Building Deep Learning Models Using PyTorch

INTRODUCTION TO PYTORCH

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

PyTorch is a deep learning framework

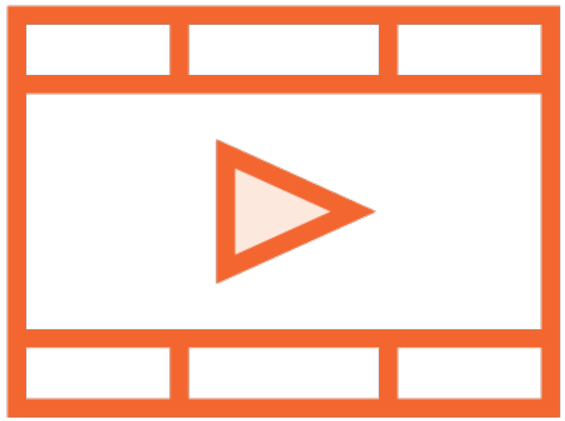
More tightly integrated with Python than TensorFlow

Can use Python libraries, debugger

Unlike TensorFlow, supports dynamic computation graphs

Like other deep learning frameworks uses a forward and backward pass for training

Prerequisites and Course Outline



Prereqs - Basic ML

Building Machine Learning Models in Python with scikit-learn

- Using the scikit-learn library for ML

How to Think About Machine Learning Algorithms

- Introduction to machine learning

Software and Skills

Be very comfortable programming in Python 3 using Jupyter notebooks

Be comfortable with ML concepts

Understand the basics of neural networks for deep learning

Course Outline

Introducing PyTorch

- Machine learning using NNs
- Deep learning framework, alternative to TensorFlow

Building Simple Neural Networks

- Autograd in PyTorch for back-propagation
- Defining neural networks and connecting them

Building Image Classification Models

- Convolutional Neural Networks in PyTorch
- Using pre-trained models for image classification

Building Text Classification Models

- Recurrent Neural Networks in PyTorch

Introducing Neural Networks

Reviews: Positive or Negative?

ML-based Classifier

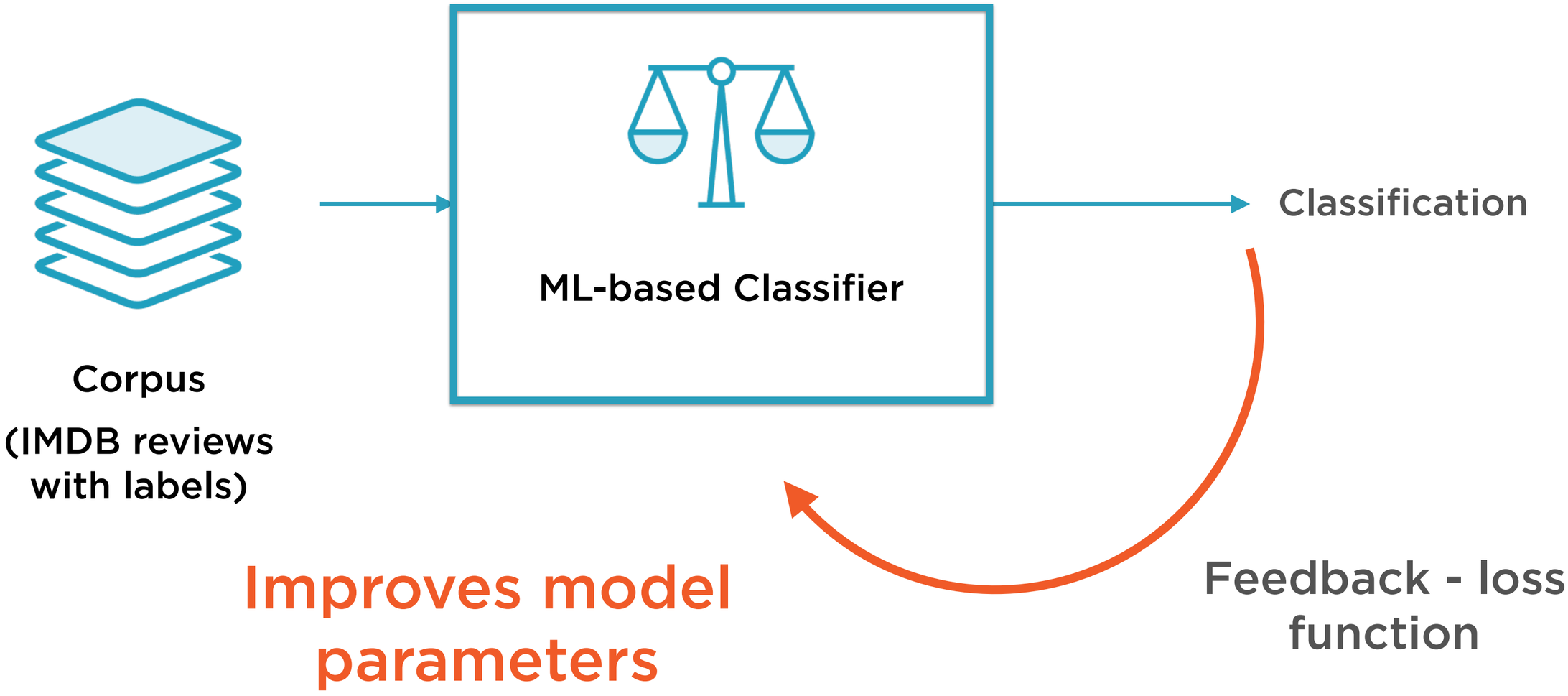
Training

Feed in a large corpus of data classified correctly

Prediction

Use it to classify new instances which it has not seen before

Training the ML-based Classifier



“Traditional” ML-based systems rely on experts to decide what features to pay attention to

“Representation” ML-based systems figure out by themselves what features to pay attention to

Neural networks are examples of such systems

What is a Neural Network?

Deep Learning

Algorithms that learn what features matter

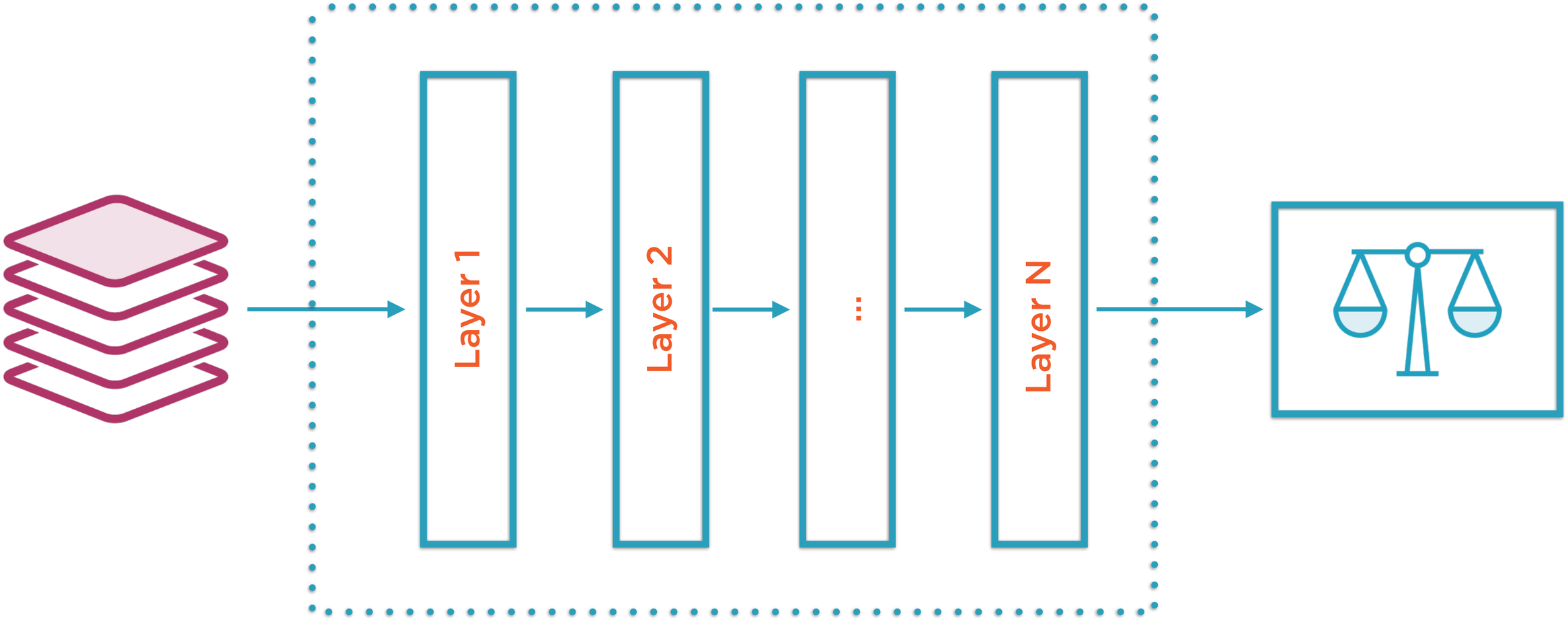
Neural Networks

The most common class of deep learning algorithms

Neurons

Simple building blocks that actually “learn”

