MAE 598 PROJECT

Self-Landing Rocket Guided by Deep Q-Learning Neural Network

Abstract

This paper describes a self-landing rocket guided by deep Q-learning using a neural network. The discussed methods involve guiding the rocket on the optimal trajectory towards the origin from a given starting point. The optimal trajectory is defined as the trajectory that will move the rocket from the initial conditions to the origin in the shortest number of steps. The optimal trajectory is implicitly given in the loss function through the defined reward. The reward for this model is the negative sum of error of x and y squared. The trained Q value weights the reward of a state-action pair along with its output probability. Different network structures and model parameters are explored, and their corresponding results are reported. It is proved in the results that a typical network structure involving trainable multipliers, trainable biases, and activation functions was able to land a rocket with nonlinear state dynamics in a sufficient length network training period.

Steven Elliott

Sjellio1@asu.edu

Code Available: https://github.com/sjellio1/MAE598_Project

Table of Contents

Nomenclature

$x - horizontal\ position$

$\dot{x} - rate\ of\ change\ of\ horizontal\ position$

$y - vertical\ position$

$\dot{y} - rate\ of\ change\ of\ vertical\ position$

$\theta - angle$

$\theta - rate\ of\ change\ of\ angle$

$T - thrust$

$\tau_1 - torque\ left$

$\tau_2 - torque\ right$

$L - loss\ function$

$\Pi - neural\ network\ numerical\ outputs$

$\varepsilon - error$

$a - action$

$r - reward$

$r^d - discounted\ reward$

Design Problem Statement

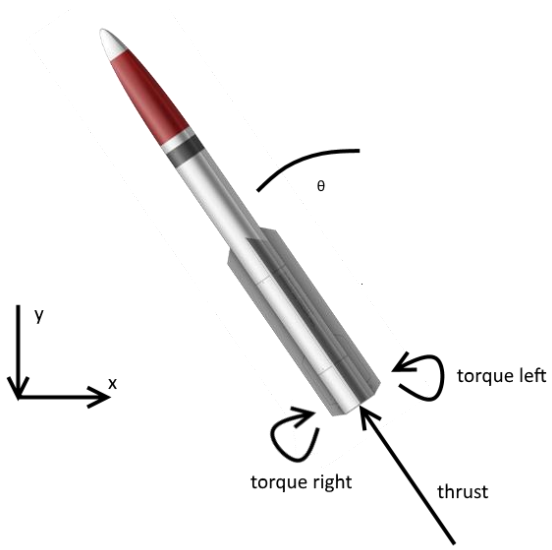


Figure 1 Rocket Model

This project started with the idea for a rocket landing game in which the user would user three controls (thrust, torque left, torque right) to steer a rocket from a random initial position in addition to a random initial angle (±30°). From the initial idea for the game, came the thought of applying machine learning aspect to optimize a neural network to land the rocket. The input consists of a 0-5 integer corresponding to 6 states. These states, shown below, correspond to position, attitude, rate of change. The coordinate system is defined in Figure 1.

$$x_1 = x \qquad x_4 = \dot{x}$$
$$x_2 = y \qquad x_5 = \dot{y}$$
$$x_3 = \theta \qquad x_6 = \dot{\theta}$$

The neural network output consists of a 0-5 integer corresponding to 3 Booleans (thrust, torque left, torque right). The output is enumerated as follows:

$$Output = 0: No\ Thrust;\ No\ Torque$$
$$Output = 1: No\ Thrust;\ Torque\ Left$$
$$Output = 2: No\ Thrust;\ Torque\ Right$$
$$Output = 3: Thrust;\ No\ Torque$$
$$Output = 4: Thrust;\ Torque\ Left$$
$$Output = 5: Thrust;\ Torque\ Right$$

