実践プログラミングセミナー

TensorFlowによるニューラルネットワーク入門

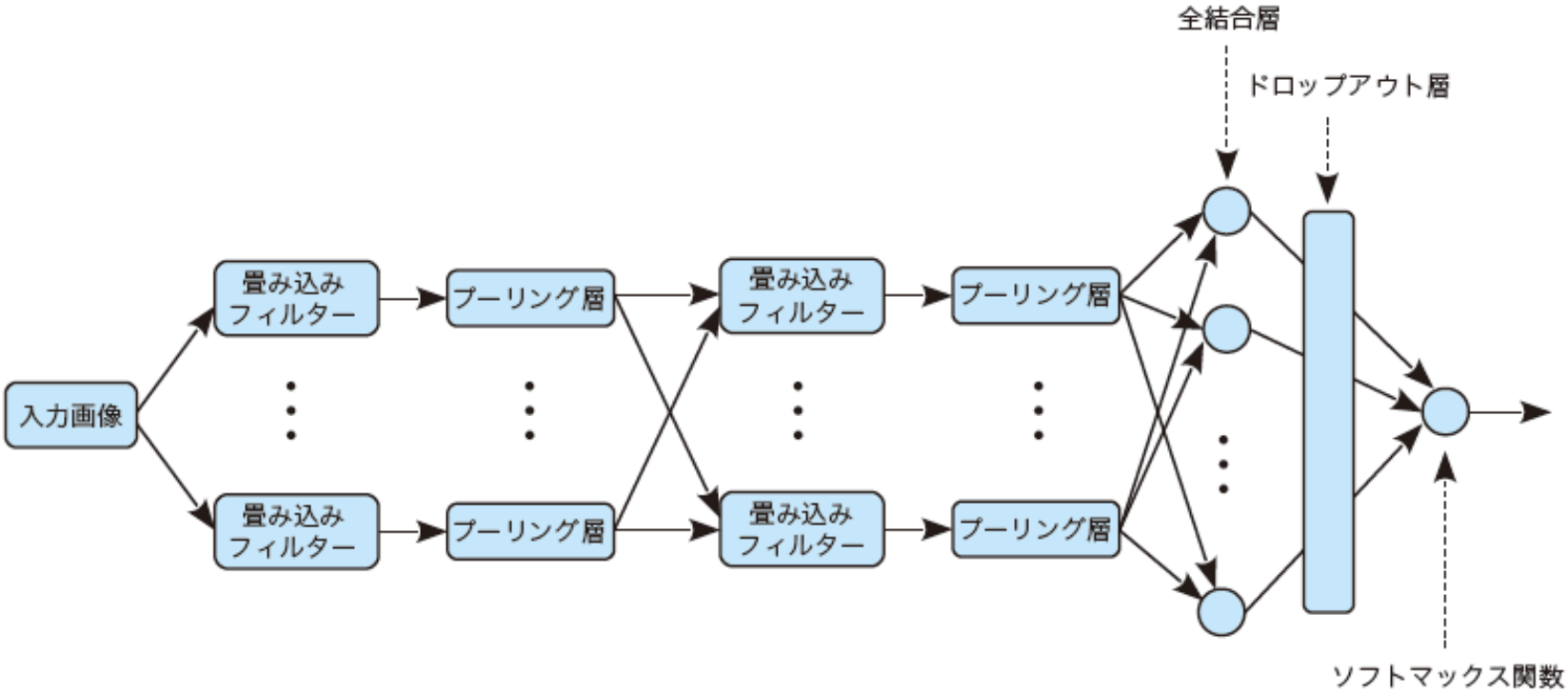
Etsuji Nakai (twitter @enakai00)
ver1.3 2017/01/24

目次

- 第１部　予備知識
  - ハンズオン環境の利用方法
  - データサイエンスと機械学習
  - 機械学習アルゴリズムの分類
  - ロジスティック回帰と最尤推定法
  - TensorFlowのコードの書き方
- 第２部　畳み込みニューラルネットワーク入門
  - 線形多項分類器
  - 多層ニューラルネットワークによる特徴抽出
  - 畳み込みフィルターによる画像の特徴抽出
  - 畳み込みフィルターの動的な学習

はじめに

- この資料では、下図の「畳み込みニューラルネットワーク」の構造を説明しています。右側のソフトマックス関数から順に、全結合層、「畳み込みフィルター＋プーリング層」へとそれぞれの役割を解説します。



参考資料

- 「ITエンジニアのための機械学習理論入門」中井悦司（技術評論社）
- 「TensorFlowで学ぶディープラーニング入門」中井悦司（マイナビ出版）
- 「戦略的データサイエンス入門」Foster Provost, Tom Fawcett（オライリージャパン）
- 「Pythonによるデータ分析入門」Wes McKinney（オライリージャパン）
- Python 機械学習プログラミング データ分析ライブラリー解説編
  - http://www.slideshare.net/enakai/it-numpy-pandas

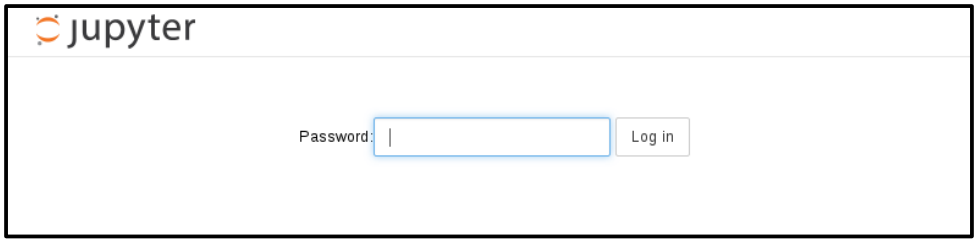
ハンズオン環境の利用方法

ハンズオン環境の利用方法

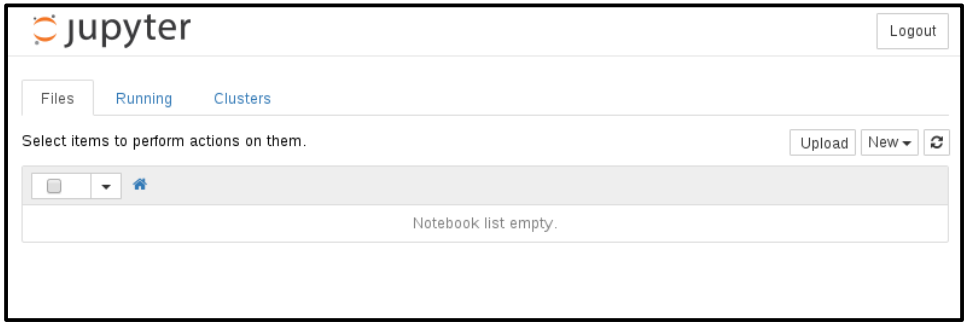
- 本講義のハンズオン環境は、Dockerコンテナを用いて提供されます。下記のBlog記事の手順にしたがって、同じ環境を自分で用意することもできます。
  - Jupyter演習環境の準備手順
    - http://enakai00.hatenablog.com/entry/2016/11/18/134724

ハンズオン環境の利用方法

- 指定のURLにアクセスするとパスワード入力画面が表示されるので、指定のログインパスワードを入力します。



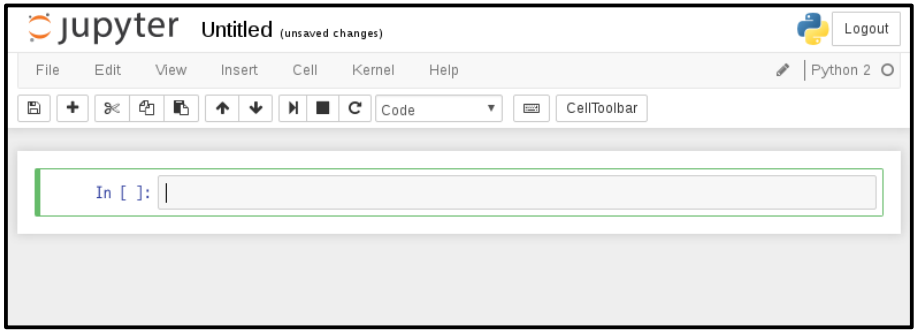
- ログインすると「ノートブックファイル」の一覧画面が表示されますが、今はまだノートブックファイルはありません。



ハンズオン環境の利用方法

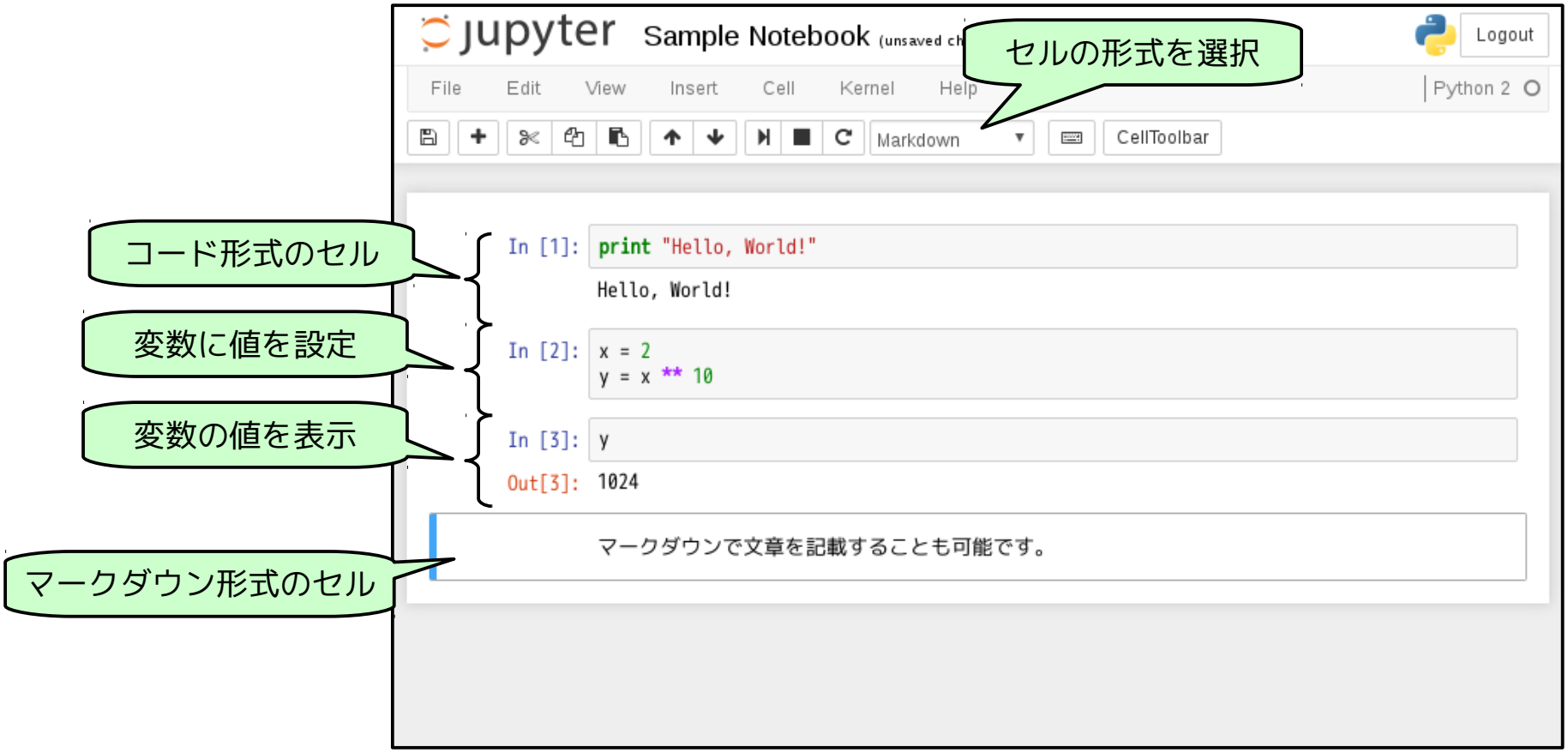
- 右上のプルダウンメニューから「New」→「Python 2」を選択すると新しいノートブックが開きます。

- タイトル部分をクリックすると、新しいタイトルが設定できます。「<タイトル>.ipynb」が対応するファイル名になります。（タイトルには日本語は使えません。）



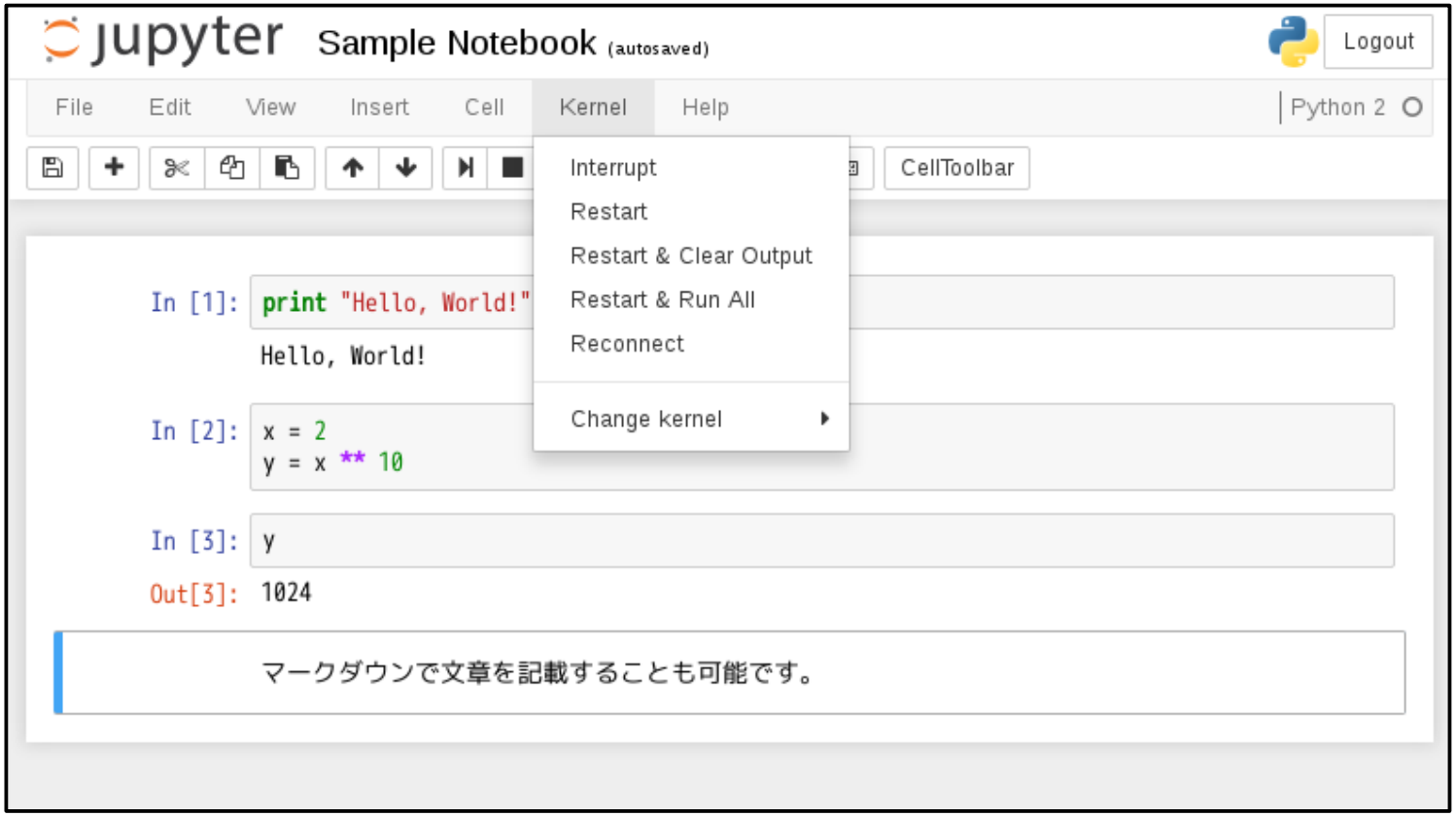
ハンズオン環境の利用方法

- ノートブック上では、セルにプログラムコードを入力して、「▶」ボタン、もしくは [Ctrl] + [Enter] で実行すると結果が表示されます。
- マークダウン形式のセルには説明文を記載することができます。



ハンズオン環境の利用方法

- ノートブック全体を最初から実行し直す場合は、メニューから「Kernel」→「Restart & Clear Output」を選択して、これまでの実行内容をクリアします。



サンプルコードの入手について

- 空のセルで次のコマンドを実行すると、本講義のサンプルノートブックがダウンロードできます。

```
!git clone https://github.com/enakai00/jupyter_tfbook
```

  - ノートブックファイルの一覧画面にフォルダー「jupyter_tfbook」が作成されるので、その中のノートブックを参照します。
  - 下記のモジュールをインポートしている前提でコードの解説を行います。

