機械学習エンジニアコース
Sprint

－ NN_ニューラルネットワーク －

DIVE INTO CODE

今回のモチベーション

目的はなにか

1. **深層学習入門**
   スクラッチを通してニューラルネットワークの基礎を理解する

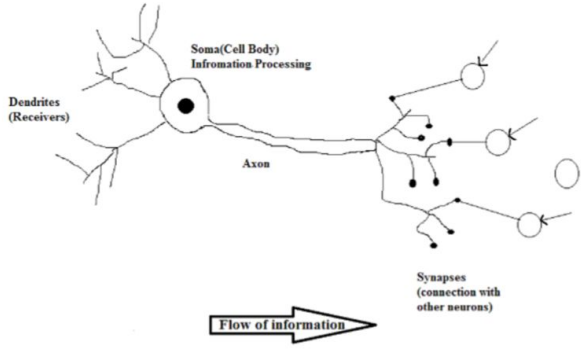
2. **MNIST data**
   画像データの扱いを知る

このスライドは?

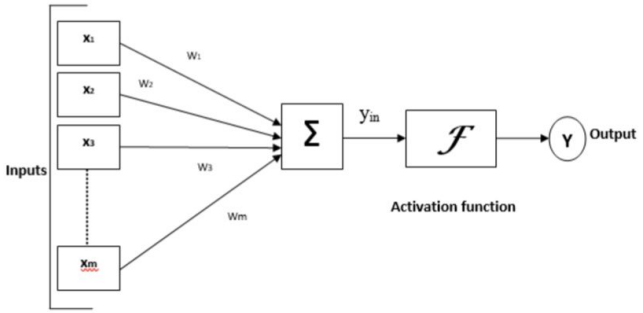
ここでは、ニューラルネットワークの基本的な知識を学びましょう

ニューラルネットワークとはなにか

脳（神経系）の情報伝達を模倣した数理モデル。神経細胞（ニューロン）では、入力刺激が閾値を超えた場合に**発火**（活性化）し、活動電位を発生させ、他の細胞に情報を伝達する。この発火が連鎖的に起こることで情報を伝え、計算を実現する。ニューロン間の**結合の強さが変化していくことを、学習**と呼んでいる。



一方、ニューラルネットワークでは、入力ベクトルを、**重み（結合荷重）**と**活性化関数**によって別のベクトル空間へ写像する。これを連鎖的に行い、情報を抽出する。**重みが調整されることを学習**と呼ぶ。



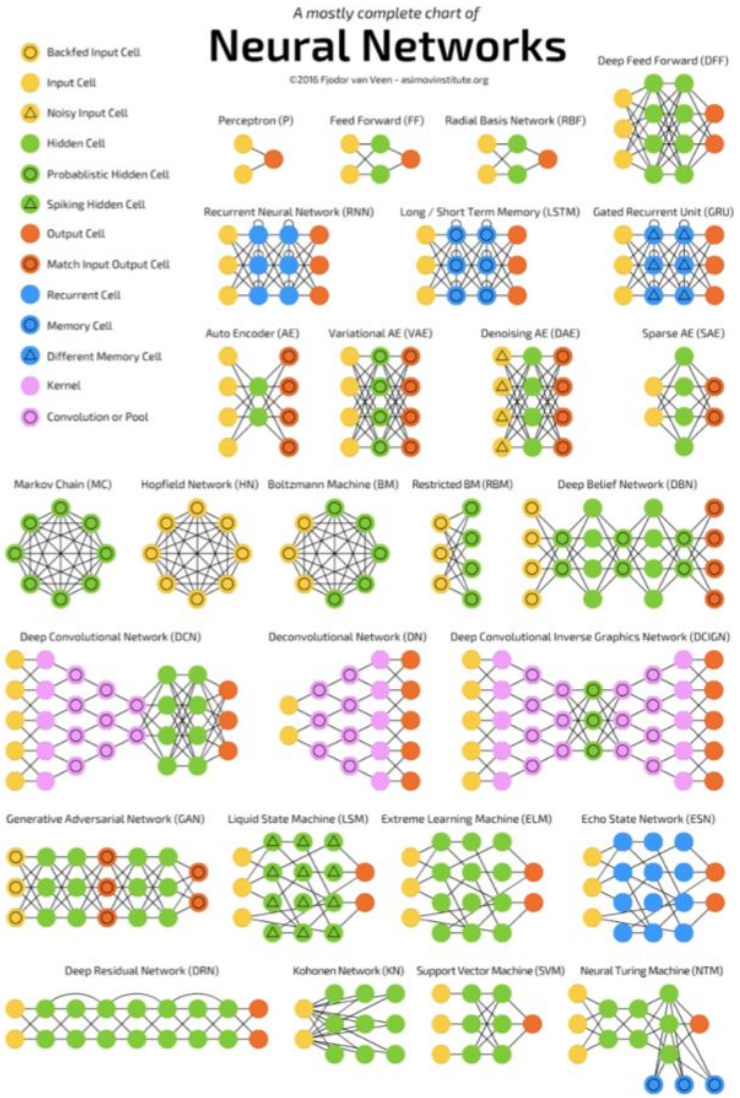
https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_basic_concepts.htm

