超入門

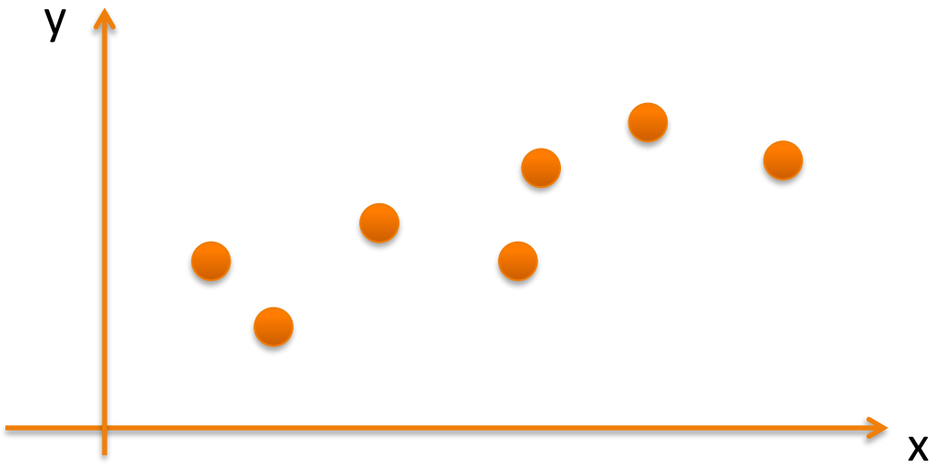
AI-CNN（Convolutional Neural Network）畳み込みニューラルネットワーク徹底理解

- AI及びCNNの仕組みを徹底理解
- Colaboratoryでマウス手書き文字認識

目　次

1－1．機械学習　例1）線形単回帰

線形単回帰は、x、yの２変数のデータを一次関数（y=ax+b）で表せるもの



直線で表せるデータがあり、関係式を求めていく。

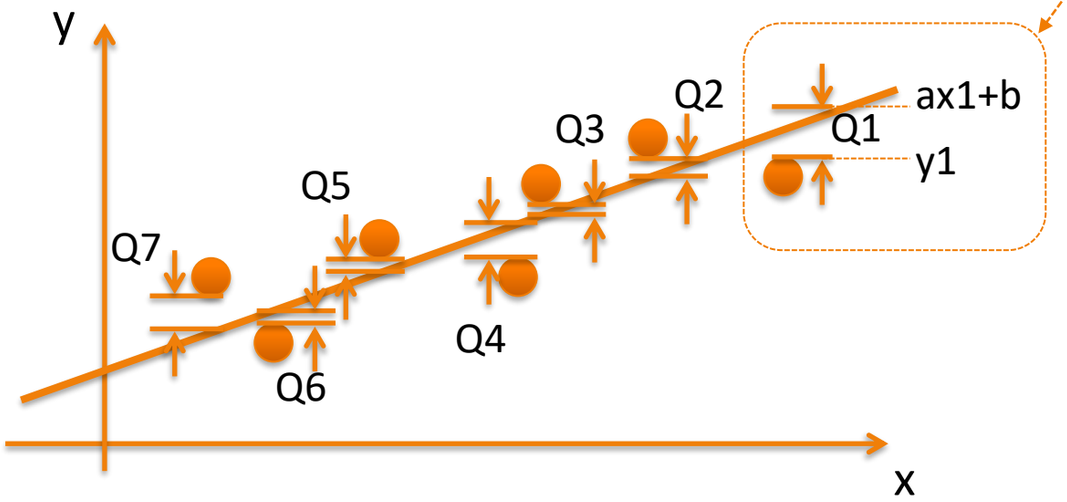
誤差は「Q1 = y1 – (ax1 + b)」

上下に誤差があるので、2乗したものを合計する

誤差の総和　：　Qt = Q1 + Q2 + Q3 + ...

$$= \{ y1 - (ax1 + b)\}^2 + \{ y2 - (ax2 + b)\}^2 + \{ y3 - (ax3 + b)\}^2 + \ldots$$

