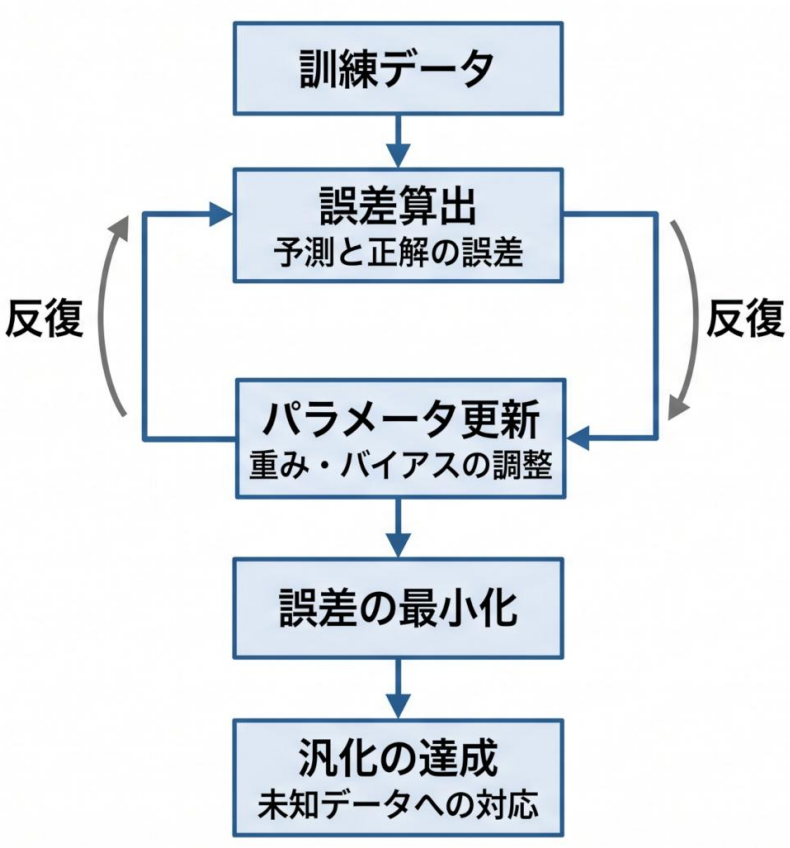
aa-6. 学習と検証、学習不足、過学習、学習曲線

（人工知能）

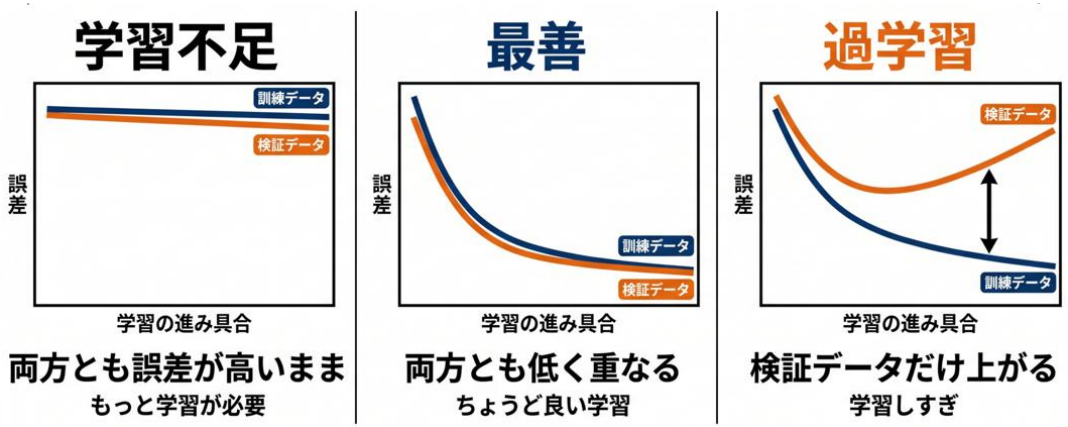
金子邦彦

内容

学習プロセス



学習曲線の活用



過学習を防ぐ方法

- データの増量・多様性向上
- ネットワークの簡素化

6-1. 画像分類とニューラルネットワーク

画像分類

（例）画像を、０，１，２，３，４，５，６，７，８，９の**１０種類に分類**



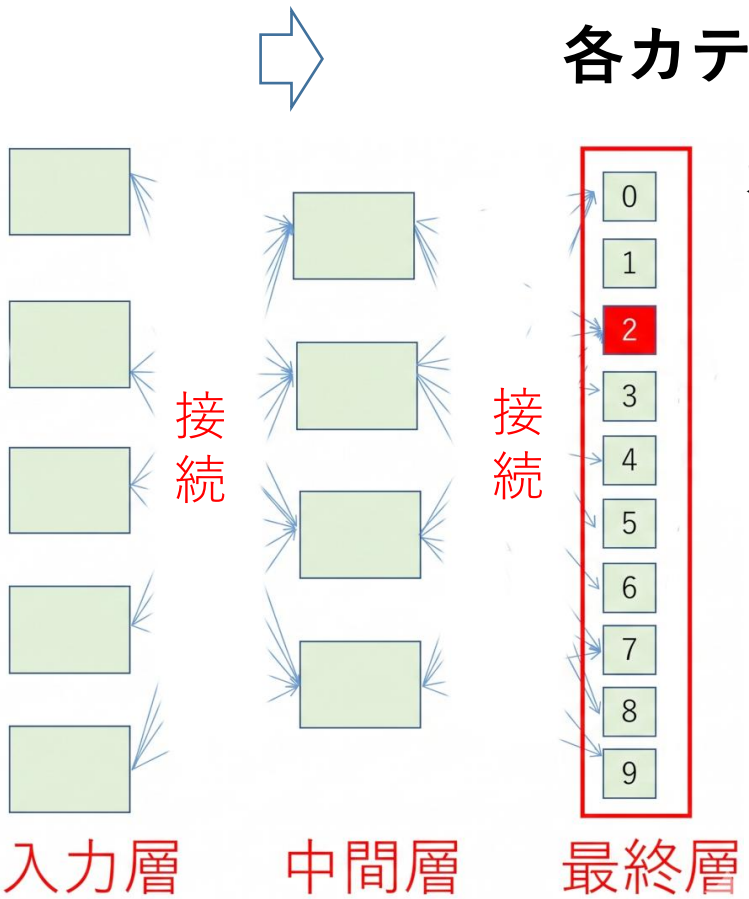

2　4

0　8

AI での画像分類

- 画像を入力
- カテゴリそれぞれの確率が出力される
- 最も活性度が高いニューロンに対応するクラスが分類結果となる。

画像

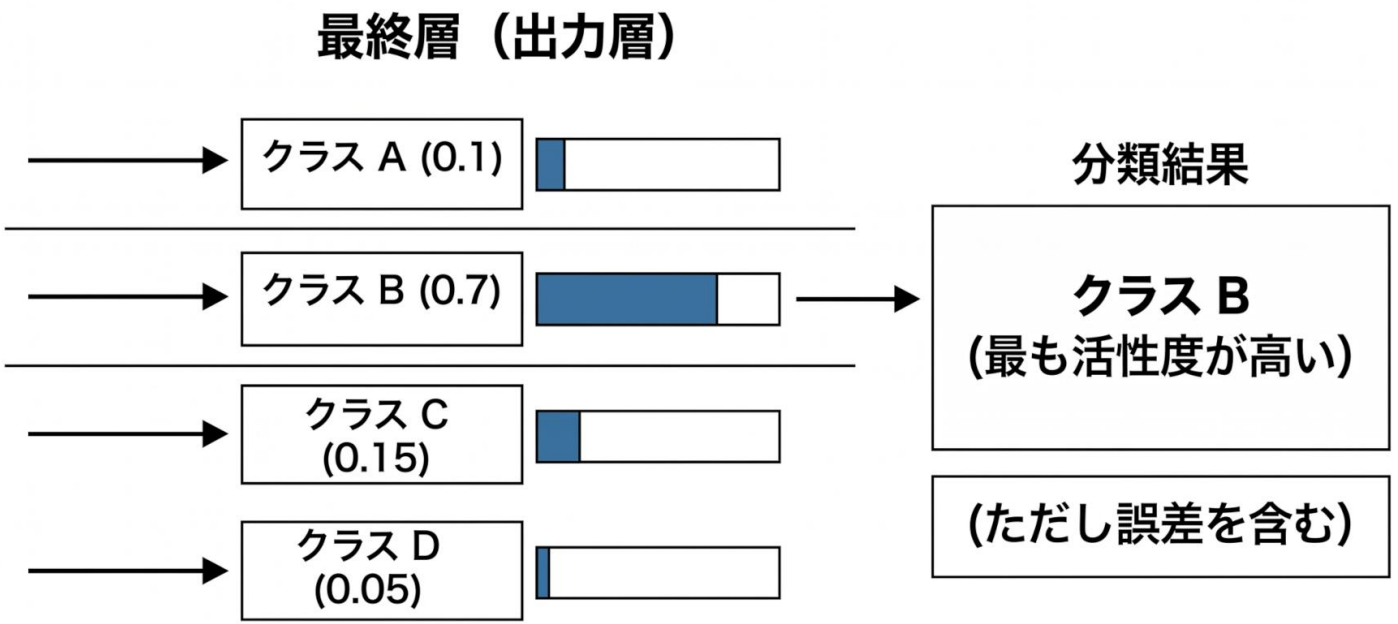


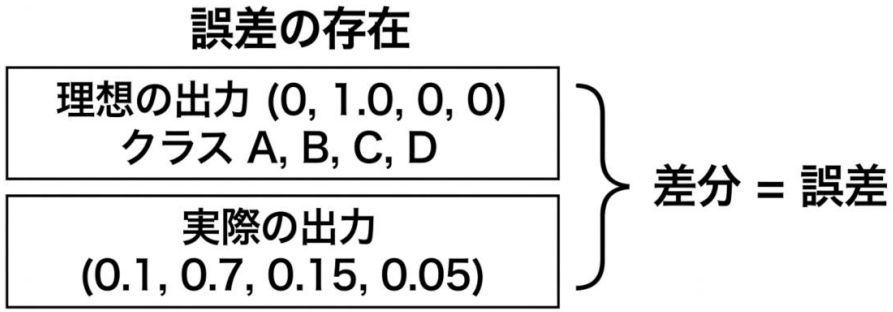
各カテゴリ（ラベル）の確率

カテゴリ（ラベル）		
0	確率	0.0000・・・
1	確率	0.0000・・・
2	確率	0.9999・・・
3	確率	0.0000・・・
4	確率	0.0000・・・
5	確率	0.0000・・・
6	確率	0.0000・・・
7	確率	0.0000・・・
8	確率	0.0000・・・
9	確率	0.0000・・・

ニューラルネットワークを用いた分類と誤差

最終層のニューロンのうち**最も活性度が高いもの**に**対応するクラス**が**分類結果**となる。分類結果には誤差がある。

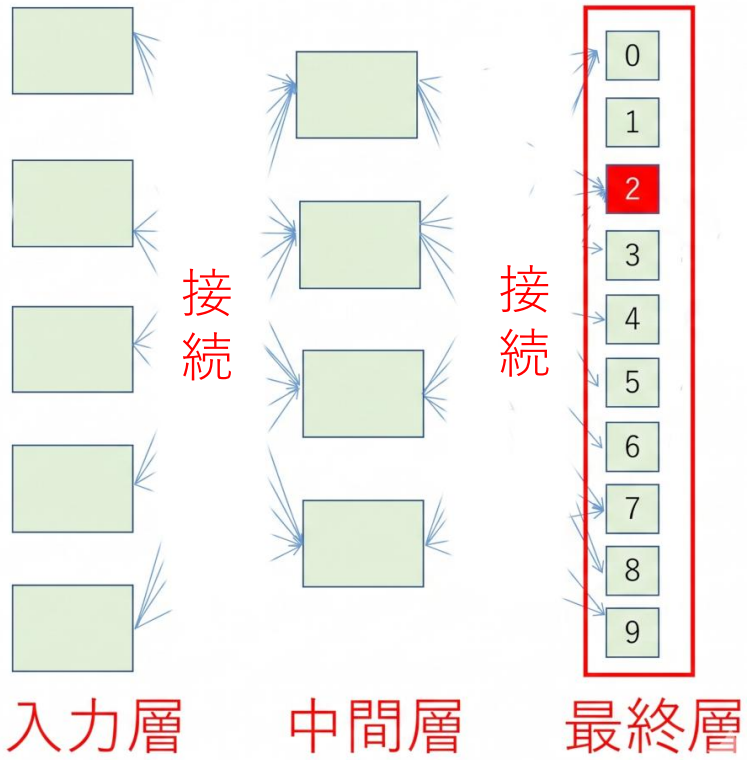




ニューラルネットワークの構造

- **入力層**：28×28 の画像を受け取る。ニューロン数は 768個
- **中間層**：特徴抽出（パターン認識）を行う
- **出力層**（ニューロン数 10）：特徴抽出（パターン認識）と分類

