

Avoiding local minima in Deep Learning: a nonlinear optimal control approach

Master's Thesis defence

Jan Scheers

Supervisor: Prof. dr. ir. Panos Patrinos

Mentor: Ir. Brecht Evens

23 June 2021

0 Outline

- ① Introduction
- ② Multiple Shooting in MATLAB
- ③ Augmented Lagrangian Method
- ④ Regression Experiment
- ⑤ Timeseries Experiment
- ⑥ Conclusion

1 Outline

- ① Introduction
- ② Multiple Shooting in MATLAB
- ③ Augmented Lagrangian Method
- ④ Regression Experiment
- ⑤ Timeseries Experiment
- ⑥ Conclusion

1 Machine Learning and Neural Networks

▶ Machine Learning

- Subset of Artificial Intelligence
- Learn relationships in data sets
- Input features \rightarrow Target output
- Learn on Training Data, Apply to Test Data

▶ Artificial Neural Networks

- Popular machine learning model
- Very expressive
- Suited for large data sets
- Can model complex nonlinear relationships

▶ Focus on deep feedforward neural networks in this thesis

1 Neuron

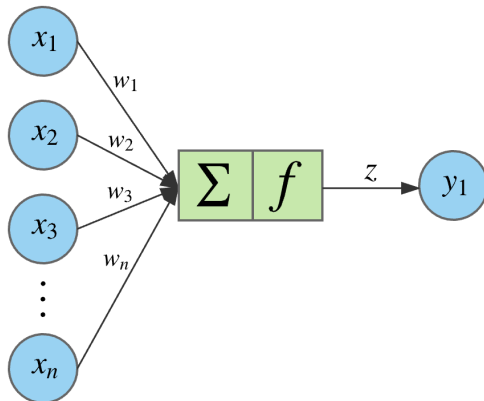


Figure: Single Neuron - McCulloch–Pitts model [A. Dertat 2017].

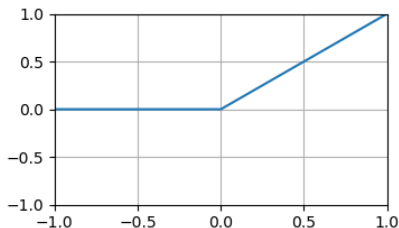
1 Activation Function

- McCulloch–Pitts neuron model

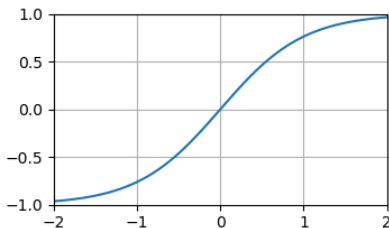
$$y = \sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n)$$

- Biases can be included by adding a constant input term $x_{n+1} = 1$
- Rectified Linear Unit (ReLU) Hyperbolic Tangent (tanh)

$$\sigma(x) = x^+ = \max(0, x)$$



$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x)$$



1 Neural Network

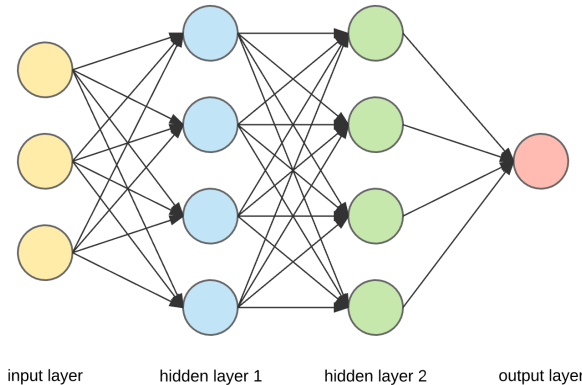


Figure: Feedforward Deep Neural Network [A. Dertat 2017].

