Fundamentals of Deep Learning

Part 2: How a Neural Network Trains

Agenda

- Part 1: An Introduction to Deep Learning
- Part 2: How a Neural Network Trains
- Part 3: Convolutional Neural Networks
- Part 4: Data Augmentation and Deployment
- Part 5: Pre-Trained Models
- Part 6: Advanced Architectures

Recap of the Exercise

What just happened?

Loaded and visualized our data

Edited our data (reshaped, normalized, to categorical)

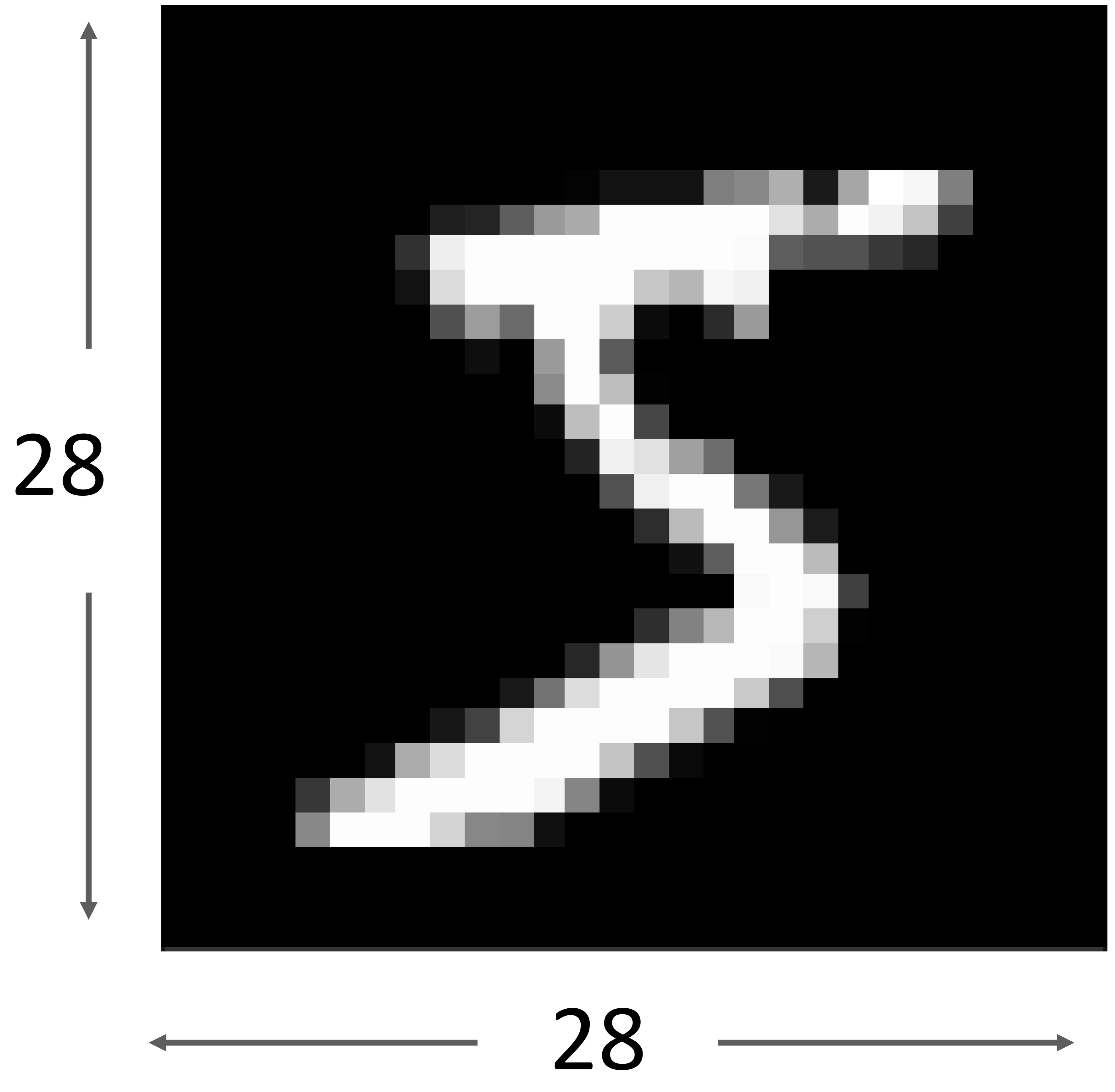
Created our model

Compiled our model

Trained the model on our data

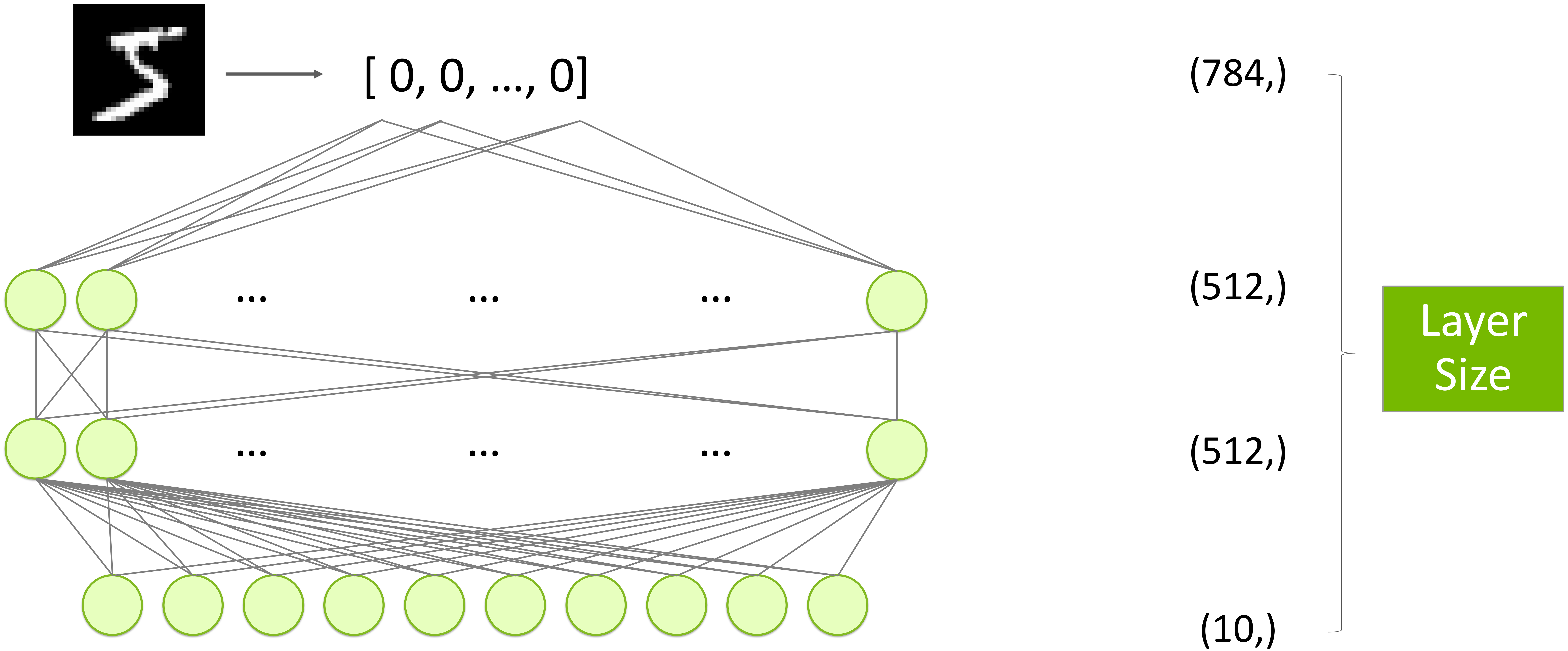
Data Preparation

Input as an Array



[0,0,0,24,75,184,185,78,32,55,0,0,0…]

