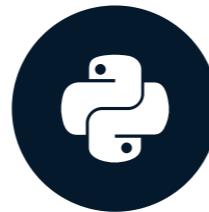


# The need for optimization

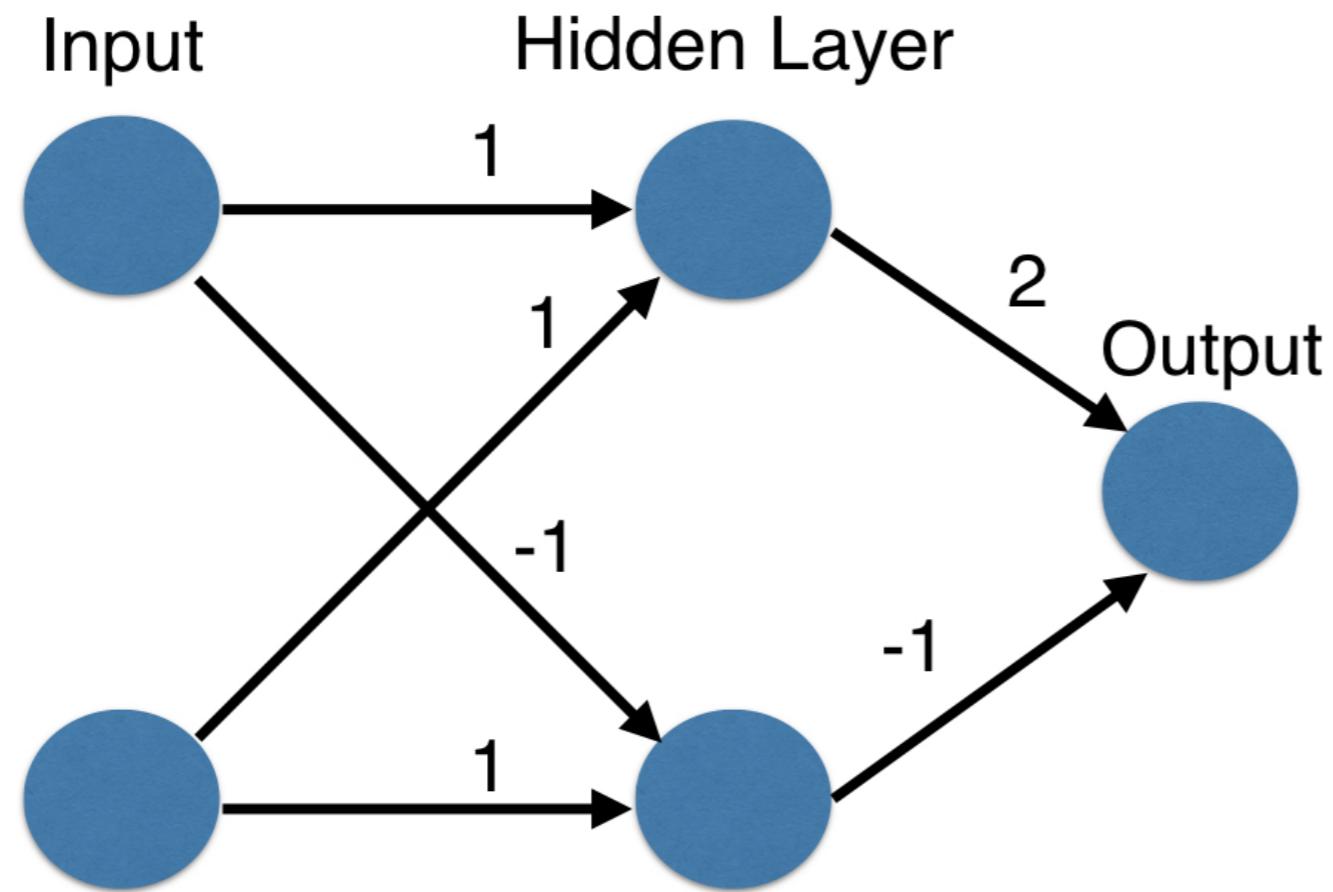
INTRODUCTION TO DEEP LEARNING IN PYTHON



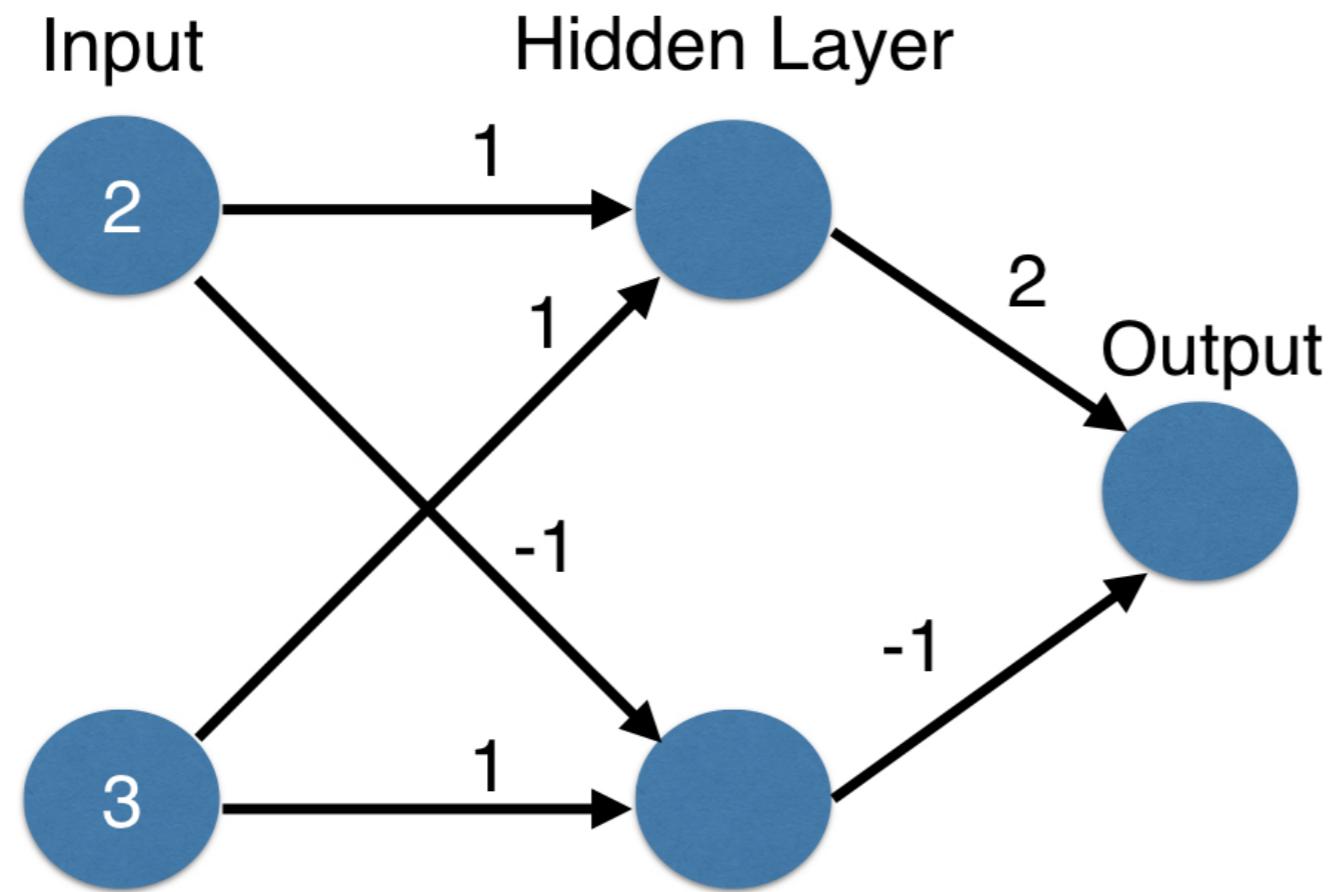
Dan Becker

Data Scientist and contributor to Keras  
and TensorFlow libraries

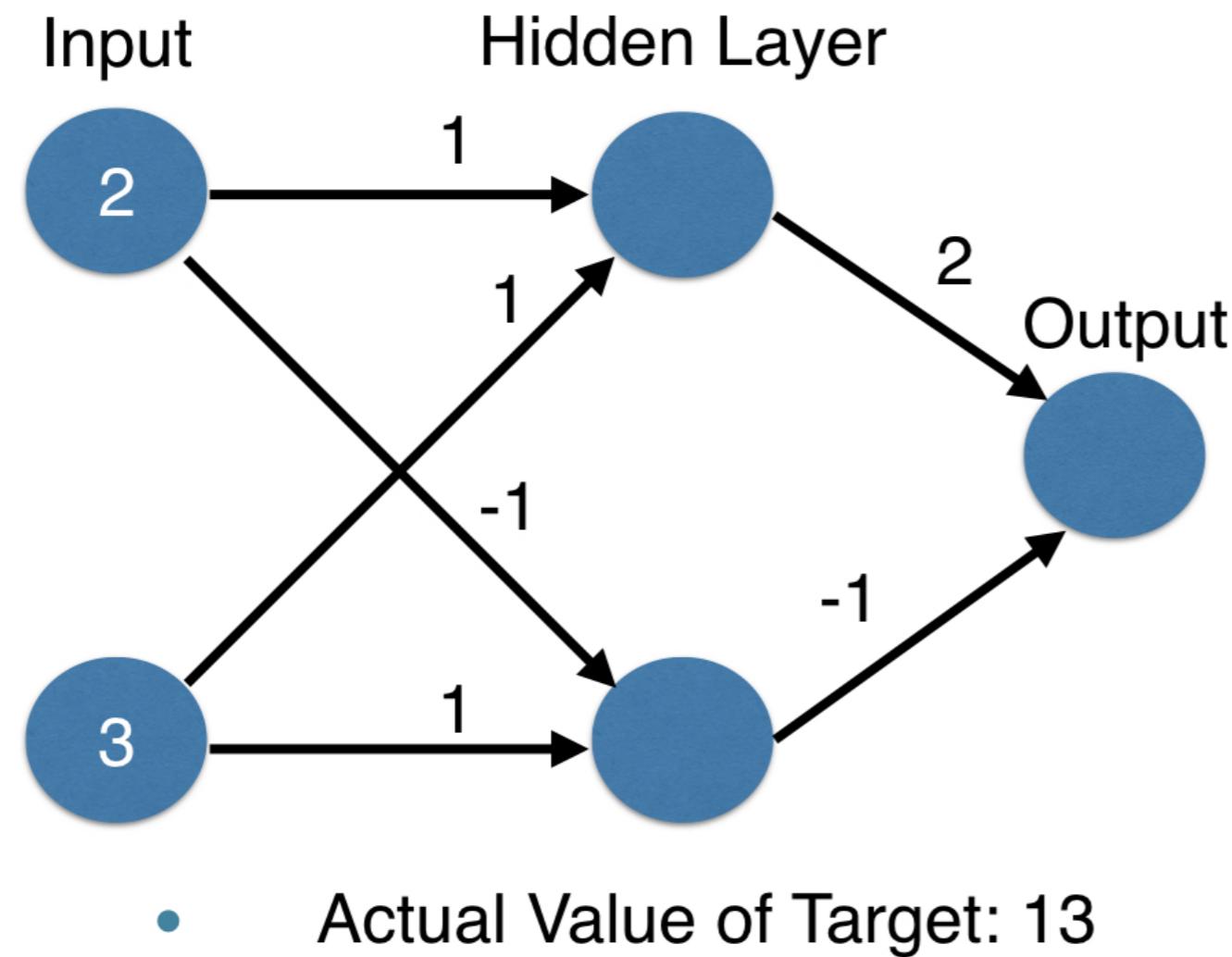
# A baseline neural network



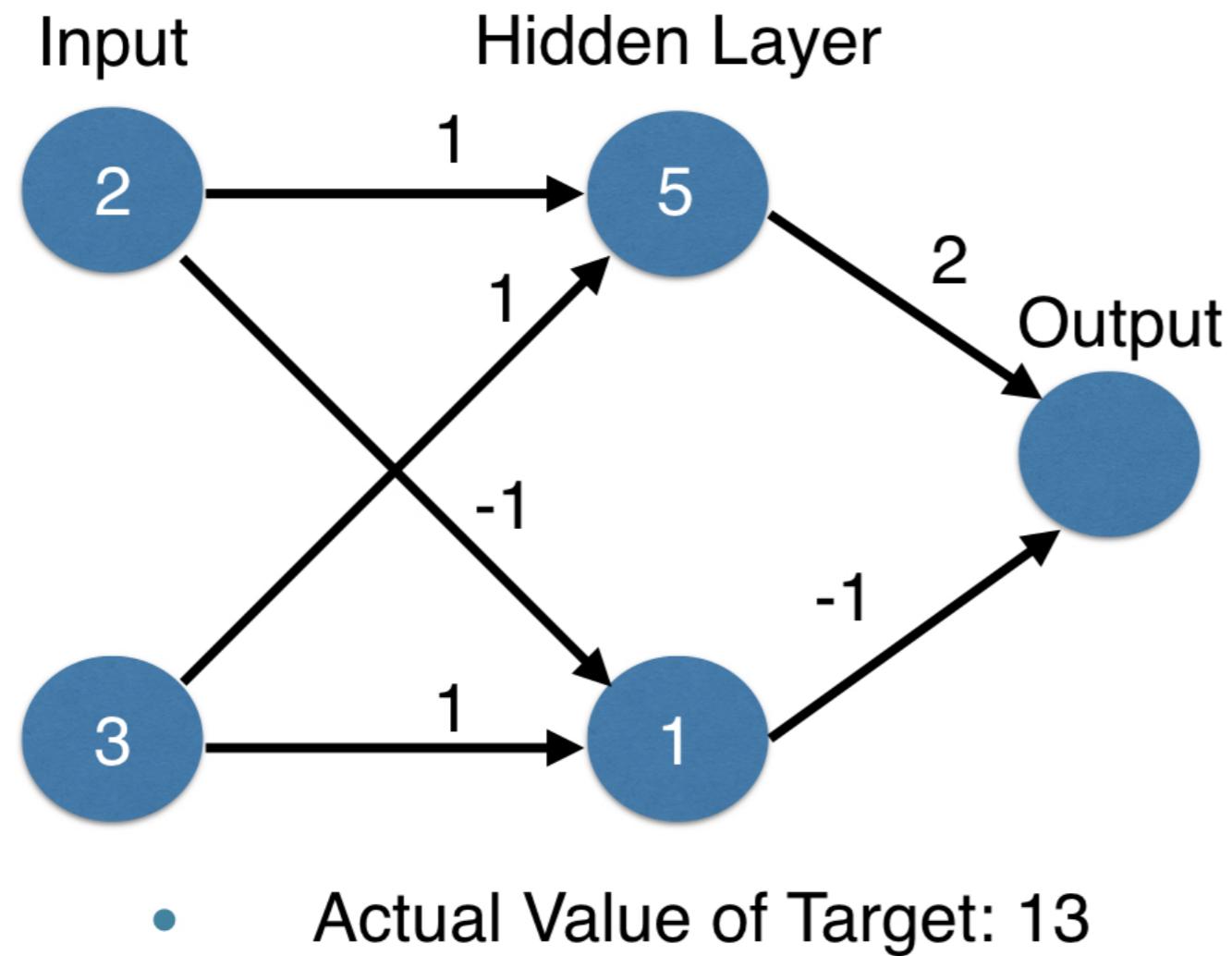
# A baseline neural network



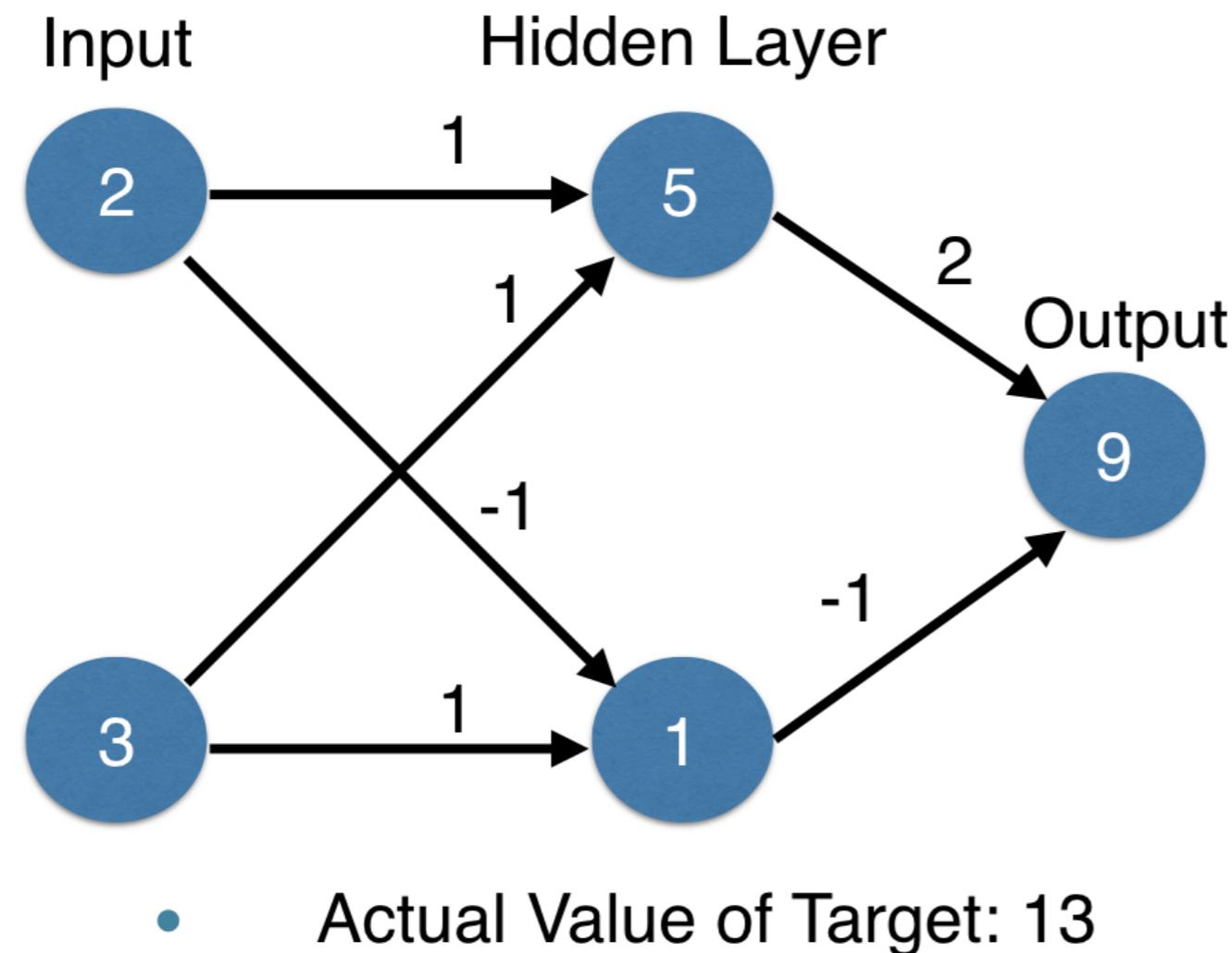
# A baseline neural network



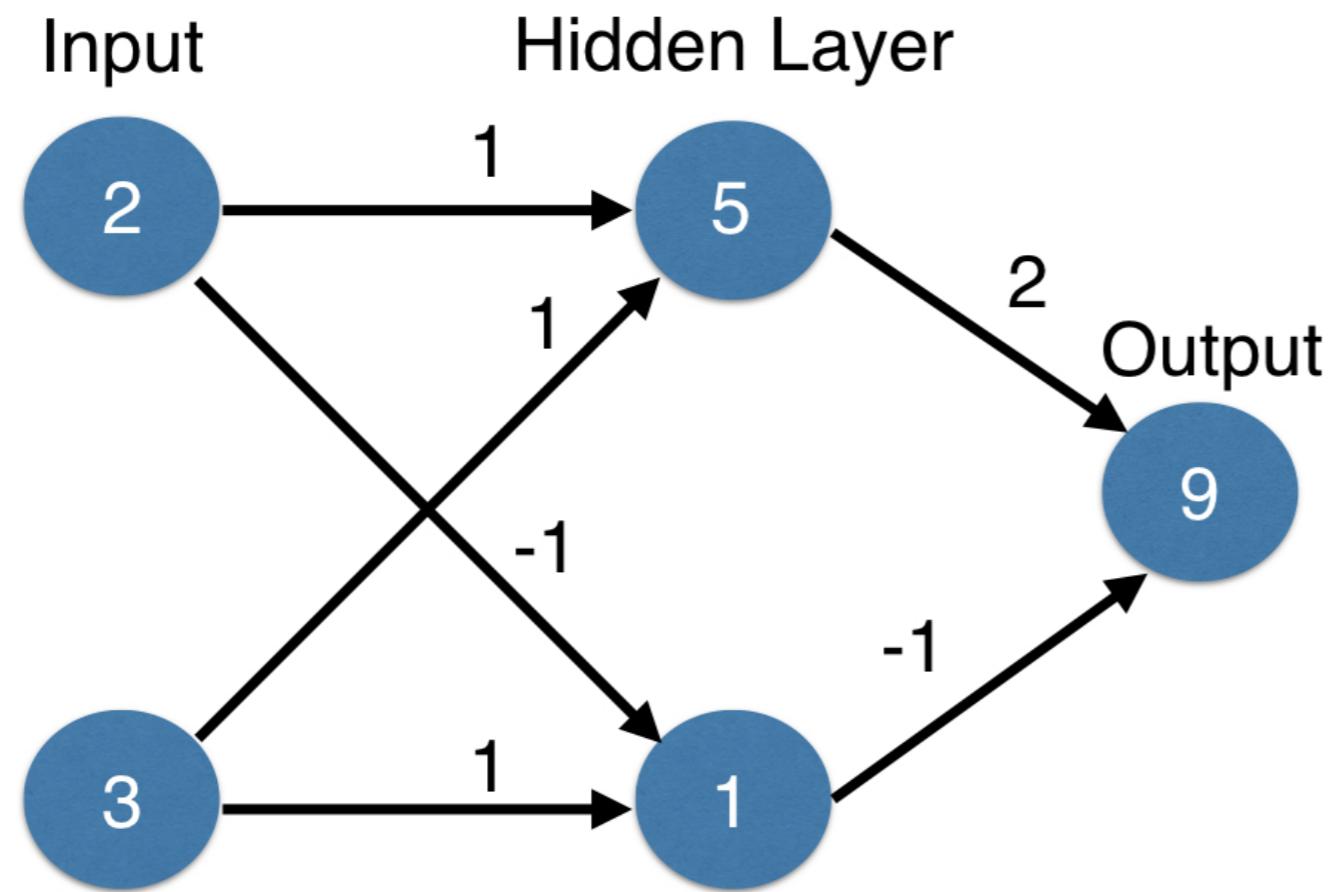
# A baseline neural network



# A baseline neural network

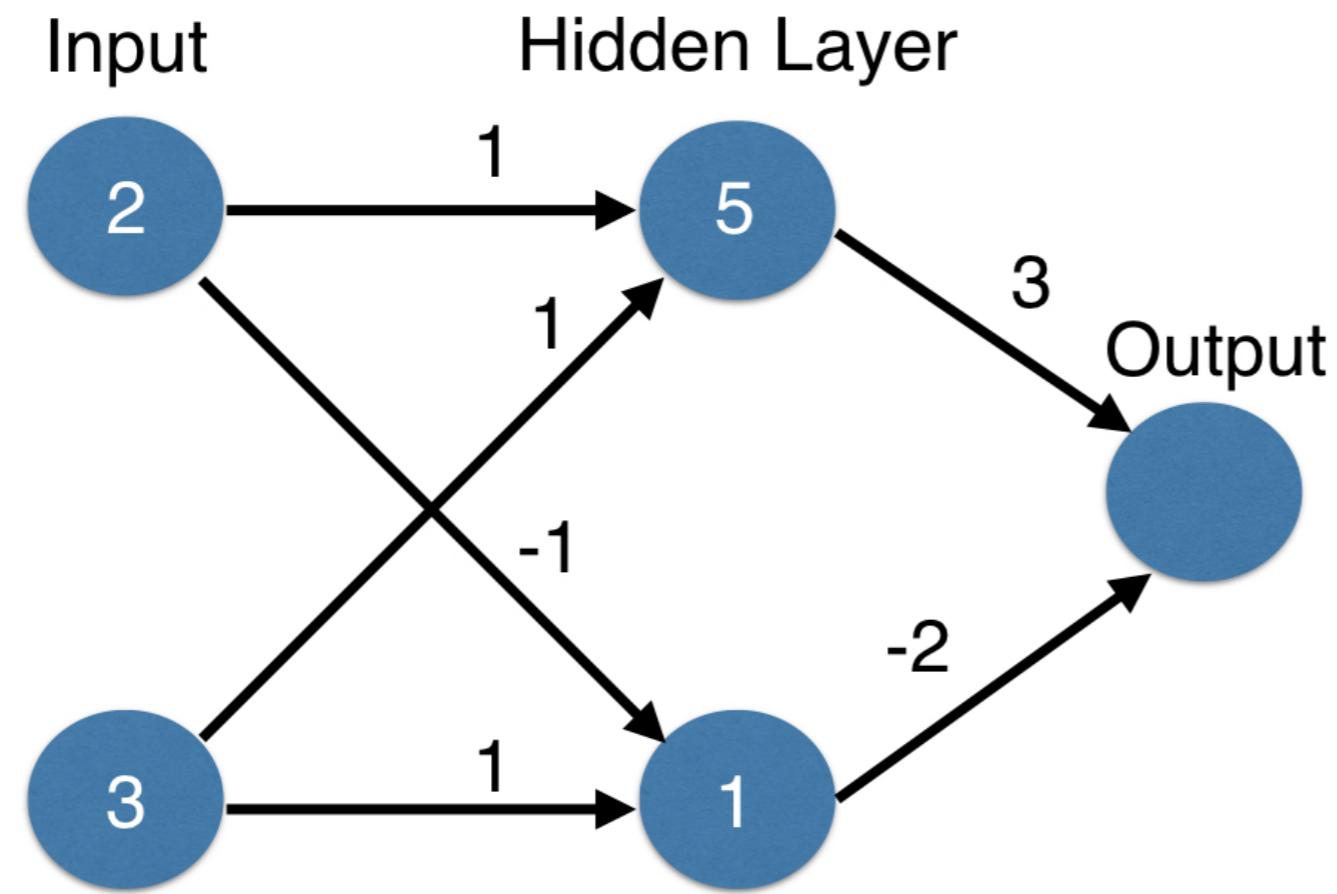


# A baseline neural network

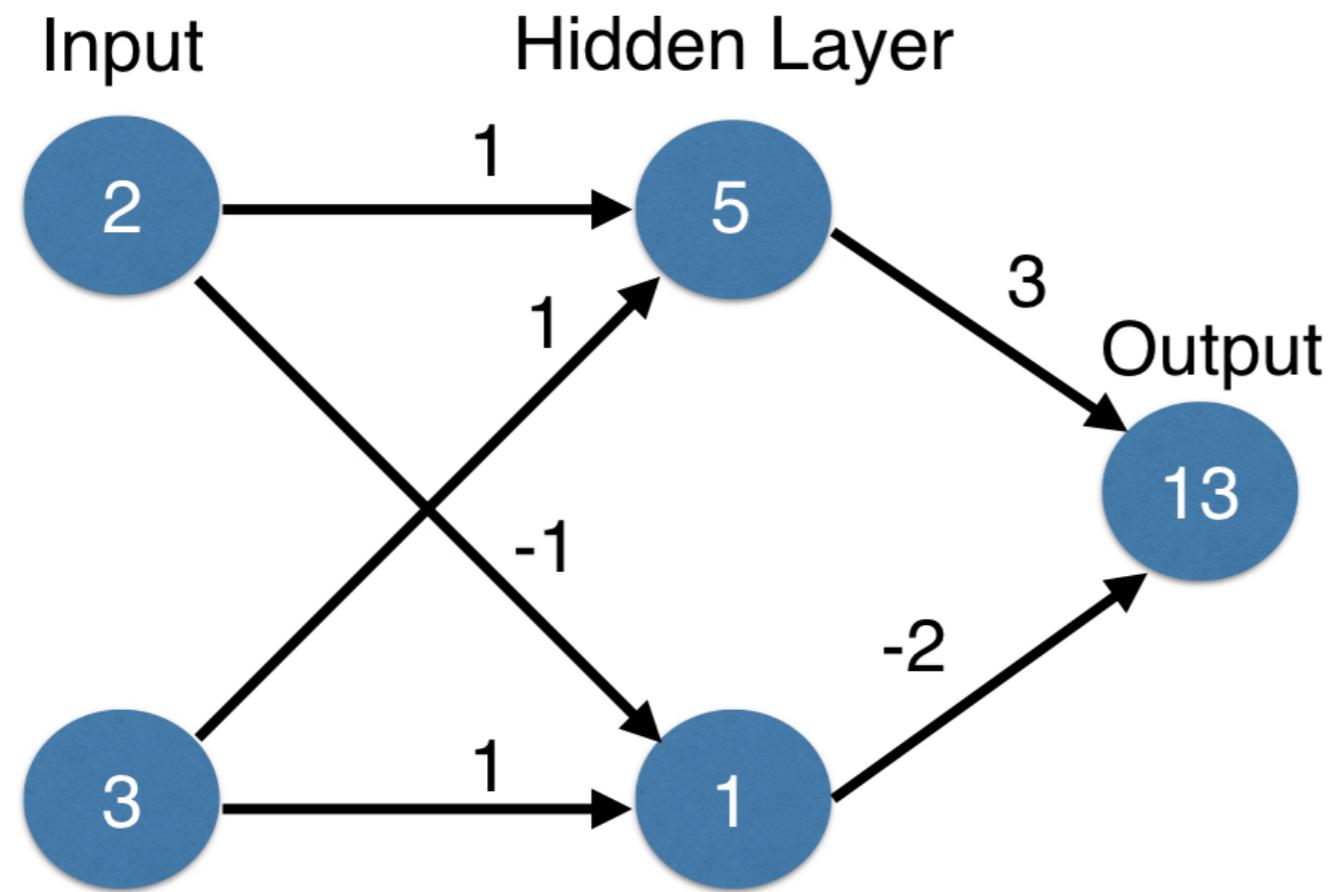


- Actual Value of Target: 13
- Error: Predicted - Actual = -4

