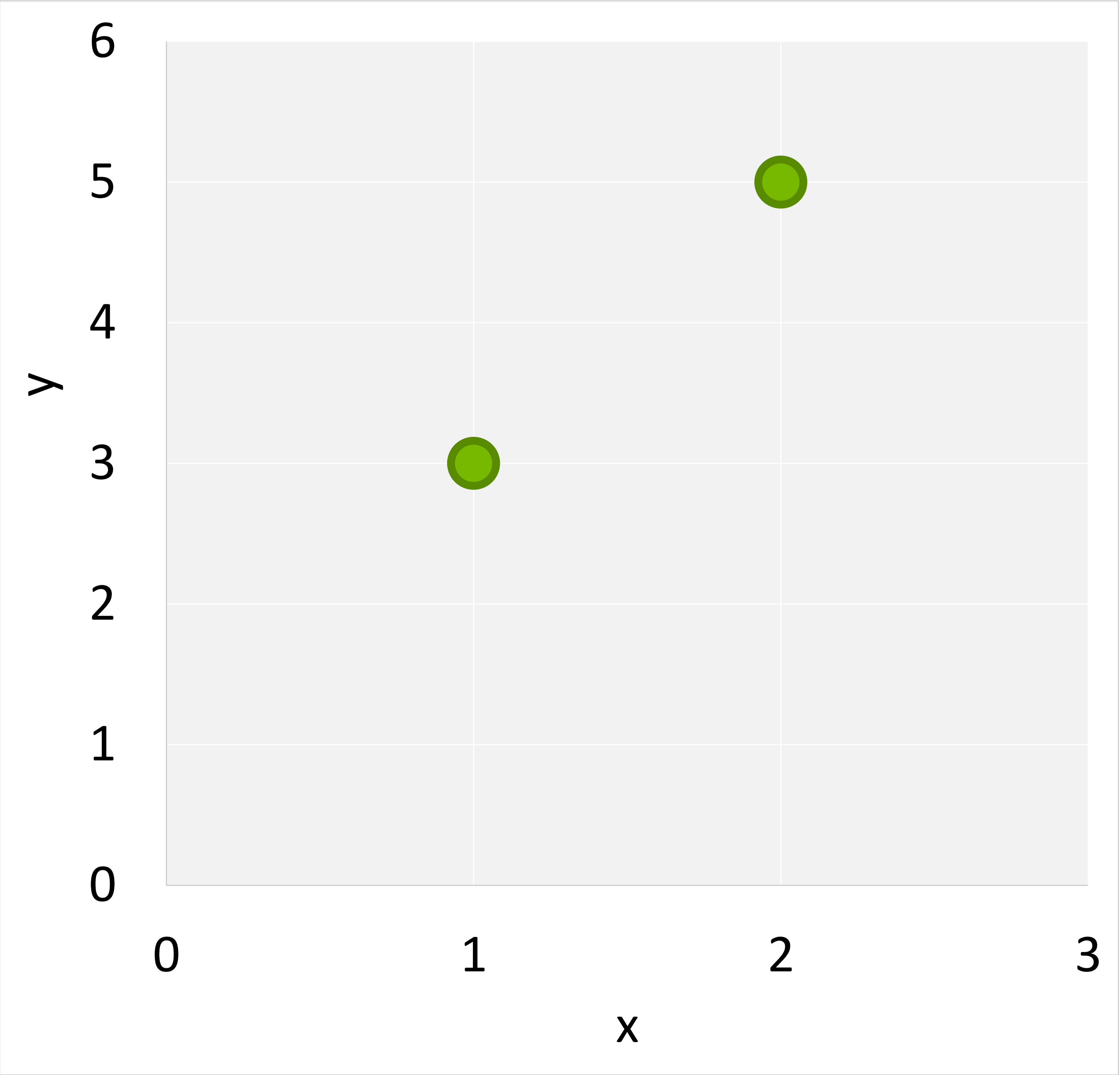
Agenda

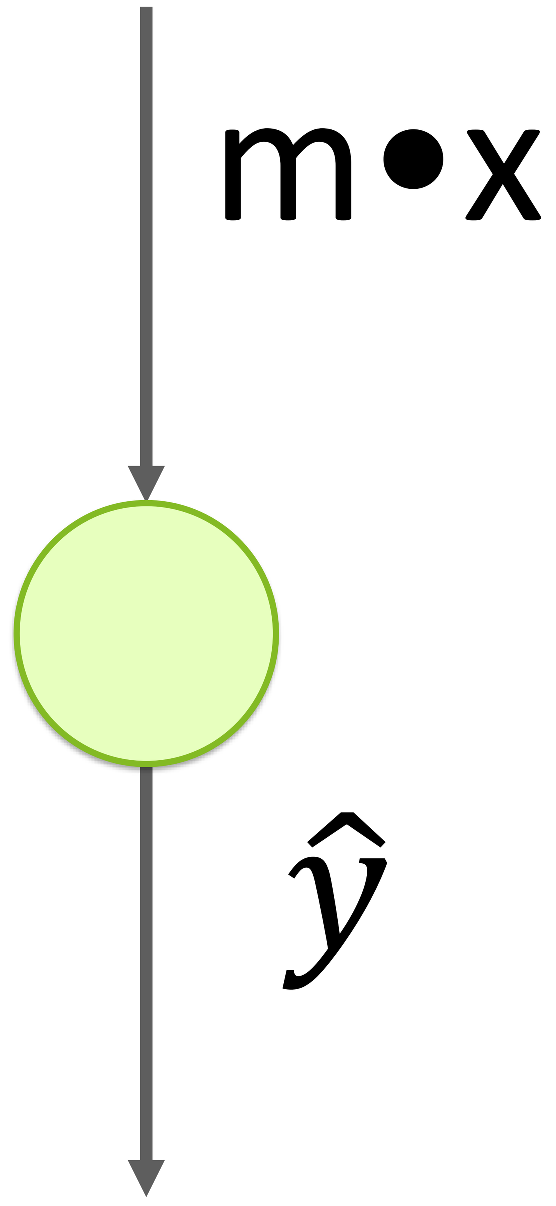
- Part 1: An Introduction to Deep Learning
- Part 2: How a Neural Network Trains
- Part 3: Convolutional Neural Networks
- Part 4: Data Augmentation and Deployment
- Part 5: Pre-Trained Models
- Part 6: Advanced Architectures

