2023年8月29日

O,S

SanaeProject

NeuralNetworkレポート

NeuralNetworkレポート-目次

目次

[**文献** 4](#_Toc144832293)

[参考文献 4](#_Toc144832294)

[**NeuralNetworkレポート** 5](#_Toc144832295)

[ニューラルネットワークとは 5](#_Toc144832296)

[活性化関数とは 5](#_Toc144832297)

[ニューラルネットワークの計算方法 6](#_Toc144832298)

[各層の計算方法 6](#_Toc144832299)

[Affine変換ってなに？ 7](#_Toc144832300)

[SoftMax関数と恒等関数 7](#_Toc144832301)

[SoftMax関数 7](#_Toc144832302)

[恒等関数 8](#_Toc144832303)

[出力層にはSoftMax関数と恒等関数どちらを利用したらいい? 8](#_Toc144832304)

[重みの改善方法 8](#_Toc144832305)

[損失関数が必要な理由 8](#_Toc144832306)

[重み改善の式の導出 9](#_Toc144832307)

[誤差逆伝番 9](#_Toc144832308)

[2層のニューラルネットワークの重みの改善方法 10](#_Toc144832309)

[1:出力値を求める 10](#_Toc144832310)

[2:SoftMaxLayerから逆伝番してWhoの重みを改善 10](#_Toc144832311)

[3:SigmoidLayerから逆伝番してWihの重みを改善 10](#_Toc144832312)

[**学習テクニック** 11](#_Toc144832313)

[画像の回転 11](#_Toc144832314)

[バッチ処理 11](#_Toc144832315)

[重み 11](#_Toc144832316)

[**NeuralNetworkレポート-補足** 12](#_Toc144832317)

[Sigmoid関数以外の活性化関数 12](#_Toc144832318)

[導関数 13](#_Toc144832319)

[数値微分 13](#_Toc144832320)

[重みの初期値 13](#_Toc144832321)

[**実験** 14](#_Toc144832322)

[活性化関数ごとの正答率の変化 14](#_Toc144832323)

[学習率の変化による正答率の変化 14](#_Toc144832324)

[画像の生成 15](#_Toc144832325)

[**開発環境** 16](#_Toc144832326)

[**ソースコード** 16](#_Toc144832327)

[**RecurrentNeuralNetworkレポート** 17](#_Toc144832328)

[RNNとは 17](#_Toc144832329)

[計算してみる 17](#_Toc144832330)

[重みの修正をする 17](#_Toc144832331)

