第5回ML論文読み会

Neural Ordinary Differential Equations

NeurIPS 2018

金融市場部CPM室　榎本 拓実

自己紹介

- 榎本 拓実
- 金融市場部CPM室ポートフォリオマネジメントGr所属
- 業務
  - XVAコントロール: 特にモデル・システム高度化, MLによるヘッジ手法開発
    - XVA: デリバティブに係る様々な価格調整(X Value Adjustment). クレジットリスク(C)やファンディングコスト(F), 規制コスト(K)等時価評価に織り込まれていないリスクを管理
  - CPM室横断でのDX推進
  - 一貫してクオンツ・デスククオンツ業務

紹介する論文

- Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David K. Duvenaud, “Neural Ordinary Differential Equations”, NeurIPS 2018
- 趣旨: 連続版のネットワーク構造, 連続性により様々なメリットを実現
- 近年様々な発展系や研究が進展(第2回での福田さん発表論文含む).
- その基礎として今回ご紹介.
- NeurIPS 2018でのベストペーパー

構成

- Neural ODEの概念, 勾配計算のアイデア(Adjoint method)
- Neural ODEのメリット
- 実験結果
- Neural ODEの制約
- まとめ

アイデア
ネットワークの連続化

- 通常のネットワーク: 層から層への変換を繰り返す
- 差分や変数変換を学習する性能の良いモデル
  - Residual Net
  - RNN
  - Normalized flow
- $x_{n+1} - x_n = f(x_n)$と変換するモデル→極限を取れば微分方程式

Residual NetとEuler法

- Residual Net: 差分を学習
- Euler法によりODEを離散化→差分による更新
- Residual Netの時の更新式を $u(t+\Delta) = u(t) + f_\theta(u_t, t)\Delta t$ として極限を取るとODE $\frac{du}{dt}(t) = f_\theta(u(t), t)$ (Neural ODE)
- 逆にNeural ODEにEuler法適用するとResidual Net
- この意味で, Neural ODEは連続版のネットワークモデル

