Aspects of Machine Learning for Engineering

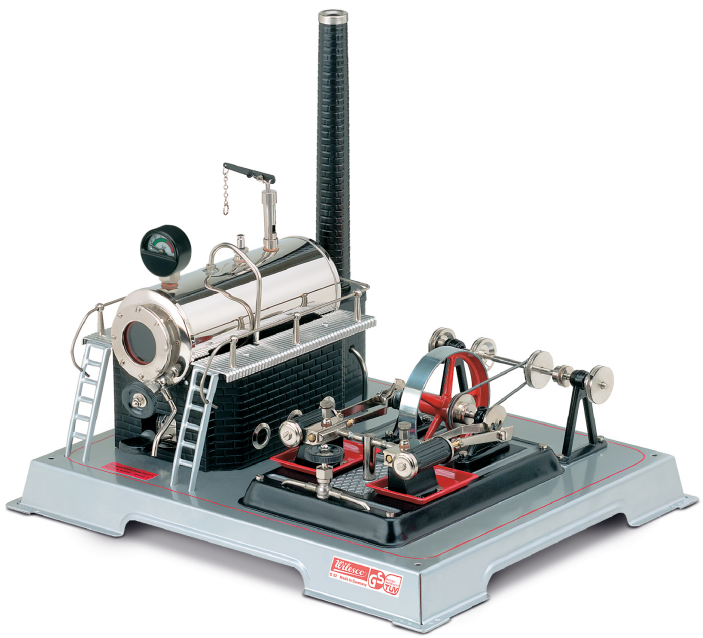
University of Warwick
19 October 2018

David K. E. Green
dgreen@turing.ac.uk
(Joint work with F. Rindler)

The Alan Turing Institute
London, United Kingdom
www.turing.ac.uk

ATI Project Goals: Estimating System Health from Streaming Sensor Data

- Quantify engineering system performance from data
- Predicting the behaviour of engineering systems
- Use sensor data to estimate preventative maintenance schedules
- AI driven reliability analysis
- Metastability based time to failure analysis
- Multidisciplinary: mathematics, statistics, computer science, engineering, electronics, physics



ML and Engineering: How to bridge the gap?

Engineering:

- Increasing data volume (sensors are cheaper).
- Increased data speed (improved sensor networking).
- Consequences of severe failures increasing (JIT economy, population density).

Application risk profile: High failure consequence requires low failure rate.

Machine learning:

- Deep Neural Networks for function approximation.
- Data processing capabilities improving.
- Ability to deal with very high dimensional spaces and large data sets.

Application risk profile: Low failure consequence tolerates high failure rate.

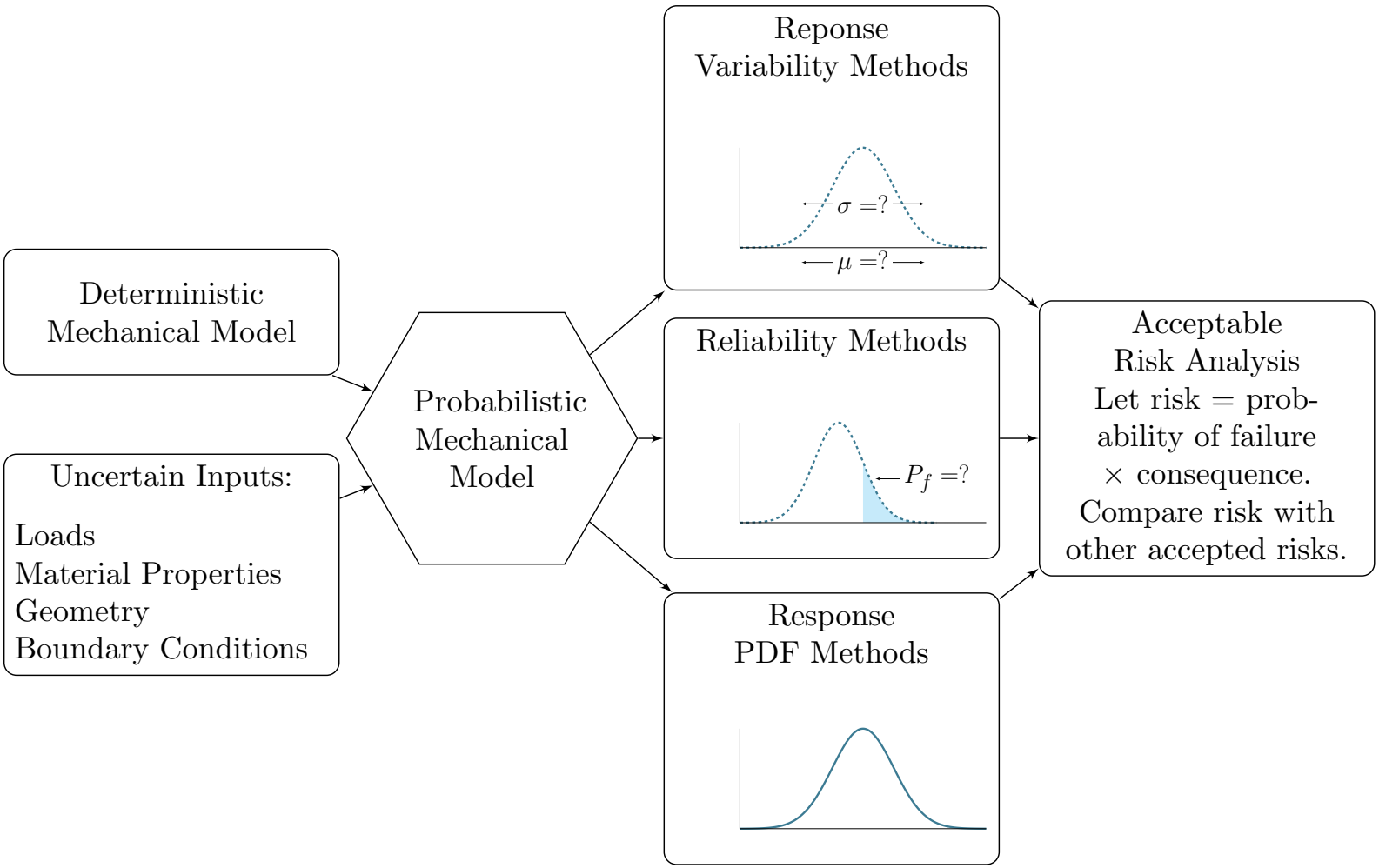
Challenge: bridging the gap while managing risk. Applying new techniques to high consequence fields.

Overview

- Preliminaries - machine learning and uncertainty quantification
- Model Inference
- Background on Neural Networks
- Model inference for ODEs
- High dimensional numerical example
- What's wrong with 'deep learning'?
- Pathfinding and metaheuristics

Preliminaries - ML and UQ

Uncertainty Quantification - Typical forward problem



Standard Uncertainty Quantification formulation

Consider a model, $y = f(x)$, with inputs, X, and outputs, Y.

Uncertainty in inputs and outputs is modelled by probability measures over X and Y.

Forward problem (model output uncertainty)

$$P(Y) = \int_X P(y|x)P(x)dx = \int_Y P(x = f^{-1}(y))dy$$

Inverse problem (inference of model parameters from data)

$$P(X) = \int_Y P(x|y)P(y)dy = \int_X P(y = f(x))dx$$

Machine Learning Models vs Uncertainty Quantification

Given:

Inputs - x and (for classification tasks) labels - y

Generative models:

Build model $P(x, y)$.
Use Bayes rule to calculate $P(y|x)$ and/or find maximum likelihood y.

Descriminative models:

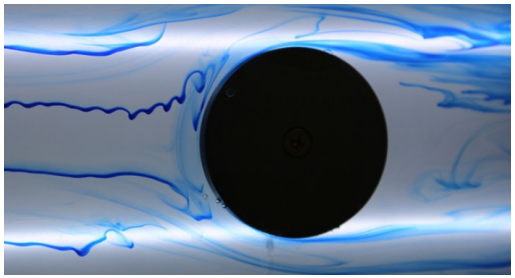
Learn $P(y|x)$ directly.
Equivalent - learn map $y = f(x)$.

Unsupervised learning:

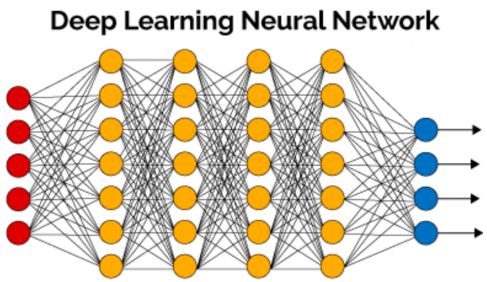
Learn $P(x)$ given observations of x

Model Inference

Types of models



Differential equations



Parametric functions

Input-output programs

Why?

- Attempt to predict future behaviour based on past data.
- Required for system health prediction.
- Purely statistical time series analysis methods interpolate well but perform poorly performance beyond observed data.
- Done well: ability to extrapolate beyond previously observed scenarios (think Newton's equations, or continuum mechanics)

Model inference in continuous spaces

- ODEs are helpful (continuous recurrence relation).
- Can wrap deterministic techniques in probabilistic techniques (e.g. Bayesian linear regression). Must first deal with the Maximum Likelihood problem.

ODE transition model Maximum Likelihood problem

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

Find f given observations of u

Model recovery

ODE transition model

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

Option 1 - Fit $u_\theta(t)$ and differentiate to recover $f(t, u(t))$

$$J(\theta) = \|u_\theta(t) - u(t)\|$$
$$f(t, u(t)) = \frac{d}{dt}u_\theta(t)$$

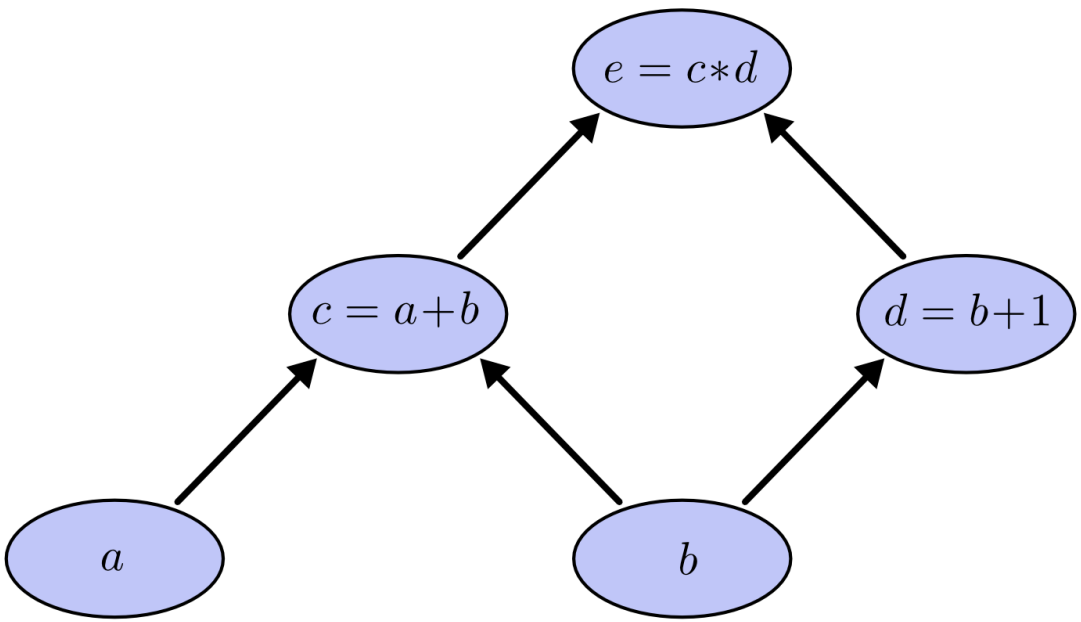
Option 2 - Fit $f_\theta(t, u(t))$ by integrating to compare to $u(t)$

$$J(\theta) = \left\| \int_t^{t+h} f_\theta(t, u(t))dt - u(t) \right\|$$

Background on Compute Graphs

Computation graphs

- Computations can be represented as graphs
- Computations without recursion: directed acyclic graphs