**文献**

## **参考文献**

・ゼロから作るDeepLearning-Pythonで学ぶディープラーニングの理論と実装

斎藤康毅[著] 出版社:オライリー・ジャパン

・ニューラルネットワーク自作入門

Tariq Rashid[著] 新納 浩幸[監訳] 出版社:マイナビ

・高校数学でわかるディープラーニングのしくみ

涌井貞美[著]　出版社:ベレ出版

・分析モデル入門

杉山聡[著] 出版社:ソシム

・スッキリわかるPythonによる機械学習入門

須藤彬秋良[著] 出版社:株式会社インプレス

# **NeuralNetworkレポート**

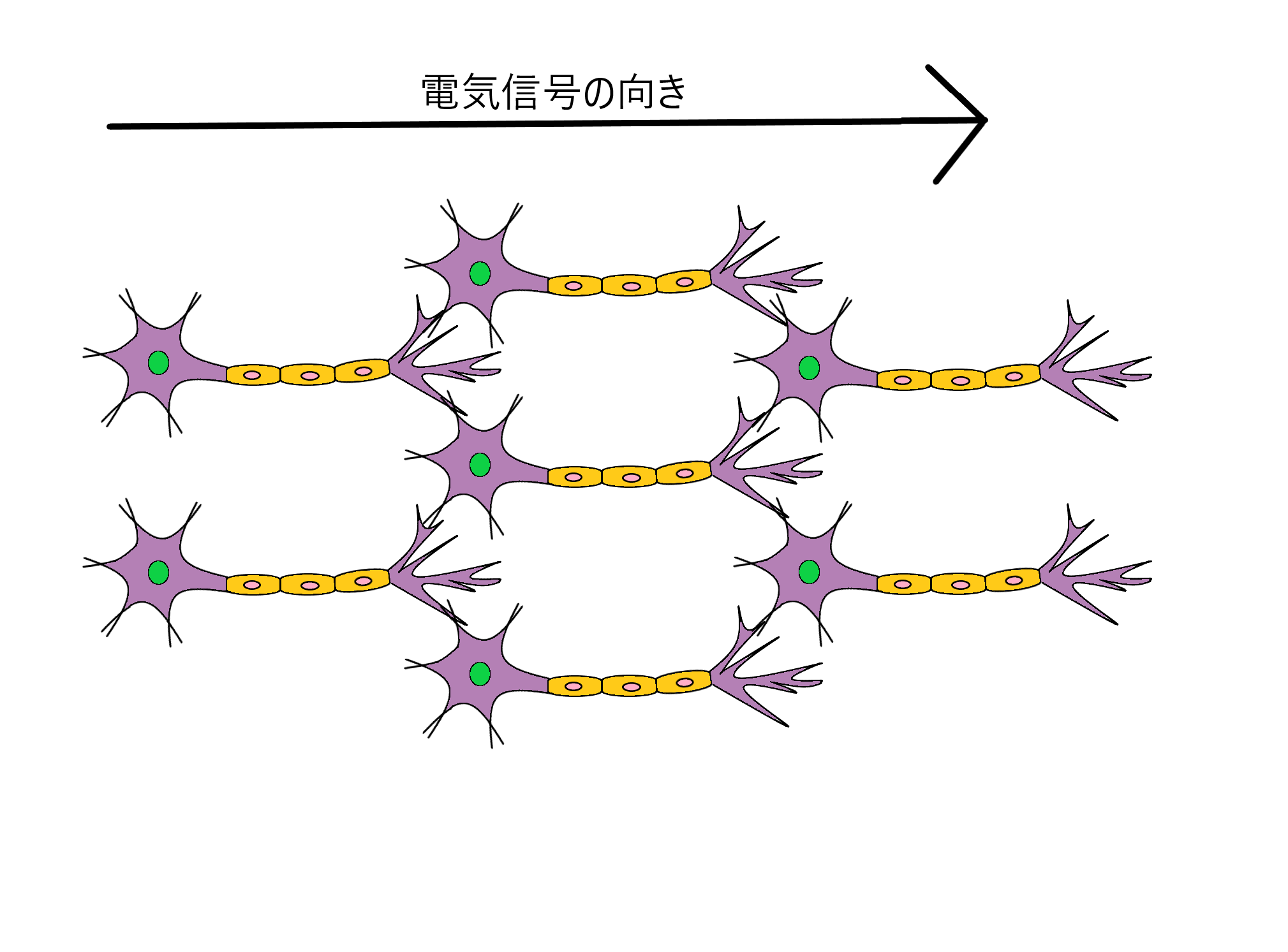
## **ニューラルネットワークとは**

ニューラルネットワーク（神経網、英: neural network;

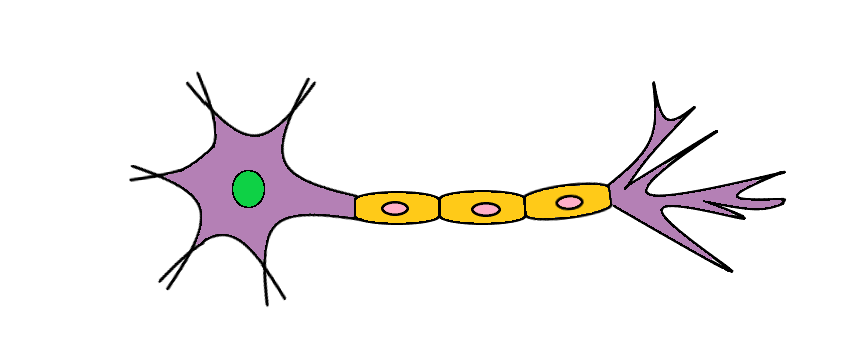
NN）は「入力を線形変換する処理単位」がネットワーク状に結合した数理モデルである。人工ニューラルネットワーク (英: artificial neural network) とも。

*ソース: ニューラルネットワーク -* [*https://ja.wikipedia.org*](https://ja.wikipedia.org/wiki/special:search/ニューラルネットワーク)

わかりやすく言えば**人間の脳神経系(ニューロン)**を数理モデル化したものを組み合わせたもののこと。



・ニューラルネットワーク

・ニューロン

"活性化関数（かっせいかかんすう、英: activation function）もしくは伝達関数（でんたつかんすう、英: transfer function）とは、ニューラルネットワークのニューロンにおける、入力のなんらかの合計（しばしば、線形な重み付け総和）から、出力を決定するための関数で、非線形な関数とすることが多い。"

*ソース: 活性化関数 -* [*https://ja.wikipedia.org*](https://ja.wikipedia.org/wiki/special:search/活性化関数)

## **活性化関数とは**

自然界のニューロンは**“入力が低い場合0,閾値に達すると出力が跳ね上がる。”**

そして自然界では境界がなめらかであるため以下の図のような動きをする。

**↑出力**

テキスト が含まれている画像

自動的に生成された説明**→入力**

左のような動きをする関数がSigmoid関数である。

Sigmoid(X)=1/(1+e^-X)