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import multivariate_normal, permutation
import pandas as pd
from pandas import DataFrame, Series
```

メモとしてお使いください

データサイエンスと機械学習

人工知能とは？！

人工知能

人工知能（じんこうちのう、英: artificial intelligence、**AI**）とは、人工的にコンピュータ上などで人間と同様の知能を実現させようという試み、或いはそのための一連の基礎技術を指す。

目次 [非表示]

E. Nakai
@enakai00

『AIで株価は予想できるのか』⇒
「AI（知性を持ったかのように見える製品やサービス）を実現するために研究・活用が進んでいる機械学習を基礎とした一連のデータ収集・分析技術は株価予測に有効活用できるのか」

7 リツイート　6 いいね

8:25 - 2016年10月19日

データサイエンスと機械学習の関係

- データサイエンスとは？
  - 過去のデータに基づいて、科学的な手法でビジネス判断を行う取り組み
  - 過去のデータから「過去の事実」を明らかにするのは簡単
  - 過去のデータから「未来のデータ」に関する予測を行う手法が必要

- 機械学習とは？
  - 過去のデータに基づいて、「はじめて見るデータ」に対する予測を行う手法
  - データサイエンスのコア技術として実用化が促進される
  - 近年、ディープラーニングの活用により、画像、音声データの領域で予測精度が圧倒的に向上。「あたかも知性を持っているかのような精度の判断」が可能に。

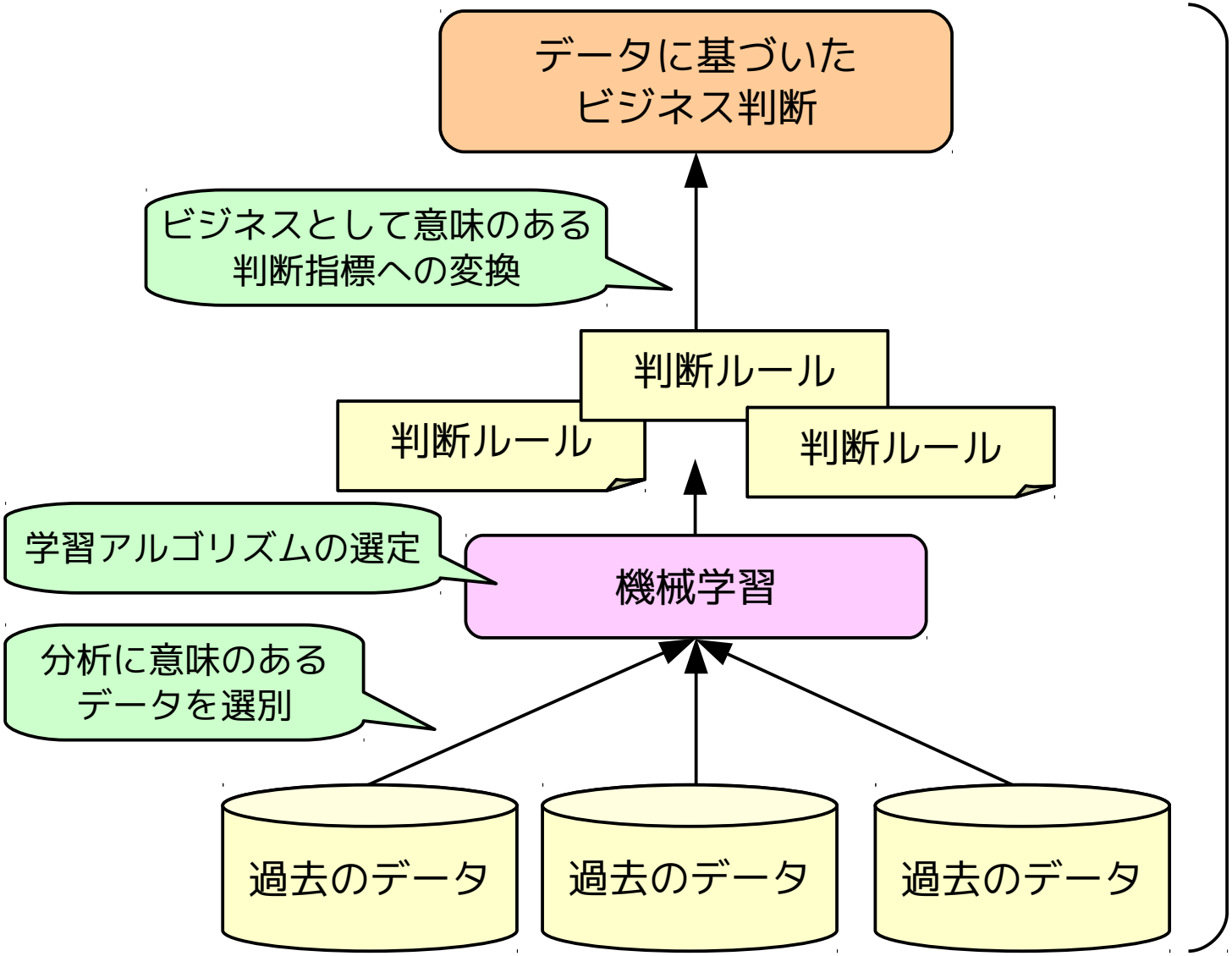
現代的な文脈におけるAIと機械学習の関係

AI：知性を持っているかのような機能を
提供する製品・サービスの総称

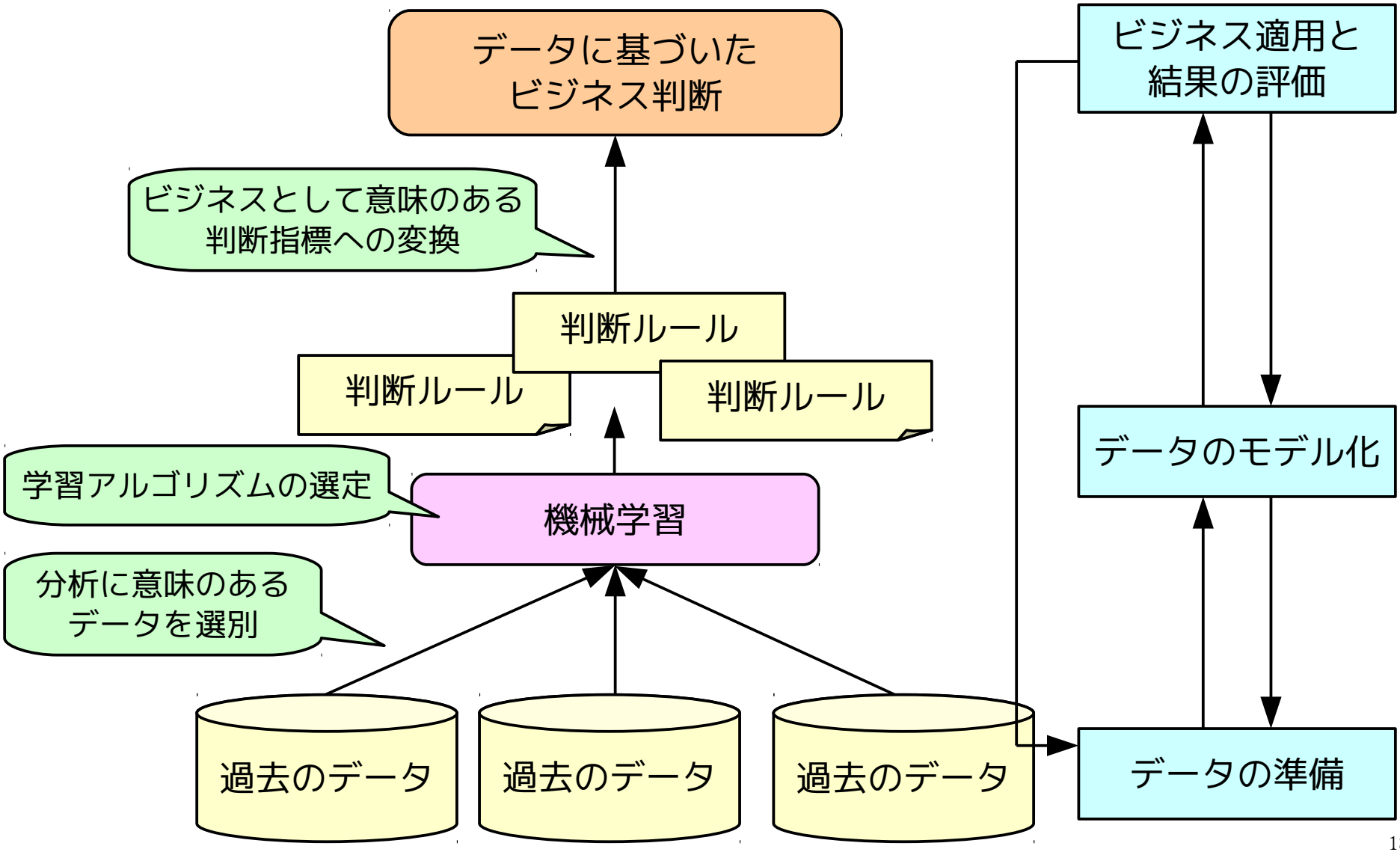
機械学習：「過去のデータ」を基にして
「未知のデータ」に対する予測を行う技術

ディープラーニング：特定のタスクで高い
予測性能を発揮する機械学習の一手法

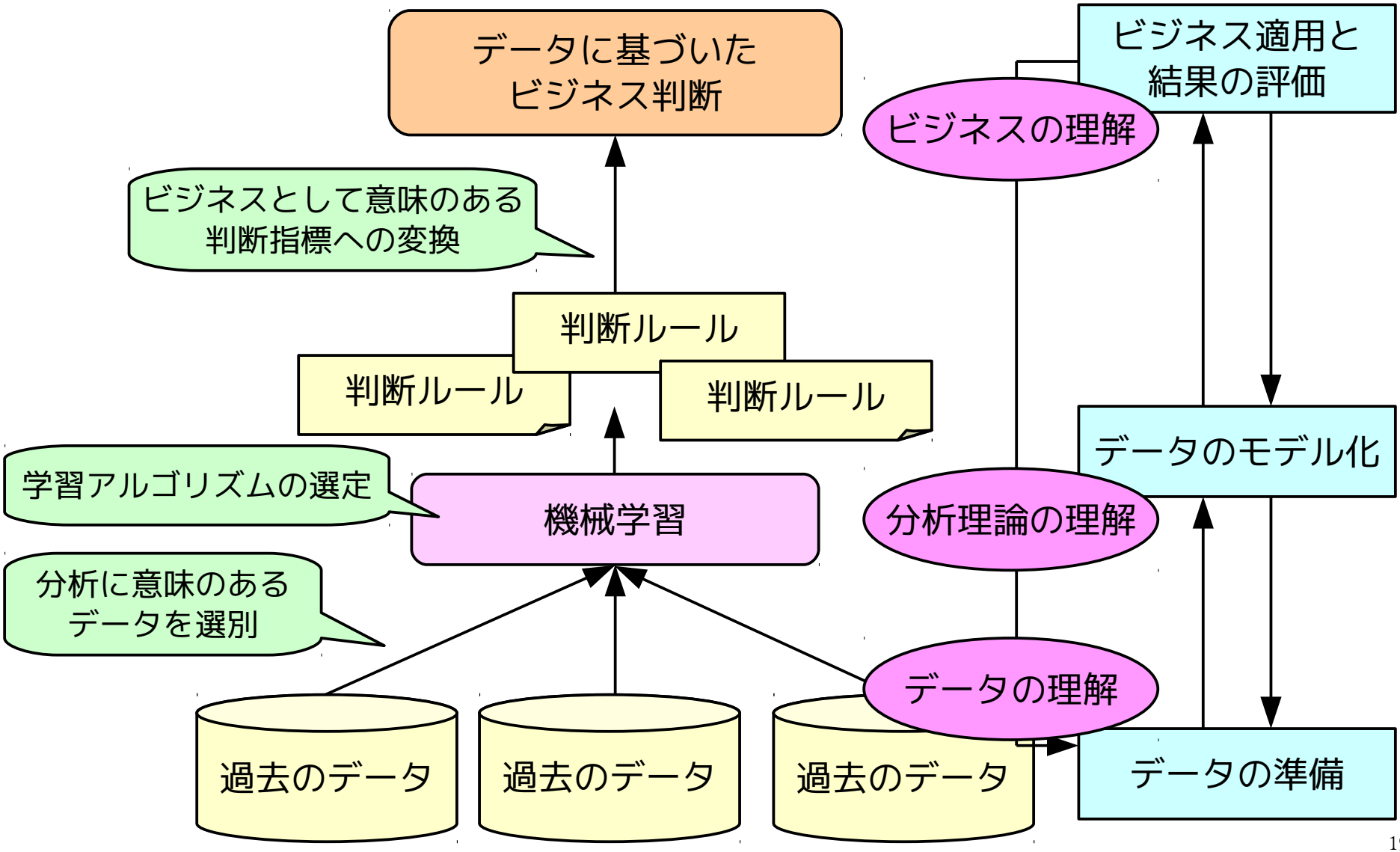
データサイエンスの全体像



データサイエンスの全体像



データサイエンスの全体像



ビジネスにおけるデータサイエンスの役割

- データ分析を通して「ビジネス判断の質を向上」すること。
  - 「未来のビジネス判断」に役立つ指標を「過去のデータ」から導きだすこと。
  - 具体的なビジネス判断を「数値的根拠」を持って提案することが大切。

- 例：台風が近づいているので、過去の台風の際にコンビニでどのような商品が売れたか分析したデータサイエンティストはどんな結果を出すべきか？
  - 水が通常よりも２割多く売れた。
    - ・水の在庫を増やせばいいの？売り切れてなかったら関係ないよね。
  - 「アナ雪」のDVDが爆発的に売れた。
    - ・その時はたまたま「アナ雪」が人気だったからでしょ。
  - コロッケが通常の10倍売れて、小さな店舗では品切れが続出した。
    - ・10倍売れても十分なコロッケを確保・・・・万一売れ残ったら廃棄するのか？
  - ビールが通常の3倍売れて、小さな店舗では品切れが続出した。
    - ・ビールの在庫が少ない店舗は、3倍売れても十分な在庫を確保しよう。でもそれで在庫確保にかけたコスト以上に儲かるの？

データサイエンスにおける機械学習の役割

- 決められたアルゴリズムにしたがって、過去のデータから「判断基準（判断ルール）」を決定する手法。
  - どのようなアルゴリズムを用いるかは、データサイエンティストが判断。
  - 得られたルールの妥当性の評価も大切。過去のデータを正しく判断できるからと言って、将来のデータを正しく判断できるとは限らない。
  - 過去のデータは、「分析のために必要なデータ」として集めているとは限らない。分析に意味のあるデータを選定するのもデータサイエンティストの役割

- 機械学習をビジネスに活かすにはデータサイエンティストとしての「知見」が必要
  - 対象ビジネスの理解：ビジネスに役立つ結果がでないと意味がない。
  - 分析データの理解：データは単なる数字の集まりではない。
  - 分析理論の理解：データとビジネスを結びつける中核となる知識。

メモとしてお使いください

メモとしてお使いください

メモとしてお使いください

機械学習アルゴリズムの分類

機械学習アルゴリズムの分類（代表例）

- Classification / Class probability estimation
  - 既存データを複数のクラスに分類して、新規データがどのクラスに属するかを予想する。特定のクラスを予想する他、各クラスに属する確率を計算する方法もある。
    - ・携帯の機種変更時に「どの機種を選択するか」を予測する。
    - ・特別割引キャンペーンのDMを送ったら「申し込む／申し込まない」を予測する。
    - ・新規メールが「スパムである確率」を計算する。

- Regression（回帰分析）
  - 既存データの背後にある「関数」を推測して、具体的な「数値」を予測する。
    - ・新規サービスを提供したら「何人のユーザー」が利用するか予測する。
    - ・広告宣伝費を２倍にしたら売上は「何倍」になるか予測する。

- Similarity matching
  - 新規データが既存データの「どれと似ているか」を予測する。
    - ・新規顧客は、既存のどの顧客と似ているかを予測する。

機械学習アルゴリズムの分類（代表例）

- Clustering
  - 既存データについて、具体的な指標を与えず、自然に分類されるグループを発見する。一般には、自然なグループを発見した後に、それぞれのグループの特性を他のアルゴリズムで分析する。
    - ・既存顧客をグループに分けて、それぞれのグループに提供する製品／サービスを決定する。
- Co-occurrence grouping
  - 既存データから、同時に発生する事象を発見する。
    - ・「Xを買った人はYも買っています」
- Reinforcement learning（強化学習）
  - エージェントが環境との相互作用の中でデータを収集して学習を行う。
    - ・囲碁プログラムがコンピューター同士の対局から有効な手筋を発見する。
    - ・自動運転プログラムがランダムな運転を通じて、正しく進む方法を発見する。

データ分析に利用できるオープンソースのツール

- Jupyter Notebook
  - ブラウザー上で、Pythonのコードを対話的に実行するツール

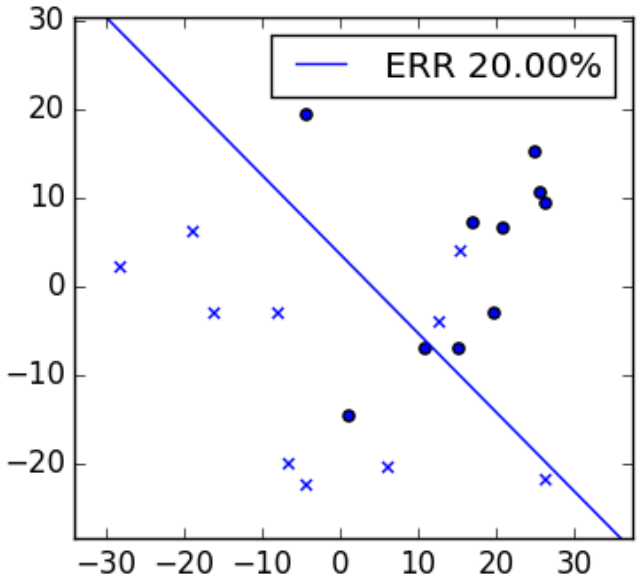
- Pythonのデータ解析ライブラリ
  - NumPy：ベクトルや行列を扱うライブラリ
  - SciPy：科学計算用ライブラリ
  - matplotlib：グラフ作成ライブラリ
  - pandas：Rに類似のデータフレームを提供
  - scikit-learn：機械学習用ライブラリ
  - TensorFlow：ニューラルネットワークの機械学習ライブラリ

ロジスティック回帰と最尤推定法

線形判別法の例題

- 学習に使用するデータ（トレーニングセット）
  - (x, y) 平面上の N 個のデータポイント $\{(x_n, y_n, t_n)\}_{n=1}^{N}$
  - データポイントは、2タイプに分かれており、$t_n = 1, 0$ にラベル付けされている。
- 解くべき問題
  - ２タイプのデータを分割する直線を求める。
  - きれいに分割できない場合は、何らかの意味で「最善」の分割を与える。

トレーニングセットと
判別直線の例



●：$t = 1$
×：$t = 0$

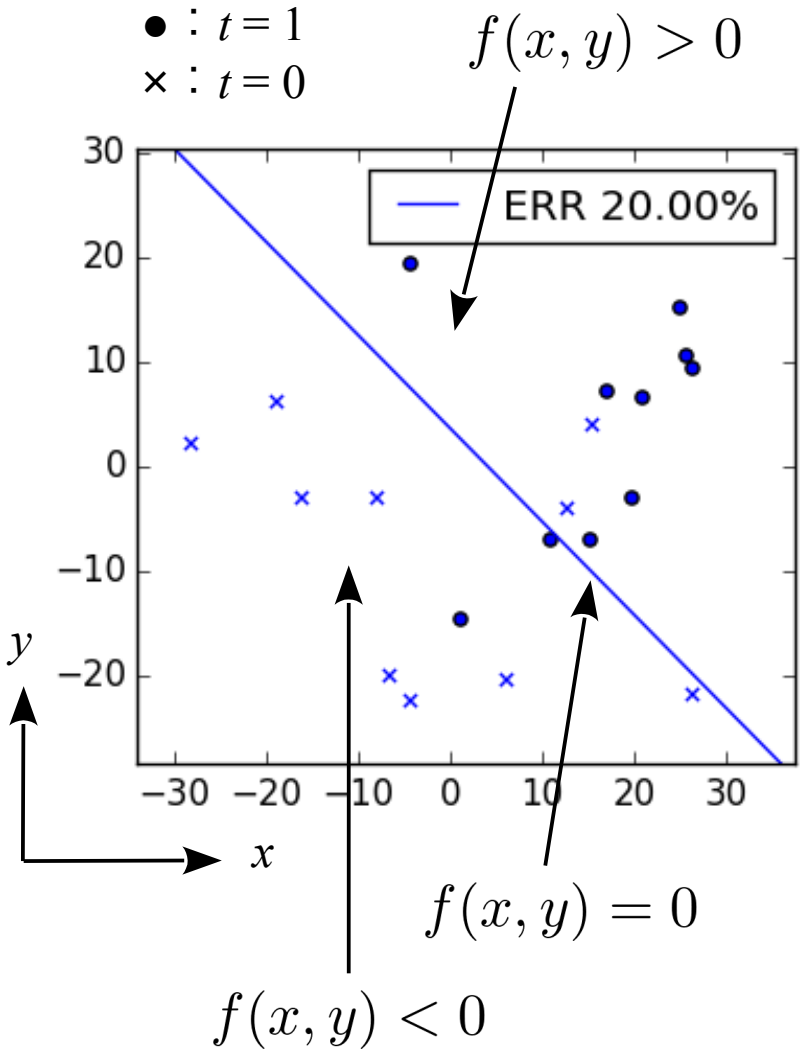
ロジスティック回帰の考え方

- ロジスティック回帰では、分割線を次式で与えます。

$$f(x, y) = w_0 + w_1 x + w_2 y = 0$$

- パラメータ **w** を決定するために「最尤推定」を用います。

- つまり、あるデータが得られる「確率」を決めて、トレーニングセットが得られる確率を最大にします。



ロジスティック関数による確率

- 点 (x, y) で新たに取得したデータが「$t = 1$」である確率を $P(x, y)$ とします。

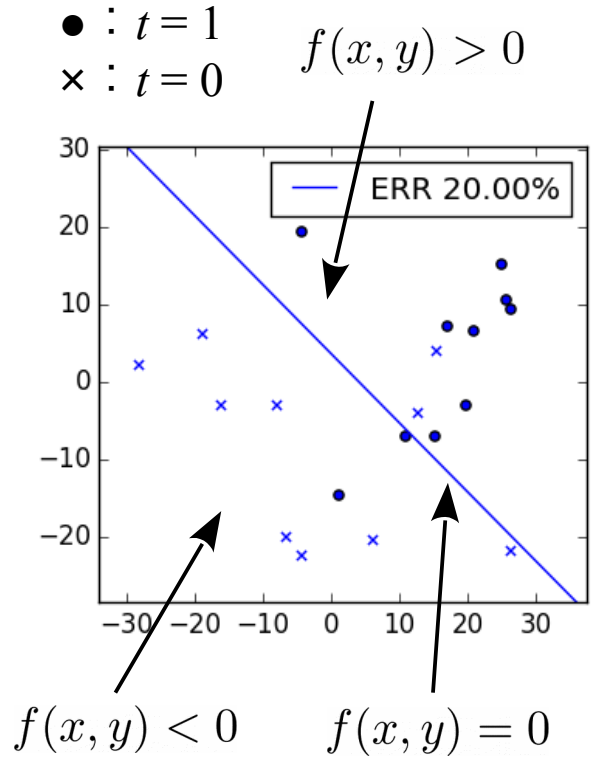
- 右図において分割線から右上に進むと P は大きくなり、左下に進むと P は小さくなると考えられます。

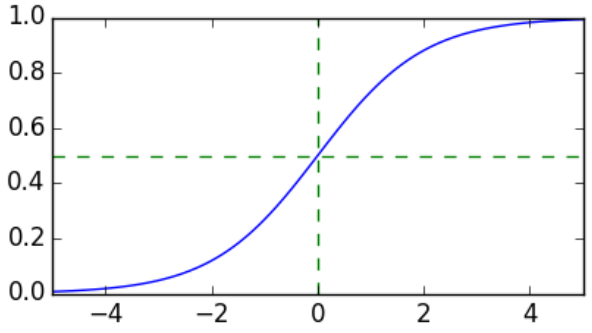
- そこでこの確率を次式で定義します。

