Dynamic System and Optimal Control Perspective of Deep Learning(Part I)

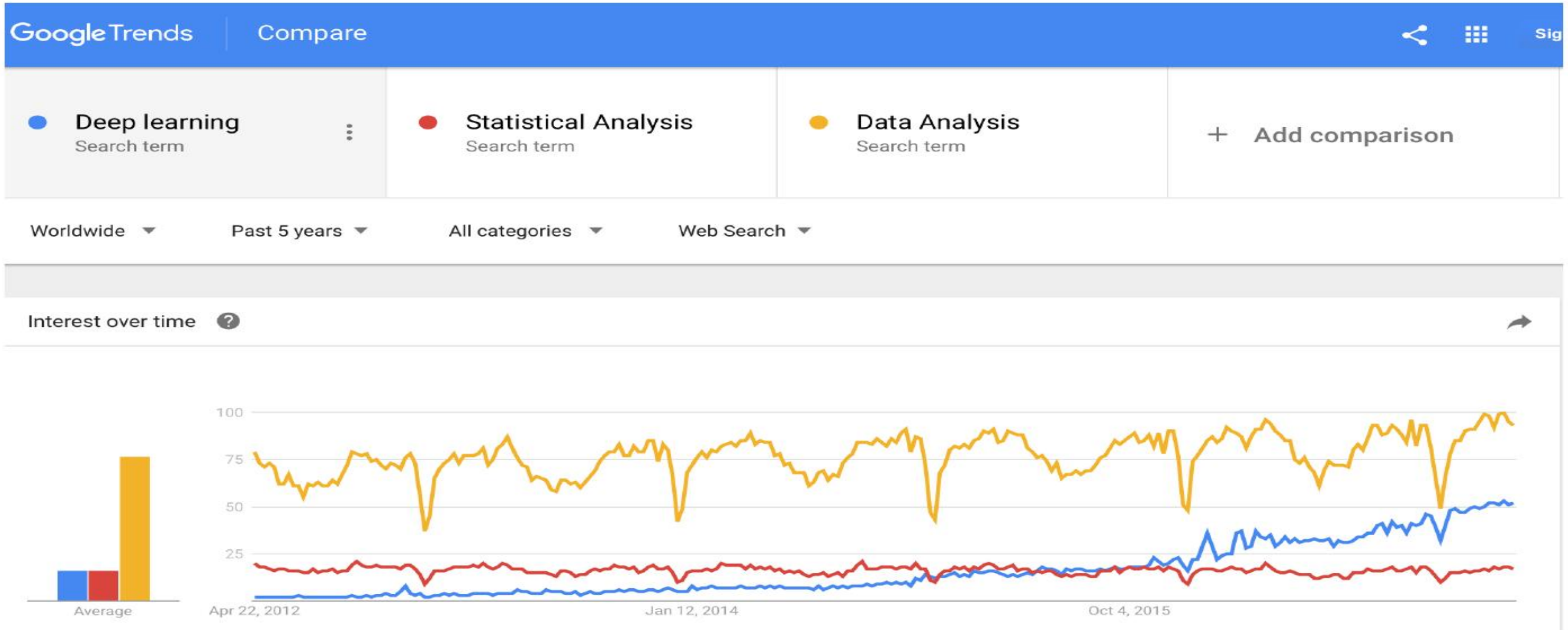
Tijin Yan

2020.07.11

Outline

- **Background & Motivation**
- **DNN & Numerical ODE**
- **Continuous ODE & Its derivatives**
- **RNN&LSTM**
- **DNN & Numerical PDE**
- **Deep Network Training**
- **Optimization Algorithms**

Background & Motivation



ReLU(@jmlr12)
AlexNet(@nips13)
Word2Vec(@nips14)
GAN(@nips15)
Adam(@iclr15)
Attention(@iclr15)
ResNet(@cvpr16)
AlphaGo(@nature16)
Transformer(@nips17)
NeuralODE(@icml8)
BERT(@naacl19)

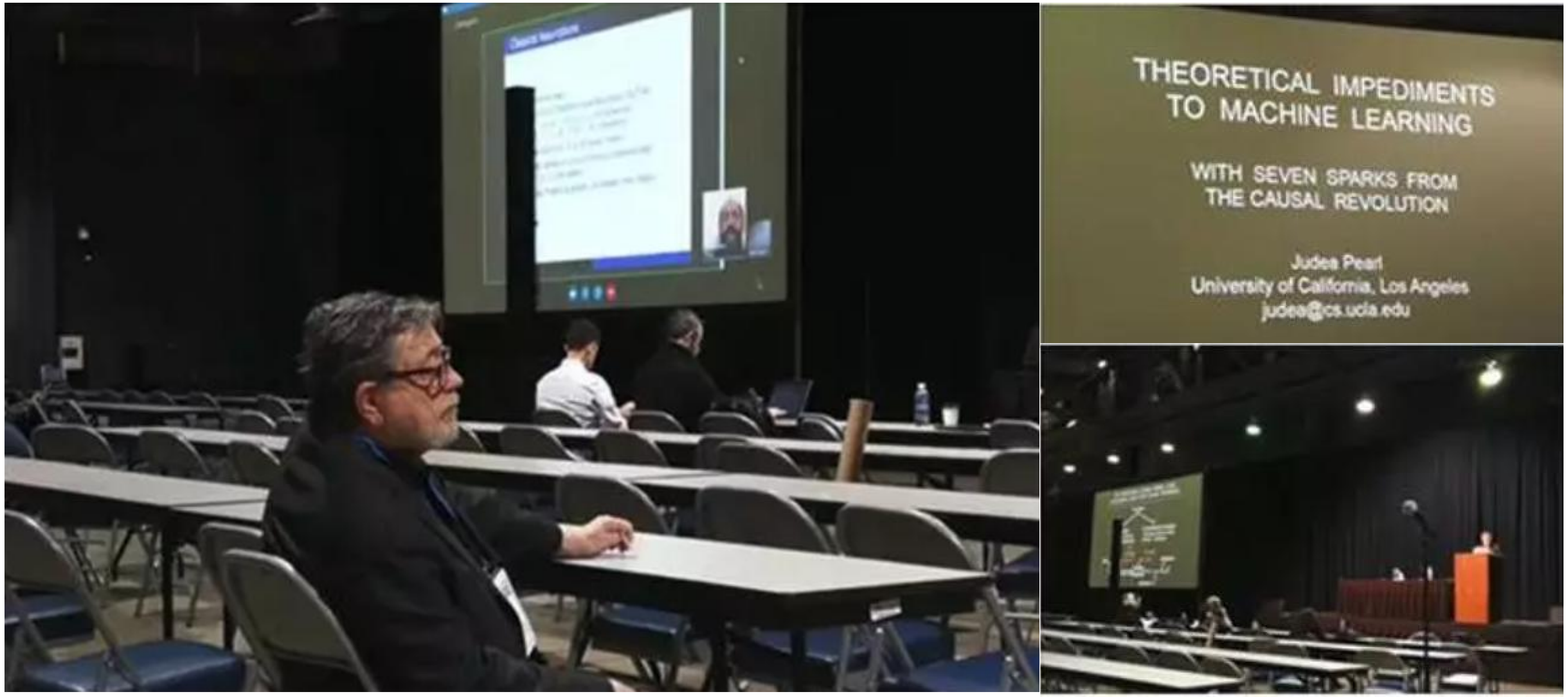
Deep Concerns

Deep learning is “alchemy” .

-- Ali Rahimi, NIPS 2017

Being a alchemy is certainly not a shame, not wanting to work on advancing to chemistry is a shame!

-- Eric Xing, NIPS 2017



Deep Learning from Dynamics Perspective

What are still challenging

- Theoretical guidance
- Transparency, interpretability, robustness

How to provide guidance and transparency to deep learning?

- Find “frameworks” and “links” with applied mathematics

Deep Network	$\Longleftrightarrow$	**Differential Equations (DE)**
Network Architecture	$\Longleftrightarrow$	**Numerical DE** **Optimization Algorithm**
Network Training	$\Longleftrightarrow$	**Optimal Control**

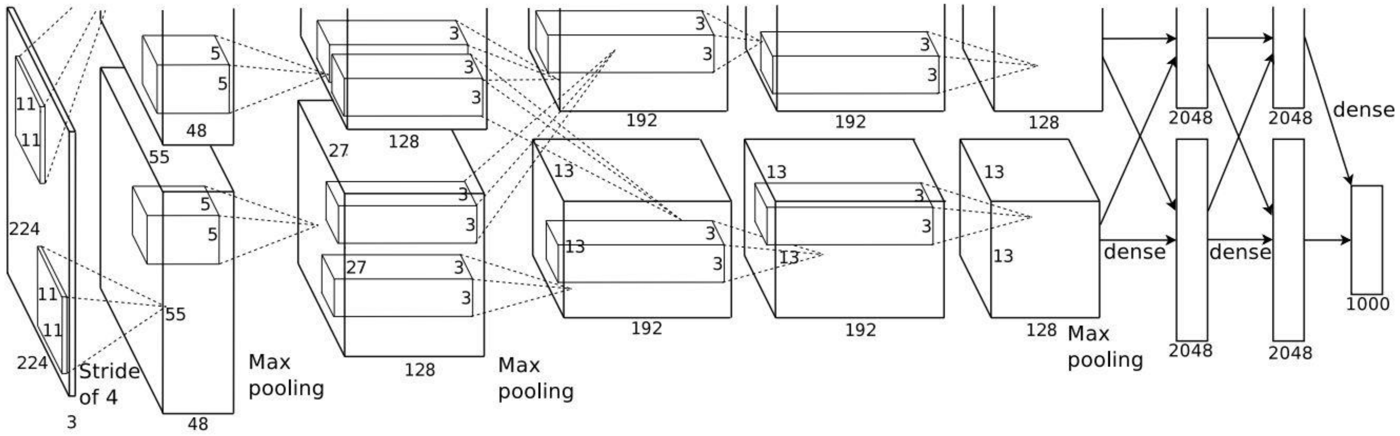
DNN and Numerical ODE

Depth Neural Network

Deep Neural Network

$$f_1\left(f_2\left(f_3\cdots(x)\right)\right)$$

A Dynamic System?



