

同志社大学「最適化法」 第15回 2024年1月26日(金)

機械学習と自動微分

瀧川 一学

京都大学 国際高等教育院

takigawa.ichigaku.8s@kyoto-u.ac.jp



この講義スライドの最新版PDF

<https://itakigawa.github.io/data/autograd.pdf>



講義中に質問があればSlidoへ

<https://app.sli.do/event/d13KtHT3VxFbmLvxPSQjn>

自己紹介：瀧川一学

- クロスアポイントメント制度により公式にダブルワーク
 - 京都大学国際高等教育院
データ科学イノベーション教育研究センター 特定教授
 - 北海道大学化学反応創成研究拠点 特任教授
- 研究上の関心
 - 機械学習 (離散構造を伴う機械学習)
 - 機械発見 (機械学習の自然科学研究での利活用)
- <https://itakigawa.github.io/>

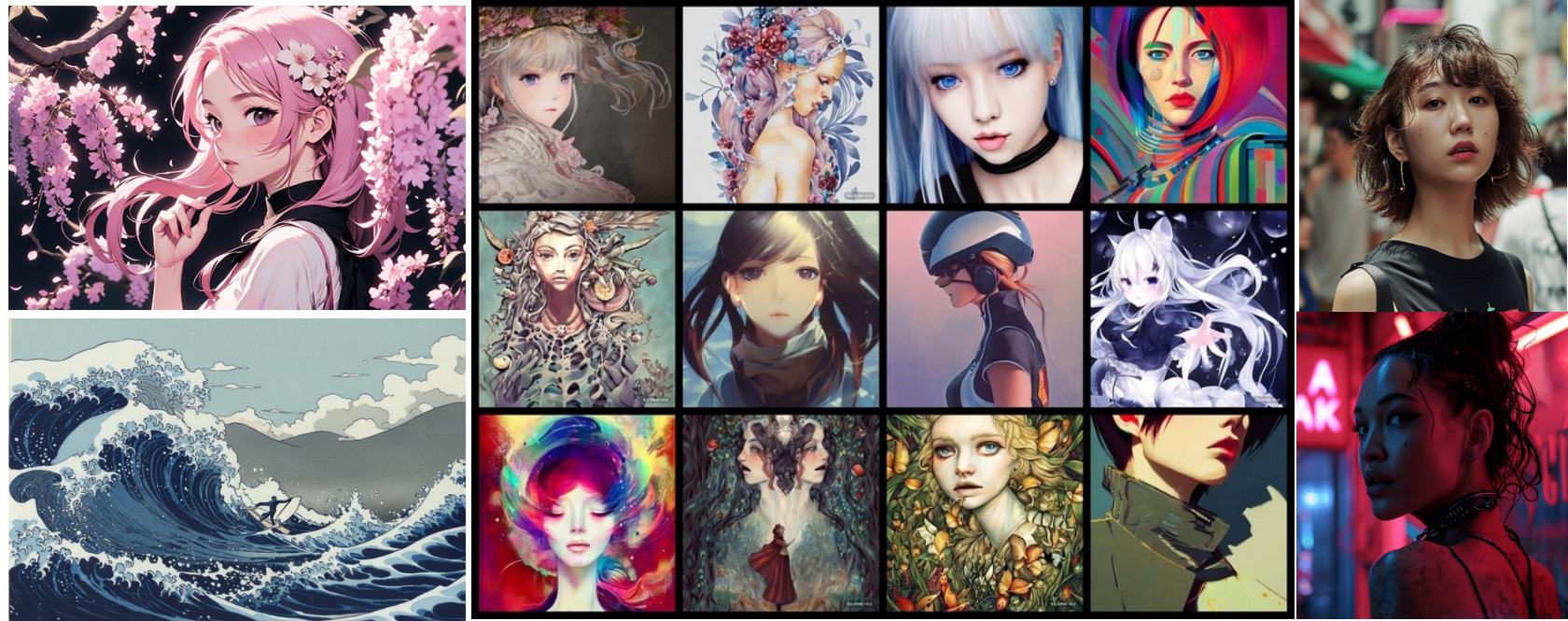
今日のたった3つの内容

1. 機械学習のしくみ
2. 深層学習と勾配法
3. 自動微分 (アルゴリズム微分)

2023年は「生成AI」ブームとやらでしたね…

- 画像生成AIが爆速で進化した2023年をまとめて振り返る

<https://ascii.jp/elem/000/004/174/4174570/>



広告とAI画像

- 広告ニーズが大事なモデル・イラストレータ・写真家の仕事にも影響！？



<https://www.itmedia.co.jp/news/articles/2310/05/news179.html>

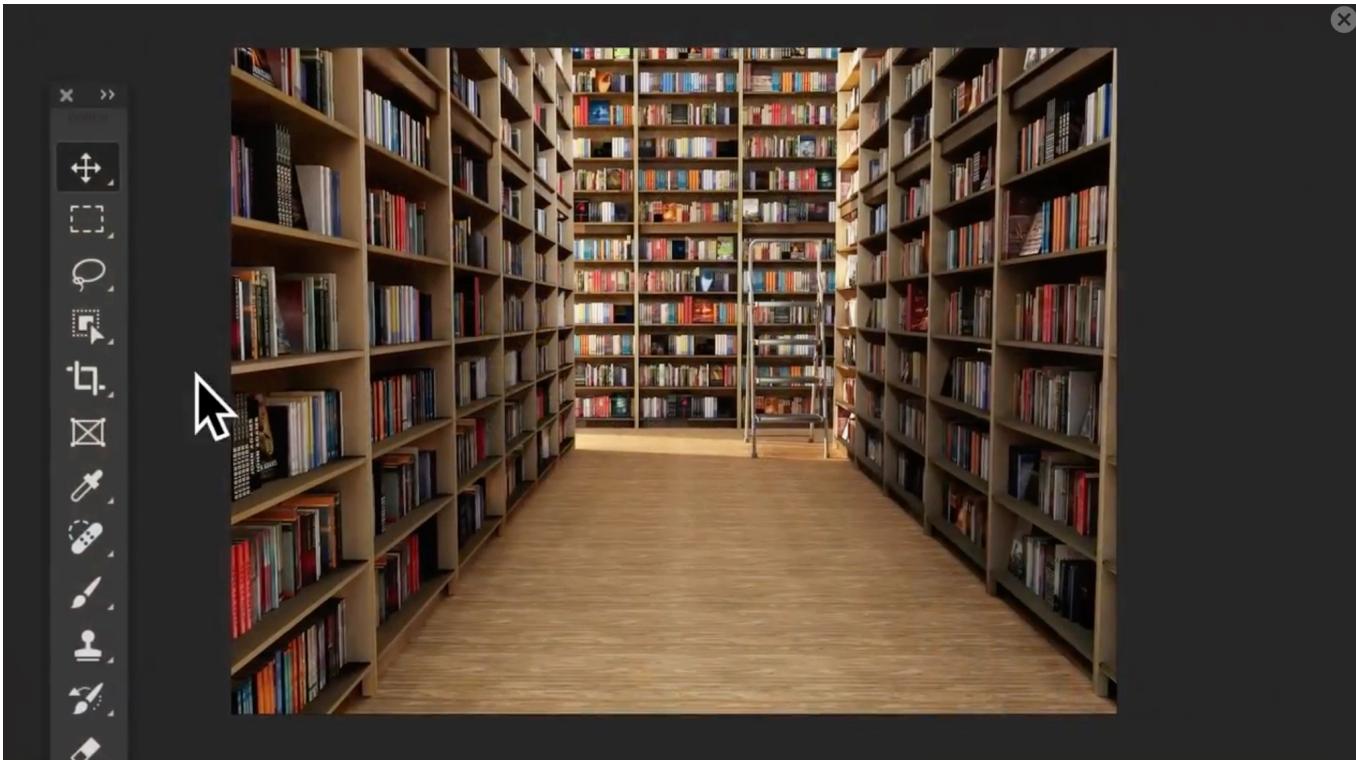
広告とAI画像

- 広告ニーズが大事なモデル・イラストレータ・写真家の仕事にも影響！？



<https://www.itmedia.co.jp/news/articles/2310/05/news179.html>

Adobe Photoshop 生成塗りつぶし・生成拡張



<https://www.adobe.com/jp/products/photoshop/generative-fill.html>

OpenAI ChatGPT

- ChatGPT 2022年11月に公開。わずか2カ月で月間1億ユーザーを獲得
- 仕事や教育のあり方への影響から企業や国を動かし生成AIブームを牽引
- 一方、GAFAMなどの巨大IT企業は2023年どこも大規模リストラを実行



OpenAI ChatGPT

- 要約・翻訳・質問応答などだけでなく、画像・音声やソフトウェアを操作・制御するインターフェースとして「自然言語」が使えるように
- ChatGPTでできること (ChatGPT自身による要約)

1. テキスト生成と対話：

- 会話形式での応答：質問に対する詳細な回答や、対話形式でのやり取りが可能。
- ストーリーテリング：物語やシナリオの作成。
- コンテンツ作成：記事、エッセイ、スピーチの作成サポート。

2. 知識ベースの活用：

- 情報検索：特定のトピックに関する情報提供。
- 教育支援：教材の解説や学習サポート。
- 語学学習：言語学習の支援や翻訳。

3. クリエイティブな支援：

- 詩や歌詞の作成。
- アイデア生成：ビジネスアイデア、デザインコンセプトなど。
- レシピやDIYプロジェクトの提案。

4. 技術的なアシスタンス：

- プログラミングのサポート：コード例の提供、デバッグ支援。
- 技術的なドキュメント作成や解説。

5. 拡張機能 (GPTsなど) :

- 画像生成 (DALL-Eなど) : テキストから画像を生成する機能。
- 音声認識と生成：テキストを音声に変換し、逆もまた可能。
- データ分析と可視化：データセットの分析やグラフ作成の支援。

6. カスタマイズと特化機能：

- 特定の業界や分野に特化した情報提供。
- ユーザーのニーズに合わせたカスタマイズされた応答。

Microsoft Copilot (ググらないでコパる未来？)

- Microsoftは早くからOpenAIに投資・連携し、ChatGPTやDALL-Eなどを技術基盤として生成AIを自社製品・自社サービスに全面展開！
- **生成AIの需要への期待に乗って時価総額は一時3兆ドル(440兆円)超え**
→ 3兆ドル超はAppleに続いて史上2社目 (🇬🇧🇫🇷🇮🇹あたりのGDP超え！)
- **検索 Bing (特にブラウザEdge)へChatGPT統合**
→ ネットと接続されていないChatGPTでWeb検索結果を使えるように
- **GitHub Copilot(コードの自動補完)を展開**
 - ・ プログラミングの方法を変革 (一度使うと離れられない！？)
 - ・ よくあるものならテキスト指示だけでプログラムが一瞬で作れるように
- **Windows 11及びMicrosoft 365でのCopilot/Copilot Pro導入**

MS製品は使用人口が多いので影響が大きい！

- 生成AIの本格導入：Windows 11/ Microsoft 365にCopilotを搭載
- MS製品 (Word, Excel, Powerpoint, Outlook, Teams)全般で生成AIを介した操作ができるよう
 - ・ 長文要約、翻訳、下書き、返信文の自動生成、英文添削や修正
 - ・ いくつかのメモ書きテキストを要約し綺麗なWordレイアウト作成
 - ・ 報告書からPowerpointスライドを自動生成
 - ・ Powerpointスライドに自動で挿絵・効果・背景などをたす
 - ・ データからExcelの表やグラフ可視化を自動生成、その傾向を要約
 - ・ Web会議の録音から議事録の文字起こし、その要約を作成
 - ・ Windowsでのファイル検索支援、操作支援

音声・音楽・動画の生成…

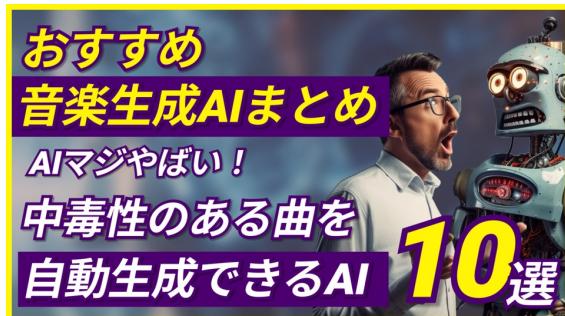
- イラストや音楽からプログラムまで望むものを誰でも手軽に作れる世界
→一方で、権利や悪用の問題などが顕在化し議論されている

ITmedia NEWS > ネットの話題 > 音楽生成AI「Suno AI」が話題、文章から楽曲を瞬時…

音楽生成AI「Suno AI」が話題、文章から楽曲を瞬時に作成 プロの音楽家も「これはヤバい」と驚愕

2023年12月14日 19時03分 公開

[松浦立樹, ITmedia]



生成AIは「機械学習」

- 人間の仕組みとは全然違う+世の中で喧伝される情報はかなりビジネス寄りの価値観で誇張されている点に留意
- 参考：人間の言語能力とは何か (岩波「科学」2023年12月号)
 - 言語学者によるChatGPT懐疑記事へのAI側2名からの討論記事
 - 全文公開されています <http://hdl.handle.net/2433/286548>
 - 人工知能という分野が謙虚であったことなど一度もない
ノバート・ホーンスティン(メリーランド大), 翻訳：折田奈甫 (早稲田大)
 - 指定討論1：言語モデルは単純な学習則で複雑な推論を実現する
岡野原大輔 (Preferred Networks)
 - 指定討論2：なぜ経験則は説明の論理として受け入れがたいか
瀧川一学 (京都大)
 - 解説
折田奈甫 (早稲田大)

1. 機械学習のしくみ

典型的な適用場面

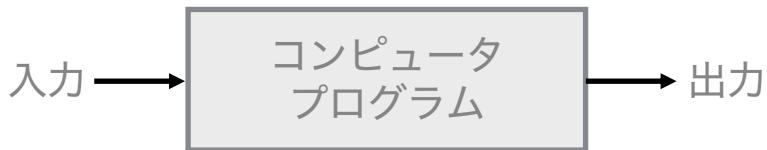
写真をAさんかBさんかに分類するコンピュータプログラムを作りたい。
(大量の写真を人手分類するのは大変)



- 人間は簡単にできるが、どうやってやっているのか原理/ルールはよく分からない
- 言語化できないルールはプログラムにできない
- 髮型、角度、照明、背景、表情、化粧、年齢、などを考えると明示的な分類ルールを考えだすのはとても難しい

プログラム=入力を出力に変換する関数

入力を受け取って、書き換え、出力する手順



```
def my_function(x1, x2, \
                 a1=4, a2=5, b1=6, b2=10):
    u = x1*x1 - a1*x1*x2 + a2*x1*x2
    v = -b1*x2 + b2
    y = u + v
    return y
```

my_function(1,1)

6

my_function(1,1, b1=3)

9

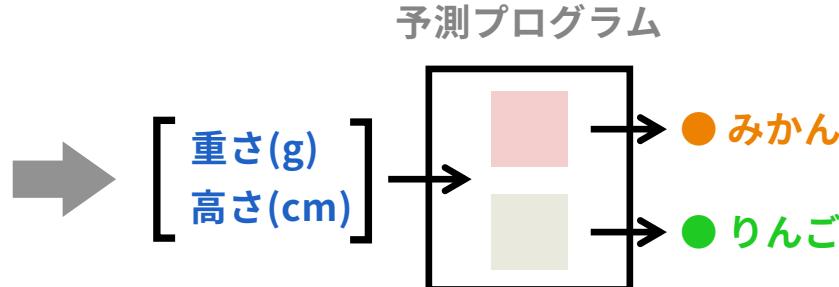
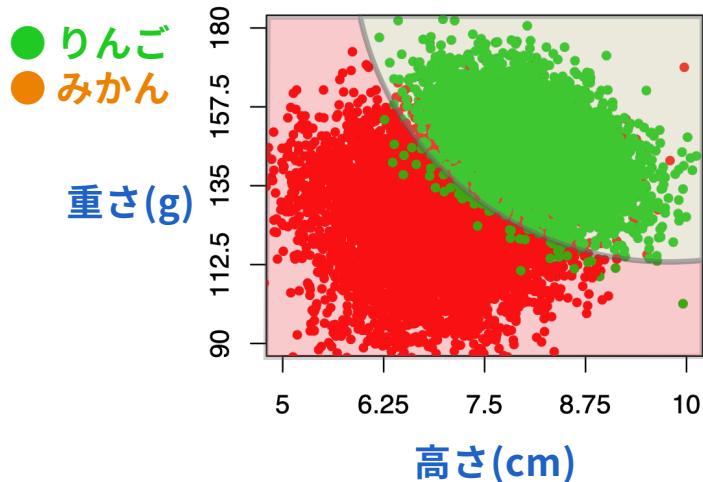
機械学習 = 新しい(雑な)プログラムの作り方

たくさんの入出力の見本データによって間接的に見本例の入出力を再現するプログラムを作り出す技術 (Software 2.0とも)



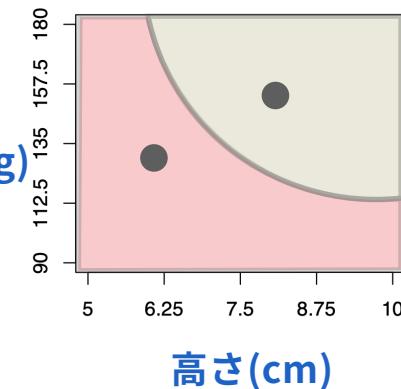
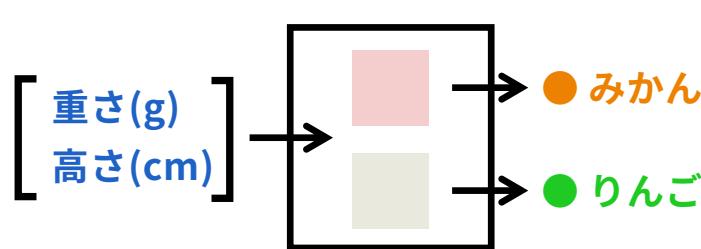
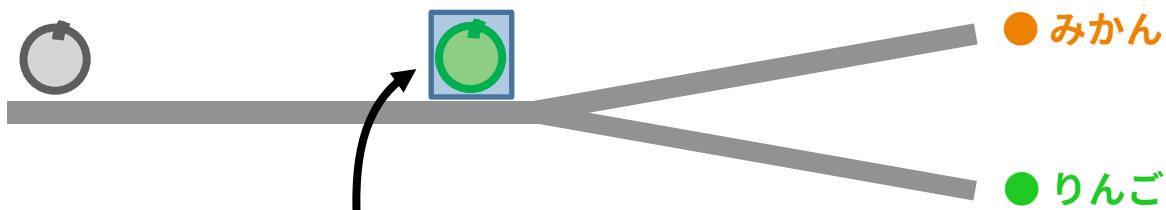
機械学習 = データを予測に変える技術

見本例のように 「みかん」 or 「りんご」 を出力するプログラムを作る



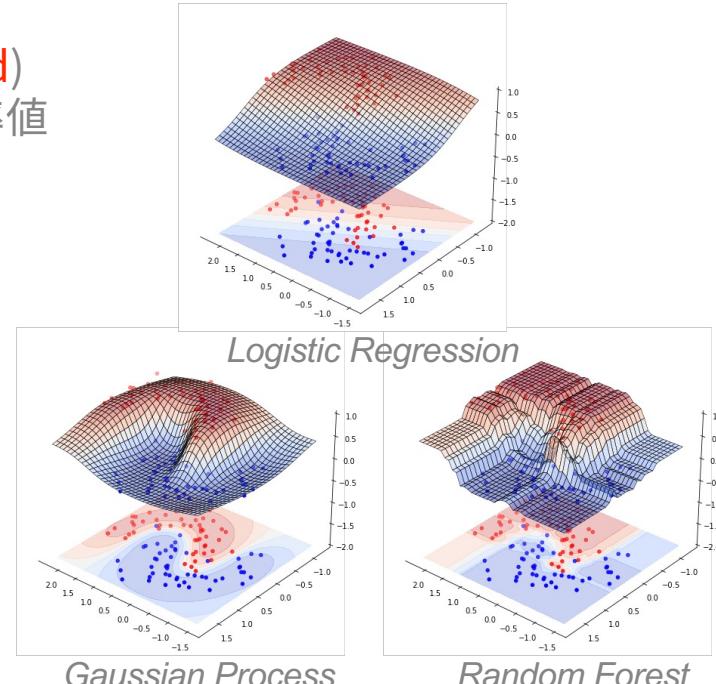
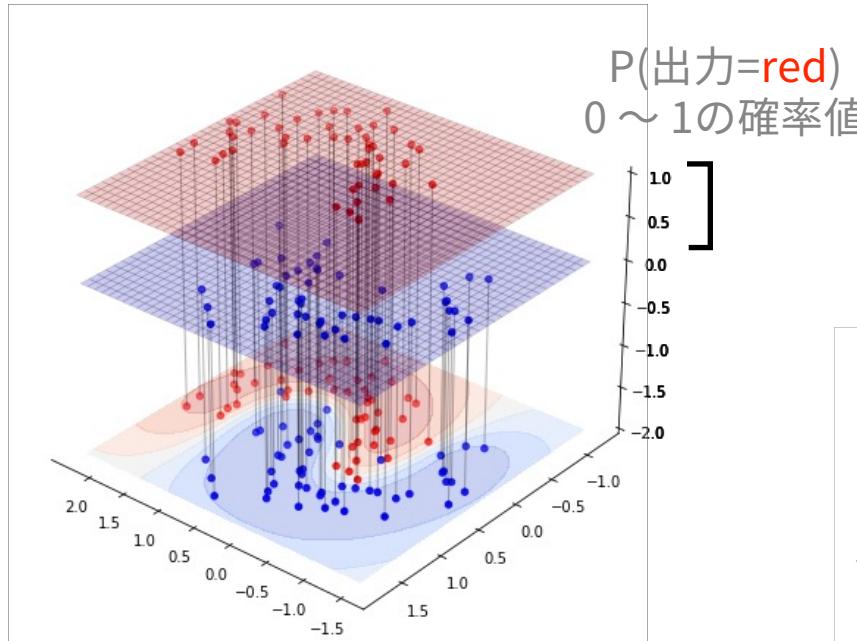
機械学習 = データを予測に変える技術

