Building Neural Networks with scikit-learn

INTRODUCING NEURAL NETWORKS IN SCIKIT-LEARN

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

Overview

scikit-learn support for neural networks

scikit-learn vs. deep learning frameworks

Perceptrons and neurons

Multi-layer perceptrons (MLPs) and neural networks

Training a neural network

Prerequisites and Course Outline

Prerequisites

Basic Python programming

Prior ML exposure, basic ML workflow

Building and training ML models in scikit-learn

Prerequisite Courses

Building Your First scikit-learn Solution

Building Regression Models with scikit-learn

Building Classification Models with scikit-learn

Course Outline

Neural networks in scikit-learn

Regression and classification with neural networks

Text and image classification

Dimensionality reduction with Restricted Boltzmann Machines

Neural Networks in scikit-learn

Support for neural networks in scikit-learn is currently quite limited

scikit-learn vs. Other Frameworks

scikit-learn	TensorFlow, PyTorch
Most popular library for general purpose ML	Widely used libraries that specialize in deep learning
Vast array of estimators for classification, regression, clustering	Relatively small number of algorithmic estimators for those problems
Implemented using traditional ML algorithms	Very little support for traditional ML algorithms
Very limited support for building neural networks	Entirely focused on building neural networks

Neural Networks on scikit-learn

scikit-learn	TensorFlow, PyTorch
Work directly with Pandas data frames and NumPy arrays	Work with special data types called tensors for multidimensional arrays
No specialized GPU support	Help leverage power of GPUs
Not suited to distributed training	Extensive support for distributed training

Neural Networks on scikit-learn

scikit-learn	TensorFlow, PyTorch
Limited number of neural network building blocks	Large numbers of neuron types, activation functions, loss functions
Support for just fully-connected neural networks with regularization	Building blocks to support different kinds of neural network layers
Impossible to build complex RNNs, CNNs	Relatively simple to build complex RNN and CNN architectures
No pre-trained models for transfer learning	Impressive array of pre-trained models available for transfer learning

Neural Networks in scikit-learn

Supervised

Unsupervised

Neural Networks in scikit-learn

Multi-layer Perceptrons (MLP)

Restricted Boltzmann Machines (RBM)

Perceptrons and Neurons

Neural Networks in scikit-learn

Supervised

Unsupervised

Neural Networks in scikit-learn

Multi-layer Perceptrons (MLP)

Restricted Boltzmann Machines (RBM)

Neural Networks in scikit-learn

Multi-layer Perceptrons (MLP)

Restricted Boltzmann Machines (RBM)

Perceptron

Simplest Artificial Neural Network architecture

Originally invented in 1957 by Frank Rosenblatt

Precursor to the neuron used today

Perceptron

Calculates the weighted sum of inputs

Applies a step function with a threshold

Output - positive

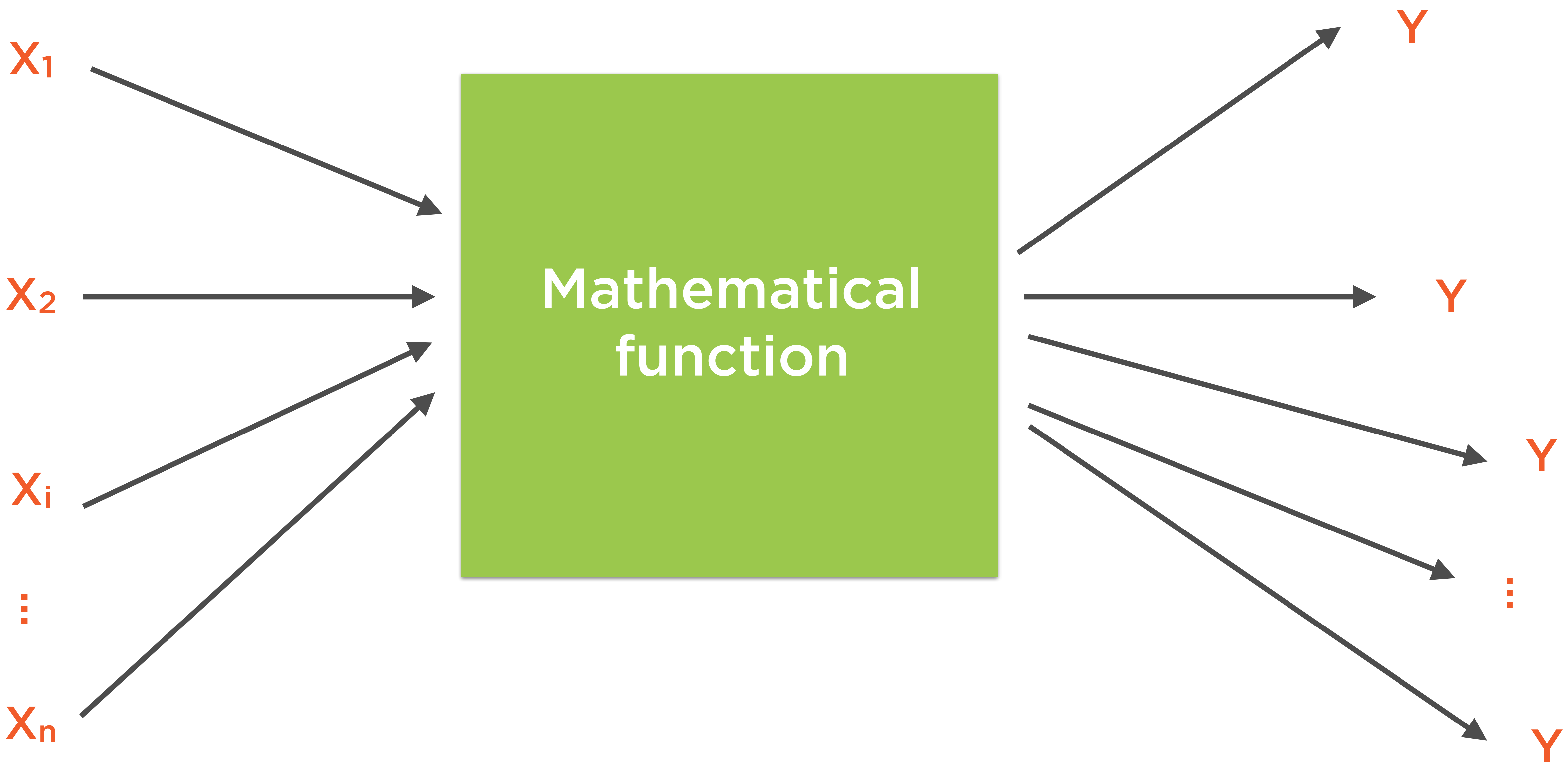
- if value above threshold

Output - negative

- if value below threshold

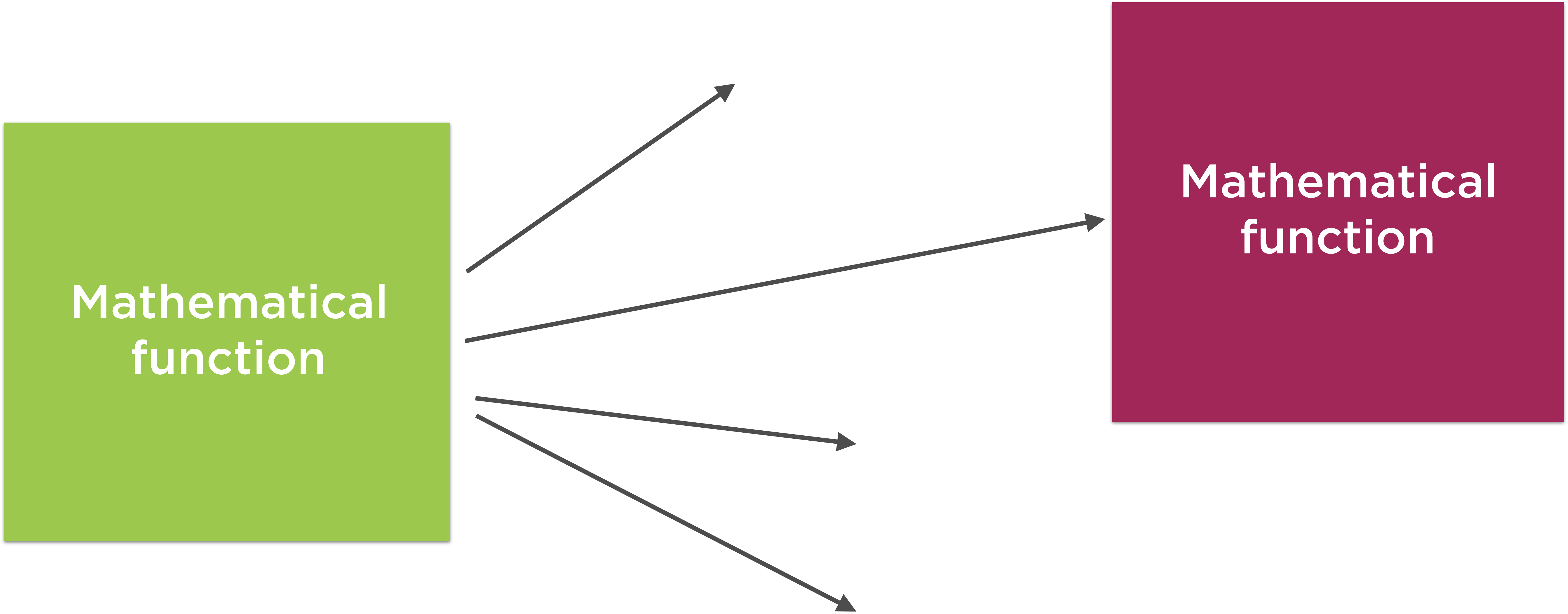
Perceptron ~ Neuron with a step activation function

Operation of a Single Neuron



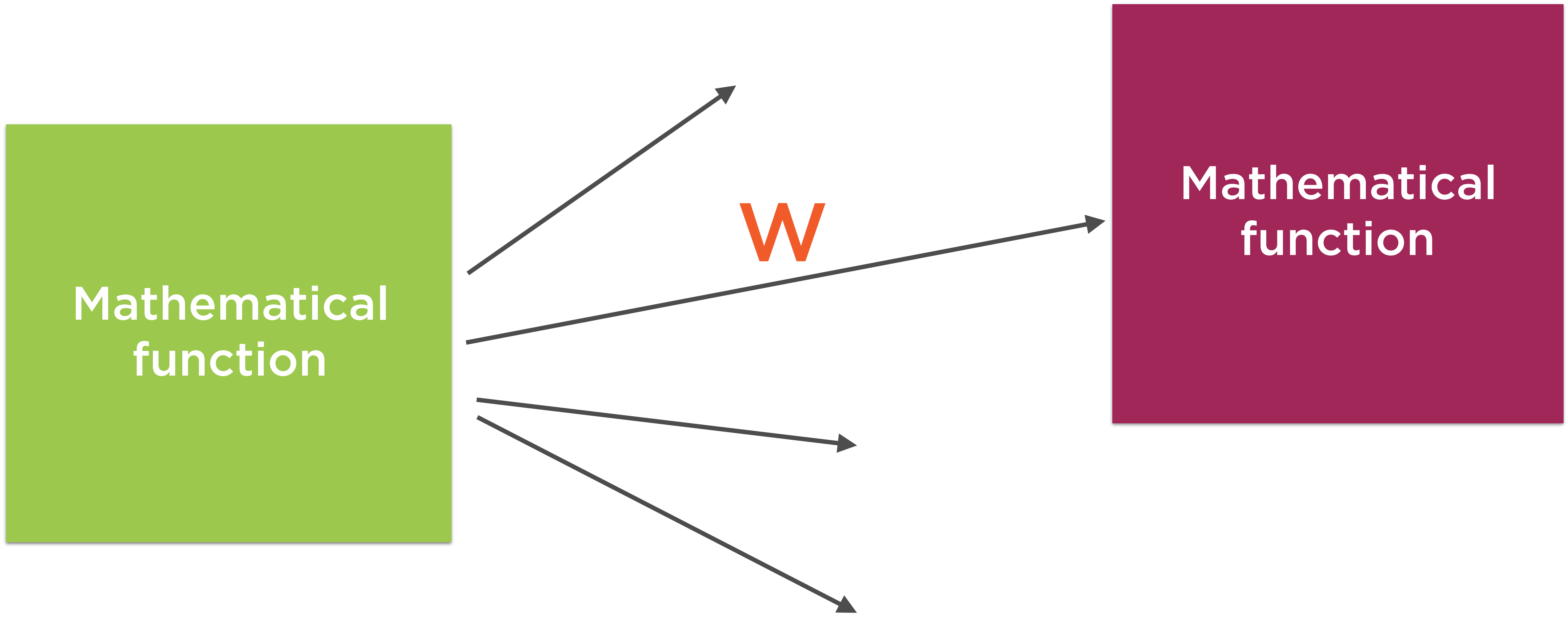
For an active neuron a change in inputs should trigger a corresponding change in the outputs

Operation of a Single Neuron



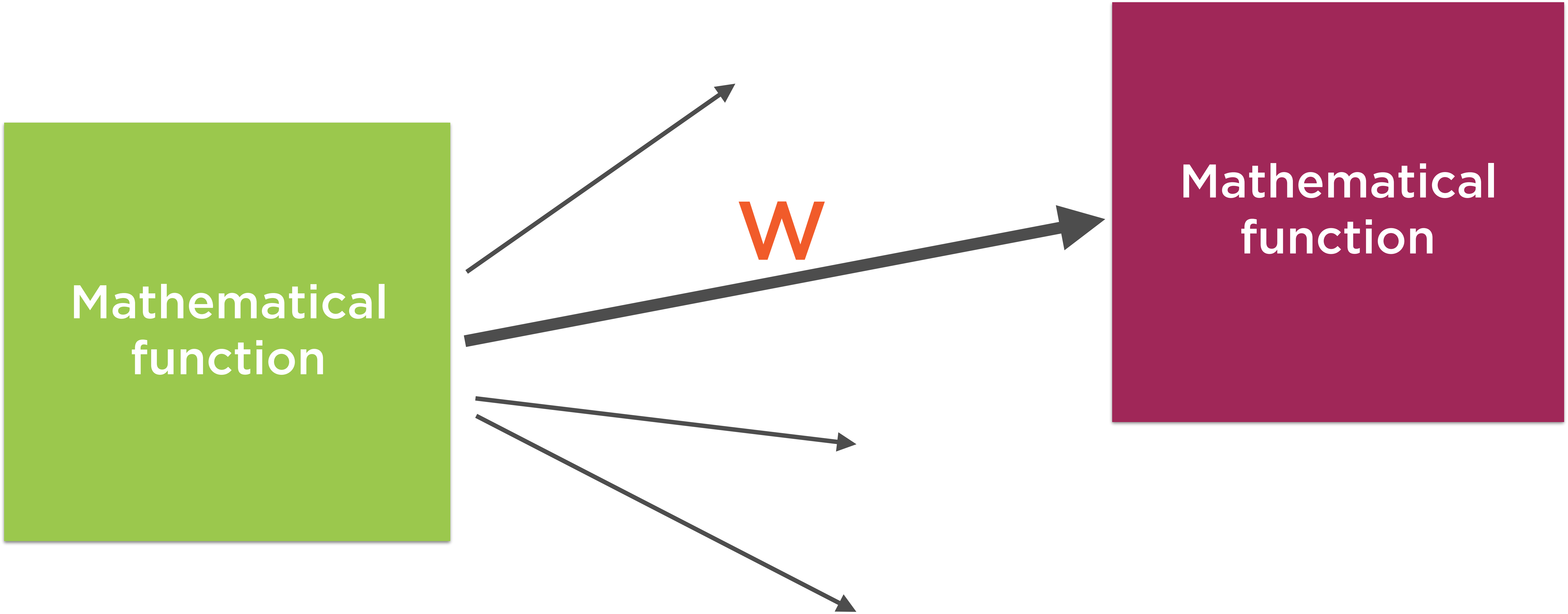
The outputs of neurons feed into the neurons from the next layer