$$c = a + b$$
$$d = b + 1$$
$$e = c * d$$

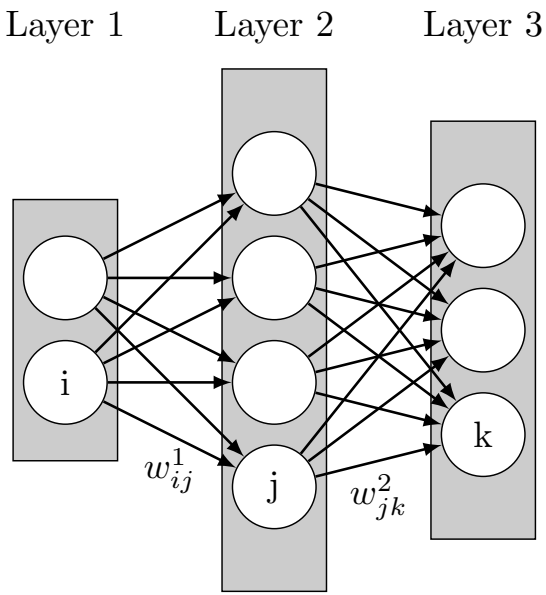
Image source: `tinyurl.com/ney26hz`

Information flows - Feedforward and Recurrent nets

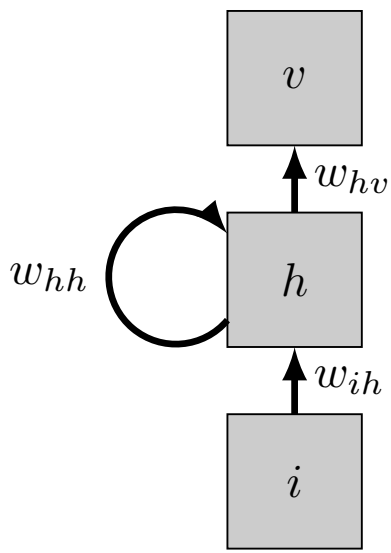
For $a_i \in \mathbb{R}^n$ and $w_i \in \mathbb{R}^{n \times m}$:

$$a^0 = x$$
$$a_j^{i+1} = \sigma^i(w_{jk} a_k^i)$$
$$h_\theta(x) = a_N$$

From layer to layer: Multiply input vector by weight matrix. Apply function $\sigma(\circ)$ to output vector.

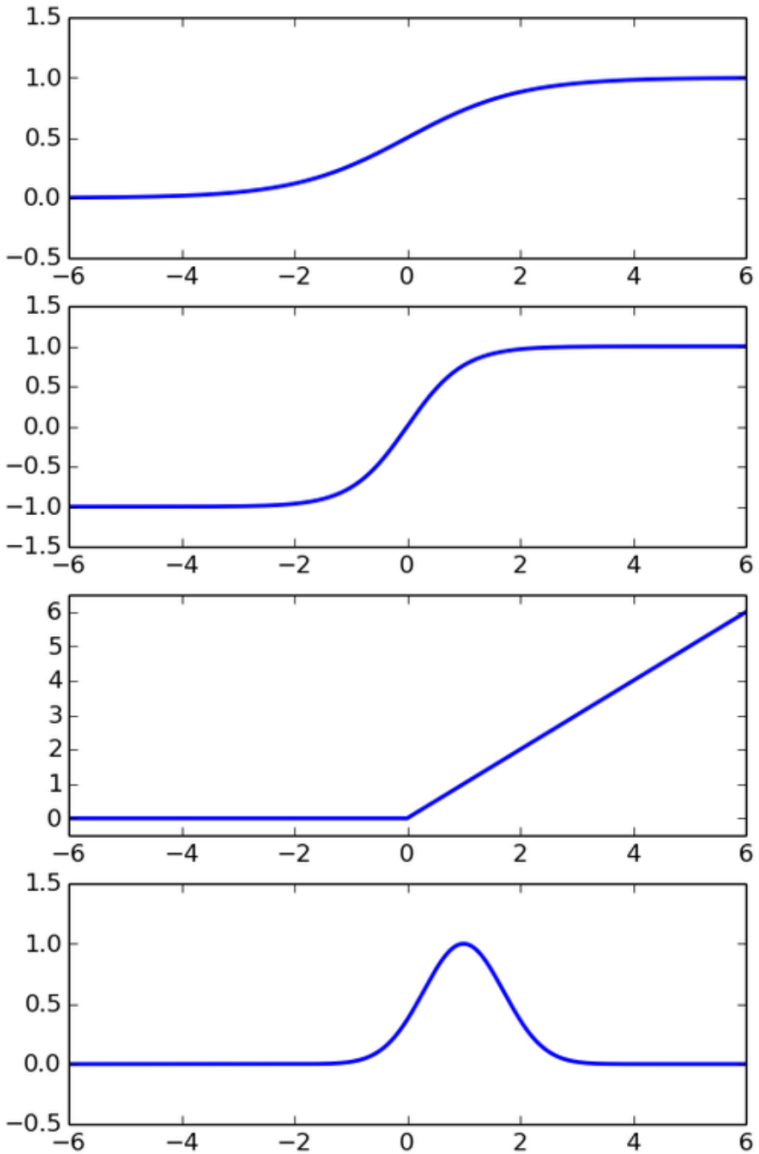


Feedforward architecture



Recurrent architecture

Neural Networks - activation functions



Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Hyperbolic Tangent

$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Rectified Linear

$$\phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

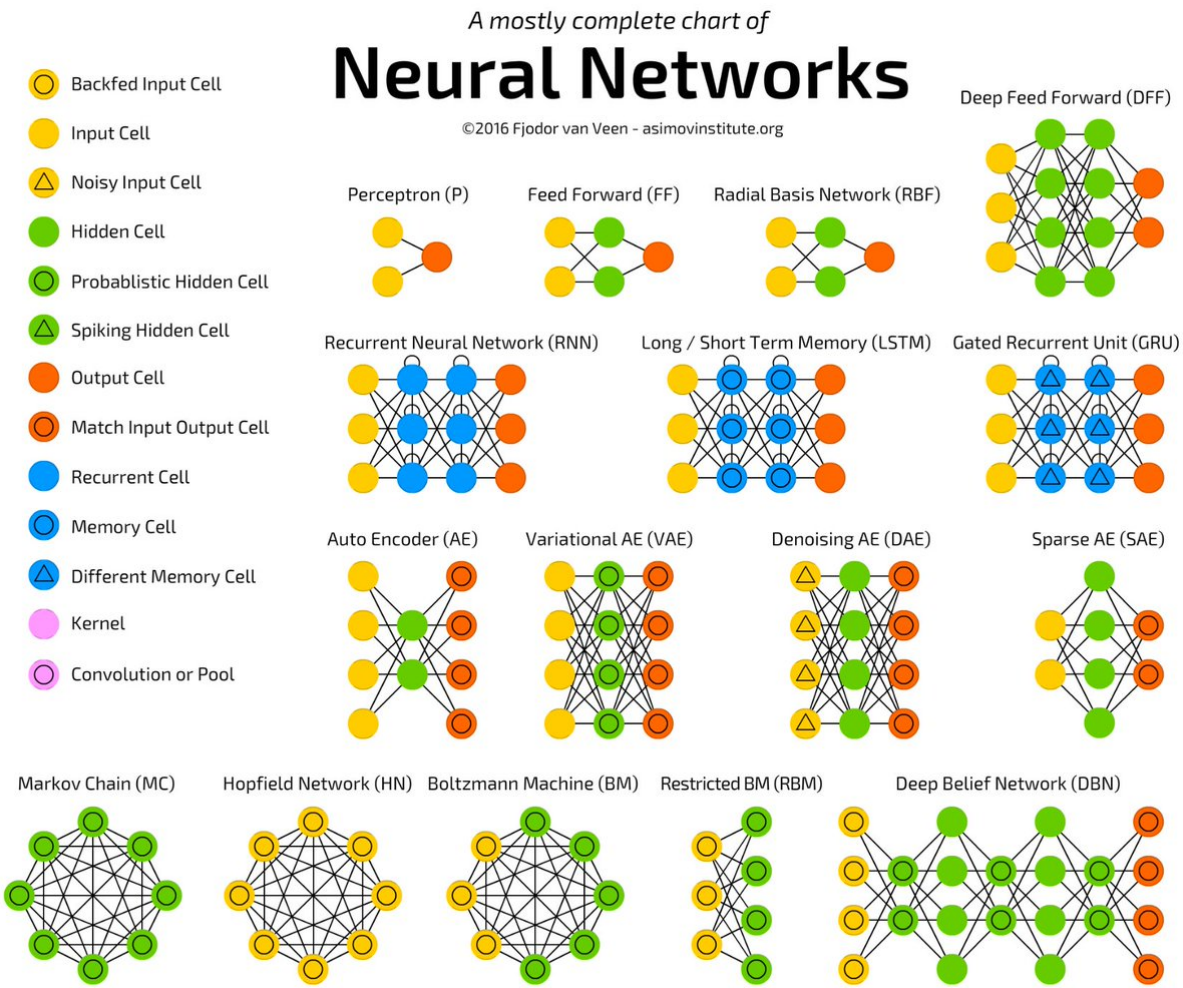
Radial Basis Function

$$\phi(z, c) = e^{-(\epsilon \|z - c\|)^2}$$

Image Source: `tinyurl.com/yc6wo6gr`

Neural Networks - different architectures

- NNs are graphs with information flows
- `http://www.asimovinstitute.org/neural-network-zoo/`
- A small sample from the full set!



Neural Networks - Training for directed graphs

Setup: Given training data samples: (x, y) (input and output).
Hypothesis space parameterised by Neural Network weights:

$$h_\theta(x) = y \text{ for } \theta \in \Theta$$

Assign loss (or error) functional:

$$J(\theta) = \|y - h_\theta(x)\| \text{ so } h^*_\theta(x) = \underset{\theta \in \Theta}{\text{argmin}}\ J(\theta)$$

Train: Minimise $J(\theta)$ by gradient descent over the training samples:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta) \text{ for all } (x, y)$$

Process: Hope that $h^*_\theta(\hat{x})$ is ok for predicting $\hat{y}$ given $\hat{x}$.

