# Neural networks for function approximation and data-driven modeling

Connor Robertson

10/7/21

# Finding the form of a differential equation

Modeling is often made up of determining the form of a differential equation for a system.

**Techniques for determining the form:**

- Build on physical knowledge - derivation
- Use "phenomenological" terms - functions that have expected behavior

# Data-driven modeling with PDE-Find

Today's data-driven modeling core idea: build many possible terms and use sparse regression to determine equation form[1],[2].

For data points $(x_i, u(x_i))$ where $u$ is an unknown function, determine time evolution:

$$\vec{u}_t = M\vec{c}$$

where $M$ is a "library" of possible terms:

$$M = \begin{bmatrix} 1 & \vec{u} & \vec{u}^2 & \dots & \vec{u}_x & \vec{u}\vec{u}_x & \dots & \vec{u}^m u_{xxxx}^m \end{bmatrix}$$
$$\vec{u} = \begin{bmatrix} u(x_1) & \dots & u(x_n) \end{bmatrix}^T$$

[1] Brunton, Proctor **and** Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems".

[2] Rudy **andothers**, "Data-driven discovery of partial differential equations".

# Key ideas in PDE-Find

1. Need to have all terms represented
2. **Need to accurately differentiate terms**
3. Need to effectively reduce library

For differentiation of noisy data, use non-local methods to average out or correct for error – i.e. fit with a differentiable function basis

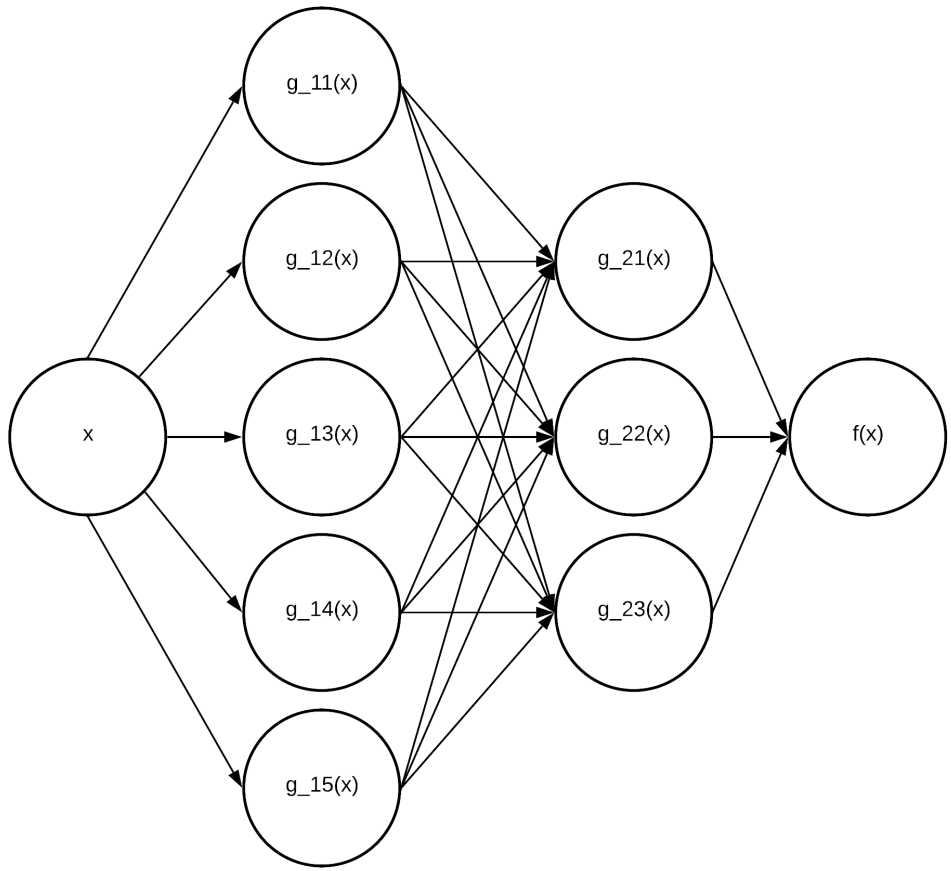
* Popular differentiable functions for fitting data right now are *neural networks*!

# Neural nets are functions

**Main components of neural network:**

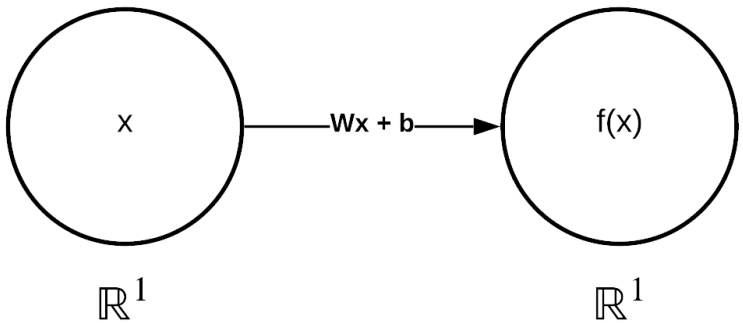
1. Linear transformations (lines)
2. Nonlinear "activation" functions (circles)
3. Loss function

Usually represented as computational graph:



# Neural net example function

**Linear regression:** Given data $(x_i, u(x_i))$




$$f(x) = Wx + b$$

---

**Loss function:** $L(f, x_i) = \|f(x_i) - u(x_i)\|_2^2$

# Neural network training

**Linear regression:**

$$f(x) = Wx + b$$

**Loss function:** $L(f, x_i) = \|f(x_i) - u(x_i)\|_2^2$

---

$$\underset{W,b}{\text{minimize}} \quad L(f, x_i)$$

- Stochastic gradient descent

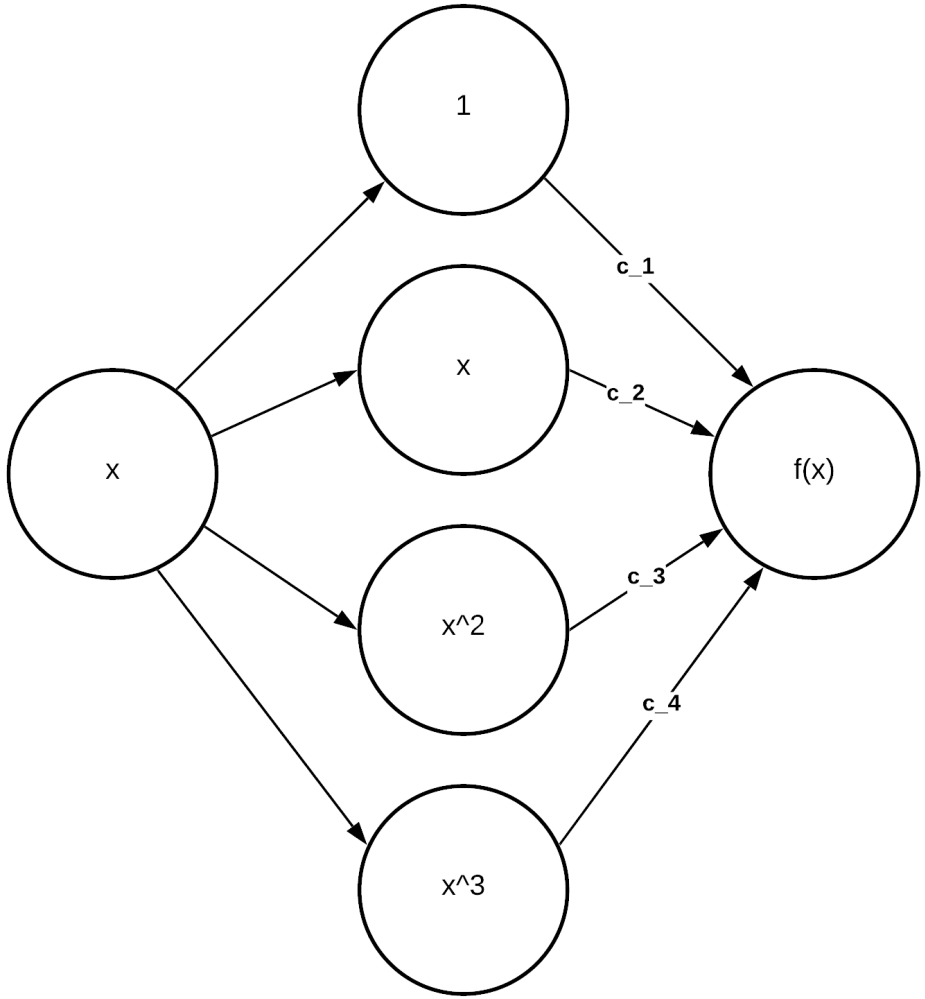
$$W = W - \alpha \frac{dL(f, x_i)}{dW}$$

$$\begin{aligned} \frac{dL(f, x_i)}{dW} &= 2(f(x_i) - u(x_i)) \frac{df(x_i)}{dW} \\ &= 2(f(x_i) - u(x_i))x_i \end{aligned}$$

# Neural networks are function compositions

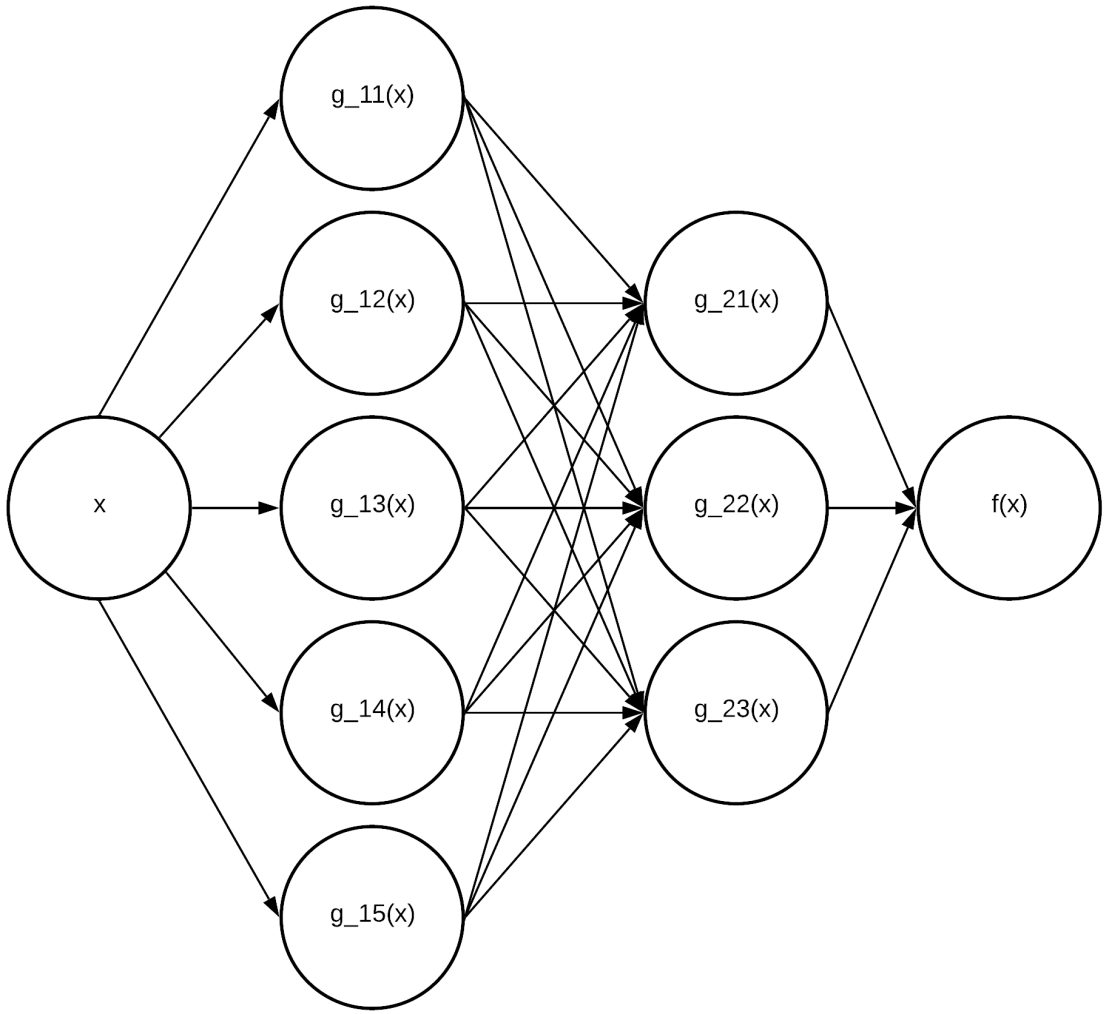
**Polynomial regression:**

$$f(x) = c_1 + c_2(W_2 x + b_2) + c_3(W_3 x + b_3)^2 + c_4(W_4 x + b_4)^3$$




**Loss function:** $L(f, x_i) = \|f(x_i) - u(x_i)\|_2^2$

# Feedforward Neural networks




$$f(x) = W_{21}g_{21}(W_{21}(g_{11}(Wx+b) + \ldots + g_{15}(Wx+b)) + b_{21} + b) + \ldots$$
$$+ W_{23}g_{23}(W(g_{11}(Wx+b) + \ldots + g_{15}(Wx+b)) + b) + b_{23}$$