Operation of a Single Neuron



Each connection is associated with a weight

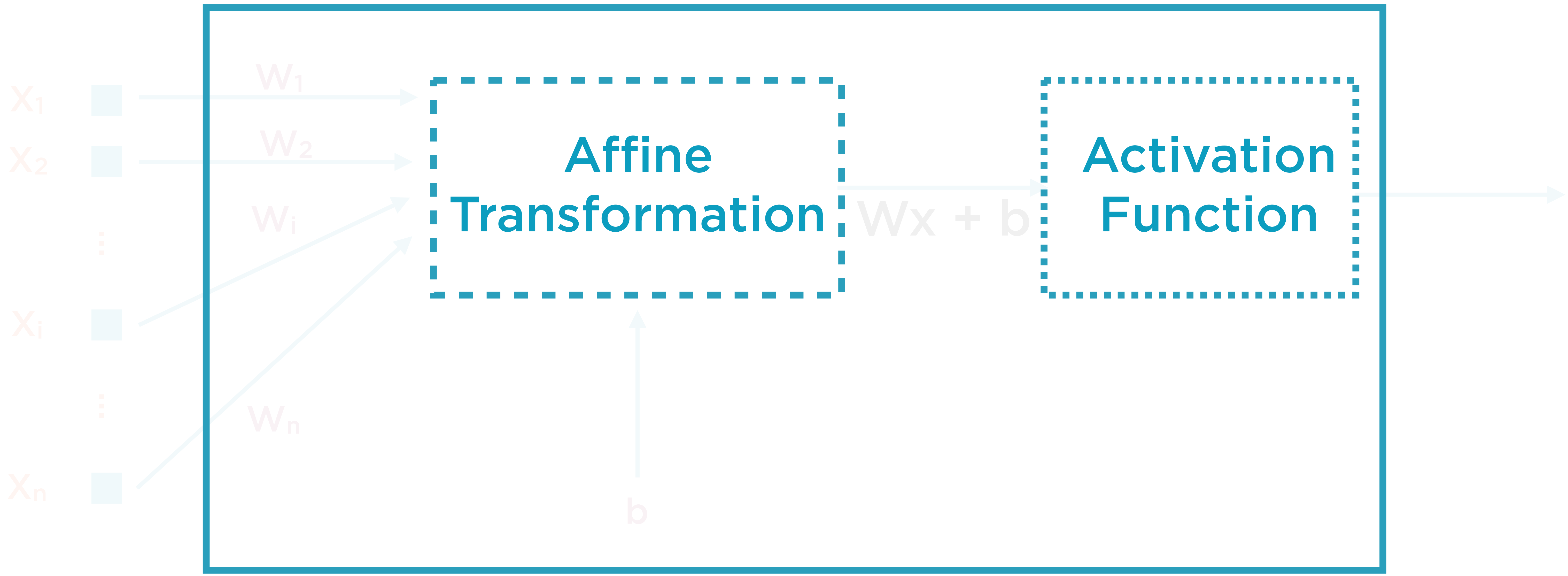
Operation of a Single Neuron



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

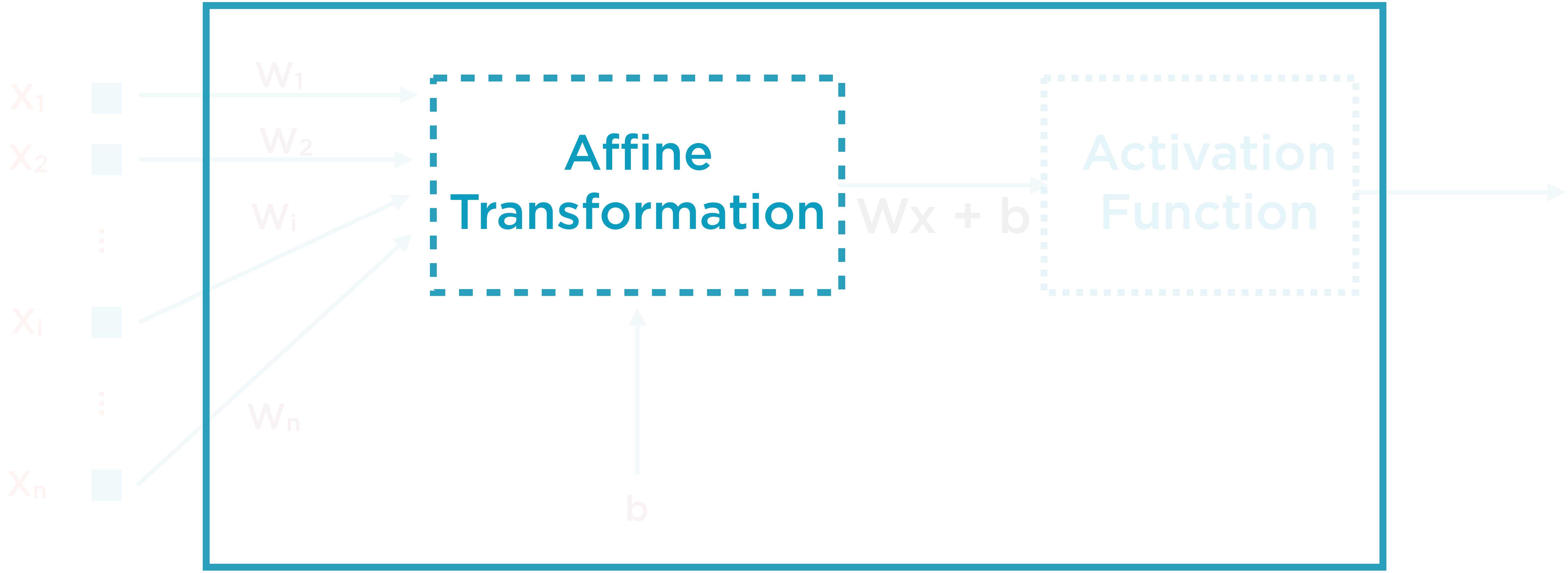
W increases

Operation of a Single Neuron



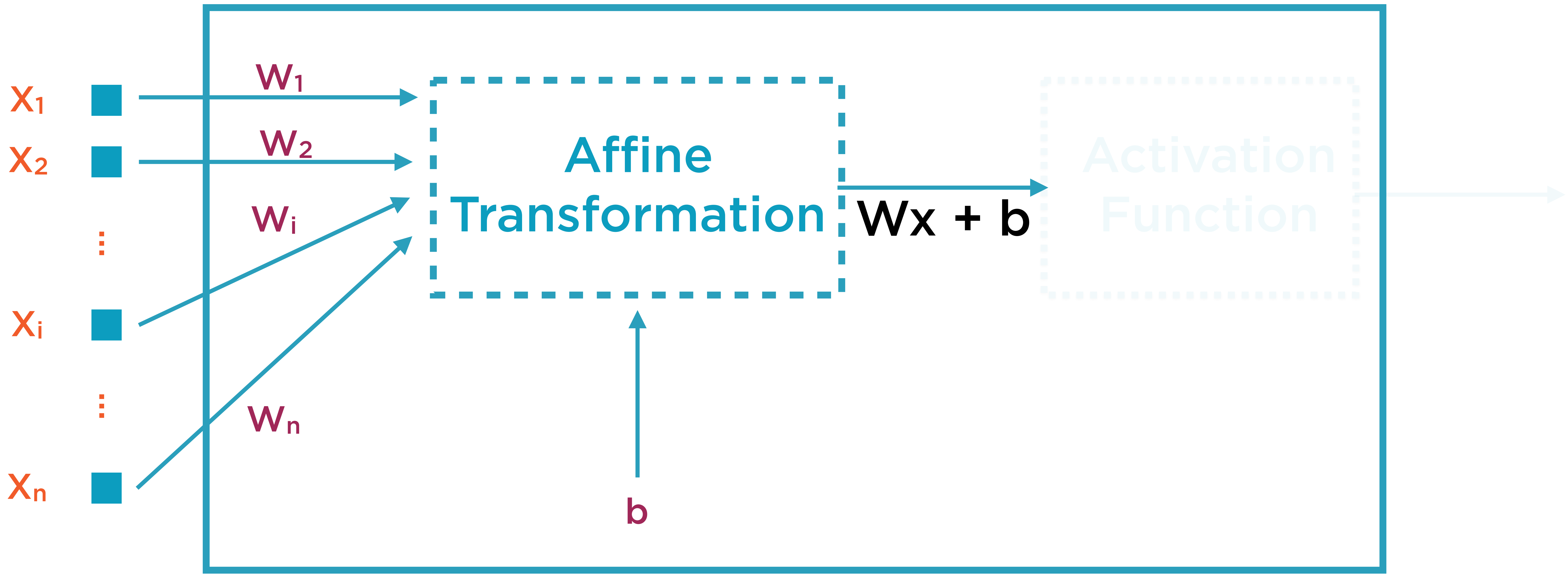
Each neuron only applies two simple functions to its inputs

Affine Transformation



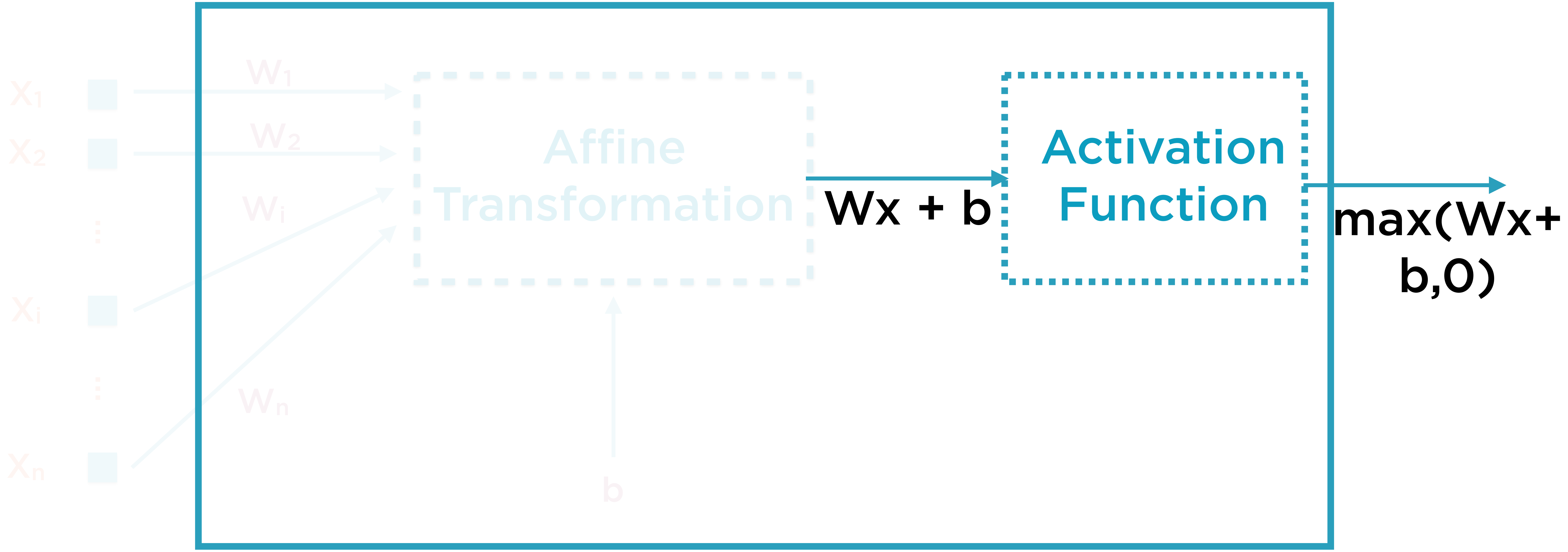
The affine transformation alone can only learn linear relationships between the inputs and the output

Affine Transformation



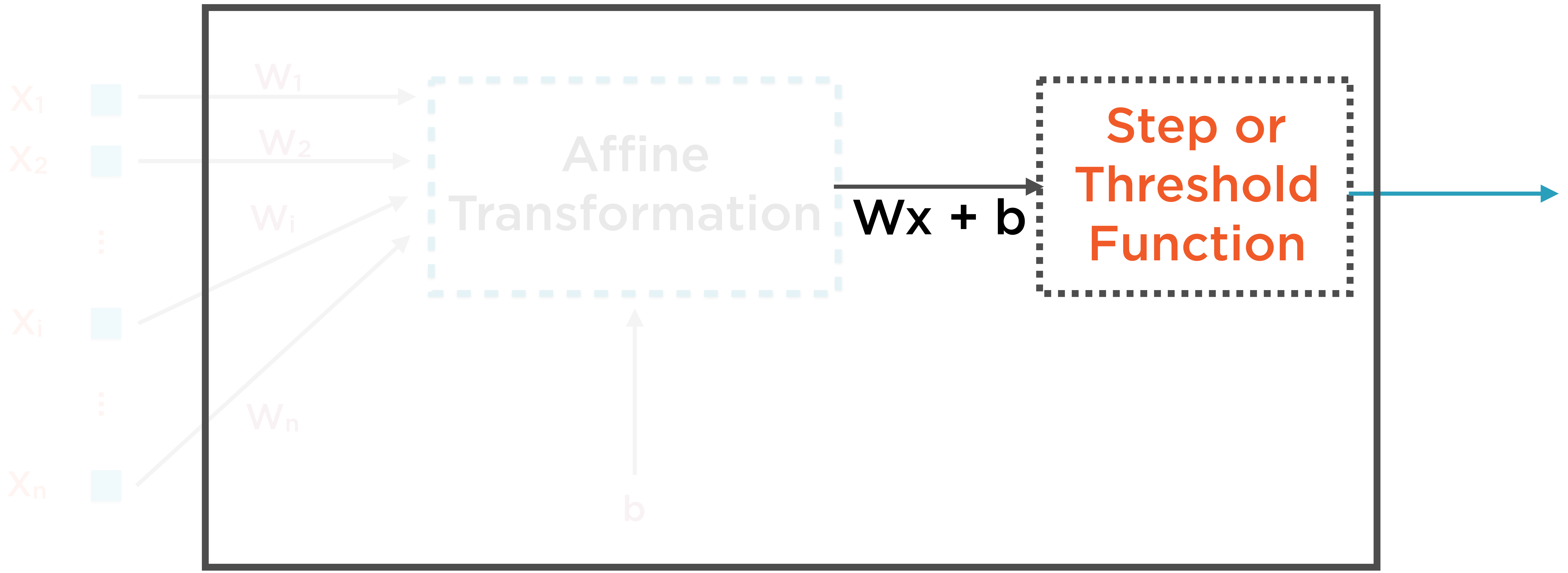
The affine transformation is just a weighted sum with a bias added: $W_1x_1 + W_2x_2 + \ldots + W_nx_n + b$

Activation Function



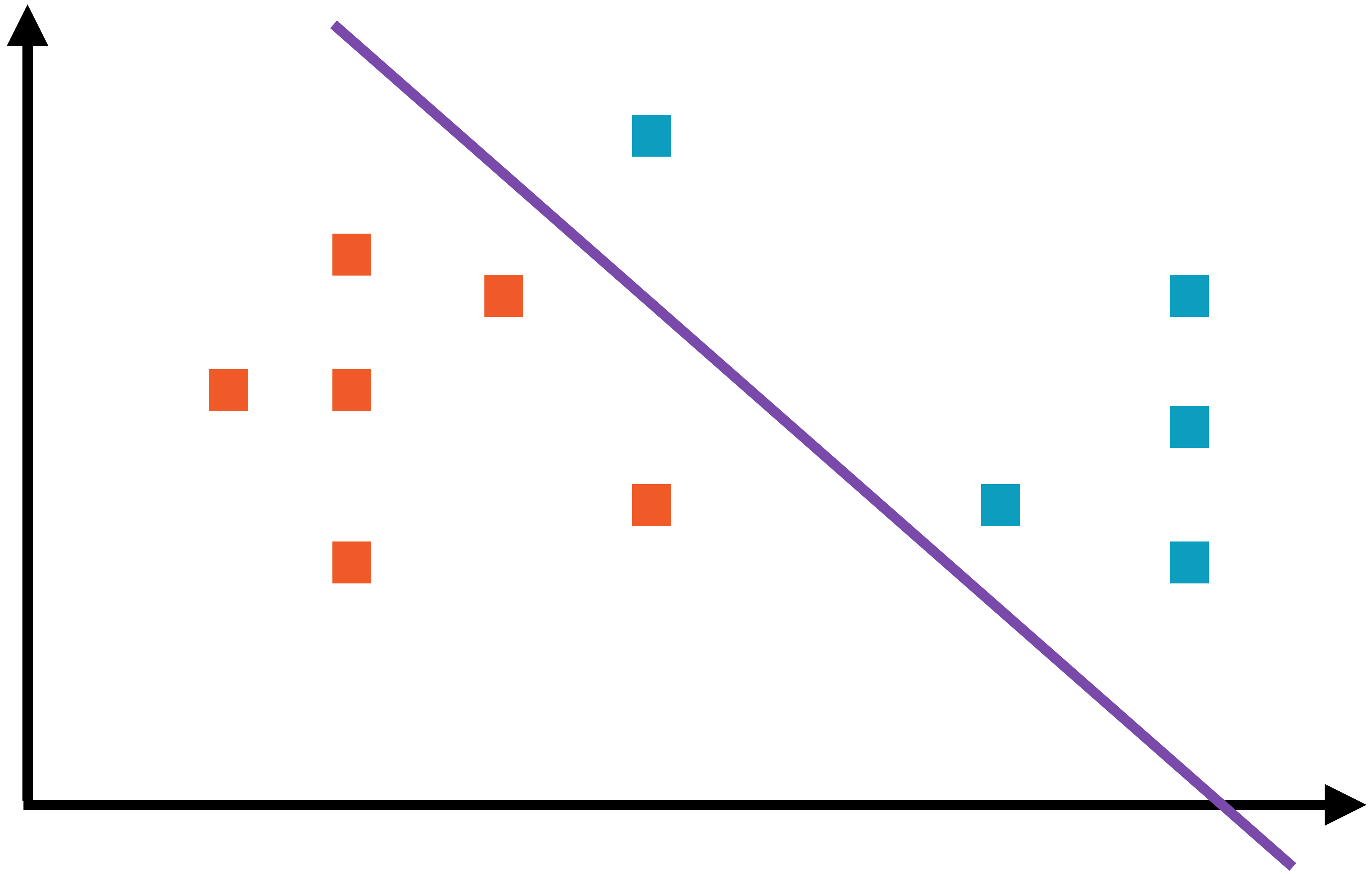
A function which helps discover non-linear relationships

Activation in a Perceptron



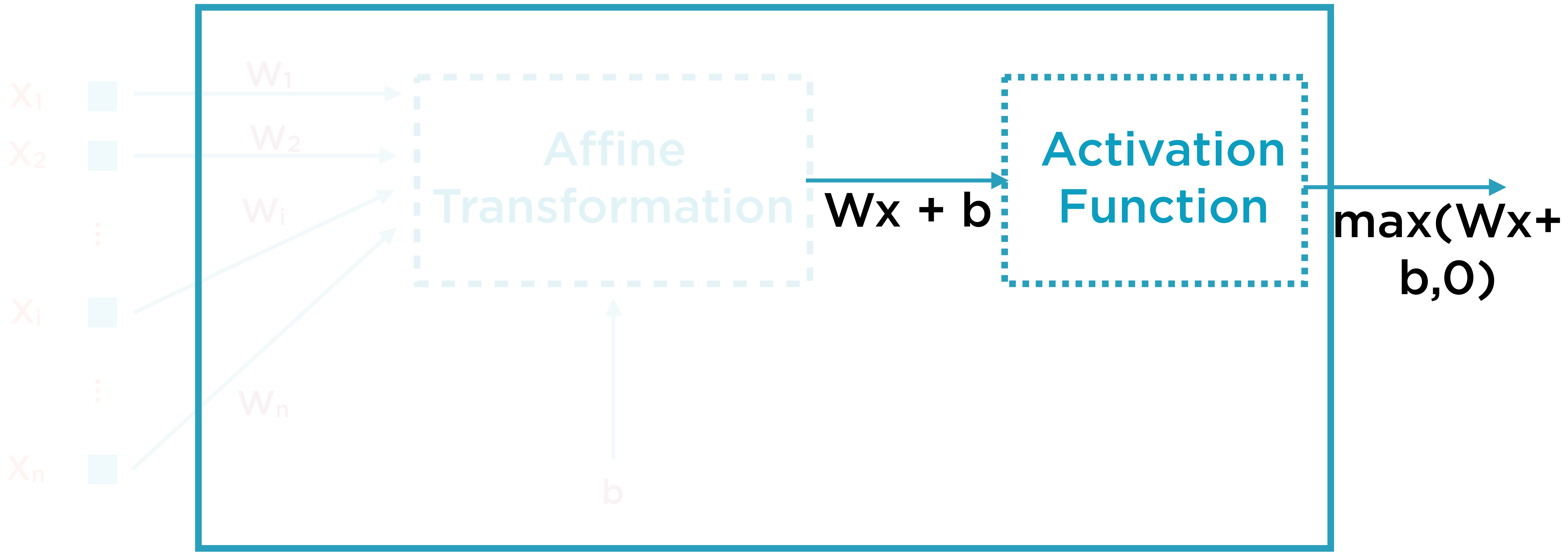
This combination allowed the perceptron to only work with linearly separable data

Linearly Separable Data



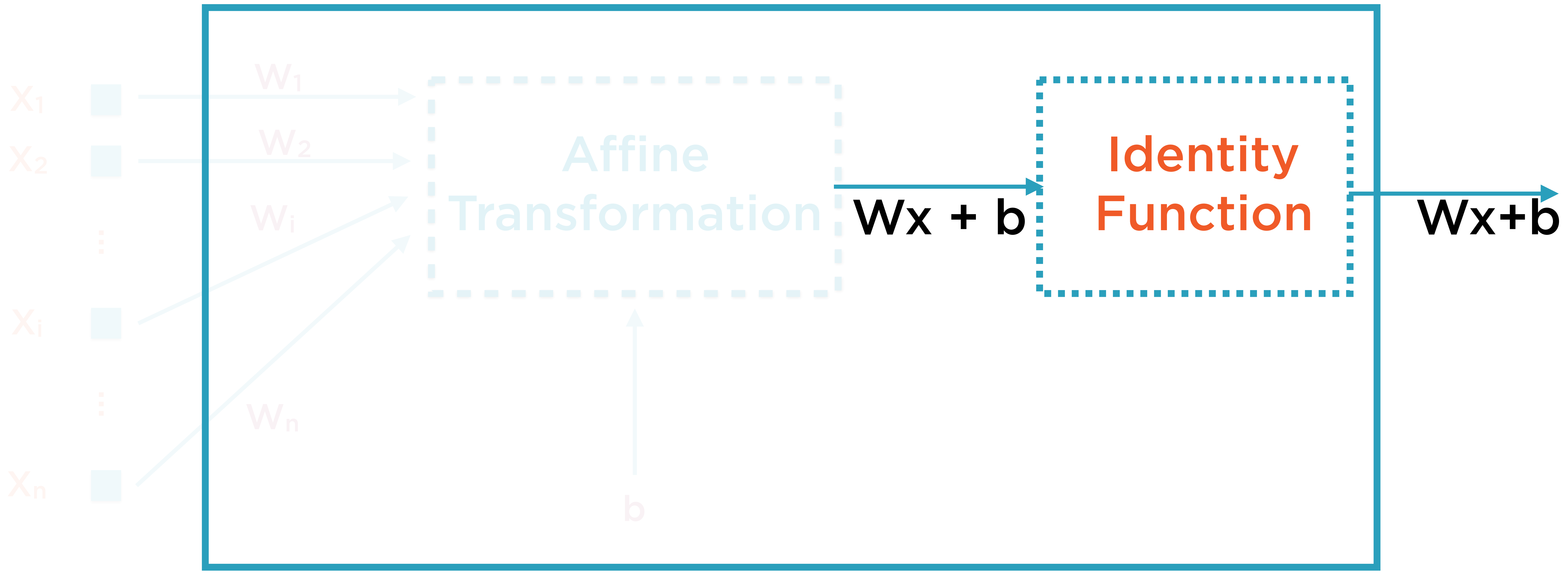
Linear boundary between classes

Activation Function



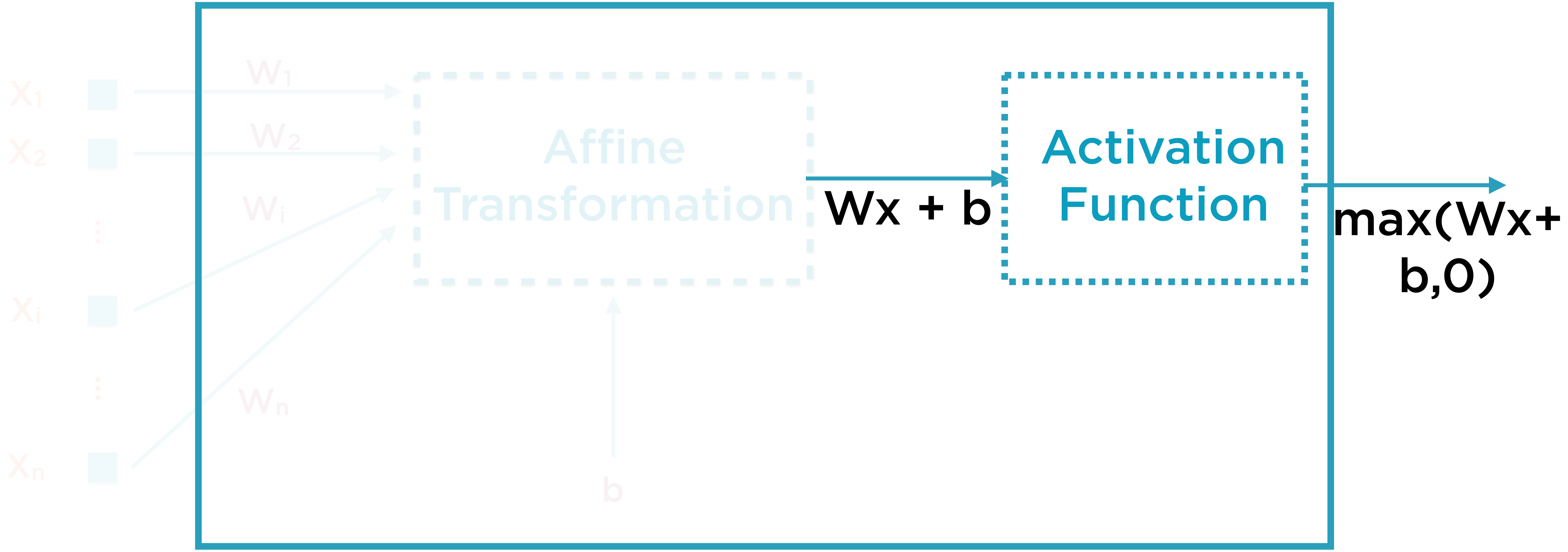
A neuron works with many more activation functions which help it learn more complex relationships