予測プログラムを作ったときに見せた見本例ではない例に対して
「みかん」 or 「りんご」 を予測することができる！



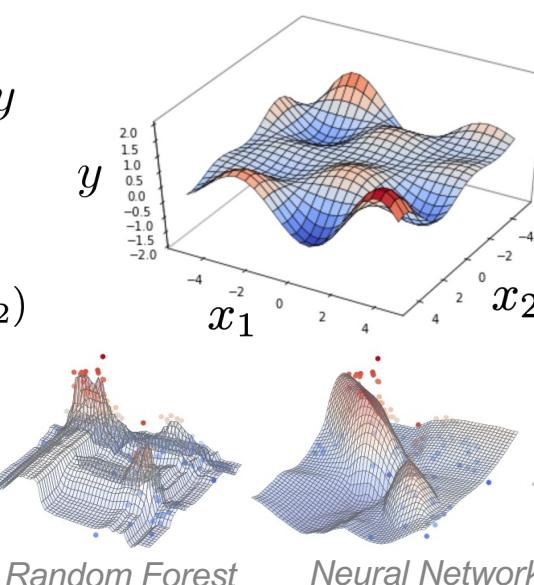
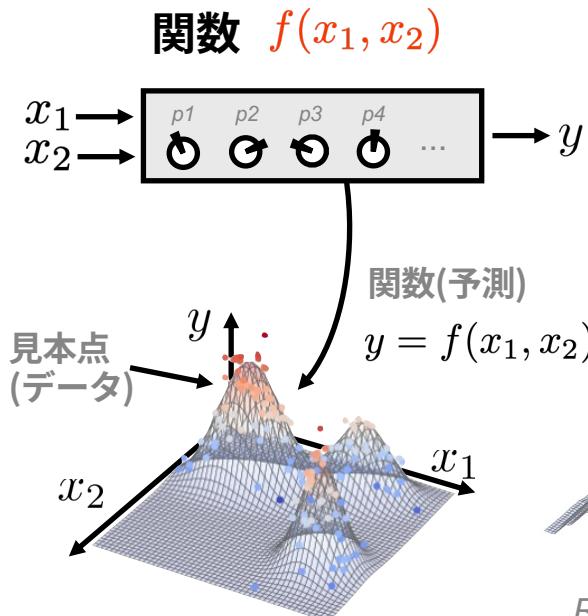
機械学習 = 関数フィッティングによるデータ内挿

「みかん」 or 「りんご」 のクラス分類のケースでは入力変数から
クラス確率を予測する関数を見本点にフィッティングする



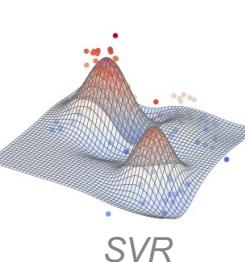
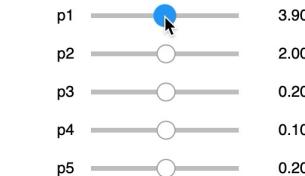
関数フィッティング=「最適化」

モデルの内部パラメタ値を変えると関数形(入出力の挙動)を変えられる
→ 機械学習=与えられた見本例に合うようにパラメタ値を最適化

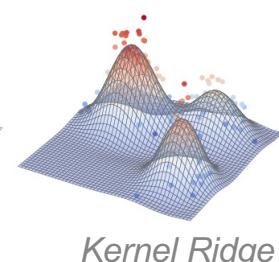


Random Forest

Neural Network



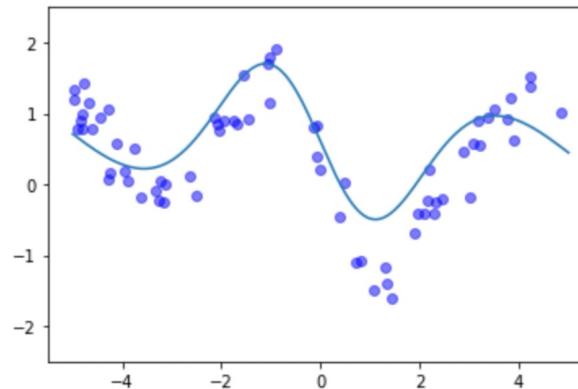
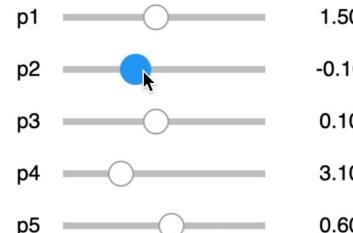
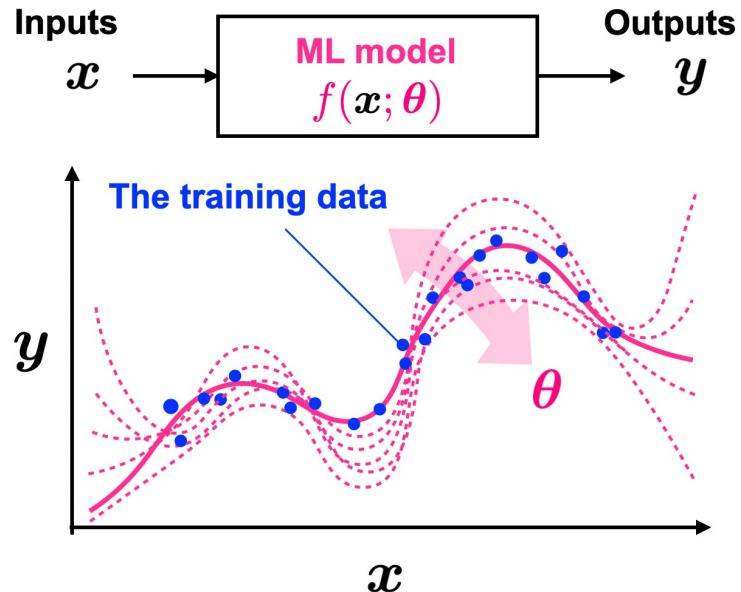
SVR



Kernel Ridge

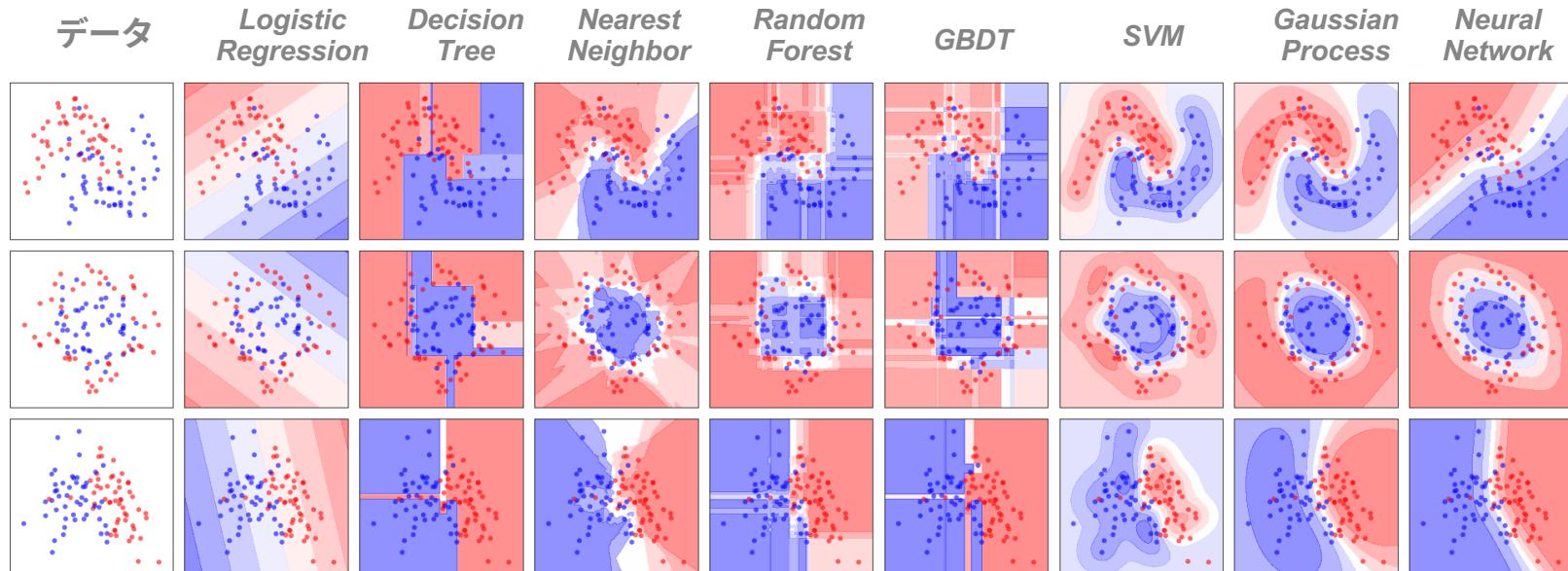
機械学習 = (高次元での)曲面のフィッティング

- 入力: 1次元、出力: 1次元であれば点への曲線あてはめ



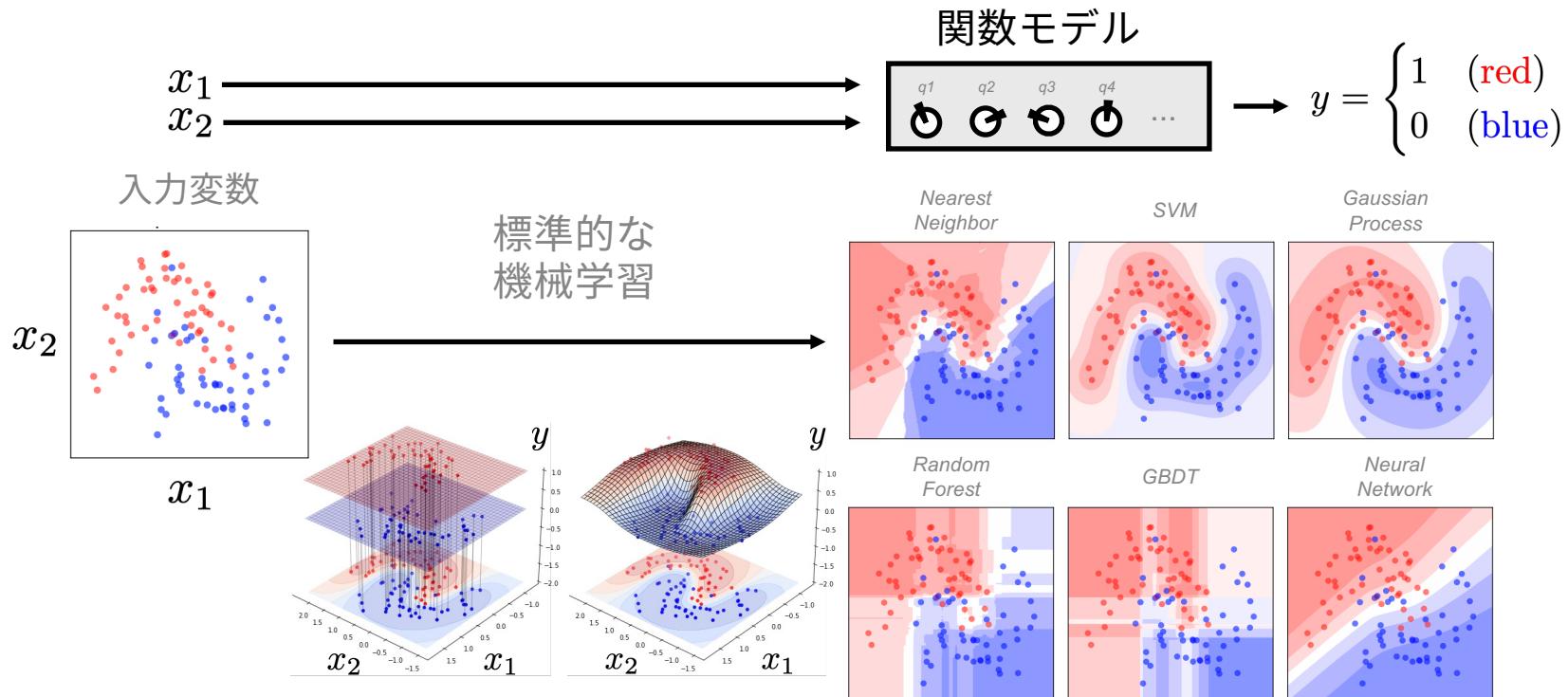
機械学習の方法は関数モデルの数だけある

フィッティングに用いる関数モデルが違えば異なる手法になる
→ 色々な方法が存在する (深層学習で使う関数モデルである
ニューラルネットワークもその一つ)

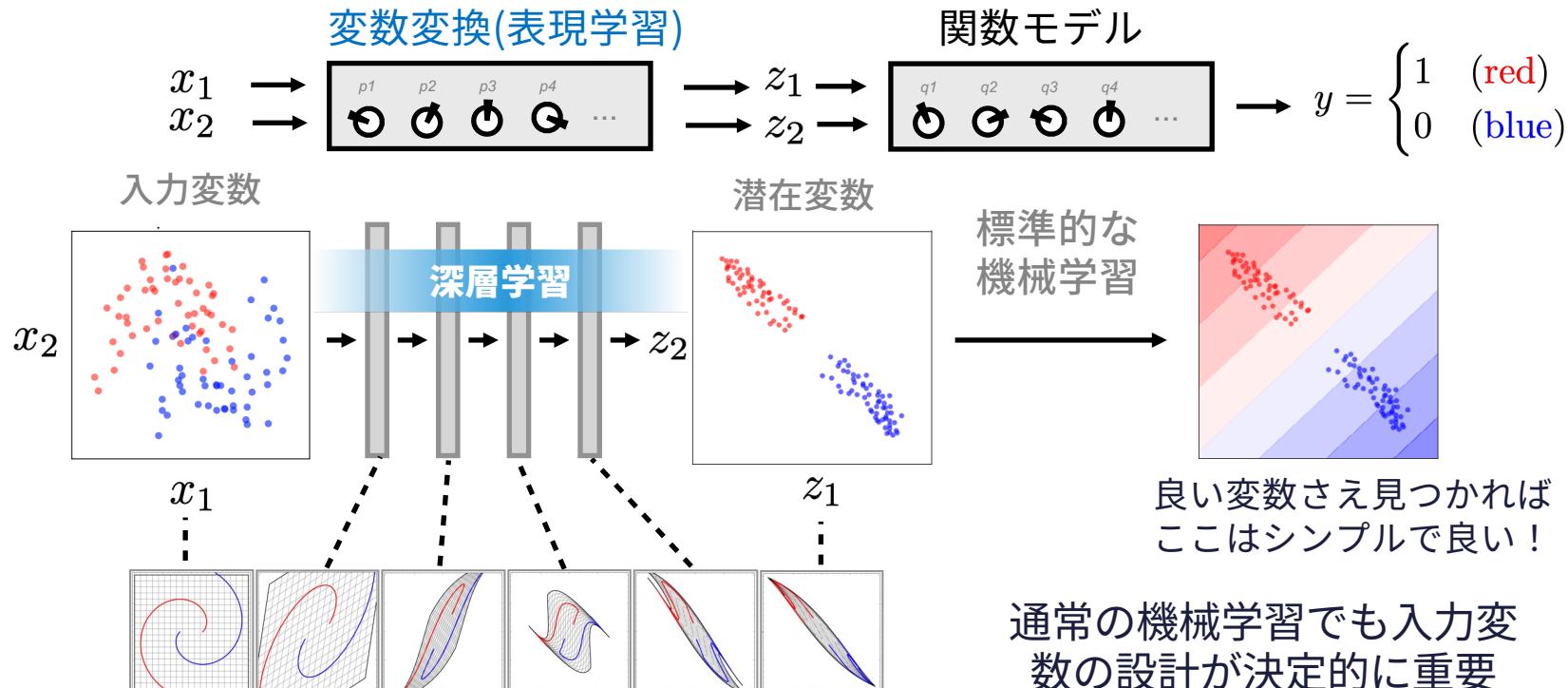


2. 深層学習と勾配法

標準的な機械学習 vs 深層学習(表現学習)



標準的な機械学習 vs 深層学習(表現学習)



深層学習モデルの学習

- 深層学習のモデルがどれだけ複雑でも
「勾配法」によって統一的にパラメタ値の最適化ができる
- 深層学習モデルの出力値 $f(x)$ と正解値 y の食いちがいを
「損失関数」 $\ell(f(x), y)$ で数値指標にする
 - 回帰の例：平均二乗誤差 (MSE)
 - 分類の例：交差エントロピー (Cross Entropy)
- 深層学習モデルの学習 = モデルの内部パラメタ値を変化させて損失関数の値を最小化する 「最適化 (optimization)」

$$\text{Minimize}_{\theta} L(\theta) \text{ ただし } L(\theta) = \sum_{i=1}^n \ell(f(x_i, \theta), y_i) + \Omega(\theta)$$

例：2値分類問題

データ $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ 各 y_i は 0 か 1

モデル $f(x) = \sigma(ax + b) = \frac{1}{1 + e^{-(ax+b)}} = P(y = 1|x)$ 0~1 の 値
(確率とみなす)

損失関数 $L(a, b) = -\log \prod_{i:y_i=1} P(y = 1|x_i) \prod_{i:y_i=0} P(y = 0|x_i)$
(負の対数尤度)

$$= -\sum_{i=1}^N (y_i \log f(x_i) + (1 - y_i) \log(1 - f(x_i)))$$

注：尤度は大きくしたい $\rightarrow (-1) \times$ 尤度は小さくしたい

「交差エントロピー(Cross Entropy)損失」や「対数損失」とも呼ばれる。

例：1変数1出力のニューラルネット

$$f(x) = \sigma(ax + b) = \frac{1}{1 + e^{-(ax+b)}}$$

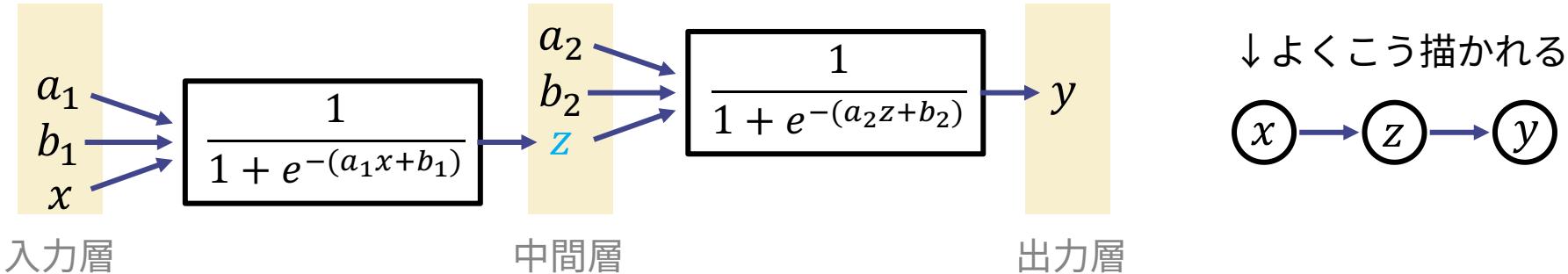
訓練データ $(x_1, y_1), (x_2, y_2), (x_3, y_3)$

$$\begin{array}{ccccccc} a & \nearrow & & & & & \\ b & \nearrow & & & & & \\ x & \nearrow & & & & & \\ & & \text{Linear} & \rightarrow z \rightarrow & \text{Sigmoid} & \rightarrow \hat{y} & \\ & & ax + b & & & & \\ & & & & \sigma(x) = \frac{1}{1 + e^{-x}} & & (\text{活性化関数}) \end{array}$$

解くべき最適化問題： Minimize $L(\textcolor{magenta}{a}, \textcolor{magenta}{b})$

$$\begin{aligned} L(\textcolor{magenta}{a}, \textcolor{magenta}{b}) &= - \left(y_1 \log \frac{1}{1 + e^{-(\textcolor{magenta}{a}x_1 + \textcolor{magenta}{b})}} + (1 - y_1) \log \left(1 - \frac{1}{1 + e^{-(\textcolor{magenta}{a}x_1 + \textcolor{magenta}{b})}} \right) \right) \\ &\quad - \left(y_2 \log \frac{1}{1 + e^{-(\textcolor{magenta}{a}x_2 + \textcolor{magenta}{b})}} + (1 - y_2) \log \left(1 - \frac{1}{1 + e^{-(\textcolor{magenta}{a}x_2 + \textcolor{magenta}{b})}} \right) \right) \\ &\quad - \left(y_3 \log \frac{1}{1 + e^{-(\textcolor{magenta}{a}x_3 + \textcolor{magenta}{b})}} + (1 - y_3) \log \left(1 - \frac{1}{1 + e^{-(\textcolor{magenta}{a}x_3 + \textcolor{magenta}{b})}} \right) \right) \end{aligned}$$

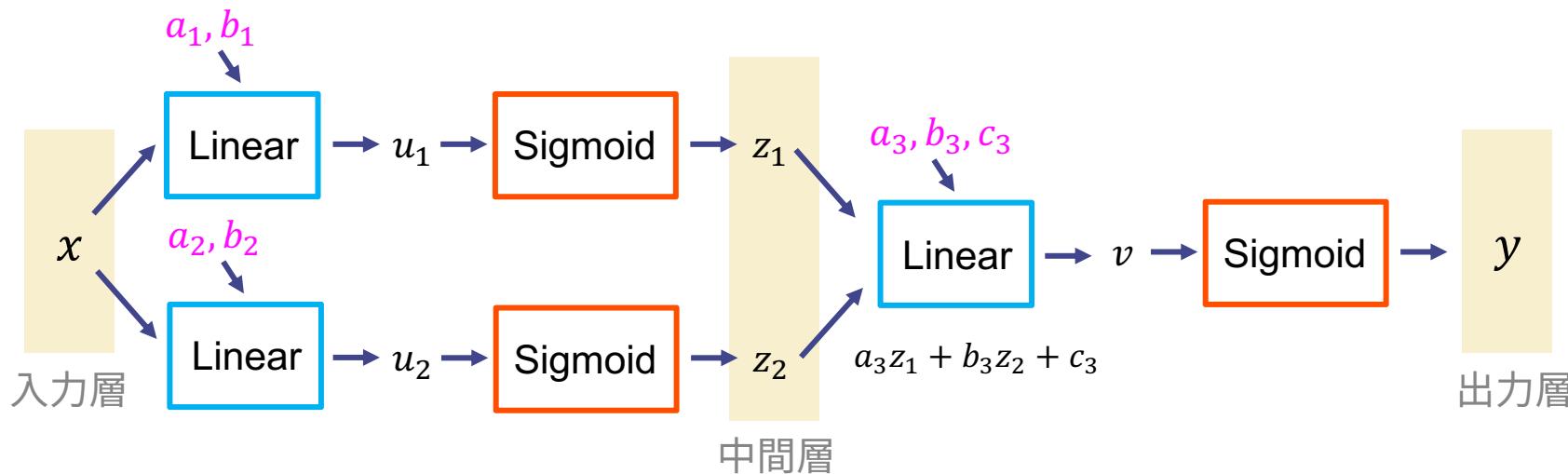
例：1変数1出力の3層ニューラルネット (中間変数1つ)