28×28
768画素



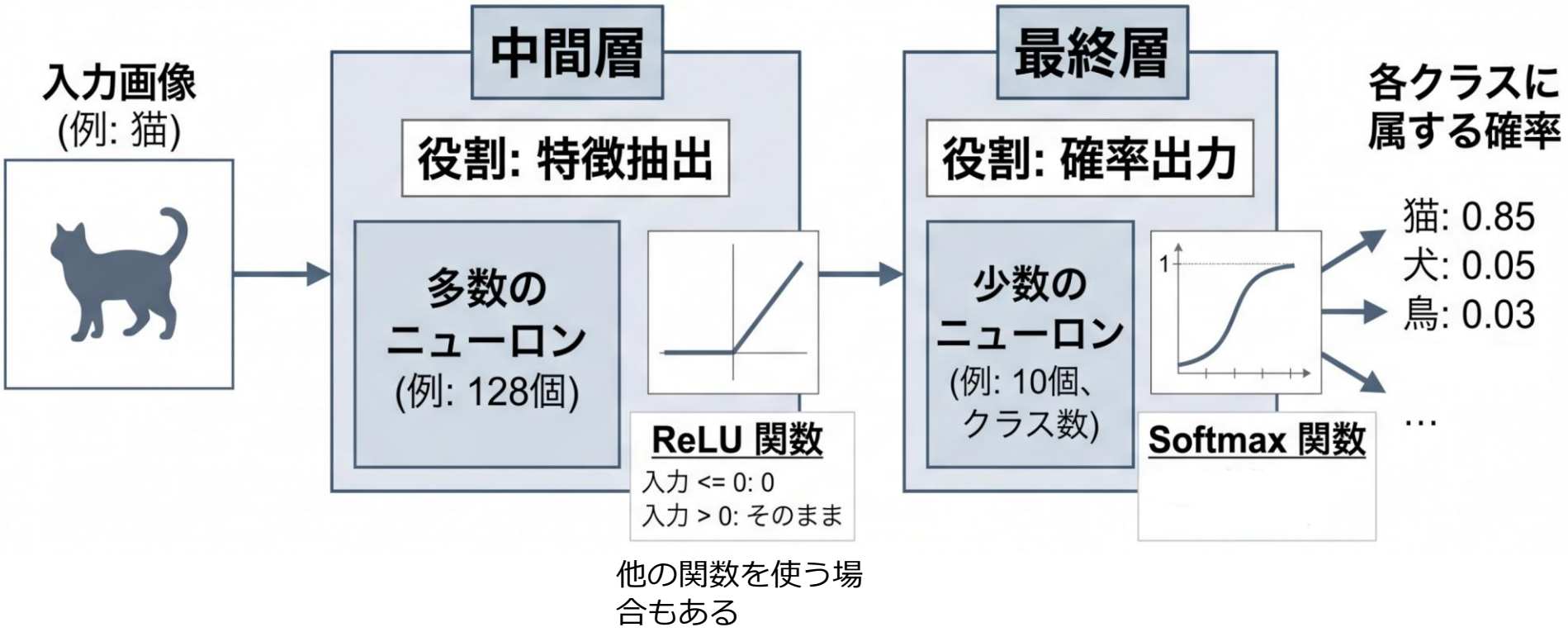
全体で 768個のニューロン

種類: relu

ニューロン数: 10
種類: softmax

活性化関数の使い分け

中間層と最終層で異なる種類の関数を用いる
（入力層はデータの受け取りのみで、もともと活性化関数がない）



Python プログラムでの実装例 (Kerasを使用)