$$P(x, y) = \sigma(w_0 + w_1 x + w_2 y)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$ ：ロジスティック関数（シグモイド関数）

  - $\sigma(a)$ は、右図のように 0 から 1 になめらかに増加する関数です。
  - $f(x, y) = 0$ の分割線上では、確率はちょうど 0.5 になります。





ロジスティック回帰における尤度関数

- 前ページで定義した確率 $P(x, y)$ を元にして、トレーニングセット $\{(x_n, y_n, t_n)\}_{n=1}^{N}$ が得られる確率を計算してみます。
  - 点 (x_n, y_n) から $t_n = 1$ のデータが得られた場合、それが起きる確率は：
    $P_n = P(x_n, y_n)$
  - 点 (x_n, y_n) から $t_n = 0$ のデータが得られた場合、それが起きる確率は：
    $P_n = 1 - P(x_n, y_n)$
  - これら2つは、（技巧的ですが）次のように１つの式にまとめられます。
    $P_n = P(x_n, y_n)^{t_n}\{1 - P(x_n, y_n)\}^{1-t_n}$
- 従って、トレーニングセットが得られる確率（尤度関数）は次式になります。

$$\mathrm{P}=\prod_{n=1}^{N} P_n = \prod_{n=1}^{N} P(x_n, y_n)^{t_n}\{1 - P(x_n, y_n)\}^{1-t_n}$$
$$=\prod_{n=1}^{N} z_n^{t_n}(1 - z_n)^{1-t_n} \quad (z_n = P(x_n, y_n))$$

勾配降下法によるパラメーターの最適化

- 次式で誤算関数 E を定義すると、「確率 P 最大 ⇔ 誤差関数 E 最小」が成り立ちます。

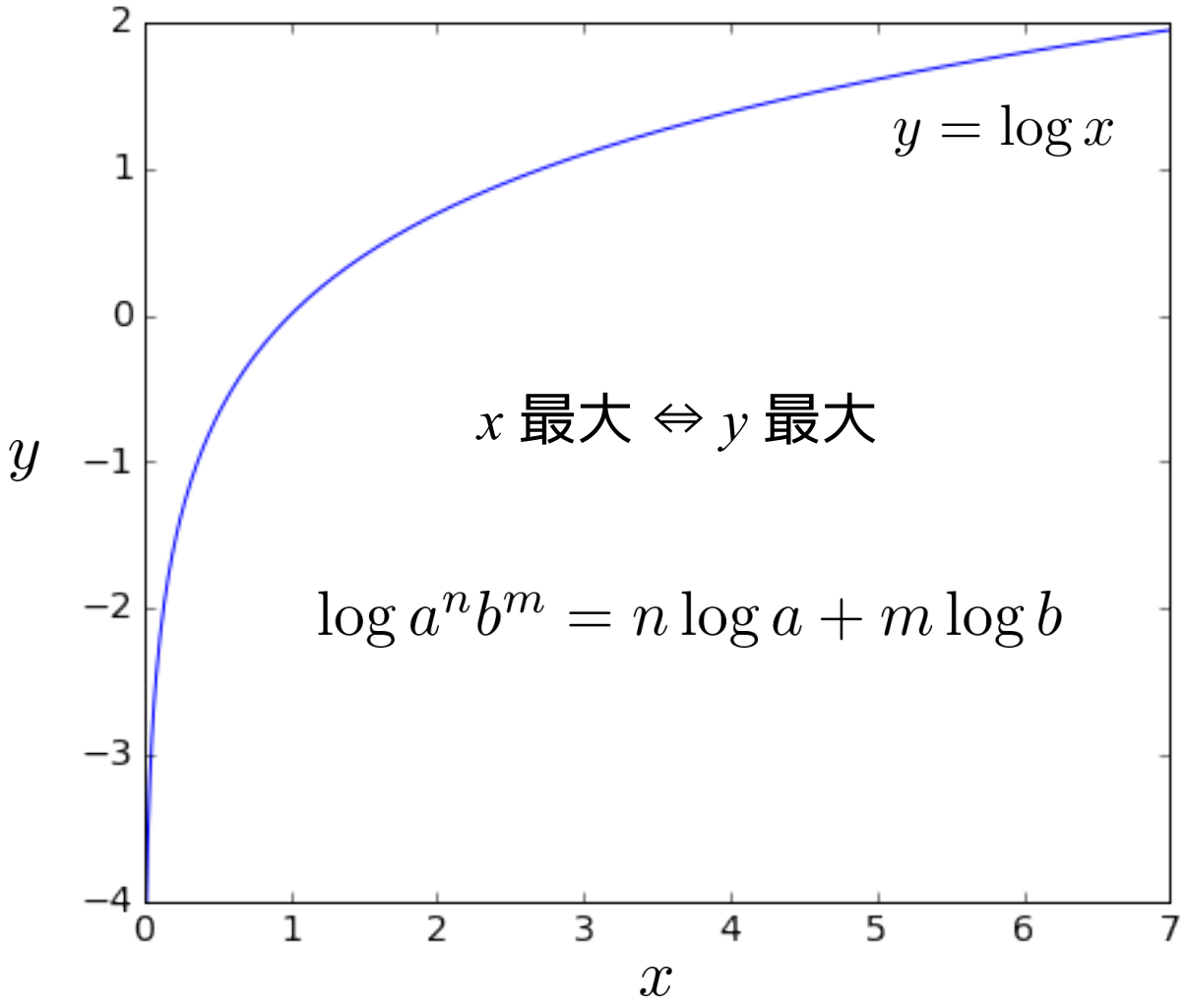
$$E = -\log P = -\sum_{n=1}^{N} \{t_n \log z_n + (1 - t_n) \log(1 - z_n)\}$$

- 一般に誤差関数 E に対して、「勾配ベクトル ∇E の反対方向」にパラメーターを調整することで誤差関数の値を小さくすることができます。

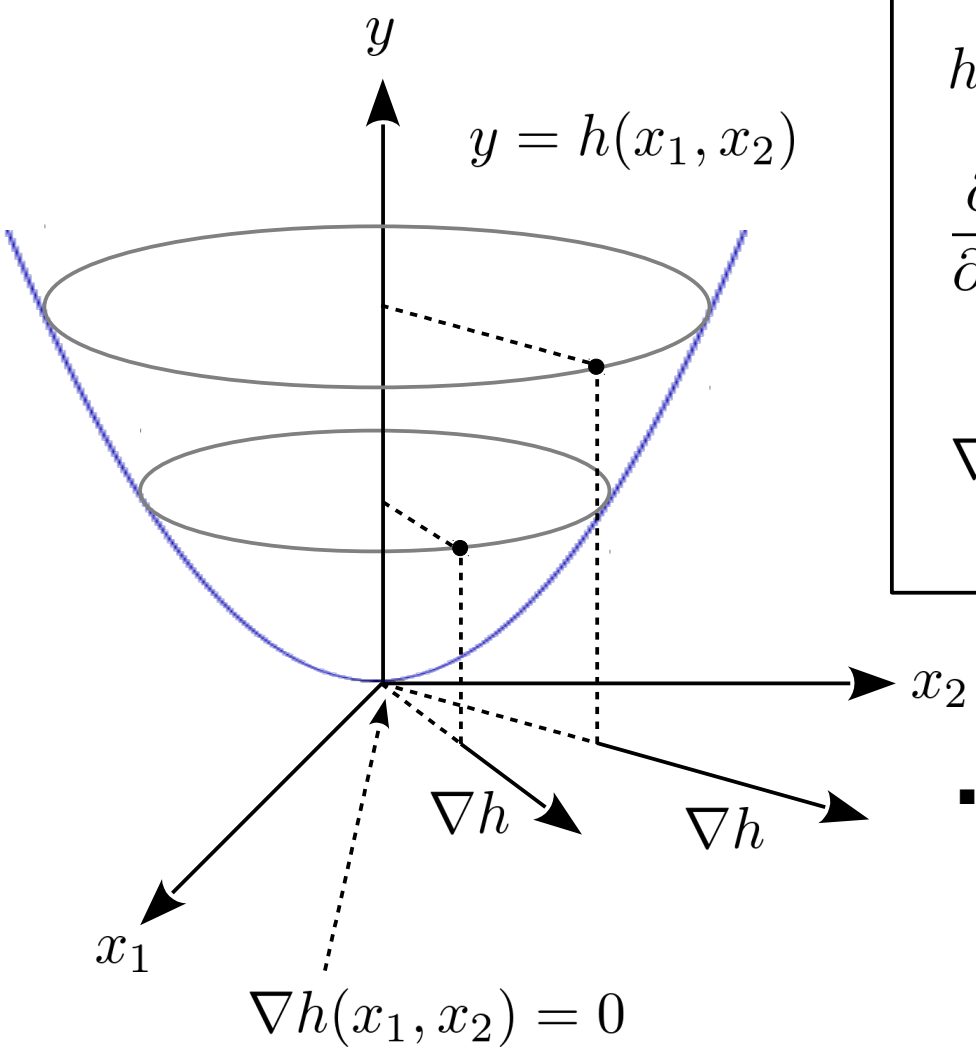
$$\mathbf{w} \to \mathbf{w} - \epsilon \nabla E \qquad \mathrm{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}, \quad \nabla E = \begin{pmatrix} \frac{\partial E}{\partial w_0} \\ \\ \frac{\partial E}{\partial w_1} \\ \\ \frac{\partial E}{\partial w_2} \end{pmatrix}$$

  - 勾配ベクトルについては、次ページを参照。
  - この性質を利用して、誤差関数を最小にするパラメーターを発見する方法が「勾配降下法」です。

（参考）対数関数の性質


$y = \log x$
x 最大 $\Leftrightarrow$ y 最大
$\log a^n b^m = n \log a + m \log b$
y
x
2
1
0
−1
−2
−3
−4
0
1
2
3
4
5
6
7

勾配ベクトルとグラフの傾きの関係



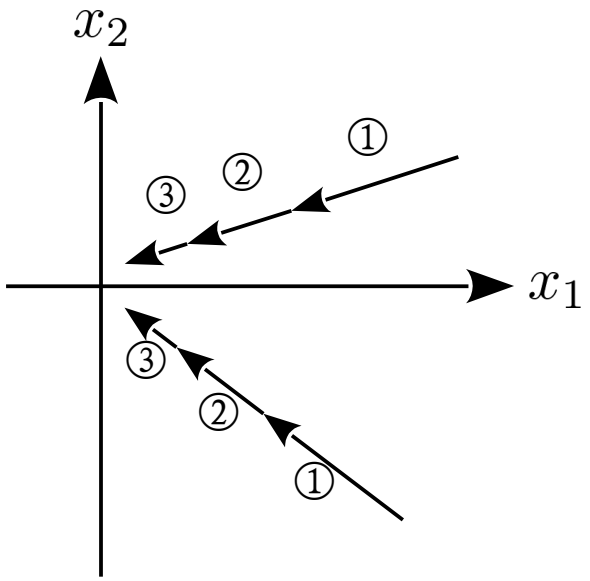
$$h(x_1, x_2) = \frac{1}{2}(x_1^2 + x_2^2)$$

$$\frac{\partial h}{\partial x_1} = x_1, \quad \frac{\partial h}{\partial x_2} = x_2$$

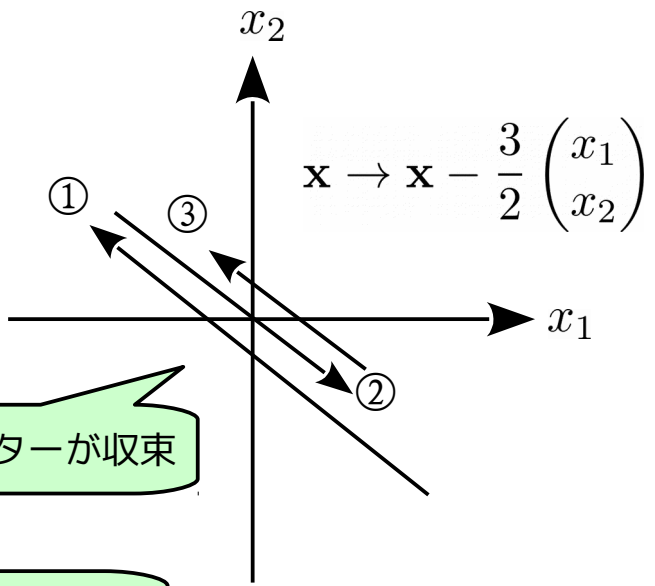
$$\nabla h(x_1, x_2) = \begin{pmatrix} \frac{\partial h}{\partial x_1} \\ \\ \frac{\partial h}{\partial x_2} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

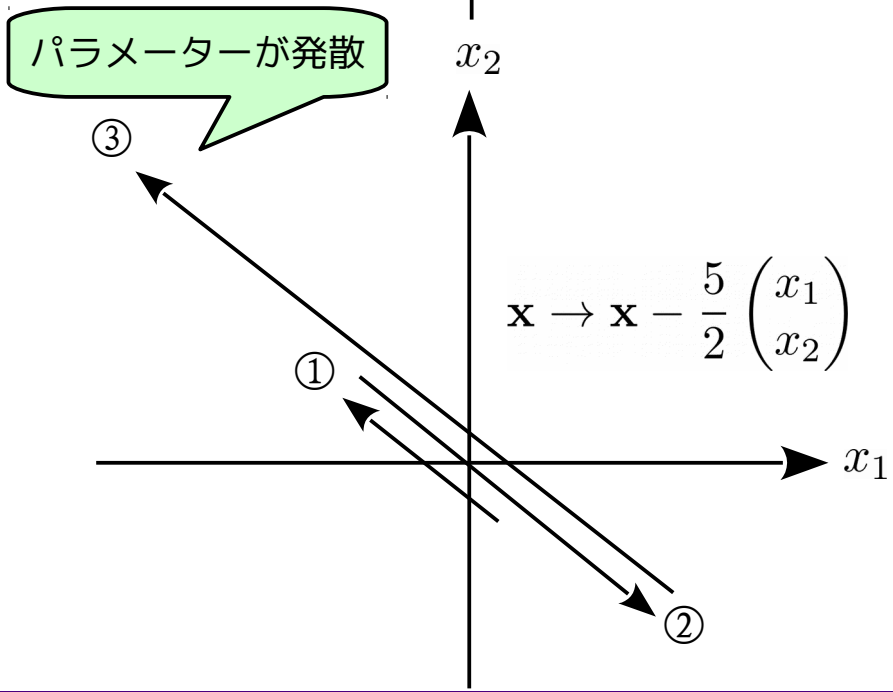
- 多変数関数 h の勾配ベクトル ∇h は、グラフの傾きが最大になる方向に対応しており、その大きさはグラフの傾きに一致します。

勾配降下法と学習率



$$\mathbf{x} \rightarrow \mathbf{x} - \epsilon \nabla h$$





- 勾配降下法では、勾配ベクトルの反対方向にパラメーターを繰り返し修正することで、誤差関数を最小にします。
- この際、学習率 ε の値により、パラメーターの変化の様子が変わります。

メモとしてお使いください

メモとしてお使いください

メモとしてお使いください

TensorFlowのコードの書き方

注意

- このセクションで扱うコードの全体像は、次のノートブックを参照してください。
  - 「Chapter02/Maximum likelihood estimation example.ipynb」

TensorFlowのコードの記述方法

- TensorFlowで扱うデータは、すべて「多次元リスト」で表現します。
  - 多くの場合、2次元リスト（すなわち「行列」）を使用します。そのため、与えられた関係式（数式）を行列形式で表現すると、自然にTensorFlowのコードに変換することができます。

- 例として、下記のモデル（数式）をTensorFlowのコードで記述してみます。

$$f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2$$

$$P(x_1, x_2) = \sigma(f(x_1, x_2))$$

  - この時、次の3種類の違いを意識して、コードを書くことに注意してください。

・トレーニングデータを代入する変数：Placeholder
・チューニング対象のパラメーター：Variable
・これらを組み合わせた計算式

TensorFlowのコードの記述方法

- 直線を表す１次関数を行列で表現すると、次のようになります。

$$f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 = (x_1, x_2) \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + w_0$$

- ただし、(x_1, x_2) は、トレーニングデータを代入する「Placeholder」なので、一般には多数のデータが入ります。
  - TensorFlowで学習処理を行う際は、複数のデータをまとめて代入して計算します。
  - そこで、n 番目のデータを (x_{1n}, x_{2n}) として、$n = 1, \cdots, N$ の全データを並べた行列を用意すると、次の関係式が成り立ちます。

$$\mathbf{f} = \mathbf{X}\mathbf{w} + w_0 \qquad \mathbf{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_N \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \vdots \\ x_{1N} & x_{2N} \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

  - ここで、$f_n = f(x_{1n}, x_{2n})$（n 番目のデータに対する f の値）とします。
  - 最後の $+w_0$ は、ブロードキャストルール（各成分に w_0 を足す）を適用しています。

TensorFlowのプログラミングモデル

- 最後に $\mathbf{f}$ の各成分にシグモイド関数 σ を適用すると、各データの確率 P_n が計算されます。

$$\mathbf{P} = \sigma(\mathbf{f}) \qquad \mathbf{P} = \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix}$$

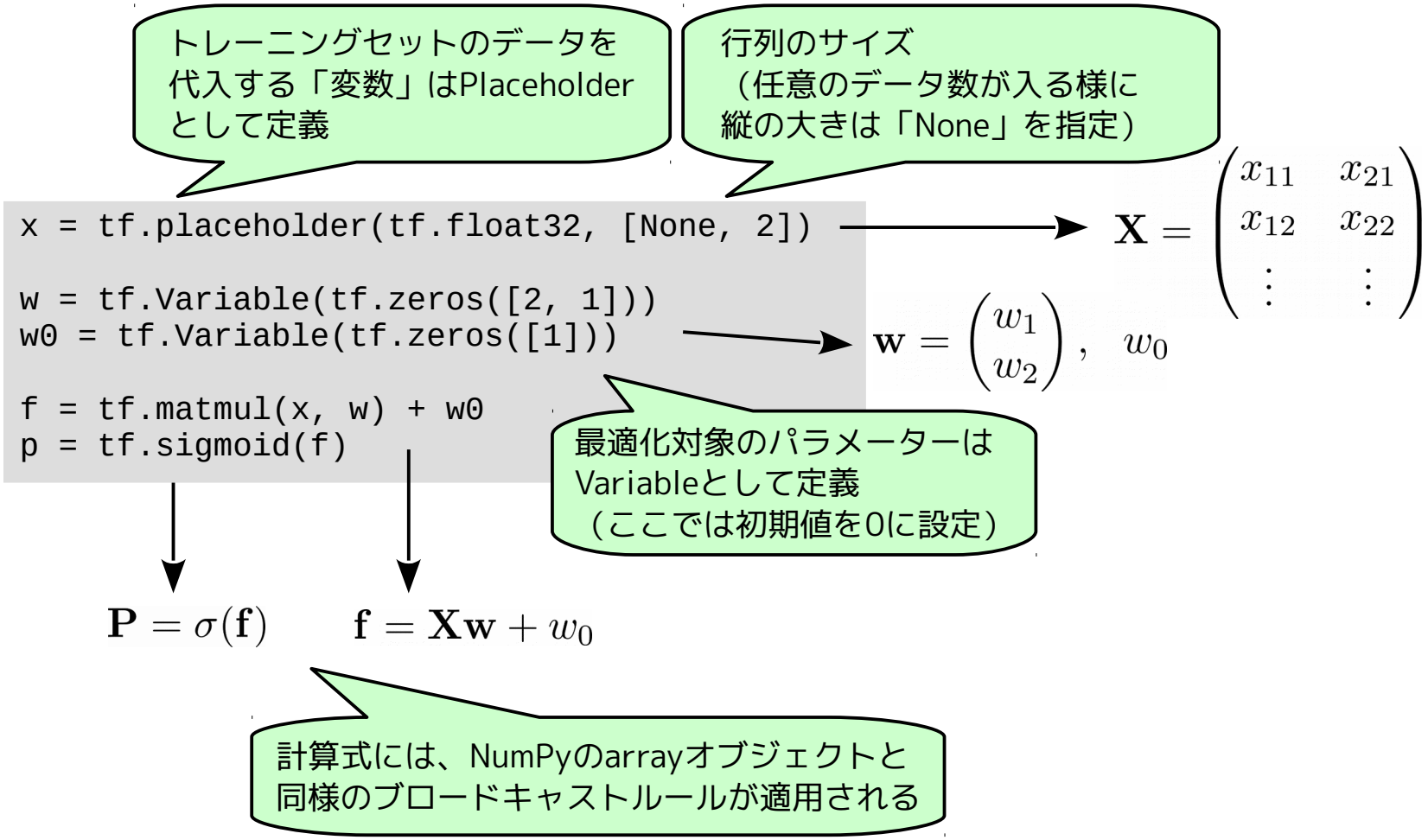
  - ここで、$P_n = P(x_{1n}, x_{2n})$ とします。
  - $\sigma(\mathbf{f})$ は、ブロードキャストルール（各成分に σ を演算する）を適用しています。

- 以上の関係をTensorFlowのコードで表現すると次になります。