A Simpler Model

$$y = mx + b$$

x	y
1	3
2	5





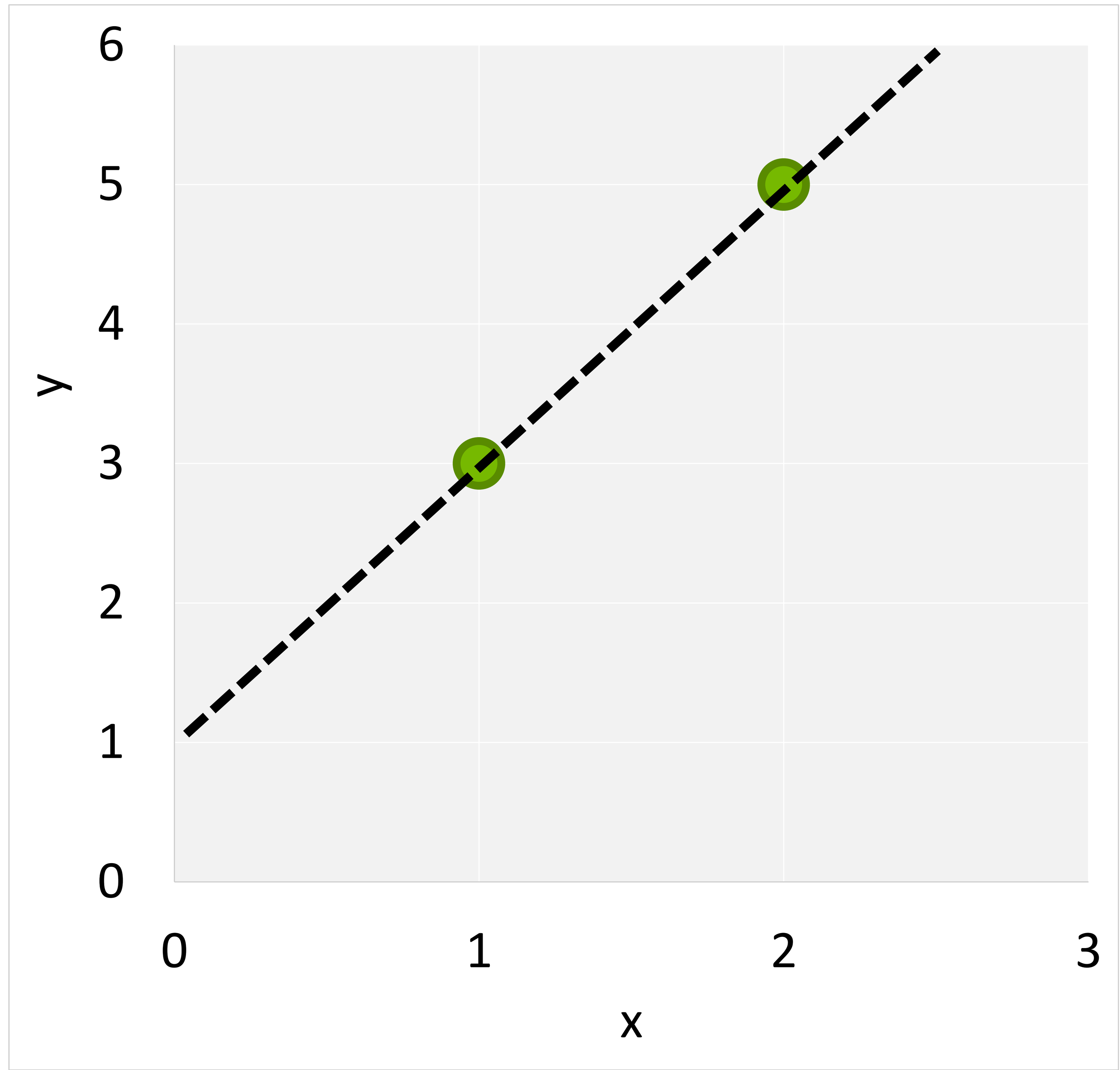
$$m = ?$$

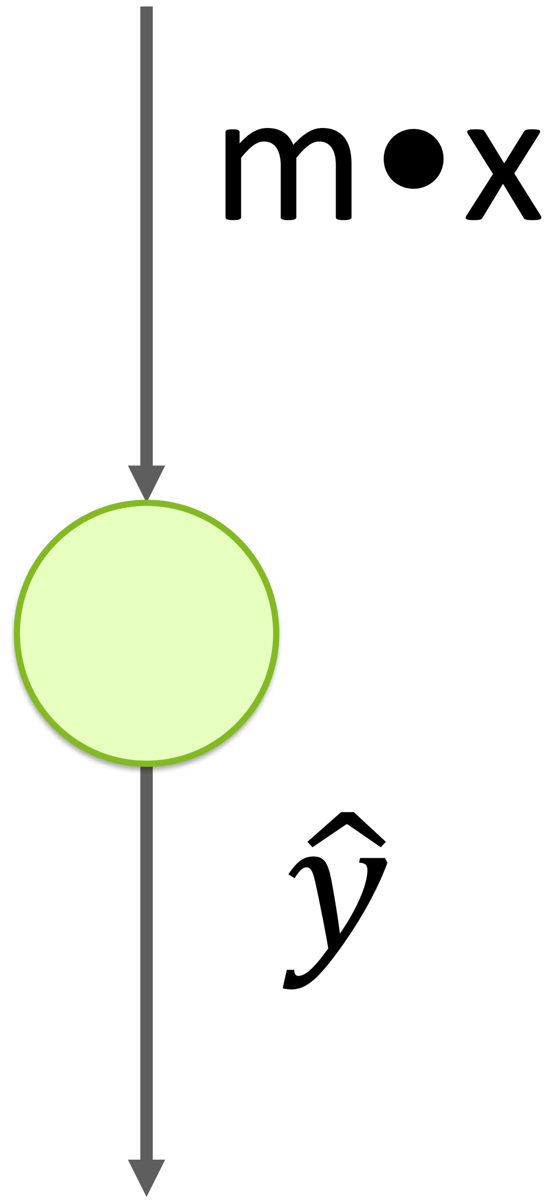
$$b = ?$$

A Simpler Model

$$y = mx + b$$

x	y
1	3
2	5





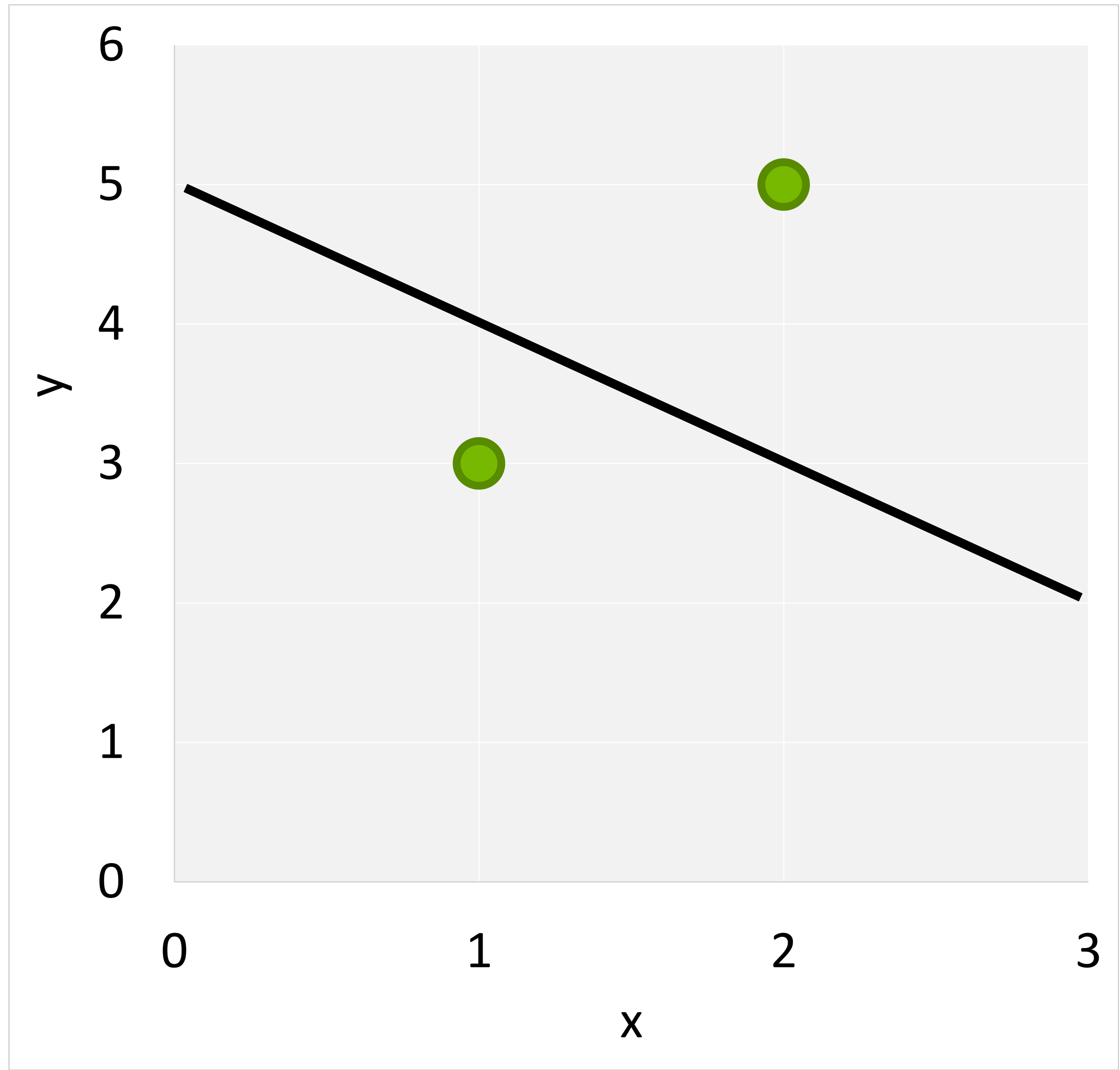
$m = ?$

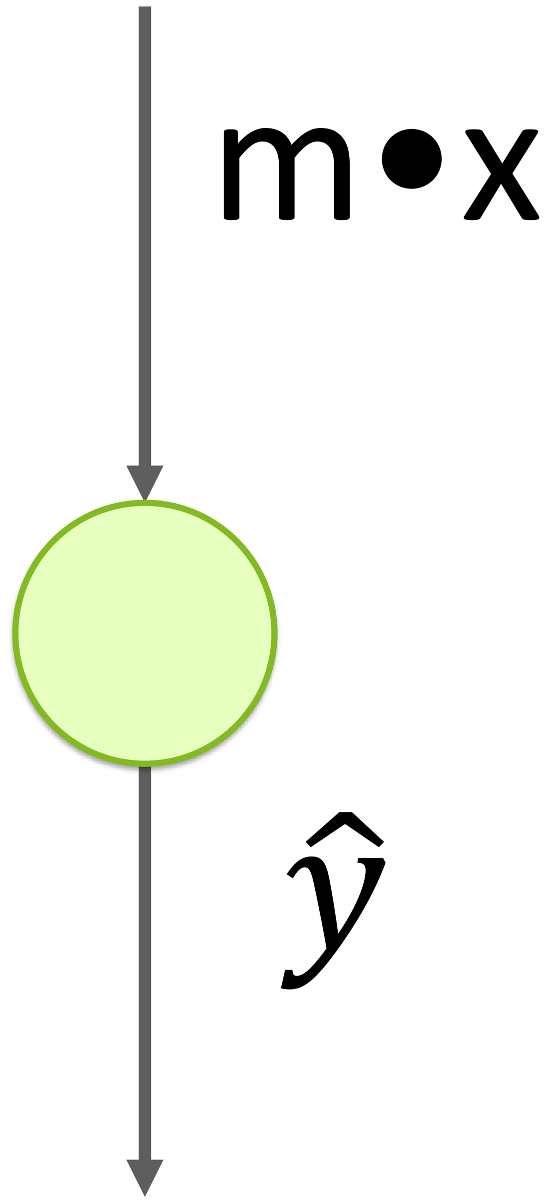
$b = ?$

A Simpler Model

$$y = mx + b$$

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





Start Random

$$m = -1$$

$$b = 5$$

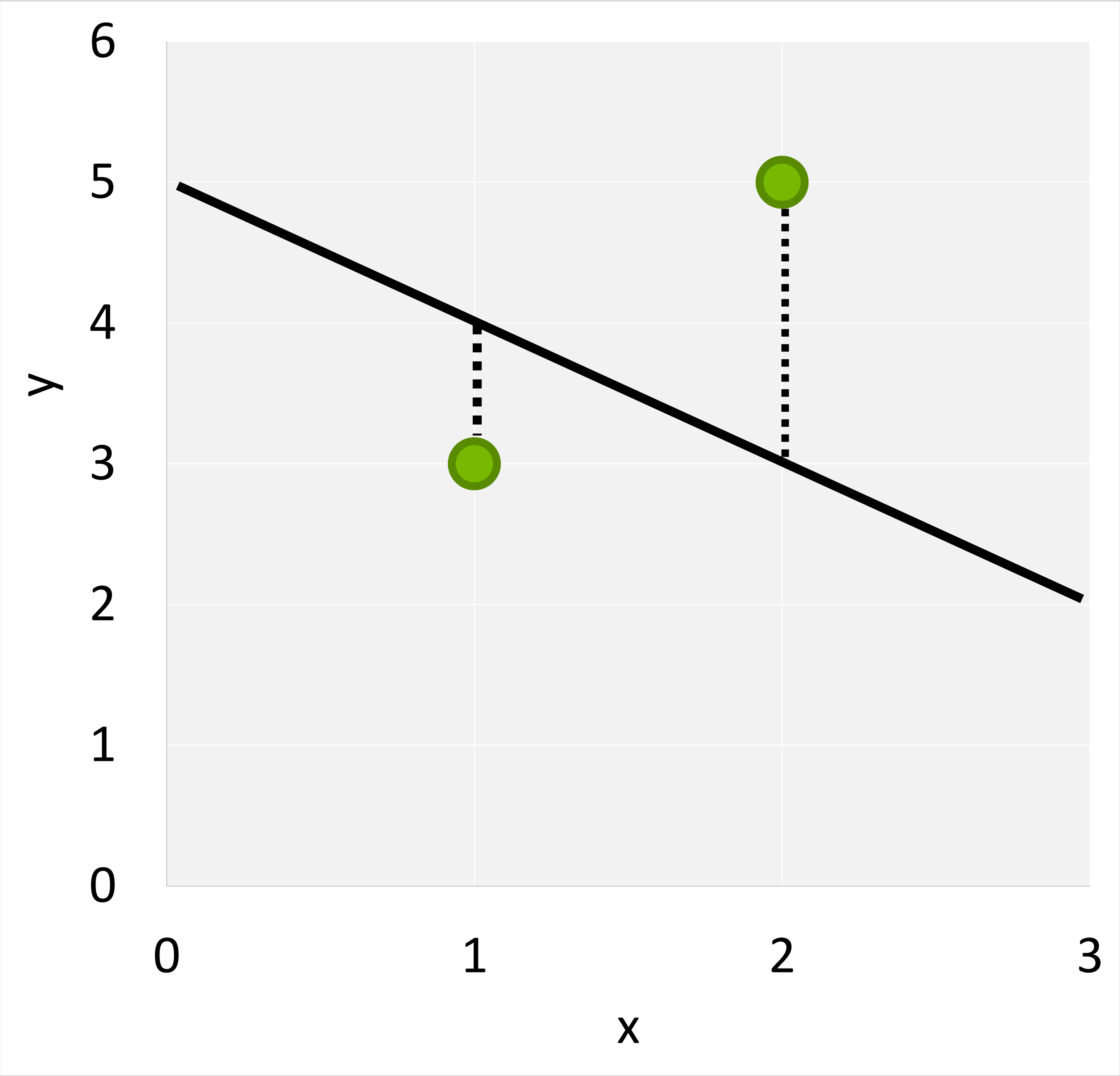
A Simpler Model

$$y = mx + b$$

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

MSE = 2.5

RMSE = 1.6



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

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

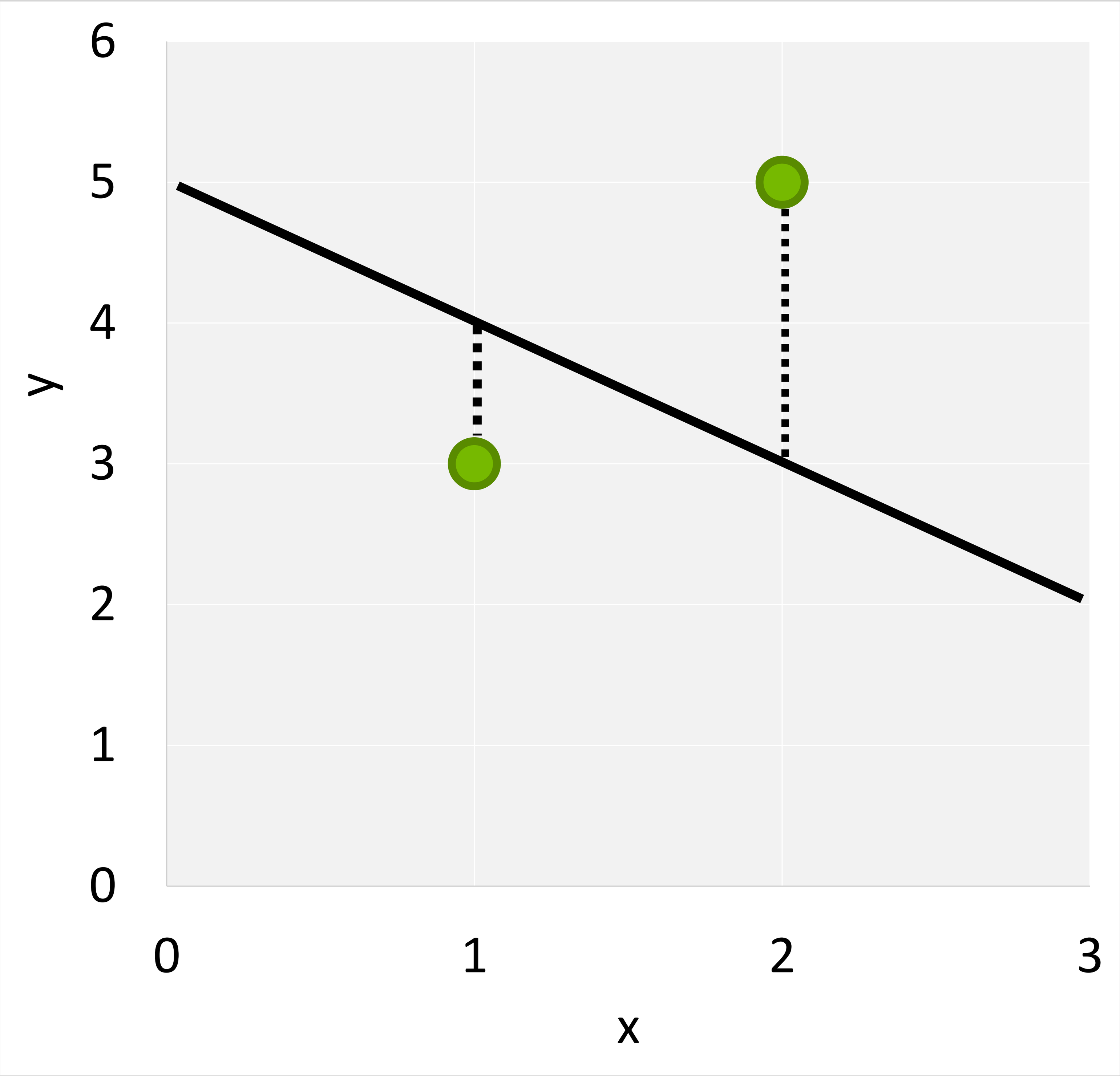
A Simpler Model

$$y = mx + b$$

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

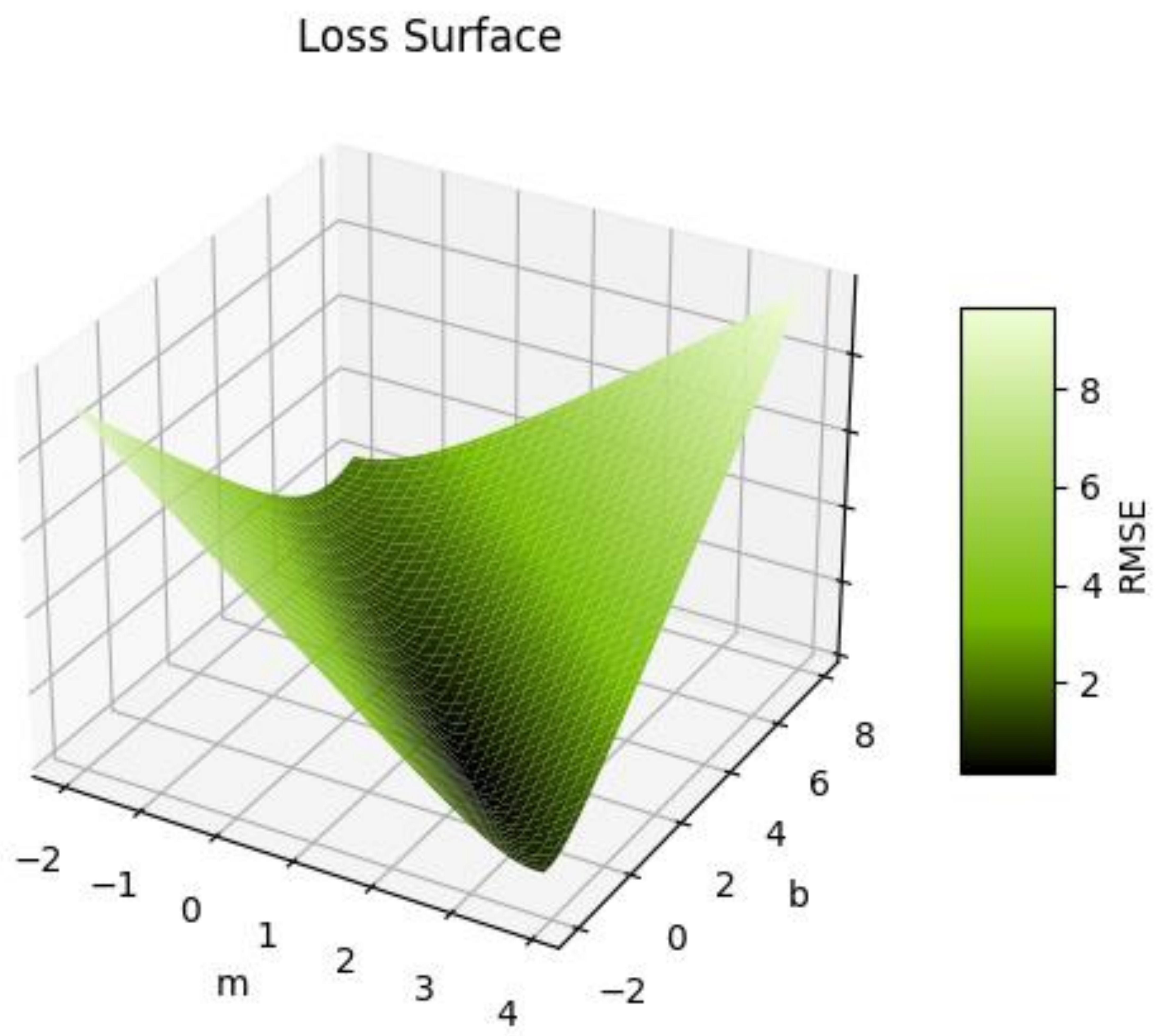
MSE = 2.5

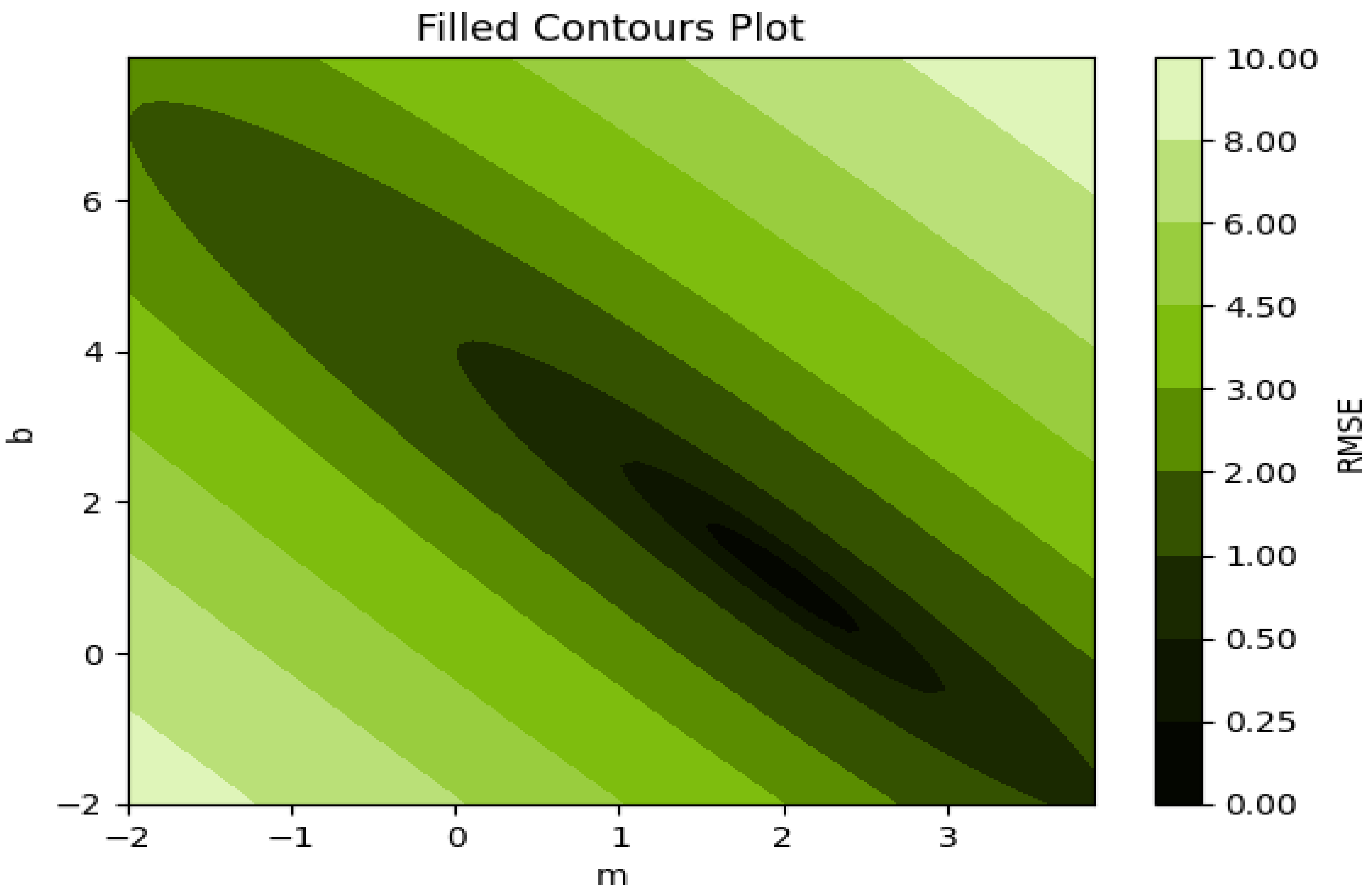
RMSE = 1.6



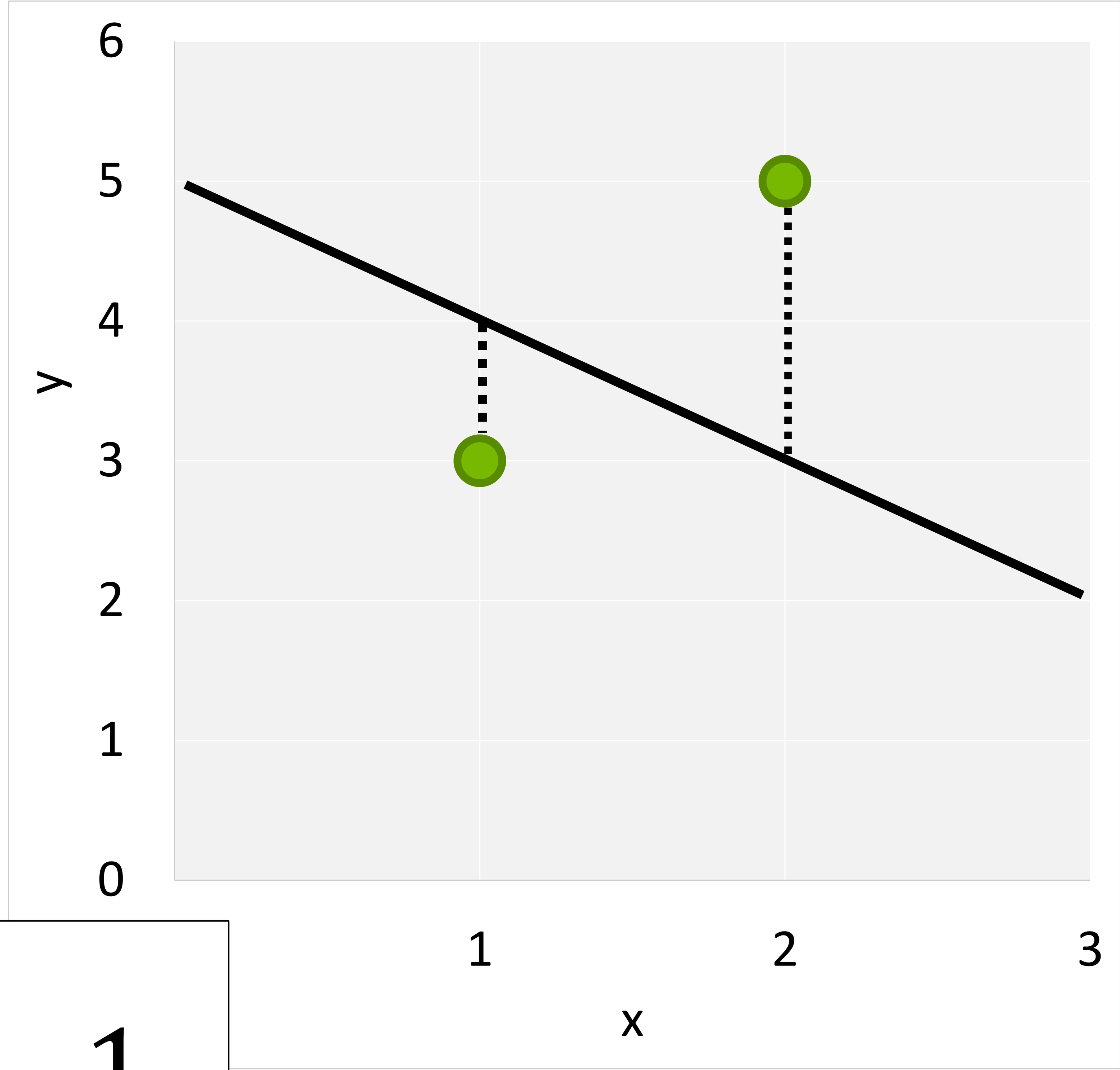

```python
data = [(1, 3), (2, 5)]
m = -1
b = 5


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



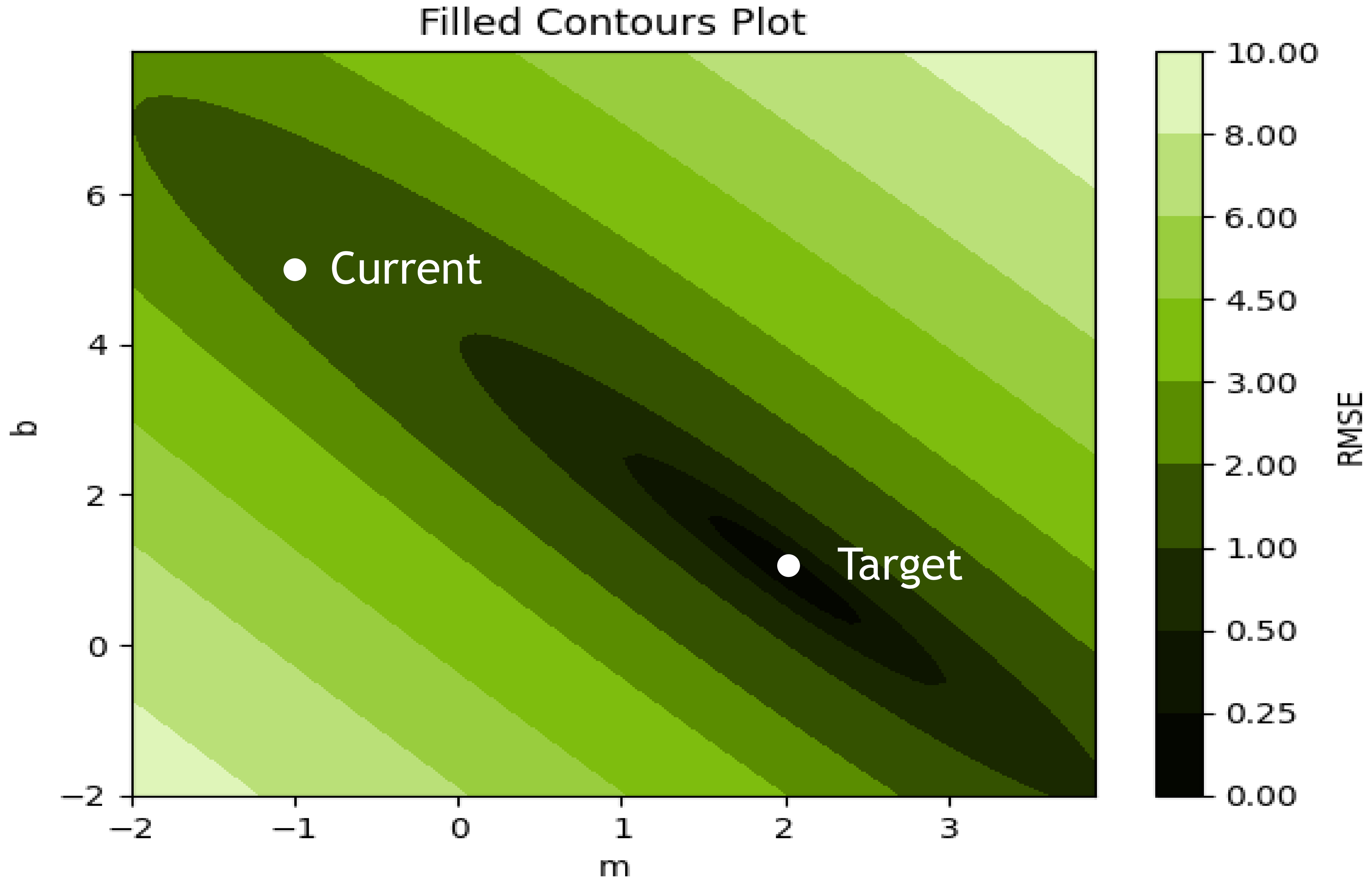


The Loss Curve

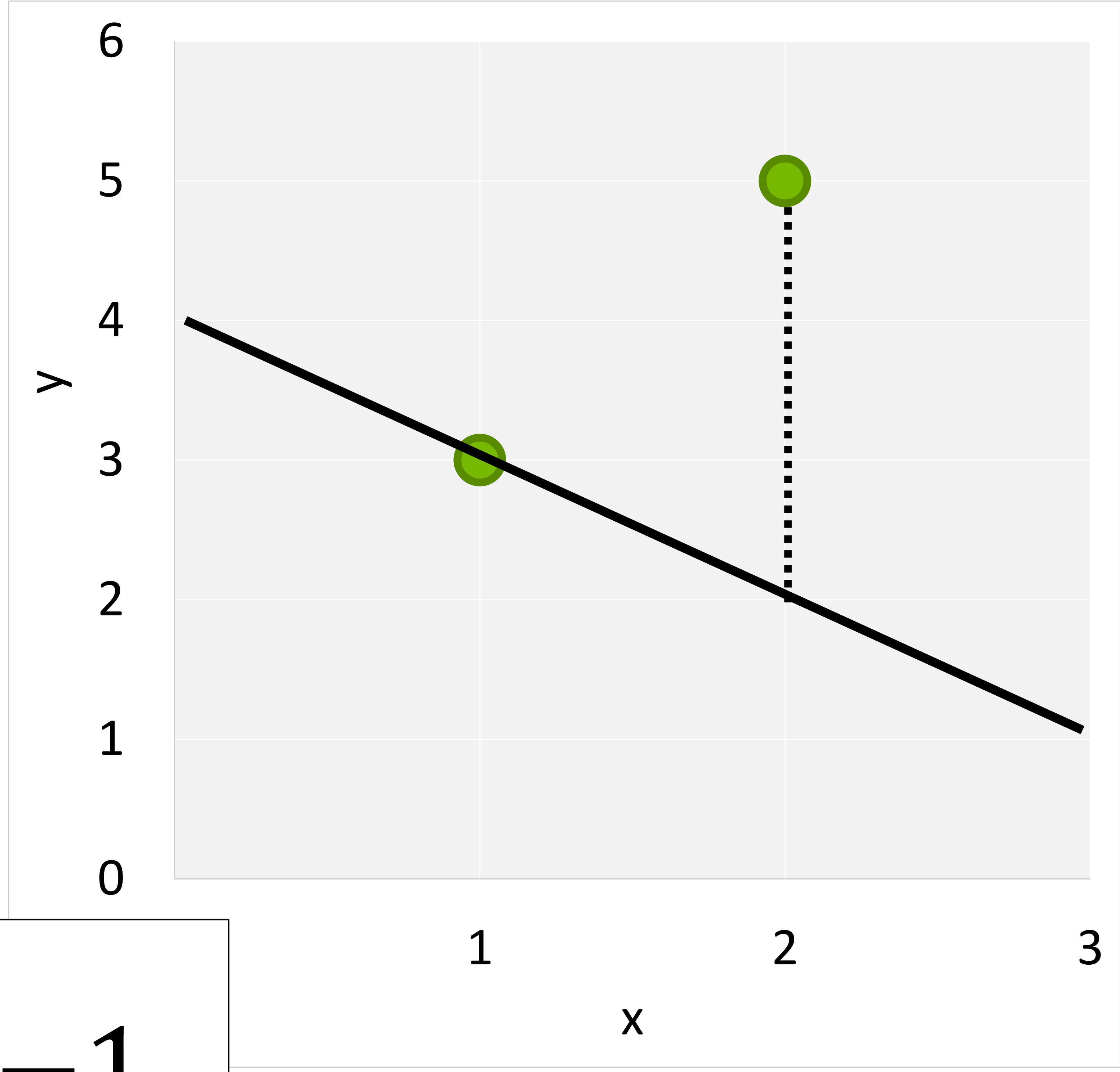


$$m = -1$$

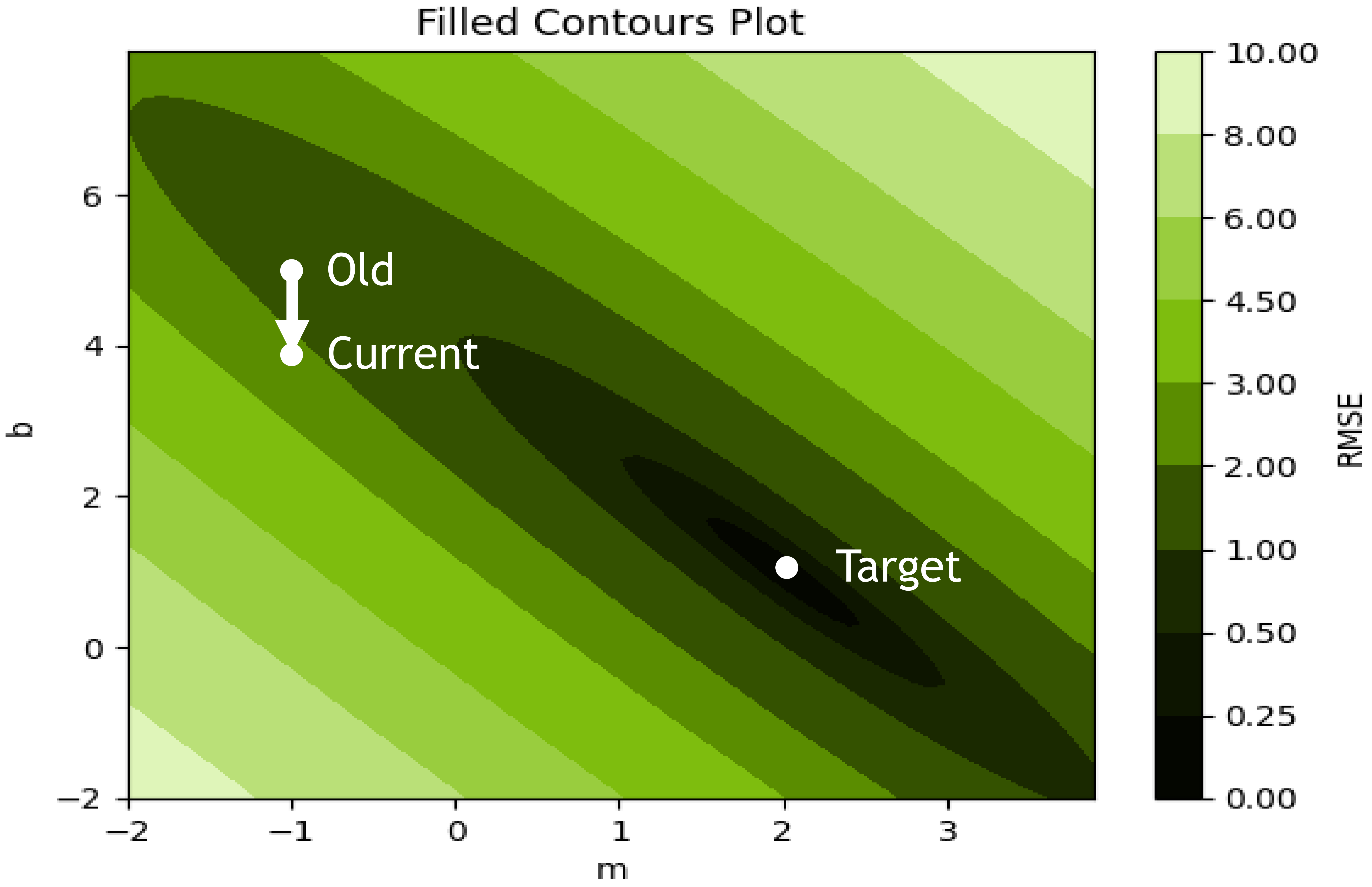
$$b = 5$$



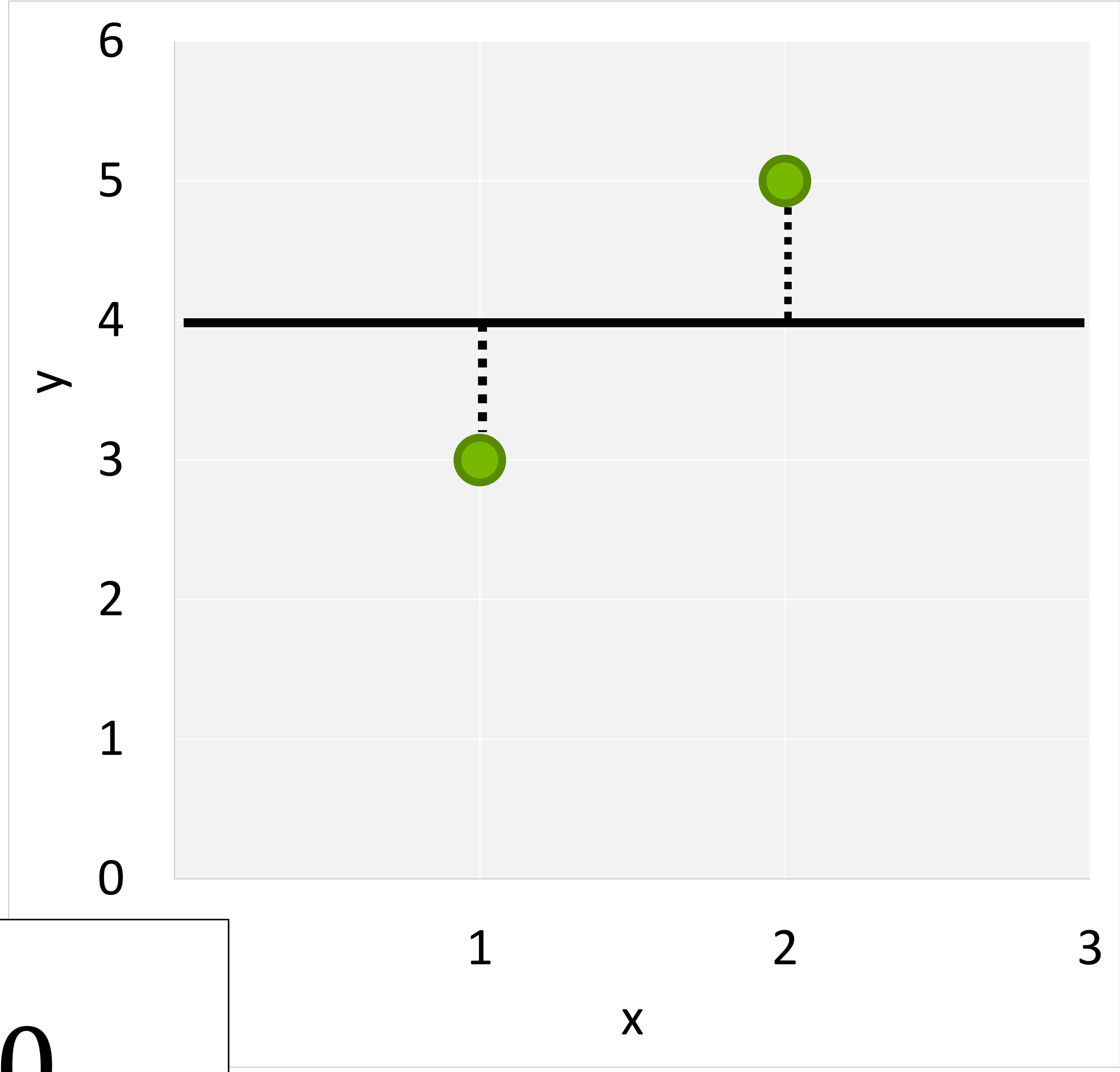
The Loss Curve



$$m = -1$$
$$b = 4$$

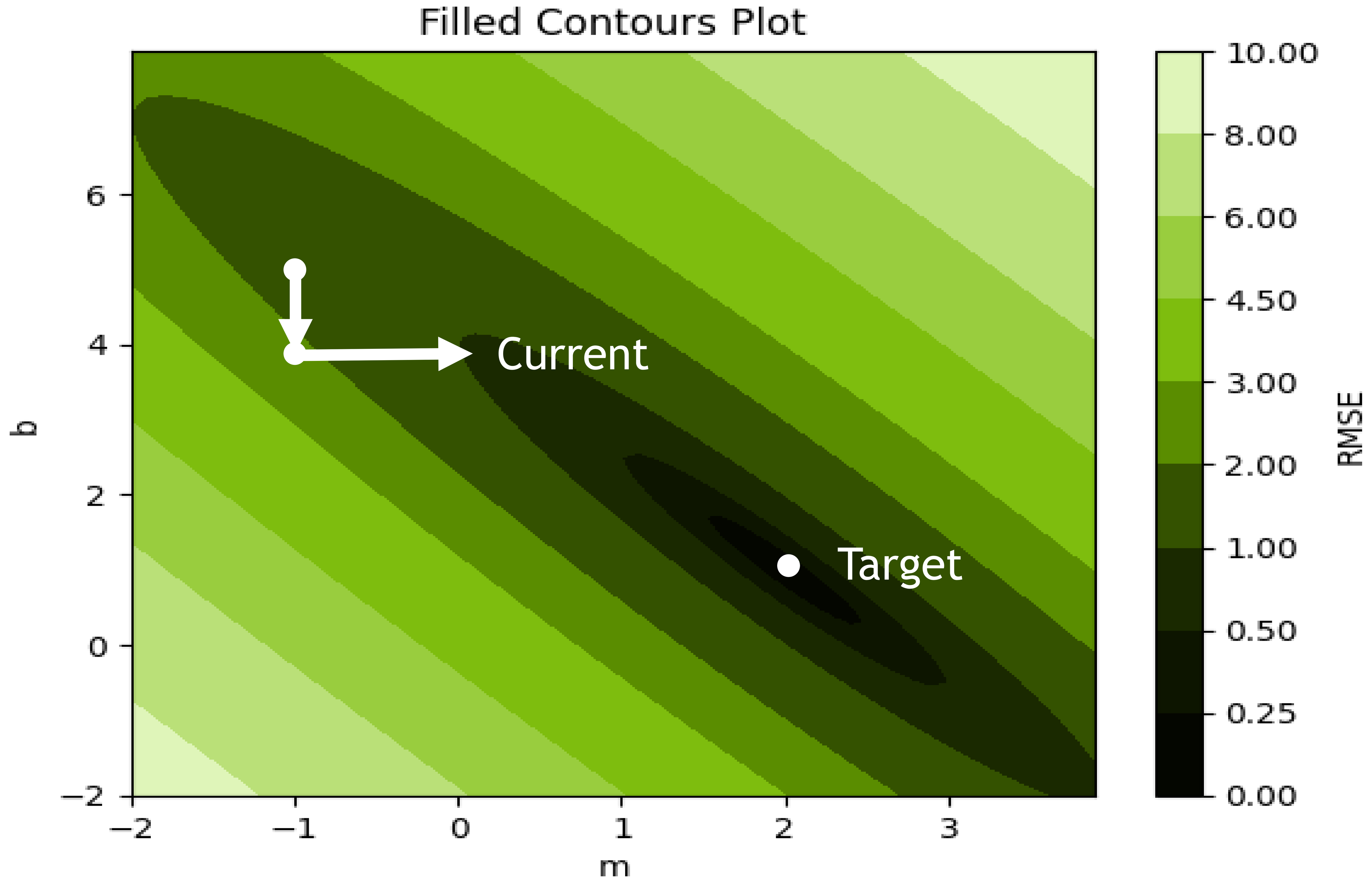


The Loss Curve



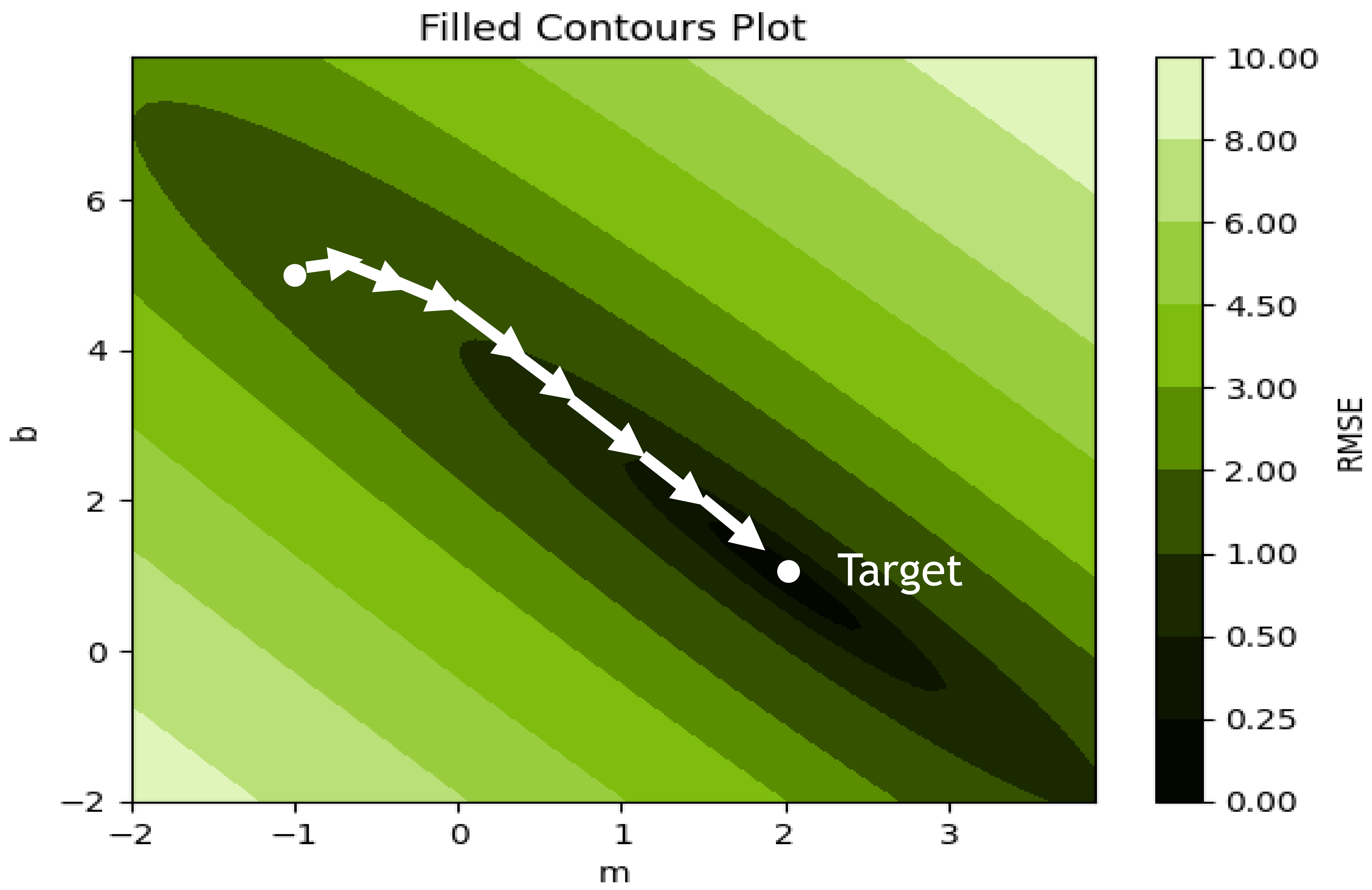
$$m = 0$$

$$b = 4$$



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



Definitions (I)