解くべき最適化問題： Minimize $L(a_1, a_2, b_1, b_2)$

$$\begin{aligned}
 L(a_1, a_2, b_1, b_2) = & - \left(y_1 \log \frac{1}{1 + e^{-(\frac{1}{1 + e^{-(a_1 x_1 + b_1)} + b_2})}} + (1 - y_1) \log \left(1 - \frac{1}{1 + e^{-(\frac{1}{1 + e^{-(a_1 x_1 + b_1)} + b_2})}} \right) \right) \\
 & - \left(y_2 \log \frac{1}{1 + e^{-(\frac{1}{1 + e^{-(a_1 x_2 + b_1)} + b_2})}} + (1 - y_2) \log \left(1 - \frac{1}{1 + e^{-(\frac{1}{1 + e^{-(a_1 x_2 + b_1)} + b_2})}} \right) \right) \\
 & - \left(y_3 \log \frac{1}{1 + e^{-(\frac{1}{1 + e^{-(a_1 x_3 + b_1)} + b_2})}} + (1 - y_3) \log \left(1 - \frac{1}{1 + e^{-(\frac{1}{1 + e^{-(a_1 x_3 + b_1)} + b_2})}} \right) \right)
 \end{aligned}$$

例：1変数1出力の3層ニューラルネット（中間変数2つ）



解くべき最適化問題： Minimize $L(a_1, a_2, a_3, b_1, b_2, b_3, c_3)$

とにかく非線形の多変数関数の最小値を求めれば良い

注：ただし深層学習モデルではパラメタ数が巨大！

■ 深層学習モデルのパラメタ数の例

- ResNet50 2600万
 - AlexNet 6100万
 - ResNeXt101 8400万
 - VGG19 1億4300万
 - LLaMa 650億
 - Chinchilla 700億
 - GPT-3 1750億
 - Gopher 2800億
 - PaLM 5400億
-
- 畳み込みニューラルネット(CNN)
- Transformerモデル(言語モデル)

機械学習と最適化

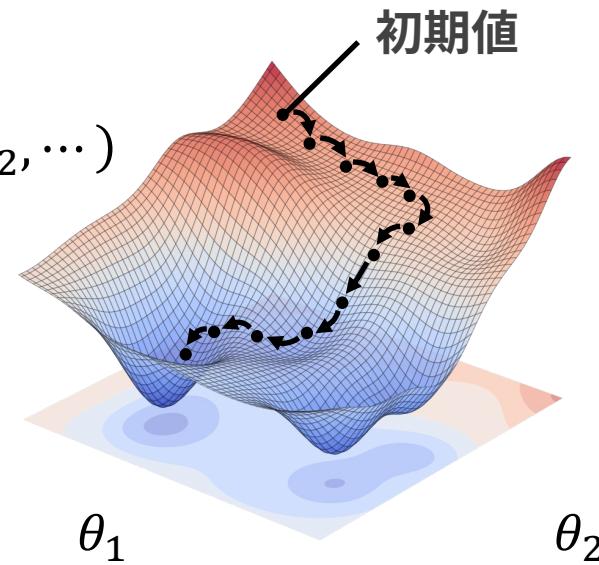
機械学習の関数モデルのフィッティング

= とにかく **非線形の多変数関数の最小値を
求めれば良い** (制約なし連続**最適化**)

方針

1. $(\theta_1, \theta_2, \dots)$ を適当に初期化
2. 関数値 $L(\theta_1, \theta_2, \dots)$ が小さくなる
ように少し $(\theta_1, \theta_2, \dots)$ を変える
3. 以上を繰り返す

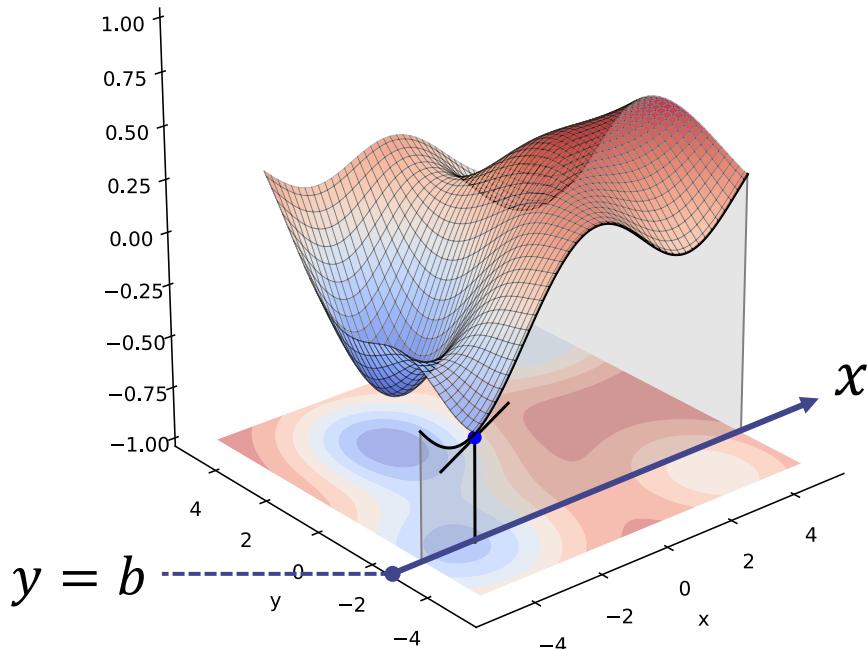
$$L(\theta_1, \theta_2, \dots)$$



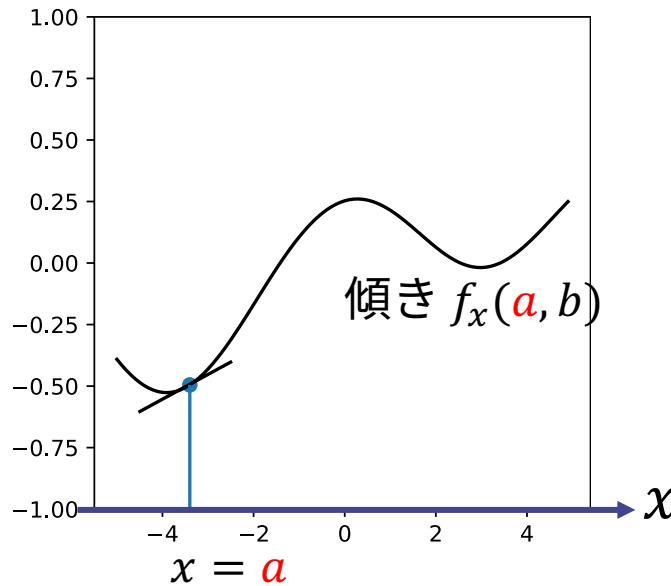
ヒントは偏微分係数

関数 $f(x, y)$ の点 (a, b) における偏微分係数

$$f_x(a, b) = \lim_{h \rightarrow 0} \frac{f(a+h, b) - f(a, b)}{h}$$



関数値 $f(a, b)$ は a を少し変えたら
増えるのか減るのか？

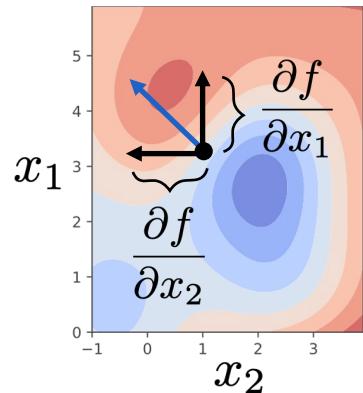


勾配

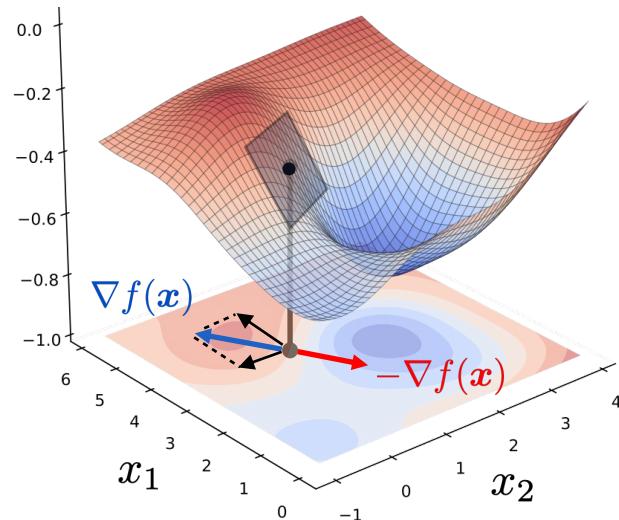
勾配 (Gradient)

- = 各変数での偏微分係数を並べたベクトル
- = 等高線に直交する方向
(等位面の法線ベクトル方向)

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{bmatrix}$$

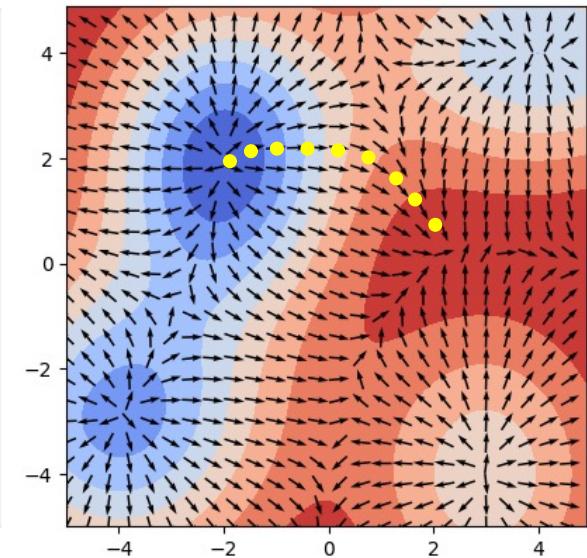
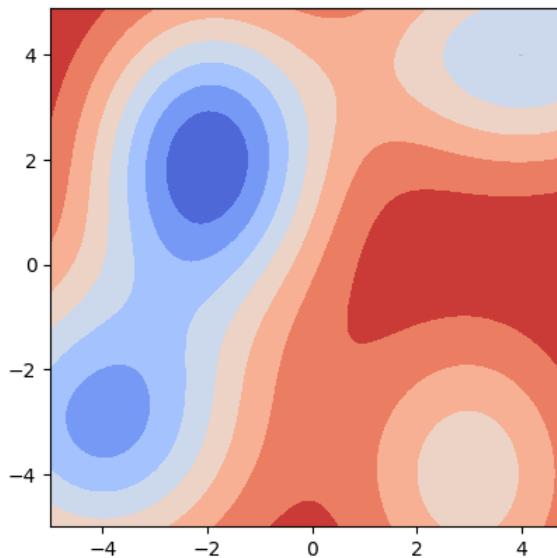
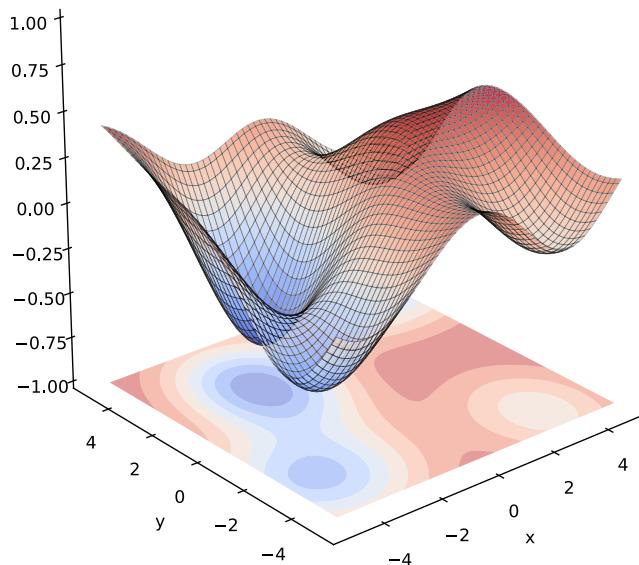


$$y = f(x_1, x_2)$$



勾配

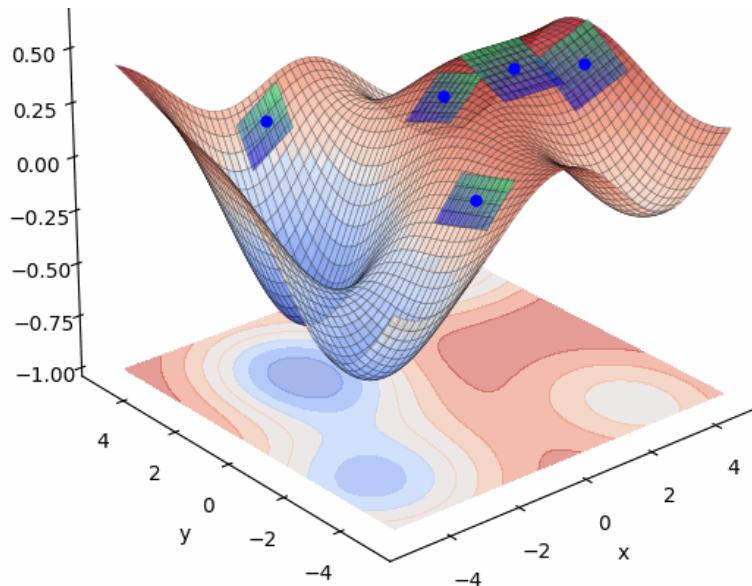
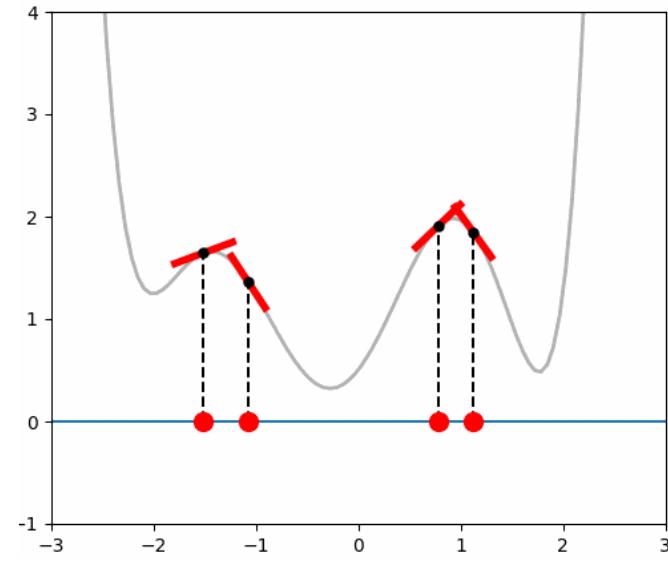
ベクトル場 = 各点でベクトルが定まったもの



初期点からベクトル場の向きに逆らう方向
(勾配の逆方向)に進めば「極小値」へ向かう

勾配法による関数最小化

- ランダムな初期解から出発して少しづつ勾配の方向へ解を更新していく → 勾配方向に徐々に進む



最急降下法 = 最もシンプルな勾配法

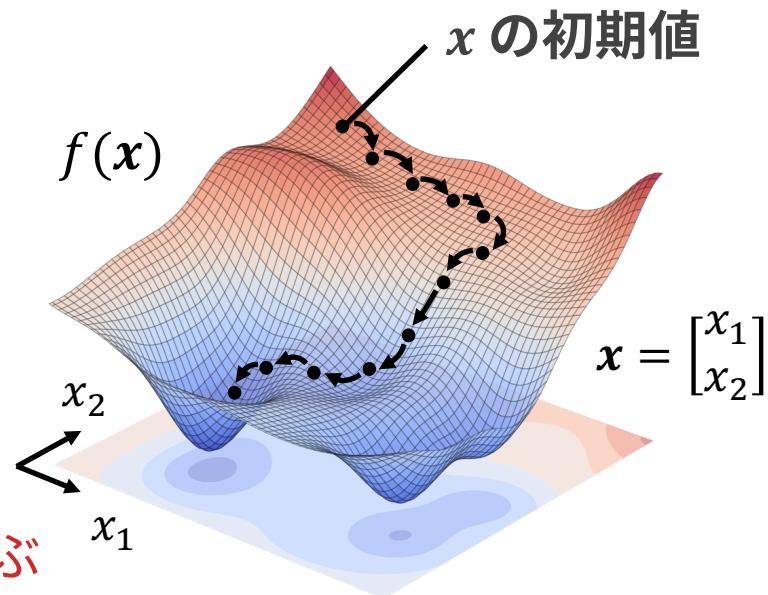
- x をランダム値で初期化
- 関数値 $f(x)$ が小さくなるように x をちょっとだけ動かす

$$\boxed{\left[\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right] \leftarrow \left[\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right] - \alpha \times \left[\begin{array}{c} \partial f / \partial x_1 \\ \partial f / \partial x_2 \\ \vdots \\ \partial f / \partial x_n \end{array} \right]}$$

くりかえし 勾配

α : 1歩の"歩幅"(step size)

機械学習では学習率 (learning rate) と呼ぶ



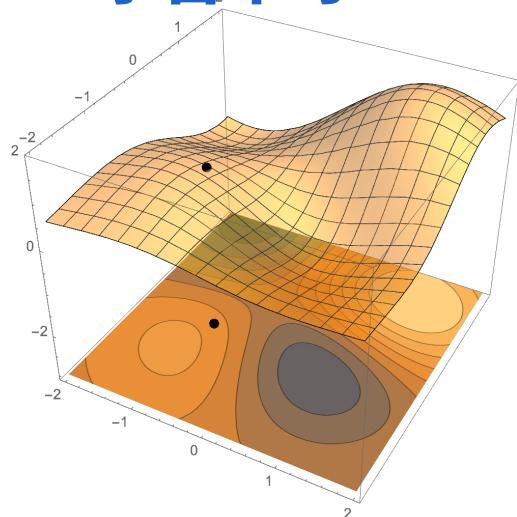
学習率(ステップサイズ)の設定

確実に最寄りの極小値に
向かうが多くの回数が必要

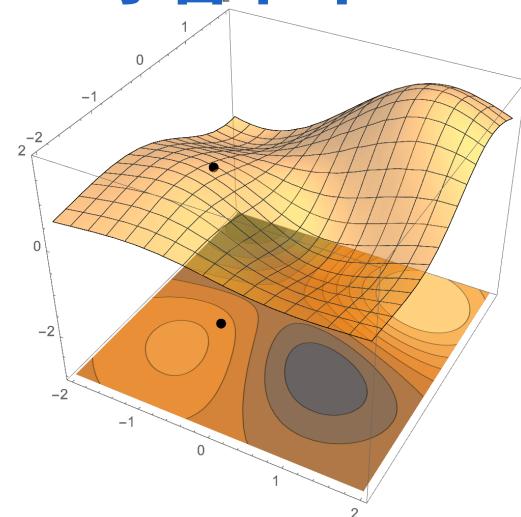
少ない回数で大きくすすめる
が粗すぎて谷を見逃すリスク
+最適化が不安定に



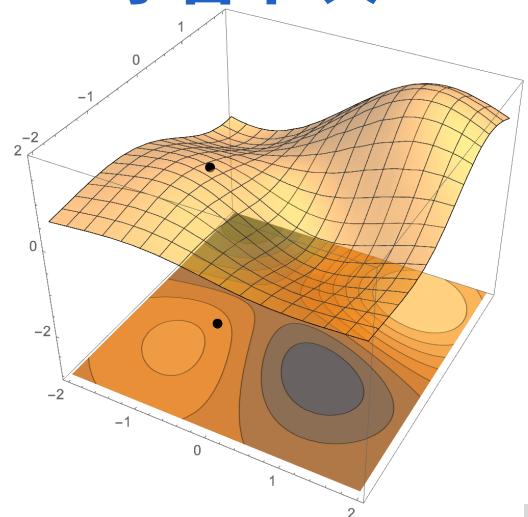
学習率 小



学習率 中

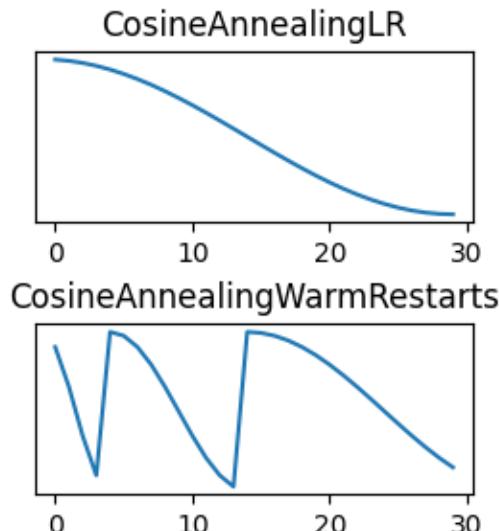
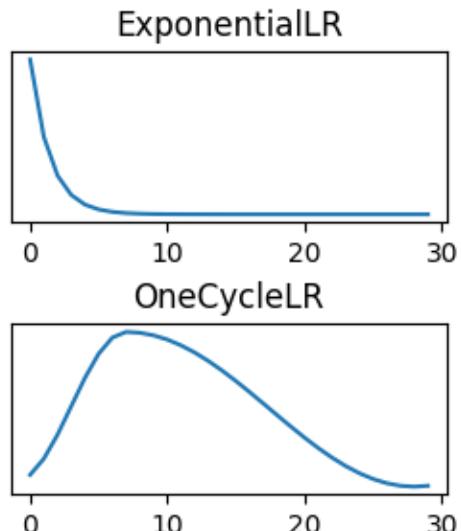
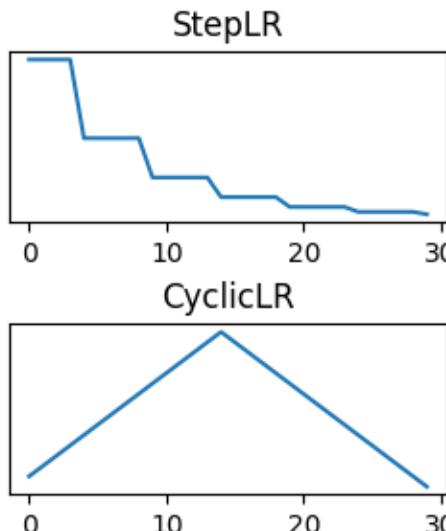


