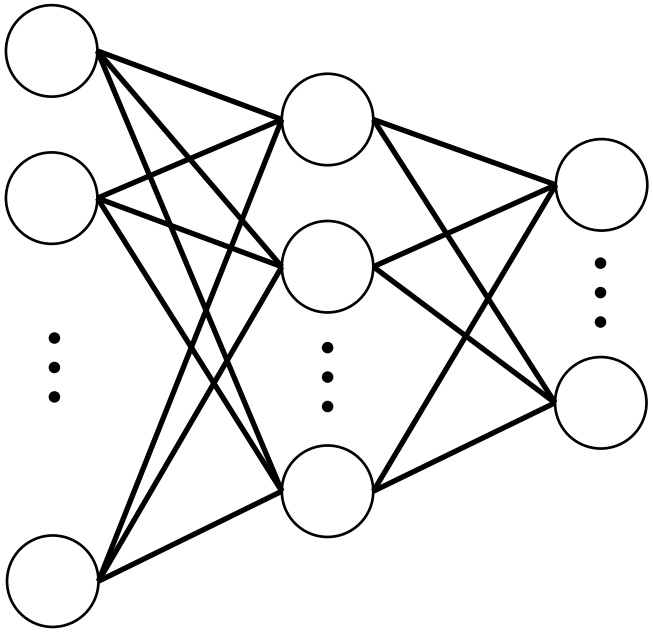
情報システム実験実習II（後期）
機械学習によるデータ解析
3回目

情報システムコース 内田雅人

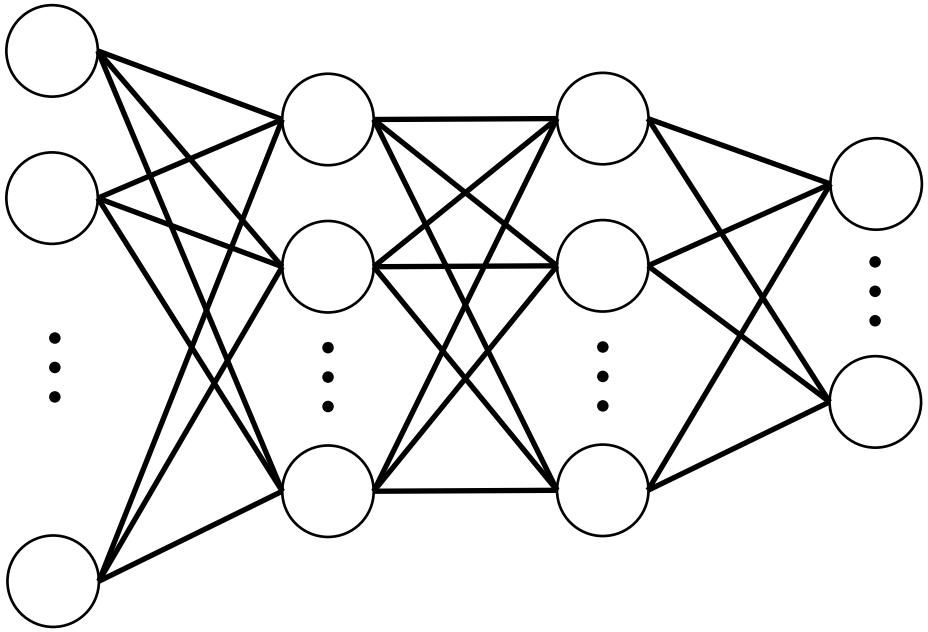
ニューラルネットワーク
多層レイヤーパーセプトロン

多層パーセプトロン（MLP）

- 層を複数持つネットワーク
  - ◆Multi Layer Perceptron → MLPと略す
  - ◆層をたくさん用意して学習 → 深層学習
- 非線形問題に対応→様々な応用が可能