誤差の総和が最小になるのが求める関係式



誤差の総和（損失関数）Qtが最小になるようにa,bのパラメータが決定されます。

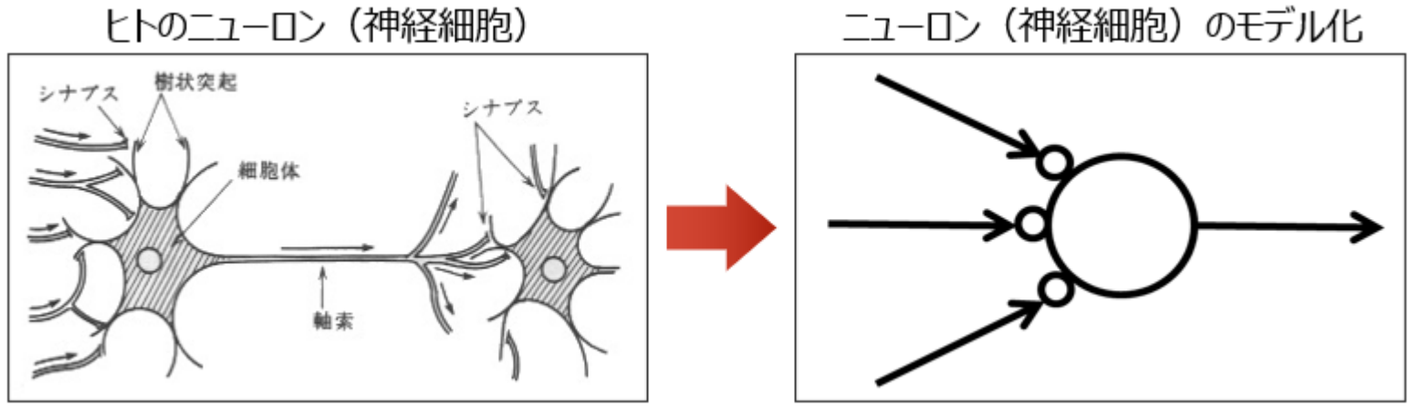
⇨ 最小２乗法

①モデルを決める（今回は一次関数）

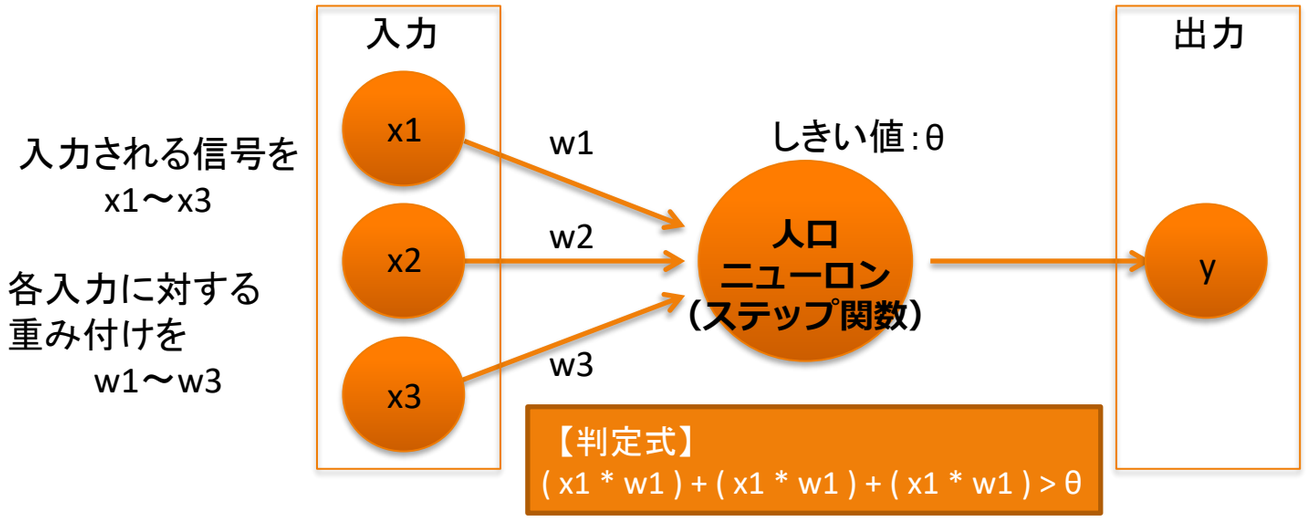
②学習データによる誤差の総和（損失関数）が最小（or最大）になるパラメータを決定

1－2. 機械学習　例3）パーセプトロン（人口ニューロン）

人の脳神経細胞をモデル化した、人口ニューロン（パーセプトロン）



参考：https://aitokuconsult.hatenablog.com/entry/neuralnetwork



例）各入力が1，2，3、重みは全て１で、しきい値が５の場合

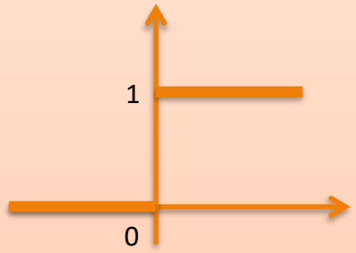
(１ * １) + (２ * １) + (３ * １) > ５

６ ＞ ５

【ステップ関数】
正しいので、出力は「１」

出力には多くの関数がある

①活性化関数：ステップ関数

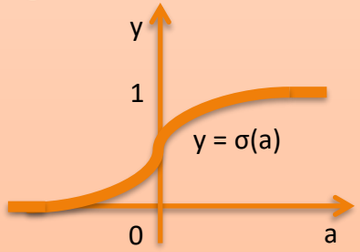


出力は0か1の2値

(x1 * w1) + (x1 * w1) + (x1 * w1)
> θ

真なら1、偽なら0

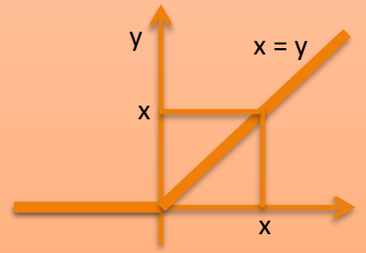
②活性化関数：シグモイド関数



出力は0から1の間で任意値

出力　**y = σ(a) で表します。**
σはシグモイド関数で**σ(x) = 1 / (1 + Exp^{-x})**
aは「入力の線形和」と呼ばれ
(x1 * w1) + (x2 * w2) + (x3 * w3) – θ
となります。
出力層でよく使われる。

