Working with Gradients Using the Autograd Library

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Gradient descent to train a neural network

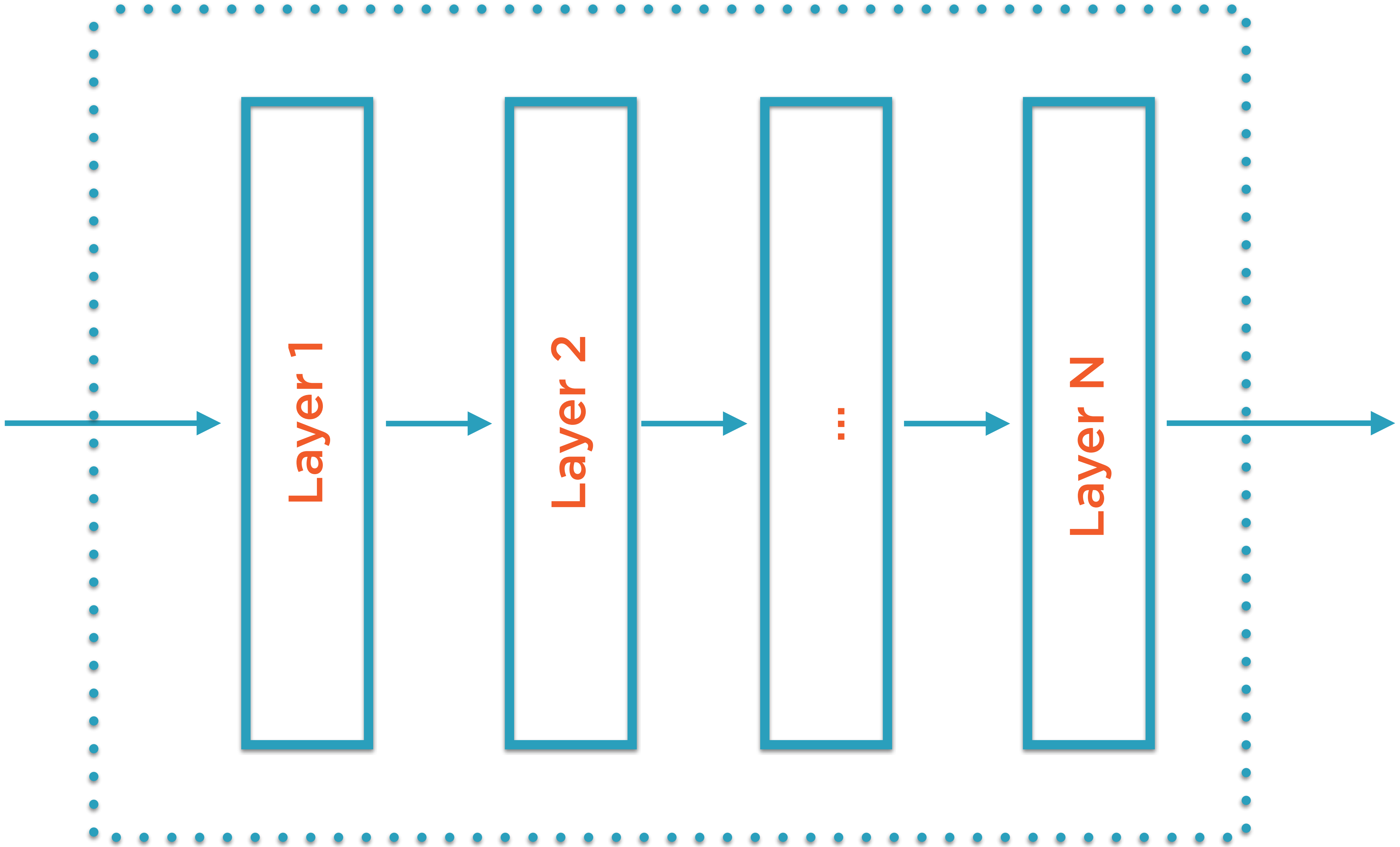
Forward and backward passes

Different methods for gradient calculation

Automatic differentiation using Autograd

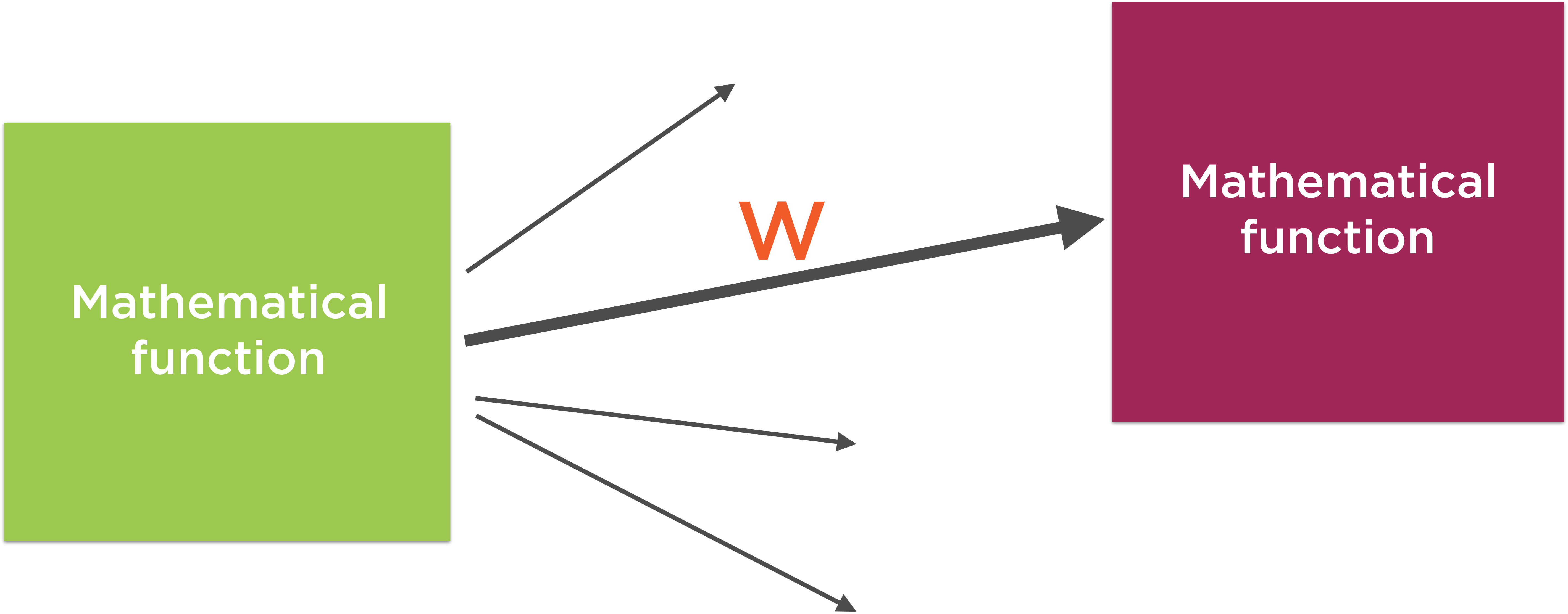
Gradient Descent

Neural Network Model



Interconnected neurons arranged in layers

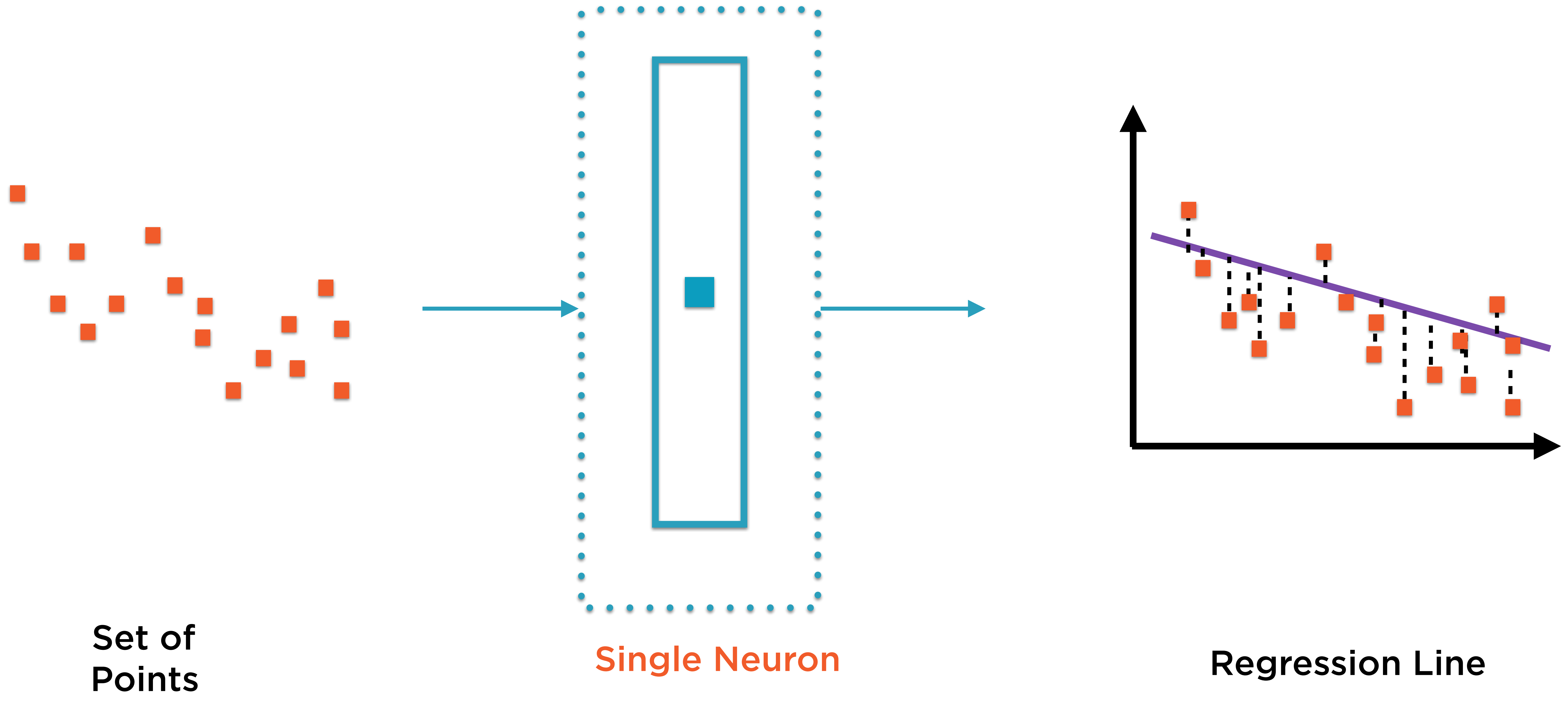
Each Connection Associated with a Weight



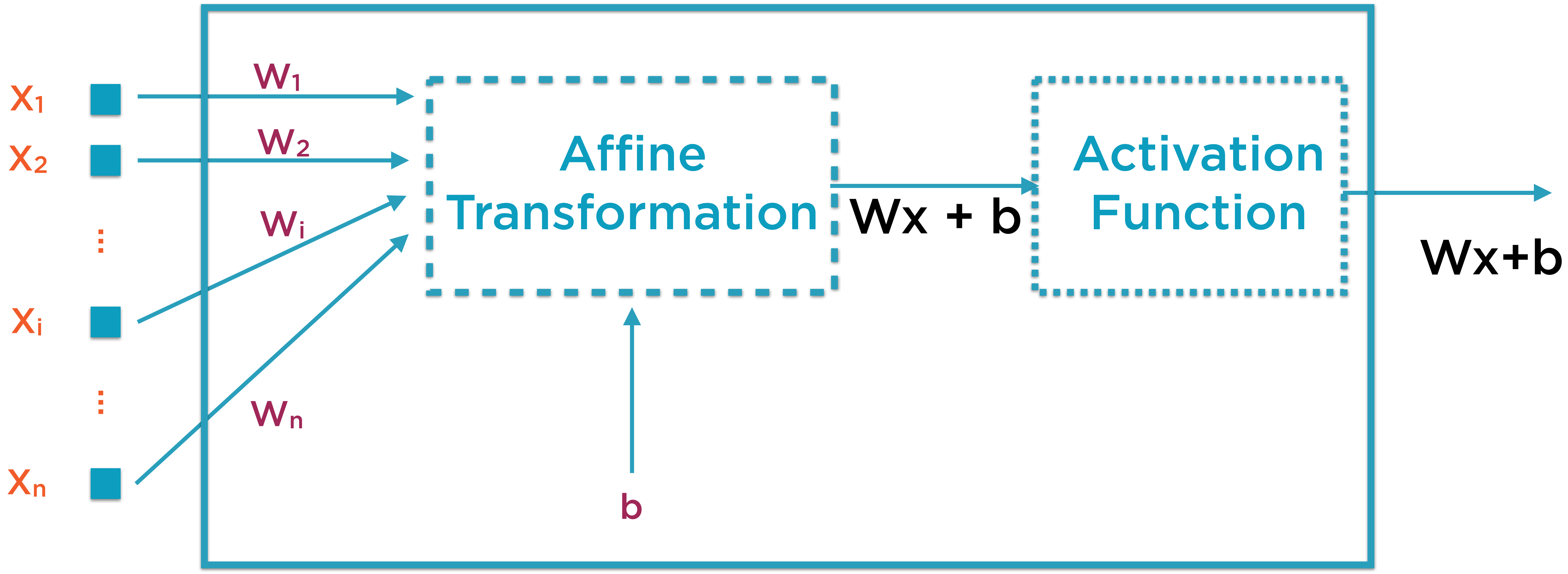
If the second neuron is sensitive to the output of the first neuron, the connection between them gets stronger