- **Parameters**: weights of the Neural Network. Their values are derived via training

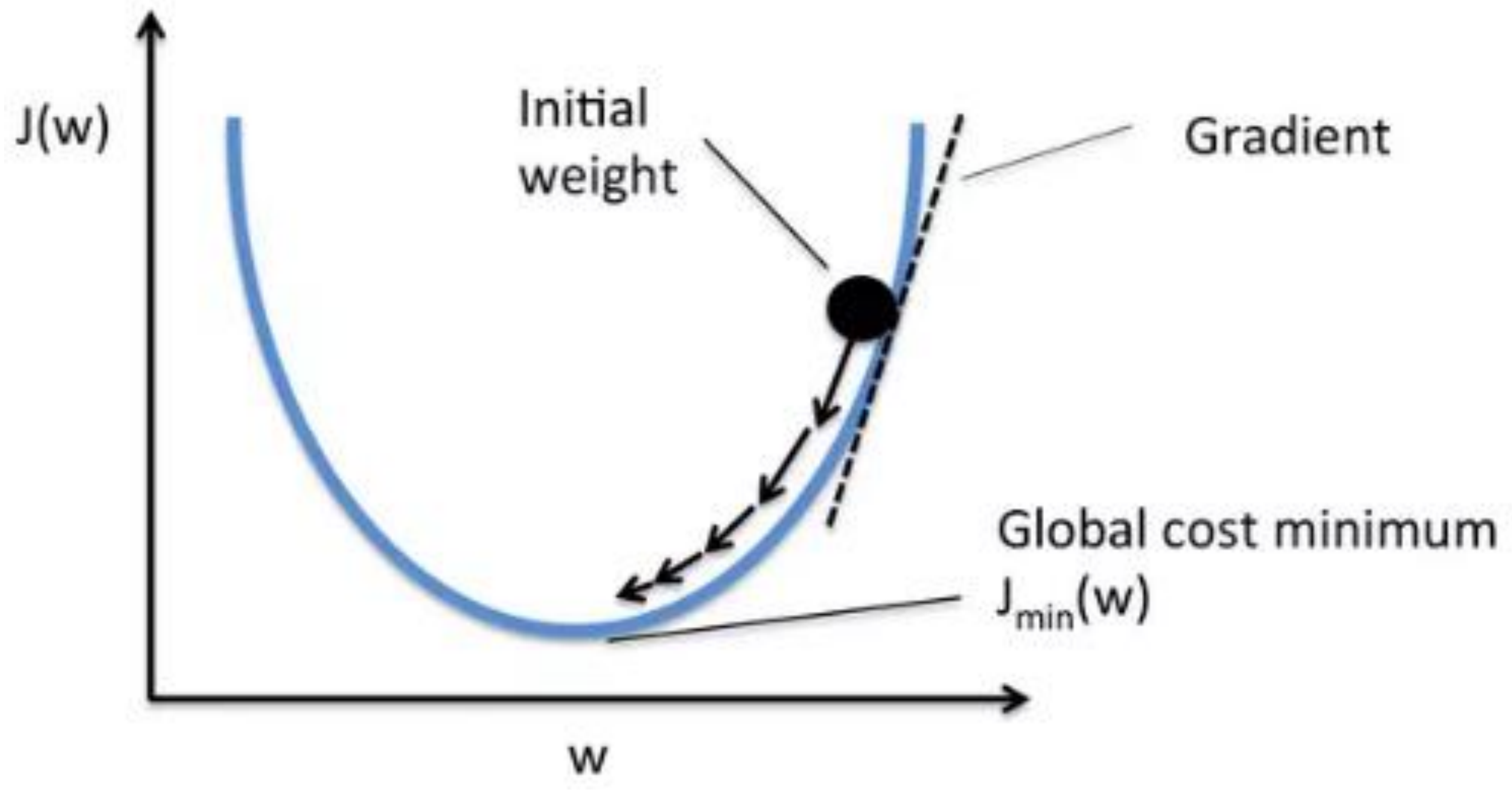
- **Hyper-parameters**: are values set before the **learning** process begins.

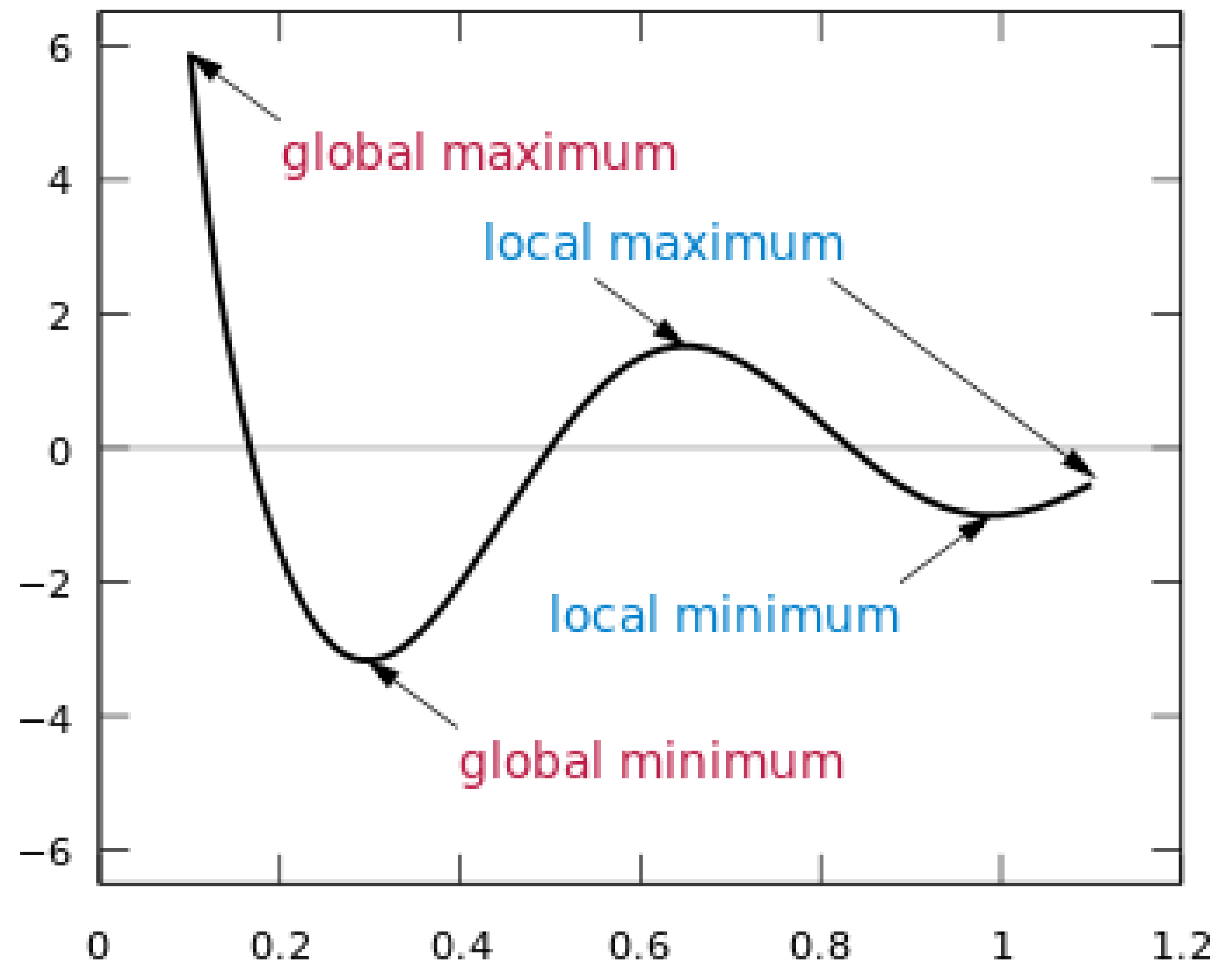
  Ex: number of epochs
  mini-batch size
  learning rate

Definitions (II)

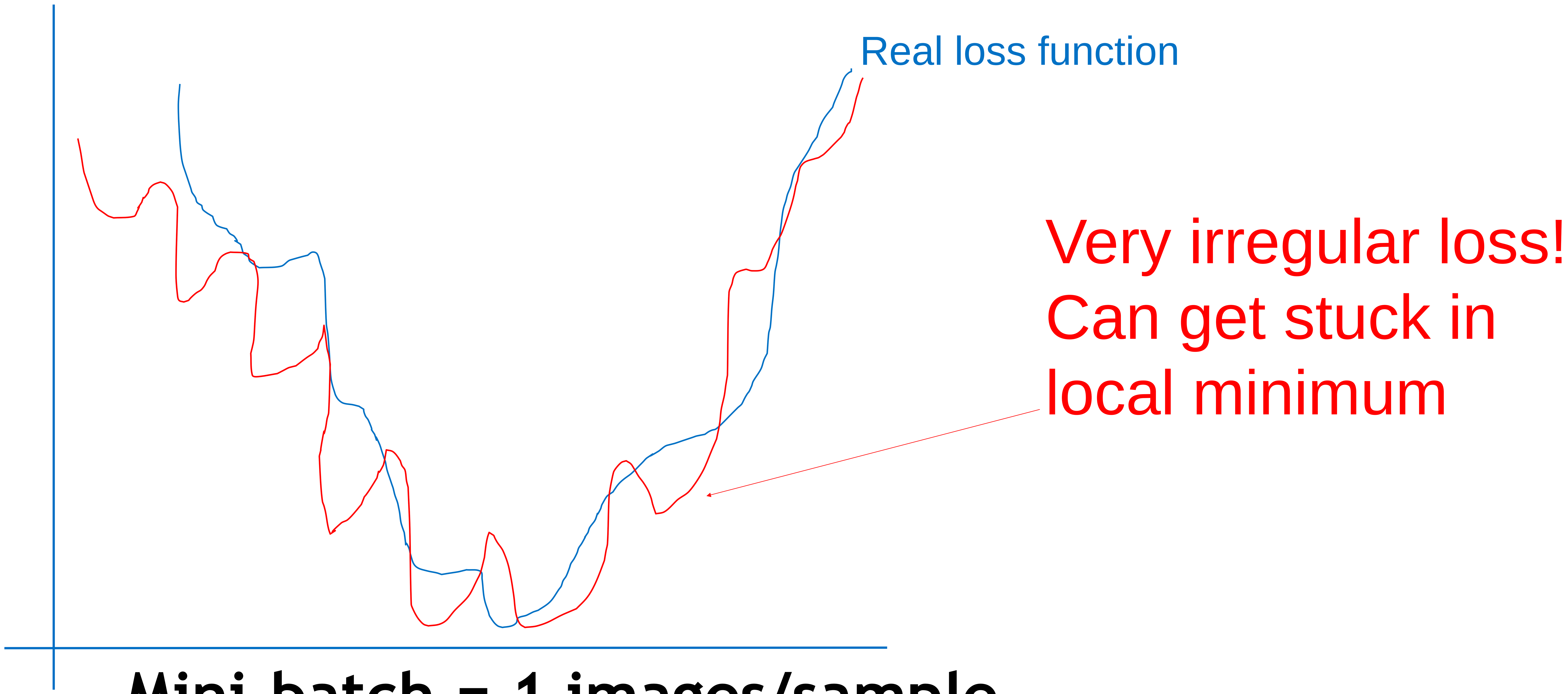
- **1 Iteration**: 1 Forward pass + 1 Backward pass
- **1 Epoch**: 1 Forward pass + 1 Backward pass for **ALL** training samples.
- **Batch Size**: Number of training samples
- **Mini-batch (<u>often called Batch Size!</u>)**: Number of training samples used in 1 iteration
- **Learning rate**: determines to what extent newly acquired information overrides old information
- **Loss:** difference between prediction and correct output
- **Accuracy:** $\dfrac{\text{number of correct predictions}}{\text{total number of predictions}}$

Non convex Cost functions





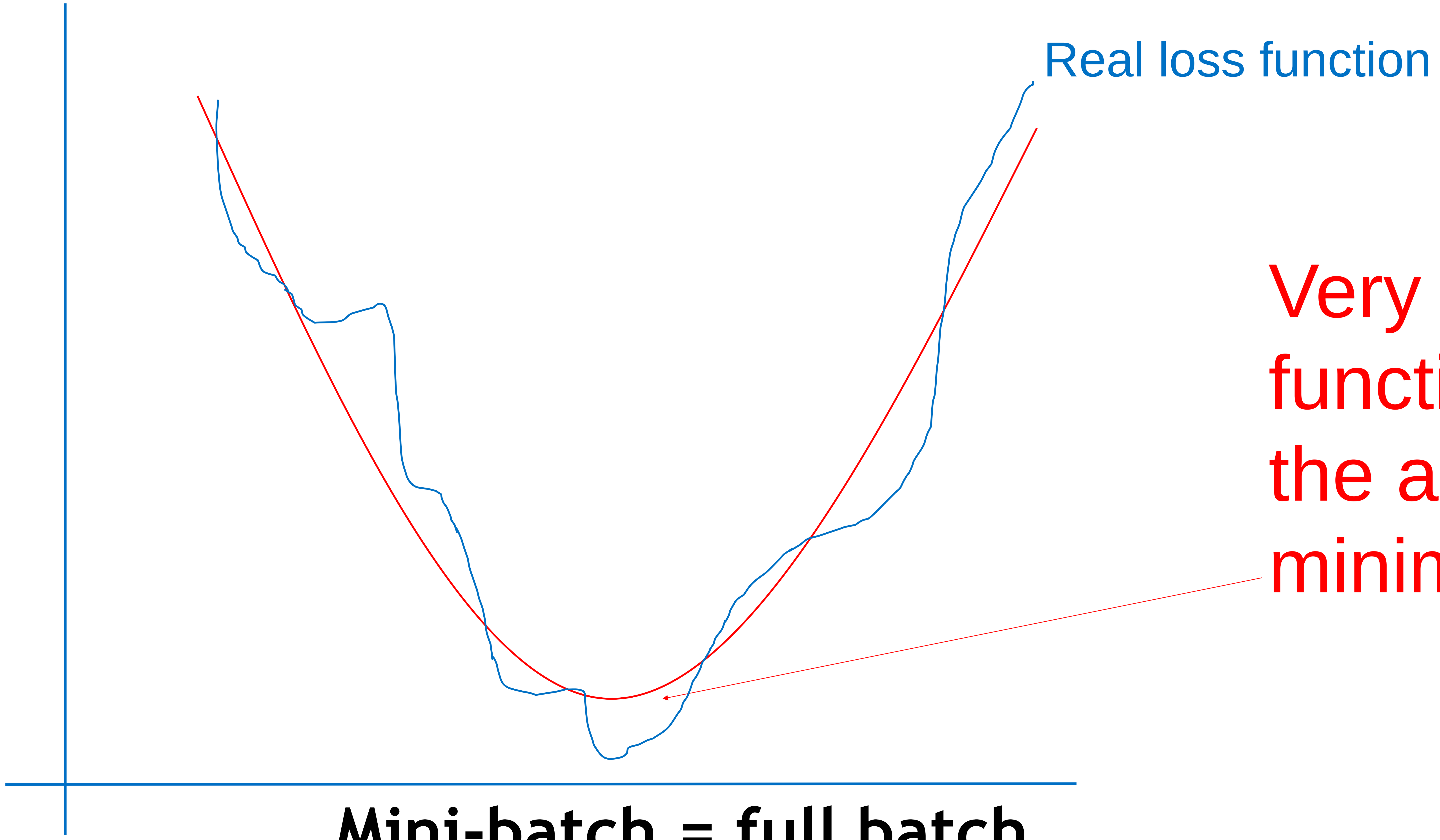
Stochastic Gradient Descent (SGD) method



Mini-batch = 1 images/sample

N iterations per epochs = n samples in the batch

Batch Gradient Descent (BGD) method

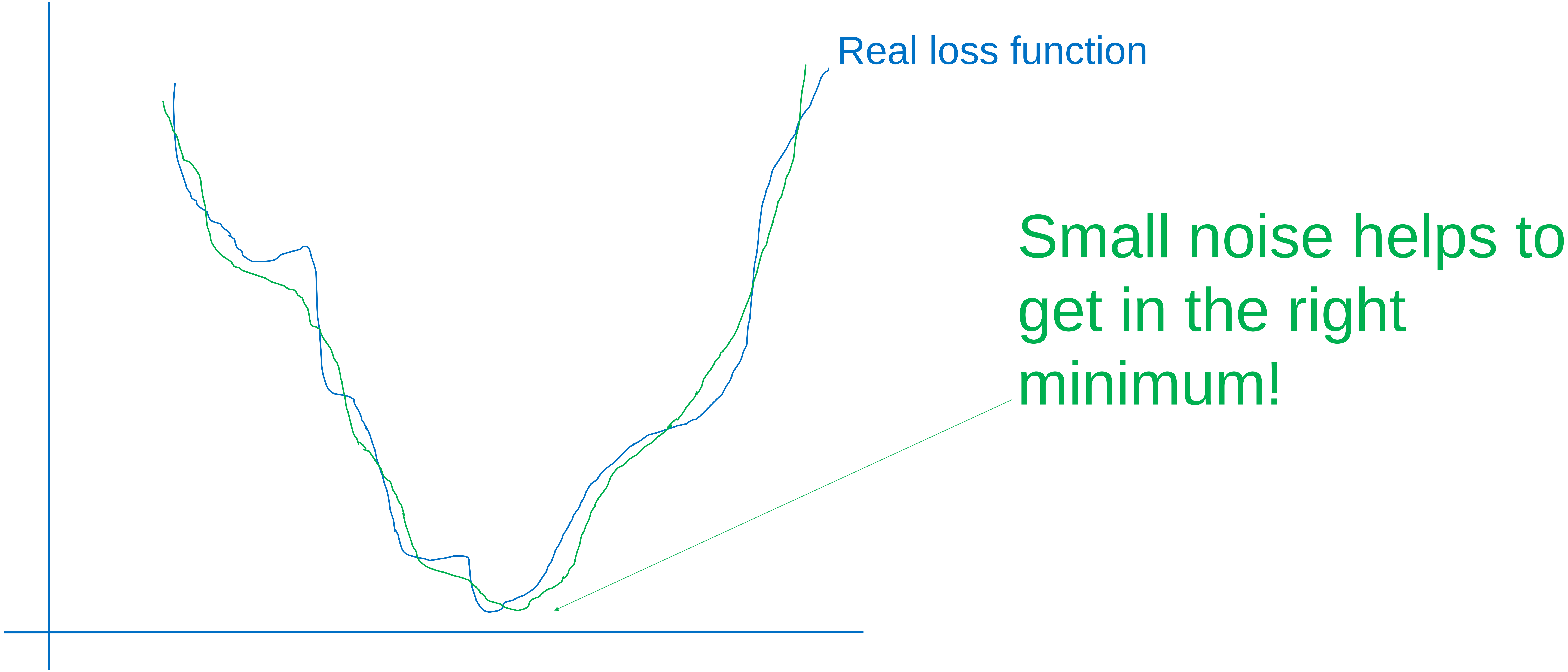


Very smooth loss function! Can kill the actual minimum

Mini-batch = full batch

1 iteration per epochs = 1

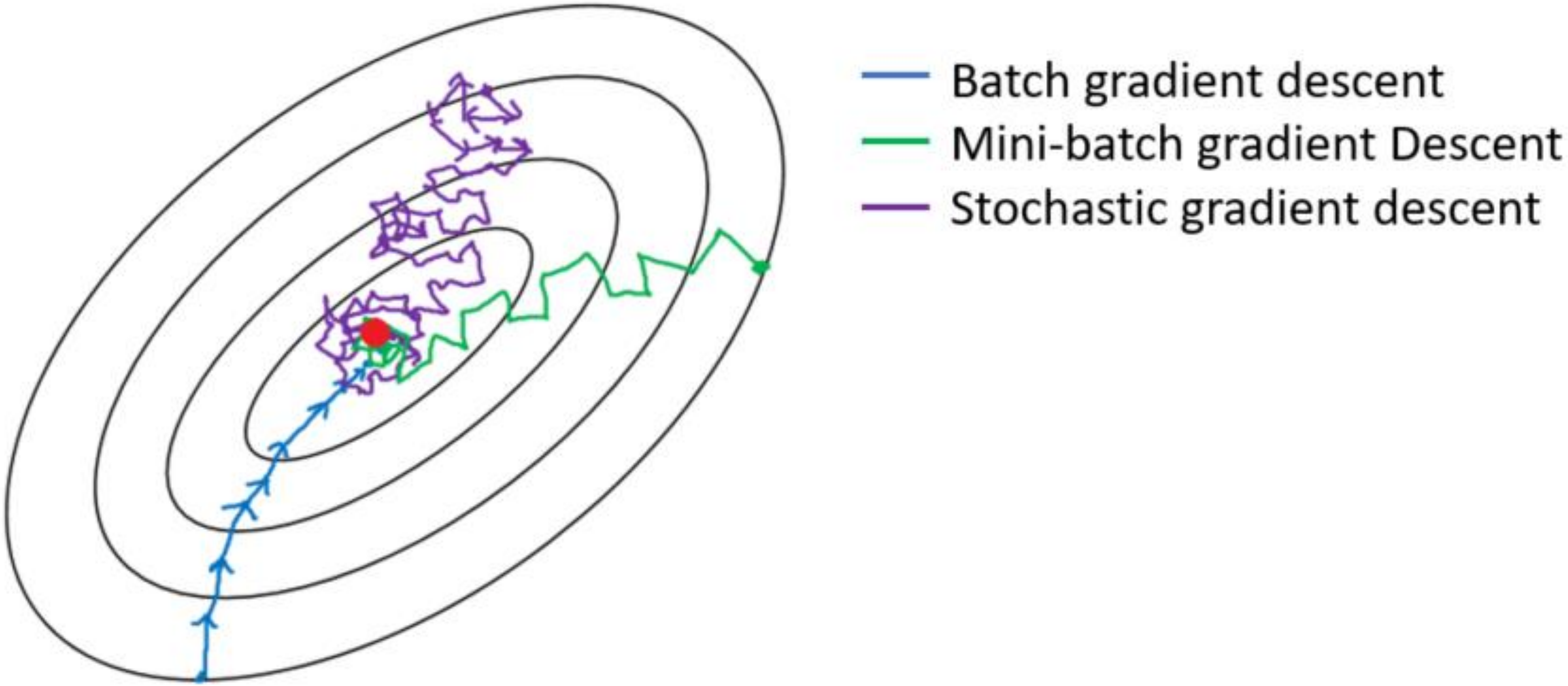
Mini-batch Stochastic Gradient Descent (mSGD) method



The mini-batch is equal to a subset of the batch size (usually 32/64/128…)

N iterations per epochs = N samples in the batch / mini-batch size

Effect of different mini-batch size

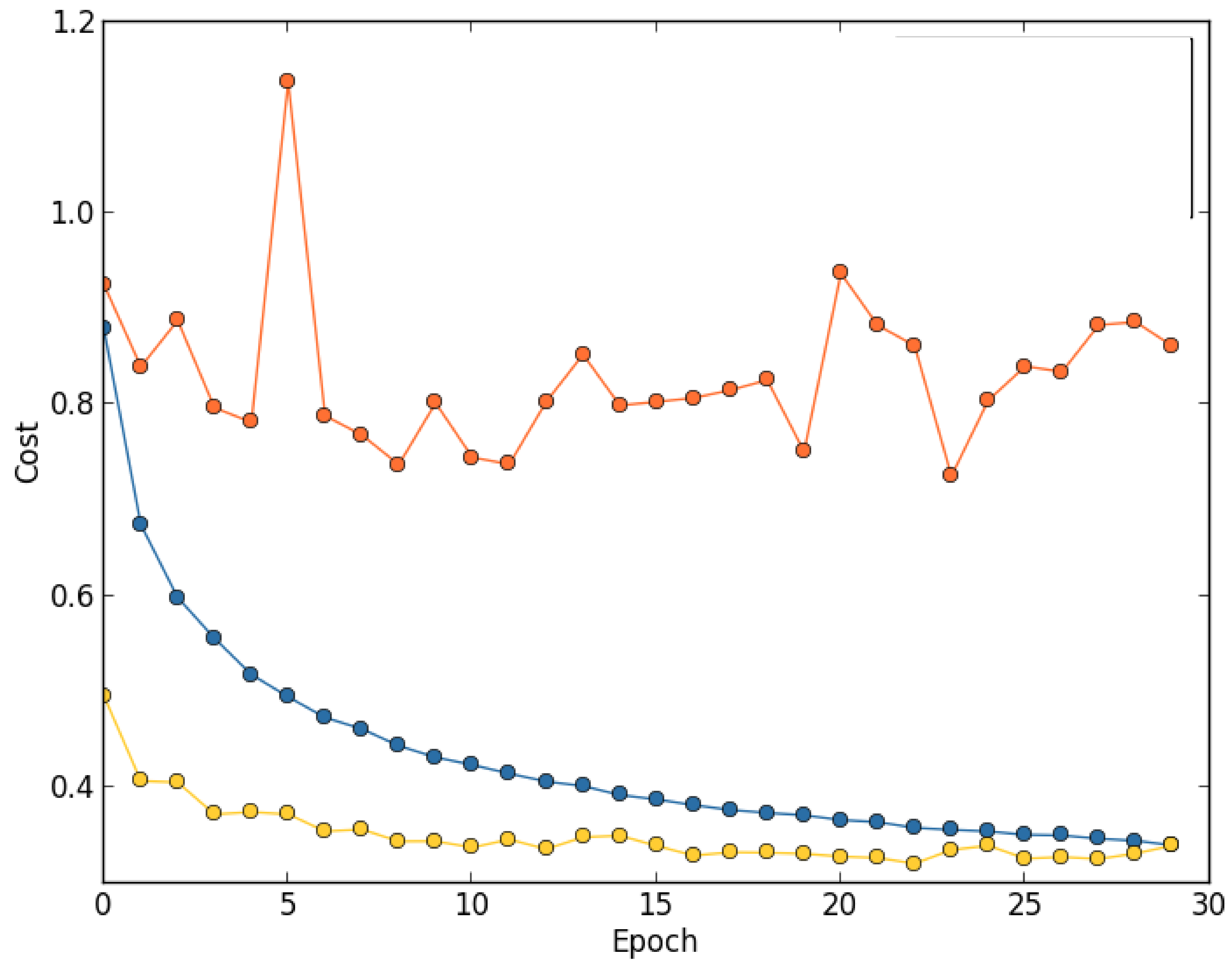


