Tangent-Space Regularization for Dynamical System Modeling using Neural Networks

Fredrik Bagge Carlson, Anders Robertsson, Rolf Johansson

Lund University, Department of Automatic Control

ISMP Bordeaux, July 2018

Introduction

Goal

Efficient (in terms of time and data) learning of black-box dynamics models.

Previous research

Plentiful.

This talk

Smart regularization.

Introduction

Dynamical control systems are often described by differential state-equations

$$\dot{x}(t) = f_c\big(x(t), u(t)\big)$$

where x is the state, u is the input

Example – Robot

$$\ddot{x} = -M^{-1}(x)\big(C(x,\dot{x})\dot{x} + G(x) + F(\dot{x}) - u\big)$$

Discretization (sampling) leads to

Objective 1

Learn the function f

$$x_{t+1} = f\big(x_t, u_t\big)$$

Classical system identification

Linear models	+ Easy to fit
	+ Easy to interpret
	- Restrictive
Grey box models	+ Easy to interpret
	+ Arbitrary complexity
	- Requires insight
Black box models	+ Arbitrary complexity
	- Can easily overfit
	- Hard to interpret
	+ Can be used on anything

Introduction

Sampling of f_c to f changes the eigenvalues of the Jacobian

- Eigenvalues at 0 (integrators) are moved to 1.
- Eigenvalues at $-\infty$ are moved to -1.
- Eigenvalues at the imaginary axis are moved to unit circle.

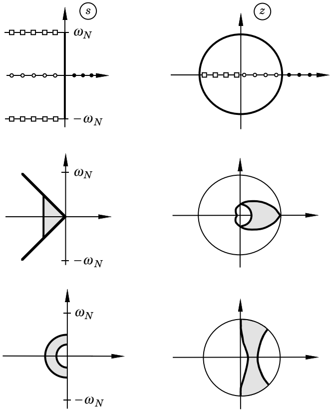


Figure: The conformal map $z = \exp(sh)$.

Introduction

Alternative form of discrete-time system

Objective 2

Learn the function g

$$x_{t+1} - x_t = g(x_t, u_t)$$

- $f \rightarrow g$ corresponds to high-pass filtering of the target sequence
- A neural network with randomly initialized weights tend to have a Jacobian with eigenvalues close to 0
- $$\frac{x_{t+1} - x_t}{h} = \frac{g(x_t, u_t)}{h}$$

  is a finite-difference approximation of derivative[1]
- Eigenvalues of g a constant multiple of f_c as sample time increases

[1] R Middleton and GC Goodwin. "Improved finite word length characteristics in digital control using delta operators". In: *IEEE transactions on automatic control* (1986).

Outline

- Regularization in the Jacobian space of f/g
- How does weight decay affect learning of f and g
- Trainability of f and g

Tangent space regularization

Many systems of practical interest for control are *smooth*

Smoothness implies that the Jacobian of the systems changes slowly along a trajectory