1 Neural Network Training

- ▶ Function definition of network with L layers

$$f_W(x) = W_L \sigma(W_{L-1} \sigma(\dots W_1 \sigma(W_0 x) \dots))$$

- ▶ Function application & matrix multiplication
- ▶ Training = unconstrained optimization problem over training set:

$$\underset{W}{\text{minimize}} \quad C(W) = \frac{1}{N} \sum_{j=0}^N l(f_W(x_j) - y_j)$$

- ▶ $f_W(x_j)$: prediction of network for x_j
- ▶ compare network prediction to target output y_j
- ▶ Minimize average loss for loss function $l(\cdot, \cdot)$
- ▶ Proxy for performance metric P defined over the test set

1 Gradient Descent

- ▶ Training problem is solved using Gradient Descent:

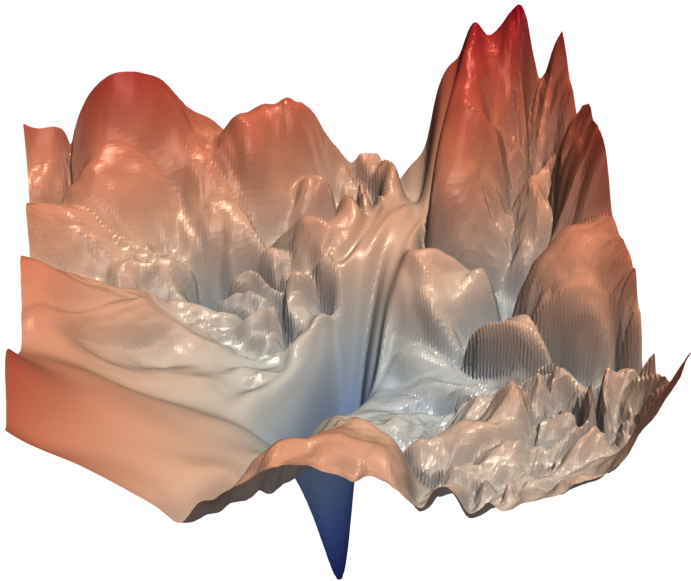
$$W_{k+1} = W_k - \eta_k \nabla C(W_k)$$



- η_k : step size a.k.a. "learning rate"
- Gradient is calculated using Backpropagation algorithm
- ▶ Cumbersome to calculate full gradient for large datasets
 - \Rightarrow Approximate gradient over "mini-batch" of datapoints
 - Stochastic Gradient Descent
- ▶ Current best algorithms use adaptive learning rates
 - AdaGrad, Rmsprop, ADAM, ...
 - ADAM algorithm is used in this thesis as comparison
- ▶ Halting based on "early stopping" criterion
 - \Leftrightarrow pure optimization halts on small gradient

1 Local Minima

- ▶ Gradient Descent is a locally converging algorithm
- ▶ Cost function will have highly non-convex surface
- ▶ Possibly uncountably many local minima
- ▶ Finding global minimum is intractible
 - "bad" local minima: have comparatively high cost value
 - Hard to define exactly
- ▶ " ... but experts now suspect that, for sufficiently large neural networks, most local minima have a low cost function value ... "
[Goodfellow et. al. 2016]
- ▶ Other optimization challenges
 - Ill-conditioned Hessian matrix
 - Exploding/Vanishing gradients
 - "Dying" ReLU



Visualisation of loss surface of ResNet-56 [Li et al. 2018].

1 Neural networks as Dynamical Systems

$$\begin{aligned}z_0 &= x \\z_{k+1} &= \sigma_k(W_k z_k), \quad k = 0, \dots, L \\f_W(x) &= z_{L+1}\end{aligned}$$

- ▶ Neural networks are dynamical systems
 - Each layer is a state
- ▶ Optimal Control objective cost function:

$$E(\theta) = \sum_{s=1}^{L-1} g^s(a^s, \theta^s) + h(a^L) \quad \text{💬}$$

- ▶ NN training only considers terminal cost

1 Training as Optimal Control Problem

$$\begin{aligned} & \underset{W}{\text{minimize}} && \sum_{j=0}^n \|\sigma_L(W_L z_L) - y_j\|_2^2 \\ & \text{subject to} && z_{1,j} = \sigma_0(W_0, x_j), && j = 1, \dots, n \\ & && z_{k+1,j} = \sigma_k(W_k z_{k,j}), && k = 1, \dots, L-1, j = 1, \dots, n \end{aligned}$$