The Model

The model for the rocket is shown in Figure 1**Error! Reference source not found.**, but for this project can be thought of as a single 6 state point in space. Thrust is applied through the current angle of the rocket while the torques are applied to the angle of the rocket. The state equations of the model are nonlinear and are designated as followed:

$$\dot{x}_1 = x_4$$

$$\dot{x}_2 = x_5$$

$$\dot{x}_3 = x_6$$

$$\dot{x}_4 = -T \sin x_3$$

$$\dot{x}_5 = -T \cos x_3$$

$$\dot{x}_6 = \tau_1 - \tau_2$$

For the purposes of this project, the thrust and torques were applied directly as accelerations while the rocket mass and inertia are divided through and canceled. The constants yielding these accelerations are defined as parameters. Each step consists of $0.01\overline{6}$sec or 60 frames per second. If the rocket does not touch y=0 in 1000 steps, the controller will time out and end the trial.

Model Analysis

The rocket defining model proposes an interesting problem for the deep Q learning technique. Since the thrust depends on current angle θ of the rocket, the network indirectly learns to move the x-position through the rocket rotation. However, not only will the thrust now affect the x-position but will now adversely disrupt the y-position by increasing the upward acceleration.

The problem was observed with many different reward definitions, including an instantaneous error to the model. Through trial and error, it was found that solely providing the rocket with an ending reward was best the solution to the problem. This way, the network accepted a penalty in the form of the ending distance from the intended point: the origin.

Furthermore, in each of the parameter sets discussed later, the thrust acceleration is greater than the acceleration of gravity. This implies that the network now needed to be trained to toggle the thrust, striking a balance between improving the x and y coordinates simultaneously. The actions and steps will be calculated at a rate of 60 frames per second and ideally, this toggling should be very fast, making the rocket's trajectory visually smooth to an observer.

Optimization Study

The optimization problem is the following: optimize a neural network that can input the current rocket state and output a discrete frame control to most efficiently land the rocket at a designated landing position and state. This objective can be lessened down into individual state-action pairs. A state-action pair is the corresponding chosen action of the network based on the inputted state of the rocket. This was achieved using a tensorflow (Google, n.d.) Python API as well as a framework developed by Open AI Gym (Open AI, n.d.). The neural network and loss function was the only remaining segment that needed to be developed. The framework for the neural network implementation and model feedback can be visualized in Figure 2.