An Untrained Model

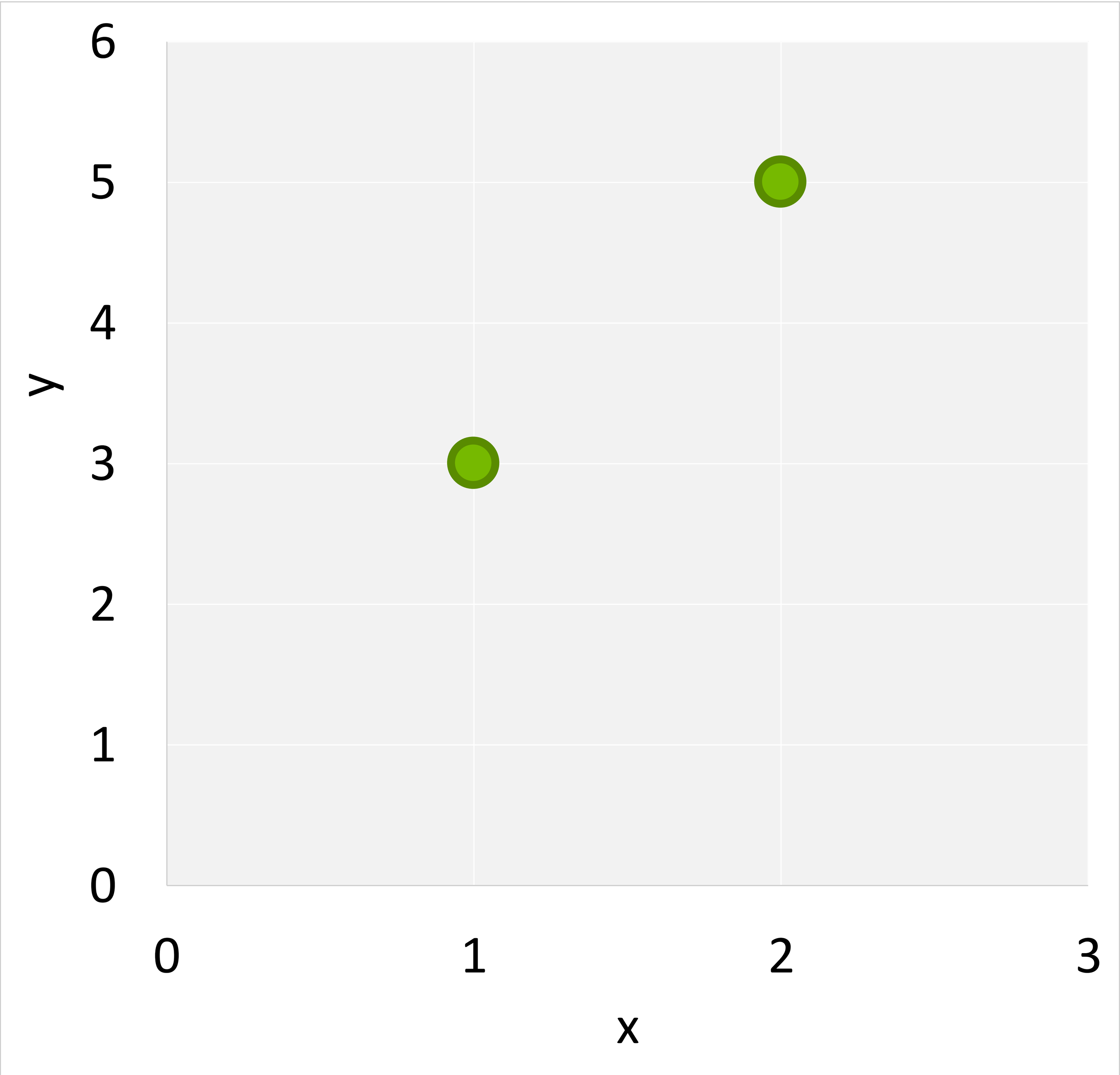


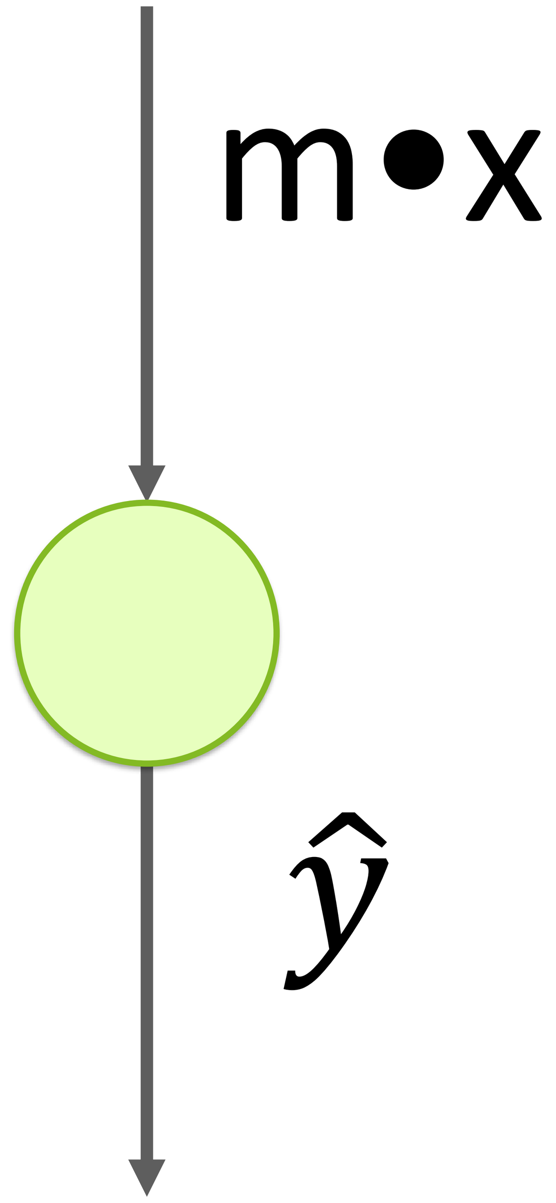
A Simpler Model

A Simpler Model

$$y = mx + b$$

x	y
1	3
2	5





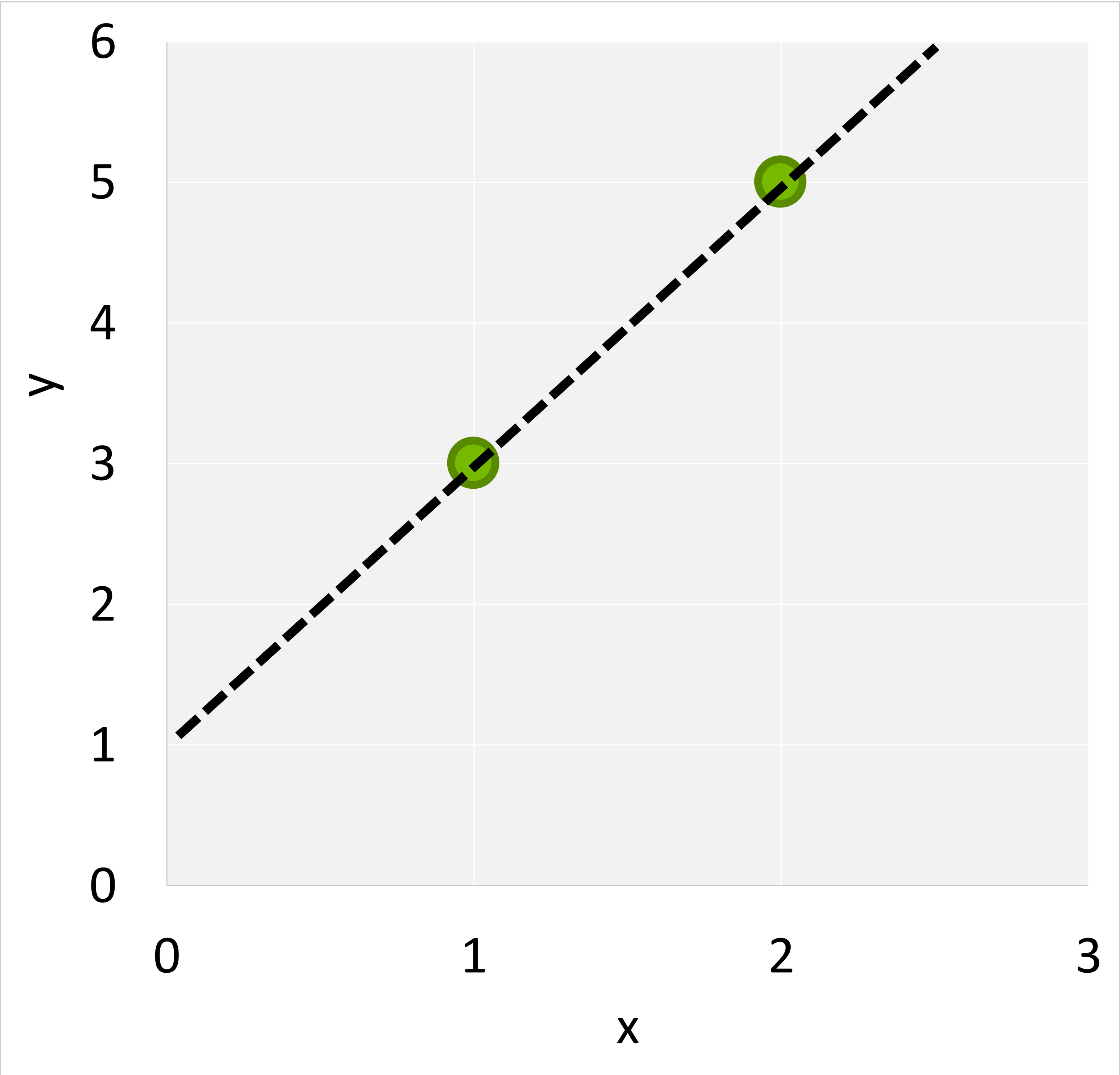
$m = ?$

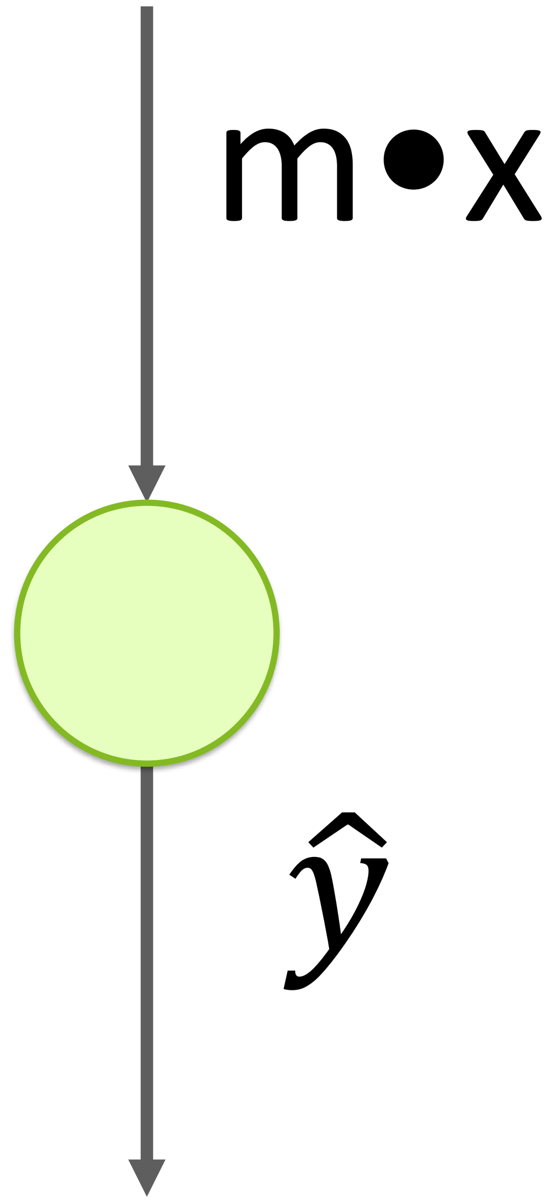
$\text{b} = ?$

A Simpler Model

$$y = mx + b$$

x	y
1	3
2	5





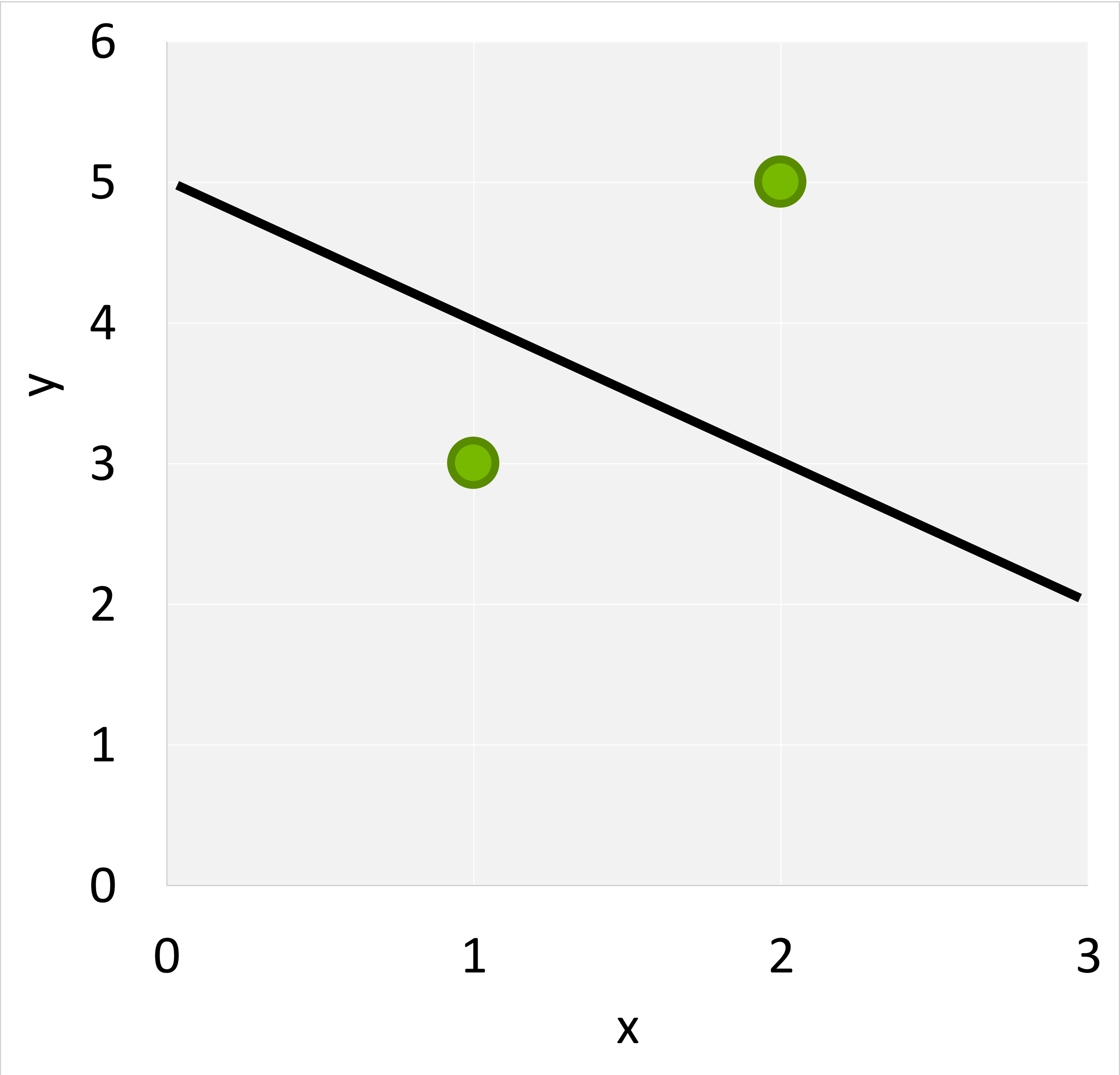
$$m = ?$$

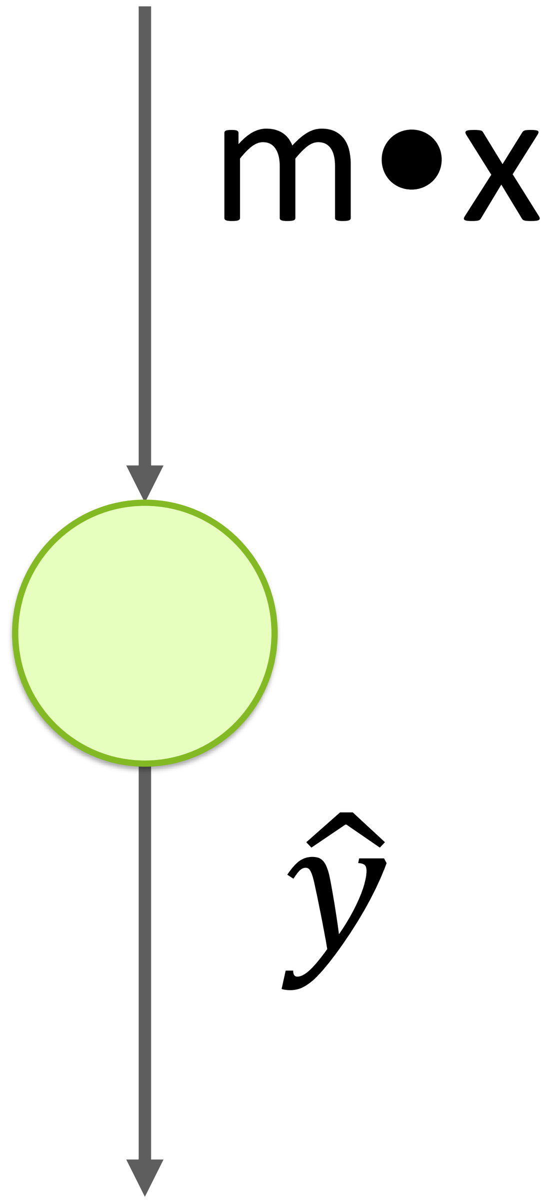
$$b = ?$$

A Simpler Model

$$y = mx + b$$

x	y	$\hat{y}$
1	3	4
2	5	3





Start Random

$$m = -1$$

$$b = 5$$

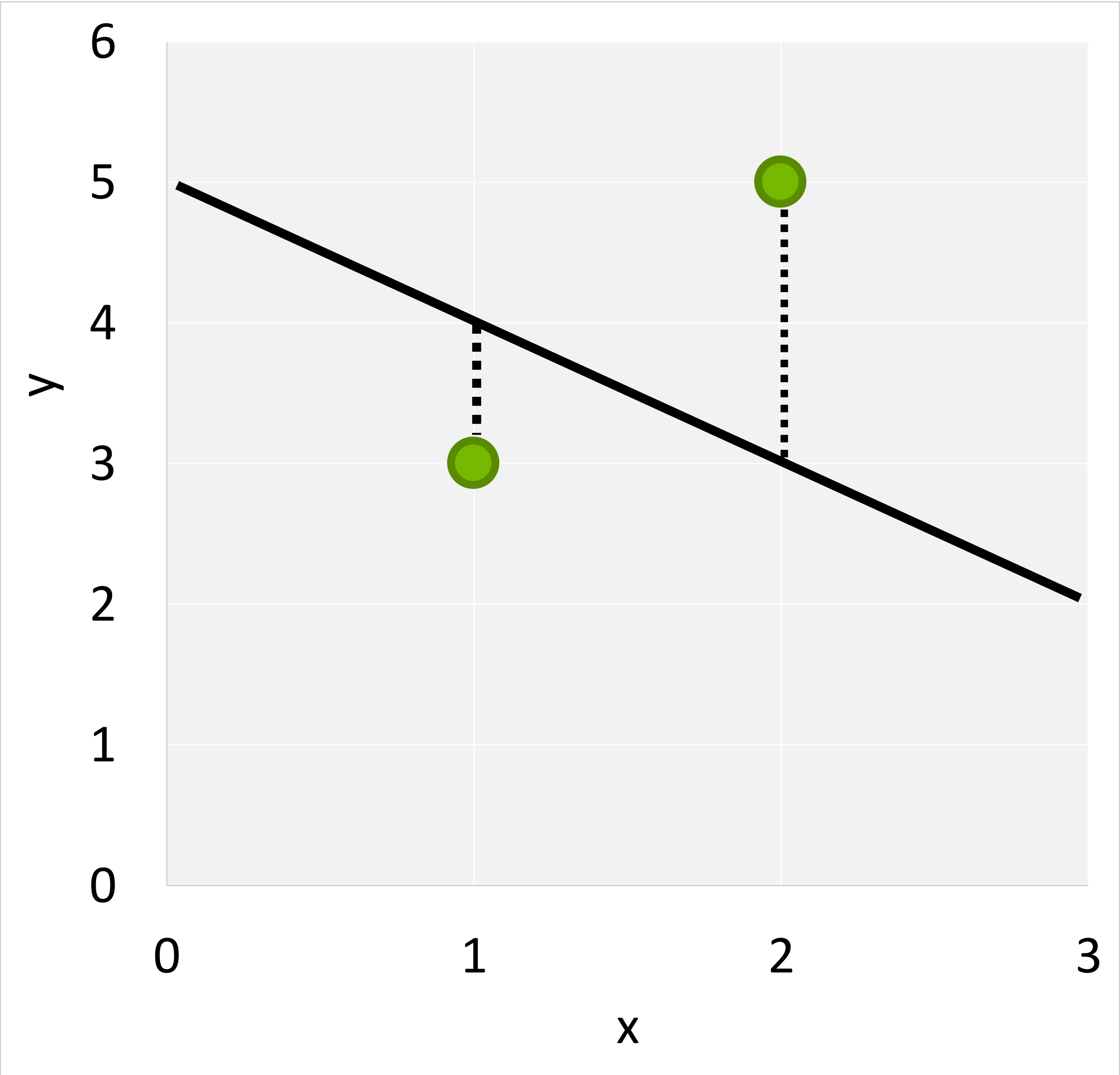
A Simpler Model

$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4

MSE = 2.5

RMSE = 1.6



$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

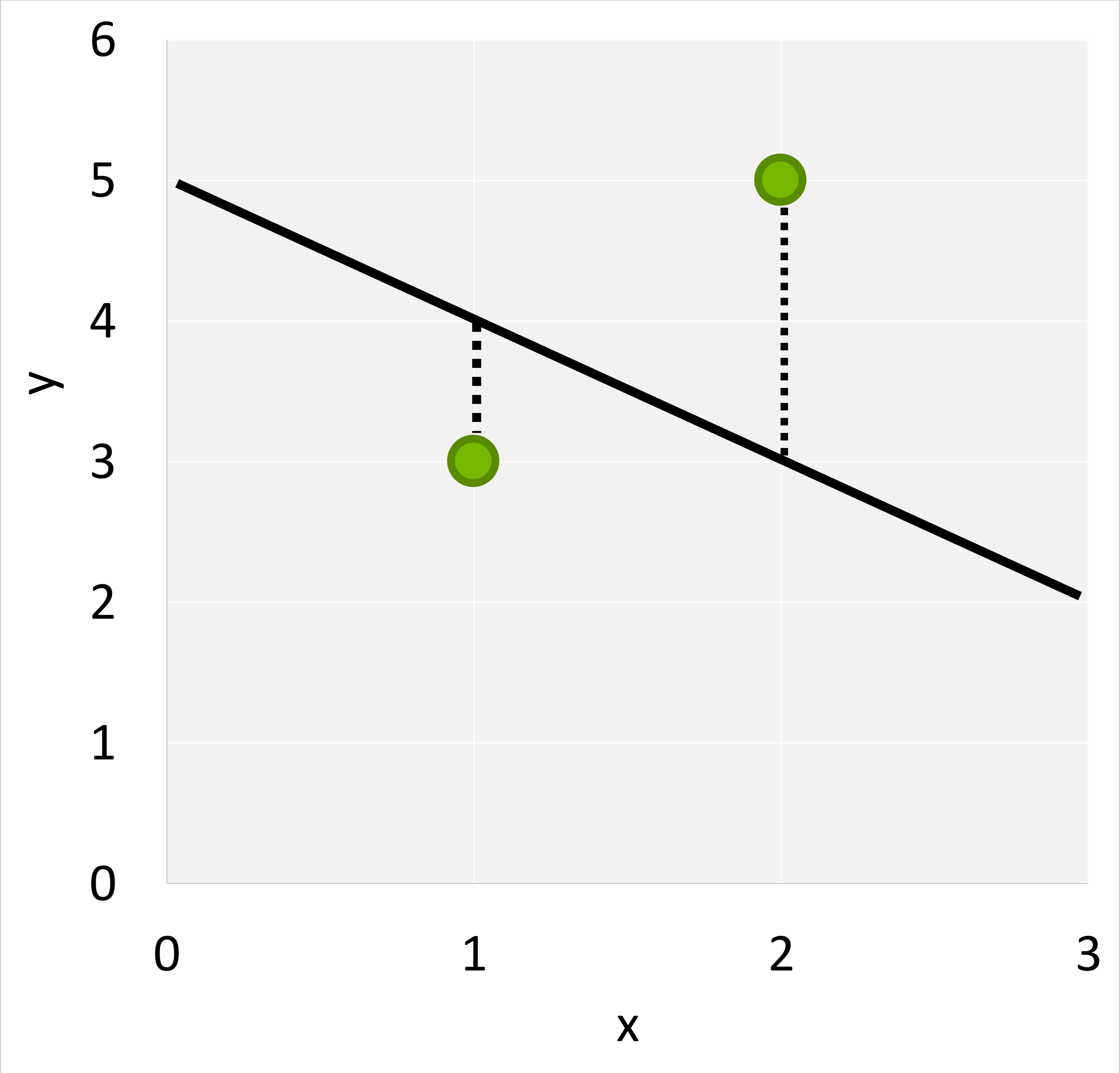
A Simpler Model

$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4

MSE = 2.5

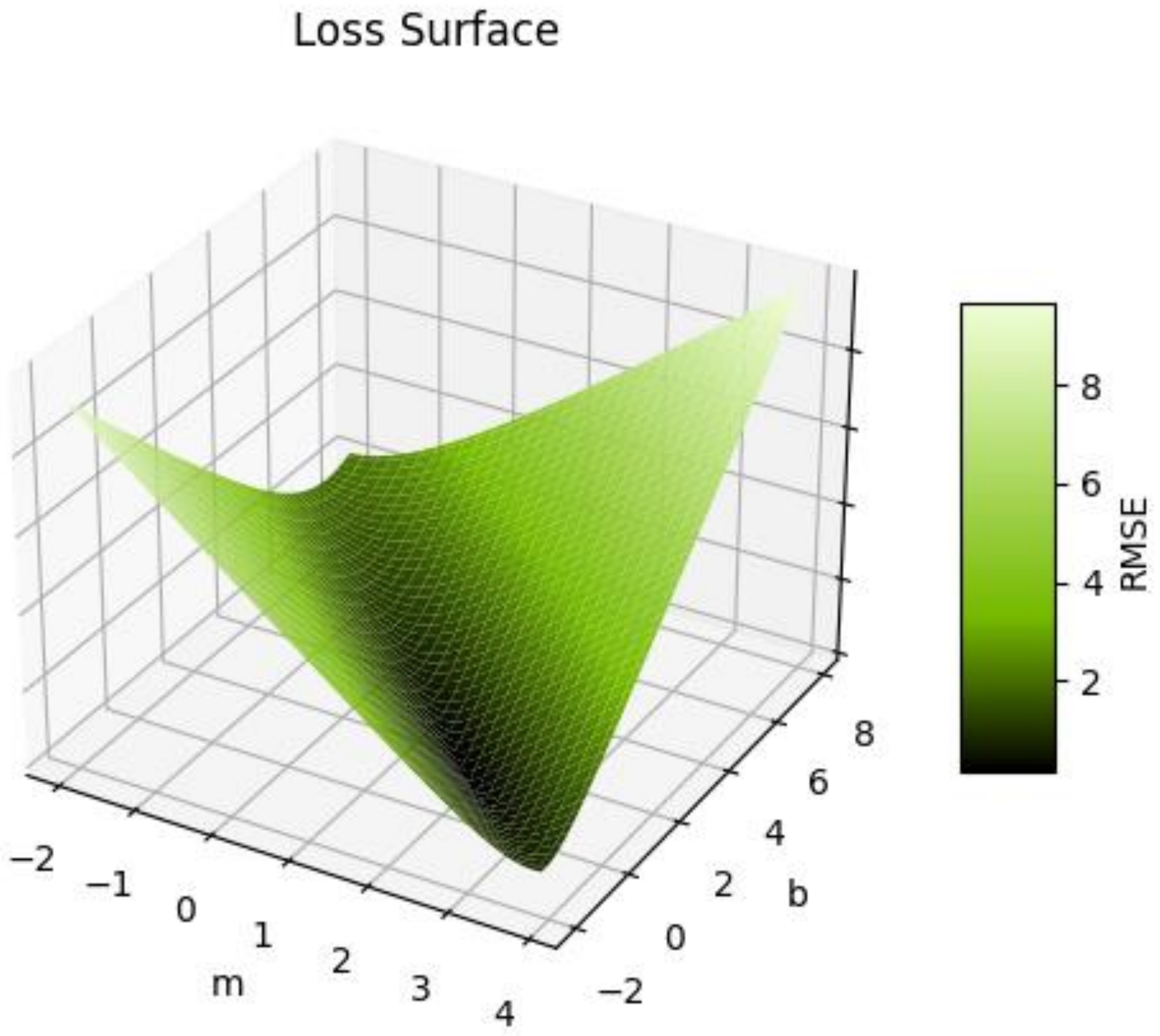
RMSE = 1.6



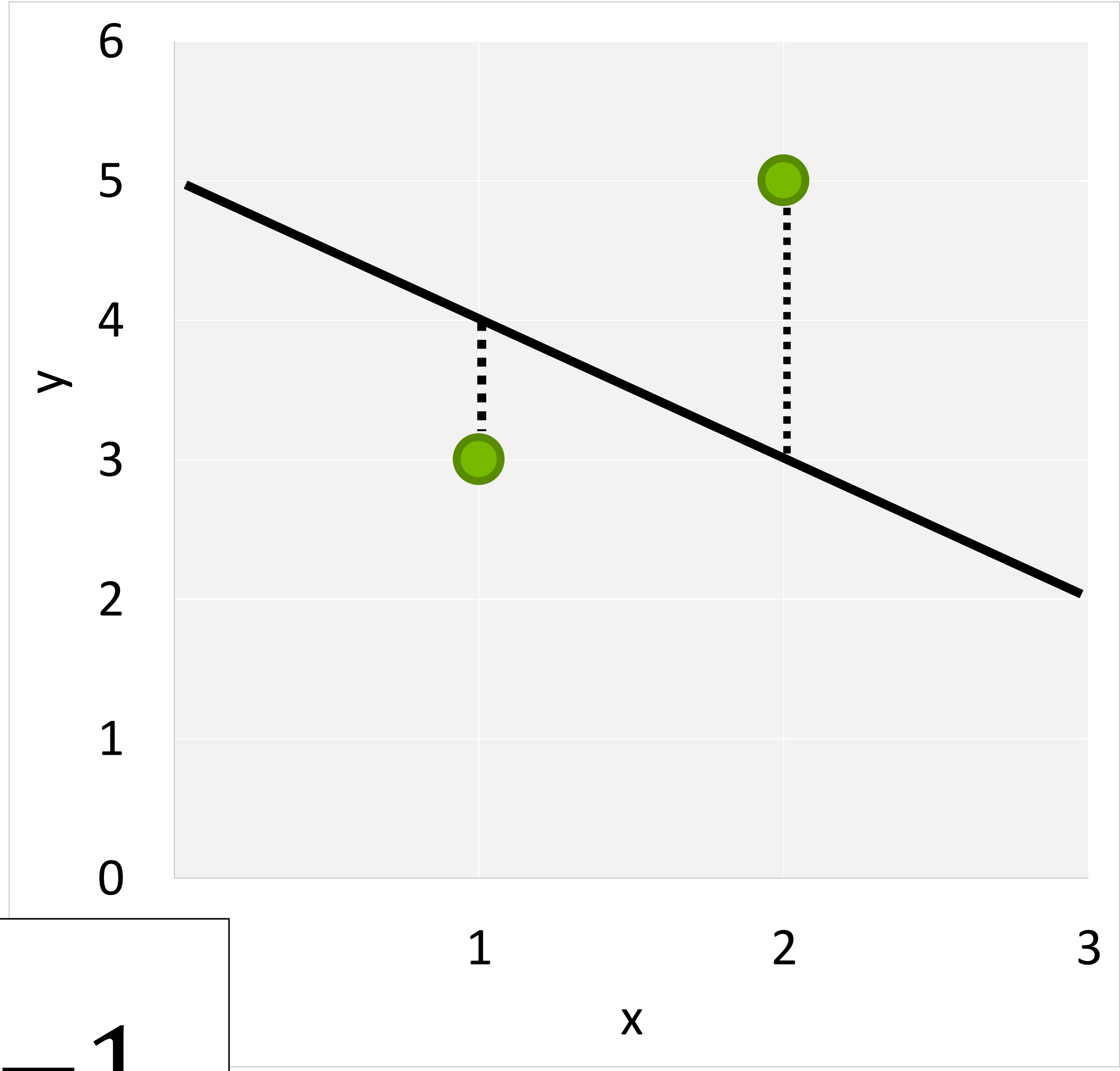

```python
data = [(1, 3), (2, 5)]
m = -1
b = 5


def get_rmse(data, m, b):
    """Calculates Mean Square Error"""
    n = len(data)
    squared_error = 0
    for x, y in data:
        # Find predicted y
        y_hat = m*x+b
        # Square difference between
        # prediction and true value
        squared_error += (
            y - y_hat)**2
    # Get average squared difference
    mse = squared_error / n
    # Square root for original units
    return mse ** .5
```