ニューラルネットワークの構成

①処理単位をノード(ニューロン)と呼び、ノードの集合によって階層構造を作る

②ある層の出力を別の層の入力とする

③出力の分布を変化させる活性化関数をもつ

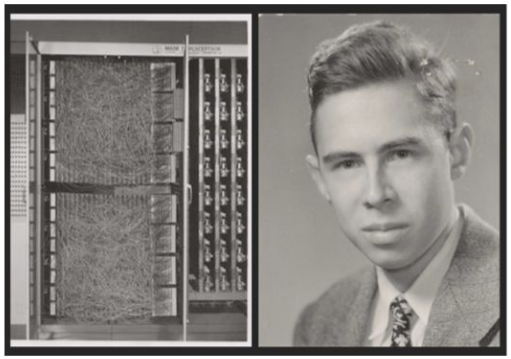


ニューラルネットワークの構成

ニューラルネットワークの歴史

1943 –ニューラルネットワークの最初の数理モデル　Walter Pitts and Warren McCulloch
1950 –機械学習の予測　Alan Turing
1952 –機械学習の父（「機械学習」を命名）　Arthur Samuel
1957 –ディープニューラルネットワークの基礎を提案（ハードウェアでパーセプトロンを作った）　Frank Rosenblatt
1959 –単純な細胞と複雑な細胞の発見　David H. Hubel and Torsten Wiesel
1960 –制御理論　Henry J. Kelley
1965 –最初のディープラーニングネットワークを作成　Alexey Ivakhnenko and V.G. Lapa
1979-80 – ANNが視覚パターンを認識する方法　Kunihiko Fukushima（福島邦彦）
1982 –ホップフィールドネットワークの作成　John Hopfield
1985 –プログラムが英単語の発音を学ぶ　Terry Sejnowski
1986 –形状認識と単語予測の改善　David Rumelhart, Geoffrey Hinton, and Ronald J. Williams
1989 –機械が手書き数字を読みとる　Yann LeCun
1989 – Qラーニング　Christopher Watkins
1993 –「非常に深い学習」タスクの達成　Jürgen Schmidhuber
1995 –サポートベクターマシン　Corinna Cortes and Vladimir Vapnik
1997 –長期短期記憶　Jürgen Schmidhuber and Sepp Hochreiter
1998 –勾配ベースの学習　Yann LeCun
2009 – ImageNet　Fei-Fei Li

https://www.import.io/post/history-of-deep-learning/



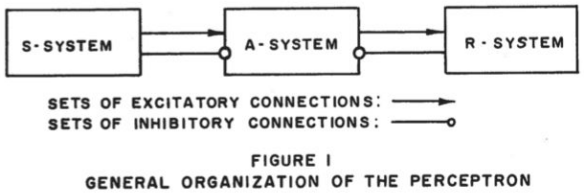


FIGURE 1
GENERAL ORGANIZATION OF THE PERCEPTRON

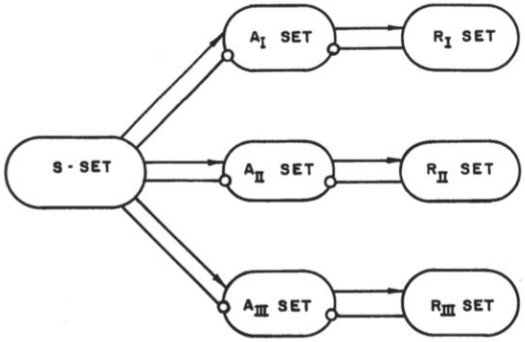


FIGURE 2
ORGANIZATION OF A PERCEPTRON WITH
THREE INDEPENDENT OUTPUT-SETS

"The Perceptron: A Perceiving and Recognizing Automaton"
https://www.import.io/wp-content/uploads/2017/06/rosenblatt-1957.pdf

この課題の対象者

① scikit-learnの分類モデルを用いて、学習、推定するコードが書ける方
② 分類モデルの計算過程(仮定関数、目的関数、最急降下法)を知っている方(1)

(1) sprint4 ロジスティック回帰スクラッチを解いた方

実装の流れ

ニューラルネットワークの問題設定を知る

① 重み(結合荷重)の初期値を決める
② 全結合層フォワードプロパゲーション(仮定関数+活性化関数)を作る
③ 目的関数(交差エントロピー誤差)を作る
④ 全結合層バックプロパゲーションを作る
⑤ 推定する