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

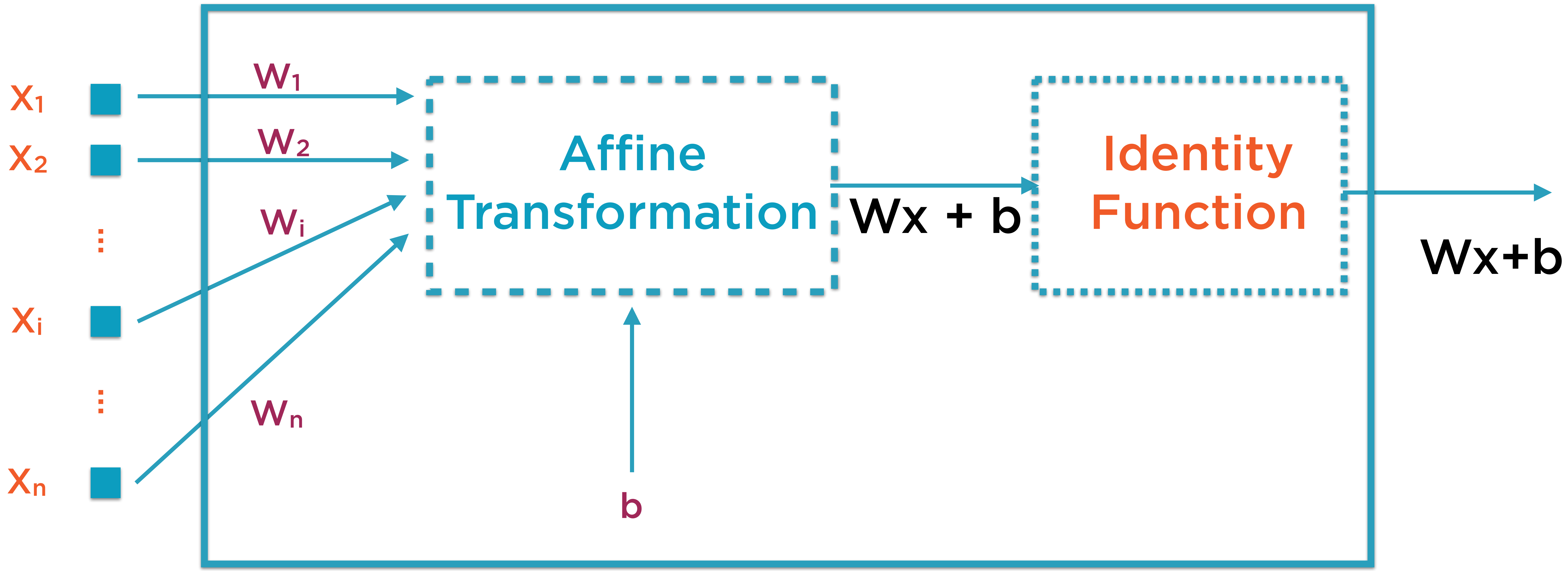
Regression: The Simplest Neural Network



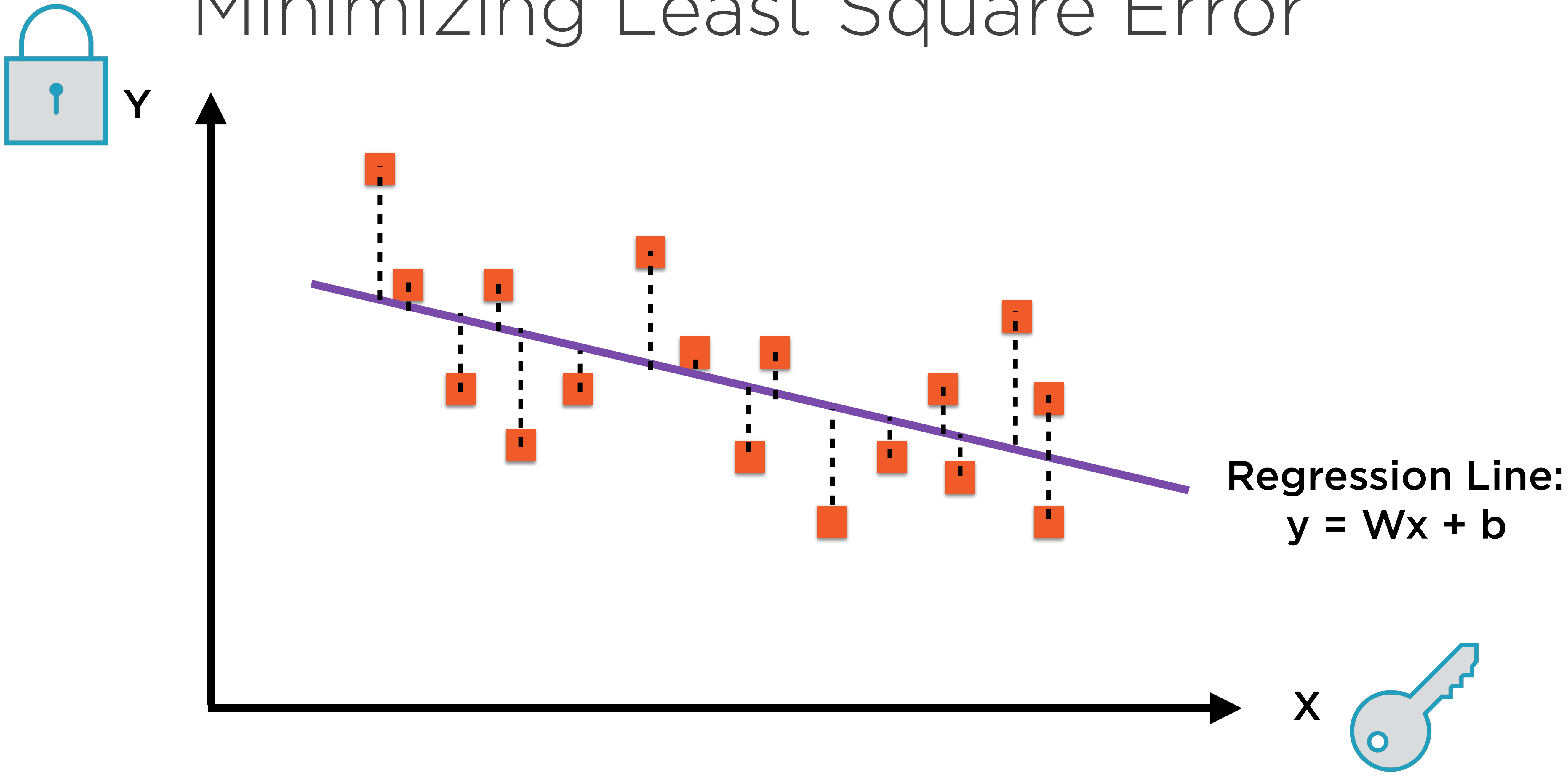
Regression: The Simplest Neural Network



Regression: The Simplest Neural Network

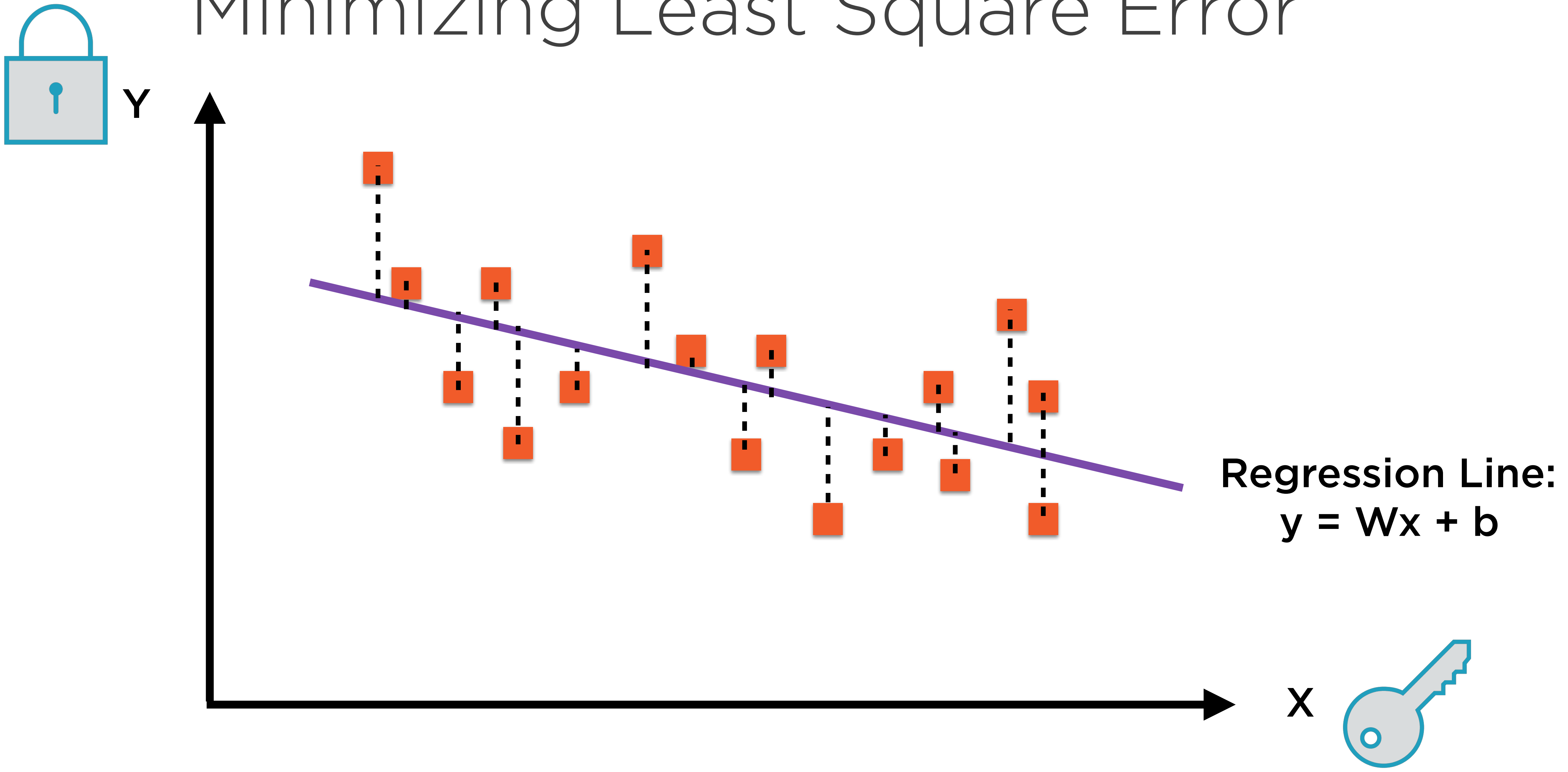


Minimizing Least Square Error



The "best fit" line is called the regression line

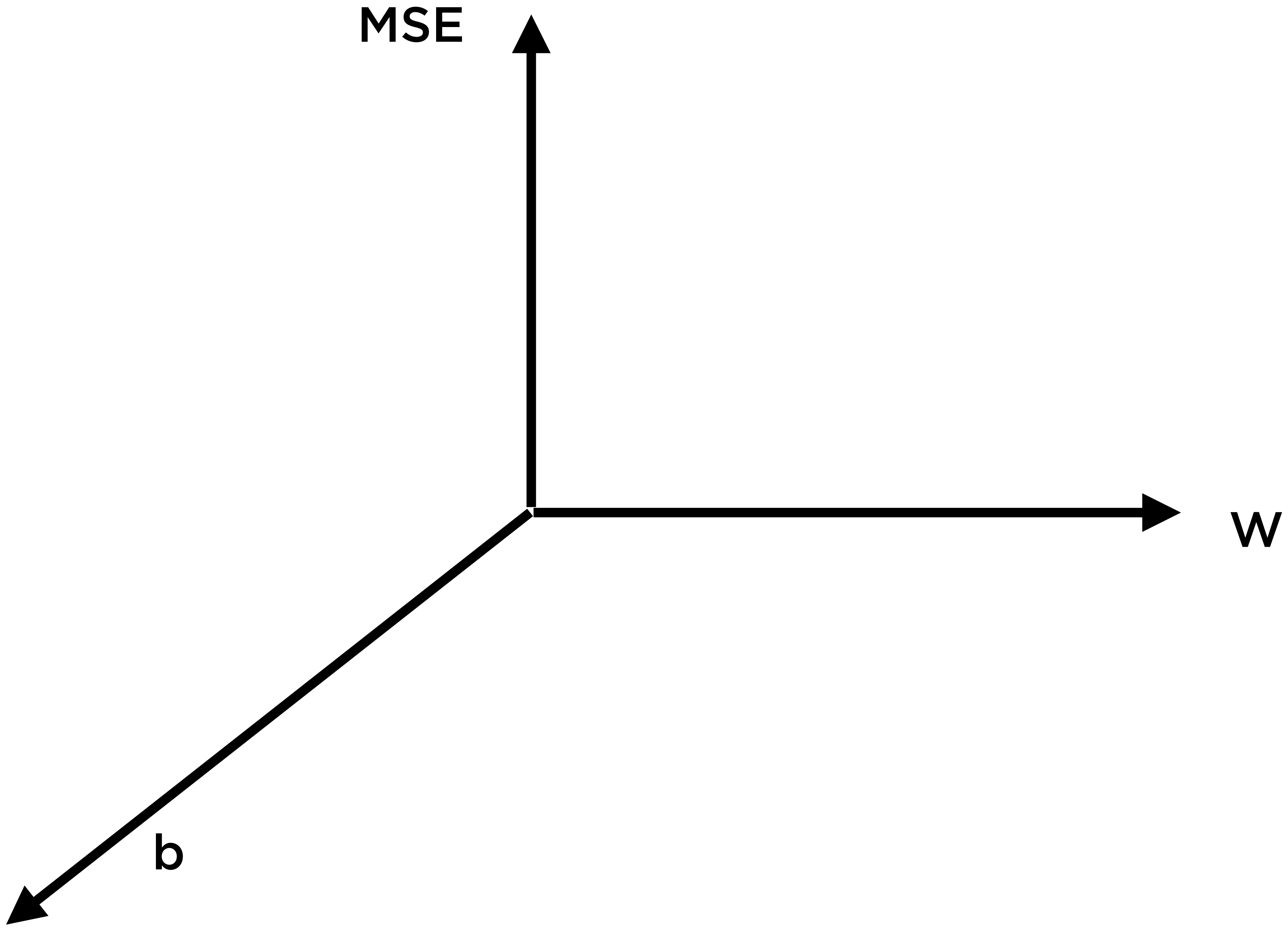
Minimizing Least Square Error



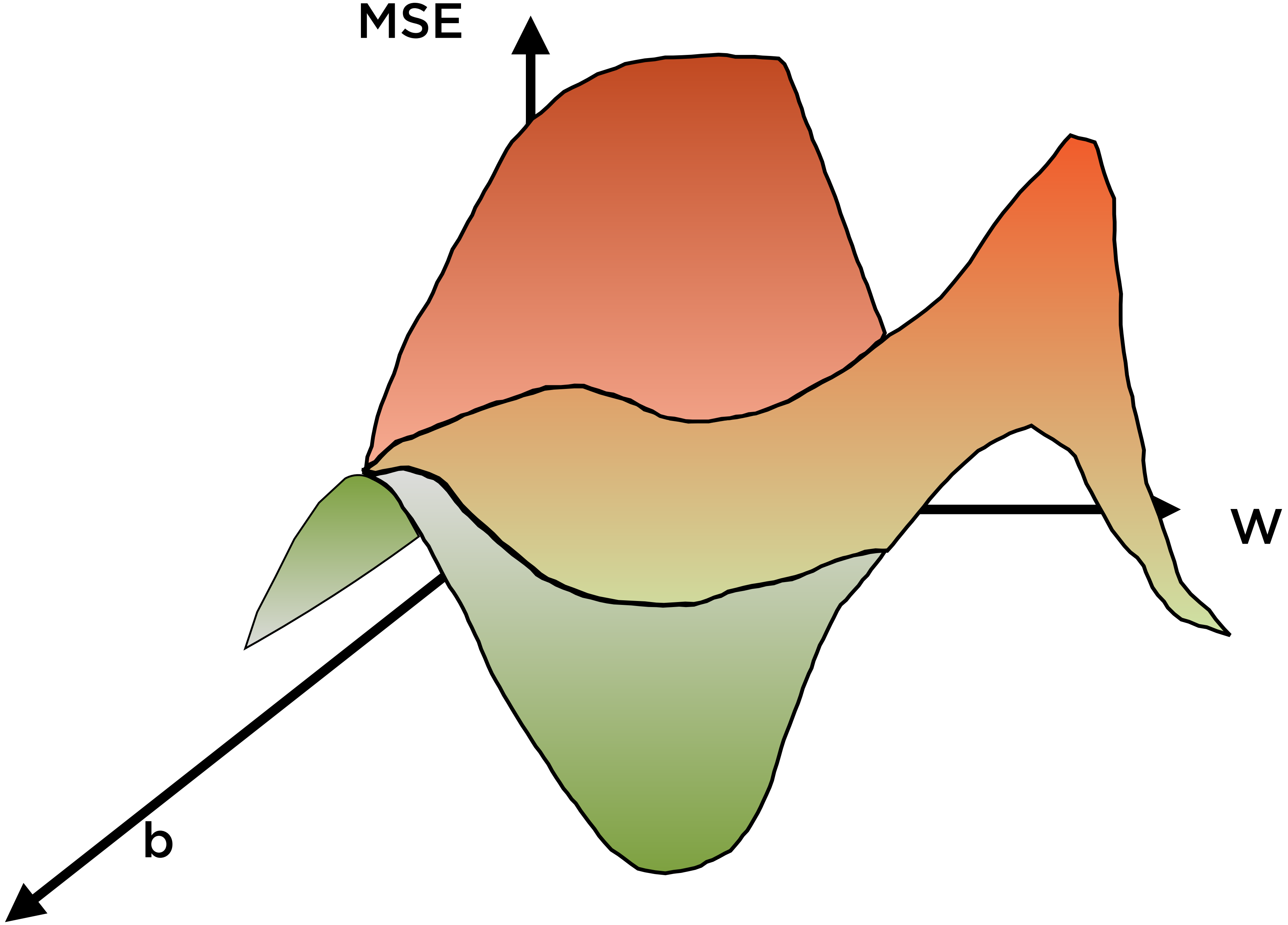
Minimize the sum of the squares of the distances of the points from the regression line

