Computing Gradients for Model Training



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

Overview

Training a neural network

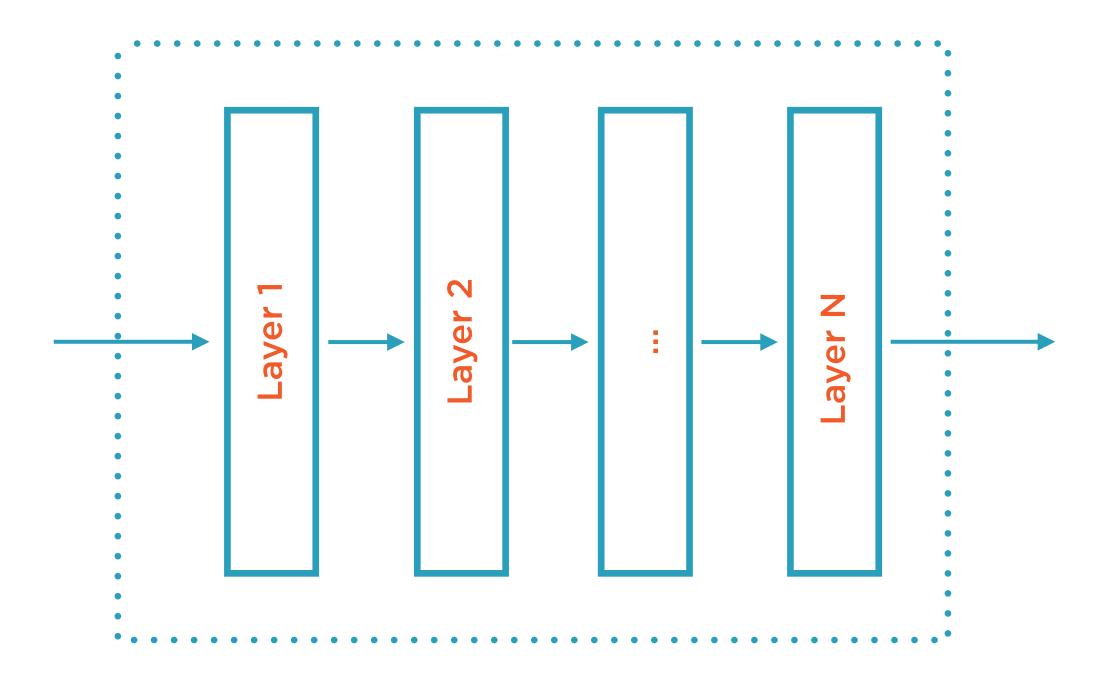
Backpropagation and gradient descent

Gradients and their calculation

Training with gradient tape

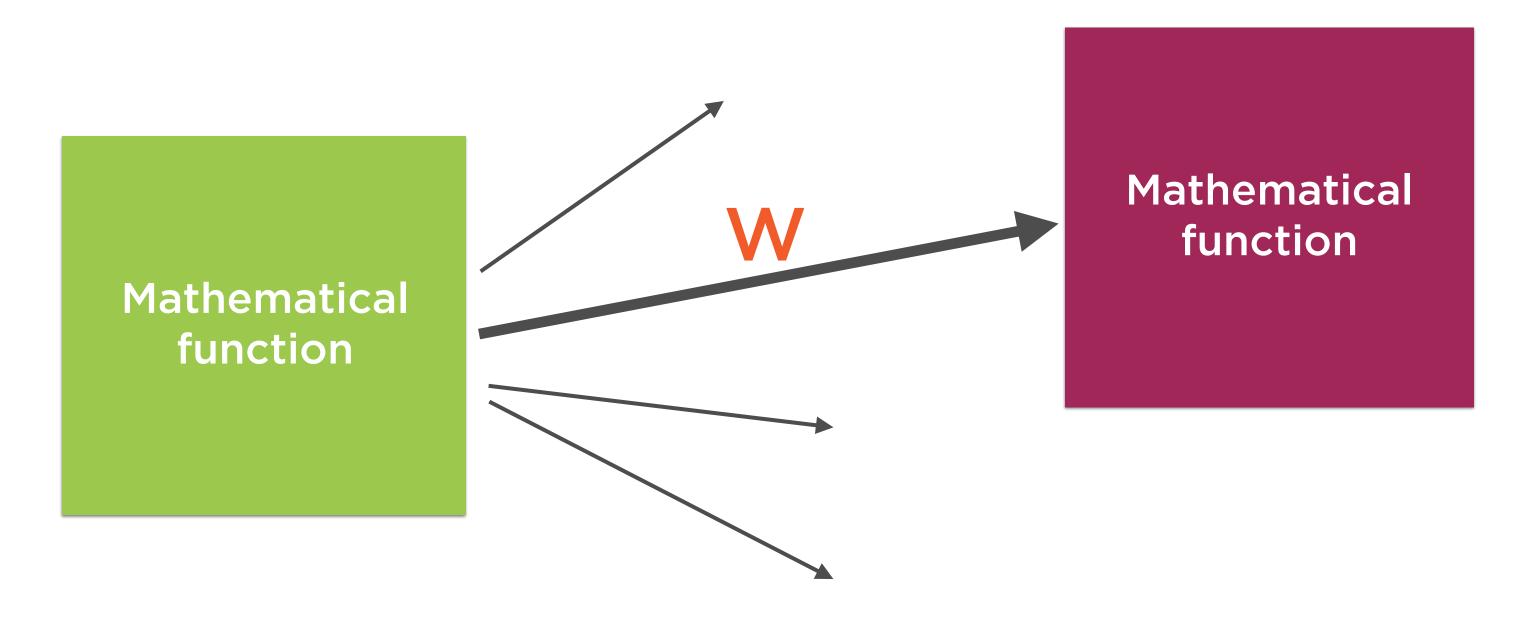
Gradient Descent

Neural Network Model



Interconnected neurons arranged in layers

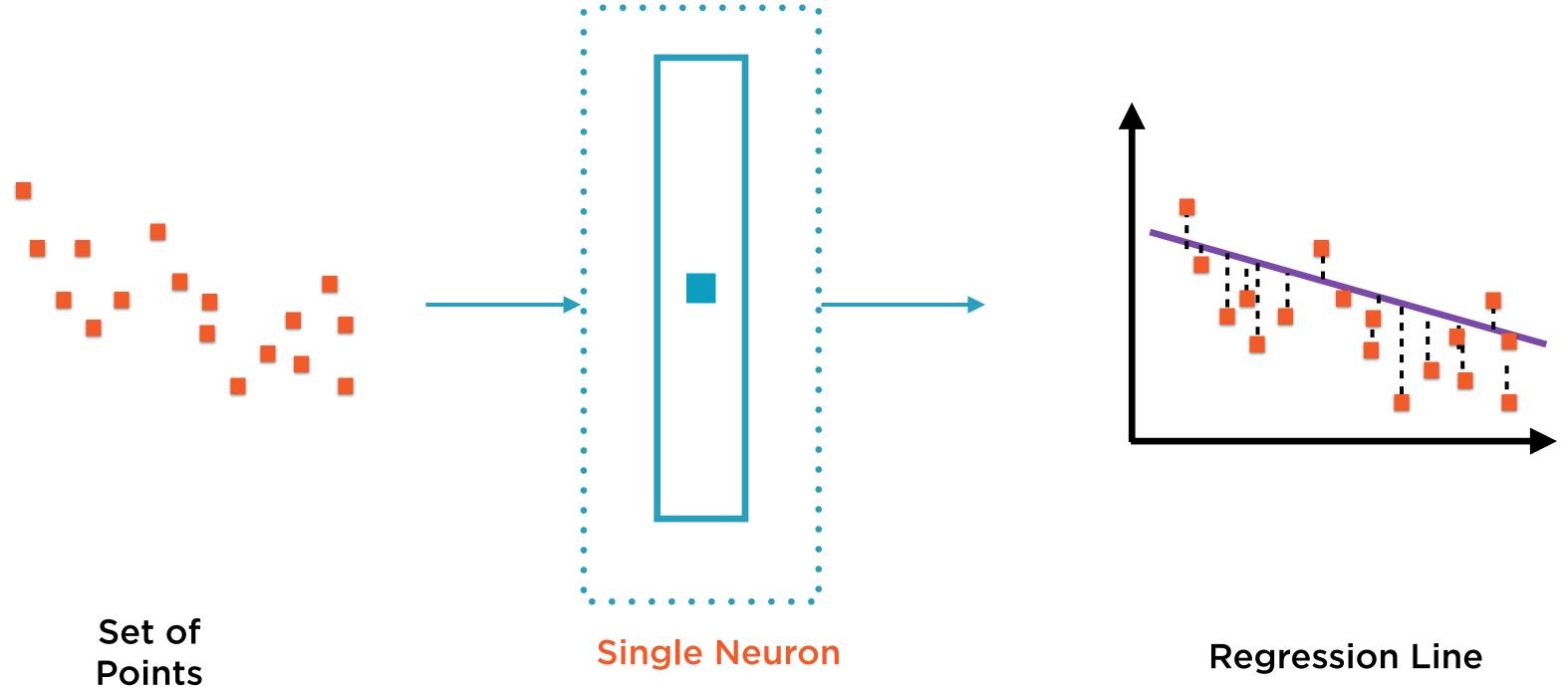
Each Connection Associated with a Weight



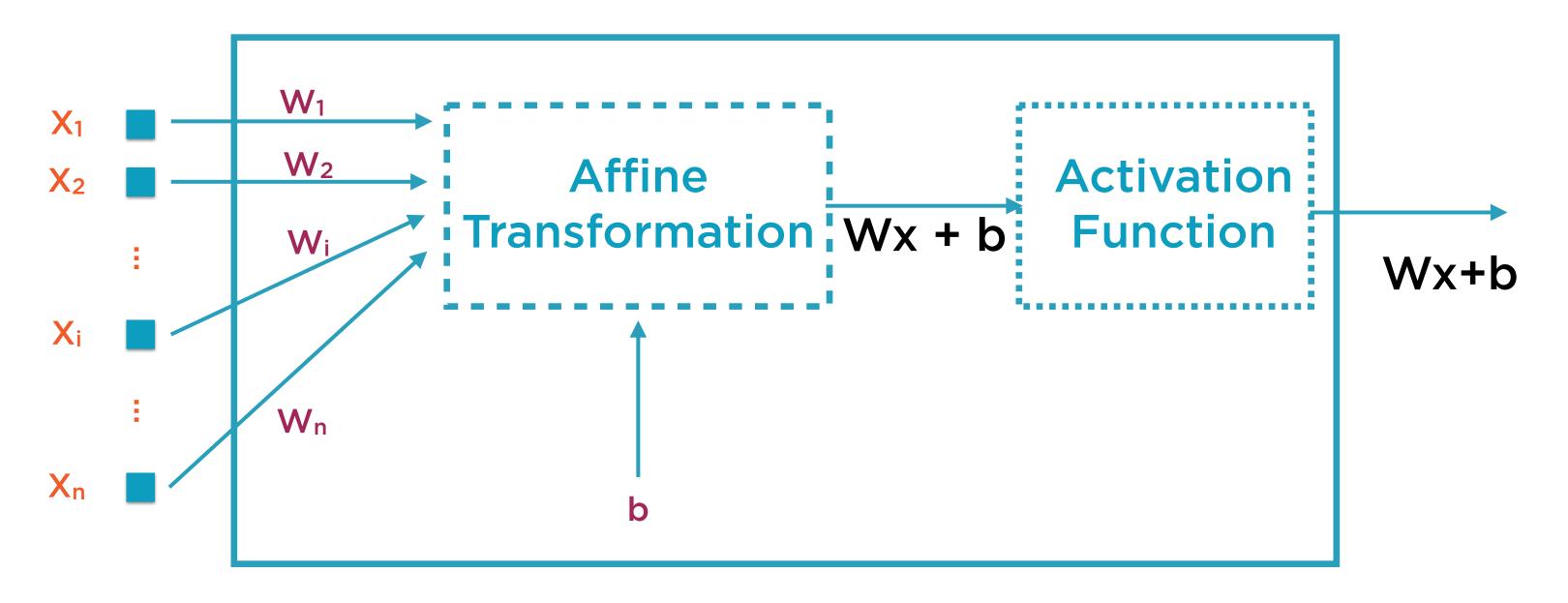
If the second neuron is sensitive to the output of the first neuron, the connection between them gets stronger