X<0:Sigmoid(X)は0に近づく

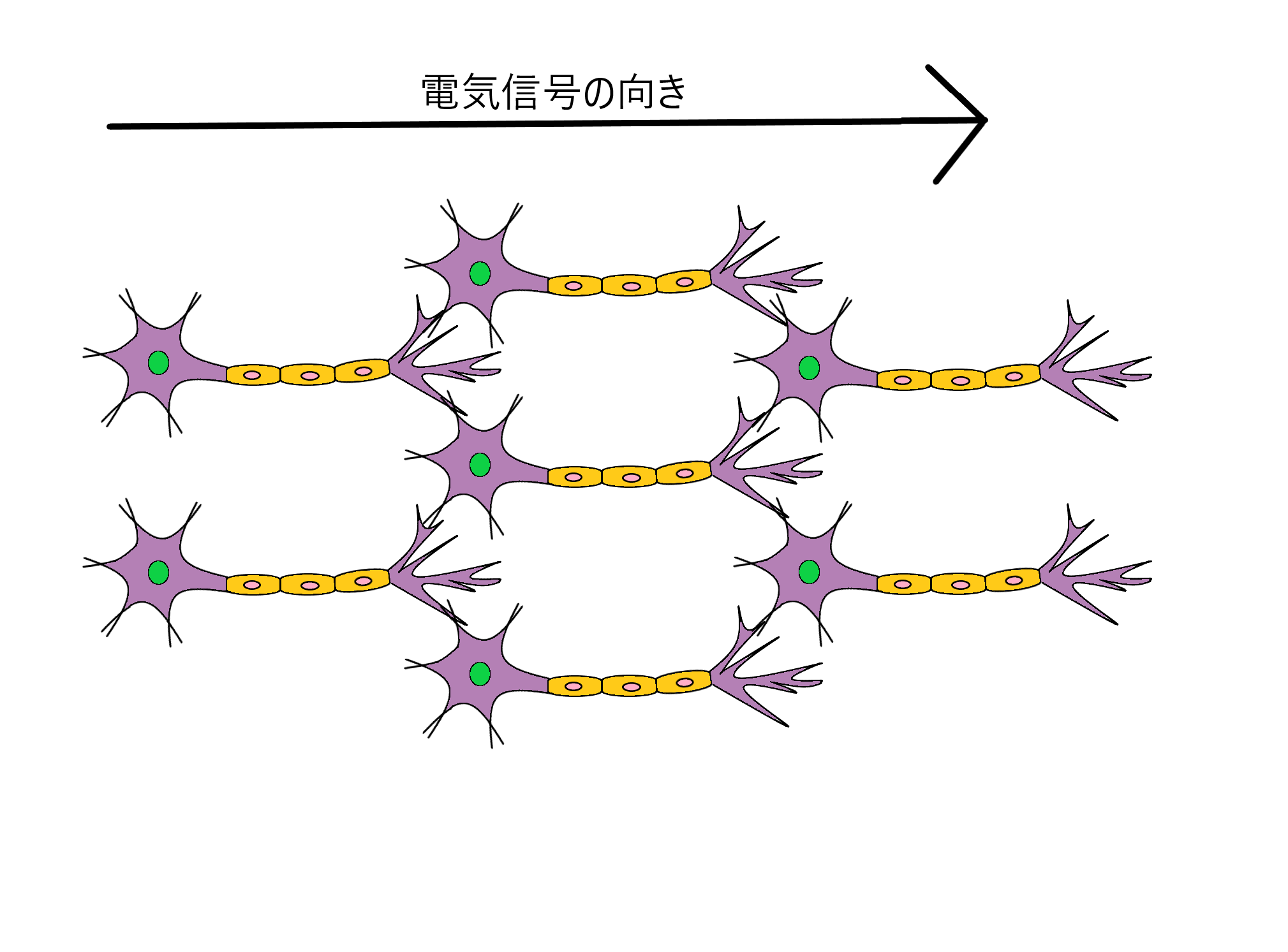
X=0:Sigmoid(X)は0.5

X>0:Sigmoid(X)は1に近づく

※活性化関数はSigmoid関数以外にもReLU,tanh,SoftMax,恒等関数などがある。(補足参照)

## **ニューラルネットワークの計算方法**

各ニューロンは多くの入力を受け取り発火した時にはニューロンに信号を送るので人工的なモデルに取り入れるには



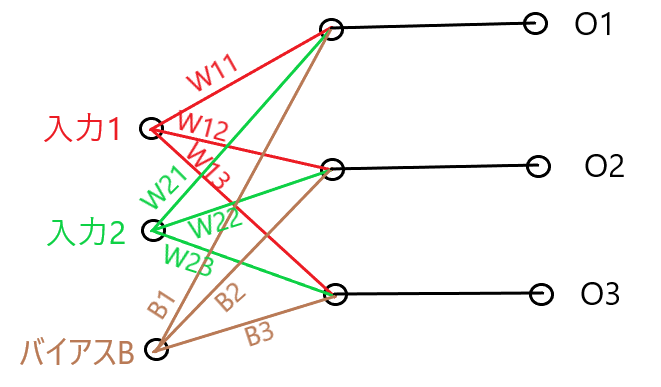
・ニューラルネットワーク

”以下の図のように前後の層の各ニューロンが接続されるようにすればよい”

