Redes Neurais em EDO

Daniel Jacob Tonn
Escola de Matemática Aplicada
Fundação Getúlio Vargas
Rio de Janeiro, RJ
danieljt.djt@gmail.com

Wendell Oliveira
Escola de Matemática Aplicada
Fundação Getúlio Vargas
Rio de Janeiro, RJ
wendellshadow13@gmail.com

Abstract

Este trabalho explora uma nova abordagem em redes neurais que se comporta como uma equação diferencial ordinária (EDO), avançando as técnicas de redes neurais residuais (ResNets). Discutimos a motivação para esta implementação e fornecemos uma fundamentação teórica para seu treinamento, que utiliza o método da adjunta e solvers de EDO. Apresentamos comparações de eficiência e ilustramos as vantagens das ODENets em relação às ResNets na mitigação do problema do vanishing gradient em redes profundas.

1 Introdução

O seguinte trabalho discute uma nova abordagem de rede neural desenvolvida como continuação direta das técnicas de redes neurais residuais (ResNets), onde consideramos o caso limite em que ela se comporta como uma EDO. Serão discutidas a motivação para a implementação desse modelo, bem como a fundamentação teórica do seu treinamento, que é baseado no método da adjunta, onde treinamos nossa rede utilizando solvers de EDO.

2 ResNets

As redes neurais residuais, uu ResNets, são uma arquitetura de redes neurais projetadas para solucionar o problema do vanishing gradient, onde o aumento da profundidade das redes neurais acaba levando a grande dificuldade de treinamento, ao aumentar significativamente o número de camadas de uma rede o gradiente calculado para fazer a atualização dos parâmetros das camadas iniciais acaba sendo muito pequeno e há uma atualização insignificante nesses parâmetros dificultando o processo de aprendizado dessa rede.

A forma em que as ResNets lidam com esse problema é introduzindo blocos residuais, que permitem a criação de atalhos (ou skip connections) entre camadas da rede. Esses atalhos adicionam a entrada da camada diretamente à sua saída após passar por algumas operações, formando uma "conexão residual". O bloco residual pode ser representado pela fórmula $y = F(x, \{W_i\}) + x$, onde x é a entrada e F é a função que essa camada aprende.

Essa estrutura permite que a rede aprenda correções (ou resíduos) para a identidade da entrada, em vez de tentar aprender a transformação completa de uma vez. Isso facilita o treinamento de redes muito profundas, pois os gradientes podem fluir diretamente através dos atalhos durante a backpropagation, mitigando o problema do vanishing gradient.

Com essa técnica, na ImageNet competiton de 2015, ResNets tiveram a melhor acurácia na competição, eles conseguiram treinar redes com uma quantidade muito maior de camadas sem ter que se precupar com o problema dos gradientes.

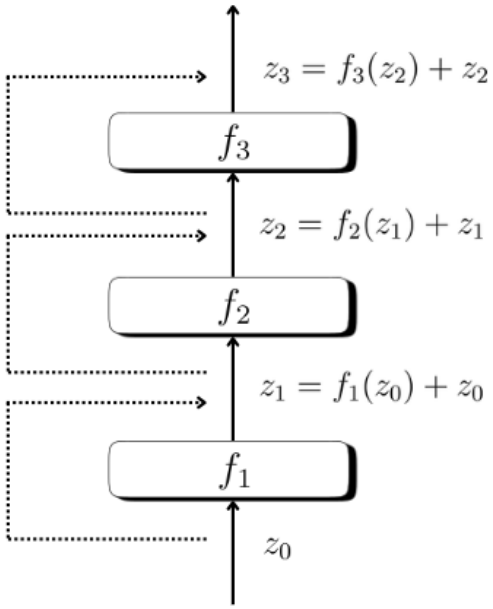


Figure 1: Ilustração de uma ResNet

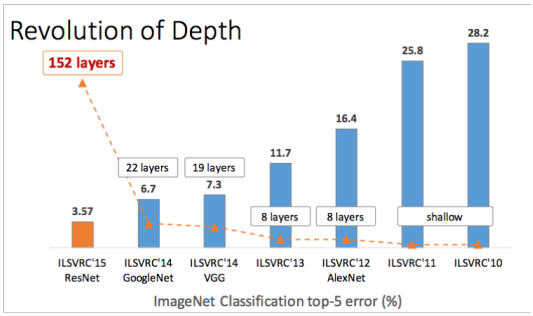


Figure 2: Comparativo de eficiência entre redes neurais

3 ODENet

De maneira resumida, nas redes residuais aprendemos transformações complicadas compondo uma sequência de transformações a partir de um estado inicial, ou seja $h_{t+1} = h_t + f(h_t, \theta_t)$, onde t varia em alguma discretização que fizemos. Essa maneira de atualizar discretamente a partir do estado inicial é análogo ao método de euler para solução de EDOs. Esse método é caracterizado como tendo um problema de valor inicial e sua EDO associada, ou seja, temos

