6.100B: Recitation 6

Classifiers: Perceptron and Neural Networks

May 12th, 2023

Actionables

PS4 checkoff is due **today, 5/12** by **5pm**

PS5 due on **Monday, 5/15** by **9pm**; no Checkoff!

No office hours on or after **Tuesday, 5/16**

Agenda

Classification

Hyperplanes

Perceptron

Neural Networks

Let’s get started!

Machine Learning | Hyperplanes

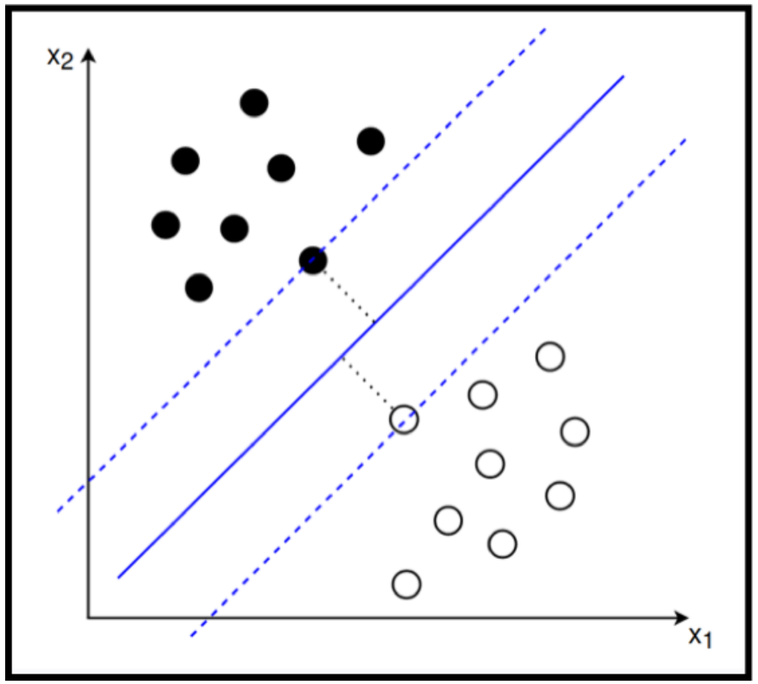
We’ve already dealt with Machine Learning through linear regressions to create a model to fit data while still being generalizable

What about data with more dimensionality?

When we would like to **classify** data, we need to create a model that can be trained to predict a quality of a sample based on its dimensionality

Machine Learning | Hyperplanes

When dealing with 2D Space, a **hyperplane** is a line that separates space into a **positive** domain and a **negative** domain

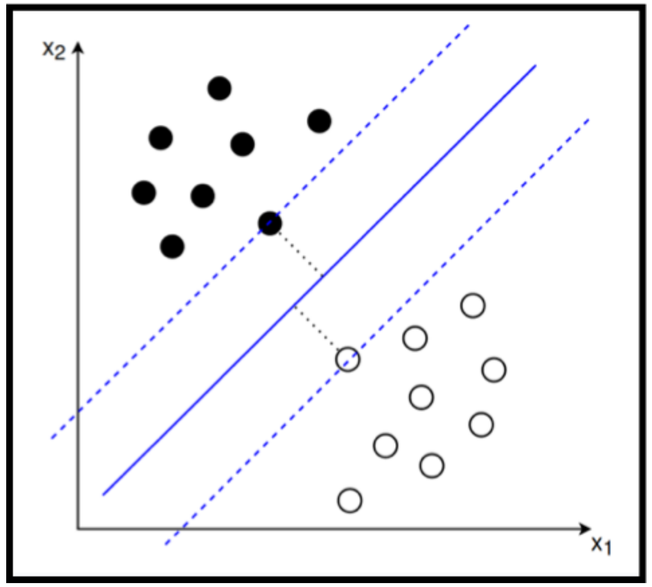


Machine Learning | Hyperplanes

We can represent our model as a **hypothesis:**

$$y = 1 \text{ if } \theta^\top x + \theta_0 > 0$$
$$y = -1 \text{ otherwise}$$

$$\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}^\top \begin{bmatrix}x_1 \\ x_2\end{bmatrix} + \theta_0 = \theta_2 x_2 + \theta_1 x_1 + \theta_0$$