Linear Neuron



When the activation function is the identity function, the neuron is often referred to as a linear neuron

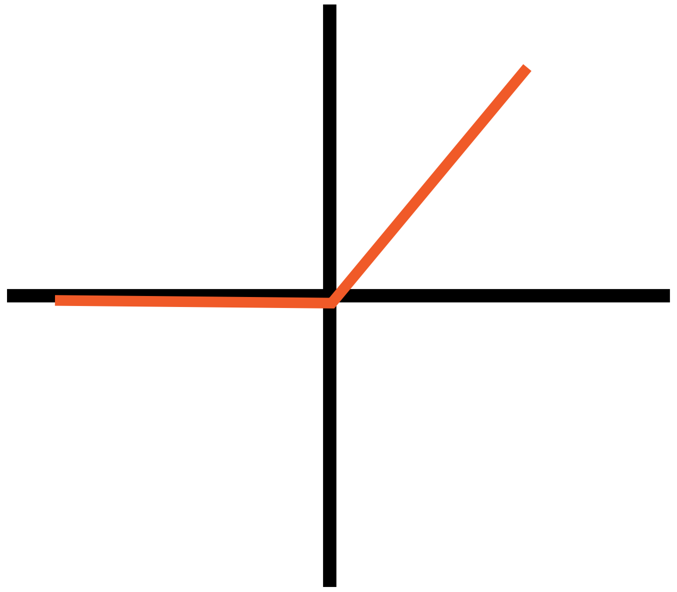
Activation Function



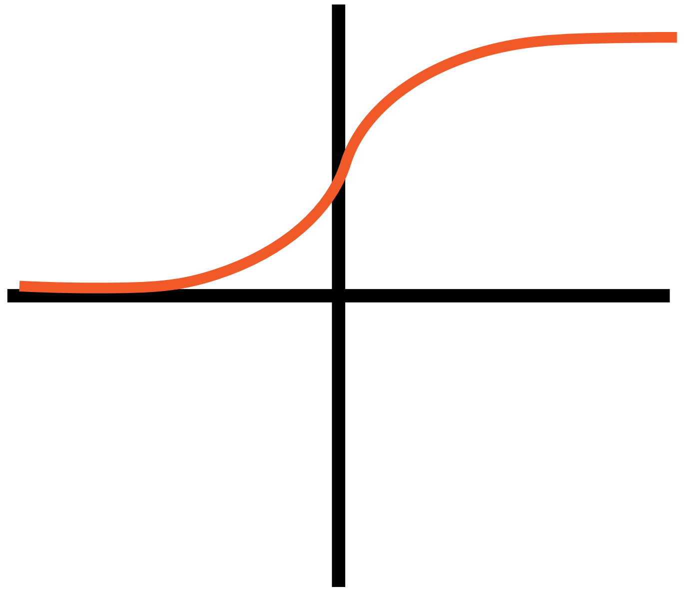
The combination of the affine transformation and the activation function can learn any arbitrary relationship

Common Activation Functions

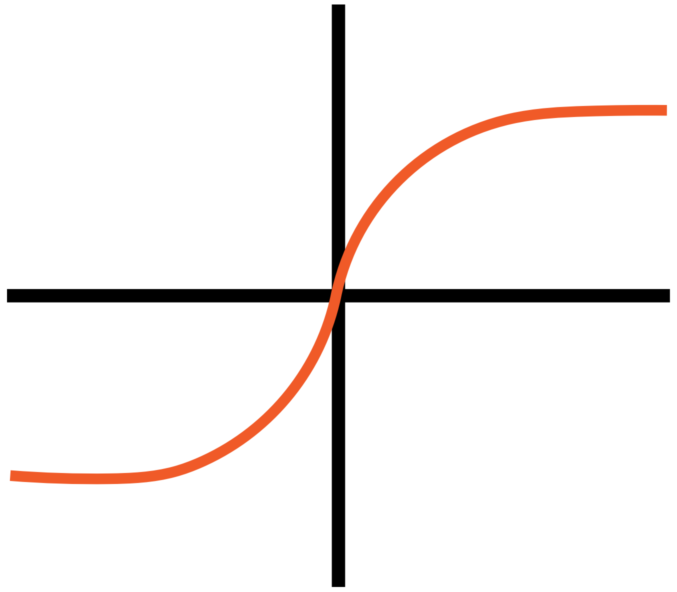
ReLU



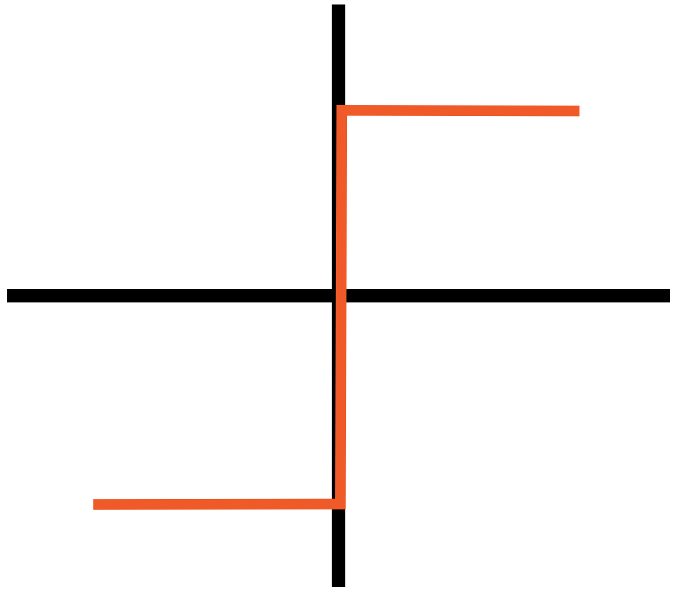
logit



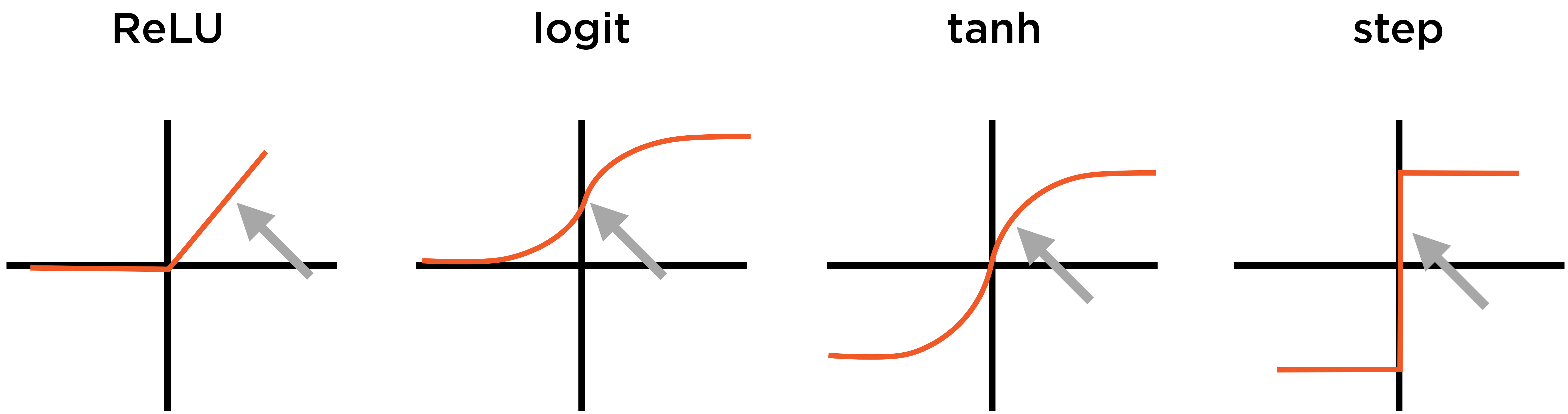
tanh



step

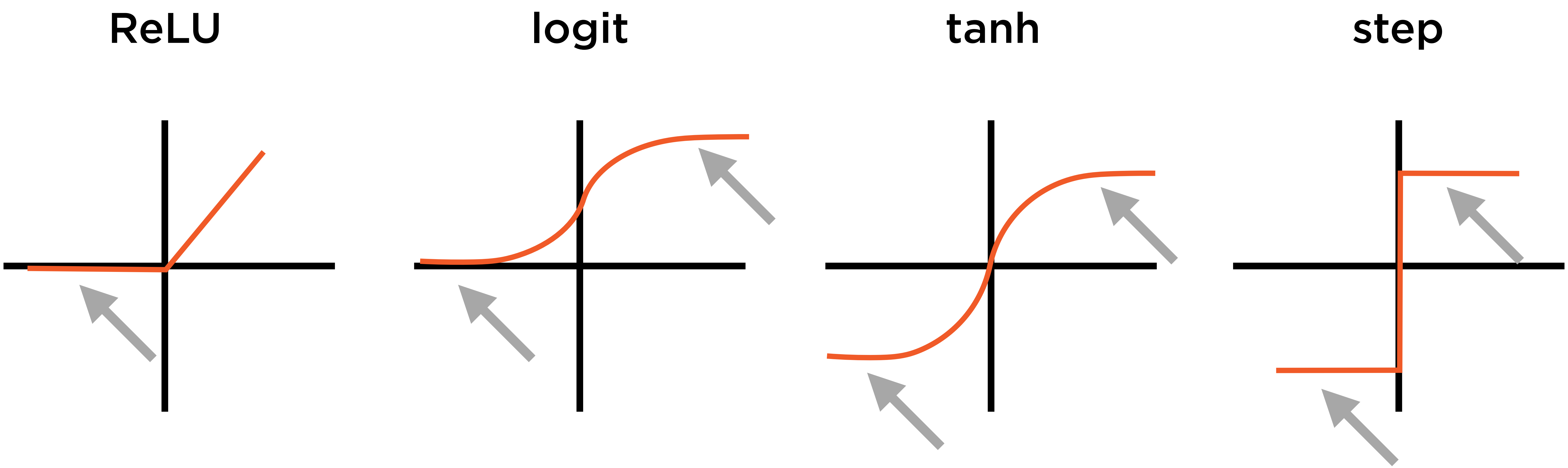


Active Region



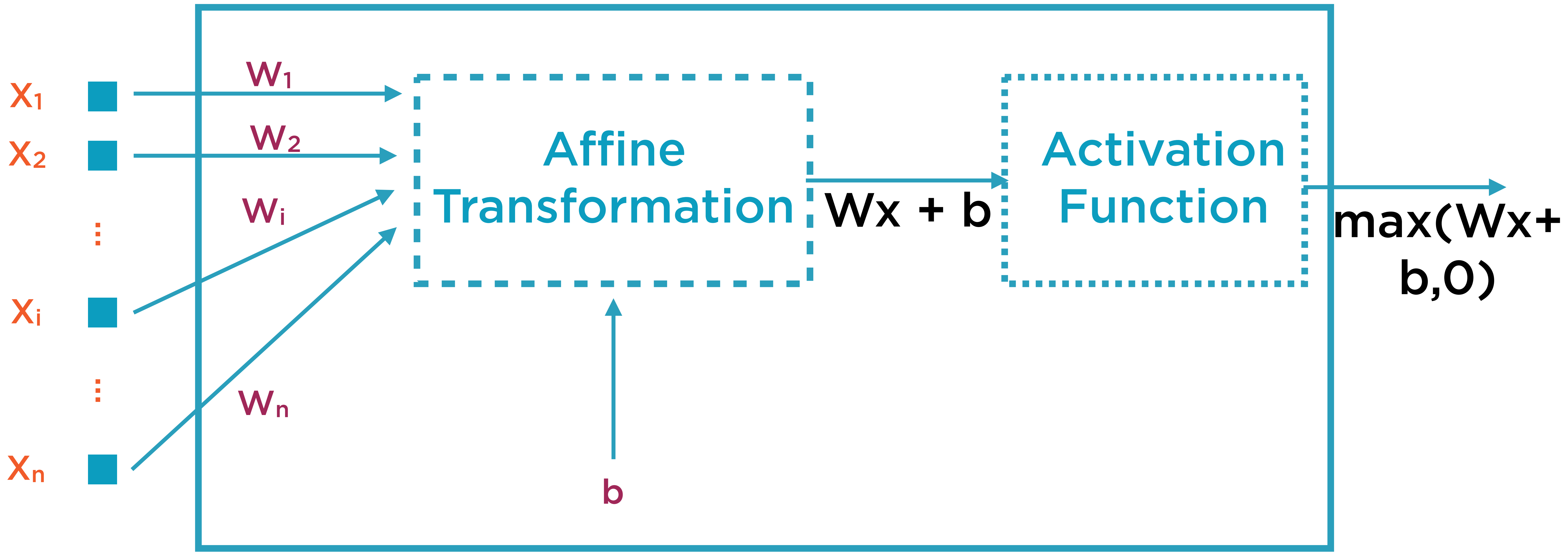
Notice how activation functions have a gradient, this gradient allows them to be sensitive to input changes

Saturation



In order to train and adjust the weights of the neural network the activation functions should operate in their active region

Neuron as a Learning Unit

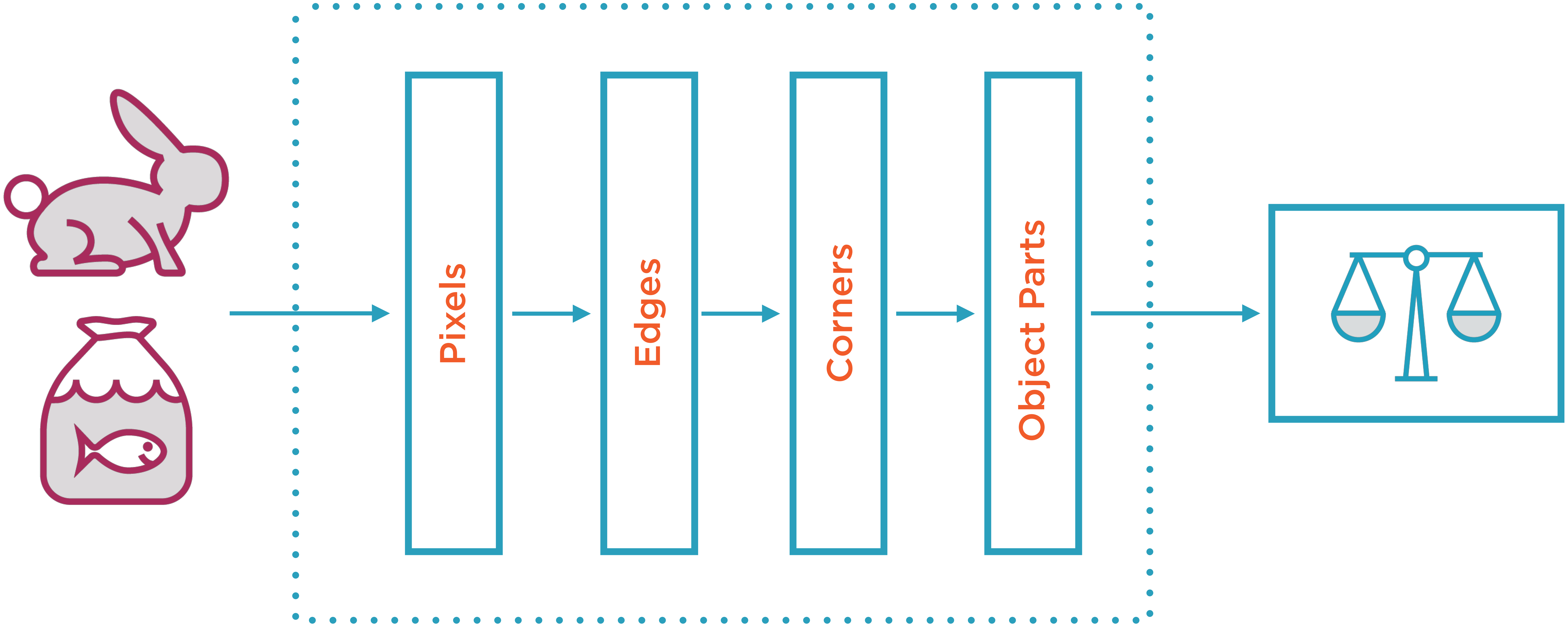


Many of these simple neurons arranged in layers can do magical stuff

Multi-layer Perceptrons and Neural Networks

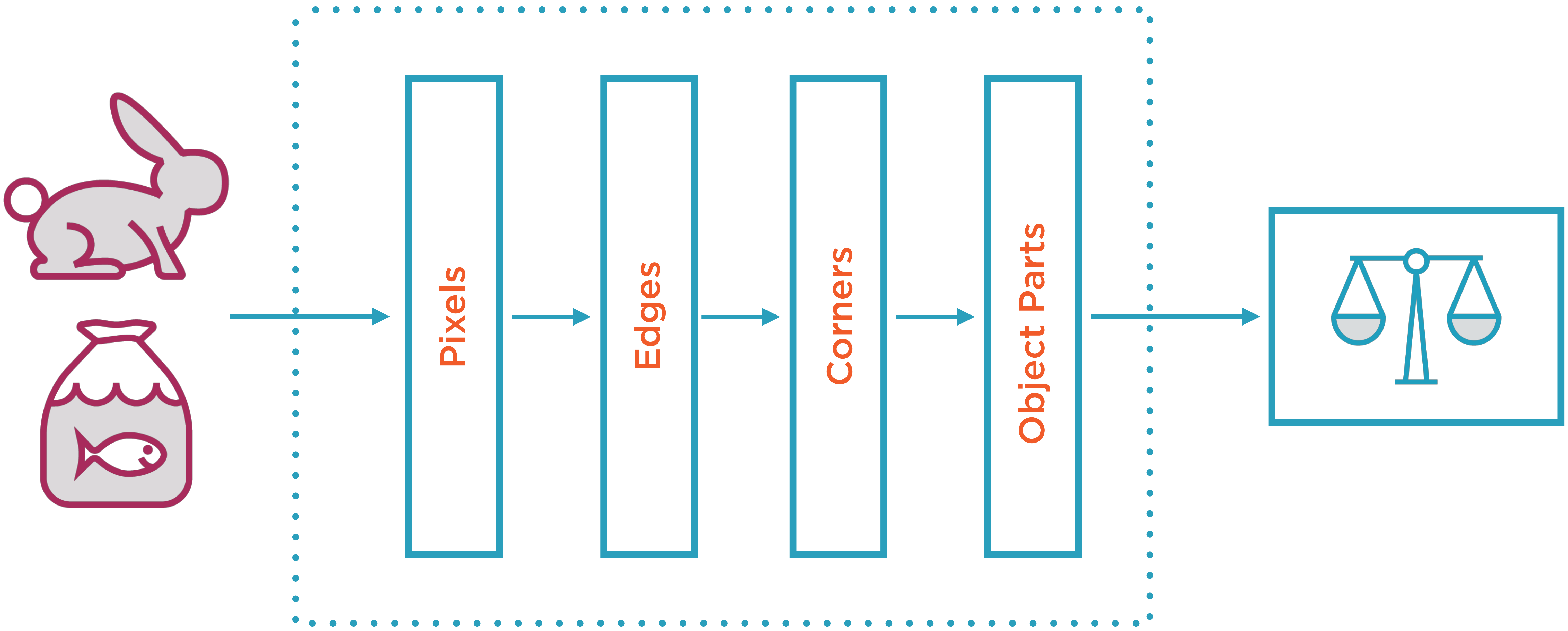
Multi-layer Perceptron ~ Feed-forward Neural Network