【ResidualNet更新式】

$$h_1 = f_1(x_{in}) + x_{in}$$

$$h_2 = f_2(h_1) + h_1$$

$$h_3 = f_3(h_2) + h_2$$

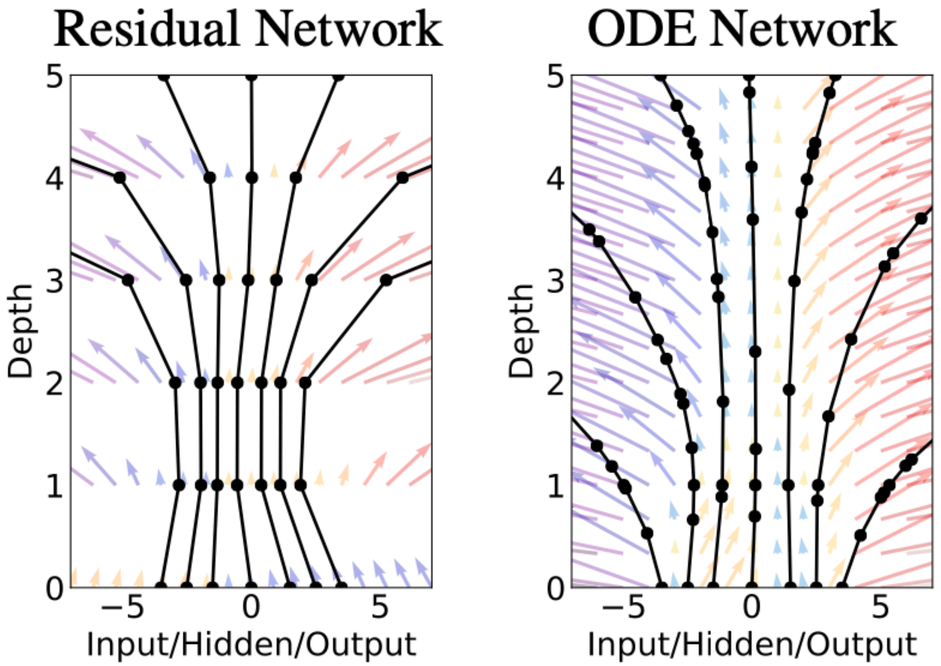
$$x_{out} = f_4(h_3) + h_3$$

ODE net

- Neural ODEは $\frac{du}{dt}(t) = f_\theta(u(t), t)$ と書ける
- この f_θ をNN等により表現し, 最適な θ を求める.
- ODEの解 $u(t)$ はODE Solverにより数値計算する
- ODEにより層を”連続”にして変換する！
  - f_θ は層毎でなく, 単一の関数により保持できる→メモリ削減

Residual net vs ODE net

- 右図: 黒丸が評価点
- Residual netでは層から層への変換のみ
- ODE netでは評価する点をコントロール可能
  (Solverの設定によって決まる)



Neural ODEのメリット

- Memory Efficiency
  - Adjoint法の応用により, 計算グラフの保持なく勾配計算できる
- Adaptive Computation
  - 性質の良いODESolverの利用により数値解の誤差と推論誤差のトレードオフやステップ幅の制御が可能
- Parameter efficiency
  - 層毎のパラメータ保持でなく層の深さをドリフト関数での変数の1つとして管理

Neural ODEのメリット

- Scalable and invertible normalizing flows
  - Normalized flowでのボトルネックであった行列式計算をトレース計算に置き換えられる
- Continuous time-series models
  - RNNでのタイムステップを連続化→時間発展を自然にモデリング

順伝播と勾配計算

- Input: ODENetのパラメータ θ, 初期値$u(t_0) = x_{in}$
- 順伝播: $u(t_1) = \int_{t_0}^{t_1} f_\theta(u(s), s)ds = \mathrm{ODESolver}(u(t_0), f_\theta)$ により計算
- 勾配計算はどうする?
  - 数値計算は結局離散化するので, 離散化した時の各計算でbackprop?
  - それだと今までのモデルと同じ. 計算グラフ保持に大量のメモリ要.

Adjoint method

- 損失関数$L = L \circ u(t) = L(u(t))$の導関数があるODEを満たすので, このODE(Adjoint ODE)を終端条件により解く(Adjoint method)
- Adjoint ODEを直接を解けば勾配が得られる→計算グラフは不要！
- 古くはPontryagin et al. (1962)に遡る
- 気象予報の数値計算によく用いられている?

Adjoint methodによる勾配計算

- 順伝播での解 $u_\theta(t)$ に対し, $L(t,\theta) = L(u_\theta(t))$ と置く
- 各時点での状態 $u(t)$ に対しての勾配を$a(t) = -\partial L/\partial u(t)$ と置く(adjoint)
  - (細かい注) ODEの解の存在と一意性により, 各時点での状態 $u(t)$ によりODEの解空間がパラメトライズできる. $a(t)$ はこの(t 毎に定まる)パラメータ表示に対する勾配. (無限次元関数空間での)変分ではない
- このadjointは以下のODEの解になっている！

$$\frac{da}{dt}(t) = -a(t)^T \frac{\partial f_\theta}{\partial u}(u_\theta(t), t)$$

Adjoint methodによる勾配計算

- Adjoinit ODE $\frac{da}{dt}(t) = -a(t)^T \frac{\partial f_\theta}{\partial u}(u_\theta(t), t)$ に偏微分が入っているが, この偏微分は未知関数 $a(t)$ に関しての微分ではないので, ODEになっている
- 同様に$a_\theta(t) = -\frac{\partial L}{\partial \theta}(t)$, $a_t(t) = -\frac{\partial L}{\partial t}(t)$ もあるODEを満たすことが示される
- 全てのODEをまとめて1つの(ベクトル値)ODEとして扱う(Augmented ODE)ことで, ODE Solverの利用1回で全ての偏微分を計算

勾配計算アルゴリズム

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient $\partial L / \partial \mathbf{z}(t_1)$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

def aug_dynamics$([\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta)$: ▷ Define dynamics on augmented state

