

Pontificia Universidad  
**JAVERIANA**  
Bogotá



T I N Y



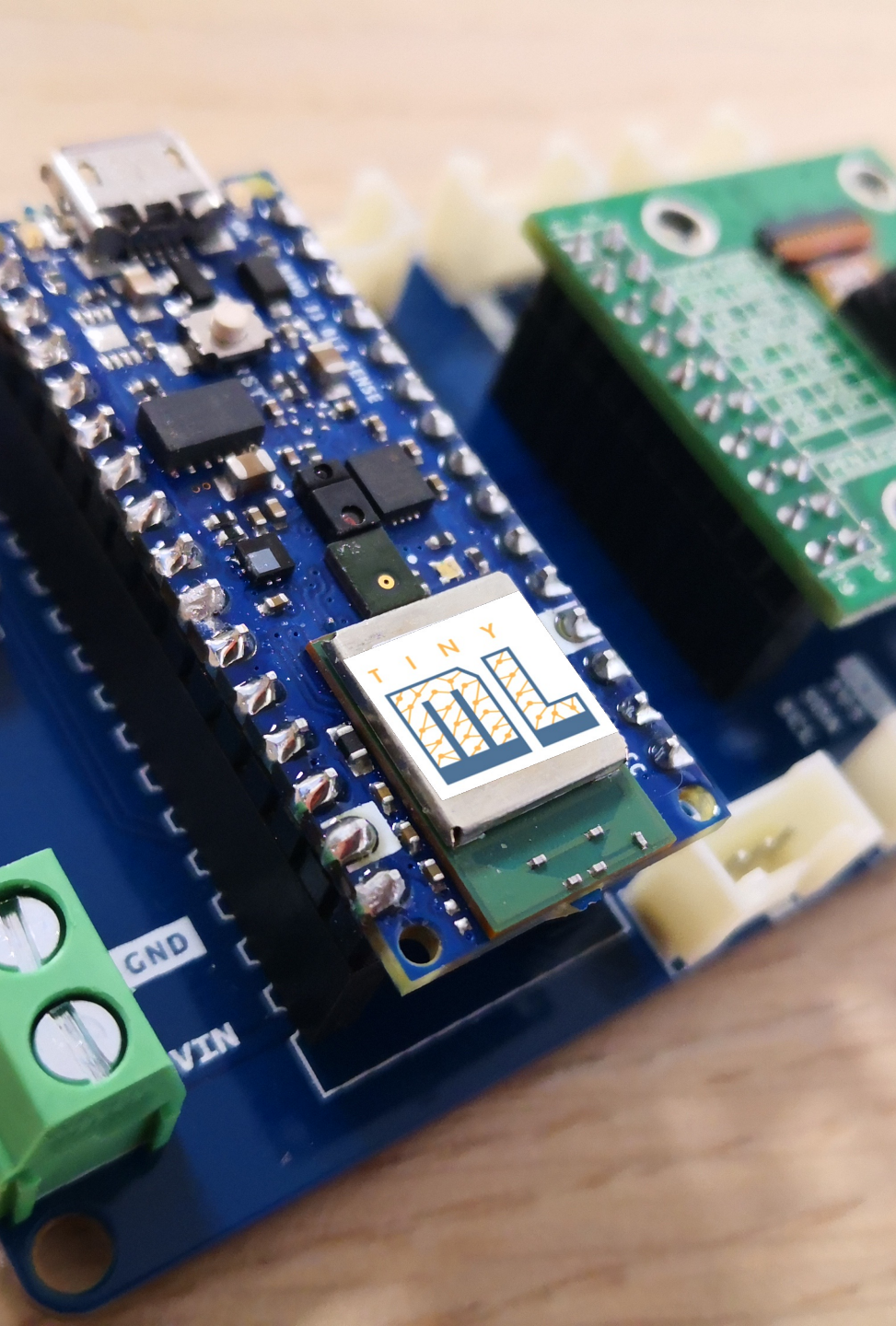
•edu

Facultad de Ingeniería

# A Brief Introduction to ML and DL

Workshop on Scientific Use of Machine Learning on  
Low-Power Devices: Applications and Advanced Topics

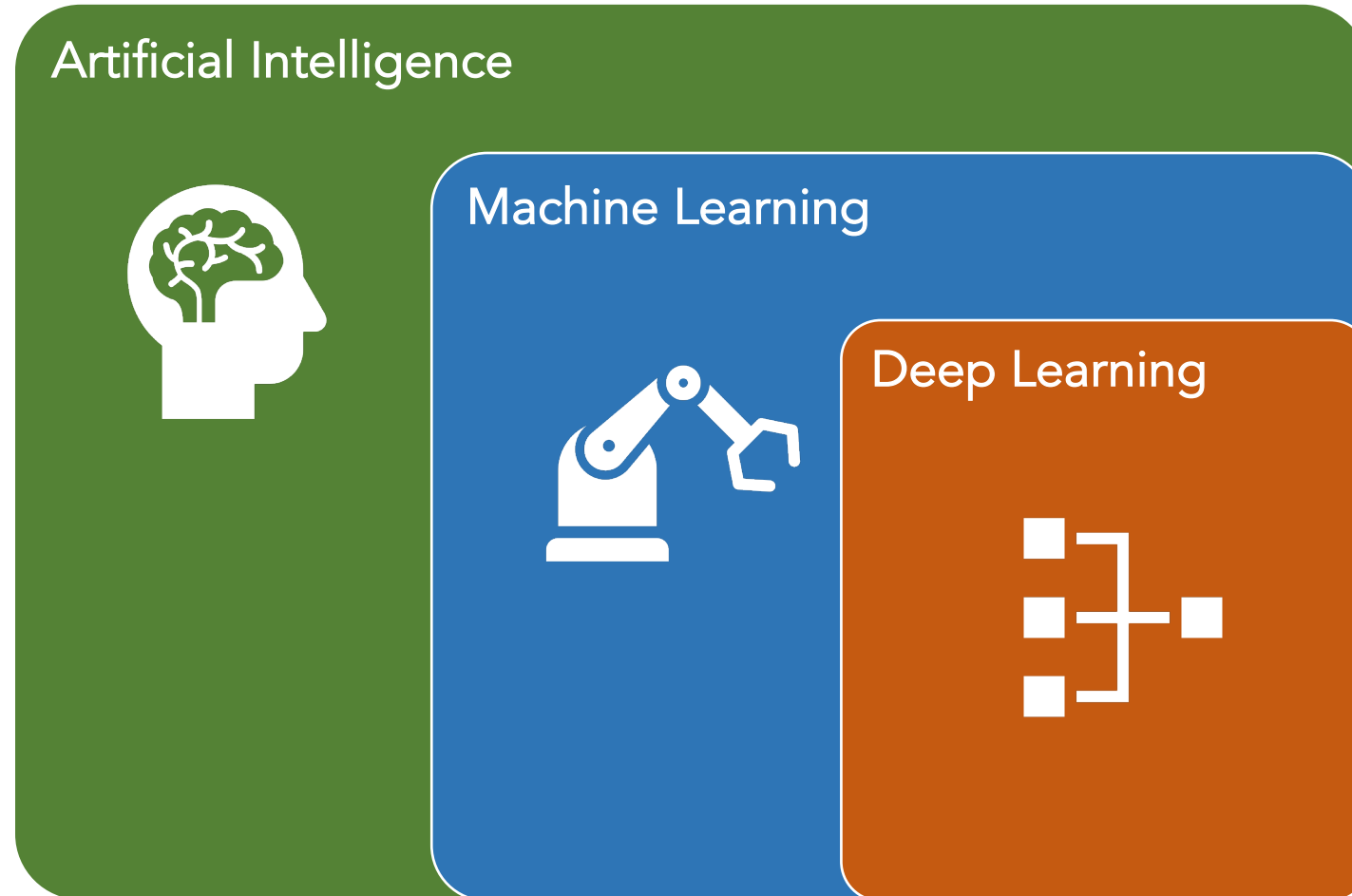
April 17<sup>th</sup>, 2023



# Outline

- AI vs ML vs DL
- The Machine Learning Paradigm
- Finding the Best Solution and Fitting a Model
- Regression and Classification with NN
- ML Issues

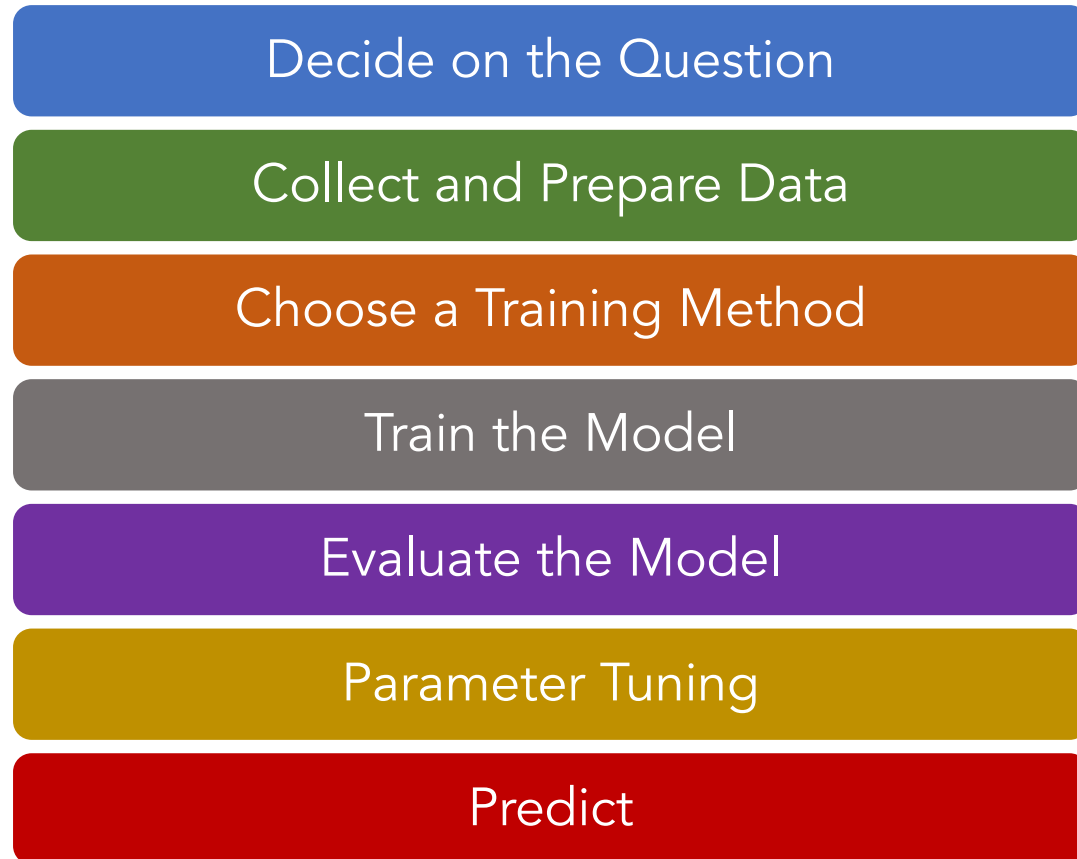
# AI vs. ML vs. DL

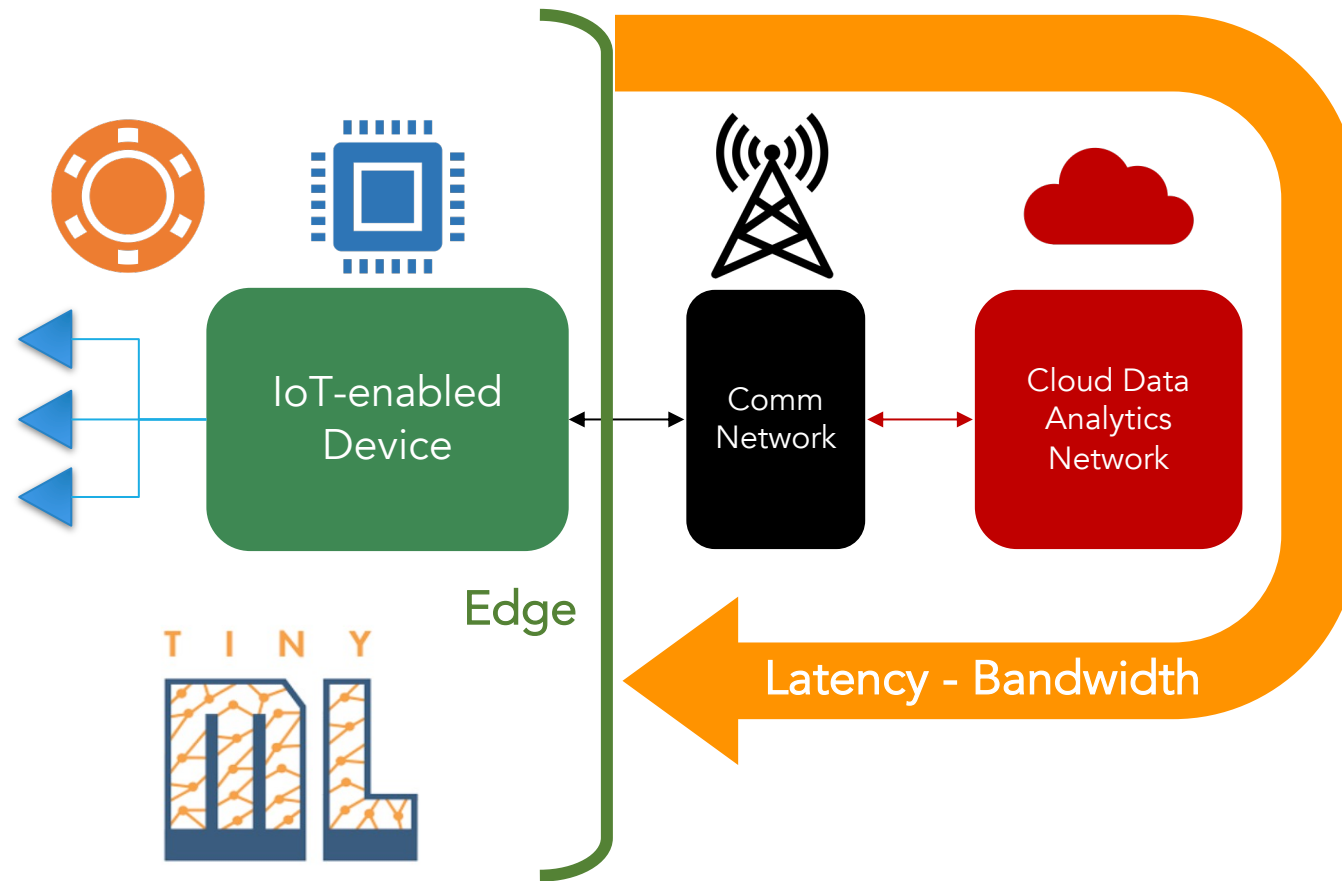


Based on <https://docs.microsoft.com/en-us/azure/machine-learning/concept-deep-learning-vs-machine-learning>

# General Steps for Machine Learning

On a high level, the craft of creating machine learning (ML) processes is comprised of several steps:





“The future of ML is *tiny* and bright.”

# We will run through this long process



This is a **first encounter with ML**, but many things will be left to be **experimented or developed**.



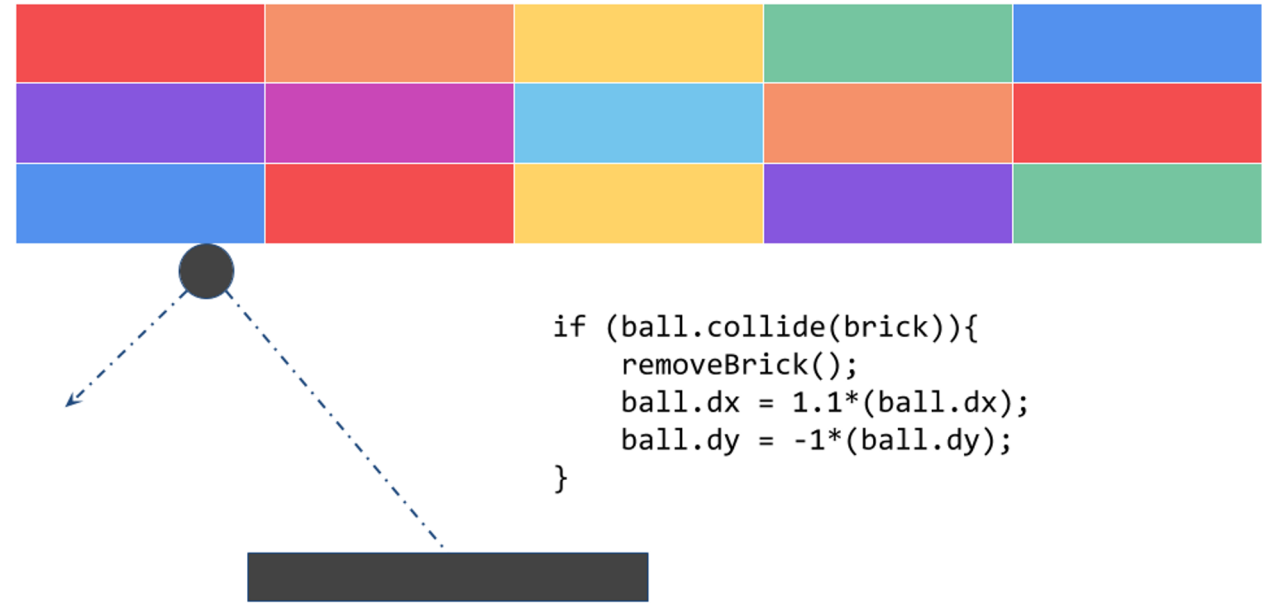
# The Machine Learning Paradigm

---



# Explicit Coding

- **Defining rules** that determine behavior of a program
- Everything is **pre-calculated and pre-determined** by the programmer
- **Scenarios are limited** by program complexity



# The Traditional Programming Paradigm



# Consider Activity Detection



```
if(speed<4){  
    status=WALKING;  
}
```



```
if(speed<4){  
    status=WALKING;  
} else {  
    status=RUNNING;  
}
```



```
if(speed<4){  
    status=WALKING;  
} else if(speed<12){  
    status=RUNNING;  
} else {  
    status=BIKING;  
}
```



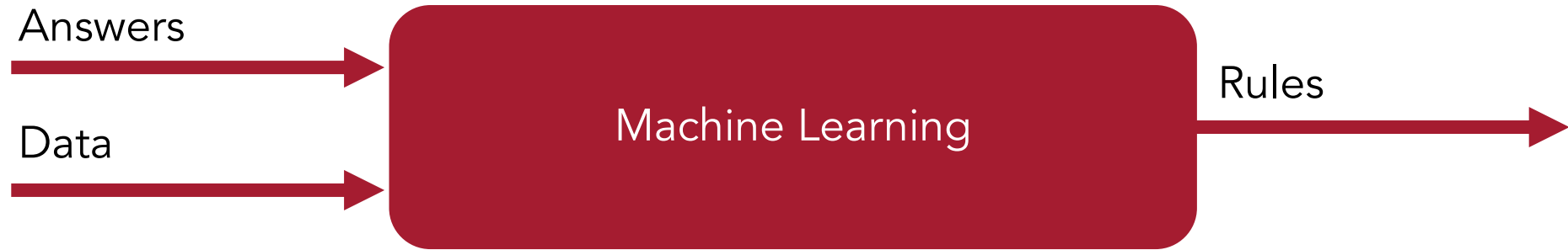
```
// ???
```

Way too  
complex  
to code!