Neural Network



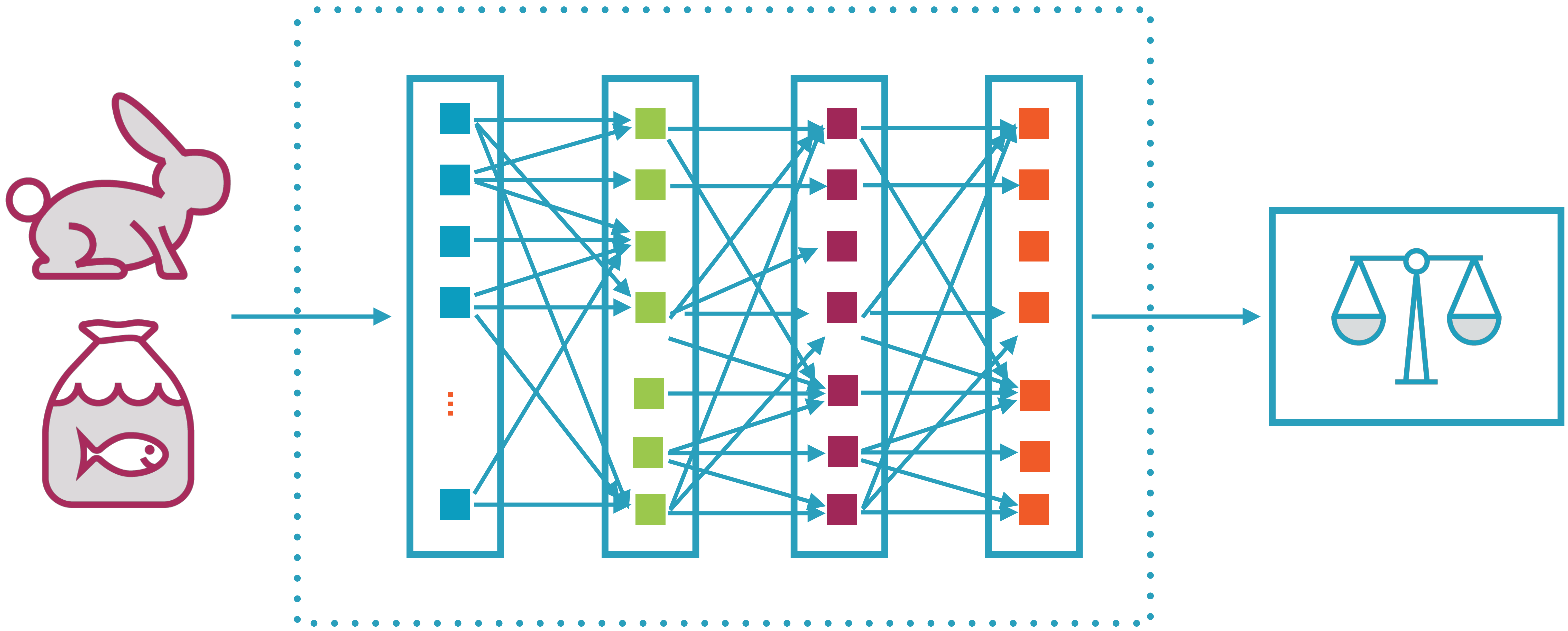
Comprised of neurons i.e. active learning units arranged in layers

Layers in a Neural Network



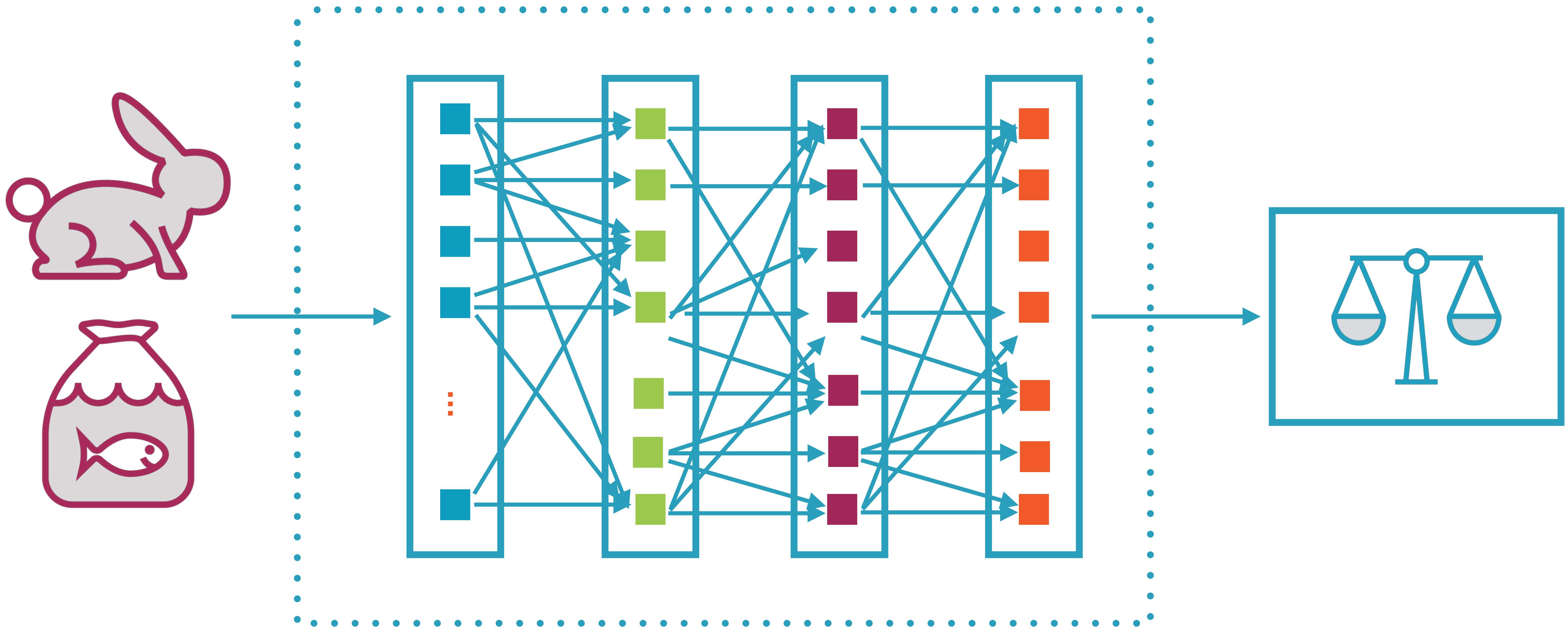
Groups of neurons that perform similar functions are aggregated into layers

Network of Neurons



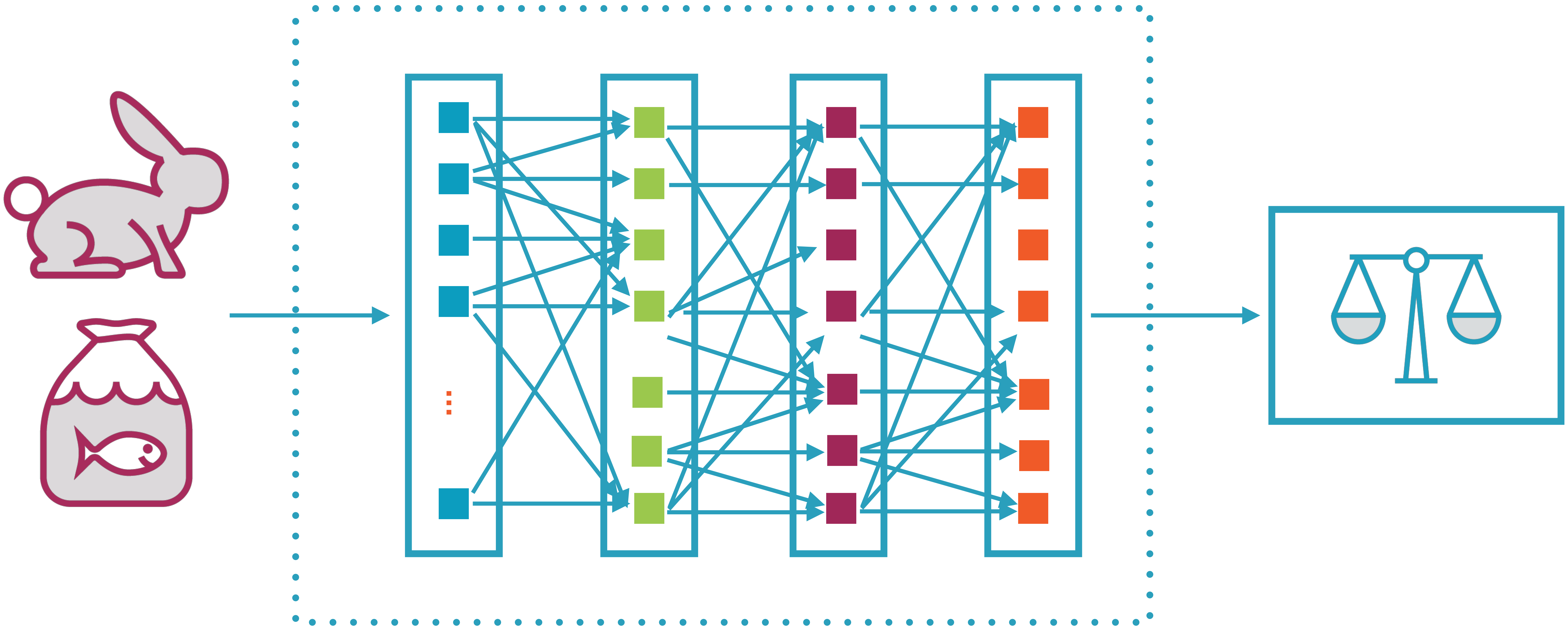
Each layer consists of individual interconnected neurons

Network of Neurons



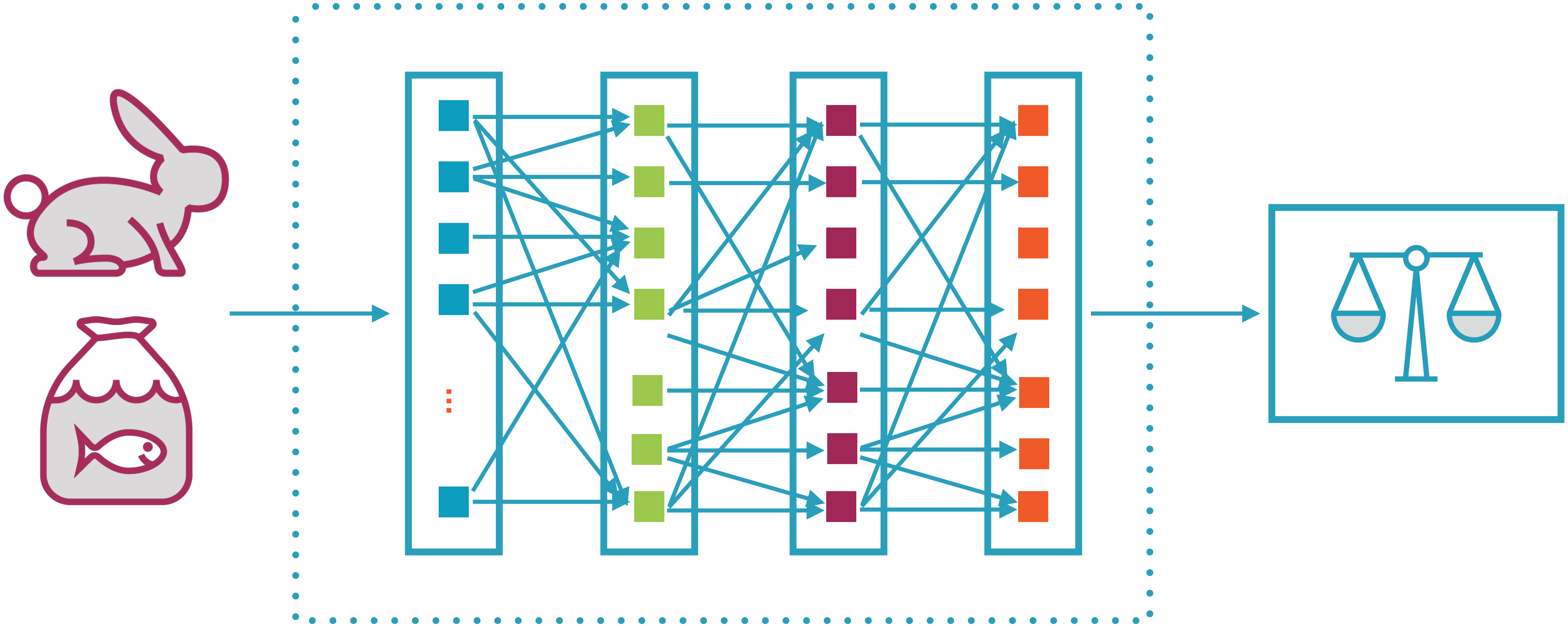
Neurons receive input from neurons in the previous layer

Network of Neurons

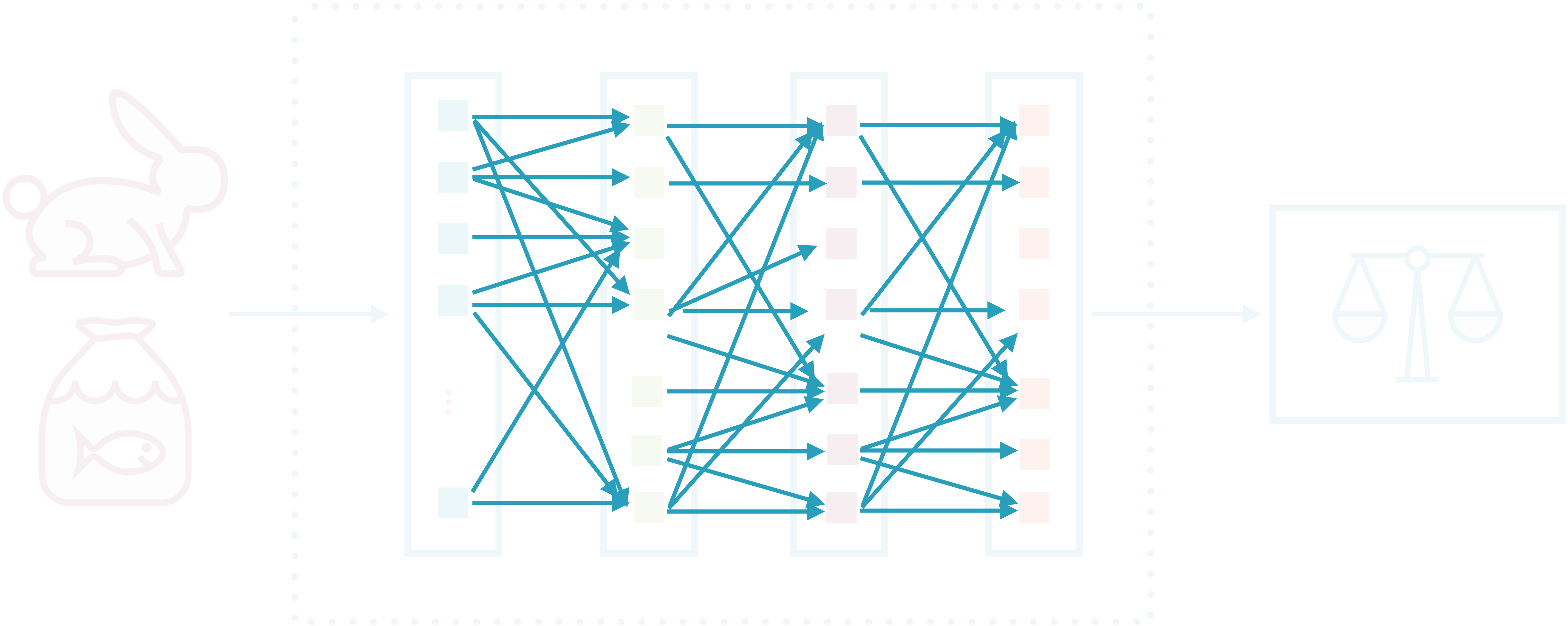


And pass their output on to neurons in the next layer

Feed-forward Networks

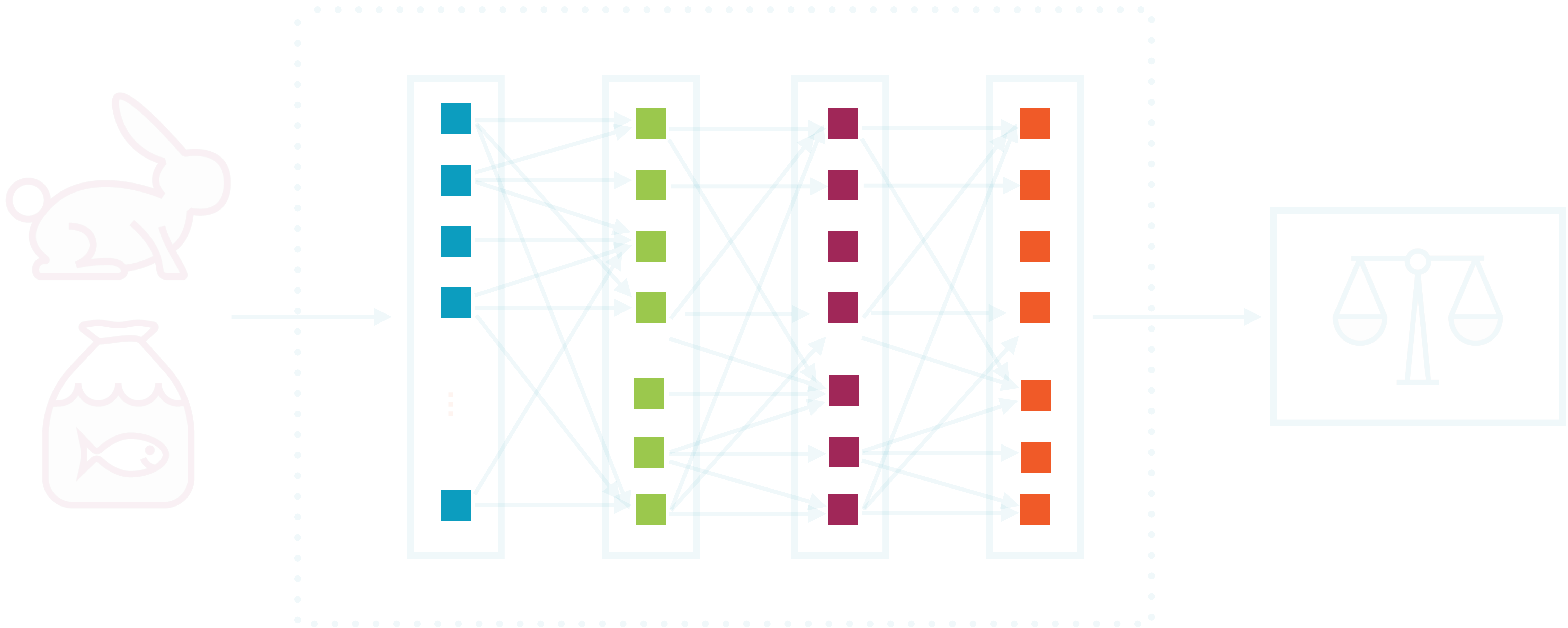


Feed-forward Networks



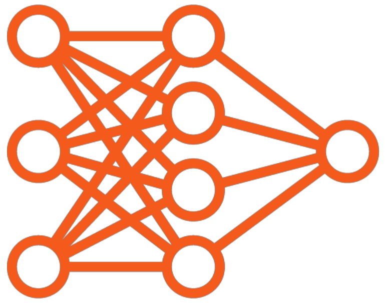
Feed-forward networks: Information moves forward through the layers

Feed-forward Networks

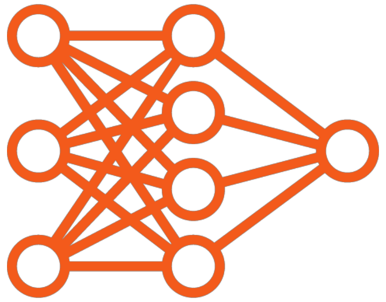


No connection exists between neurons in the same layer

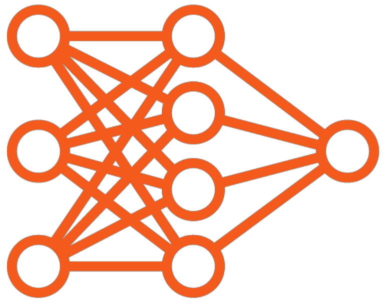
Deep Learning with Neural Networks



Directed computation graphs “learn” relationships between data



The more complex the graph, the more relationships it can “learn”



“Deep” Learning: Depth of the computation graph

```
y = Wx + b
```

“Learning” Regression

Regression can be reverse-engineered by a single neuron

```
def doSomethingReallyComplicated(x1,x2…):
   …
   …
   …
   return complicatedResult
```

“Learning” Arbitrarily Complex Functions

Adding layers to a neural network can “learn” (reverse-engineer) pretty much anything

