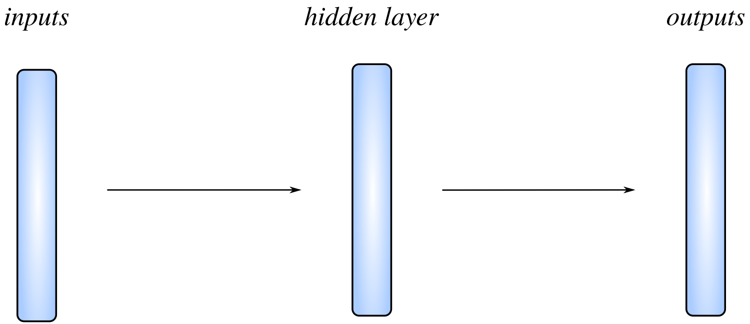
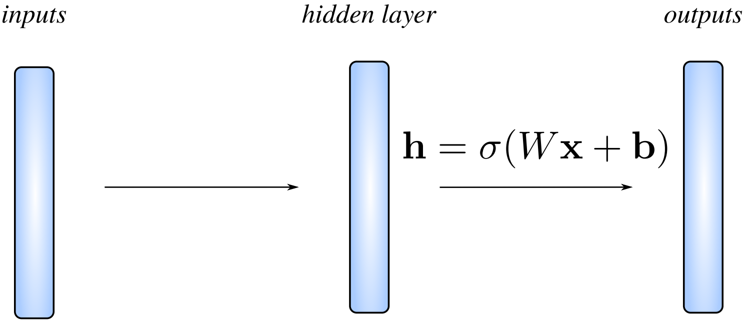
Neural ordinary differential equations

Olof Mogren, Research institutes of Sweden

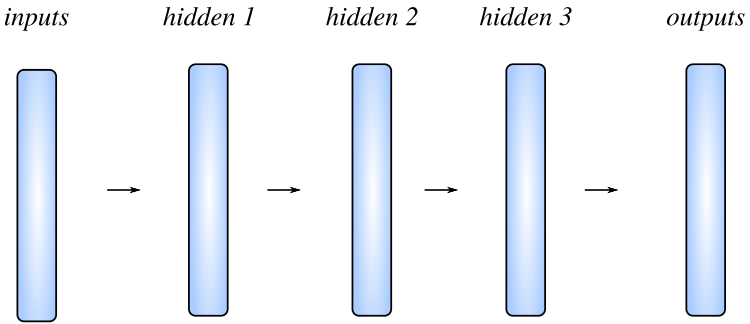
Neural networks



Neural networks



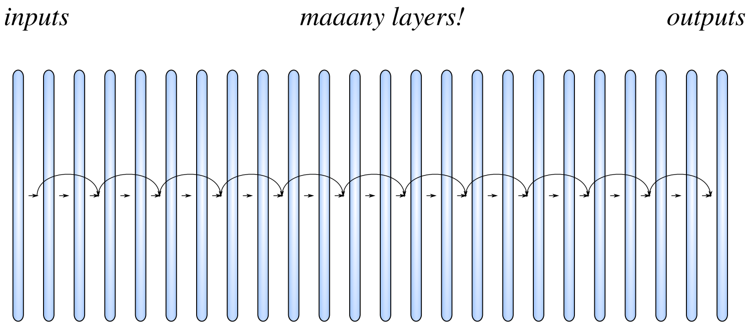
Deep neural networks



Deep nets

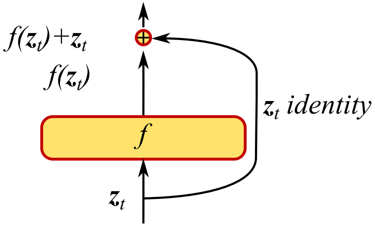
- More layers ->
  - Decreased length of step taken in each layer

Residual neural networks



Residual connections

- $\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \theta_t)$
- A layer learns the difference between $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$
- Deeper net $\rightarrow$ smaller differences
- What happens in the limit?