Neural Networks - Minimising $J(\theta)$

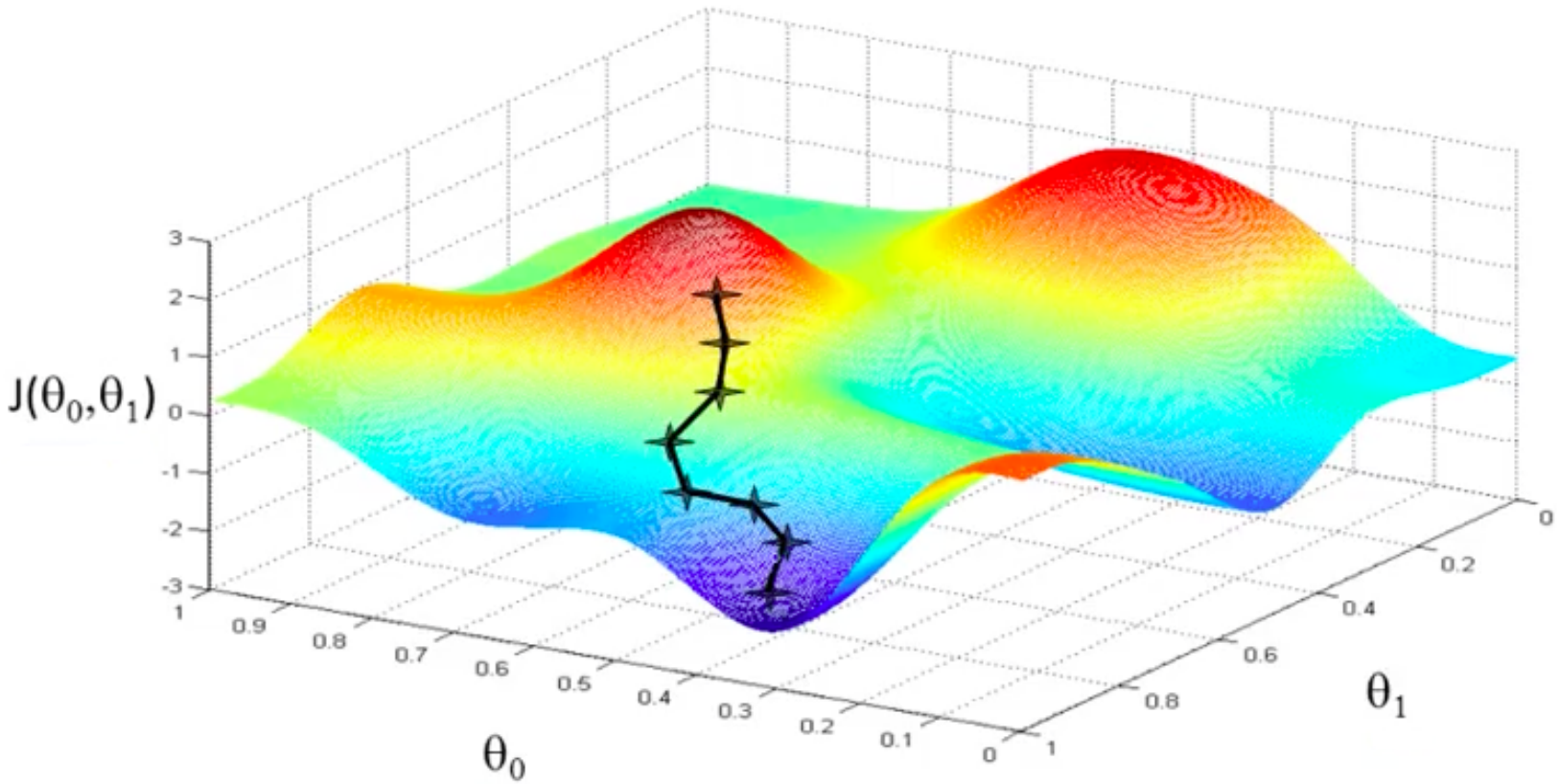
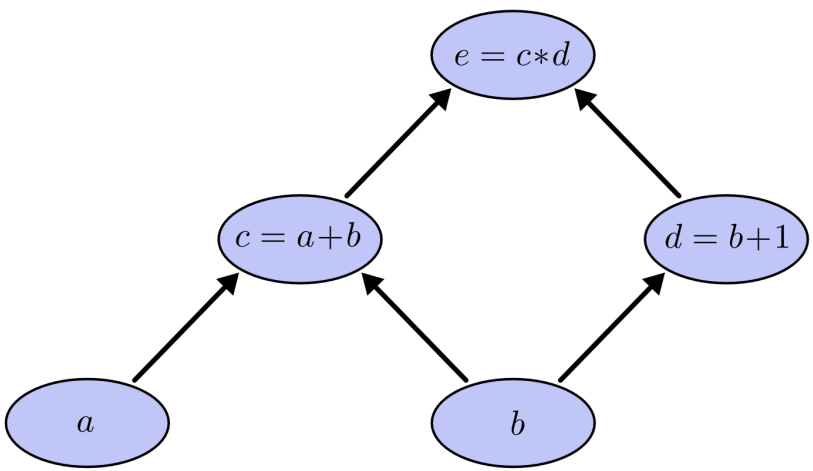


Image source: tinyurl.com/y8j9gxof

Passing gradients through Directed Acyclic Graphs

- Example from: `tinyurl.com/ney26hz`
- How to calculate $\nabla_\theta J(\theta)$?
- Can pass error gradients backward through a graph using the chain rule. This is backpropagation.
- This calculates derivative of inputs (weights) with respect to outputs (loss functions).

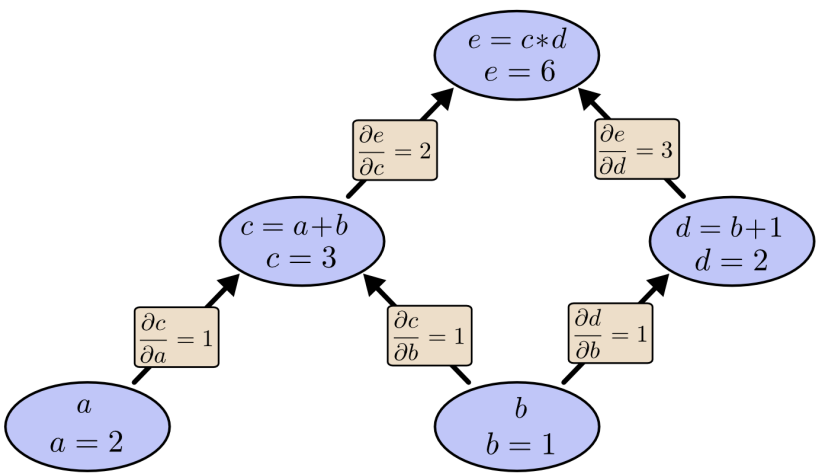


Need to calculate derivative of e at all nodes.

$$c = a + b$$
$$d = b + 1$$
$$e = c * d$$

Passing gradients through Directed Acyclic Graphs

- Example from: `tinyurl.com/ney26hz`
- How to calculate $\nabla_\theta J(\theta)$?
- Can pass error gradients backward through a graph using the chain rule. This is backpropagation.
- This calculates derivative of inputs (weights) with respect to outputs (loss functions).

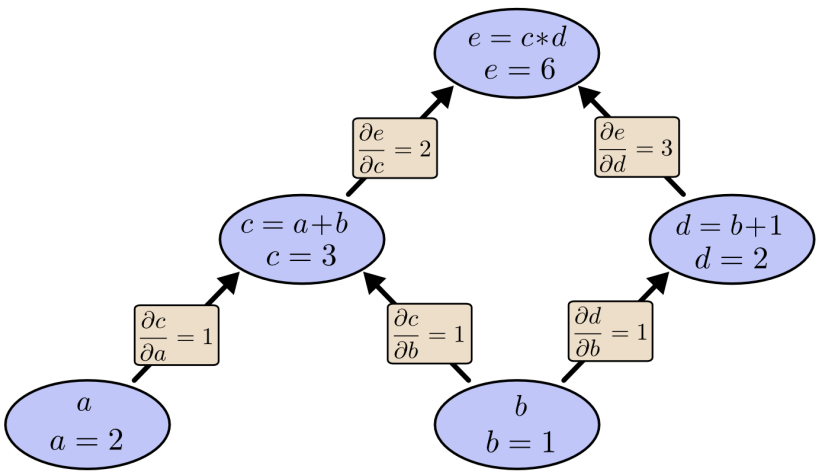


Need to calculate derivative of e at all nodes.

$$c = a + b$$
$$d = b + 1$$
$$e = c * d$$

Passing gradients through Directed Acyclic Graphs

- Example from: `tinyurl.com/ney26hz`
- How to calculate $\nabla_\theta J(\theta)$?
- Can pass error gradients backward through a graph using the chain rule. This is backpropagation.
- This calculates derivative of inputs (weights) with respect to outputs (loss functions).

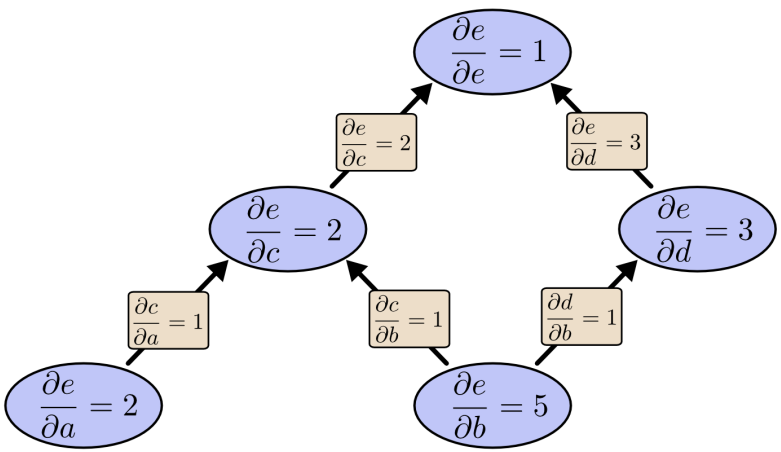


Need to calculate derivative of e at all nodes.

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial a}$$
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

Passing gradients through Directed Acyclic Graphs

- Example from: `tinyurl.com/ney26hz`
- How to calculate $\nabla_\theta J(\theta)$?
- Can pass error gradients backward through a graph using the chain rule. This is backpropagation.
- This calculates derivative of inputs (weights) with respect to outputs (loss functions).



Need to calculate derivative of e at all nodes.

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial a}$$
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

Backpropagation through ODEs

Continuous model inference

ODE transition model

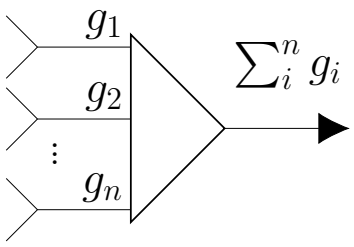
$$\frac{d}{dt}u(t) = f(t, u(t))$$

Fit $f_\theta(t, u(t))$ by integrating to compare to $u(t)$

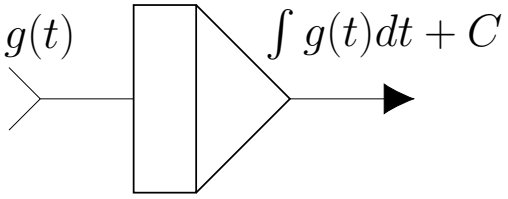
$$J(\theta) = \left|\left| \int_t^{t+h} f_\theta(t, u(t))dt - u(t) \right|\right|$$

- Technique developed for the System Health research program.
- Extends on discrete backpropagation through time technique for Recurrent Neural Networks. Developed by many e.g. Werbos 1988 `doi:10.1016/0893-6080(88)90007-x`..
- Relies on recent developments in gradient descent optimisers e.g. ADAM by Kingma 2014 `https://arxiv.org/abs/1412.6980`.

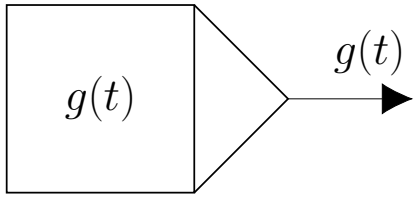
Analogue circuit representation of an oscillator ODE



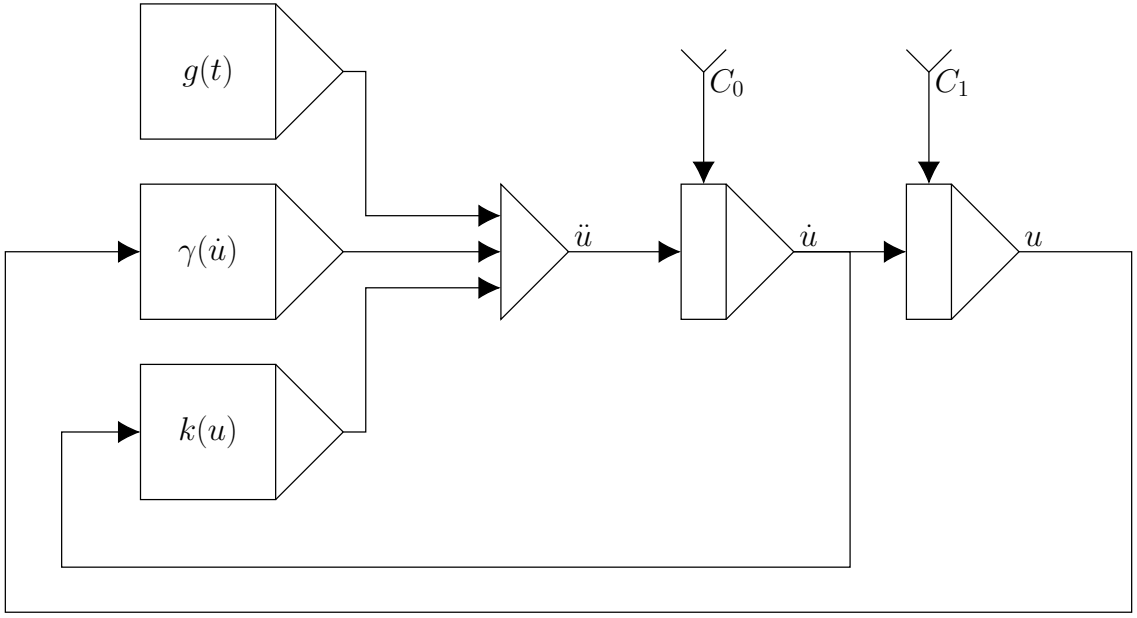
(a) Summer



(b) Integrator



(c) Function generator



$$\frac{d}{dt}\dot{u}(t) = k(u(t)) - \gamma(\dot{u}(t)) + g(t)$$

Loop unrolling: backprop through time

Backpropagation: A particular way of using the chain rule to compute $\nabla_\theta J(\theta)$. Ok as long as the network graph has no loops.