Which curve has the largest batch size?

Which curve has the largest batch size?

Which curve has the smallest learning rate?

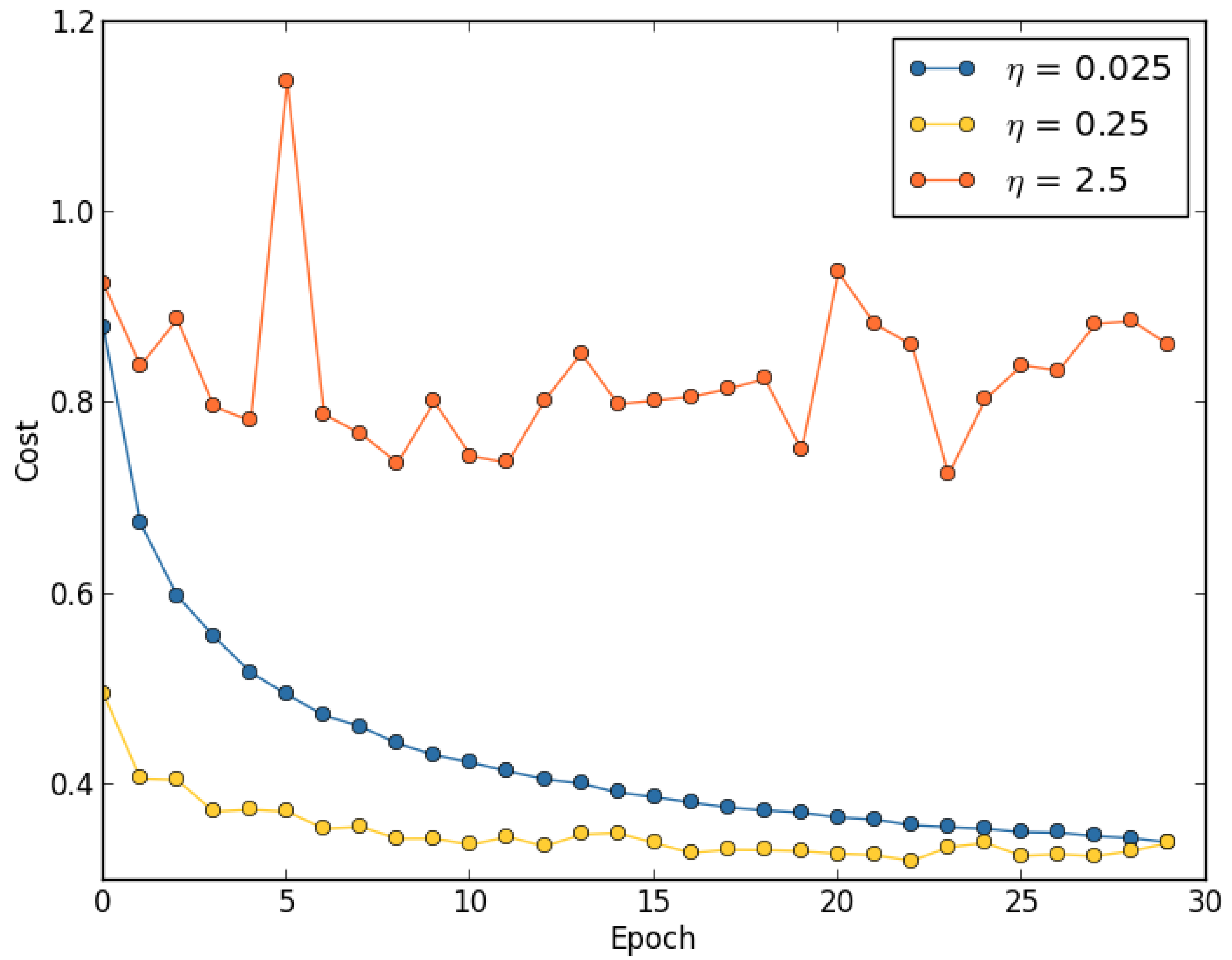
$$w_t = w_{t-1} + \eta \frac{dE}{dw_t}$$



NVIDIA DEEP LEARNING INSTITUTE

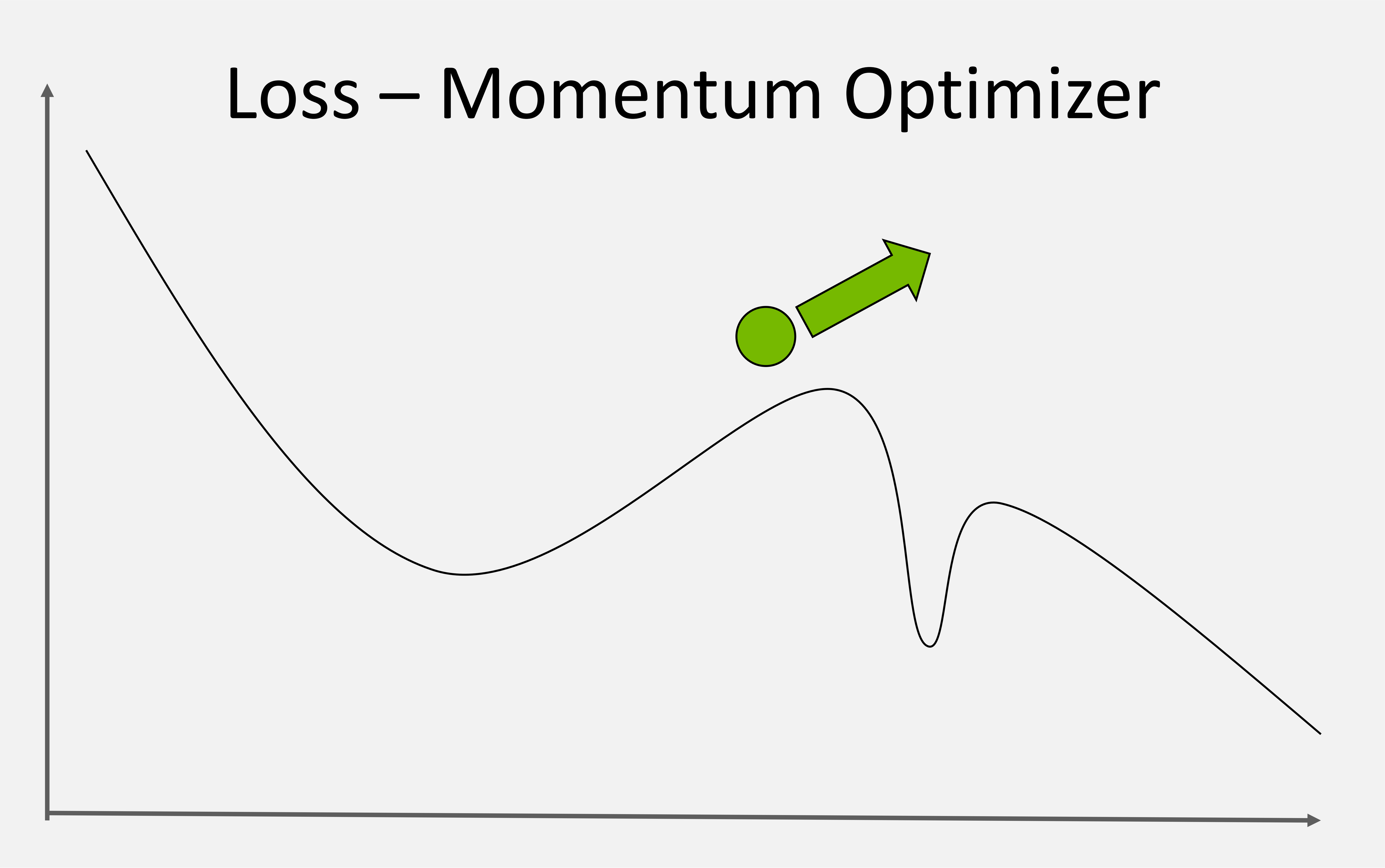
Which curve has the smallest learning rate?

$$w_t = w_{t-1} + \eta \frac{dE}{dw_t}$$



NVIDIA DEEP LEARNING INSTITUTE

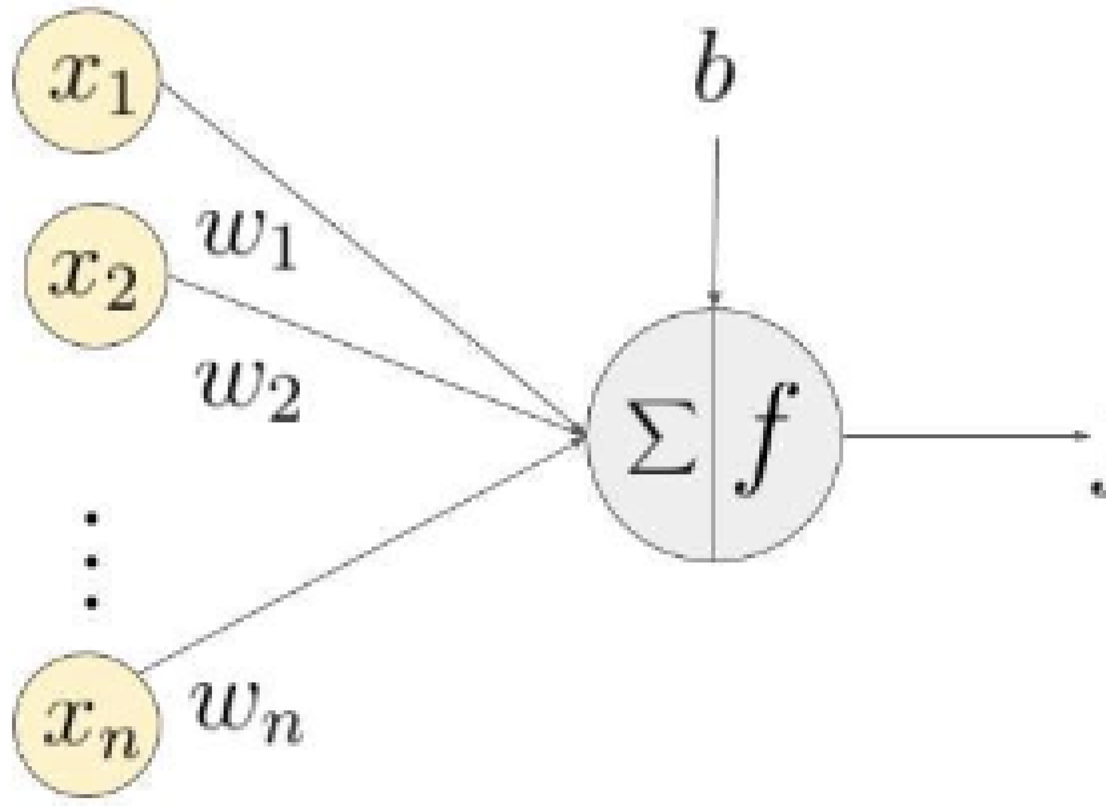
Optimizers



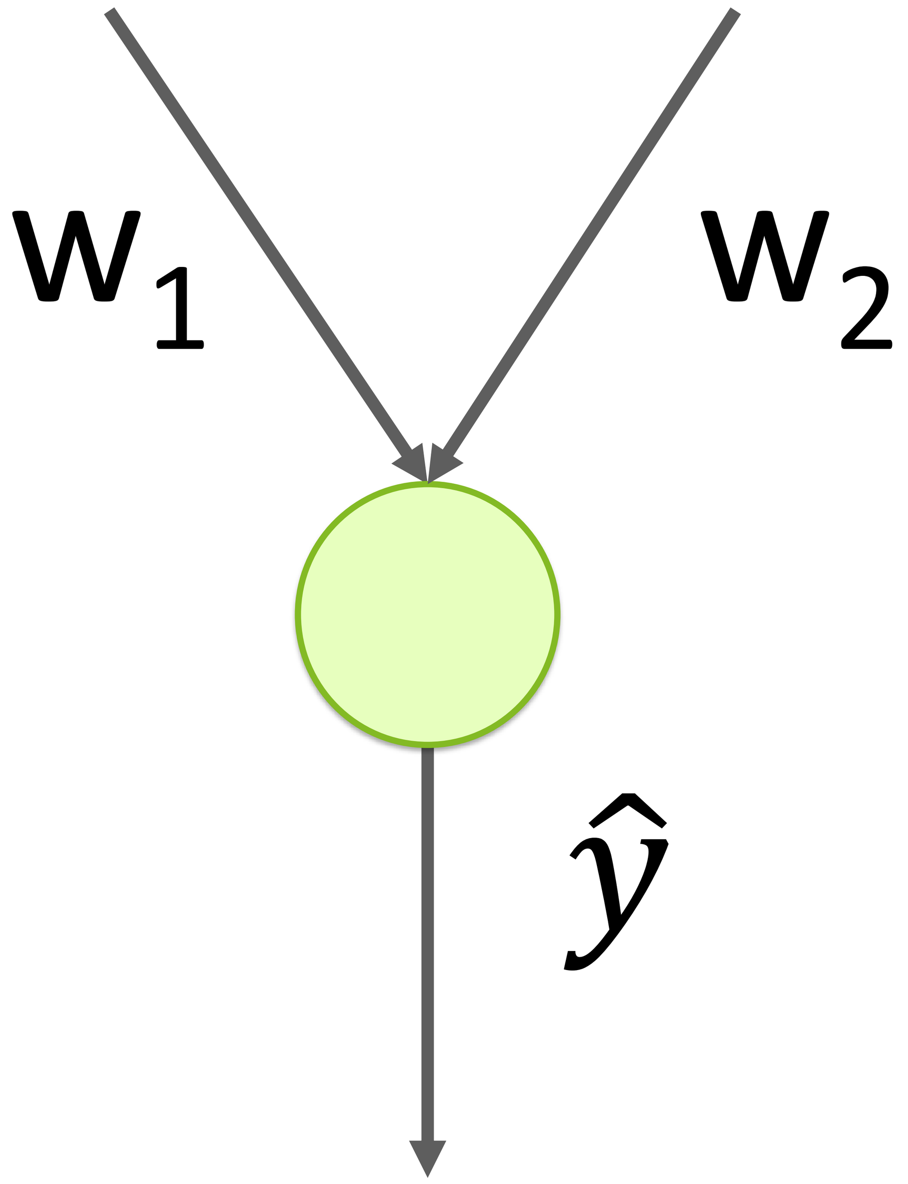
- Adam
- Adagrad
- RMSprop
- SGD

From Neuron to Network

Building a Network

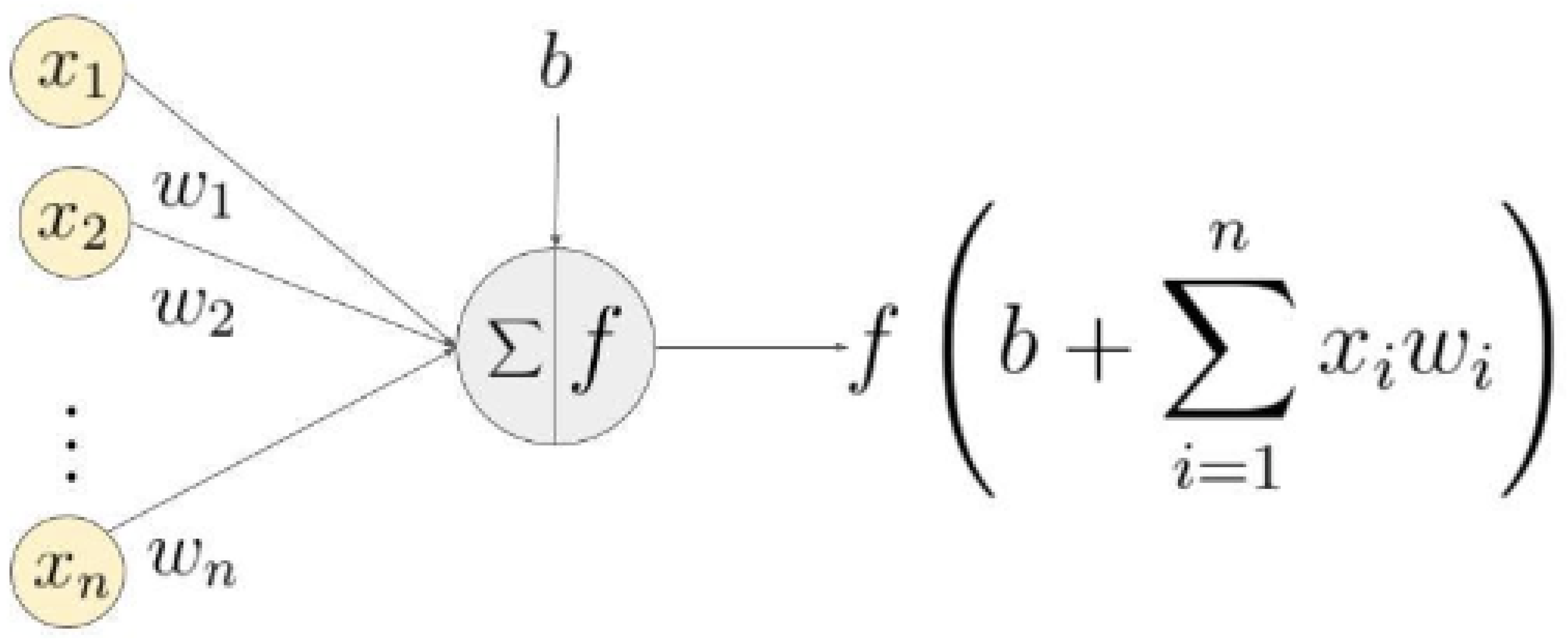


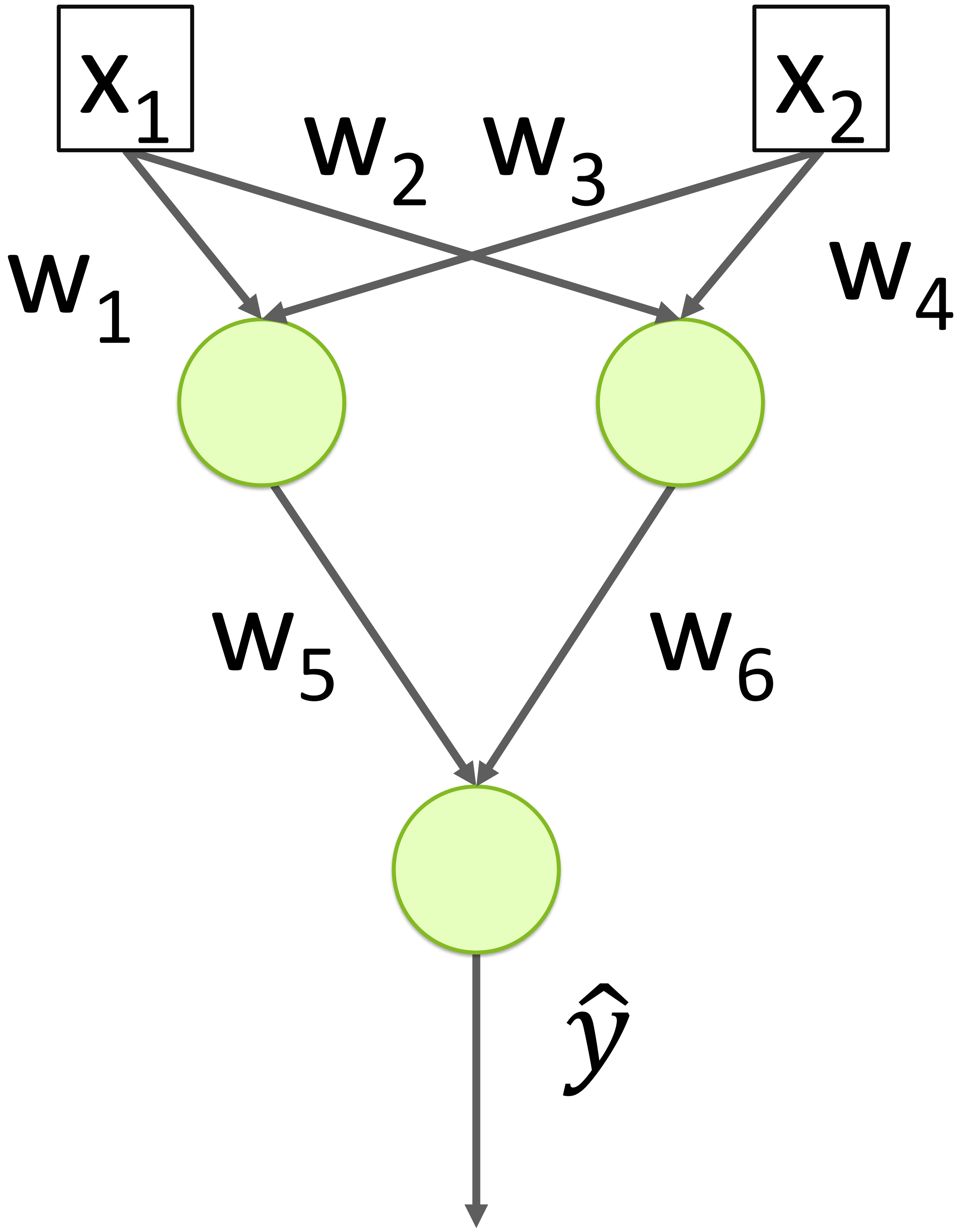
$$f\left(b+\sum_{i=1}^{n} x_i w_i\right)$$



- Scales to more inputs

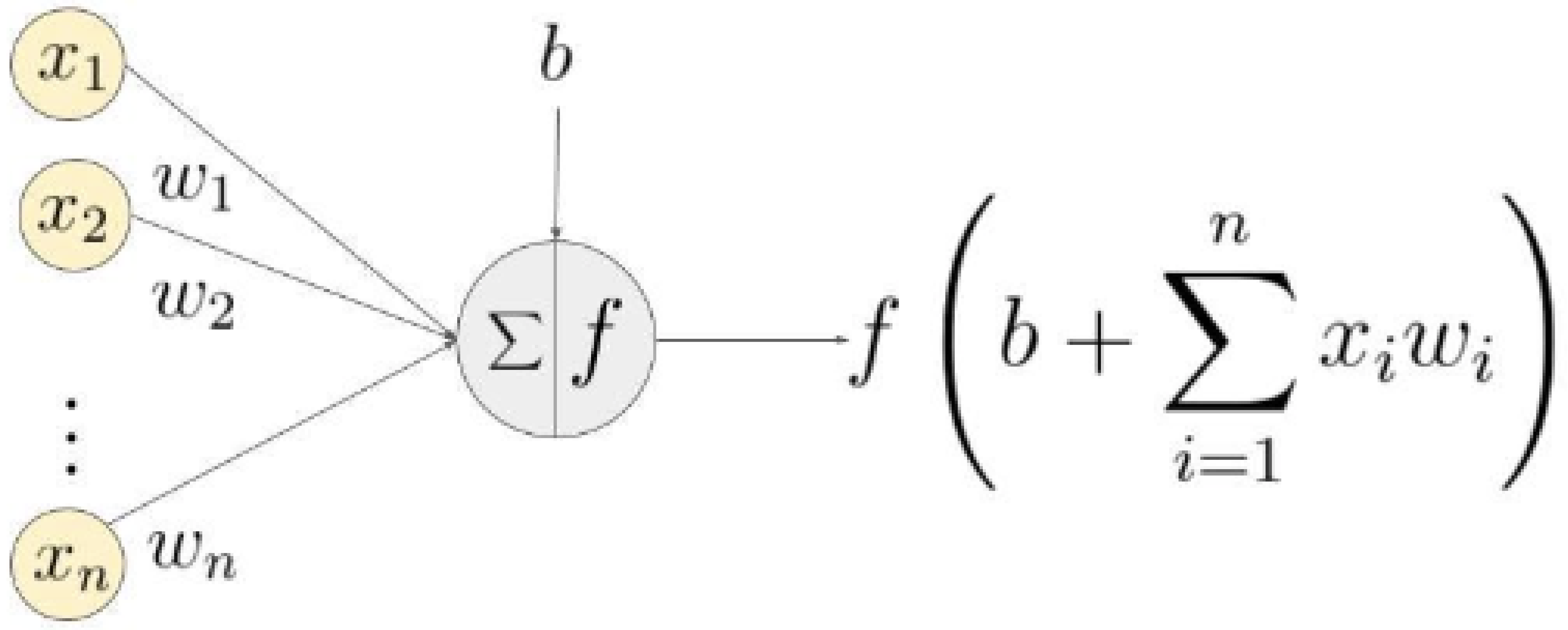
Building a Network

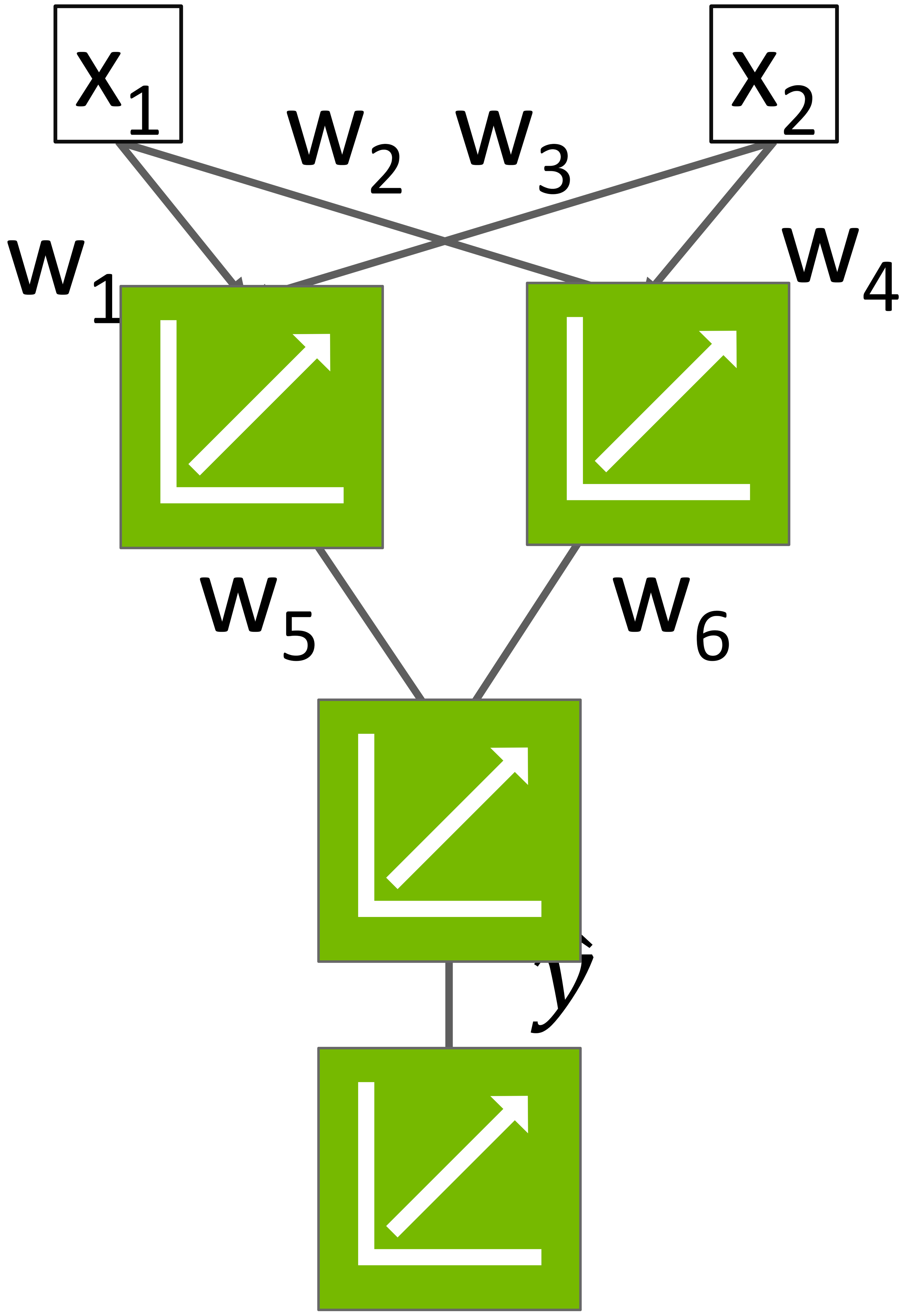




- Scales to more inputs
- Can chain neurons

Building a Network





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

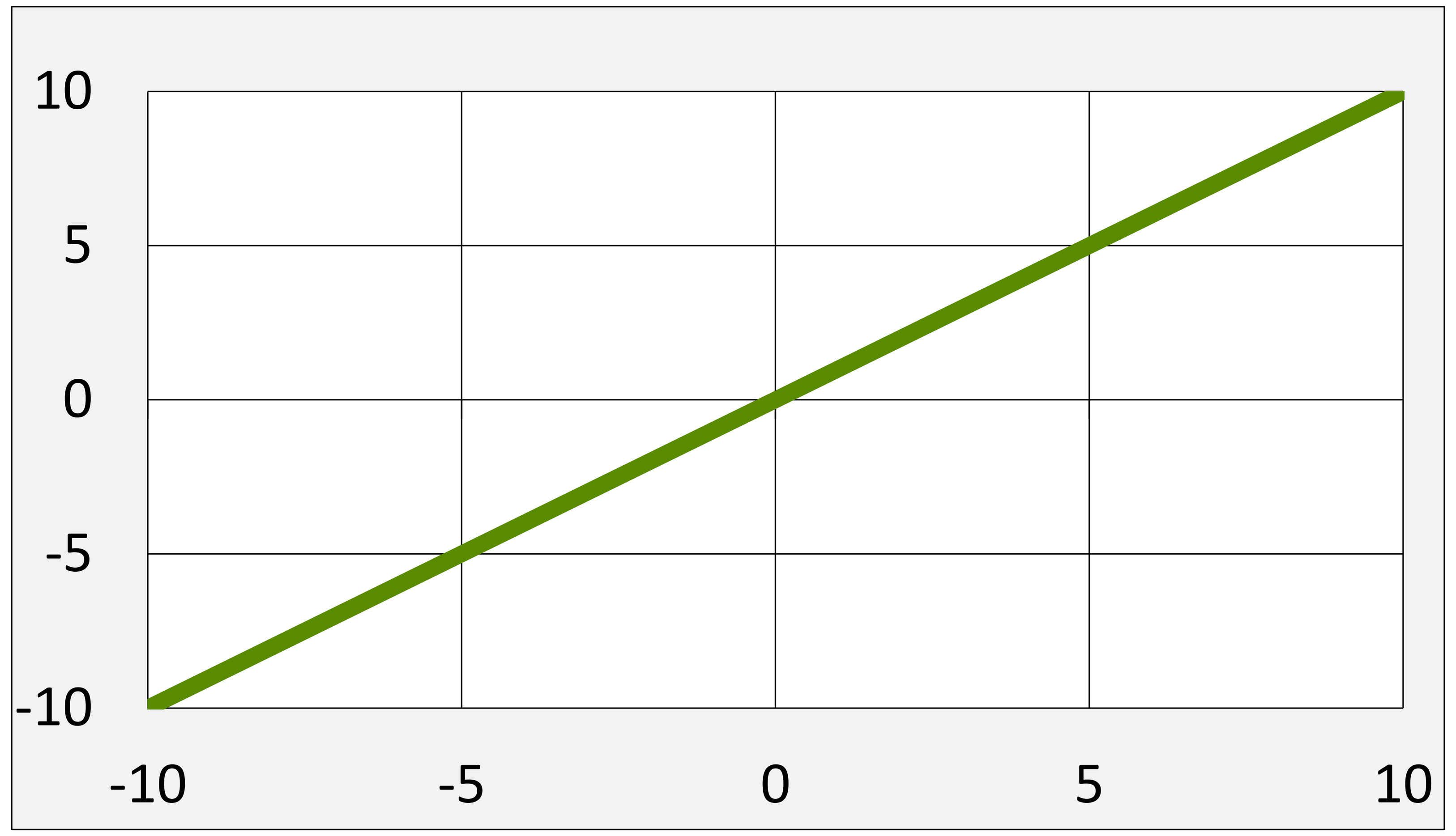
Activation Functions

Activation Functions

Linear

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

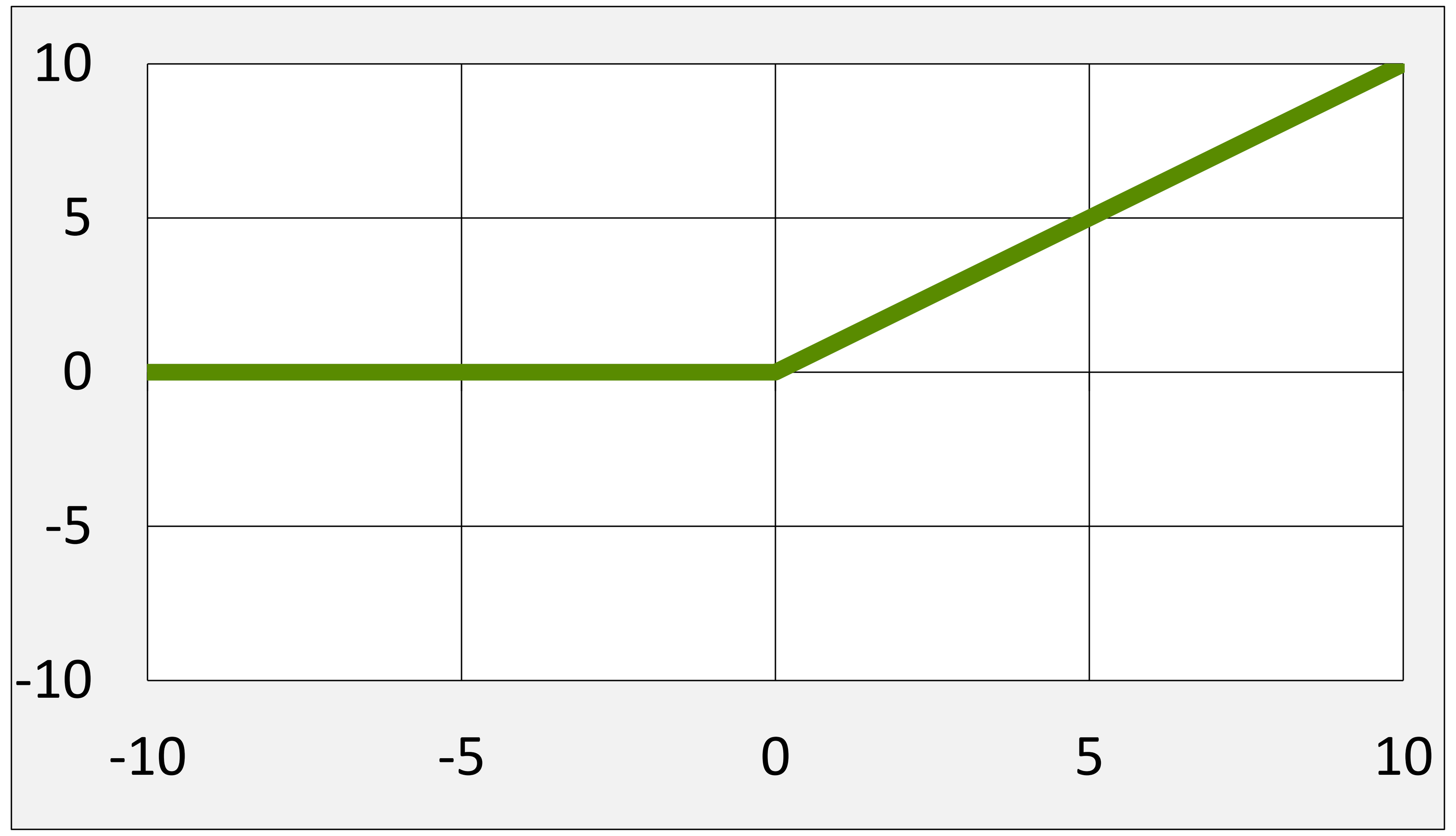
```
# Multiply each input
# with a weight (w) and
# add intercept (b)
y_hat = wx+b
```



ReLU

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

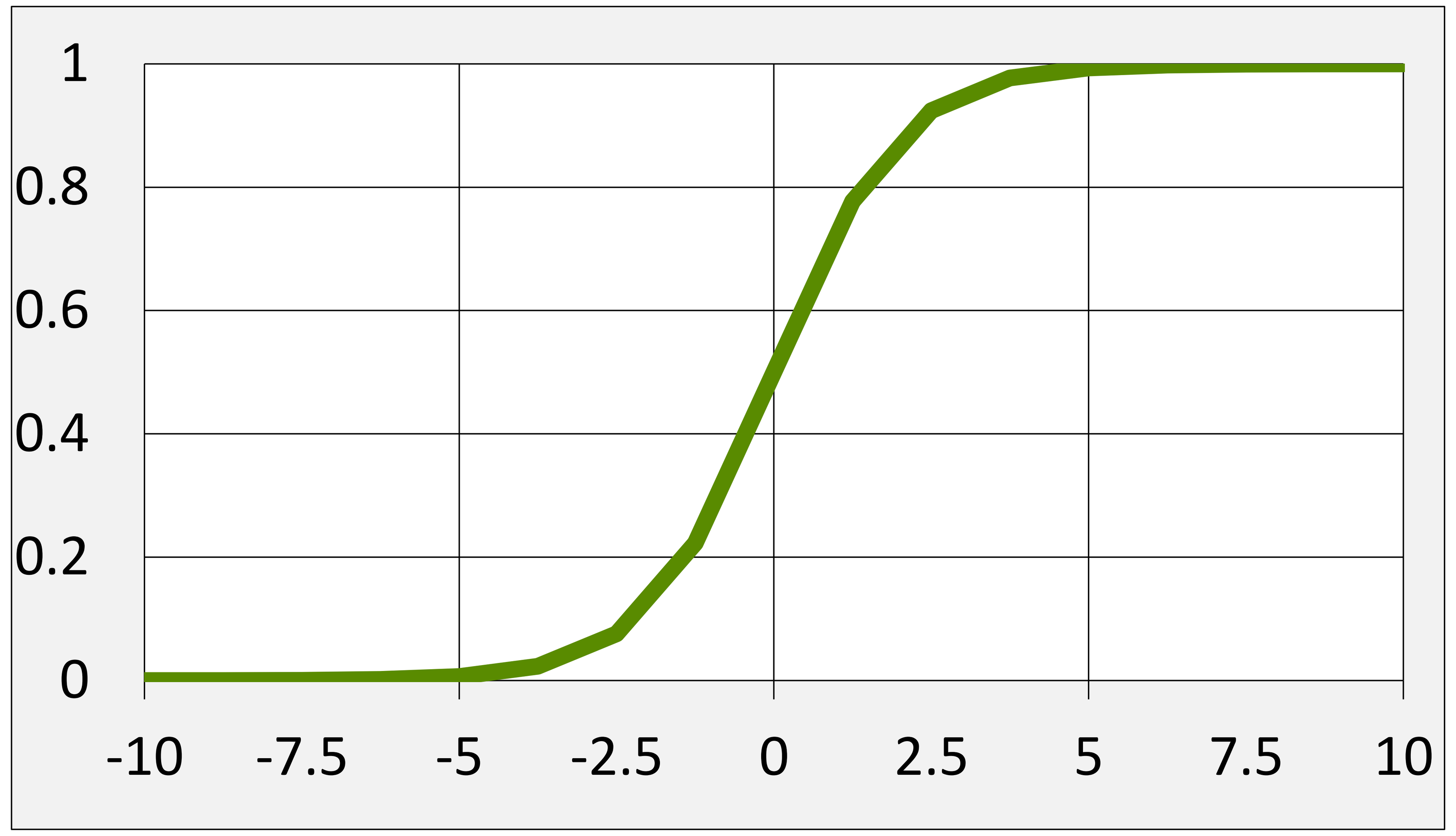
```
# Only return result
# if total is positive
linear = wx+b
y_hat = linear * (linear > 0)
```



Sigmoid

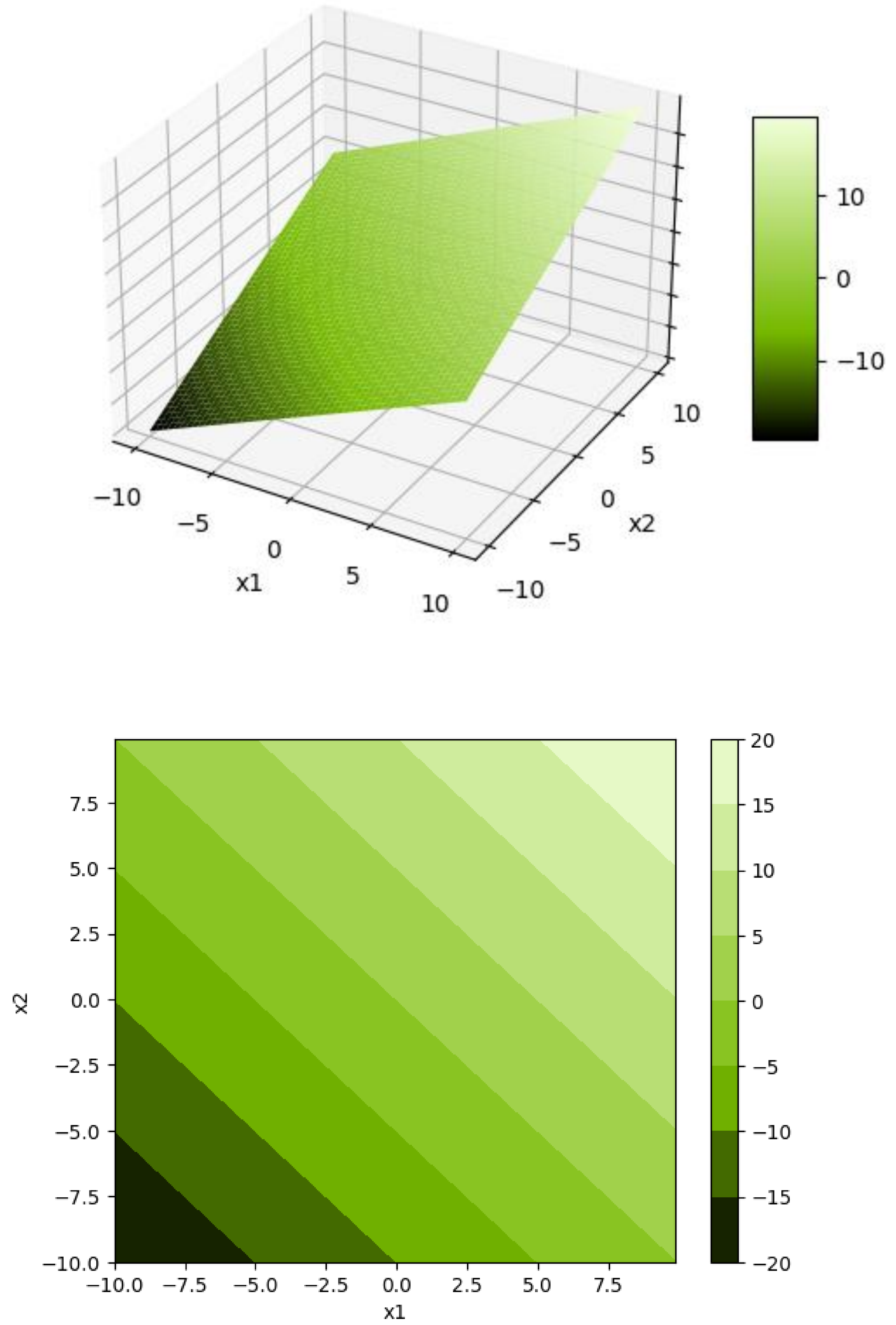
$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

