Neural Networks with TensorFlow

Overview

Neural networks are representation based machine learning algorithms

Neural networks are made up of building blocks called neurons

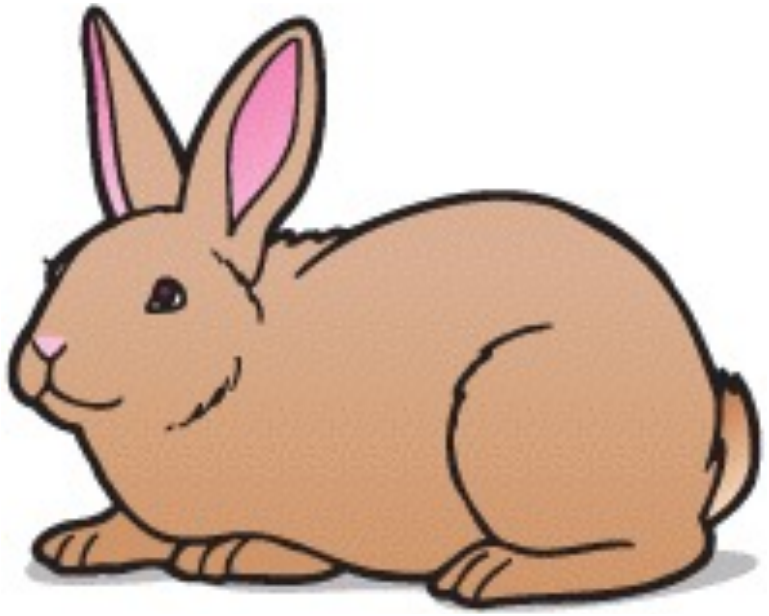
Each neuron is made up of a linear function and an activation function

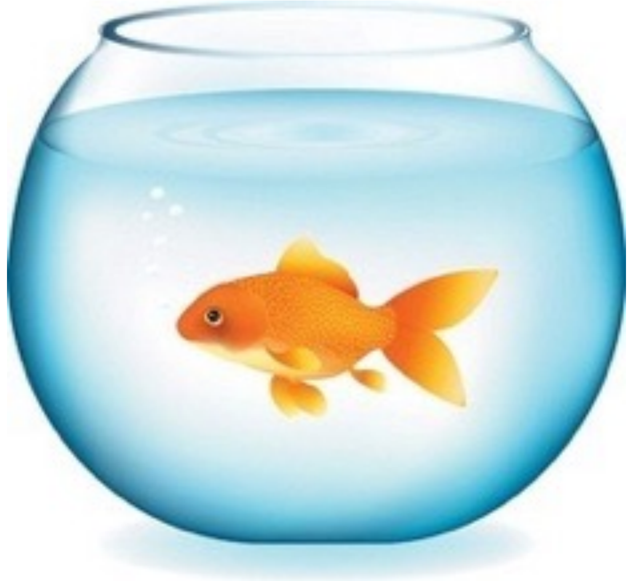
Performance is very sensitive to details such as proper choice of activation

Overfitting in neural networks is mitigated using techniques such as dropout

Understanding Machine Learning

Whales: Fish or Mammals?





Mammals

Members of the infraorder Cetacea

Fish

Look like fish, swim like fish, move with fish

Whales: Fish or Mammals?