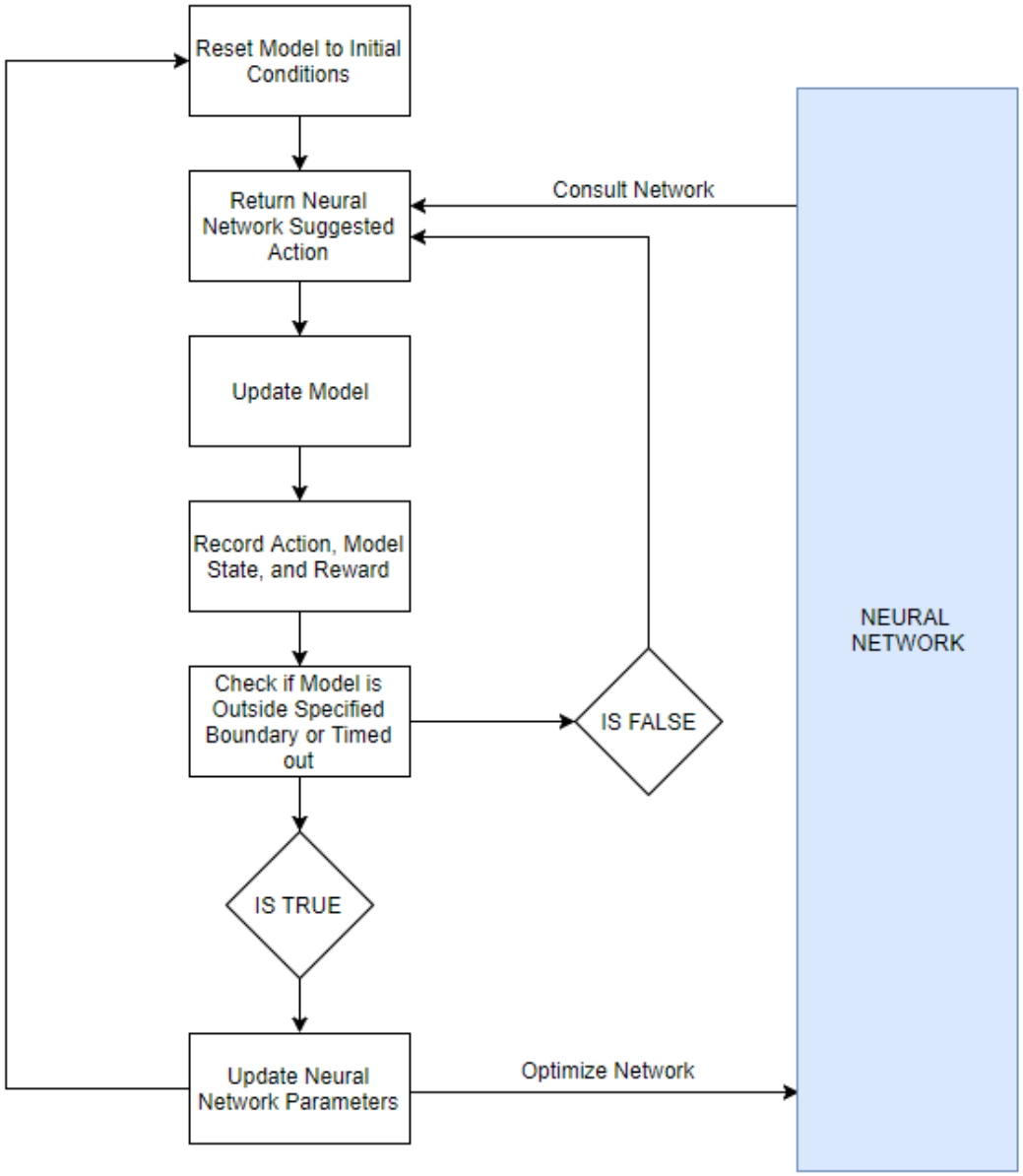


Figure 2 Neural Network Implementation

The concept of optimizing the action for any given state is called “Q-Learning”. However, because the states are continuous as opposed to discrete values, “Deep Q-Learning” is the employed technique, where the goal is to fit the continuous state’s space enough that the output node with the greatest value will be the optimum action to take (Mnih, et al., 2013). During the training period, the outputs are normalized and treated as probabilities; therefore, exploring all possible trajectories and ensuring there are no unknown optimum paths,

The structure of a neural network has a huge impact on the performance of the network (Kenneth O. Stanley & Miikkulainen). This has been one of the leading aspects of research in the field of machine learning. Although the parameters describing the relationship between inputs and outputs are refined through numerous trials, a specific network topology had to be decided on for this project.

To test the network structure’s effect on performance, different structures were tested. Due to the inherent nonlinearity in the rocket model, a least 1 hidden layer (layer between input and output layer) was chosen. From there, an extra hidden layer and an increase in nodes at these layers was tested. Each hidden layer consisted of a bias parameter and a ReLU (Rectified Linear Unit) activation function. The structures can be visualized in Figure 3, Figure 4, and Figure 5.

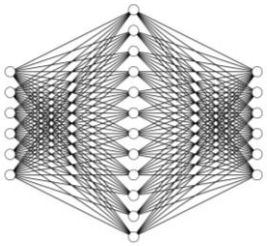


Figure 3 One 12-Node Hidden Layer

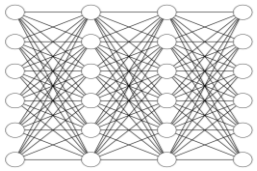


Figure 4 Two 6-Node Hidden Layers

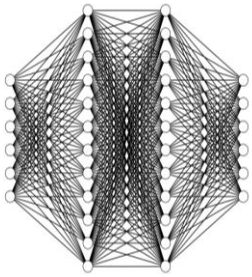


Figure 5 Two 12-Node Hidden Layers

Even more important than the neural network structure is the function being optimized. The loss function is what the network uses to move its characteristics in the intended optimal direction. The implementation of the network aims to minimize the following loss function:

$$L = -mean\left(\log\left(\Pi_i^T a_i\right) r_i^d\right)$$

The first term, $\log\left(\Pi_i^T a_i\right)$, is a term to determine how likely the network is to make that action again. The term Π is the output which is a 6-element vector consisting of the numerical network outputs (these outputs correspond to the probabilities of each of the 6 actions to take). The controller will choose an action randomly given these 6 probabilities. The a can be thought of as a 6-element vector consisting of five 0’s and a single 1 corresponding to the action taken. So, if the controller took an action with a high probability, the logarithm term will be closer to 0. If the controller took an action that is unlikely to be taken again given the same state inputs, this term will approach negative infinity. The purpose of this loss function is to weight the rewards of the actions that the network prefers higher than the rewards of actions that the network chose ‘by accident’.

The loss function is used to calculate the gradients for each of the variables being optimized in the network. The full list of trainable variables consists of the weights from each node connection as well as biases added at each node. These gradients were applied in Tensorflow's built in Adam Optimizer algorithm that had a learning rate of 0.001. The learning rate is used to decide the rate at which the trainable variables will change with the inputted gradients.

In the case of an optimal landing point, the reward will be thought of as the negative error. For this model, the error is only applied on the last iteration of a trial. The goal of the rocket is to reach the origin, so the penalties are considered the distance from zero squared.

The reward is discounted, to apply a penalty throughout all state-action pairs depending on how successful the trial was. The parameter γ is used as a weight of the importance of future errors on instantaneous state-action pairs. In a n-step long trial, the following equations show how the discounted reward is calculated.

$$\varepsilon = {x_n}^2 + {y_n}^2$$

$$r_n = -\varepsilon$$

$$r_i^d = r_i + \sum_{s=0}^{i-1} \gamma r_s$$

In this project, γ is chosen to be 1. This makes the rewards for each state-action pair equal to the reward at the last state. The reason for this was because this model was not defined as time sensitive, but rather had a focus solely on the final position of the rocket.

Parameters

The parameters in this project are defined as chosen values that are constant in a trainable period and will affect the solution. The parameters are defined as variables that are not optimized but that affect the behavior of the rocket in its environment. To reduce the number of parameters, normalization was applied in every possible instance. This includes things like treating the rocket as a discrete point in space and normalizing x and y coordinates. The parameters are shown in Table 1.

	Parameter Set 1	Parameter Set 2	Parameter Set 3	Parameter Set 4
Thrust Acceleration (units/sec^2)	0.18	0.09	0.18	0.18
Gravitational Acceleration (units/sec^2)	0.12	0.12	0.12	0.12
Rotational Acceleration (degrees/sec^2)	20	20	20	20
Initial x	-0.8	-0.8	-0.8	-0.8
Initial y	-0.8	-0.8	-0.2	-0.2
Initial θ	0	0	0	Random ±30
Frame Time (frames/sec)	60	60	60	60

Table 1 Model Parameters

Results Discussion

The results for a neural network optimization problem across a 10,000-trial training period are difficult to convey. However, overall trends can be identified and are visualized below. The goal of the project was to train to network to consistently land the rocket as close to the origin as possible. For each presented set of parameters, the x-position of every landed trial out of 10,000 is shown in a histogram along with a percentage of landed cases in the appendix. All cases that did not successfully reach y=0 are omitted.