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

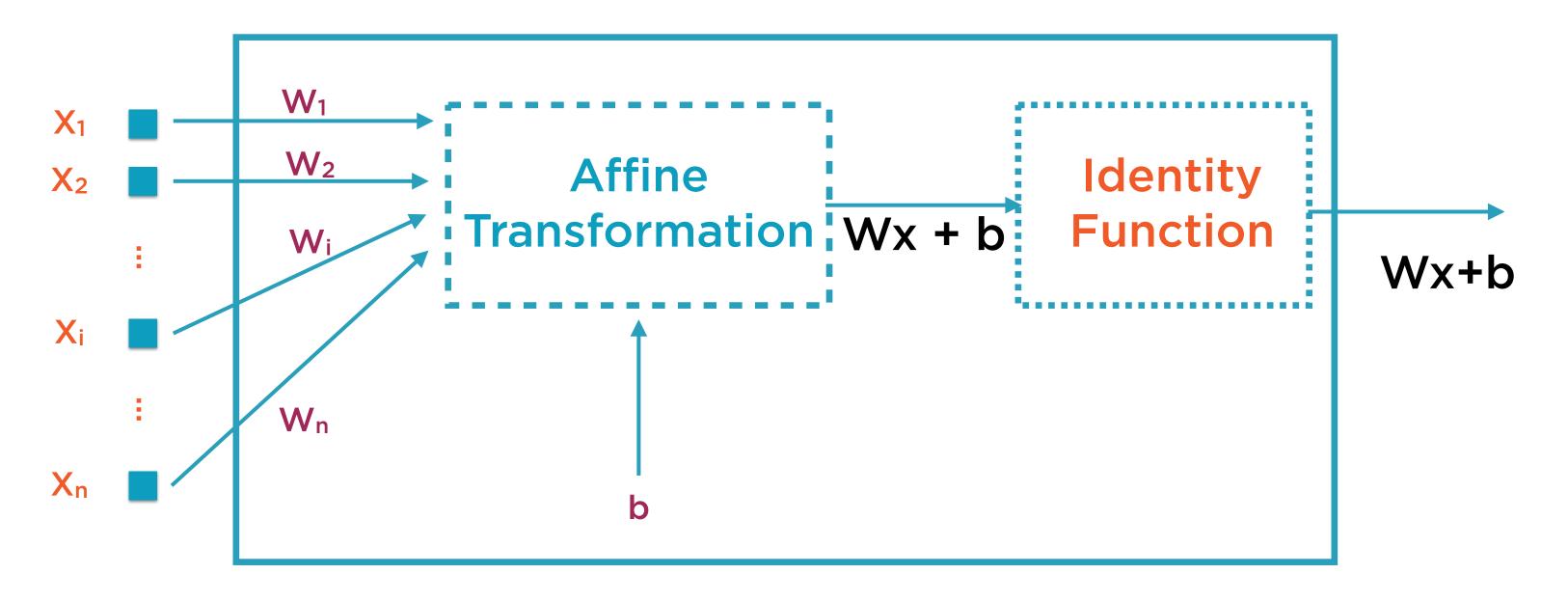
Regression: The Simplest Neural Network

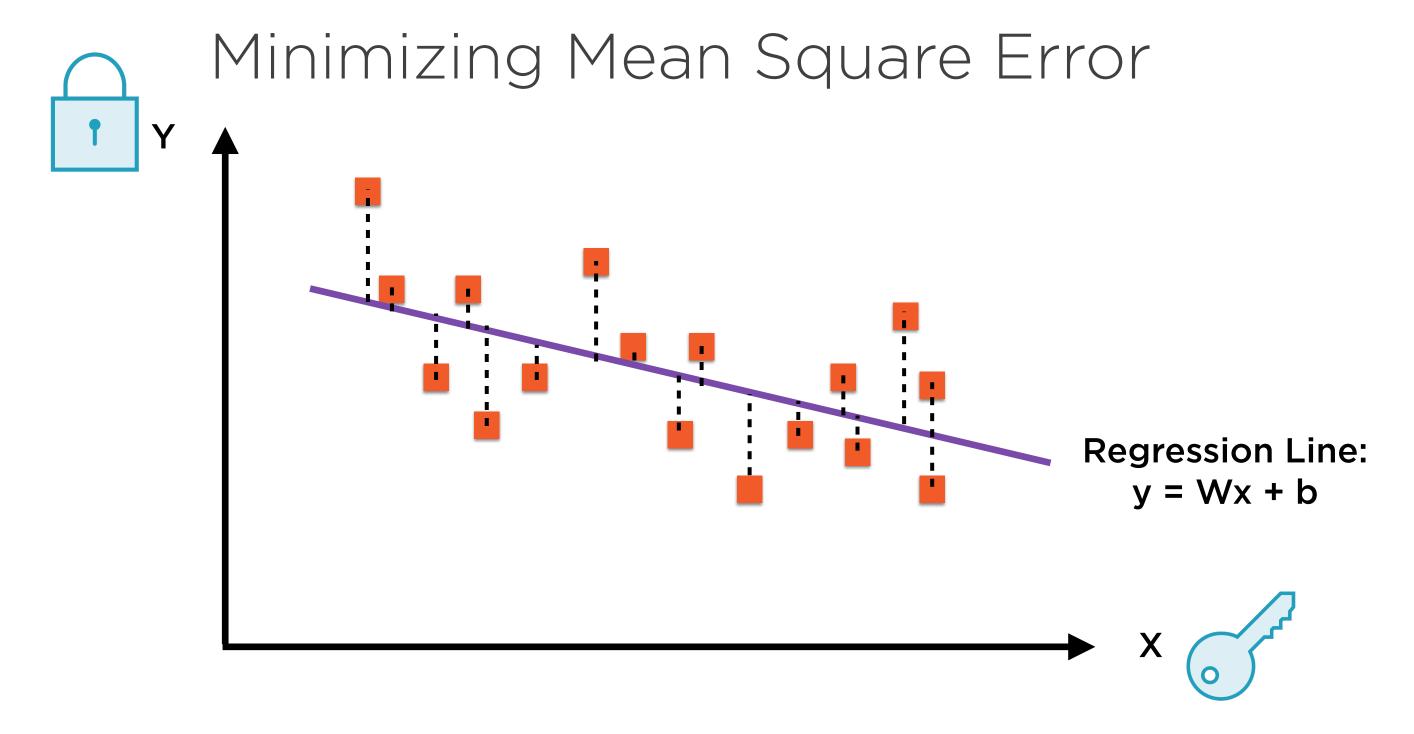


Regression: The Simplest Neural Network

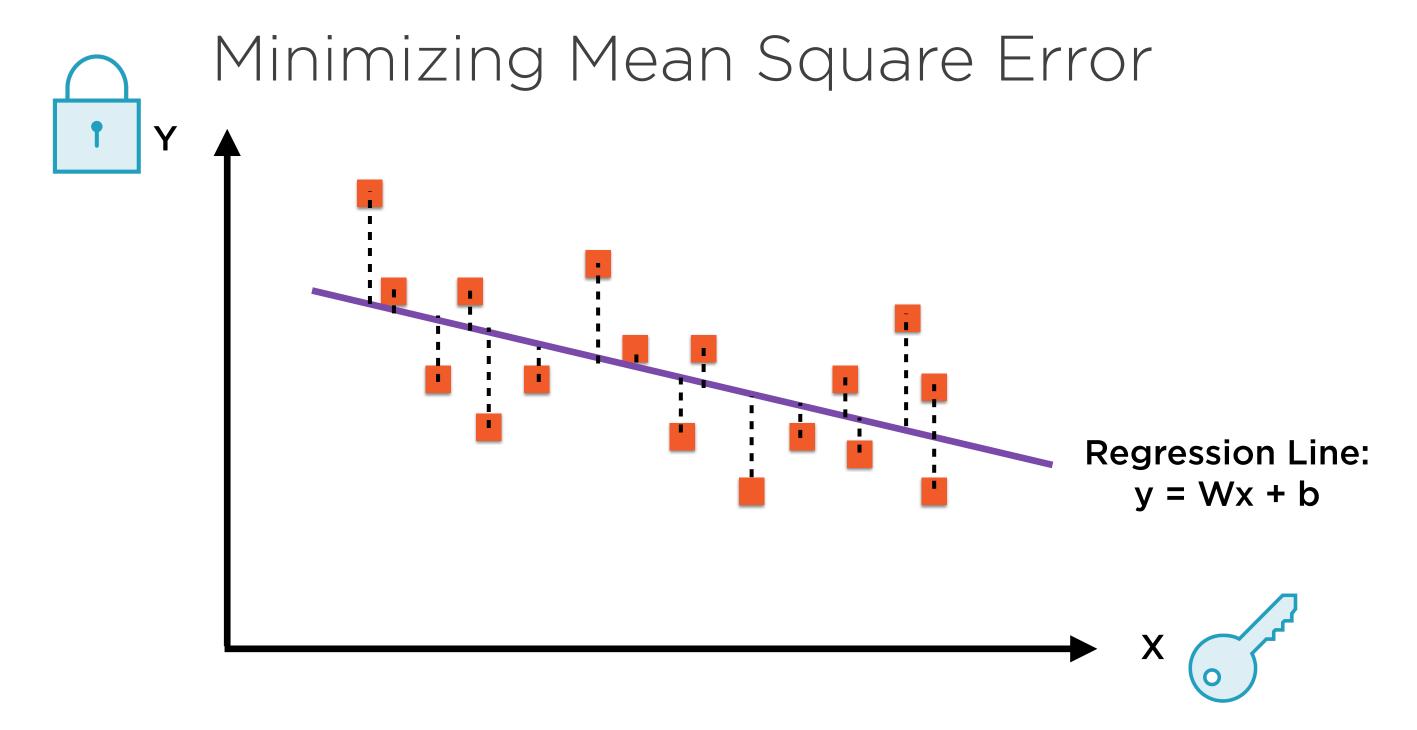


Regression: The Simplest Neural Network