③活性化関数：ランプ関数（ReLU：Rectified Linear Unit）

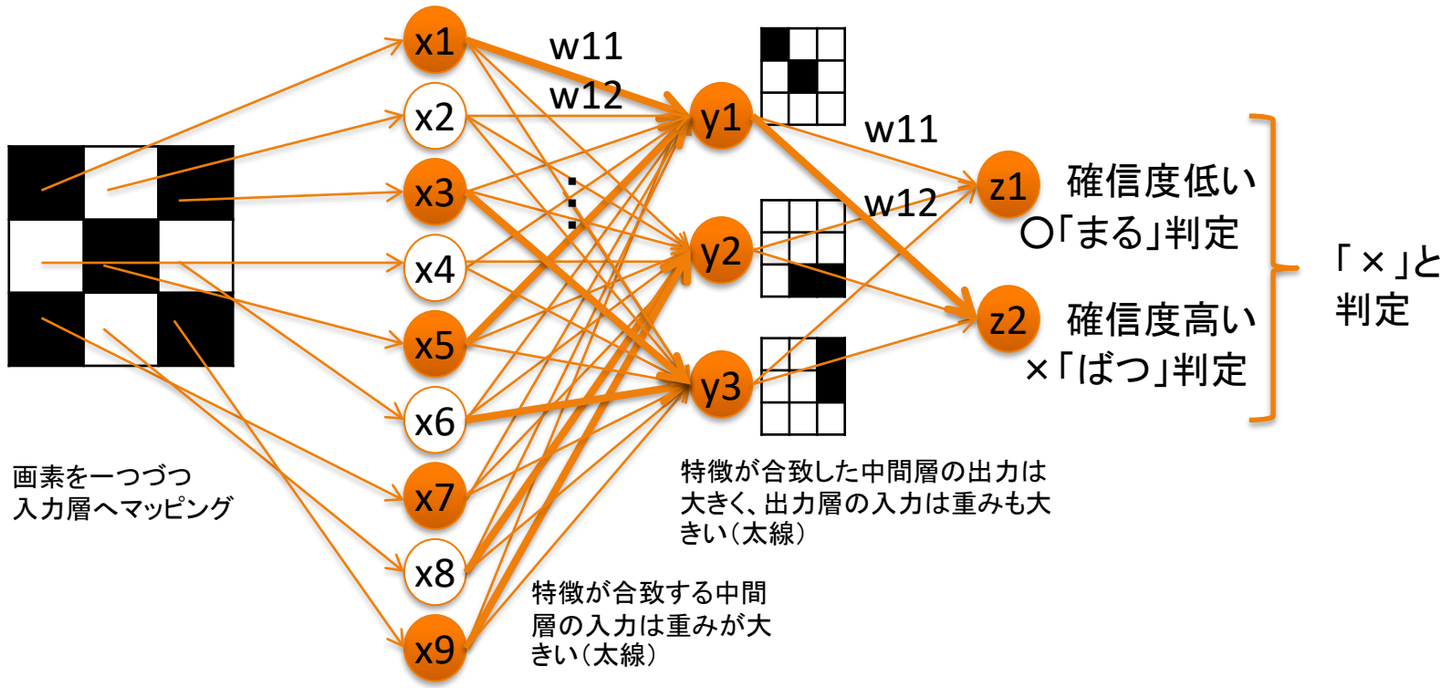


出力は0より任意の値

単純で計算しやすい。
中間層でよく使われる。

1－3. 機械学習　例4）ニューラルネットワーク

ニューラルネットワークを理解するため、中間層(隠れ層)が1層モデルで学習
（3*3画像の〇「まる」と×「ばつ」を判定するニューラルネットワーク）



画像

入力層

入力情報をそのまま全ての隠れ層へ

中間（隠れ層）

各隠れ層には検出するパターンがあり特徴抽出を行う

出力層

〇×判定を行う

各ニューロンでは、入力に重みをかけて、しきい値で判定した値が出力される

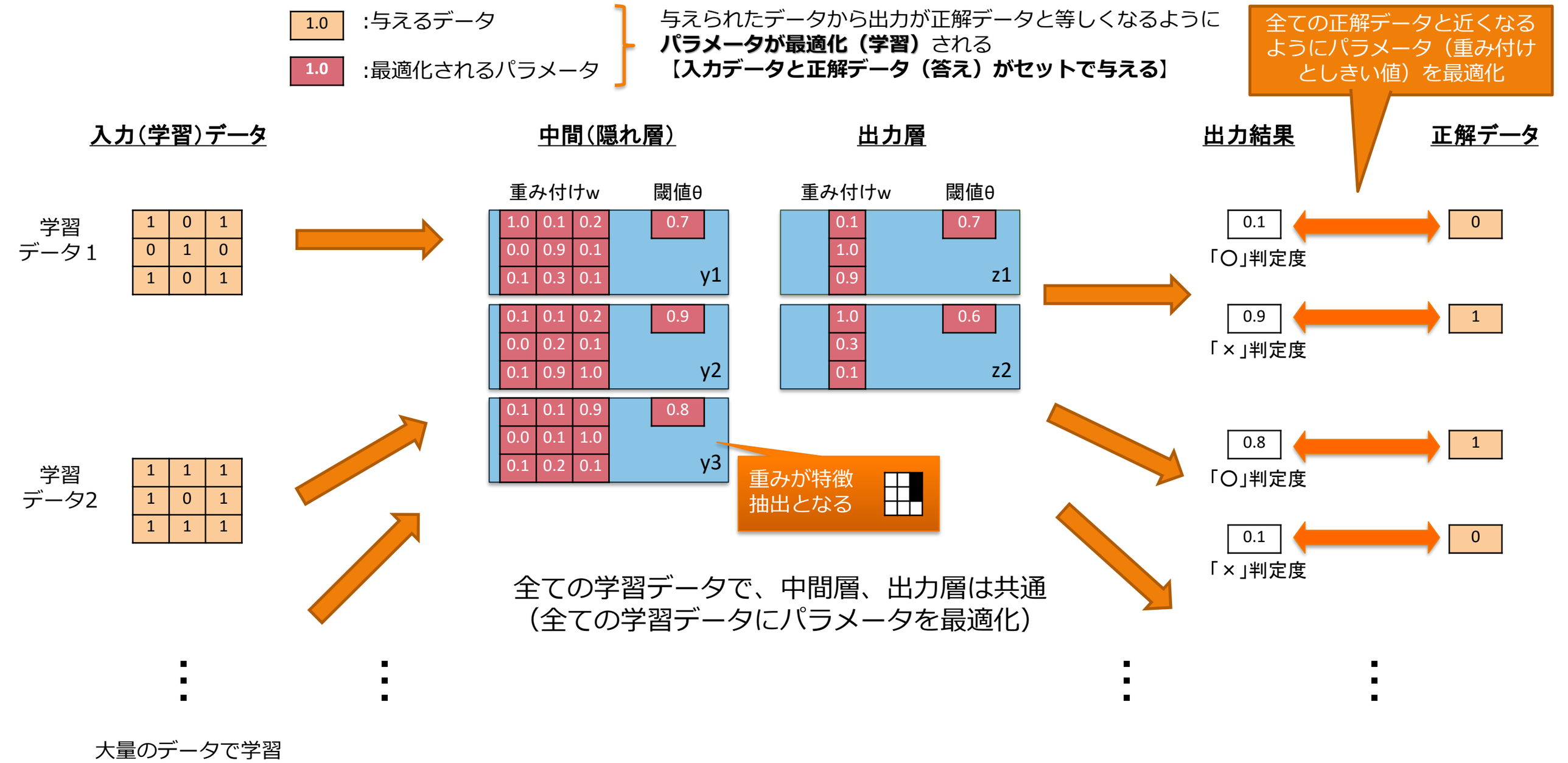
【中間層の計算式】

a1 = (x1 * w11) + (x2 * w12) ・・・ + (x9 * w19) – θ1
a2 = (x1 * w21) + (x2 * w22) ・・・ + (x9 * w29) – θ2
a3 = (x1 * w31) + (x2 * w32) ・・・ + (x9 * w39) – θ3
y1 = σ(a1), y2 = σ(a2), y3 = σ(a3)
σはシグモイド関数

【出力層の計算式】

z1 = (y1 * w11) + (y2 * w12) + (y3 * w13) – θ1
z2 = (y1 * w11) + (y2 * w22) + (y3 * w23) – θ2

1－4. 機械学習　例4）ニューラルネットワーク



２．GoogleColabratoryとMNISTデータ

GoogleColabratory

GoogleがAIの研究や学習用に提供するPython開発環境です。
インストールなど準備不要で、Webブラウザですぐに利用できます。
また、無料で利用できますが、１回１２時間など制限があります。

MNISTデータ

MNISTは「Mixed National Institute of Standards and Technology database」の略です。
インターネットに公開されているデータセットでAIを学習するために無料で提供されています。

手書き数字画像60,000枚と、テスト画像10,000枚を集めた、画像データセットとなります。

3－1．AIプログラム

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

model = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(10, activation="softmax"),
    ]
)

model.summary()

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

model.fit(x_train, y_train, batch_size=128, epochs=15, validation_split=0.1)

score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

①利用するライブラリを定義

②データを準備
- 利用する学習、検証用データ（MNIST）を読み込み（x:画像データ、y:正解ラベル、train:6万枚、test：1万枚）
- 0から255のグレースケールを0から1の値に変換
- 次元を1つ追加（AIで扱うデータ形式に合わせる）
- ラベルデータの整数値を2値クラスの行列に変換（例「整数 2」を「0,0,1,0,0,0,0,0,0,0」と表現