- Recurrent Neural Network: Representation of a discrete recurrence relation:

$$u(t+h) = h(u(t))$$

- Can backprop through recurrent functions by unrolling the loop to a finite order.
- This gives a directed acyclic computation graph approximation of the recursive network function.

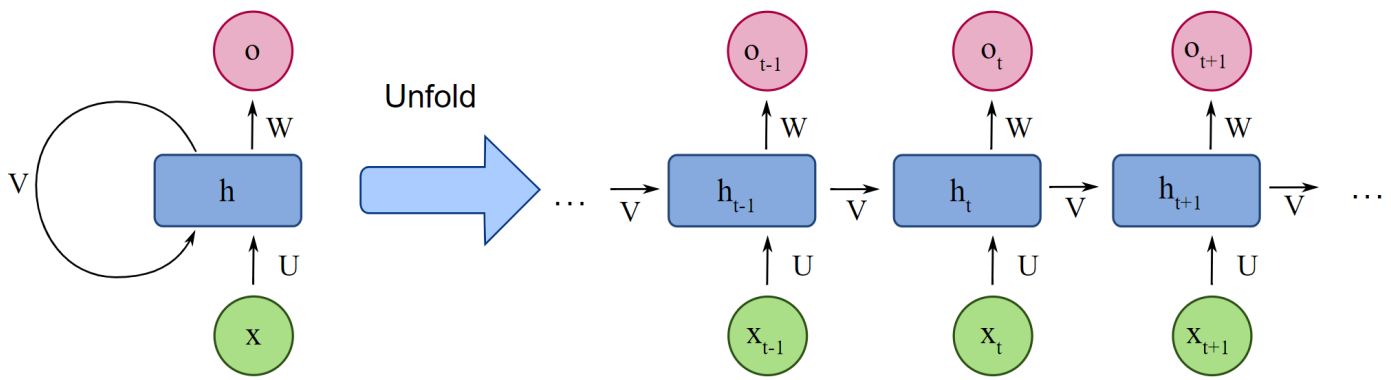


Image source: `tinyurl.com/yaewmrvj`

Example - Unforced double well potential damped oscillator

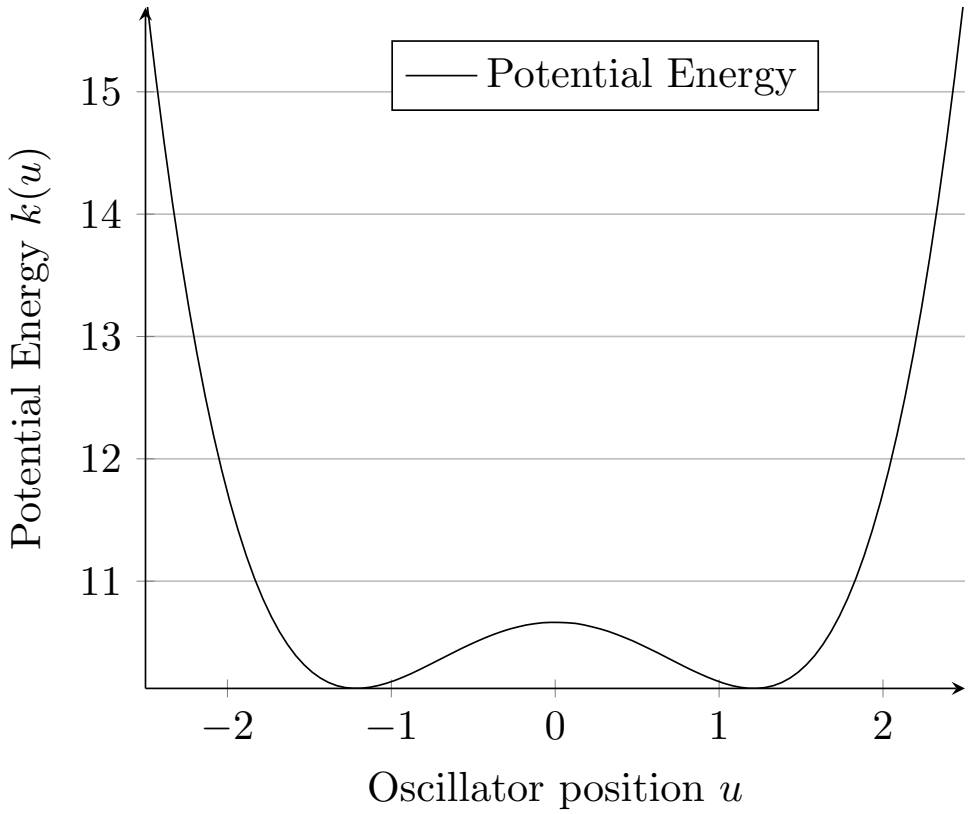
ODE Equation

$$\frac{d}{dt}\dot{u}(t) = k(u) - \gamma\dot{u}$$

$$k_\theta(u) = \sum_{i=0}^{4} \theta_i u^i$$

Given Observations $u(t)$

Task Recover $k_\theta(u)$ and parameter γ for $\theta_i, \gamma \in \mathbb{R}$.



Forward Euler example

- Define $u_i := u(t_0 + ih)$
- Integrate and approximate:

$$\int_t^{t+h} \frac{d}{dt}\dot{u}(t)dt = \int_t^{t+h} k_\theta(u(t)) - \gamma\dot{u}(t)dt$$

$$u_{i+1} - u_i = \int_t^{t+h} k_\theta(u(t)) - \gamma\dot{u}(t)dt$$

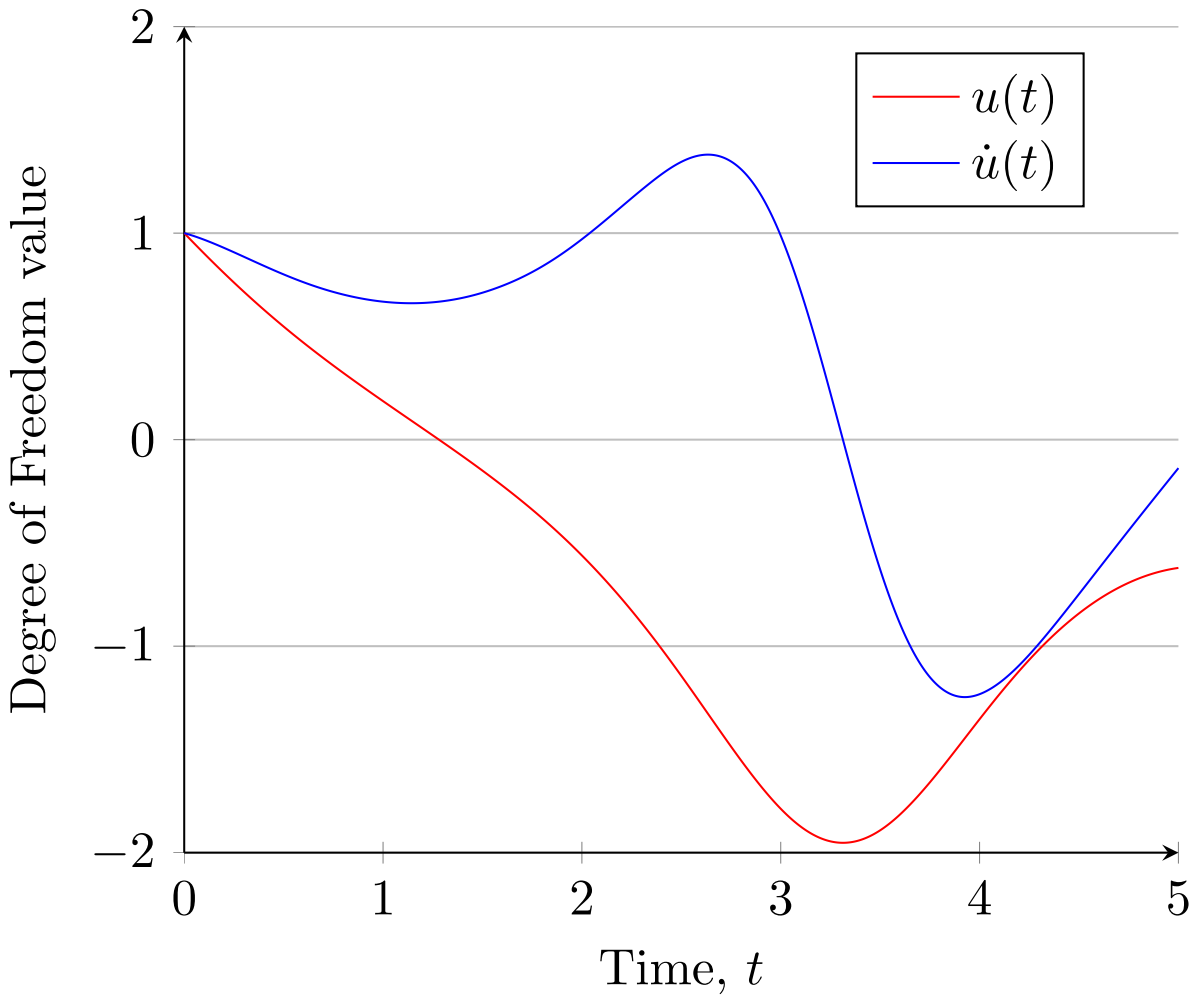
$$u_{i+1} - u_i \approx h\left(k_\theta(u_i) - \gamma\dot{u}_i\right)$$

- Prepare loss functional:

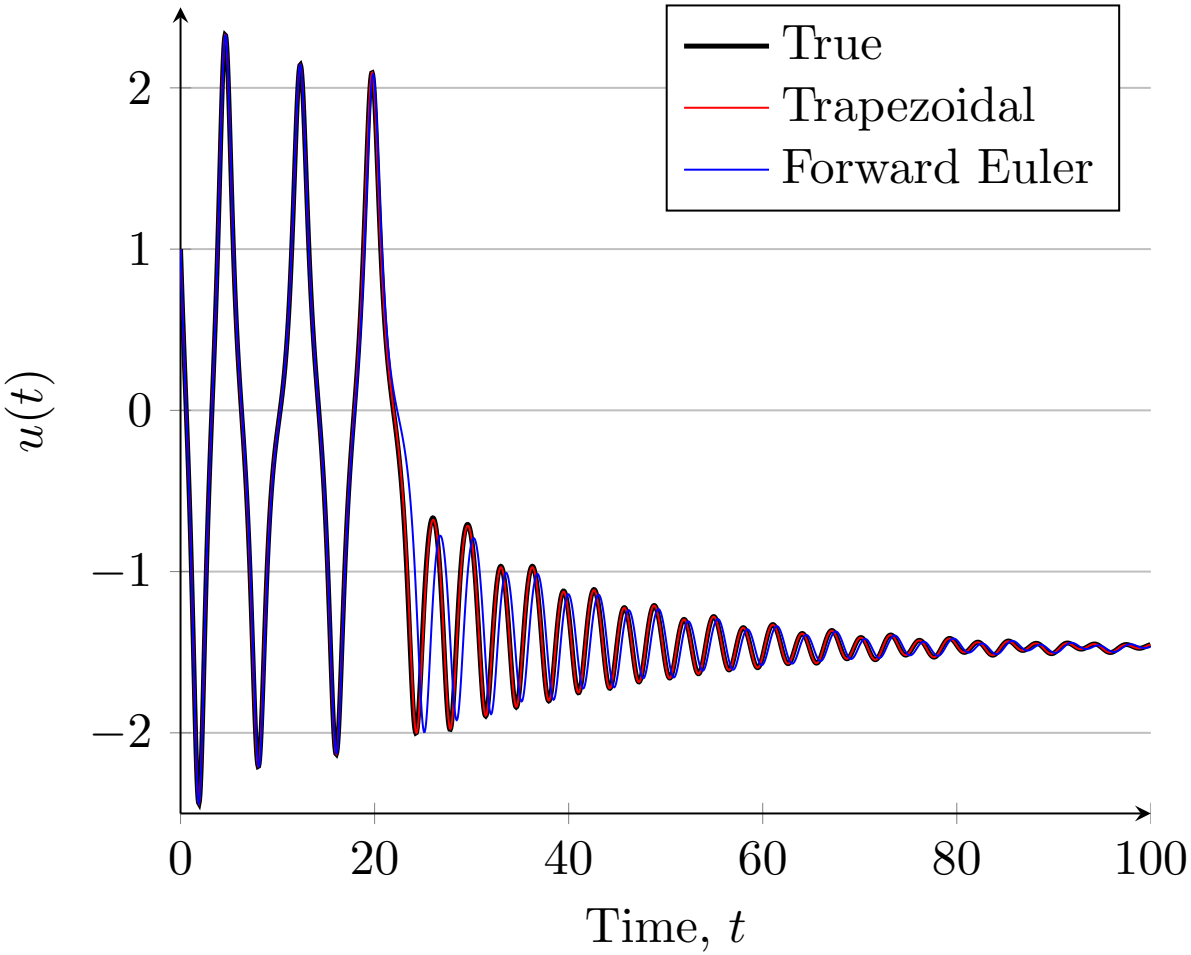
$$J(\theta) = \left|\left| \left(u_{i+1} - u_i\right) - \left(h\left(k_\theta(u_i) - \gamma\dot{u}_i\right)\right) \right|\right|$$

- Minimise $J(\theta)$ with SGD and recover θ_i and γ.
- Use recovered parameters to predict future behaviour.
- Try different integration techniques. Extension to linear multistep methods easy given evenly spaced data...

