aa-7. 学習と検証、学習不足、過学習、学習曲線

（人工知能）

URL: https://www.kkaneko.jp/ai/mi/index.html

金子邦彦

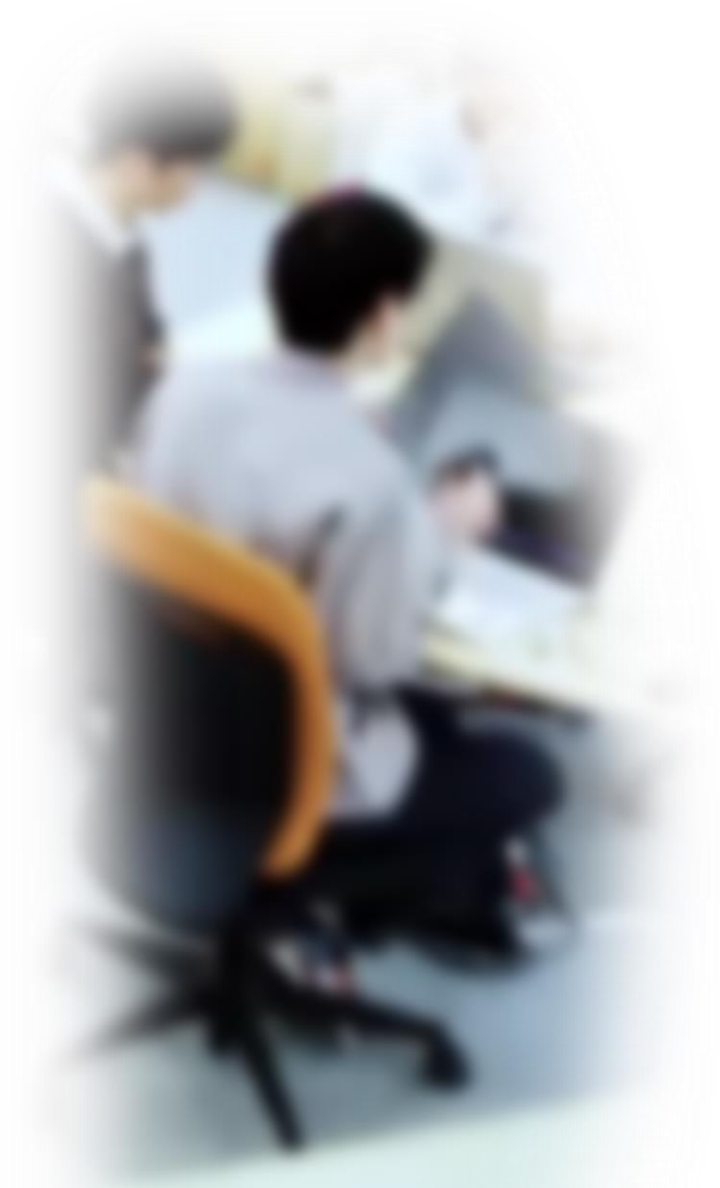
CC BY NC SA

① **機械学習の基本**を分かりやすく解説

② **汎化**、**学習曲線**、**過学習**を知る。これは、機械学習の実用知識である。

③ **データを用いた検証**による**学習不足・過学習のチェック**が成功のポイントである。

④ **ニューラルネットの過学習抑制技術**についても知る。実践に役立つ内容。



アウトライン

1. 機械学習と汎化
2. 汎化の失敗
3. 学習の繰り返し，学習曲線
4. 学習曲線に関する演習
5. ユニット数の異なるニューラルネットの学習曲線の比較
6. 別の学習曲線の例

機械学習の特徴

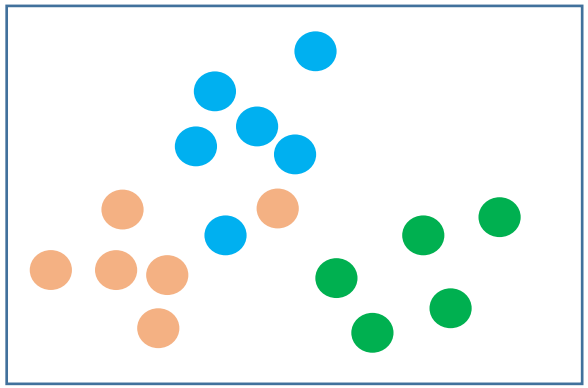
機械学習は，**コンピュータ**が**データ**を使用して**学習**することにより**知的能力を向上**させる技術

- **情報の抽出**：データの中からパターンや関係性を自動で見つけ出す能力
- **簡潔さ**：人間が設定しなければならなかったルールを、自動で生成できるようになる
- **限界の超越**：他の方法では難しかった課題でも、機械学習を用いることで解決策や視点を得られる可能性がある

機械学習と訓練データ

機械学習は，**コンピュータ**が**データ**を使用して**学習**することより**知的能力を向上**させる技術

訓練データ



3 種類に分類済み

学習

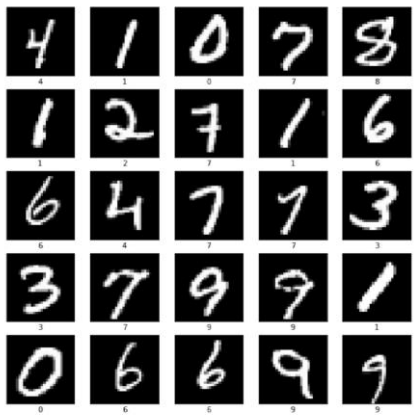
学習者

大量の訓練データを用いて
学習を行う

機械学習と訓練データとプログラム

機械学習のプログラム

学習に使用する訓練データ
（抜粋）



4 1 0 7 8
1 2 7 1 6
6 4 7 7 3
3 7 9 9 1
0 6 6 9 9

画像 60000枚
（うち一部）

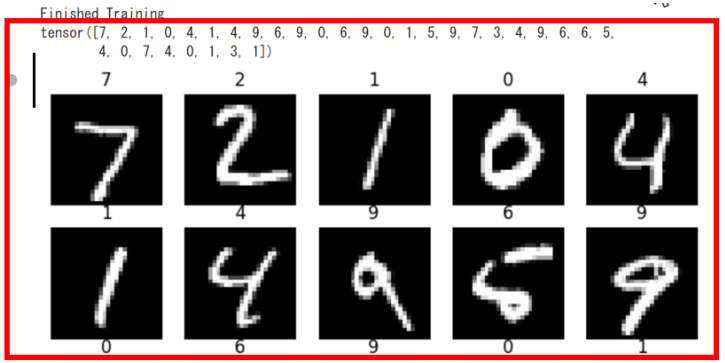
正解 60000個
（うち一部）

プログラム

データを用いて学習を行う
学習ののち，画像分類を行う

```
[4] !pip install -U scikit-learn matplotlib

    import torch
    import torch.nn as nn
    import torch.optim as optim
    from sklearn import datasets
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler
    import matplotlib.pyplot as plt

    # データの取得と前処理
    iris = datasets.load_iris()
    X = iris.data
    y = iris.target

    # データの標準化
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # 訓練データとテストデータの分割
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    X_train = torch.tensor(X_train, dtype=torch.float32)
    y_train = torch.tensor(y_train, dtype=torch.long)
    X_test = torch.tensor(X_test, dtype=torch.float32)
    y_test = torch.tensor(y_test, dtype=torch.long)

    # ニューラルネットワークの定義
    class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            self.fc1 = nn.Linear(4, 10)  # 入力は4次元（Irisの特徴量）
            self.fc2 = nn.Linear(10, 3)  # 出力は3クラス

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

    net = Net()
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.01)
```

学習の結果、文字認識の能力を獲得



```
Finished Training
tensor([7, 2, 1, 0, 4, 1, 4, 9, 6, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, 9, 6, 6, 5,
        4, 0, 7, 4, 0, 1, 3, 1])
```

機械学習まとめ

機械学習の特徴

- データを用いて知的能力を向上
- 自動でデータのパターンを抽出
- さまざまなタスクを自動実行

応用事例

画像理解、自然言語処理、予測など多数

機械学習の定義：

- **訓練データ**を用いて**学習**し、その結果として知的能力が向上
- **訓練データの追加**により、さらに**知的能力が向上**する可能性

機械学習の特徴

データ駆動型学習

- コンピュータは**訓練データ**を使用して**学習**し、知的能力を向上させる。
- **新しいデータが追加**されることで、能力のさらなる向上が期待できる

パターン認識と情報処理

- **訓練データ**を基に、**データ**内の**パターン**や**関連性**を認識
- **情報の抽出や予測**能力を獲得

機械学習の特徴

自動化と効率化

- 学習によって得られた知的能力を用いて、さまざまなタスクを自動実行

多様な能力

- 分類とバリエーション（検出，識別，文字認識，画像認識，音声認識）
- 予測
- 合成（翻訳，画像合成，文書合成など）
- 特徴抽出
- ランク付け，情報推薦

汎化

- **汎化**は，**訓練データを基に学習**し，**未知のデータも適切に処理**する能力

- **機械学習**の大きな特徴の１つ

汎化

訓練データ

入力	正解
9	500
11	500
12	1000
14	1000

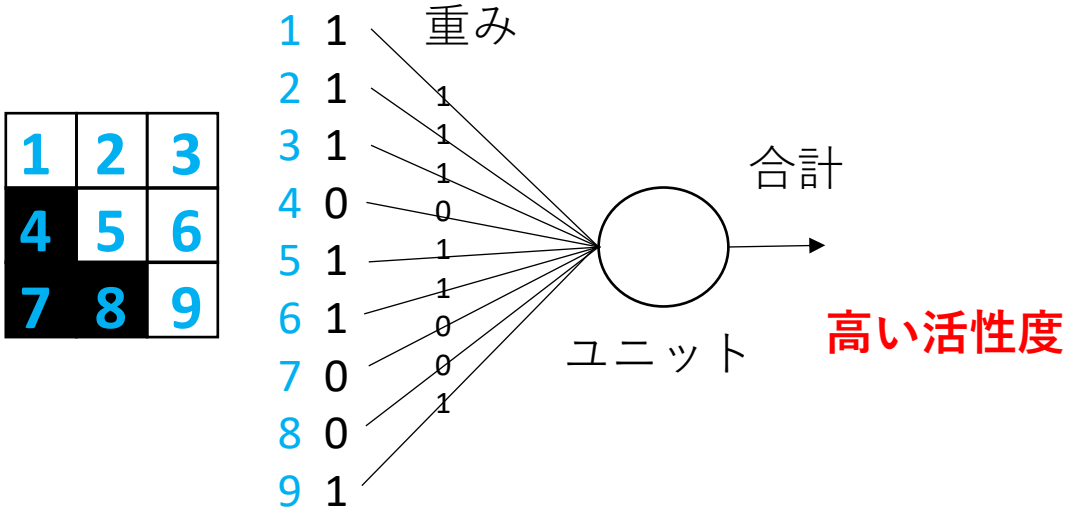
訓練データと未知のデータ

入力	予測結果
7	**500**
8	**500**
9	500
10	**500**
11	500
12	1000
13	**1000**
14	1000
15	**1000**
16	**1000**