$$\tilde{f}_{L,\boldsymbol{N}}(\boldsymbol{x};\boldsymbol{\Theta}) : \mathbb{R}^n \mapsto \mathbb{R},$$

can be recursively defined as: $\boldsymbol{\Theta}^\ell = (\boldsymbol{\Theta}^{\ell-1}, \theta^\ell)$, $\tilde{f}_{\boldsymbol{\Theta}^\ell} = (\theta^\ell \circ \sigma \circ \tilde{f}_{\boldsymbol{\Theta}^{\ell-1}})$, $\theta^\ell : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_{\ell+1}}$ with $\theta^\ell(\boldsymbol{x}) = \boldsymbol{W}^\ell \boldsymbol{x} + \boldsymbol{b}^\ell$, and $\tilde{f}_{L,\boldsymbol{N}} := \tilde{f}_{\boldsymbol{\Theta}^L}$.

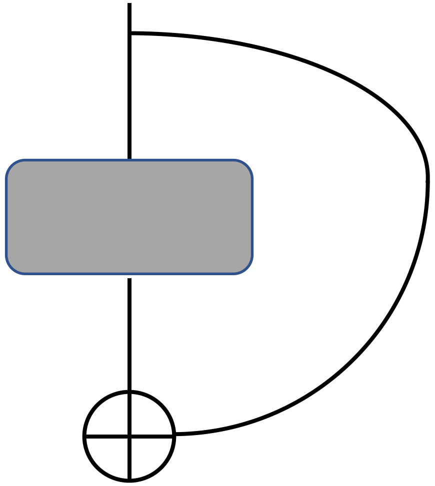
Preliminary

$$\begin{cases} \dfrac{dx(t)}{dt} = f(t, x(t)) \\ x(t_0) = x_0 \end{cases}$$

Methods	Formula	Truncation Error
Forward Euler Scheme	$x_{n+1} = x_n + hf(x_n, t_n)$	**O(h²)**
Backward Euler Scheme	$x_{n+1} = x_n + hf(x_{n+1}, t_{n+1})$	**O(h²)**
Two-points Euler	$x_{n+1} = x_{n-1} + 2hf(x_n, t_n)$	**O(h²)**
Trapezoidal Formula	$x_{n+1} = x_n + \frac{h}{2}[f_n + f_{n+1}]$	**O(h³)**
Runga-Kutta Formula(2Order)	$\begin{cases} \hat{x}_{n+1} = x_n + hf(x_n, t_n) \\ x_{n+1} = x_n + \dfrac{h}{2}[f(x_n, t_n), f(\hat{x}_{n+1} t_{n+1})] \end{cases}$	**O(h³)**
Linear Multi-step	$x_{i+1} = \sum_{k=0}^{K} \alpha_k x_{i-k} + \mathrm{h}\sum_{k=-1}^{K-1} \beta_k f_{i-k}$	

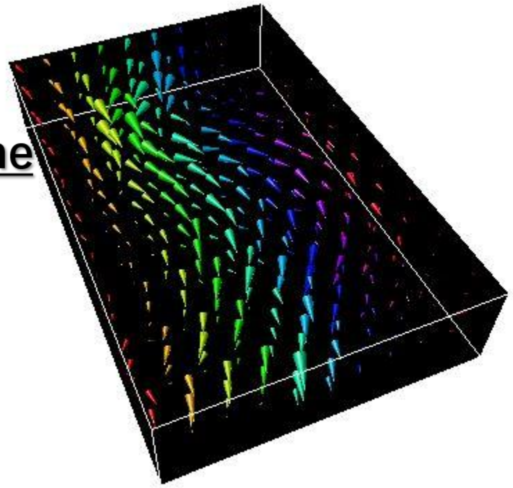
https://zhuanlan.zhihu.com/p/70255604

Deep Residual Learning(@CVPR2016)



$$x_{n+1} = x_n + f(x_n,\ t_n)$$

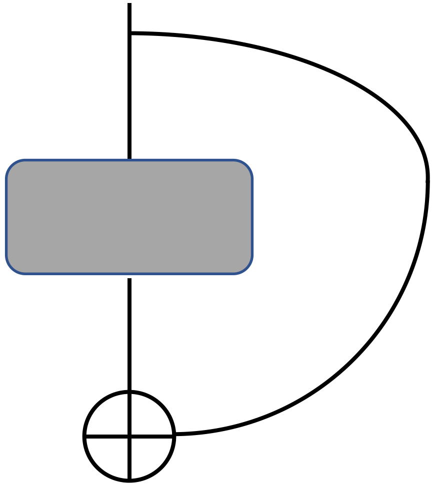
$$x_t = f(x, t)$$



<u>Forward Euler Scheme</u>

- Weinan E. A Proposal on Machine Learning via Dynamical Systems. Communications in Mathematical Science, 2017.
- Haber E, Ruthotto L. Stable architectures for deep neural networks[J]. Inverse Problems, 2017.
- Bo C, Meng L, et al. Reversible Architectures for Arbitrarily Deep Residual Neural Networks, AAAI 2018
- Lu Y. et al., Beyond Finite Layer Neural Network: Bridging Deep Architects and Numerical Differential Equations, ICML 2018.

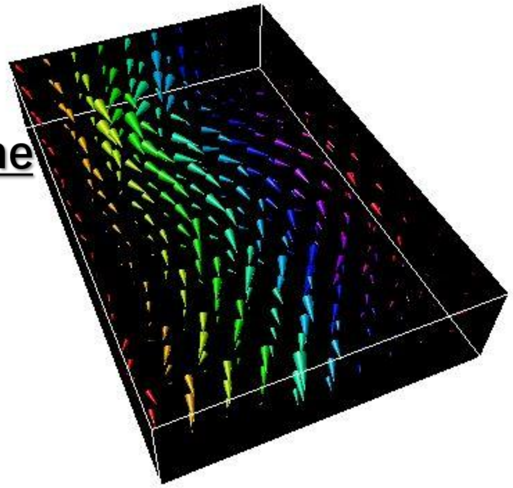
Deep Residual Learning(@CVPR2016)



$$x_{n+1} = x_n + f(x_n,\ t_n)$$

Forward Euler Scheme

$$x_t = f(x, t)$$



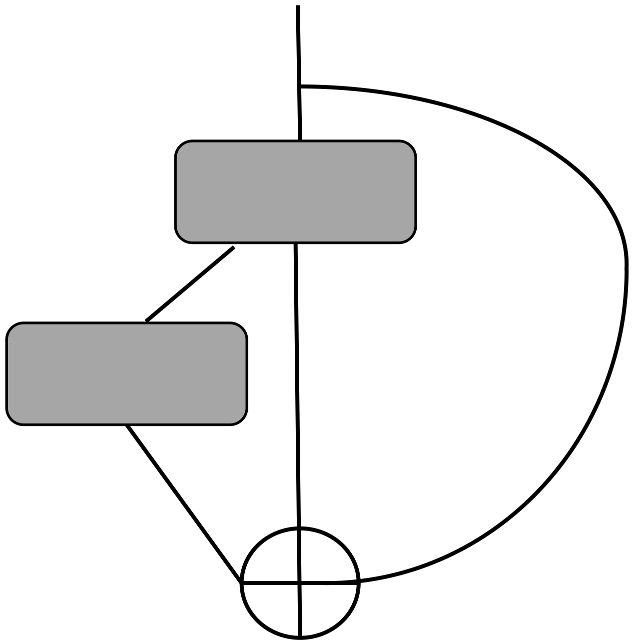
Theoretical results:

Thorpe, Matthew, and Yves van Gennip. "Deep Limits of Residual Neural Networks." ***arXiv preprint arXiv:1810.11741*****(2018).**

A mean-field control perspective:

E, Weinan, Han, Jiequn, and Qianxiao Li. "A mean-field optimal control formulation of deep learning." Research in the Mathematical Sciences, vol. 6, no. 10, pp. 1–41, 2019. (arXiv:1807.01083).

PolyNet(@CVPR2017)



Revisiting previous efforts in deep learning, we found that diversity, another aspect in network design that is relatively less explored, also plays a significant role.