$$\begin{cases} h'(t) = f(t, h(t)), \\ h(t_0) = h_0, \end{cases}$$

para descobrir o valor de h em um determinado t_*, escolhemos um tamanho de passo Δt, e definimos a sequência, $t_0, t_1, t_2, ..., t_*$, de modo que a diferença entre cada um desse satisfaz $t_{i+1} - t_i = \Delta t$, e daí, calculamos a sequência de imagens utilziando a EDO, com

$$h(t_{i+1}) = h(t_i) + \Delta t f(t_i, h(t_i))$$

E começando no valor h_0 que é dado descobrimos o valor em t_* para qualquer t_*.

Baseado nisso ODENets são redes neurais que não discretizam o tempo, ao invés de aprender a função cujos passos discretos se adequam ao que estamos modelando, queremos aprender a função

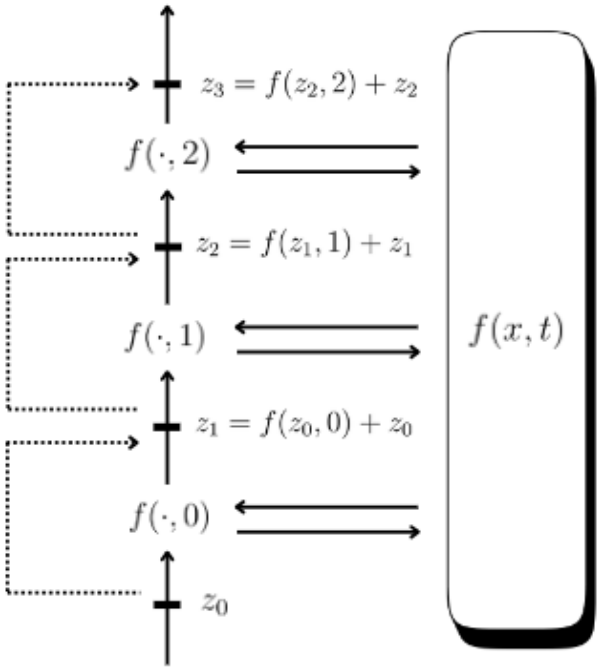
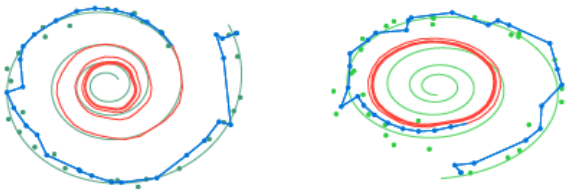


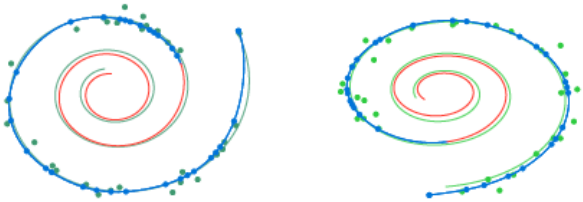
Figure 3: Ilustração de uma ODENet

cuja dinâmica contínua melhor se adequa ao que estamos modelando, se temos os pontos de dado $\{(y_1, t_1), (y_2, t_2), ..., (y_n, t_n)\}$, definimos a função h a partir de sua EDO:

$$\frac{dz}{dt} = f(z(t), t, \theta)$$



(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation

Para essa f colocamos uma rede neural padrão(um MLP, ou uma rede convolucional) cujos parâmetros estão em θ. Tendo essa estrutura, podemos calcular o valor de z em qualquer instante do tempo usando um solver de EDO, que recebe a função f, um valor inicial, e o tempo onde queremos a função avaliada, ou seja, para avaliar z no tempo t_*, podemos utilizar qualquer solver de EDO e

chamar

$$z(t_*) = ODESolve(z(t_0), f, t_0, t_*, \theta),$$

sabendo avaliar z em qualquer ponto, podemos calcular a perda($L(z)$) associada a essa z com relação aos nossos pontos de dado $\{(y_1, t_1), (y_2, t_2), ..., (y_n, t_n)\}$, tomando, por exemplo o MSE, avaliando z nos tempos que temos o valor da função, ou alguma outra métrica.

