

最適化理論

—深層ニューラルネットワーク—

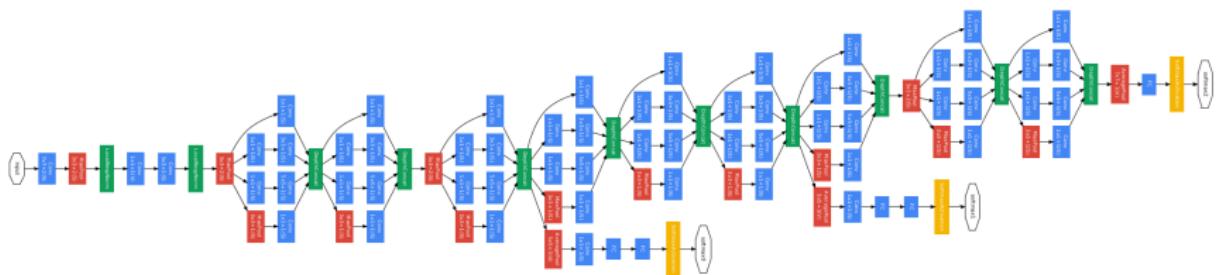
本講義の概要

- ▶ 深層学習の概要
- ▶ ニューラルネットワークの研究の進展を辿る
- ▶ 順伝播型ネットワーク

深層学習の登場と隆盛の例: GoogLeNet

Google 社が開発した深層畳込みニューラルネットワーク

- ▶ Deep Convolutional Neural Network (DCNN)
- ▶ ILSVRC2014(詳細は後ほど) で首位
- ▶ 1000 クラスの画像認識を行う



ImageNet

スタンフォード大学が ImageNet を立ち上げる

- 現在 2 万カテゴリのラベル付けがされた , 1400 万枚の画像が集められている
- すべて人工によるラベル付け



ILSVRC(ImageNet Large Scale Visual Recognition Challenge)

- 画像認識の精度を競うコンペティション

全部で 1000 カテゴリの画像が用意され，各テスト画像に含まれている対象のカテゴリをどれだけ正しく認識できるかを競う

使用データ

- ▶ 学習用画像：120 万枚
- ▶ 確認用画像：5 万枚
- ▶ テスト用画像：15 万枚



画像認識精度

GoogLeNet は 7 つのモデルを用いたアンサンブル推論を行う

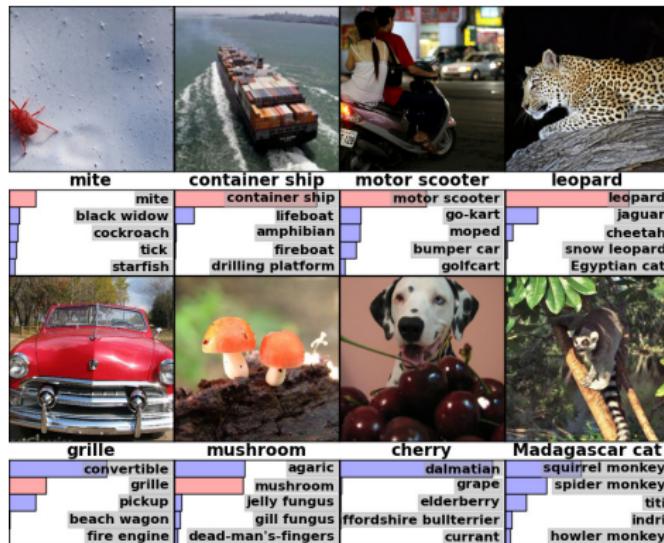
- ▶ サンプリング方法と入力画像の順序のみ異なる

Team	Year	Place	Error(top-5)
SuperVision	2012	1st	16.4%
Clarifai	2013	1st	11.7%
MSRA	2014	3rd	7.35%
VGG	2014	2nd	7.32%
GoogLeNet	2014	1st	6.67%
GoogLeNet(1)	2014	–	7.89%