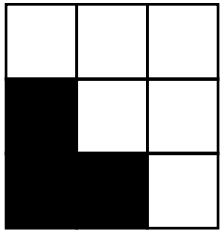
汎化は，**訓練データを基に学習**し，
未知のデータも適切に処理する能力

7-1 機械学習と汎化

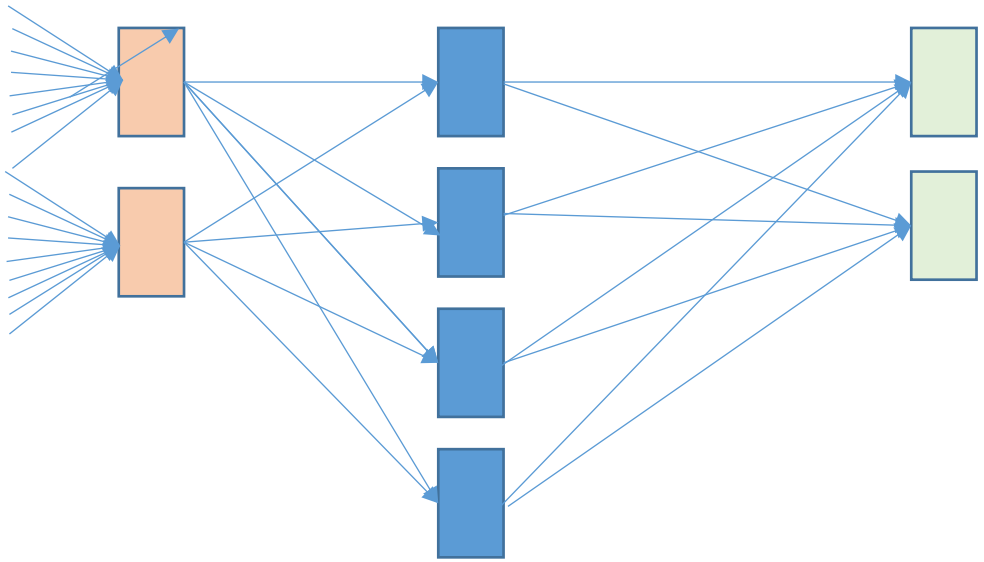
ニューラルネットワークの学習では，**結合の重みとバイアスの調整**により，訓練データの**パターンをより正確に認識できる**ようになる



ニューラルネットワークでの教師あり学習



入力



結果　　正解
0.8 ⟷ 1

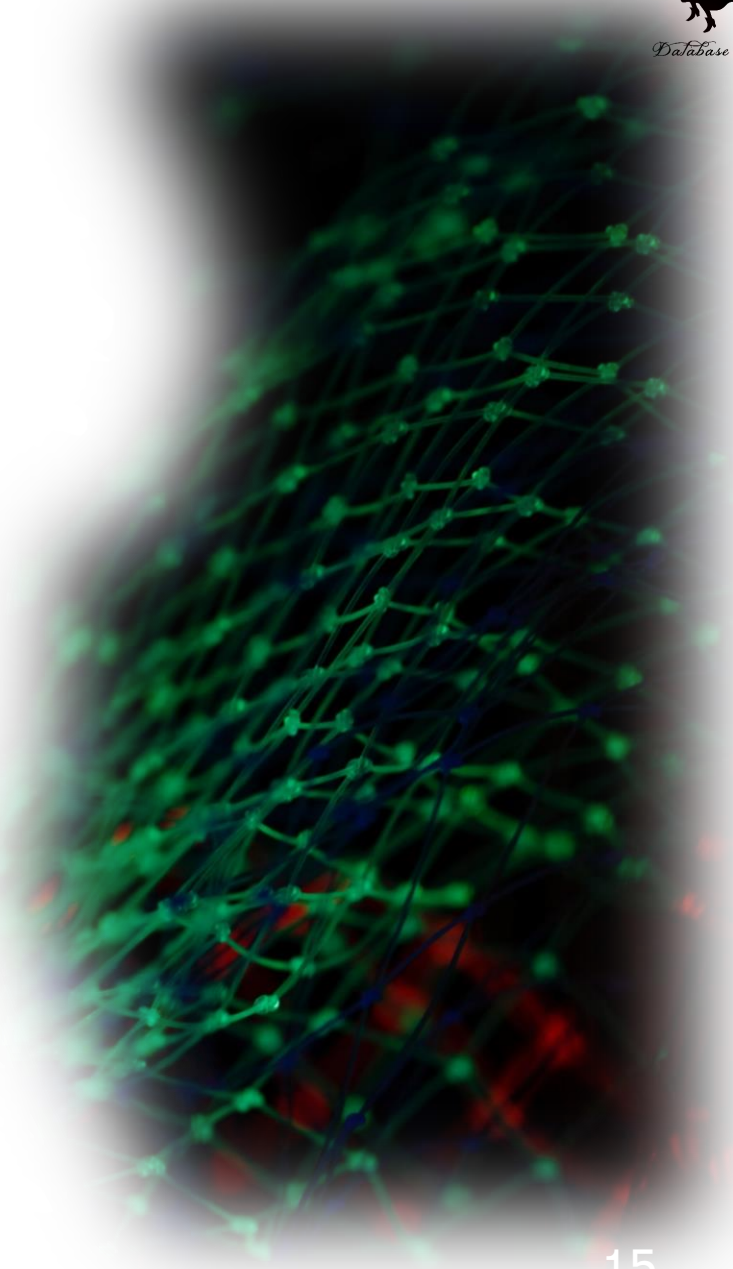
結果　　正解
0.4 ⟷ 0

誤差

実際に一度動作させてみて，
誤差が**少なくなるよう**に，
結合の重みと**バイアス**を調整
＝**ニューラルネットワーク**の**学習**

機械学習が目標とする作業

- ルールや知識のプログラム化が難しい作業
  - 直感
  - 主観
  - 経験
- データから自動的にパターンや関連性を見つけ出す



7-2 汎化の失敗

汎化の失敗

- 訓練データで学習しても，**未知のデータに対して適切な処理ができない可能性**がある

　　＝　**汎化の失敗**

- **汎化の失敗の原因**として：

訓練データに過剰に適合し、**訓練データには存在しない特徴やパターンに対して誤った結果を出す**ことがある。これを「**過学習**」という。

汎化の成功と失敗

訓練データに対して
→　**正しい**結果を出す

未知のデータに対し
→　**正しい**結果を出す

汎化の成功

訓練データに対して
→　**正しい**結果を出す

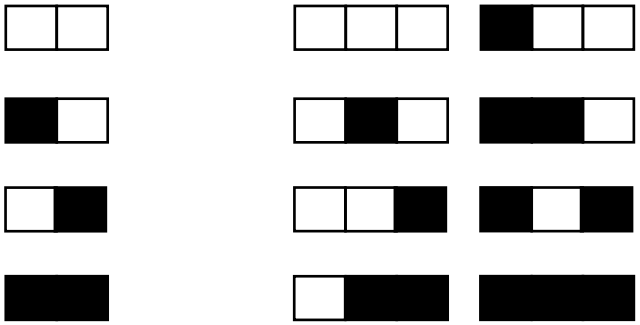
未知のデータに対し
→　**誤った**結果を出す

汎化の失敗
（過学習）

2011年までの常識

機械学習の**難問**：**組み合わせが多い場合に汎化が失敗**

それほど
難しくない問題



２マスのパターン
（**４通り**）

３マスのパターン
（**８通り**）

難しい問題

１００マスのパターン
1267650600228229401496703205376 通り
（およそ１兆の１兆の１２６万倍）

組み合わせが多い

2011年までの常識

機械学習の**難問：組み合わせが多い場合に汎化が失敗**

- **カラー画像: サイズ 32×32 の場合**

  → 画素数: 32 × 32 × 3，階調 256

  約 12000 … 000 通り

  7397個の 0

- **ステレオ音声: 10秒間，サンプリング周波数 44100Hz の場合**

  → 10 × 44100，階調 256

  約 1000 … 000 通り

  1062033個の 0

過学習を防ぐために

①まずは，**大量の訓練データが大切**

②訓練データの多様性と品質の向上も大切

③ニューラルネットワークの簡素化や調整も大切

ニューラルネットワークの**層やユニット数を増やす**ことが必ずしも良い結果をもたらすわけでない.

過去は，ニューラルネットワークの**層の数やユニット数を増やす**ことは**現実的ではない**と考えられることもあった

2009年以降の進展

- **従来からある技術と，新技術の組み合わせにより，性能向上，過学習の抑止が進展**

【新技術の例】

- 非線形性の正規化 (rectified nonlinearity), 2009年
- 活性化関数の ReLU, 2011年
- ドロップアウト
- 正則化（L1，L2）
- 学習率の調整

さらなる進展：

He の初期化, 2015年　など

- **巨大な訓練データの利用**
- **コンピュータの高速化**

→　ニューラルネットワークの**層やユニット数を増やすことが一定程度可能に.**

まとめ

- **汎化の失敗**：訓練データでの学習**だけ**では，**未知のデータ**に**適切に処理できない可能性**がある
- **汎化の失敗**は，**訓練データに過剰に適合**し，訓練データ内に存在しない特徴やパターンに対して誤った結果を出す（**過学習**）により**発生**
- **過学習の防止**は，**単純でなく**，様々な技術，調整，巨大な訓練データの準備が必要（それでも完全に防止できない）
- 技術の進展，巨大な訓練データの利用，コンピュータの高速化により、**ニューラルネットワークの層やユニット数を増やすことが一定程度可能に**なった