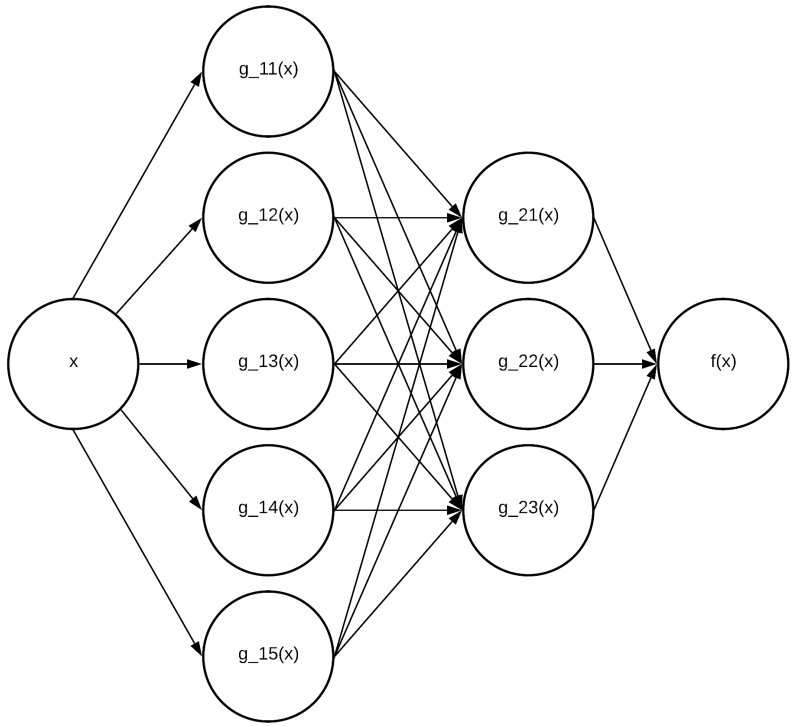
# Universal approximators

Fully connected neural network with a single hidden layer containing infinite nodes (possibly) can approximate any Borel measurable function on a compact domain to arbitrary uniform accuracy[3].

[3] Hornik, Stinchcombe **and** White, "Multilayer feedforward networks are universal approximators".

# Neural network parameters

1. **Depth:** Number of layers (compositions)
2. **Width:** Number of functions in each layer (sums)
3. **Activation functions:** Functions $g(x)$
   - 3.1 Sigmoid: $g(x) = \frac{1}{1+e^{-x}}$
   - 3.2 Tanh: $g(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
   - 3.3 ReLU (rectified linear unit): $g(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$



# Practical advantages and disadvantages

**Advantages:**

- Extremely flexible - discontinuous and piecewise functions
- Memory efficient in high dimension - parameter nesting

**Disadvantages:**

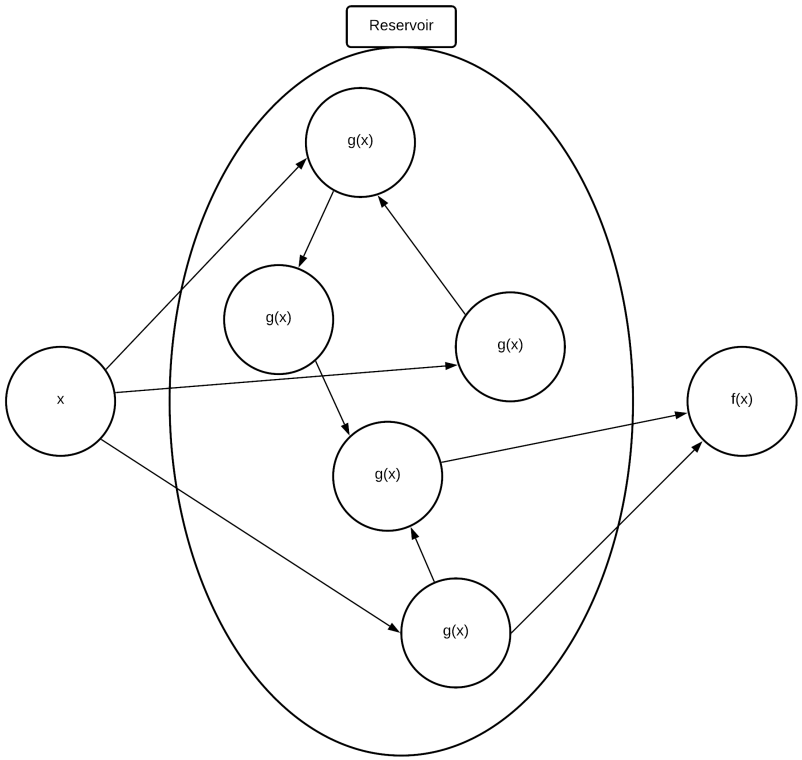
- Nonconvex optimization - local minima
- Theory is lacking - stability, accuracy, computational cost[4]
- Need massive amounts of data
- More of an art than a science

[4] Adcock **and** Dexter, "The Gap between Theory and Practice in Function Approximation with Deep Neural Networks".

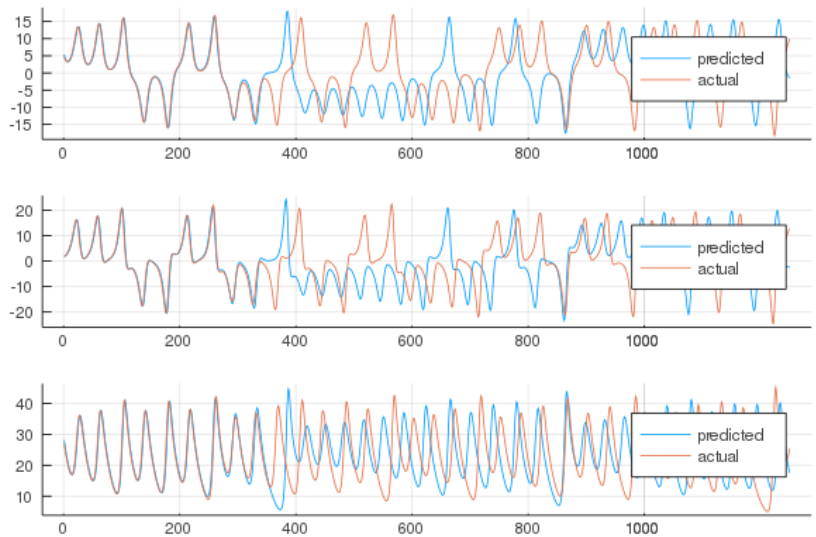
# Reservoir Computing

Hard to choose depth and width, so you can randomly fix weights with some rules to get a randomly generated nonlinear basis.

