DEEP LEARNING JP
[DL Papers]

"Stochastic Taylor Derivative Estimator: Efficient amortization for arbitrary differential operators"

Kensuke Wakasugi, Panasonic Holdings Corporation.

http://deeplearning.jp/

書誌情報

■タイトル：
Stochastic Taylor Derivative Estimator:
Efficient amortization for arbitrary differential operator

■著者：Shi, Z., Hu, Z., Lin, M., & Kawaguchi, K.

■所属：National University of Singapore、Sea AI Lab

■選書理由

- NeurIPS2024 Best Paper
- 自身の業務（マテリアルズ・インフォマティクス）において、
物理法則を満たした予測が重要となるため

※特に記載しない限り、図表は上記論文からの引用です。

概要

損失関数に微分演算子を含む最適化問題を考える

$$\arg\min_{\theta} f\left(\mathbf{x}, u_\theta(\mathbf{x}), \mathcal{D}^{\alpha^{(1)}} u_\theta(\mathbf{x}), \ldots, \mathcal{D}^{\alpha^{(n)}} u_\theta(\mathbf{x})\right), \quad u_\theta : \mathbb{R}^d \to \mathbb{R}^{d'}.$$

入力　**NNの出力**　**微分演算子**

n階微分
※ただし、入力がd次元の場合、
どの次元で微分するかで場合の数が多数ある

$$\mathcal{D}^\alpha = \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \ldots \partial x_d^{\alpha_d}}, \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d)$$

微分テンソルサイズ $O(d^k)$

■応用例：

- 偏微分方程式の求解
- Physics-informed machine learning（PINN）
- 拡散モデルにおける入力変数の更新
- Adversarial Attack
- attentionの可視化

Contribution

汎用的で効率的な高次元自動微分手法を提案

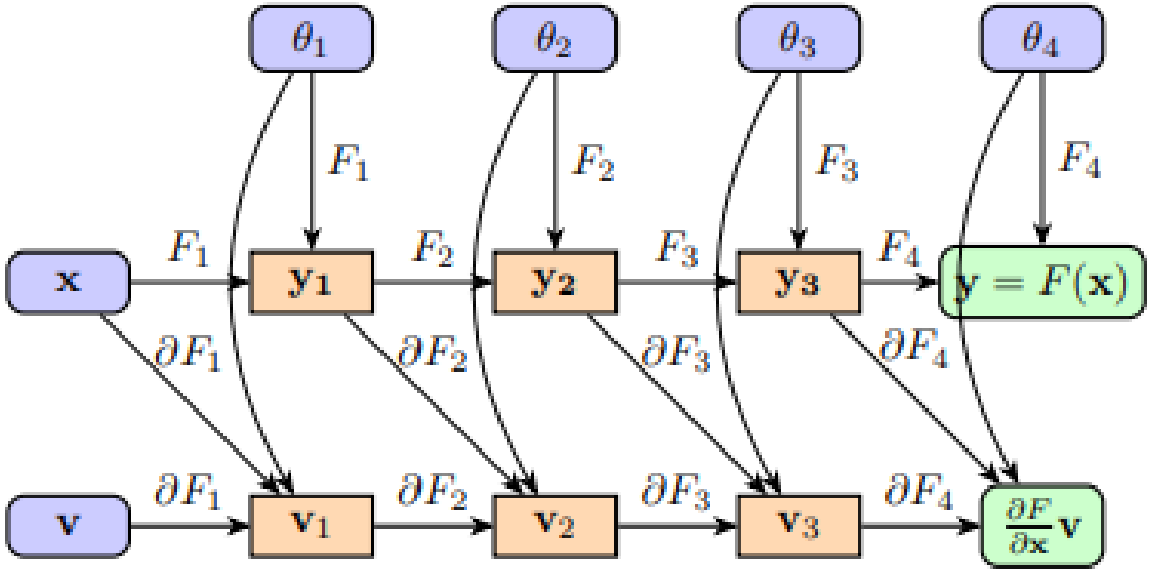
1. 効率的な高次元自動微分方法を提案
2. 既存手法では主にラプラス演算に焦点を当てていたが、
   本手法では一般の偏微分方程式に適用可能
3. 既存手法のSDGD(偏微分の確率的計算)や
   HTE（トレースの確率的計算）を一般化
4. 100万次元の偏微分方程式を８分で解けることを、実験的に示した
   @NVIDIA A100 40GB GPU