図形

自動的に生成された説明

### **各層の計算方法**



入力 I= {I1,I2}

出力 O= {O1,O2,O3}

バイアスB= {B1,B2,B3}

重み W=

{W11,W12,W13}

{W21,W22,W23}

活性化関数をσ(X)と表記する。

**O1=σ(I1\*W11+I2\*W21+B1) O2=σ(I1\*W12+I2\*W22+B2) O3=σ(I1\*W13+I2\*W23+B3)**

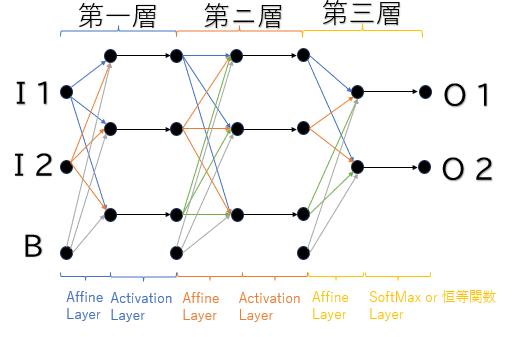
なので”Affineレイヤ”の出力をX={X1,X2,X3}とすると以下のように表現できる。

**O1=σ(X1) O2=σ(X2) O3=σ(X3)**

Xi = Σn,j=1(Ij \* Wji)+Bi　なので行列積を用いて表すと以下のようになる。

**X = {X1,X2,X3} = I \* W + B**

### **Affine変換ってなに？**



通常Affineレイヤと活性化レイヤを合わせて1層とするがここでは実装するときのことを考え

Affineレイヤと活性化レイヤを分割して表記している。

**活性化レイヤ**

活性化レイヤは文字通り入力されたものを活性化関数に通し出力するレイヤである。

**Affineレイヤ**

アフィン変換は、一次変換に平行移動を加えたものです。

中学数学で習う比例y=axを一般化したものが線形変換（一次変換）で、１次関数y=ax+bを一般化したものがアフィン変換です。

[アフィン変換の意味と4つの基本要素 – 具体例で学ぶ数学](https://mathwords.net/affine#:~:text=%E3%82%A2%E3%83%95%E3%82%A3%E3%83%B3%E5%A4%89%E6%8F%9B%E3%81%AF%E3%80%81%E4%B8%80%E6%AC%A1%E5%A4%89%E6%8F%9B,%E3%82%82%E3%81%AE%E3%81%8C%E3%82%A2%E3%83%95%E3%82%A3%E3%83%B3%E5%A4%89%E6%8F%9B%E3%81%A7%E3%81%99%E3%80%82)

つまり上部で求めたI\*W+Bを意味する。しかしAffine変換という言葉を使ってはいるが単にax+bを行列計算させたいので使っているというだけでアフィン変換本来の目的では利用していないということがわかる。

## **SoftMax関数と恒等関数**

SoftMax関数と恒等関数は”出力層”の活性化関数として使用される。

### **SoftMax関数**

入力X={X1,X2,X3} 出力O={O1,O2,O3}

**Oi = e^Xi / Σn,j=1(e^Xj)** で表される。

しかしこの式はe^Xj、e^Xiが**無限大**になり正常に計算できないことがあるそのため上記の式を変形する。

Oi **=**{e^Xi/Σn,j=1(e^Xj)}={ C \* e^Xi / C\*Σn,j=1(e^Xj)}

={e^(Xi+logC)/Σn,j=1(e^(Xj+logC))} = **e^(Xi+C’)/ Σn,j=1(e^(Xj+C’))**とすればよい。

通常C’=　-最大値　とする。 O=SoftMax(X)である。

### **恒等関数**

恒等関数はそのまま出力する関数である。

入力X={X1,X2,X3} 出力O={O1,O2,O3} **X=O** である。

### **出力層にはSoftMax関数と恒等関数どちらを利用したらいい?**

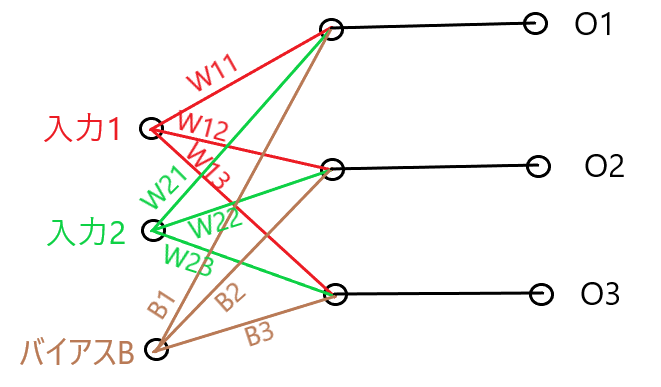
分類問題と回帰問題で使い分ける必要がある。

分類問題:SoftMax関数

回帰問題:恒等関数

## **重みの改善方法**

前ページでは計算方法を紹介したがどのように重みやバイアスをいじって正答率を改善することができるか考えていく。



まず一層のネットワークでの改善方法を考えてみる。

入力 I={I1,I2} 出力O={O1,O2,O3}

バイアスB={B1,B2,B3} 答えT={T1,T2,T3}

Affineレイヤの出力X={X1,X2,X3}

活性化関数をσ(X)と表記する。

重みを改善するためには損失関数が必要であるそのため損失関数について考える。

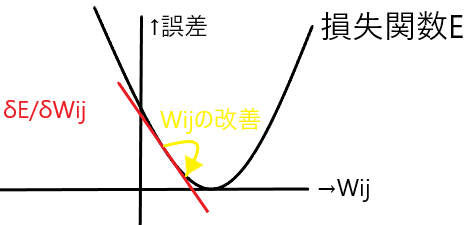
損失関数:

二乗和誤差 **E= Σn,i=1(Oi –Ti ) / 2**

交差エントロピー誤差 **E=-Σn,i=1(Ti log Oi)**

### **損失関数が必要な理由**

損失関数を使う理由は微分を行うためである。以下に損失関数によりWijを改善する方法の図を記す。

左記のように誤差関数をWijについて偏微分することで傾きを求めその傾きをWijから引くことで改善することができる。これを**勾配法**という。

**重みWijの改善の式 Wij = Wij –δE/δWij \* η**となる。 η:学習率

### **重み改善の式の導出**

重み改善の式:

**Wij = Wij – δE/δWij \* η** η:学習率

重み改善の式のδE/δWijについて考えてみる。

※損失関数を交差エントロピー誤差、活性化関数をSoftMaxとする。

連鎖率よりδE/δWijは以下のようになる。

**δE/δWij =δE/δOi \*δOi/δXi \*δXi/δWij**

δE/δOi \* δOi/δXiは導出が難しいので結果のみを以下に記述する。

δE/δOi \* δOi/δXi= **Oi – Ti**となる。

O-Tという形になるのはわざとそう関数が作られているからである。

SoftMax-交差エントロピー誤差,恒等関数-二乗和誤差で**O-T**となる。

**δXi/δWij=δ{Σn,j=1(Ij \* Wji)+Bi }/δWij = Ii** となる。

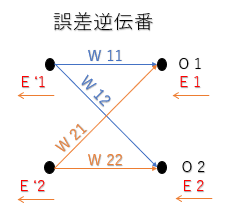
上記の式より出力層Affineレイヤの重み(Who)の改善の式は以下の通りになる。

**Who ij = Who ij – (Oi-Ti) \* Ii** となる。

## **誤差逆伝番**

重みの改善を多層にわたって行うためには誤差を逆伝番しなければならない。逆伝番の方法を以下に示す。

E’1 = E1 \* {W11 / (W11+W21)} + E2 \* {W12/(W12+W22)}



E’2 = E1 \* {W21 / (W11+W21)} + E2 \* {W22/(W12+W22)}

である。

しかし/ (W11+W21)の部分は単なる正規化因子に過ぎないので本質的には

誤差×重みである。そのため以下のように簡潔な式に直せる。

E’1 =E1 \* W11 + E2 \* W12

E’2 =E1 \* W21 + E2 \* W22となり

行列で表現すると**E’= {E’1,E’2} =E \* W^T** となる。

※W^TはWの転置を意味する。

## **2層のニューラルネットワークの重みの改善方法**

男, 空気, テーブル, コンピュータ が含まれている画像

自動的に生成された説明

I ={I1,I2,I3} Bi={BI1,BI2,BI3}

H={H1,H2,H3} Bh={BH1,BH2}

O={O1,O2}

Wih(1層Affineレイヤの重み)

Who(2層Affineレイヤの重み)

1層:入力 I,重みWih ,出力H,バイアスBi

2層:入力H,重みWho,出力O,バイアスBh

### **1:出力値を求める**

H={H1,H2,H3}=Sigmoid(I \* Wih + Bi)

O={O1,O2,O3}=Sigmoid(H\*Who+Bh)

### **2:SoftMaxLayerから逆伝番してWhoの重みを改善**

逆伝番の出力をDoutとする。

SoftMaxLayerから逆伝番Dout={Dout1,Dout2}=**{O1-T1, O2-T2}**

**Who = Who- H^T \* Dout\*η**

**Bh = Bh - Dout\*η**

**Dout =Dout \* Who^T**

### **3:SigmoidLayerから逆伝番してWihの重みを改善**

Dout = {H1,H2,H3} = σ’(I1),σ’(I2),σ’(I3)

=**{H1(1 – H1),H2(1-H2),H3(1-H3)}**※補足参照

**Wih= Wih-I^T \* Dout \*η**

**Bi= Bi-Dout\*η**

**Dout=Dout \* Who^T**

となる。

層が多い場合も同じことを繰り返せばよい。

**学習テクニック**

## **画像の回転**

**グラフ, 折れ線グラフ

自動的に生成された説明**X,Y地点にあるものを50°傾けた地点X’,Y’を求めてみる。

元の角度をθ次の角度をθ’として半径をRとすると

X=Rcosθ Y=Rsinθ

加法定理:

Rsin(θ±θ’)=Rsin(θ)cos(θ’)±Rcos(θ)sin(θ’)