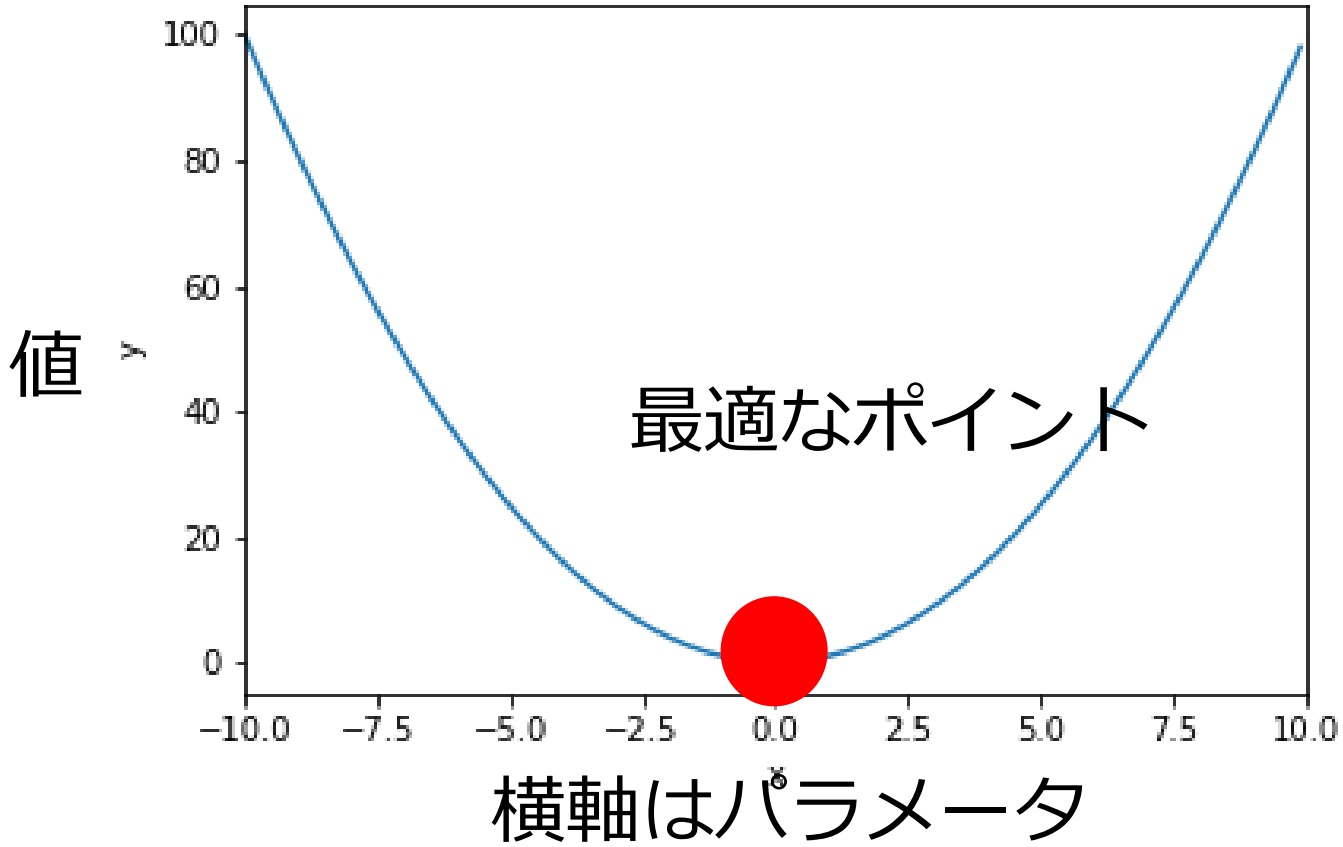
7-3 学習の繰り返し，学習曲線

ニューラルネットワークの学習

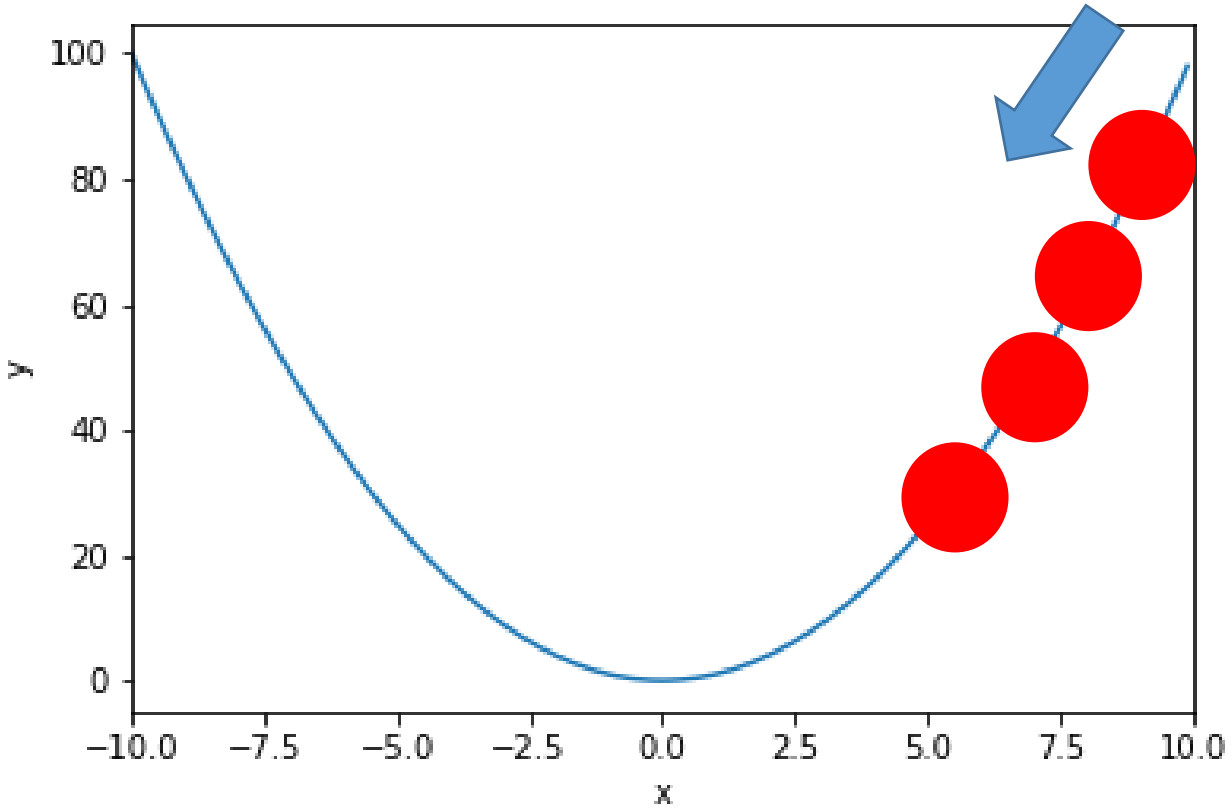
- **学習**には，**訓練データ**を使用
- **ユニットの結合の重み**が**変化する**
  - ① **訓練データ**により，ニューラルネットワークを動かし，正解との誤差を得る
  - ② 正解との誤差が少なくなるように **結合の重みが変化**

