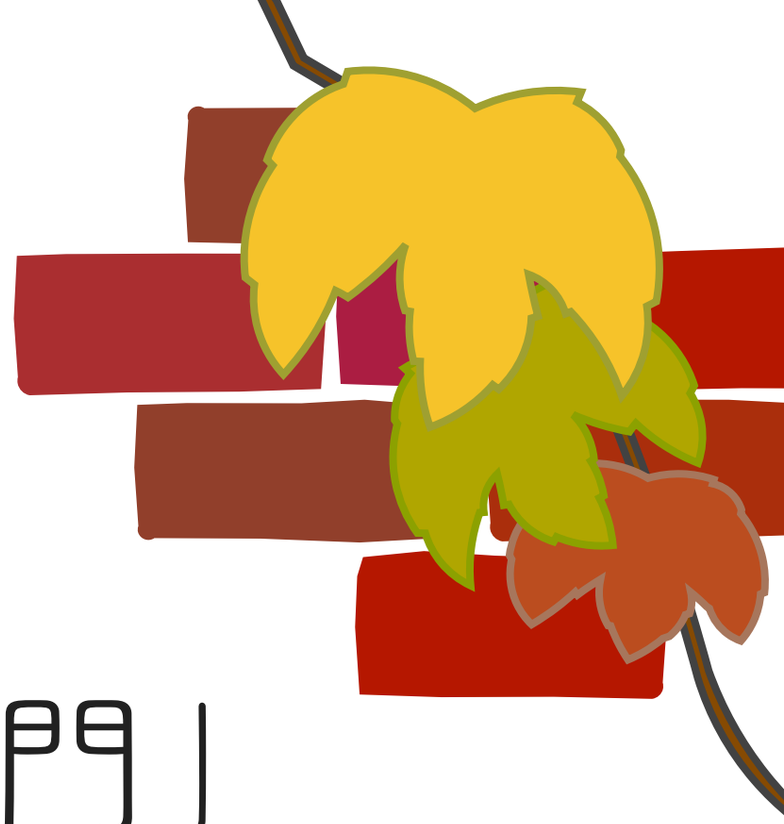


講義「機械学習入門」

前半

立教大学 人工知能科学研究科

瀧 雅人

目次

機械学習と深層学習の概論

- **機械学習とは？**帰納推論と汎化
- **線形回帰とロジスティック回帰**
- **ニューラルネットとディープラーニング**
- **Transformer**
- **ライブラリーの簡単な紹介**

今日は簡単なモデル・手法しか紹介しませんが、十分に機械学習のエッセンスが詰まっています。興味を持たれた方は、参考書でぜひ勉強なさってください。

例：ビショップ「パターン認識と機械学習」

1. 機械学習とは？

ー帰納推論と汎化ー

1. 色々な機械学習アルゴリズム

- インスタンス学習・メモリ学習・怠惰学習
- モデル学習
  - **教師あり学習**　データが模範解答も含む
    - 半教師あり学習
  - **教師なし学習**　データに模範解答なし
  - 質問学習
  - **強化学習**

    } 学習者（モデル）が環境（現実世界）に作用できる
  - 敵対的学習
  - ⋮

1. 色々な機械学習アルゴリズム

- 教師あり学習
  - 線形回帰
  - ロジスティック回帰
  - SVM
  - 決定木
  - ランダムフォレスト
  - ...
- 教師なし学習
  - 主成分分析
  - k平均法
  - 混合ガウス法
  - 階層クラスタリング
  - 行列因子分解
  - ...

1. 色々な機械学習タスク

回帰

分類

クラスタリング

次元削減

異常検知：「未定義」の異常を発見

欠損値補完

密度推定

生成(サンプリング)

...

1. 色々な機械学習の応用

画像認識・画像生成

顔認識

機械翻訳

推薦システム

音声認識・音声合成

自動運転

ゲームAI

金融

診断

・・・

2. なぜいま機械学習なのか？

“big data”

データの巨大さ、多様性、複雑性、データフローの速度

↓

人間の把握能力を超えている

強力な計算能力 → ↓

機械学習の活用で価値を生む

2. 機械学習はいつ使う？

・人間にはできる作業だが、明示的にプログラムするのが困難なタスク

2. 機械学習はいつ使う？

・人間にはできる作業だが、明示的にプログラムするのが困難なタスク

・人間の手に負えない「規模」のデータ（PとN）

2. 機械学習はいつ使う？

・人間にはできる作業だが、明示的にプログラムするのが困難なタスク

・人間の手に負えない「規模」のデータ（PとN）

・**明示プログラムにはない適応性・柔軟性**

ビジネス環境の変化に対するシステム側の迅速な対応。

2. 機械学習はいつ使う？

・人間にはできる作業だが、明示的にプログラムするのが困難なタスク

・人間の手に負えない「規模」のデータ（PとN）

・明示プログラムにはない適応性・柔軟性

・スケーラビリティを要する

データが増えるたびに統計分析要員を雇用していては、ビジネス成長の律速になる。新データに人が慣れるまでの時間もかかる。

3. 機械学習とは？

A.Samuel (1959)

field of study that gives computers the ability to learn without being explicitly programmed

経験から学習するプログラムが作れれば、我々がいちいち事細かにプログラムする必要がなくなる

3. 機械学習とは？: formalな定義

T.M.Mitchell (1997)

コンピュータプログラムが**タスクのクラスT**と**性能指標P**に関し**経験E**から学習するとは、T内のタスクのPで測った性能が、経験Eにより改善される事を言う。

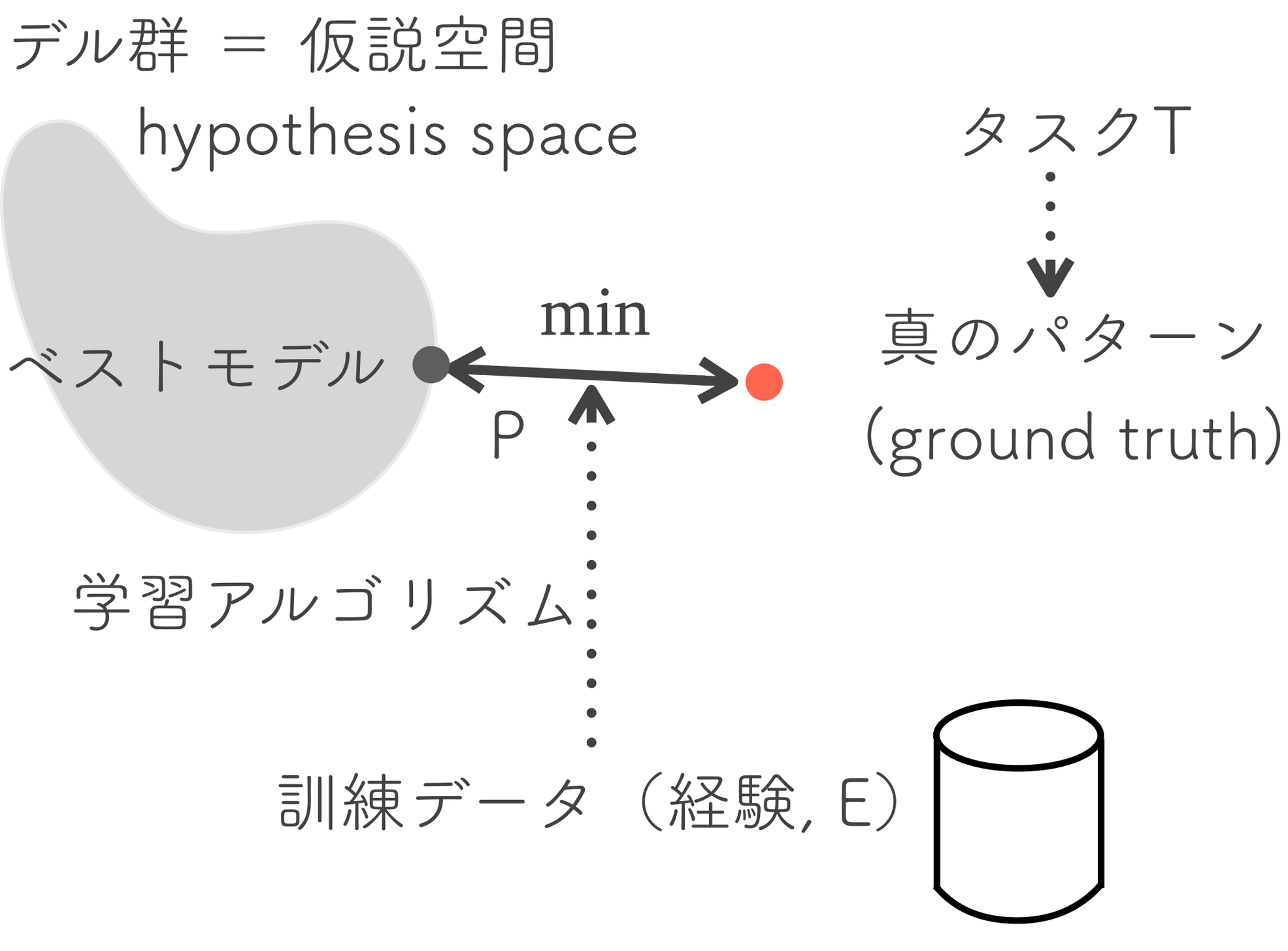
T：機械学習に解かせたい課題

P：性能を定量評価するために人間が設計した数値指標

E：データ

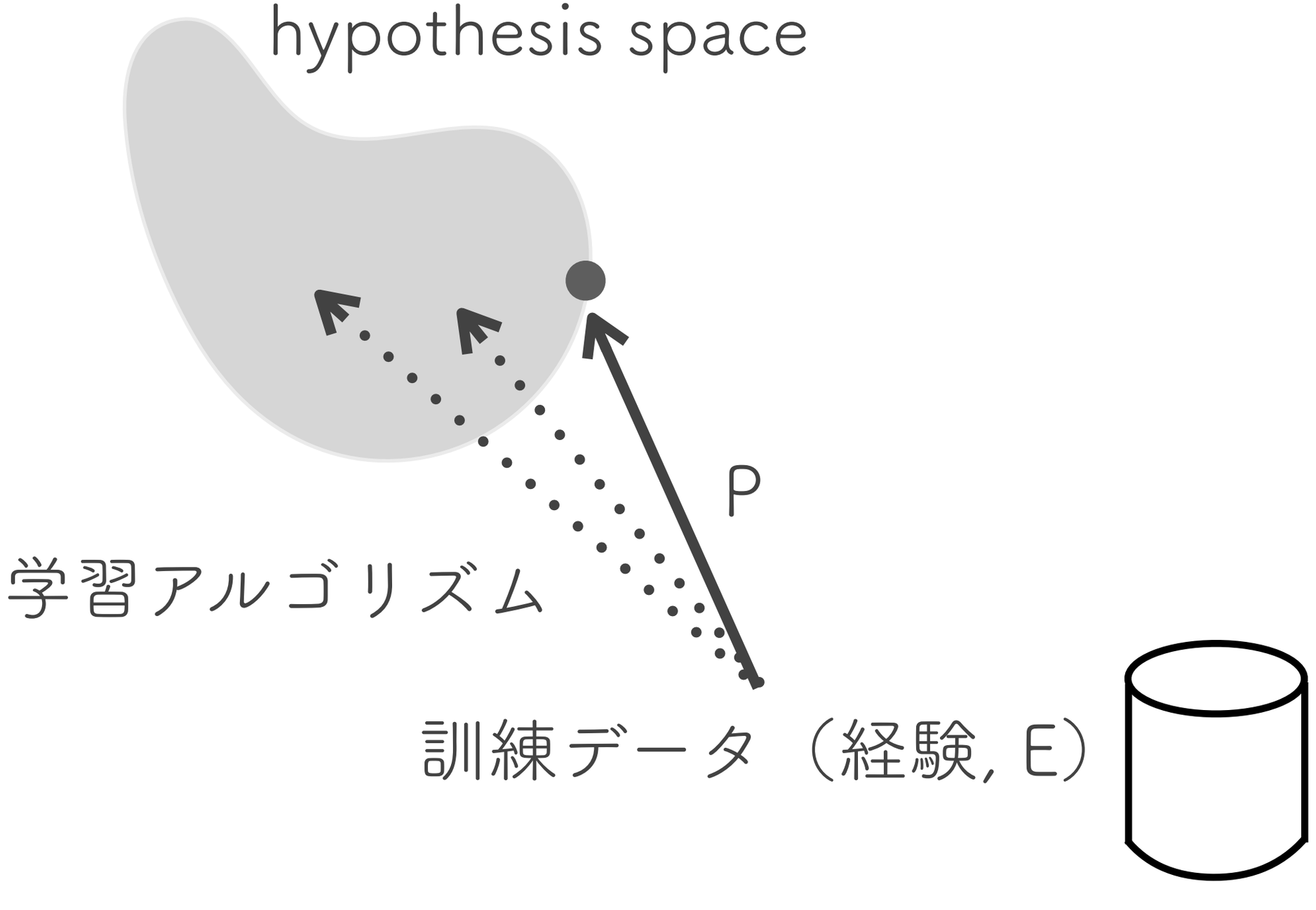
model/architecture：プログラム本体 (+学習アルゴリズム)

3. 機械学習とは？: formalな定義



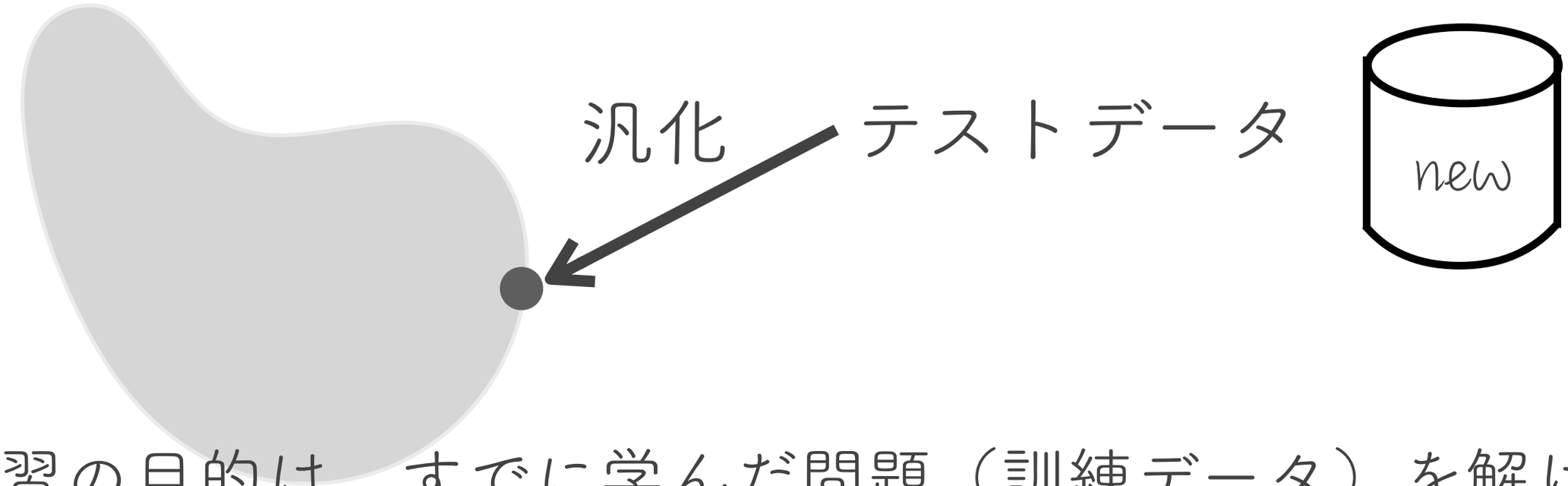
3. 機械学習とは？: formalな定義

訓練フェーズ：データと一番マッチする仮説を探す



3. 機械学習とは？: formalな定義

予測フェーズ：未知のデータにも仮説が通用する

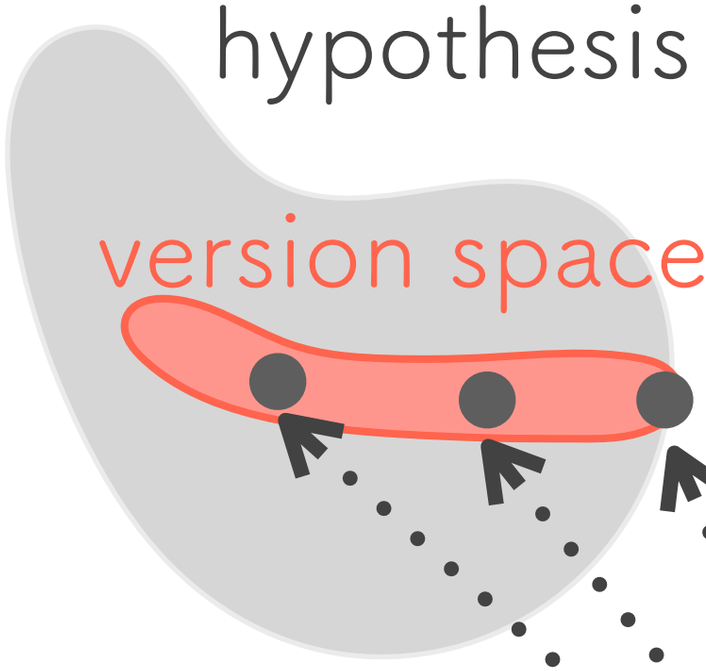


機械学習の目的は、すでに学んだ問題（訓練データ）を解けることではなく、**見たことのない新しい問題・データにも正しく推論**を行えること。機械学習≠最適化数学。

はんか
汎化 generalization

3. 機械学習とは？: formalな定義

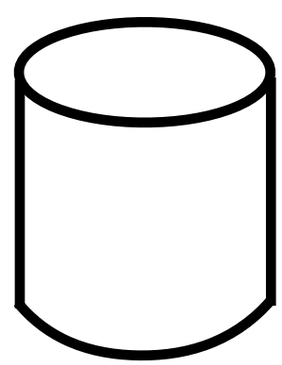
一般に、データとマッチする仮説はたくさんある



（それぞれ一般に汎化性能は違う）

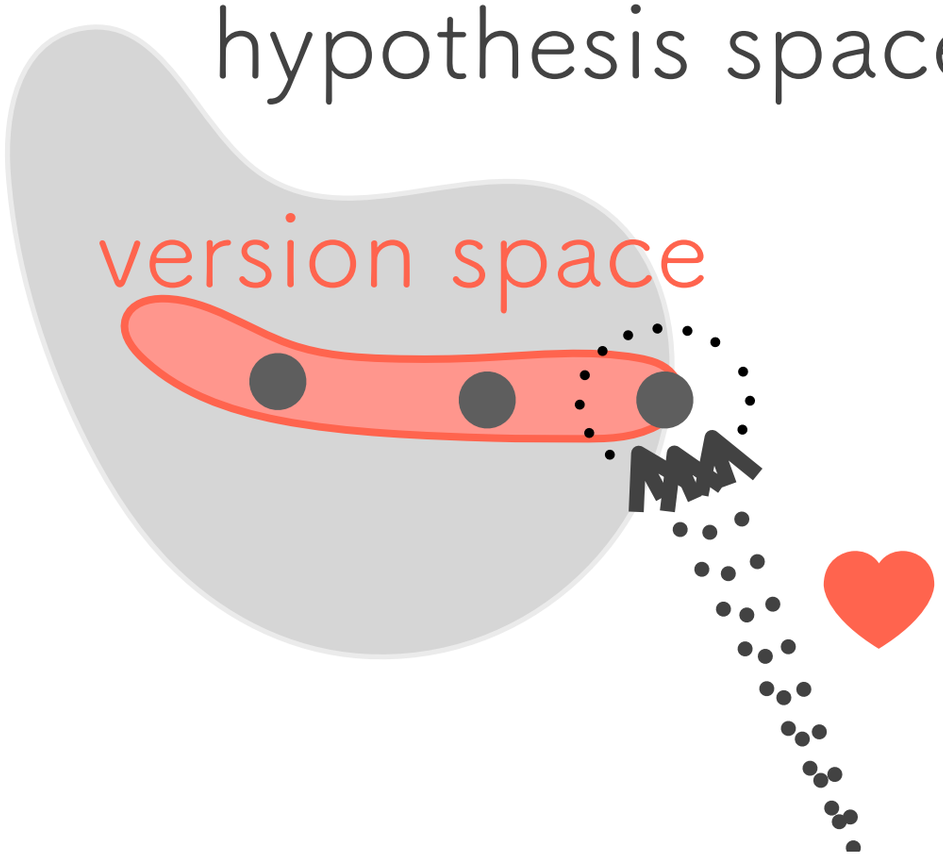
P

訓練データ（経験, E）



3. 機械学習とは？: formalな定義

一般に、データとマッチする仮説はたくさんある



学習アルゴリズムに**特定の仮説への選好（バイアス）**を持たせない限りは決まらない：**帰納バイアス**

→ データ外の知識・バイアス

4. 帰納バイアスとヒュームの懐疑：科学哲学史

- **機械学習＝帰納推論 (inductive reasoning)**

観察した**有限個**の事例 → **普遍**法則

フランシス・ベーコン

ちょっとした観察だけから普遍的な
真理なんて本当にわかるの！？

4. 帰納バイアスとヒュームの懐疑：科学哲学史

- **機械学習＝帰納推論 (inductive reasoning)**

観察した**有限個**の事例 → **普遍**法則

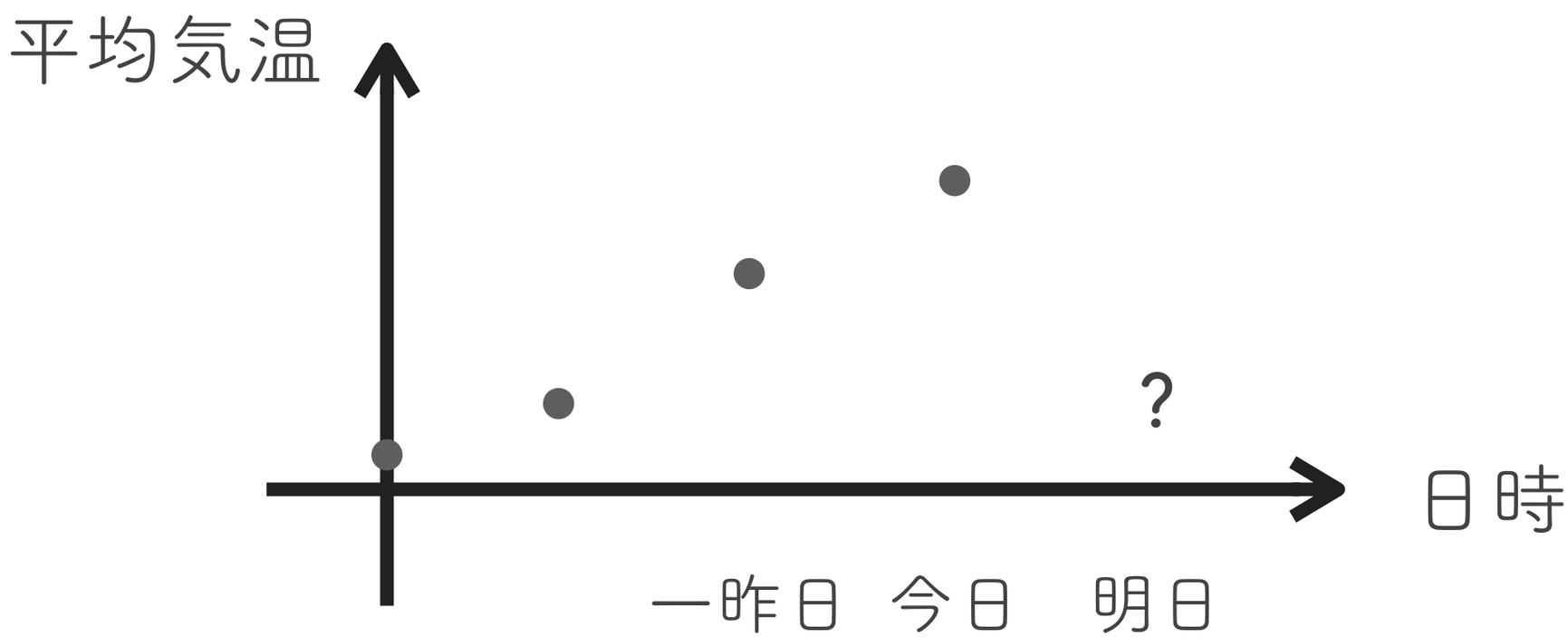
フランシス・ベーコン

帰納は合理的**正当化が不可能**

デイヴィット・ヒューム

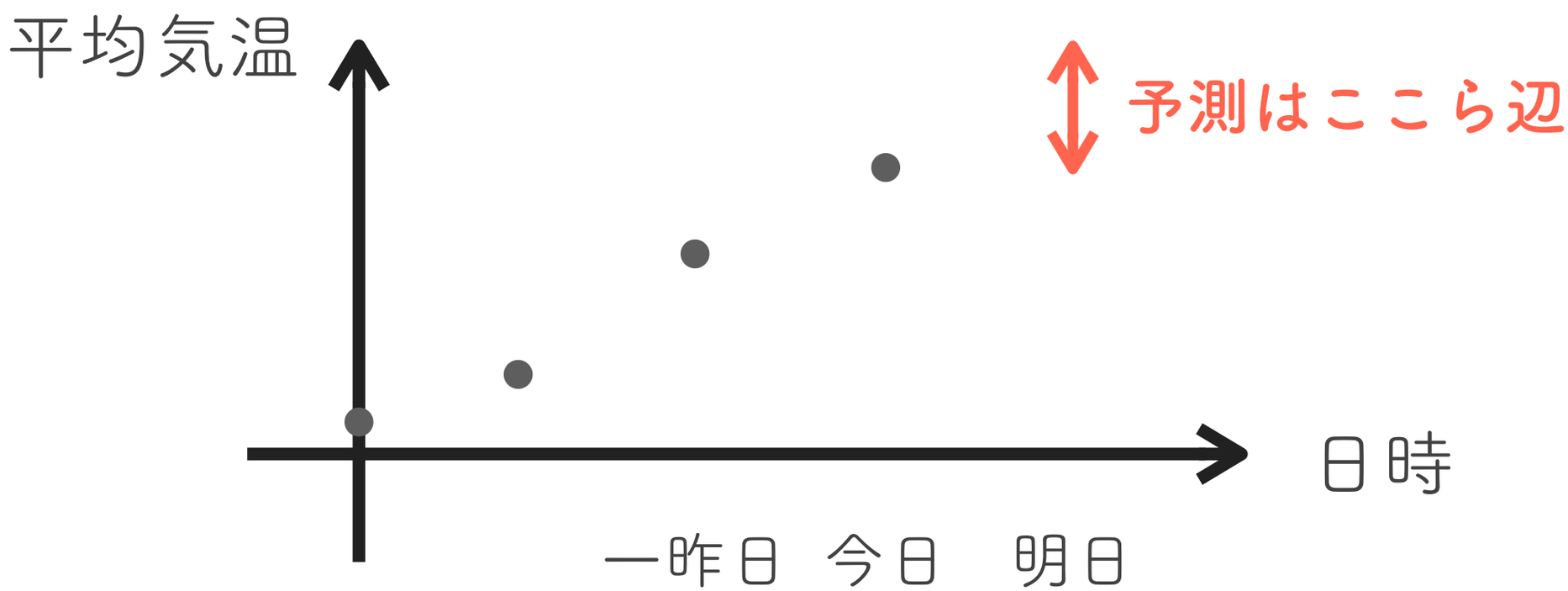
4. 帰納バイアスとヒュームの懐疑：科学哲学史

- **機械学習＝帰納推論 (inductive reasoning)**



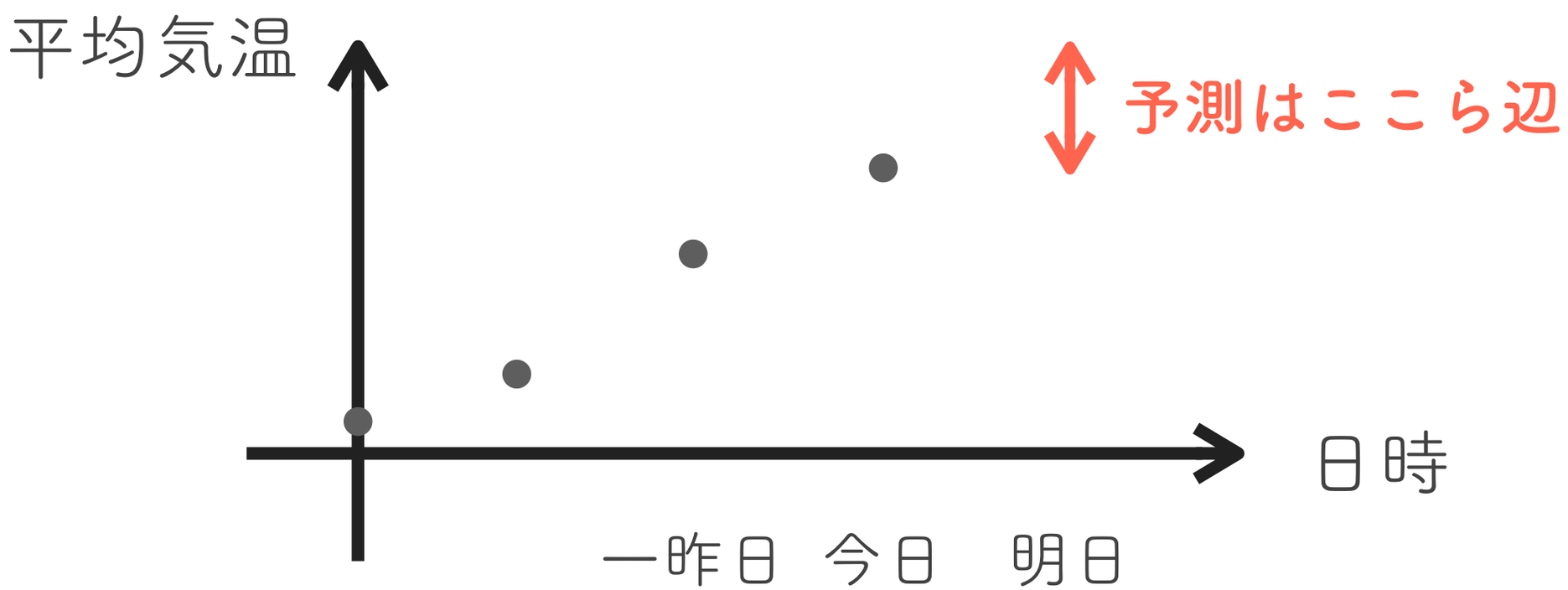
4. 帰納バイアスとヒュームの懐疑：科学哲学史

- 機械学習＝帰納推論 (inductive reasoning)



4. 帰納バイアスとヒュームの懐疑：科学哲学史

- 機械学習＝帰納推論 (inductive reasoning)



斉一性・単純性などを仮定して初めて予測できる

一切の仮定がないと、論理的には丸暗記した四日分の気温を答える事しかできない

4. 帰納バイアスとヒュームの懐疑：科学哲学史

- **機械学習＝帰納推論 (inductive reasoning)**

  人間の与える**帰納バイアス**が、普遍的なパターンの獲得や、未来への予測に不可欠。**データだけでは不十分**。

  → 学習が容易になる(人間が「穴埋め問題」にしてくれる)
  柔軟性を犠牲にする(仮定の範囲でしか学習不可)

このトレードオフをバランスするのが我々の仕事

No-Free-Lunch定理 [Wolpert,1996]

全てのタスクで勝るアルゴリズムは存在しない

6. 帰納バイアスの初等的な例

- **オッカムの剃刀 (Occam's razor)**

  データを説明するモデルが複数あるならば、**一番シンプルな(小さな)もの**を選びなさい。**simplicity**

  → 科学、機械学習のオーソドオックスな基準
  自明でない。論争あり。「**シンプル**」**とは？**

6. 帰納バイアスの初等的な例

- **オッカムの剃刀 (Occam's razor)**

  データを説明するモデルが複数あるならば、**一番シンプルな(小さな)もの**を選びなさい。**simplicity**

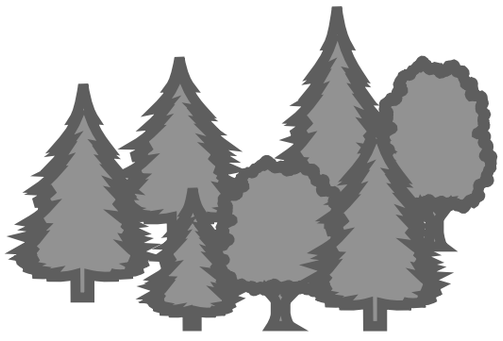
  → 科学、機械学習のオーソドオックスな基準
  自明でない。論争あり。「シンプル」とは？

- **エピクロスの多説明原理 (multi explanation)**

  データを説明するモデルが複数あるならば、**全てを保持**しなさい。**indifference**

  多くの同じ結果を与えるモデルから一つを選ぶ原理的な根拠はない

  → アンサンブル法 (別の良い汎化法)



モデルと真実

“All models are wrong, some are useful.”