ML-based Classifier

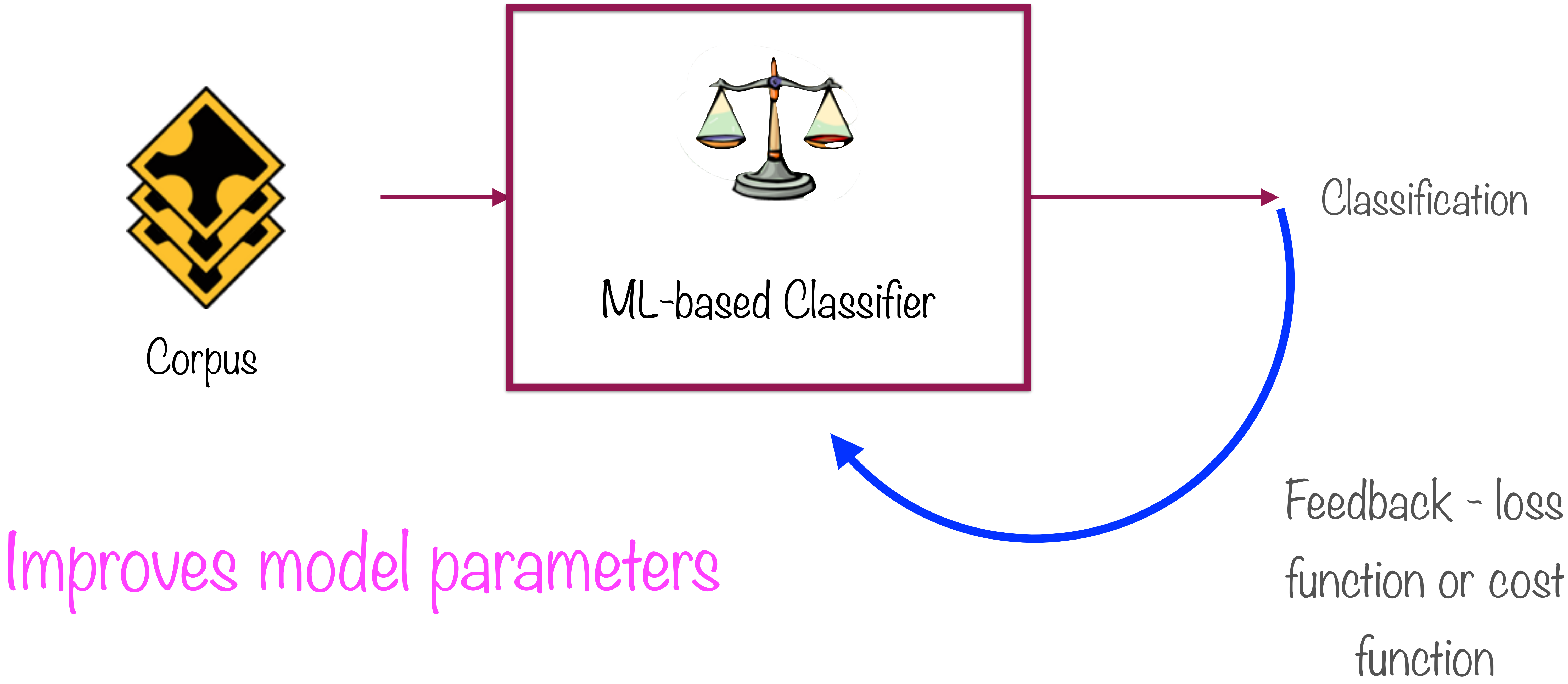
Training

Feed in a large corpus of data classified correctly

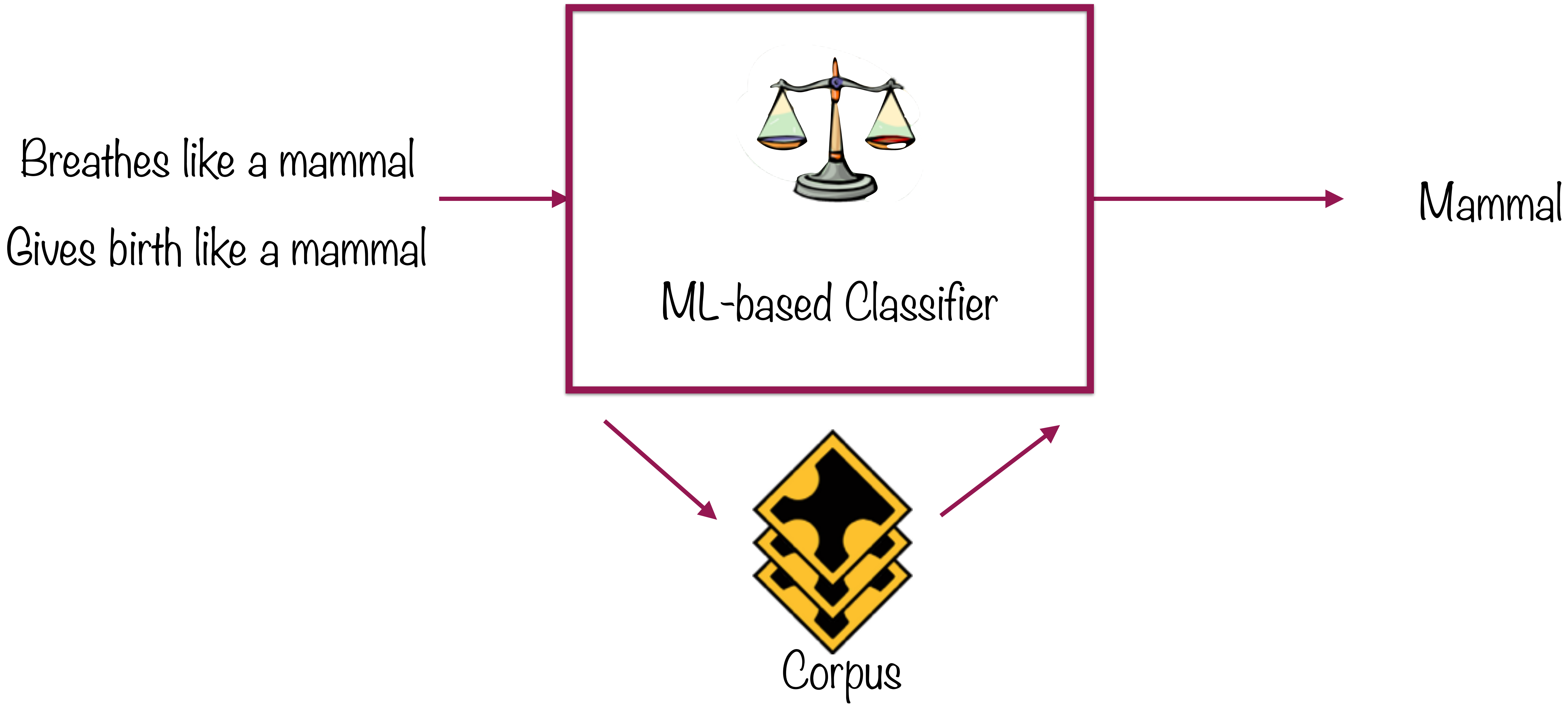
Prediction

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

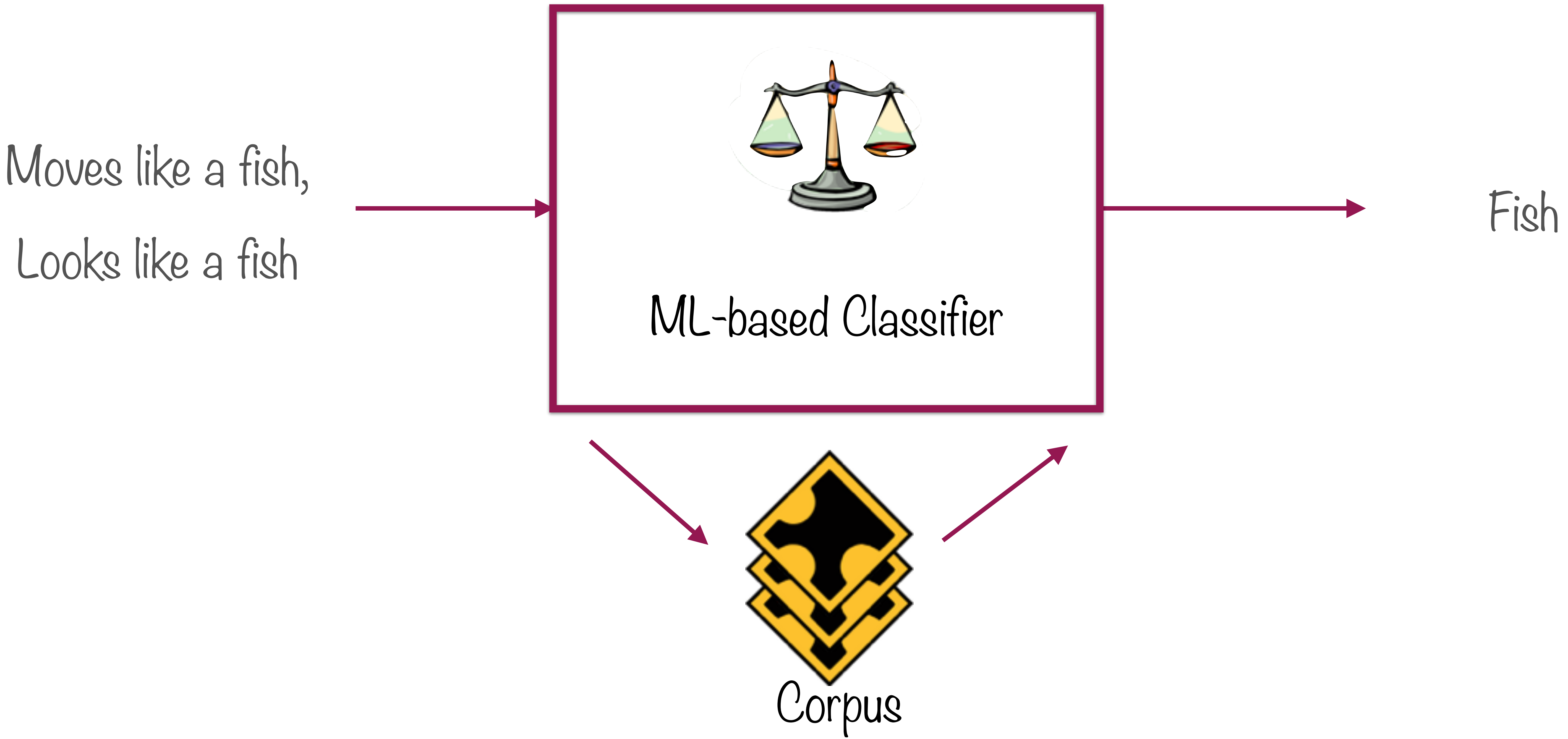
Training the ML-based Classifier



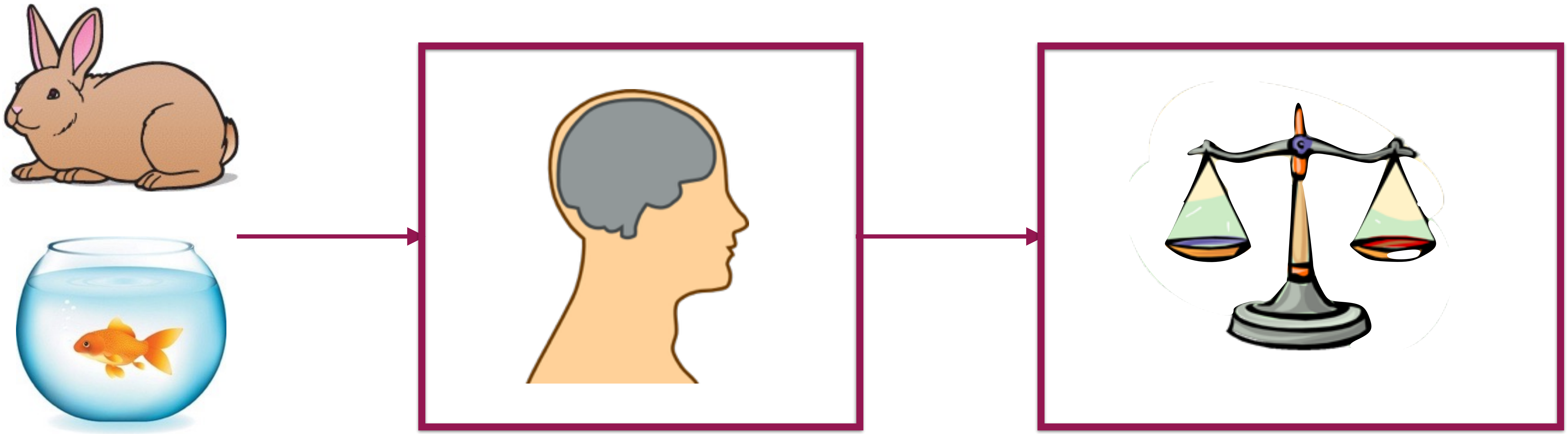
ML-based Binary Classifier



“Traditional” ML-based Binary Classifier



ML-based Binary Classifier

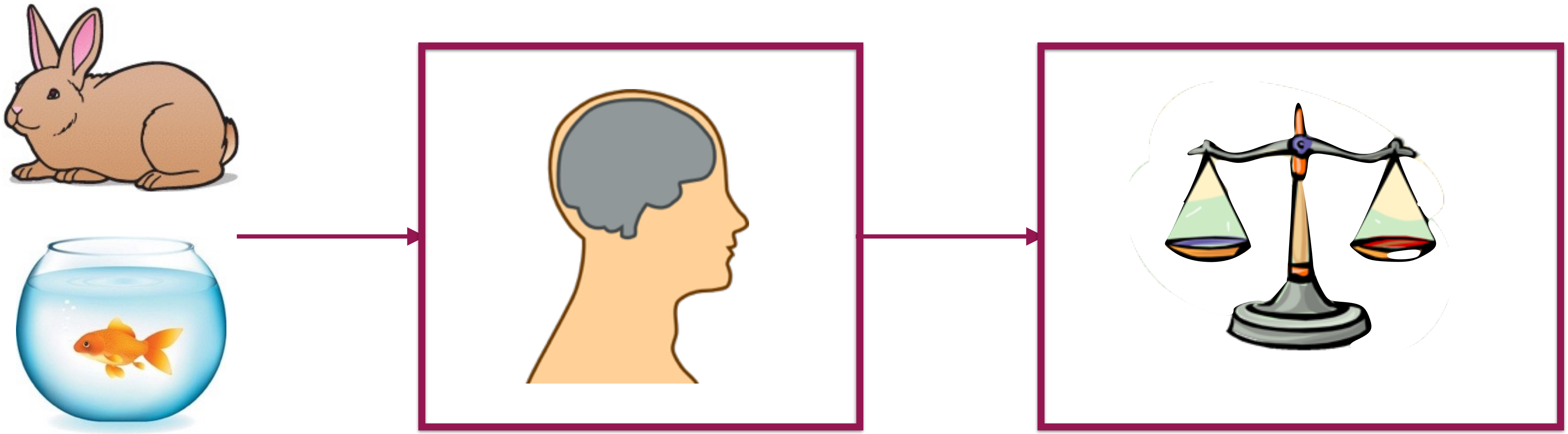


Corpus

Classification Algorithm

ML-based Classifier

ML-based Binary Classifier

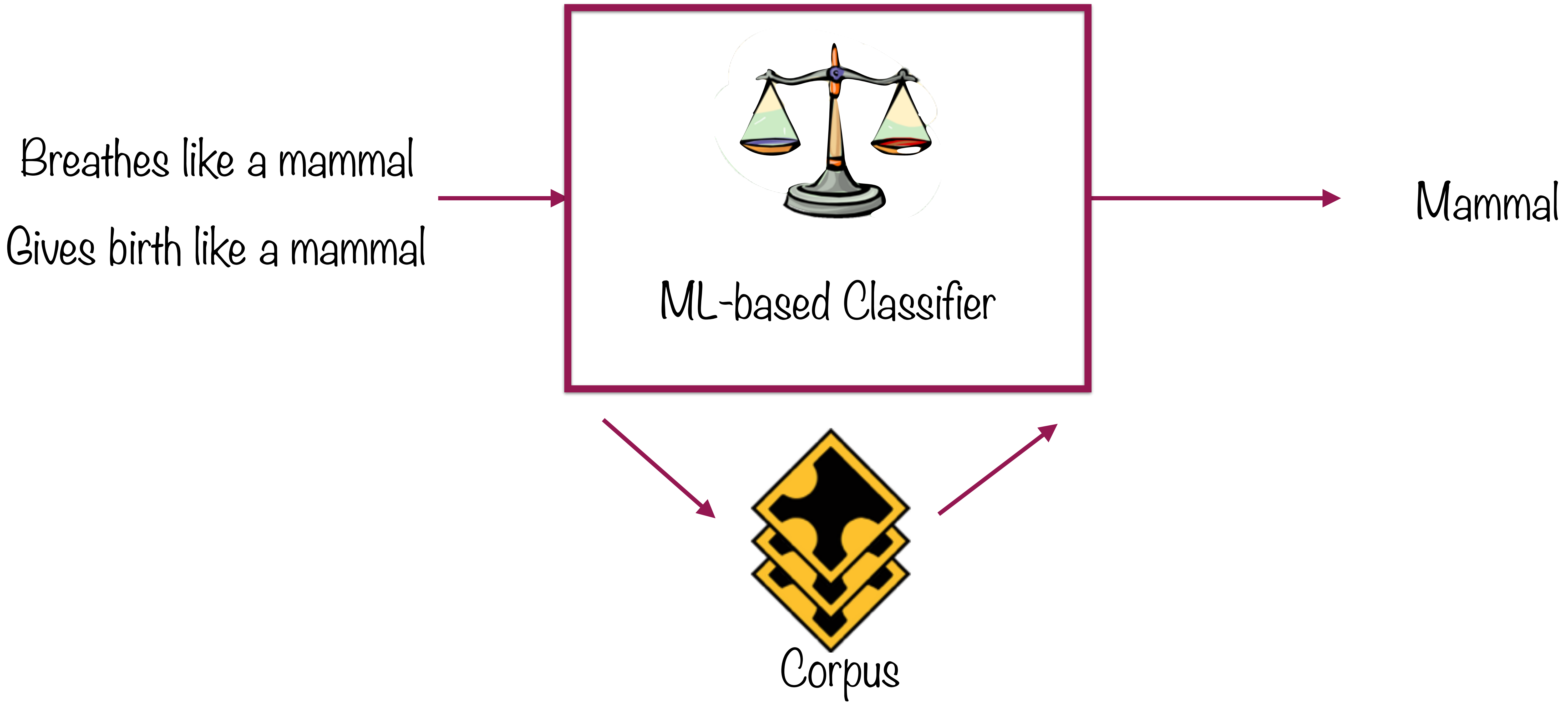


Corpus

Naive Bayes, Support Vector Machines, Decision Trees

ML-based Classifier

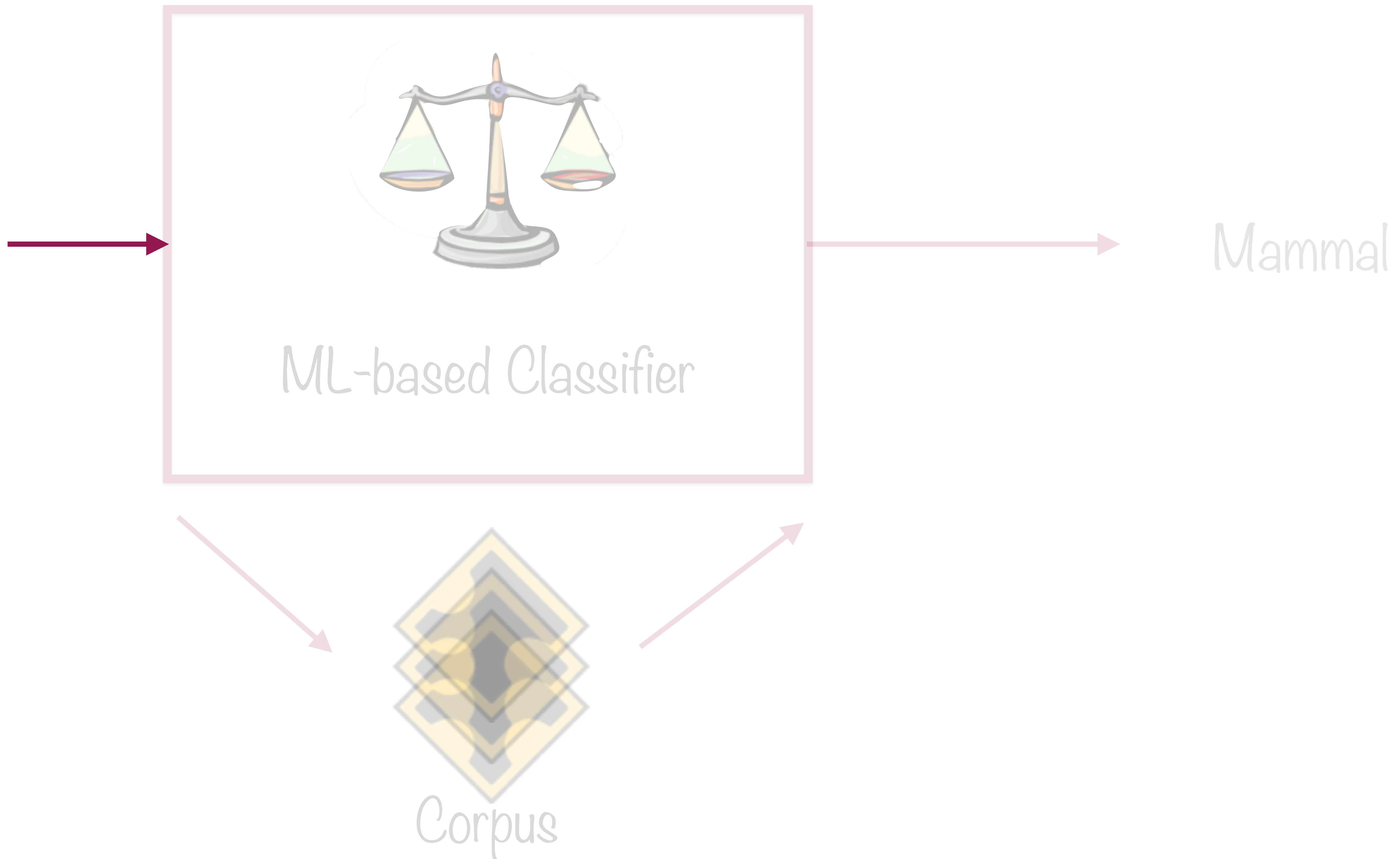
ML-based Binary Classifier



ML-based Binary Classifier

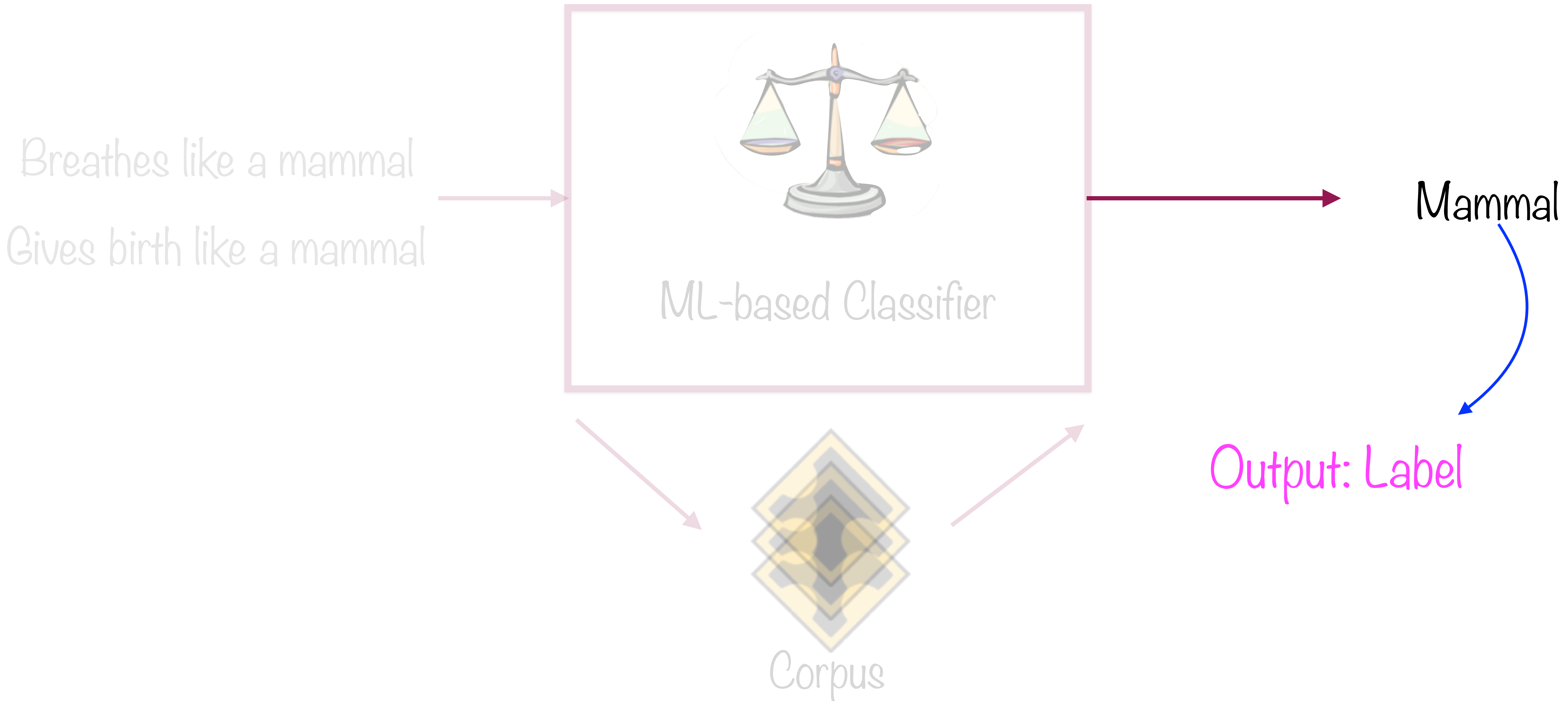
Breathes like a mammal

Gives birth like a mammal

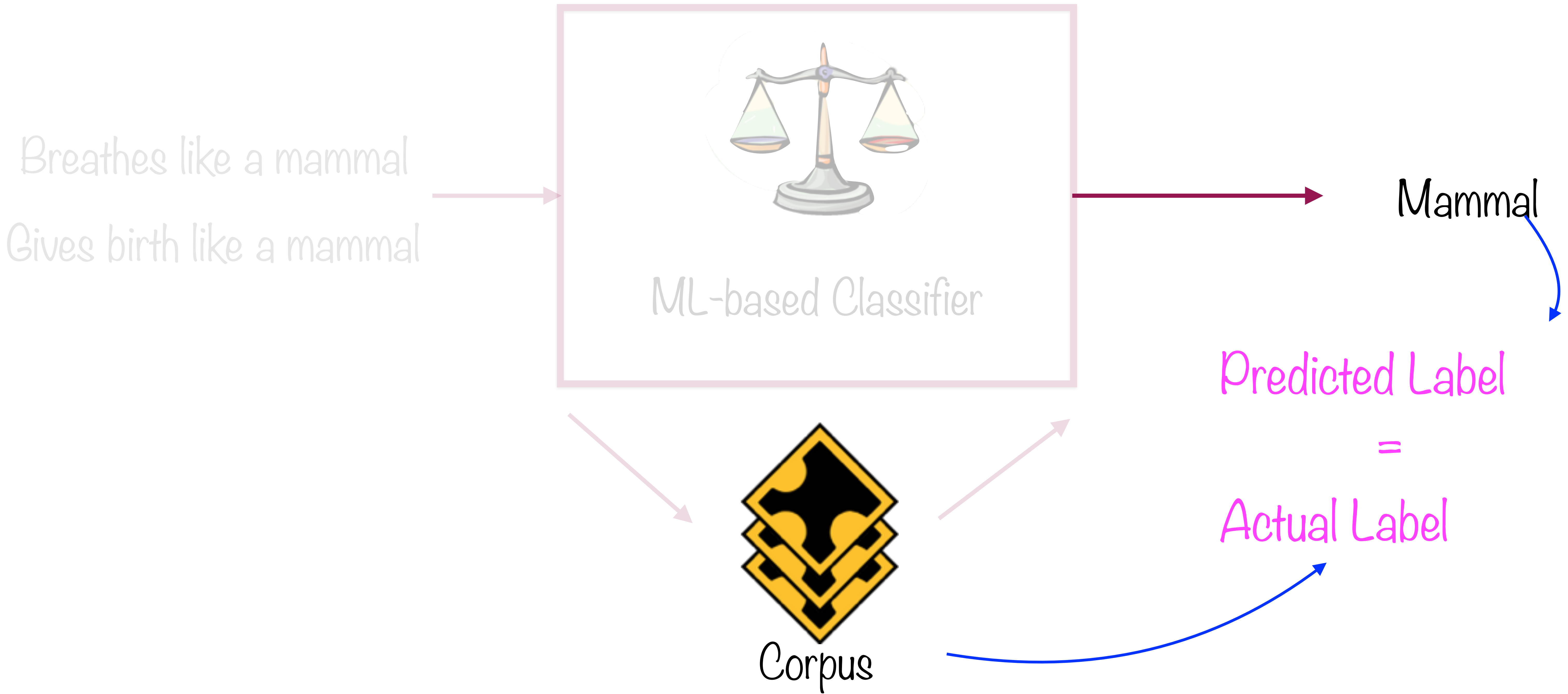


Input: Feature Vector

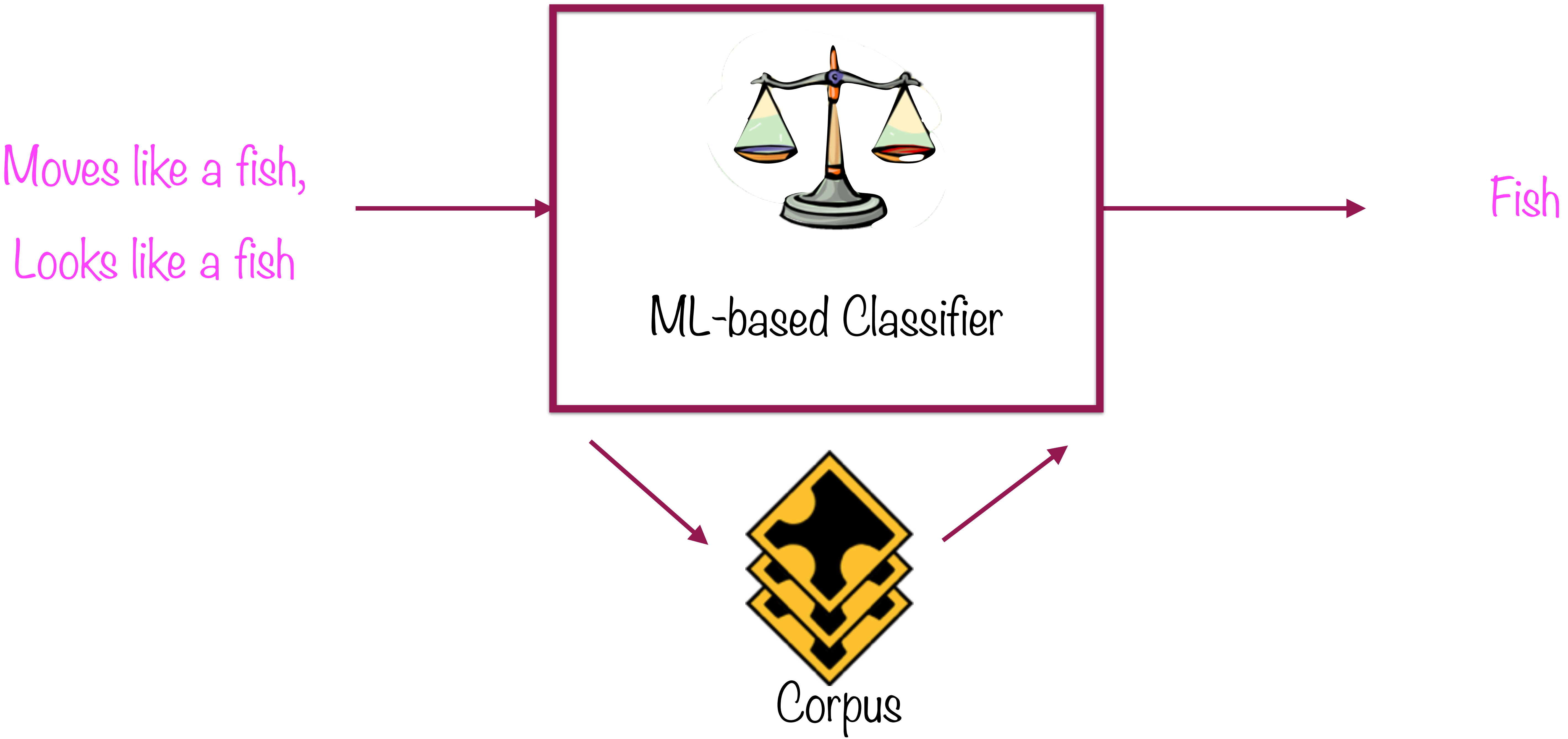
ML-based Binary Classifier



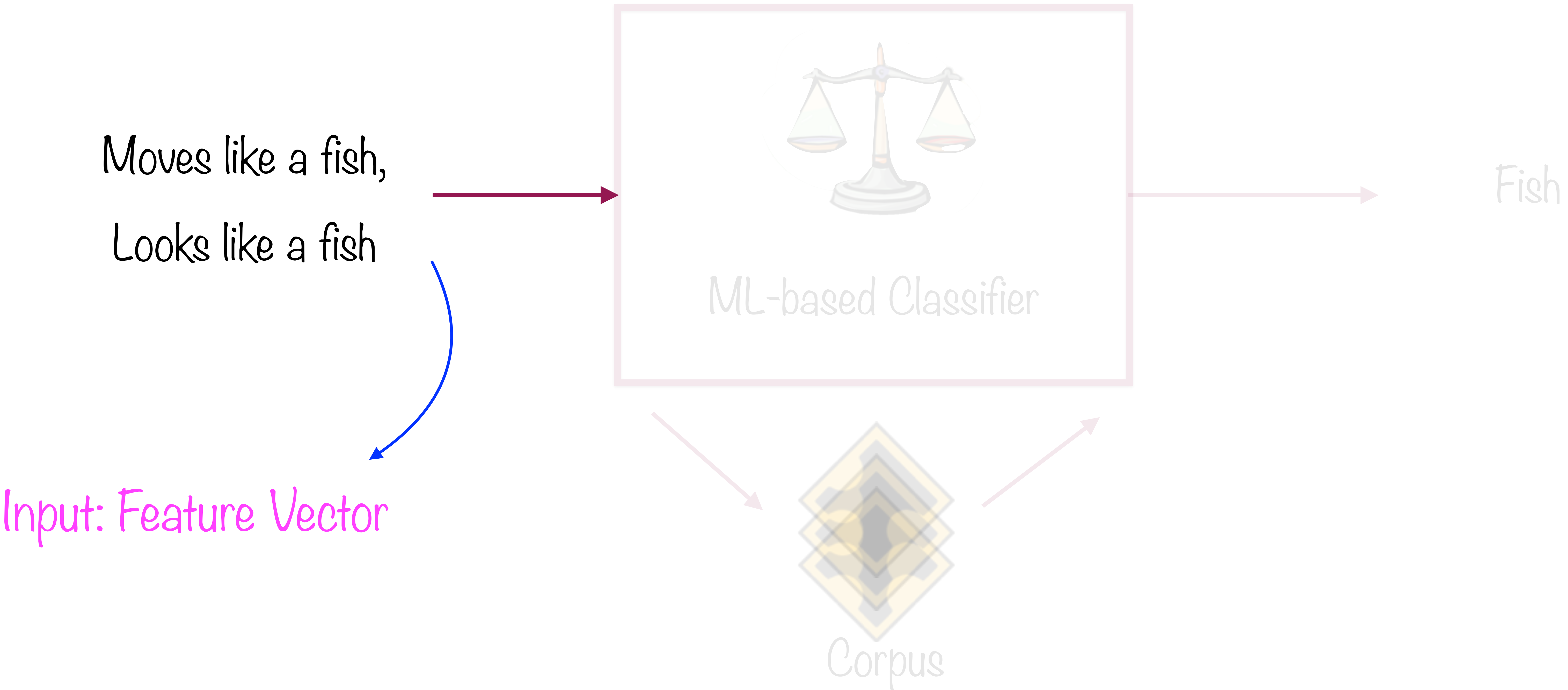
ML-based Binary Classifier



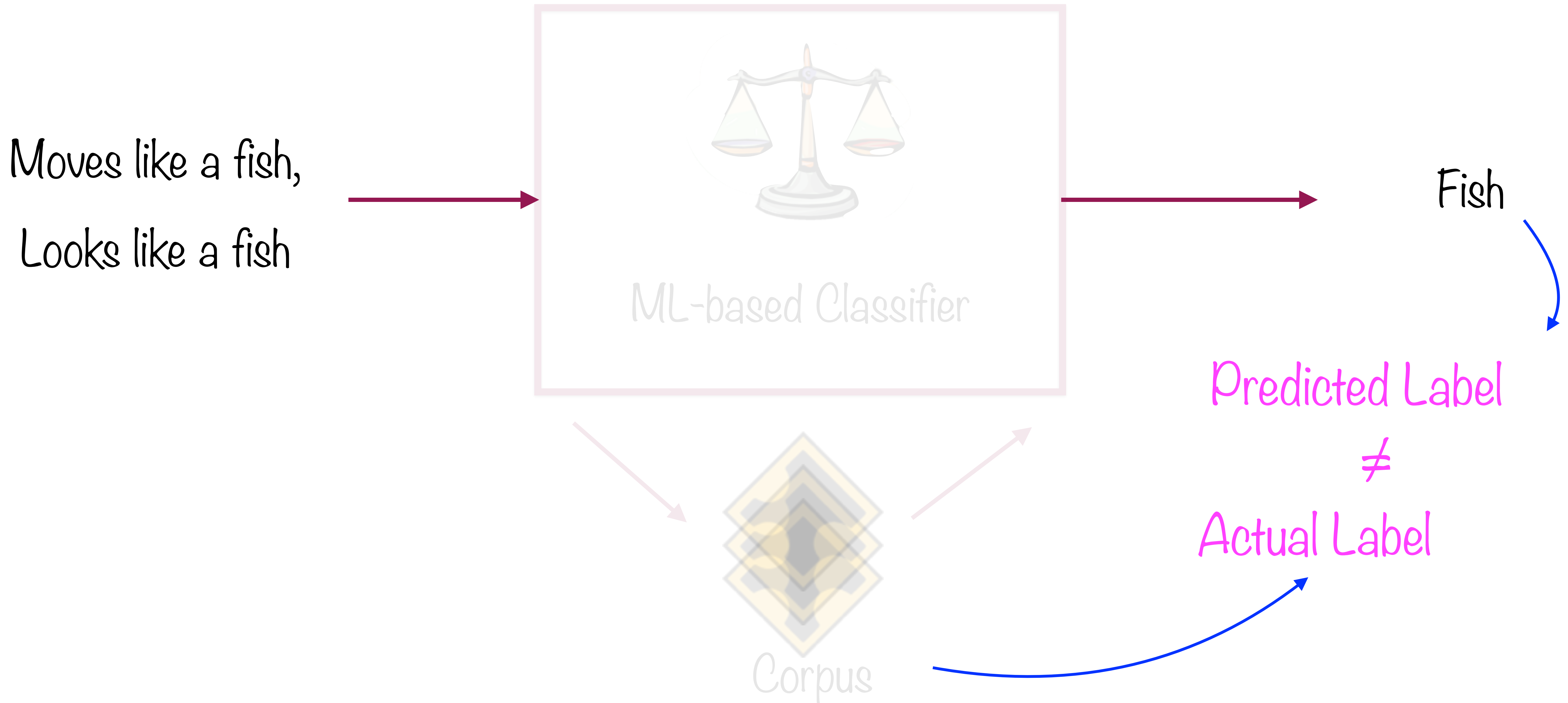
ML-based Binary Classifier



ML-based Binary Classifier

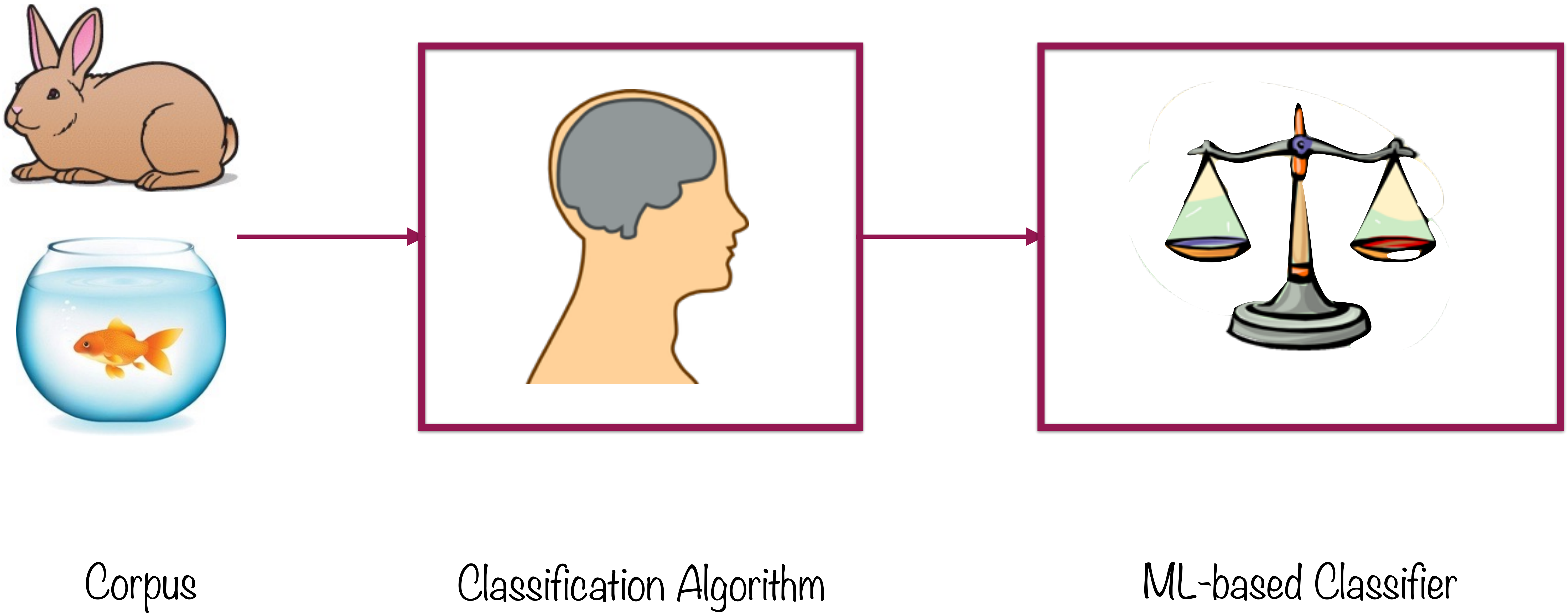


ML-based Binary Classifier

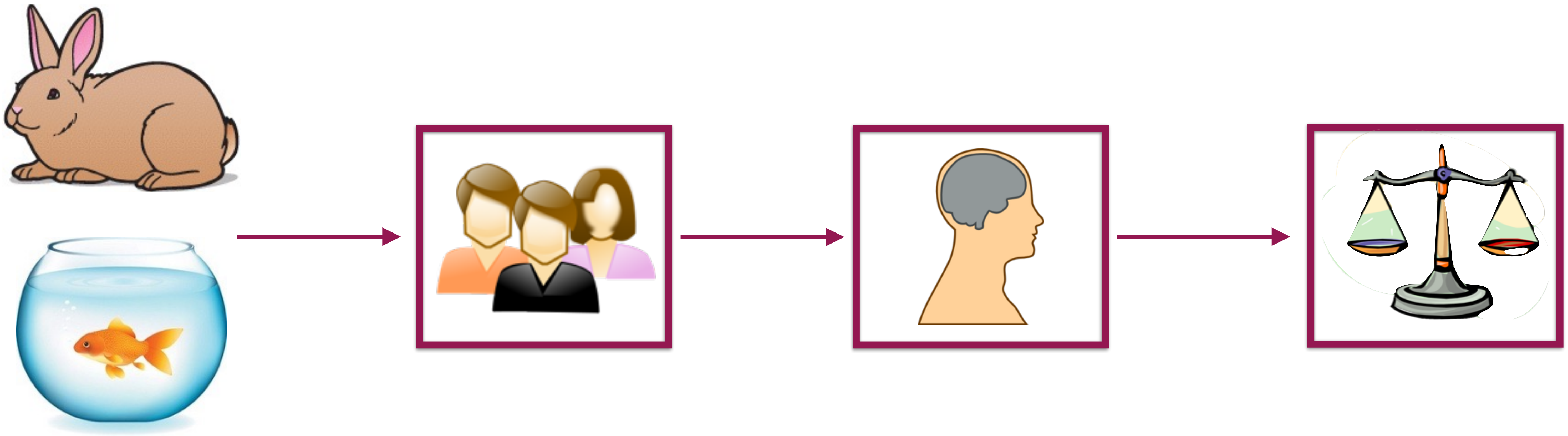


Understanding Deep Learning

“Traditional” ML-based Binary Classifier



"Traditional" ML-based Binary Classifier



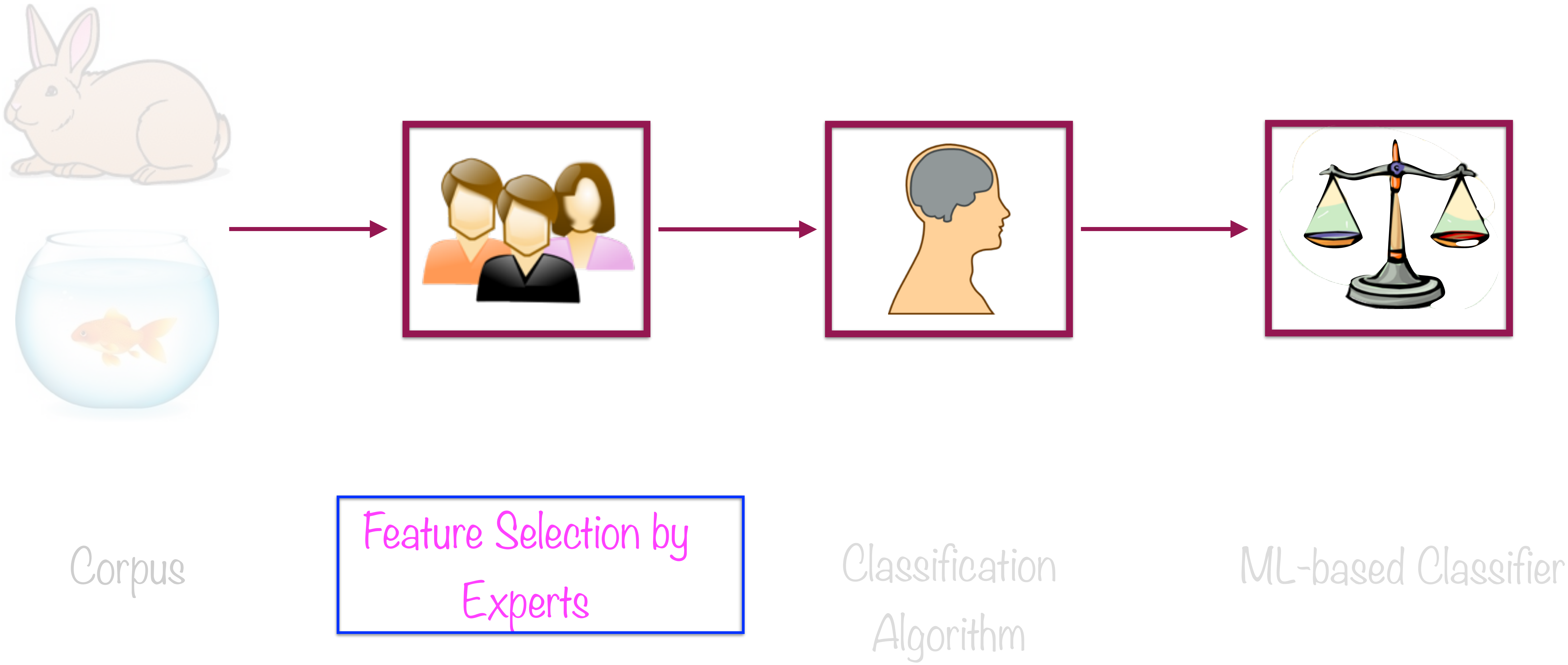
Corpus

Feature Selection by Experts

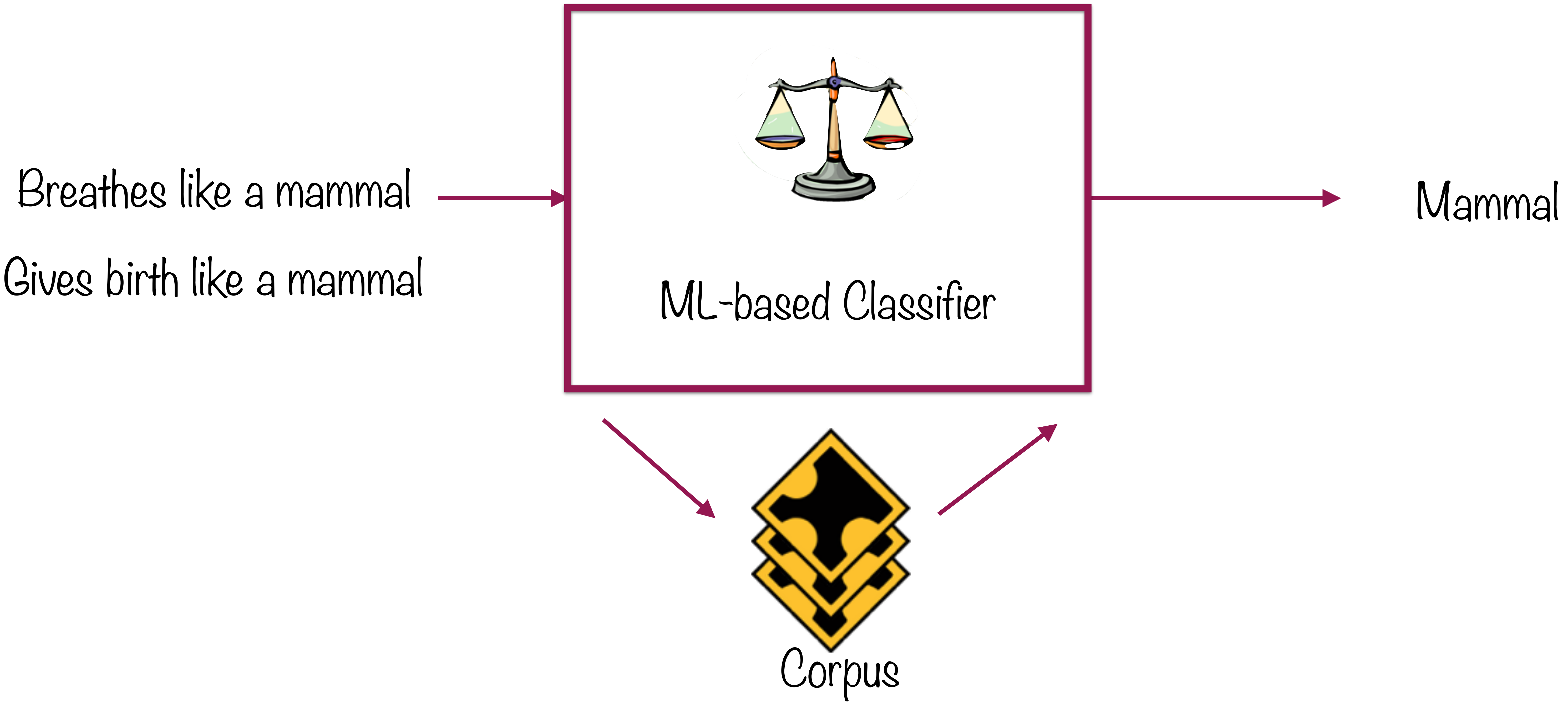
Classification Algorithm

ML-based Classifier

“Traditional” ML-based Binary Classifier

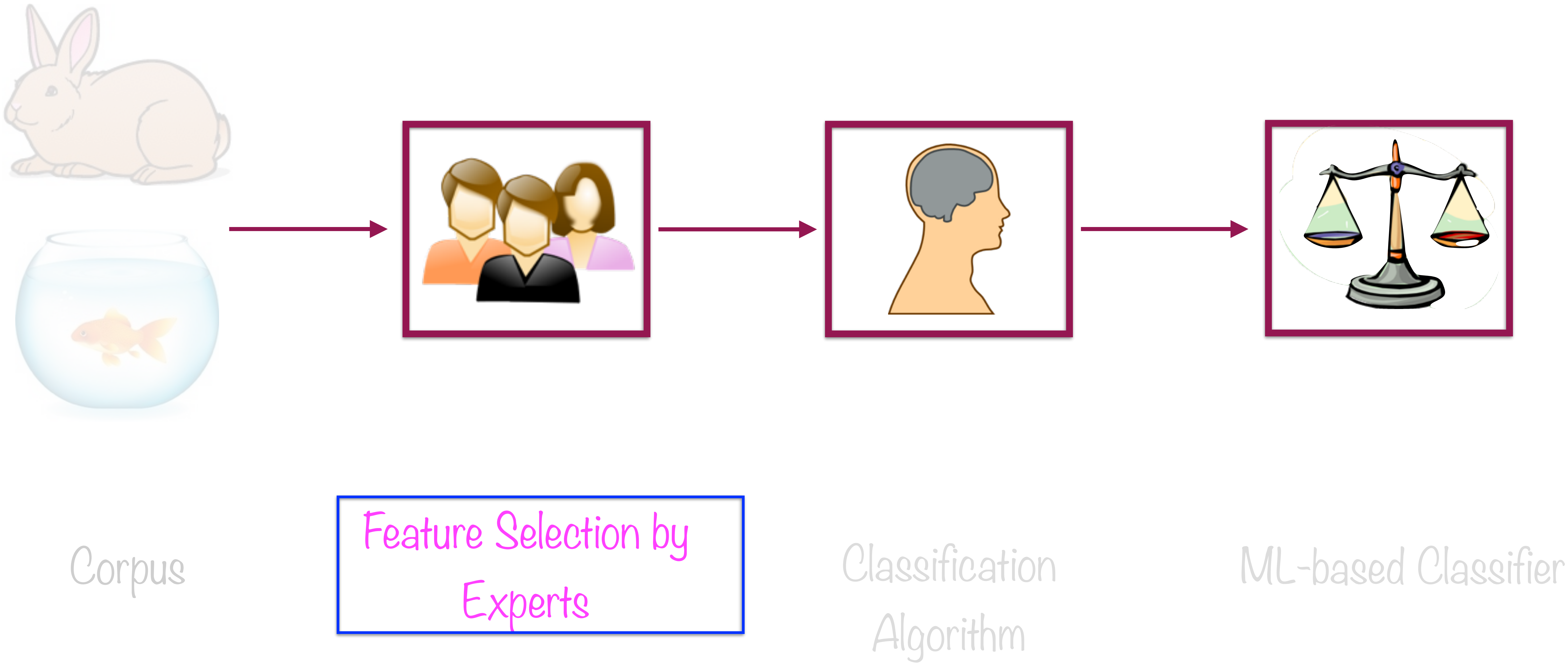


“Traditional” ML-based Binary Classifier

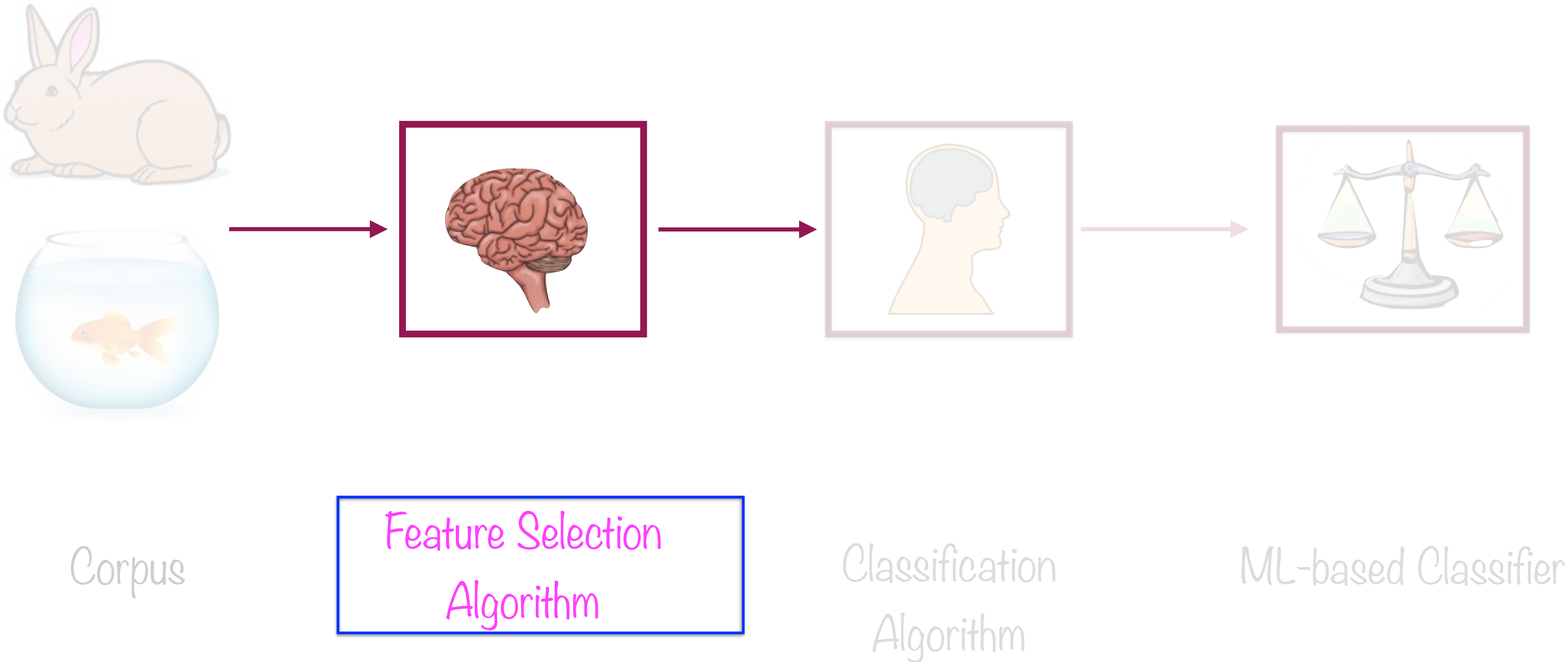


"Traditional" ML-based systems still rely on experts to decide what features to pay attention to

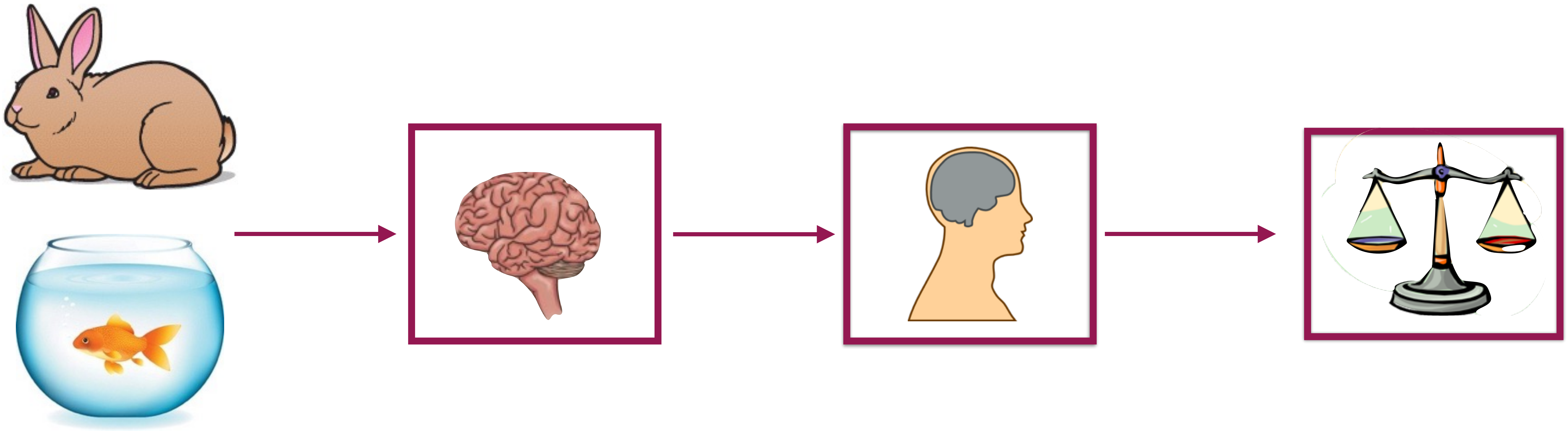
“Traditional” ML-based Binary Classifier



"Representation" ML-based Binary Classifier



“Representation” ML-based Binary Classifier



Corpus

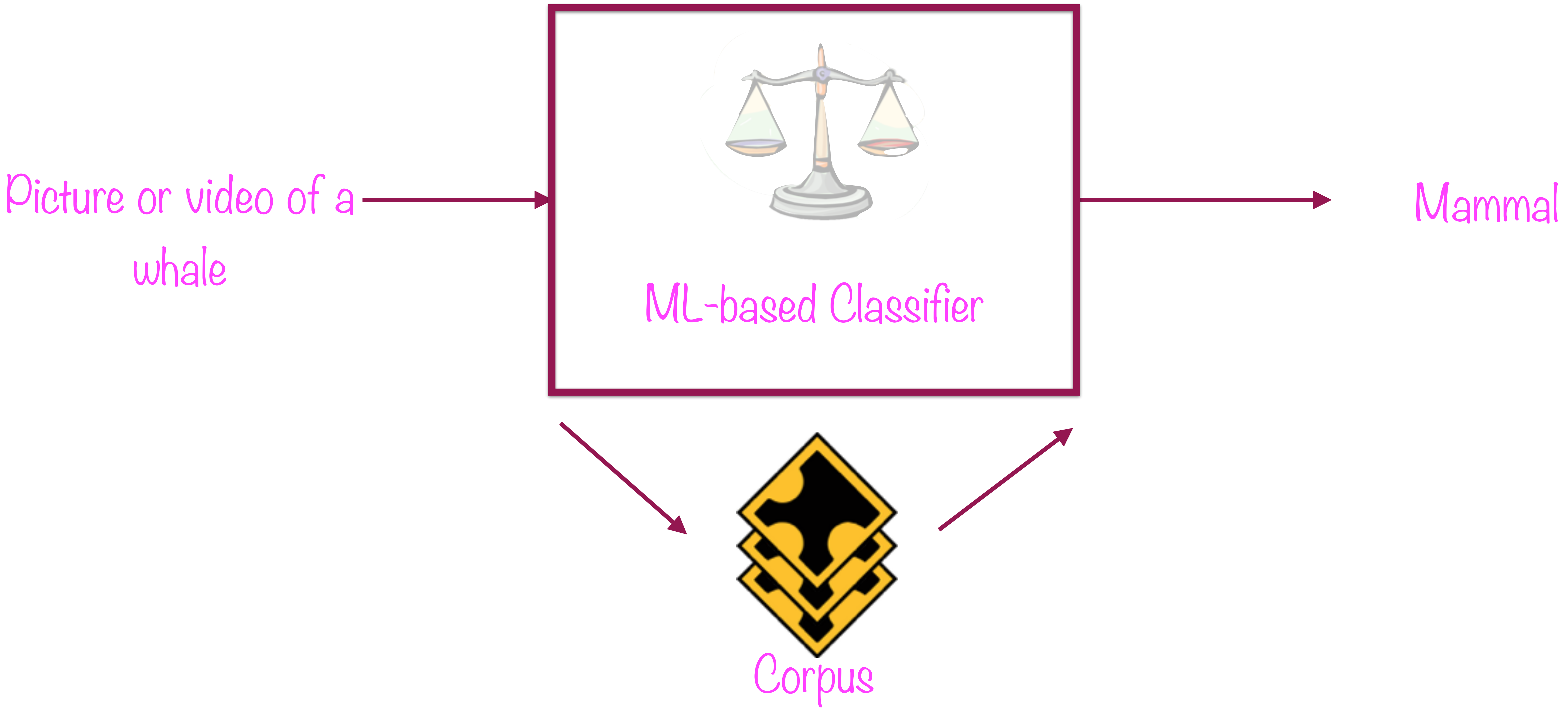
Feature Selection Algorithm

Classification Algorithm

ML-based Classifier

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

"Representation" ML-based Binary Classifier



“Deep Learning” systems are one type of representation systems

Deep Learning and Neural Networks

Deep Learning

Algorithms that learn what features matter

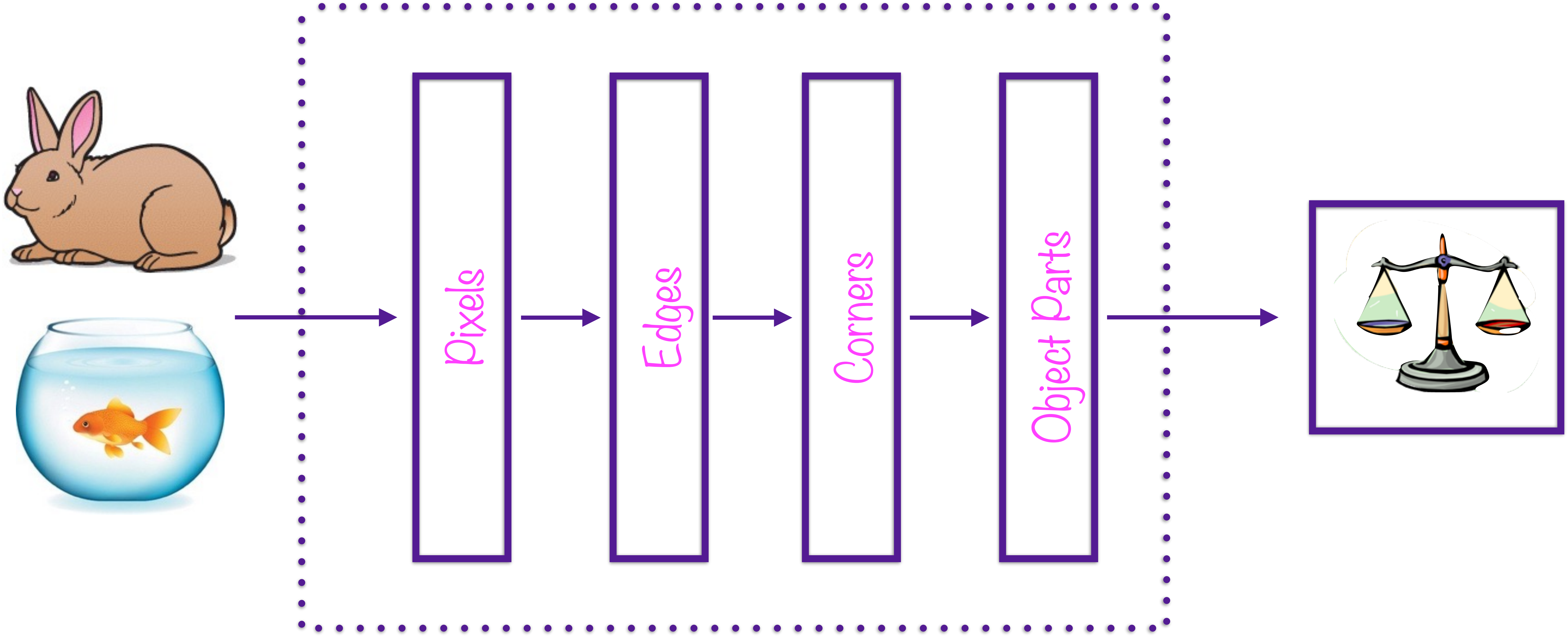
Neural Networks

The most common class of deep learning algorithms

Neurons

Simple building blocks that actually "learn"

"Deep Learning"-based Binary Classifier

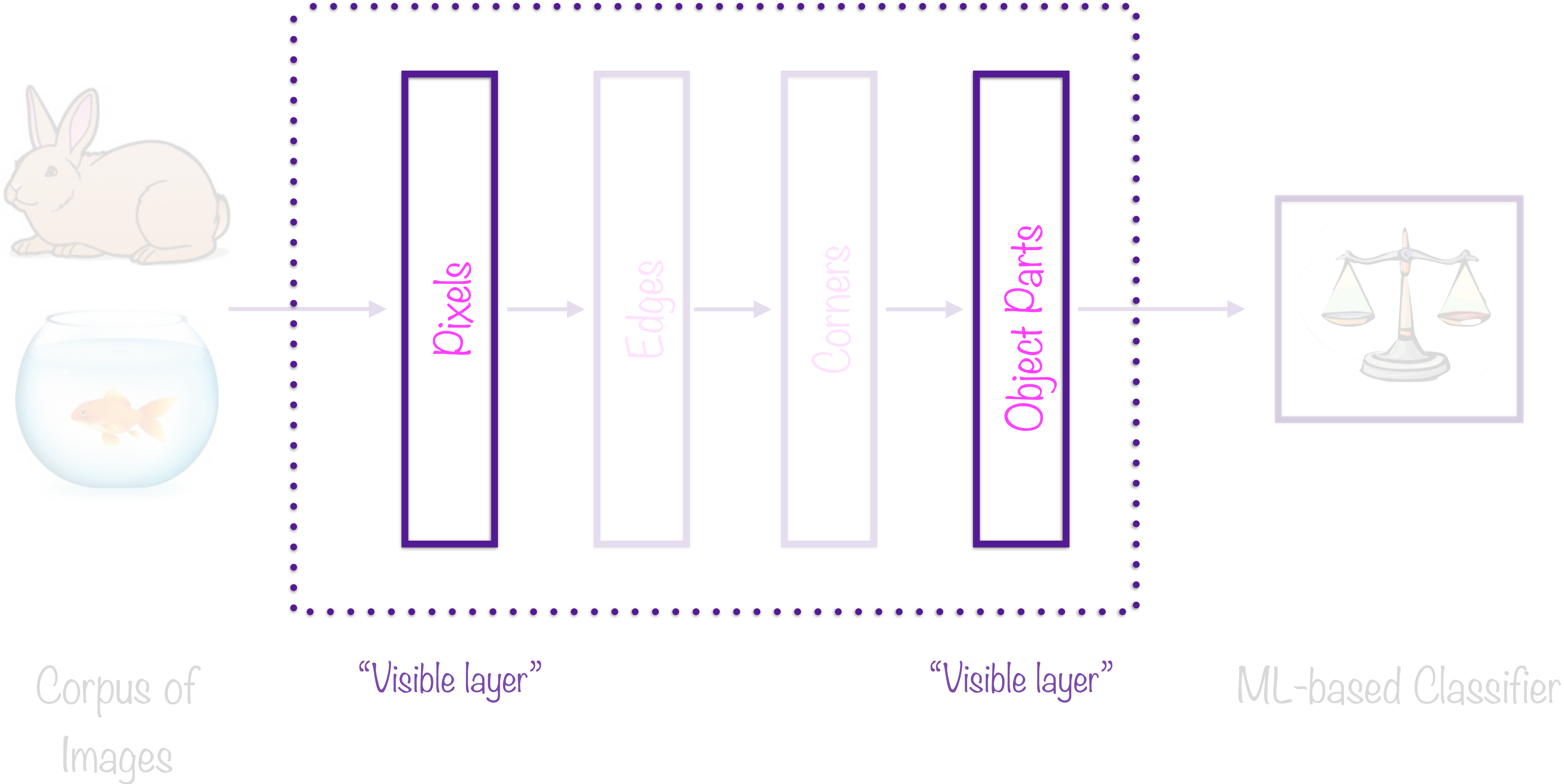


Corpus of Images

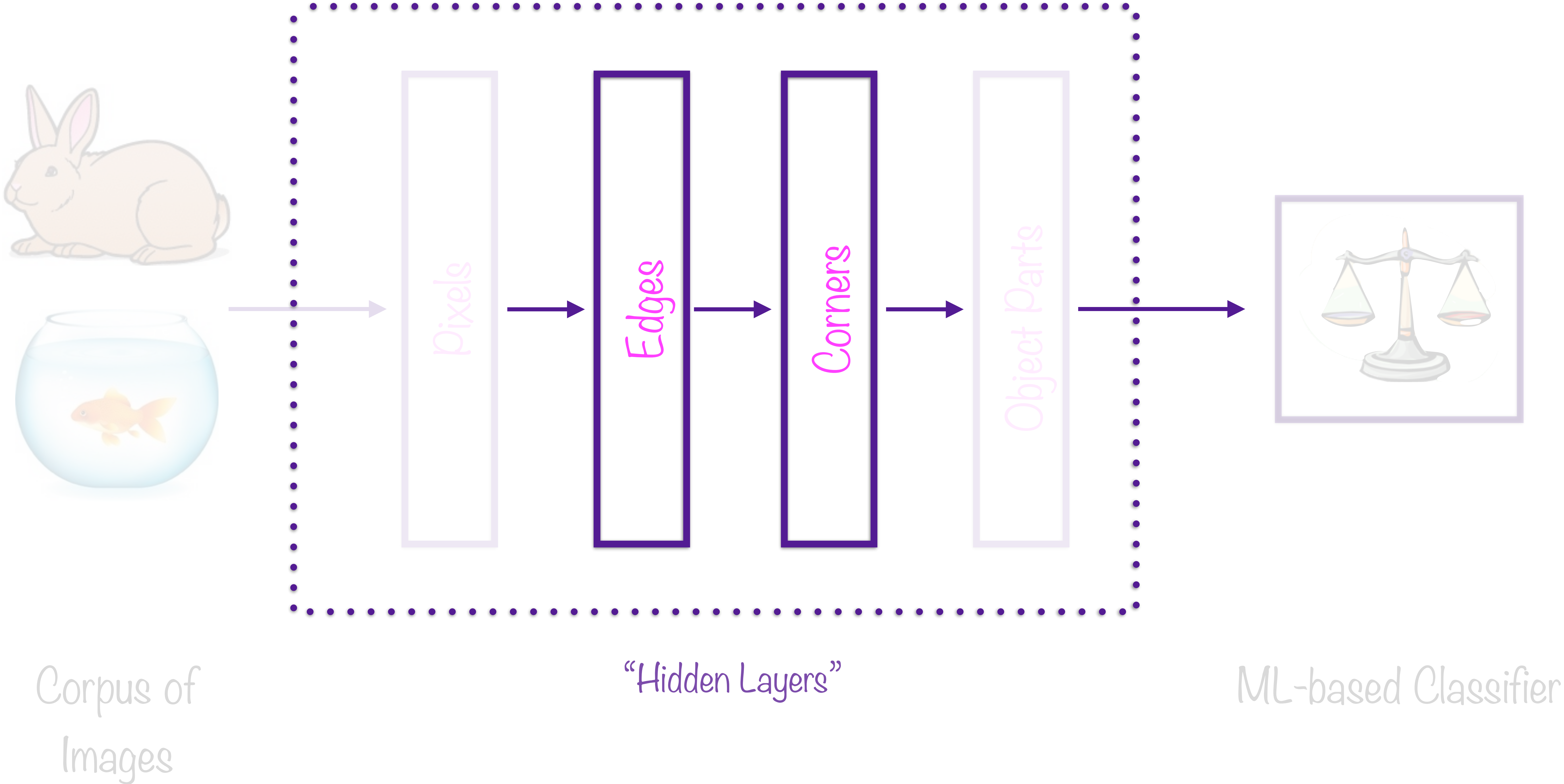
Feature Selection & Classification Algorithm

ML-based Classifier

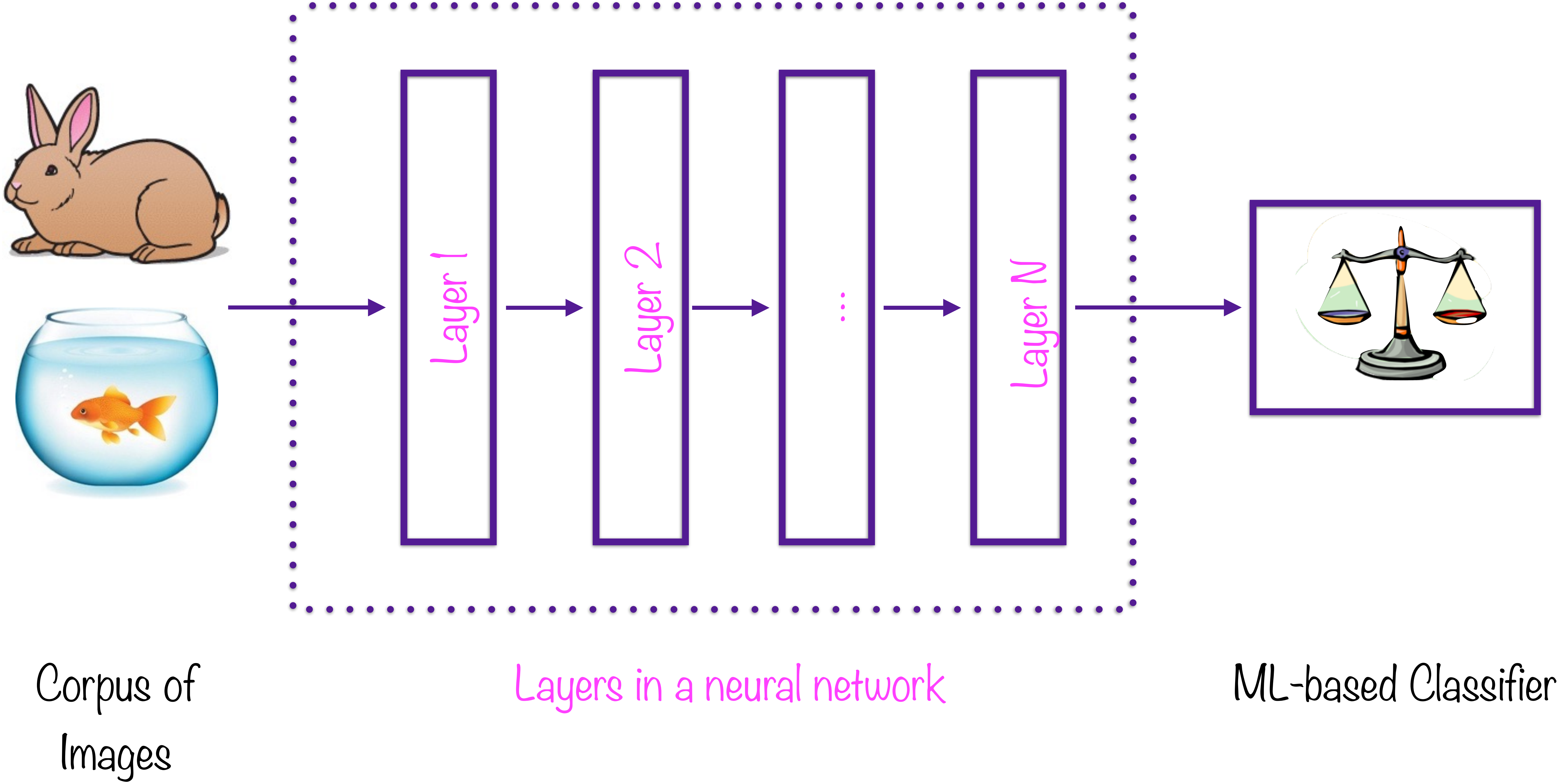
“Deep Learning”-based Binary Classifier



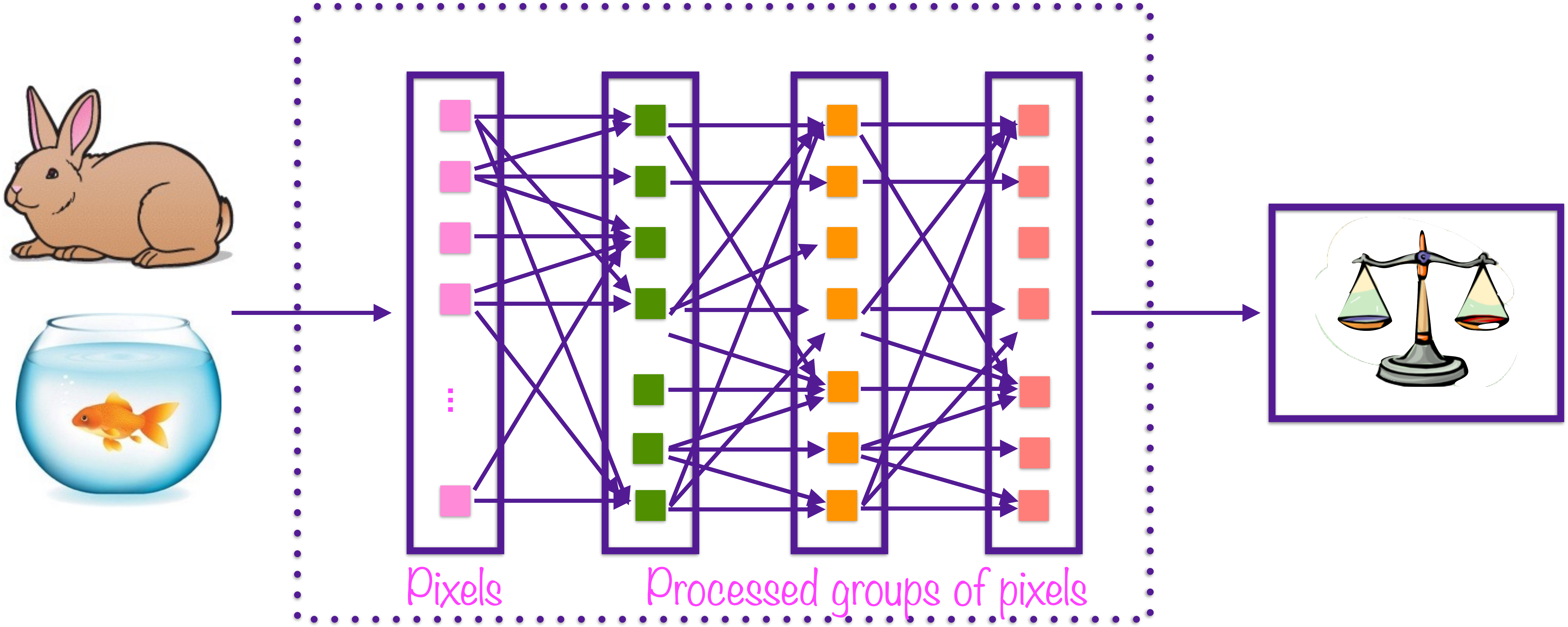
"Deep Learning"-based Binary Classifier



Neural Networks Introduced



Neural Networks Introduced

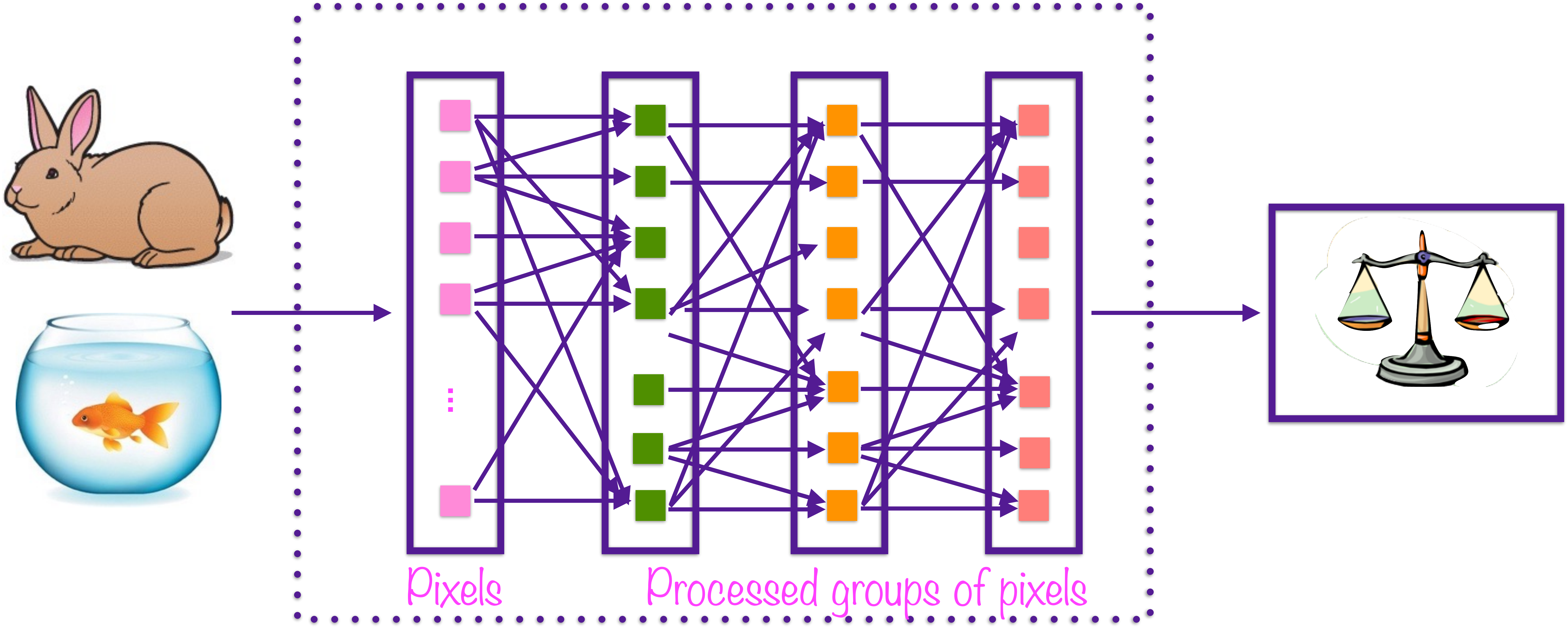


Corpus of Images

Each layer consists of individual interconnected neurons

ML-based Classifier

The Computational Graph

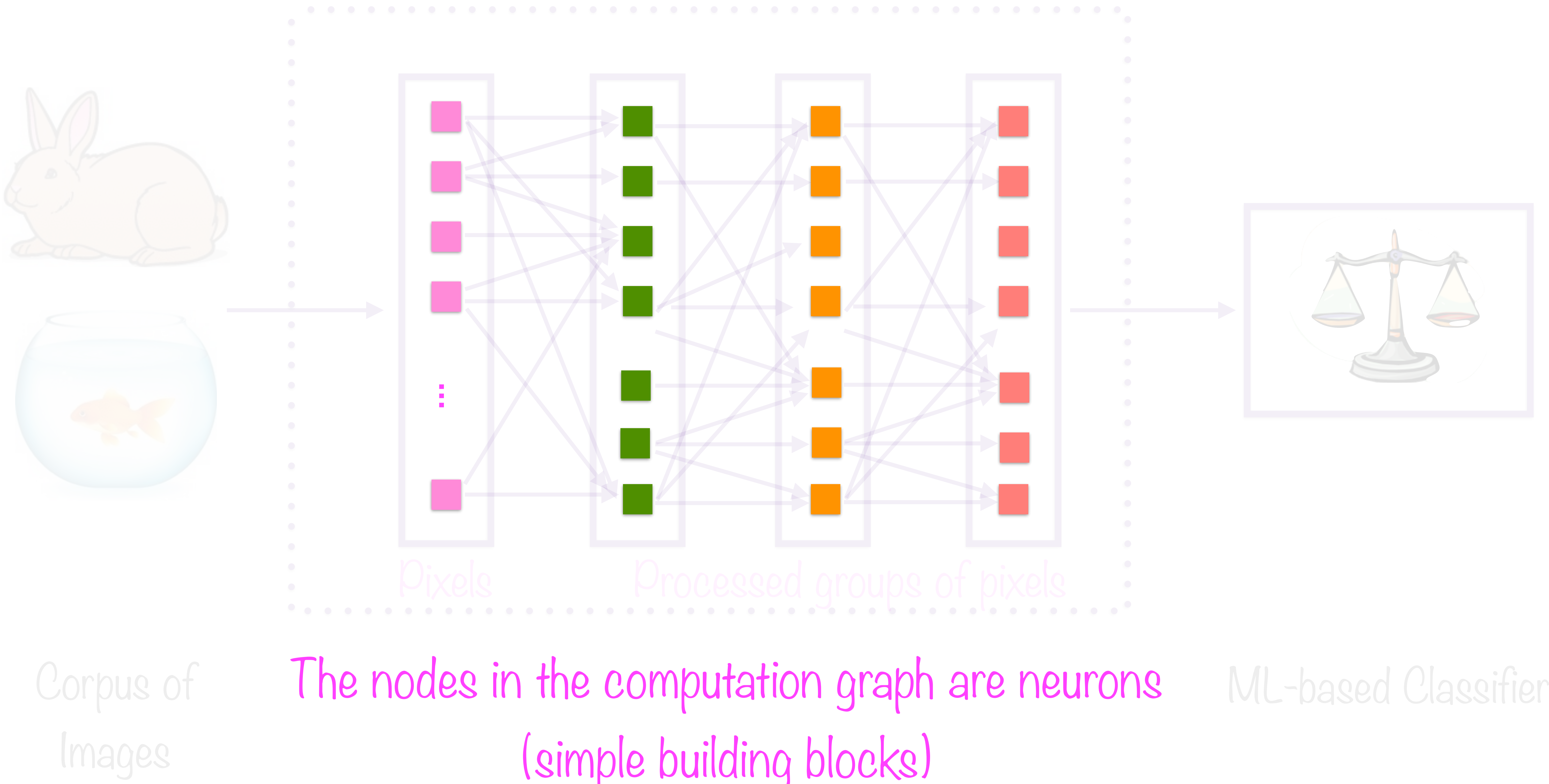


Corpus of Images

Operations (nodes) on data (edges)

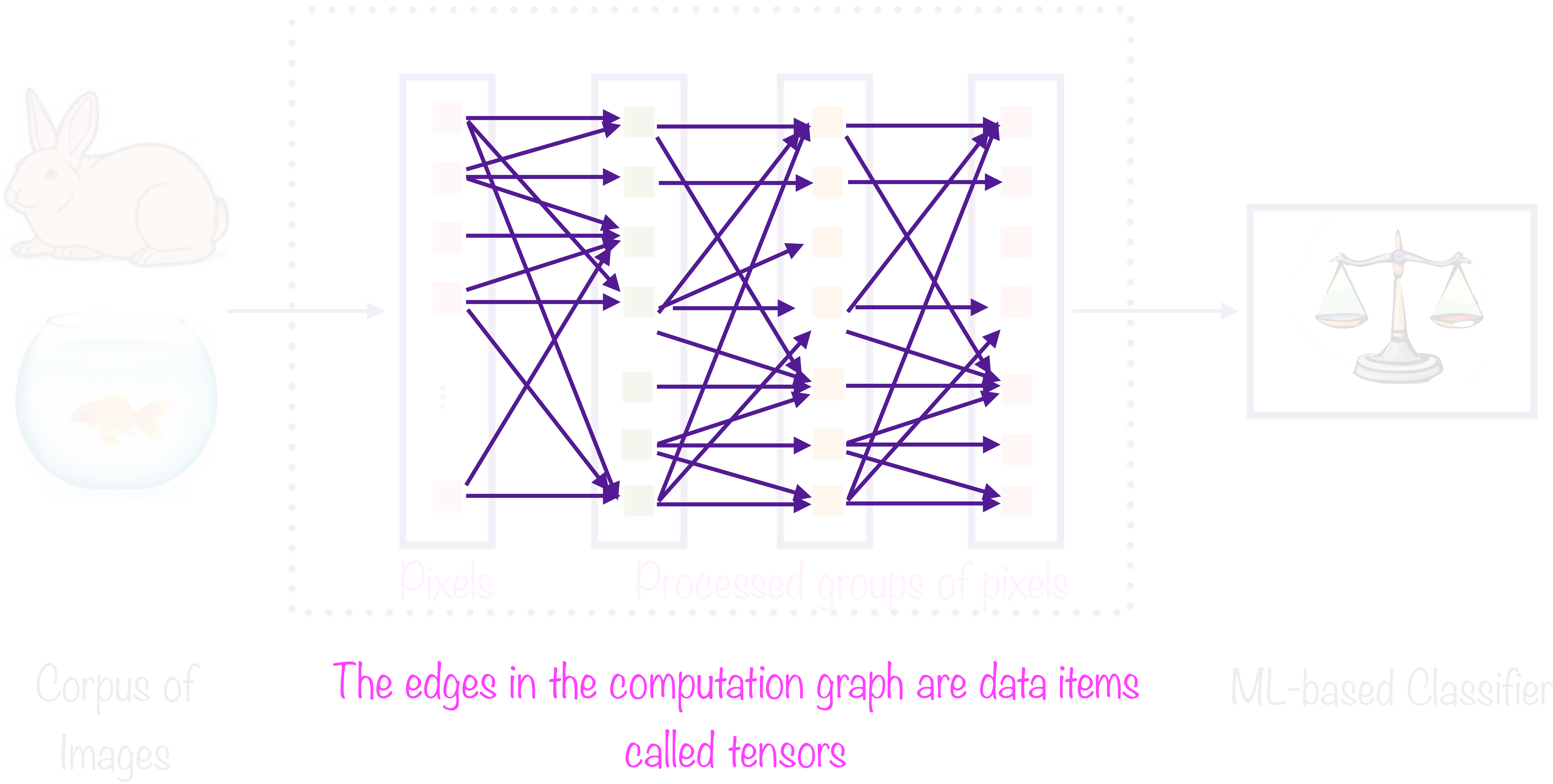
ML-based Classifier

The Computational Graph



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

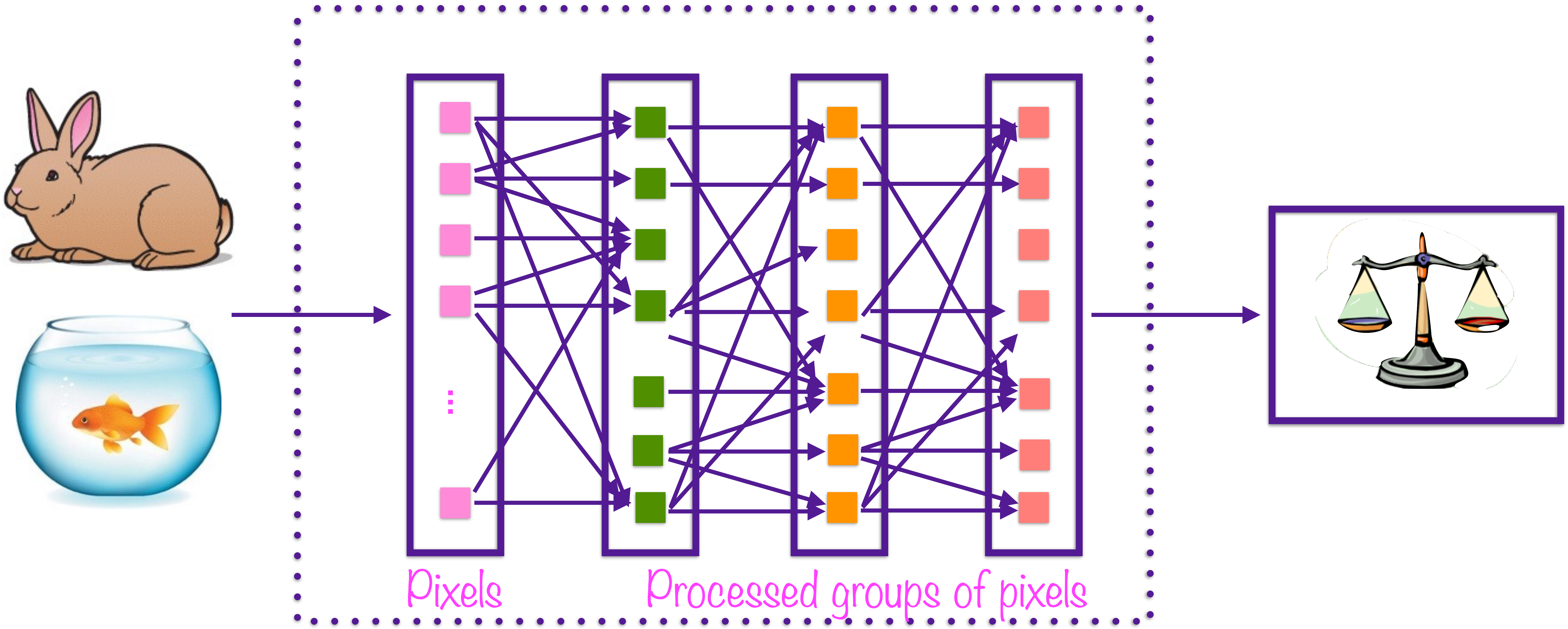
The Computational Graph



The edges in the computation graph are data items called tensors

Neuron as a Learning Unit

A Neural Network

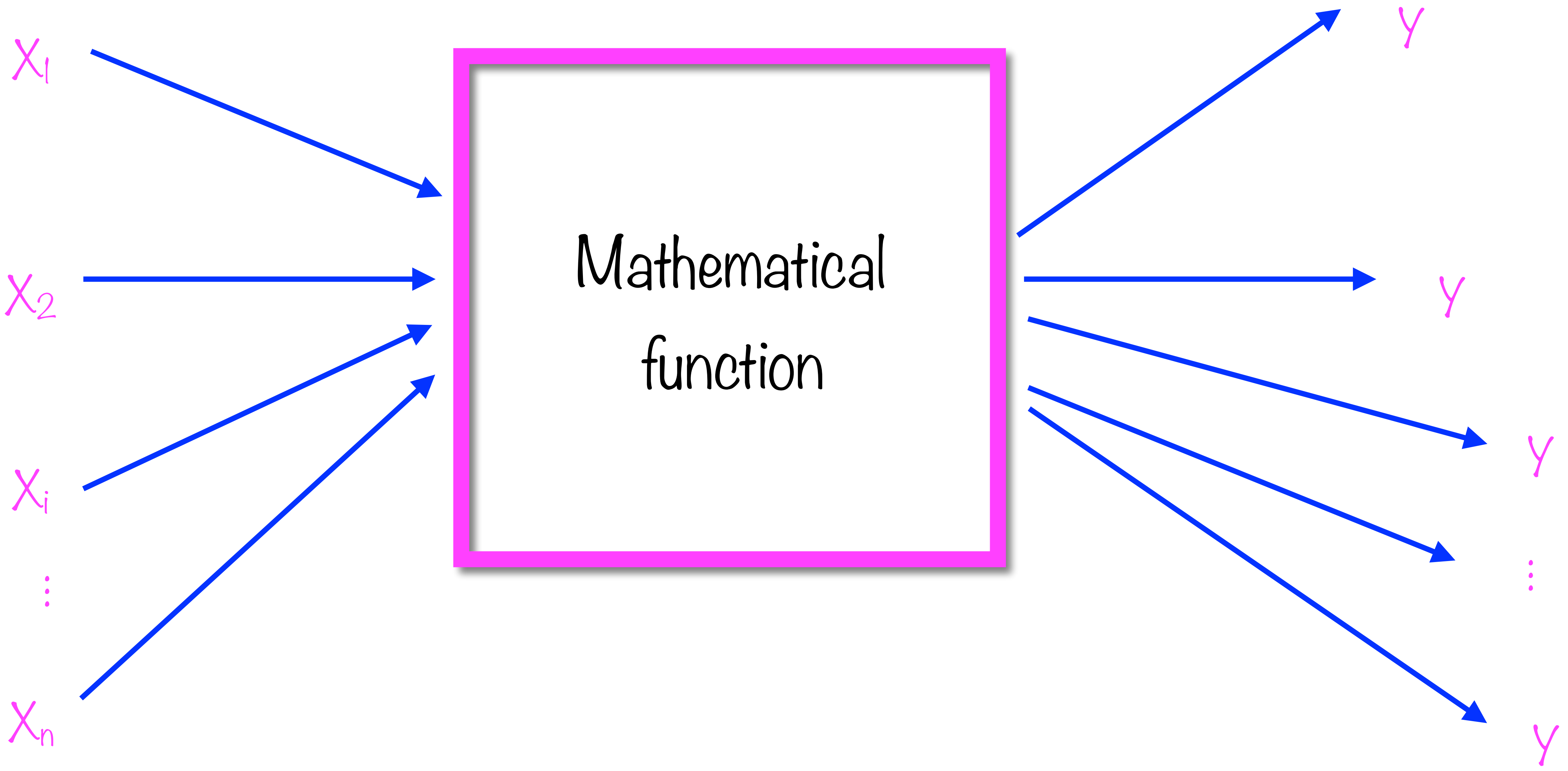


Corpus of Images

Each layer consists of individual interconnected neurons

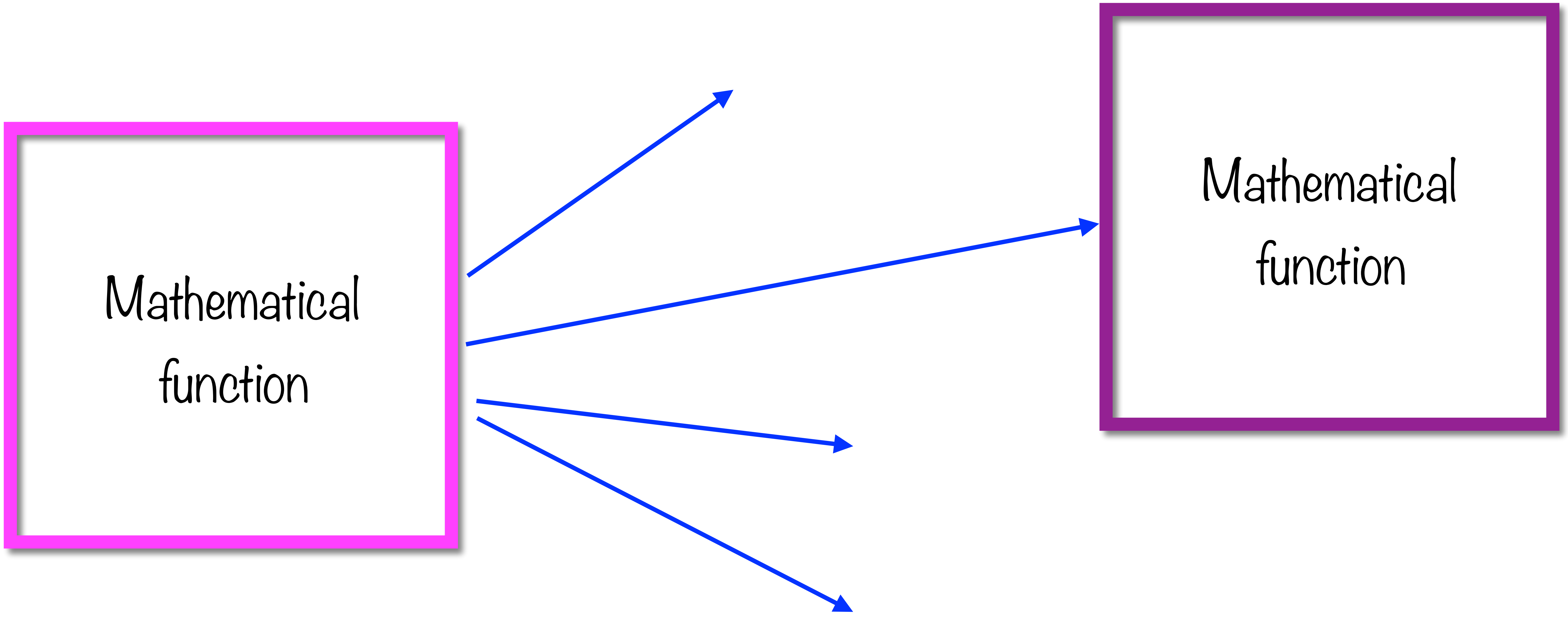
ML-based Classifier

Operation of a Single Neuron



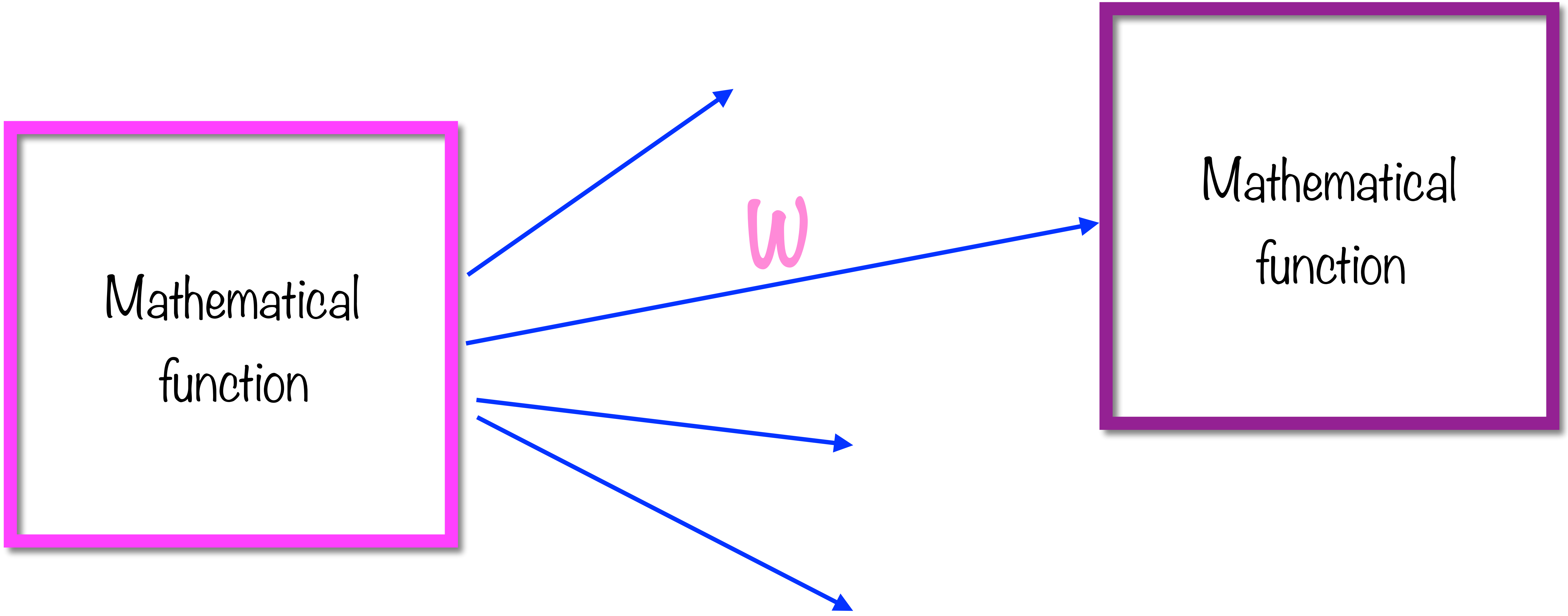
For an active neuron a change in inputs should trigger a corresponding change in the outputs