入力は **28×28**個の画素

１層目のニューロン数は **128**
活性化関数は **relu**

```
import tensorflow as tf
m = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

２層目のニューロン数は **10**
種類は **softmax**

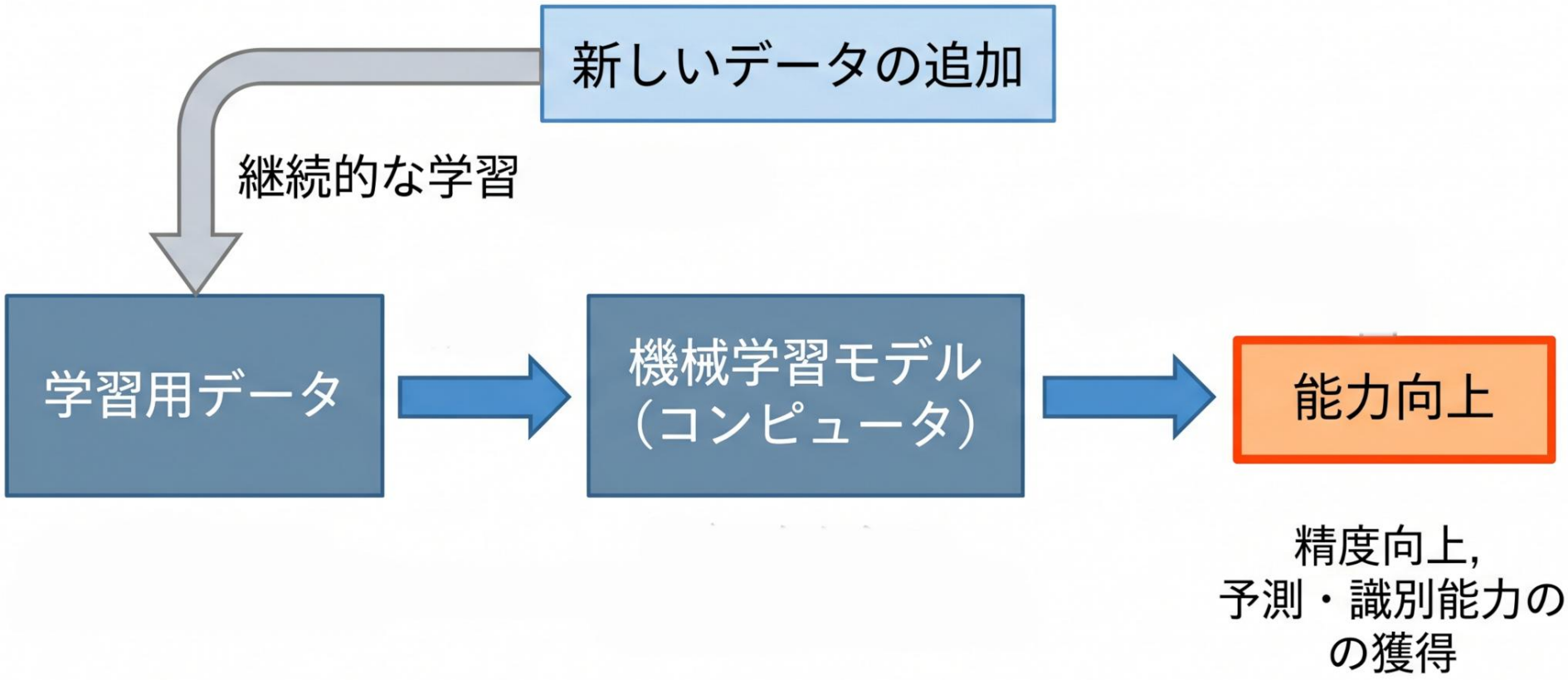
Flatten は，2次元の配列を，
ニューラルネットワークの
入力形式に変換するもの

設定すべき項目

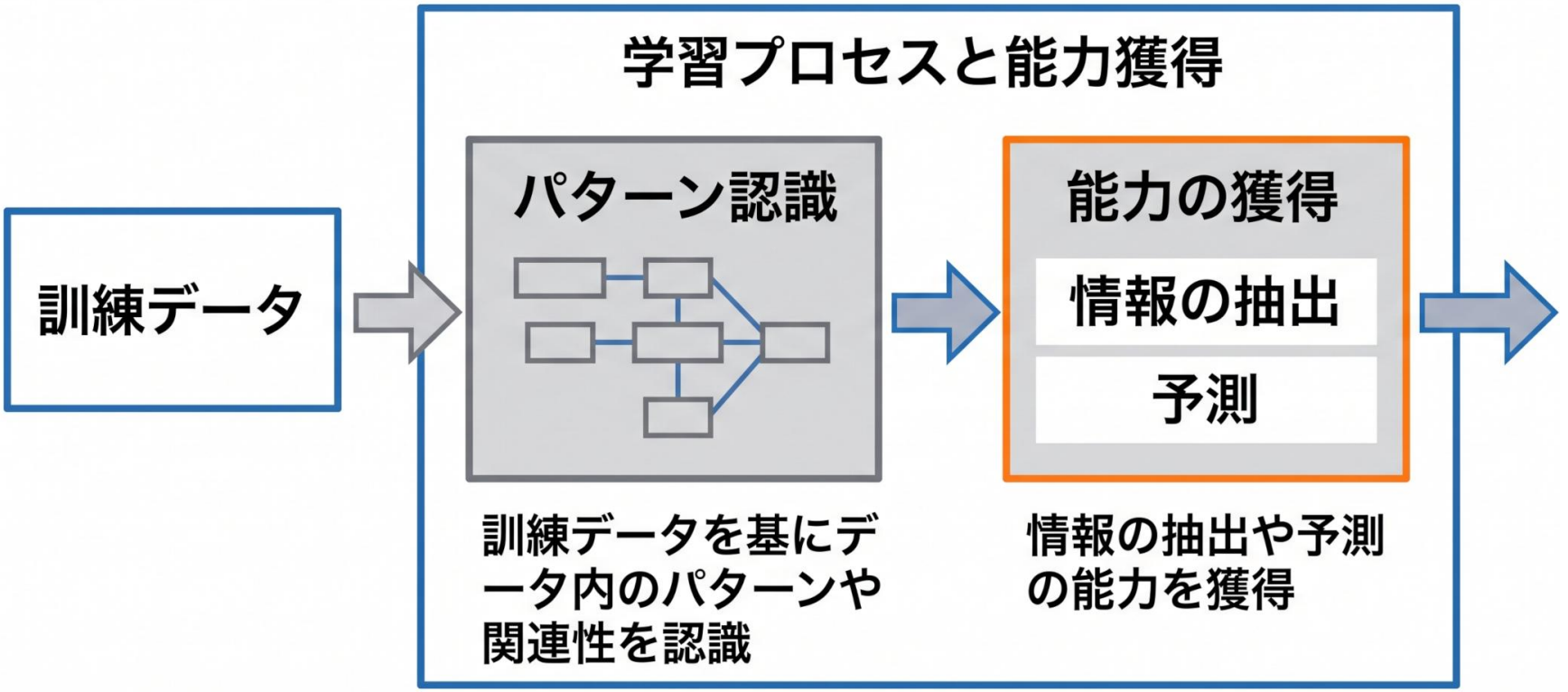
- 入力の**数値の個数（ここでは画素数）**
- **層ごとのニューロン数**
- **層ごとの活性関数の種類**

6-2. 機械学習の特徴

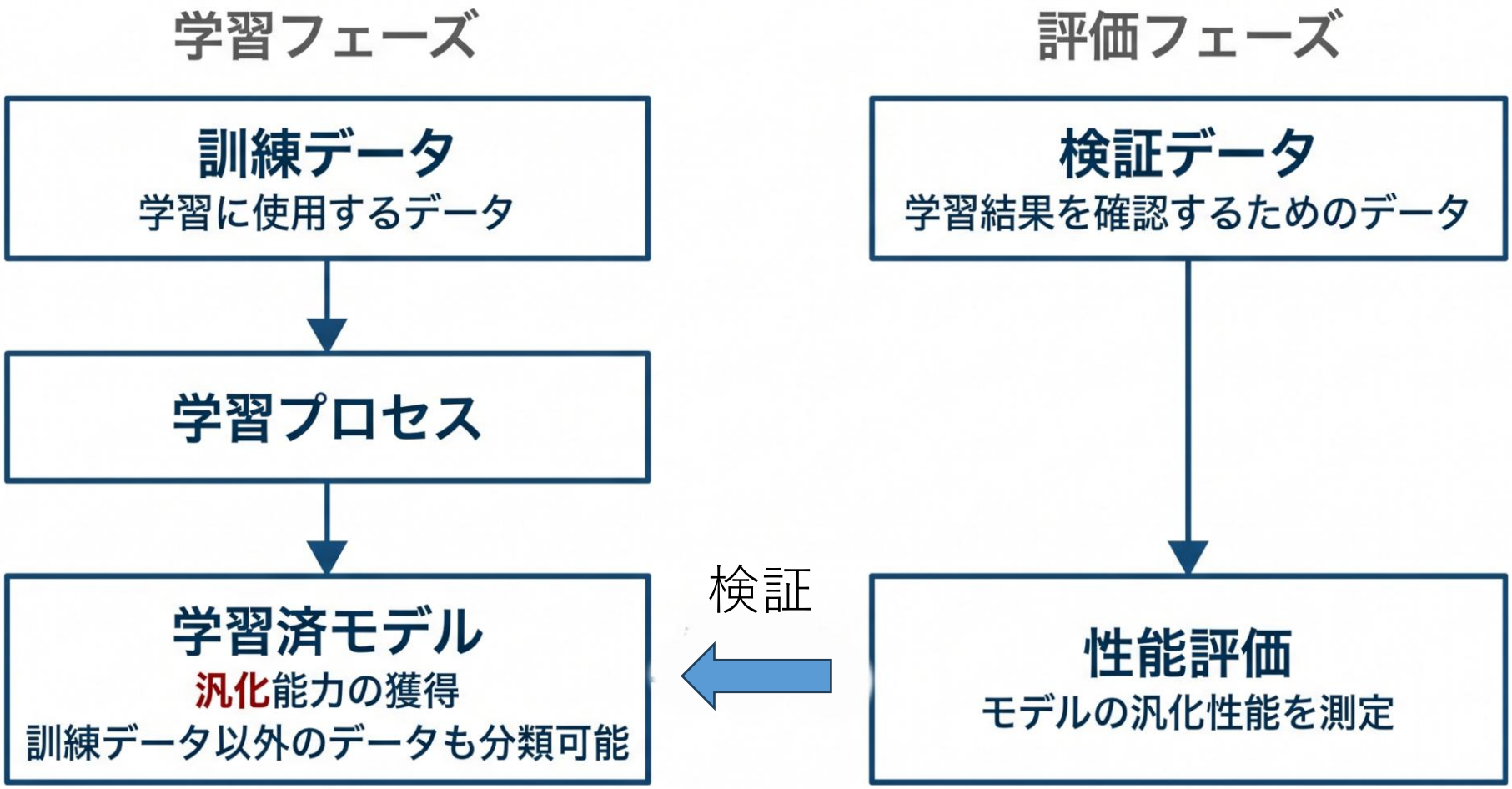
機械学習の特徴　①データ駆動型学習



機械学習の特徴　②パターン認識と情報処理



機械学習の特徴　③学習と検証



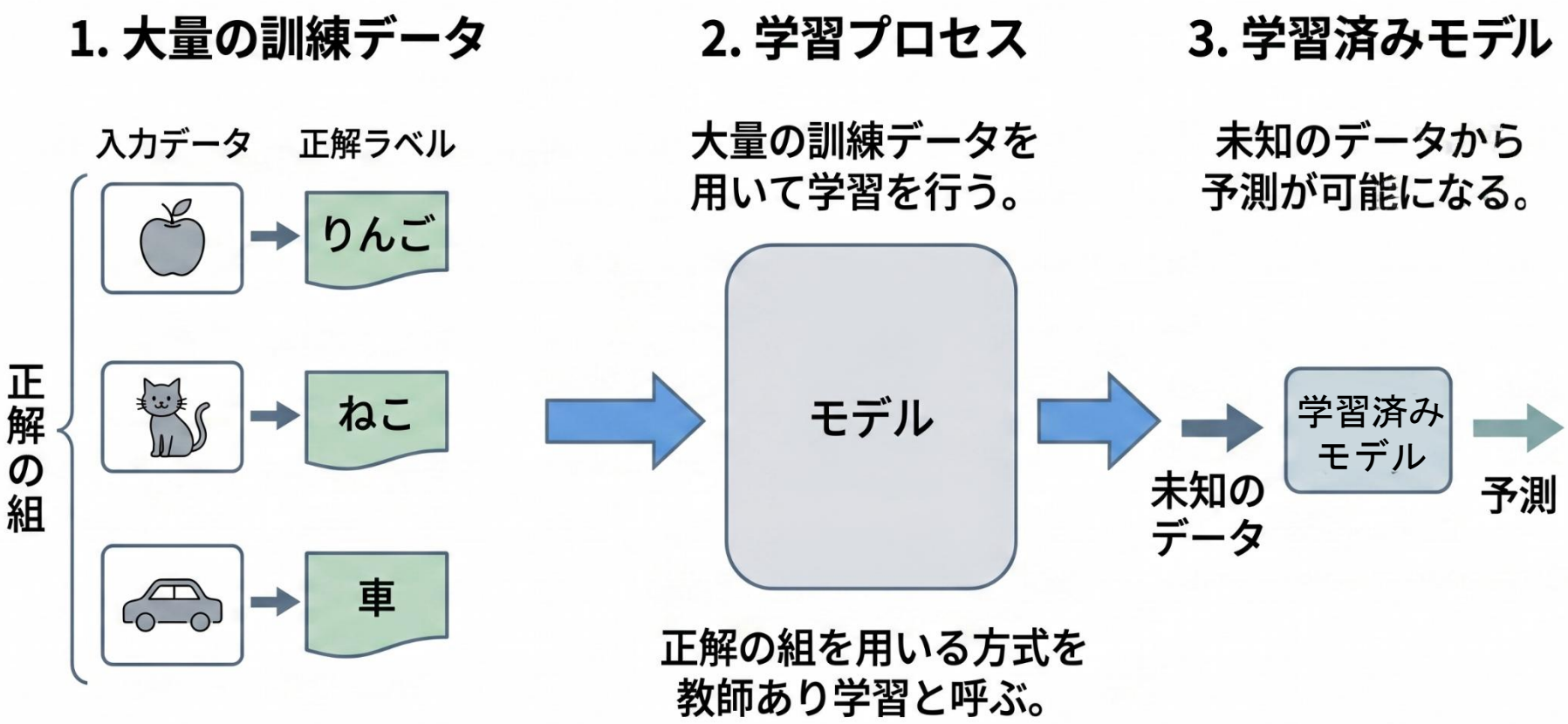
汎化：訓練データ以外も
分類・予測などできる能力

訓練データと検証データは異なるデータを使用する

6-3. ニューラルネットワークの学習プロセス

教師あり学習の仕組み

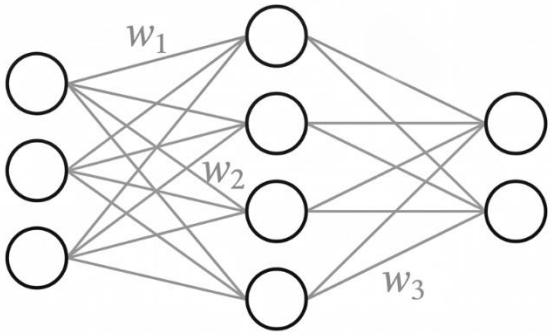
正解の組を用いる方式を **教師あり学習**と呼ぶ.
（このほかに教師なし学習や強化学習といった方式もある.）



ニューラルネットワークの学習における最適化

ニューラルネットワークの重みとバイアスを調整し，損失を最小化するプロセスである

学習前

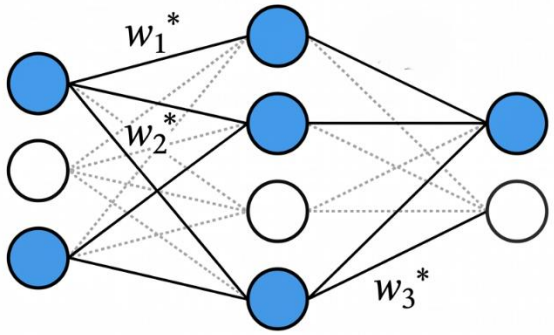


予測値：不正確

学習による最適化

重み w とバイアス b を調整

学習後

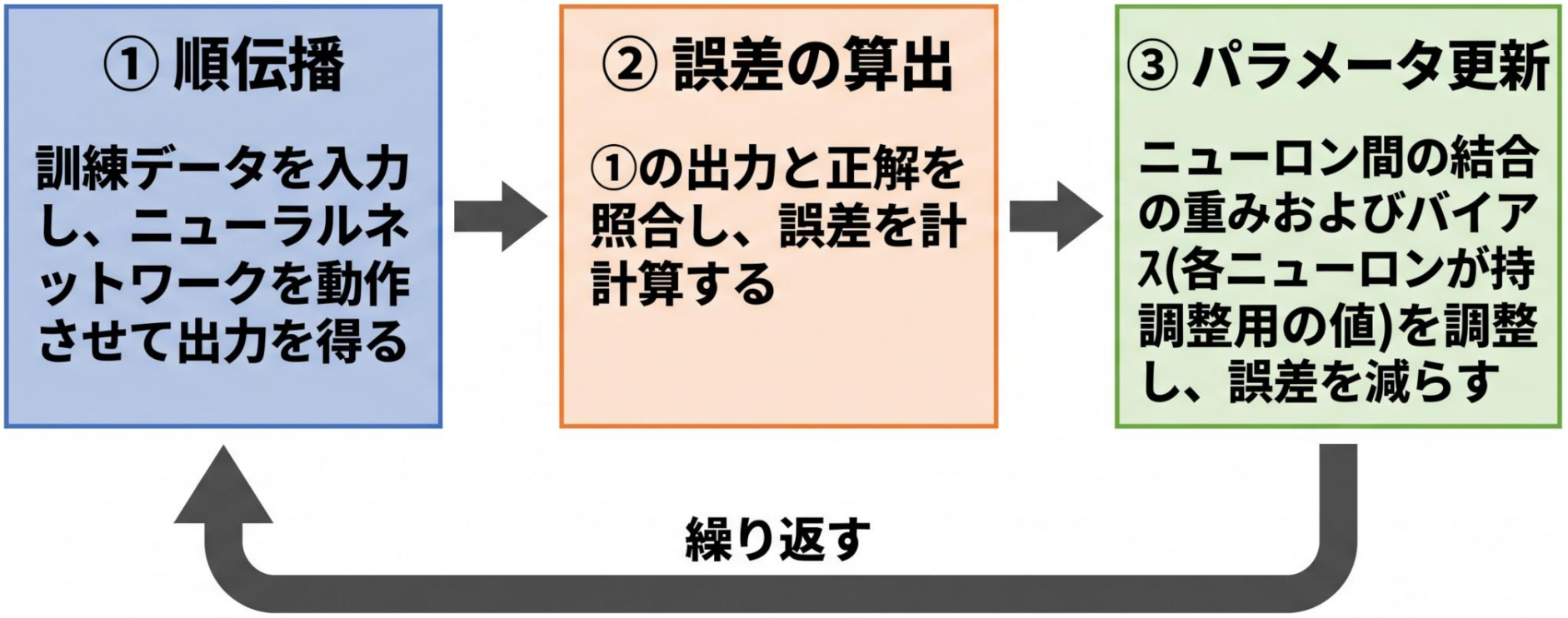


予測値：正確

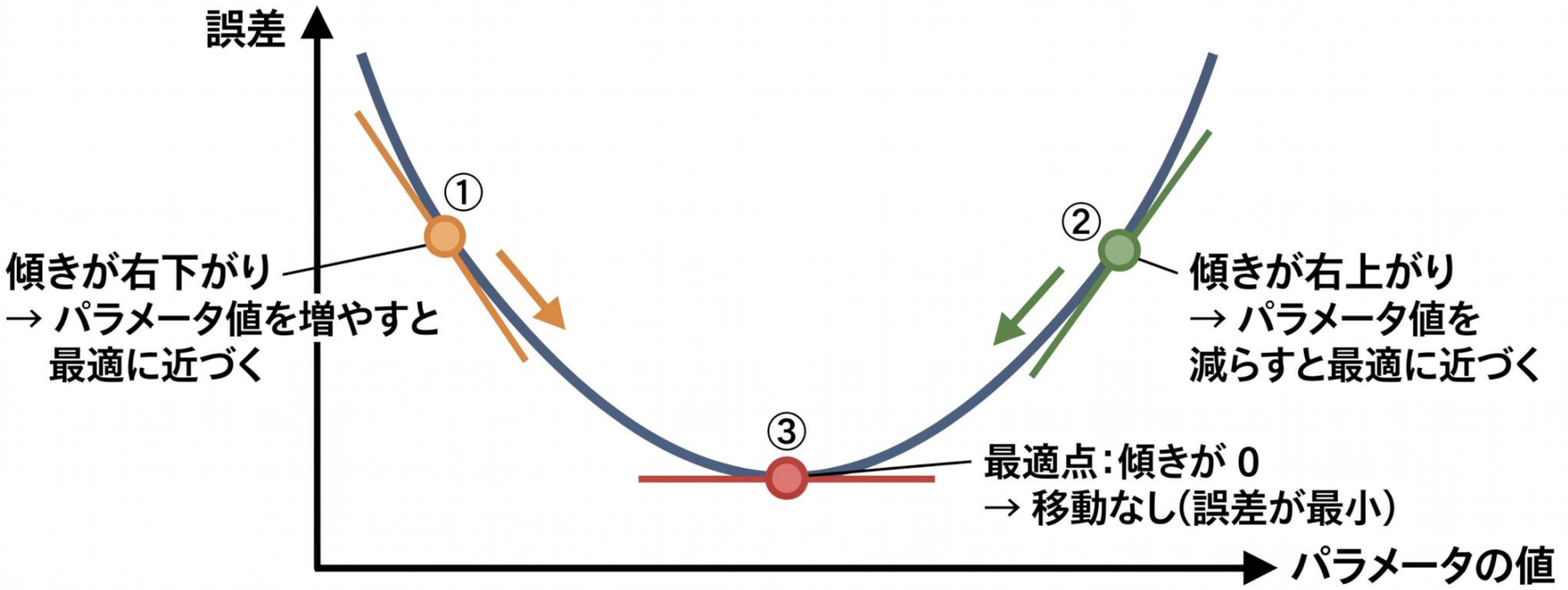
パラメータ	目的関数（損失）	最適化
重み w バイアス b	予測値と正解との誤差	w と b の値を調整し L が最小になる組合せを探す