PolyStrure: $\boldsymbol{x_{n+1} = x_n + F(x_n) + F(F(x_n))}$

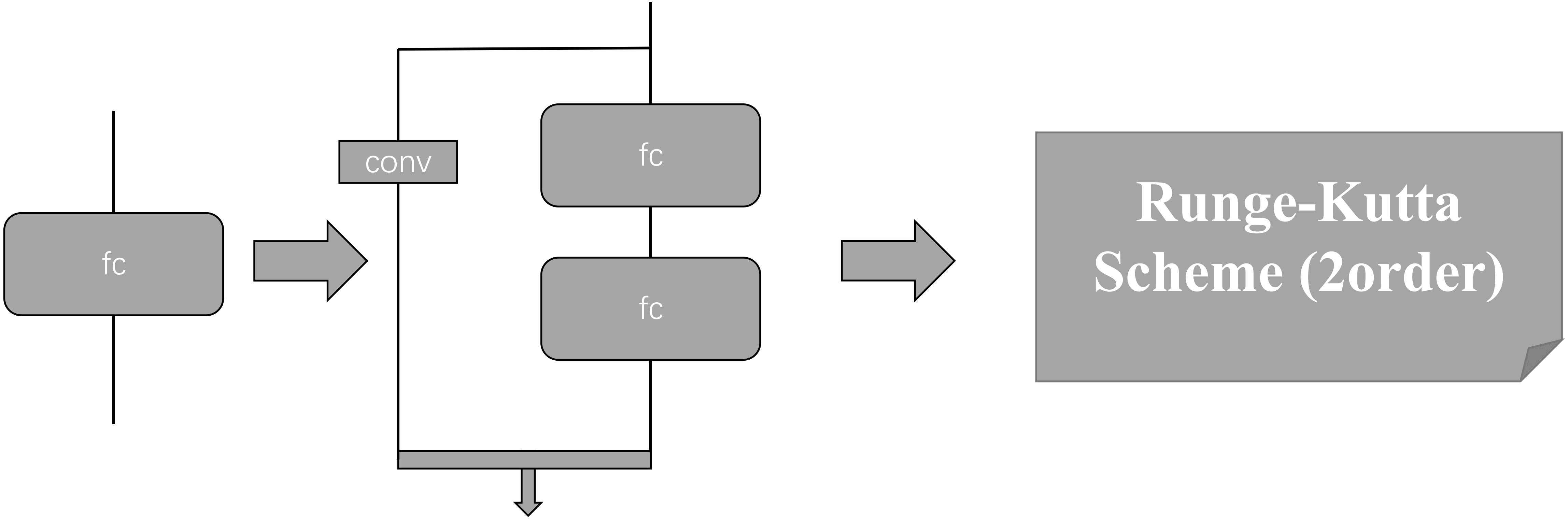
Backward Euler Scheme:

$$x_{n+1} = x_n + F(x_{n+1}) \Rightarrow x_{n+1} = (I - F)^{-1} x_n$$

Approximate the operator $(I - F)^{-1}$ by $I + F + F^2 + \cdots$

Zhang X, Li Z, Loy C C, et al. PolyNet: A Pursuit of Structural Diversity in Very Deep Networks. CVPR 2017

FractalNet(@ICLR2017)



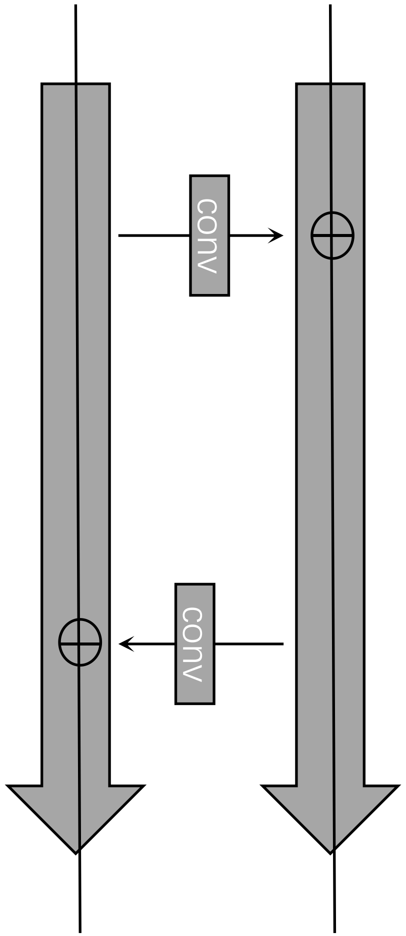
f_0

$$\boldsymbol{f_{c+1} = \frac{1}{2}k_c * + \frac{1}{2}f_c \circ f_c}$$

$$x_{n+1} = k_1 * x_n + k_2 * \left(k_3 * x_n + f_1(x_n)\right) + f_2\left(k_3 * x_n + f_1(x_n)\right)$$

Larsson G, Maire M, Shakhnarovich G. FractalNet: Ultra-Deep Neural Networks without Residuals. ICLR 2017.

RevNet(@NIPS2017)



$$x_{n+1} = x_n + f(y_n)$$
$$y_{n+1} = y_n + g(x_{n+1})$$

$$\Rightarrow$$

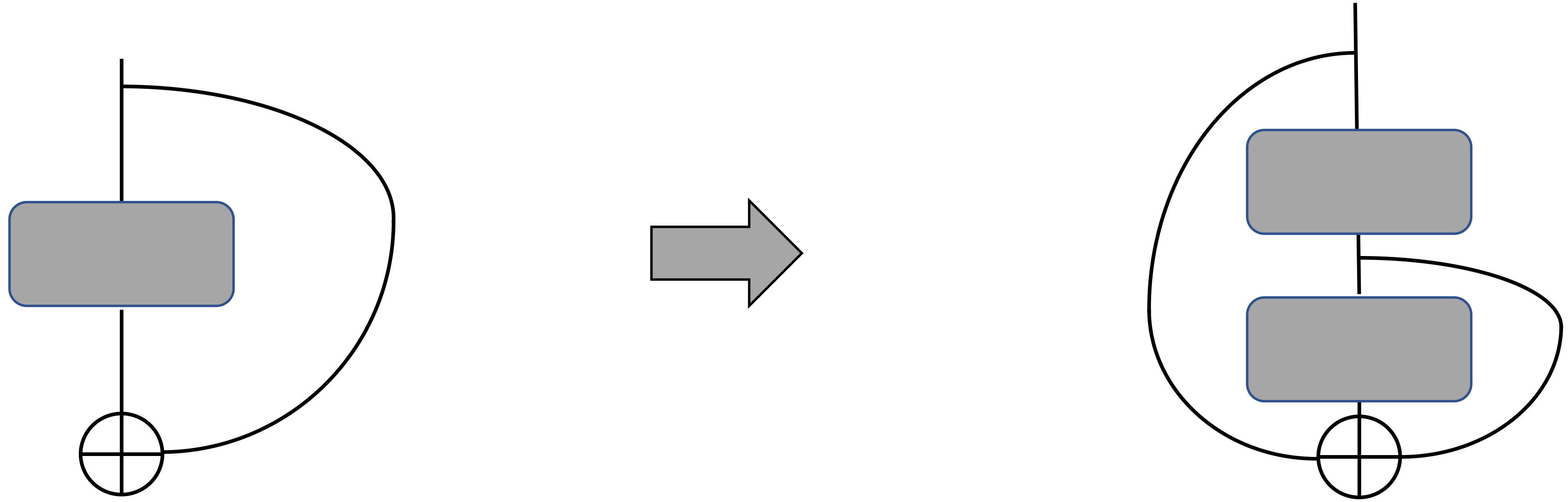
$$\dot{x} = f(y)$$
$$\dot{y} = g(x)$$

Forward Euler Scheme

Aidan N. Gomez et al. The reversible residual network: Backpropagation without storing activations. NIPS 2017.