データセットの準備

データセットの準備

手書き文字データセット（MNISTデータセット）を
ニューラルネットワークに入力し、

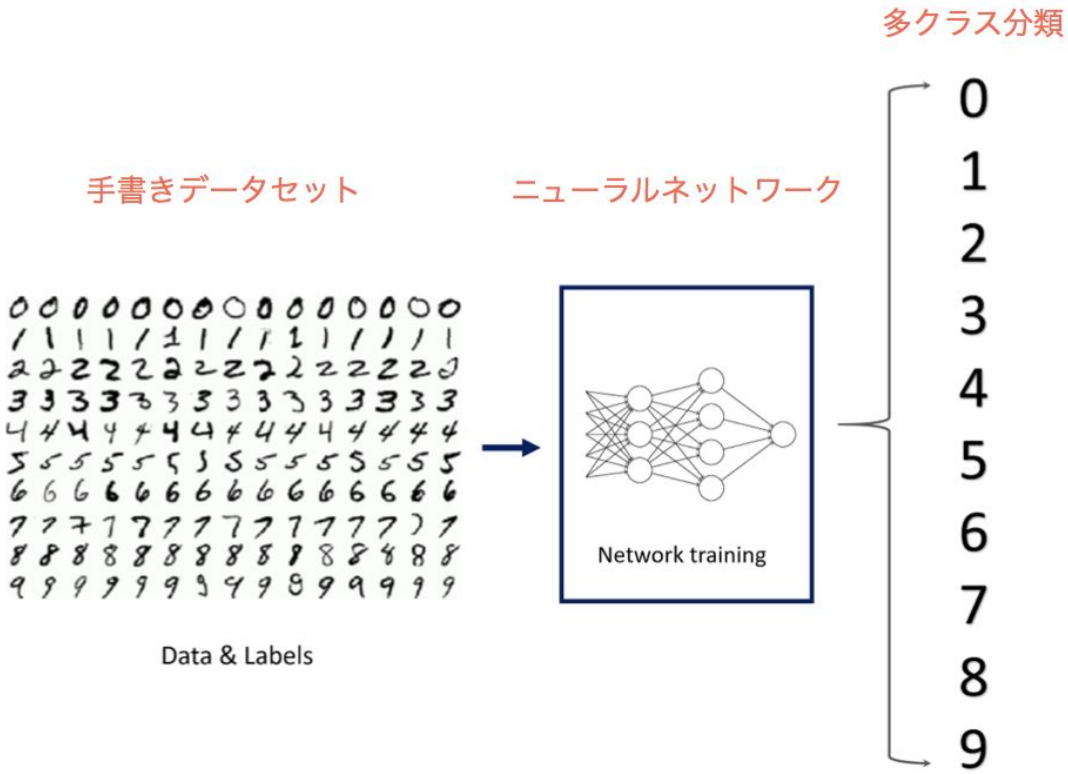
0〜9（数字）の多クラス分類を行う。

アイデア：（y_train：正解ラベルについて）

2クラス分類の場合：
ひとつのサンプルに対応する正解ラベルを、0あるいは1のような **スカラ**（int型）のまま学習に用いる。

多クラス分類の場合：
ひとつのサンプルに対応する正解ラベルが与えられたら、**クラス分の長さを持つワンホットベクトル**に変換させて学習に用いる。

Diverテキストの **y_train_one_hot [0]**をプリントして確かめてみよう。



データセットの準備

データセットの準備

画像データを**reshape**して(3次元のshape) から(2次元のshape)へ変換する。

アイデア：（X_train：画像データについて）

手書き文字データセットは、3次元shape (1,28,28) の形状でダウンロードされる(1)。

ニューラルネットワークに入力するために、これを2次元shape (1,784) データへ変換する。

さらに特徴量の値に対して以下のような正規化処理を行う。

X_train /= 255.0
X_test /= 255.0

(1) ここでのサンプル数は１と考えている。

重み（線形荷重）による線形変換

重み（結合荷重）による線形変換

ニューラルネットワークは、入力と出力の間に中間の層をもつ。

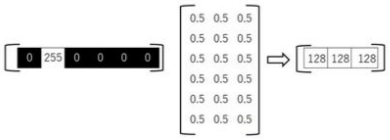
入力から**中間層への出力の数**が多いほどモデルの**表現力**が上がる。

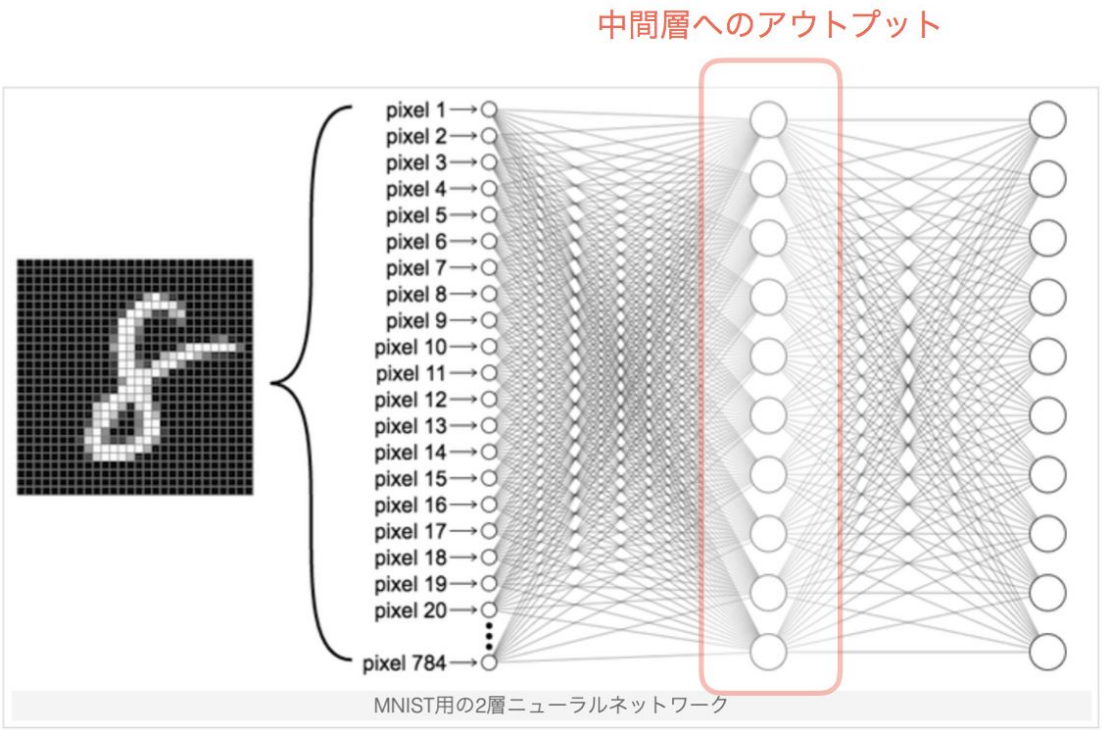
Sprint 7 のstackingの**次のステージへの出力 (Out of Fold) の特徴量数**は、用意した**モデルの種類の数**（M個）に対応していたことを思い出そう。表現力については、Sprint9で扱う。

