Reverse-Mode 自動微分を理解する

Hiromi ISHII

2023-03-12

Tsukuba Computer Mathematics Seminar 2023

Forward-Mode と Reverse-Mode 自動微分

自動微分とは？

- 自動微分：与えられた関数の値とその微分係数を，合成関数の微分公式（連鎖律）を使って効率的・厳密に計算する方法
  - 関数の値を順に計算しながら，連鎖律を使って微分係数も計算する
  - 深層学習で損失関数の最適化の際に用いる誤差逆伝播法で注目を集める
- 関連するが異なる計算方法：
  - 記号微分：数式を記号的な構文木で表現し，微分法則に従い記号的に処理.
    - 厳密だが項数が組合せ爆発して非効率になりがち.
  - 数値微分：極限 $\lim_{\Delta x \to 0} \dfrac{f(x+\Delta x) - f(x)}{\Delta x}$ の Δx を微小数で置き換えて近似.
    - あくまで近似. Δt をどう近似するかで振る舞いが変わってくる.

連鎖律

- 皆さんお馴染の基本公式

$$\mathbb{R}^n \xrightarrow{f} \mathbb{R}^k \xrightarrow{g} \mathbb{R}^m$$

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{x}}(g \circ f) = \frac{\mathrm{d}g}{\mathrm{d}f}\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$$

- 統一性のため $\boldsymbol{u}_0 = \boldsymbol{x}, \boldsymbol{u}_1 = f(\boldsymbol{u}_0), \boldsymbol{u}_2 = g(\boldsymbol{u}_1) = g(f(\boldsymbol{u}_0))$ として並び換えれば,

$$\mathbb{R}^m \xleftarrow{\boldsymbol{u}_2} \mathbb{R}^k \xleftarrow{\boldsymbol{u}_1} \mathbb{R}^n \xleftarrow{\boldsymbol{u}_0} \mathbb{R}^0$$

$$\frac{\mathrm{d}\boldsymbol{u}_2}{\mathrm{d}\boldsymbol{u}_0} = \frac{\mathrm{d}\boldsymbol{u}_2}{\mathrm{d}\boldsymbol{u}_1}\frac{\mathrm{d}\boldsymbol{u}_1}{\mathrm{d}\boldsymbol{u}_0}$$

連鎖律　（多変数）

多変数も同様（但し，各 $\partial \boldsymbol{u}_i / \partial \boldsymbol{u}_j$ はヤコビ行列 $J_{ij} = [\partial u_{ik} / \partial u_{jl}]_{k<N_i, l<N_j}$）

$$\mathbb{R}^{n_N} \xleftarrow{\boldsymbol{u}_N} \mathbb{R}^{n_{N-1}} \xleftarrow{\boldsymbol{u}_{N-1}} \cdots \xleftarrow{\boldsymbol{u}_1} \mathbb{R}^{n_0} \xleftarrow{\boldsymbol{u}_0} \mathbb{R}^0$$

$$\frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_0} = \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_{N-1}} \frac{\partial \boldsymbol{u}_{N-1}}{\partial \boldsymbol{u}_{N-2}} \cdots \frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1} \frac{\partial \boldsymbol{u}_1}{\partial \boldsymbol{u}_0}$$

連鎖律　（多変数）

多変数も同様（但し，各 $\partial \boldsymbol{u}_i / \partial \boldsymbol{u}_j$ はヤコビ行列 $J_{ij} = [\partial u_{ik} / \partial u_{jl}]_{k<N_i, l<N_j}$）

$$\mathbb{R}^{n_N} \xleftarrow{\boldsymbol{u}_N} \mathbb{R}^{n_{N-1}} \xleftarrow{\boldsymbol{u}_{N-1}} \cdots \xleftarrow{\boldsymbol{u}_1} \mathbb{R}^{n_0} \xleftarrow{\boldsymbol{u}_0} \mathbb{R}^0$$

$$\frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_0} = \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_N} \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_{N-1}} \frac{\partial \boldsymbol{u}_{N-1}}{\partial \boldsymbol{u}_{N-2}} \cdots \frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1} \frac{\partial \boldsymbol{u}_1}{\partial \boldsymbol{u}_0} \frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{u}_0}$$

連鎖律　（多変数）

多変数も同様（但し，各 $\partial \boldsymbol{u}_i / \partial \boldsymbol{u}_j$ はヤコビ行列 $J_{ij} = [\partial u_{ik} / \partial u_{jl}]_{k<N_i, l<N_j}$）

$$\mathbb{R}^{n_N} \xleftarrow{\boldsymbol{u}_N} \mathbb{R}^{n_{N-1}} \xleftarrow{\boldsymbol{u}_{N-1}} \cdots \xleftarrow{\boldsymbol{u}_1} \mathbb{R}^{n_0} \xleftarrow{\boldsymbol{u}_0} \mathbb{R}^0$$

$$\frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_0} = \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_N} \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_{N-1}} \frac{\partial \boldsymbol{u}_{N-1}}{\partial \boldsymbol{u}_{N-2}} \cdots \frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1} \frac{\partial \boldsymbol{u}_1}{\partial \boldsymbol{u}_0} \frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{u}_0}$$