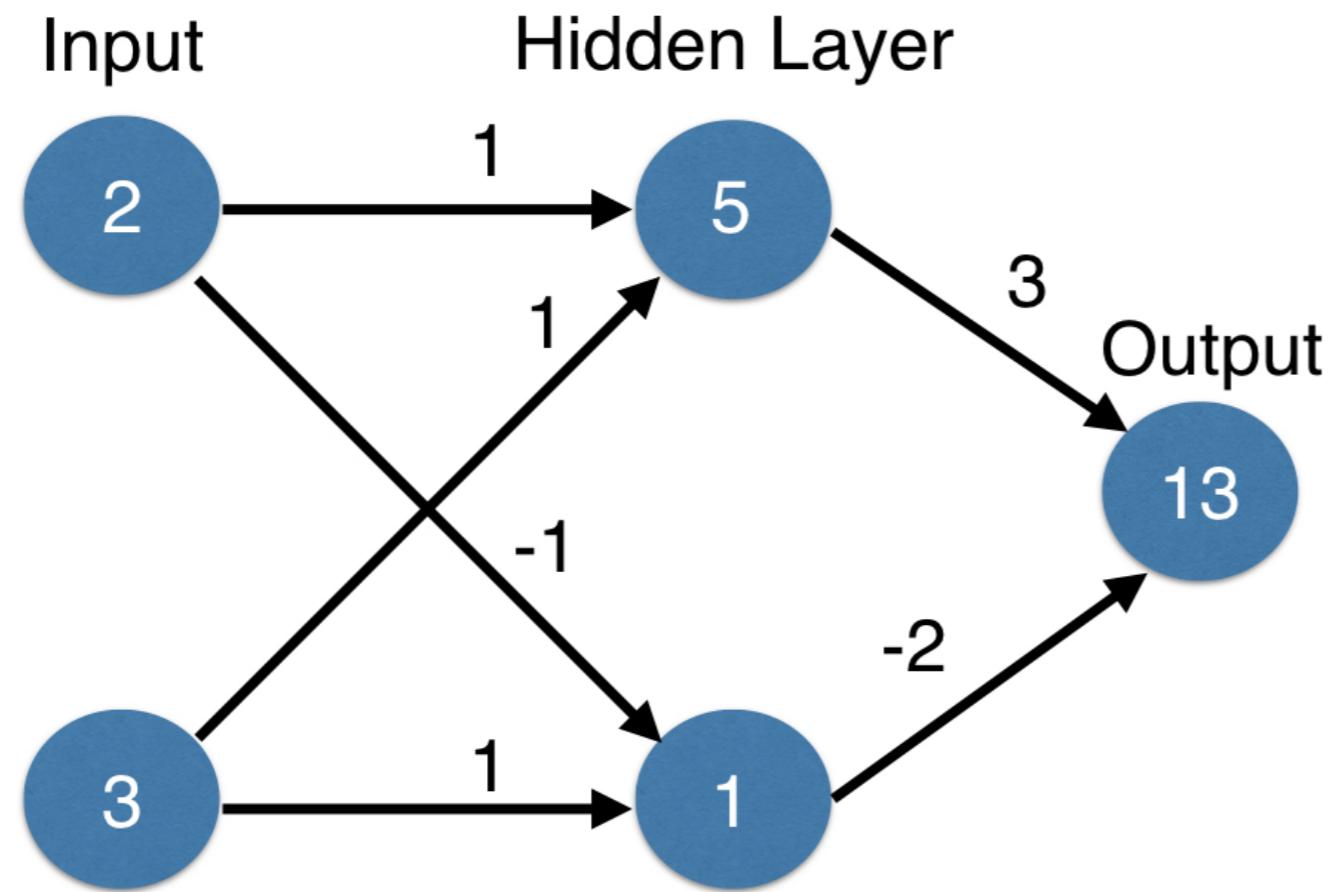
# A baseline neural network



# A baseline neural network



# A baseline neural network



- Actual Value of Target: 13
- Error: Predicted - Actual = 0

# Predictions with multiple points

- Making accurate predictions gets harder with more points
- At any set of weights, there are many values of the error
- ... corresponding to the many points we make predictions for

# Loss function

- Aggregates errors in predictions from many data points into single number
- Measure of model's predictive performance

# Squared error loss function

Prediction	Actual	Error	Squared Error
10	20	-10	100
8	3	5	25
6	1	5	25

# Squared error loss function

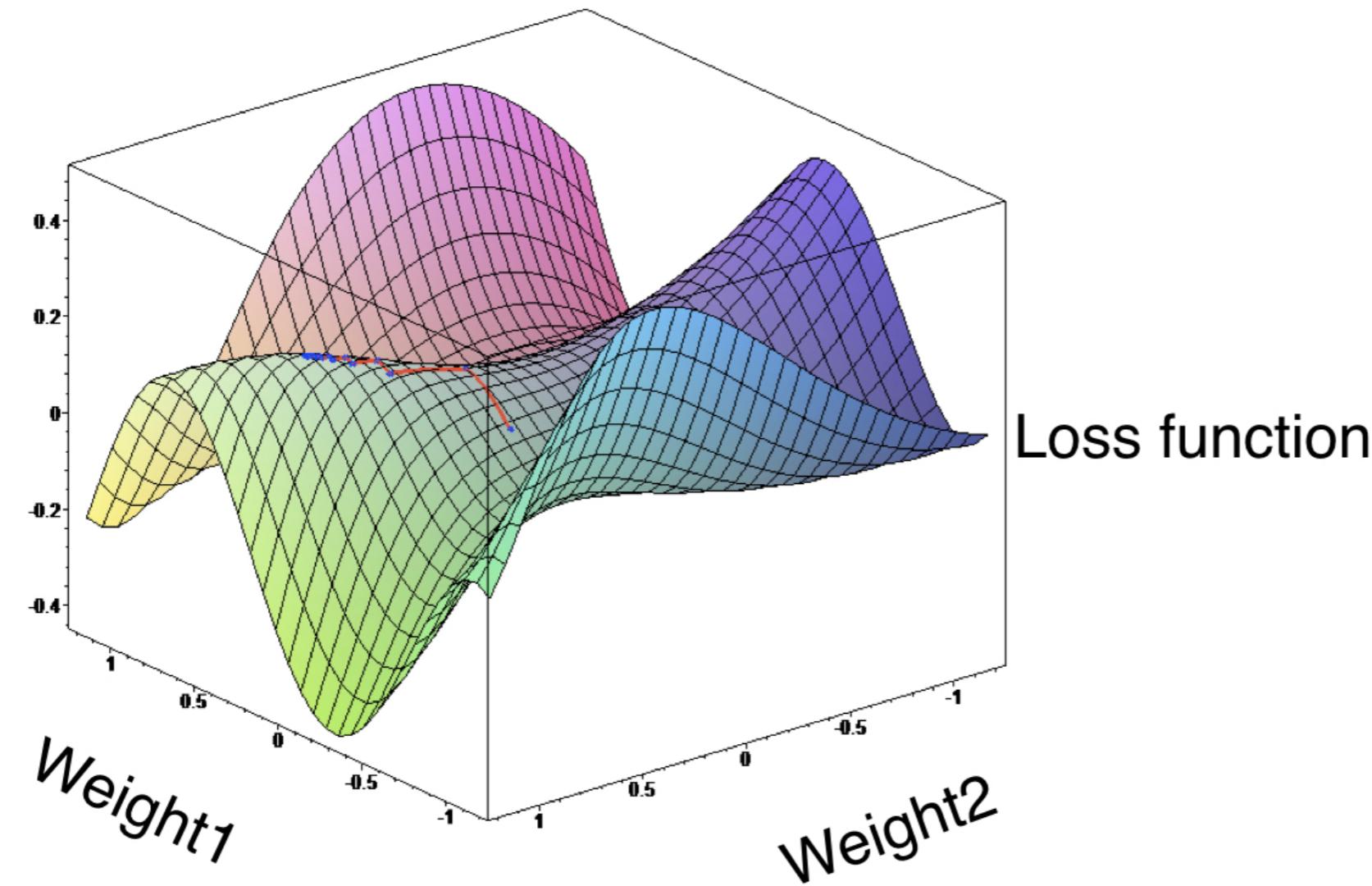
Prediction	Actual	Error	Squared Error
10	20	-10	100
8	3	5	25
6	1	5	25