5




6




[5] Shi **and** Han, "Support Vector Echo-State Machine for Chaotic Time-Series Prediction".

[6] http://reservoir.sciml.ai/

# Deep Learning PDE form

Neural networks are differentiable via the chain rule as used in the training process.

For data $(x_i, u(x_i))$, make a function approximation network $f(x) \mapsto u(x)$, then consider the PDE-Find system[7]:

$$\vec{u}_t \approx \vec{f}_t = M\vec{c}$$

where $M$ is a "library" of possible terms:

$$M = \begin{bmatrix} 1 & \vec{f} & \vec{f}^2 & \dots & \vec{f}_x & \vec{f}\vec{f}_x & \dots & \vec{f}^m u^m_{xxxx} \end{bmatrix}$$
$$\vec{u} = \begin{bmatrix} f(x_1) & \dots & f(x_n) \end{bmatrix}^T$$

[7] Xu, Chang **and** Zhang, "DL-PDE".

# Deep learning PDE solutions

If you already know the form of your PDE, can use "Physics Informed Neural Networks (PINNs)" to numerically compute the solution.

For PDE $u_t = F(x, t, u)$ and data $(x_i, t_j, u(x_i, t_j))$, create neural network $f(x, t) \mapsto u(x, t)$ with loss function[8]:

$$\|u_t(x_i, t_j) - F(x_i, t_j, f(x_i, t_j))\|_2^2$$

* Some challenging numerical questions when differentiating PDE loss function

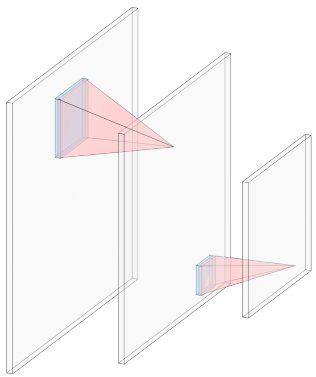
[8] Raissi, Perdikaris **and** Karniadakis, "Physics-informed neural networks".

# Convolutional operators

- There is a lot of flexibility in the activation functions $g(x)$
- One function that revolutionized neural networks for image processing was the convolution (discrete)
- Compile local spatial information from images

$$g(x)[m] = \sum_{n=-\infty}^{\infty} k[n]x[m-n]$$

where $k$ is a discrete convolutional kernel.



[9]Jaswal, Sowmya **and** Soman, "Image classification using convolutional neural networks".

# Convolutions as differentiation stencils

Discretely, local differentiation (e.g. not spectral methods) can be represented as the application of a discrete convolutional stencil.

**Centered finite difference (2D):**

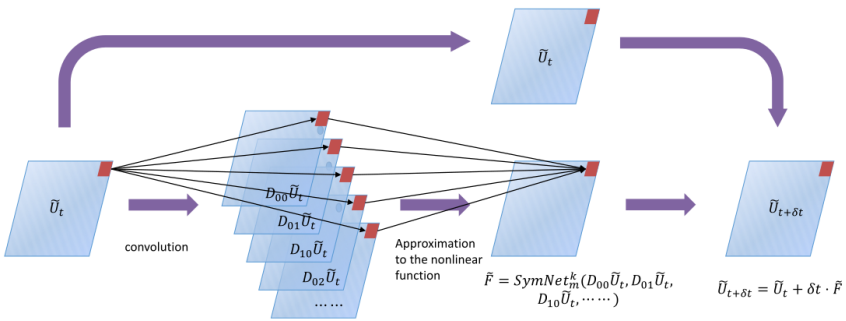
$$k = \frac{1}{2h} \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

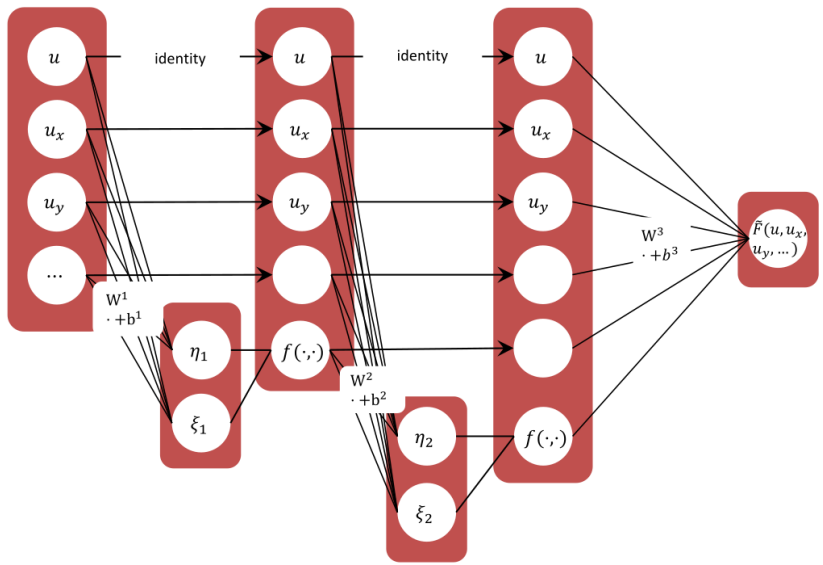
**5-point stencil (laplacian):**

$$k = \frac{1}{h^2} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

# Differentiation in neural networks

Using Wavelet theory, you can constrain learned convolutional filters to be derivatives of a given order and construct a PDE-Find neural network[10].