学習率 大



学習率(ステップサイズ)のスケジューリング

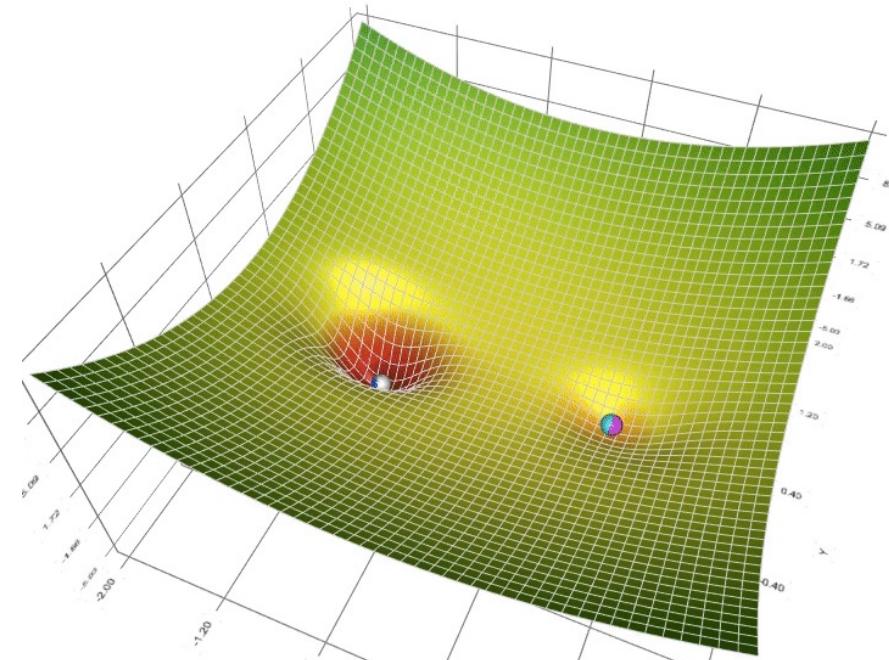
- 基本：大きめでスタートして徐々に小さくする
- 巨大なモデルでは大きめのlrでスタート+ランダム初期化状態で更新が非常に不安定になるので、小さめ→大きめ→小さめも(アニーリング)



勾配法のバリエーション

- 深層学習の学習を加速化・安定化させる
勾配法が多数開発されている

- 最急降下法
- Momentum
- AdaGrad
- RMSProp
- Adam

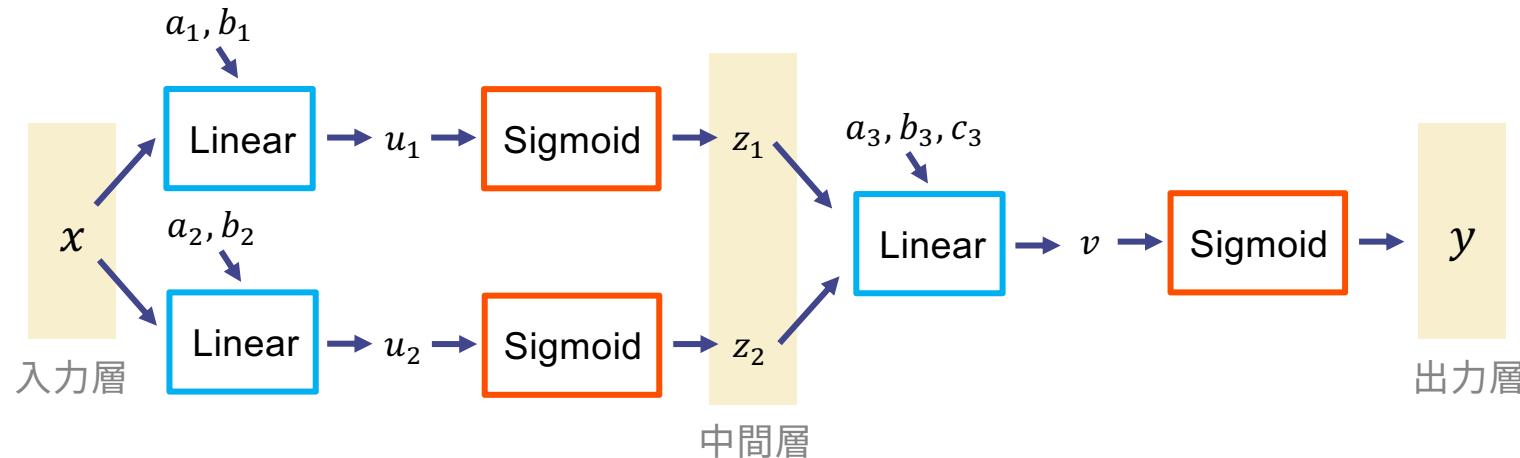


<https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>

残る問題：深層学習モデルの勾配計算

45

- ニューラルネットは単純な演算の結果が次の別の演算の入力となり入れ子構造になった複雑な合成関数



勾配法を適用するにはこのような複雑な合成関数の出力 y の各パラメタについての偏微分係数 $\partial y / \partial a_i, \partial y / \partial b_i, \partial y / \partial c_i, \dots$ の計算が必要

45

3. 自動微分 (アルゴリズム微分)

多変数関数の合成関数の偏微分：連鎖律

■ 連鎖律 (Chain Rule)

・ その1：直列の合成は「かけ算」

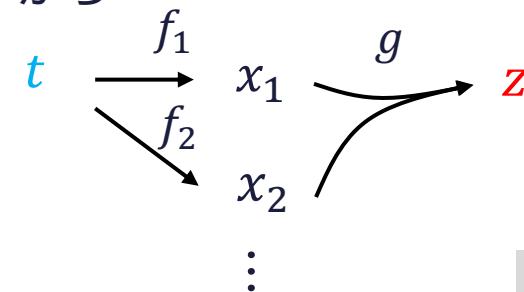
入力 x からの関数変換値 $u = f(x)$ を別の関数の入力 $y = g(u)$ にするとき

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$
$$x \xrightarrow{f} u \xrightarrow{g} y$$

・ その2：並列の合成は「たし算」

入力 t からの複数の関数変換値 $x_1 = f_1(t), x_2 = f_2(t), \dots$ から
合成量 $z = g(x_1, x_2, \dots)$ を得るとき、

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x_1} \frac{\partial x_1}{\partial t} + \frac{\partial z}{\partial x_2} \frac{\partial x_2}{\partial t} + \dots$$



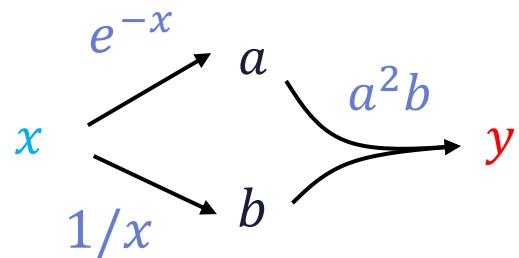
例

- $y = (e^{-x})^2 \left(\frac{1}{x}\right)$ を微分せよ

$$a = e^{-x}$$

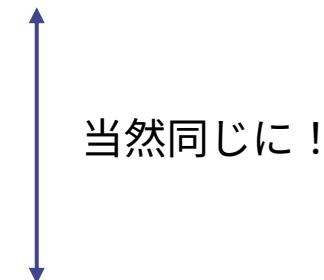
$$b = \frac{1}{x}$$

$$y = a^2 b$$



$$\frac{\partial f}{\partial x} = \left(\frac{e^{-2x}}{x} \right)' = -\frac{e^{-2x}(2x+1)}{x^2}$$

(積の微分 or 商の微分)



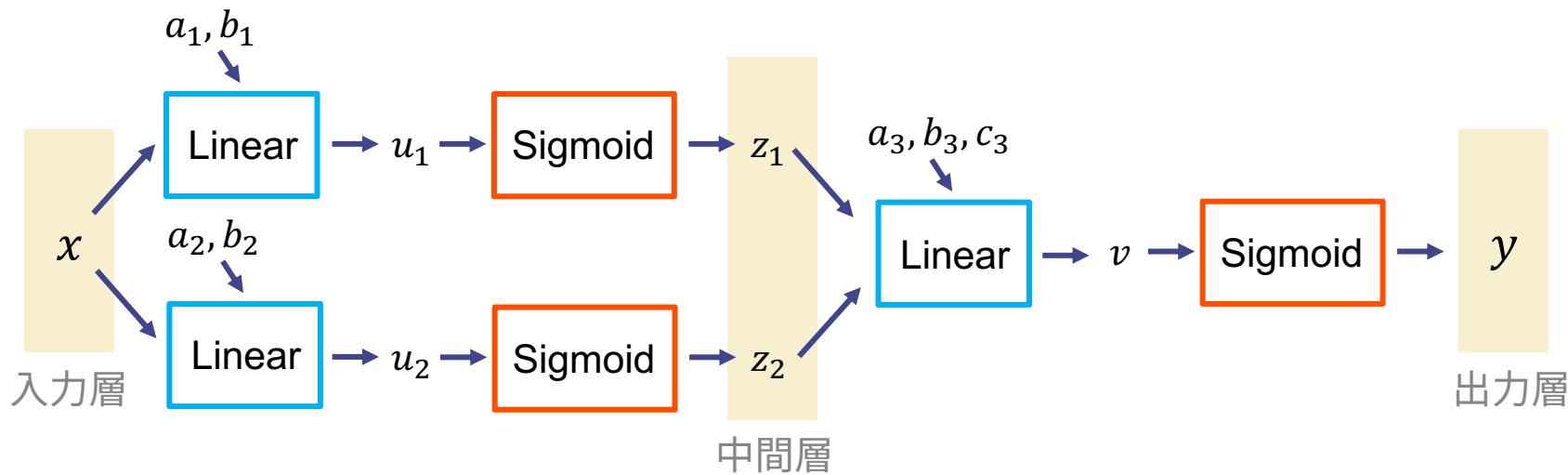
当然同じに！

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial a} \frac{\partial a}{\partial x} + \frac{\partial y}{\partial b} \frac{\partial b}{\partial x} = (2ab)(-e^{-x}) + a^2 \left(-\frac{1}{x^2}\right)$$

(連鎖律)

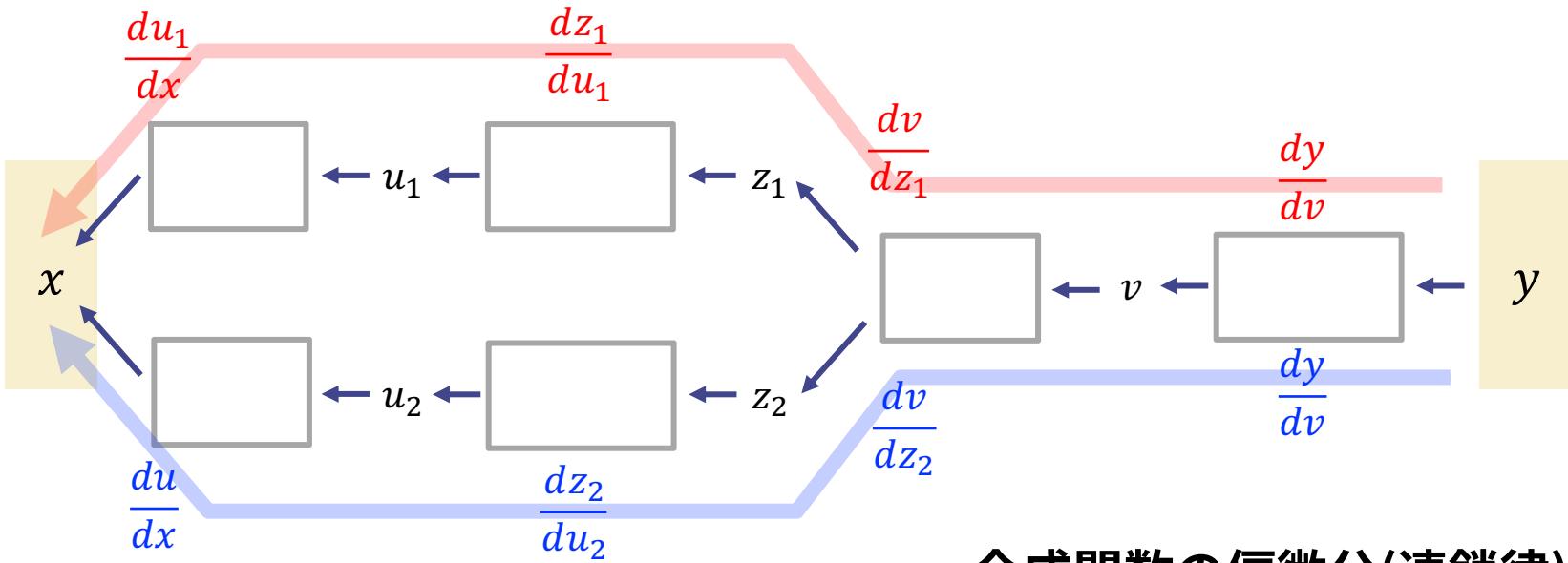
$$= \left(2e^{-x} \cdot \frac{1}{x}\right) (-e^{-x}) + (e^{-x})^2 \left(-\frac{1}{x^2}\right) = -\frac{e^{-2x}(2x+1)}{x^2}$$

例1：3層ニューラルネット



$\partial y / \partial x$ を計算してみよう

例1：3層ニューラルネット



合成関数の偏微分(連鎖律)

直列は掛け算！
並列は足し算！

$$\frac{dy}{dx} = \frac{du_1}{dx} \frac{dz_1}{du_1} \frac{dv}{dz_1} \frac{dy}{dv} + \frac{du_2}{dx} \frac{dz_2}{du_2} \frac{dv}{dz_2} \frac{dy}{dv}$$

例2：微分可能な関数部品の合成

$$y = g_1(g_2(f_3(x), f_2(x)), f_1(x)) = 3\sqrt{x} \left(\log(x) + \frac{1}{x^2} \right)$$

1変数部品

$$\begin{array}{ccc} u & \xrightarrow{f_1} & v \\ & & \\ \end{array}$$
$$\begin{array}{ccc} u & \xrightarrow{f_2} & v \\ & & \\ \end{array}$$
$$\begin{array}{ccc} u & \xrightarrow{f_3} & v \\ & & \\ \end{array}$$

$$v = 3\sqrt{u}$$

$$v = \log u$$

$$v = 1/u$$

$$\frac{dv}{du} = \frac{3}{2\sqrt{u}}$$

$$\frac{dv}{du} = \frac{1}{u}$$

$$\frac{dv}{du} = -\frac{1}{u^2}$$

2変数部品

$$\begin{array}{ccc} a & \xrightarrow{g_1} & v \\ b & \nearrow & \\ \end{array}$$
$$\begin{array}{ccc} a & \xrightarrow{g_2} & v \\ b & \nearrow & \\ \end{array}$$

$$v = a b$$

$$v = a^2 + b$$

$$\frac{\partial v}{\partial a} = b$$

$$\frac{\partial v}{\partial a} = 2a$$

$$\frac{\partial v}{\partial b} = a$$

$$\frac{\partial v}{\partial b} = 1$$

例2：微分可能な関数部品の合成

$$y = g_1(g_2(f_3(x), f_2(x)), f_1(x)) = 3\sqrt{x} \left(\log(x) + \frac{1}{x^2} \right)$$

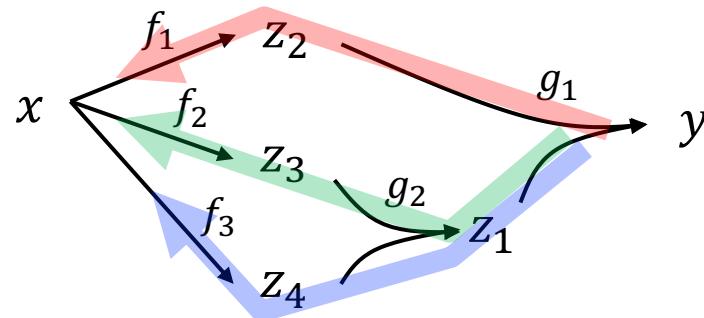
$$y = g_1(z_1, z_2) = z_1 z_2$$

$$z_1 = g_2(z_4, z_3) = z_3 + z_4^2$$

$$z_2 = f_1(x) = 3\sqrt{x}$$

$$z_3 = f_2(x) = \log x$$

$$z_4 = f_3(x) = 1/x$$



$$\frac{dy}{dx} = \frac{dz_4}{dx} \frac{dz_1}{dz_4} \frac{dy}{dz_1} + \frac{dz_3}{dx} \frac{dz_1}{dz_3} \frac{dy}{dz_1} + \frac{dz_2}{dx} \frac{dy}{dz_2} = 3\sqrt{x} \left(\frac{1}{x} - \frac{2}{x^3} \right) + \frac{3 \left(\log(x) + \frac{1}{x^2} \right)}{2\sqrt{x}}$$

例2：x=1.2の偏微分係数を「自動微分」で計算

```
from torch import tensor, sqrt, log
```

```
x = tensor(1.2, requires_grad=True)
z2 = 3*sqrt(x)
z3 = log(x)
z4 = 1/x
z1 = z4**2 + z3
y = z1 * z2
```

```
y.backward() ←
print(x.grad)
```

tensor(0.14)

x=1.2でのdy/dxの値
(勾配法で使える)

$$y = 3\sqrt{x} \left(\log(x) + \frac{1}{x^2} \right)$$



$$z_2 = 3\sqrt{x}$$

$$z_3 = \log x$$

$$z_4 = 1/x$$

$$z_1 = z_4^2 + z_3$$

$$y = z_1 z_2$$

自動微分(1行)！！

以降の目標は下記2点の理解

- この裏で何が起こっている？
- なぜbackwardという名前？

自動微分 (アルゴリズム微分)

- 自動微分 (アルゴリズム微分) : 連鎖律の適用をアルゴリズム的に
行って、与えられた入力点での関数の偏微分係数を求める計算
- 勾配法に必要なのはある点での偏微分係数
- 先ほどの例では偏導関数の関数形

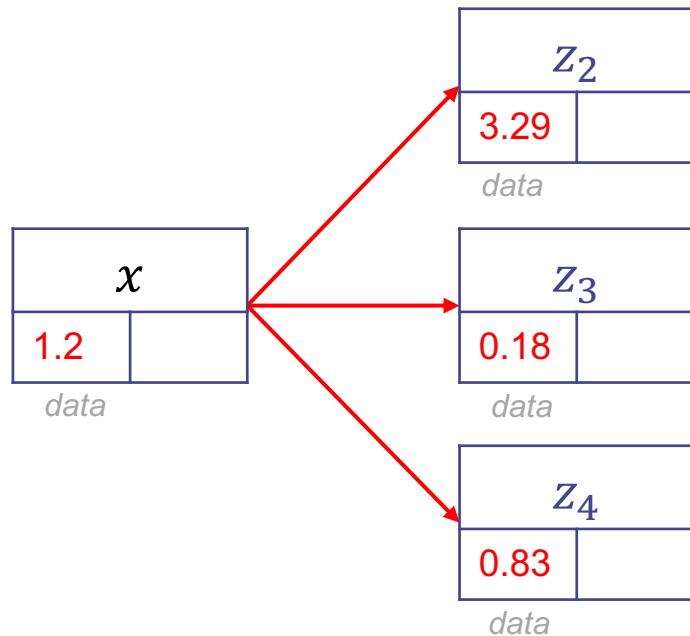
$$\frac{dy}{dx} = 3\sqrt{x} \left(\frac{1}{x} - \frac{2}{x^3} \right) + \frac{3 \left(\log(x) + \frac{1}{x^2} \right)}{2\sqrt{x}}$$

までは必要なく、例えば $x = 1.2$ での $\frac{dy}{dx}$ の値 0.14 がわかれば十分

- 深層学習モデルは単純な関数部品の複雑で巨大な合成関数

順計算：偏微分を計算したい値での出力値を計算する

Forward

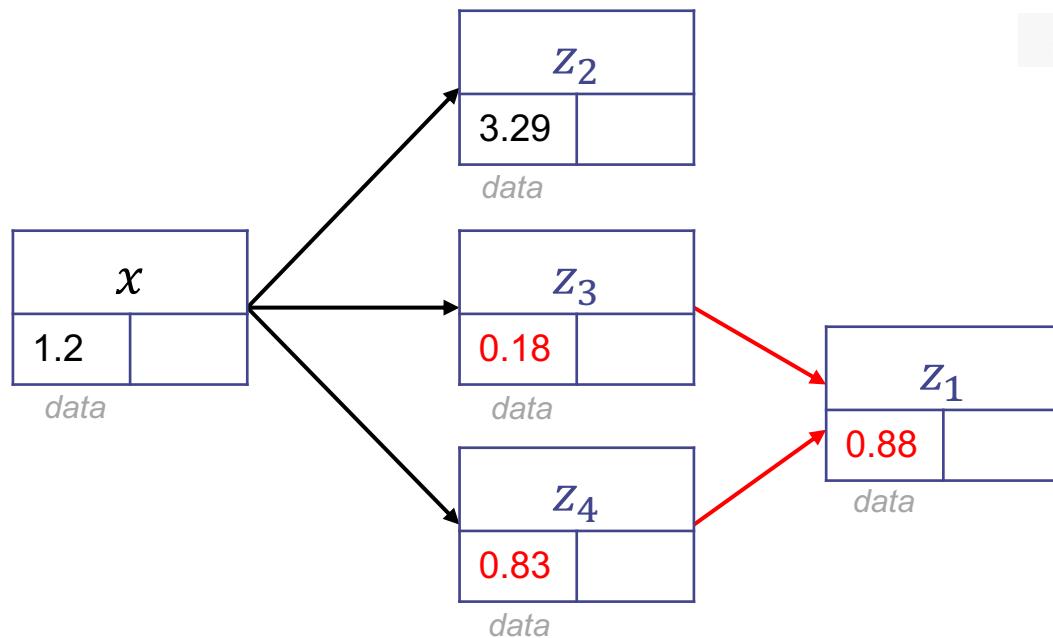


```
x = tensor(1.2, requires_grad=True)
z2 = 3*sqrt(x)
z3 = log(x)
z4 = 1/x
```