Training data



One dimensional exmaple - learnt vs true

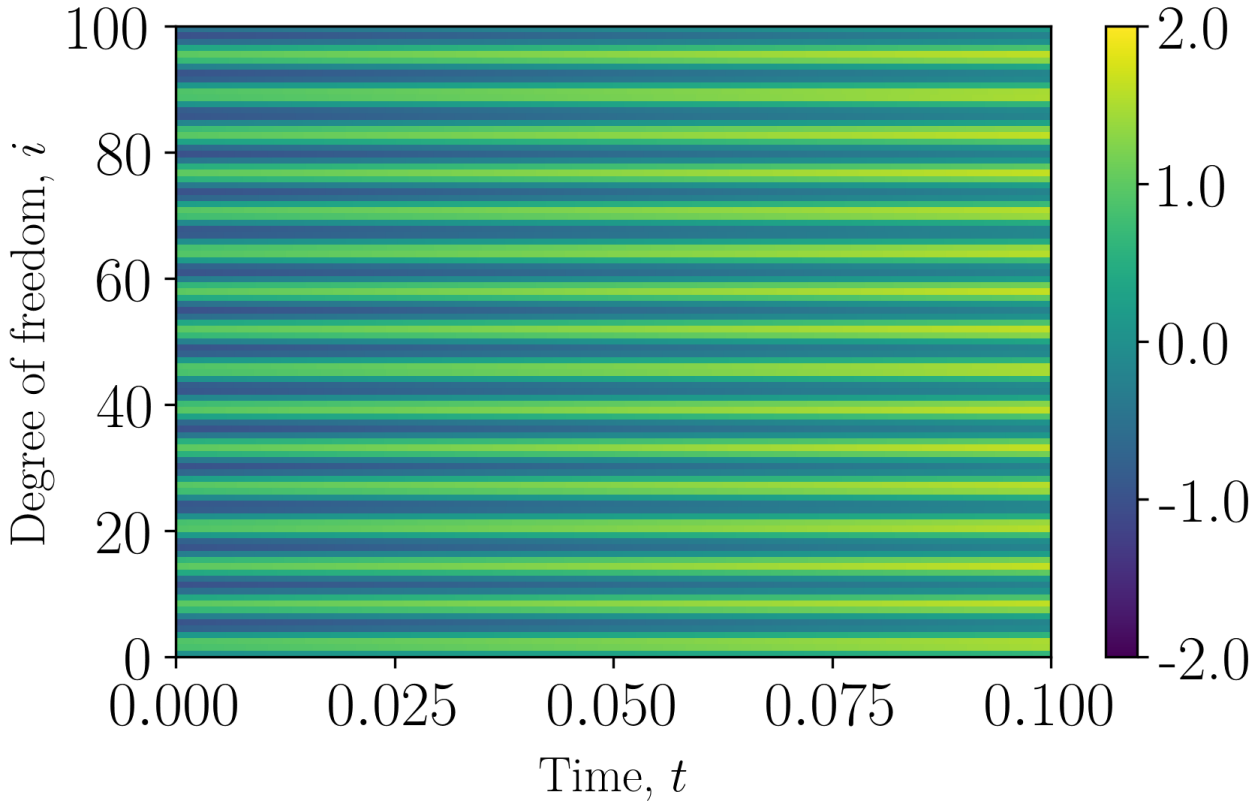


Extrapolated data for this example. Try doing this with ARMA...

Trapezoidal integrator: Forward Euler predictor with average of Forward and Backward Euler corrector.

High dimensional example

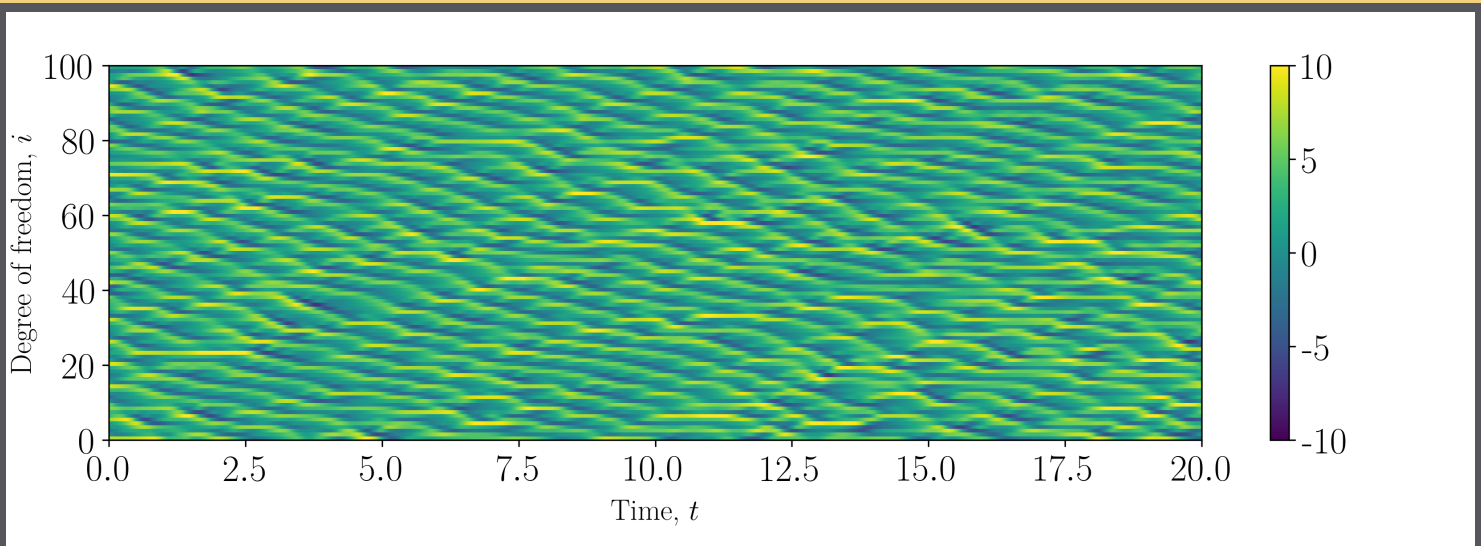
High dimensional chaotic system - training data



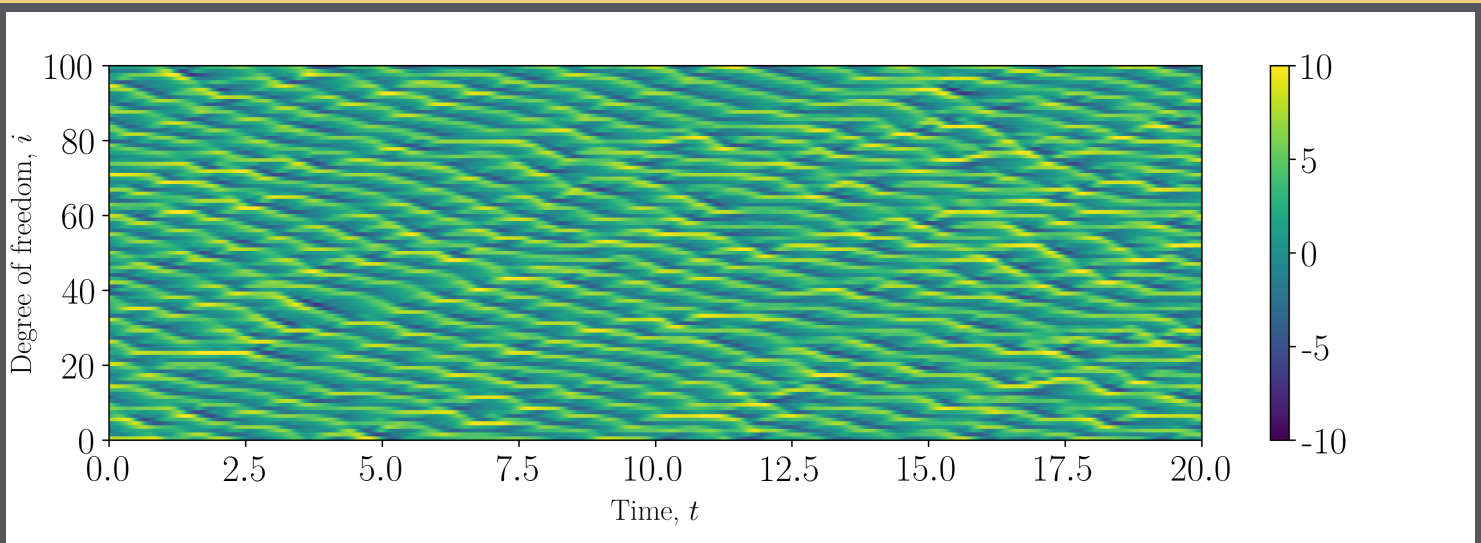
Training data for a system (polynomial in the 100 degrees of freedom) with unknown (and randomised) coefficients.