Com isso, para treinar o modelo queremos minimizar a função L com relação aos parâmetros(θ). A dependência entre L e z se dá a depender da função de perda que escolhermos, então será algo razoável, porém a forma que z depende de θ é atráves de um solver de EDO, que recebe a f e essa é uma rede neural caracterizada pelos parâmetros θ. O desafio no treinamente é encontrar o gradiente de algo que depende dos parâmetros passando por um solver de EDO.

Para calcular esse gradiente de forma otimizada é utilizado o método da adjunta, que ao invés de calcular de forma explicita usando as funções, o descreve a partir de outra EDO , e podemos avaliar essa função com outra função chamada para um solver.

4 O método da adjunta

Como visto, só aprendemos a EDO associada a nossa função para avaliar ela e por conseguinte para calcular a perda utilizamos um solver de EDO. Queremos treinar nossa rede usualmente utilizando descida de gradiente nos parâmetros com relação a perda, isso envolve computar $\nabla_\theta L = \frac{\partial L}{\partial \theta}$, mas nosso L depende de um solver de EDO, então é possível diferenciar a a partir do foward pass. Entretanto, como ele é um método de solução de EDO, fazer dessa maneira pode resultar em um custo de memória muito grande e também introduzir uma grande parcela de erro numérico. Para que possamos usar backpropagation através disso de forma eficiente é utilizado o método da adjunta, no qual esse gradiente não é calculado de forma explitica, mas também é encontrado a partir de uma EDO.

$z(t)$ segue a EDO $\dfrac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t, \theta)$, θ são os parâmetros. Definimos o estado adjunto como

$$a(t) = \frac{\mathrm{d}L}{\mathrm{d}z(t)}$$

Dessa definição conseguimos a equação diferencial

$$\frac{\mathrm{d}a(t)}{\mathrm{d}t} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z(t)},$$

que nos permite avaliar $a(t)$ onde quisermos, já que aprendemos a função f que depende de z como uma rede neural, a partir de $a(t)$ podemos finalmente calcular $\frac{\partial L}{\partial \theta}$, a partir de

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt$$

Já que para $a(t)$ usamos um solver de EDO na equação que apresentamos anteriormente, e para $\frac{\partial f}{\partial \theta}$ sabemos como calcular já que θ são os parametros da rede neural que colocaremos em f. Como calcular esse gradiente envolveria avaliar essa integral, mas o que estamos integrando depende de $a(t)$ que envolve resolver a outra EDO, a maneira mais eficiente de encontrar esse gradiente é resolver uma EDO aumentada e encontramos esse gradiente chamando um solver de EDO nela, daí o algoritmo pra encontrar o gradiente fica:

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

Figure 4: algoritmo pra encontrar$\partial L/\partial \theta$

4.1 Prova do método da adjunta

Como $z(t)$ segue $\dfrac{\mathrm{d}z(t)}{\mathrm{d}t} = f(z(t), t, \theta)$, temos que

$$z(t+\epsilon) = z(t) + \int_t^{t+\epsilon} f(z(t), t, \theta)dt = T_\epsilon(z(t), t)$$

A partir disso, pela regra da cadeia temos que

$$\frac{\mathrm{d}L}{\mathrm{d}z(t)} = \frac{\mathrm{d}L}{\mathrm{d}z(t+\epsilon)}\frac{\mathrm{d}z(t+\epsilon)}{\mathrm{d}z(t)}$$

Ou seja,

$$a(t) = a(t+\epsilon)\frac{\partial T_\epsilon(z(t), t)}{\partial z(t)}$$

Daí podemos fazer o cálculo de $\dfrac{\mathrm{d}a(t)}{\mathrm{d}t}$:

$$\frac{\mathrm{d}a(t)}{\mathrm{d}t} = \lim_{\epsilon\to 0^+}\frac{a(t+\epsilon) - a(t)}{\epsilon} = \lim_{\epsilon\to 0^+}\frac{a(t+\epsilon - a(t+\epsilon)\frac{\partial}{\partial z(t)}T_\epsilon(z(t), t)}{\epsilon}$$