LM-ResNet(@ICML2018)

Linear Multi-step Scheme

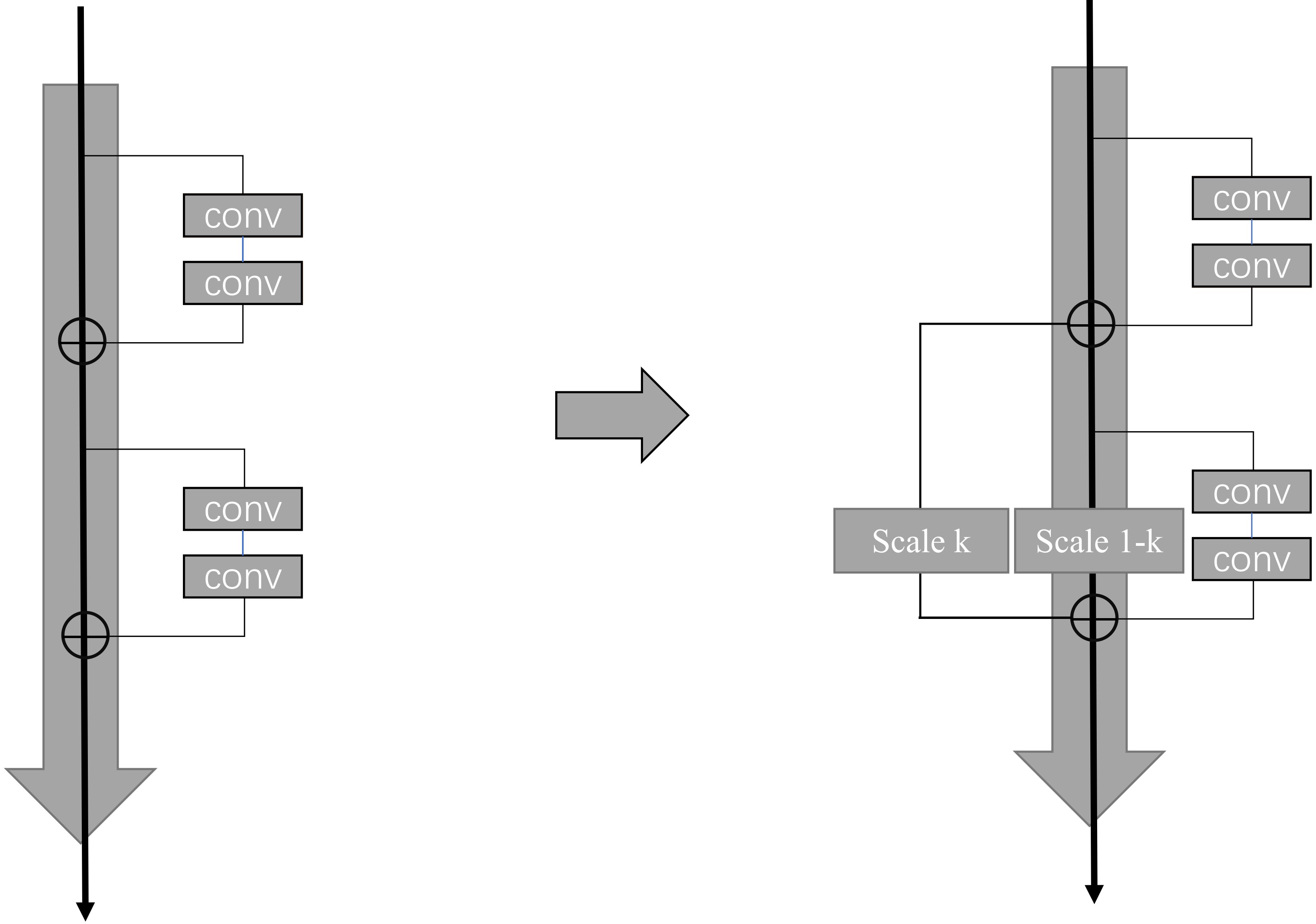


$$x_{n+1} = x_n + f(x_n, t_n)$$

$$x_{n+1} = (1 - k_n)x_n + k_n x_{n-1} + f(x_n, t_n)$$

Lu, Yiping, et al. "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations." ICML 2018

LM-ResNet(@ICML2018)



Lu, Yiping, et al. "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations." ICML 2018

LM-ResNet(@ICML2018)

ResNet

$$x_{n+1} = x_n + \Delta t f(x_n, t_n)$$

$$\dot{u} + \frac{\Delta t}{2}\ddot{u}_n = f(u, t)$$

LM-ResNet

$$x_{n+1} = (1 - k_n)x_n + k_n x_{n-1} + \Delta t f(x_n, t_n)$$

$$(1 + k_n)\,\dot{u} + (1 - k_n)\frac{\Delta t}{2}\ddot{u}_n = f(u, t)$$

[1] Dong B, Jiang Q, Shen Z. Image restoration: wavelet frame shrinkage, nonlinear evolution PDEs, and beyond. Multiscale Modeling and Simulation: A SIAM Interdisciplinary Journal, 15(1)，606-660, 2017.

[2] Su W, Boyd S, Candes E J. A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights. Advances in Neural Information Processing Systems, 2015.

[3] A. Wibisono, A. Wilson, and M. I. Jordan. A variational perspective on accelerated methods in optimizationProceedings of the National Academy of Sciences 2016.

LM-ResNet(@ICML2018)

Connection to Stochastic Dynamics

Shake-Shake regularization

$$x_{n+1} = x_n + \eta f_1(x_n) + (1-\eta) f_2(x_n), \eta \sim U[0,1]$$
$$= x_n + f_2(x_n) + \frac{1}{2}\big(f_1(x_n) - f_2(x_n)\big) + \boxed{(\eta - \frac{1}{2})\big(f_1(x_n) - f_2(x_n)\big)}$$

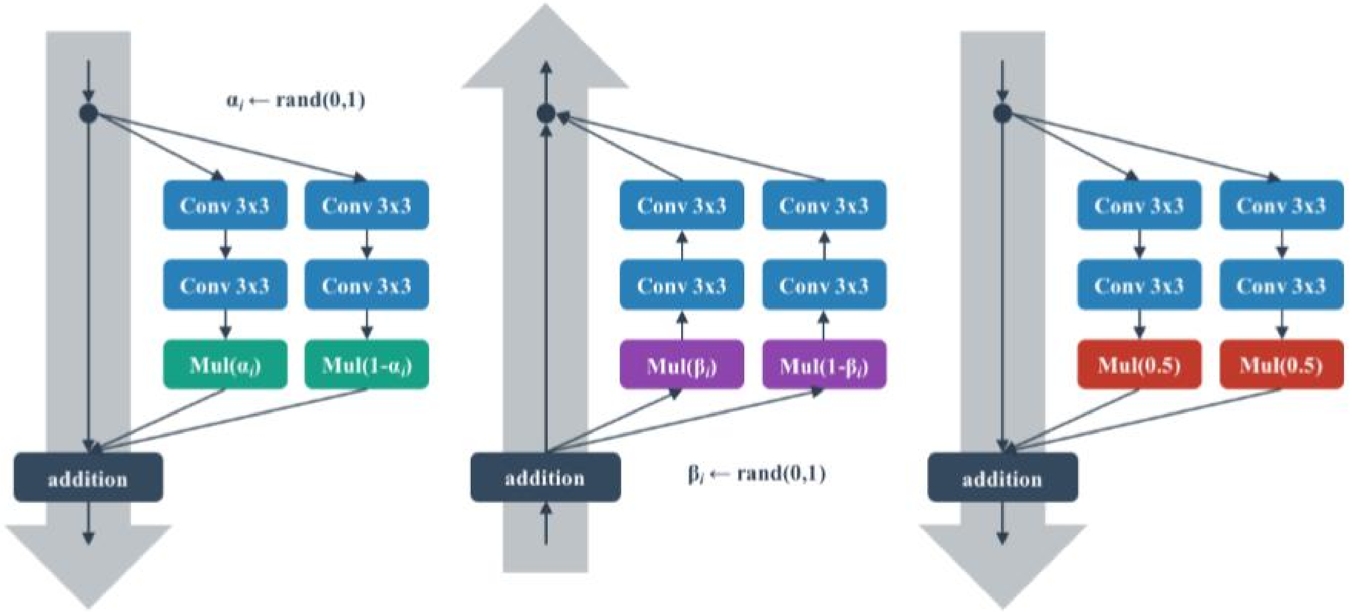


Figure 1: **Left:** Forward training pass. **Center:** Backward training pass. **Right:** At test time.

Gastaldi X. Shake-Shake regularization. ICLR Workshop Track2017.

$$\frac{1}{\sqrt{12}}(f_1(X) - f_2(X)) \odot [\mathbf{1}_{N\times 1}, \mathbf{0}_{N,N-1}]dB_t$$

$$\min \mathbb{E}_{X(0)\sim data}\left(\mathbb{E}(L(X(T)) + \int_0^T R(\theta))\right)$$
$$s.t.\ dX = f(X,\theta) + g(X,\theta)dB_t$$

Apply data augmentation techniques to internal representations.

Lu, Yiping, et al. "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations." ICML 2018

LM-ResNet(@ICML2018) Connection to Stochastic Dynamics

Stochastic Path

$$x_{n+1} = x_n + \eta_n f(x_n)$$
$$= x_n + E\eta_n f(x_n) + \boxed{(\eta_n - E\eta_n) f(x_n)}$$