Operation of a Single Neuron



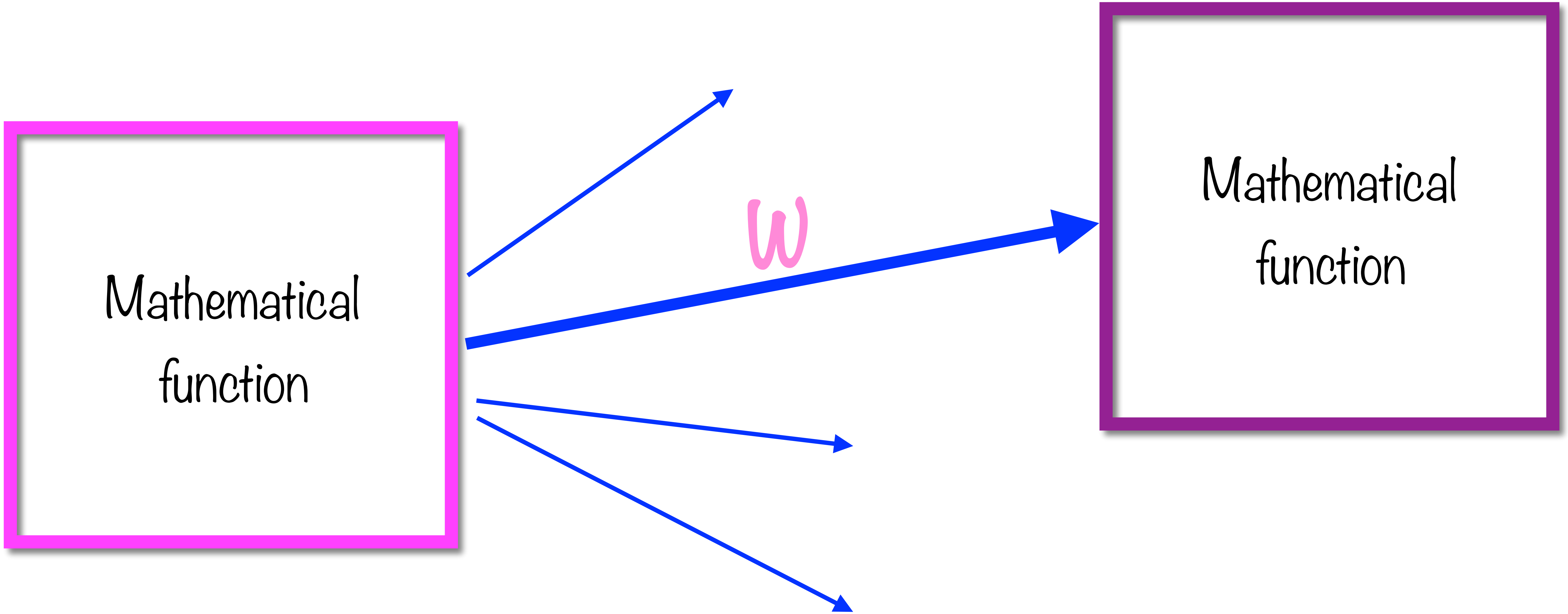
The outputs of neurons feed into the neurons from the next layer

Operation of a Single Neuron



Each connection is associated with a weight

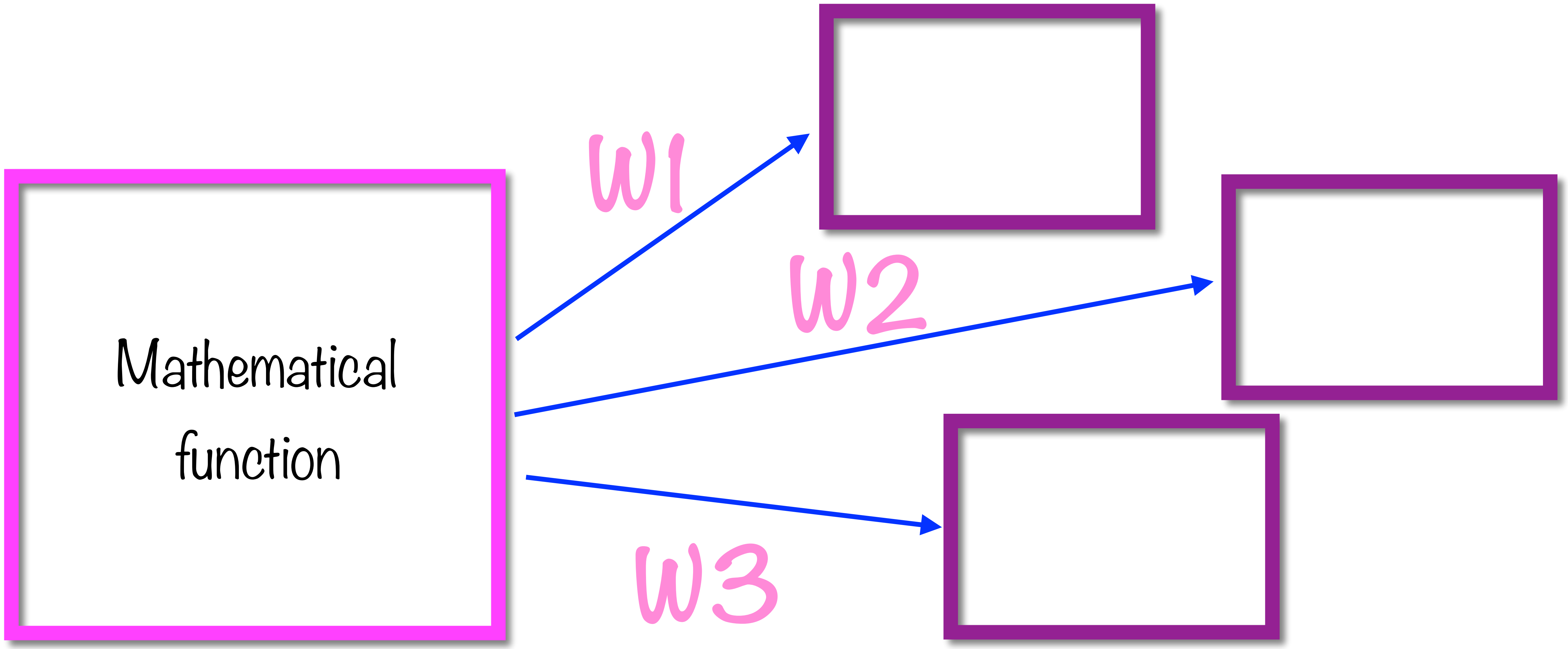
Operation of a Single Neuron



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

W increases

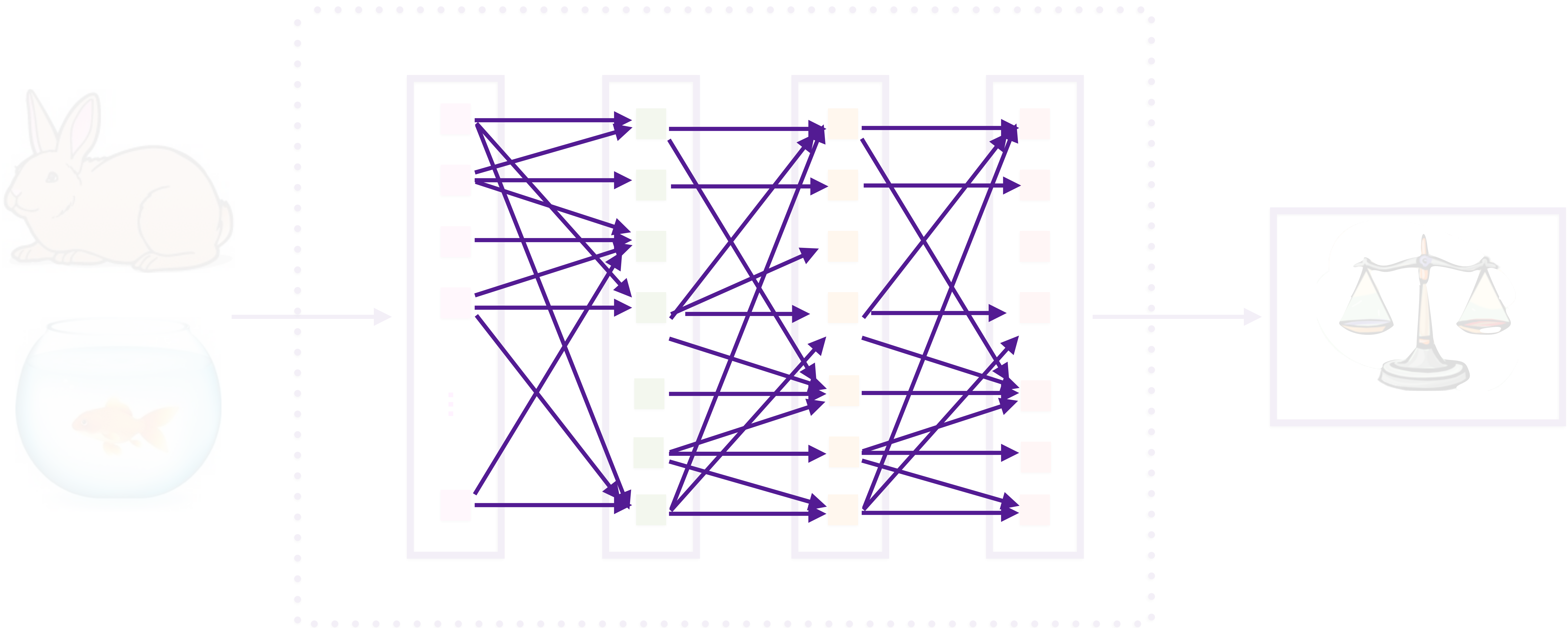
Operation of a Single Neuron



A single neuron is generally connected to multiple neurons in the next layer.

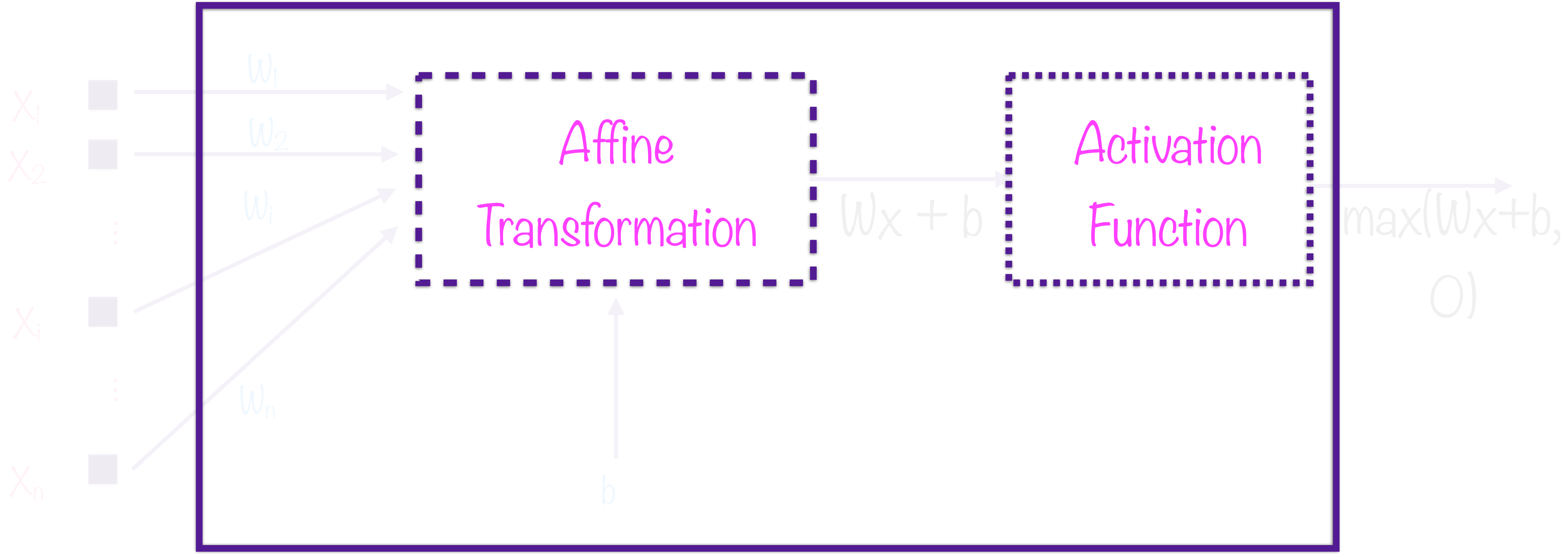
Each connection will have its **own weight**

A Neural Network



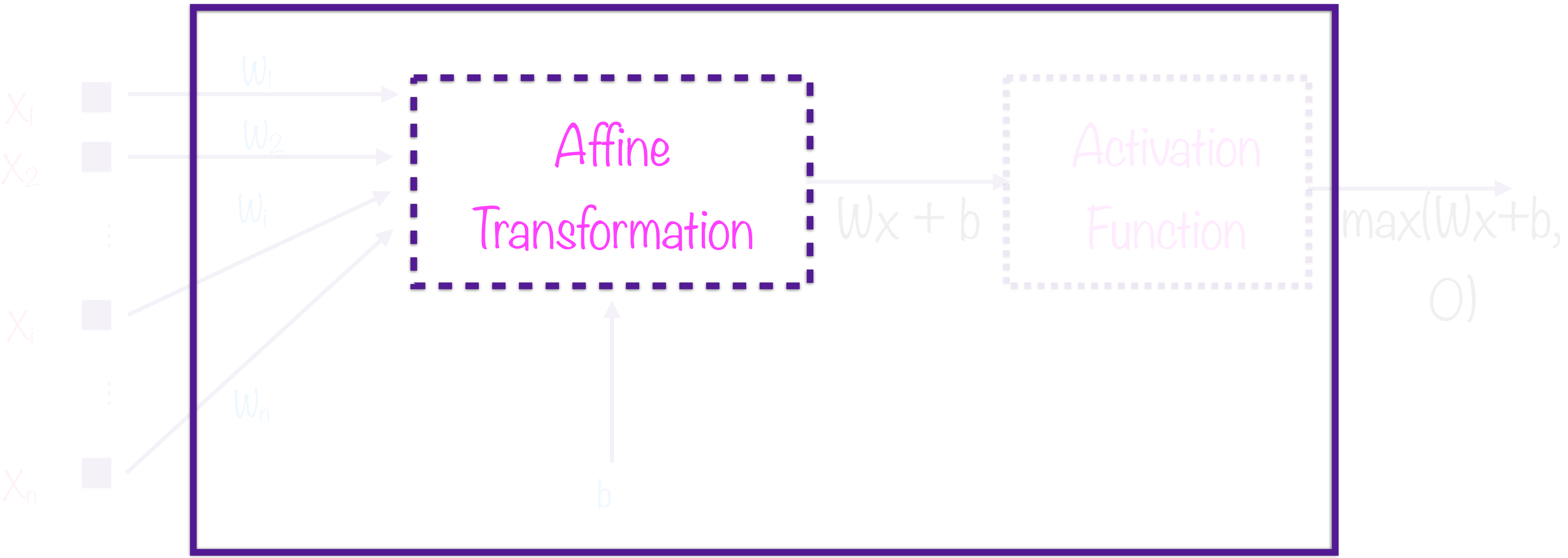
Once a neural network is trained all edges have weights which help it make predictions

Operation of a Single Neuron



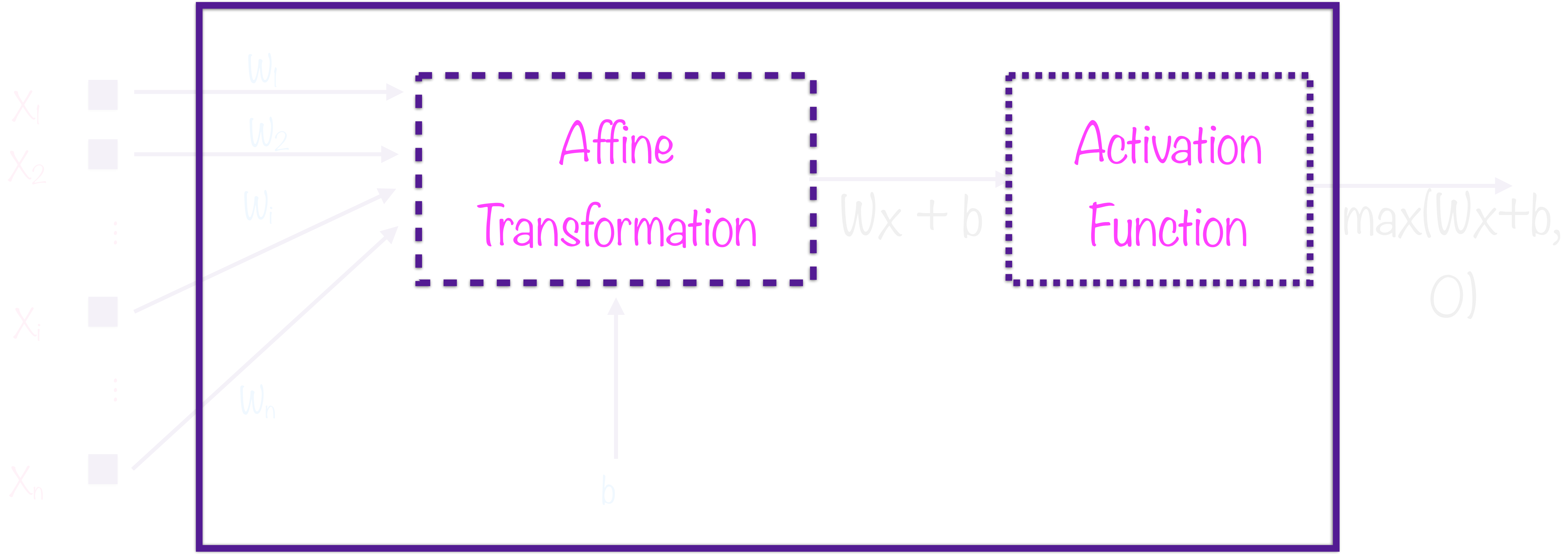
Each neuron only applies two simple functions to its inputs

Operation of a Single Neuron



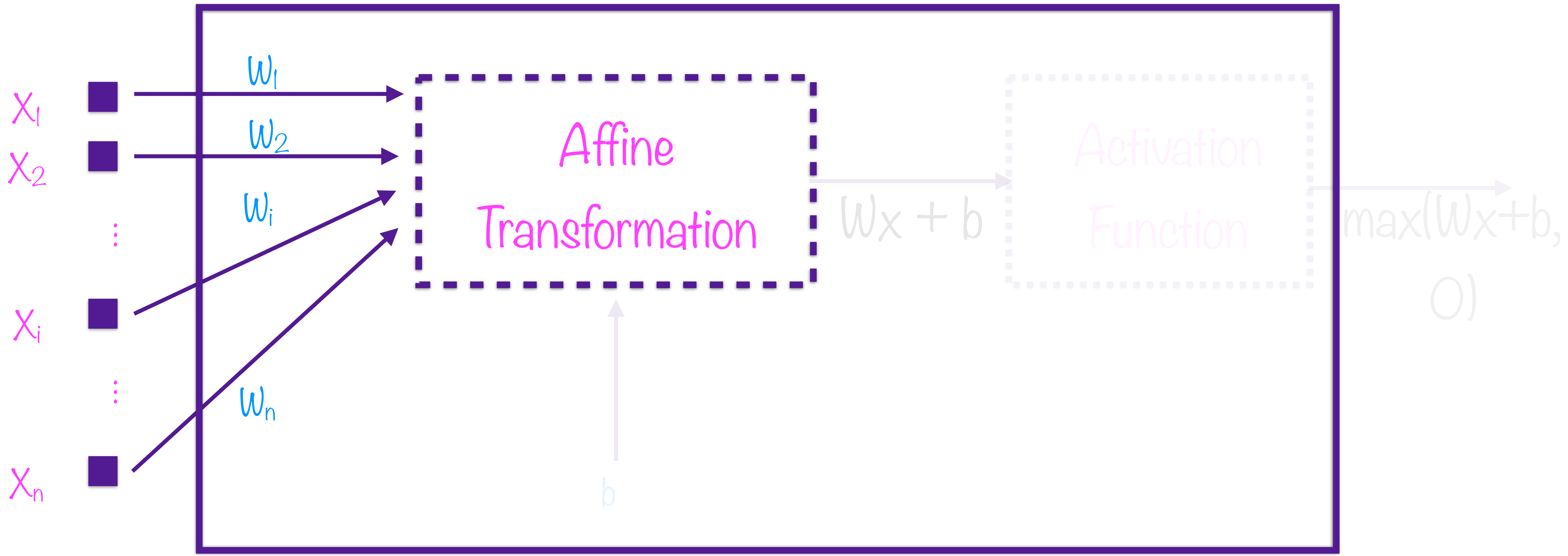
The affine transformation alone can only learn linear relationships between the inputs and the output

Operation of a Single Neuron



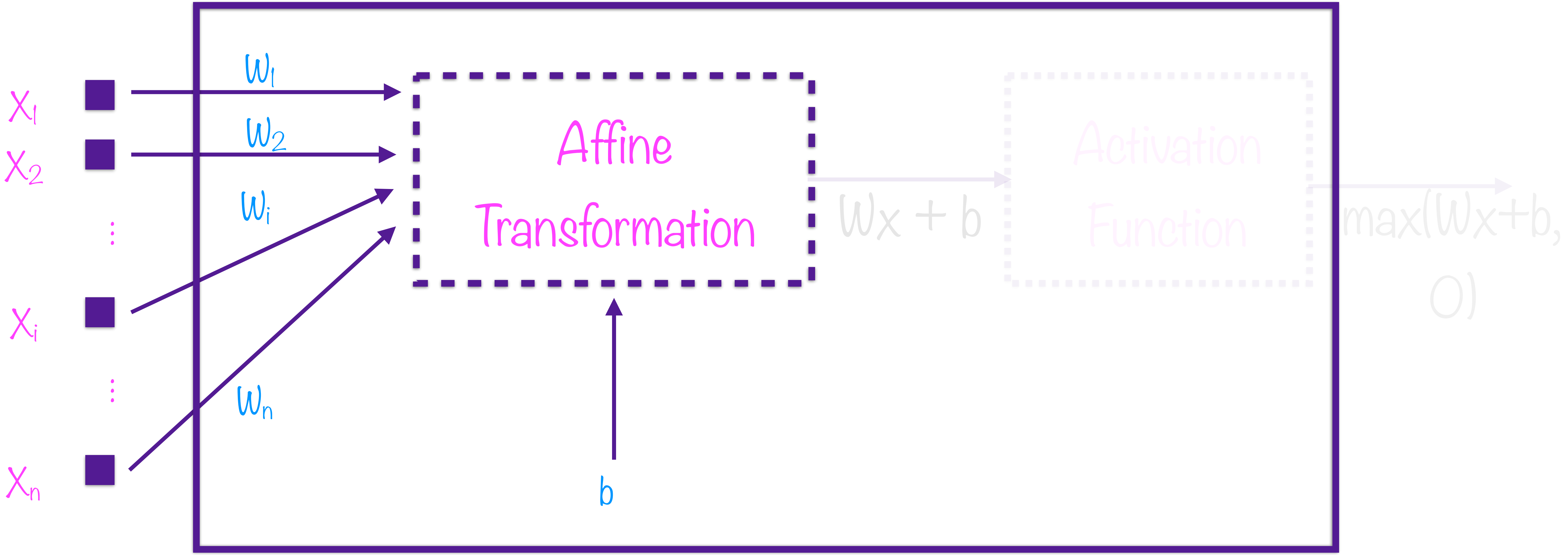
The combination of the affine transformation and the activation function can learn any arbitrary relationship

Operation of a Single Neuron



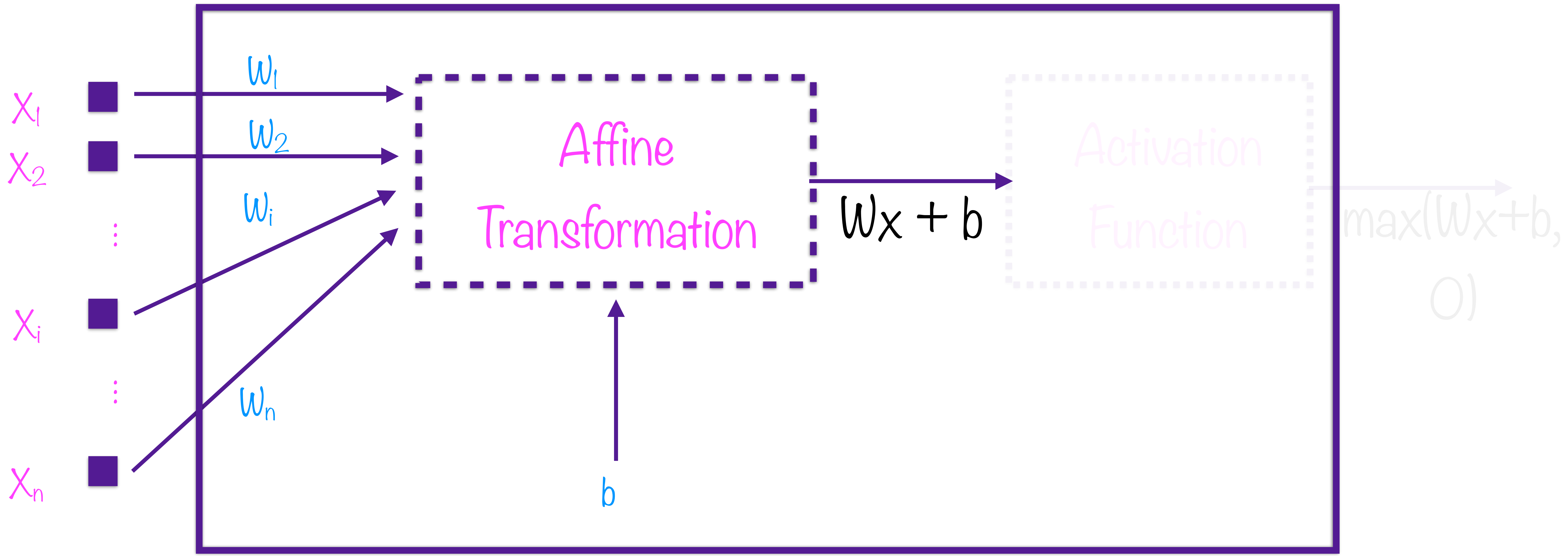
The values W_1, W_2...W_n are called the weights

Operation of a Single Neuron



The value b is called the bias

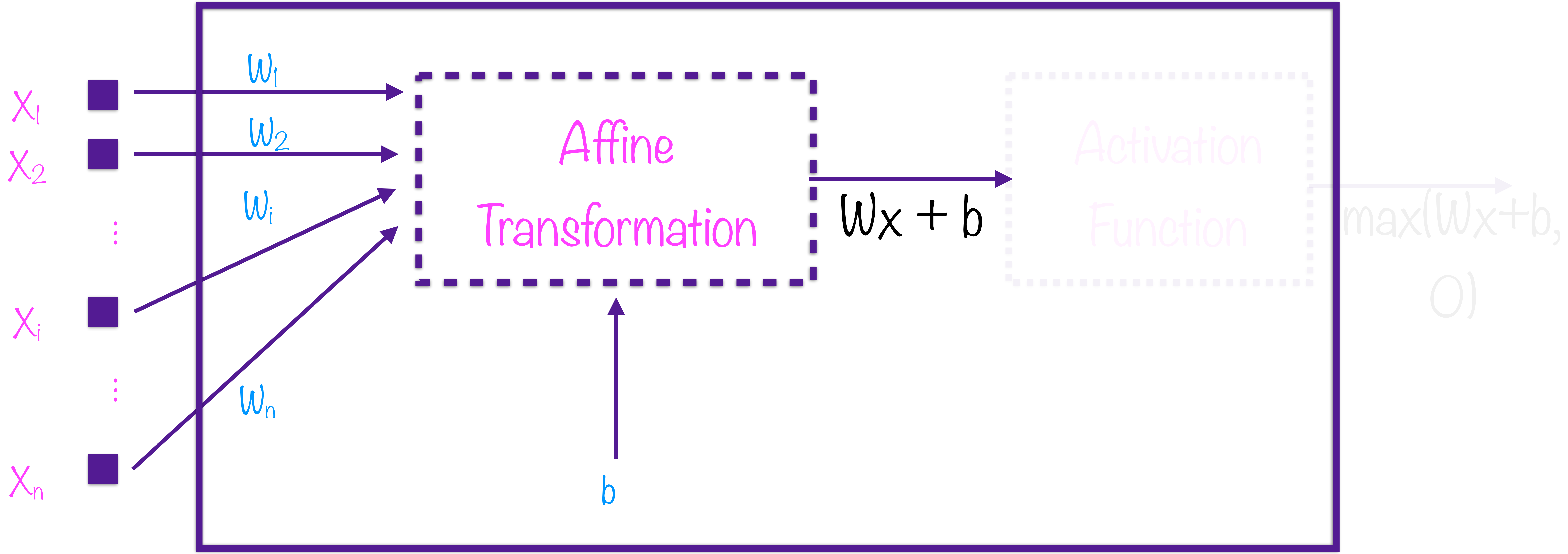
Operation of a Single Neuron



The affine transformation is just a weighted sum with a bias added:

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

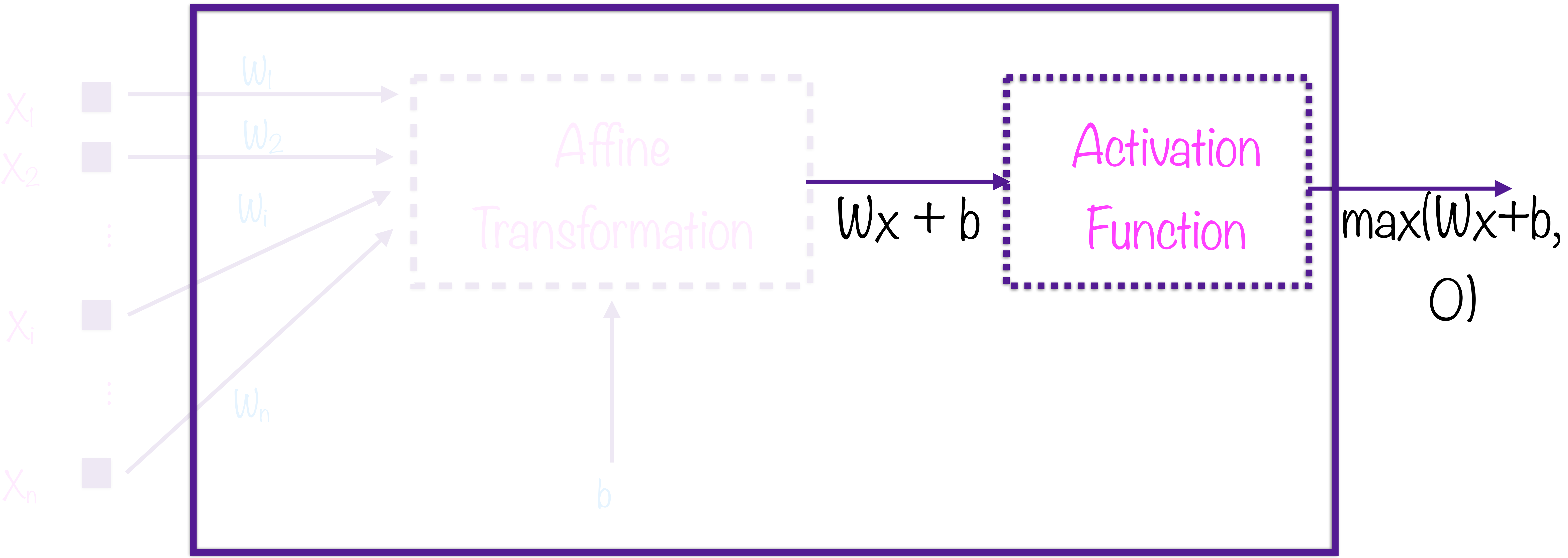
Operation of a Single Neuron



Where do the values of W and b come from?

The weights and biases of a neuron are determined by the training process

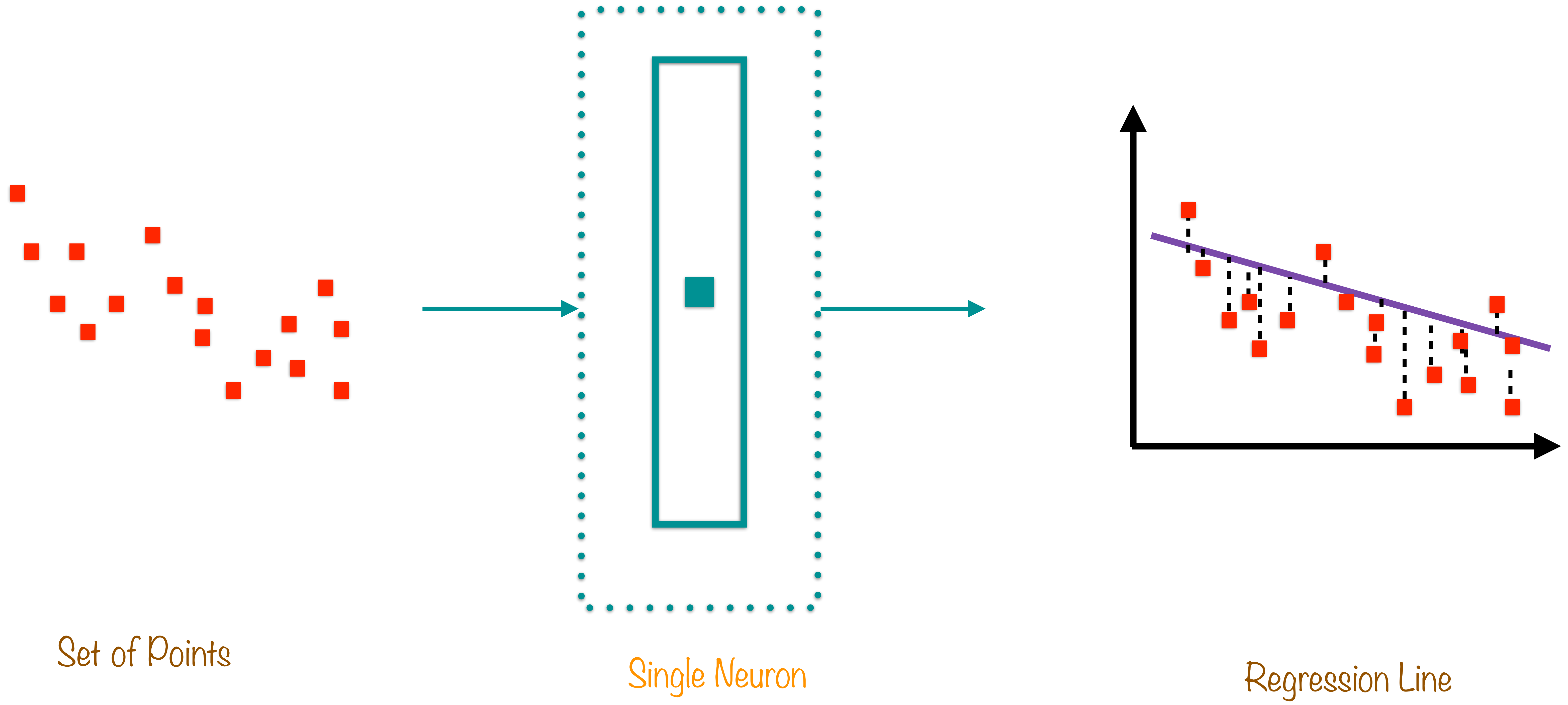
Operation of a Single Neuron



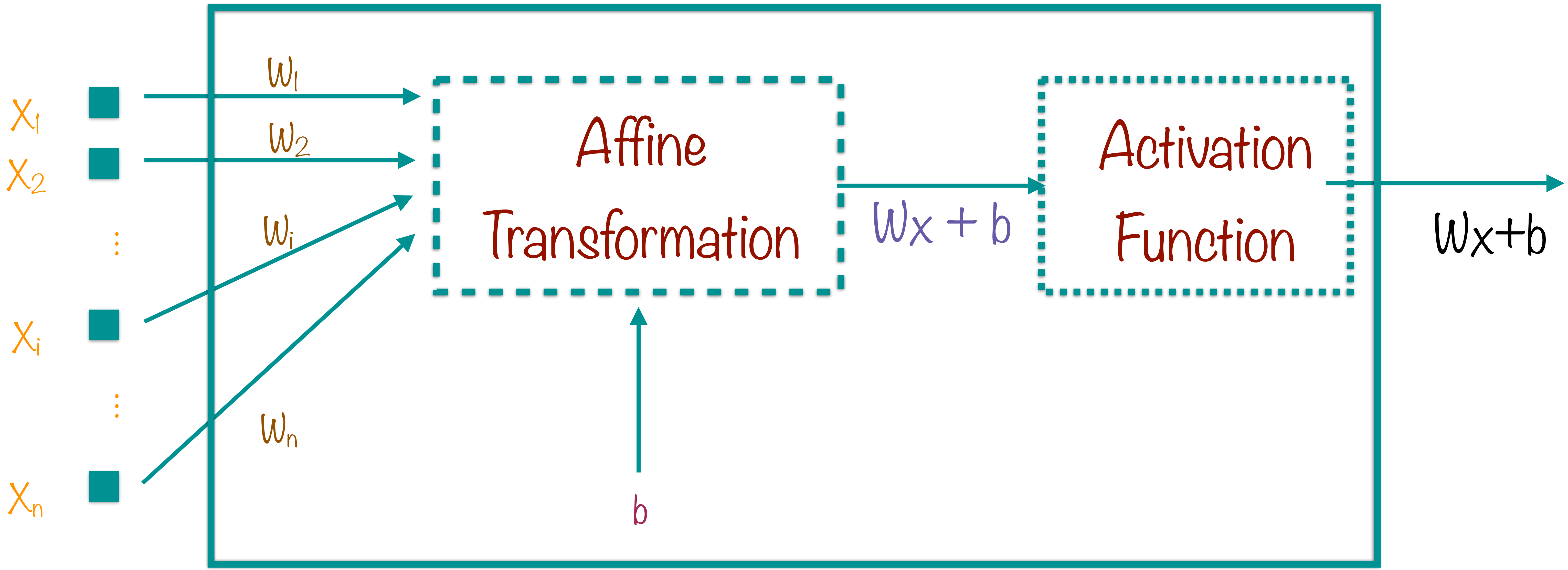
The combination of the affine transformation and the activation function can learn any arbitrary relationship

Activation Functions for Non-Linear Relationships

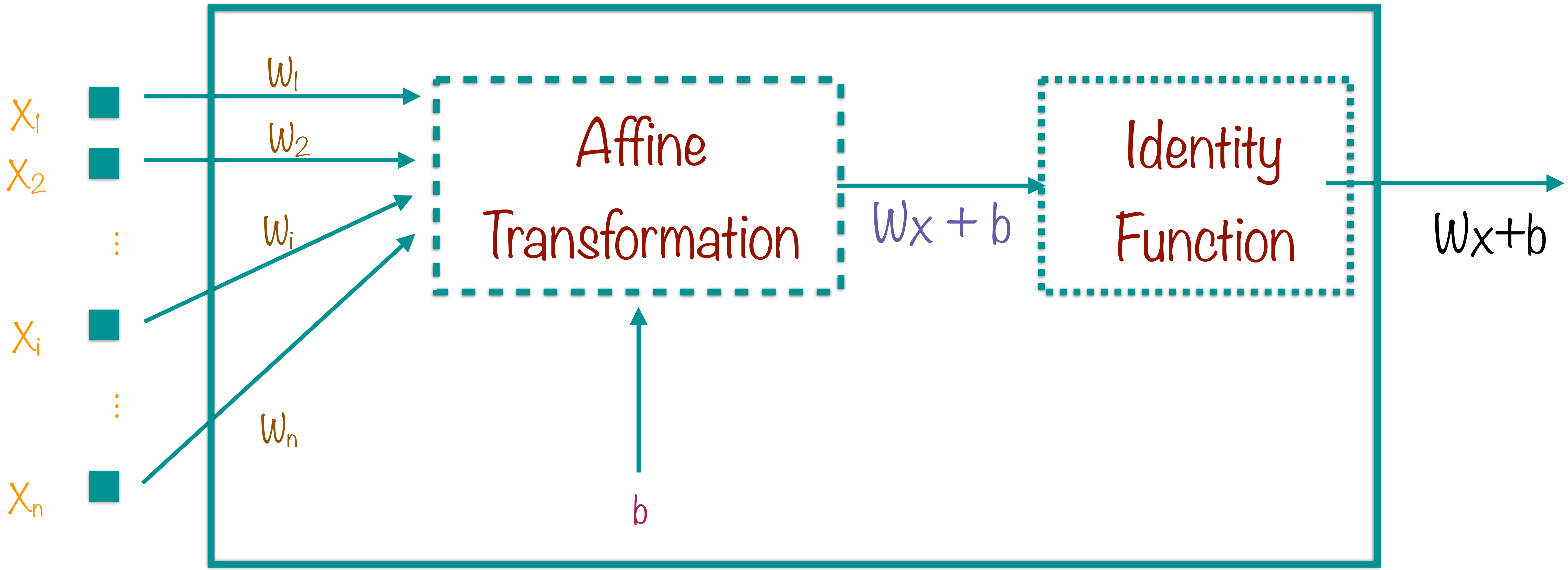
Regression: The Simplest Neural Network



Regression: The Simplest Neural Network

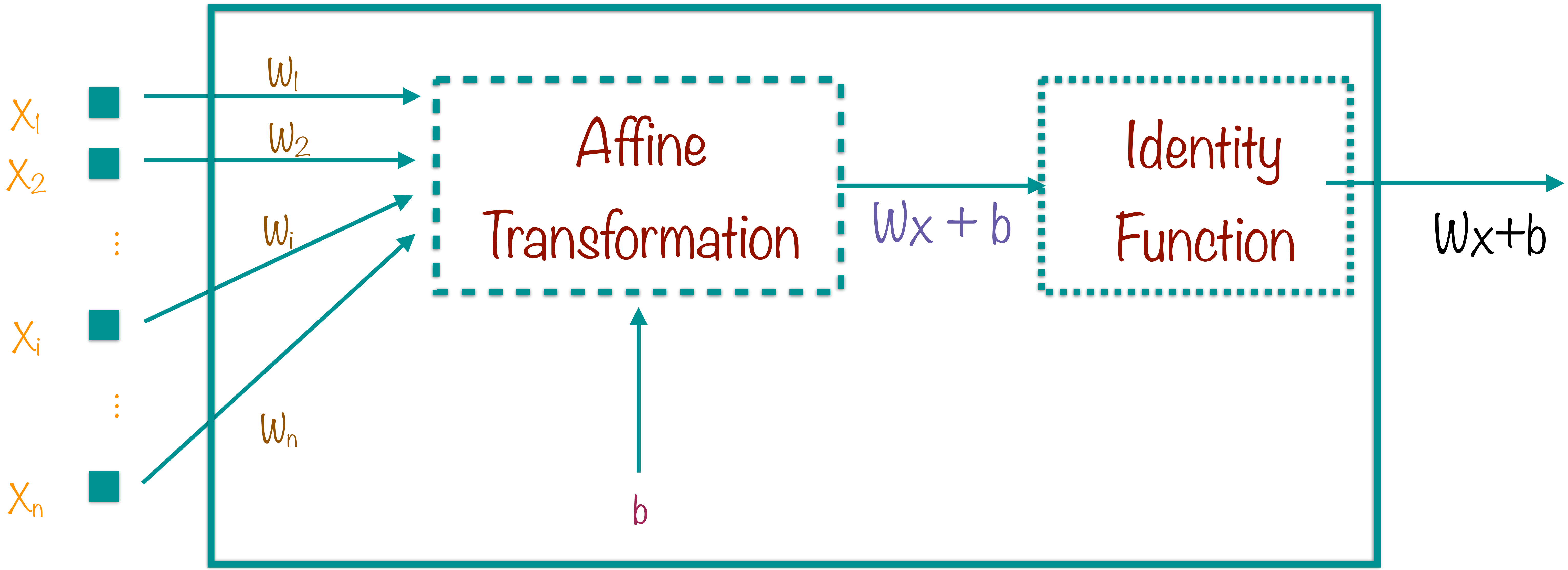


Regression: The Simplest Neural Network



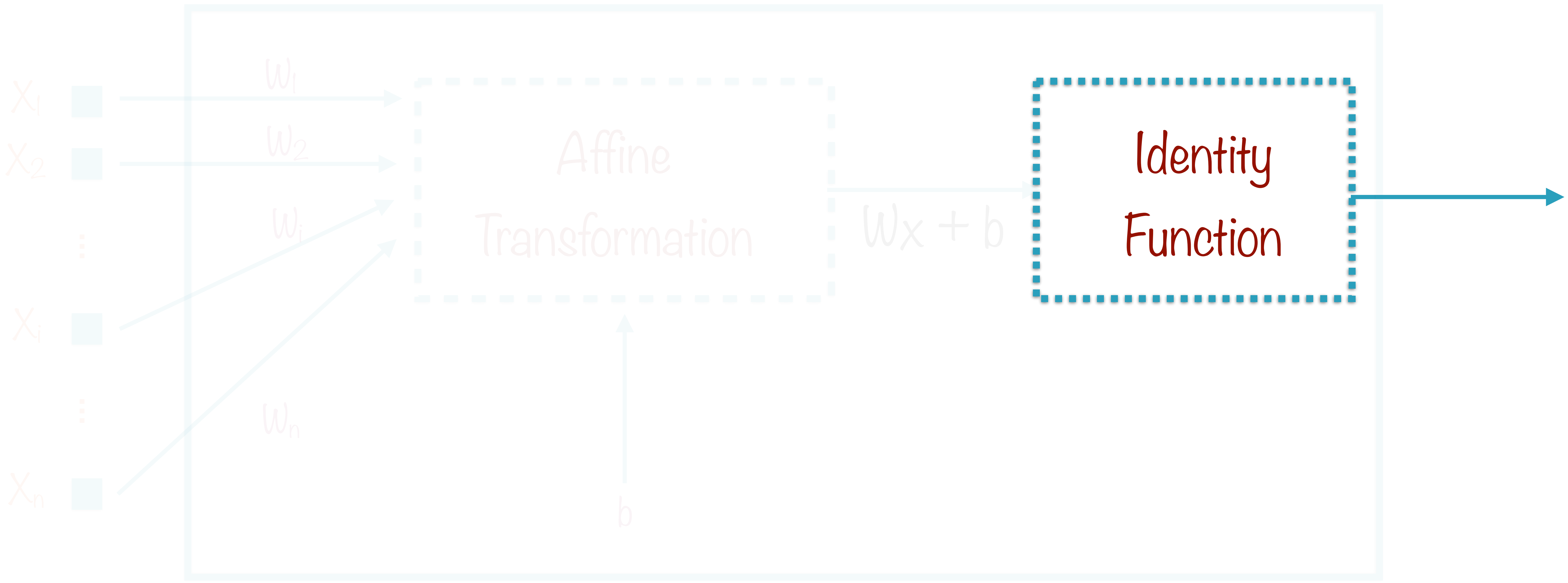
activation=None, simply passes the output of the linear affine transformation to the output of the neuron