Ordinary differential equation for Resnets

- In the limit, state **z** updates:

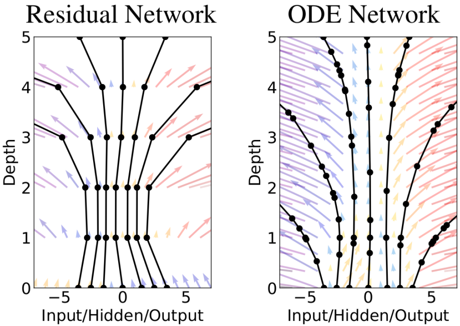
$$\frac{\partial \mathbf{z}(t)}{\partial t} = f(\mathbf{z}(t), t, \theta)$$

Continuous depth neural networks

- $\mathbf{z}(t)$ - the state (in the paper introduction: $\mathbf{h_t}$)
  - $\mathbf{z}(0)$ (input vector)
  - $\mathbf{z}(T)$ (output vector)
  - $\mathbf{z}(t)$, $t \in (0, T)$ (internal state)
- State is transformed continuously from $\mathbf{z}(0)$ to $\mathbf{z}(T)$
- Parameterize the gradient of the state with a neural net f:

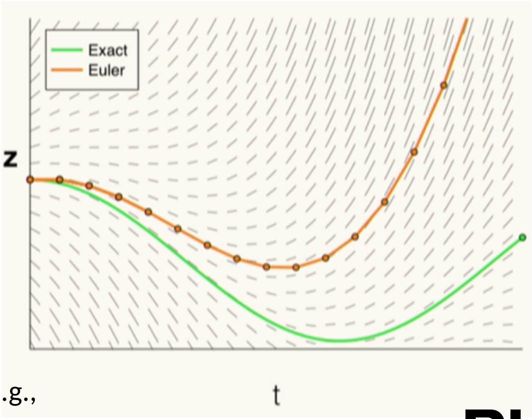
$$\frac{\partial \mathbf{z}(t)}{\partial t} = f(\mathbf{z}(t), t, \theta)$$

Continuous depth neural networks (2)



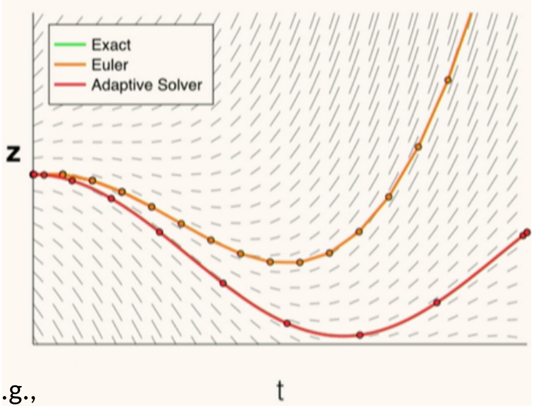
Ordinary differential equation (ODE) solvers

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{\partial \mathbf{z}}{\partial t} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}_{t_0}$, find
  - $\mathbf{z}_{t_1} = \mathbf{z}_{t_0} + \int_{t_0}^{t_1} f(\mathbf{z}_t, t, \theta) dt$
- Oldest and simplest: Euler's method
- Takes a small step h in gradient's direction
  - $\mathbf{z}(t+h) = \mathbf{z} + hf(\mathbf{z}, t)$
- Modern solvers: 120 years of improvements e.g., (Hairer, et.al., 1987)
  - Approximation error guarantees
  - Adaptive evaluation strategy

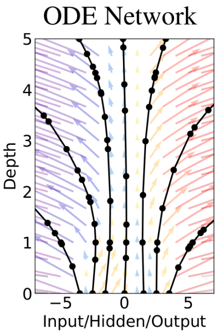


Ordinary differential equation (ODE) solvers

- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{\partial \mathbf{z}}{\partial t} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}_{t_0}$, find
  - $\mathbf{z}_{t_1} = \mathbf{z}_{t_0} + \int_{t_0}^{t_1} f(\mathbf{z}_t, t, \theta)dt$
- Oldest and simplest: Euler's method
- Takes a small step h in gradient's direction
  - $\mathbf{z}(t+h) = \mathbf{z} + hf(\mathbf{z}, t)$
- Modern solvers: 120 years of improvements e.g., (Hairer, et.al., 1987)
  - Approximation error guarantees
  - Adaptive evaluation strategy



ODENet: The steps of the ODE solver defines the neural network.