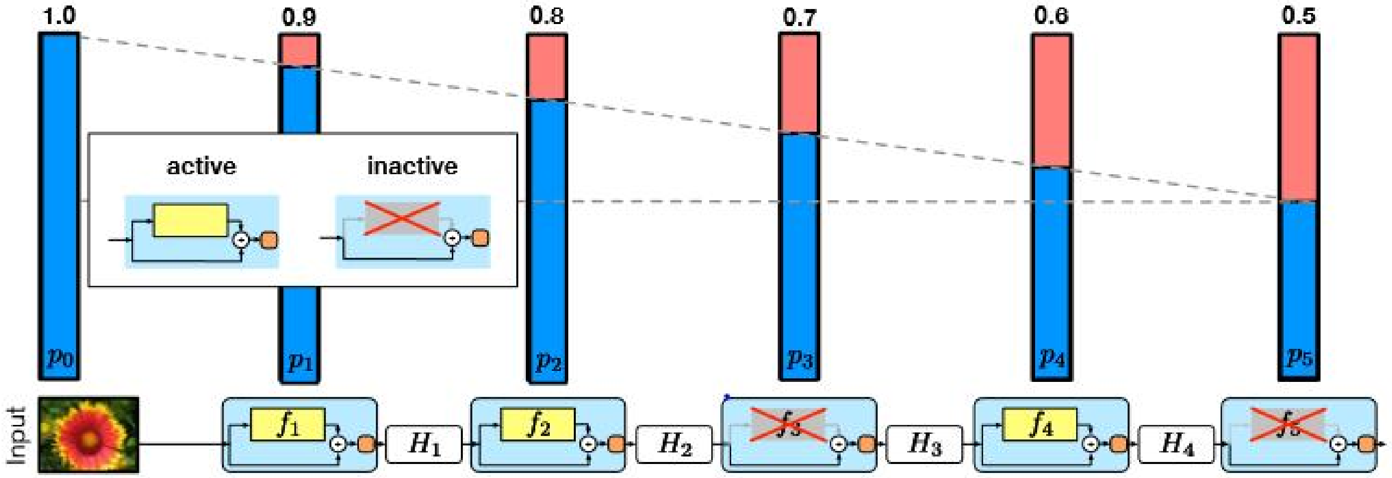


Fig. 2. The linear decay of p_ℓ illustrated on a ResNet with stochastic depth for $p_0 = 1$ and $p_L = 0.5$. Conceptually, we treat the input to the first ResBlock as H_0, which is always active.

$$\sqrt{p(t)(1 - p(t))} f(X) \odot [\mathbf{1}_{N\times 1}, \mathbf{0}_{N,N-1}] dB_t.$$

To reduce the effective length of a neural network during training, we randomly skip layers entirely.

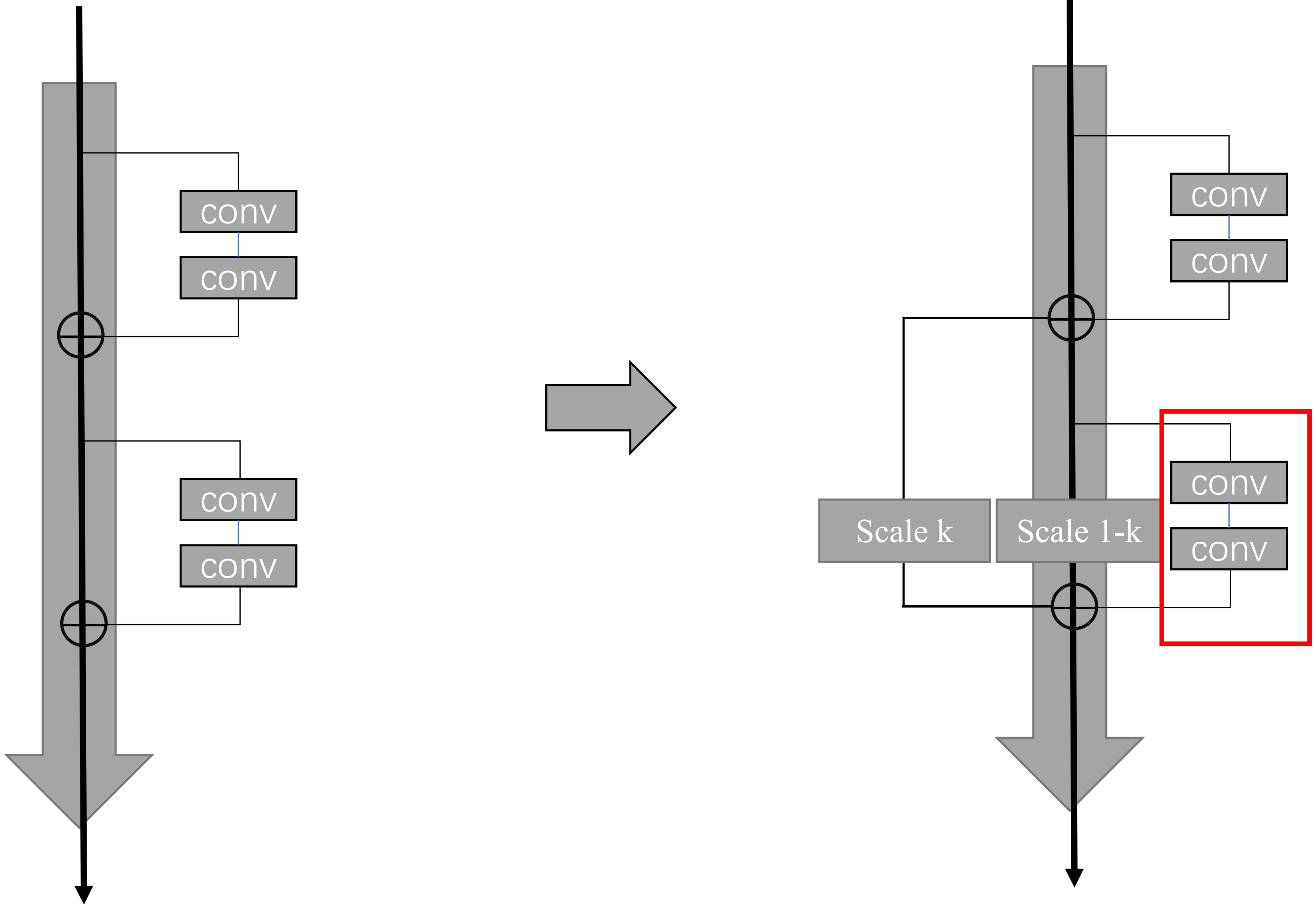
Huang G, Sun Y, Liu Z, et al. Deep Networks with Stochastic Depth ECCV2016.

Lu, Yiping, et al. "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations." ICML 2018

LM-ResNet(@ICML2018)

$$(1+k_n)\,\dot{u} + (1-k_n)\frac{\Delta t}{2}\ddot{u}_n + o(\Delta t^3) = f(u) + g(u)dW_t$$

$$\Delta W_t = B_{t_{n+1}} - B_{t_n}$$



Lu, Yiping, et al. "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations." ICML 2018

LM-ResNet(@ICML2018)

Neural Network ⟺ Dynamic System

Stochastic Learning ⟺ Stochastic Dynamic System

New Discretization

LM-ResNet

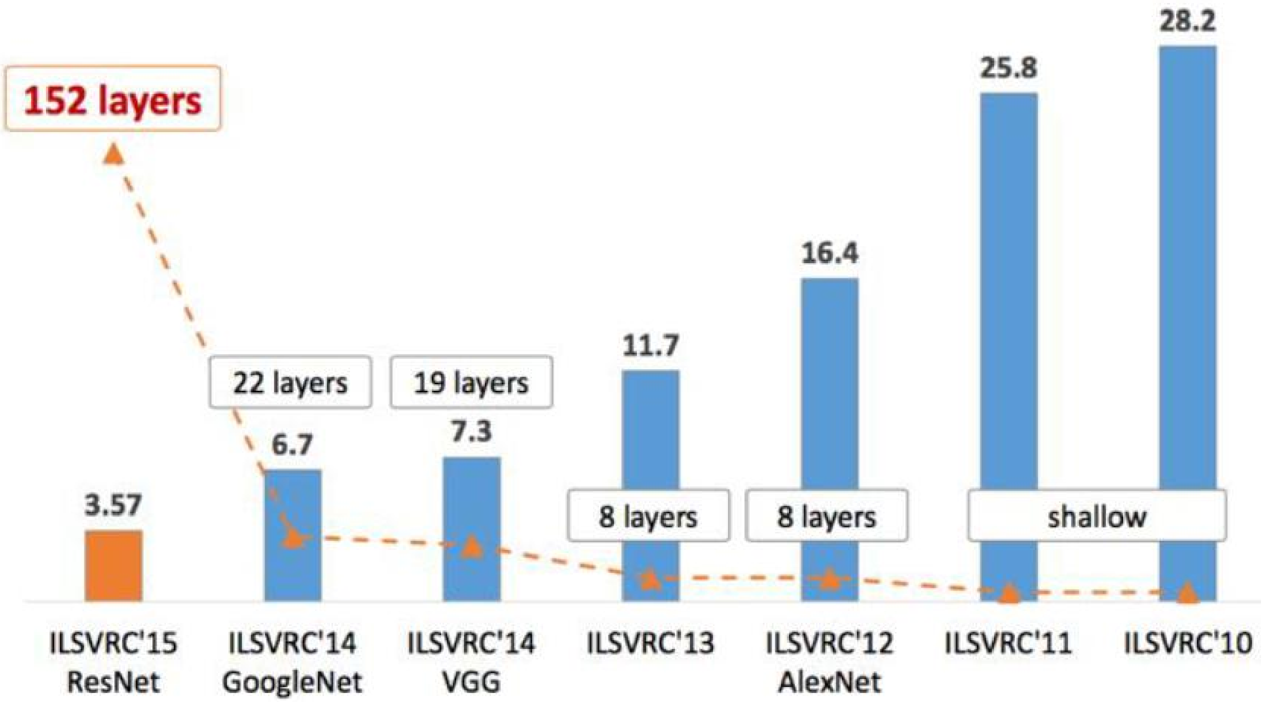
- **Original One:** LM-Resnet56 Beats Resnet110
- **Stochastic Depth One:** LM-Resnet110 Beats Resnet1202

