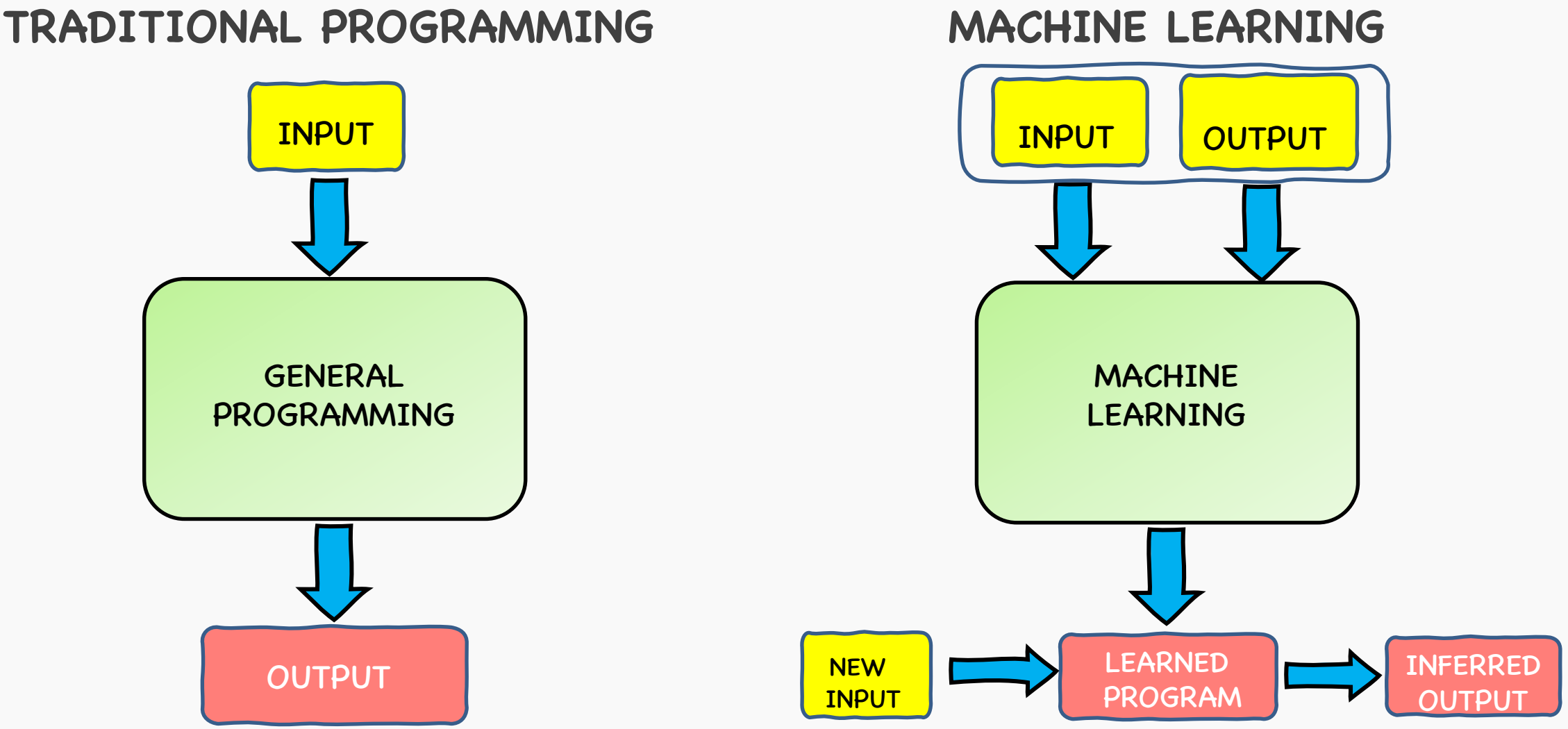
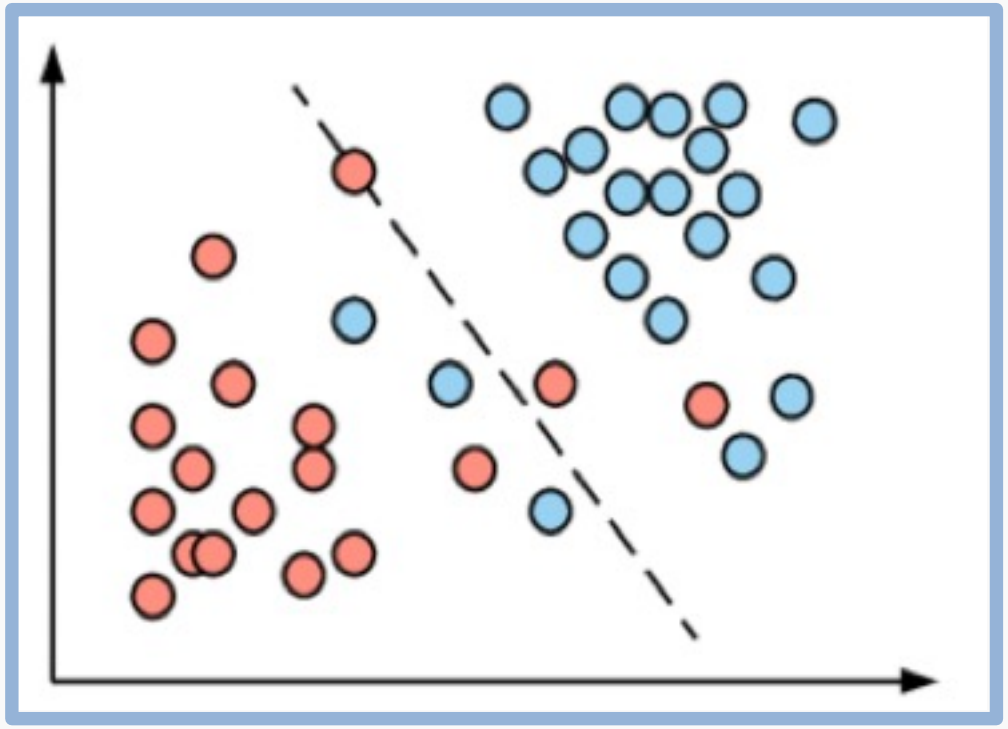
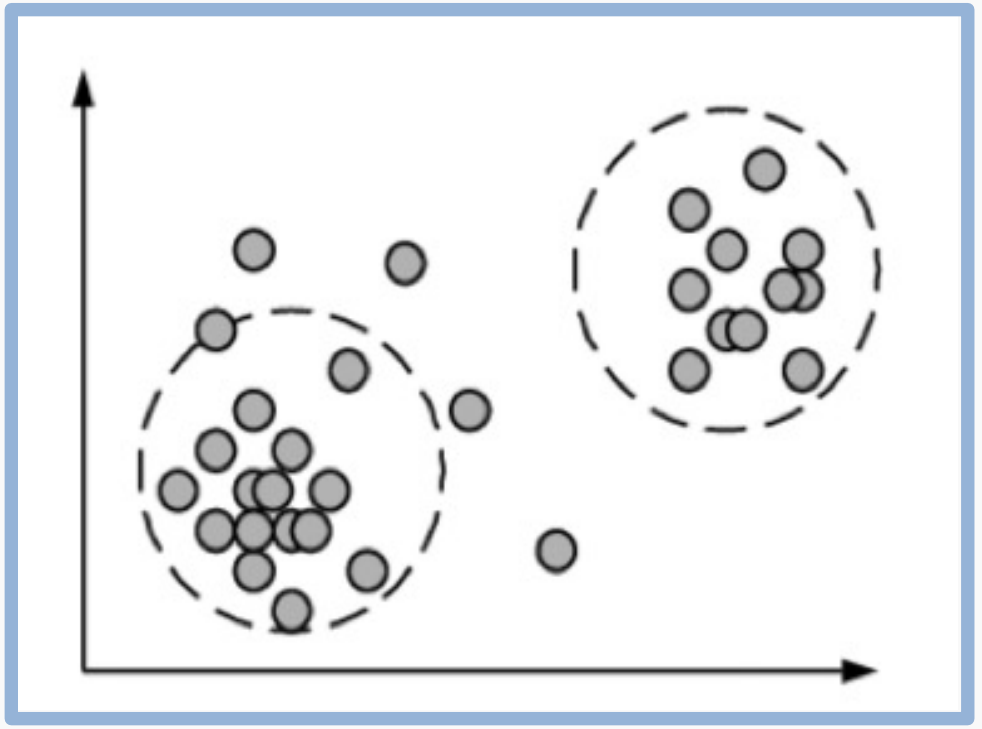
Machine Learning

What is Machine Learning?



Supervised v/s Unsupervised Machine Learning

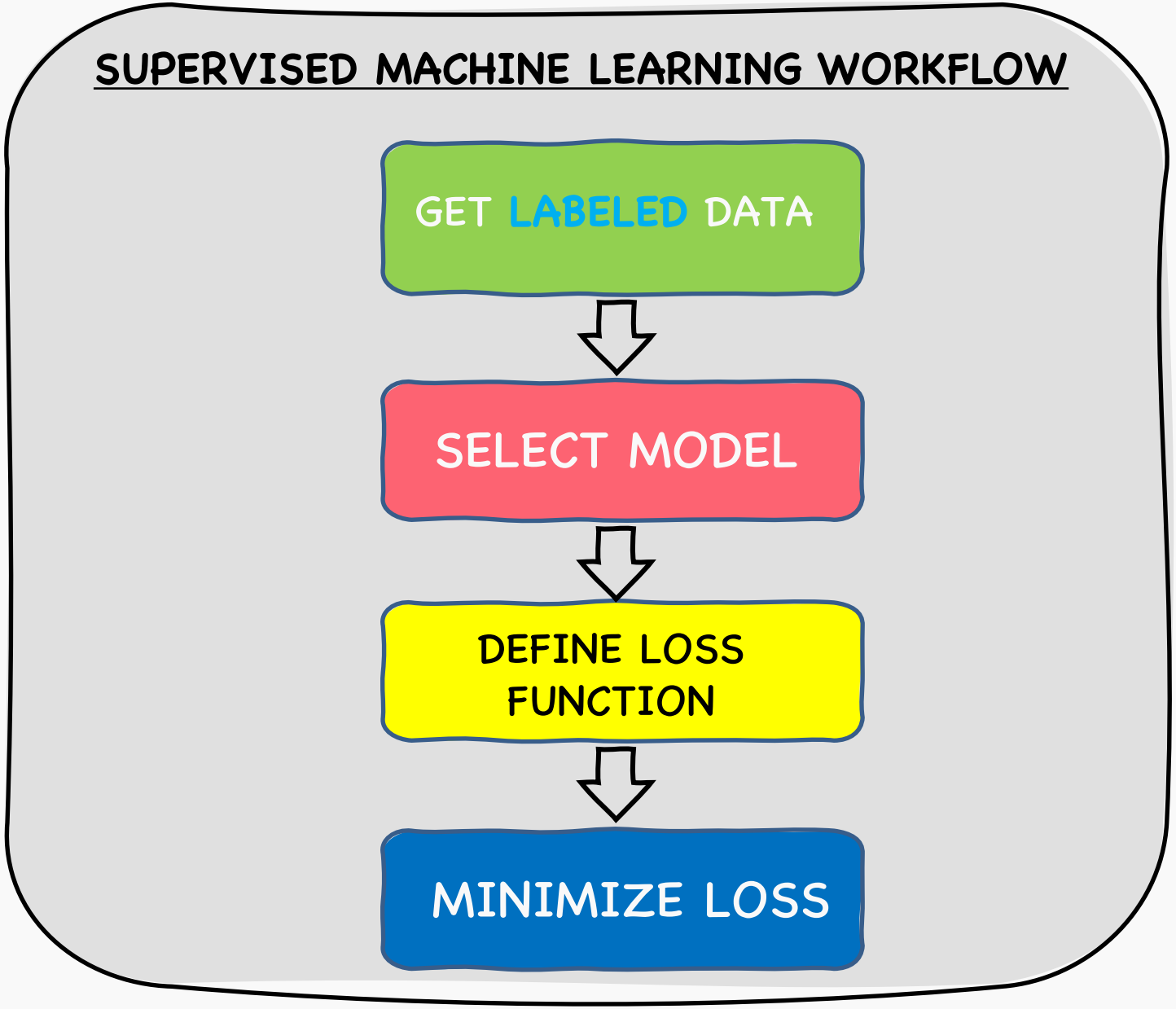




Supervised Learning: Learns with “labeled” data

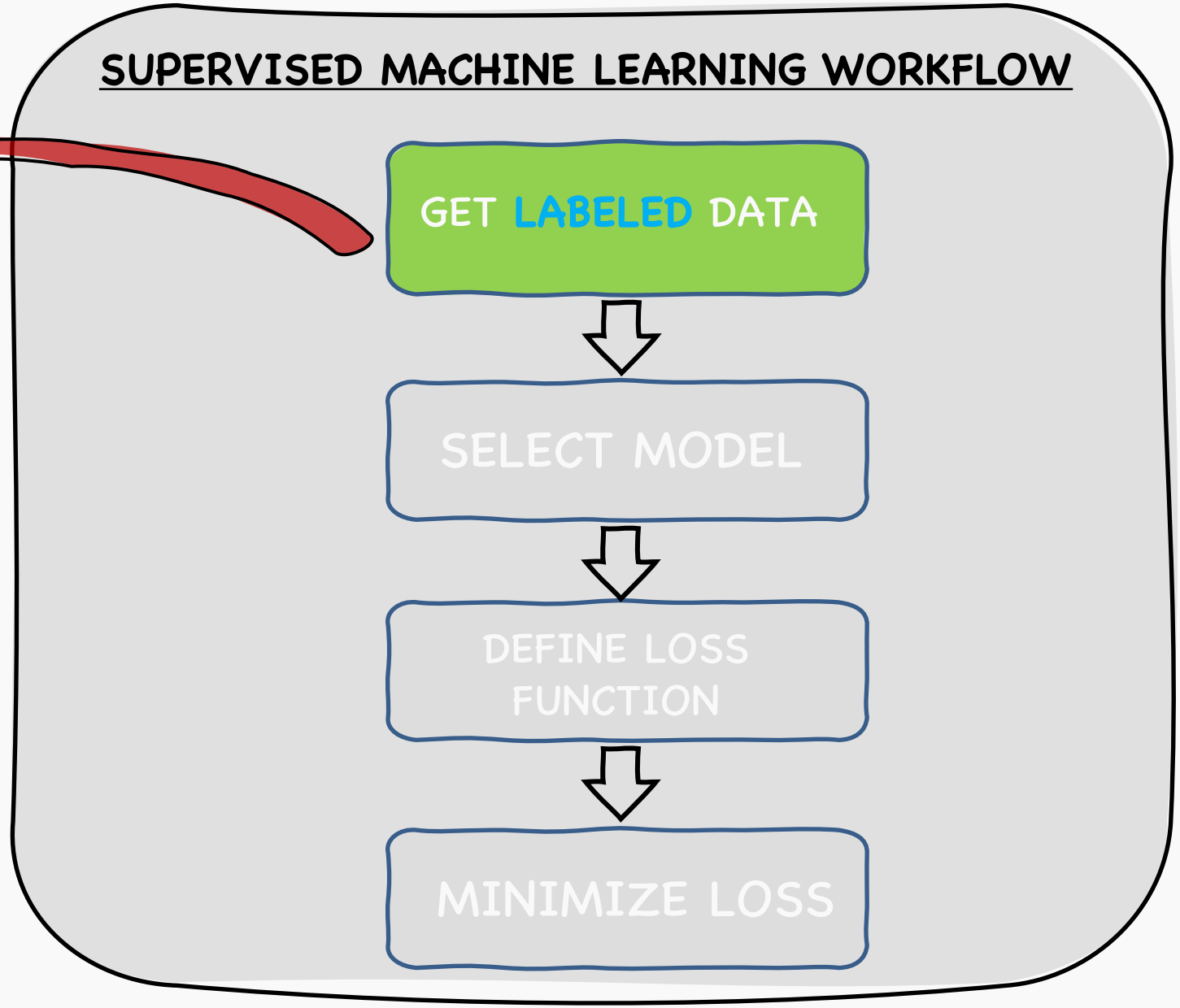
Unsupervised Learning: Learns by clustering or association

