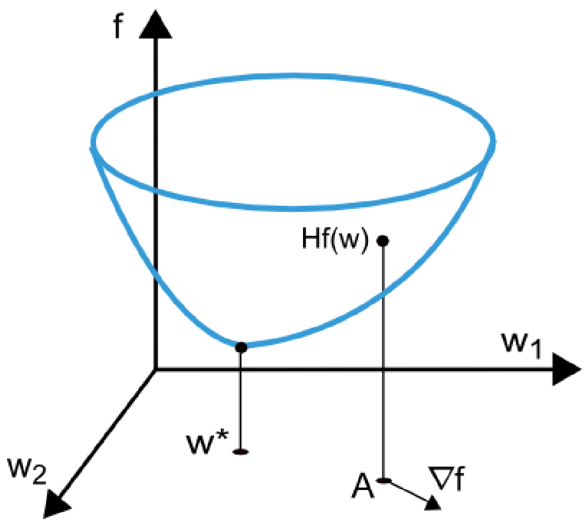
Mathematical Modelling and Simulation

Artificial Neural Networks

Revision – Training of Networks

- Cost function f defined
- Searching for the global minimum



First derivative

$$\nabla_i f(w) = \frac{\partial f}{\partial w_i}$$

Second derivative

$$H_{i,j} f(w) = \frac{\partial^2 f}{\partial w_i \partial w_j}$$

https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

Revision – Training of Networks

Mathematical Modelling and Simulation

Gradient descent:
Going towards steepest descent

$$\boxed{w \rightarrow w \; - \eta \cdot \nabla f(w)}$$

Trainings direction:
$d = -\nabla f$

- Needs many iterations if not very steep
- convergence is not the fastest
- \+ no Hassian matrix necessary => faster for many parametes

Training Algorithms

Mathematical Modelling and Simulation

- Gradient Descent
- Newton's Method
- Conjugate Gradient
- Quasi Newton
- Levenberg Marquardt

Training – Newton's Method

Mathematical Modelling and Simulation

Second order algorithm

Taylor Approximation 2.Order

$$f = f(w_0) + \nabla f(w_0) \cdot (w - w_0) + \frac{1}{2}(w - w_0)H_f(w_0)$$

Assume $\nabla f = 0$ at minimum of $f(w)$

$$\nabla f = \nabla f(w_0) + H_f(w_0)(w - w_0) = 0$$
$$\Rightarrow \Delta w = H_f^{-1}(w_0)\nabla f(w_0)$$

Training – Newton's Method

Mathematical Modelling and Simulation

Newton trainings step

$$\Delta w = H_f^{-1}(w_0)\nabla f(w_0)$$

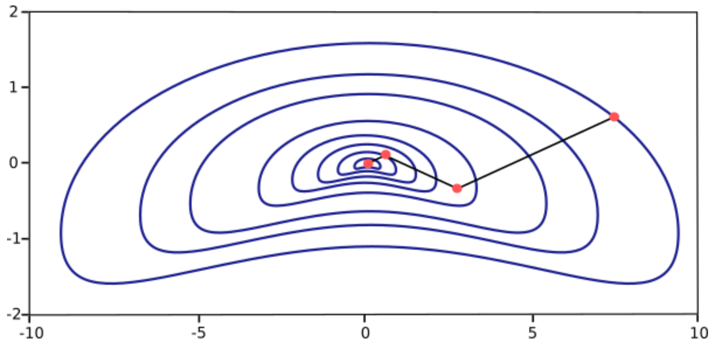
- might lead to maximum if H is not pos. def.
- introduce the training rate

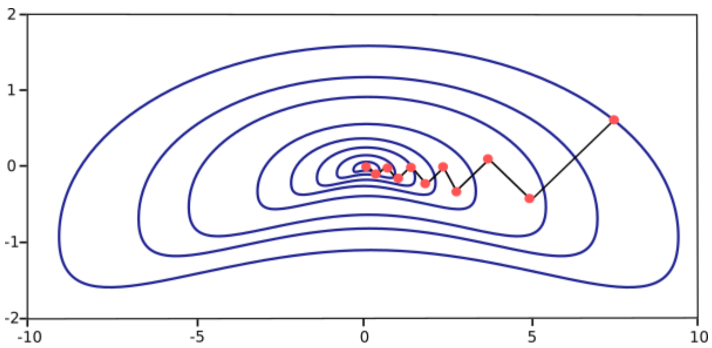
$$\boxed{w \rightarrow w - \eta \cdot H_f^{-1}(w)\,\nabla f(w)}$$

Trainings direction:
$d = H_f^{-1}\,\nabla f$

Revision – Newton's Method

- calculation of Hassian Matrix
- evaluating the inverse of H
- much faster than Gradient Descent





https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

Training Algorithms

Mathematical Modelling and Simulation

- Gradient Descent
- Newton‘s Method
- Conjugate Gradient
- Quasi Newton
- Levenberg Marquardt

Training – Conjugate Gradient

Mathematical Modelling and Simulation

Accelerate slow convergence of gradient descent

Search along conjugate directions w.r.t. H

$$d^0 = -\nabla f(w_0) \coloneqq g^0 \text{ and } d^{i+1} = g^{i+1} + d^i \cdot \gamma^i$$

γ conjugate parameter

Training – Conjugate Gradient

Mathematical Modelling and Simulation

γ^i conjugate parameter:

- Fletcher – Reeves $$\gamma^i = \frac{{g^i}^T g^i}{{g^{i-1}}^T g^{i-1}}$$
- Polak-Ribière $$\gamma^i = \frac{{g^i}^T (g^i - g^{i-1})}{{g^{i-1}}^T g^{i-1}}$$
- Hestenes-Stiefel $$\gamma^i = \frac{{g^i}^T (g^i - g^{i-1})}{{d^{i-1}}^T (g^i - g^{i-1})}$$

Training – Newton's Method

Mathematical Modelling and Simulation

Training direction is periodically reset to the negative gradient

$$\boxed{w \rightarrow w + \eta \cdot d}$$

+ no calculation of H and inverse
+ also possible for big neutal networks
+ more effecitive than Gradient Descent

Training Algorithms

Mathematical Modelling and Simulation

- Gradient Descent
- Newton's Method
- Conjugate Gradient
- Quasi Newton
- Levenberg Marquardt

Training – Quasi Newton's Method

Computationallz expensive to compute Hessian Matrix and it's inverse

(as in Newton Method)

➡ Approximate this matrix with Matrix G of first partial derivatives

$$\boxed{w \to w - \eta \cdot (G\, \nabla f(w))}$$

Trainings direction:
$d = G\, \nabla f$

Training – Quasi Newton's Method

Mathematical Modelling and Simulation

DFP (David-Fletcher-Powell)

$$G_{k+1} = G_k + \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T \Delta g_k} - \frac{G_k \Delta g_k \Delta g_k^T G_k}{\Delta g_k^T G_k \Delta g_k}$$