High dimensional chaotic system - prediction

True

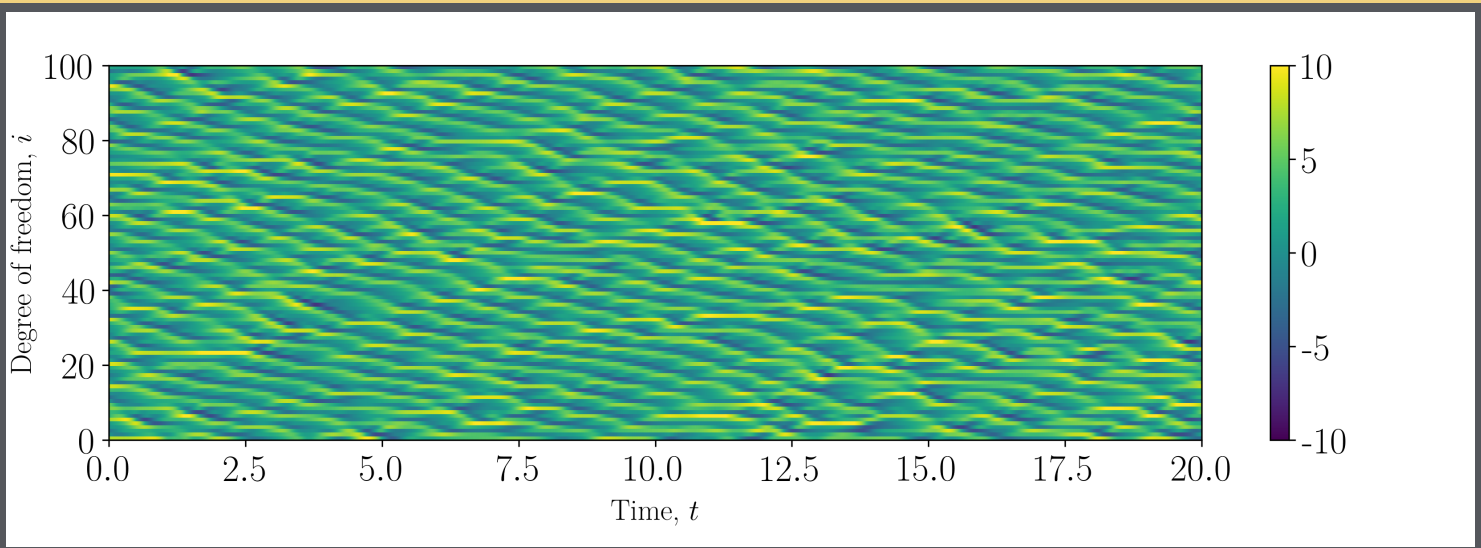


Predicted

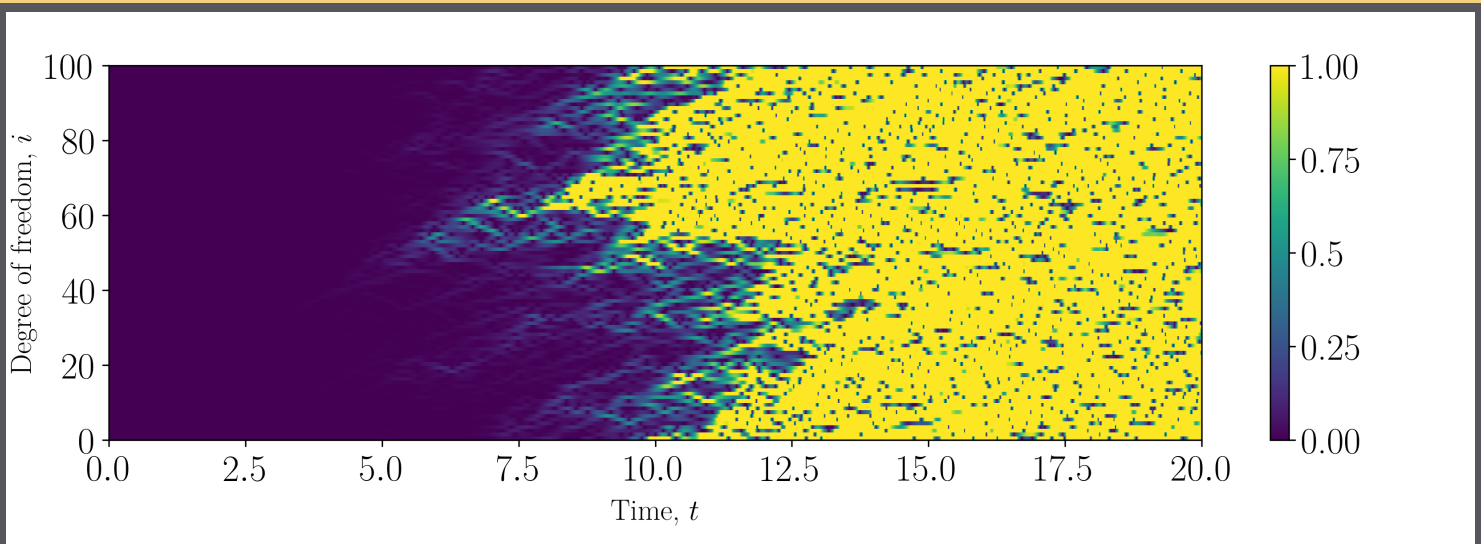


High dimensional chaotic system - prediction

True



Prediction absolute error

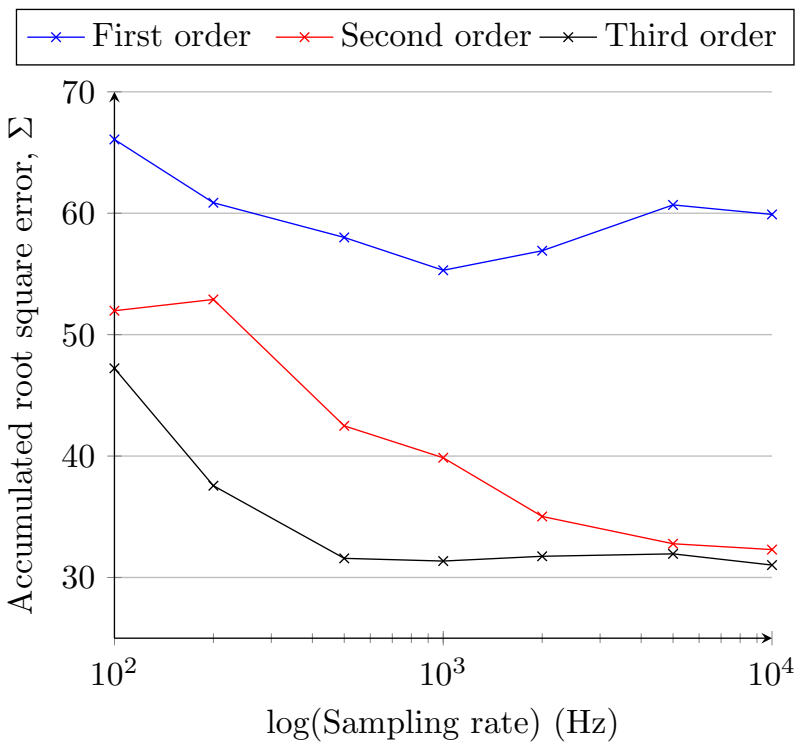


Lorenz-Emmanuel System (Very chaotic!)

ODE to learn

$$\dot{x}_i = A\left(x_{i+1} + x_{i+2}\right) x_{i-1} - Bx_i + F$$

Recovered: Coefficients A_{jk}, B_j, F



- Effect of integral estimator order and training data sampling rate on approximation error.
- Explicit Adams-Bashforth linear multistep integrators used.
- Sampling rate only helps up to a point!

Discussion on the numerical examples

What works: Can easily recovery polynomial coefficients by backpropagation through integration techniques when system is known.

Nonstationarity: A real problem for systems that degrade over time or subject to random forcing. How to deal with this nonstationarity? (Easy with repeated measurements of the same system, hard otherwise!)

Issue: Computational complexity of high dimensional, high order polynomials. State space explosion. Cannot simply fit all polynomials up to some order for high dimensional problems.

Search heuristics: How to choose what polynomials to test? What if you have very little a priori information on the model?

What's wrong with 'deep learning'?

Data driven methods

Why not try deep learning techniques? **(Spoiler: doesn't work)**

Standard: Use huge parameter space to avoid local minima.

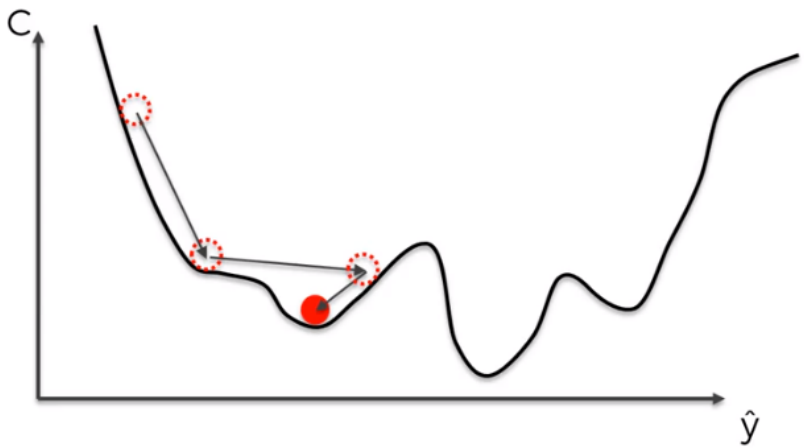


Image Source: `tinyurl.com/ycoomwnz`

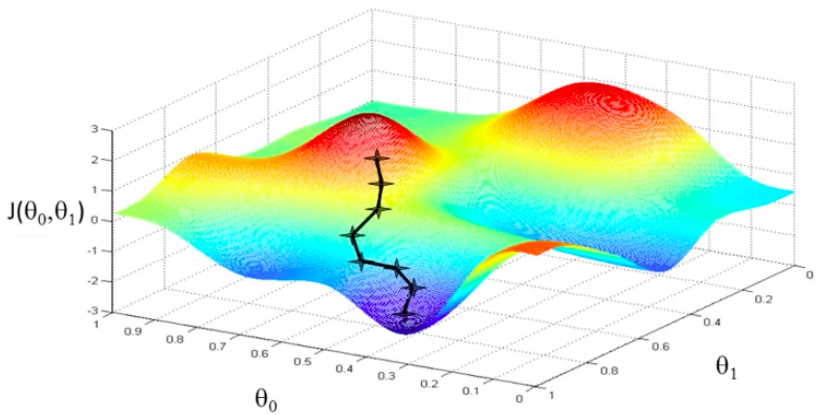


Image Source: `tinyurl.com/y8j9gxof`

What does a parametric model really predict?

Bayes rule - probability of the choice θ

$$P(\theta|D) \propto P(D|\theta)P(\theta)$$

Posterior predictive probability

$$P(h|x) = \int_{\theta} P(h|x,\theta)P(\theta|D)d\theta$$

$J(\theta)$ of a model is related to $P(h_\theta|x)$ over D by:

$$P(D|\theta) \propto \exp(-J(\theta)) = \exp\left(-\|h_\theta(x) - D\|\right)$$

Models predict measure valued functions described by the distribution over their parameters

What does a parametric model really predict?

Models predict measure valued functions described by the distribution over their parameters

- Bayesian regression - model predicts a measure over the function range.
- For models with many parameters, $P(h_\theta|x)$ will
- Can formulate more rigorously in terms of cross entropy minimisation between true process p and model q:

$$H(q|p) = -\sum_x p(x)\log(q(x))$$
$$= H(p) + D_{KL}(p\|q)$$

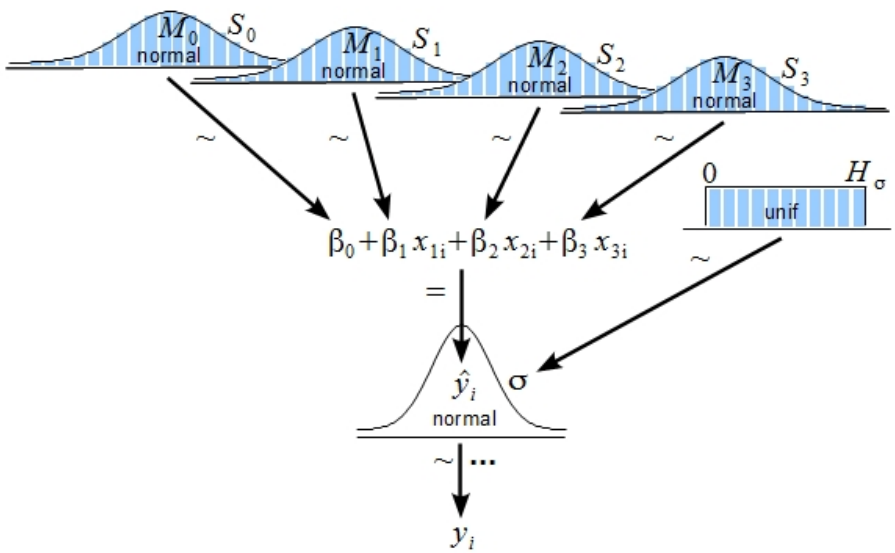


Image Source: `tinyurl.com/ya22sdx4`

Measure of a good model - why large NNs are poor models

Continuous spaces Standard RNN formulation troublesome for continuous models.

Convergence problems An LSTM RNN will not converge unless there are a very large number of parameters.

Too many parameters Many local minima means model predicts many equivalent descriptions.

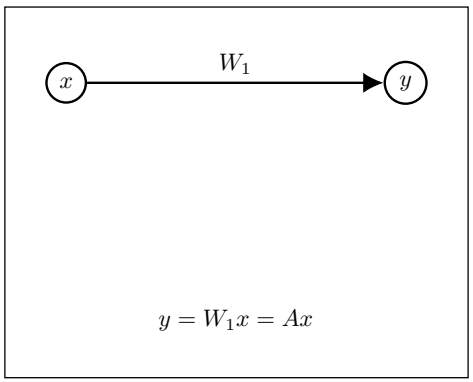
Not good for engineering For reliability engineering, need to predict states away from previously observed data points.

Occam's Razor Model is too complicated!

Conclusion Gradient descent in a massive function space is an ineffective model search technique.

Metaheuristics in function spaces

Pathfinding in function spaces - polynomial example



Function space search

Since we prefer simple models, start simple.