Building blocks of **supervised** machine learning

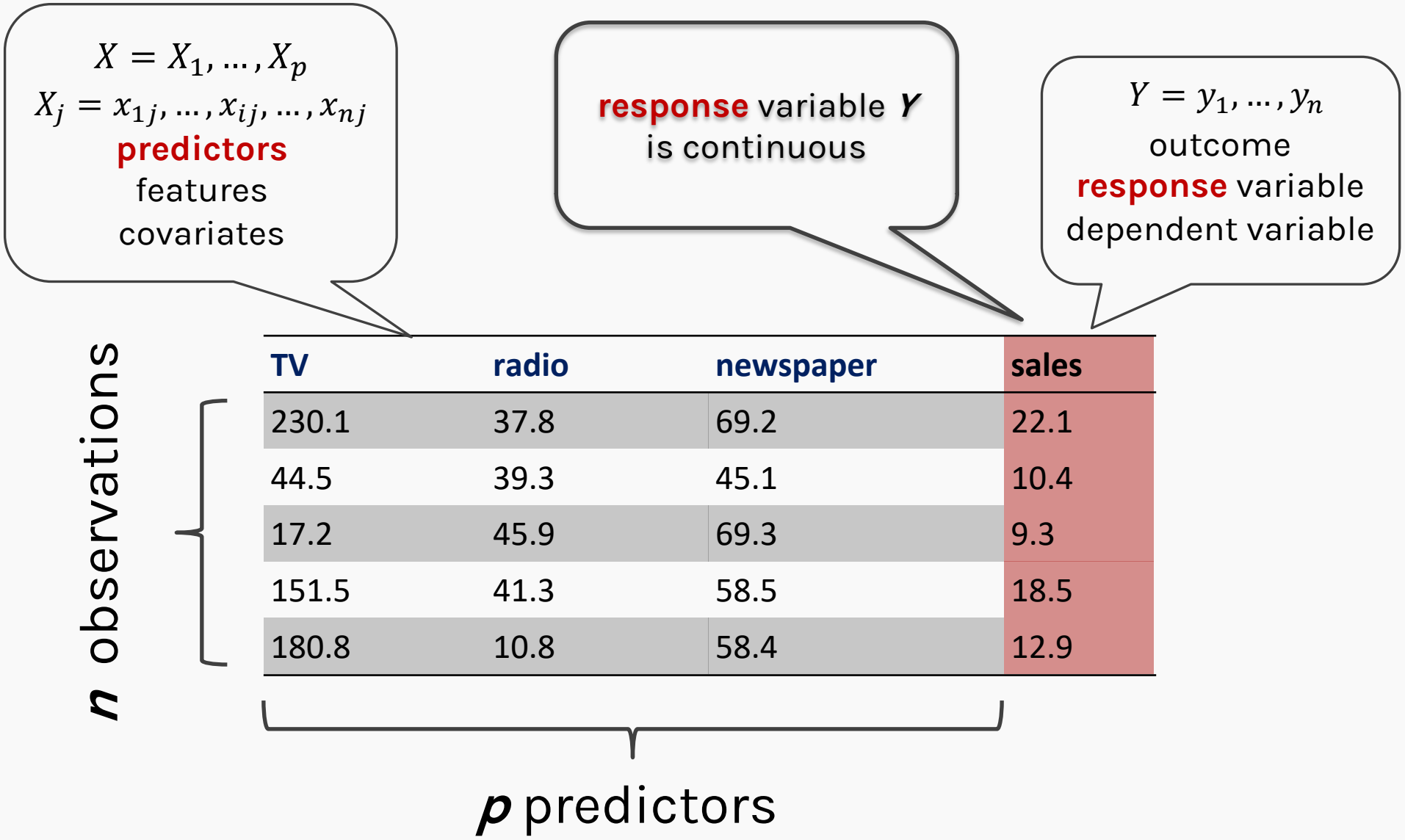


Building blocks of **supervised** machine learning

Sales vs Advertising budget,
Patient history vs heart disease
(Response & Predictor variables)



Response vs. Predictor Variables



TV	radio	newspaper	sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

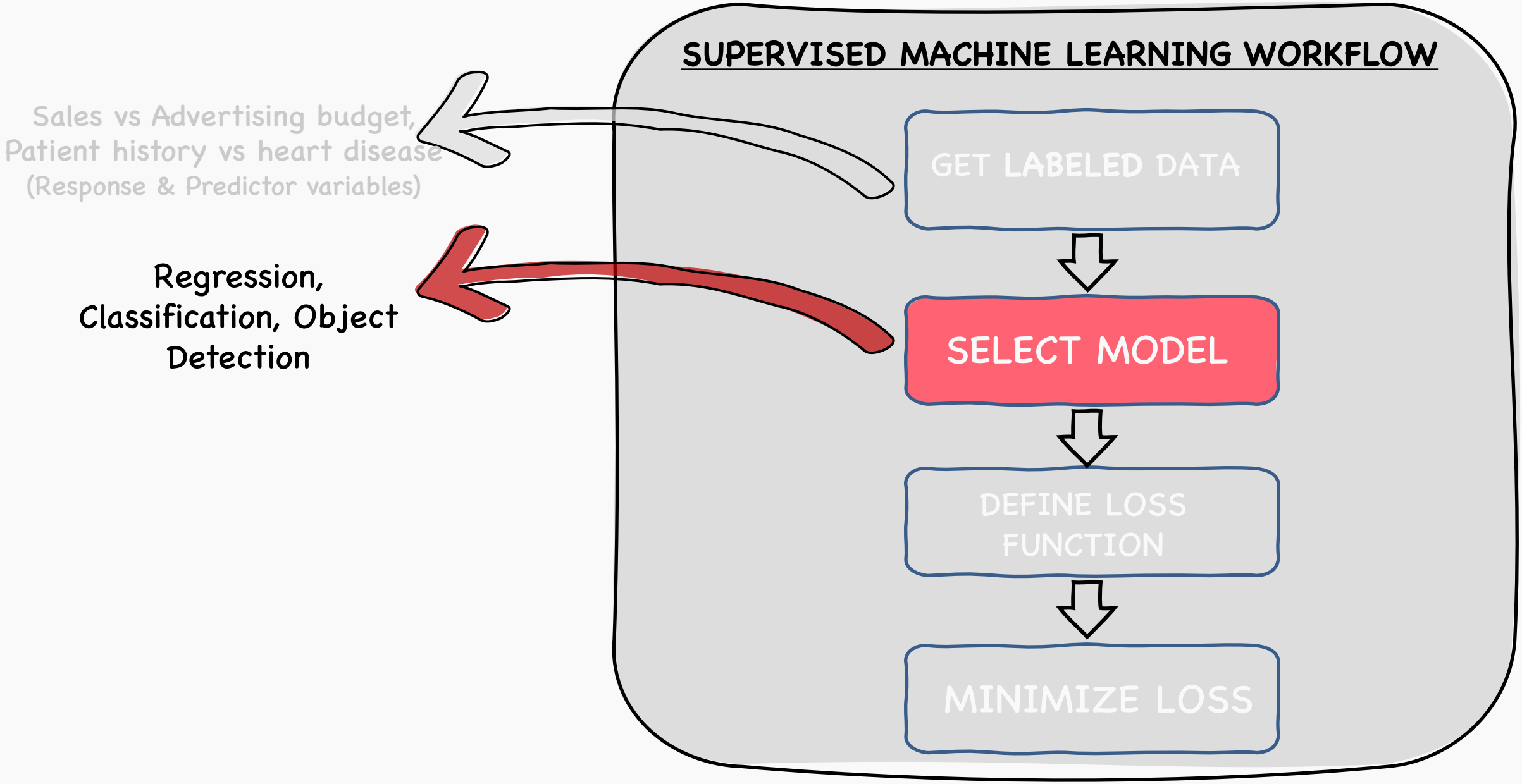
Heart Data

These data contain a binary outcome AHD for 303 patients who presented with chest pain.

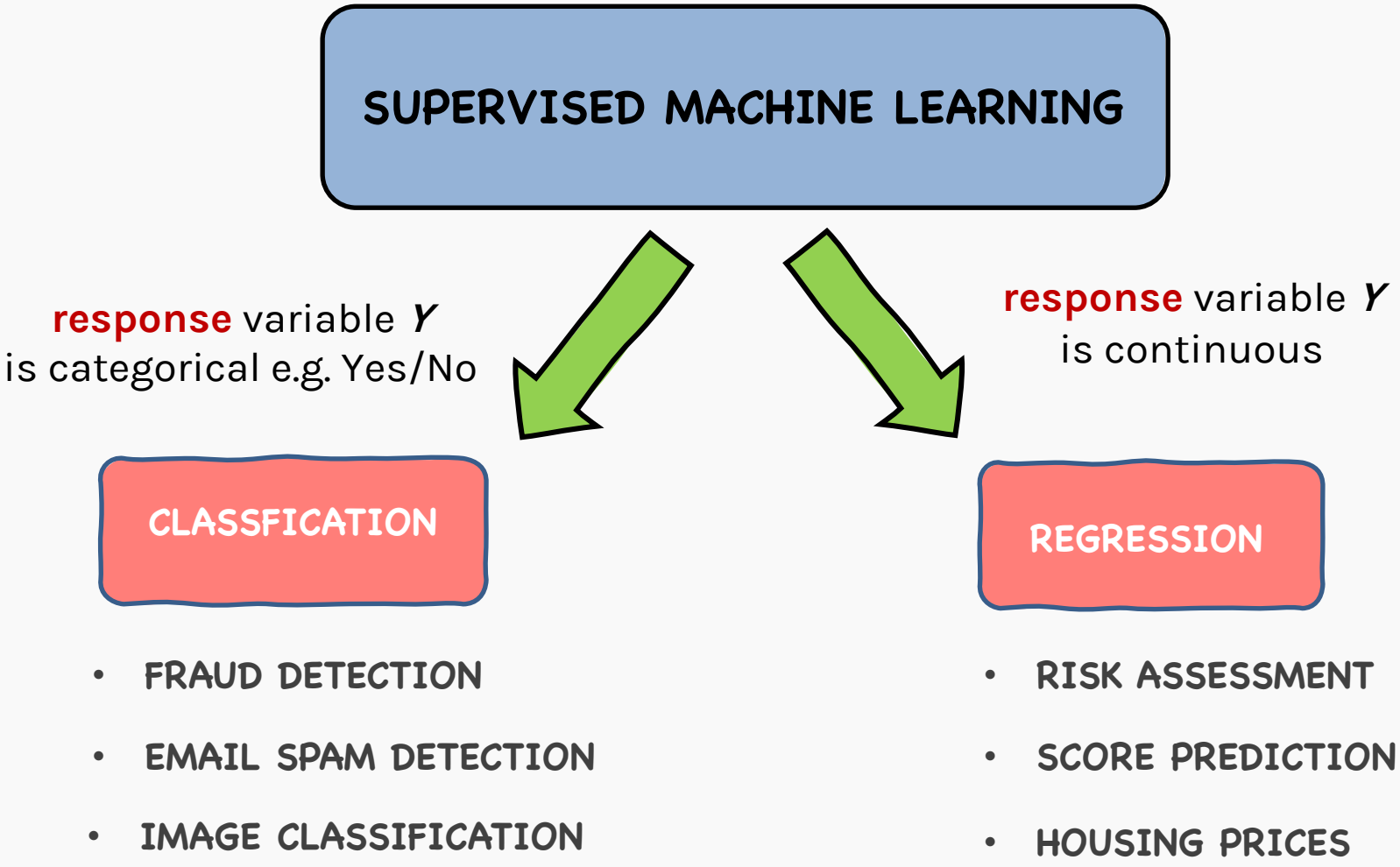
response variable ***Y*** is Yes/No

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

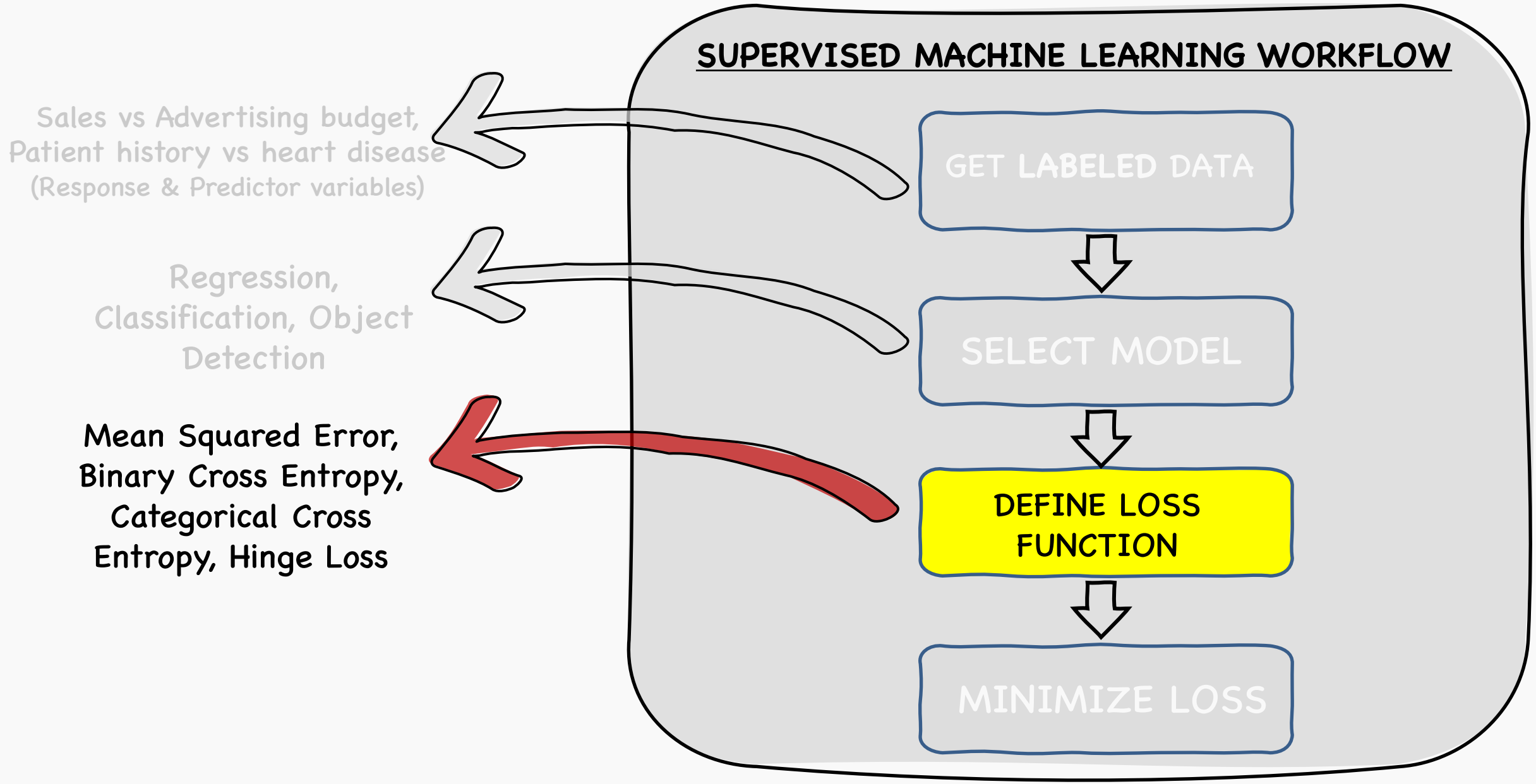
Building blocks of **supervised** machine learning