- • MSE loss function
- n parallel systems, concatenated through the loss function
- constrained non-linear program

Notation	Optimal Control	Neural Network	Notation
θ	decision variables	weight parameters	W
a	state variables	(neuron) activation	z
	stage	layer	

1 Goal of the thesis: Solving the OCP

- ▶ Optimal control theory: two direct approaches
- ▶ Sequential Approach
 - Direct Single Shooting
 - Eliminate states using dynamics
 - Reduces to unconstrained problem
 - Small, highly nonlinear NLP
 - Equivalent to backpropagation for NN [Dreyfus et al. 1990]
- ▶ Simultaneous Approach
 - Direct Multiple Shooting
 - Keep states as variables
 - Large, structured problem
 - Novel approach for NN
 - Topic of this thesis

2 Outline

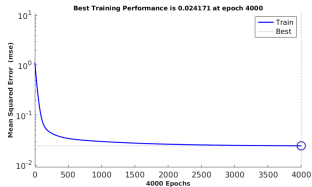
- 1 Introduction
- 2 Multiple Shooting in MATLAB**
- 3 Augmented Lagrangian Method
- 4 Regression Experiment
- 5 Timeseries Experiment
- 6 Conclusion

2 Multiple Shooting in MATLAB

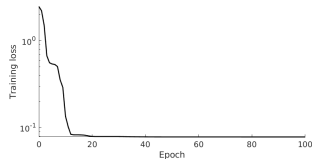
- ▶ Multiple shooting increases dimension of problem
 - For network of width W , depth L , N datapoints:
 - Single shooting: $\mathcal{O}(W^2L)$ variables
 - Multiple shooting: $\mathcal{O}(W^2L + WLN)$ variables
- ▶ Exploration of the multiple shooting approach
 - Implemented in MATLAB
 - Constrained NLP solved by `fmincon`
 - Comparison with `nntoolbox`
 - Training algorithm is `traingd`

Algorithm	tanh		ReLU	
	avg tr MSE	avg run time	avg tr MSE	avg run time
Gradient Descent	0.0299	1.577s	0.0274	1.480s
Multiple Shooting	0.2350	23.82s	0.2111	115.8s

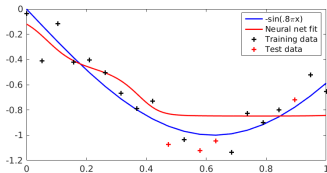
2 Gradient Descent vs Multiple Shooting



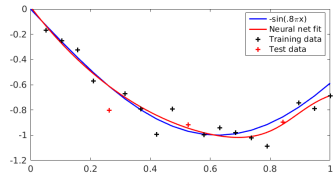
(a) Training performance of Gradient Descent



(b) Training performance of Multiple Shooting



(c) Neural network fit with Gradient Descent



(d) Neural network fit with Multiple Shooting

3 Outline

- 1 Introduction
- 2 Multiple Shooting in MATLAB
- 3 Augmented Lagrangian Method**
- 4 Regression Experiment
- 5 Timeseries Experiment
- 6 Conclusion

3 Augmented Lagrangian Method

- ▶ `fmincon` not well suited for this problem
 - \Rightarrow implement specialized algorithm
 - exploit structure in problem
- ▶ Augmented Lagrangian Method
 - Algorithmic framework for constrained NLP
 - Similar to penalty method
- ▶ Constrained NLP:

$$\begin{array}{ll}\min_u & f(u) \\ \text{s.t.} & h(u) = 0\end{array}$$

3 Classical Augmented Lagrangian Method

- ▶ β -augmented Lagrangian:

$$\min_u \max_{\lambda} \mathcal{L}_{\beta}(u, \lambda) = f(u) + \langle \lambda, h(u) \rangle + \frac{\beta}{2} \|h(u)\|_2^2$$