How to train the ODENet

- Adjoint sensitivity method (Pontryagin et al., 1962)
- Continuous time limit of standard back-propagation
- Solve another ODE in reverse direction
- Error guarantees
- Dynamic step sizes

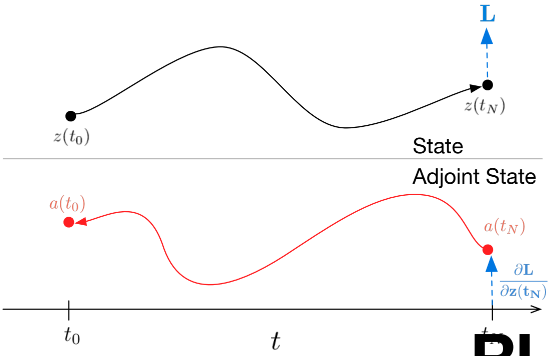
$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$$

$$\frac{\partial \mathbf{a}(t)}{\partial t} = \mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)}$$

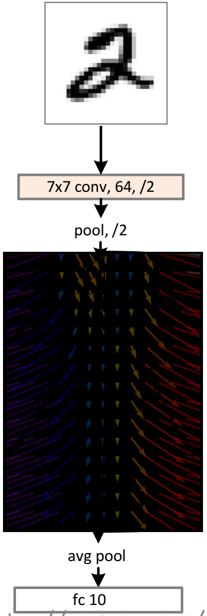
$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_o} \mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta}$$

O(1) Memory Gradients

- No need to store activations, just run dynamics backwards from output.
- Reversible ResNets (Gomez et al., 2018) must partition dimensions.



Drop-in replacement for Resnets



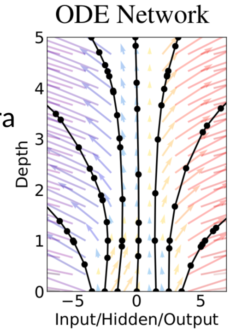
- Same performance with fewer parameters.

	Test Error	# Params
1-Layer MLP	1.60%	0.24 M
ResNet	0.41%	0.60 M
ODE-Net	0.42%	0.22 M

Chen, Rubanova, Bettencourt, Duvenaud

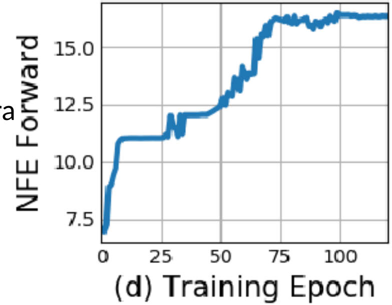
How deep are ODE-nets?

- ‘Depth’ is left to ODE solver.
- Dynamics become more demanding during tra
- 2-4x the depth of resnet architectures



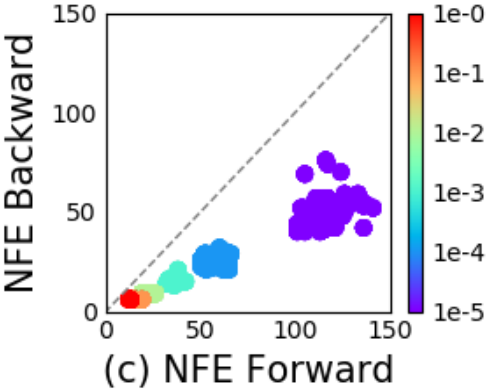
How deep are ODE-nets?