The Loss Curve

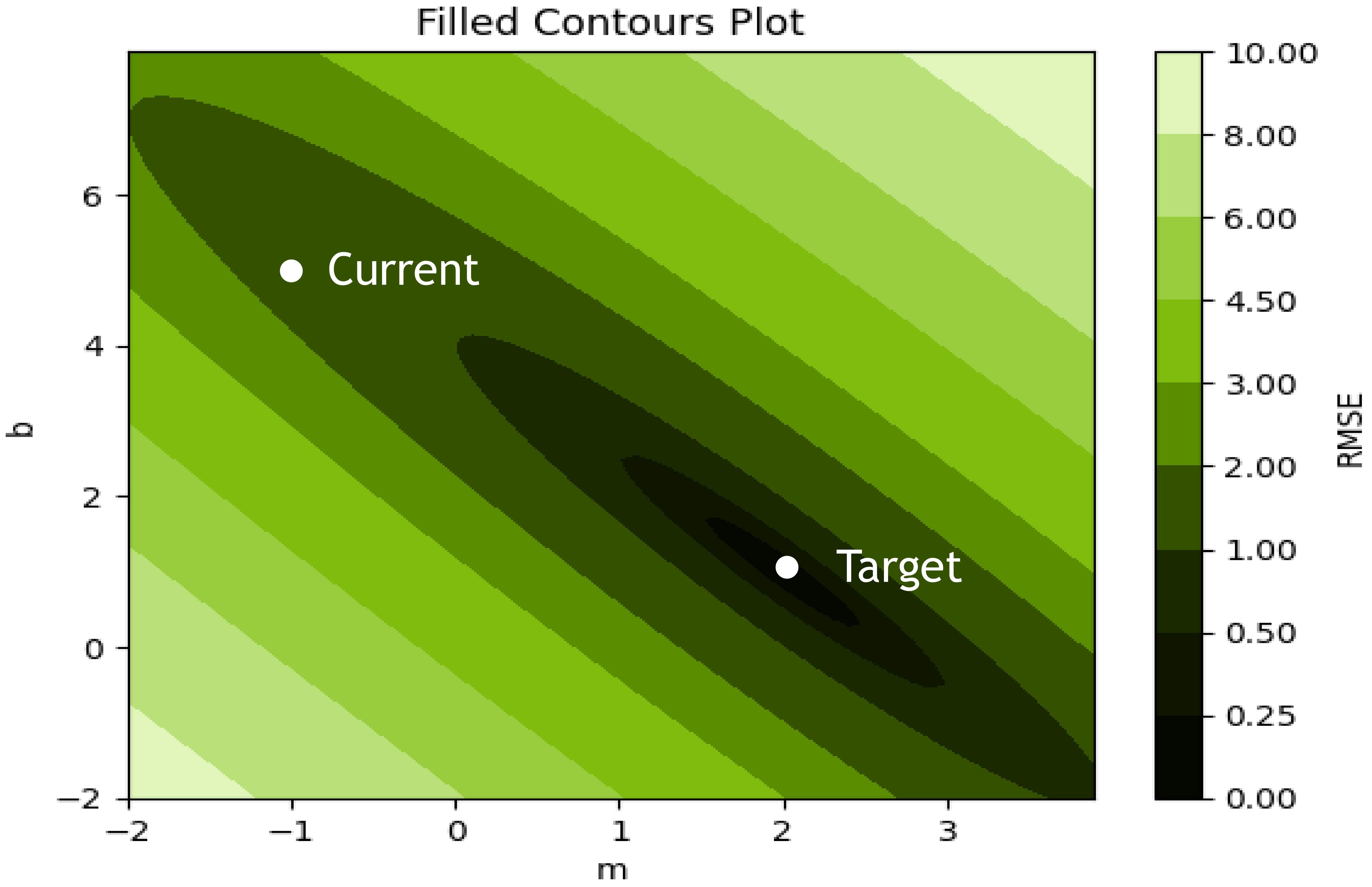


The Loss Curve

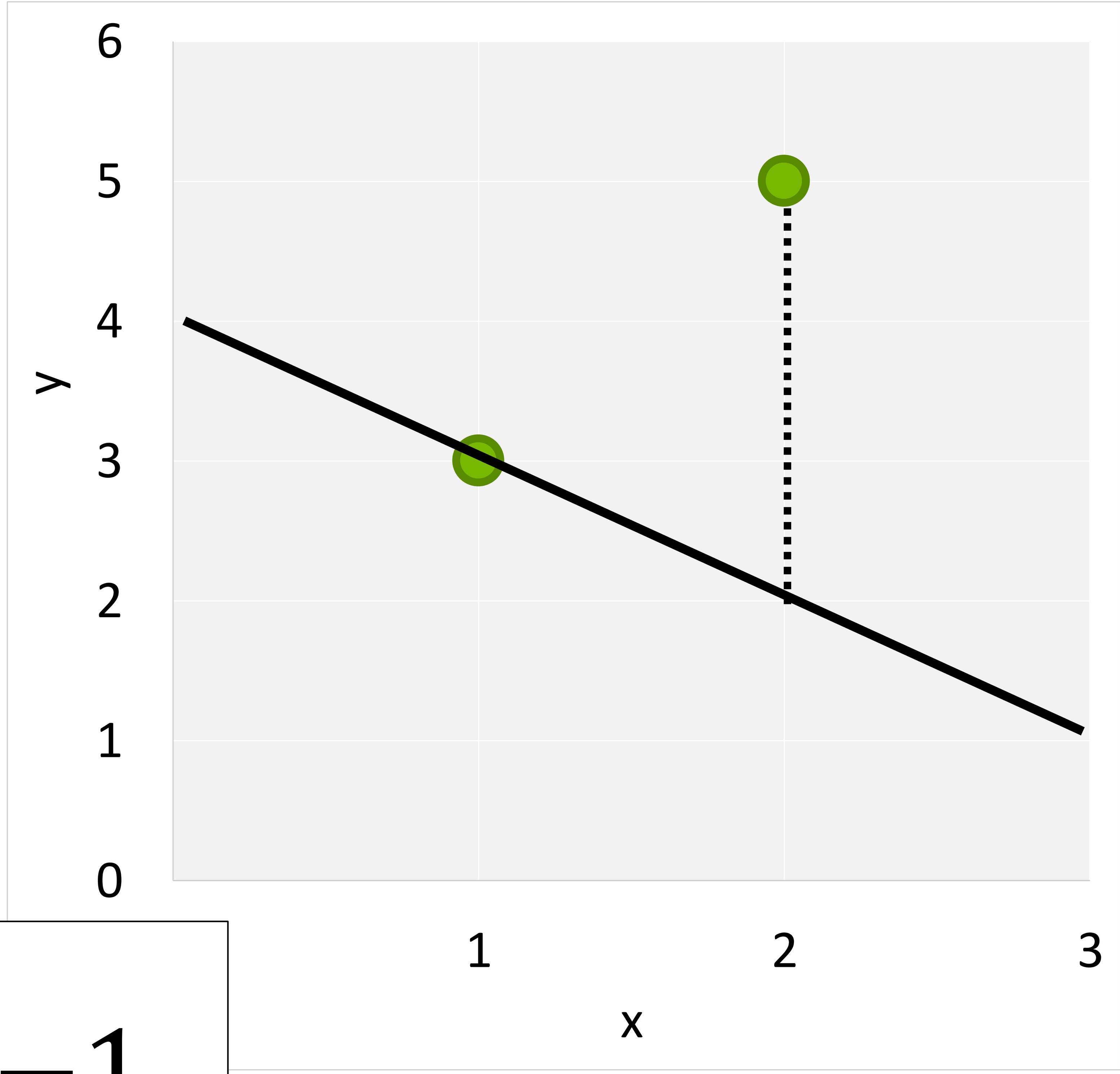


$$m = -1$$

$$b = 5$$

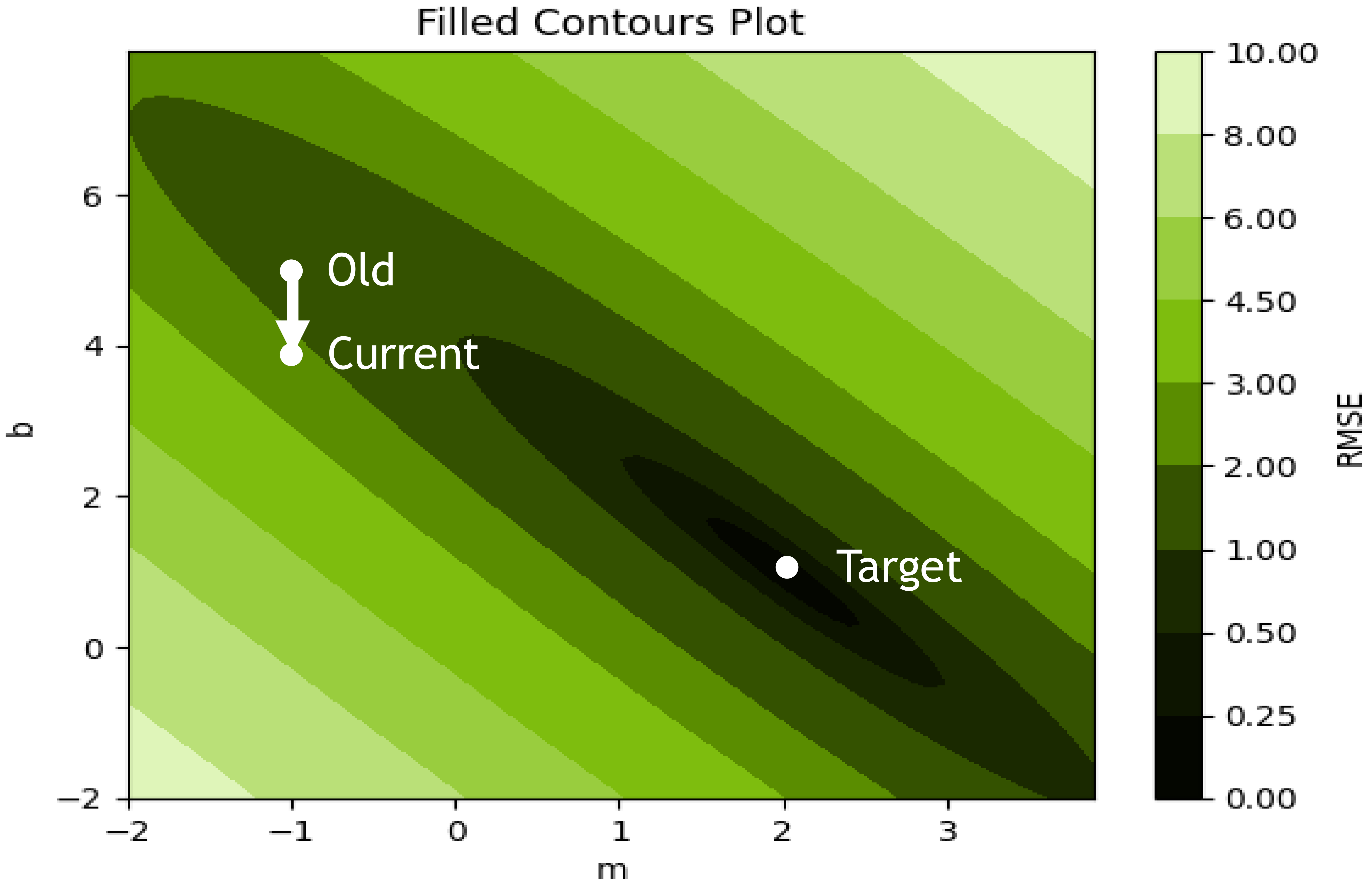


The Loss Curve

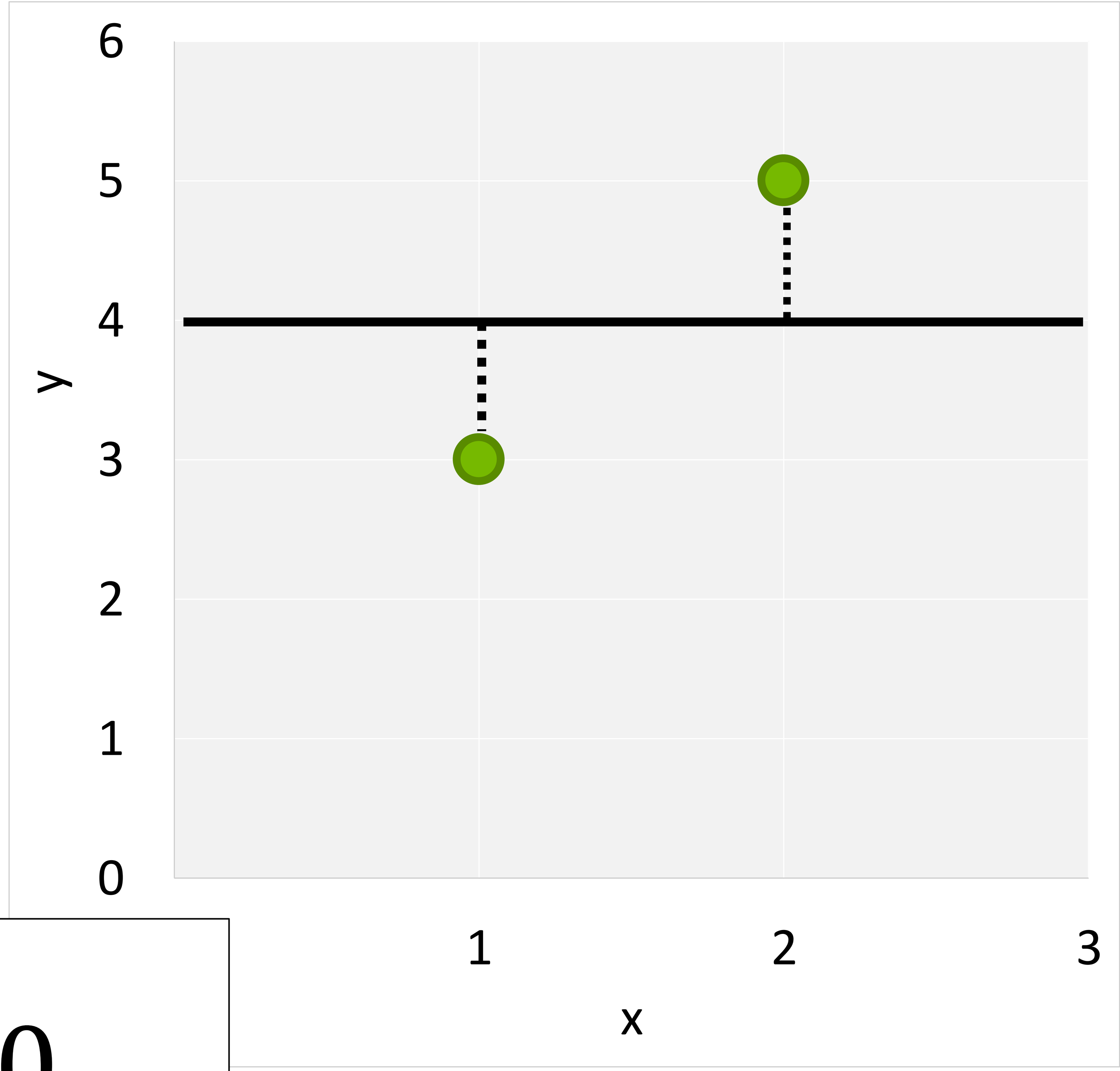


$$m = -1$$

$$b = 4$$

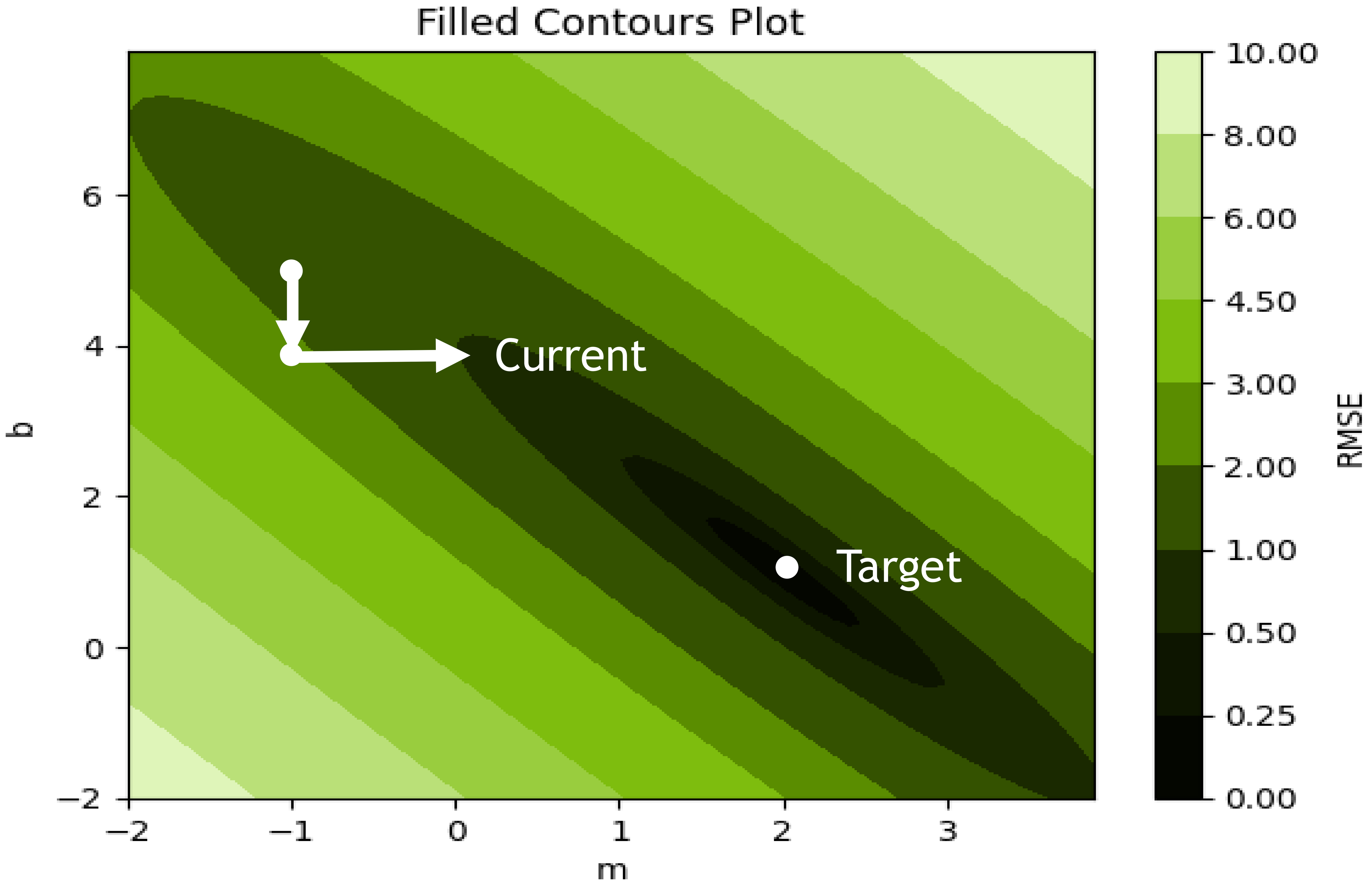


The Loss Curve



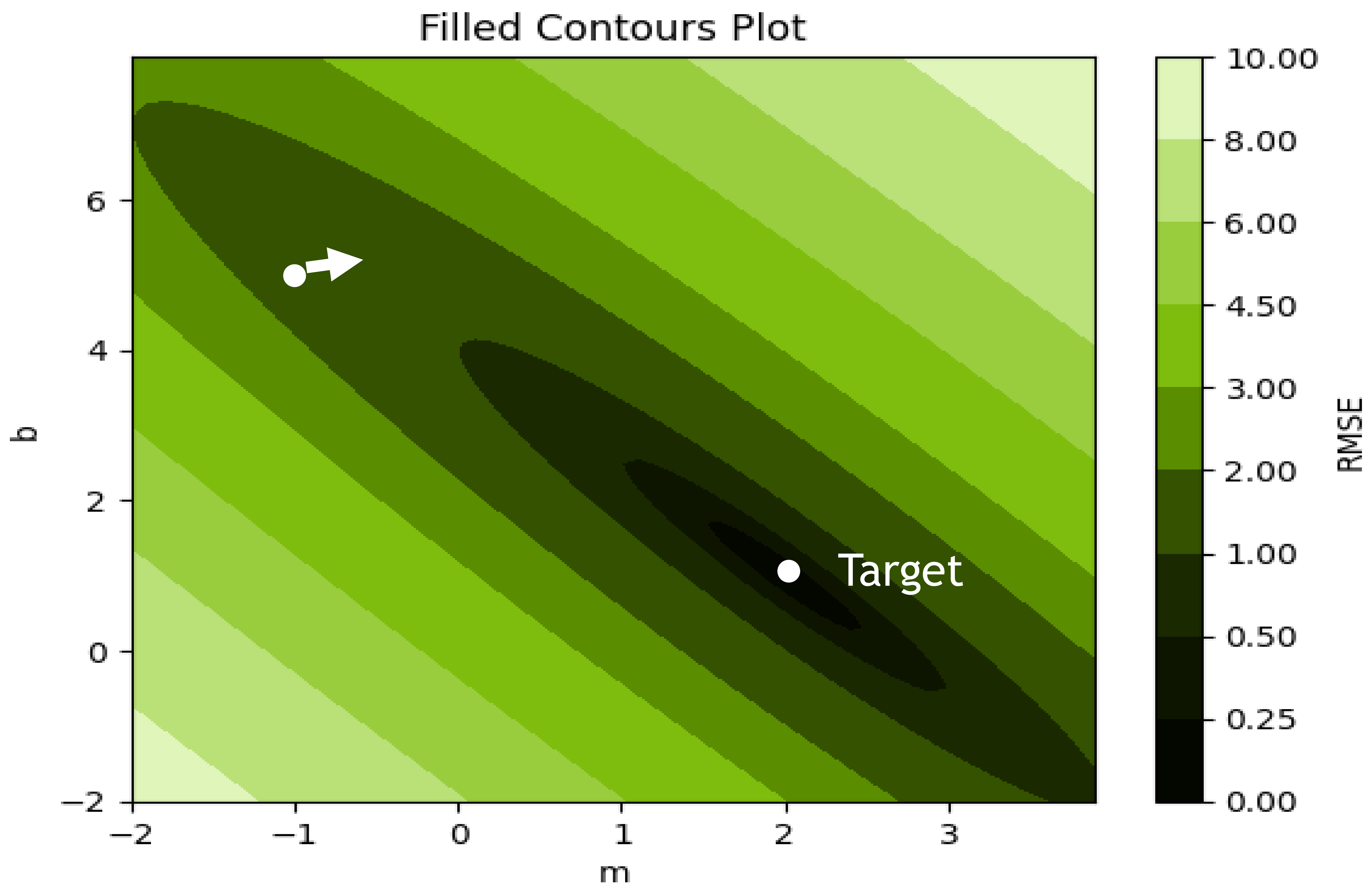
$$m = 0$$

$$\mathrm{b} = 4$$



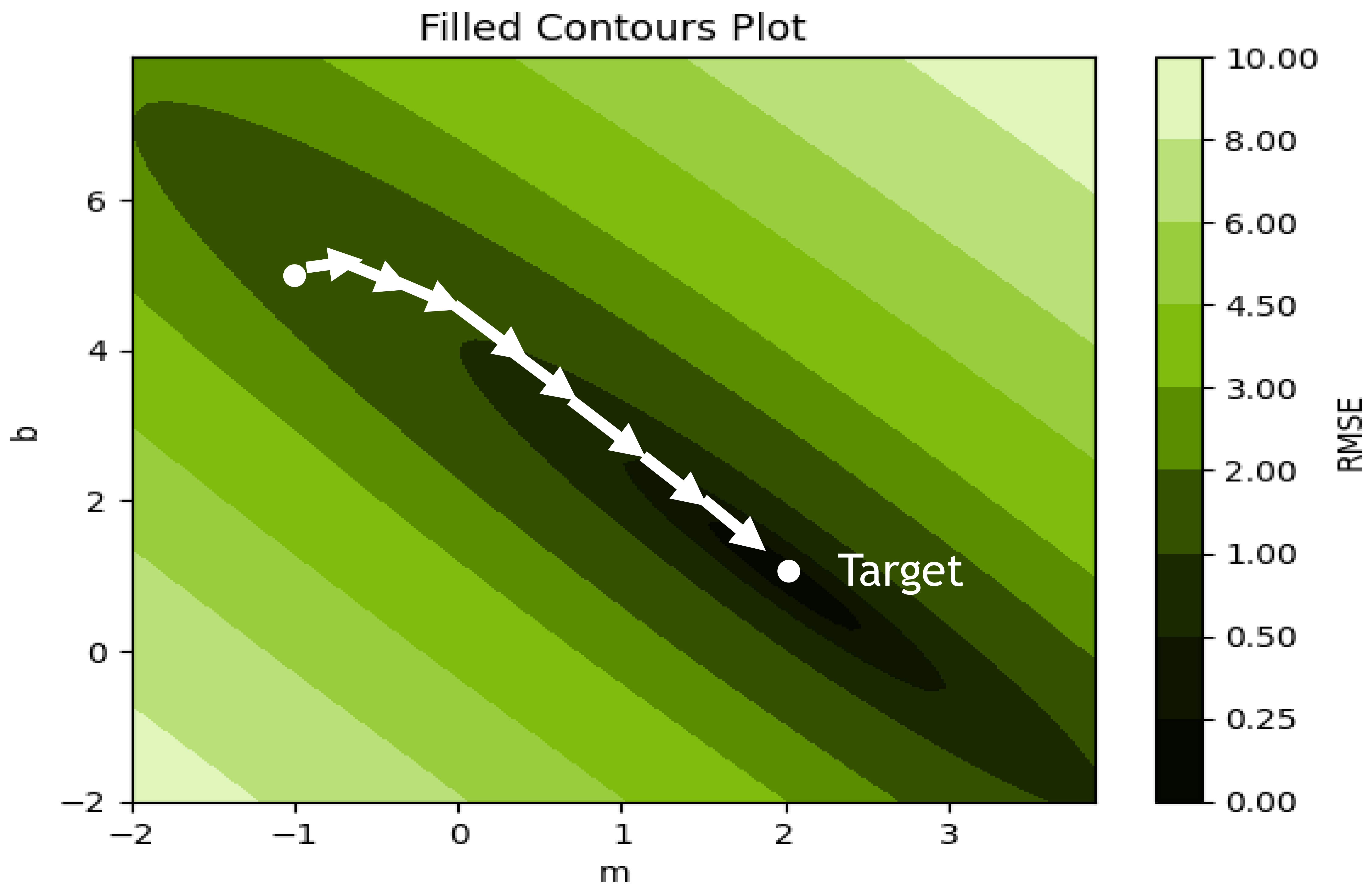
The Loss Curve

The Gradient	Which direction loss decreases the most
λ: The learning rate	How far to travel
Epoch	A model update with the full dataset
Batch	A sample of the full dataset
Step	An update to the weight parameters

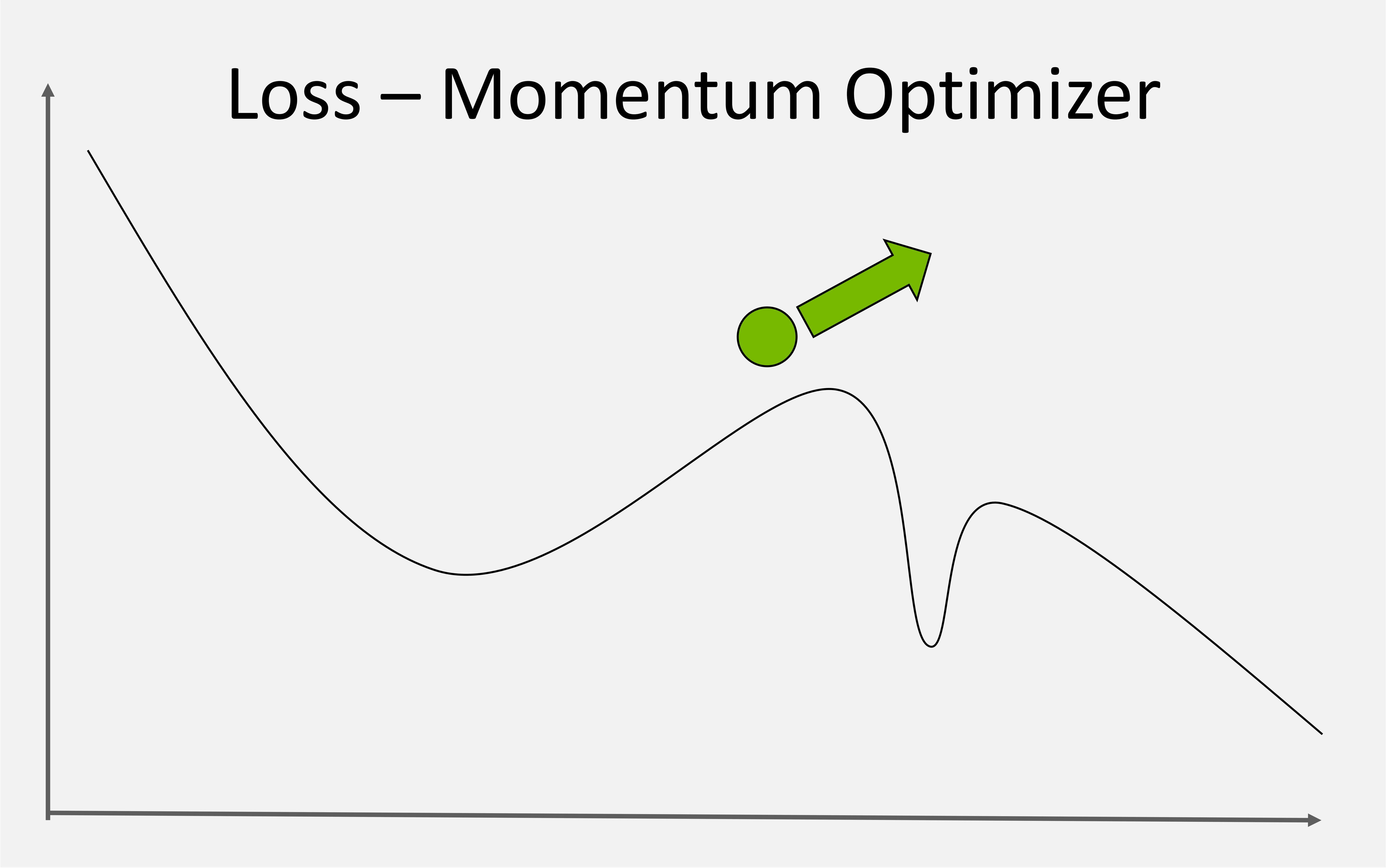


The Loss Curve

The Gradient	Which direction loss decreases the most
λ: The learning rate	How far to travel
Epoch	A model update with the full dataset
Batch	A sample of the full dataset
Step	An update to the weight parameters