Tangent space regularization

Penalize

$$\sum_t \left\| \hat{J}_f(t+1) - \hat{J}_f(t) \right\|_2^2$$

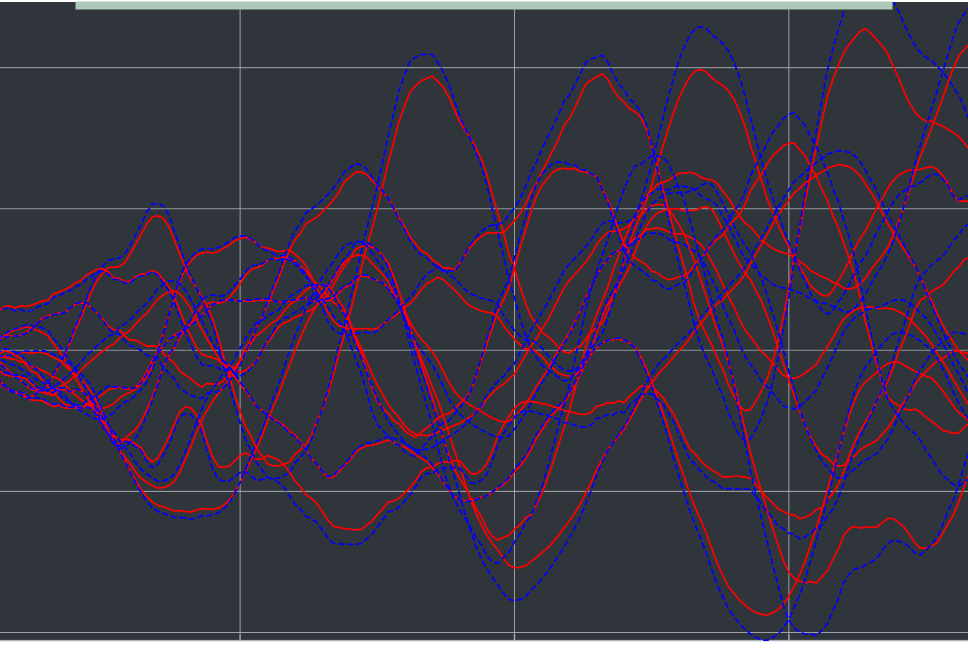
Implementation

In practice, adding $\sum_t \left\| \hat{J}_f(t+1) - \hat{J}_f(t) \right\|_2^2$ to the cost function might not be supported by the neural-network software.

1. Estimate LTV model $x_{t+1} = A_t x_t + B_t u_t$ with regularization term $\sum_t \left\| \hat{J}_f(t+1) - \hat{J}_f(t) \right\|_2^2$ using dynamic programming[2]
2. Sample $x_t, u_t,\ A_t x_t + B_t u_t$ around trajectory
3. Add samples to training set
4. Corresponds to sampled finite-difference approximation

[2] Fredrik Bagge Carlson, Anders Robertsson, and Rolf Johansson. "Identification of LTV Dynamical Models with Smooth or Discontinuous Time Evolution by means of Convex Optimization". In: *IEEE International Conference on Control and Automation (ICCA)*. 2018.

Linear system simulation



Linear system simulation

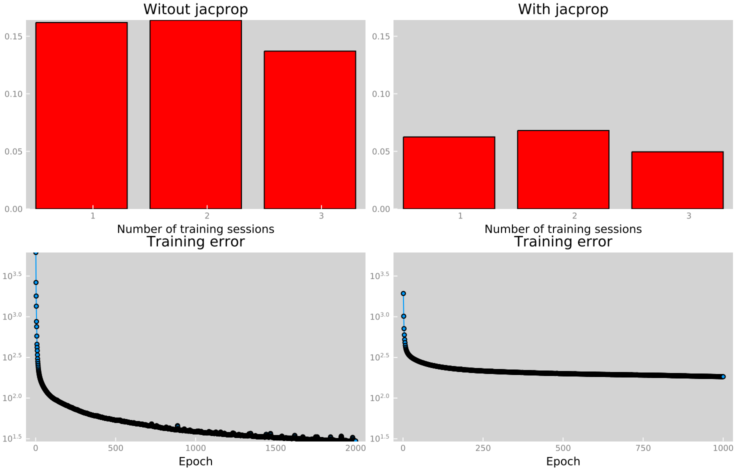
1. Generate a random, stable, linear system with low damping

$A_0 =$	10×10 matrix of random coefficients
$A = A_0 - A_0^\top$	skew-symmetric = pure imaginary eigenvalues
$A = A - \Delta t\, I$	Make 'slightly' stable
$A = \exp(\Delta t\, A)$	discrete time
$B =$	random coefficients

2. Sample trajectory $\tau = \{x_t, u_t\}$ where $u \sim N(0, \sigma_u)$
3. Train neural network (20 hidden, single layer) using τ
4. Repeat 2. and add to dataset