1978年 George Box（統計学者）

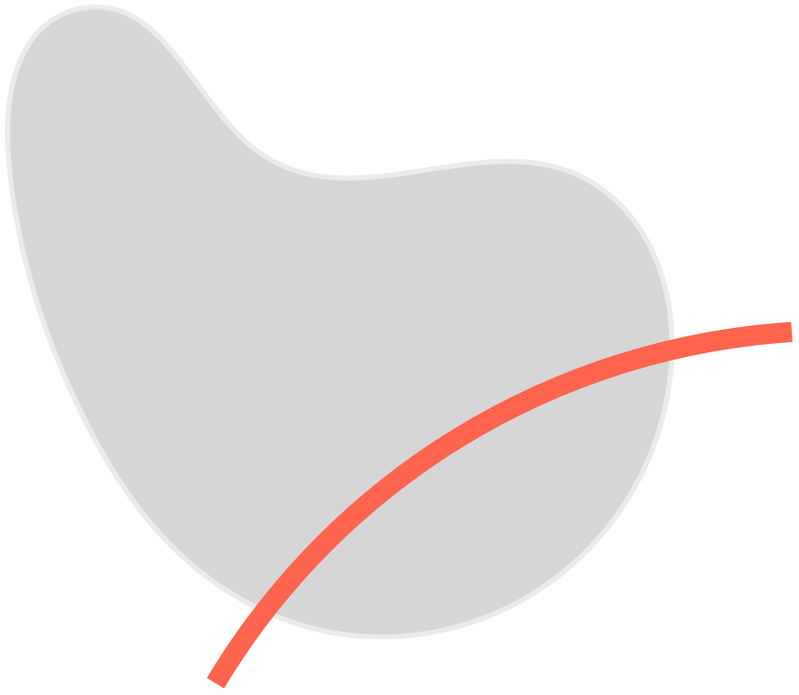
現代の統計学・機械学習は**予測に関する汎化**を支柱にしている。データだけから「普遍法則」が正確にわかるはずはなく、「正しいモデル」があるわけでもない。我々は、事前知識（帰納バイアス）をうまく使い汎化がうまくいった（様に見える）成功事例だけを知っている。

George Box, ‘Science and Statistics’, (1976)

2. 線形回帰とロジスティック回帰

1. 線形モデル：パラメトリックモデルの例

hypothesis space



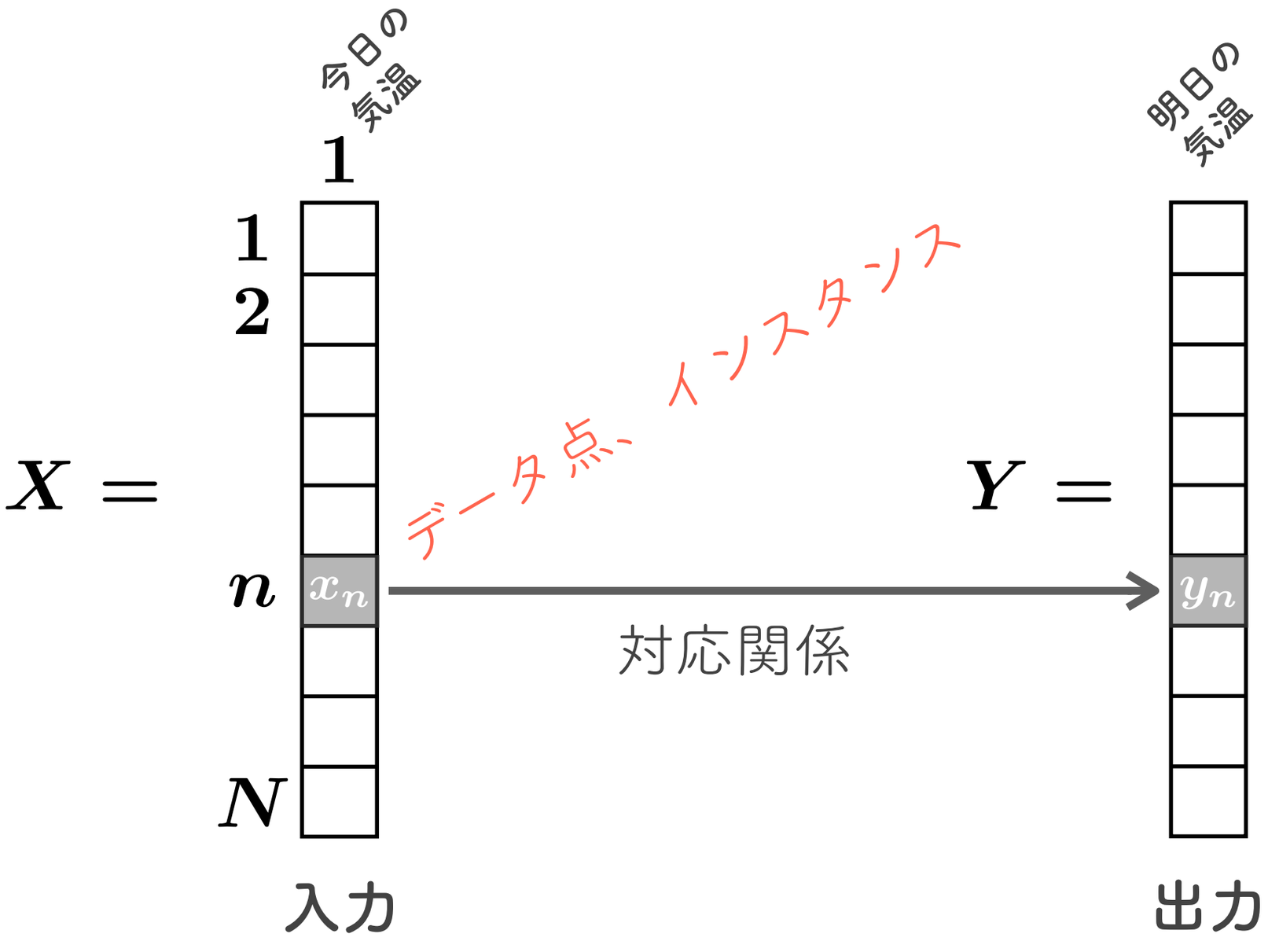
パラメトリックモデル

具体的なパラメータ付き関数でモデル化される仮説のみに範囲を絞る

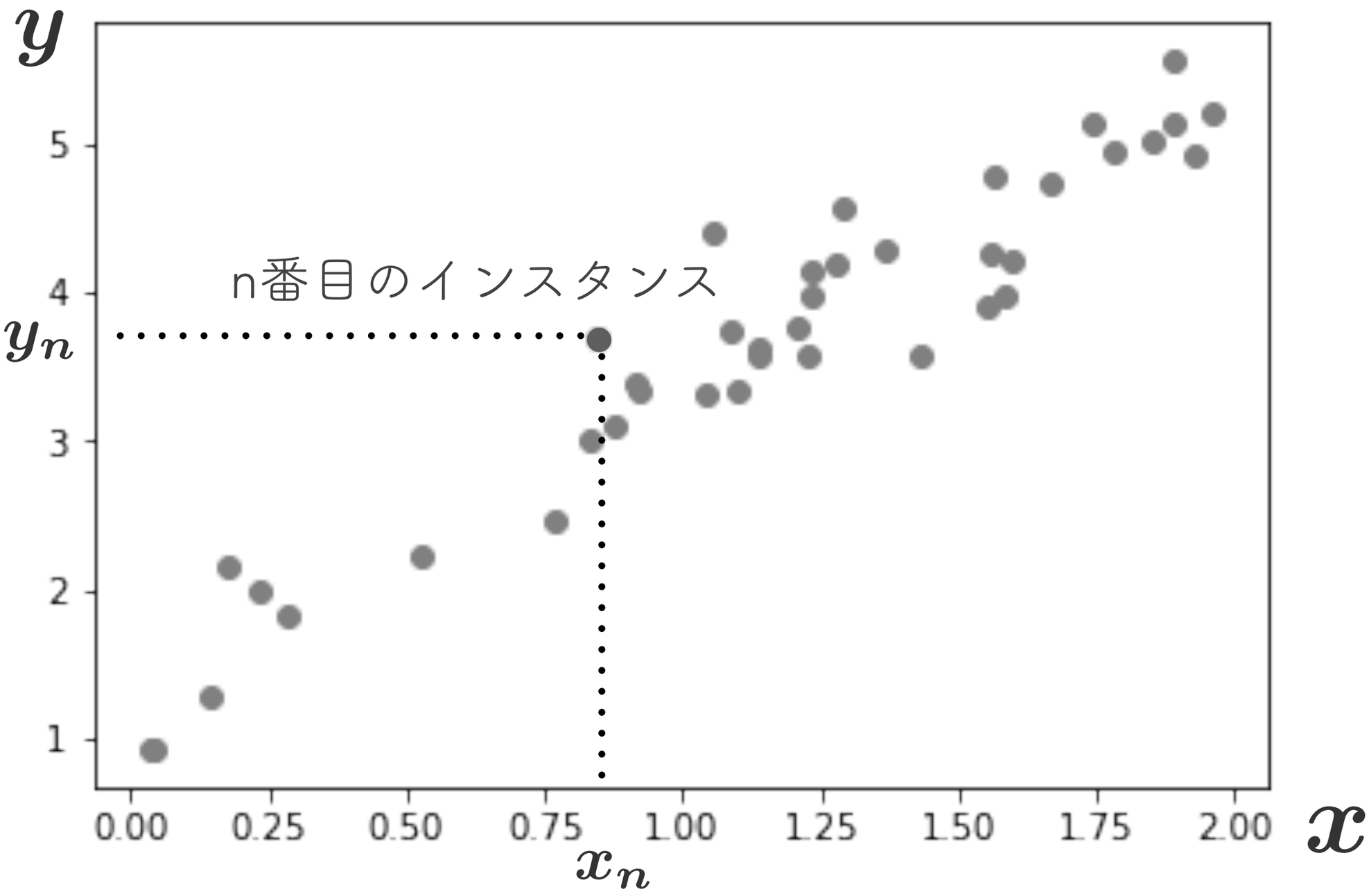
特にパラメータに関して本質的に線形なモデルに話を限る

2. 線形単回帰：教師あり学習の一例

XからYを予測する問題：P=1の場合

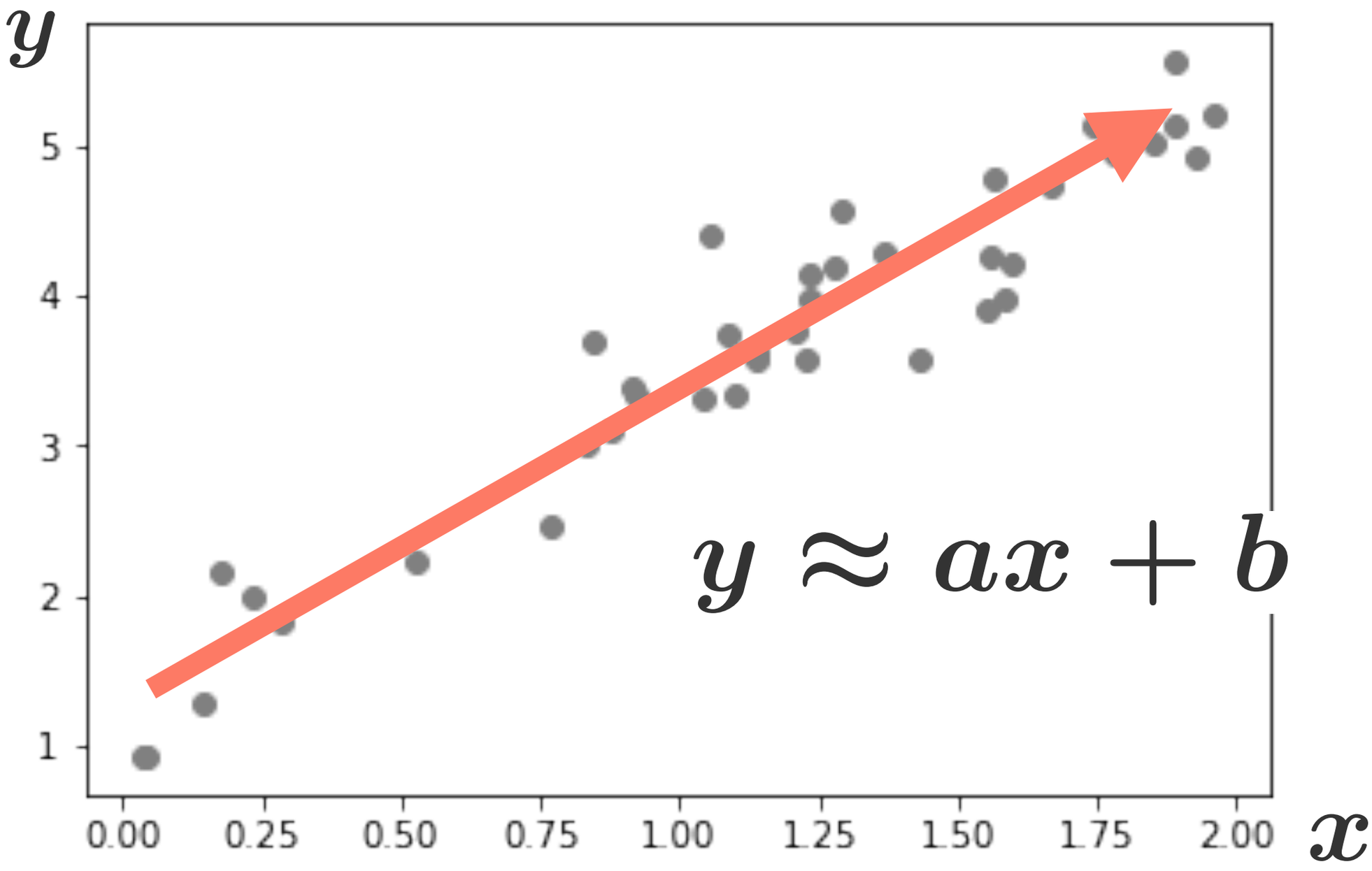


2. 線形単回帰：教師あり学習の一例



散布図

2. 線形単回帰：教師あり学習の一例



xが増えると、yも増える単純な直線的（＝**線形**）な傾向

2. 線形単回帰：教師あり学習の一例

yの予測値（推定値）を与える**線形単回帰モデル**

$$\hat{y}(x, a, b) = ax + b$$

$\hat{y}$：真の値ではなく、**データから推定されるy**の予測値

2. 線形単回帰：教師あり学習の一例

yの予測値（推定値）を与える**線形単回帰モデル**

$$\hat{y}(x, a, b) = ax + b$$

$\hat{y}$：真の値ではなく、**データから推定されるy**の予測値

(a, b)：**学習パラメータ**。事前には適切な値はわからない。データから最適値を推定（学習）する

2. 線形単回帰：教師あり学習の一例

yの予測値（推定値）を与える**線形単回帰モデル**

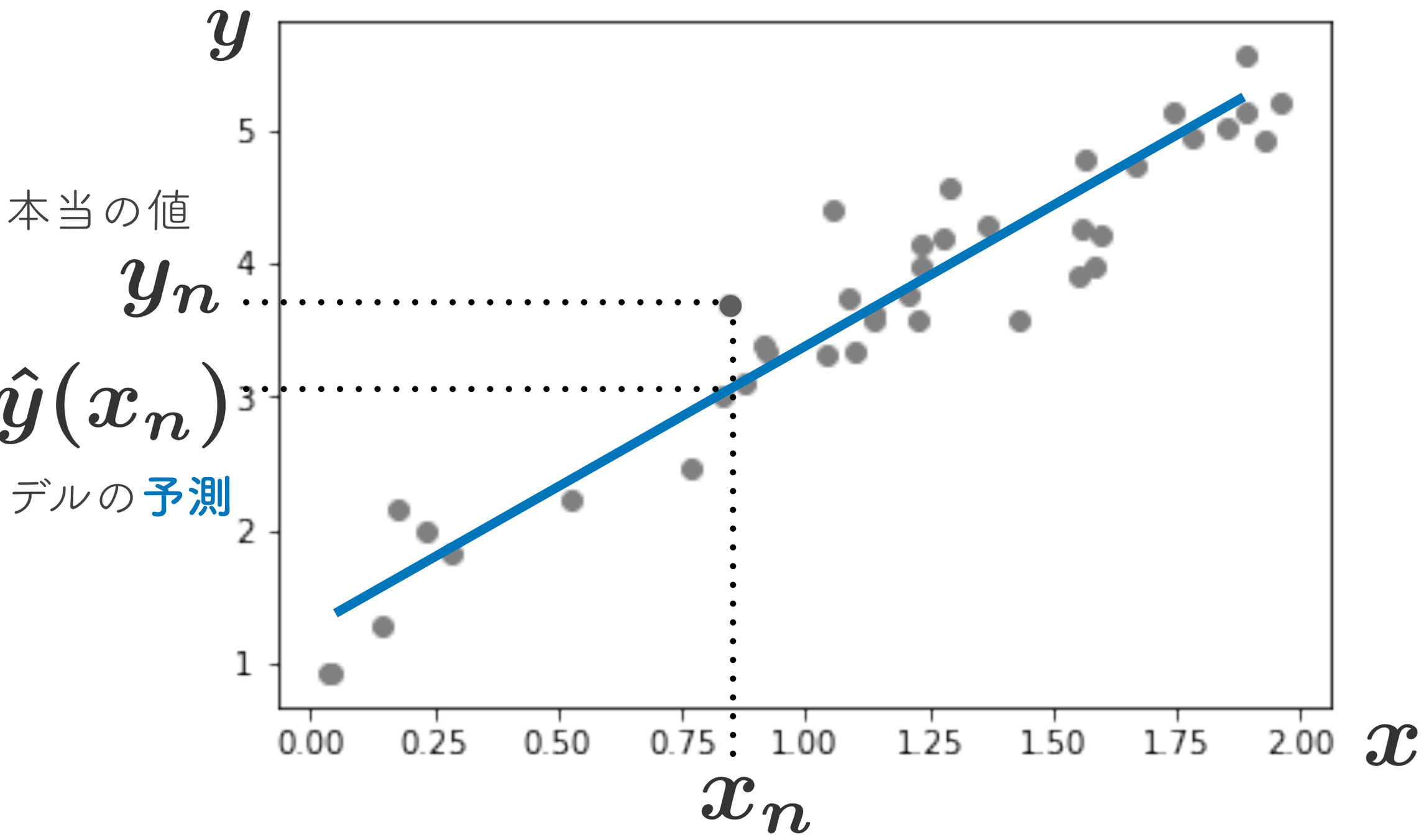
$$\hat{y}(x, a, b) = ax + b$$

$\hat{y}$：真の値ではなく、**データから推定されるy**の予測値

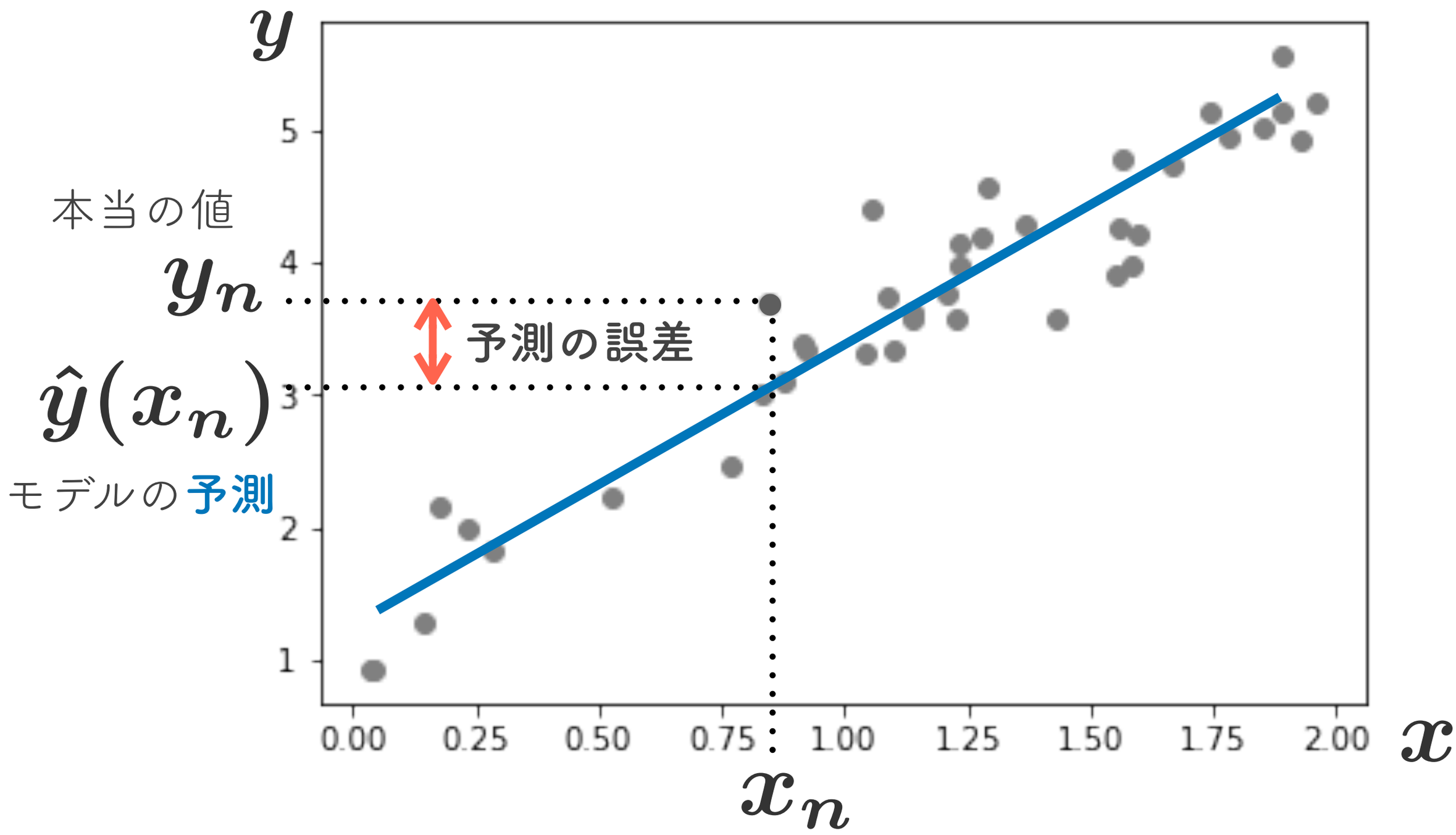
(a, b)：**学習パラメータ**。事前には適切な値はわからない。データから最適値を推定（学習）する

学習＝データと最もマッチするパラメータの値$(\hat{a}, \hat{b})$を推定すること。

2. 線形単回帰：教師あり学習の一例



2. 線形単回帰：教師あり学習の一例



3. 誤差関数と学習

$$E(a, b) = \frac{1}{N} \sum_{n=1}^{N} \left(\hat{y}(x_n) - y_n \right)^2$$

訓練データで測った、**現在のパラメータ値のもとでの**平均的な予測誤差（**平均二乗誤差**、MSE）。

誤差関数を最小にするようなパラメータが、最も**予測誤差が小さい**、つまり**データへの当てはまりの良い**モデル $\hat{y}(x) = ax + b$ を与える。

3. 誤差関数と学習：正規分布モデルと最尤法

$$\hat{y} = ax + b + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

観測（データ）は、線形なパタンの周りに**正規分布ノイズでばらつく**と確率でモデル化してみる

3. 誤差関数と学習：正規分布モデルと最尤法

$$\hat{y} = ax + b + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

$$\Downarrow$$

$$P(\hat{y}|x) = \mathcal{N}(ax + b, \sigma^2)$$

データ分布を線形正規分布でモデル化した。

→ 確率モデルの学習は**最尤推定法**が「定番」。

3. 誤差関数と学習：正規分布モデルと最尤法

ある確率モデル $\boldsymbol{P(\mathrm{y}|\vec{x})}$ を**仮定**した時、ある $\vec{x}_n$ に対してインスタンス $(\vec{x}_n, y_n)$ が得られる確率：

$$\boldsymbol{P(y_n|\vec{x}_n)}$$

データセット $\{(\vec{x}_1, y_1), \cdots, (\vec{x}_N, y_N)\}$ が得られる確率

$$\prod_{n=1}^{N} \boldsymbol{P(y_n|\vec{x}_n)} \qquad *i.i.d.\text{を仮定！}$$

3. 誤差関数と学習：正規分布モデルと最尤法

ある確率モデル $\boldsymbol{P(\mathrm{y}|\vec{x})}$ を**仮定**した時、ある $\vec{x}_n$ に対してインスタンス $(\vec{x}_n, y_n)$ が得られる確率：

$$\boldsymbol{P(y_n|\vec{x}_n)}$$

$\vec{\theta} = (\vec{a}, b)$ の関数

データセット $\{(\vec{x}_1, y_1), \cdots, (\vec{x}_N, y_N)\}$ が得られる確率

$$\boldsymbol{L(\vec{\theta}) = \prod_{n=1}^{N} P(y_n|\vec{x}_n)} \qquad *i.i.d.\text{を仮定！}$$

尤度関数（likelihood）

3. 誤差関数と学習：正規分布モデルと最尤法

尤度関数（likelihood）

$$L(\vec{\theta}) = P(y_1|\vec{x}_1)P(y_2|\vec{x}_2)\cdots P(y_N|\vec{x}_N)$$
$$= \prod_{n=1}^{N} P(y_n|\vec{x}_n)$$

データセット$\{(\vec{x}_1, y_1), \cdots, (\vec{x}_N, y_N)\}$が得られる「確率」

逆に、このデータセットが得られたのは、**この確率(尤度)が大きくなるようなパラメータ$\vec{\theta}$が選ばれているからだ**、と考えてみる → **データに対して尤もらしいパラメータ**

3. 誤差関数と学習：正規分布モデルと最尤法

与えられたデータセットと確率モデルに対して、最適なモデルパラメータは、**尤度関数を最大化するものである：**

$$\vec{\theta}^* = \mathbf{argmax}_{\vec{\theta}}\ L(\vec{\theta})$$

log（単調増加関数）をっても最大値の位置は不変

$$\vec{\theta}^* = \mathbf{argmax}_{\vec{\theta}}\ \log L(\vec{\theta})$$

＊尤度は確率の積なので、小さい値になってアンダーフローしないように対数をとる。

3. 誤差関数と学習：正規分布モデルと最尤法

マイナスをつけると**最小化**：

$$\vec{\theta}^* = \mathbf{argmin}_{\vec{\theta}} \left(-\log L(\vec{\theta}) \right)$$

→ **負の対数尤度**を誤差関数とする最小化問題（学習）

最尤法による学習

$$\vec{\theta}^* = \mathbf{argmin}_{\vec{\theta}}\, E(\vec{\theta})$$

$$E(\vec{\theta}) = -\log L(\vec{\theta})$$

3. 誤差関数と学習：正規分布モデルと最尤法

今モデルは正規分布

$$\mathcal{N}(ax+b, \sigma^2) \propto e^{-\frac{1}{2\sigma^2}(y-(ax+b))^2}$$

→ 負の対数尤度 $E(\vec{\theta}) = -\log L(\vec{\theta})$ が二乗誤差関数を導く

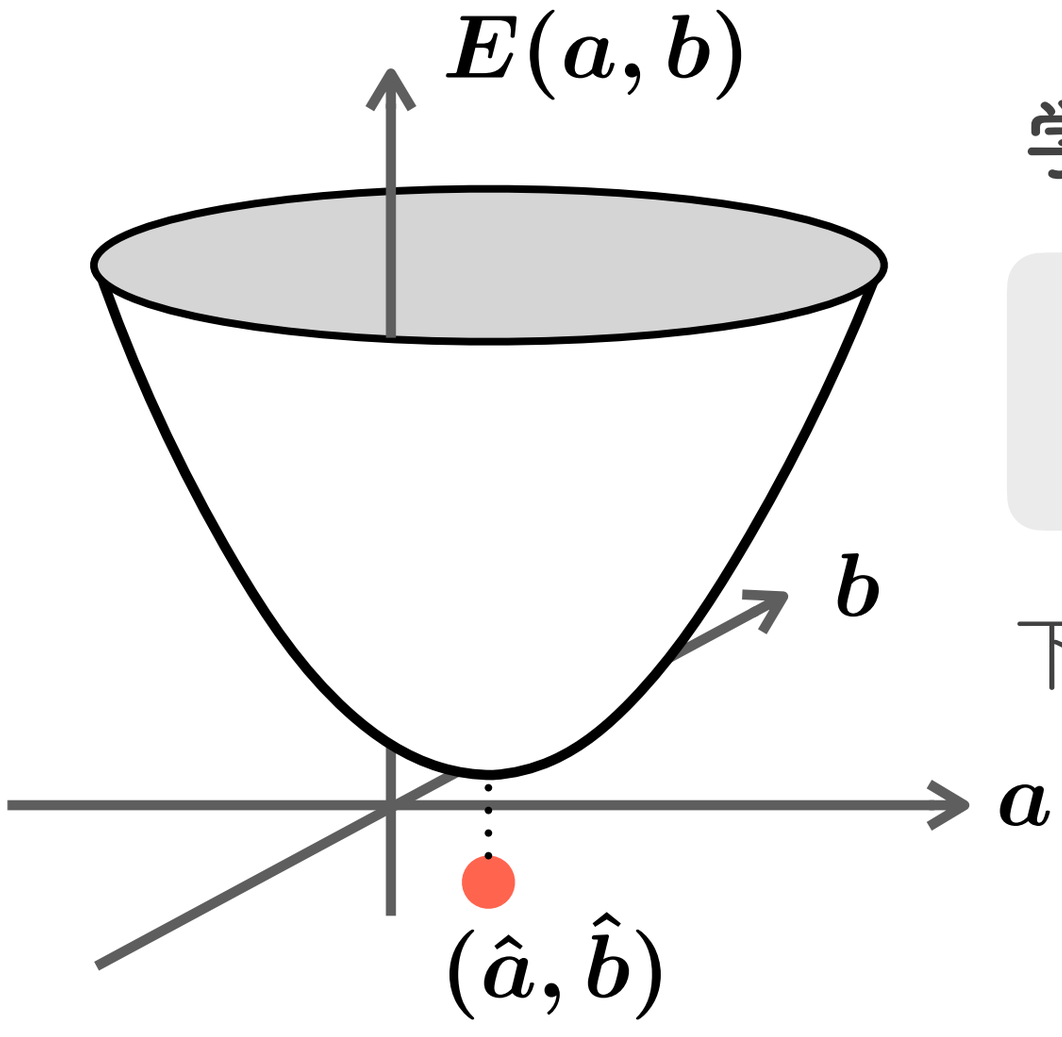
$$E = \frac{1}{2\sigma^2}\sum_n (y_n - (ax_n+b))^2 + C$$

4. 勾配降下法：学習の幾何学的な見方

では、どう学習させる？（＝最小化する？）

4. 勾配降下法：学習の幾何学的な見方

今の場合、誤差関数はそれぞれのパラメータの二次関数（ $\hat{y}(x) = ax + b$ の二乗 → a,bの二次関数）



学習＝最小化問題

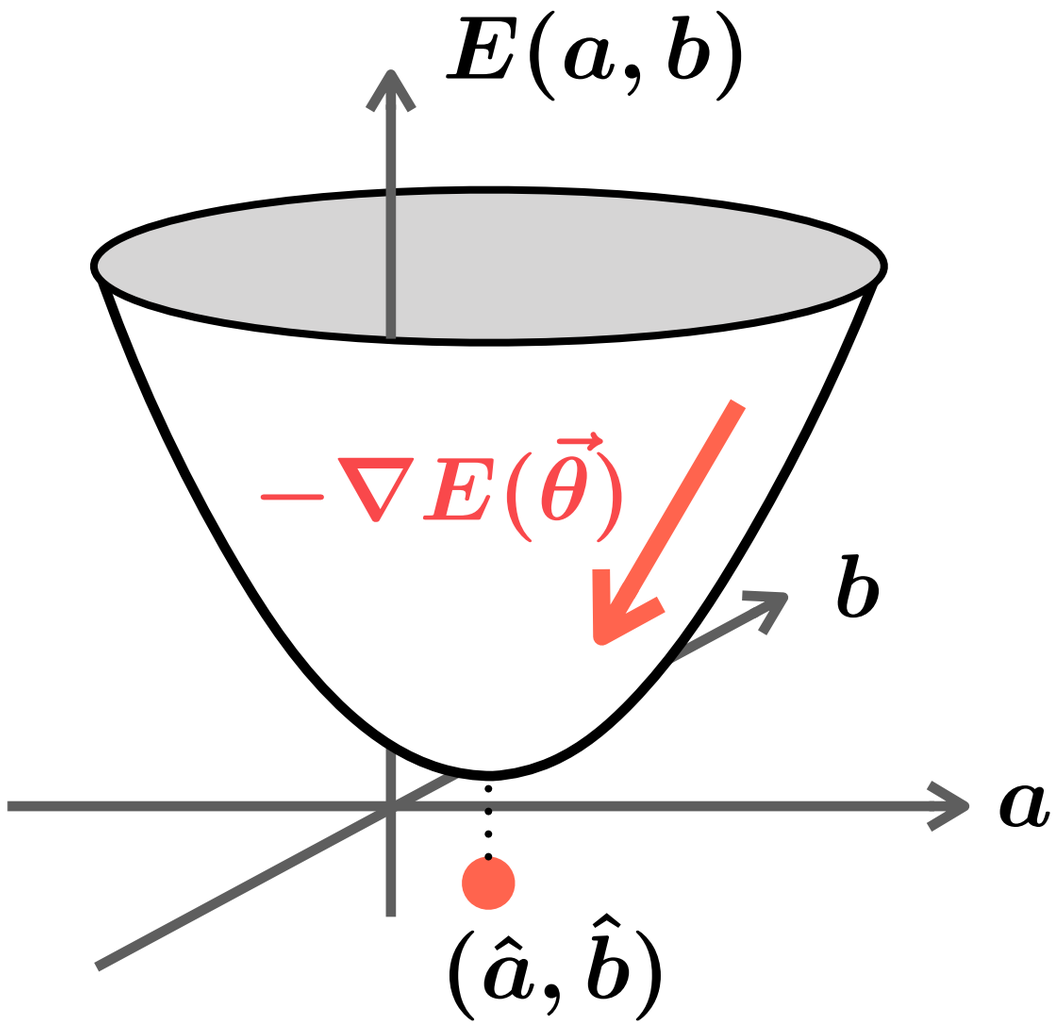
$$\hat{a},\hat{b} = \arg\min_{a,b} E(a,b)$$

下に凸なグラフ上での最小化

4. 勾配降下法：学習の幾何学的な見方

$$E(\vec{\theta}+\delta\vec{\theta}) \approx E(\vec{\theta}) + \nabla E(\vec{\theta}) \cdot \delta\vec{\theta} \longrightarrow \delta\vec{\theta} = -\nabla E(\vec{\theta})$$

最急降下方向



勾配の逆方向にパラメータを反復的に動かしていけは、**凸最適化問題は数値的に解ける**

4. 勾配降下法：学習の幾何学的な見方

学習率 η：学習では決まらないパラメータ、**ハイパーパラメータ**の一例。**程よい**1ステップのスケール

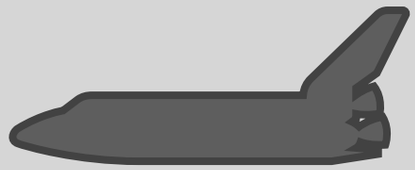
$$a_{new} = a - \eta \frac{\partial E(a,b)}{\partial a}$$

$$b_{new} = b - \eta \frac{\partial E(a,b)}{\partial b}$$

勾配降下法の１ステップ

→ これを数値収束まで繰り返す。

4. 勾配降下法のアルゴリズムのまとめ



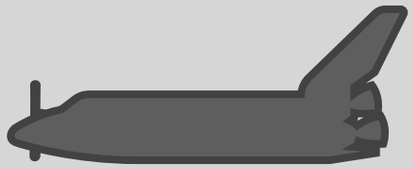
1. 初期値 a, b ← 乱数
   学習率 η ← 0.1など丁度良い値を設定
   反復回数 ← 適当な回数を設定
2. 反復回数だけ次を繰り返す：
   2-1. 現在の a, b に対する勾配を計算
   2-2. 勾配を使って a, b の値を更新

$$a \leftarrow a - \eta \frac{\partial E(a,b)}{\partial a}$$

$$b \leftarrow b - \eta \frac{\partial E(a,b)}{\partial b}$$

3. うまくいけば得られる最終的な値が $\hat{a}, \hat{b}$ の近似値

5. 汎化性能の簡単な推定法：ホールドアウト

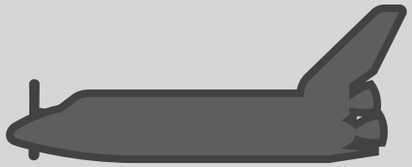


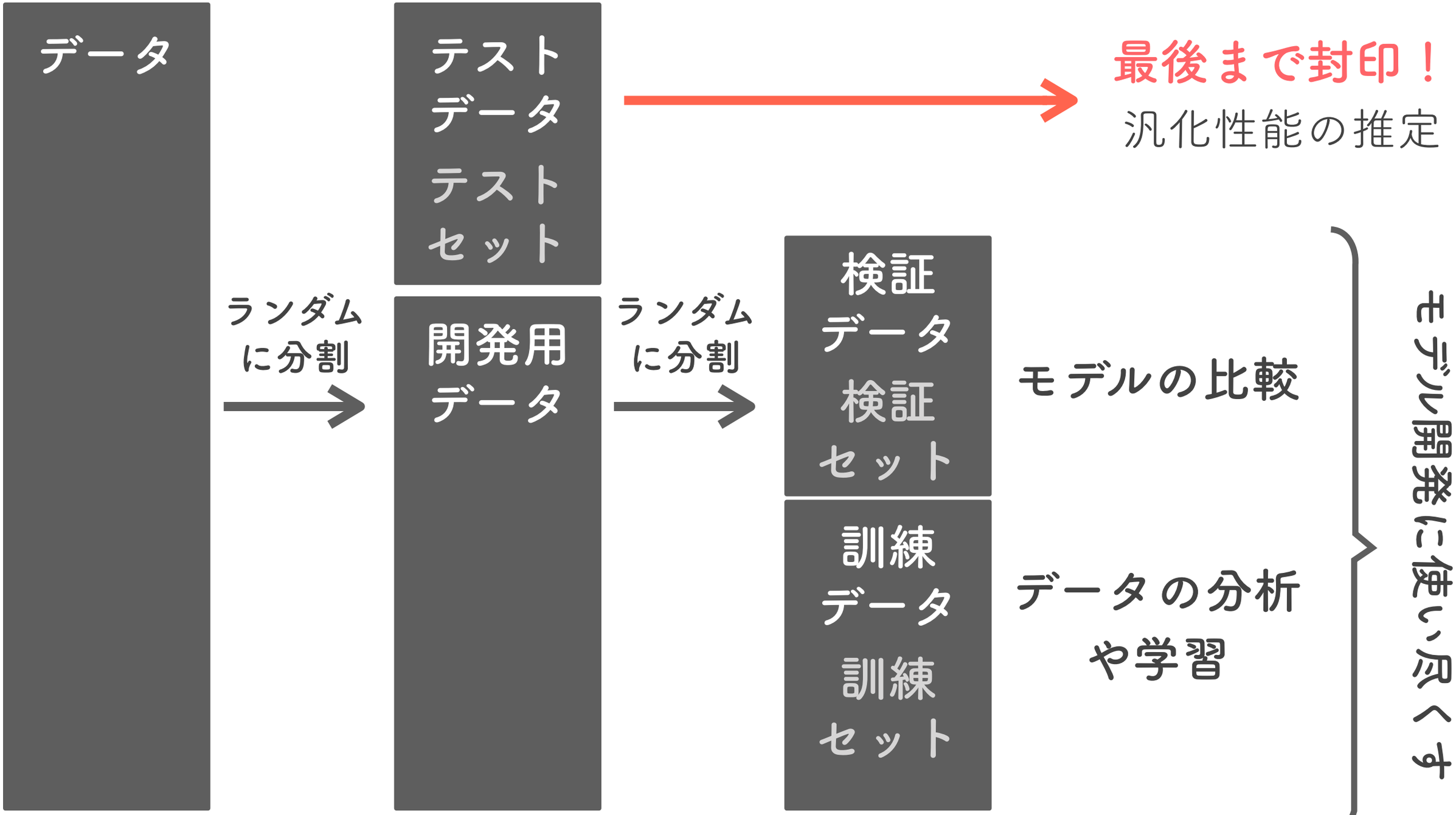
- 学習させたデータ(問題)に答えられることは、機械学習の目的ではない：そんなものは**訓練データの丸暗記**でできる
- **学習していないデータ(問題)に答えられる**ことが目標：**汎化**