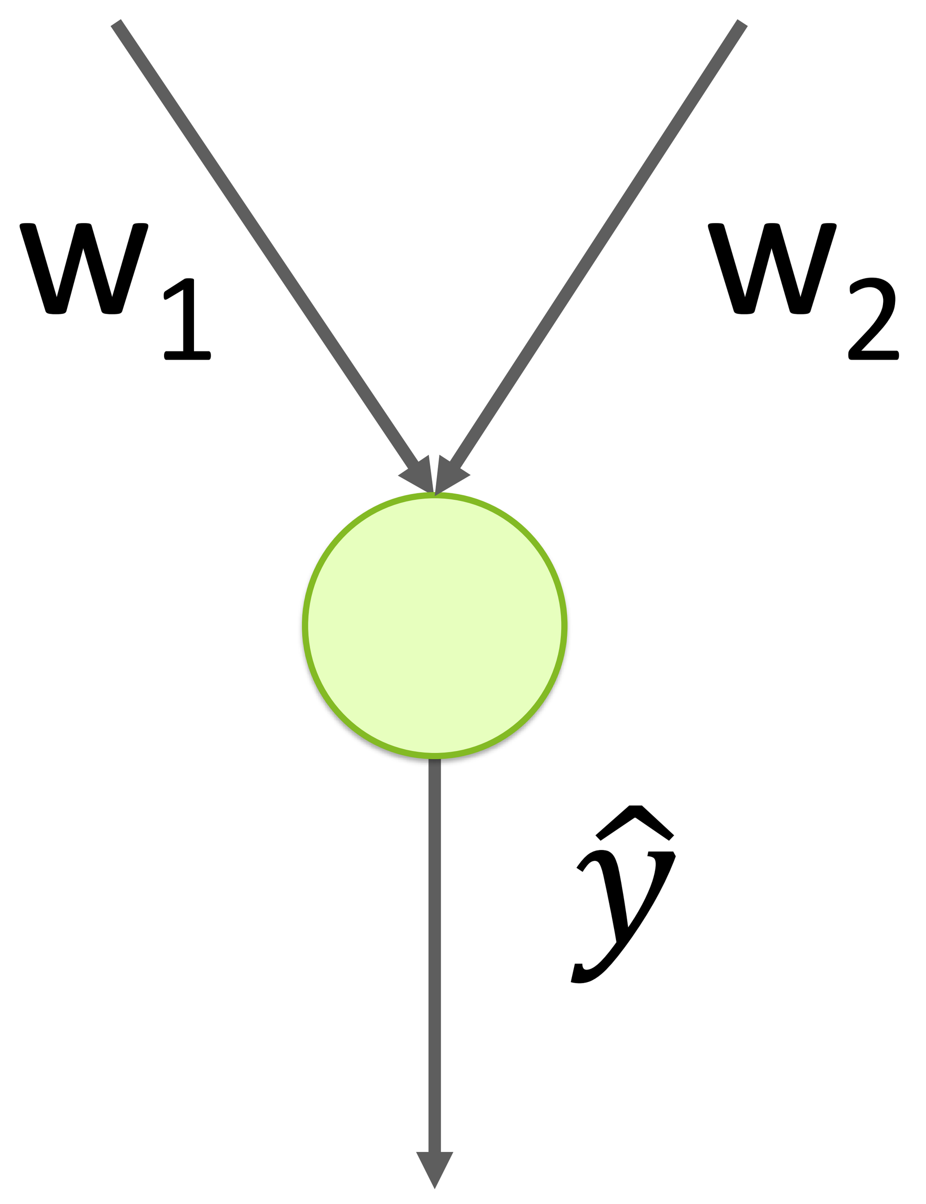
Optimizers



- Adam
- Adagrad
- RMSprop
- SGD

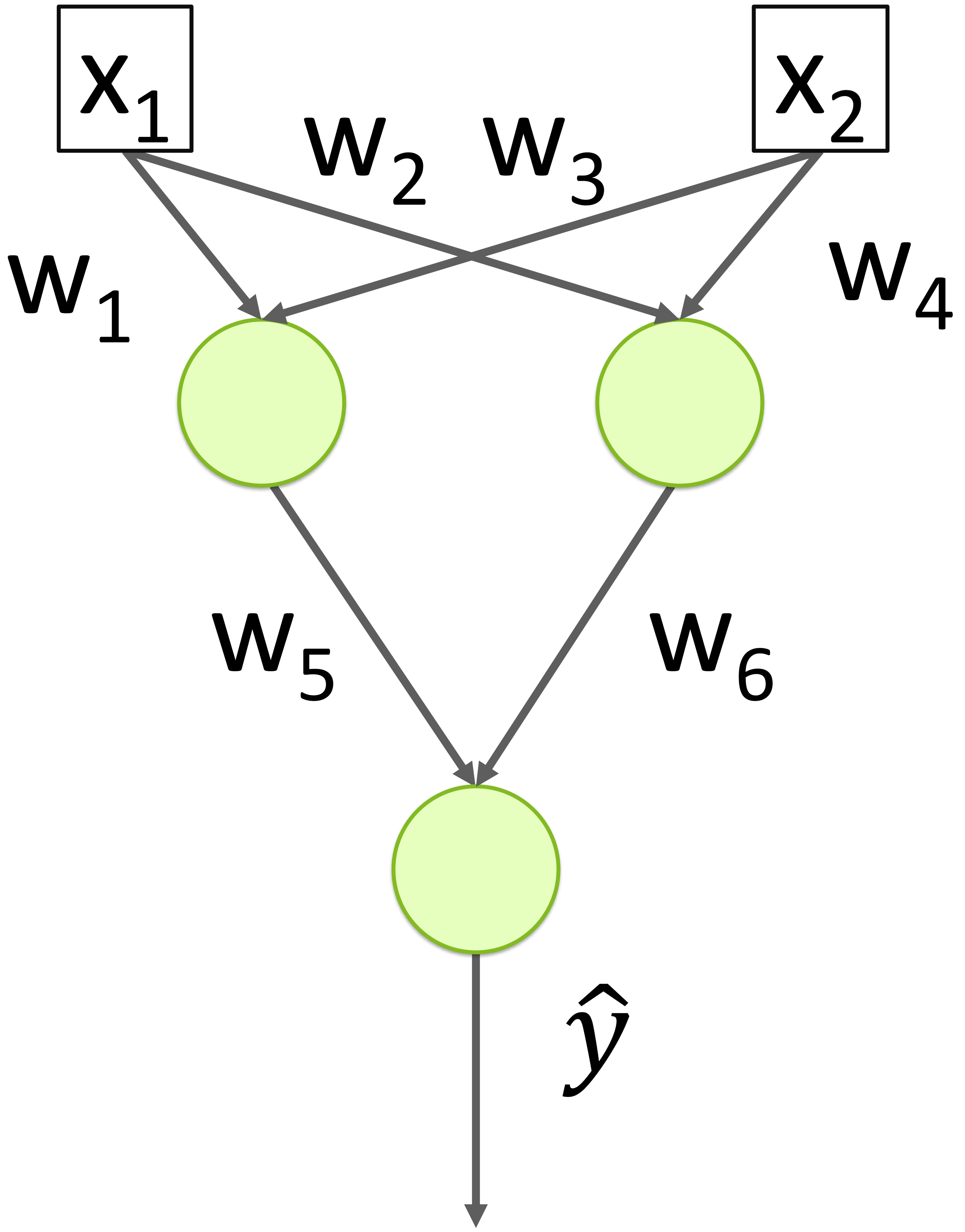
From Neuron to Network

Building a Network



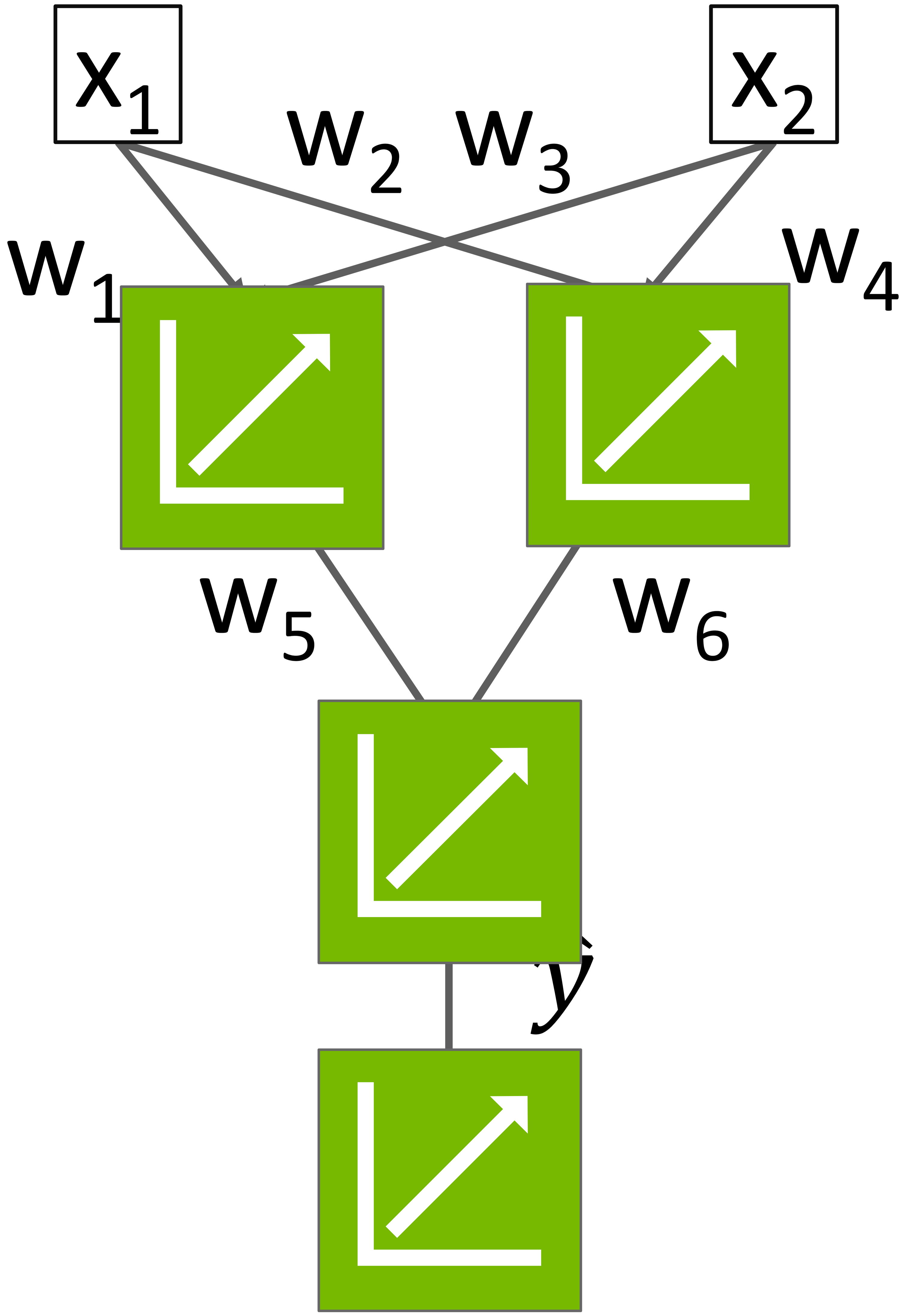
- Scales to more inputs

Building a Network



- Scales to more inputs
- Can chain neurons

Building a Network



- Scales to more inputs
- Can chain neurons
- If all regressions are linear, then output will also be a linear regression

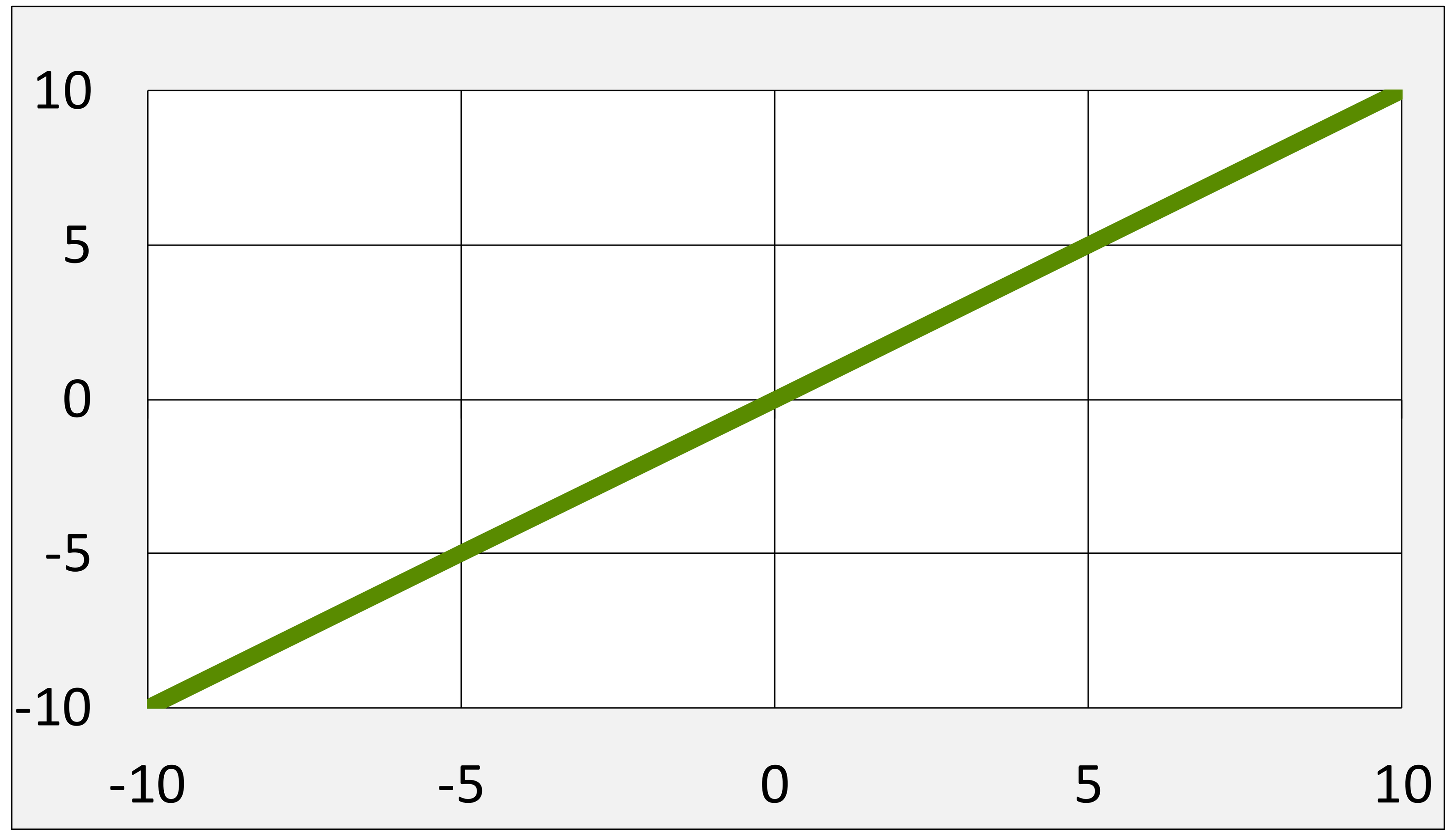
Activation Functions

Activation Functions

Linear

$$\hat{y} = wx + b$$

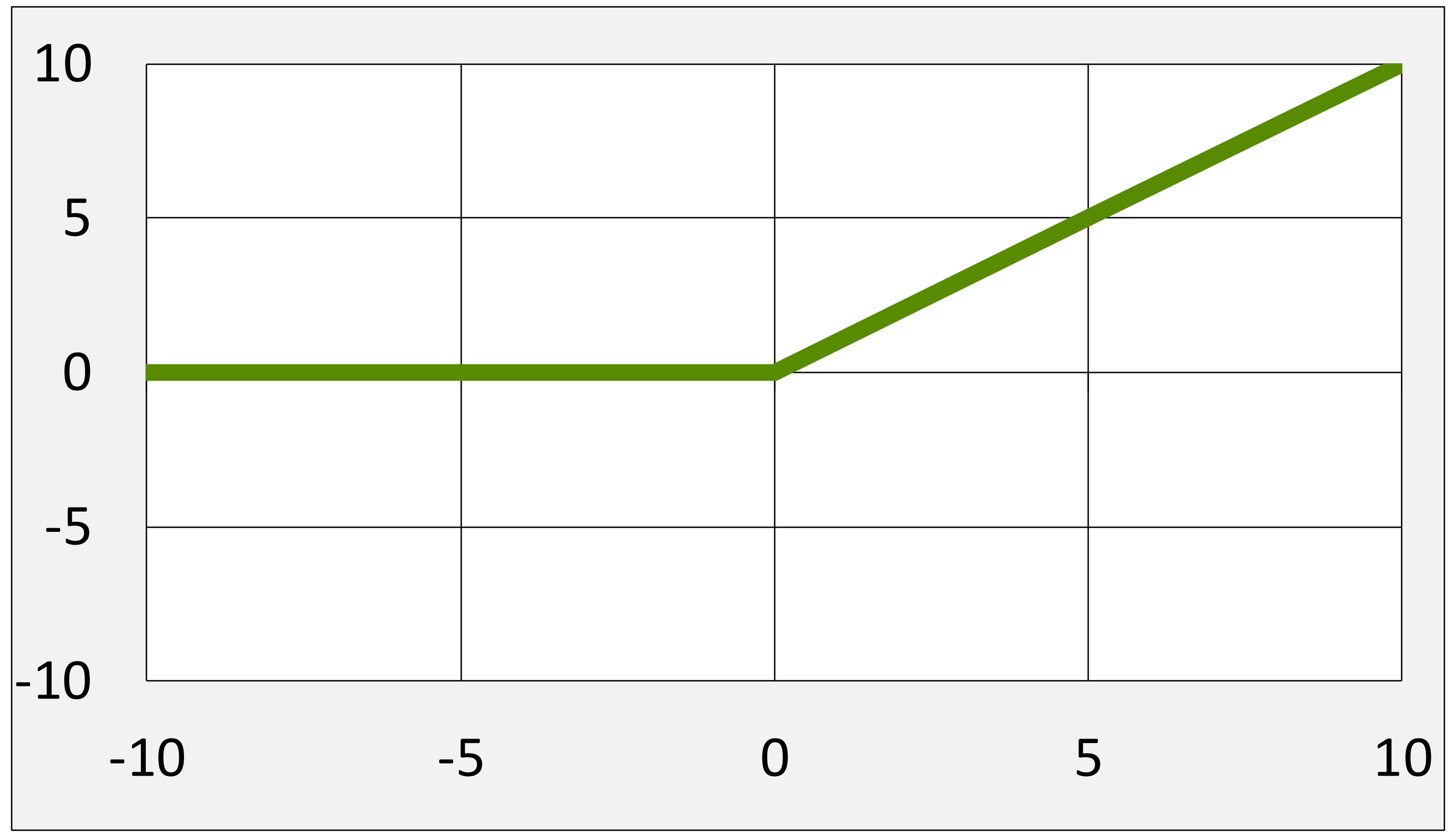
```python
# Multiply each input
# with a weight (w) and
# add intercept (b)
y_hat = wx+b
```



ReLU

$$\hat{y} = \begin{cases} wx + b \; if \; wx + b > 0 \\ 0 \;\; otherwise \end{cases}$$

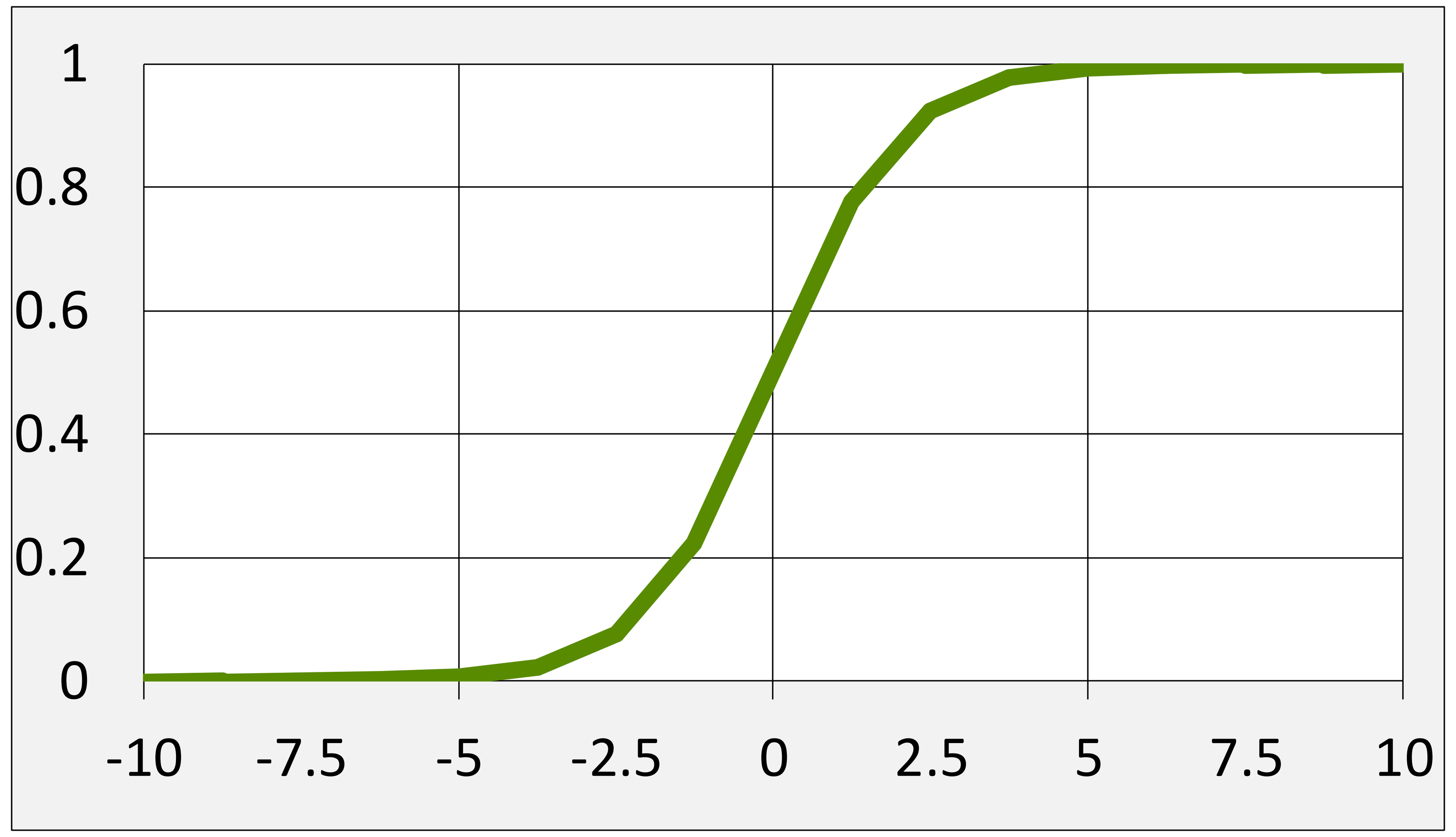
```python
# Only return result
# if total is positive
linear = wx+b
y_hat = linear * (linear > 0)
```



Sigmoid

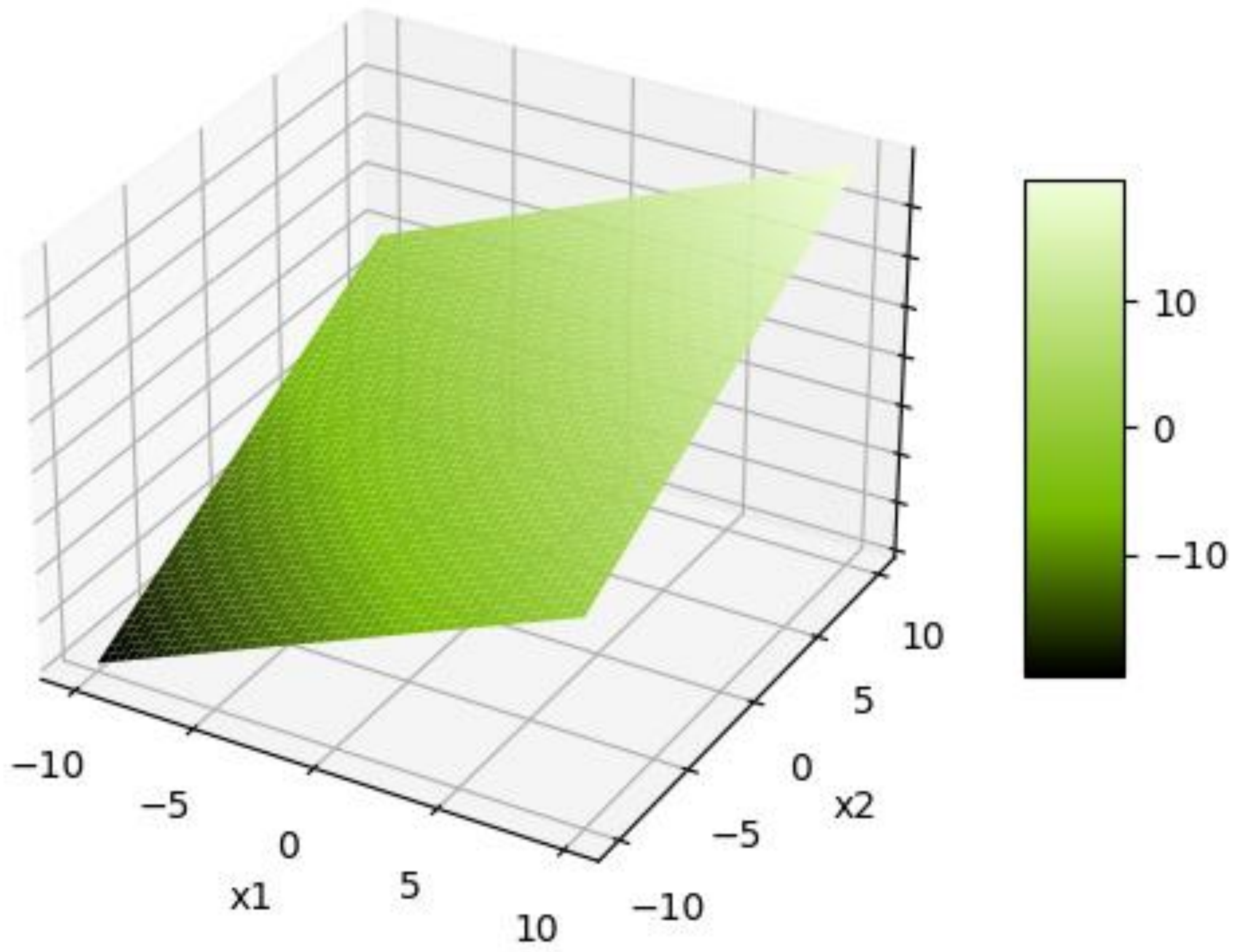
$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

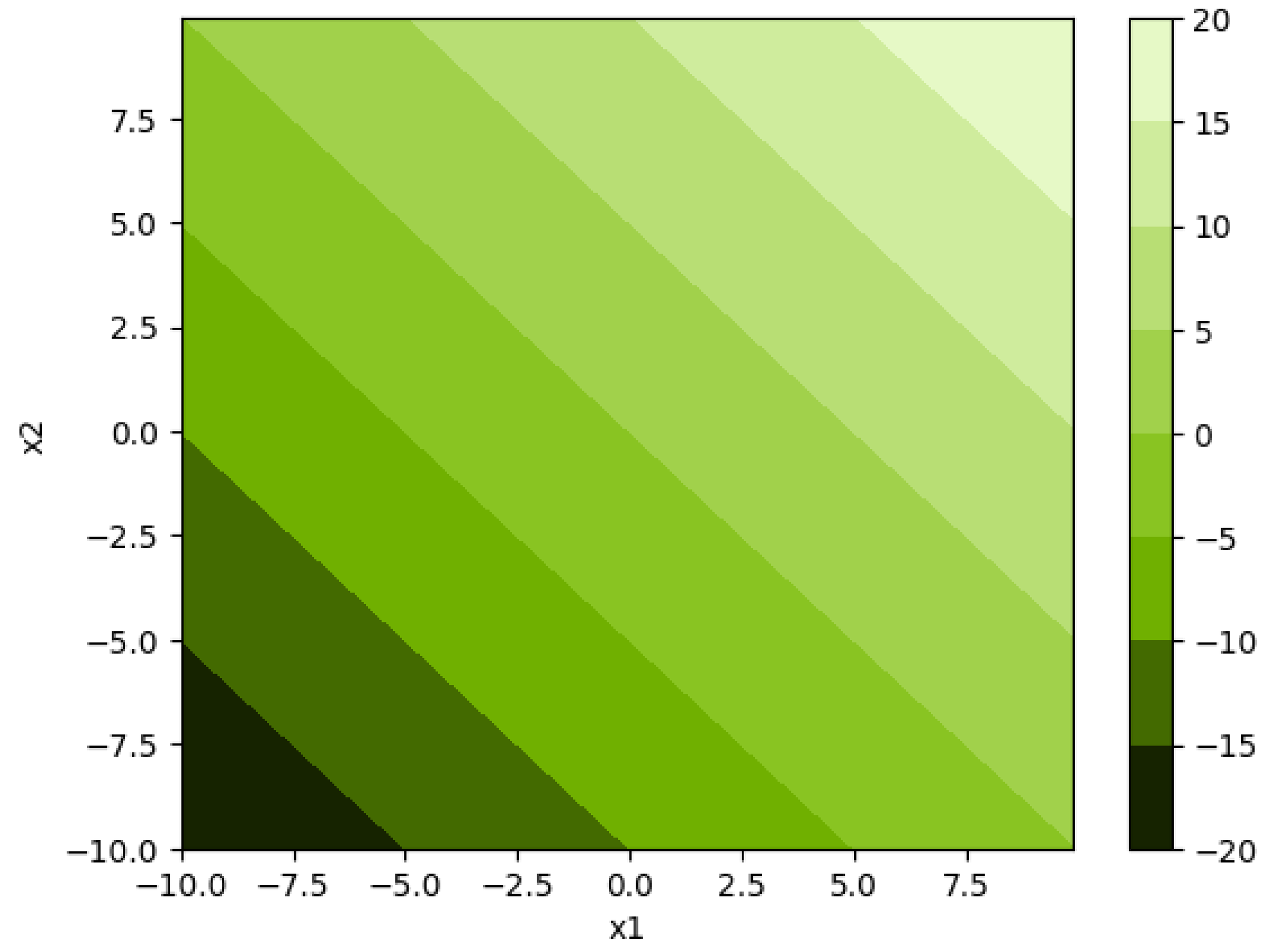
```python
# Start with line
linear = wx + b
# Warp to - inf to 0
inf_to_zero = np.exp(-1 * linear)
# Squish to -1 to 1
y_hat = 1 / (1 + inf_to_zero)
```



