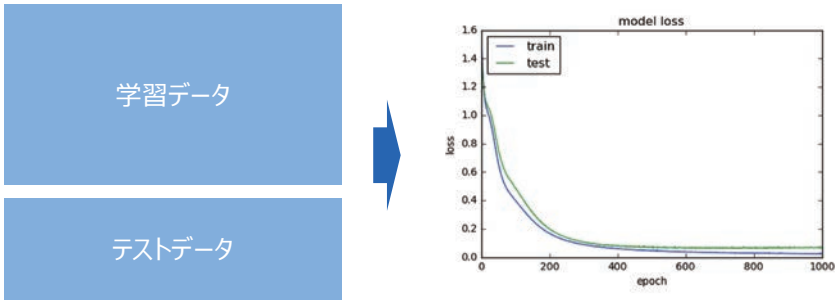
- Fit関数の戻り値を時系列で表示させることができます。

```
# 損失率の確認
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

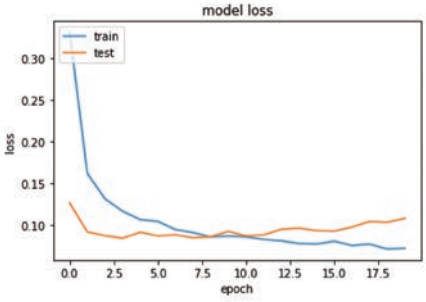
演習1：損失曲線の確認

- モデル学習時とテスト時の損失関数の出力を確認してください。



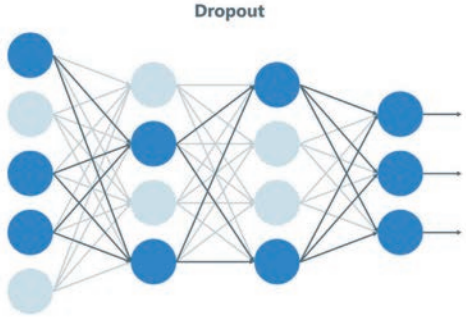
過学習とは

- 学習データによる損失が小さくなっているにも関わらず、テストデータにおける損失が大きくなっていくことを過学習といいます。
- 学習データに過度に適応してしまい、テストデータの予測精度が落ちることが原因です。



過学習の回避：ドロップアウト

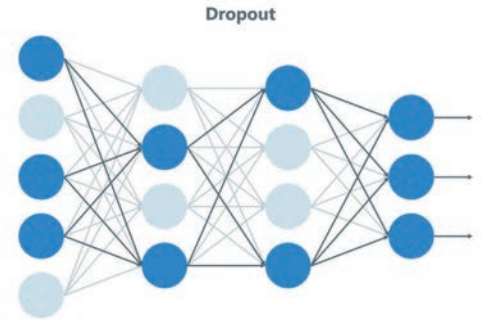
- 階層の深いニューラルネットを精度よく最適化するためにHintonらによって提案された手法です。
- ニューラルネットワークを学習する際に、ある更新で層の中のノードのうちのいくつかを無効にして（そもそも存在しないかのように扱って）学習を行い、次の更新では別のノードを無効にして学習を行うことを繰り返します。これにより学習時にネットワークの自由度を強制的に小さくして汎化性能を上げ、過学習を避けることができます。



出展：http://sonickun.hatenablog.com/entry/2016/07/18/191656

演習2：ドロップアウトの実行

- ドロップアウト率を変更して実行してください。



出展：http://sonickun.hatenablog.com/entry/2016/07/18/191656

過学習の回避：その他の手法

- 最適化アルゴリズムを変更してみる。

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

- 活性化関数を変更する。
- 正則化を実施する。
- データの正規化、標準化を実施する。
- ハイパーパラメータの自動設定を実行する。

参考URL
https://www.hellocybernetics.tech/entry/2016/11/13/035443
https://qiita.com/jiny2001/items/1d3f2d0370b2689d2da7
https://www.hellocybernetics.tech/entry/2018/02/10/084840#グリッドサーチ

第14回：モデルの保存と読み込み

Kerasにおける操作

アジェンダ

- Kerasによるモデルの保存
- Kerasによるモデルの読み込み
- 読み込んだモデルの追加学習

機械学習モデルの保存

- 学習データの量が増えるほど、またニューラルネットワークが複雑になるほど、モデルの学習には時間が掛かるため、長時間の学習後にはモデルを保存することが必要となります。

- モデルの精度を向上させるために、パラメータの組み合わせを変えて学習を何パターンも行います。最良の組み合わせパターンを探索するには、学習のたびにモデルを保存して記録を取ることが必要になります。

Kerasにおけるモデルの保存

- Kerasのmodelインスタンスのsave関数を利用します。

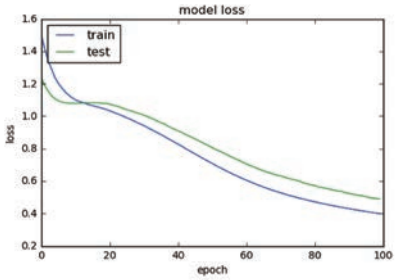
```
# 保存するモデルの名称
path_model = "resources/Chapter14_1.h5"
model.save(path_model)
```

演習1：Kerasによるモデルの保存

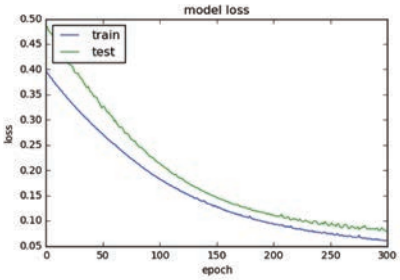
- モデルを作成して学習させた後、モデルを保存してください。

機械学習モデルの読み込みと追加学習

- 過去に作成されたモデルを読み込みテストを実施したり、または読み込んだモデルを更に学習させて精度向上を図ることがあります。



一回目の学習で作成したモデルの損失



作成したモデルを読み込み、追加学習した後のモデルの損失

Kerasにおけるモデルの読み込みと追加学習

- load_model関数によりmodelインスタンスを作成し、一回目の学習と同様にmodelインスタンスのfit関数で追加学習を実行できます。