Regression: The Simplest Neural Network



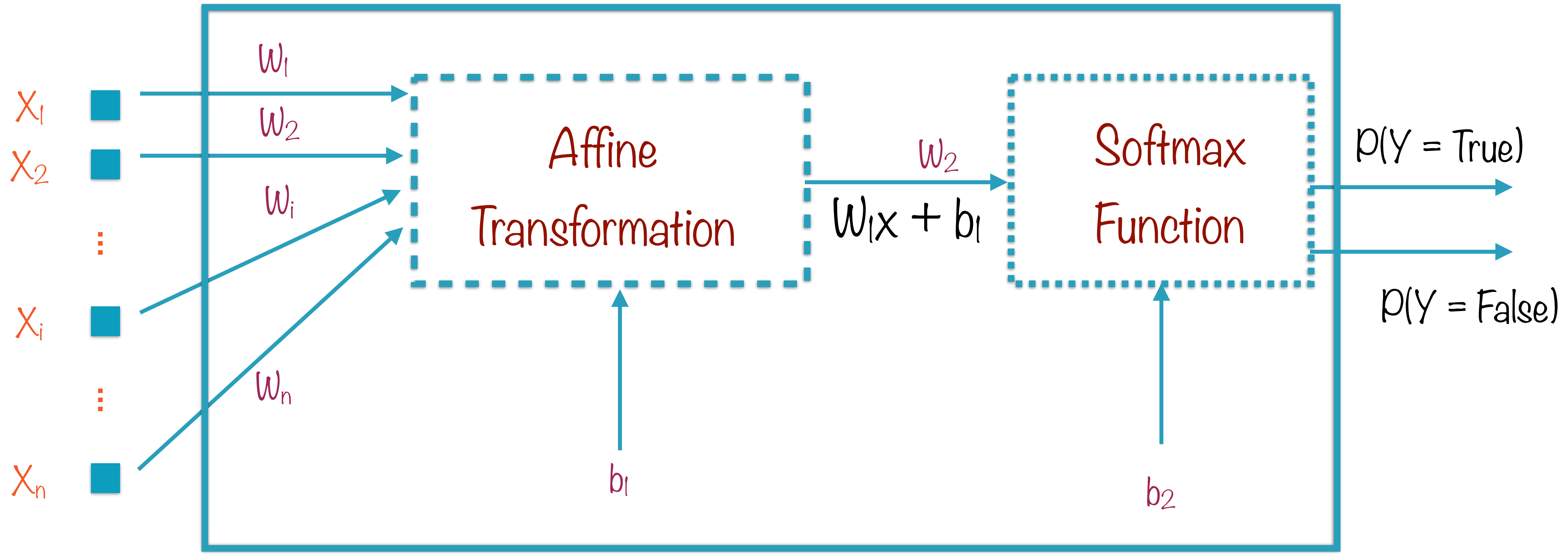
Also called a **linear neuron**

Linear Regression with One Neuron

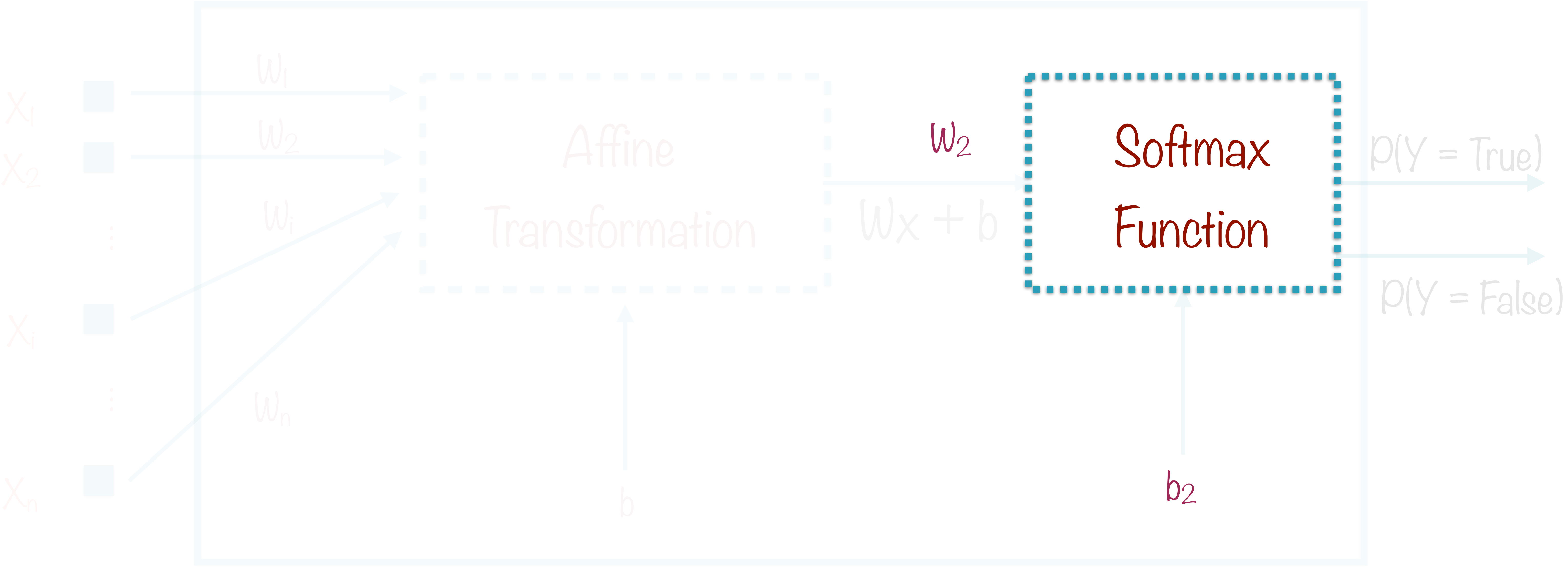


Logistic Regression with One Neuron

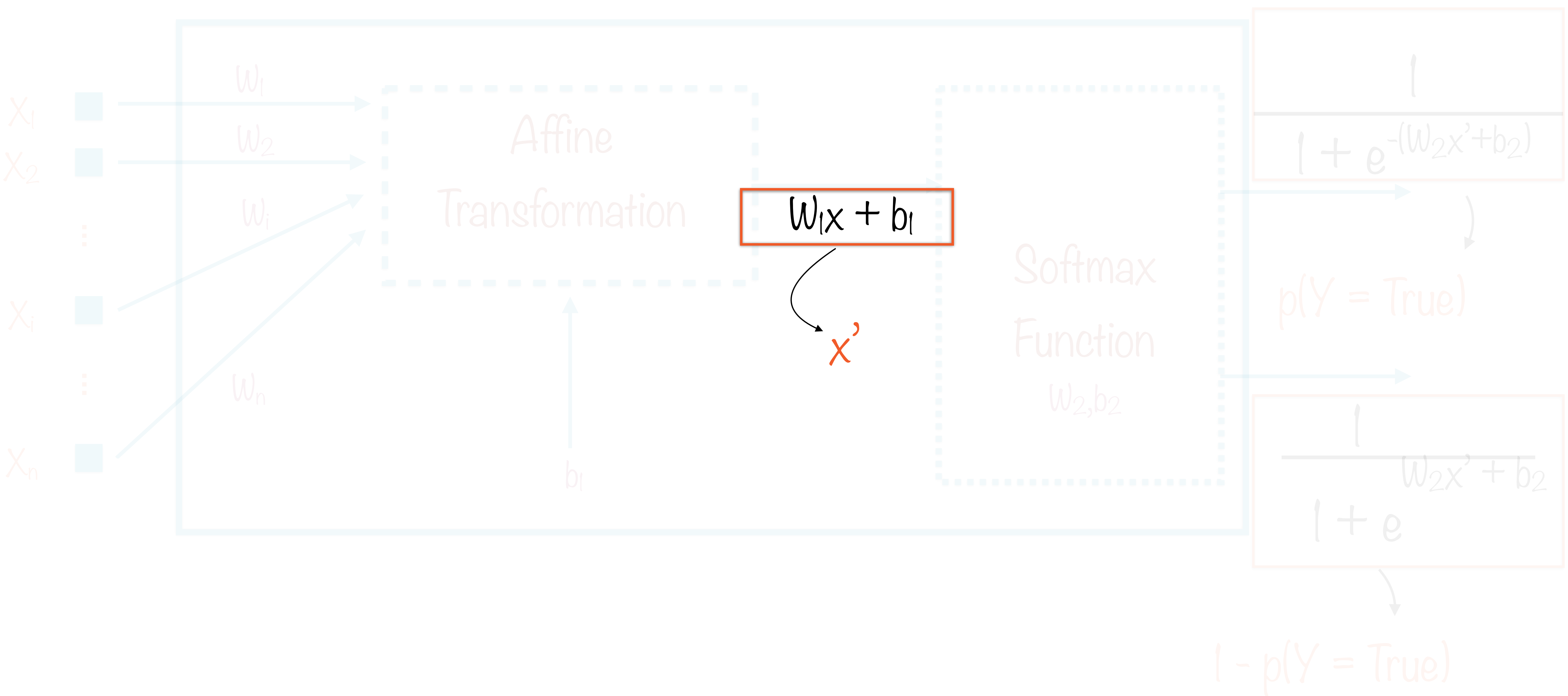
Logistic Regression with One Neuron



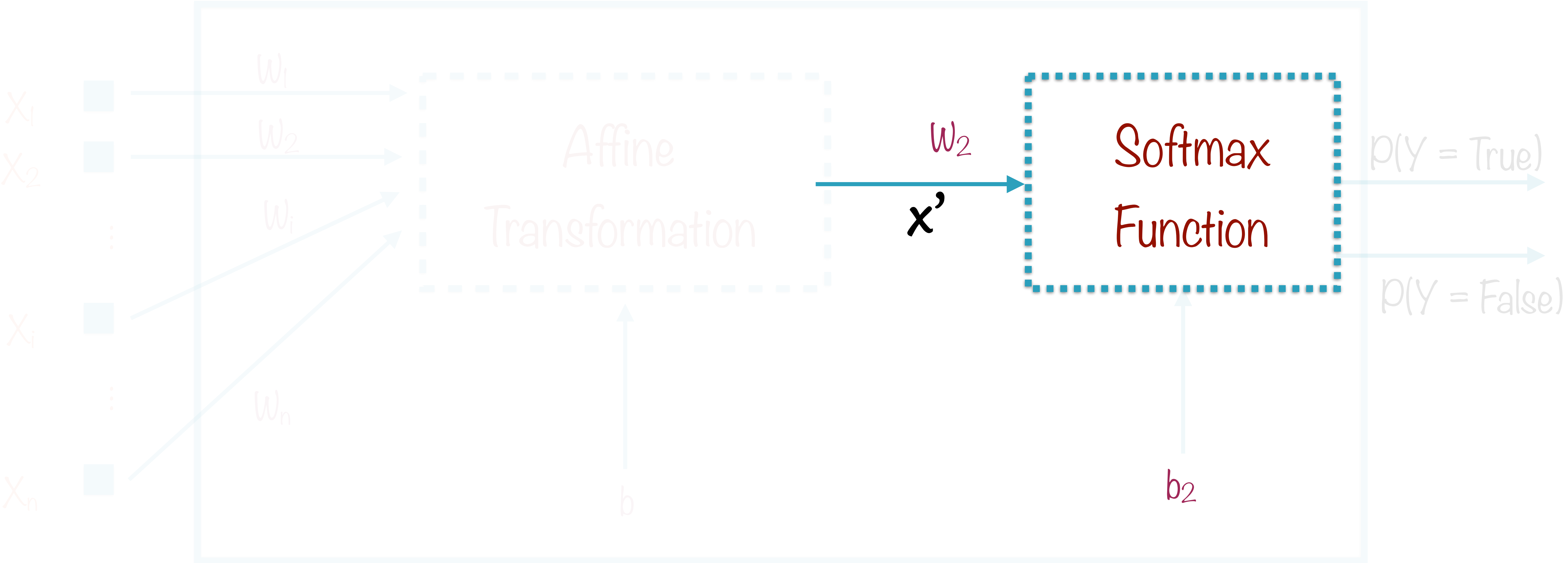
Logistic Regression with One Neuron



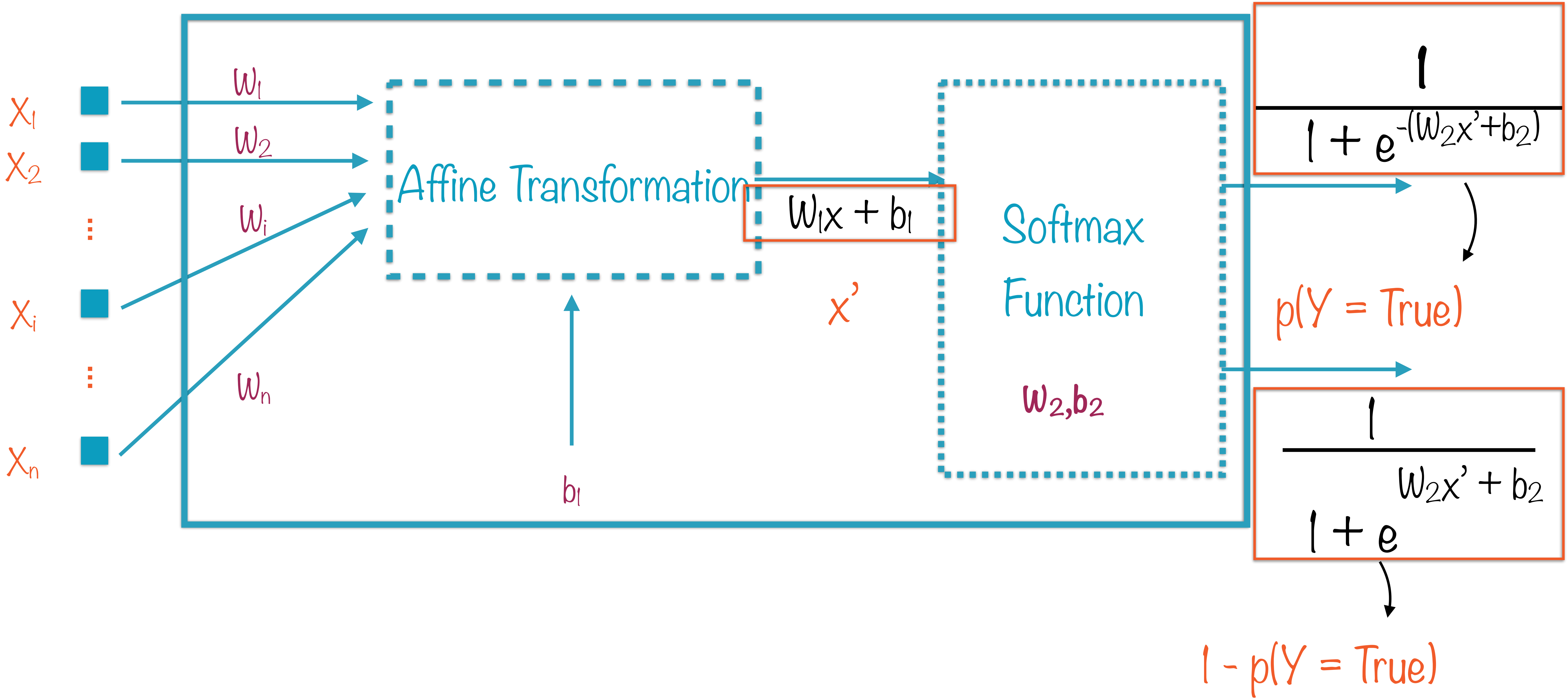
Logistic Regression with One Neuron



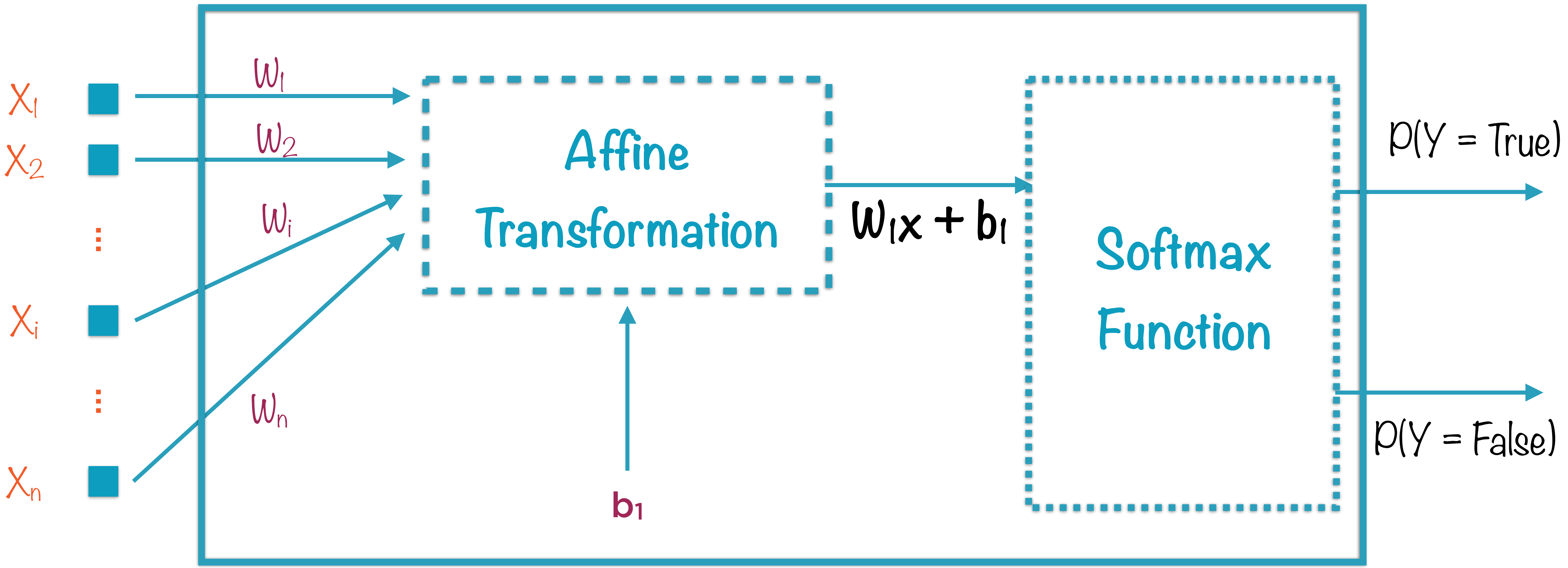
Logistic Regression with One Neuron



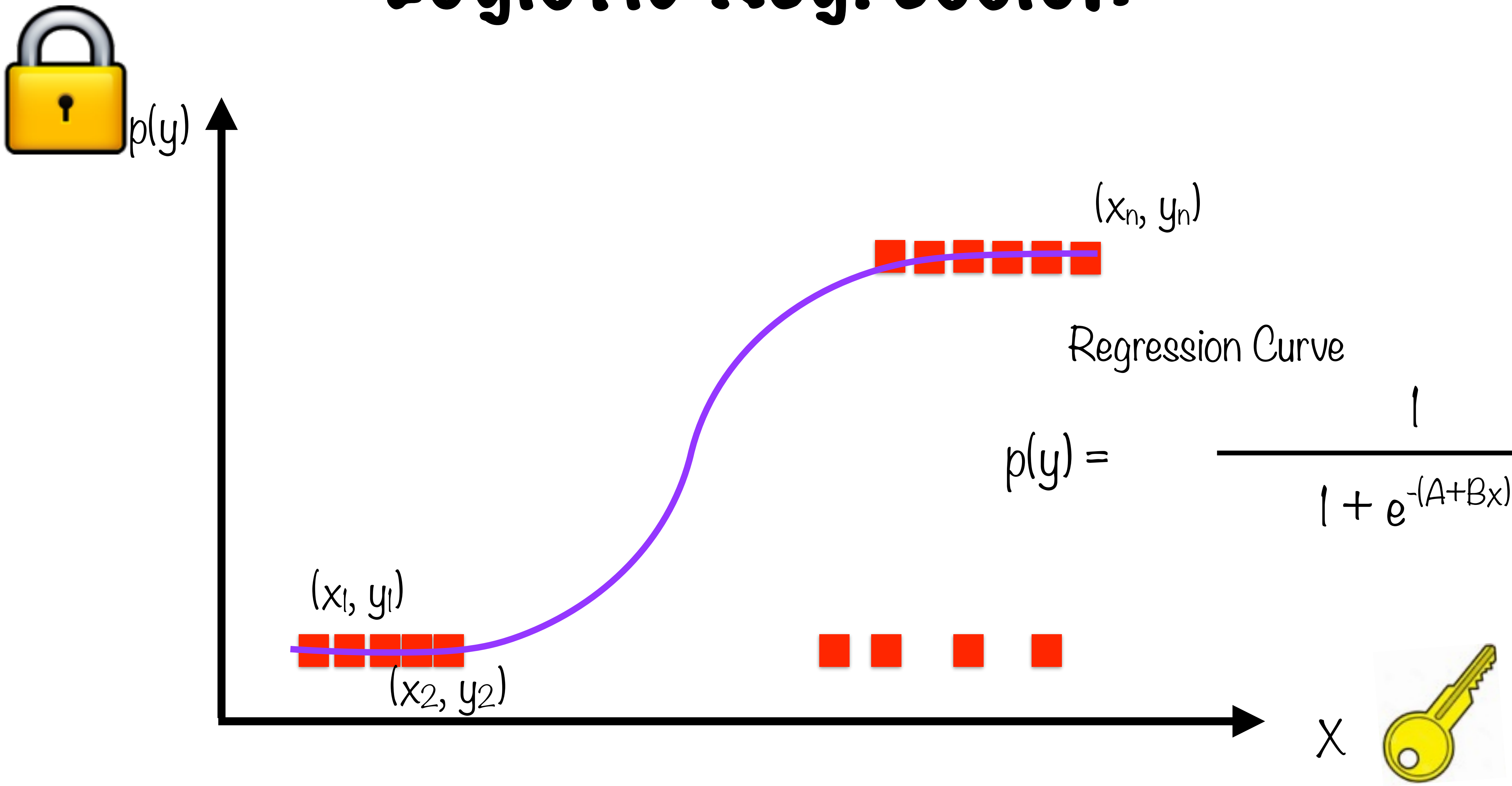
Logistic Regression with One Neuron



Logistic Regression with One Neuron

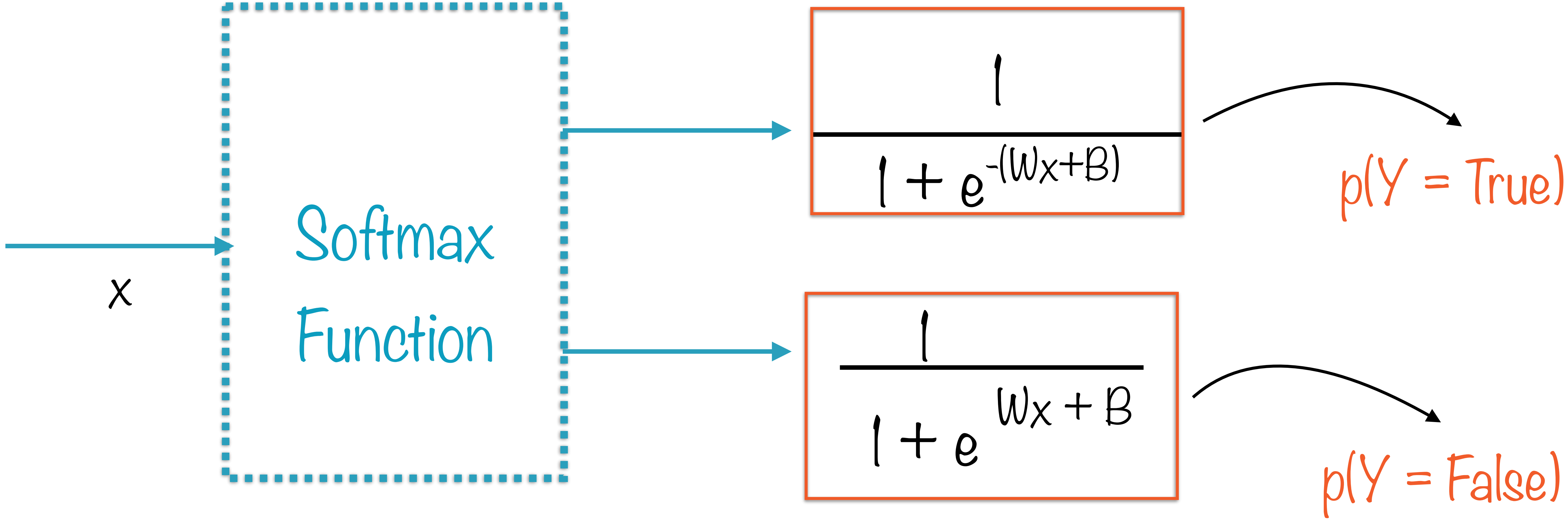


Logistic Regression

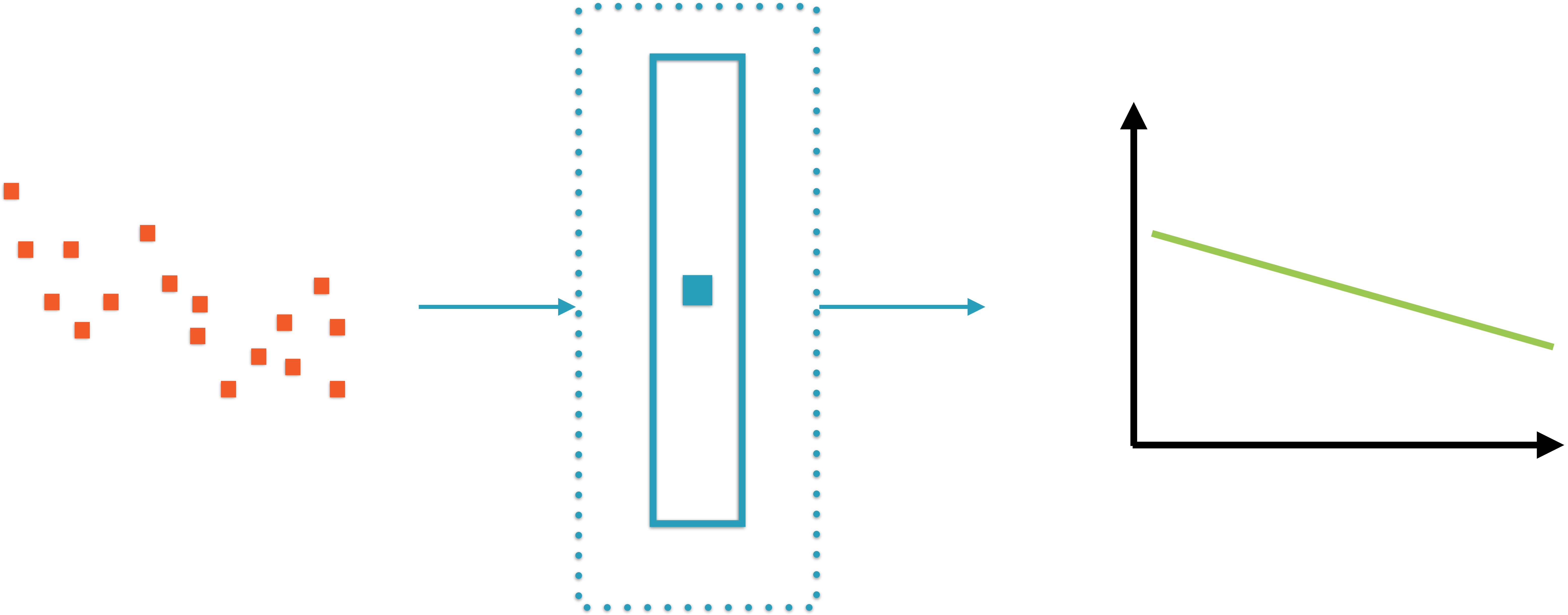


$$p(y) = \frac{1}{1 + e^{-(A+Bx)}}$$

SoftMax for True/False Classification



Linear Regression with One Neuron



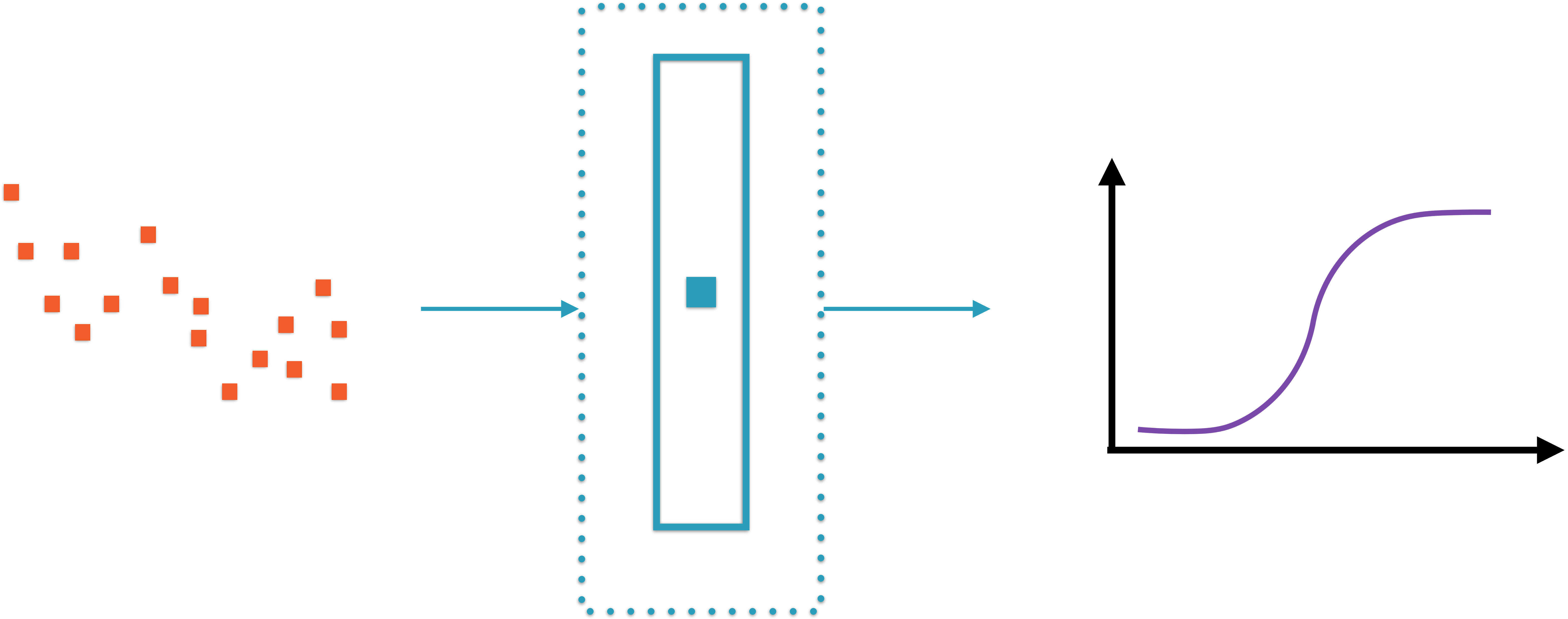
1-dimensional feature vector

Shape (W) = [1,1]

Shape (b) = [1]

Regression Line

Logistic Regression with One Neuron



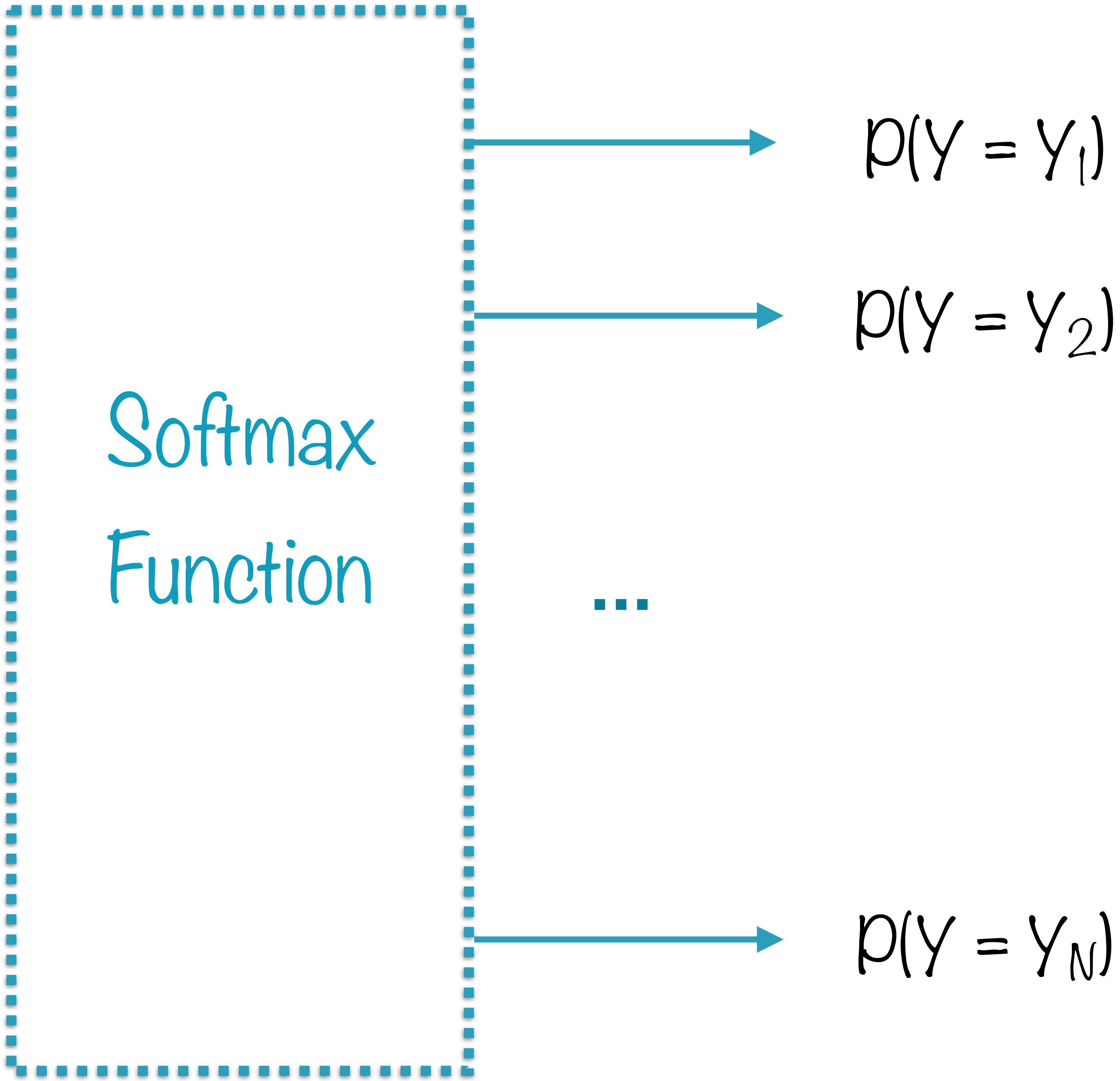
1-dimensional feature vector

Shape (W) = [1,2]

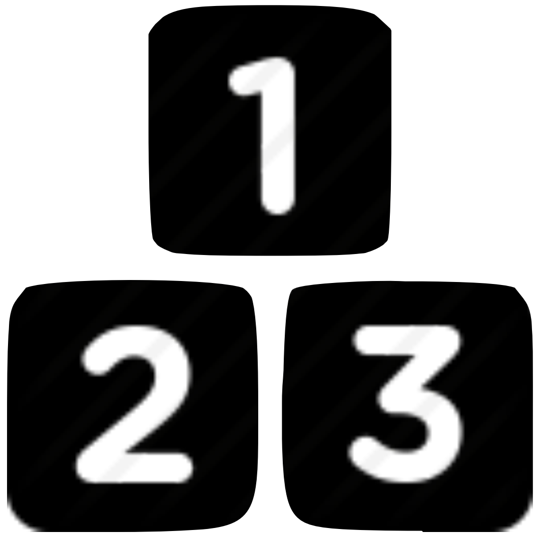
Shape (b) = [2]

S-Curve

SoftMax N-category Classification



Multilabel Digit Classification



One-versus-all: Train 10 binary classifiers

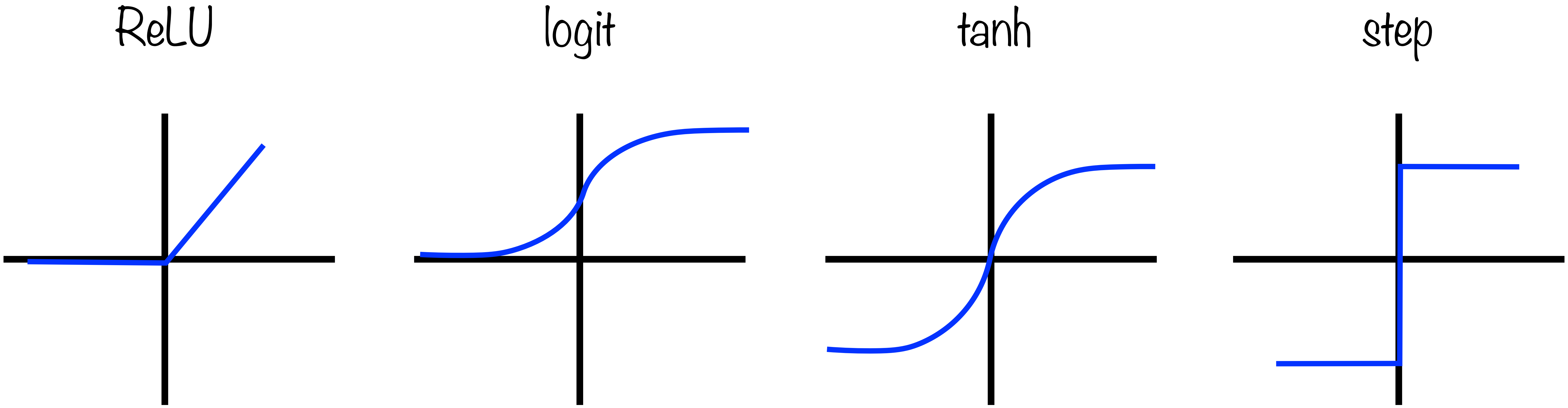
- 0-detector, 1-detector...
- Predicted label = output of detector with highest score

One-versus-one: Train 45 binary classifiers

- One detector for each pair of digits
- For N labels, need N(N-1)/2 classifiers
- Predicted label = output of digit that wins most duels

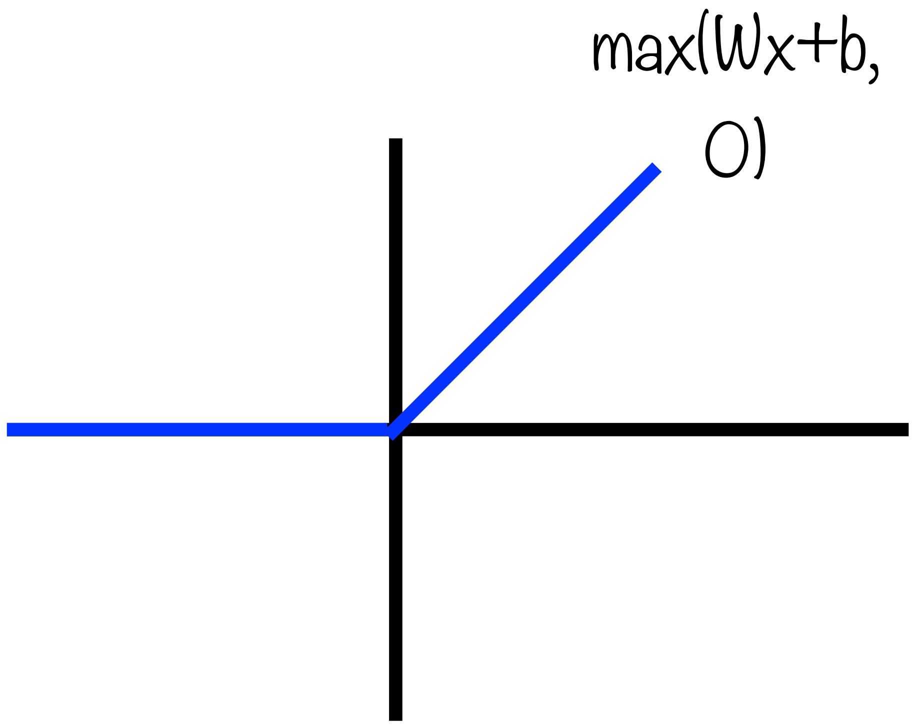
The logistic or softmax function is just one of many that can be used for activation

Activation Function



Various choices of activation functions exist and drive the design of your neural network

ReLU Activation

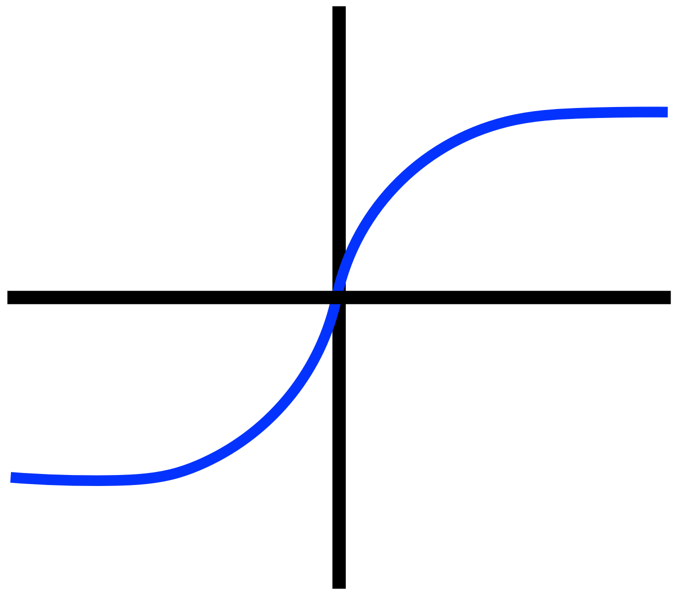


The most common form of the activation function is the ReLU

ReLU : Rectified Linear Unit

$ReLU(x) = max(0,x)$

Tanh Activation

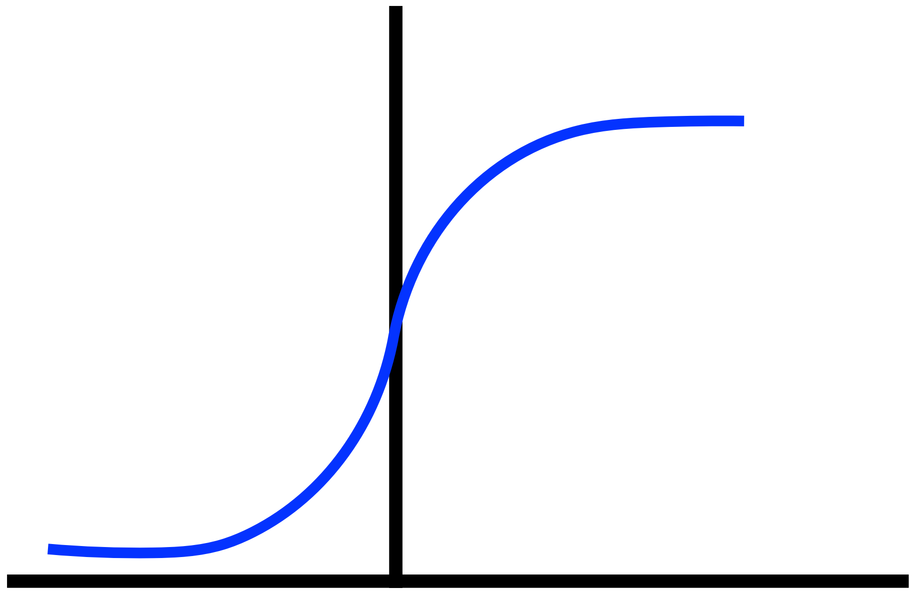


S-shaped, continuous and differentiable

Output ranges from -1 to 1

Makes each layer's output normalized (centered around 0)

SoftMax Activation



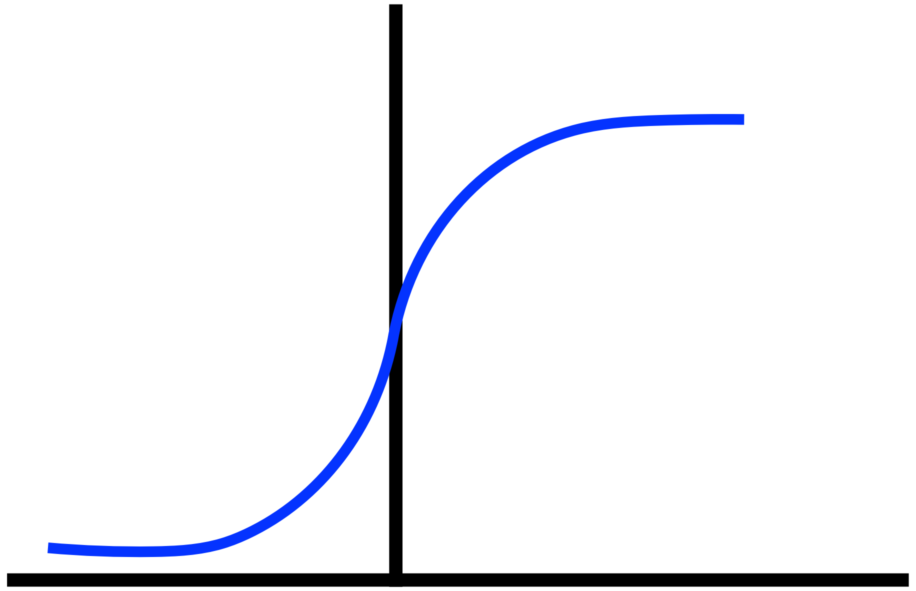
Another very common form of the activation function is the SoftMax

SoftMax(x) outputs a number between 0 and 1

This output can be interpreted as a probability

This curve is also called a logit curve

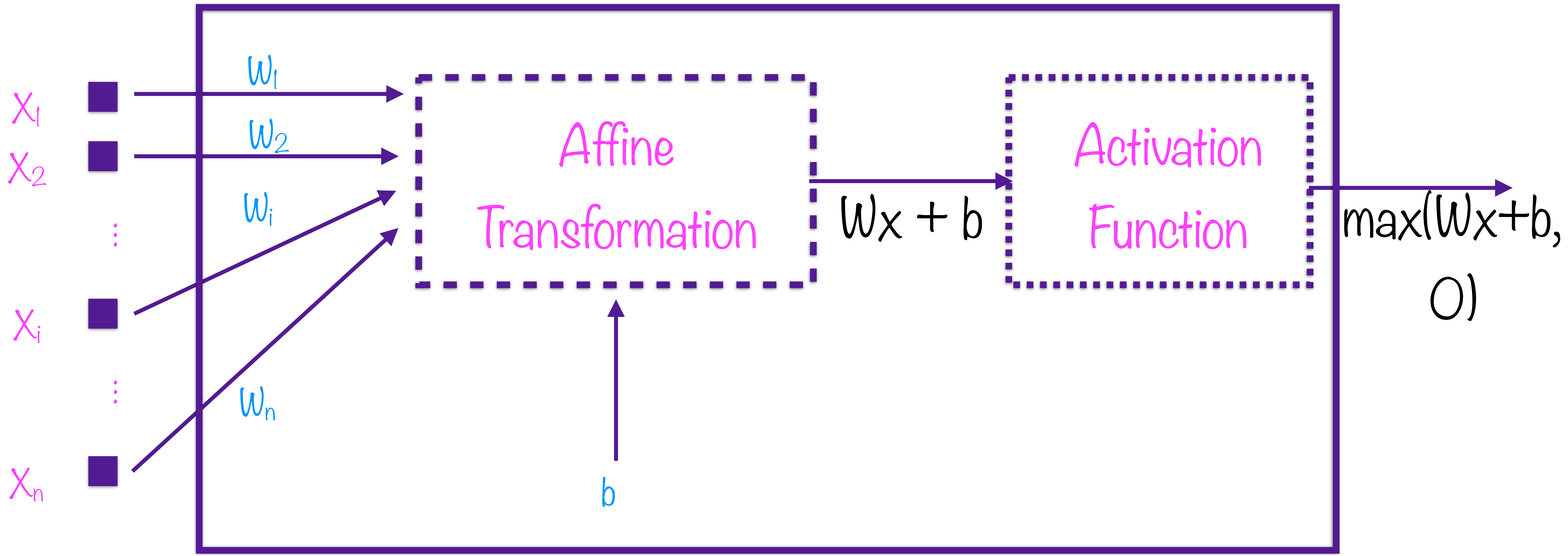
Importance of Activation



The choice of activation function is crucial in determining performance

To see why, we must understand the training process of a Neural Network

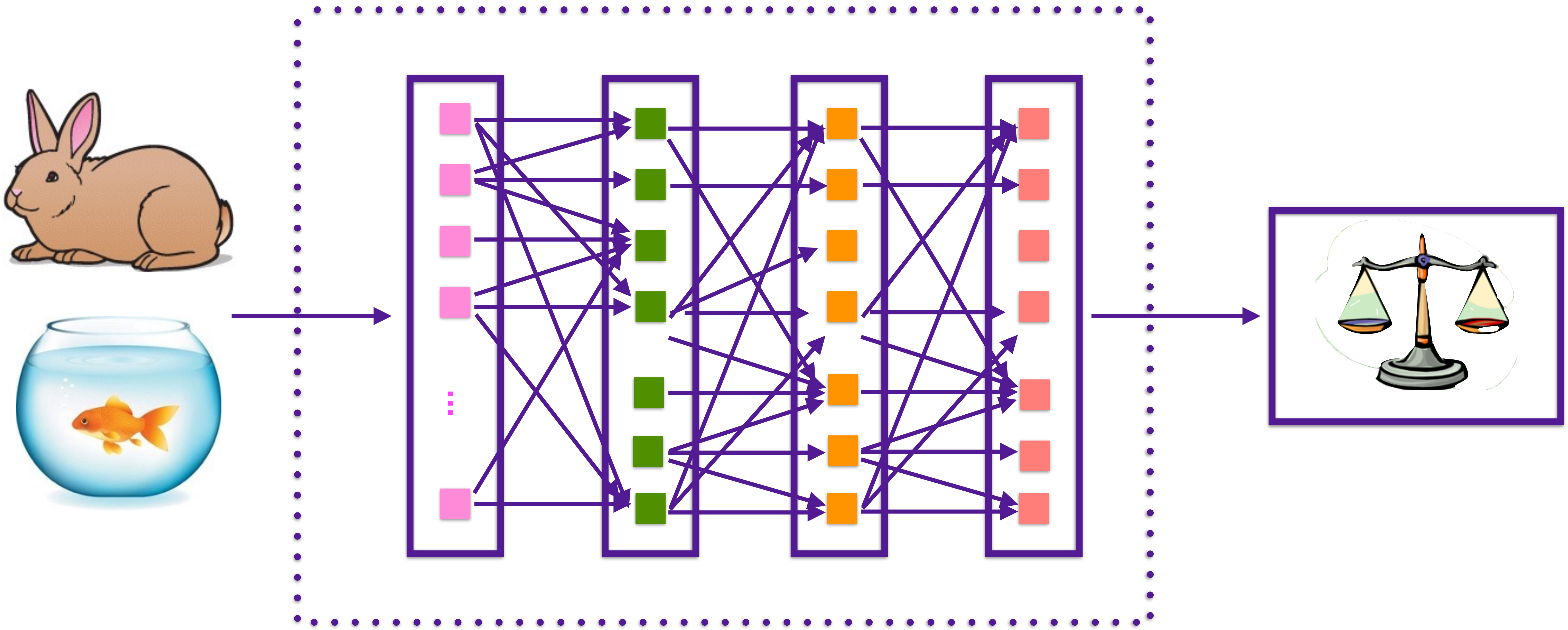
Neuron as a Learning Unit



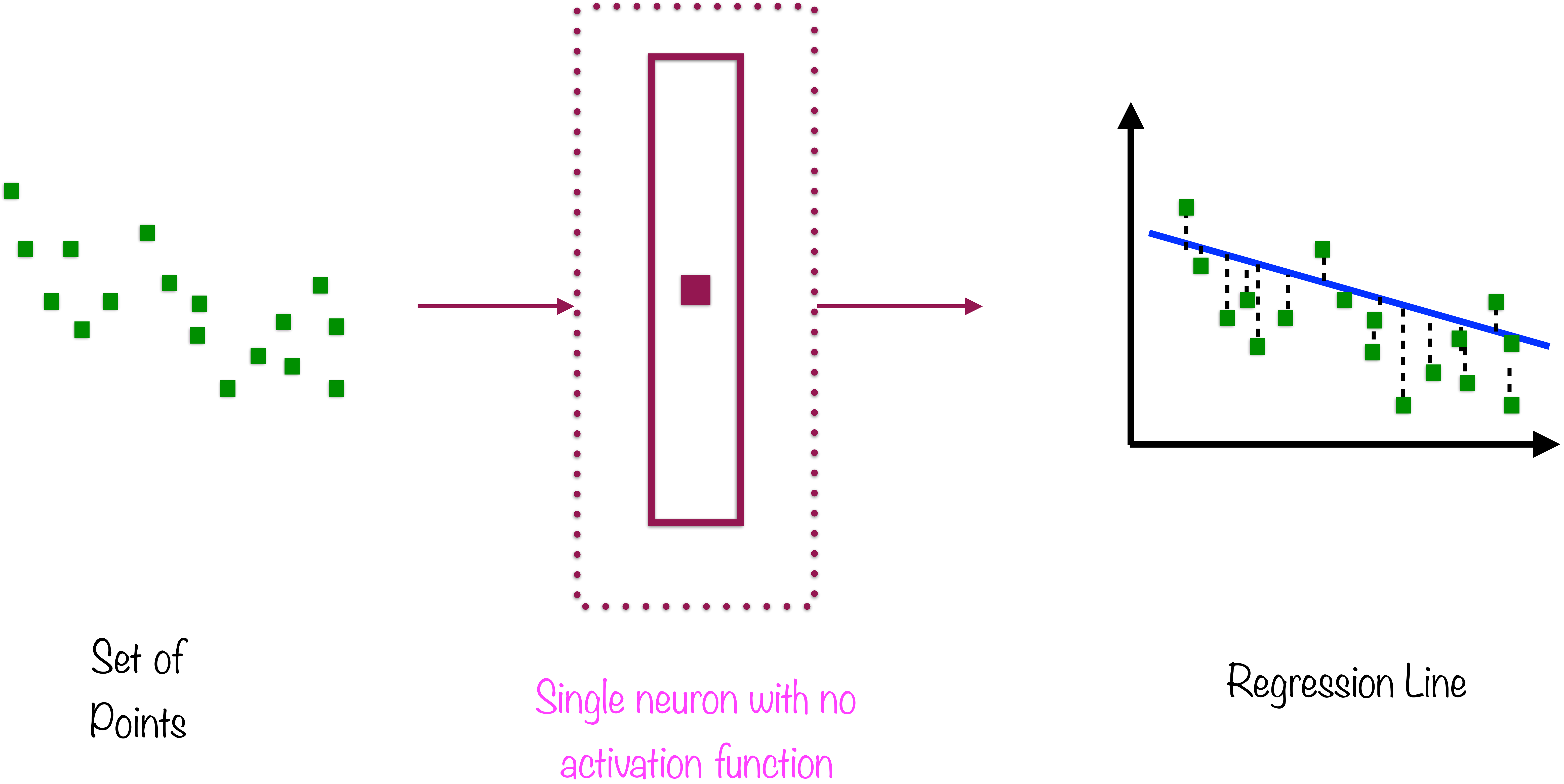
Many of these simple neurons arranged in layers can do magical stuff