これを前から計算するのが Forward-Mode 自動微分

連鎖律　（多変数）

多変数も同様（但し，各 $\partial \boldsymbol{u}_i / \partial \boldsymbol{u}_j$ はヤコビ行列 $J_{ij} = [\partial u_{ik} / \partial u_{jl}]_{k<N_i, l<N_j}$）

$$\mathbb{R}^{n_N} \xleftarrow{\boldsymbol{u}_N} \mathbb{R}^{n_{N-1}} \xleftarrow{\boldsymbol{u}_{N-1}} \cdots \xleftarrow{\boldsymbol{u}_1} \mathbb{R}^{n_0} \xleftarrow{\boldsymbol{u}_0} \mathbb{R}^0$$

$$\frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_0} = \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_N} \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_{N-1}} \frac{\partial \boldsymbol{u}_{N-1}}{\partial \boldsymbol{u}_{N-2}} \cdots \frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1} \frac{\partial \boldsymbol{u}_1}{\partial \boldsymbol{u}_0} \frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{u}_0}$$

これを前から計算するのが Forward-Mode 自動微分

これを後から計算するのが Reverse-Mode 自動微分

Foward Mode 自動微分

- Dual Number $\mathbb{R}[d]/(d^2)$ を使って計算できる
  - $x + yd$ の y が一次微分係数を覚える役割を果す（$f(x) + f'(x)d$ と思う）
- 代数演算はそのまま延長，（区分的に）滑らかな関数についても Weil 環の標準的な C^∞- 環構造が入る
  - 難しく聞こえるが，以下のような感じ：

$$\sin(x + yd) = \sin(x) + y\cos(x)d,$$
$$\cos(x + yd) = \cos(x) - y\sin(x)d,$$
$$e^{(x+yd)} = e^x + ye^x d$$

Forward Mode 計算例

- 次で定まる関数 $f : \mathbb{R} \to \mathbb{R}^3$ の x に関する微分を求めてみよう：

$$f(x) = (\sin(x^2+1), e^{x^2}, x^3 e^{-x})$$

- 入口は $x + (\mathrm{d}x/\mathrm{d}x)d = x + d$ を入れて $f(x+d)$ を評価してみる：

$$\begin{aligned}
&f(x+d) \\
&= (\sin((x+d)^2+1), e^{(x+d)^2}, (x+d)^3 e^{-(x+d)}) \\
&= (\sin(x^2+2xd+1), e^{x^2+2xd}, (x^3+3x^2d)(e^{-x}-e^{-x}d)) \\
&= (\sin(x^2+1)+2x\cos(x^2+1)d, e^{x^2}+2xe^{x^2}d, (x^3+3x^2d)(e^{-x}-e^{-x}d)) \\
&= (\sin(x^2+1), e^{x^2}, x^3e^{x-x}) + \underbrace{(2x\cos(x^2+1), 2xe^{x^2}, -x^3e^{-x}+3x^2e^{-x})}_{\text{一階微分係数}}d
\end{aligned}$$

Forward-Mode だけでいいのでは？

- 入力が一変数，出力が多変数の場合は Forward-Mode で十分効率的
- しかし，機械学習のように $f : \mathbb{R}^N \to \mathbb{R}$ で $N \gg 0$ の場合，Forward-Mode では効率がわるすぎる！
  (a) 素朴にやると，$\mathbb{R}[d]^N$ を用意して，$i < N$ ごとに $(0, ..., 0, x + d_i, 0, ...)$ みたいなのを何回も渡すことになる
  (b) これを組にして $(\mathbb{R}[d]^N)^N$ について一挙に計算すると one-pass
  (c) いずれにせよ変数の数について自乗オーダーかかってしまい非効率！
- $\mathrm{d}\boldsymbol{x}/\mathrm{d}\boldsymbol{x} = \mathrm{d}\boldsymbol{u}_0/\mathrm{d}\boldsymbol{u}_0$ ではなく $\mathrm{d}\boldsymbol{u}_N/\mathrm{d}\boldsymbol{u}_N$ の側から辿ったらどうか？
  - これが Reverse-Mode 自動微分！

Reverse-Mode 自動微分いろいろ

Reverse-Mode 自動微分

- Reverse-Mode の概念図はこうだった

$$\mathbb{R}^{n_N} \xleftarrow{\boldsymbol{u}_N} \mathbb{R}^{n_{N-1}} \xleftarrow{\boldsymbol{u}_{N-1}} \cdots \xleftarrow{\boldsymbol{u}_1} \mathbb{R}^{n_0} \xleftarrow{\boldsymbol{u}_0} \mathbb{R}^0$$

$$\frac{\mathrm{d}\boldsymbol{u}_N}{\mathrm{d}\boldsymbol{u}_0} = \frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_N}\frac{\partial \boldsymbol{u}_N}{\partial \boldsymbol{u}_{N-1}}\frac{\partial \boldsymbol{u}_{N-1}}{\partial \boldsymbol{u}_{N-2}}\cdots\frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1}\frac{\partial \boldsymbol{u}_1}{\partial \boldsymbol{u}_0}\frac{\partial \boldsymbol{u}_0}{\partial \boldsymbol{u}_0}$$