# Squared error loss function

Prediction	Actual	Error	Squared Error
10	20	-10	100
8	3	5	25
6	1	5	25

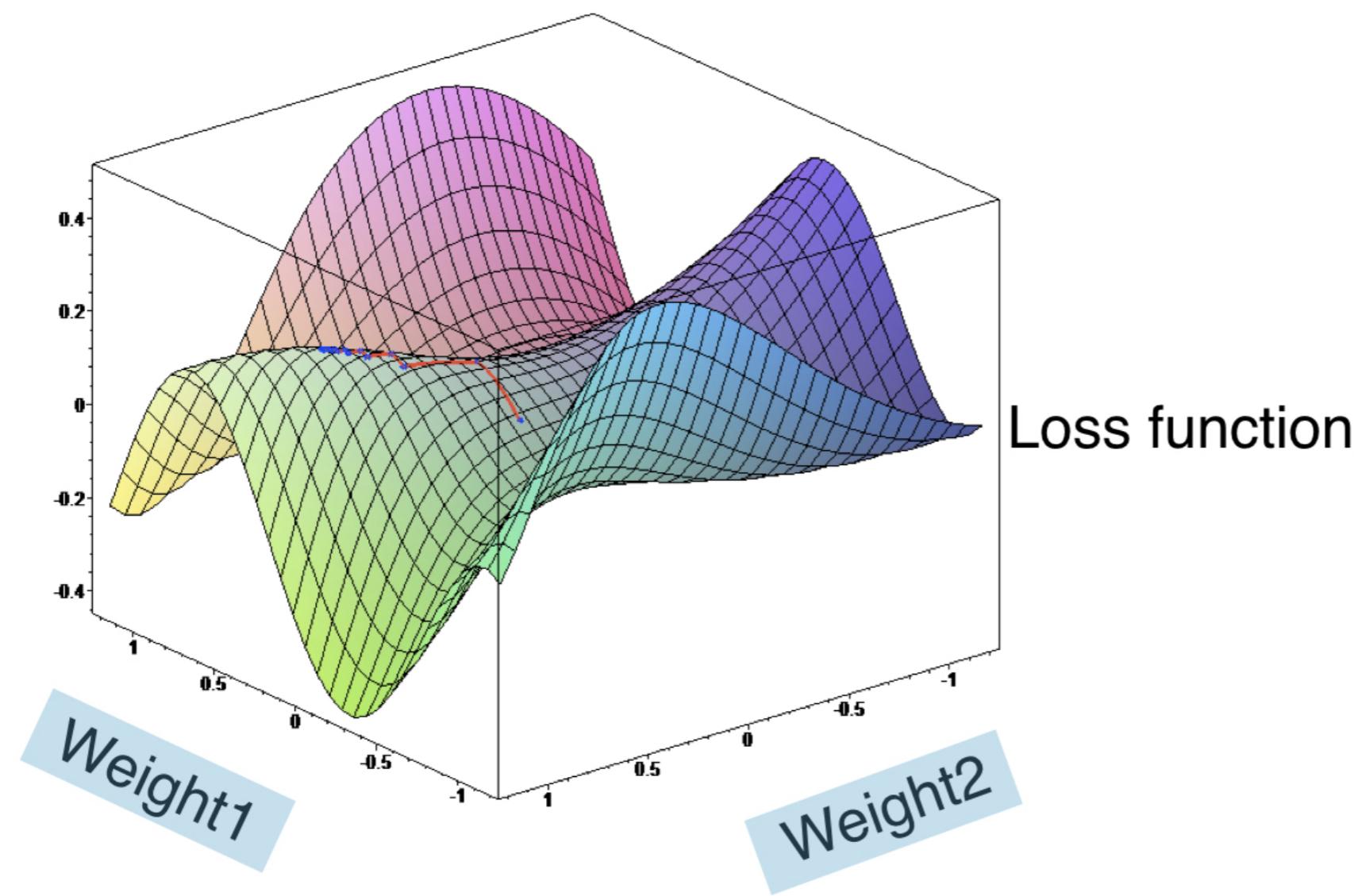
- Total Squared Error: 150
- Mean Squared Error: 50

# Loss function



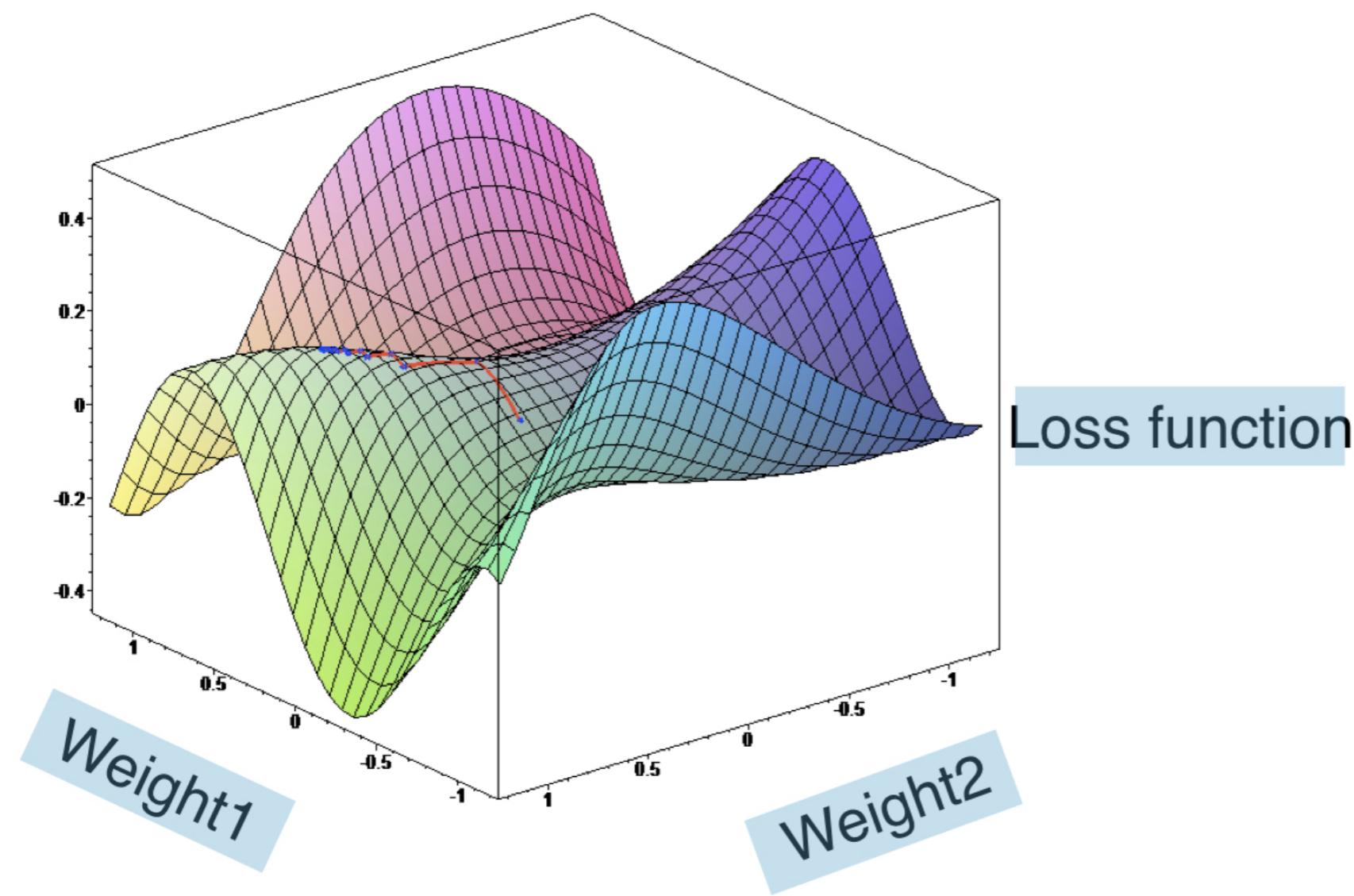
Loss function

# Loss function



Loss function

# Loss function



# Loss function

- Lower loss function value means a better model
- Goal: Find the weights that give the lowest value for the loss function
- Gradient descent

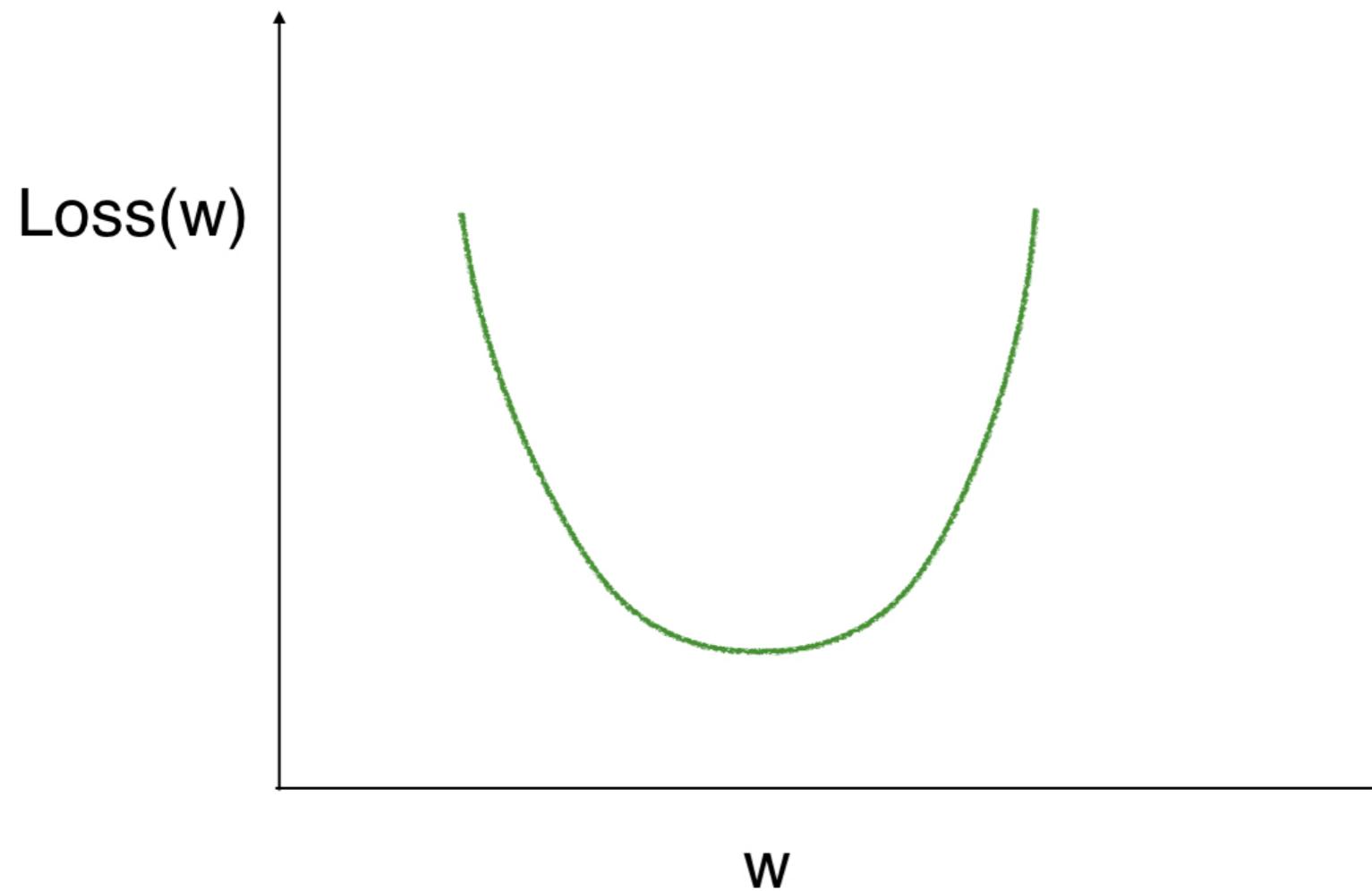
# Gradient descent

- Imagine you are in a pitch dark field
- Want to find the lowest point
- Feel the ground to see how it slopes
- Take a small step downhill
- Repeat until it is uphill in every direction

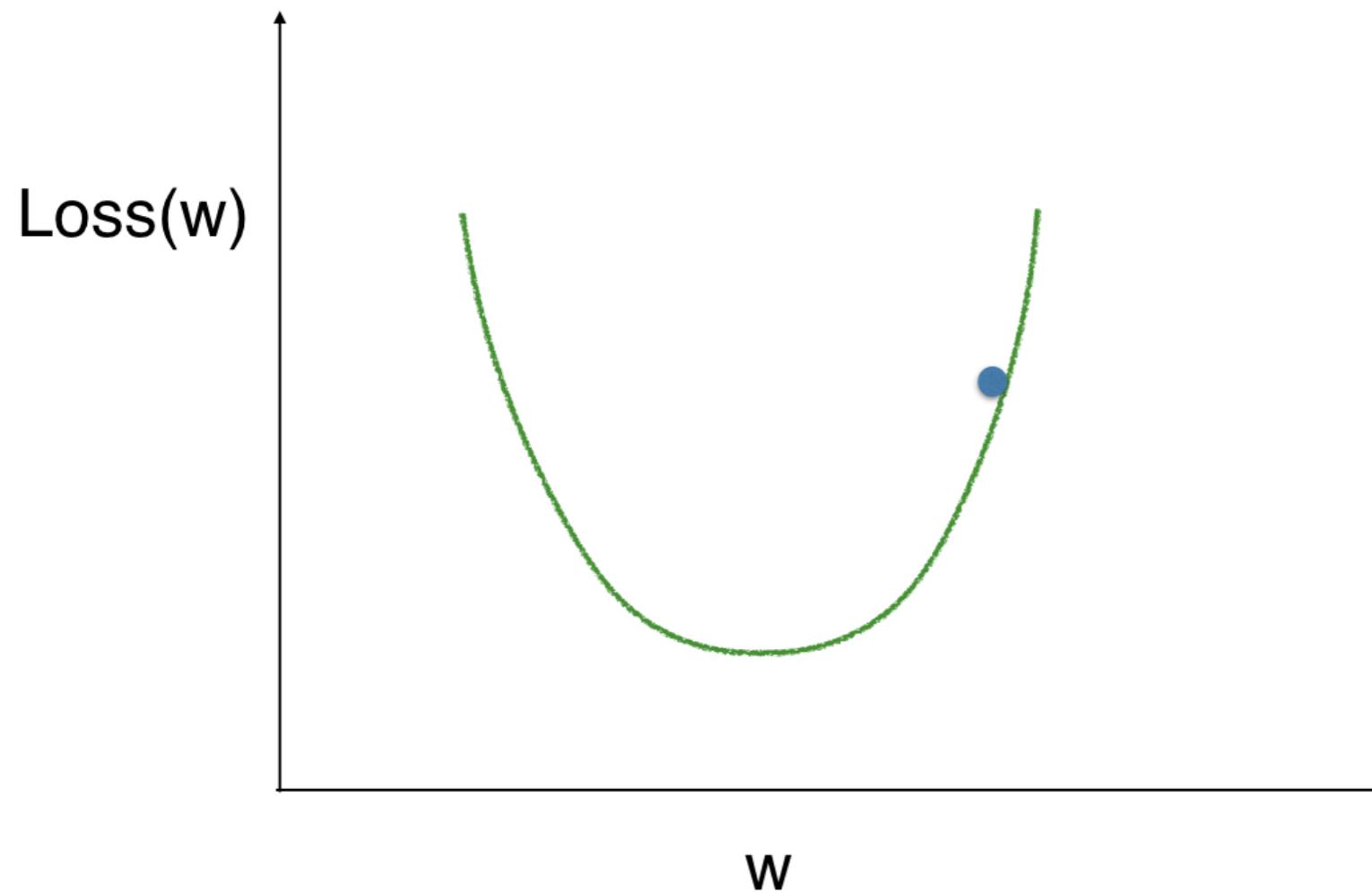
# Gradient descent steps

- Start at random point
- Until you are somewhere flat:
  - Find the slope
  - Take a step downhill

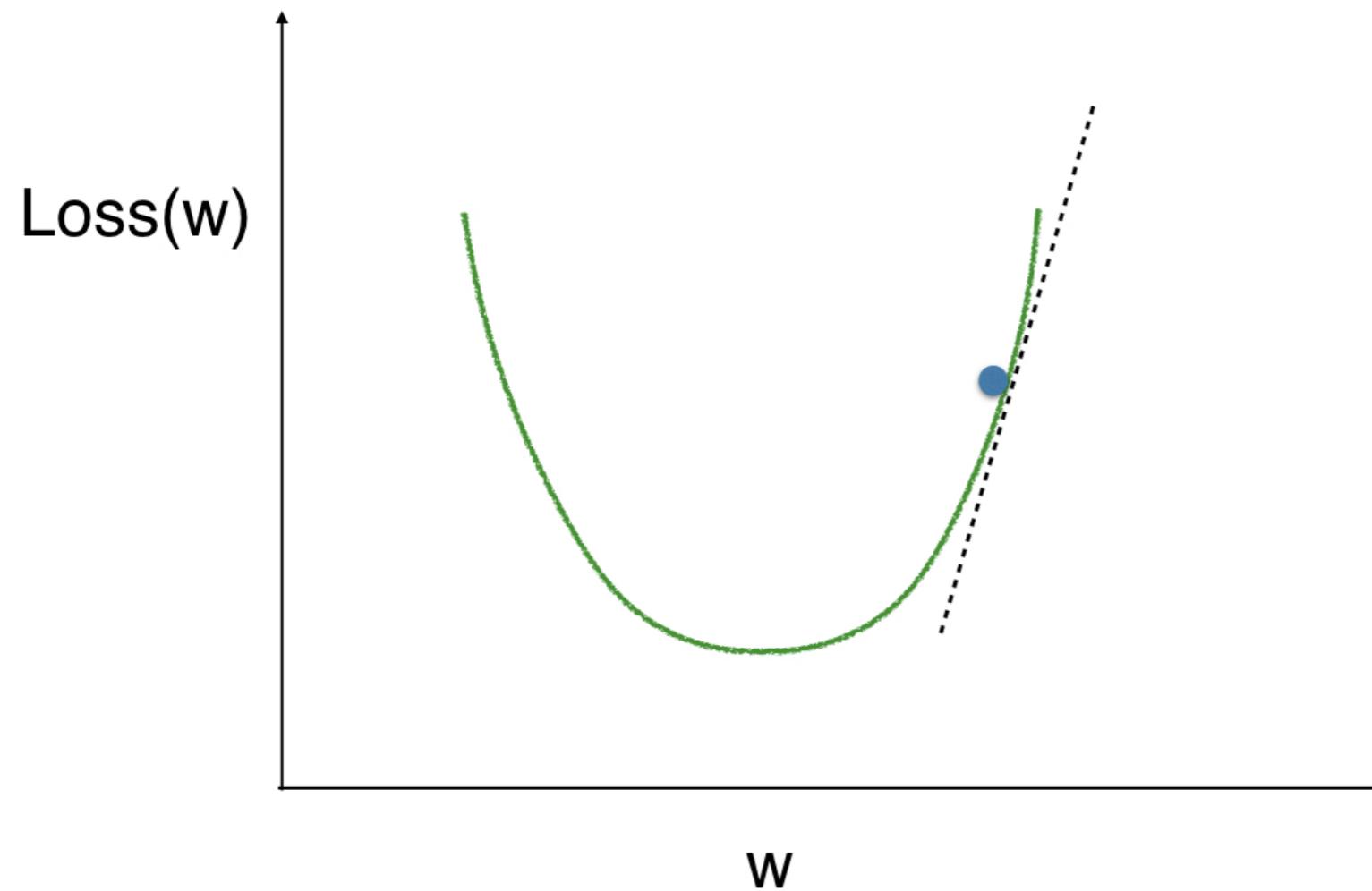
# Optimizing a model with a single weight