```
# 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)
```

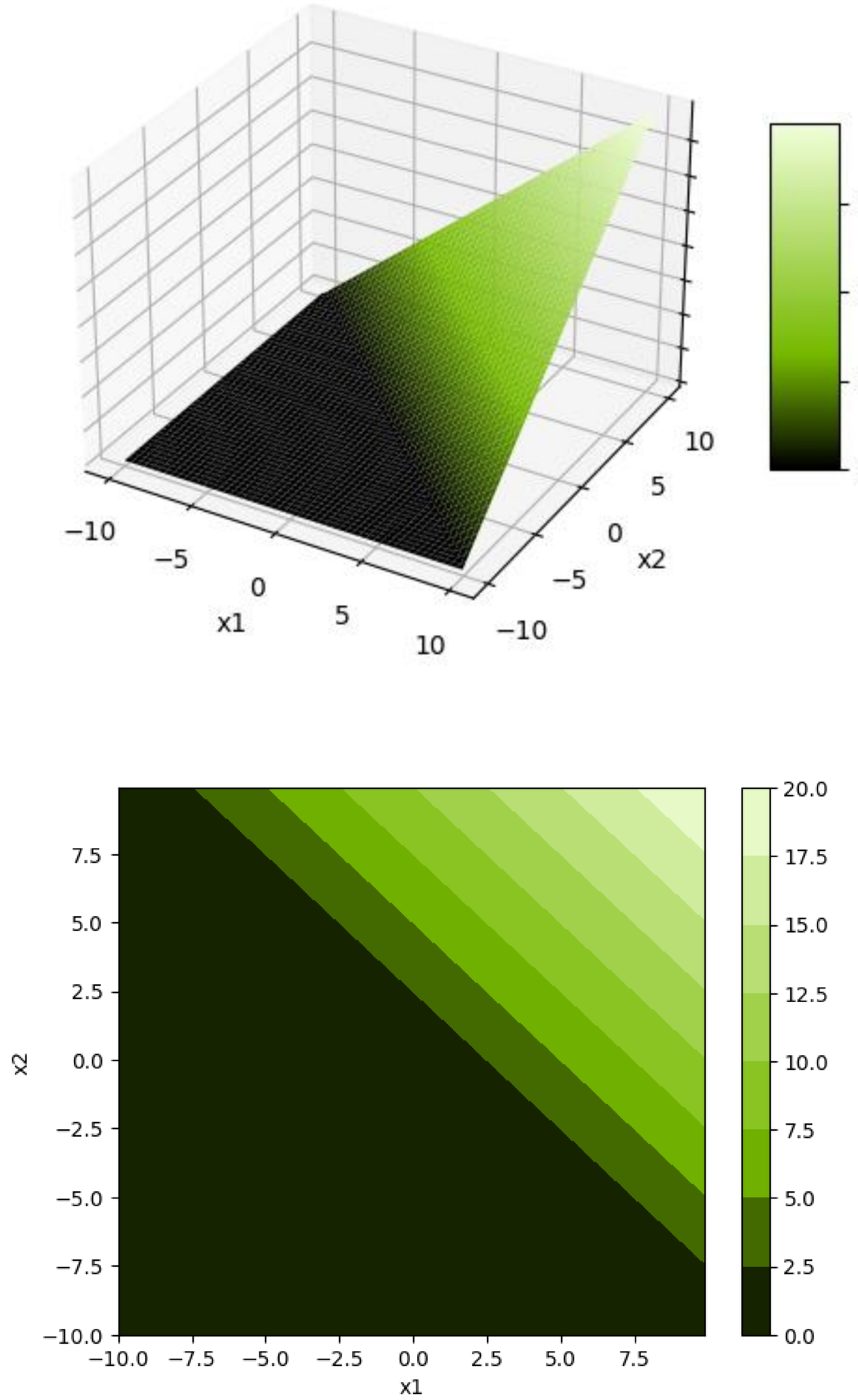


Activation Functions

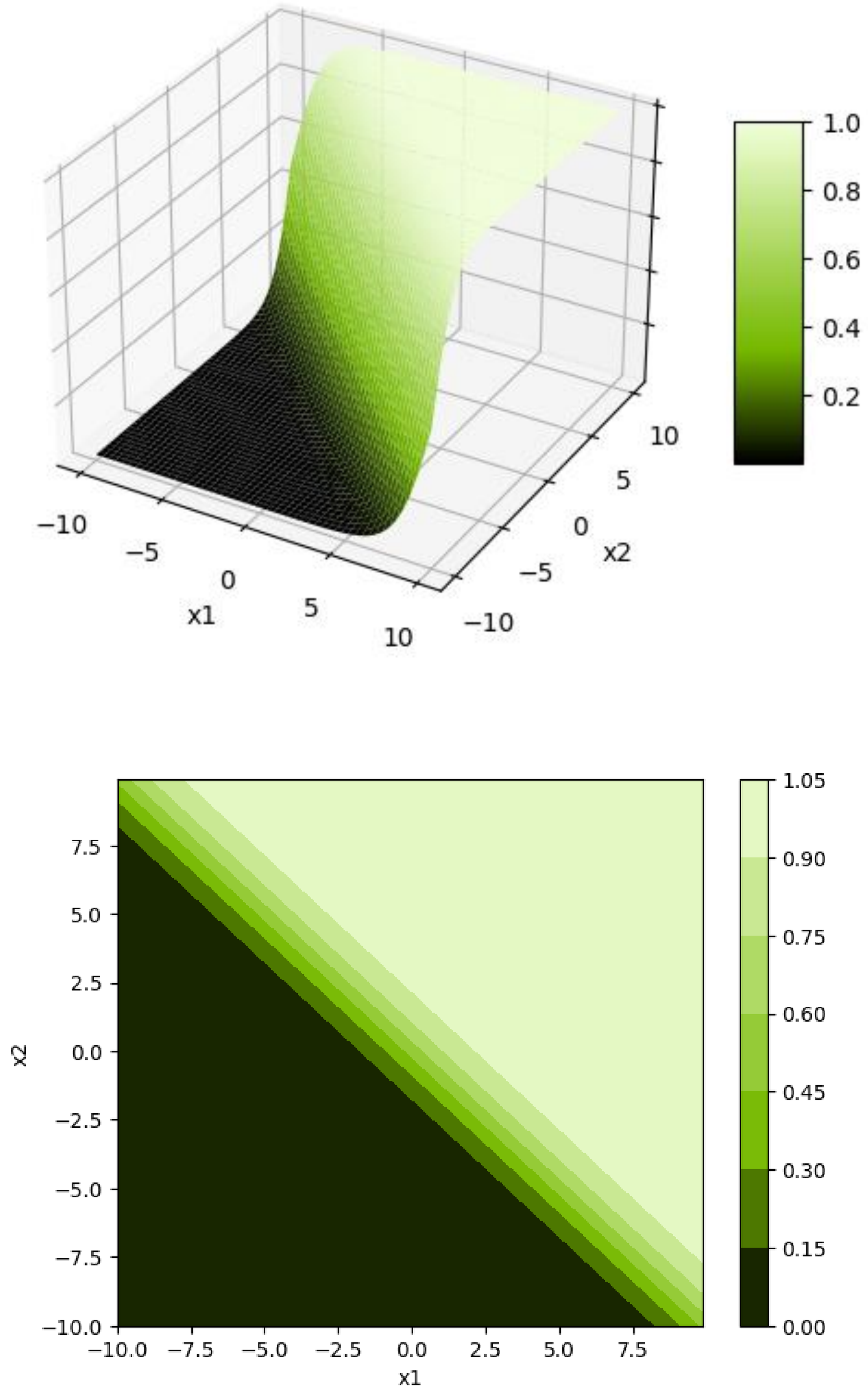
Linear



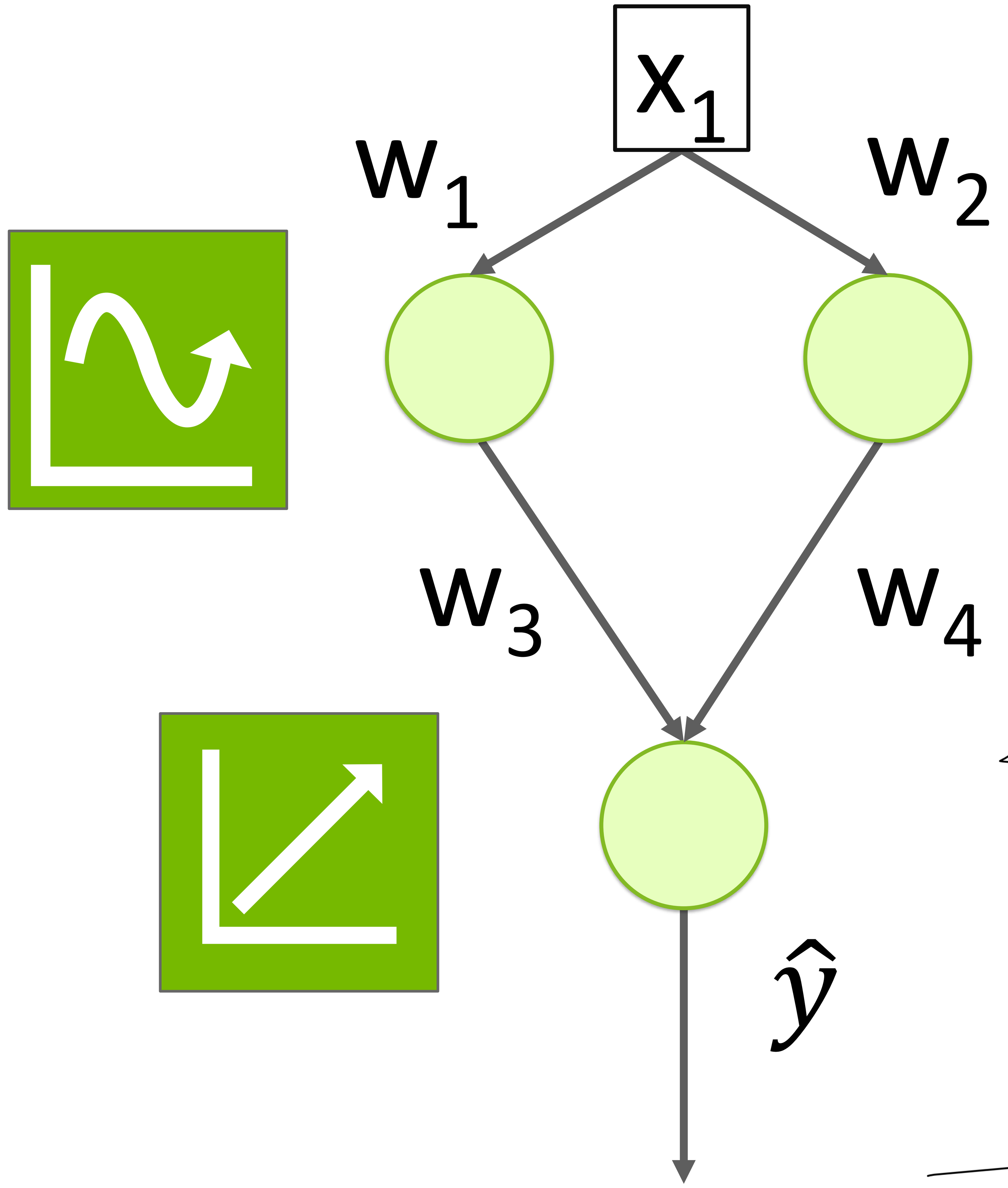
ReLU

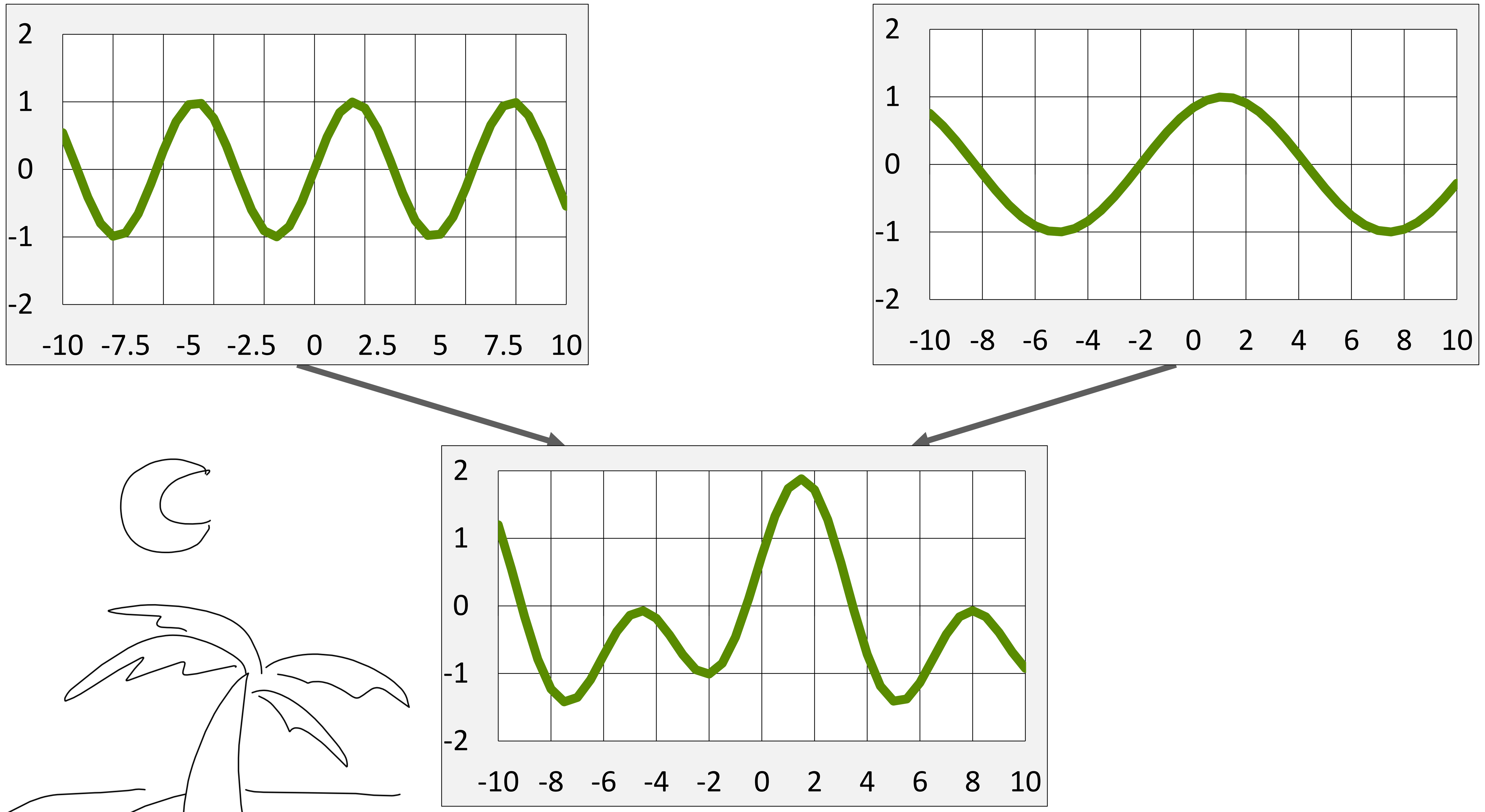


Sigmoid

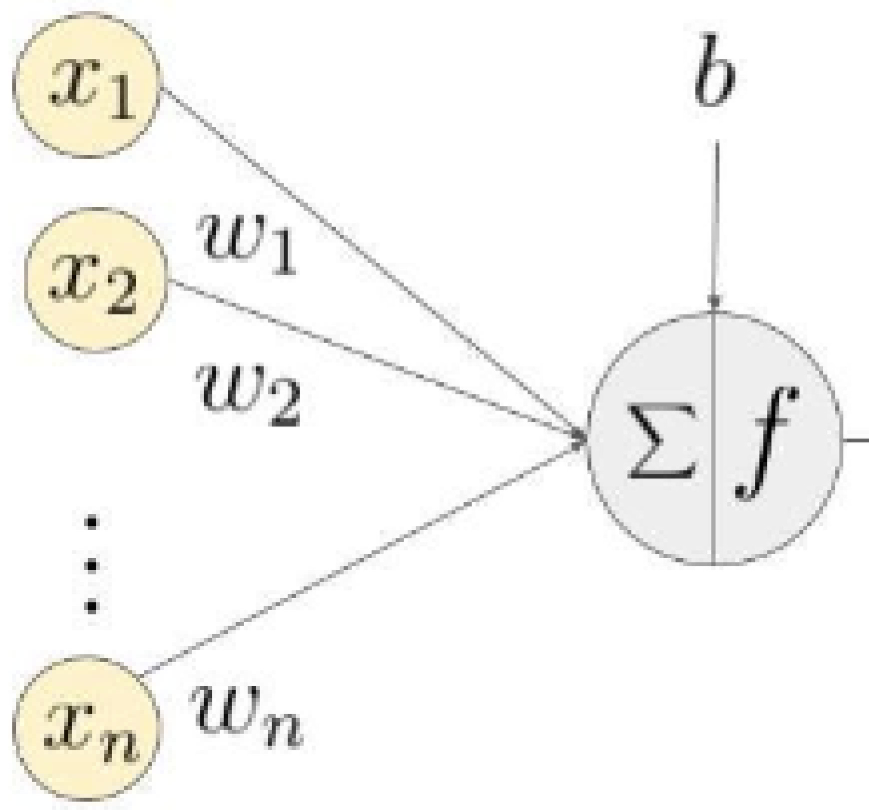


Activation Functions

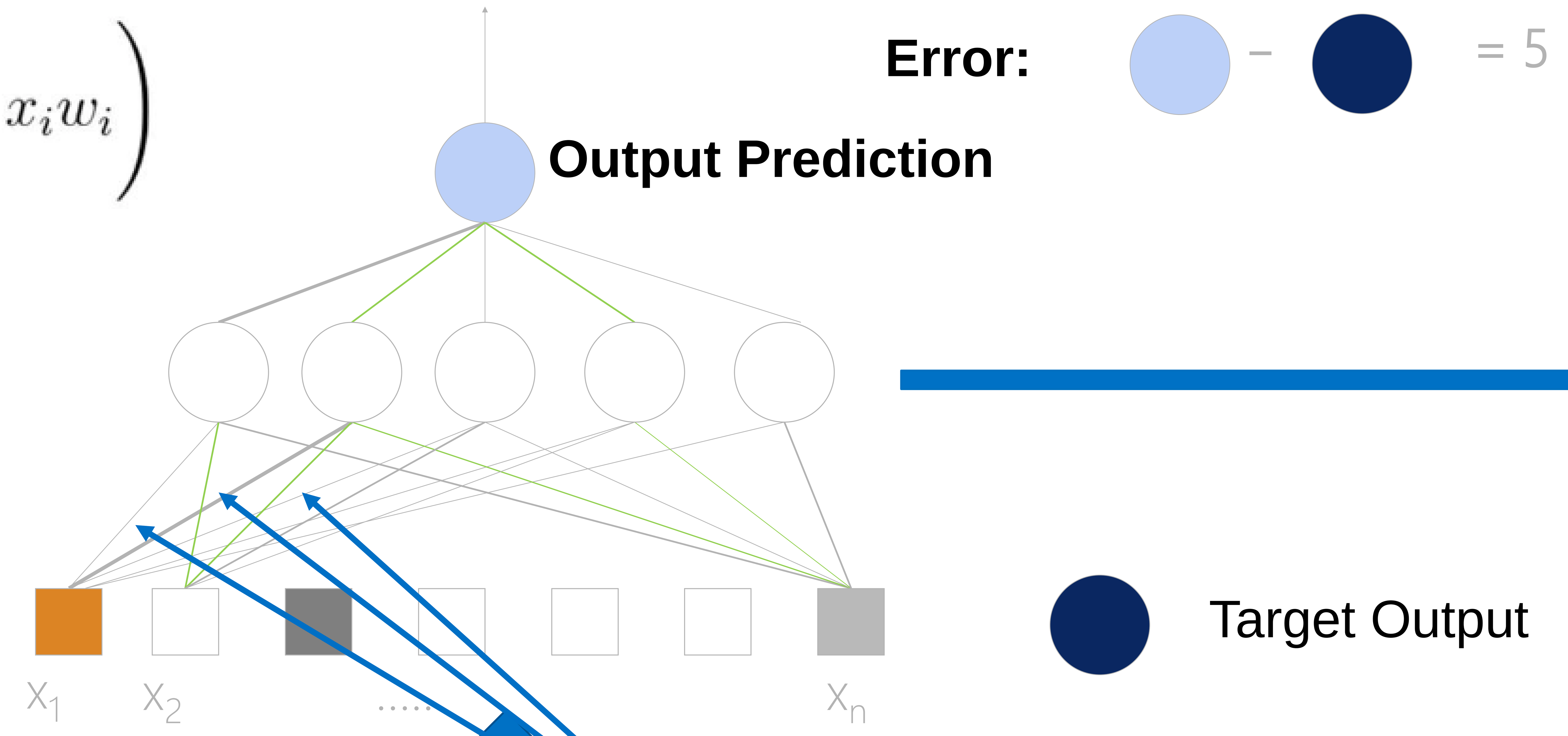




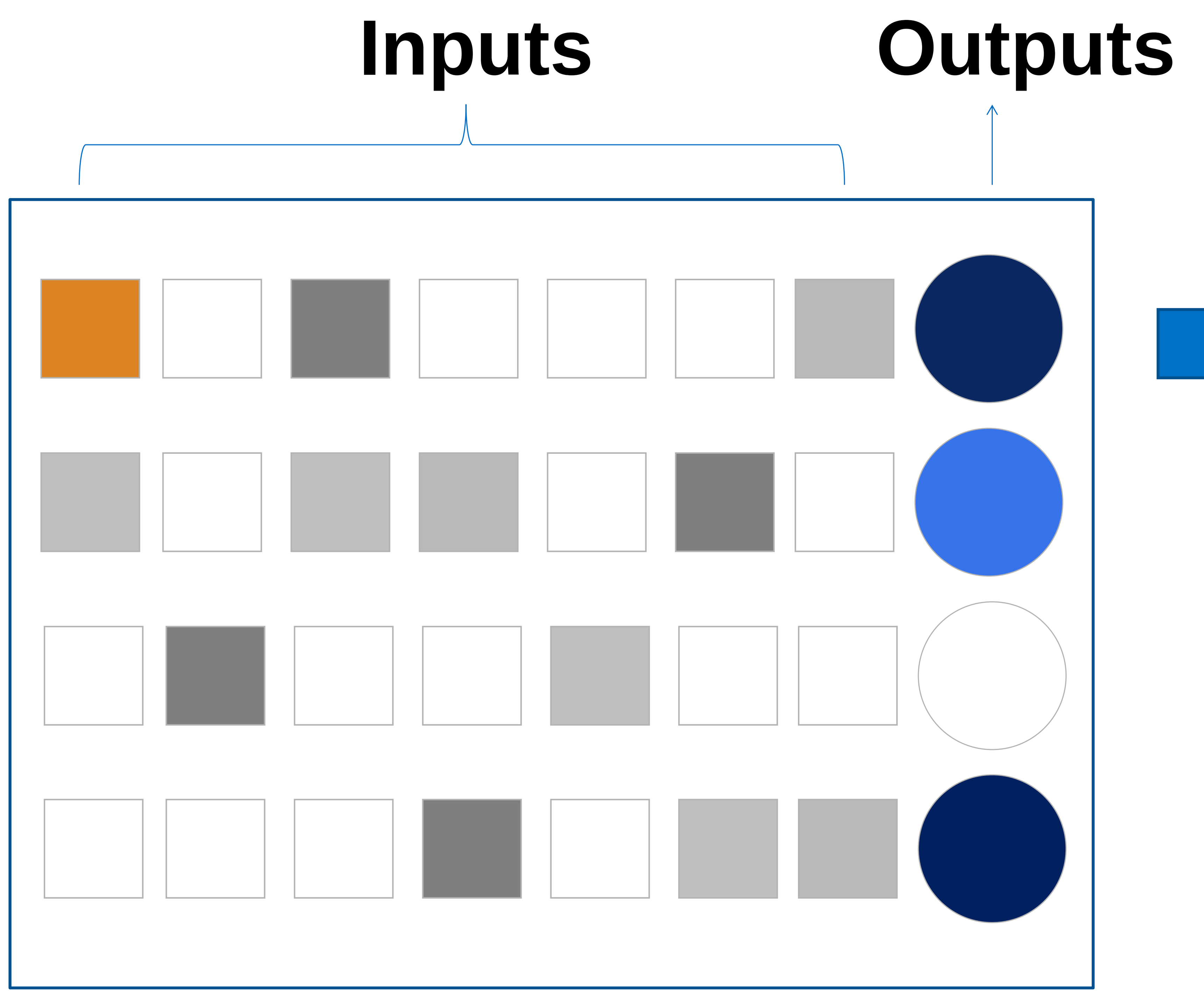
Learning Principle



$$f\left(b+\sum_{i=1}^{n} x_i w_i\right)$$



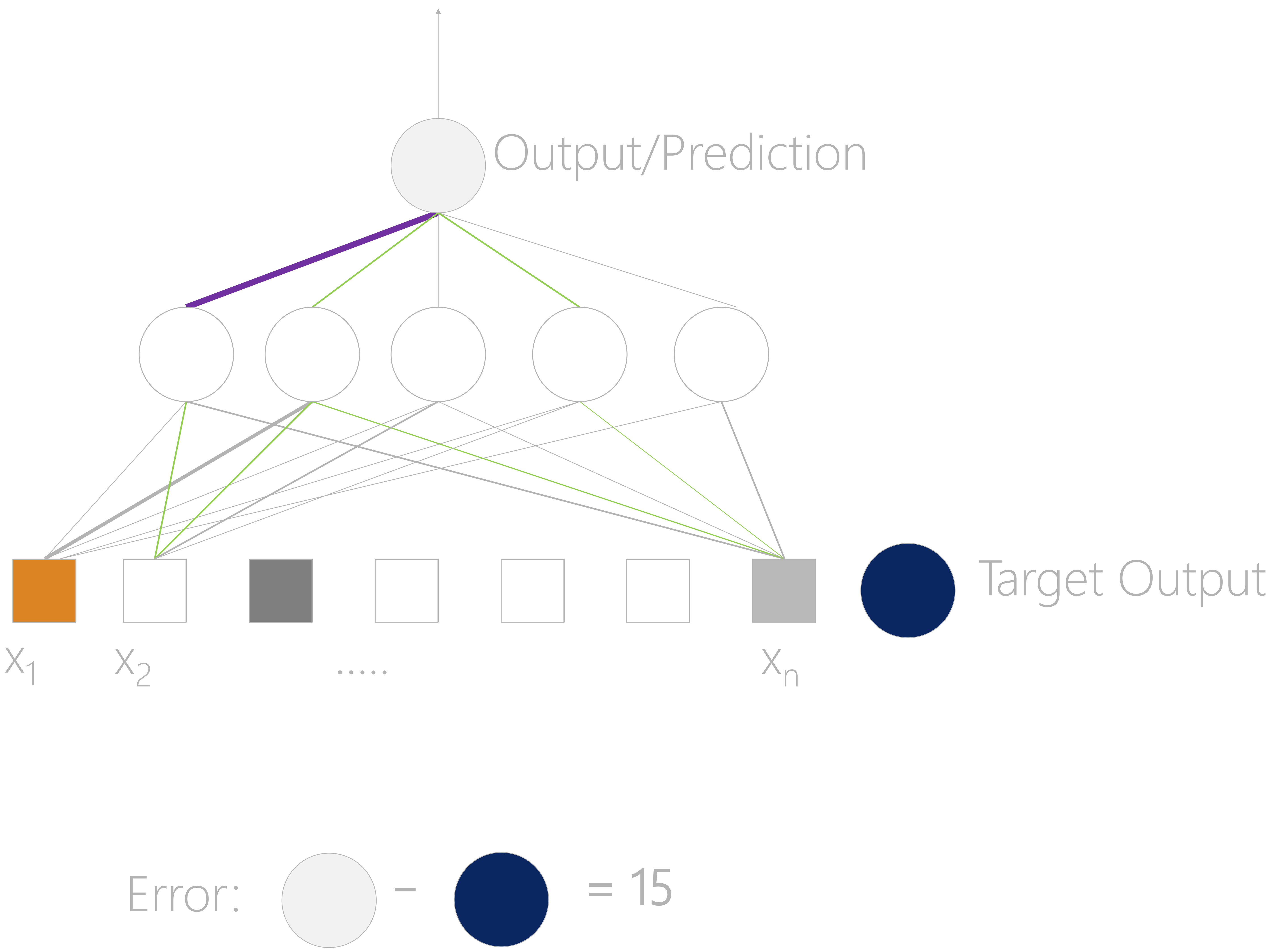
Dataset



Each of these lines is a weight! So, if I say a dense layer with N inputs and M outputs (neurons) the number of weights is: NxM + M. For example:

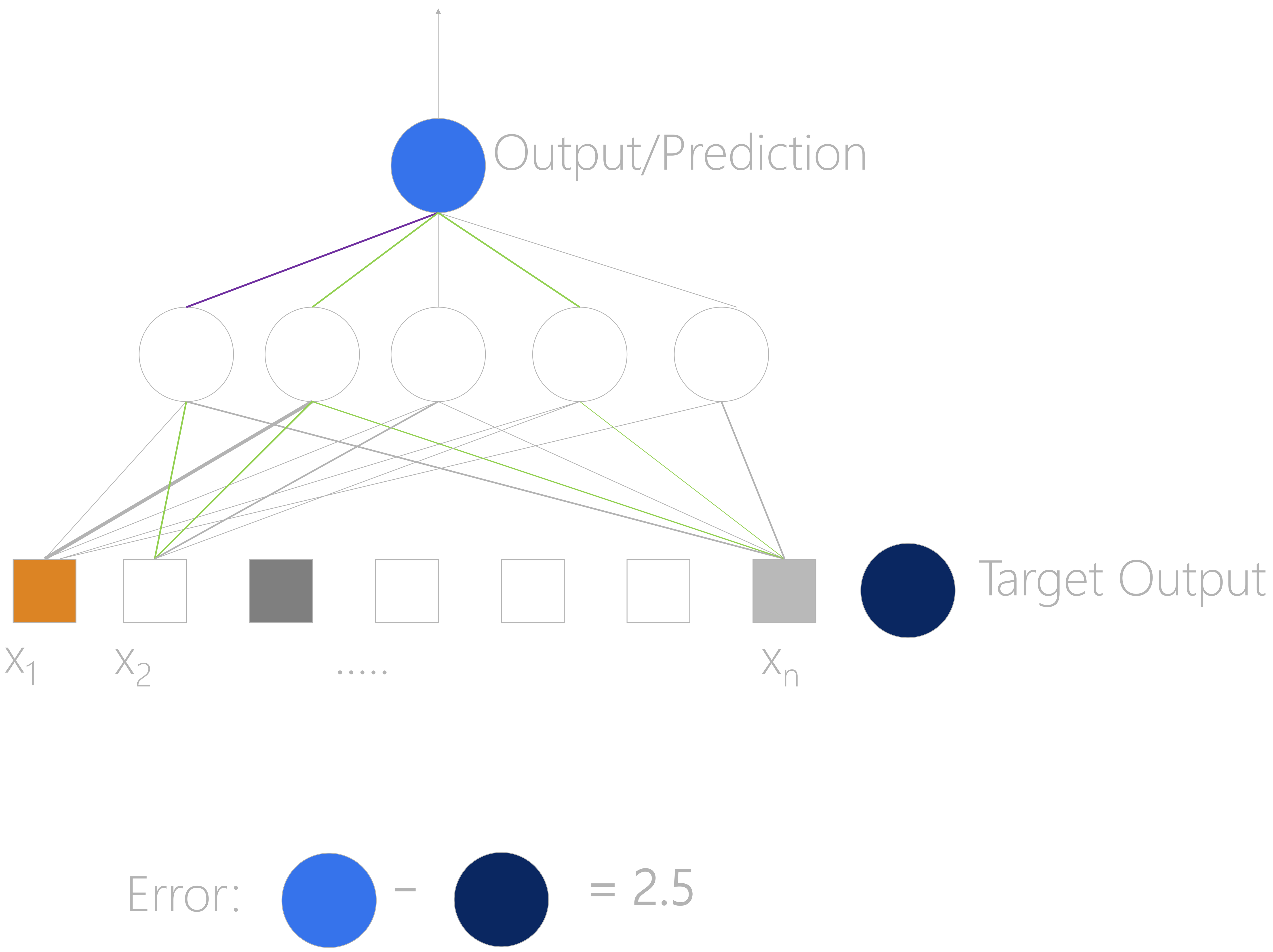
nn.Linear(7,10) => 7x10+10 = 80 learning parameters!

Learning Principle: adjust weigths



NVIDIA DEEP LEARNING INSTITUTE

Learning Principle: adjust weights

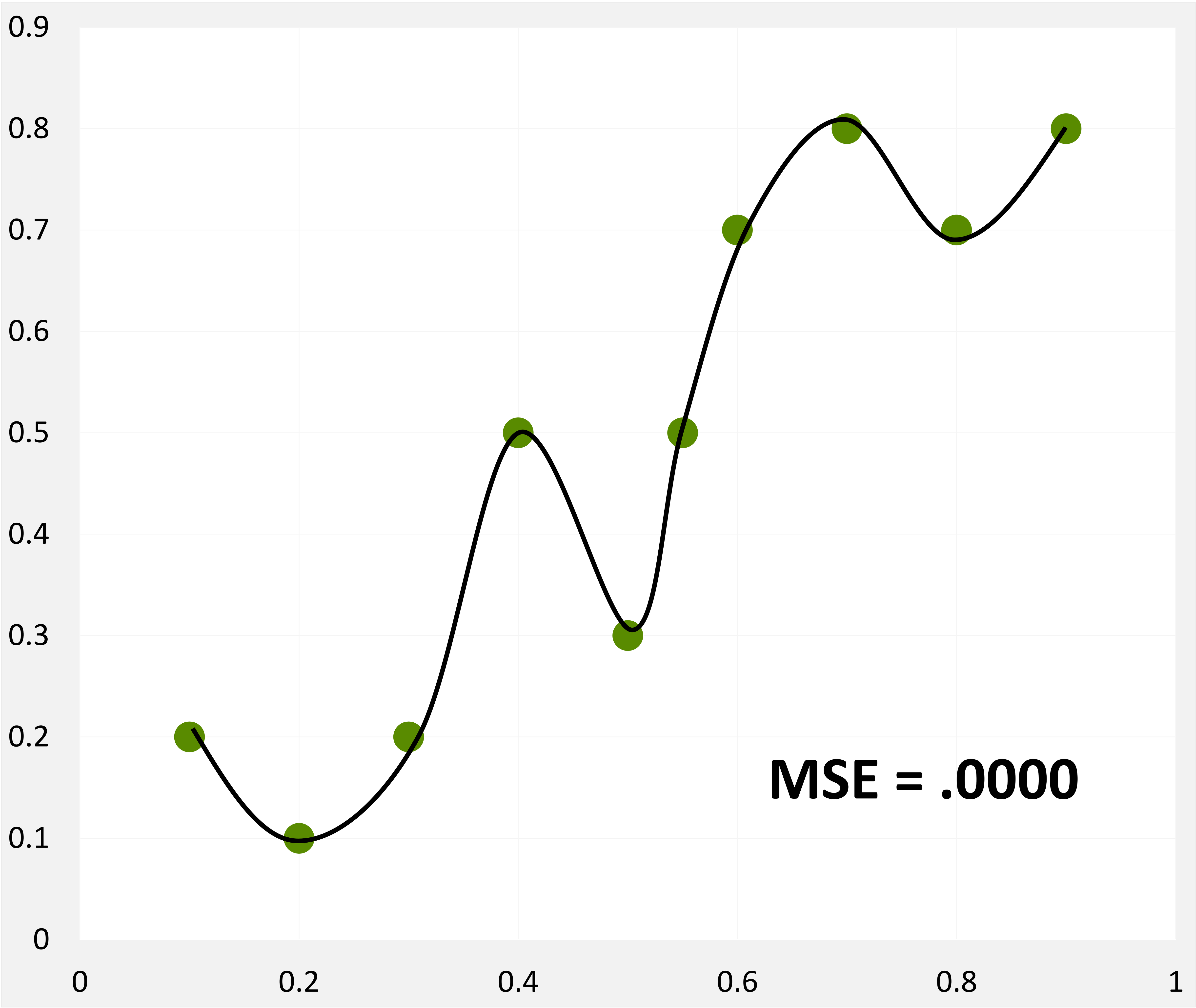


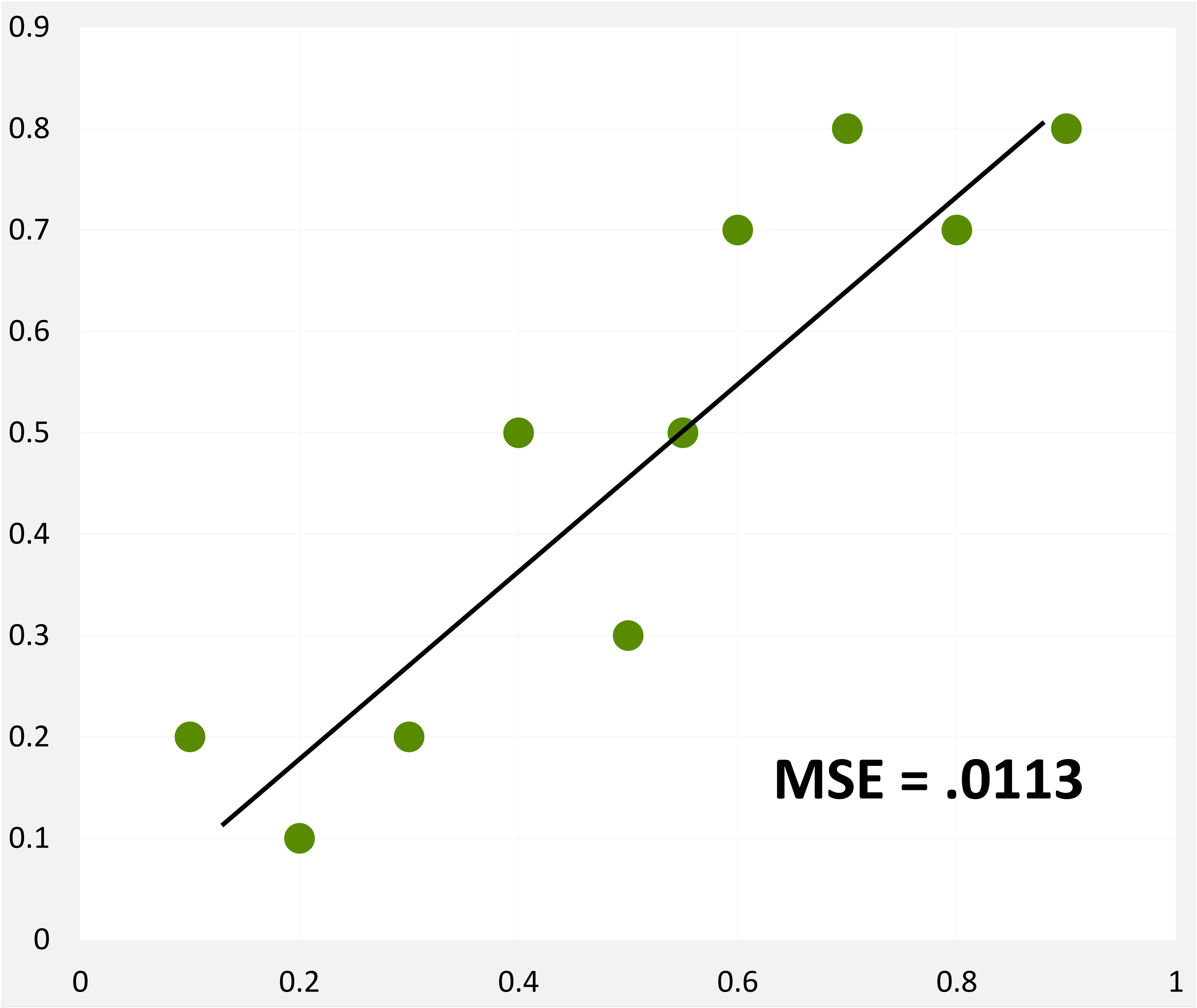