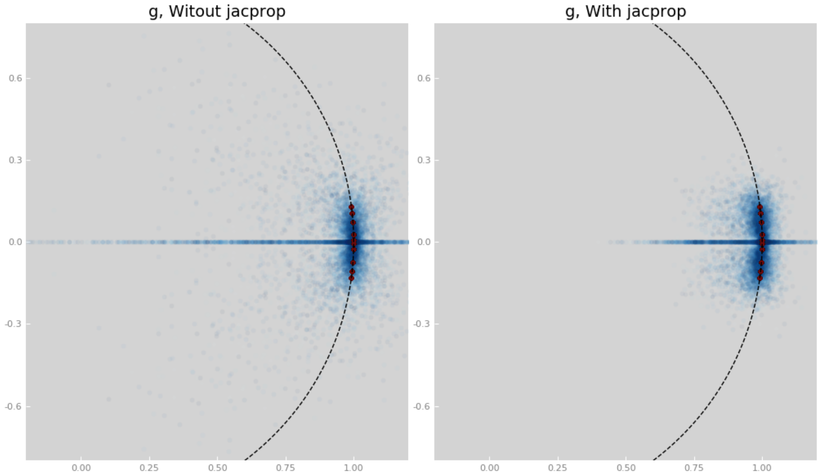
Tangent space regularization

- Lower training error without jacprop, but high error on Jacobian.
- Jacprop prevents this overfitting.



Tangent space regularization

Learned Jacobian eigenvalues for randomly sampled points in the state-space. Jacprop leads to better estimation of the Jacobian

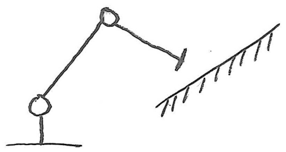


Nonlinear system

Testing on a linear system is cheating...