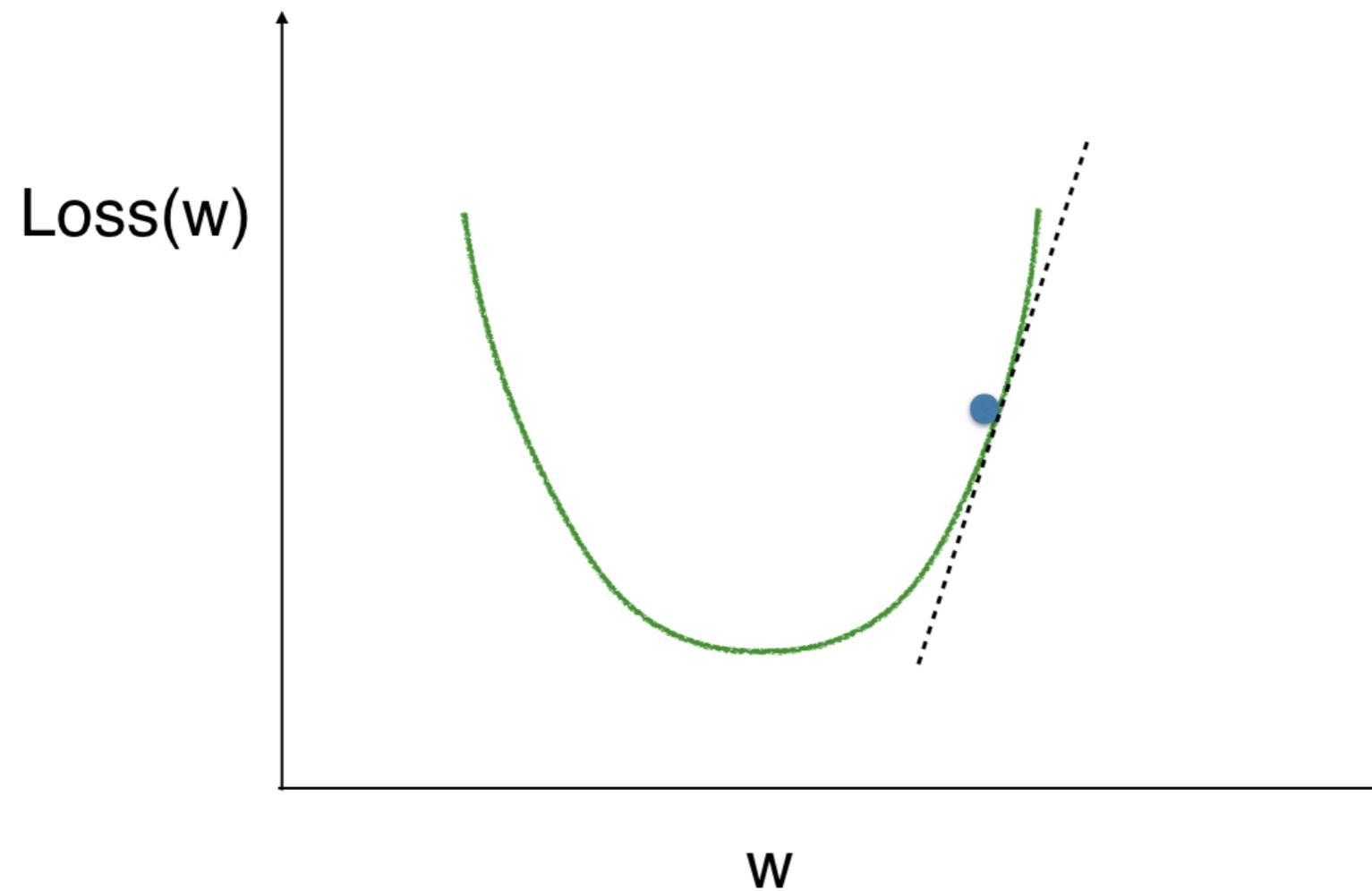
# Optimizing a model with a single weight



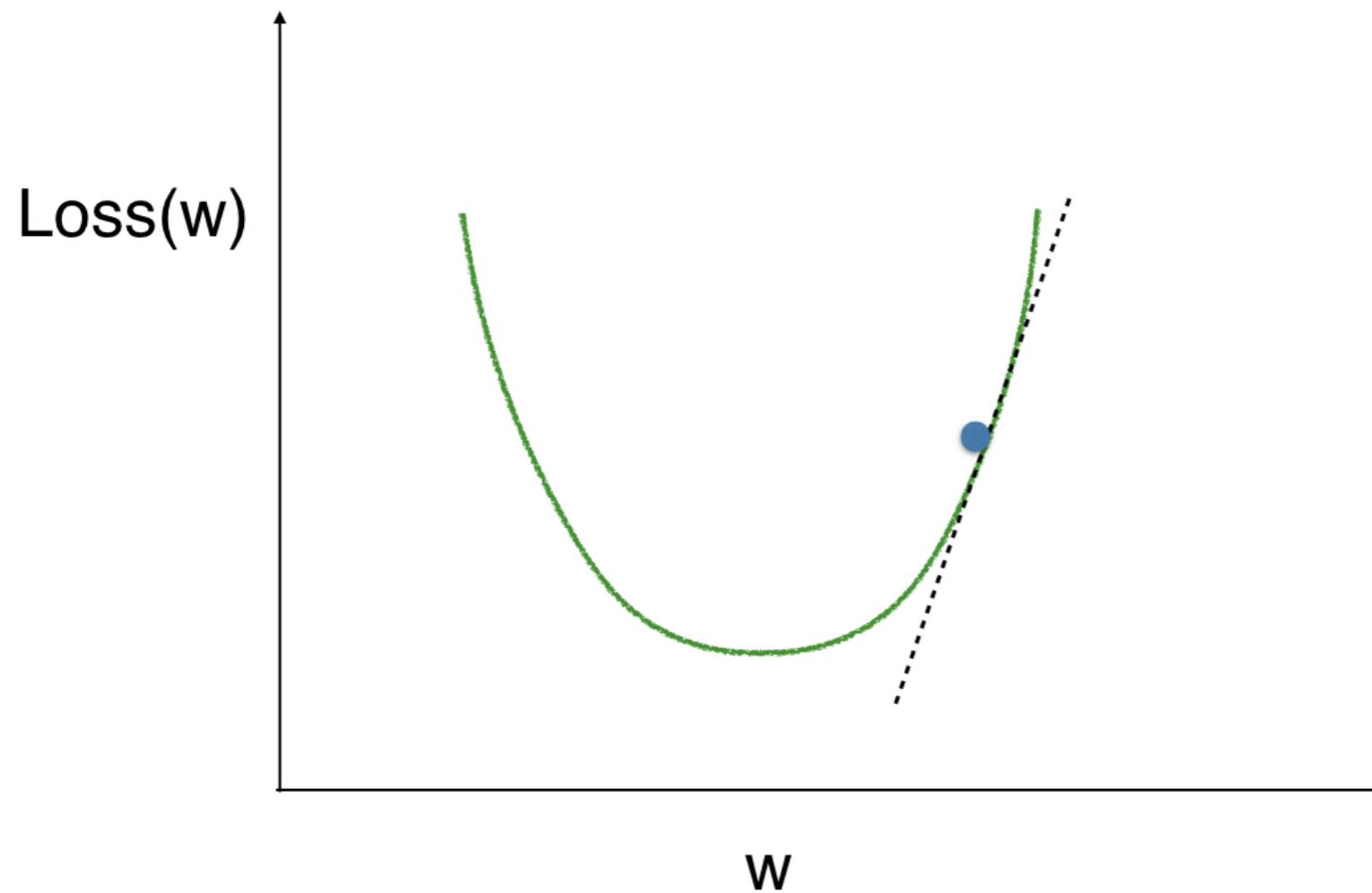
# Optimizing a model with a single weight



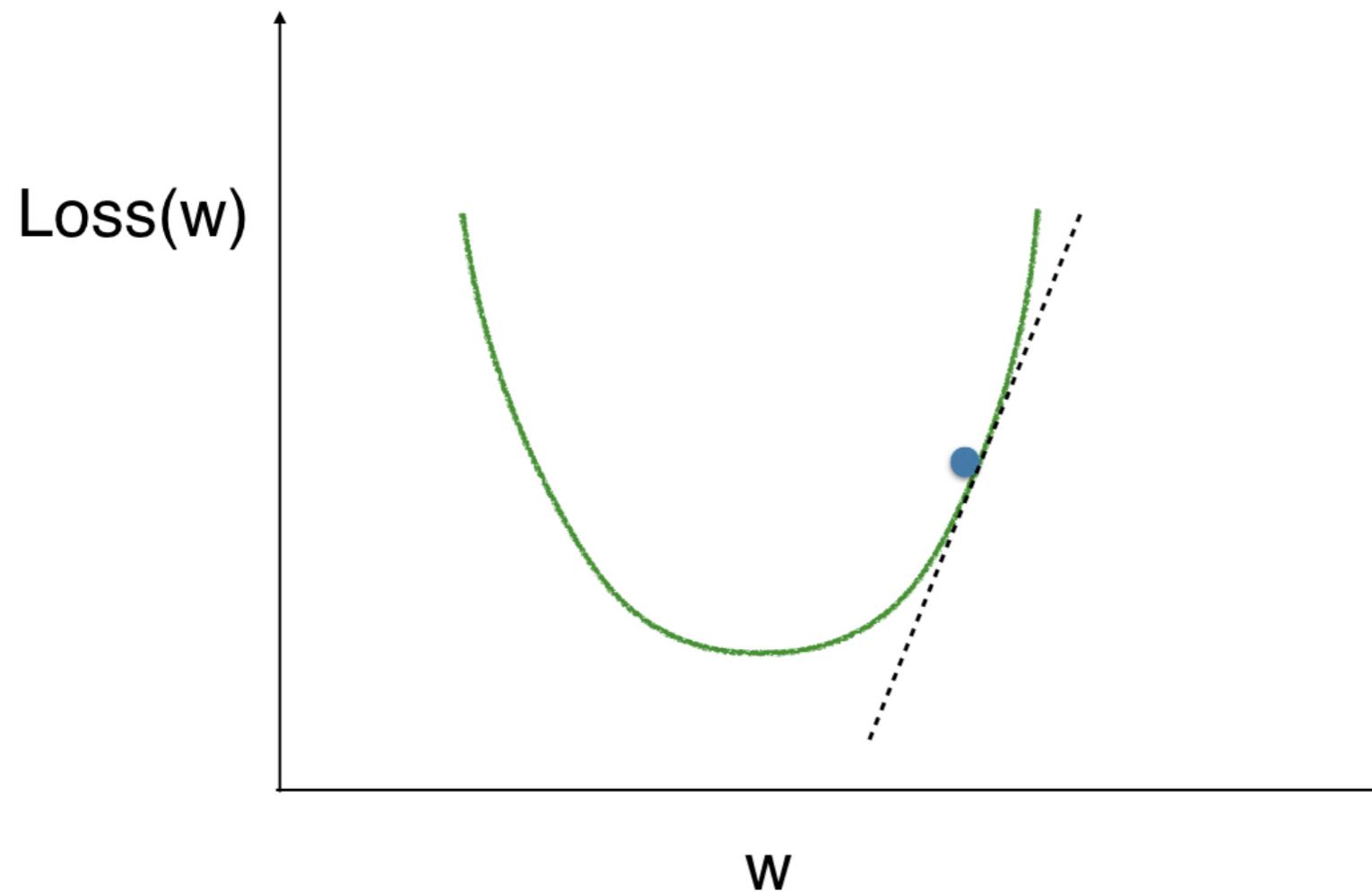
# Optimizing a model with a single weight



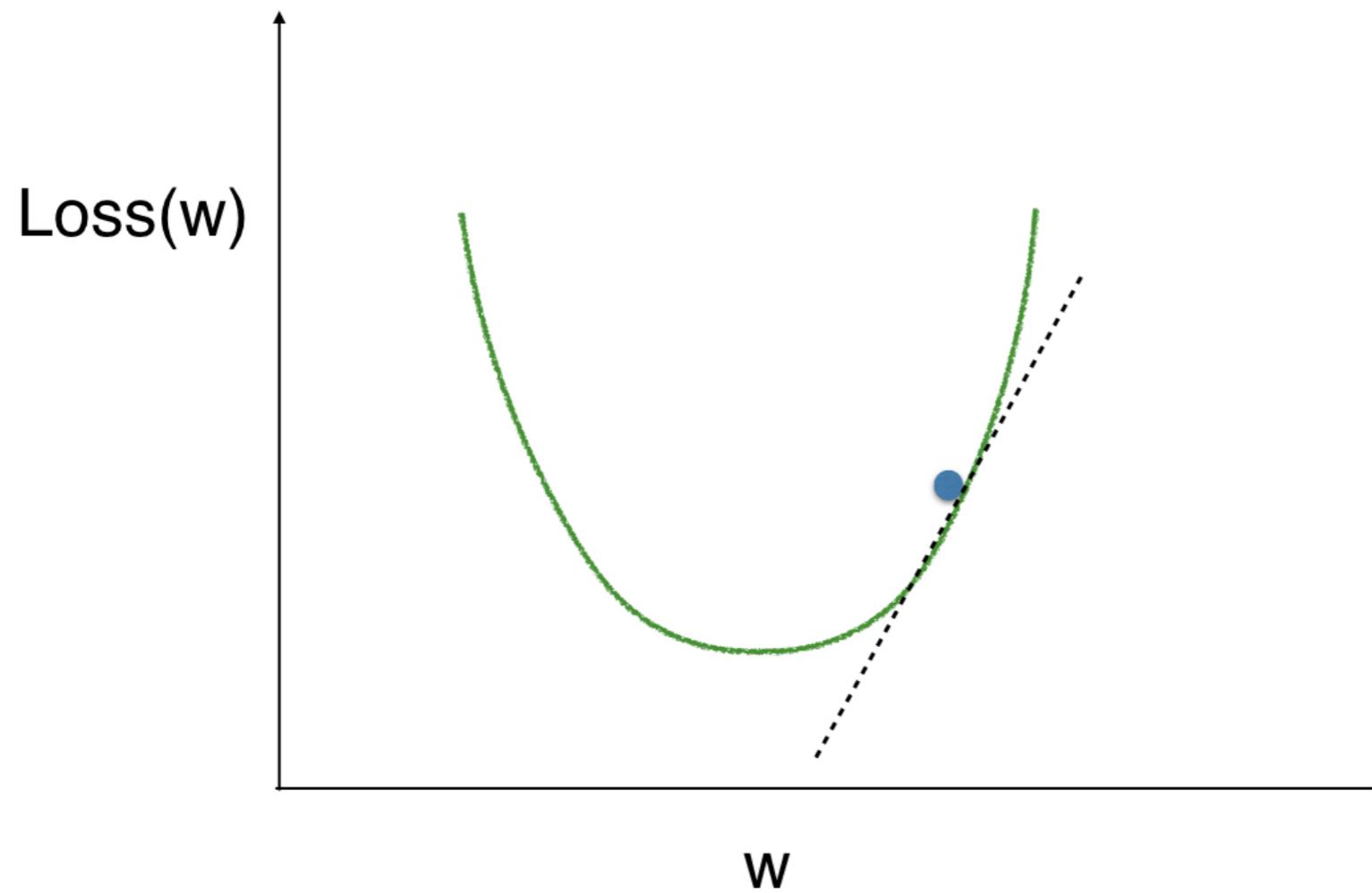
# Optimizing a model with a single weight



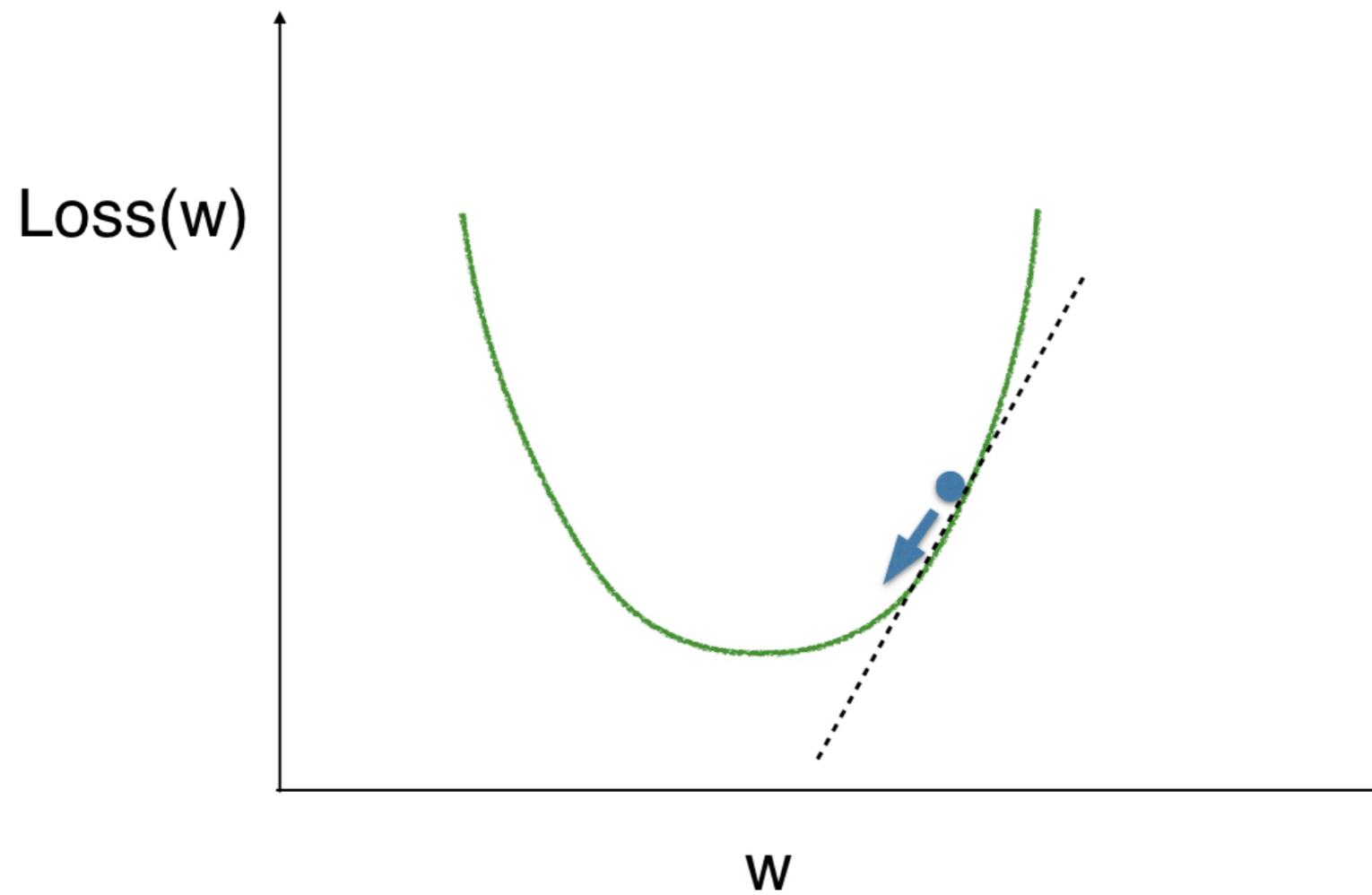
# Optimizing a model with a single weight



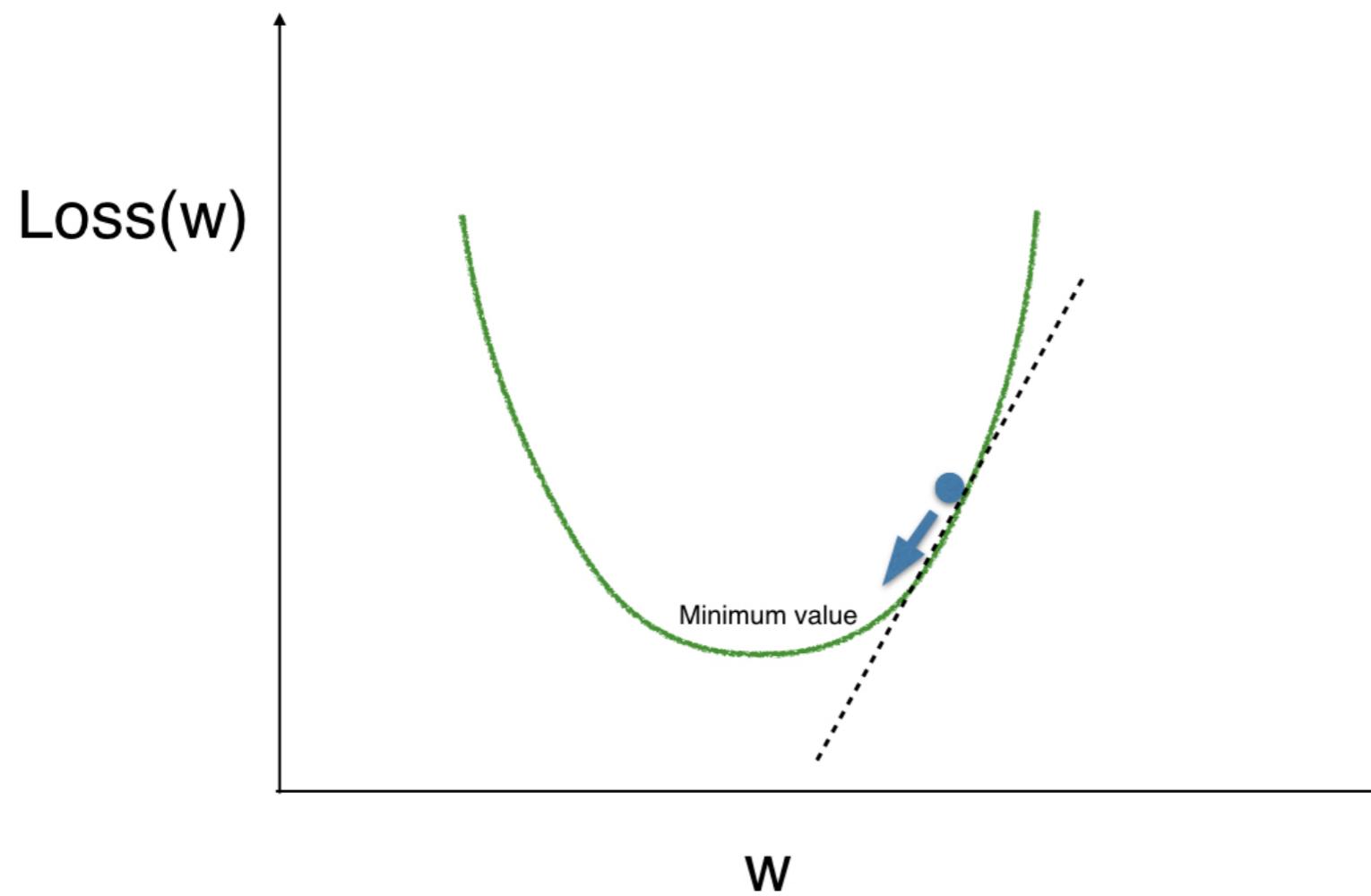
# Optimizing a model with a single weight