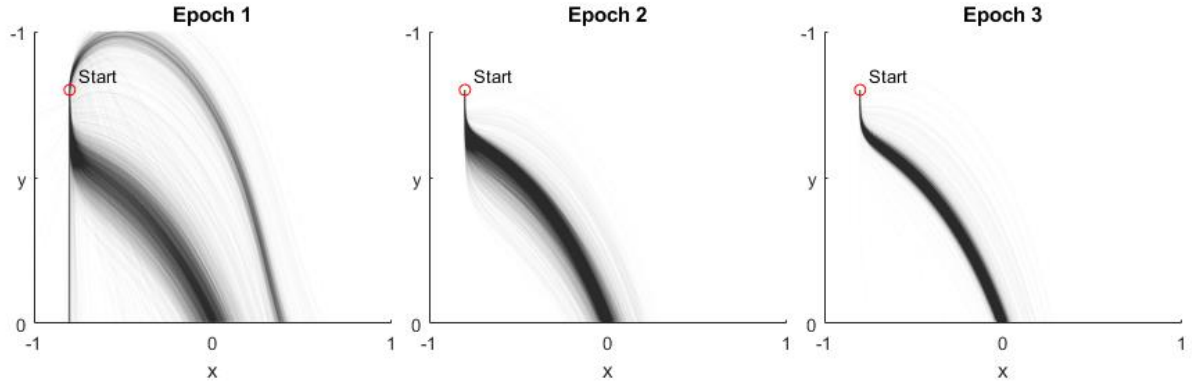


Figure 6 Full Trial Visualization for Two 12-Node Hidden Layer Structure / Parameter Set 1

Figure 6 shows all 10,000 trajectories split up by 3,333 trial epochs and overlaid on one another. Epoch 1 shows the main trends that kept reoccurring. In the loss function, there appeared to be local minimums going 'over the top' and just falling straight to the ground. This came from the indirect effect of thrust on x-position through angle θ. If the rocket had been rotated counter-clockwise and thrusted, the x-position would have worsened, therefore causing the reward to drop. This would lead to a momentary diversion until the controller learned to steer the rocket clockwise; this was also a large factor in the chosen slow learning rate. A similar occurrence would happen with the over-the-top trajectory occurring before the network was optimized to toggle the thrust, rather than thrust every step.

	Epoch 1	Epoch 2	Epoch 3
Two 12-Node Hidden Layers Parameter Set 1	0.12518	0.20353	0.00110
Two 6-Node Hidden Layers Parameter Set 1	0.02684	0.00557	0.00161
One 12-Node Hidden Layer Parameter Set 1	0.07491	0.12733	0.00059
Two 12-Node Hidden Layers Parameter Set 2	0.23933	0.07062	0.02998
Two 12-Node Hidden Layers Parameter Set 3	0.84448	0.34232	0.09765
Two 12-Node Hidden Layers Parameter Set 4	0.43843	0.04513	0.05649

Table 2 Average Final X Error for Multiple Parameter Sets

Table 2 shows the absolute value of the mean of the final distance from the origins across all 1000 trials of a learning period. This provides a rough quantitative value into the performance of individual epochs.

There are different criteria to consider in the results. The first is the final mean position after a sufficient training period, the other is the rate of convergence. Ideally, the network should be most optimized at Epoch 3 and therefore be most indicative of network performance. In this instance, the Two 12-Node Hidden Layers structure with Parameter Set 1 had the most consistent final Epoch. This was the expected result due to the nonlinearity between the states and controls of the rocket (generally a more complex model requires a more complex network topology). It is very possible and even likely that there is an alternate and/or more complex network structure that will land the rocket more efficiently and from the initial conditions. Though, with the algorithm and process discussed in the report, a solution can only be found through iterations of different topologies and trial and error. An additional subject of optimization of a neural network is network topologies that adapt to their model (Kenneth O. Stanley & Miikkulainen).

References

Google. (n.d.). *TensorFlow*. Retrieved from https://www.tensorflow.org/

Kenneth O. Stanley, & Miikkulainen, R. (n.d.). *Evolving Neural Networks Through Augmenting Topologies.* The MIT Press Journals.