	$L =$ 誤差の合計	最適な w, b が求まる

最適化は，ニューラルネットワーク学習の中核をなすプロセスである

機械学習の学習３ステップ



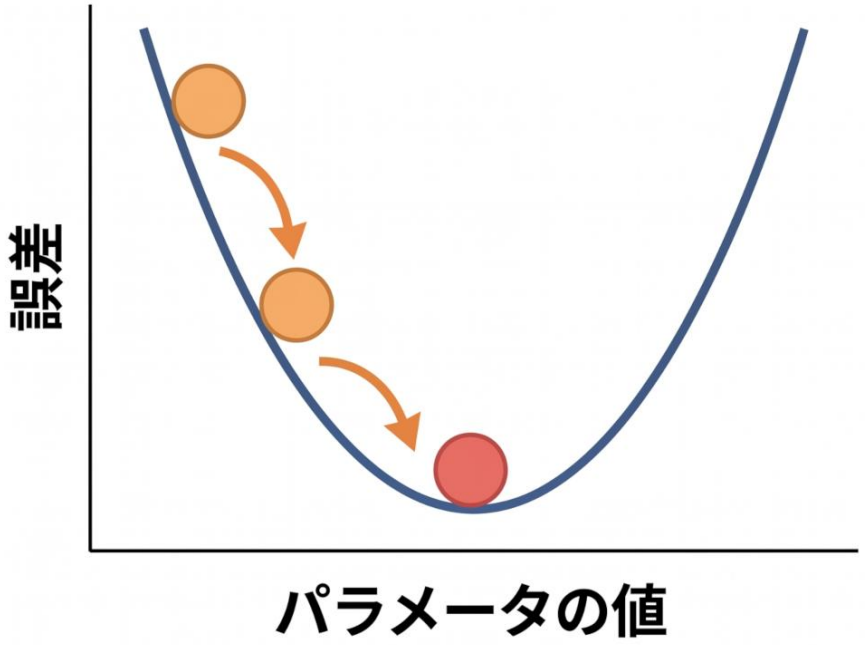
勾配降下のイメージ　坂道を降りるように誤差を減らす



誤差を減らす過程は、坂道を降りる動作にたとえられる。傾き(パラメータに対する誤差の変化方向)を手がかりに、誤差が最小となる点を探索する。

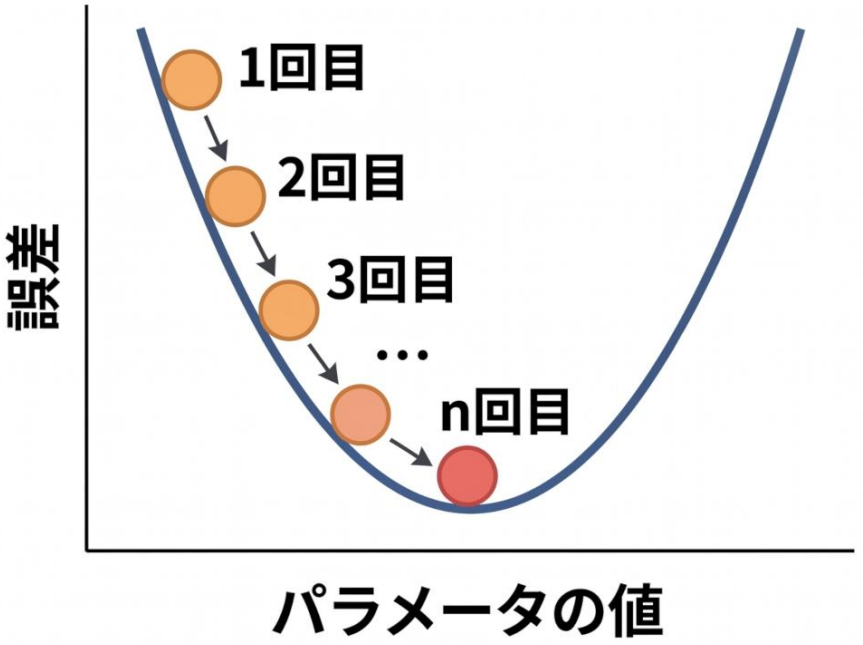
学習の繰り返しのイメージ

勾配降下のイメージ



誤差を減らす過程は、坂道を降りる動作にたとえられる

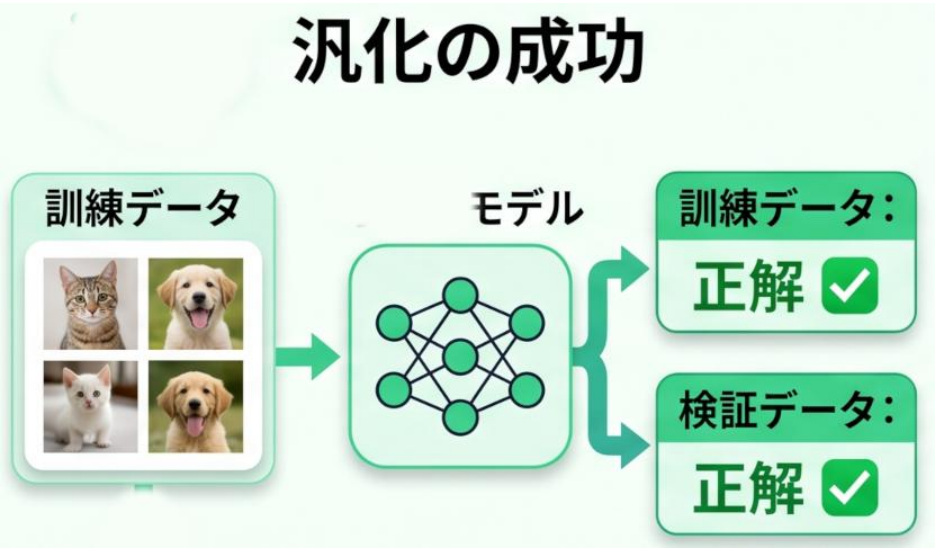
学習の繰り返し

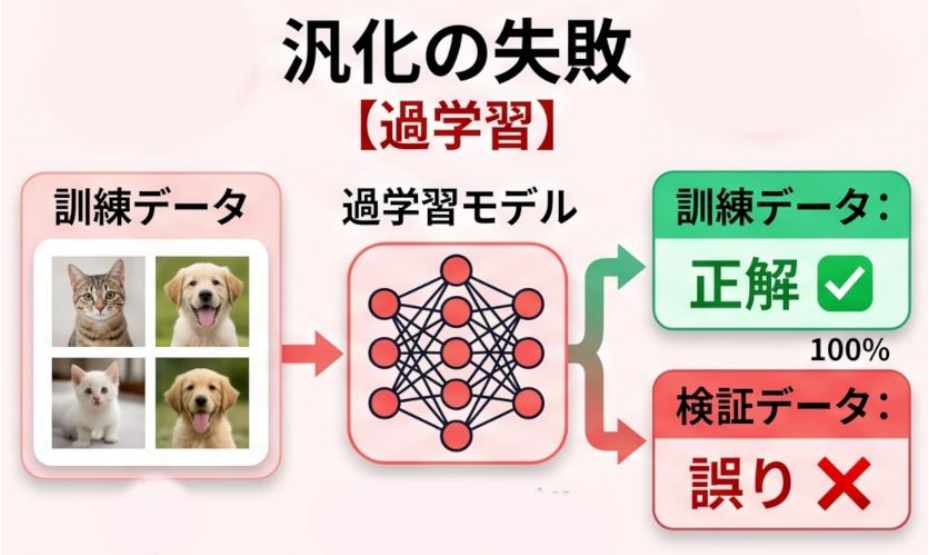


同じ訓練データを繰り返し使用することで、誤差をさらに減らす

6-4. 汎化・過学習

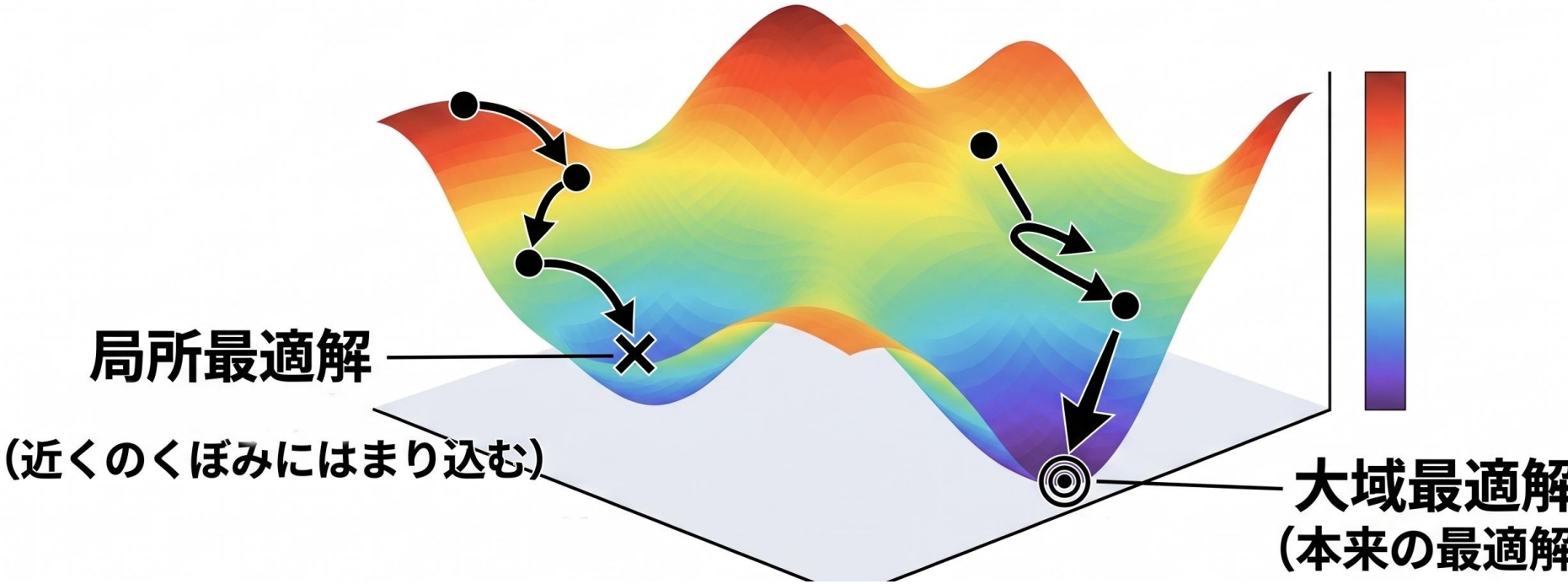
汎化の成功と失敗





- **汎化の成功**：訓練データに対して正しい結果を出し、**検証データに対しても正しい結果を出す**
- **汎化の失敗**：訓練データに対しては正しい結果を出すが、**検証データに対して誤った結果を出す**

問題①　見せかけの正解



勾配降下法は、坂を下るようにして、最適解を探す。しかし、浅いくぼみ（局所最適解）にはまり最善の解（大域最適解）を見逃す場合がある

問題②　組み合わせ爆発

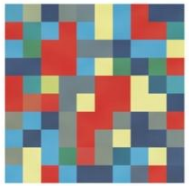
課題

入力データの組み合わせは膨大。
訓練データはそのごく一部しか
カバーできない。汎化の失敗(過学習)
は過去の最大の難問だった。

全組み合わせ

●
訓練データ

例1：カラー画像 32×32



画素数 32×32×3
階調 256

組み合わせ
≒10の7397乗
（0が7397個続く数）

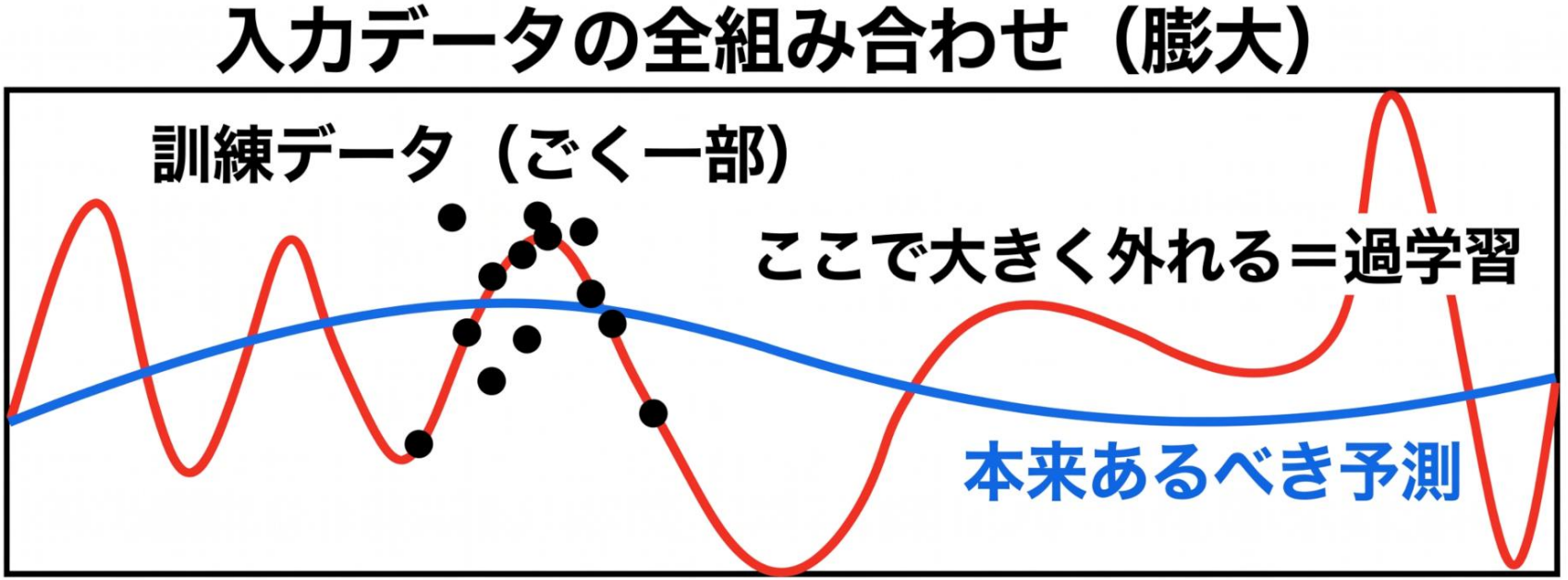
例2：ステレオ音声 10秒

44100Hz、階調 256

組み合わせ
≒10の1062033乗