```
x = tf.placeholder(tf.float32, [None, 2])

w = tf.Variable(tf.zeros([2, 1]))
w0 = tf.Variable(tf.zeros([1]))

f = tf.matmul(x, w) + w0
p = tf.sigmoid(f)
```

TensorFlowのプログラミングモデル

- TensorFlowのコードと数式の関係は、次の様になります。



誤差関数とトレーニングアルゴリズムの指定

- トレーニングセットを用いた学習処理を実施するには、誤差関数とトレーニングアルゴリズムの指定が必要です。

```
t = tf.placeholder(tf.float32, [None, 1])

loss = -tf.reduce_sum(t*tf.log(p) + (1-t)*tf.log(1-p))

train_step = tf.train.AdamOptimizer().minimize(loss)
```

$$\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \end{pmatrix}$$

tf.reduce_sumは行列の全成分を足し算する関数

AdamOptimizerで誤差関数lossを最小化

$$E = -\sum_{n=1}^{N} \left[t_n \log P(x_{1n}, x_{2n}) + (1 - t_n) \log \left\{ 1 - P(x_{1n}, x_{2n}) \right\} \right]$$

（参考）ブロードキャストルールのまとめ

- ここまでのコードで用いられたブロードキャストルールは次の通りです。

(1) 行列とスカラーの足し算は、各成分に対する足し算になる

$$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}+(10)=\begin{pmatrix}11&12&13\\14&15&16\\17&18&19\end{pmatrix}$$

(2) 同じサイズの行列の「*」演算は、成分ごとの掛け算になる

$$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}*\begin{pmatrix}10&100&1000\\10&100&1000\\10&100&1000\end{pmatrix}=\begin{pmatrix}10&200&3000\\40&500&6000\\70&800&9000\end{pmatrix}$$

(3) スカラーを受け取る関数を行列に適用すると、各成分に関数が適用される

$$\sigma\begin{pmatrix}1\\2\\3\end{pmatrix}=\begin{pmatrix}\sigma(1)\\\sigma(2)\\\sigma(3)\end{pmatrix}$$

`t*tf.log(p)` + `(1-t)*tf.log(1-p)`

$$\begin{pmatrix}t_1\log P_1\\t_2\log P_2\\t_3\log P_3\\\vdots\end{pmatrix}+\begin{pmatrix}(1-t_1)\log(1-P_1)\\(1-t_2)\log(1-P_2)\\(1-t_3)\log(1-P_3)\\\vdots\end{pmatrix}$$

`f = tf.matmul(x, w)` + `w0`

ブロードキャストルール

$$\begin{pmatrix}f_1\\f_2\\f_3\\\vdots\end{pmatrix}=\begin{pmatrix}x_{11}&x_{21}\\x_{12}&x_{22}\\x_{13}&x_{23}\\\vdots&\vdots\end{pmatrix}\begin{pmatrix}w_1\\w_2\end{pmatrix}+\begin{pmatrix}w_0\\w_0\\w_0\\\vdots\end{pmatrix}$$

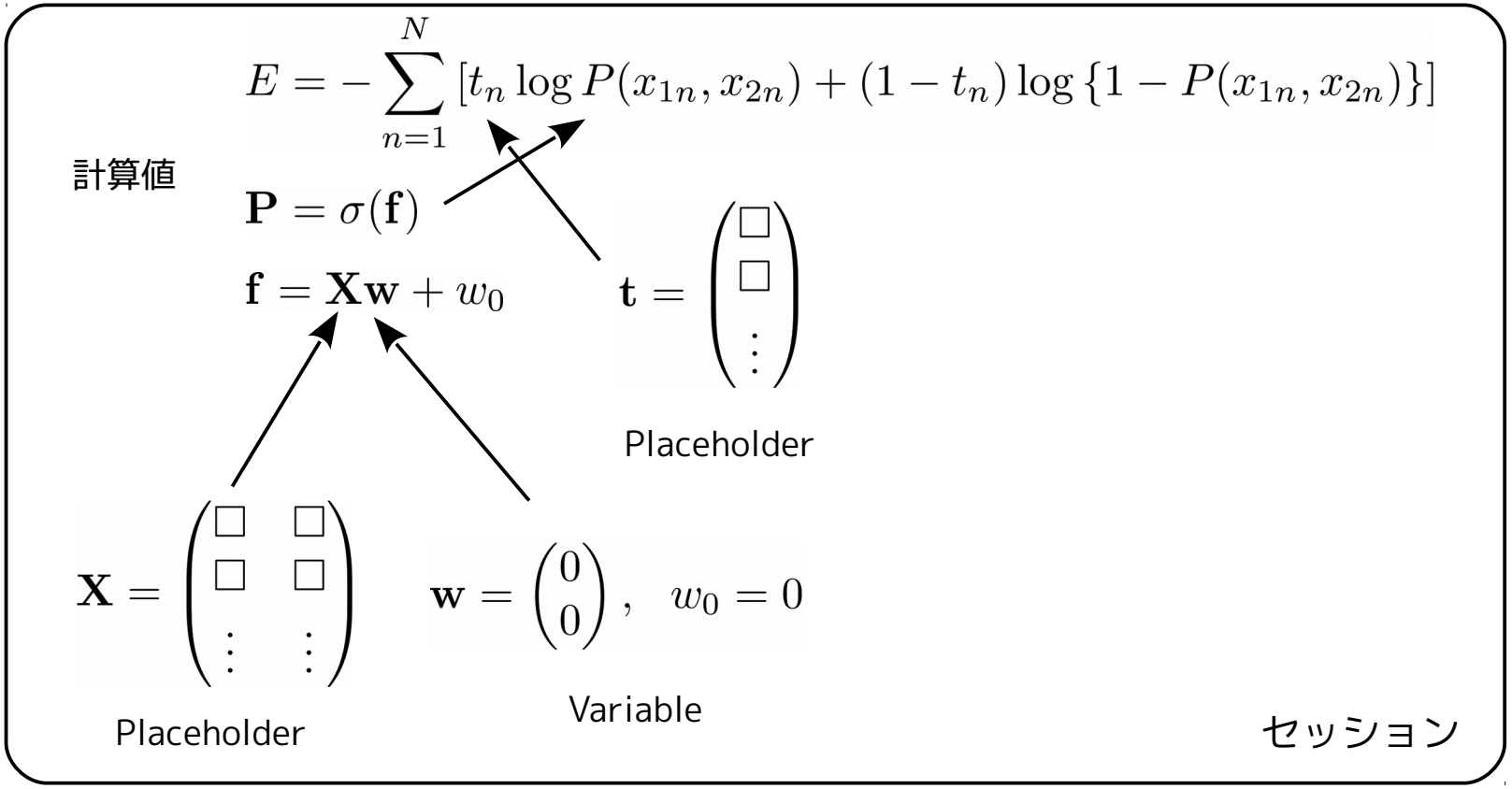
`p = tf.sigmoid(f)`

ブロードキャストルール

$$\begin{pmatrix}P_1\\P_2\\P_3\\\vdots\end{pmatrix}=\begin{pmatrix}\sigma(f_1)\\\sigma(f_2)\\\sigma(f_3)\\\vdots\end{pmatrix}$$

セッションを用いた計算処理

- これまでのコードは、各種の関係式を定義しただけで実際の計算は何も行われていません。TensorFlowでは、「セッション」を用意して、その中で計算処理を行います。
  - 複数のセッションを用意すると、それぞれで異なる値を保持することもできます。



セッションを用いた計算処理

- はじめにセッションを用意して、セッション内のVariableを初期化します。

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())
```

- セッション内でトレーニングアルゴリズムを評価すると、勾配降下法によるパラメーターの最適化が実施されます。
  - プレースホルダーに代入する値は、feed_dictオプションで指定します。
  - 任意の計算値を評価すると、その時点でのパラメーターの値を用いて計算した結果が得られます。

```
i = 0
for _ in range(20000):
    i += 1
    sess.run(train_step, feed_dict={x:train_x, t:train_t})
    if i % 2000 == 0:
        loss_val, acc_val = sess.run(
            [loss, accuracy], feed_dict={x:train_x, t:train_t})
        print ('Step: %d, Loss: %f, Accuracy: %f'
               % (i, loss_val, acc_val))
```

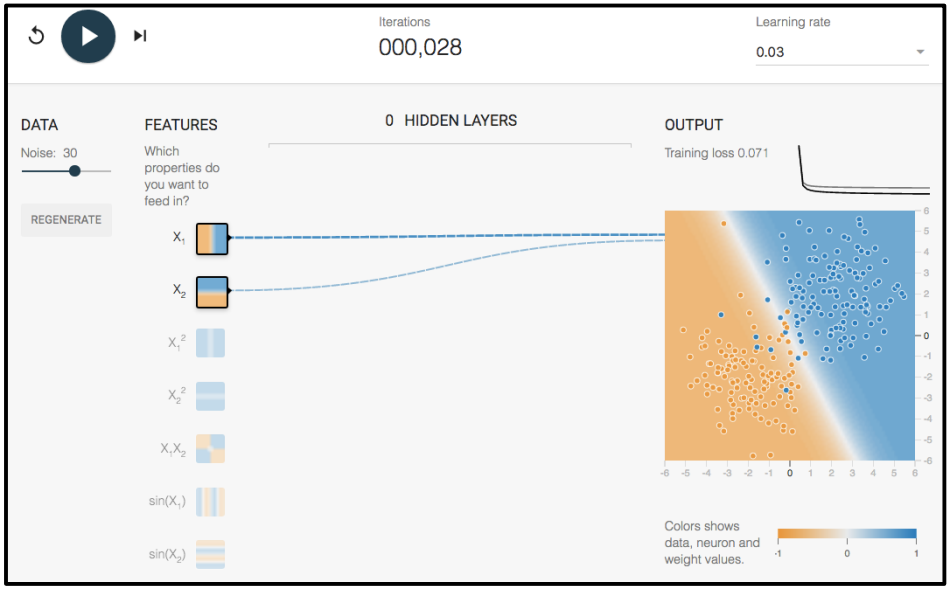
勾配降下法による最適化を実施

その時点でのパラメーターの値による計算結果を取得

練習問題

- Neural Network Playgroundを用いて、勾配降下法による学習の様子を観察してください。
  - 特にノイズ（Noise）の大きなデータに対して、学習率（Learning Rate）を大きく設定するとどうなるでしょうか？

- Neural Network Playground - https://goo.gl/nYYRjR



メモとしてお使いください

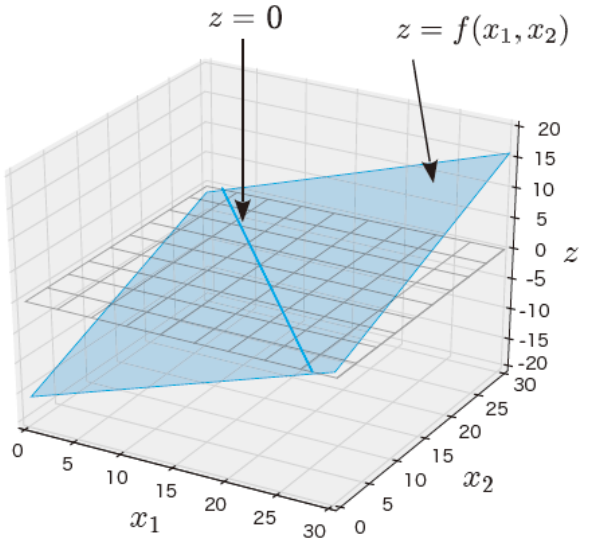
線形多項分類器

注意

- このセクションで扱うコードの全体像は、次のノートブックを参照してください。
  - 「Chapter02/MNIST softmax estimation.ipynb」
- また、次のノートブックを参照して、MNISTデータセットの利用方法を確認しておいてください。
  - 「Chapter02/MNIST dataset sample.ipynb」

線形多項分類器の仕組み

- 関数 $z = f(x_1, x_2)$ のグラフを描くと、図のように「斜めに配置した板」で (x_1, x_2) 平面が分割されることがわかります。

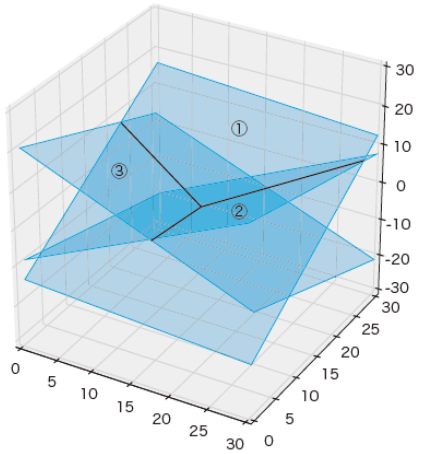


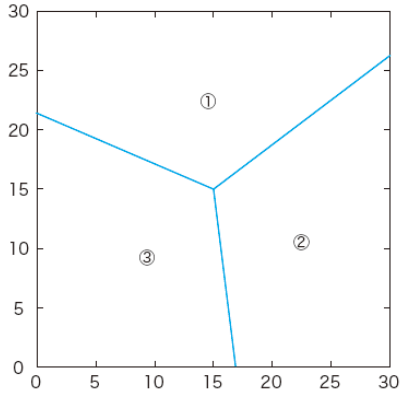
- 同様に、3つの１次関数を用意して、「どの関数が一番大きいか」で平面を3分割することができます。

$$f_1(x_1, x_2) = w_{01} + w_{11}x_1 + w_{21}x_2$$

$$f_2(x_1, x_2) = w_{02} + w_{12}x_1 + w_{22}x_2$$

$$f_3(x_1, x_2) = w_{03} + w_{13}x_1 + w_{23}x_2$$





ソフトマックス関数による確率への変換

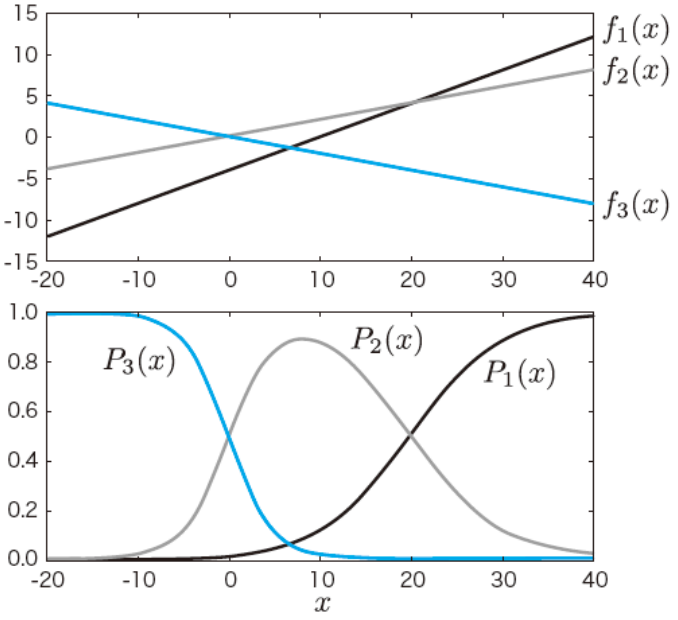
- 点 (x_1, x_2) が i 番目の領域である確率を次式で定義します。

$$P_i(x_1, x_2) = \frac{e^{f_i(x_1,x_2)}}{e^{f_1(x_1,x_2)} + e^{f_2(x_1,x_2)} + e^{f_3(x_1,x_2)}}$$

- これは、f_1, f_2, f_3 の大小関係を確率に変換したもので、次の条件を満たすことがすぐにわかります。

$$0 \leq P_i \leq 1 \quad (i = 1, 2, 3)$$

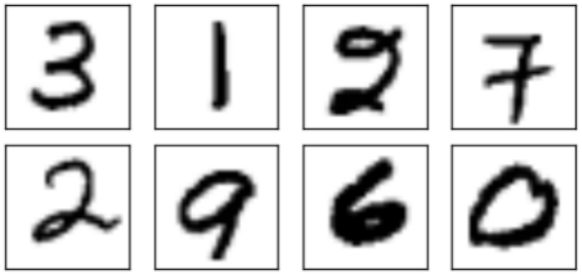
$$P_1 + P_2 + P_3 = 1$$

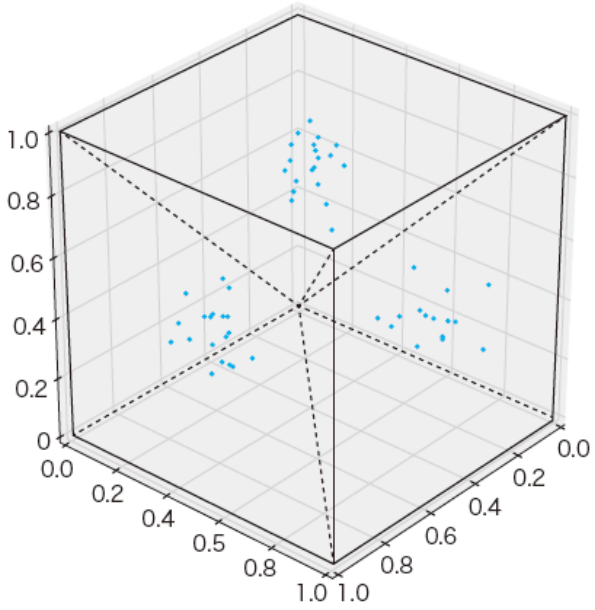


ソフトマックス関数による画像分類

- たとえば、28x28ピクセルのグレイスケール画像は、各ピクセルの値を一列にならべると、784次元空間の点とみなすことができます。

- 大量の画像データを784次元空間にばらまくと、類似画像は互いに近くに集まると考えられないでしょうか？
  - ソフトマックス関数で784次元空間を分割することで、画像を分類できるかも知れません。





線形多項分類器の行列表現

- 一般に M次元空間をK種類に分類するとして、K個の一次関数を用意します。

$$f_k(x1, \cdots, x_M) = w_{1k}x_1 + w_{2k}x_2 + \cdots + w_{Mk}x_M + w_{0k}, \quad (k = 1, \cdots, K)$$

- n 番目の画像データを $\mathbf{x}_n = (x_{1n}, \cdots, x_{Mn})$ として、行列を用いて書き直すと次の様になります。（最後の足し算はブロードキャストルールを適用）

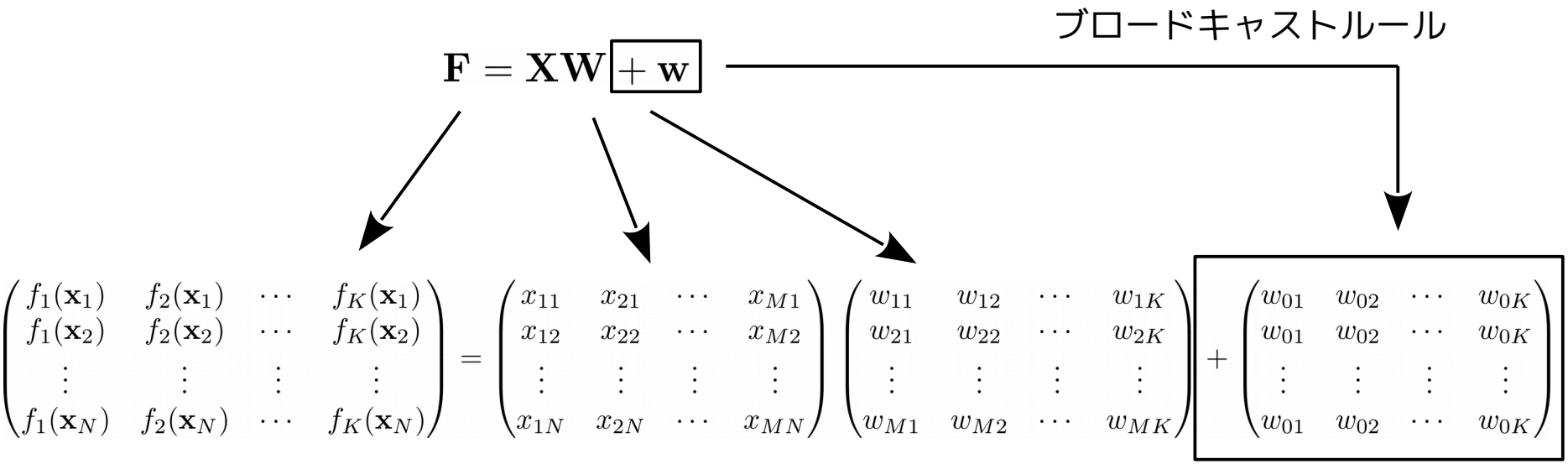
$$\mathbf{F} = \mathbf{X}\mathbf{W} + \mathbf{w}$$

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{M1} \\ x_{12} & x_{22} & \cdots & x_{M2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1N} & x_{2N} & \cdots & x_{MN} \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1K} \\ w_{21} & w_{22} & \cdots & w_{2K} \\ \vdots & \vdots & \vdots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MK} \end{pmatrix}$$

$$\mathbf{F} = \begin{pmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_K(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_K(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_K(\mathbf{x}_N) \end{pmatrix} \quad \mathbf{w}_n = (w_{01}, \cdots, w_{0K})$$

線形多項分類器の行列表現

- 行列計算を整理すると次のようになります。



線形多項分類器の行列表現

- n 番目の画像データ $\mathbf{x}_n$ が k 番目の領域に属する確率は、ソフトマックス関数を用いて、次式で計算されます。

$$P_k(\mathbf{x}_n) = \frac{e^{f_k(\mathbf{x}_n)}}{\sum_{k'=1}^{K} e^{f_{k'}^{(\mathbf{x}_n)}}}$$

- TensorFlowの組み込み関数 tf.nn.softmax を用いると、この確率を行列 $\mathbf{F}$ から直接に計算してくれます。

$$\mathbf{P} = \mathrm{tf.nn.softmax}(\mathbf{F})$$

$$\mathbf{P} = \begin{pmatrix} P_1(\mathbf{x}_1) & P_2(\mathbf{x}_1) & \cdots & P_K(\mathbf{x}_1) \\ P_1(\mathbf{x}_2) & P_2(\mathbf{x}_2) & \cdots & P_K(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ P_1(\mathbf{x}_N) & P_2(\mathbf{x}_N) & \cdots & P_K(\mathbf{x}_N) \end{pmatrix}$$

TensorFlowのコードによる表現

- $M = 784,\ K = 10$ に注意してTensorFlowのコードで書き直すと、次の様になります。
  - PlaceholderとVariableの違いに注意してください。

```
x = tf.placeholder(tf.float32, [None, 784])
w = tf.Variable(tf.zeros([784, 10]))
w0 = tf.Variable(tf.zeros([10]))
f = tf.matmul(x, w) + w0
p = tf.nn.softmax(f)
```

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{M1} \\ x_{12} & x_{22} & \cdots & x_{M2} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1K} \\ w_{21} & w_{22} & \cdots & w_{2K} \\ \vdots & \vdots & \vdots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MK} \end{pmatrix}$$

$$\mathbf{w}_n = (w_{01}, \cdots, w_{0K})$$

$$\mathbf{F} = \mathbf{X}\mathbf{W} + \mathbf{w}$$

$$\mathbf{P} = \mathrm{tf.nn.softmax}(\mathbf{F})$$

誤差関数の計算

- n 番目のデータ $\mathbf{x}_n$ の正解ラベルは、次の 1-of-K ベクトルで与えられます。

$$\mathbf{t}_n = (0, \cdots, 1, \cdots, 0)$$

  - 正解を k として、k 番目の要素 t_{kn} のみが 1 になっています。

- 今のモデルでこのデータが得られる確率は $P_k(\mathbf{x}_n)$ なので、全データが得られる確率は次式になります。

$$P = \prod_{n=1}^{N} \prod_{k'=1}^{K} P_{k'}(\mathbf{x}_n)^{t_{k'n}}$$

正解 k の所のみ1になる

$$\rightarrow \mathrm{P}_1(\mathbf{x}_n)^0 P_2(\mathbf{x}_n)^0 \cdots P_k(\mathbf{x}_n)^1 \cdots P_K(\mathbf{x}_n)^0 = P_k(\mathbf{x}_n)$$

- 次式で誤算関数 E を定義すると、「確率 P 最大 ⇔ 誤差関数 E 最小」が成り立ちます。

$$E = -\log P = -\sum_{n=1}^{N} \sum_{k'=1}^{K} t_{k'n} \log P_{k'}(\mathbf{x}_n)$$

TensorFlowのコードによる表現

- 誤差関数と最適化アルゴリズムをコードに直すと次になります。

```
t = tf.placeholder(tf.float32, [None, 10])

loss = -tf.reduce_sum(t * tf.log(p))

train_step = tf.train.AdamOptimizer().minimize(loss)
```