top-5 エラー率

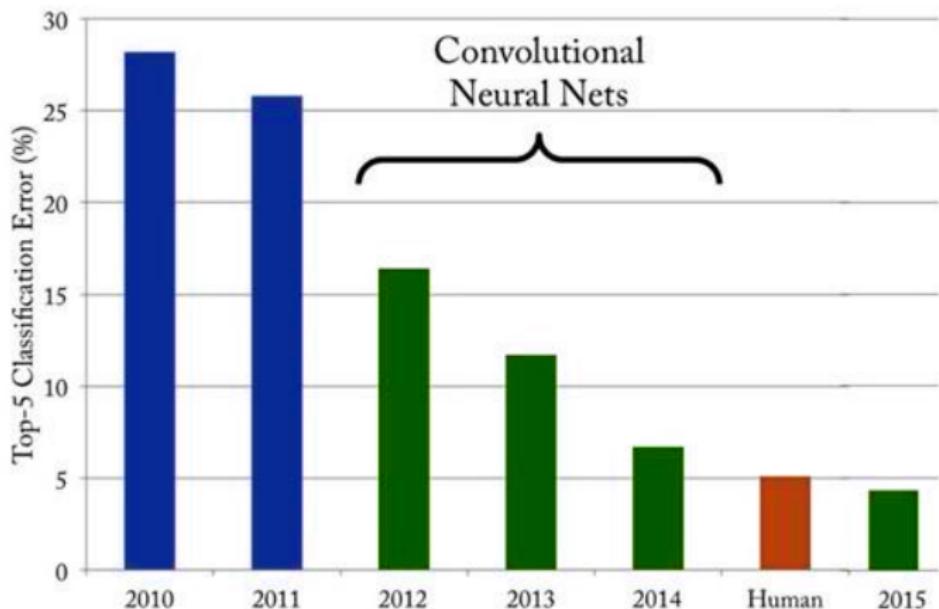
top-5 エラー率

認識システムが候補としてあげた上位 5 つの答えの中に正解が含まれていれば、認識成功
ILSVRC では、このエラー率を競う



画像認識率の進歩

ImageNet Classification (2010 – 2015)



ジェフリー・ヒントン



<https://t.co/T7bloktxne>

AlphaGo の登場



(左)2016年 AlphaGo と対戦したイ・セドル
(右)DeepMind のデミス・ハサビス

AlphaGo の詳細

AlphaGo プロ棋士を破った世界初の囲碁のコンピュータープログラム

モンテカルロ木探索とディープラーニングを組み合わせたシステム

3週間、50GPUを用いて3億4千万回のトレーニング・ステップでDNNをトレーニング

対局: 40の検索スレッド、1202個のCPUと176個のGPUを使用して計算

バリューネットワークで基盤の状況を予測し、ポリシーネットワークで手の評価を行う

全5局の対局、囲碁のヨーロッパチャンピオンと世界チャンピオンに勝利

<http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>

<http://deepmind.com/alpha-go.html>



音声認識

Baidu Deep Speech 2

英語と中国語のディープラーニングを用いたエンドツーエンド音声認識

英語と中国語(北京語)の音声認識



シンプルにエンドツーエンドのディープラーニングを用いて英語から中国語への変換を行う

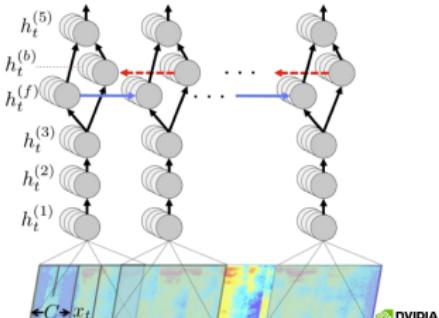
人間より高い認識精度

エラー率(Deep Speech 2) 3.7%

エラー率(人間へのテスト) 4%

<http://svail.github.io/mandarin/>

<http://arxiv.org/abs/1512.02595>

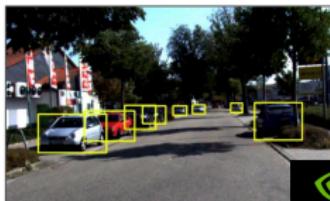
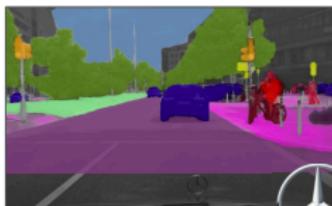


自動運転

自律走行車の為のディープラーニング



Audi



画像合成

ディープラーニングを用いた画像合成



CNNを用いたテクスチャ合成と変換 NVIDIA Research、Timo Aila 他

ゲームの自動プレイ（深層強化学習）

ディープラーニングの事例

ゲームプレイ



Figure 1: Screen shots from five Atari 2600 Games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider



Google DeepMind

ニューラルネットワーク研究の流れ

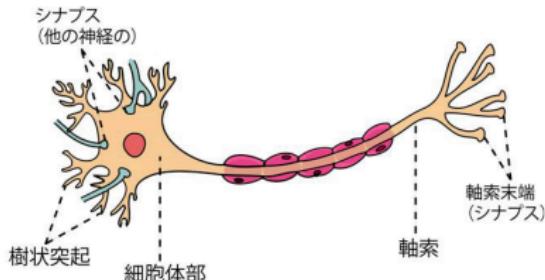
本講義では深層学習の導入として、ニューラルネットワーク研究の歴史を振り返るとともにニューラルネットワークに関する基礎的な概念を紹介していきます。

- ▶ ニューラルネットワークの創始 (1940 年代)
- ▶ 第一次ブームの終焉
- ▶ ニューラルネットワークの第二次ブーム (80 年代後半から 90 年代前半)
- ▶ 第二次ブームの終焉
- ▶ 深層ネットワークの登場 (第三次ブーム)

形式ニューロンの登場

- ▶ 脳神経学者のマカロックと数学者のピットは、神経細胞を模した「形式ニューロン」を導入した。(McCulloch, W. and Pitts, W. (1943)).
- ▶ 現実のニューロンの振る舞いを簡略化して得られた
- ▶ 形式ニューロンは、現実のニューロンが「発火するか(1)」または「発火しないか(0)」という2状態を持つとしてモデル化を行っている。

ニューロン（脳の神経細胞）



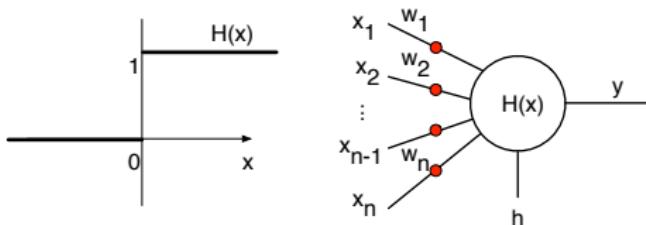
出典

<https://qiita.com/nishiy-k/items/1e795f92a99422d4ba7b>

形式ニューロンの定義とその能力

形式ニューロンは、ステップ関数と線形重み付け計算の組み合わせにより構成される。

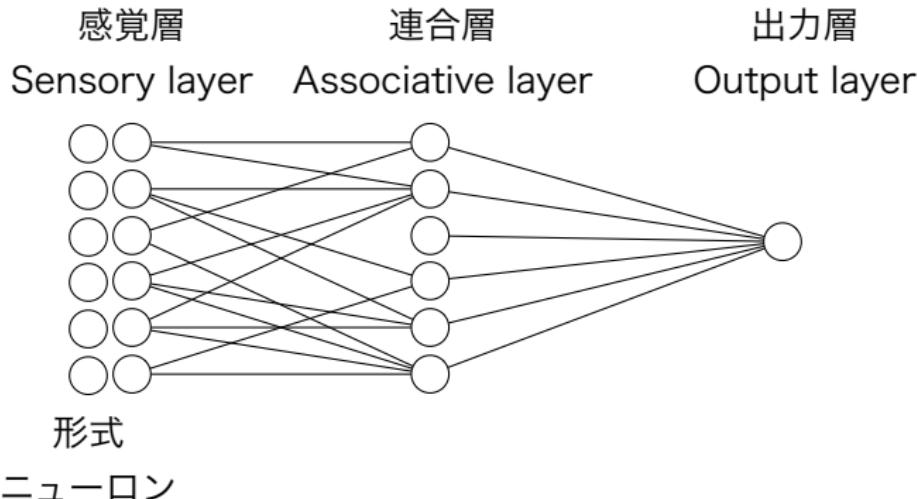
$$y = H \left(\sum_{i=1}^n w_i x_i - h \right) \quad (1)$$



単純なモデルだが、複数の形式ニューロンを組み合わせる強力な
能力を持たせることができる

パーセプトロンの登場

ローゼンブラット (1958) は、形式ニューロンをもとにした3層の階層型ネットワーク (パーセプトロン) を提案した。



- ▶ パターン認識問題への応用を念頭においている
- ▶ 多層型・順伝播型アーキテクチャの元祖
- ▶ ミンスキー・パパートにより線形非分離な問題が扱えないことが示される 第一次ブームの終焉