Rcos(θ±θ’)=Rcos(θ)cos(θ’)∓Rsin(θ)sin(θ’)より

X’=Xcos(θ’)-Ysin(θ’) Y’=Ycos(θ’)+Xsin(θ’) となる。

中心(MX,MY)から画像を回転する場合X=MX+DX,Y=MY+DYなので

**X’ = MX + (X-MX)cos(θ’)-(Y-MY)sin(θ’)**

**Y’ = MY + (Y-MY)cos(θ’)+(X-MX)sin(θ’)** となる。

MNISTデータの“5”の角度を上記の式で変えた結果

グラフ

自動的に生成された説明

-50°

QR コード

自動的に生成された説明

50°

グラフ が含まれている画像

自動的に生成された説明

0°

なお上記の絵は色を反転させている。

学習の際に角度を変えて学習させることで正答率が上がる。

## **重み**

適度に多様性のあるデータが各層を流れることにより学習を効率よく行うことができる。

Xavierらの論文で推奨される重みの初期値は前層のノード数をnとして

1/√nの標準偏差を持つ分布の乱数を用いることである。

C++では

std::default\_random\_engine Engine(\_Seed);

std::normal\_distribution<> DistIn(0, (1/std::sqrt(\_Input\_Nodes)));

と書くことができる。

## **バッチ処理**

バッチ処理とは一定量のデータを一括処理することである。

今まで入力Iは一行のみだったがこれを複数行入力する。

通常 {行数,列数}=I{1,5}=>W{5,10}=>O{1,10}

バッチ処理 =I{5,5}=>W{5,10}=>O{5,10}

以下の図は入力:2 出力:2 のノードのAffine層のバッチ処理(3入力)する場合の図である。

ダイアグラム

自動的に生成された説明I=

I11,I12

I21,I22

I31,I32

W=

W11,W12

W21,W22

W11,W12

W21,W22

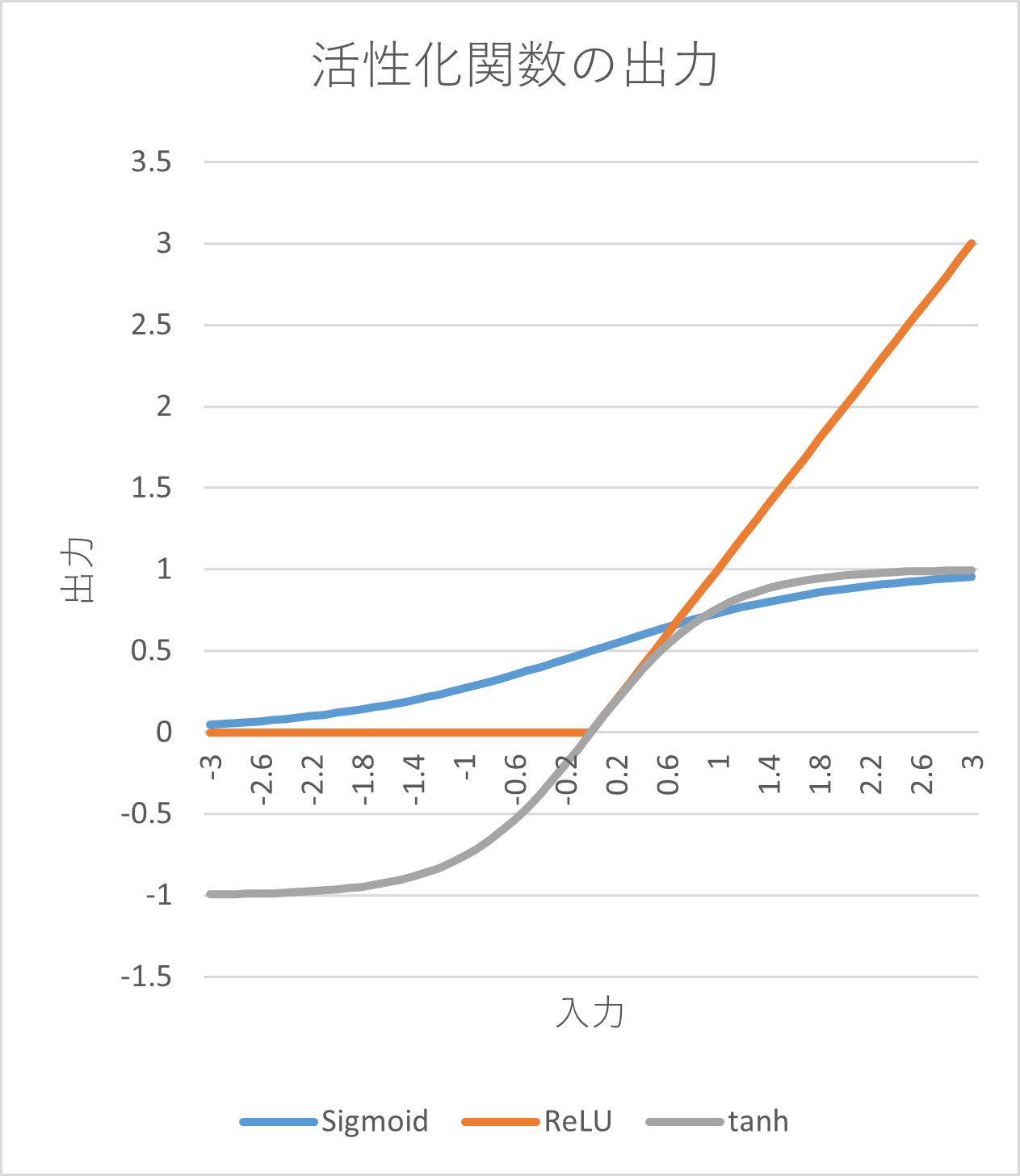
W11,W12

W21,W22

O=

**NeuralNetworkレポート-補足**

## **Sigmoid関数以外の活性化関数**



**ReLU関数**

ReLU関数はReLU(X)=max(0,x)で表される。

X>0=X X<=0=0を返す。

"ランプ関数（傾斜路関数）としても知られ、電気工学における半波整流回路と類似している。この活性化関数は、2000年にHahnloseらによって強い生物学的動機と数学的正当化を持って、動的ネットワークへ最初に導入された。2011年以前に広く使われていた活性化関数、例えばロジスティックシグモイド（これは確率論から発想を得ている。ロジスティック回帰を参照。）