# The Traditional Programming Paradigm



# The Machine Learning Paradigm



# Activity Detection with Machine Learning



```
0101001010100101010
1001010101001011101
0100101010010101001
0101001010100101010
```

Label = WALKING



```
1010100101001010101
0101010010010010001
0010011111010101111
1010100100111101011
```

Label = RUNNING



```
1001010011111010101
1101010111010101110
1010101111010101011
1111110001111010101
```

Label = BIKING



```
1111111111010011101
0011111010111110101
0101110101010101110
1010101010100111110
```

Label = GOLFING

# The Machine Learning Paradigm



```
0101001010100101010
1001010101001011101
0100101010010101001
0101001010100101010
```

Label = WALKING



```
1010100101001010101
0101010010010010001
0010011111010101111
1010100100111101011
```

Label = RUNNING



```
1001010011111010101
1101010111010101110
1010101111010101011
1111110001111010101
```

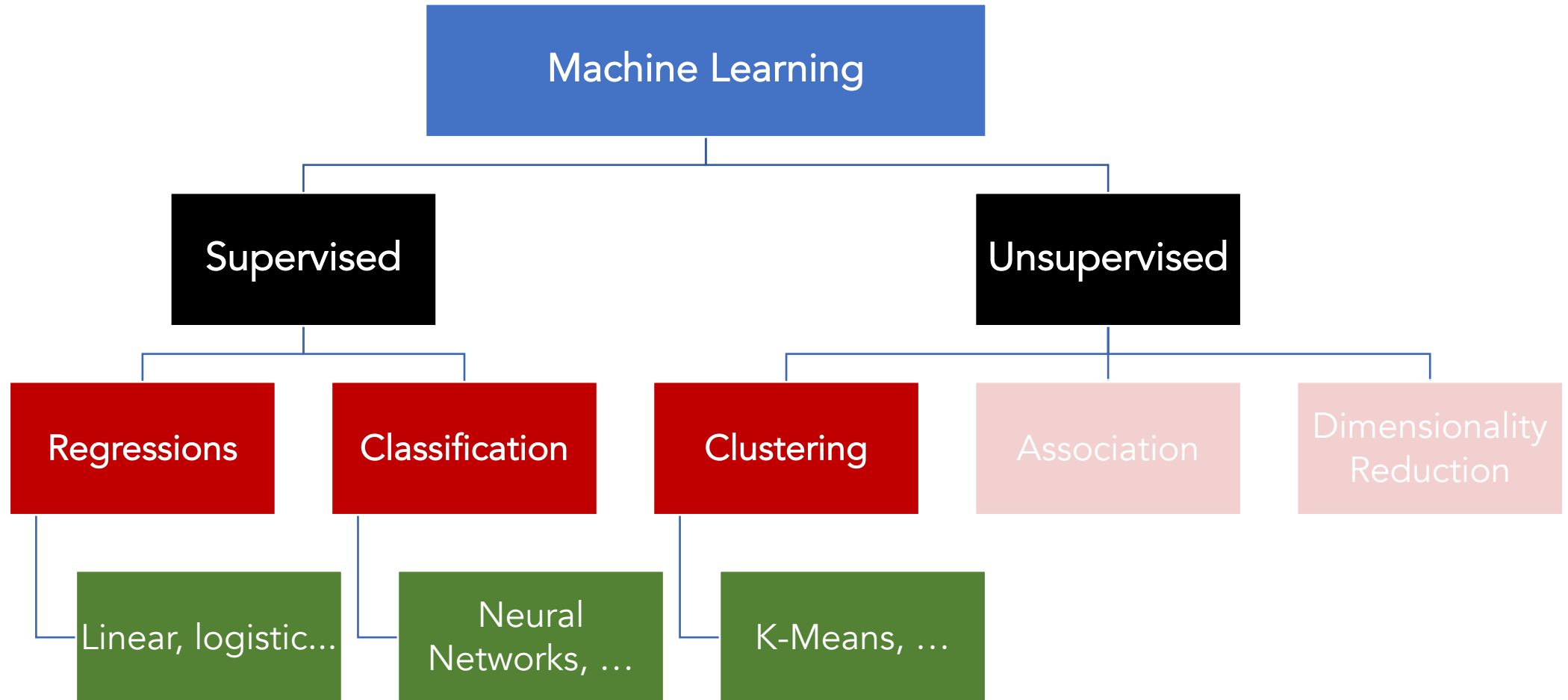
Label = BIKING



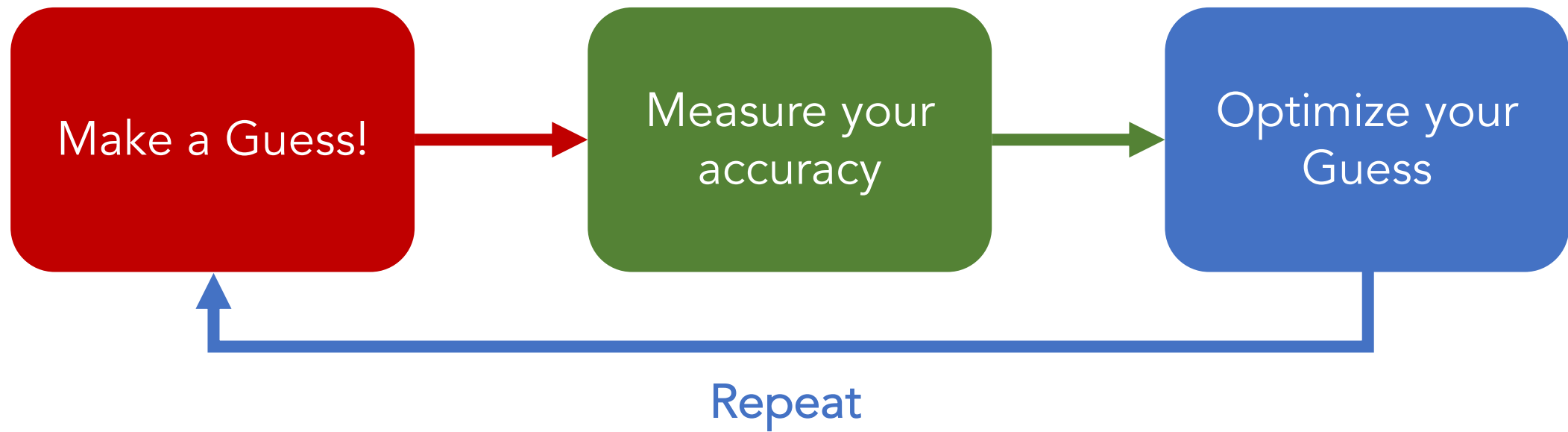
```
1111111111010011101
0011111010111110101
0101110101010101110
1010101010100111110
```

Label = GOLFING

# Two Approaches



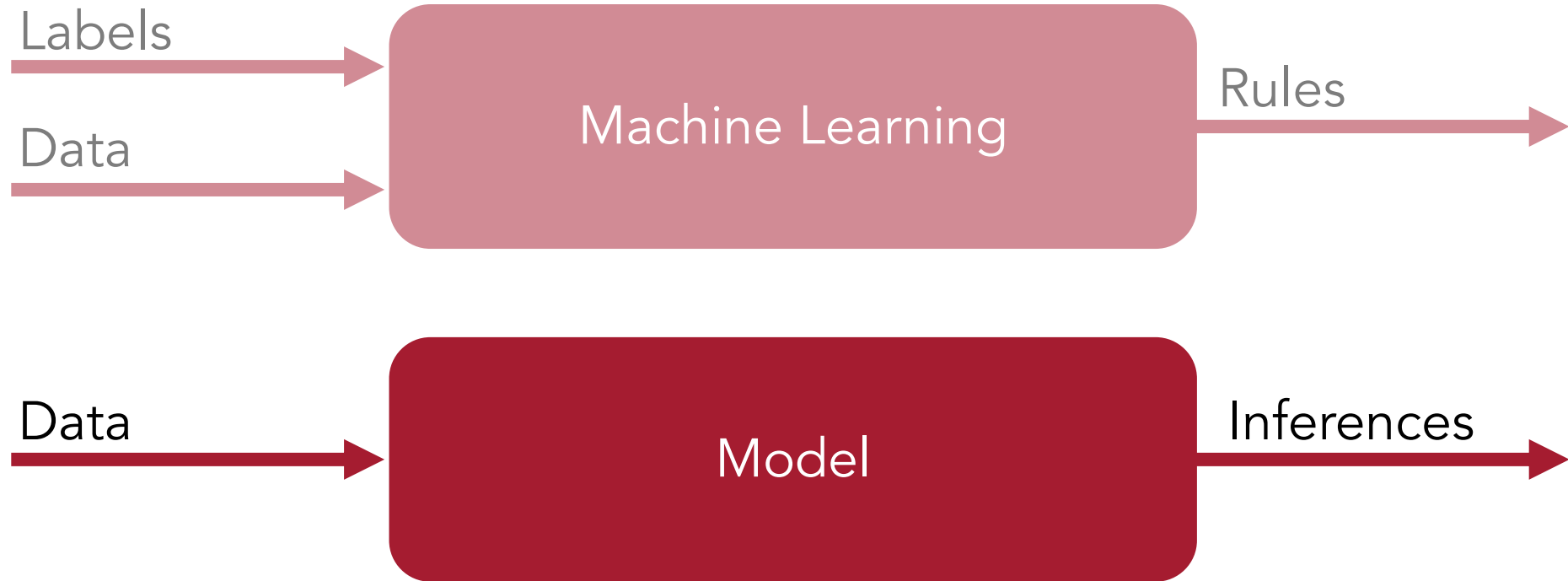
# The Machine Learning Paradigm



# The Machine Learning Paradigm



# The Machine Learning Paradigm



# How good is your model?

a way to measure your accuracy

# Matching X to Y

$$X = \{-1, 0, 1, 2, 3, 4\}$$

$$Y = \{-3, -1, 1, 3, 5, 7\}$$



Make a guess!

$$Y = 3X - 1$$

$$X = \{-1, 0, 1, 2, 3, 4\}$$

$$\text{My } Y = \{-4, -1, 2, 5, 8, 11\}$$

# How good is the guess?

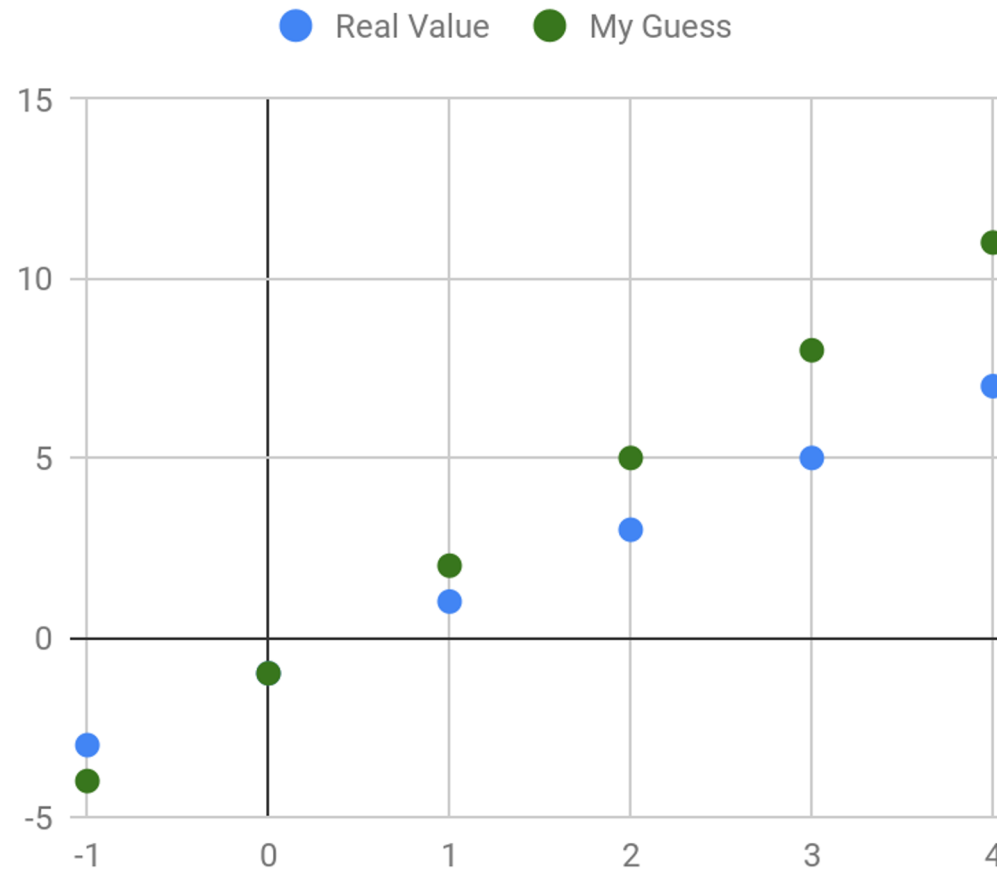
$$Y = 3X - 1$$

$$X = \{-1, 0, 1, 2, 3, 4\}$$

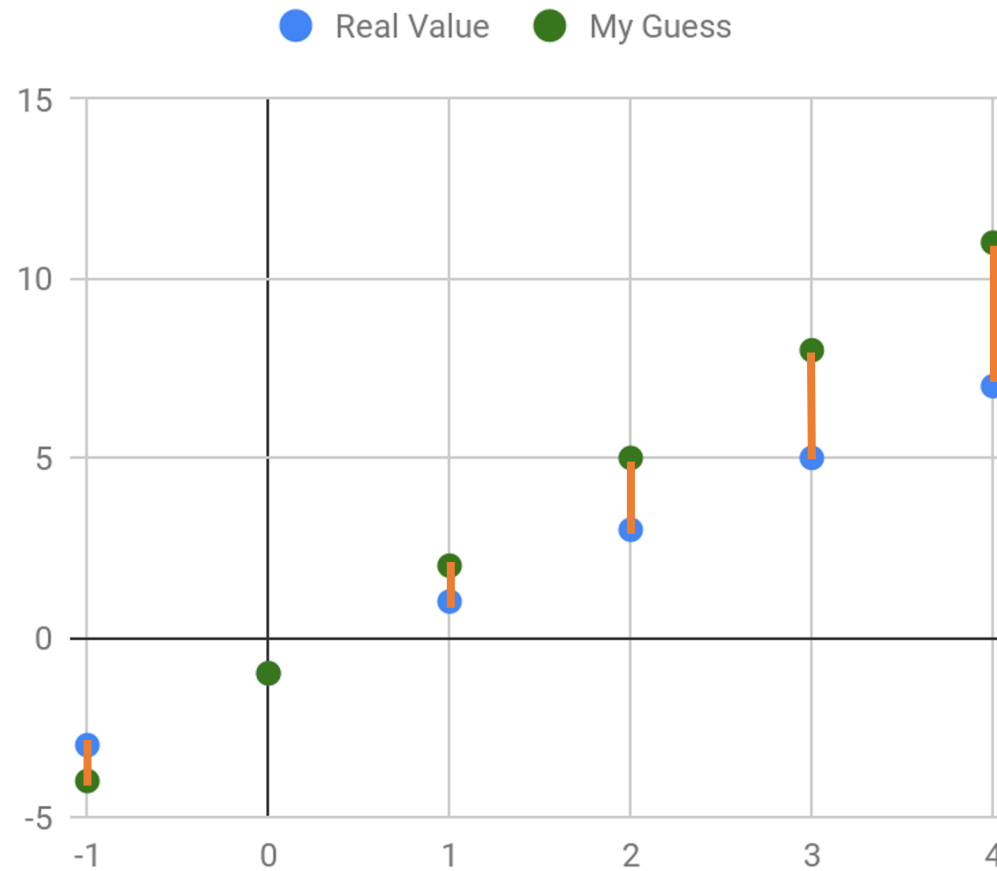
$$\text{My } Y = \{-4, -1, 2, 5, 8, 11\}$$

$$\text{Real } Y = \{-3, -1, 1, 3, 5, 7\}$$

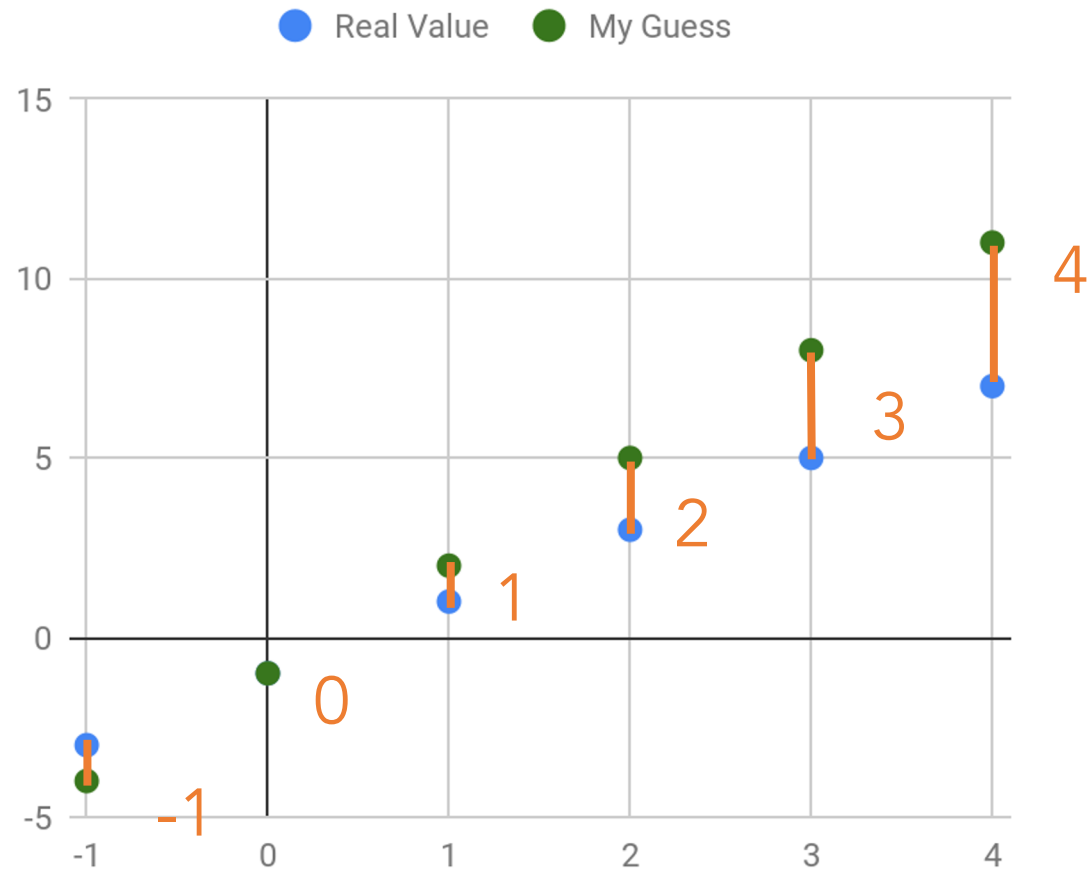
# Let's measure it!



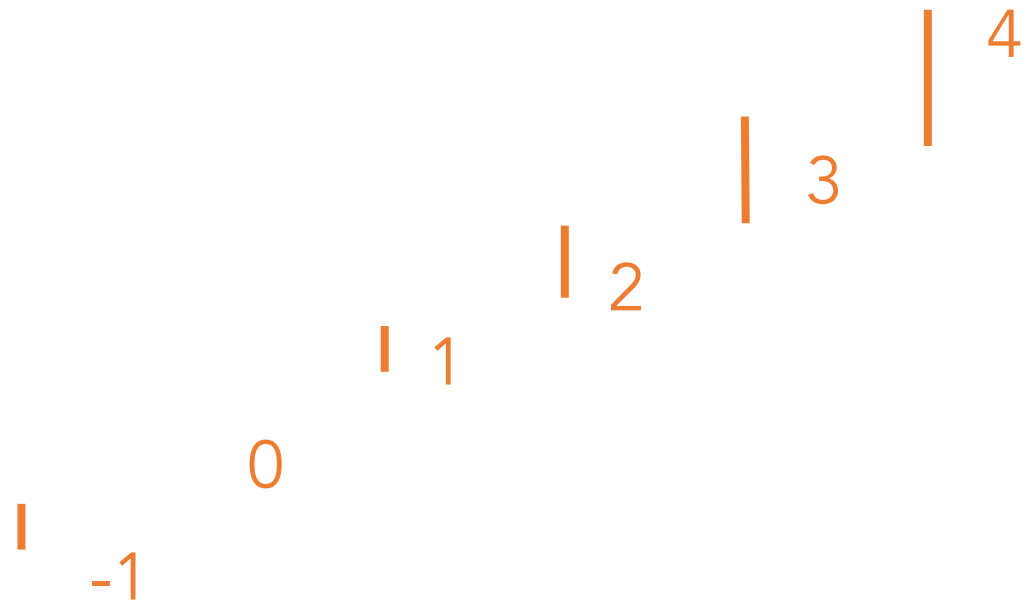
# Let's measure it!



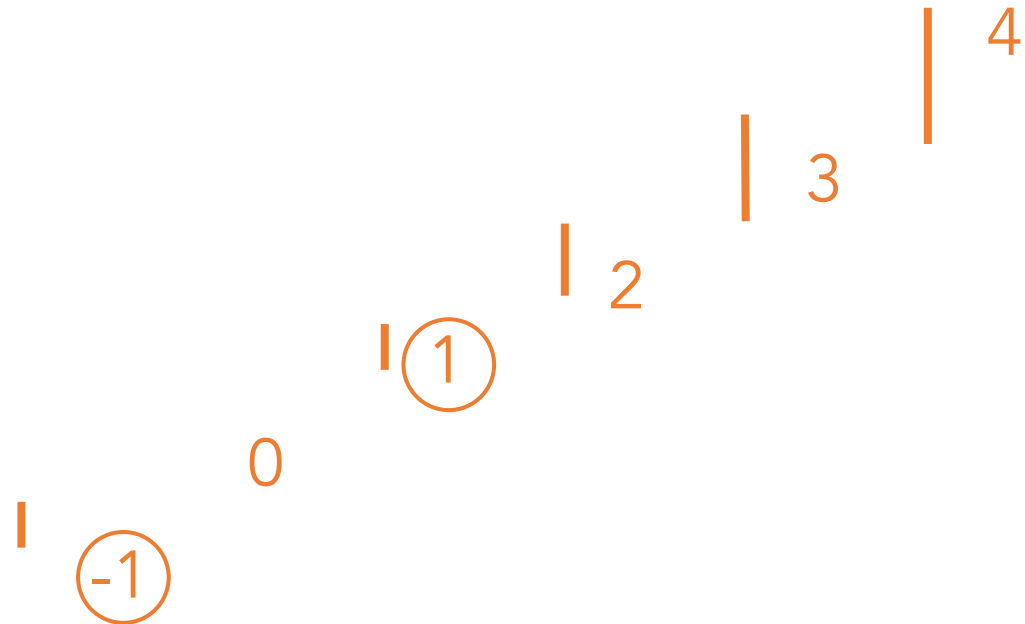
# Let's measure it!



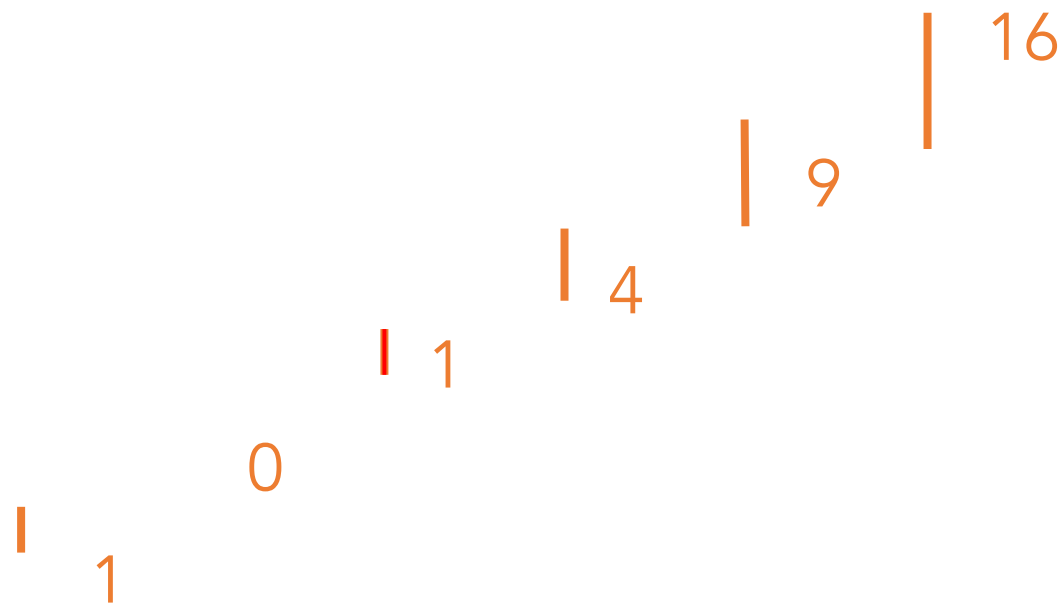
# Let's measure it!