2DOF robot arm

- Nonlinear dynamics
- Non-smooth friction
- Lowpass filtered Gaussian input

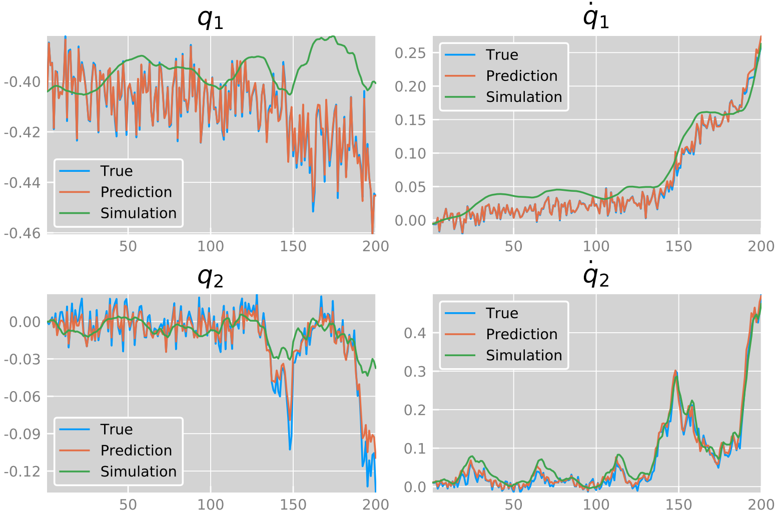


Tests

- Prediction RMSE
- Simulation RMSE
- Errors in Jacobian

Nonlinear system

Example performance[3] – Validation trajectory



[3] g with jacprop, without weight decay, 30 hidden, 3 training trajectories

Nonlinear system

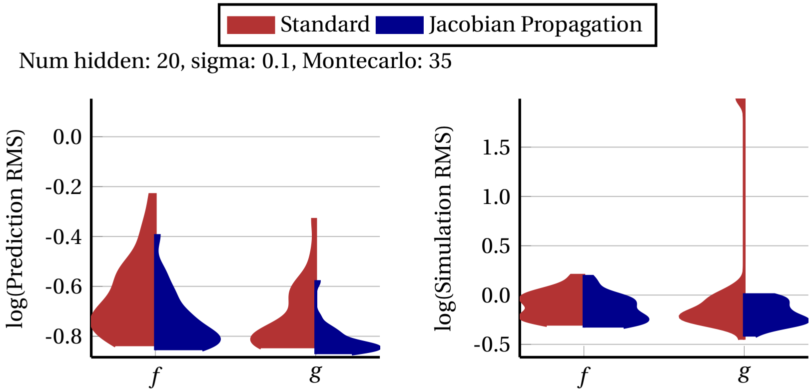


Figure: Distribution of prediction and simulation errors on the validation data. Each violin represents 35 Monte-Carlo runs.

Nonlinear system

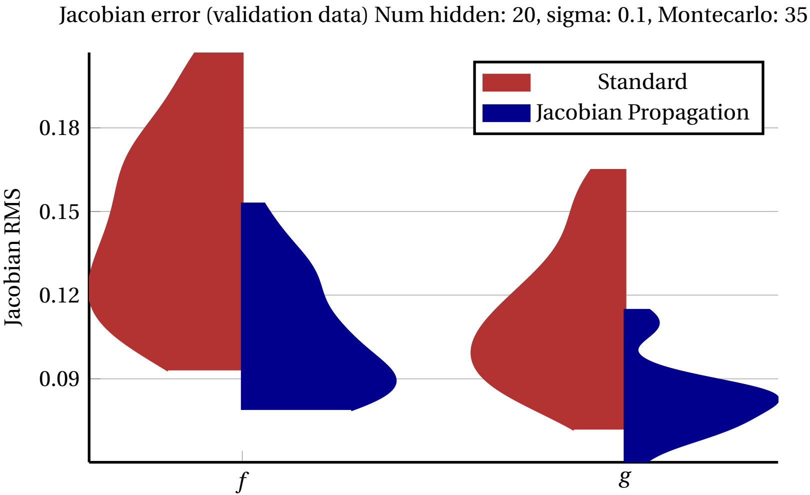
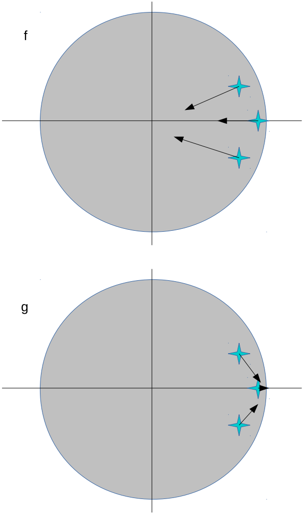


Figure: Distribution of errors in estimated Jacobians.

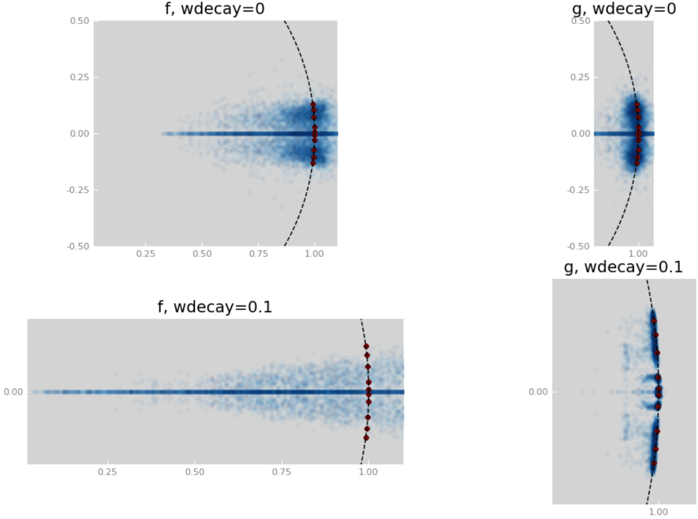
Weight decay



- L_2 Weight decay has different effect on f and g
- J vs $J - I$
- Weight decay shrinks eigenvalues of J
  to 0 for f
  to 1 for g
- With fast sampling, most eigenvalues are located around 1