BFGS (Broyden-Fletcher-Goldfarb-Shanno)

$$G_{k+1} = G_k + \frac{\Delta g_k \Delta g_k^T}{\Delta g_k^T \Delta x_k} - \frac{G_k \Delta x_k \Delta x_k^T G_k^T}{\Delta x_k^T G_k \Delta x_k}$$

$$\Delta g_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$
$$\Delta x_k = x_{k+1} - x_k$$

whereas $\alpha_k = argmin\, f(x_k + \alpha p_k) \Rightarrow s_k = \alpha_k p_k \Rightarrow x_{k+1} = x_k + s_k$

Training Algorithms

- Gradient Descent
- Newton's Method
- Conjugate Gradient
- Quasi Newton
- Levenberg Marquardt

Training – Levenberg Marquardt

Mathematical Modelling and Simulation

Approximate H with Gradient and Jacobi

Assume: loss function $\quad f = \sum_i e_i^2$

Jacobi matrix $\quad J_{i,j} = \frac{\partial e_i}{\partial w_j}$

Gradient of loss function $\nabla f = 2J^T e$

$\Rightarrow$ $H_f = 2J^T J + \lambda I \qquad \lambda$ ... damping factor

Training – Levenberg Marquardt

Update of weights

$$\boxed{w \to w - (J^T J + \lambda I)^{-1} \cdot (2J^T e)}$$

$\lambda = 0 \Rightarrow$ Newton Method

$\lambda \gg 0 \Rightarrow$ Gradient Descent

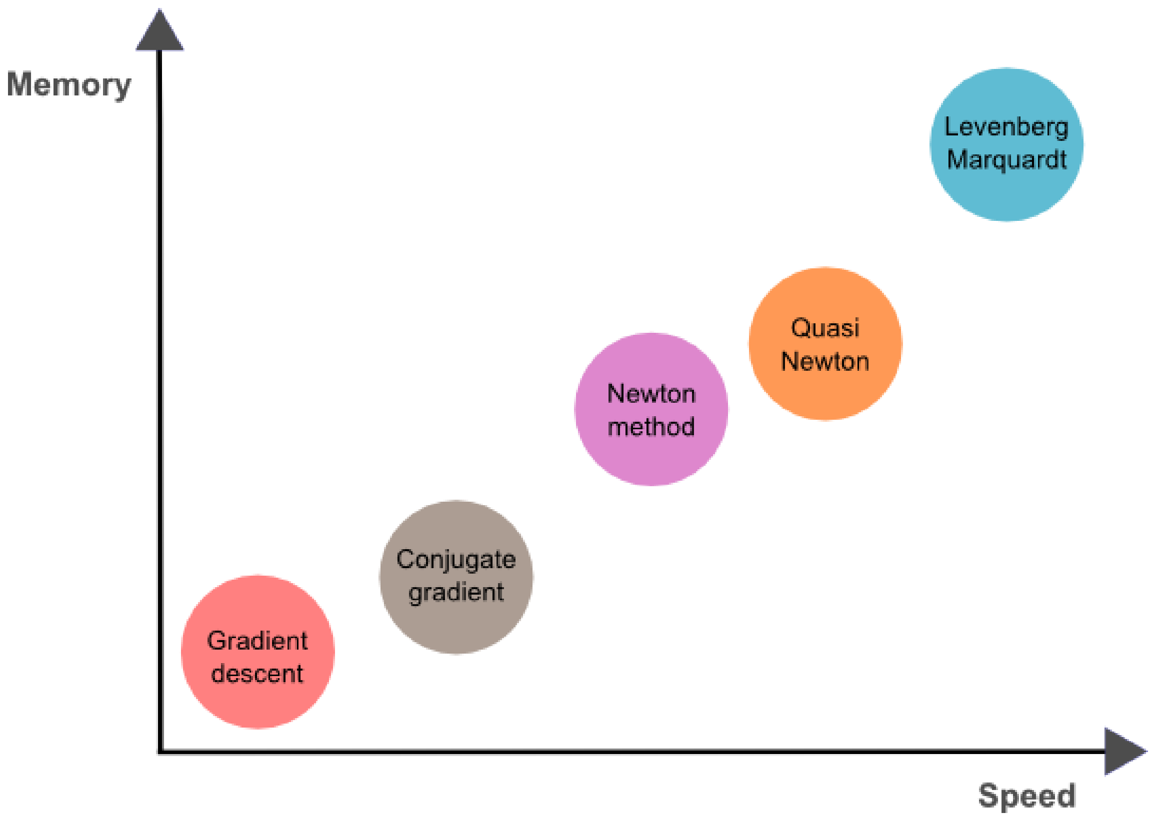
+ very fast for neural networks
− only for mean squared error
− big networks require big memory for J

Training Algorithms

For big neural network:
Gradient descend and conjugate gradient

For only few hundred parameters:
Levenberg-Marquardt

Every thing else:
Quasi - Newton



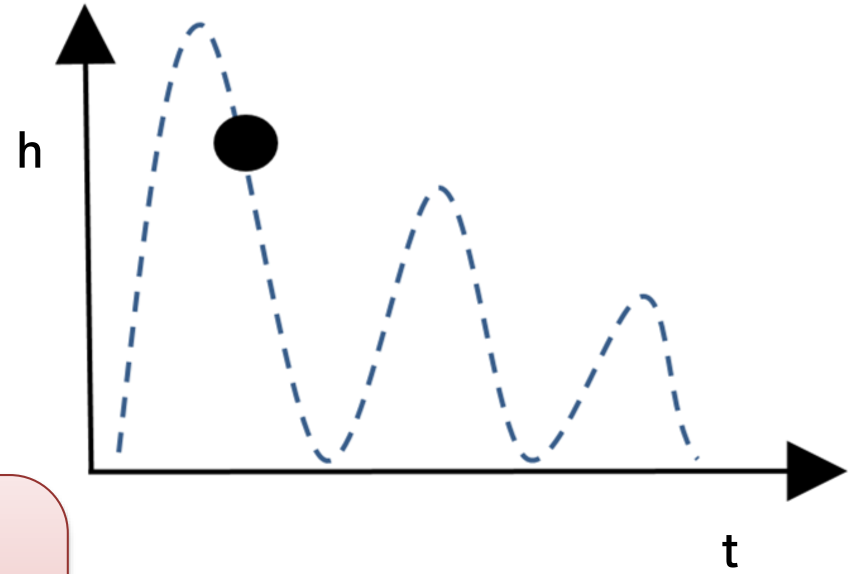
Case Study – Bouncing Ball

Mathematical Modelling and Simulation

System description:

$$\ddot{h}(t) = -g$$

$$h(0) = h_0, \dot{h}(0) = v_0$$



analytical solution:

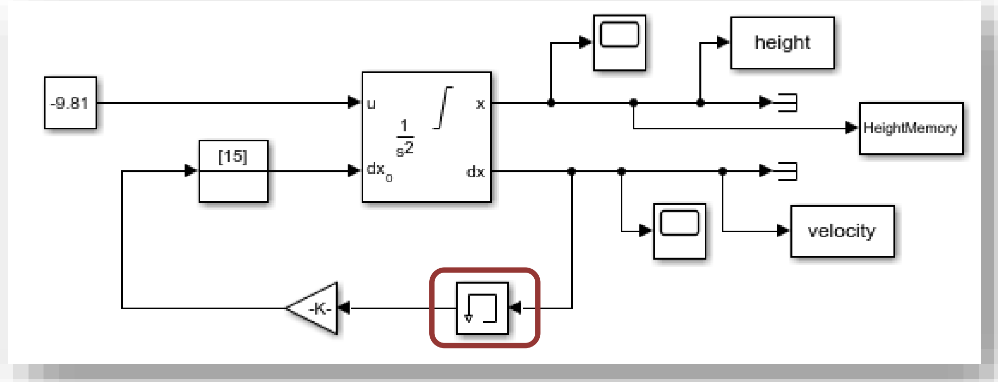
$$\begin{pmatrix} h(t) \\ \dot{h}(t) \end{pmatrix} = \begin{pmatrix} -\frac{g}{2}t^2 + v_0 t + h_0 \\ -gt + v_0 \end{pmatrix}$$

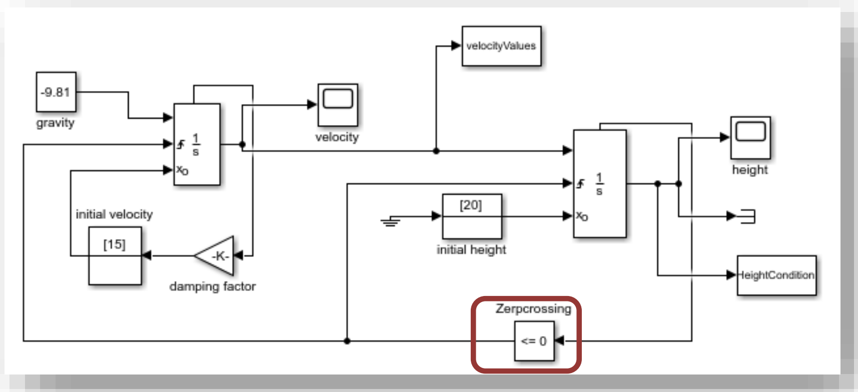
Case Study – Implementations HS

Mathematical Modelling and Simulation

MATLAB Implementations:

- Event function for ODE
- Analytical solution



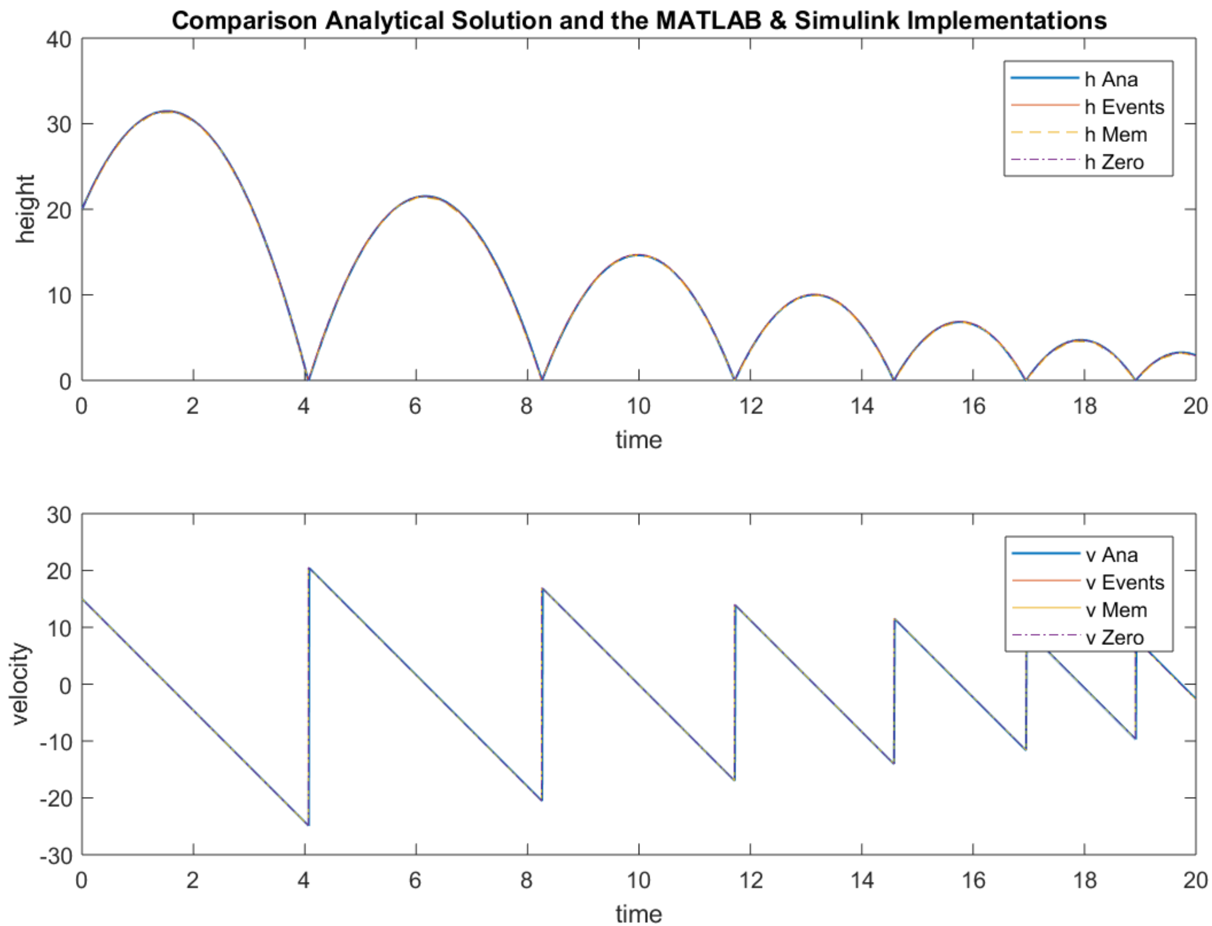


Simulink Implementation:

- Memory Block
- Zero crossing detection

Results – Hybrid System

Mathematical Modelling and Simulation

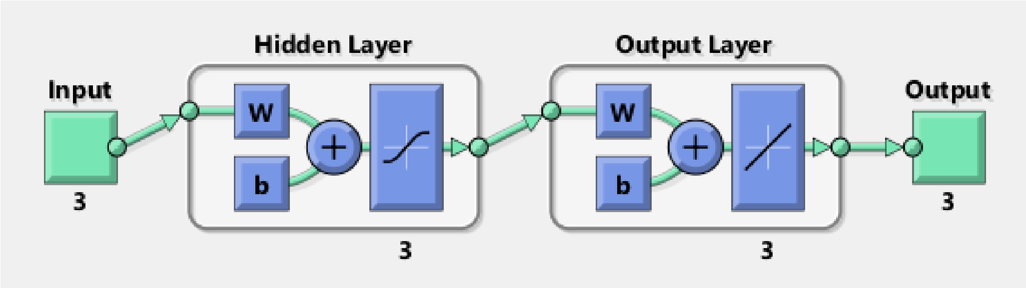


Case Study – Implementation NN

MATLAB Neural Network Tool Box

Structure:

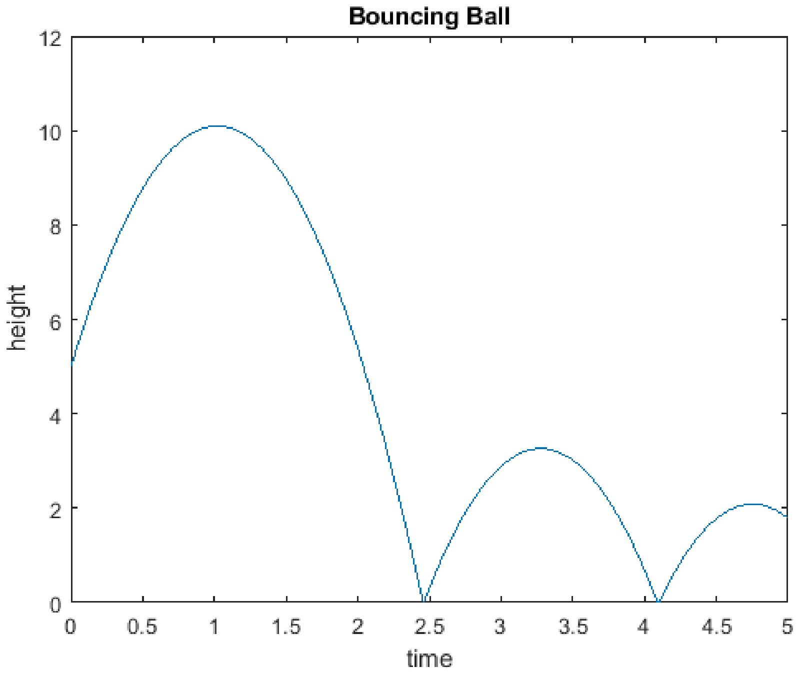
- Input layer with 3 nodes
- Various number of hidden neurons
- Output layer with 3 to 5 nodes



Activation function: sigmoid function

Case Study – Input & Output Values

Scenario 1:
- 3 inputs: intial values and energy loss
- 3 outputs: first event time, time and height of first bounce
- Varying number of hidden neurons: 3, 6 or 9 hidden neurons
- Size of data sets: 61, 122 or 7676 data points



Scenario 2:
- 3 inputs: intial values and energy loss
- 5 outputs: first 5 events
- Varying number of hidden neurons: 3, 5 or 10 hidden neurons
- Size of data sets: 336, 1271 or 7676 data points

Results – NN 1: Bounce & Event

TU WIEN Mathematical Modelling and Simulation

1.Dataset: #61

- Input: $h_0\ 0\ to\ 15;\ v_0 15\ to\ 0$
- $\Delta = 0.5$
- Output: first event, height and timing of first bounce

2.Dataset: #122

- Input:
  $h_0\ 0\ to\ 15\ to\ 0\ ;\ v_0\ 15\ to\ 0\ to\ 15$
- $\Delta = 0.5$
- Output: first event, height and timing of first bounce

3.Dataset: #7676

- Input:
  $h_0 \in [0{,}15];\ v_0\ \in [0{,}15]$
- $\Delta = 0.5$
- Output: first event, height and timing of first bounce

Error calculation: $\left\|\sum_{i=1}^{5} e_i - y_i\right\|$

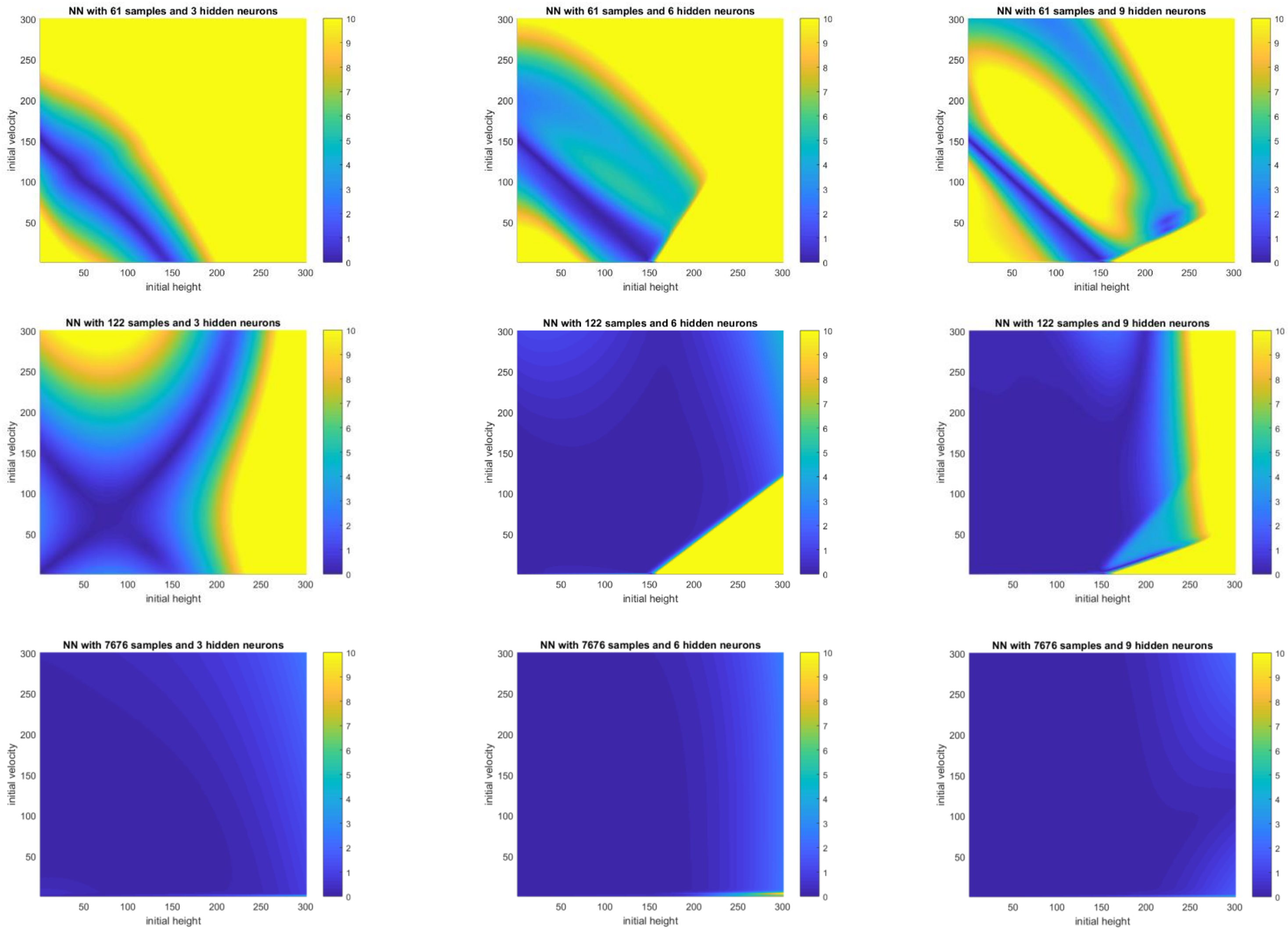
Plotting area:

- $h_0 \in [0{,}30]$
- $v_0 \in [0{,}35]$
- $\Delta = 0.1$

Results – NN 1: Bounce & Event

Mathematical Modelling and Simulation

Increasing sample size



Increasing number of hidden neurons

Results – NN 2: Event Times

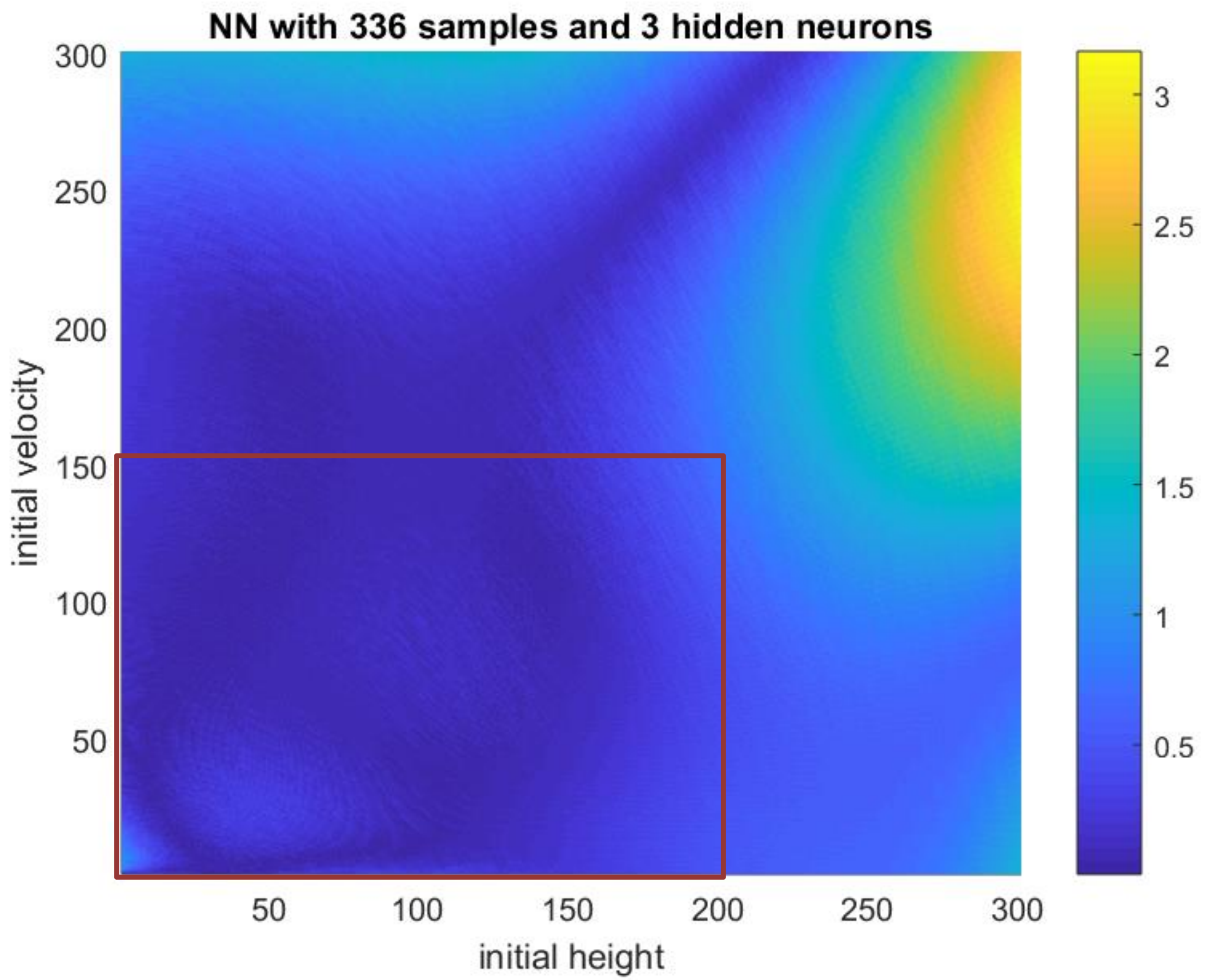
Mathematical Modelling and Simulation

Input:
- $h_0 \in [0,20]$
- $v_0 \in [0,15]$
- $\Delta \in \{0.2, 0.5, 1\}$

Output:
First 5 events

Error calculation
$\left\|\sum_{i=1}^{5} e_i - y_i\right\|$

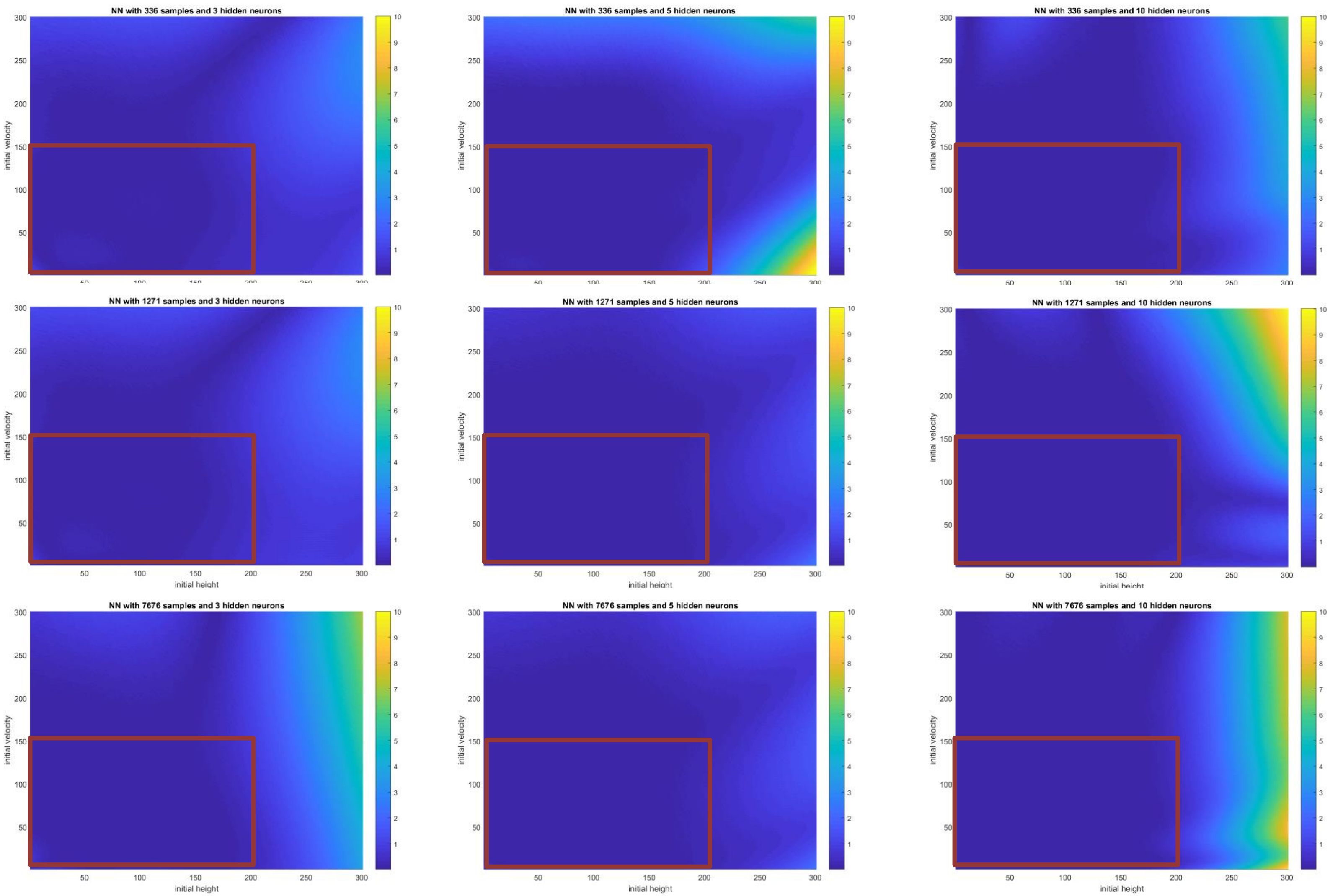
Plotting area:
- $h_0 \in [0,30]$
- $v_0 \in [0,35]$
- $\Delta = 0.1$



Results – NN 2: Event Times

Mathematical Modelling and Simulation

Increasing sample size



Increasing number of hidden neurons

Conclusion

Mathematical Modelling and Simulation

- Simulating of hybrid systems with NN possible
- Analytical solution for data set creation helpful