Supervised Machine Learning examples



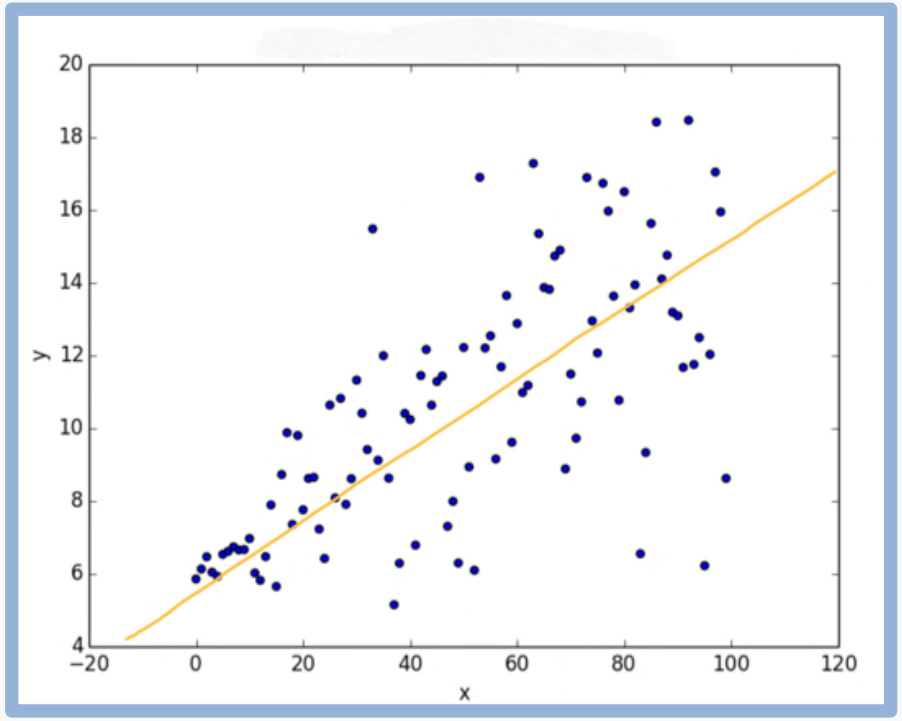
Building blocks of **supervised** machine learning



Loss for linear regression

MSE as the **loss function**,

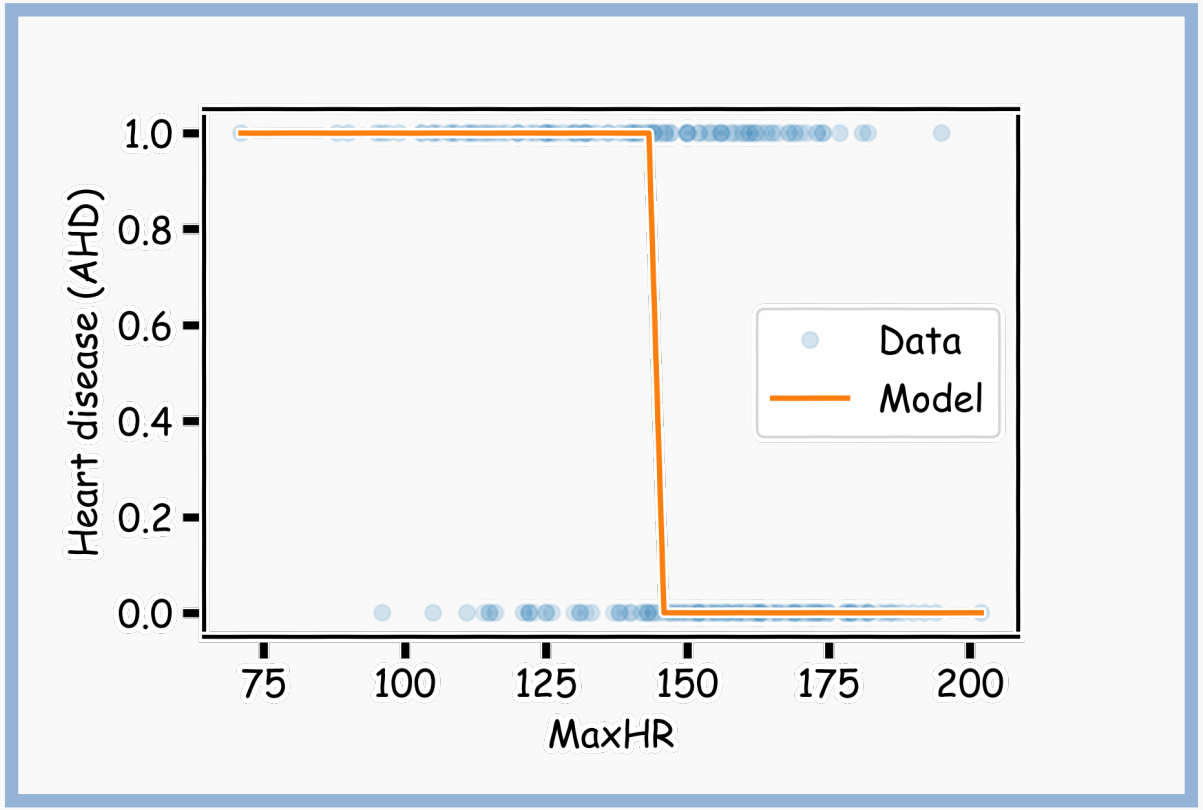
$$L(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - (\beta_1 X + \beta_0)\right]^2 .$$



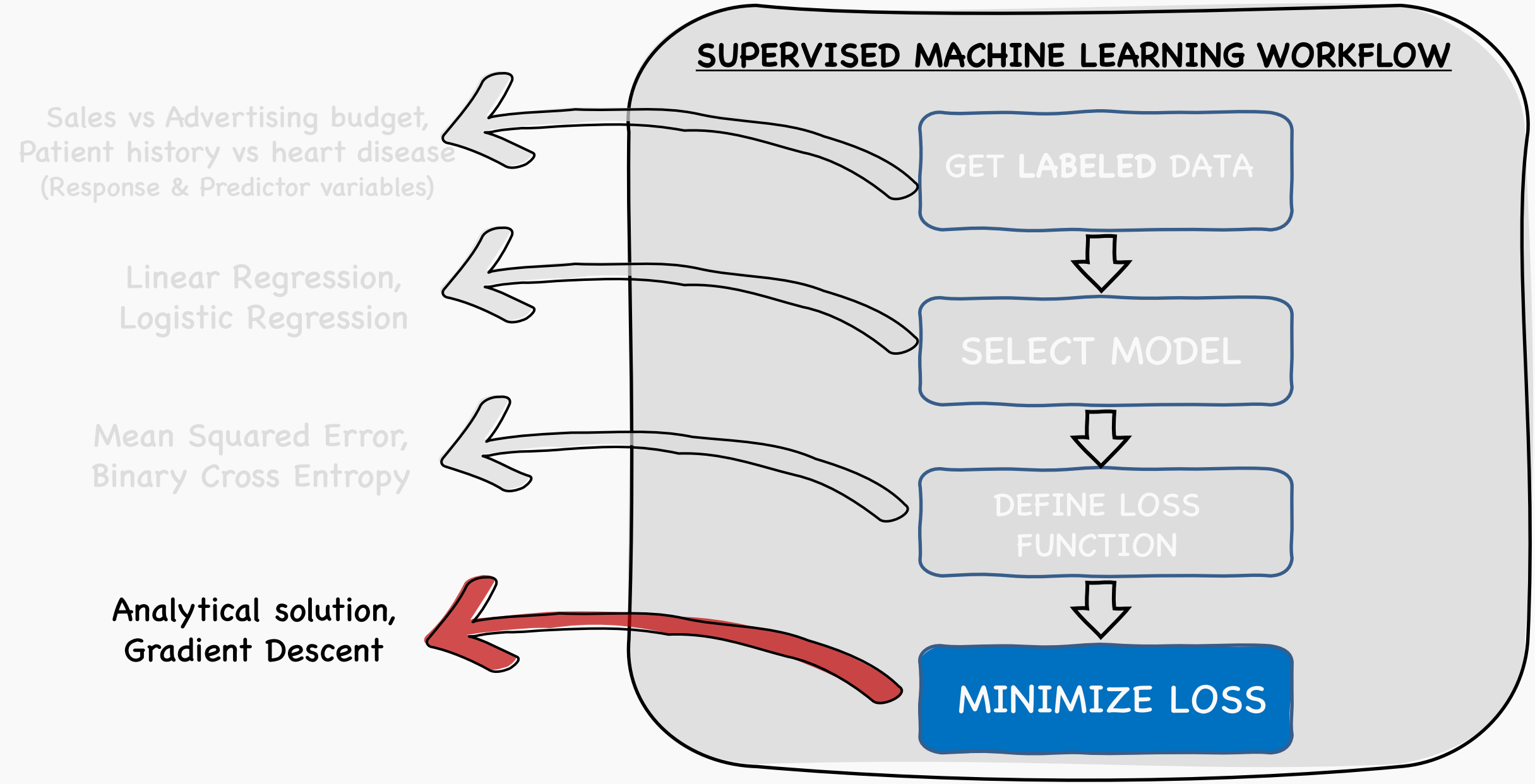
Loss function for Logistic Regression

Cross Entropy as a loss function

$$\mathcal{L}(\beta_0, \beta_1) = -\sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

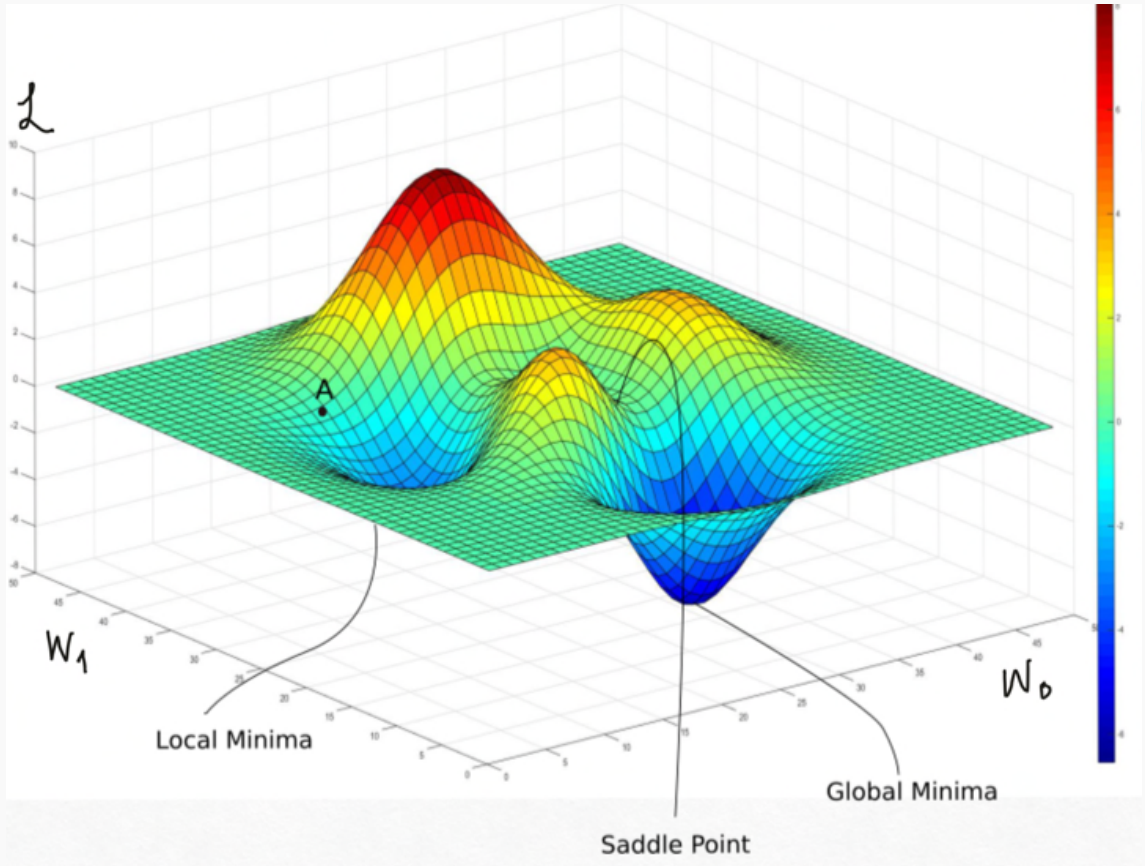


Building blocks of **supervised** machine learning



Optimization

How does one minimize a loss function?