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

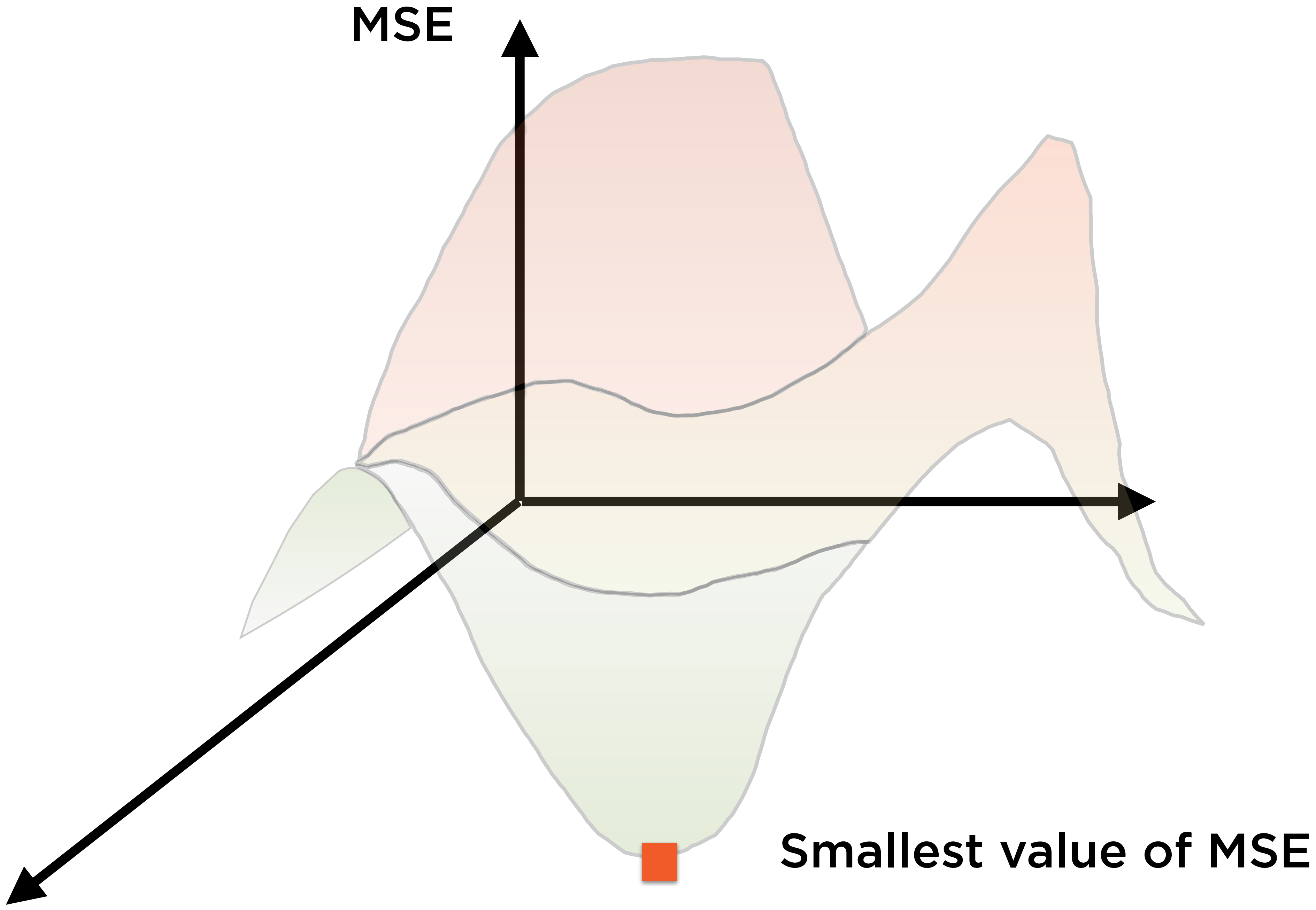
Minimizing MSE



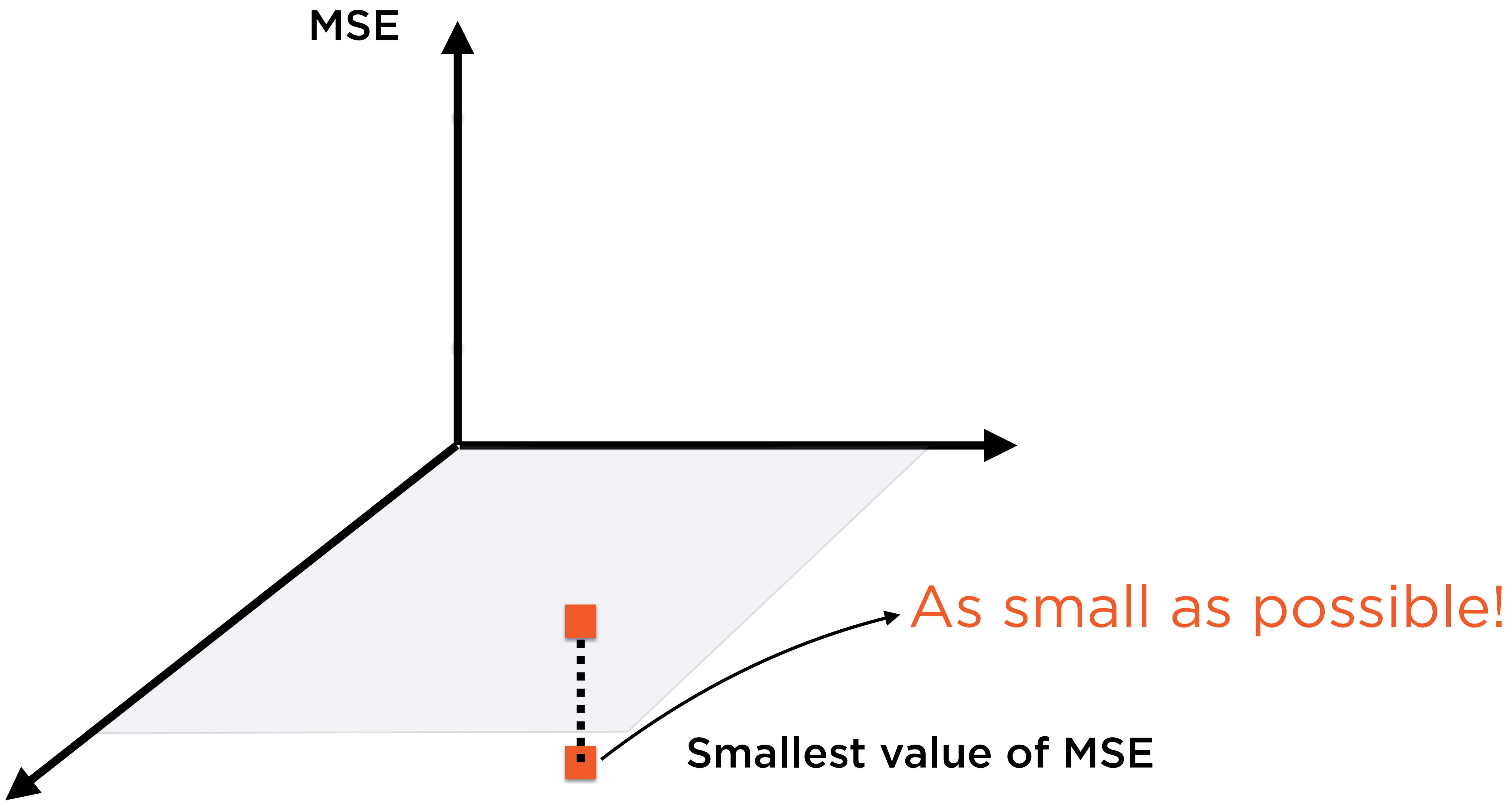
Minimizing MSE



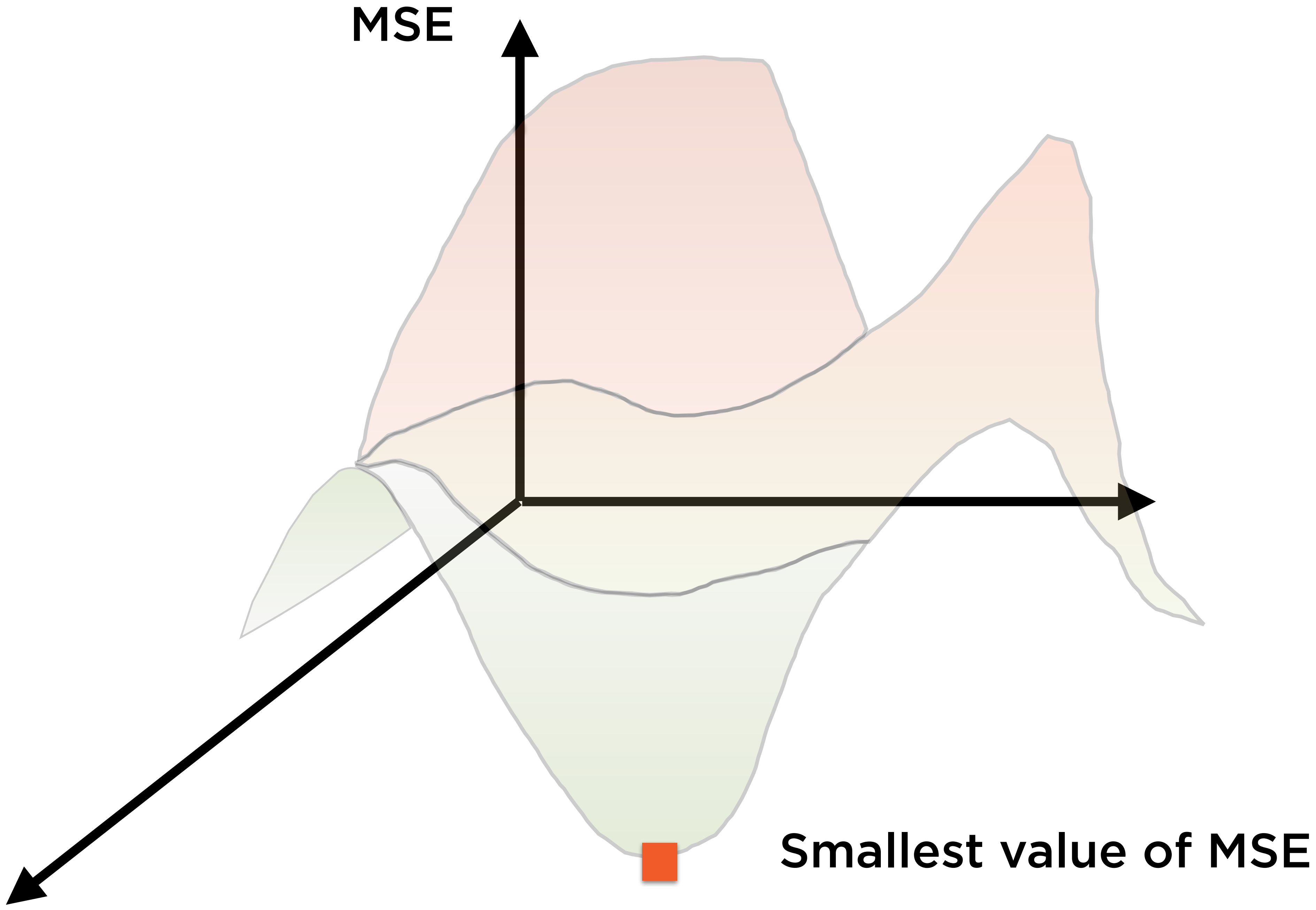
Minimizing MSE



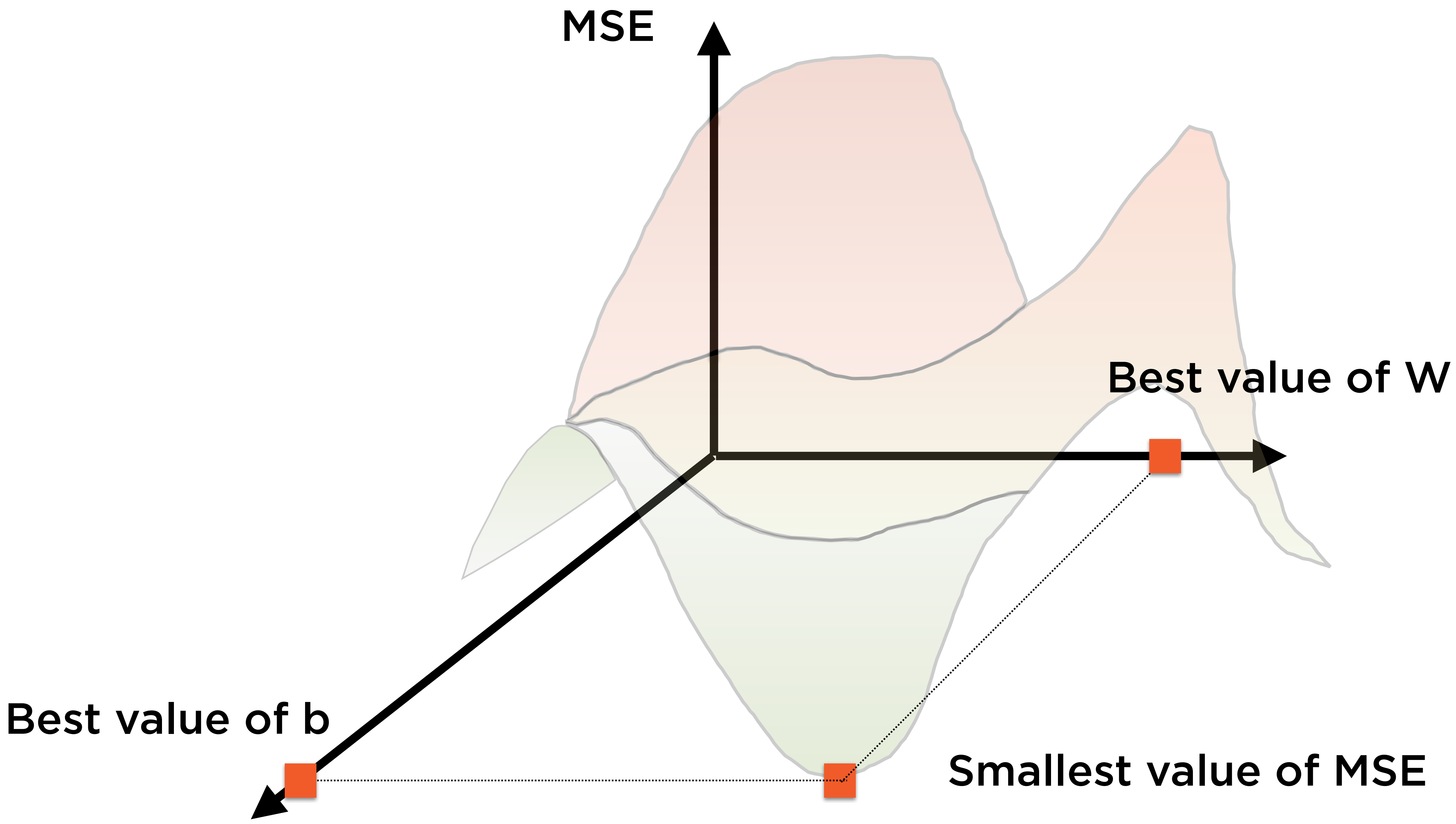