Training a Neural Network: Optimization and Back Propagation

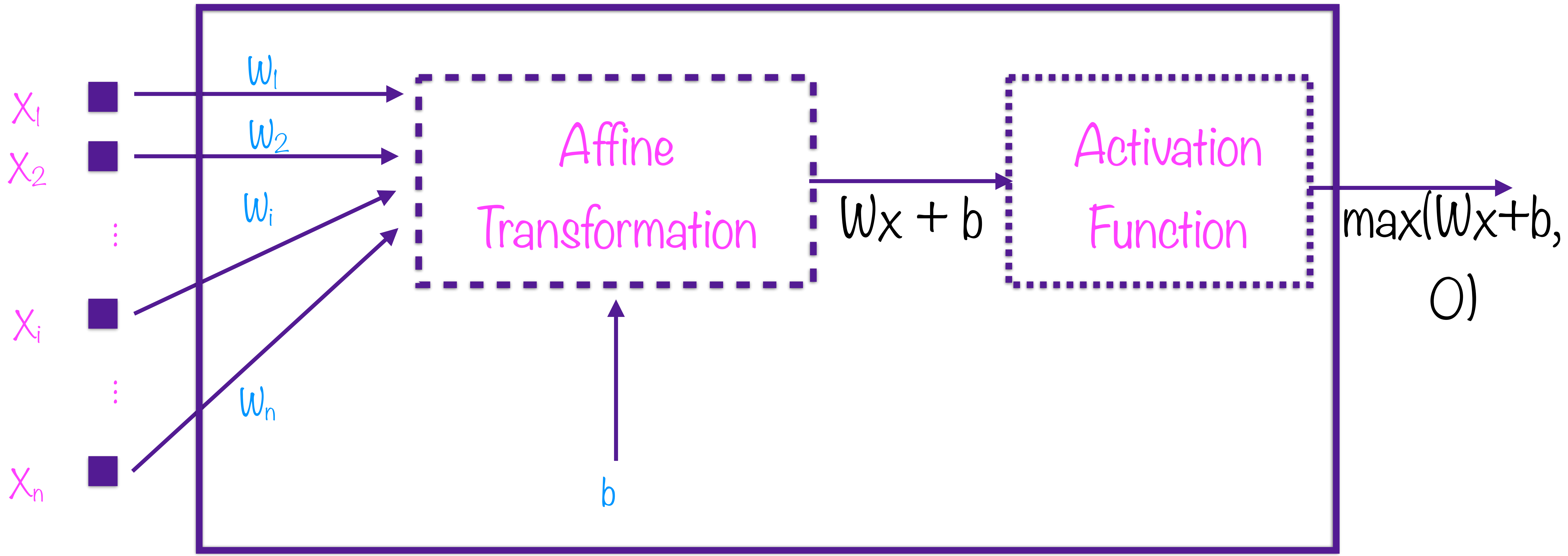
A Neural Network



Example: Training for Linear Regression



Example: Training for Linear Regression



The activation function to learn linear regression is simply the identity function

Training as an Optimization Problem

Objective Function

Minimize variance of the residuals (MSE)

Constraints

Express relationship as a straight line

$y = Wx + b$

Decision Variables

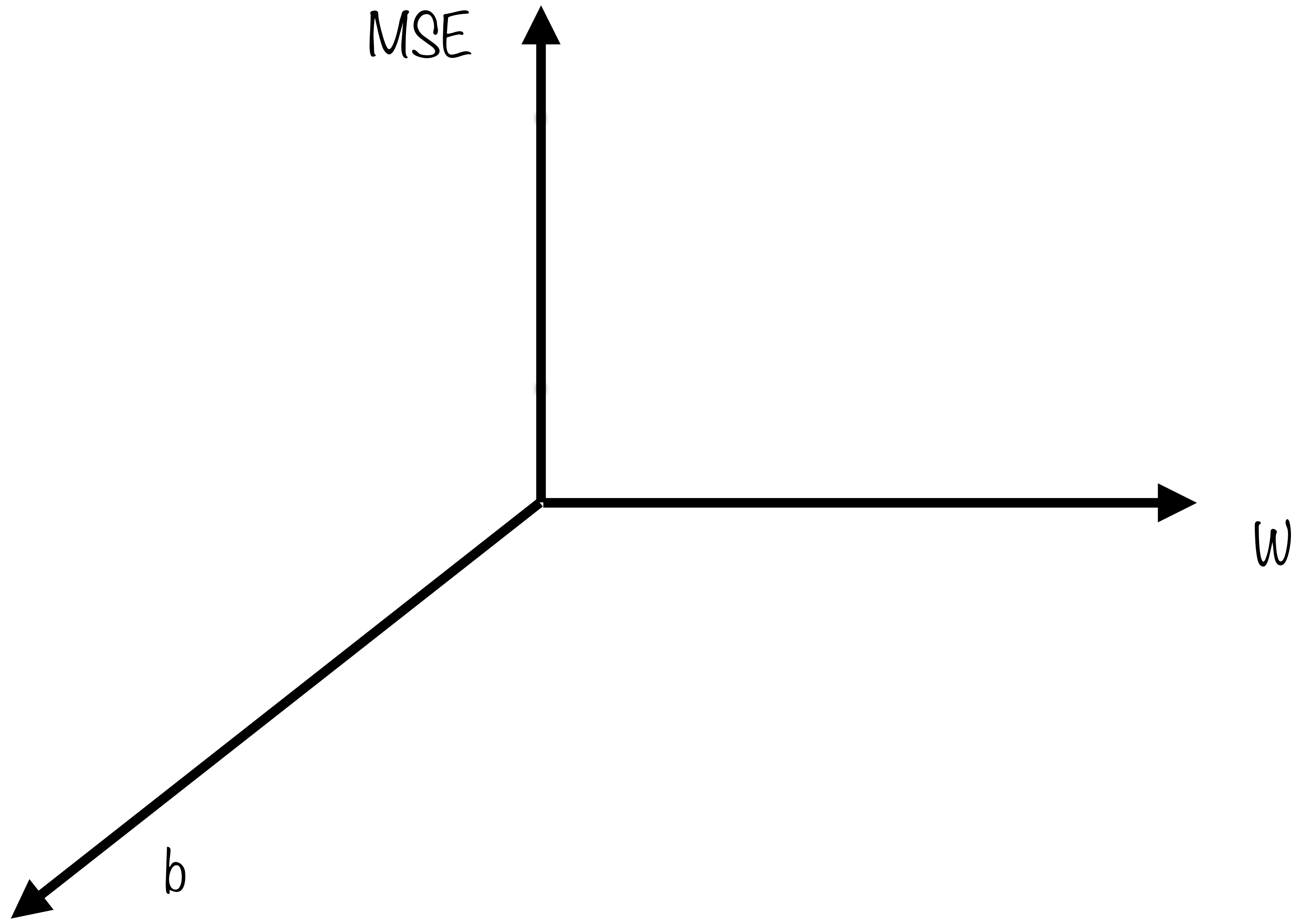
Values of W and b

The "Best" Regression Line

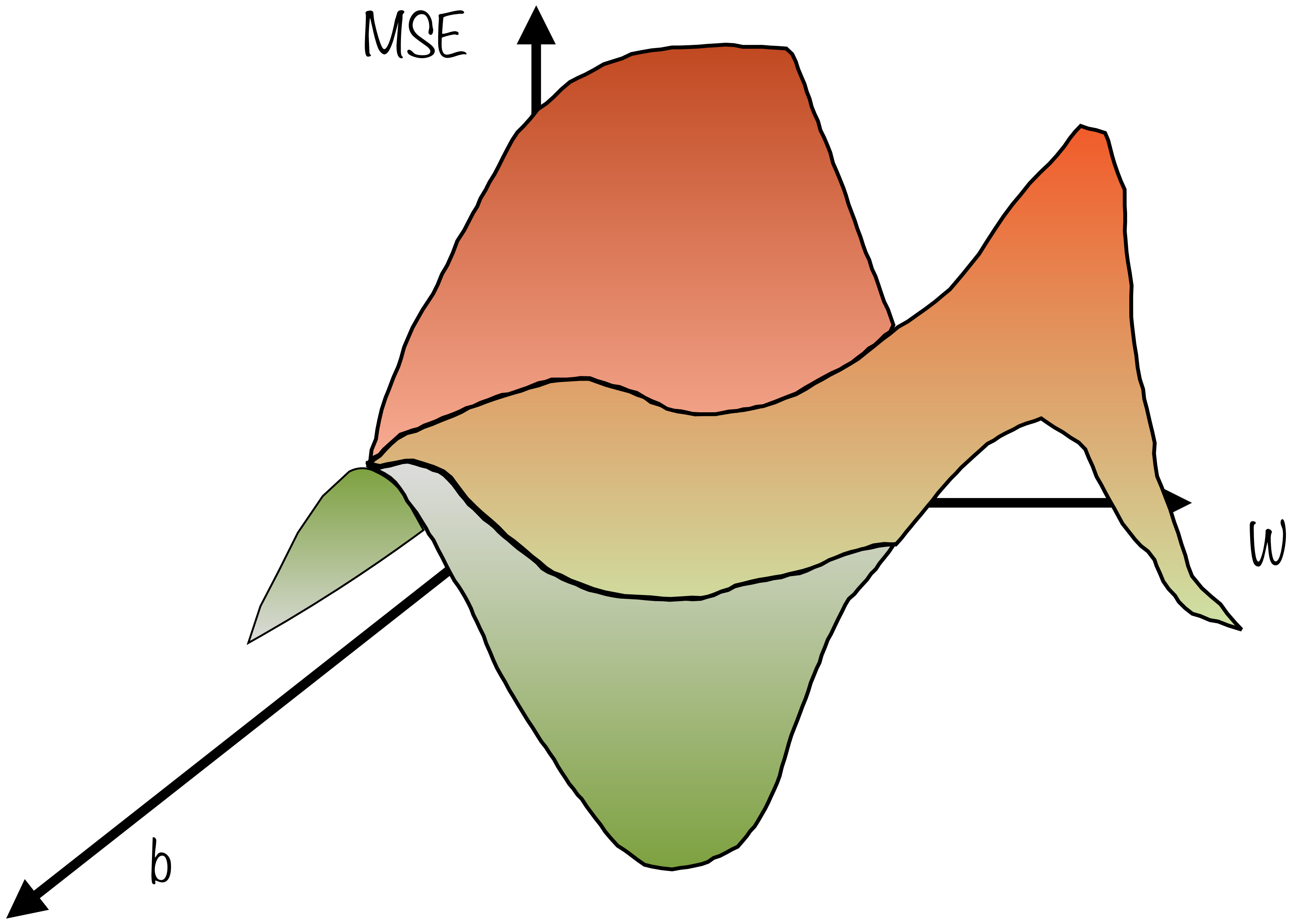
The "best fit" line is called the regression line

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

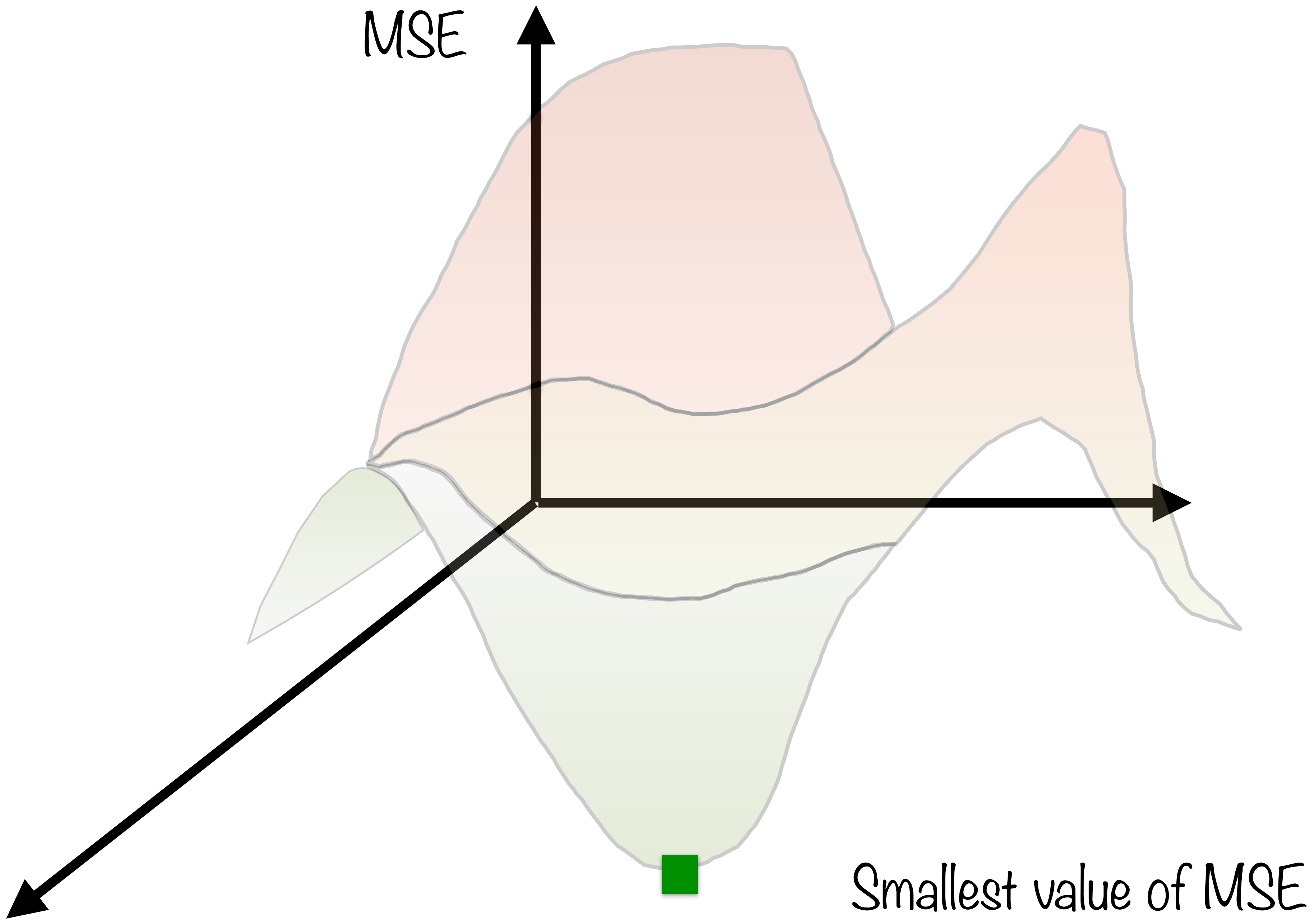
Minimizing MSE



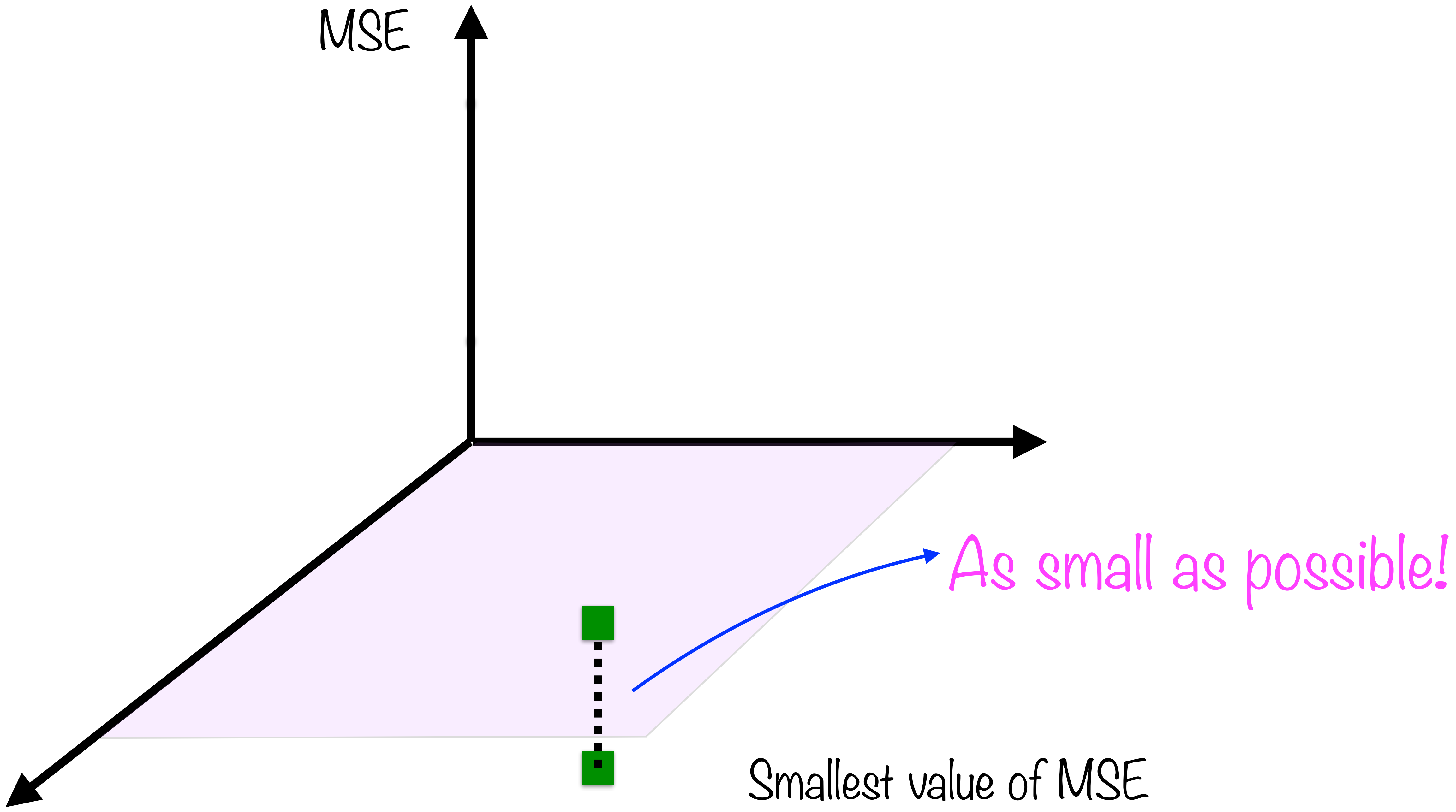
Minimizing MSE



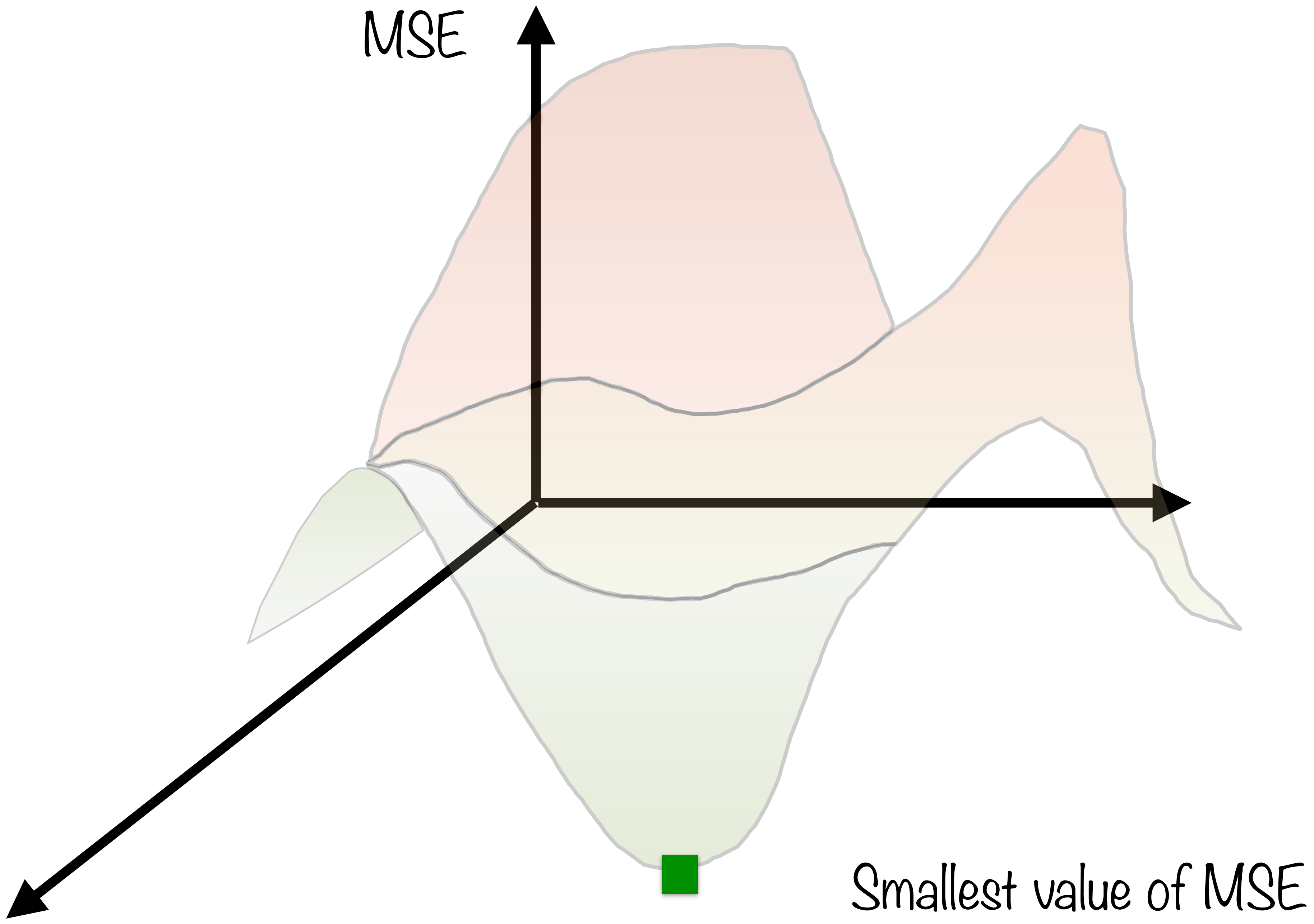
Minimizing MSE



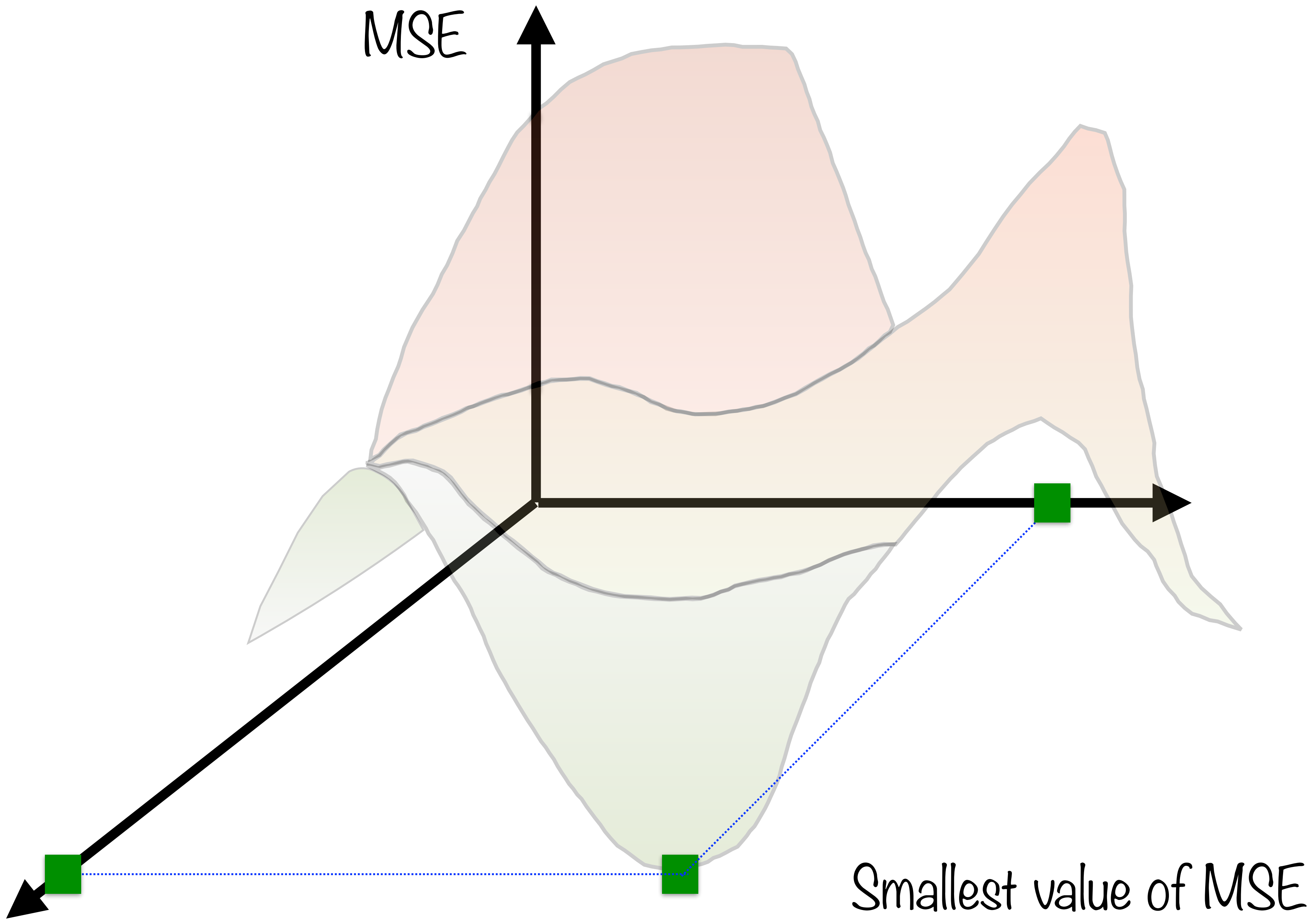
Minimizing MSE



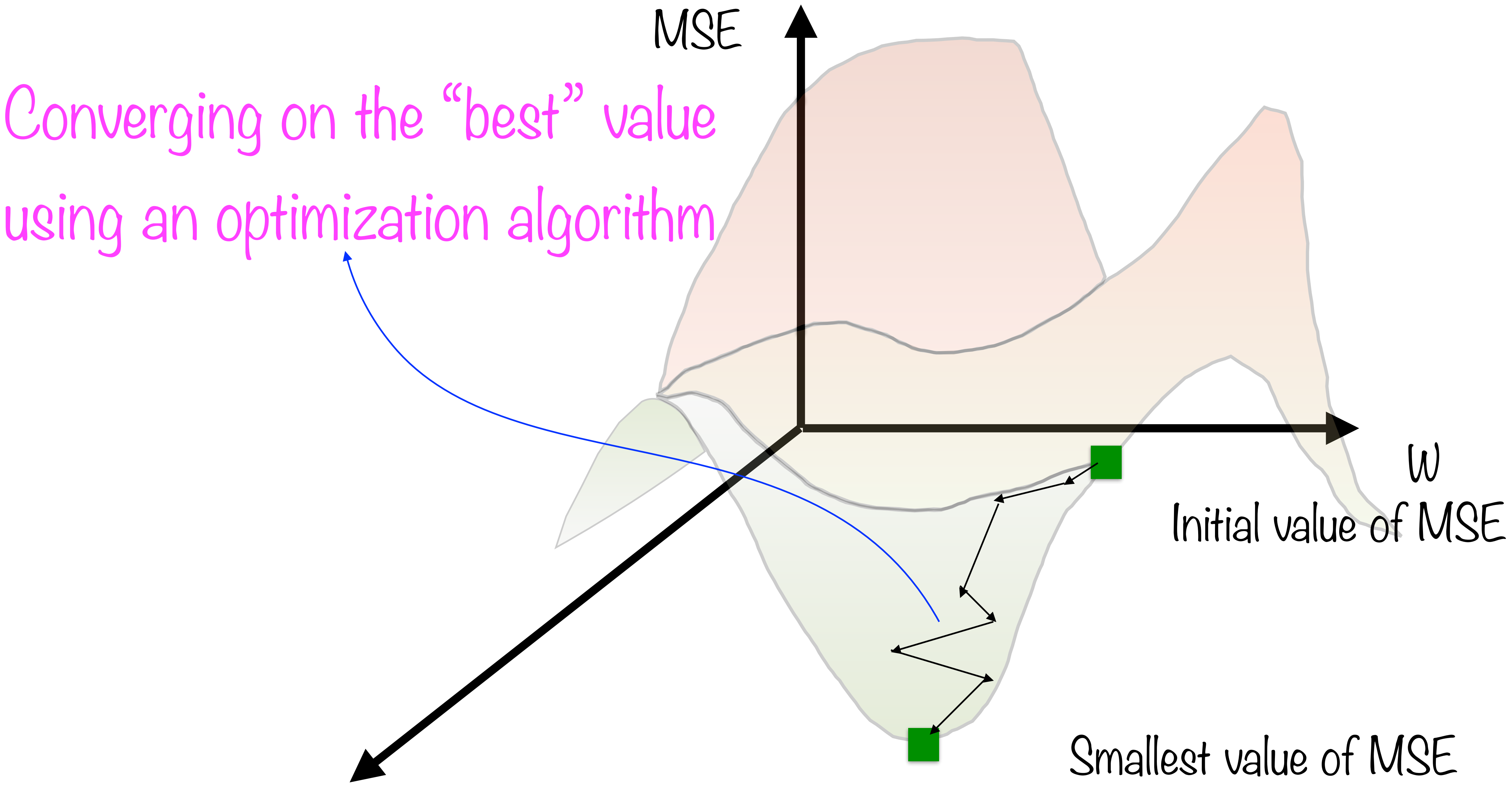
Minimizing MSE



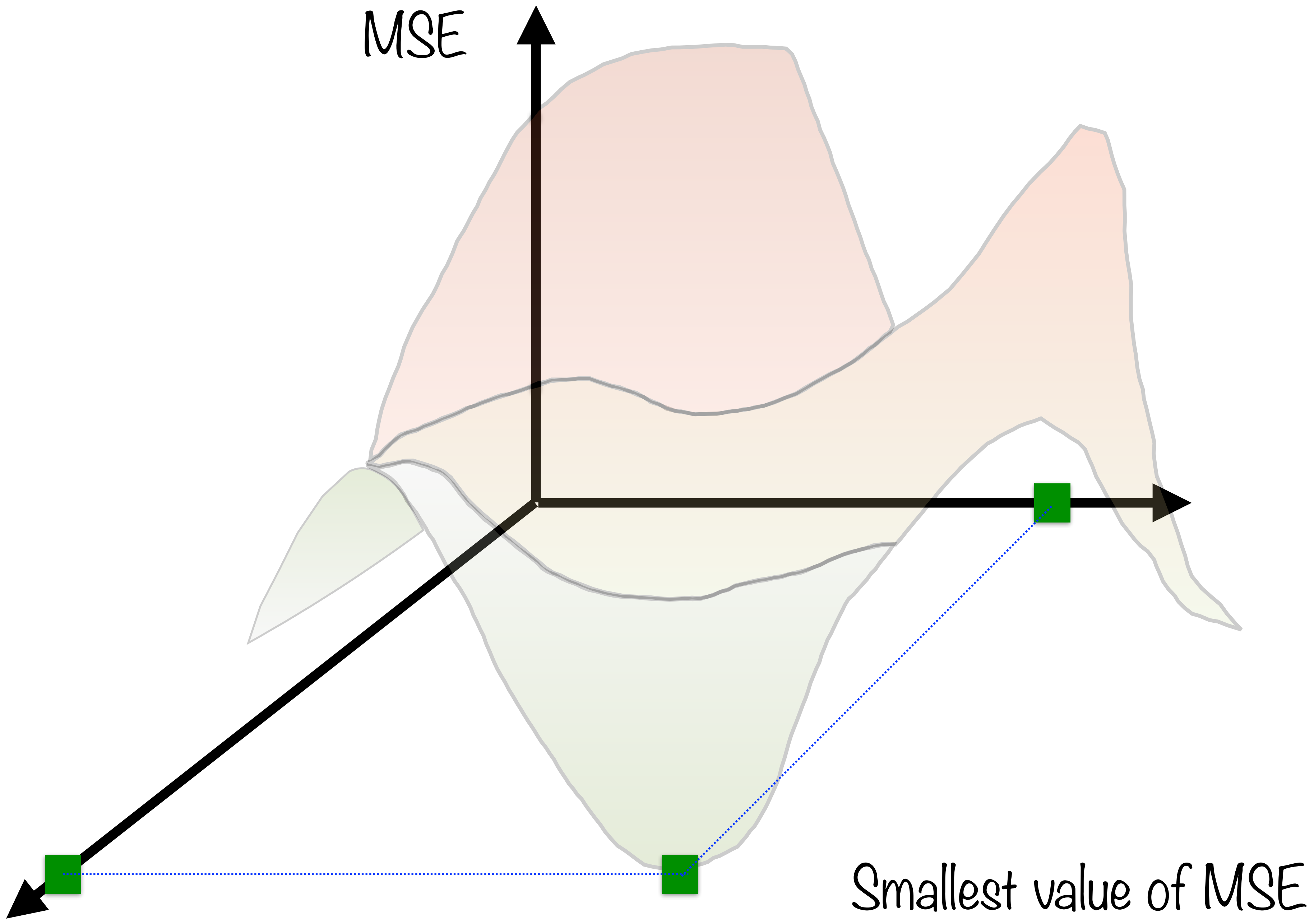
Minimizing MSE



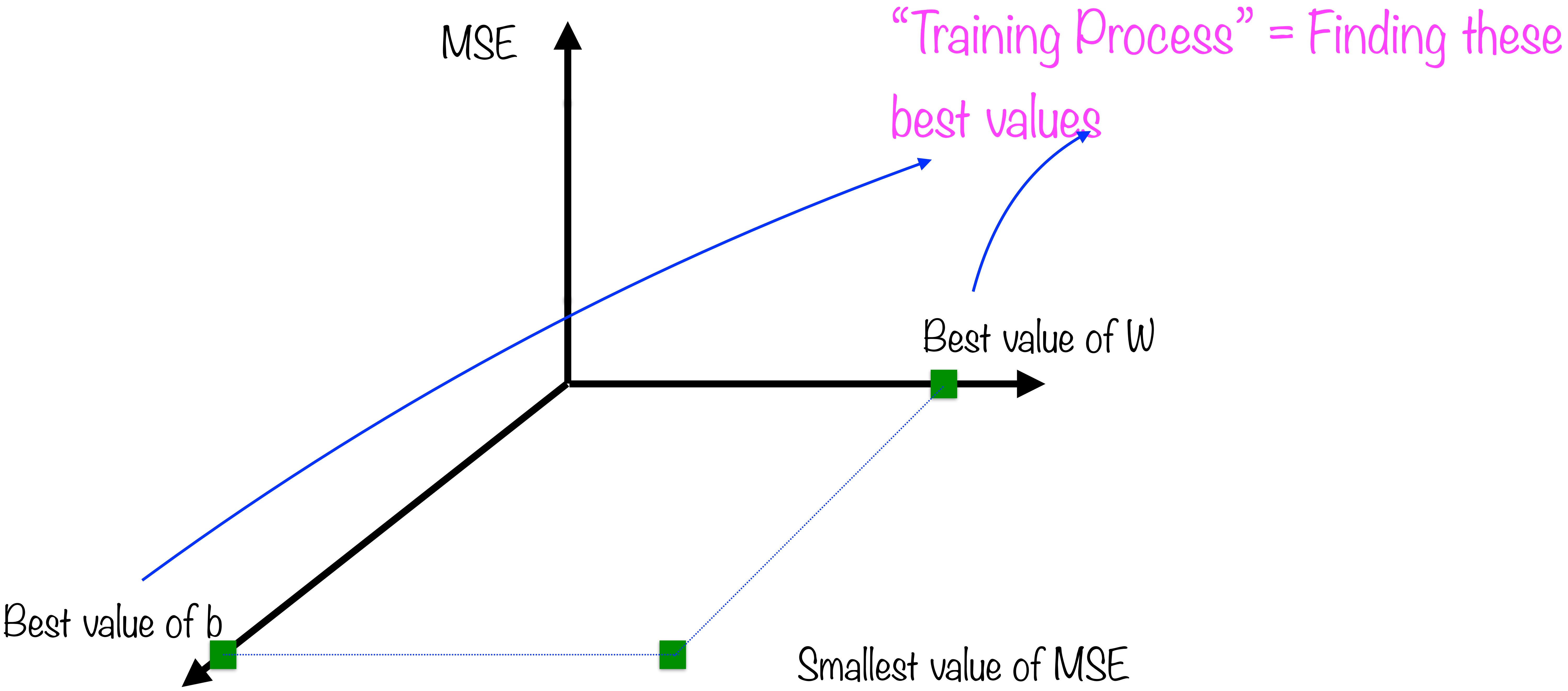
"Gradient Descent"



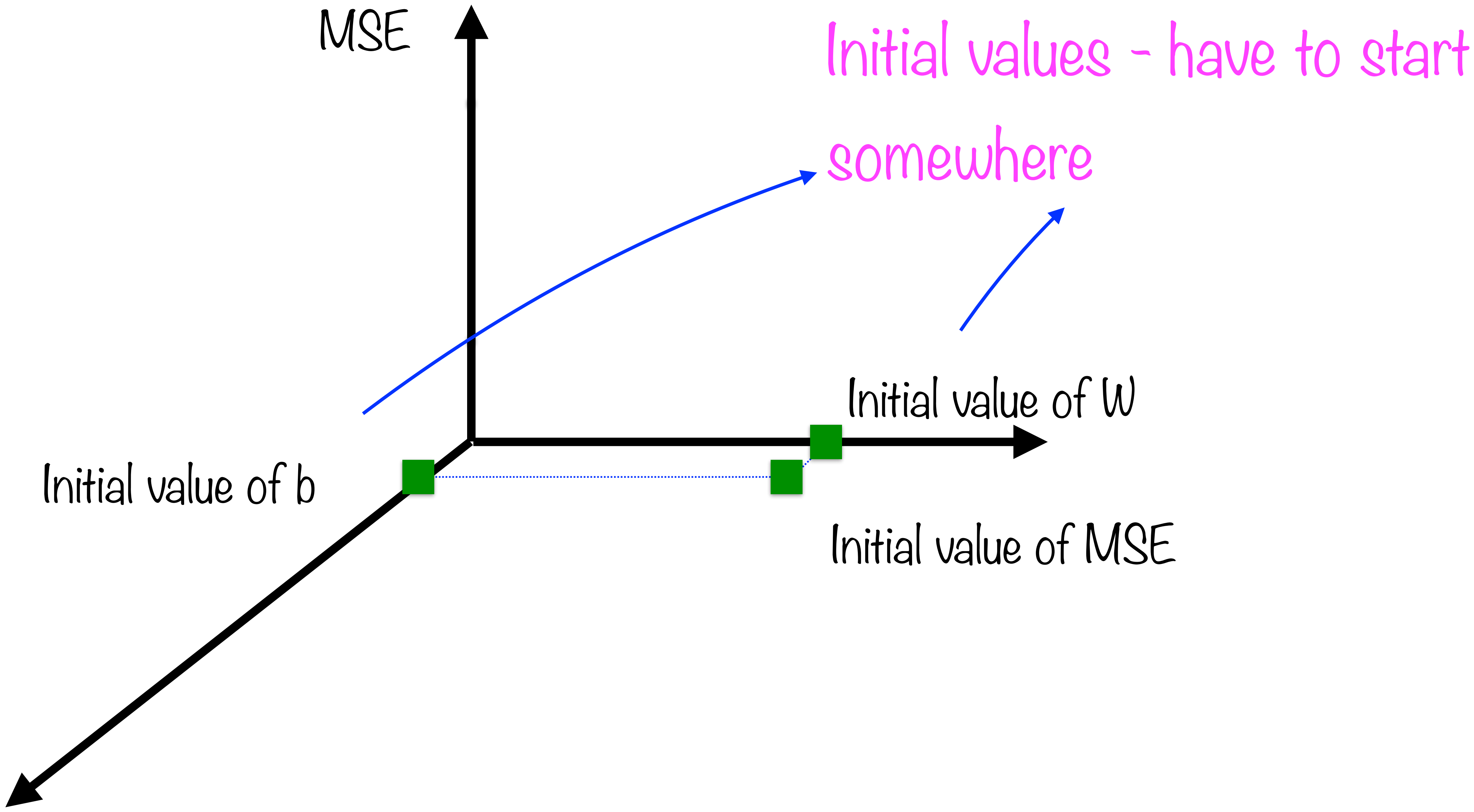
Minimizing MSE



"Training" the Algorithm

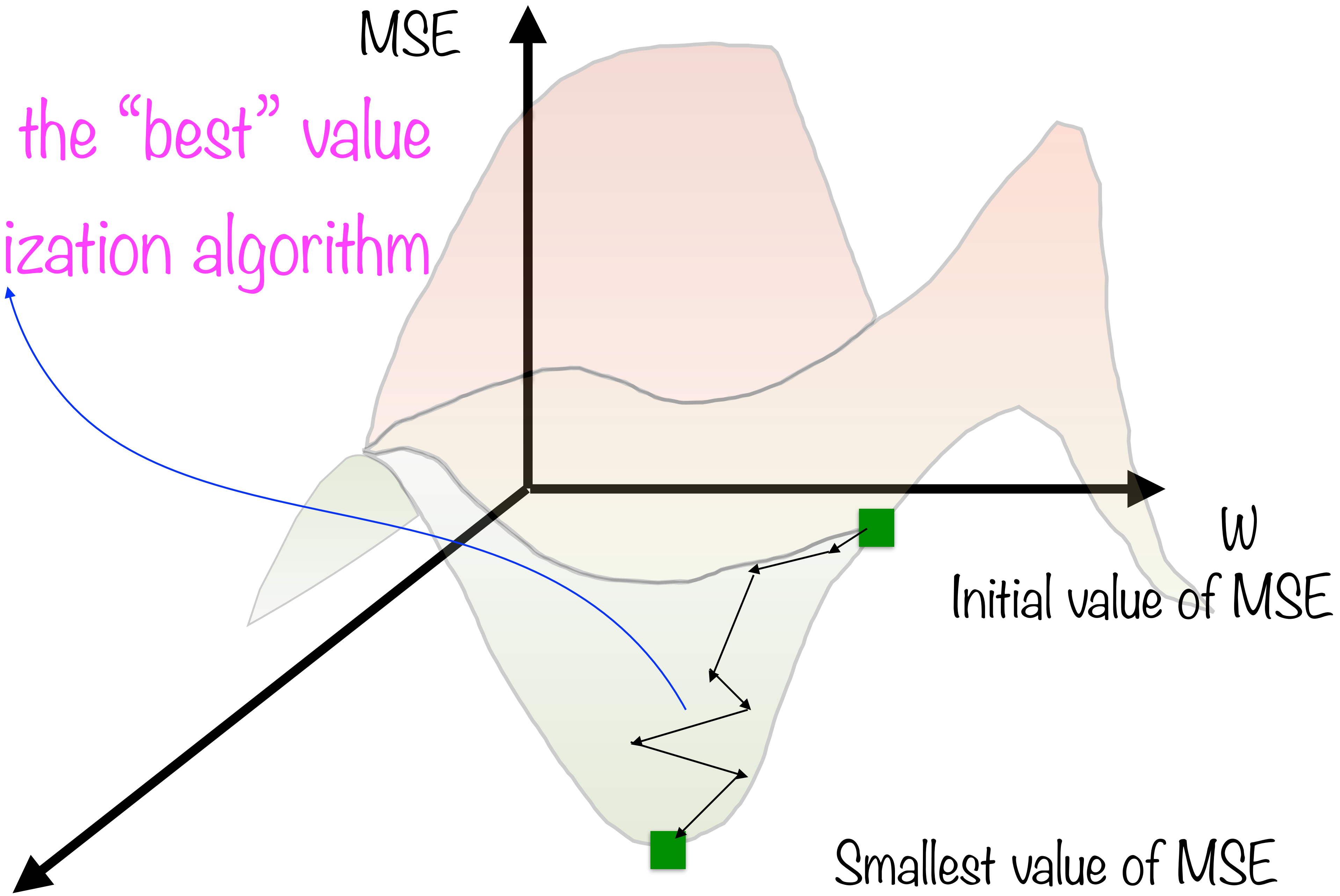


Start Somewhere



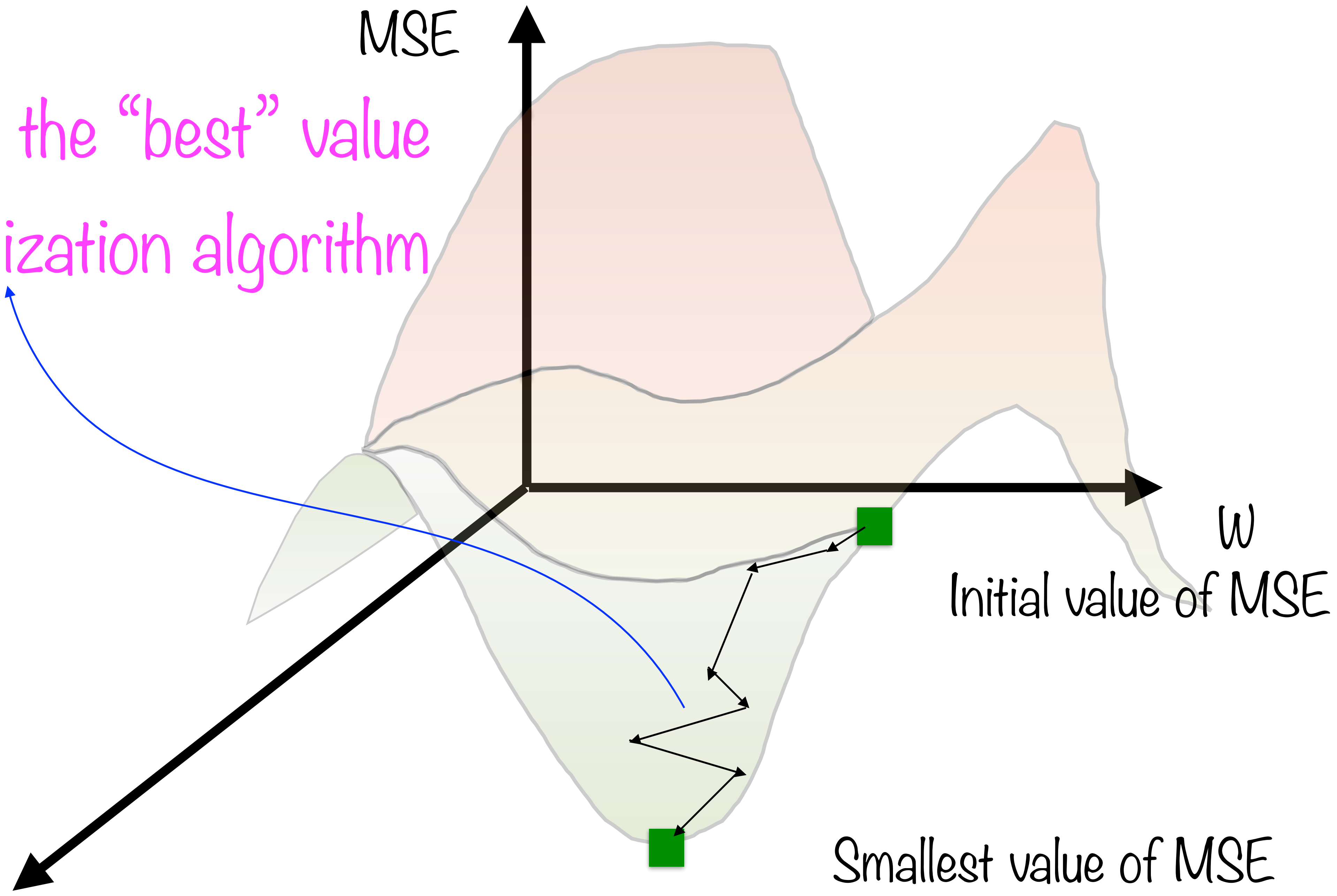
“Gradient Descent”