# Optimizing a model with a single weight



# Optimizing a model with a single weight

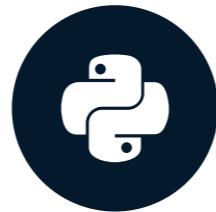


# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING IN PYTHON**

# Gradient descent

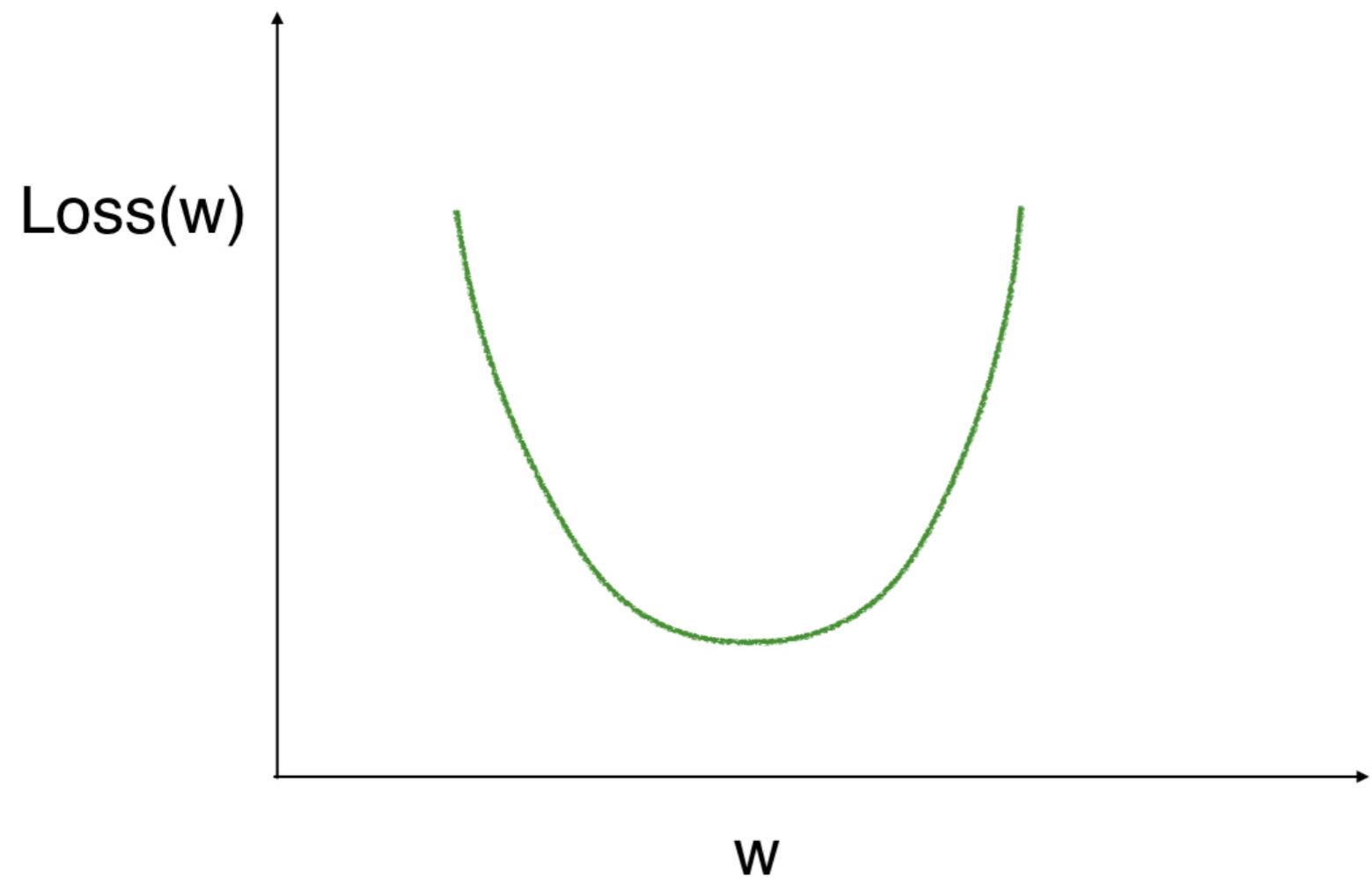
INTRODUCTION TO DEEP LEARNING IN PYTHON



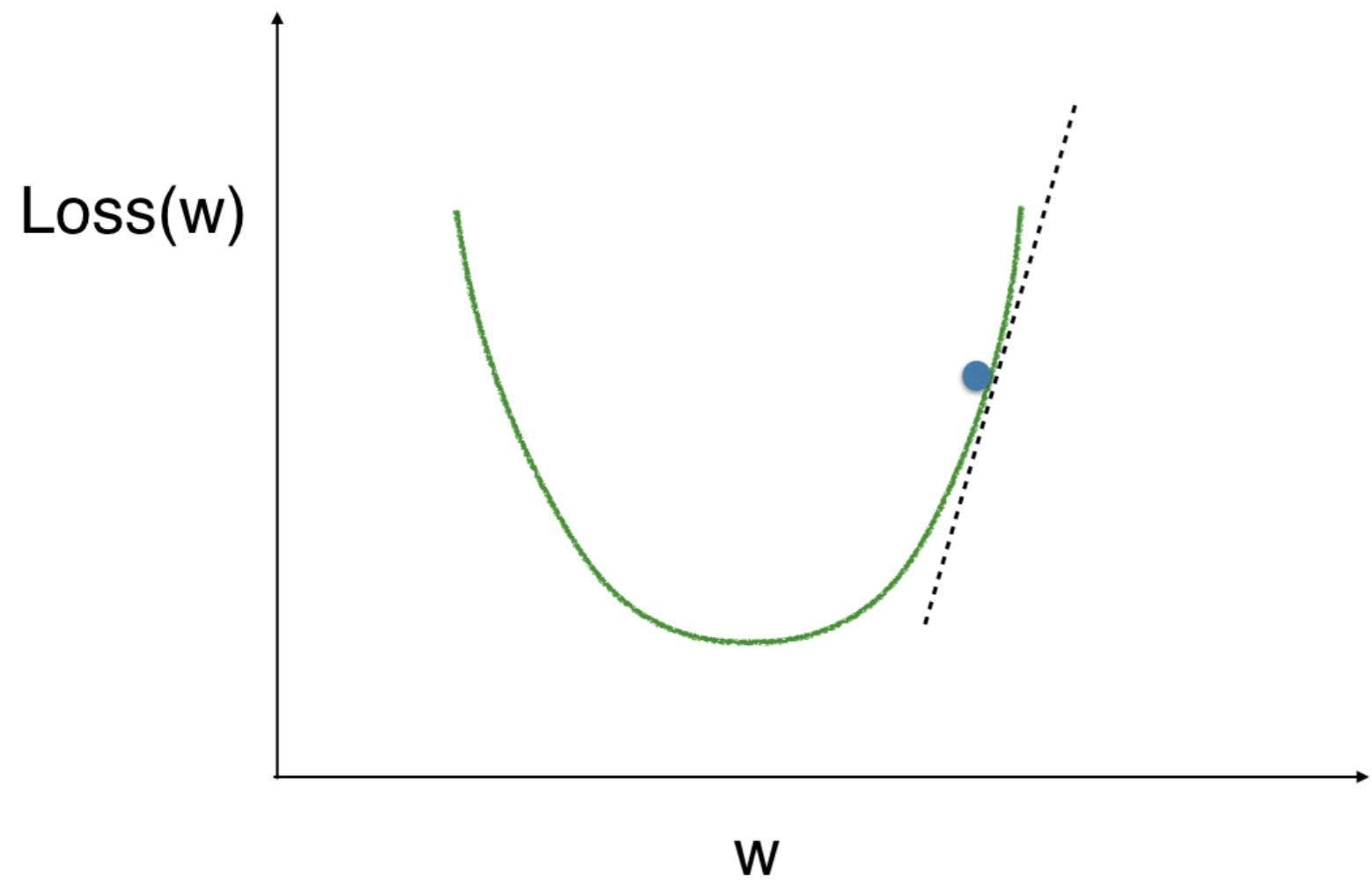
Dan Becker

Data Scientist and contributor to Keras  
and TensorFlow libraries

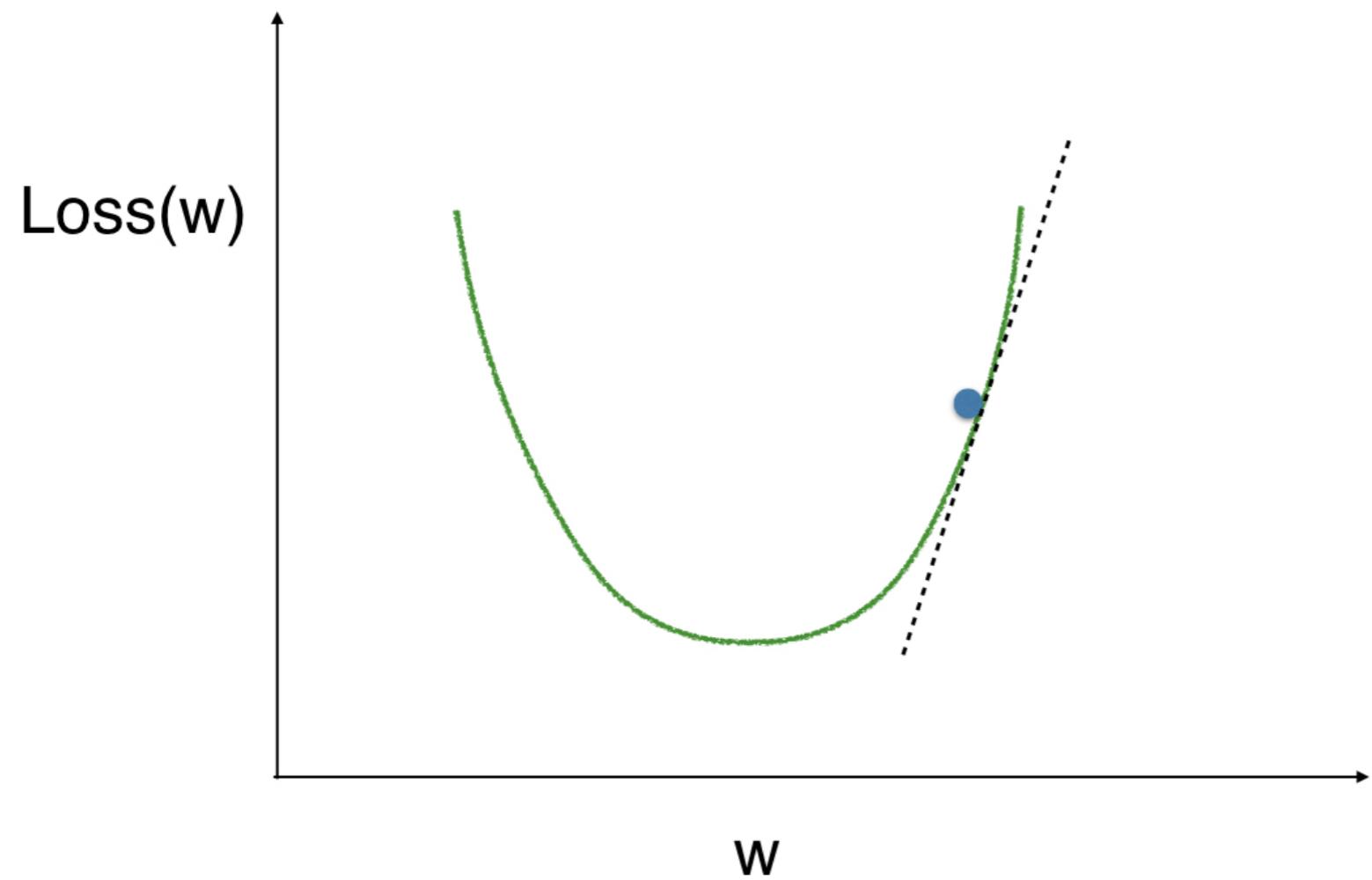
# Gradient descent



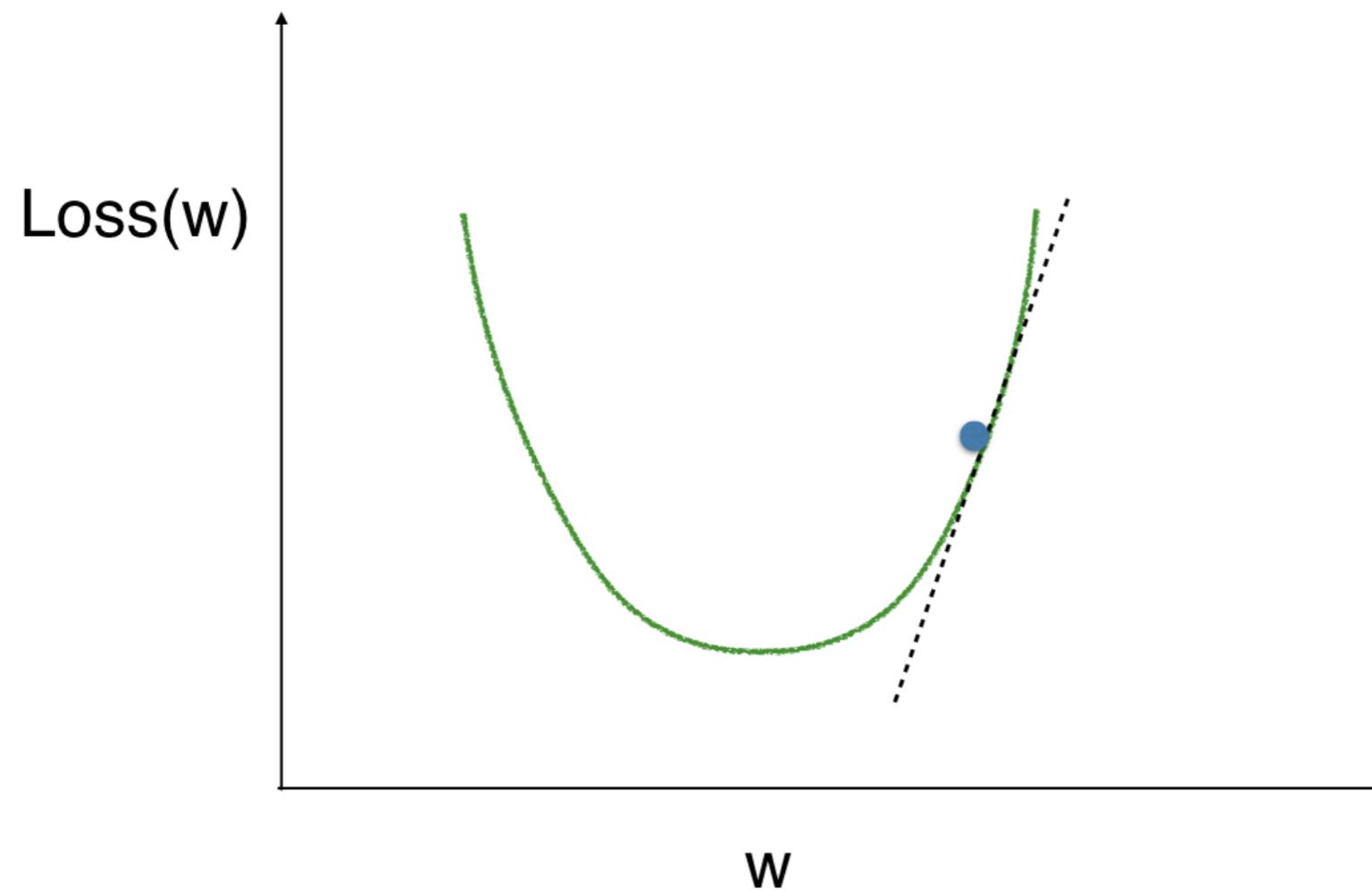
# Gradient descent



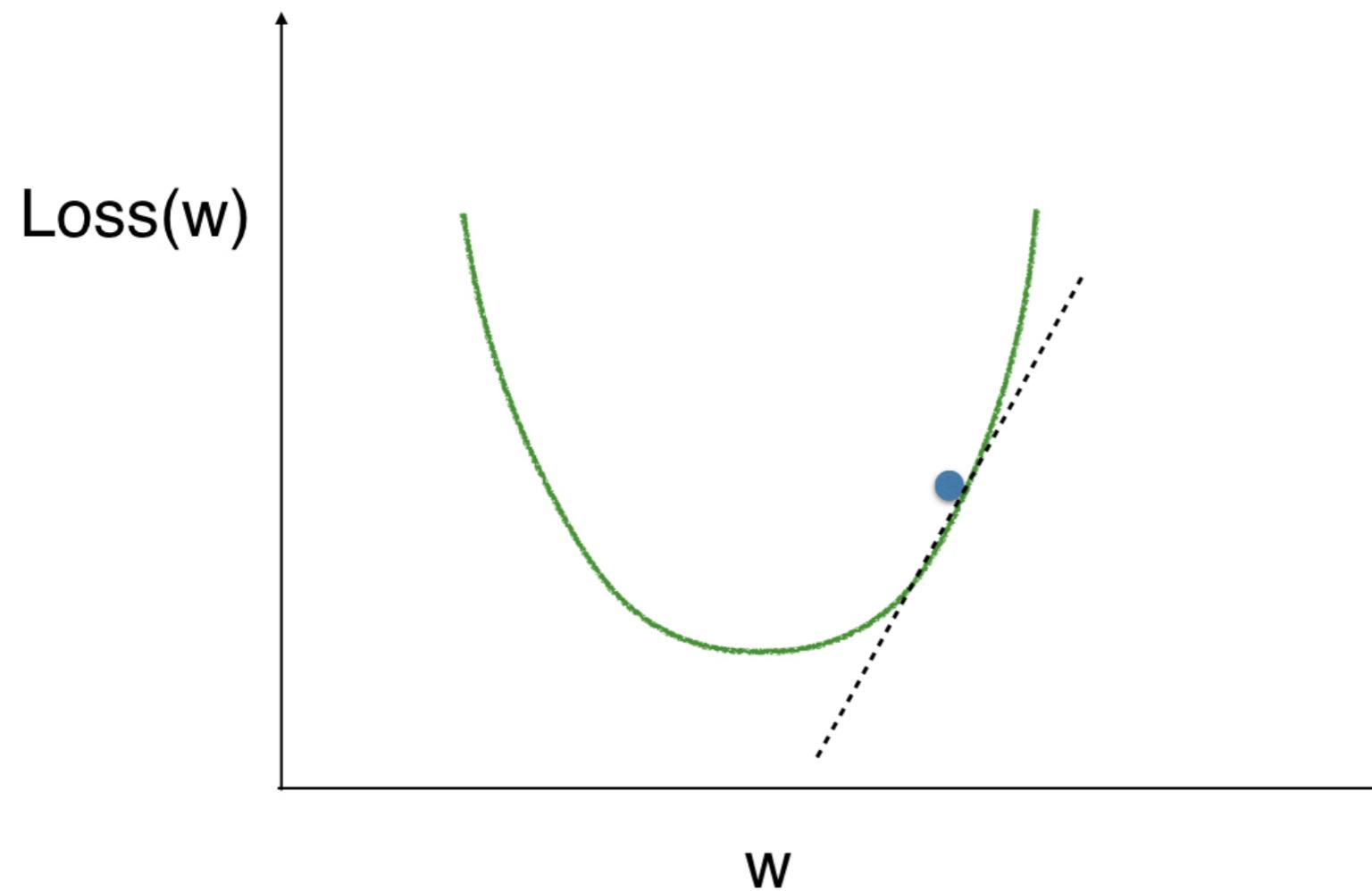
# Gradient descent



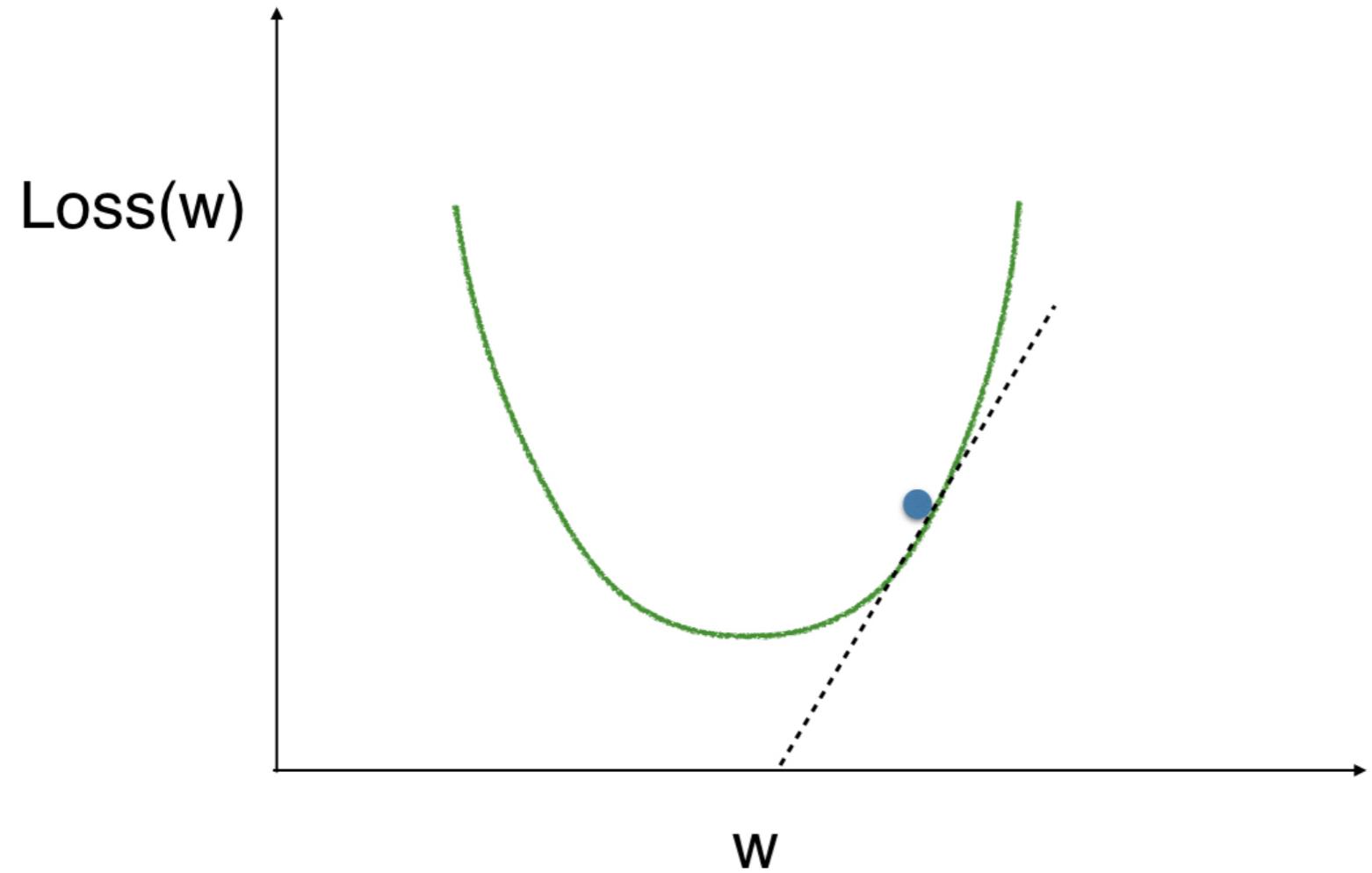
# Gradient descent



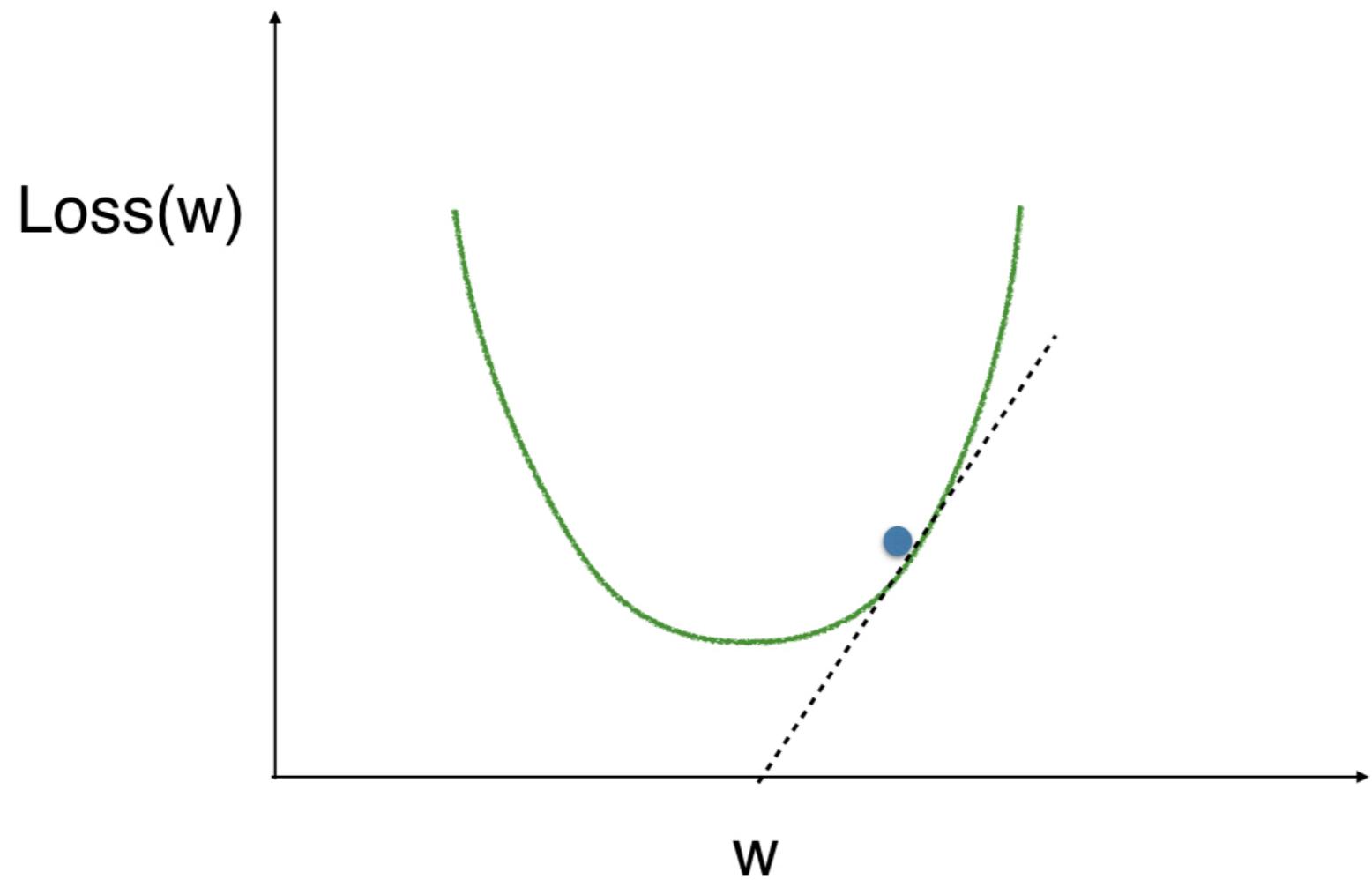
# Gradient descent



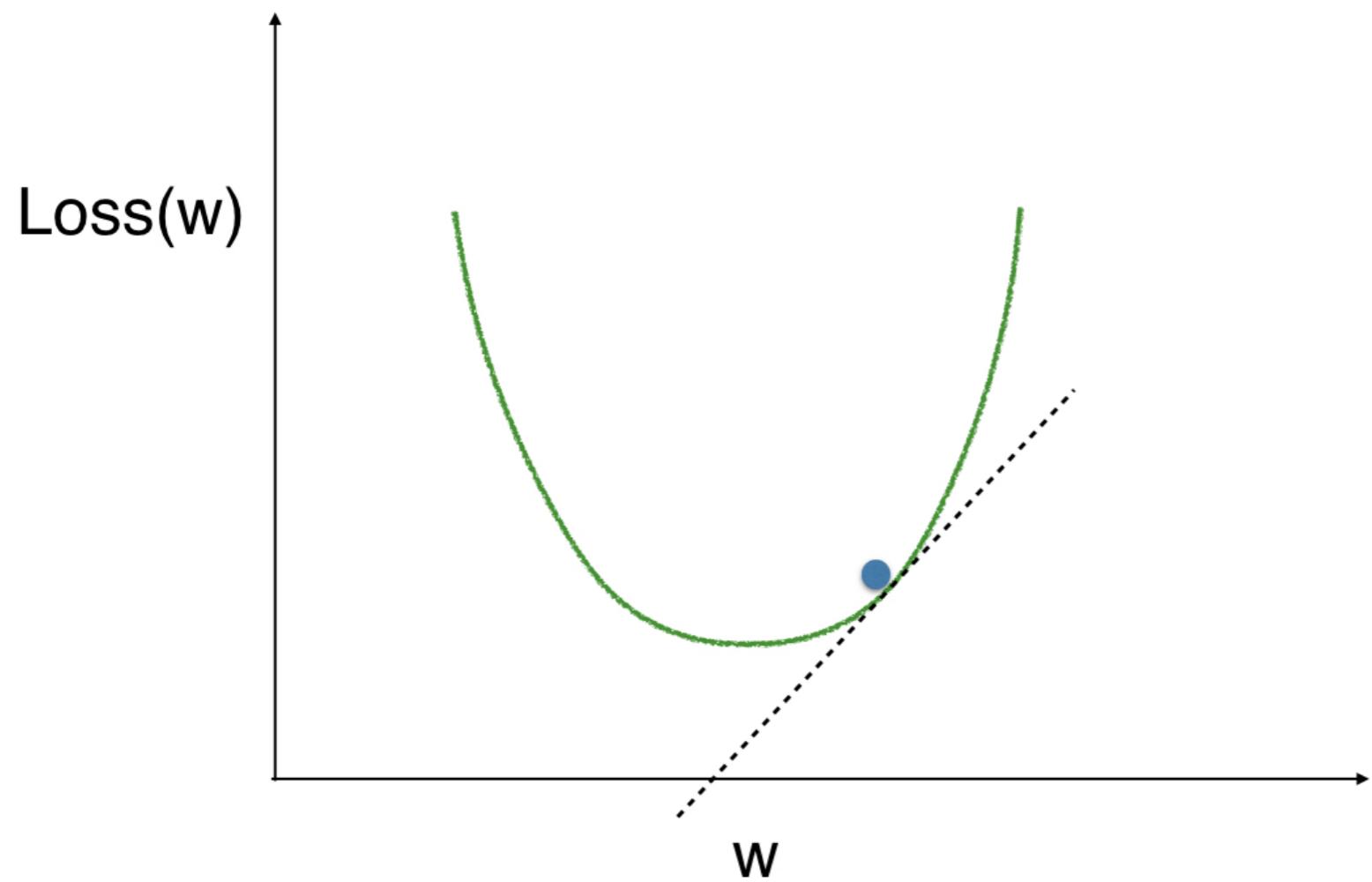
# Gradient descent



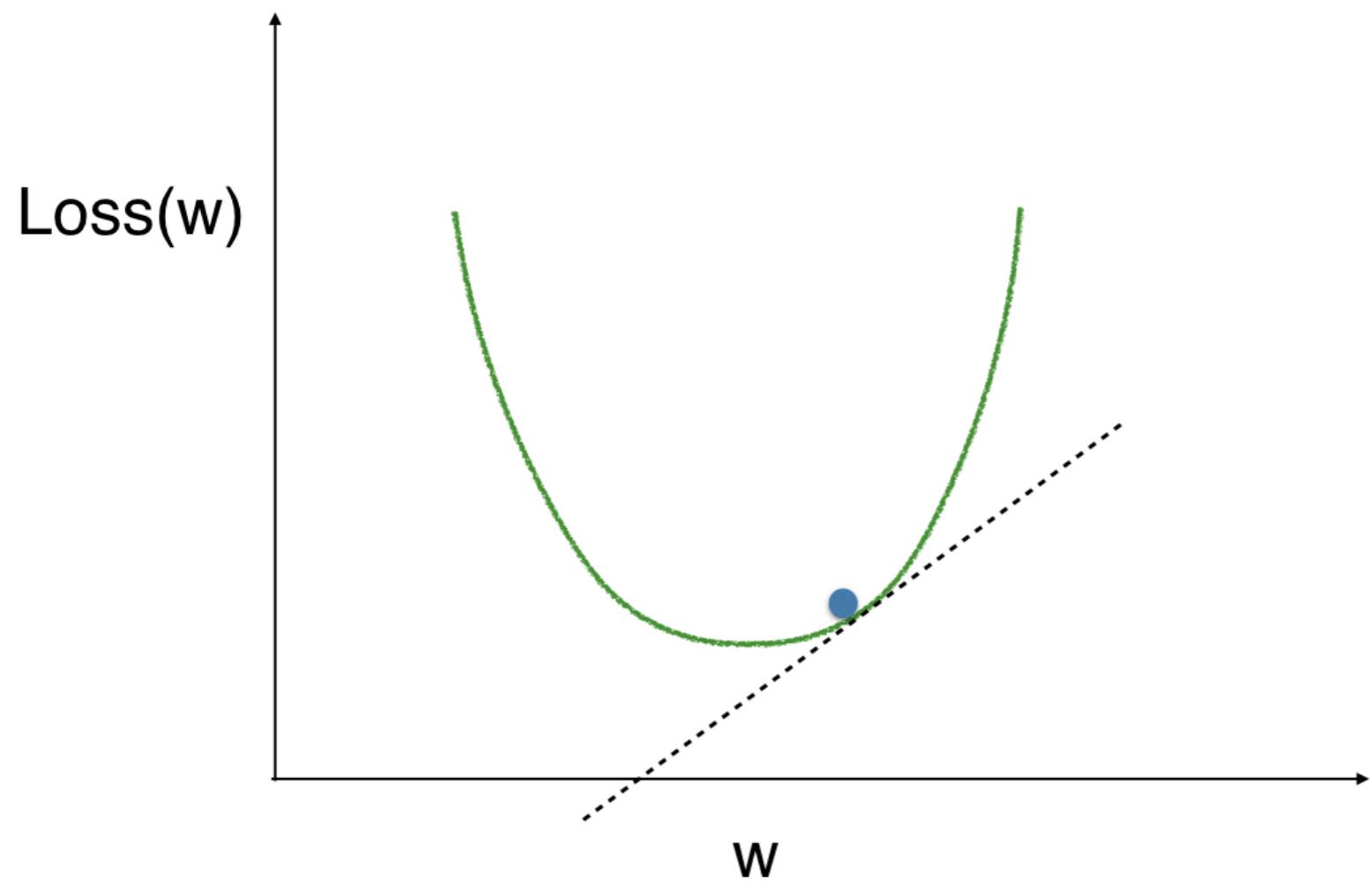
# Gradient descent



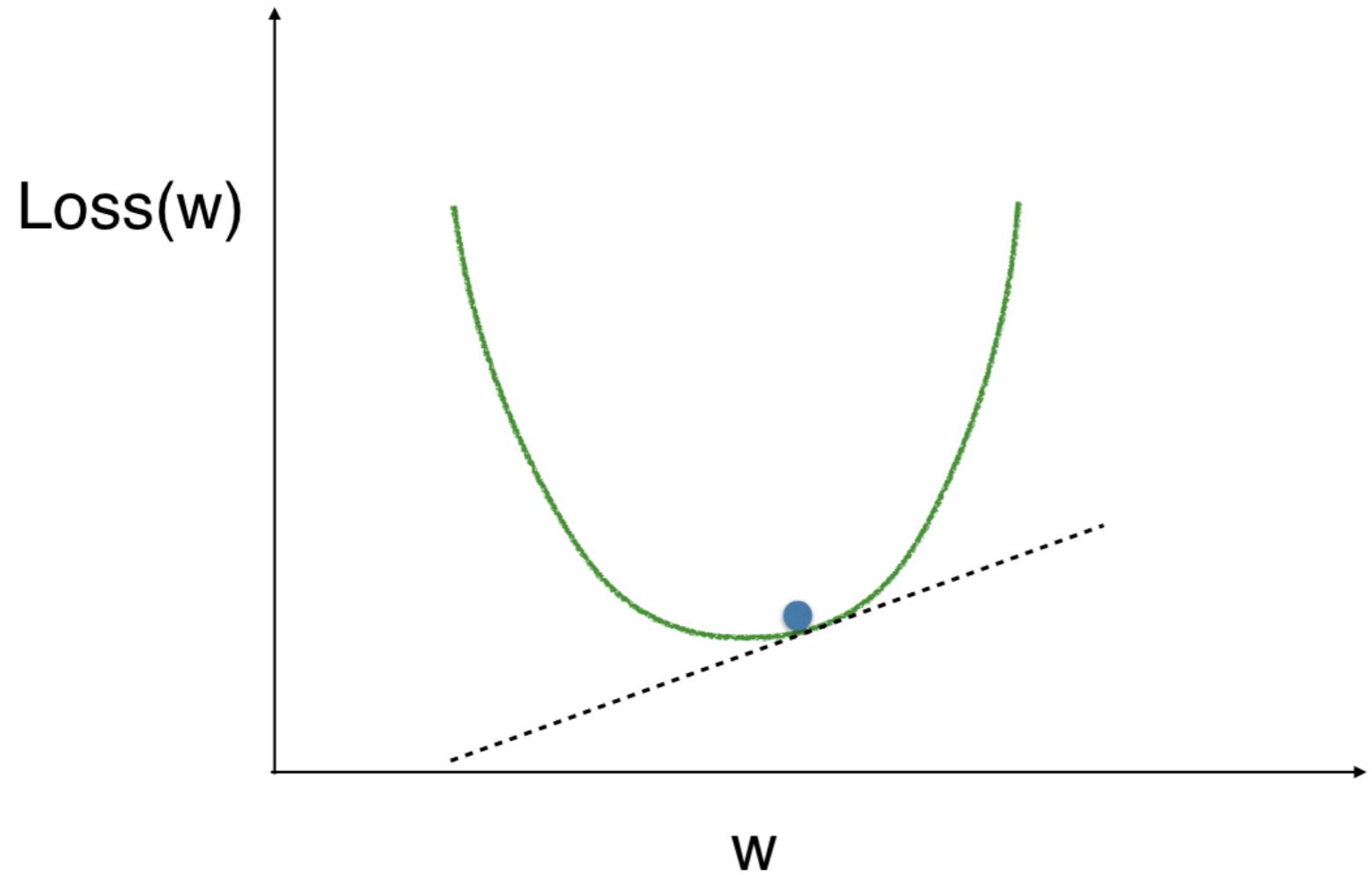
# Gradient descent



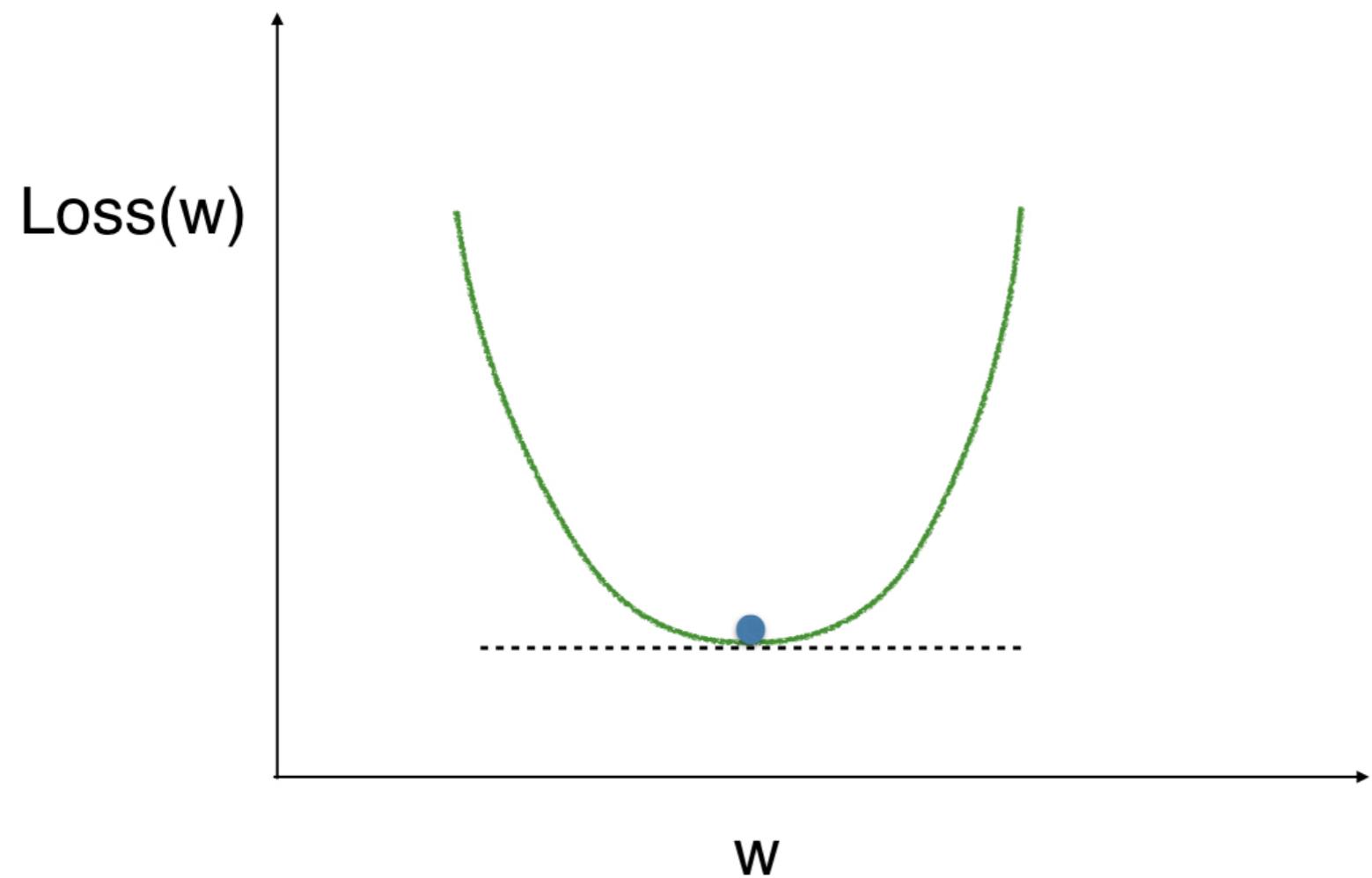
# Gradient descent



# Gradient descent



# Gradient descent



# Gradient descent

- If the slope is positive:
  - Going opposite the slope means moving to lower numbers
  - Subtract the slope from the current value
  - Too big a step might lead us astray
- Solution: learning rate
  - Update each weight by subtracting learning rate \* slope