Houston, we have a problem!



# What if we square<sup>2</sup> them?



Total that ( $\Sigma$ ) and take  
the square root  $\sqrt{\quad}$

$$\text{sqrt}(1 + 1 + 4 + 9 + 16)$$

$$= \text{sqrt}(31)$$

$$= 5.57$$



# Make another guess!

$$Y = 2X - 2$$

$$X = \{-1, 0, 1, 2, 3, 4\}$$

$$\text{My } Y = \{-4, -2, 0, 2, 4, 6\}$$

$$\text{Real } Y = \{-3, -1, 1, 3, 5, 7\}$$

$$\text{Diff}^2 = \{1, 1, 1, 1, 1\}$$



Get the same  
difference, repeat the  
same process.

$$\text{sqrt}(1 + 1 + 1 + 1 + 1)$$

$$= \text{sqrt}(5)$$

$$= 2.23$$



# Make another guess!

$$Y = 2X - 1$$

$$X = \{-1, 0, 1, 2, 3, 4\}$$

$$\text{My } Y = \{-3, -1, 1, 3, 5, 7\}$$

$$\text{Real } Y = \{-3, -1, 1, 3, 5, 7\}$$

$$\text{Diff}^2 = \{0, 0, 0, 0, 0\}$$



# Root-mean-square deviation

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^T (\hat{y}_t - y_t)^2}{T}}.$$



# Finding out the best solution

Trial and error approach

Loss  
Function

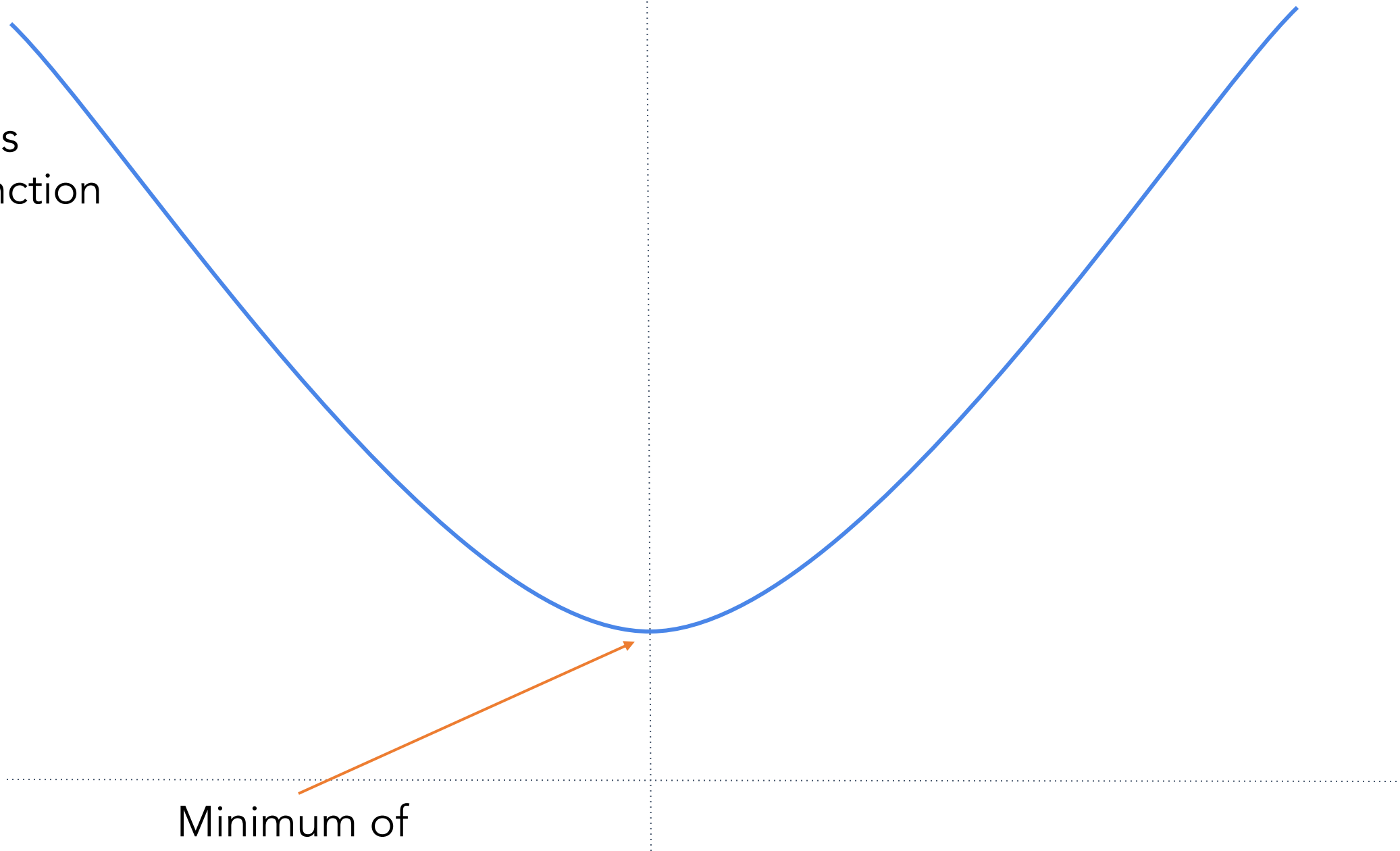
Parameter



The image shows a graph of a loss function. A blue U-shaped curve represents the loss, with its minimum point at the center. A vertical dotted line passes through this minimum. A horizontal dotted line is positioned below the curve. A red double-headed arrow is placed on the horizontal axis, centered under the minimum of the curve, indicating the range of parameters being considered.

Loss  
Function

Minimum of  
Loss Function

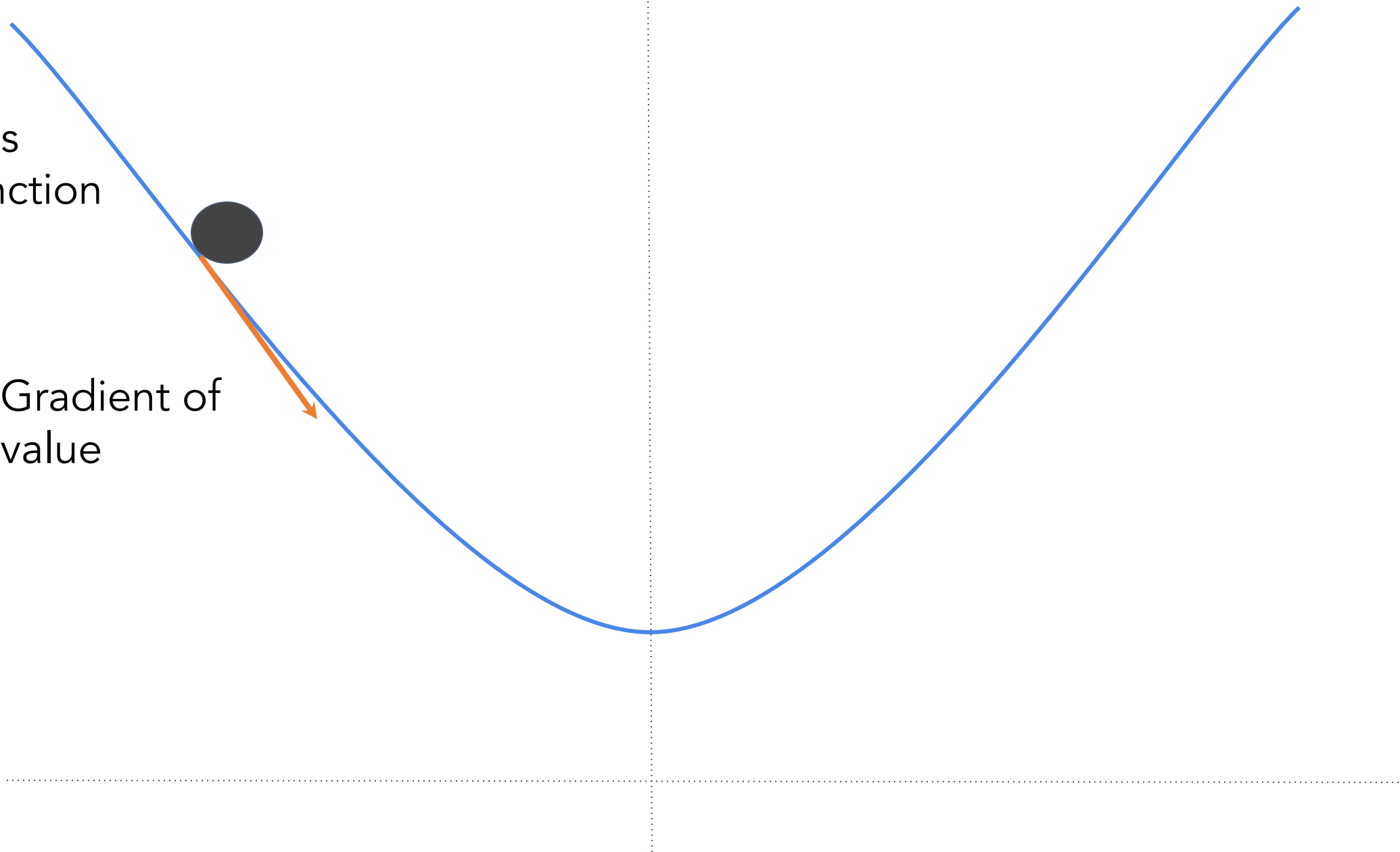


Loss  
Function



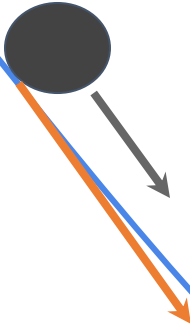
Loss  
Function

Gradient of  
value



Loss  
Function

Move in Direction of Gradient  
Learning Rate is size of the step to take



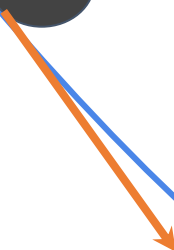
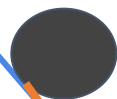
Loss  
Function