- Simple neural network structure satisfying results

- Data set big ➡ less neurons necessary
- more neurons ➡ better approximation outside of data sets

- No actually interpretation of net structure with weight matrix

High Order Neural Networks (HONN)

MLP (Multilayer Perceptron):

$$out_j = \sigma\left(\sum_k w_{jk}\, x_k + b_j\right)$$

Linear combination of input

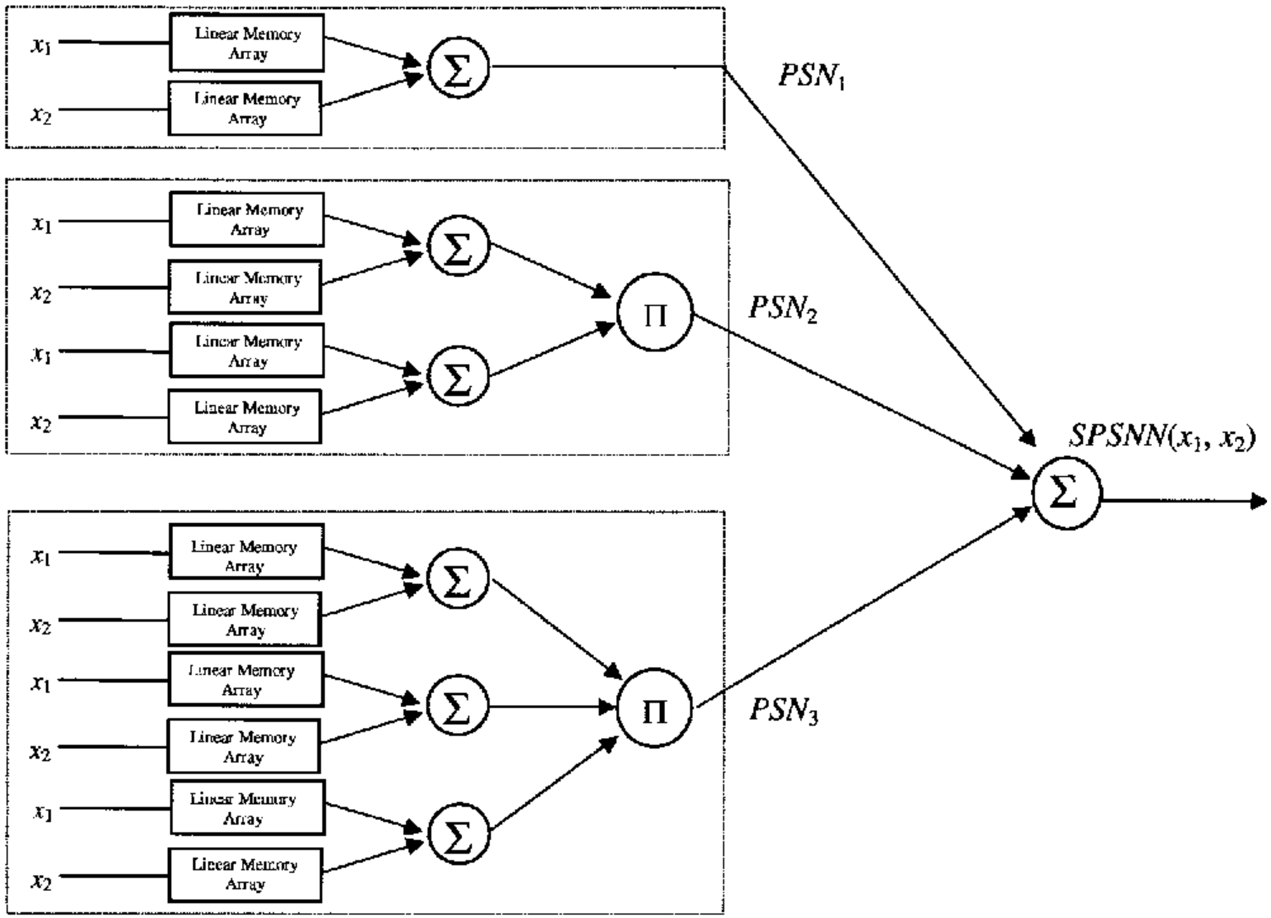
HONN (High Order Neural Network):

combine their inputs nonlinear

High Order Neural Networks (HONN)

- – more expensive computationally
- + capture high-order correlations
- + better mapping of nonlinear systems
  (might be possible with a lot of hidden neurons in MLP)
- + better generalization properties

High Order Neural Networks (HONN)



Li, Chien-Kuo. "A Sigma-Pi-Sigma Neural Network (SPSNN)." Neural Processing Letters 17 (2003): 1-19.