③AIモデルを定義

④AIモデルを表示

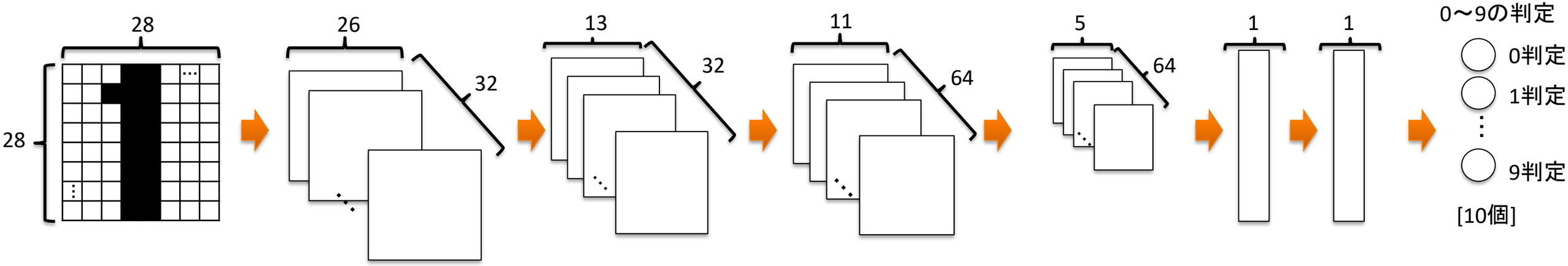
⑤パラメータ最適化（AI学習）の設定

⑥パラメータ最適化（AI学習）の実行

⑦学習後のAI性能を検証データを用いて評価

3－2. 畳み込みニューラルネットワーク

ディープラーニング　　：　中間層が2層以上のニューラルネットワーク
畳み込みニューラルネットワーク　：　中間層で小分けにして学習する手法



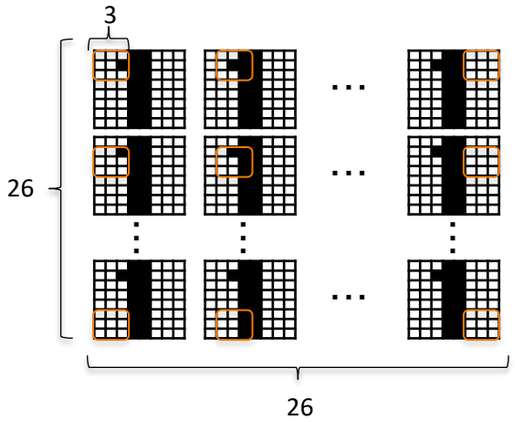
入力層

28*28の画素数なので
28*28=784ニューロン

畳み込み層1

3*3枠で26*26箇所の小分けにして32フィルタで特徴抽出

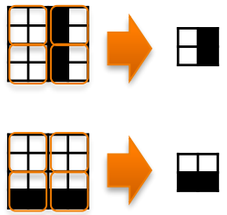
26*26箇所の小分けイメージ



MAXプーリング層

4*4を2*2の枠で縮小
ズレに強い特徴抽出

Maxプーリングイメージ

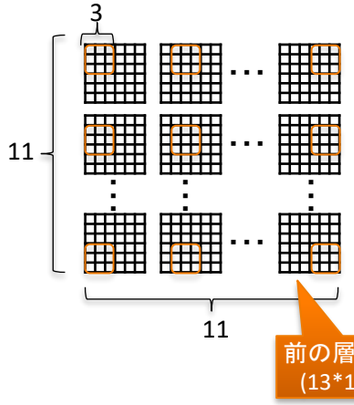


各枠の最大値のみ残す

畳み込み層2

3*3枠で11*11箇所の小分けにして64フィルタを特徴抽出

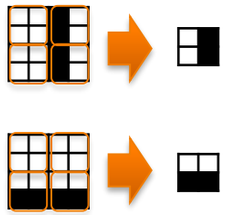
11*11箇所の小分けイメージ



MAXプーリング層

4*4を2*2の枠で縮小
ズレに強い特徴抽出

Maxプーリングイメージ



各枠の最大値のみ残す

Flat層

1列に並べる（1次元配列へ）

Drop Out層

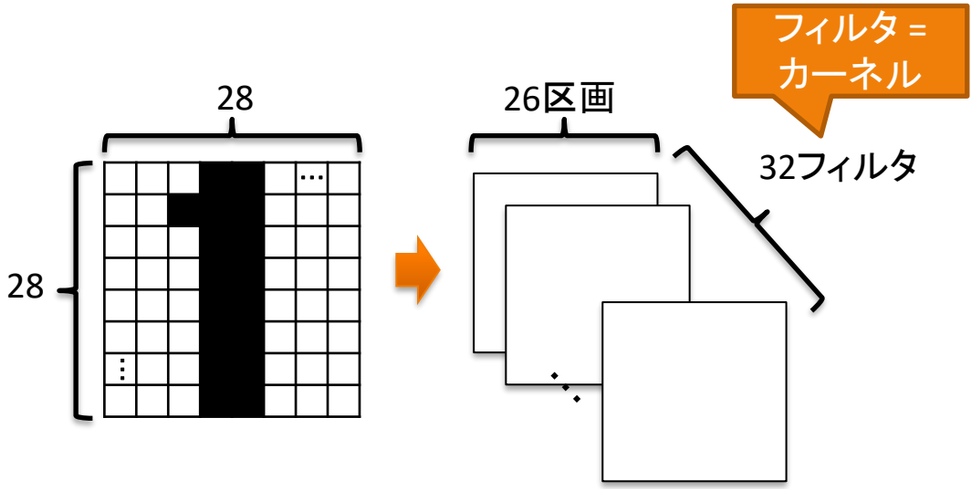
一定割合で削除

Dense層

0から9の各要素を判定し割合を出力

3－3. 畳み込み層

```
layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
```



入力層

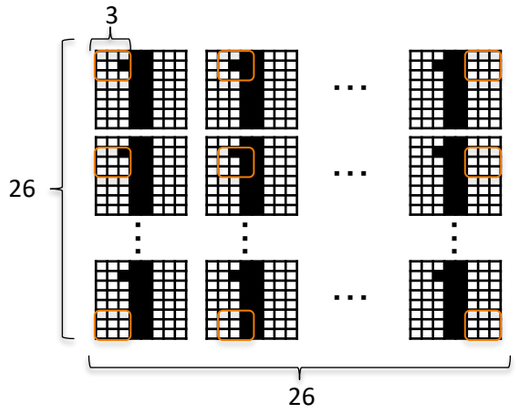
28*28の画素数なので
28*28=784ニューロン

畳み込み層1

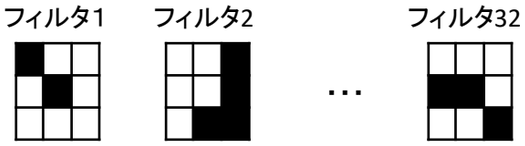
3*3枠で26*26箇所の小分け
にして32フィルタで特徴抽出

26*26箇所の小分けイメージ

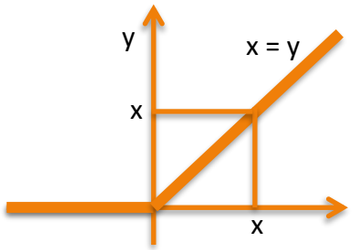
同じ1つのフィルタ(3*3)で28*28画像
全体を区分けして特徴抽出する
（26*26区画に分けて特徴抽出される）