汎化したかしていないかを本当に判断ためには、**可能なすべての状況でのデータ**が揃わなくてはいけないが、それは不可能。

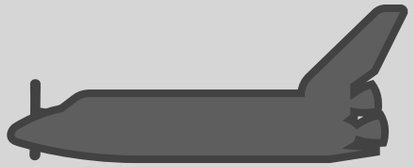
汎化したかしていないかを「ざっくり」近似的に測るために、**学習には一切使っていない**テストデータを用意する：**ホールドアウト法**

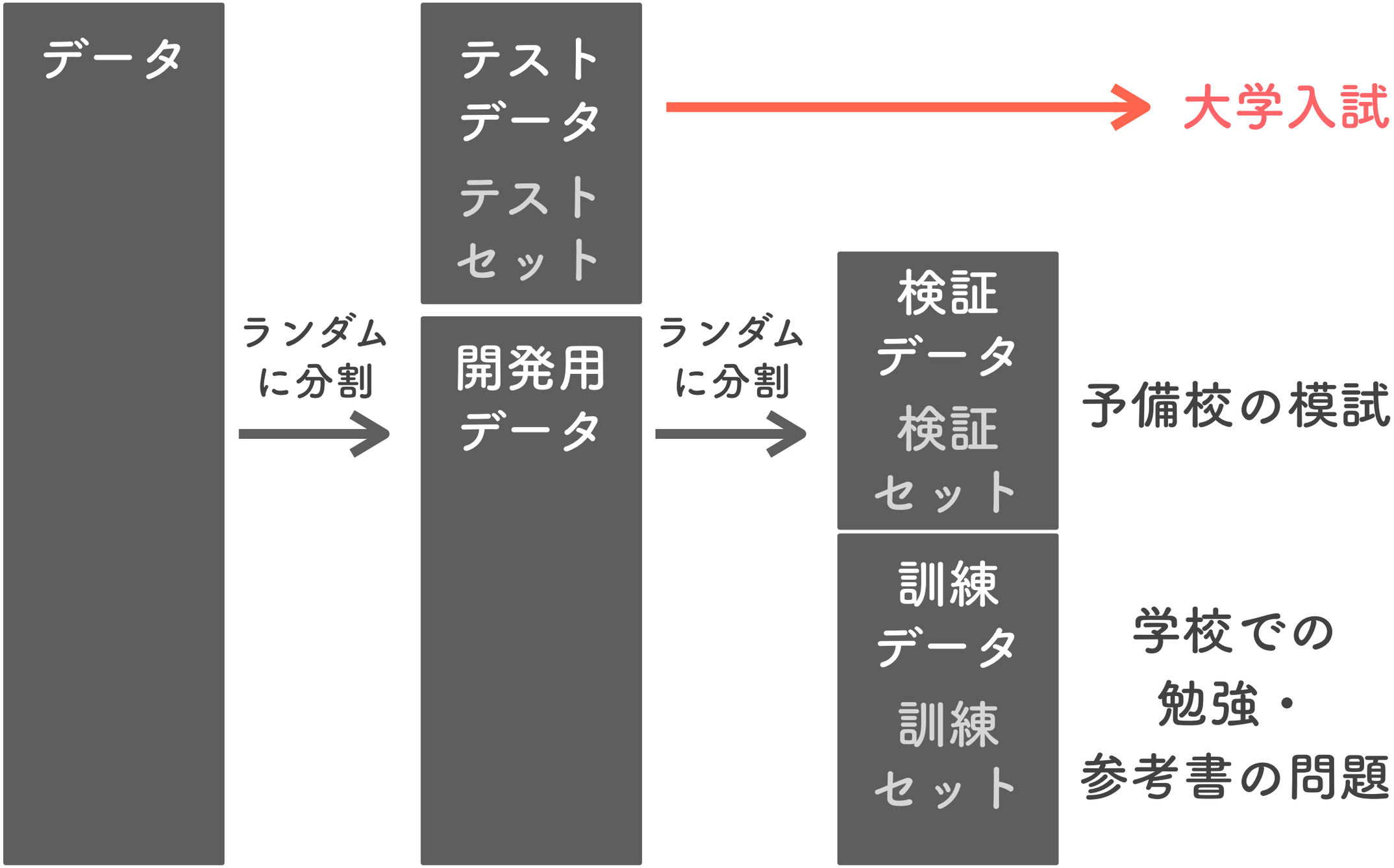
5. 汎化性能の簡単な推定法：ホールドアウト





5. 汎化性能の簡単な推定法：ホールドアウト



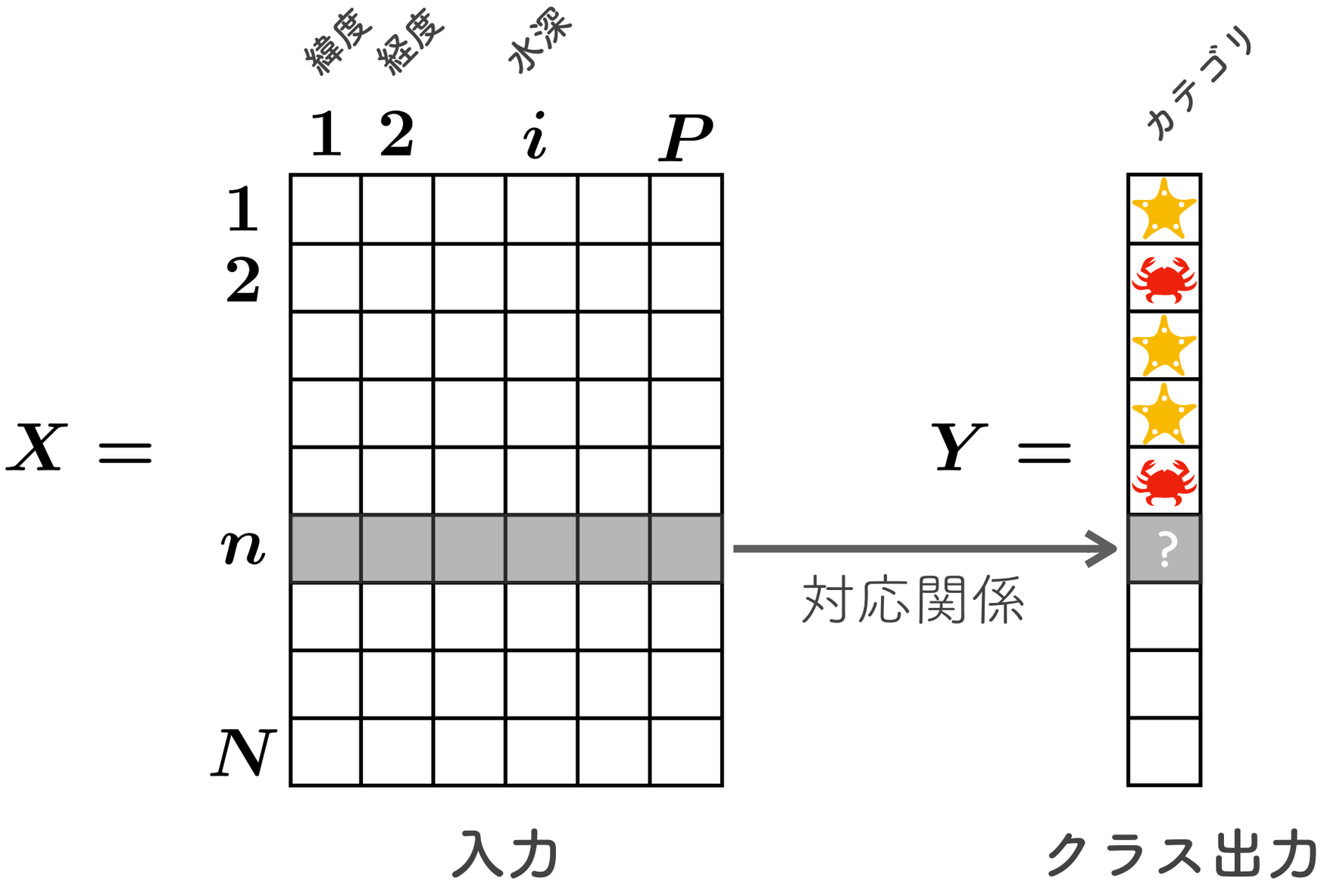


6. 2クラス分類（バイナリ分類）

入力属性から、**カテゴリカル**(非数値)**な出力**を予測

6. 2クラス分類（バイナリ分類）

入力属性から、**カテゴリカル**(非数値)**な出力**を予測



6. 2クラス分類（バイナリ分類）

入力属性から、**カテゴリカル**(非数値)**な出力**を予測

$$\vec{x} \longrightarrow y = \text{⭐} \ or \ \text{🦀}$$

2クラス

6. 2クラス分類（バイナリ分類）

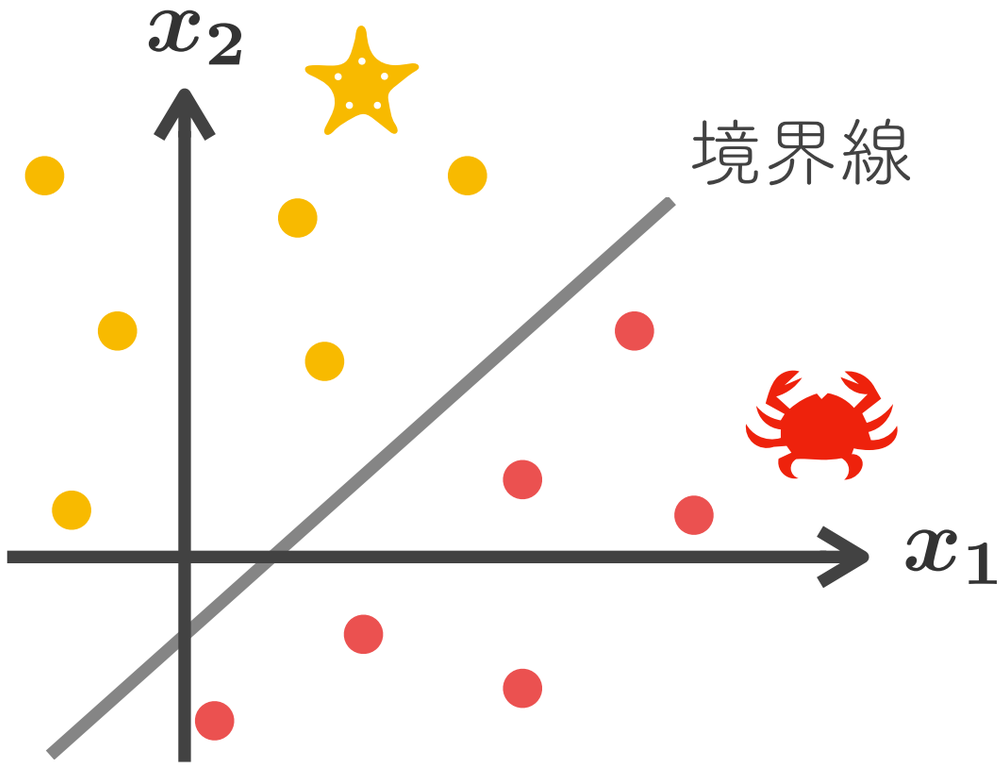
入力属性から、**カテゴリカル**(非数値)**な出力**を予測

$$\vec{x} \longrightarrow y = \text{★ or 🦀}$$

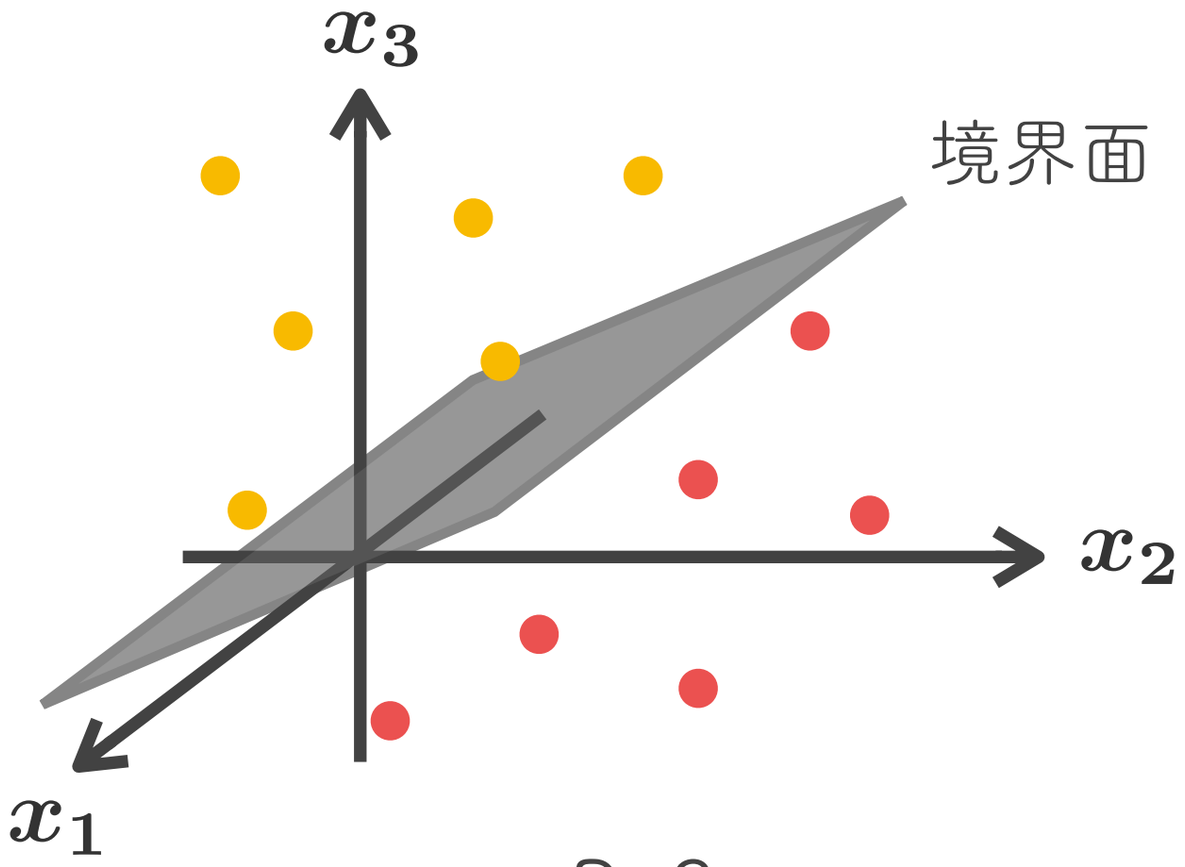
ダミー変数(**数値**)で表現 $y = 0, 1$

6. 2クラス分類（バイナリ分類）

分類は**決定境界**(decision boundary)を決める問題。
とりあえず線形な場合のみ考える。



P=2



P=3

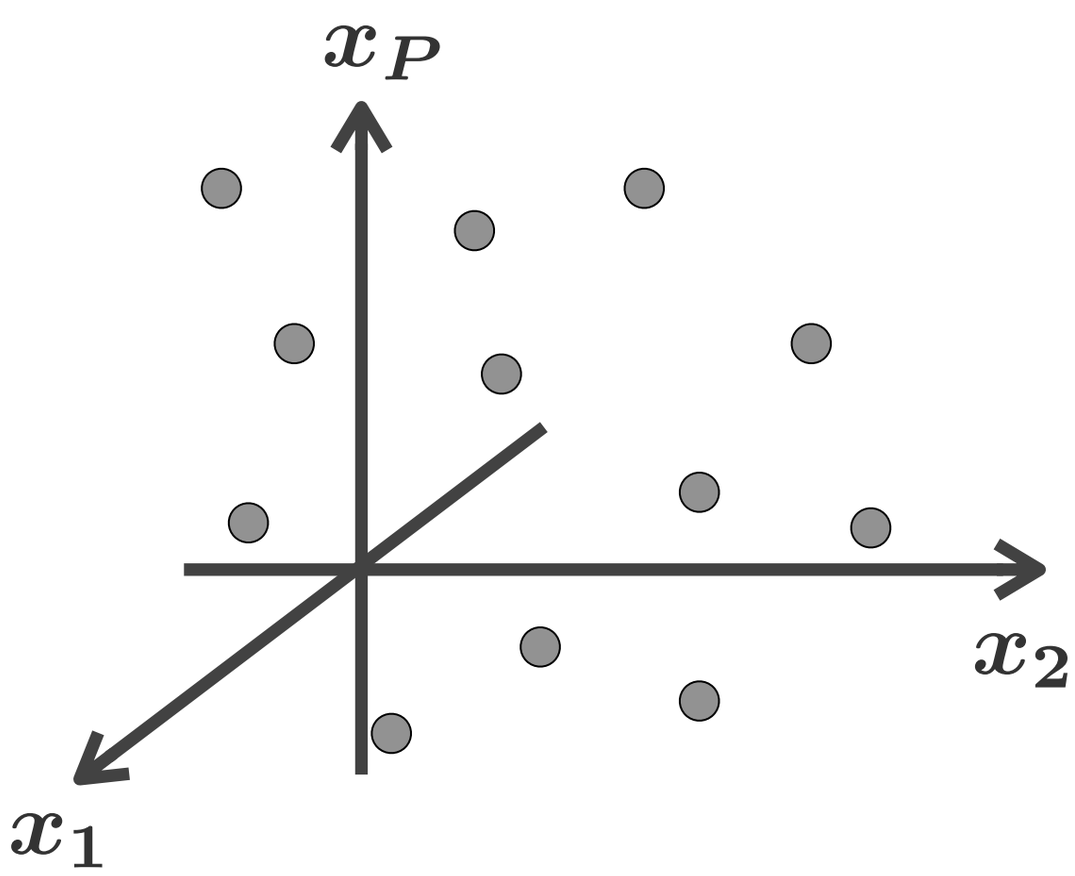
＊ここでxの添え字は、インスタンスのラベルではなく属性xの次元

6. 2クラス分類（バイナリ分類）

$\vec{x}$を与えた時、その入力がクラスyに所属する「確率」

$$P(y|\vec{x})$$

を推定すればいい。もし確率モデルが得られたら

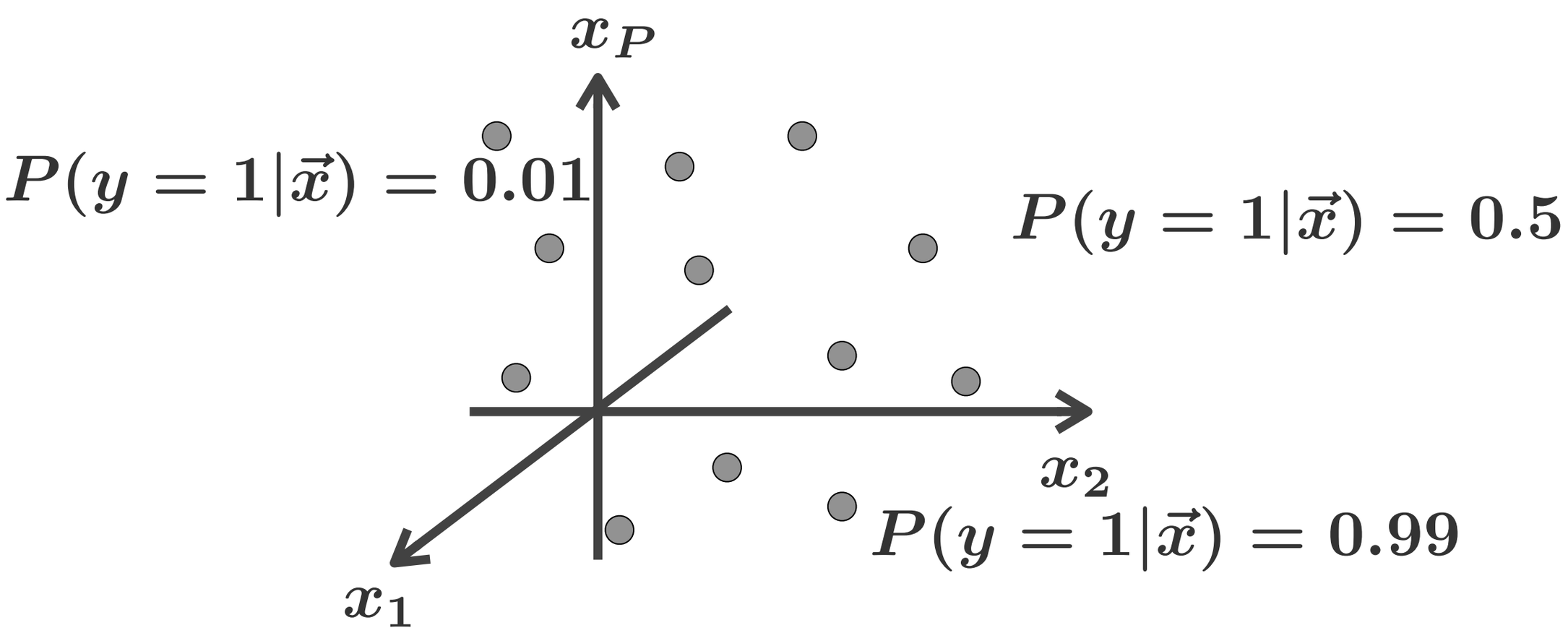


6. 2クラス分類（バイナリ分類）

$\vec{x}$を与えた時、その入力がクラスyに所属する「確率」

$$P(y|\vec{x})$$

を推定すればいい。もし確率モデルが得られたら

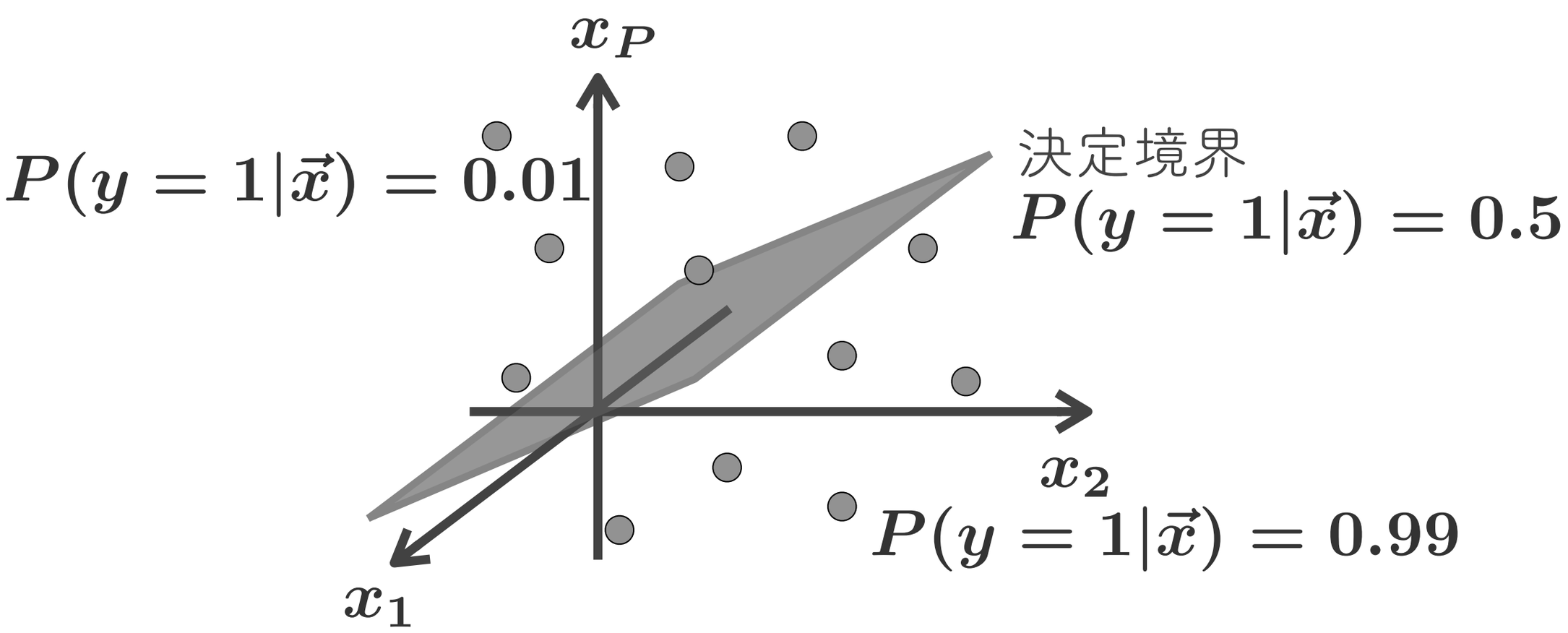


6. 2クラス分類（バイナリ分類）

$\vec{x}$を与えた時、その入力がクラスyに所属する「確率」

$$P(y|\vec{x})$$

を推定すればいい。もし確率モデルが得られたら

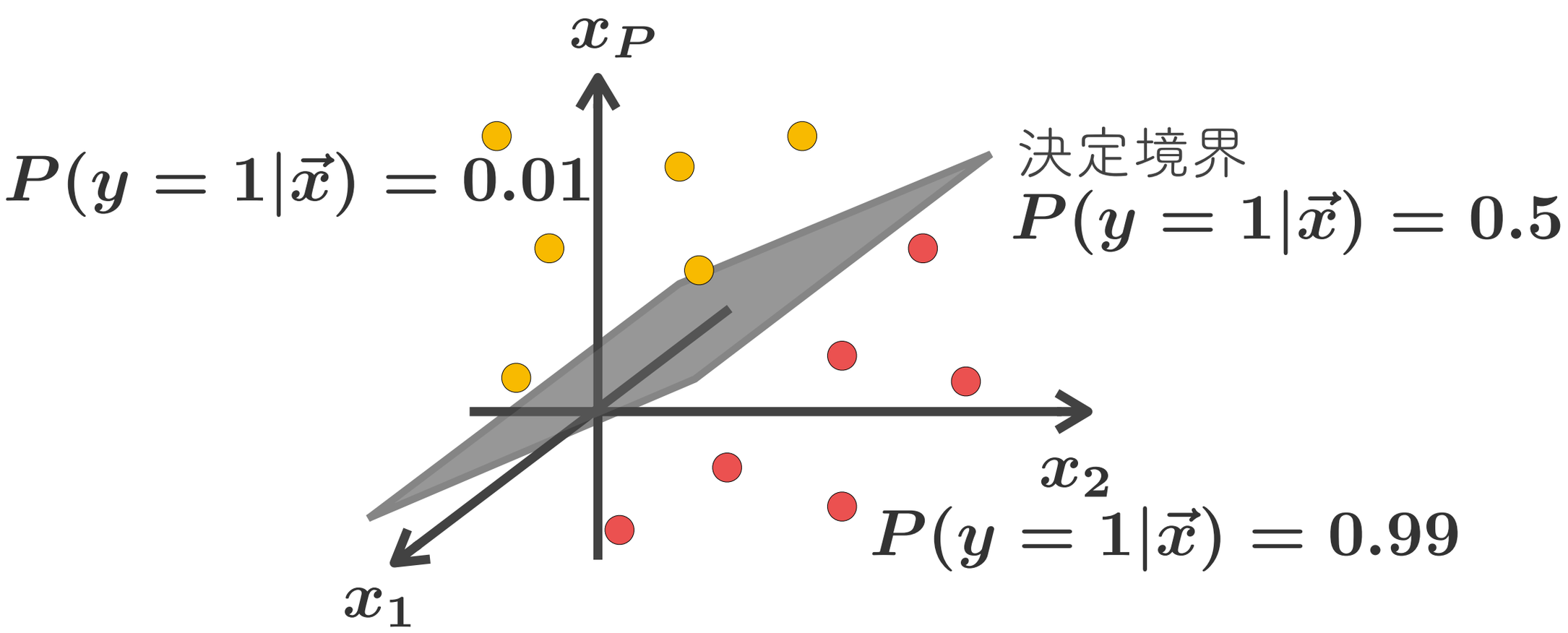


6. 2クラス分類（バイナリ分類）

$\vec{x}$を与えた時、その入力がクラスyに所属する「確率」

$$P(y|\vec{x})$$

を推定すればいい。もし確率モデルが得られたら



6. 2クラス分類（バイナリ分類）

簡単のため、**決定境界が線形になるモデル**を考えてみる。

この超平面を学習

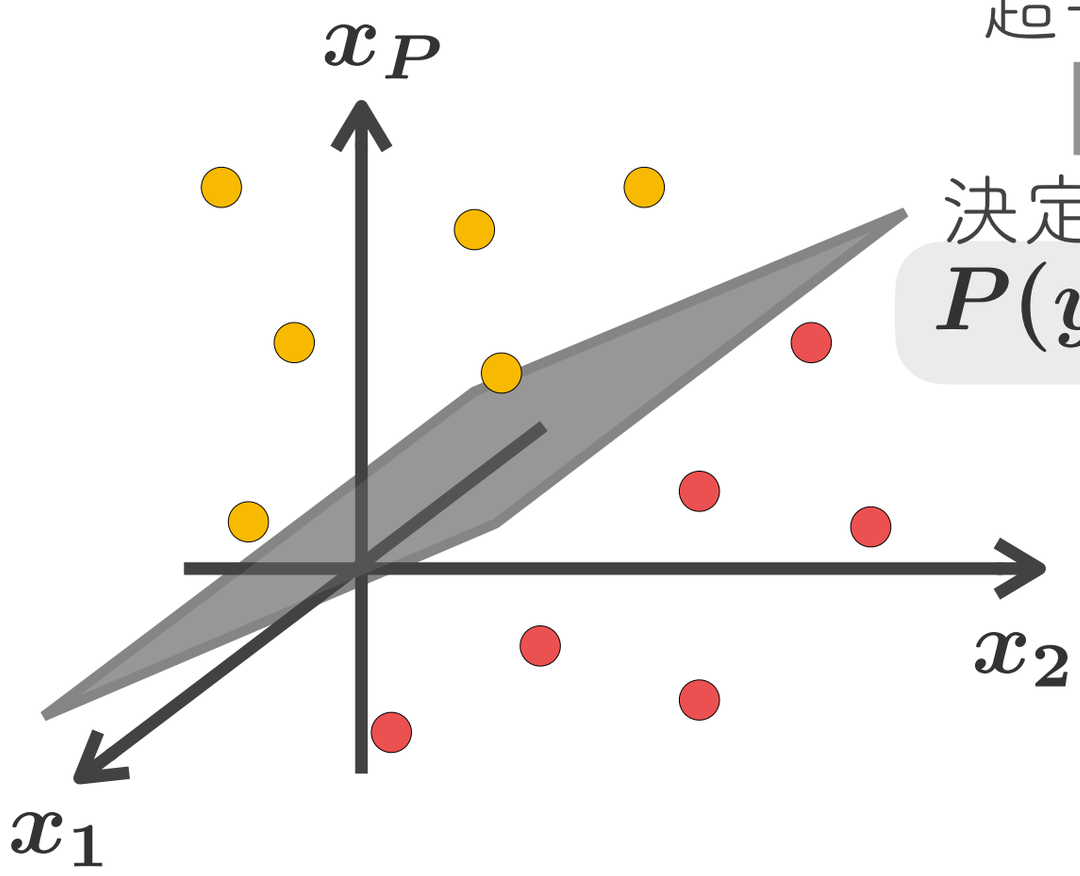
$$f(\vec{x}) = \vec{x}\vec{a} + b = 0$$

超平面

||

決定境界

$$P(y = 1|\vec{x}) = 0.5$$



7. ロジスティック回帰モデル

ロジスティック回帰モデル（logistic regression model）

$$P(1|\vec{x}) = \frac{1}{1 + e^{-(\vec{x}\,\vec{a}+b)}}$$

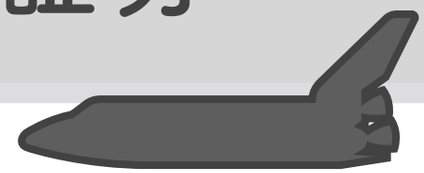
$$= \sigma(\vec{x}\,\vec{a} + b)$$

ここがゼロなら
決定境界

確率が線形モデルの**シグモイド**変換で与えられるモデル

$$P(0|\vec{x}) = 1 - P(1|\vec{x})$$

No Free Lunch 定理（トイバージョン）の証明



色々な帰納バイアスを持った学習アルゴリズム

$$\mathcal{A}_a \quad \mathcal{A}_b \quad \mathcal{A}_c \quad \cdots$$

を使って、分類パタンのground truth f を仮説

$$h \in \{0,1\}^{|\mathcal{X}|}$$

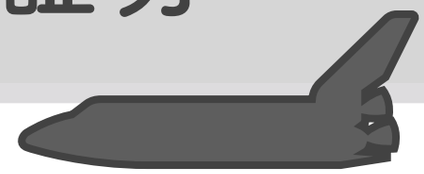
で近似できるように学習し、性能を比較したい。

$\mathcal{X}$ ：母集団（離散的）

$X \subset \mathcal{X}$ ：訓練データ（離散的）

$\mathcal{H}$ ：仮説空間。ラベル付与パタンなので $\{0,1\}^{|\mathcal{X}|}$

No Free Lunch 定理（トイバージョン）の証明



$\mathcal{A}_a$ がデータ $X \subset \mathcal{X}$ を学習し、h を得る確率

$$P(h|X, \mathcal{A}_a)$$

訓練データ以外で測ったアルゴリズム $\mathcal{A}_a$ の平均予測誤差は

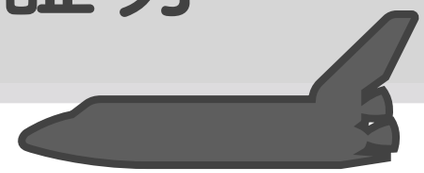
$$E_{OOD}[\mathcal{A}_a|X, f]$$

正解との不一致

$$= \sum_h \sum_{x \in \mathcal{X} \setminus X} P(x) P(h|X, \mathcal{A}_a) \delta_{f(x) \neq h(x)}$$