アイデア：
中間層への出力を2つ以上に増やすために、入力に対して**重み行列**（**重みベクトルではなく**）を掛ける。
また中間層の出力の数だけバイアスを足し合わせる。この操作を**線形変換**と**並進**（平行移動）と呼ぶ（機械学習の文脈では、両方合わせて線形変換と呼ぶことが多い）。
事例：

https://arakan-pgm-ai.hatenablog.com/entry/2018/11/05/090000

前提：（そもそも）
線形回帰における**線形結合**は、入力に対する**重みベクトル**の**積とバイアスによる並進**だった。





中間層の出力

中間層の出力

中間層への**ひとつの出力**だけに注目すると、**線形結合**（加重和）と同じである。

アイデア：

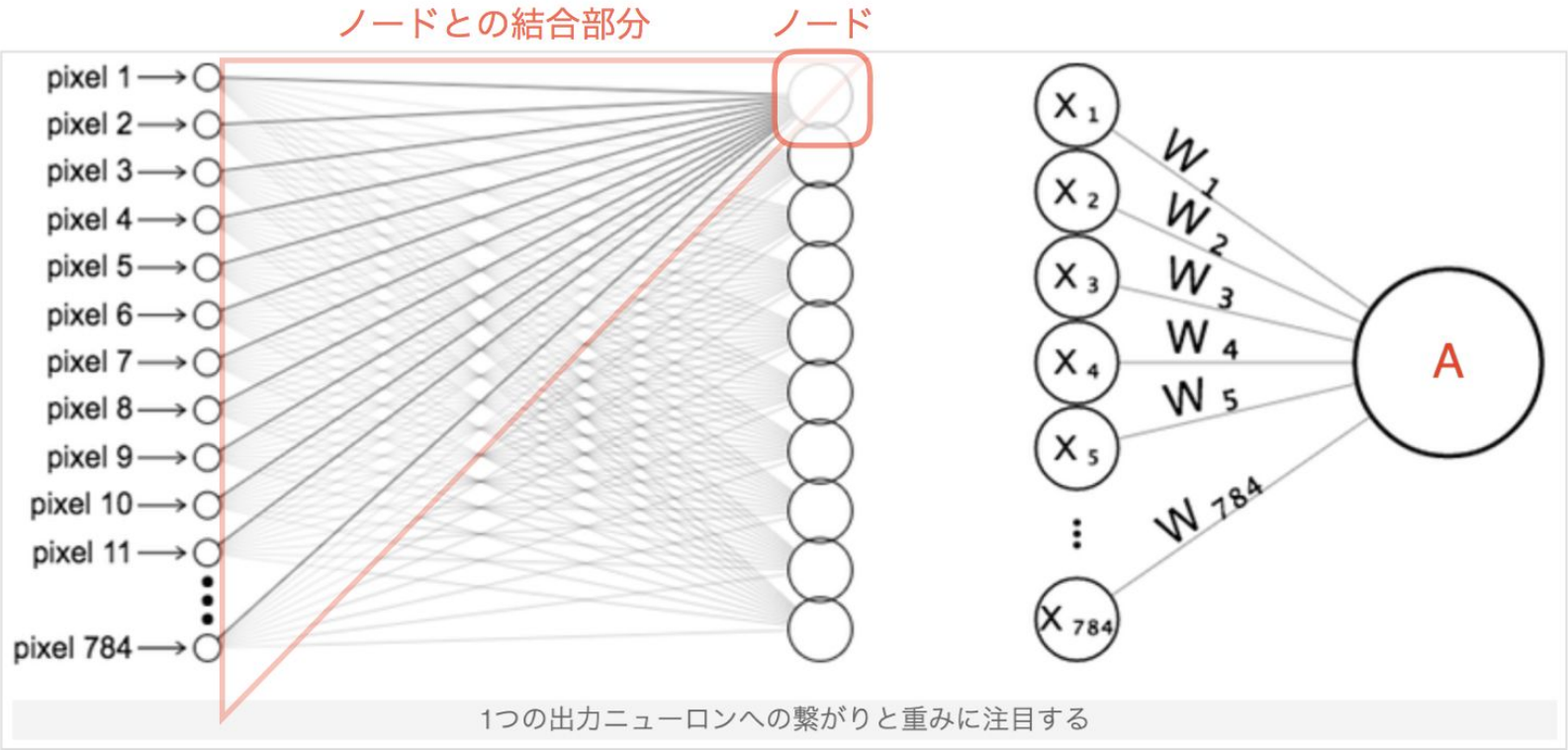
中間層へのひとつの出力を式で表すと線形結合と同様の形である（バイアス付き）。

ベクトル**W** : W_0 ,....., W_{784}

$A = W_0X_0 + W_1X_1 + W_2X_2 + W_3X_3 + \ldots + W_{784}X_{784}$

前提：

ニューラルネットワークでは、ひとつの出力を計算の基本単位として**ノード**（あるいはニューロン、ユニット）と呼ぶ。