32フィルタは各々個別に特徴抽出する



[ニューロン出力]
活性化関数：ランプ関数
（ReLU： Rectified Linear Unit）



出力は0より任意の値
入力に応じて出力の大きさが決定

フィルタと個別に区分けして特徴抽出することで、最適化するパラメータを大幅に削減できる

効率よく
特徴抽出できる

3－4．MAXプーリング層

```
layers.MaxPooling2D(pool_size=(2, 2)),
```

MaxPooling2D 処理 (pool_size = 2)

2*2枠の中で最大値を抽出する

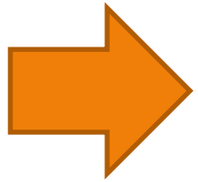
1	5	9	5
0	3	8	3
2	1	9	6
0	1	0	2

5	

1	5	9	5
0	3	8	3
2	1	9	6
0	1	0	2

	9

全ての区画で実施すると



各々の区画の
最大値を取得

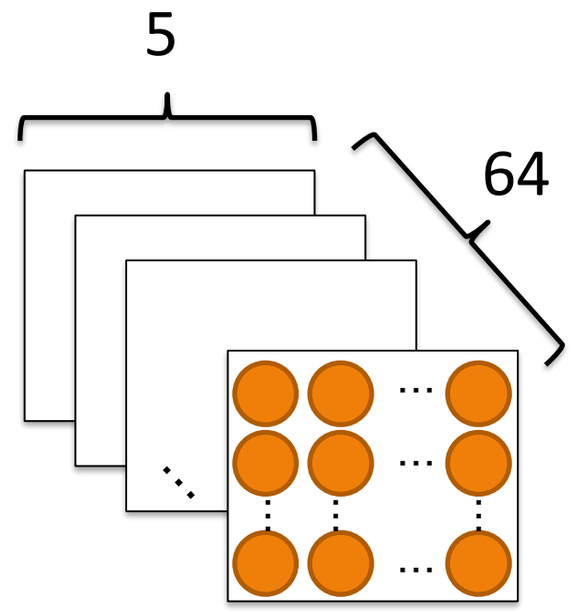
1	5	9	5
0	3	8	3
2	1	9	6
0	1	0	2

5	9
2	9

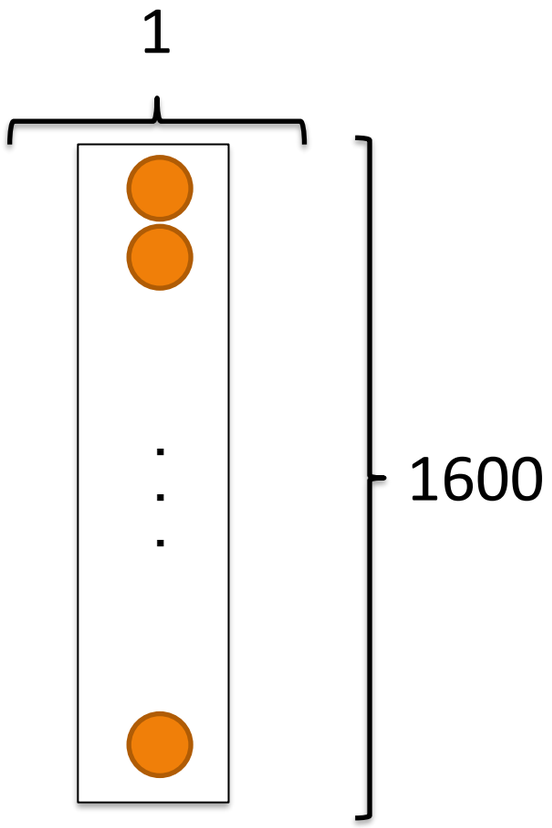