入力層　　中間層（隠れ層）　　出力層

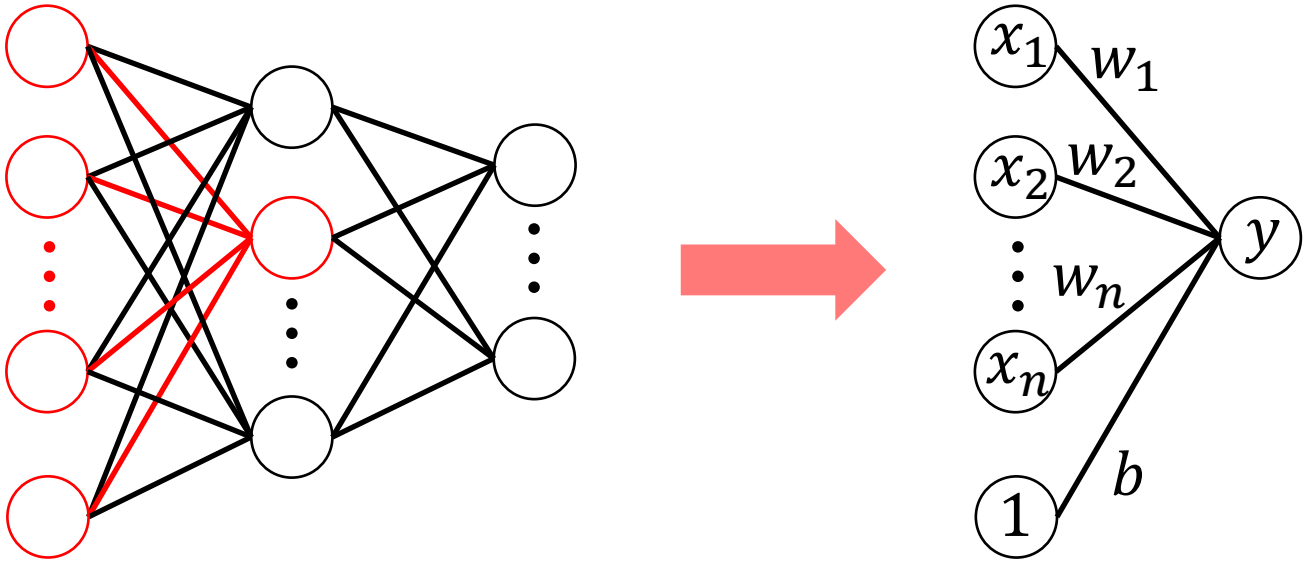


中間層を複数持つ場合

ネットワークの一部を取り出して考える

- 入力を $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^T$，バイアスを b
  重みを $\boldsymbol{w} = [w_1, w_2, \cdots, w_n]$とすると，
  あるニューロンの出力yは

$$y = \boldsymbol{x}^T \boldsymbol{w} + b = w_1 x_1 + w_2 x_2 + \cdots w_n x_n + b$$

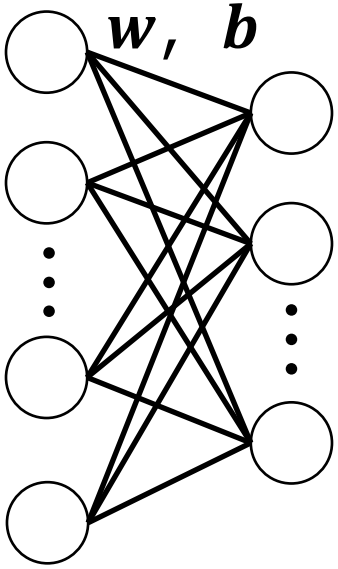


出力が複数の出力に拡張

- 出力$\boldsymbol{y} = [y_1, y_2, \cdots, y_m]^T$,
  バイアス$\boldsymbol{b} = [b_1, b_2, \cdots, b_m]^T$,

$$\text{重み}\boldsymbol{w} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix}$$ ：m行n列

$\rightarrow \boldsymbol{y} = \boldsymbol{x}^T\boldsymbol{w} + \boldsymbol{b}$とまとめる



$\boldsymbol{x}$：n個の入力
（説明変数）

$\boldsymbol{y}$：m個の出力
（目的変数）

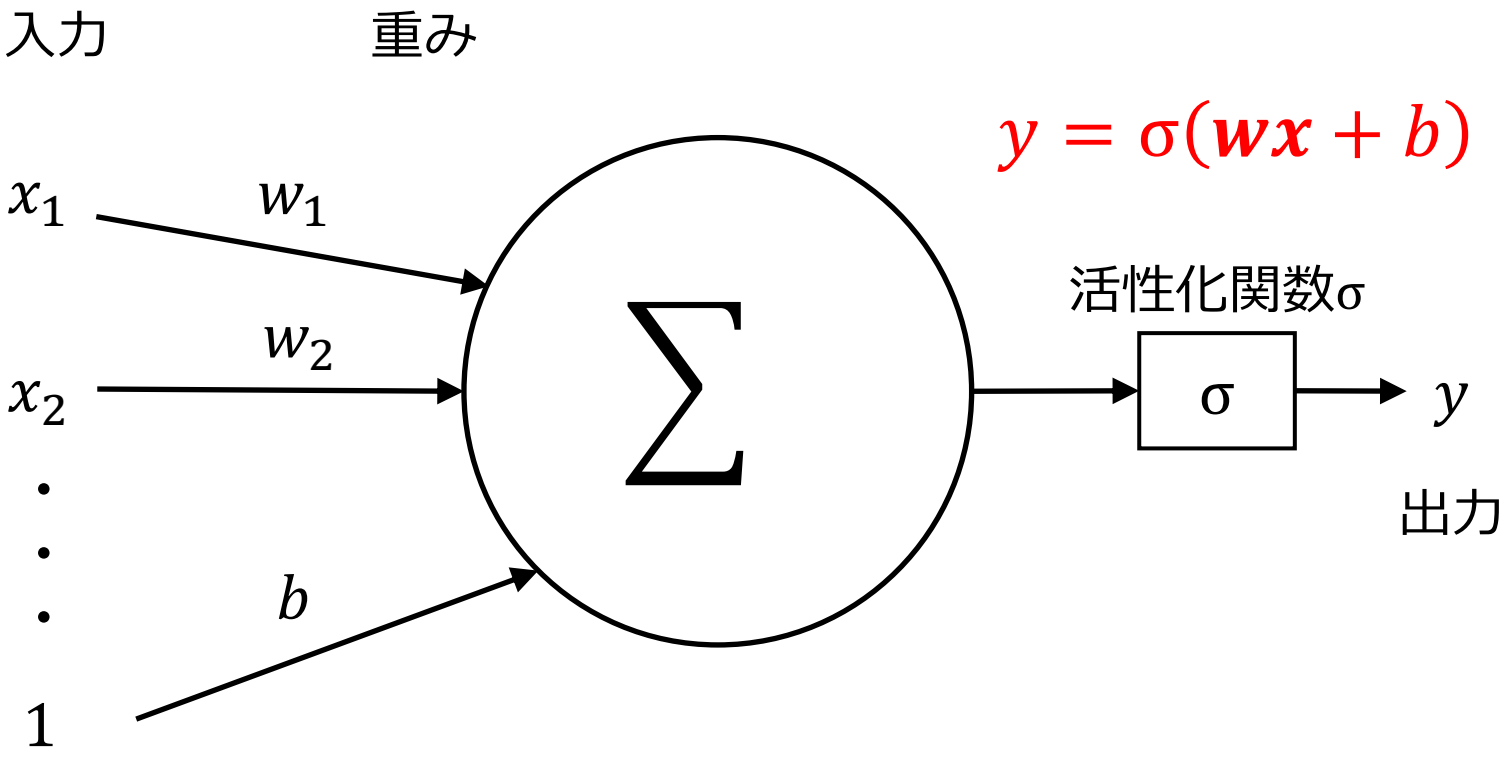
複数のデータに対応（バッチ対応）

- k個のデータがあるとする
  - ◆$\boldsymbol{x}$，$\boldsymbol{y}$の行方向に定義
  - ◆入力の次元数は(k, n)：(データ数, 入力数)
  - ◆重みの次元数は(n, m)：(入力数, 出力数)
  - ◆出力の次元数は(k, m)：(データ数, 出力数)
- 内積計算では