Influence of weight decay

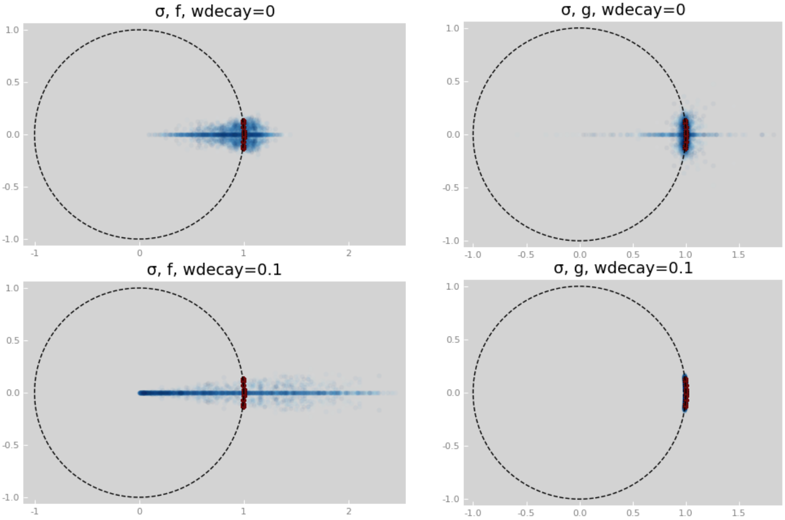
Weight decay has deteriorating effect on learning f
Weight decay is beneficial for learning g



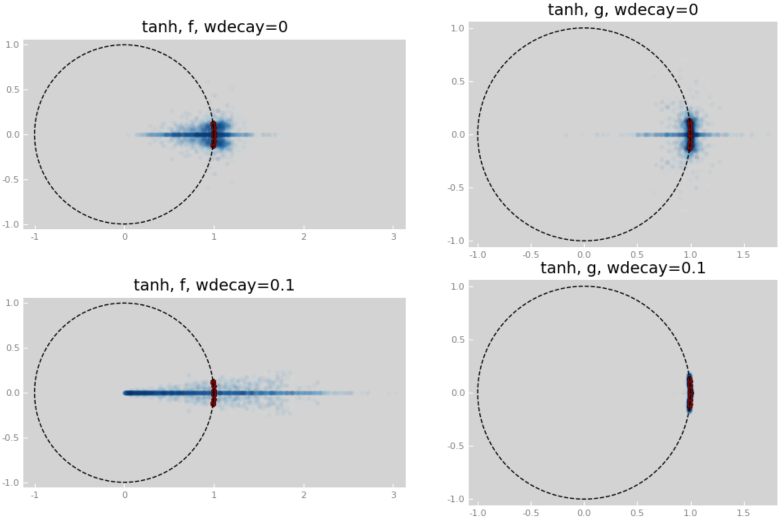
Different activation functions

The choice of activation function can have a large impact on the learned eigenvalue spectrum