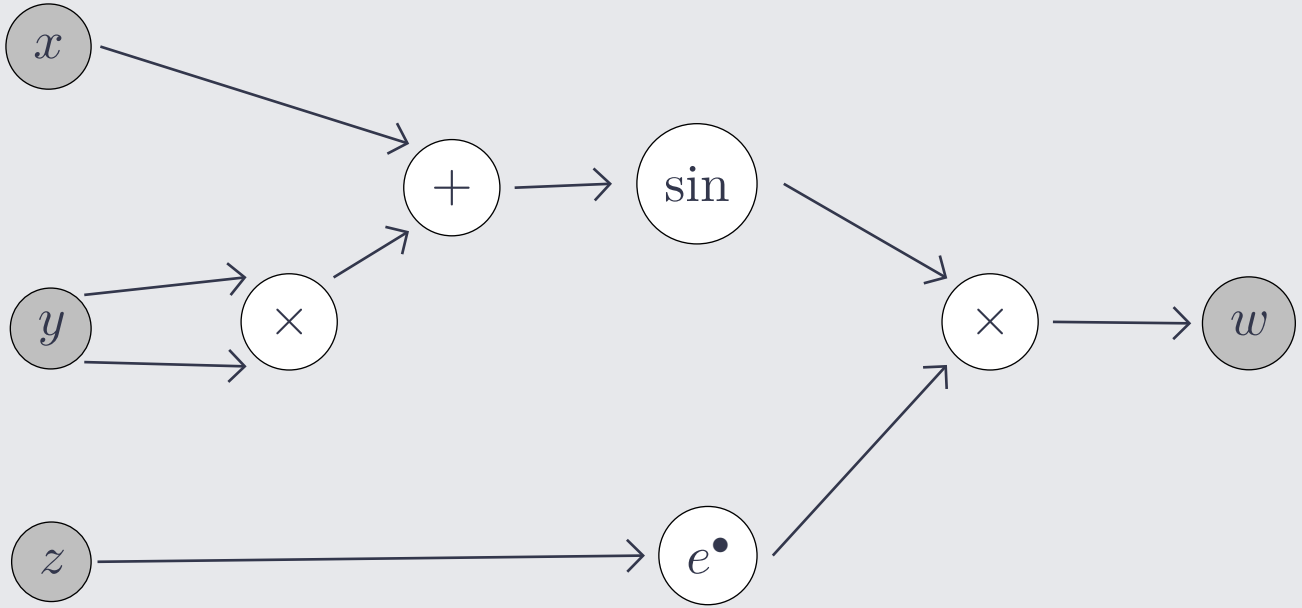
Reverse

- 問題：計算の入出力と逆順に辿る必要がある！
  - 微分係数の計算には関数の値そのものが必要，往復する必要がある
    - 機械学習で誤差逆伝播法 (back propagation) と呼ばれるもの
  - これを実現するためにいくつかやり方が知られている

Reverse-Mode 概説　（説明は[1]を参考にした）

- 計算グラフをつくって，最後から手繰っていく

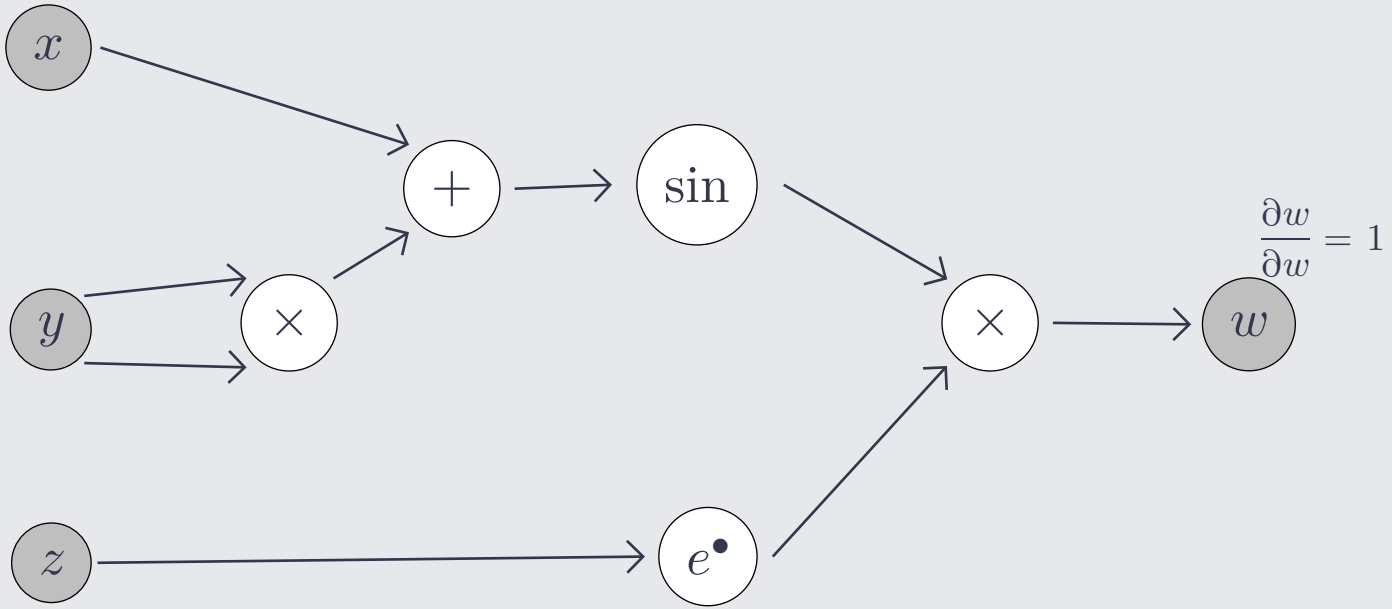
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1]を参考にした）