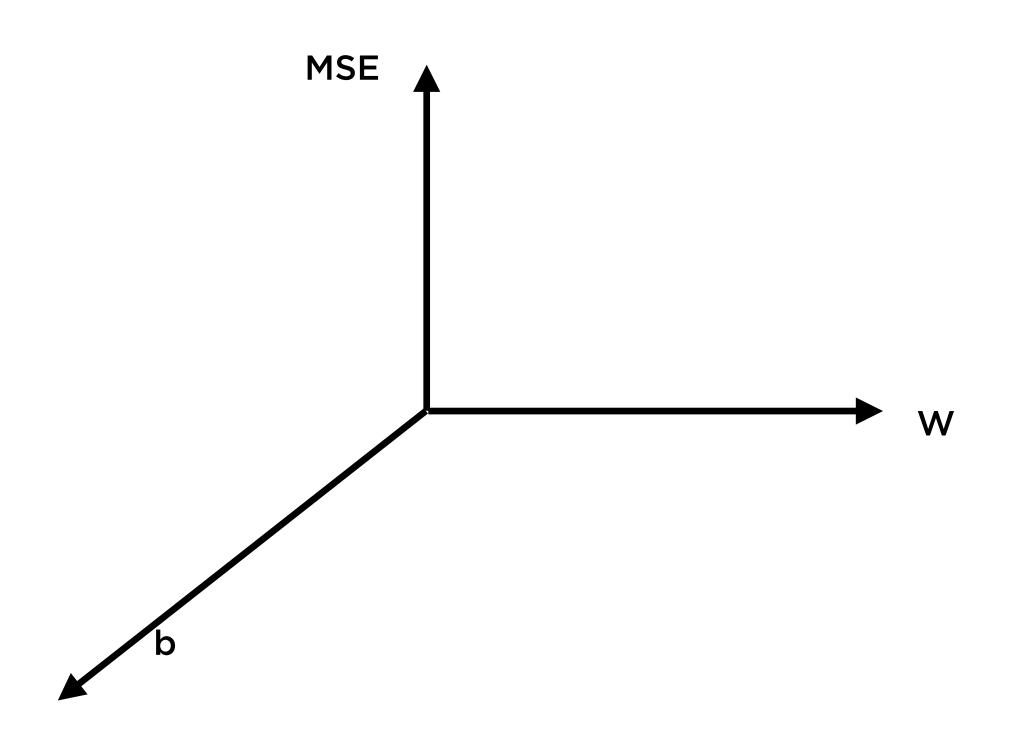


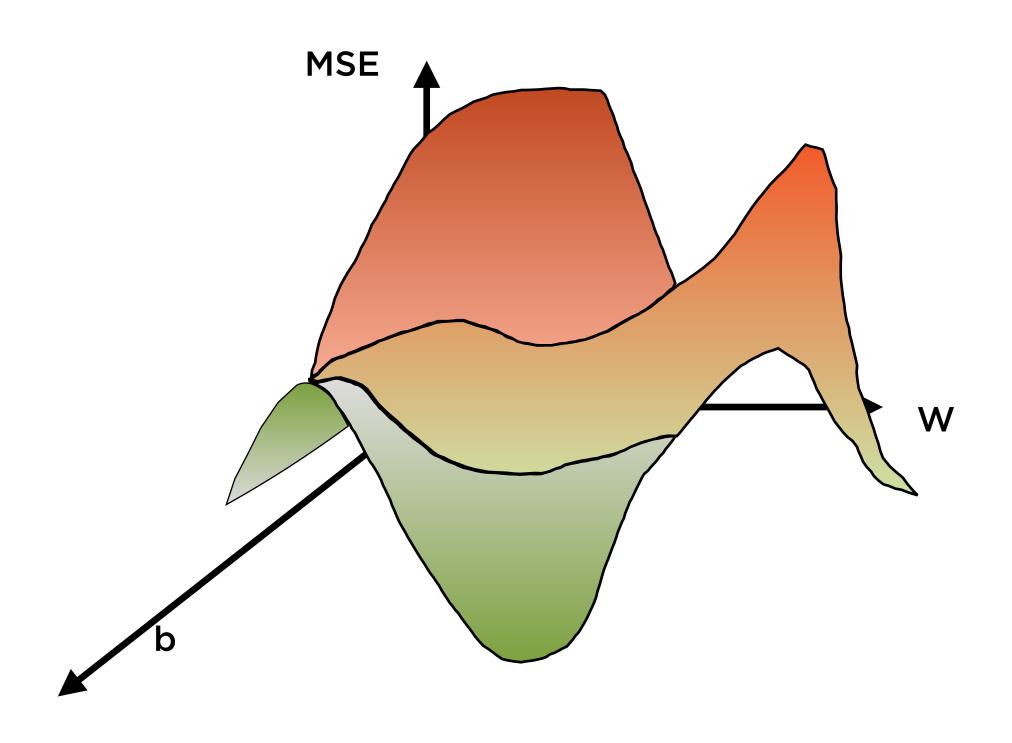
The "best fit" line is called the regression line

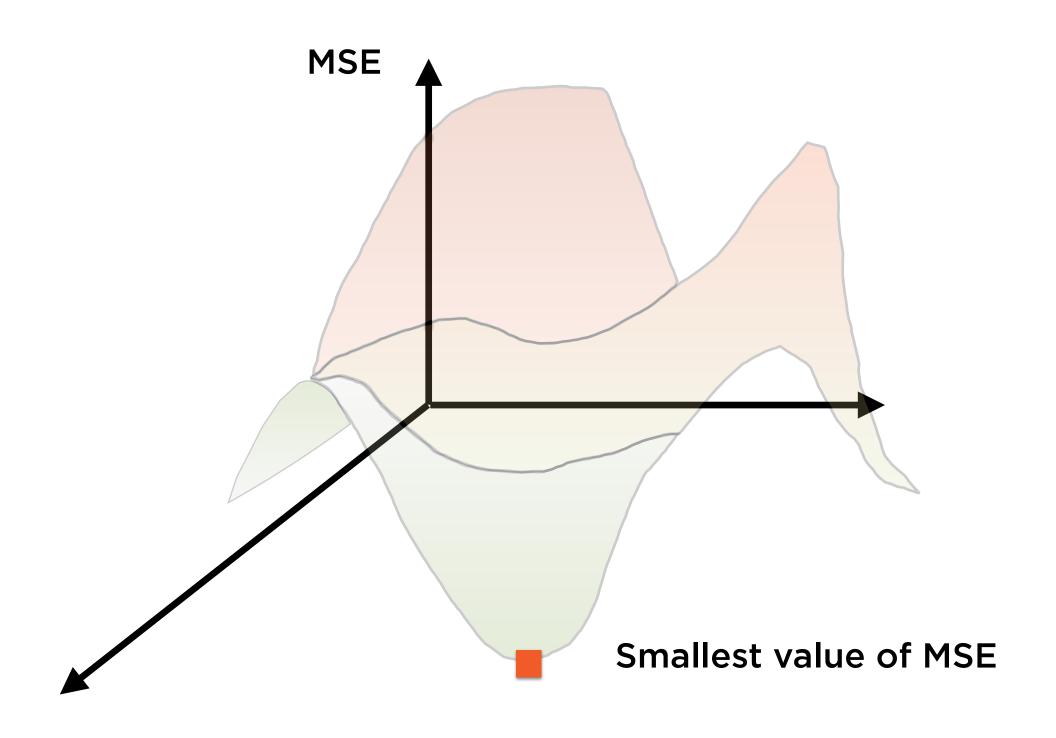


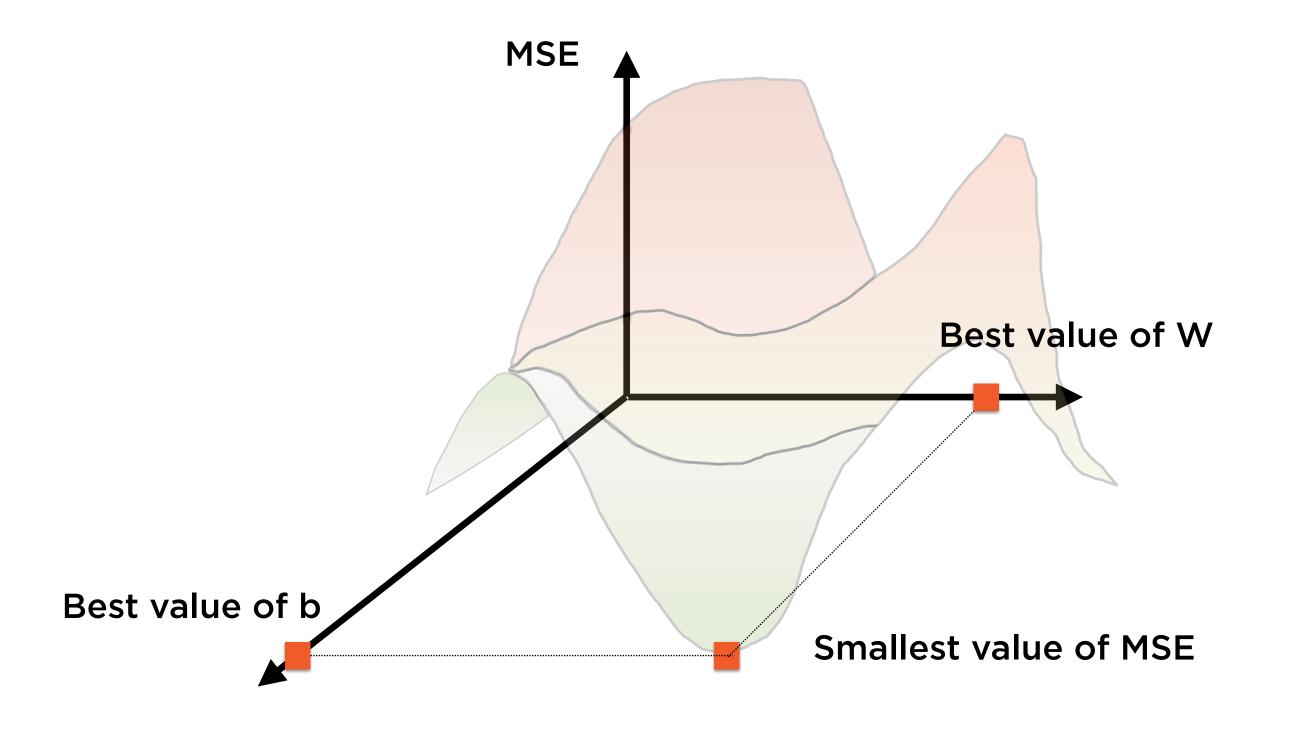
Minimize the sum of the squares of the distances of the points from the regression line