Minimizing MSE



Minimizing MSE

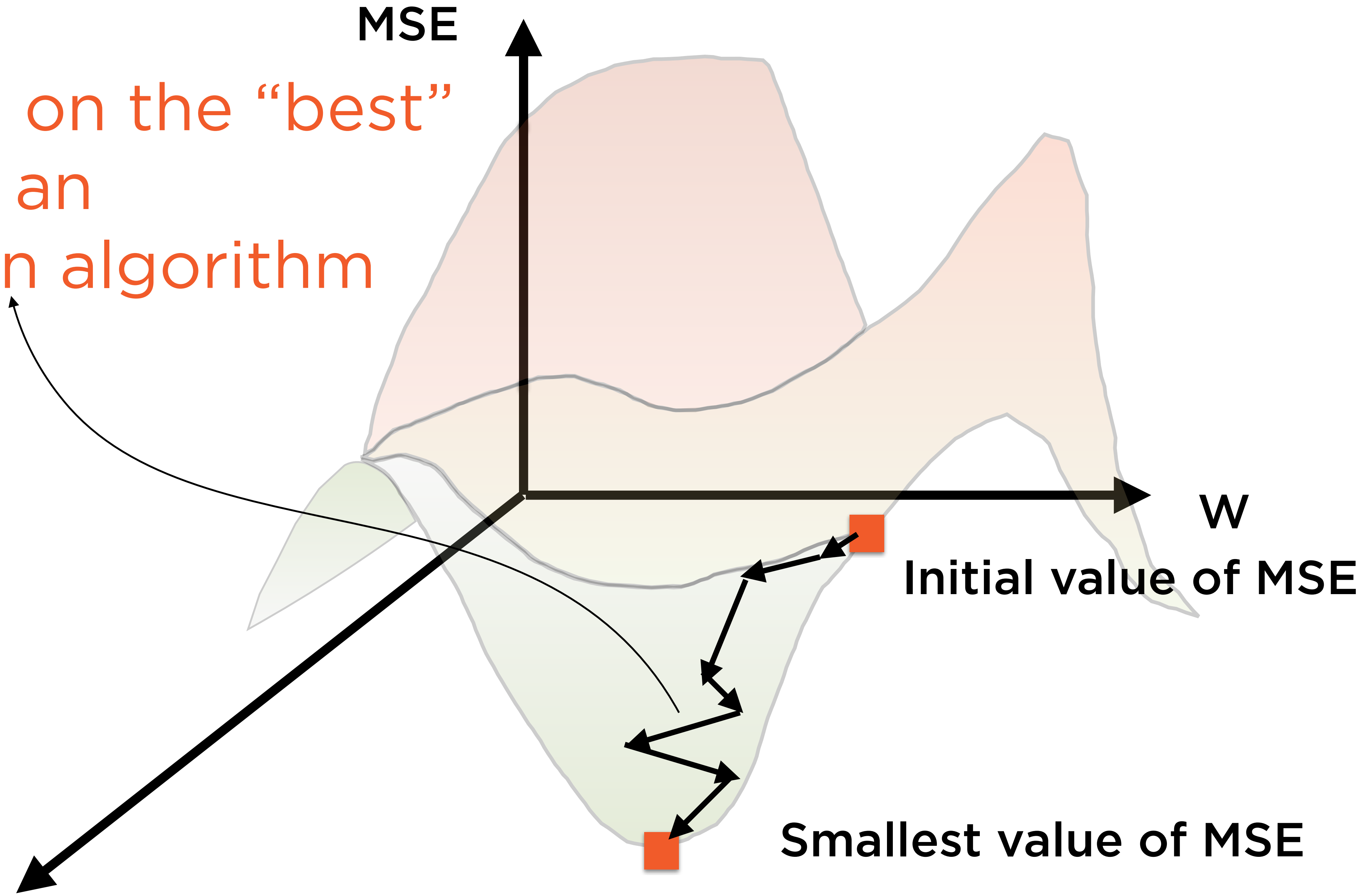


Minimizing MSE

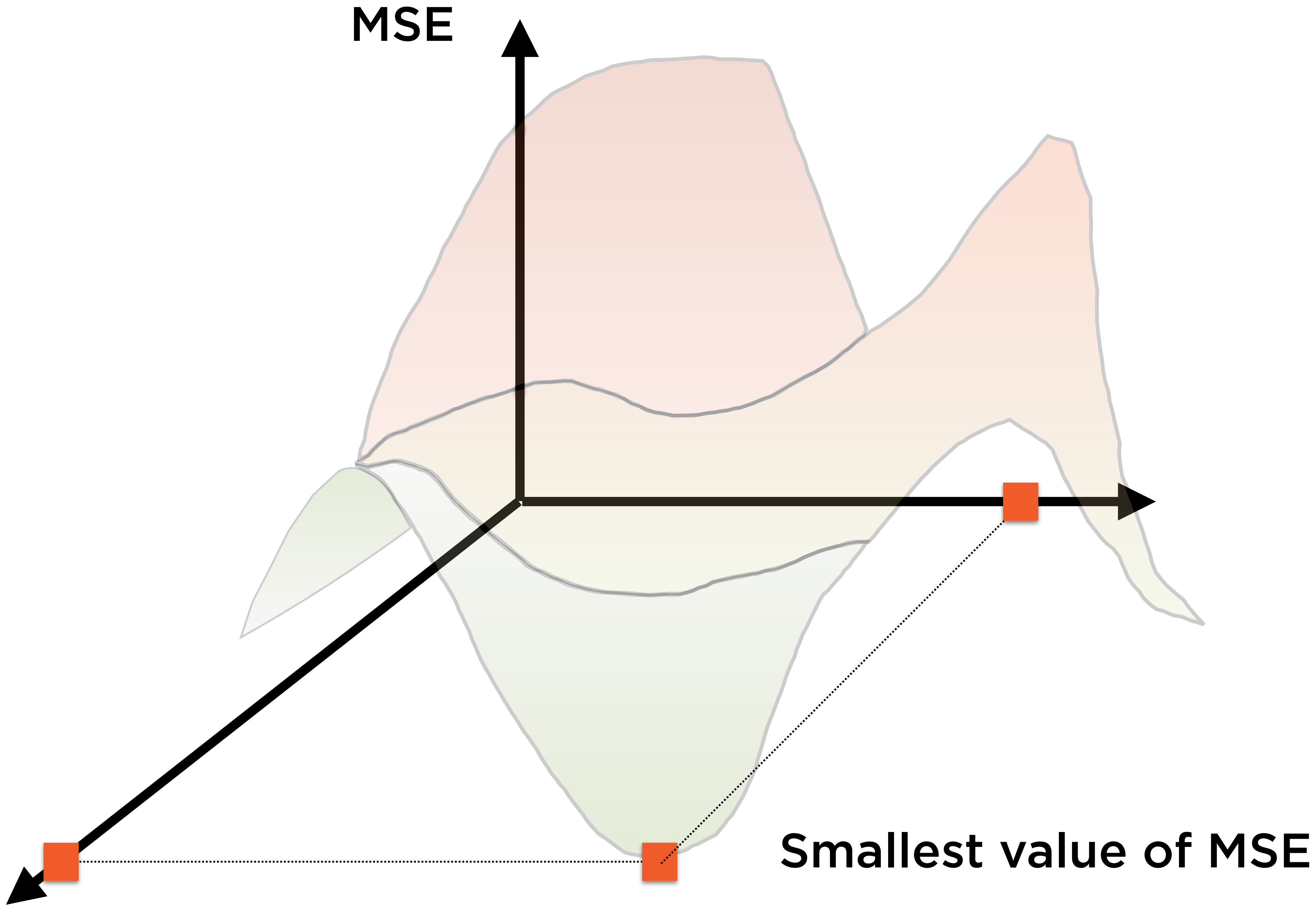


Gradient Descent

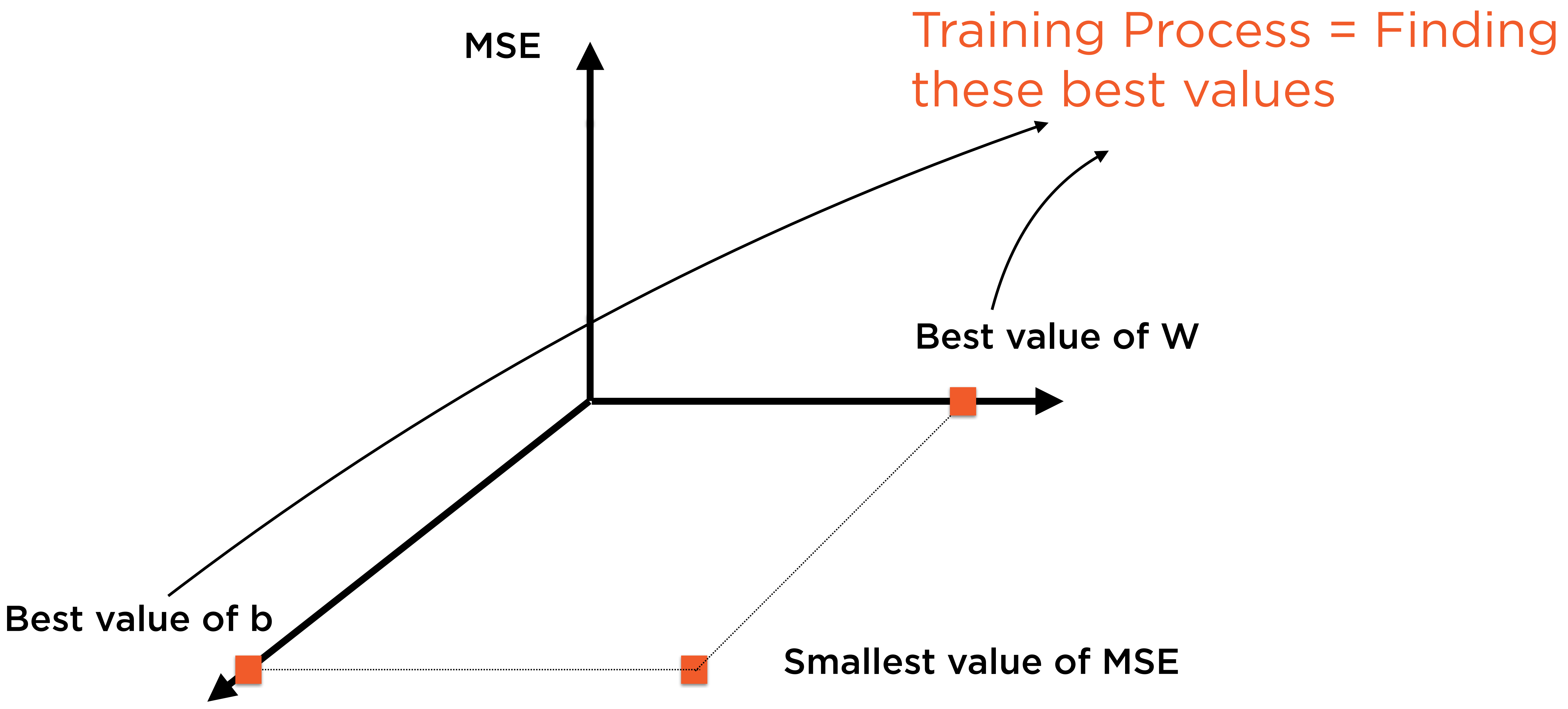
Converging on the “best” value using an optimization algorithm