Neural Networks

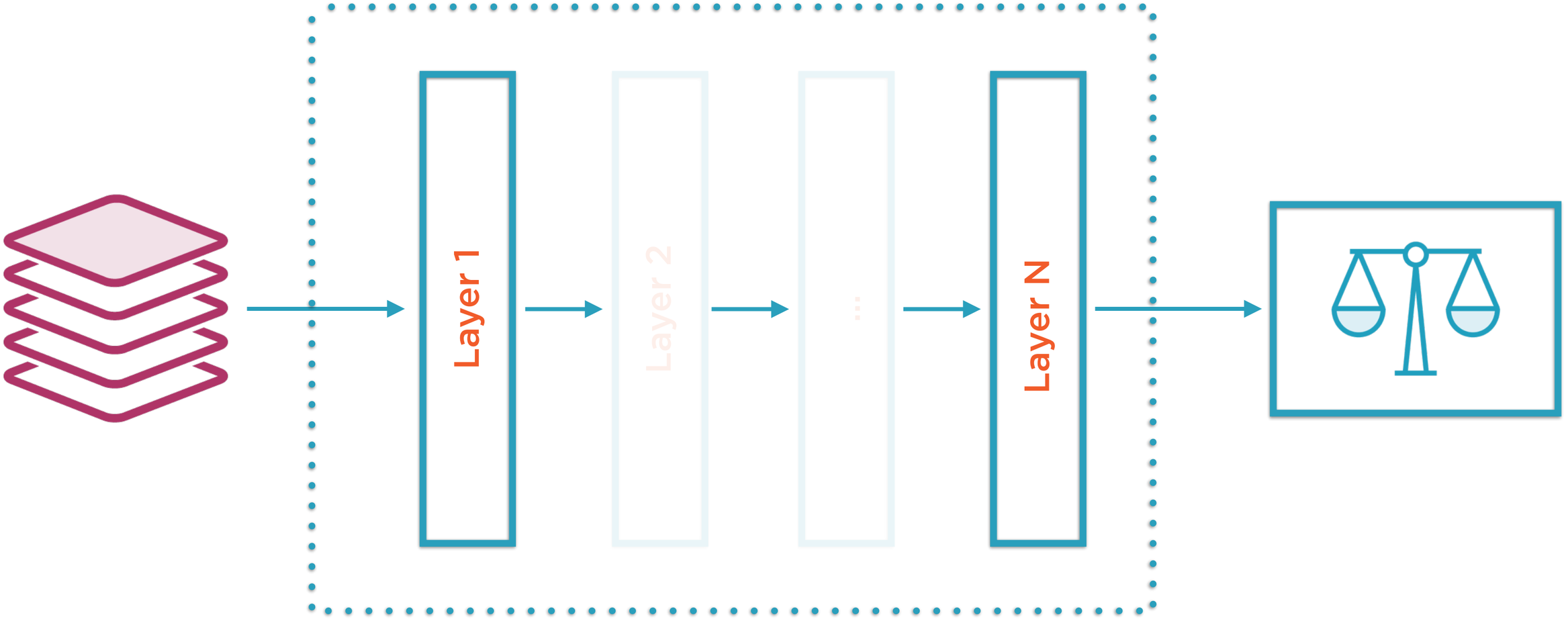


Corpus

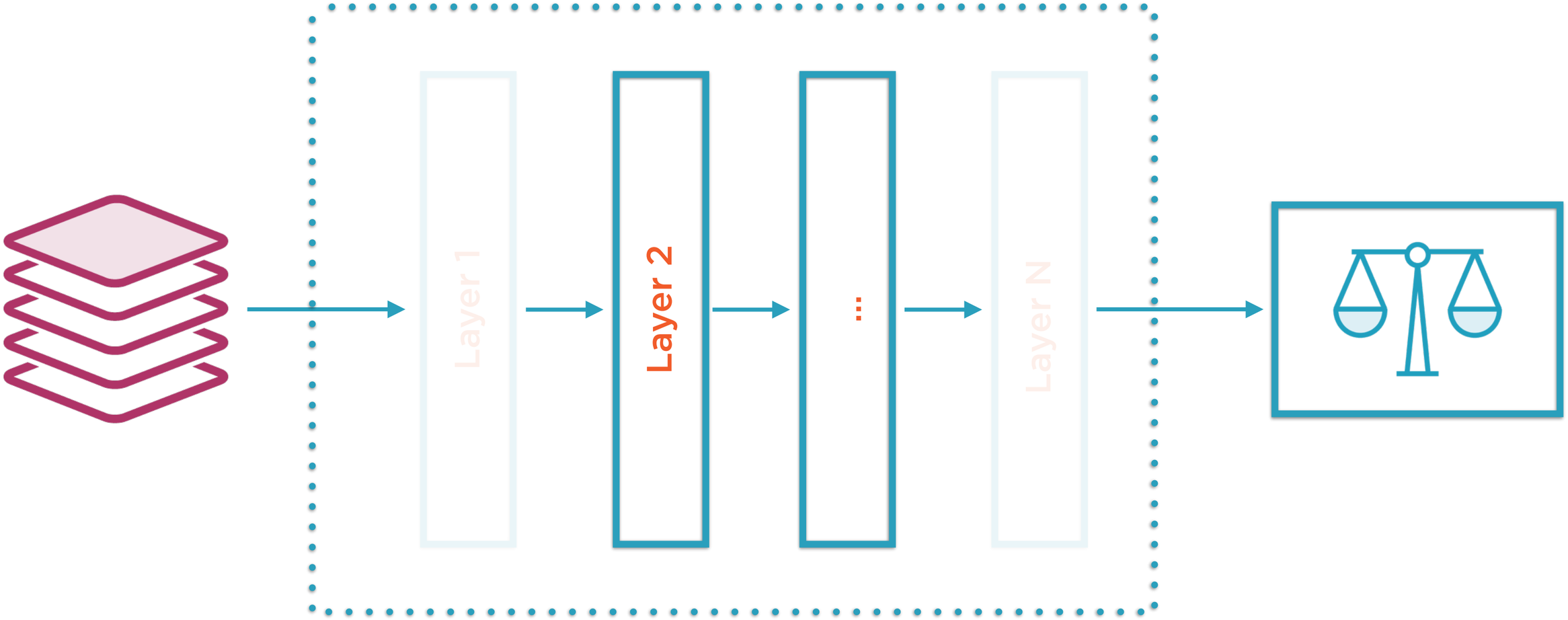
Layers in a neural network

ML-based Classifier

Neural Networks



Neural Networks

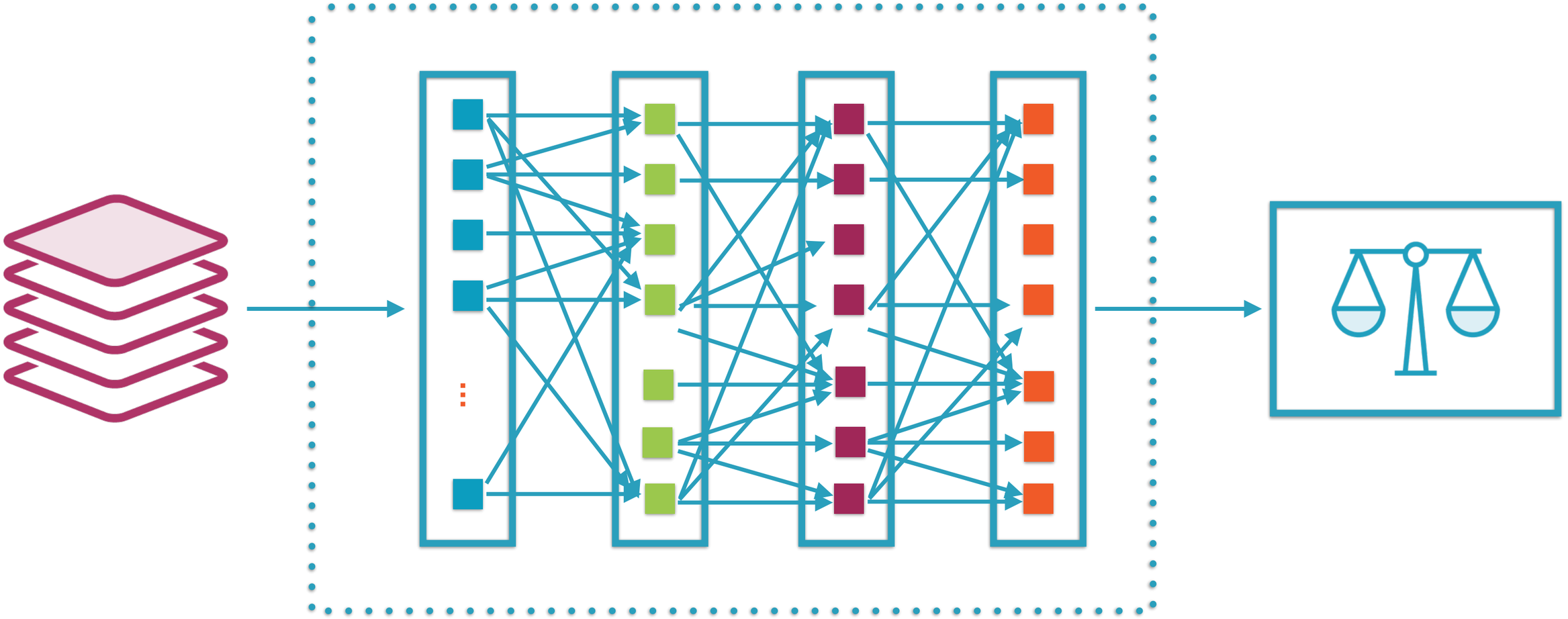


Corpus

ML-based Classifier

Hidden layers

Neural Networks

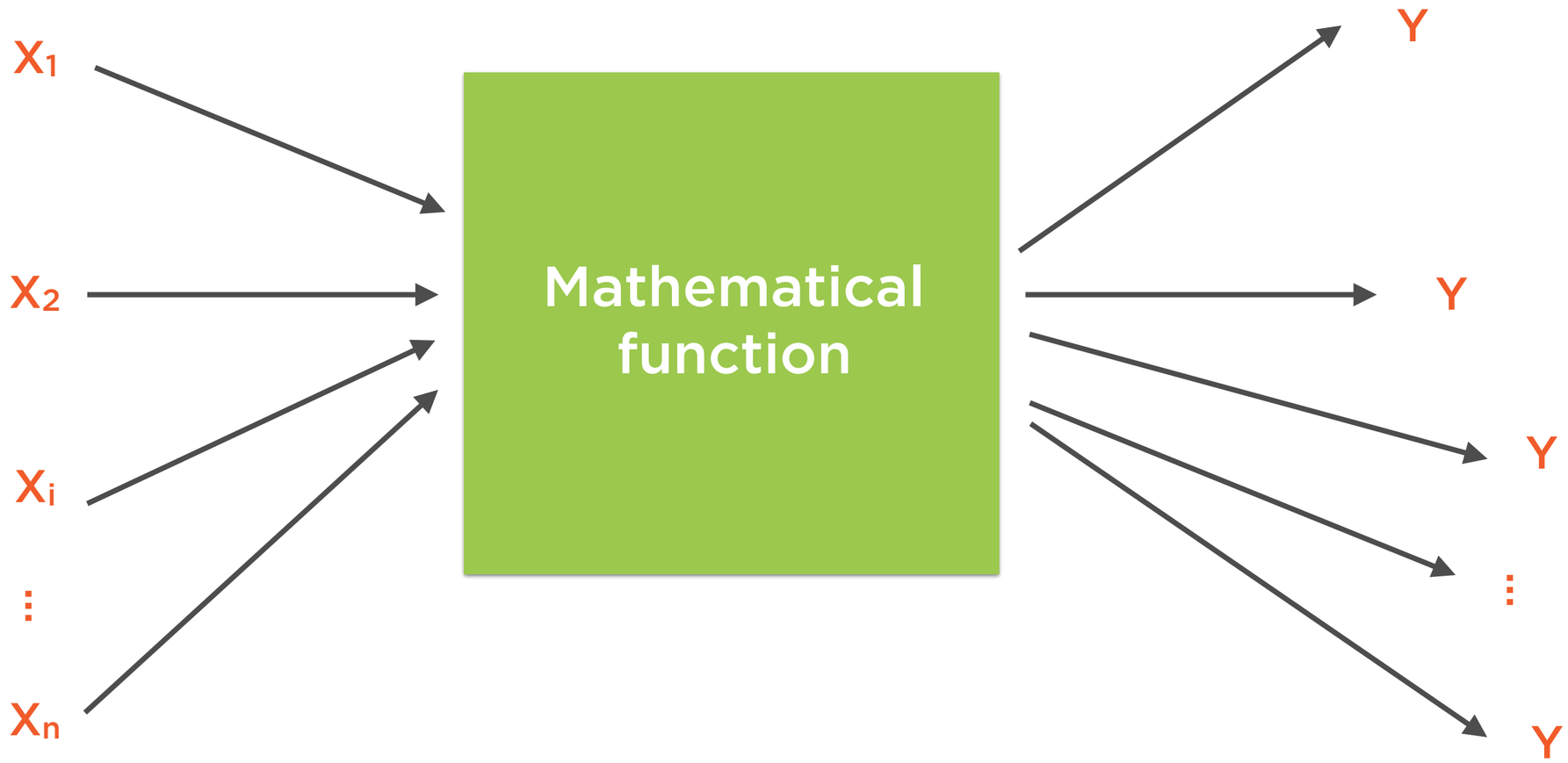


Corpus

Each layer consists of individual interconnected neurons

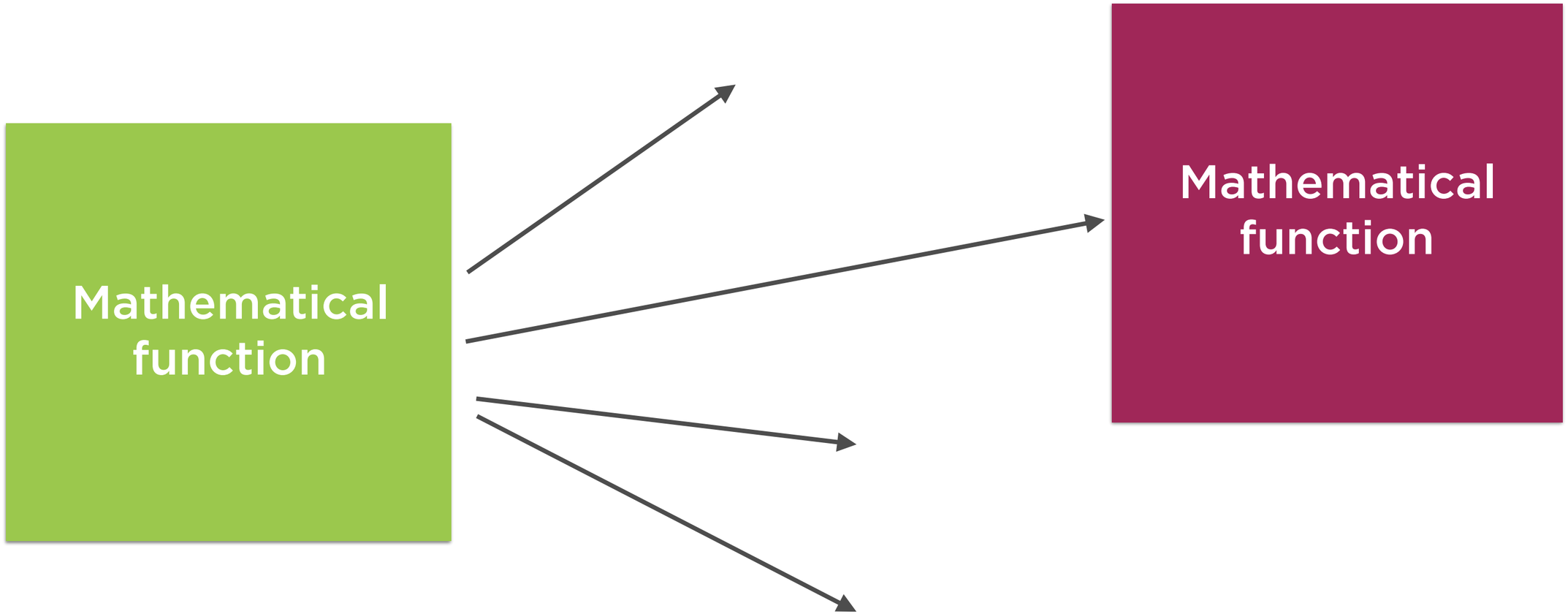
ML-based Classifier

Operation of a Single Neuron



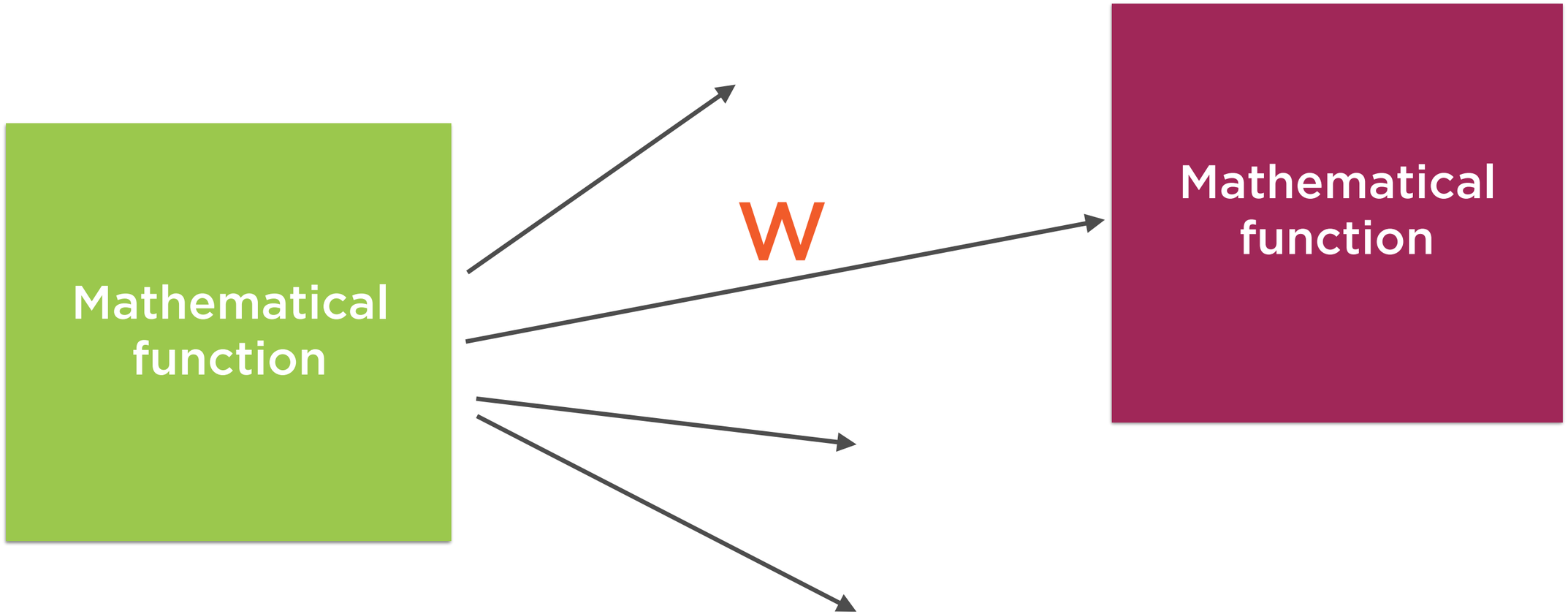
For an active neuron a change in inputs should trigger a corresponding change in the outputs

Operation of a Single Neuron



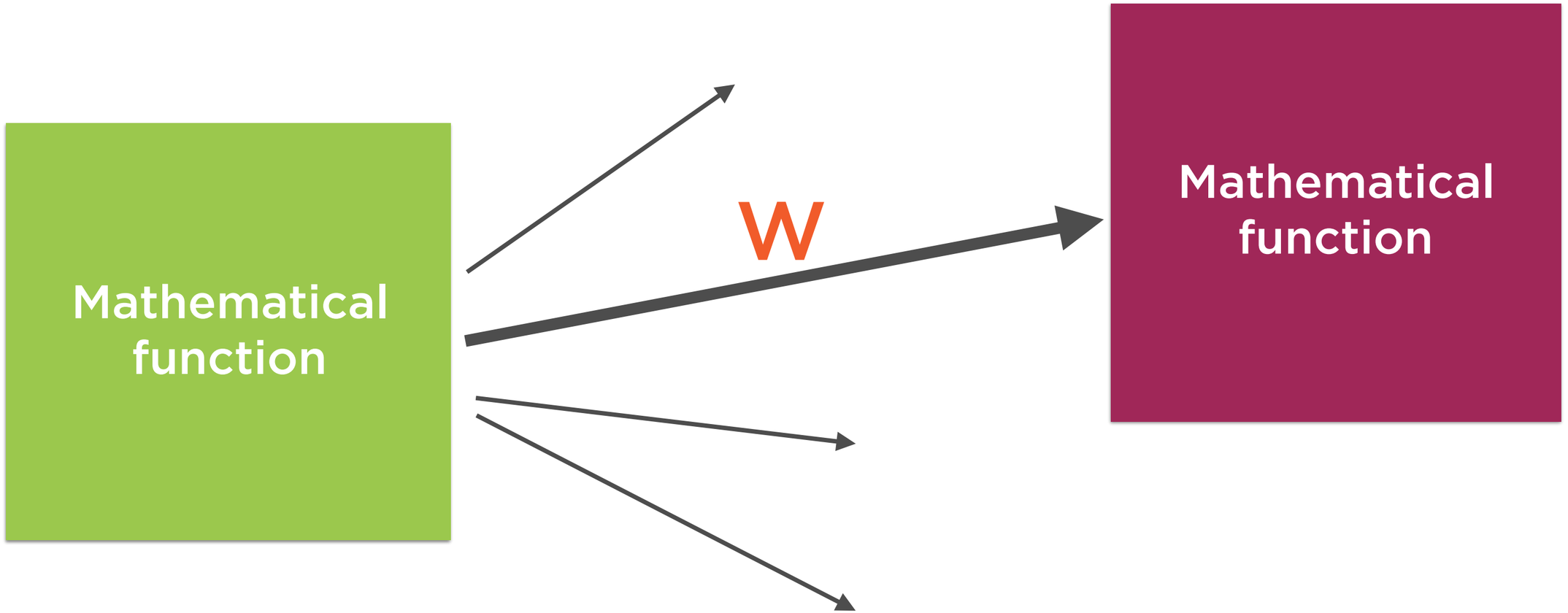
The outputs of neurons feed into the neurons from the next layer

Operation of a Single Neuron



Each connection is associated with a weight

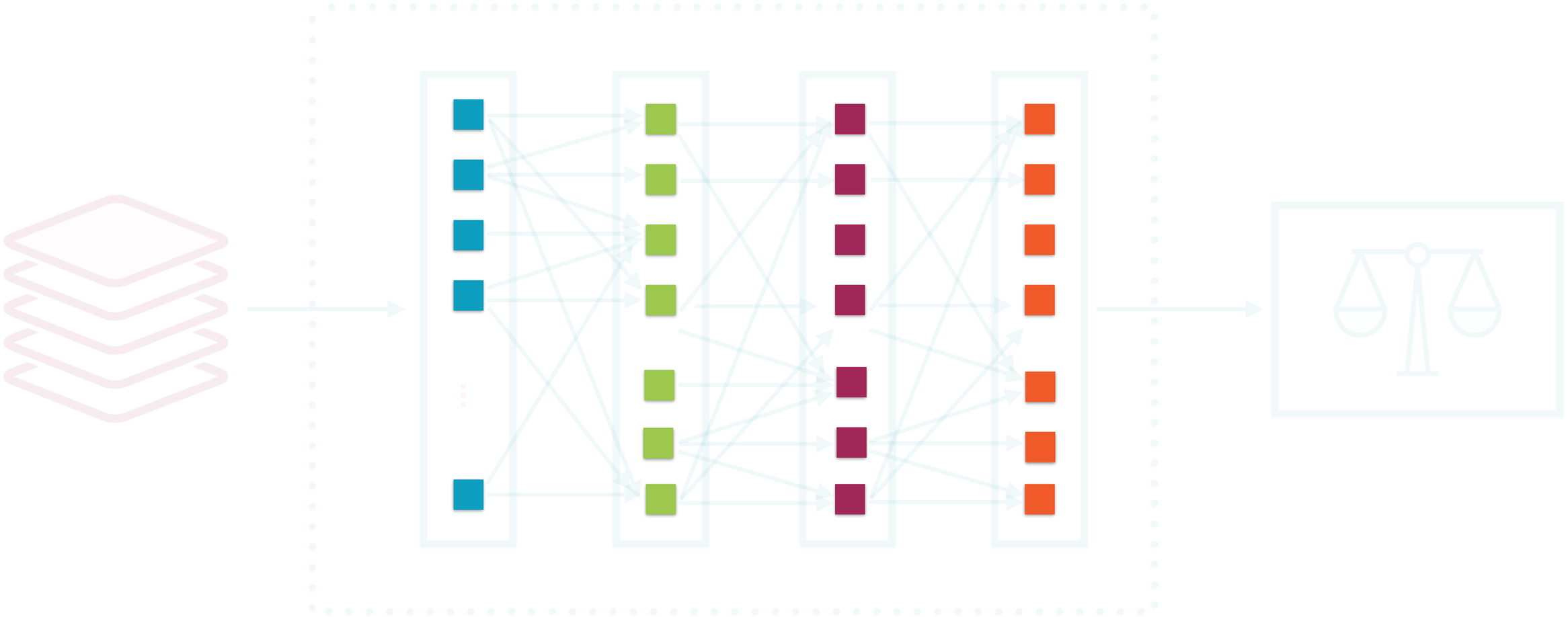
Operation of a Single Neuron



If the second neuron is sensitive to the output of the first neuron, the connection between them gets stronger

W increases

The Computational Graph

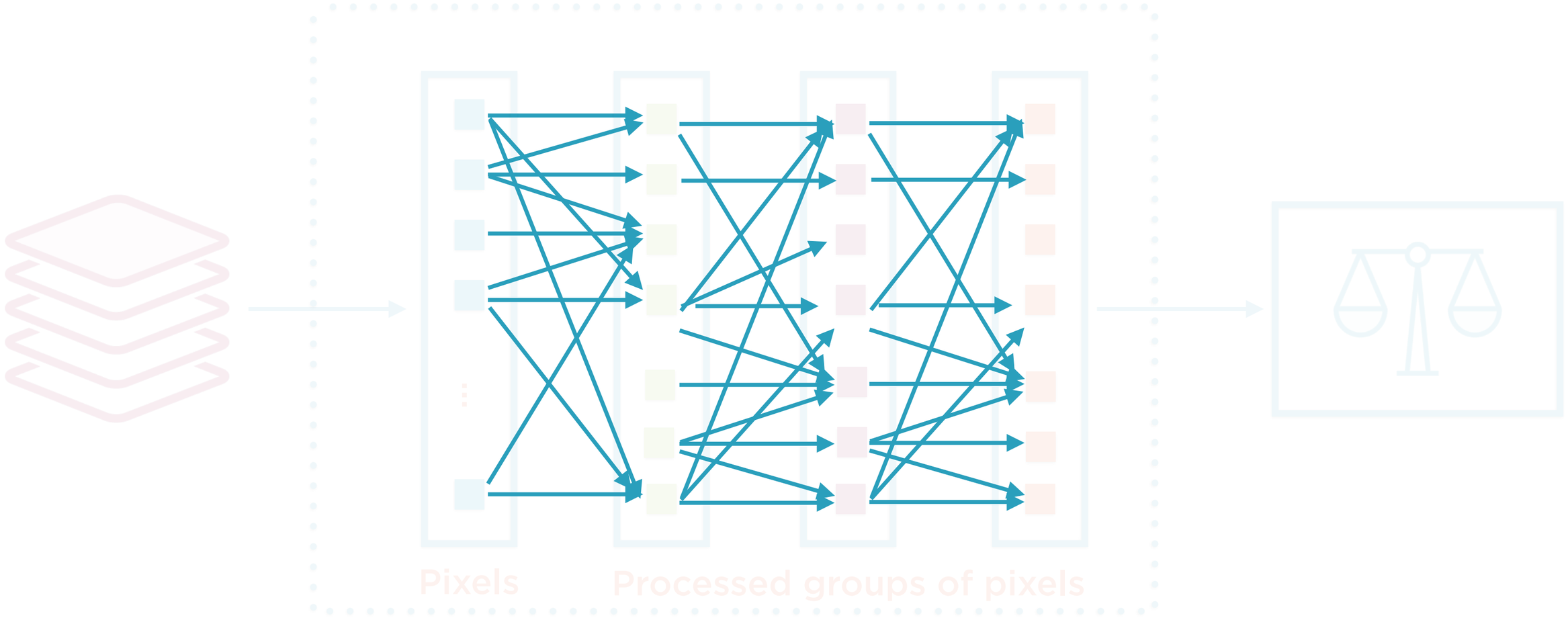


Corpus

ML-based Classifier

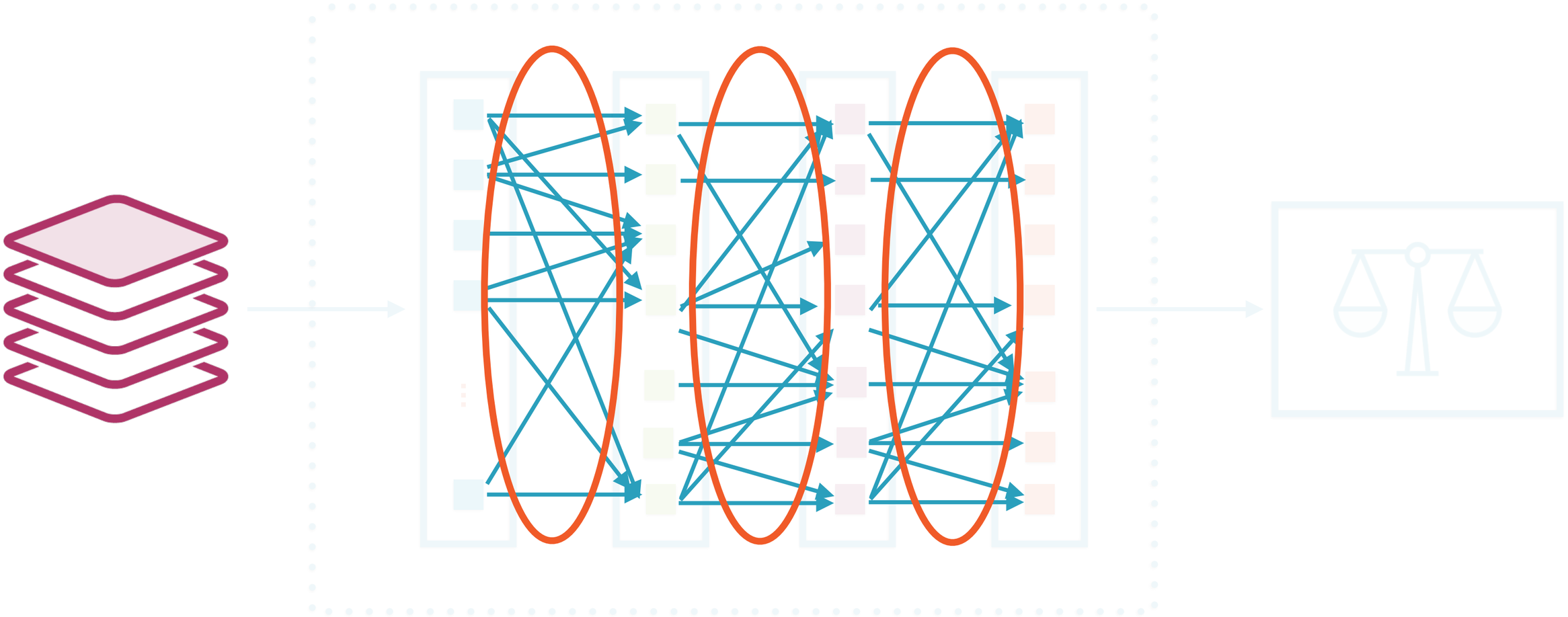
The nodes in the computation graph are neurons (simple building blocks)

The Computational Graph



The edges in the computation graph are data called tensors

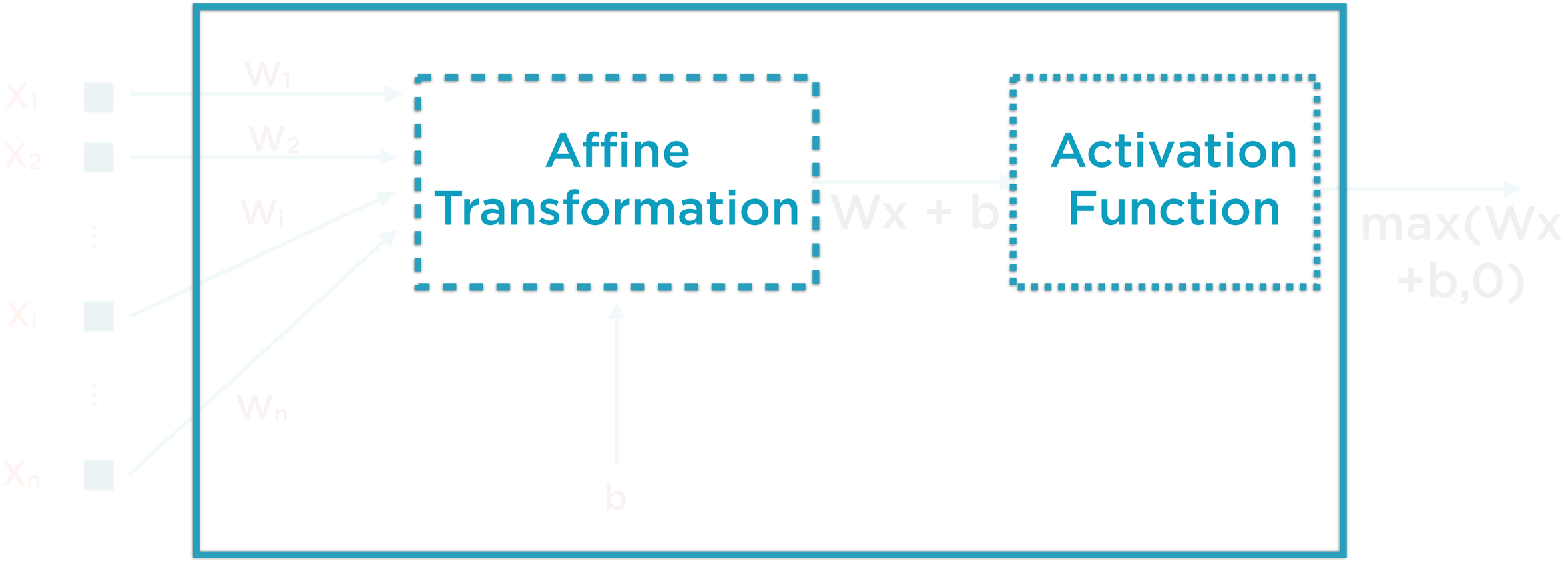
A Neural Network



Corpus

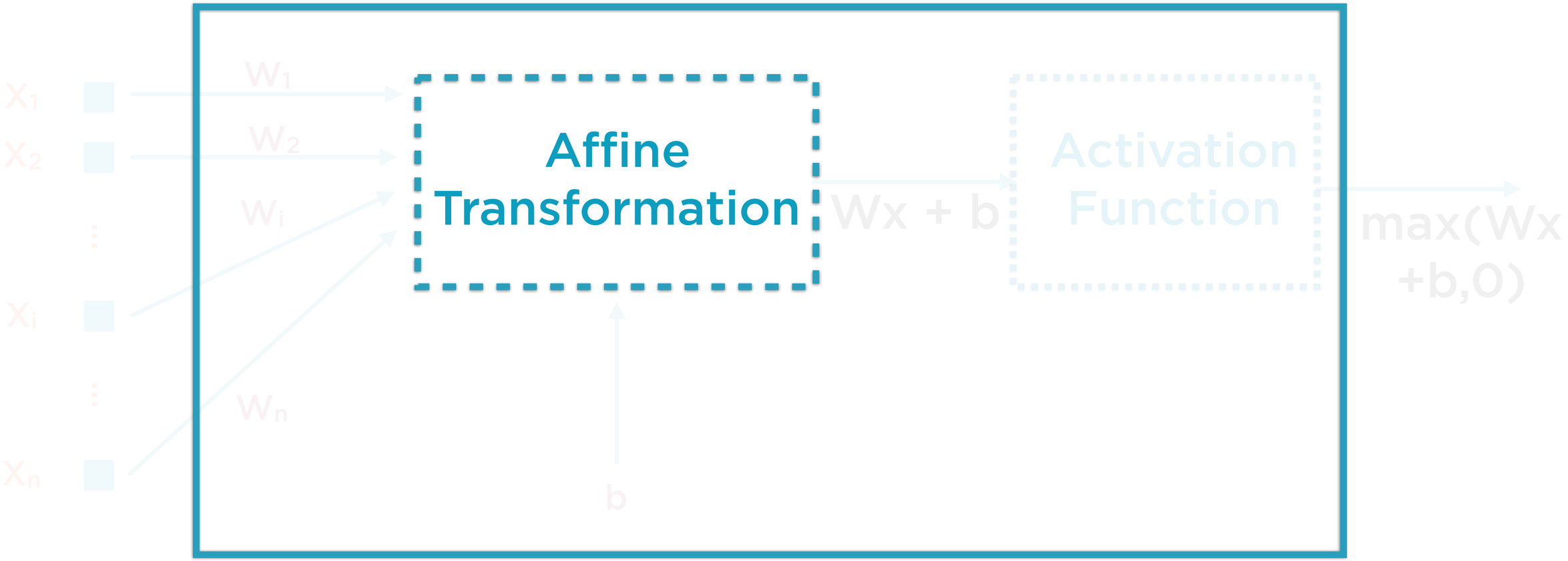
Once a neural network is trained, all edges have weights which help it make predictions