# Slope calculation example



- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

# Slope calculation example



- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

# Slope calculation example



- Slope of mean-squared loss function w.r.t prediction:
  - $2(Predicted\ Value - Actual\ Value) = 2\ Error$
  - $2 * -4$

# Slope calculation example



- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - **The value of the node that feeds into our weight**
  - Slope of the activation function w.r.t value we feed into

# Slope calculation example



- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - **Slope of the activation function w.r.t value we feed into**

# Slope calculation example



- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

# Slope calculation example



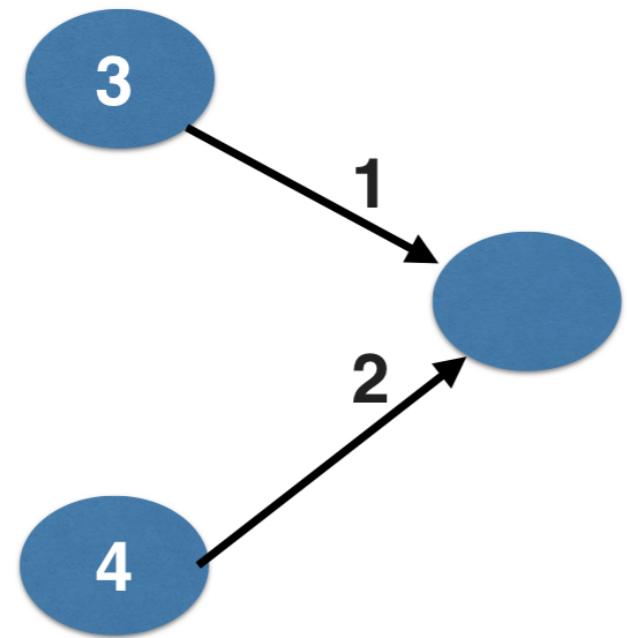
- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - ~~Slope of the activation function w.r.t value we feed into~~

# Slope calculation example



- $2 * -4 * 3$
- $-24$
- If learning rate is  $0.01$ , the new weight would be
- $2 - 0.01(-24) = 2.24$

# Network with two inputs affecting prediction



# Code to calculate slopes and update weights

```
import numpy as np  
weights = np.array([1, 2])  
input_data = np.array([3, 4])  
target = 6  
learning_rate = 0.01  
preds = (weights * input_data).sum()  