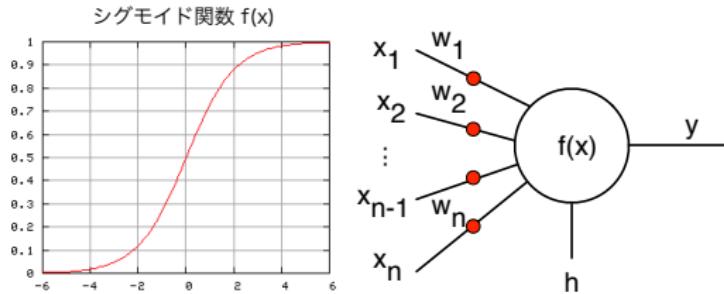
ニューラルネットワークの登場

人工ニューロン素子は次の構造を持つ:

$$y = f \left(\sum_{i=1}^n w_i x_i + b \right) \quad (2)$$

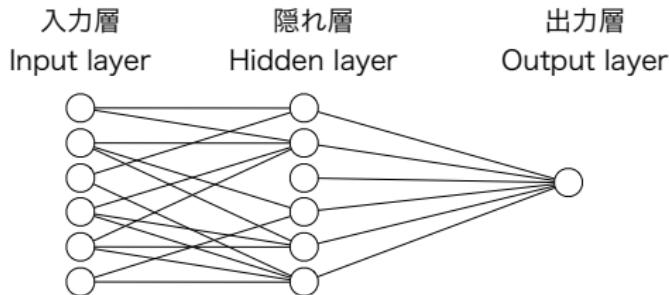
関数 f は活性化関数 (activation function) と呼ばれる関数で、1980 年代には、シグモイド関数 (sigmoid function) が利用されることが多かった。(出力が実数値になったことに注意!)

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3)$$



3層ネットワークの隆盛 (1980年代)

- ▶ 人工ニューロンからなる3層順伝播型ネットワークの優れたパターン認識性能が知られてきた。



- ▶ 活性化関数が微分可能な関数となったことで、 w_i や b の微分を考えることができるようになった → 勾配法による学習パラメータの更新
- ▶ ルメルハートらによる逆誤差伝播法 (1986) により、学習パラメータの勾配ベクトル (gradient) の高速計算が可能に。
- ▶ 3層以上のネットワークの学習 → 勾配消失による学習の困難性の認識が広がった → 研究活動の沈滞 (2回目の冬)

順伝播型ネットワーク

ここでは、ニューラルネットワークの基礎となる順伝播型ネットワークについて、その詳細を見ていきます。

人工ニューラル素子(ふたたび)

人工ニューロン素子は次の構造を持つ:

$$y = f \left(\sum_{i=1}^n w_i x_i + b \right) \quad (4)$$

関数 f は活性化関数であり、さまざまな種類の関数が利用されている:

シグモイド関数

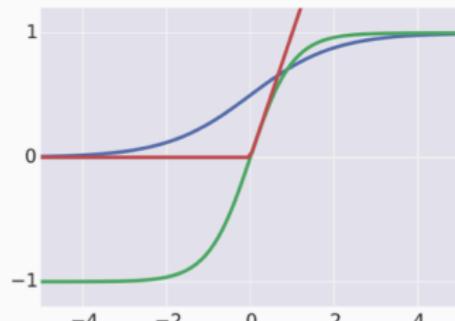
$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

双曲線正接関数

$$\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

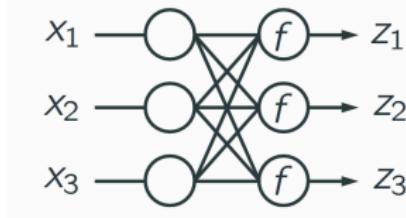
ランプ関数

$$\text{ReLU}(u) = \max(0, u)$$



レイヤー(層)

順伝播型ネットワークでは、複数のレイヤーが直列に接続される。ひとつのレイヤーは次のような形状をしている：



ここで、レイヤーへの入力ベクトルを $x = (x_1, x_2, \dots, x_n)^T$ 、出力ベクトルを $z = (z_1, z_2, \dots, z_m)^T$ とするとき、