これを全てのタスク f（一様分布）で平均化すると

No Free Lunch 定理（トイバージョン）の証明



$$\sum_{f \in \{0,1\}^{|\mathcal{X}|}} E_{OOD}[\mathcal{A}_a | X, f]$$

$$= \sum_{x \in \mathcal{X} \setminus X} P(x) \sum_{h} P(h | X, \mathcal{A}_a) \sum_{f \in \{0,1\}^{|\mathcal{X}|}} \delta_{f(x) \neq h(x)}$$

$$= \mathbf{1} \qquad \frac{\mathbf{1}}{\mathbf{2}} \mathbf{2}^{|\mathcal{X}|}$$

$= \mathcal{A}_a$ に関係ない

あり得る全ての正解パタンを考えると、そのうち半分のは間違える

よってタスクで平均化するとアルゴリズムの性能には差がない

前半のまとめに変えて

紹介した機械学習の構成要素4つ：

データの入力表現 ← 前処理等、後述

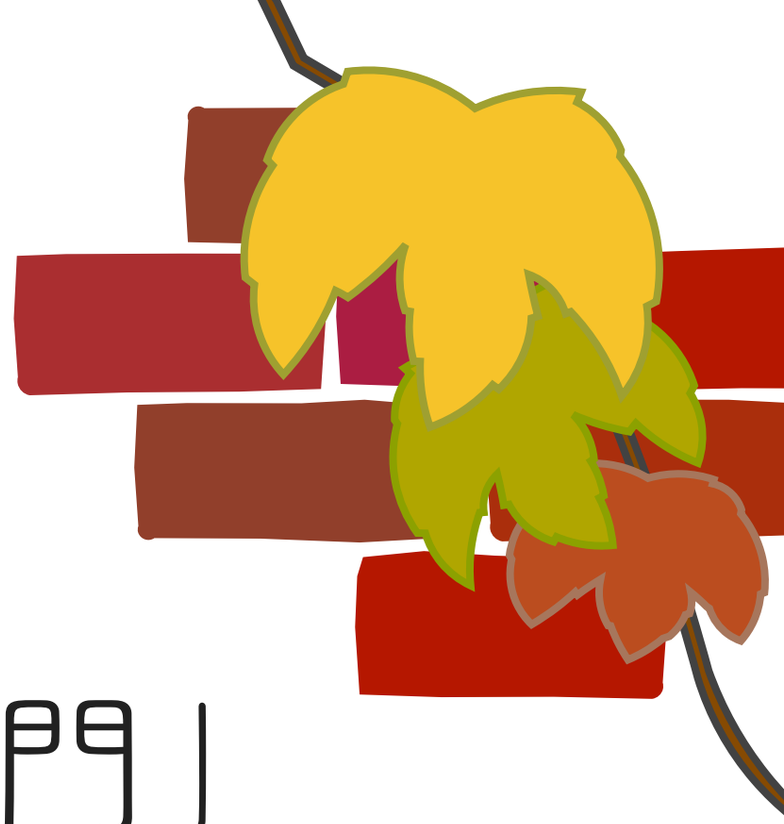
モデル ← 線形モデル

誤差関数（loss） ← 尤度

（大体）微分可能に作る

最適化手法（optimizer） ← 勾配効果法

データや具体的タスクと切り離して手法選択の良し悪しを議論することはできない（No Free Lunch）



講義「機械学習入門」

後半

立教大学 人工知能科学研究科

瀧 雅人

1. 特徴量エンジニアリング

1. 特徴量エンジニアリング

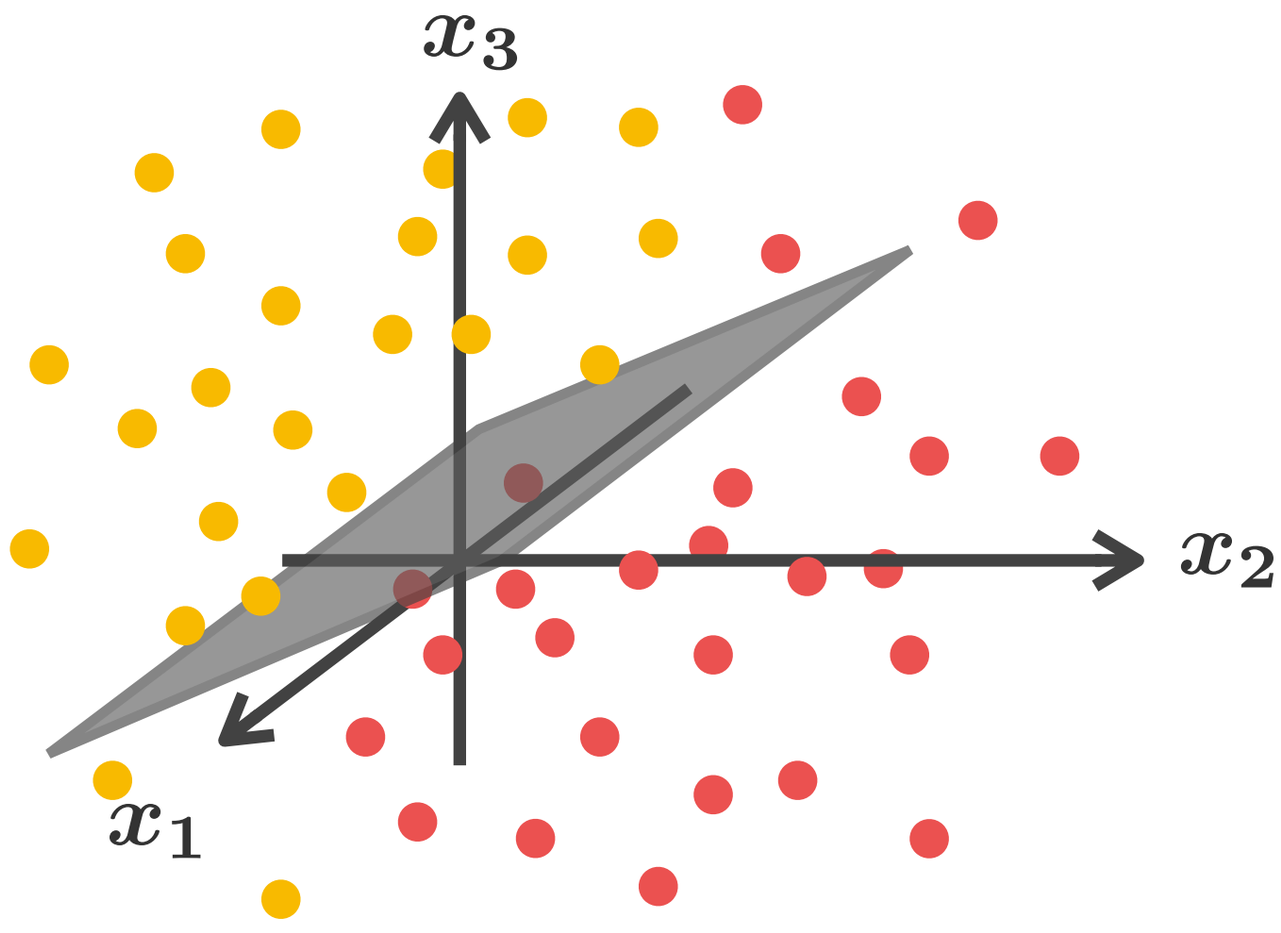
一般に、データの中の生の特徴量(変数)のままで良いわけではない。

$$X = \begin{array}{c} \begin{array}{cccccc} \mathbf{1} & \mathbf{2} & & \boldsymbol{i} & & \boldsymbol{P} \end{array} \\ \begin{array}{c} \mathbf{1} \\ \mathbf{2} \\ \\ \\ \boldsymbol{n} \\ \\ \\ \boldsymbol{N} \end{array} \begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} \end{array}$$

i番目の**特徴量**、**属性**

1. 特徴量エンジニアリング

・線形モデルでは単純なパターン認識しかできない



1. 特徴量エンジニアリング

・線形モデルでは単純なパターン認識しかできない

・人間の事前知識を組み込む余地

予測：**生徒のやる気、教師の能力、…　→　生徒の成長**

→ **(生徒のやる気)＊(教師の能力)**を変数として加える

ANDの寄与(交互作用)

1. 特徴量エンジニアリング

・線形モデルでは単純なパターン認識しかできない

・人間の事前知識を組み込む余地

・次元の呪いを被らないように次元圧縮

1. 特徴量エンジニアリング

より高レベルの**特徴量(表現)**を人間の知識で設計

$$\vec{x} \longrightarrow \vec{\phi}(\vec{x})$$

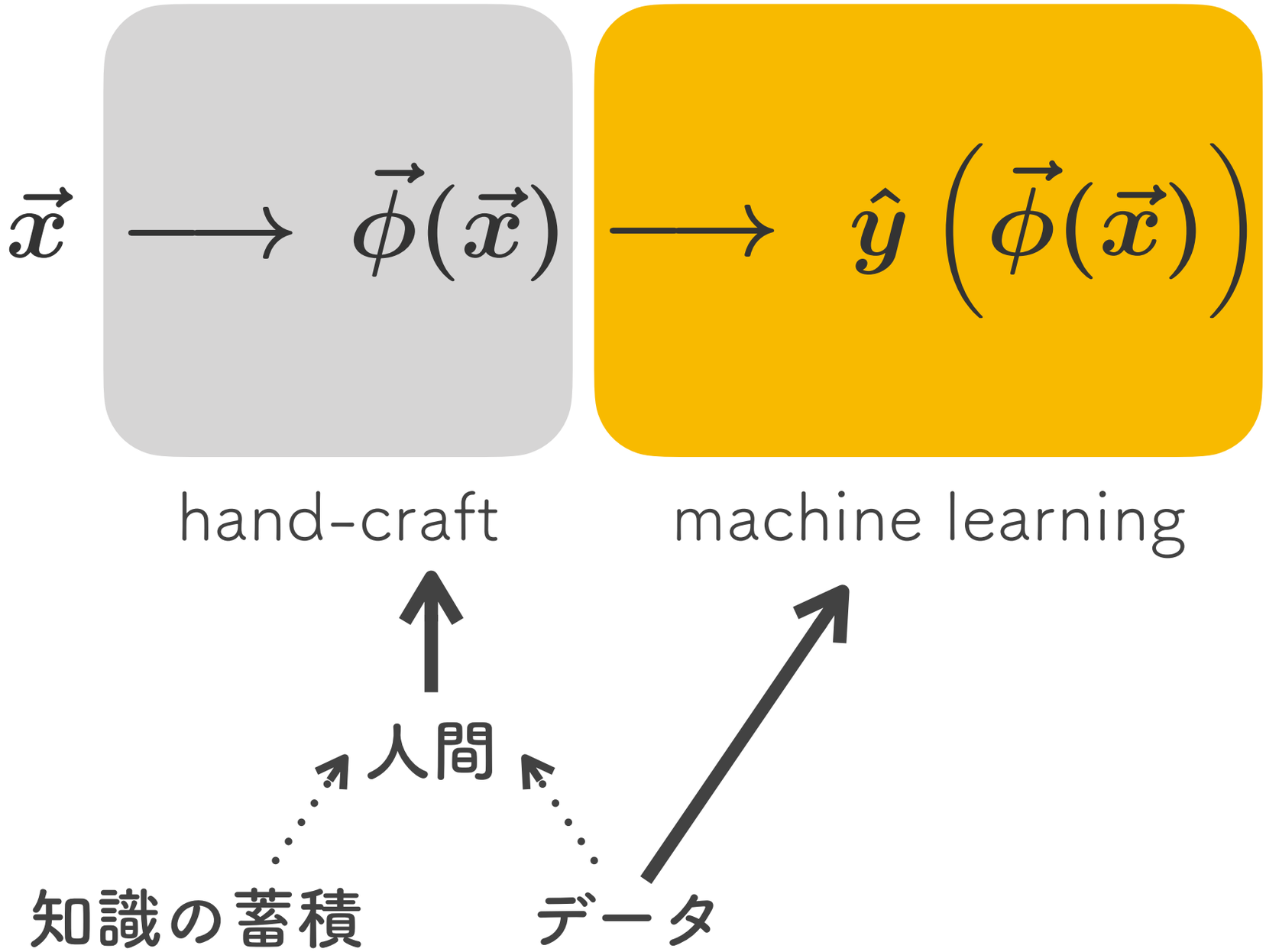
例）多項式回帰（PolynomialFeatures）, degree=3

$$x \longrightarrow \phi_1(x), \phi_2(x), \phi_3(x)$$

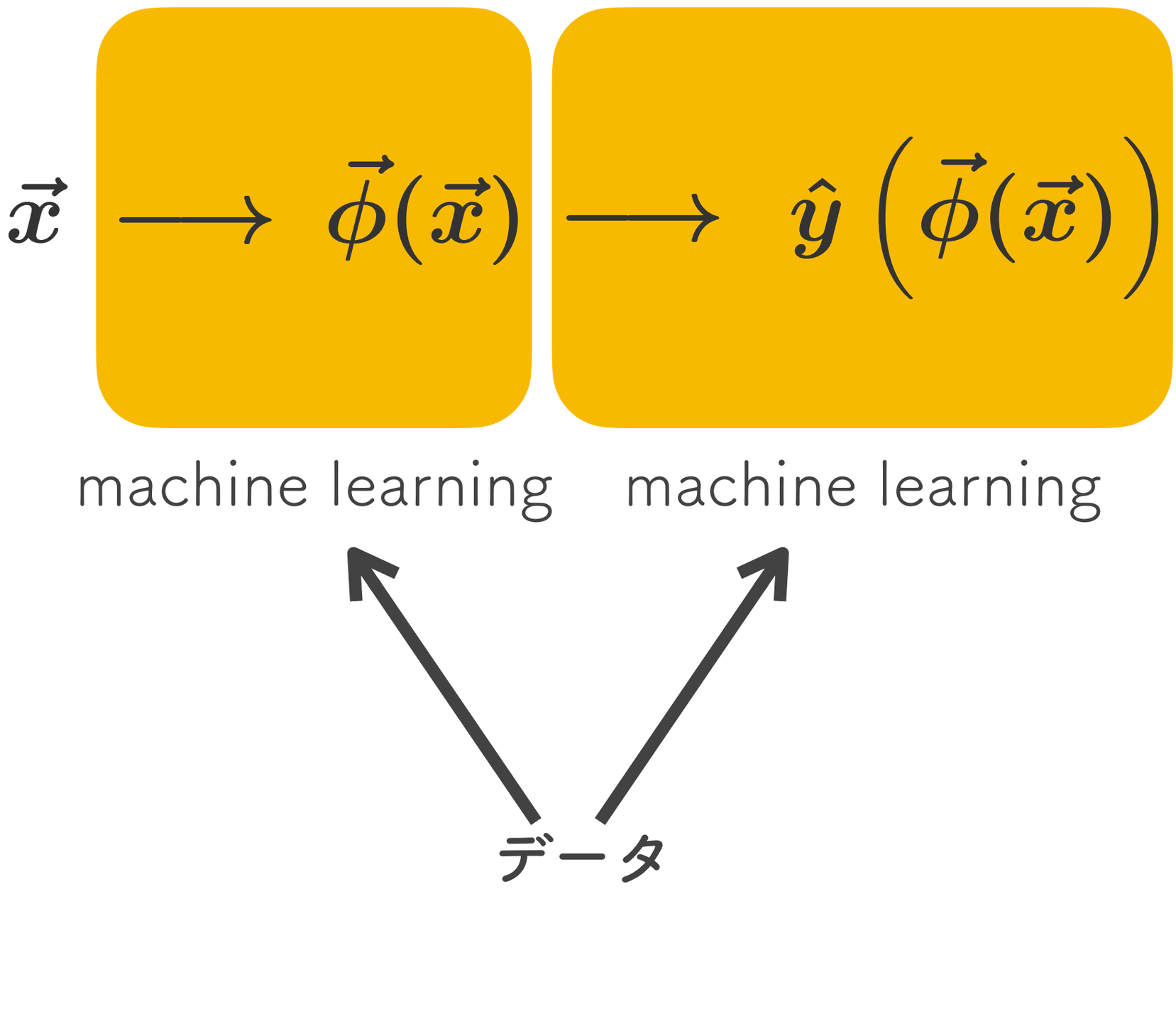
$$\begin{aligned}\phi_1(x) &= x \\ \phi_2(x) &= x^2 \\ \phi_3(x) &= x^3\end{aligned}$$

→ 複雑な識別境界、多項式回帰

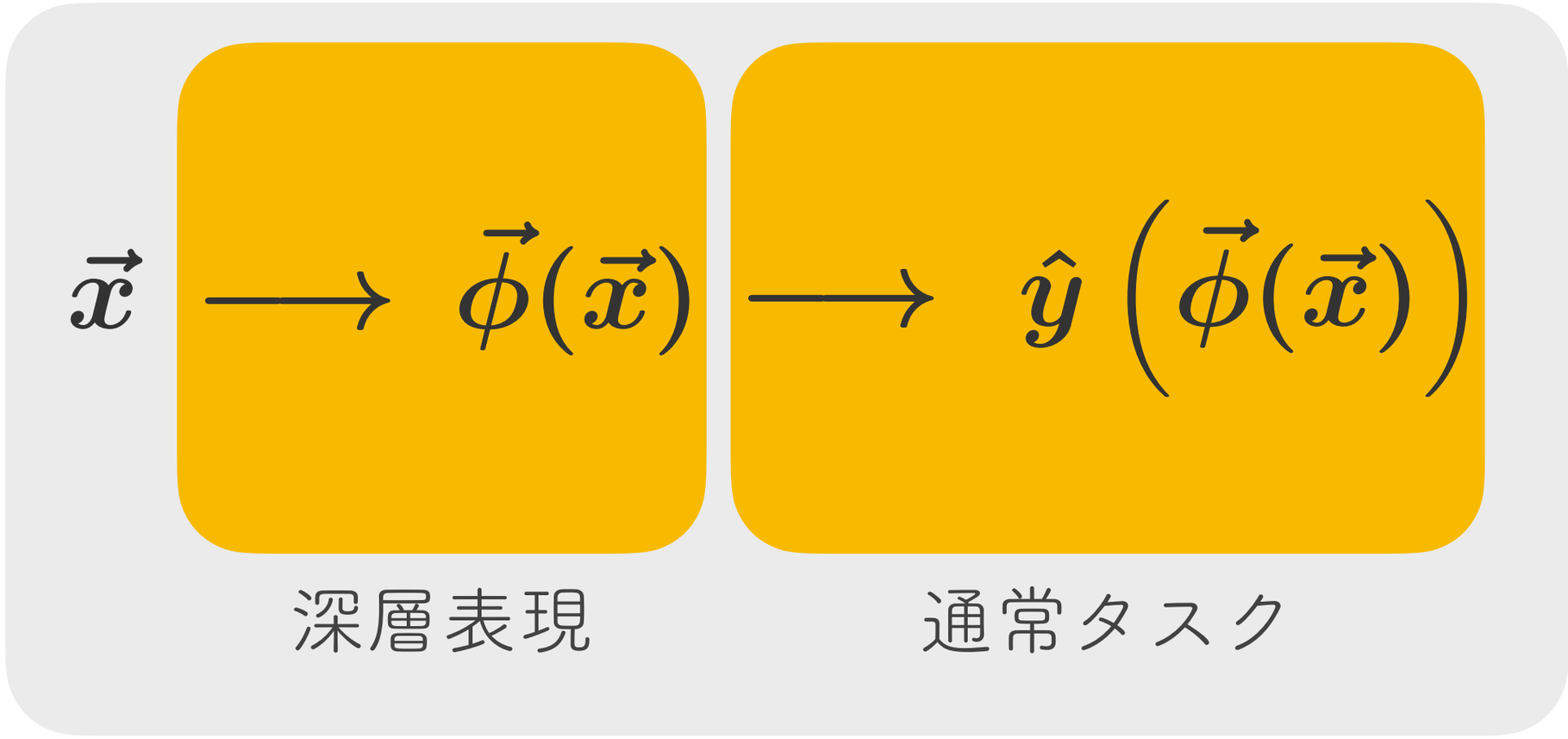
1. 特徴量エンジニアリング



2. 表現学習


$\vec{x}$
$\vec{\phi}(\vec{x})$
$\hat{y}\left(\vec{\phi}(\vec{x})\right)$
machine learning
machine learning
データ