Overfitting

Why not have a super large neural network?

Overfitting

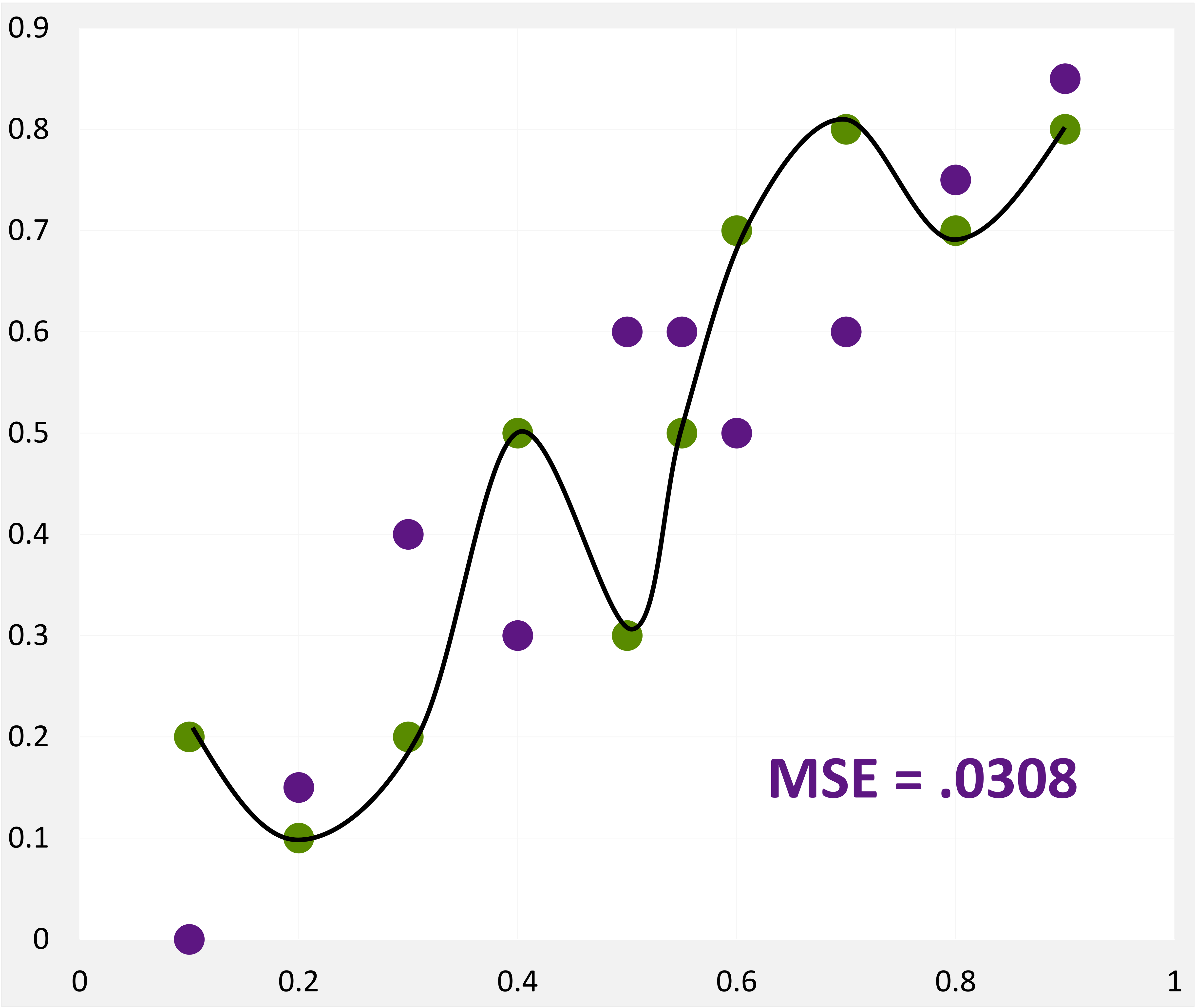
Which Trendline is Better?

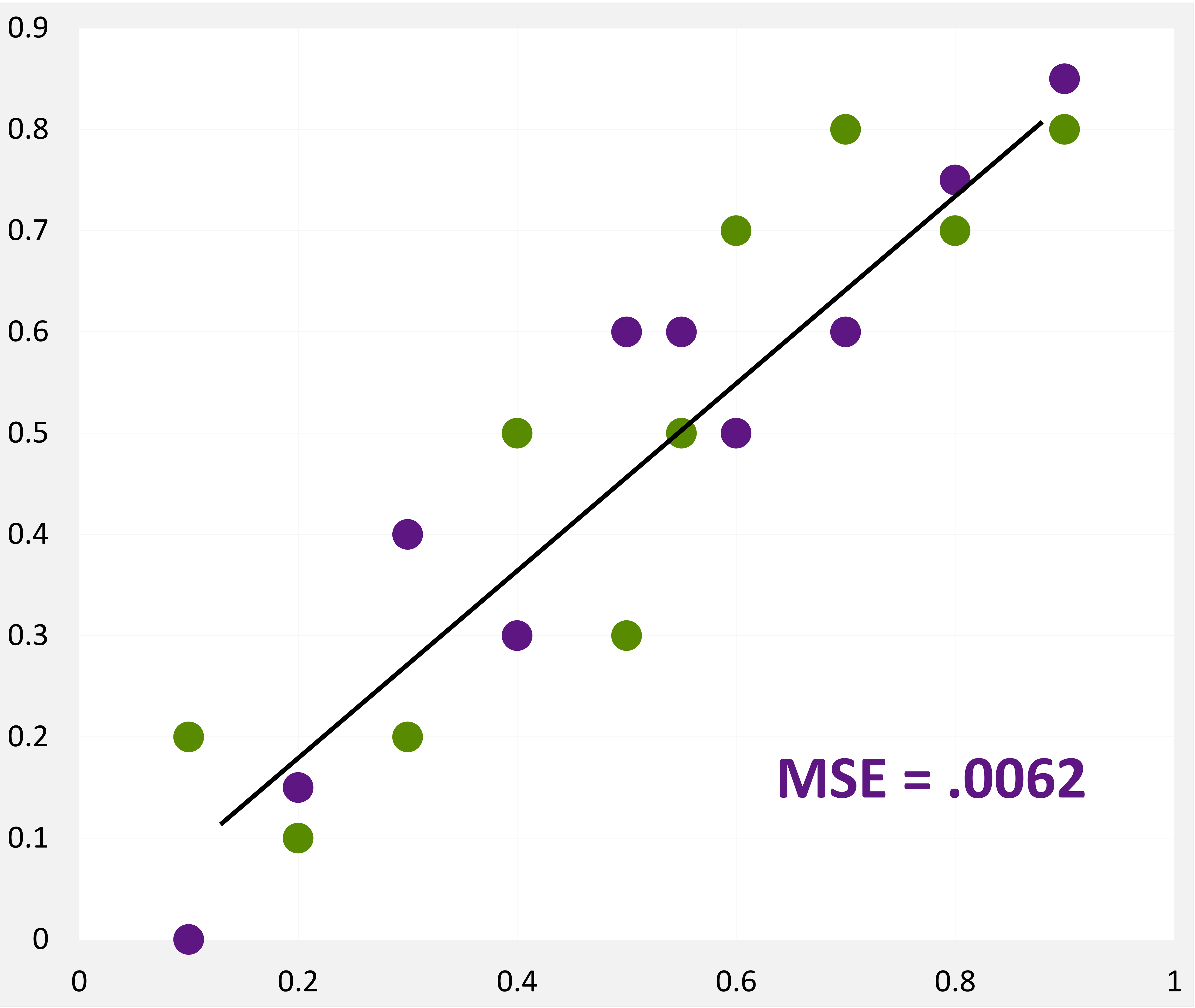




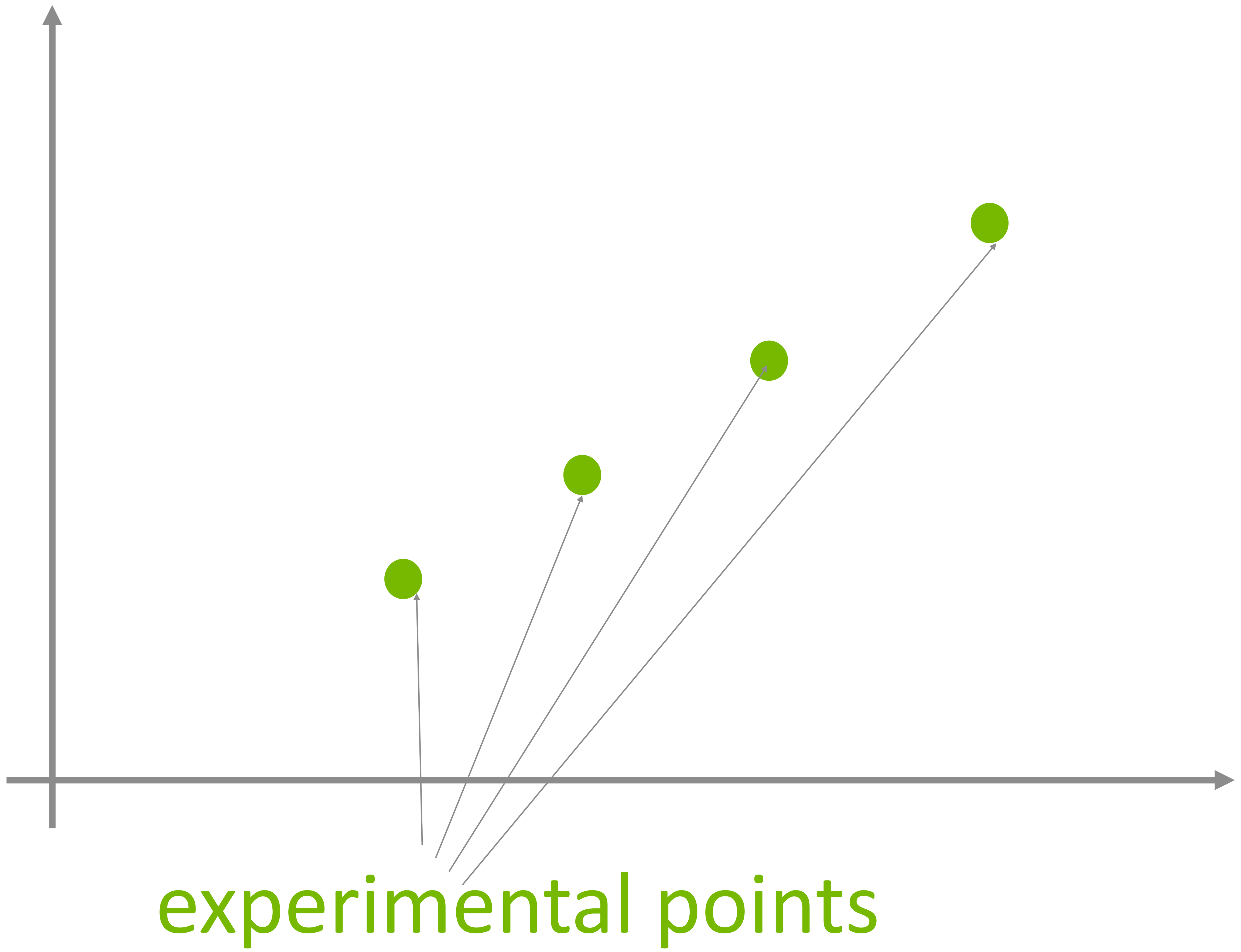
Overfitting

Which Trendline is Better?

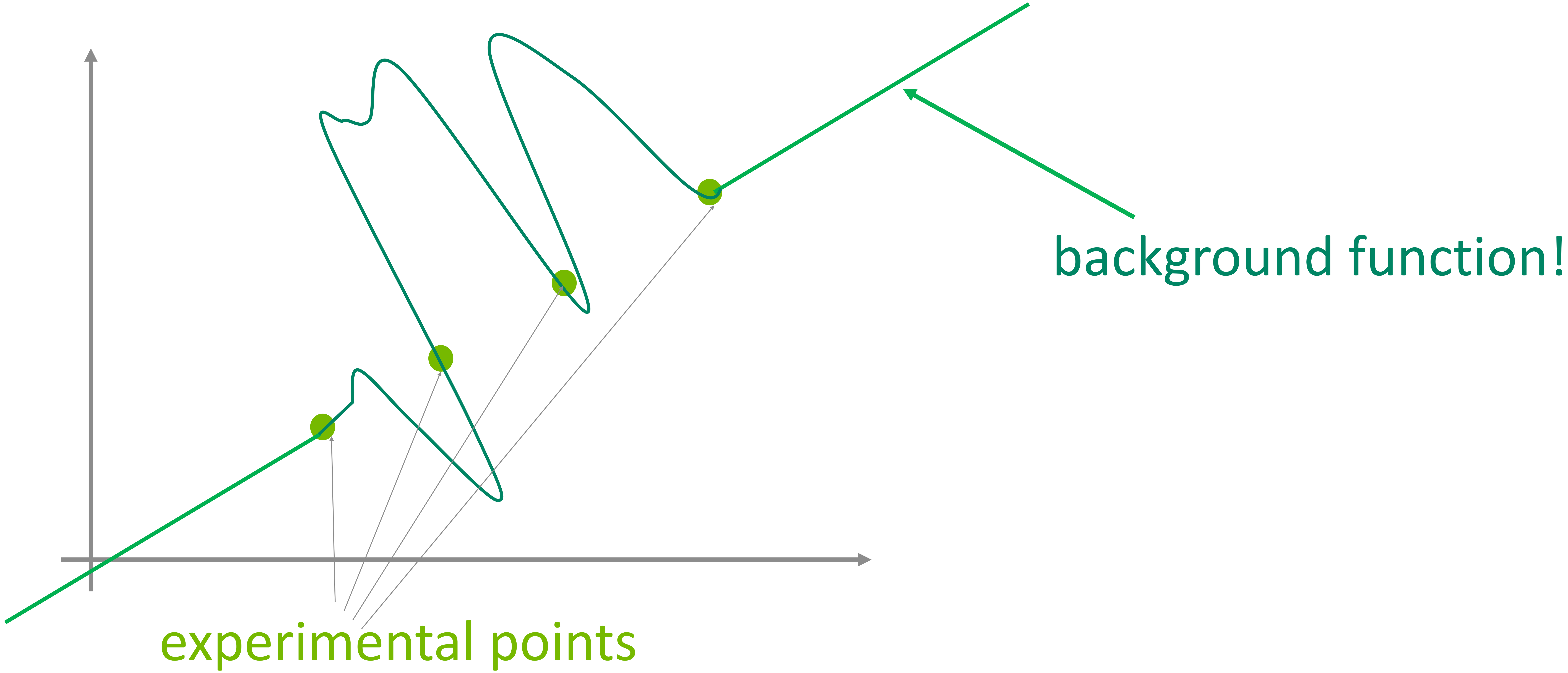




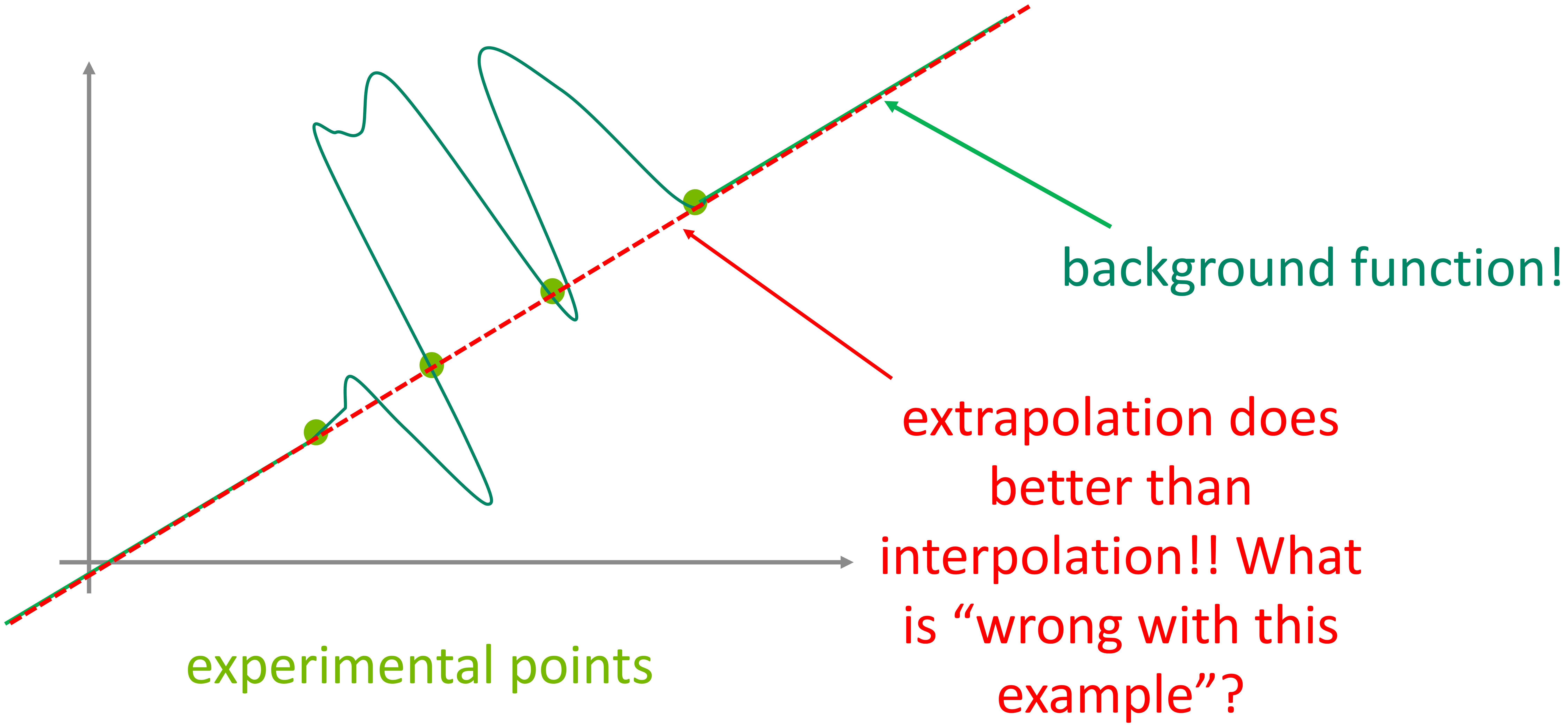
Why interpolation is better than extrapolation?



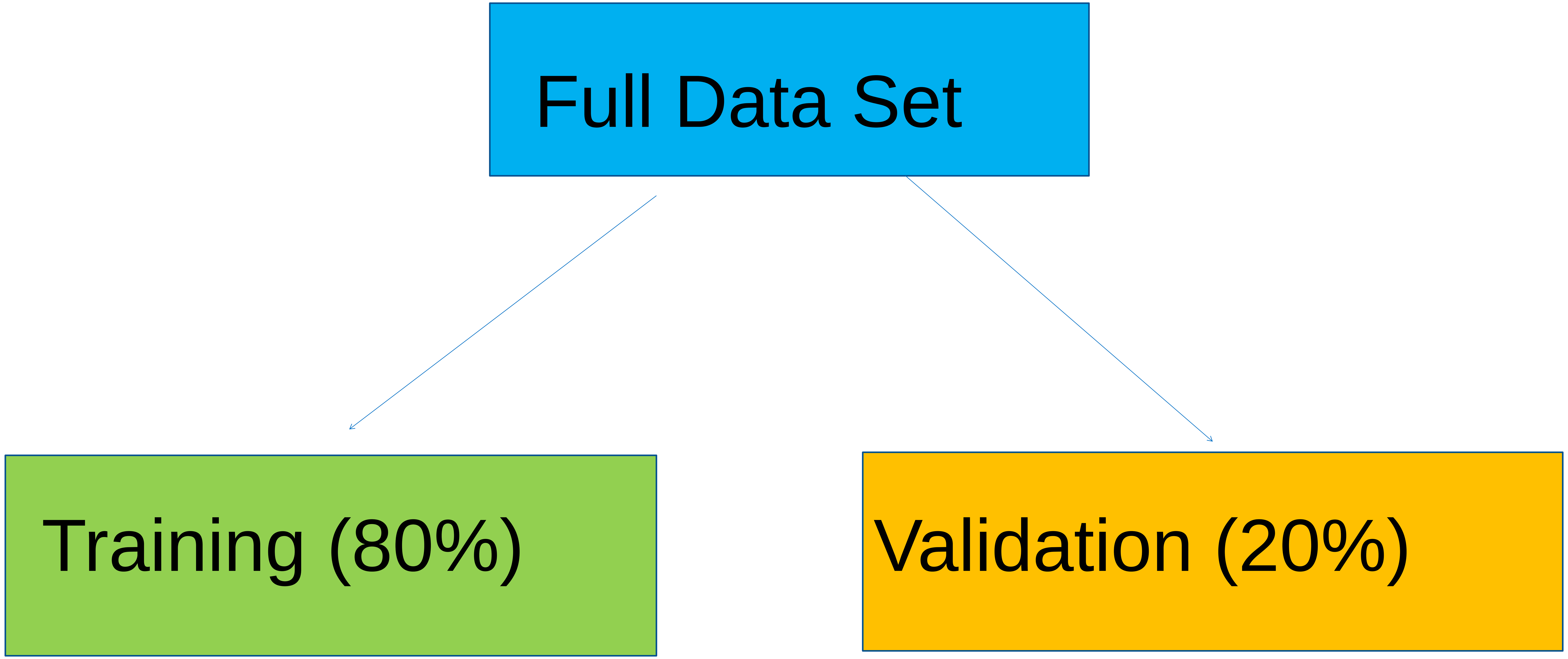
Why interpolation is better than extrapolation?



Why interpolation is better than extrapolation?



Definitions (III)



Definitions (IV)

Training vs Validation Data

Avoid memorization

Training data

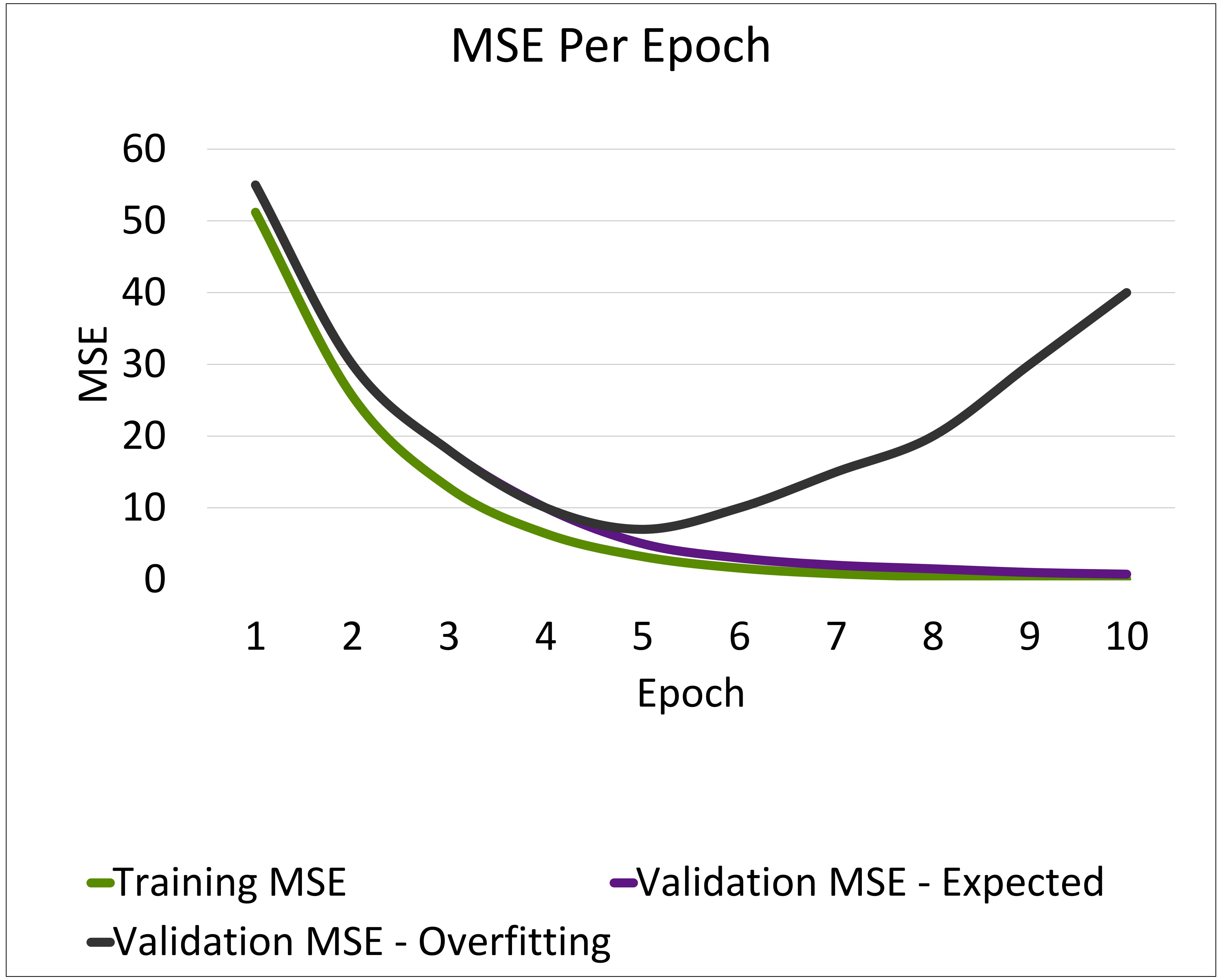
- Core dataset for the model to learn on

Validation data

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

Overfitting

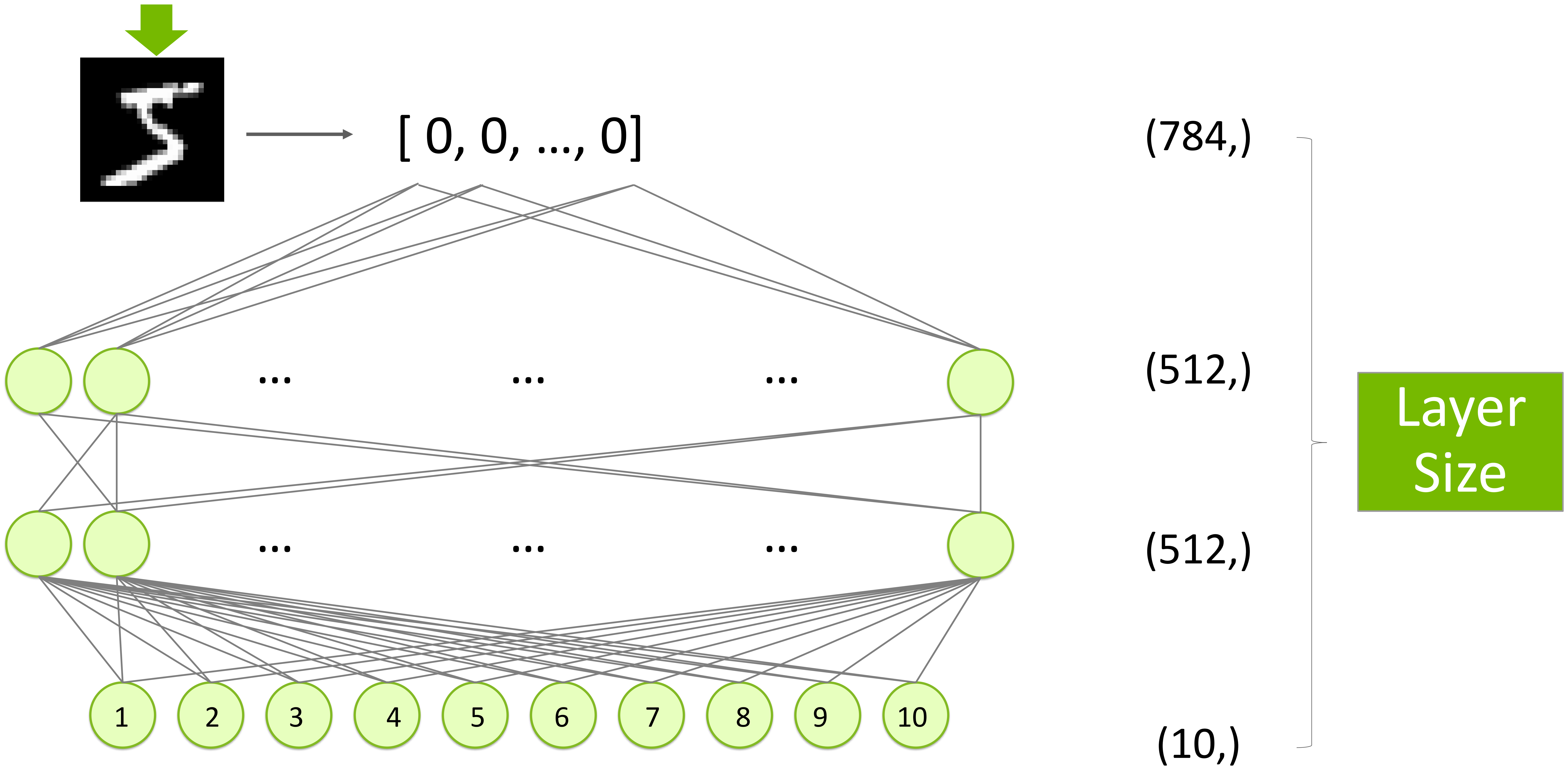
- When model performs well on the training data, but not the validation data (evidence of memorization)