- ▶ Converges to local minimizer of constrained problem if
 - β is large
 - λ is close to λ^*
- ▶ Algorithm solves series of unconstrained problems

$$u_{k+1} = \operatorname{argmin}_u \mathcal{L}_{\beta_k}(u, \lambda_k)$$
$$\lambda_{k+1} = \lambda_k + \sigma_k h(u_{k+1})$$

3 Classical Augmented Lagrangian Method

- ▶ Penalty parameter increases or is kept the same in each iteration
 - depends on constraint violation
- ▶ Stopping criterion:

$$\|h(u_k)\| \leq \tau_1 \quad \text{and} \quad \|\nabla_u \mathcal{L}_{\beta_k}(u_k, \lambda_k)\| \leq \tau_2$$

- ▶ NN training has nonconvex f with nonlinear constraints, inexact solution to inner problem
- ▶ Textbook theory requires strong assumptions, exact solution
- ▶ Sahin et al. 2019 proposes alternative framework:
 - Promises $\tilde{O}(1/\epsilon^3)$ calls to first-order solver
 - Theoretical estimates on convergence

3 β -Augmented Lagrangian

$$\begin{aligned} \underset{W, z}{\text{minimize}} \quad & \sum_{j=0}^n \|\sigma_L(W_L z_L) - y_j\|_2^2 & \min_u \quad & \frac{1}{2} \|F(u)\|_2^2 \\ \text{subject to} \quad & z_{1,j} = \sigma_0(W_0, x_j) & \Leftrightarrow \text{s. t.} \quad & h(u) = 0 \\ & z_{k+1,j} = \sigma_k(W_k z_{k,j}) \end{aligned}$$

- ▶ $u = \{W, z\}$ collects weight and state variables in single vector
- ▶ minimizing β -Augmented Lagrangian is LS problem

$$\begin{aligned} \underset{u}{\operatorname{argmin}} \quad \mathcal{L}_\beta(u, \lambda) &= \frac{1}{2} \|F(u)\|_2^2 + \langle \lambda, h(u) \rangle + \frac{\beta}{2} \|h(u)\|_2^2 \\ &= \frac{\beta}{2} \left\| \begin{bmatrix} F(u)/\sqrt{\beta} \\ h(u) + \lambda/\beta \end{bmatrix} \right\|_2^2 \end{aligned} \tag{1}$$

3 Applied Augmented Lagrangian Method

Input: Initial weights vector W , penalty parameter β , stopping tolerance τ , input-target pairs $(x_i, y_i), i = 1, \dots, n$

Initialization $u_0 = \{W, f_W(x)\}, \lambda_0 \in \mathcal{N}(0, 1)$; *Initialize state variables by simulating network, initialize dual variables randomly*

for $k = 0, 1, \dots$ **do**

$\eta_k = 1/\beta^k$; *Update tolerance*

find u_{k+1} such that

$\|\nabla_{u_k} \mathcal{L}_{\beta^k}(u_k, \lambda_k)\| \leq \eta_k$; *Approx. primal solution*

$\sigma_{k+1} = \min\left(\frac{\|h(u_0)\| \log^2 2}{\|h(u_{k+1})\| k \log^2(k+1)}, 1\right)$; *Update dual step size*

$\lambda_{k+1} = \lambda_k + \sigma_{k+1} h(u_{k+1})$; *Update dual variables*

$\|\nabla_{u_{k+1}} \mathcal{L}_{\beta^k}(u_{k+1}, \lambda_k)\| + \|h(u_{k+1})\| < \tau$; *Stopping Criterion*