- 計算グラフをつくって，最後から手繰っていく

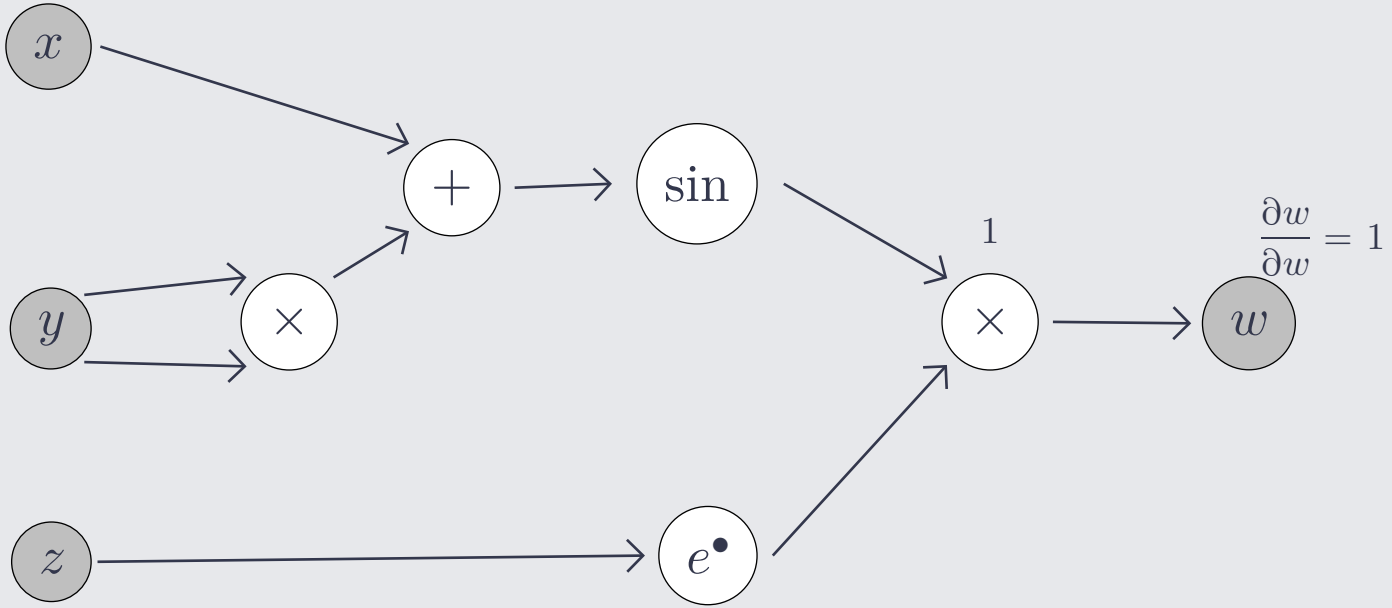
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1]を参考にした）