Training a Neural Network

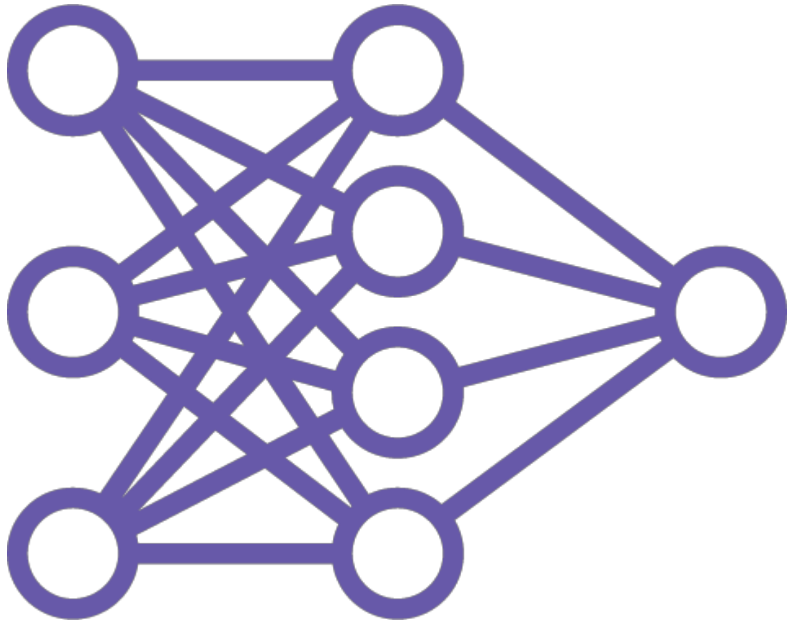
Neurons

The nodes in the computation graph are simple entities called neurons

Each neuron performs very simple operations on data

The neurons are connected in very complex, sophisticated ways

Neural Network

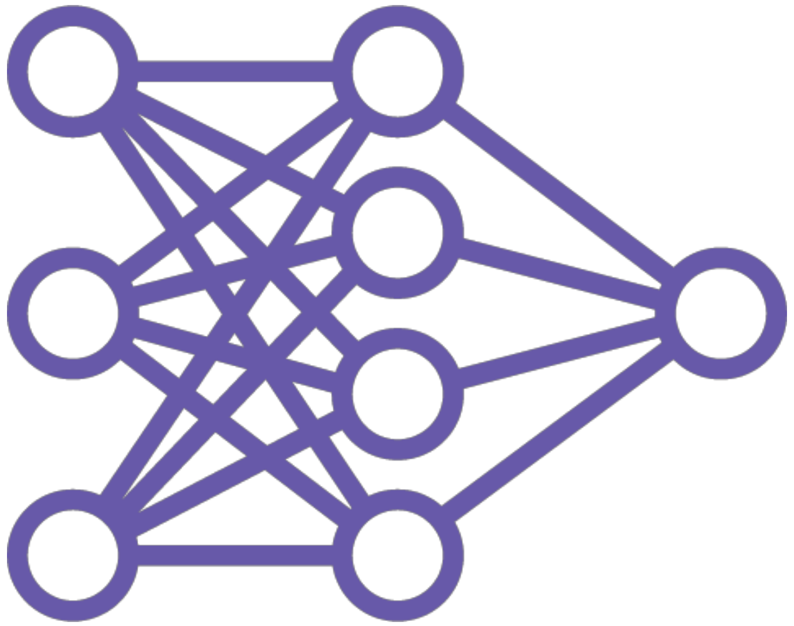


The complex interconnections between simple neurons

Different network configurations => different types of neural networks

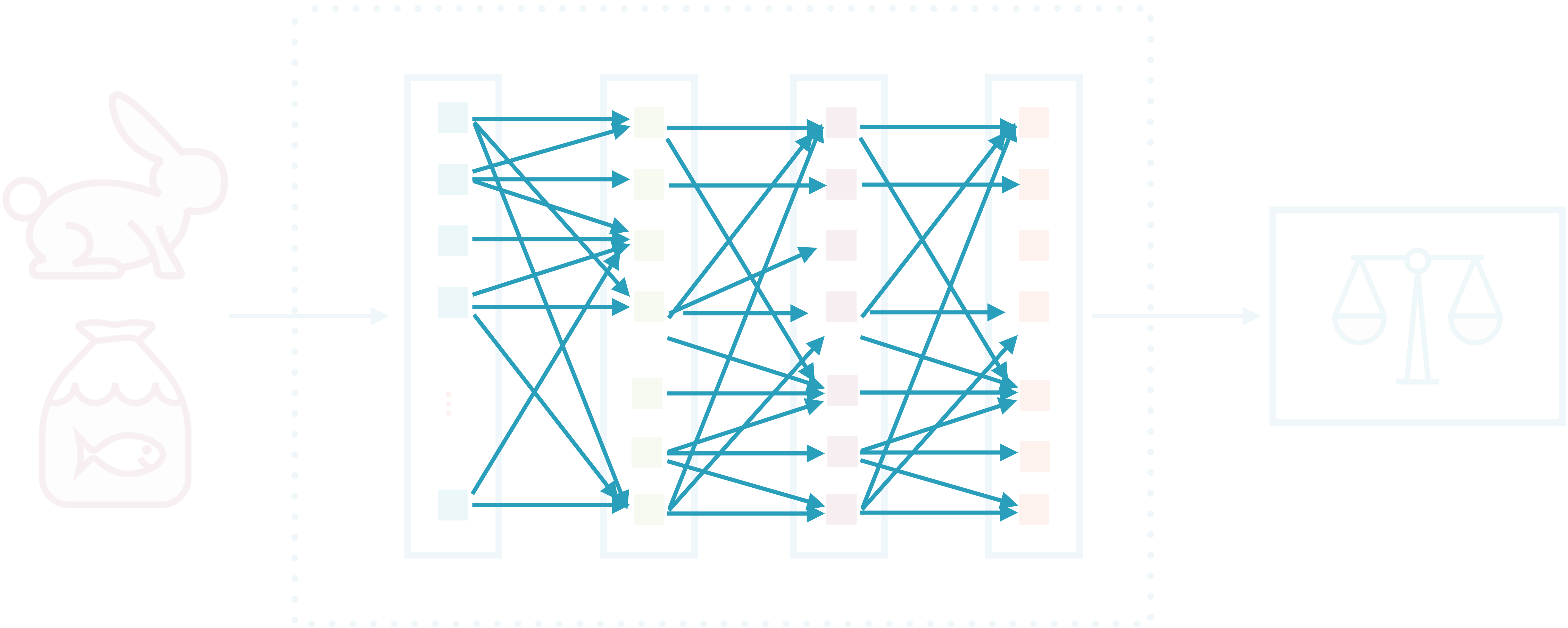
- Convolutional
- Recurrent

Neural Network



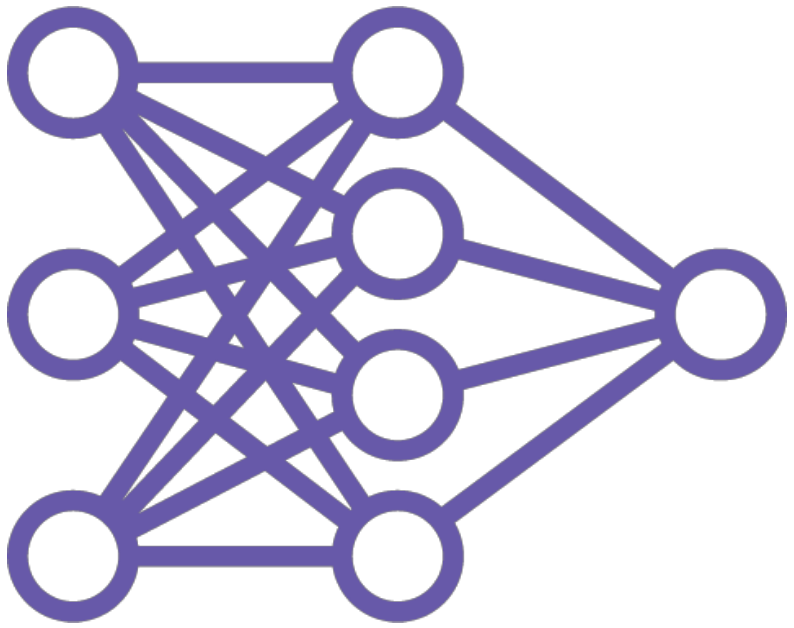
Groups of neurons that perform similar functions are aggregated into layers

Complex Interconnections



Neurons in a neural network can be connected in very complex ways...

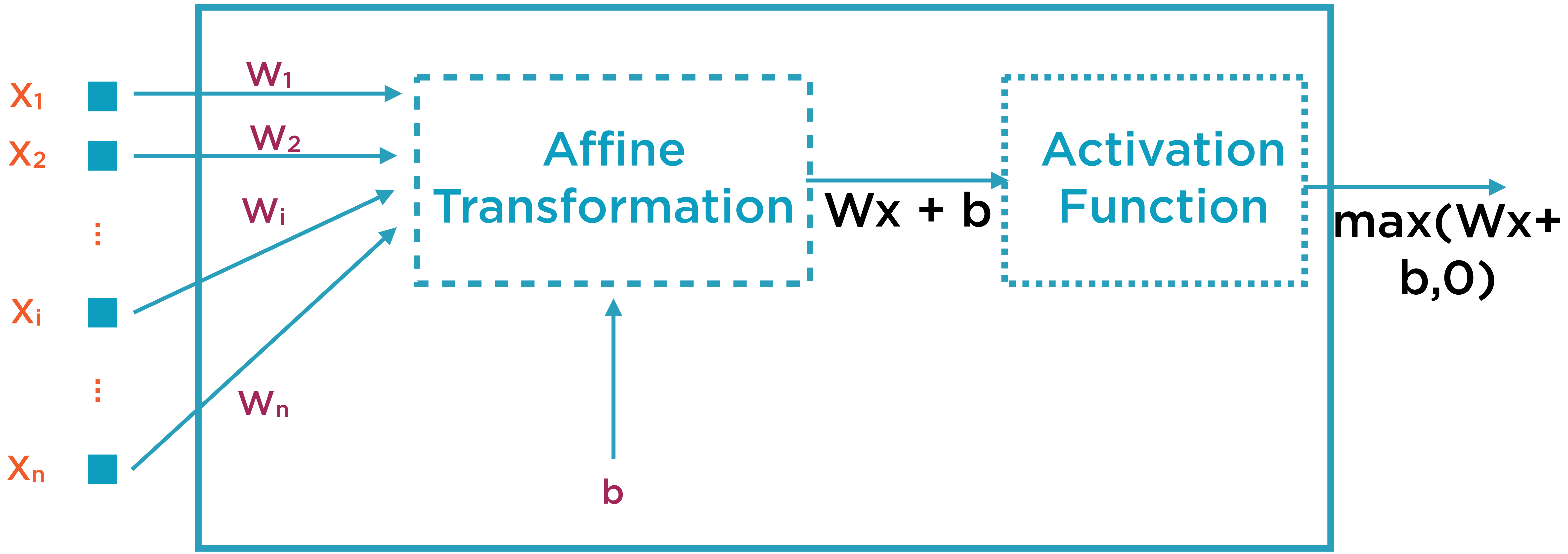
Neural Network



Neurons in a neural network can be connected in very complex ways...

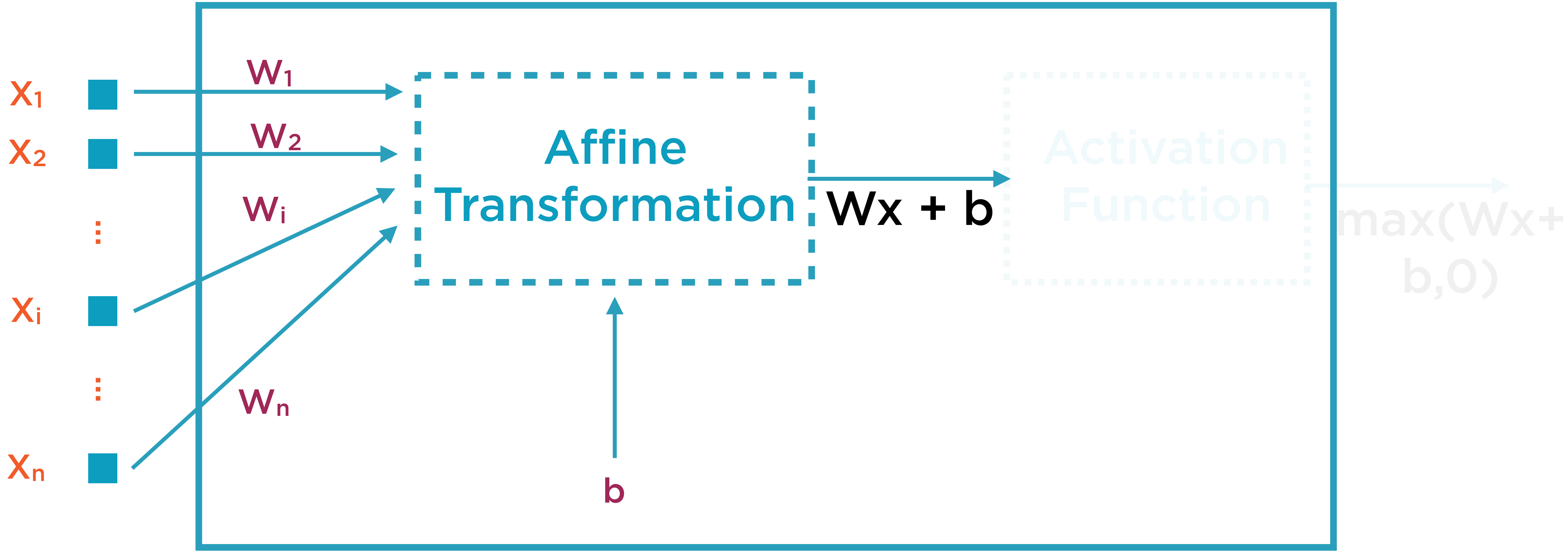
...But each neuron only applies two simple functions to its inputs

Operation of a Single Neuron

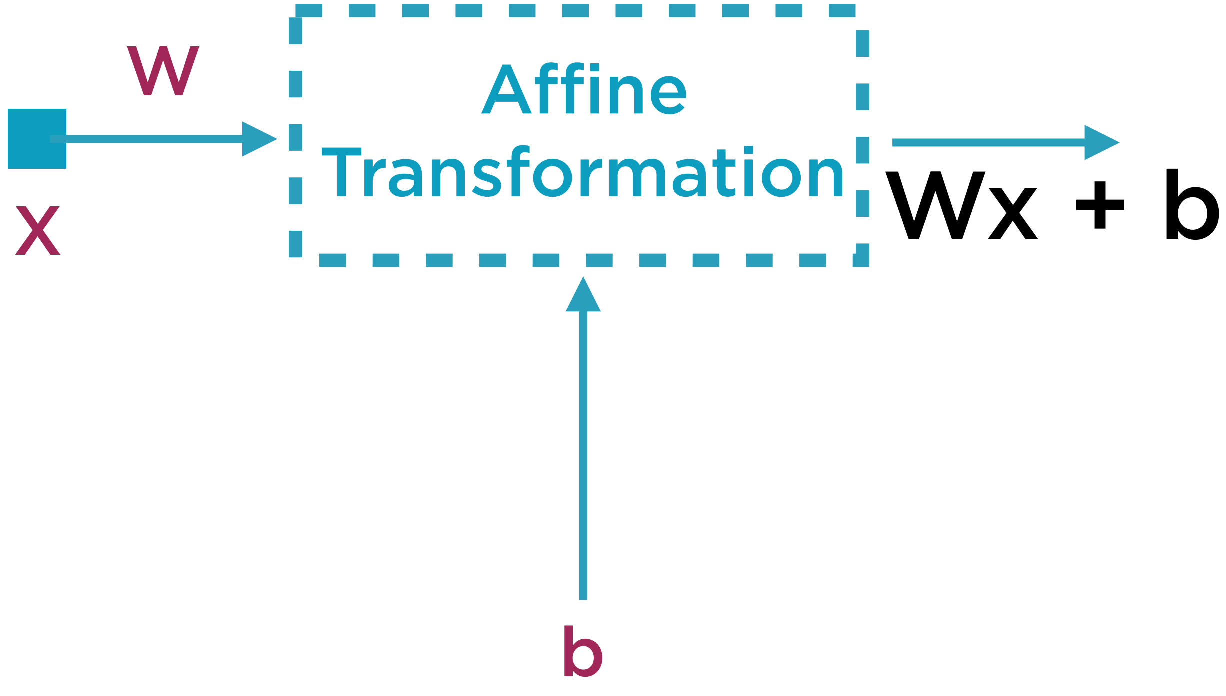


Each neuron only applies two simple functions to its inputs

Operation of a Single Neuron



Where do the values of W and b come from?



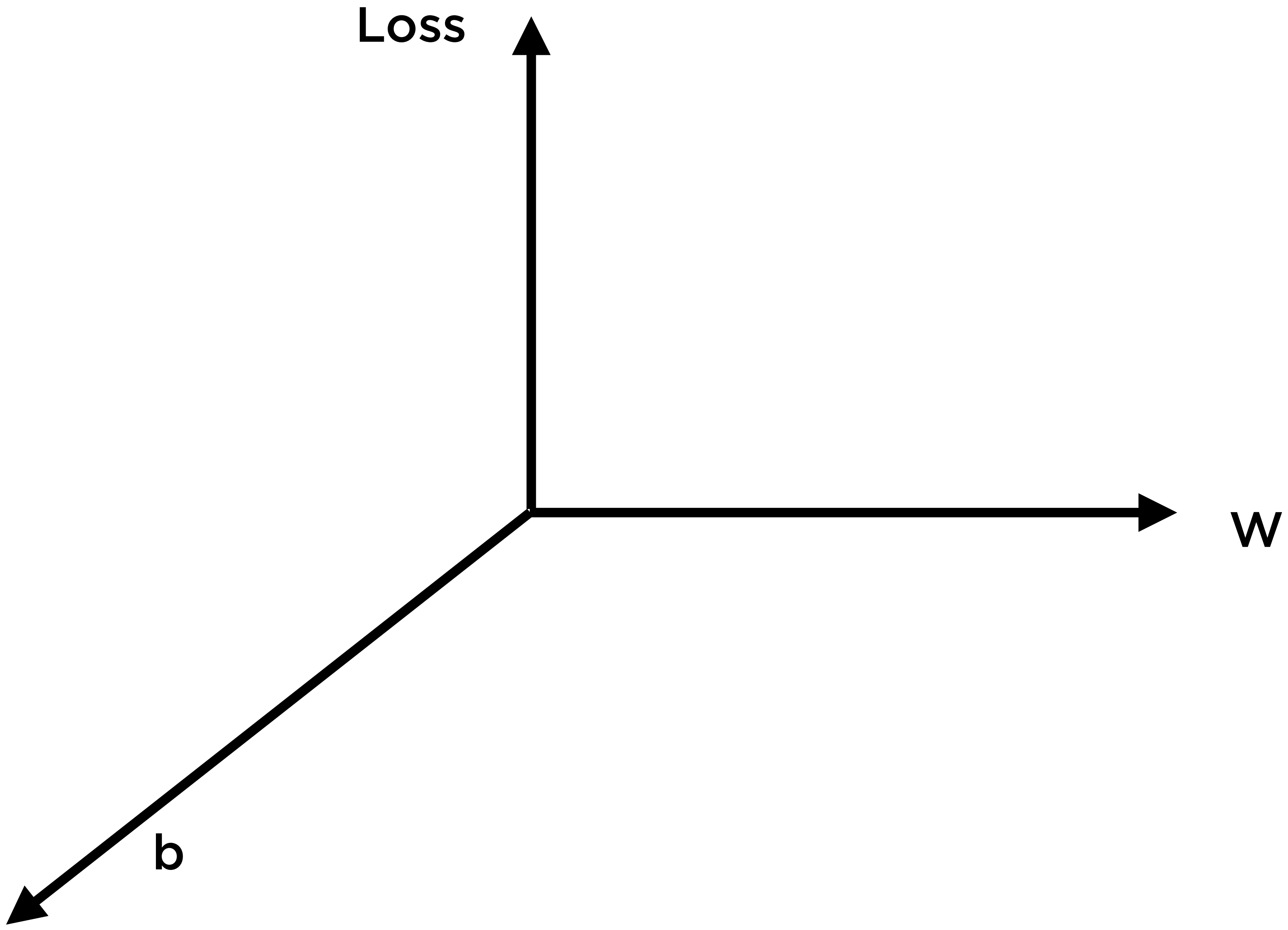
Finding the “best” values of W and b for each neuron is crucial