Converging on the “best” value using an optimization algorithm

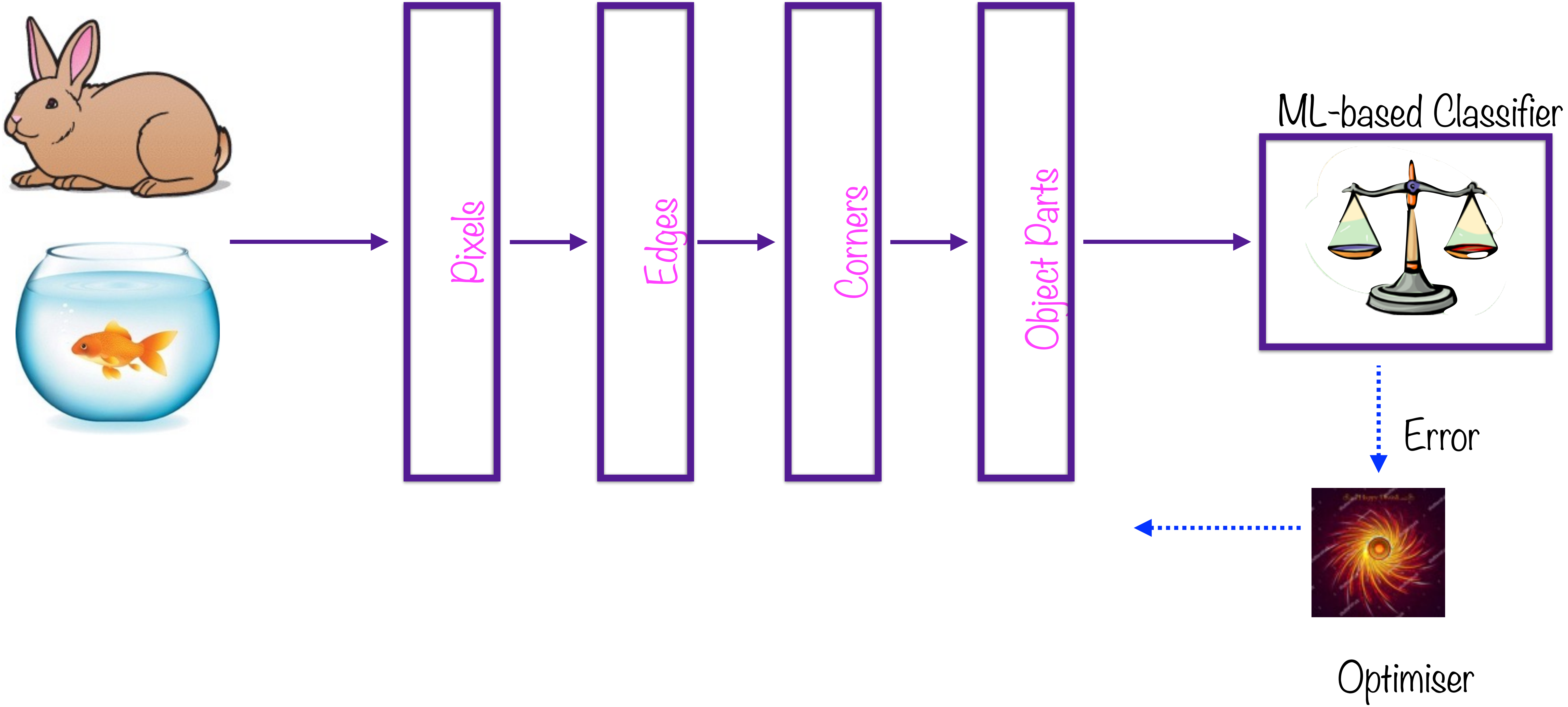


“Gradient Descent”

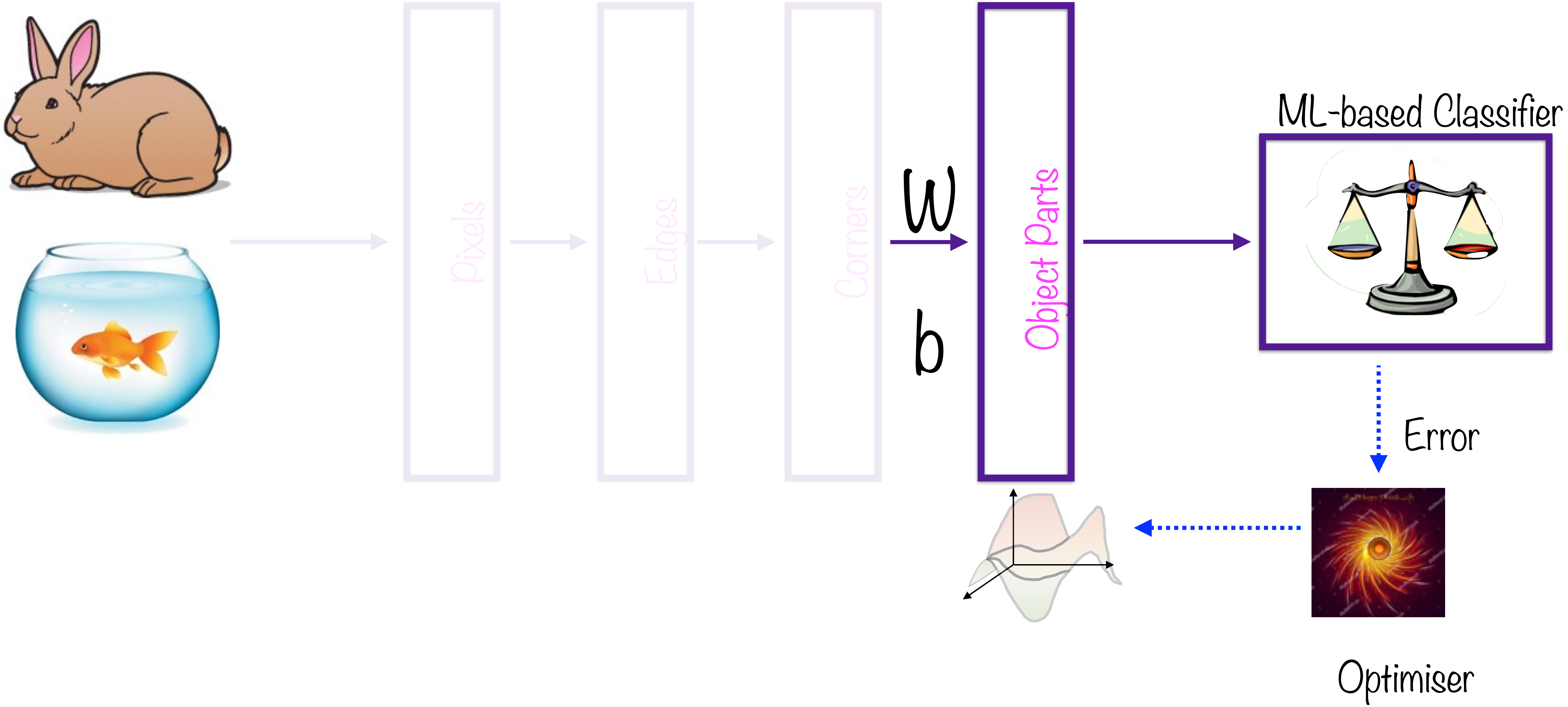
Converging on the “best” value using an optimization algorithm



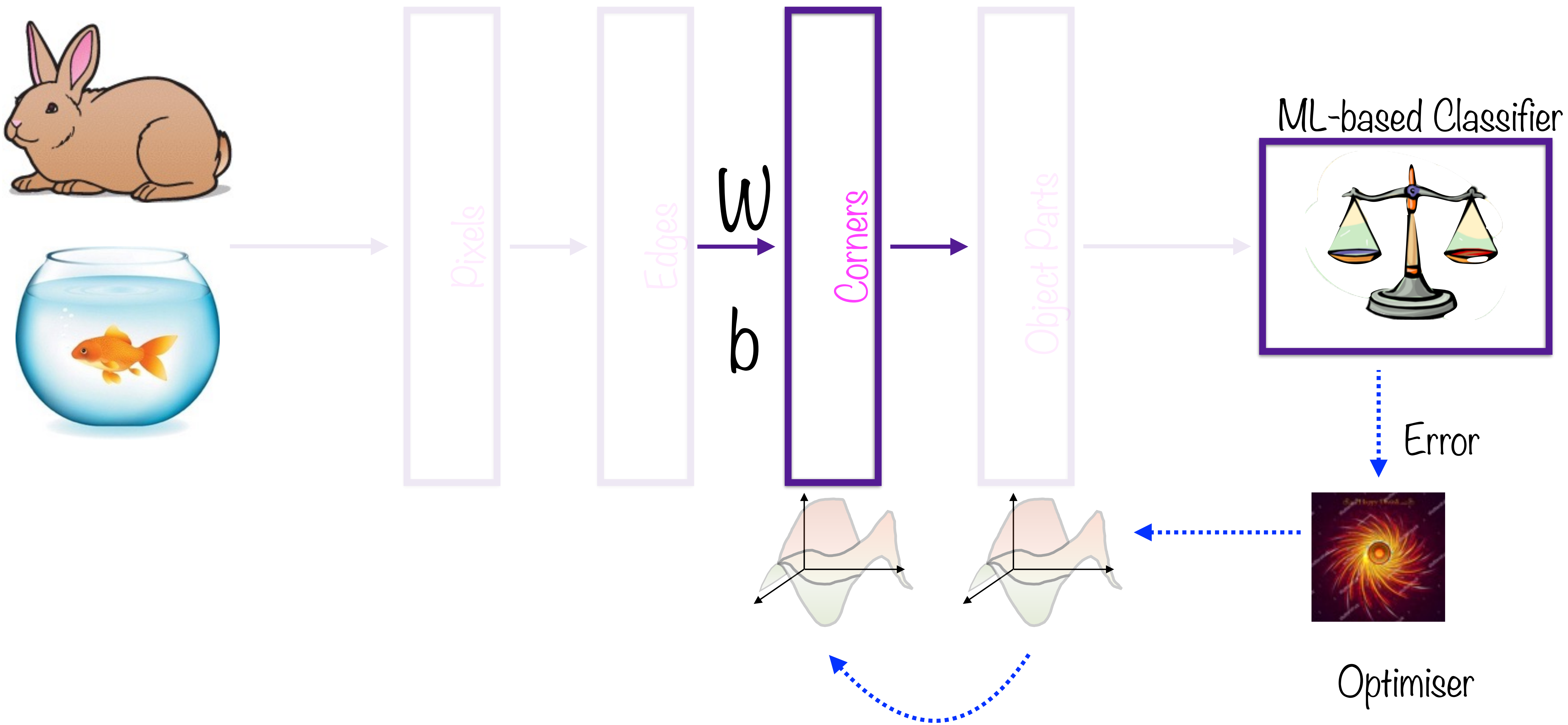
Training via Back Propagation



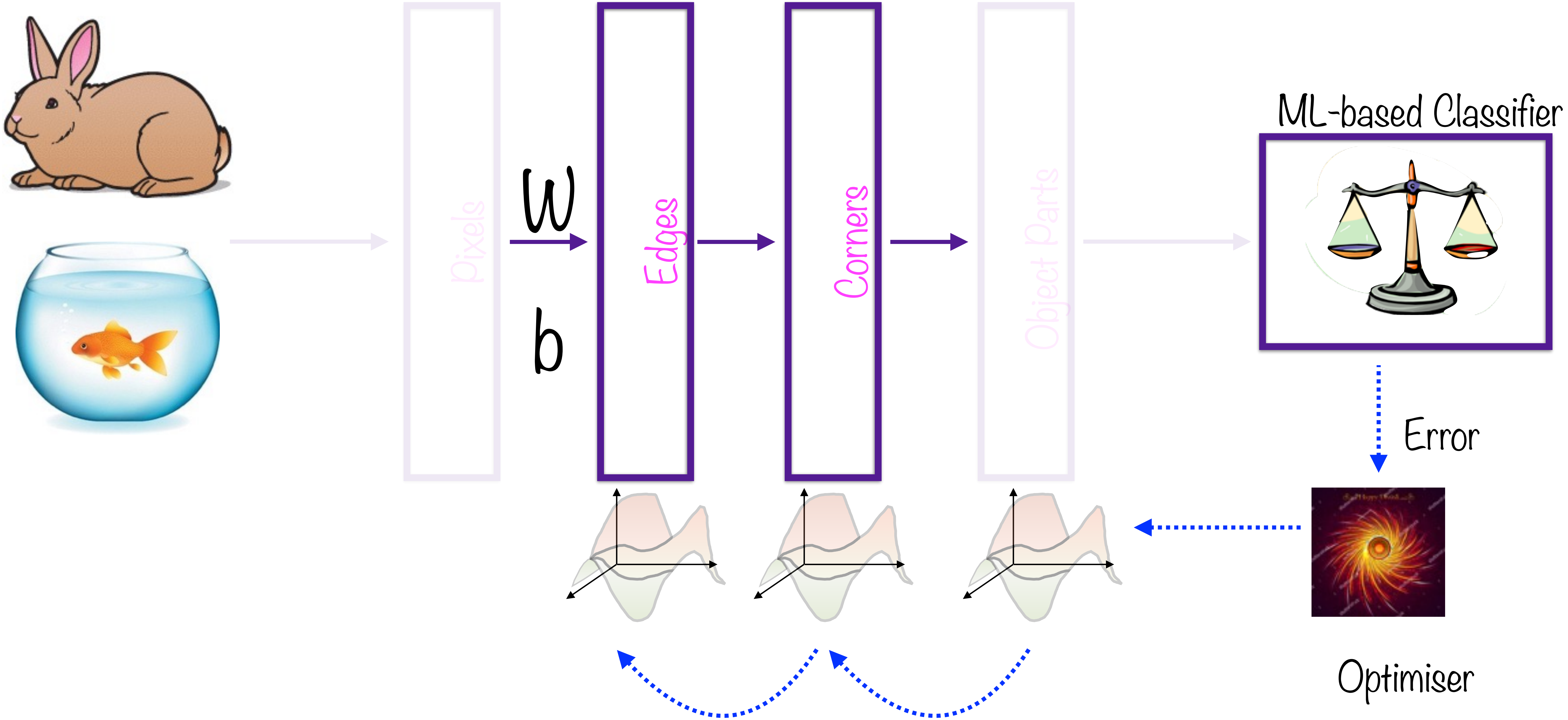
Training via Back Propagation



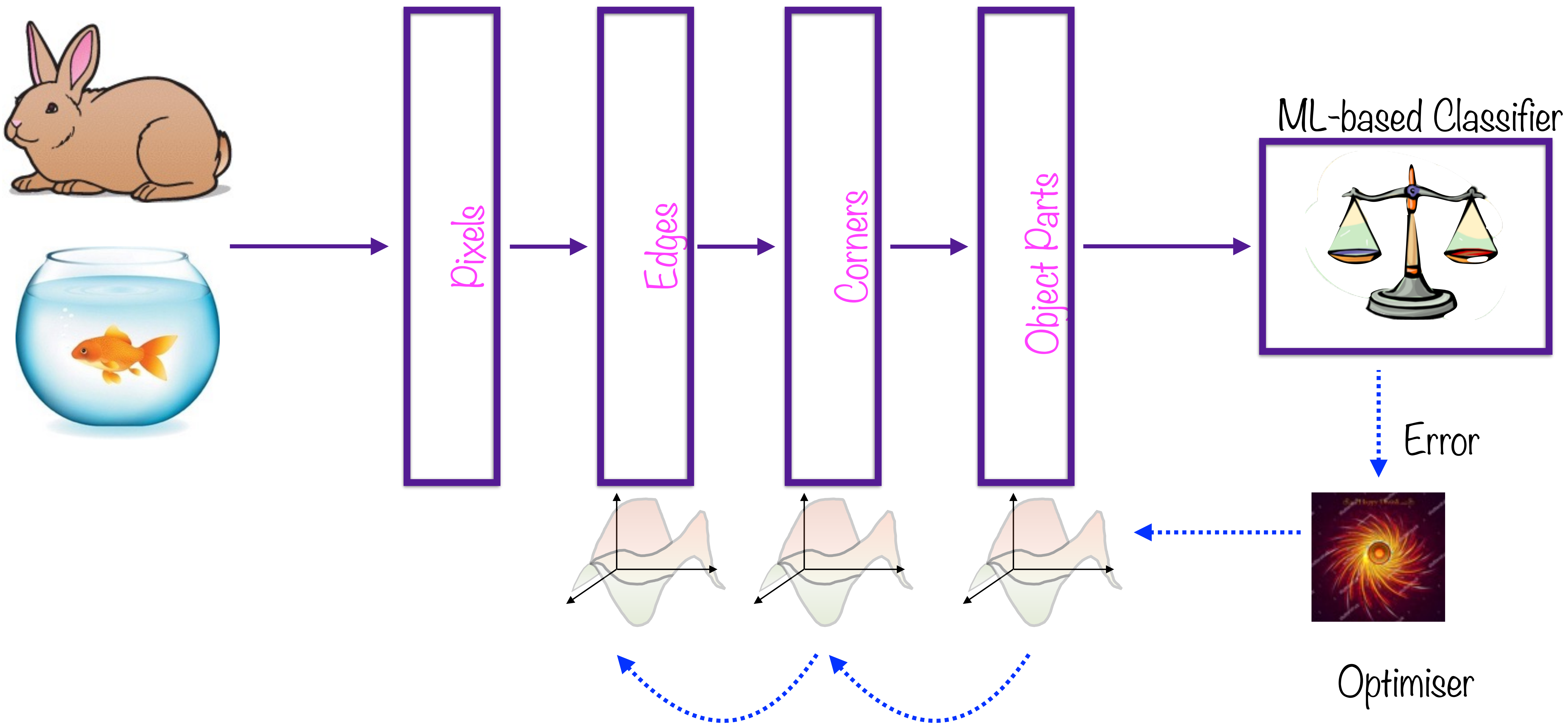
Training via Back Propagation



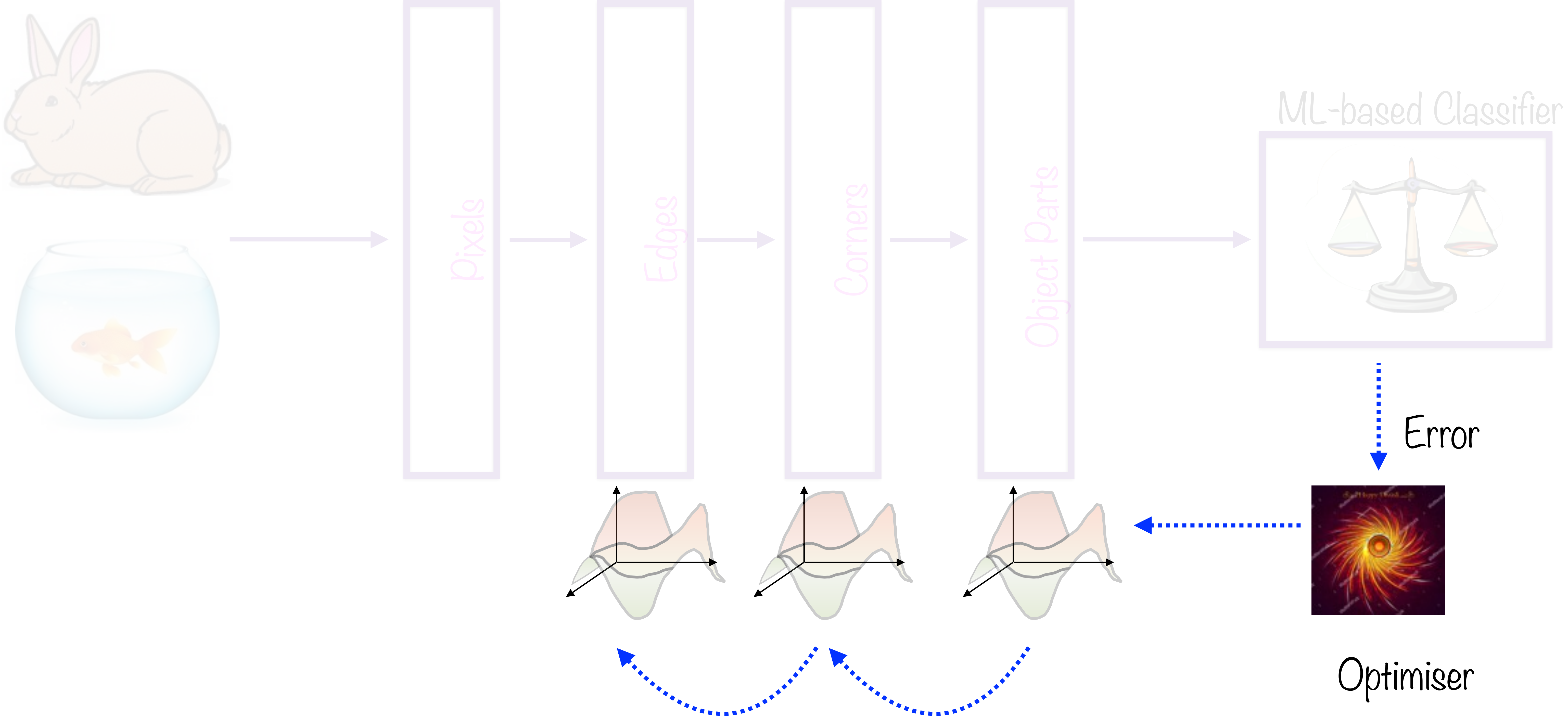
Training via Back Propagation



Training via Back Propagation



Training via Back Propagation



Back propagation allows the weights and biases of the neurons to converge to their final values

Hyperparameters

Decisions in Traditional ML Models

Initial values	Type of optimizer
Number of epochs	Batch size

More Decisions in Neural Networks

Network Topology i.e. neuron connections	Number of layers
Number of neurons in each layer	Activation function

How well the model performs is sensitive to these decisions

These are hyperparameters of our model

Model Parameters vs. Hyperparameters

Model parameters

The weights and biases determined **during the training** process

Result of the training process

Used to **make predictions**

Measure using **validation datasets** to find the best possible model

Hyperparameters

The **design of the actual model** determined before the training process begins

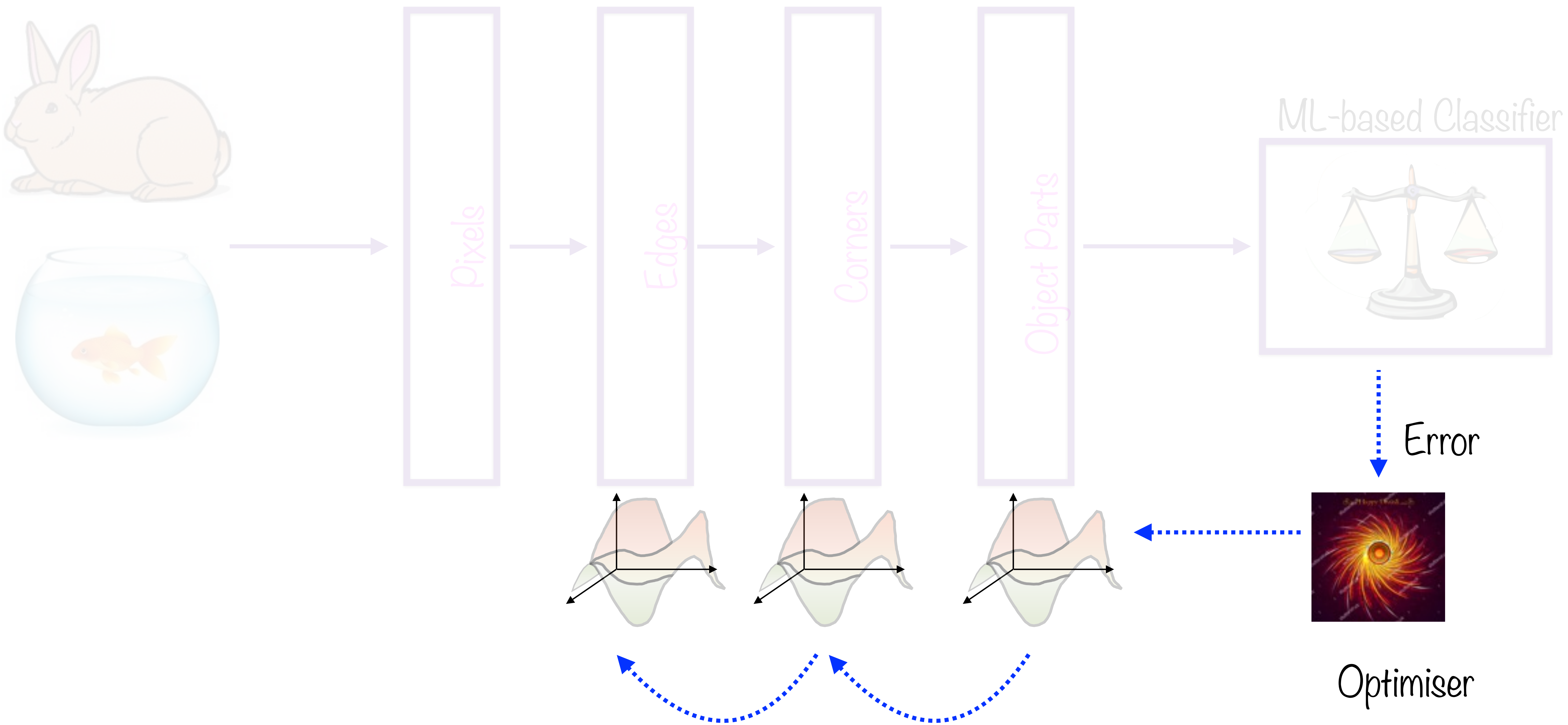
Input to the training process

Used to **generate** the **best** possible model

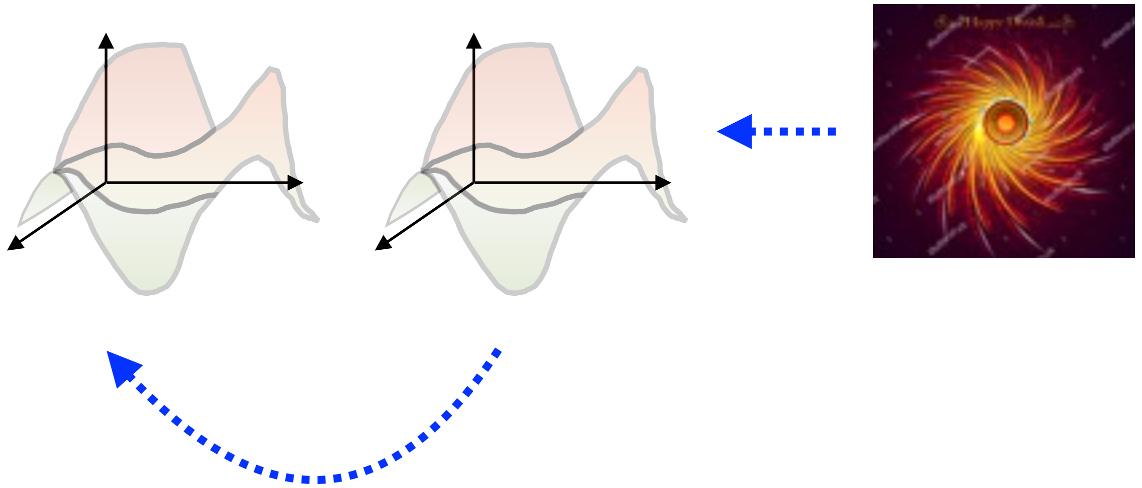
Hyperparameter tuning to generate the model which is then evaluated using validation datasets

Vanishing, Exploding Gradients, Dying Neurons

Training via Back Propagation



Back Propagation



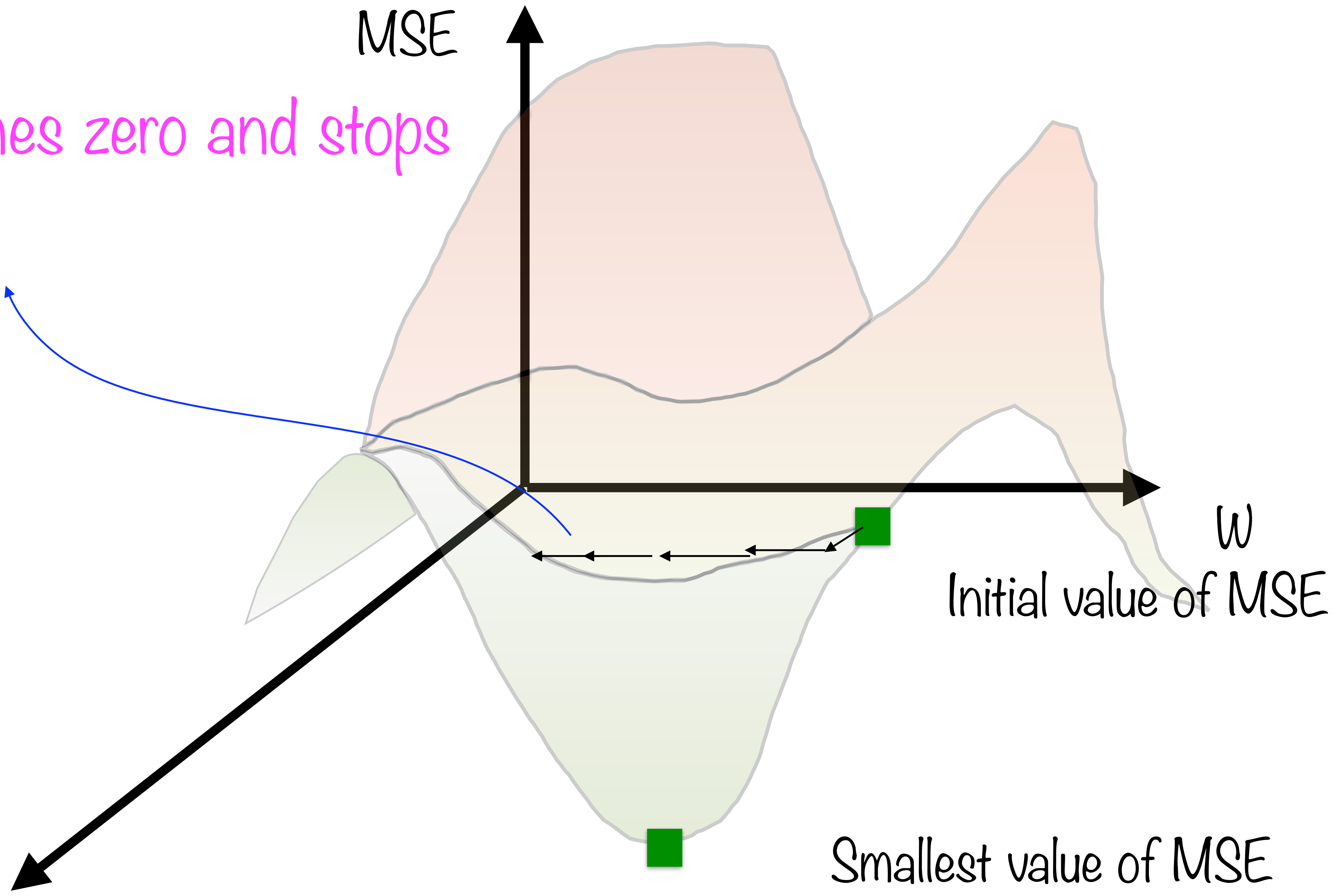
This is an iterative process

Fails either if

- gradients don't change at all
- gradients change too fast

“Vanishing Gradient Problem”

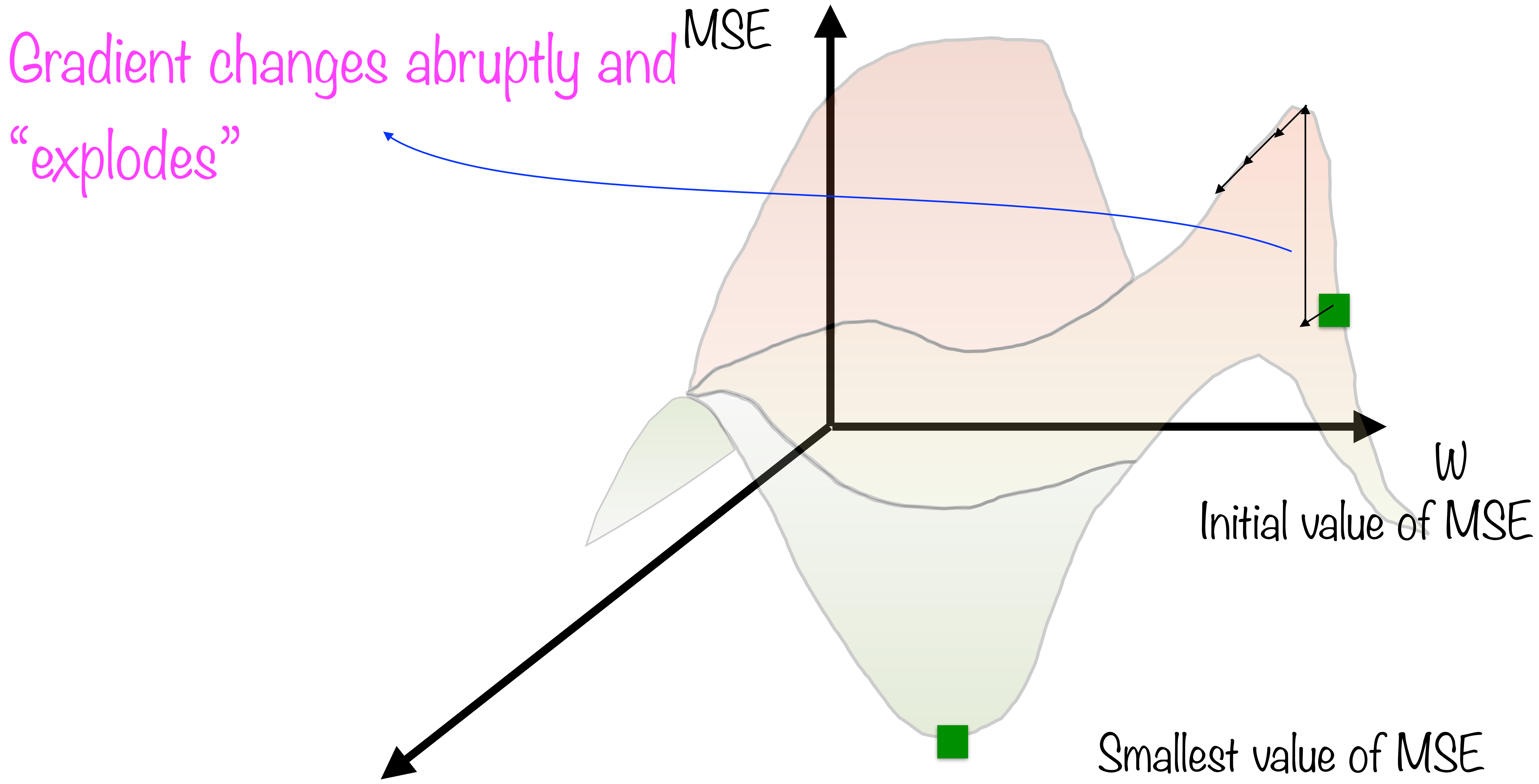
Gradient becomes zero and stops changing



The weights of the earlier layers remain unchanged

The algorithm never converges to a good solution

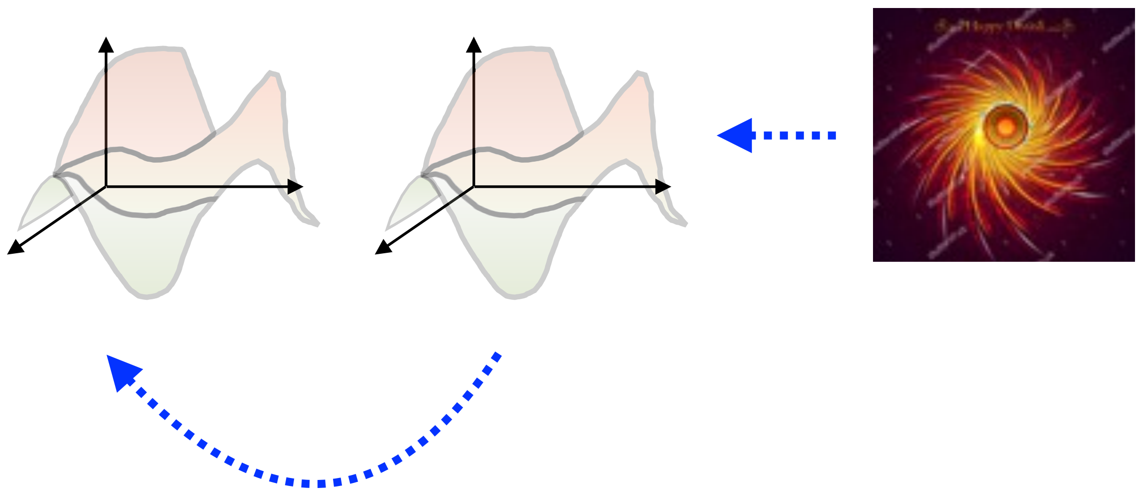
“Exploding Gradient Problem”



The weights of the layers become larger and meaningless

The algorithm diverges and never reaches a good solution

Vanishing and Exploding Gradients



Back propagation fails if

- gradients are vanishing
- gradients are exploding

This was an important reason why DNNs were mostly abandoned for a long time

Coping with Vanishing/Exploding Gradients

Proper initialisation	Gradient clipping
Batch normalisation	Non-saturating activation function

Xavier and He Initialization

Proper initialisation

The variance of the outputs in each direction is equal to variance of inputs

Connections weights must be initialized randomly

Xavier and He Initialization

Proper initialisation

Normal distribution:

- mean 0
- standard deviation based on num_inputs and num_outputs for that layer

Xavier and He Initialization

Proper initialisation

Uniform distribution:

- between **-r** and **+r**
- **r** based on num_inputs and num_outputs for that layer

Coping with Vanishing/Exploding Gradients

Proper initialisation	Gradient clipping
Batch normalisation	Non-saturating activation function

Gradient Clipping

Gradient clipping

Limit the gradients to under a threshold during back propagation

Most often used with recurrent neural networks

Coping with Vanishing/Exploding Gradients

Proper initialisation	Gradient clipping
Batch normalisation	Non-saturating activation function

Batch Normalisation

Batch normalisation

Zero center the inputs before passing to the activation functions

Subtract the mean and divide by the standard deviation

Allows use of saturating activation functions as well

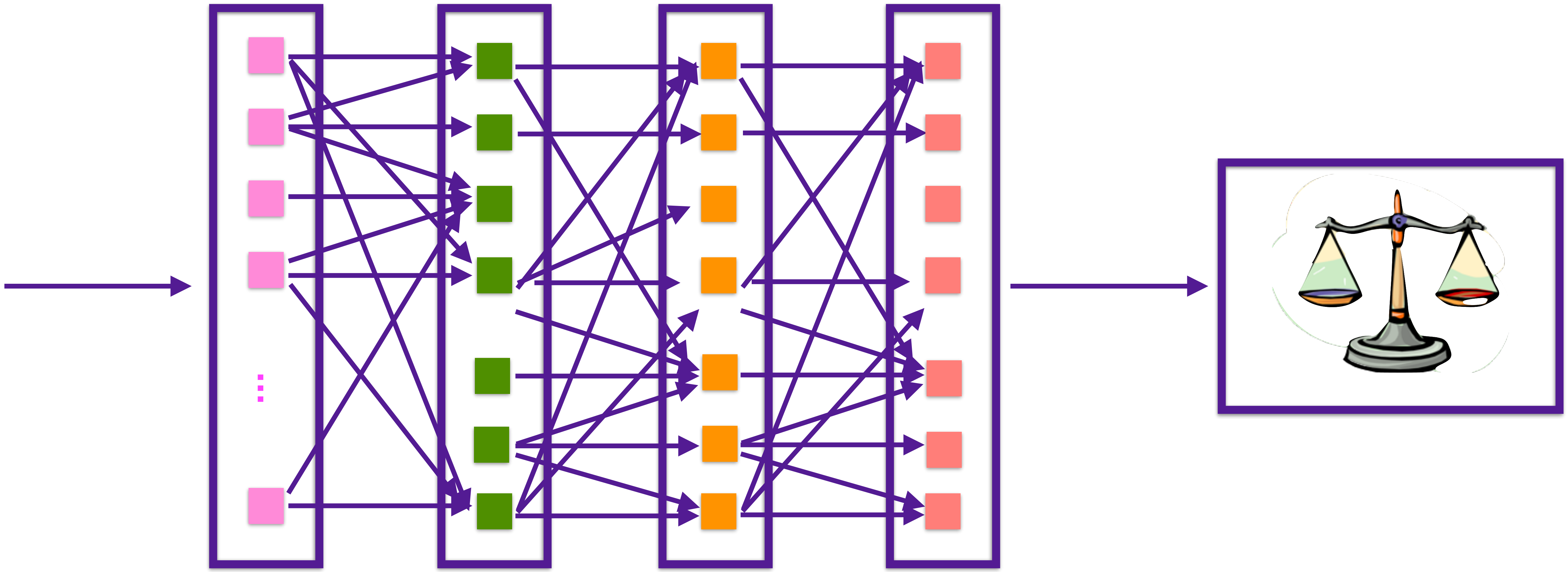
Coping with Vanishing/Exploding Gradients

Proper initialisation	Gradient clipping
Batch normalisation	Non-saturating activation function

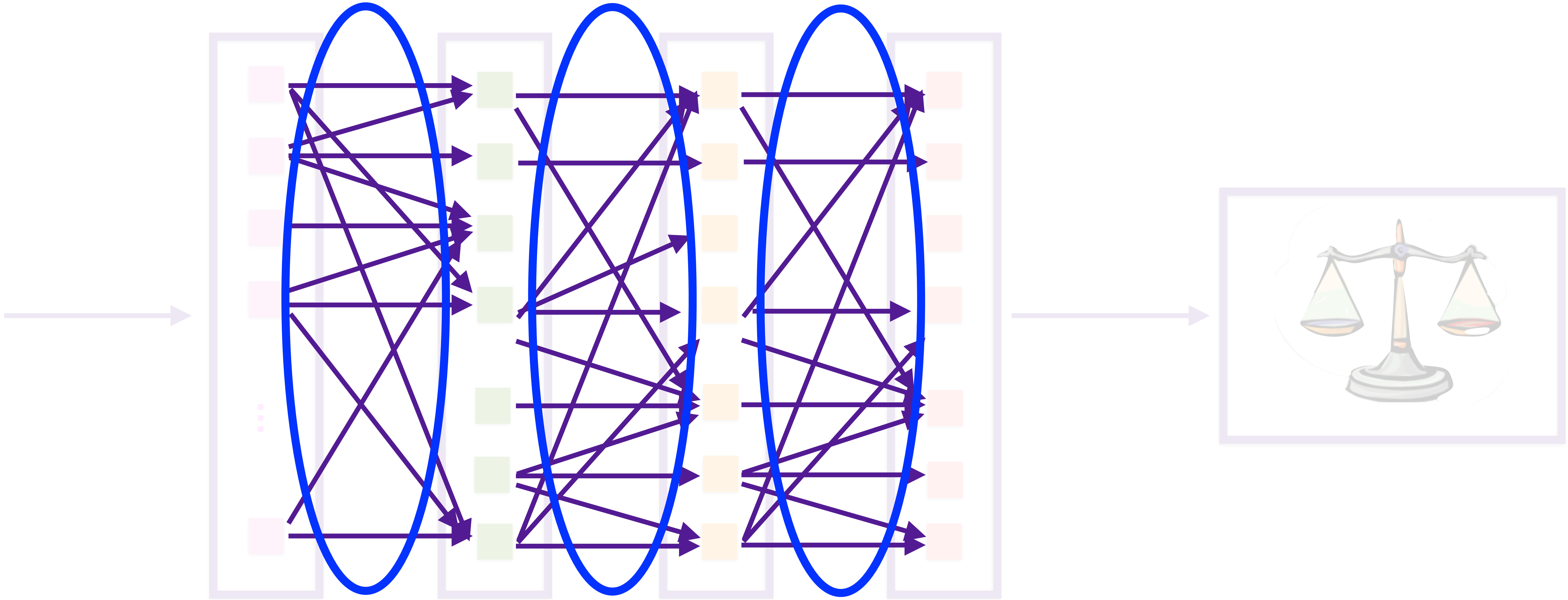
Coping with Vanishing/Exploding Gradients

Proper initialisation	Gradient clipping
Batch normalisation	Non-saturating activation function

A Neural Network

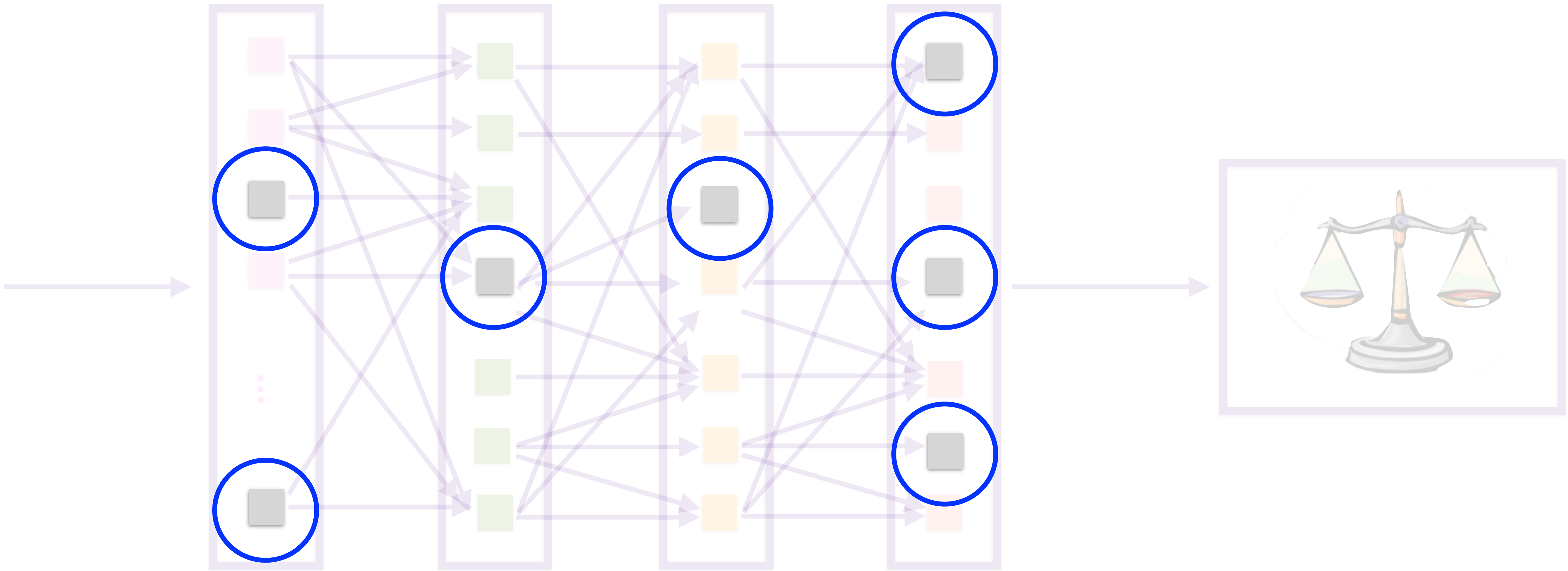


Unresponsive Neurons



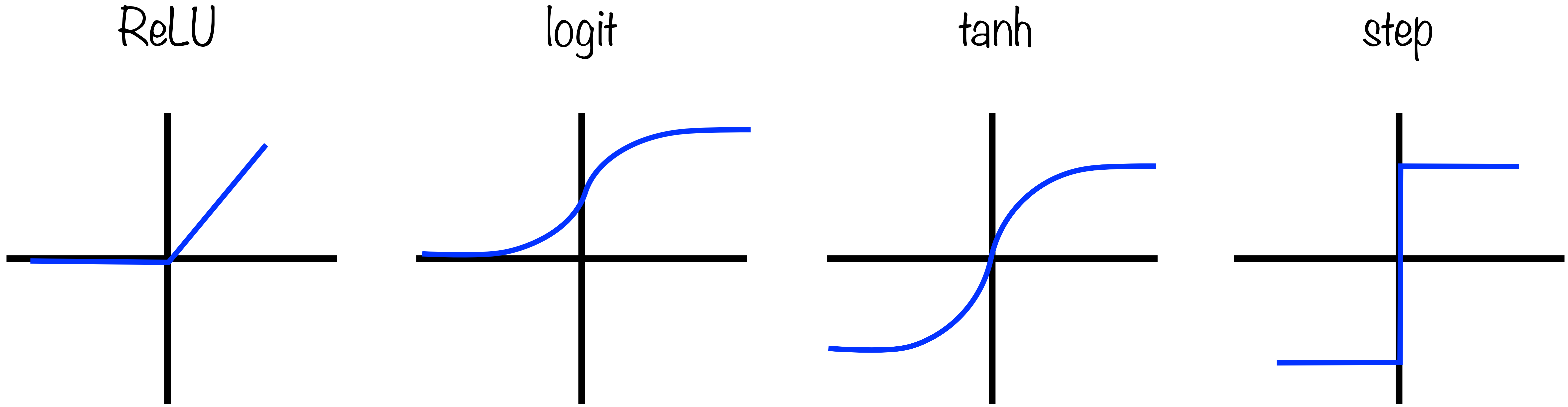
What if the weights of the connections do not change in response to changing input?