End up here



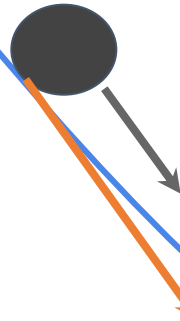
Loss  
Function

Get the  
gradient



Loss  
Function

Move in Direction of Gradient

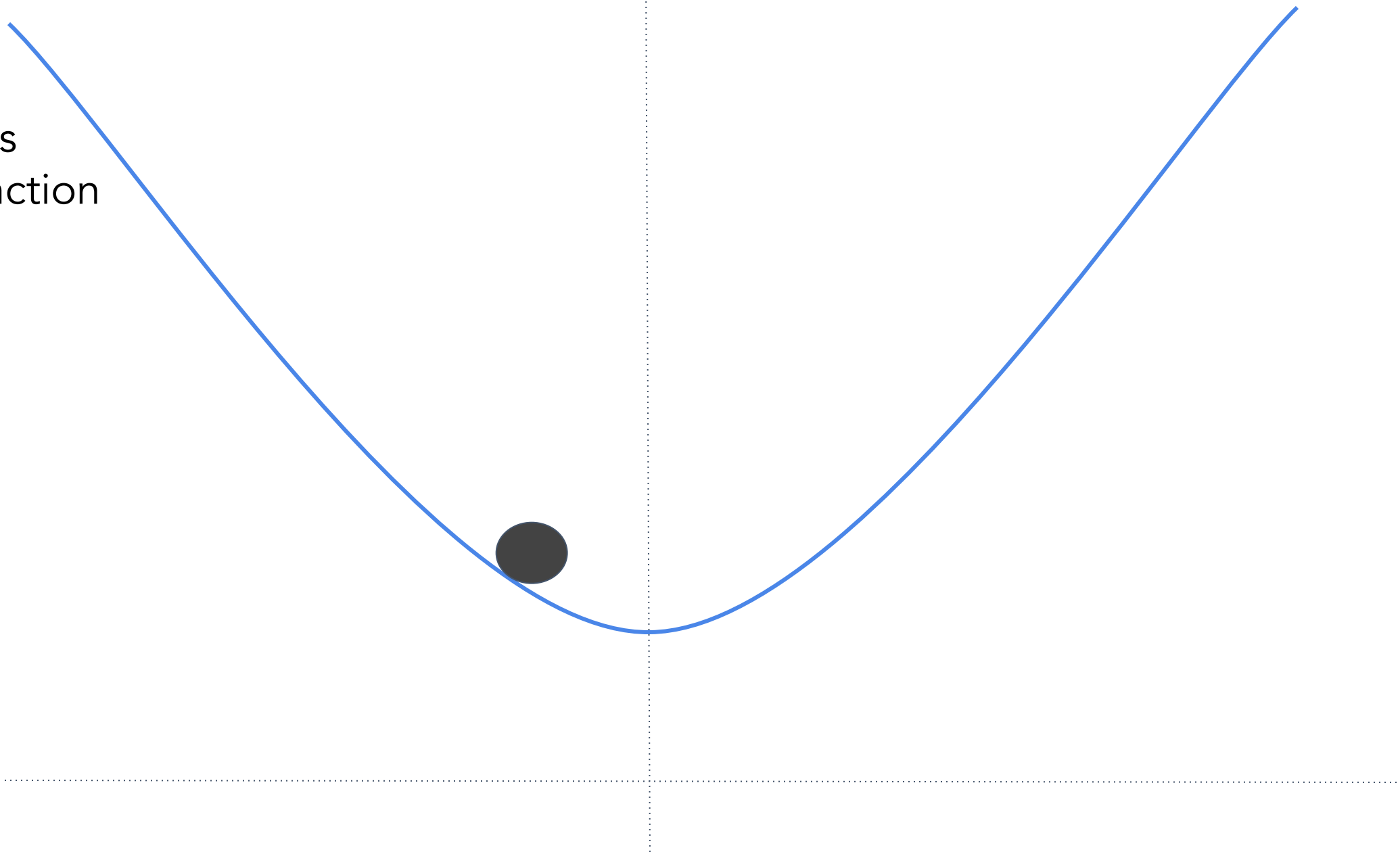


Loss  
Function

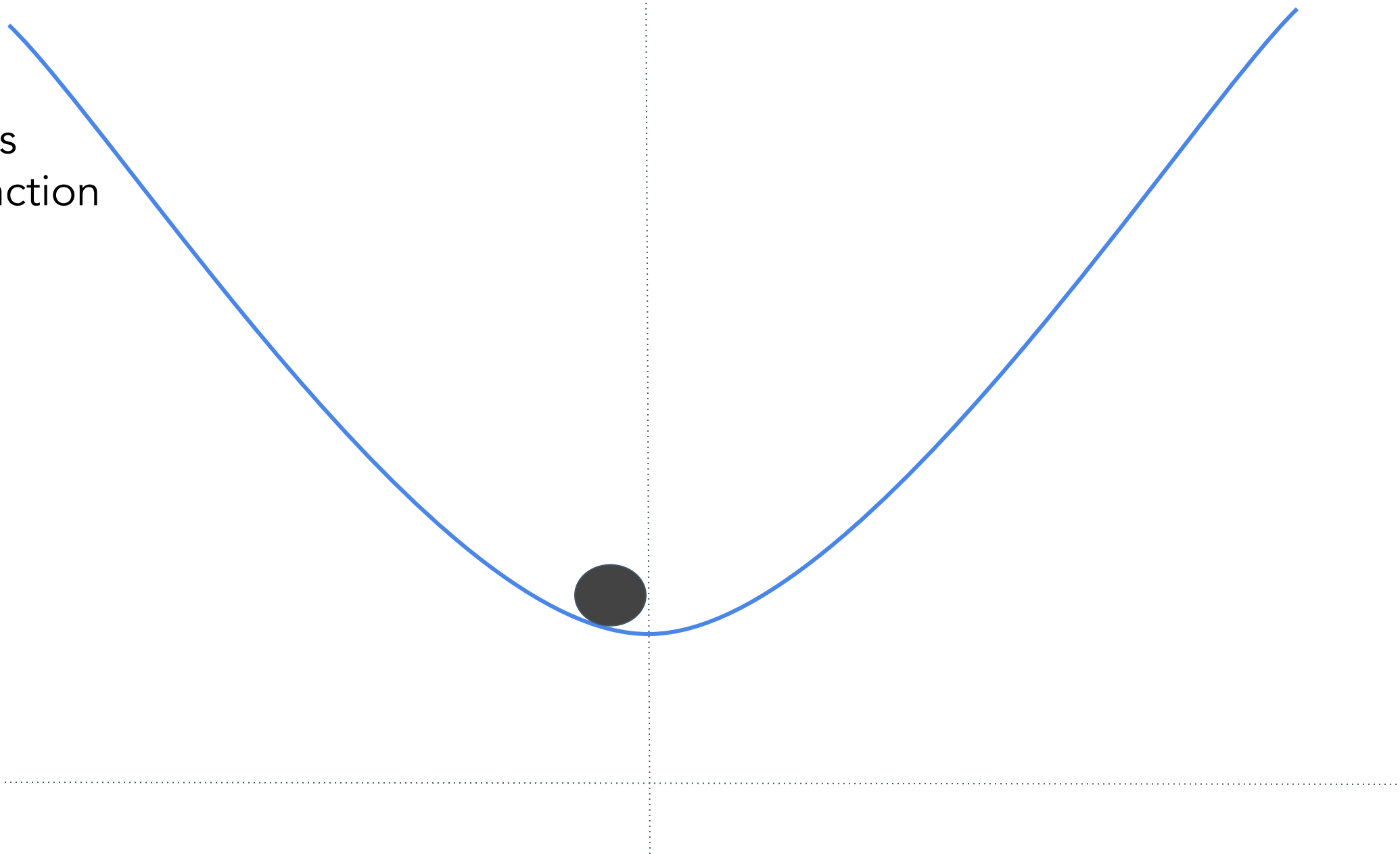
End Up here



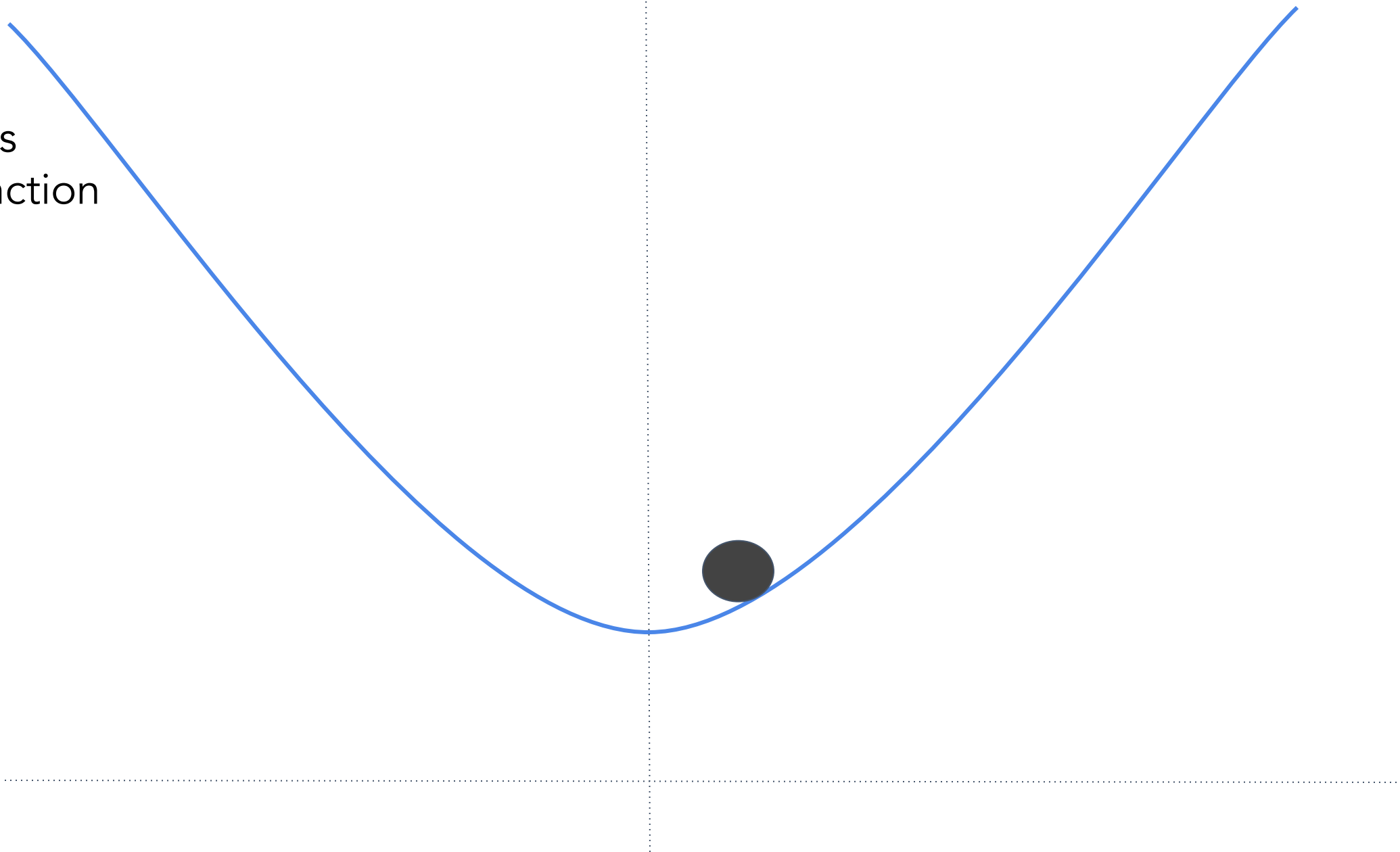
Loss  
Function



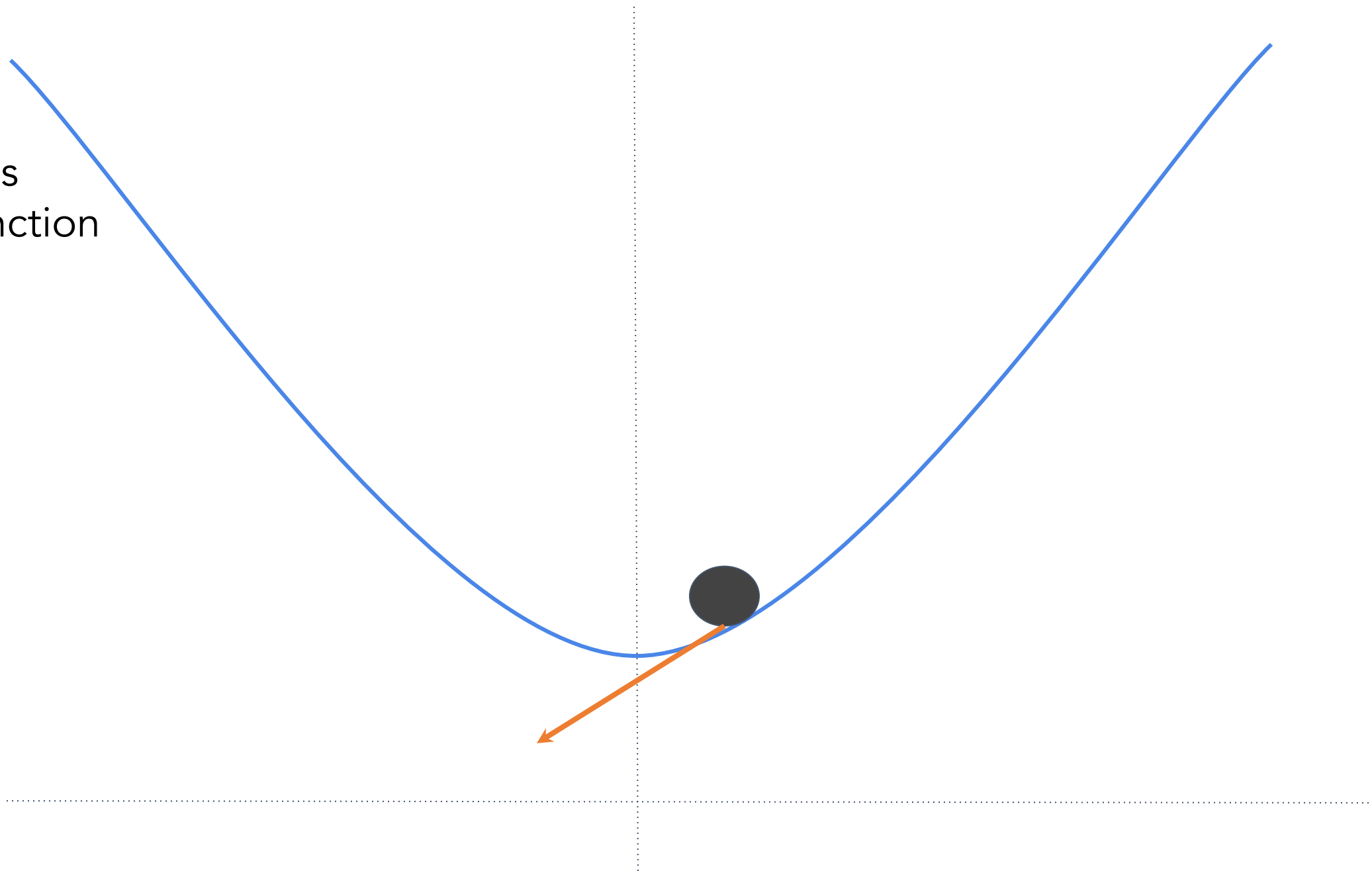
Loss  
Function



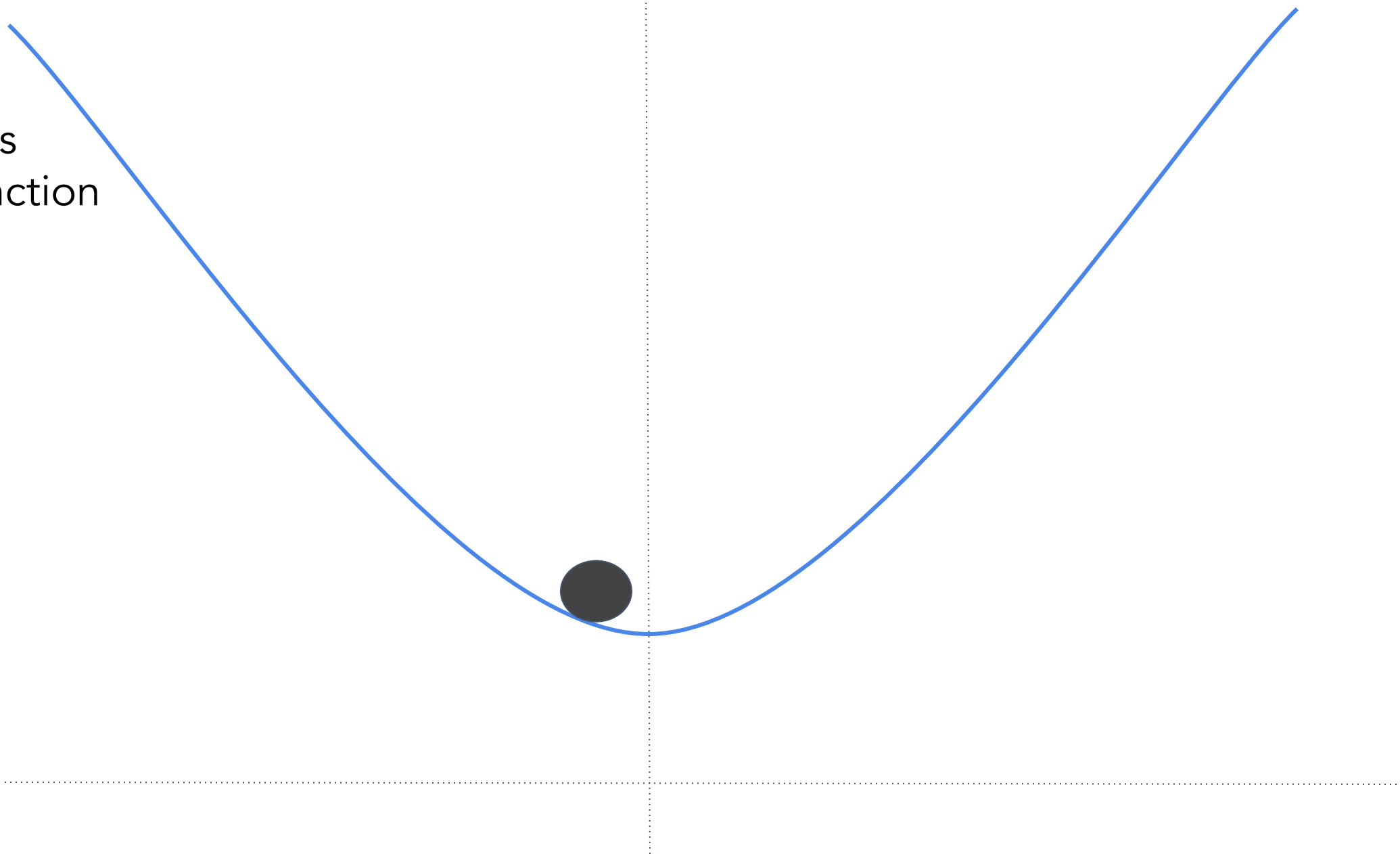
Loss  
Function



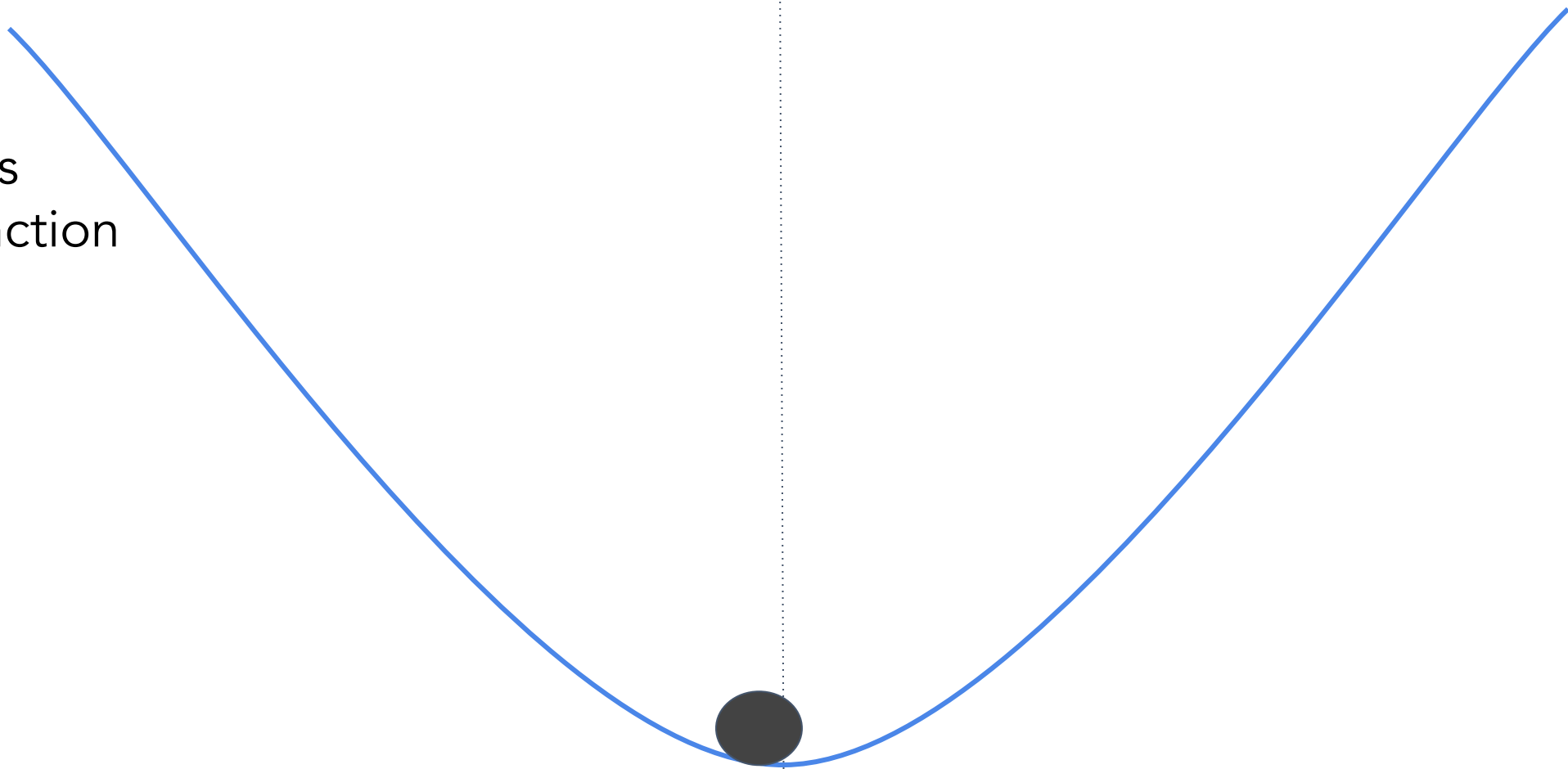
Loss  
Function



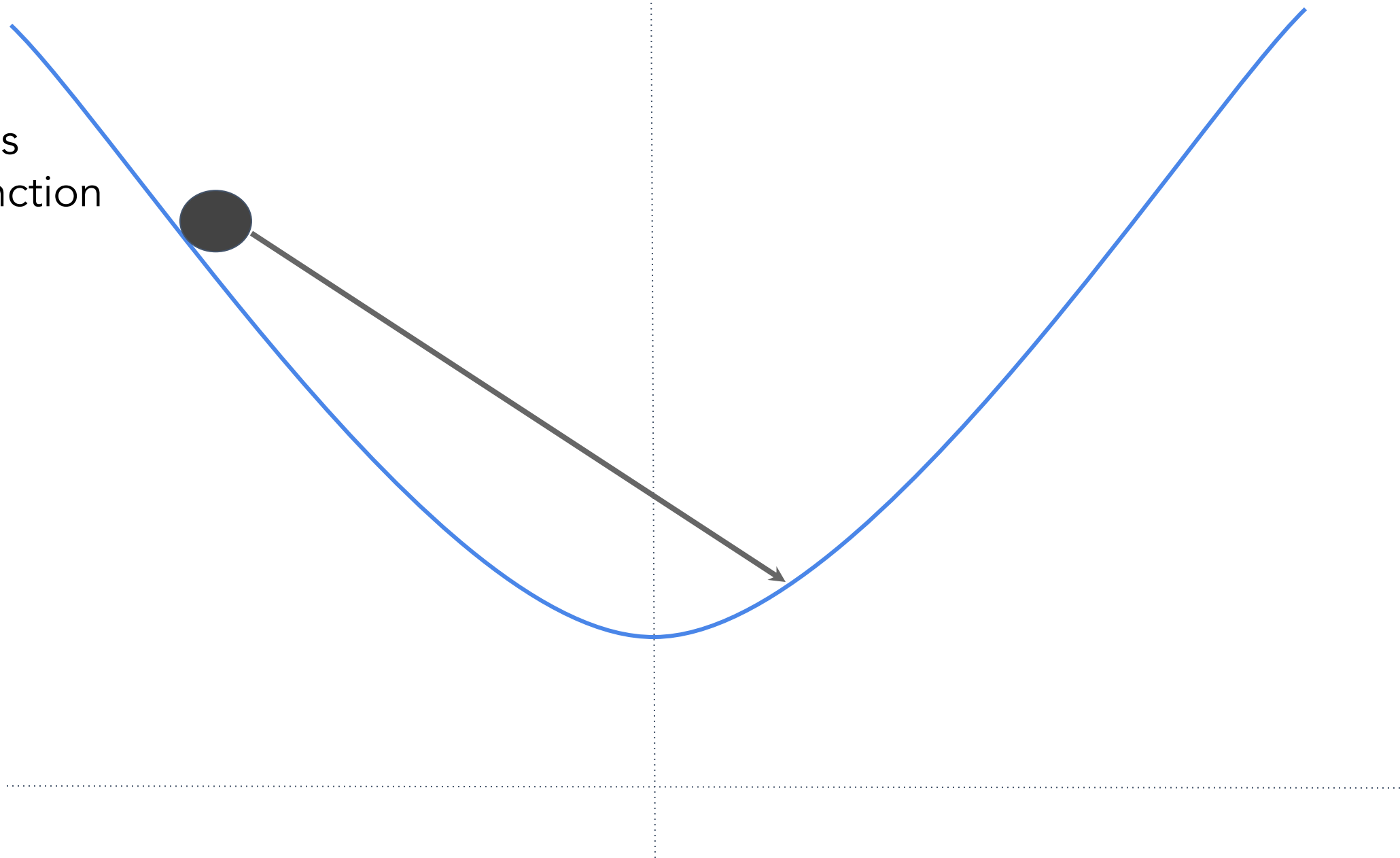
Loss  
Function



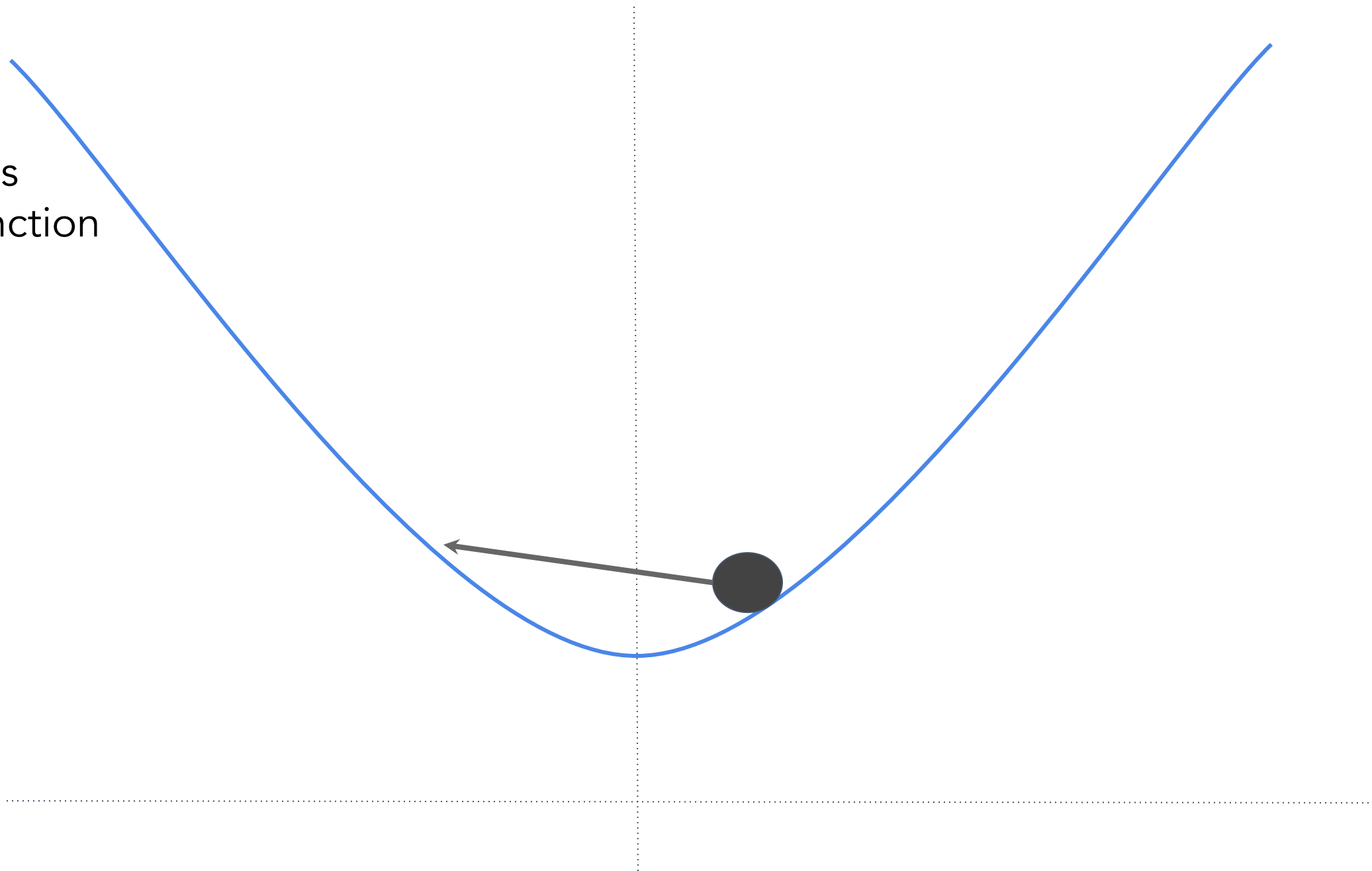
Loss  
Function



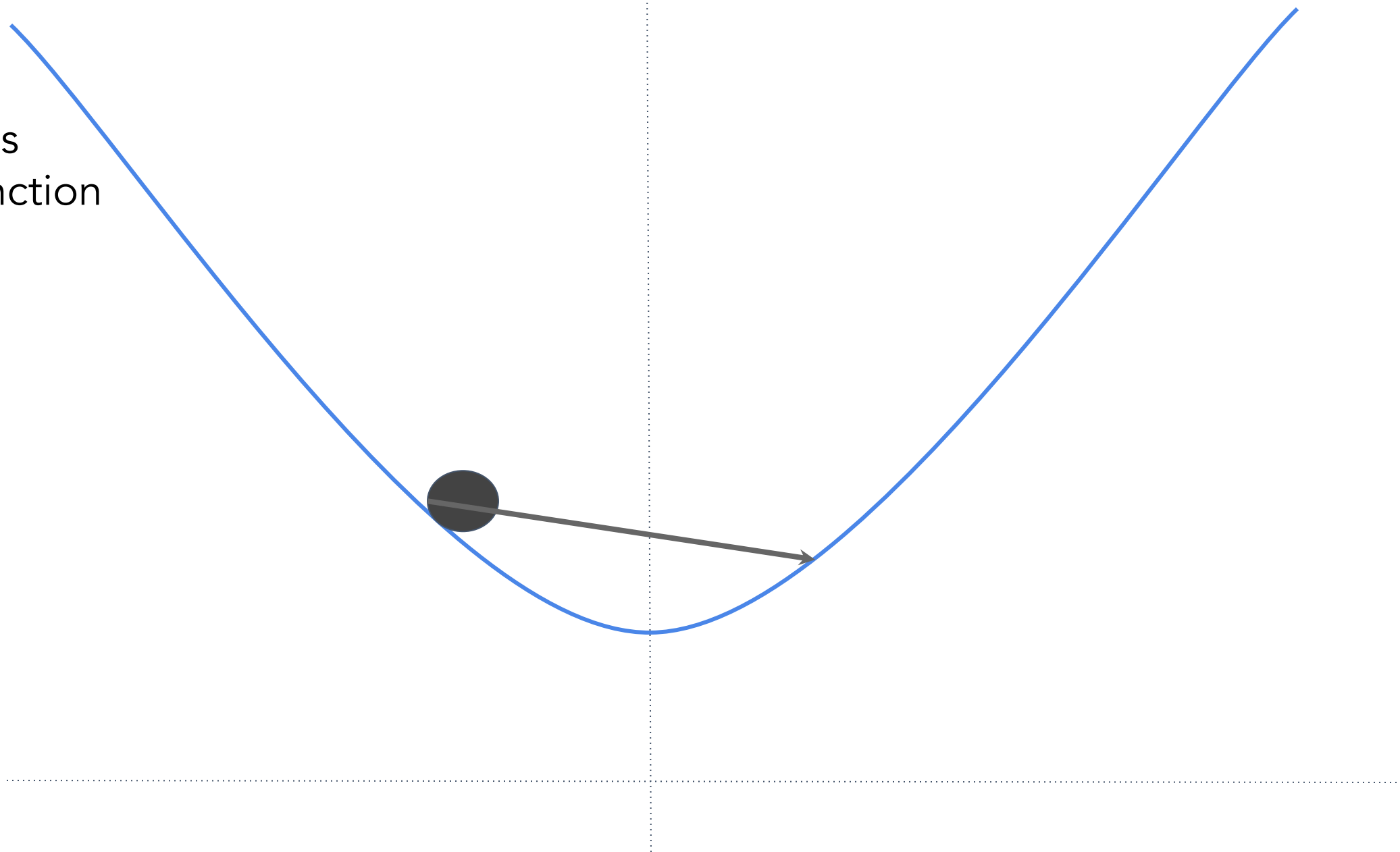
Loss  
Function



Loss  
Function



Loss  
Function



Loss  
Function

Move in Direction of Gradient



Loss  
Function

Move in Direction of Gradient



Loss  
Function

Move in Direction of Gradient



Loss  
Function

Move in Direction of Gradient

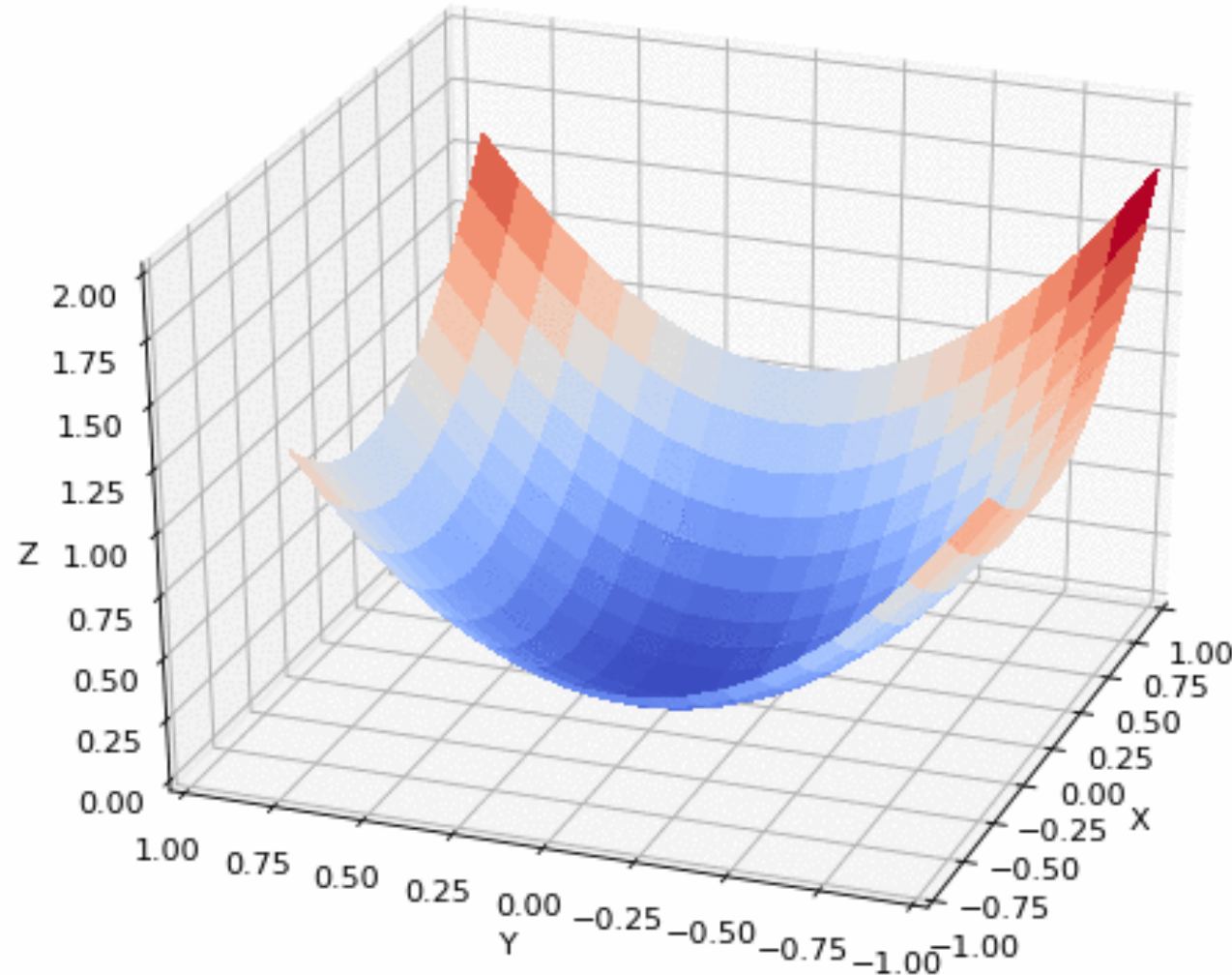


Loss  
Function

Move in Direction of Gradient

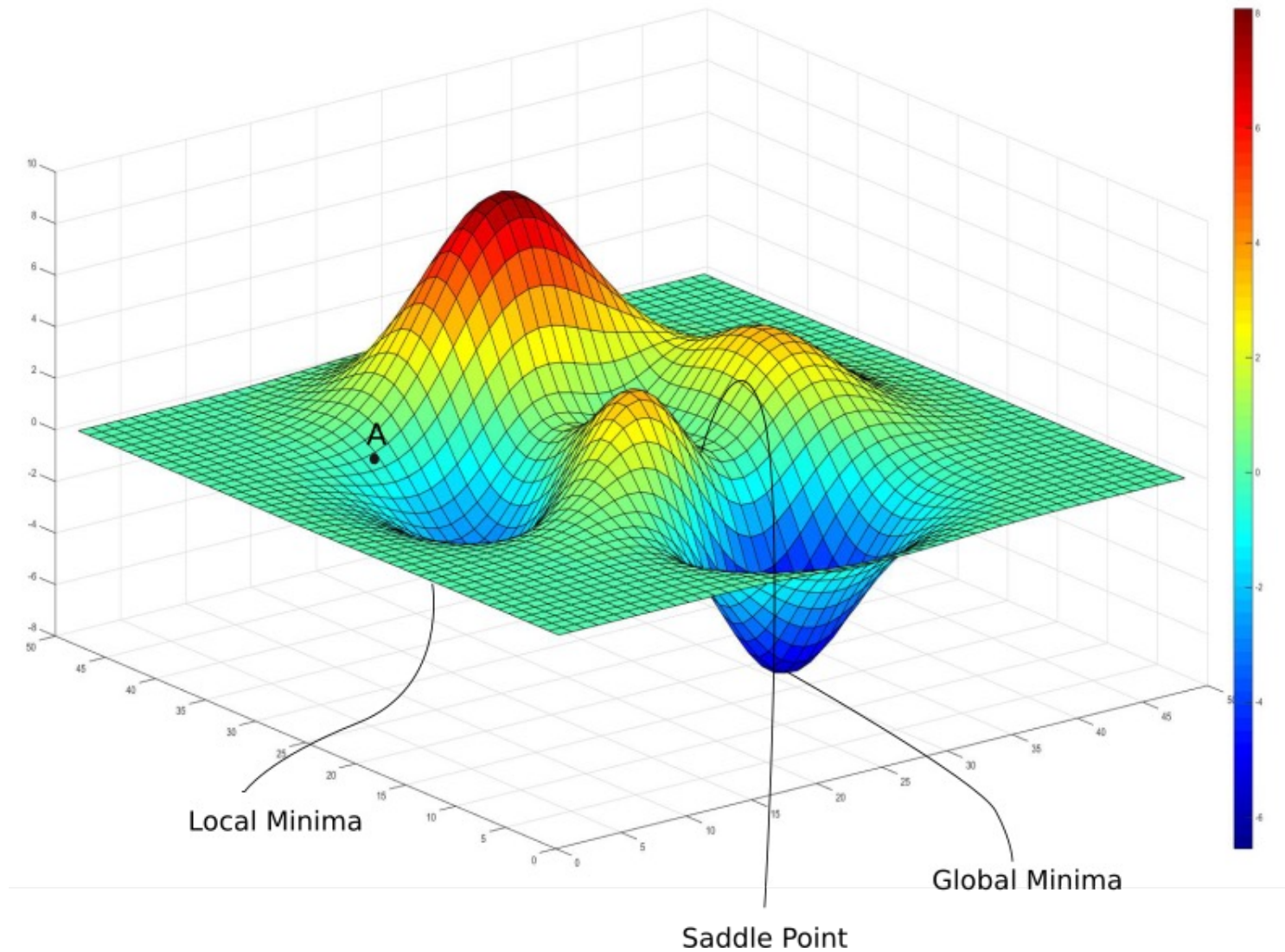



# Gradient Descent for Two Parameters



A single minima  
**Global minima**

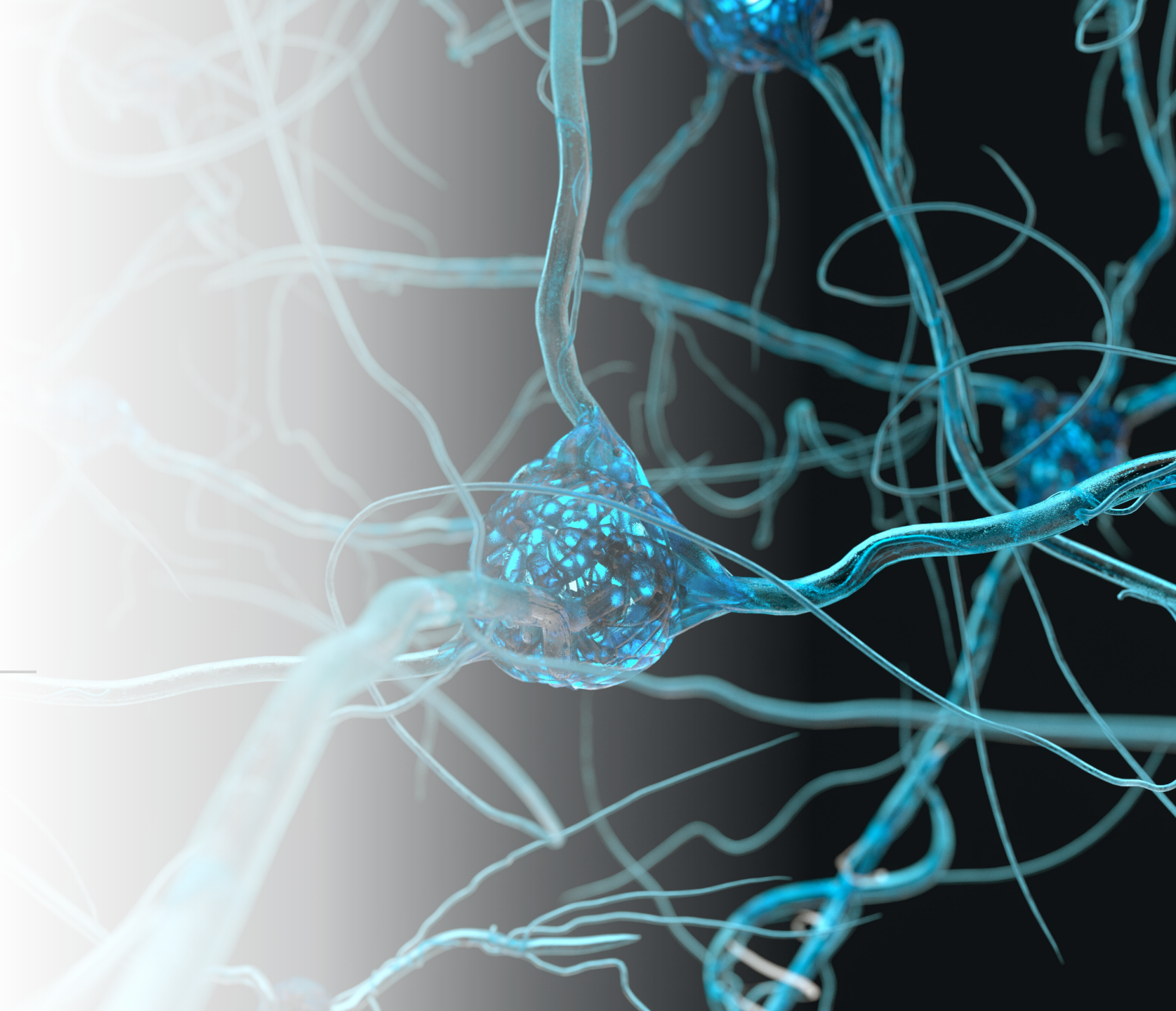
# Gradient Descent for Two Parameters



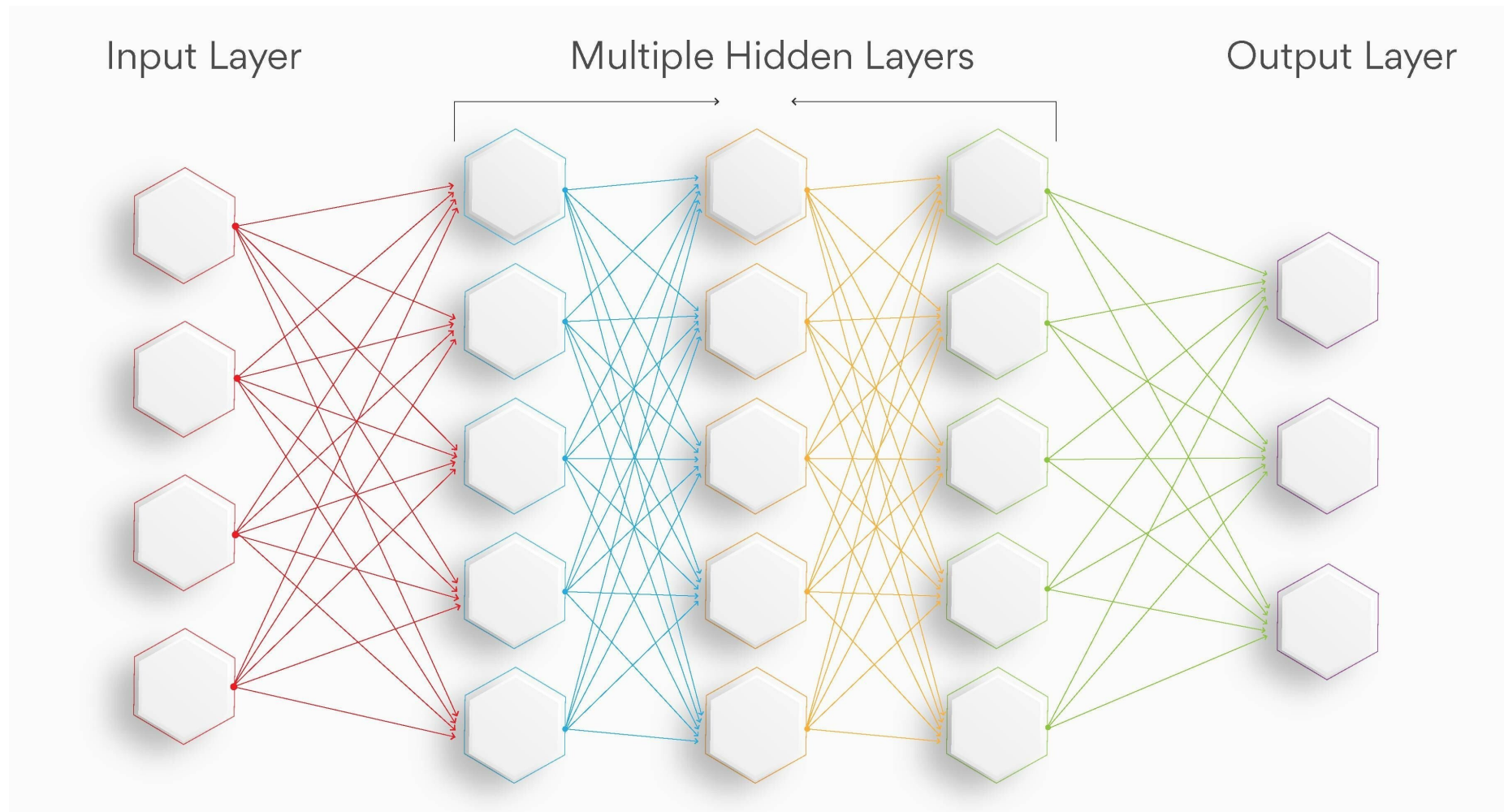


# Artificial Neural Networks

---

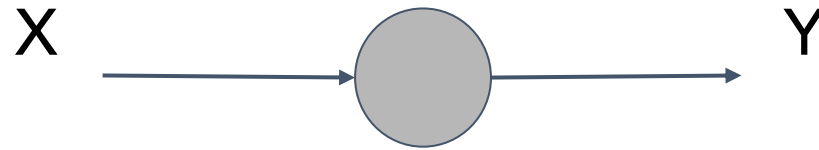


# What is an Artificial Neural Network (ANN)?



# A neuron

a neuron's output is a function of its inputs (in this case only one)



$$y = f(x) = wx + b$$

There are only **two parameters** to adjust:  
The **weight** for each input and a **bias**

First scenario: a regression

# Linear Regression with a Single Neuron

colab.research.google.com

Regression.ipynb

```
[2] import tensorflow as tf
import numpy as np
from tensorflow import keras
```

```
# define a neural network with one neuron
# for more information on TF functions see: https://www.tensorflow.org/api\_docs
my_layer = keras.layers.Dense(units=1, input_shape=[1])
model = tf.keras.Sequential([my_layer])
```

```
# use stochastic gradient descent for optimization and
# the mean squared error loss function
model.compile(optimizer='sgd', loss='mean_squared_error')
```

```
# define some training data (xs as inputs and ys as outputs)
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
```

```
# fit the model to the data (aka train the model)
model.fit(xs, ys, epochs=500)
```

1 layer, 1 neuron

Stochastic gradient descent

Inputs and outputs (labels)