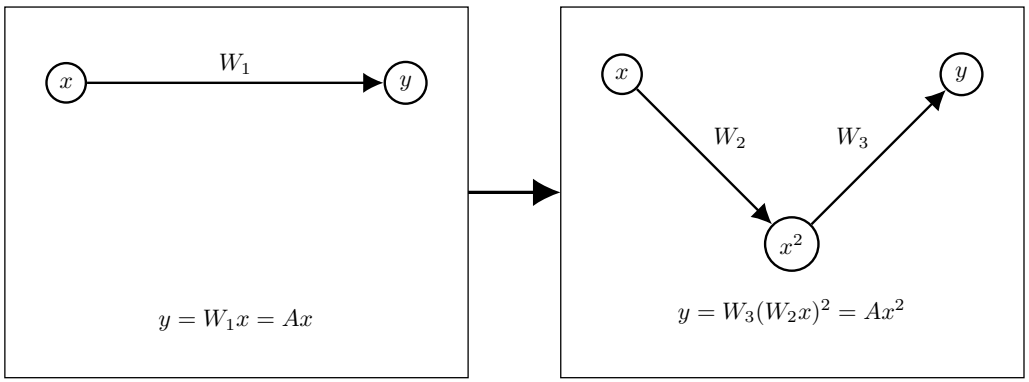
Search space grows with function complexity.

Represent functions as graphs.

Allowed to insert a node or add a link.

Assess quality of each function by regression loss after training function weights.

Pathfinding in function spaces - polynomial example



Function space search

Since we prefer simple models, start simple.

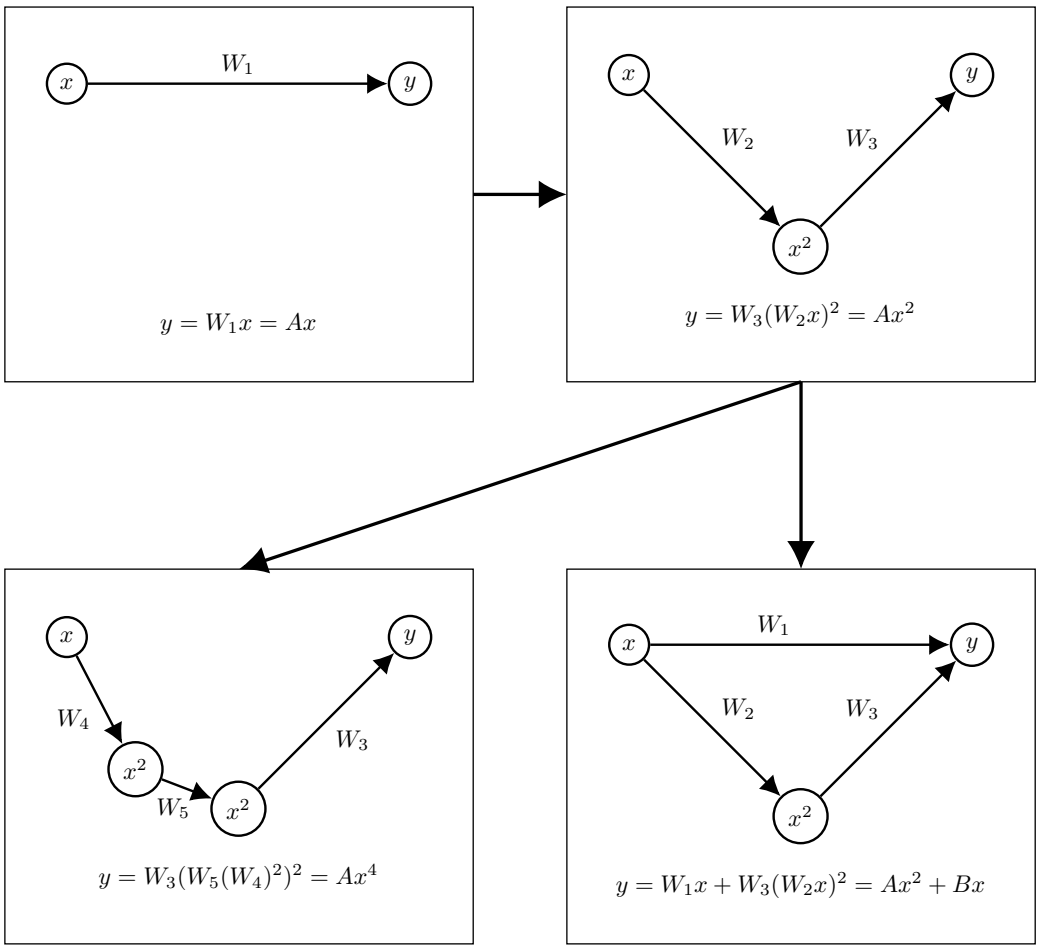
Search space grows with function complexity.

Represent functions as graphs.

Allowed to insert a node or add a link.

Assess quality of each function by regression loss after training function weights.

Pathfinding in function spaces - polynomial example



Function space search

Since we prefer simple models, start simple.

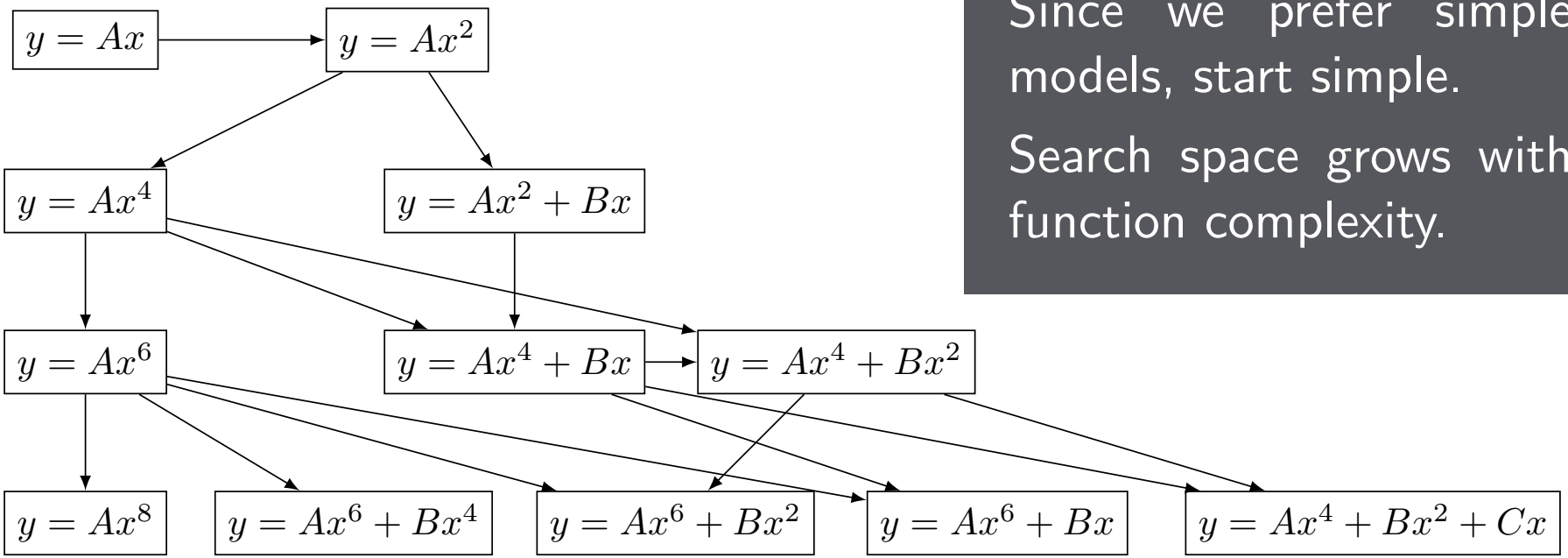
Search space grows with function complexity.

Represent functions as graphs.

Allowed to insert a node or add a link.

Assess quality of each function by regression loss after training function weights.

Pathfinding in function spaces - polynomial example



Function space search

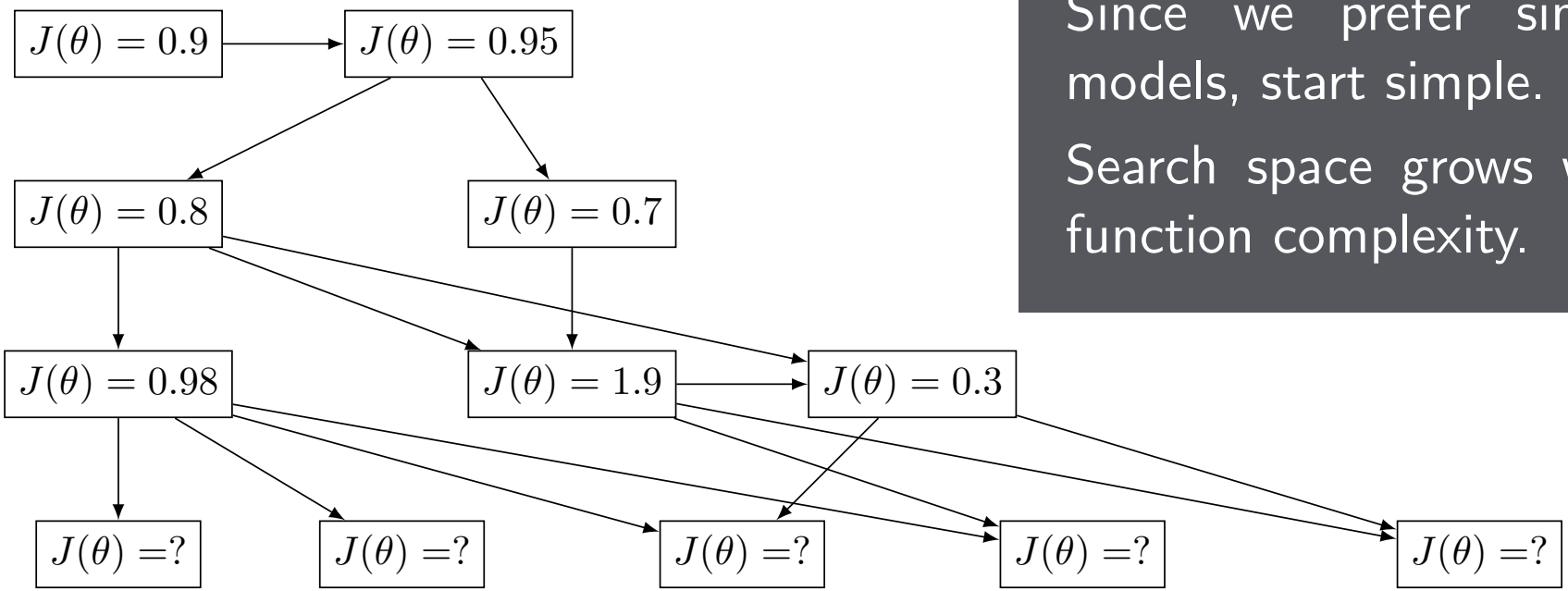
Since we prefer simple models, start simple.

Search space grows with function complexity.

Even in one dimension, state space explodes

Neural Network architecture search? Need to use some clever heuristic to search in the right places. Only expand useful nodes.

Pathfinding in function spaces - polynomial example



Function space search

Since we prefer simple models, start simple.

Search space grows with function complexity.

Even in one dimension, state space explodes

Neural Network architecture search? Need to use some clever heuristic to search in the right places. Only expand useful nodes.