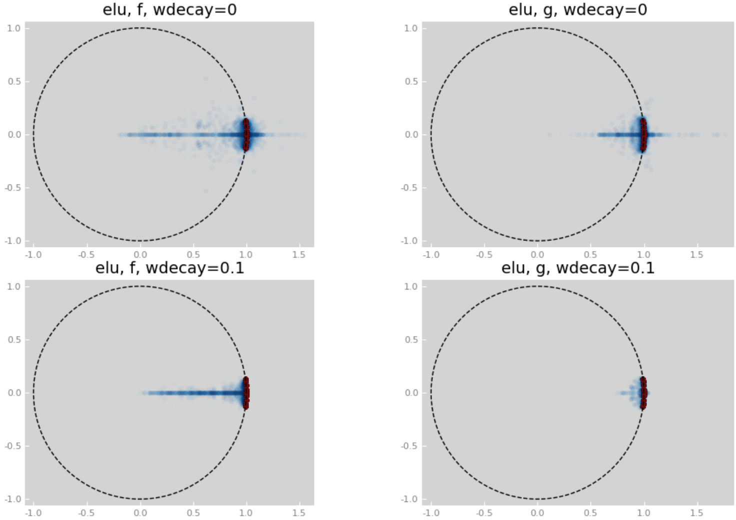
Different activation functions



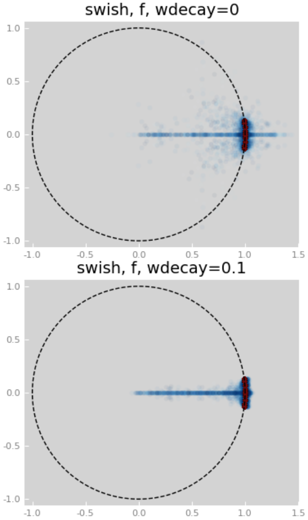
Different activation functions



Different activation functions

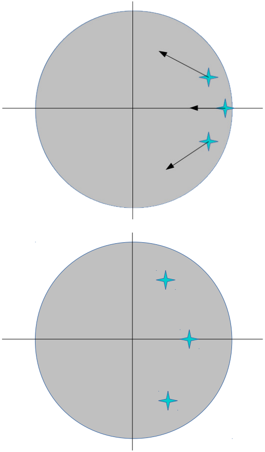


Different activation functions



Arbitrary eigenvalues

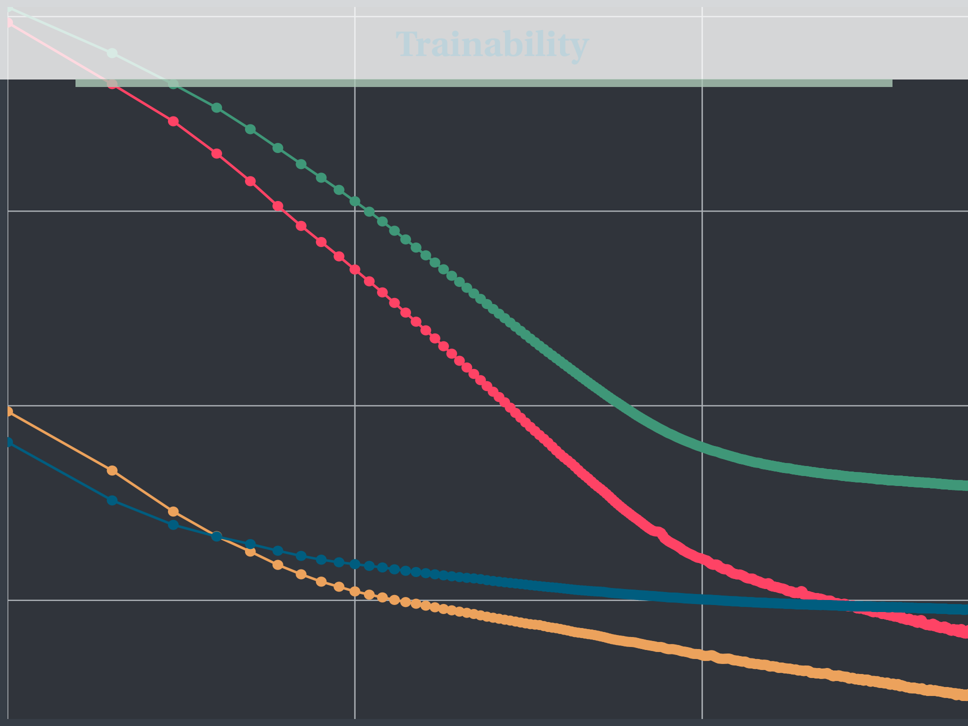
Can we have weight decay move the eigenvalues to arbitrary positions?