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{21} & \cdots & t_{M1} \\ t_{12} & t_{22} & \cdots & t_{M2} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$E = -\mathrm{tf.reduce\_sum}(\mathbf{T} * \log \mathbf{P})$$

- 正解率は次で計算できます。

正解/不正解のブール値を並べた行列

```
correct_prediction = tf.equal(tf.argmax(p, 1), tf.argmax(t, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

ブール値を1/0に変換して平均値を計算

  - tf.argmax(X, 1) は、行列 X の各列について、最大要素のインデックスを返します。確率が最大のインデックスと正解ラベルが最大のインデックスが一致すれば、正解した事になります。

確率最大の文字による判定

- モデルから計算される確率が最大となる文字で予測した場合、次の比較で正解／不正解が判定できます。

$$\mathbf{P} = \begin{pmatrix} P_1(\mathbf{x}_1) & P_2(\mathbf{x}_1) & \cdots & P_K(\mathbf{x}_1) \\ P_1(\mathbf{x}_2) & P_2(\mathbf{x}_2) & \cdots & P_K(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ P_1(\mathbf{x}_N) & P_2(\mathbf{x}_N) & \cdots & P_K(\mathbf{x}_N) \end{pmatrix}$$

これらの最大から $\mathbf{x}_1$ が表す文字を予測

これらが一致すれば正解

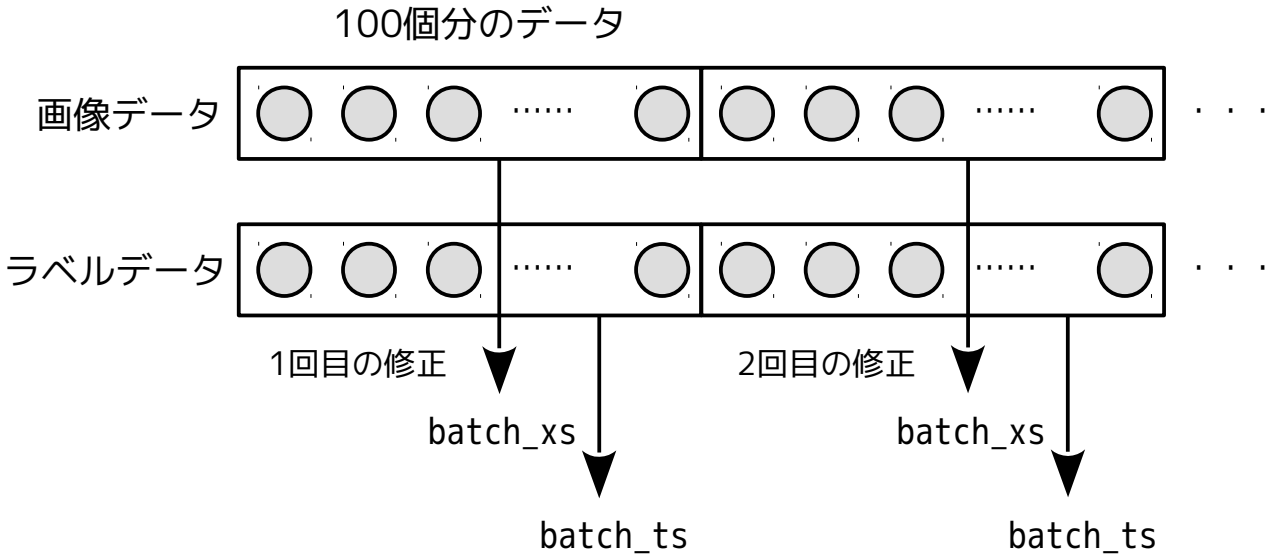
$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{21} & \cdots & t_{K1} \\ t_{12} & t_{22} & \cdots & t_{K2} \\ \vdots & \vdots & \vdots & \vdots \\ t_{1N} & t_{2N} & \cdots & t_{KN} \end{pmatrix}$$

$t_{k1} = 1$ となる k が $\mathbf{x}_1$ の正しい分類を示す

トレーニングセットによる最適化処理

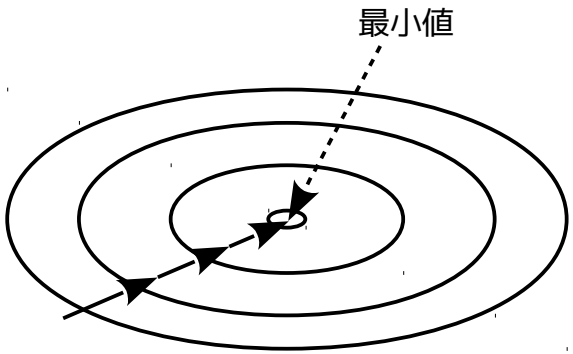
- ここでは、100個分のデータを順番にPlaceholderに代入して、最適化処理を繰り返します。

```
i = 0
for _ in range(2000):
    i += 1
    batch_xs, batch_ts = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, t: batch_ts})
    if i % 100 == 0:
        loss_val, acc_val = sess.run([loss, accuracy],
            feed_dict={x:mnist.test.images, t: mnist.test.labels})
        print ('Step: %d, Loss: %f, Accuracy: %f'
               % (i, loss_val, acc_val))
```

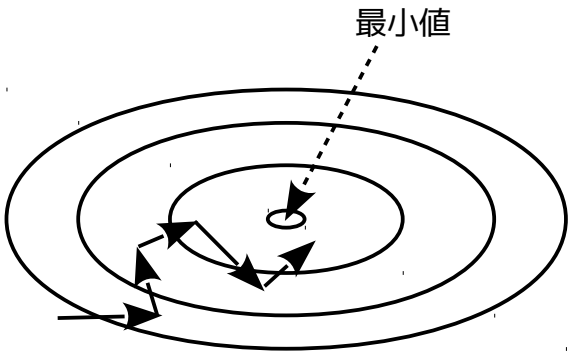


ミニバッチによる確率的勾配降下法の適用

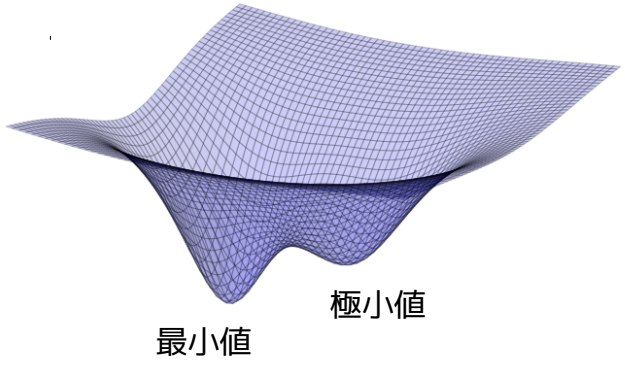
- ミニバッチを用いると次のようなメリットがあります。
  - 1回の最適化処理に必要なメモリー容量がおさえられます。
  - 最小値に向かってランダムに最適化が進むため、極小値にはまり込む可能性が低くなります。



すべてのデータを使用した
勾配降下法



ミニバッチによる
確率的勾配降下法



練習問題

- ノートブック「MNIST softmax estimation.ipynb」の内容を変更して、次のような実験をしてください。
  - 勾配降下法のアルゴリズムは、[MSE-04] の以下の部分で指定されています。AdamOptimizerは、学習率（勾配ベクトルの逆方向にどの程度パラメーターを修正するか）を自動調整するという特徴があります。

```
train_step = tf.train.AdamOptimizer().minimize(loss)
```

  - この部分を次のように変更すると、学習率を0.01に固定した単純な勾配降下法が用いられます。この時、実行結果がどのように変わるか確認してください。

```
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
```

  - 学習率を0.005、0.001などに変更した場合はどうでしょうか？ [MSE-07] の以下の部分で指定されている学習処理の繰り返し回数（2000）を変更することも可能です。