Unresponsive Neurons



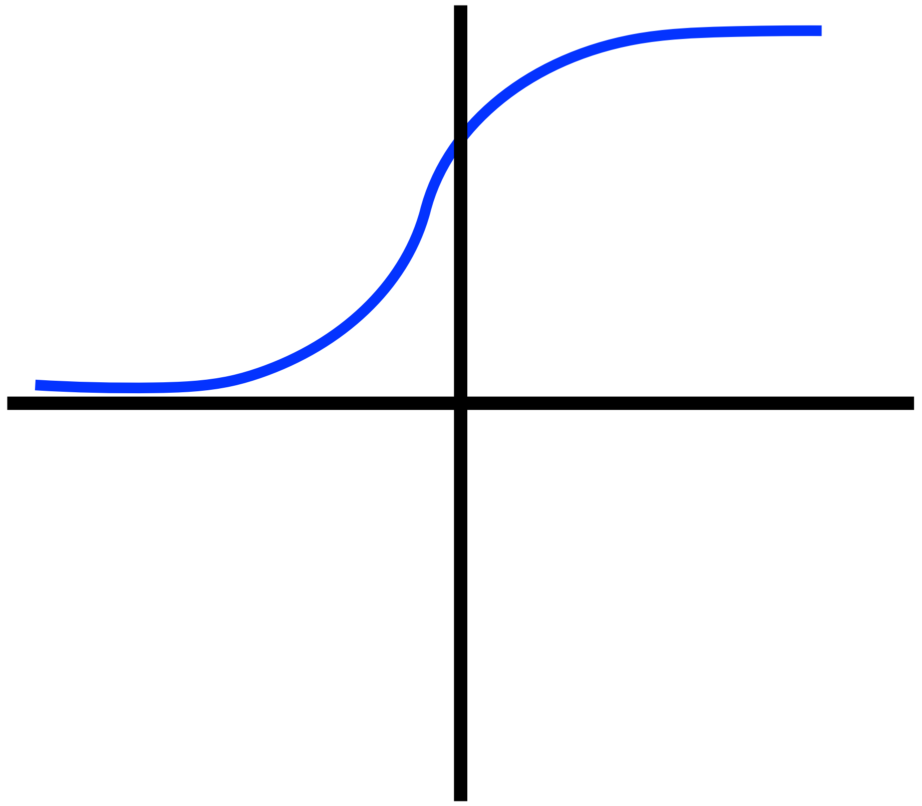
Neurons may be dead

Activation Functions



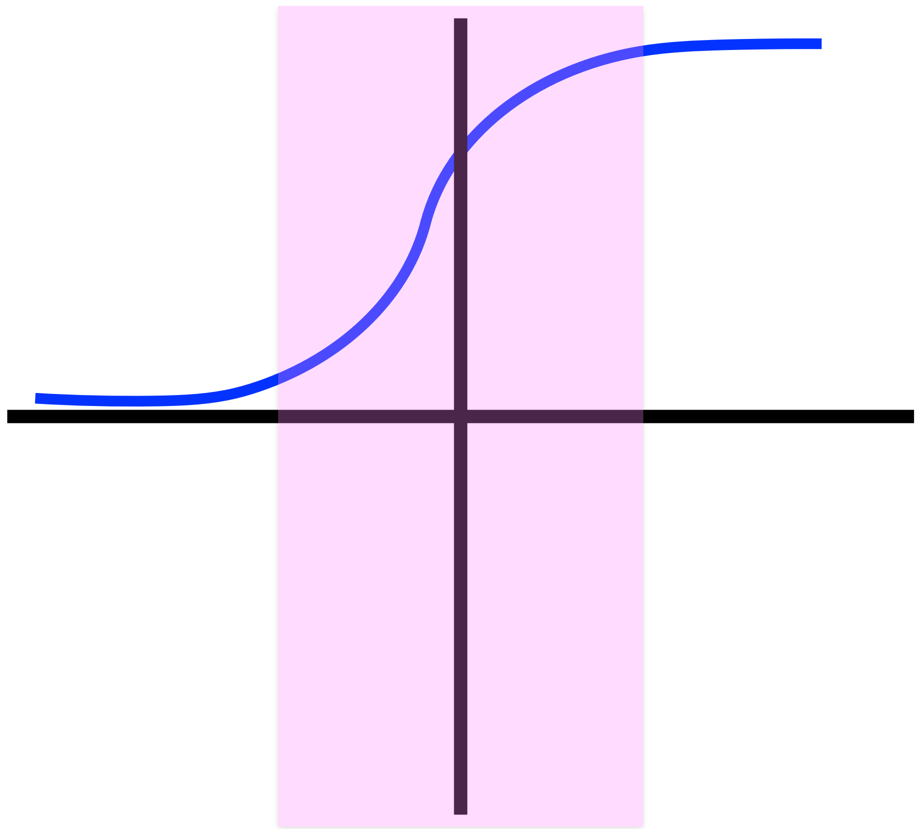
Various choices of activation functions exist and drive the design of your neural network

Activation Functions



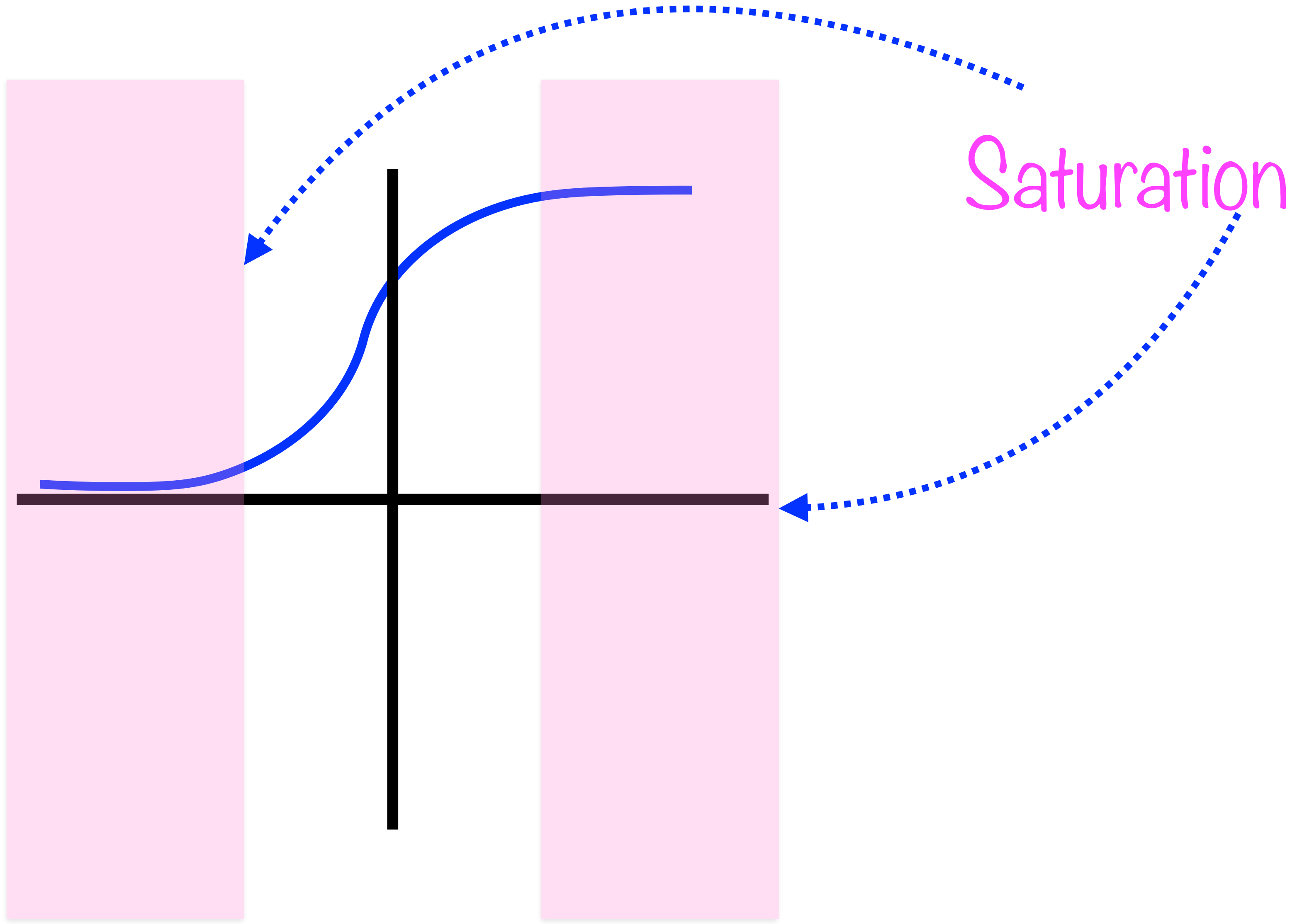
Consider an S-shaped (sigmoid) activation function

Activation Functions



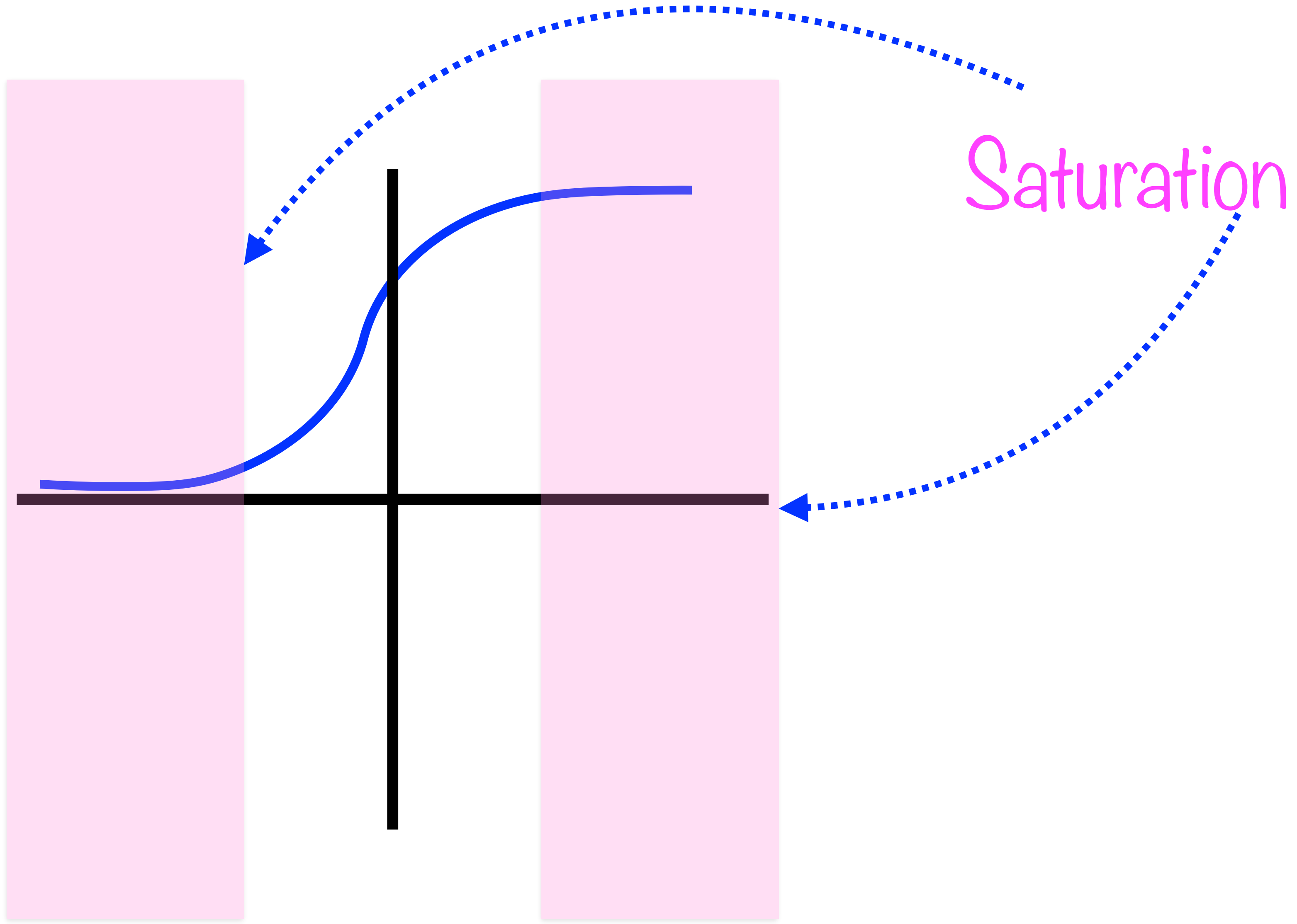
This is the active or responsive region of the function

Saturating Activation Functions

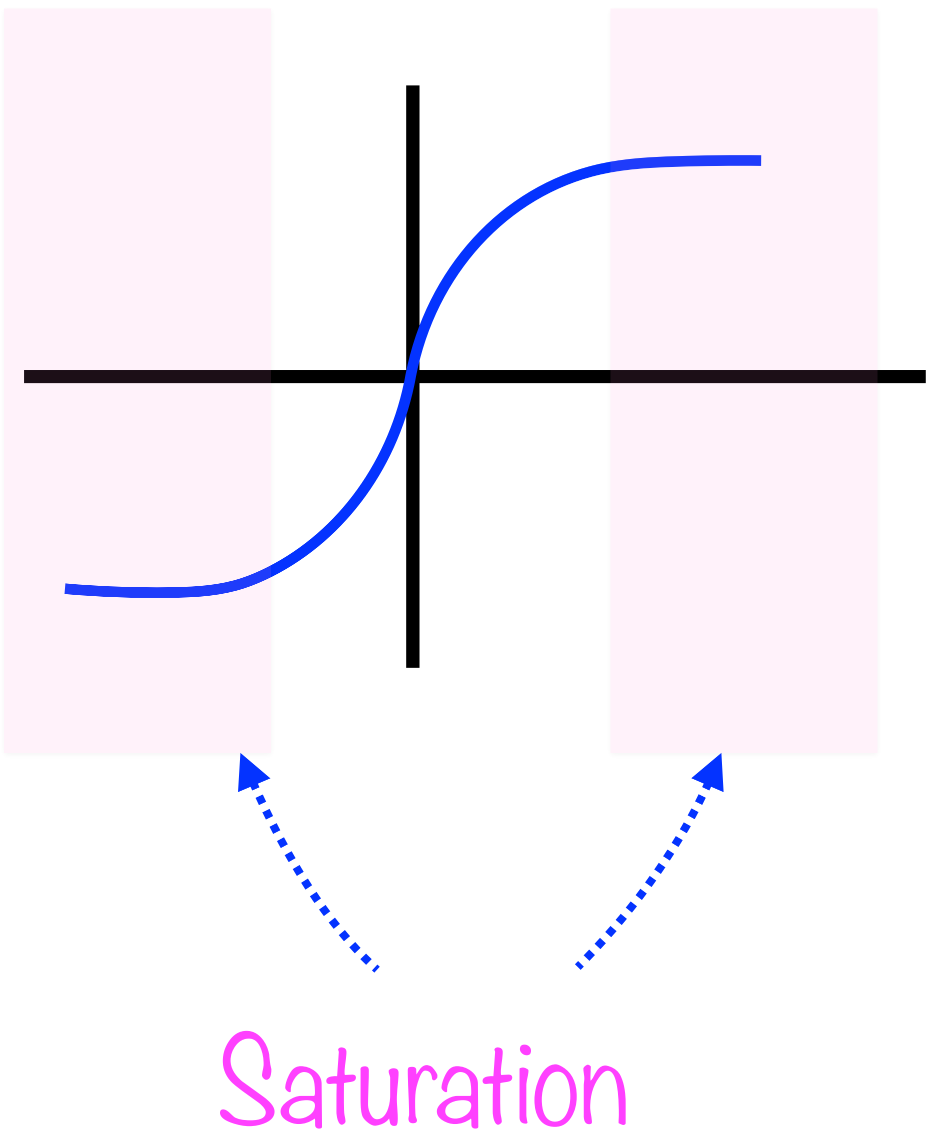


The activation function saturates at either end

Saturating Activation Functions



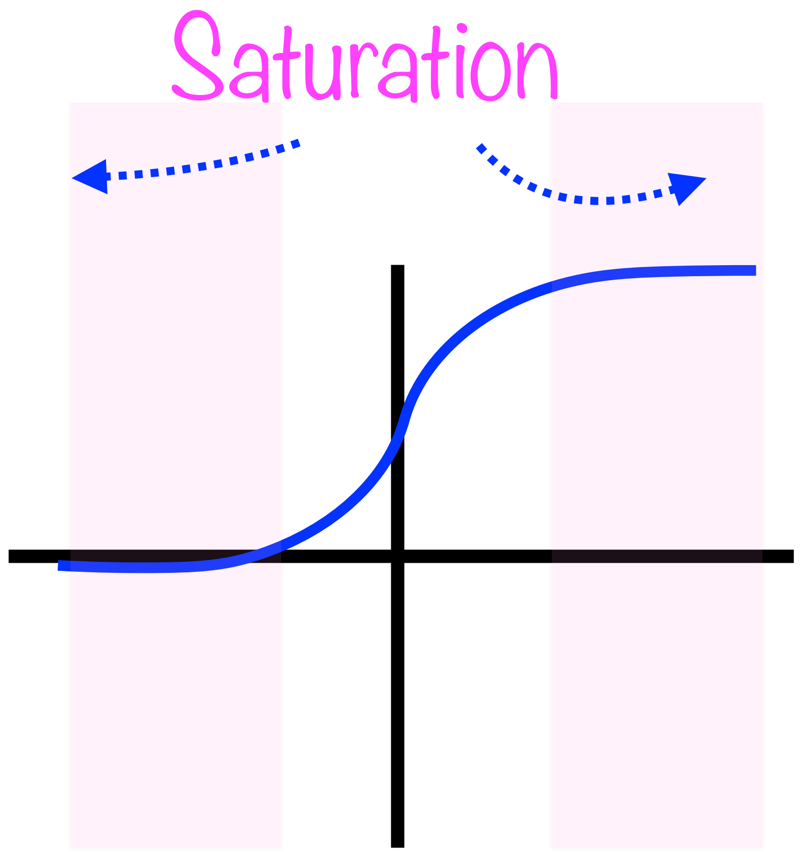
If a neuron operates within these saturation regions throughout training it might become unresponsive



Dying Neurons

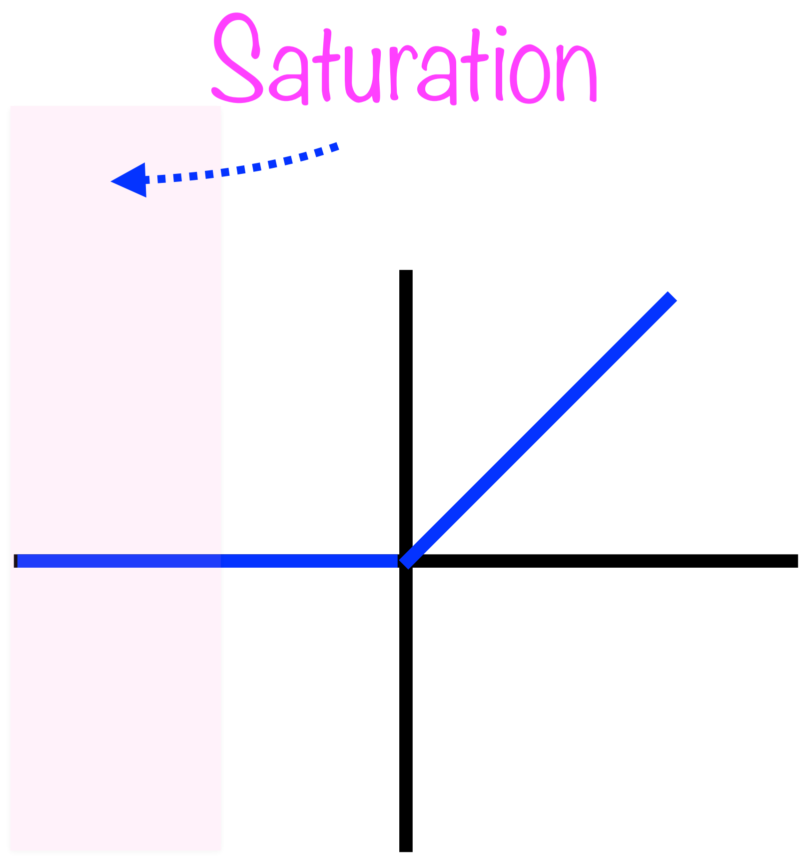
- Neuron might become unresponsive - output won't change as input changes
- If this continues throughout training, neuron is "dead"
- Saturation of neuron occurs at both ends of S-curve, for instance

Saturating Activation Functions



Logit Activation

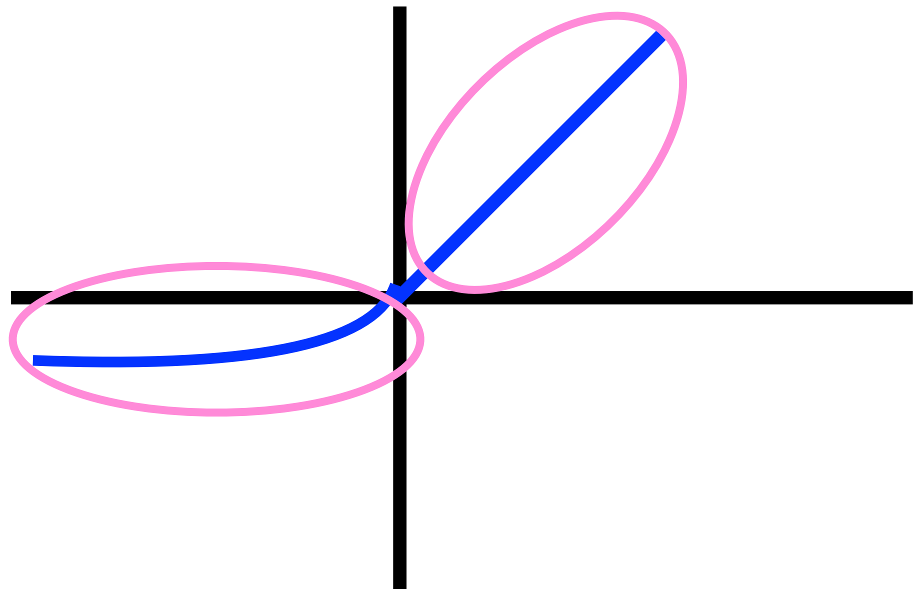
Saturates for very large and very small values of input



ReLU Activation

Saturates for very small (negative values) of inputs

ELU Activation



Mitigates dying-neuron problem of ReLU

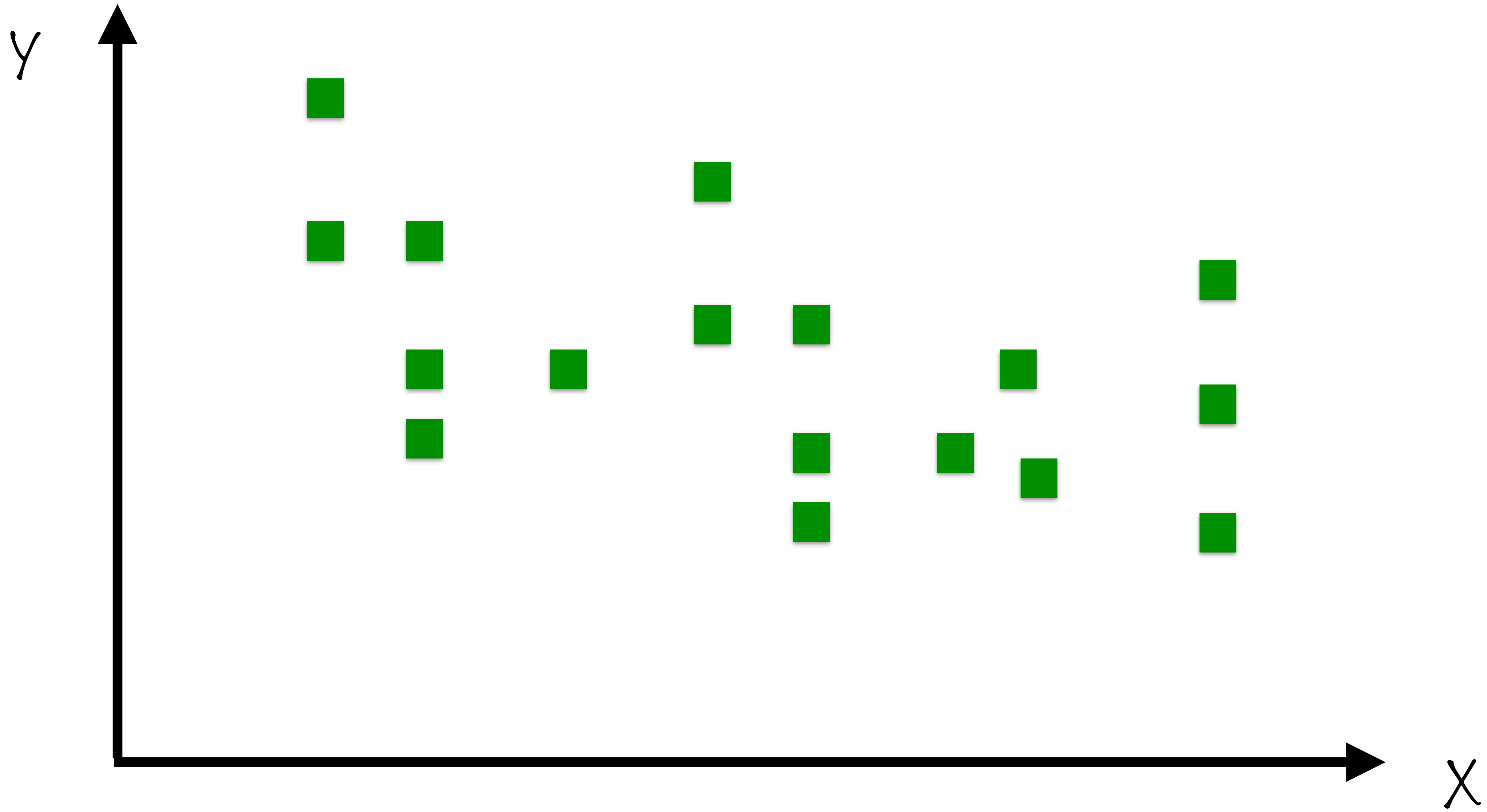
- Linear for positive values
- Exponential for negative values

ELU is the new favorite activation function

Dying and unresponsive neuron issues can be mitigated by using **activations functions** such as **ELU**

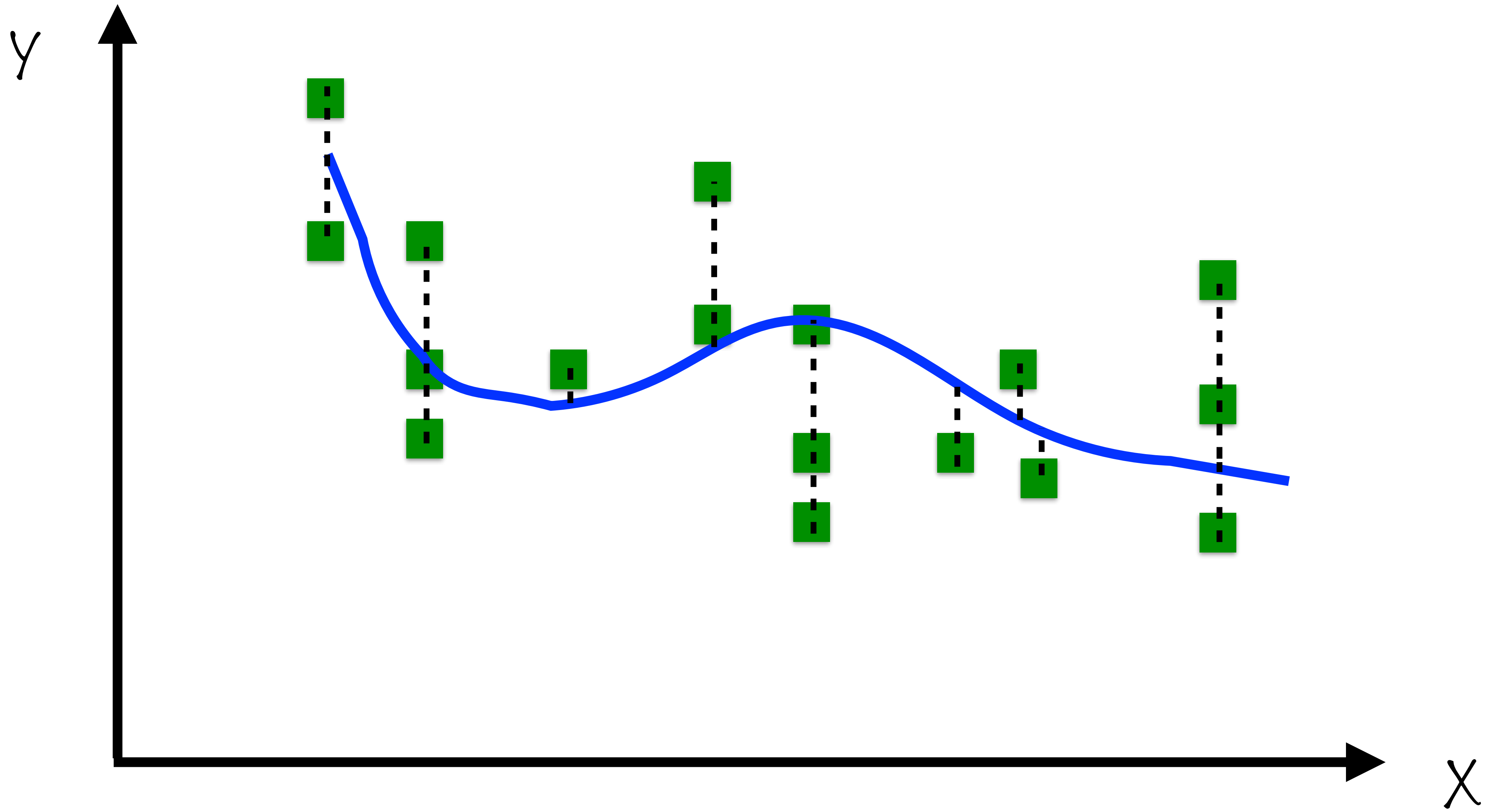
Overfitting and the Bias-Variance Trade-off

Connecting the Dots



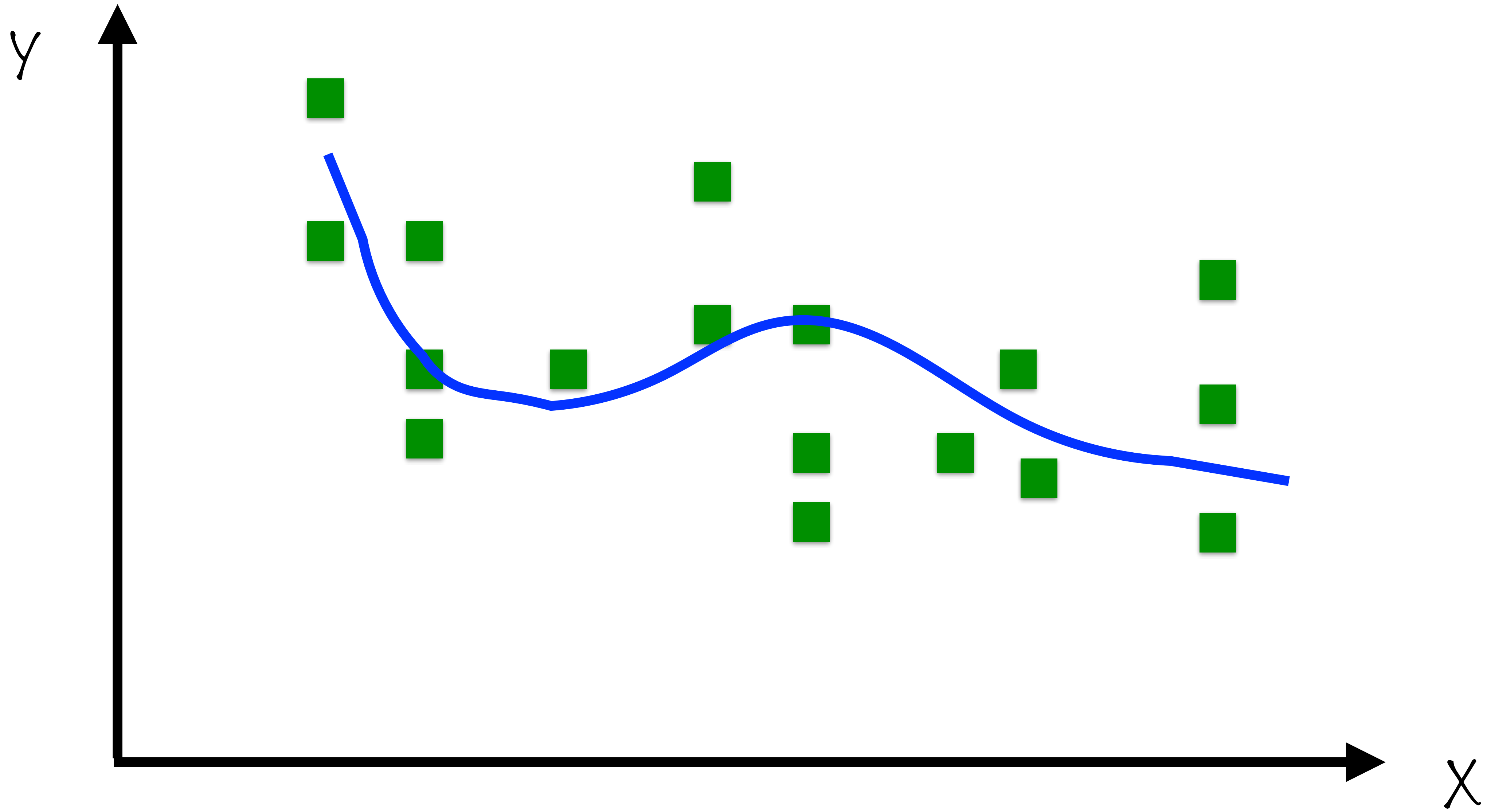
Challenge: Fit the "best" curve through these points

Good Fit?



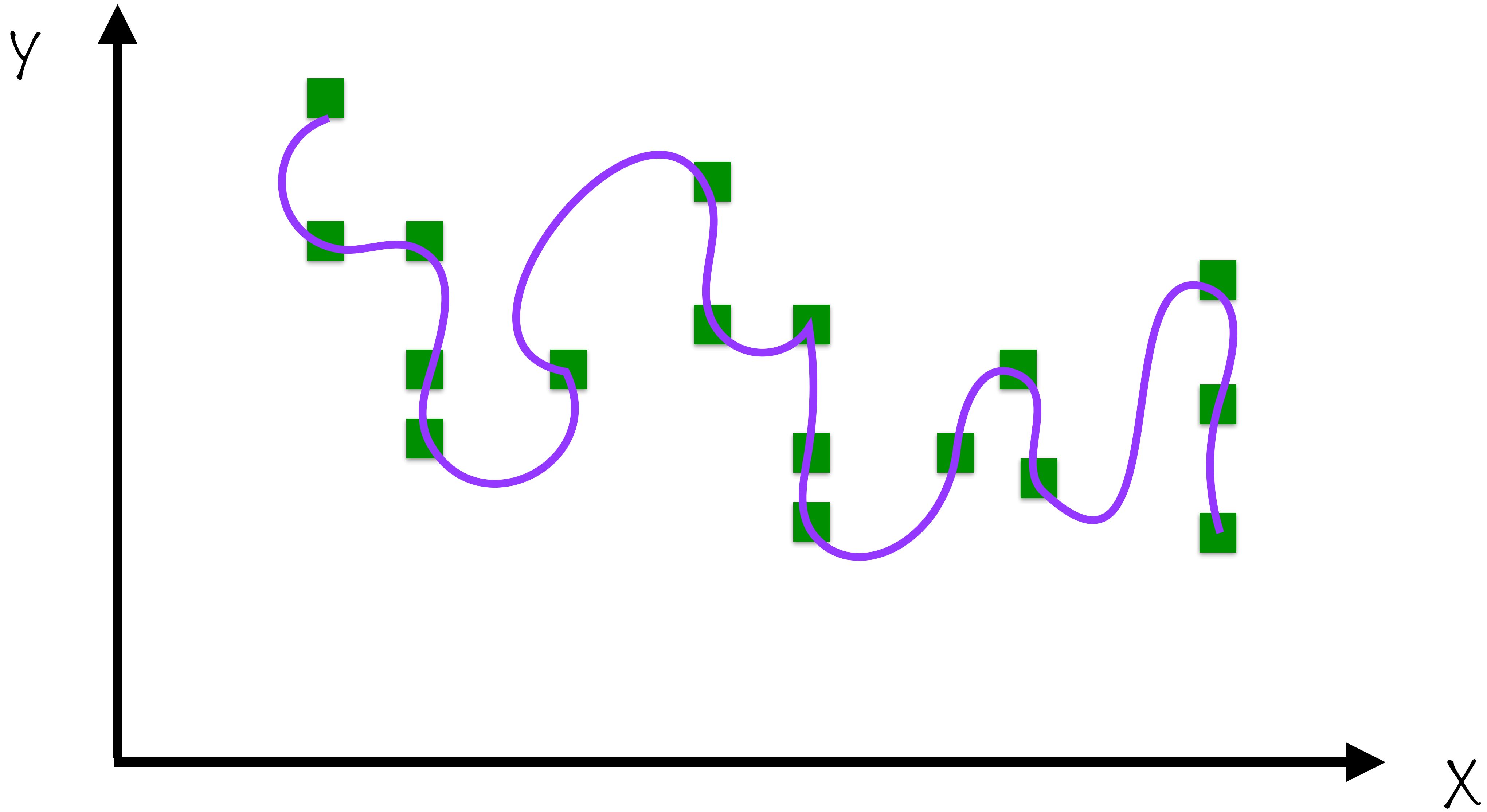
A curve has a “good fit” if the distances of points from the curve are small

Connecting the Dots



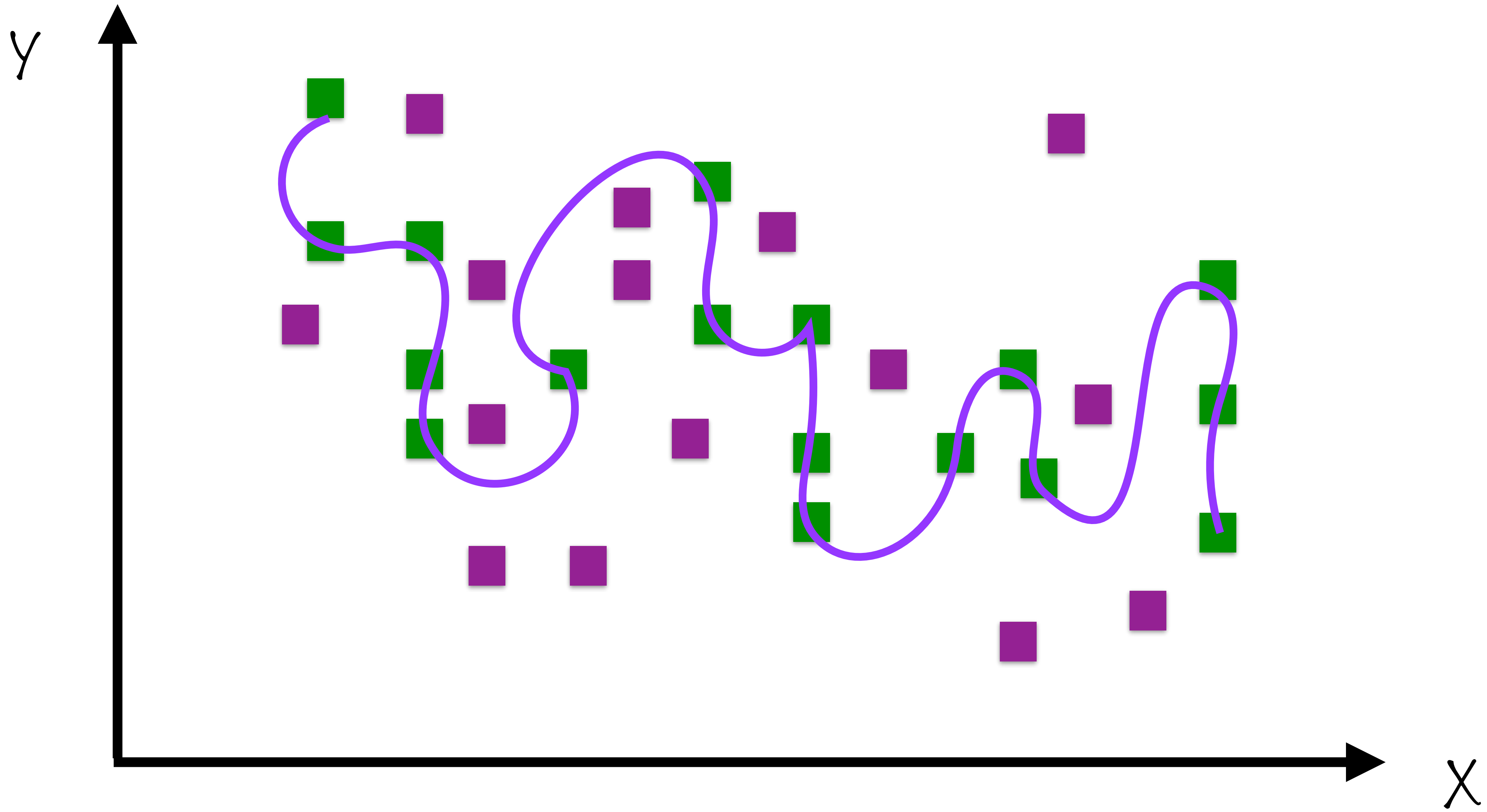
We could draw a pretty complex curve

Connecting the Dots



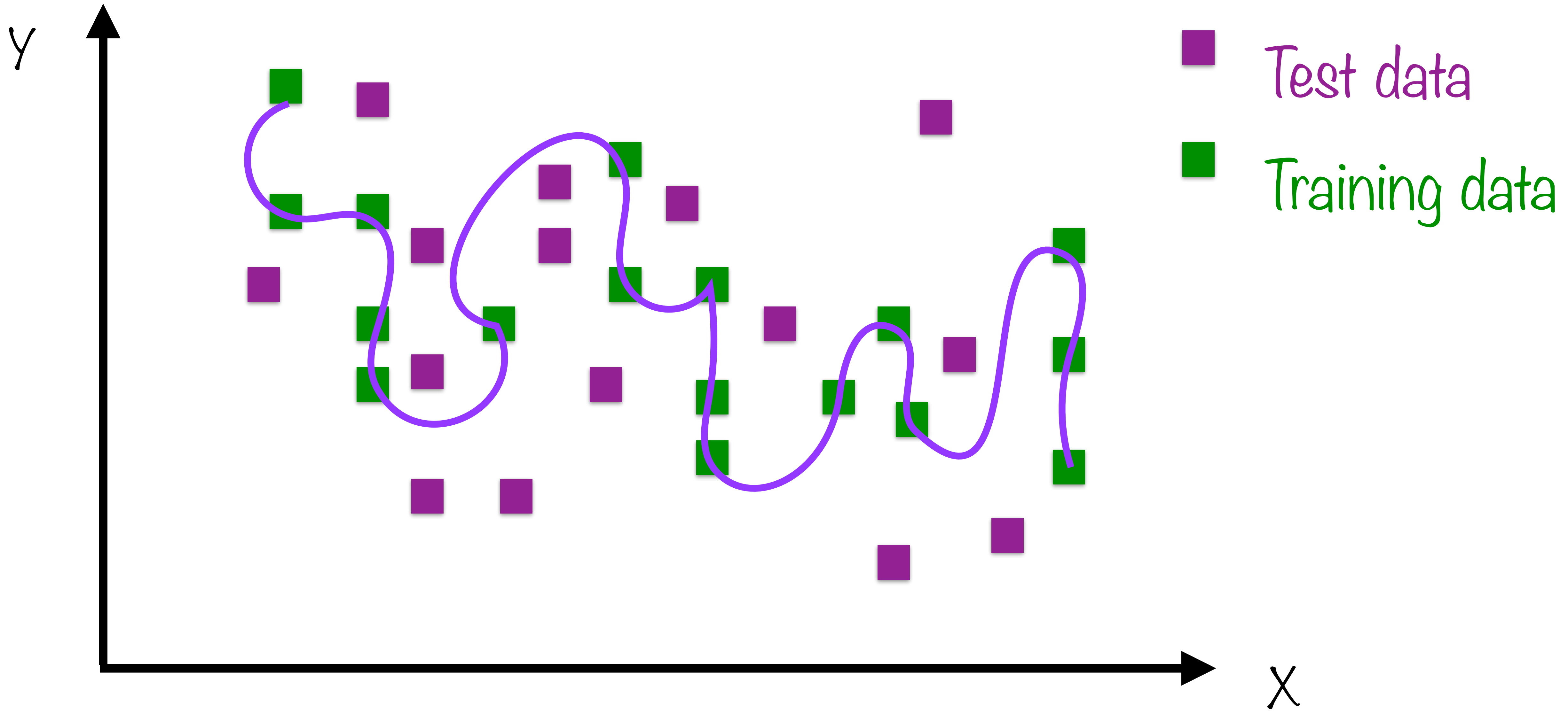
We can even make it pass through every single point

Connecting the Dots



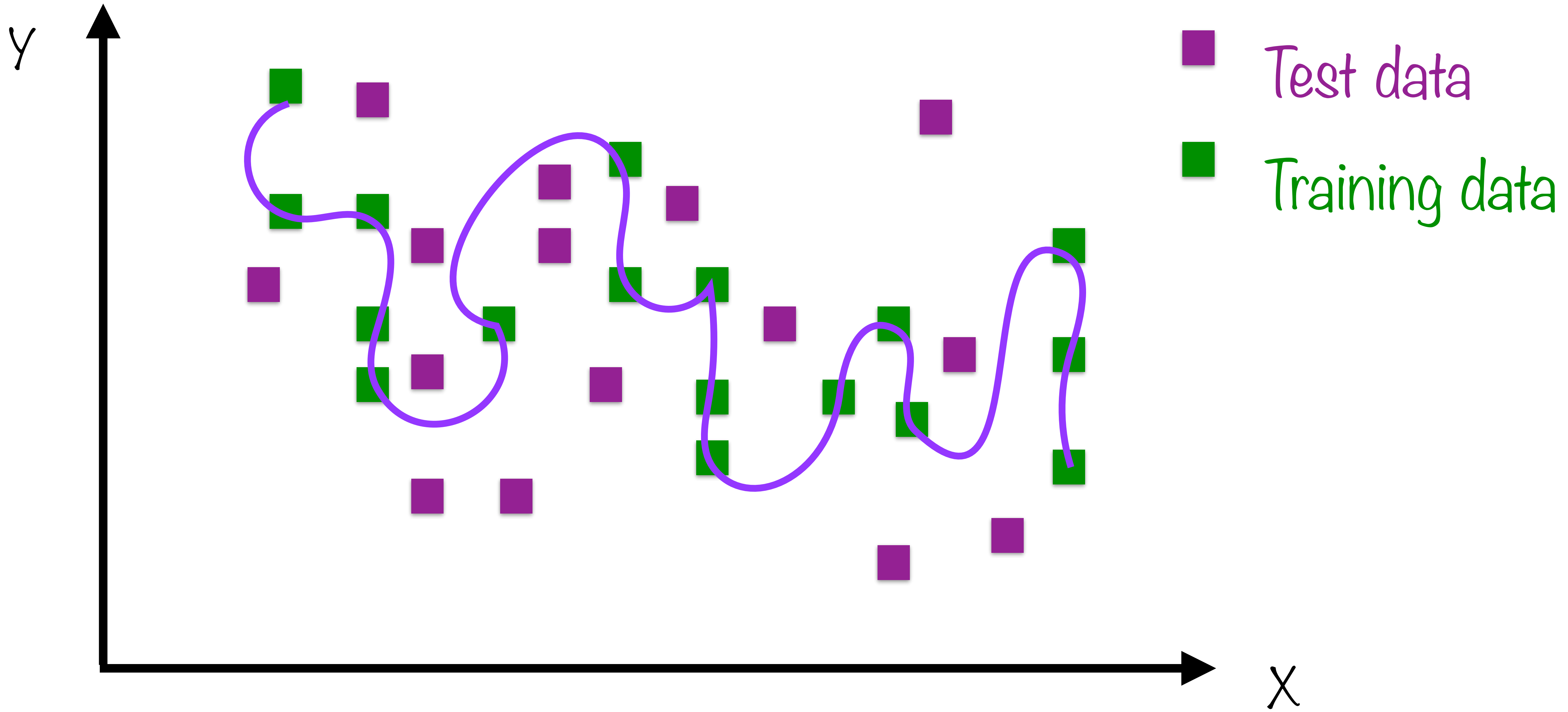
But given a new set of points, this curve might perform quite poorly

Connecting the Dots



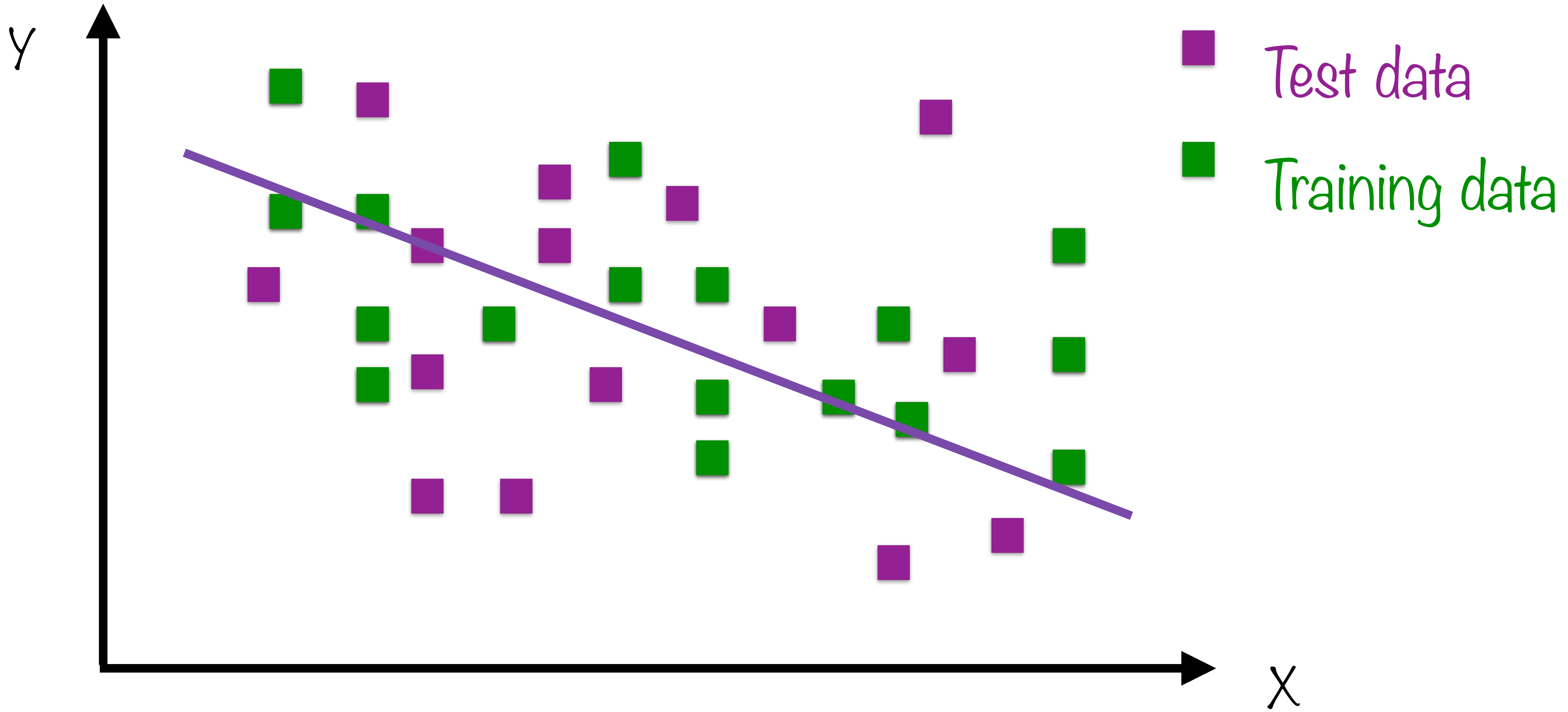
The original points were "training data", the new points are "test data"

Overfitting



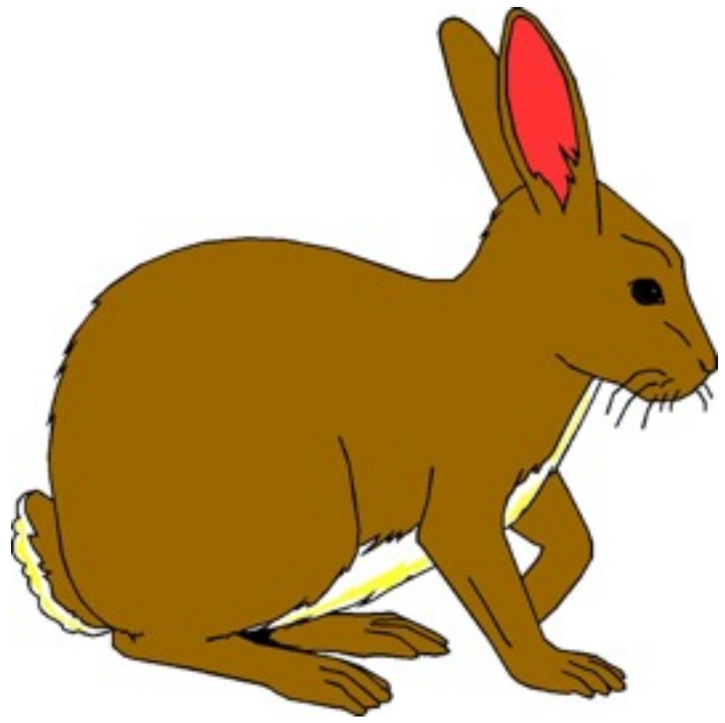
Great performance in training, poor performance in real usage

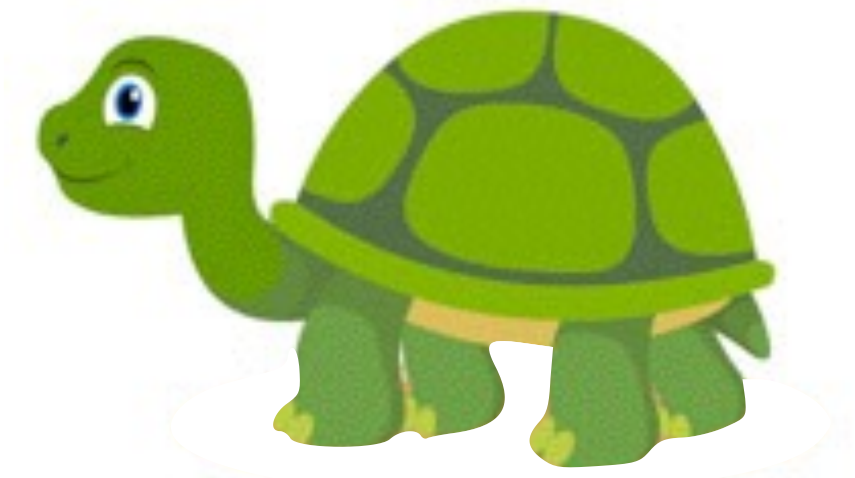
Connecting the Dots



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

Overfitting





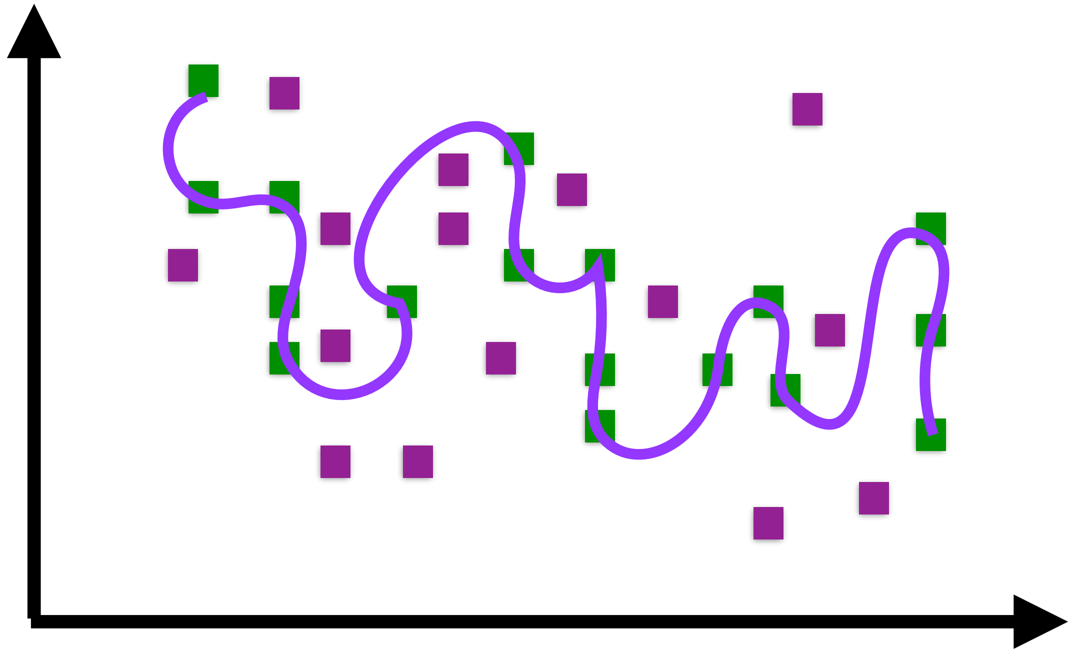
Low Training Error

Model does very well in training...