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

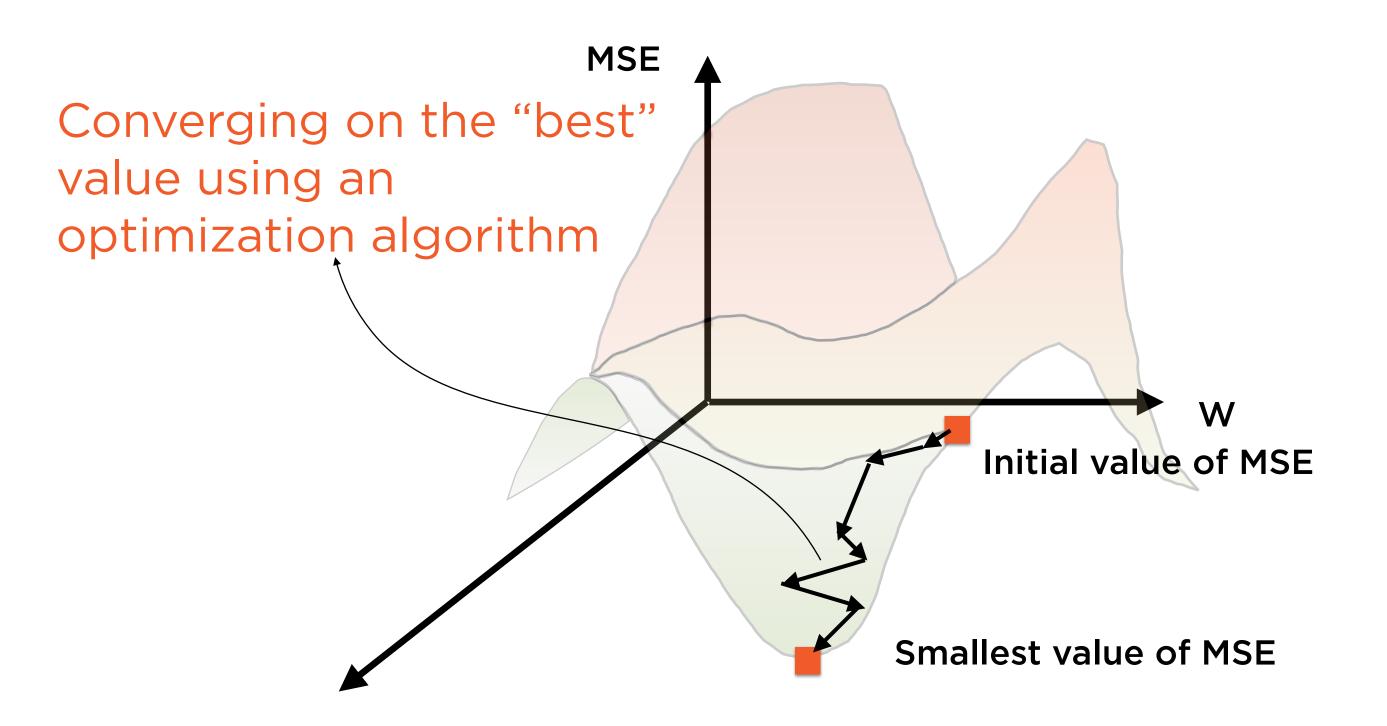




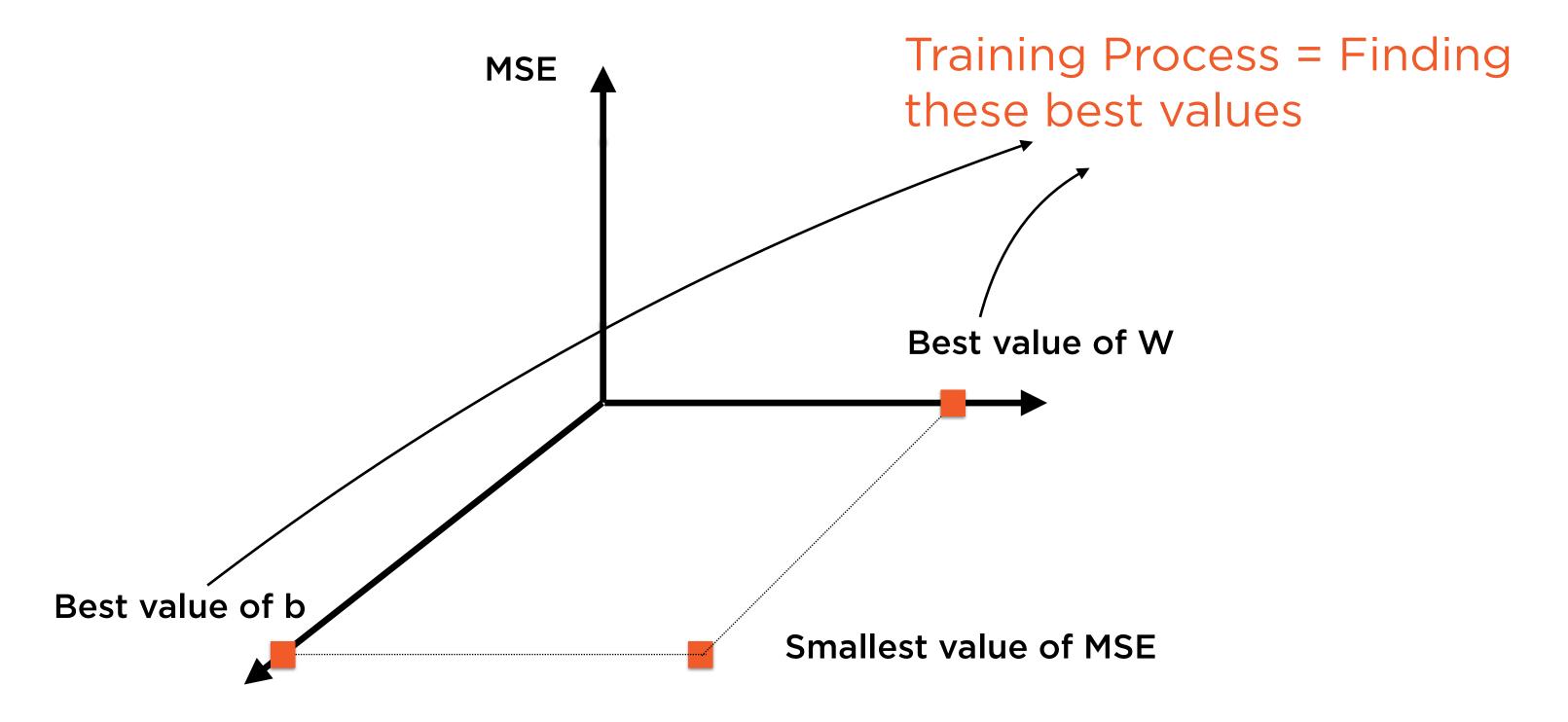




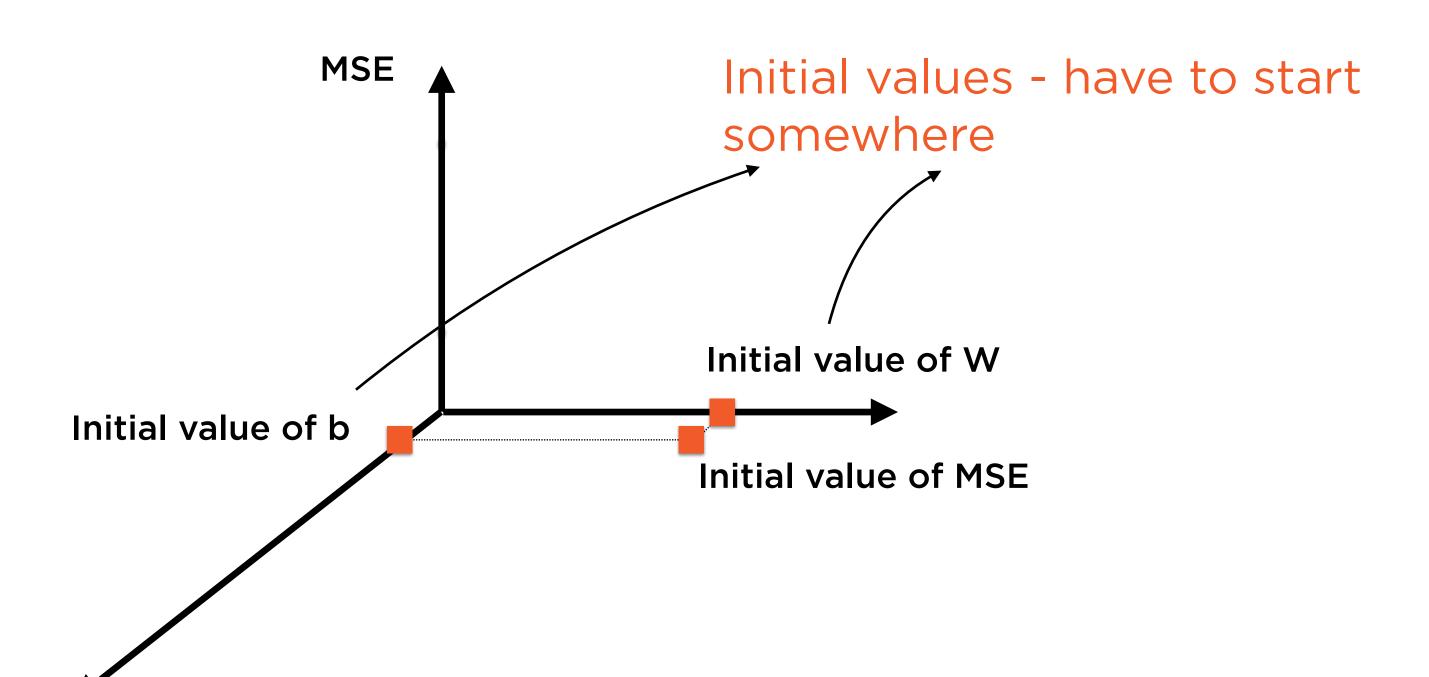
Gradient Descent



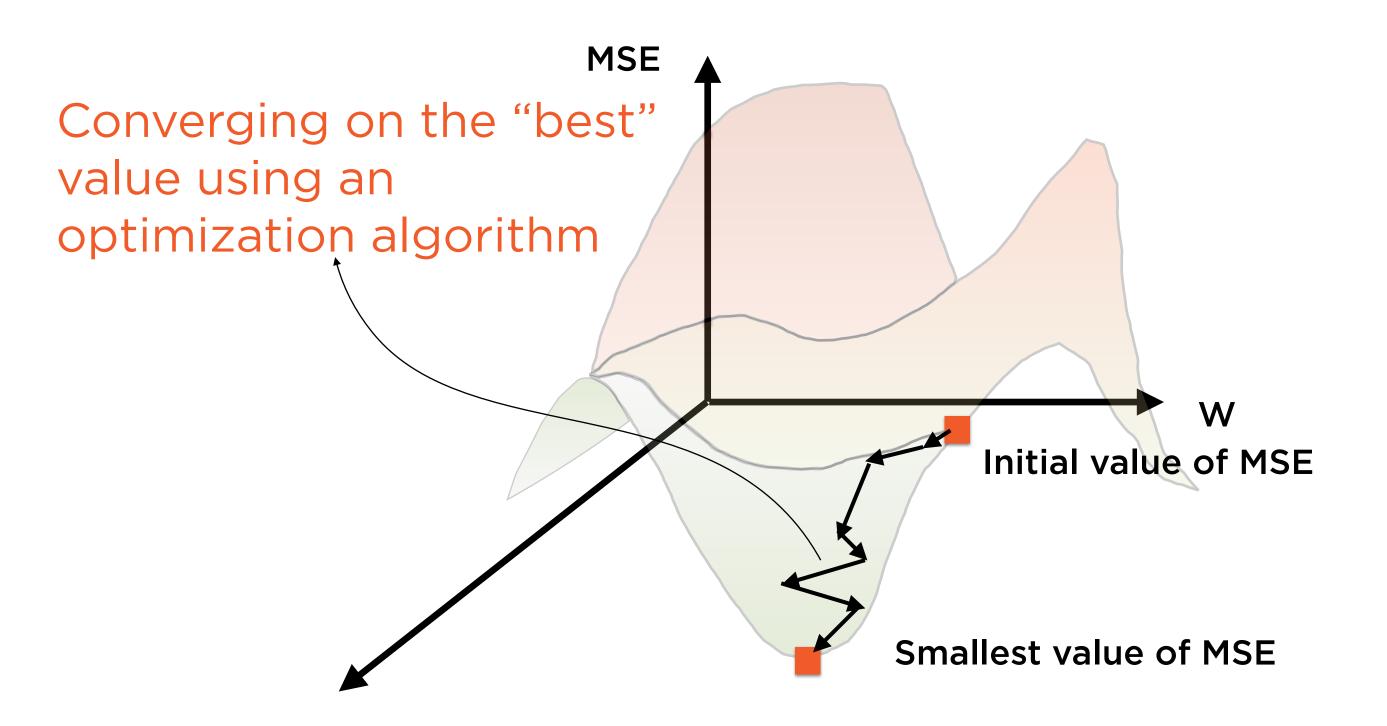
Training the Algorithm