およびそのより実践的な機能的に同等な関数である双曲線正接関数と比較して、より深いネットワークのより良い訓練を可能にすることが2011年に初めて実証された。正規化線形関数は、2018年現在、ディープニューラルネットワークのための最も人気のある活性化関数である。" *ソース: ReLU -* [*https://ja.wikipedia.org*](https://ja.wikipedia.org/wiki/special:search/ReLU)

**Tanh関数**

ニューラルネットワークの基礎となっている情報処理モデル「パーセプトロン」（後日解説）では「ステップ関数」という活性化関数が用いられ、「バックプロパゲーション」（後日解説）が登場してからは「シグモイド関数」が活性化関数として使われるようになった。しかしシグモイド関数の微分係数（Derivative： 導関数の出力値）の最大値は0.25と小さいため、勾配が小さくなりがちで学習に時間がかかるという問題があった。そのため学習がより高速化するように、最大値が1.0（図2）となる「tanh関数」がよく採用されるようになった。

ソース: <https://atmarkit.itmedia.co.jp/ait/articles/2003/19/news015.html>

## **導関数**

Sigmoid関数の導関数:Sigmoid’(x)=Sigmoid(x)(1-Sigmoid(x))

ReLU関数の導関数 :ReLU’ (x)=(x>0=1),(x<=0=0)

Tanh関数の導関数 :tanh’ (x)=1-tanh(x)^2

## **数値微分**

導関数を使う以外にも数値微分を行うことで傾きを求めることができる。しかし時間がすごくかかるためお勧めしない。この方法は導関数が正しいか確かめる際に用いられることがある。

数値微分関数NumDiff(x)={σ(x+h)-σ(x)}/h 丸め誤差があるため**h=10^-4**とする。

しかし誤差が大きいため中心差分と遠方差分を利用して数値微分関数を実装する。

**数値微分関数NumDiff(x)={σ(x+h)-σ(x-h)}/2h**

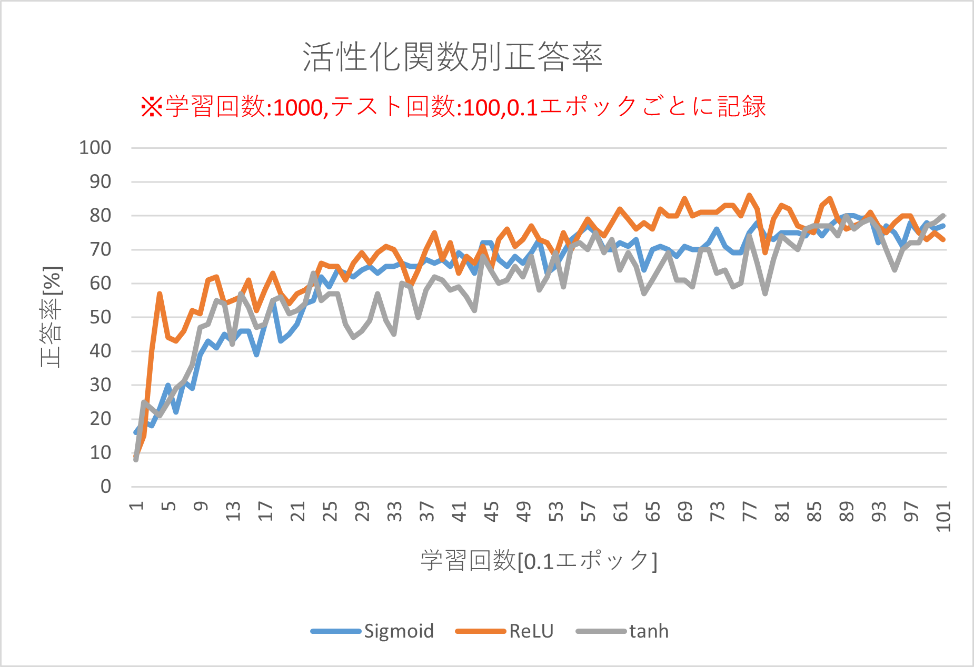
## **重みの初期値**

重み(W)の初期値は乱数でなければならない。

重みを均一な値にしてしまうと誤差逆伝番に於いて誤差がすべて均一に更新されてしまうからである。

# **実験**

## 活性化関数ごとの正答率の変化

この記録を見てみると少しだけ

ReLU関数の方が正答率の上昇が早いように感じる。しかし2層のニューラルネットワークではさほど違いがないようだ。

## 学習率の変化による正答率の変化

この記録を見ると学習率を上げることによりノイズに影響され正答率が著しく低下していることが推測される。

学習率が0.3%のものでは学習率が低すぎて正答率が上がりにくくなっている。(Sigmoid関数によるものの可能性も考えられる)