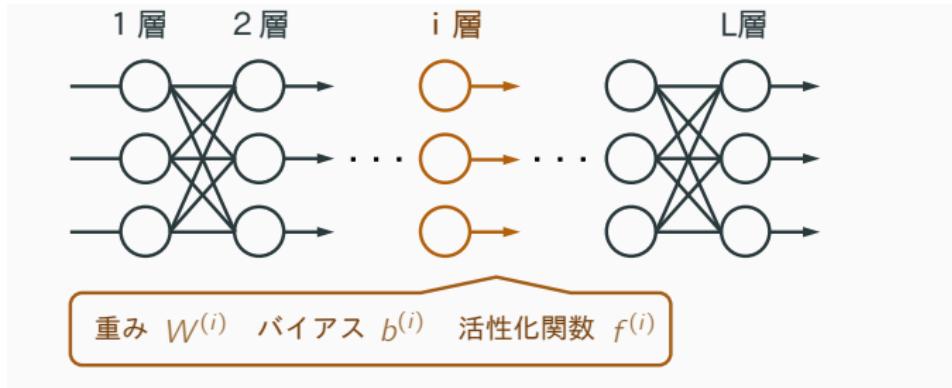
$$z = f(Wx + b) \quad (5)$$

と書くことができる。ここで、

$$W = \begin{pmatrix} W_{1,1} & \dots & W_{1,n} \\ W_{2,1} & \dots & W_{2,n} \\ \vdots & \vdots & \vdots \\ W_{m,1} & \dots & W_{m,n} \end{pmatrix}, \quad b = (b_1, b_2, \dots, b_m)^T \quad (6)$$

である。

順伝播型ネットワーク



$$z^{(i)} = f^{(i)} \left(W^{(i)} z^{(i-1)} + b^{(i)} \right) \quad (7)$$

第1層(入力層)の入力を x 、第 L 層(出力層)に出力を y とするとき、順伝播型ネットワーク全体をベクトル入力・ベクトル出力関数として、

$$y = F(x) \quad (8)$$

という形で表すことができる。

ネットワークの含む学習パラメータ

順伝播型ネットワーク $y = F(x)$ は、調節可能なパラメータ 1

$$\Theta = \left\{ W^{(1)}, \dots, W^{(L)}, b^{(1)}, \dots, b^{(L)} \right\}$$

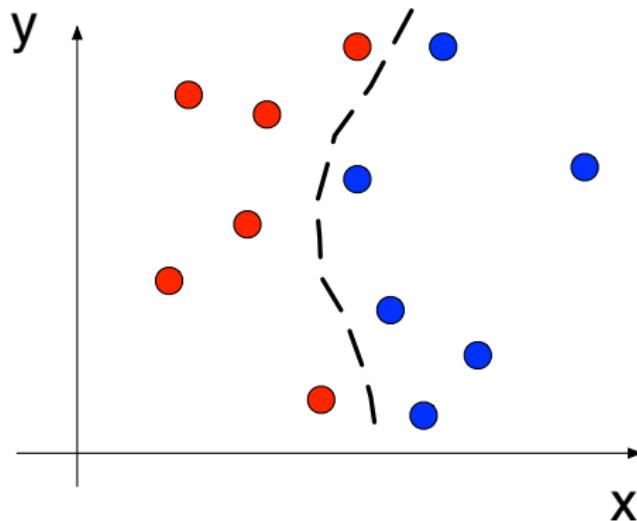
を持つ。これらの学習パラメータを訓練データに従い、調節することを「学習」または「訓練(トレーニング)」という。パラメータを明示したい場合には、

$$y = F(x; \Theta) \tag{9}$$

と表記する場合もある。学習パラメータの調節(学習)においての目標は、**損失関数値(クラス判別問題)**・**誤差関数値(回帰問題)**を小さくすることにある。

クラス判別問題 (2値判別を例として)

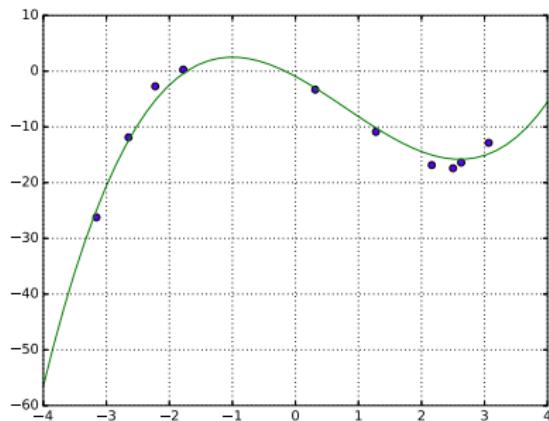
各データ (x, y) の組は、ラベル（赤、青）を持つ。訓練データは $(x_i, y_i, t_i) (i = 1, 2, \dots, T)$ の形式である。 t_i はラベルを意味する。



順伝播型ネットワークは、未知のデータ (x, y) に対して、そのデータが属するクラス (= ラベル) を可能な限り正しく推定するように学習したい クラス判別問題