Start Somewhere

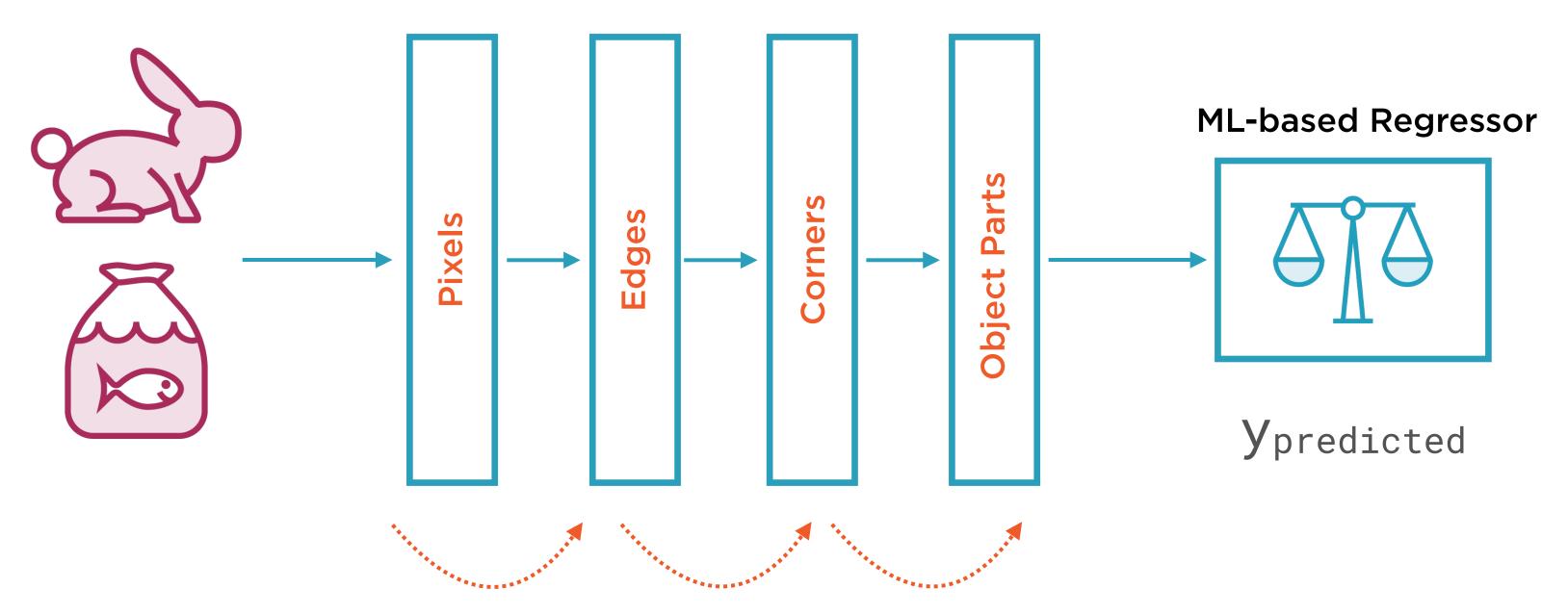


Gradient Descent



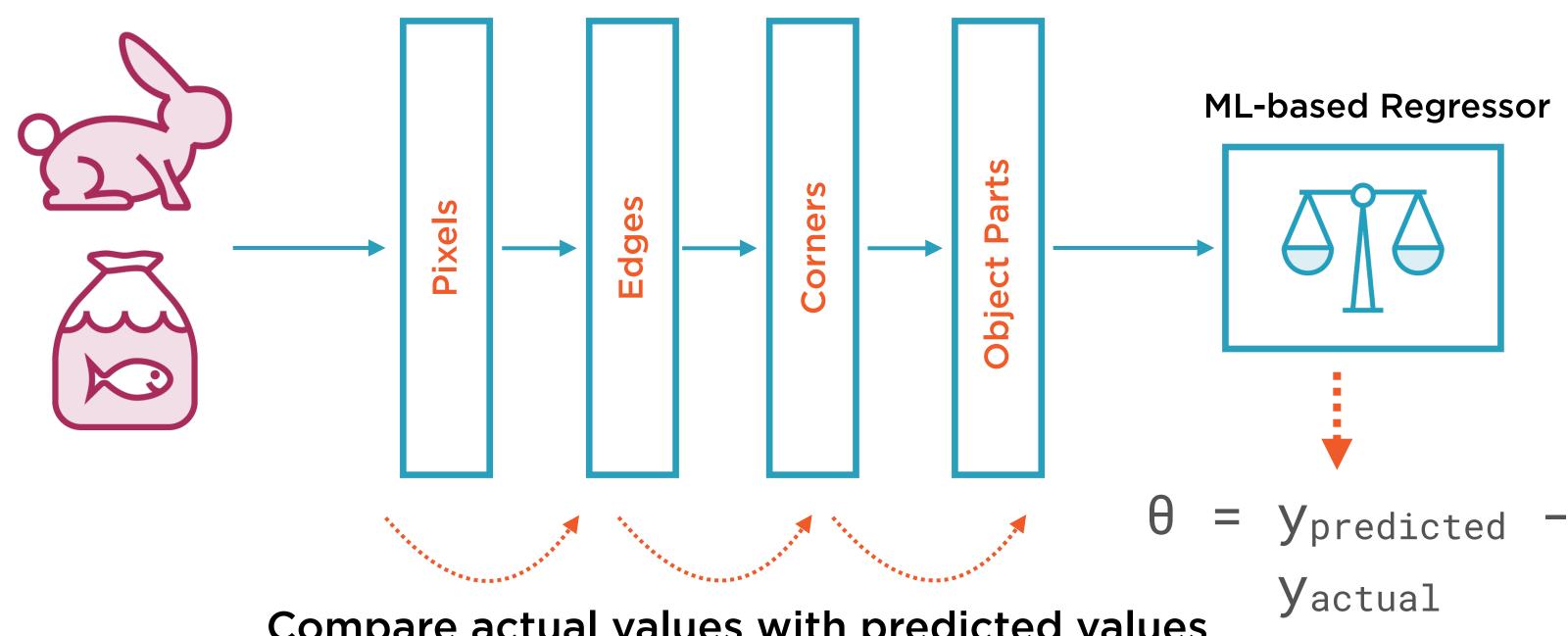
Forward and Backward Passes

Forward Pass



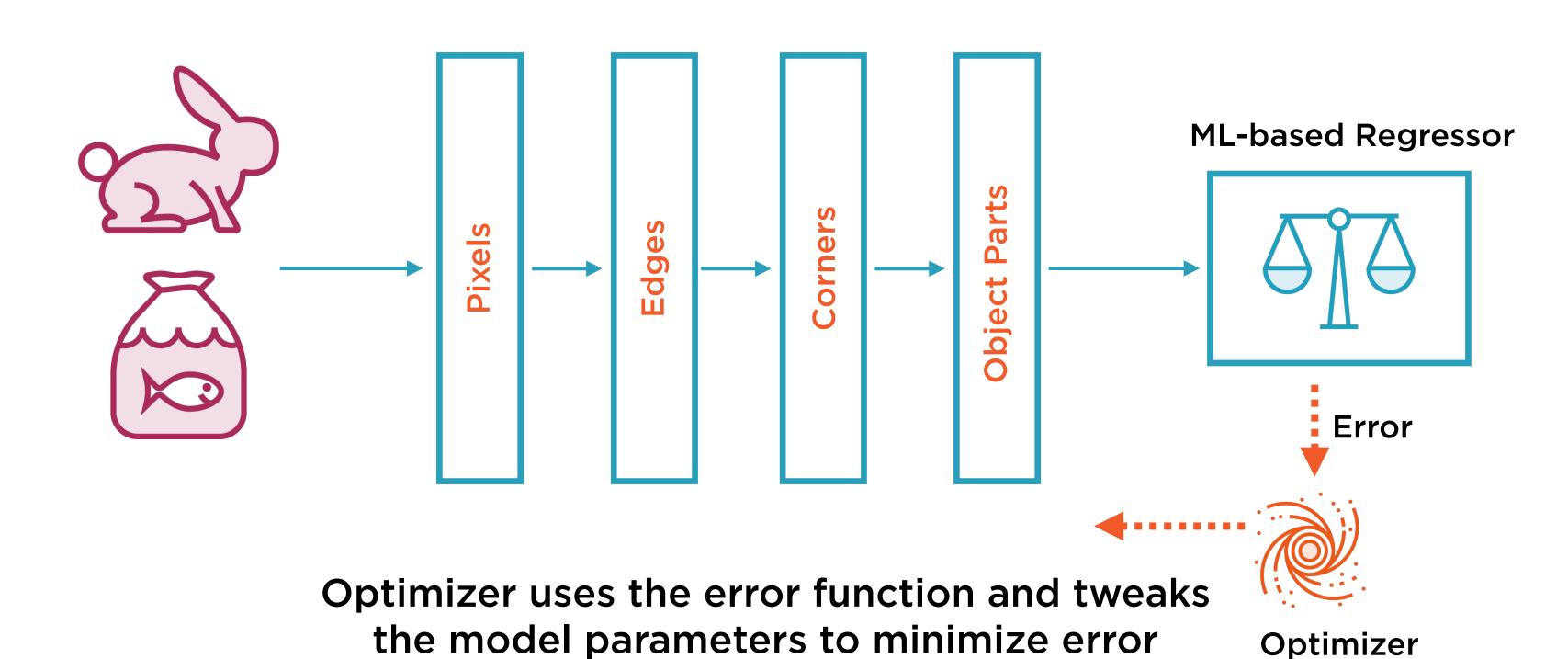
Use the current model weights and biases to make a prediction