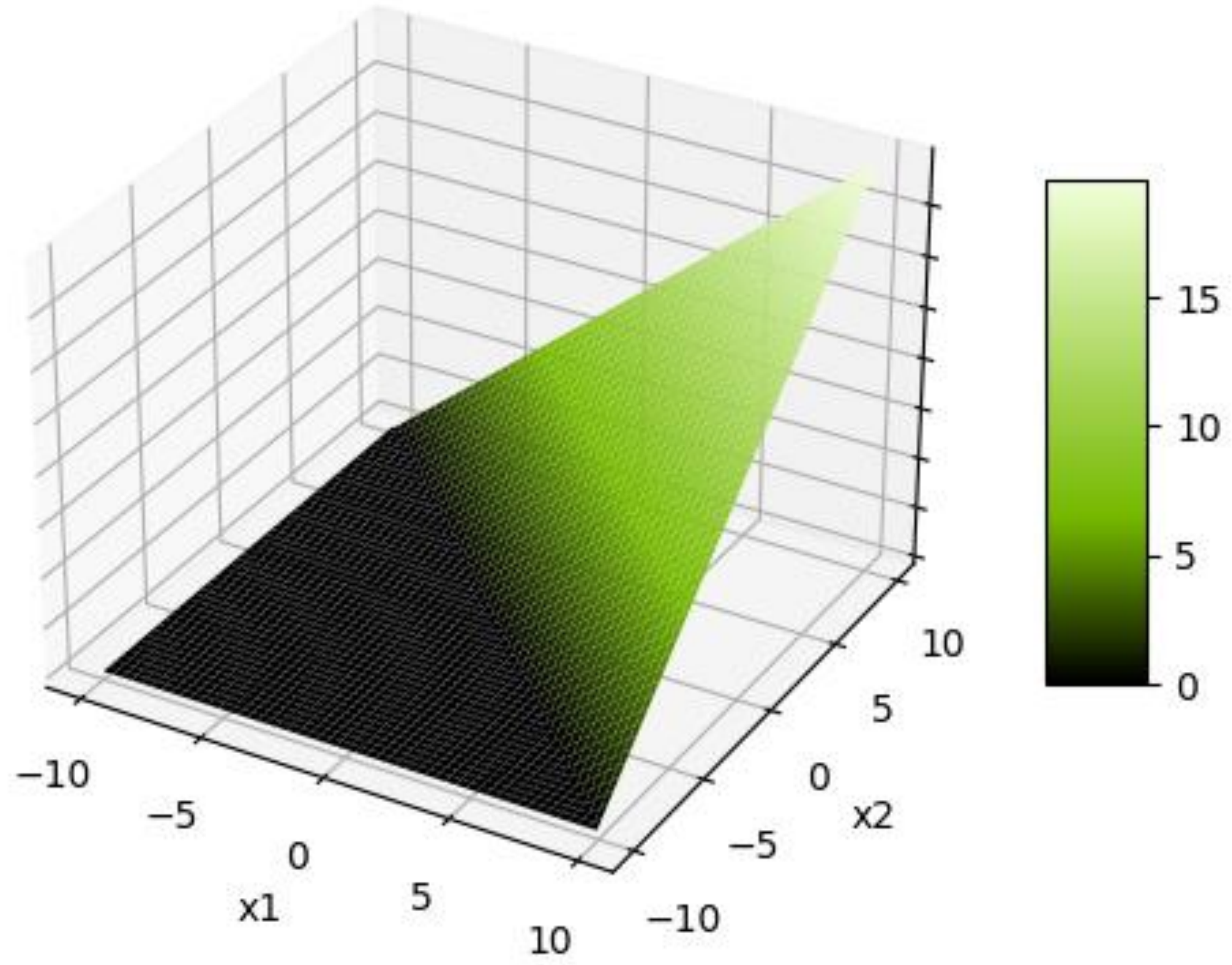
Activation Functions

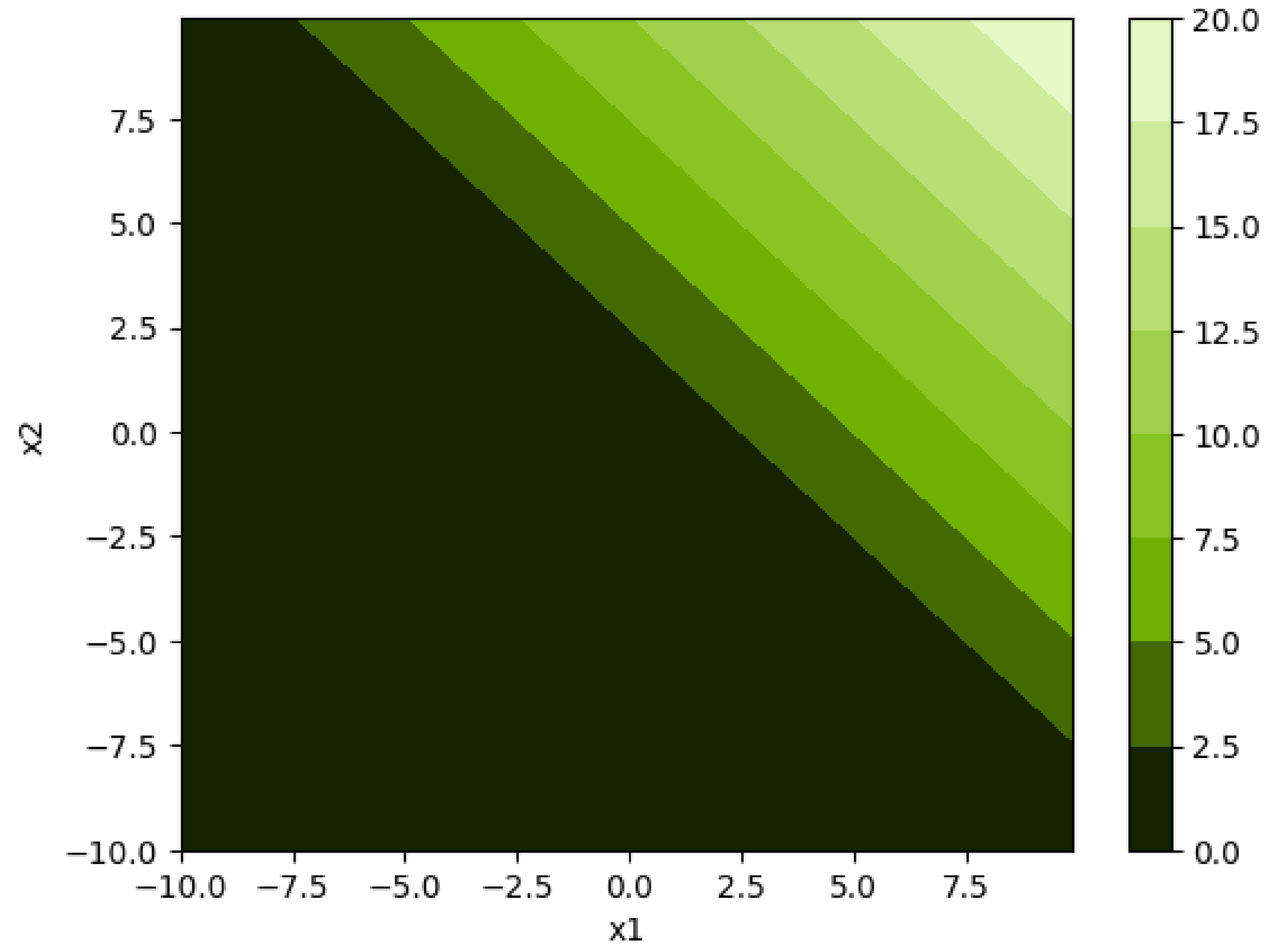
Linear



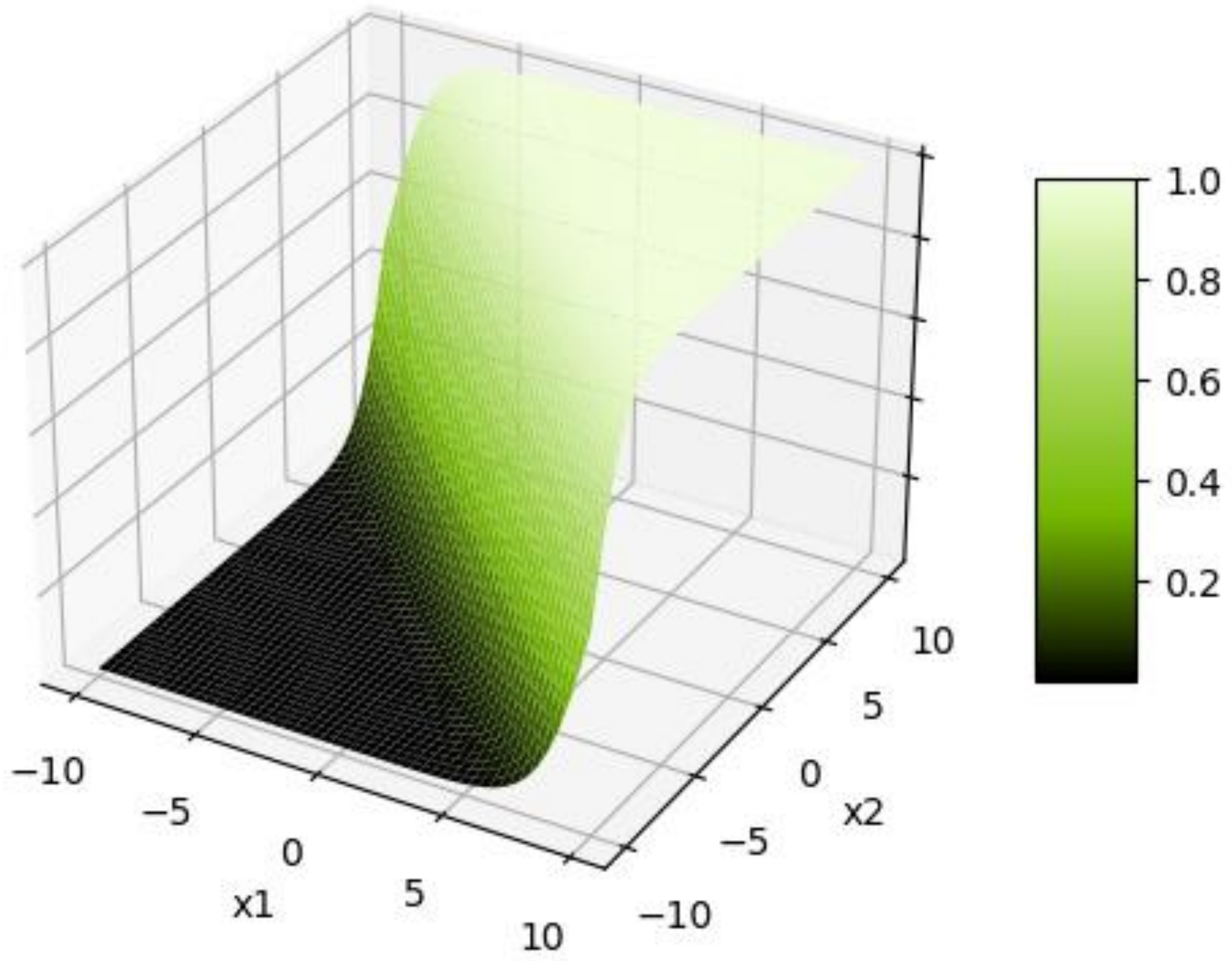


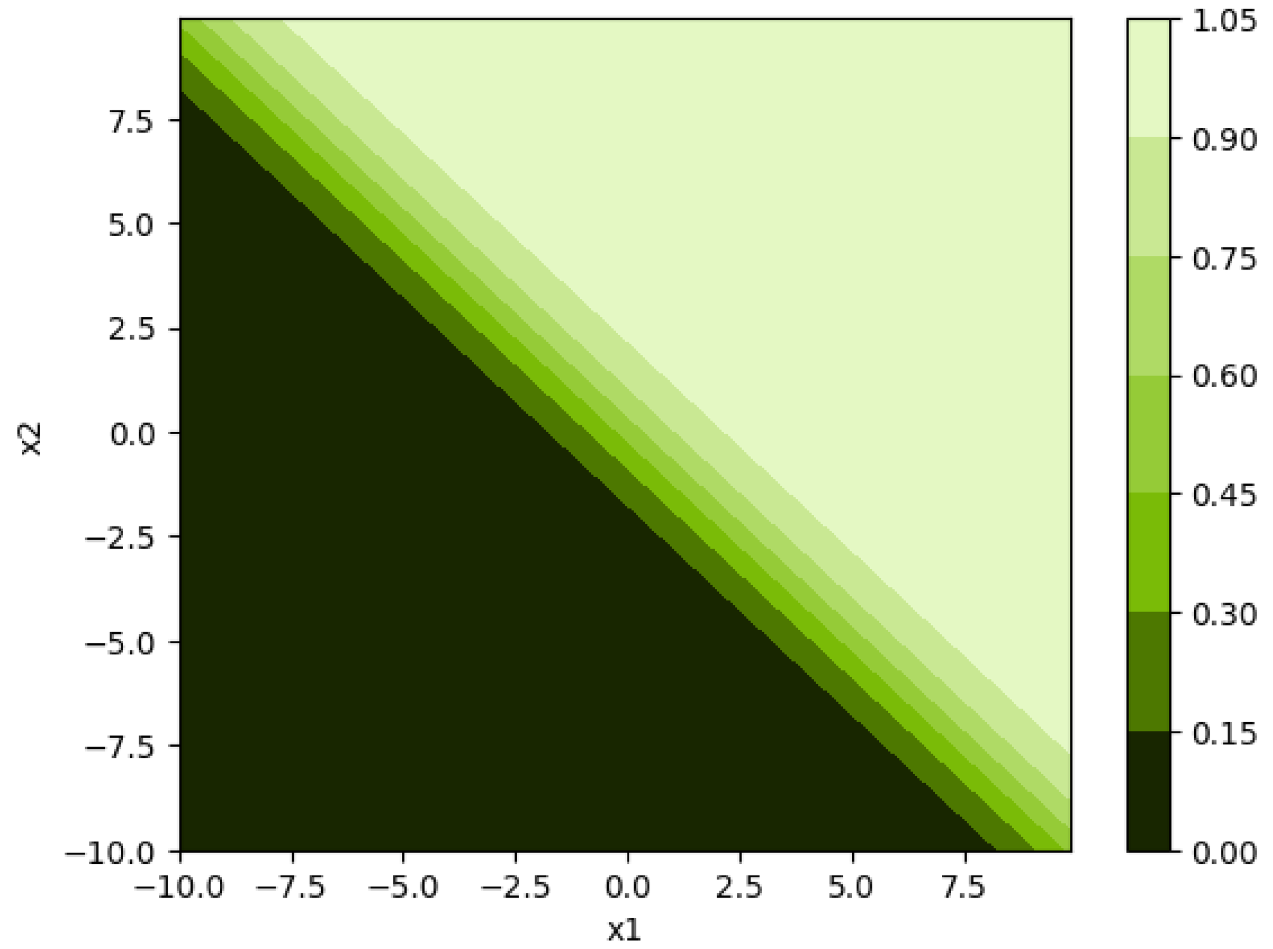
ReLU



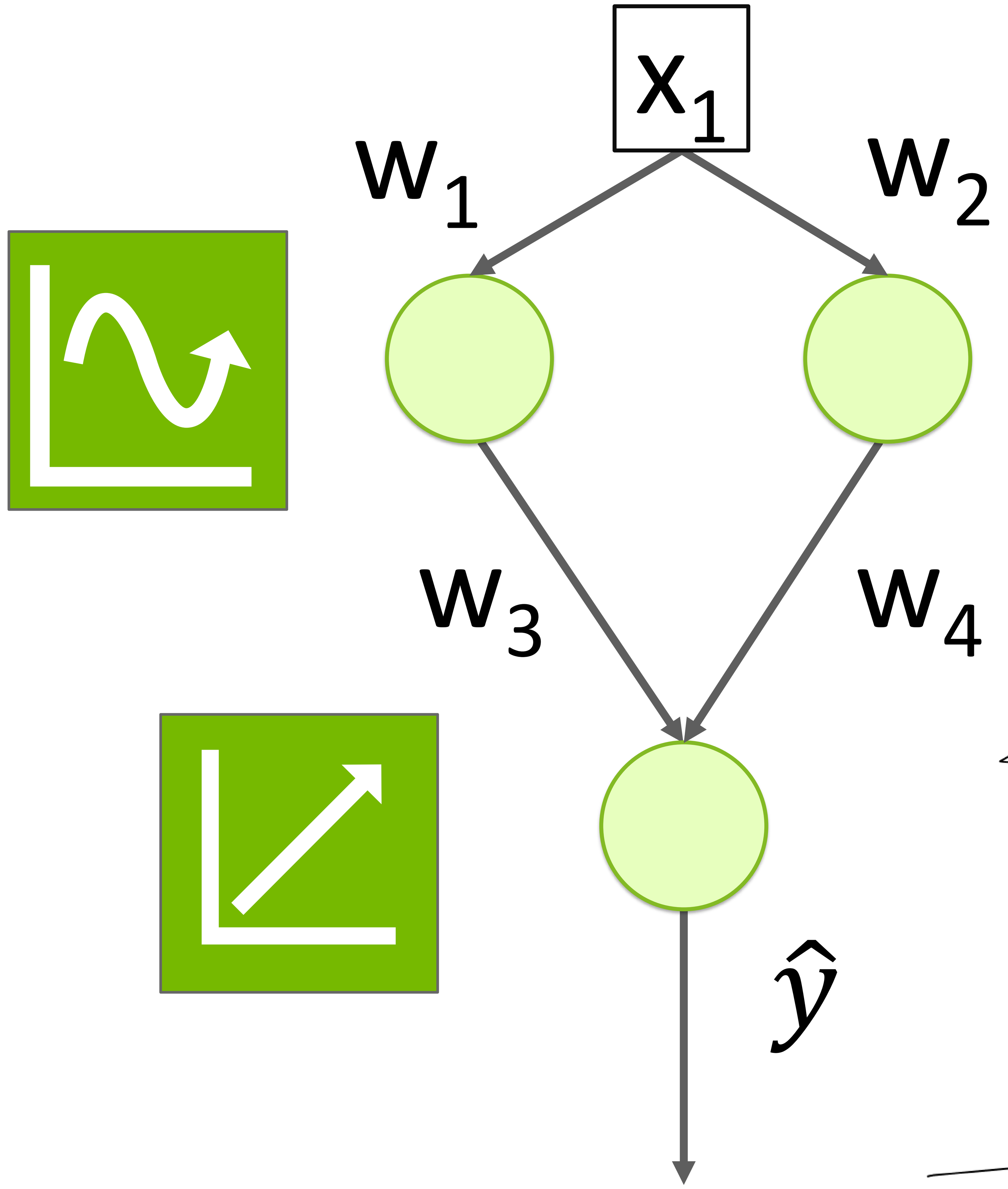


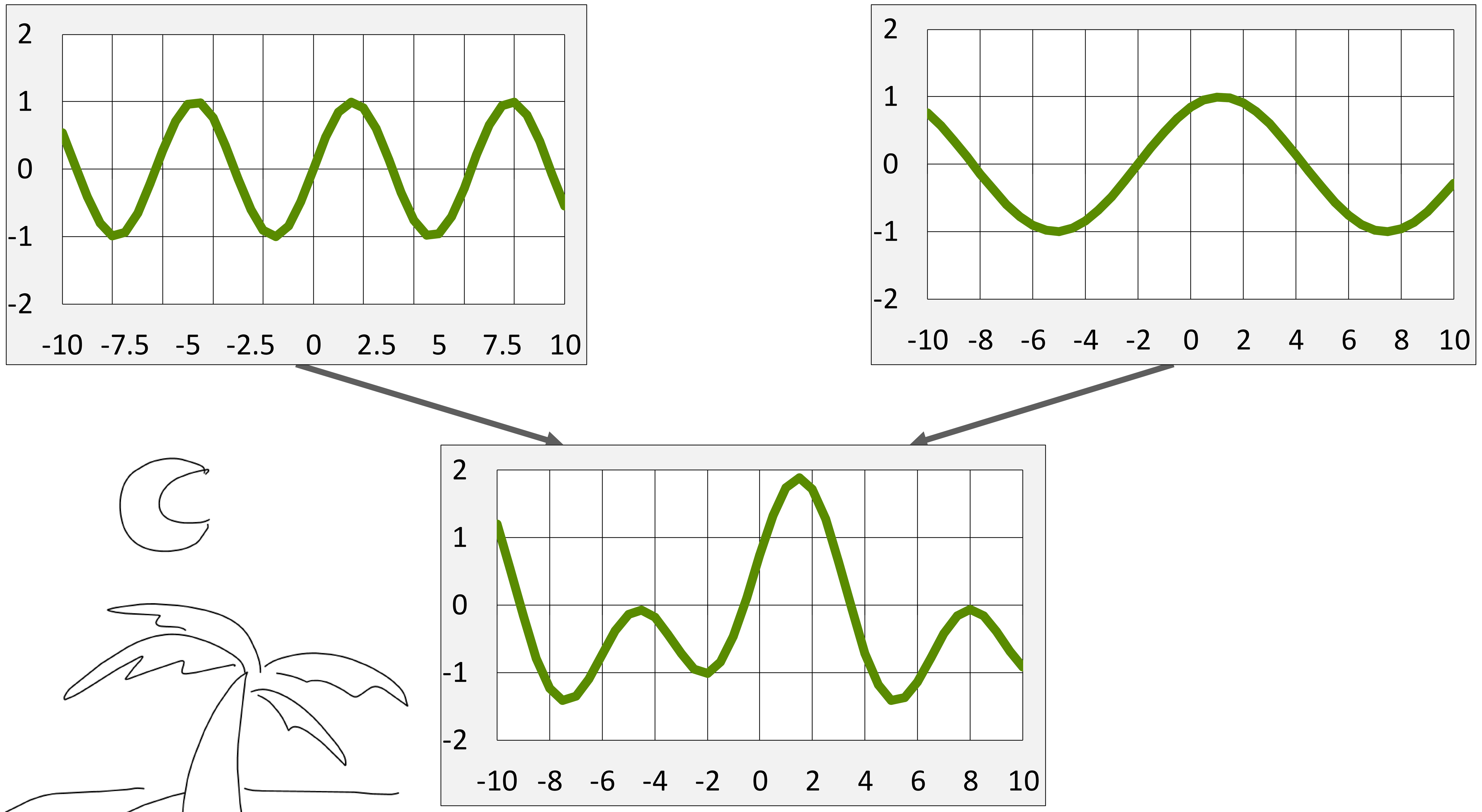
Sigmoid





Activation Functions





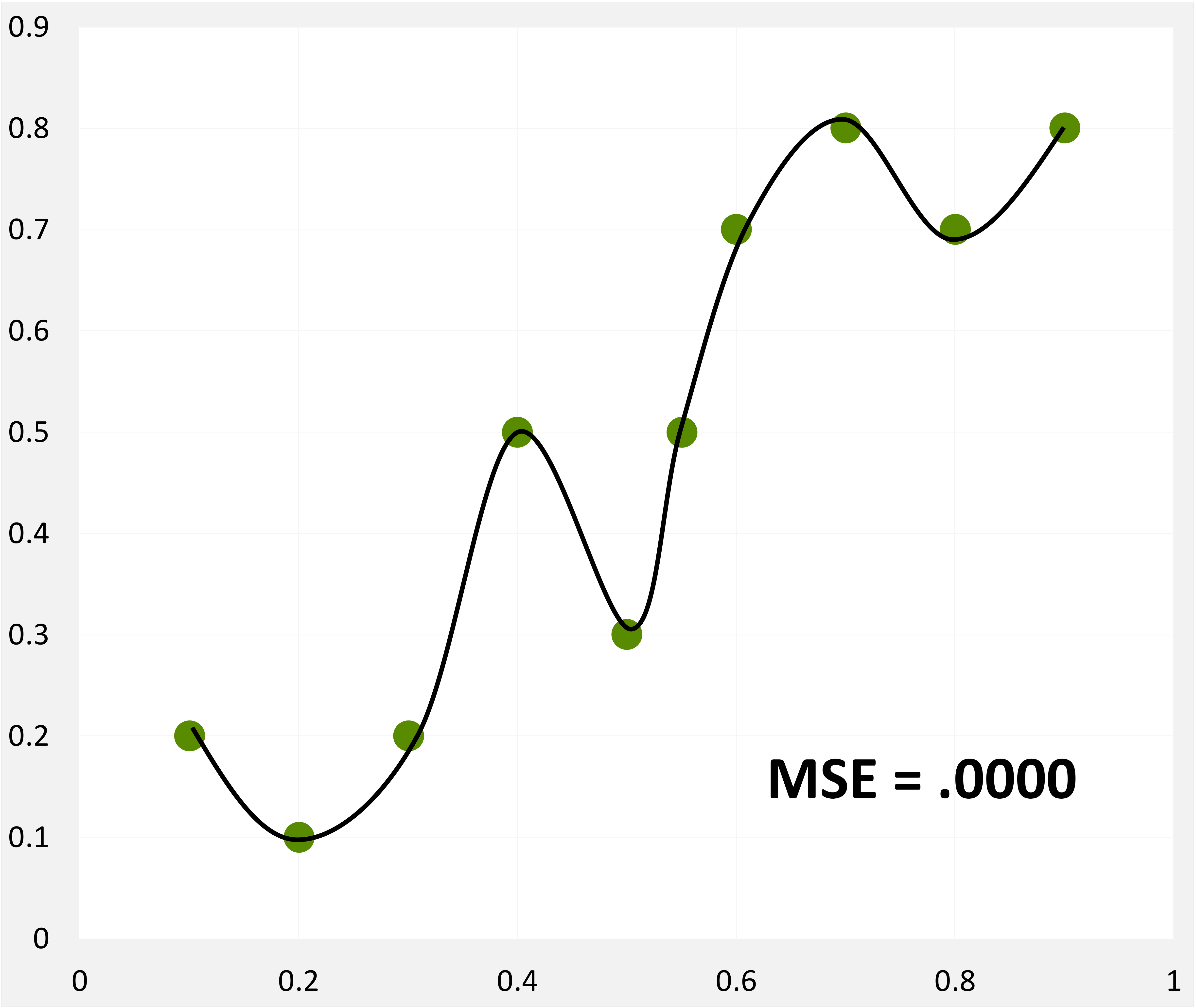
Overfitting

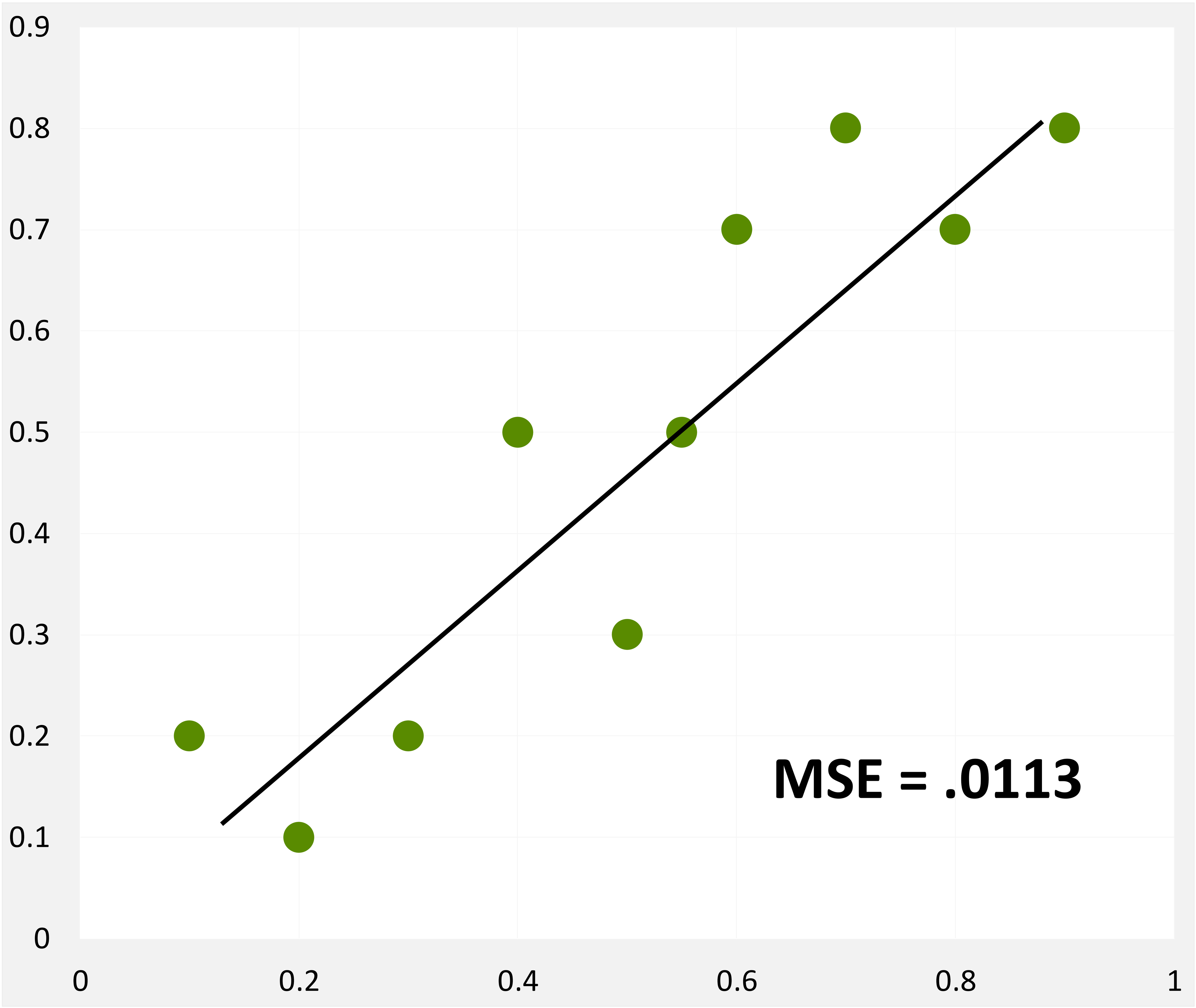
Overfitting

Why not have a super large neural network?