回帰問題

- ▶ いま、未知の関数 $G(x)$ からひとつの観測データ (x^*, y^*) が得られたものとしよう。ここで、 $y^* = G(x^*)$ である。
- ▶ 順伝播ネットワーク $F(x; \Theta)$ を利用して、関数 $G(x)$ を近似したい。
- ▶ 未知関数の近似問題 **回帰問題**



回帰問題はどんなところで有用か？

例えば、興味のある対象について、過去のデータから将来の観測値を予測する、という使い方ある。

例 1: 株価予測

訓練集合 第 i 日の円ドル為替レート、第 i 日のX社の株価、
そのほか経済指標

予測目標 将来ある時点でのX社の株価

例 2: 自動車事故率(保険会社の視点から)

訓練集合 契約者の年齢、過去の事故歴、車種

予測目標 将来のある人の事故率

誤差関数の最小化

二乗誤差関数

$\mathbf{x} = (x_1, \dots, x_n)^T$, $\mathbf{y} = (y_1, \dots, y_n)^T$ に対して定義されるスカラーフィルタ

$$E(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n (x_i - y_i)^2 \quad (10)$$

を二乗誤差関数と呼ぶ。

学習の方針 ($G(\mathbf{x})$ の近似問題 = 回帰問題)

- ▶ 観測データ (= 訓練データ) $(\mathbf{x}^*, \mathbf{y}^*)$
- ▶ 順伝播型ネットワーク $\mathbf{y} = F(\mathbf{x}; \Theta)$

学習パラメータ Θ の最適化

$$E(\mathbf{y}^*, F(\mathbf{x}^*; \Theta))$$

を最小化するように Θ を定める。(この場合、この学習フレームワークは、「最小二乗法」と呼ばれる)

最小化の手法: 勾配法 (gradient descent method)

制約無し最適化問題

$$\text{minimize}_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad (11)$$

勾配法のステップ

Step 1 (初期点設定) $\mathbf{x} := \mathbf{x}_0$

Step 2 (勾配の計算) $\mathbf{g} := \nabla f(\mathbf{x})$

Step 3 (探索点更新) $\mathbf{x} := \mathbf{x} - \alpha \mathbf{g}$ (α は学習係数)

Step 4 (反復) Step 2 に戻る

(注)

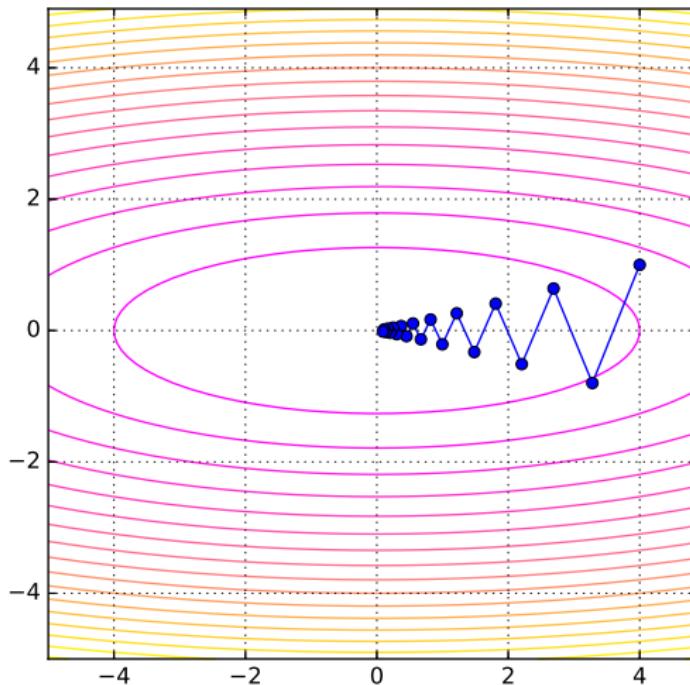
- ▶ $\nabla f(\mathbf{x})$ は勾配ベクトル (gradient vector) である。例えば、 $f(x_0, x_1)$ の場合、

$$\nabla f(x_0, x_1) = \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \right)^T$$

である。勾配ベクトルは f の等高線に直交する。

勾配法における探索点の軌跡

$$f(x) = \frac{1}{2}x_0^2 + 5x_1^2, \quad x_0 = (4, 1)^T, \alpha = 0.18$$



勾配法（雛形プログラム）

勾配法の実装例を示す (python)

```
# -*- coding: utf-8 -*-
import numpy as np
#  $y = x^2 + y^2$  を目的関数とする。
def grad(x):                                # 勾配
    return 2.0*x
xt = np.array([5,5])                          # 初期点
alpha = 0.05                                  # ステップ係数
for i in range(100):                         # メインループ
    xt = xt - alpha*grad(xt)
    print "step=", i, xt
print
```