- 計算グラフをつくって，最後から手繰っていく

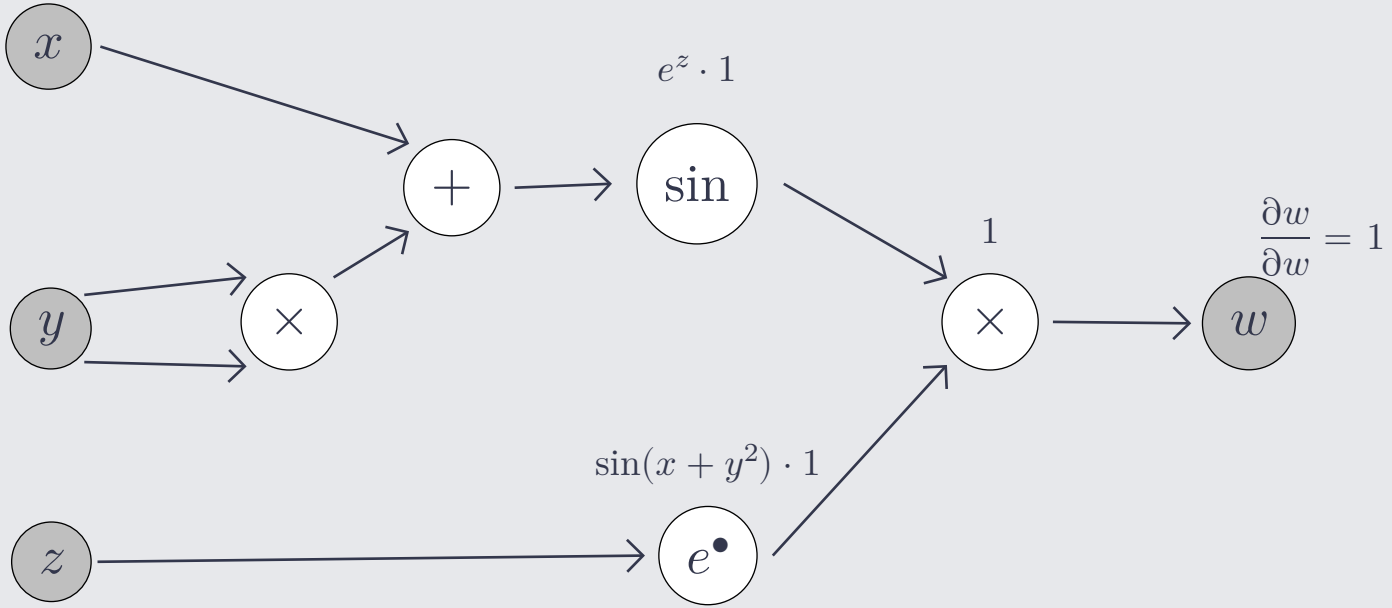
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1]を参考にした）

- 計算グラフをつくって，最後から手繰っていく

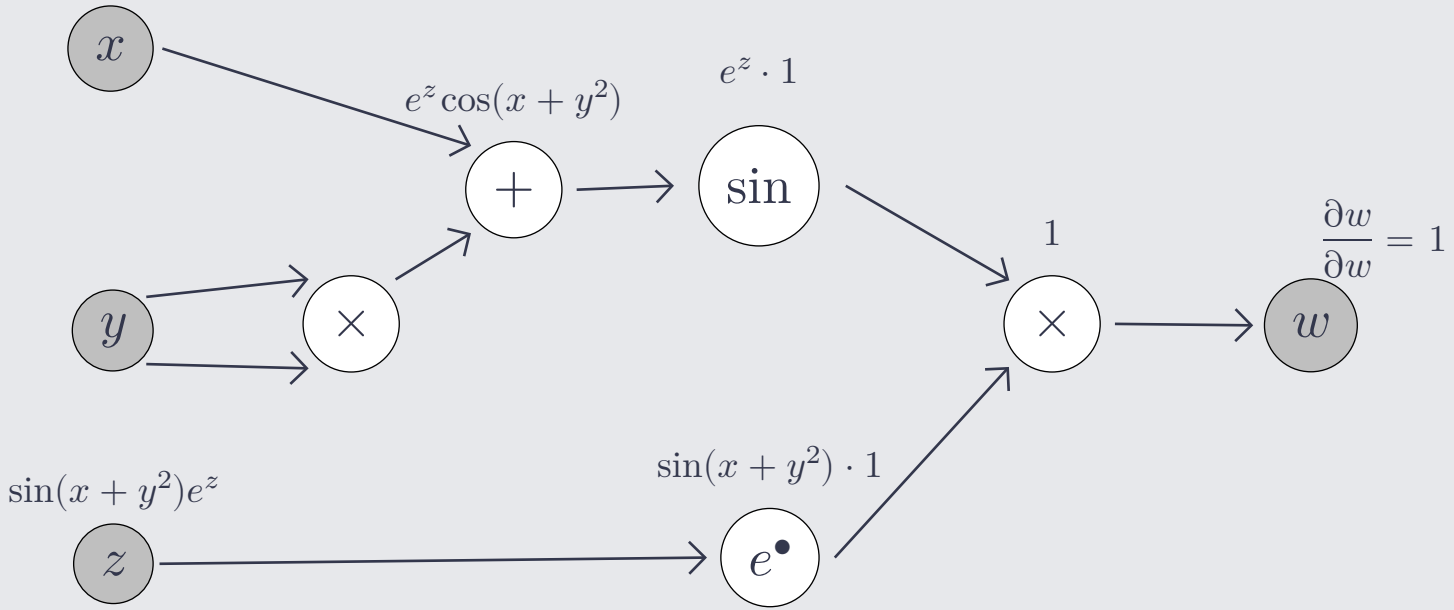
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1]を参考にした）