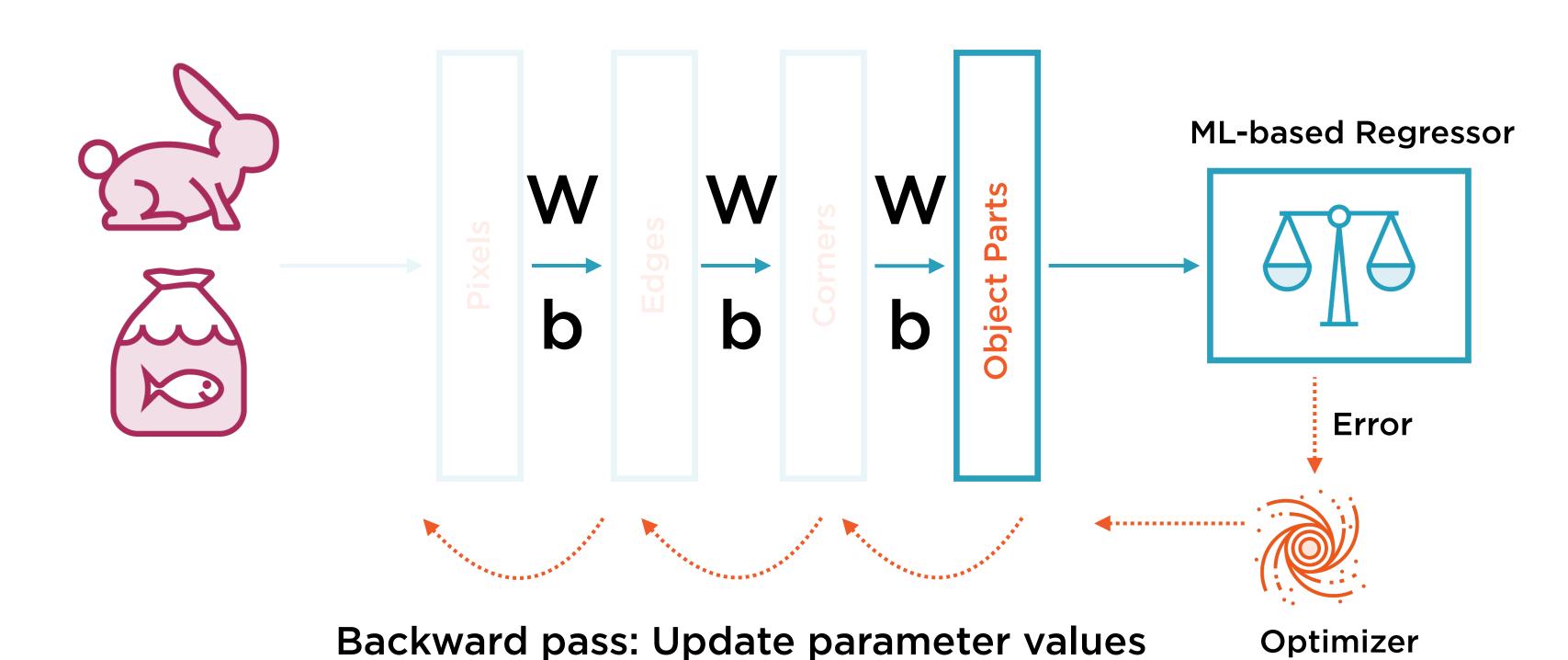
Forward Pass

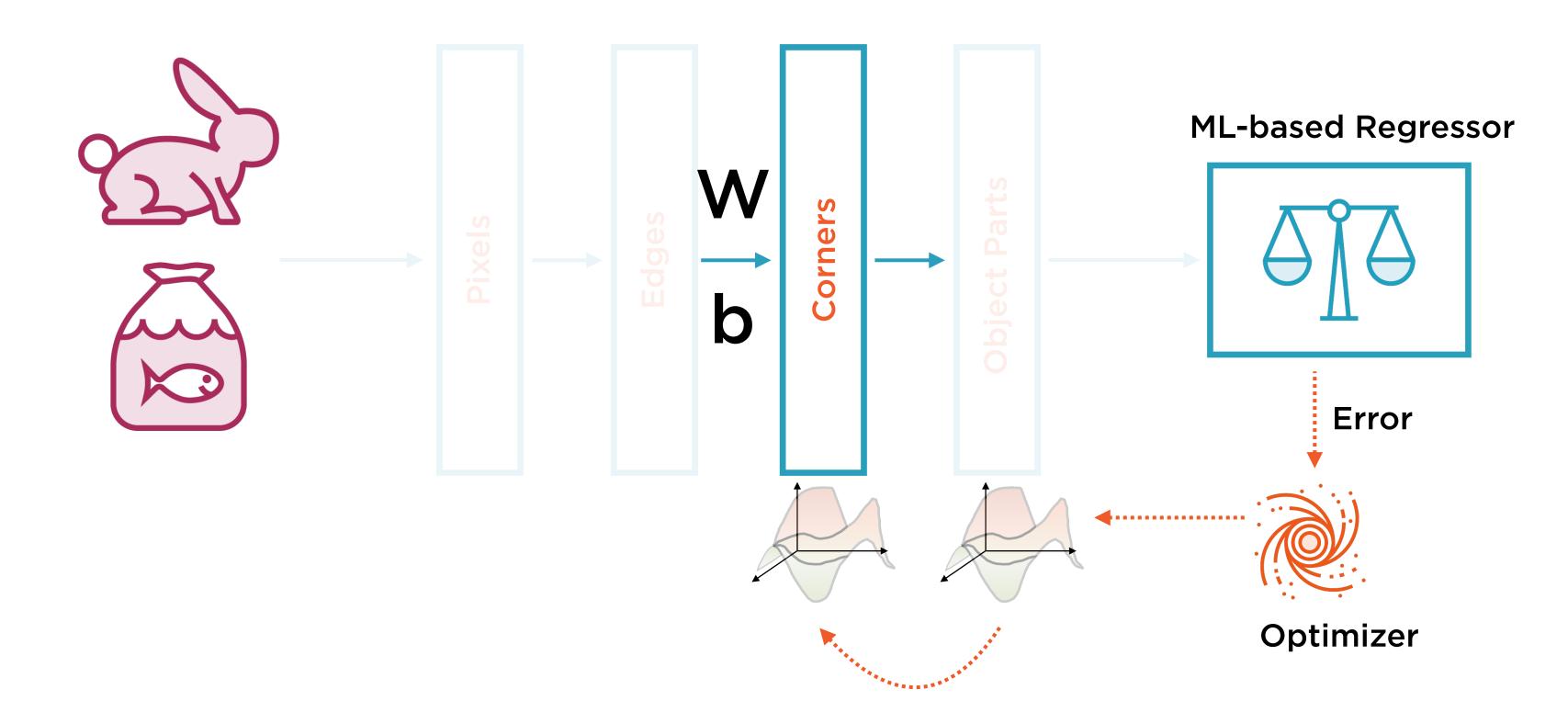


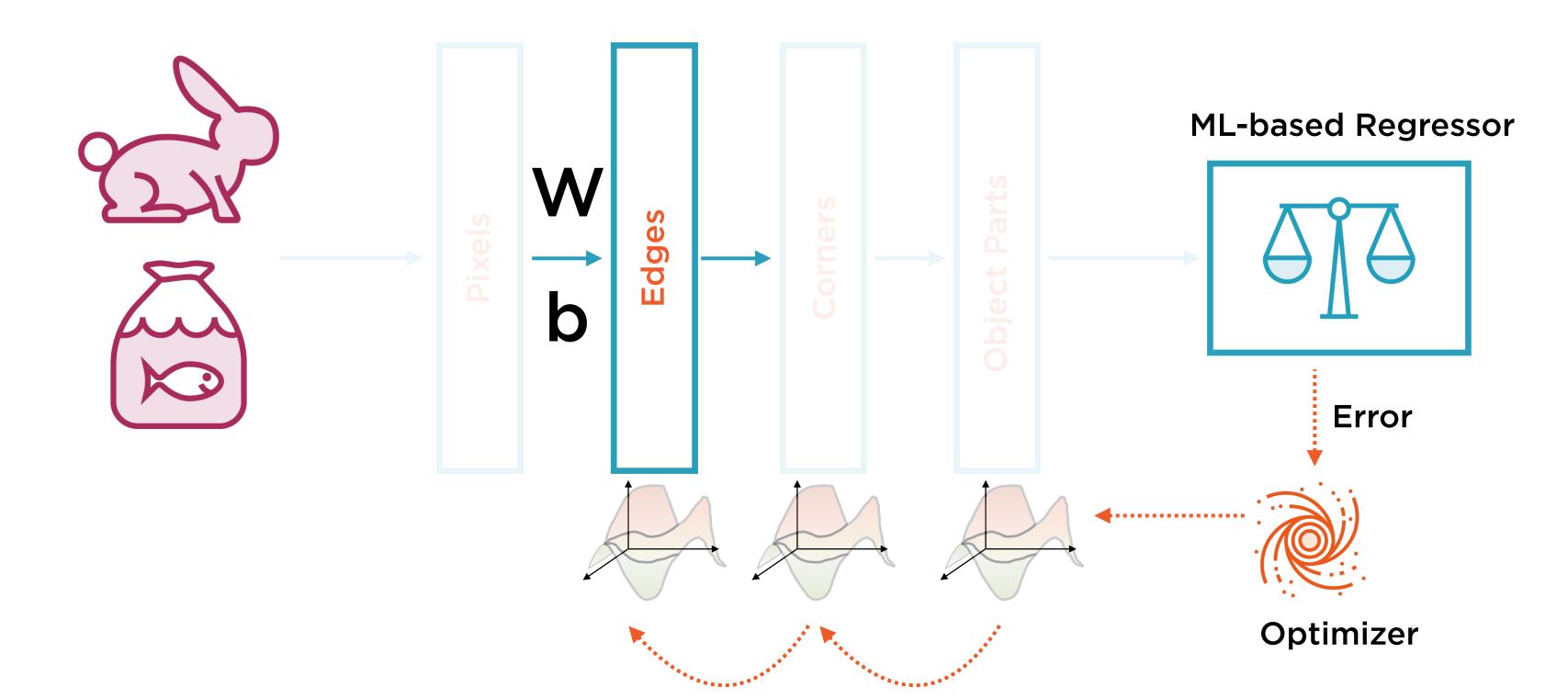
Compare actual values with predicted values and calculate the error