$$\begin{bmatrix} y_{1,1}, y_{1,2}, \cdots y_{1,m} \\ y_{2,1}, y_{2,2}, \cdots y_{2,m} \\ \vdots \\ y_{k,1}, y_{k,2}, \cdots y_{k,m} \end{bmatrix} = \begin{bmatrix} x_{1,1}, x_{1,2}, \cdots x_{1,n} \\ x_{2,1}, x_{2,2}, \cdots x_{2,n} \\ \vdots \\ x_{k,1}, x_{k,2}, \cdots x_{k,n} \end{bmatrix} \begin{bmatrix} w_{1,1} & \cdots & w_{m,1} \\ w_{1,2} & \cdots & w_{m,2} \\ \vdots & \cdots & \vdots \\ w_{1,n} & \cdots & w_{m,n} \end{bmatrix}$$

非線形の活性化関数

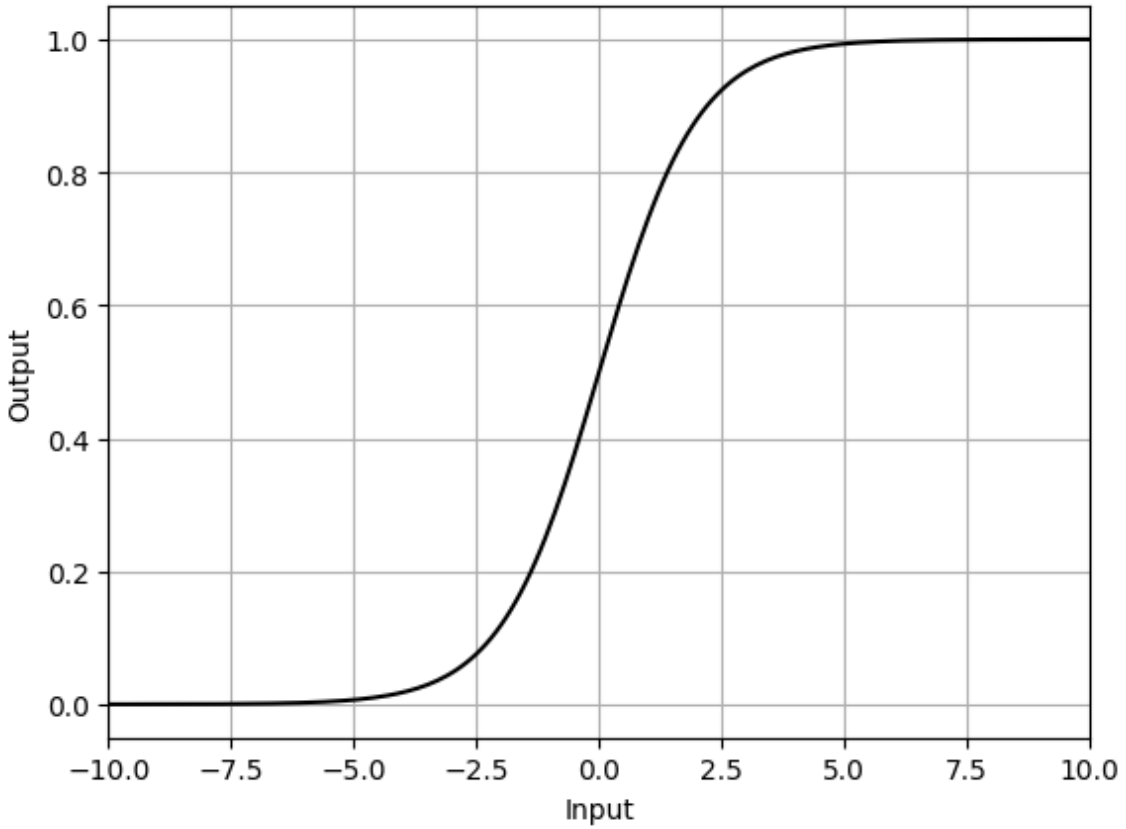
- 層間は活性化関数σをかける：$y = \sigma(\boldsymbol{w}\boldsymbol{x} + b)$
  - ◆σ = Sigmoid，Tanh，ReLuなどが用いられる
  - ◆活性化関数により効率よく学習可能



活性化間数の例：シグモイド関数

■ 関数$f(x) = \frac{1}{1-\exp(-x)}$で表される関数

◆ロジスティック変換という
0～1の値に変換（確率への変換にも利用される）



PyTorchを用いた
ニューラルネットワークの構築

インポートするもの

```
import torch   → PyTorch本体
import torch.nn as nn  → ネットワークの記述など
import torch.optim as optim  → 最適化器

# 以下は画像を使う場合のもの
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
```