大まかに特徴をとらえ、
ズレなどに強い特徴抽出ができる

3－5. Flat層

```
layers.Flatten(),
```



5*5*64(1600個)に並んでいるニューロン

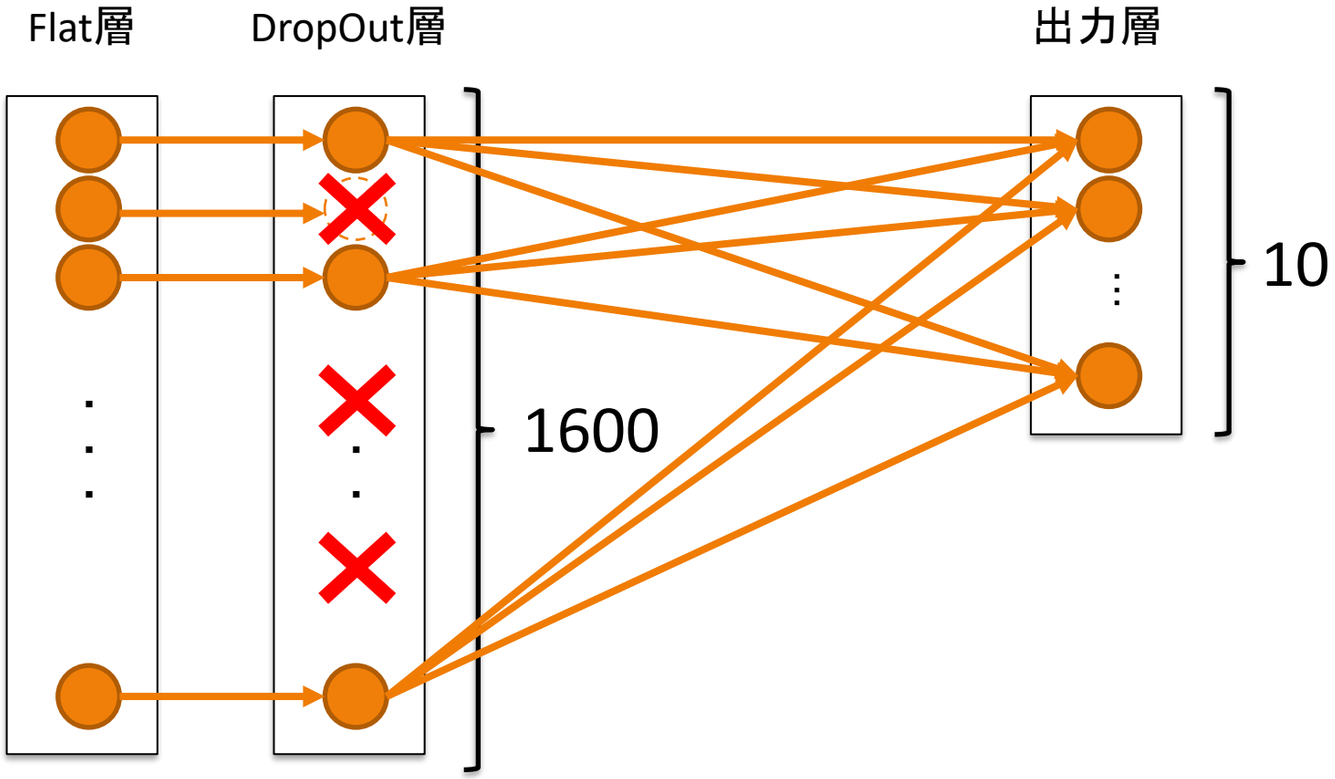


1列に並びなおす
（1次元配列に並びなおす）

次の層に向けてデータを整える役割

3－6．DropOut層

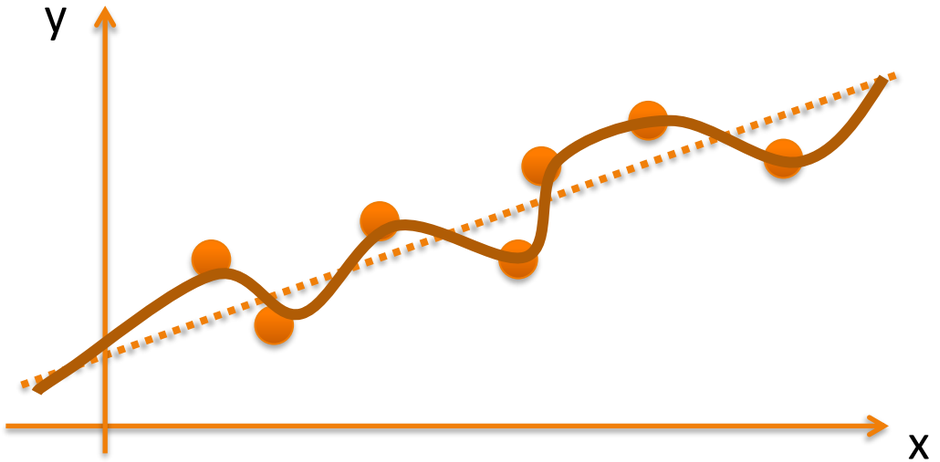
```
layers.Dropout(0.5),
```



ニューロンの出力を一定の割合で
ランダムに無効化します。
今回、除外化率は0．5なので50%
の割合で削除される

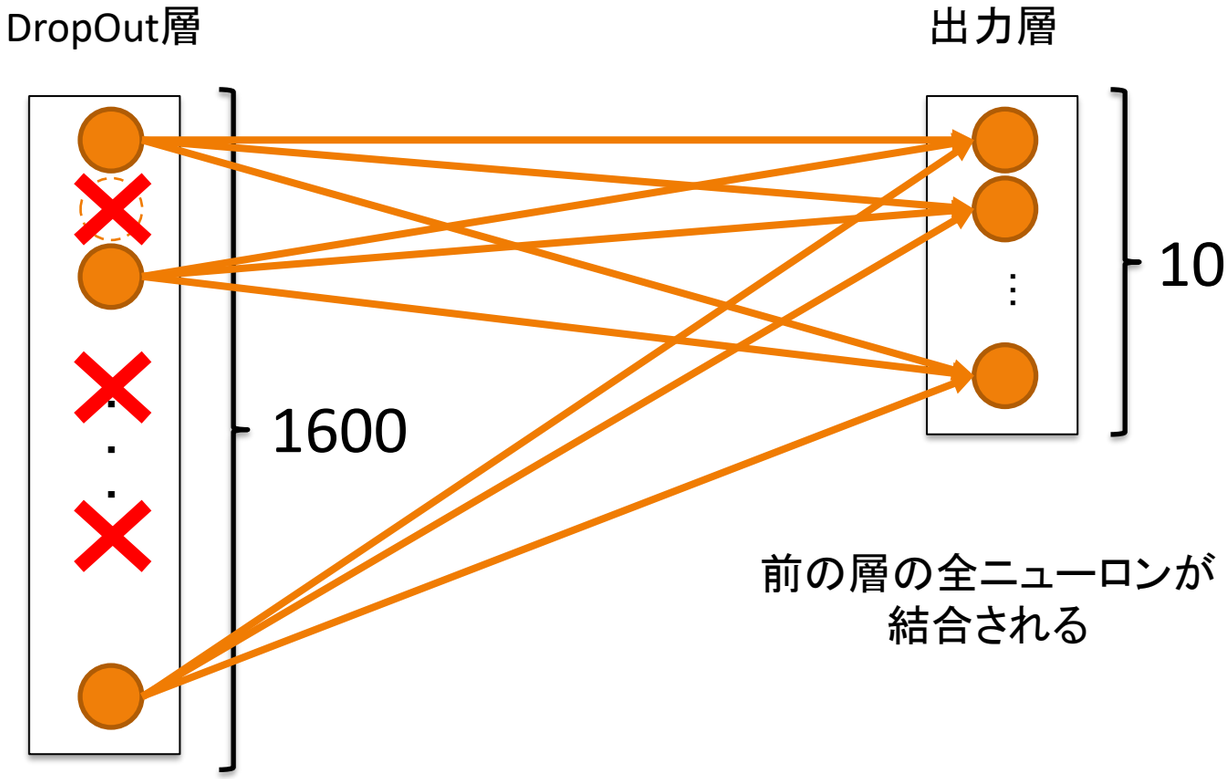
過学習（overfitting）防止に
効果的

過学習とは求めたい関係式（点線）に対して
与えたデータに偏りすぎて関係になってしまうこと



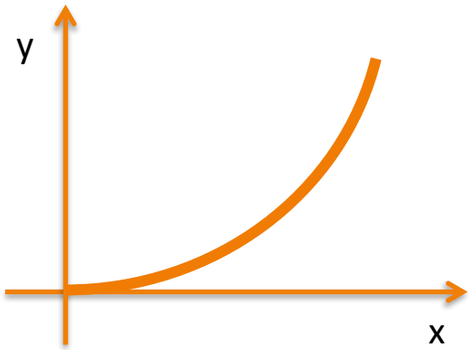
3－7. Dense層（全結合層）

```
layers.Dense(10, activation="softmax"),
```