```
# モデルの読み込み
from keras.models import load_model
model = load_model(path_model)
```

```
history = model.fit(train_X, train_y_ohe,
        epochs=300,
        batch_size=10,
        validation_data=(test_X, test_y_ohe))

loss, accuracy = model.evaluate(test_X, test_y_ohe, batch_size=10)
print("Accuracy = {:.2f}".format(accuracy))
```

演習2：Kerasによるモデルの読み込みと追加学習

- 過去に保存したモデルを読み込み、追加学習を実行してください。

第15回：機械学習Ⅱ総復習

第1回：「機械学習の概要」の振り返り

- 機械学習の代表的なアルゴリズムとその概要について述べてください。
- 1969年にM. Minsky氏によって提示されたパーセプトロンの限界について説明してください。

第2回：「単純パーセプトロン」の振り返り

- 単純パーセプトロンで定義すべき変数について説明してください。
- 入力ニューロンが2つの場合の、単純パーセプトロンの計算式をpythonで実装してください。

第3回：「単純パーセプトロンによる論理演算」の振り返り

- 単純パーセプトロンで論理和が計算できるアルゴリズムを実装してください。
- 単純パーセプトロンで論理積が計算できるアルゴリズムを実装してください。
- 単純パーセプトロンで否定論理積が計算できるアルゴリズムを実装してください。

第4回：「多層パーセプトロン」の振り返り

- 論理和（OR）、論理積（AND）、否定論理積（AND）を組み合わせて排他的論理和（XOR）が計算できるパーセプトロンを実装してください。

第5回：「ニューラルネットワークの種類」の振り返り

- 順伝播型ニューラルネットワークを構築するには、どの様なパラメータを設定する必要があるのか列挙してください。
- 畳み込み型ニューラルネットワークを構築するには、どの様なパラメータを設定する必要があるのか列挙してください。
- 「自然言語処理における質問の分類問題」以外に、再帰型ニューラルネットワークが適用できそうな事例を挙げてください。

第6回：「活性化関数」の振り返り

- ステップ関数を実装し、入力値に対する挙動を確認してください。
- シグモイド関数を実装し、入力値に対する挙動を確認してください。
- ReLU関数を実装し、入力値に対する挙動を確認してください。
- 恒等写像関数を実装し、入力値に対する挙動を確認してください。
- ソフトマックス関数を実装し、入力値に対する挙動を確認してください。

第7回：「勾配降下法」の振り返り

- 誤差関数J = x^2 - 4x + 5が最小となるxを求めるプログラムを実装してください。
- 最適値よりも小さな値から次第に最適値に向けて収束する様子を観測してください。

第8回：「確率的勾配降下法」の振り返り

- 確率的勾配降下法について説明してください。
- 局所最適化について説明してください。

第9回：「誤差逆伝播法」の振り返り

- 誤差関数について説明してください。
- 誤差逆伝播法の式の導出を復習してください。

第10回：「 TensorFlowとKeras 」の振り返り

- TensorFlowとKerasについて説明してください。
- TensorFlowだけでなくKerasを使用するメリットについて述べてください。

第11回：「 Kerasによる多層ニューラルネットワークの実装」の振り返り

- Irisデータ（アヤメ）データを使用し、多層ニューラルネットワークを実行してください。

第12回：「学習アルゴリズム」の振り返り

- アヤメデータを説明変数と目的変数に分離してください。
- アヤメデータセットを学習データとテストデータに分離してください。
- アヤメデータの目的変数をone hot vectorに変換してください。
- ミニバッチ数を変えて学習を実行し、精度が異なることを確認してください。

第13回：「過学習の回避」の振り返り

- モデル学習時とテスト時の損失関数の出力を確認してください。
- 過学習について説明してください。
- ドロップアウト率を変更してニューラルネットワークを実行してください。

第14回：「モデルの保存と読み込み」の振り返り

- モデルを作成して学習させた後、モデルを保存してください。
- 過去に保存したモデルを読み込み、追加学習を実行してください。

2019 年度「専修学校による地域産業中核的人材養成事業」
Society5.0 実現のための IT 技術者養成モデルカリキュラム開発と実証事業

2019 年度「専修学校による地域産業中核的人材養成事業」
Society5.0 実現のための IT 技術者養成モデルカリキュラム開発と実証事業

機械学習Ⅱ教材

令和２年２月

学校法人電子学園（日本電子専門学校）
〒169-8522　東京都新宿区百人町 1-25-4
TEL 03-3369-9333　FAX 03-3363-7685