end



3 Convergence

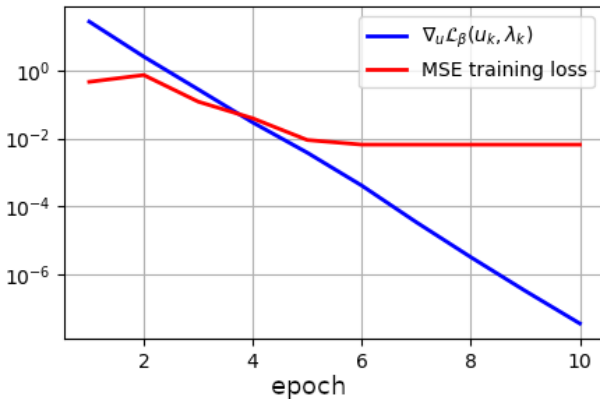


Figure: Typical convergence behaviour of Algorithm 1

3 Jacobian

- ▶ Inner problem is nonlinear least squares
 - Solved using Trust Region Reflective method
 - `scipy.optimize.least_squares`
 - Works best with analytical solution for Jacobian matrix
- ▶ Jacobian matrix
 - Dimension $\tilde{Q}(DWN \times DW(N + W))$
 - Sparse, banded structure
 - every layer only connected to prev and next layer
- ▶ Numerical Verification
 - Automatic differentiation
 - AlgoPy library
 - Confirms correctness of analytical solution

3 Jacobian Derivation

$$M_\beta(u, \lambda) := \begin{bmatrix} F(u)/\sqrt{\beta} \\ h(u) + \lambda/\beta \end{bmatrix}, J_{M_\beta} := \begin{bmatrix} \frac{\partial M_\beta}{\partial u_1} & \frac{\partial M_\beta}{\partial u_2} & \cdots & \frac{\partial M_\beta}{\partial u_n} \end{bmatrix}$$

$$\frac{\partial M_\beta}{\partial W_k} = -z_{k,j} \sigma'_k(W_k z_{k,j}) \quad , j = 1, \dots, n, k = 0, \dots, L$$

$$\frac{\partial M_\beta}{\partial z_k} = \begin{bmatrix} 1 \\ -W_k \sigma'_k(W_k z_k) \end{bmatrix} \quad , k = 1, \dots, L$$

- ▶ W_k, z_k are matrices \rightarrow derivatives are tensors
- ▶ W_k, z_k vectorized \rightarrow derivatives are block matrices

3 Visualisation of Jacobian

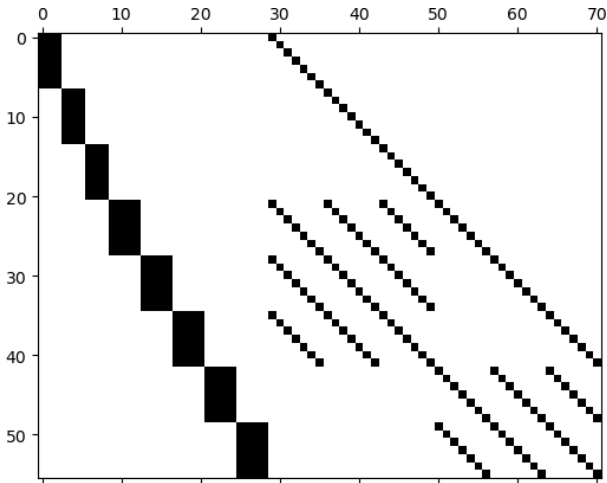


Figure: Network has 2 layers, 3 width, 2 inputs, 2 outputs and 7 datapoints.

4 Outline

- 1 Introduction
- 2 Multiple Shooting in MATLAB
- 3 Augmented Lagrangian Method
- 4 Regression Experiment**
- 5 Timeseries Experiment
- 6 Conclusion

4 Test setup

▶ Regression problem

- Approximate:

$$y = \sin(x^2), x \in [0, \pi]$$

- Using fully connected feedforward network

▶ ADAM optimizer for comparison

- ADaptive Moment Estimation
- Robust, industry standard optimizer
- Stochastic gradient descent with adaptive learning rate

▶ 20 training runs per test configuration

- Same initial conditions each run
- Same data, noise, weight initialization
- Gradient estimation over full dataset in each epoch