事前知識：偏微分の計算方向

偏微分の計算方法として、前向き・後向きがある

Forward mode AD

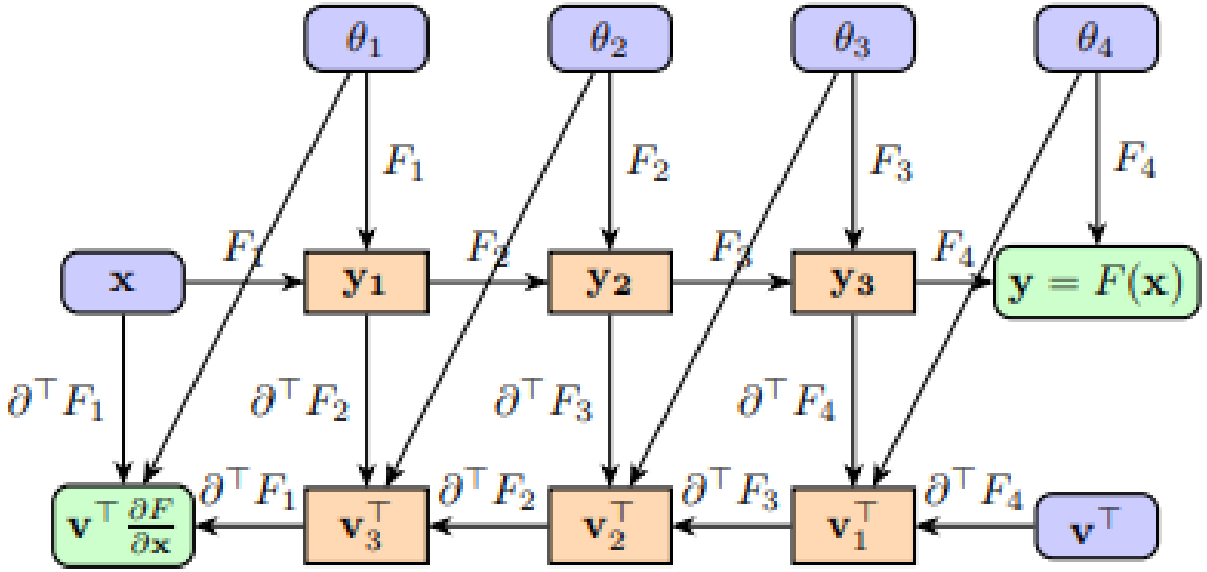
$$\frac{\partial F}{\partial \mathbf{x}}\mathbf{v} = \partial F(\mathbf{x})(\mathbf{v}) = [\partial F_L \circ \partial F_{L-1} \circ \cdots \circ \partial F_1](\mathbf{x})(\mathbf{v}).$$



- 提案手法はこちらに該当
- 計算グラフ長 O(L)、必要メモリ O（max(d,h)）

Backward mode AD

$$\mathbf{v}^\top \frac{\partial F}{\partial \mathbf{x}} = \partial^\top F(\mathbf{x})(\mathbf{v}^\top) = [\partial^\top F_1(\mathbf{x}) \circ \cdots \circ \partial^\top F_{L-1}(\mathbf{y}_{L-2}) \circ \partial^\top F_L(\mathbf{y}_{L-1})](\mathbf{v}^\top),$$