```
for _ in range(2000):
```

メモとしてお使いください

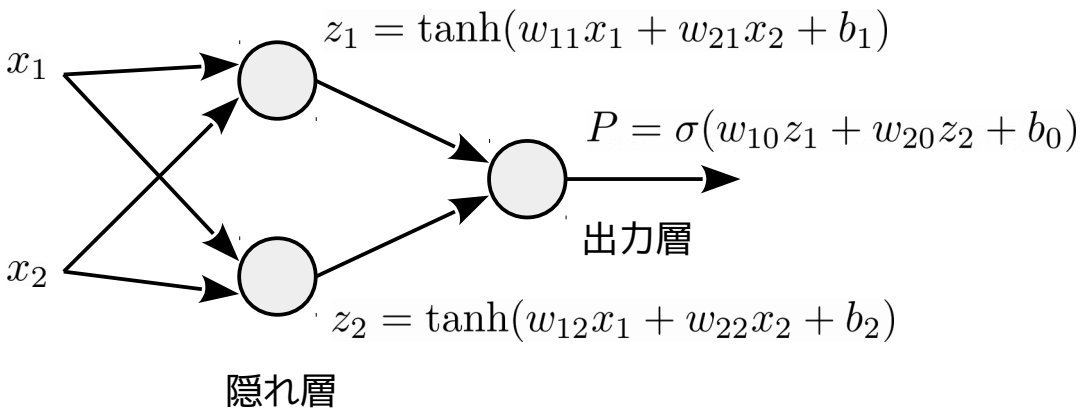
多層ニューラルネットワークによる特徴抽出

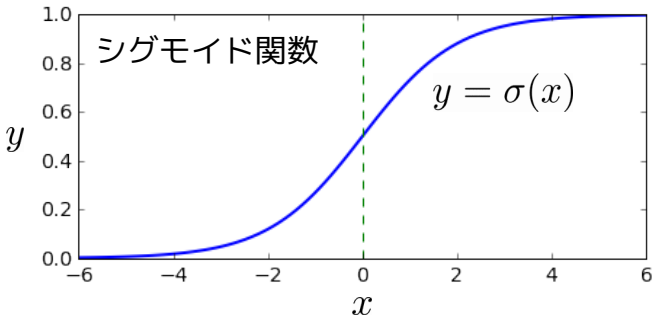
注意

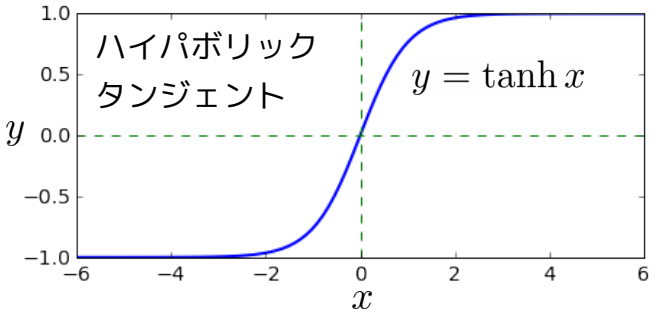
- このセクションで扱うコードの全体像は、次のノートブックを参照してください。
  - 「Chapter03/MNIST single layer network.ipynb」

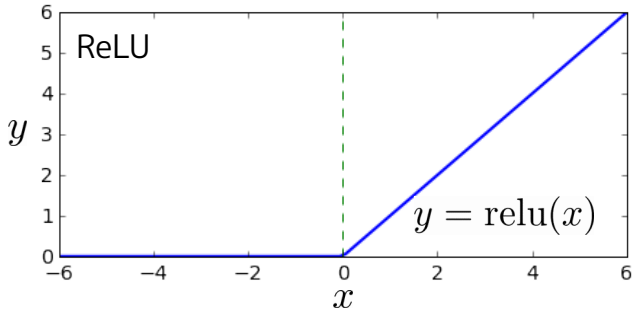
シンプルなニューラルネットワークの例

- 下図は、最もシンプルなニューラルネットワークの例です。
  - 隠れ層のノードは、１次関数の計算結果を「活性化関数」で変換した値を出力します。
  - 活性化関数には右図のようなものがあります。



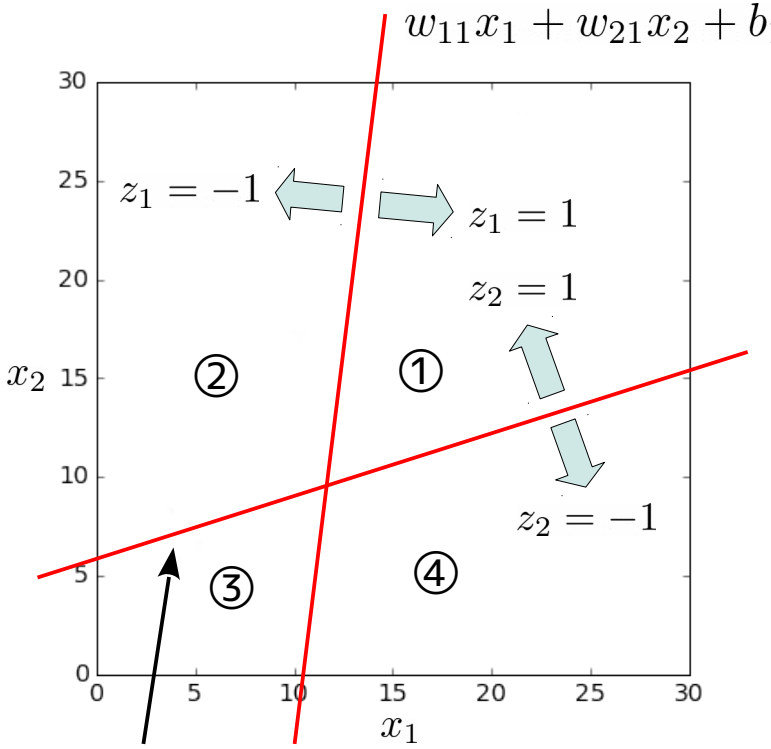


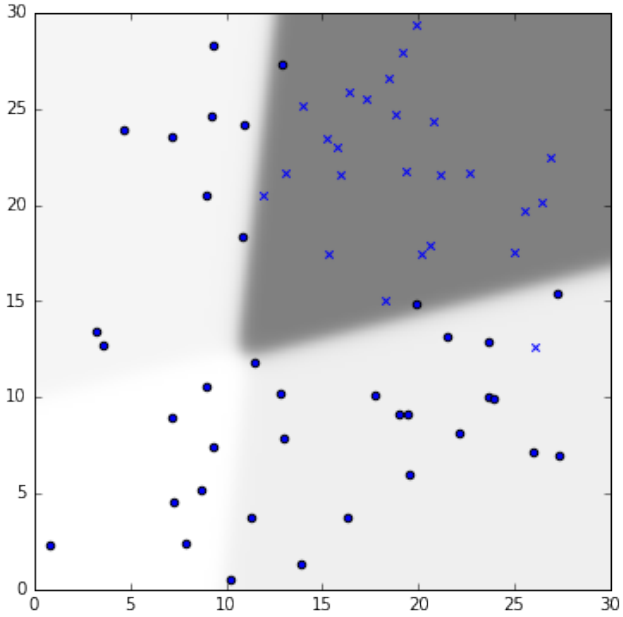




シンプルなニューラルネットワークの例

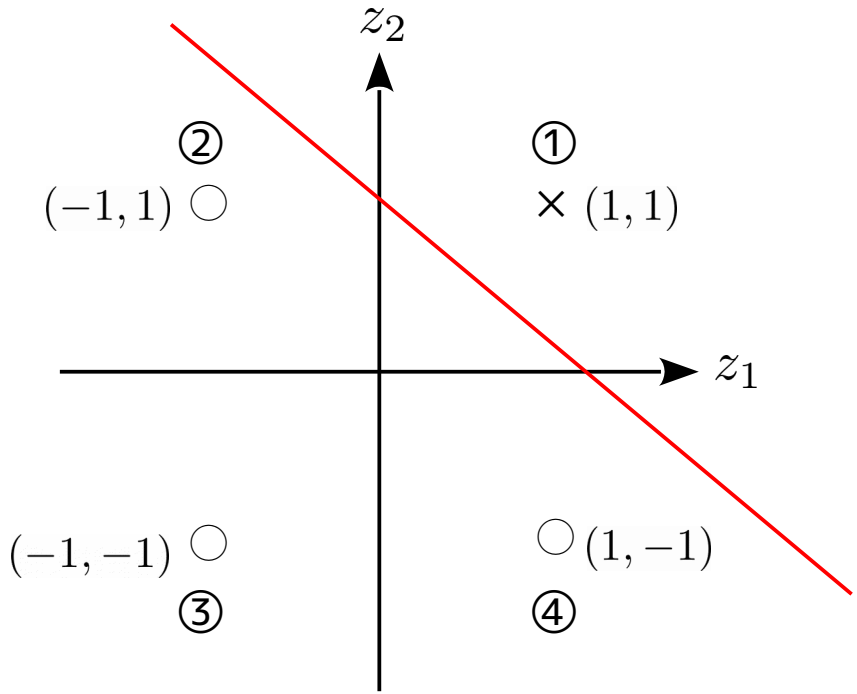
- ハイパボリックタンジェントは、-1から1に急激に変化するので、隠れ層のノードの出力は、１次関数が決定する直線の両側で、-1 と 1 に分かれると考えられます。
  - この例では、(x_1, x_2) 平面は、4つの領域に分類されることになります。

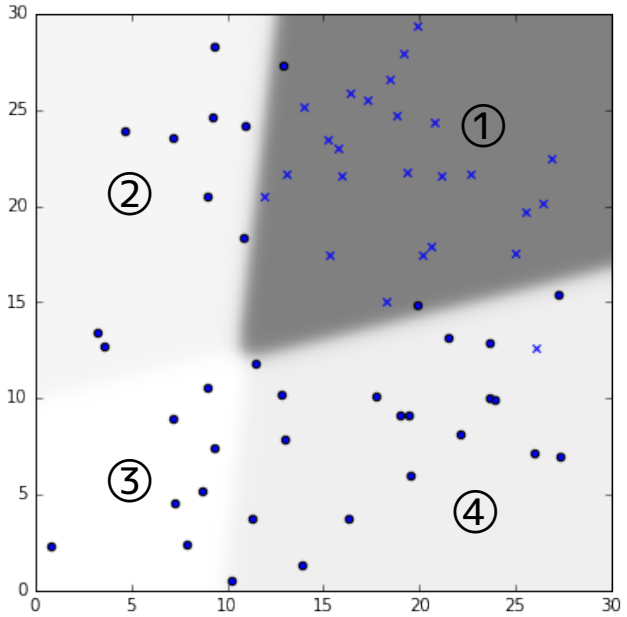




シンプルなニューラルネットワークの例

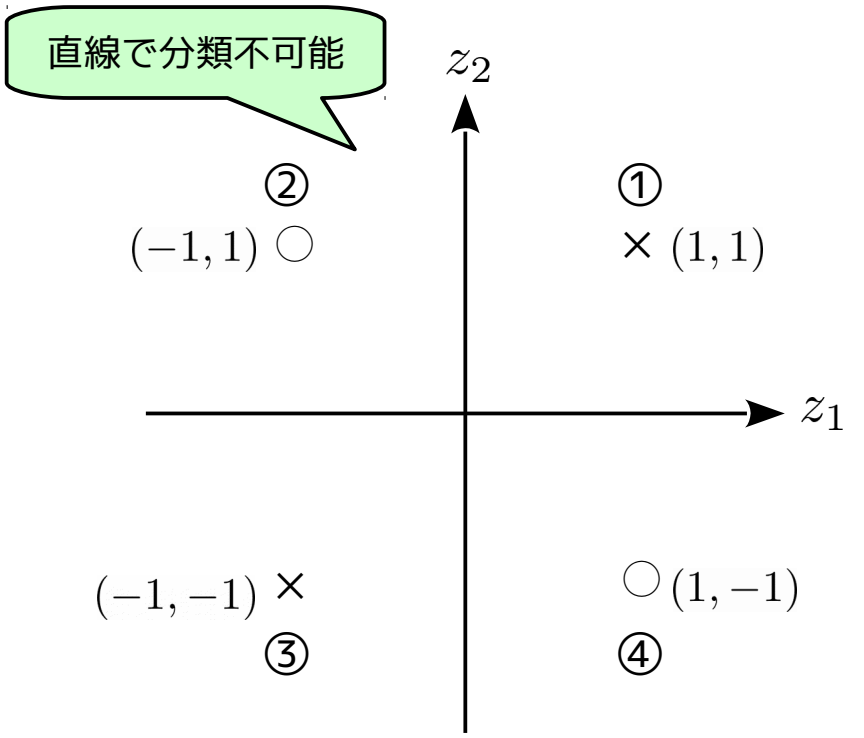
- 前ページの分類結果は、次のように理解することができます。
  - 出力層のシグモイド関数は、(z_1, z_2) 平面を直線で分類する機能を持つので、下図のように4つの領域を2種類に分類していると考えられます。

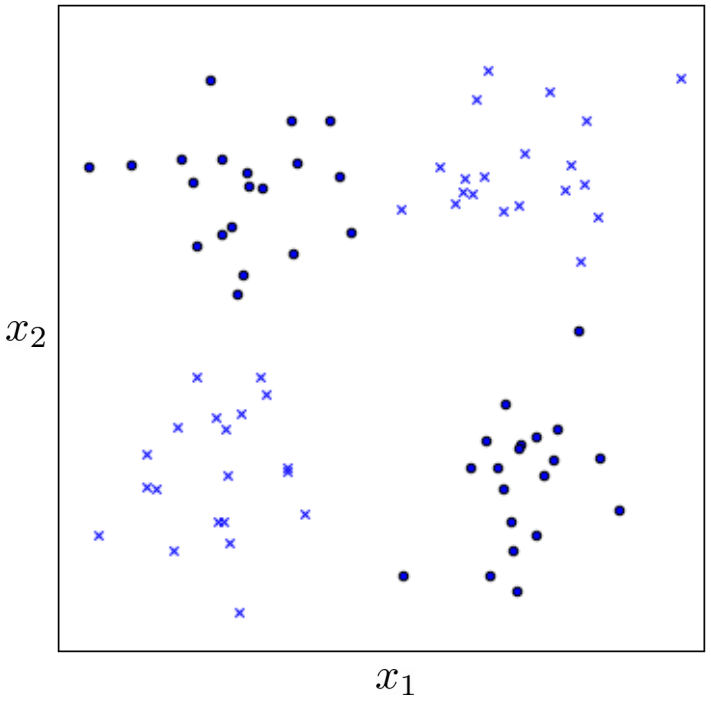




シンプルなニューラルネットワークの例

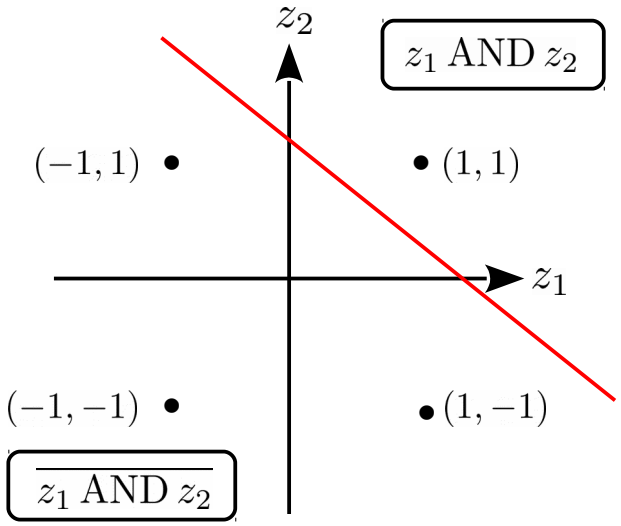
- 一方、下図のような配置パターンは、このニューラルネットワークでは分類できないことが分かります。
  - このような配置を分類するには、ニューラルネットワークをどのように拡張すればよいのでしょうか・・・？

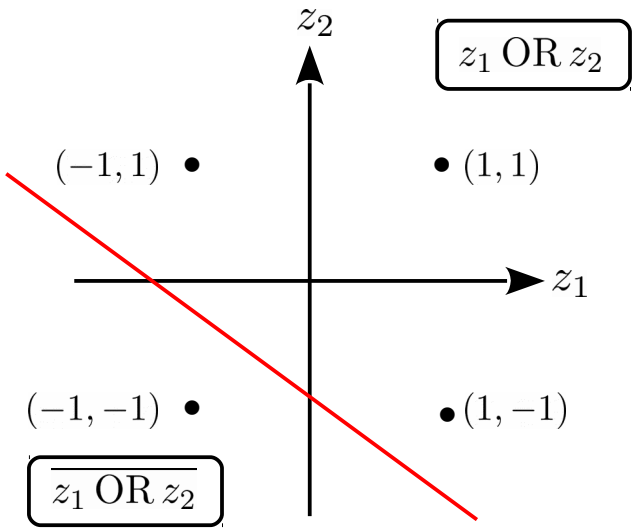




論理回路としてのニューラルネットワーク

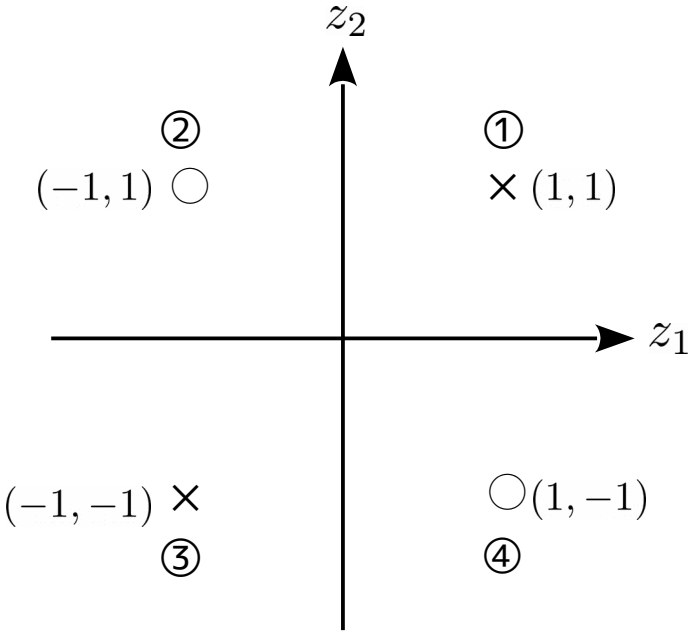
- １次関数と活性化関数を組み合わせた1つのノードは、図のように、AND もしくは OR 演算回路と同等であることが分かります。





論理回路としてのニューラルネットワーク

- 先ほどの配置パターンは、XOR演算に相当するので、下図のようにノードを組み合わせると分類可能になることが分かります。



AND計算

1 AND 1	1
1 AND 0	0
0 AND 1	0
0 AND 0	0

$\overline{1\ \text{AND}\ 1}$	0
$\overline{1\ \text{AND}\ 0}$	1
$\overline{0\ \text{AND}\ 1}$	1
$\overline{0\ \text{AND}\ 0}$	1

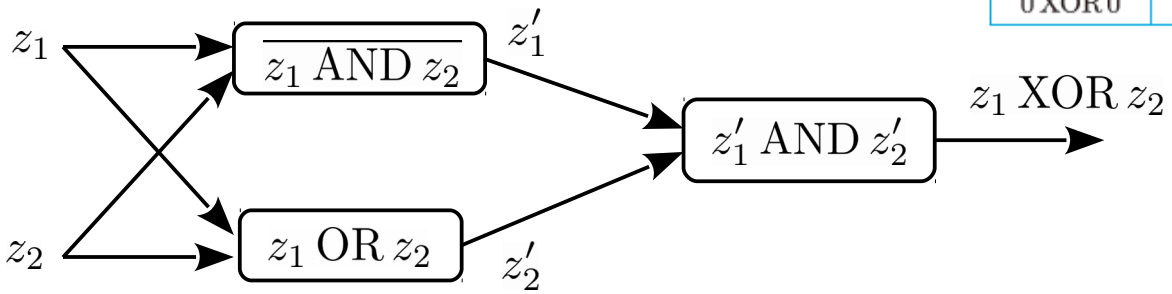
OR計算

1 OR 1	1
1 OR 0	1
0 OR 1	1
0 OR 0	0

$\overline{1\ \text{OR}\ 1}$	0
$\overline{1\ \text{OR}\ 0}$	0
$\overline{0\ \text{OR}\ 1}$	0
$\overline{0\ \text{OR}\ 0}$	1

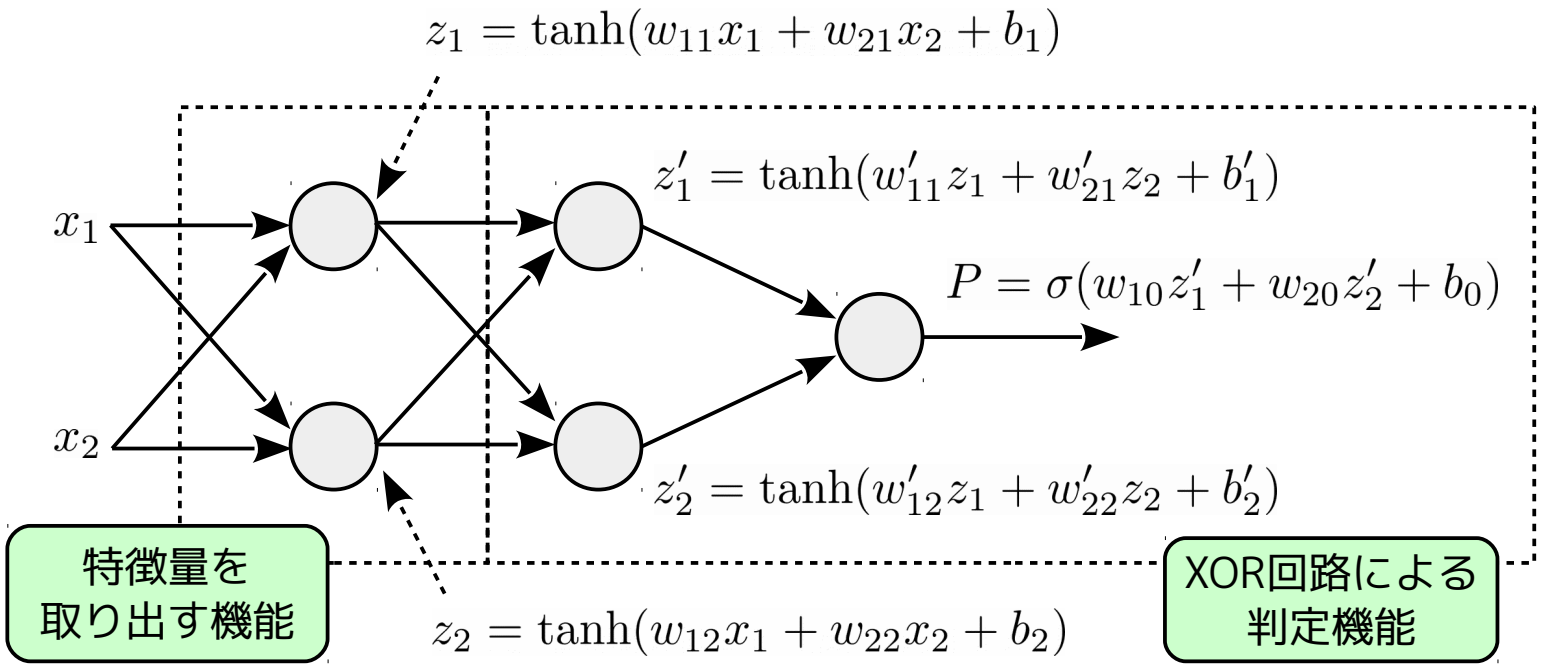
XOR計算

1 XOR 1	0
1 XOR 0	1
0 XOR 1	1
0 XOR 0	0



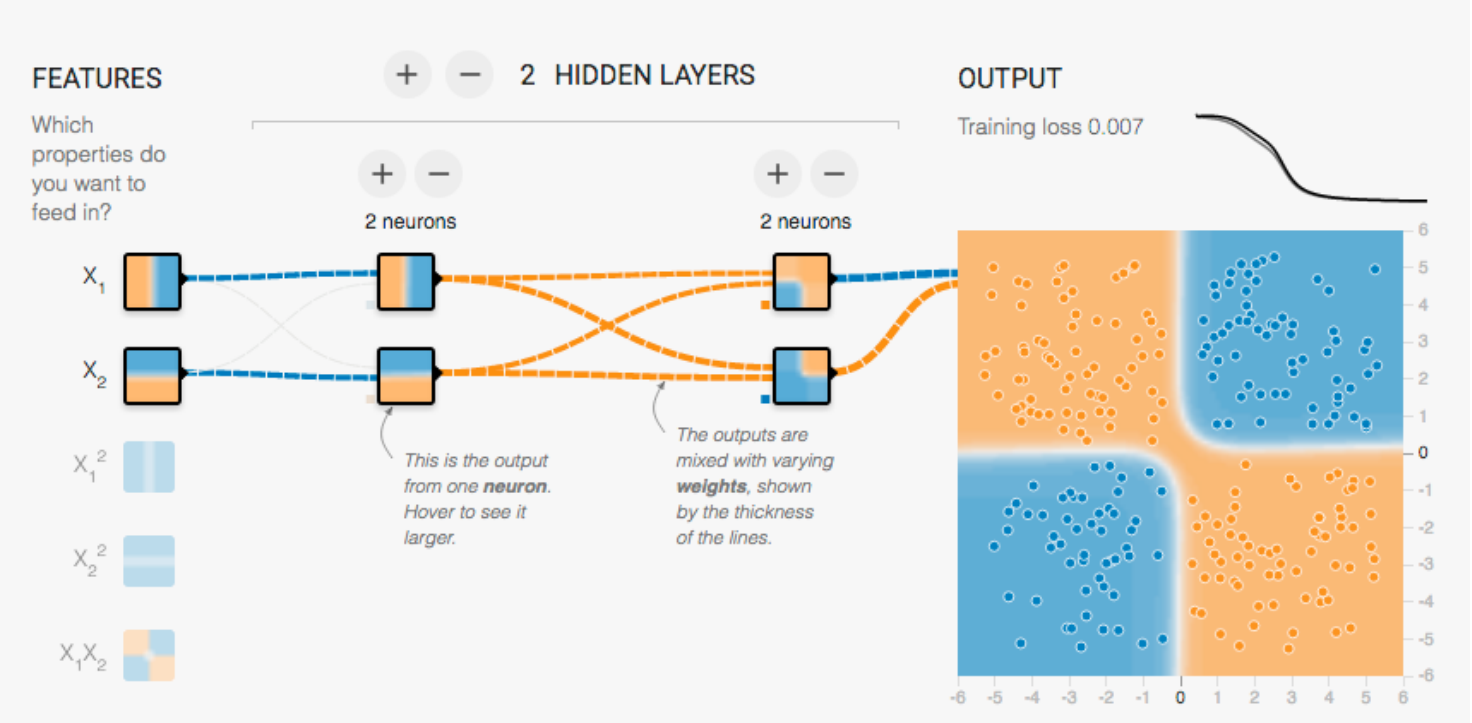
論理回路としてのニューラルネットワーク

- (x_1, x_2) 平面を4分割する1層目の隠れ層に、XOR演算機能を持つ2層目の隠れ層を結合すると次のニューラルネットワークが得られます。
  - 1層目の隠れ層は、入力値 (x_1, x_2) を $z_1 = \pm 1, z_2 = \pm 1$ のバイナリー値に変換しています。これは、与えられたデータは、2つのバイナリー変数 (z_1, z_2) で特徴づけられることを意味しています。



論理回路としてのニューラルネットワーク

- Neural Network Playgroundを利用して、実際に確認してみましょう。
  - http://goo.gl/VIvOaQ



（参考）MNISTへのニューラルネットワークの適用

- 隠れ層が1層のニューラルネットワークでMNISTデータセットを分類する場合の計算例は次のようになります。
  - 下記は隠れ層の出力値 Z を計算する部分です。この結果をソフトマックス関数に入力して、「0確率を計算します。

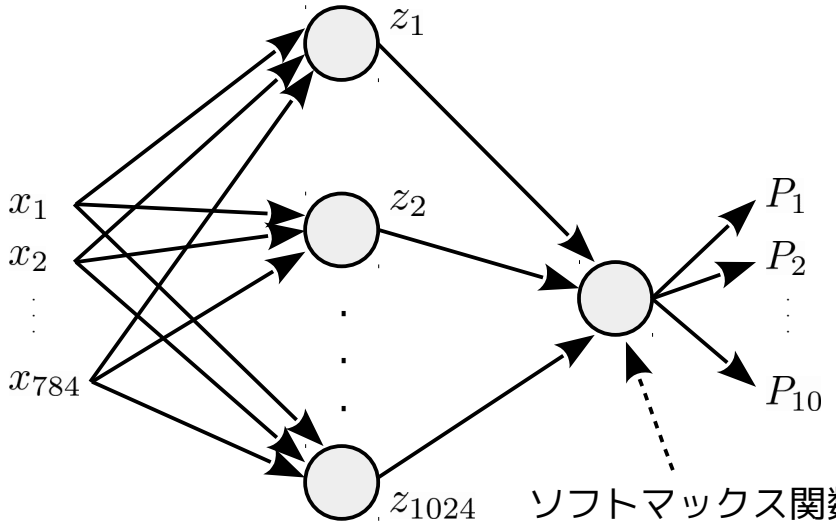
$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{784\,1} \\ x_{12} & x_{22} & \cdots & x_{784\,2} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1\,1024} \\ w_{21} & w_{22} & \cdots & w_{2\,1024} \\ \vdots & \vdots & \vdots & \vdots \\ w_{784\,1} & w_{784\,2} & \cdots & w_{784\,1024} \end{pmatrix}$$

$$\mathbf{Z} = \begin{pmatrix} z_{11} & z_{21} & \cdots & z_{1024\,1} \\ z_{12} & z_{22} & \cdots & z_{1024\,2} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\mathbf{w} = (w_{01}, \cdots, w_{0\,1024})$$

$$\mathbf{Z} = \mathrm{relu}(\mathbf{XW} + \mathbf{w})$$



（参考）MNISTへのニューラルネットワークの適用

- 隠れ層の計算、および、ソフトマックス関数の計算をTensorFlowのコードに直すと下記になります。

```
num_units = 1024

x = tf.placeholder(tf.float32, [None, 784])

w1 = tf.Variable(tf.truncated_normal([784, num_units]))
b1 = tf.Variable(tf.zeros([1, num_units]))
hidden1 = tf.nn.relu(tf.matmul(x, w1) + b1)

w0 = tf.Variable(tf.zeros([num_units, 10]))
b0 = tf.Variable(tf.zeros([10]))
p = tf.nn.softmax(tf.matmul(hidden1, w0) + b0)
```

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{784\,1} \\ x_{12} & x_{22} & \cdots & x_{784\,2} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1\,1024} \\ w_{21} & w_{22} & \cdots & w_{2\,1024} \\ \vdots & \vdots & \vdots & \vdots \\ w_{784\,1} & w_{784\,2} & \cdots & w_{784\,1024} \end{pmatrix}$$

$$\mathbf{w} = (w_{01}, \cdots, w_{0\,1024})$$

$$\mathbf{P} = \begin{pmatrix} P_1(\mathbf{x}_1) & P_2(\mathbf{x}_1) & \cdots & P_{10}(\mathbf{x}_1) \\ P_1(\mathbf{x}_2) & P_2(\mathbf{x}_2) & \cdots & P_{10}(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

$$\mathbf{Z} = \begin{pmatrix} z_{11} & z_{21} & \cdots & z_{1024\,1} \\ z_{12} & z_{22} & \cdots & z_{1024\,2} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

- $\mathbf{Z}$ から10種類の1次関数 f を構成して、ソフトマックス関数に入力する以降の計算と、その後の最適化処理は、これまでと同じになります。

練習問題

- Neural Network Playgroundで、隠れ層やノードの数を変更して、結果がどのように変わるか観察してください。

- ノートブック「MNIST single layer network.ipynb」の内容を変更して、次のような実験をしてください。
  - 隠れ層のノード数は、[MSL-03] の以下の部分で指定されています。ノード数を 128, 512, 4096 などに変更して、結果がどのように変わるか観察してください。