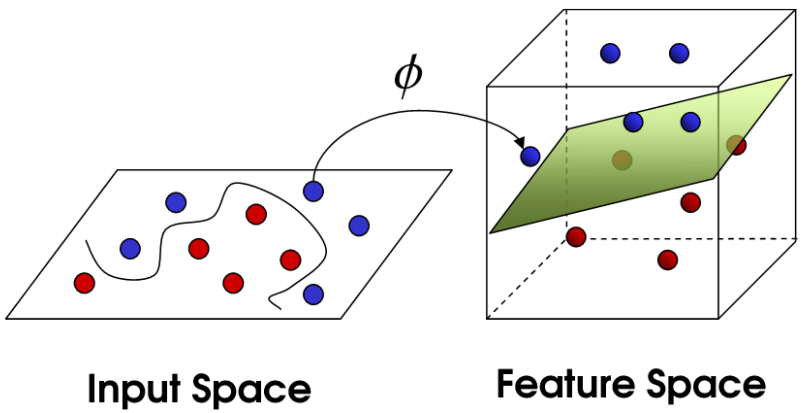



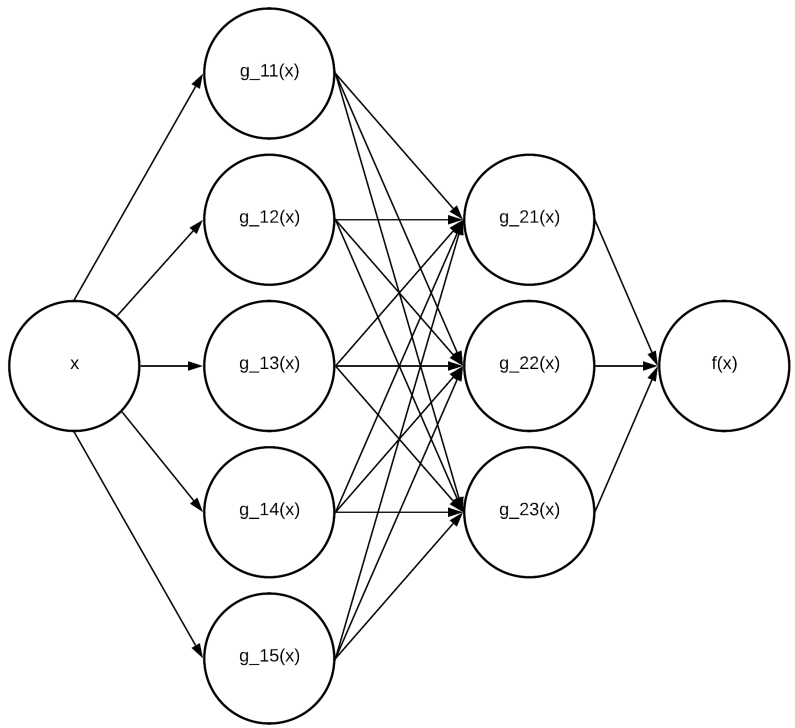

Convolutions contain the idea of dimensionality reduction.

[10] Long, Lu **and** Dong, "PDE-Net 2.0".

# Neural networks expanding dimensionality

Classic idea in machine learning for classification is to map data to higher dimension where it is linearly separable.