前の層の全ニューロンが
結合される

[ニューロン出力]
活性化関数：ソフトマックス関数



$$y1 = \frac{exp(x1)}{exp(x1) + exp(x2) \ldots exp(xn)}$$

$$y2 = \frac{exp(x2)}{exp(x1) + exp(x2) \ldots exp(xn)}$$

$$\ldots$$

$$yn = \frac{exp(xn)}{exp(x1) + exp(x2) \ldots exp(xn)}$$

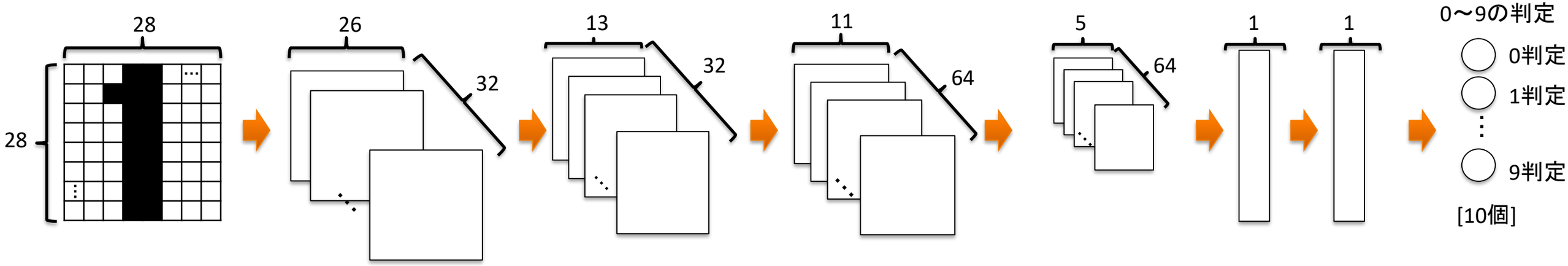
入力が大きくなるに連れて出力が大きく変化していく関数
全体から各々の割合を示す関係式となっているので、出力値の各要素（分類ラベルの確率）合計が1（100%）になります。

要素数が2つの場合(分類ラベルが2つ)のソフトマックス関数はシグモイド関数と同じです。

0から9の数値判定が行われる

3－2. 畳み込みニューラルネットワーク

ディープラーニング　　：　中間層が2層以上のニューラルネットワーク
畳み込みニューラルネットワーク　：　中間層を小分けにして学習する手法



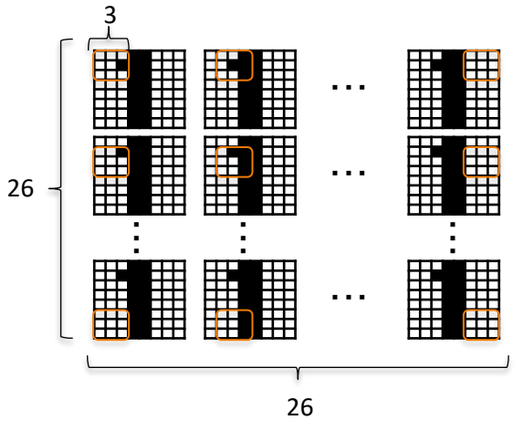
入力層

28*28の画素数なので
28*28=784ニューロン

畳み込み層1

3*3枠で26*26箇所の小分けにして32フィルタで特徴抽出

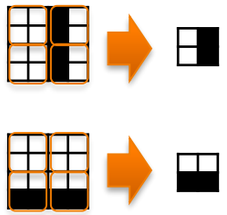
26*26箇所の小分けイメージ



MAXプーリング層

4*4を2*2の枠で縮小
ズレに強い特徴抽出

Maxプーリングイメージ

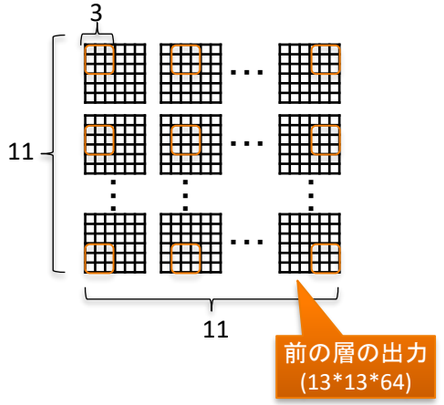


各枠の最大値のみ残す

畳み込み層2

3*3枠で11*11箇所の小分けにして64フィルタを特徴抽出

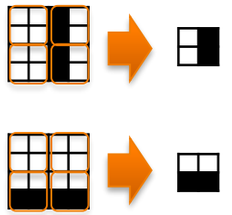
11*11箇所の小分けイメージ



MAXプーリング層

4*4を2*2の枠で縮小
ズレに強い特徴抽出

Maxプーリングイメージ



各枠の最大値のみ残す

Flat層

1列に並べる（1次元配列へ）

Drop Out層

一定割合で削除

Dense層

0から9の各要素を判定し割合を出力

３－８．AI学習（パラメータ[重みとしきい値]の最適化）

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

`loss="categorical_crossentropy"`

損失（目的）関数の指定。
（損失関数は、モデルの学習中に最小化しようとする指標）
多クラス分類タスクにおいて一般的に使用される損失関数

`optimizer="adam"`

adam(Adaptive Moment Estimation)は、最適化アルゴリズムの1つで、勾配降下法の改良版として広く使用されています。adamは、学習率の調整やモーメンタムの利用など、自動的に学習の適応性を向上させる機能があります。

`metrics=["accuracy"]`

設定される評価指標です。評価指標は、学習中や学習後にモデルの性能を評価するために使用されます。
学習中に訓練データと検証データの正解率が計算され、表示されます。

参考．方程式の解き方について

式を変形して求める

$$3x + 2y = 13 \quad \cdots (1)$$
$$5x + 3y = 21 \quad \cdots (2)$$

(1)を変形すると

$$3x = 13 - 2y$$
$$x = (13 - 2y) \div 3$$

(2)に代入すると

$$(5 * (13 - 2y) \div 3) + 3y = 21$$
$$5 * (13 - 2y) + 9y = 63$$
$$65 - 10y + 9y = 63$$
$$y = 2$$