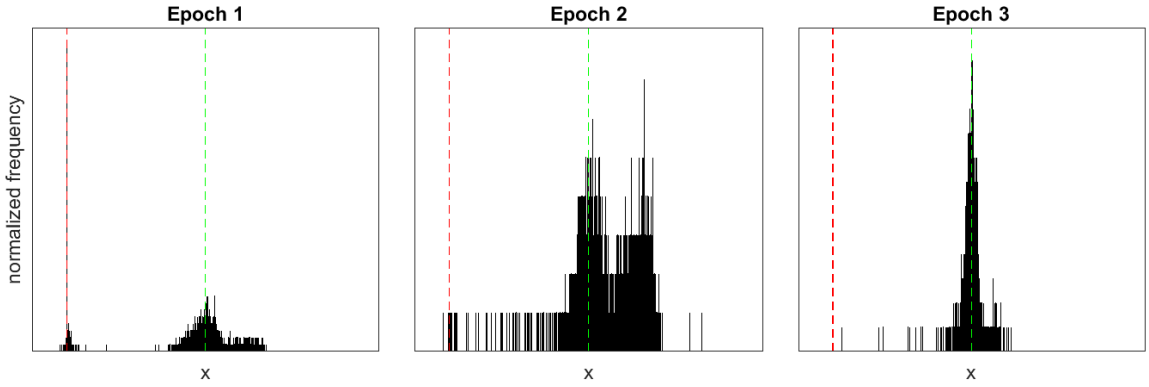
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* DeepMind Technologies.

Open AI. (n.d.). *Gym*. Retrieved from https://github.com/openai/gym.

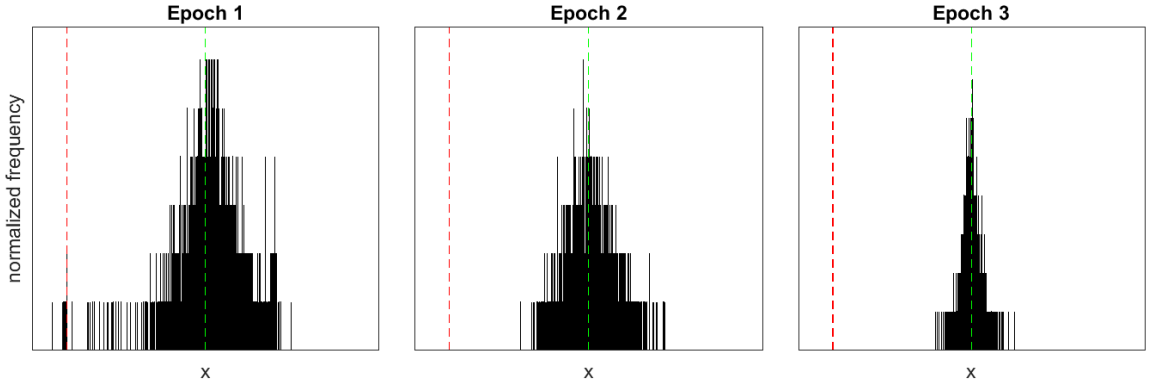
Appendix

The red dotted line shows the initial starting point, while the green dotted line shows the origin.

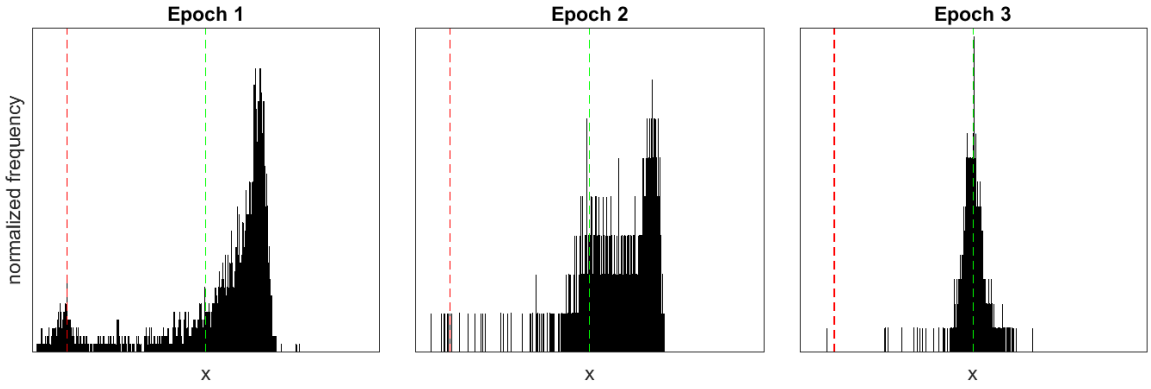
Two Hidden Layers with 12 Nodes – Parameter Set 1 (100.00% landed)