Overfitting

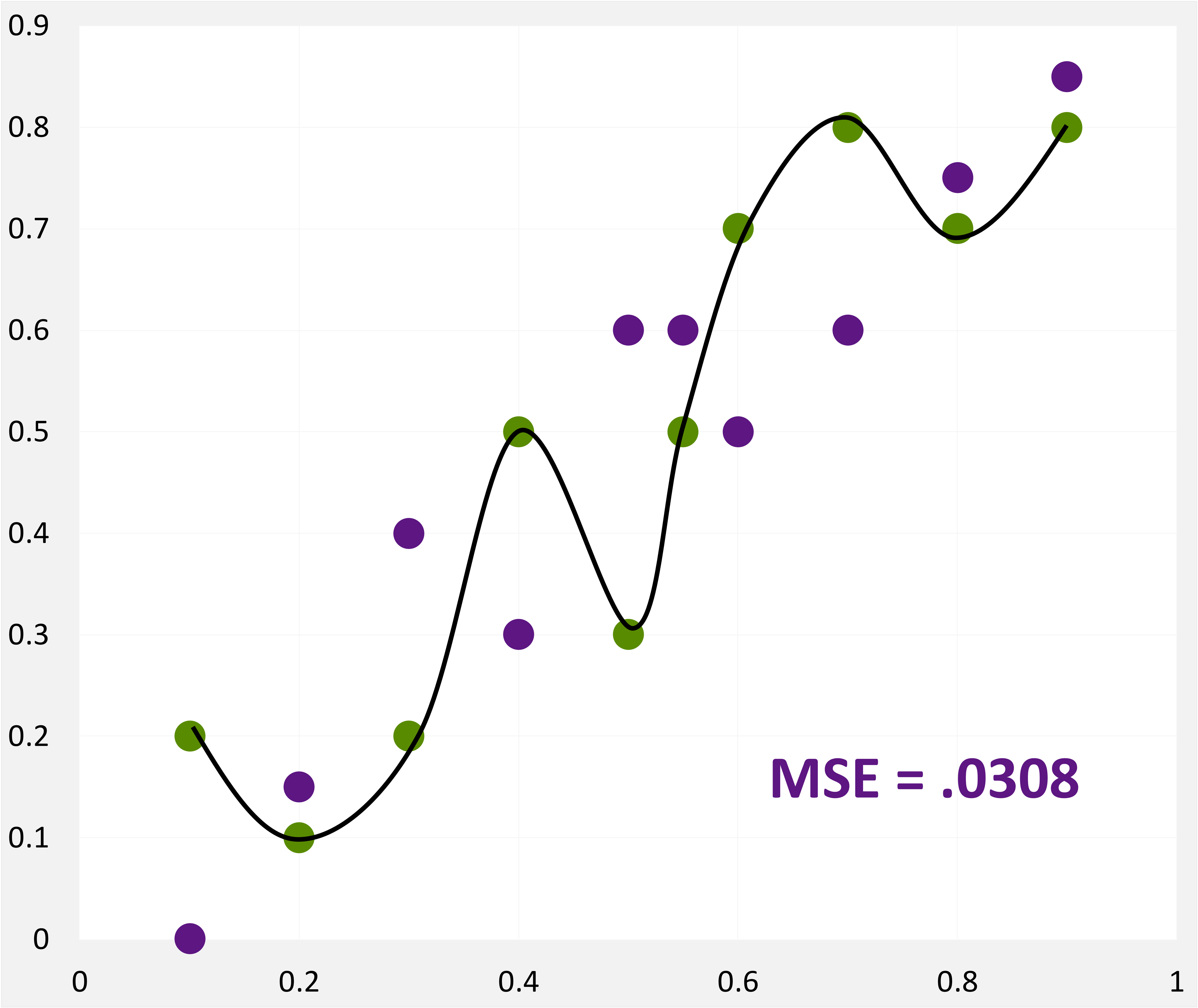
Which Trendline is Better?

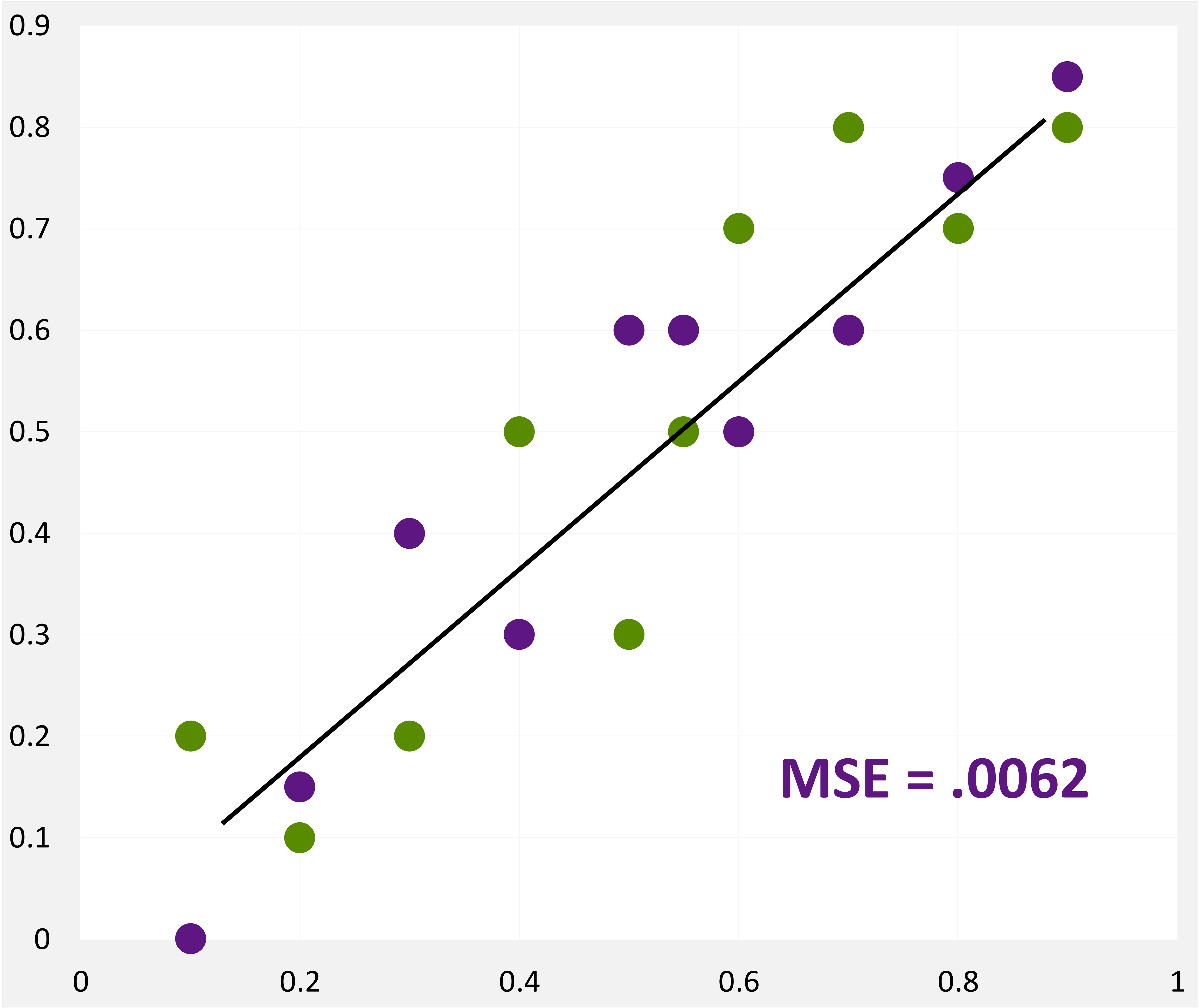




Overfitting

Which Trendline is Better?





Training vs Validation Data

Avoid memorization

Training data

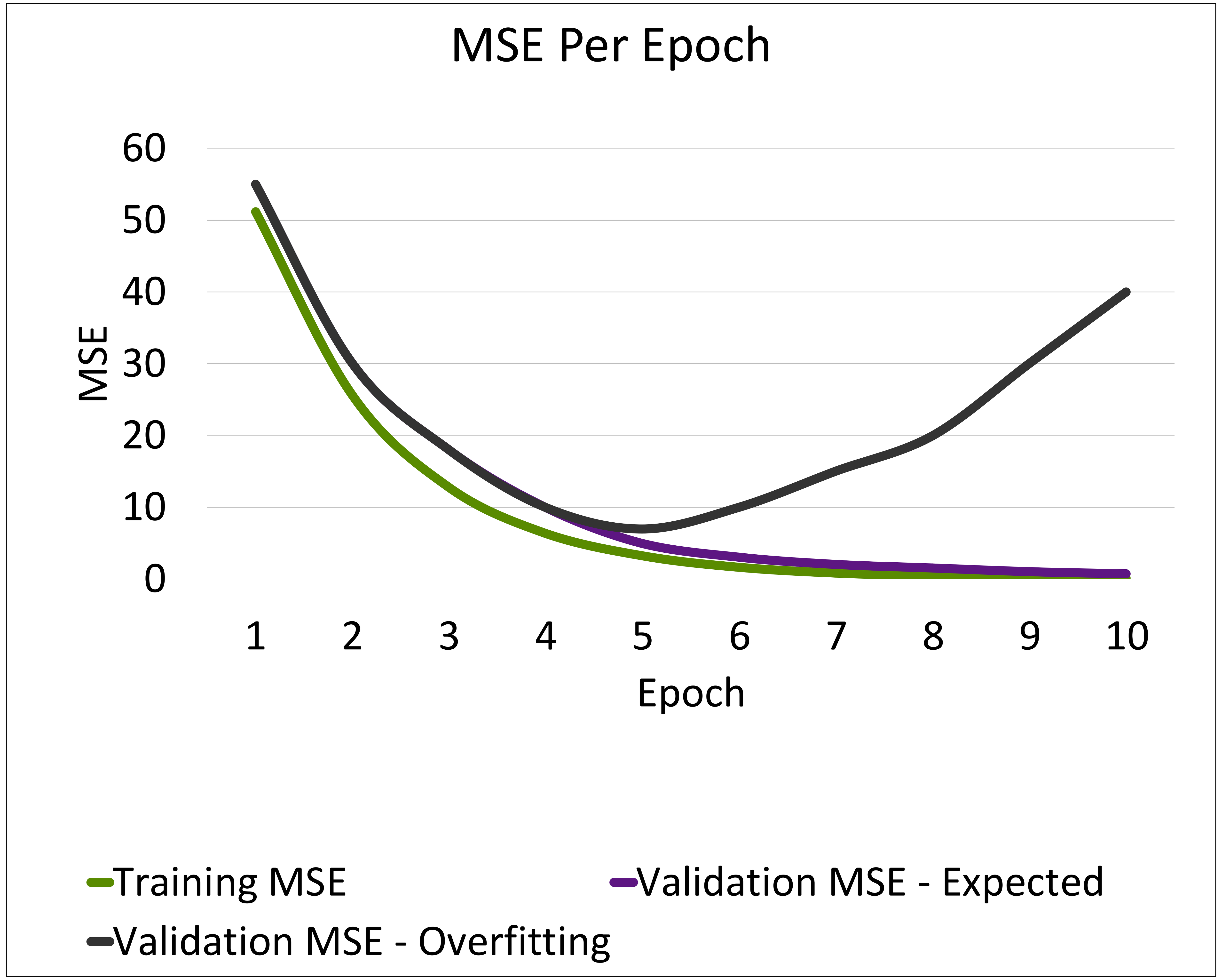
- Core dataset for the model to learn on

Validation data

- New data for model to see if it truly understands (can generalize)

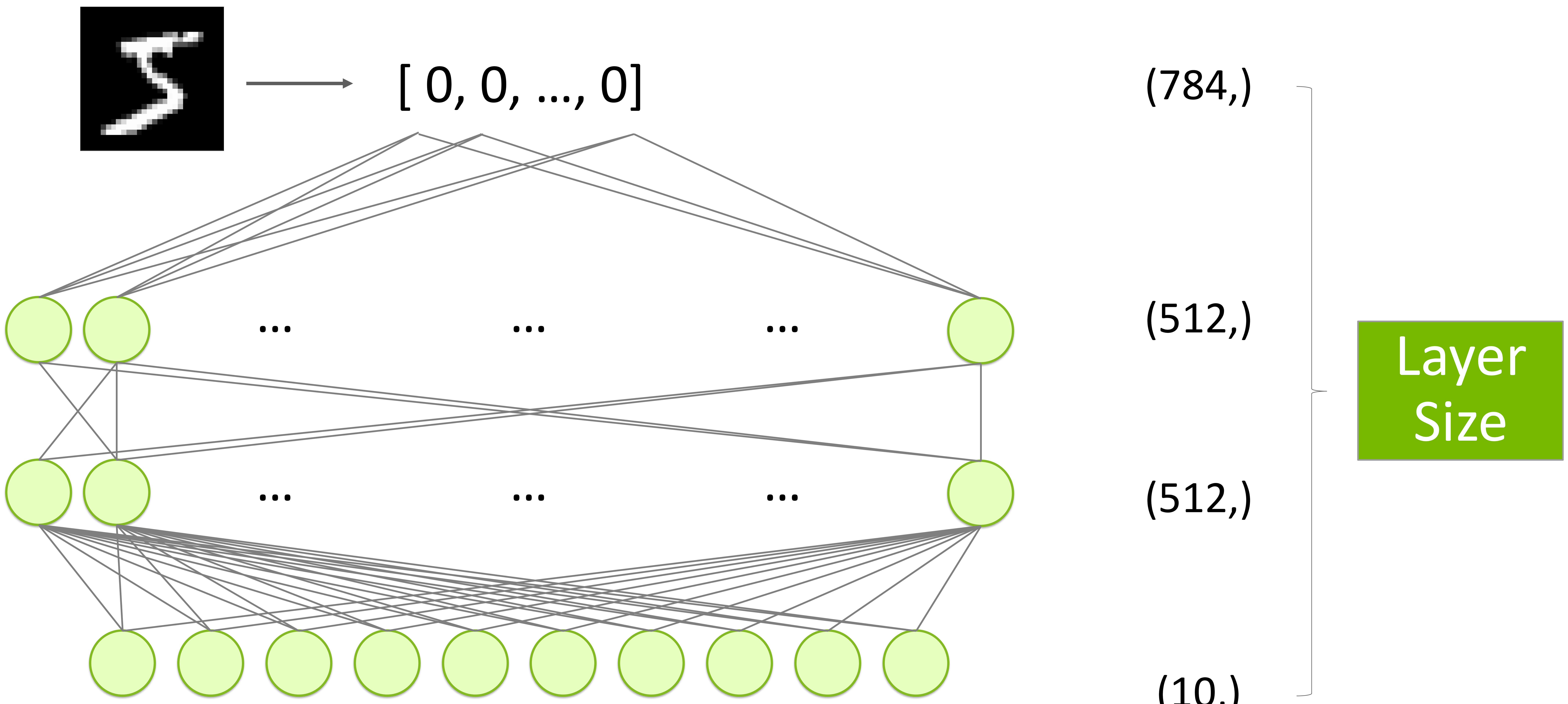
Overfitting

- When model performs well on the training data, but not the validation data (evidence of memorization)
- Ideally the accuracy and loss should be similar between both datasets



From Regression to Classification

An MNIST Model



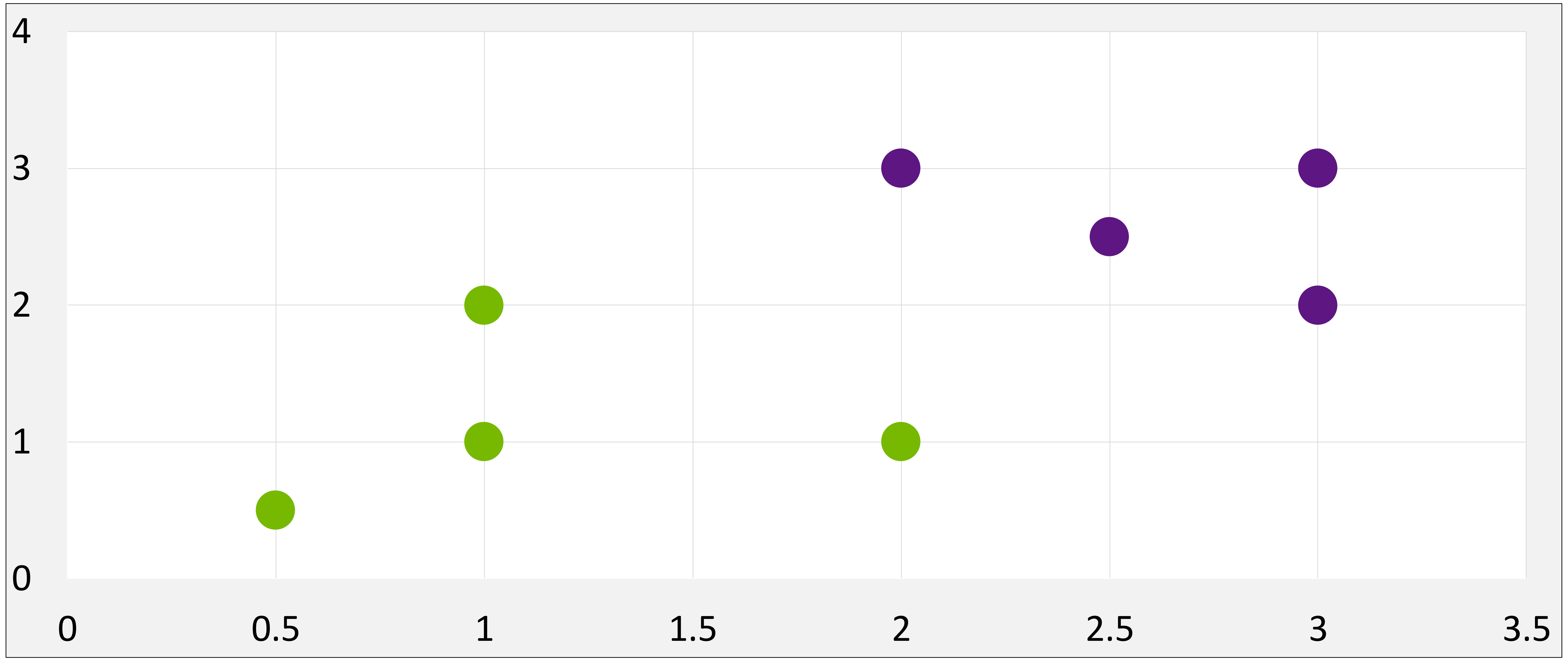
An MNIST Model

During Prediction

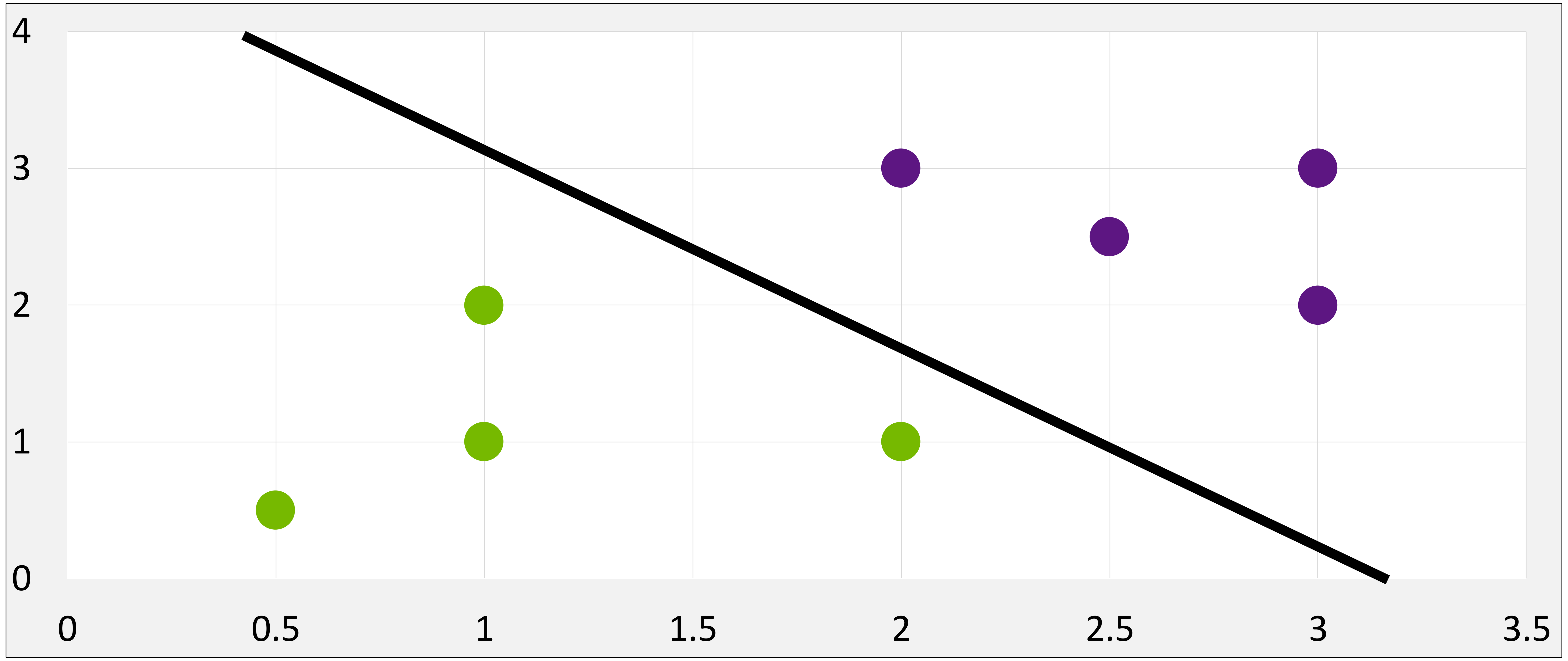
An MNIST Model

During Training

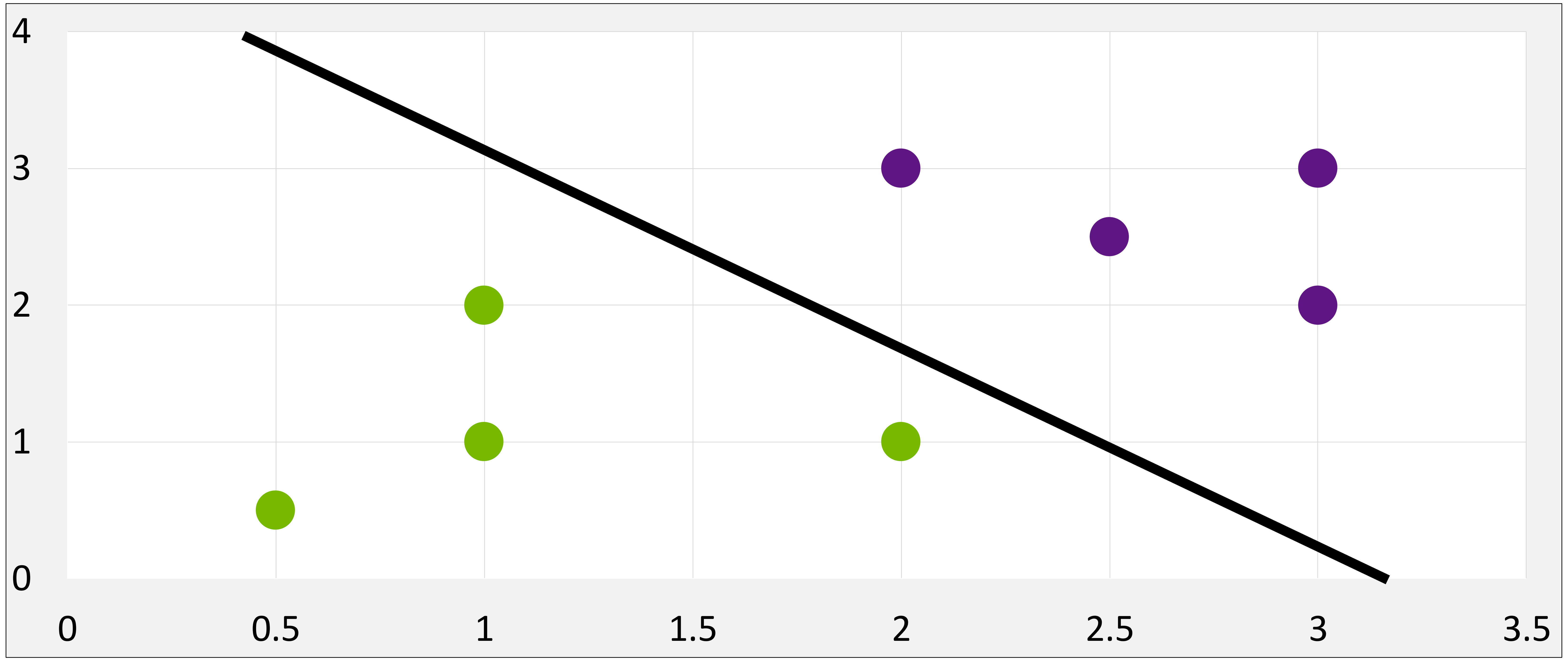
RMSE For Probabilities?