Operation of a Single Neuron



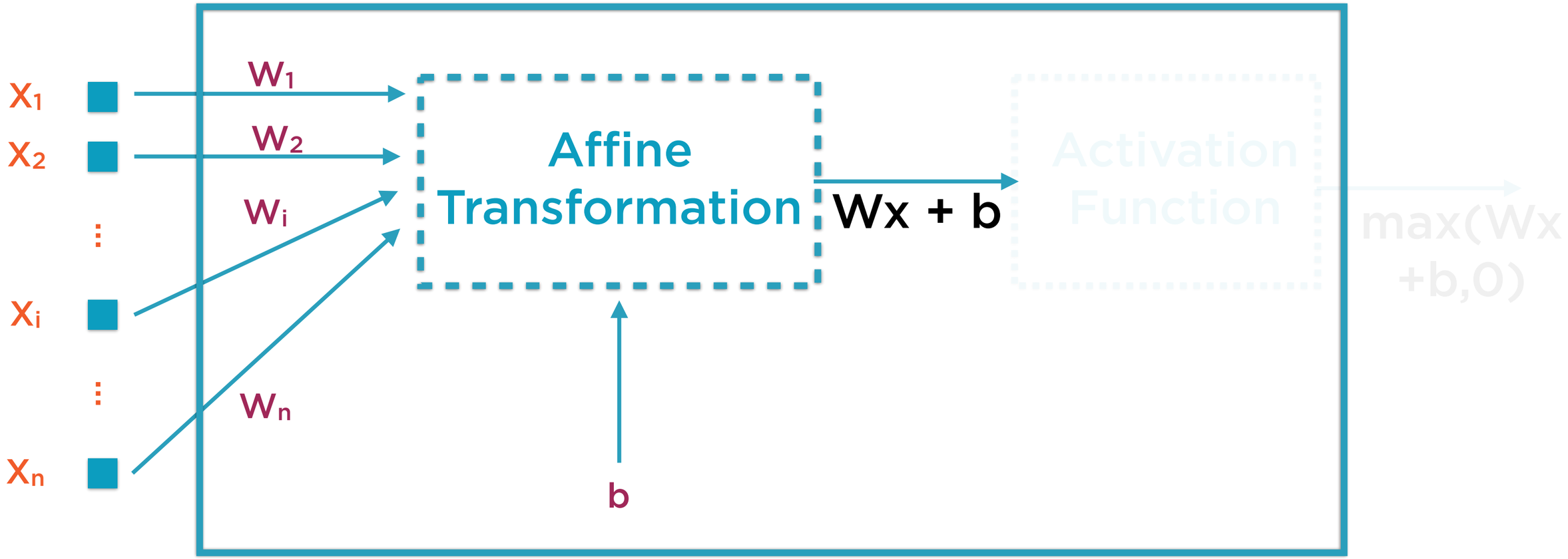
Each neuron only applies two simple functions to its inputs

Operation of a Single Neuron



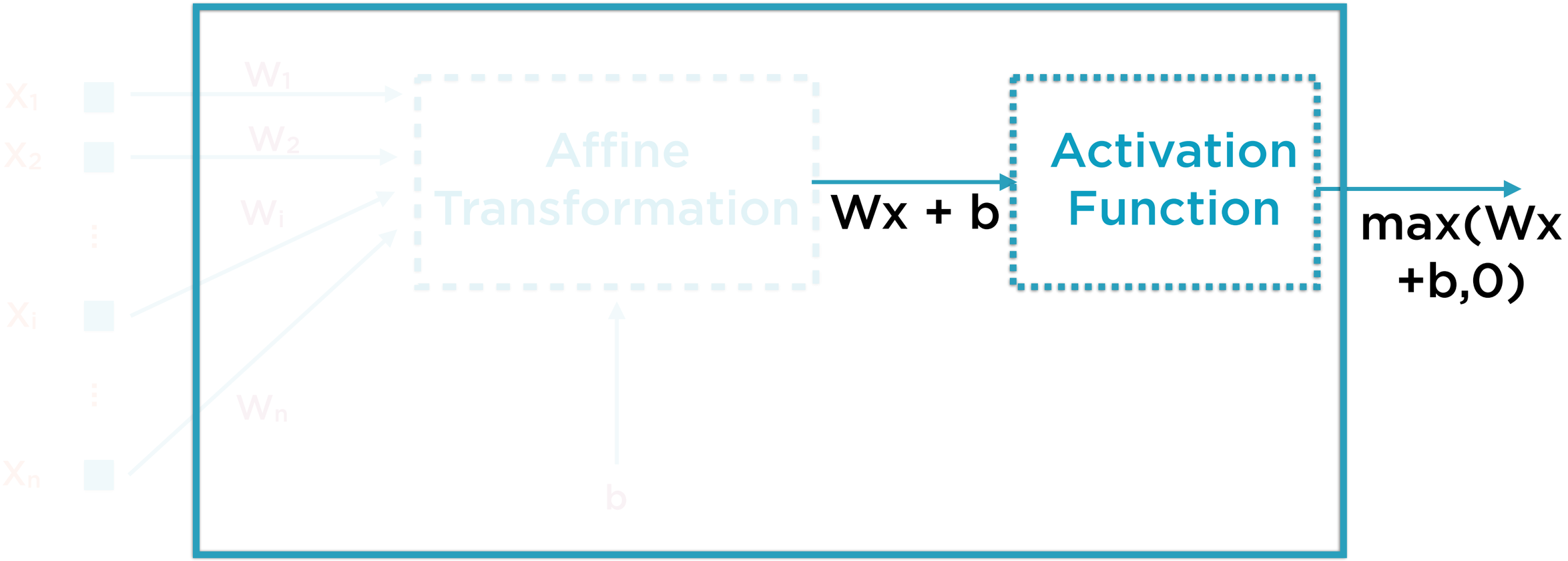
The affine transformation alone can only learn linear relationships between the inputs and the output

Operation of a Single Neuron



The affine transformation is just a weighted sum with a bias added: $W_1x_1 + W_2x_2 + \ldots + W_nx_n + b$

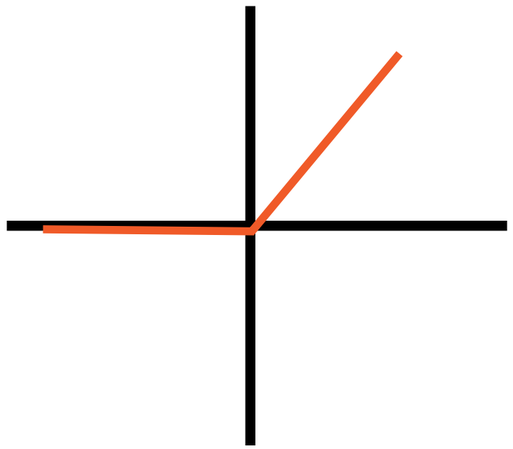
Operation of a Single Neuron



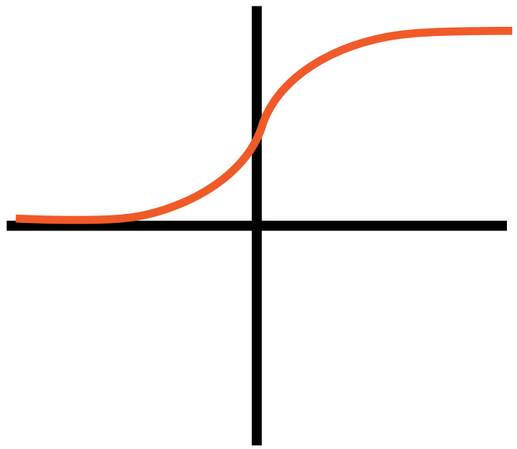
The combination of the affine transformation and the activation function can learn any arbitrary relationship

Common Activation Functions

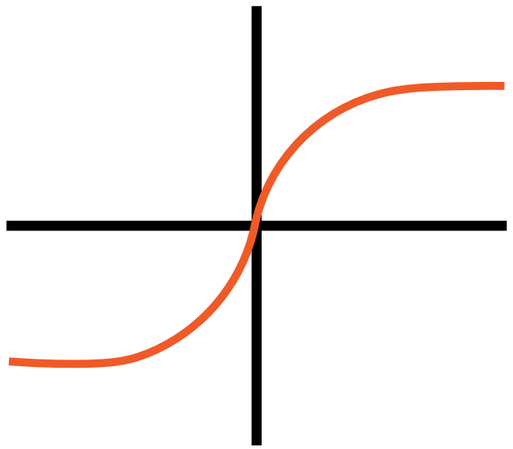
ReLU



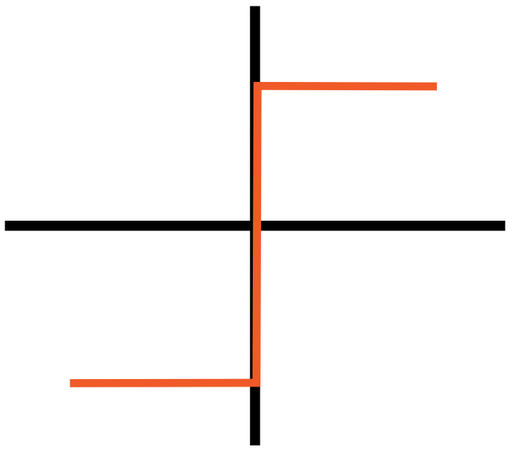
logit



tanh

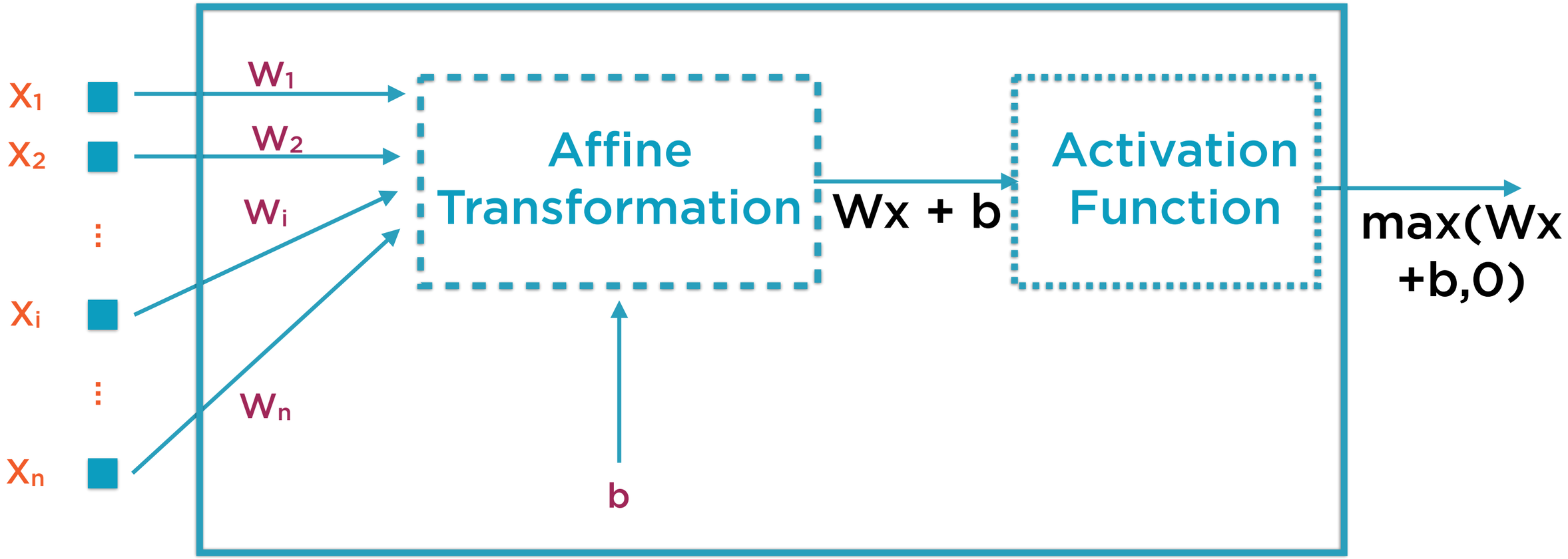


step



Notice how activations functions have a gradient, this gradient allows them to be sensitive to input changes

Neuron as a Learning Unit



Many of these simple neurons arranged in layers can do magical stuff

The **weights** and **biases** of individual neurons are determined during the **training** process

Introducing PyTorch

PyTorch

A deep learning framework for fast, flexible experimentation.

https://pytorch.org/

PyTorch Use-cases

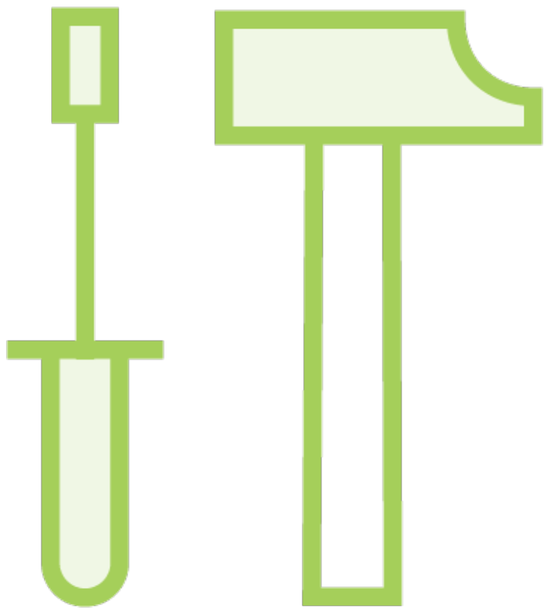
Two primary uses

As alternative to TensorFlow

- Deep learning framework

As alternative to NumPy

- Tensor computations but with “strong GPU acceleration”

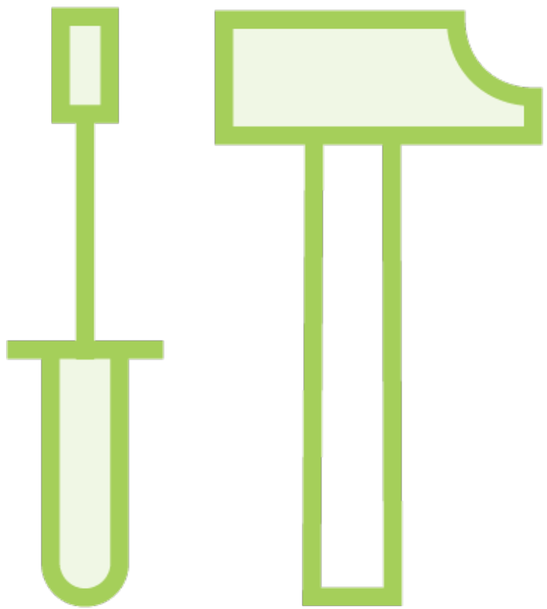


Deep Learning Framework

“Tape-based Autograd”

Neural network can be redefined dynamically

Different from TensorFlow, CNTK

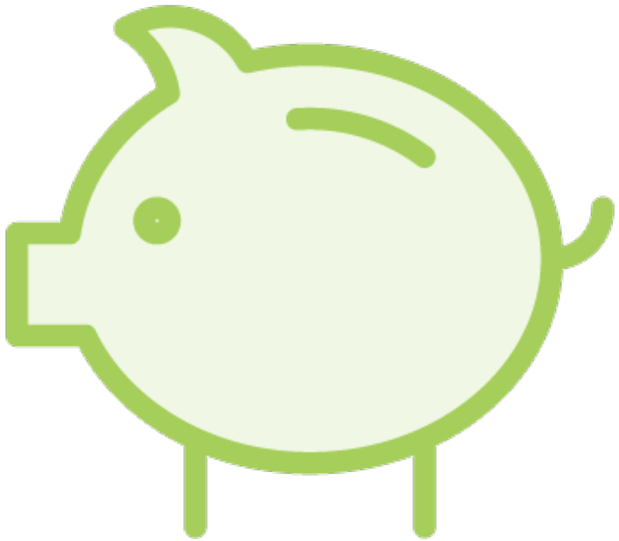


GPU-ready Tensor Library

Tensors for either CPU or GPU

Powerful, fast NumPy-like functionality

- slicing
- indexing
- reductions
- linear algebra



Tight Python Integration

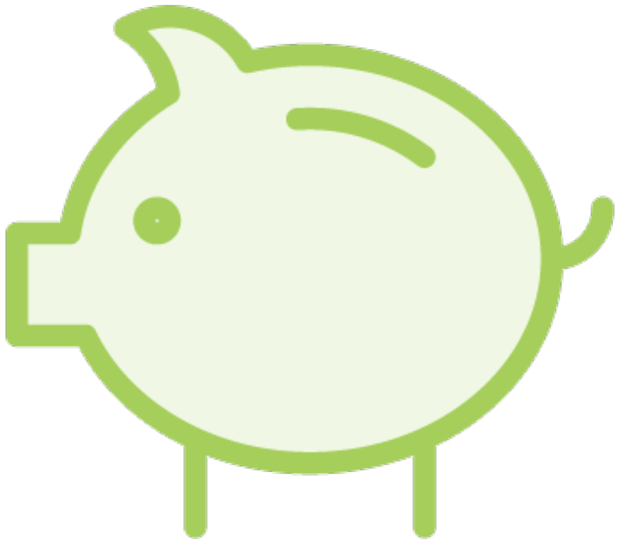
Deeply tied to Python

Approach similar to numpy/scikit-learn

Create neural networks in Python

Use existing Python libraries...

...and debuggers

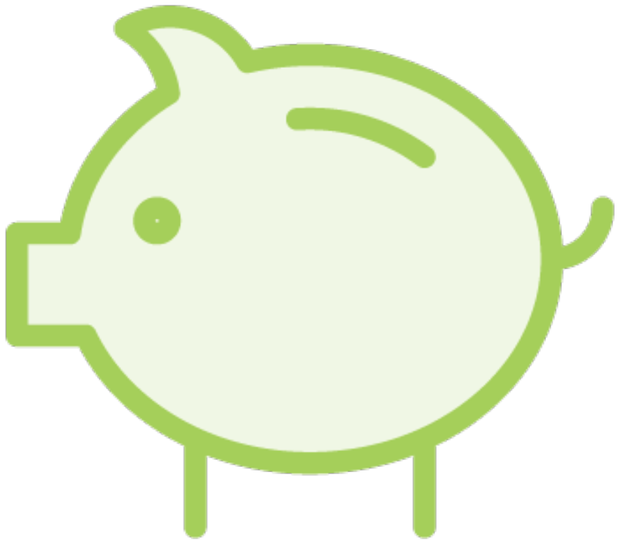


Imperative Execution

Write code, run immediately

No separate build and run phases

Debugging easier



Ease of Extensibility

Easily write new neural network layers

Different alternatives

- Torch API
- Scipy
- C/C++ extension API

PyTorch Background

Relatively new

- Initial release October 2016
- Stable release April 2018

Based on Torch

- Open-source ML library (since 2002!)