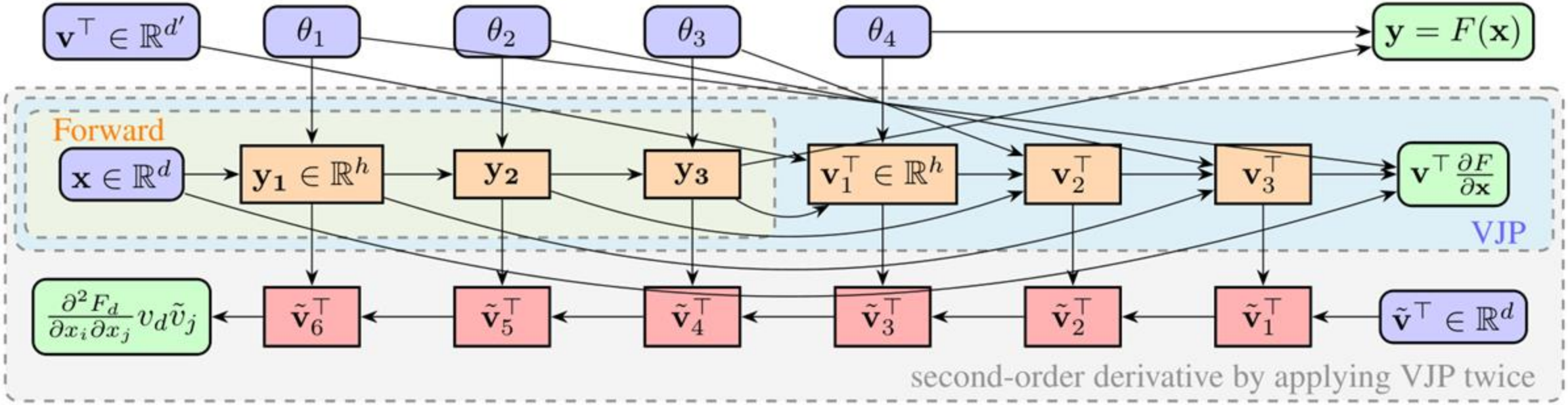


- Back Propagationはこちら
- 計算グラフ長 O(2L)、必要メモリ O（2(d＋(L-1)h)）

高階微分の非効率性

Backwardの場合、微分するごとに計算量が２倍に

3.2 Inefficiency of the first-order AD for high-order derivative on inputs



Forwardで高階微分を求めたい

Forward mode AD 再考

高次微分をNNの層毎の演算で求める

$$\frac{\partial F}{\partial \mathbf{x}}\mathbf{v} = \partial F(\mathbf{x})(\mathbf{v}) = [\partial F_L \circ \partial F_{L-1} \circ \cdots \circ \partial F_1](\mathbf{x})(\mathbf{v}).$$

■ ポイント

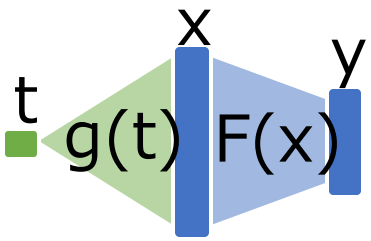
- Fについて合成関数に拡張できる → 層毎の計算で全体微分を計算
- 右辺をNNで計算する → 左辺が求まる

- 上記を高次微分にも拡張したい

合成関数の偏微分とテイラー展開

適当な関数g(t)を置くと、一変数合成関数の微分をみなせる

$$\partial F(\mathbf{x})(\mathbf{v}) = \partial F(g(t))(g'(t)) = \left.\frac{\partial F}{\partial \mathbf{x}}\right|_{\mathbf{x}=g(t)} g'(t) = \frac{\mathrm{d}}{\mathrm{d}t}[F \circ g](t).$$



0階微分、1階微分に拡張

$$\mathrm{d}F(J_g(t)) = J_{F\circ g}(t) = \left([F \circ g](t), \frac{\mathrm{d}}{\mathrm{d}t}[F \circ g](t)\right) = (F(\mathbf{a}), \partial F(\mathbf{a})(\mathbf{v})).$$

k階微分に拡張

$$\begin{aligned}\mathrm{d}^k F(J_g^k(t)) =& J_{F\circ g}^k(t) = \left([F \circ g](t), \frac{\partial}{\partial t}[F \circ g](t), \frac{\partial^2}{\partial t^2}[F \circ g](t), \ldots, \frac{\partial^k}{\partial t^k}[F \circ g](t)\right) \\ =& (F(\mathbf{a}), \partial F(\mathbf{a})(\mathbf{v}^{(1)}), \partial^2 F(\mathbf{a})(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}), \ldots, \partial^k F(\mathbf{a})(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(k)})),\end{aligned}$$

合成関数の偏微分とテイラー展開

合成関数の微分が、関数g(t)のt微分とF(x)のx微分の積であらわされる

Faa di Bruno's formula：一変数合成関数の高次微分の公式

$$\partial^2 F(\mathbf{a})(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) = \frac{\partial^2}{\partial t^2}[F \circ g](t) = D_F(\mathbf{a})\mathbf{v}^{(2)} + D_F^2(\mathbf{a})_{d',d_1,d_2} v_{d_1}^{(1)} v_{d_2}^{(1)}.$$