```
num_units = 1024
```

メモとしてお使いください

メモとしてお使いください

メモとしてお使いください

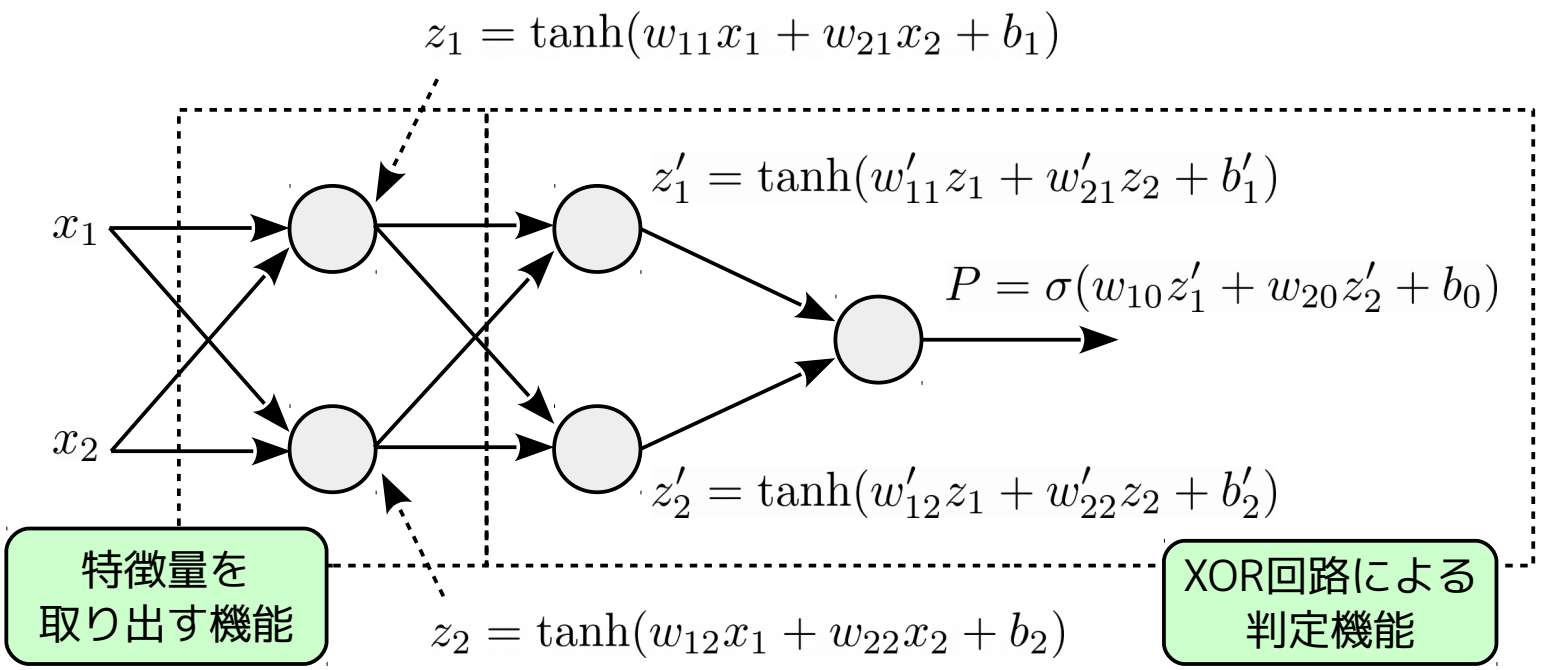
畳み込みフィルターによる画像の特徴抽出

注意

- このセクションで扱うコードの全体像は、次のノートブックを参照してください。
  - 「Chapter04/ORENIST filter example.ipynb」
  - 「Chapter04/ORENIST classification example.ipynb」

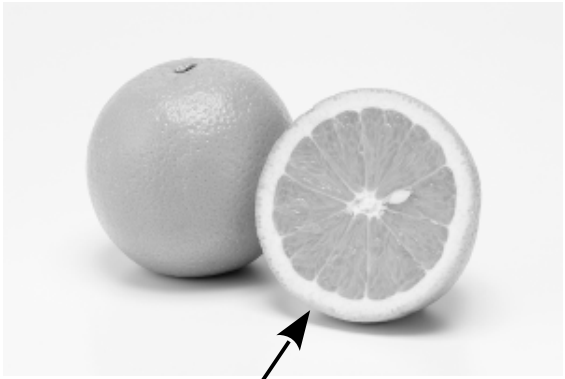
多層ニューラルネットワークの構造

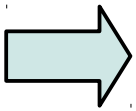
- 前セクションで次のような構造を確認しました。
  - この例では、第1層の2個のノードでデータの分類に必要な特徴 (z_1, z_2) が抽出できました。
  - より一般的な画像ファイルの特徴を抽出するには、どのような隠れ層を用いればよいのでしょうか？



畳み込みフィルターの仕組み

- 下図は画像をぼかす効果を持った畳み込みフィルターの例です。
  - あるピクセルの色を周りのピクセルの色とフィルターの重みで混ぜあわせます。

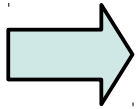




91	46	35
141	140	135
156	153	153

×

0.11	0.11	0.11
0.11	0.12	0.11
0.11	0.11	0.11

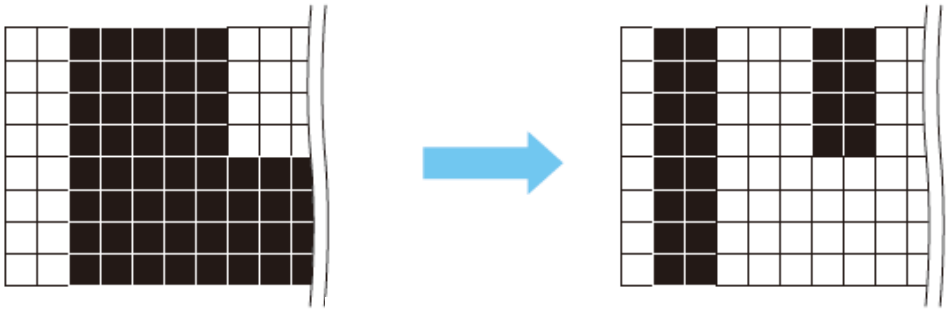


117

$$91 \times 0.11 + 46 \times 0.11 + 35 \times 0.11$$
$$+141 \times 0.11 + 140 \times 0.12 + 135 \times 0.11$$
$$+156 \times 0.11 + 153 \times 0.11 + 153 \times 0.11 = 116.9$$

エッジを抽出する畳み込みフィルター

- 右図は縦のエッジを抽出するフィルターの例です。
  - 横に同じ色が続く部分は、左右の±がキャンセルして0になります。
  - 計算した合計が負になる場合は絶対値を取ります。



-1	0	1
-2	0	2
-1	0	1

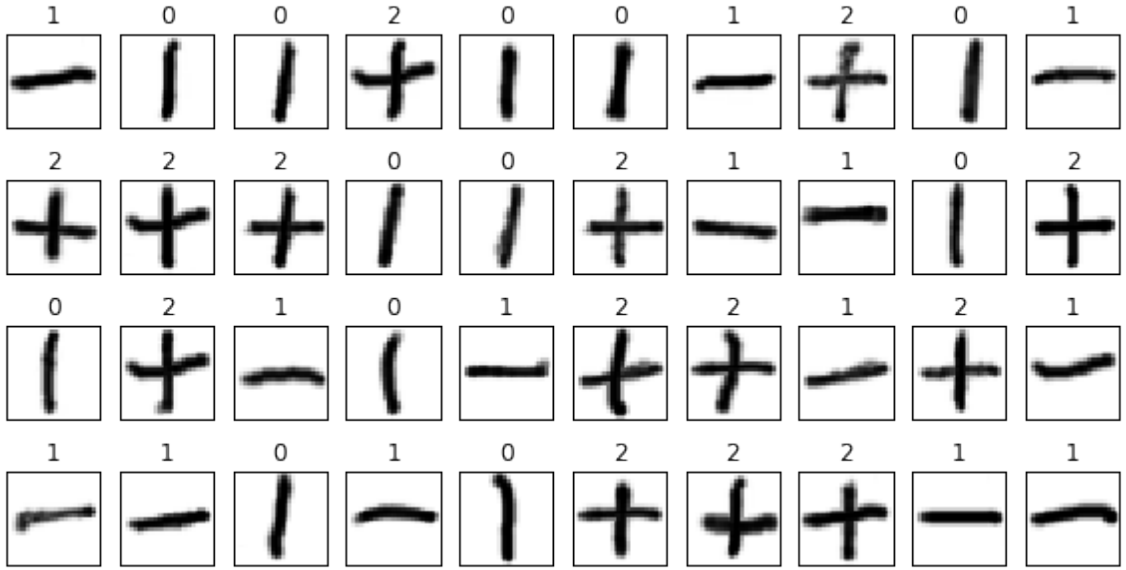
- フィルターのサイズを大きくすると、さらに太い幅でエッジを残します。
  - 右図は、縦／横、それぞれのエッジを抽出します。
  - フィルターの値が大きいと色が濃くなる効果が入るため、実際には、全体を23.0で割った値を使用します。

2	1	0	-1	-2
3	2	0	-2	-3
4	3	0	-3	-4
3	2	0	-2	-3
2	1	0	-1	-2

2	3	4	3	2
1	2	3	2	1
0	0	0	0	0
-1	-2	-3	-2	-1
-2	-3	-4	-3	-2

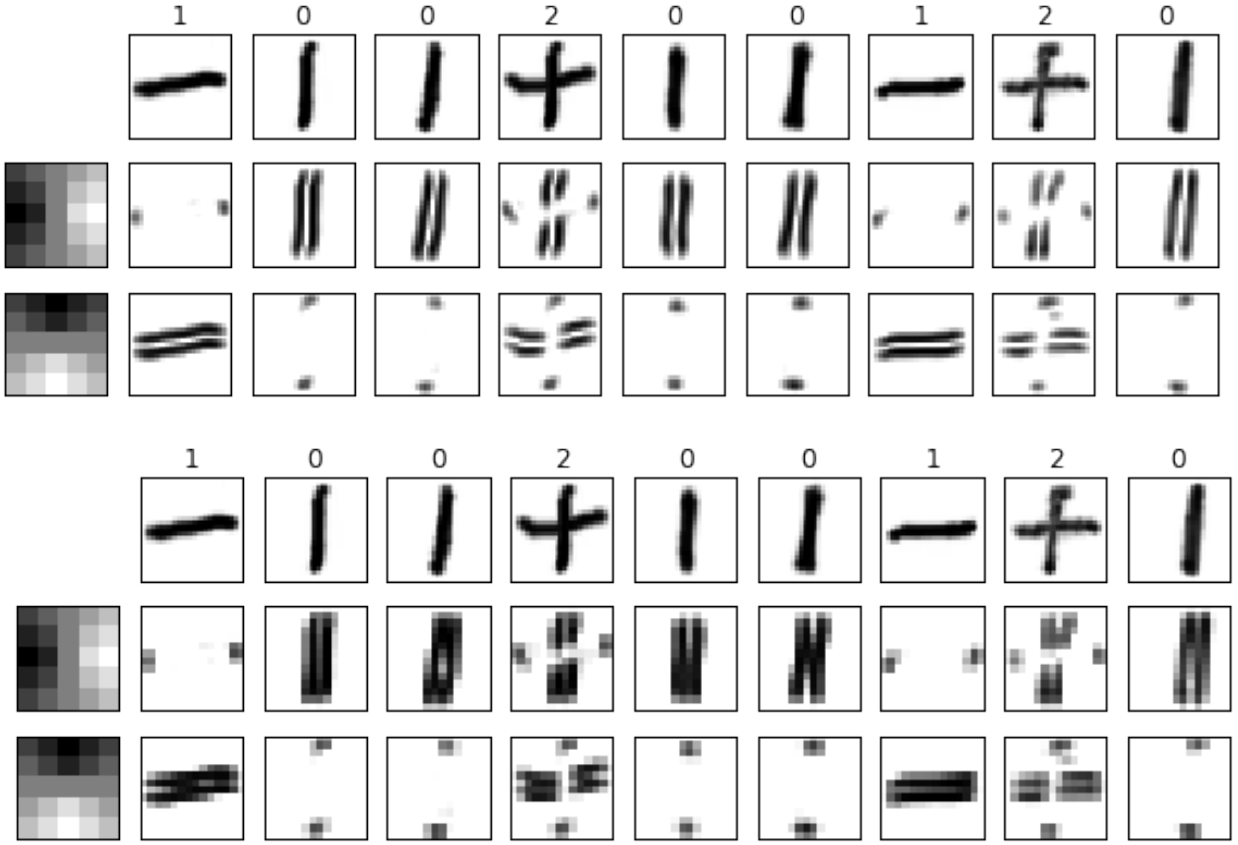
ORENISTデータセットによる考察

- たとえば、下図の画像を分類する場合、「縦棒」と「横棒」という特徴を抽出すればよいことが分かります。
  - 画像処理の世界では、「畳み込みフィルター」によって特定方向のエッジを抽出する手法が知られています。



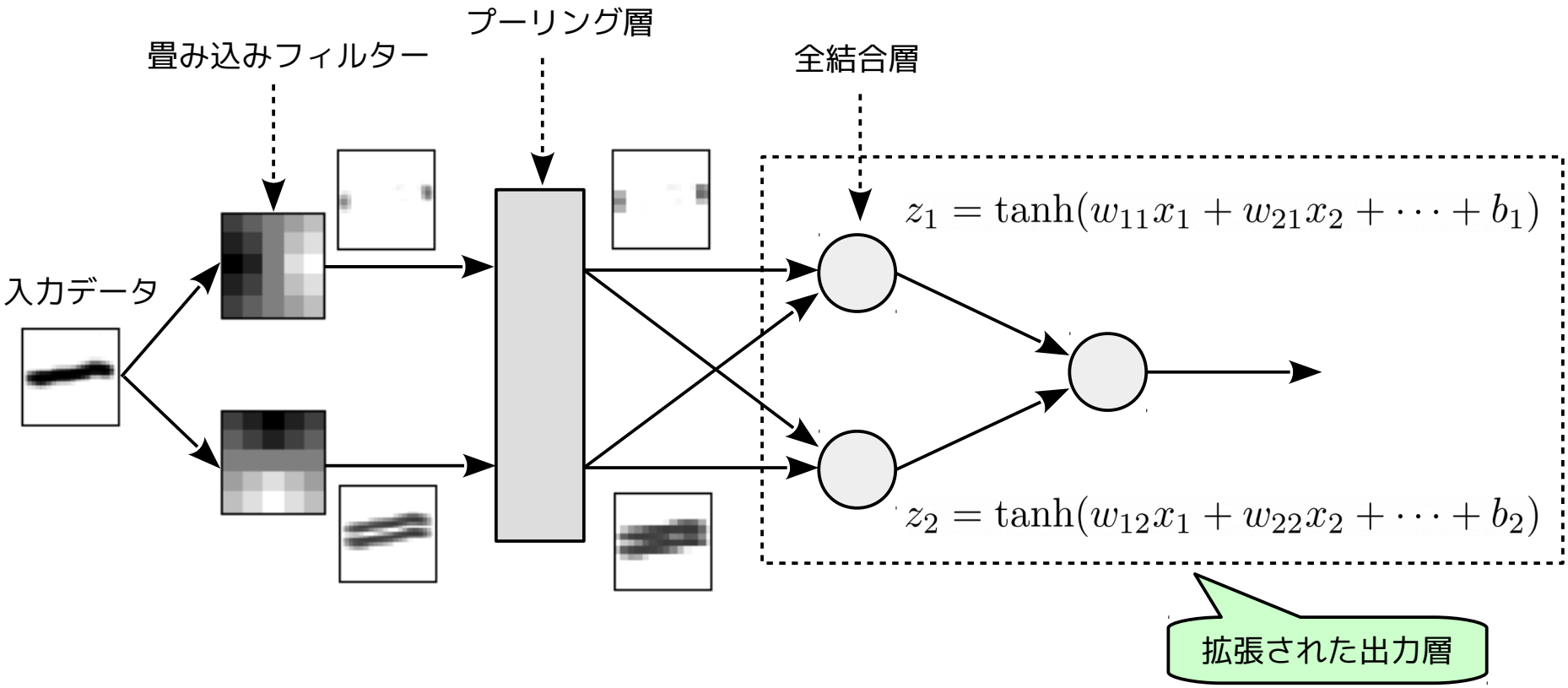
エッジを抽出する畳み込みフィルター

- ORENISTデータセットに前ページのフィルターを適用すると、次の結果が得られます。
  - 下段は、プーリング層でさらに解像度を落とした結果です。2×2ピクセルの領域を1ピクセルで置き換えています。



エッジを抽出する畳み込みフィルター

- 縦横のエッジを抽出した結果がこのデータを分類する「特徴」だとすると、その結果を全結合層に入力することで、正しく分類できると期待できます。



TensorFlowにおけるフィルターの構造

- TensorFlowのコード内で畳み込みフィルターを定義する際は、すべてのフィルターをまとめて、下記のサイズの多次元リストに格納します。
  - 「縦×横×入力レイヤー数×出力レイヤー数」
  - カラー画像の場合、入力レイヤー数は、RGBの3種類になります。今はグレイスケール画像なので入力レイヤー数は1です。また、1枚の画像から2枚の画像を出力するので、出力レイヤー数は2です。

```
def edge_filter():
    filter0 = np.array(
            [[ 2, 1, 0,-1,-2],
             [ 3, 2, 0,-2,-3],
             [ 4, 3, 0,-3,-4],
             [ 3, 2, 0,-2,-3],
             [ 2, 1, 0,-1,-2]]) / 23.0
    filter1 = np.array(
            [[ 2, 3, 4, 3, 2],
             [ 1, 2, 3, 2, 1],
             [ 0, 0, 0, 0, 0],
             [-1,-2,-3,-2,-1],
             [-2,-3,-4,-3,-2]]) / 23.0

    filter_array = np.zeros([5,5,1,2])
    filter_array[:,:,0,0] = filter0
    filter_array[:,:,0,1] = filter1

    return tf.constant(filter_array, dtype=tf.float32)
```

縦／横のエッジを抽出するフィルターを定義する関数

それぞれのフィルターを2次元リストとして用意

すべてのフィルターを格納する多次元リストを用意

該当部分にフィルターの値を格納

定数オブジェクトに変換して返却

TensorFlowにおけるフィルターの構造

- 畳み込みフィルターを適用する際は、次の手順に従います。
  - 入力画像データを「データ数×縦×横×レイヤー数」というサイズのリストにまとめます。
  - 関数tf.nn.conv2dに入力画像データとフィルターを入力します。

```
x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])

W_conv = edge_filter()
h_conv = tf.abs(tf.nn.conv2d(x_image, W_conv,
                             strides=[1,1,1,1], padding='SAME'))
h_conv_cutoff = tf.nn.relu(h_conv-0.2)

h_pool =tf.nn.max_pool(h_conv_cutoff, ksize=[1,2,2,1],
                       strides=[1,2,2,1], padding='SAME')
```

入力画像データをリストにまとめる（先頭の「-1」はデータ数に応じて適切な値に自動変換される）

畳み込みフィルターを適用

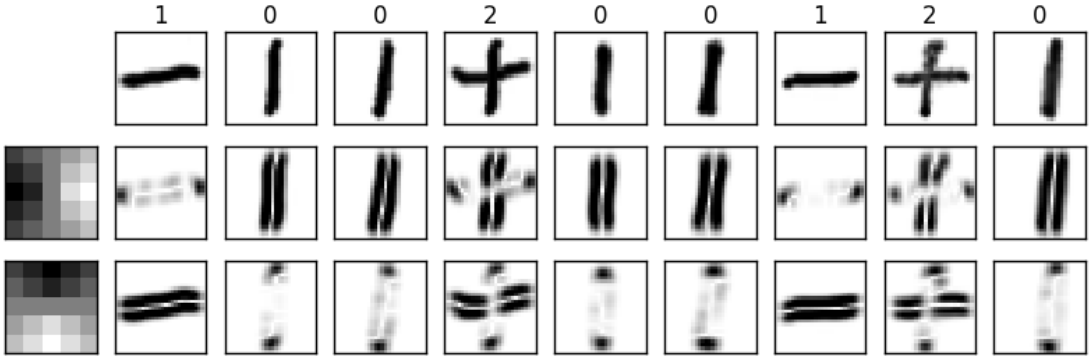
ピクセル値のカットオフ（次ページ参照）

プーリング層を適用

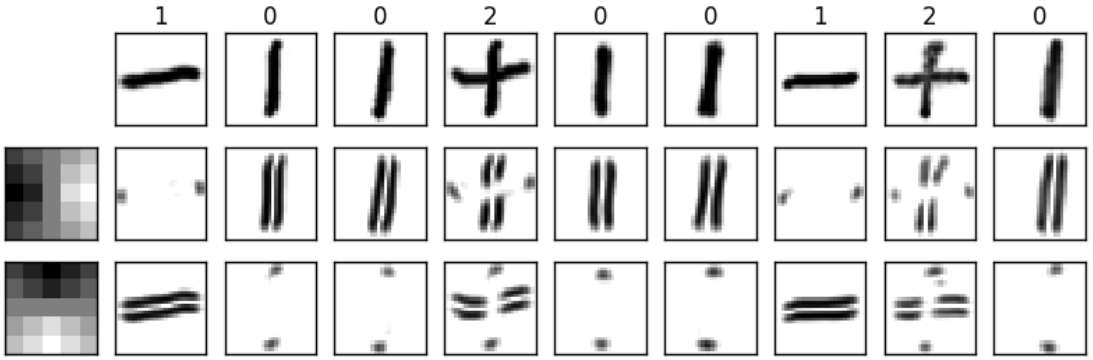
- その他のオプションは次の通りです。
  - strides=[1,dy,dx,1] : (dy, dy) ピクセルごとに計算する
  - padding='SAME' : フィルターが画像をはみ出る部分は値を「0」として計算する
  - ksize =[1,2,2,1]（プーリング層）：2x2ピクセルを1ピクセルに置き換える

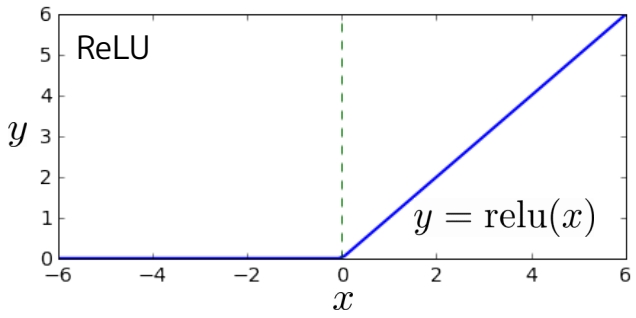
ReLUを用いたピクセル値のカットオフ

- 畳み込みフィルターでエッジを抽出した後に、ピクセル値が0.2以下の部分を強制的に0にすることで、エッジの抽出効果を強調しています。



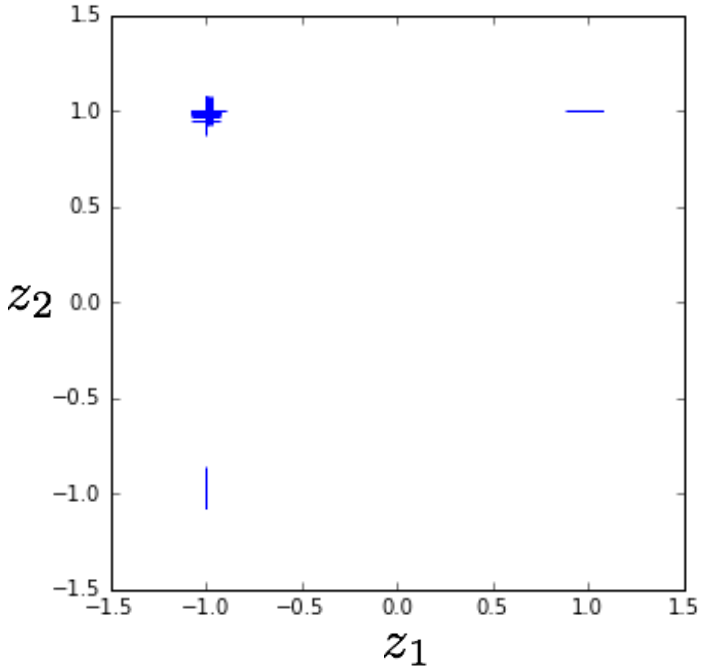