  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

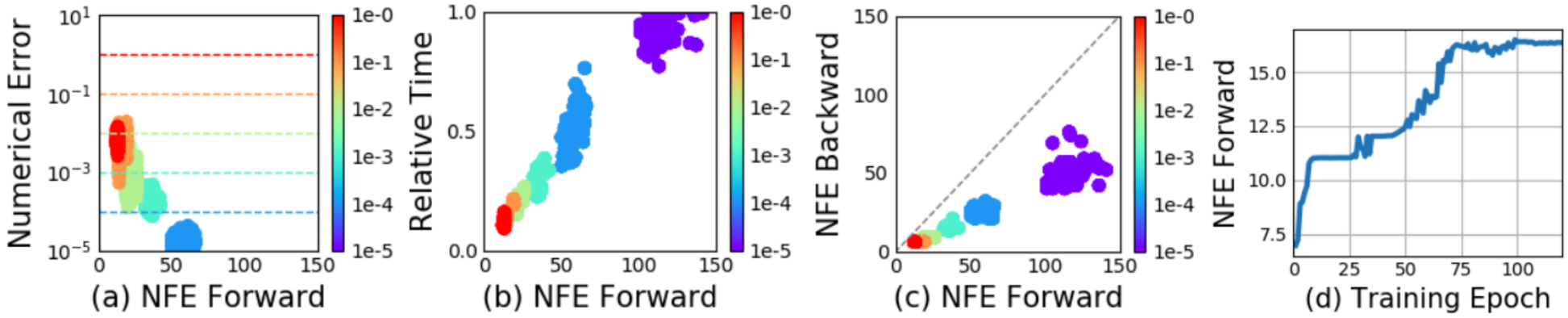
実験1: 教師あり学習
MNIST定量評価

- 1-Layer MLP, ResNet, RK-Net(Runge-Kuttaによる順伝播＋通常の逆伝播), ODE-Net(adjoint methodでの逆伝播)を比較
  - L: レイヤー数, $\tilde{L}$: ODEステップ数(レイヤー数と解釈できる)
  - ODE-Netでの順伝播はDormand-Prince法らしい(刻み幅が自動調整される)
- ResNetより少ないパラメータ数、メモリ使用量で同程度の精度
- 高次元ODEを解く問題でもワーク

	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

実験1: 教師あり学習
MNIST定性

- 関数評価回数(NFE)≈層の深さ. ODE netでは自動調整可能
- 関数評価回数が増えると数値誤差減少…自明
- 関数評価回数が増えると時間増加…自明
- 逆伝播時の関数評価回数は順伝播の半分…Adjoint methodは計算効率も良い！
- 訓練が進むにつれて関数評価回数増加…ダイナミクスが複雑化



Normalizing Flows

- Rezende et al.(2015)により提案された手法
- 変分推論において探索する分布の範囲を上手く定義する必要がある
- 解析的に表示することが難しい分布, 解析的に表示できる分布の変数変換により与えることを目指す(normalizing=解析的に表示できる分布に(逆)変換)
- 確率変数 y から z への変換を f , 逆変換を g とすれば$(z = f(y),\ y = g(z))$,

$$p_Y(y) = p_Z(z) \left| \frac{\partial g}{\partial z} \right|^{-1}$$

Normalizing Flows

- Normalizing Flow(NF)には2つの課題
  - この g をNNにより表現したいが, 可逆な変換になる必要がある
  - ヤコビアンの計算が必要だが工夫しないとコスト大(次元 D に対し$O(D^3)$)
- この変換をODEにより表現することでどちらも解決可能！
  - この論文でContinuous Normalizing Flows(CNF)と呼ぶ手法を提案