この様子は，坂道を降りる様子にも似ている

誤差が**最小になる**ポイントを探索

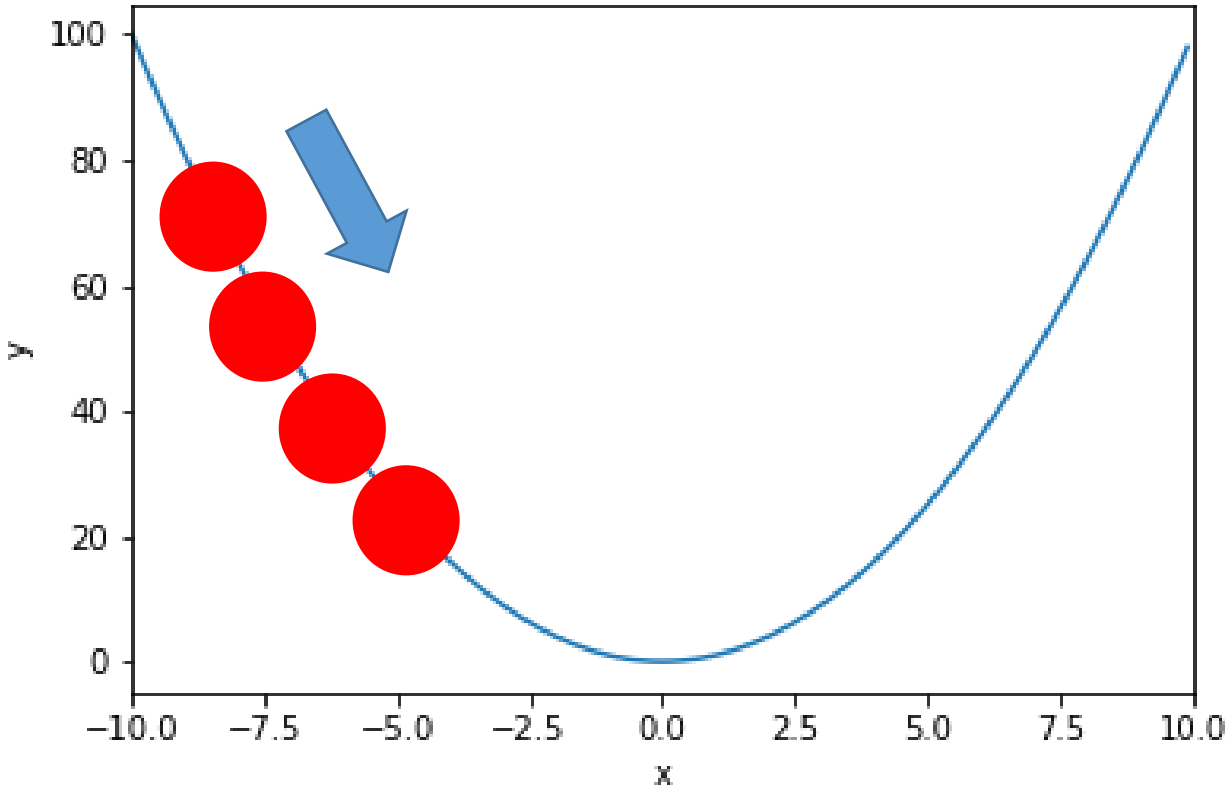


傾きが右**上がり**のとき：
パラメータ値を**減らす**と、
最適に近づく



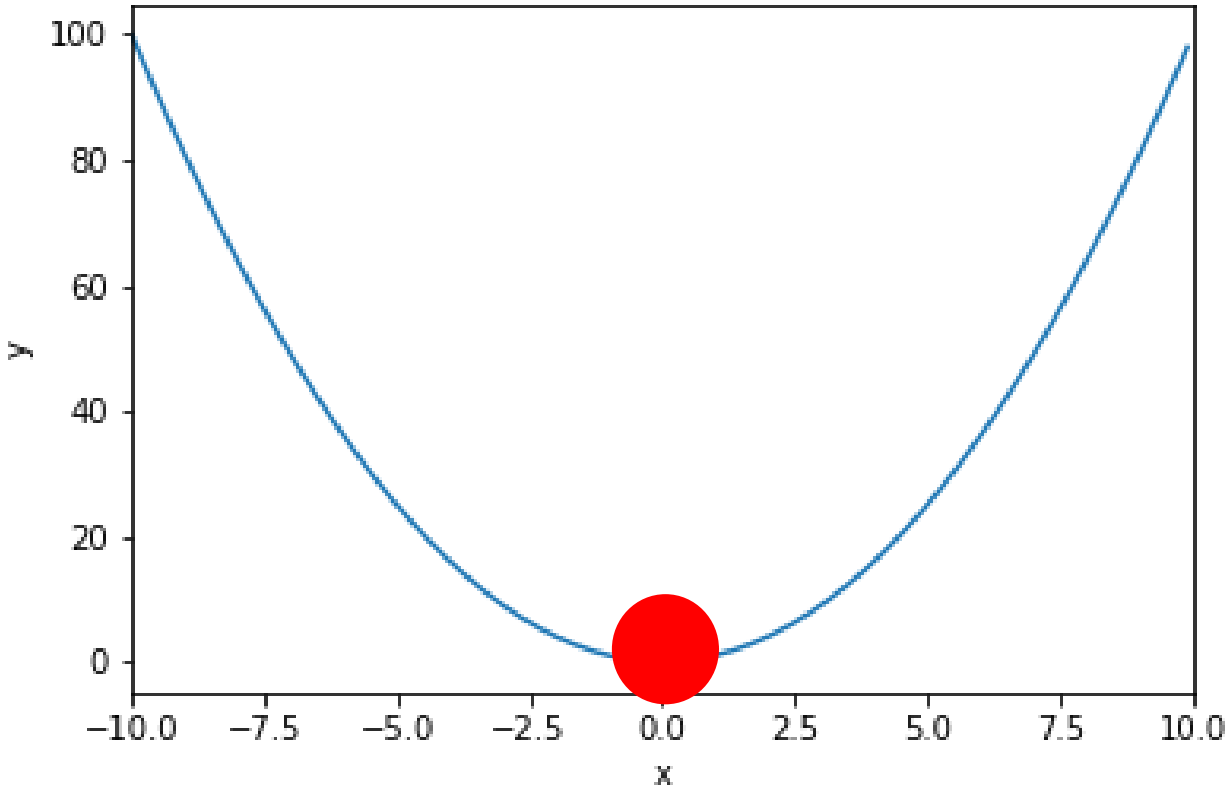
１回では最適にならない。
移動を繰り返す

傾きが右**下がり**のとき：
パラメータ値を**増やす**と、
最適に近づく



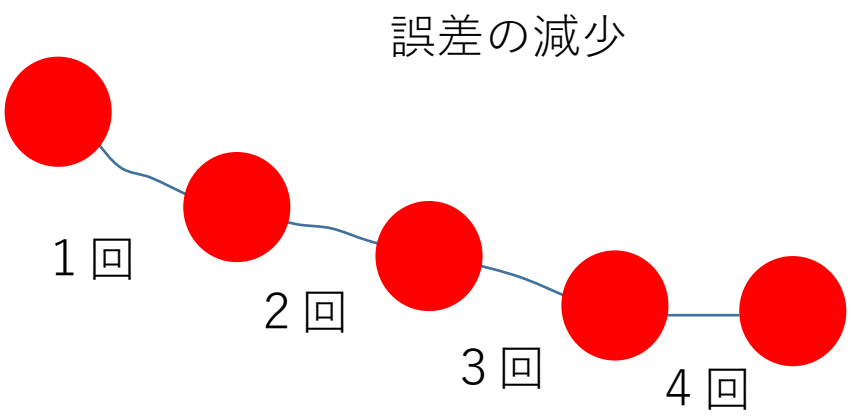
１回では最適にならない。
移動を繰り返す

最適のとき**傾きは 0**
移動はない



学習の繰り返し

- **訓練データ**を１回使っただけでは，**学習不足**の場合がある．
- 同じ**訓練データ**を**繰り返し**使用することになる



学習の繰り返しと過学習

- **訓練データ**を**繰り返し**使用するほど，**過学習の危険が高まる**
- **過学習の完全に防ぐことは不可能**

→　**訓練データ**とは別のデータ（**検証データ**）を用いて**過学習が無いことを検証する**ことが極めて重要

訓練データと検証データ

訓練データ：　**学習**に使用

訓練データによる**学習**により，**未知のデータ**を**適切に処理できる能力**（**汎化**）のために利用

検証データ：　**検証**に利用

訓練データとは別のデータによる**検証**により，**過学習が無いこと**を**確認**するために利用

汎化の成功と失敗

訓練データに対して
→　**正しい**結果を出す

検証データに対し
→　**正しい**結果を出す

汎化の成功

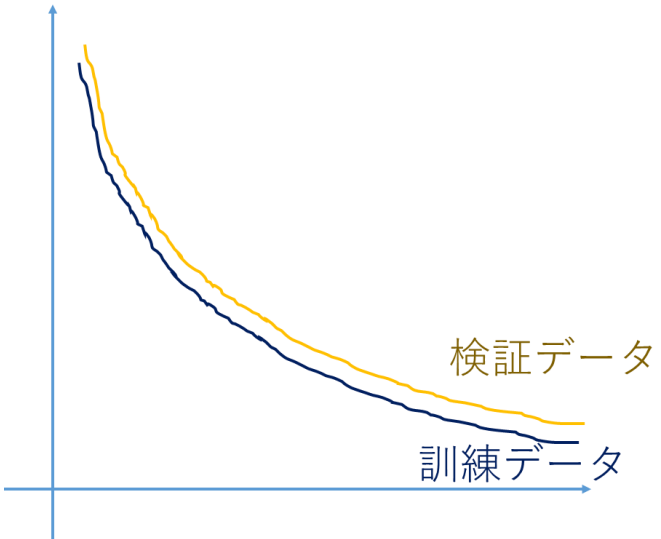
訓練データに対して
→　**正しい**結果を出す

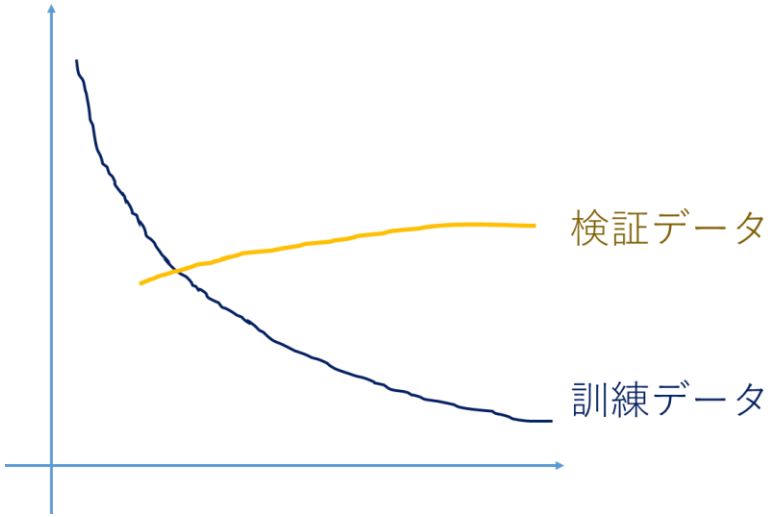
検証データに対し
→　**誤った**結果を出す

汎化の失敗
（過学習）

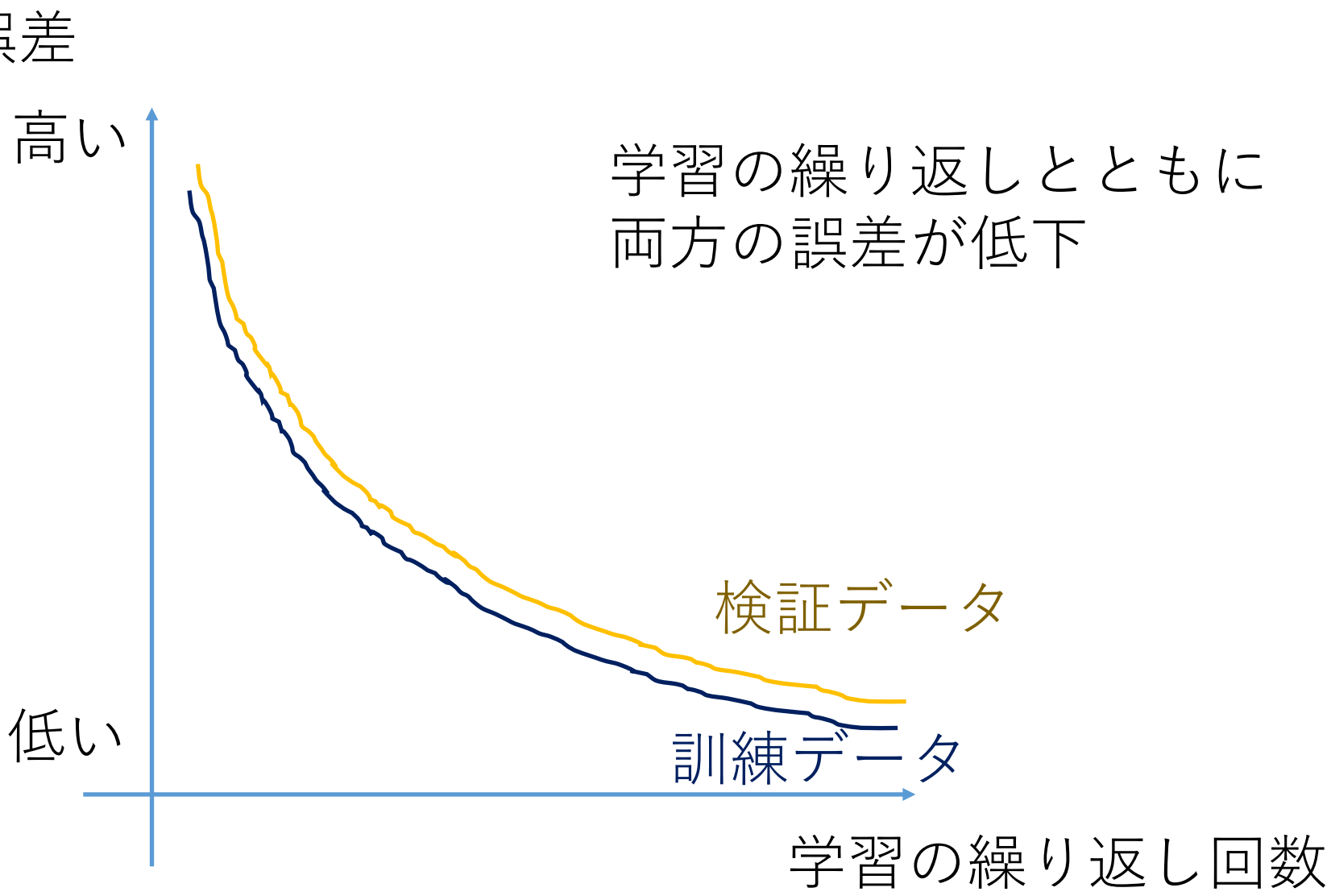
学習曲線

- **学習曲線**は、訓練データと検証データの**誤差の推移**を表したグラフ
- **訓練データを繰り返し使用する**ことで、**訓練データの誤差は徐々に減少**します
- **訓練データを繰り返し使用する**ほど，**過学習のリスクが高まる．過学習では，検証データの誤差が増加**
- **学習曲線を見る**ことで、**訓練データと検証データの誤差の差異を確認**し、**過学習の有無**を把握できる

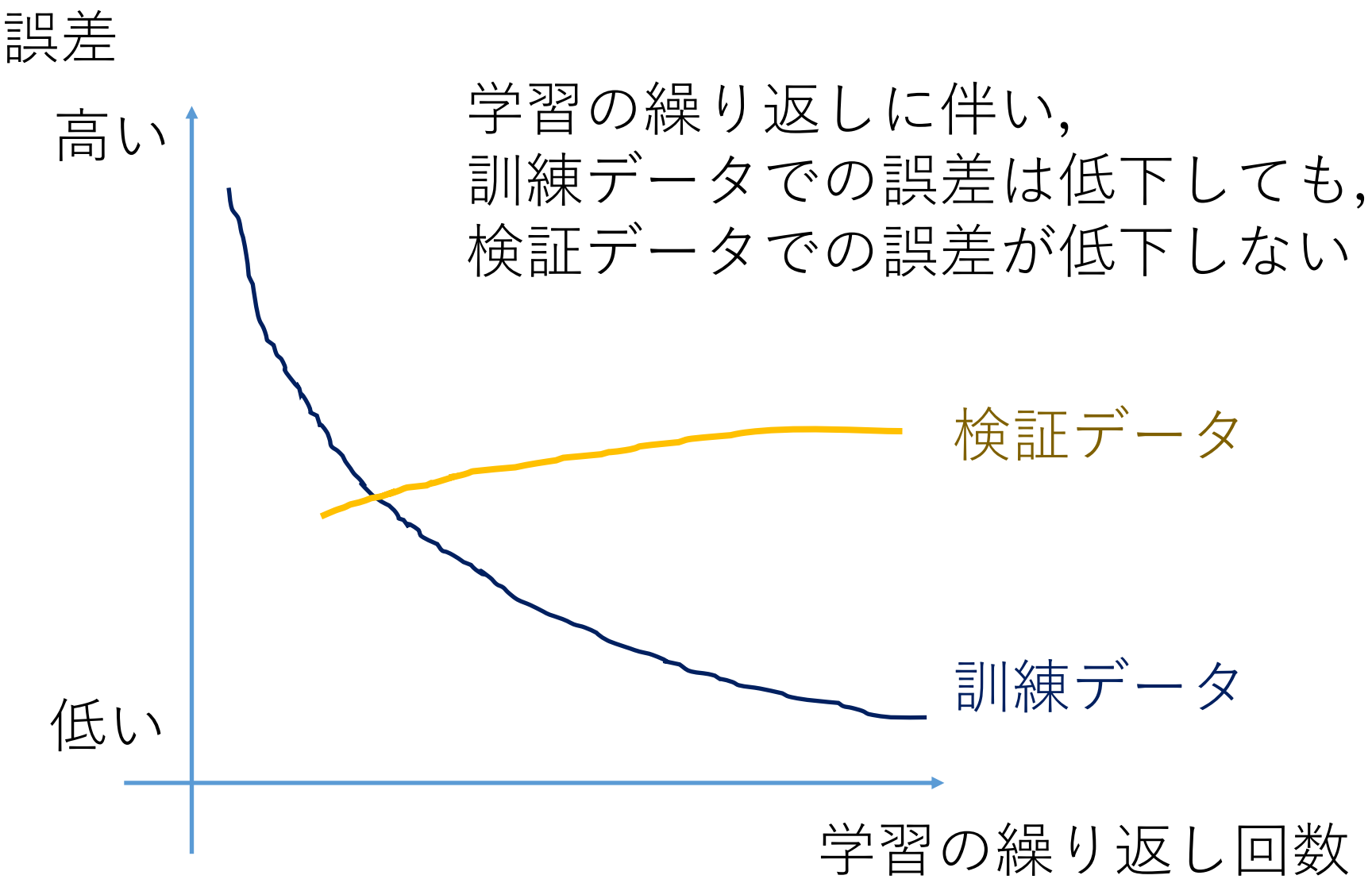




過学習なし



過学習あり


誤差
高い
低い
学習の繰り返しに伴い,
訓練データでの誤差は低下しても,
検証データでの誤差が低下しない
検証データ
訓練データ
学習の繰り返し回数

学習曲線の有用性

- **学習曲線**は，**訓練データと検証データの誤差の推移**を表すグラフ
- 訓練データを**１回だけ使用**すると**学習不足**の可能性がある
- **同じ訓練データを繰り返し使用**するほど**過学習の危険性が高まる**
- 過学習を完全に防ぐことは不可能．**検証データ**を使用して**過学習の有無を確認**することが重要
- **学習曲線を見る**ことで、**訓練データと検証データの誤差の差異**を確認し、**過学習が無いことの判断**が可能