Facebook connection

- Developed by Facebook AI researchers

TensorFlow vs. PyTorch

TensorFlow	PyTorch
Computation graph is static...	Computation graph is dynamic...
...must be defined before being run	...can be defined and run as you go
tf.Session for separation from Python	Tightly integrated with Python
Debugging via tfdbg	Debugging with PyCharm, pdb
Visualization using TensorBoard	Visualization using matplotlib, seaborn
Deployment using TF Serving	Need to set up REST API e.g. Flask
tf.device and tf.DeviceSpec to use GPUs (relatively hard)	torch.nn.DataParallel to use GPUs (relatively easy)

Demo

Installing PyTorch on your local machine

Tensors in PyTorch

Tensors in PyTorch: Conceptually identical to tensors in TensorFlow

Tensor

The central unit of data in TensorFlow. A tensor consists of a set of primitive values shaped into an array of any number of dimensions.

https://www.tensorflow.org/

Tensor

The central unit of data in TensorFlow. A tensor consists of a set of primitive values shaped into an array of any number of dimensions.

https://www.tensorflow.org/

Tensor

The central unit of data in TensorFlow. A tensor consists of a set of primitive values shaped into an array of any number of dimensions.

https://www.tensorflow.org/

Data Is Represented as Tensors

Scalars are 0-D tensors

3, 6.7, “a”

Data Is Represented as Tensors

Vectors are **1-D** tensors

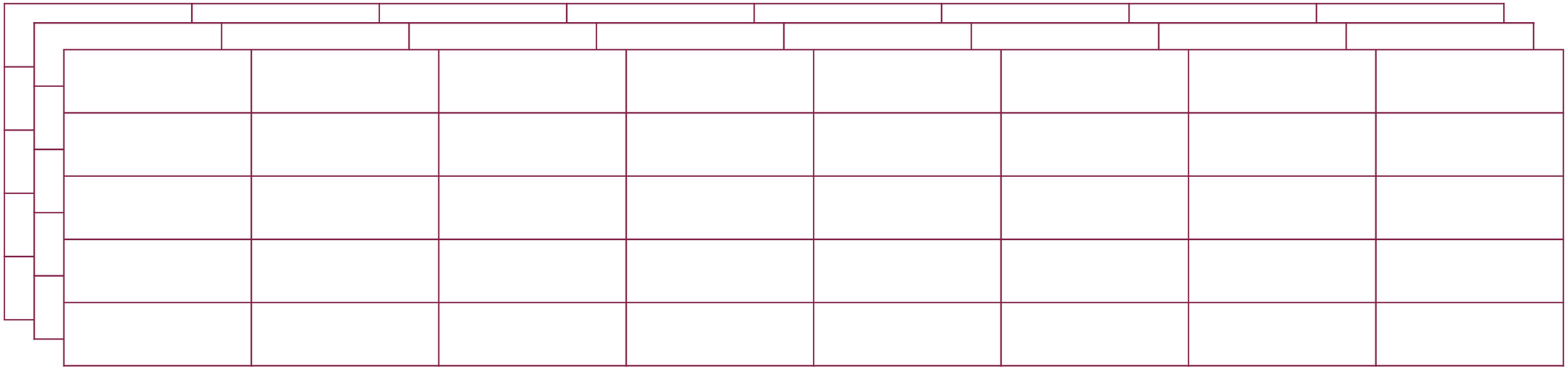
[1, 3, 5, 7, 9]

Data Is Represented as Tensors

Matrices are 2-D tensors

```
[[ ], 3, 5],
 [7, 9, 1[]]
```

Data Is Represented as Tensors



N-Dimensional matrices are **N-D** tensors

```
[[[1, 2], [3, 4], [5, 6]],
 [[7, 8], [9, 10], [11, 12]]]
```

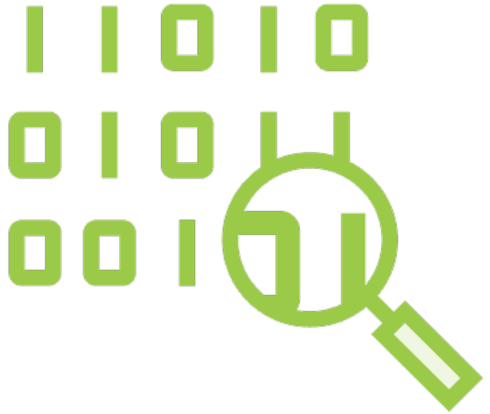
Characterization of Tensors

Rank

The number of dimensions in a tensor

Shape

The number of elements in each dimension



Data Type

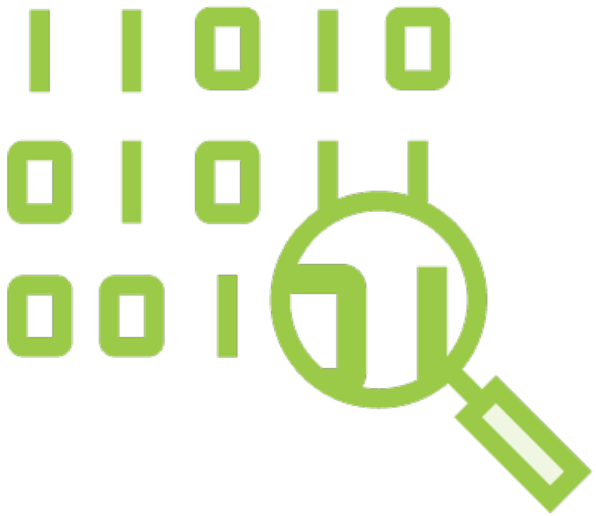
The data type of each element in the tensor

Rank

Tensor	Rank
4	0
[1, 2, 3]	1
[[1, 2], [3, 4]]	2
[[[1], [2]], [[3], [4]]]	3

Shape

Tensor	Shape
4	[]
[1, 2, 3]	[3]
[[1, 2, 3], [4, 5, 6]]	[3, 2]
[[[1], [2]], [[3], [4]]]	[2, 2, 1]



Data Type

int
float
string
boolean

Rank, shape and data types are
3 important characteristics
which define a Tensor

Demo

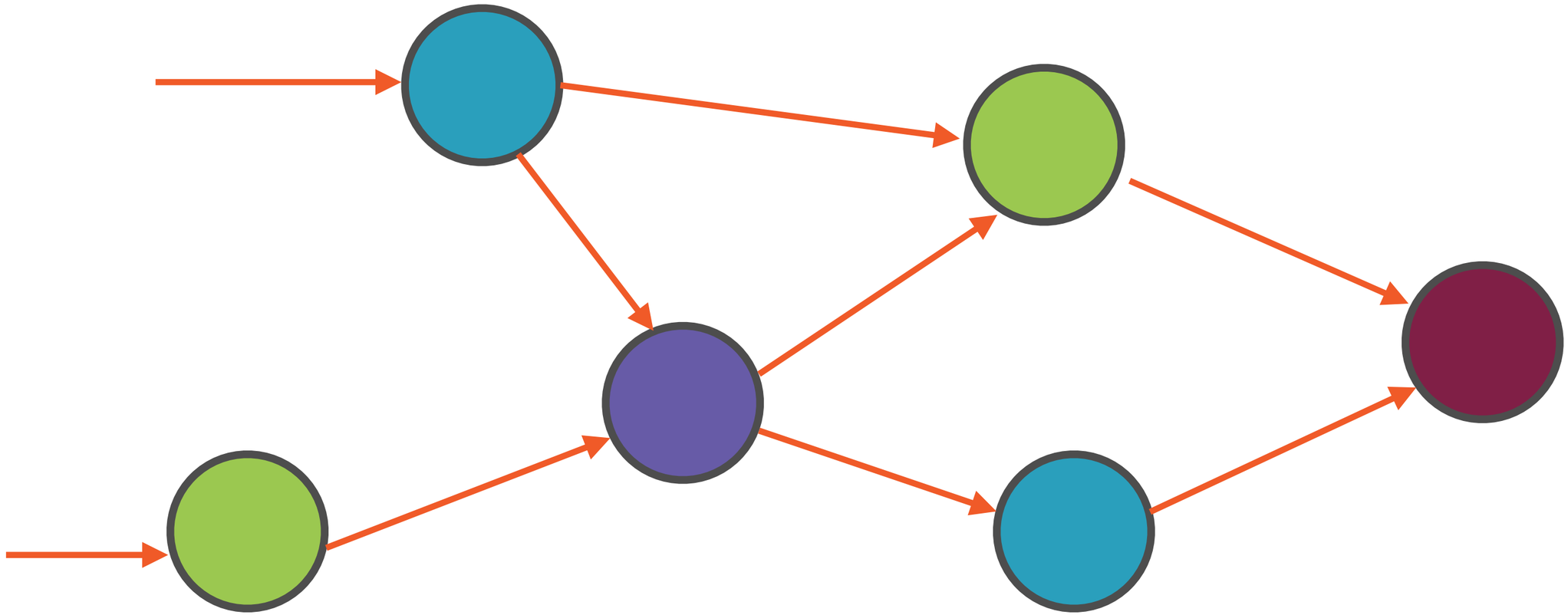
Tensors in PyTorch

Operations with Tensors

Computation Graphs in PyTorch

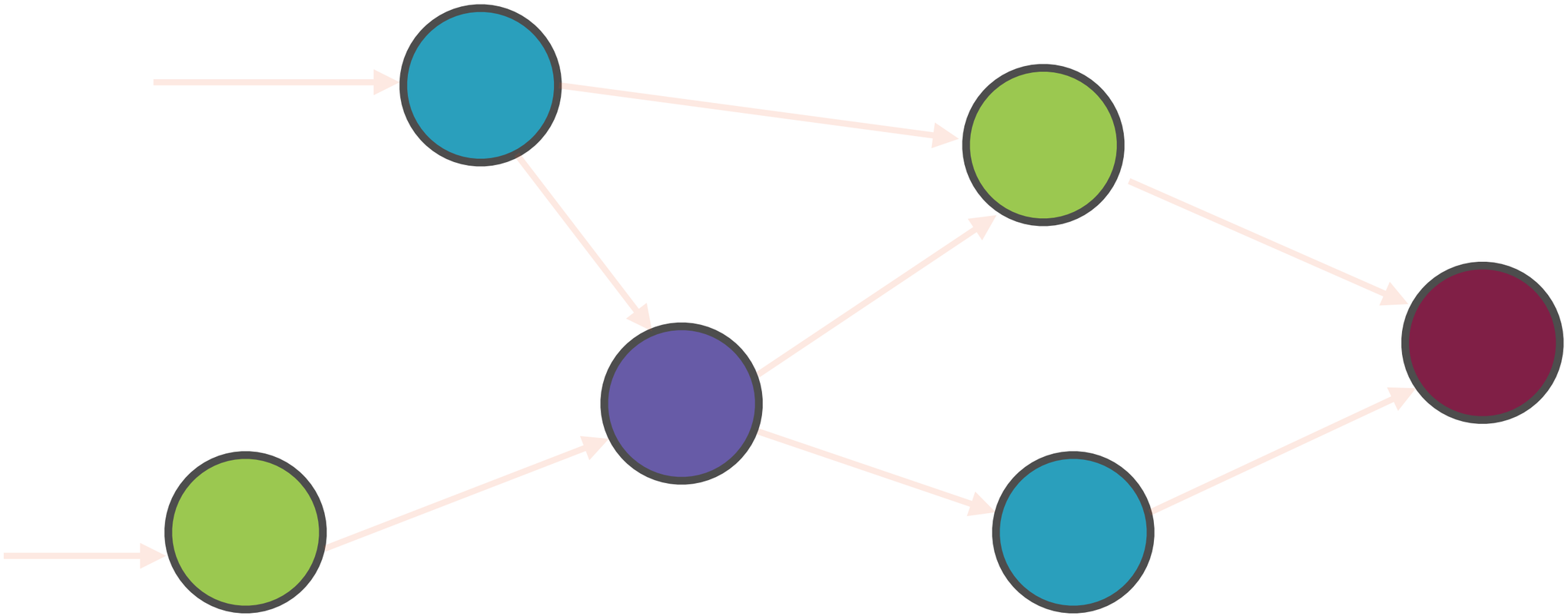
Similar to TF graphs but with one important difference: PyTorch computation graphs are **dynamic**

Everything Is a Graph



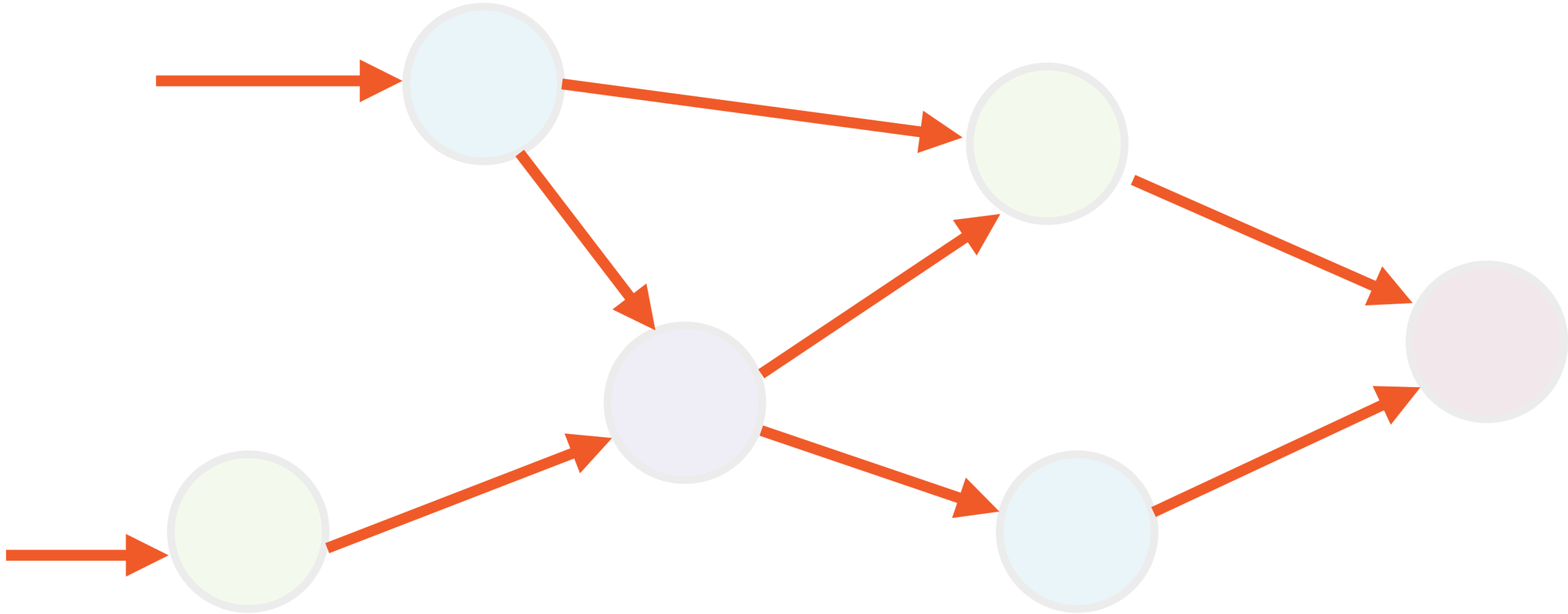
A network

Everything Is a Graph



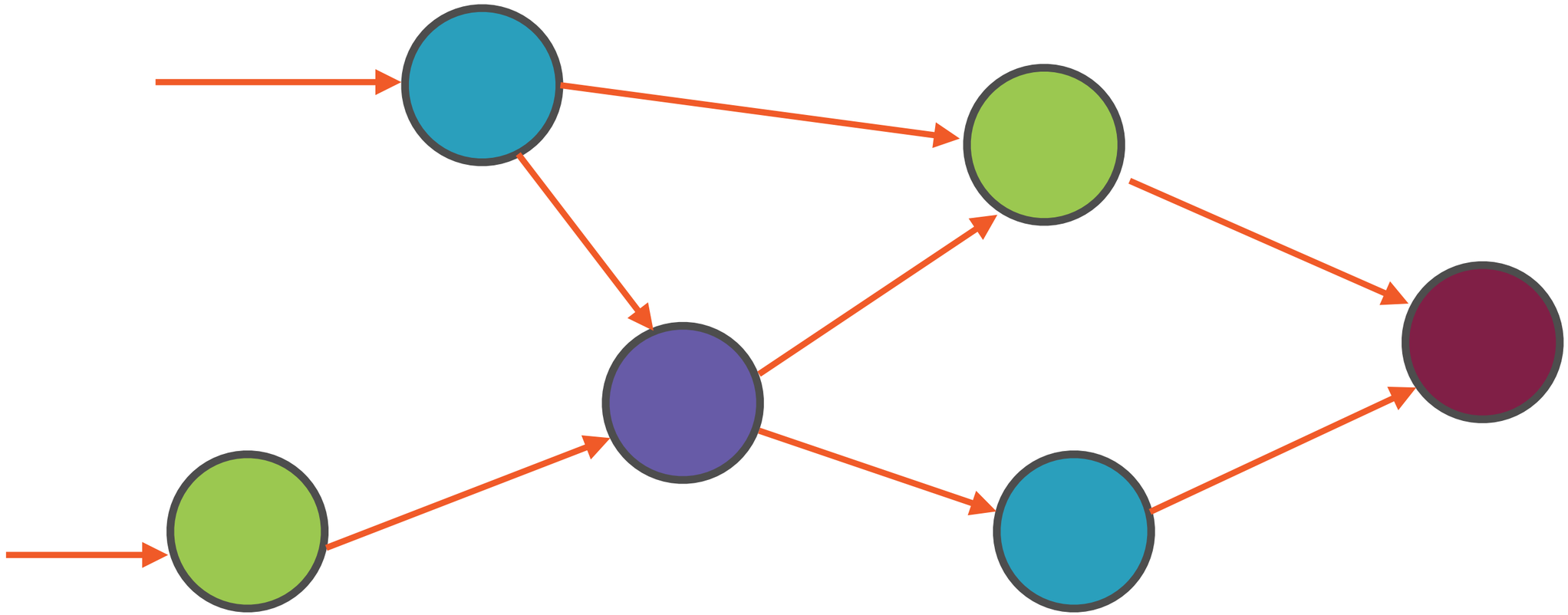
Computations Operations

Everything Is a Graph



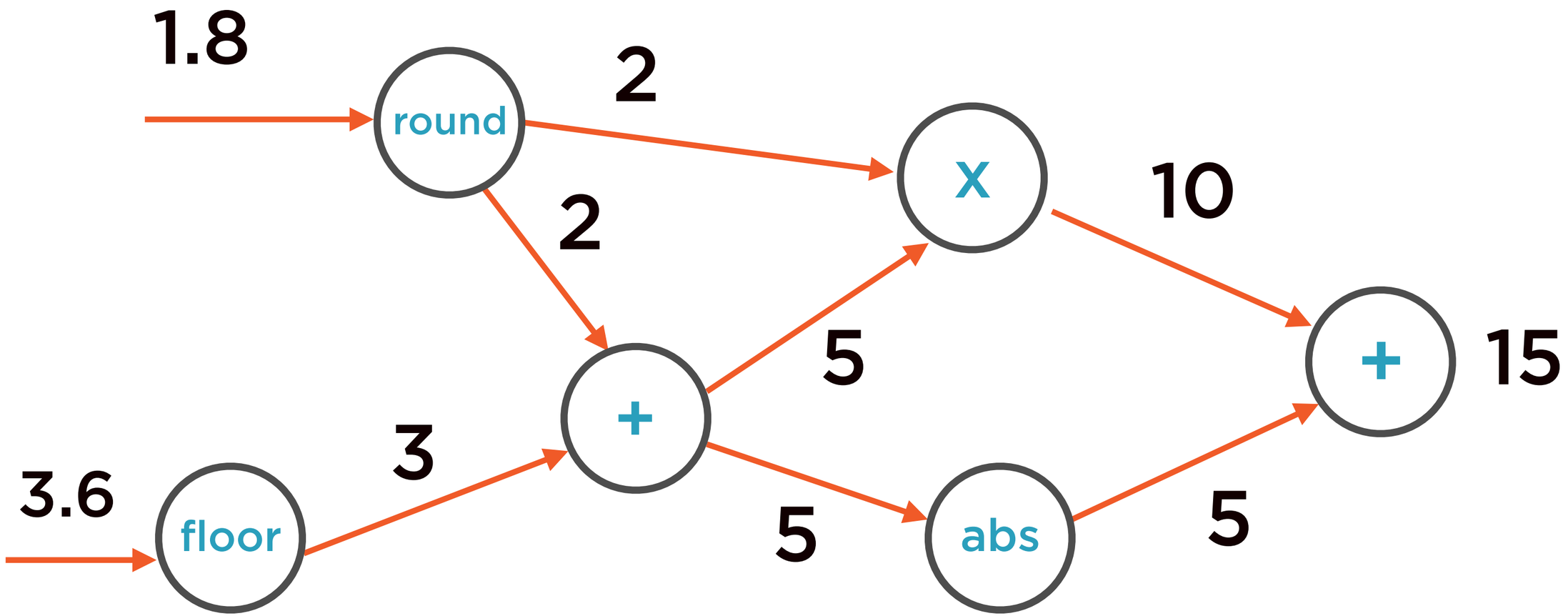
Tensors Data

Tensors Flow Through the Graph

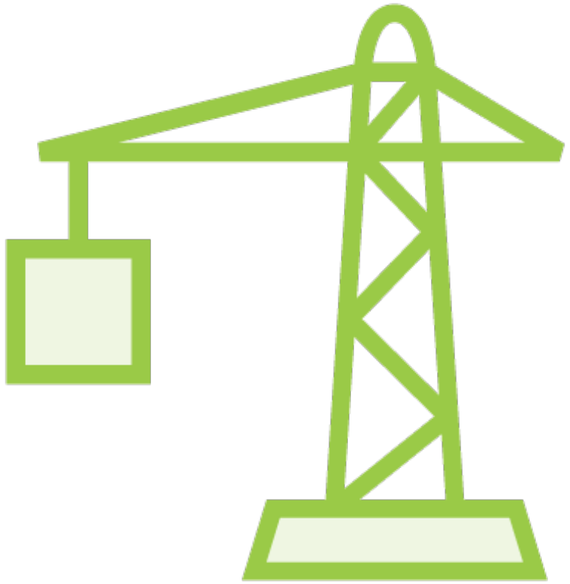


...and get transformed along the way

Tensors Flow Through the Graph



TensorFlow: “Define, then Run”