Train the model

# Linear Regression with a Single Neuron

colab.research.google.com

Regression.ipynb

```
[2] import tensorflow as tf
import numpy as np
from tensorflow import keras

# define a neural network with one neuron
# for more information on TF functions see: https://www.tensorflow.org/api\_docs
my_layer = keras.layers.Dense(units=1, input_shape=[1])
model = tf.keras.Sequential([my_layer])

# use stochastic gradient descent for optimization and
# the mean squared error loss function
model.compile(optimizer='sgd', loss='mean_squared_error')

# define some training data (xs as inputs and ys as outputs)
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

# fit the model to the data (aka train the model)
model.fit(xs, ys, epochs=500)
```

```
Epoch 500/500
1/1 [=====] - 0s 6ms/step - loss: 3.4704e-05
<keras.callbacks.History at 0x7f1d6ccd7f10>
```

```
[4] print(model.predict([10.0]))

[[18.982813]]
```

```
[5] print(model.predict(xs))

[[-2.9897861]
 [-0.992277 ]
 [ 1.005232 ]
 [ 3.0027409]
 [ 5.00025  ]
 [ 6.997759 ]]
```

```
[6] print(my_layer.get_weights())

[array([[1.997509]], dtype=float32), array([-0.992277], dtype=float32)]
```

# Linear Regression with a Single Neuron

colab.research.google.com

Regression.ipynb

```
[2] import tensorflow as tf
import numpy as np
from tensorflow import keras

# define a neural network with one neuron
# for more information on TF functions see: https://www.tensorflow.org/api\_docs
my_layer = keras.layers.Dense(units=1, input_shape=[1])
model = tf.keras.Sequential([my_layer])

# use stochastic gradient descent for optimization and
# the mean squared error loss function
model.compile(optimizer='sgd', loss='mean_squared_error')

# define some training data (xs as inputs and ys as outputs)
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

# fit the model to the data (aka train the model)
model.fit(xs, ys, epochs=500)
```

$$Y = 2X - 1$$

```
Epoch 500/500
1/1 [=====] - 0s 6ms/step - loss: 3.4704e-05
<keras.callbacks.History at 0x7f1d6ccd7f10>
```

```
[4] print(model.predict([10.0]))

[[18.982813]]
```

```
[5] print(model.predict(xs))

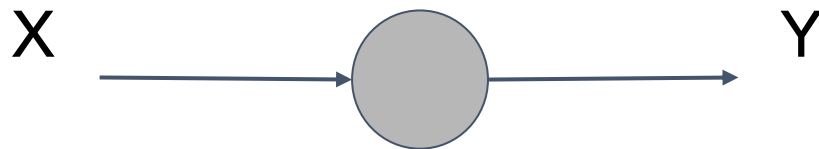
[[-2.9897861]
 [-0.992277 ]
 [ 1.005232 ]
 [ 3.0027409]
 [ 5.00025 ]
 [ 6.997759 ]]
```

```
[6] print(my_layer.get_weights())

[array([[1.997509]], dtype=float32), array([-0.992277], dtype=float32)]
```

$$Y = 1.9975X - 0.9922$$

Not perfect,  
but good enough for most cases!