error = preds - target  
print(error)
```

5

# Code to calculate slopes and update weights

```
gradient = 2 * input_data * error  
gradient
```

```
array([30, 40])
```

```
weights_updated = weights - learning_rate * gradient  
preds_updated = (weights_updated * input_data).sum()  
error_updated = preds_updated - target  
print(error_updated)
```

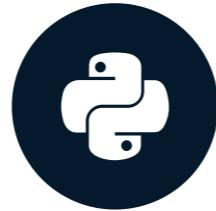
```
2.5
```

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING IN PYTHON**

# Backpropagation

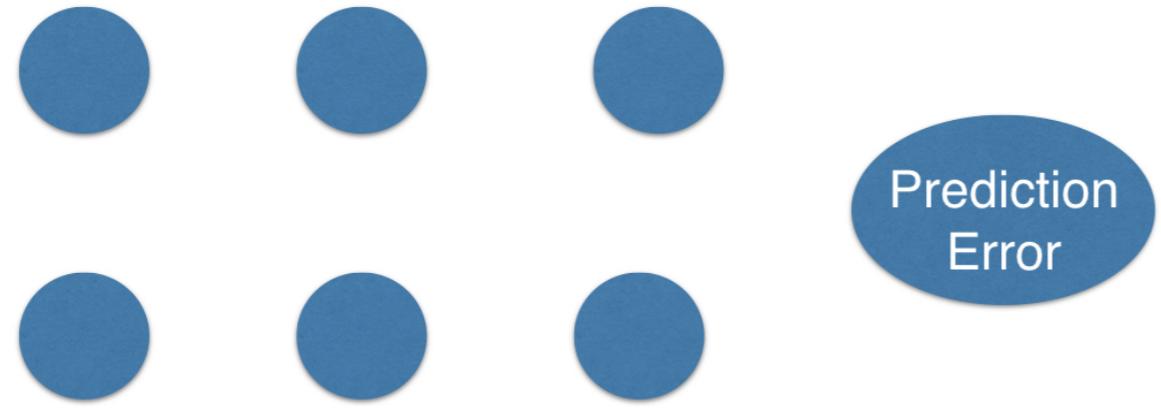
INTRODUCTION TO DEEP LEARNING IN PYTHON



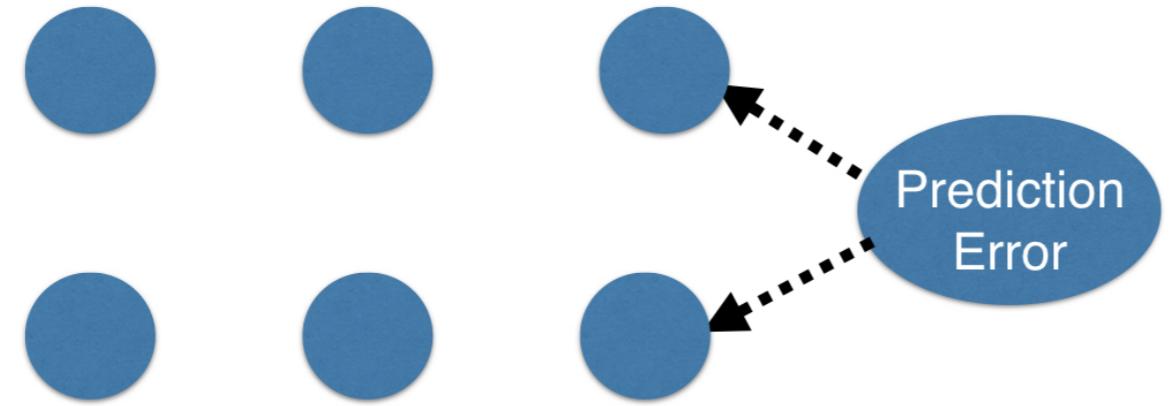
Dan Becker

Data Scientist and contributor to Keras  
and TensorFlow libraries

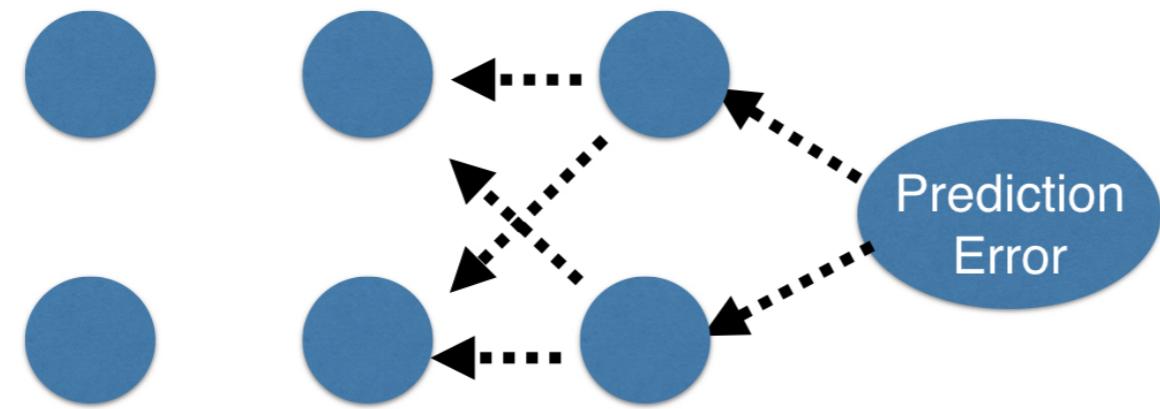
# Backpropagation



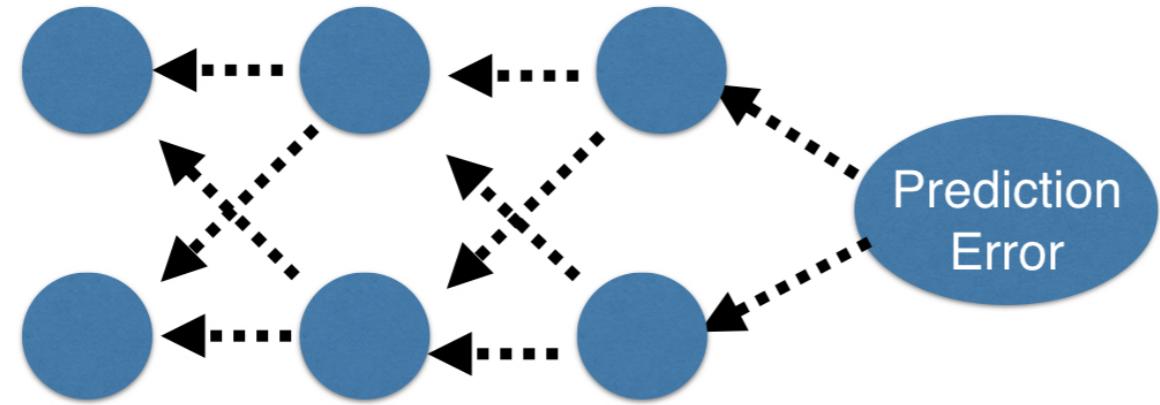
# Backpropagation



# Backpropagation



# Backpropagation

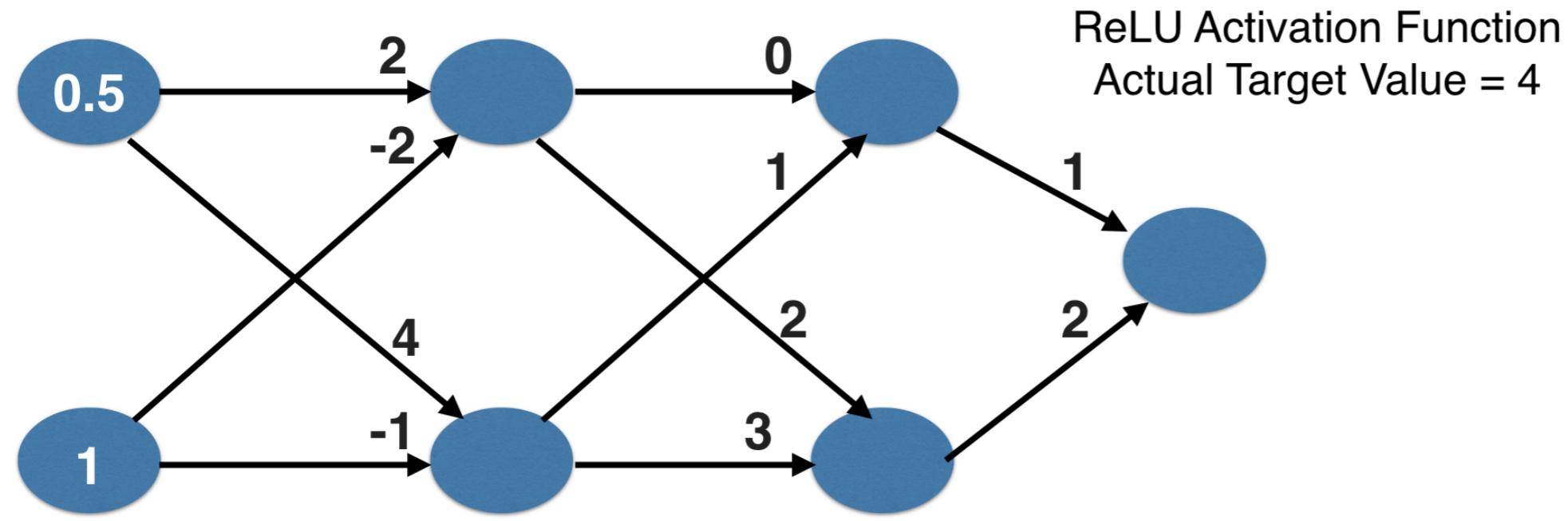


- Allows gradient descent to update all weights in neural network (by getting gradients for all weights)
- Comes from chain rule of calculus
- Important to understand the process, but you will generally use a library that implements this

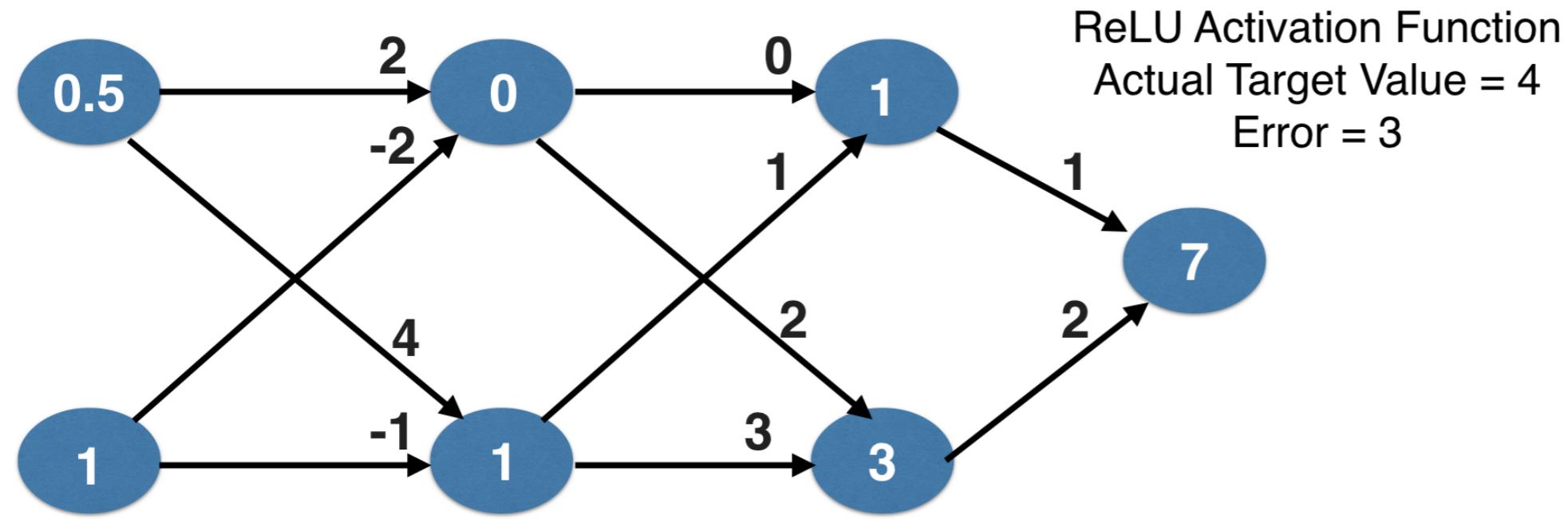
# Backpropagation process

- Trying to estimate the slope of the loss function w.r.t each weight
- Do forward propagation to calculate predictions and errors