1つの出力ニューロンへの繋がりと重みに注目する

非線形変換

非線形変換

各ノードにおいて、**線形変換**はさらに**活性化関数によって非線形変換**される。

アイデア：

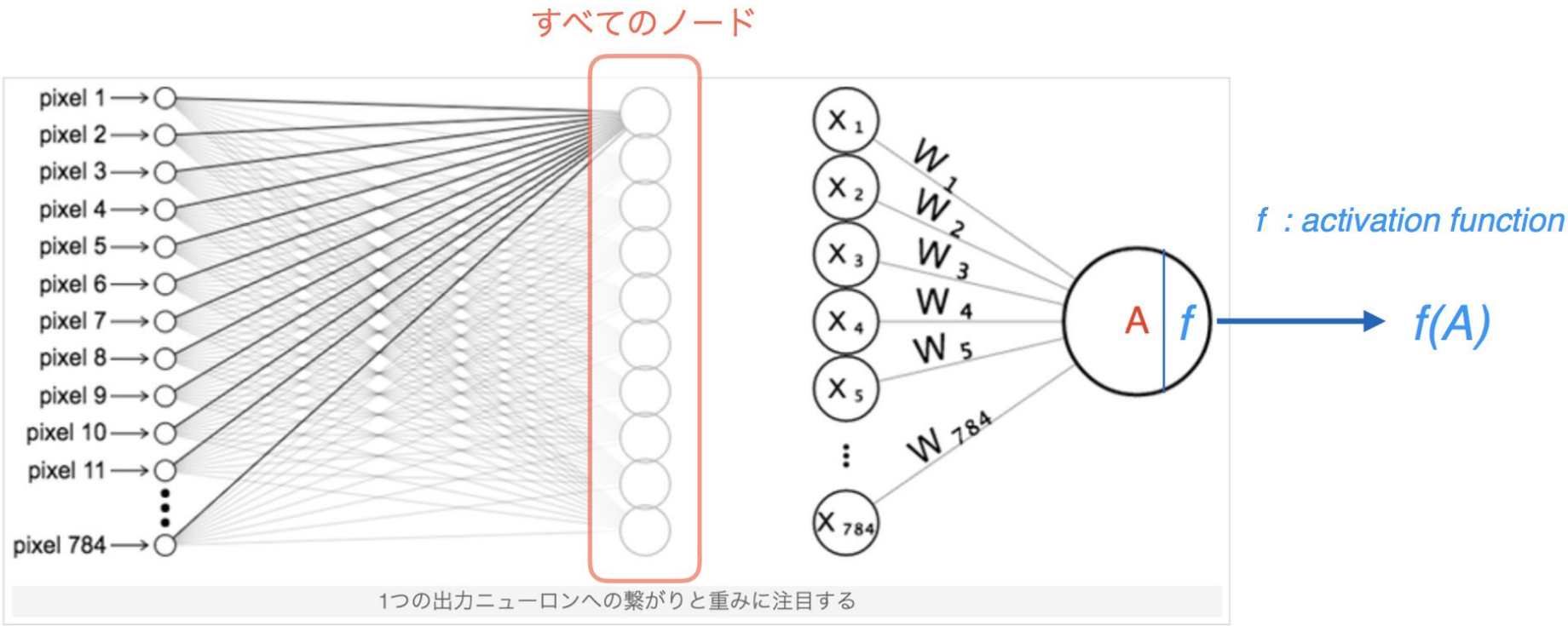
非線形変換とは、重みによって線形変換された入力を非線形の関数 (f)に通すことである。ニューラルネットワークの非線形関数は活性化関数と呼ばれる。

前提：（そもそも）

ひとつのノードでの計算は、ロジスティック回帰での出力と同様である。

一層だけのアーキテクチャ（中間層なしで出力ノード数を1とする）のニューラルネットワークを考え、活性化関数をシグモイド関数とすると、全体をロジスティック回帰モデルとみなすことができる。

$$Y = \mathbf{sigmoid}(W_0X_0 + W_1X_1 + W_2X_2 + \ldots + W_{784}X_{784})$$



活性化関数の種類

活性化関数の種類

多くの活性化関数は、入力に対して出力を単調増加させる(2)。判別可能（discriminatory）でシグモイド的な性質をもつ（sigmoidal）関数を用いることで、任意の（可測）関数を近似することができる。