合成関数F○gのk階t微分がF,gそれぞれの微分で表現される。

$J_g = \{g(t), g'(t)\}$

$J_{F\circ g} = \{[F\circ g](t), \frac{\partial}{\partial t}[F\circ g](t)\}$

補足：合成関数の微分の一般項

一変数合成関数の微分に関する公式

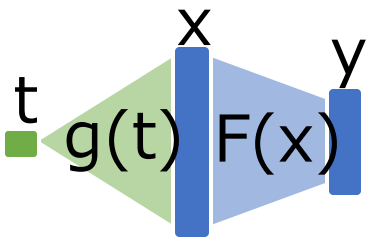
$$
\begin{aligned}
\frac{\partial}{\partial x} f(g(x)) =& f^{(1)}(g(x)) \cdot g^{(1)}(x) \\
\frac{\partial^2}{\partial x^2} f(g(x)) =& f^{(1)}(g(x)) \cdot g^{(2)}(x) + f^{(2)}(g(x)) \cdot [g^{(1)}(x)]^2 \\
\frac{\partial^3}{\partial x^3} f(g(x)) =& f^{(1)}(g(x)) \cdot g^{(3)}(x) + 3 f^{(2)}(g(x)) \cdot g^{(1)}(x) \cdot g^{(2)}(x) + f^{(3)}(g(x)) \cdot [g^{(1)}(x)]^3
\end{aligned}
\tag{37}
$$

$$
\frac{\partial^k}{\partial x^k} f(g(x)) = \sum_{\substack{(p_1,\ldots,p_k)\in\mathbb{N}^k, \\ \sum_{i=1}^k i\cdot p_i = k}} \frac{k!}{\prod_i^k p_i!(i!)^{p_i}} \cdot (f^{(\sum_{i=1}^n p_i)} \circ g)(x) \cdot \prod_{j=1}^k \left(\frac{1}{j!} g^{(j)}(x) \right)^{p_j} . \tag{38}
$$

合成関数の偏微分とテイラー展開 再掲

適当な関数g(t)を置くと、一変数合成関数の微分をみなせる

$$\partial F(\mathbf{x})(\mathbf{v}) = \partial F(g(t))(g'(t)) = \left.\frac{\partial F}{\partial \mathbf{x}}\right|_{\mathbf{x}=g(t)} g'(t) = \frac{\mathrm{d}}{\mathrm{d}t}[F \circ g](t).$$



0階微分、1階微分に拡張

$$\mathrm{d}F(J_g(t)) = J_{F\circ g}(t) = \left([F \circ g](t), \frac{\mathrm{d}}{\mathrm{d}t}[F \circ g](t)\right) = (F(\mathbf{a}), \partial F(\mathbf{a})(\mathbf{v})).$$

k階微分に拡張

$$\begin{aligned}\mathrm{d}^k F(J_g^k(t)) =& J_{F\circ g}^k(t) = \left([F \circ g](t), \frac{\partial}{\partial t}[F \circ g](t), \frac{\partial^2}{\partial t^2}[F \circ g](t), \ldots, \frac{\partial^k}{\partial t^k}[F \circ g](t)\right) \\ =& (F(\mathbf{a}), \partial F(\mathbf{a})(\mathbf{v}^{(1)}), \partial^2 F(\mathbf{a})(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}), \ldots, \partial^k F(\mathbf{a})(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(k)})),\end{aligned}$$

提案手法

Forward計算で高次微分を伝搬

$$\partial^2 F(\mathbf{a})(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) = \frac{\partial^2}{\partial t^2}[F \circ g](t) = D_F(\mathbf{a})\mathbf{v}^{(2)} + D_F^2(\mathbf{a})_{d',d_1,d_2} v_{d_1}^{(1)} v_{d_2}^{(1)}.$$

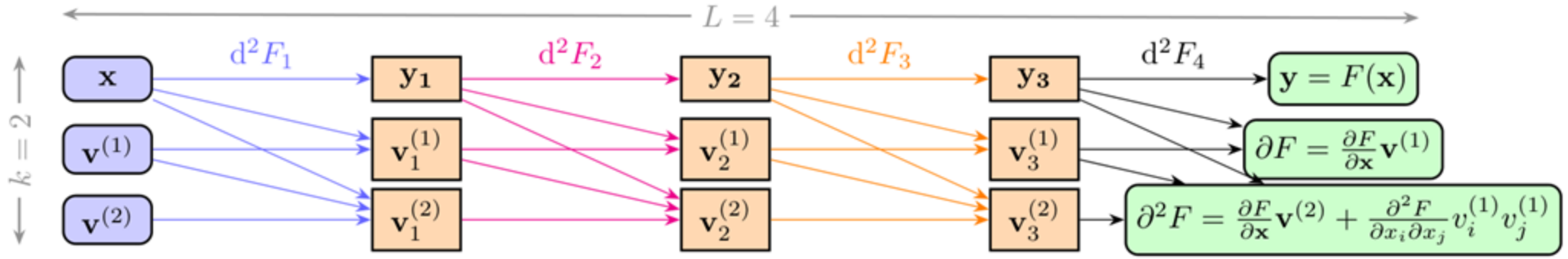


Figure 2: The computation graph of d^2F for F with 4 primitives. Parameters θ_i are omitted. The first column from the left represents the input 2-jet $J_g^2(t) = (\mathbf{x}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$, and d^2F_1 pushes it forward to the 2-jet $J_{F_1 \circ g}^2(t) = (\mathbf{y}_1, \mathbf{v}_1^{(1)}, \mathbf{v}_1^{(2)})$ which is the subsequent column. Each row can be computed in parallel, and no evaluate trace needs to be cached.