# Backpropagation process



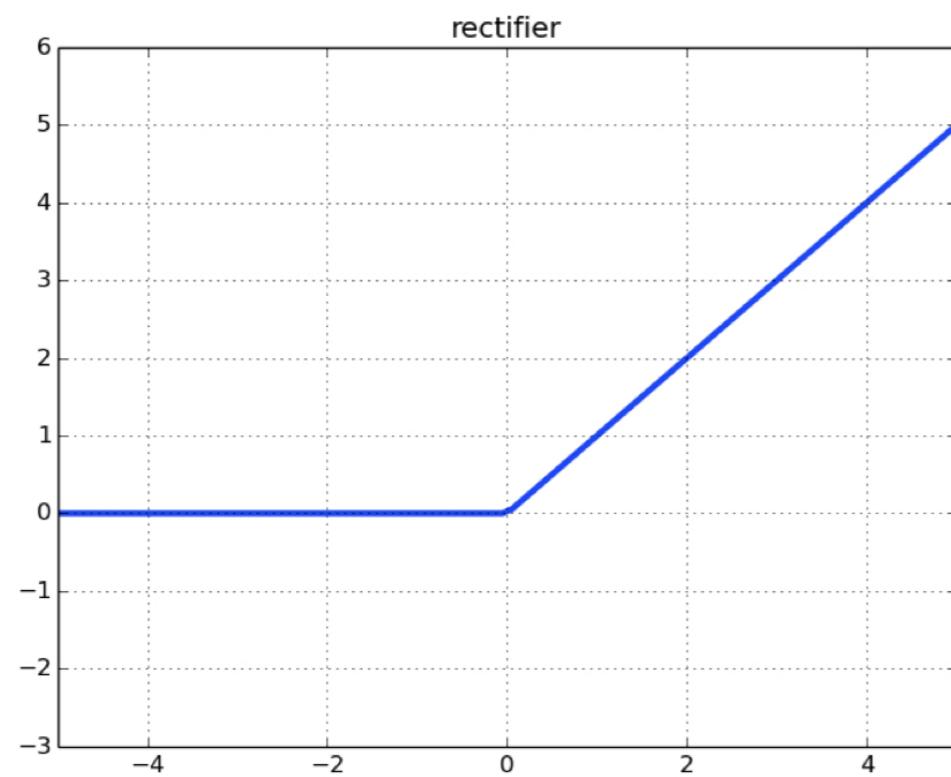
# Backpropagation process



# Backpropagation process

- Go back one layer at a time
- Gradients for weight is product of:
  1. Node value feeding into that weight
  2. Slope of loss function w.r.t node it feeds into
  3. Slope of activation function at the node it feeds into

# ReLU Activation Function



# Backpropagation process

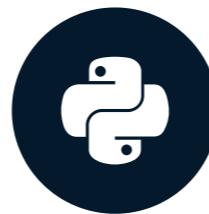
- Need to also keep track of the slopes of the loss function w.r.t node values
- Slope of node values are the sum of the slopes for all weights that come out of them

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING IN PYTHON**

# Backpropagation in practice

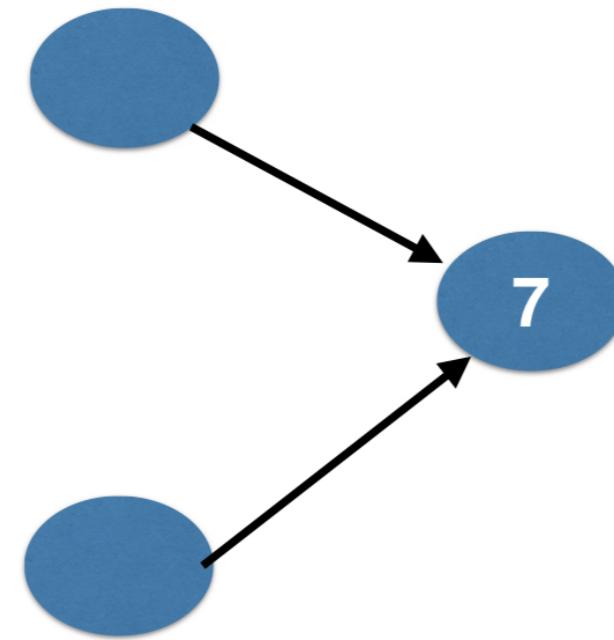
INTRODUCTION TO DEEP LEARNING IN PYTHON



Dan Becker

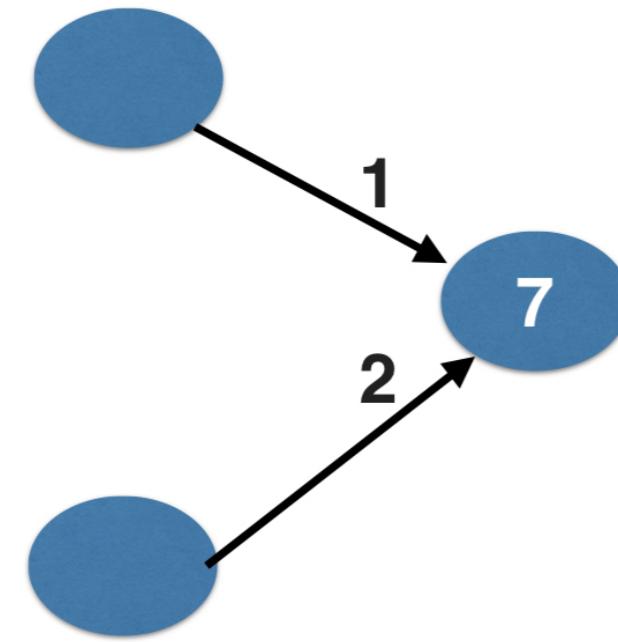
Data Scientist and contributor to Keras  
and TensorFlow libraries

# Backpropagation



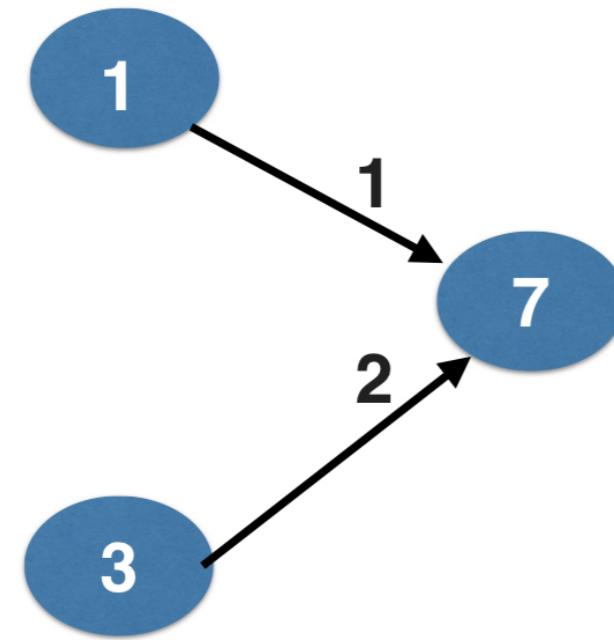
ReLU Activation Function  
Actual Target Value = 4  
Error = 3

# Backpropagation



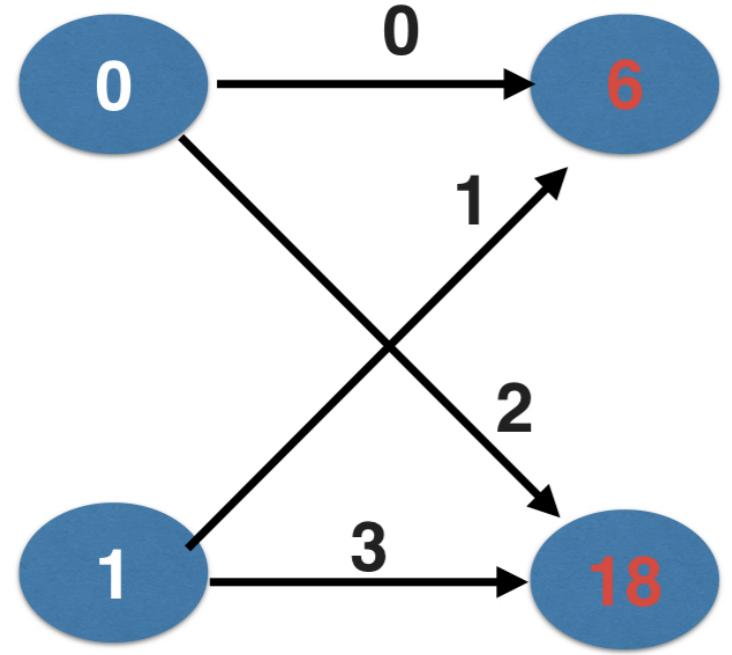
ReLU Activation Function  
Actual Target Value = 4  
Error = 3

# Backpropagation



ReLU Activation Function  
Actual Target Value = 4  
Error = 3

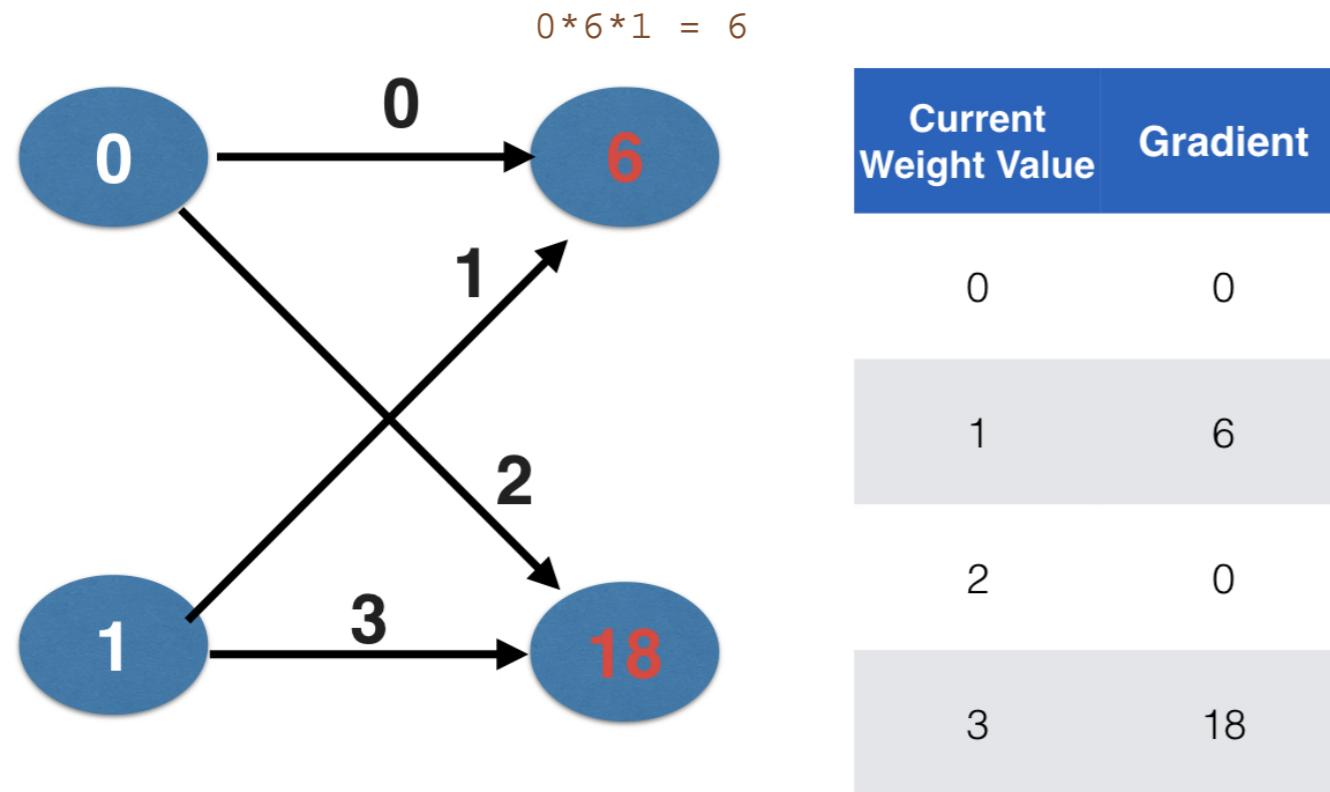
# Backpropagation



# Calculating slopes associated with any weight

- Gradients for weight is product of:
  1. Node value feeding into that weight
  2. Slope of activation function for the node being fed into
  3. Slope of loss function w.r.t output node

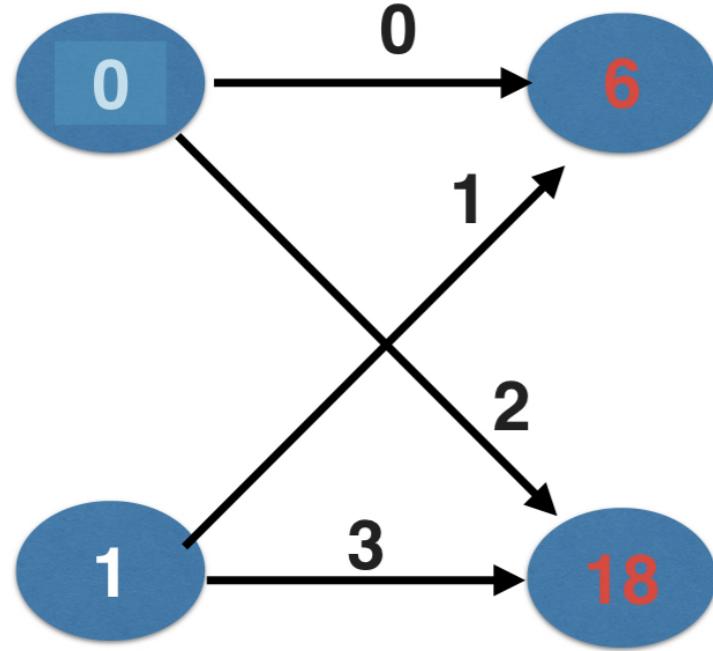
# Backpropagation



`input_node_value * output_node_slope * derivative of activation function`

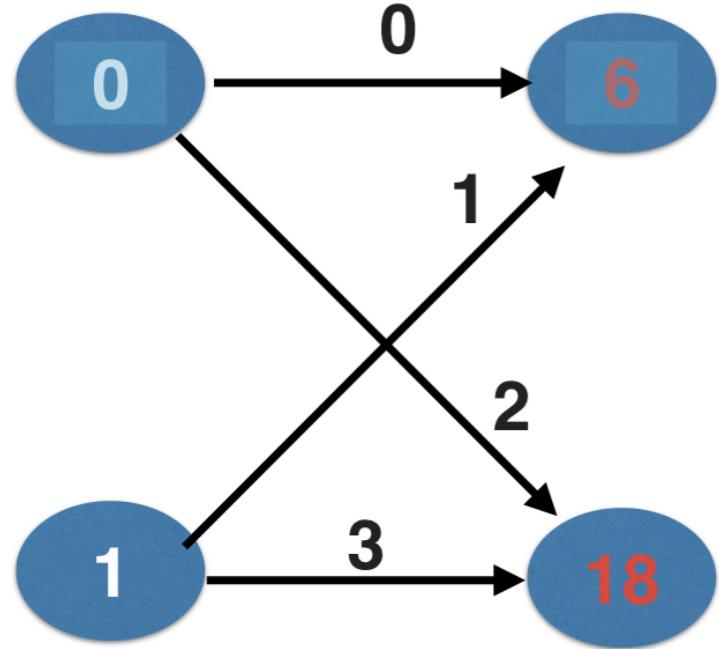
That output node has a positive value for the input, so the ReLU activation has slope of 1

# Backpropagation



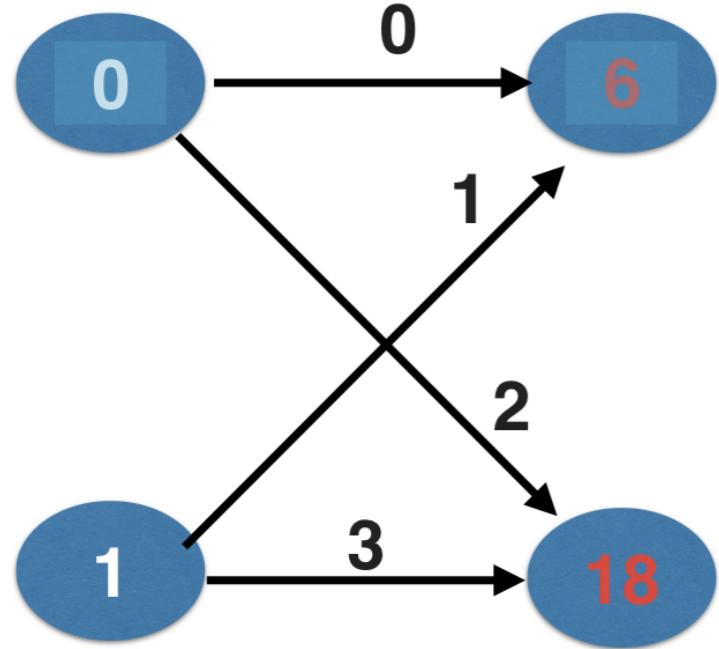
Current Weight Value	Gradient
0	0
1	6
2	0
3	18

# Backpropagation



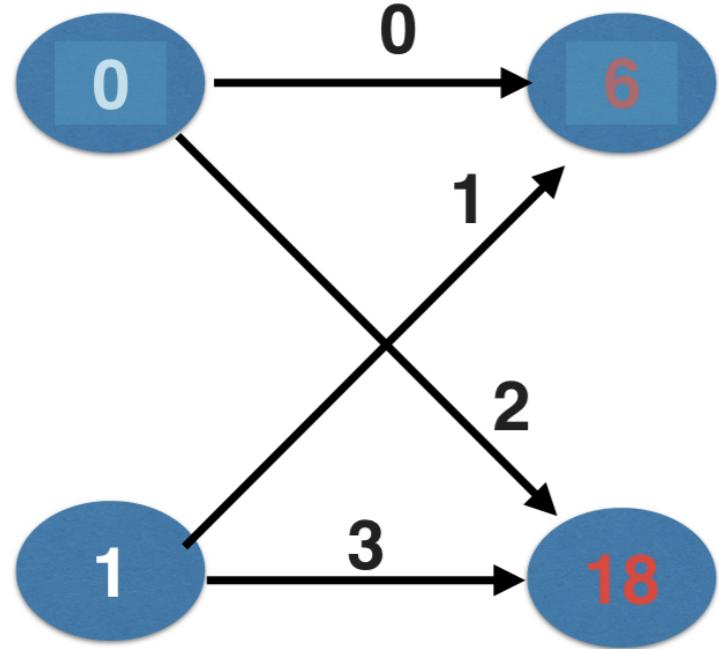
Current Weight Value	Gradient
0	0
1	6
2	0
3	18

# Backpropagation



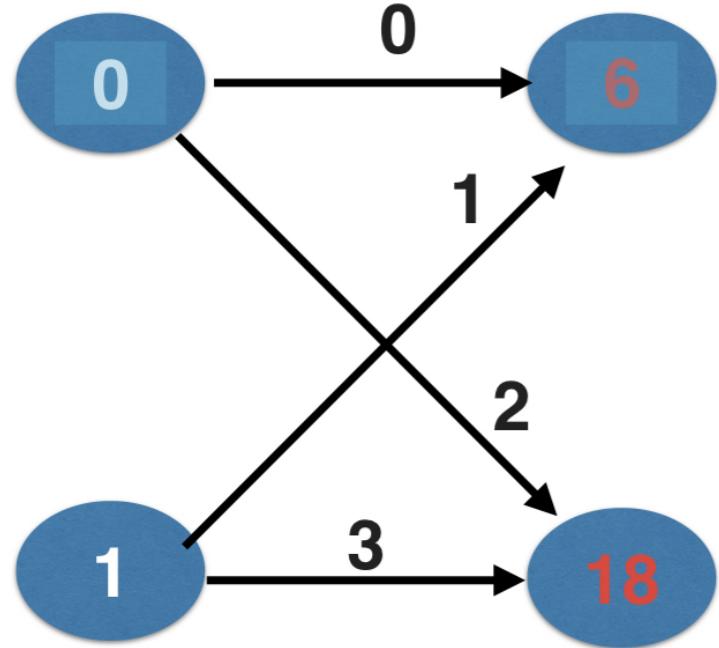
Current Weight Value	Gradient
0	0
1	6
2	0
3	18

# Backpropagation



Current Weight Value	Gradient
0	0
1	6
2	0
3	18

# Backpropagation



Current Weight Value	Gradient
0	0
1	6
2	0
3	18

# Backpropagation: Recap

- Start at some random set of weights
- Use forward propagation to make a prediction
- Use backward propagation to calculate the slope of the loss function w.r.t each weight
- Multiply that slope by the learning rate, and subtract from the current weights
- Keep going with that cycle until we get to a flat part

# Stochastic gradient descent

- It is common to calculate slopes on only a subset of the data (*a batch*)
- Use a different batch of data to calculate the next update
- Start over from the beginning once all data is used
- Each time through the training data is called an epoch
- When slopes are calculated on one batch at a time:  
**stochastic gradient descent**

# **Let's practice!**

**INTRODUCTION TO DEEP LEARNING IN PYTHON**