Machine Learning | Hyperplanes

For example, let's consider a point $(2,5)$ on the coordinate plane

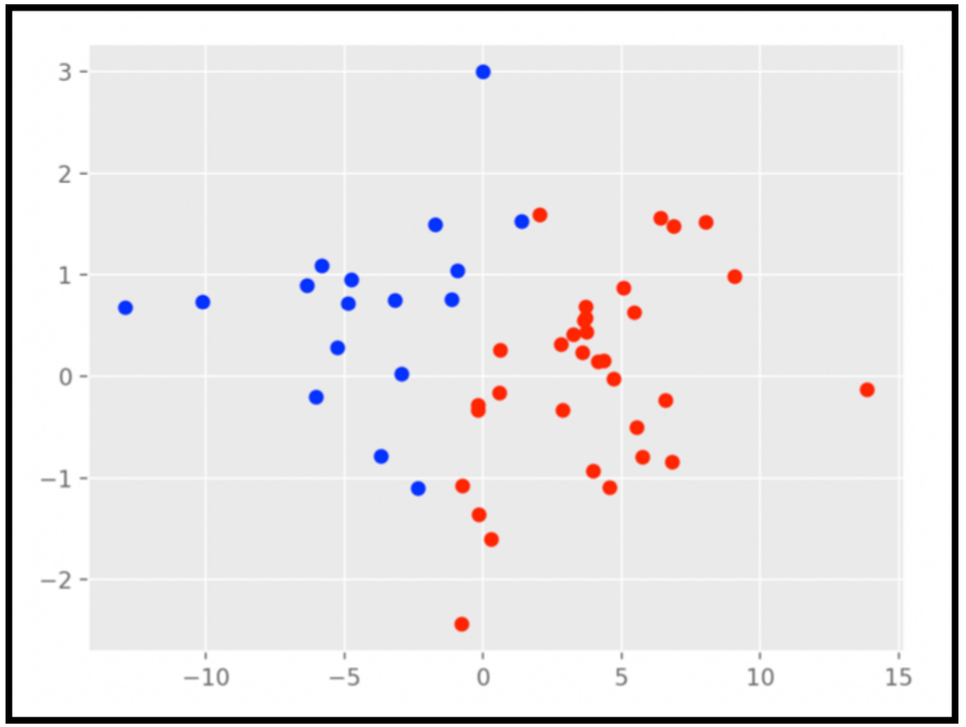
A model with $\theta = [1, -3]$ and $\theta_0 = 2$ will classify this point:

$$\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}^\top \begin{bmatrix}x_1\\ x_2\end{bmatrix} + \theta_0 = \begin{bmatrix}1\\ -3\end{bmatrix}^\top \begin{bmatrix}2\\ 5\end{bmatrix} + 2$$
$$= 2 - 15 + 2 < 0$$

Therefore, this point would be classified as **y = -1**

Machine Learning | Hyperplanes

Given a cluster of data in which **Blue** represents a positive classification and **Red** represents a negative classification, how do we create a model to fit the data?



Machine Learning | Hyperplanes

Our objective function will follow roughly the same idea as our linear regression, only we're concerned now with **classifying points correctly**, rather than how far we are from the point

Therefore, we only assign loss when we **misclassify** a point

$$J(\theta, \theta_0) = -\sum_{i}^{N} \left(y_i(\theta^\top x_i + \theta_0) \right)$$

Otherwise, $J(\theta, \theta_0) = 0$

This loss function is called the **hinge loss**

Machine Learning | Hyperplanes

If our model is wrong, we want to ensure that our model steps in the right direction:

$$J(\theta, \theta_0) = -\frac{1}{N} \sum_{i}^{N} \left(y_i (\theta^\top x_i + \theta_0) \right)$$