- ‘Depth’ is left to ODE solver.
- Dynamics become more demanding during tra
- 2-4x the depth of resnet architectures



Reverse vs Forward Cost

- Empirically, reverse pass roughly half as expensive as forward pass
- Again, adapts to instance difficulty
- Num evaluations comparable to number of layers in modern nets



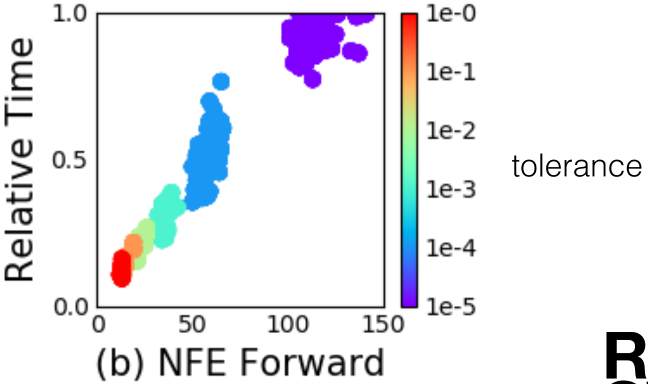
Chen, Rubanova, Bettencourt, Duvenaud

Speed-Accuracy Tradeoff

output = ODESolve(f, z0, t0, t1, theta, tolerance)

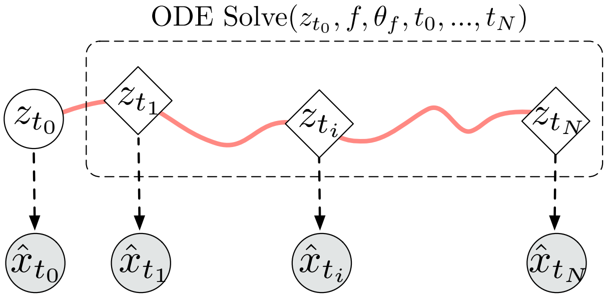
- Time cost is dominated by evaluation of dynamics

- Roughly linear with number of forward evaluations



Chen, Rubanova, Bettencourt, Duvenaud

Continuous-time models



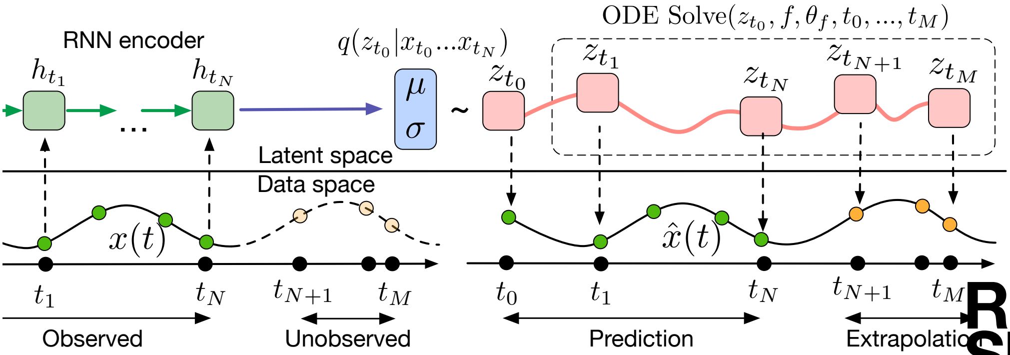
- Well-defined state at all times
- Dynamics separate from inference
- Irregularly-timed observations.

$$\mathbf{z}_{t_0} \sim p(\mathbf{z}_{t_0})$$
$$\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} = \mathrm{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N)$$
$$\text{each} \quad \mathbf{x}_{t_i} \sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}})$$

Chen, Rubanova, Bettencourt, Duvenaud

Continuous-time RNNs

- Can do VAE-style inference with an RNN encoder
- Actually, more like a Deep Kalman Filter