（0が1,062,033個続く数）

→ ニューラルネットワークのニューロン数や層の数を増やすことは現実的ではないと考えられていた

問題②　組み合わせ爆発と過学習

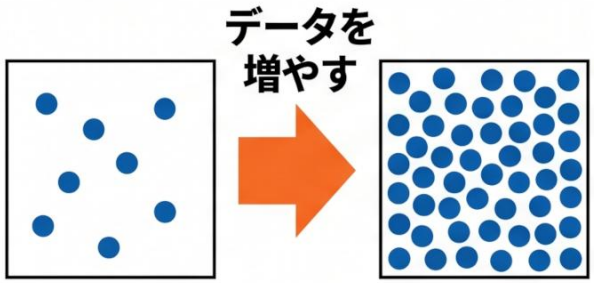


入力の組み合わせは膨大で、訓練データはその一部しか覆えない。点に合わせ込みすぎると、データの無い広い領域で予測が大きく外れる。これが過学習

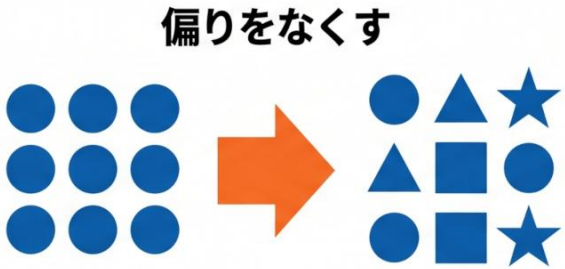
過学習を防ぐために

Database Lab.

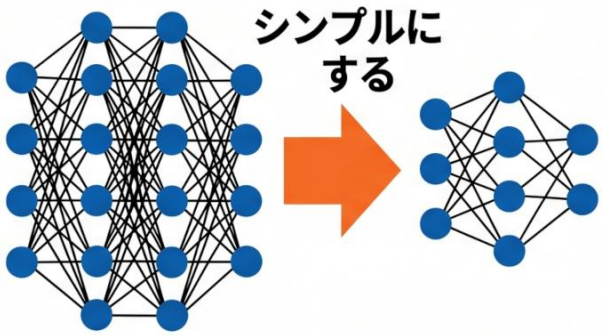
①大量の訓練データの確保



②多様性と品質の向上



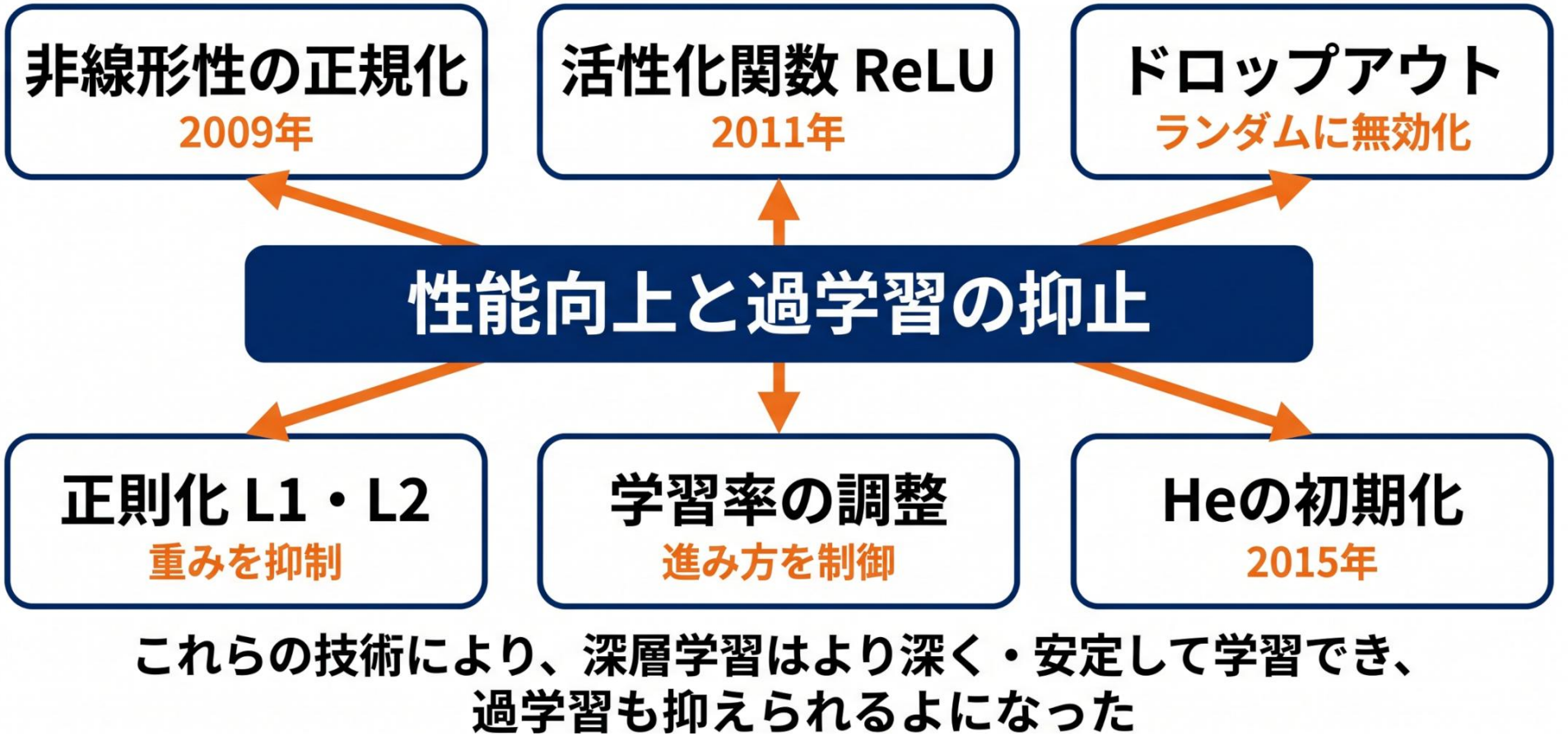
③ネットワークの簡素化



層やニューロンを増やせば良いとは限らない

過学習を防ぐには、データを増やす・データの偏りをなくす・モデルを複雑にしすぎない、の3つが基本

過学習を抑える新技術

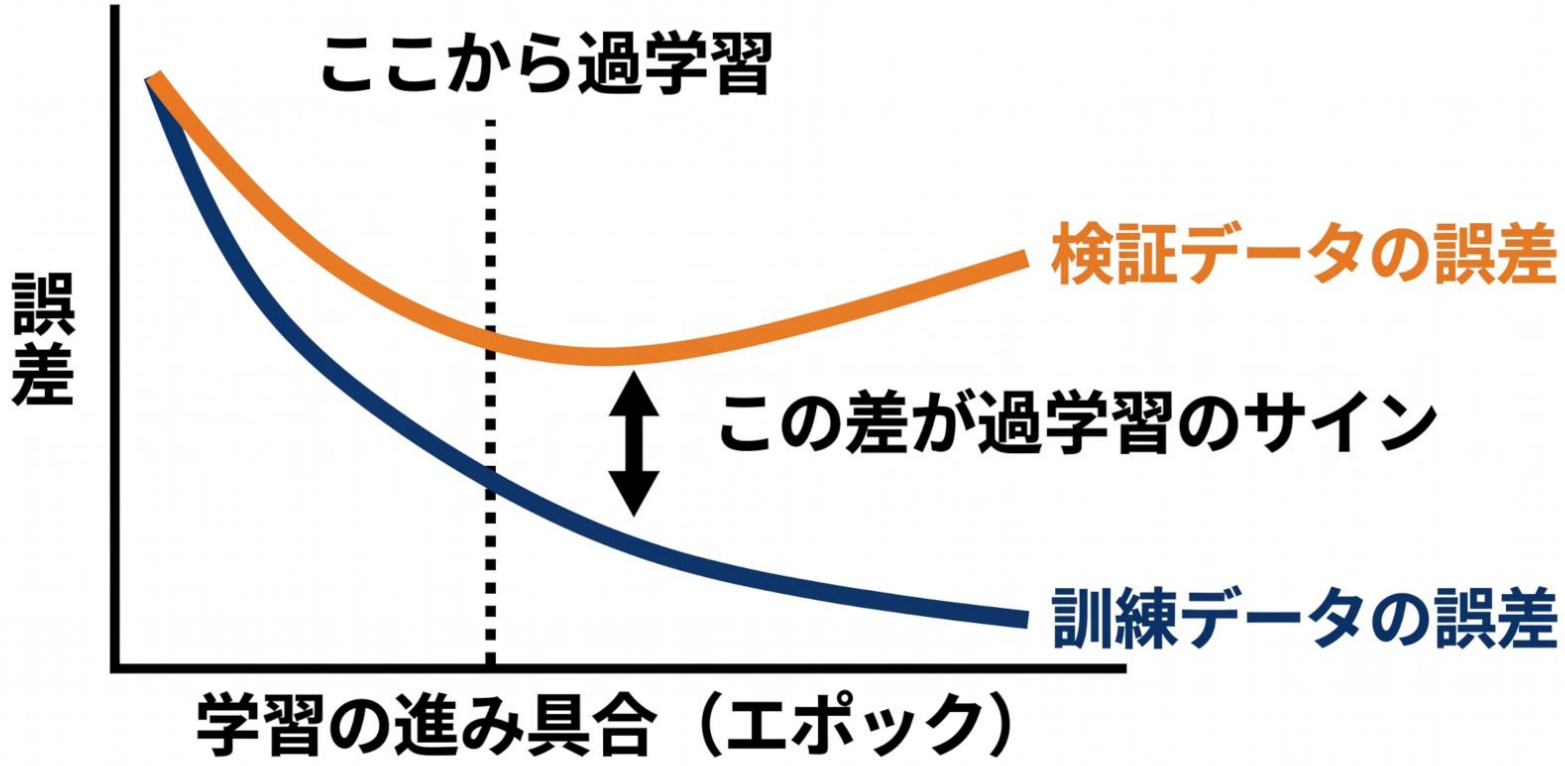


※ 多様な技術を存在を説明（詳細は説明しない）。現在のプログラミングでは、これらが自動、もしくは簡単な設定で使用できる

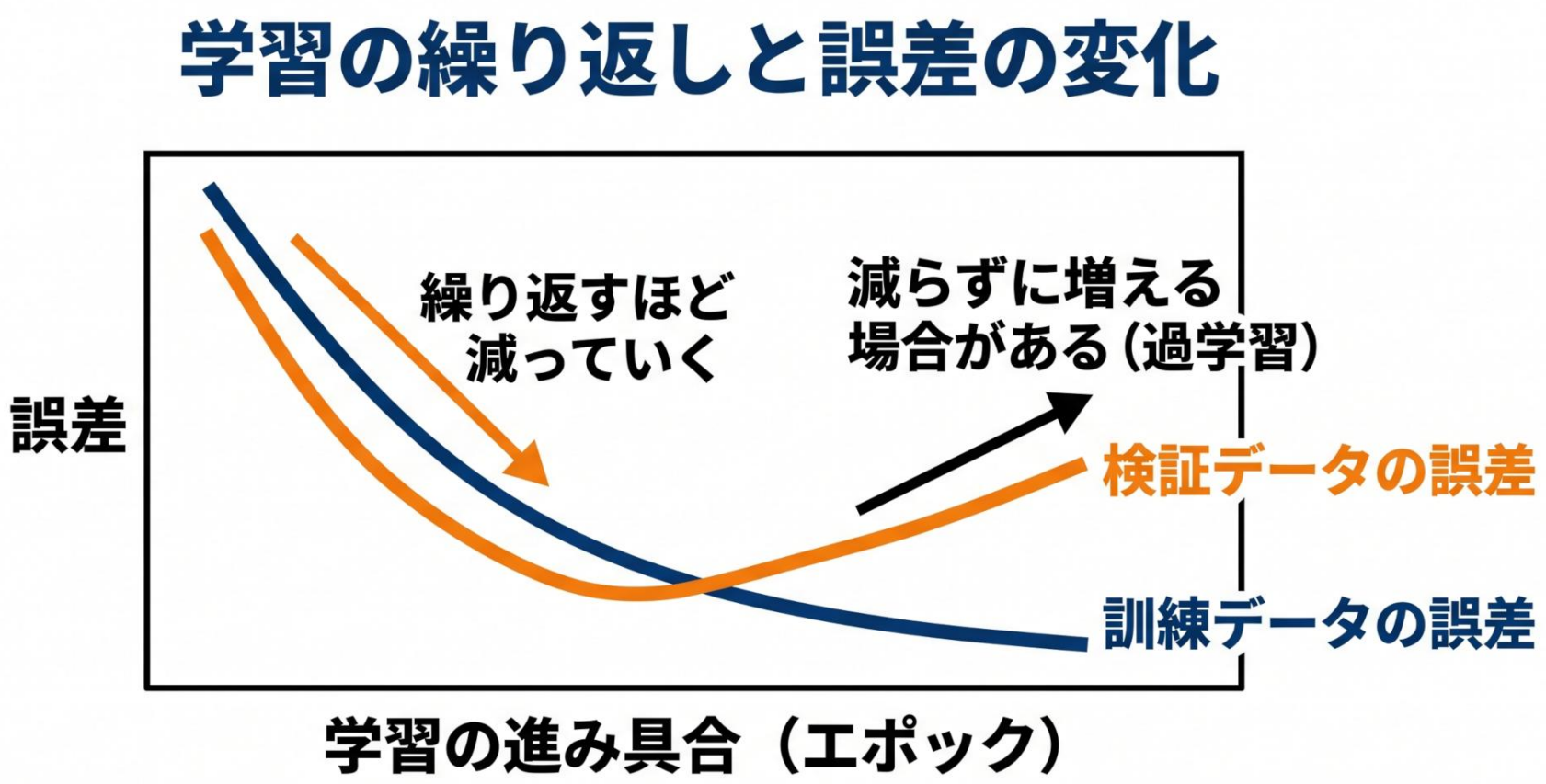
6-5. 学習曲線

検証のための学習曲線

検証データを用いて、モデルの汎化性能を測定。学習が成功したかを検証。



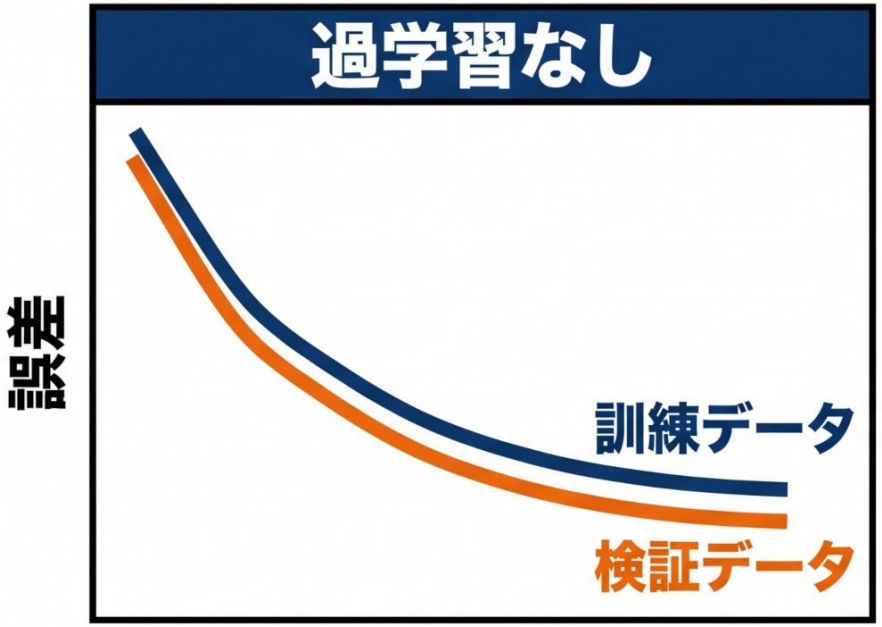
学習曲線は訓練データと検証データの誤差の推移を示すグラフ。
2本の差が開き始めた時点が過学習の始まり

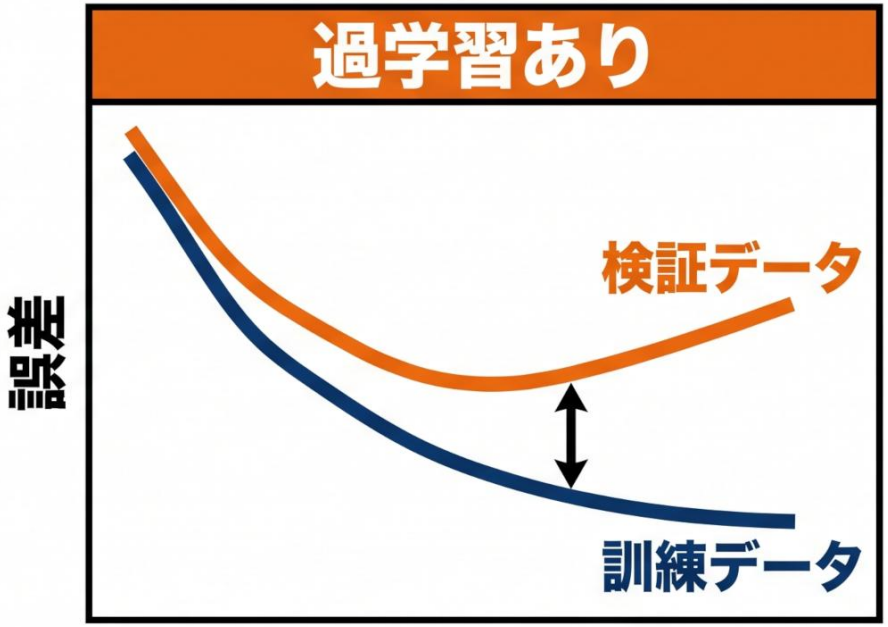


訓練データを繰り返し使うほど訓練データの誤差は減るが、
繰り返しが多すぎると検証データの誤差は逆に増えることがある

過学習なしのパターンと過学習ありのパターン

学習の繰り返しで、過学習が起きない場合と起きる場合がある

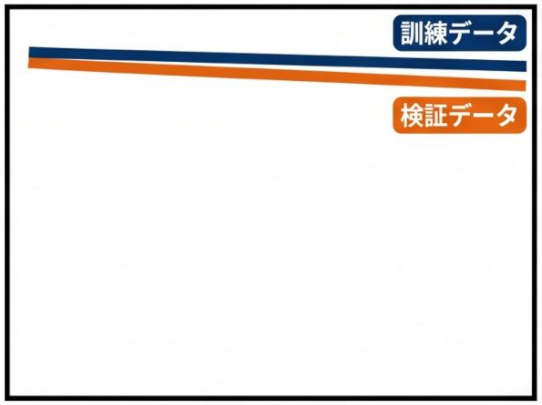




過学習を完全に防ぐことはできない。学習曲線を見て過学習の有無を確認する必要がある