$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

順計算：偏微分を計算したい値での出力値を計算する

Forward

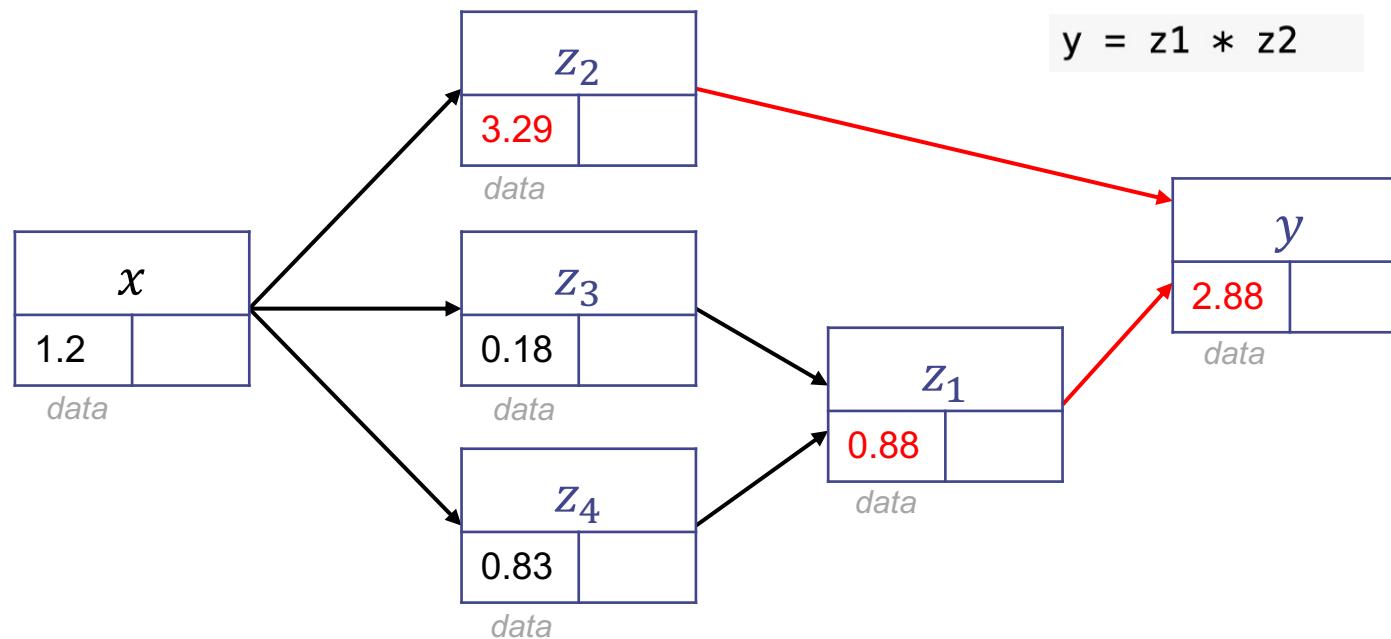


$$z_1 = z_4 * 2 + z_3$$

$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

順計算：偏微分を計算したい値での出力値を計算する

Forward



$$y = z_1 z_2$$

$$z_1 = z_4^2 + z_3$$

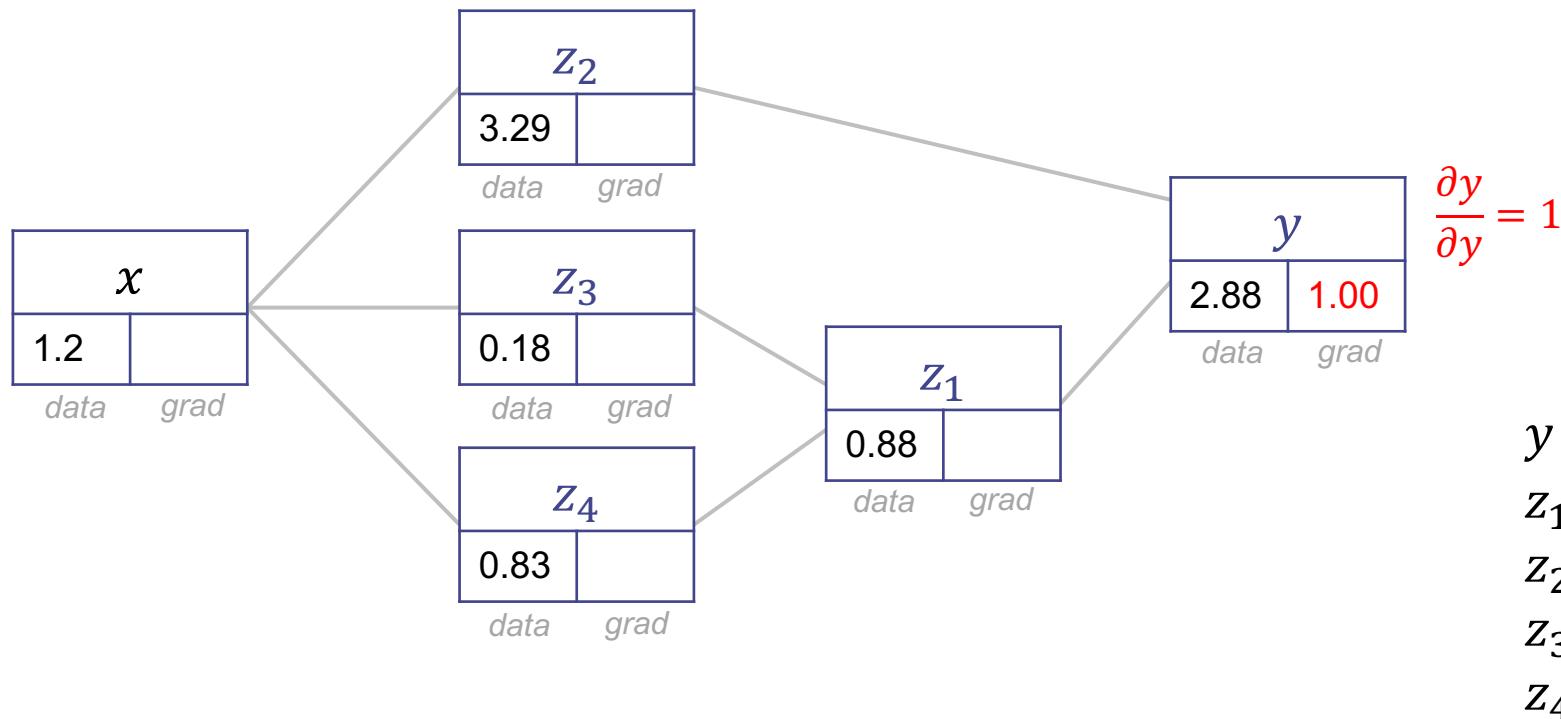
$$z_2 = 3\sqrt{x}$$

$$z_3 = \log x$$

$$z_4 = 1/x$$

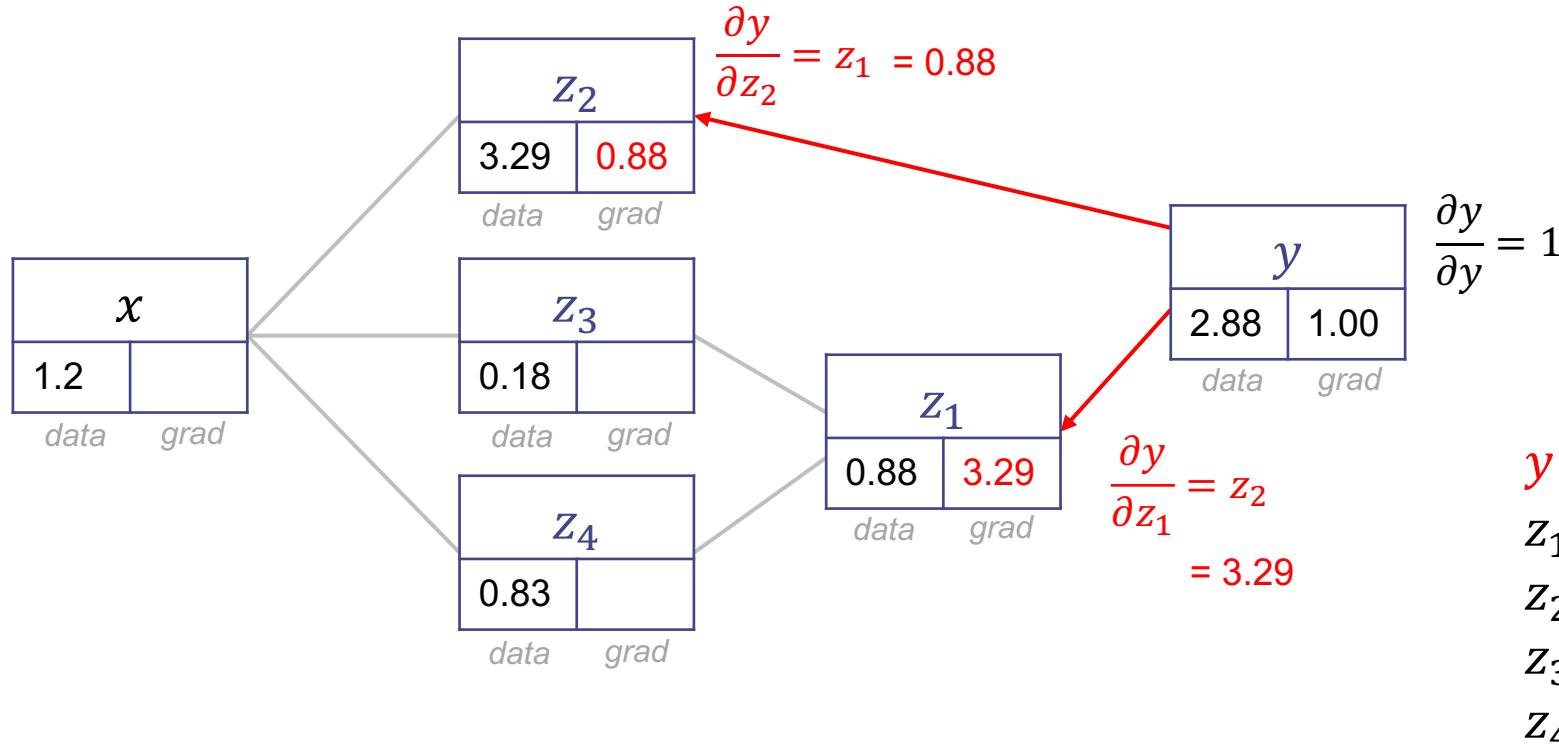
逆計算：出力の各変数での偏微分係数を計算

Backward



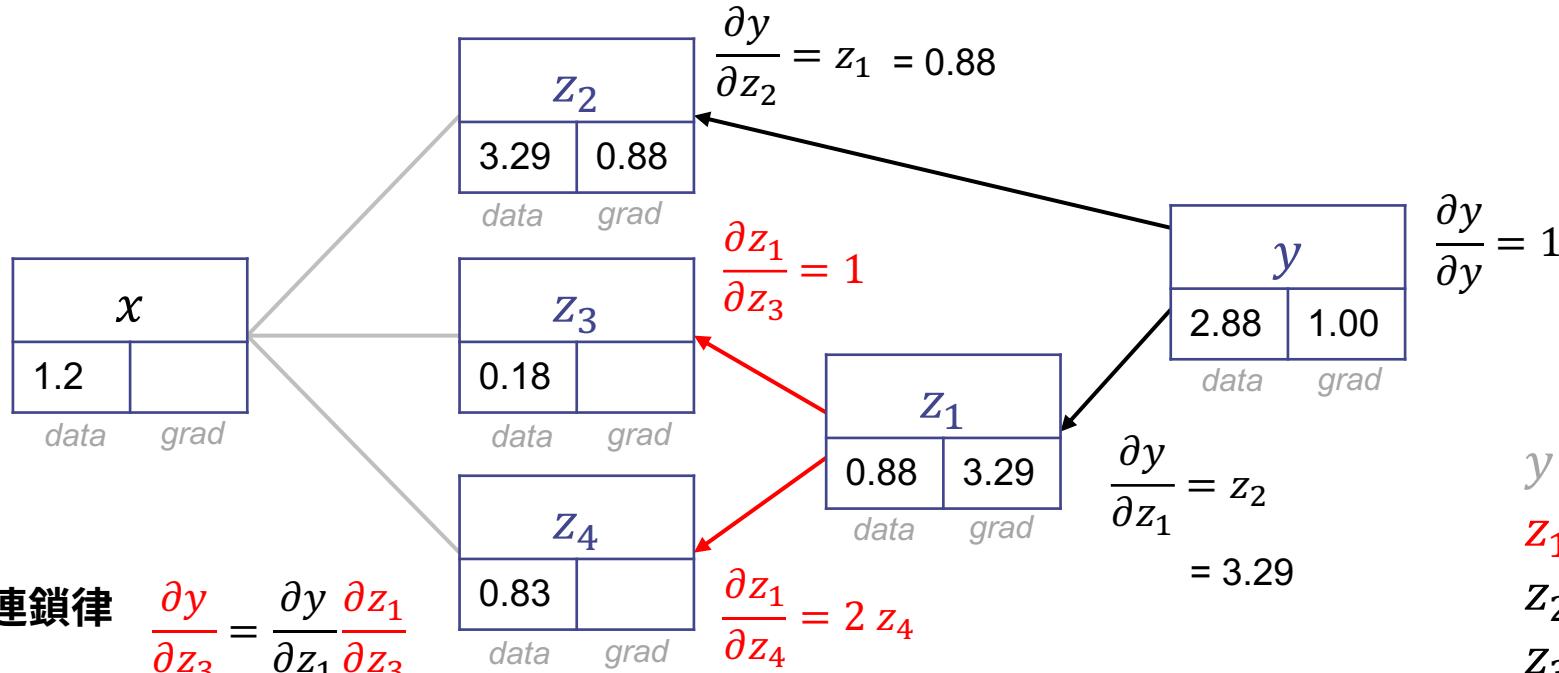
逆計算：出力の各変数での偏微分係数を計算

Backward



逆計算：出力の各変数での偏微分係数を計算

Backward



連鎖律

$$\frac{\partial y}{\partial z_3} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3}$$

$$\frac{\partial y}{\partial z_4} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4}$$

$$y = z_1 z_2$$

$$z_1 = z_4^2 + z_3$$

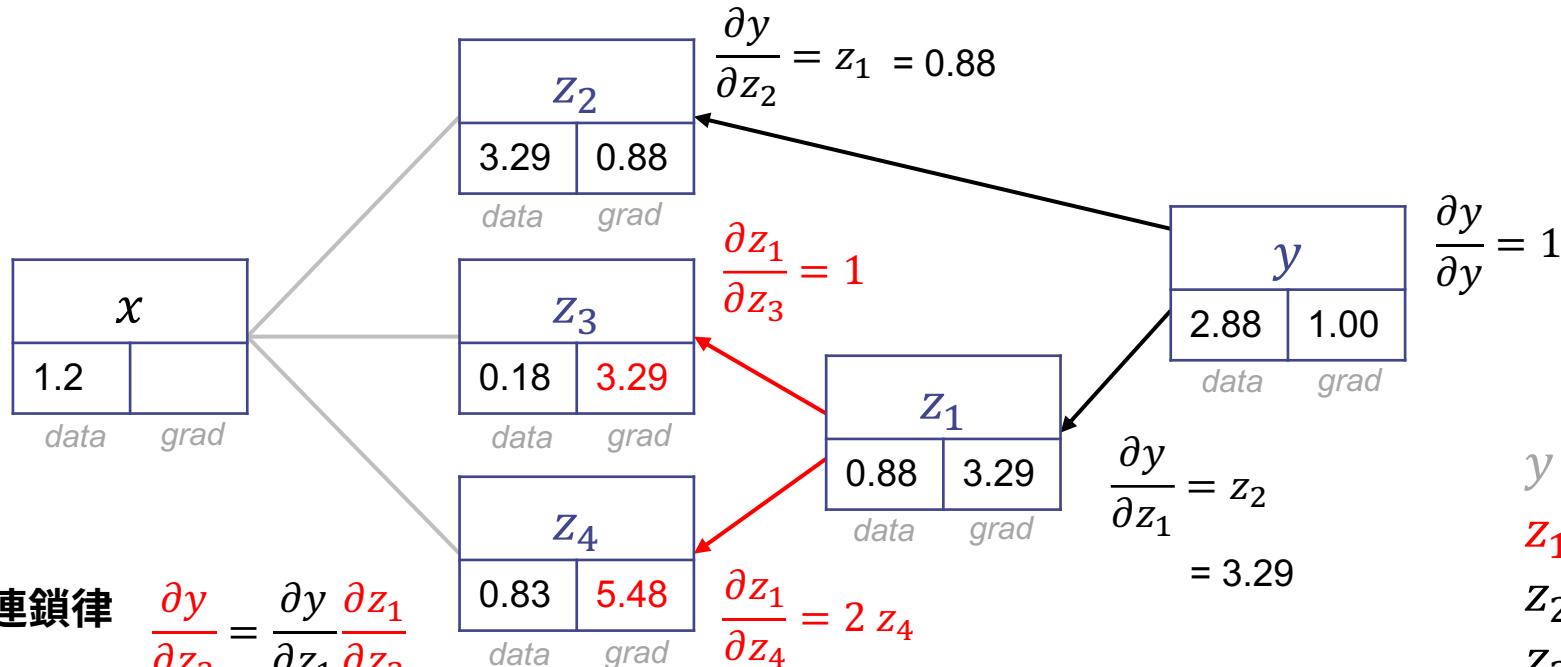
$$z_2 = 3\sqrt{x}$$

$$z_3 = \log x$$

$$z_4 = 1/x$$

逆計算：出力の各変数での偏微分係数を計算

Backward



連鎖律

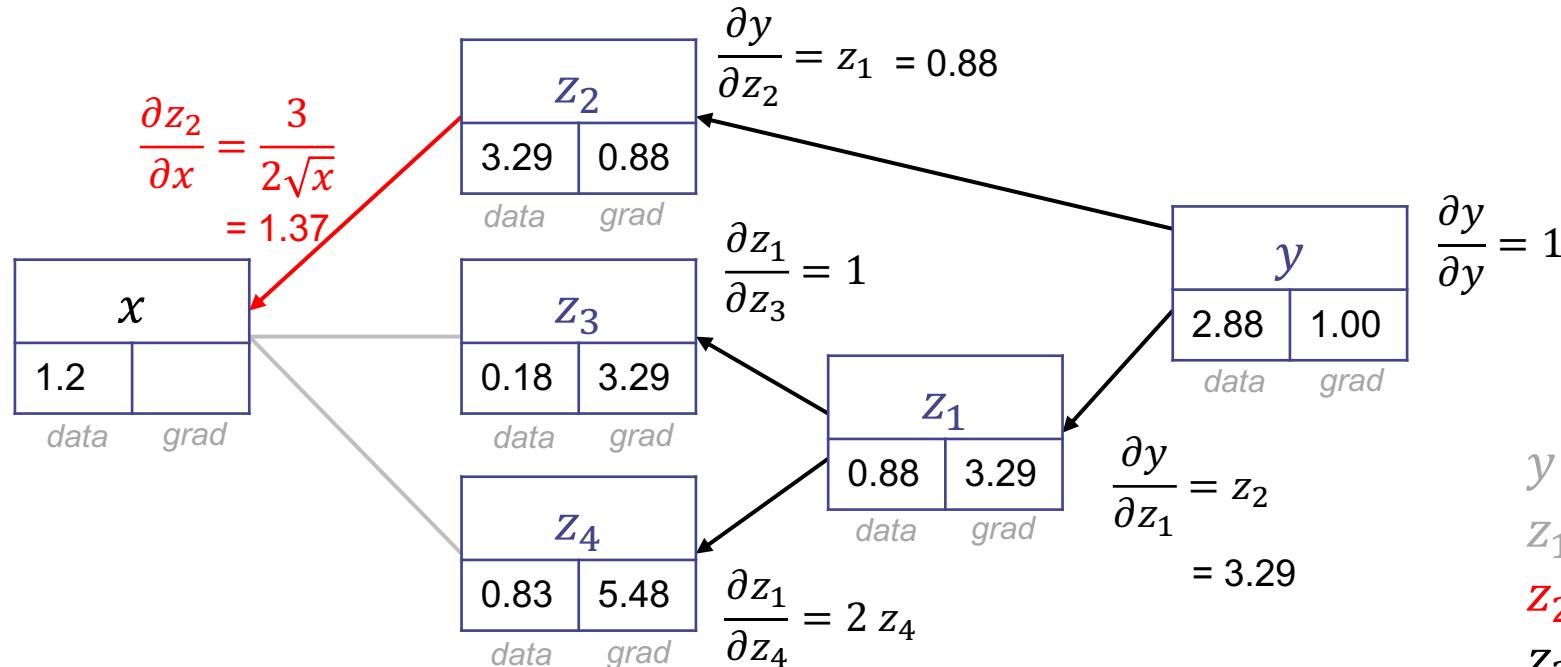
$$\frac{\partial y}{\partial z_3} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3}$$

$$\frac{\partial y}{\partial z_4} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4}$$

$$\begin{aligned}
 y &= z_1 z_2 \\
 z_1 &= z_4^2 + z_3 \\
 z_2 &= 3\sqrt{x} \\
 z_3 &= \log x \\
 z_4 &= 1/x
 \end{aligned}$$