- 計算グラフをつくって，最後から手繰っていく

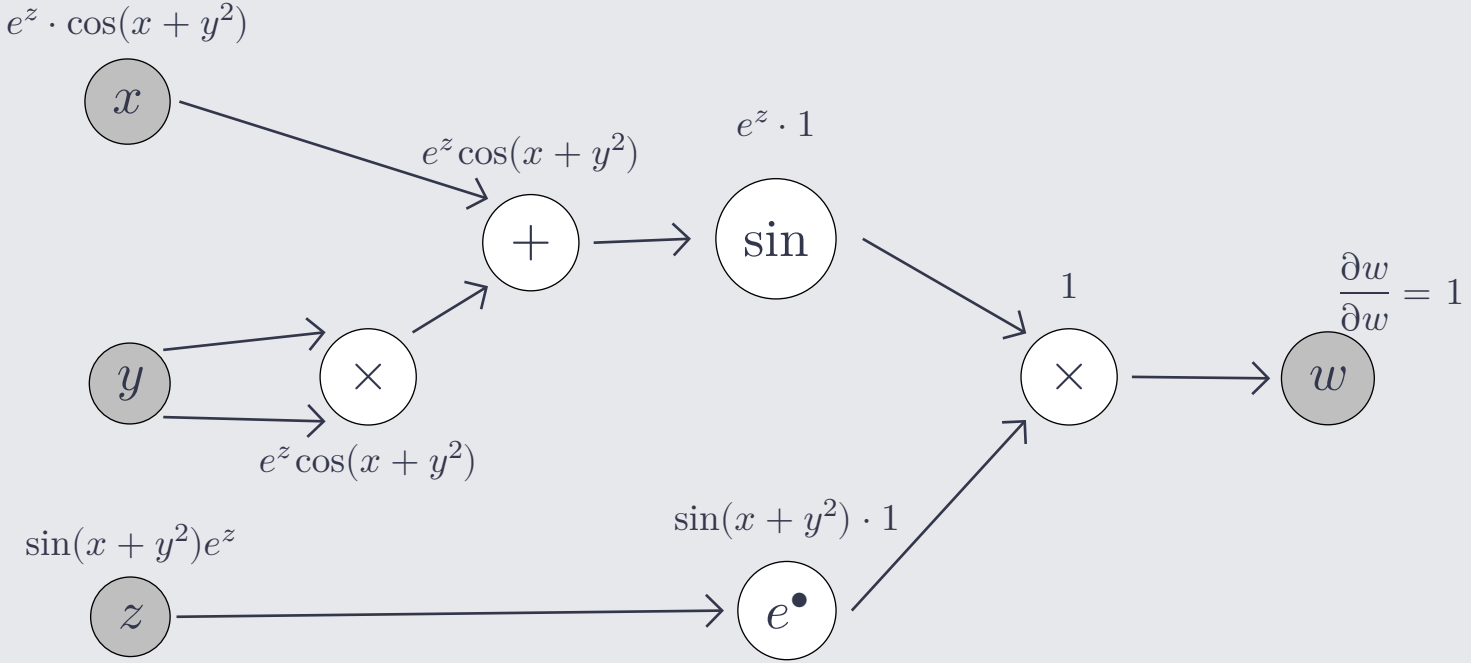
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1] を参考にした）