順伝播型ネットワークの学習の流れ (回帰) の基本形

- ▶ 訓練データを用意する。訓練データ: (入力値のベクトル、出力値のベクトル)
- ▶ 誤差関数を用意する。
- ▶ 順伝播型ネットワークを用意する。

$$z^{(i)} = f^{(i)} \left(W^{(i)} z^{(i-1)} + b^{(i)} \right)$$

- ▶ 勾配法を利用して誤差関数值を最小とする学習パラメータ Θ^* を求める (以上が「訓練 (training) フェーズ」)。
- ▶ $F(x; \Theta)$ を利用して、予測計算などを行う (これは「推論 (inference) フェーズ」と呼ばれることが多い)

1980 年代までの 3 層ネットワークの学習には、おおむね上記の手順が利用されていた。現在の深層学習においても「学習の大枠」は以上の通りである。

単純な学習手法が抱えるいくつかの困難

層の数を増していくことで、順伝播ネットワークの性能は向上していきそうな雰囲気は感じていた（80年代初頭）ところが。。。

- ▶ 過学習
- ▶ 勾配ベクトルの計算がかなり大変（計算量的な意味で）
- ▶ 勾配消失問題（入力層に近いところの学習パラメータの微分値がほとんどゼロになる）
- ▶ 学習プロセスの停滞（プラトーに陥る。。。）

3層より多層のネットワークの学習は、ほとんど望みがなさそうに思われた 第2次ブームの終焉

単純な学習手法が抱えるいくつかの困難

層の数を増していくことで、順伝播ネットワークの性能は向上していきそうな雰囲気は感じていた(80年代初頭)ところが。。。

- ▶ 過学習
- ▶ 勾配ベクトルの計算がかなり大変(計算量的な意味で)
- ▶ 勾配消失問題(入力層に近いところの学習パラメータの微分値がほとんどゼロになる)
- ▶ 学習プロセスの停滞(プラトーに陥る。。。)

80年代後半の進展、現在の深層学習において利用される技術により、解決(もしくは部分的に解決)

勾配ベクトルの計算における困難

極端に簡単化した順方向ネットワークを考える。

$$F(x; A) = a_3 f(a_2 f(a_1 f(x)))$$

という関数が与えられている。ここで、 $A = \{a_1, a_2, a_3\}$ である。



$$\frac{\partial F(x; A)}{\partial a_1}$$

を求めよ。

合成関数の微分(微分の連鎖律)

$$y = f(g(x))$$

について、 $\frac{\partial y}{\partial x}$ を求めたい。このとき、まず

$$u = g(x) \tag{12}$$

$$y = f(u) \tag{13}$$

と分ける。このとき、合成関数の微分公式は(微分の連鎖律)は次のとおり:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \times \frac{\partial u}{\partial x} \tag{14}$$

連鎖律にもとづき計算してみる

$F(x; A) = a_3 f(a_2 f(a_1 f(x)))$ を

$$u_1 = a_1 f(x) \quad (15)$$

$$u_2 = a_2 f(u_1) \quad (16)$$

$$u_3 = a_3 f(u_2) \quad (17)$$

と書き換える。ここで、連鎖律を使うと

$$\frac{\partial u_3}{\partial a_1} = \frac{\partial u_3}{\partial u_2} \times \frac{\partial u_2}{\partial u_1} \times \frac{\partial u_1}{\partial a_1} \quad (18)$$

である。ここで、

$$\frac{\partial u_1}{\partial a_1} = f(x), \quad \frac{\partial u_2}{\partial u_1} = a_2 f'(u_1), \quad \frac{\partial u_3}{\partial u_2} = a_3 f'(u_2)$$

より、

$$\frac{\partial u_3}{\partial a_1} = a_3 f'(u_2) a_2 f'(u_1) f(x) \quad (19)$$

を得る。

前向き・後ろ向き計算 (バックプロパゲーションの離形)

$$u_1 = a_1 f(x) \quad (20)$$

$$u_2 = a_2 f(u_1) \quad (21)$$

$$u_3 = a_3 f(u_2) \quad (22)$$

$$\frac{\partial u_3}{\partial a_1} = a_3 f'(u_2) a_2 f'(u_1) f(x) \quad (23)$$

前向き計算



$$f(x)$$

$$a_2 f'(u_1)$$

$$a_3 f'(u_2)$$

後ろ向き計算

逆誤差伝播法 (バックプロパゲーション)