(2) 単調増加であるとは、入力が大きければ出力もより大きくなることなので、結果としての値の順位が変わることはない。

アイデア：

今回のSprintではtanh関数とsoftmax関数を用いる。

多クラス分類では出力ノードの活性化関数としてsoftmaxを用いる。この関数は、0.0~1.0の値を返すのでこれを確率とみなすことができる。今回は10クラス分類を行うため、出力層には10個のノードを配置し各クラスに対応させる。これらの各ノードからの0.0~1.0の間の確率を足し合わせると全体として1となる。確率の高いノードが、予測したクラスに相当する。

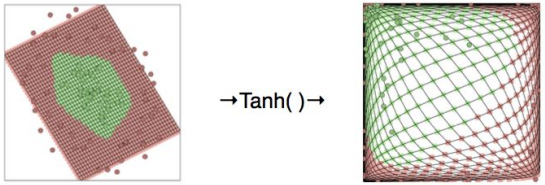
事例：

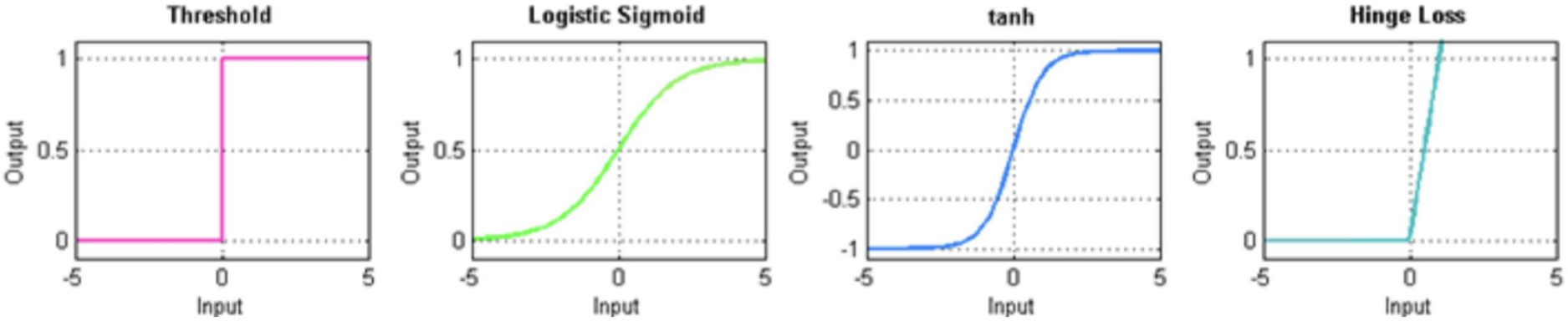
https://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html

前提：

隠れ層は、線形変換と非線形変換を行い、最終的にデータが**線形分離可能であるような表現**を出力する。

softmax関数を可視化する。→ https://qiita.com/rtok/items/b1affc619d826eea61fd





誤差を利用して学習する

誤差を利用して学習する

入力画像の真の値と予測結果との誤差を見積もって、
それを減少させるように学習する。

アイデア：
予測値 $\hat{y}$ と真の値 y (正解ラベル) の誤差をクラス分合計し、バッチサイズ（一回に学習するデータのサイズ）で平均をとり、誤差を評価する。

前提：
深層学習においても、これまでの機械学習と同様に、目的関数を最小化する最適化問題を解く。今回は分類問題のため、上の処理を行う目的関数は、Sprint3で用いたクロスエントロピー損失関数である。