- We can use arbitrary linear model as baseline

$$x_{t+1} - Ax_t = h(x_t, u_t)$$

- Weight decay will shrink eigenvalues to A
- Requires a nominal linear model


Trainability

Trainability

For a linear model $y = \Phi\theta$ and a least-squares cost function

$$V(\theta) = \frac{1}{2}(y - \Phi\theta)^\top(y - \Phi\theta)$$

the gradient and the Hessian are given by

$$\nabla_\theta V = -\Phi^\top(y - \Phi\theta) \tag{1}$$

$$\nabla_\theta^2 V = \Phi^\top\Phi \tag{2}$$

The Hessian is clearly independent of both the output y and the parameters θ

Trainability

$$V_f(w) = \frac{1}{2}\sum_t \left(x^+ - f(x,u,w)\right)^\top \left(x^+ - f(x,u,w)\right)$$

f

$$\nabla_w^2 V_f = \sum_{t=1}^{T}\sum_{i=1}^{n} \nabla_w f_i \nabla_w f_i^\top - \left(x_i^+ - f_i(x,u,w)\right)\nabla_w^2 f_i$$

In this case, the Hessian depends on both the parameters and the target x^+ / Δx_i.

g

$$\nabla_w^2 V_g = \sum_{t=1}^{T}\sum_{i=1}^{n} \nabla_w g_i \nabla_w g_i^\top - \left(\Delta x_i - g_i(x,u,w)\right)\nabla_w^2 g_i$$

Trainability

g is more well behaved in the beginning of training and thus trains faster

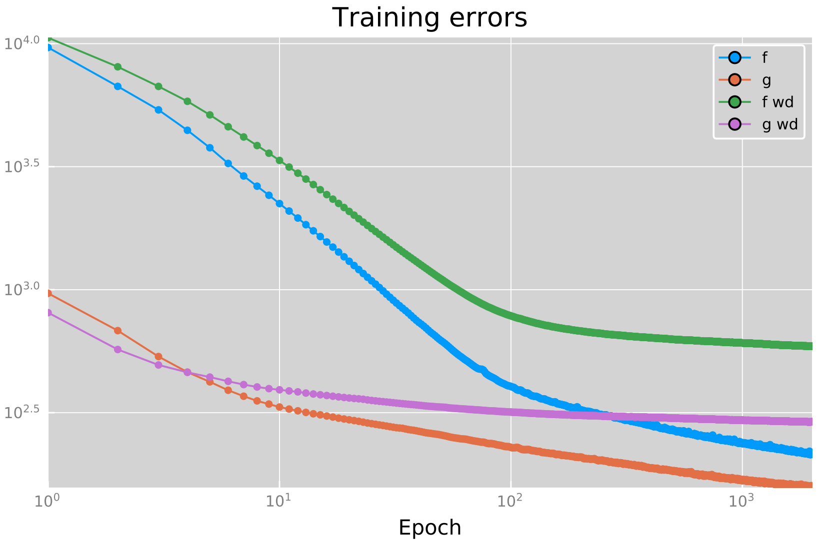
Trainability

For networks initialized with small random weights

- Eigenvalues of f start out in 0
- Eigenvalues of g start out in 1

g begins training closer to the goal and thus trains faster

Training errors



Conclusion

Two topics discussed

Tangent-space regularization

- Regularize training using knowledge of system smoothness etc.
- Can be implemented using auxiliary LTV model and sampling
- Learns better Jacobians and improves simulation/prediction performance

Learning of output time-differences

- Weight decay has beneficial influence
- Faster learning (more convex, better start)

Future work

Recurrent networks

- + Conclusions regarding weight decay remains for recurrent f/g
- ? Jacprop

Open source

Code to train the models presented in this talk available at
`https://github.com/baggepinnen/JacProp.jl`

Acknowledgment

Neural networks trained with Flux.jl in the Julia programming language