The “best” values are found using the cost function, optimizer and corpus

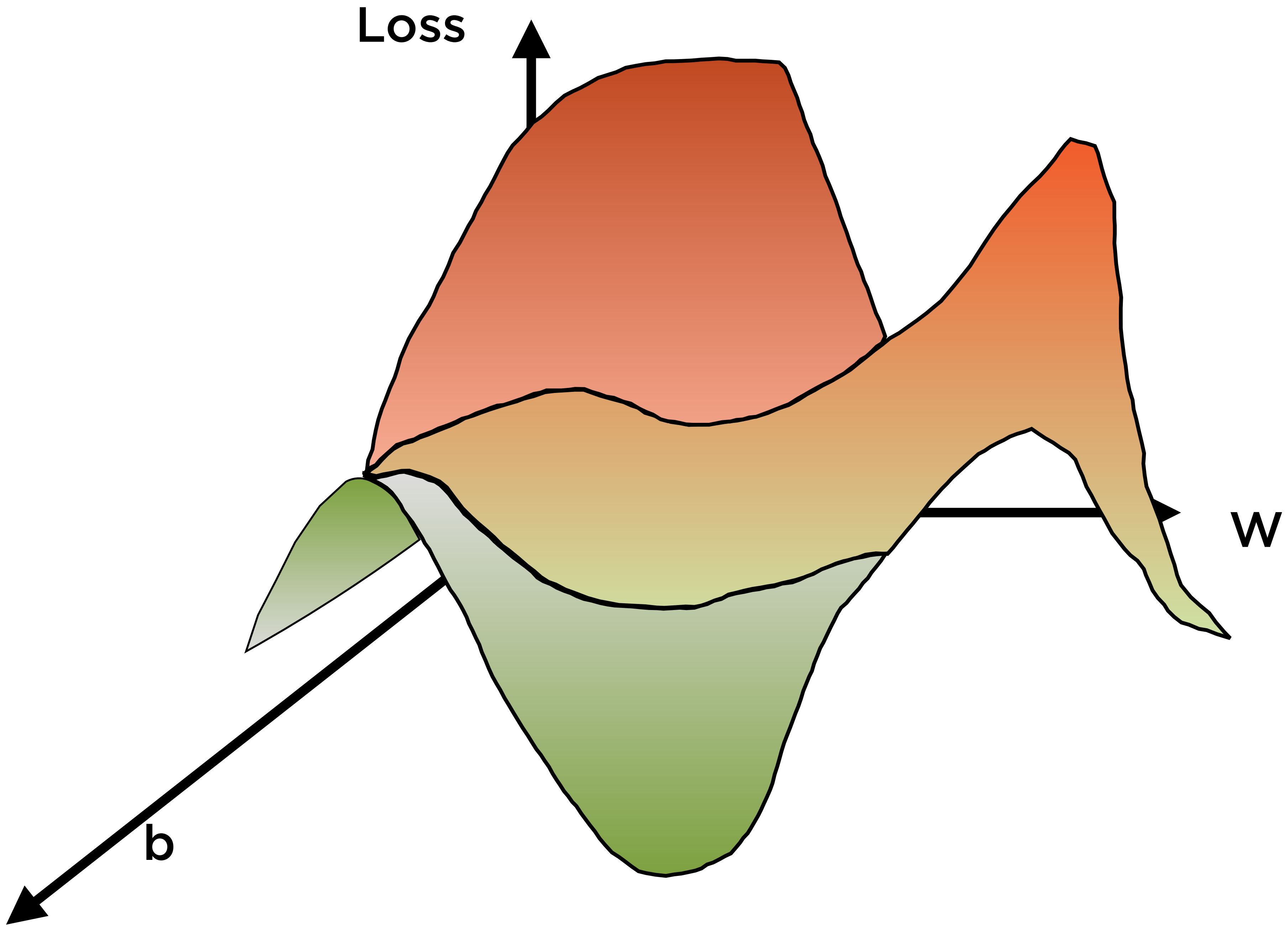
And the process of finding them is called the training process

Training a neural network happens using gradient descent

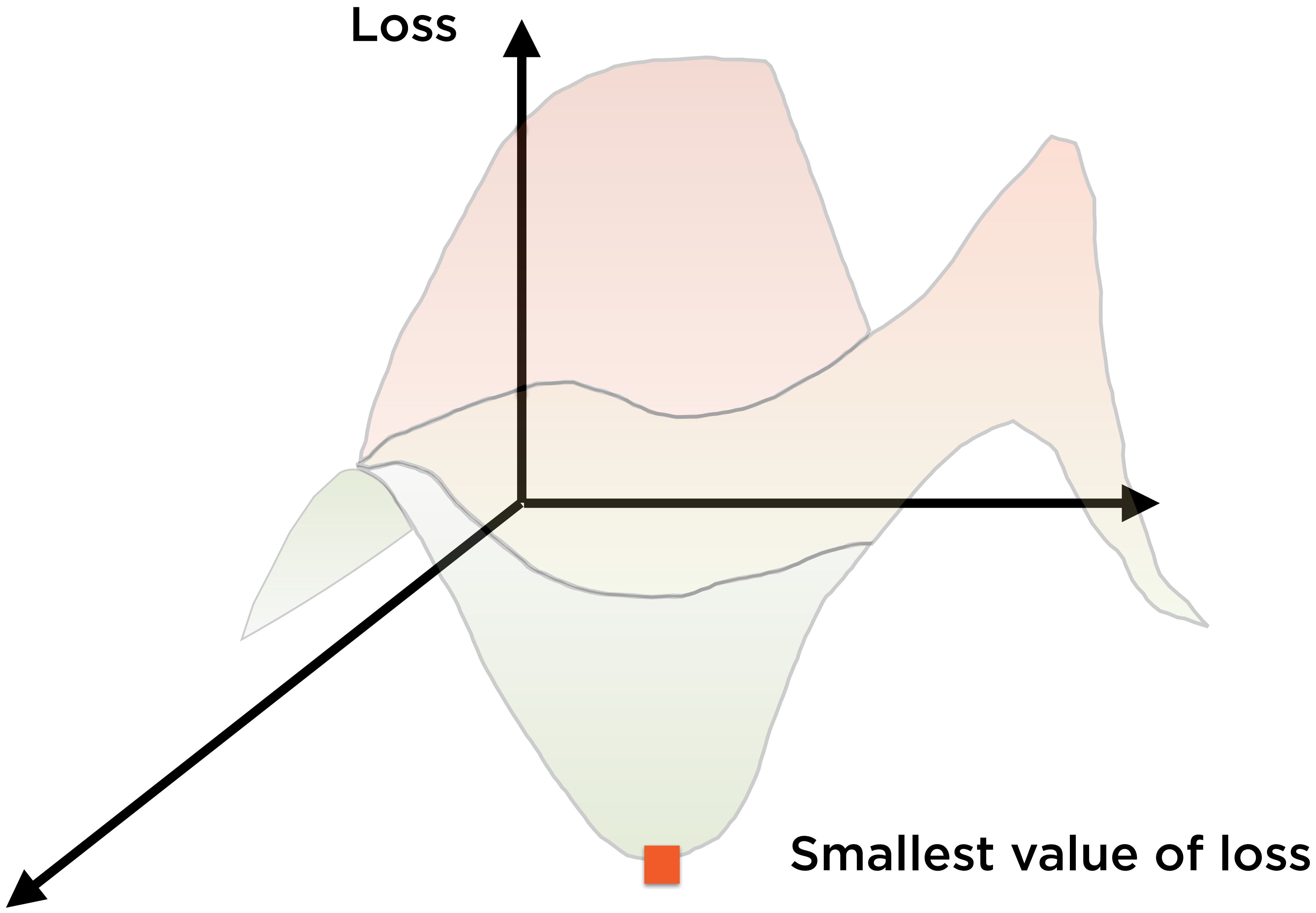
Gradient Descent



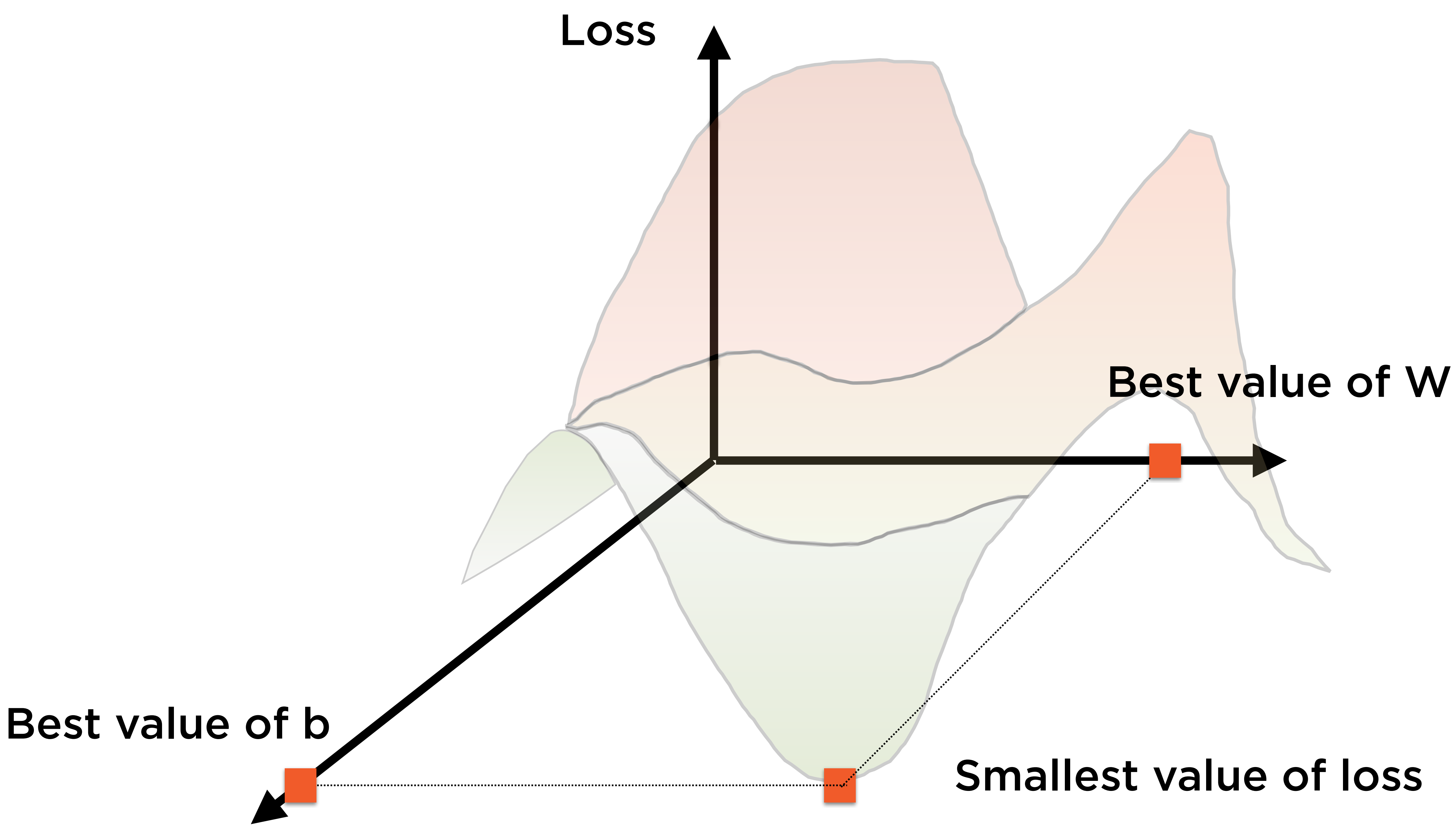
Gradient Descent



Gradient Descent

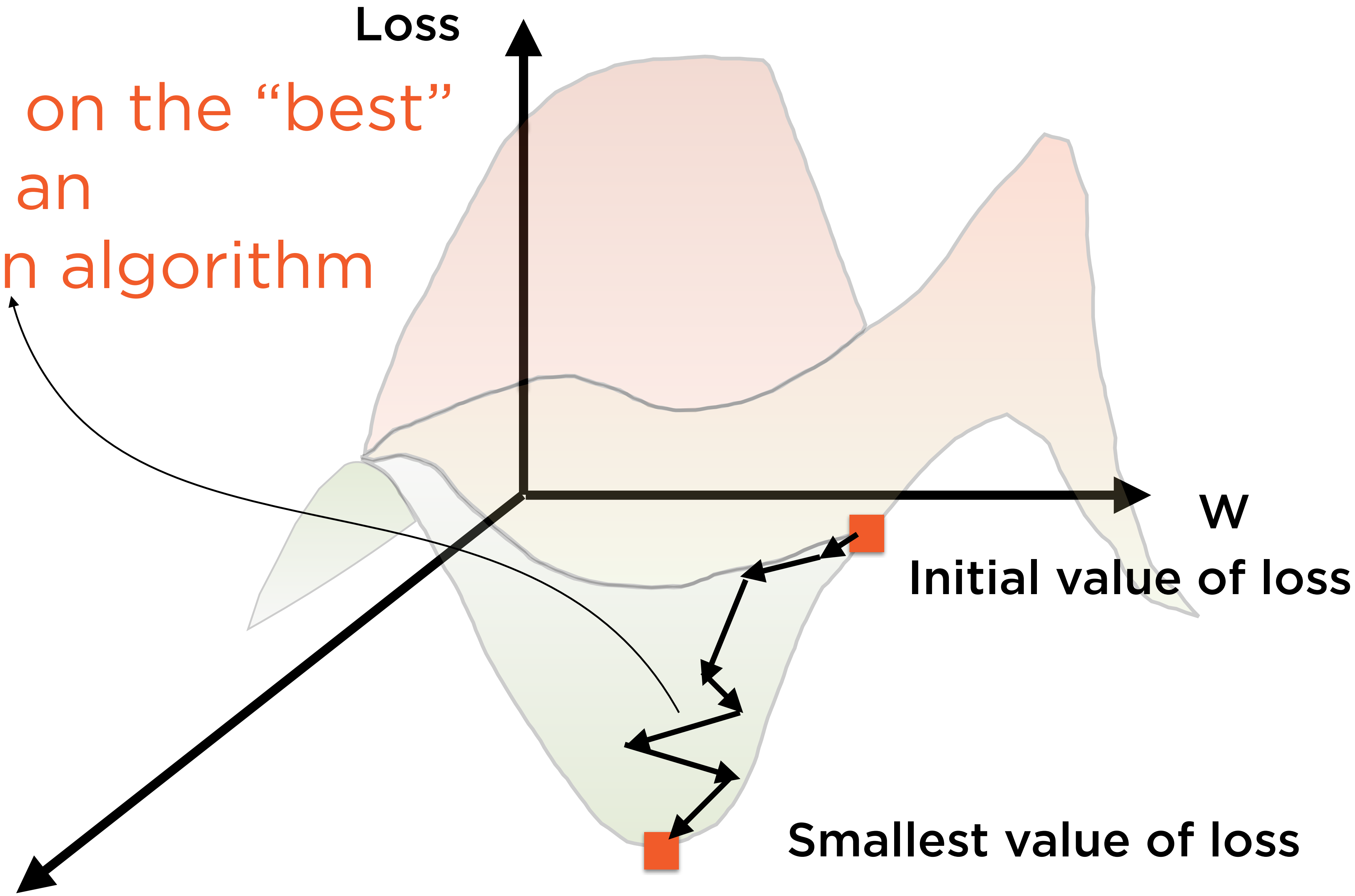


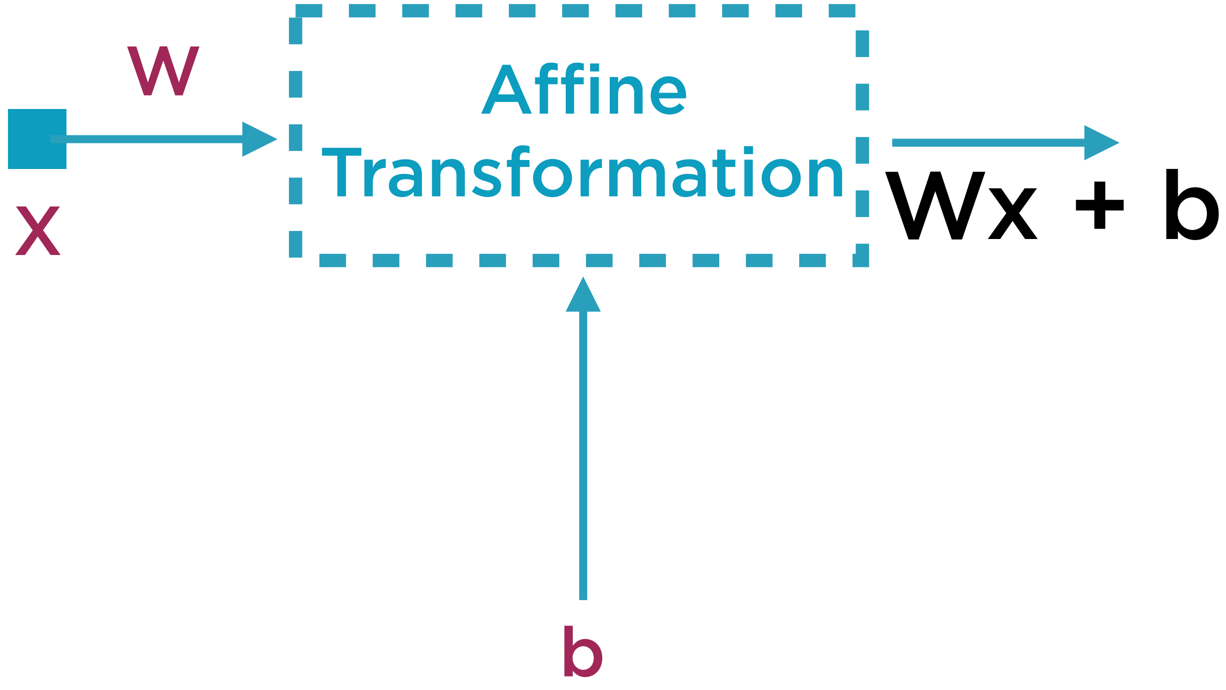
Gradient Descent



Gradient Descent

Converging on the "best" value using an optimization algorithm



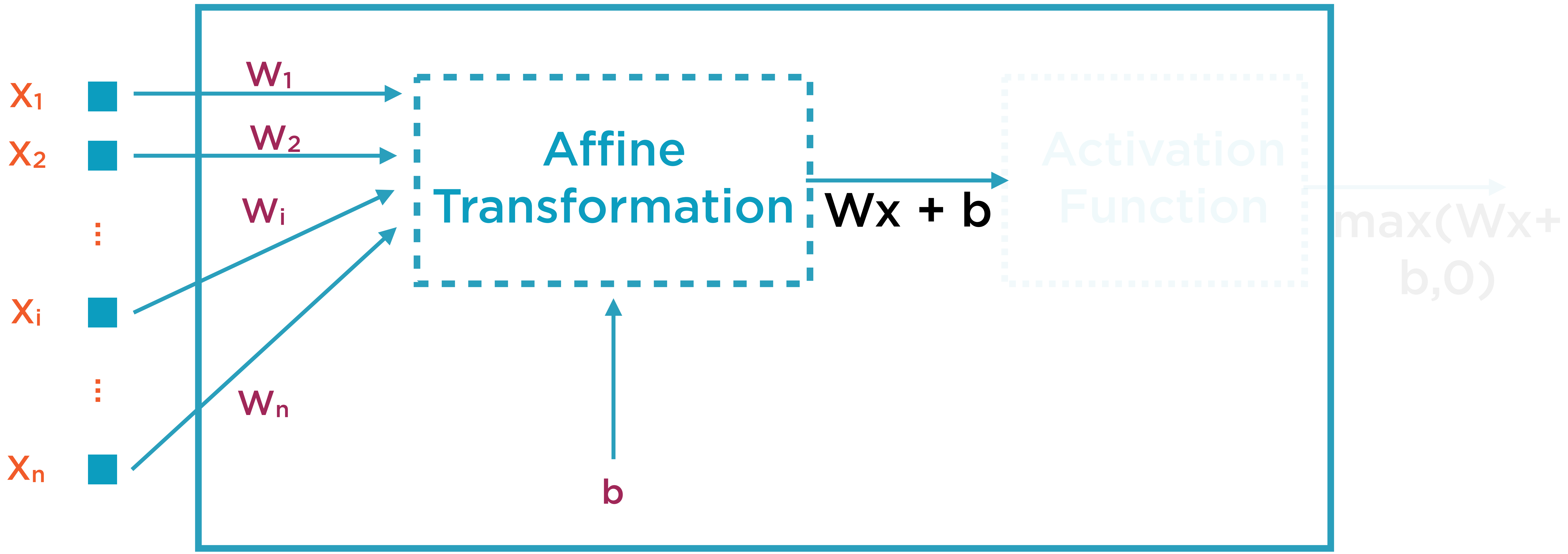


During training, the output of deeper layers may be “fed back” to find the best W, b