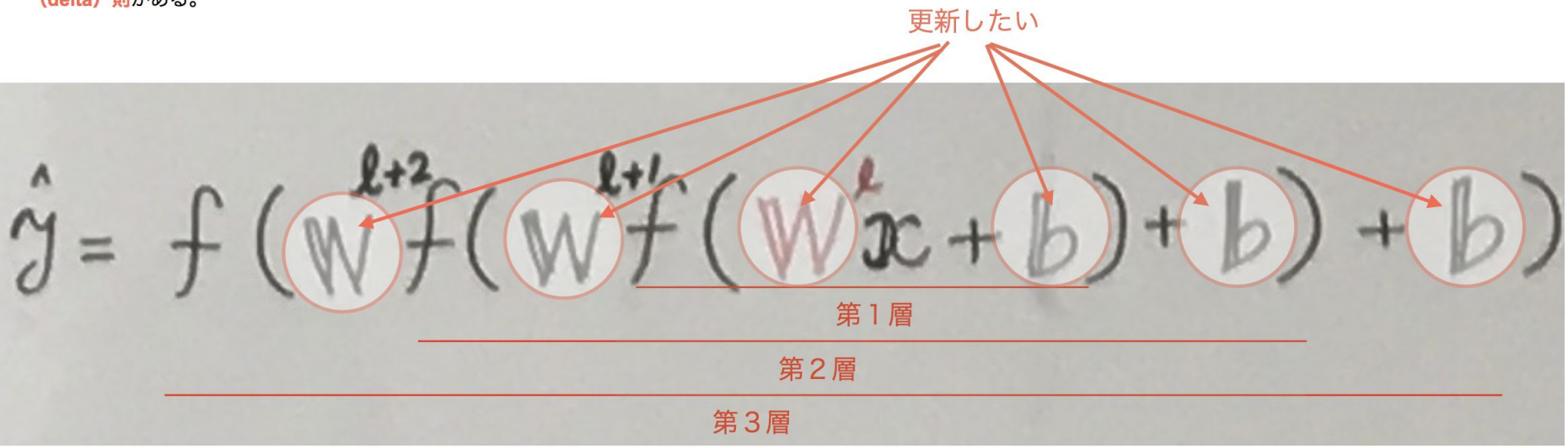
デルタ則

デルタ則

ニューラルネットワークでは、重み（結合荷重）を更新することを学習と呼ぶ。また、学習時における重みを更新するルールを**学習則** (learning rule) という。脳内の学習則はまだ明らかでないが、どのような場合にシナプスが強化されるかという条件は知られている。
ニューラルネットワークで有名な学習則に、「同時に発火したニューロン間のシナプス結合は強められる」ことを模倣した**ヘブ（Hebb）則**と、漸化式 $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta \boldsymbol{w}_t$ に従って逐次重みを更新する**デルタ（delta）則**がある。

アイデア：
隠れ層のない2層のニューラルネットワークについては、出力と真値の誤差から確率的勾配降下法（SGD）を用いて勾配を求め、その勾配によって重みを更新することができた（1960年）。SGDはこの当時、Widrow-Hoff法（あるいはデルタ則）と呼ばれた。

問題：
3層のニューラルネットワークにおいてもデルタ則で**最後の出力に対する誤差**は計算できそうだが、中間の層の出力に対する誤差を計算できない。従って、中間層の重みの勾配を求められないため、中間層の重みが更新できない。



3層以上のニューラルネットワークの出力は、多数の関数の合成になる

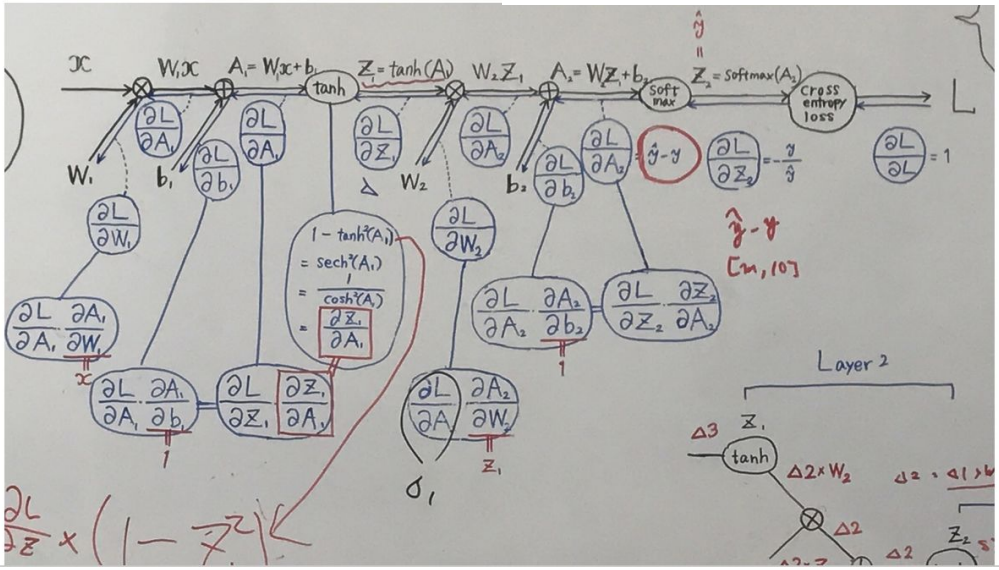
誤差逆伝搬法（一般化デルタ則）を用いる

誤差逆伝播法（一般化デルタ則）を用いる

誤差逆伝播法では、誤差を順伝搬の逆方向に流しながら、代数的に勾配を求めることができる。

1.ランダムな初期値で重みとバイアスの値を作成し、全てのユニットの出力を求める（ここまで順伝播）
2．全ての出力ユニットの誤差を求める（ここから逆伝搬）
3．全ての隠れユニットの誤差を求める
4．必要な微分を評価する

最終出力層が正準連結関数(3)ならば、誤差関数をこの最終出力層ユニットで微分した値は、最終出力と真値の差分にzをかけた形式($\hat{y}$ - y) *zになる
したがって、誤差逆伝播の**伝搬のスタートにおける誤差を $\hat{y}$ - y として**伝搬を始めればよい。
(3) 連結関数（活性化関数の逆関数）の一種。 https://www.slideshare.net/satoshikawamoto77/prml-436



合成関数の微分について

合成関数の微分はチェーンルール（連鎖律）で計算する

チェーンルールとは、合成関数の微分を各々の関数の偏微分の積で求める関係式のことである。
微分のチェーンルールによって、層ごとの誤差（部分）を求めることができる。求めた誤差は、順伝搬の逆方向へ流していく。