7-4 学習曲線に関する演習

画像分類の結果

４つの画像を分類

分類結果　１０個の数値が４つ

```
[2.82898581e-20 2.11713194e-20 1.00000000e+00 1.01542401e-08
 3.07505820e-18 6.13309550e-19 1.73079867e-17 3.75218205e-16
 2.25546036e-12 1.27005634e-19]                                  2
[1.00000000e+00 0.00000000e+00 7.43346550e-24 7.99614704e-31
 1.60335865e-35 7.09844889e-24 1.09644683e-16 3.93163016e-20
 4.51085197e-28 9.15929917e-33]                                  0
[4.02610817e-21 5.27186802e-21 1.43940942e-19 6.05407705e-22
 1.00000000e+00 1.87728168e-24 1.43710434e-20 5.10361284e-13
 1.50390224e-20 5.03197449e-11]                                  4
[1.59357307e-06 3.12464854e-09 4.91066432e-09 9.94732474e-09
 1.01848738e-11 1.25744277e-08 1.01212549e-08 3.50234806e-11
 9.99998331e-01 2.44763920e-09]                                  8
```

プログラムは、次で公開

https://colab.research.google.com/drive/1IfArIvhh-FsvJIE9YTNO8T44Qhpi0rIJ?usp=sharing

- 実行結果とプログラムと説明を**見るだけ**なら、**Google アカウントは不要**

- **プログラムを変更し再実行する**には，**Google アカウント**が必要.

学習の繰り返しを行うプログラム例

学習の繰り返し回数は **20**

```
EPOCHS=20
history = m.fit(x=ds_train[0],
          y=ds_train[1],
          epochs=EPOCHS,
          validation_data=(ds_test[0], ds_test[1]),
          callbacks=[tensorboard_callback], verbose=2)
```

訓練データの指定

検証データの指定

学習の繰り返しを行うプログラムと実行結果

同じ訓練データを用いた学習を２０回繰り返し．
そのとき，検証データで検証

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

EPOCHS=20
history = m.fit(x=ds_train[0],
          y=ds_train[1],
          epochs=EPOCHS,
          validation_data=(ds_test[0], ds_test[1]),
          callbacks=[tensorboard_callback], verbose=2)

m.evaluate(ds_test[0], ds_test[1], verbose=2)
```

```
Epoch 1/20
1875/1875 - 5s - loss: 0.2589 - accuracy: 0.9262 - val_loss: 0.1427 - val_accuracy: 0.9594 - 5s/epoch - 3ms/step
Epoch 2/20
1875/1875 - 4s - loss: 0.1151 - accuracy: 0.9668 - val_loss: 0.0944 - val_accuracy: 0.9713 - 4s/epoch - 2ms/step
Epoch 3/20
1875/1875 - 4s - loss: 0.0778 - accuracy: 0.9763 - val_loss: 0.0808 - val_accuracy: 0.9736 - 4s/epoch - 2ms/step
Epoch 4/20
1875/1875 - 4s - loss: 0.0573 - accuracy: 0.9828 - val_loss: 0.0786 - val_accuracy: 0.9755 - 4s/epoch - 2ms/step
Epoch 5/20
1875/1875 - 4s - loss: 0.0450 - accuracy: 0.9859 - val_loss: 0.0743 - val_accuracy: 0.9765 - 4s/epoch - 2ms/step
Epoch 6/20
1875/1875 - 4s - loss: 0.0343 - accuracy: 0.9892 - val_loss: 0.0762 - val_accuracy: 0.9756 - 4s/epoch - 2ms/step
Epoch 7/20
1875/1875 - 4s - loss: 0.0285 - accuracy: 0.9912 - val_loss: 0.0720 - val_accuracy: 0.9791 - 4s/epoch - 2ms/step
Epoch 8/20
1875/1875 - 4s - loss: 0.0226 - accuracy: 0.9931 - val_loss: 0.0770 - val_accuracy: 0.9785 - 4s/epoch - 2ms/step
Epoch 9/20
1875/1875 - 4s - loss: 0.0183 - accuracy: 0.9943 - val_loss: 0.0774 - val_accuracy: 0.9792 - 4s/epoch - 2ms/step
Epoch 10/20
1875/1875 - 4s - loss: 0.0150 - accuracy: 0.9955 - val_loss: 0.0797 - val_accuracy: 0.9784 - 4s/epoch - 2ms/step
Epoch 11/20
1875/1875 - 4s - loss: 0.0141 - accuracy: 0.9956 - val_loss: 0.0828 - val_accuracy: 0.9784 - 4s/epoch - 2ms/step
Epoch 12/20
1875/1875 - 4s - loss: 0.0107 - accuracy: 0.9965 - val_loss: 0.0821 - val_accuracy: 0.9786 - 4s/epoch - 2ms/step
Epoch 13/20
1875/1875 - 4s - loss: 0.0106 - accuracy: 0.9967 - val_loss: 0.0914 - val_accuracy: 0.9774 - 4s/epoch - 2ms/step
Epoch 14/20
1875/1875 - 4s - loss: 0.0085 - accuracy: 0.9974 - val_loss: 0.0931 - val_accuracy: 0.9766 - 4s/epoch - 2ms/step
Epoch 15/20
1875/1875 - 4s - loss: 0.0068 - accuracy: 0.9981 - val_loss: 0.0915 - val_accuracy: 0.9781 - 4s/epoch - 2ms/step
Epoch 16/20
1875/1875 - 4s - loss: 0.0076 - accuracy: 0.9977 - val_loss: 0.0885 - val_accuracy: 0.9798 - 4s/epoch - 2ms/step
Epoch 17/20
1875/1875 - 5s - loss: 0.0065 - accuracy: 0.9980 - val_loss: 0.0965 - val_accuracy: 0.9792 - 5s/epoch - 3ms/step
Epoch 18/20
1875/1875 - 5s - loss: 0.0060 - accuracy: 0.9981 - val_loss: 0.0936 - val_accuracy: 0.9785 - 5s/epoch - 3ms/step
Epoch 19/20
1875/1875 - 4s - loss: 0.0063 - accuracy: 0.9981 - val_loss: 0.1017 - val_accuracy: 0.9784 - 4s/epoch - 2ms/step
Epoch 20/20
1875/1875 - 4s - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.1105 - val_accuracy: 0.9761 - 4s/epoch - 2ms/step
313/313 - 0s - loss: 0.1105 - accuracy: 0.9761 - 394ms/epoch - 1ms/step
[0.1105121374130249, 0.9761000275611877]
```

プログラム

実行結果

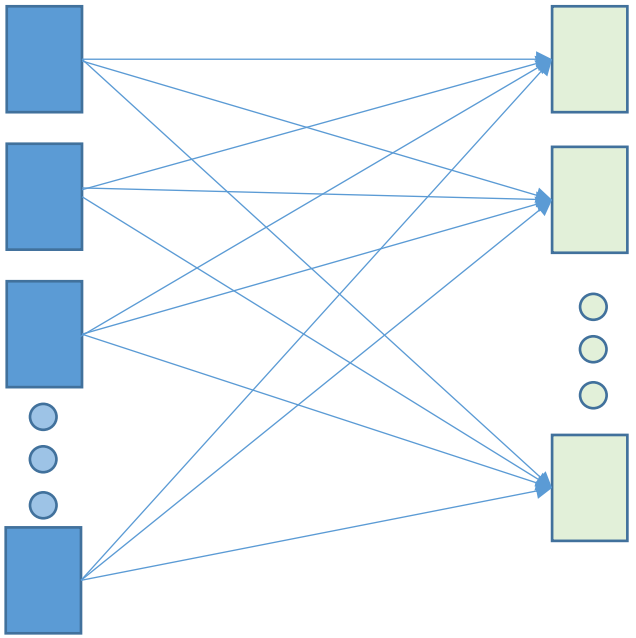
学習の**繰り返し**ごとに，
訓練データや検証データ
での**精度**や**誤差**
の変化を確認

学習が終了したときの結合の重み

１層目のユニットは**１２８個**

２層目のユニットは**１０個**

従って，**結合**は　１２８×１０の　**１２８０個**

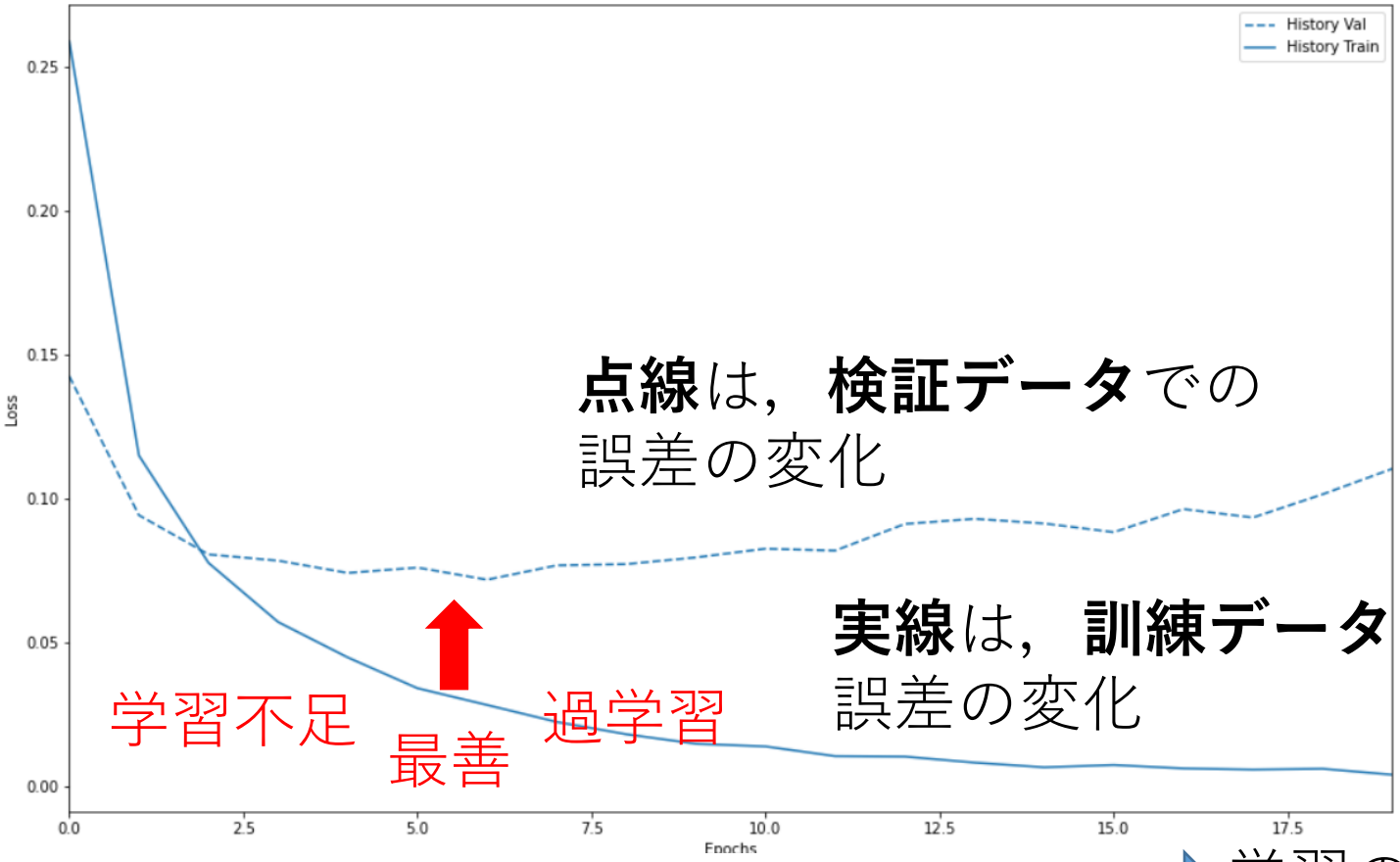