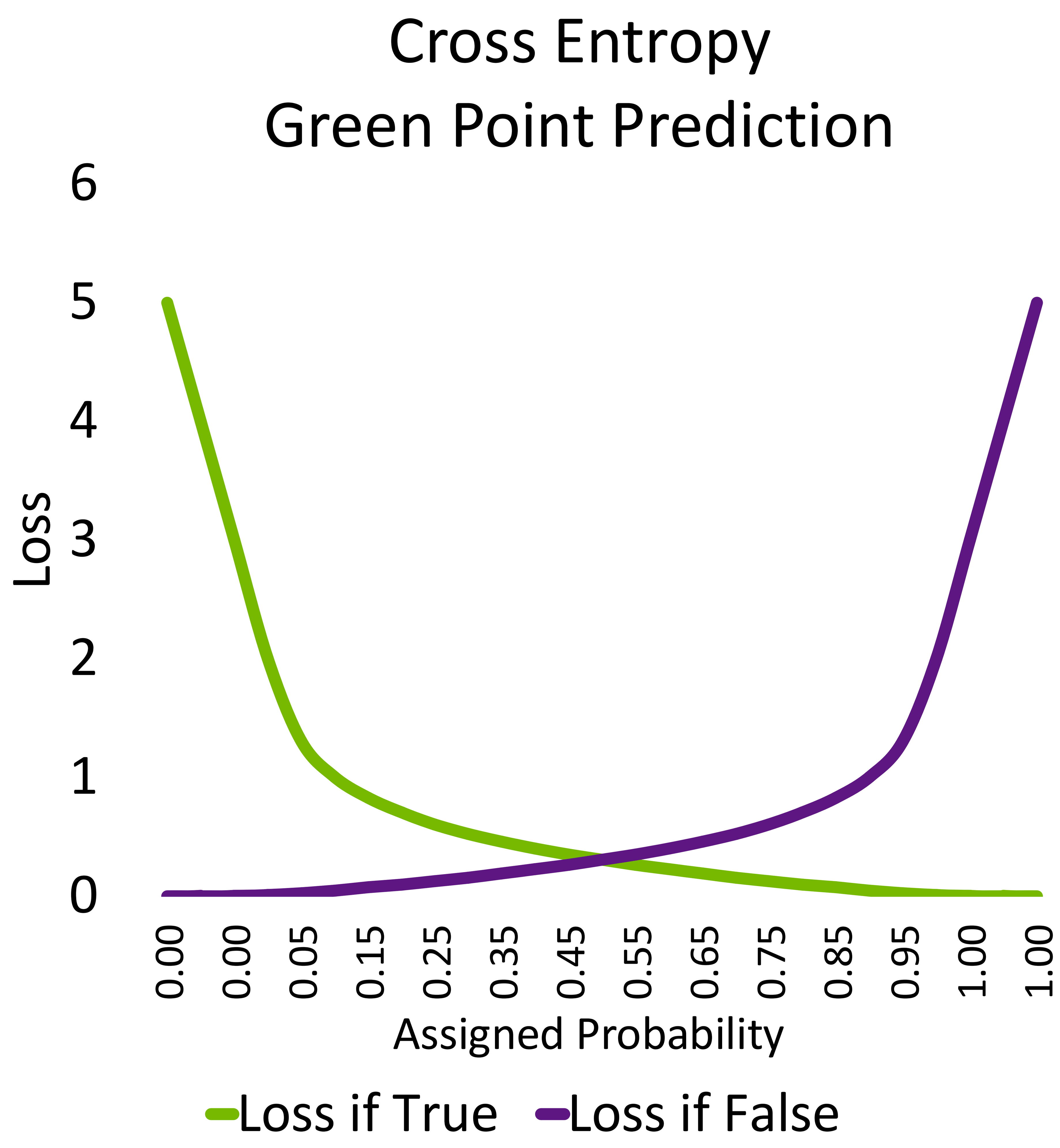


RMSE For Probabilities?

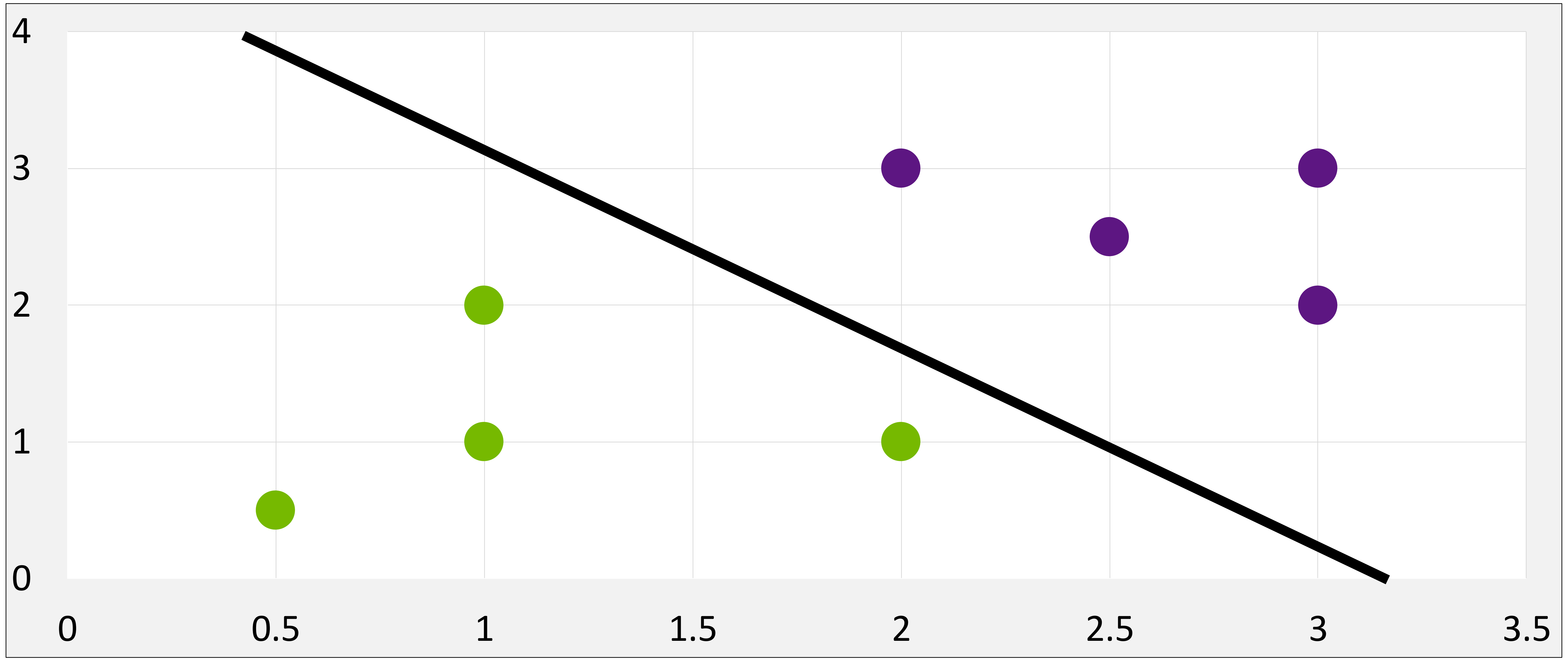


Cross Entropy





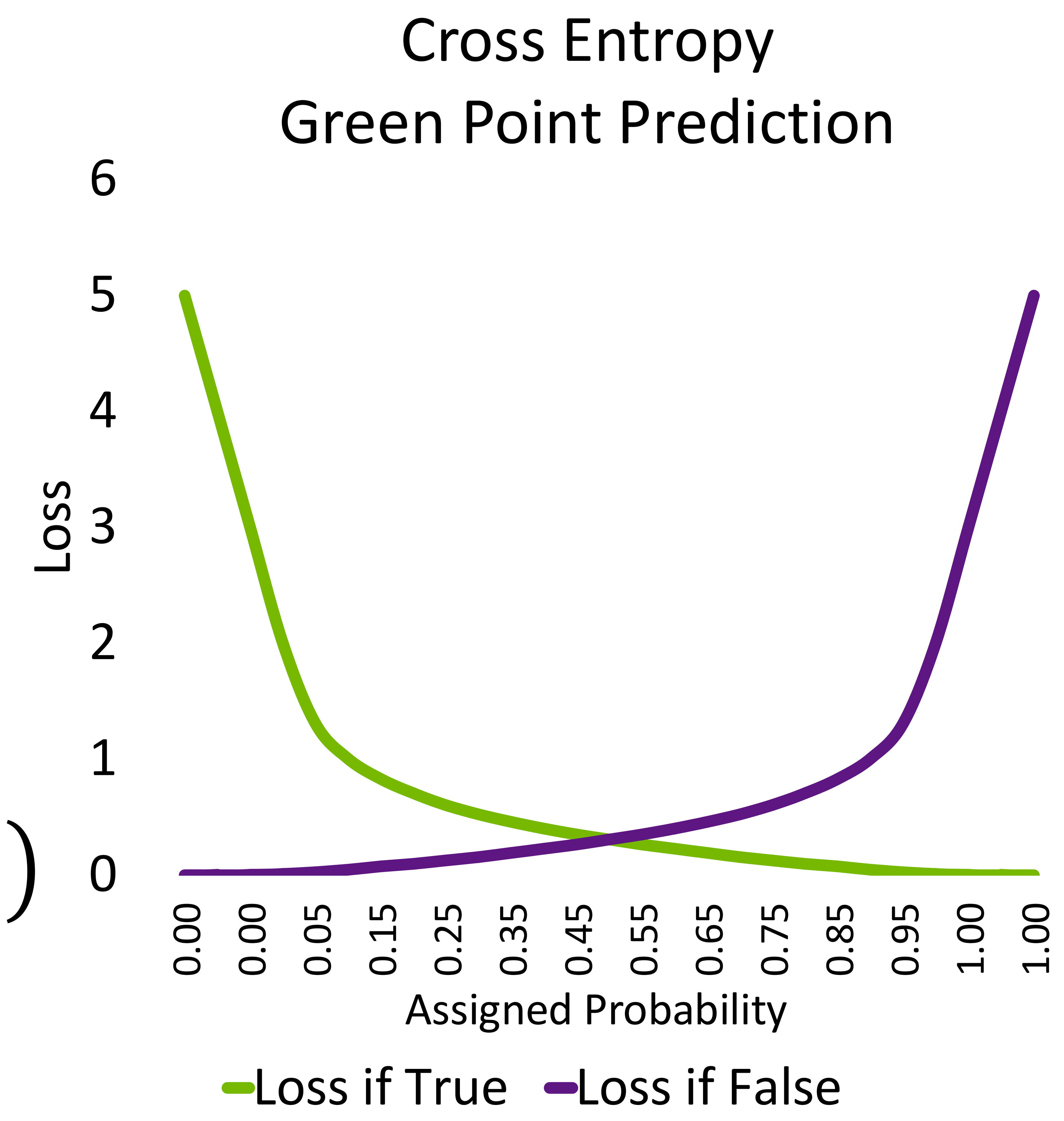
Cross Entropy



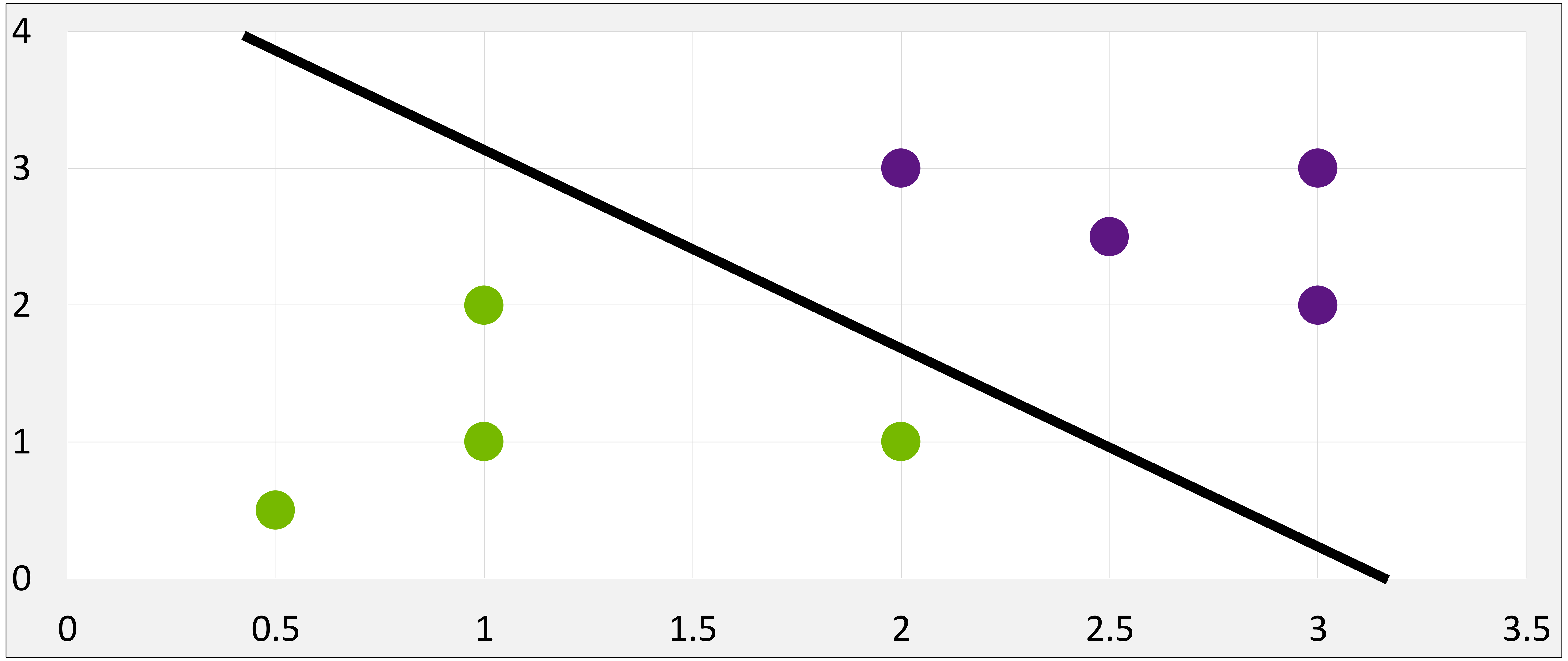
$$Loss = -\left(\left(t(x) \cdot \log(p(x)\right) + \left(1 - t(x)\right) \cdot \log(1 - p(x))\right)$$

$$t(x) = target\ (0\ if\ False, 1\ if\ True)$$

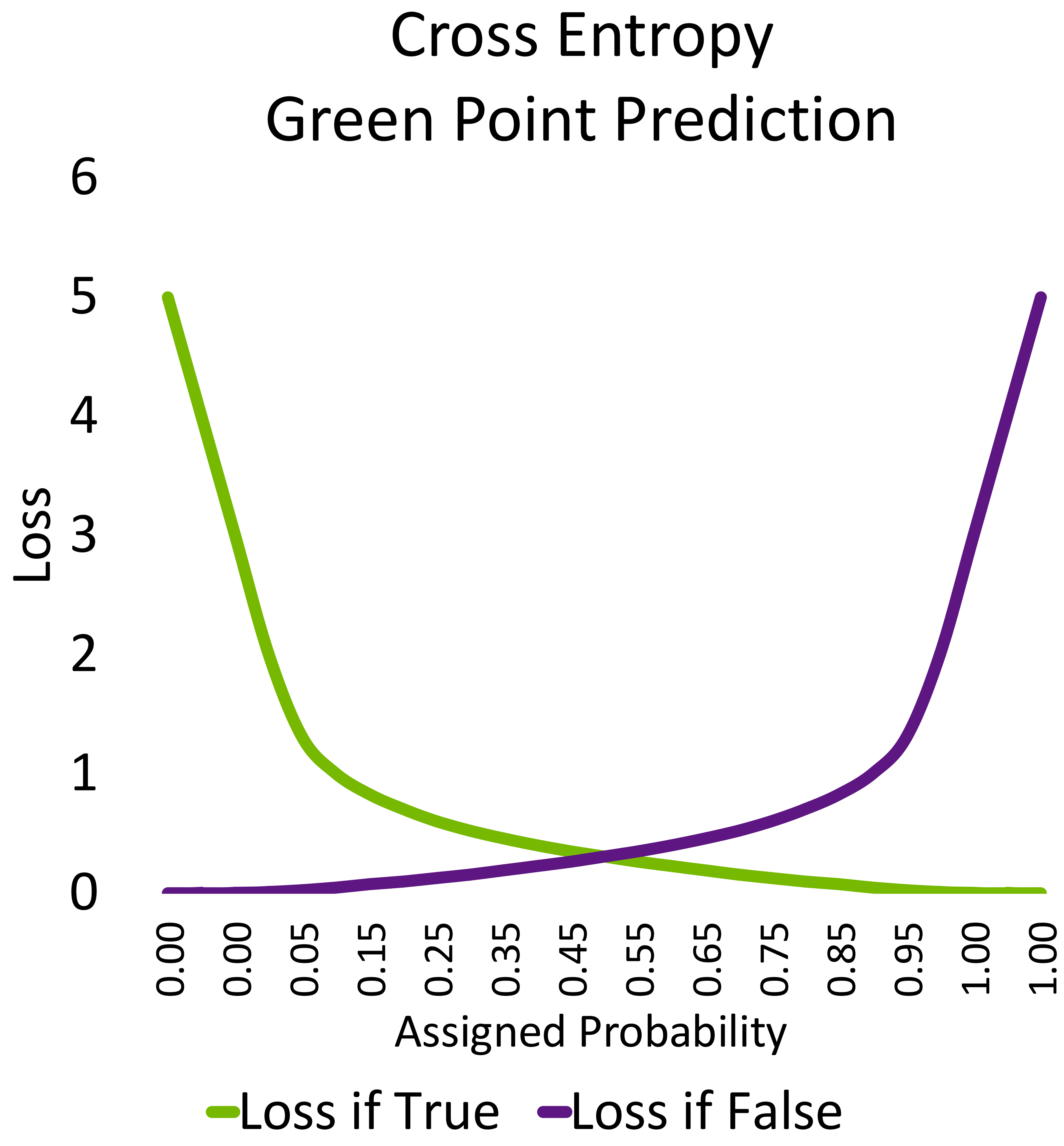
$$p(x) = probability\ prediction\ of\ point\ x$$



Cross Entropy



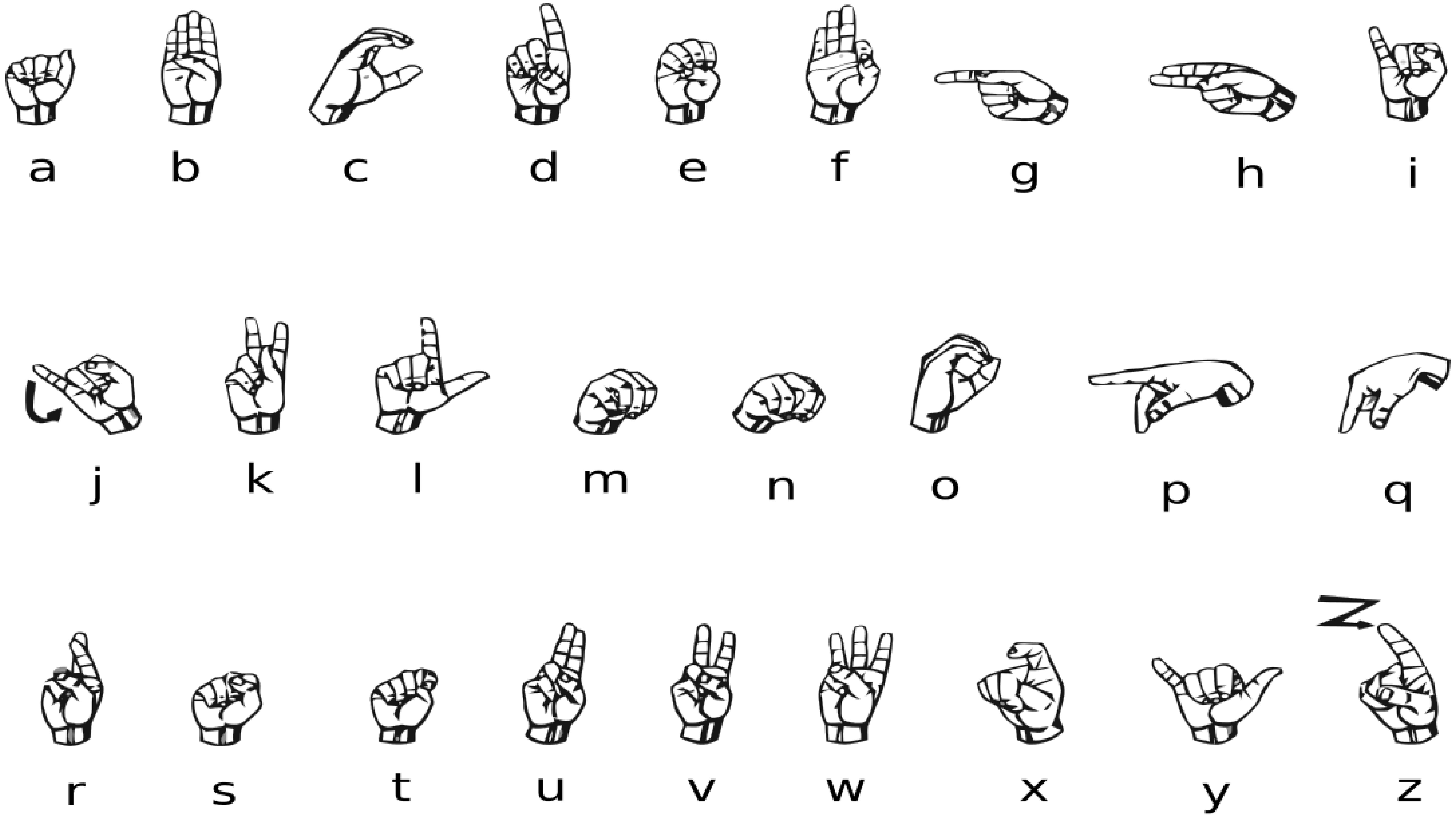

```python
def cross_entropy(y_hat, y_actual):
    """Infinite error for misplaced confidence."""
    loss = log(y_hat) if y_actual else log(1-y_hat)
    return -1*loss
```



Bringing it Together

The Next Exercise

The American Sign Language Alphabet



Let’s go!