学習不足・最善・過学習の見分け方

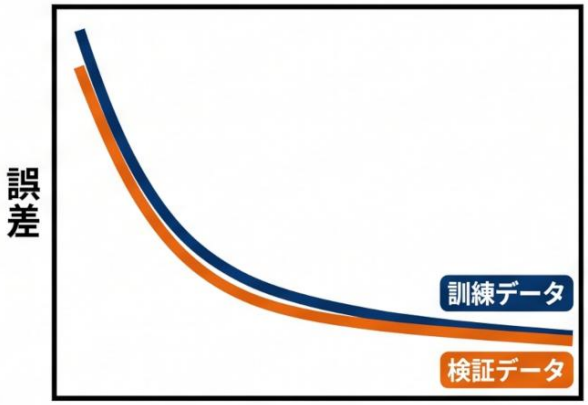
学習不足



両方とも誤差が高いまま

もっと学習が必要

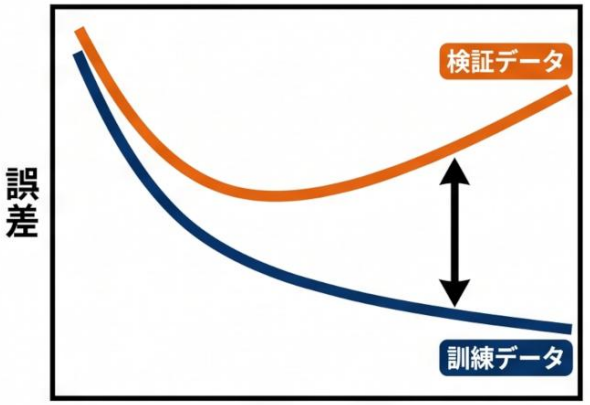
最善



両方とも低く重なる

ちょうど良い学習

過学習



検証データだけ上がる

学習しすぎ

2本の曲線の形と位置関係を見れば、学習不足・最善・過学習のどの状態かが分かる

「誤差」と「精度」の違いと使い分け

誤差（損失）

予測と正解のズレを数値化したもの
学習中に最小化する値

精度（accuracy）

8 / 10 = 80%

正しく分類できた割合
人が結果を評価する値

なぜ2つ必要か

誤差は連続した数値で、わずかな改善も捉えられる→学習に使う
精度は分かりやすい割合で示せる→人が性能を判断するのに使う

演習

演習①　学習曲線の観察

- **4 つの画像**（例：2、0、4、8）を**分類**し、各画像について**10 個の数値**（各カテゴリの確率）が出力される様子を**観察**する演習である。

- プログラムの公開：Google Colab 上で公開されている。

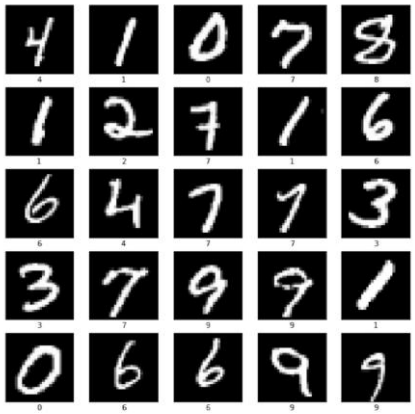
https://colab.research.google.com/drive/1IfArIvhh-FsvJIE9YTNO8T44Qhpi0rIJ?usp=sharing

- 実行結果、プログラム、説明を**閲覧するだけ**であれば、**Google アカウントは不要**

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

機械学習の訓練データと学習の仕組み

大量の訓練データを用いて学習を行う



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

例：手書き数字の画像60000枚

それぞれの正解（4, 1, 0, 7, 8, ...）60000個

⇒訓練データとして使用する.

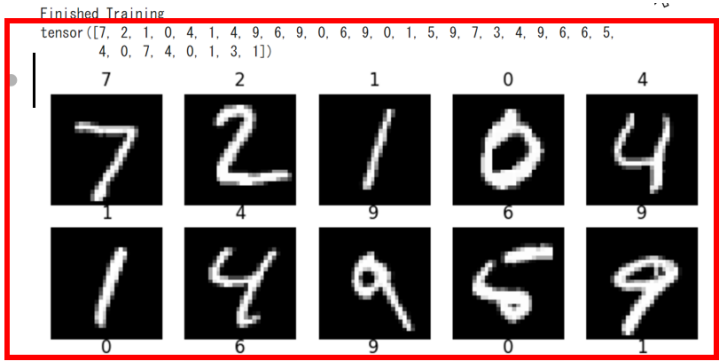
機械学習のプログラムが訓練データを用いて学習を行う

プログラム