- Ideally the accuracy and loss should be similar between both datasets



From Regression to Classification

An MNIST Model

BATCH_SIZEx1x28x28 = BATCH_SIZEx784



10 possible numbers

An MNIST Model

During Prediction

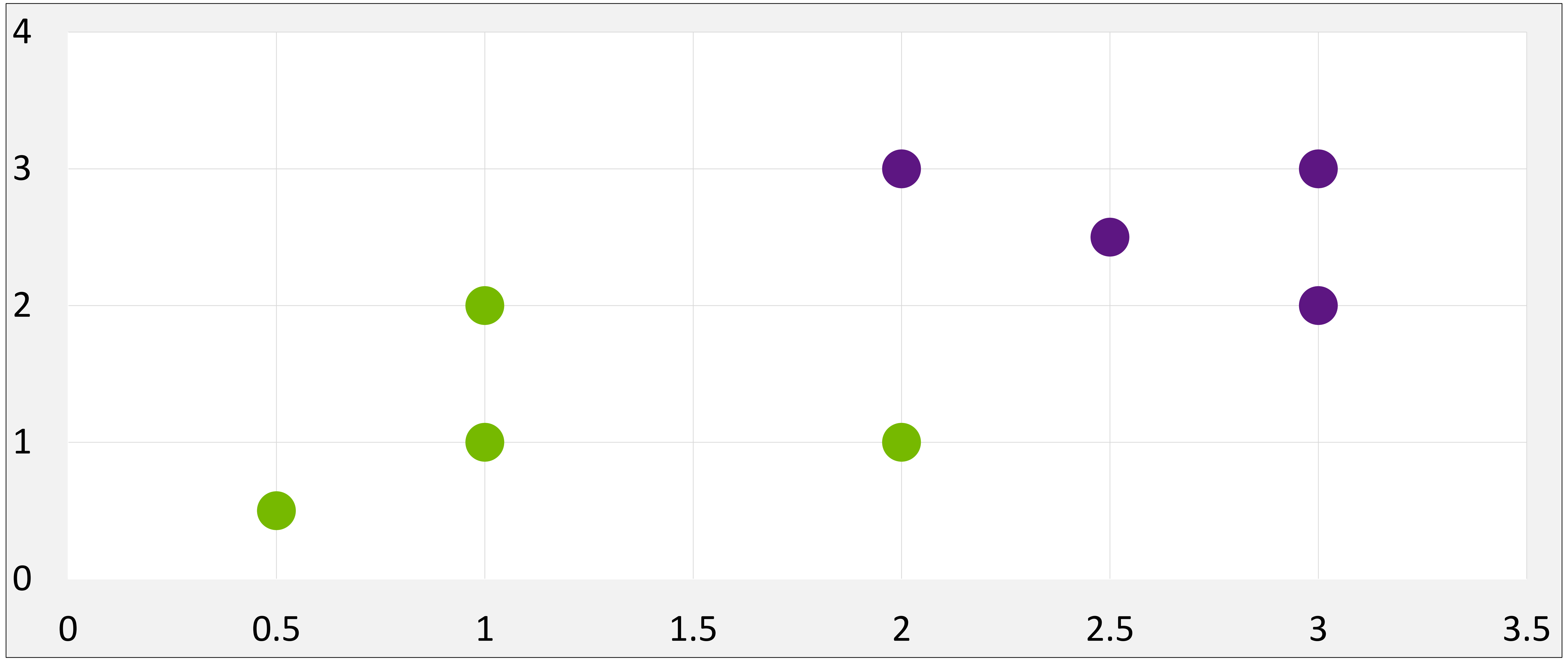
BATCH_SIZEx1x28x28 = BATCH_SIZEx784

An MNIST Model

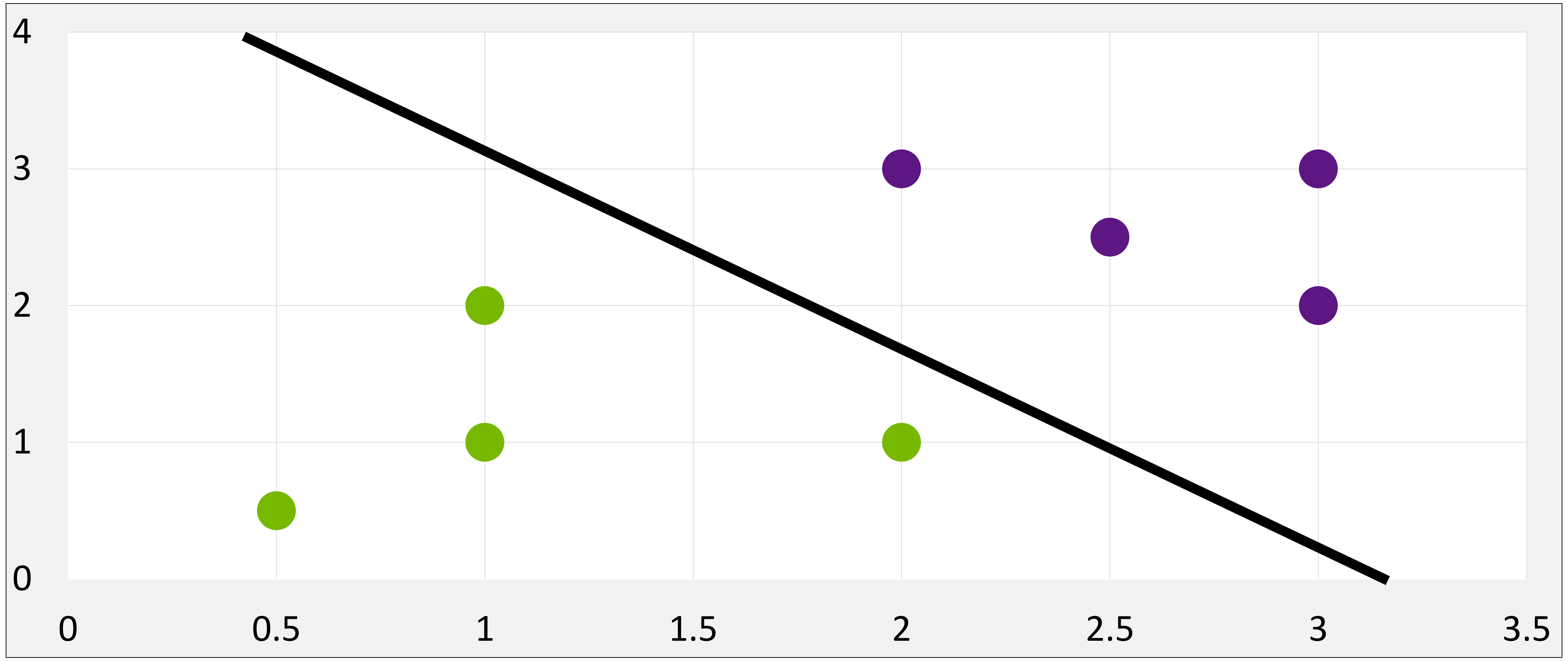
During Training

BATCH_SIZEx1x28x28 = BATCH_SIZEx784

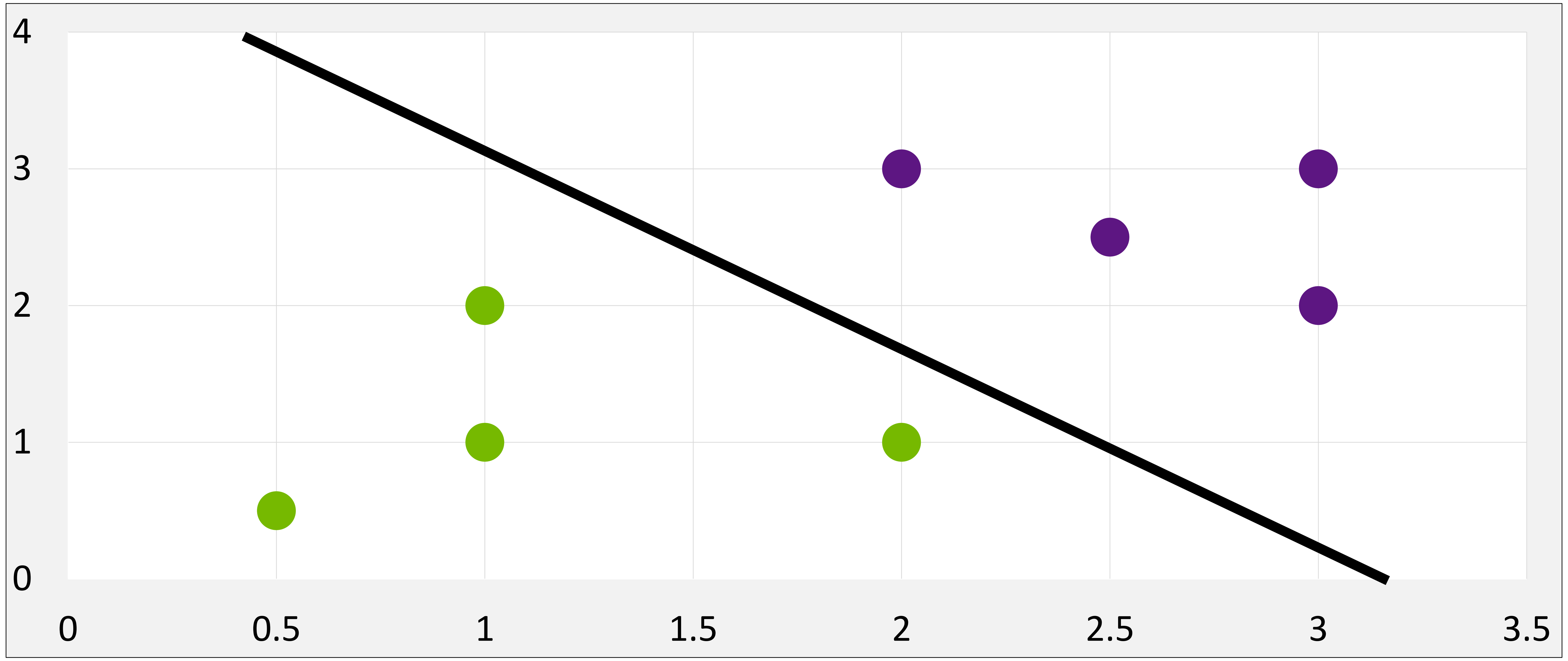
RMSE For Probabilities?

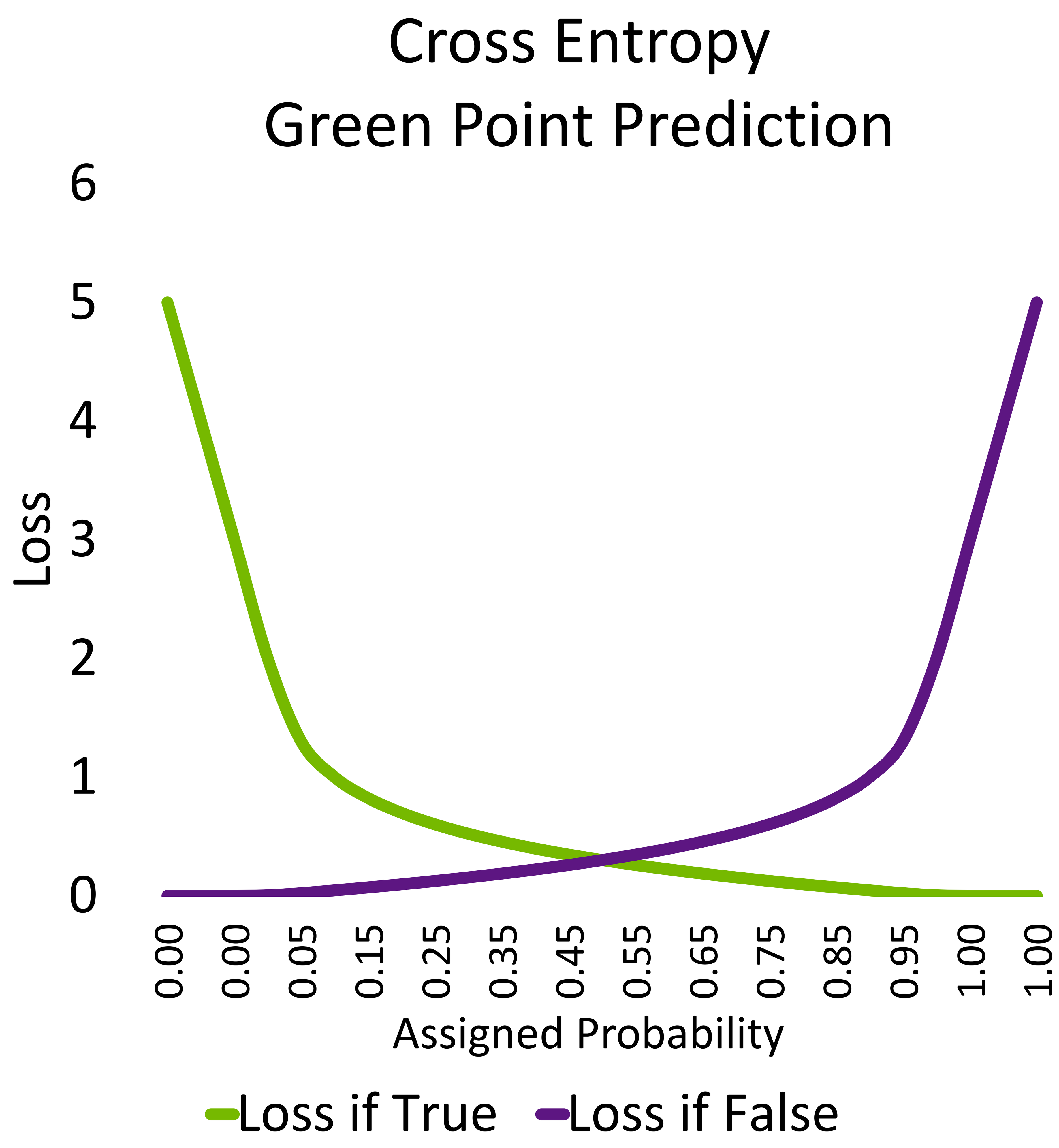


RMSE For Probabilities?

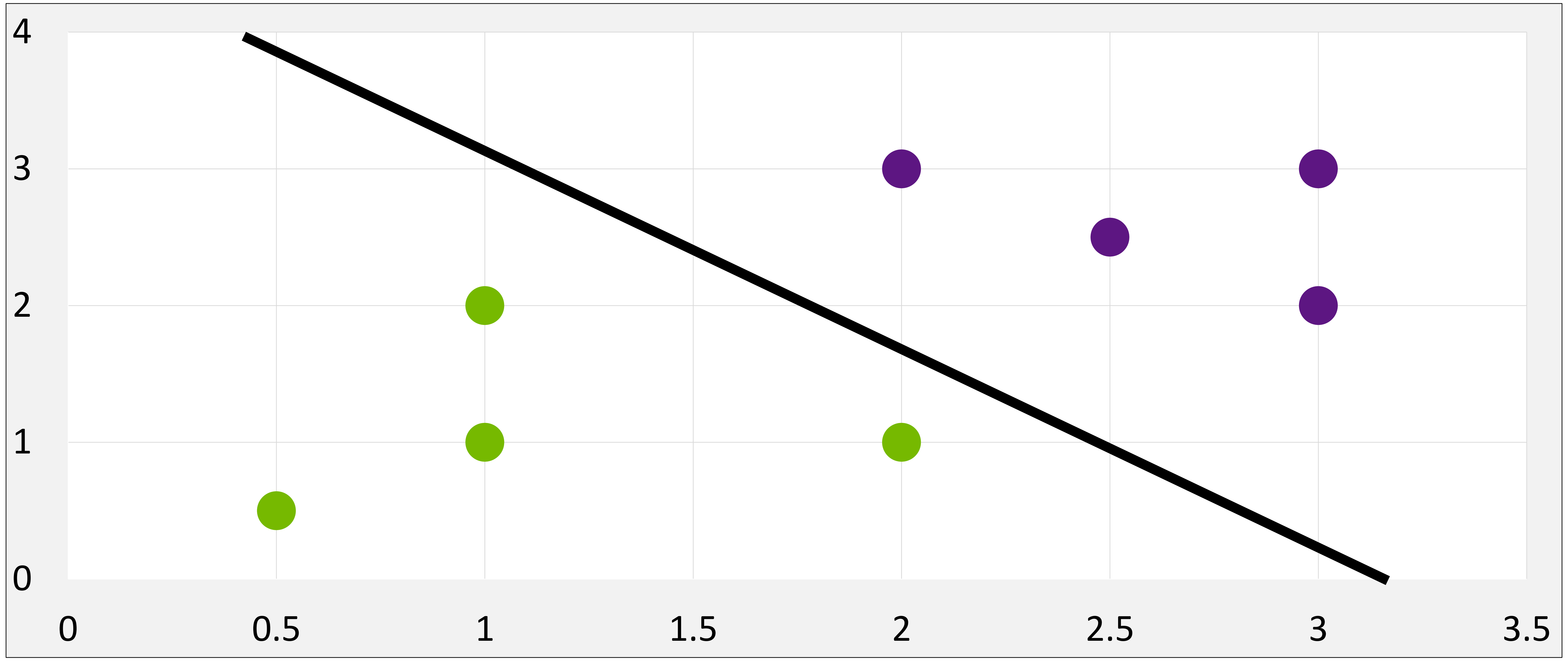


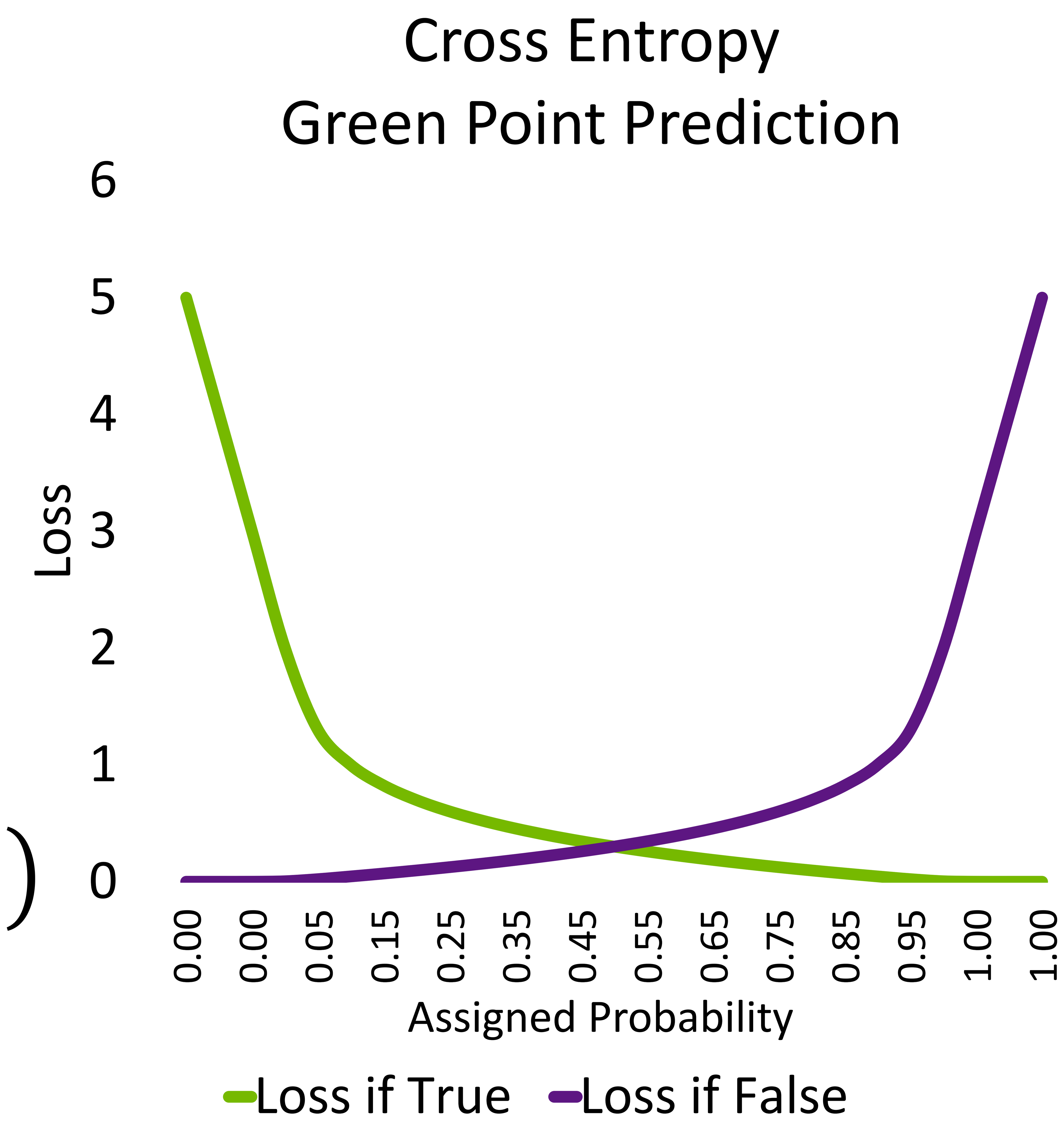
Cross Entropy





Cross Entropy



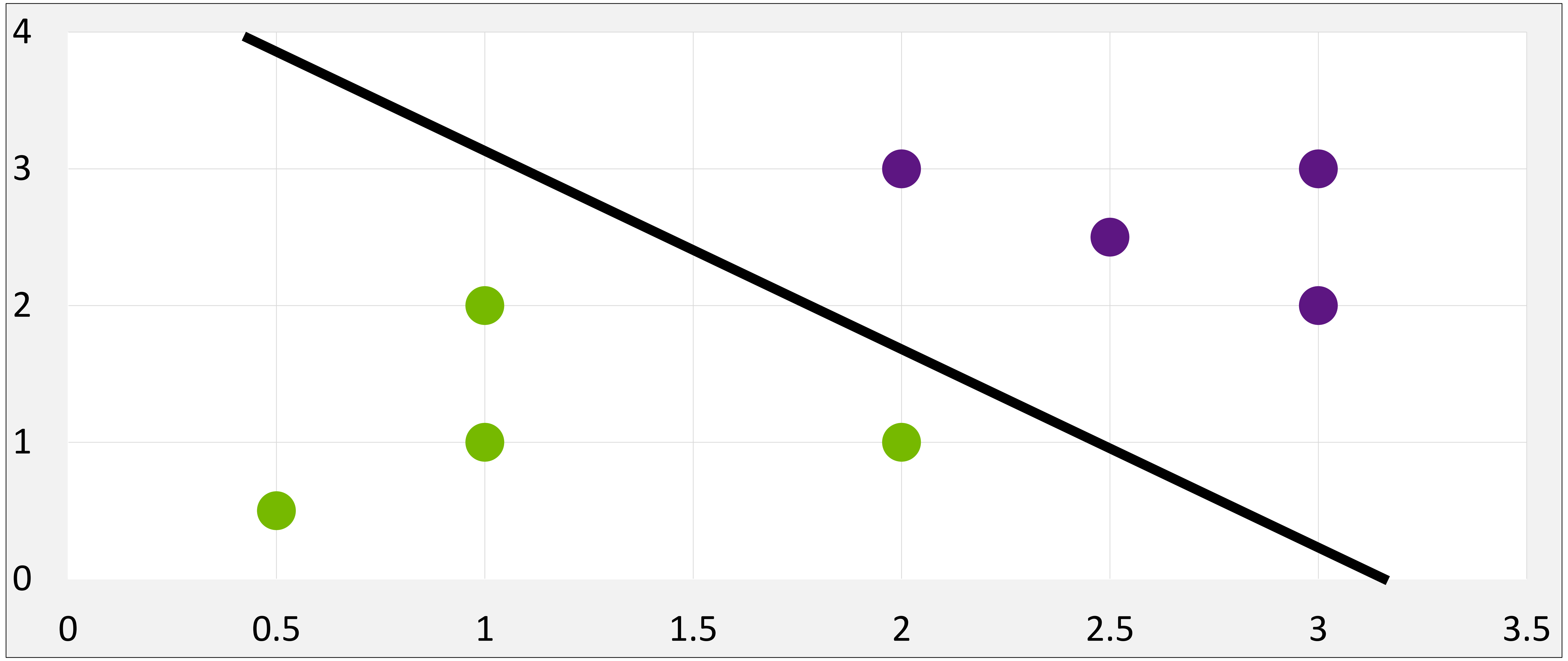


$$Loss = -\left((t(x) \cdot \log(p(x)) + \left(1 - t(x)\right) \cdot \log(1 - p(x))\right)$$

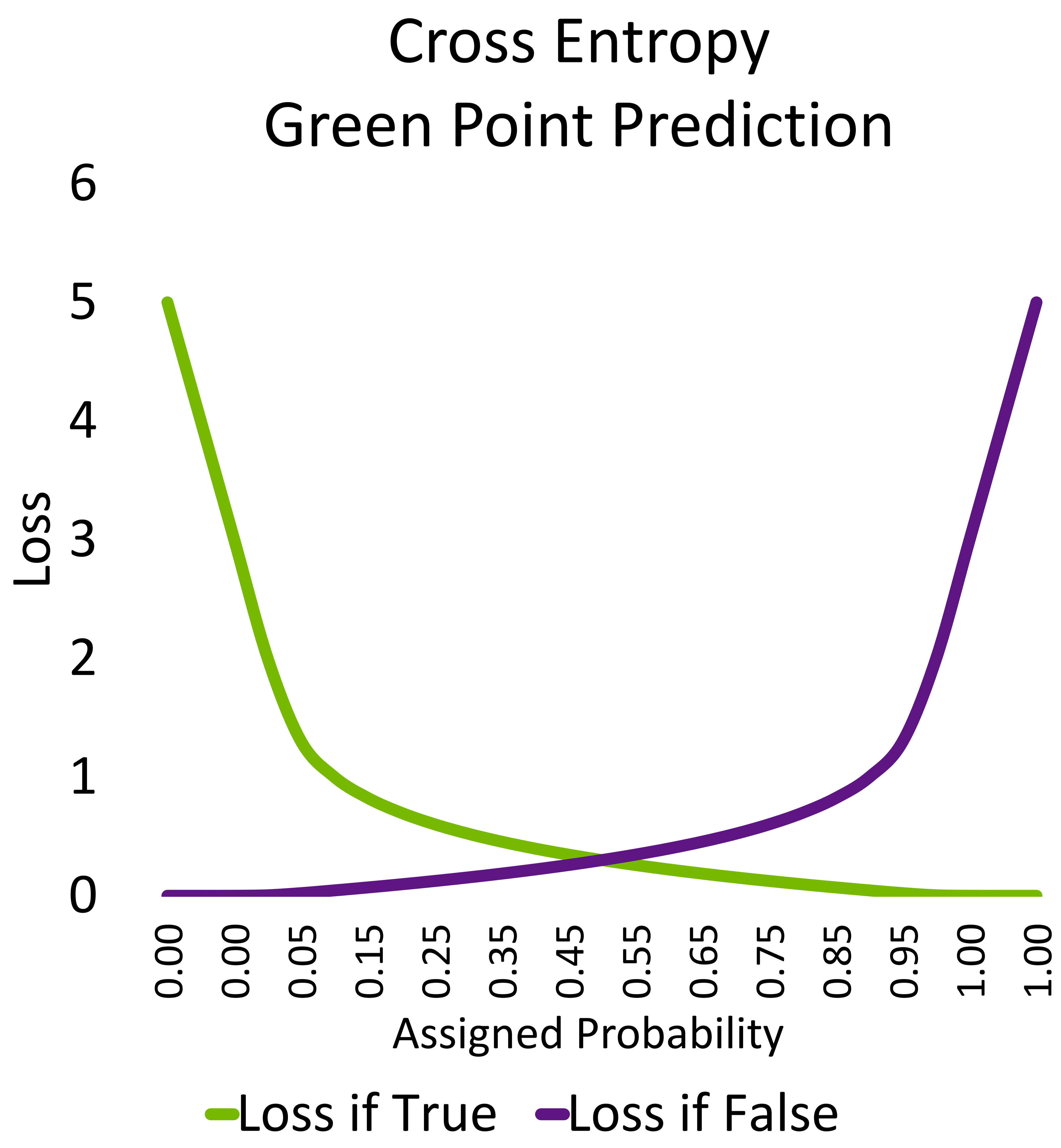
$$t(x) = target\ (0\ if\ False, 1\ if\ True)$$

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

Cross Entropy