逆計算：出力の各変数での偏微分係数を計算

Backward



$$y = z_1 z_2$$

$$z_1 = z_4^2 + z_3$$

$$z_2 = 3\sqrt{x}$$

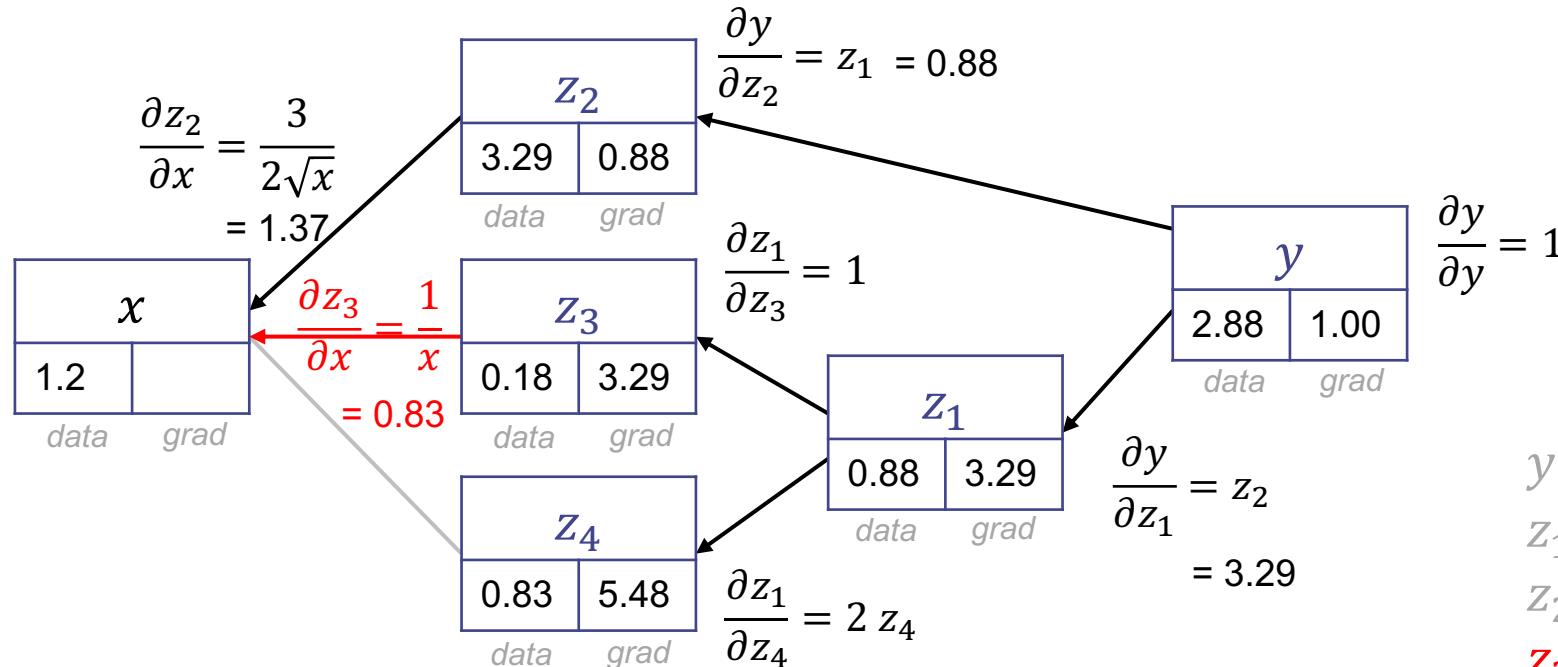
$$z_3 = \log x$$

$$z_4 = 1/x$$

$$\text{連鎖律 } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$$

逆計算：出力の各変数での偏微分係数を計算

Backward

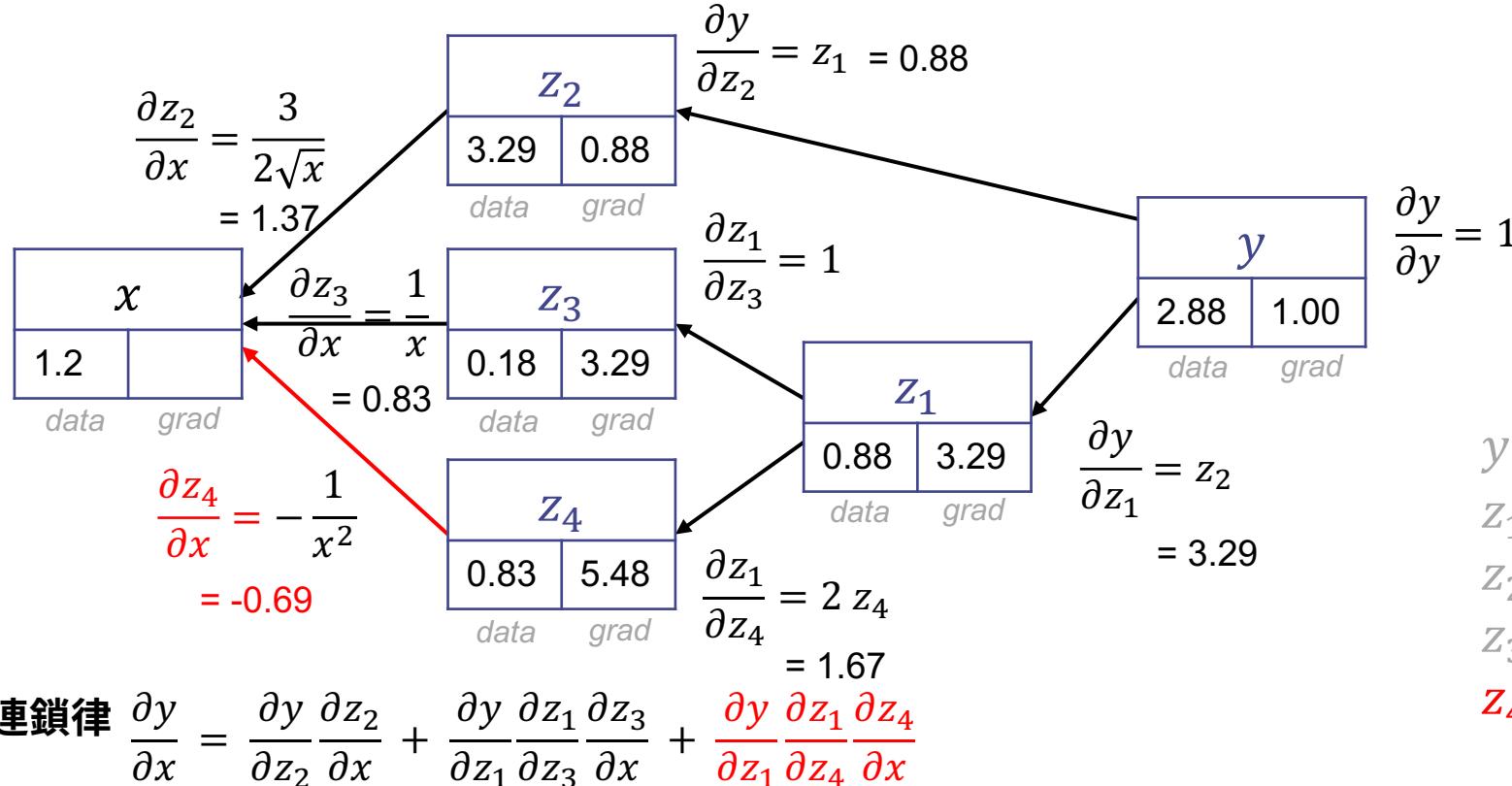


$$\text{連鎖律 } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$$

$$\begin{aligned}
 y &= z_1 z_2 \\
 z_1 &= z_4^2 + z_3 \\
 z_2 &= 3\sqrt{x} \\
 z_3 &= \log x \\
 z_4 &= 1/x
 \end{aligned}$$

逆計算：出力の各変数での偏微分係数を計算

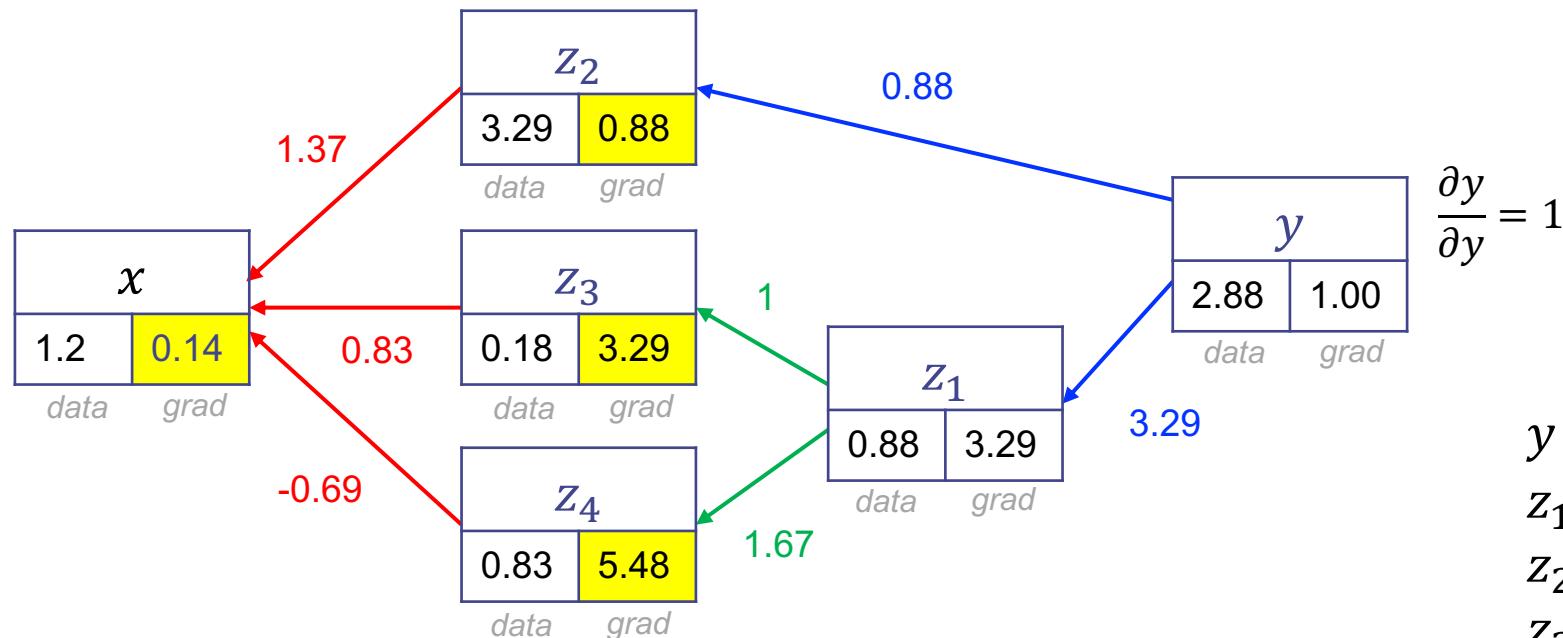
Backward



連鎖律 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$

逆計算：出力の各変数での偏微分係数を計算

Backward

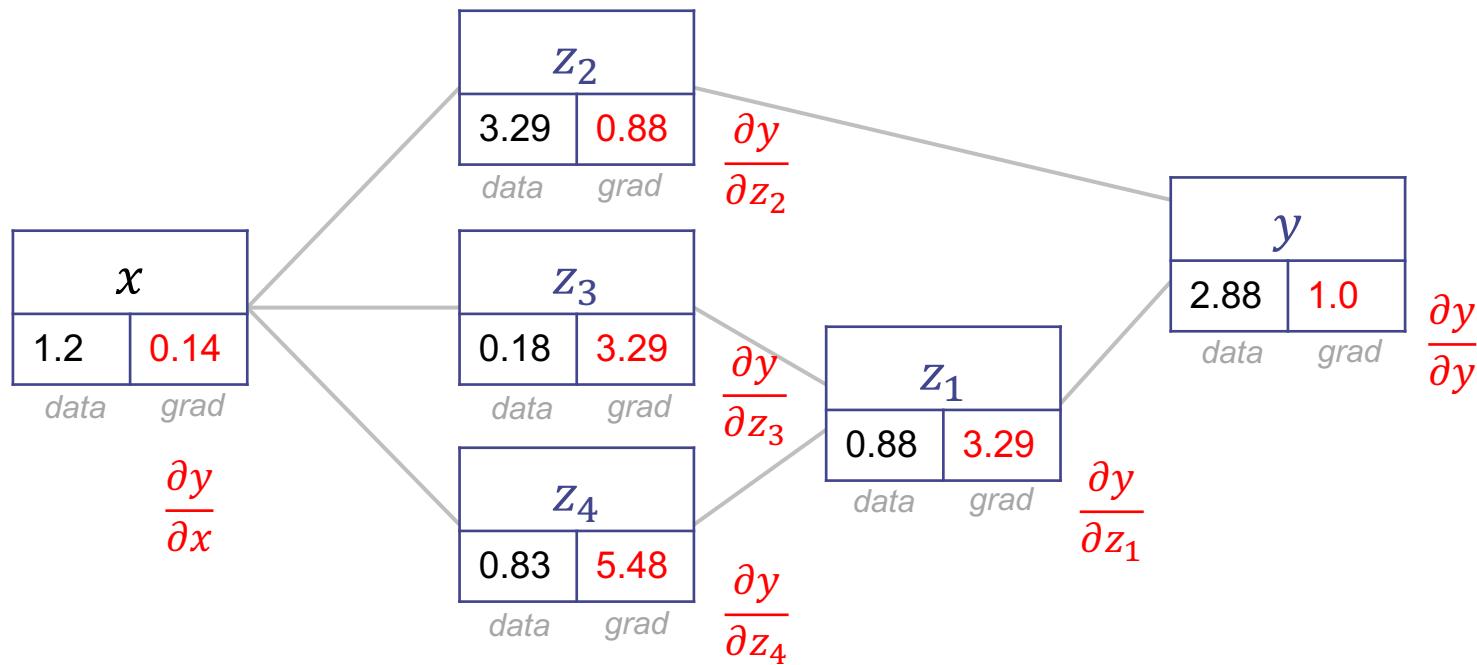


$$\frac{\partial y}{\partial y} = 1$$

$$\begin{aligned}
 y &= z_1 z_2 \\
 z_1 &= z_4^2 + z_3 \\
 z_2 &= 3\sqrt{x} \\
 z_3 &= \log x \\
 z_4 &= 1/x
 \end{aligned}$$

連鎖律 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$

自動微分 (アルゴリズム微分)



Forward→Backwardで y の各変数に対する偏微分係数が全部求まる！

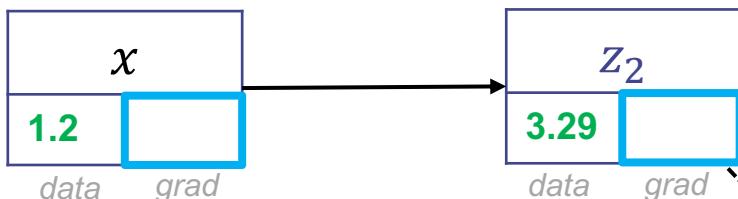
もう少し中身を詳解：自動微分の要素計算

$$z_2 = 3\sqrt{x}$$

`z2.data`
 $\leftarrow 3 * \text{sqrt}(x.data)$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \times \frac{\partial z_2}{\partial x}$$

Forward



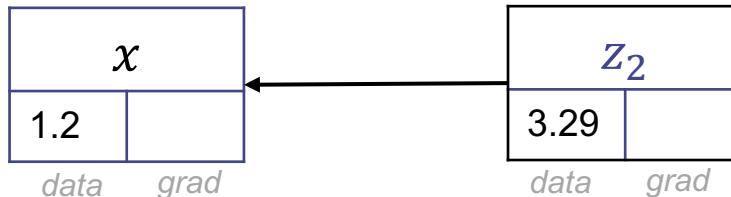
backward用の処理を
変数に紐づけておく

`x.grad`
 $\leftarrow 3/(2 * \text{sqrt}(x.data)) * z2.grad$

`x.grad`
 $\leftarrow 3/(2 * \text{sqrt}(x.data))$
 $* z2.grad$

関数部品ごとに
予め分かる

Backward



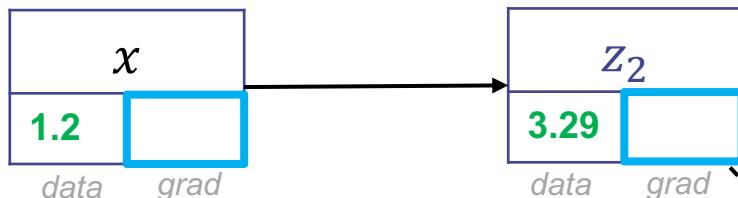
もう少し中身を詳解：自動微分の要素計算

$$z_2 = 3\sqrt{x}$$

`z2.data`
 $\leftarrow 3 * \text{sqrt}(x.data)$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \times \frac{\partial z_2}{\partial x}$$

Forward



backward用の処理を
変数に紐づけておく

$$\frac{\partial z_2}{\partial x} = \frac{3}{2\sqrt{x}}$$

▶ `x.grad`
 $\leftarrow 3/(2*\text{sqrt}(x.data)) * z2.grad$

`x.grad`
 $\leftarrow 3/(2*\text{sqrt}(x.data))$
 $* z2.grad$

関数部品ごとに
予め分かる

Backward

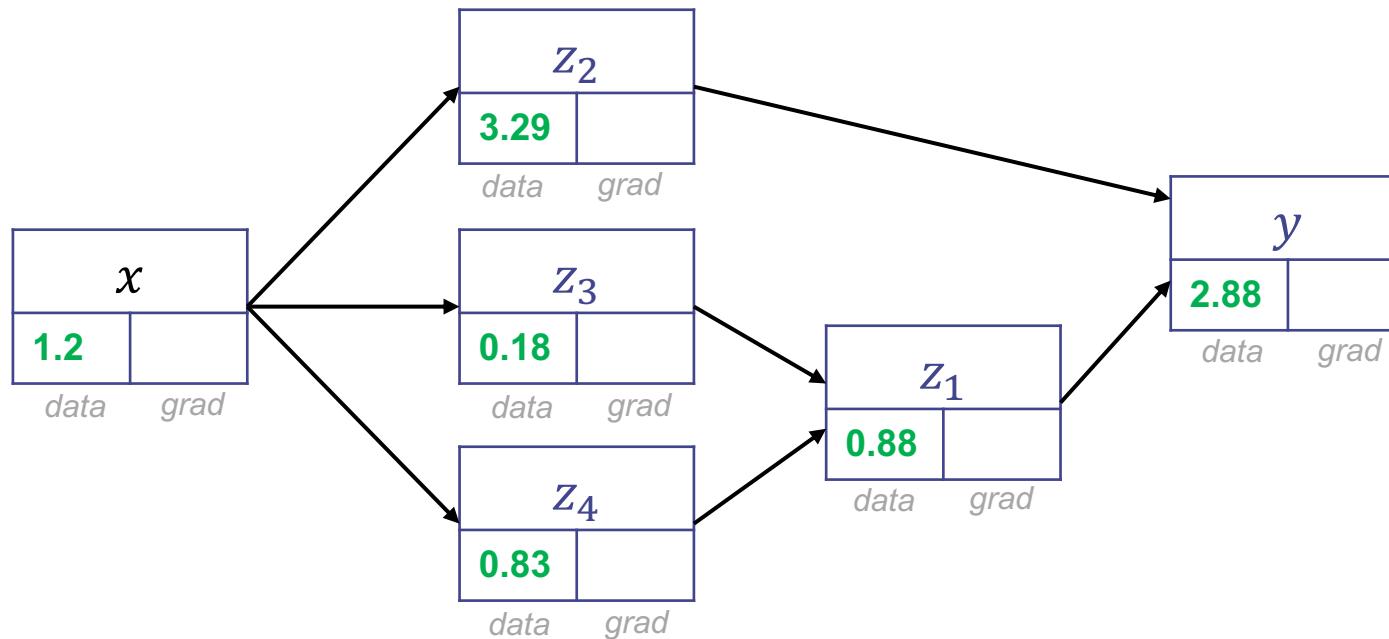


ここまで計算されて
きたとする

$$\frac{\partial y}{\partial z_2} = 0.88$$

処理実行

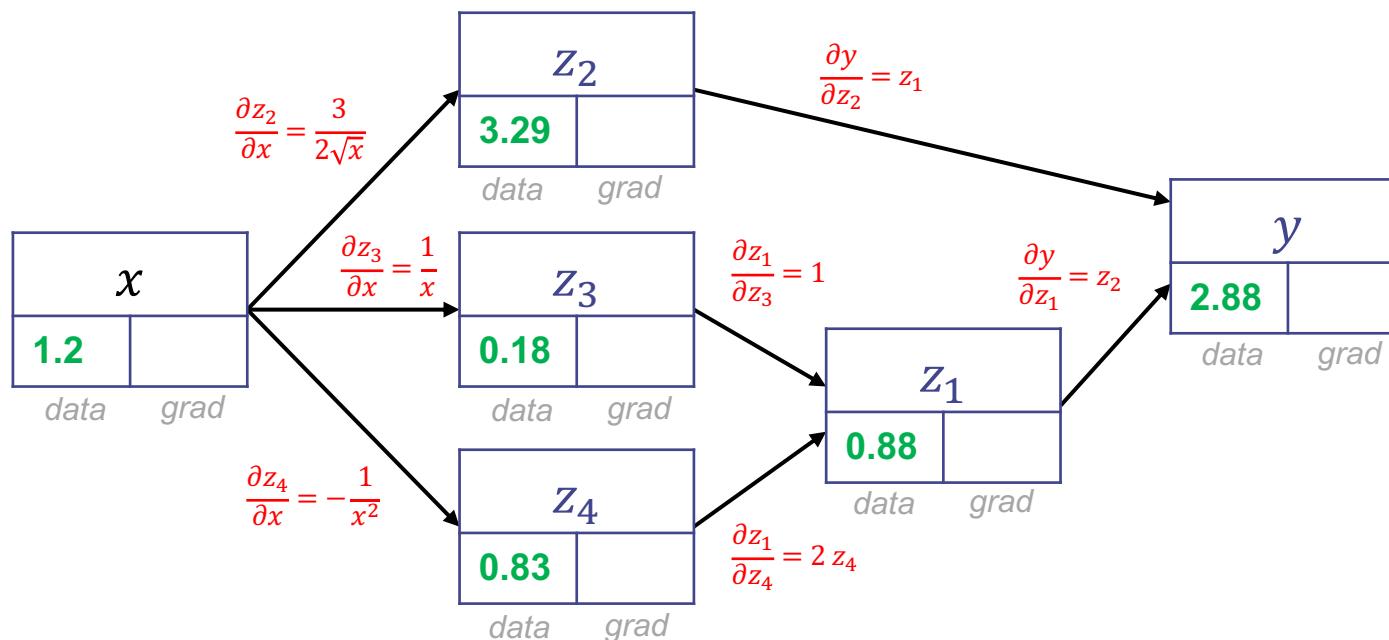
1. Forward計算 (関数值を計算 & 計算グラフ作成)