Minima or maxima of $L(\beta_0, \beta_1)$ must occur at points where the gradient (slope)

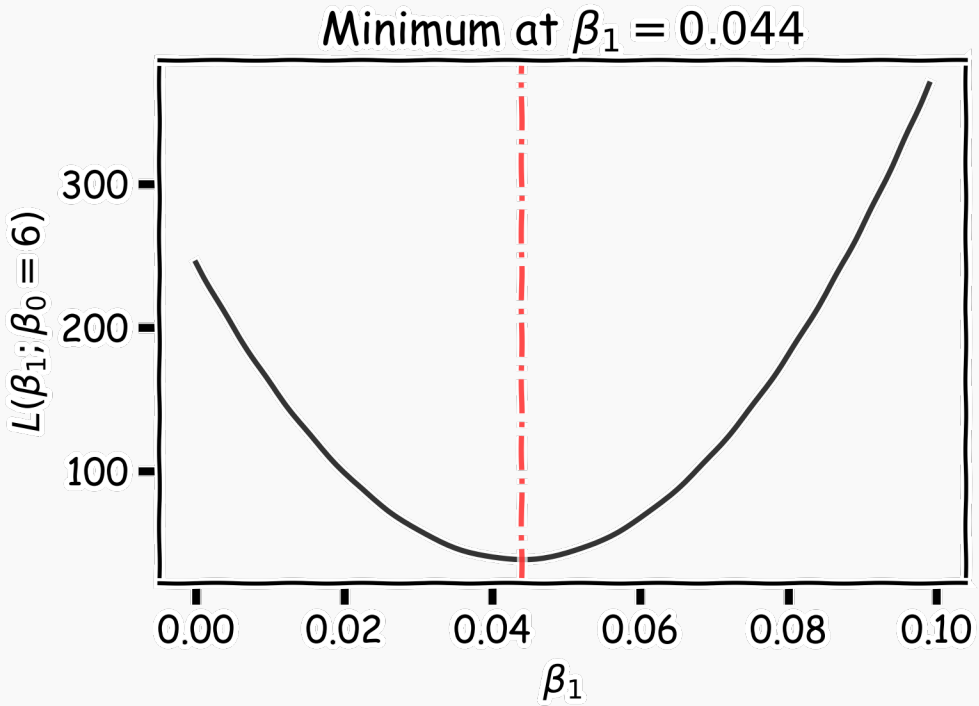
$$\nabla L = \left[\frac{\partial L}{\partial \beta_0}, \frac{\partial L}{\partial \beta_1}\right] = 0$$

- Brute Force: Try every combination
- Exact: Solve the above equations
- Greedy Algorithm: Gradient Descent

Optimization: Brute force

A way to estimate $\text{argmin}_{\beta_0,\beta_1} L$ is to calculate the loss function for every possible β_0 and β_1. Then select the β_0 and β_1 where the loss function is minimum.

E.g. the loss function for different β_1 when β_0 is fixed to be 6:



Very **computationally expensive** with many coefficients

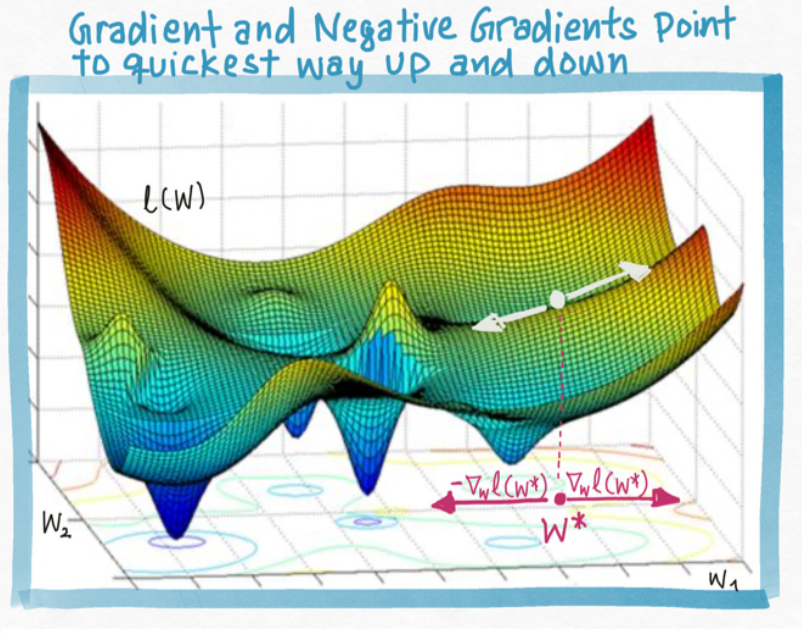
Gradient Descent

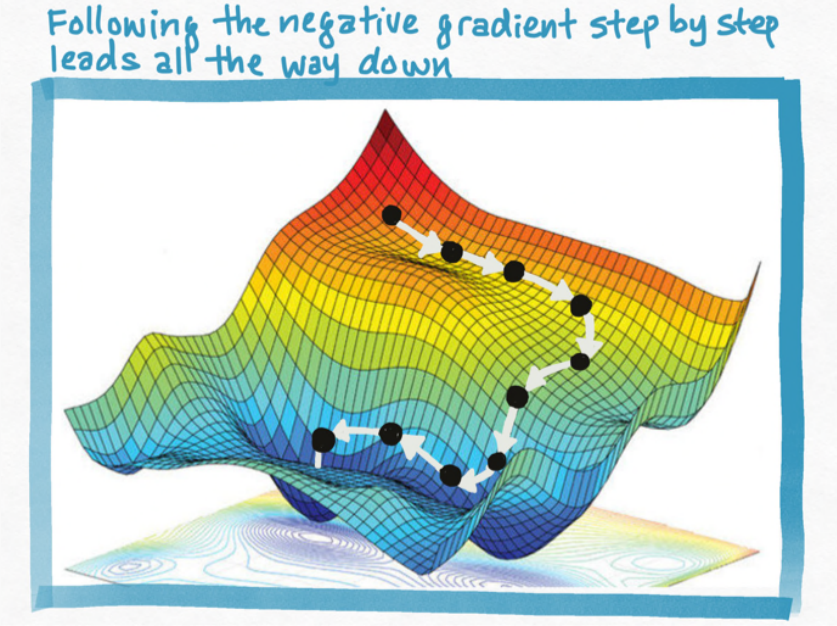
When we can't analytically solve for the stationary points of the gradient, we can still exploit the information in the gradient.

The gradient ∇L at any point is the direction of the steepest increase. The negative gradient is the direction of steepest decrease.

By following the –ve gradient, we can eventually find the lowest point.