実用例

求めたい微分項に応じて、g(t)を設計

Laplacian From Eq. 9 we know that the quadratic form of Hessian can be computed through ∂^2 by setting $\mathbf{v}^{(2)} = \mathbf{0}$ and $\mathbf{v}^{(1)} = \mathbf{e}_j$. Therefore, the STDE of the Laplacian operator is given by

$$\tilde{\nabla}^2{}_J u_\theta(\mathbf{a}) = \frac{d}{|J|}\sum_{j\in J}\frac{\partial^2}{\partial x_j^2}u_\theta(\mathbf{a}) = \frac{d}{|J|}\sum_{j\in J}\partial^2 u_\theta(\mathbf{a})(\mathbf{e}_j,\mathbf{0}) = \frac{d}{|J|}\sum_{j\in J}\mathrm{d}^2 u_\theta(\mathbf{a},\mathbf{e}_j,\mathbf{0})_{[2]} \tag{16}$$

where J is the sampled index set, and the subscript $[2]$ means taking the second-order tangent from the output jet. See example implementation in JAX in Appendix A.4.

High-order *diagonal* differential operators We call a differential operator *diagonal* if it is a linear combination of diagonal elements from the derivative tensor: $\mathcal{L} = \sum_{j=1}^{d}\frac{\partial^k}{\partial x_j^k}$. From Eq. 43 we see that setting the first-order tangent $\mathbf{v}^{(1)}$ to $\mathbf{e}_j$ and all other tangents $\mathbf{v}^{(i)}$ to the zero vector gives the desired high-order diagonal element:

$$\tilde{\mathcal{L}}_J u_\theta(\mathbf{a}) = \frac{d}{|J|}\sum_{j\in J}\frac{\partial^k}{\partial \mathbf{x}_j^k}u_\theta(\mathbf{a}) = \frac{d}{|J|}\sum_{j\in J}\partial^k u_\theta(\mathbf{a})(\mathbf{e}_j,\mathbf{0},\dots). \tag{17}$$

二次の偏微分方程式

Second-order parabolic PDEs Second-order parabolic PDEs are a large class of PDEs. It includes the Fokker-Planck equation in statistical mechanics to describe the evolution of the state variables in stochastic differential equations (SDEs), which can be used for generative modeling [38]. It also includes the Black-Scholes equation in mathematical finance for option pricing, the Hamilton-Jacobi-Bellman equation in optimal control, and the Schrödinger equation in quantum physics and chemistry. Its form is given by

$$\frac{\partial}{\partial t}u(\mathbf{x},t) + \frac{1}{2}\operatorname{tr}\left(\sigma\sigma^\top(\mathbf{x},t)\frac{\partial^2}{\partial \mathbf{x}^2}u(\mathbf{x},t)\right) + \nabla u(\mathbf{x},t)\cdot\mu(\mathbf{x},t) + f(t,\mathbf{x},u(\mathbf{x},t),\sigma^\top(\mathbf{x},t)\nabla u(\mathbf{x},t)) = 0. \tag{18}$$

We have a second order derivative term $\frac{1}{2}\operatorname{tr}\left(\sigma(\mathbf{x},t)\sigma(\mathbf{x},t)^\top\frac{\partial^2}{\partial \mathbf{x}^2}u(\mathbf{x},t)\right)$ with *off-diagonal* term. The off-diagonals can be easily removed via a change of variable:

$$\frac{1}{2}\operatorname{tr}\left(\sigma(\mathbf{x},t)\sigma(\mathbf{x},t)^\top\frac{\partial^2}{\partial \mathbf{x}^2}u(\mathbf{x},t)\right) = \frac{1}{2}\sum_{i=1}^{d}\partial^2 u(\mathbf{x},t)(\sigma(\mathbf{x},t)\mathbf{e}_i,\mathbf{0}). \tag{19}$$

See derivation in Appendix E. Its STDE samples over the d terms in the expression above.