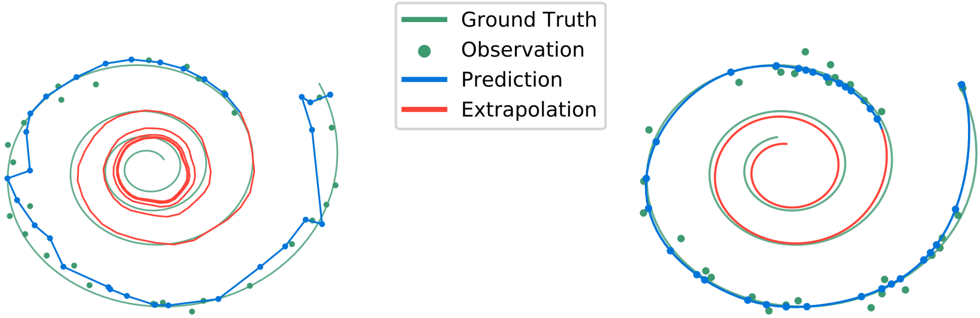


Chen, Rubanova, Bettencourt, Duvenaud

Continuous-time models

Recurrent Neural Net

Latent ODE



Chen, Rubanova, Bettencourt, Duvenaud

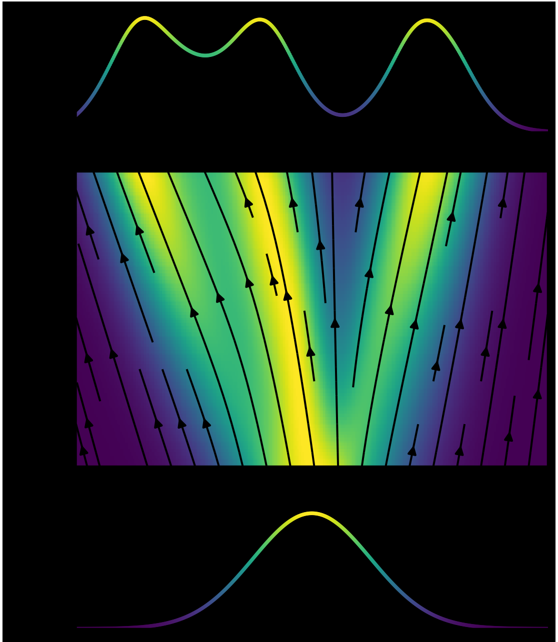
RISE

Normalizing flows

Tabak & Vanden-Eijnden 2010

- The transformation of a probability density through a sequence of invertible mappings
- Change of variables rule
- Produces a valid probability distribution
- Requires computing the determinant: $O(M^3)$

$$q(\mathbf{z}') = q(\mathbf{z}) \left| det \frac{\partial f^{-1}}{\partial \mathbf{z}'} \right| = q(\mathbf{z}) \left| det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1}$$



Instantaneous Change of Variables

$$\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$$

$$\Downarrow$$

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathbf{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

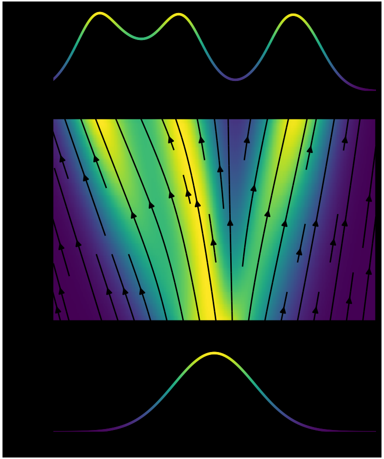
- Worst-case cost O(D^2).
- Only need continuously differentiable f

http:// / *Bettencourt, Duvenaud*

Continuous Normalizing Flows

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt$$

- Reversible dynamics, so can train from data by maximum likelihood
- No discriminator or recognition network, train by SGD
- No need to partition dimensions