```
[15] m.get_weights()[2]

     array([[-0.06206315,  0.08728168,  0.1853286 , ...,  0.40357363,
             -0.693989  ,  0.42249945],
            [ 0.45693907,  0.37475306, -0.7932407 , ..., -0.00343985,
             -0.01948859,  0.42894104],
            [ 0.5918742 ,  0.11614478, -0.8738301 , ..., -0.28256747,
              0.29315227,  0.23105301],
            ...,
            [-0.59012157,  0.43835095, -0.8013934 , ..., -0.48010653,
             -1.2864426 ,  0.44753826],
            [ 0.26202166, -0.09861455,  0.17655255, ...,  0.14542623,
             -0.5842476 ,  0.12834716],
            [ 0.19861923, -0.34796983, -0.37779674, ...,  0.10915083,
              0.23489822,  0.02567334]], dtype=float32)
```

結合の重みを表示した結果（抜粋）

学習曲線

学習の繰り返しでの，**誤差などの変化**をプロットしたグラフ



訓練データと検証データでは，違う形になることに注意

7-5 ユニット数の異なるニューラルネットの学習曲線の比較

いまから行うこと

- ニューラルネットワークを**３種類**作る

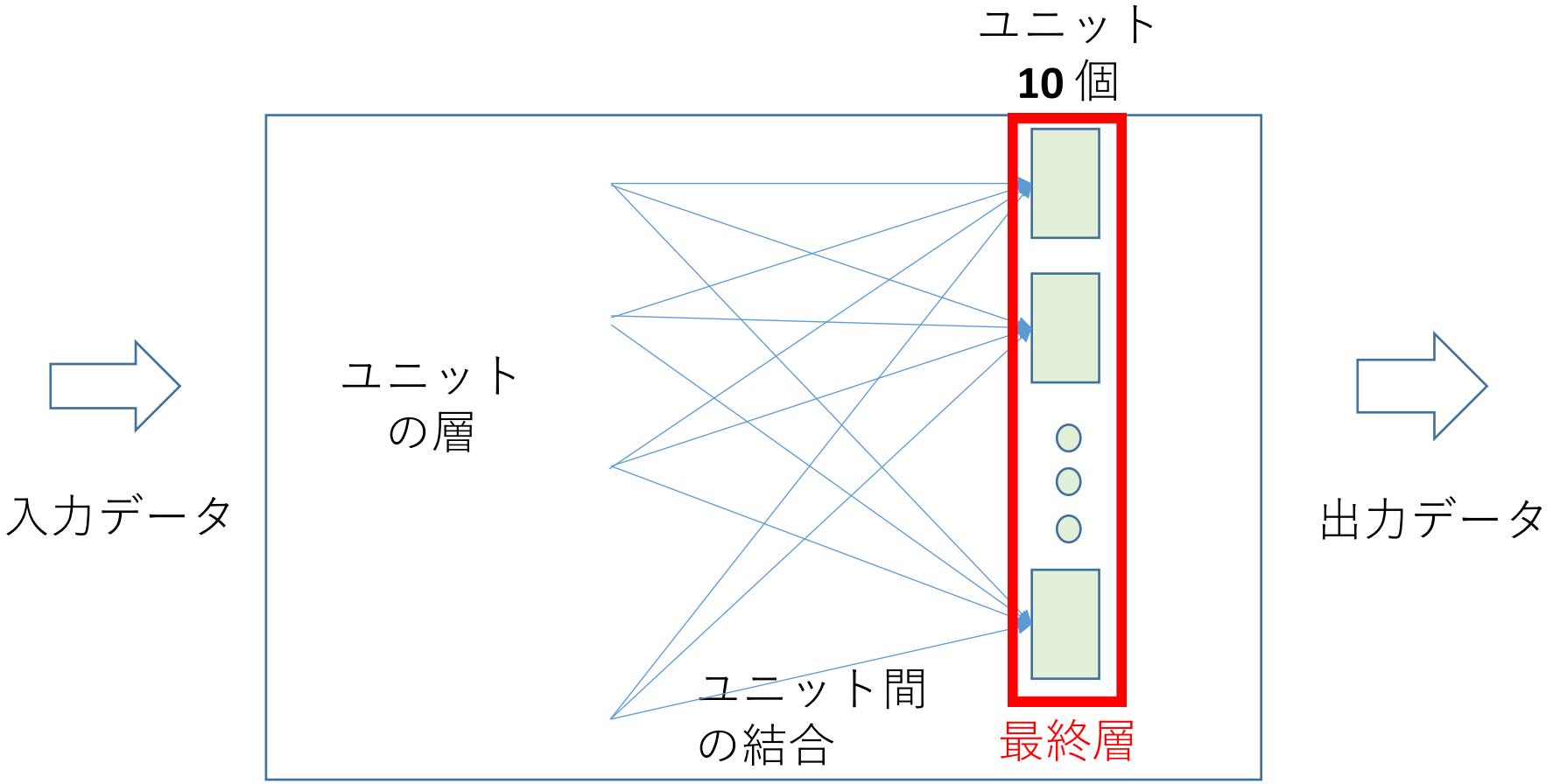
  ＜３種類の違い＞

  １層目のユニット数: **400**, **4000**, **40000**

- **どれが精度よく分類できるか**，検証データで確認

１０種類に分類するニューラルネットワーク

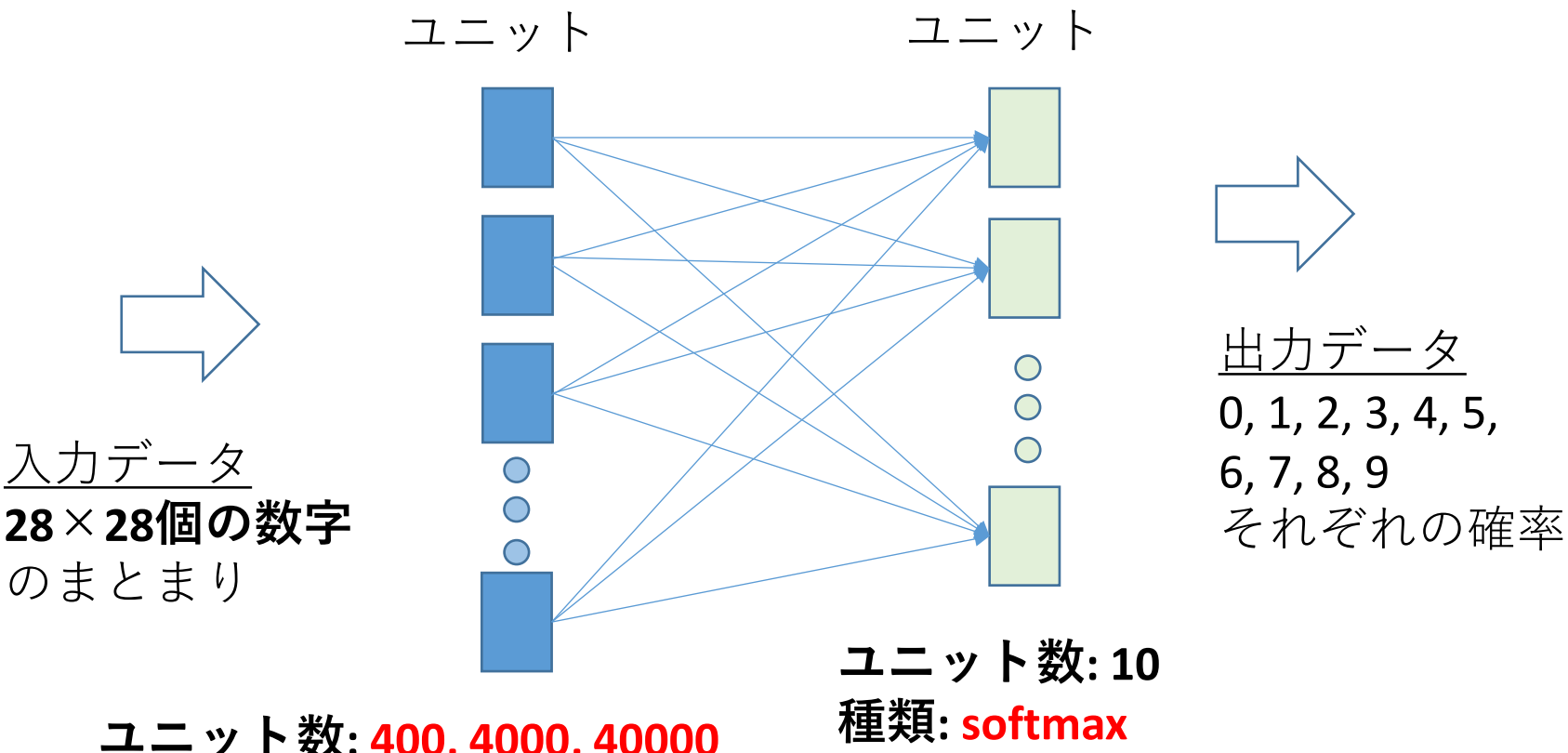
最終層について，１つが強く
活性化するように調整



データは入力から出力の方向へ

作成するニューラルネットワーク

- **１層目：ユニット数 400, 4000, 40000（3通り），種類は relu**
- **２層目：ユニット数 10, 種類は softmax**



全体で 2層

ニューラルネットワーク作成のプログラム例

１層目のユニット数：400 の場合

入力データは **28×28**個の数字

１層目のユニット数は **400**
種類は **relu**

```
import tensorflow as tf
m = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(units=400, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