浅水波の方程式

2D Korteweg-de Vries (KdV) equation Consider the following 2D KdV equation

$$u_{ty} + u_{xxxy} + 3(u_y u_x)_x - u_{xx} + 2u_{yy} = 0. \tag{20}$$

All the derivative terms can be found in the pushforward of the following jet:

$$\begin{aligned} \mathfrak{J} = \mathrm{d}^{13}u(\mathbf{x}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(13)}),\ \mathbf{v}^{(3)} = \mathbf{e}_x, \mathbf{v}^{(4)} = \mathbf{e}_y, \mathbf{v}^{(7)} = \mathbf{e}_t, \mathbf{v}^{(i)} = \mathbf{0}, \forall i \notin \{3,4,7\}, \\ u_x = \mathfrak{J}_{[1]},\ u_y = \mathfrak{J}_{[2]},\ u_{xx} = \mathfrak{J}_{[4]},\ u_{xy} = \mathfrak{J}_{[5]}/35, \\ u_{yy} = \mathfrak{J}_{[6]}/35,\ u_{ty} = \mathfrak{J}_{[9]}/330,\ u_{xxxy} = \mathfrak{J}_{[11]}/200200. \end{aligned} \tag{21}$$

補足

複数回の演算を組み合わせて算出する場合も

$$\frac{\partial}{\partial x_i^2 \partial x_j} u_\theta(\mathbf{x}) = [\partial^4 u_\theta(\mathbf{x})(\mathbf{e}_i, \mathbf{e}_j, \mathbf{0}, \mathbf{0}) - \partial^4 u_\theta(\mathbf{x})(\mathbf{e}_i, \mathbf{0}, \mathbf{0}, \mathbf{0}) - 3\partial^2 u_\theta(\mathbf{x})(\mathbf{e}_j, \mathbf{0})]/6. \quad (48)$$

- 三階微分の要素の一つを計算する例
- 任意の微分項に対する、jetの設計は自明ではないが、求めることはできる？

提案手法 再掲

Forward計算で高次微分を伝搬

$$\partial^2 F(\mathbf{a})(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}) = \frac{\partial^2}{\partial t^2}[F \circ g](t) = D_F(\mathbf{a})\mathbf{v}^{(2)} + D_F^2(\mathbf{a})_{d',d_1,d_2} v_{d_1}^{(1)} v_{d_2}^{(1)}.$$

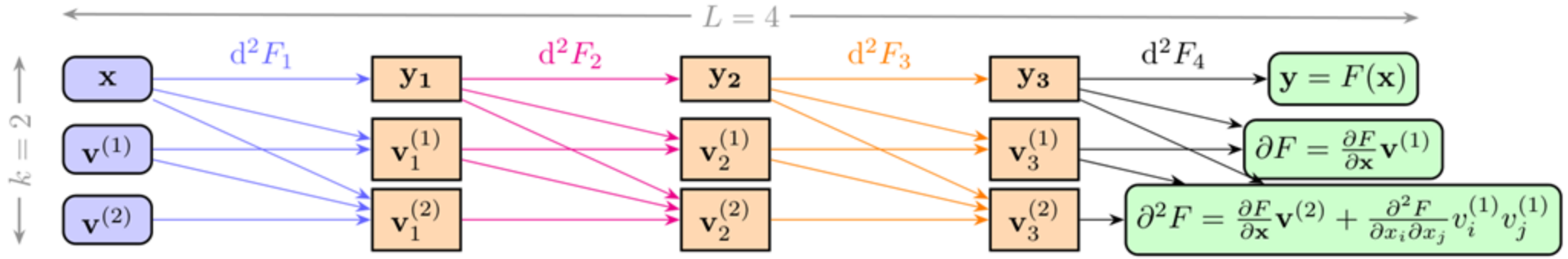


Figure 2: The computation graph of d^2F for F with 4 primitives. Parameters θ_i are omitted. The first column from the left represents the input 2-jet $J_g^2(t) = (\mathbf{x}, \mathbf{v}^{(1)}, \mathbf{v}^{(2)})$, and d^2F_1 pushes it forward to the 2-jet $J_{F_1 \circ g}^2(t) = (\mathbf{y}_1, \mathbf{v}_1^{(1)}, \mathbf{v}_1^{(2)})$ which is the subsequent column. Each row can be computed in parallel, and no evaluate trace needs to be cached.