▶ Intel® Core™ i5-7300HQ CPU 2.50GHz with 8Gi RAM

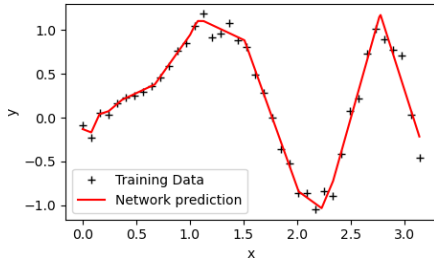
4 Halting criterion

- ▶ Nonconvex Optimization
 - Local minimizer of cost function: small gradient
- ▶ Machine Learning
 - "Early Stopping"
 - Based on performance metric on validation set
- ▶ Time based, epoch based halting
- ▶ Goal of thesis: local minima \Rightarrow Halt at local minimum
 - ADAM: 10 epochs without any improvement to cost function
 - ALM: 1 epoch without significant improvement:

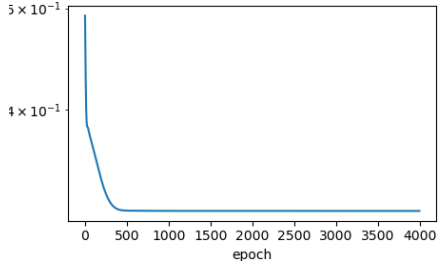
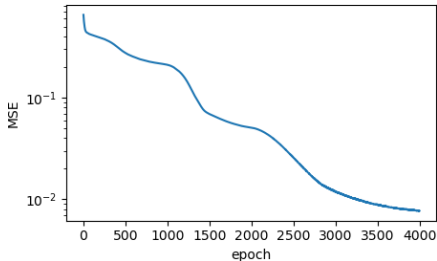
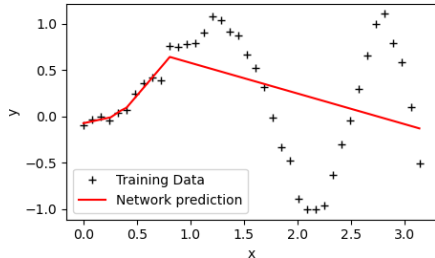
$$(1 + \epsilon)C(W_{k+1}) > C(W_k)$$

- Halting based on gradient
- Discards influence of states, constraint violation

Good convergence

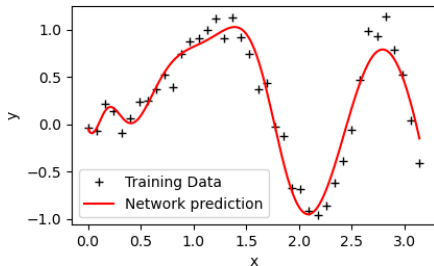


Bad convergence

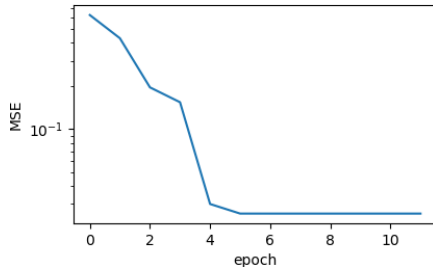


Examples of convergence behaviour of ADAM optimizer for a network using ReLU activation and 40 data points. The network has 2 hidden layers of 16 nodes.

Good convergence

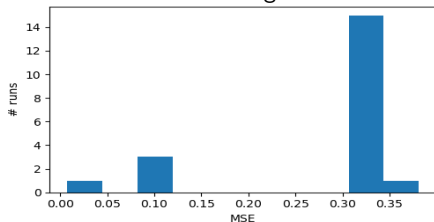


Convergence History

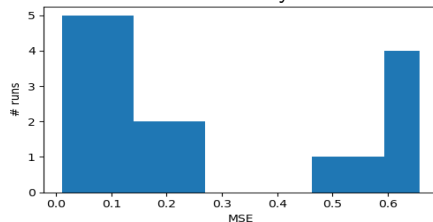


Example of convergence behaviour of ALM optimizer for network using *tanh* activation and 40 data points. The network has 2 hidden layers of 8 nodes.