## 画像の生成

今までは画像を入力してその数値が何であるかをニューラルネットワークで分類していた。

この実験では逆に数値を入力して画像を生成してみる。

層は合計で3層のディープラーニングで学習させ入力中間層(1,2)はSigmoidレイヤ

出力層は”恒等関数”である。

1,2層でSigmoid関数を利用しているため正規化を行っている。

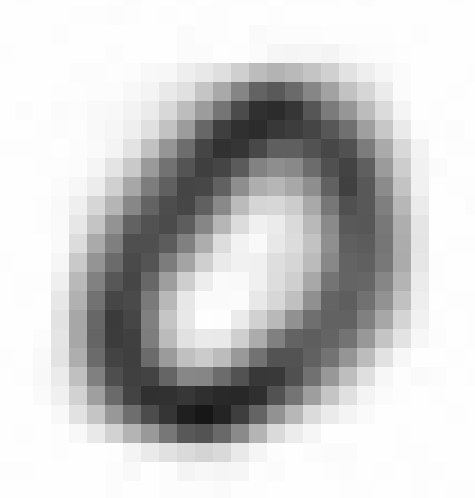
(画像を0~255を0.01~0.99に分布させた)

その結果を以下に表示する。

ロゴ

低い精度で自動的に生成された説明背景パターン が含まれている画像

自動的に生成された説明ロゴ

中程度の精度で自動的に生成された説明アイコン が含まれている画像

自動的に生成された説明

アイコン

自動的に生成された説明アイコン

自動的に生成された説明アイコン

自動的に生成された説明アイコン

自動的に生成された説明背景パターン が含まれている画像

自動的に生成された説明

(※色を反転させている。)

画像が少しぼけているようにも見えるが十分に読み取れる範囲であるため成功と言えるだろう。

尚生成にはSanaeProjectで作成したニューラルネットワークのヘッダファイルを利用している。

(ソースコード参照)

# **開発環境**

**ハードウェア**

CPU:Core i5 8250U

Memory:8GB

OS:Windows 11 pro

IDE:Microsoft Visual Studio2022

開発言語:C++14,C11

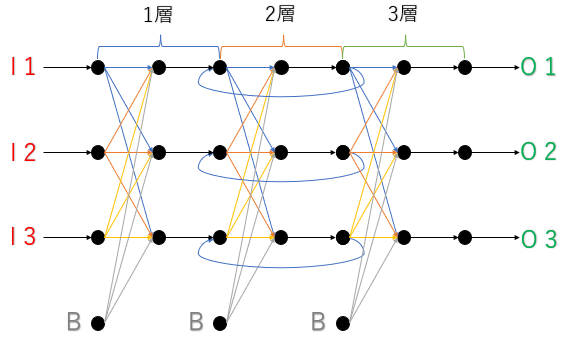
# **ソースコード**

<https://github.com/SanaePRJ/NeuralNetworkProject>

当レポートのグラフは上記にあるヘッダファイルから得たデータにより作成されている。

# **RecurrentNeuralNetworkレポート**

## **RNNとは**

左記にRNNの図を示す。

"回帰型ニューラルネットワーク（かいきがたニューラルネットワーク、英: Recurrent neural network; RNN）は内部に循環をもつニューラルネットワークの総称・クラスである。"

*ソース: 回帰型ニューラルネットワーク -* [*https://ja.wikipedia.org*](https://ja.wikipedia.org/wiki/special:search/回帰型ニューラルネットワーク)

中間層(2層)への入力はNNに加えて前回の出力値を入力している。

このようにすることで過去のデータにより出力値を変化させることができる。

入力層(1層)と出力層(3層)はNNと同じなので中間層(2層)のみ考える。

入力H={H1,H2,H3} 出力O={O1,O2,O3} バイアスB={B1,B2,B3}

前回の出力値P={P1,P2,P3} 重みWho 前回の出力値の重みWγ

### **計算してみる**

Oi=σ(Xi)とし、Dot関数を内積関数とする。

Xi=σn,j=1(Hj\*Who ji)+Bi+Pi \* Wγi となるので

**X=H\*Who+B+Dot(P,Wγ)** O=σ(X) となる。

### **重みの修正をする**

次の層からの逆伝番をDoutとすると2層の活性化レイヤの逆伝番XDout

XDout=Dout\*{σ’(X1),σ’(X2),σ’(X3)}である。

Who=Who - I^T \* XDout \* η

Bi=Bi-XDout\*η

**Wγ =Wγ-Dot(XDout,P)\* η**

Dout=XDout\*Who^T

となる。

基本的な構造はNNと変わらないためNNを継承してバイアスのように処理することでRNNを実装することができる。