Lu, Yiping, et al. "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations." ICML 2018

Continuous ODE & Its Derivatives

Depth Revolution

Differential Equation as Finite Layer



Neural ODE

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

$$\frac{\partial h(t)}{dt} = \mathrm{f}(\mathrm{h}(t),\ t,\ \theta)$$

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 $\partial L / \partial \mathbf{z}(t_1)$

$\frac{\partial L}{\partial t_1} = \frac{\partial L}{\partial \mathbf{z}(t_1)}^{\mathsf{T}} f(\mathbf{z}(t_1), t_1, \theta)$ ▷ Compute gradient w.r.t. t_1

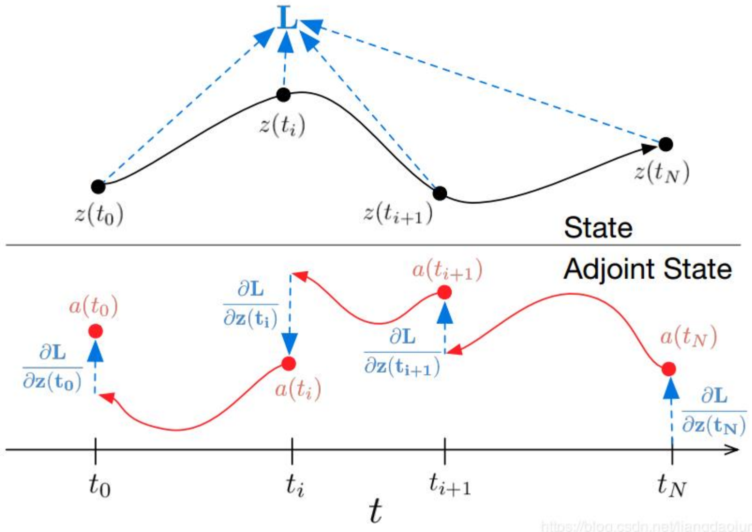
$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}, -\frac{\partial L}{\partial t_1}]$ ▷ Define initial augmented state

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

  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \theta}, -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial t}]$ ▷ Concatenate time-derivatives

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

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_1}$ ▷ Return all gradients



Chen, Tian Qi, et al. "Neural Ordinary Differential Equations."NeurIPS2018 (best paper)

Neural SDE

$$dh_t = f(h_t, t; \omega)dt + G(h_t, t; v)dB_t$$

Gaussian Noise Injection

$$h_{n+1} = h_n + f(h_n; \omega_n) + \Sigma_n z_n \qquad \Sigma_n = \sigma_n I, z_n \sim \mathcal{N}(0, 1)$$

DropOut

$$h_{n+1} = h_n + f(h_n; \omega_n) + f(h_n; \omega_n) \circ (\frac{\gamma_n}{p} - I) \qquad \gamma_n \sim \mathcal{B}(1, p)$$

Theorem 3.1. *For continuously differentiable loss $\ell(h_{t_1})$, we can obtain an unbiased gradient estimator as*

$$\boxed{\frac{\widehat{\partial L}}{\partial w} = \frac{\partial \ell(h_{t_1})}{\partial w} = \frac{\partial \ell(h_{t_1})}{\partial h_{t_1}} \cdot \frac{\partial h_{t_1}}{\partial w}.} \tag{9}$$

Moreover, if we define $\beta_t = \frac{\partial h_t}{\partial w}$, then β_t follows another SDE

$$\boxed{\mathrm{d}\beta_t = \Big(\frac{\partial f(h_t, t; w)}{\partial w} + \frac{\partial f(h_t, t; w)}{\partial h_t}\beta_t\Big)\,\mathrm{dt} + \Big(\frac{\partial G(h_t, t; w)}{\partial w} + \frac{\partial G(h_t, t; w)}{\partial h_t}\beta_t\Big)\,\mathrm{dB}_t.} \tag{10}$$

Liu X, Xiao T, Si S, et al. Neural sde: Stabilizing neural ode networks with stochastic noise[J]. arXiv:1906.02355, 2019.

Deep Equilibrium Models

$$\lim_{i\to\infty} \mathbf{z}_{1:T}^{[i]} = \lim_{i\to\infty} f_\theta\left(\mathbf{z}_{1:T}^{[i]}; \mathbf{x}_{1:T}\right) \equiv f_\theta\left(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T}\right) = \mathbf{z}_{1:T}^{\star}$$

Equilibrium Point

$$\frac{\partial \mathbf{z}_{1:T}^{\star}}{\partial(\cdot)} = \frac{\mathrm{d} f_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T})}{\mathrm{d}(\cdot)} + \frac{\partial f_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T})}{\partial \mathbf{z}_{1:T}^{\star}} \frac{\partial \mathbf{z}_{1:T}^{\star}}{\partial(\cdot)}$$

Back Propagation

$$\left(I - \frac{\partial f_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T})}{\partial \mathbf{z}_{1:T}^{\star}}\right) \frac{\partial \mathbf{z}_{1:T}^{\star}}{\partial(\cdot)} = \frac{\mathrm{d} f_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T})}{\mathrm{d}(\cdot)}$$

$$\frac{\partial \mathbf{z}_{1:T}^{\star}}{\partial(\cdot)} = -\left(J_{g_\theta}^{-1}\big|_{\mathbf{z}_{1:T}^{\star}}\right) \frac{\mathrm{d} f_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T})}{\mathrm{d}(\cdot)} \qquad J_{g_\theta}\big|_{\mathbf{z}_{1:T}^{\star}} = -\left(I - \frac{\partial f_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T})}{\partial \mathbf{z}_{1:T}^{\star}}\right)$$

$$g_\theta(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T}) = f_\theta\left(\mathbf{z}_{1:T}^{\star}; \mathbf{x}_{1:T}\right) - \mathbf{z}_{1:T}^{\star} \to 0$$

Broyden Iterations

$$\mathbf{z}_{1:T}^{[i+1]} = \mathbf{z}_{1:T}^{[i]} - \alpha B g_\theta(\mathbf{z}_{1:T}^{[i]}; \mathbf{x}_{1:T}) \quad \text{for } i = 0, 1, 2, \ldots$$

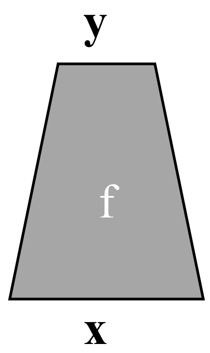
α – step size, $B \approx J_{g_\theta}^{-1}\big|_{\mathbf{z}_{1:T}^{[i]}}$

Shaojie Bai et al. Deep Equilibrium Models. NeurIPS 2019

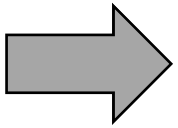
Normalizing Flows

$$y = f(x)$$

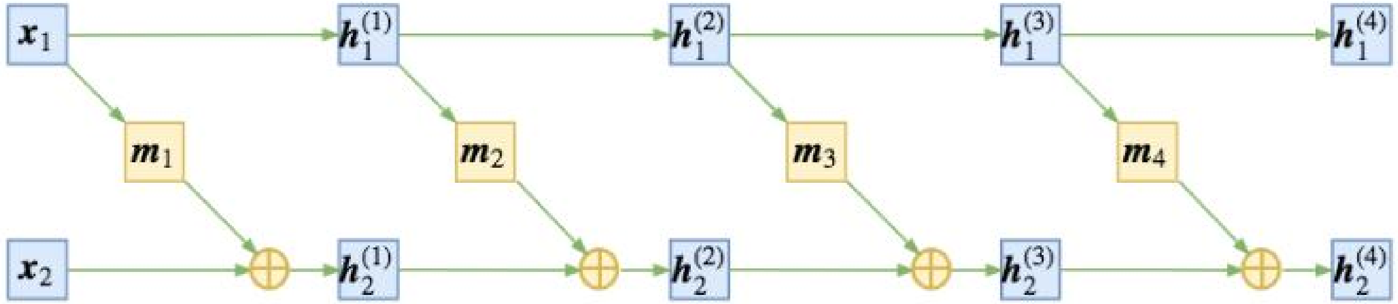
$$p(x) = p(y)|\det \frac{\partial f(x)}{\partial x}|$$



$$h_1 = x_1$$
$$h_2 = s(x_1) \circ x_2 + m(x_1)$$



$$x_1 = h_1$$
$$x_2 = s(h_1)^{-1}(h_2 - m(h_1))$$



Laurent Dinh et al. NICE: Non-linear Independent Components Estimation. ICLR2015

Normalizing Flows

Algorithm 1 Variational Inf. with Normalizing Flows

Parameters: ϕ variational, $\boldsymbol{\theta}$ generative
while not converged **do**
  $\mathbf{x} \leftarrow \{\text{Get mini-batch}\}$
  $\mathbf{z}_0 \sim q_0(\bullet|\mathbf{x})$
  $\mathbf{z}_K \leftarrow f_K \circ f_{K-1} \circ \ldots \circ f_1(\mathbf{z}_0)$
  $\mathcal{F}(\mathbf{x}) \approx \mathcal{F}(\mathbf{x}, \mathbf{z}_K)$
  $\Delta\boldsymbol{\theta} \propto -\nabla_{\theta}\mathcal{F}(\mathbf{x})$
  $\Delta\phi \propto -\nabla_{\phi}\mathcal{F}(\mathbf{x})$
end while

$$\mathbf{z}_K = f_K \circ \ldots \circ f_2 \circ f_1(\mathbf{z}_0)$$

$$\ln q_K(\mathbf{z}_K) = \ln q_0(\mathbf{z}_0) - \sum_{k=\mathbf{1}}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|$$