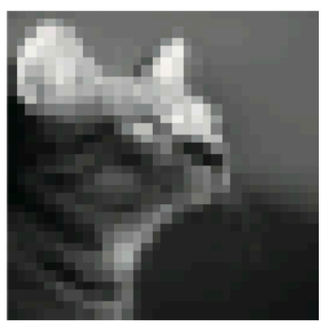
2. 深層学習による深層表現学習

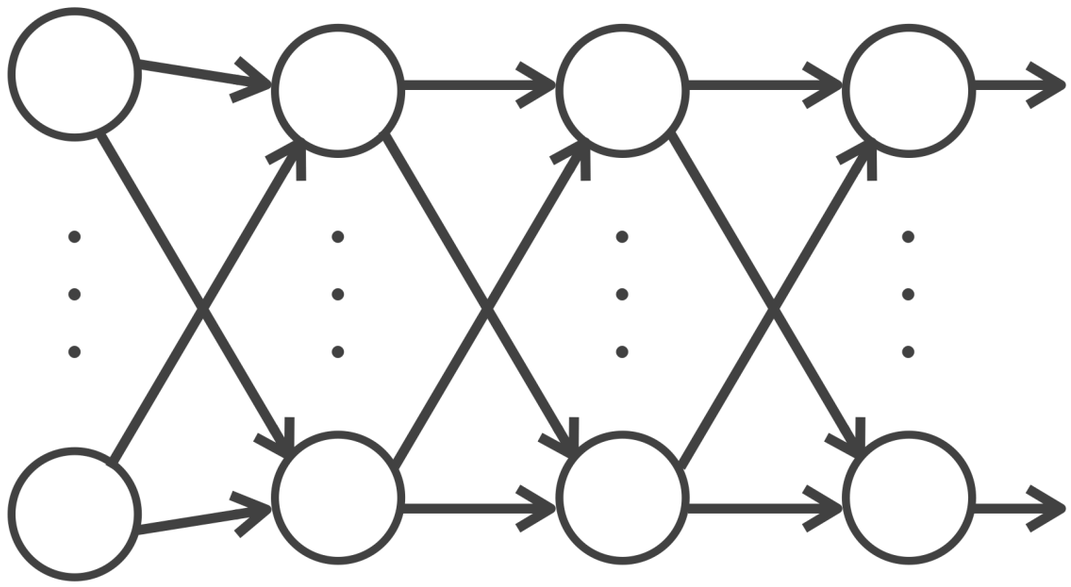


深層ニューラルネット（DNN）

深い情報表現の学習と、与えられたタスクの機械学習が、**end-to-endで**行われる。

2. 深層学習による深層表現学習





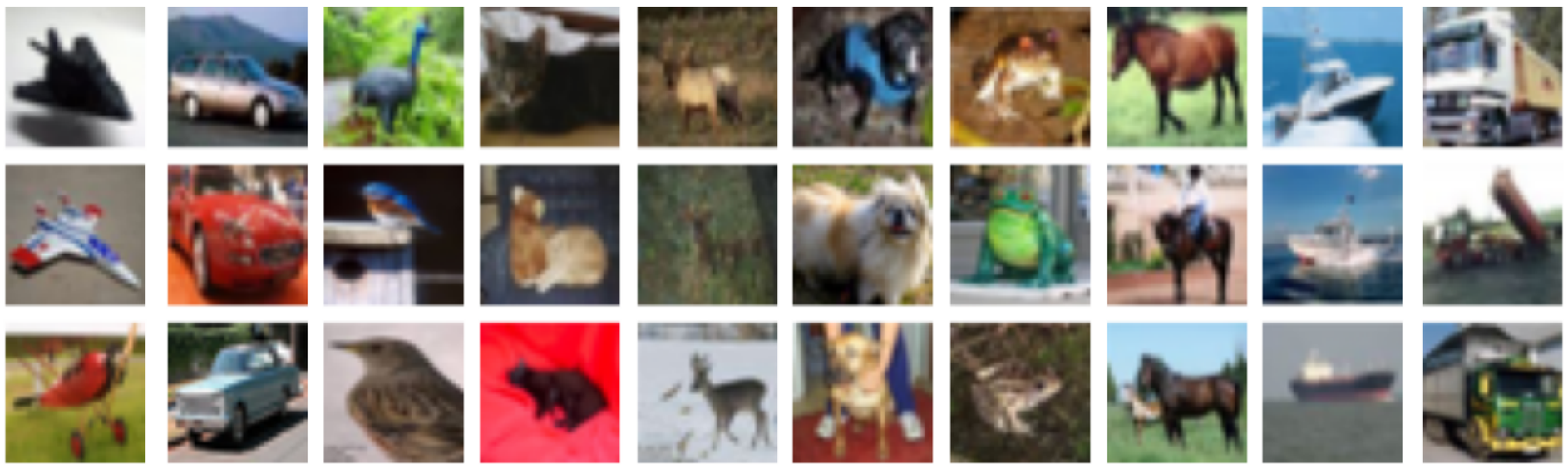
$P_{airplane} = 0.0001$

10クラスの確率

$P_{car} = 0.0002$

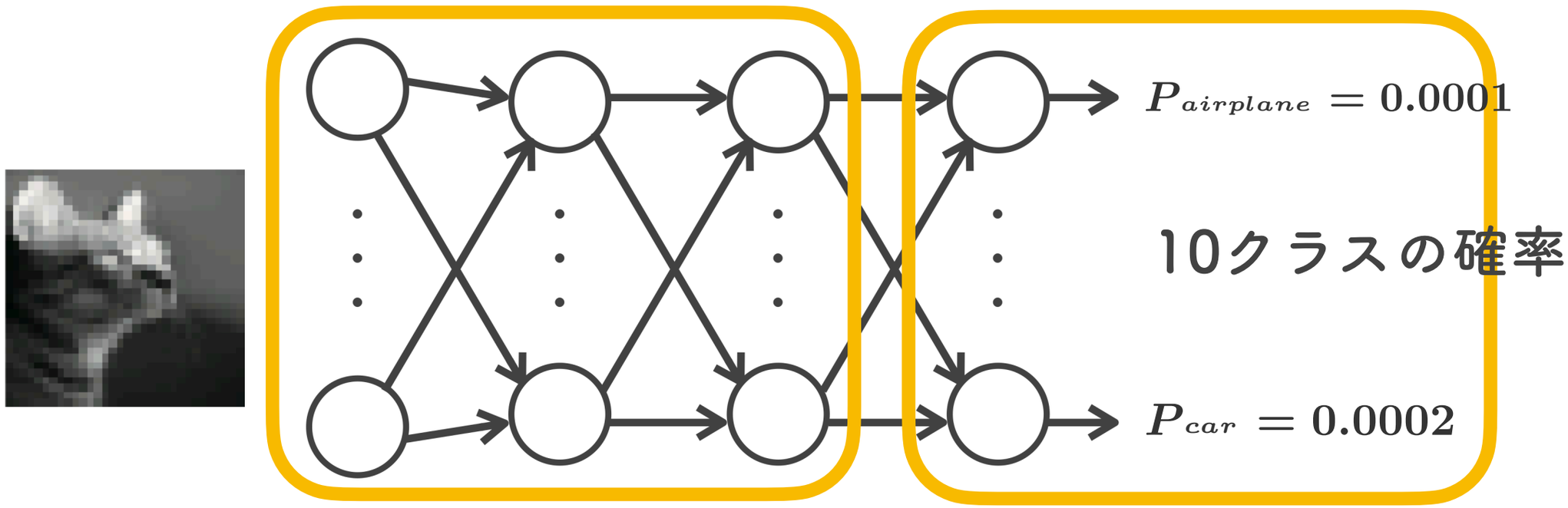
画像分類を学習

飛行機　車　鳥　猫　鹿　犬　カエル　馬　船　トラック



CIFAR10データセット

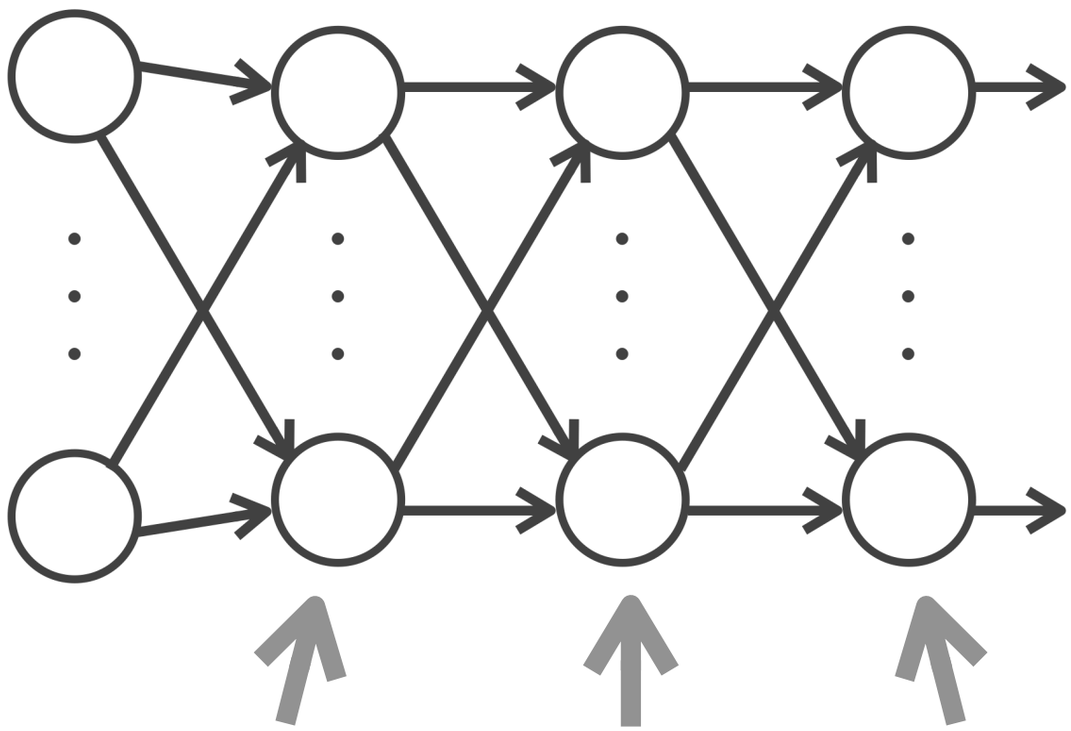
2. 深層学習による深層表現学習

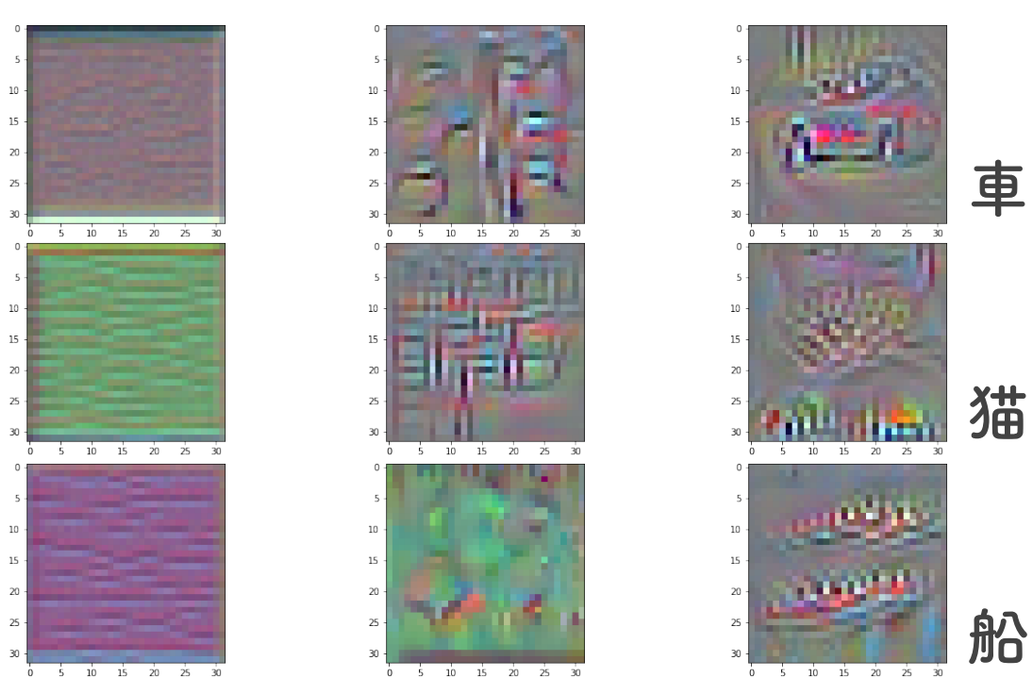


画像分類を学習

CIFAR10データセット

2. 深層学習による深層表現学習





各層の変数たちが強く反応する入力パターン例を探索

低次の視覚概念・パターン

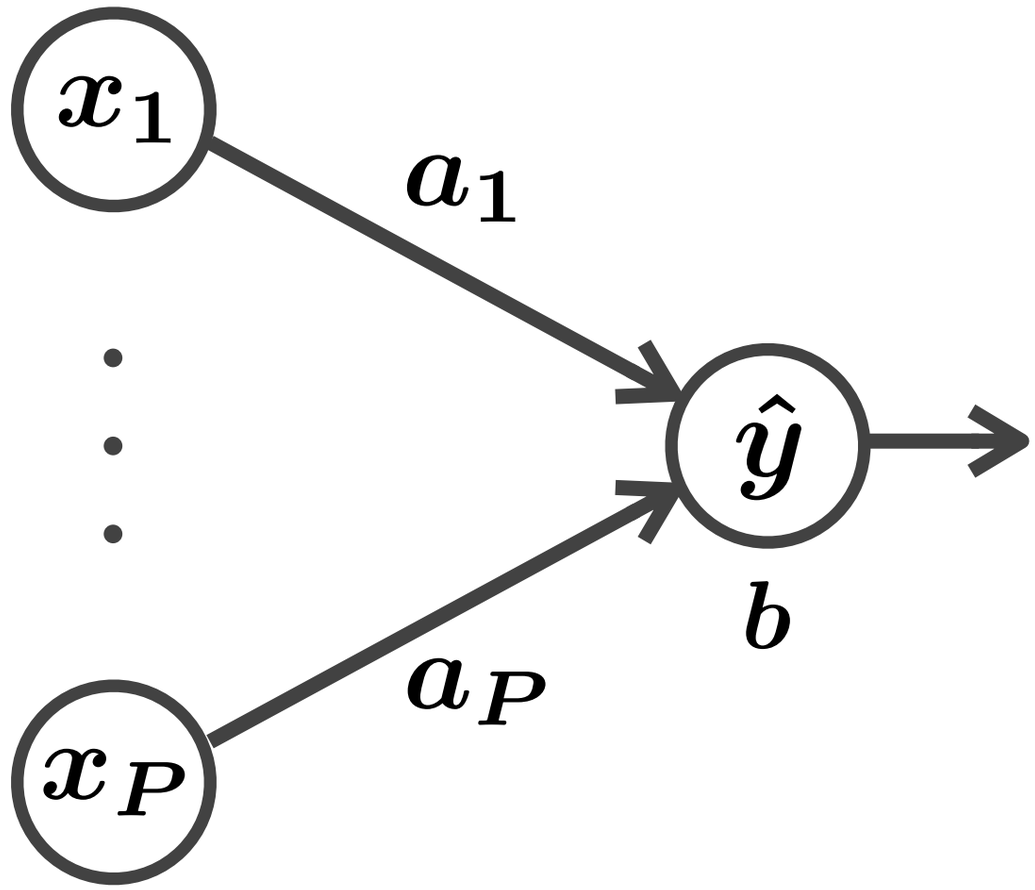
高次の視覚概念・パターン

2. ニューラルネット

1. 重回帰モデルとグラフ

$$\hat{y}(x) = a_1 x_1 + \cdots a_P x_P + b$$

$$\hat{y} = \vec{x}\,\vec{a} + b$$

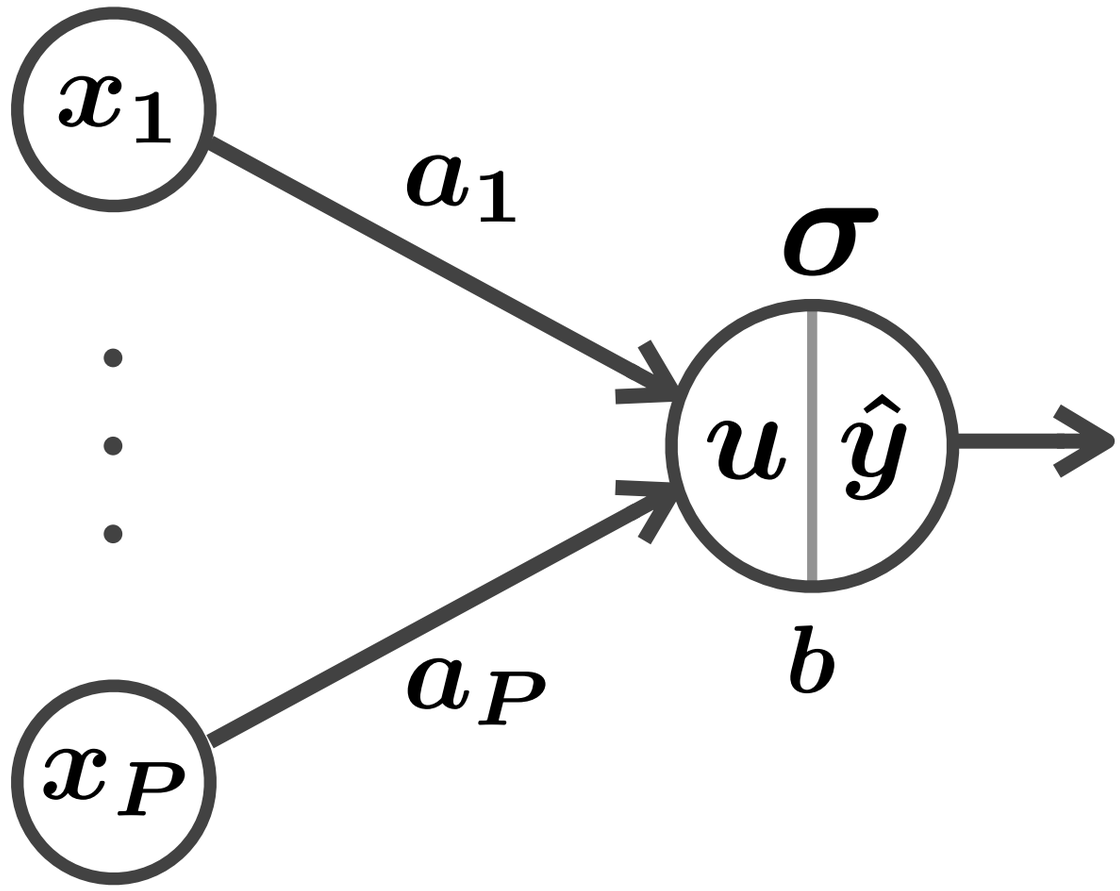


1. ロジスティック回帰モデルとグラフ

$$\hat{y}(x) = \sigma\left(a_1 x_1 + \cdots a_P x_P + b\right)$$

$$\hat{y} = \sigma(u)$$

$$u = \vec{x}\,\vec{a} + b$$

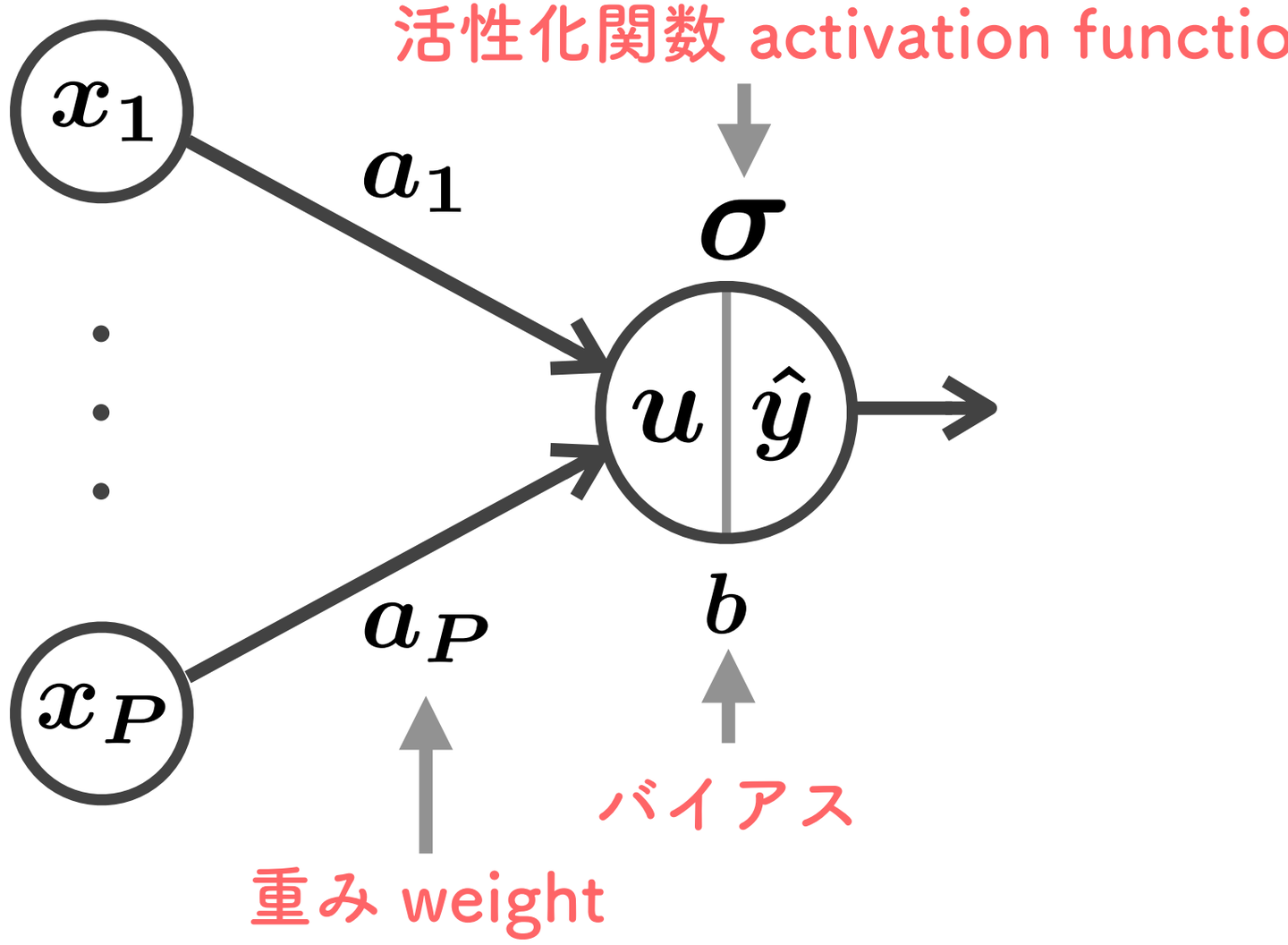


1. ロジスティック回帰モデルとグラフ

$$\hat{y}(x) = \sigma\left(a_1 x_1 + \cdots a_P x_P + b\right)$$

$$\hat{y} = \sigma(u)$$

$$u = \vec{x}\,\vec{a} + b$$

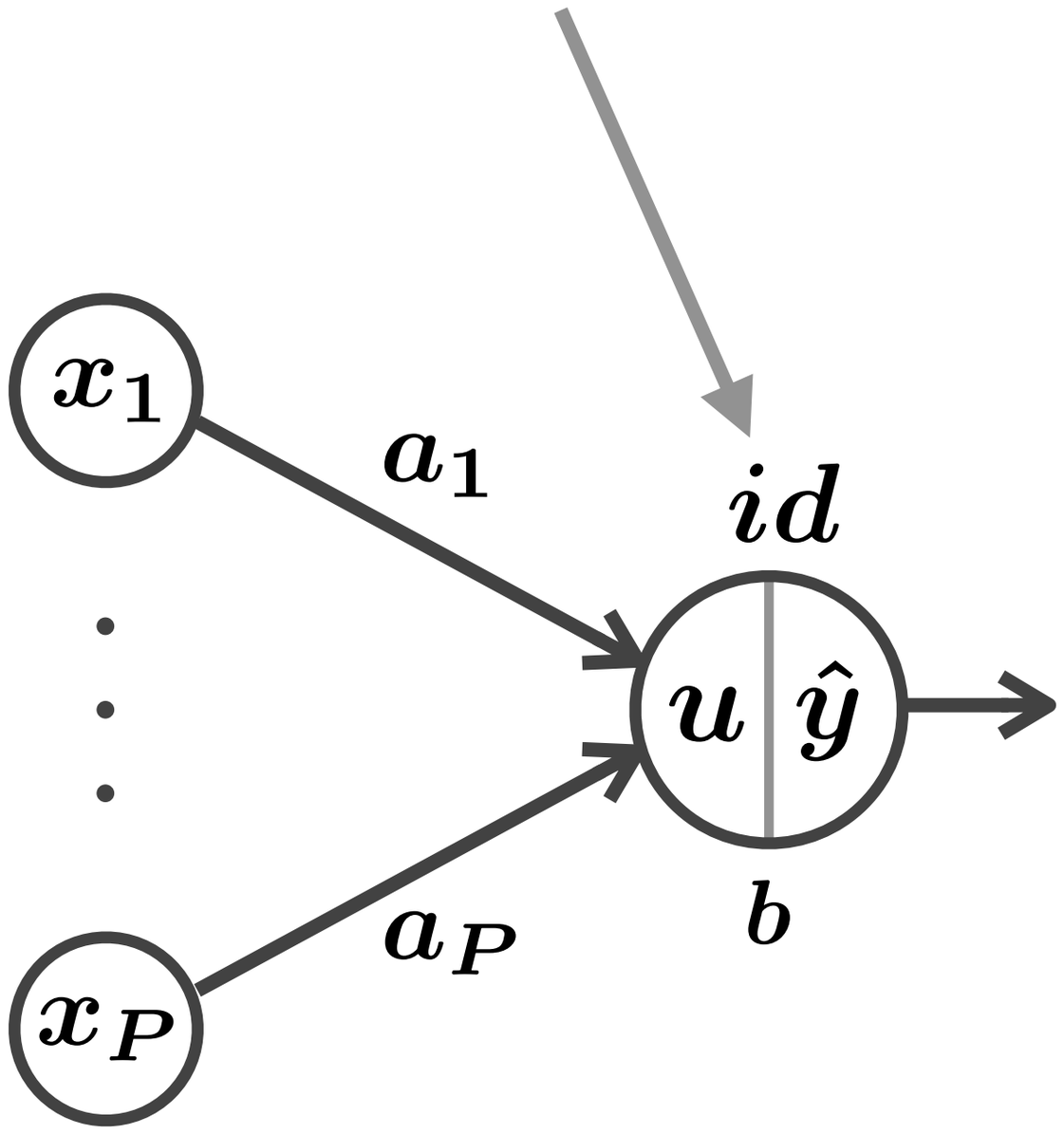


1. 重回帰モデルとグラフ

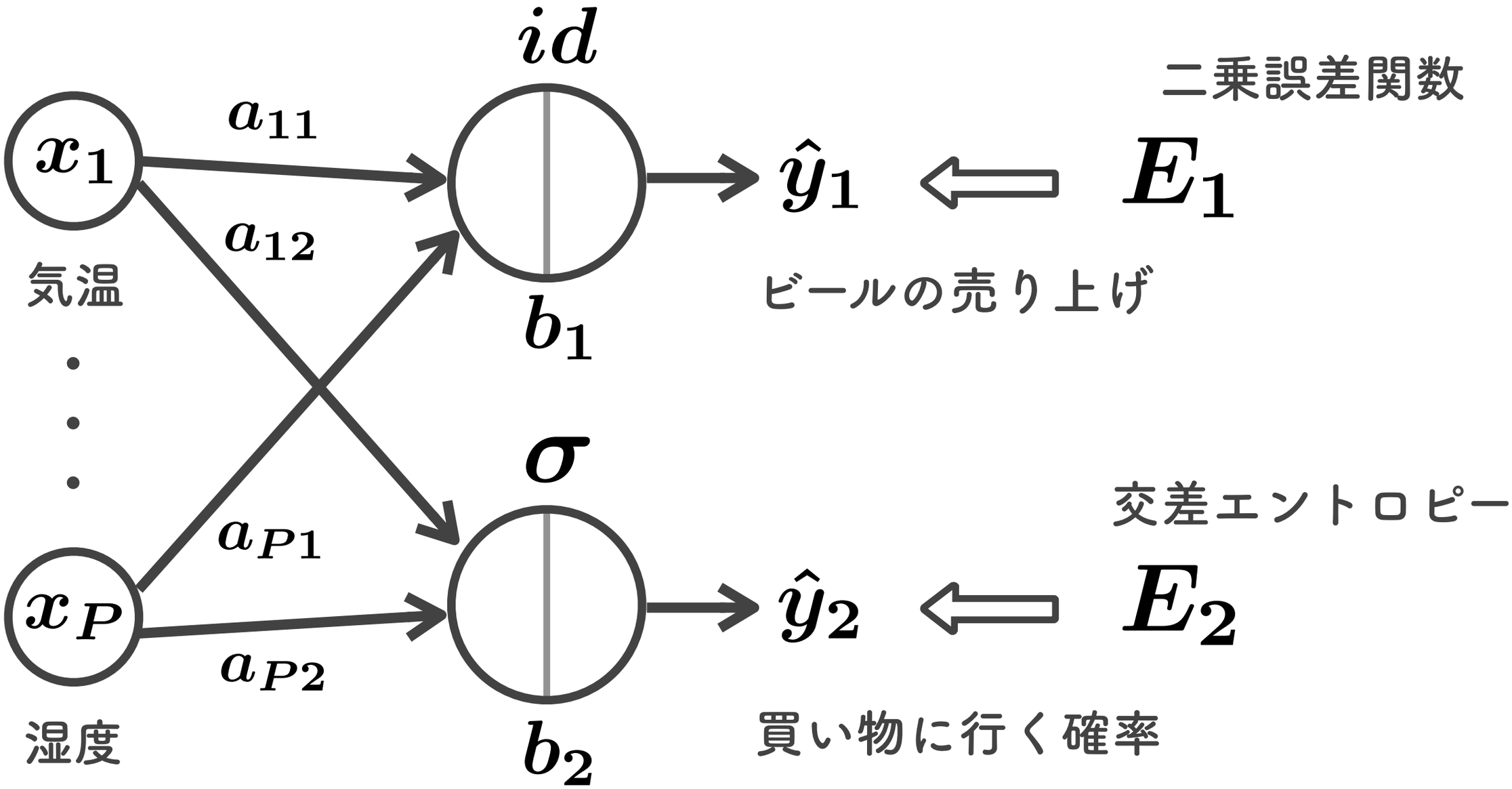
恒等写像 identity map（何もしない $id(u) = u$ ）

$$\hat{y} = u$$

$$u = \vec{x}\,\vec{a} + b$$

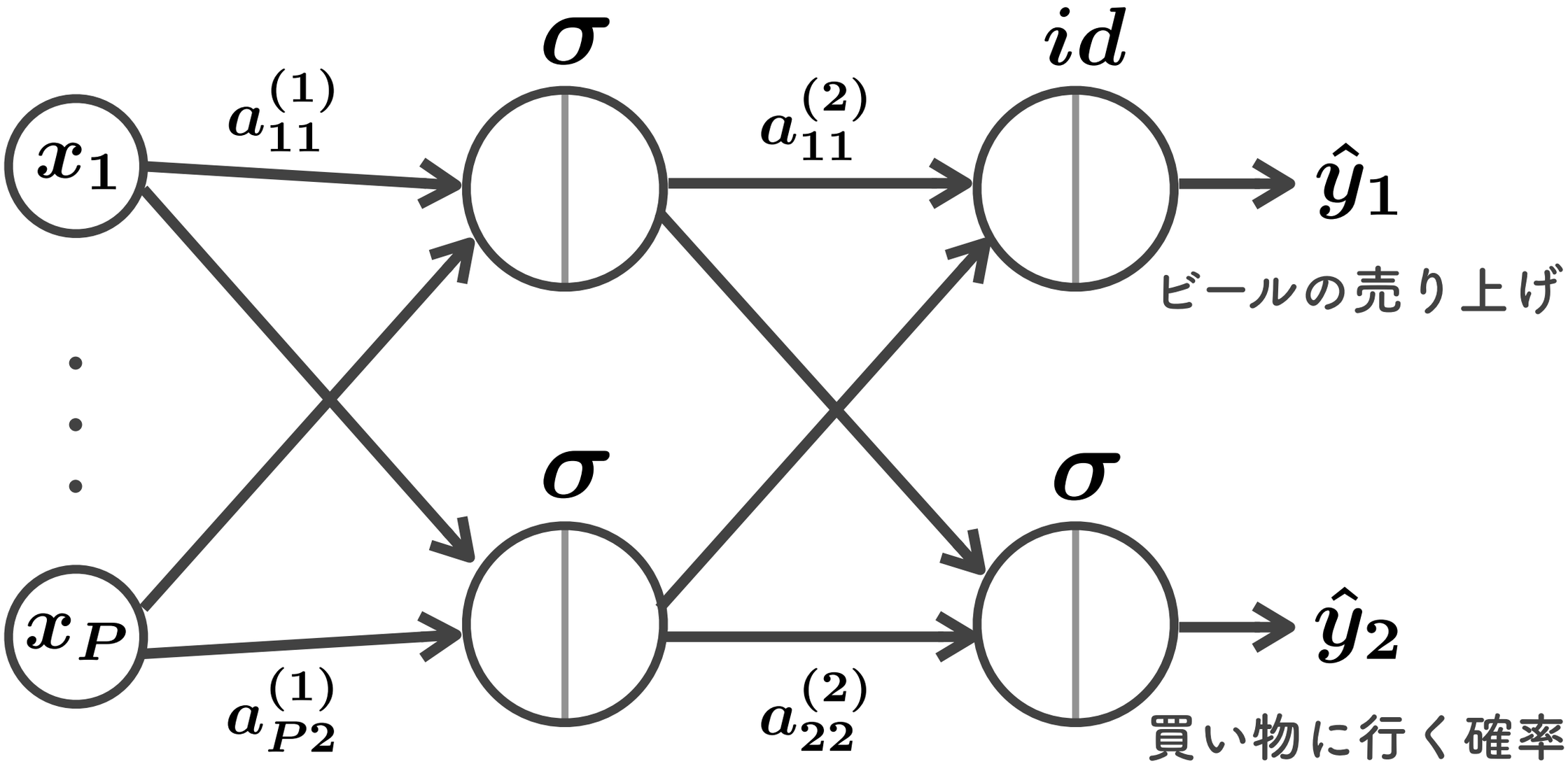


2. グラフを使ってモデルを設計する



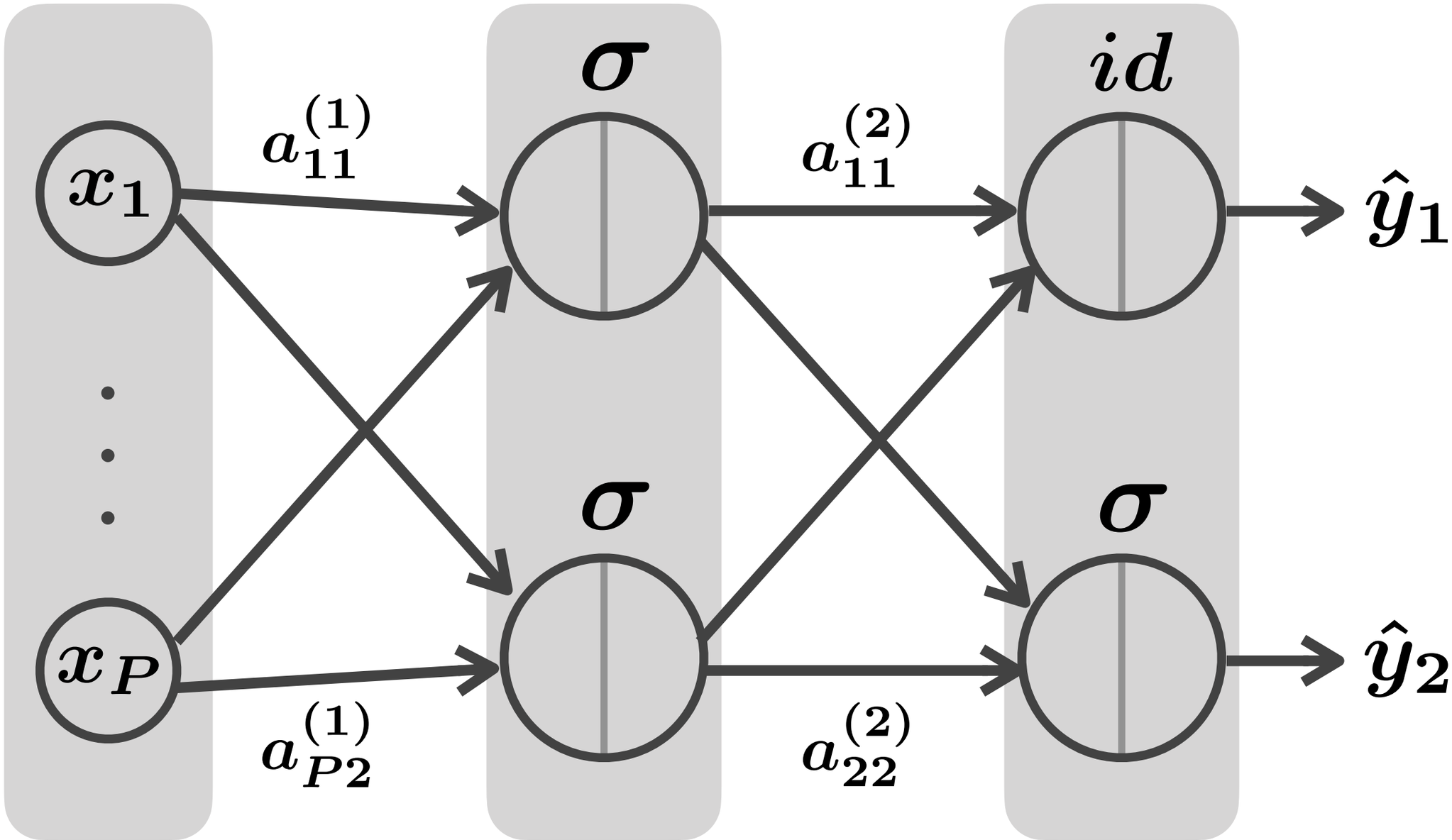
$$E = E_1 + E_2 \text{ を最小化}$$

3. 多層ニューラルネット（2層）



順伝搬ニューラルネット feedforward neural network

3. 多層ニューラルネット（２層）

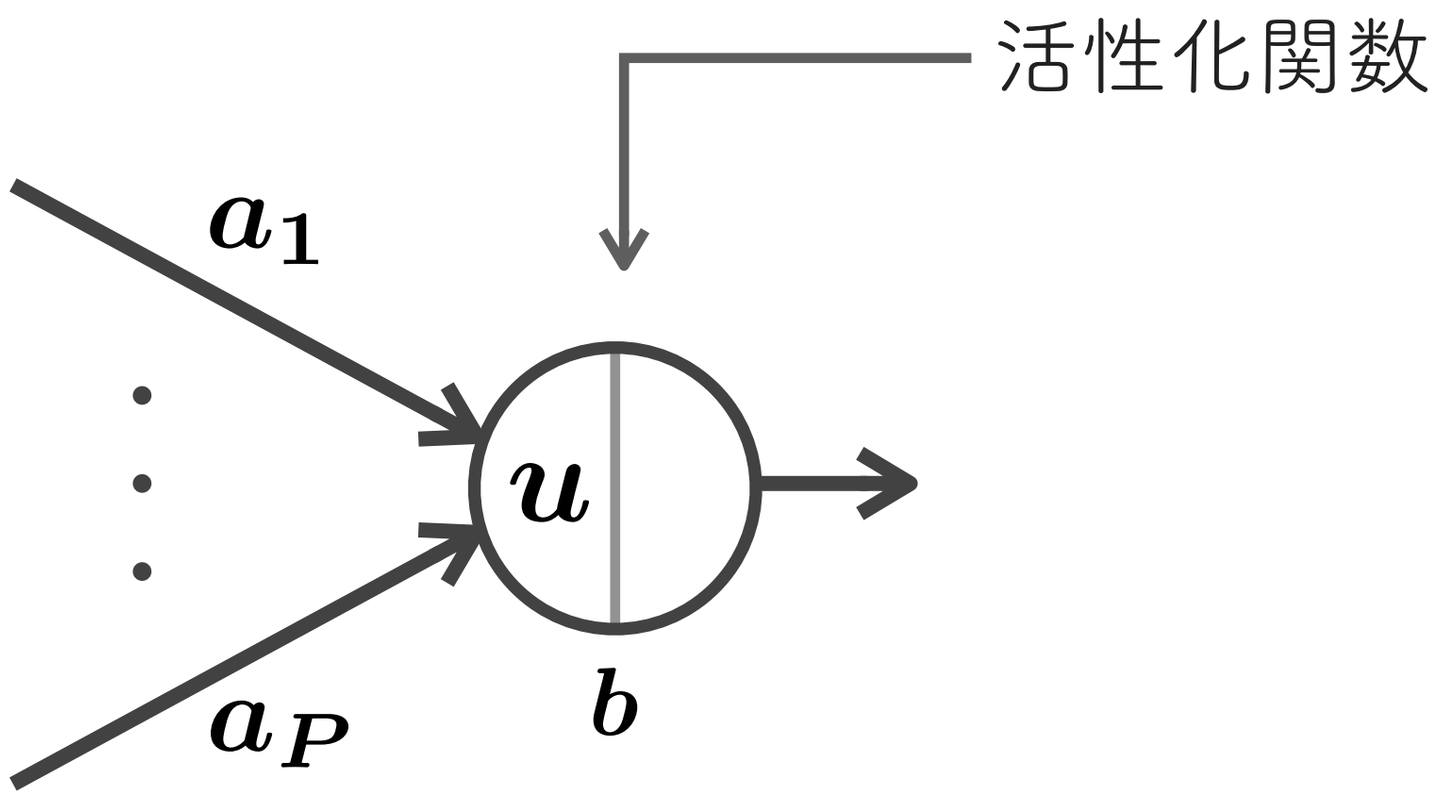


入力層
$\ell = 0$

中間層、隠れ層
$\ell = 1$

出力層
$\ell = 2$

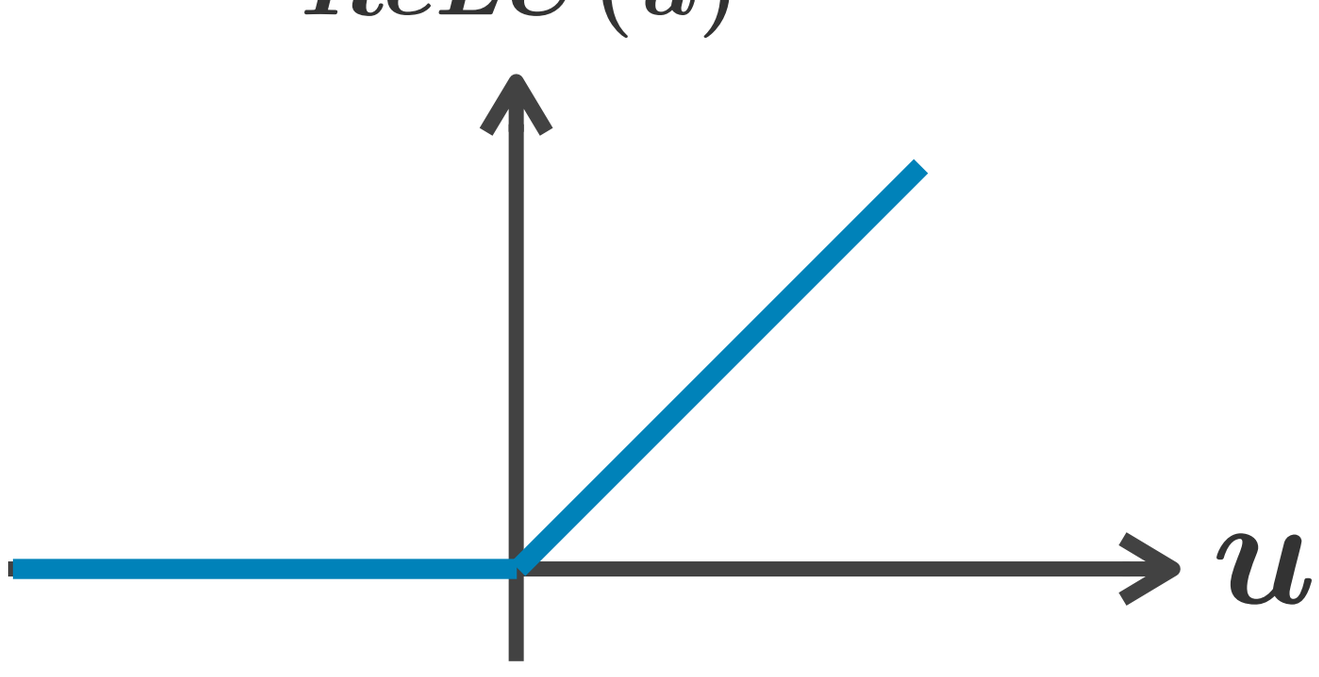
3. 活性化関数



ユニット

3. ReLU (rectifier linear unit) 活性関数

$$ReLU(u) = \begin{cases} u & (u \geq 0) \\ 0 & (u < 0) \end{cases}$$



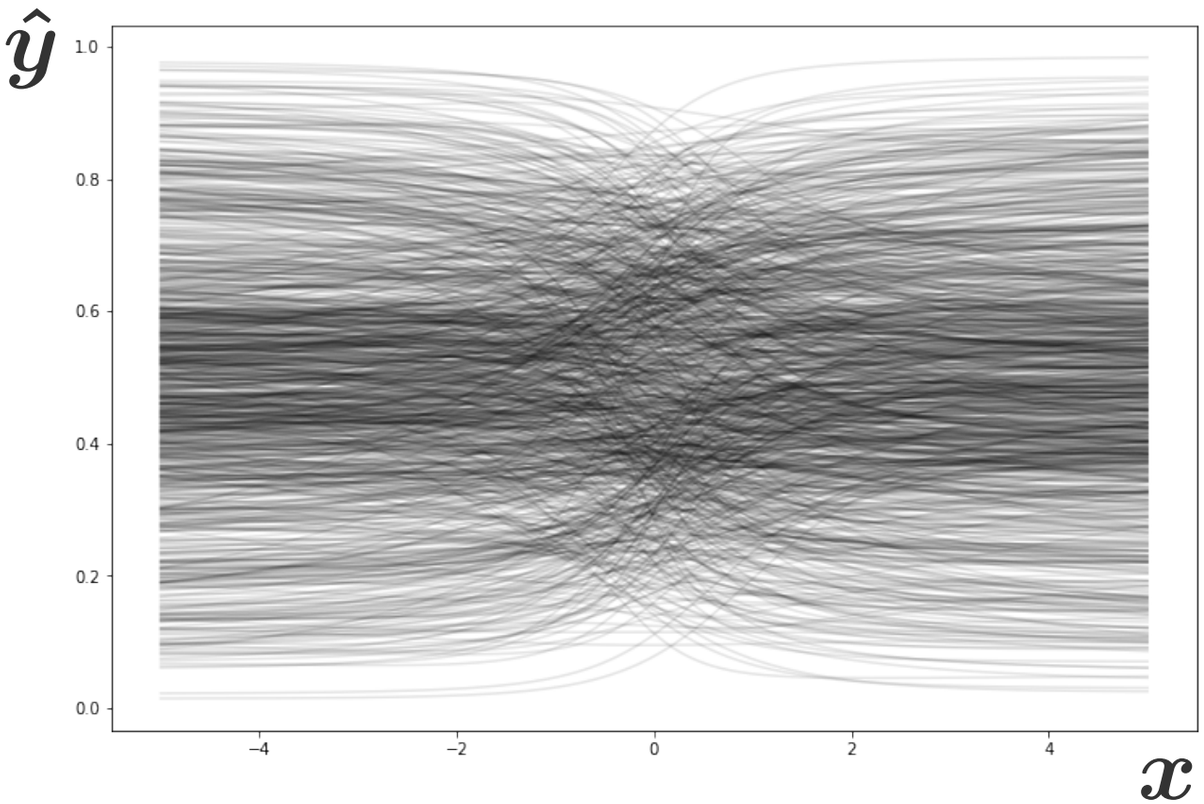
4. 万能近似定理（普遍近似定理）

万能近似定理（universal approximation theorem）[Cybenko, 1989]

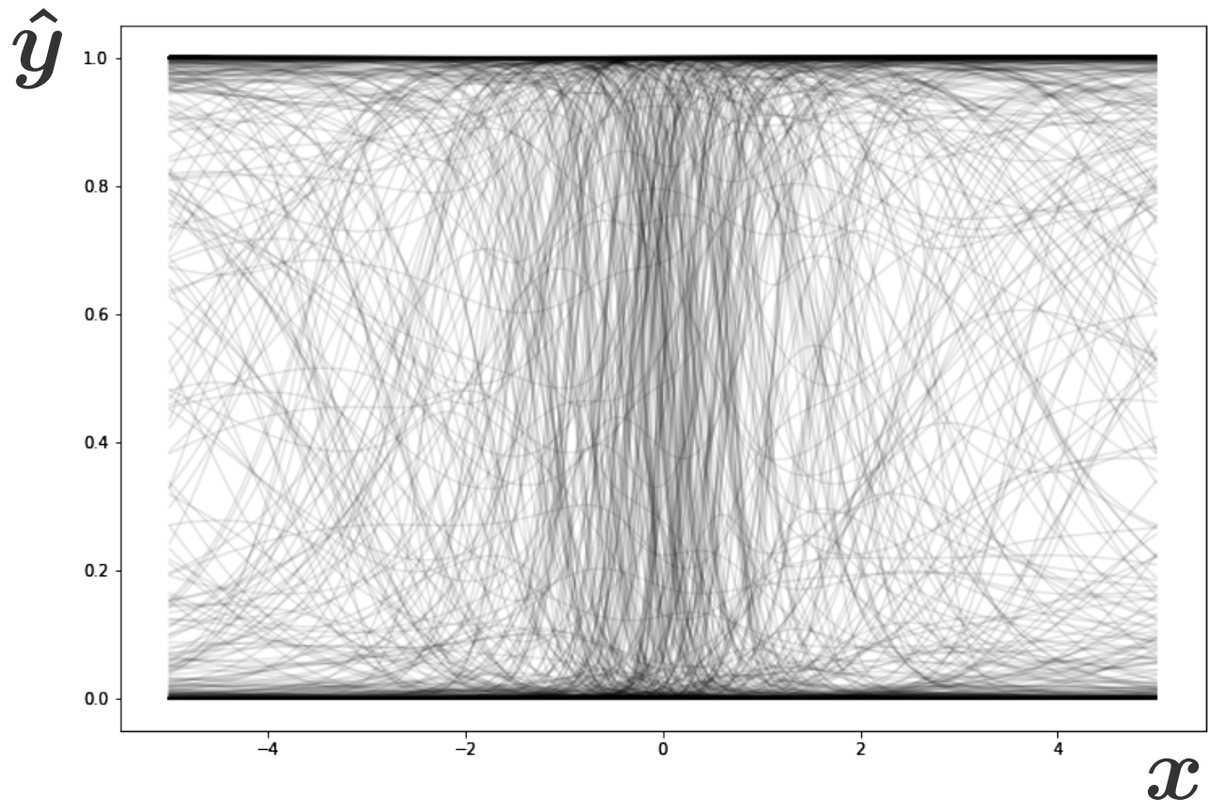
一層の中間層を持つニューラルネットは、幅を大きくしていくと、「どんな」関数でも近似して表現できる