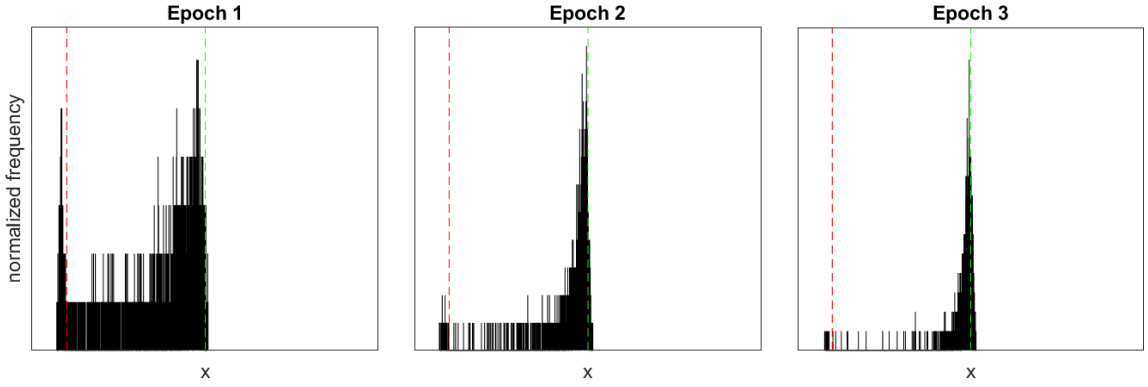
Two Hidden Layers with 6 Nodes – Parameter Set 1 (100.00% landed)



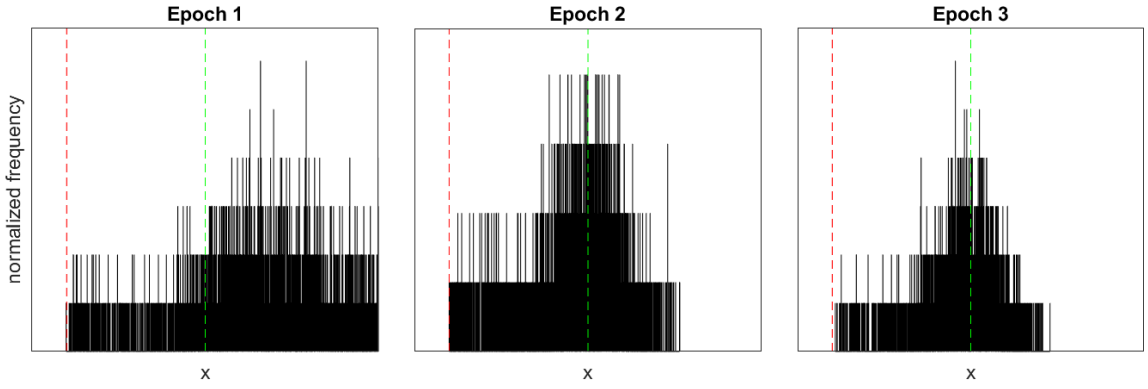
Two Hidden Layers with 12 Nodes – Parameter Set 1 (99.83% landed)



Two Hidden Layers with 12 Nodes – Parameter Set 2 (100.00% landed)



Two Hidden Layers with 12 Nodes – Parameter Set 4 (100.00% landed)



Two Hidden Layers with 12 Nodes – Parameter Set 3 (100.00% landed)