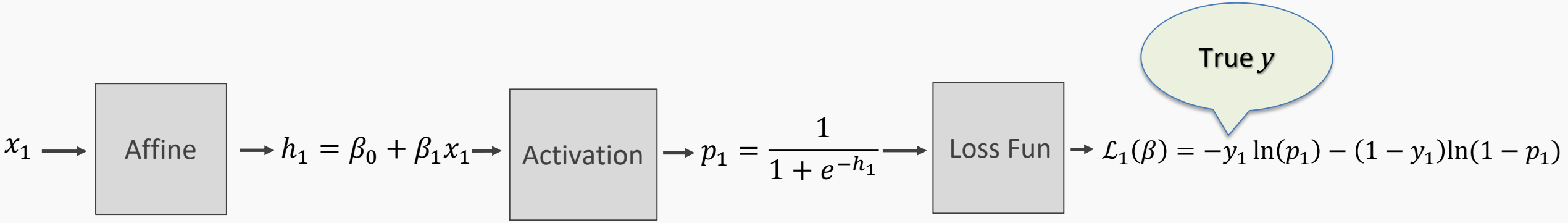
This method is called Gradient Descent



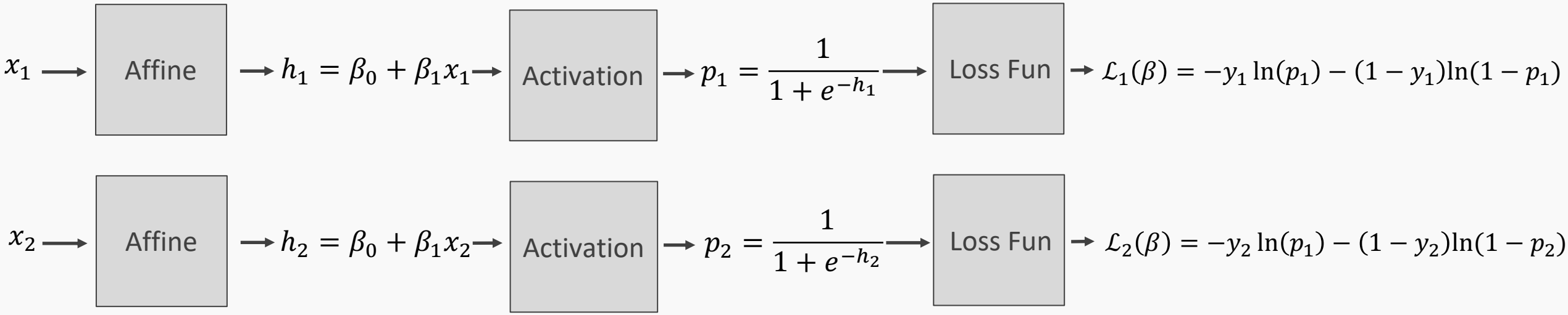


Neural Networks

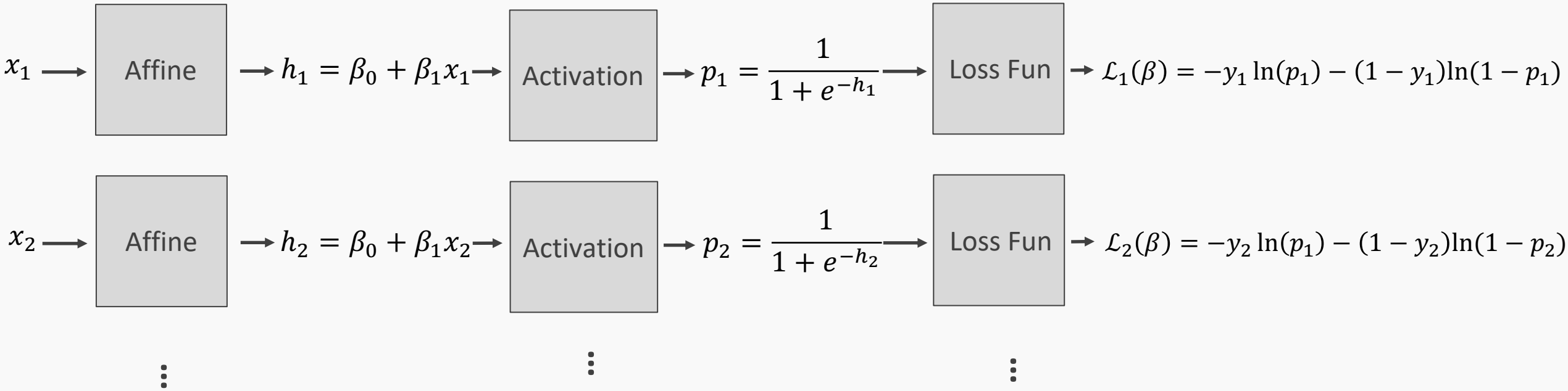
Logistic Regression Revisited



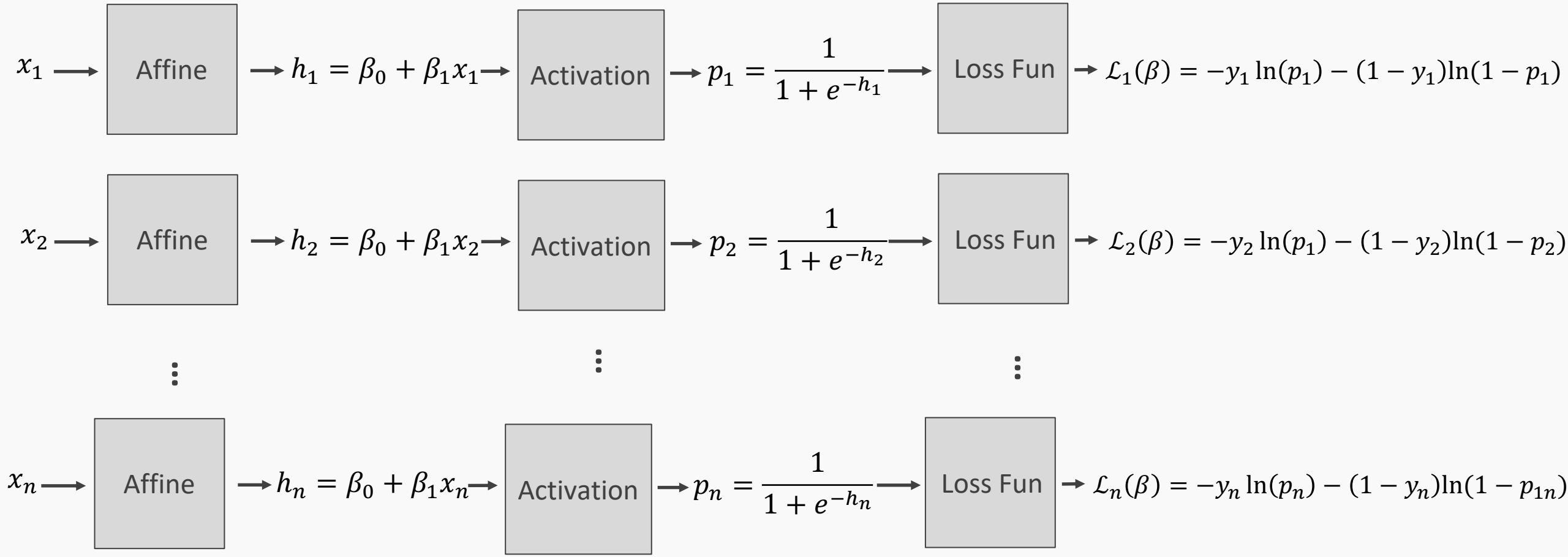
Logistic Regression Revisited



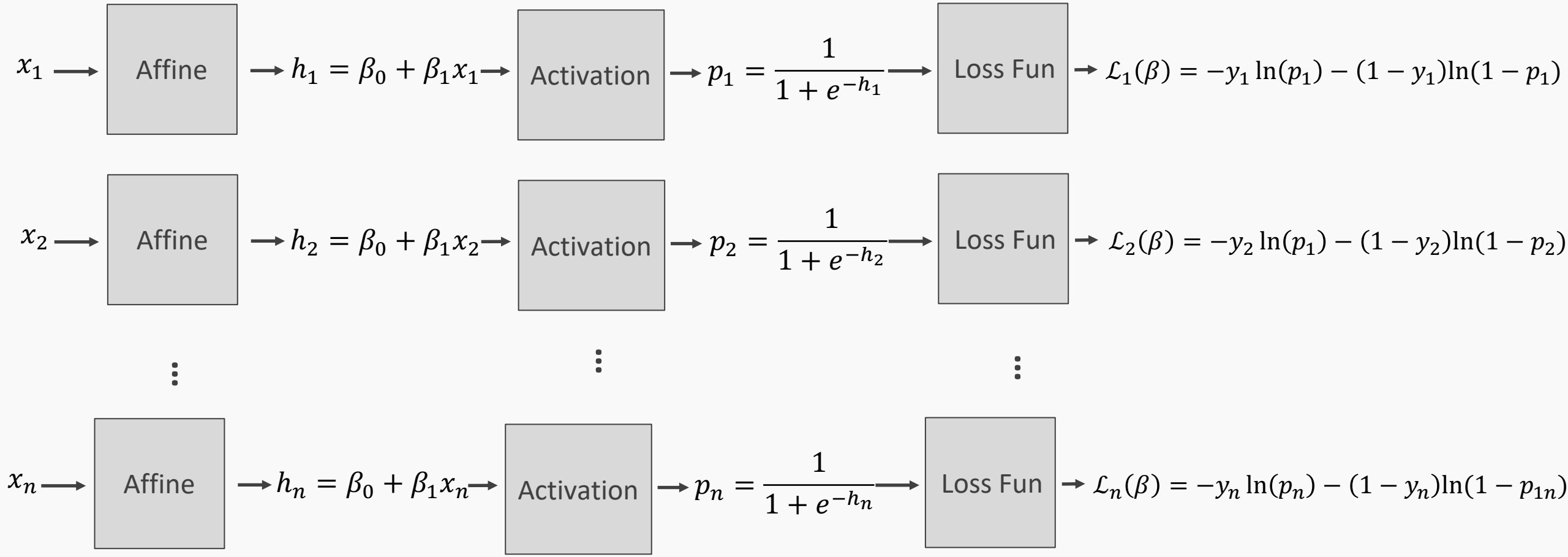
Logistic Regression Revisited



Logistic Regression Revisited

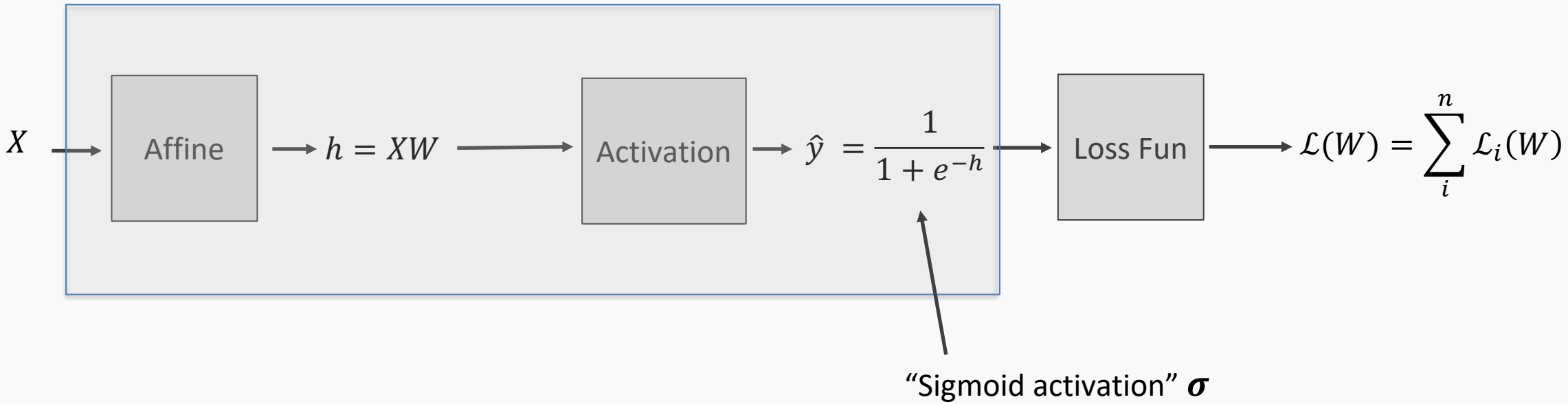


Logistic Regression Revisited

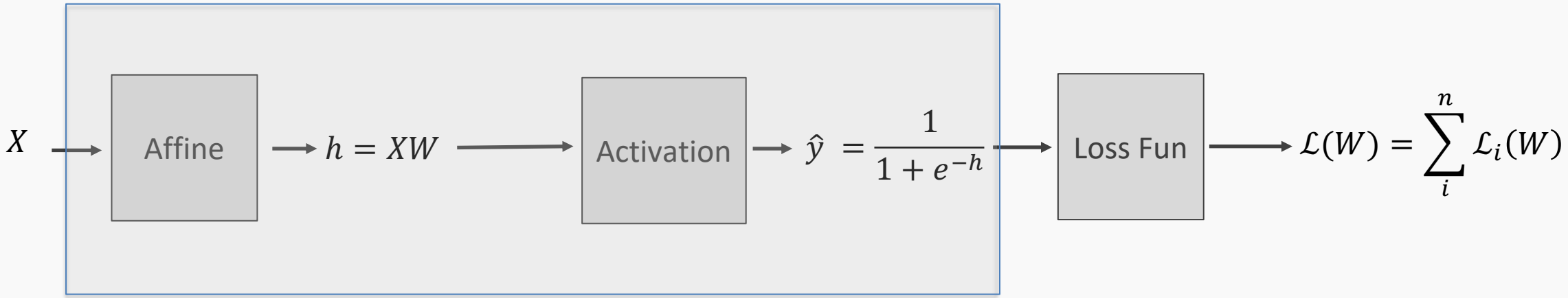


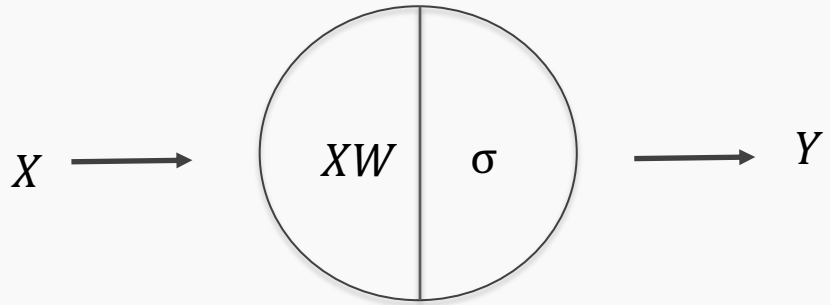
$$\mathcal{L}(\beta) = \sum_{i}^{n} \mathcal{L}_i(\beta)$$

Build our first ANN

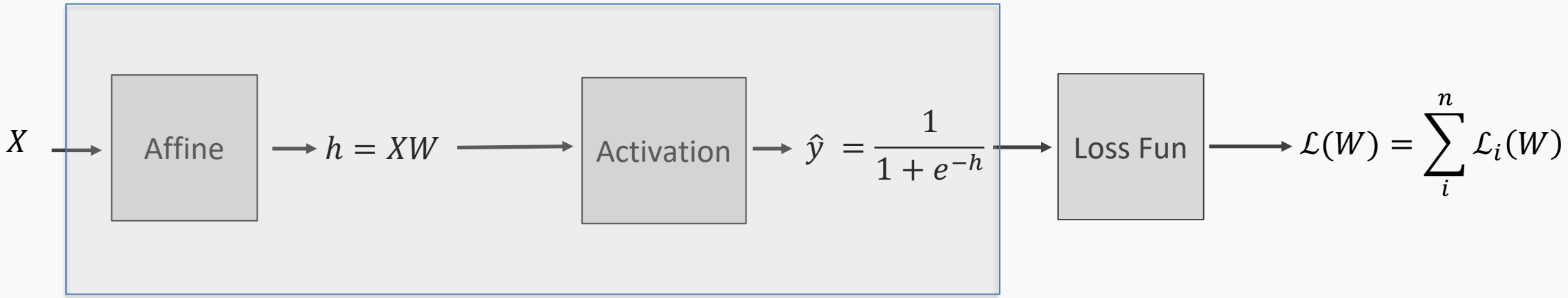


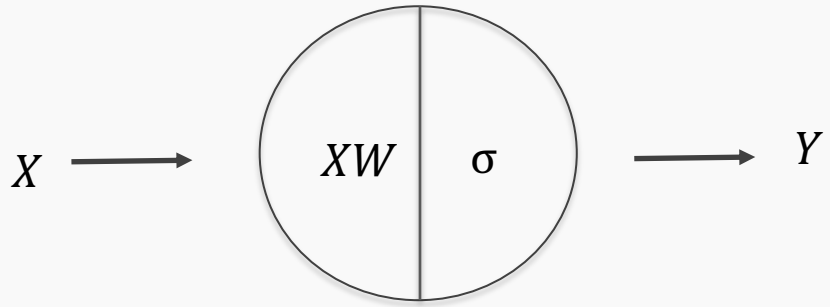
Build our first ANN





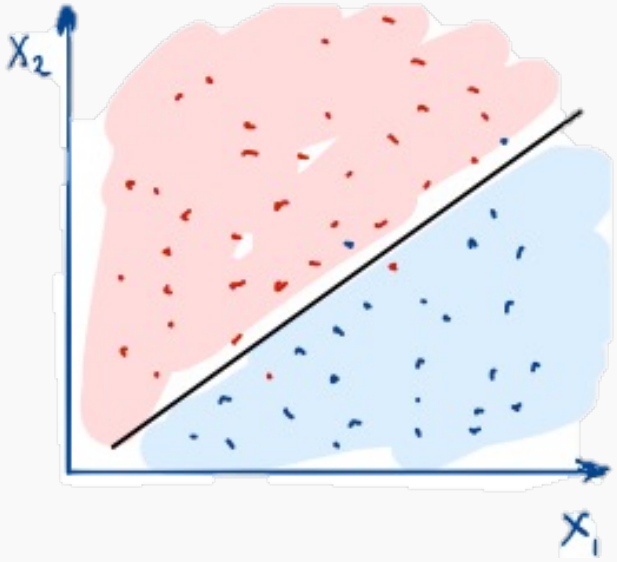
Build our first ANN

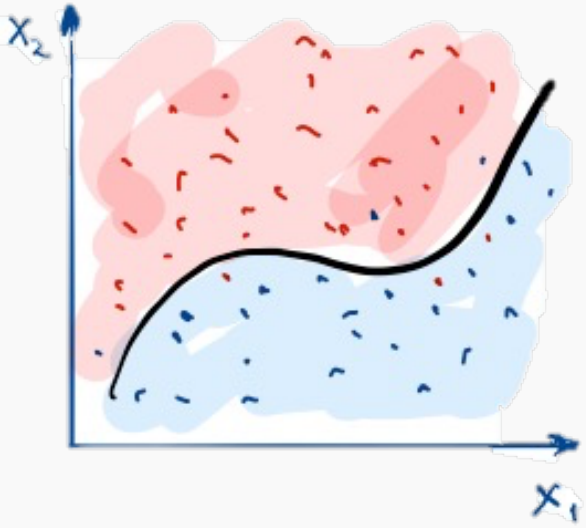


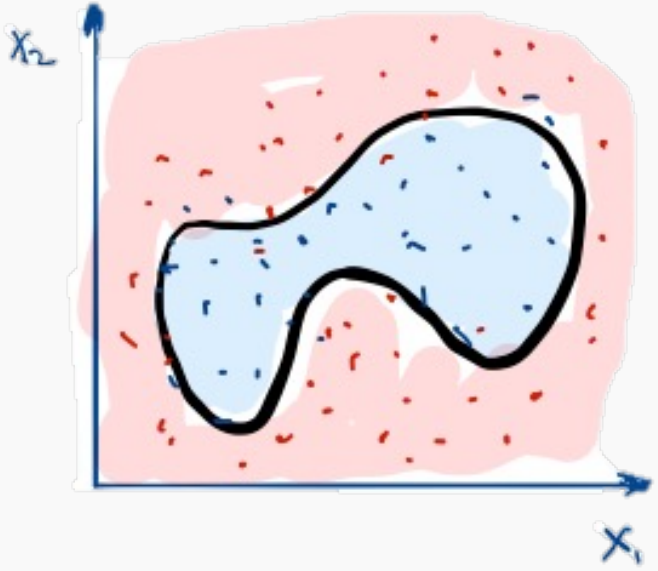


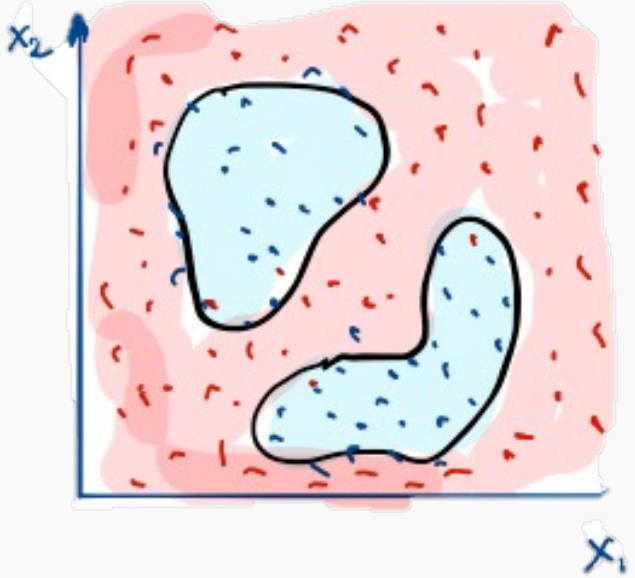
Single Neuron Neural “Network”

So what's the big deal about Neural Networks?