4. 万能近似定理（普遍近似定理）

width = 2



width = 1024



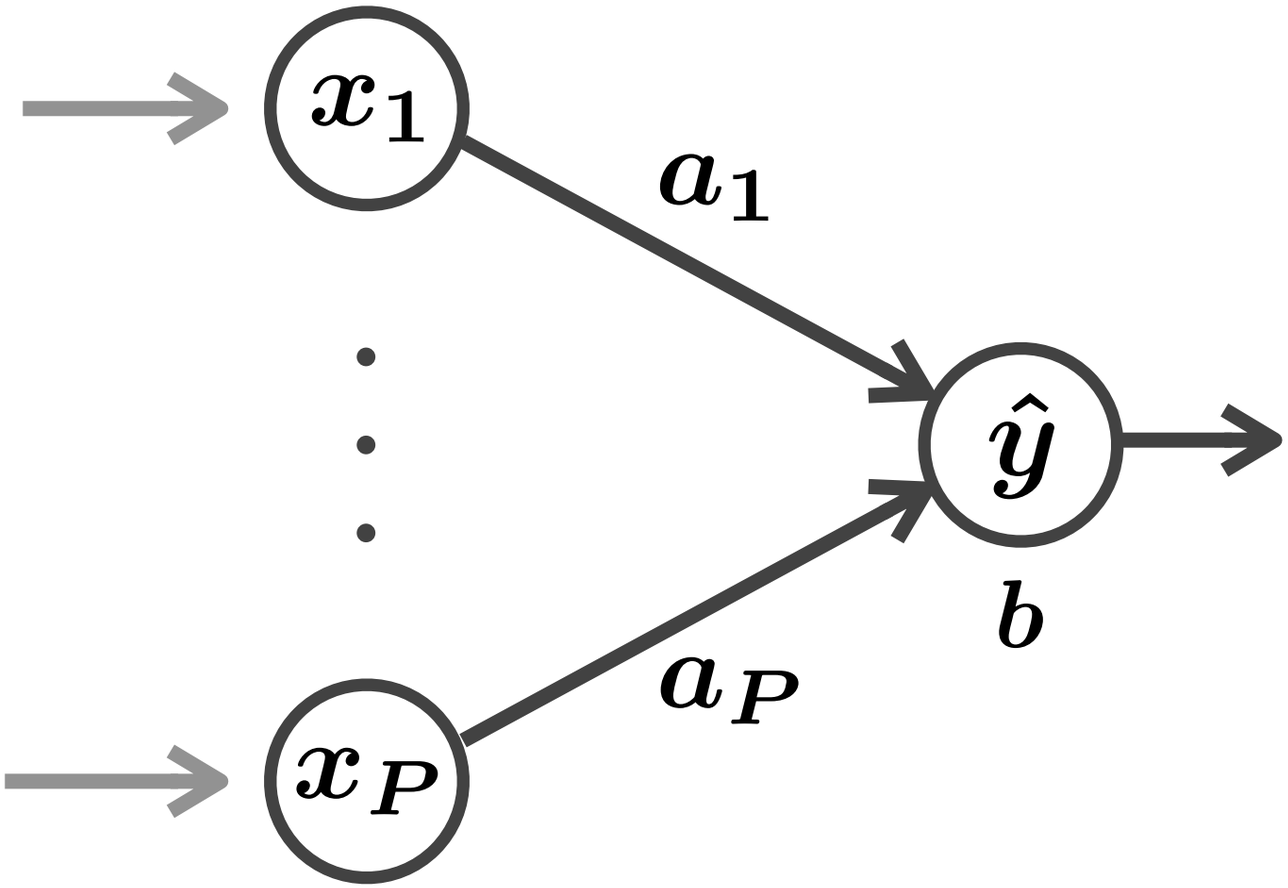
乱数で重みを決めた様々な場合のニューラルネットの定める関数のプロット。**中間層が重要** → 深さはなぜ？？

3. 深層ニューラルネット

1. なぜ多層化したい？：表現学習

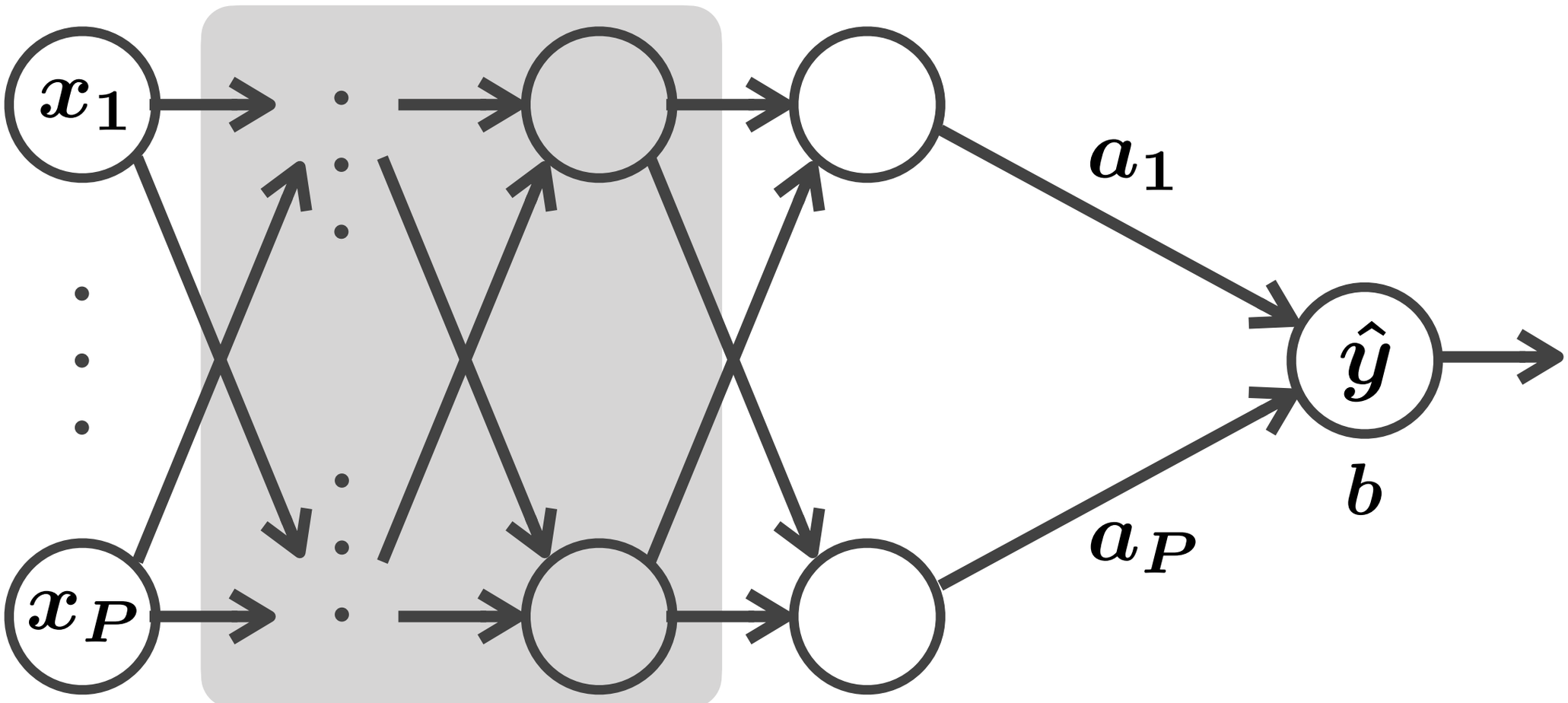
↓ この部分もデータドリブンに学習させたい（**表現学習**）

頑張って人手で特徴量エンジニアリングをした後のデータ

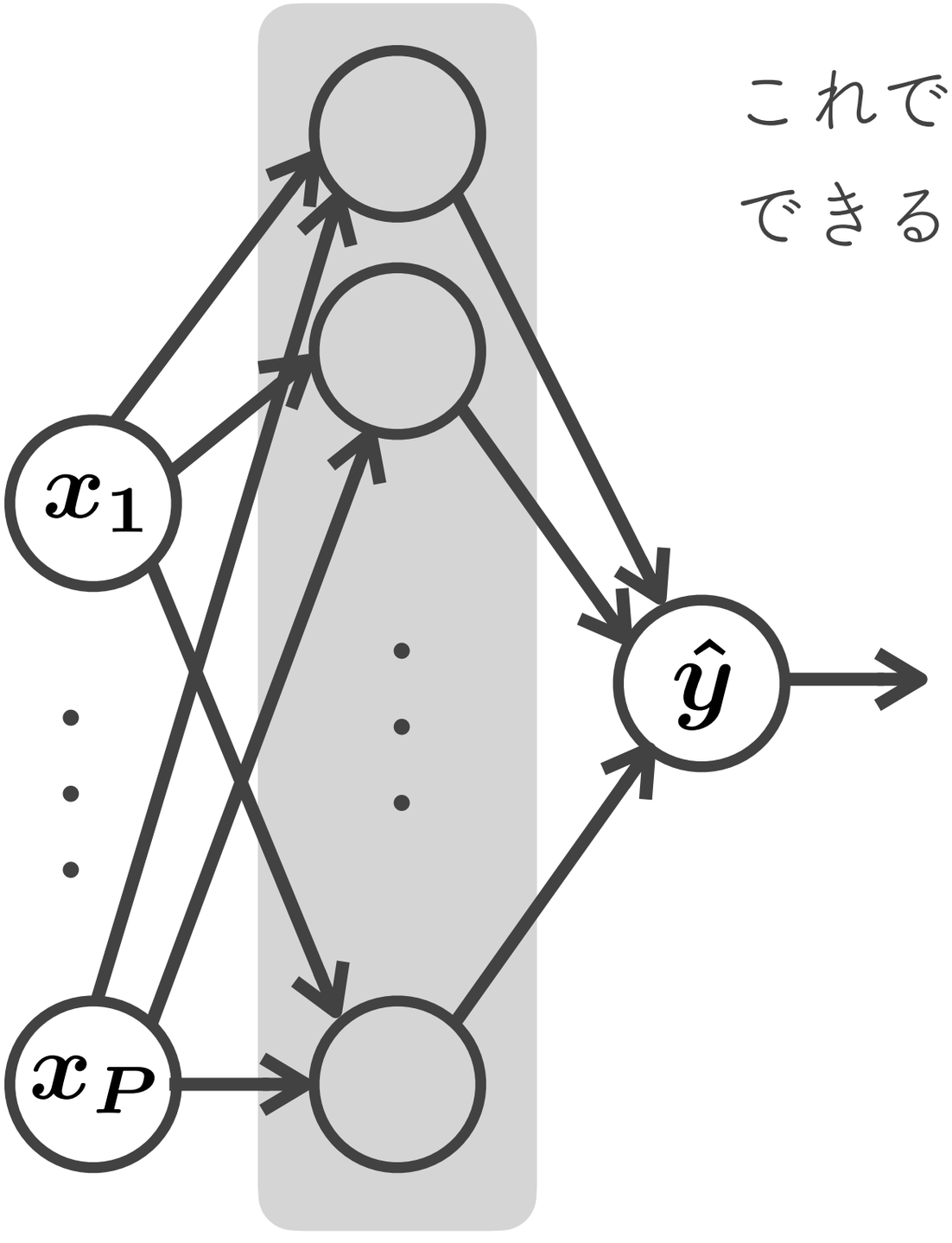


1. なぜ多層化したい？：表現学習

深い階層性により、複雑な入力データの情報をうまく処理した表現（特徴量）が得られると期待



1. なぜWideではなくDeep？



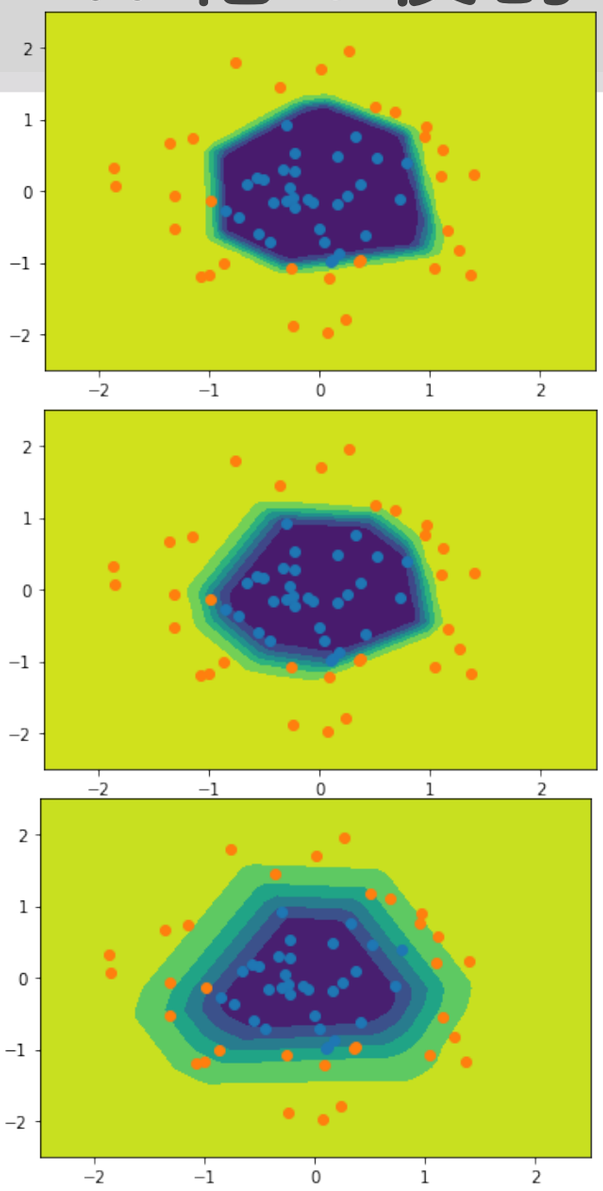
これでもいくらでも複雑なものが表現できるのでは？（万能近似定理）

↓

深い方がいい理由は完全にはわかっていない（c.f. 深さに関する万能近似定理）

（深さだけの問題でもない）

1. 幅の役割：ロジスティック回帰



4 hidden layers

60 weights

3 hidden layers

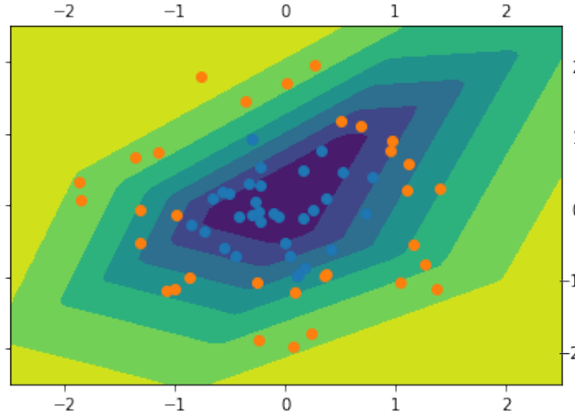
44 weights

学習された分類境界の様子。

深さを増やしたほうが、**複雑な表現を少ないパラメータで**学べる。

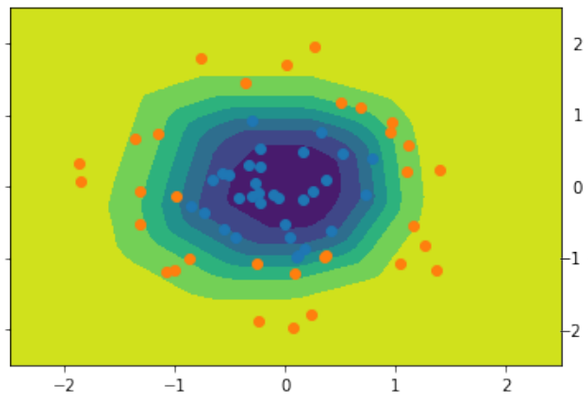
2 hidden layers

28 weights



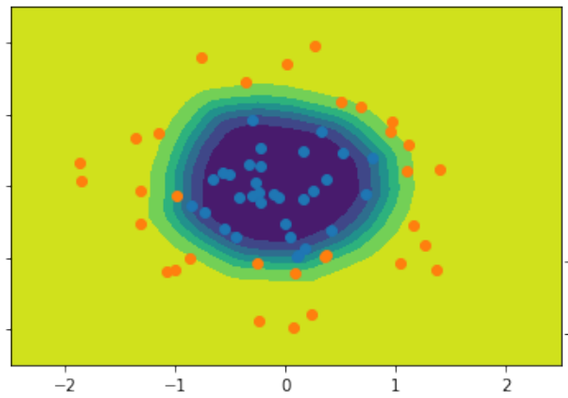
width=4

12 weights



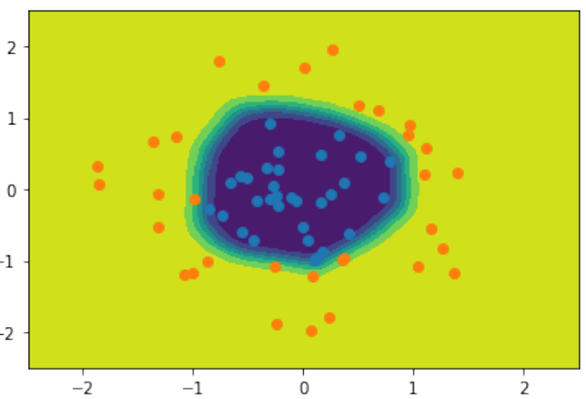
width=16

48 weights



width=64

192 weights



width=256

768 weights

2. 深層学習の難しさ？：loss landscape

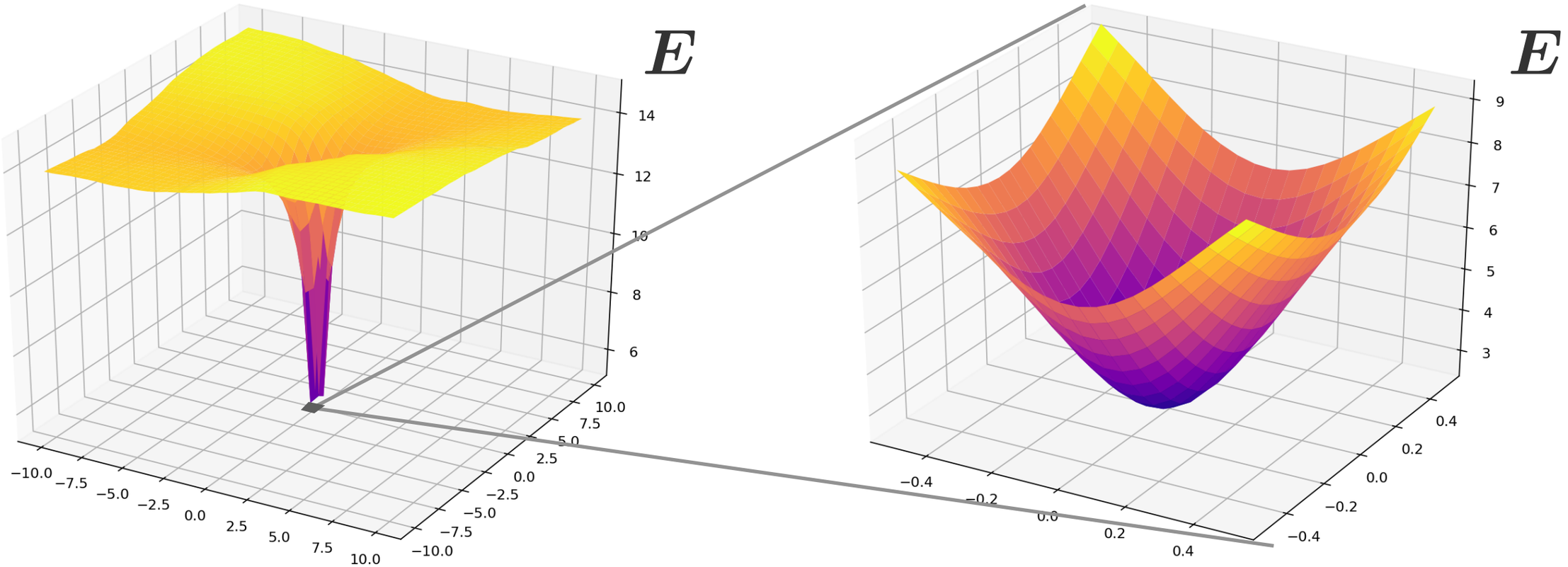
ニューラルネットという非線形なモデルの誤差関数も、普通の機械学習同様、勾配降下法で学習させます。

2. 深層学習の難しさ？：loss landscape

ニューラルネットという非線形なモデルの誤差関数も、普通の機械学習同様、勾配降下法で学習させます。

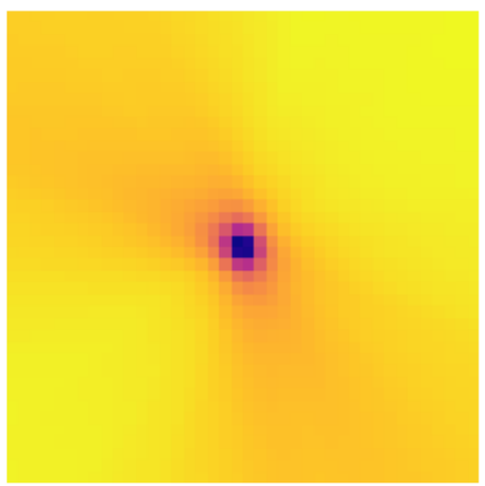
計算機物理などでは「複雑な最適化問題を勾配法で解きます」と言うと、普通は先輩から怒られること間違いなし

2. 深層学習の難しさ？：loss landscape



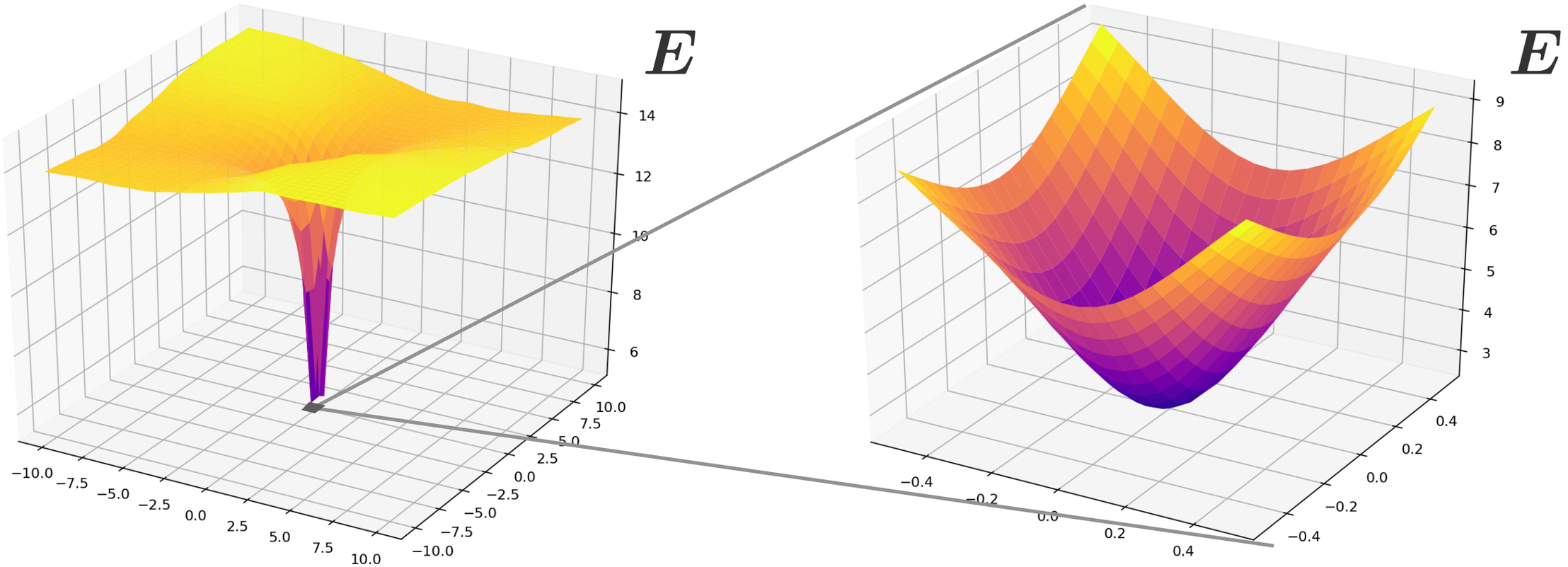
広域にわたるloss landscape

初期値周りのloss landscape



ソフトマックス回帰モデルにおけるクロスエントロピー誤差関数のlandscale（MNISTデータ）

2. 深層学習の難しさ？：loss landscape

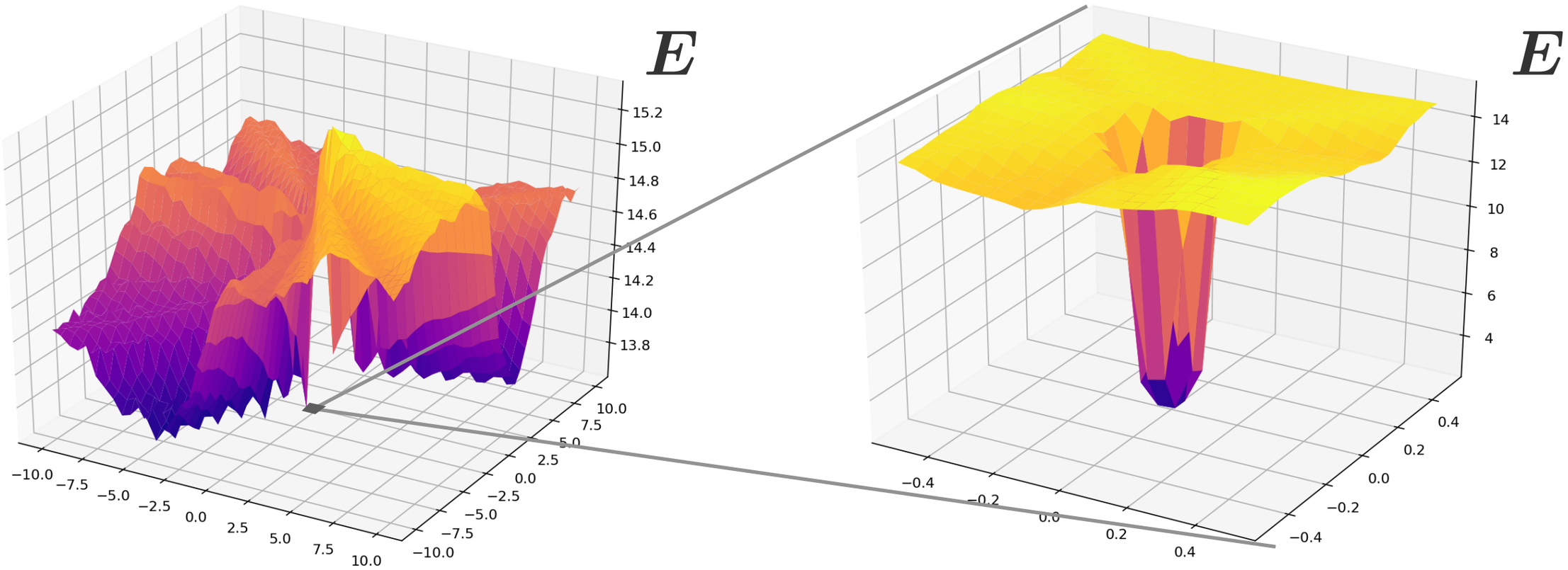


広域にわたるloss landscape　　初期値周りのloss landscape

ソフトマックス回帰

→ 比較的単純な形で、最小値周りは**綺麗な「お椀型」**

2. 深層学習の難しさ？：loss landscape



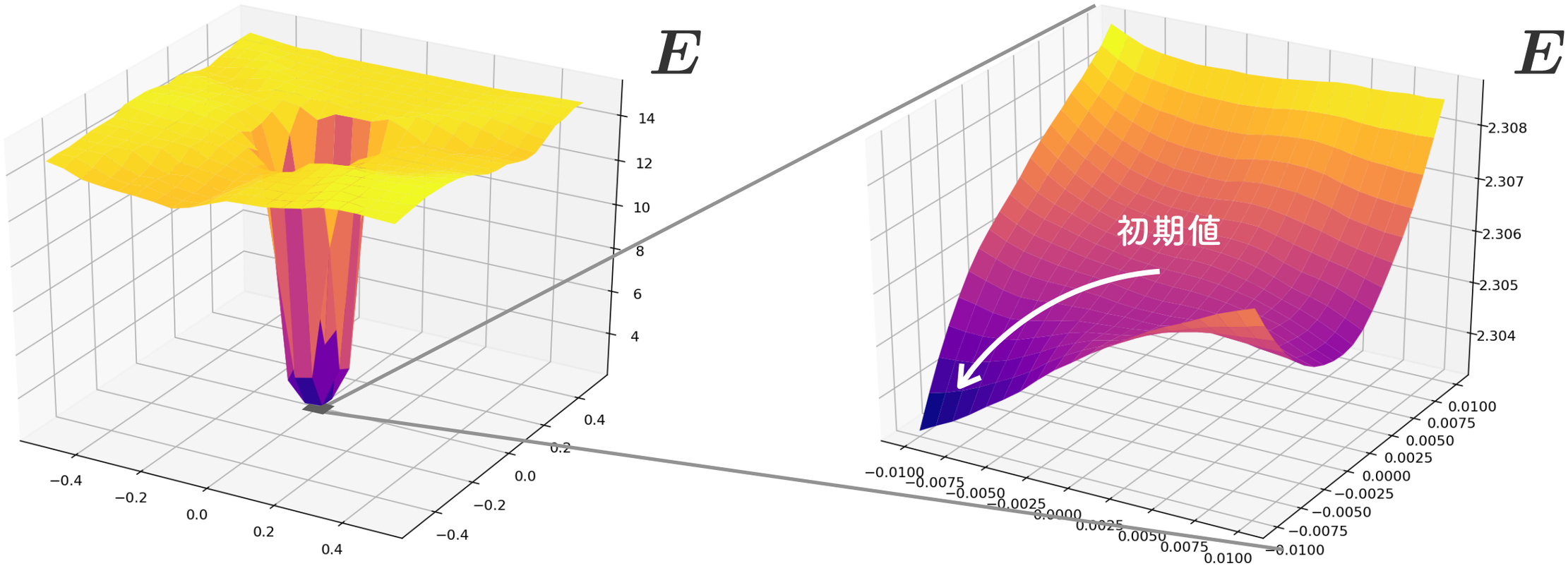
広域にわたるloss landscape　　初期値周りのloss landscape

多層ニューラルネット*におけるクロスエントロピー誤差関数(loss)のlandscale（MNISTデータ）

→ **複雑な形状**

* 6層MLP 784→265→265→265→265→265→10

2. 深層学習の難しさ？：loss landscape



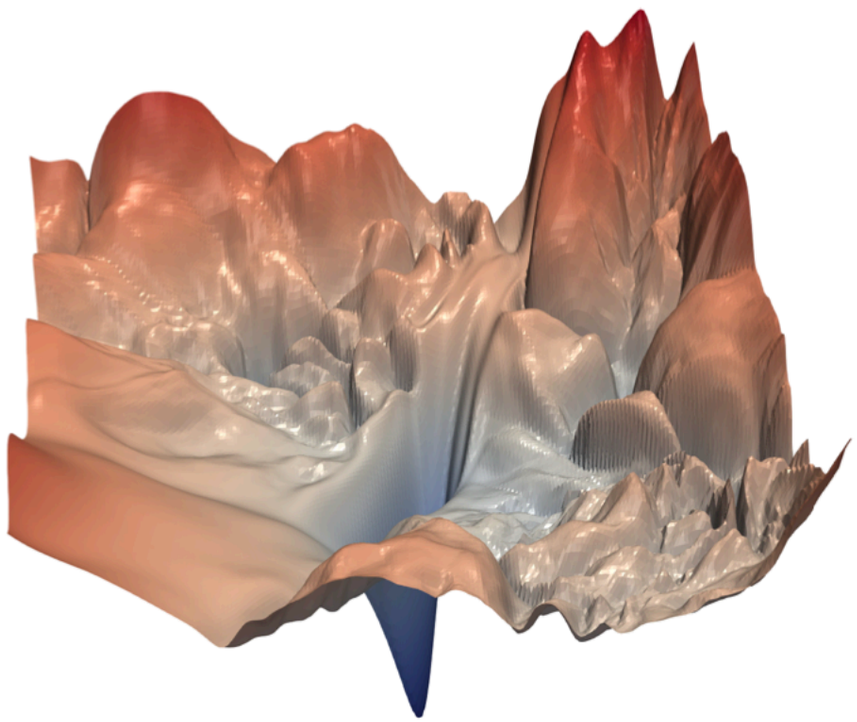
プラトー（平坦部）、**鞍点**（saddle point）、**局所最小値**（local minima）など、勾配降下法ではまずい領域が増える。

→ **勾配降下法で切り抜けられるか！？**

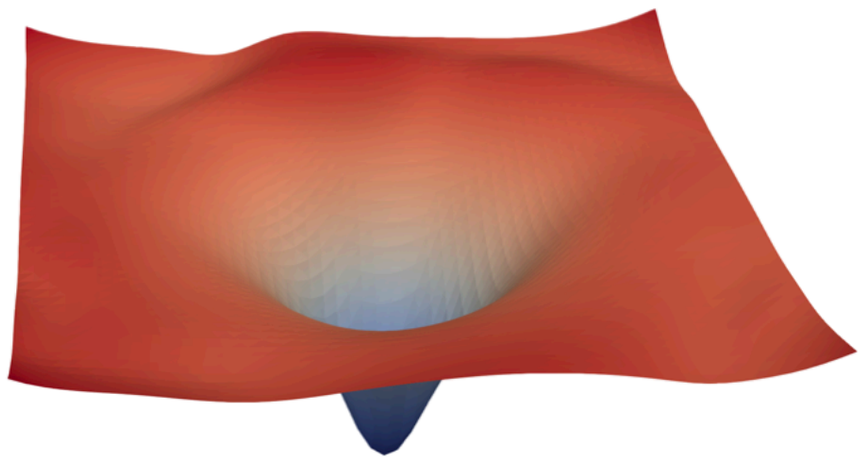
2. 深層学習の難しさ？：loss landscape

良いアーキテクチャは、滑らかなランドスケープを導く

→ これが**訓練可能性を保証**していると予想される



(a) without skip connections



(b) with skip connections

右のネットワークの residual構造を破壊

訓練がうまくいき、よく汎化するResNet56（56層）

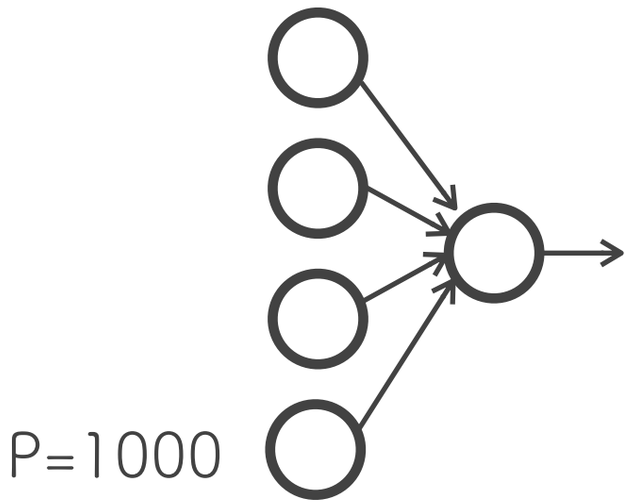
[H.Li, Z.Xu, G.Taylor, C.Studer, T.Goldstein, NIPS 2018]