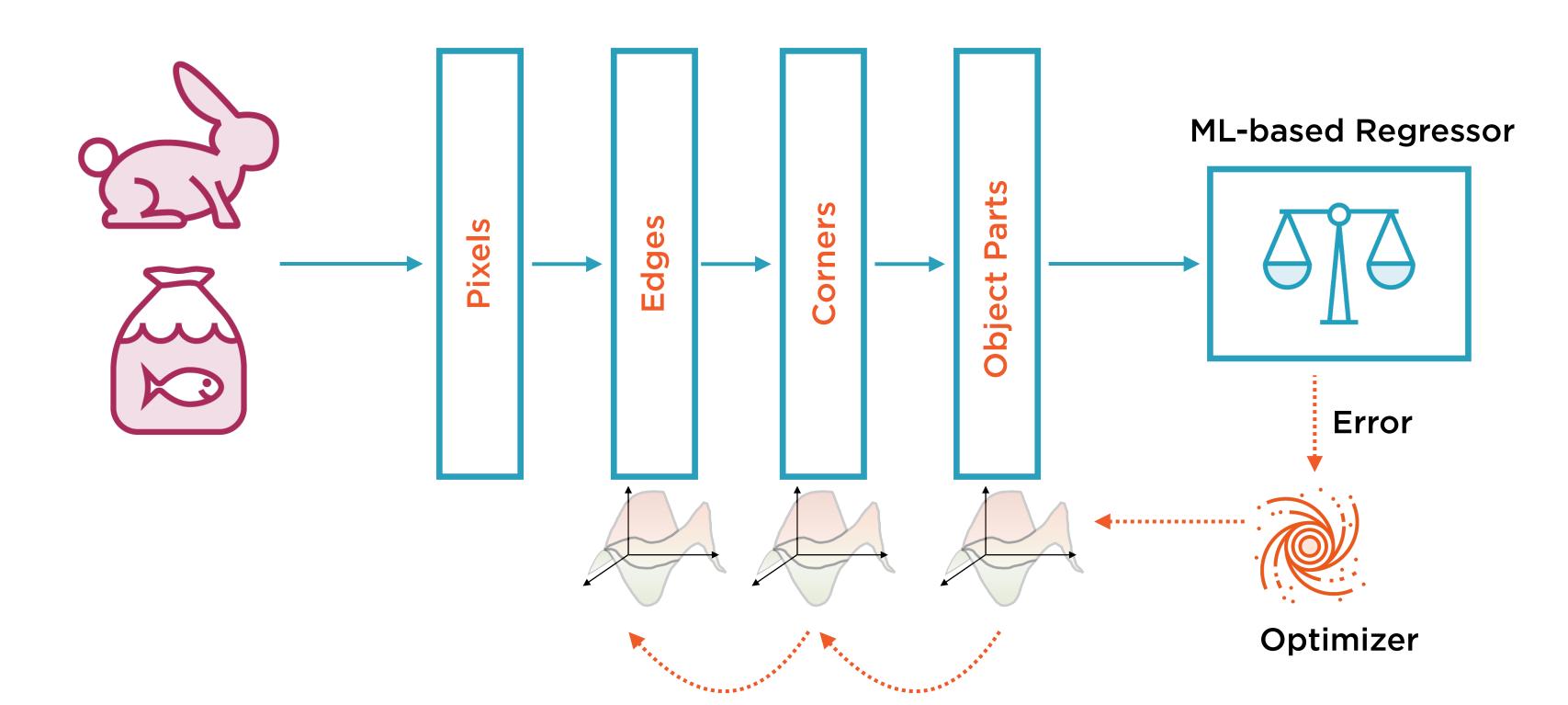
Optimizer Calculates Gradients

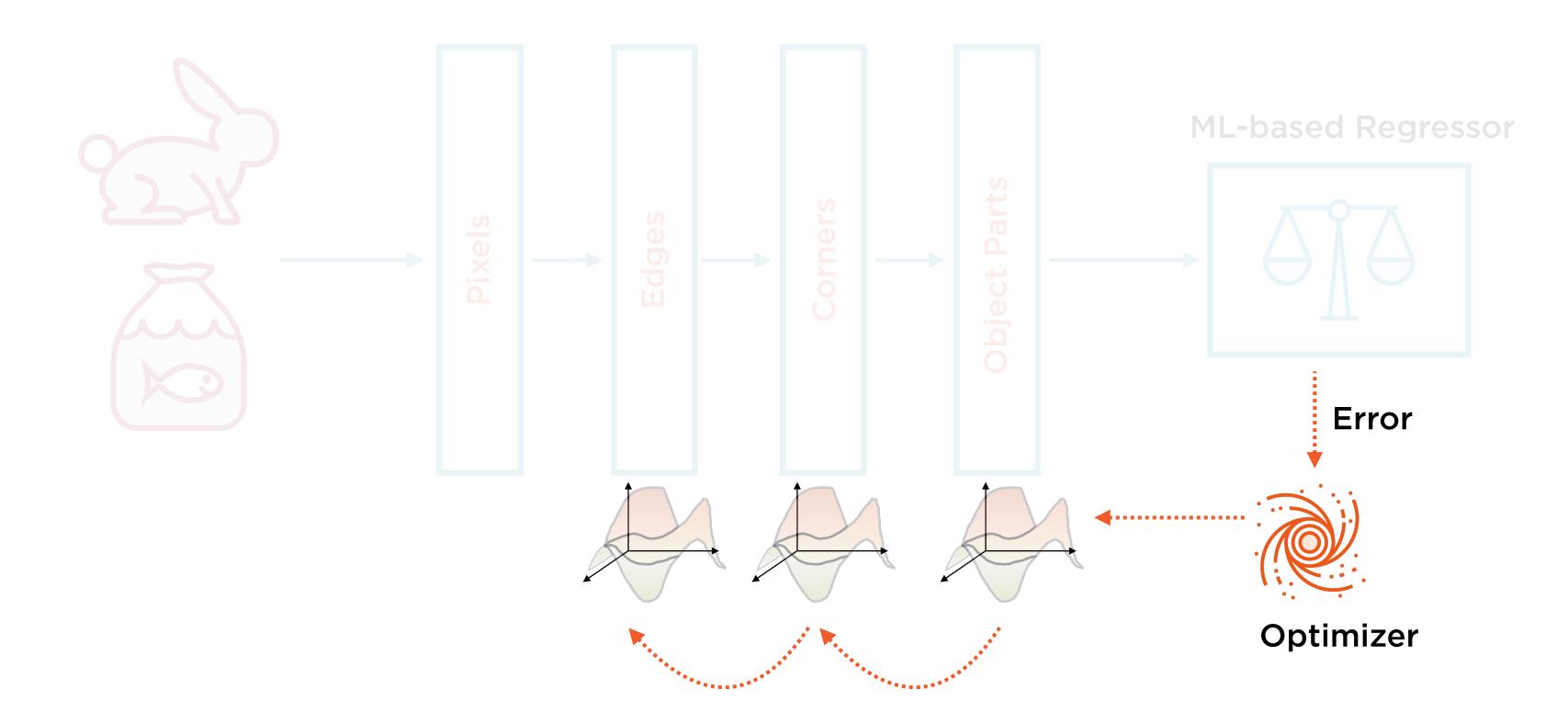












The backward pass allows the weights and biases of the neurons to converge to their final values

Calculating Gradients Using Gradient Tape

MSE = 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

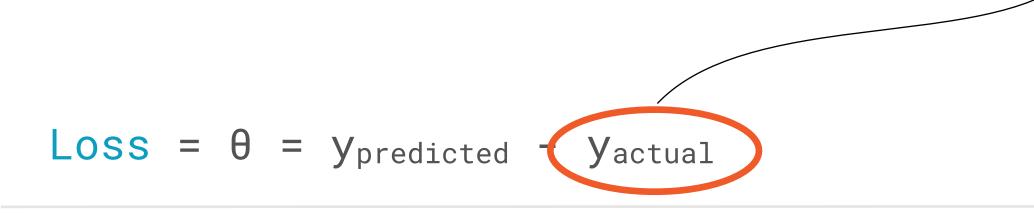
Given x, model outputs predicted value of y

Loss = θ = $y_{predicted}$ - y_{actual}

Loss Function θ

Loss function measures inaccuracy of model on a specific instance

Actual label, available in training data



Loss Function θ

Loss function measures inaccuracy of model on a specific instance

In NN with 1 Neuron:

Gradient(θ) = $\nabla \theta(W_1, b_1)$ = $(\partial \theta/\partial W_1, \partial \theta/\partial b_1)$

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:

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:

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:

Gradient(θ) = $\nabla \theta(W_1, b_1)$ = $(\partial \theta/\partial W_1, \partial \theta/\partial b_1)$