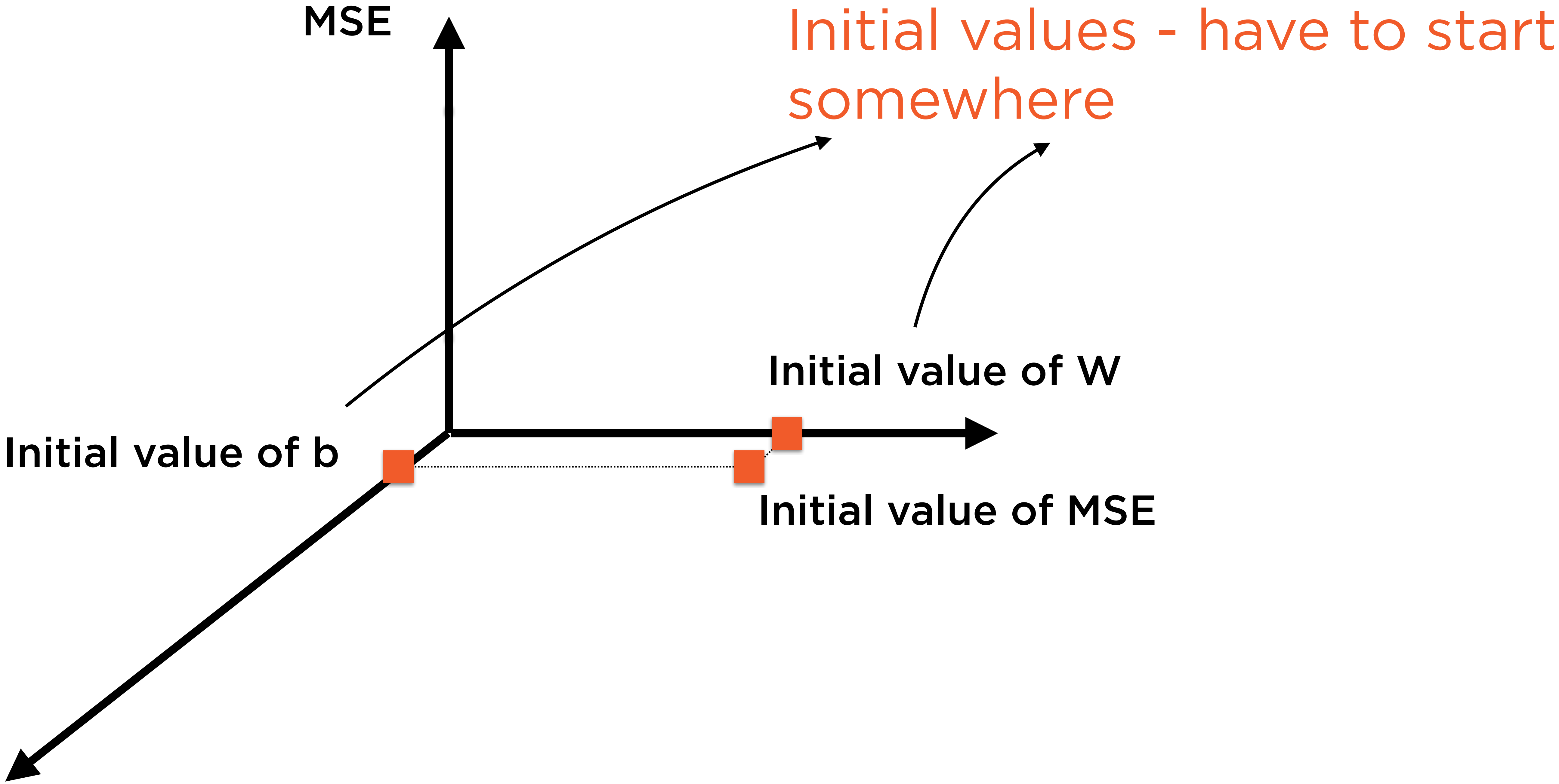
Minimizing MSE



Training the Algorithm

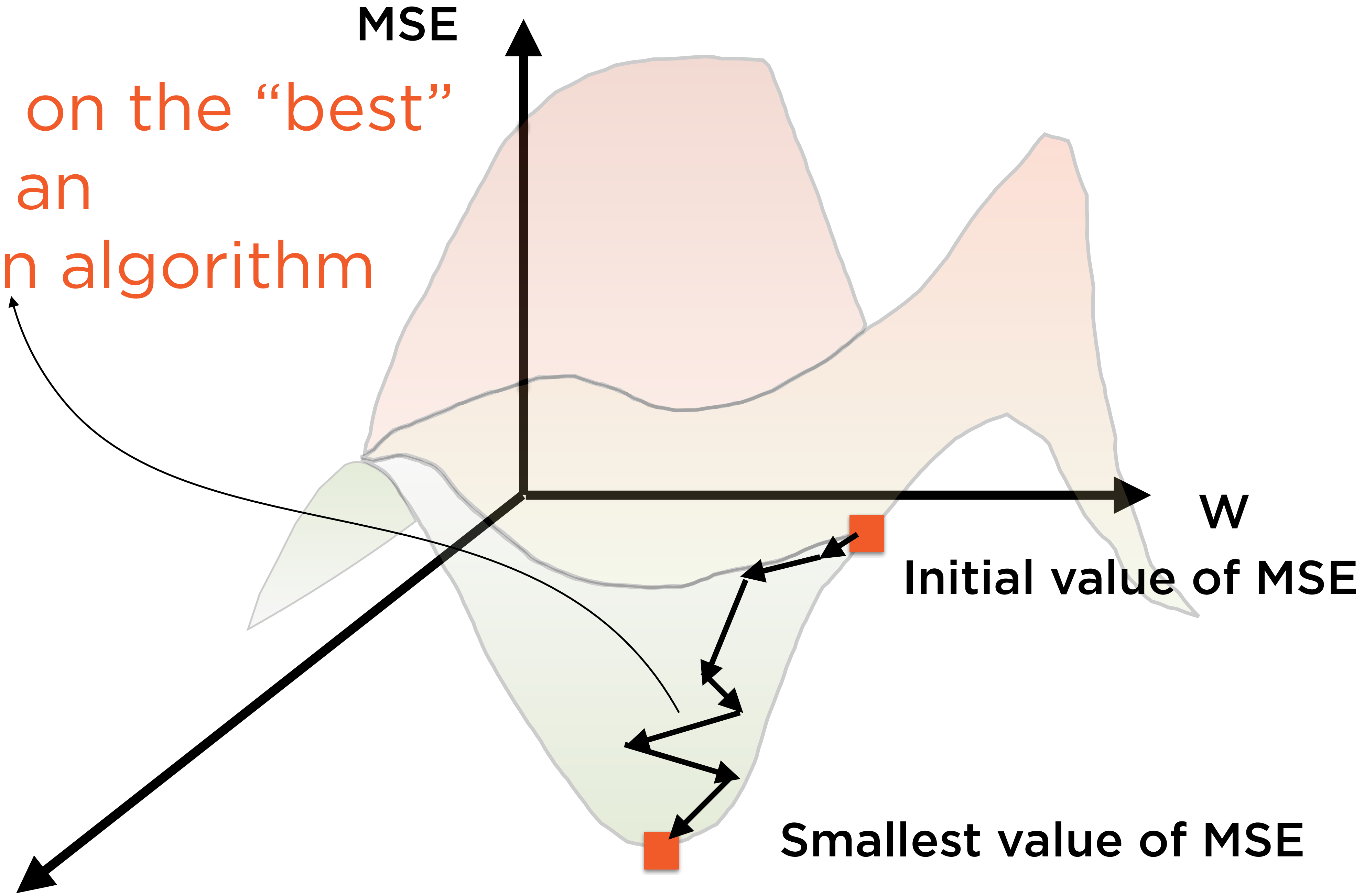


Start Somewhere



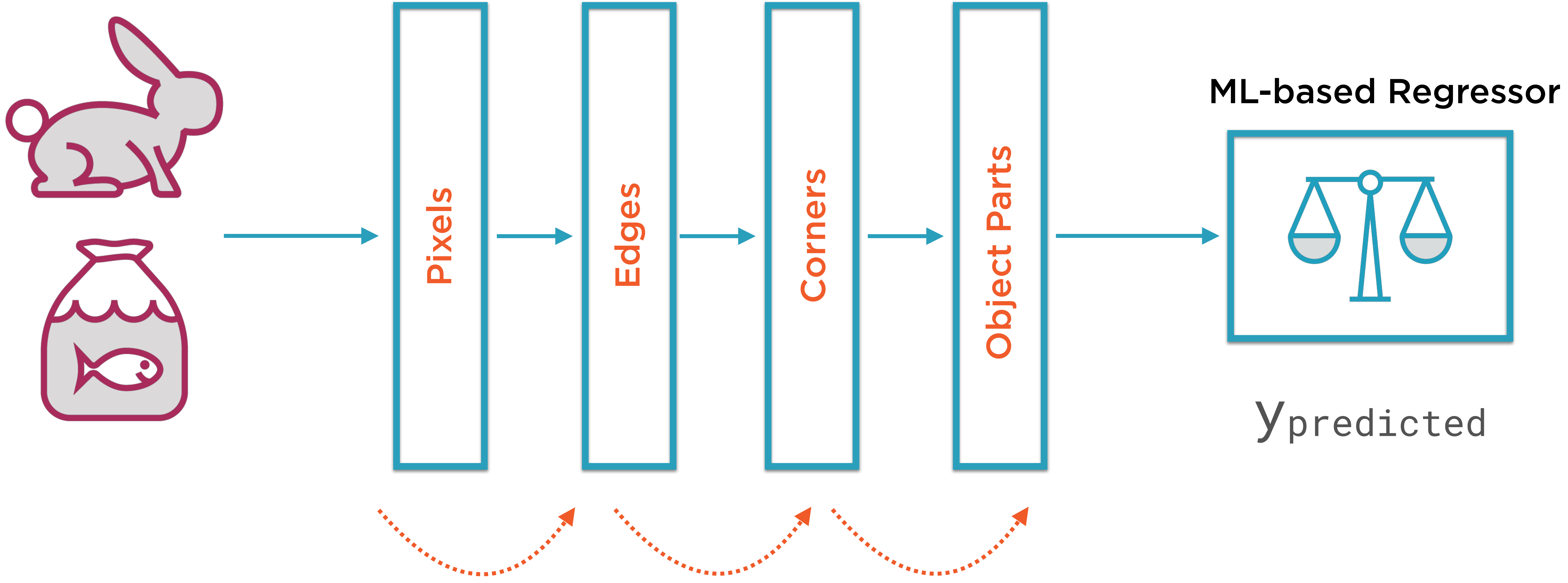
Gradient Descent

Converging on the “best” value using an optimization algorithm



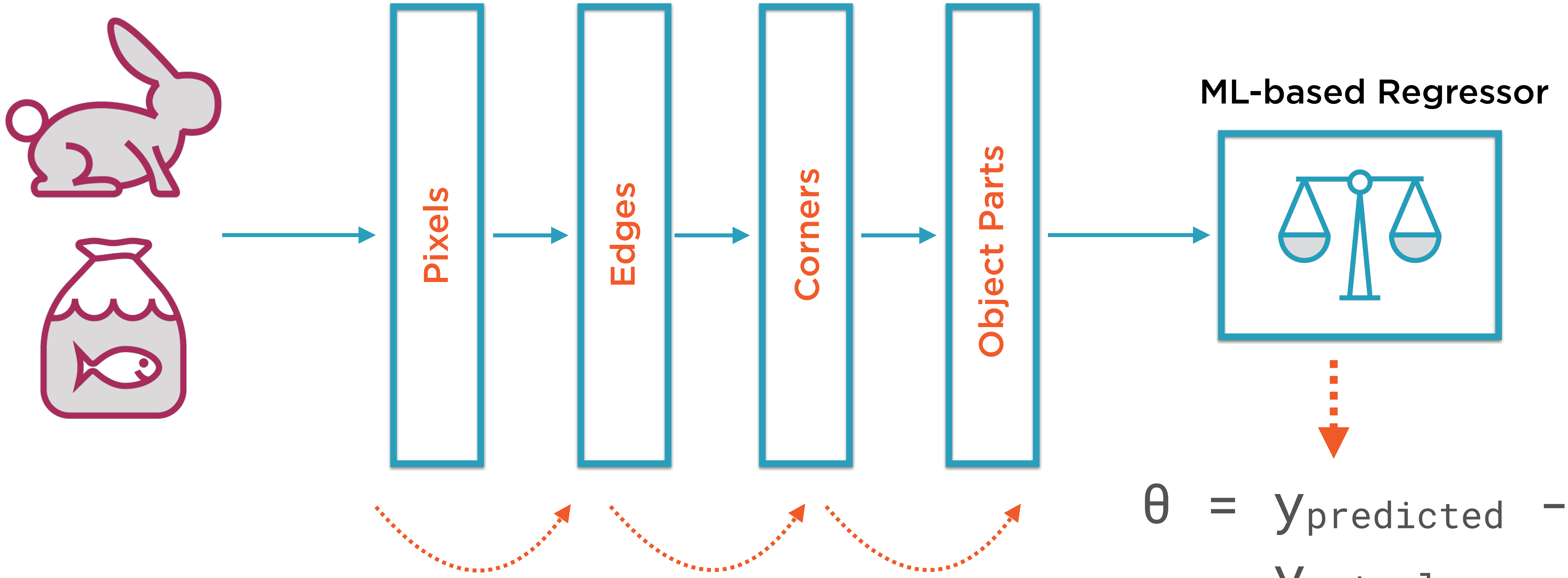
Forward and Backward Passes

Forward Pass



Use the current model weights and biases to make a prediction

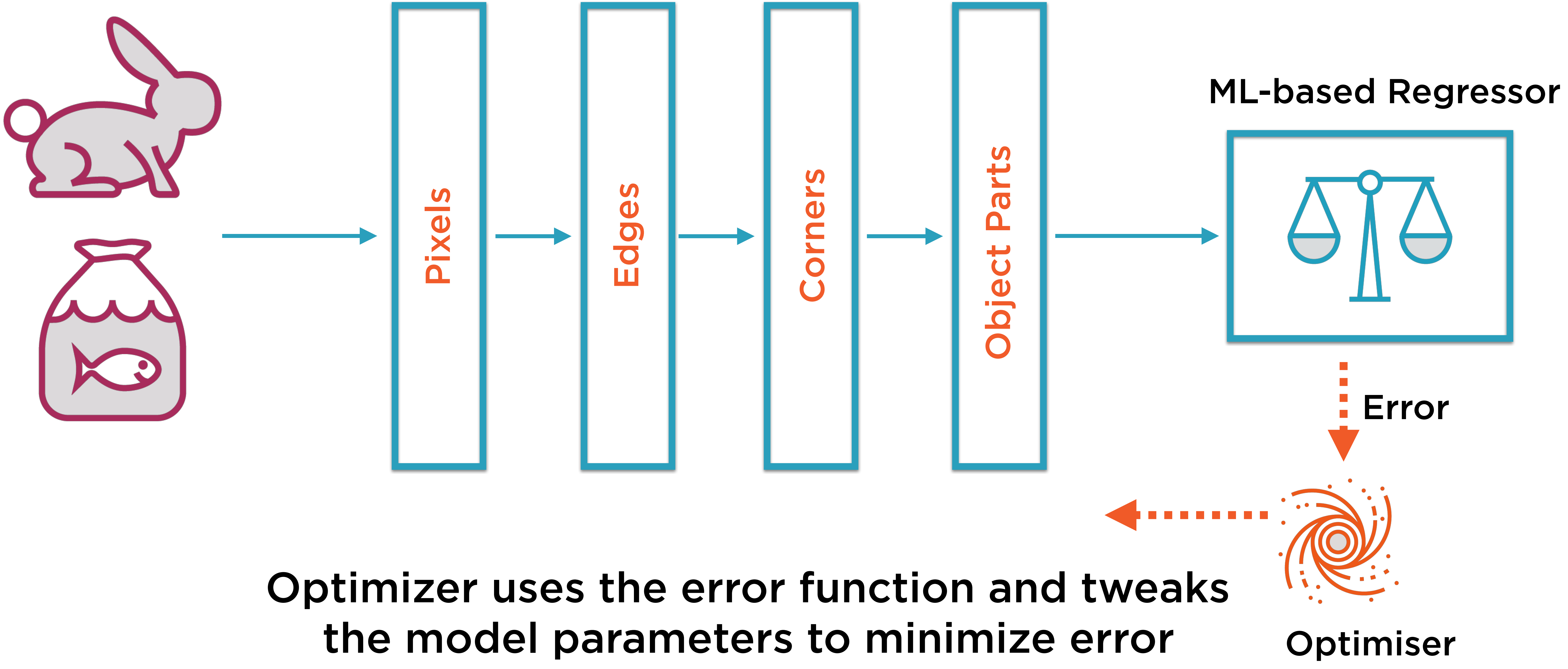
Forward Pass



$\theta = y_{predicted} - y_{actual}$

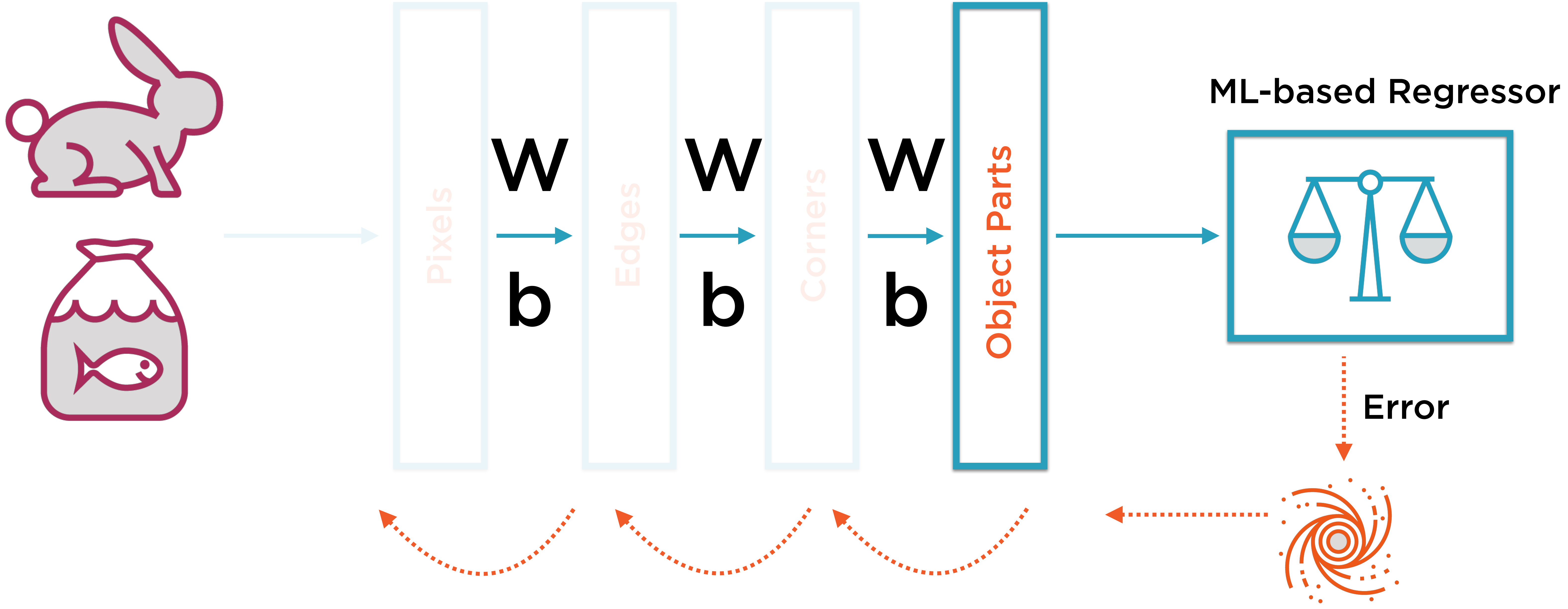
Compare actual values with predicted values and calculate the error

Optimizer Calculates Gradients



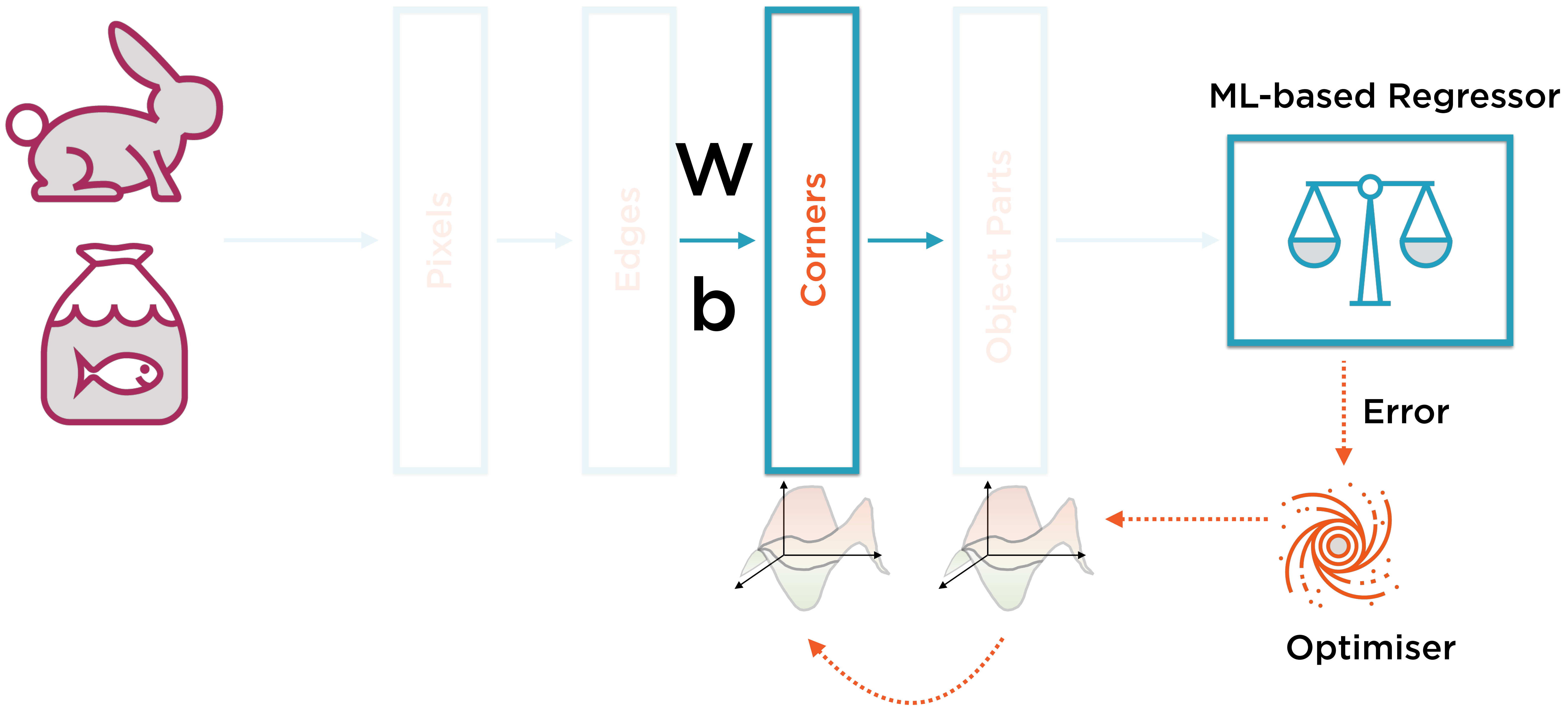
Optimizer uses the error function and tweaks the model parameters to minimize error