２層目のユニット数は **10**
種類は **softmax**

Flatten は，2次元の配列（アレイ）を，ニューラルネットワークの入力にできるようにするためのもの

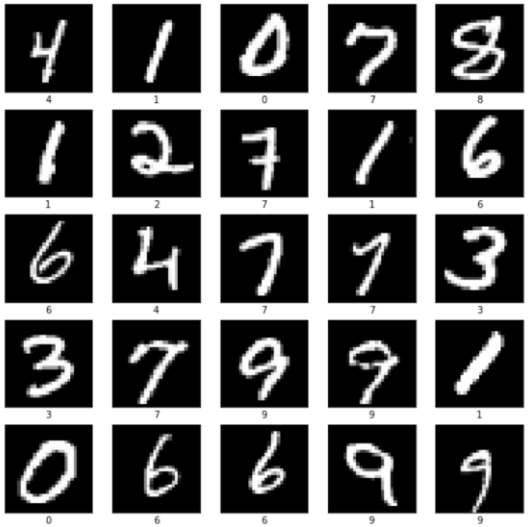
ニューラルネットワークの作成では，次を設定する

- 入力データでの数値の個数
- **ユニット**の数（**層**ごと）
- **ユニット**の種類（**層**ごと）

学習のための準備

- **ニューラルネットワーク**の作成
- **訓練データ**　（学習用）

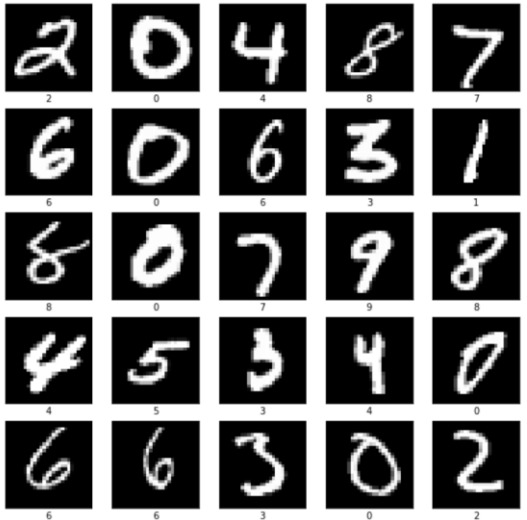
60000枚の画像と正解



抜粋

- **検証データ**　（検証用）

10000枚の画像と正解



抜粋

学習の繰り返しを行うプログラム例

学習の繰り返し回数は **20**

```
EPOCHS=20
history = m.fit(x=ds_train[0],
          y=ds_train[1],
          epochs=EPOCHS,
          validation_data=(ds_test[0], ds_test[1]),
          callbacks=[tensorboard_callback], verbose=2)
```

訓練データの指定

検証データの指定

学習の繰り返しを行うプログラムと実行結果

同じ訓練データを用いた学習を２０回繰り返し．
そのとき，検証データで検証

```
EPOCHS=20
history = m.fit(x=x_train,
          y=y_train,
          epochs=EPOCHS,
          validation_data=(x_test, y_test),
          callbacks=[tensorboard_callback],
          verbose=2)
```

プログラム

```
---------- m.fit ----------
Epoch 1/20
1875/1875 - 27s - loss: 0.2683 - accuracy: 0.9242 - val_loss: 0.1488 - val_accuracy: 0.9569 - 27s/epoch - 14ms/step
Epoch 2/20
1875/1875 - 26s - loss: 0.1204 - accuracy: 0.9658 - val_loss: 0.1064 - val_accuracy: 0.9680 - 26s/epoch - 14ms/step
Epoch 3/20
1875/1875 - 26s - loss: 0.0822 - accuracy: 0.9768 - val_loss: 0.0820 - val_accuracy: 0.9746 - 26s/epoch - 14ms/step
Epoch 4/20
1875/1875 - 26s - loss: 0.0618 - accuracy: 0.9828 - val_loss: 0.0755 - val_accuracy: 0.9782 - 26s/epoch - 14ms/step
Epoch 5/20
1875/1875 - 26s - loss: 0.0476 - accuracy: 0.9872 - val_loss: 0.0649 - val_accuracy: 0.9794 - 26s/epoch - 14ms/step
Epoch 6/20
1875/1875 - 26s - loss: 0.0377 - accuracy: 0.9900 - val_loss: 0.0613 - val_accuracy: 0.9814 - 26s/epoch - 14ms/step
Epoch 7/20
1875/1875 - 25s - loss: 0.0304 - accuracy: 0.9924 - val_loss: 0.0602 - val_accuracy: 0.9823 - 25s/epoch - 13ms/step
Epoch 8/20
1875/1875 - 25s - loss: 0.0246 - accuracy: 0.9941 - val_loss: 0.0608 - val_accuracy: 0.9805 - 25s/epoch - 14ms/step
Epoch 9/20
1875/1875 - 25s - loss: 0.0199 - accuracy: 0.9959 - val_loss: 0.0554 - val_accuracy: 0.9823 - 25s/epoch - 13ms/step
Epoch 10/20
1875/1875 - 25s - loss: 0.0163 - accuracy: 0.9973 - val_loss: 0.0565 - val_accuracy: 0.9819 - 25s/epoch - 14ms/step
Epoch 11/20
1875/1875 - 26s - loss: 0.0137 - accuracy: 0.9981 - val_loss: 0.0567 - val_accuracy: 0.9816 - 26s/epoch - 14ms/step
Epoch 12/20
1875/1875 - 25s - loss: 0.0114 - accuracy: 0.9987 - val_loss: 0.0557 - val_accuracy: 0.9827 - 25s/epoch - 13ms/step
Epoch 13/20
1875/1875 - 25s - loss: 0.0098 - accuracy: 0.9988 - val_loss: 0.0535 - val_accuracy: 0.9829 - 25s/epoch - 13ms/step
Epoch 14/20
1875/1875 - 25s - loss: 0.0083 - accuracy: 0.9992 - val_loss: 0.0531 - val_accuracy: 0.9831 - 25s/epoch - 13ms/step
Epoch 15/20
1875/1875 - 25s - loss: 0.0071 - accuracy: 0.9995 - val_loss: 0.0536 - val_accuracy: 0.9828 - 25s/epoch - 13ms/step
Epoch 16/20
1875/1875 - 25s - loss: 0.0063 - accuracy: 0.9996 - val_loss: 0.0546 - val_accuracy: 0.9827 - 25s/epoch - 13ms/step
Epoch 17/20
1875/1875 - 26s - loss: 0.0055 - accuracy: 0.9997 - val_loss: 0.0544 - val_accuracy: 0.9828 - 26s/epoch - 14ms/step
Epoch 18/20
1875/1875 - 26s - loss: 0.0049 - accuracy: 0.9998 - val_loss: 0.0531 - val_accuracy: 0.9840 - 26s/epoch - 14ms/step
Epoch 19/20
1875/1875 - 26s - loss: 0.0043 - accuracy: 0.9999 - val_loss: 0.0554 - val_accuracy: 0.9826 - 26s/epoch - 14ms/step
Epoch 20/20
1875/1875 - 25s - loss: 0.0040 - accuracy: 0.9999 - val_loss: 0.0546 - val_accuracy: 0.9828 - 25s/epoch - 13ms/step
```

実行結果

学習の**繰り返し**ごとに，
訓練データや検証データ
での**精度**や**損失**
の変化を確認

分類精度

1層目の**ユニット数**と**分類精度**の**関係**

20回の学習の繰り返しののち，**分類精度**を計測

分類精度は，検証データでの分類の**正解率**

1層目のユニット数	分類精度
400	**0.983**
4000	**0.983**
40000	**0.983**

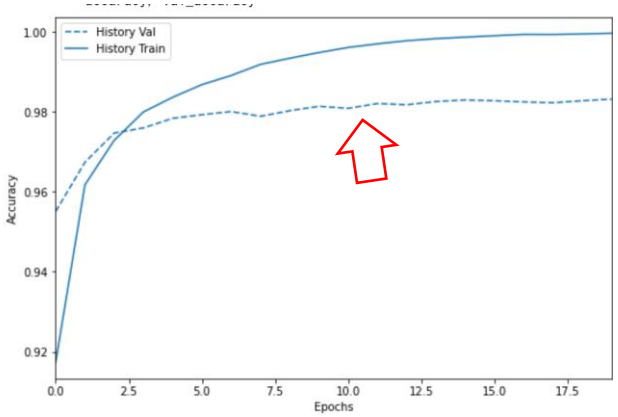
【考察】この場合，**ユニット数を変化**させても，
分類精度はほとんど**変化しない**.
実際のプログラム実行により**確認**.

学習曲線

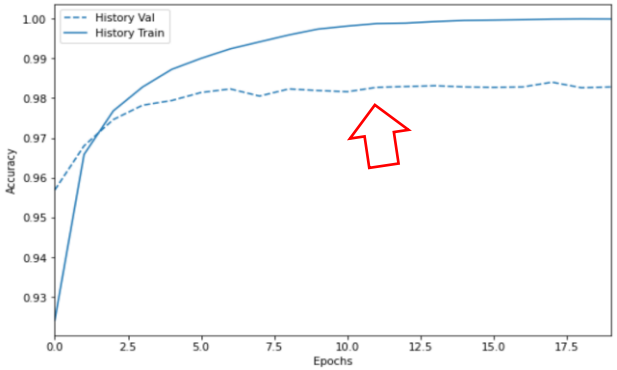
縦軸：分類精度，横軸：学習の繰り返し回数

実線：訓練データでの分類精度の変化

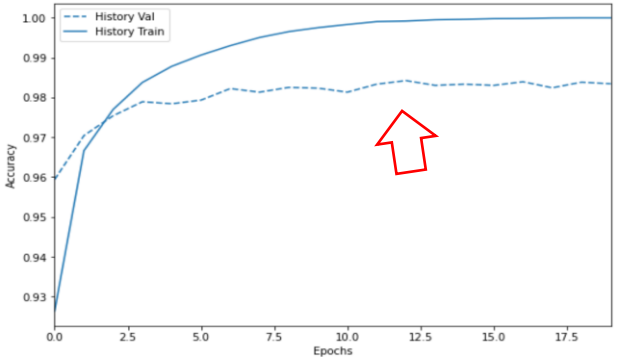
点線：検証データでの分類精度の変化



1層目のユニット数：**400**

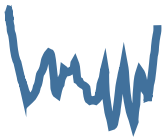


1層目のユニット数：**4000**

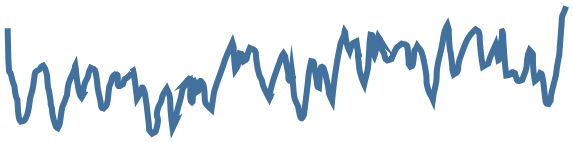


1層目のユニット数：**40000**

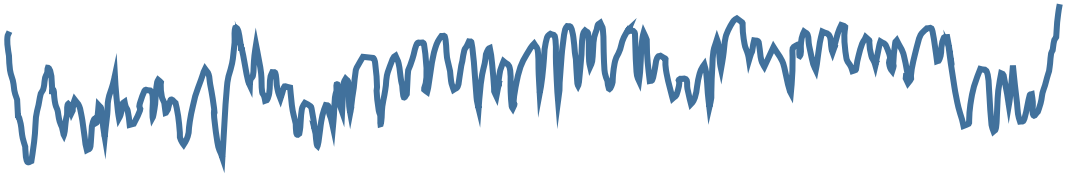
ユニット数 400



ユニット数 4000



ユニット数 40000



1 層目のユニット数が 10倍, 100倍

↓

結合の数が 10倍, 100倍

↓

探索空間の大きさが 10倍, 100倍

↓

精度の向上が約束されるものではない

コンピュータの動作は
遅くなる．過学習の
可能性が増える場合もある．

プログラムは、次で公開

https://colab.research.google.com/drive/1-UWl-WEPmmNo-S_O17E5XPkF4tphE6xz?usp=sharing

- **プログラムの再実行**，**プログラムの変更**には，**Google アカウント**が必要.
- プログラムを変更した場合でも，特別な操作をしない限り，他の人には公開されない

7-6 別の学習曲線の例

精度向上等の改良の試み

- **ドロップアウト**
- **結合の重みの正則化**

L1，L2，Elastic Net など

- **最適化手法**のバリエーション

Adadelta, Adagrad, Adam, RMSprop, SGD など

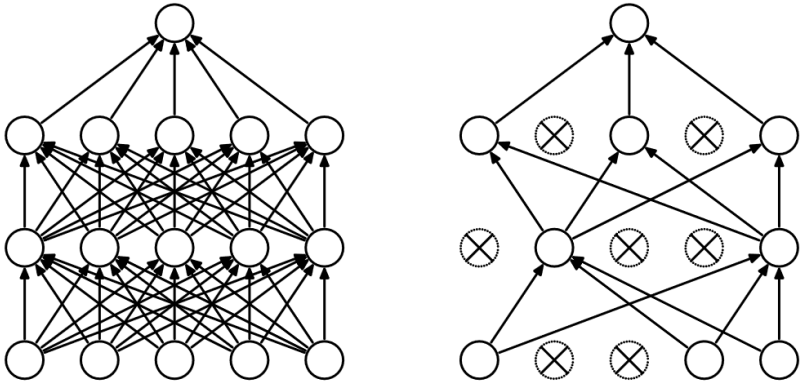
- **誤差である損失の算出法（損失関数）**のバリエーション

binary_crossentropy, categorical_crossentropy,
cosine_similarity, kld, kullback_leibler_divergence, mae など

- **学習の高速化**

バッチ など

ドロップアウト



(a) Standard Neural Net　　(b) After applying dropout.