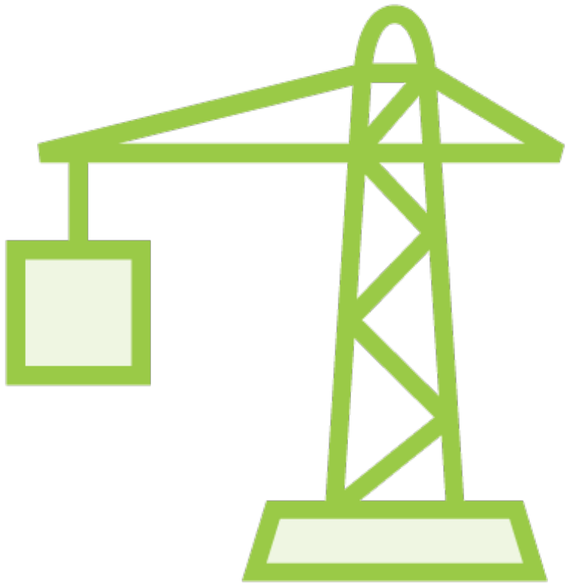
Building a Graph

Specify the operations and the data

Running a Graph

Execute the graph to get the final result

PyTorch: “Define by Run”



Building a Graph

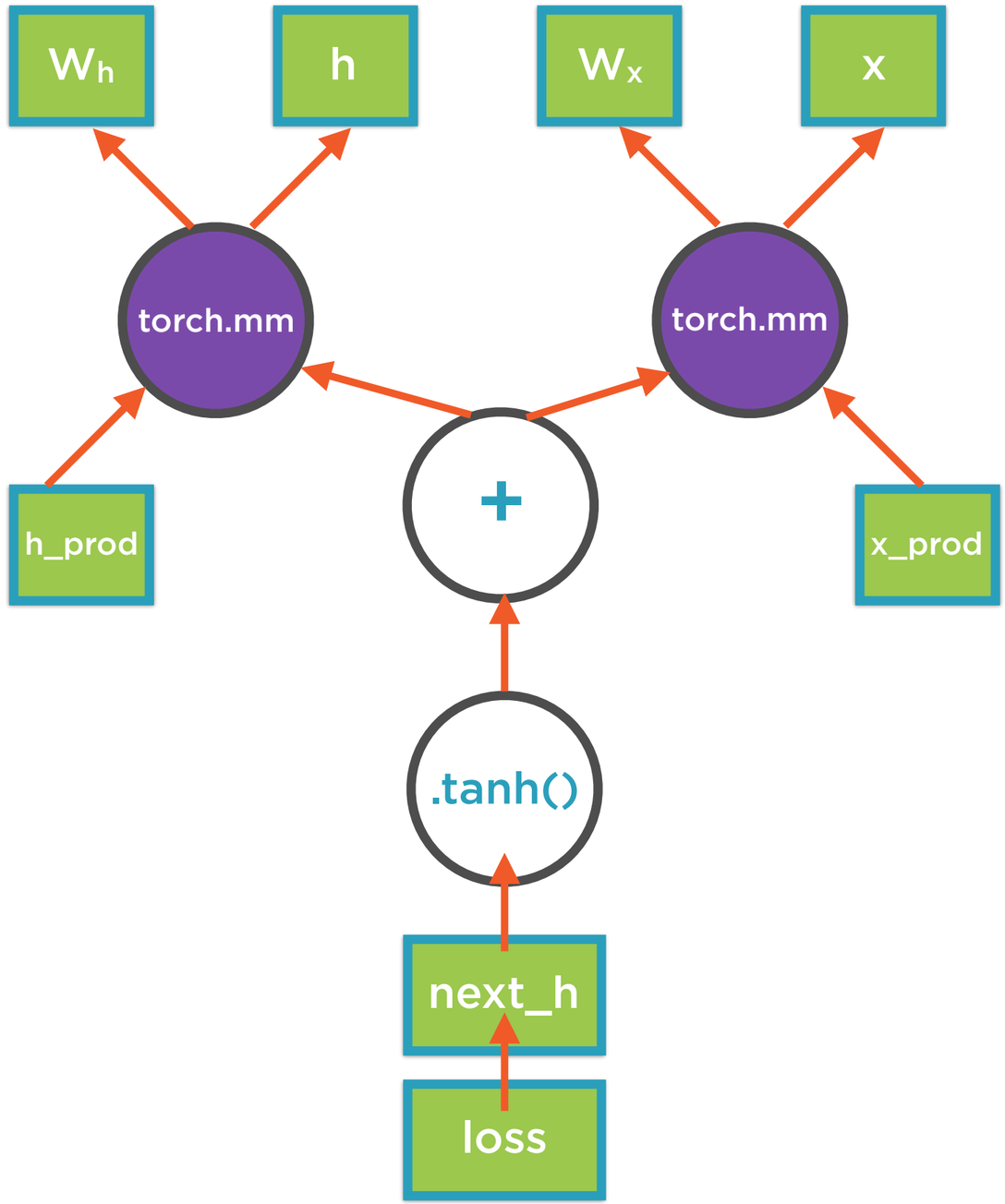
Specify the operations and the data

Running a Graph

Execute the graph to get the final result

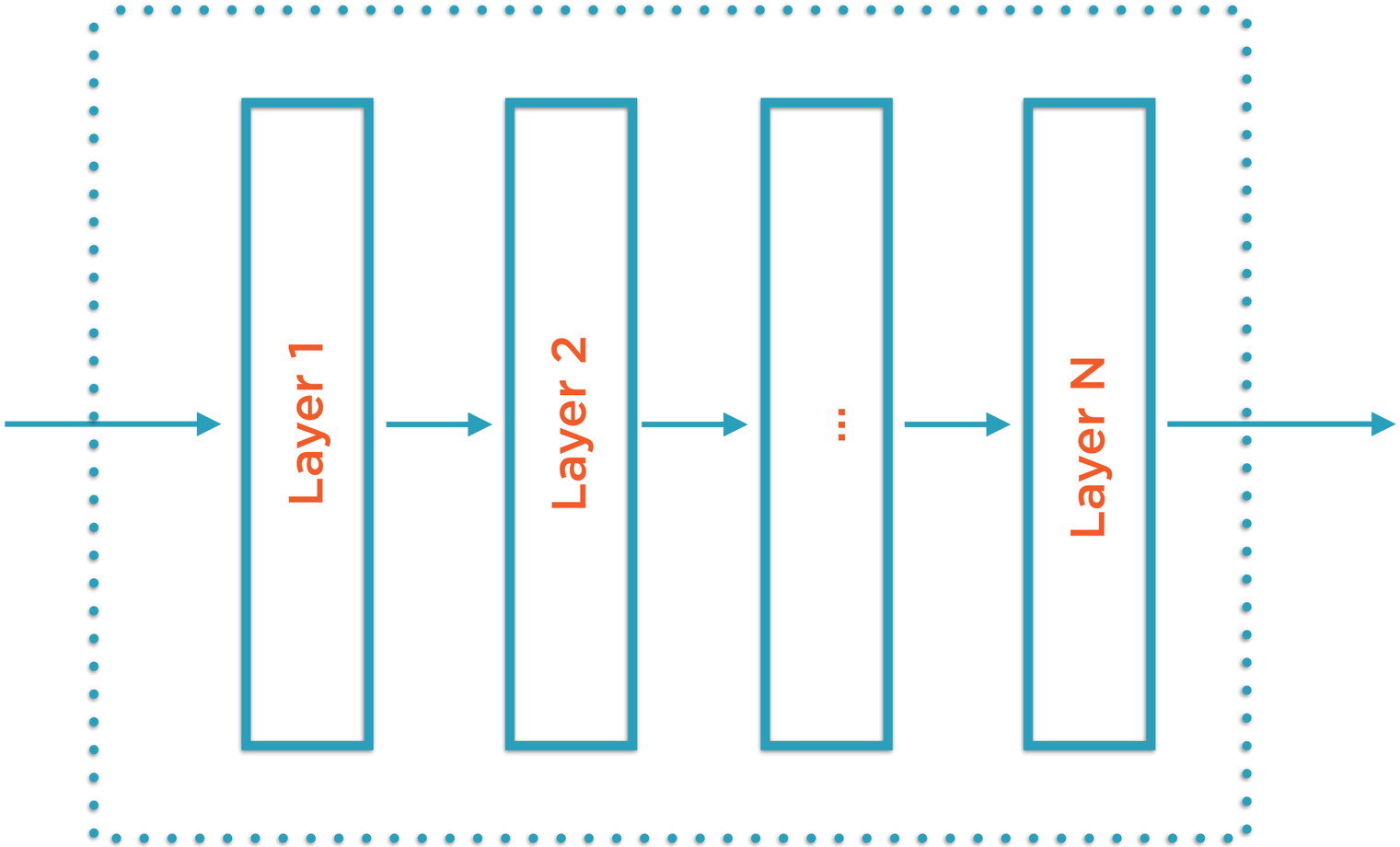
Build and execute the graph in one go - execute as you build

```
from torch.autograd import Variable

x = Variable(torch.randn(1,10))

h = Variable(torch.randn(1,20))

W_h = Variable(torch.randn(20,20))

W_x = Variable(torch.randn(20,10))


h_prod = torch.mm(W_h,h.t())

x_prod = torch.mm(W_x,x.t())

next_h = (h_prod + x_prod).tanh()

loss = next_h.sum()

loss.backward()
```



Training an NN Using Gradient Descent

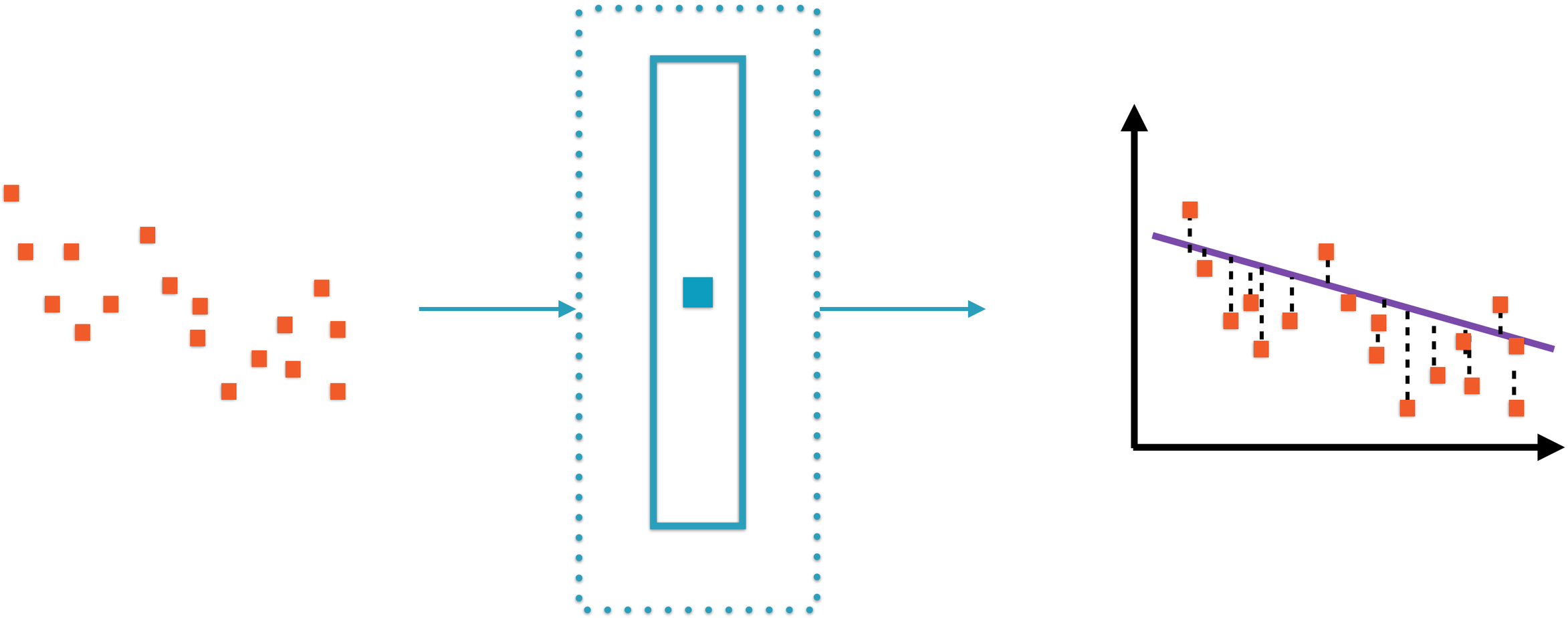
Neural Network Model



Network of interconnected layers

The **weights** and **biases** of individual neurons are determined during the **training** process

Regression: The Simplest Neural Network

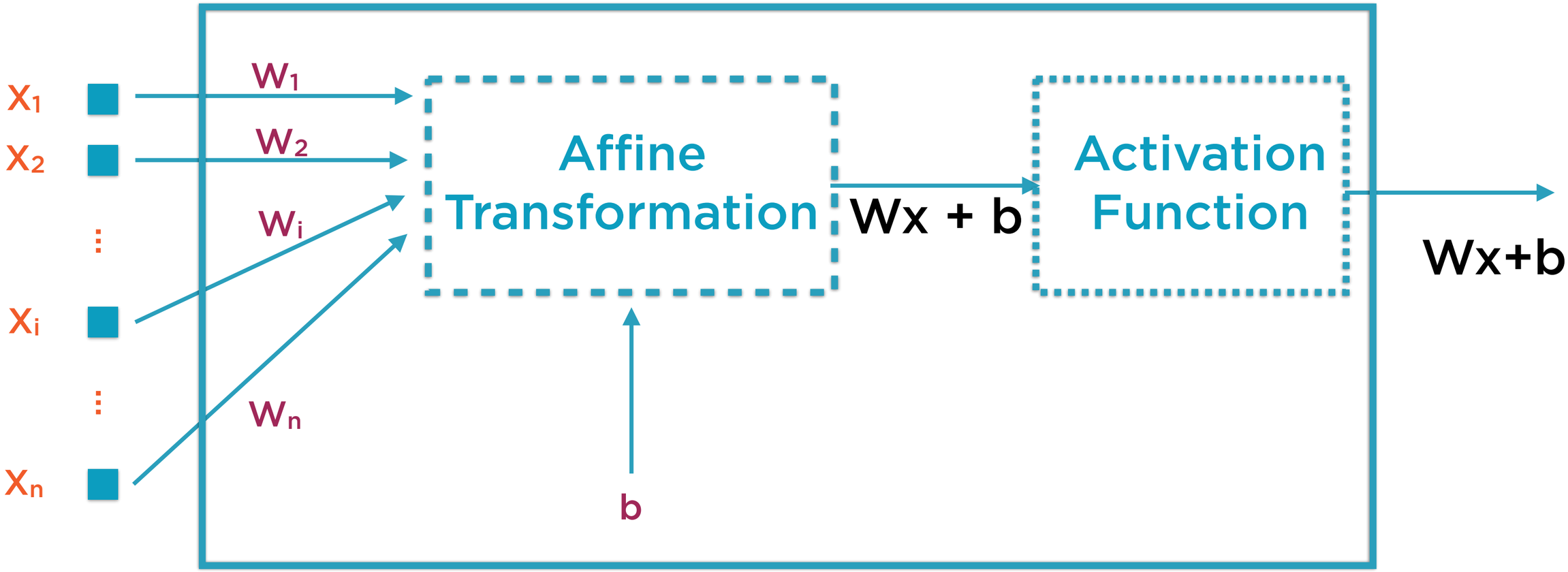


Set of Points

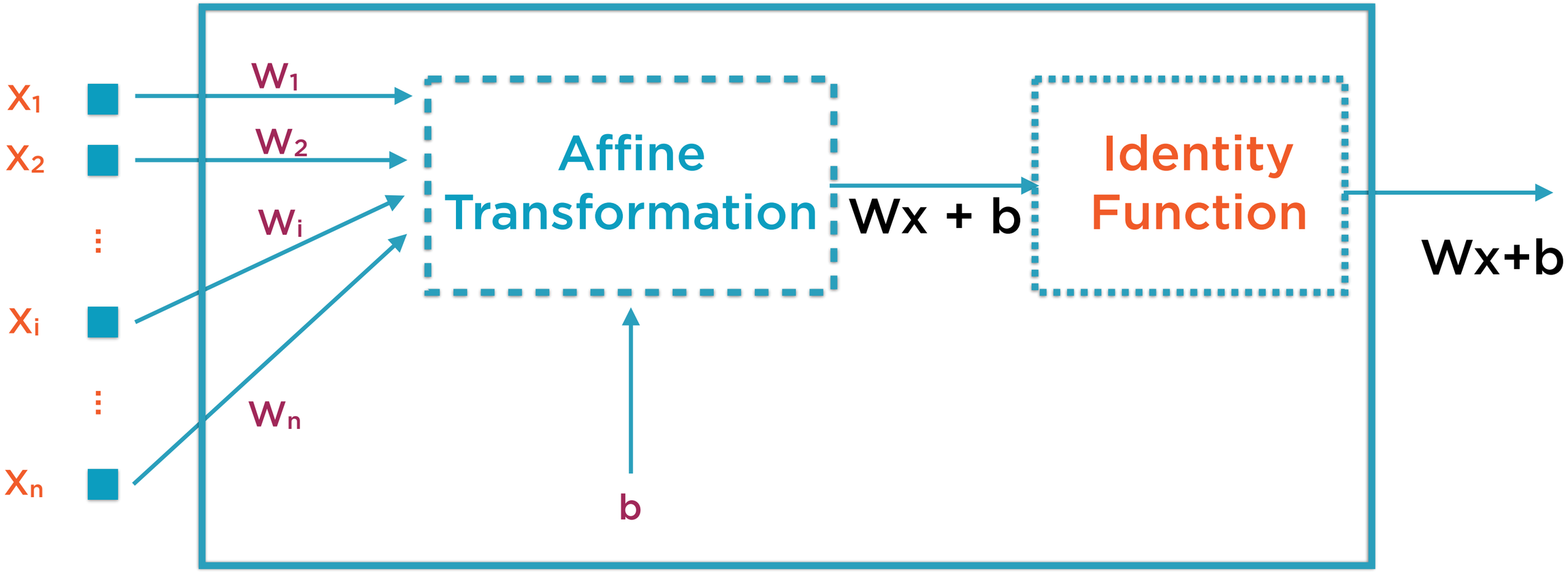
Single Neuron

Regression Line

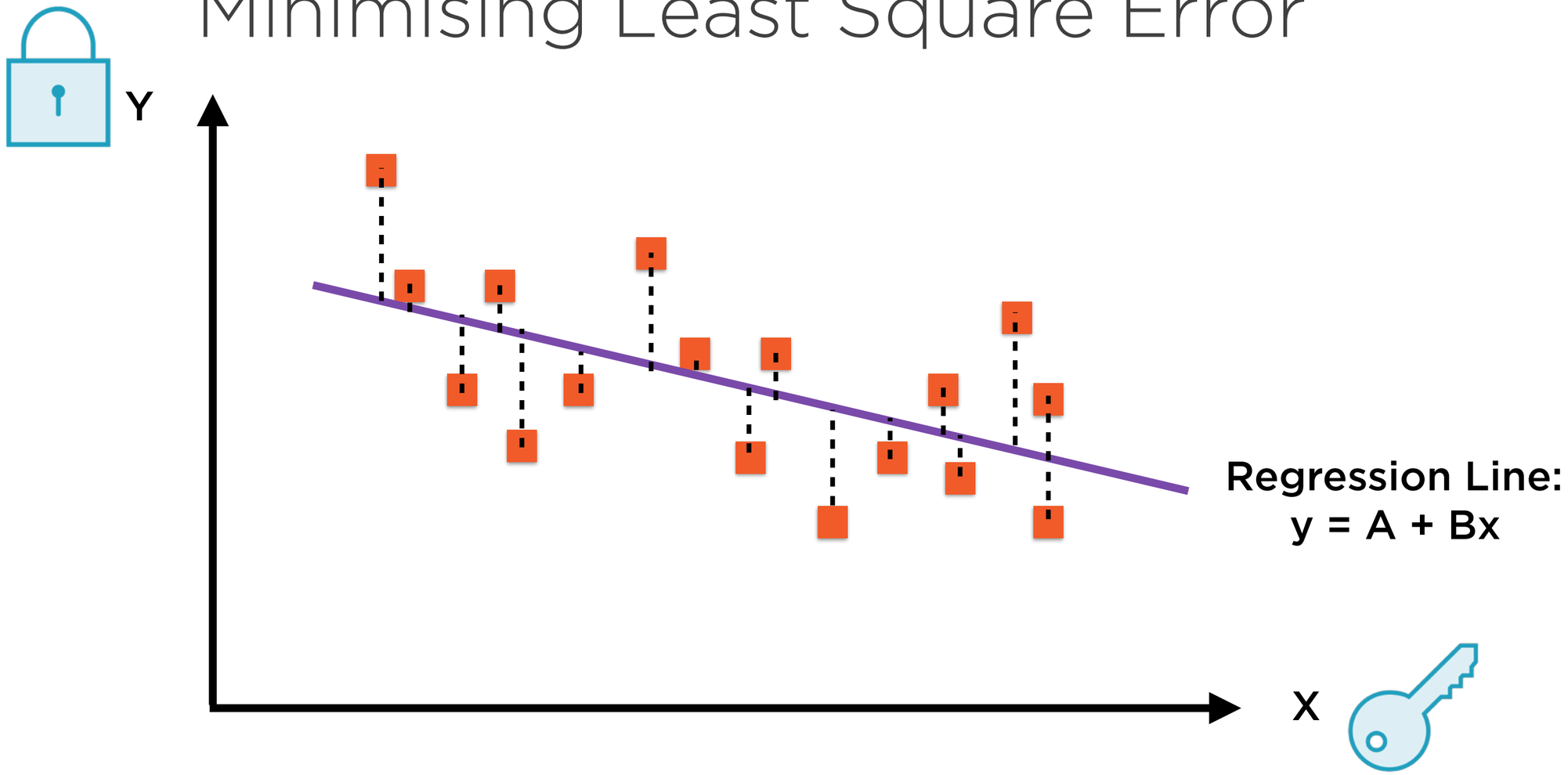
Regression: The Simplest Neural Network



Regression: The Simplest Neural Network



Minimising Least Square Error



The “best fit” line is called the regression line

The actual training of a neural network happens via Gradient Descent Optimization

Linear Regression as an Optimization Problem

Objective Function

Minimize variance of the residuals (MSE)

Linear Regression as an Optimization Problem

Objective Function

Minimize variance of the residuals (MSE)