Agora, usando a expansão de Taylor de T_ϵ ao redor de $z(t)$:

$$= \lim_{\epsilon\to 0^+}\frac{a(t+\epsilon) - a(t+\epsilon)\frac{\partial}{\partial z(t)}(z(t) + \epsilon f(z(t), t, \theta) + O(\epsilon^2))}{\epsilon}$$

$$= \lim_{\epsilon\to 0^+}\frac{a(t+\epsilon) - a(t+\epsilon)(I + \epsilon\frac{\partial}{\partial z(t)}f(z(t), t, \theta) + O(\epsilon^2))}{\epsilon}$$

$$= \lim_{\epsilon\to 0^+}\frac{-\epsilon a(t+\epsilon)\frac{\partial}{\partial z(t)}f(z(t), t, \theta) + O(\epsilon^2)}{\epsilon}$$

$$= \lim_{\epsilon\to 0^+} -a(t+\epsilon)\frac{\partial}{\partial z(t)}f(z(t), t, \theta) + O(\epsilon)$$

$$\frac{\mathrm{d}a(t)}{\mathrm{d}t} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z(t)}$$

Como queríamos, Agora, queremos computar $\dfrac{\mathrm{d}L}{\mathrm{d}\theta}$ usando a EDO adjunta e não de forma direta, primeiro definimos a EDO aumentada(augmented), notando que podemos olhar tanto pra θ quanto pra t como funções de t, sendo a θ constante com relação ao tempo e a t a identidade, daí temos

$$\frac{\partial\theta(t)}{\partial t} = 0 \text{ e } \frac{\partial \mathbf{t}(\mathbf{t})}{\partial t} = 1$$

Com isso temos a EDO aumentada:

$$\frac{d}{dt}\begin{bmatrix} z \\ \theta \\ t \end{bmatrix}(t) = \begin{bmatrix} f([z, \theta, t]) \\ 0 \\ 1 \end{bmatrix} := f_{aug}([z, \theta, t])$$

Definindo agora, $a_\theta = \dfrac{\mathrm{d}L}{\mathrm{d}\theta(t)}$ e $a_t = \dfrac{\mathrm{d}L}{\mathrm{d}\mathbf{t}(\mathbf{t})}$ temos a adjunta aumentada

$$a_{aug} := [a \quad a_\theta \quad a_t]$$

Agora, podemos olhar pra jacobiana da EDO aumentada, e como antes, ela se relacionará com a darivada da adjunta. Temos

$$\frac{\partial f_{aug}}{\partial[z, \theta, t]} = \begin{bmatrix} \frac{\mathrm{d}f}{\mathrm{d}z} & \frac{\mathrm{d}f}{\mathrm{d}\theta} & \frac{\mathrm{d}f}{\mathrm{d}t} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Agora queremos olhar pra derivada da adjunta aumentada com relação ao tempo, temos que

$$\frac{\mathrm{d}a_{aug}(t)}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}a}{\mathrm{d}t} & \frac{\mathrm{d}a_\theta}{\mathrm{d}t} & \frac{\mathrm{d}a_t}{\mathrm{d}t} \end{bmatrix}(t)$$

Mas como provamos anteriormente, $\frac{\mathrm{d}a}{\mathrm{d}t} = -a\frac{\partial f}{\partial z}$, de forma análoga encontramos que $\frac{\mathrm{d}a_\theta}{\mathrm{d}t} = -a(t)\frac{\partial f}{\partial \theta}$ e $\frac{\mathrm{d}a_t}{\mathrm{d}t} = -a\frac{\partial f}{\partial t}$ Logo, temos que

$$\frac{\mathrm{d}a_{aug}(t)}{\mathrm{d}t} = \begin{bmatrix} -a\frac{\partial f}{\partial z} & -a\frac{\partial f}{\partial \theta} & -a\frac{\partial f}{\partial t} \end{bmatrix}(t) = \begin{bmatrix} a & a_\theta & a_t \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f}{\mathrm{d}z} & \frac{\mathrm{d}f}{\mathrm{d}\theta} & \frac{\mathrm{d}f}{\mathrm{d}t} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Portanto,

$$\frac{\mathrm{d}a_{aug}}{\mathrm{d}t} = a_{aug}\frac{\partial f_{aug}}{\partial [z, \theta, t]}$$

Então temos uma EDO para a adjunta aumentada, logo podemos resolvê-la para encontrar seu valor em qualquer ponto, e logo conseguimos determinar $a_\theta = \frac{\mathrm{d}L}{\mathrm{d}\theta(t)}$, usando um solver de EDO na adjunta aumentada, e se conseguimos calcular a derivada da perda com relação aos parâmetros podemos fazer descida de gradiente.