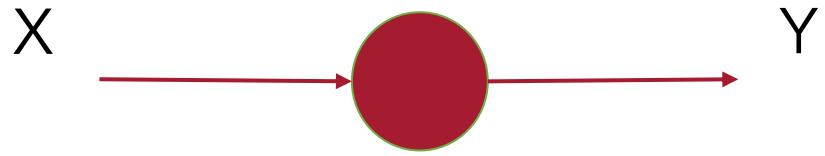
$$y = f(x) = \mathbf{w}x + \mathbf{b}$$

$$y = \mathbf{1.9975}x - \mathbf{0.9922}$$

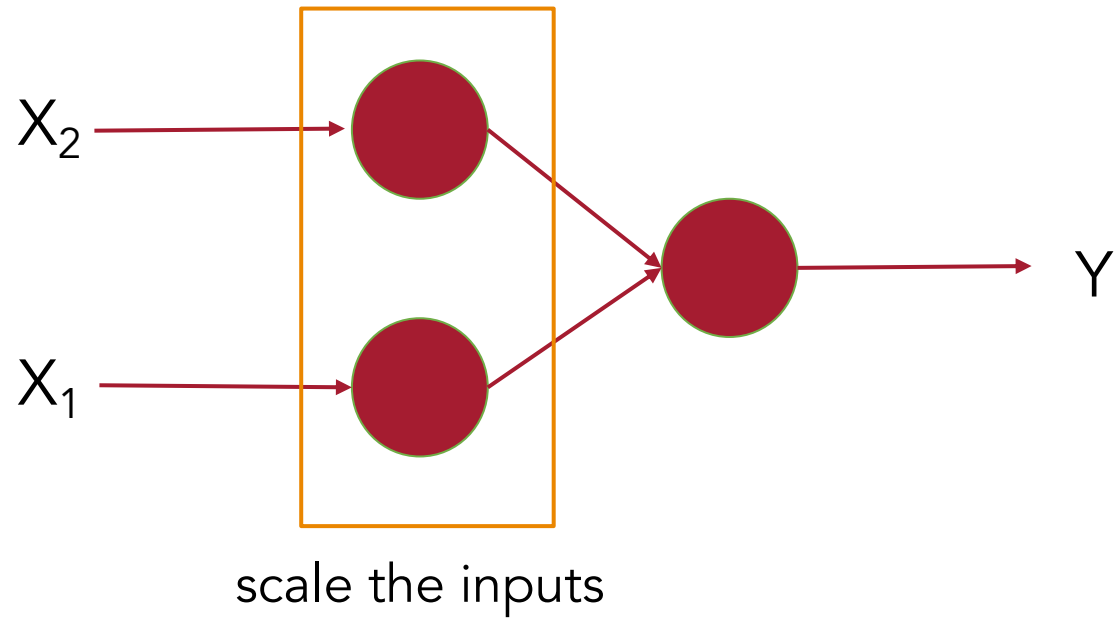
# Now, Classification

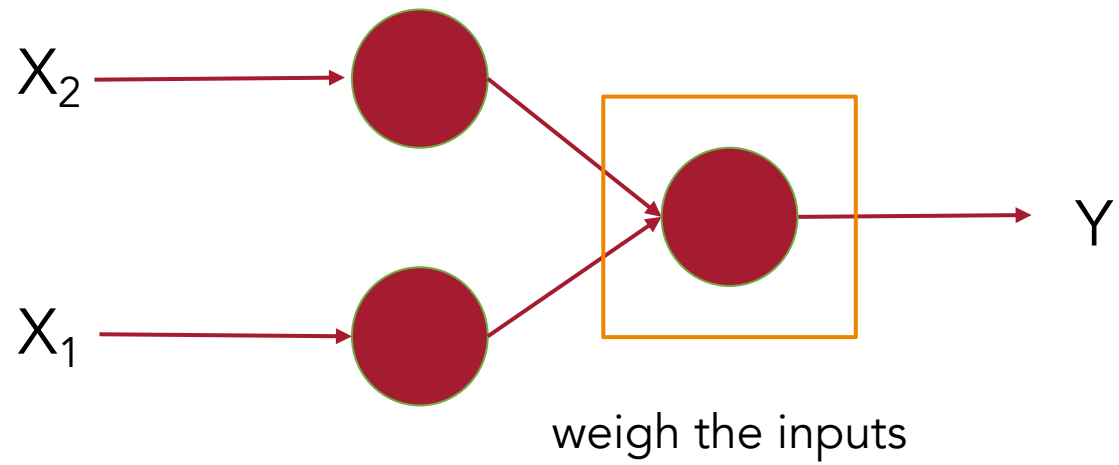
---



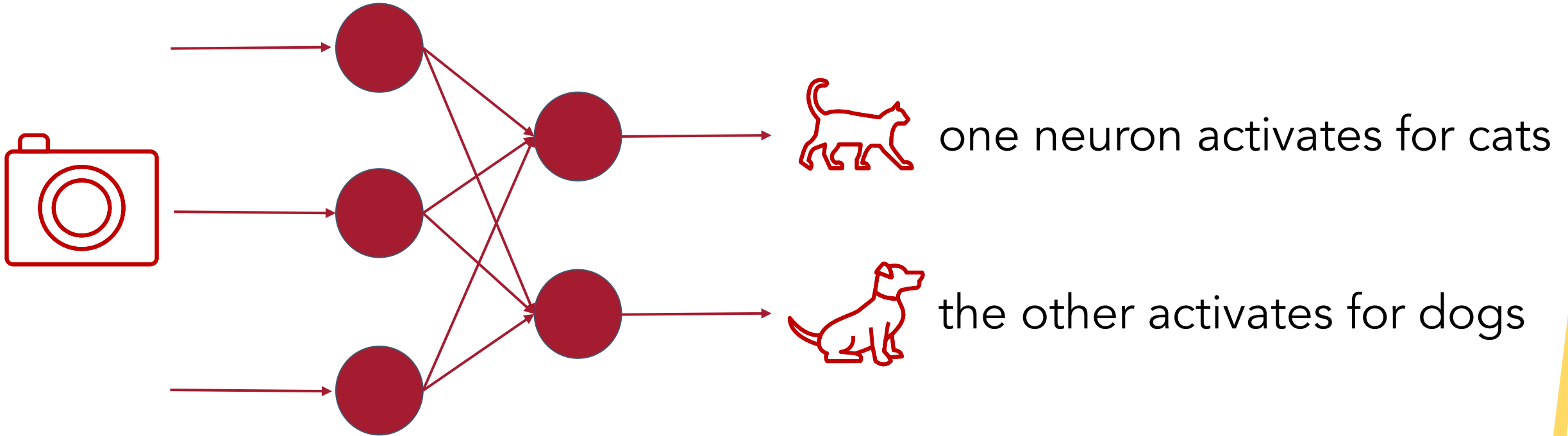


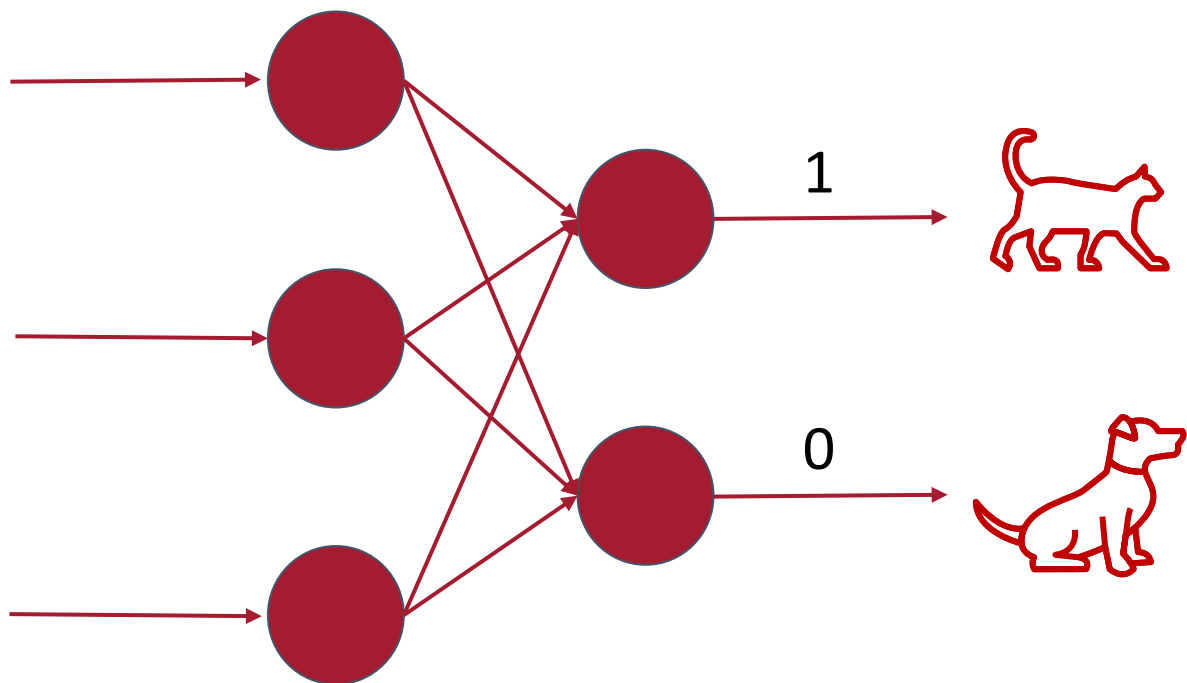
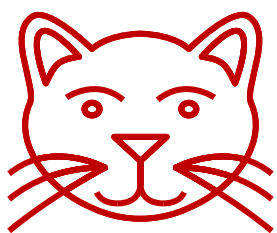
What about more than one input?

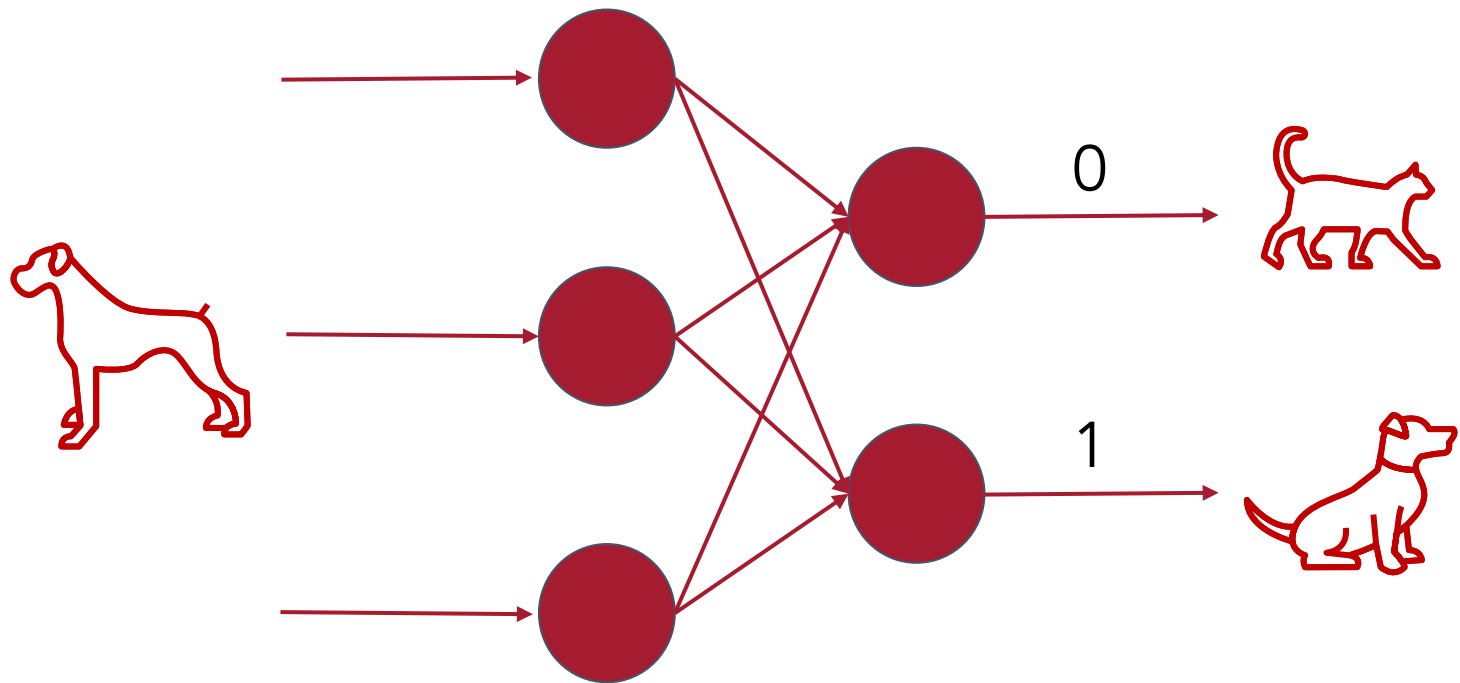




More inputs?

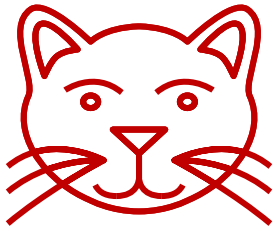




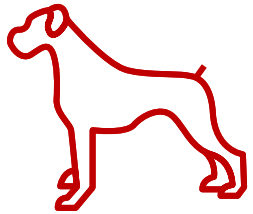


Data

Label



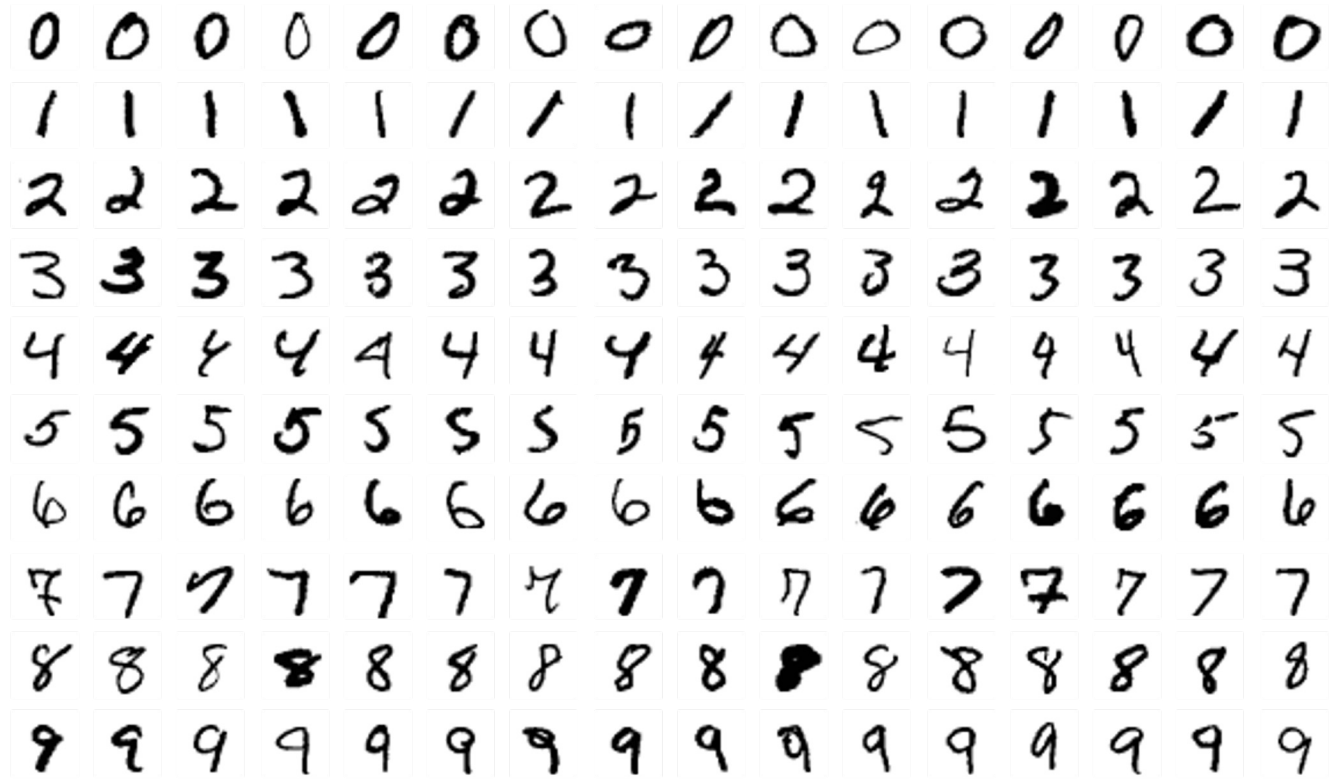
[ 1, 0 ]



[ 0, 1 ]

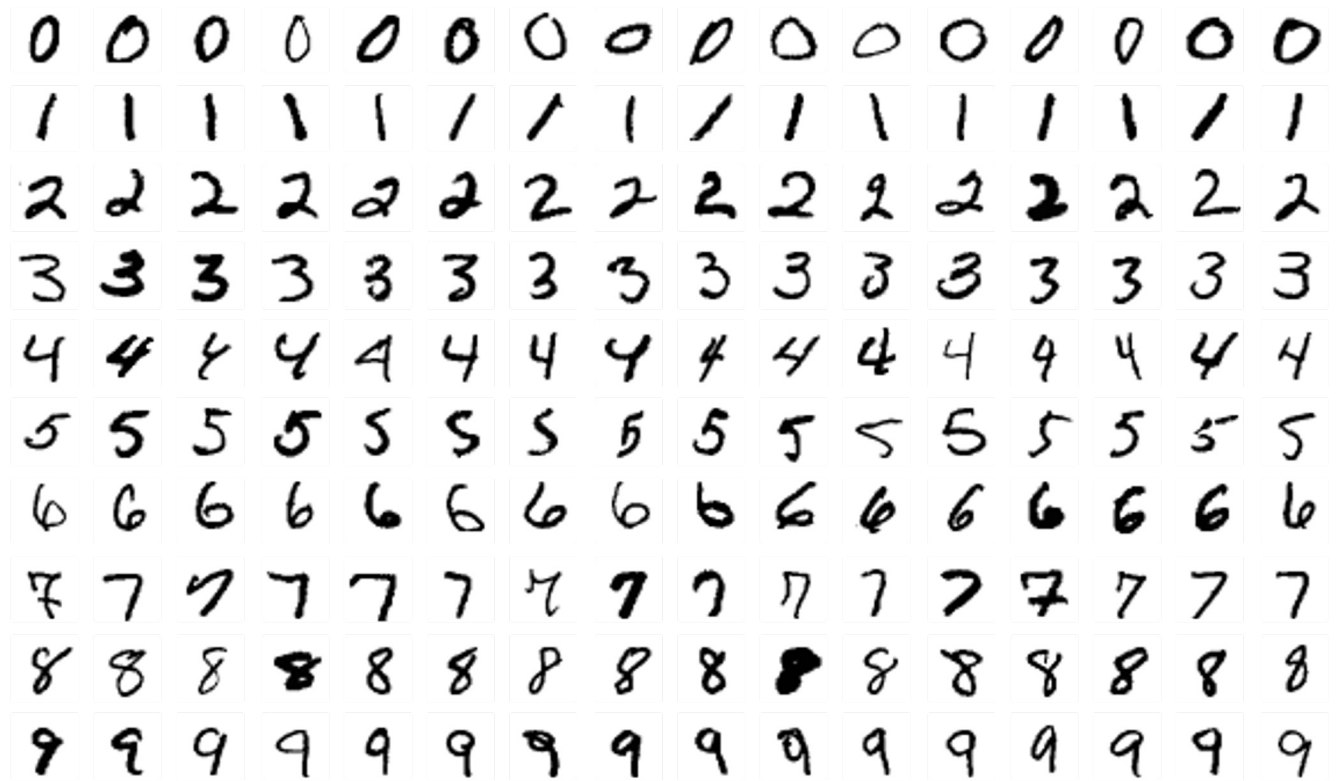
We can extend this example to other domains

0	[ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
1	[ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
2	[ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ]
3	[ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 ]
4	[ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ]
5	[ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 ]
6	[ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 ]
7	[ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 ]
8	[ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ]
9	[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 ]



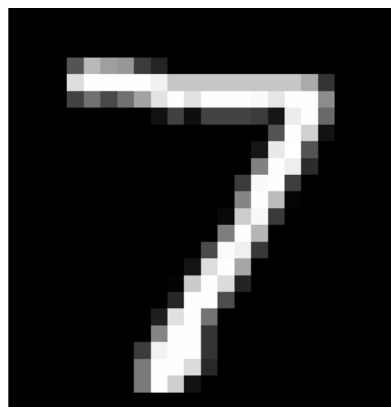
The **MNIST** (Modified National Institute of Standards and Technology database) is a large database of **handwritten digits** that is **commonly used for training** various image processing systems.