```
h_conv_cutoff = tf.nn.relu(h_conv-0.2)
```





全結合層による特徴変数の変換

- 先ほどのニューラルネットワークでパラメーターの最適化を行った後に、それぞれの入力データに対応する (z_1, z_2)の値を散布図に描くと下記が得られます。
  - z_1, z_2 のそれぞれが、縦棒と横棒の有無を表すバイナリー変数になっていることが分かります。



畳み込みフィルターの動的な学習

注意

- このセクションで扱うコードの全体像は、次のノートブックを参照してください。
  - 「Chapter04/MNIST dynamic filter classification.ipynb」
  - 「Chapter04/MNIST dynamic filter result.ipynb」
  - 「Chapter05/MNIST double layer CNN classification.ipynb」

畳み込みフィルターの最適化

- 一般の画像データの場合、どのようなフィルターが適切なのかは、すぐには分かりません。そこで、フィルターを最適化対象のVariableにすることで、適切なフィルターそのものを学習させてしまいます。
  - フィルターの初期値は乱数で設定します。
  - ORENISTデータセットの場合、「エッジを抽出する」という目的のために畳み込みフィルターを適用した後に絶対値をとりました。ここでは、エッジ抽出ではなく、より一般的な「特徴の抽出」が目的なので、絶対値をとる処理は省略します。

MNISTデータセットに16個のフィルターを適用するコード

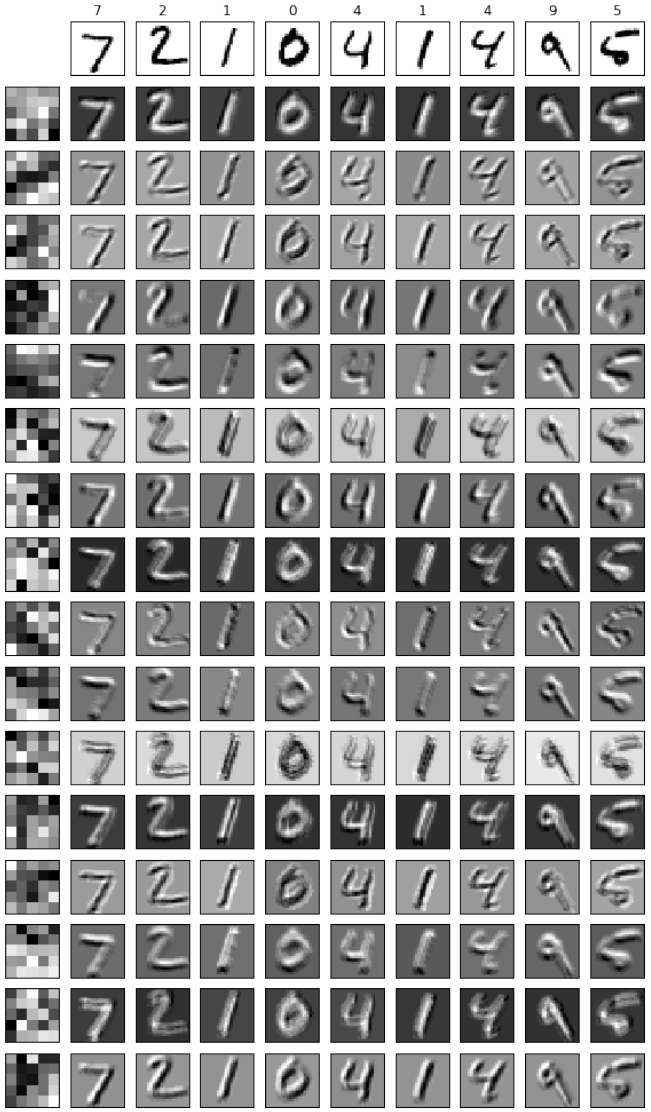
```
num_filters = 16

x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])

W_conv = tf.Variable(tf.truncated_normal([5,5,1,num_filters], stddev=0.1))
h_conv = tf.nn.conv2d(x_image, W_conv,
                      strides=[1,1,1,1], padding='SAME')
h_pool =tf.nn.max_pool(h_conv, ksize=[1,2,2,1],
                       strides=[1,2,2,1], padding='SAME')
```

Variableとしてフィルターを定義

畳み込みフィルターの学習例

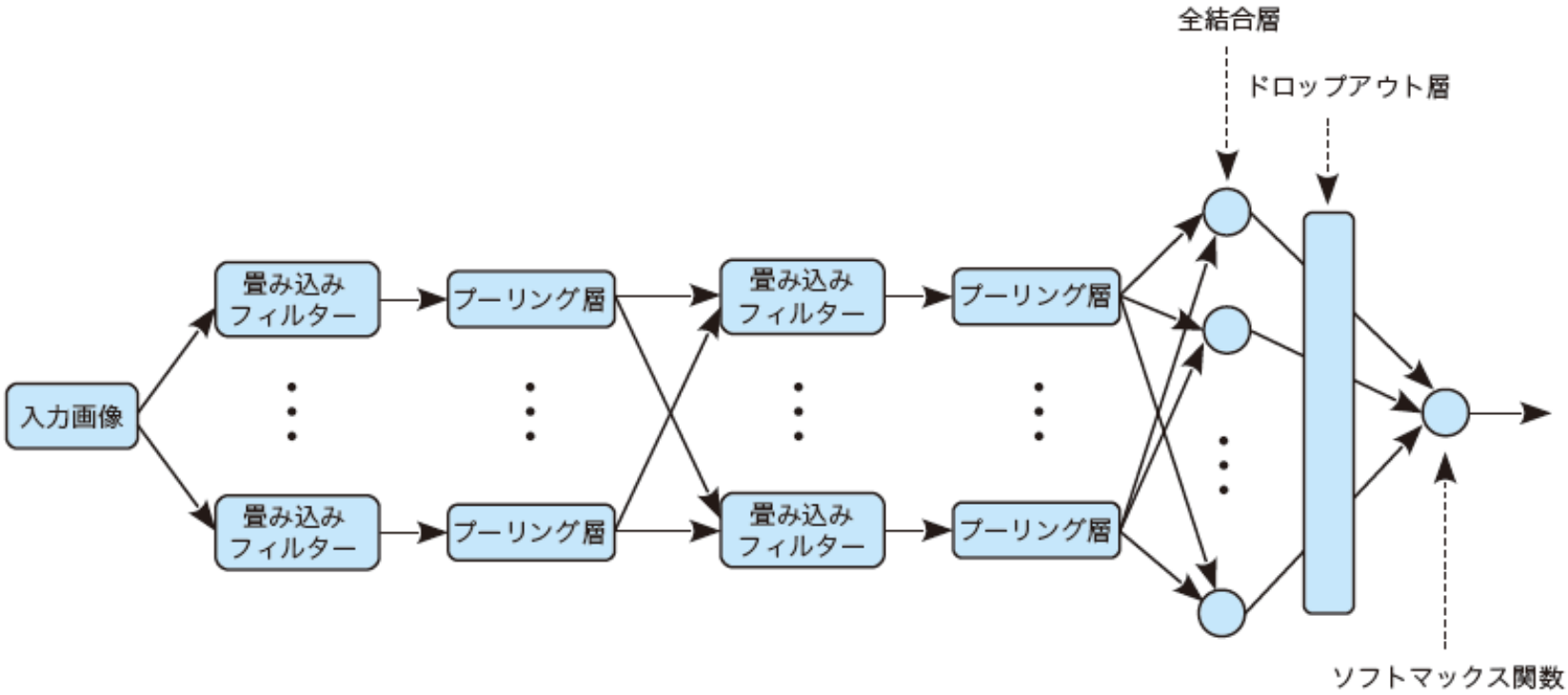
- 右図は動的に学習したフィルターの例です。
  - 左端がフィルターで、右側はフィルターを適用した画像の例です。
  - 絶対値を取っていないので、フィルター後の値が負になる場合があるため、背景が白になっていません。

- 最上段の文字画像の代わりに、その下にある16枚の画像を全結合層に入力することで、より高精度な分類が可能になります。



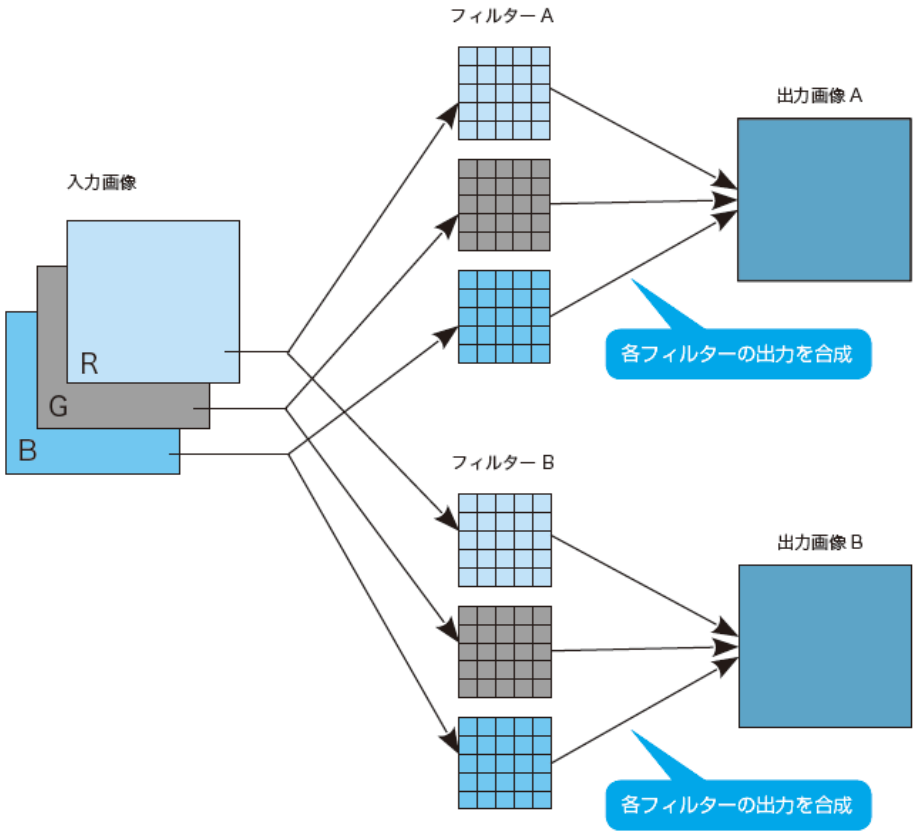
畳み込みフィルターの多層化

- 「畳み込みフィルター＋プーリング層」を通過した画像に対して、さらにもう一度、畳み込みフィルターを適用することで、新たな「特徴」が抽出される可能性はないでしょうか？
  - この考え方を実装したものが下図の「畳み込みニューラルネットワーク」にほかなりません。



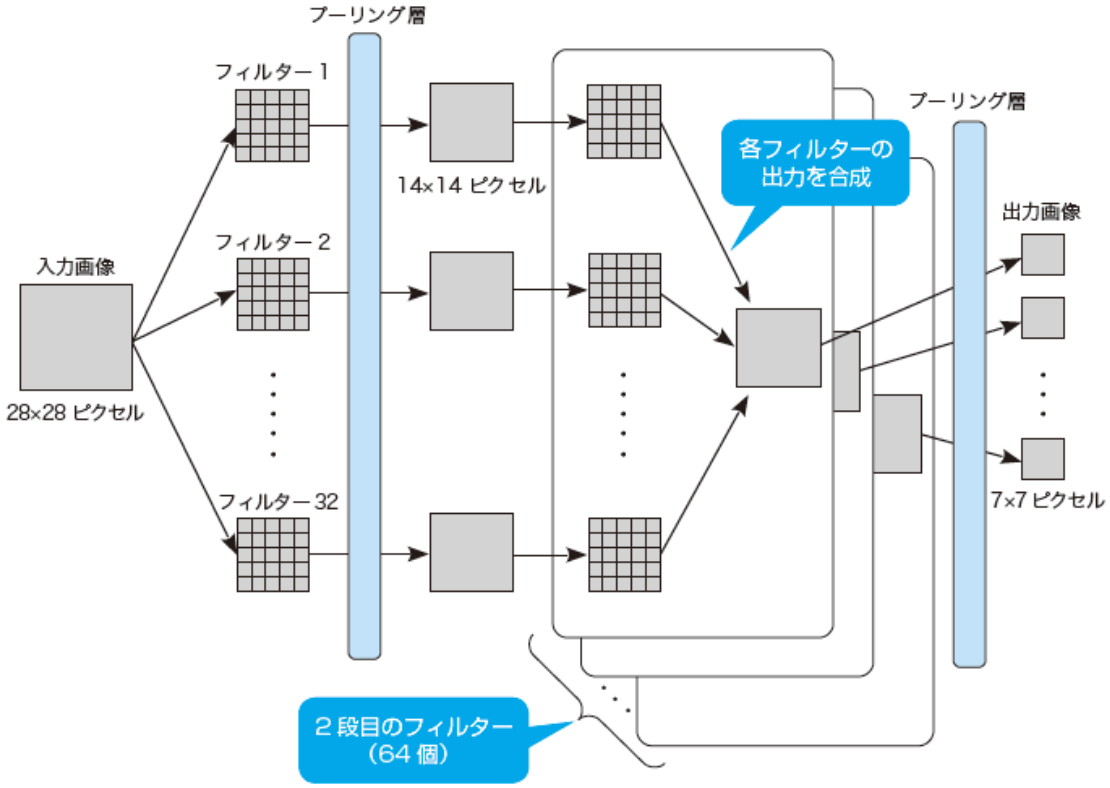
複数レイヤーに対するフィルター適用の仕組み

- １つのフィルターは、各レイヤーに適用するサブフィルターを内部的に保持しており、各サブフィルターの出力を合成したものが出力画像になります。
  - RGB画像ファイルを入力しても、その出力はRGB画像ファイルになるわけではありません。



２層目のフィルターの適用方法

- １層目のフィルターから得られた n 枚の画像データを「n 個のレイヤーからなる画像」として、2層目のフィルターを適用します。
  - 下図は1層目に32個、2層目に64個のフィルターを用意した例です。2層目の64個のフィルターは、それぞれ内部的に32個のサブフィルターを持ちます。

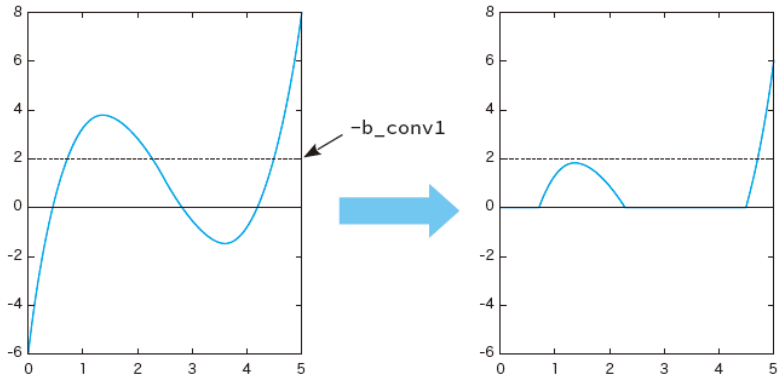


畳み込みフィルターを適用するコード

- 1層目の畳み込みフィルターとプーリング層を適用するコードの例です。

```
num_filters1 = 32

x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1,28,28,1])

W_conv1 = tf.Variable(tf.truncated_normal([5,5,1,num_filters1],
                                          stddev=0.1))
h_conv1 = tf.nn.conv2d(x_image, W_conv1,
                       strides=[1,1,1,1], padding='SAME')

b_conv1 = tf.Variable(tf.constant(0.1, shape=[num_filters1]))
h_conv1_cutoff = tf.nn.relu(h_conv1 + b_conv1)

h_pool1 = tf.nn.max_pool(h_conv1_cutoff, ksize=[1,2,2,1],
                         strides=[1,2,2,1], padding='SAME')
```

- 畳み込みフィルターからの出力に対して、一定値以下の値を0にカットする処理を挿入しています。これはフィルターの影響を強調する効果があります。



畳み込みフィルターを適用するコード

- ２層目の畳み込みフィルターとプーリング層を適用するコードの例です

```
num_filters2 = 64

W_conv2 = tf.Variable(
           tf.truncated_normal([5,5,num_filters1,num_filters2],
                               stddev=0.1))
h_conv2 = tf.nn.conv2d(h_pool1, W_conv2,
                       strides=[1,1,1,1], padding='SAME')

b_conv2 = tf.Variable(tf.constant(0.1, shape=[num_filters2]))
h_conv2_cutoff = tf.nn.relu(h_conv2 + b_conv2)

h_pool2 = tf.nn.max_pool(h_conv2_cutoff, ksize=[1,2,2,1],
                         strides=[1,2,2,1], padding='SAME')
```

  - フィルターのデータは、「縦×横×入力レイヤー数×出力レイヤー数」というサイズの多次元リストに格納しました。この例では、[5,5,32,64] というサイズになります。

ドロップアウト層の役割について

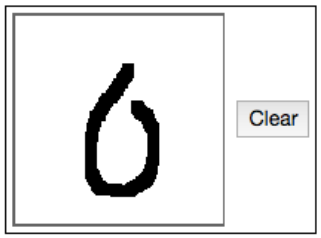
- このモデルでは、全結合層と出力層の間に「ドロップアウト層」が挿入されています。
  - ドロップアウト層は、一定の割合で全結合層からの入力を0にします。（「結合を残す割合」をオプションで指定します。）
  - 学習時は50%をドロップすることで、オーバーフィッティングを防止します。

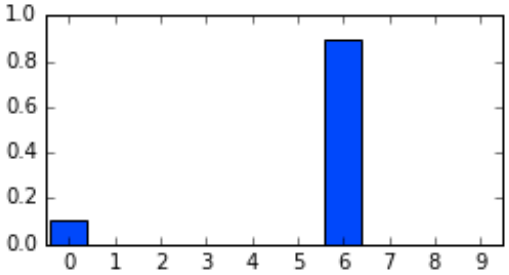
```
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*num_filters2])

num_units1 = 7*7*num_filters2
num_units2 = 1024

w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.constant(0.1, shape=[num_units2]))
hidden2 = tf.nn.relu(tf.matmul(h_pool2_flat, w2) + b2)

keep_prob = tf.placeholder(tf.float32)
hidden2_drop = tf.nn.dropout(hidden2, keep_prob)

w0 = tf.Variable(tf.zeros([num_units2, 10]))
b0 = tf.Variable(tf.zeros([10]))
p = tf.nn.softmax(tf.matmul(hidden2_drop, w0) + b0)
```

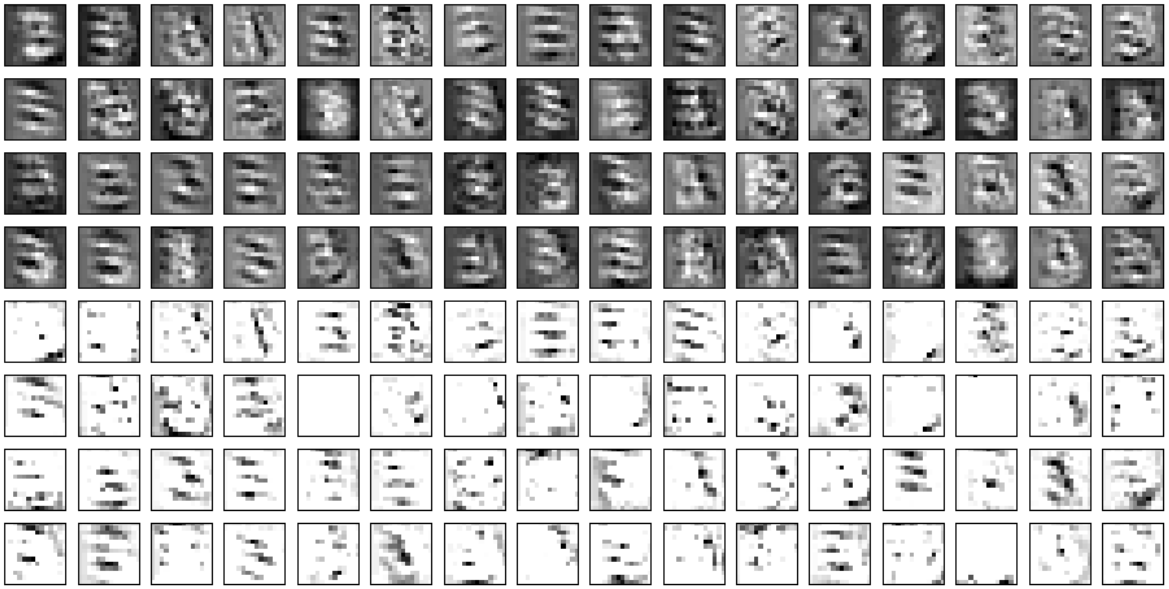
結合を残す割合を指定する変数（Placeholder）

（参考）手書き数字の認識アプリケーション

- TensorFlowで学習した結果は、ファイルに保存することができます。
  - 学習済みの結果を読み込んで、手書き数字の認識アプリケーションを作ることも可能です。







練習問題

- この練習問題を始める前に、インストラクターの指示に従って、Jupyterで稼働中のカーネルをすべて停止しておいてください。
- 単層CNNについて、次のような実験を行います。
  - ノートブック「MNIST dynamic filter classification.ipynb」を実行して、学習済みのパラメーターファイル「mdc_session-4000」を作成してください。（トレーニングが完了するまで、15分程度かかります。）
  - ノートブック「MNIST dynamic filter result.ipynb」を実行して、学習済みのモデルによる分類結果を確認してください。
  - 各ノートブックの[MDC-03]と[MDR-03]における、下記の部分で畳み込みフィルターの数が指定されています。フィルター数を変更すると学習結果がどのように変わるか確認してください。

```
num_filters = 16
```

練習問題

- ２層CNNを用いた手書き文字の認識処理を行います。
  - 2層CNNの学習処理には時間がかかるため、ここでは、事前に学習済みのパラメーターファイルをダウンロードして使用します。ノートブック「Handwriting recognizer.ipynb」を開いた後、先頭に空のセルを追加して、次のコマンドを実行します。

```
!curl -OL http://<インストラクターから指示があります>/cnn_session-20000
```

  - その後のセルを順番に実行していき、[HWR-07] まで実行したら、画面上で手書き文字を入力行った後、さらにその後のセルの実行を続けます。

メモとしてお使いください

メモとしてお使いください