(1)に代入すると

$$3x + 2 * 2 = 13$$
$$x = 3$$

正確に求められるが、関係式が変形して求められる場合に限られる

値を代入して求める

$$3x + 2y = 13 \quad \cdots (1)$$
$$5x + 3y = 21 \quad \cdots (2)$$

仮にx＝2，y＝2を代入すると

$$3 * 2 + 2 * 2 = 10 < 13$$
$$5 * 2 + 3 * 2 = 16 < 21$$

値が小さいので、より大きな値を入力
x＝3，y＝2を代入すると

$$3 * 3 + 2 * 2 = 13 = 13$$
$$5 * 3 + 3 * 2 = 21 = 21$$

正解が合致したので正解は

x＝3，y＝2

どのような複雑な関係式でも正解を求めていくことができる。

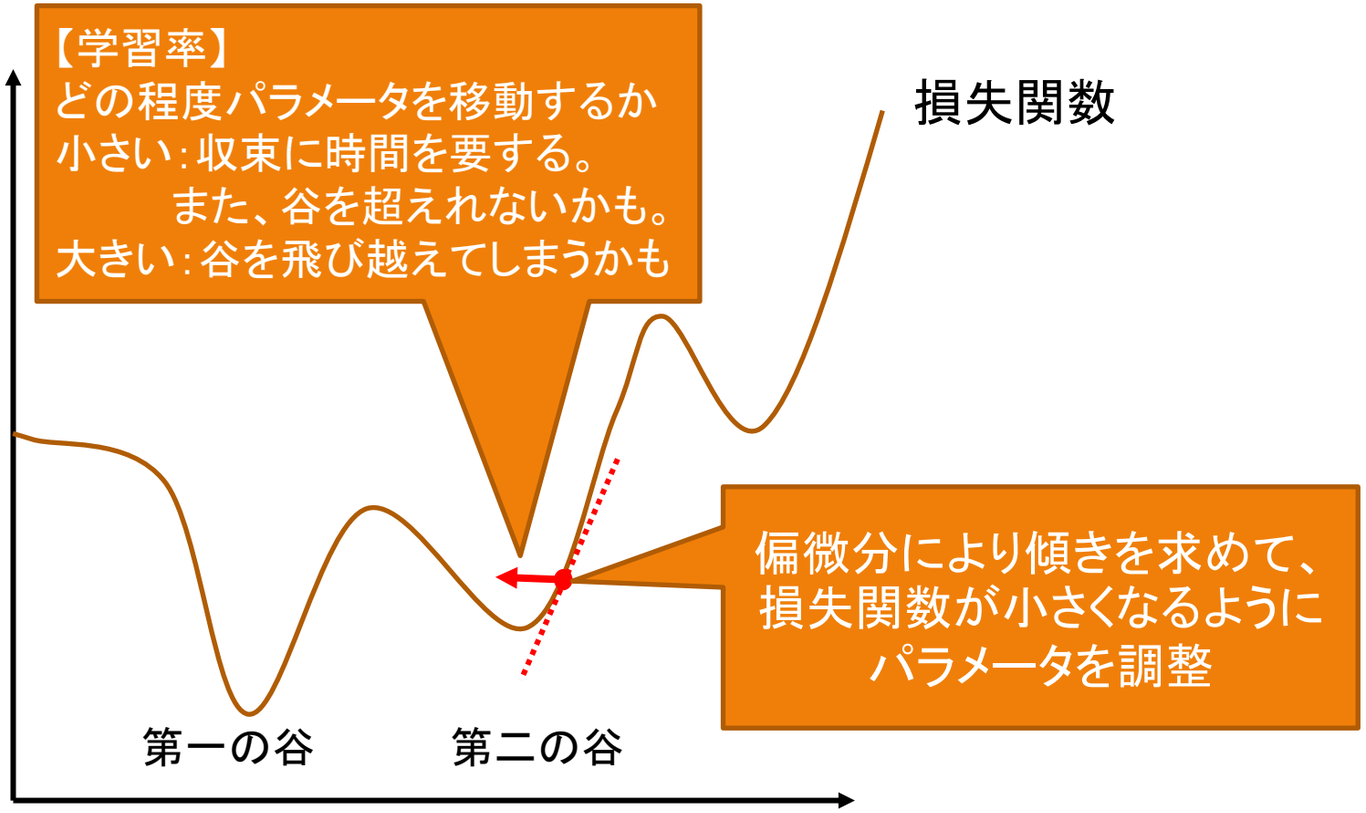
3－9. 勾配降下法

勾配降下法（Gradient Descent）は、最適化アルゴリズムの一種で、損失関数を最小化するパラメータを見つけるために使用されます。

勾配降下法の基本的な考え方は、損失関数の勾配（すなわち、各パラメータに対する損失関数の微分）を計算し、勾配が指す方向にパラメータを更新することで、損失関数の値を減らすことです。
具体的には、以下の手順で行われます。

1. パラメータの初期値を設定します。
2. 損失関数の勾配を計算します。これは、各パラメータに対する損失関数の偏微分です。
3. 勾配に対して学習率（通常は小さい正の値）をかけた値を使って、パラメータを更新します。これにより、損失関数の値が減少する方向にパラメータが更新されます。
4. 収束するまで、または指定されたエポック数に達するまで、2と3の手順を繰り返します。

勾配降下法のイメージ



3－10．AI学習（パラメータ[重みとしきい値]の最適化）

```
model.fit(x_train, y_train, batch_size=128, epochs=15, validation_split=0.1)
```

validation_split=0.1

学習後の検証用データに10%を設定しています。従って、学習用データは90%になります。
MNISTデータは全体で60000枚ありますので、学習には54000枚が利用されます。

epochs=15

エポックは、データを一度全て学習することになります。今回は、54000枚の画像データを全て学習したら1エポックになります。
15を設定していますので、54000枚の画像データを15回学習します。

batch_size=128

バッチサイズは一度に学習するデータ枚数です。バッチとはまとめて処理するということになります。
128枚をまとめて処理しますので、54000枚は422回（=54000/128）のバッチ処理が必要です。このため、1エポックは、422回のバッチ処理となり、バッチ単位に処理状況の確認ができます。

3－11．AIの評価（AI学習終了後）

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

`model.evaluate(x_test, y_test, verbose=0)`

：　MNISTデータのテストデータ1万枚で、AIの性能を評価します。
（verboseは学習状況の情報表示パラメータ。0は特に表示しない。進捗バーなどを表示する場合は1を設定）

Loss　：　テストデータでの損失関数の値。
損失関数が小さくなるようにパラメータを最適化（AIが学習）してきました。

Accuracy　：　テストデータでの正解率。正確に判定できているかの割合。