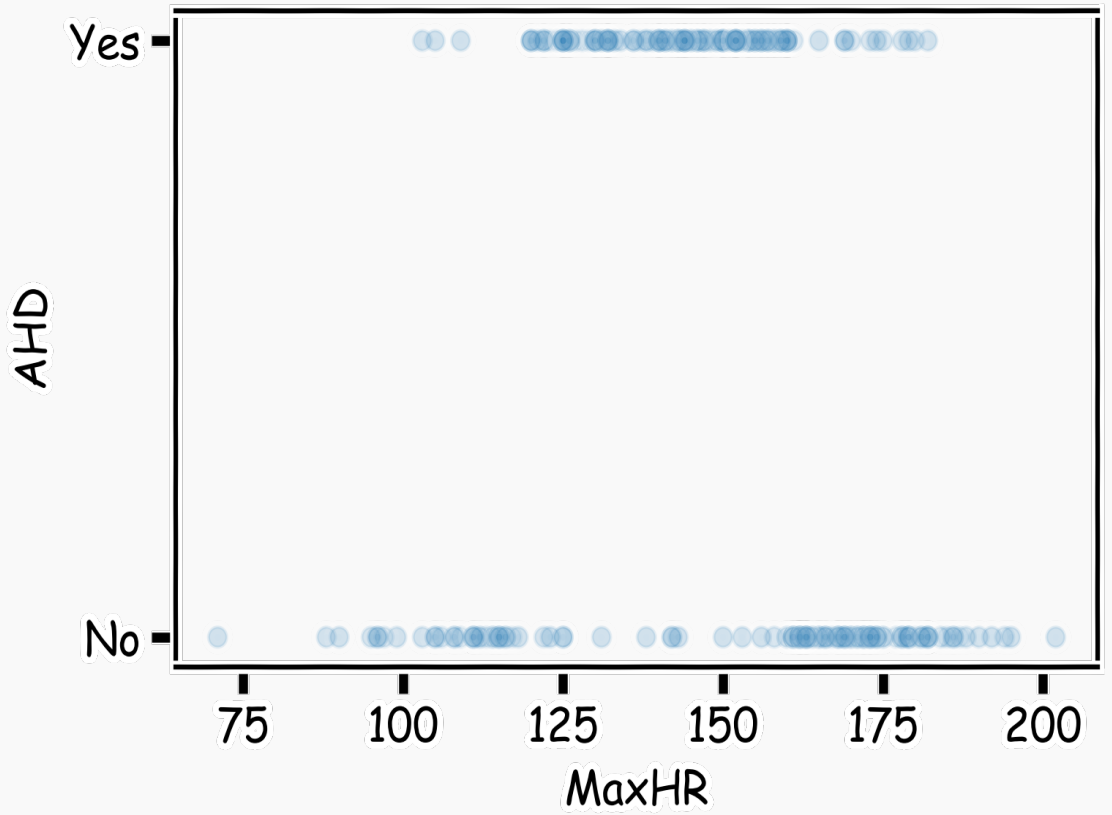




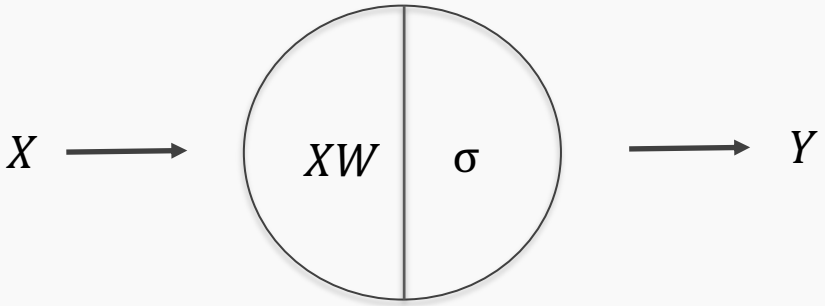




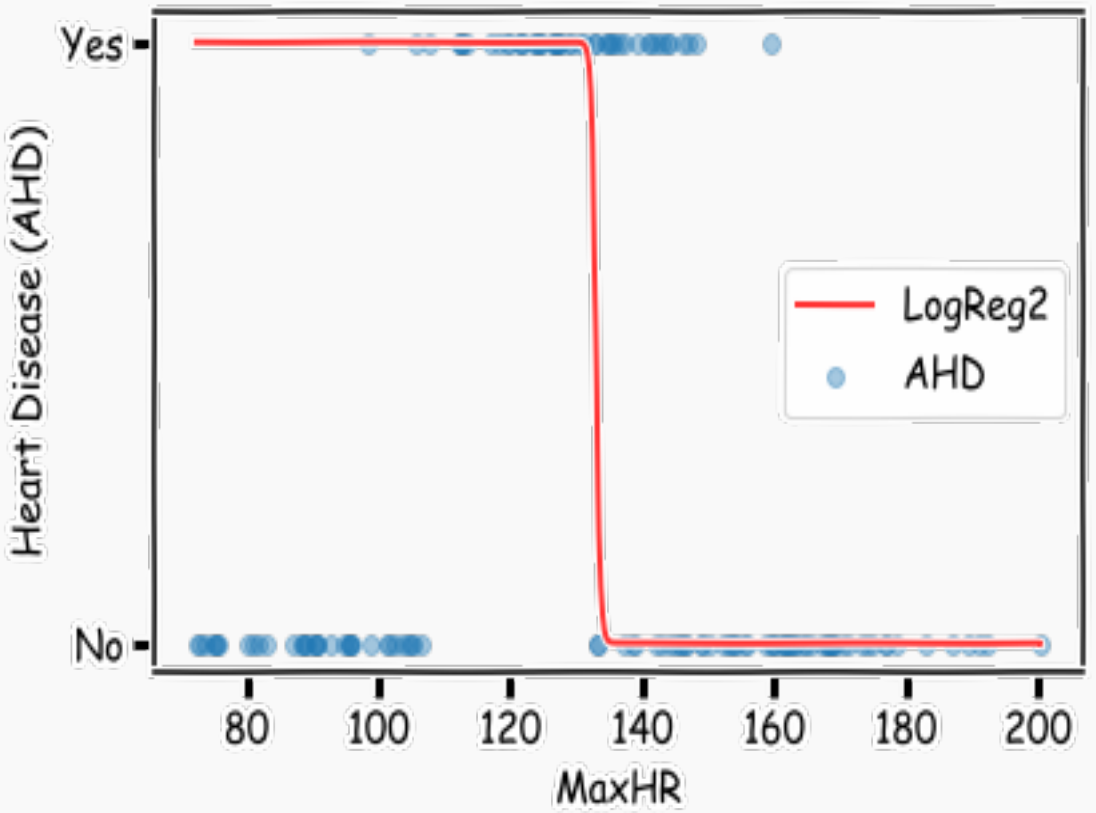
Example Using Heart Data



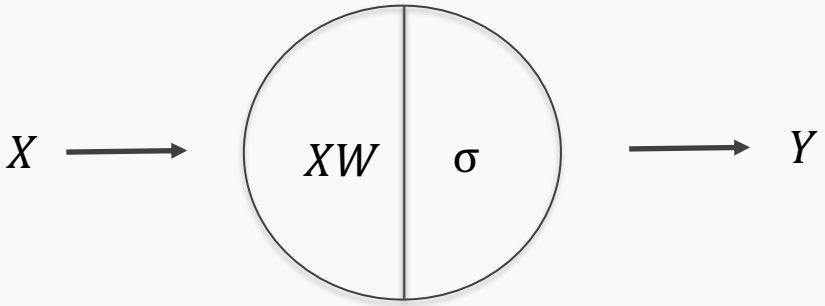
Example Using Heart Data



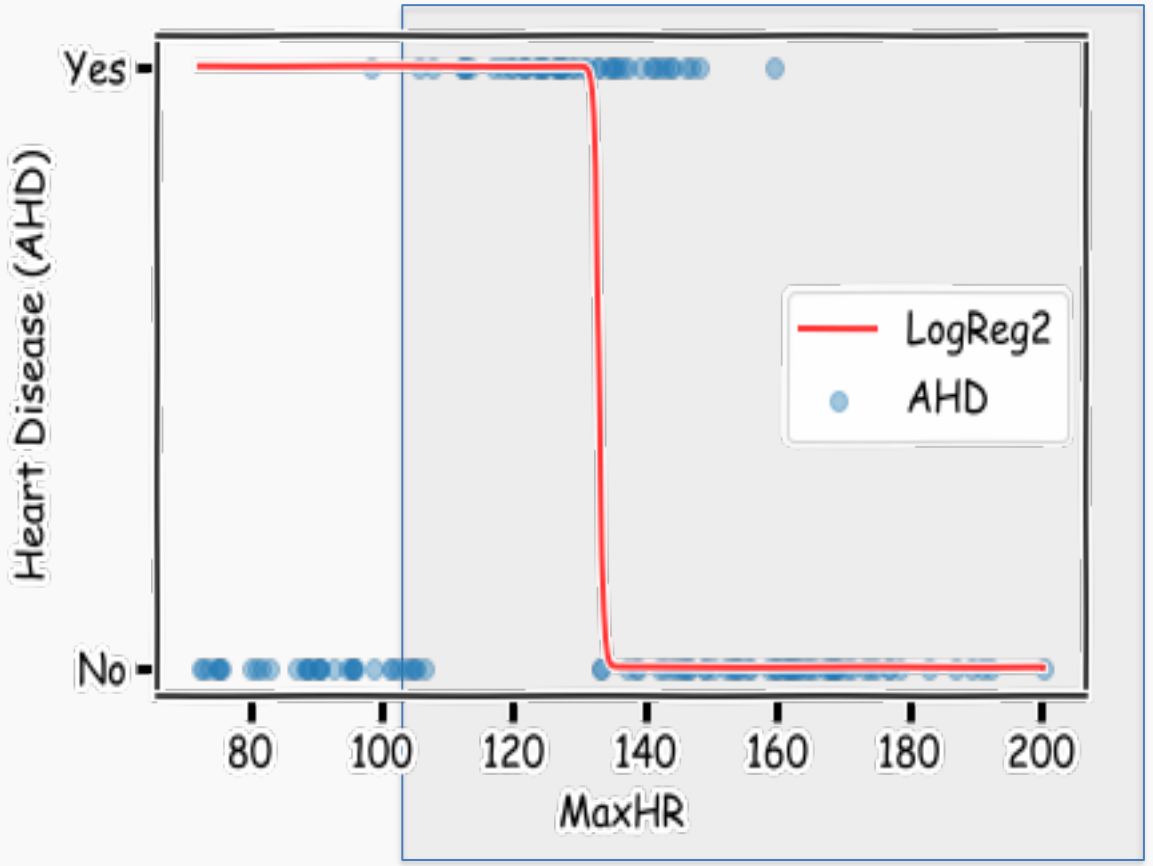
Choose W such as



Example Using Heart Data

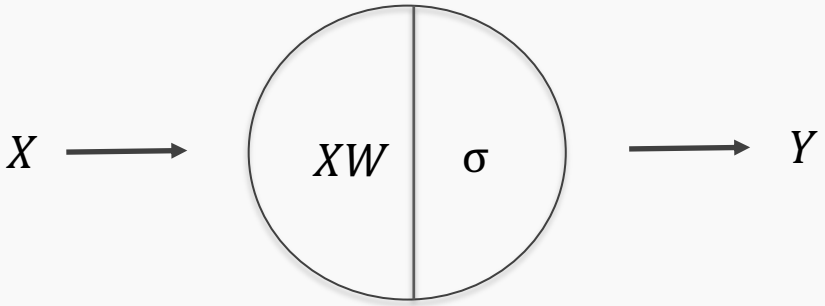


Choose W such as

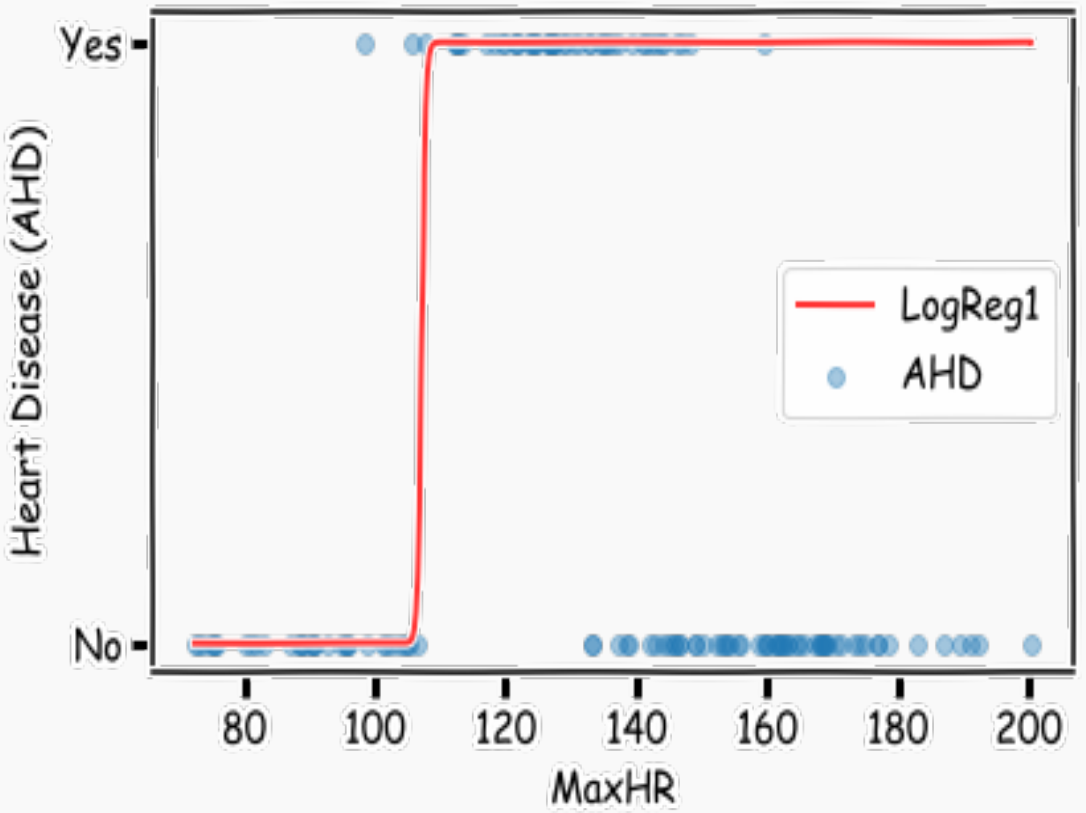


Right part of data is fitted well

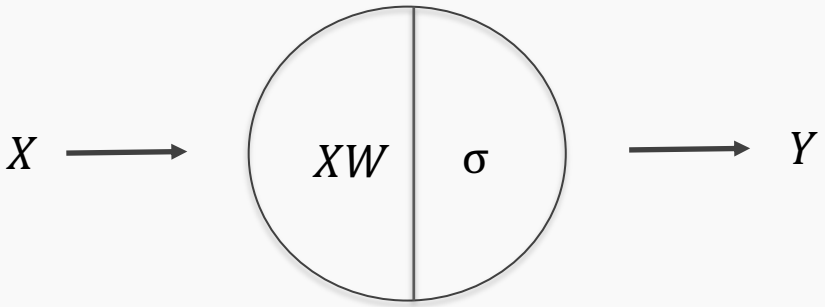
Example Using Heart Data



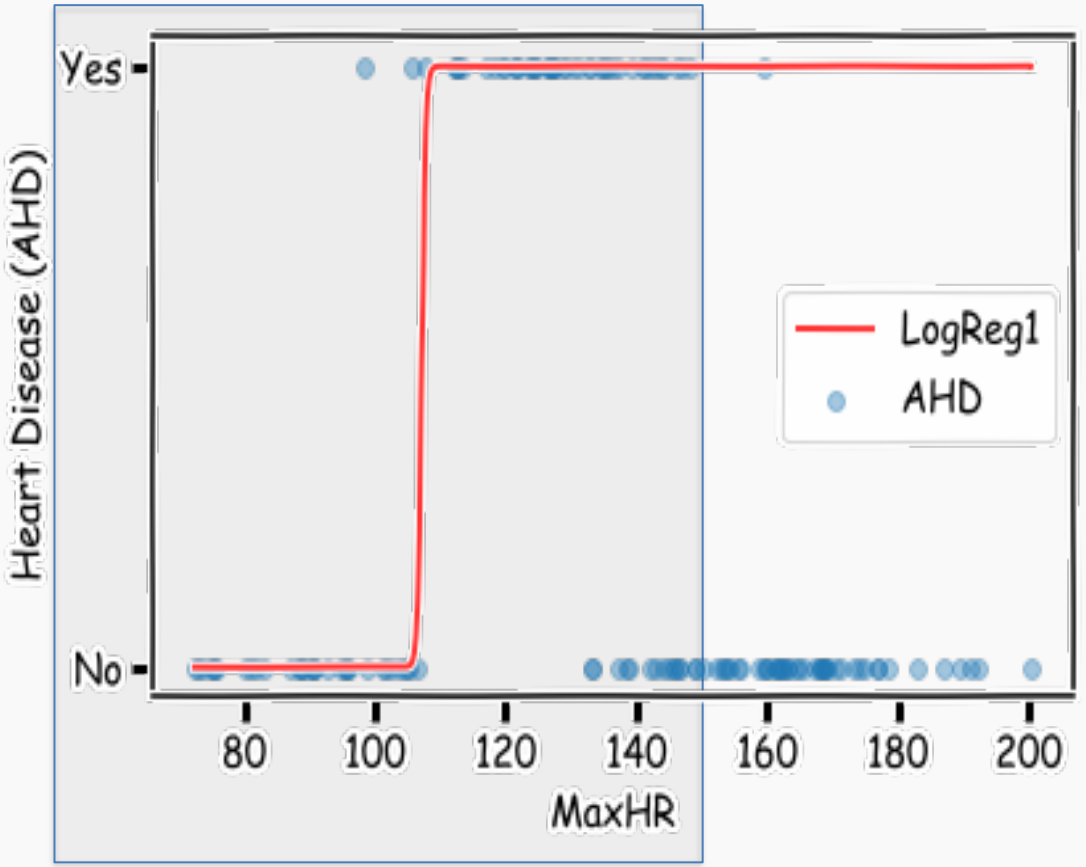
Choose W such as



Example Using Heart Data



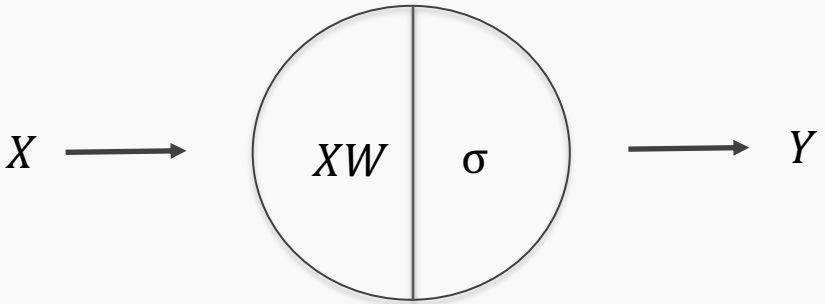
Choose W such as

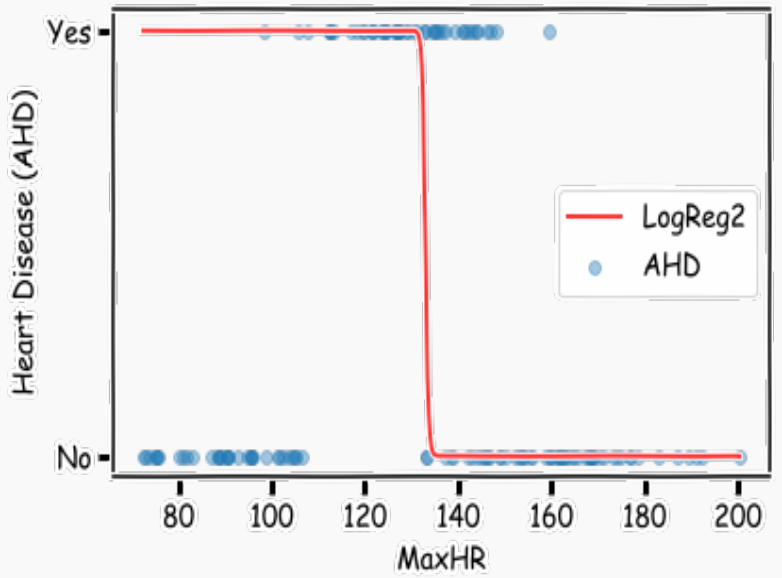