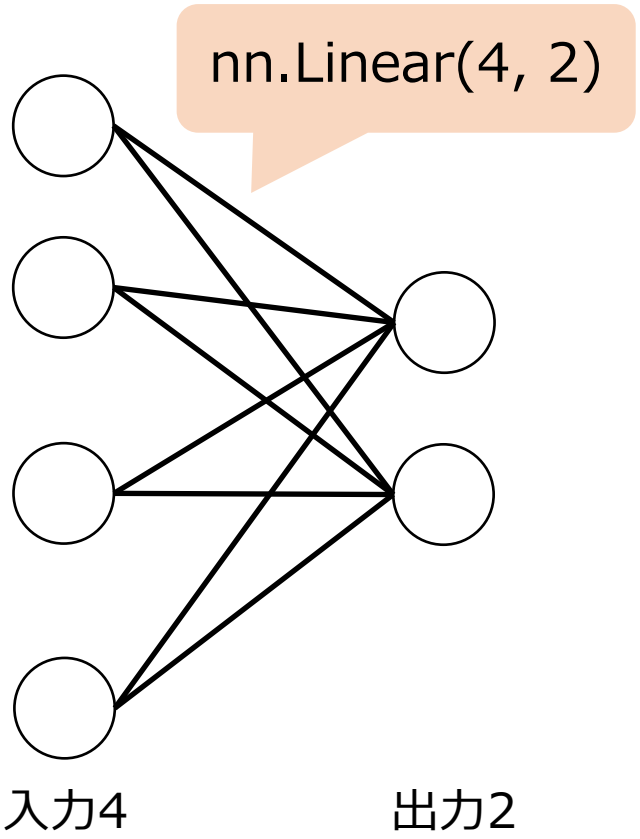
ニューラルネットワークのクラスの外観

```
class クラス名(nn.Module):
    def __init__(self, 引数は「 ,」区切りで記述):
        super(クラス名, self).__init__()
        # 各層の記述など

    def forward(self, 入力):
        # 順伝播の中身を書く
        return 出力
```

ネットワークの定義：例1

■ 入力～出力をfc1とする

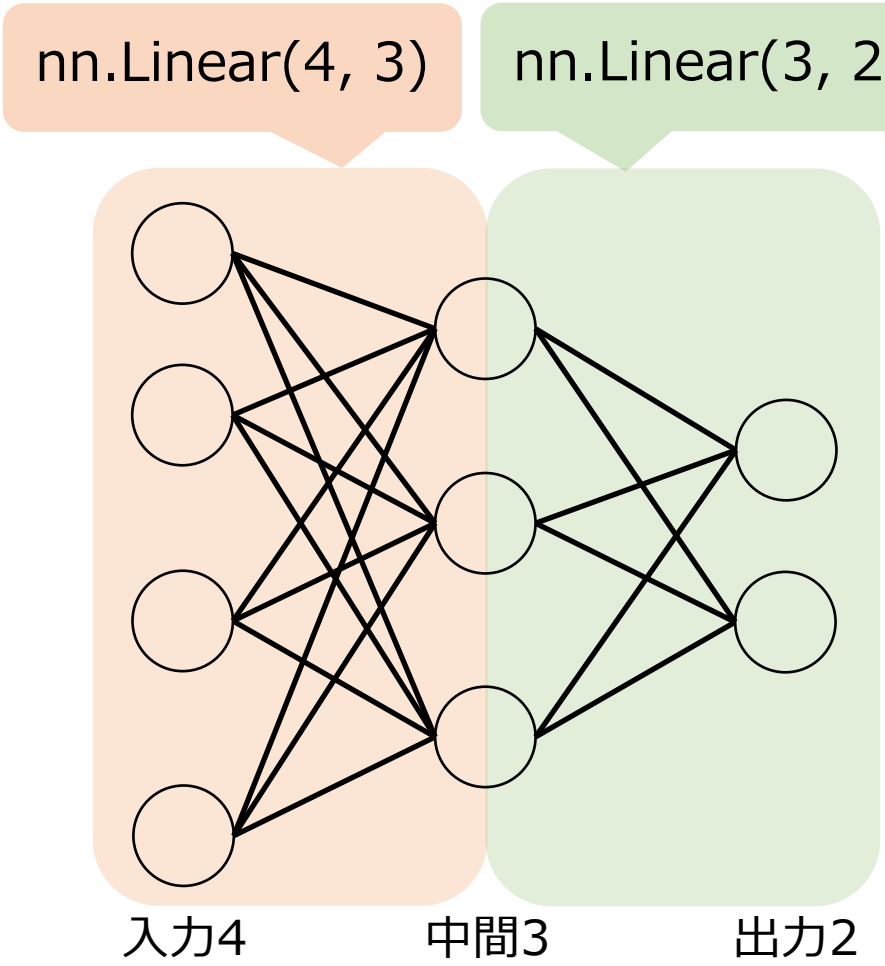



```
class クラス名(nn.Module):
    def __init__(self):
        super(クラス名, self).__init__()
        self.fc1 = nn.Linear(4, 2)

    def forward(self, x):
        x = self.fc1(x)
        return x
```