Chen, Rubanova, Bettencourt, Duvenaud

Trading Depth for Width

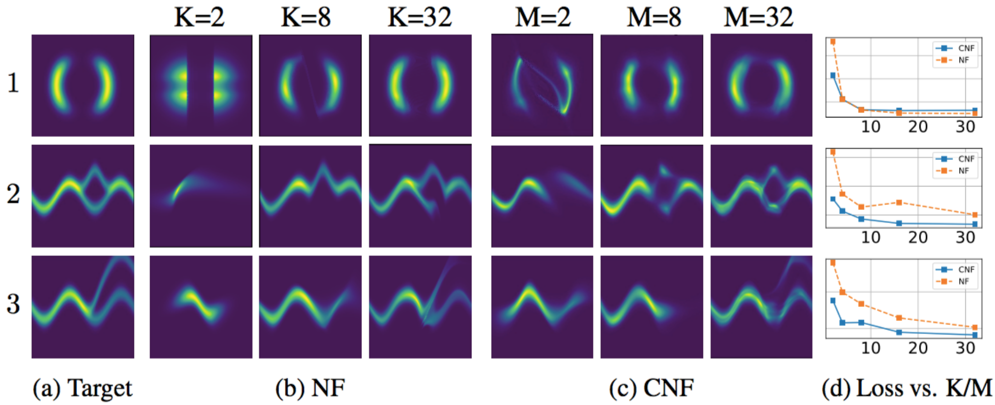


Figure 5: Comparison of NF and CNFs on learning generative models (noise → data) trained to minimize the reverse KL.

Chen, Rubanova, Bettencourt, Duvenaud

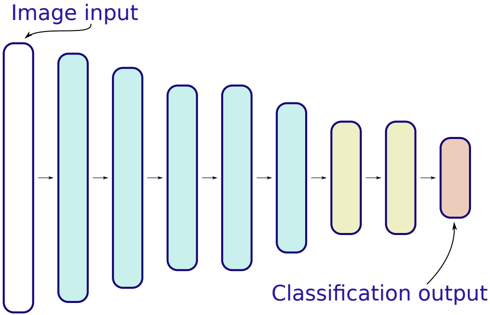
Concluding remarks

- Memory efficiency (constant)
- The ODE solver takes a tolerance parameter, trade-off accuracy vs running time
- Time-series with irregular observation times
- Continuous normalizing flows
- Computation time not not guaranteed
- 2-4 times slower than Resnets

Appendix

Deep learning

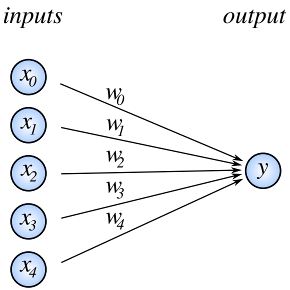
- Transforming data in sequential steps
- Distributed hierarchical internal representations
- End-to-end training
- **Scales well with large datasets and available computing power**



Back

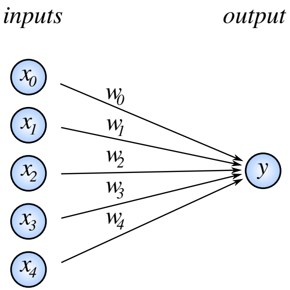
Deep learning building block

- Each layer contains a number of units
- *Loosely* inspired by biological neurons
- Deep networks can consist of millions of units
- $w_1, \ldots, w_n$ learned parameters



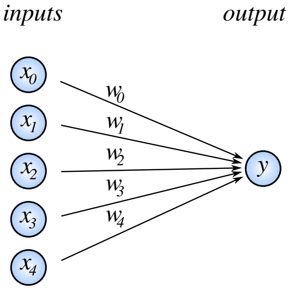
Deep learning building block

- Each layer contains a number of units
- *Loosely* inspired by biological neurons
- Deep networks can consist of millions of units
- $w_1, \ldots, w_n$ learned parameters



Deep learning building block

- Each layer contains a number of units
- *Loosely* inspired by biological neurons
- Deep networks can consist of millions of units
- $w_1, \ldots, w_n$ learned parameters



Deep learning building block

- Each layer contains a number of units
- *Loosely* inspired by biological neurons
- Deep networks can consist of millions of units
- $w_1, \ldots, w_n$ learned parameters