High Order Neural Networks (HONN)

Mathematical Modelling and Simulation

Sigma-Pi model:

Inputs of sigma-pi neurons are grouped – conjuncts

Output: applying activation function on weighted sum of products of each conjunct

$$y_i = \sum_j w_{ij} \prod_{k \in A_j} x_k$$

Number of conjunct and its member are fixed at the beginning – a priopri knowledge can help

High Order Neural Networks (HONN)

Mathematical Modelling and Simulation

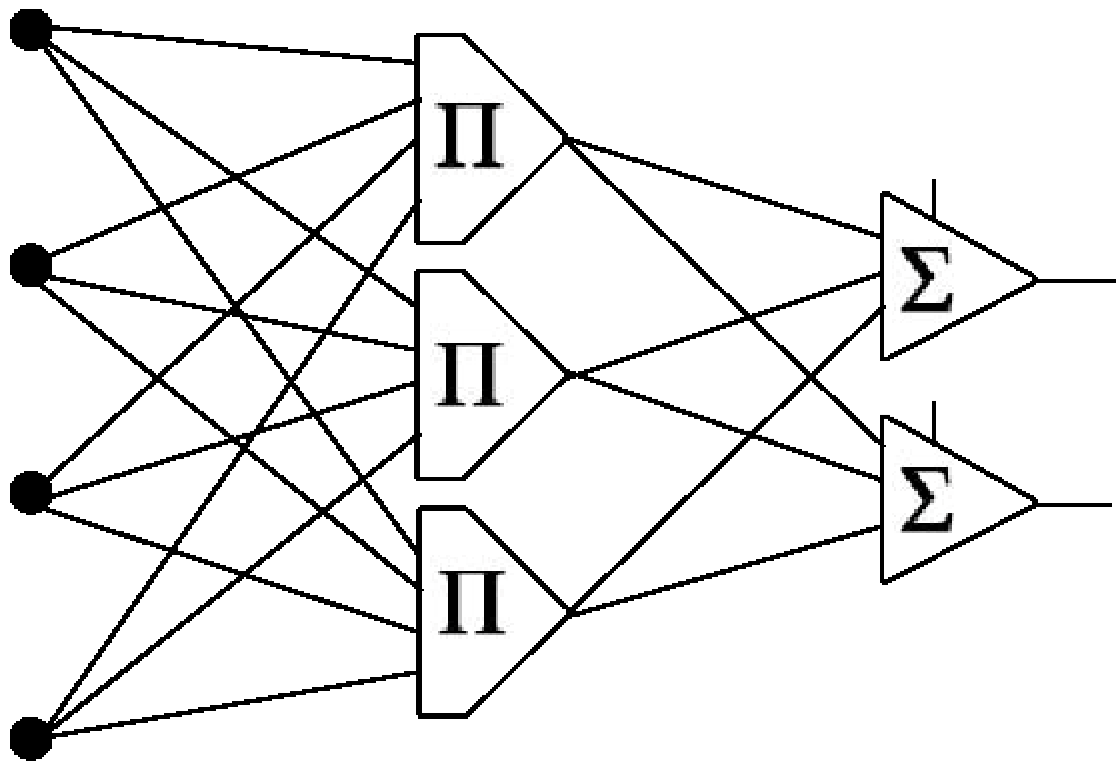


Fig. 1. HONEST network

El-Nabarawy, Islam et al. "Levenberg-Marquardt and Conjugate Gradient methods applied to a high-order neural network." The 2013 International Joint Conference on Neural Networks (IJCNN) (2013): 1-7.

High Order Neural Networks (HONN)

Mathematical Modelling and Simulation

HONEST (High order network with exponentail synaptic links)

$$y_i = \sum_j w_{ij} o_j + b_i$$

$$y_i = \sum_{h=1}^{Hidden} w_{hi} \prod_{j=1}^{Inputs} x_j^{p_{hj}} + b_j$$

p_{hj} ... is exponentail power associated with the synaptic conneting input h and neuron j

Number of terms in resulting polynom correspond to hidden neurons

High Order Neural Networks (HONN)

HONEST network

- result is the correlation of the input values
- possible to fix some variables and vary others to analyse
- validate the network solution

Equation Learner (EQL)

MLP are only as good as the data set

Test data outside of trainings set

→ possibility for bad results

Solution?

EQL – Equation Learner

concept for NN to learn a dynamic

Equation Learner (EQL)

Mathematical Modelling and Simulation

For the Algorithm:

$$C(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n}\left|\left|\psi(x_i) - y_i\right|\right|^2$$

$$z^l = w^l a^{l-1} + b^l$$

Input: u+2v

$$a^l := \begin{pmatrix} f_1(z_1^l), \dots, f_u(z_u^l), \\ g_1(z_{u+1}^l, z_{u+2}^l), \dots, g_v(z_{u+2v-1}^l, z_{u+2v}^l) \end{pmatrix}$$

Output: u+v

Equation Learner (EQL)

Mathematical Modelling and Simulation

$$a^l := (f_1(z_1^l), \ldots, f_u(z_u^l), g_1(z_{u+1}^l, z_{u+2}^l), \ldots, g_v(z_{u+2v-1}^l, z_{u+2v}^l))$$