ネットワークの定義：例2

■ 入力～中間をfc1，中間～出力をfc2とする

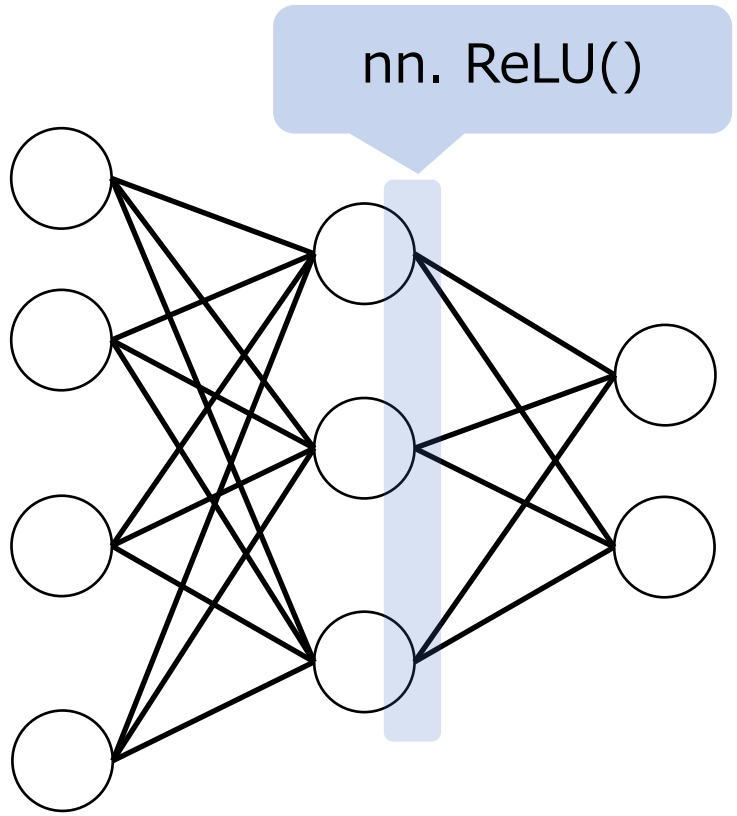



```
class クラス名(nn.Module):
    def __init__(self):
        super(クラス名, self).__init__()
        self.fc1 = nn.Linear(4, 3)
        self.fc2 = nn.Linear(3, 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

ネットワークの定義：例3

- 例2に加え，中間層の出力にReLU関数を適用した場合




```
class クラス名(nn.Module):
    def __init__(self):
        super(クラス名, self).__init__()
        self.fc1 = nn.Linear(4, 3)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(3, 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

インスタンスを生成と推論

- モデルを作る
  - class LinearModel(nn.Module):
    ⋮
- インスタンスの生成（クラス名：LinearModel）
  - model = LinearModel()
- モデルの推論
  - outputs = model(inputs)

誤差関数

- 誤差関数
  - ◆回帰：平均二乗誤差（MSE）
    → nn.MSELoss()
  - ◆分類：クロスエントロピー
    → nn.CrossEntropyLoss()
- 例
  - ◆criterion = nn.MSELoss() ← 誤差関数をMSEで宣言
  - ◆loss = criterion(outputs, targets)
    モデルの出力値　教師データ，事前に用意

最適化器（最適化アルゴリズム）

- 更新値を求める方法は最適化問題
  - ◆Adam：収束速い，失敗することがある
    - optim.Adam(model.parameters())
  - ◆SGD：簡単な計算，失敗しにくい
    - optim.SGD(model.parameters(), lr=学習率)

- 例
  - ◆optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    最適化器にSGDを使う

- 学習（以下は基本セットで使う）
  - ◆optimizer.zero_grad() ← 勾配をリセット
    loss.backward() ← 誤差逆伝播で勾配を計算
    optimizer.step() ← 重みを更新

基本的な学習のステップ

```
model = クラス名()
criterion = nn.MSELoss() # 回帰の場合はMSE
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(epochs):
    outputs = model(inputs) # 1. 推論
    loss = criterion(outputs, targets) # 2. 誤差を計算
    optimizer.zero_grad() # 3. 勾配をリセット
    loss.backward() # 4. 誤差逆伝播
    optimizer.step() # 5. 重みの更新
```

データセットに対する
学習回数をエポックという

データの型を変換(1)

- NumPyの配列をPyTorchの配列に変換
- 自前でデータセットを用意するとき要注意
- 例　　回帰の場合，float32でないとエラーになる

```
inputs = torch.tensor(x, dtype=torch.float32)
targets = torch.tensor(t, dtype=torch.float32)
                    ⋮
outputs = model(inputs)
loss = loss = criterion(outputs, targets)
```

演習(5)：PyTorchによる実装

- PyTorchでニューラルネットワークを実装し学習を実行
  - ◆データセットはdata/advertising.csvを使う
  - ◆説明変数：TV広告費、ラジオ広告費、新聞広告費
  - ◆目的変数：Sales（売上）
  - ◆誤差関数：平均二乗誤差
  - ◆最適化器：SGD
  - ◆入力3，出力1の全結合層を実装