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:

```
Gradient(\theta) = \nabla \theta(W_1, b_1...W_{10000}, b_{10000})
= (\partial \theta / \partial W_1, \partial \theta / \partial b_1, ... \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

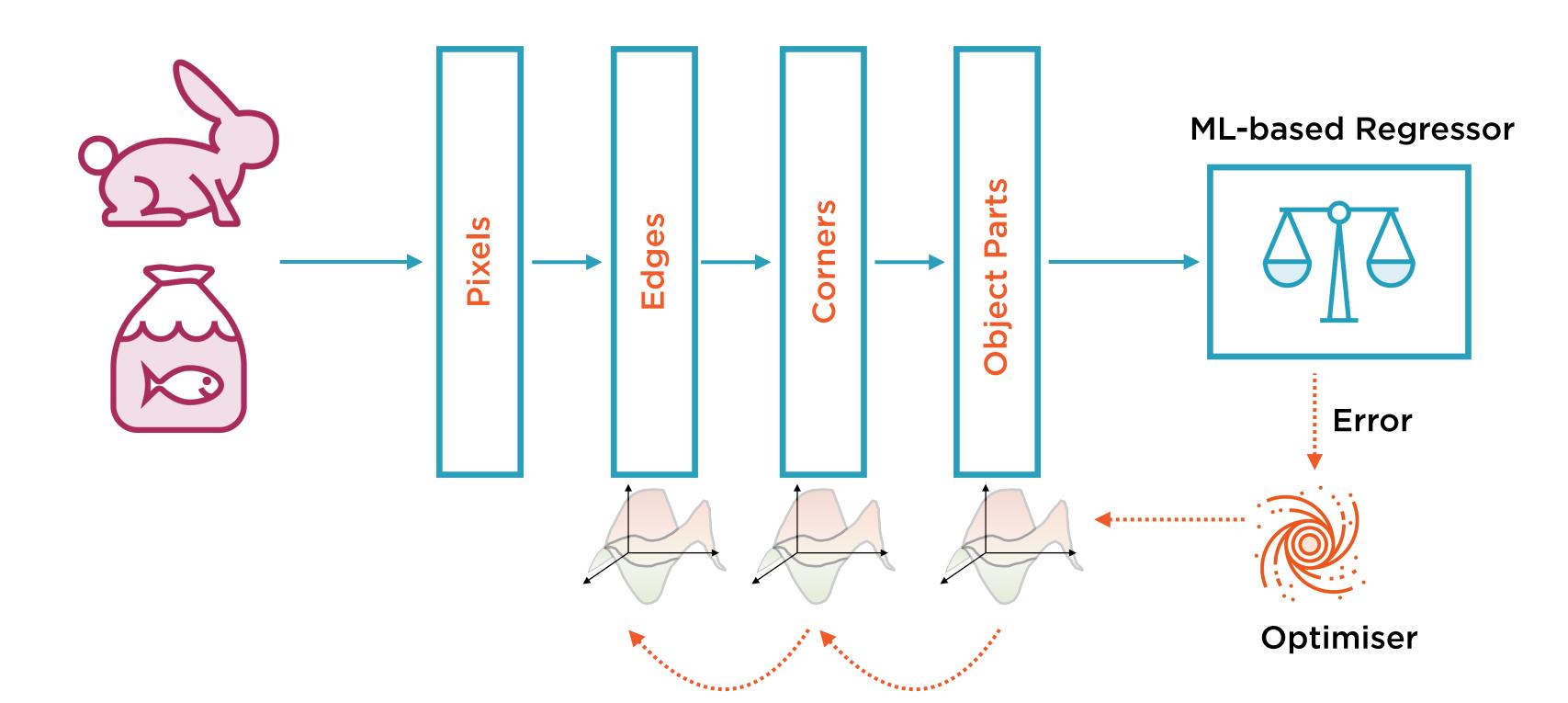
Automatic Differentiation

Conceptually difficult but easy to implement

TensorFlow, PyTorch and other packages rely on automatic differentiation

These gradients are used to update the model parameters

Backward Pass



Gradient Tape is the TF2.0 package for calculating gradients for back propagation

Reverse Mode Automatic Differentiation

 $Gradient(\theta)$

Gradient: Vector of Partial Derivatives

These gradients apply to a specific time t



Gradient: Vector of Partial Derivatives

These gradients apply to a specific time t

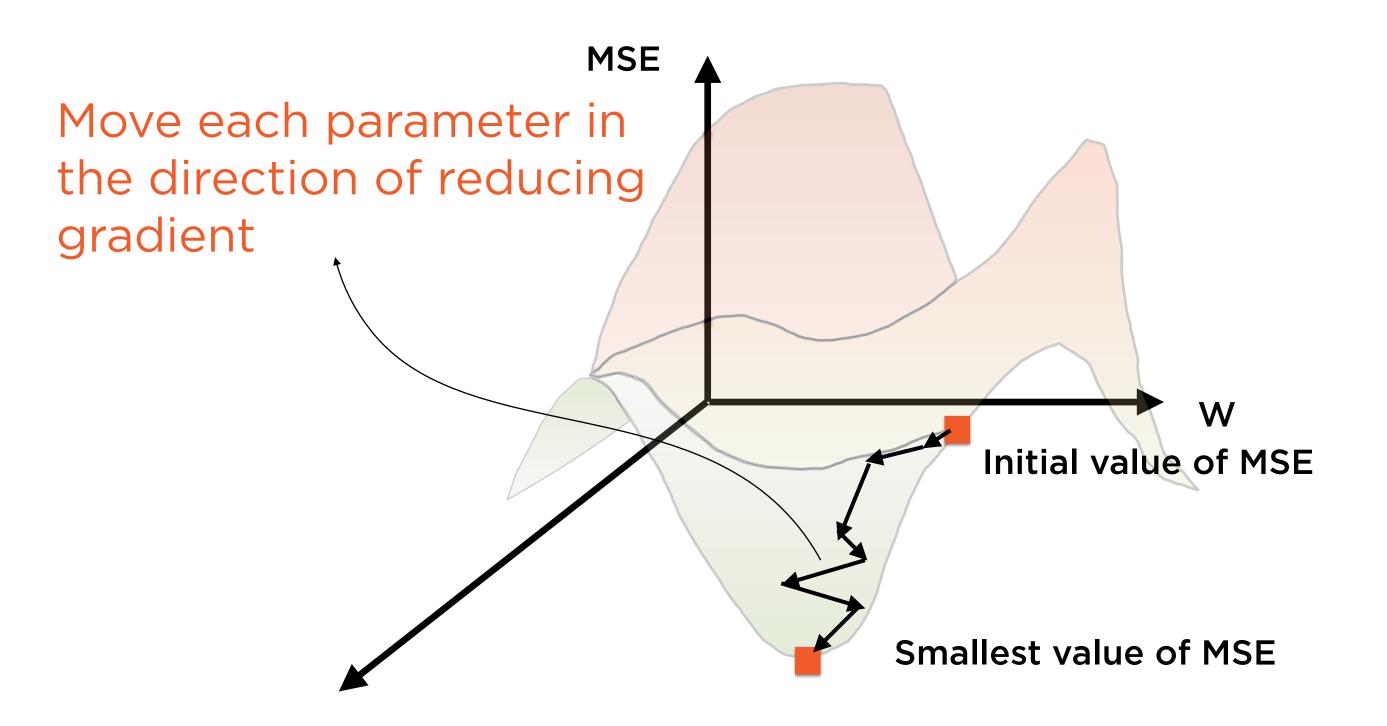
t+1
Parameters = Parameters - learning_rate x Gradient(θ)

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



```
t+1 t
Parameters = Parameters - learning_rate x Gradient(θ)
```

For Next Time Step: Update Parameter Values

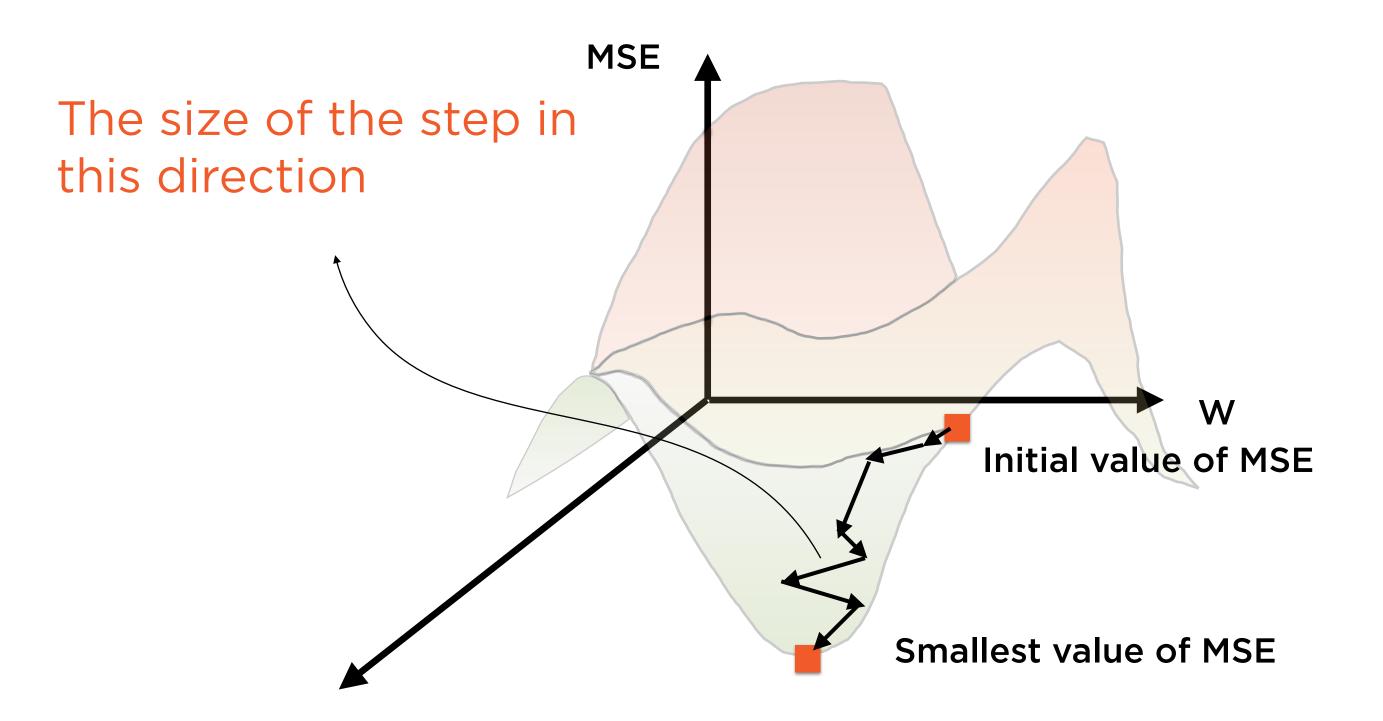
t+1
Parameters = Parameters - learning_rate x Gradient(θ)

For Next Time Step: Update Parameter Values

t+1
Parameters = Parameters - learning_rate x Gradient(θ)

For Next Time Step: Update Parameter Values

Learning Rate



Calculated in backward pass of time t

Parameters = Parameters - learning_rate x $Gradient(\theta)$

For Next Time Step: Update Parameter Values

```
Updated in backward pass
of time t...

Parameters = Parameters - learning_rate x Gradient(θ)
```

For Next Time Step: Update Parameter Values

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

Parameters = Parameters - learning_rate x Gradient(θ)
```

For Next Time Step: Update Parameter Values

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 mode auto-differentiation needs two passes

Back propagation is only required during training: in TF2.0, invoke the tape.gradient() method

Demo

Calculating gradients using tf.GradientTape()

Demo

Training a simple regression model

Use GradientTape to calculate gradients

Manually update model parameters using gradients

Summary

Training a neural network

Backpropagation and gradient descent

Gradients and their calculation

Training with gradient tape

Up Next:

Using the Sequential API in Keras