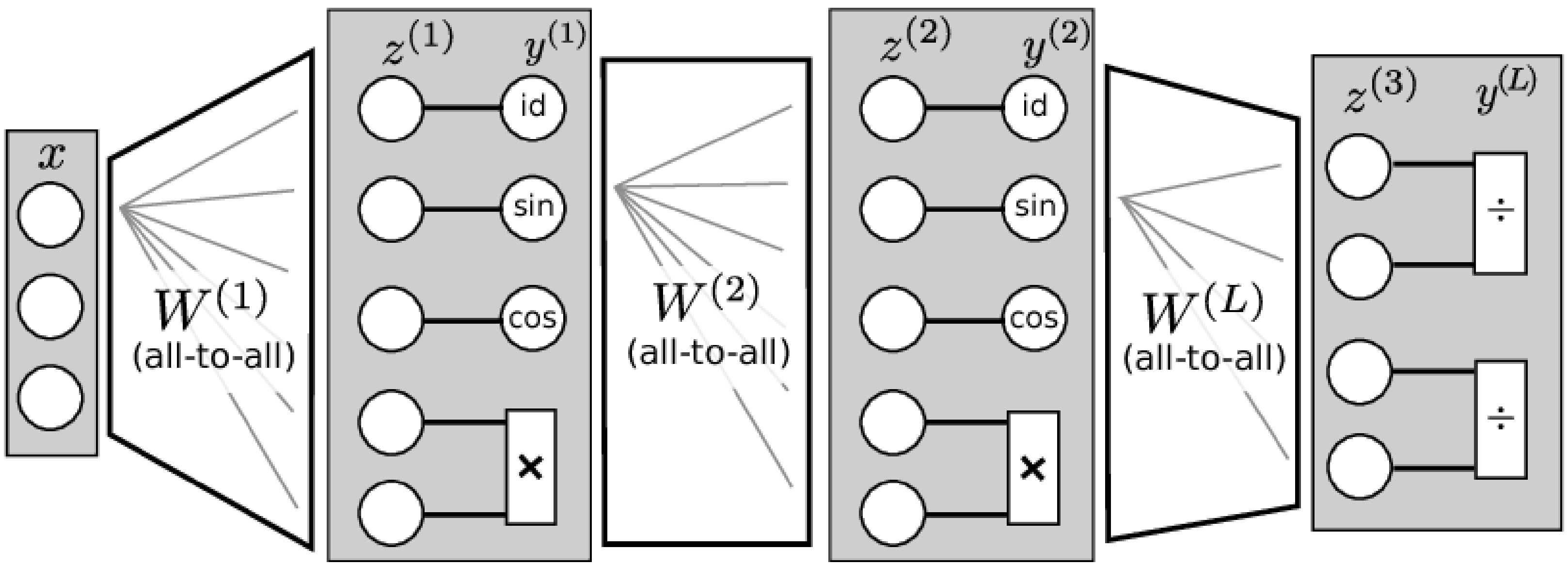
Binary Units:

$$f_i(z_i) = \begin{cases} z_i & I_i = 0 \\ \sin(z_i) & I_i = 1 \\ \cos(z_i) & I_i = 2 \\ sigm(z_i) & I_i = 3 \end{cases}$$

Multiplication Units:

$$g_j(z_{u+2j-1}, z_{u+2j}) = z_{u+2j-1} \cdot z_{u+2j}$$

Equation Learner (EQL)



Not included:

Radial basis functions
Logarithm function
Root functions

Equation Learner (EQL)

For Training:

$$C(\boldsymbol{x}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left|\left|\psi(x_i) - y_i\right|\right|^2}_{L_2 \text{ loss}} + \lambda \underbrace{\sum_{l=1}^{L}\left|W^l\right|_1}_{L_1 \text{ regulation}}$$

$\lambda = 0$ starting value to prevent over regulation

$\lambda > 0, t > t_1$ to prevent negativity

$\lambda = 0, t > t_2$ in the end of the algorithm

Stoachstic gradient descent algorithm with mini-patch and Adam algorithm

$$\theta_{t+1} = \theta_t + Adam\left(\frac{\partial C_t}{\partial \theta}, \alpha\right)$$

$a = 0.001$ step size

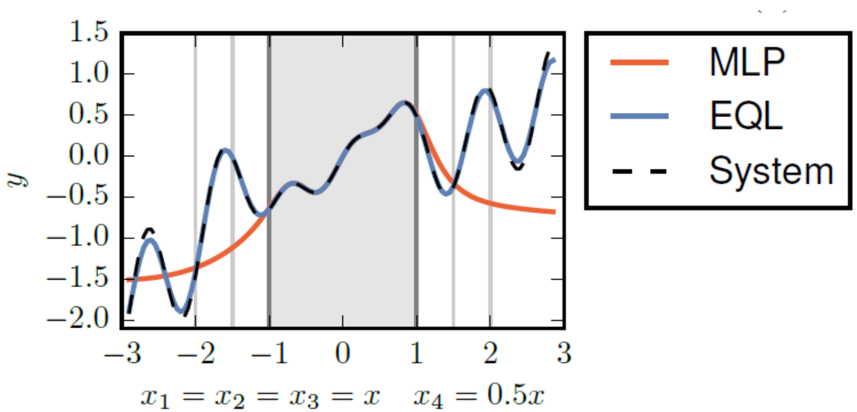
20 mini-patch size

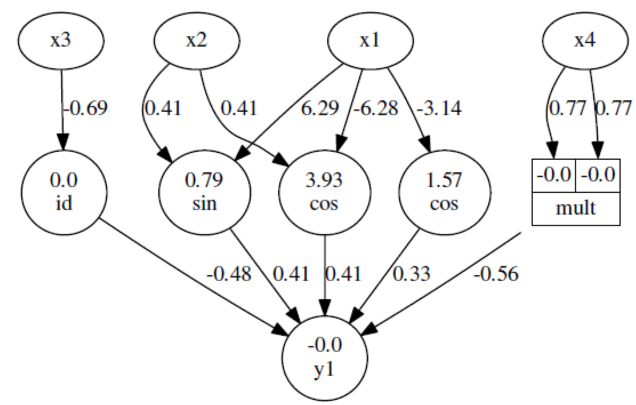
(standard gradient descent also works)

Equation Learner (EQL)

Mathematical Modelling and Simulation

L_1 regulation should sparse connections:





learned formula: $0.33\cos(3.14x_1 + 1.57) + 0.33x_3 - 0.33x4^2 +$

$0.41\cos(-6.28x_1 + 3.93 + 0.41x_2) + 0.41\sin(6.29x_1 + 0.79 + 0.41x_2)$

https://www.is.mpg.de/uploads_file/attachment/attachment/493/ICML2018-poster.pdf

References

- Different training algorithms https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network
- Winkler, Stefanie, Martin Bicher, Andreas Körner, und Felix Breitenecker. „Modelling and Simulation of Hybrid Systems with Neural Networks“, 111–12. ARGESIM Publisher Vienna, 2018. https://doi.org/10.11128/arep.55.a55276.
- Martius, Georg, und Christoph H. Lampert. „Extrapolation and Learning Equations“. *ArXiv:1610.02995 [Cs]*, 10. Oktober 2016. http://arxiv.org/abs/1610.02995.
- Abdelbar, Ashraf M., und Gene A. Tagliarini. „HONEST: a new high order feedforward neural network“, 1996. https://doi.org/10.1109/ICNN.1996.549078.
- Cheng, Min-Yuan, Hsing-Chih Tsai, und Erick Sudjono. „Evolutionary Fuzzy Hybrid Neural Network for Dynamic Project Success Assessment in Construction Industry“. *Automation in Construction* 21 (Januar 2012): 46–51. https://doi.org/10.1016/j.autcon.2011.05.011.