- 計算グラフをつくって，最後から手繰っていく

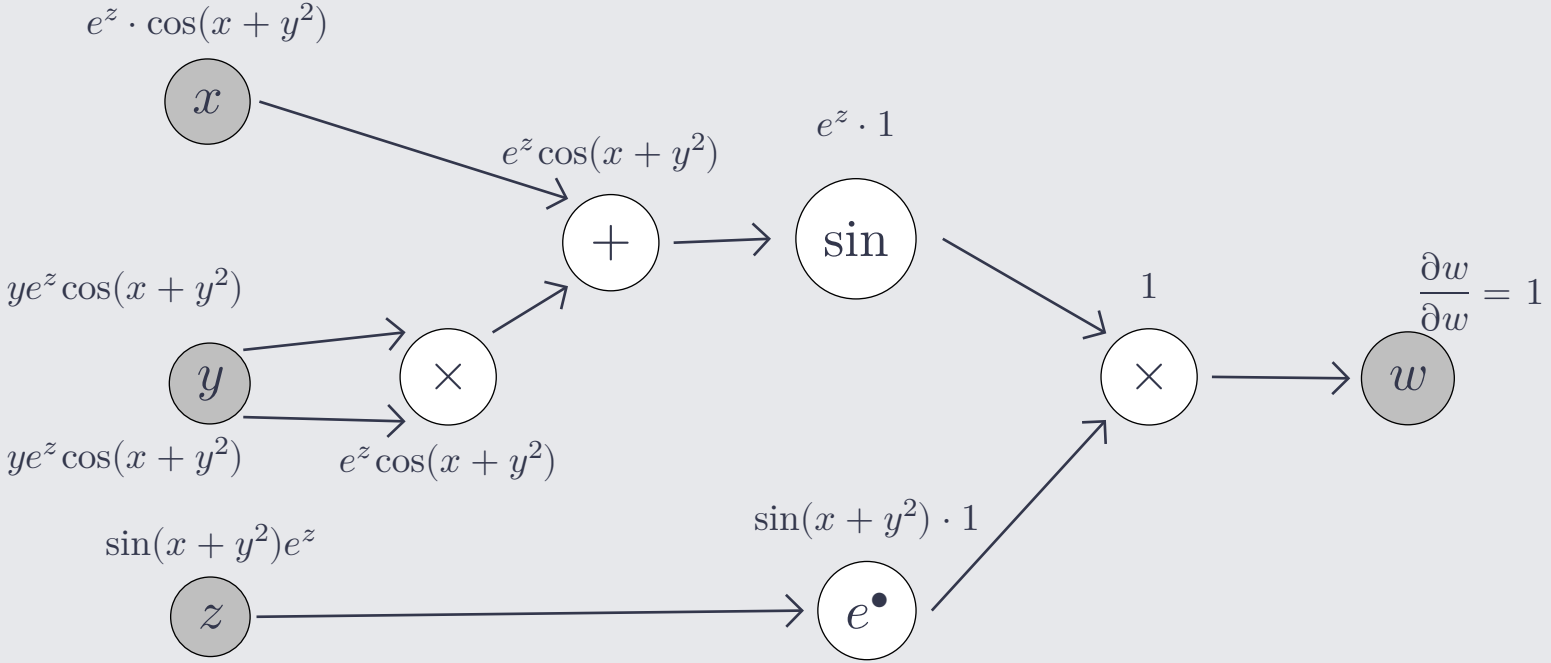
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1] を参考にした）

- 計算グラフをつくって，最後から手繰っていく

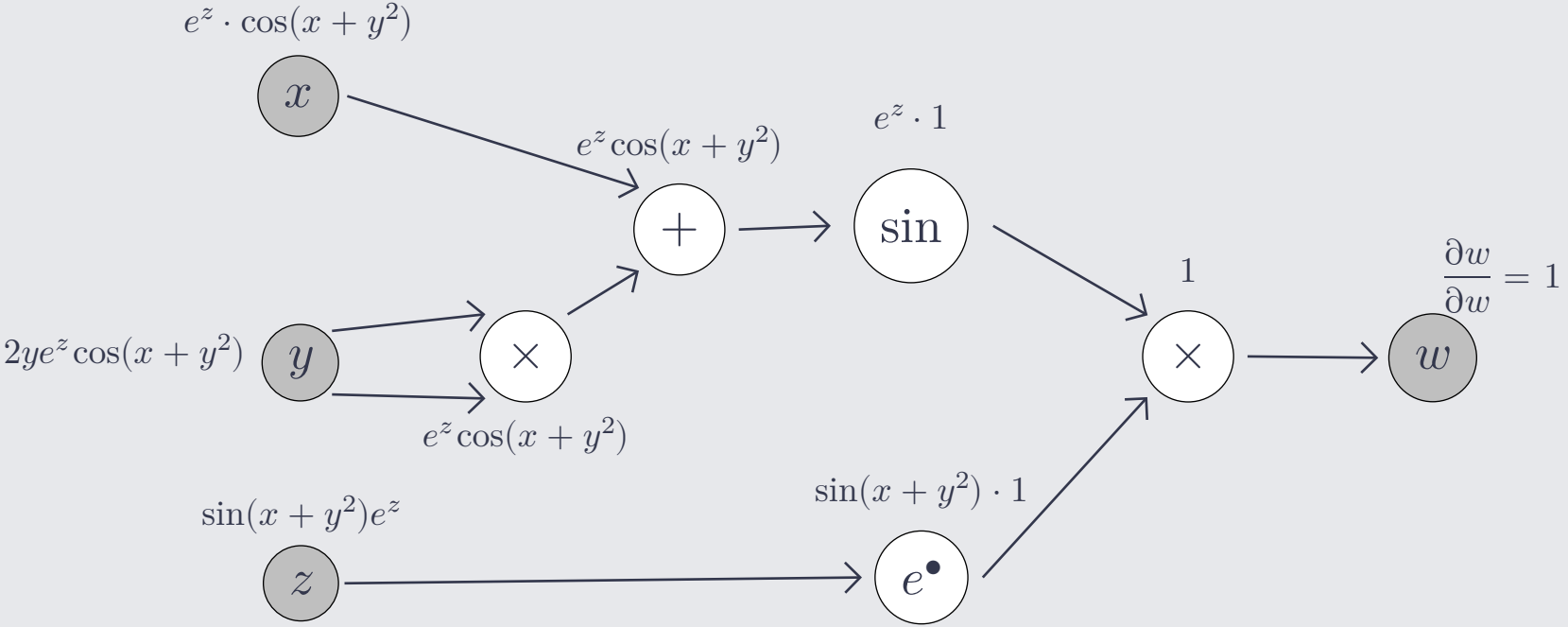
$$f(x, y, z) = \sin(x + y^2)e^z$$



Reverse-Mode 概説　（説明は[1] を参考にした）

- 計算グラフをつくって，最後から手繰っていく

$$f(x, y, z) = \sin(x + y^2)e^z$$



テープに記録していく

- 一番メジャーなやりかた（Wengert Tape や Wengert List などと呼ばれる）
- 式が出て来た順にテープに変数や部分式を記録していく
- 最後までいったら先程手繰ってノードの上に書いたような係数を，受け取り先のテープに書き込む
- 全部終わったら適宜読み出して結果を返す
- 実装の正当性・気にすべき点が職人芸