5 Implementação

Implementamos esse modelo aos datasets vistos ao longo do semestre nas listas de exercícios. Disponibilizamos ainda uma tabela comparativa entre as acurácias dos modelos implementados como exercícios e a acurácia alcançada pelo treinamento da ODENet. Para as tarefas de classificação de imagens, o processo envolve inicialmente o downsample dos dados iniciais para um estado latente associado usando redes convolucionais, que reduzem a dimensão da saída e geram vários estados latentes. Em seguida, esses estados latentes evoluem conforme a dinâmica aprendida, seguido pela aplicação de uma camada fully connected para produzir a saída adequada, que retorna um vetor com as 10 classes no caso do MNIST e CIFAR. Para conjuntos de dados como California Housing e Breast Cancer, onde as entradas não são imagens, o downsample é mais simples, geralmente consistindo em uma rede de duas camadas. No entanto, o esquema é o mesmo: evoluir o estado latente usando a dinâmica aprendida e, finalmente, aplicar uma camada fully connected que retorna a predição, sendo um valor único para o Breast Cancer (classificação binária) e uma regressão para o California Housing. Os resultado foram os seguintes:

Dataset	Modelo	Resultado do exercício	Resultado ODENet
MNIST	KMEANS	accuracy 99.11	accuracy 99.17
BREAST CANCER	EMP	accuracy 94	accuracy 94
CALIFORNIA HOUSING	ridge regression	mse 0.536	mse 0.009
CIFAR10	CNN	accuracy 64.2	accuracy 71.67

Table 1: Comparativo de desempenho entre ODENet e outros modelos

Observamos que, em relação às métricas de acurácia para problemas de classificação e à métrica de MSE para problemas de regressão, em comparação aos modelos utilizados nas listas o desempenho foi satisfatório

6 Conclusão

As ODENets representam uma evolução lógica a partir das redes residuais, herdando a propriedade de contornar o problema do gradiente que desaparece. A principal contribuição do artigo é apresentar uma forma eficiente de treinar esse modelo usando o método adjunto, permitindo o cálculo dos gradientes necessários independentemente do solver de EDO usado para o forward pass na rede. Também foi demonstrado como essa arquitetura pode ser aplicada a problemas de classificação e

regressão, alcançando desempenhos comparáveis aos modelos estudados ao longo do semestre. Em suma, há ganho em termos de memória e eficiência, já que aprendemos uma única função para a dinâmica, em vez de precisar aprender a função completa, o que exigiria mais camadas e tornaria o treinamento desproporcionalmente difícil. No entanto, o trade-off é que é necessário resolver uma EDO sempre que quisermos avaliar nossa função, e o processo de treinamento se complexifica.

References

[1] Ricky T. Q. Chen & Yulia Rubanova & Jesse Bettencourt & David Duvenaud (2018) Neural Ordinary Differential Equations. *Advances in Neural Information Processing Systems* **vol. 31**

[2] Neural Ordinary Differential Equations. Direção de Andriy Drozdyuk. S.I., 2020. PB. Disponível em: https://www.youtube.com/watch?v=uPd0B0WhH5w. Acesso em: 18 jun. 2024.

[3] Residual Networks and Skip Connections (DL 15). S.I., 2022. PB. Disponível em: https://www.youtube.com/watch?v=Q1JCrG1bJ-A. Acesso em: 18 jun. 2024.

[4] Chen, Ricky. Torchdiffeq. Disponível em: https://github.com/rtqichen/torchdiffeq. Acesso em: 18 jun. 2024.

[5] Surtsukov, Mikhail. Neural Ordinary Differential Equations. Disponível em: https://msurtsukov.github.io/Neural-ODE/. Acesso em: 18 jun. 2024.