```
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
```



Overview of the Course

Structure of the Course

1. "Hello World" of Deep Learning
2. Train a more complicated model
3. New architectures and techniques to improve performance
4. Pre-trained models
5. Transfer learning

Platform of the Course

GPU powered cloud server

JupyterLab platform

Jupyter notebooks for interactive coding

Software of This Course

- Major deep learning platforms:
  - TensorFlow + Keras (Google)
  - PyTorch (Meta)
  - MXNet (Apache)
- We’ll be using PyTorch
- Good idea to gain exposure to others moving forward

Hands on Exercises

- Get comfortable with the process of deep learning
- Exposure to different models and datatypes
- Get a jump-start to tackle your own projects

- Have fun!!

Hands on: work through Introduction Section

1. WIFI Info: eduroam
2. Browser Recommendation: Chrome
3. websocketstest.courses.nvidia.com
4. https://learn.nvidia.com/dli-event
5. create an Account (if you have not done yet!)
6. event code: STFC_FDL_AMBASSADOR_NO24
7. work through the Introduction Section and ‘Start’ launching your first GPU task

NVIDIA DEEP LEARNING INSTITUTE

First exercise: MNSIT dataset classification (Hello Neural Networks)

Train a network to correctly classify handwritten digits

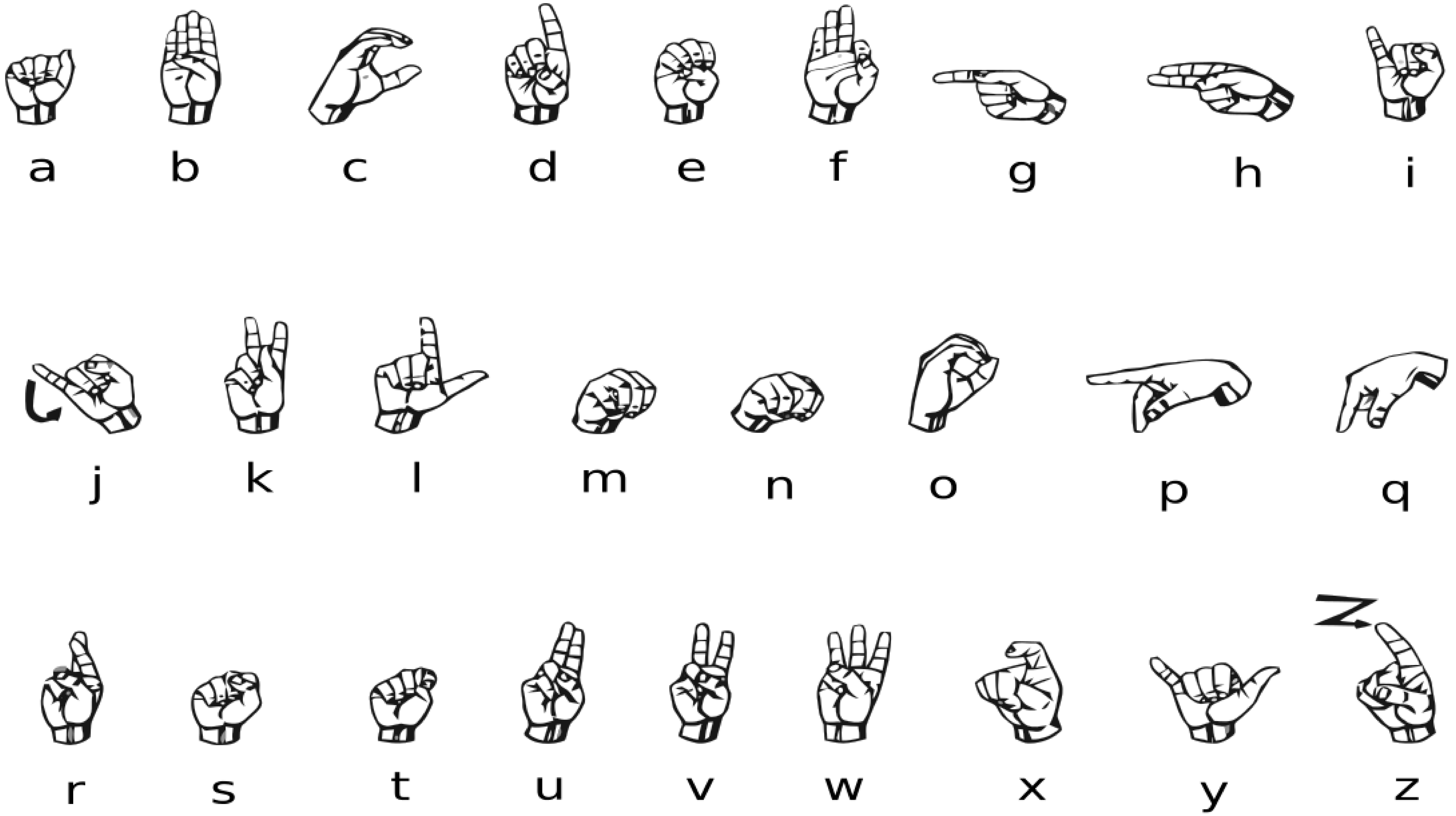
Historically important and difficult task for computers

Try learning like a Neural Network

Get exposed to the example, and try to figure out the rules to how it works

Second Exercise

The American Sign Language Alphabet



Questions?

Appendix: Gradient Descent

Helping the Computer Cheat Calculus

Learning from Error

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

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

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

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

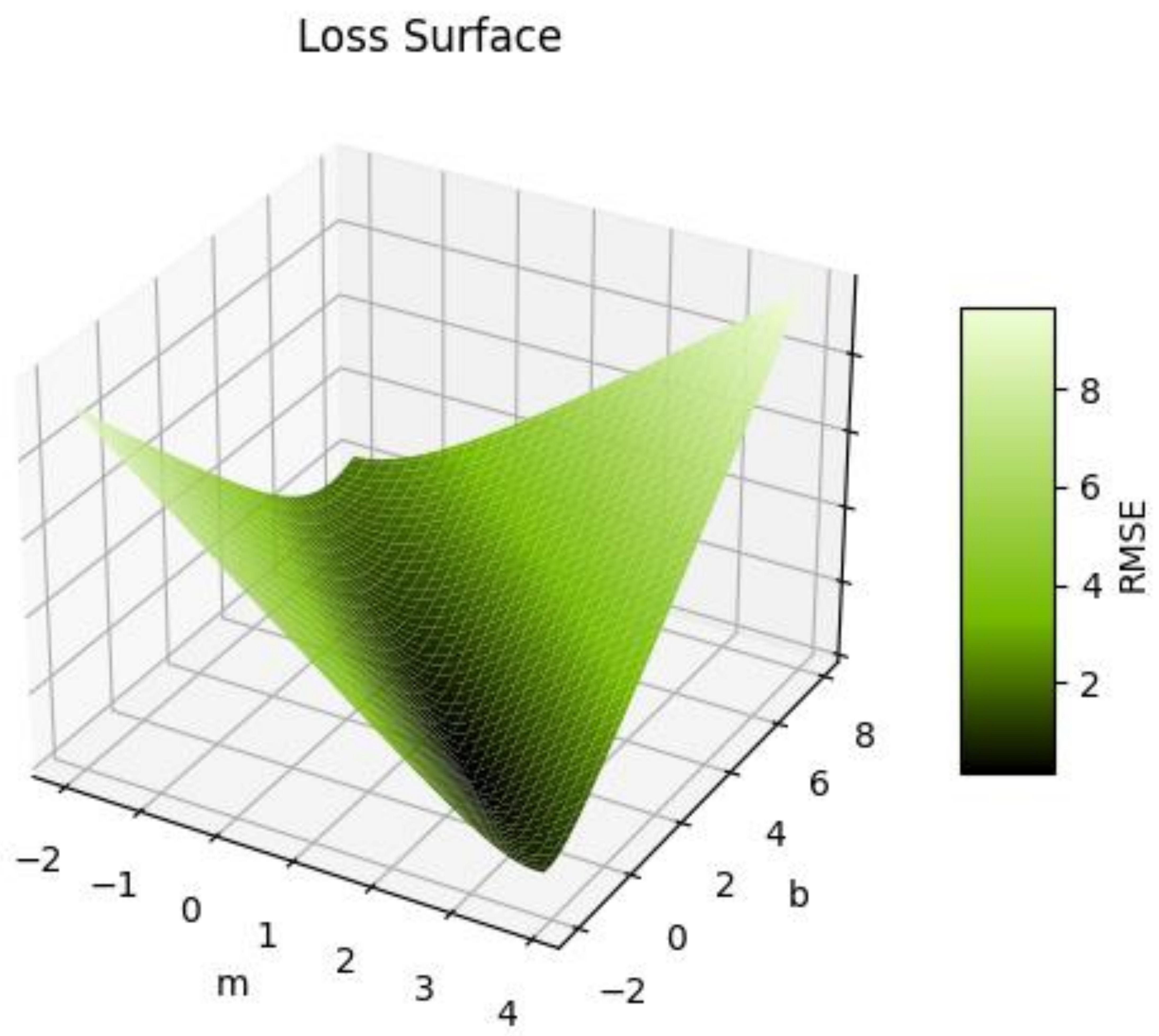
$$\frac{\partial MSE}{\partial m} = -3$$

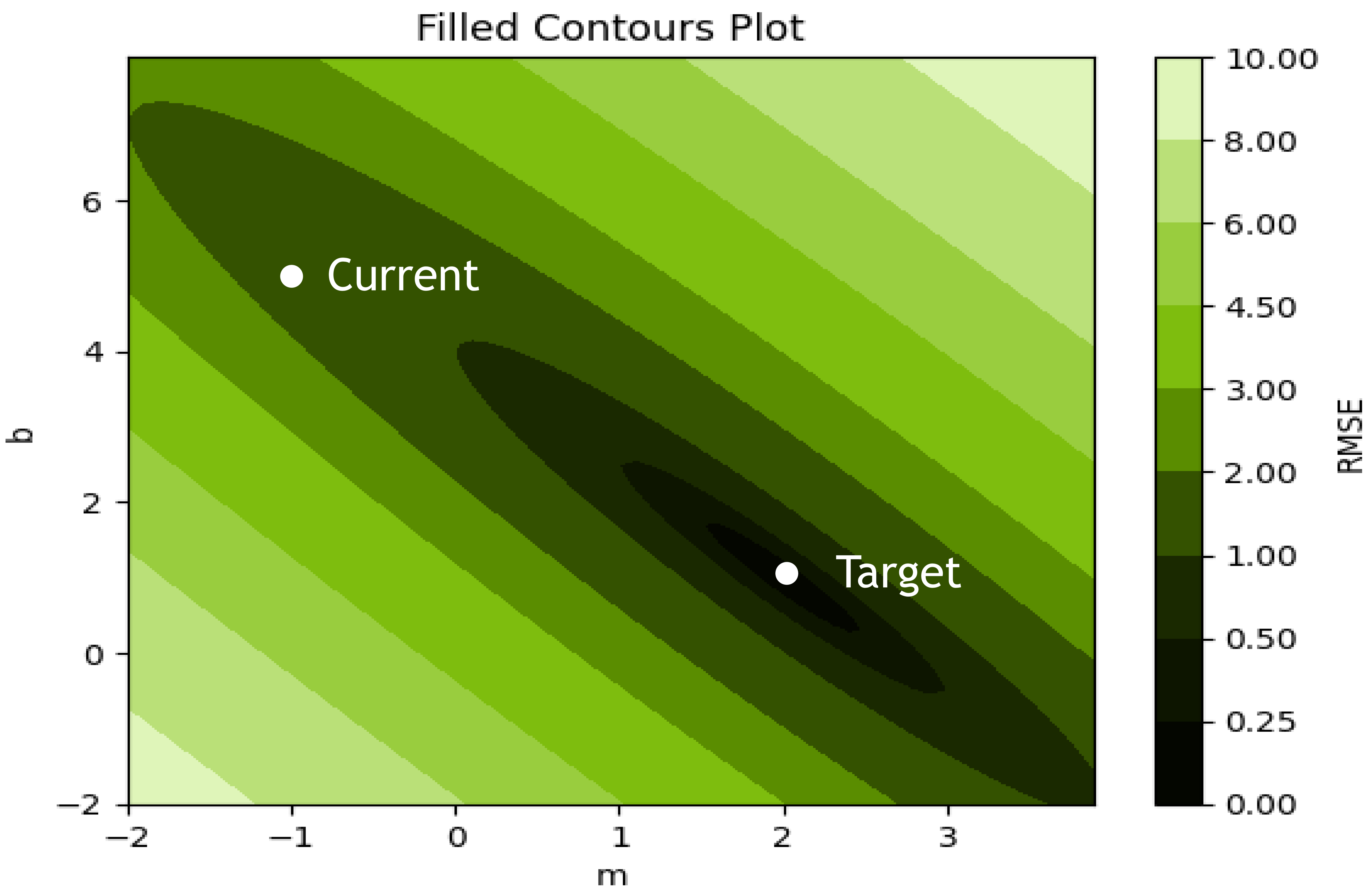
$$\frac{\partial MSE}{\partial b} = -1$$

$m = -1$

$\mathrm{b} = 5$

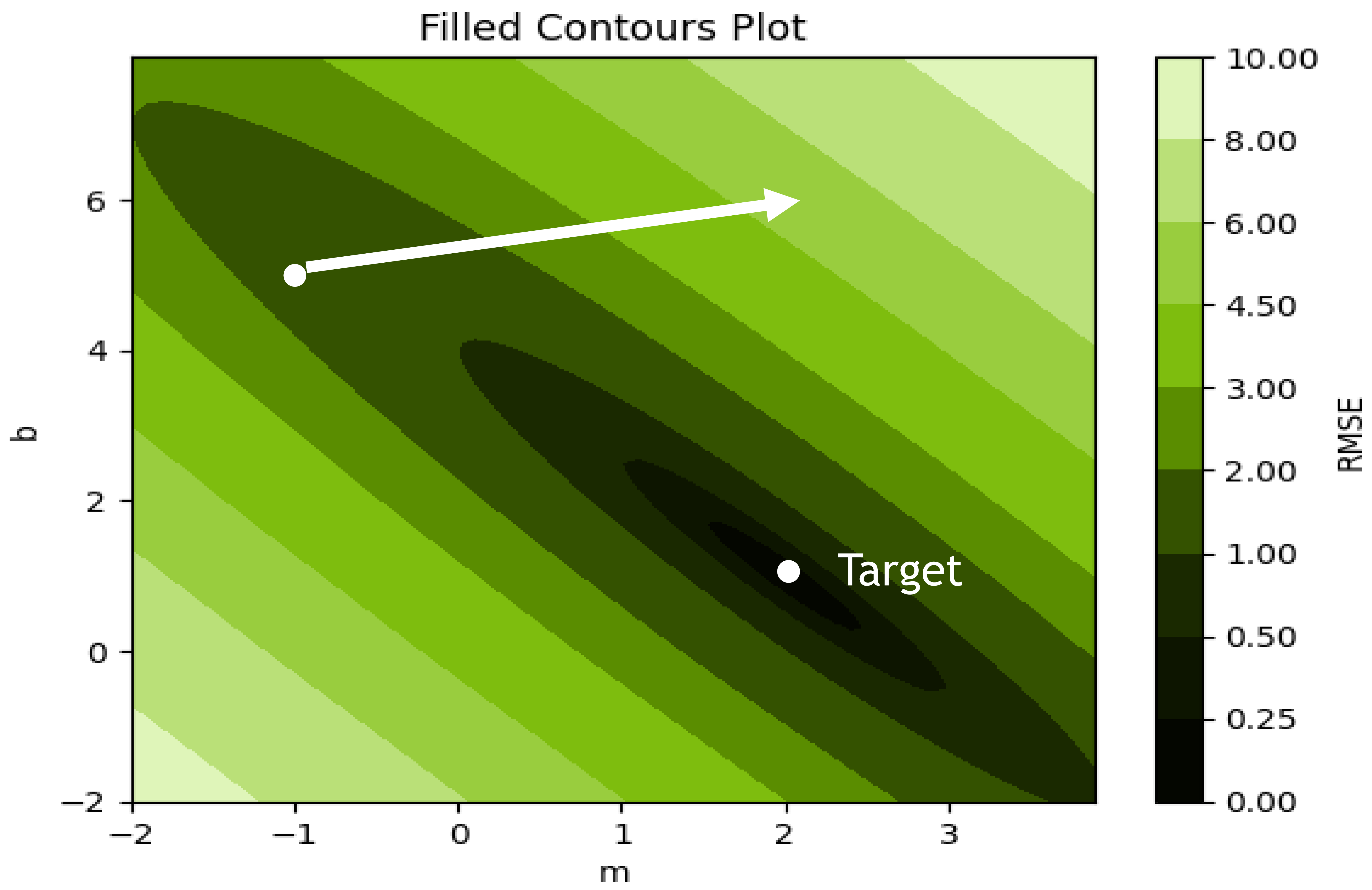
The Loss Curve





The Loss Curve

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

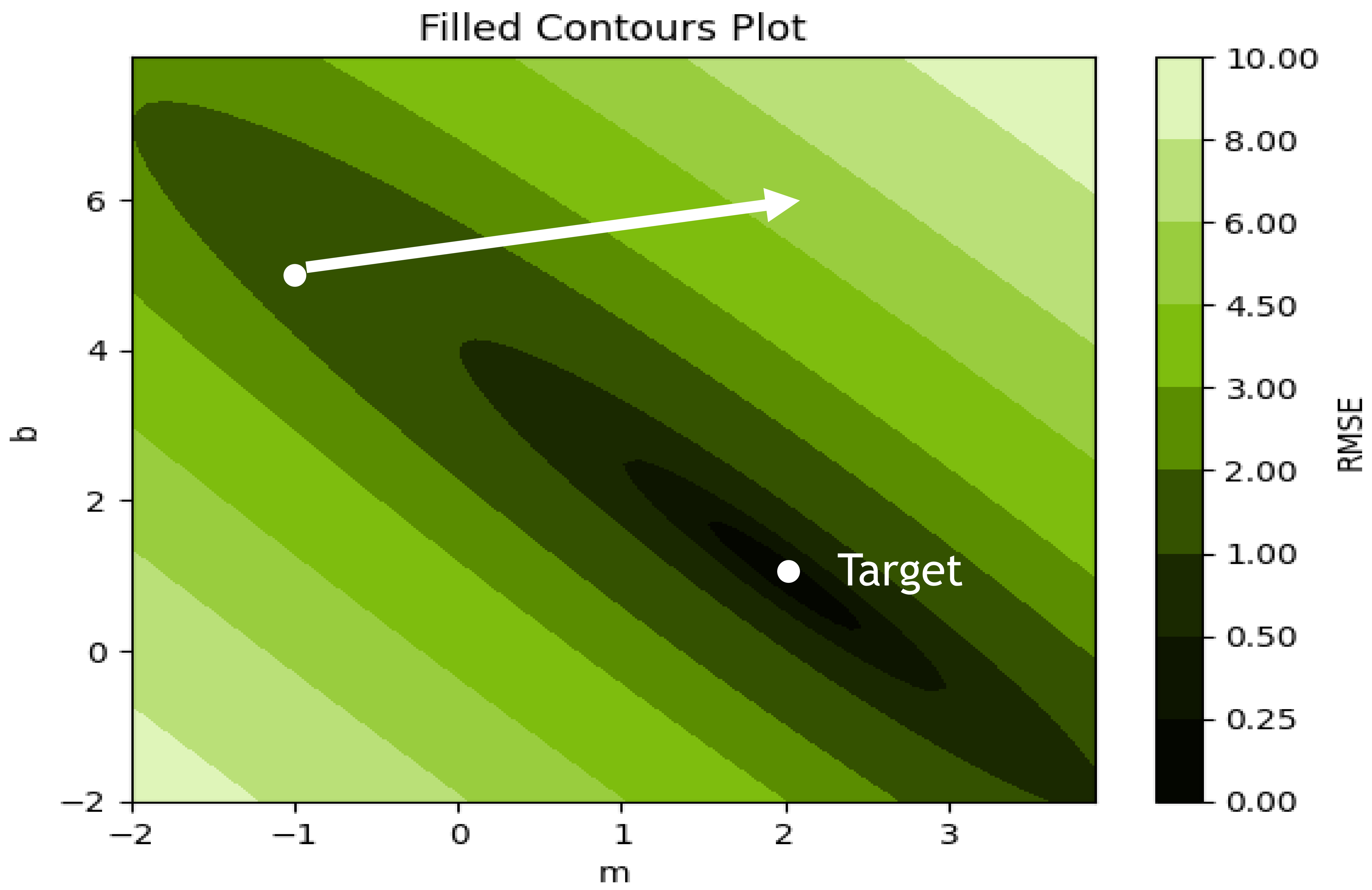


The Loss Curve

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

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

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



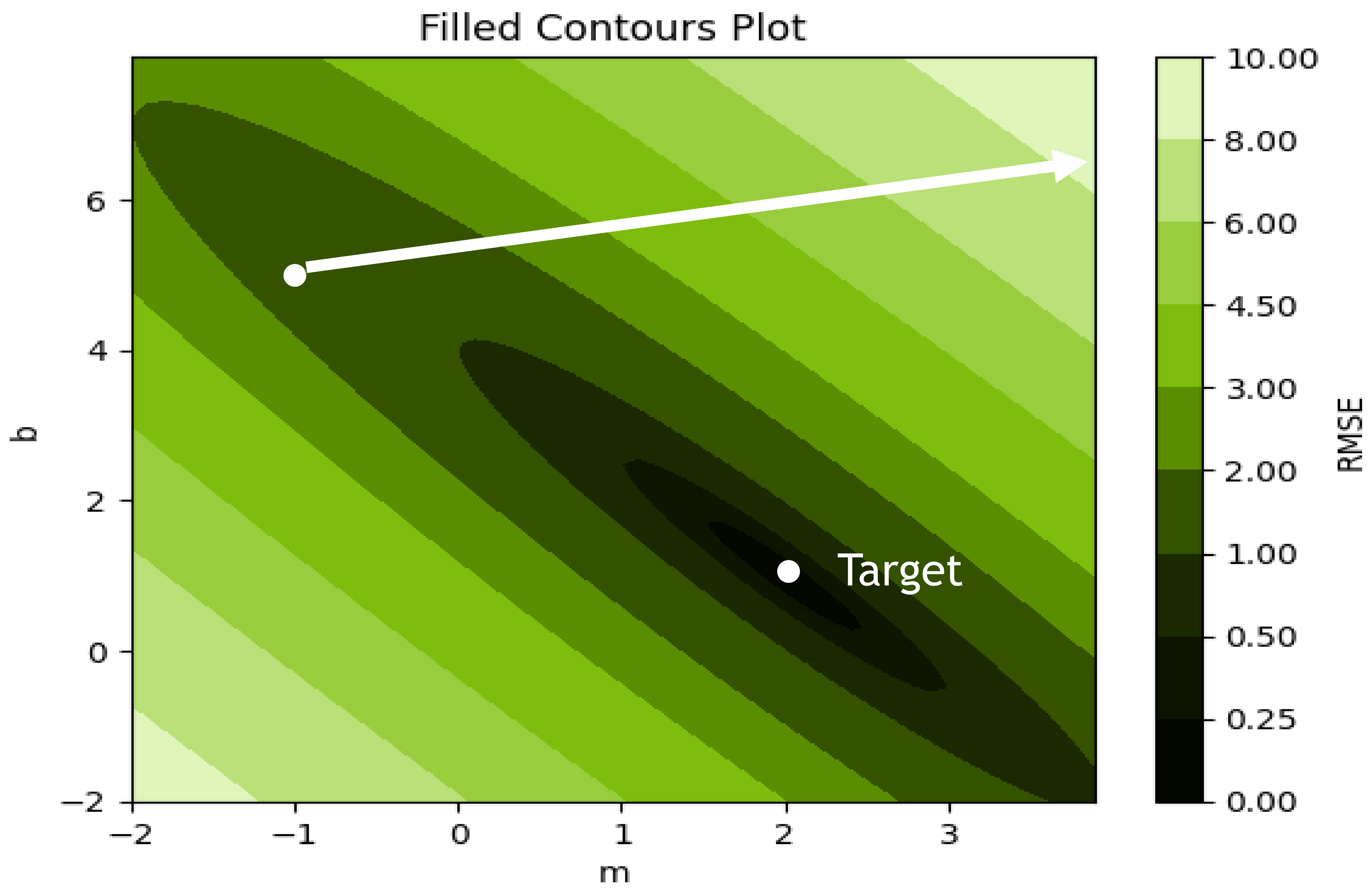
The Loss Curve

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

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

$$\lambda = 2$$

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



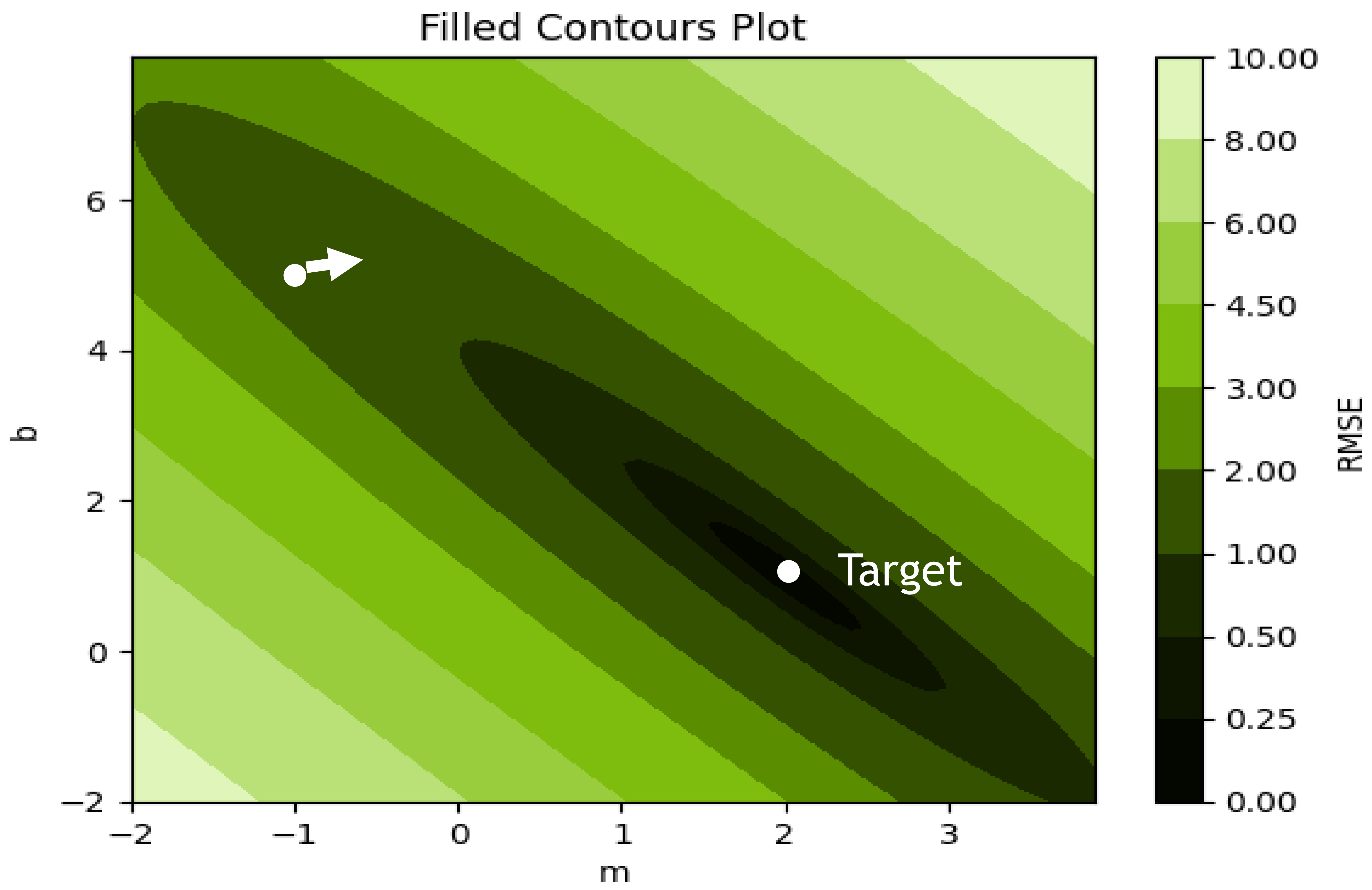
The Loss Curve

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

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

$$\lambda = .1$$

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

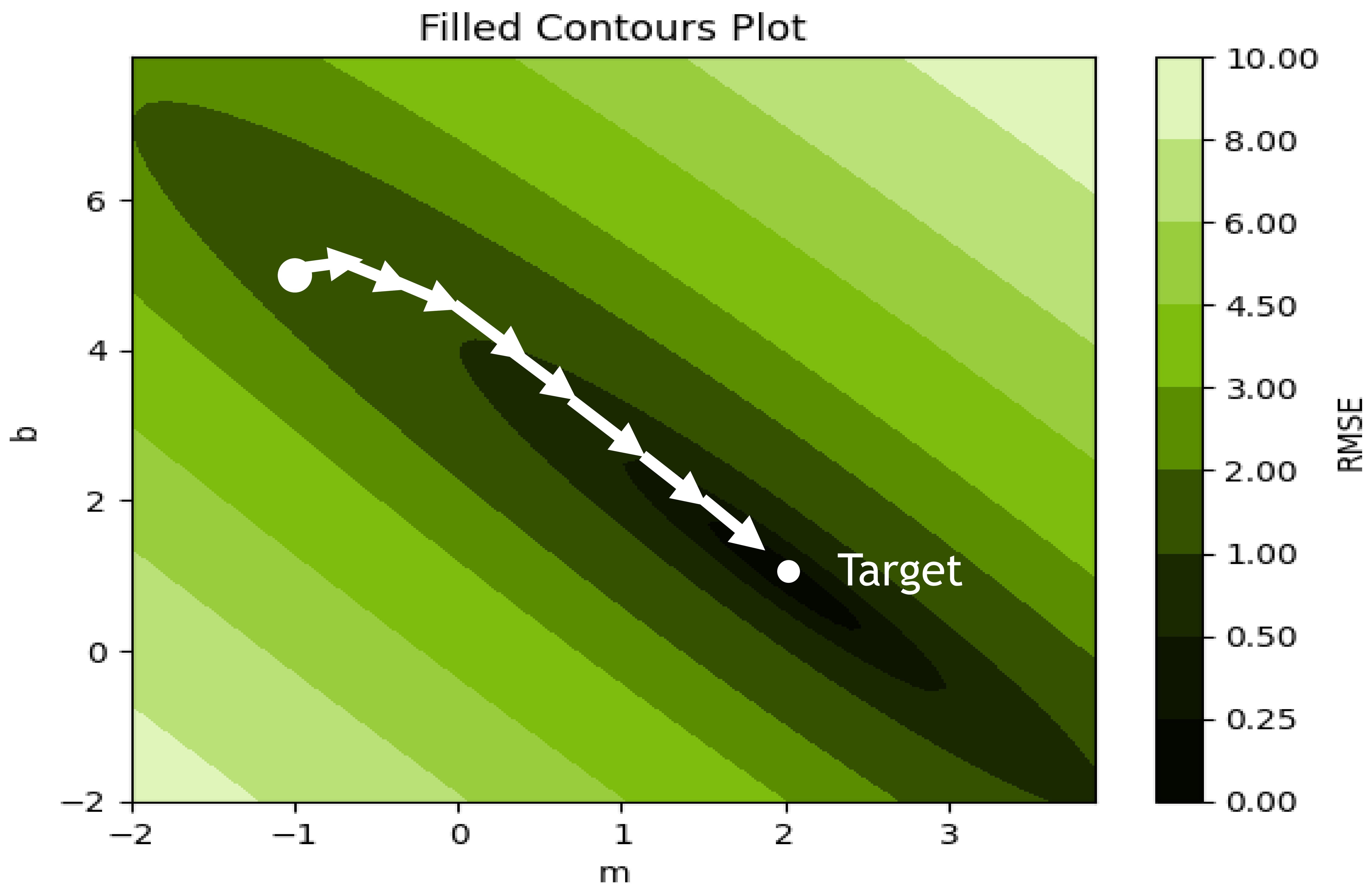


The Loss Curve

$$\lambda = .1$$

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

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



Other

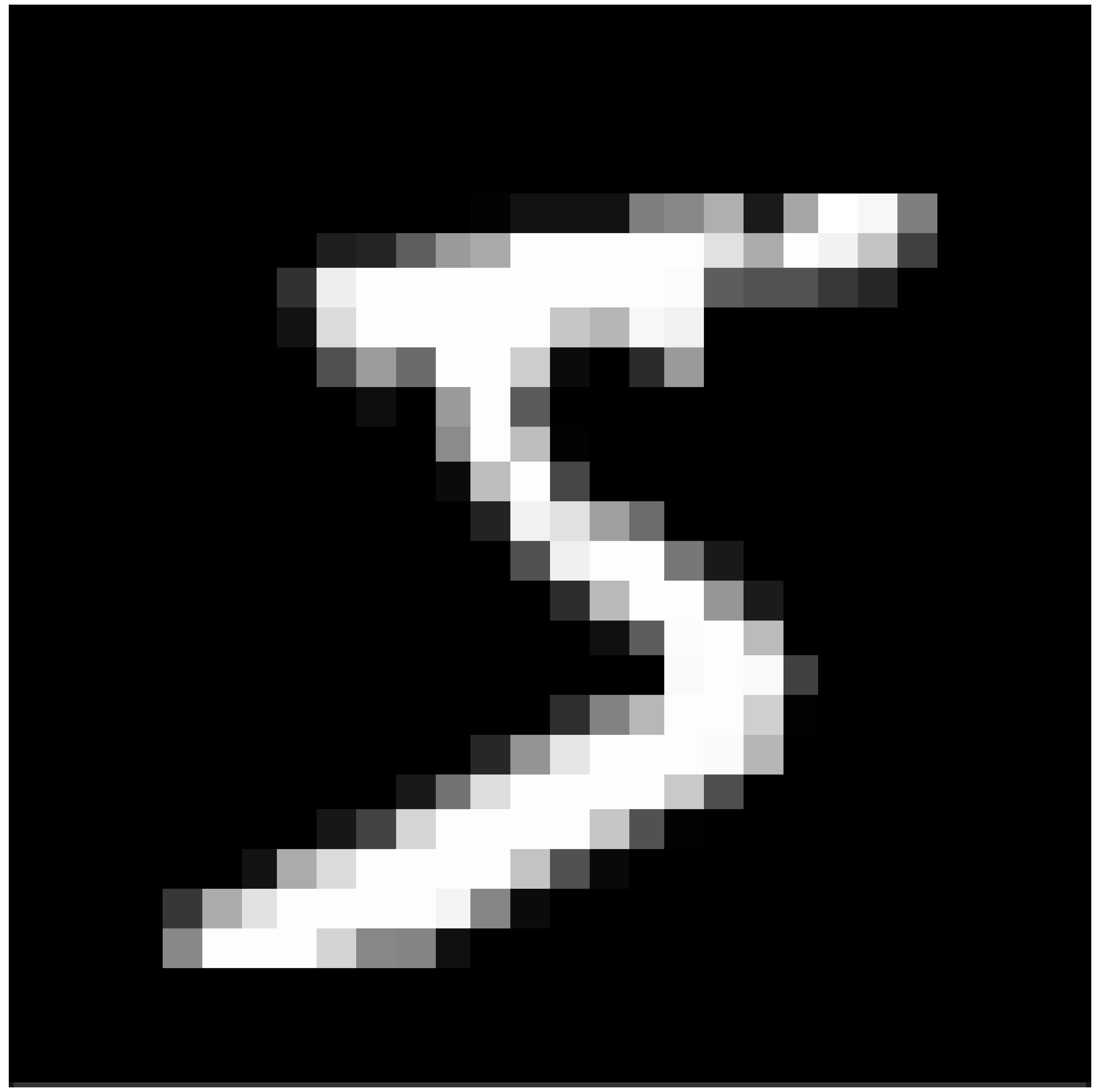
Recap of the Exercise

What just happened?

1. Loaded and visualized our data
2. Edited our data (reshaped, normalized, to categorical)
3. Created our model
4. Compiled our model
5. Trained the model on our data

Data Preparation

Input as an Array



28

28

[0,0,0,24,75,184,185,78,32,55,0,0,0…]

An Untrained Model