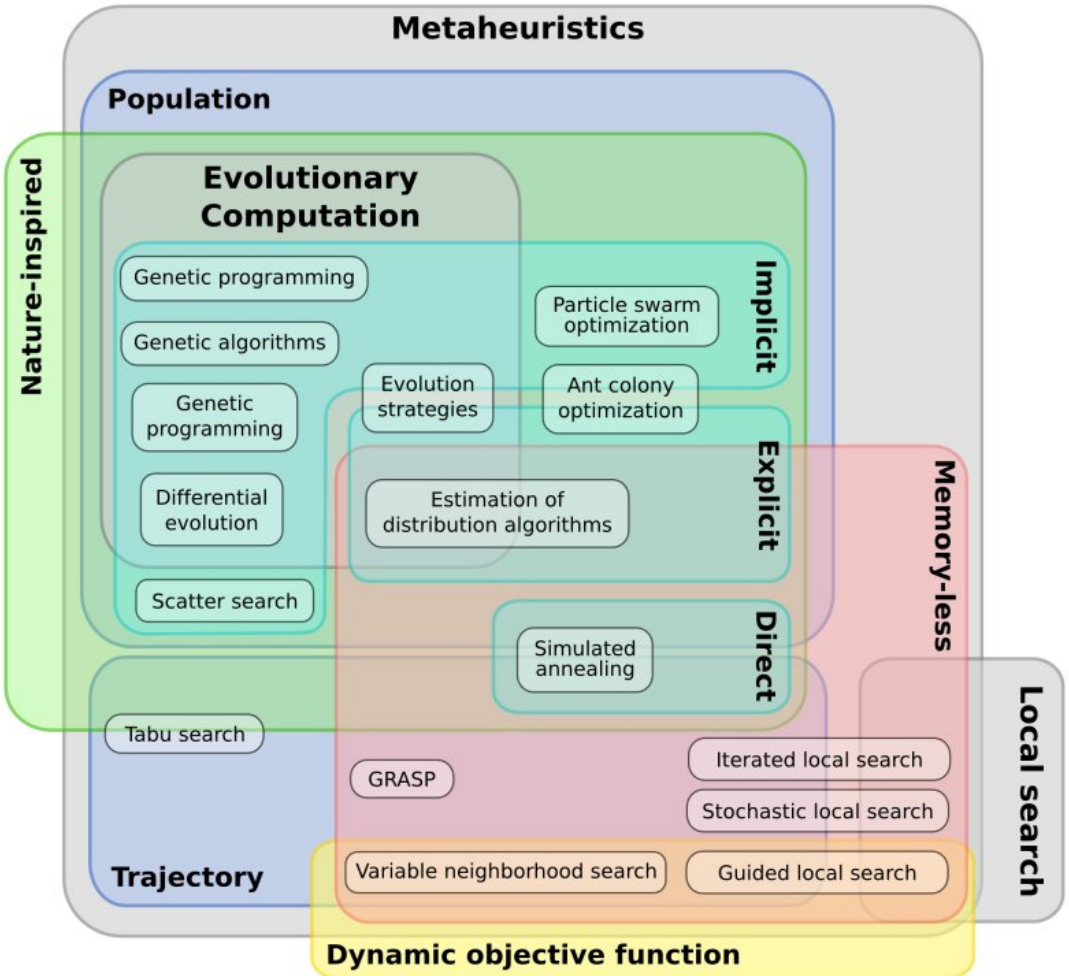
Doesn't look so bad?

Higher dimensional polynomial search blows up MUCH more quickly (for n variables with maximum degree k):

$$\binom{n+k}{k} = \frac{(n+k)!}{k!n!}$$

If additional activation functions are used (as in Neural Networks), state space branching factor is higher. Exponentially harder!

Metaheuristic algorithms



- Genetic algorithms: Pathfinding based on parent regression loss.
- Perturbative approximations like sparse Polynomial Chaos are just another metaheuristic.
- Open ended spaces cause Monte Carlo Tree Search and Reinforcement Learning to fail.
- AlphaZero only works because certain games have a finite search space.

Image Source: `tinyurl.com/yc44bjoh`

Conclusions

Conclusions

Background on engineering, machine learning

Short introduction to directed compute graphs

Model inference for ODEs

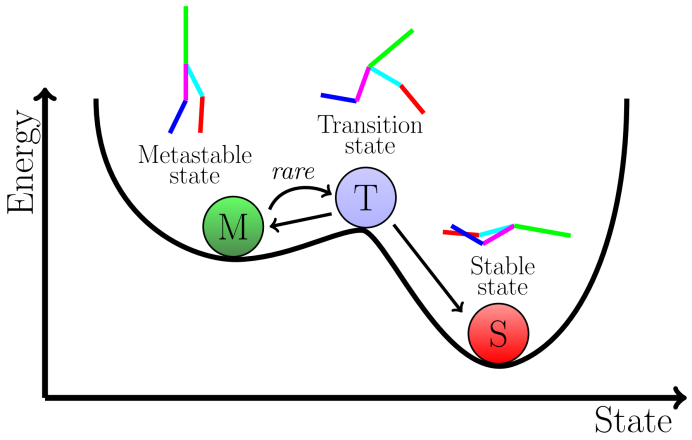
Problems when searching for an unknown model

The need for new metaheuristics

Thank you!

Slide appendix

System robustness and health



tinyurl.com/y6v4lcop

Energy decay

Need to consider how energy surface changes over time. Description of $P(s_{t+1}|s_t)$ fine, but metastability analysis depends on $\mathcal{L}$...

Robustness Depth of metastable basin.

Time to failure Time to hit transition state.

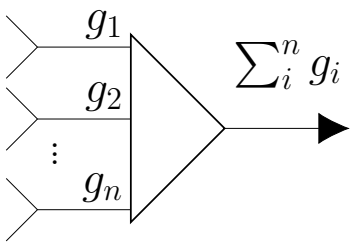
Health Expected future value of robustness and time to failure.

Efficient descriptions Describe system in terms of basins, rather than all states (use SVD/PCA/POD).

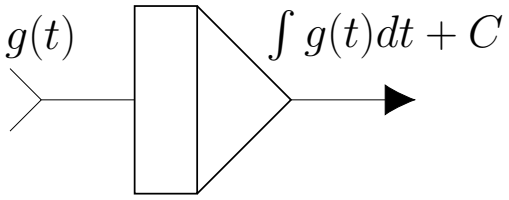
Utility Assign to basins, rather than to all states.

Efficiency As a utility measure.

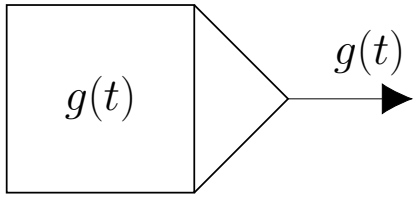
Analogue circuit representation of an oscillator ODE



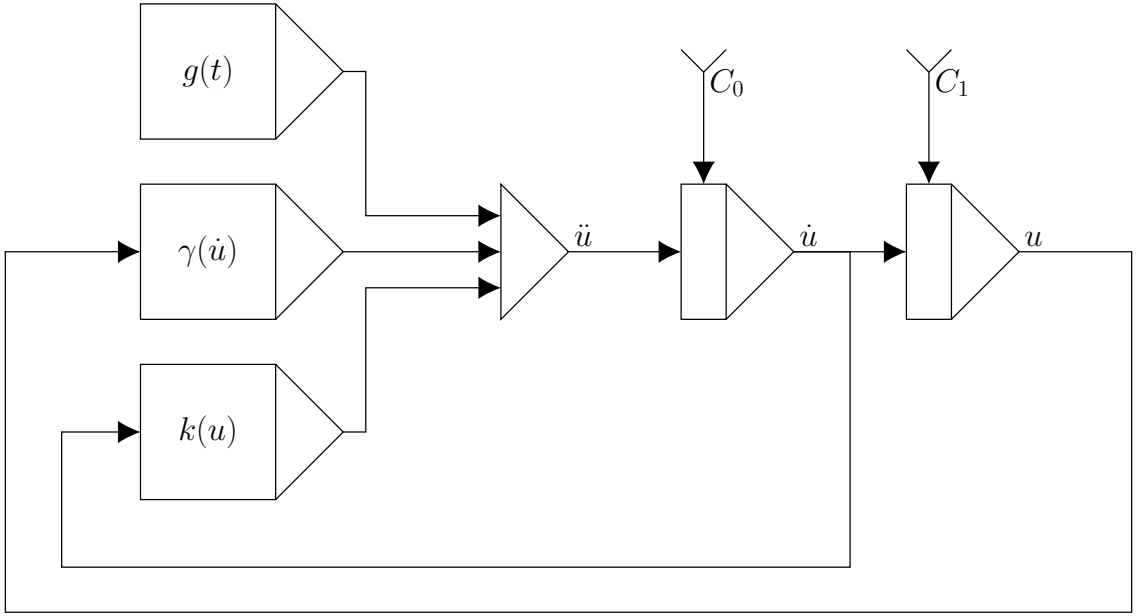
(a) Summer



(b) Integrator



(c) Function generator



$$\frac{d}{dt}u(t) = k(u(t)) - \gamma(\dot{u}(t)) + f(t)$$

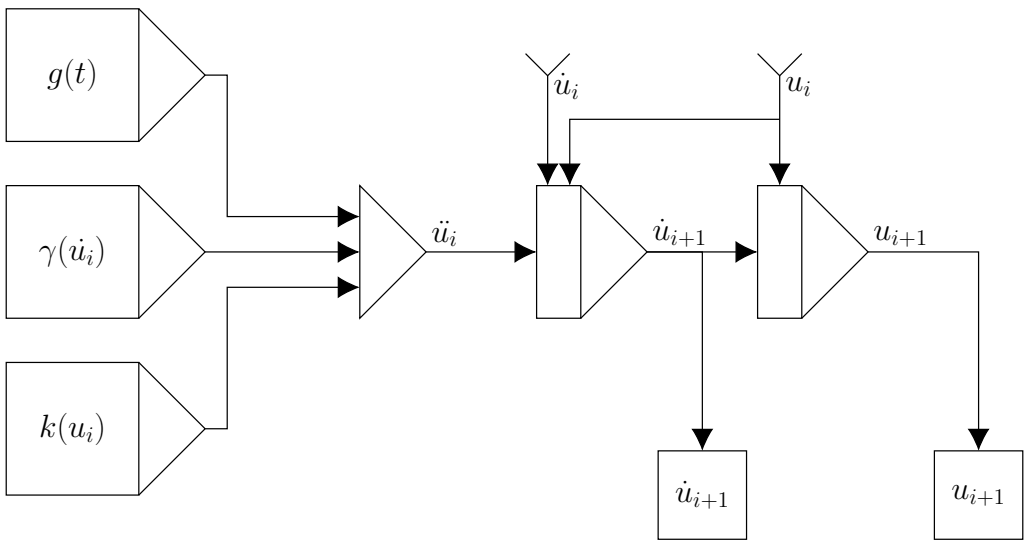
Backprop through ODEs

- Replace integrators with numerical approximations
- ODE circuit is unrolled in time and over the integrators!

Forward Euler

$$u_{i+1} - u_i = \int_t^{t+h} f(t, u(t))dt \qquad u_i := u(t_0 + ih)$$

$$u_{i+1} \approx u_i + hf_i$$



$$\frac{d}{dt}u(t) = k(u(t)) - \gamma(\dot{u}(t)) + f(t)$$

Optimal model encodings

A model, q should optimally encode the process, p, it is describing

In terms of cross entropy:

$$H(q|p) = -\sum_x p(x) \log(q(x)) = H(p) + D_{KL}(p\|q)$$

where:

$H(p) = -\sum_x p(x) \log p(x)$ is the entropy of $p(x)$

$D_{KL}(p\|q) = -\sum_x p(x) \log \frac{q(x)}{p(x)}$ KL divergence of q from p

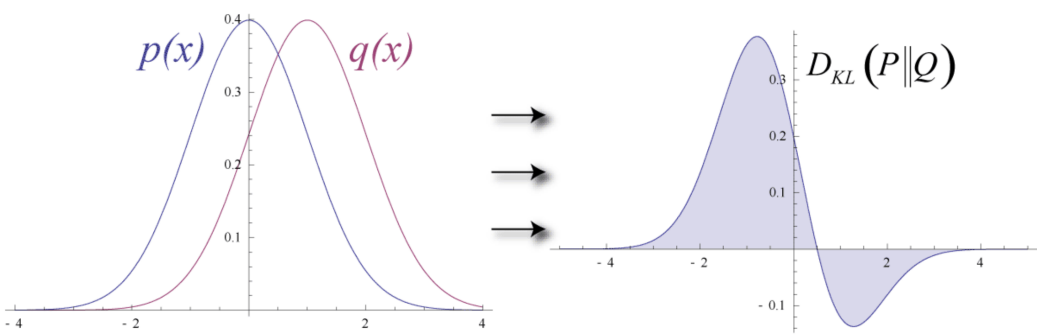


Image Source: `tinyurl.com/y79y44dl`

Backprop is the chain rule

- Use the chain rule to calculate SGD derivatives $\nabla_\theta J(\theta)$.
- Requires: ANN functions are differentiable
- Layered deep neural network structure has a convenient form for backprop.