合成関数の偏微分の例

$$\frac{d((f \circ g)(x))}{dx} = \frac{d(f(g(x)))}{d(g(x))}\frac{d(g(x))}{dx}$$

$z = f(y), y = g(x)$ とすると、

局所的勾配

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

更新用の勾配

上流の勾配
= デルタΔ

$$y = f(u), u = g(x), y = f(g(x)) \ldots (1)$$

$$\frac{dy}{dx} = \frac{dy}{du} \bullet \frac{du}{dx} \ldots (2)$$

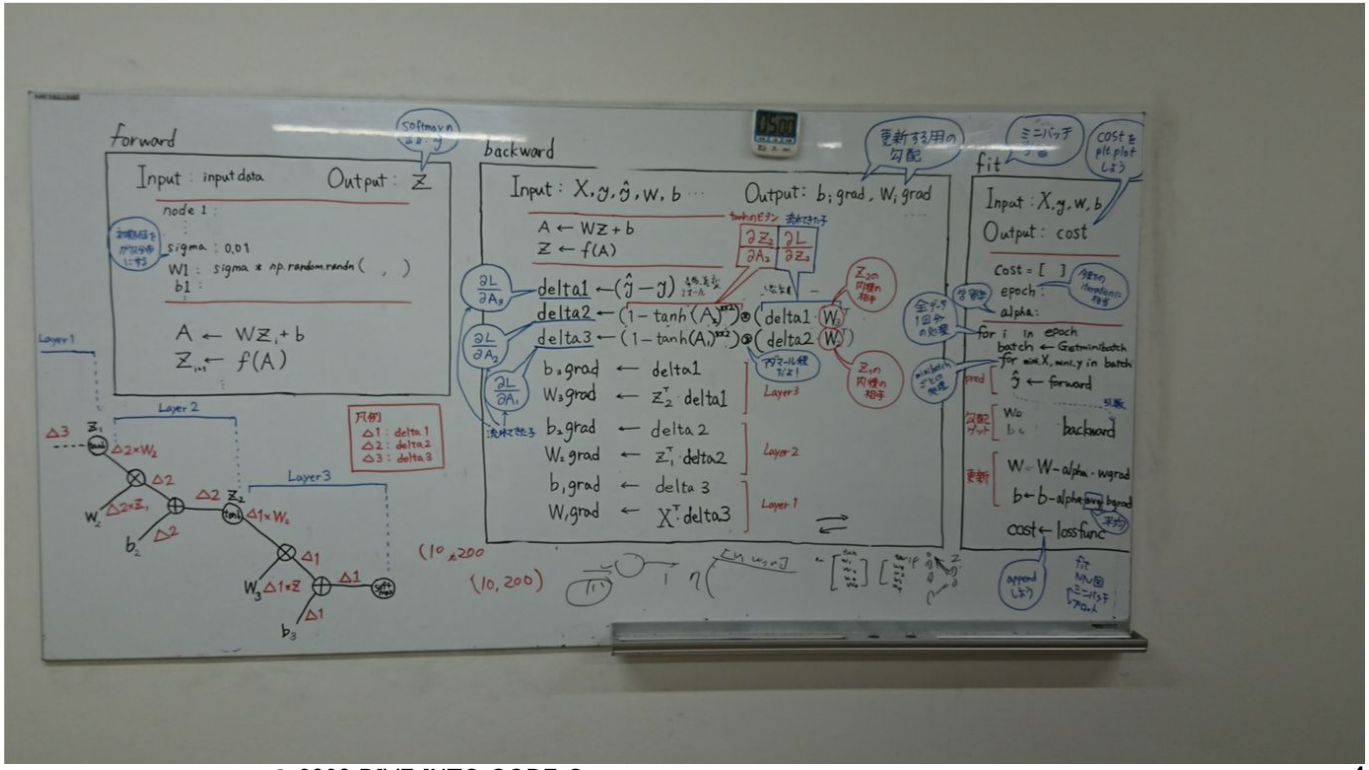
$$z = f(y1, y2, \cdots, ym), yi = gi(x1, x2, \cdots xn) \ldots (3)$$

$$\frac{\partial z}{\partial xi} = \frac{\partial f}{\partial y1} \bullet \frac{\partial y1}{\partial xi} + \frac{\partial f}{\partial y2} \bullet \frac{\partial y2}{\partial xi} + \cdots + \frac{\partial f}{\partial ym} \bullet \frac{\partial ym}{\partial xi} \ldots (4)$$

計算グラフでデルタを追う

計算グラフでデルタを追う。

計算グラフで考えると、デルタΔ（上流の勾配）はフォワードの逆方向に流れてゆく。
フォワードにおいて、dot積したフェーズでは、元々dot積の計算相手であった値を、
流れてきたデルタΔに対してdot積演算すると更新用の勾配が得られる。一方、フォ
ワードで足し算だったフェーズでは、流れてきたデルタΔをそのまま更新用の勾配とす
る。さらに、フォワードにおける活性化関数のフェーズでは、その活性化関数の微分を
流れてきたデルタΔに対しアダマール積することによって、新しいデルタΔが得られる。



ポイント

1. ニューラルネットワーク
   重み、バイアス、ノード、活性化関数

2. バックプロパゲーション
   クロスエントロピー損失関数（目的関数）、
   Loss、勾配

3. フルバッチ、ミニバッチ、オンライン
   イテレーション、エポック、
   バッチサイズ（学習のデータサイズ）

ニューラルネットワーク
完