Planar NF

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$

$$\psi(\mathbf{z}) = h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w}$$

$$\det \left| \frac{\partial f}{\partial \mathbf{z}} \right| = |\det(\mathbf{I} + \mathbf{u}\psi(\mathbf{z})^\top)| = |1 + \mathbf{u}^\top \psi(\mathbf{z})|.$$

Radial NF

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_0)$$

$$\det \left| \frac{\partial f}{\partial \mathbf{z}} \right| = [1 + \beta h(\alpha, r)]^{d-1} [1 + \beta h(\alpha, r) + h'(\alpha, r)r)]$$

Langevin NF

$$d\mathbf{z}(t) = \mathbf{F}(\mathbf{z}(t), t)dt + \mathbf{G}(\mathbf{z}(t), t)d\boldsymbol{\xi}(t)$$

$$\frac{\partial}{\partial t} q_t(\mathbf{z}) = -\sum_i \frac{\partial}{\partial z_i} [F_i(\mathbf{z}, t) q_t] + \frac{1}{2} \sum_{i,i} \frac{\partial^2}{\partial z_i \partial z_j} [D_{ij}(\mathbf{z}, t) q_t]$$

$$D = GG^T$$

Continuous NF

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

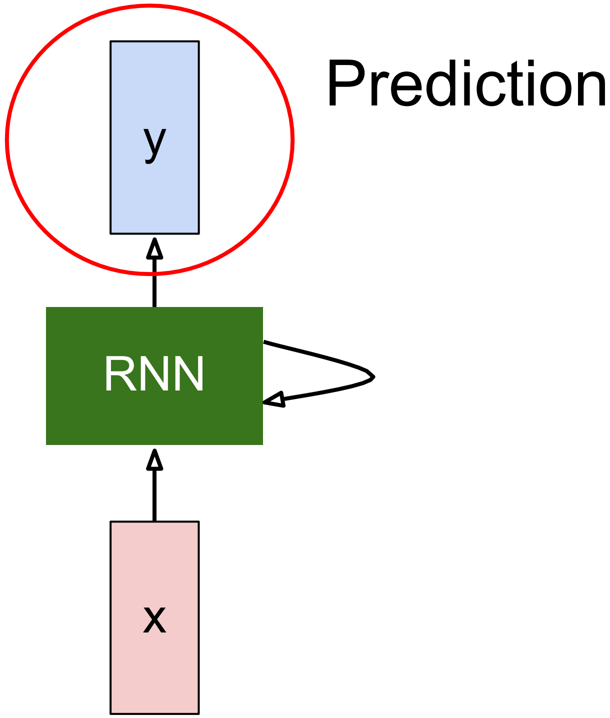
$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{tr}\left(\frac{\partial g(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)} \right)$$

Danilo Jimenez Rezende et al. Variational Inference with Normalizing Flows. ICML 2015

[1] Chen, Ricky TQ, et al. "Neural ordinary differential equations." *Advances in neural information processing systems*. 2018.
[2] Dupont, Emilien, Arnaud Doucet, and Yee Whye Teh. "Augmented neural odes." *Advances in Neural Information Processing Systems*. 2019.
[3] Finlay, Chris, et al. "How to train your neural ode." *arXiv preprint arXiv:2002.02798* (2020).
[4] Grathwohl, Will, et al. "Ffjord: Free-form continuous dynamics for scalable reversible generative models." *arXiv preprint arXiv:1810.01367* (2018).
[5] Shaojie Bai et al. Deep Equilibrium Models. *Advances in Neural Information Processing Systems*. 2019.
[6] Heinonen, Markus, et al. "Learning unknown ODE models with Gaussian processes." *arXiv preprint arXiv:1803.04303* (2018).
[7] Quaglino, Alessio, et al. "Snode: Spectral discretization of neural odes for system identification." *arXiv preprint arXiv:1906.07038* (2019).
[8] Yıldız, Çağatay, Markus Heinonen, and Harri Lähdesmäki. "ODE2VAE: Deep generative second order ODEs with Bayesian neural networks." *arXiv preprint arXiv:1905.10994* (2019).
[9] Zhang, H., et al. "Approximation Capabilities of Neural ODEs and Invertible Residual Networks." ICML, 2020.
[10] Jia, Junteng, and Austin R. Benson. "Neural jump stochastic differential equations." *Advances in Neural Information Processing Systems*. 2019.
[11] Zhuang, Juntang, et al. "Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE." *arXiv preprint arXiv:2006.02493* (2020).
[12] Massaroli, Stefano, et al. "Dissecting neural odes." *arXiv preprint arXiv:2002.08071* (2020).
[13] Zhong, Yaofeng Desmond, Biswadip Dey, and Amit Chakraborty. "Symplectic ode-net: Learning hamiltonian dynamics with control." *arXiv preprint arXiv:1909.12077* (2019).
[14] Norcliffe, Alexander, et al. "On Second Order Behaviour in Augmented Neural ODEs." *arXiv preprint arXiv:2006.07220* (2020).
[15] Papamakarios, George, et al. "Normalizing flows for probabilistic modeling and inference." arXiv preprint arXiv:1912.02762 (2019).

RNN&LSTM

RNN and LSTM



Input variable at t

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

Output at t

Output from $t-1$

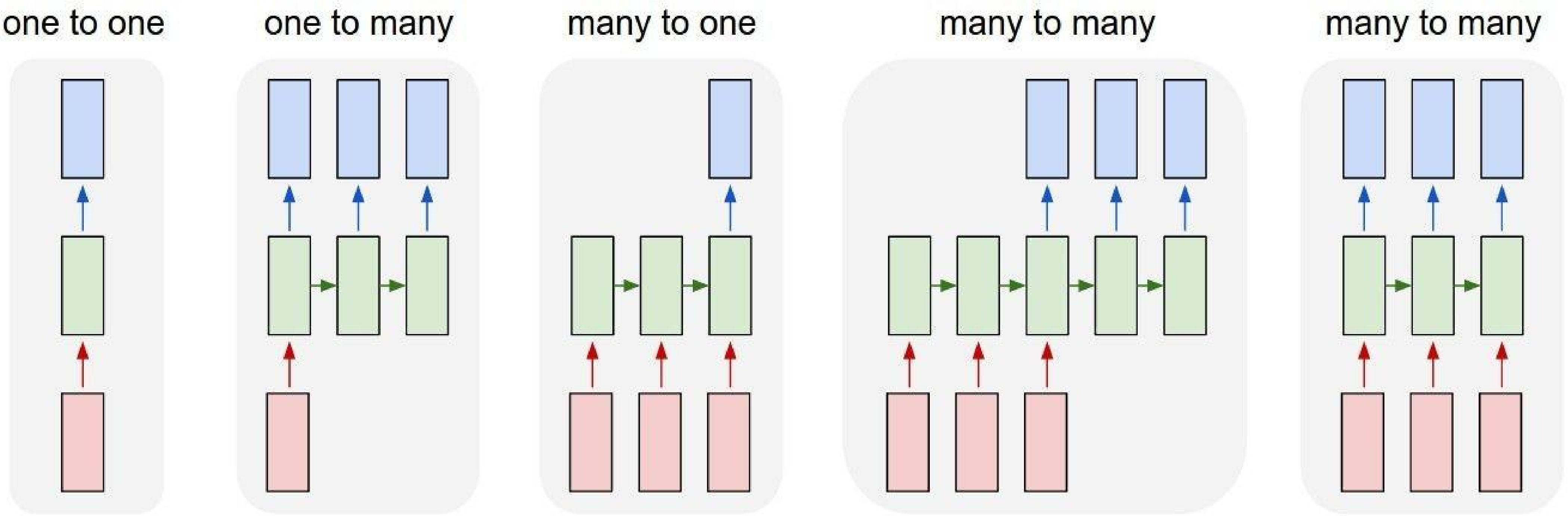
Nonlinear function
with trainable W

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

RNN&LSTM

Flexibility in applications:



RNN&LSTM

Starting with a ODE

$$\frac{d\vec{s}(t)}{dt} = \vec{f}(t) + \vec{\phi}(t)$$

$$\vec{f}(t) = \vec{h}(\vec{s}(t), \vec{x}(t))$$

$$\vec{f}(t) = \vec{a}(t) + \vec{b}(t) + \vec{c}(t)$$

$$\vec{a}(t) = \sum_{k=0}^{K_s-1} \vec{a}_k(\vec{s}(t-\tau_s(k)))$$

$$\vec{b}(t) = \sum_{k=0}^{K_r-1} \vec{b}_k(\vec{r}(t-\tau_r(k)))$$

$$\vec{r}(t-\tau_r(k)) = G\left(\vec{s}(t-\tau_r(k))\right)$$

$$\vec{c}(t) = \sum_{k=0}^{K_x-1} \vec{c}_k(\vec{x}(t-\tau_x(k)))$$

$$\frac{d\vec{s}(t)}{dt} = \sum_{k=0}^{K_s-1} \vec{a}_k(\vec{s}(t-\tau_s(k))) + \sum_{k=0}^{K_r-1} \vec{b}_k(\vec{r}(t-\tau_r(k))) + \sum_{k=0}^{K_x-1} \vec{c}_k(\vec{x}(t-\tau_x(k))) + \vec{\phi}$$