$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

計算グラフ：どの変数でどの変数を作ったかの情報を取っておく
(情報科学ではネットワーク状の構造情報を「グラフ」と呼ぶ)

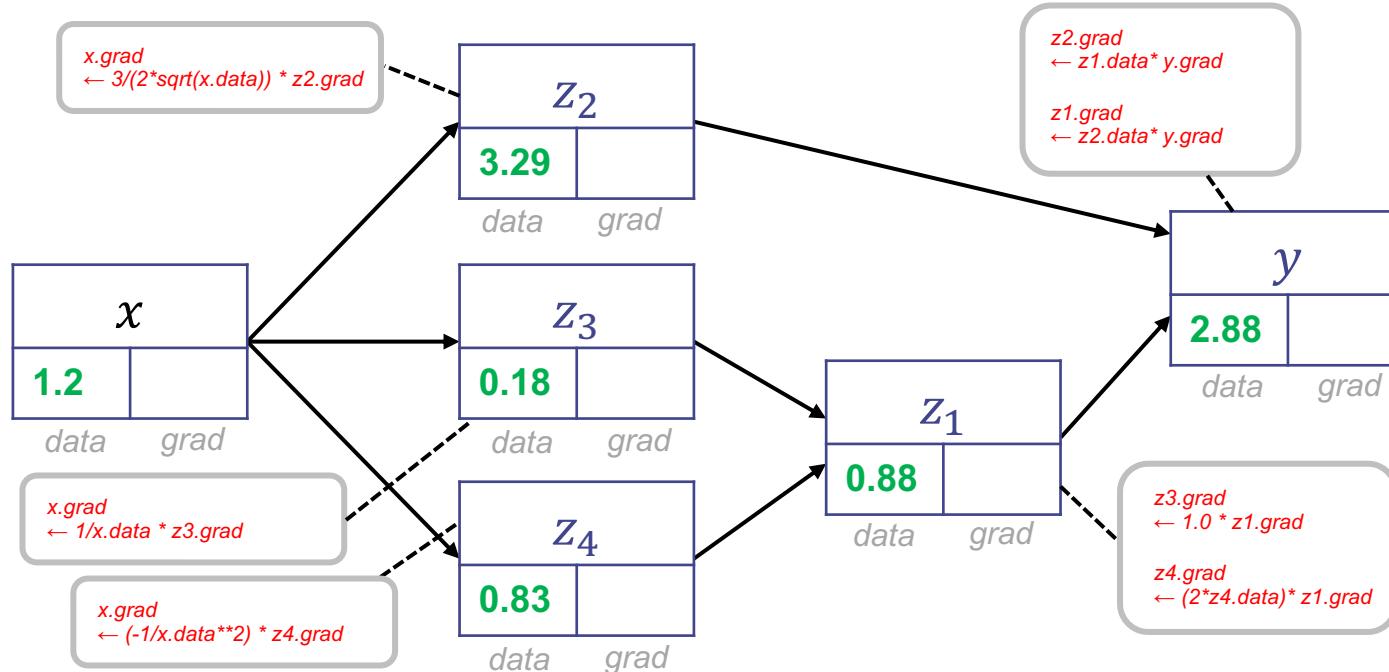
1. Forward計算 (関数值を計算 & 計算グラフ作成)



$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

各合成変数の作成時に Backward用の導関数計算も定義して、
合成元の変数に紐づけておく

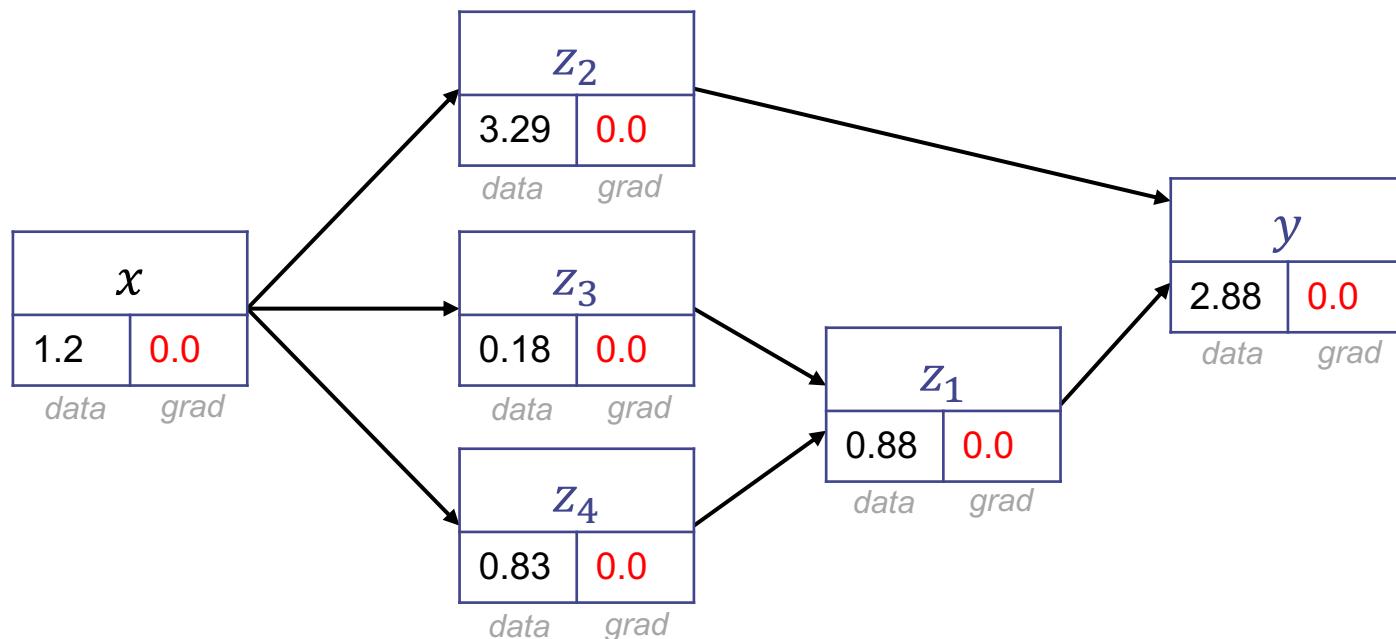
1. Forward計算 (関数值を計算 & 計算グラフ作成)



各合成変数の作成時に Backward用の導関数計算も定義して、
合成元の変数に紐づけておく

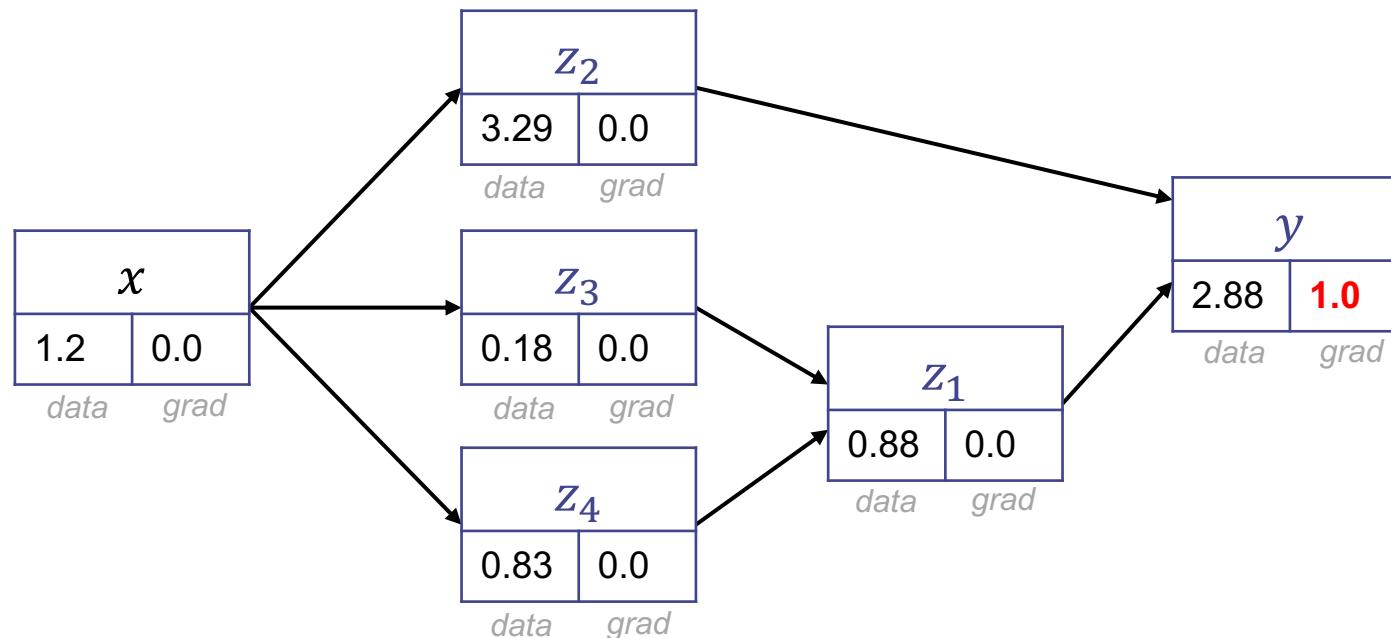
$$\begin{aligned}y &= Z_1 Z_2 \\Z_1 &= Z_4^2 + Z_3 \\Z_2 &= 3\sqrt{x} \\Z_3 &= \log x \\Z_4 &= 1/x\end{aligned}$$

2. Backward計算 (gradを0で初期化)



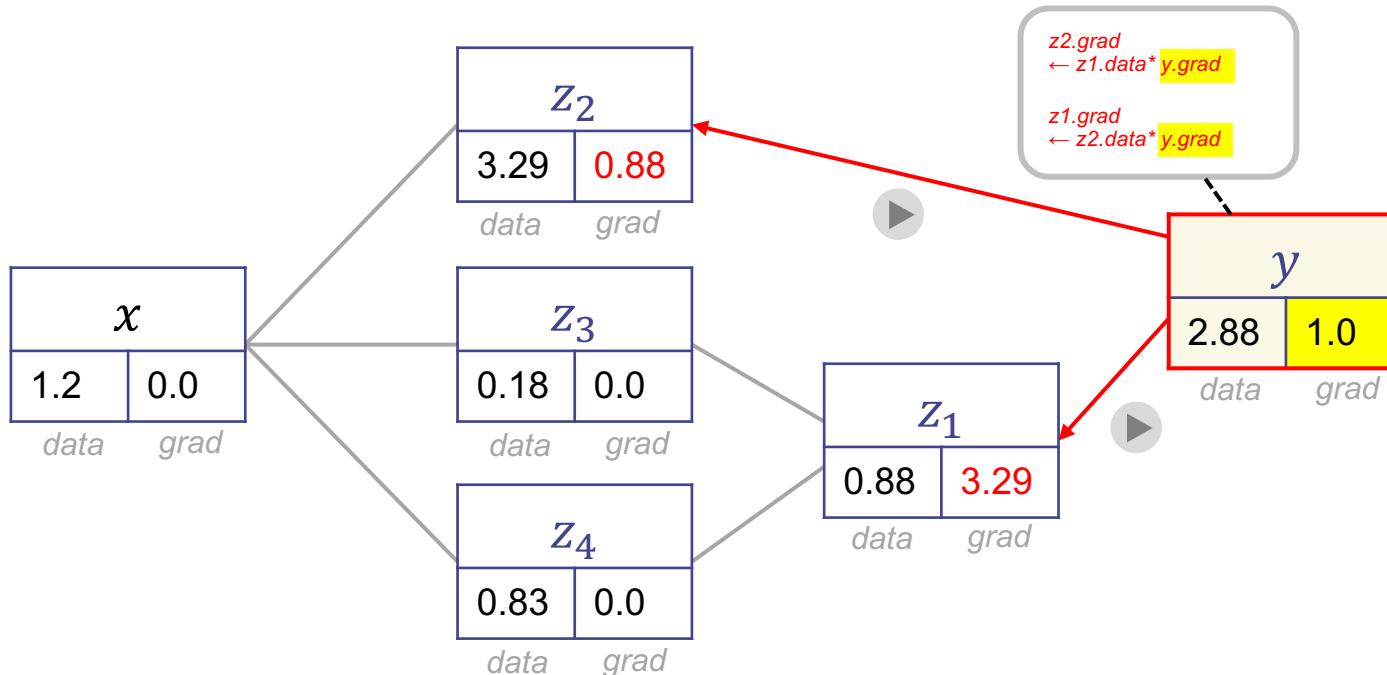
$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

3. Backward計算(開始点のgradを1にセット)



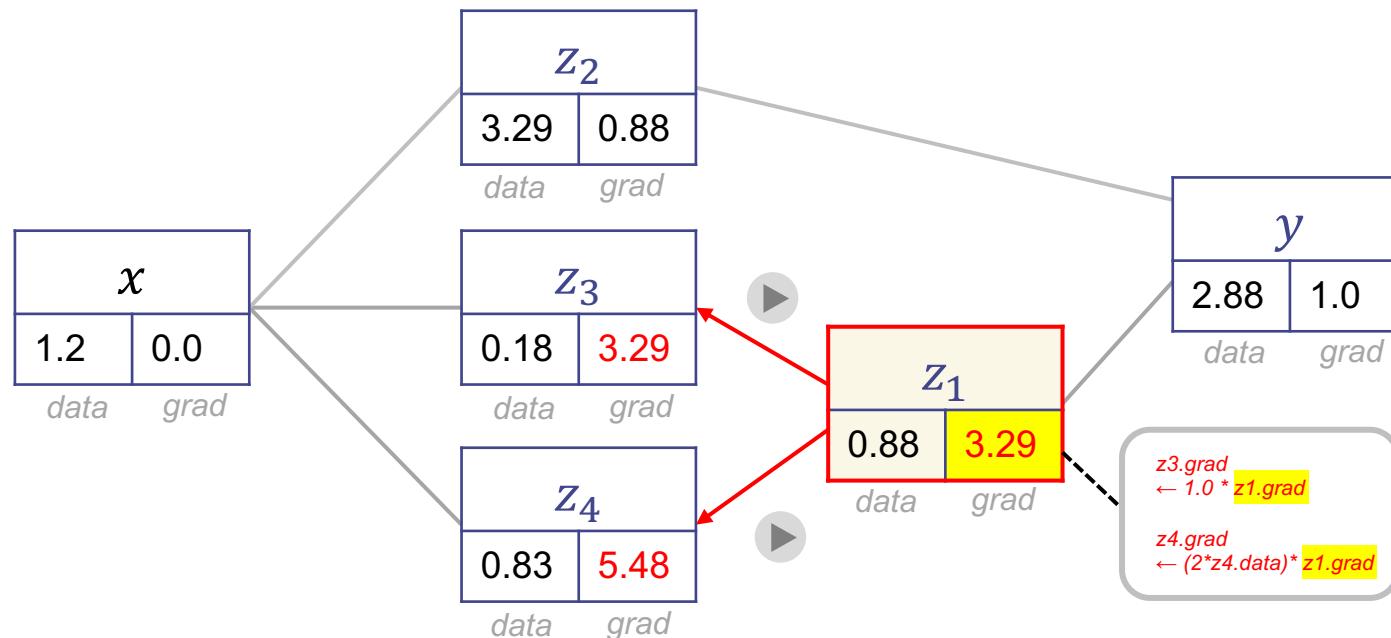
$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

4. Backward計算 (yを処理しgradに累積)



$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

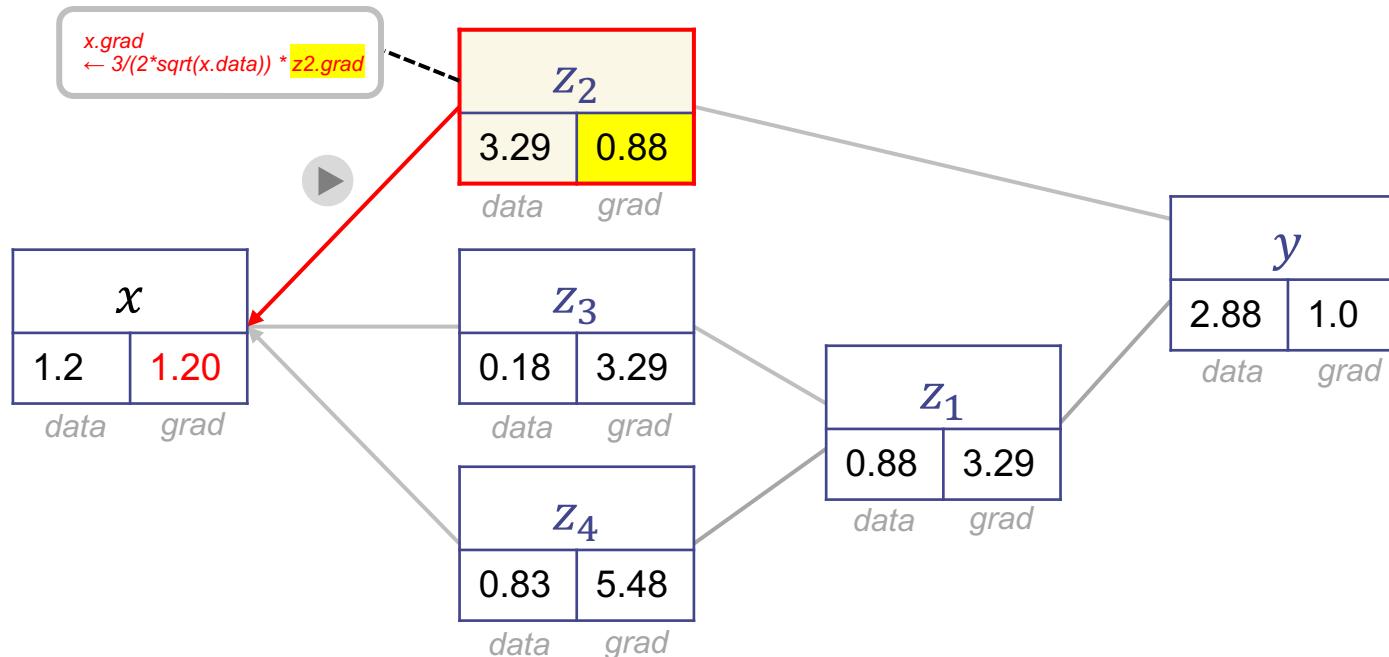
5. Backward計算 (z1を処理しgradに累積)



$$\frac{\partial y}{\partial z_3} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \quad \frac{\partial y}{\partial z_4} = \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4}$$

$$\begin{aligned}
 y &= z_1 z_2 \\
 z_1 &= z_4^2 + z_3 \\
 z_2 &= 3\sqrt{x} \\
 z_3 &= \log x \\
 z_4 &= 1/x
 \end{aligned}$$