80年代に開発された (Rumelhart ら) 学習パラメータの勾配ベクトルの高速算法

- ▶ 前述の例のように「合成関数の微分に関する連鎖律」を利用
- ▶ 前向き計算(推論計算)と後向き計算(誤差逆伝播計算)を順に実行
- ▶ 計算結果として得られる学習パラメータの勾配ベクトルを利用して、学習パラメータを更新(勾配法)
- ▶ 深層学習の学習計算で一番計算時間がかかっている部分である。
- ▶ 幸いなことに、最近のフレームワーク (TensorFlow, Chainer, Caffe など) では、前向き計算の記述をすれば、逆伝播計算は自動で行ってくれる。

深層学習向けのハードウェア

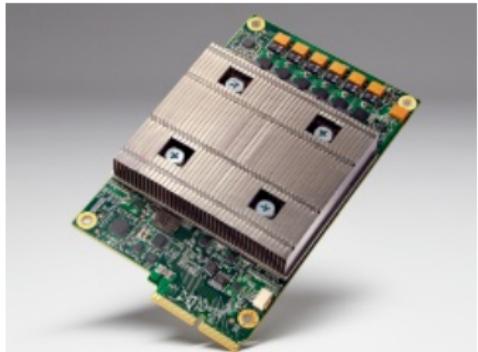
推論フェーズ(前向き計算)、逆伝播フェーズ(後ろ向き計算)のいずれにも大量の行列ベクトル積計算(より正確にはテンソル積計算)が必要とされる。汎用CPU以上に効率の良い計算が可能なGPU、または、専用のハードウェアへの注目が集まっている。

NVIDIA TESLA GPU



出典 <http://www.nvidia.co.jp/object/tesla-servers-jp.html>

Google Tensor Processing Unit (TPU)



出典 <http://itpro.nikkeibp.co.jp/atcl/ncd/14/457163/052001464/>

この講義のまとめ

- ▶ ニューラルネットワークの進展を辿る
- ▶ 順伝播型ネットワーク
- ▶ 深層学習を支える技術

参考文献

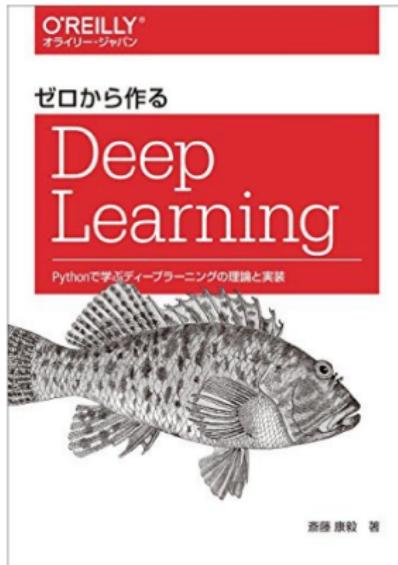
岡谷貴之、「深層学習」、講談社



- ▶ コンパクトに必要な事項がまとまっており、最初の一冊に好適
- ▶ 正確な記述

参考文献

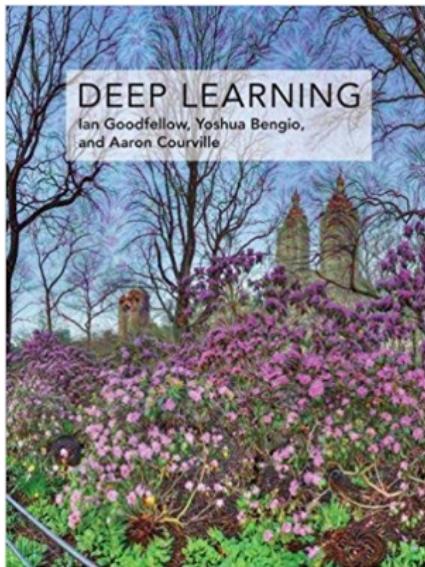
斎藤康毅、「ゼロから作る Deep Learning」、オライリー・ジャパン



- ▶ コードを書いて理解していく、というスタンス
- ▶ フレームワークも含めて全部書きたい、という場合に特に役立ちます。

参考文献

Ian Goodfellow, Yoshua Bengio, Aaron Courville、「Deep Learning (Adaptive Computation and Machine Learning series)」、MIT Press



- ▶ 世界的にみて、教科書の定番
- ▶ 深層ネットワーク研究の第一人者である Yoshua Bengio 教授のグループが執筆