Constraints

Express relationship as a straight line

$$y = Wx + b$$

Linear Regression as an Optimization Problem

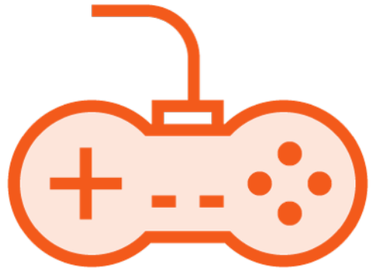
Objective Function

Minimize variance of the residuals (MSE)

Constraints

Express relationship as a straight line

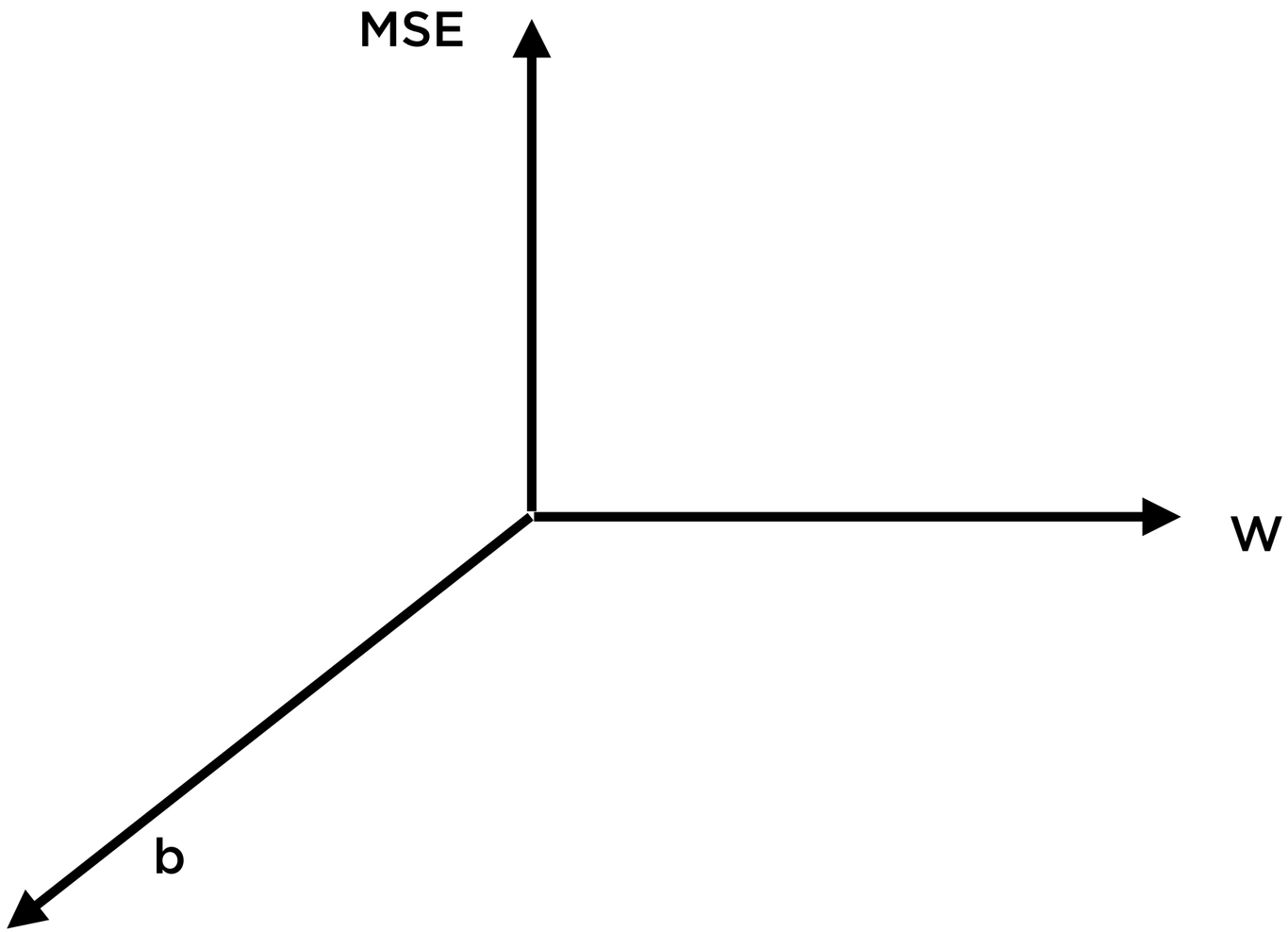
$y = Wx + b$



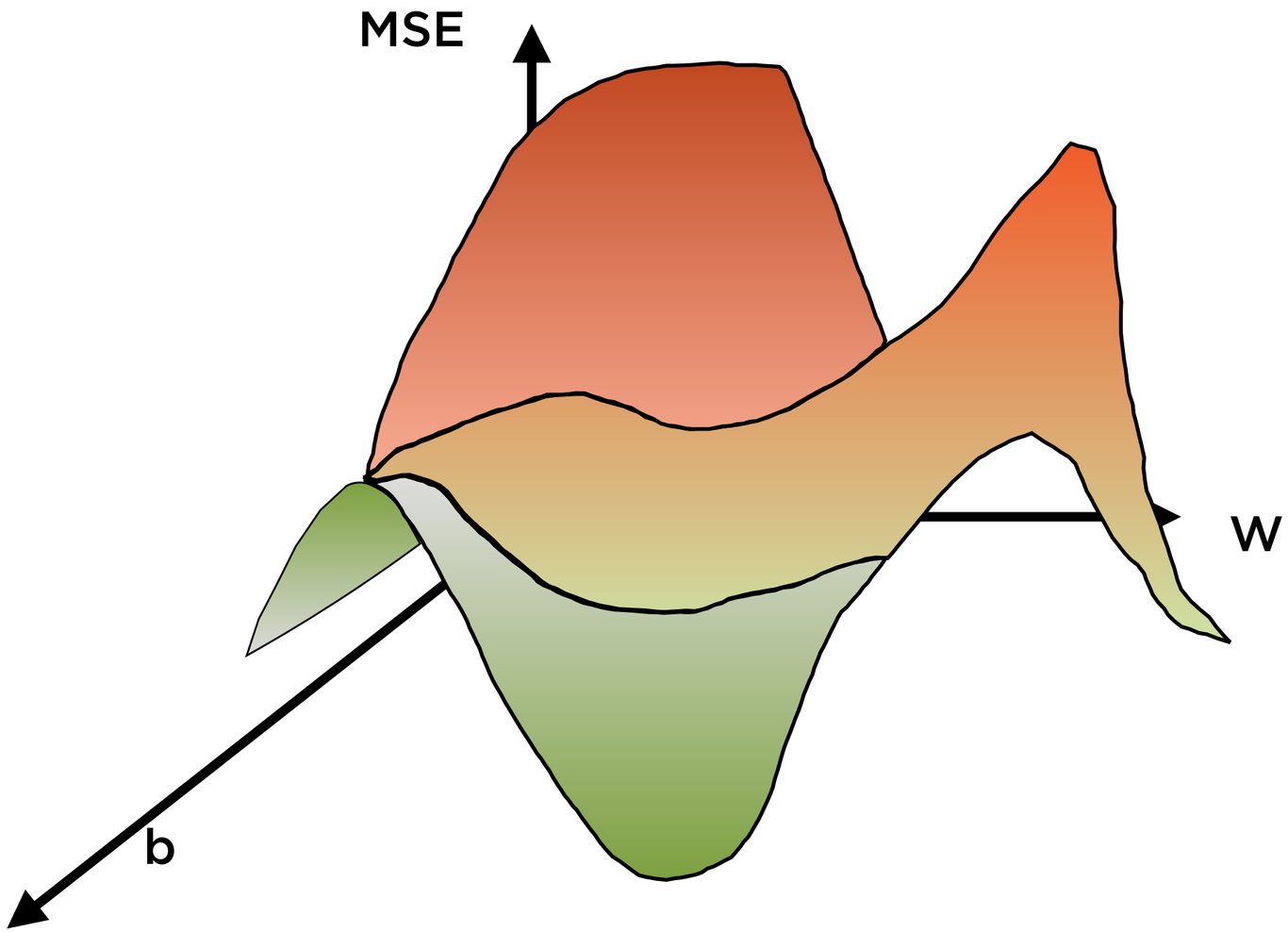
Decision Variables

Values of W and b

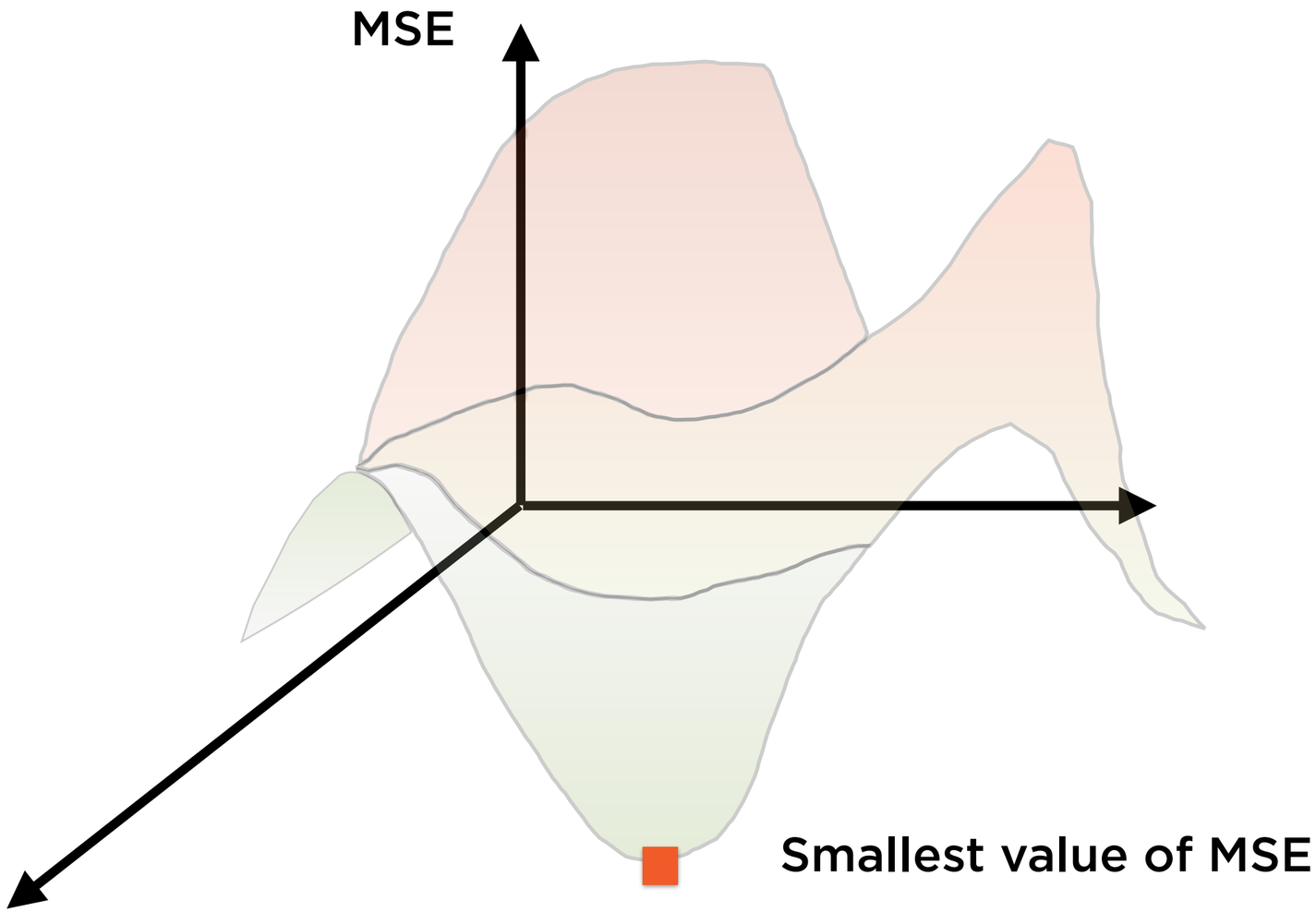
Minimizing MSE



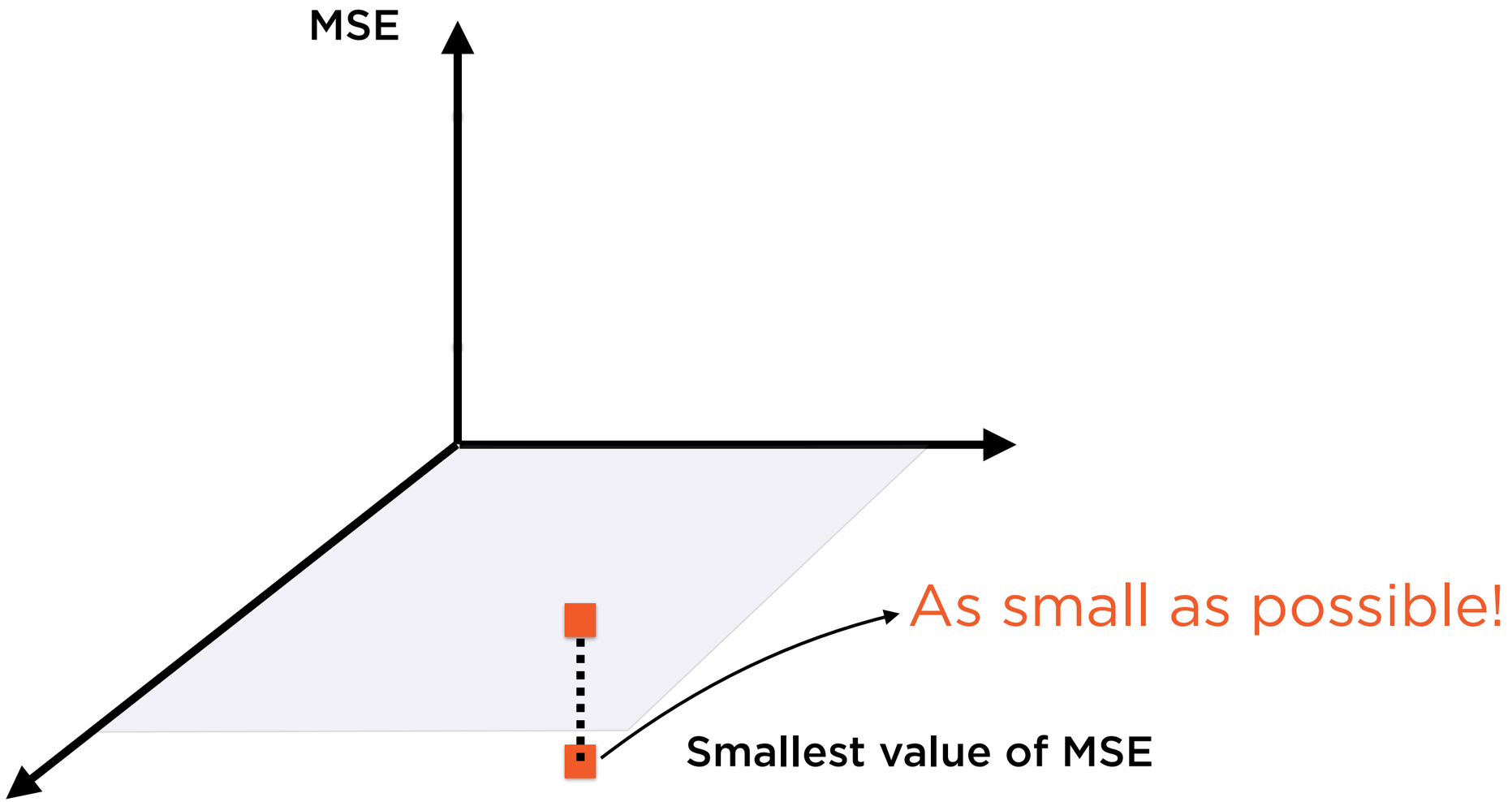
Minimizing MSE



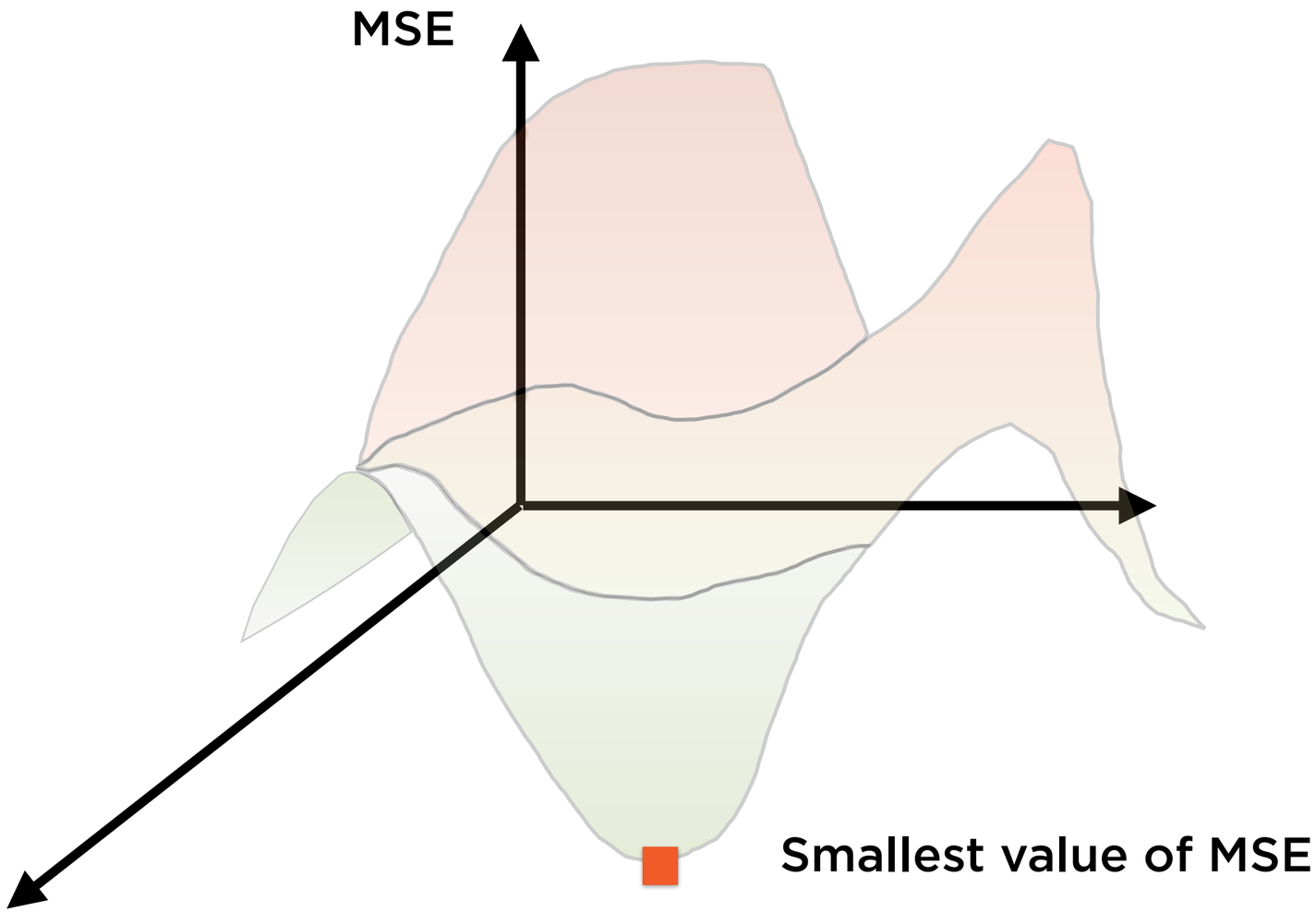