関連研究：Stochastic Dimension Gradient Descent

入力次元をミニバッチ化し、計算効率を改善

$$\mathcal{D} := \sum_{j=1}^{N_{\mathcal{D}}} \mathcal{D}_j \approx \frac{N_{\mathcal{D}}}{|J|} \sum_{j \in J} \mathcal{D}_j := \tilde{\mathcal{D}}_J,$$

ミニバッチ学習と類似した考え方

入力次元d → J と置き換わりは計算量など軽減できるが、
微分次数 kへの依存性は減らない。高階微分における計算量軽減が望まれる

ランダム次元削減との併用

計算対象の次元を限定することで、計算量を削減

先行手法（再掲）

$$\mathcal{D} := \sum_{j=1}^{N_{\mathcal{D}}} \mathcal{D}_j \approx \frac{N_{\mathcal{D}}}{|J|} \sum_{j \in J} \mathcal{D}_j := \tilde{\mathcal{D}}_J,$$

提案手法

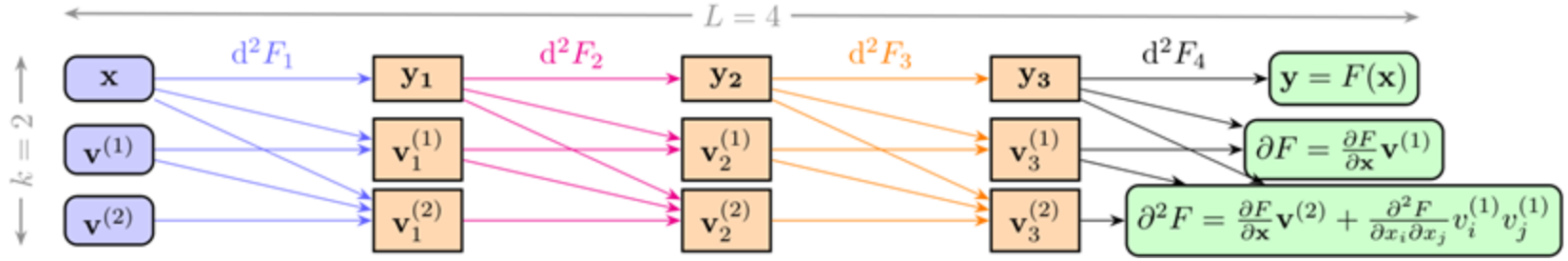
$$\mathbb{E}_{J_g^l \sim p}[\partial^l u(J_g^l)] = \mathbb{E}[v_{d_1}^{(v_1)} \dots v_{d_k}^{(v_k)}] = D_u^k(\mathbf{a}) \cdot \mathbf{C}(\mathcal{L}) = \mathcal{L}u(\mathbf{a}),$$

サンプリングした次元に関しての期待値が
全体で求めたい微分の値に一致（不偏推定）

Stochastic Taylor Derivative Estimator (STDE)のメリット

Forward計算で微分計算実現し、計算量の発散を抑制

1. 任意の次元、次数の微分演算子に適用可能
2. データ次元dと微分次数kのスケーリング問題を同時に解決
3. サンプリングJに関して、並列計算が可能



■Backwardと比較
メモリ：O（$2^{k-1}(d + (L - 1)h)$）→O（kd）
計算量：O（$2^k(dh + (L - 1)h2)$）→ O（k^2dL）

補足：jet形式のsparseとdense

sparseなjetと、denseなjetを設計可能（場合による）

- 基本はスパースな（標準ベクトルと0で構成）jet
- 場合によりdenseなjetを作れる。
  下記例では、ガウス分布からサンプリングしたvを利用

We generalize the dense construction to **arbitrary second-order differential operators** using a multivariate Gaussian distribution with the eigenvalues of the corresponding coefficient tensor as its covariance. Suppose $\mathcal{D}$ is a second-order differential operator with coefficient tensor $\mathbf{C}$. With the eigendecomposition $\mathbf{C}'' = \frac{1}{2}(\mathbf{C} + \mathbf{C}^\top) + \lambda\mathbf{I} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$ where $-\lambda$ is smaller than the smallest eigenvalue of $\mathbf{C}$, we can construct a STDE for $\mathcal{D}$:

$$\mathbb{E}_{\mathbf{v}\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma})}[\partial^2 u(\mathbf{a})(\mathbf{U}\mathbf{v},\mathbf{0})] - \lambda\mathbb{E}_{\mathbf{v}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\partial^2 u(\mathbf{a})(\mathbf{v},\mathbf{0})] = D_u^2(\mathbf{a})\cdot[\mathbf{C}'' - \lambda\mathbf{I}] = D_u^2(\mathbf{a})\cdot\mathbf{C}. \quad (22)$$