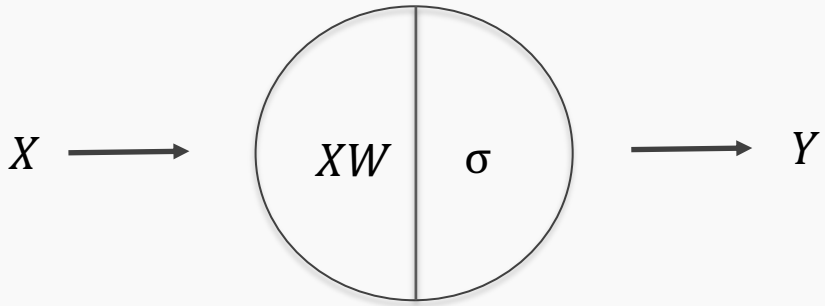
Left part of data is fitted well

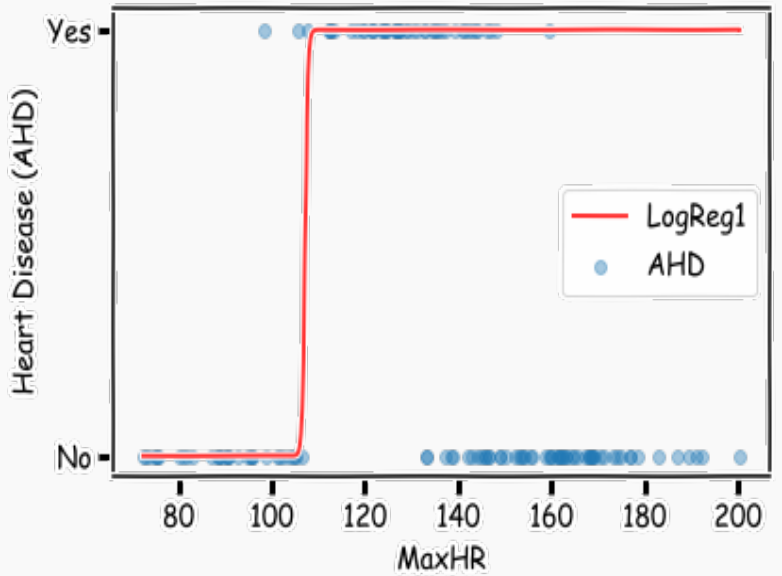
Example Using Heart Data

Two regions, two nodes

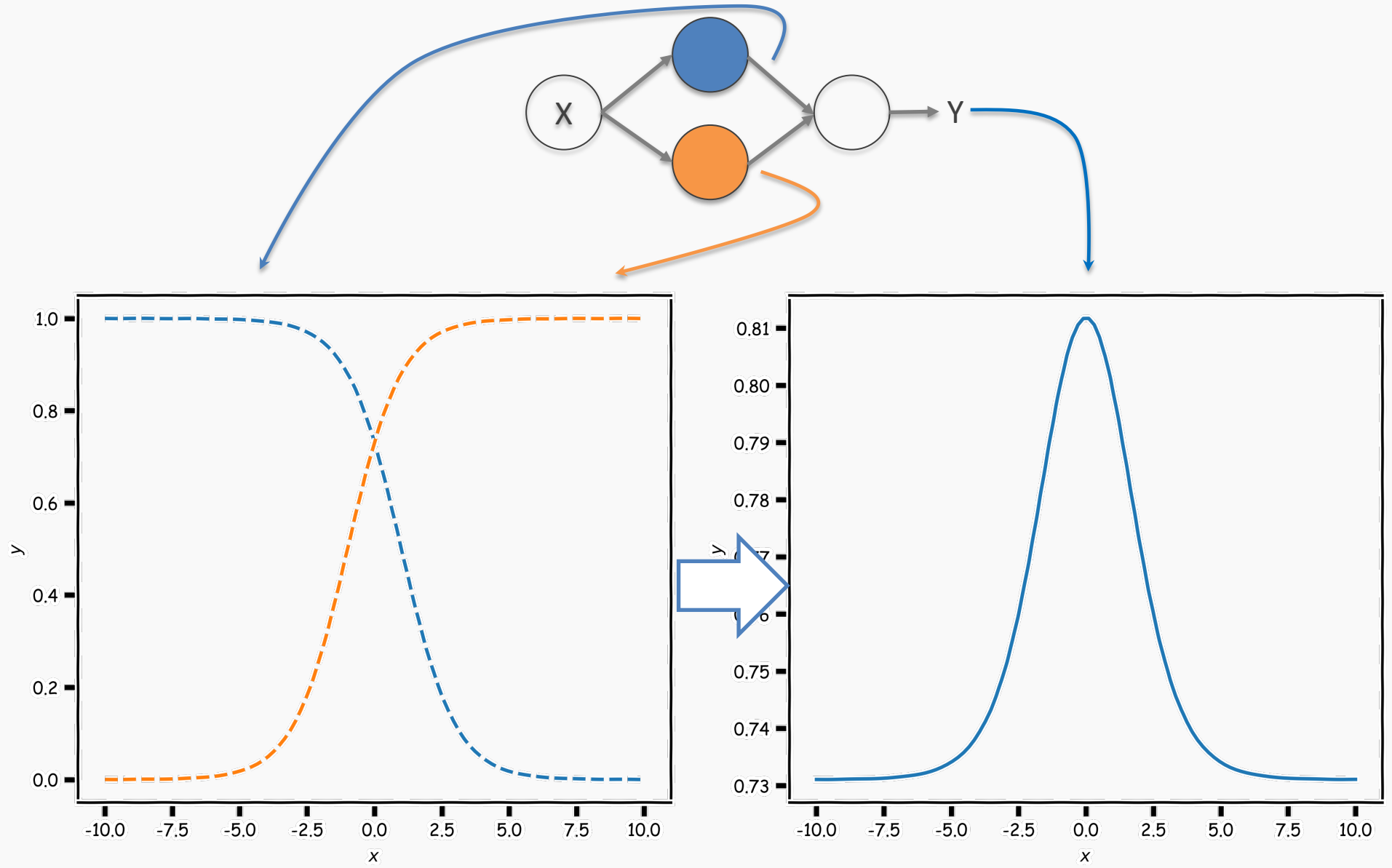




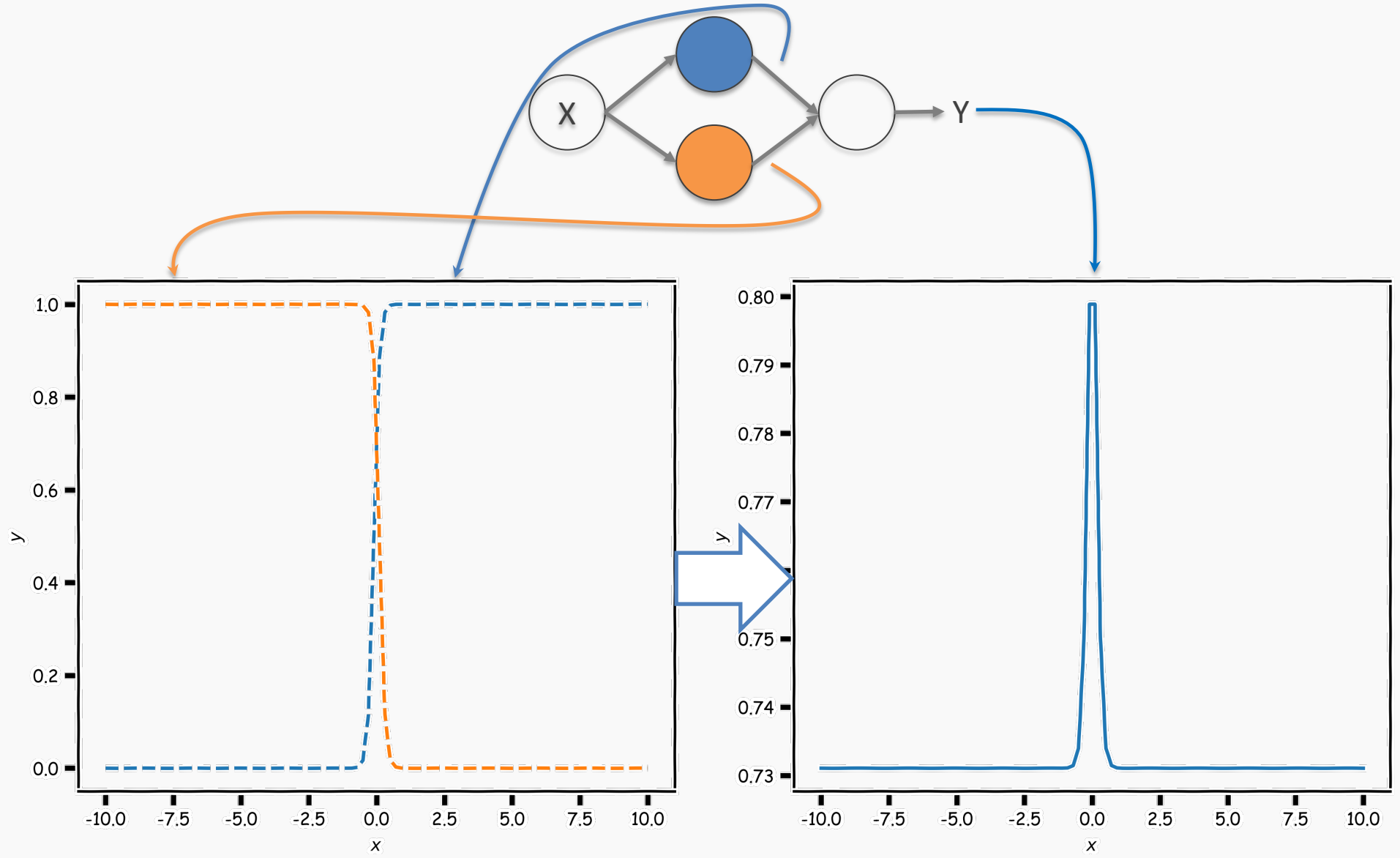




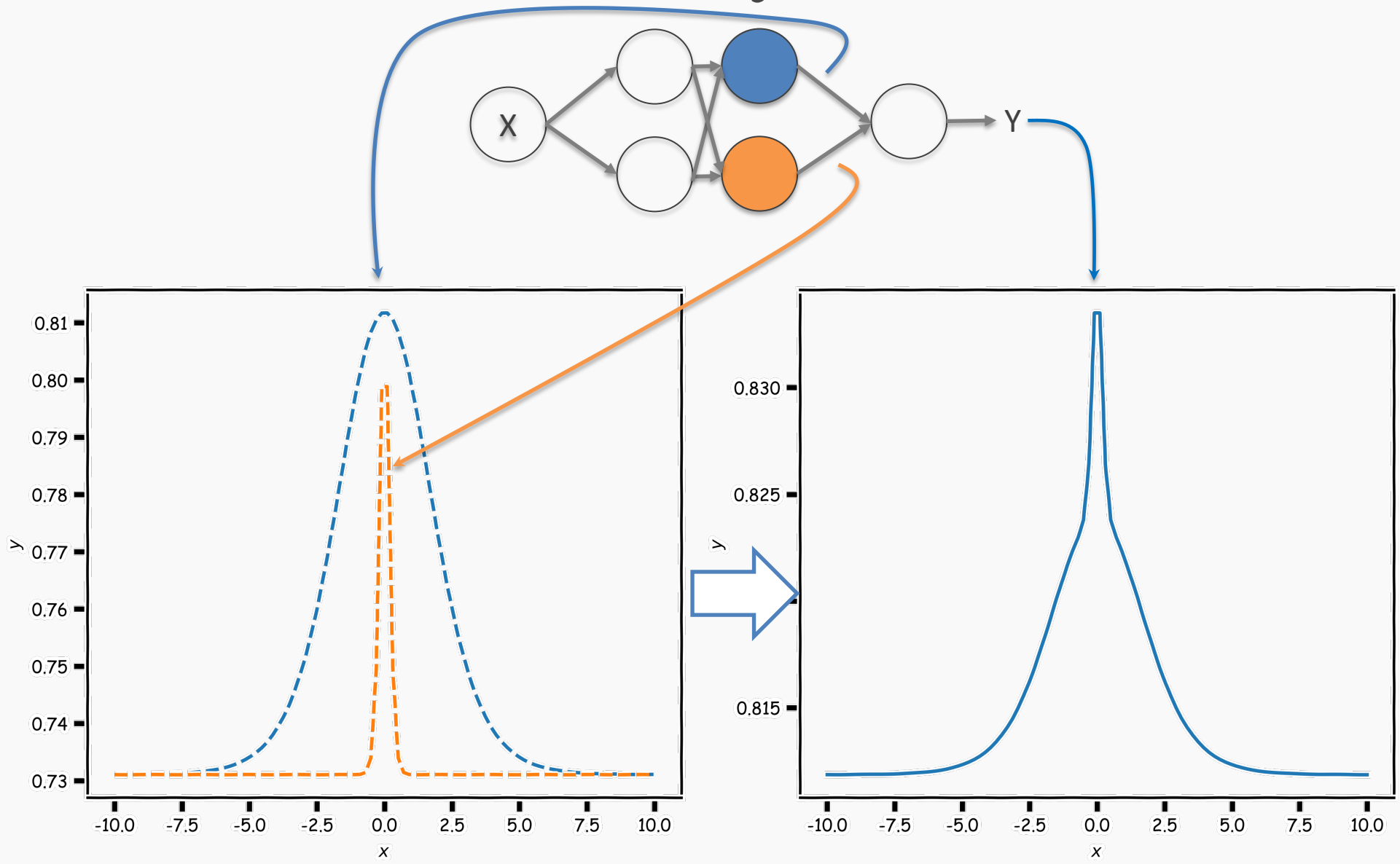
Combining neurons allows us to model interesting functions



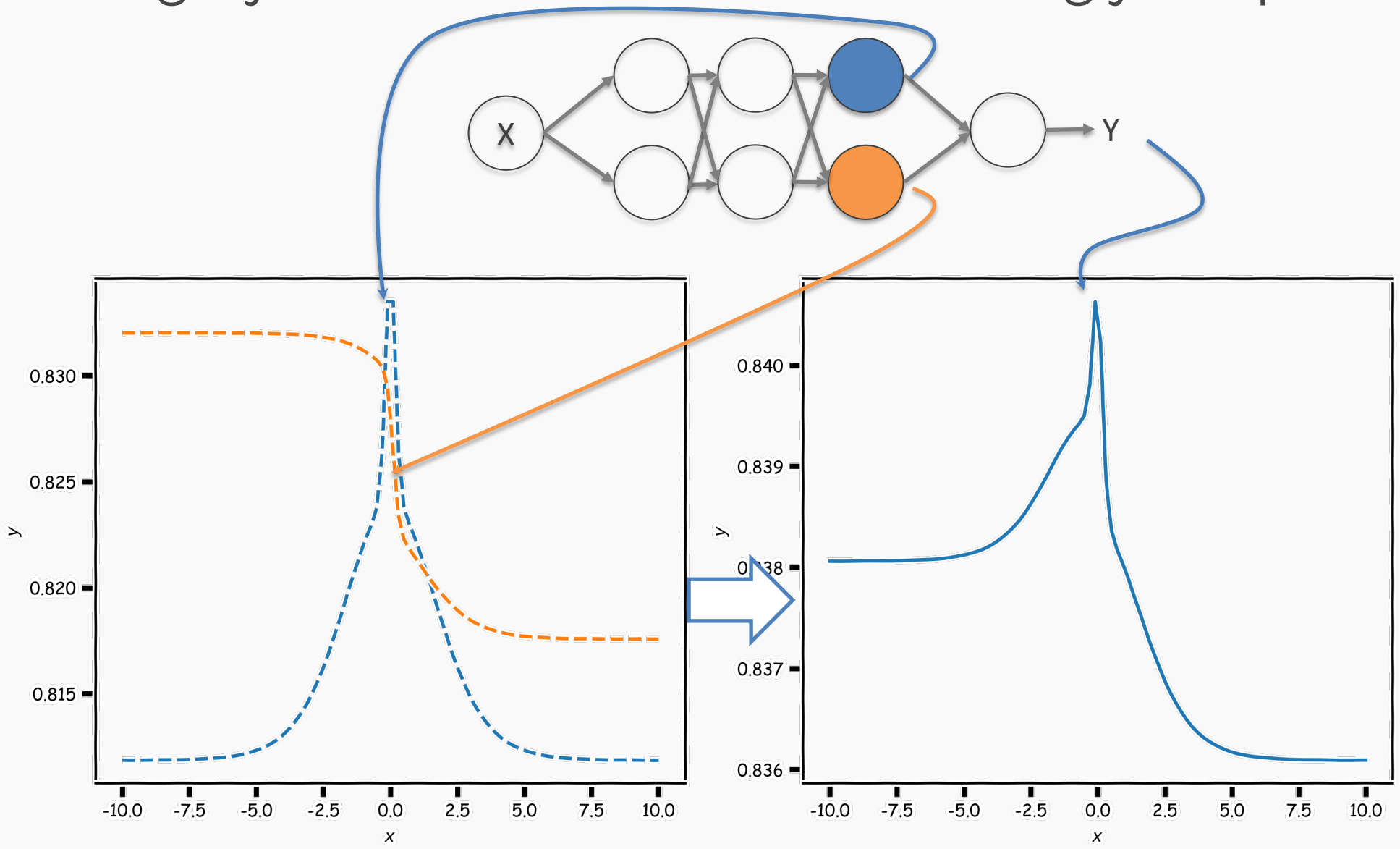
Different weights change the shape and position



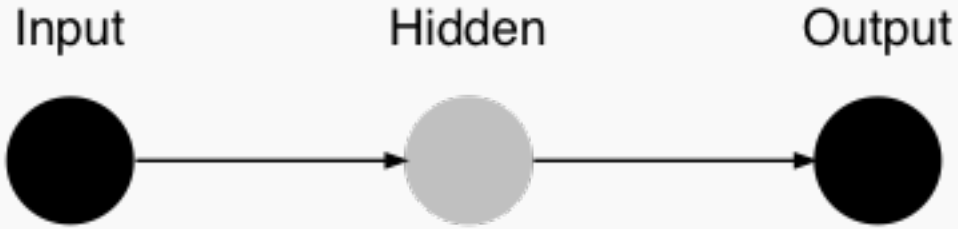
Neural networks can model *any* reasonable function

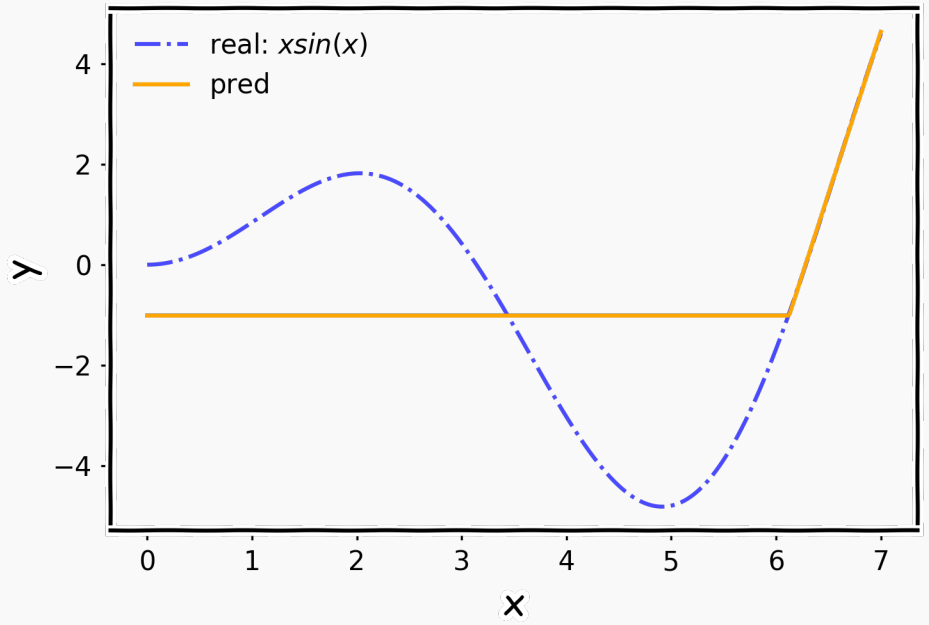


Adding layers allows us to model increasingly complex functions

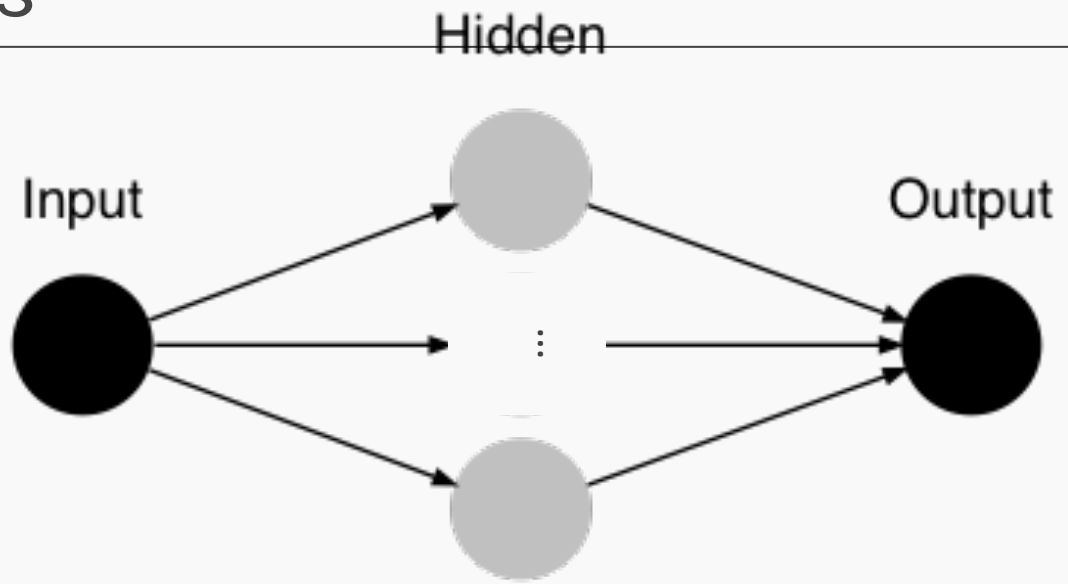