This way, we correct each attribute of our model in the following way when the model finds a mistake, **y** is the same sign as our prediction:

Machine Learning | Hyperplanes

This way, we correct each attribute of our model in the following way when the model finds a mistake, **y** is the same sign as our prediction:

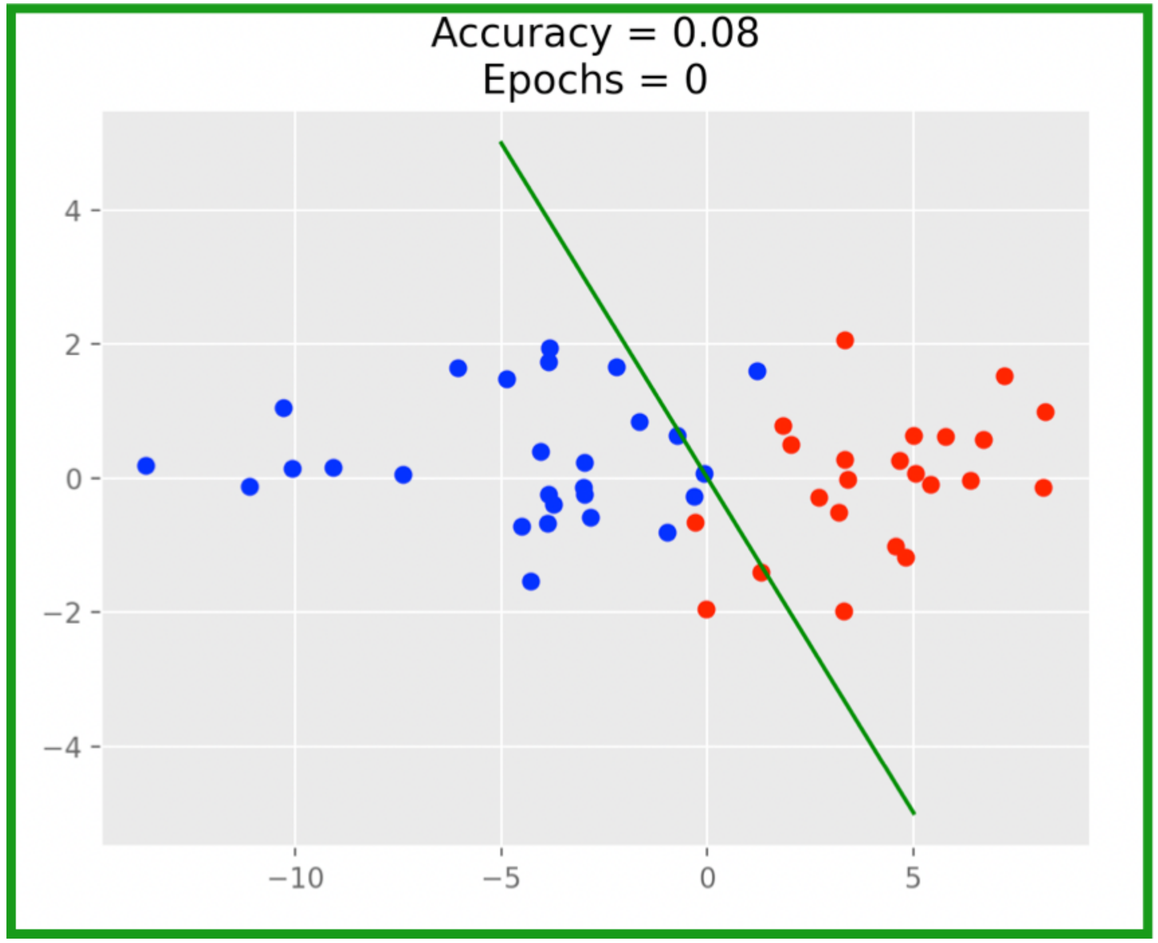
$$\theta_2 := \theta_2 + \frac{1}{N}\sum_{j}^{R}(y_j)(x_j[2])$$