Backward Pass

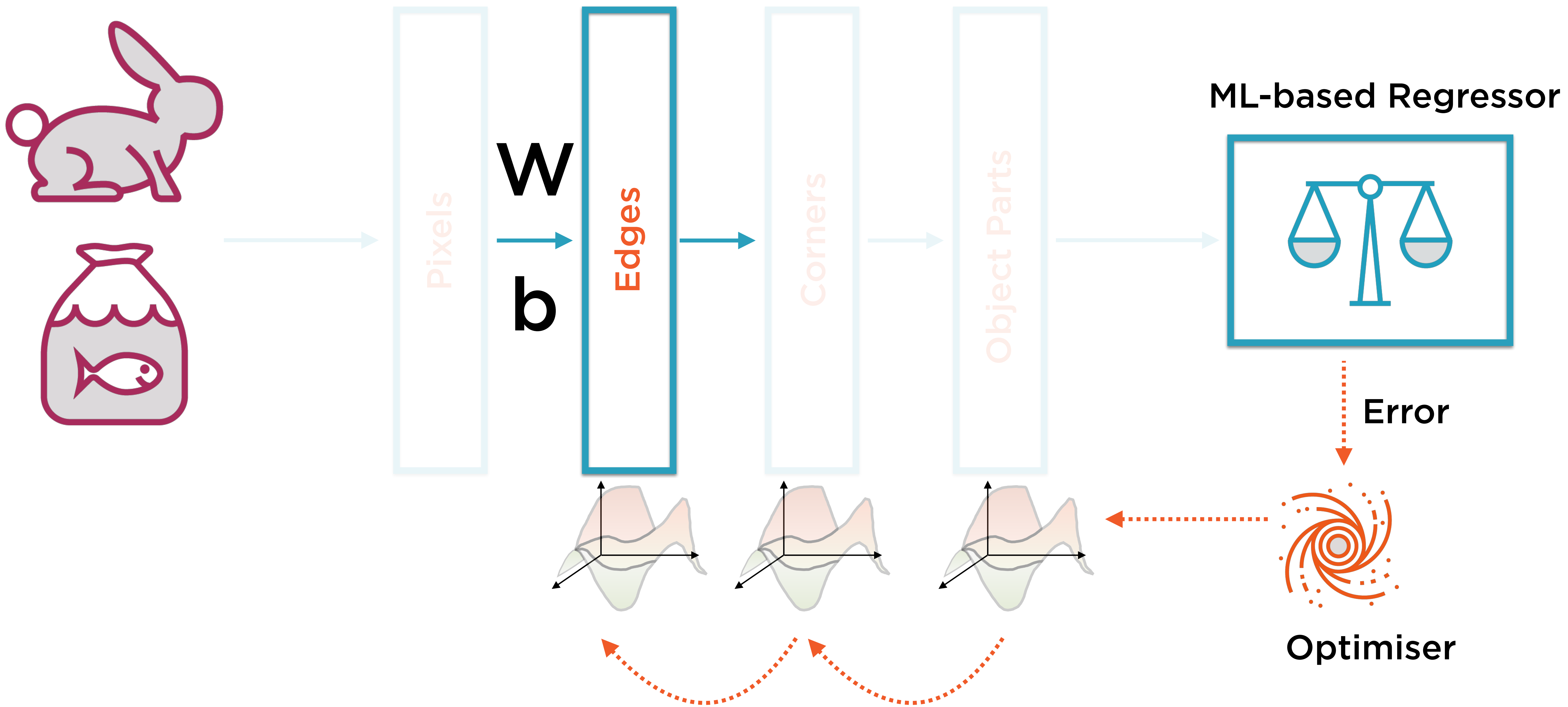


Backward pass: Update parameter values

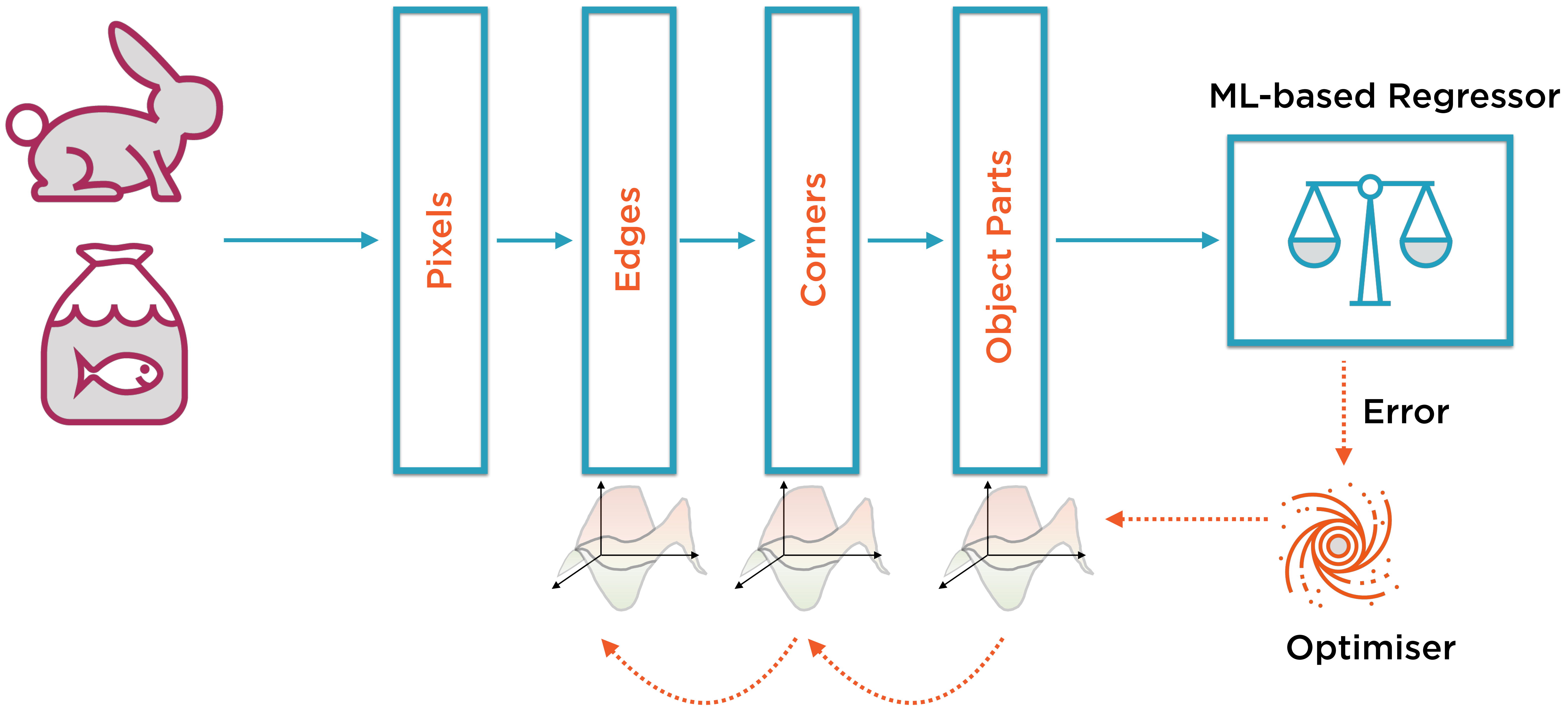
Backward Pass



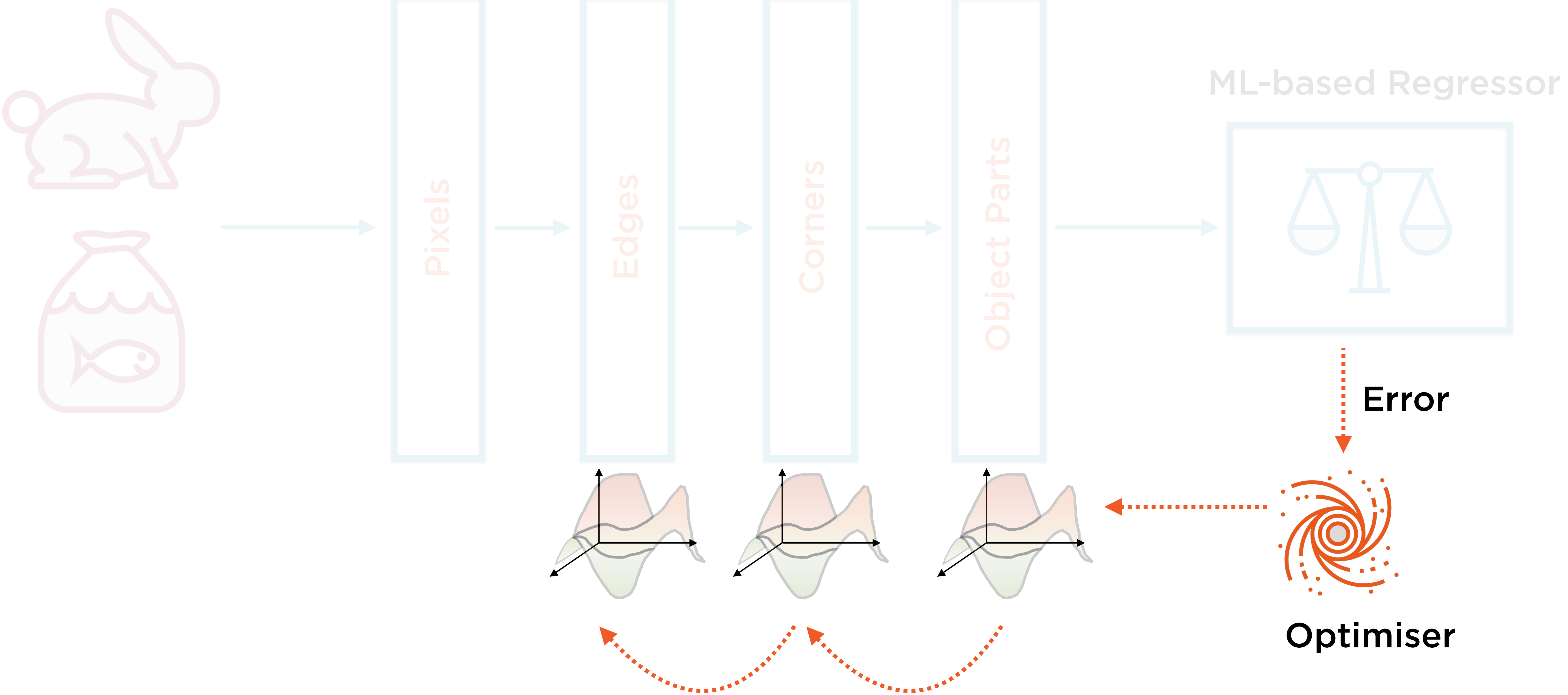
Backward Pass



Backward Pass



Backward Pass



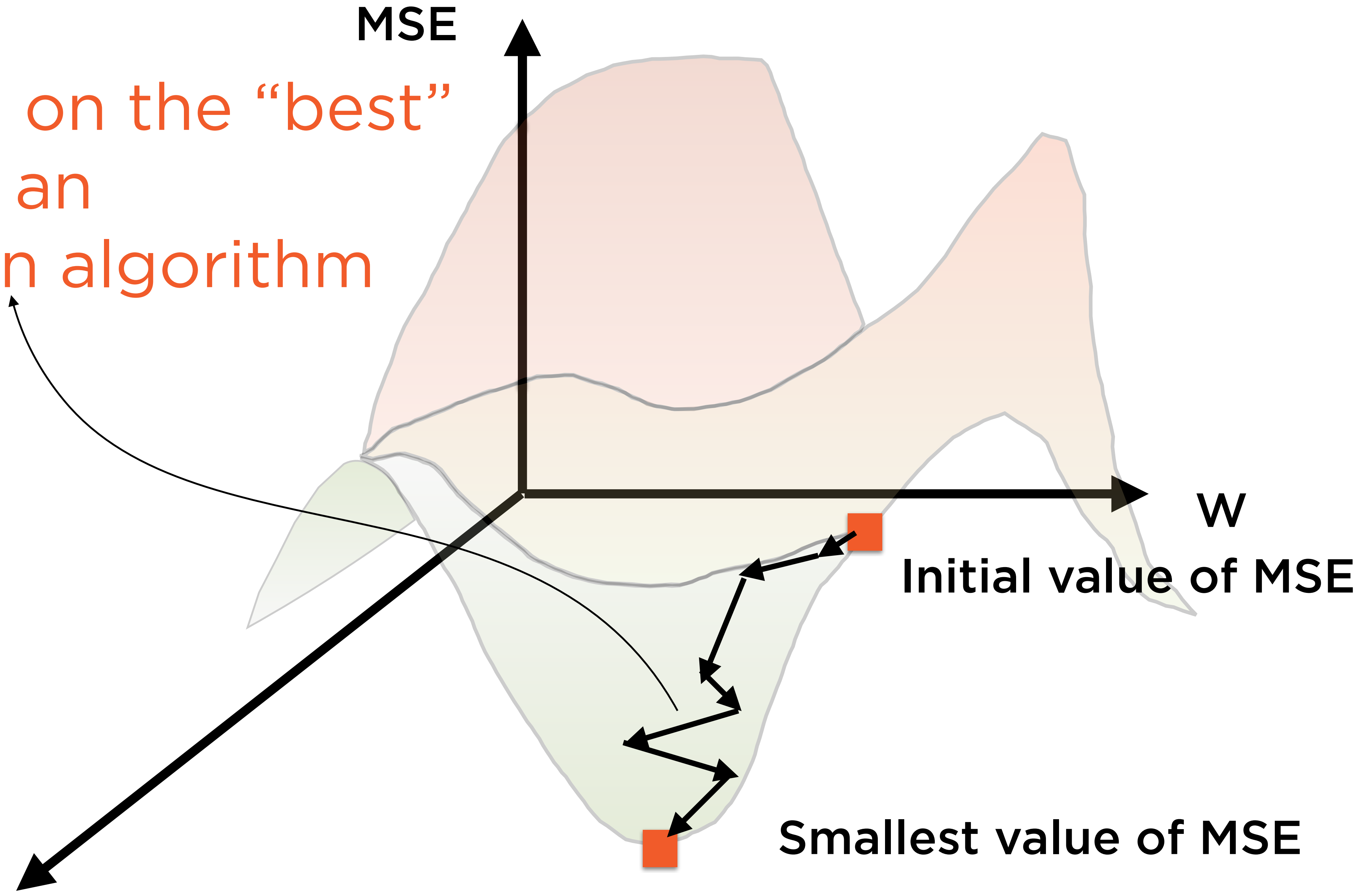
The backward pass allows the weights and biases of the neurons to converge to their final values

Training Using Autograd in PyTorch

Training a neural network uses Gradient Descent to find the weights of the model parameters

Gradient Descent

Converging on the “best” value using an optimization algorithm



$$MSE = \text{Mean Square Error of Loss}$$
$$Loss = \theta = y_{predicted} - y_{actual}$$

MSE

Mean Square Error (MSE) is the metric to be minimized during training of regression model

Given x, model outputs predicted value of y

$$\text{Loss} = \theta = y_{predicted} - y_{actual}$$

Loss Function θ

Loss function measures inaccuracy of model on a specific instance

Actual label, available in training data

$$\text{Loss} = \theta = y_{predicted} - y_{actual}$$

Loss Function θ

Loss function measures inaccuracy of model on a specific instance

In NN with 1 Neuron:

$$\text{Gradient}(\theta) = \nabla\theta(W_1, b_1) = (\partial\theta/\partial W_1, \partial\theta/\partial b_1)$$

Gradient: Vector of Partial Derivatives

For a function $y = f(x_1, x_2, x_3)$, the Greek character "nabla" (∇) denotes the gradient

Partial derivative of loss w.r.t to parameter W

Holding all other parameters and the input constant - **how much does loss change when you change W**

In NN with 1 Neuron:

$$\text{Gradient}(\theta) = \nabla\theta(W_1, b_1) = (\partial\theta/\partial W_1, \partial\theta/\partial b_1)$$

Gradient: Vector of Partial Derivatives

For a function $y = f(x_1, x_2, x_3)$, the Greek character "nabla" (∇) denotes the gradient

In NN with 1 Neuron:

$$\text{Gradient}(\theta) = \nabla\theta(W_1, b_1) = (\partial\theta/\partial W_1, \partial\theta/\partial b_1)$$

Gradient: Vector of Partial Derivatives

For a function $y = f(x_1, x_2, x_3)$, the Greek character "nabla" (∇) denotes the gradient

In NN with 1 Neuron:

$$\text{Gradient}(\theta) = \nabla\theta(W_1, b_1) = (\partial\theta/\partial W_1, \partial\theta/\partial b_1)$$

Gradient Descent to Minimize Loss

Find values of W_1, b_1 where loss has "lowest" gradient - i.e. minimize gradient of θ

```
Condition of minimum:

Gradient(θ) = ∇θ(W1, b1) = (∂θ/∂W1, ∂θ/∂b1) = zero
```

Gradient Descent to Minimize Loss

Find values of W_1, b_1 where loss has “lowest” gradient - i.e. minimize gradient of θ

```
In NN with 1 Neuron:
```

$$\text{Gradient}(\theta) = \nabla\theta(W_1, b_1) = (\partial\theta/\partial W_1, \partial\theta/\partial b_1)$$

Gradient Descent to Minimize Loss

Find values of W_1, b_1 where loss has “lowest” gradient - i.e. minimize gradient of θ