Appendix: Gradient Descent

Helping the Computer Cheat Calculus

Learning from Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 = \frac{1}{n}\sum_{i=1}^{n}(y - (mx + b))^2$$

$$MSE = \frac{1}{2}((3 - (m(1) + b))^2 + (5 - (m(2) + b))^2)$$

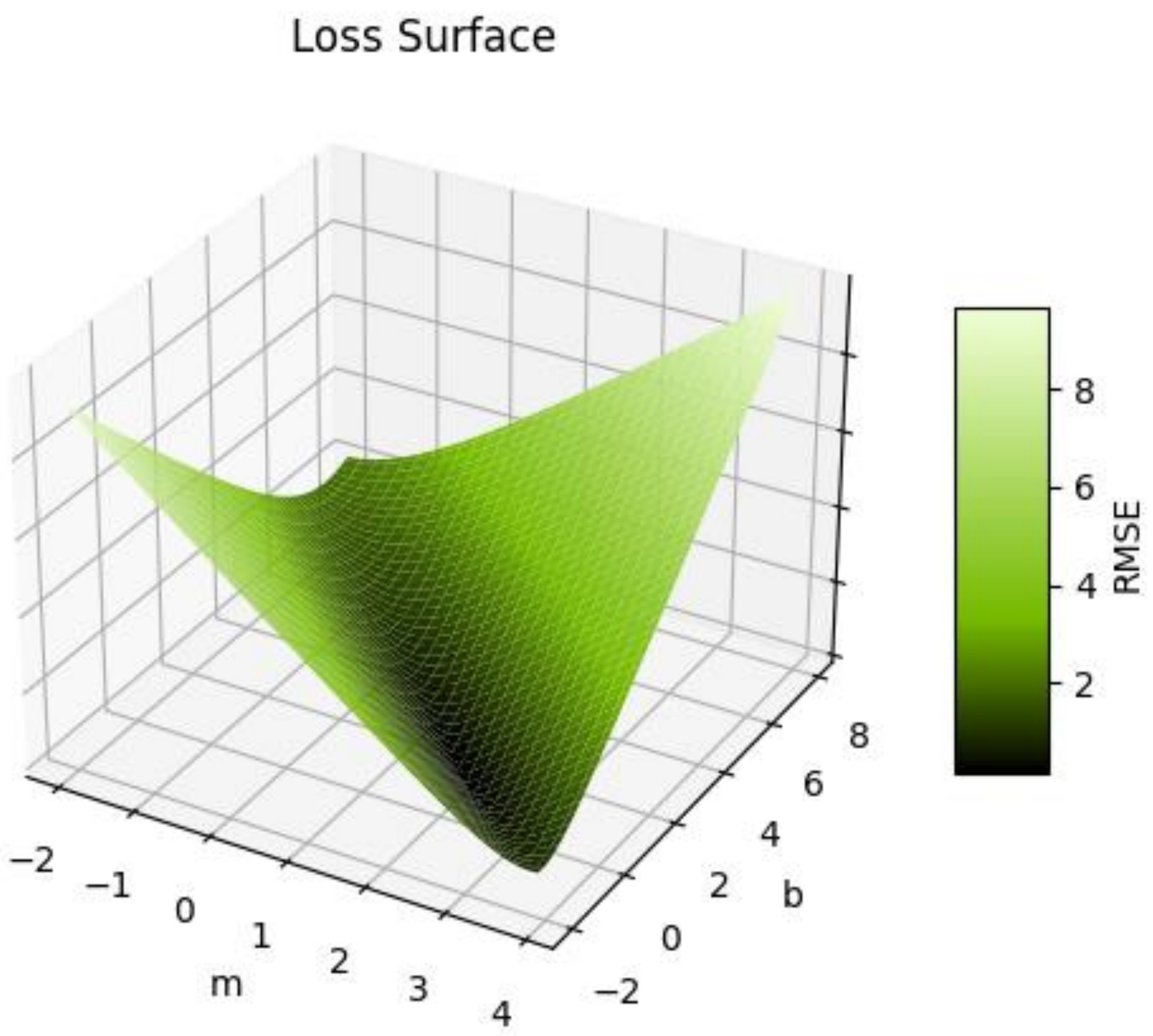
$$\frac{\partial MSE}{\partial m} = 5m + 3b - 13 \qquad \frac{\partial MSE}{\partial b} = 3m + 2b - 8$$

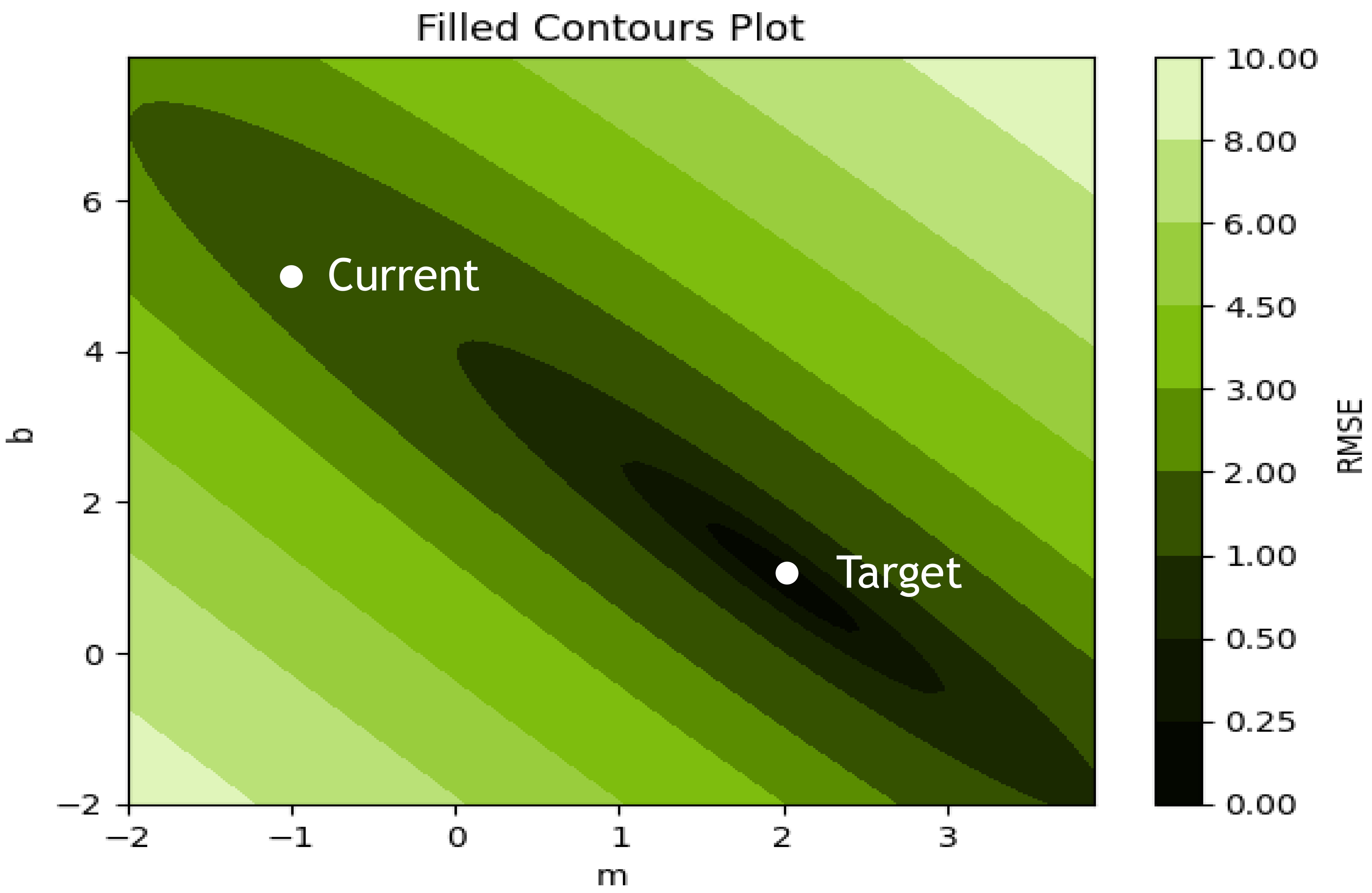
$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$m = -1$

$\mathrm{b} = 5$

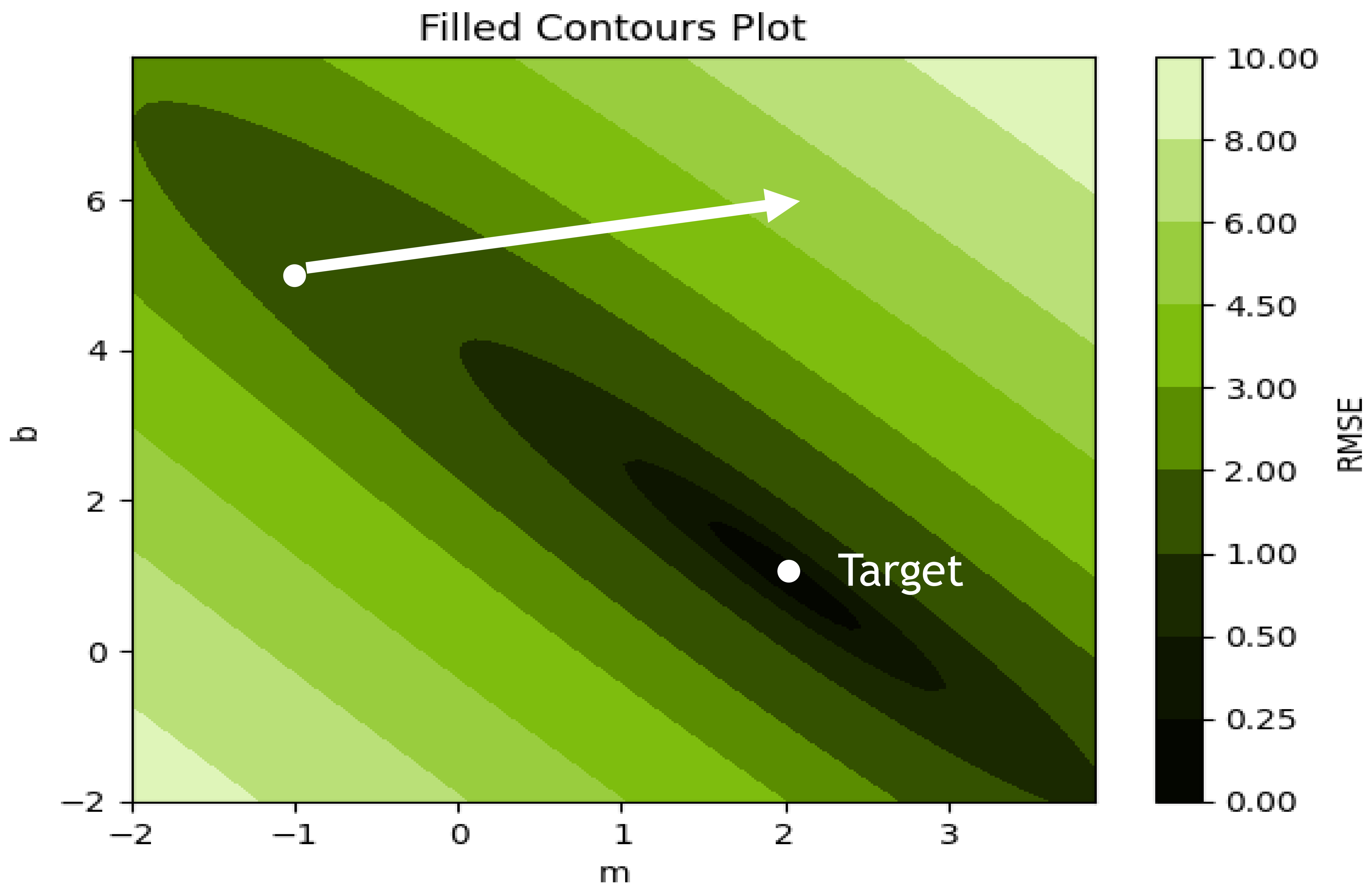
The Loss Curve





The Loss Curve

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

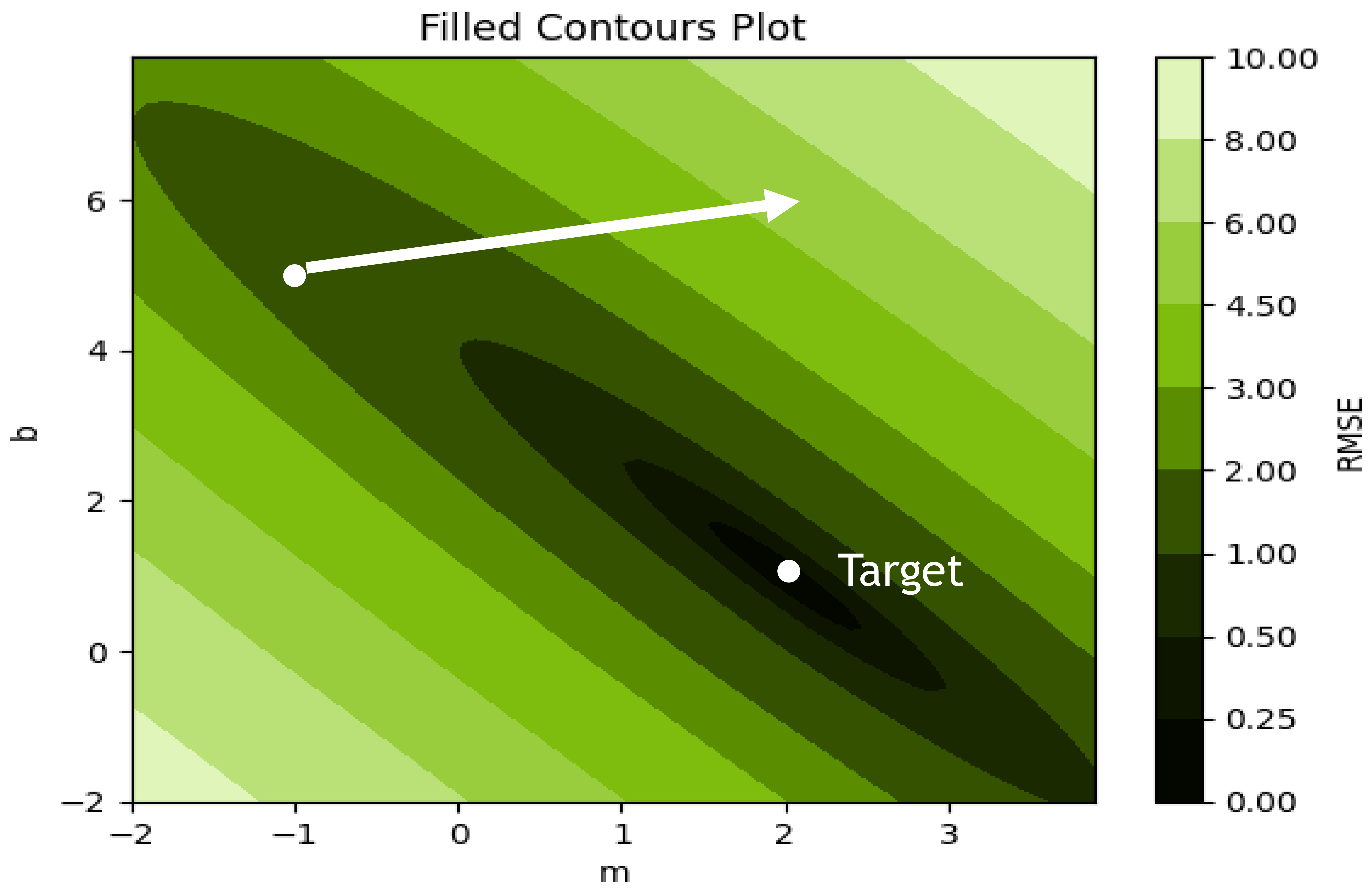


The Loss Curve

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$



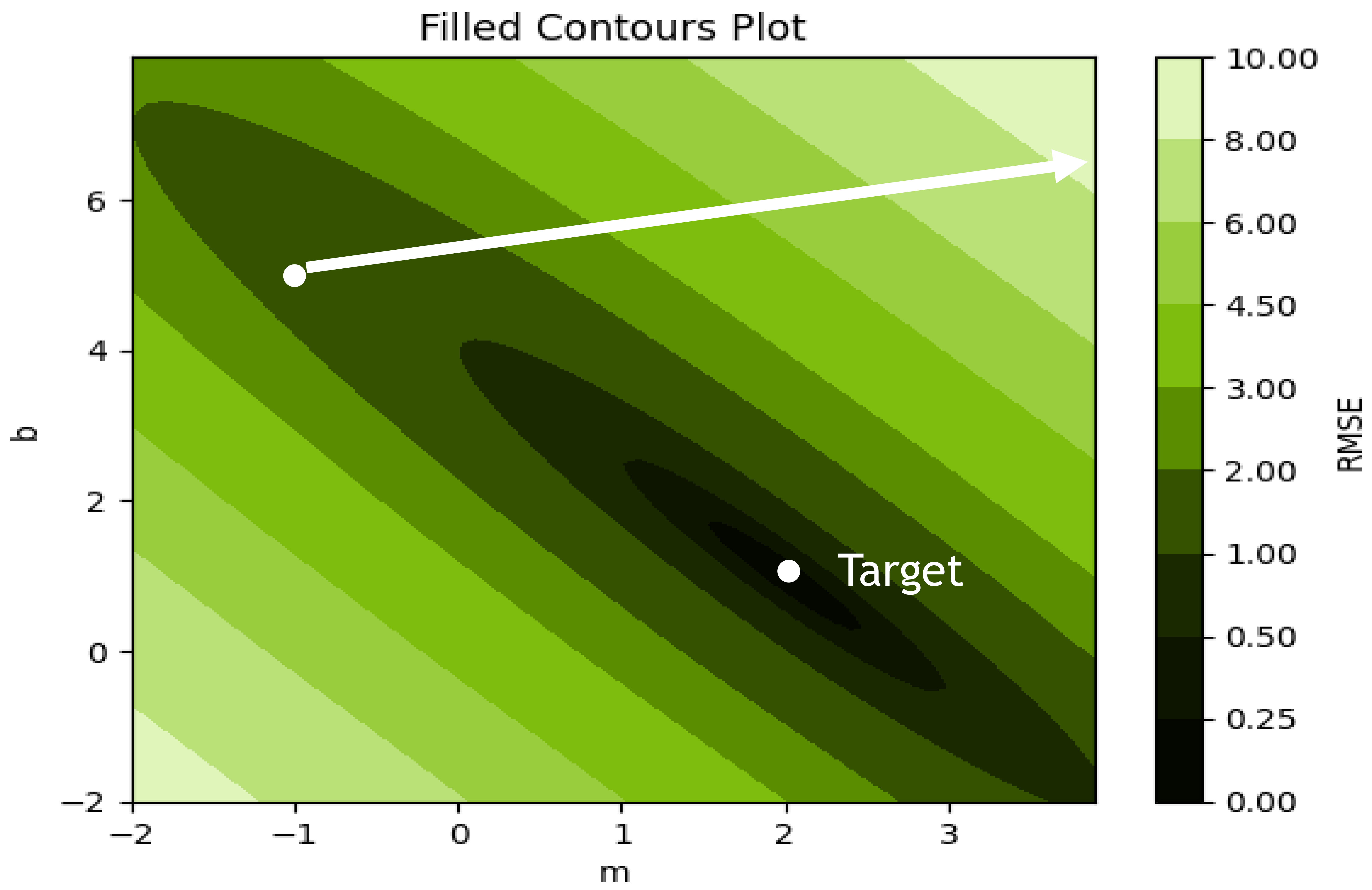
The Loss Curve

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\lambda = 2$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$



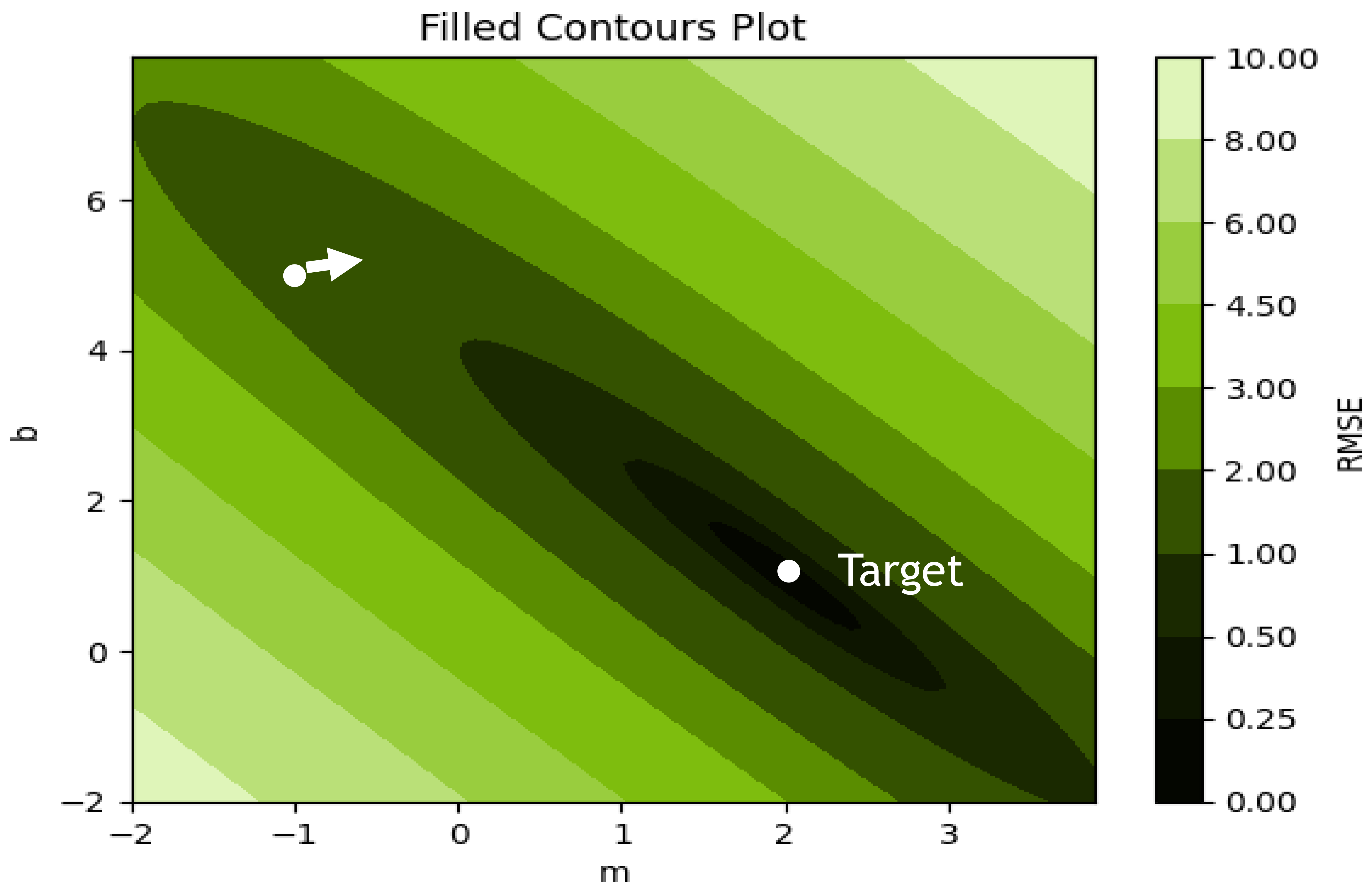
The Loss Curve

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\lambda = .1$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$

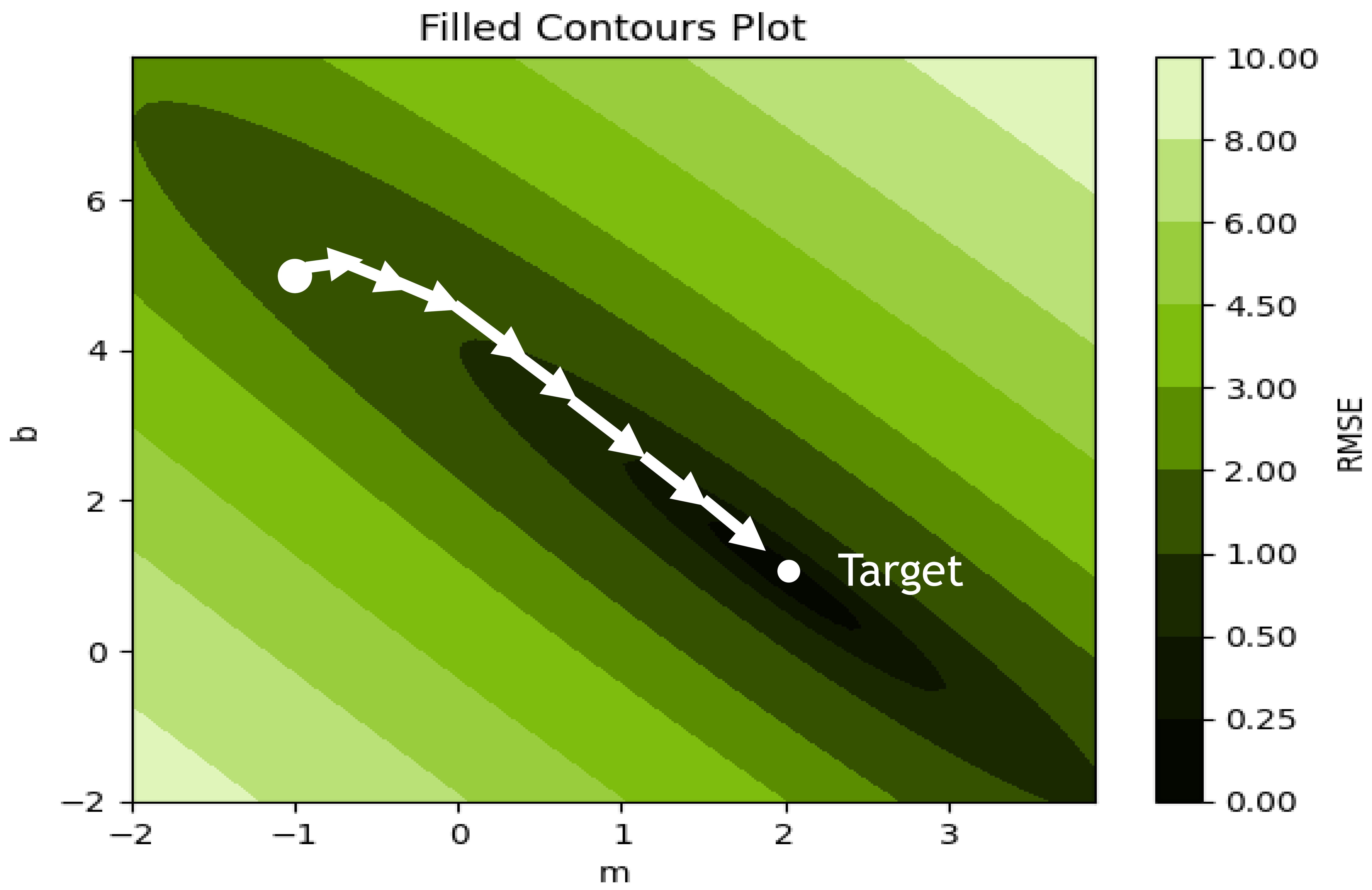


The Loss Curve

$$\lambda = .1$$

$$m \coloneqq -1 + 3\,\lambda = -0.7$$

$$b \coloneqq 5 + \lambda = 5.1$$



NVIDIA