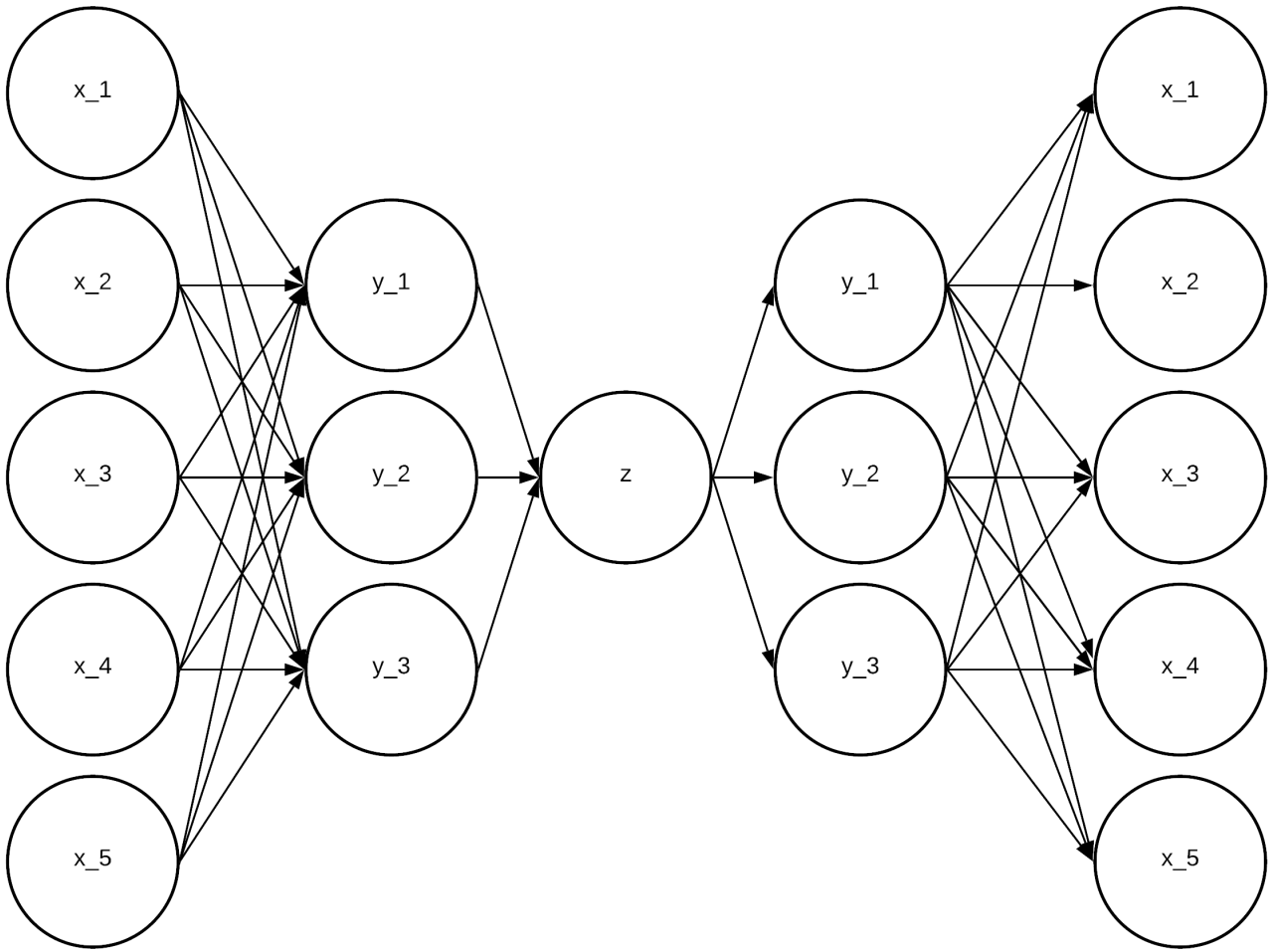



[11] https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

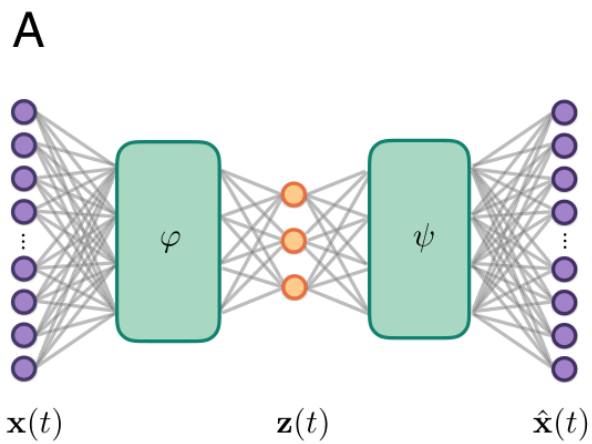
# Autoencoders

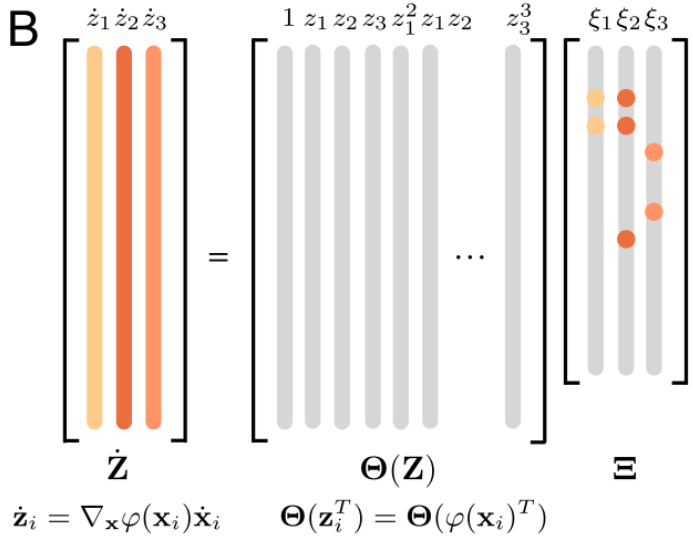
Can also reduce data to lower dimensionality using **nonlinear** functions. (nonlinear SVD?)



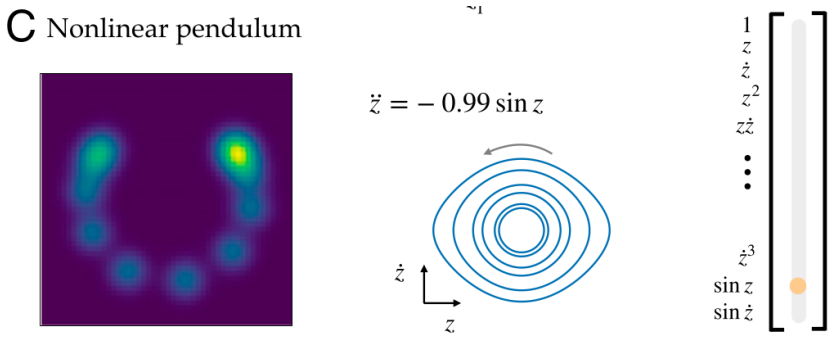
# Discovering dynamics on encoded data

Can we analyze dynamics on encoded data[12]? (AlphaFold does this!)



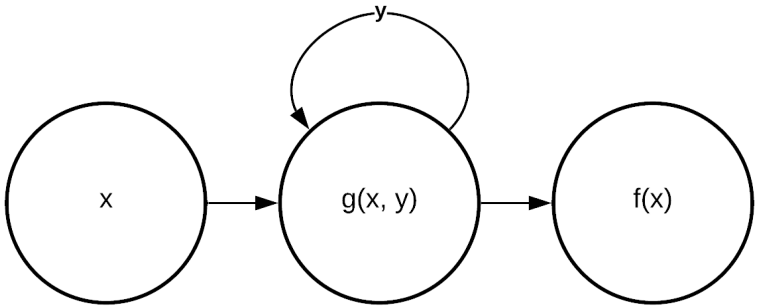

$$\underbrace{\|\mathbf{x}-\psi(\mathbf{z})\|_2^2}_{\text{reconstruction loss}} + \underbrace{\lambda_1\left\|\dot{\mathbf{x}}-(\nabla_{\mathbf{z}}\psi(\mathbf{z}))\left(\Theta(\mathbf{z}^T)\Xi\right)\right\|_2^2}_{\text{SINDy loss in }\dot{\mathbf{x}}} + \underbrace{\lambda_2\left\|(\nabla_{\mathbf{x}}\mathbf{z})\,\dot{\mathbf{x}}-\Theta(\mathbf{z}^T)\Xi\right\|_2^2}_{\text{SINDy loss in }\dot{\mathbf{z}}} + \underbrace{\lambda_3\|\Xi\|_1}_{\text{SINDy regularization}}$$

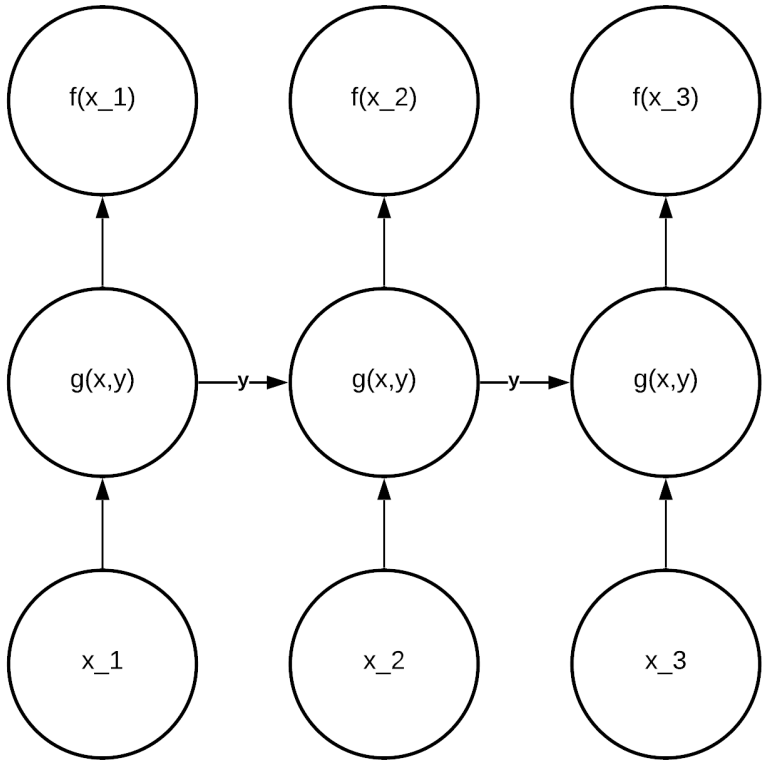



[12] Champion **andothers**, "Data-driven discovery of coordinates and governing equations".