Assume linear

$$\frac{d\vec{s}(t)}{dt} = \sum_{k=0}^{K_s-1} A_k(\vec{s}(t-\tau_s(k))) + \sum_{k=0}^{K_r-1} B_k(\vec{r}(t-\tau_r(k))) + \sum_{k=0}^{K_x-1} C_k(\vec{x}(t-\tau_x(k))) + \vec{\phi}$$

$$\boxed{\frac{d\vec{s}(t)}{dt} = A\vec{s}(t) + B\vec{r}(t-\tau_0) + C\vec{x}(t) + \vec{\phi}}$$

$$\left.\begin{aligned} K_s &= 1 \\ \tau_s(0) &= 0 \\ A_0 &= A \\ K_r &= 1 \\ \tau_r(0) &= \tau_0 \\ B_0 &= B \\ K_x &= 1 \\ \tau_x(0) &= 0 \\ C_0 &= C \end{aligned}\right\}$$

A. Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network, arXiv:1808.03314.

RNN&LSTM

$$\frac{d\vec{s}(t)}{dt} = A\vec{s}(t) + B\vec{r}(t-\tau_0) + C\vec{x}(t) + \vec{\phi}$$

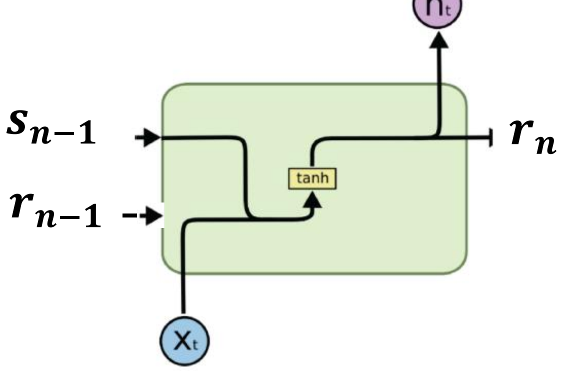
Discretize

$$\frac{\vec{s}(n\Delta T + \Delta T) - \vec{s}(n\Delta T)}{\Delta T} \approx A\vec{s}(n\Delta T + \Delta T) + B\vec{r}(n\Delta T + \Delta T - \tau_0) + C\vec{x}(n\Delta T + \Delta T) + \vec{\phi}$$

Let $\boldsymbol{\Delta T = 1}$ and $\boldsymbol{\tau_0 = \Delta T}$

$$\vec{s}[n] = W_s\vec{s}[n-1] + W_r\vec{r}[n-1] + W_x\vec{x}[n] + \vec{\theta}_s$$
$$\vec{r}[n] = G(\vec{s}[n])$$

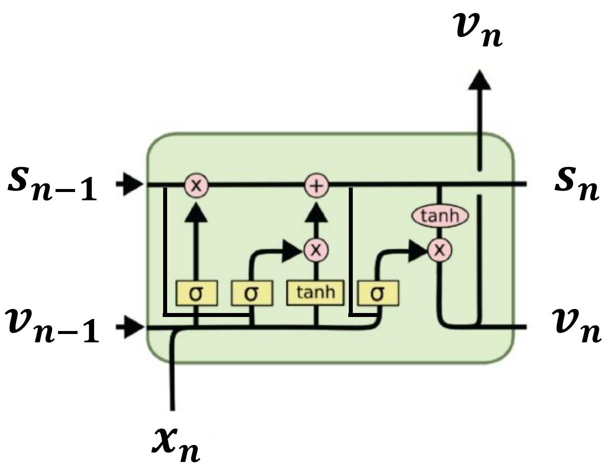
$$W_s = (I - (\Delta T)A)^{-1}$$
$$W_r = (\Delta T)W_sB$$
$$W_x = (\Delta T)W_sC$$
$$\vec{\theta}_s = (\Delta T)W_s\vec{\phi}$$



RNN if $W_s = 0$

A. Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network, arXiv:1808.03314.

RNN&LSTM



$$\vec{s}[n] = \vec{\mathcal{F}}_s(\vec{s}[n-1]) + \vec{\mathcal{F}}_u(\vec{r}[n-1], \vec{x}[n])$$
$$\vec{r}[n] = G_d(\vec{s}[n])$$
$$\vec{\mathcal{F}}_s(\vec{s}[n-1]) = W_s \vec{s}[n-1]$$
$$\vec{\mathcal{F}}_u(\vec{r}[n-1], \vec{x}[n]) = W_r \vec{r}[n-1] + W_x \vec{x}[n] + \vec{\theta}_s$$

LSTM

$$\vec{s}[n] = \vec{g}_{cs}[n] \odot \vec{\mathcal{F}}_s(\vec{s}[n-1]) + \vec{g}_{cu}[n] \odot \vec{\mathcal{F}}_u(\vec{r}[n-1], \vec{x}[n])$$
$$\vec{0} \leq \vec{g}_{cs}[n], \vec{g}_{cu}[n] \leq \vec{1}$$

$$\vec{u}[n] = G_d\left(\vec{\mathcal{F}}_u(\vec{v}[n-1], \vec{x}[n])\right)$$
$$\vec{v}[n] = \vec{g}_{cr}[n] \odot \vec{r}[n]$$

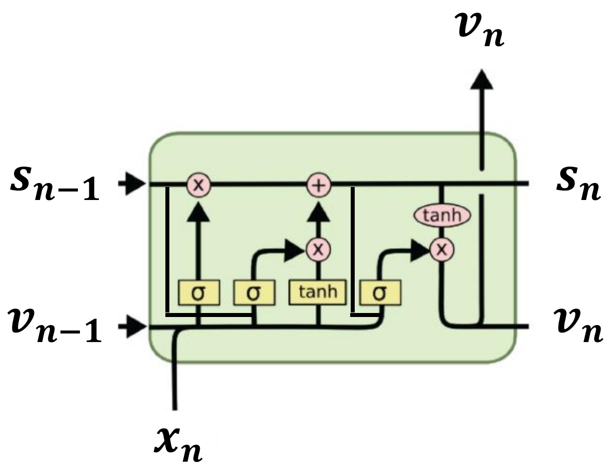
A. Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network, arXiv:1808.03314.

RNN&LSTM

Vanilla LSTM

$$
\begin{aligned}
\vec{a}_{cu}[n] &= W_{x_{cu}}\vec{x}[n] + W_{s_{cu}}\vec{s}[n-1] + W_{v_{cu}}\vec{v}[n-1] + \vec{b}_{cu} \\
\vec{a}_{cs}[n] &= W_{x_{cs}}\vec{x}[n] + W_{s_{cs}}\vec{s}[n-1] + W_{v_{cs}}\vec{v}[n-1] + \vec{b}_{cs} \\
\vec{a}_{cr}[n] &= W_{x_{cr}}\vec{x}[n] + W_{s_{cr}}\vec{s}[n] + W_{v_{cr}}\vec{v}[n-1] + \vec{b}_{cr} \\
\vec{a}_{du}[n] &= W_{x_{du}}\vec{x}[n] + W_{v_{du}}\vec{v}[n-1] + \vec{b}_{du} \\
\vec{u}[n] &= G_d(\vec{a}_{du}[n]) \\
\vec{g}_{cu}[n] &= G_c(\vec{a}_{cu}[n]) \\
\vec{g}_{cs}[n] &= G_c(\vec{a}_{cs}[n]) \\
\vec{g}_{cr}[n] &= G_c(\vec{a}_{cr}[n]) \\
\vec{s}[n] &= \vec{g}_{cs}[n] \odot \vec{s}[n-1] + \vec{g}_{cu}[n] \odot \vec{u}[n] \\
\vec{r}[n] &= G_d(\vec{s}[n]) \\
\vec{v}[n] &= \vec{g}_{cr}[n] \odot \vec{r}[n]
\end{aligned}
$$

$$
\Theta \equiv \left\{ \begin{array}{llll}
W_{x_{cu}}, & W_{s_{cu}}, & W_{v_{cu}}, & \vec{b}_{cu}, \\
W_{x_{cs}}, & W_{s_{cs}}, & W_{v_{cs}}, & \vec{b}_{cs}, \\
W_{x_{cr}}, & W_{s_{cr}}, & W_{v_{cr}}, & \vec{b}_{cr}, \\
W_{x_{du}}, & & W_{v_{du}}, & \vec{b}_{du}
\end{array} \right\}
\begin{array}{l}
\text{Input gate} \\
\text{Forget gate} \\
\text{Output gate} \\
\text{Gate gate}
\end{array}
$$



A. Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network, arXiv:1808.03314.

Thanks and Questions?