ADAM Histogram



ALM History



MSE of batch using 40 data points and ReLU activation function

4 Results for network with \tanh activation

$$\sigma(\cdot) = \tanh(\cdot)$$

	N	avg MSE	best MSE	time(s)	epochs	conv
ADAM	10	$2.09e^{-2}$	$1.20e^{-11}$	18.7	3790	17
ALM		$2.10e^{-1}$	$1.59e^{-1}$	1.85	6.38	16
ADAM	20	$1.25e^{-2}$	$9.84e^{-4}$	19.8	3960	17
ALM		$5.87e^{-2}$	$7.69e^{-3}$	8.06	7.50	20
ADAM	40	$2.56e^{-2}$	$7.58e^{-3}$	19.2	3920	15
ALM		$1.61e^{-2}$	$7.90e^{-3}$	11.6	6.90	20

Fully connected feedforward network, 2 hidden layers, 8 nodes / layer

4 Results for network with ReLU activation

$$\sigma(\cdot) = \max(\cdot, 0)$$

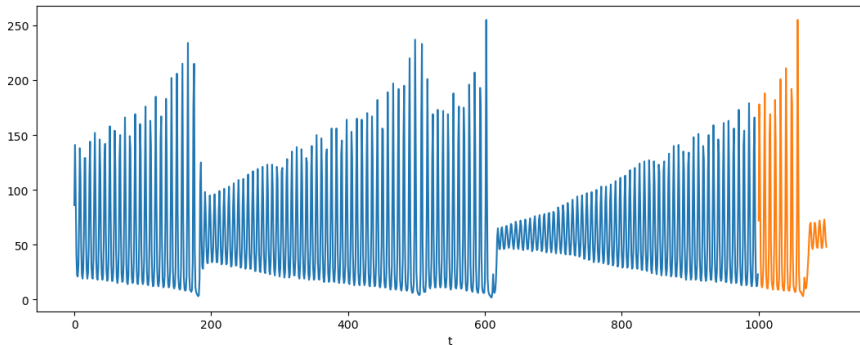
	N	avg MSE	best MSE	time(s)	epochs	conv
ADAM	20	$1.22e^{-1}$	$3.15e^{-3}$	30.3	3500	5
ALM		$1.63e^{-1}$	$1.08e^{-2}$	4.41	5.47	15
ADAM	40	$6.28e^{-2}$	$8.85e^{-3}$	22.4	2690	5
ALM		$9.89e^{-2}$	$1.37e^{-2}$	8.09	5.11	18
ADAM	80	$7.61e^{-2}$	$7.64e^{-3}$	29.9	3150	4
ALM		$1.02e^{-1}$	$1.03e^{-2}$	25.3	4.21	14

Fully connected feedforward network, 2 hidden layers, 16 nodes / layer

5 Outline

- 1 Introduction
- 2 Multiple Shooting in MATLAB
- 3 Augmented Lagrangian Method
- 4 Regression Experiment
- 5 Timeseries Experiment**
- 6 Conclusion

5 Santa Fe timeseries data



- ▶ Far-infrared laser fluctuations, coupled nonlinear ODEs
- ▶ 1000 data points training, 100 points test data

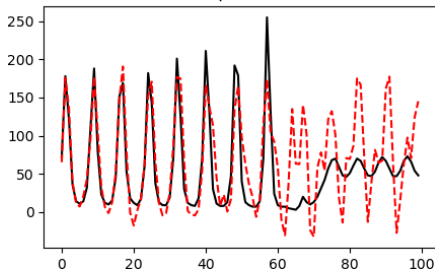
5 Test Setup

- ▶ Approximate using recurrent neural network:

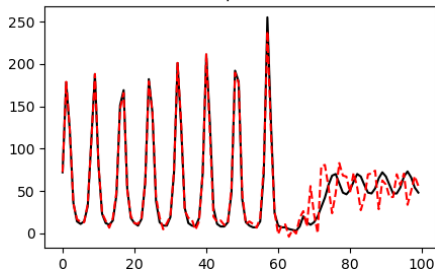
$$\hat{y}_{k+1} = f_W(\left[\hat{y}_k \quad \hat{y}_{k-1} \quad \dots \quad \hat{y}_{k-p}\right]^T)$$