Normalizing FlowとNeural ODE

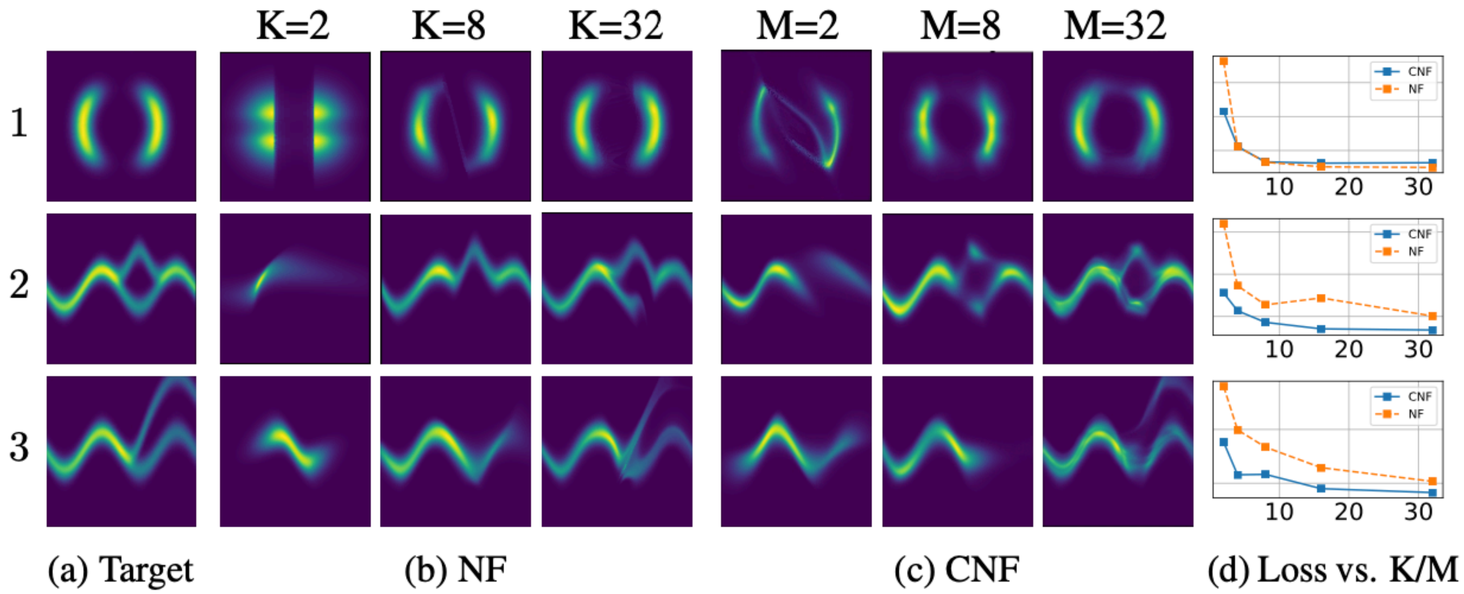
- $z(t)$: t毎に確率変数, 分布 $p(t)$, f : 適当な正則性を満たす
- (サンプル毎に)$\frac{dz}{dt}(t) = f(z(t), t)$ を満たすならば, $p(t)$ は以下を満たす:

$$\frac{\partial \log p}{\partial t}(t) = -\operatorname{tr}\left(\frac{\partial f}{\partial z}(z(t), t)\right)$$

  - ODEにより変換するので, 可逆性は保証される(解の存在と一意性)
  - $\log p(t)$が求めたいもの. その計算に使うのは行列式ではなくトレース！
- 線形性も効率的に利用できる. 論文では $f(z, t) = \sum_{n=1}^{M} \sigma_n(t)\, f_n(z)$としたケースをCNTと呼んでいる.
  - Mは隠れ層ユニット数と解釈