In NN with 10,000 Neurons:

$$\text{Gradient}(\theta) = \nabla\theta(W_1,\ b_1 \ldots W_{10000},\ b_{10000})$$
$$= (\partial\theta/\partial W_1,\ \partial\theta/\partial b_1,\ \ldots \partial\theta/\partial W_{10000},\ \partial\theta/\partial b_{10000})$$

Gradient Descent for Complex Networks

The gradient vector gets very large for complex networks, need sophisticated math to calculate and optimize

Actually Calculating Gradients

Symbolic Differentiation

Conceptually simple but hard to implement

Numeric Differentiation

Easy to implement but won’t scale

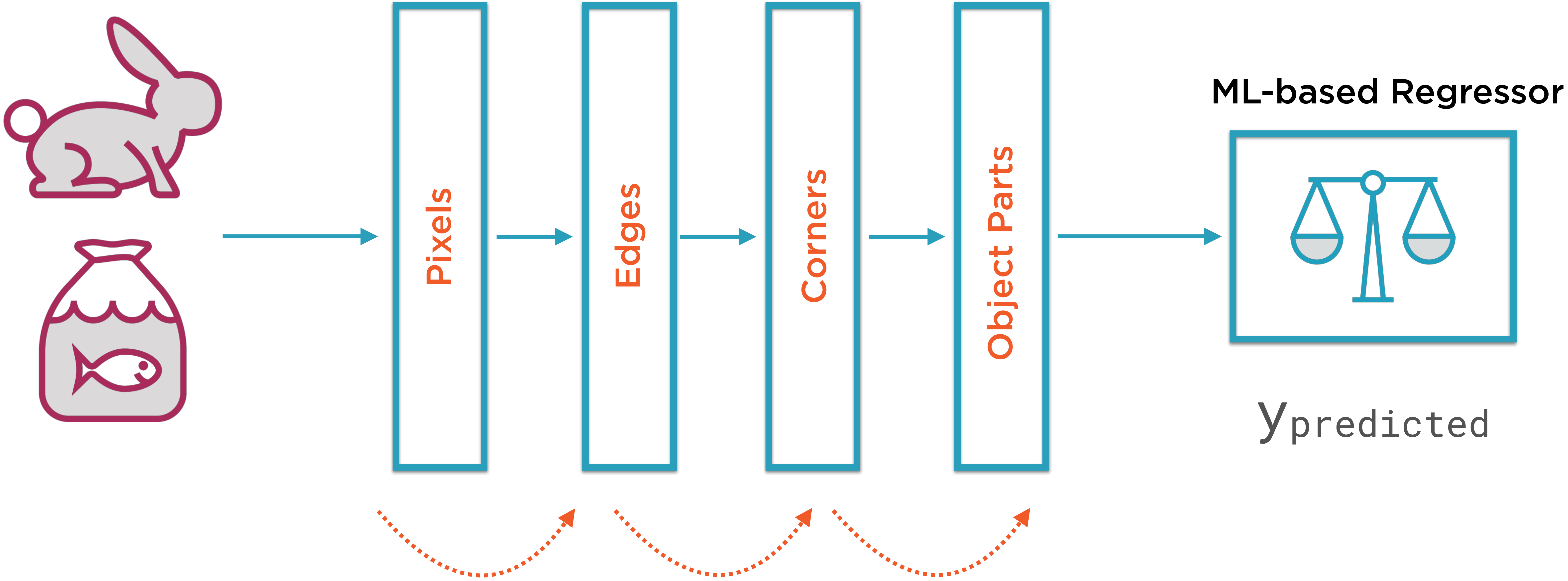
Automatic Differentiation

Conceptually difficult but easy to implement

PyTorch, TensorFlow and other packages rely on automatic differentiation

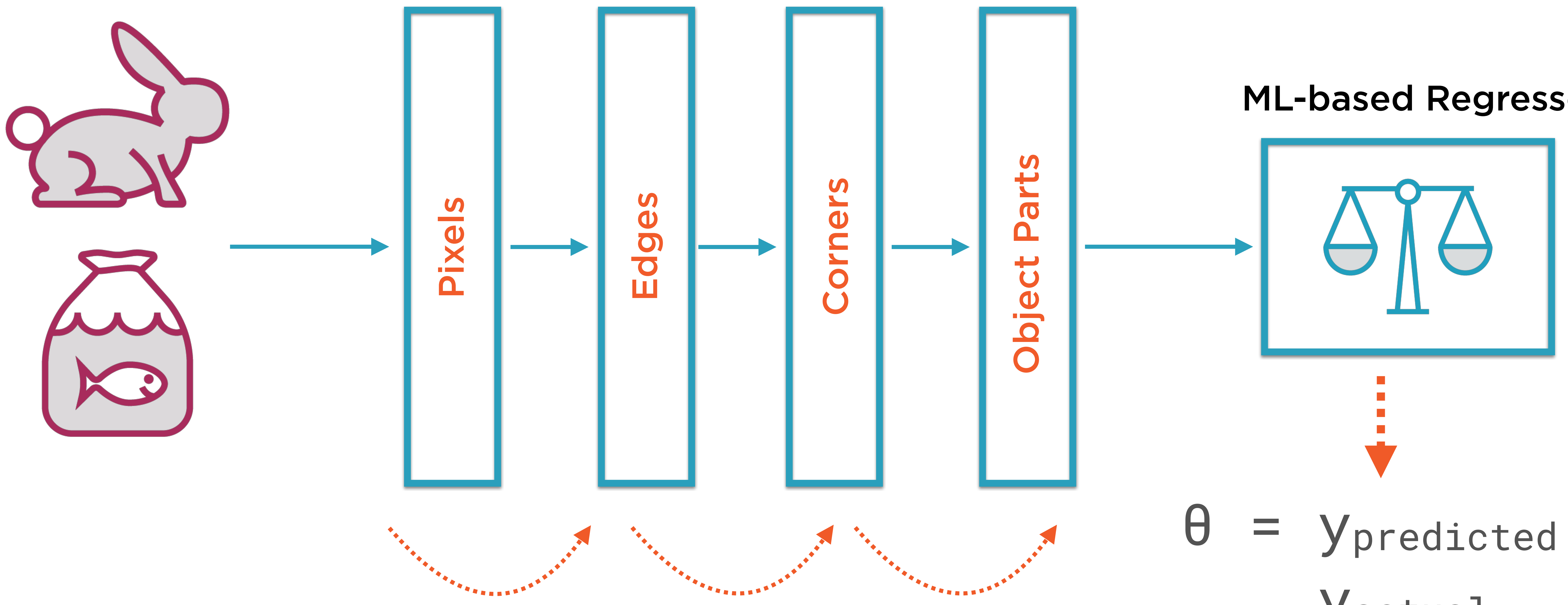
These gradients are used to update the model parameters

Forward Pass



Use the current model weights and biases to make a prediction

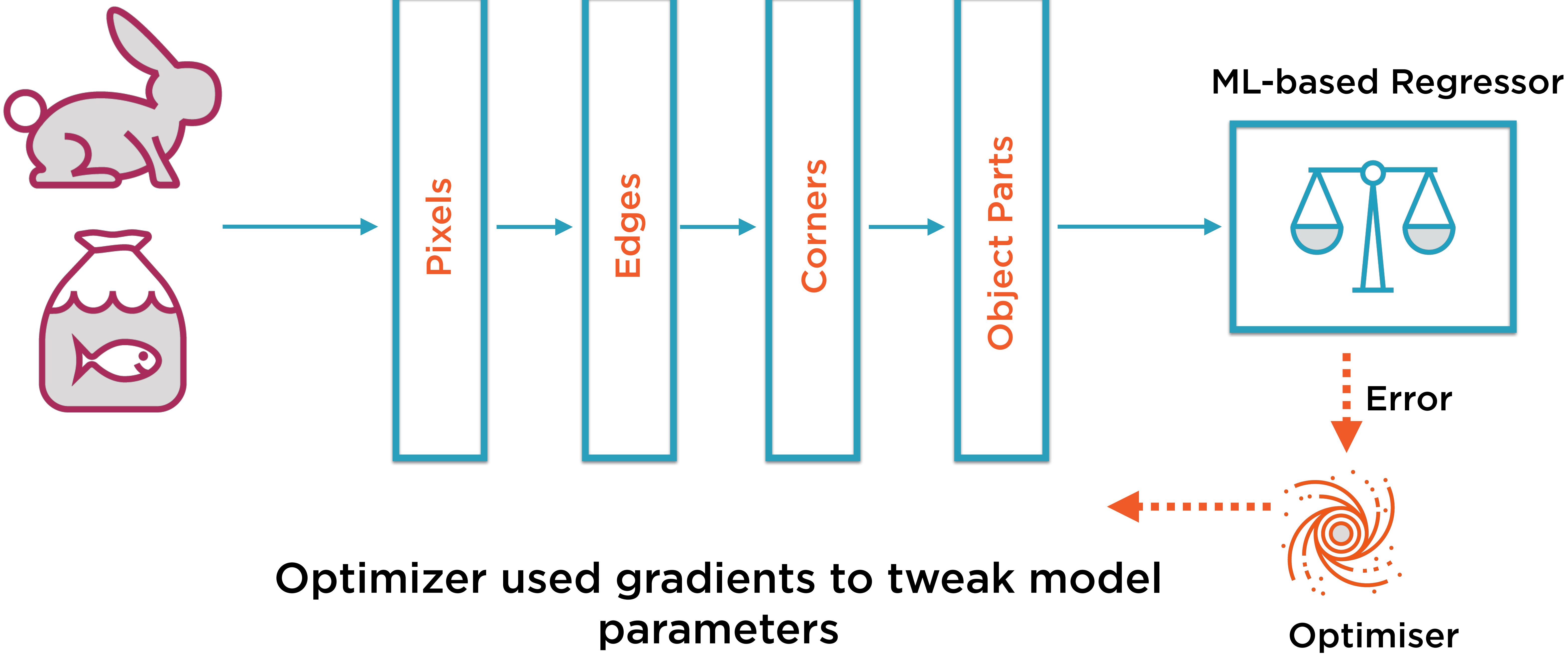
Forward Pass



$\theta = y_{predicted} - y_{actual}$

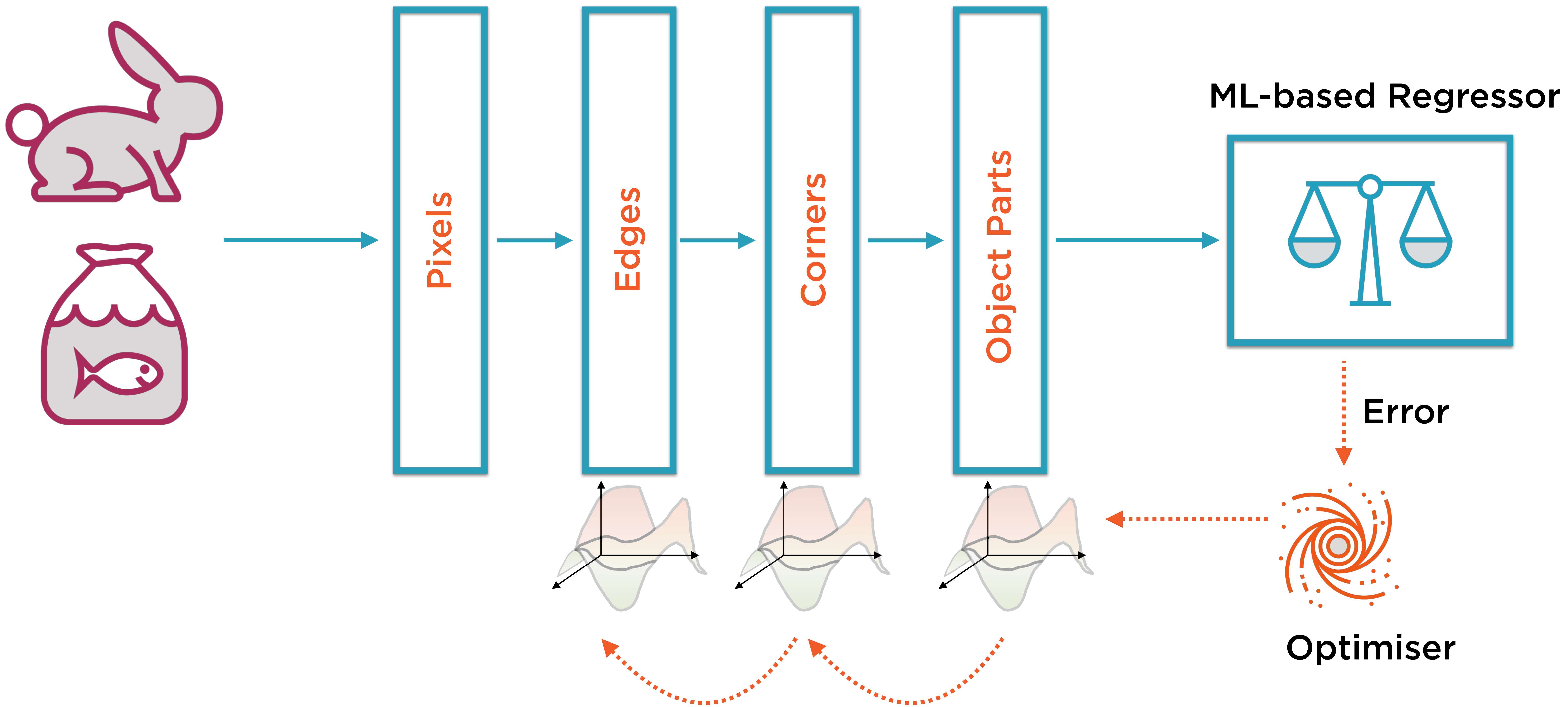
Compare actual values with predicted values and calculate the error

Optimizer Calculates Gradients



Optimizer used gradients to tweak model parameters

Backward Pass



Autograd is the PyTorch package for calculating gradients for back propagation

Back propagation is implemented using a technique called reverse auto-differentiation

$\text{Gradient}(\theta^{t})$

Gradient: Vector of Partial Derivatives

These gradients apply to a specific time t

`Gradient(0)` t

Gradient: Vector of Partial Derivatives

These gradients apply to a specific time t

$$\text{Parameters}^{t+1} = \text{Parameters}^{t} - \text{learning\_rate} \times \text{Gradient}(\theta)^{t}$$

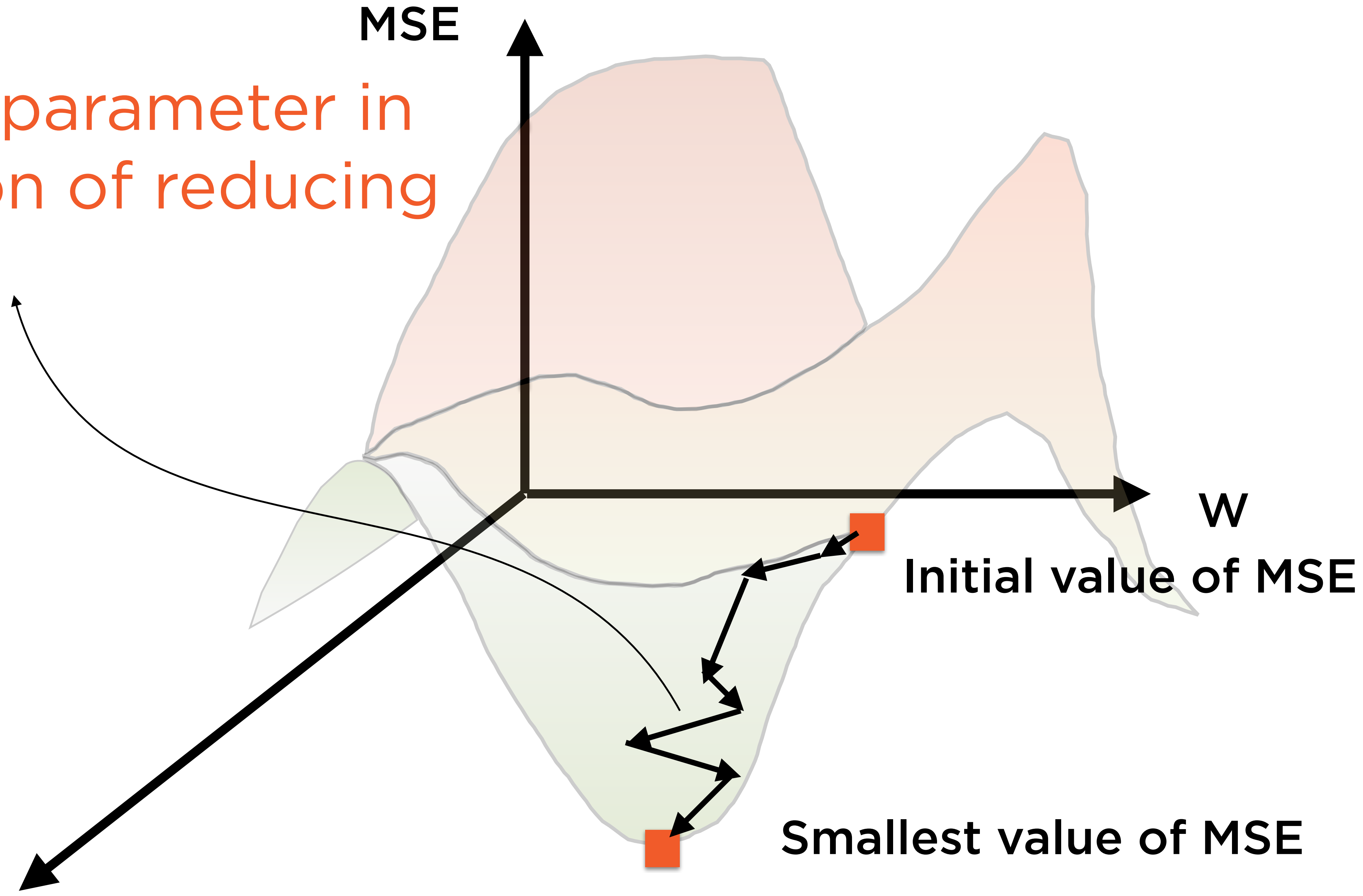
For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