- プログラムの実行（/scripts内で実行）
  $ uv run nn_prediction.py

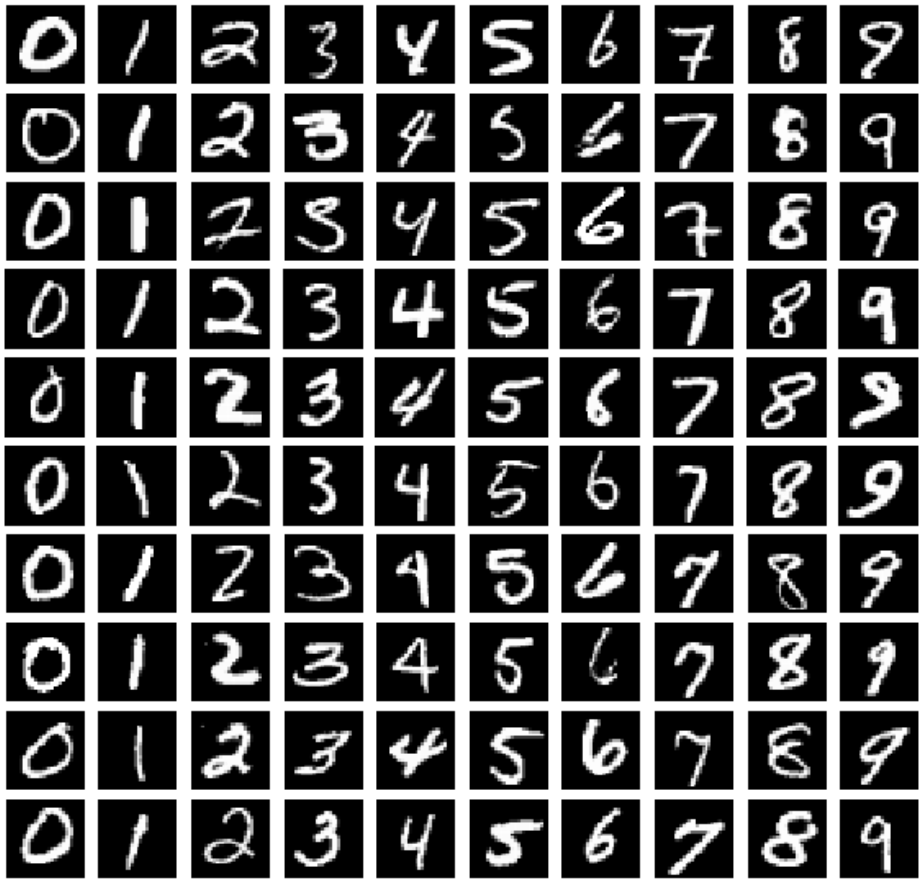
ディレクトリ構造

```
03_pytorch
├ outputs
┃        ├ advertising_actual_vs_predicted.png（出力結果）
┃        └ advertising_loss_history.png（出力結果）
├ scripts
┃        ├ nn_predict.py（演習）
┃        └ nn_cls.py
├ pyproject.toml
└ requirements.txt
```

手書き文字画像の分類（分類問題）

データセット（MNIST）

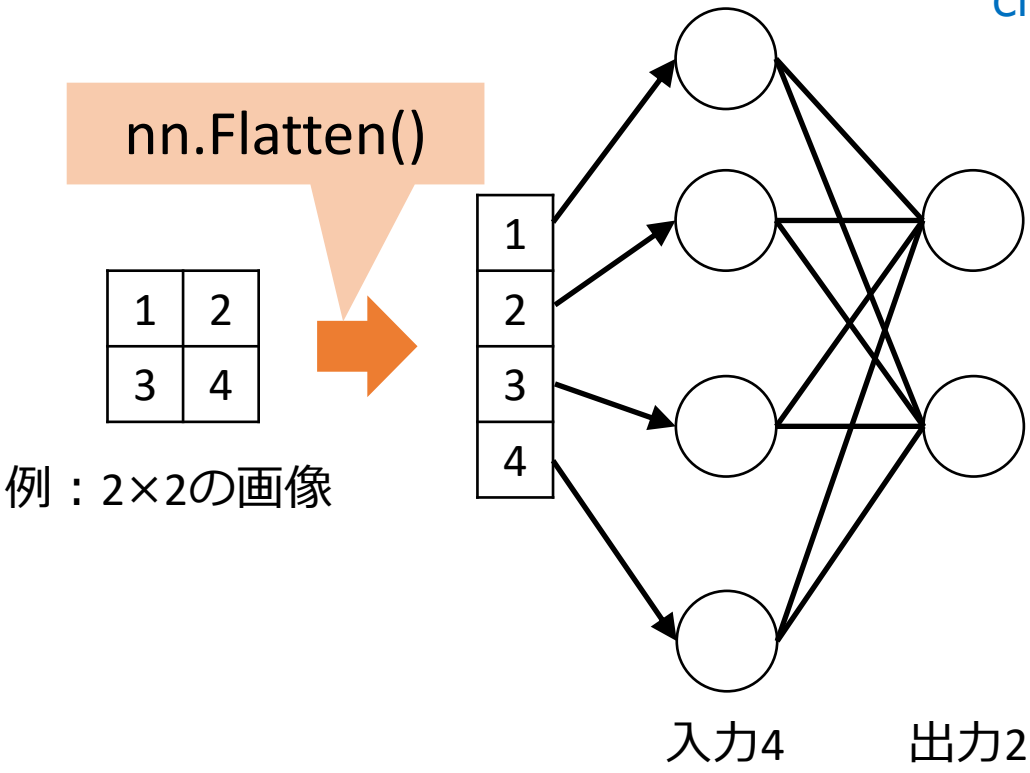
- 手書き文字のデータセット
- 学習データ60000枚，テストデータ10000枚
  - ◆それぞれにラベル（0～9）が付与



それぞれサイズは
28×28×1

全結合層へ画像データ入力時の注意

- 画像は2次元方向にあるが全結合層は1次元
- 2次元を1次元に変換してから入力

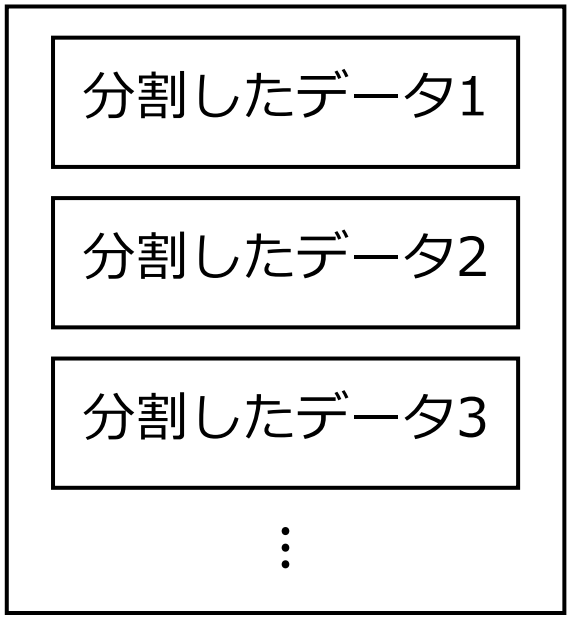



```
class クラス名(nn.Module):
    def __init__(self):
        super(クラス名, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(4, 2)

    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        return x
```