```
[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する．
学習ののち，新たな手書き数字の画像分類を行うことができる．

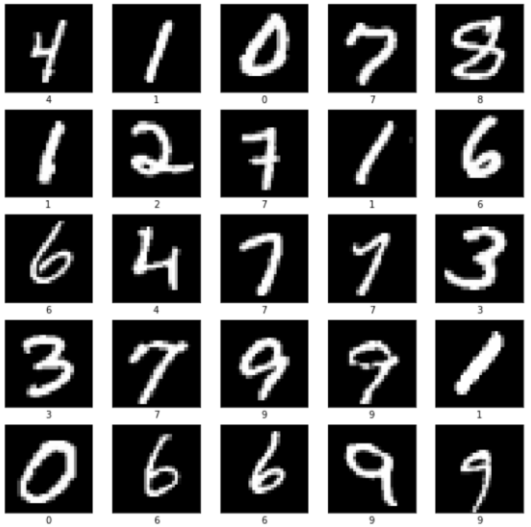
```
Finished Training
tensor([7, 2, 1, 0, 4, 1, 4, 9, 6, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, 9, 6, 6, 5,
        4, 0, 7, 4, 0, 1, 3, 1])
```



使用する訓練データと検証データ

- **訓練データ**

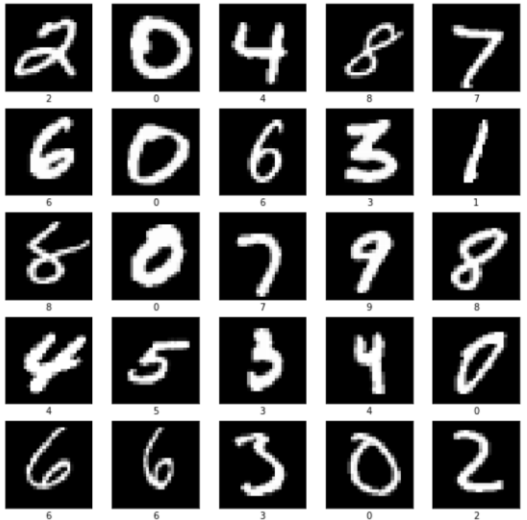
60000枚の画像と正解



抜粋

- **検証データ**

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

同じ訓練データを用いた**学習**を **20 回**繰り返す。学習の繰り返しごとに、**訓練データ**と**検証データ**に対する**精度や誤差**を算出

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

学習の繰り返しごとに、**訓練データ**と**検証データ**に対する**精度や誤差**が表示される

```
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)
```

プログラム

```
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]
```

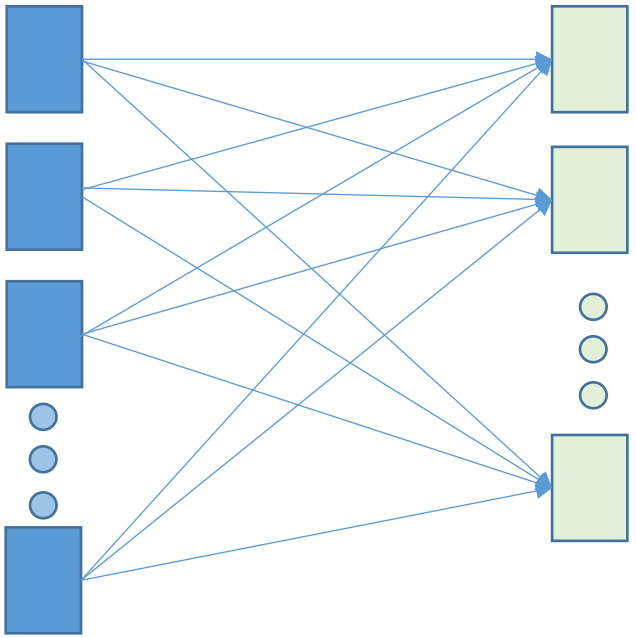
実行結果

重みの観察

学習終了時の結合の重みが表示される

中間層：１２８個

最終層：１０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）

画像分類結果の確認

４つの画像を分類

分類結果　１０個の数値（最終層のニューロン数からの出力）

```
[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]
[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]
[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]
[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]
```