Exact math and mechanics are complex and will vary by optimization algorithm

Gradient Descent

Move each parameter in the direction of reducing gradient



$$\text{Parameters}^{t+1} = \text{Parameters}^{t} - \text{learning\_rate} \times \text{Gradient}(\theta)^{t}$$

For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

$$\text{Parameters}^{t+1} = \text{Parameters}^{t} - \text{learning\_rate} \times \text{Gradient}(\theta)^{t}$$

For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

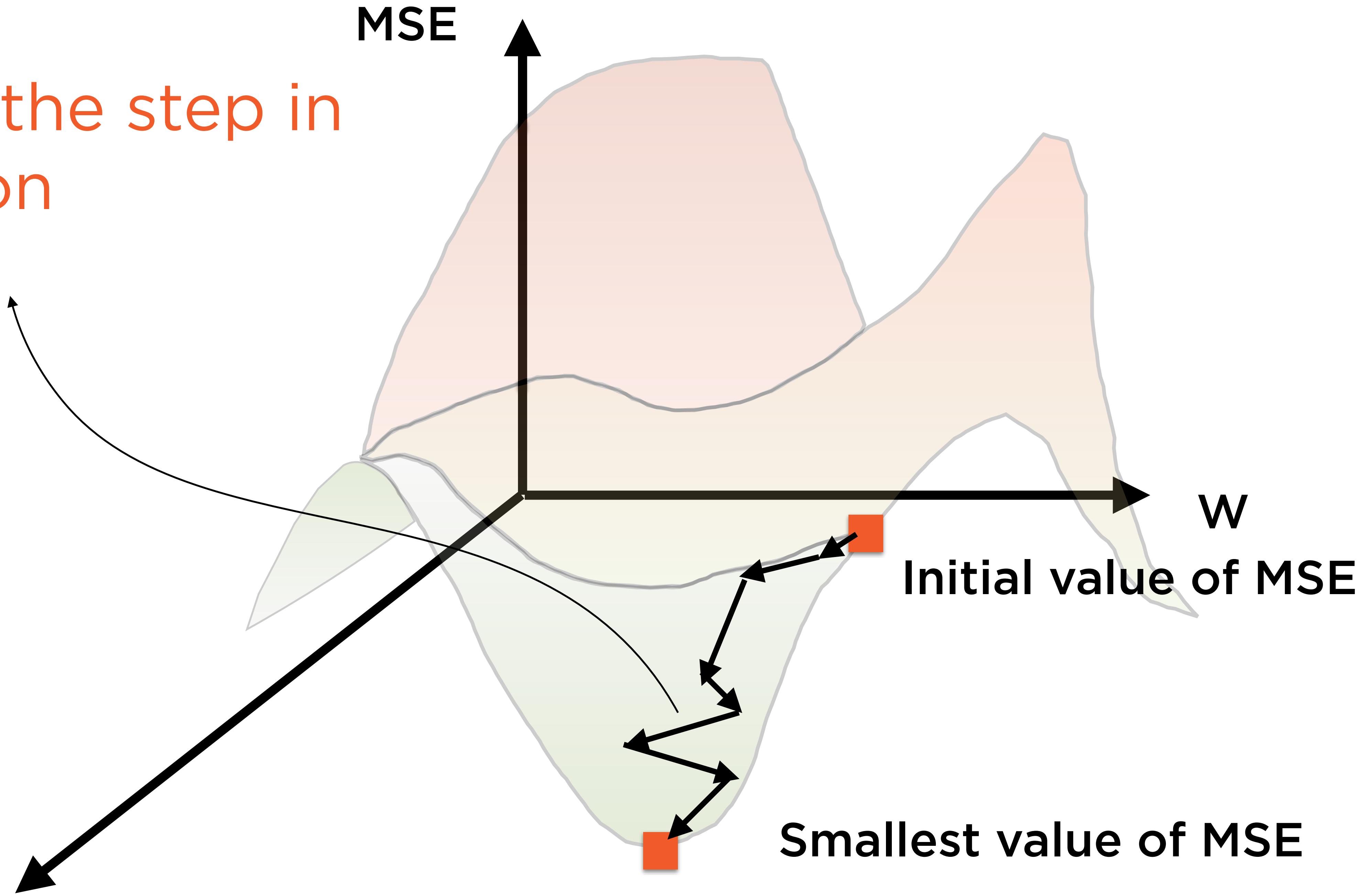
$$\text{Parameters}^{t+1} = \text{Parameters}^{t} - \text{learning\_rate} \times \text{Gradient}(\theta^{t})$$

For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

Learning Rate

The size of the step in this direction



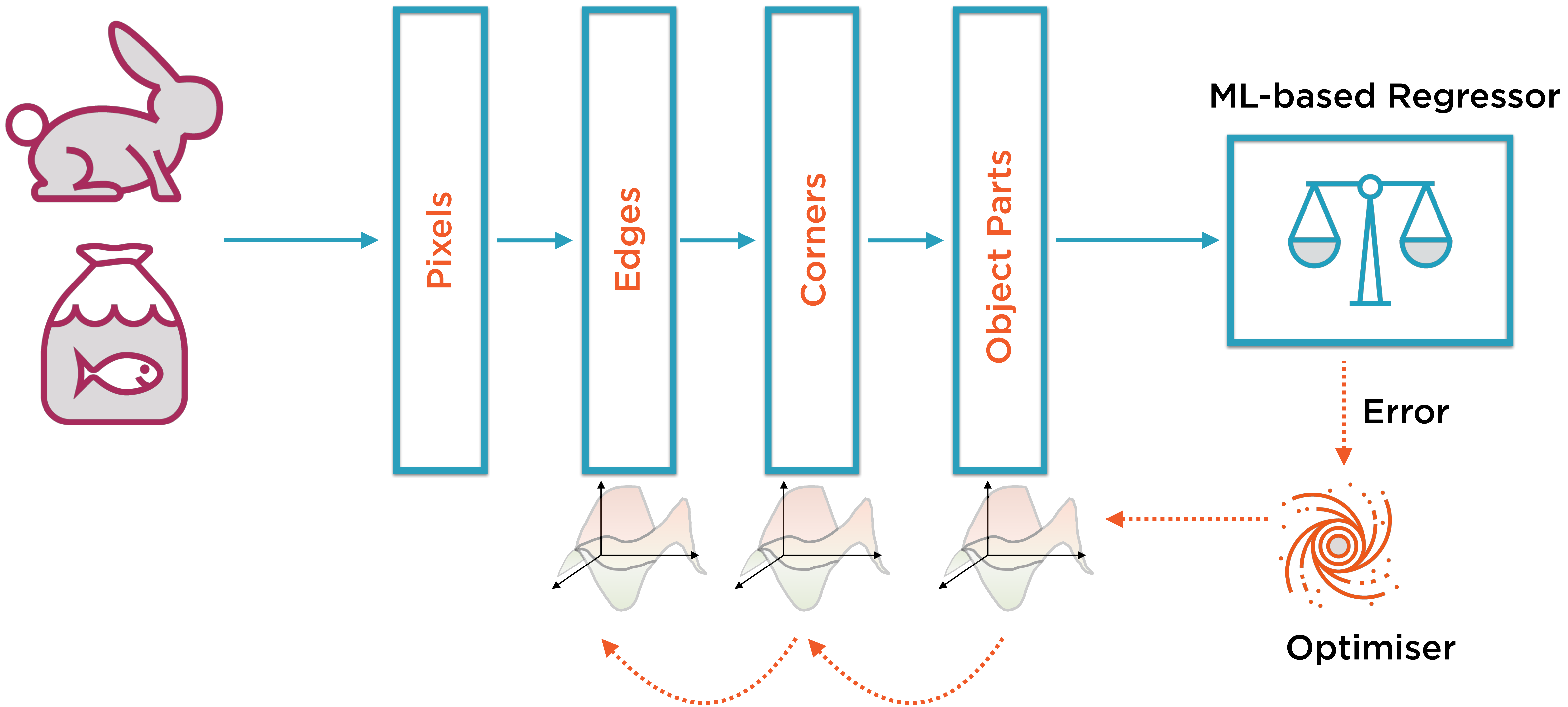
Calculated in backward pass
of time t

$$\text{Parameters}^{t+1} = \text{Parameters}^{t} - \text{learning\_rate} \times \text{Gradient}(\theta)^{t}$$

For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

Backward Pass at Time t



Updated in backward pass
of time t...

$$\text{Parameters}^{t+1} = \text{Parameters}^{t} - \text{learning\_rate} \times \text{Gradient}(\theta)^{t}$$

For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

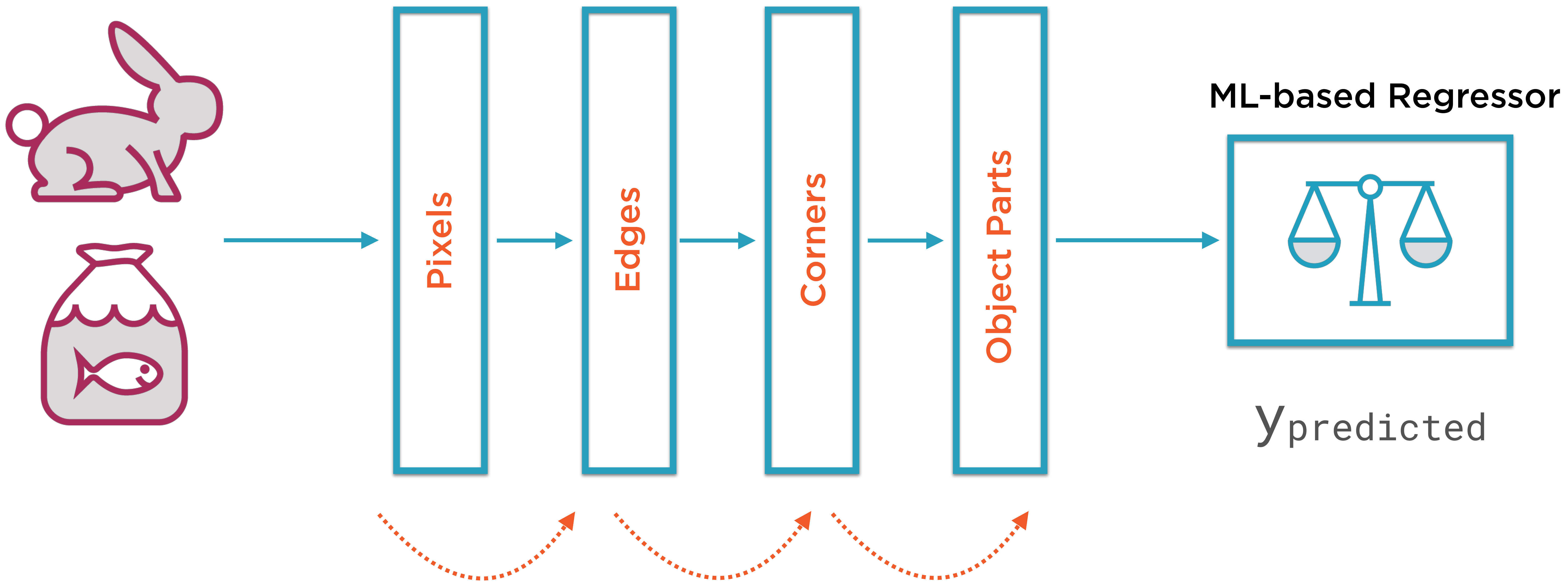
...then used in forward pass
of time t+1

$$Parameters^{t+1} = Parameters^{t} - learning\_rate \times Gradient(\theta)^{t}$$

For Next Time Step: Update Parameter Values

Move each parameter value in the direction of reducing gradient

Forward Pass at Time t+1



Use the current model weights and biases to make a prediction

Why Two Passes?

Symbolic Differentiation

Conceptually simple but hard to implement

Numeric Differentiation

Easy to implement but won't scale

Automatic Differentiation

Conceptually difficult but easy to implement

Because reverse auto-differentiation needs two passes

Automatic Differentiation

Conceptually difficult but easy to implement

Reverse-mode auto-differentiation

Used in TF, PyTorch

Two passes in each training step

- Forward step: Calculate loss
- Backward step: Update parameter values

Actually Calculating Gradients

Symbolic Differentiation

Conceptually simple but hard to implement

Numeric Differentiation

Easy to implement but won’t scale

Automatic Differentiation

Conceptually difficult but easy to implement

PyTorch, TensorFlow and other packages rely on automatic differentiation

Actually Calculating Gradients

Symbolic Differentiation

Conceptually simple but hard to implement

Numeric Differentiation

Easy to implement but won't scale

Automatic Differentiation

Conceptually difficult but easy to implement

Symbolic Differentiation

Conceptually simple but hard to implement

Actually calculate each element of gradient vector

(Approach adopted in example above)

Easy to understand, hard to implement

- complex neural networks
- Some activation functions are hard to differentiate
- Output function may be non-differentiable

Actually Calculating Gradients

Symbolic Differentiation

Conceptually simple but hard to implement

Numeric Differentiation

Easy to implement but won't scale

Automatic Differentiation

Conceptually difficult but easy to implement

Numeric Differentiation

Easy to implement but won't scale

Trivial to implement

$y = f(x)$

Add small "perturbation" ∂x to x

$y + \partial y = f(x + \partial x)$

$\partial y/\partial x = [f(x + \partial x) - f(x)]/\partial x$

Numeric Differentiation

Easy to implement but won't scale

Problem: need to do for each parameter

Will not scale to complex networks

(May have thousands of parameters)

Actually Calculating Gradients

Symbolic Differentiation

Conceptually simple but hard to implement

Numeric Differentiation

Easy to implement but won’t scale

Automatic Differentiation

Conceptually difficult but easy to implement

Automatic Differentiation

Conceptually difficult but easy to implement

Relies on a mathematical trick

Based on Taylor’s Series Expansion

Allow fast approximation of gradients

Automatic Differentiation

Conceptually difficult but easy to implement

Two flavors

- Forward-mode
- Reverse-mode

Automatic Differentiation

Conceptually difficult but easy to implement

Forward-mode ~ numeric differentiation

Suffers from same flaw...

...Requires one pass per parameter

Will not scale to complex networks

Automatic Differentiation

Conceptually difficult but easy to implement

Reverse-mode used in TF, PyTorch...

Two passes in each training step

- Forward step: Calculate loss
- Backward step: Update parameter values

Back propagation is only required during training: to do so in PyTorch, invoke the .backward() method

Deprecation of the Variable API in PyTorch

PyTorch used to wrap Tensors within Variables which kept track of changing parameter values; Variables are now deprecated

Variables (Deprecated)

$\{y,x\}$

Used to be necessary to use autograd with tensors

Now autograd automatically supports Tensors with requires_grad = True

Variables (Deprecated)

Variable creation now returns tensors instead of variables

var.data is identical to tensor.data

Methods such as var.backward(), var.detach() work on tensors

Tensor method names stay the same

Demo

Introducing Autograd in PyTorch

Demo

Using Autograd with variables

Demo

Building a linear model with Autograd

Summary

Gradient descent to train a neural network

Forward and backward passes

Different methods for gradient calculation

Automatic differentiation using Autograd