Minimizing MSE



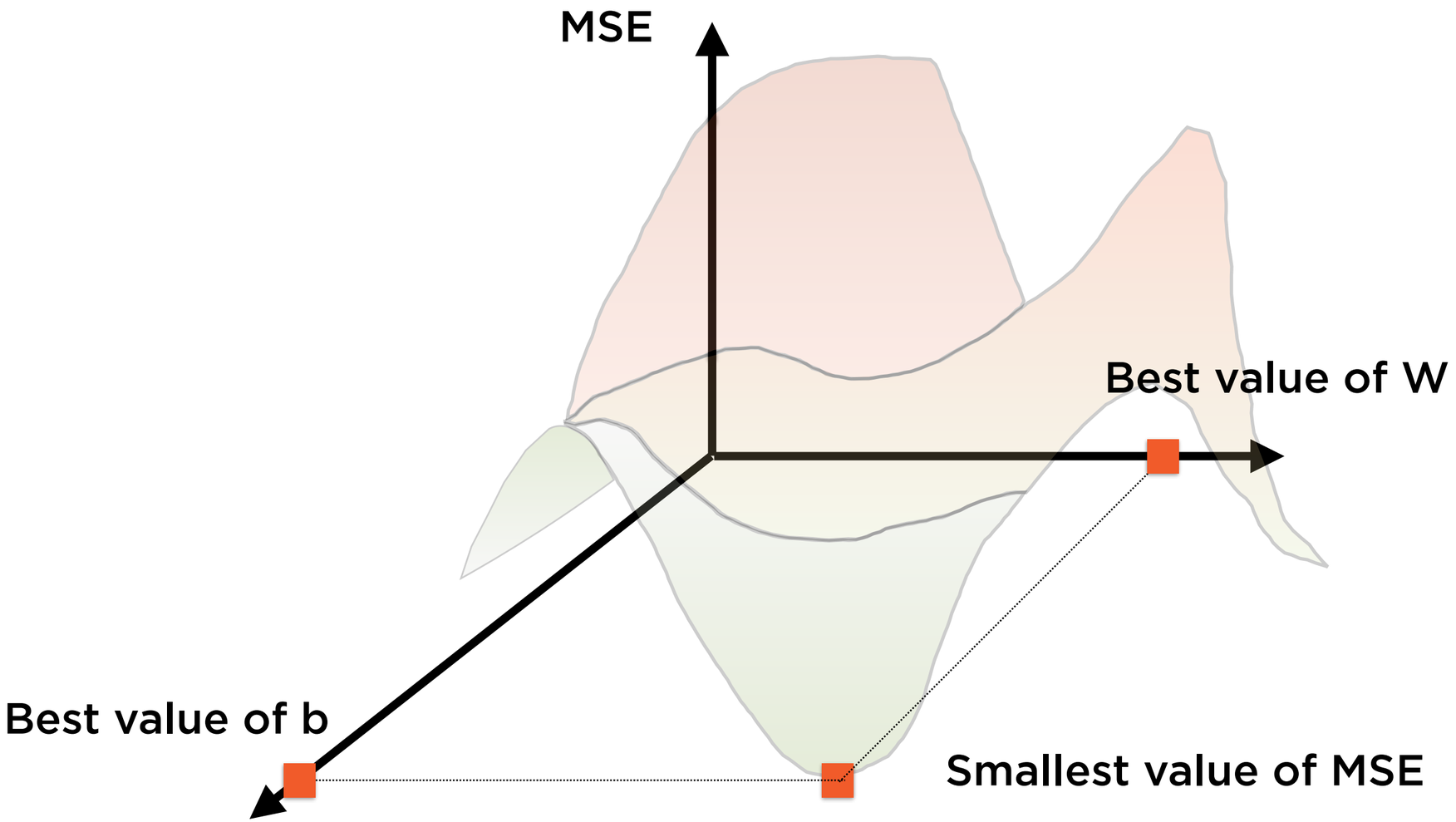
Minimizing MSE



Minimizing MSE

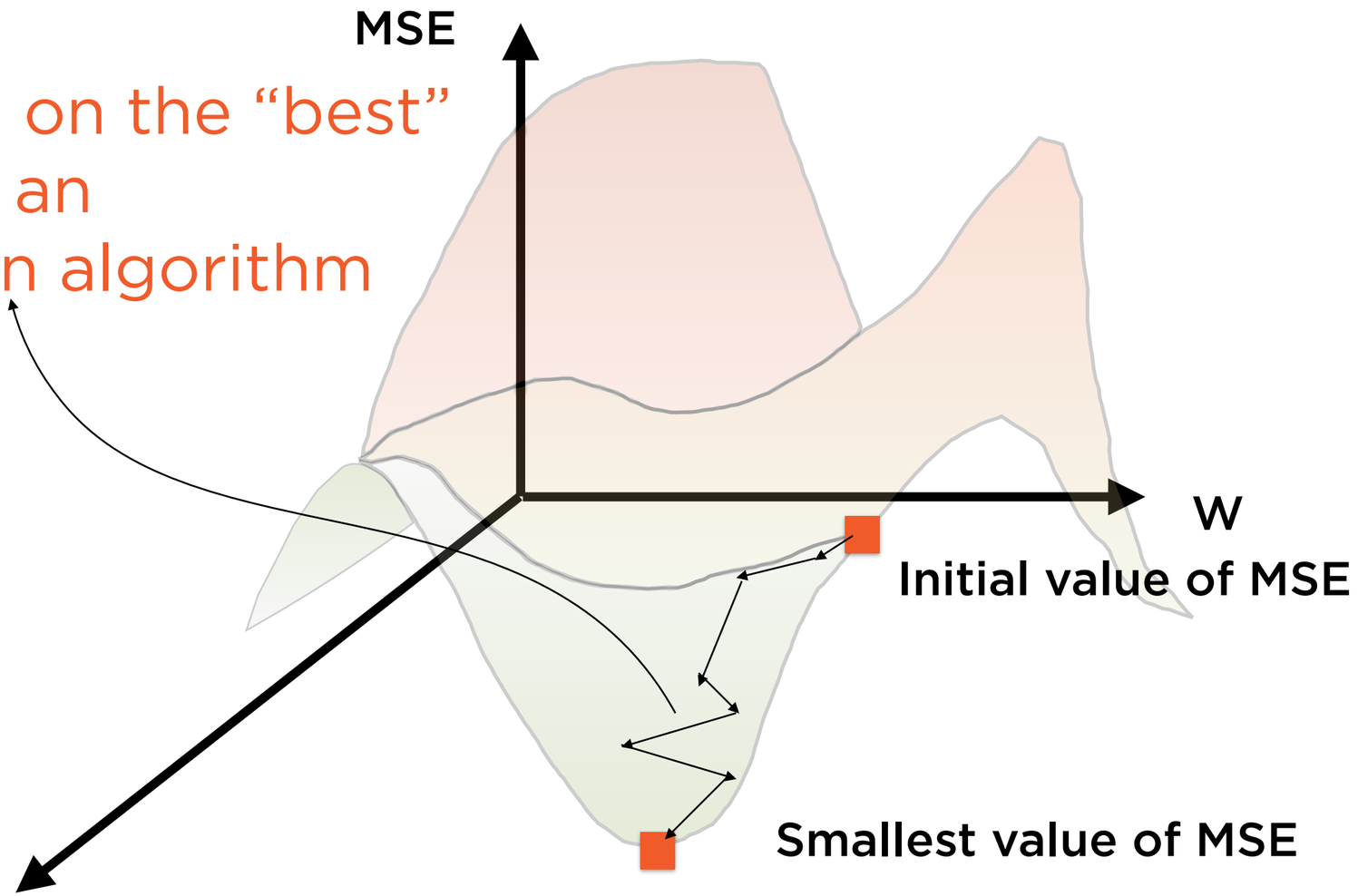


Minimizing MSE

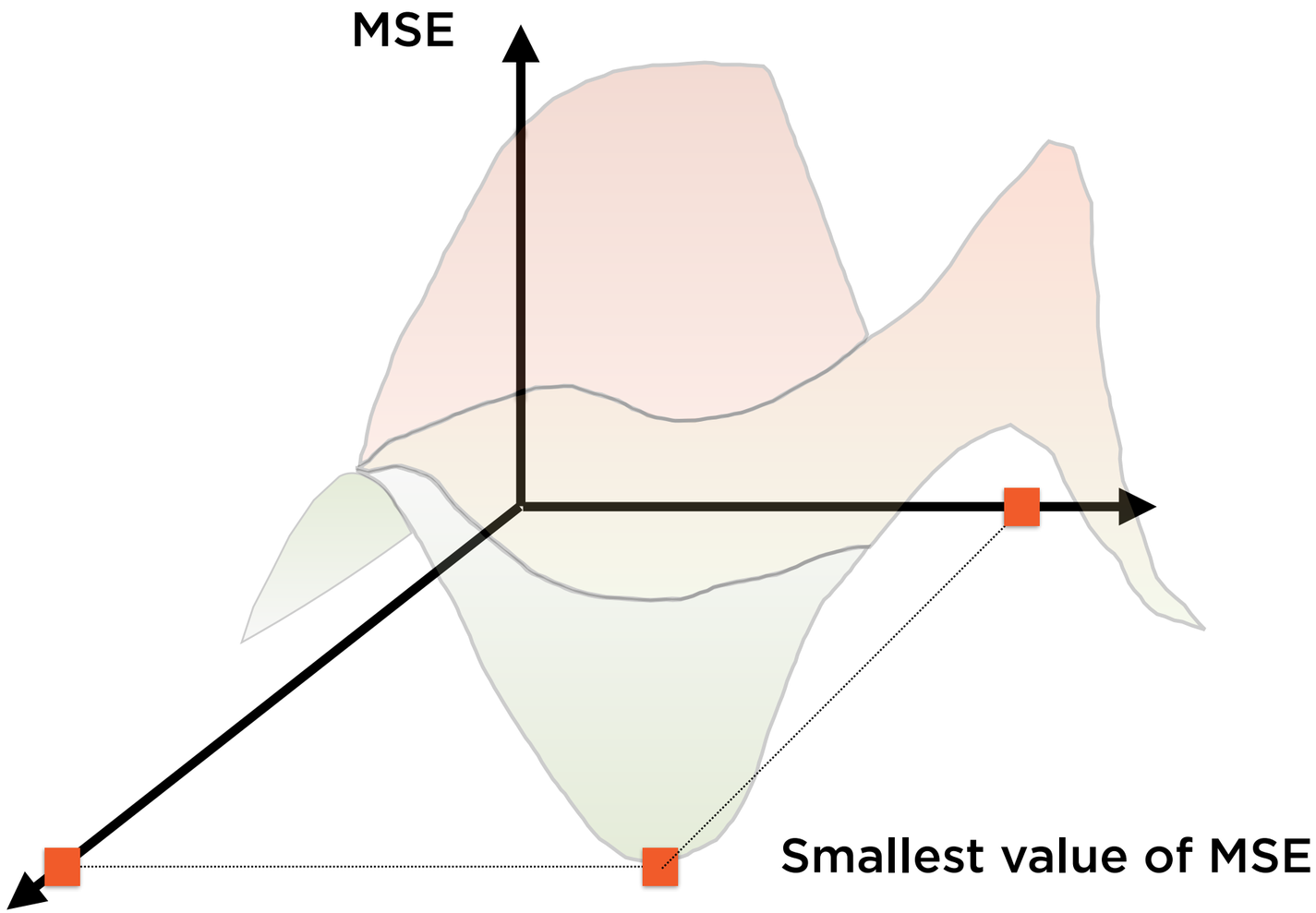


“Gradient Descent”

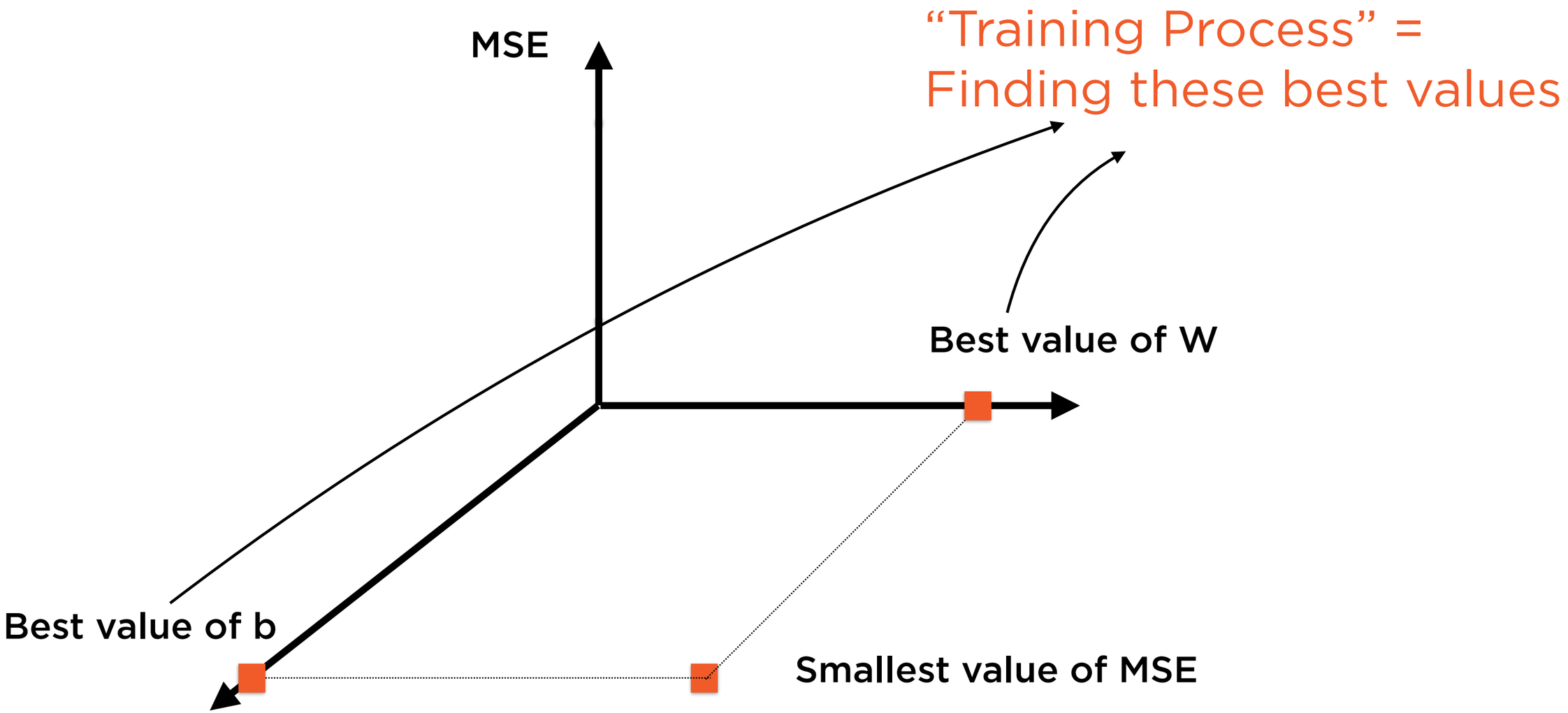
Converging on the “best” value using an optimization algorithm



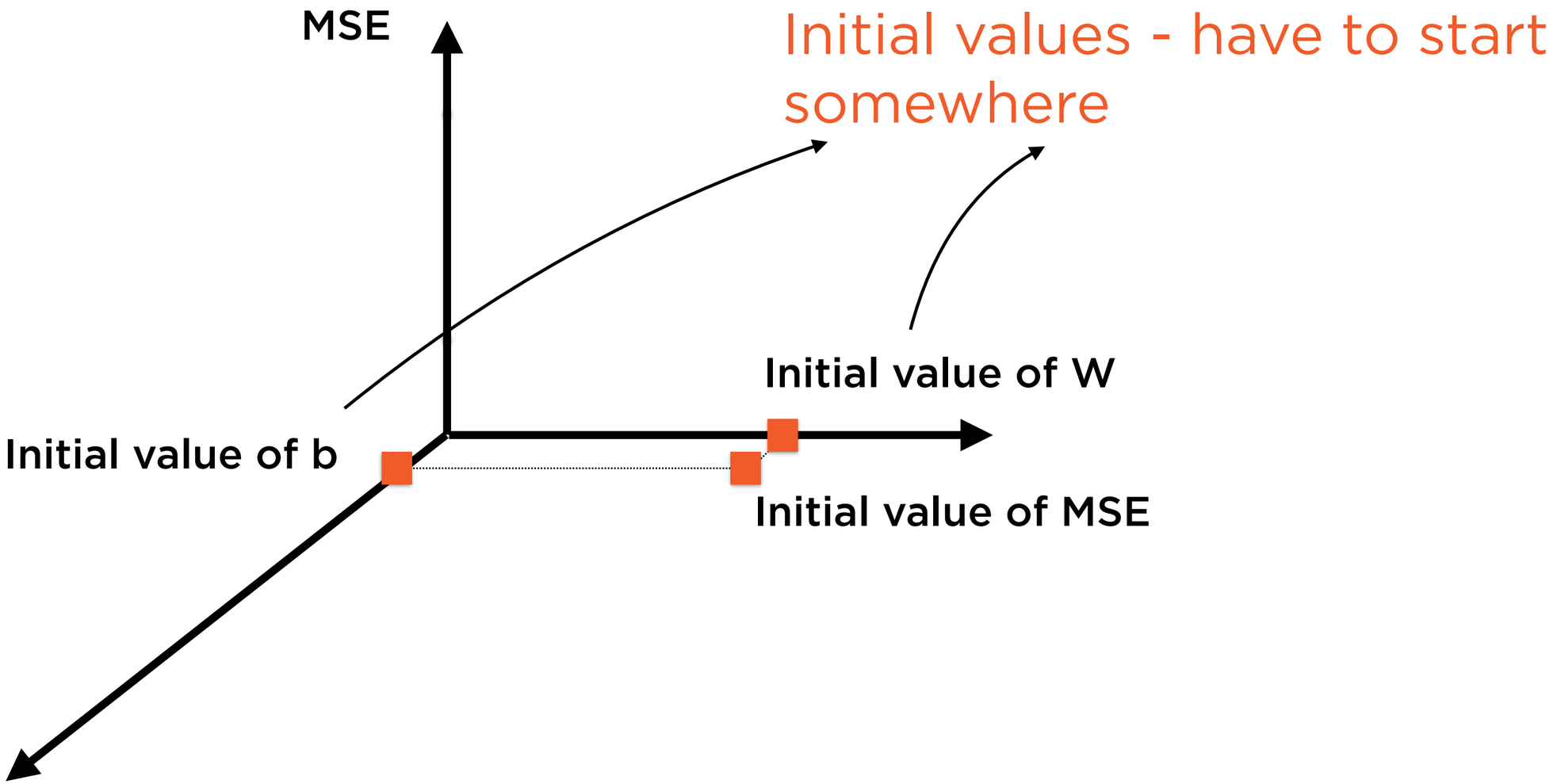
Minimizing MSE



“Training” the Algorithm

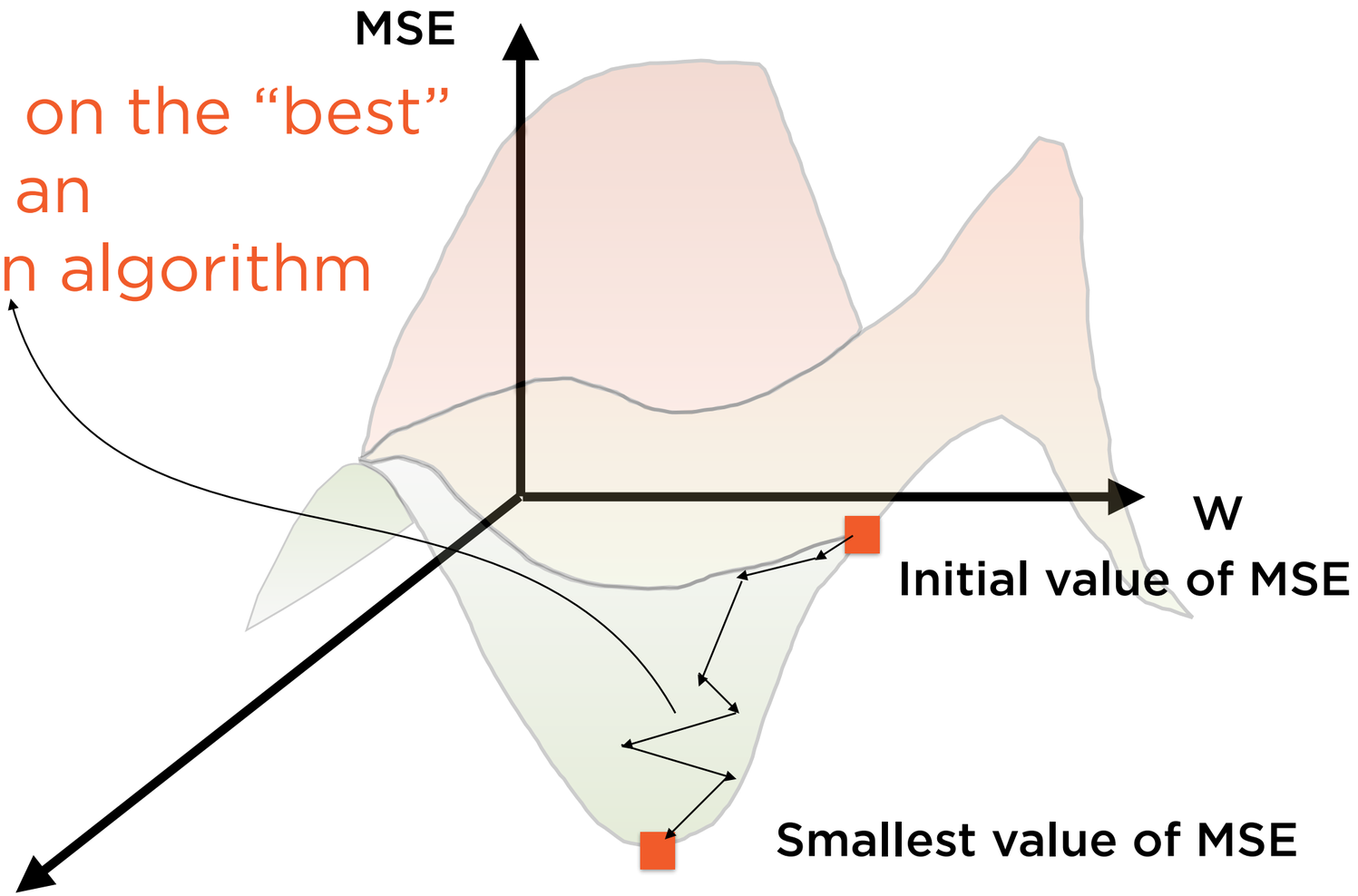


Start Somewhere



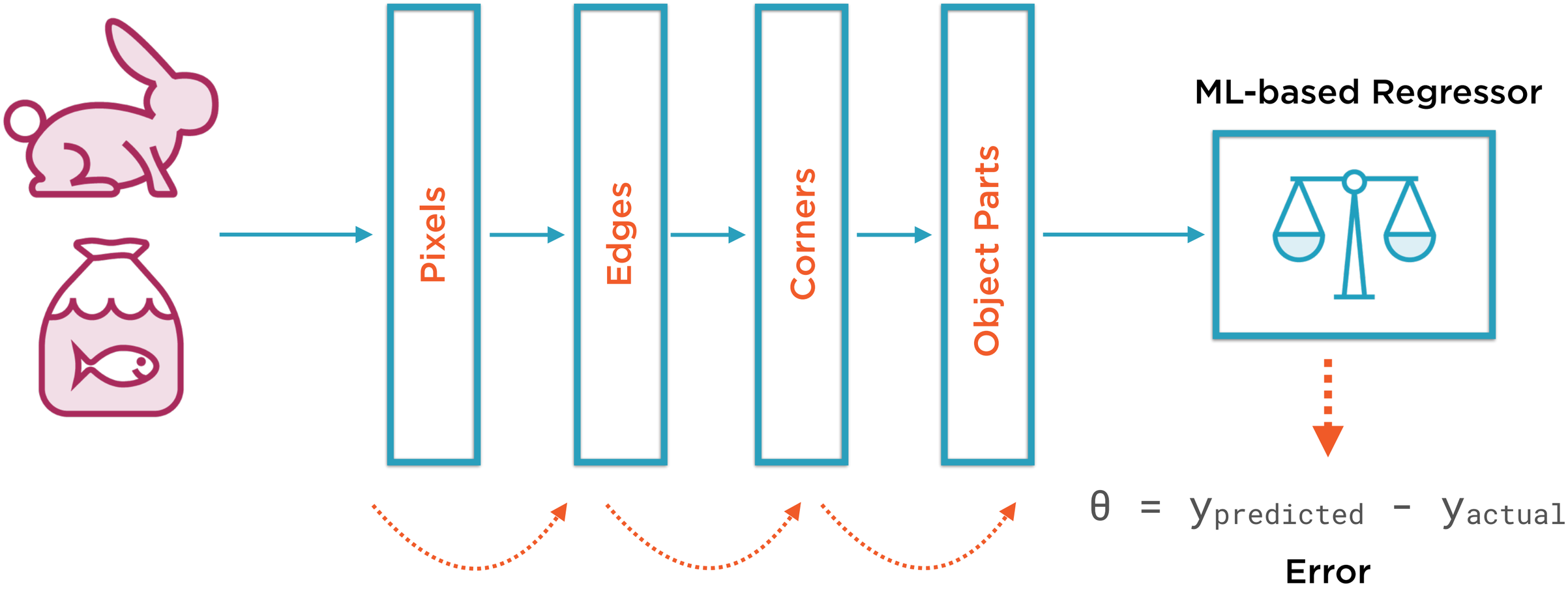
“Gradient Descent”