参考：Hutchinson Trace Estimation

Hutchinson Trace Estimation

This example illustrates the estimation the Hessian trace of a neural network using Hutchinson's method [Hutchinson, 1990], which is an algorithm to obtain such an an estimate from matrix-vector products:

Let $A \in \mathbb{R}^{D \times D}$ and $v \in \mathbb{R}^{D}$ be a random vector such that $\mathbb{E}[vv^T] = I$. Then,

$$\mathrm{Tr}(A) = \mathbb{E}[v^T A v] = \frac{1}{V} \sum_{i=1}^{V} v_i^T A v_i.$$

引用：Hutchinson Trace Estimation — BackPACK 1.2.0 documentation

Experiments

Physics-informed neural networksで実験

■ PDEs（ただし、境界条件は先行文献利用し、自動的に満たすように設計）

$$\mathcal{L}u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \quad \mathcal{B}u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega,$$

$$\ell_{\text{residual}}(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N_r}) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \mathcal{L}u_\theta(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(i)}) \right|^2$$

■ Amortized PINNs

$$\tilde{\ell}_{\text{residual}}(\theta; \{\mathbf{x}^{(i)}\}_{i=1}^{N_r}, J, K) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left[\tilde{\mathcal{L}}_J u_\theta(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(i)}) \right] \cdot \left[\tilde{\mathcal{L}}_K u_\theta(\mathbf{x}^{(i)}) - f(\mathbf{x}^{(i)}) \right],$$

■ 方程式の例

- Allen-Cahn equation: $\mathcal{L}u(\mathbf{x}) = \nabla^2 u(\mathbf{x}) + u(\mathbf{x}) - u(\mathbf{x})^3.$

Ablation study on the performance gain

一定の誤差範囲の下での計算時間を比較

Table 1: Speed ablation for the two-body Allen-Cahn equation.

Speed (it/s) ↑	100 D	1K D	10K D	100K D	1M D
Backward mode SDGD (PyTorch) [13]	55.56	3.70	1.85	0.23	OOM
Backward mode SDGD	40.63	37.04	29.85	OOM	OOM
Parallelized backward mode SDGD	1376.84	845.21	216.83	29.24	OOM
Forward-over-Backward SDGD	778.18	560.91	193.91	27.18	OOM
Forward Laplacian [24]	**1974.50**	373.73	32.15	OOM	OOM
STDE	1035.09	**1054.39**	**454.16**	**156.90**	**13.61**

Ablation study on the performance gain

メモリ量の比較

Table 2: Memory ablation for the two-body Allen-Cahn equation.

Memory (MB) ↓	100 D	1K D	10K D	100K D	1M D
Backward mode SDGD (PyTorch) [13]	1328	1788	4527	32777	OOM
Backward mode SDGD	553	565	1217	OOM	OOM
Parallelized backward mode SDGD	539	579	1177	4931	OOM
Forward-over-Backward SDGD	537	579	1519	4929	OOM
Forward Laplacian [24]	**507**	913	5505	OOM	OOM
STDE	543	**537**	**795**	**1073**	**6235**

比較結果

ベースラインとなるSDGDを最適化知したうえでも、STDEが大幅に優位

■ JAX vs PyTorch
- SDGDはPythorchのため、JAXも実装。JAXだと~15倍速、~4倍メモリ効率

■ Parallelization
- 元のSDGDは非並列のため、並列化も検証
- ~15倍速、ピークメモリの低減

■ Forward Laplacian
- 入力次元が100までは最良だが、ランダマイズがないのでスケールしない
- 1000次元を超えるとSDGDが有利

■ STDE(提案手法)
- 最良のSDGDと比較して、10倍速、4倍メモリ効率

まとめ

偏微分方程式を含む問題における、汎用性の高い手法を提案

■ Applicability
- 偏微分方程式に汎用的に適用できる
- 敵対的攻撃、特徴貢献度解析、メタ学習

■ Limitations
- 分散提言は考慮できておらず、future works
- ランダムバッチサイズを小さくすると、速度・メモリが改善するが、分散との兼ね合いは不明
- ネットワークパラメータの学習には適していない

■ Future works
- AD とランダム化数値線形代数のつながりを示唆
- 高次元ラプラシアンを計算する必要がある多体シュレーディンガー方程式や
  数理金融で多数の用途がある高次元ブラック ショールズ方程式への応用を期待

所感

- 現実的に大きなk階微分を使うことは少なそうではあるが、汎用性の高さが際立つ
- 実用面では、g(t)の設計に苦労しそうではあるが、
  よく使われる方程式であれば、ハードルは低そう。使い勝手に期待。