- ▶ Previous predictions used to make next prediction
- ▶ tanh activation, 80 inputs, 1 layer, 48 nodes
- ▶ Data normalized: mean 0, std dev 1
- ▶ 920 training pairs, 600 in training set, 320 in validation set
- ▶ 4 training runs per algorithm, same initial conditions
- ▶ Intel® Core™ i5-7300HQ CPU 2.50GHz with 8Gi RAM

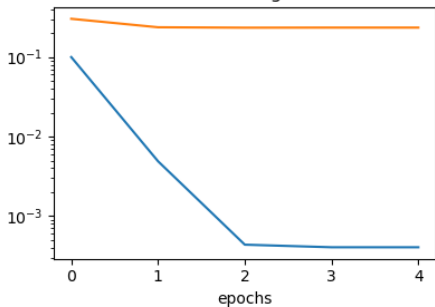
ALM prediction



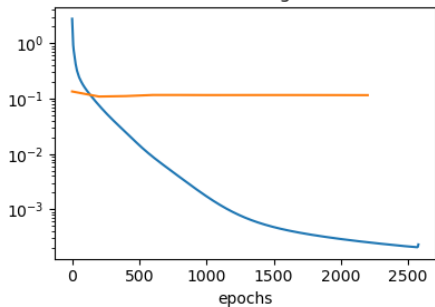
ADAM prediction



ALM convergence



ADAM convergence



Test MSE in blue, Validation MSE in orange

5 Timeseries data test results

Algo	avg MSE	best MSE	avg pred MSE	best pred MSE	epochs	time
ALM	$3.712e^{-4}$	$2.779e^{-4}$	$1.070e^0$	$8.288e^{-1}$	5	1201s
ADAM	$2.517e^{-4}$	$2.329e^{-4}$	$2.016e^{-1}$	$9.100e^{-2}$	2353	20.9s

- ▶ ALM is very slow: 20min on avg, compared to 20s for ADAM
- ▶ Similar performance for training data, ADAM much better on test data
- ▶ Most calls to jacobian are in first epochs:


Epoch	1	2	3	4	5
#evals	42.25	12.75	17.0	3.0	2.0

Table: Average number of Jacobian evaluations per epoch

6 Outline

- 1 Introduction
- 2 Multiple Shooting in MATLAB
- 3 Augmented Lagrangian Method
- 4 Regression Experiment
- 5 Timeseries Experiment
- 6 Conclusion**

6 Conclusion

- ▶ Novel perspective on neural network training using control theory
 - Single shooting: backpropagation
 - Multiple shooting: new algorithm
 - MATLAB provides PoC
- ▶ ALM framework implemented
 - Framework adapted from [Sahin et al. 2019]
 - Analytical solution for Jacobian
- ▶ Numerical Tests
 - ALM outperforms ADAM for network with many bad local minima
 - ALM becomes impractical for larger datasets
- ▶ Mini-batch approach for ALM can mitigate scaling problem
- ▶ Larger networks usually don't have bad local minima

7 References

- [1] A. Dertat. Applied deep learning - part 1: Artificial neural networks. URL: <https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6>, retrieved 2021-05-31
- [2] S. E. Dreyfus. Artificial neural networks, back propagation, and the kelley-bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*, 13(5):926–928, 1990.
- [3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [4] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets, 2018.
- [5] M. F. Sahin, A. eftekhari, A. Alacaoglu, F. Latorre, and V. Cevher. An inexact augmented lagrangian framework for nonconvex optimization with nonlinear constraints. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Questions

7 Loss Functions

quadratic loss function

$$||y^2 - \hat{y}^2||$$

Mean Squared Error(MSE)

Regression

cross-entropy loss

$$y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

negative log-likelihood

Classification