ミニバッチ学習

- データをいくつかのセットに分割し，その小さな学習データで学習する方法
  - ◆学習時に全てのデータを入力できるとは限らない
    - メモリの使用量などの都合
    - 特に画像データは容量が大きいので問題になる



分割は毎回ランダムで決める

PyTorchでのミニバッチ学習は
DataLoaderという形式で
データセットを変換し実行

データの型を変換(2)

- PyTorchのTensorDataset
  →DataLoaderの順番で変換
- 例

```
inputs = torch.tensor(x, dtype=torch.float32)
targets = torch.tensor(t, dtype=torch.float32)

#入力データと教師データをセットにしてくれる
dataset = torch.utils.data.TensorDataset(inputs, targets)
# batch_sizeで分割したデータの数kを決める
# shuffle=Trueにするとミニバッチをランダムに取り出す
loader = DataLoader(
            dataset, batch_size=batch_size, shuffle=shuffle)
```

代表的なデータセットをPyTrochで用意

- MNISTやCIFAR-10などはPyTorchの関数でダウンロードと読み込みが可能
- MNISTをダウンロードし，学習データとテストデータを用意する例
- 例

```
transform = transforms.Compose([transforms.ToTensor()])
train_dataset = torchvision.datasets.MNIST(
        root="../data", train=True, download=True, transform=transform
)
test_dataset = torchvision.datasets.MNIST(
        root="../data", train=False, download=True, transform=transform
)

train_loader = DataLoader(
                train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(
                test_dataset, batch_size=batch_size, shuffle=False)
```

ミニバッチ学習を取り入れた学習ループ

```
model = クラス名()
criterion = nn.MSELoss() # 回帰の場合はMSE
optimizer = optim.SGD(model.parameters(),
lr=learning_rate)

for epoch in range(epochs):
    for inputs, targets in loader:
        outputs = model(inputs) # 1. 推論
        loss = criterion(outputs, targets) # 2. 誤差を計算
        optimizer.zero_grad() # 3. 勾配をリセット
        loss.backward() # 4. 誤差逆伝播
        optimizer.step() # 5. 重みの更新
```

DataLoaderの中に
入力データと教師データがある
←これを取り出す作業

※ 1エポックの中でミニバッチごとに学習 = トータルの学習回数が増える
この学習回数をイテレーションという

各エポックごとにテストを実行

- テストデータを使って，学習データが汎用的な性能（未知データに対して対応できるか）検証する
  - （本当は回帰でもやるべき）
  - 例

```
for epoch in range(epochs):
    for inputs, targets in train_loader:
        # 学習
    for inputs, targets in test_loader:
        outputs = model(inputs)
        loss = criterion(outputs, targets)
```

やるのは大体
順伝播と誤差計算
重みの更新は不要

演習(6)：手書き文字画像の分類

- PyTorchでニューラルネットワークを実装し学習を実行
  - ◆学習データのDataLoader：train_loader
  - ◆テストデータのDataLoader ：test_loader
  - ◆誤差関数：クロスエントロピー
  - ◆最適化器：SGD
  - ◆入力784，出力10の全結合層を実装

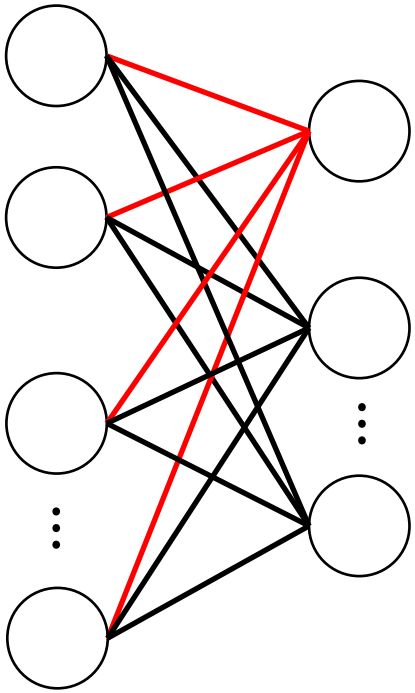
- プログラムの実行（/scripts内で実行）
  $ uv run nn_cls.py

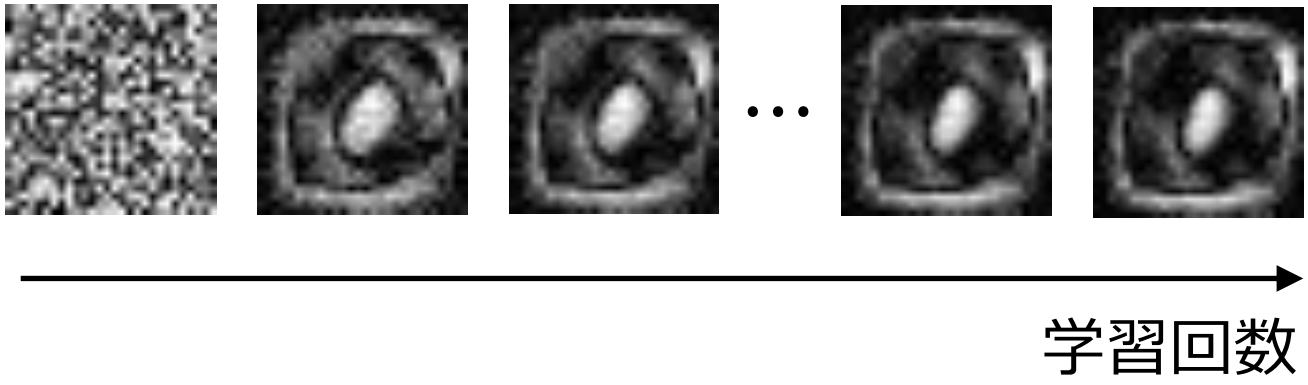
ディレクトリ構造

```
03_pytorch
├ outputs
┃        ├ mnist_accuracy.png（出力結果）
┃        └ mnist_loss.png（出力結果）
├ scripts
┃        ├ nn_predict.py
┃        └ nn_cls.py（演習）
├ pyproject.toml
└ requirements.txt
```