0	[ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
1	[ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
2	[ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ]
3	[ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 ]
4	[ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ]
5	[ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 ]
6	[ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 ]
7	[ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 ]
8	[ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ]
9	[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 ]

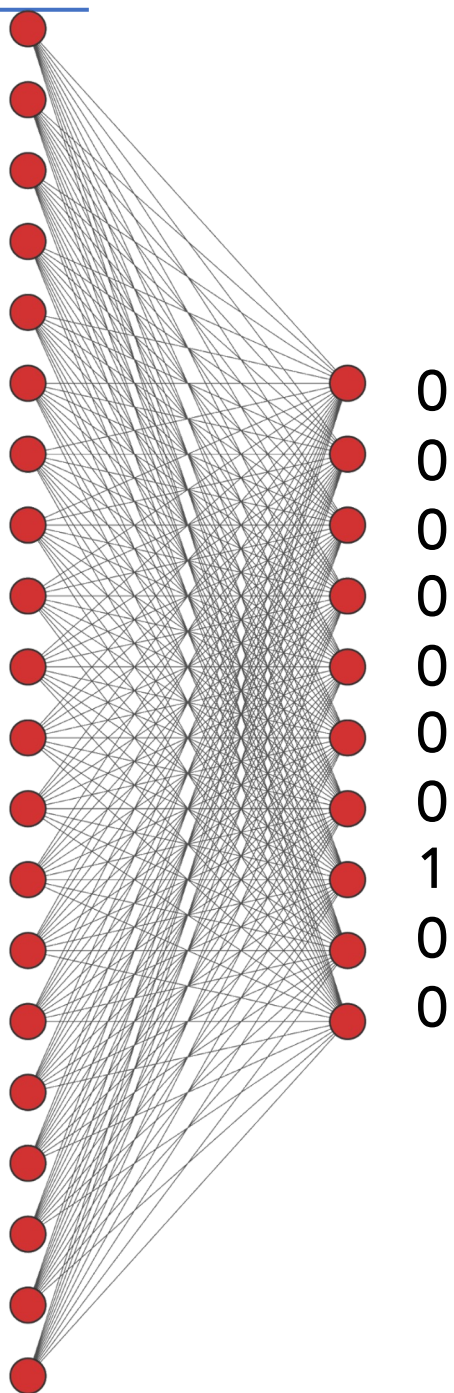


60,000 Labelled Training Examples  
10,000 Labelled Validation Examples

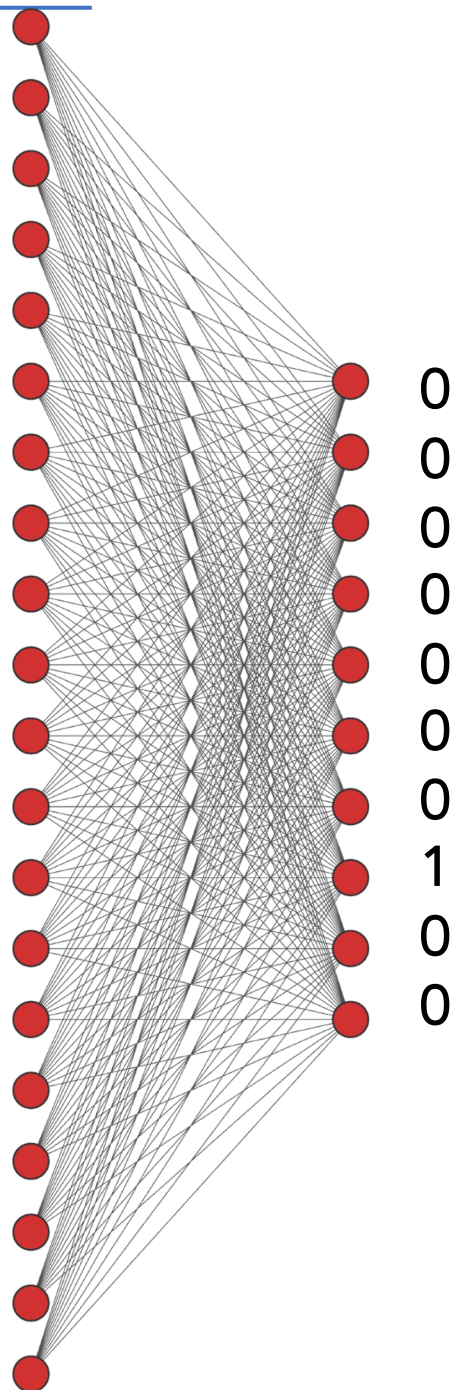
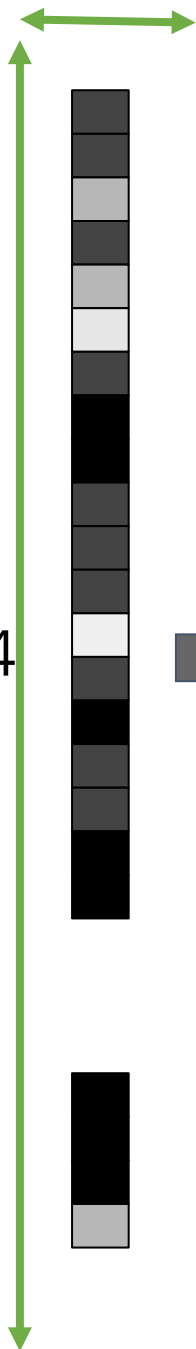
28  
px



28  
px



784



# a NN to classify the MNIST DB

[colab.research.google.com](https://colab.research.google.com)

[MNIST\\_NN.ipynb](#)

```
import tensorflow as tf
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (val_images, val_labels) = mnist.load_data()
training_images=training_images / 255.0
val_images=val_images / 255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(20, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_labels, validation_data=(val_images, val_labels), epochs=20)
```

# a NN to classify the MNIST DB

[colab.research.google.com](https://colab.research.google.com)

[MNIST\\_NN.ipynb](#)

```
Epoch 9/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3555 - accuracy: 0.8724 - val_loss: 0.4090 - val_accuracy: 0.8516
Epoch 10/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3509 - accuracy: 0.8752 - val_loss: 0.4061 - val_accuracy: 0.8537
Epoch 11/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3452 - accuracy: 0.8768 - val_loss: 0.3980 - val_accuracy: 0.8580
Epoch 12/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3398 - accuracy: 0.8783 - val_loss: 0.4052 - val_accuracy: 0.8586
Epoch 13/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3355 - accuracy: 0.8798 - val_loss: 0.4160 - val_accuracy: 0.8533
Epoch 14/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3332 - accuracy: 0.8812 - val_loss: 0.3913 - val_accuracy: 0.8609
Epoch 15/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3279 - accuracy: 0.8818 - val_loss: 0.3971 - val_accuracy: 0.8588
Epoch 16/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3250 - accuracy: 0.8839 - val_loss: 0.3945 - val_accuracy: 0.8597
Epoch 17/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3221 - accuracy: 0.8839 - val_loss: 0.3985 - val_accuracy: 0.8578
Epoch 18/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3184 - accuracy: 0.8853 - val_loss: 0.3988 - val_accuracy: 0.8595
Epoch 19/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3158 - accuracy: 0.8857 - val_loss: 0.3984 - val_accuracy: 0.8578
Epoch 20/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.3140 - accuracy: 0.8856 - val_loss: 0.4069 - val_accuracy: 0.8567
<keras.callbacks.History at 0x7fe50180b750>
```

# a NN to classify the MNIST DB

[colab.research.google.com](https://colab.research.google.com)

[MNIST\\_NN.ipynb](#)

```
Epoch 19/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.3022 - accuracy: 0.8914 - val_loss: 0.3834 - val_accuracy: 0.8659
Epoch 20/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.2996 - accuracy: 0.8910 - val_loss: 0.3911 - val_accuracy: 0.8642
<keras.callbacks.History at 0x7f033e5f5bd0>
```

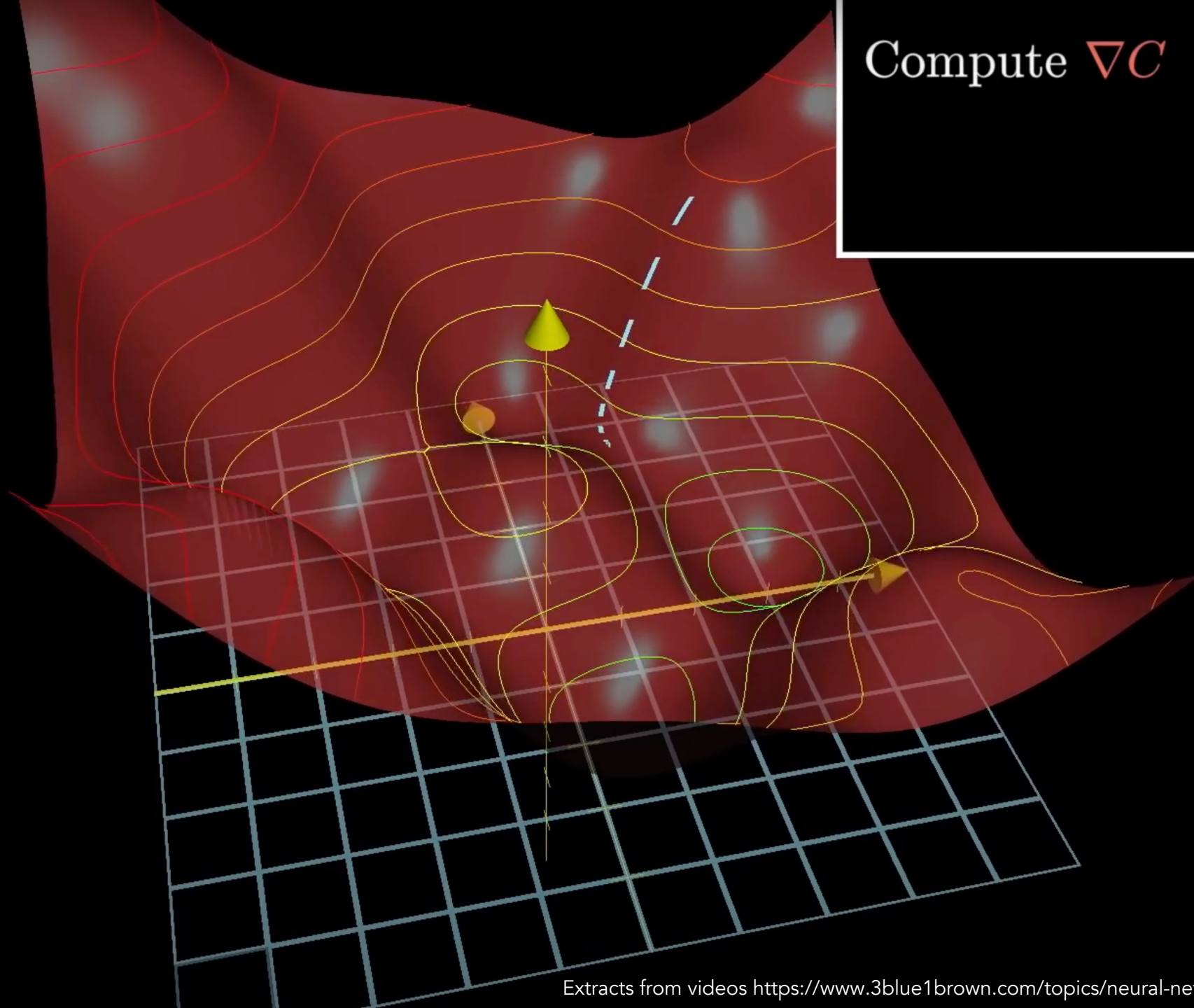
```
▶ model.evaluate(val_images, val_labels)

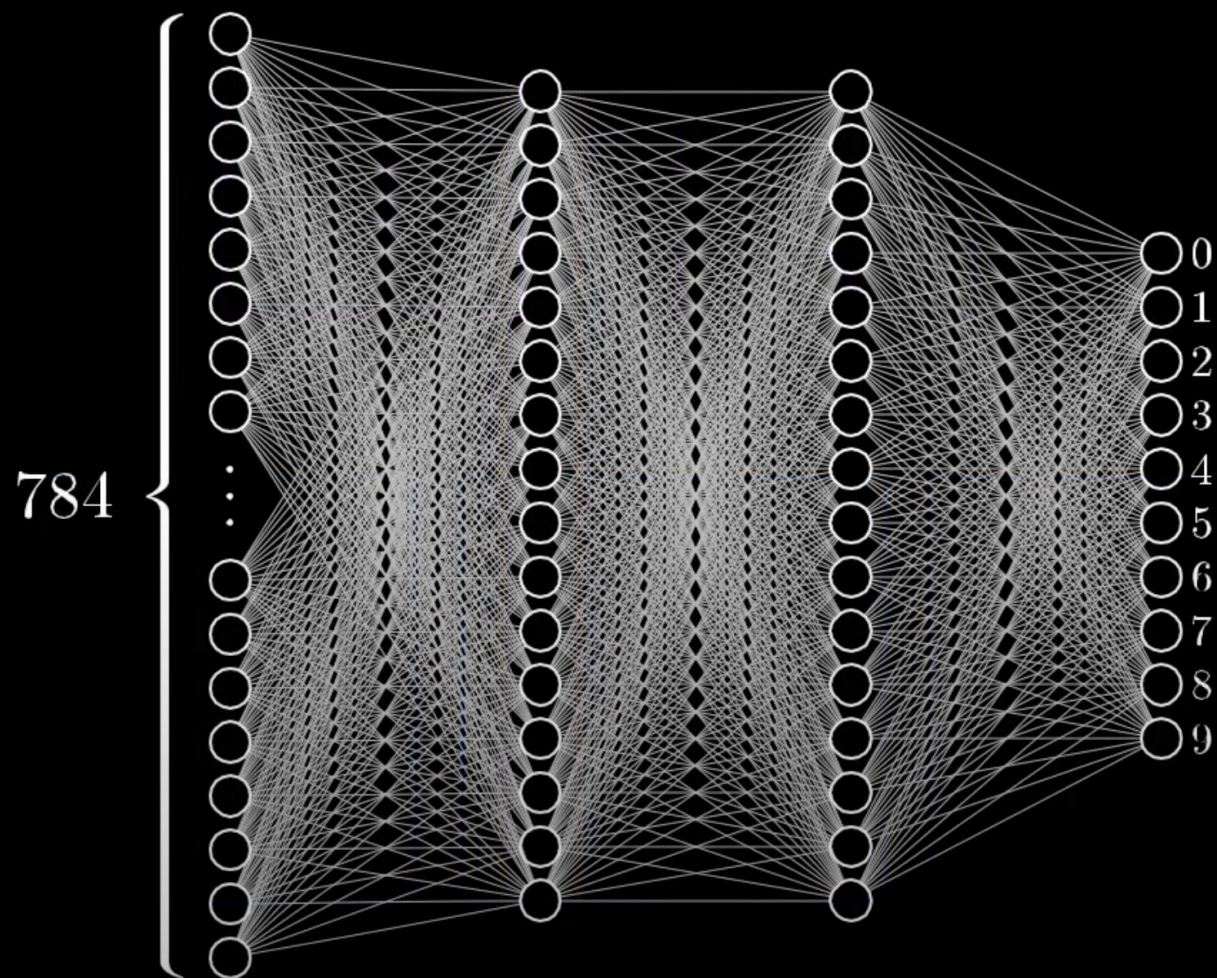
classifications = model.predict(val_images)
print(classifications[0])
print(val_labels[0])
```

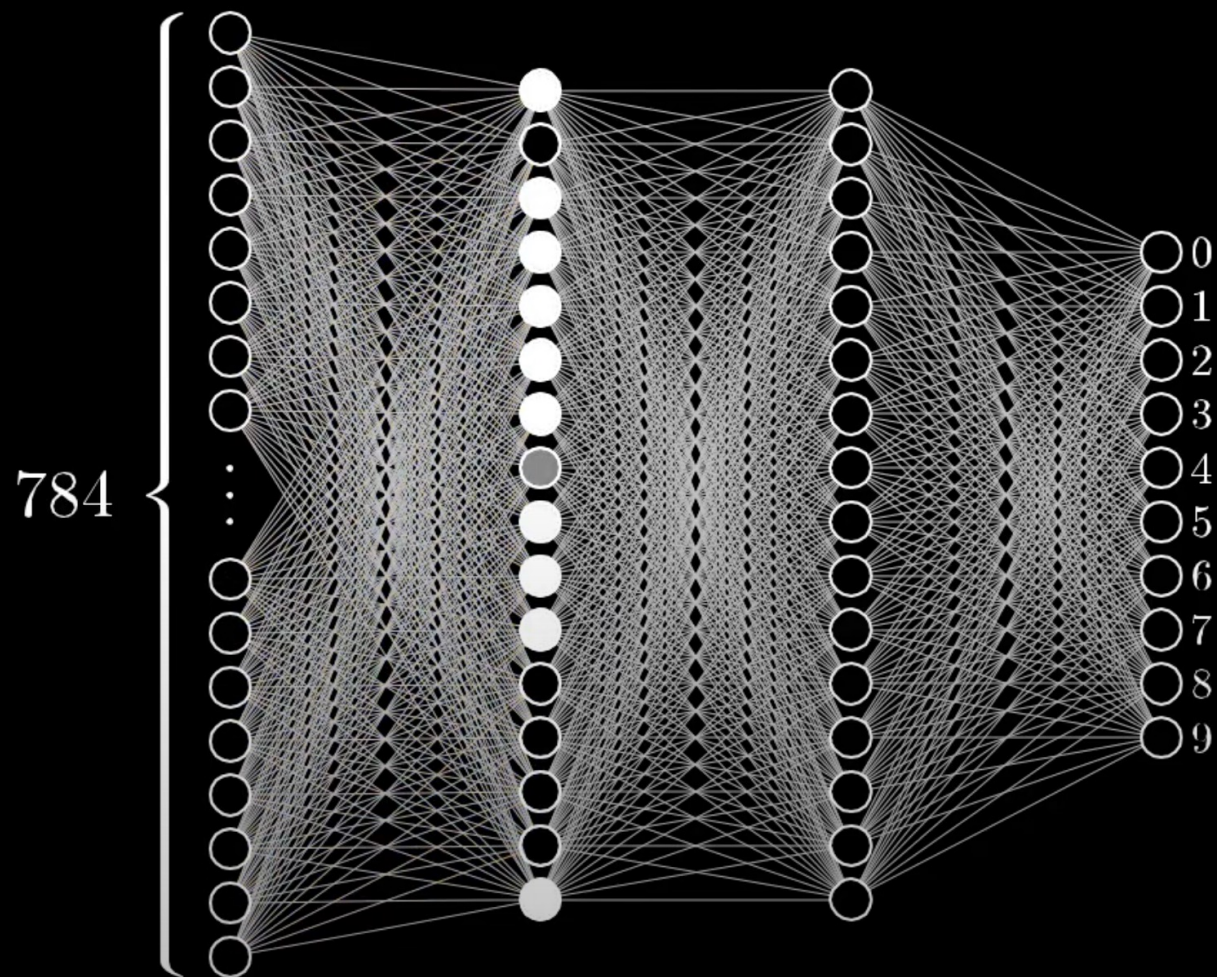
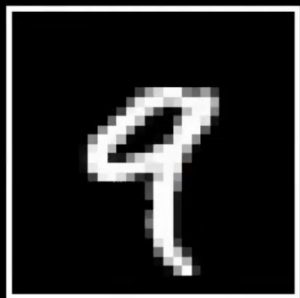
```
313/313 [=====] - 0s 1ms/step - loss: 0.3911 - accuracy: 0.8642
[5.2699960e-09 4.4460235e-10 2.9260536e-07 1.1081011e-04 1.4583268e-08
 8.1817927e-03 5.3513944e-09 5.8446459e-02 2.9248906e-05 9.3323141e-01]
```

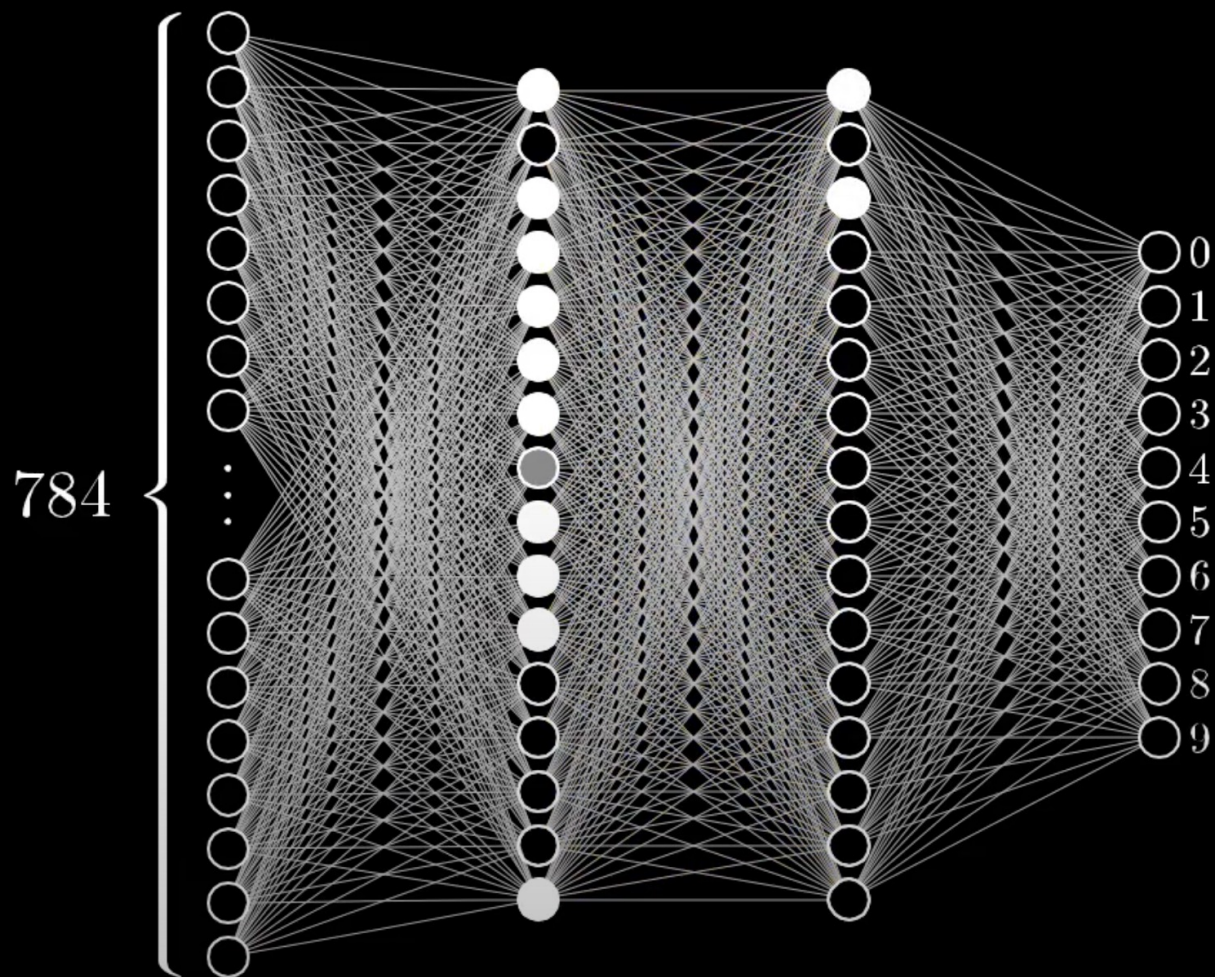
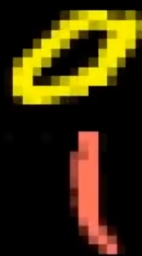
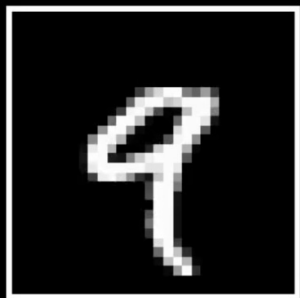
9