実験2: CNT実例
Density matching

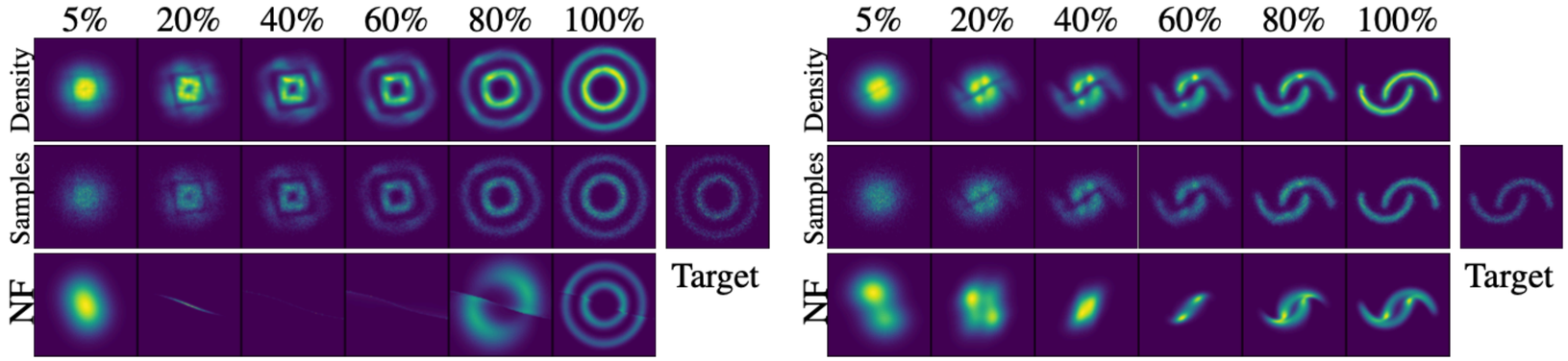
- Target分布になるよう訓練
- 損失関数はKL divergence
- K: (planar) NFの深さ
  - 基本パーツの合成回数
- M: CNFでの隠れ層ユニット数



- CNFの方がNFより損失小. 出来上がり分布も定性的に合っているように見える

実験2: CNT実例
Maximum Likelihood Training

- CNTのメリット: 逆変換が元の変換と同コスト(ODE逆向きに解くだけ)
  - 分布計算は$p_Y(y) = p_Z(f(y))\left|\frac{\partial f}{\partial y}\right|$だが, サンプリングは$y = f^{-1}(z)$
- %はflowの時間(深さ)
- CNF (1段目)は上手くフィットするだけでなく, 遡っても元のGauss分布に戻れる
- サンプリング(2段目)も上手くできている



(a) Two Circles　　(b) Two Moons

実験3: 時系列データ生成

- RNN: 時間が離散化→時間不均一なデータの扱いが難しい
- 従来は以下の対策を行なっていた
  - 欠損値補完
  - タイムスタンプ情報も入力にする
- Neural ODEは時間が連続であるため, 不均一なものを簡単に扱える！
- 実験結果説明は論文ご参照…

Neural ODEの制約

- 論文では以下制約に触れている:
  - ミニバッチ化が非自明: データ毎に初期値を変えてODEを解く為
  - ODEのトレランスをハイパラに持ってしまう
  - 状態遷移を逆向きに再現するとき誤差が増加: checkpointingで逓減化
- Neural ODEでの数値解が不安定. 4次Runge-Kuttaでも不十分らしい(実装はDormand-Prince)
- 量子化も危険では? 単精度でのODE数値計算はかなり怪しい…
- 表現の柔軟性を高めるにはアーキテクチャに工夫が必要(特にCNF)…FFJOLD[G]で解決

まとめ

- NNによるODEダイナミクスを与え, ODE Solverにより順伝播・勾配を計算するという全く新しい手法
- 連続ダイナミクスゆえの様々なメリット
- 様々な拡張・研究が進んでいるポテンシャルのあるモデル

References

- [C] Chen et al., “Neural Ordinary Differential Equations”, NeurIPS 2018
- [R] Rezende et al. “Variational Inference with Normalizing Flows”, ICLR 2015
- [G] Grathwohl et al., “FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models”, ICLR 2019.

Appendix

Neural ODEの発展

- ダイナミクスの複雑化
  - Neural SDE
  - Neural PDE
  - Neural ODE processes
  - Neural manifold ODE
  - Fractional SDE-net (第2回での福田さん発表)
  - Neural Rough SDE
  - Neural Delay ODE

Neural ODEの発展

- ODE数値計算方法
  - Homotopy method
  - Symplectic method
  - Nesterov acceleration
- Scalable gradient for SDE: adjoint methodのSDE版