限定継続

- 限定継続：ある点から特定の点までの続きの計算を表す
  - ここでは`shift`/`reset`という一組のオペレータを使った定式化を使う
  - 例：`1 + reset( 10 + shift ( \k -> k (k 100) + 1000 ) * 2 )`
    - `shift`から見て`reset`にブチ当るまでの続きの計算が`k`に束縛される.
    - 今回の場合は`k = \n -> 10 + n * 2`
    - 上式は`1 + (\k -> k (k 100)+1000) (\n -> 10 + n*2)`と同値
    - 評価を進めれば，`1 + ((10+(10+100*2)*2)+1000) = 1431`となる.
  - `shift`がネストしている場合，外側の`shift`は`reset`の役割もする.
- Wang et al. [2] では限定継続による Reverse-Mode 自動微分を提案している

限定継続による Reverse-Mode 自動微分の実装

アイデア：順方向の計算結果から継続を逆に辿って微分係数を伝播させる

```
D x dx + D y dy =
  shift { fun k ->
    z <- D (x + y) 0.0
    k z
    dx += z.d
    dy += z.d
    return z
  }
```

```
D x dx * D y dy =
  shift { fun k ->
    z <- D (x * y) 0.0
    k z
    dx += y * z.d
    dy += x * z.d
    return z
  }
```

限定継続による Reverse-Mode 自動微分の実装 II

微分係数を求めるところで`reset`を噛ませて出口の自明な微分係数を渡してやる.

```
grad f x = {
  z <- D x 0.0
  reset { w <- f z; w.d = 1.0 }
  return z.d
}
```

- Multi-Prompt 限定継続を使えば変数の操作を減らせる
  - やってるひとが見当たらなかったので実験中（[3]）

正当性の証明

- 「うまくいきそう」以上をどう証明するか？
- Krawiec et al. [4] ではプログラム変換として捉え，プログラミング言語意味論の手法を使ってその正当性を証明している.
  - 実数値とその有限直和・直積を含む簡単な単純型付き λ- 計算の体系を定義
  - 自動微分をその体系に対するプログラム変換として定式化
  - 様々な計算体系の正規化定理の証明などに用いられる logical relation の論法を応用して，その正当性を示している
  - 線型型などを使って空間・時間計算量の漸近最適性も示している
- 最終的にはより Wengert Tape に近い手法になるが，よりシステマティックな方法で導出され，正当性がきちんと証明されている.

まとめ

まとめ

- 自動微分：合成関数の微分公式（連鎖律）を使って，関数の値とその微分係数を効率的に求める手法
  - Forward-Mode：入力の自明な微分係数を順方向に伝播させていって解く
  - Reverse-Mode：出力の自明な微分係数を逆方向に伝播させていって解く
- (入力次元) $\ll$ (出力次元) のときは Forward-Mode が効率的
- (入力次元) $\gg$ (出力次元) のときは Reverse-Mode が効率的
  - 深層学習で誤差逆伝播法と呼ばれているもの
- Reverse-Mode は往復する必要があるので実装が複雑，複数の手法がある
  - 本講演では目についたものについて説明した
  - Krawiec et al. [4] の説明が数学的にも厳密で，実装の導出方法がわかりやすいのでおすすめ

参考文献

[1] 岡谷貴之 , “深層学習 ,” 改訂第 2 版 ed., ser. 機械学習プロフェッショナルシリーズ . 講談社 , January 2022, ISBN: 978-4-06-513332-3.

[2] F. Wang et al., “Backpropagation with Continuation Callbacks: Foundations for Efficient and Expressive Differentiable Programming,” in *NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Association for Computing Machinery. New York, NY, USA: Association for Computing Machinery, December 2018, doi: 10.5555/3327546.3327682.

[3] Hiromi ISHII, `ad-delcont-primop`, URL: `https://github.com/konn/ad-delcont-primop`, January 2023.

[4] F. Krawiec et al., “Provably Correct, Asymptotically Efficient, Higher-Order Reverse-Mode Automatic Differentiation,” in *Proceedings of the ACM on Programming Languages, Volume 6, Issue POPL*, Association for Computing Machinery. New York, NY, USA: Association for Computing Machinery, January 2022, ISBN: 9784797341379; doi: 10.1145/3498710.