Compute  $\nabla C$

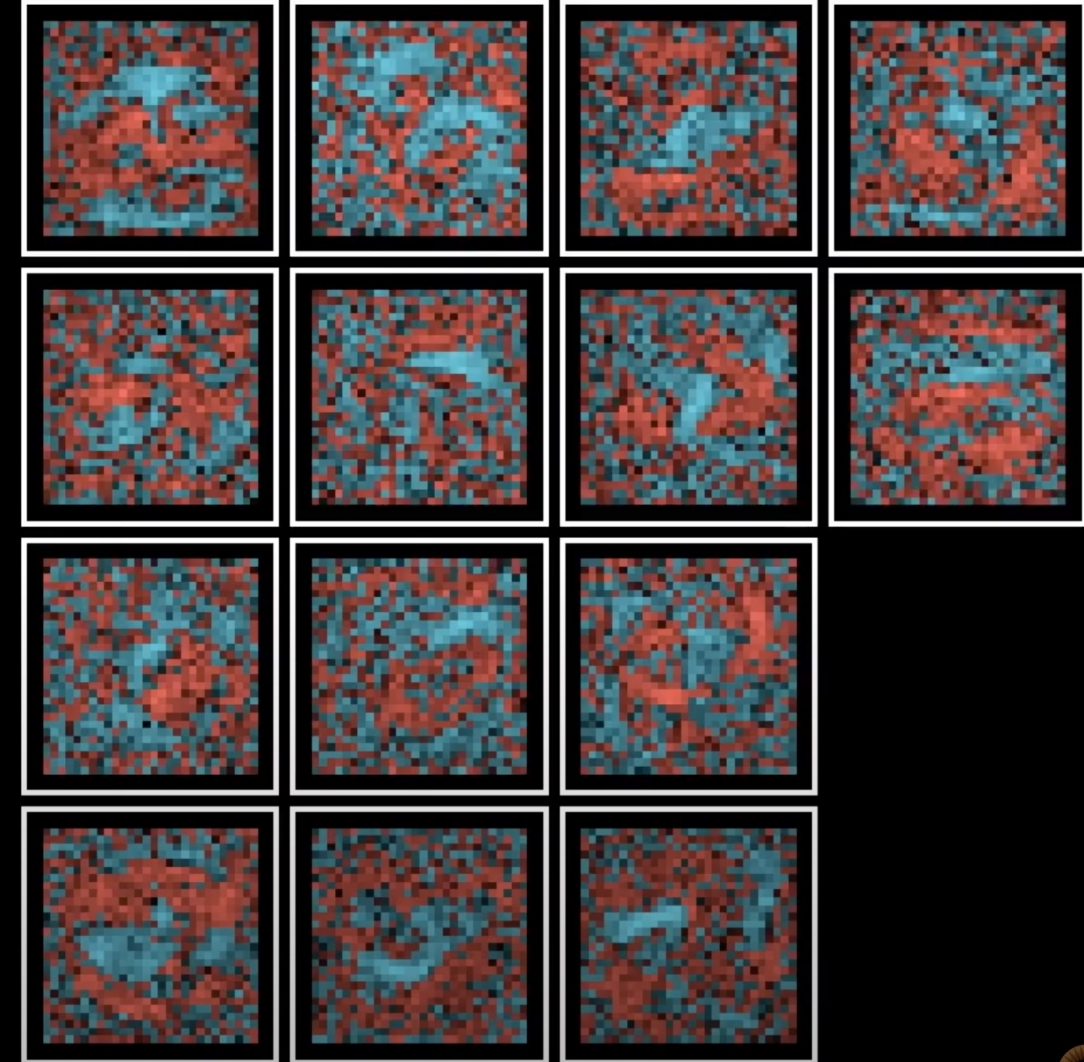
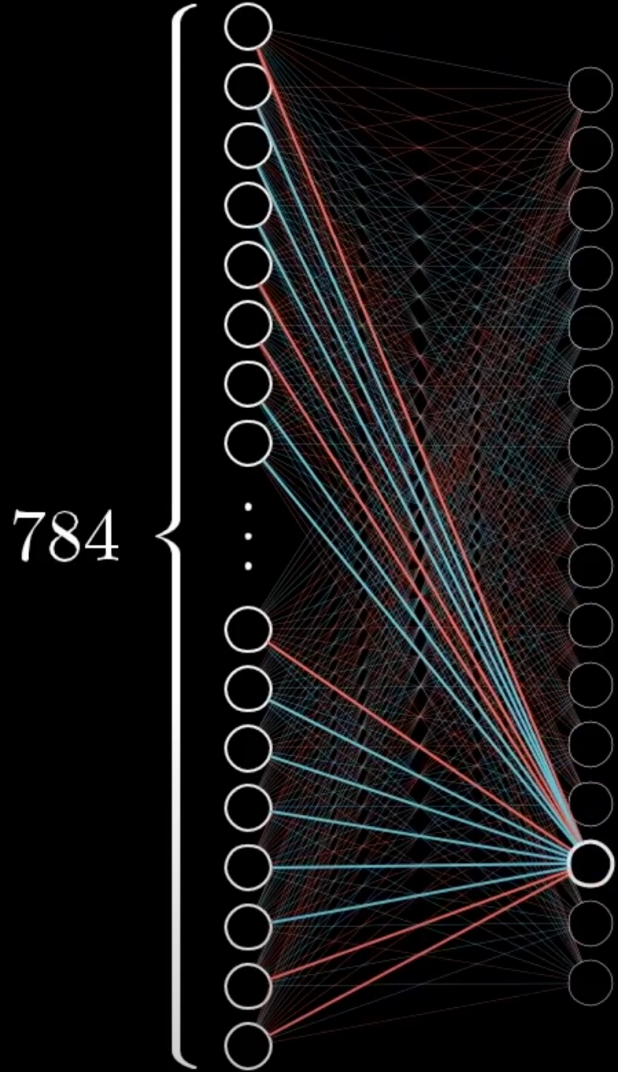


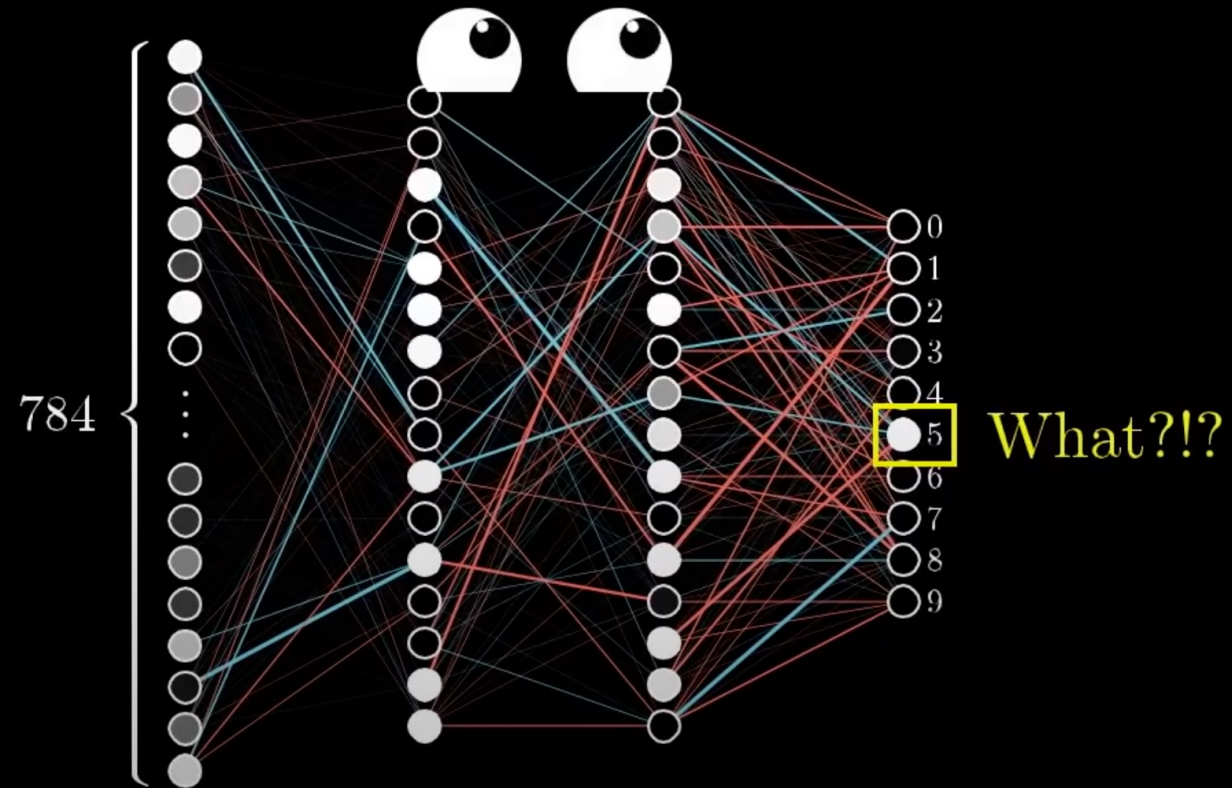
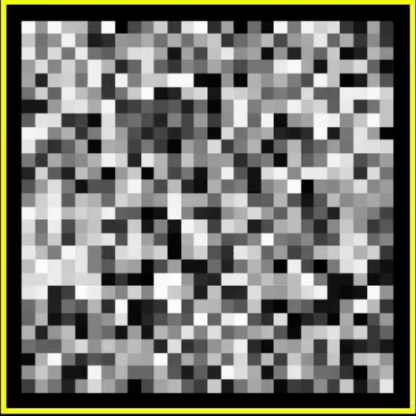






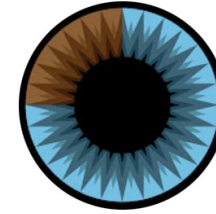
# What second layer neurons look for





# A very nice introduction to NN

- 3Blue1Brown playlist on Neural Networks
  - **But what is a neural network?**
    - Chapter 1 – Deep learning
    - <https://youtu.be/aircAruvnKk>
  - **Gradient descent, how neural networks learn**
    - Chapter 2 – Deep learning
    - <https://youtu.be/IHZwWFHWa-w>
  - **What is backpropagation really doing?**
    - Chapter 3 – Deep learning
    - <https://youtu.be/Ilg3gGewQ5U>
  - (Optional) **Backpropagation calculus**
    - Chapter 4 – Deep learning
    - <https://youtu.be/tleHLnjs5U8>



3Blue1Brown ✓

@3blue1brown 5.09M subscribers 129 videos

3Blue1Brown, by Grant Sanderson, is some combi



<https://www.3blue1brown.com/topics/neural-networks>

and some  
issues?

---



# Data

The network **'sees' everything**.  
Has no context for measuring  
how well it does with data it  
has never previously been  
exposed to.

Data

Validation Data

The network **'sees'** a subset of **your data**. You can use **the rest** to **measure its performance** against previously unseen data.



Data

Validation Data

Test Data

The network **'sees' a subset of your data**. You can use an **unseen subset to measure its accuracy while training (validation)**, and then **another subset to measure its accuracy after it's finished training (test)**.

Data

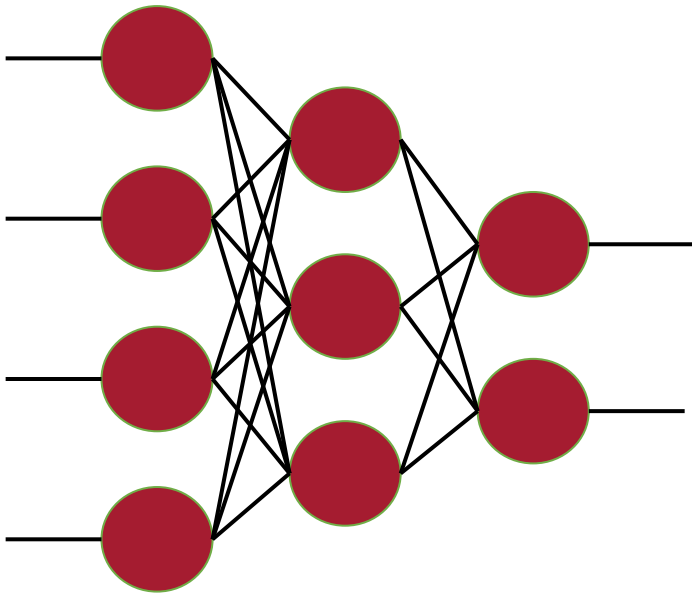
Validation Data

Test Data

Accuracy:  
0.999

Accuracy:  
0.920

Accuracy:  
0.800



Data

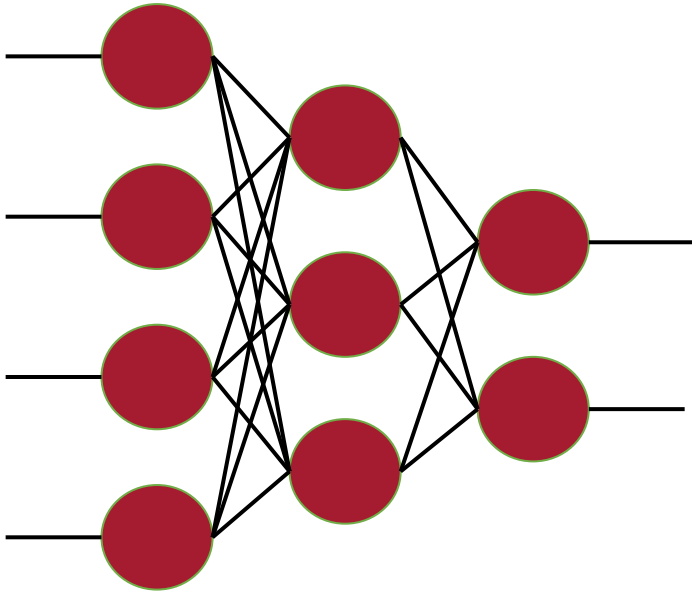
Validation Data

Test Data

Accuracy:  
0.999

Accuracy:  
0.920

Accuracy:  
0.800



Data

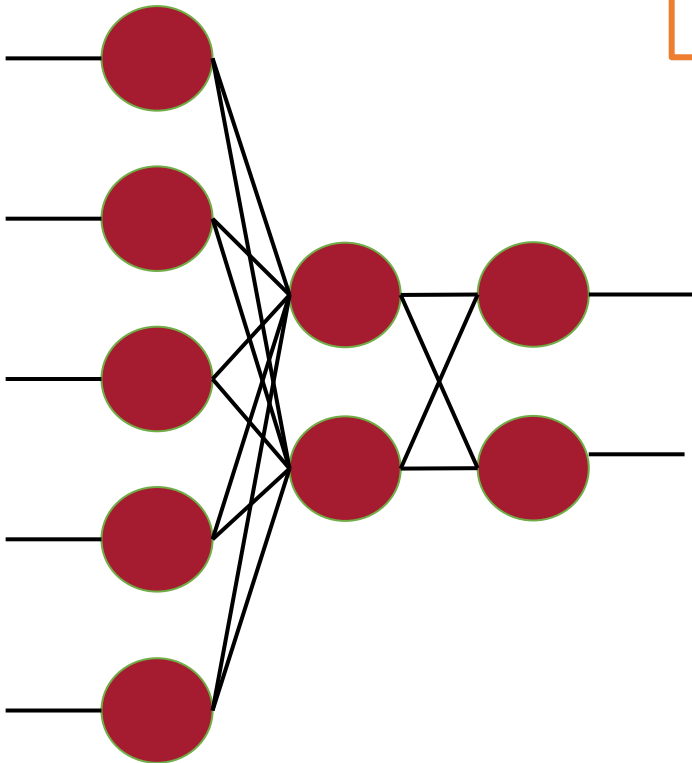
Validation Data

Test Data

Accuracy:  
0.942

Accuracy:  
0.930

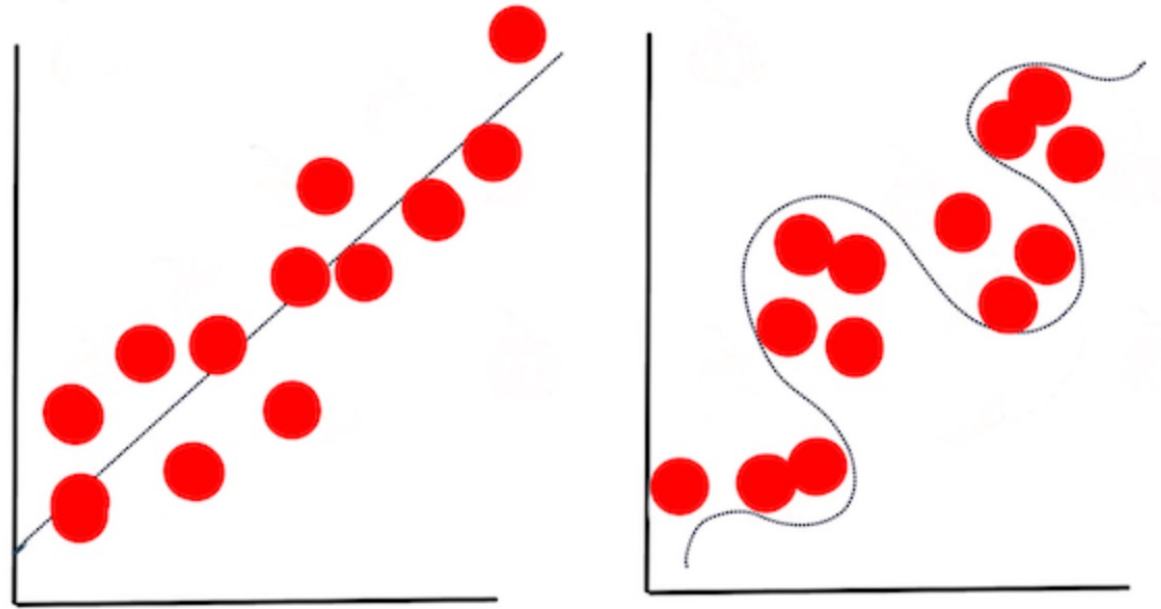
Accuracy:  
0.925



# Correct vs. Overfit Model

**Model fitting** refers to the accuracy of the model's underlying function as it attempts to analyze data with which it is not familiar.

**Underfitting** and **overfitting** are common problems that degrade the quality of the model, as the model fits either not well enough or too well.



Correct vs overfit model

# Prevent Overfitting and Imbalanced Data

Model	Train Accuracy	Test Accuracy
A	99,9%	95%
B	87%	87%
C	99,9%	45%

Test accuracy should be lower than train accuracy, but **how much less accurate?**

**Model A is better than model B** because it has a higher test accuracy, regardless its difference with the train accuracy.

**Model C is a clear case of overfitting** as the train accuracy is very high but the test accuracy isn't anywhere near as high.

This **distinction is subjective**, but comes from knowledge of your problem and data, and **what magnitudes of error are acceptable.**



**EDGE  
IMPULSE**

T I N Y



• edu

I would like to thank:

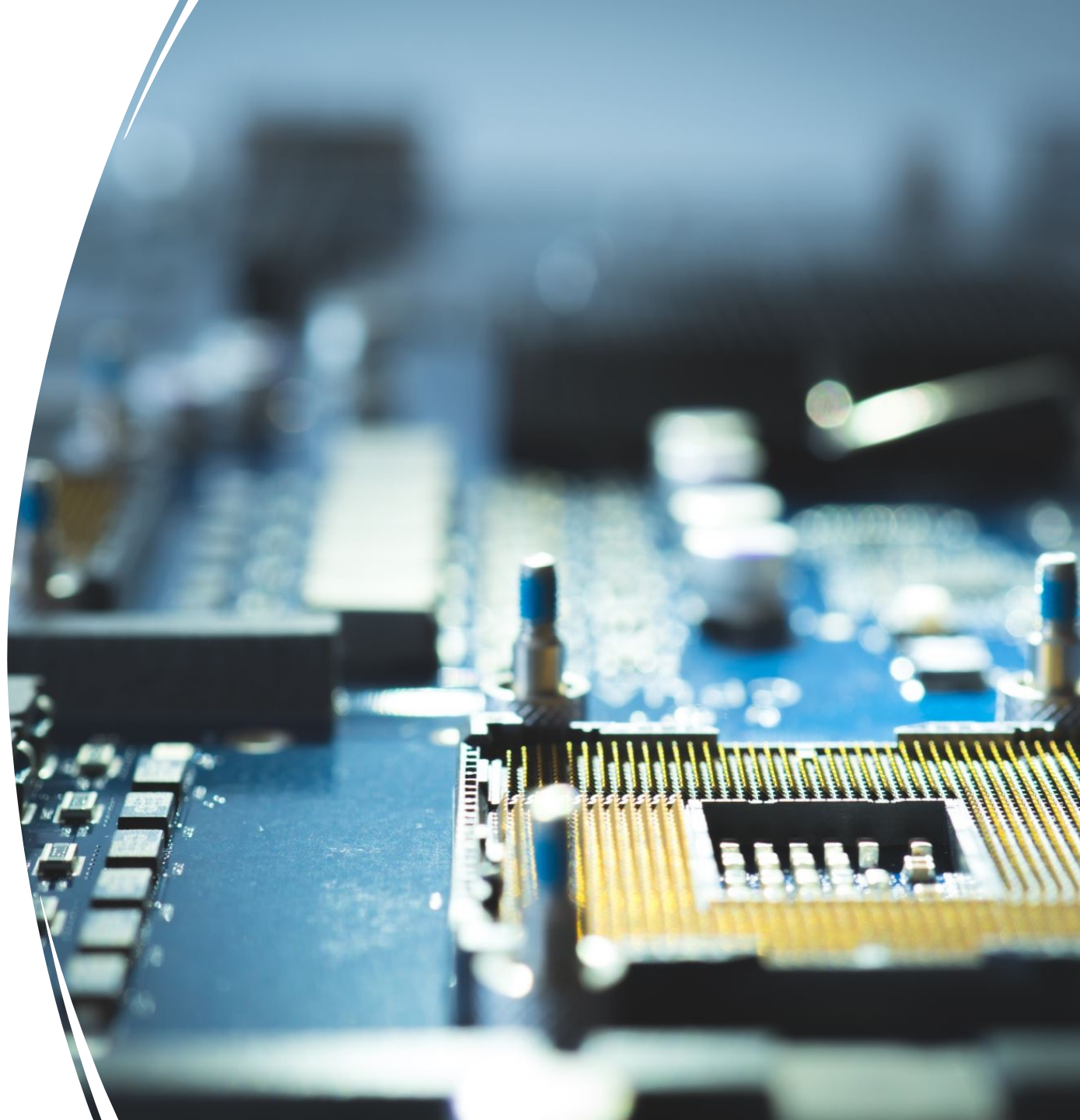
[Shawn Hymel](#) and **Edge Impulse**,

[Pete Warden](#) and [Laurence Moroney](#) from **Google**,

[Prof. Vijay Janapa Reddi](#) and [Brian Plancher](#) from **Harvard** and **Columbia University**,

[Marcelo Rovai](#) from **UNIFEI**,

and the rest of the [TinyML](#) team for preparing the excellent material on TinyML that was the main input for this presentation.



# Gracias!

Prof. Diego Méndez Chaves, Ph.D

Associate Professor - Electronics Engineering Department  
Director of the Master Program in Internet of Things  
Director of the Master Program in Electronics Engineering  
email: [diego-mendez@javeriana.edu.co](mailto:diego-mendez@javeriana.edu.co)