# Recurrent neural networks (RNNs)

Can reuse parameters in the network to try to incorporate information from previous steps (useful for time series).

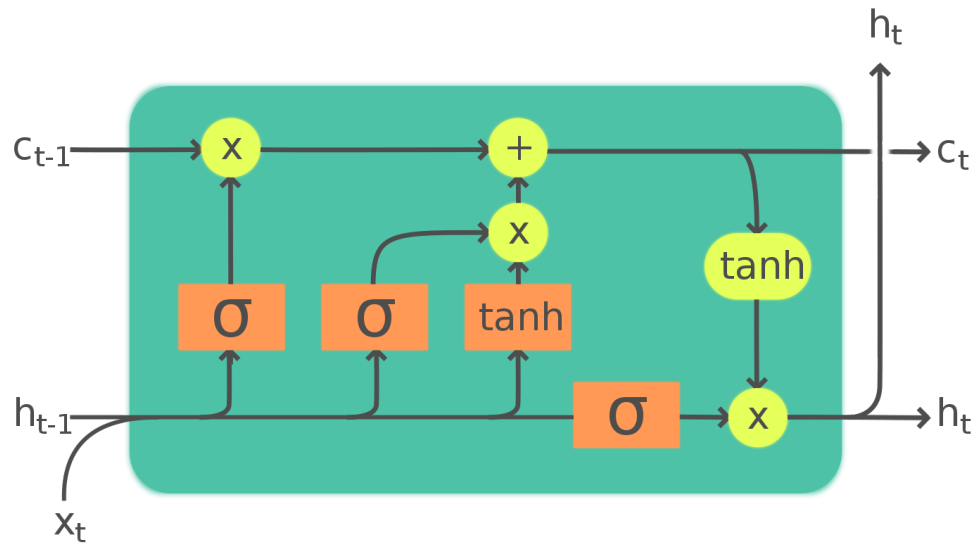


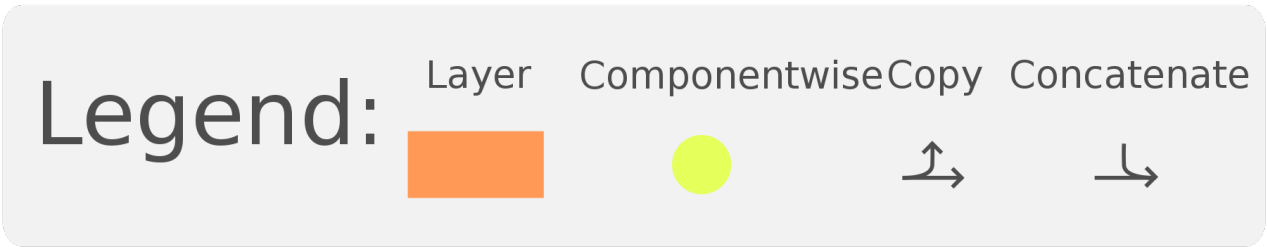




* Analogous to multistep Markov chains

# Recurrence computational considerations

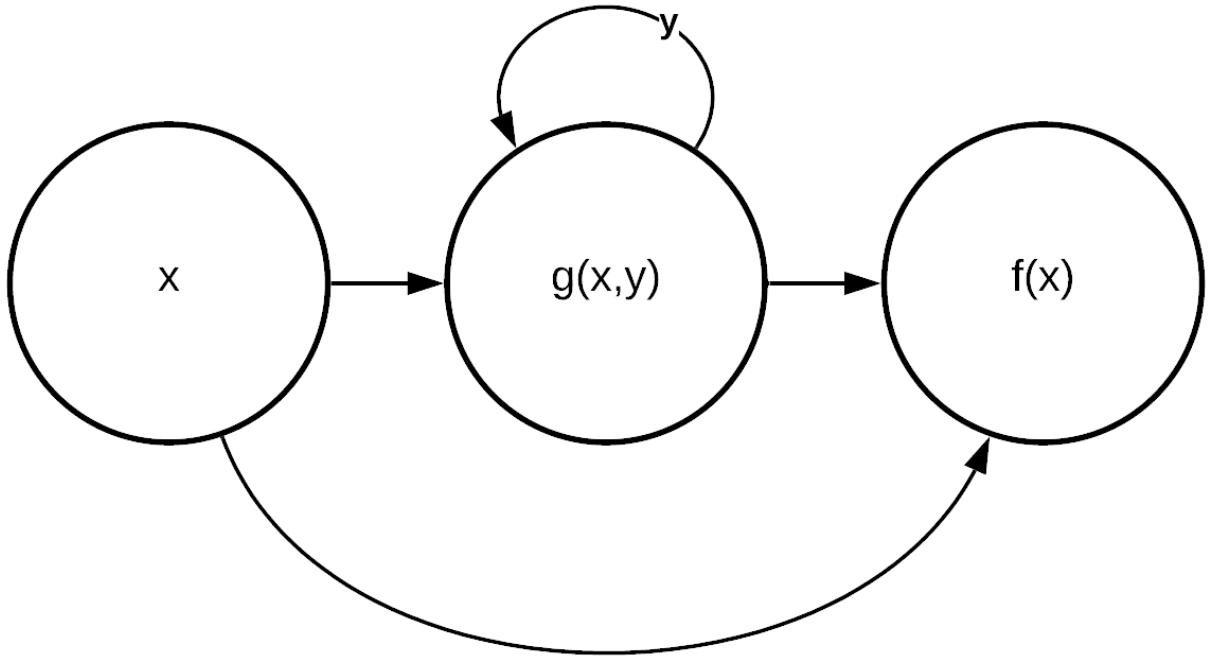
Nested evaluation of $g(x, y)$ in RNNs causes numerical instability.
Alternative Long Short Term Memory (LSTM) form:



# Residual neural network

Recurrence to account for evolution.




$$x^{t+1} = x^t + g(x, y)$$

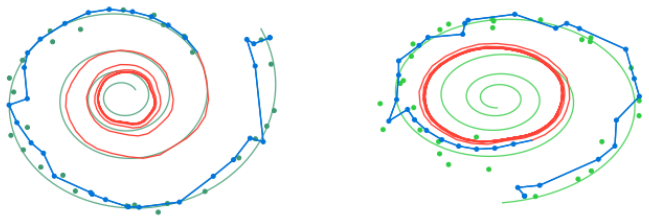
A forward Euler step!