This is called back propagation

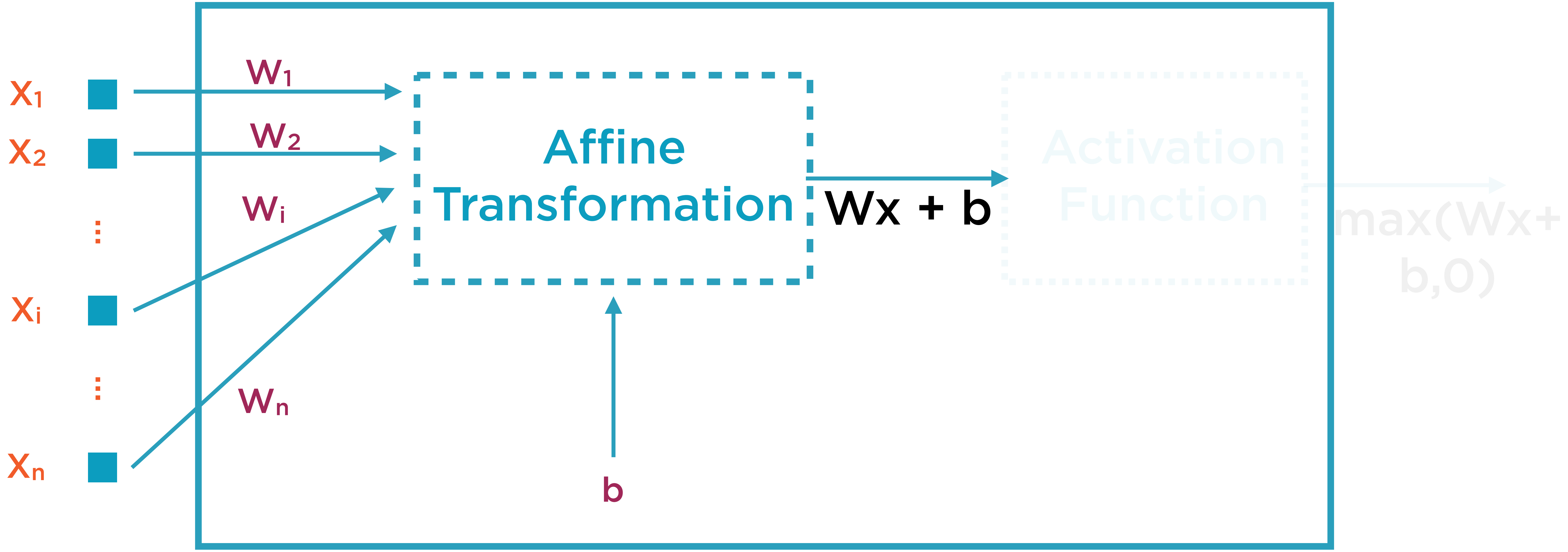
Back propagation is the standard algorithm for training neural networks

Operation of a Single Neuron



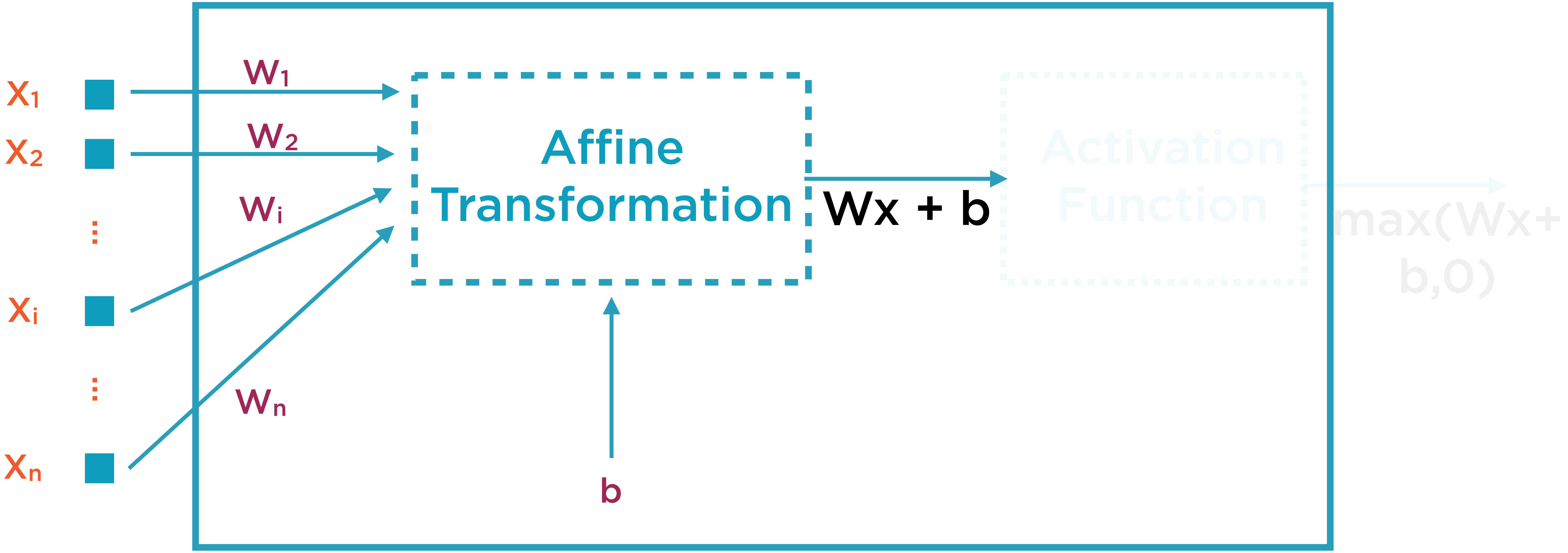
The training algorithm will use the weights to tell the neuron which inputs matter, and which do not...

Operation of a Single Neuron



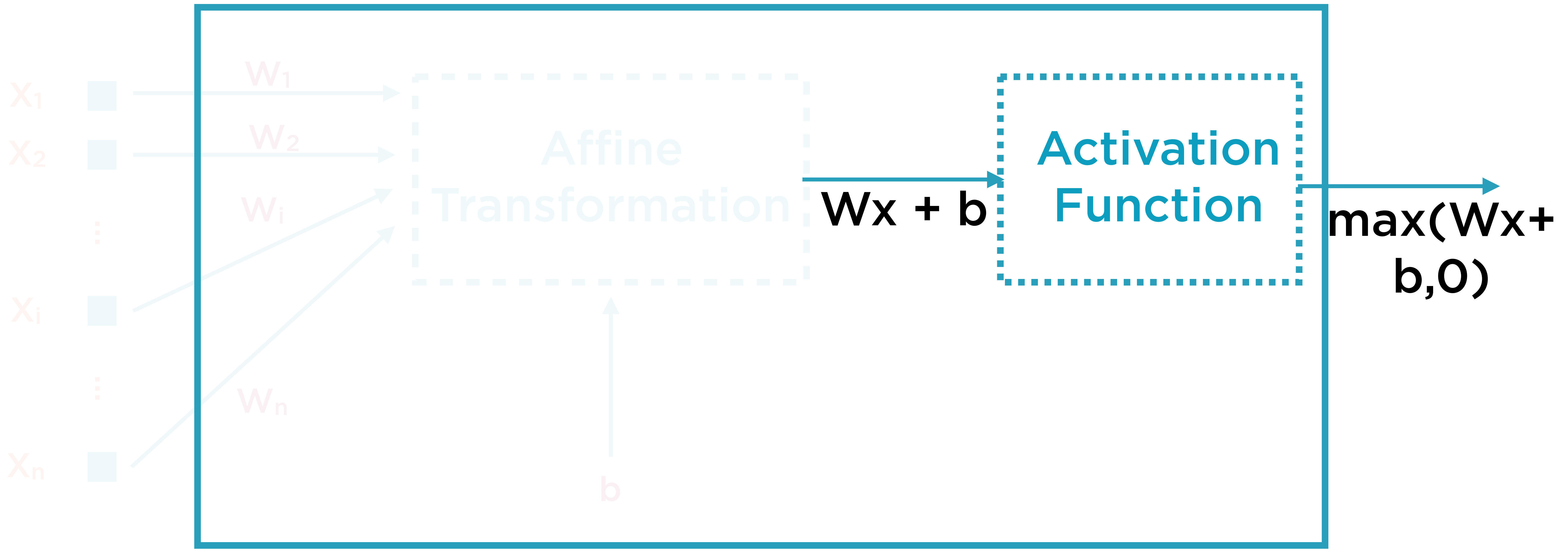
...and apply a corrective bias if needed

Operation of a Single Neuron



The linear output can only be used to learn linear functions, but we can easily generalize this...

Operation of a Single Neuron



The Activation function is a non-linear function, very often simply the max(0,...) function

The output of the affine transformation is chained into an activation function

The activation function is needed for the neural network to predict non-linear functions

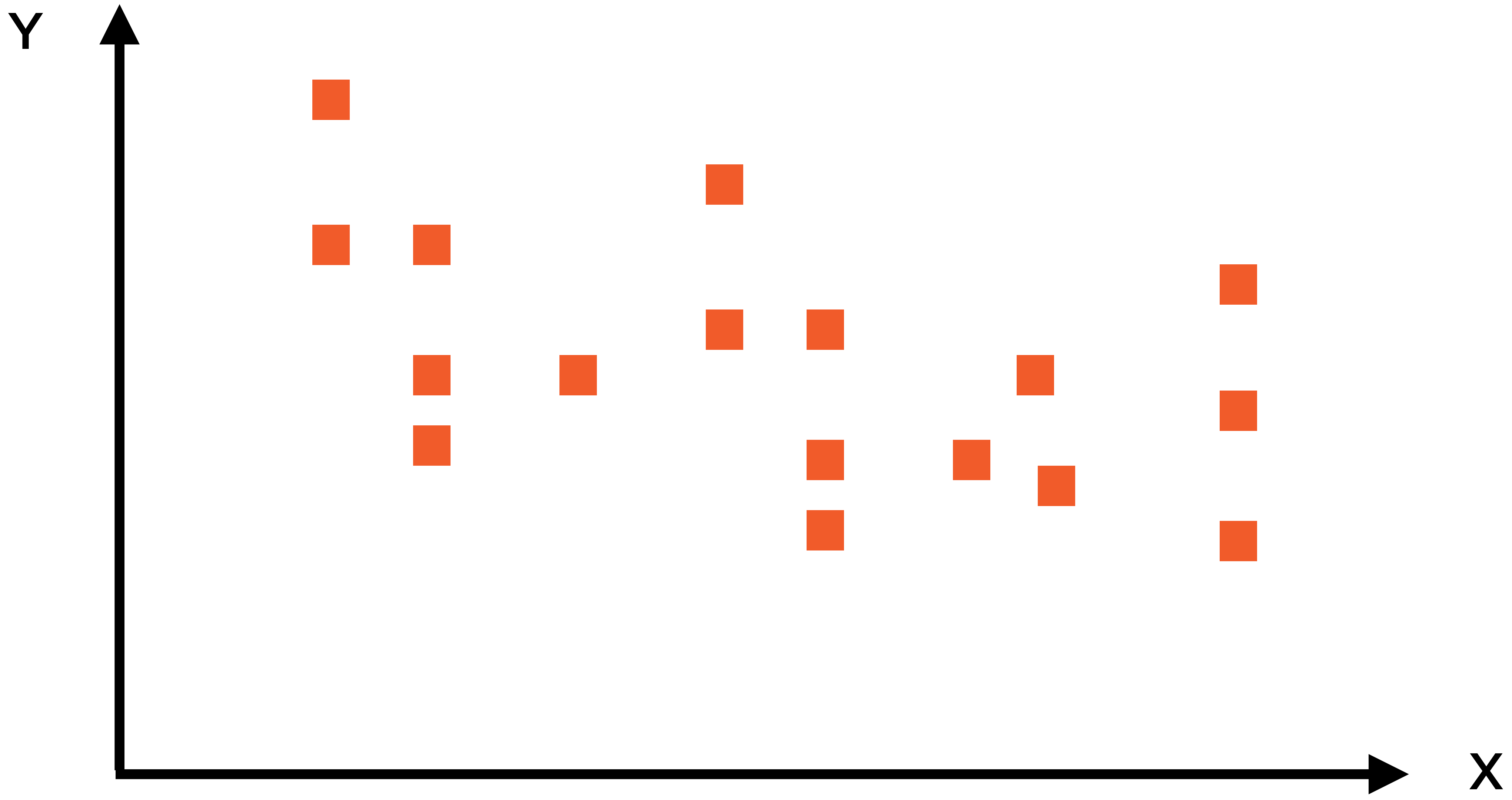
The most common form of the activation function is the ReLU

ReLU : Rectified Linear Unit

ReLU(x) = max(0,x)

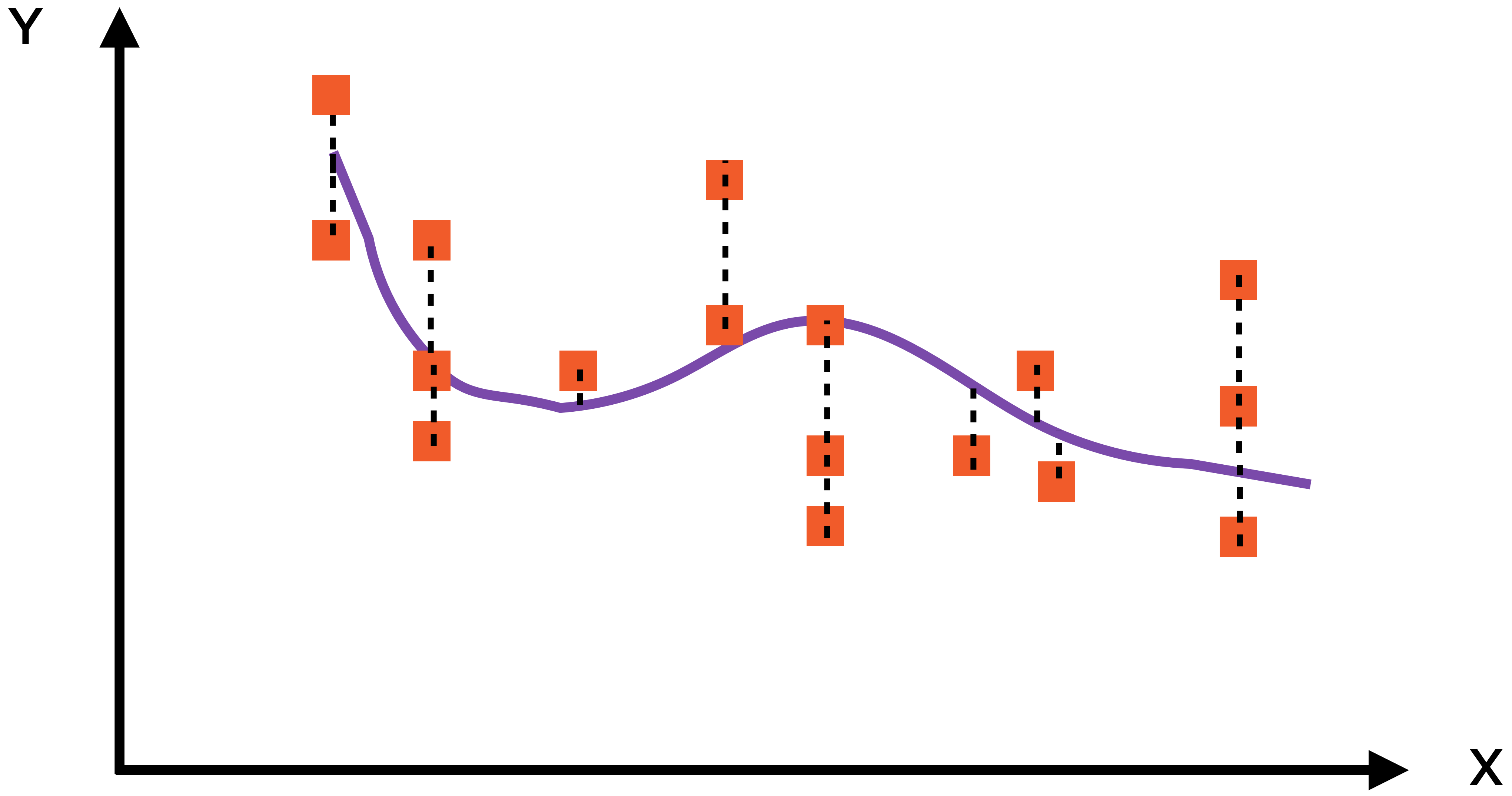
Overfitting and Underfitting

Connecting the Dots



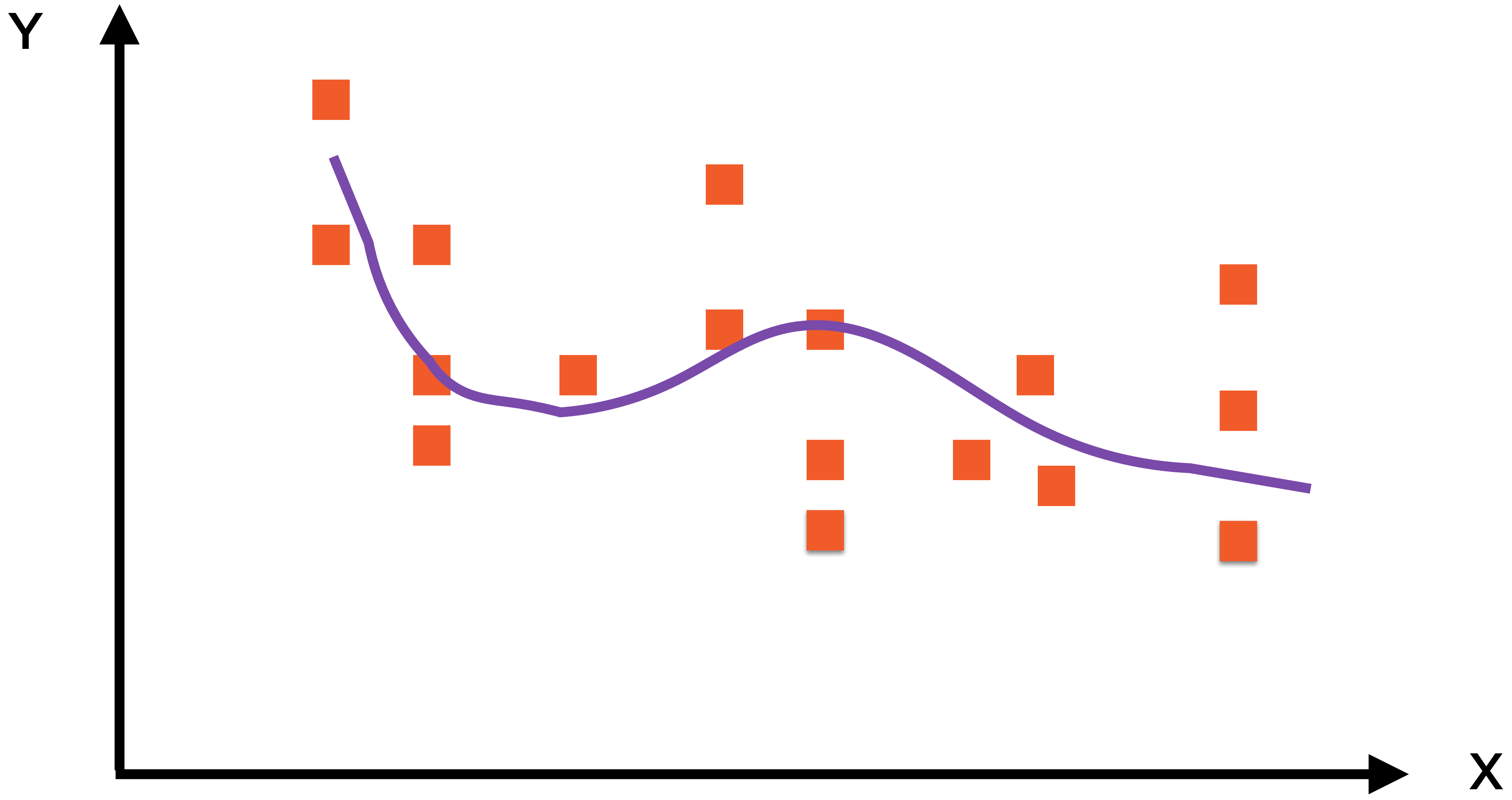
Challenge: Fit the "best" curve through these points

Good Fit?