4. ディープラーニング のための正則化

1. ディープラーニングの巨大さ

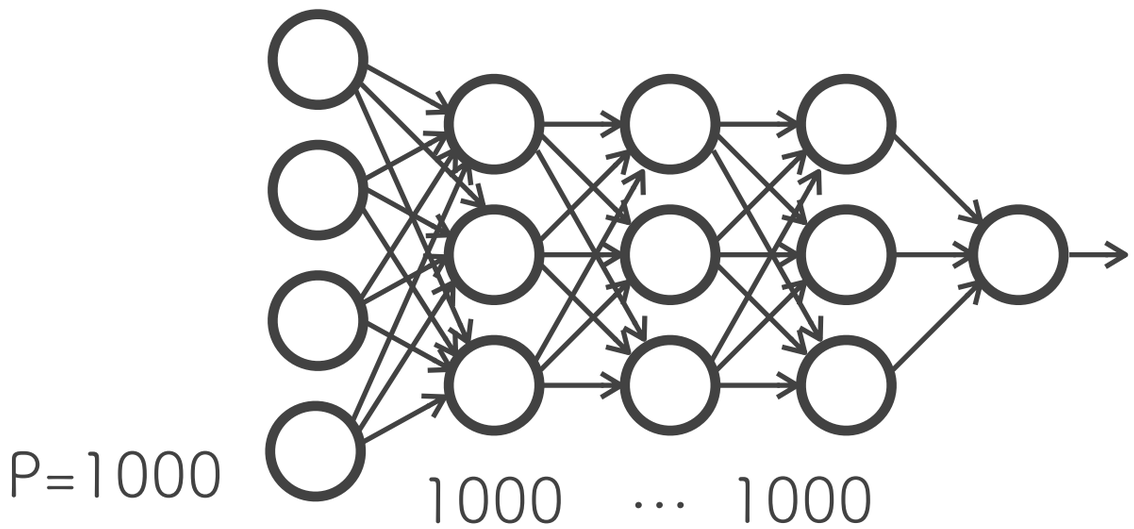
深層ニューラルネットは一般に**膨大なパラメータ**を持つ。

ロジスティック回帰



パラメータ数 ≒ 1000 = 10**3

ニューラルネット(L=11)



パラメータ数

≒ 10×1000×1000+1000

≒ 10**7

→ **過学習の恐れ**

2. ドロップアウト DropOut

モデルのパラメータが多いので過学習しなねない

→ ネットワークから不要なユニットを削除したい

→ 単純に削除するとモデルが小さくなり、逆に性能低下

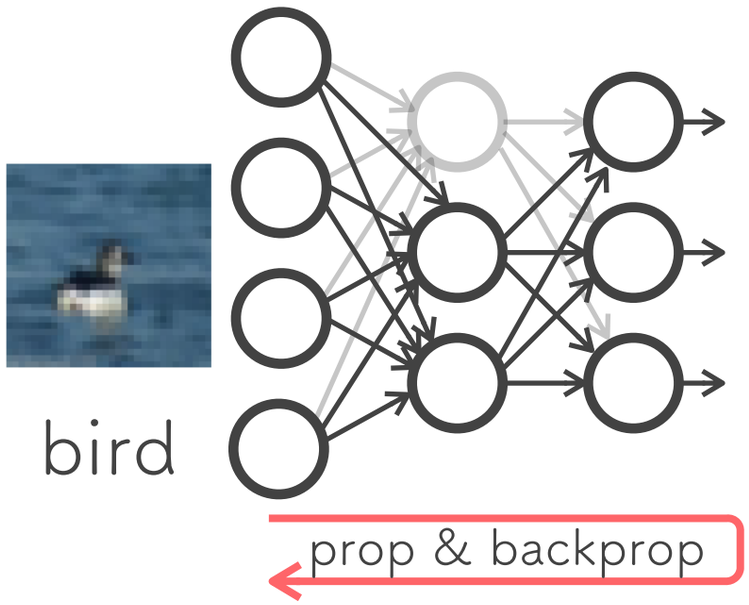
→ ドロップアウトは、**学習中だけ仮想的にネットワークを縮小**させるアイデア

ディープラーニングの初期を支えた手法 [Hinton et al., '12]

2. ドロップアウト DropOut

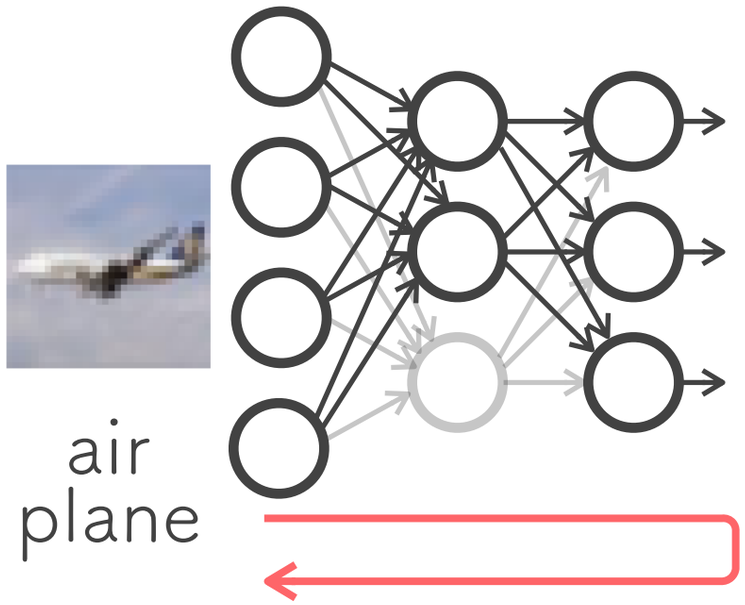
p：ハイパーパラメータ（ドロップアウト率）。何割のユニットを削除するのか、という割合（0.25）。

勾配法1回目



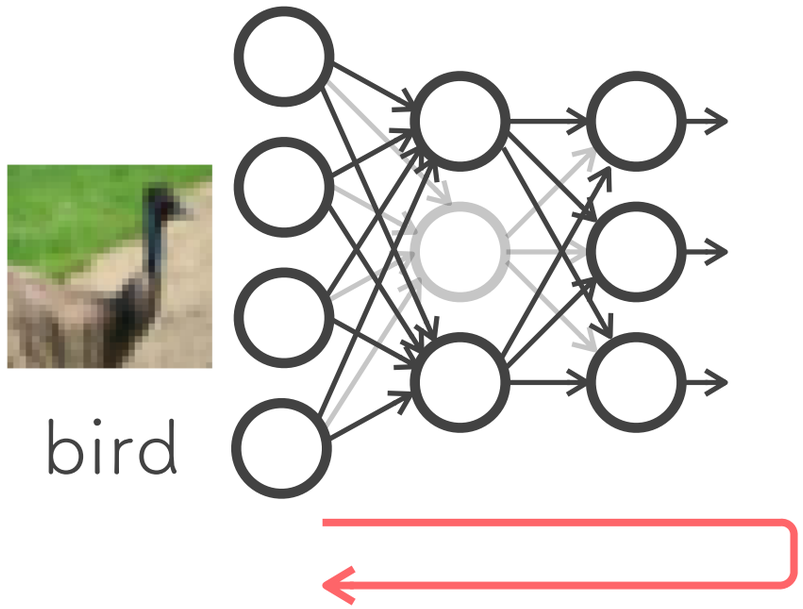
ランダムに p だけユニットを間引く

勾配法2回目



ランダムに p だけユニットを間引く

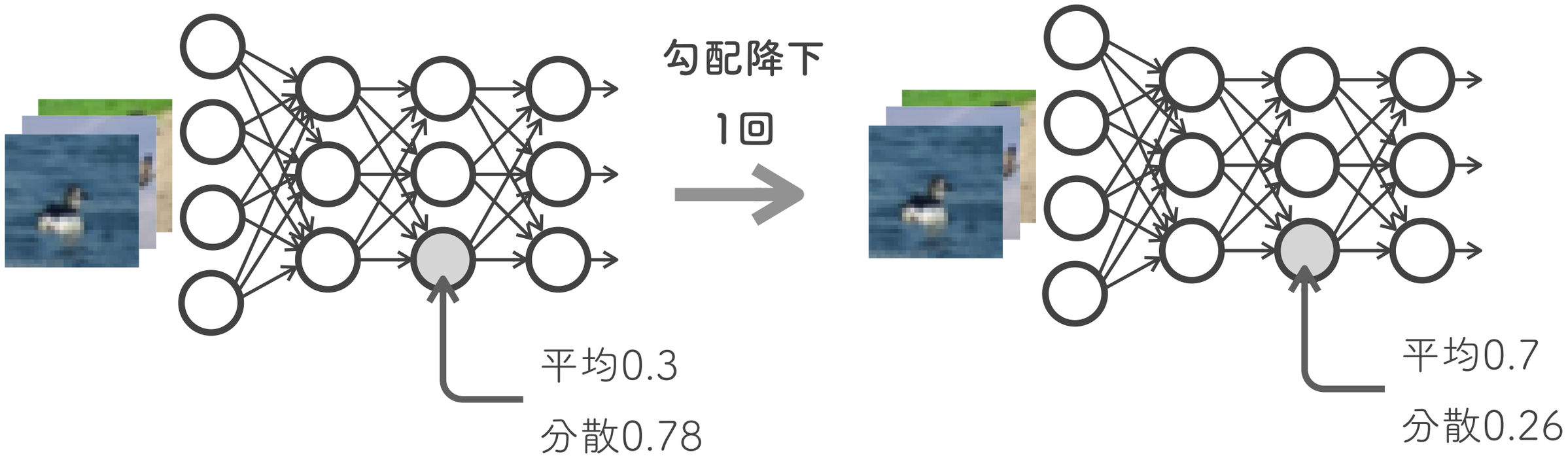
勾配法3回目



ランダムに p だけユニットを間引く

3. バッチ規格化 batch normalization

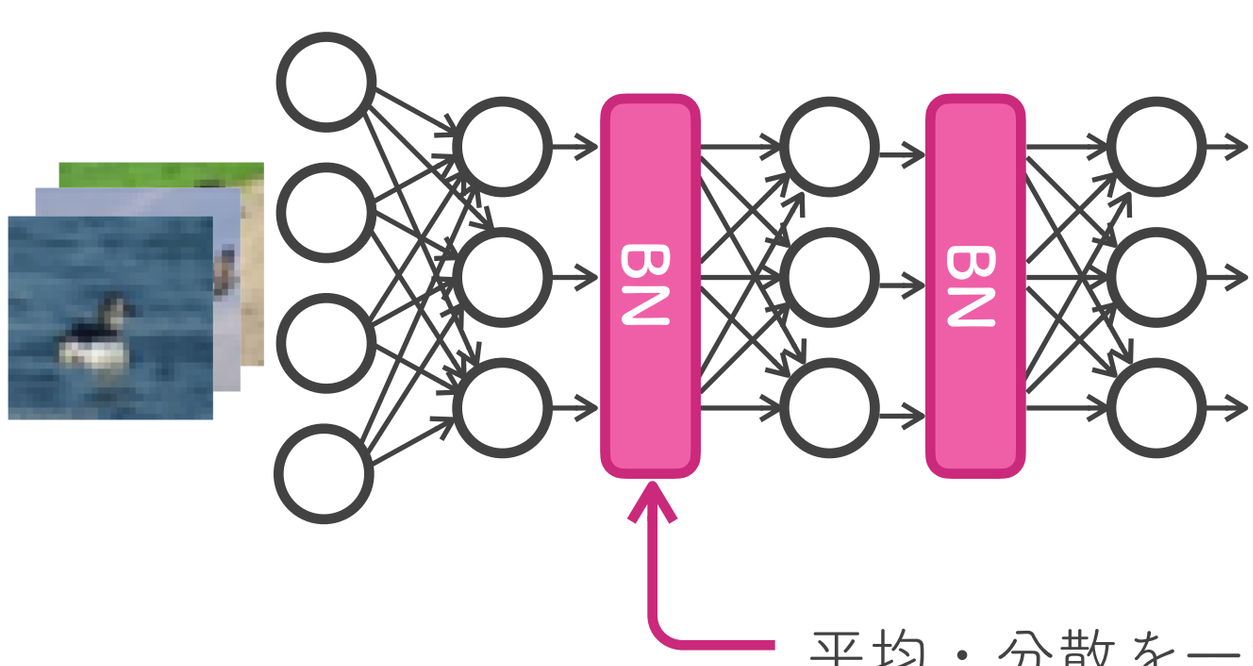
各層の出力変数の統計的な性質（平均、分散、…）は勾配法を行うたびに変わってしまう → 出力側が右往左往



内部共変量シフト internal covariate shift

3. バッチ規格化 batch normalization

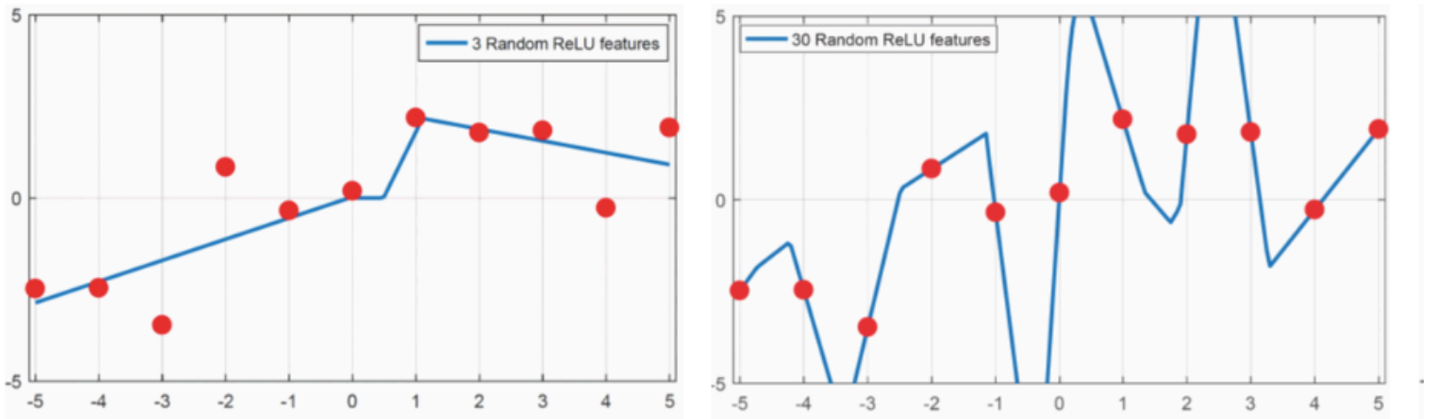
そこで各層の間に、**平均・分散を一定化**させるように学習する層(**バッチ規格化層**)を入れる



＊実は、内部共変量シフトの防止はバッチ規格化の主要な役割ではなく、今では他の仕組みで汎化を導いていることがわかっている。

4. 深層学習自体の不思議な性質：良性過学習

モデルの容量を上げることで、仮説空間に含まれる典型的な関数の滑らかさ等がより改善→より汎化



(a)
3パラメータ
（過少適合）

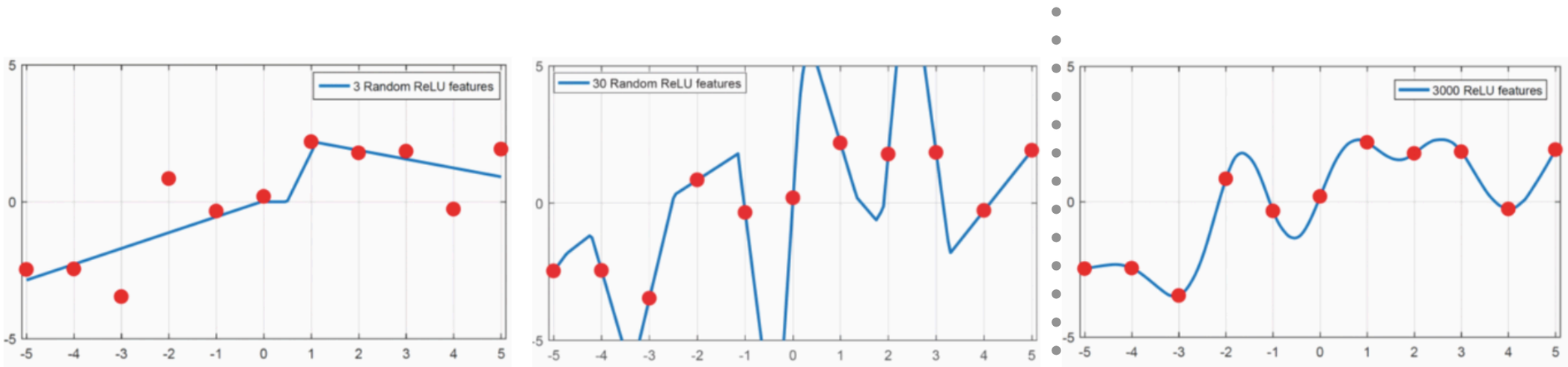
(b)
30パラメータ
（過剰適合）

汎化！？

[K.Belkin, Acta Numerica (2021)]

4. 深層学習自体の不思議な性質：良性過学習

モデルの容量を上げることで、仮説空間に含まれる典型的な関数の滑らかさ等がより改善→より汎化



(a)
3パラメータ
（過少適合）

(b)
30パラメータ
（過剰適合）

(c)
3000パラメータ
（汎化、現代的regime）

[K.Belkin, Acta Numerica (2021)]

5. アーキテクチャデザイン

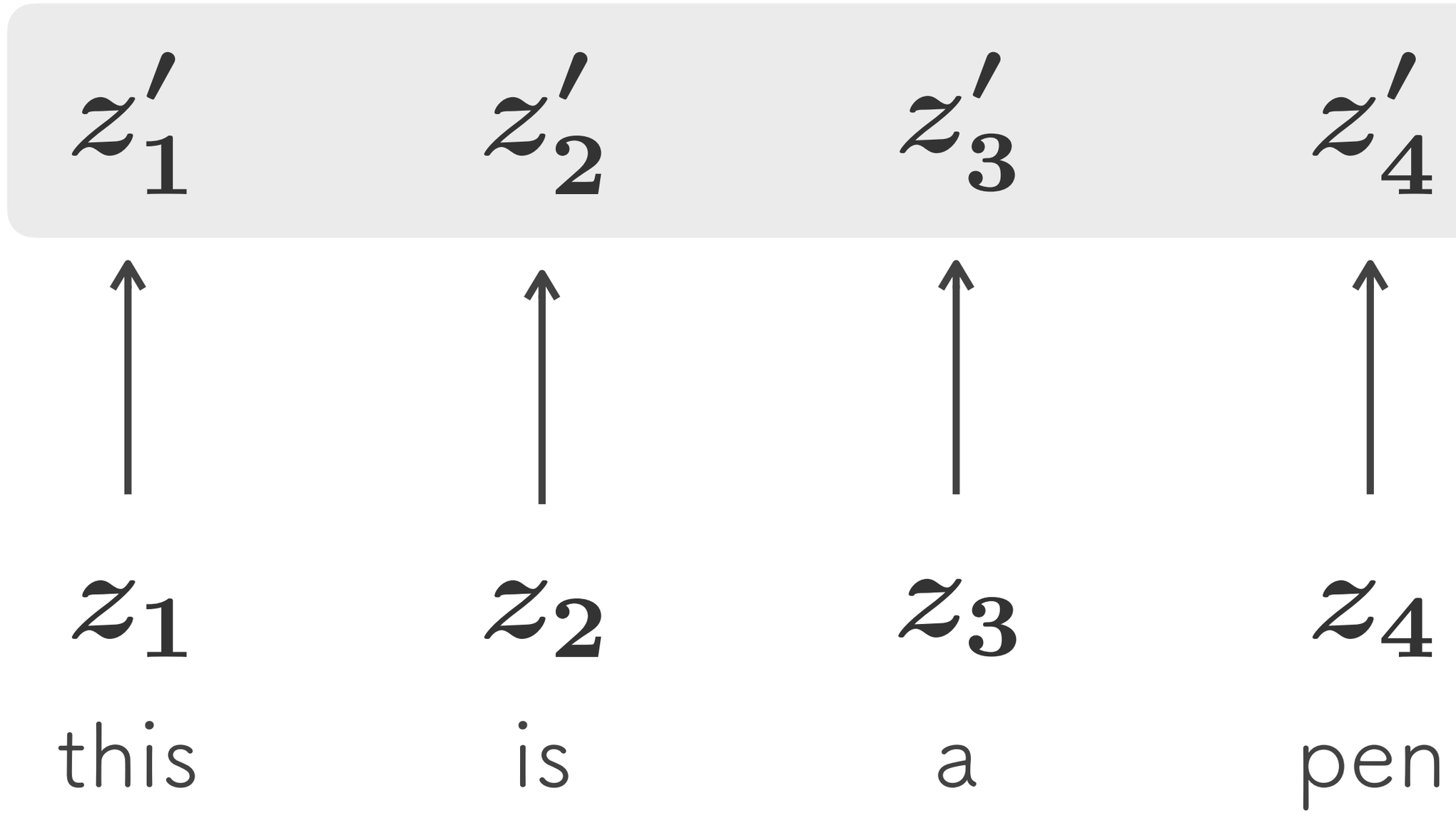
さらに、**ネットワーク構造のデザインが強力な正則化効果**（のようなもの）を生み出し、深層学習の汎化能力を支えてることが知られている。**[Chiyuan Zhang et al., ICLR 2017]**

→ Transformerの例

5. Transformer

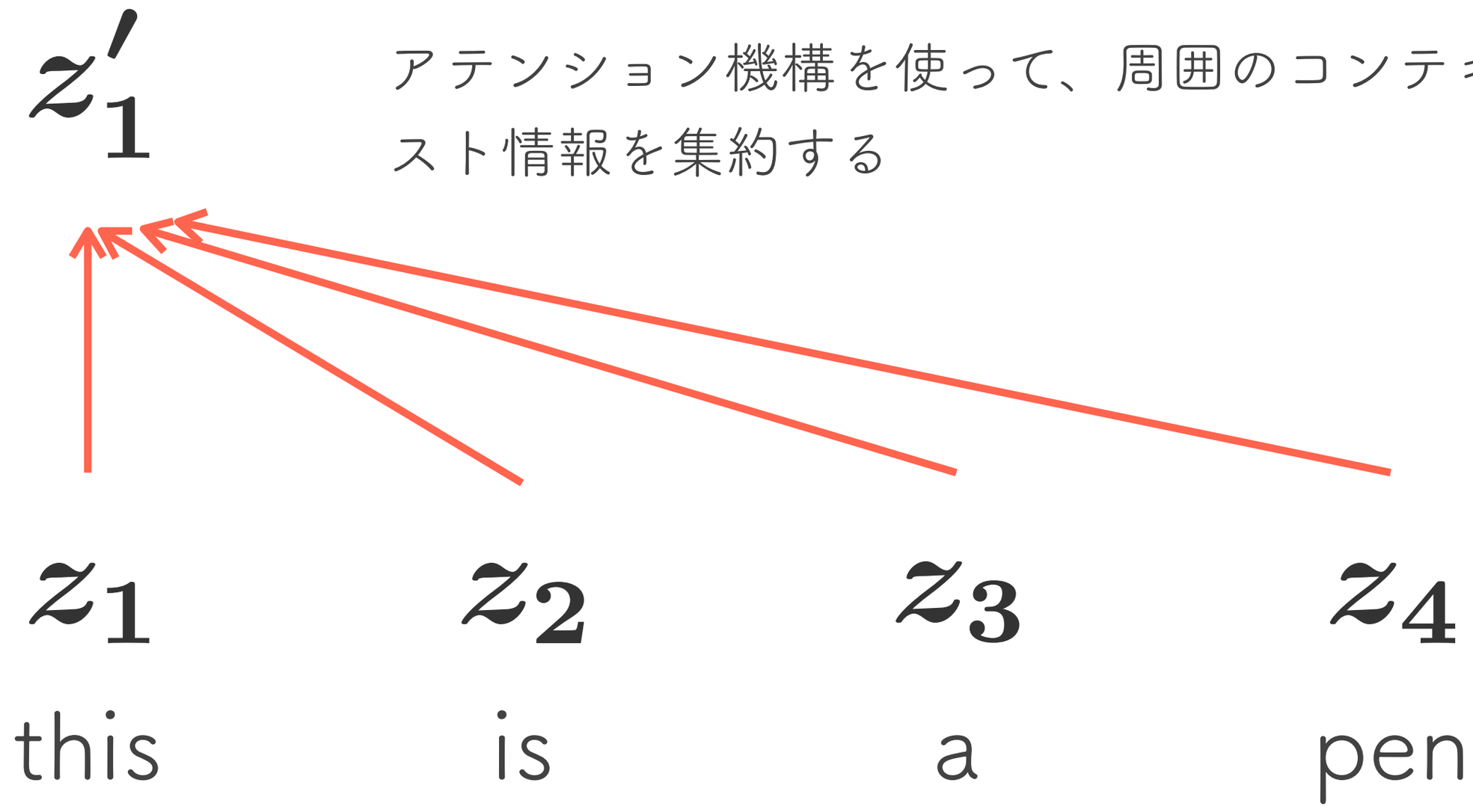
Transformerと自然言語処理

周辺のコンテキスト情報も取り込んだ単語の表現ベクトルを作りたい
（この文章におけるthisの意味ベクトル、etc）



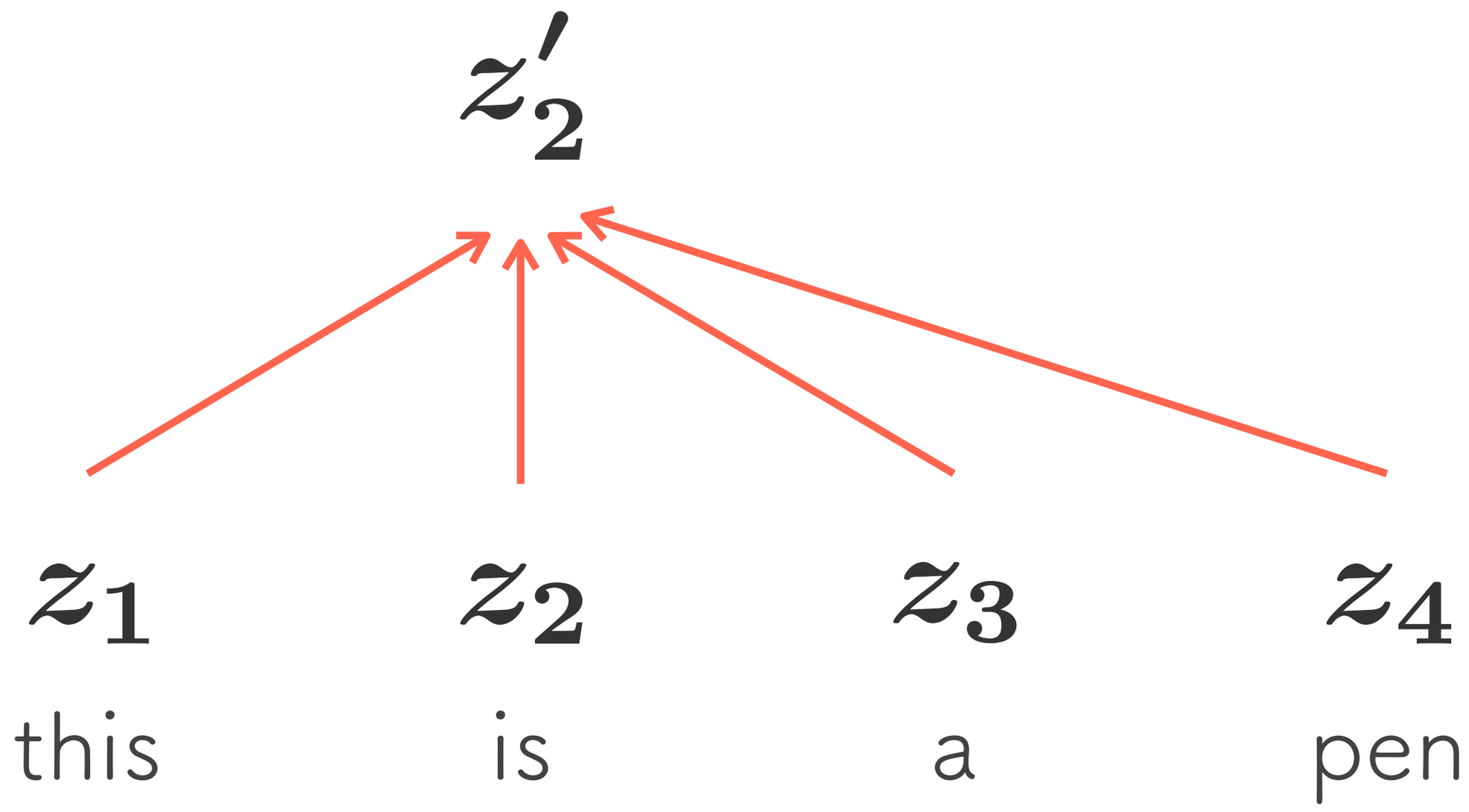
Transformerと自然言語処理

周辺のコンテキスト情報も取り込んだ単語の表現ベクトルを作りたい
（この文章におけるthisの意味ベクトル、etc）



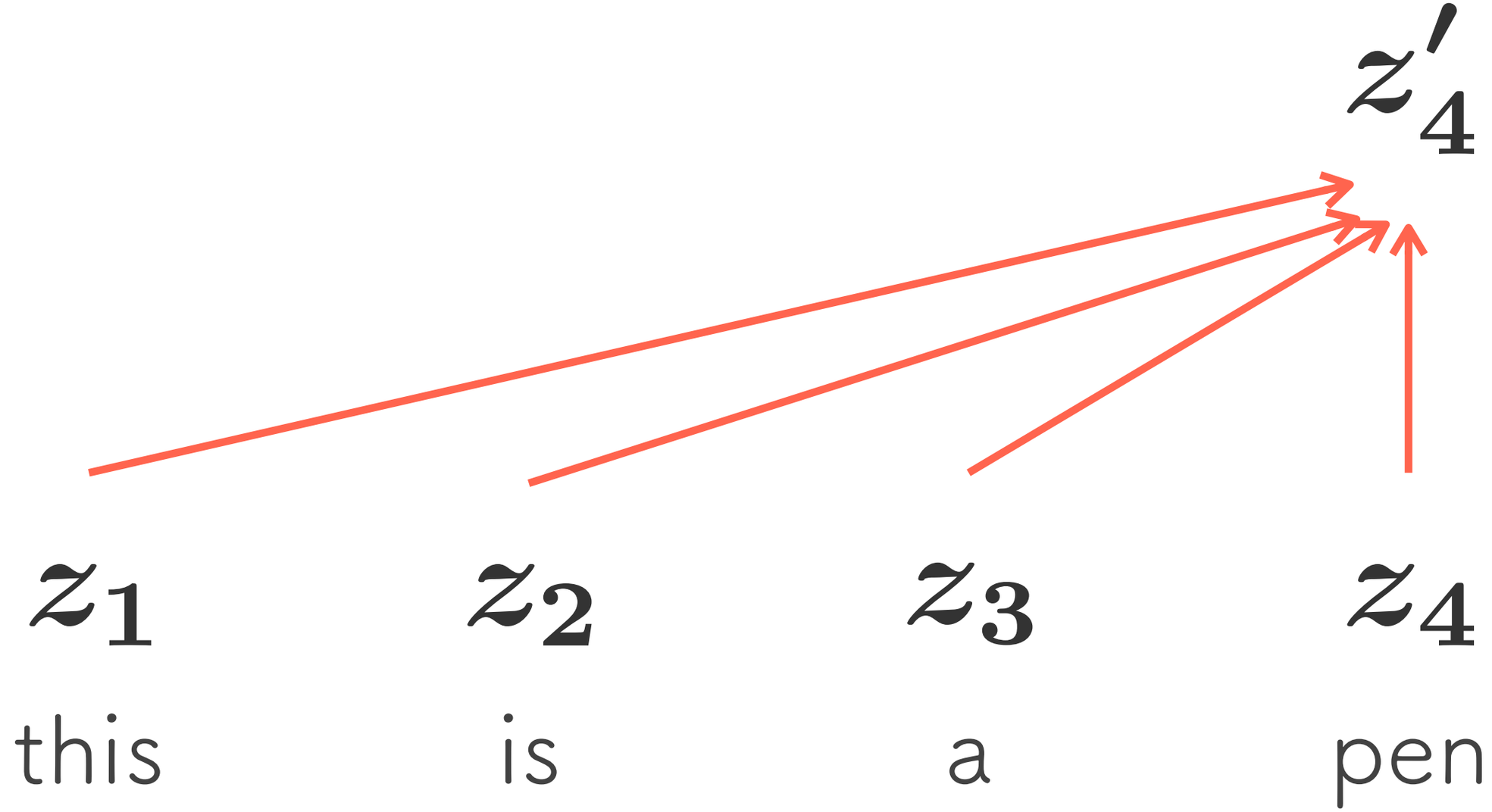
Transformerと自然言語処理

周辺のコンテキスト情報も取り込んだ単語の表現ベクトルを作りたい
（この文章におけるthisの意味ベクトル、etc）



Transformerと自然言語処理

周辺のコンテキスト情報も取り込んだ単語の表現ベクトルを作りたい
（この文章におけるthisの意味ベクトル、etc）



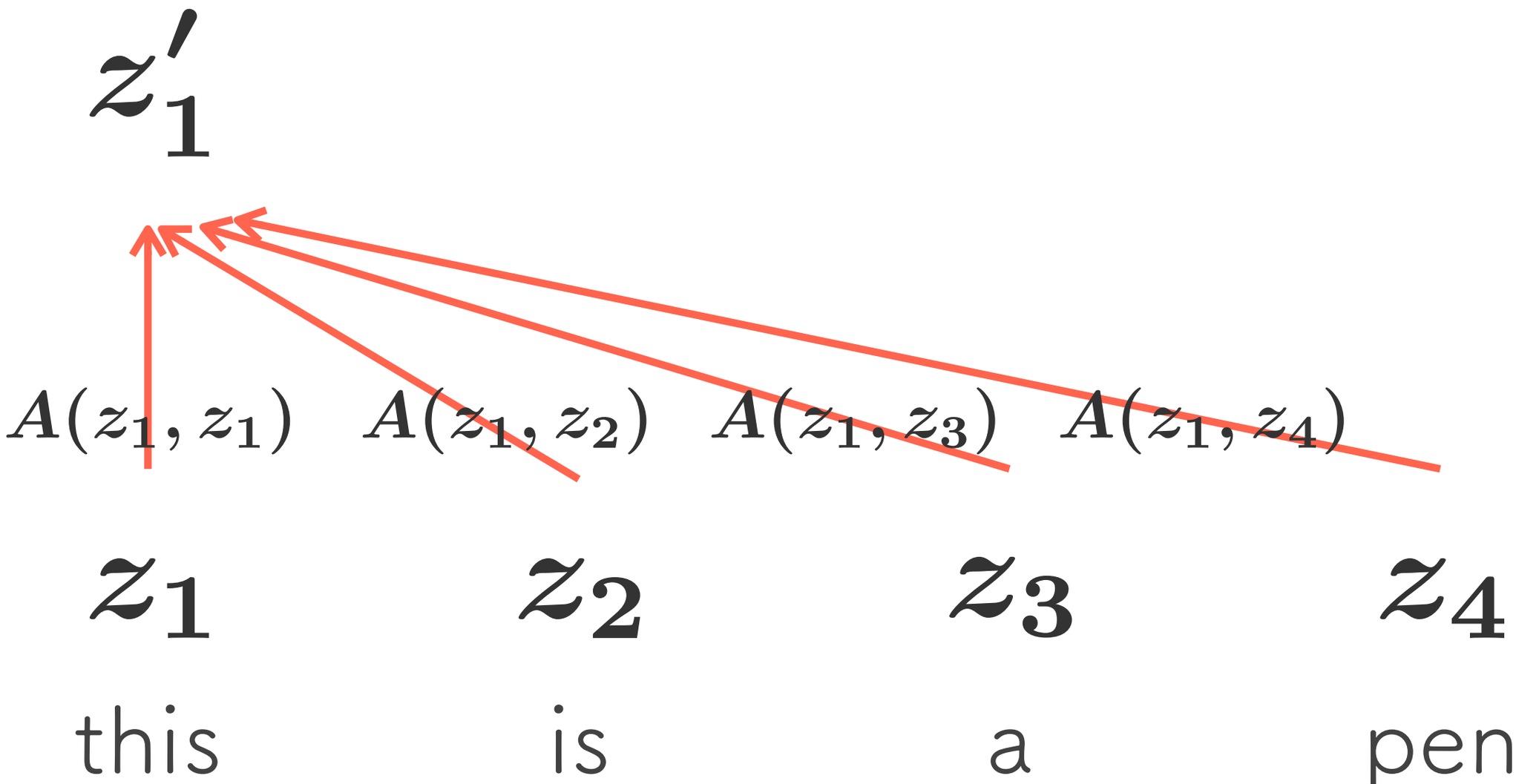
Transformerと自然言語処理

$$z_1' = \sum_i A(query = z_1, key = z_i)\, z_i$$

小さなニューラルネット。queryに対するkeyの重要性の重みを推定している（気持ち）。学習パラメータをもつのでここも訓練。

Transformerと自然言語処理

RNN等と違い、再帰的な計算はないのでどんなに長い系列に対しても計算が完全に**並列化可能**！！（訓練時、あるいはEncoder部分のみ）



言語モデルの事前学習戦略

ラベル付きデータは自然言語では収集コストが高い。
ただのラベル無しテキストならwebにたくさんある。

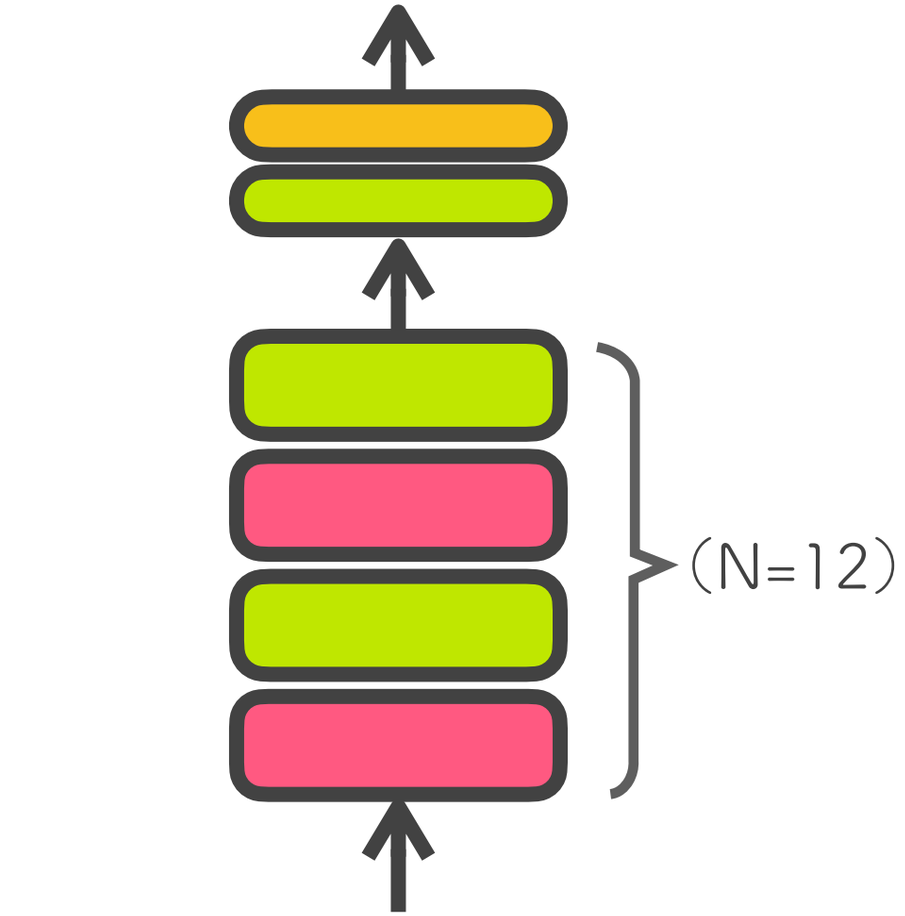
→ **教師なし事前学習**がいい戦略（転移学習・fine-tuning）

OpenAI GPT [Radford et al., ’18]