6. Backward計算 (z2を処理しgradに累積)

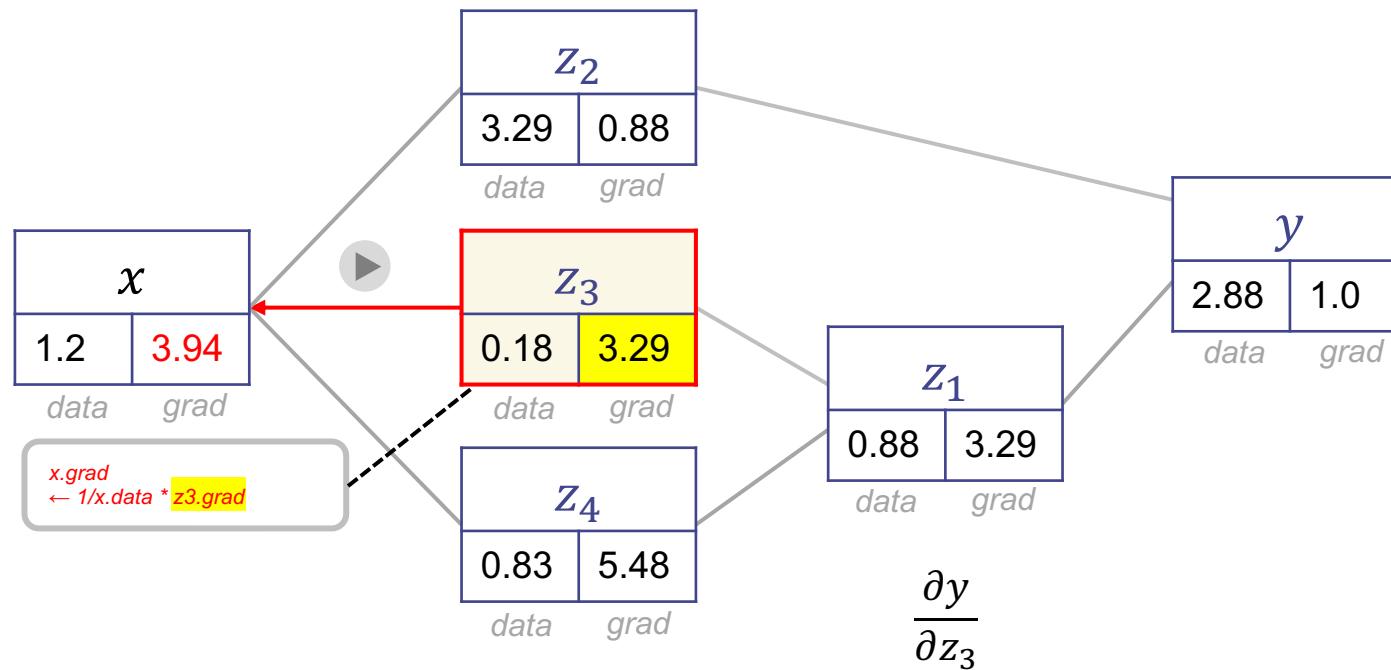


算出した1.20を
元々のgrad=0.0に
累積して加算

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$$

$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

6. Backward計算 (z3を処理しgradに累積)

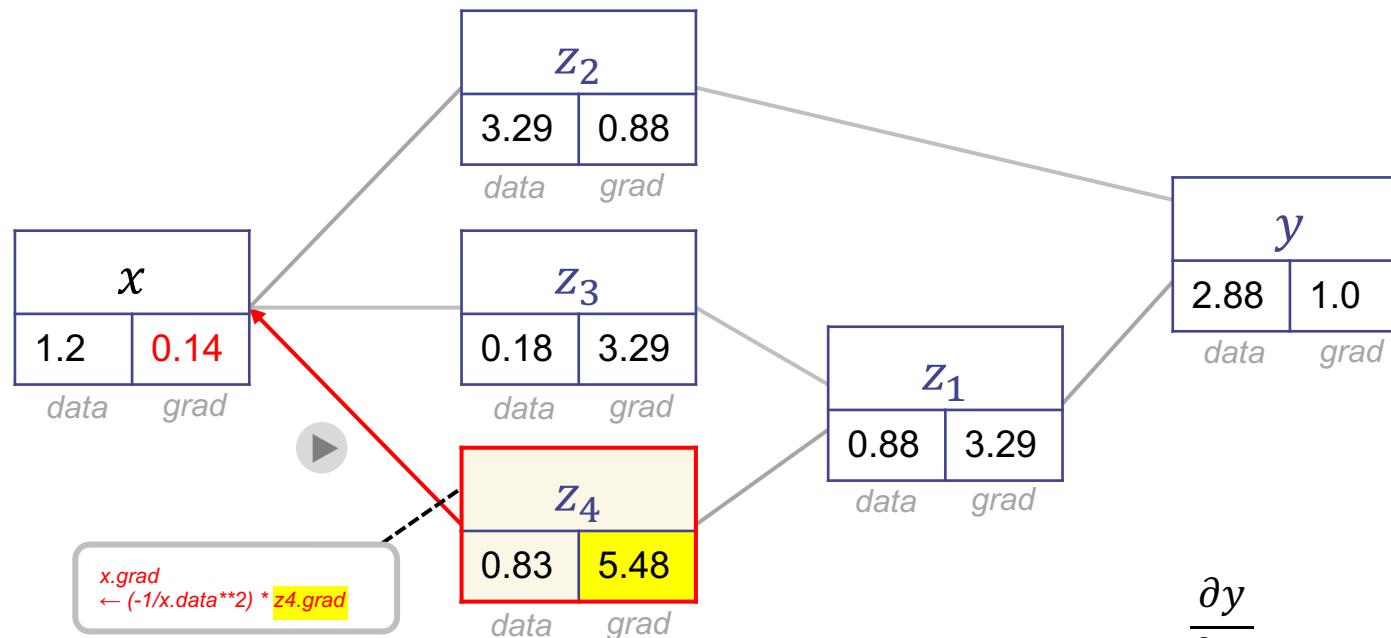


算出した2.74を
元々のgrad=1.20に
累積して加算

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$$

$$\begin{aligned}
 y &= z_1 z_2 \\
 z_1 &= z_4^2 + z_3 \\
 z_2 &= 3\sqrt{x} \\
 z_3 &= \log x \\
 z_4 &= 1/x
 \end{aligned}$$

6. Backward計算 (z4を処理しgradに累積)



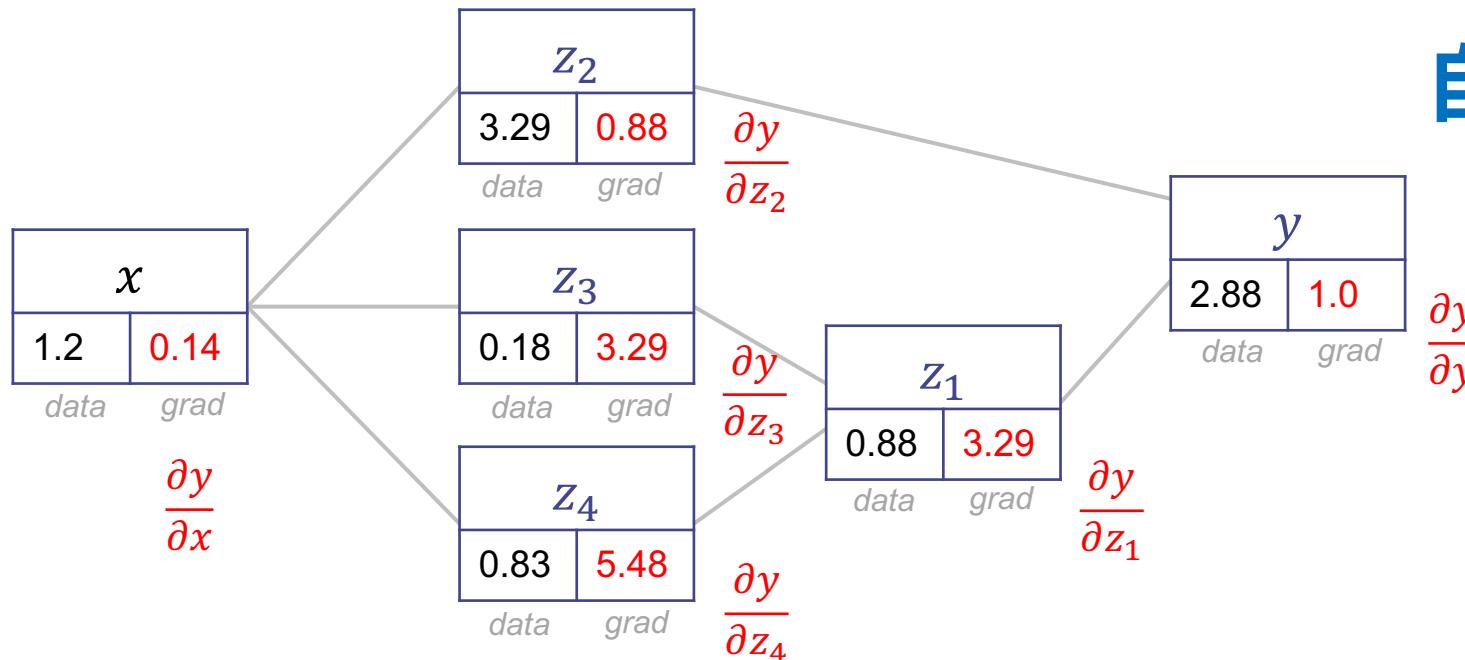
算出した-3.80を
元々のgrad=3.94に
累積して加算

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_3} \frac{\partial z_3}{\partial x} + \frac{\partial y}{\partial z_1} \frac{\partial z_1}{\partial z_4} \frac{\partial z_4}{\partial x}$$

$$\begin{aligned}
 y &= z_1 z_2 \\
 z_1 &= z_4^2 + z_3 \\
 z_2 &= 3\sqrt{x} \\
 z_3 &= \log x \\
 z_4 &= 1/x
 \end{aligned}$$

y の各変数に対する偏微分係数が全部求まる！

自動微分



$$\begin{aligned}y &= z_1 z_2 \\z_1 &= z_4^2 + z_3 \\z_2 &= 3\sqrt{z_1} \\z_3 &= \log x \\z_4 &= 1/x\end{aligned}$$

バックプロパゲーション(誤差逆伝播法)として知られているものと同じ
「リバースモードの自動微分法」と呼ばれる

自動微分とPyTorch

```
from torch import tensor, sqrt, log
```

```
x = tensor(1.2, requires_grad=True)
z2 = 3*sqrt(x)
z3 = log(x)
z4 = 1/x
z1 = z4**2 + z3
y = z1 * z2
```

```
y.backward()
print(x.grad)
```

tensor(0.14)

Forward

$$\begin{aligned}z_2 &= 3\sqrt{x} \\z_3 &= \log x \\z_4 &= 1/x \\z_1 &= z_4^2 + z_3 \\y &= z_1 z_2\end{aligned}$$

Backward

自動微分とPyTorch

確認のために途中の変数のgrad値も確認してみる

```
x = tensor(1.2, requires_grad=True)
z2 = 3*sqrt(x)
```

```
z3 = log(x)
```

```
z4 = 1/x
```

```
z1 = z4**2 + z3
```

```
y = z1 * z2
```

```
y,retain_grad()
```

```
z1,retain_grad()
```

```
z2,retain_grad()
```

```
z3,retain_grad()
```

```
z4,retain_grad()
```

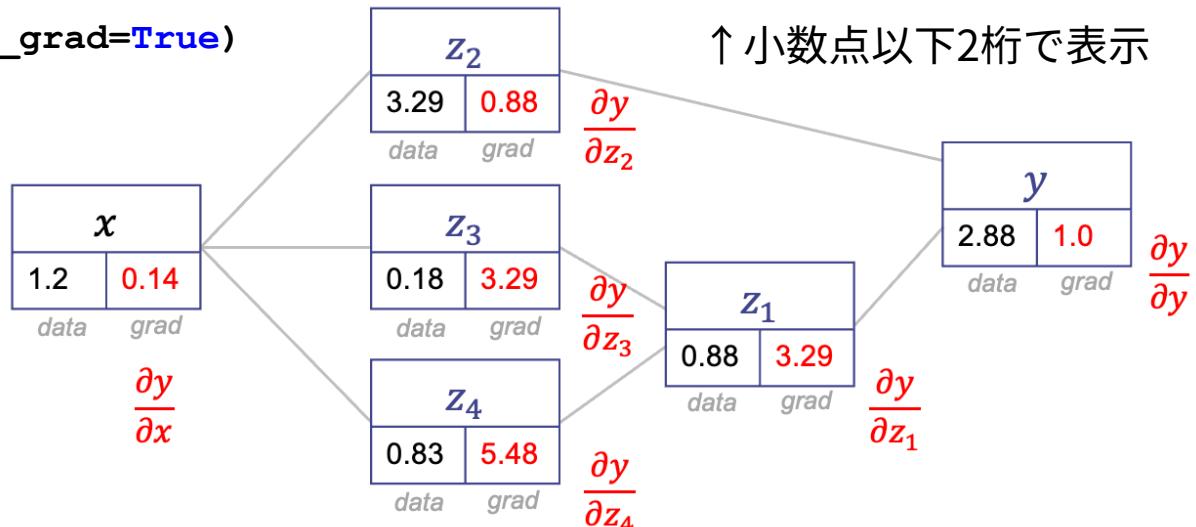
```
y.backward()
```

```
print(x.data, z2.data, z3.data, z4.data, z1.data, y.data)
print(x.grad, z2.grad, z3.grad, z4.grad, z1.grad, y.grad)
```

```
tensor(1.20) tensor(3.29) tensor(0.18) tensor(0.83) tensor(0.88) tensor(2.88)
tensor(0.14) tensor(0.88) tensor(3.29) tensor(5.48) tensor(3.29) tensor(1.)
```

```
import torch
torch.set_printoptions(2)
```

↑ 小数点以下2桁で表示



応用①：勾配法と組み合わせて関数値の最小値を計算

関数 $y = x^2 + 2x + 3 = (x + 1)^2 + 2$ の最小値を求めよ。

```
from torch import tensor
```

```
x = tensor(2.0, requires_grad=True) ← 初期値(2.0)をセット
```

```
for epoch in range(100):
```

```
    y = x**2 + 2.0*x + 3.0
```

```
    y.backward()
```

```
    x.data = x.data - 0.1 * x.grad
```

```
    x.grad = tensor(0.0)
```

- 自動微分 (Forward)
- 自動微分 (Backward)
- 勾配法
- x.gradをゼロクリア

```
x.data, y.data
```

```
(tensor(-1.0000), tensor(2.))
```

← $x = -1$ で $y = 2$ が最小値

応用②：勾配法と組み合わせて非線形回帰

```
from torch import tensor, linspace, rand
from torch.nn import Sequential, Linear, Sigmoid
import matplotlib.pyplot as plt

def true_func(x):
    return x**3 - 3.2*x**2 + 1.3*x - 2.0

num_sample = 20
xmin, xmax = -1.5, 1.5

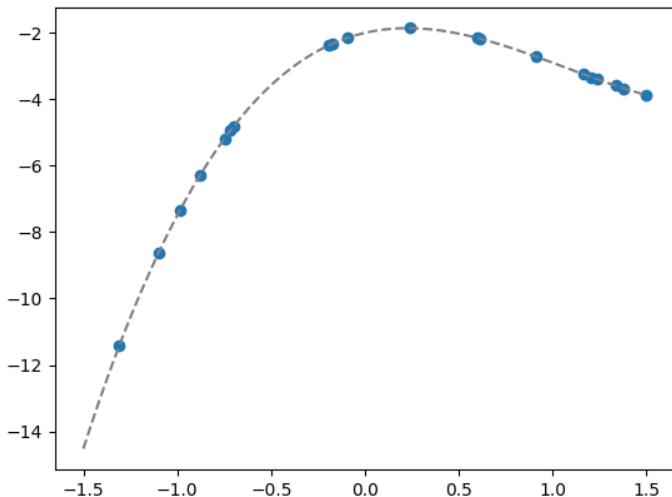
x_true = linspace(xmin, xmax, 100).view(-1,1)
y_true = true_func(x_true)

x_sample = ((xmax-xmin)*rand(num_sample) + xmin).view(-1,1)
y_sample = true_func(x_sample)

plt.plot(x_true, y_true, color='gray', linestyle='dashed')
plt.scatter(x_sample, y_sample)
plt.show()
```

←下記データの生成コード

曲線 $y = x^3 + 3.2x^2 + 1.3x - 2$
上のランダム20点 (範囲-1.5~1.5)



応用②：勾配法と組み合わせて非線形回帰

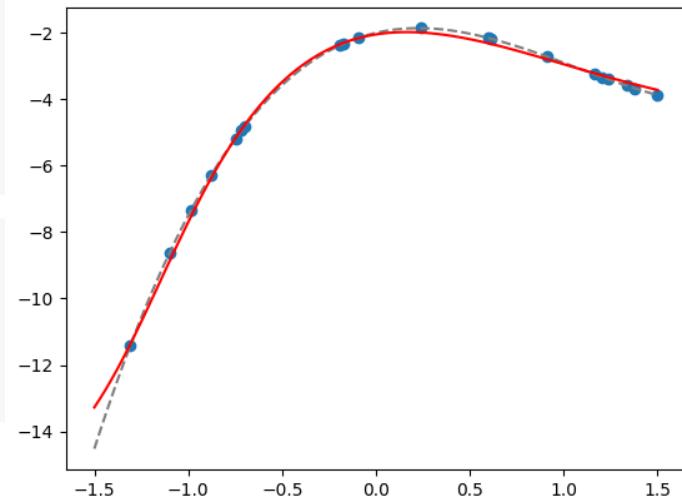
```
model = Sequential(Linear(1,10), Sigmoid(), Linear(10,1)) ← 3層ニューラルネット
```

```
for epoch in range(10000):
    y_pred = model(x_sample)          自動微分 (Forward)
    loss = ((y_pred - y_sample)**2).mean()  自動微分 (Backward)
    loss.backward()
```

```
for p in model.parameters():
    p.data = p.data - 0.01 * p.grad
    p.grad.zero_()
```

```
plt.plot(x_true, y_true, color='gray', linestyle='dashed')
plt.scatter(x_sample, y_sample)
plt.plot(x_true, model(x_true).data, color='red')
plt.show()
```

結果



PyTorchのユーティリティを使った定型

```
from torch.nn import Sequential, Linear, Sigmoid, MSELoss
from torch.optim import SGD

model = Sequential(Linear(1,10), Sigmoid(), Linear(10,1))

loss_fn = MSELoss()
optimizer = SGD(model.parameters(), lr=0.01)

for epoch in range(10000):
    y_pred = model(x_sample)
    loss = loss_fn(y_pred, y_sample)
    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

MSE = mean squared errors (平均二乗誤差)

SGD = stochastic gradient descent
(確率的勾配降下)

深層学習へ

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor
from torch.utils.data import Dataset, DataLoader

dataset = MNIST('./data', train=True, download=True, transform=ToTensor())
loader = DataLoader(dataset, batch_size=64, shuffle=True)

model = nn.Sequential(
    nn.Conv2d(1, 32, 3, 1), nn.ReLU(), nn.MaxPool2d(2), nn.Dropout(0.25),
    nn.Flatten(), nn.Linear(5408, 100), nn.ReLU(), nn.Linear(100, 10))
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

for epoch in range(10):
    for data, target in loader:
        output = model(data)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

確率的勾配法

データが大きくなるので
小出し(ミニバッチ)で処理

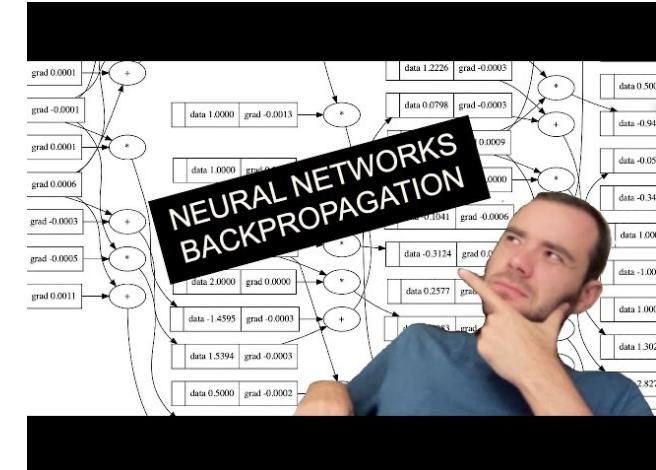
モデルは自由に設計できる

実際には下記を加えていく

- 予測精度の評価
- 学習制御 (lrスケジューリング)
- データ前処理
- 結果やモデルの保存
- GPU処理

発展：micrograd

- <https://github.com/karpathy/micrograd>
- Pythonでたった94行で書かれた单変数用の自動微分ライブラリ
- 自動微分エンジン
<https://github.com/karpathy/micrograd/blob/master/micrograd/engine.py>
- 作者はAndrey Karpathy
OpenAI社の創業メンバーで機械学習分野のスター研究者の一人
(TeslaのAI部門を率いたのち、
2023年よりOpenAIに復帰)
- 本人による素晴らしい動画解説
<https://youtu.be/VMj-3S1tku0?si=91ZWzaA4ECidua4g>

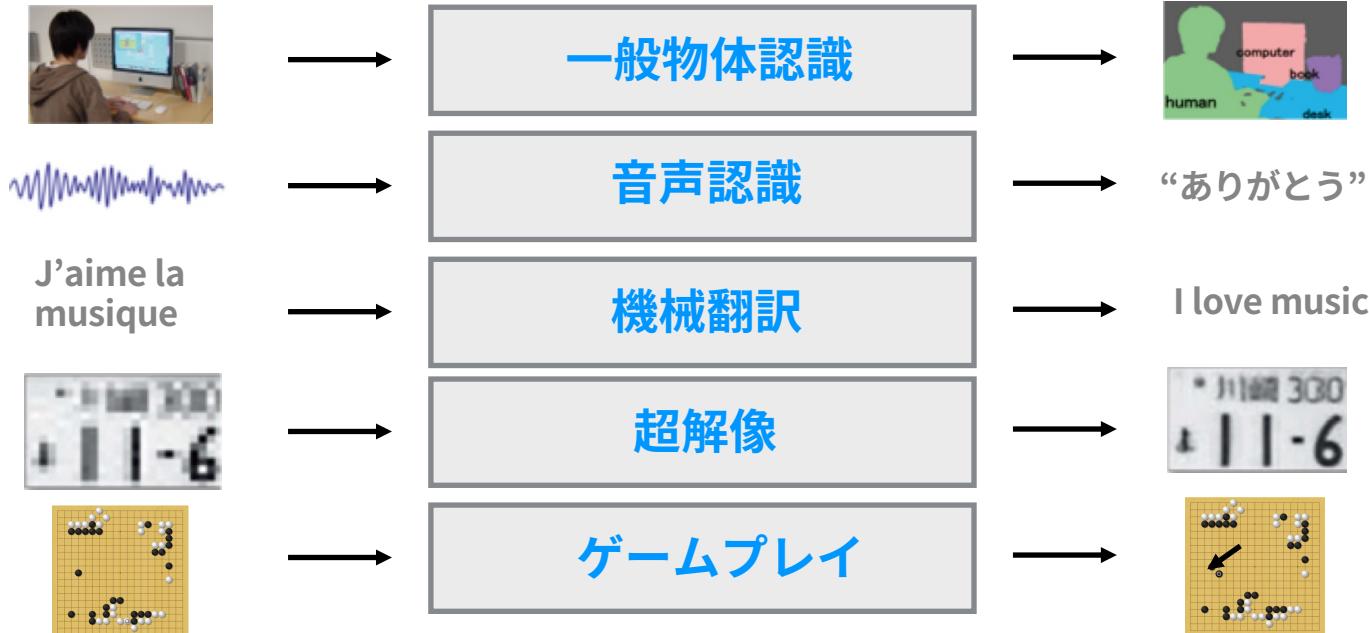


要点：自動微分による「微分可能プログラミング」

- 実際には $y = 3\sqrt{x}$ も $z = \sqrt{x}$ と $y = 3z$ の合成で書ける
- 加減乗除、三角関数、指数・対数・べき、行列計算、選択操作など、基本的な素演算の一式を用意しておけば十分
→ PyTorchの例 <https://pytorch.org/docs/stable/torch.html#math-operations>
- どんなプログラムも素演算の合成関数になってさえいれば
自動微分+勾配法でパラメタを最適にできる！
- 微分可能プログラミング = プログラムの不確定な箇所は
可変パラメタにしておき後で機械学習

機械学習＝新しい(雑な)プログラムの作り方

たくさんの入出力の見本データによって間接的に見本例の入出力を再現するプログラムを作り出す技術 (Software 2.0とも)



まとめ・Q & A

今日のたった3つの内容

1. 機械学習のしくみ
2. 深層学習と勾配法
3. 自動微分 (アルゴリズム微分)



講義中に質問があればSlidoへ

<https://app.sli.do/event/d13KtHT3VxXFbmLvxFSQjn>