補足：MNISTを学習した重みの可視化

- 重みの絶対値を28×28に整形して表示
- 学習が進むほど数字様の形を帯びていく
  - ◆重みが特徴抽出をしている証拠

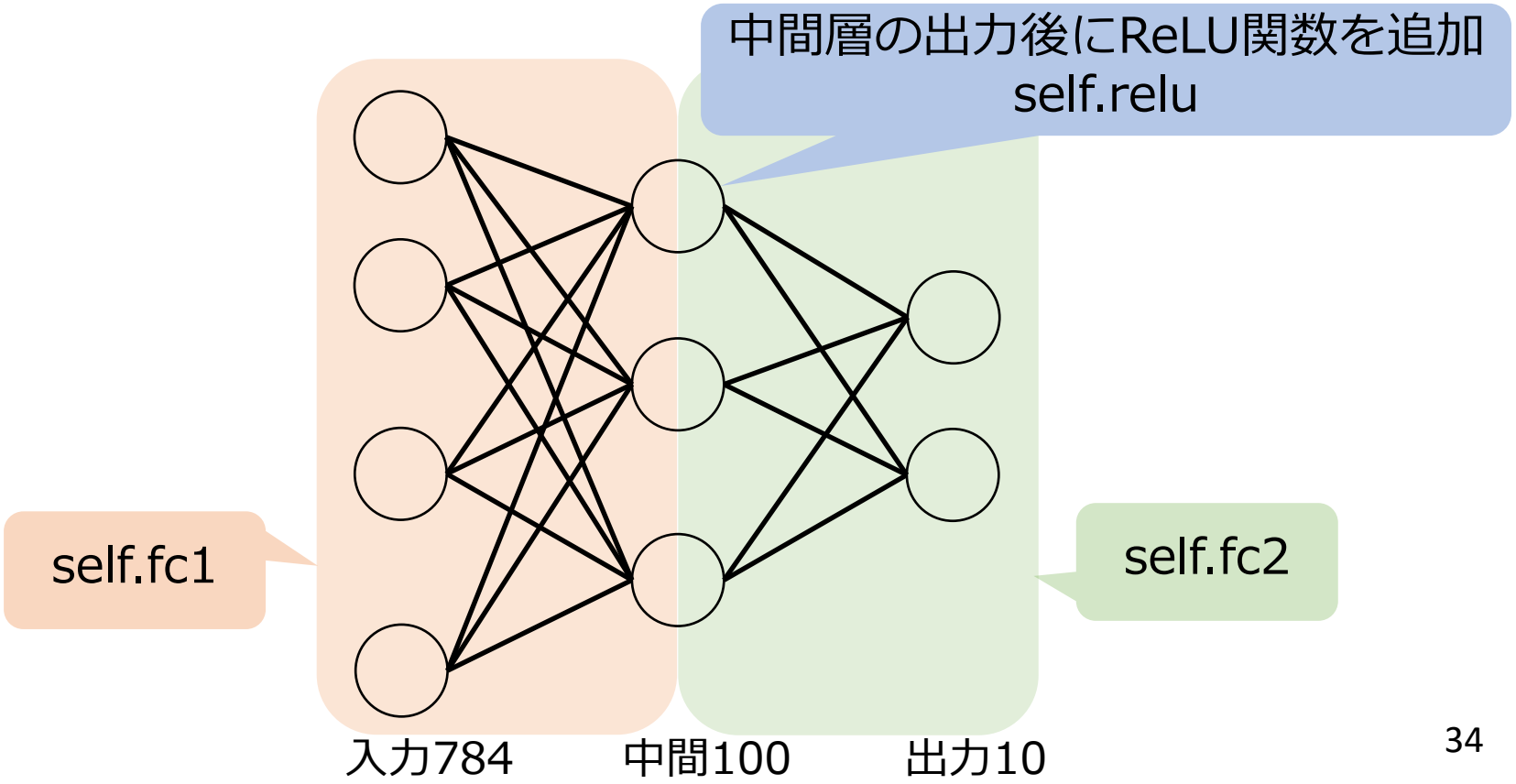




課題・考察

考察(3)

- モデルを変更し，変更前と比較し述べよ.
  - ◆変更後：入力784→中間100→ReLU関数→出力10
    - モデルの構造以外にプログラムを追記しても良い
  - ◆プログラムはex_03_学籍番号.pyとして作成せよ.



考察(4)

- 変更後のモデルでエポックを30など増やし，学習回数を増やす．するとテストデータの正解率が下がり誤差が上昇してく．この理由を調べよ.

考察(5)

- ニューラルネットワークの学習では非線形な活性化関数を用いる．活性化関数を用いる理由を調査しまとめよ.
  - ◆数式を添えて説明する
  - ◆ヒント：活性化関数がない状態はただの線形変換の積み重ねなので，多層にする意味がなくなる