# Neural ODEs

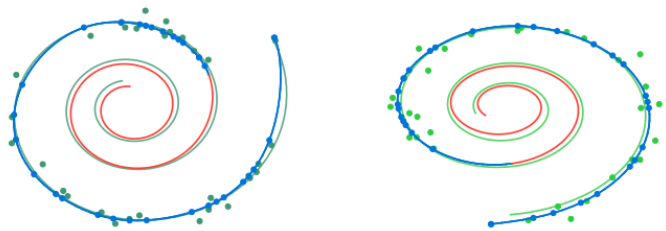
Building on residual network, let's define:

$$x_t = g(x, y)$$

and use ODE solvers to evolve $x$[13].



(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation

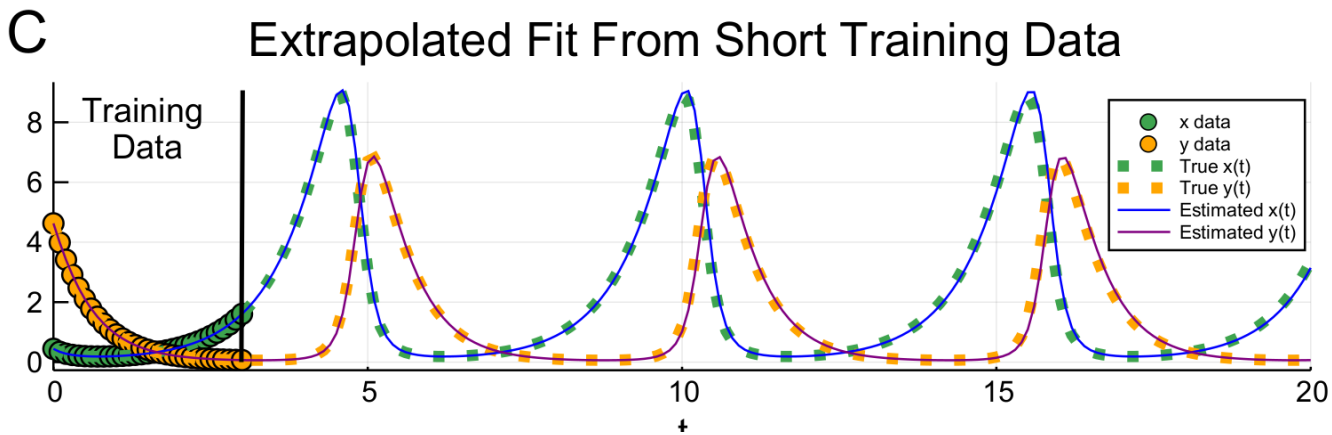
* Usually requires stiff solvers!

[13] Chen **andothers**, "Neural ordinary differential equations"

# Adding to known dynamics

What if we already have some idea of dynamics? Learn unknown dynamics as part of simulation process[14].

$$\dot{x} = \alpha x - \beta xy \qquad \rightarrow \qquad \dot{x} = \alpha x + U_1(x, y)$$
$$\dot{y} = \gamma xy - \delta y \qquad\qquad \dot{y} = -\delta y + U_2(x, y)$$




* Need to simultaneously simulate and backpropogate (chain rule) - some numerical challenges there

[14] Rackauckas **andothers**, "Universal Differential Equations for Scientific Machine Learning".

# Conclusions

- Data-driven modeling usually needs to first fit a function to data
- Neural networks are composed functions
- Can tune composed function complexity with depth, width, activation and loss functions
- Can reduce dimension for more concise dynamics
- Can formulate recurrent neural networks as ODEs
- Can embed neural networks in known PDEs to uncover missing dynamics

**Challenges:**

- Theory - stability, accuracy, training procedure etc.
- Need a lot of data
- Need a lot of patience for the architecture

## References


[1] Ben Adcock **and** Nick Dexter. "The Gap between Theory and Practice in Function Approximation with Deep Neural Networks". **in***SIAM Journal on Mathematics of Data Science*: 3.2 (**january** 2021), **pages** 624–655. ISSN: 2577-0187.

[2] Steven L. Brunton, Joshua L. Proctor **and** J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". **in***Proceedings of the National Academy of Sciences*: 113.15 (12 **april** 2016). Publisher: National Academy of Sciences Section: Physical Sciences, **pages** 3932–3937. ISSN: 0027-8424, 1091-6490.

[3] Kathleen Champion **andothers**. "Data-driven discovery of coordinates and governing equations". **in***Proceedings of the National Academy of Sciences*: 116.45 (5 **november** 2019), **pages** 22445–22451. ISSN: 0027-8424, 1091-6490.

[4] Ricky TQ Chen **andothers**. "Neural ordinary differential equations". **in***arXiv preprint arXiv:1806.07366*: (2018).

[5] Kurt Hornik, Maxwell B. Stinchcombe **and** Halbert L. White. "Multilayer feedforward networks are universal approximators". **in***Neural Networks*: 2 (1989), **pages** 359–366.

[6] Deepika Jaswal, V Sowmya **and** KP Soman. "Image classification using convolutional neural networks". **in***International Journal of Scientific and Engineering Research*: 5.6 (2014), **pages** 1661–1668.

[7] Zichao Long, Yiping Lu **and** Bin Dong. "PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network". **in***Journal of Computational Physics*: 399 (15 **december** 2019), **page** 108925. ISSN: 0021-9991.

[8] Christopher Rackauckas **andothers**. "Universal Differential Equations for Scientific Machine Learning". **in***arXiv:2001.04385 [cs, math, q-bio, stat]*: (6 **august** 2020). arXiv: 2001.04385.

[9] M. Raissi, P. Perdikaris **and** G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". **in***Journal of Computational Physics*: 378 (1 **february** 2019), **pages** 686–707. ISSN: 0021-9991.

[10] Samuel H. Rudy **andothers**. "Data-driven discovery of partial differential equations". **in***Science Advances*: 3.4 (1 **april** 2017), e1602614. ISSN: 2375-2548.

[11] Zhiwei Shi **and** Min Han. "Support Vector Echo-State Machine for Chaotic Time-Series Prediction". **in***IEEE Transactions on Neural Networks*: 18.2 (2007), **pages** 359–372.

[12] Hao Xu, Haibin Chang **and** Dongxiao Zhang. "DL-PDE: Deep-learning based data-driven discovery of partial differential equations from discrete and noisy data". **in***arXiv:1908.04463 [physics, stat]*: (12 **august** 2019). arXiv: 1908.04463.