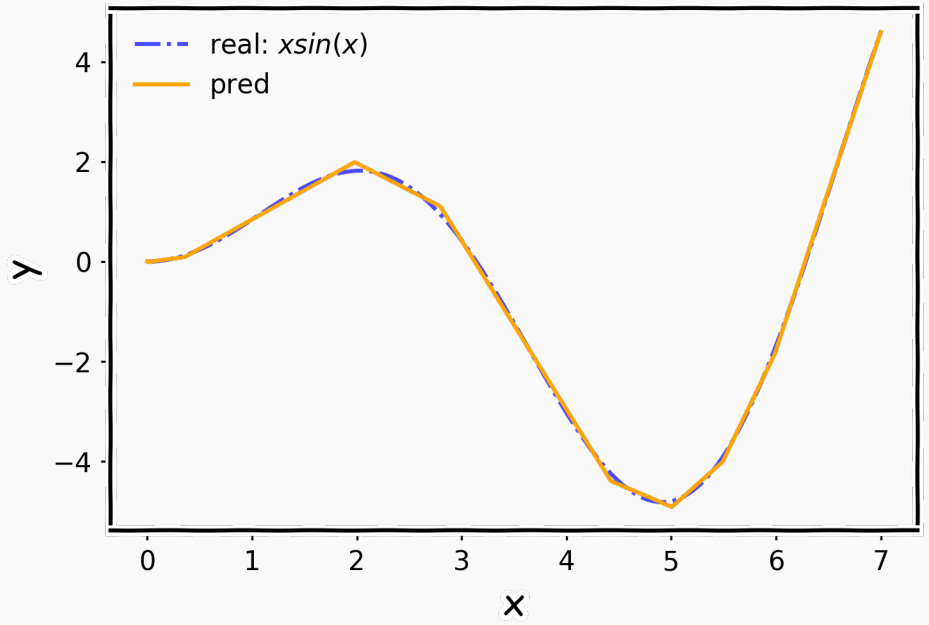
Number of nodes



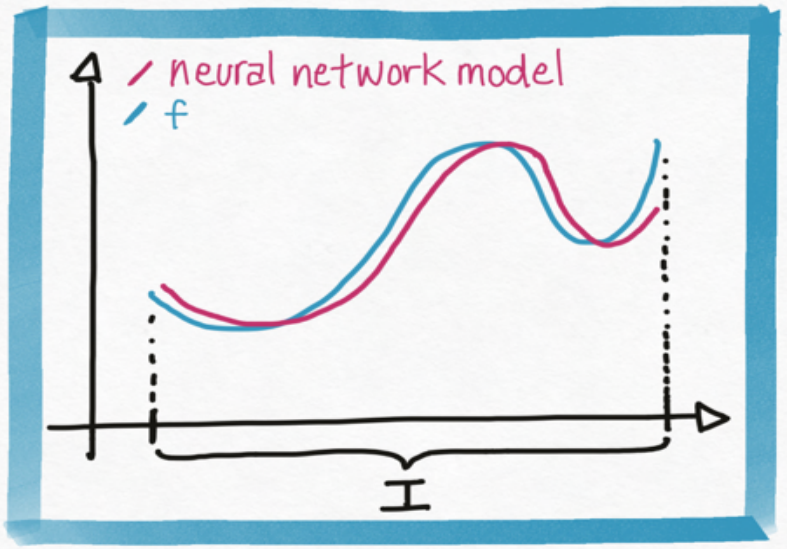


Number of nodes





Neural Networks as Universal Approximators



We have seen that neural networks can represent complex functions, but are there limitations on what a neural network can express?

Theorem:

For any continuous function f defined on a bounded domain, we can find a neural network that approximates f with an arbitrary degree of accuracy.

Summary

So far:

- A single neuron can be a logistic regression or linear unit.
- A neural network is a combination of logistic regression (or other types) units.
- A neural network can approximate non-linear functions either for regression or classification.