High Test Error

...but poorly with real data

Cause of Overfitting

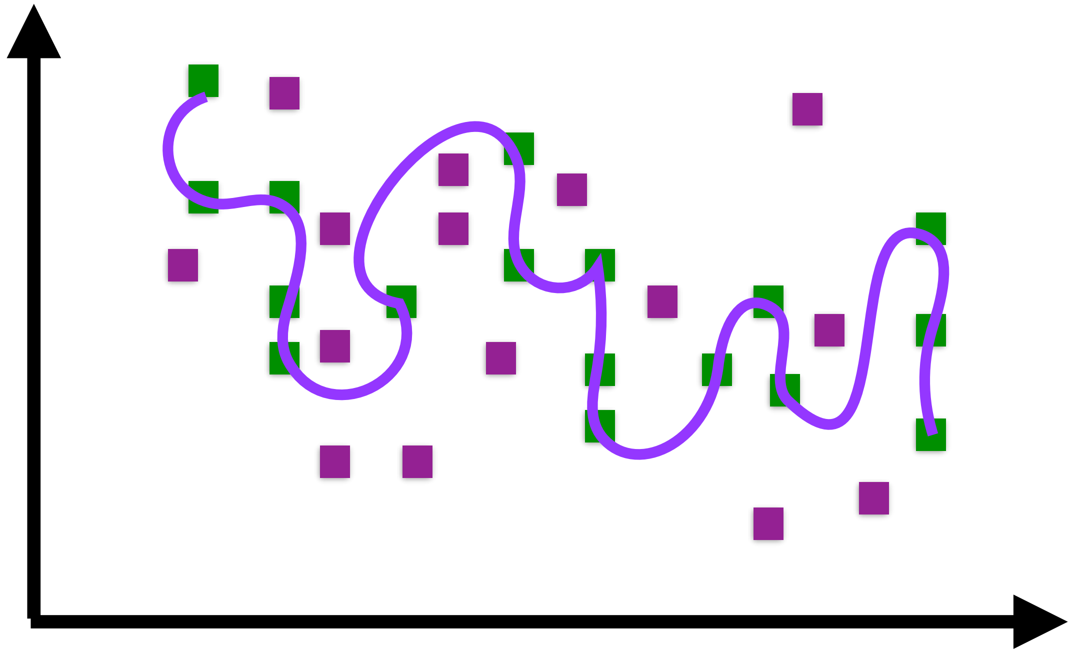


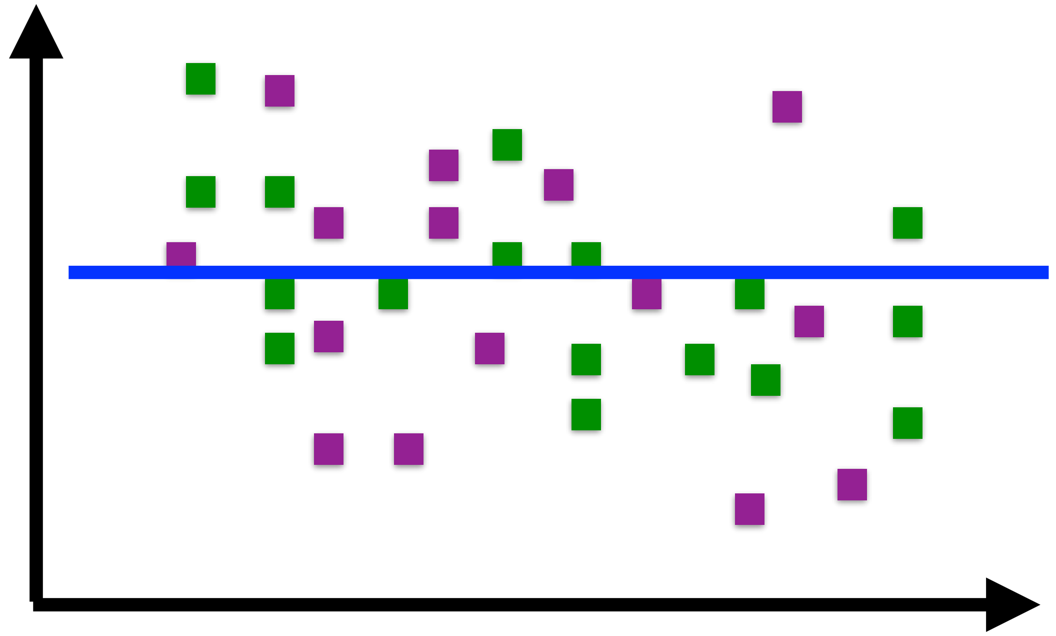
Sub-optimal choice in the bias-variance trade-off

An overfitted model has:

- high variance error
- low bias error

Bias





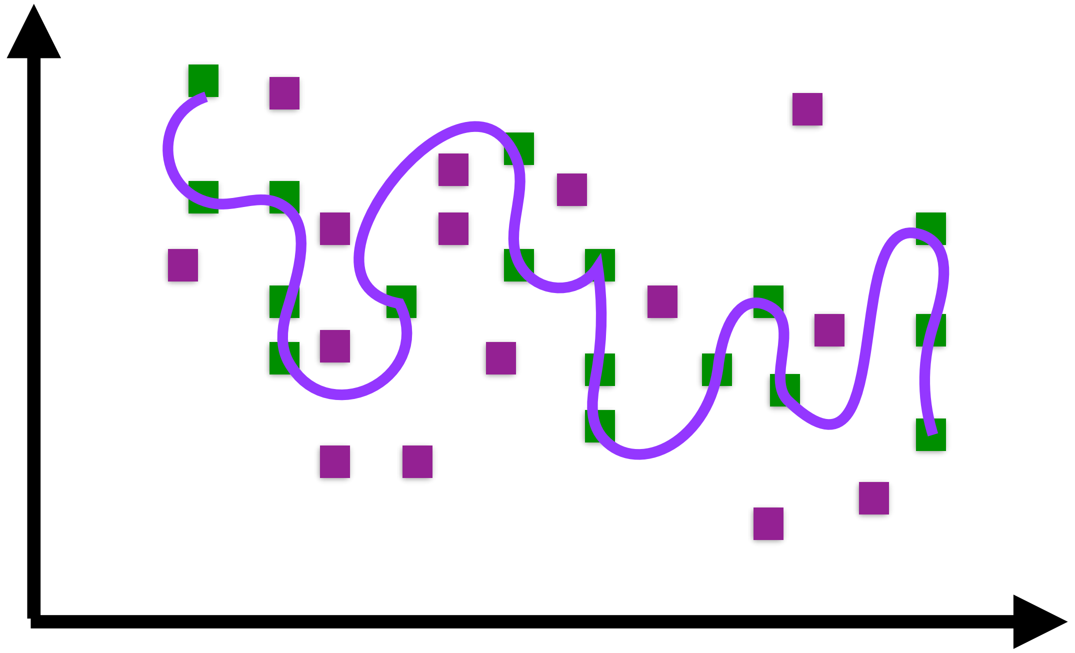
Low bias

Few assumptions about the underlying data

High bias

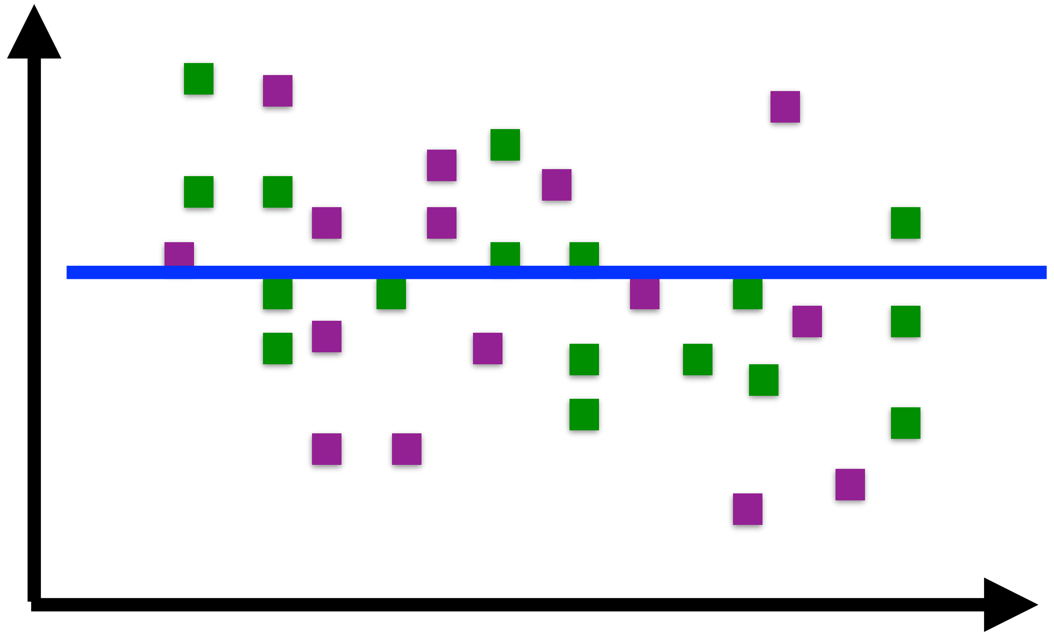
More assumptions about the underlying data

Bias



Model too complex

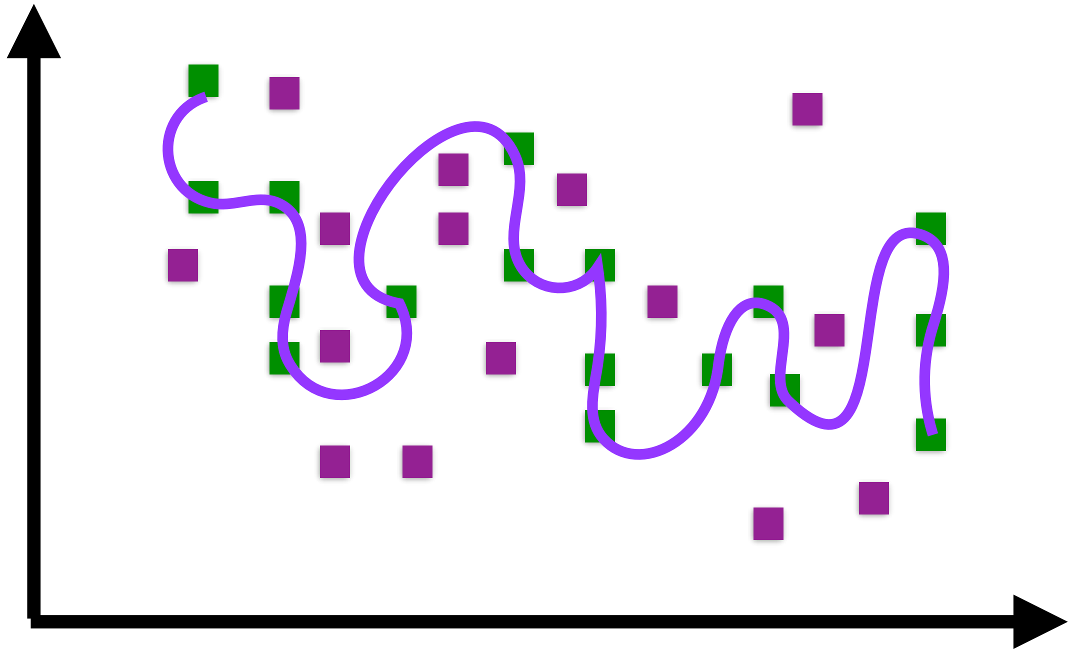
Training data all-important, model parameter counts for little

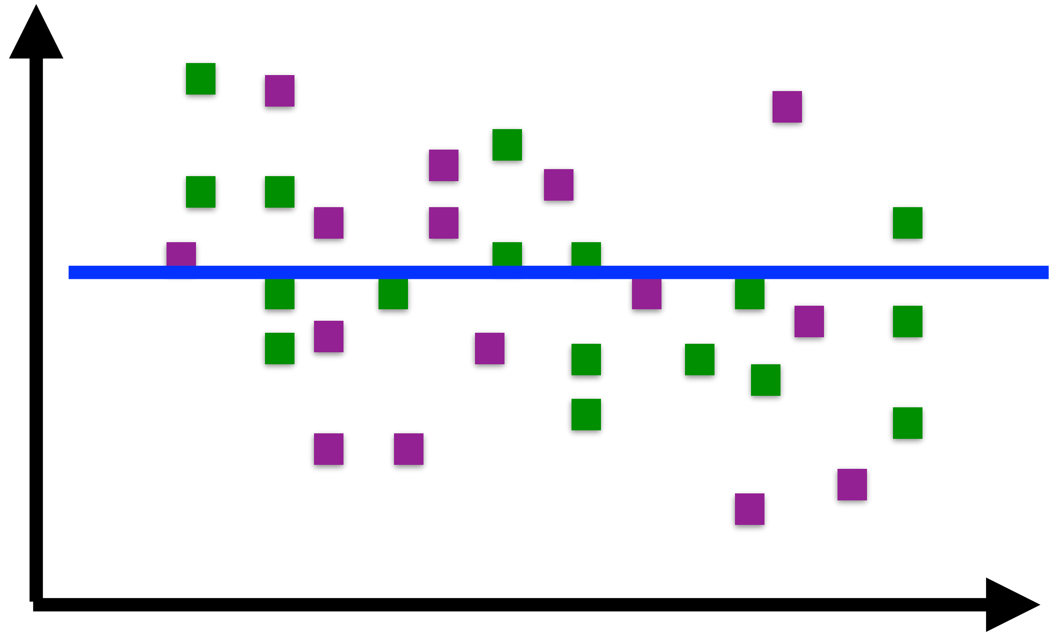


Model too simple

Model parameter all-important, training data counts for little

Variance





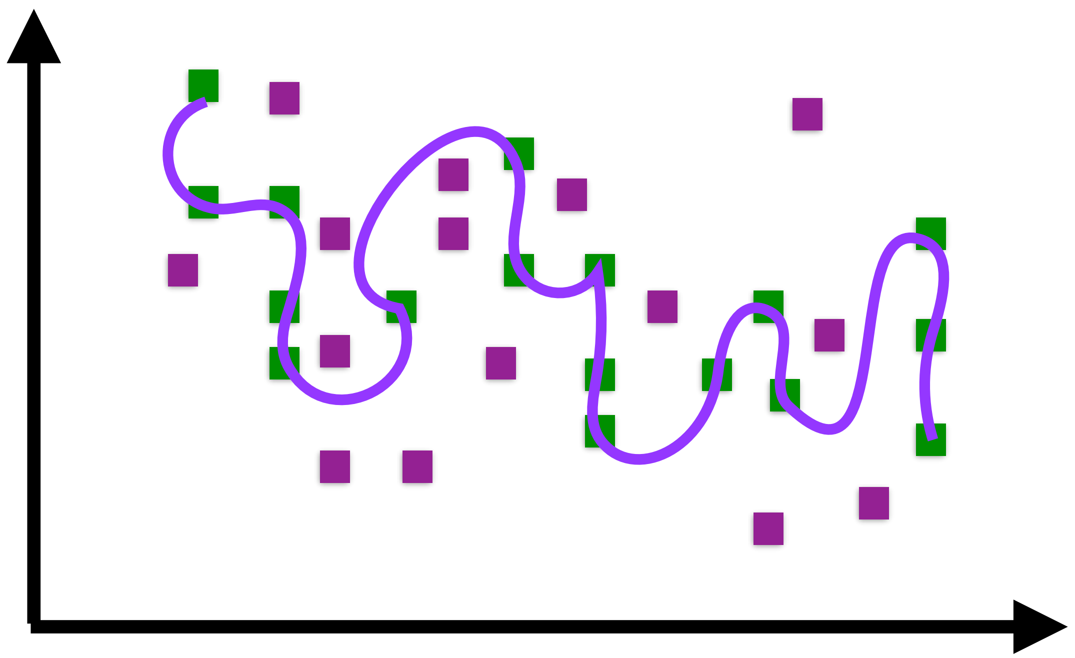
High variance

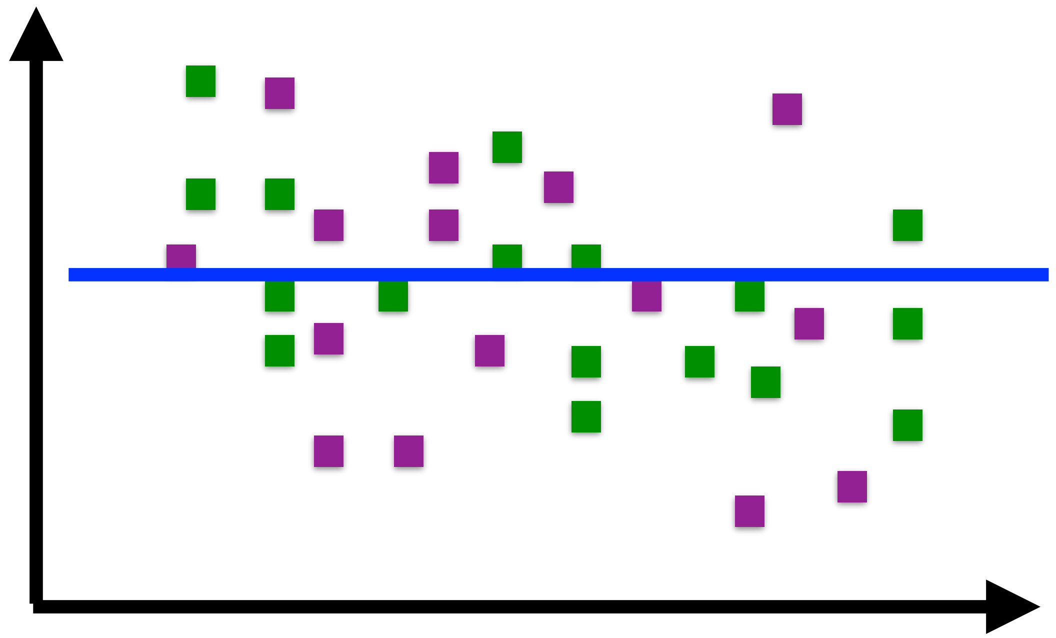
The model changes significantly when training data changes

Low variance

The model doesn't change much when the training data changes

Variance





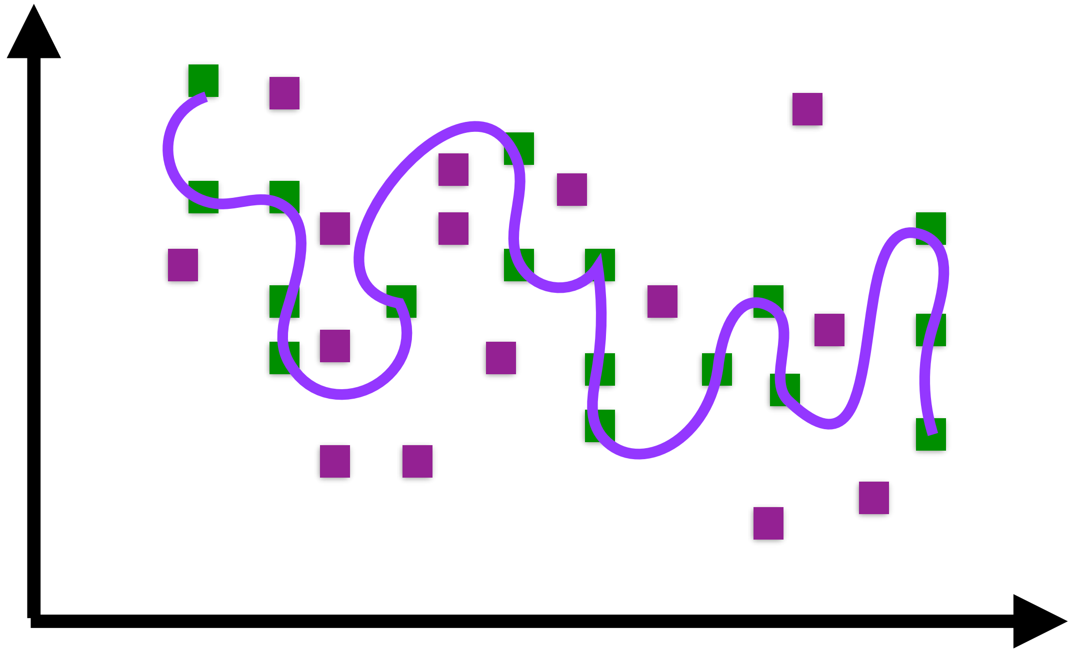
Model too complex

Model varies too much with changing training data

Model too simple

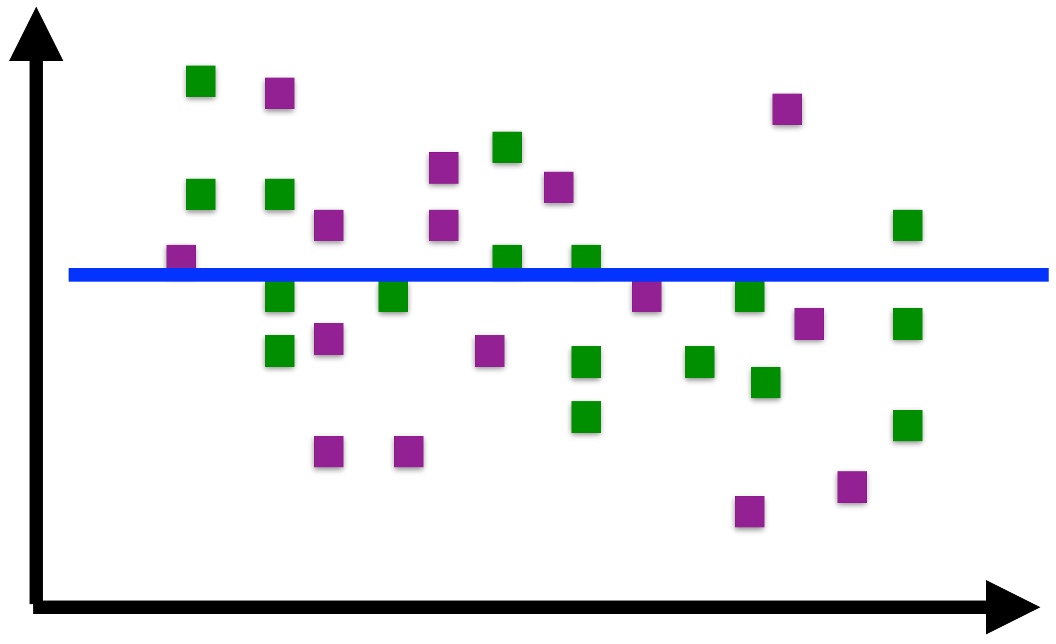
Model not very sensitive to training data

Bias-Variance Trade-off



Model too complex

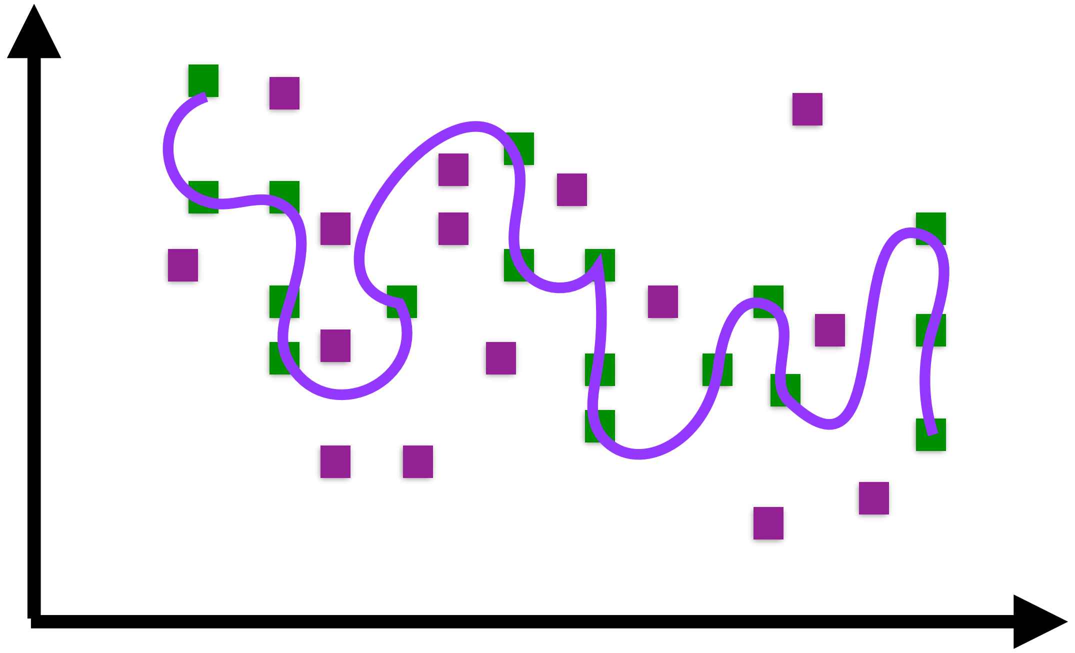
High variance error



Model too simple

High bias error

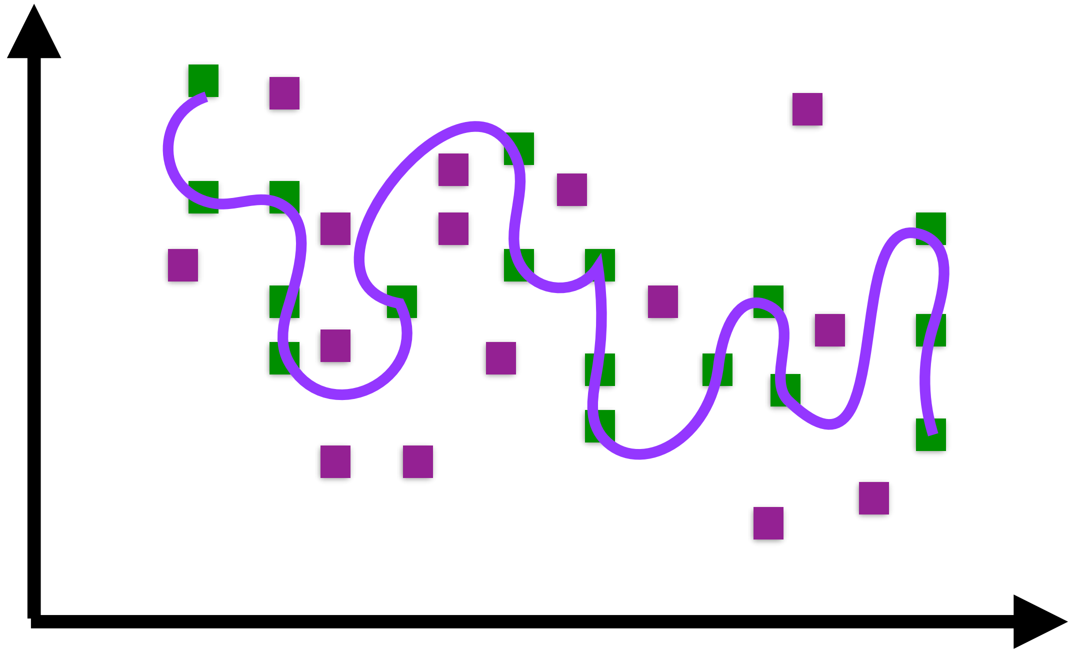
Bias-Variance Trade-off



- High-bias algorithms: simple parameters
    - Regression
- High-variance algorithms: complex parameters
    - Decision trees
    - Dense neural networks

Mitigating Overfitting

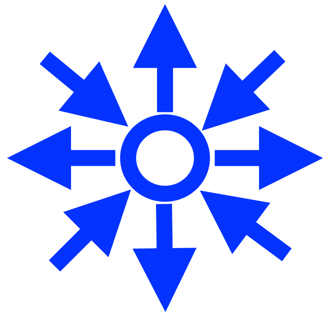
Preventing Overfitting



- Regularisation
- Cross-validation
- Ensemble learning
    - Dropout

Preventing Overfitting

Regularisation - Penalise complex models



Cross-validation - Distinct training and validation phases

Dropout - Intentionally turn off some neurons during training

Regularisation

Penalise complex models

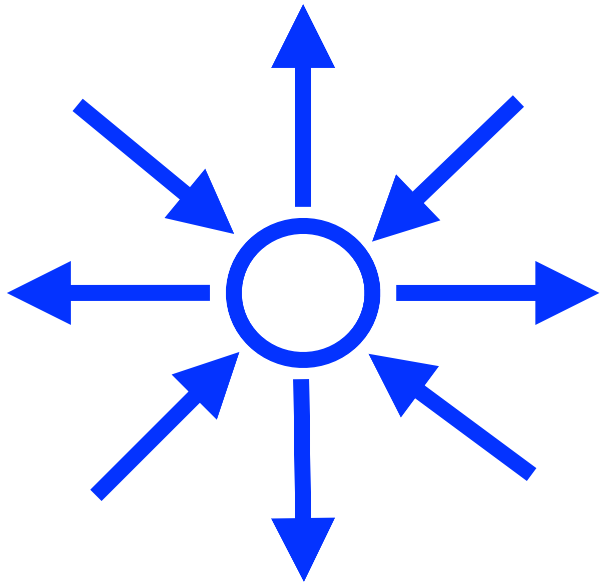
Simple in Gradient Descent

Add penalty to objective function

Penalty as function of neuron weights

Forces optimiser to keep it simple

Cross-Validation

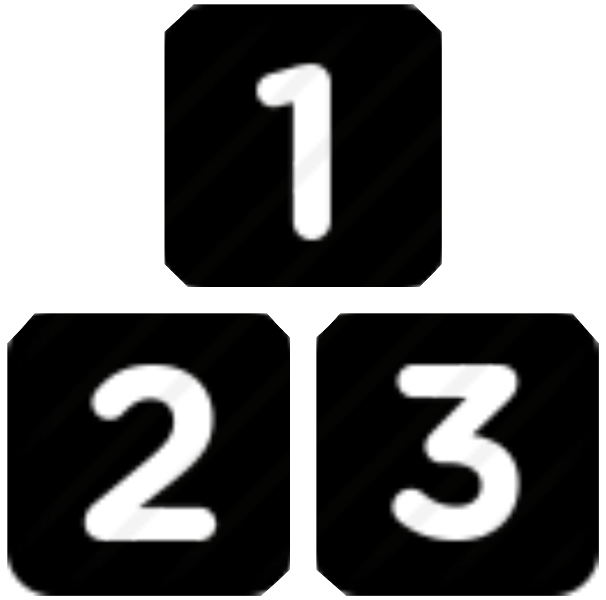


Distinct training and validation phases

Train different models (with training data only)

Select model that does best on validation data

“Hyperparameter tuning”



Dropout

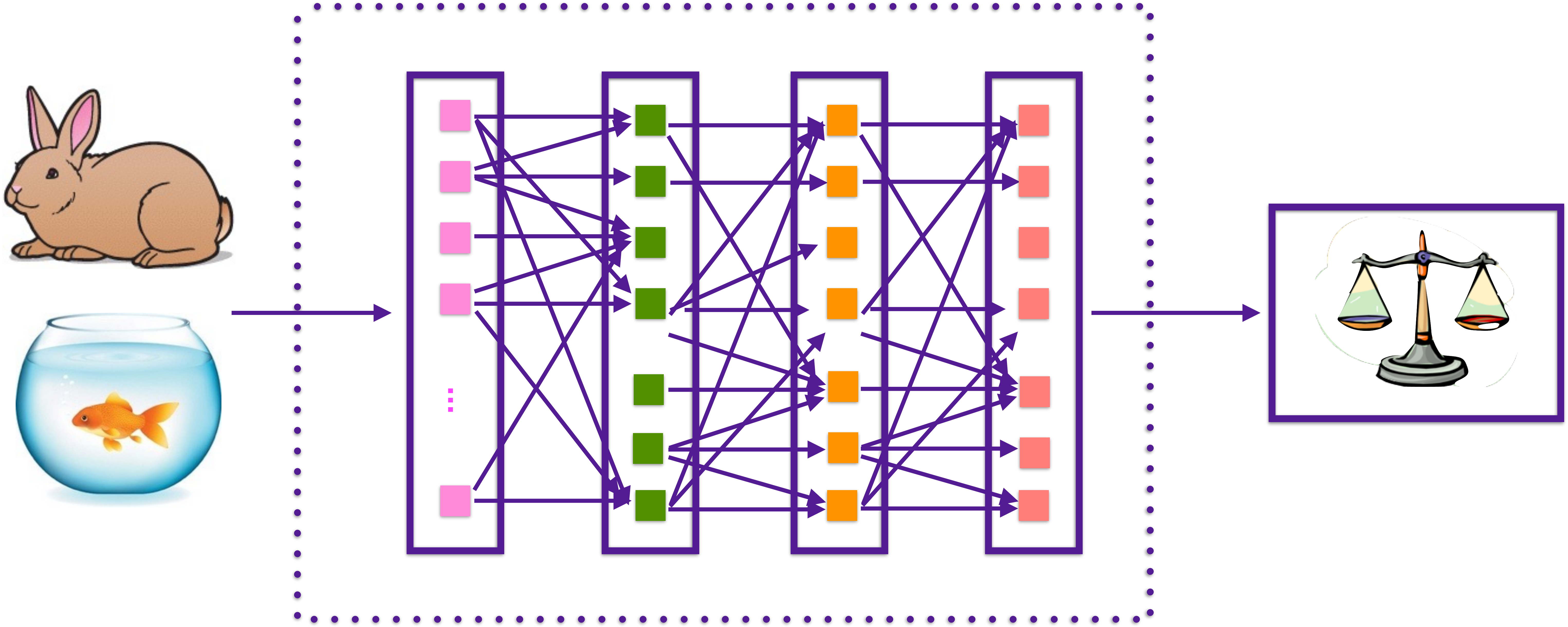
Specify a fraction of neurons that will stay off in each training step

"Dropout" neurons chosen at random

Different neurons off in each training step

In effect, each training step builds different network configuration

Densely Connected Neural Network

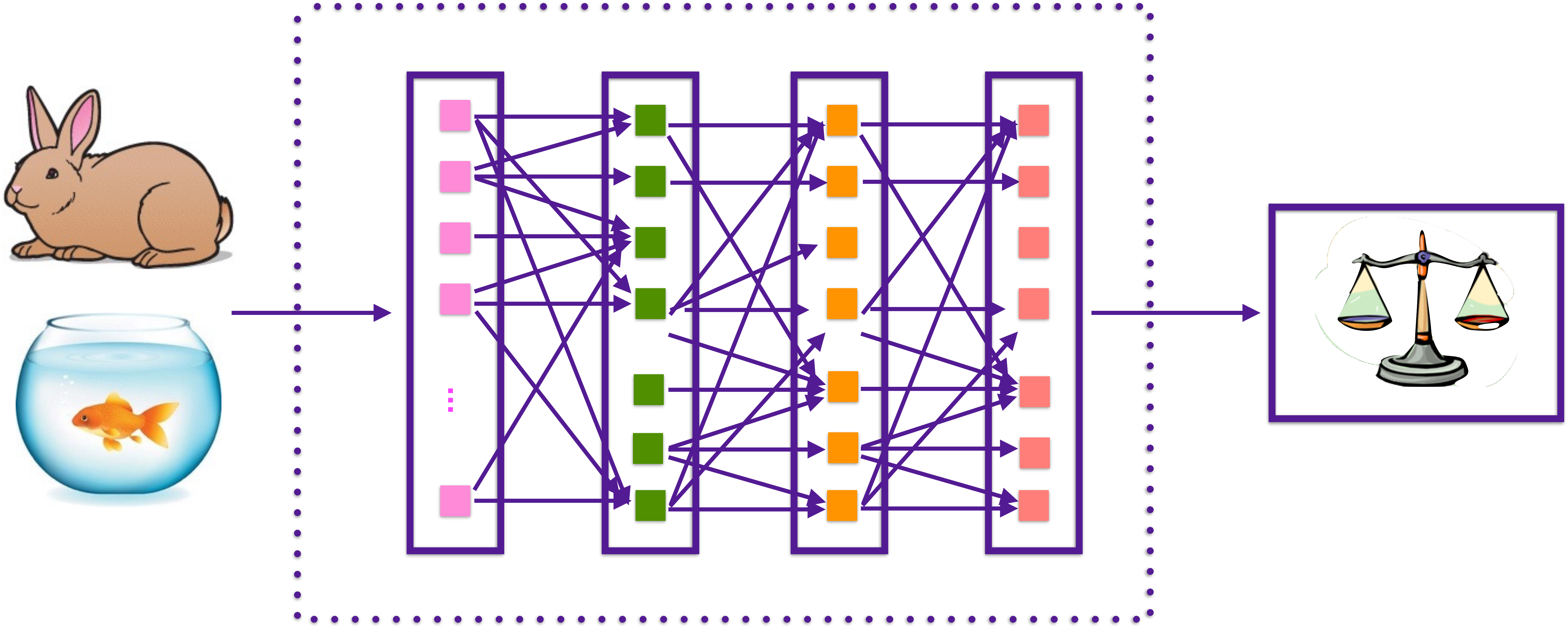


Corpus of Images

High risk of overfitting during training due to dense, complex network

ML-based Classifier

Dropout = 50%

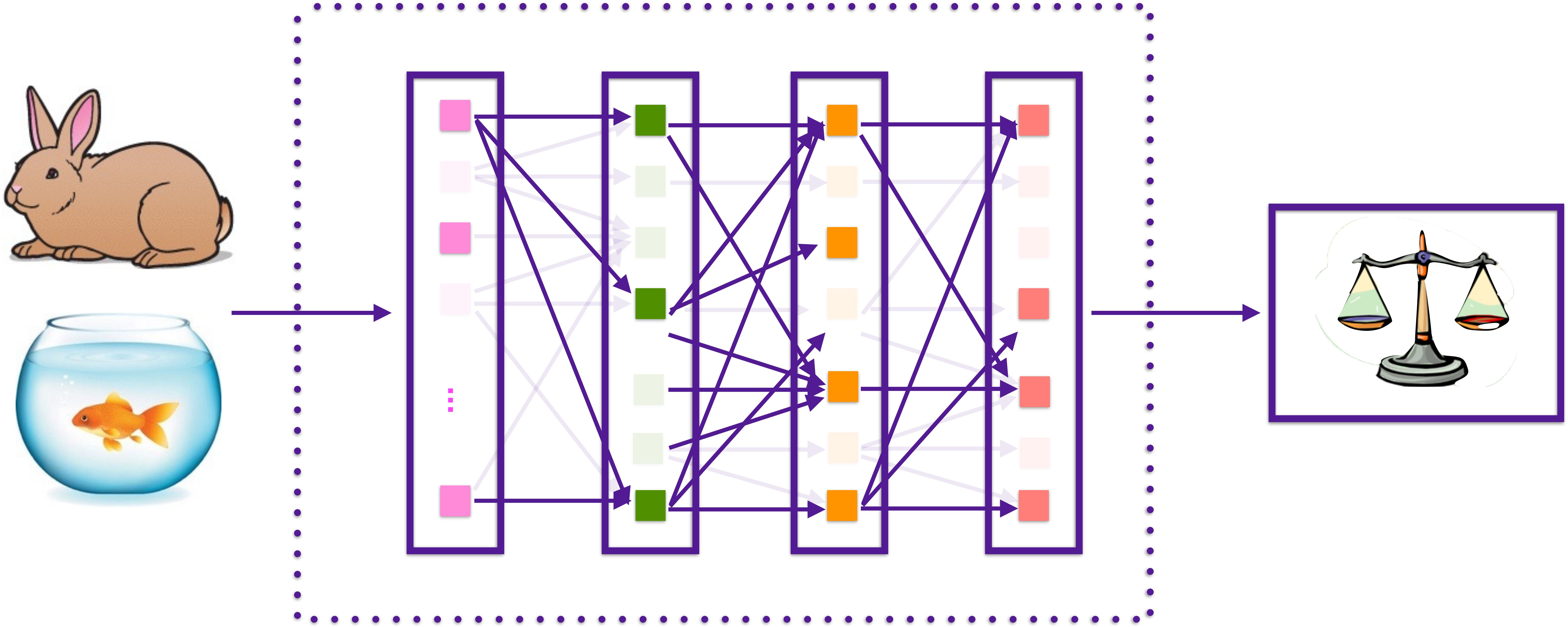


Corpus of Images

Randomly switch off say 50% of neurons in each training step

ML-based Classifier

Dropout = 50%

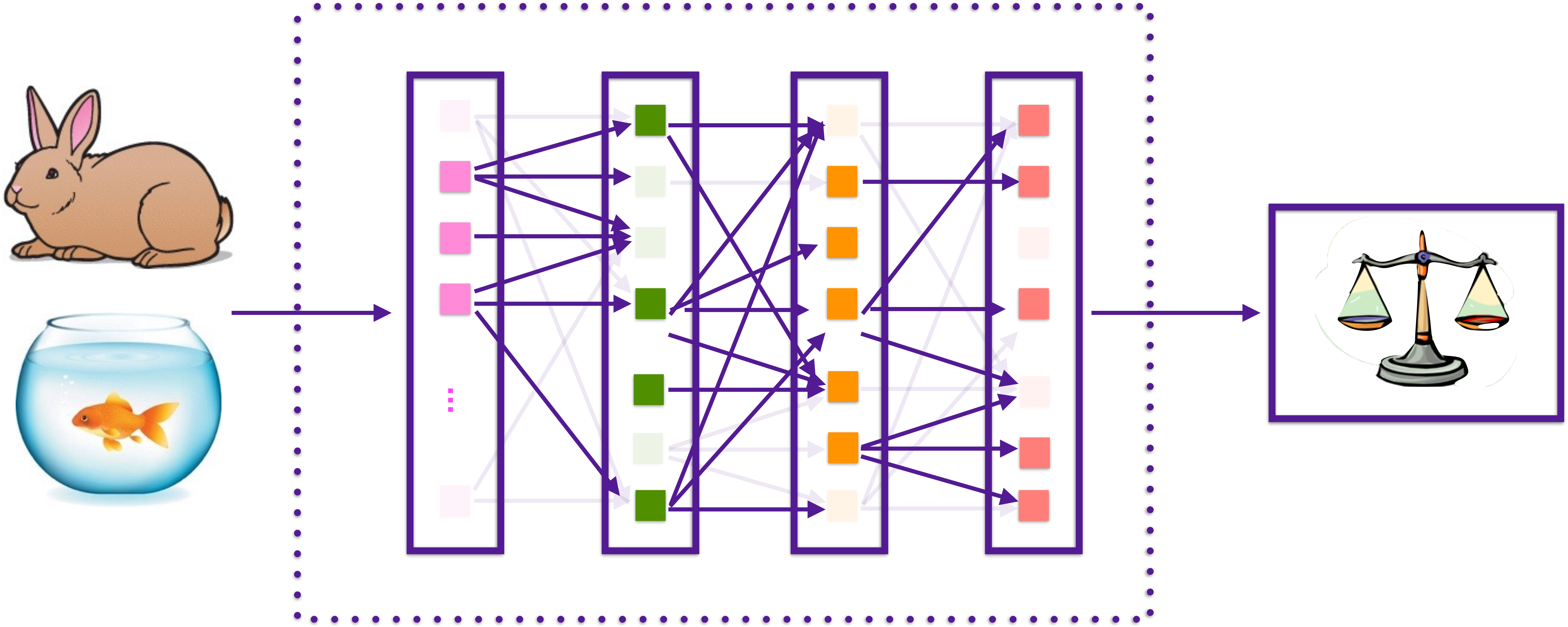


Corpus of Images

Training forced to rely on a much simpler neural network

ML-based Classifier

Dropout = 50%

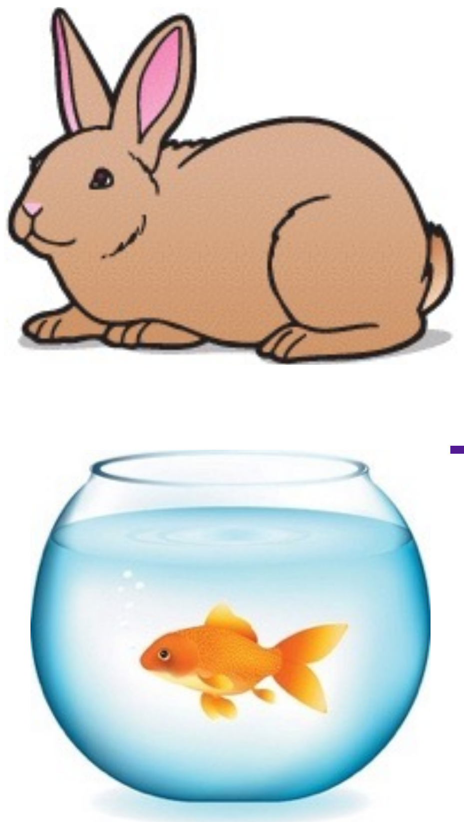


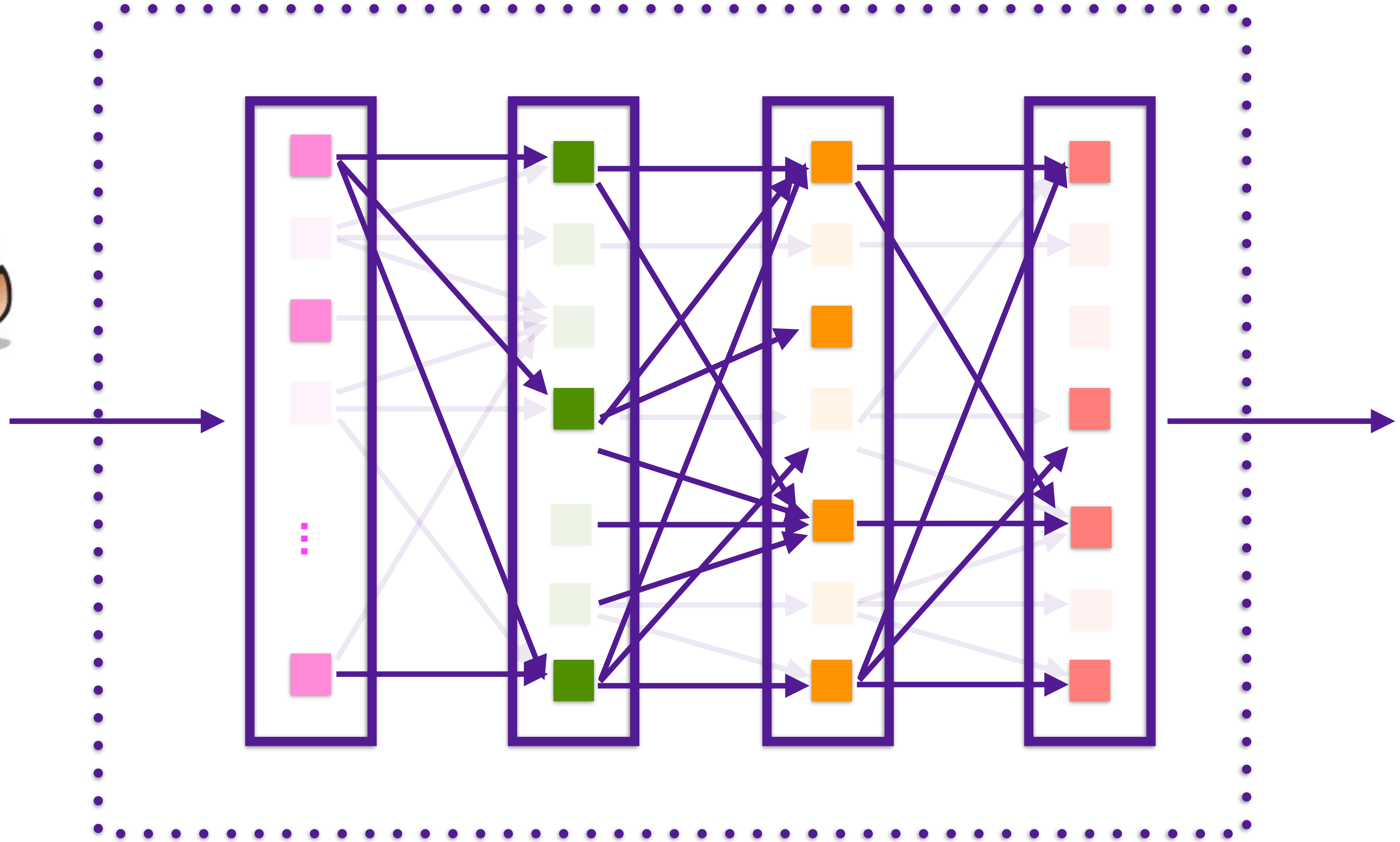
Corpus of Images

Each training step will build a different configuration

ML-based Classifier

Dropout = 50%



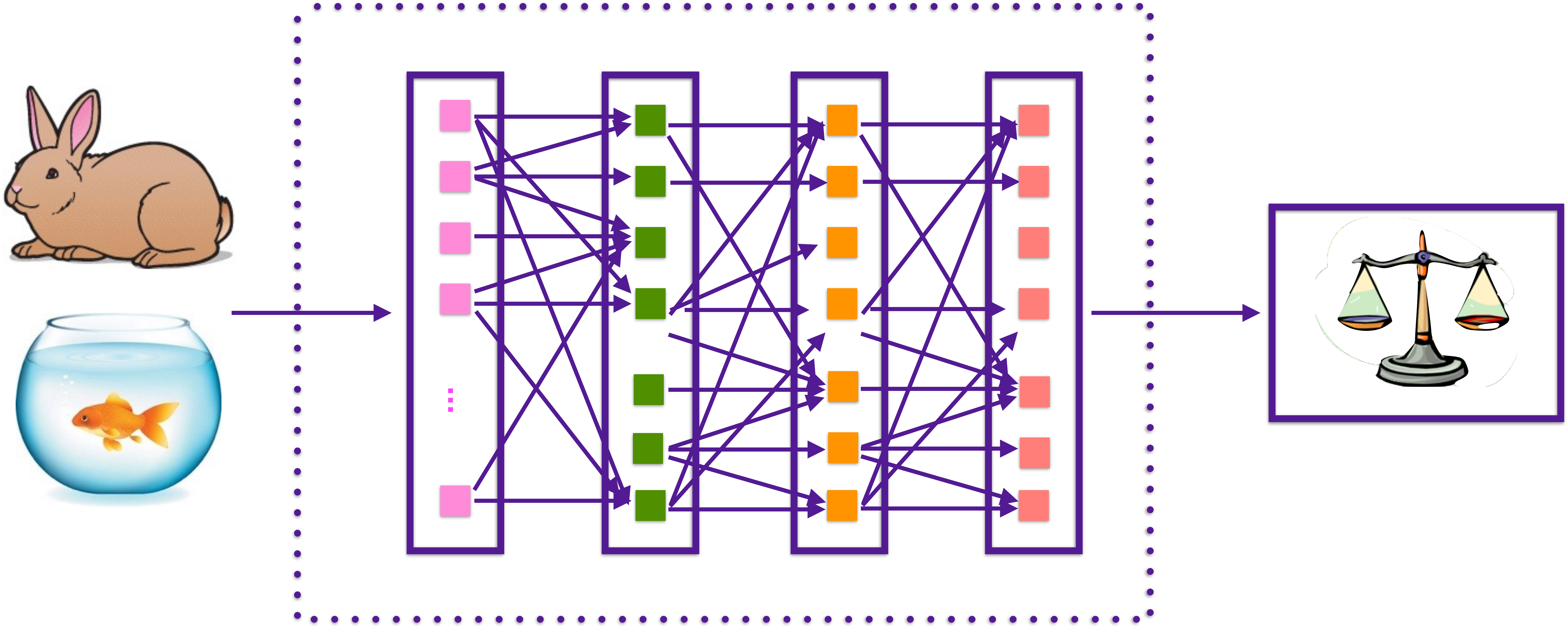


Corpus of Images

Each training step will build a different configuration

ML-based Classifier

Dropout During Training Only



Corpus of Images

During actual usage in test mode, full dense neural network is used

ML-based Classifier