2

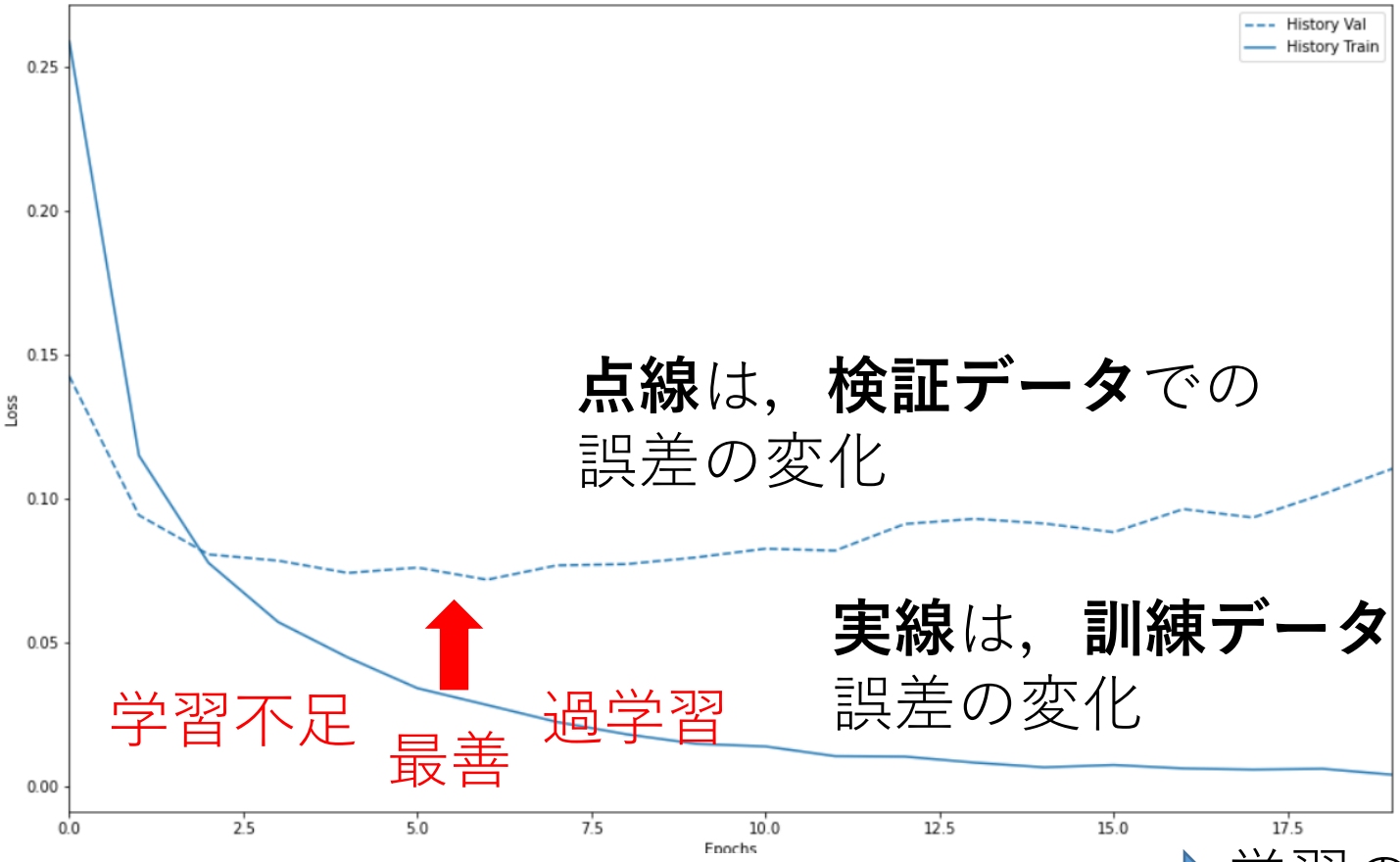
0

4

8

学習曲線の読み取り

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



演習②　ニューロン数の異なるニューラルネットの学習曲線の比較

- ３種類のモデル

中間層のニューロン数 **400**

中間層のニューロン数 **4000**

中間層のニューロン数 **40000**

- プログラムの公開：Google Colab 上で公開されている。

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

- 実行結果、プログラム、説明を**閲覧するだけ**であれば、**Google アカウントは不要**
- **プログラムを変更し再実行する場合**は，**Google アカウント**が必要.

分類精度の比較

20 回の学習の繰り返しののち、検証データでの分類の正解率を計測

ニューロン数	分類精度
400	**0.983**
4000	**0.983**
40000	**0.983**

結果はいずれも 0.983。この場合、分類精度はほとんど変化しない。

【考察】10 倍、100 倍になると、結合の数も 10 倍、100 倍となり、探索空間（**学習で調整すべきパラメータの組み合わせ全体の広がり**）の大きさも **10 倍、100 倍**になる。しかし、**精度の向上が保証されるわけではない**。コンピュータの動作は遅くなり、過学習の可能性が高まる場合もある。

演習③　学習曲線の例

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

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

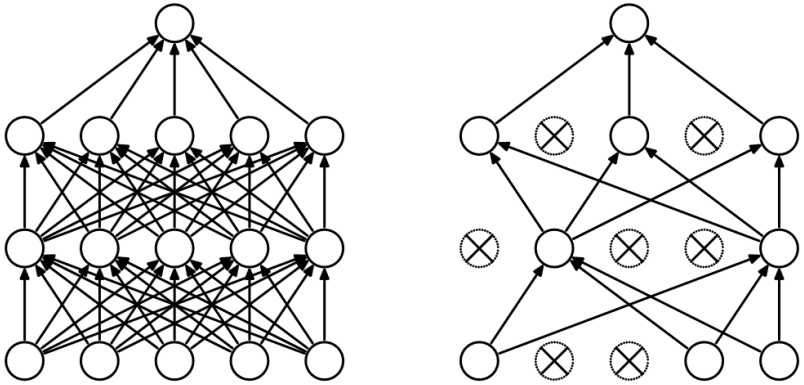
binary_crossentropy, categorical_crossentropy,
cosine_similarity, kld, kullback_leibler_divergence, mae など

- **学習の高速化**

バッチ など

ドロップアウト

学習時にユニットをランダムに選び、存在しないものとして扱う手法。過学習の抑制を目的とする（2014 年発表、Srivastava ら）

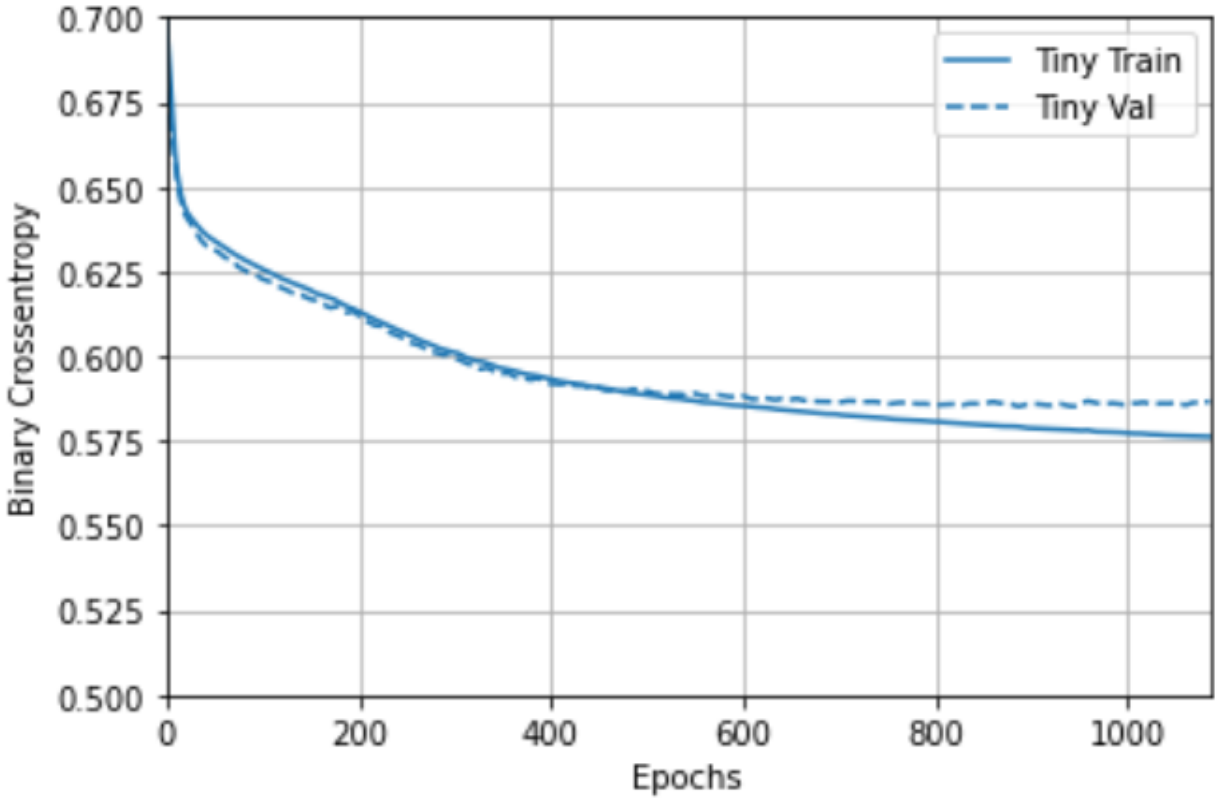


(a) Standard Neural Net　　(b) After applying dropout.

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

見どころ①　学習曲線

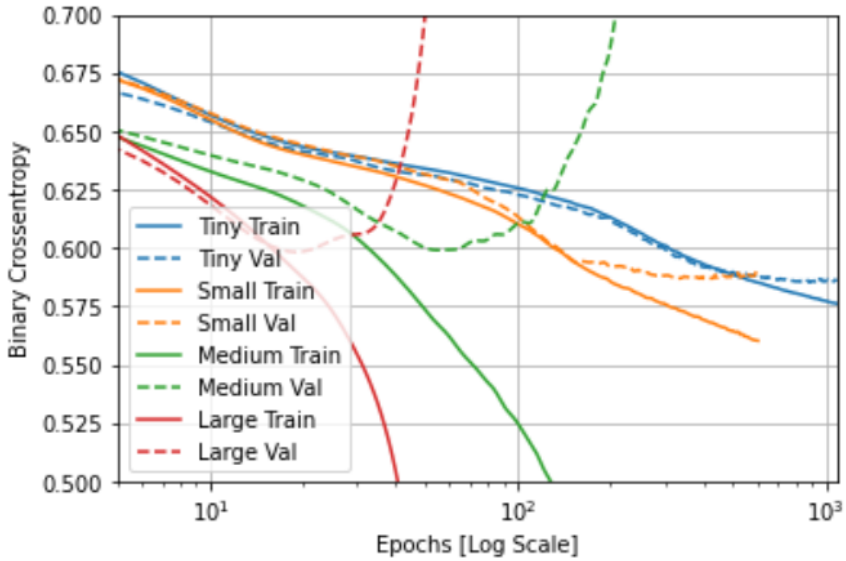
学習曲線は、訓練データと検証データの**誤差の推移**を表したグラフ



Train （実線）は訓練データ、Val（破線）は検証データに対する誤差を示す

見どころ②　4 種類のニューラルネットワークの比較

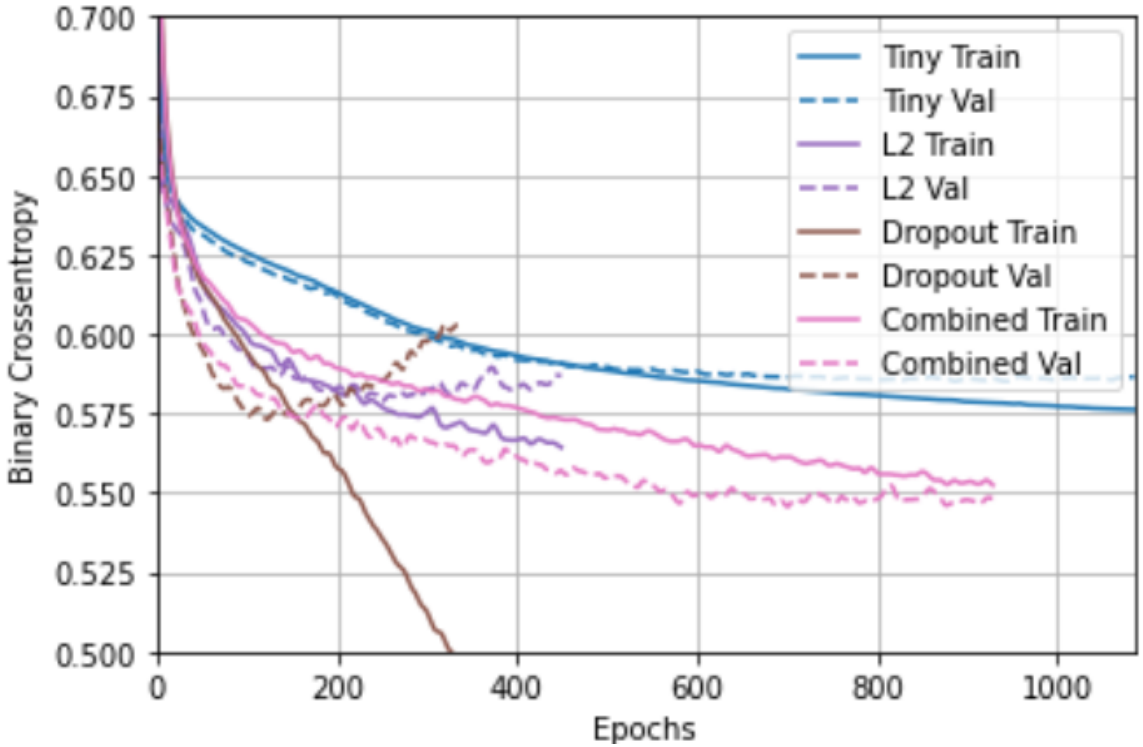
ニューラルネットワークの**ニューロン数や層を増やす**ことが4 種類のニューラルネットワークの比較



Tiny（1 層目のユニット数 16）
Small（1 層目 16、2 層目 16）
Medium（1 層目 64、2 層目 64、3 層目 64）
Large（1 層目 512、2 層目 512、3 層目 512、4 層目 512）を比較
〇　**Medium と Large では過学習が発生**

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」に対し、L2正則化とドロップアウトを適用



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