ドロップアウト：
過学習の解決のため，**学習時に，ユニットをランダム**に選び，
存在しないことにする．(2014年発表)

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. ***Dropout: A Simple Way to Prevent Neural Networks from Overfitting.*** *The Journal of Machine Learning Research, Volume 15 Issue 1, January 2014 Pages 1929-1958*
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

手順

① パソコンの Web ブラウザで，次のページを開く

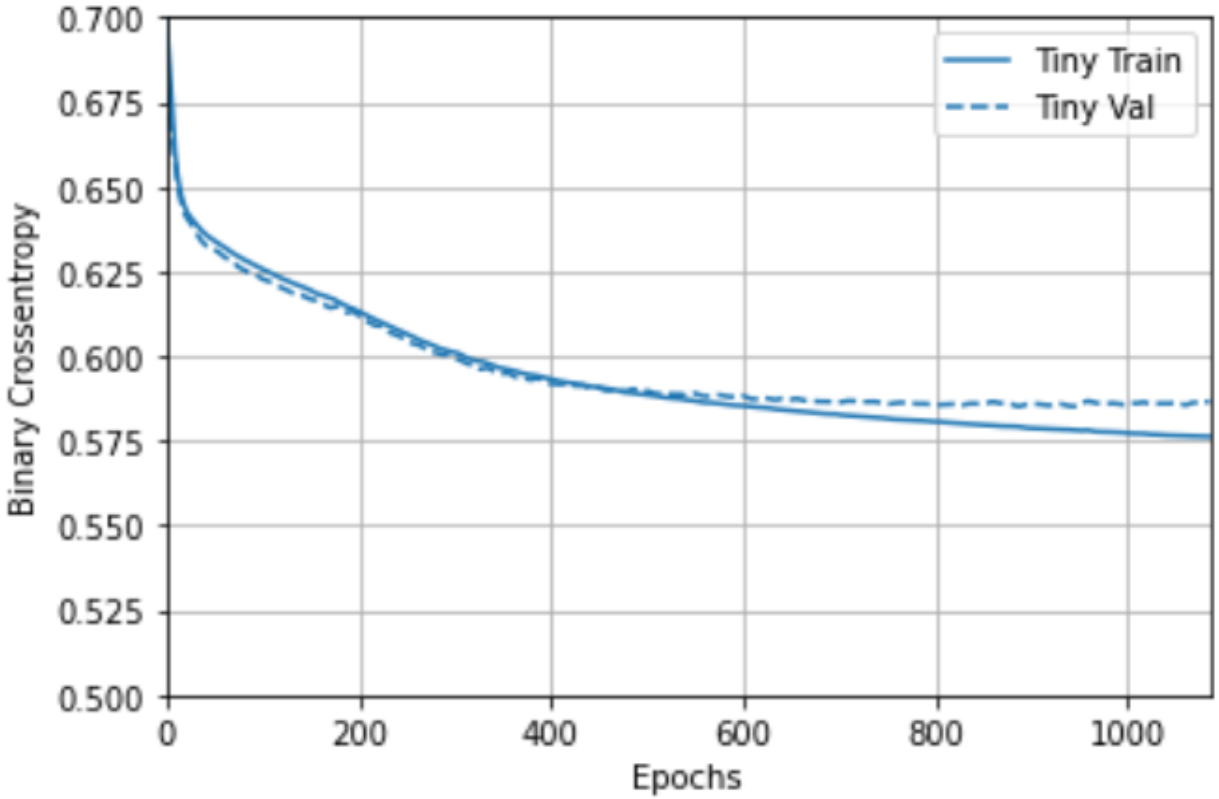
https://www.tensorflow.org/tutorials

② 左側のメニューの「**Keras による ML の基本**」を**展開**，「**オーバーフィットとアンダーフィット**」をクリック

③ 記載の説明等をよく読み、理解を深める

見どころ①　学習曲線

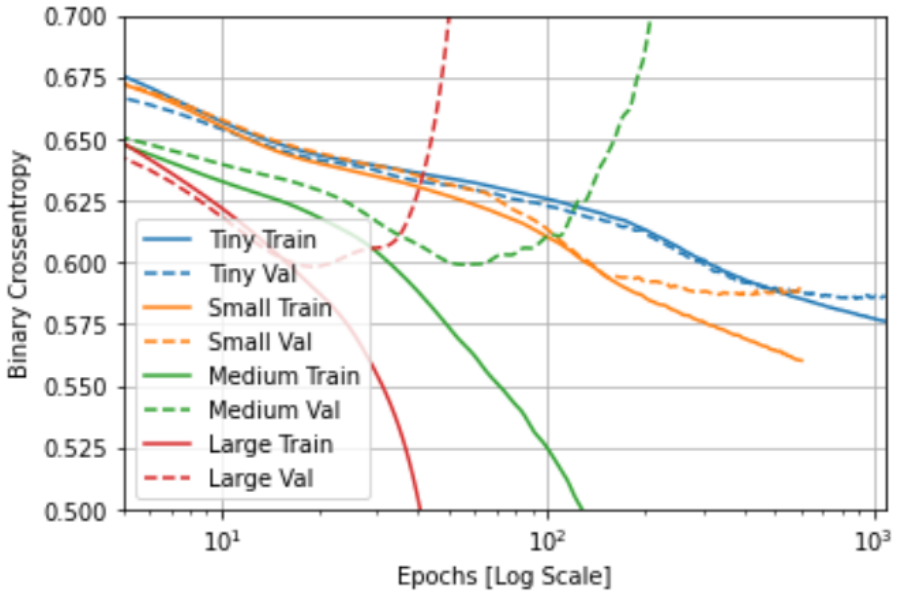
学習曲線は、訓練データと検証データの**誤差の推移**を表したグラフ



Train （実線）は訓練データでの誤差。Val（破線）は検証データでの誤差

見どころ②　学習曲線

ニューラルネットワークの**層やユニット数を増やす**ことが必ずしも良い結果をもたらすわけでない

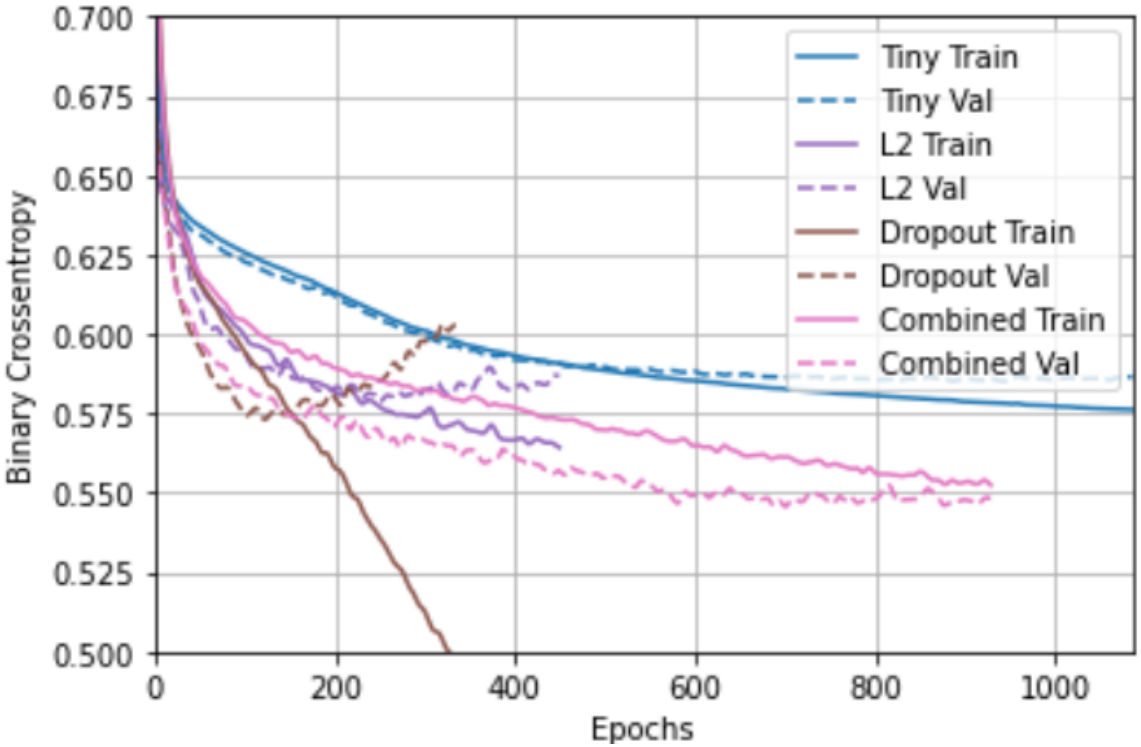


Medium, Large では
過学習が発生

Train （実線）は訓練データでの誤差。Val（破線）は検証データでの誤差
４種類のニューラルネットワーク
Tiny: １層目のユニット数：１６
Small: １層目のユニット数：１６，２層目のユニット数：１６
Medium: １層目のユニット数：６４，２層目のユニット数：６４，３層目のユニット数：６４
Large: １層目のユニット数：５１２，２層目のユニット数：５１２，３層目のユニット数：５１２，４層目のユニット数：５１２

見どころ③　過学習を防止する技術

過学習が起きていた「Large」について、L2正則化、ドロップアウトの技術をつってみる



青：Tiny

紫：Large + L2正則化
茶：Large + ドロップアウト
赤紫：Large + L2 正則化 + ドロップアウト

Train （実線）は訓練データでの誤差。Val（破線）は検証データでの誤差
４種類のニューラルネットワーク

全体まとめ

- **機械学習**はデータを利用して知的能力を向上させる手法であり、**汎化能力**を持つ。
- **汎化能力**は**訓練データに基づいて学習**し、**未知のデータも適切に処理できる能力**を指す。
- **過学習**は**訓練データに過剰に適合**し、未知の特徴やパターンに**誤った結果**を出す。
- **過学習**の**防止**は**困難。検証データを使用して確認する**ことが重要。
- **学習曲線**は**訓練データと検証データの誤差の推移**を表し、**学習不足や過学習を確認**するために使用される。
- **大量のデータの利用**と，**検証データ**と**学習曲線**を利用して学習不足や過学習がないことを確認することが成功の鍵である。

① 機械学習の**高度な技術**を知り、その有用性を知る。あわせて、**過学習や学習不足など**の、機械学習の限界、課題を知る。こうした、体系的理解により、**AI技術を知る満足感**が深まる。

② 今後、卒業研究や就職後、**機械学習の効果的に活用できるようになる。大きな成長**である。

③ AIの応用、未来の可能性について、**広い視野で考察**できるようになる。

④ **汎化、学習曲線という機械学習の知識**は、AIを活用する様々な職種で重要な基礎となり、**将来にわたり有用**。