Generative Pretrained Transformers

ラベルのない文章でも**文生成**はできる → **generative pretraining**

これ は ペン です EOS



- 自然言語12タスクのうち9個のタスクでSOTAを達成
- OpenAIの研究チーム
- **1.7億パラメータ**
- BookCorpusデータ（7000冊の本、800M words）

GPT-2 [Ranford et al., '19]

マルチタスク学習により事前学習

$$P(output \mid input,\ task)$$

言語モデルなので、**ここも文章にできる：** `'translate to french'`

・8のzero-shotタスクのうち7個でSOTAを達成

・WebTextデータセット（800万文書から40GBのテキスト）

・**N=48：15億パラメータ**

→ zero-shot、few-shotに強い

→ 文生成に強い（GPTの事前学習の仕組み上）

→ 翻訳や文理解などのタスクにはあまり強くない
（一方向モデルの限界）

GPT-3 [Ranford et al., '20]

・さまざまなlow-shotタスクで極めて高い性能を達成
・仏英、独英翻訳では、zero-shotで教師ありのSOTAを凌駕
・SAT類推問題では、few-shotでsuper-human（平均受験生以上）
・Common Crawlデータセット（1T words）も使う
・**N=96：1750億パラメータ**
・アテンションをスパース化する工夫

→ BERT系のfine-tuning戦略ではなく、**zero/one/few-shot戦略**により、fine-tuningでの過学習やデータ準備のコストを避ける

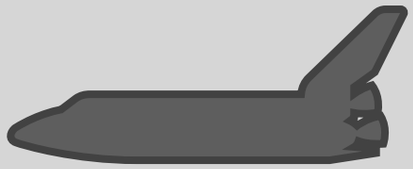
→ 依然としてqueation answeringや文理解などでは双方向＋fine-tuningアプローチには負ける（一方向モデルの限界）

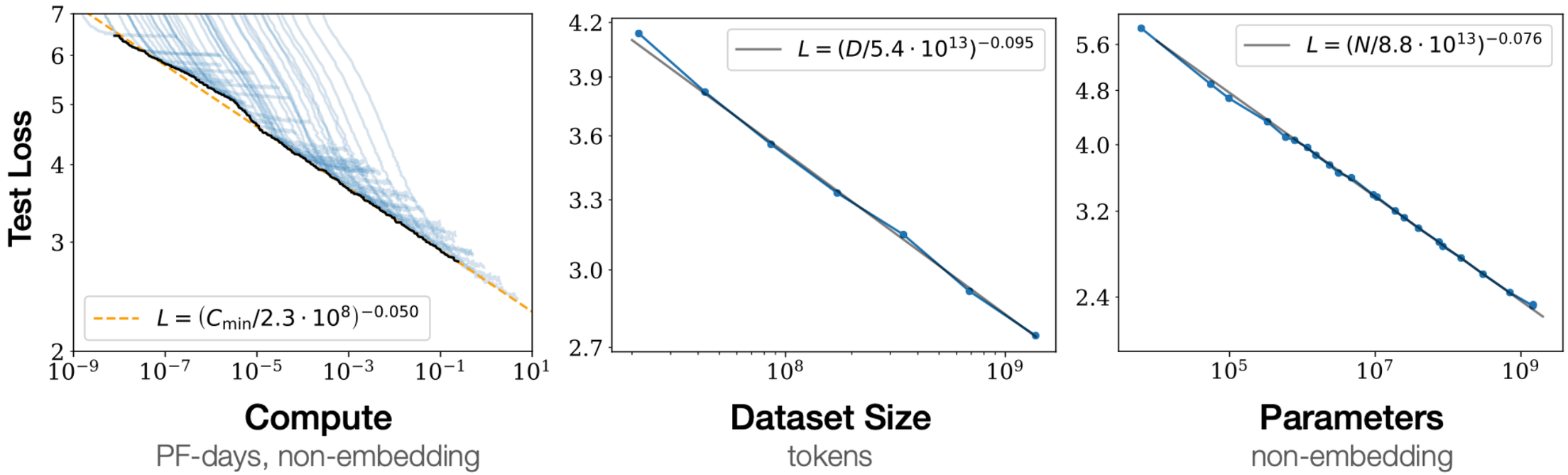
プロンプト工学

プロンプトへ「答えやすい優しい問いかけ」をする工夫。良い「呪文」を探す試行錯誤。

https://arxiv.org/pdf/2107.13586.pdf

GPTのスケーリング則

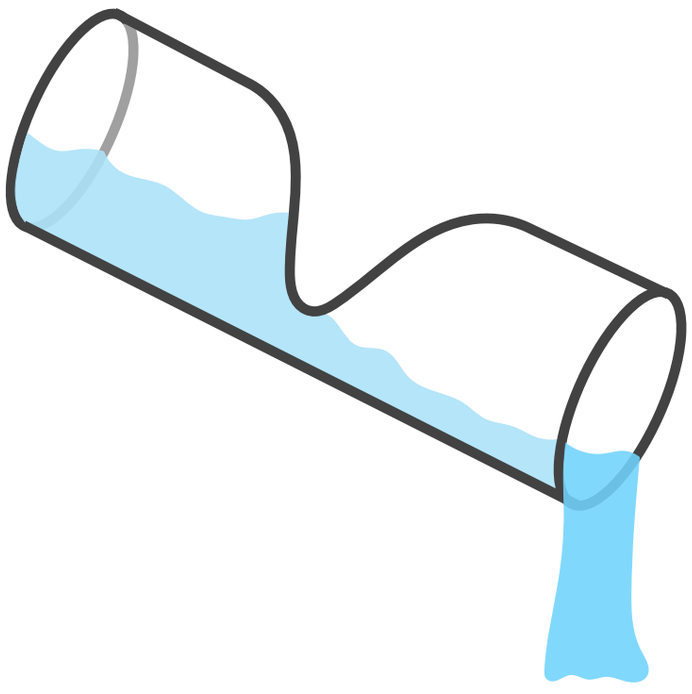




$\mathbf{1\,PF\text{-}day = 10^{15} \times 24 \times 60 \times 60 = 8.64 \times 10^{19}}$不動小数点演算

WebText2 datasetで訓練

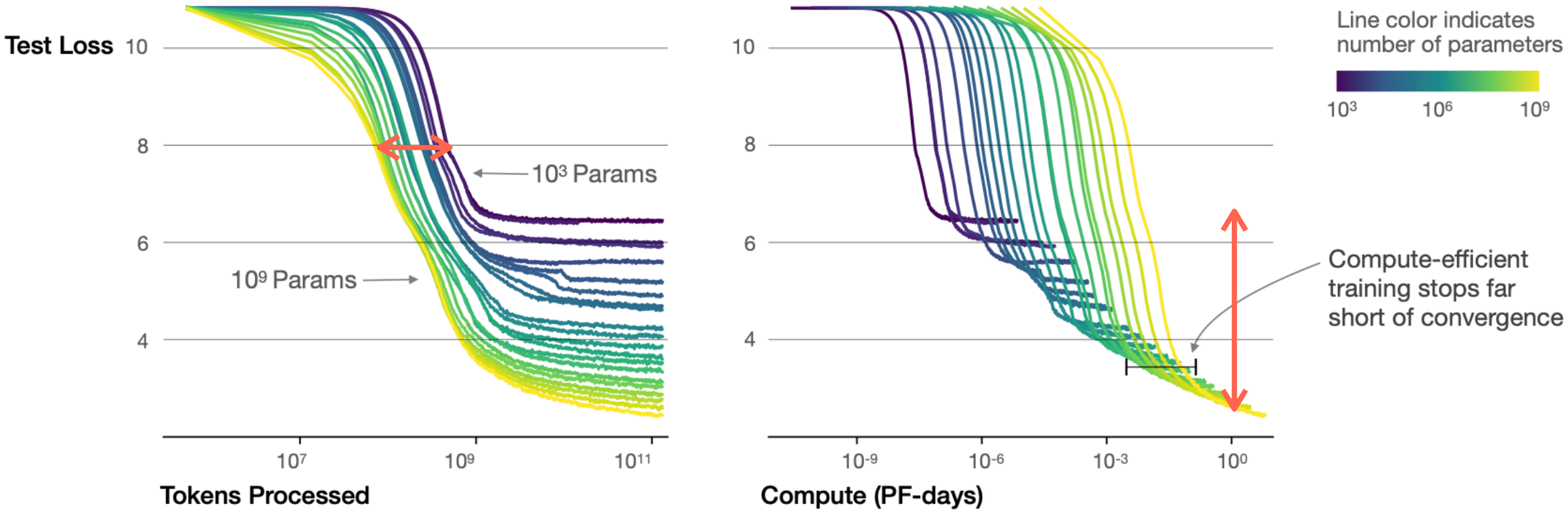
[OpenAI, Scaling Laws for Neural Language Models, 2020]



（深さや幅のような詳細にはあまり依存せず）

３スケールさえ増やせばボトルネックなくスケールしてゆく！まだ飽和は見られない。

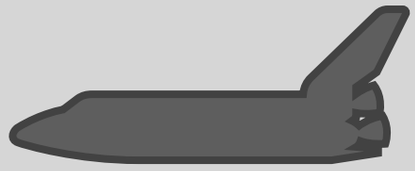
GPTのスケーリング則



[OpenAI, Scaling Laws for Neural Language Models, 2020]

データ効率：大きいモデルの方が少ないサンプル(と少ないステップ)で同じ性能に到達する

GPT-4？（一昨年のスライド）



- ~~100兆パラメータ？~~

~~“From talking to OpenAI, GPT-4 will be about 100 trillion parameters,” Feldman says.~~

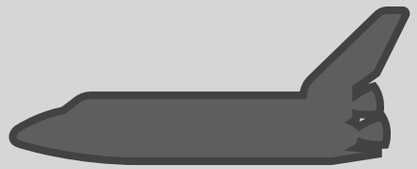
- ~~訓練用の専用チップ？~~
- 数年後？

https://www.wired.com/story/cerebras-chip-cluster-neural-networks-ai/

今週ついに公開されたが…

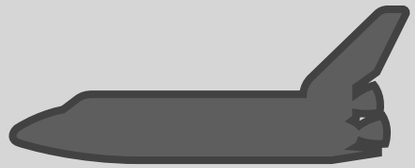
this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar.

ChatGPT



- GPT-3系のモデル（実際にはGPT-3.5シリーズ）
- GPT-3.5シリーズのベースはcode-davinci-002（Codex）
- 人間の書いた出力で教師あり学習をかけたバーションがdavinci-instruct-betaやtext-davinci-002
- それに人間のフィードバックを強化学習でかけたものがtext-davinci-003（InstructGPT）。ChatGPTも同じ系統の対話データ版のものらしい。
- ChatGPT以外は既にAPIが公開されていたので、エンジニアリングの努力による対話に対する洗練以外は、本質的な驚きはほとんどないのでは。
- 生成モデルブームに乗ってブラウザ版を公開し、驚くべき勢いでユーザーを獲得。ビジネスインパクト大。Bingへ？

InstructGPTとRLHF



人間のフィードバックによる強化学習（RLHF）も用いる。

① プロンプトデータに、40人程の作業者が望ましい出力を実演。その出力で、GPT-3を教師ありfine-tuning（SFT）。

→ `davinci-instruct-beta`

② モデルの入出力データをサンプリングし、最悪から最高までのラベルをアノテータがつける。このデータで報酬モデル（RM、①の最終層を外したモデルベース）を訓練。

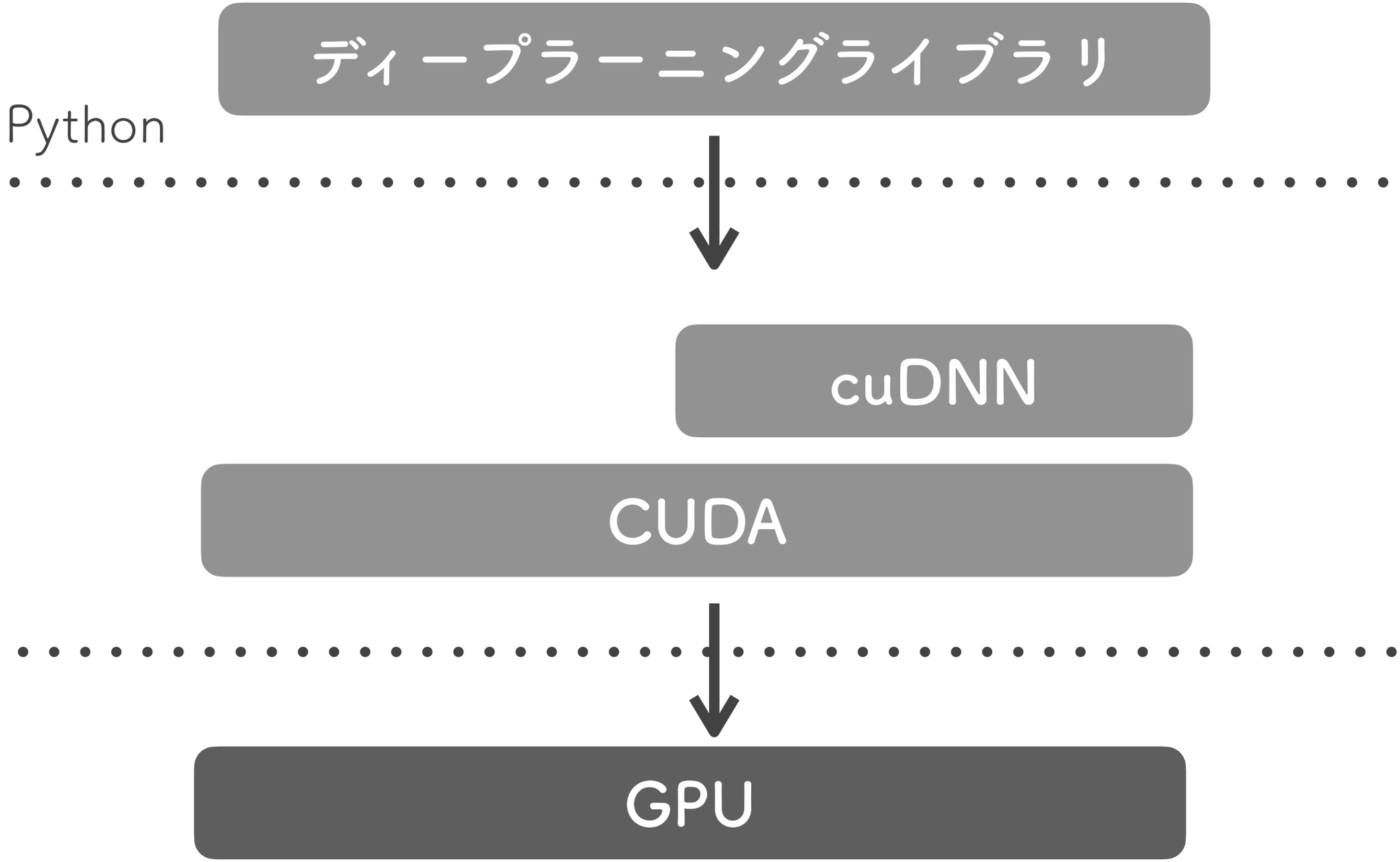
③ この報酬モデルを使いPPO（方策勾配法の代わりに、代理目的関数を最大化）によって、①の方策を強化学習させる。

→ helpful, honest, and harmless（13B IGPT > 175B GPT-3）

詳細：https://platform.openai.com/docs/model-index-for-researchers

6. 様々なライブラリについて

1. GPU (Graphics Processing Unit)への対応



2. テンソルへの対応

c

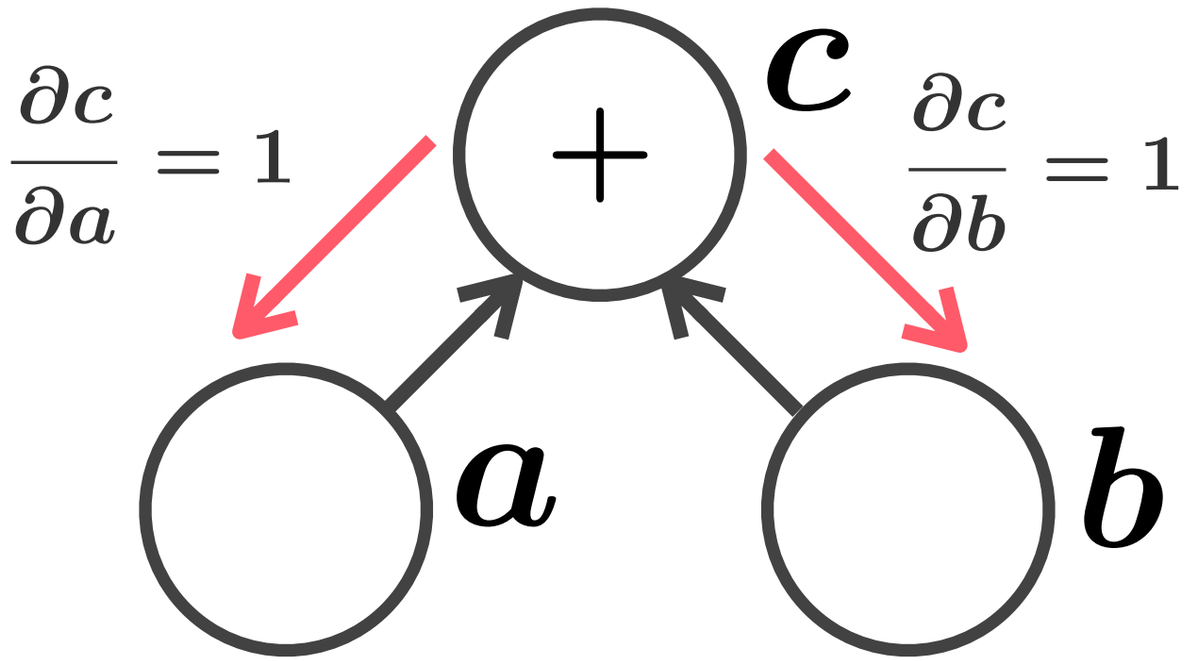
値

Ndarray

c

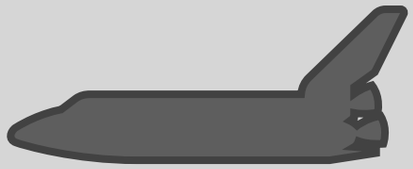
値	
a	b
$\frac{\partial c}{\partial a} = 1$	$\frac{\partial c}{\partial b} = 1$

Tensor



テンソルには、**自動微分**のために、ノードの値以外にも微分に関連する情報や、それらへの操作手続きも実装されている。

3. 深層学習ライブラリ小史：牧歌的な時代



2002 Torch

アカデミア・個人により開発
・メンテナンス

20

2007 Theano

2011

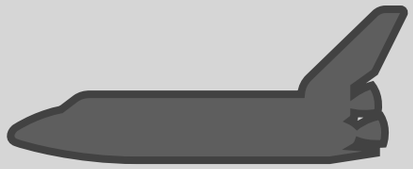
2012 ディープラーニング革命

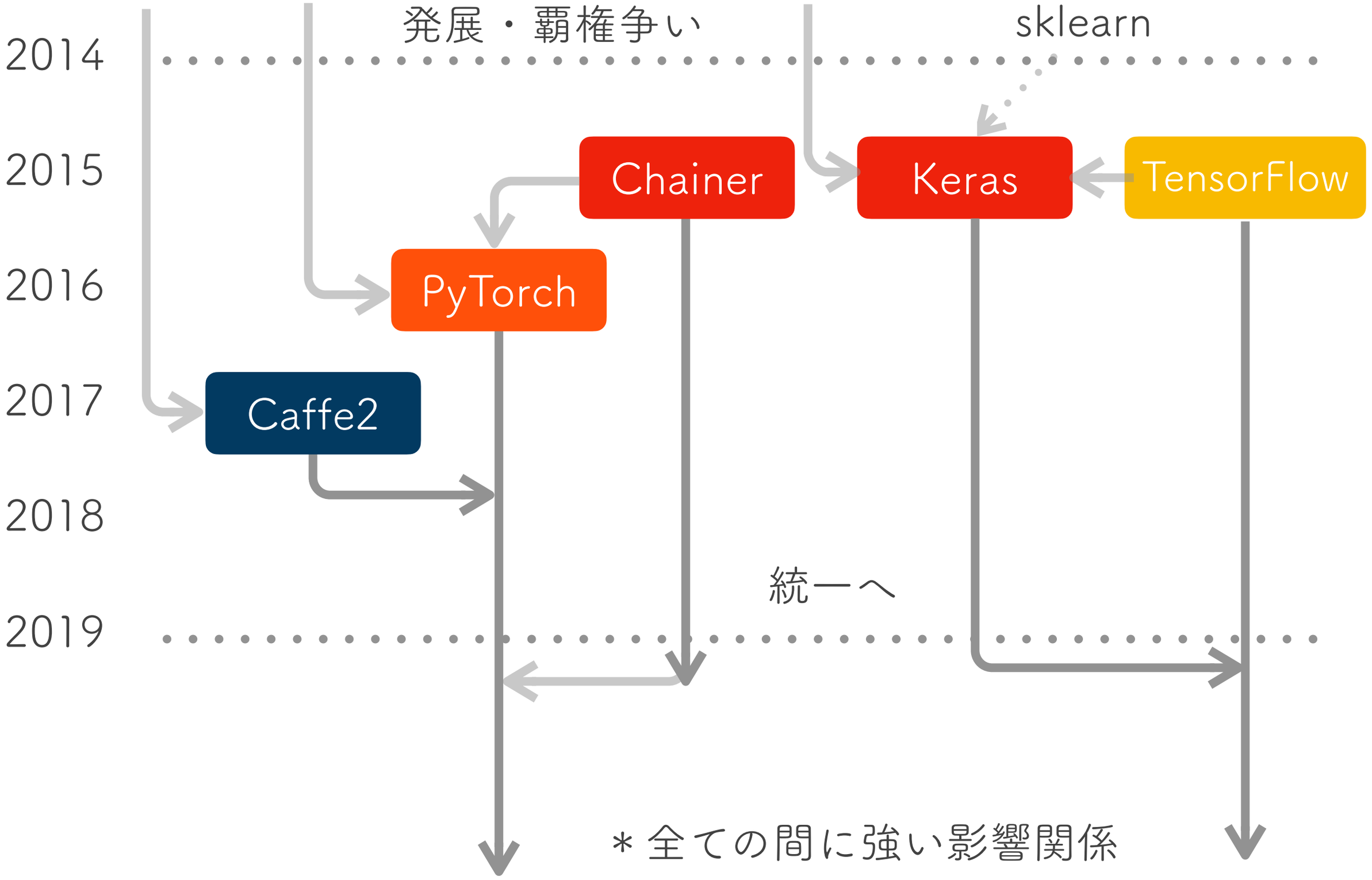
2013 Caffe

PyLearn2

https://www.edge-ai-vision.com/2016/01/the-caffe-deep-learning-framework-an-interview-with-the-core-developers/

3. 深層学習ライブラリ小史：戦国時代





4. PyTorch

- Facebook AI Research labが開発。TorchのPython化として始まったプロジェクトが起源→自動微分部分が主に
- シンプルなデザインのため、Pythonに習熟していれば簡単に（Pythonicに）使用できる（低レベルAPI）。
- 豊富な情報とデバッグが容易なため、研究者・開発者の間で大人気。

5. TensorFlow2

・Google Brainにより開発。グーグル内の非公開ライブラリDistBeliefを置き換える形で開発され公開。

・初心者から専門家、特に製品レベルにまで対応(TFX etc)

・最近になり素晴らしいデモ・exampleが拡充

・深層学習専用ではなく、広く科学技術計算用途に使用できる柔軟性 (例：量子計算 https://www.tensorflow.org/quantum)

・勉強・使用しにくさが最近著しく改善

・大規模モデルをデプロイするなら多分TF

6. Keras

multibackend Keras（リファレンス実装）

・F.Cholletにより研究プロジェクトの一環として開発。
Theano、TensorFlowなどの計算をバックエンドに使い、
エレガントなインターフェースを提供するため開発。

tf.Keras

・TensorFlowの公式高レベルAPIとして組み込まれたKeras。
TensorFlowを利用する際にはtf.Kerasを使うことが推奨。
TensorFlowの低レベルAPIとシームレスに
組み合わせることができ、自由度がさらに
向上した。

おすすめ（？）

PyTorch

Pythonに慣れている・機械学習側にも突っ込んだ研究目的なら一番おすすめ。**たくさんの実装済みモデル**が利用可。

Keras API

膨大な**ボイラープレートコードを毎回書くのが馬鹿馬鹿しい**人向け。機械学習はたくさん回すけれども、そこが本質じゃない時は特におすすめ。抽象化度高め。

おすすめ（？）

TensorFlow 2

デプロイまで考えているなら特に！推論高速化や量子化など。Kerasとシームレスに使得るので、自在な粒度も魅力。

JAX/Flax

関数型風、モデルとデータ（パラメータ）を切り分ける思想（Flax）。GPU/TPU向けに関数をJITコンパイル（実行時に、ハード固有のXLAコンパイル）。JIT後は訓練が早い。vampでバッチ化も容易。などなど魅力がいろいろ。**これからの若者向け**？

その他

HuggingFace Transformers

Transformerモデルを多量に集めたライブラリ。必須。
PyTorch/TensorFlow/JAX対応。

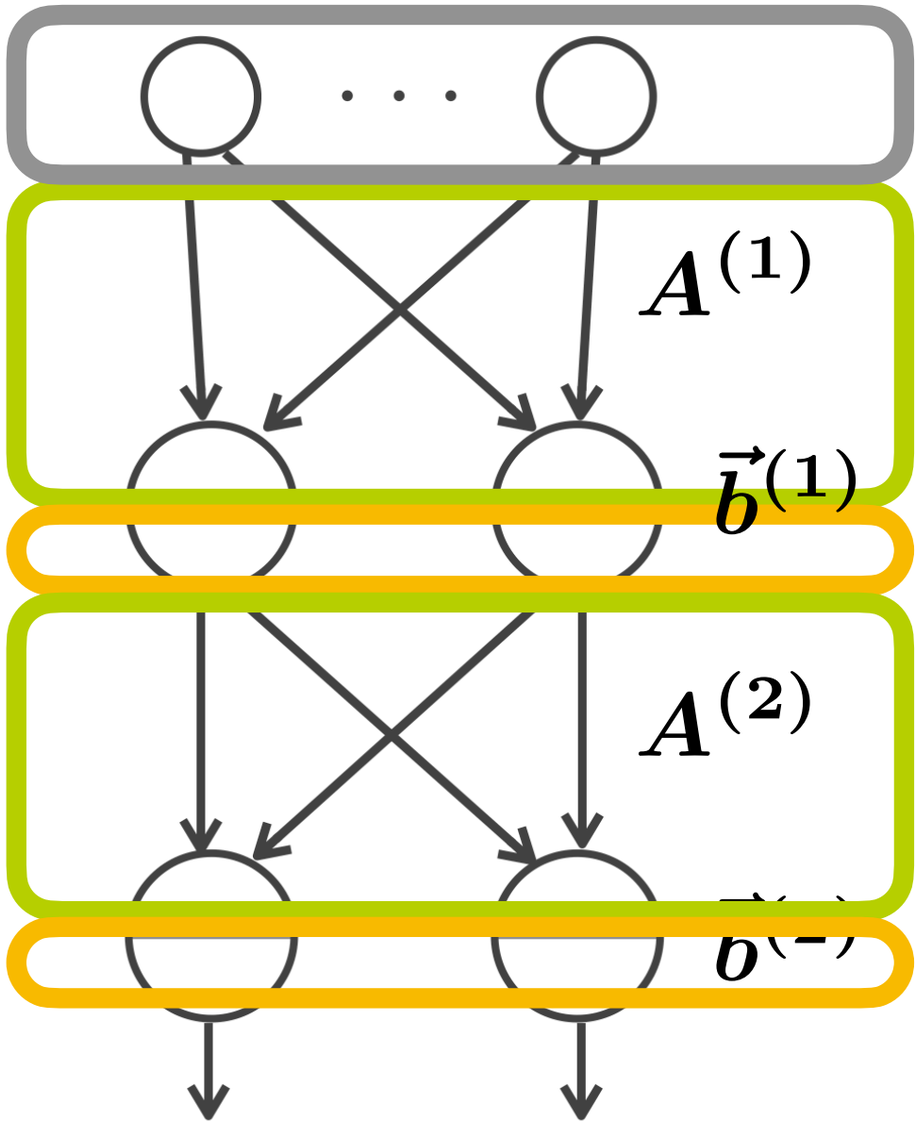
計算処理の観点からの層

$$\vec{u}^{(1)} = \vec{x}\, A^{(1)} + \vec{b}^{(1)}$$

$$\vec{z}^{(1)} = \sigma(\vec{u}^{(1)})$$

$$\vec{u}^{(2)} = \vec{z}^{(1)}\, A^{(2)} + \vec{b}^{(2)}$$

$$\hat{\vec{y}} = \sigma(\vec{u}^{(2)})$$



特徴量ベクトル単位での線形変換（アフィン変換）と活性化関数での変換処理（層）に分けられる。

PyTorch①：モデルの作成

Torch.nn.Moduleクラスを継承し、自身のモデルをクラスとして実装する

PyTorch①：モデルの作成

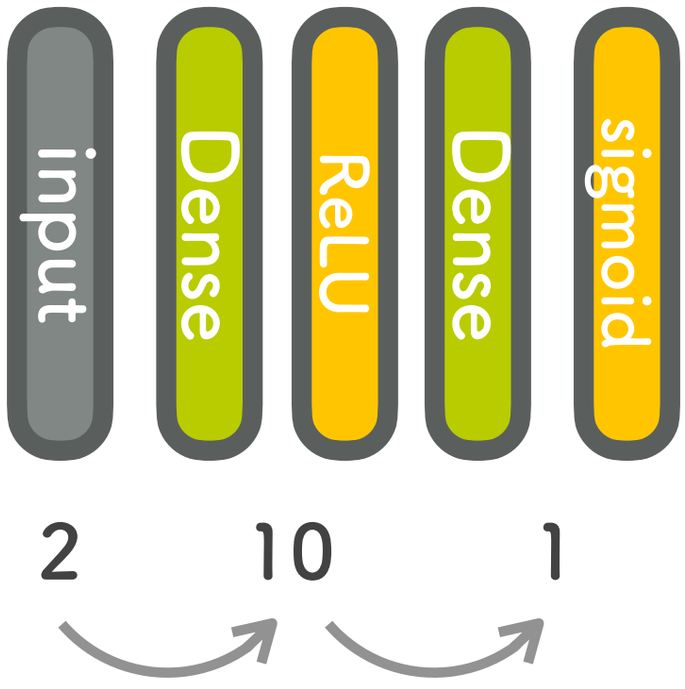
Torch.nn.Moduleクラスを継承し、自身のモデルをクラスとして実装する

```
class MyNet(torch.nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        '''
        順伝播に使用する層を用意
        '''
    def forward(self, x):
        '''
        順伝播計算の式
        '''
        return x
```

PyTorch①：モデルの作成

Torch.nn.Moduleクラスを継承し、自身のモデルをクラスとして実装する

```
class MyNet(torch.nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        sefl.dense1 = torch.nn.Linear(2,10)
        sefl.dense2 = torch.nn.Linear(10,1)
    def forward(self, x):
        x = self.dense1(x)
        x = torch.relu(x)
        x = self.dense2(x)
        x = torch.sigmoid(x)
        return x
```



PyTorch①：モデルの作成

Torch.nn.Moduleクラスを継承し、自身のモデルをクラスとして実装する

そのクラスのインスタンスとして、一個のモデルを実際に作成する：

```
model = MyNet()
```

PyTorch②モデルの訓練

以下の自動微分計算をiteration回数だけ繰り返す

```
model.train()

for i in range(iterations):

    y_pred = model(x_train)
    loss = criterion(y_pred, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

PyTorch②モデルの訓練

以下の自動微分計算をiteration回数だけ繰り返す

```
model.train() 学習モード

for i in range(iterations):

    y_pred = model(x_train) 順伝播、計算グラフ構築
    loss = criterion(y_pred, y_train) 誤差関数の計算

    optimizer.zero_grad() 計算グラフにおいて、勾配をゼロ初期化
    loss.backward() 自動微分
    optimizer.step() パラメータ更新
```