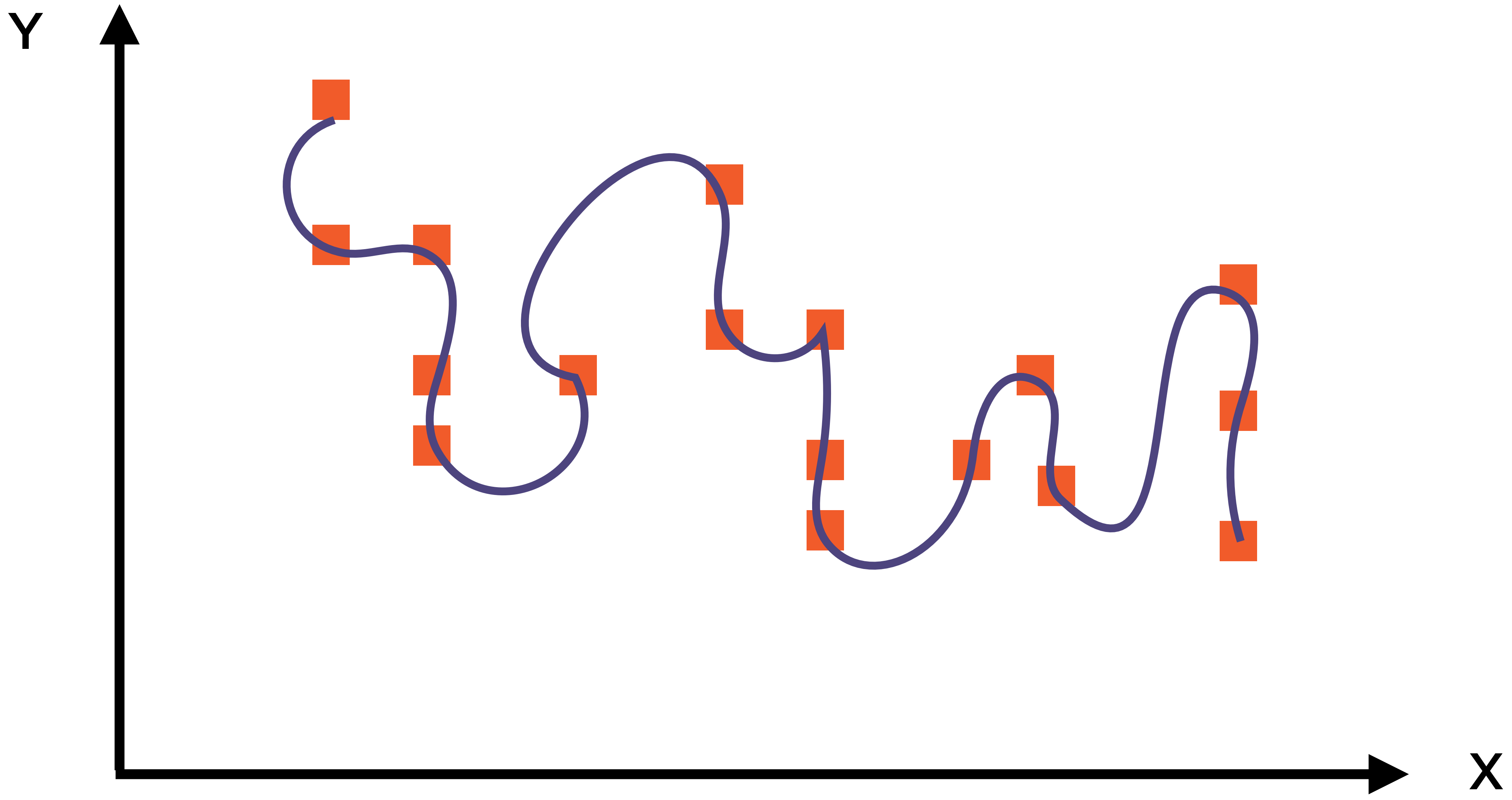
A curve has a “good fit” if the distances of points from the curve are small

Connecting the Dots



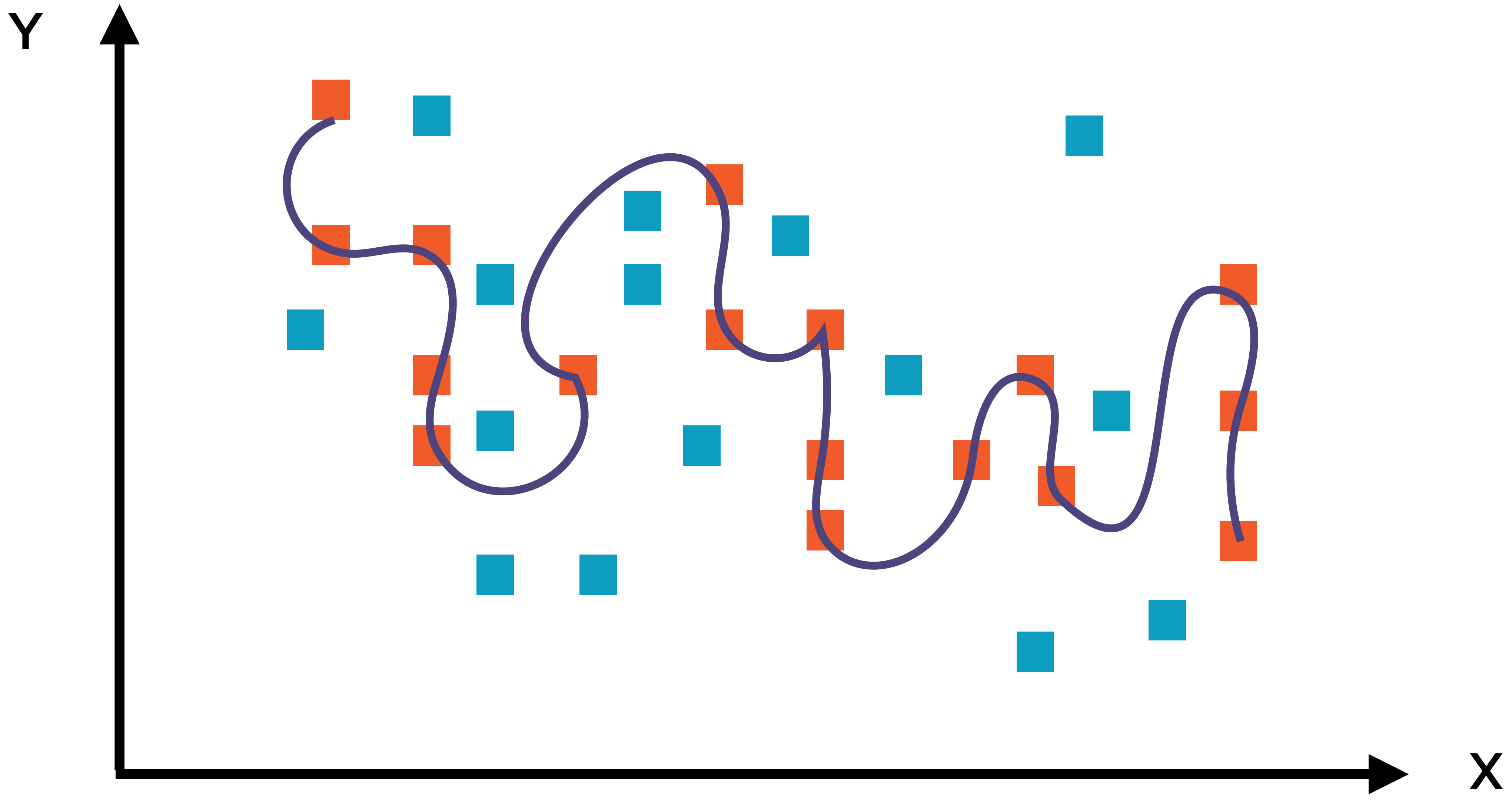
We could draw a pretty complex curve

Connecting the Dots



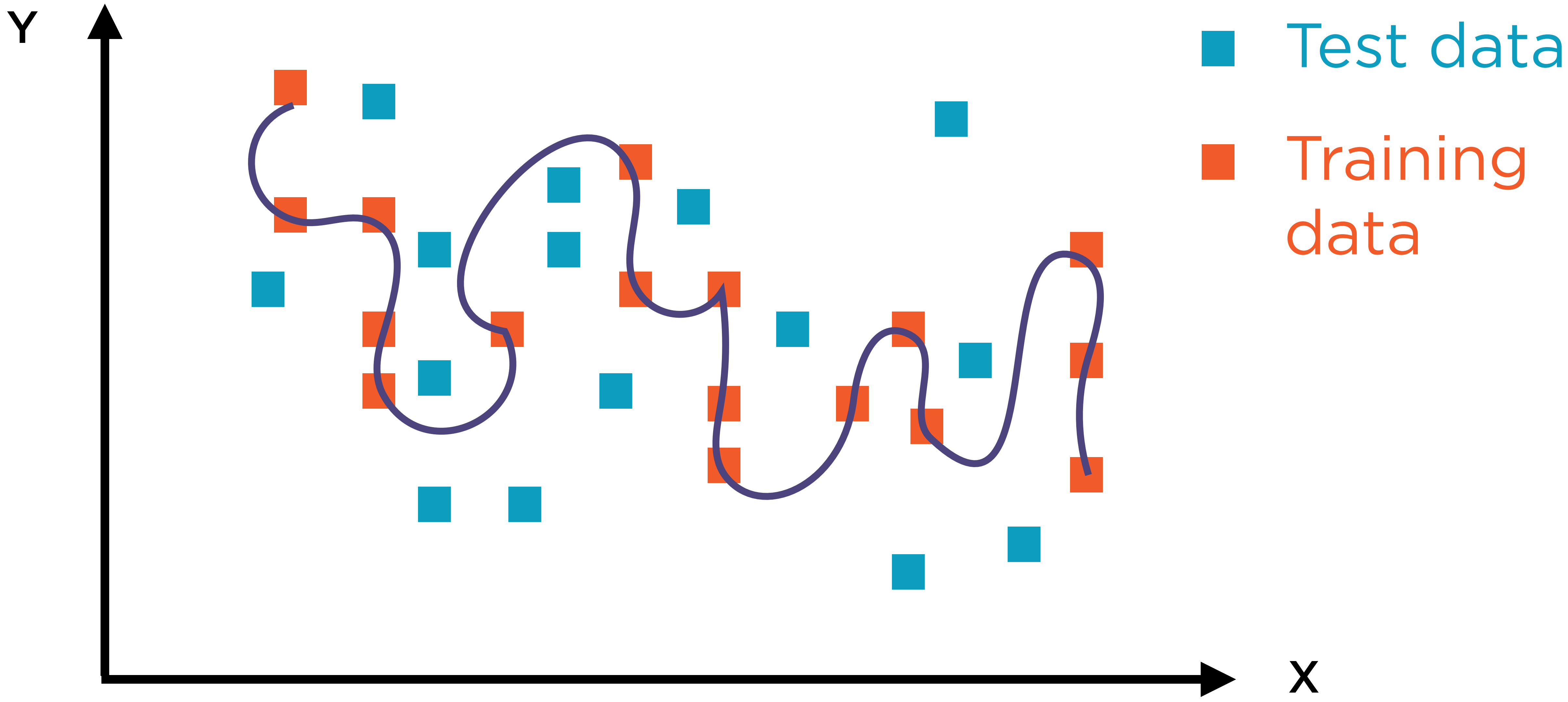
We can even make it pass through every single point

Connecting the Dots



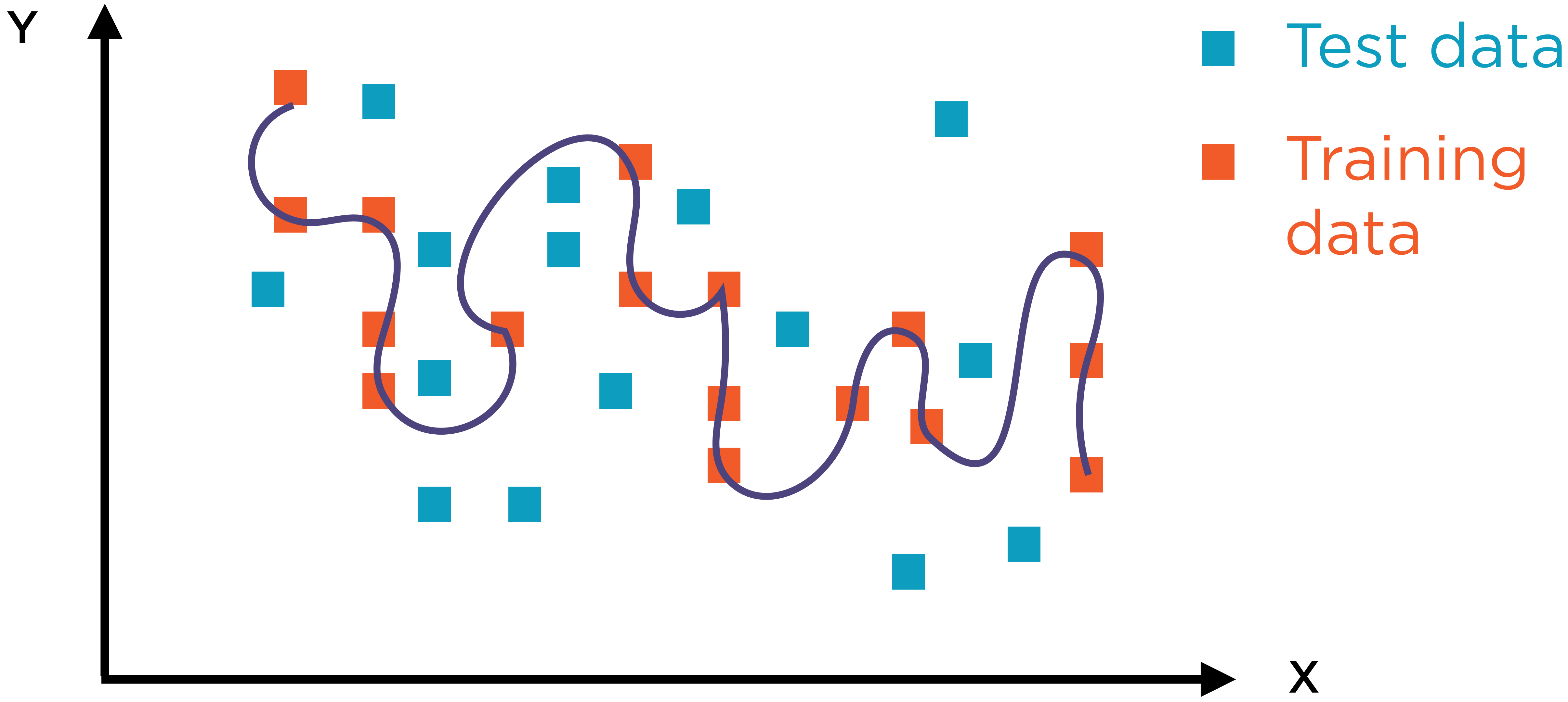
But given a new set of points, this curve might perform quite poorly

Connecting the Dots



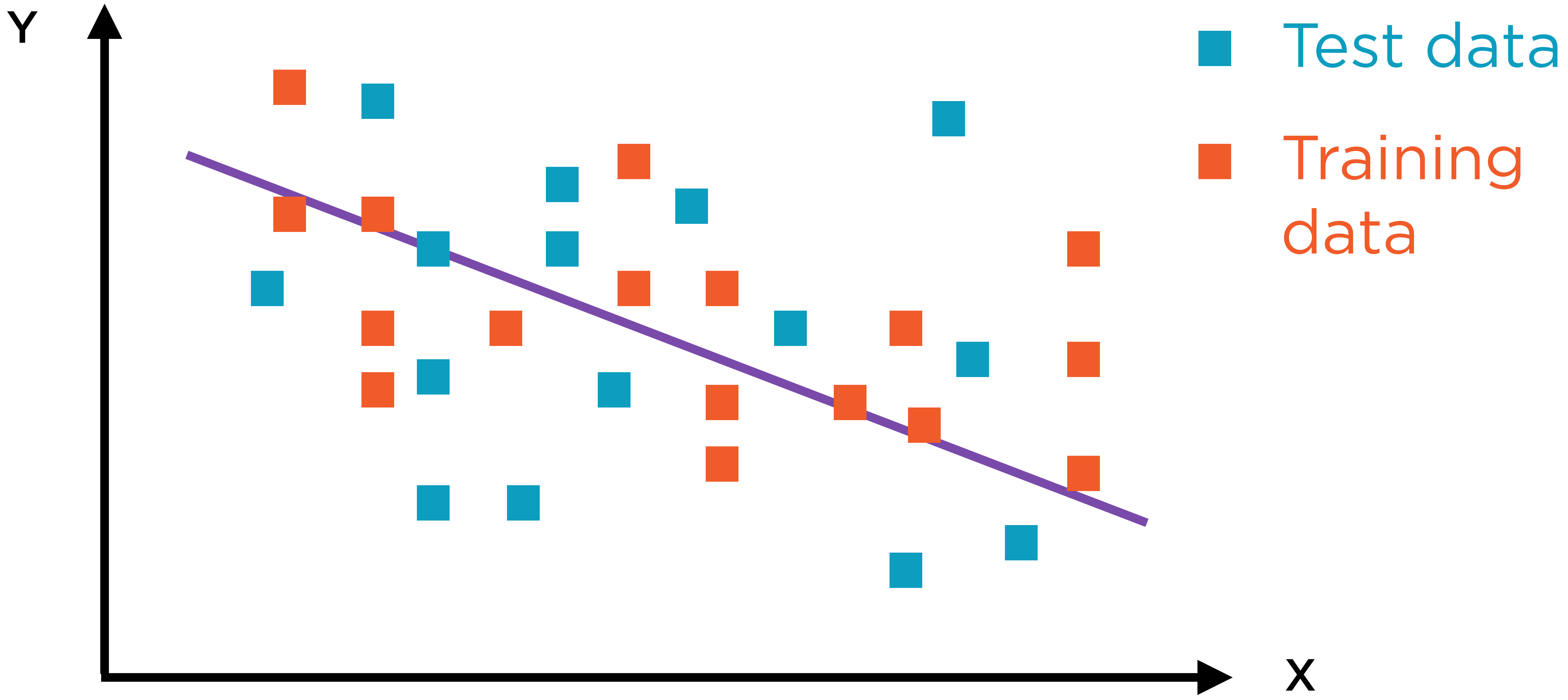
The original points were "training data", the new points are "test data"

Overfitting



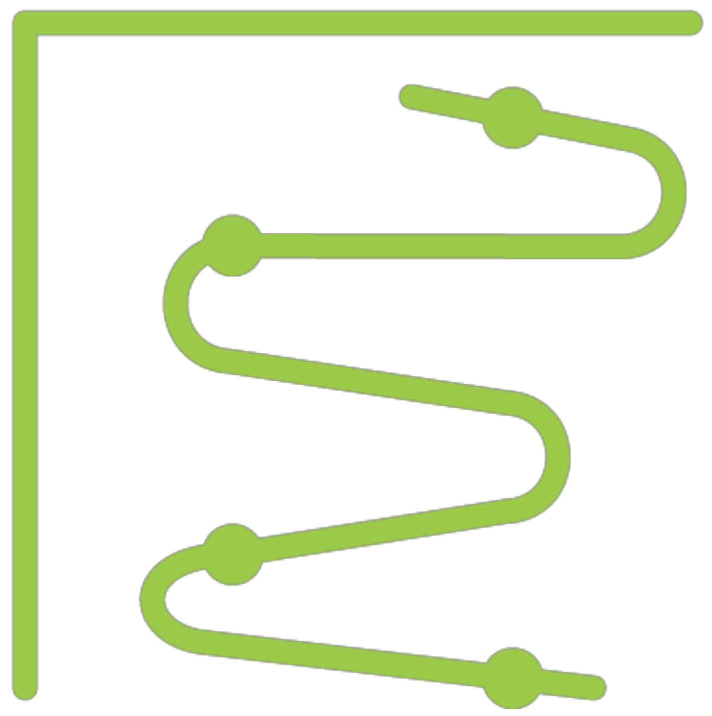
Great performance in training, poor performance in real usage

Connecting the Dots



A simple straight line performs worse in training, but better with test data

Overfitting



Model has memorized the training data

Low training error

Does not work well in the real world

High test error

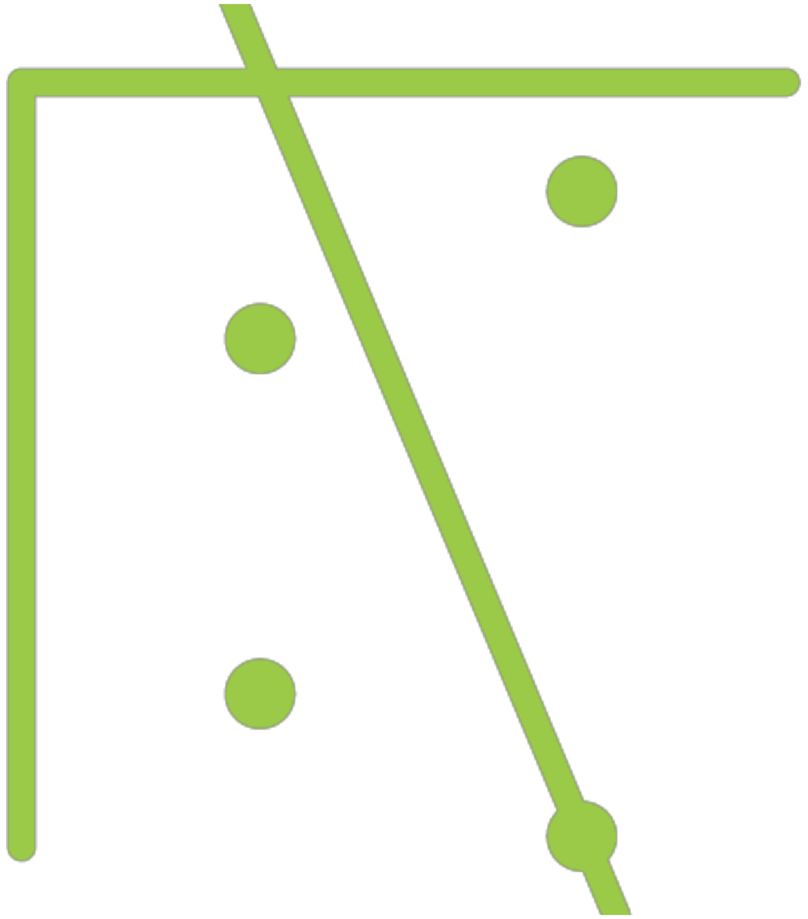
Preventing Overfitting

Regularization - Penalize complex models

Cross-validation - Distinct training and validation phases

Dropout (NNs only) - Intentionally turn off some neurons during training

Underfitting



Model unable to capture relationships in data

Performs poorly on the training data

Model too “simple” to be useful

Summary

scikit-learn support for neural networks

scikit-learn vs. deep learning frameworks

Perceptrons and neurons

Multi-layer perceptrons (MLPs) and neural networks

Training a neural network