Converging on the “best” value using an optimization algorithm



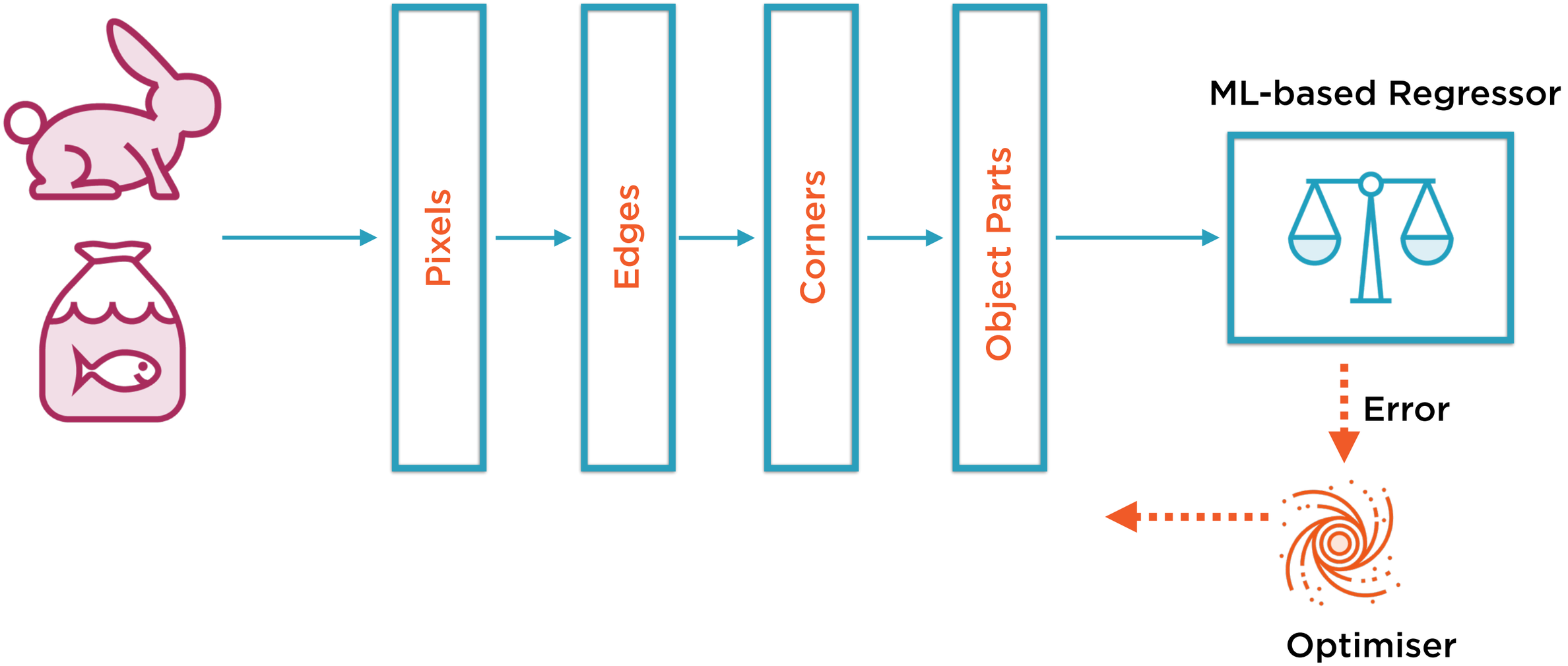
Forward and Backward Passes

Forward Pass

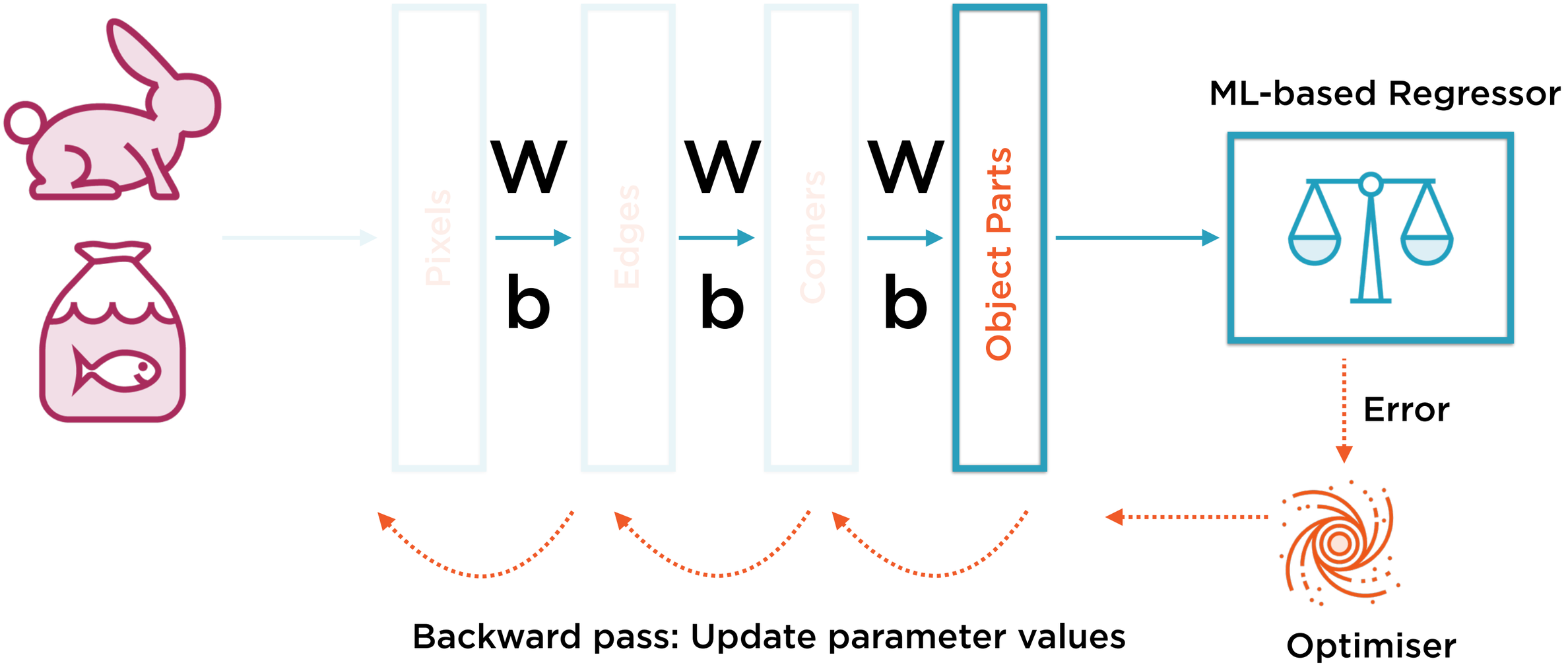


Forward pass: Calculate $y_{predicted}$ and error

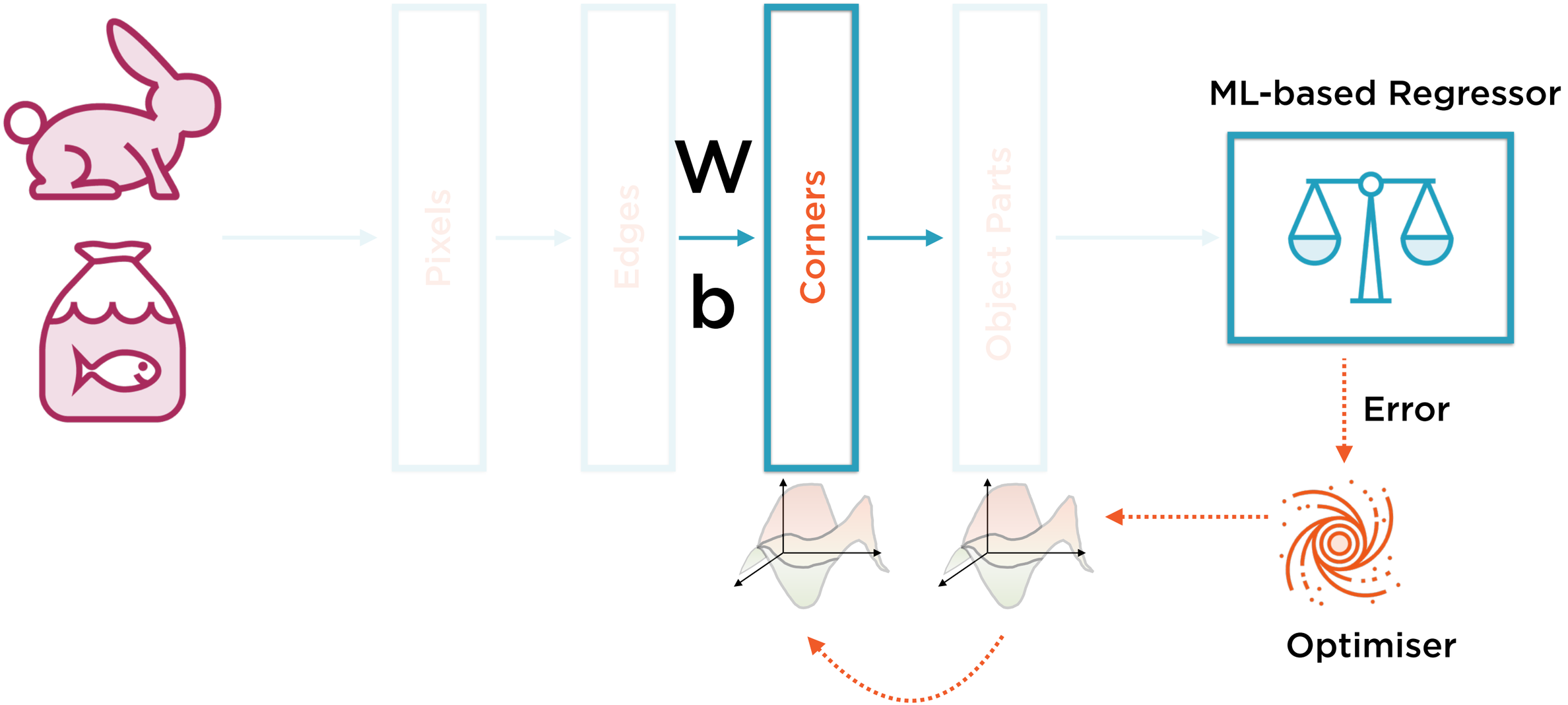
Backward Pass



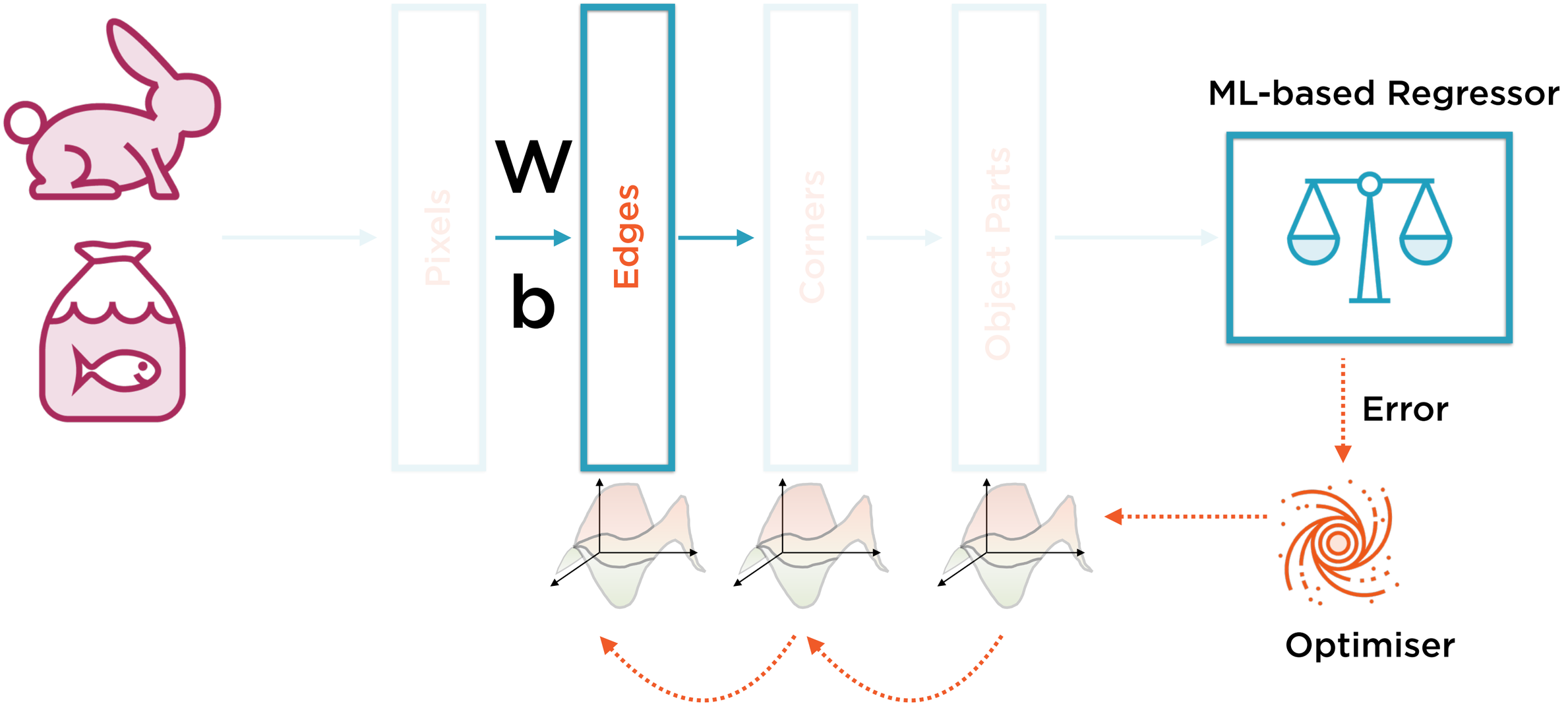
Backward Pass



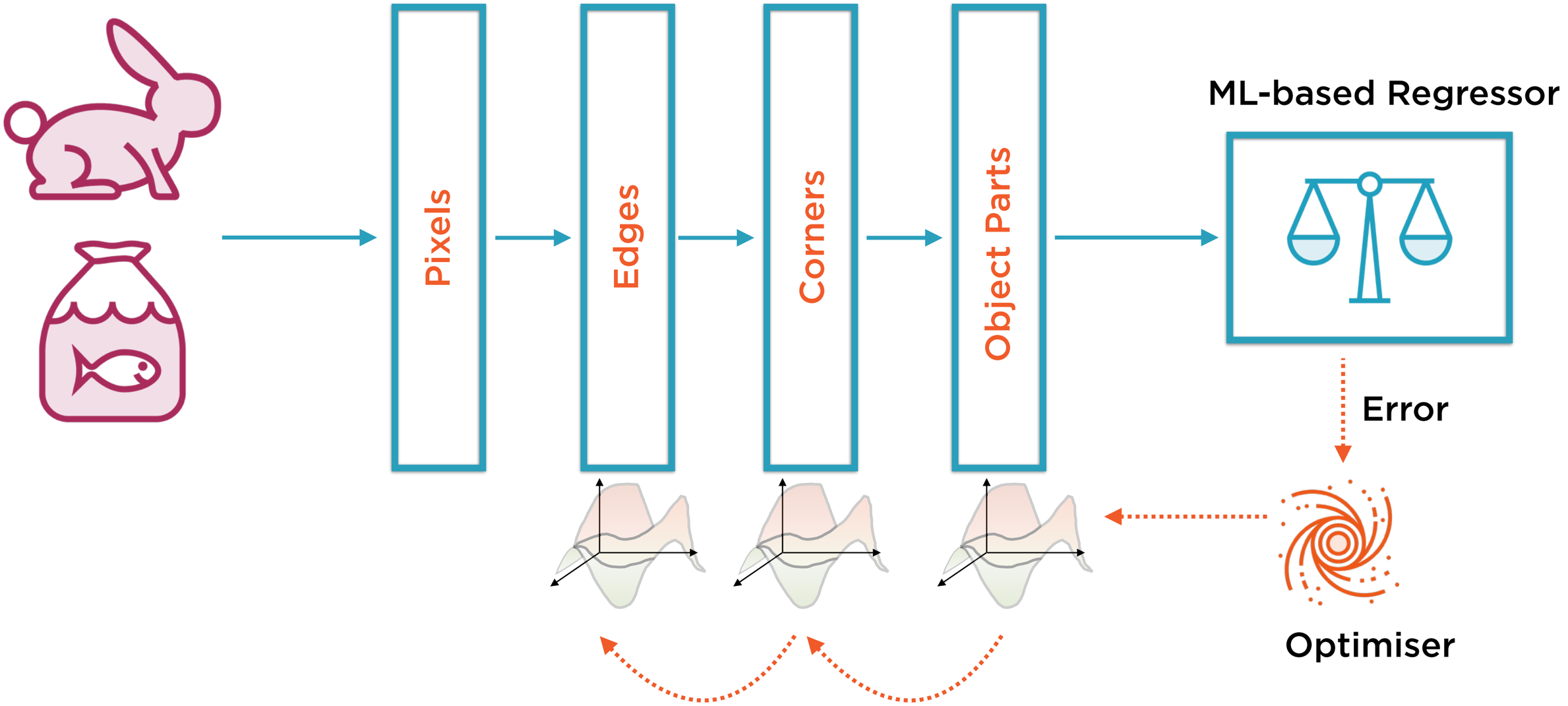
Backward Pass



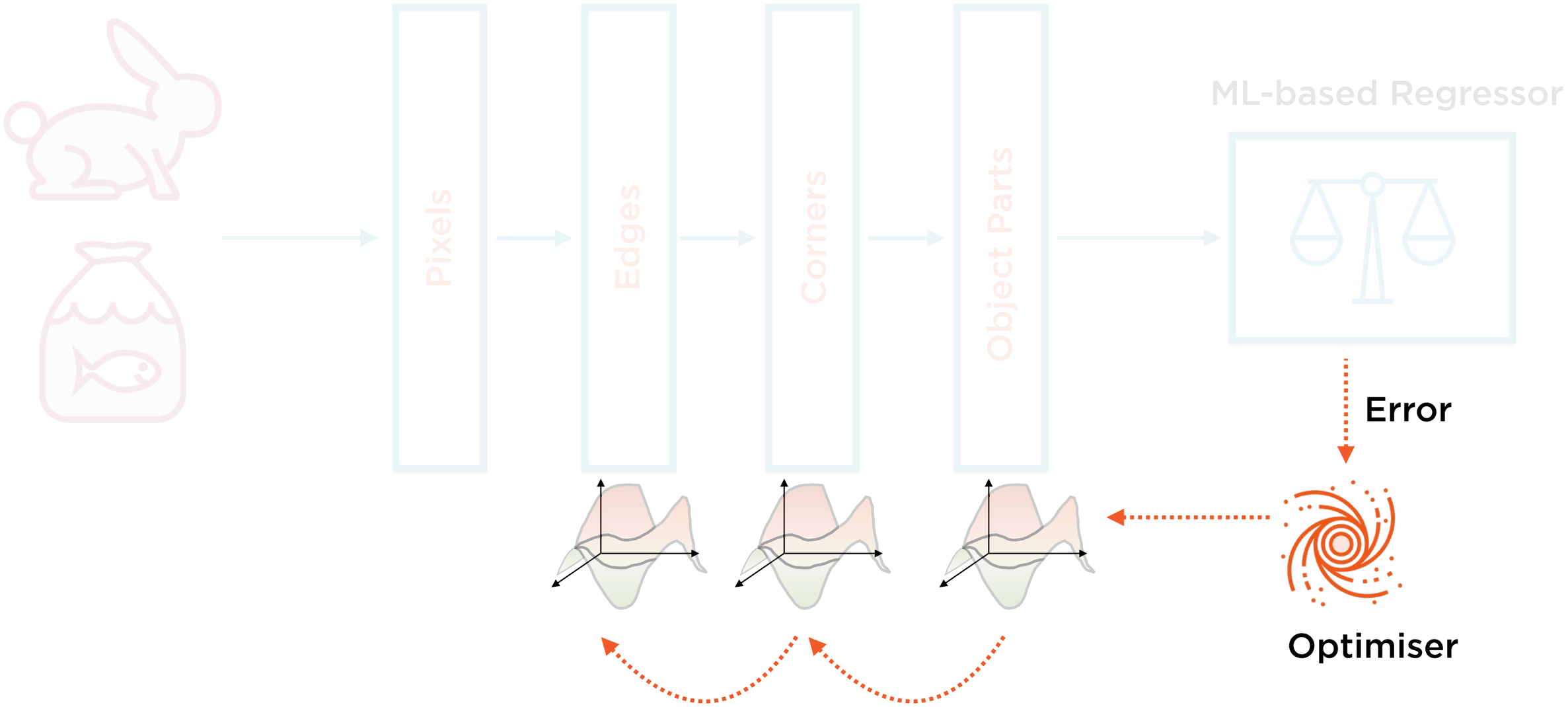
Backward Pass



Backward Pass



Backward Pass



The backward pass allows the weights and biases of the neurons to converge to their final values

Overview

PyTorch is a deep learning framework

More tightly integrated with Python than TensorFlow

Can use Python libraries, debugger

Supports dynamic computation graphs, update the graph for each epoch

Uses a forward pass for prediction, backward pass to update weights