$$\theta_1 := \theta_1 + \frac{1}{N}\sum_{j}^{R}(y_j)(x_j[1])$$

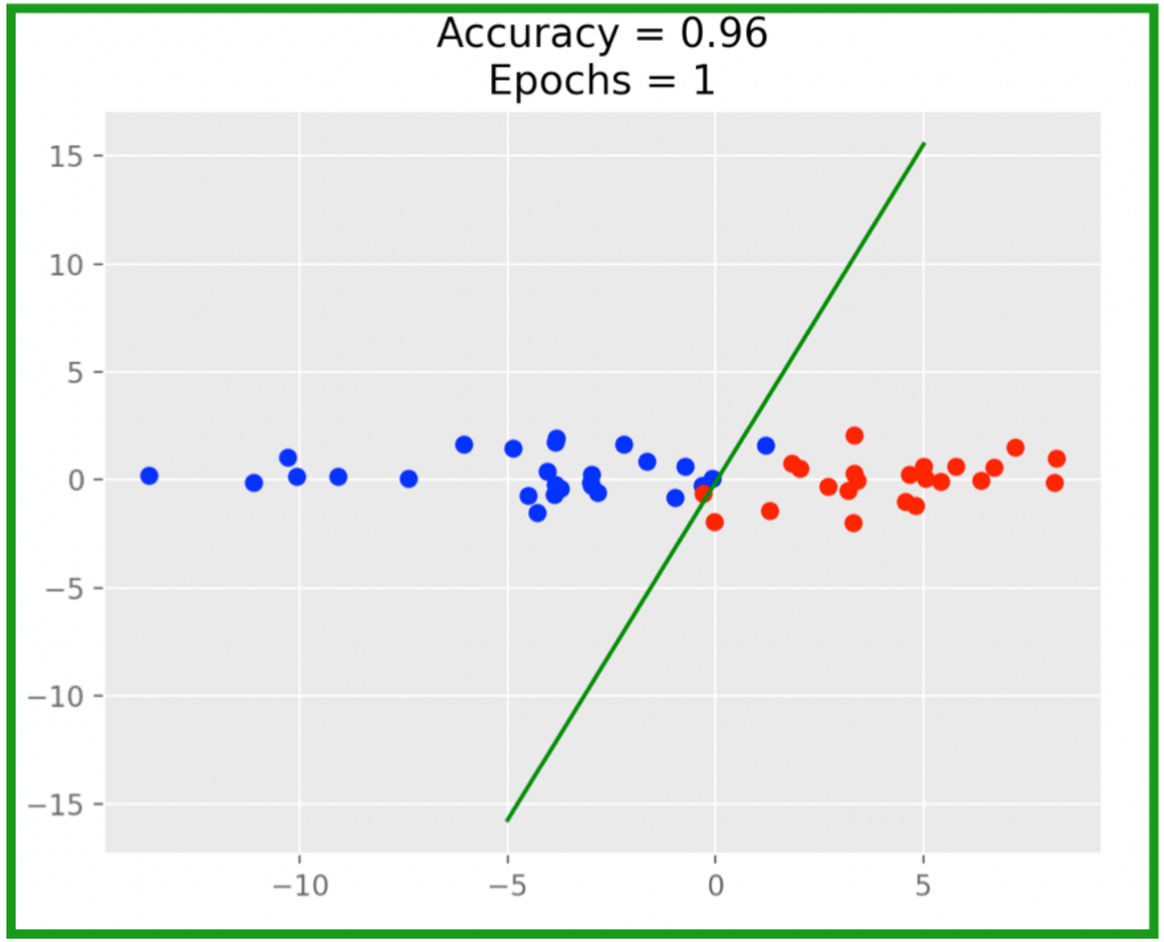
$$\theta_0 := \theta_0 + \frac{1}{N}\sum_{j}^{R}y_j$$

Where y_j represents one of R misclassified points. We **accumulate the error** in each epoch before making our step, though this strategy can be altered to our needs

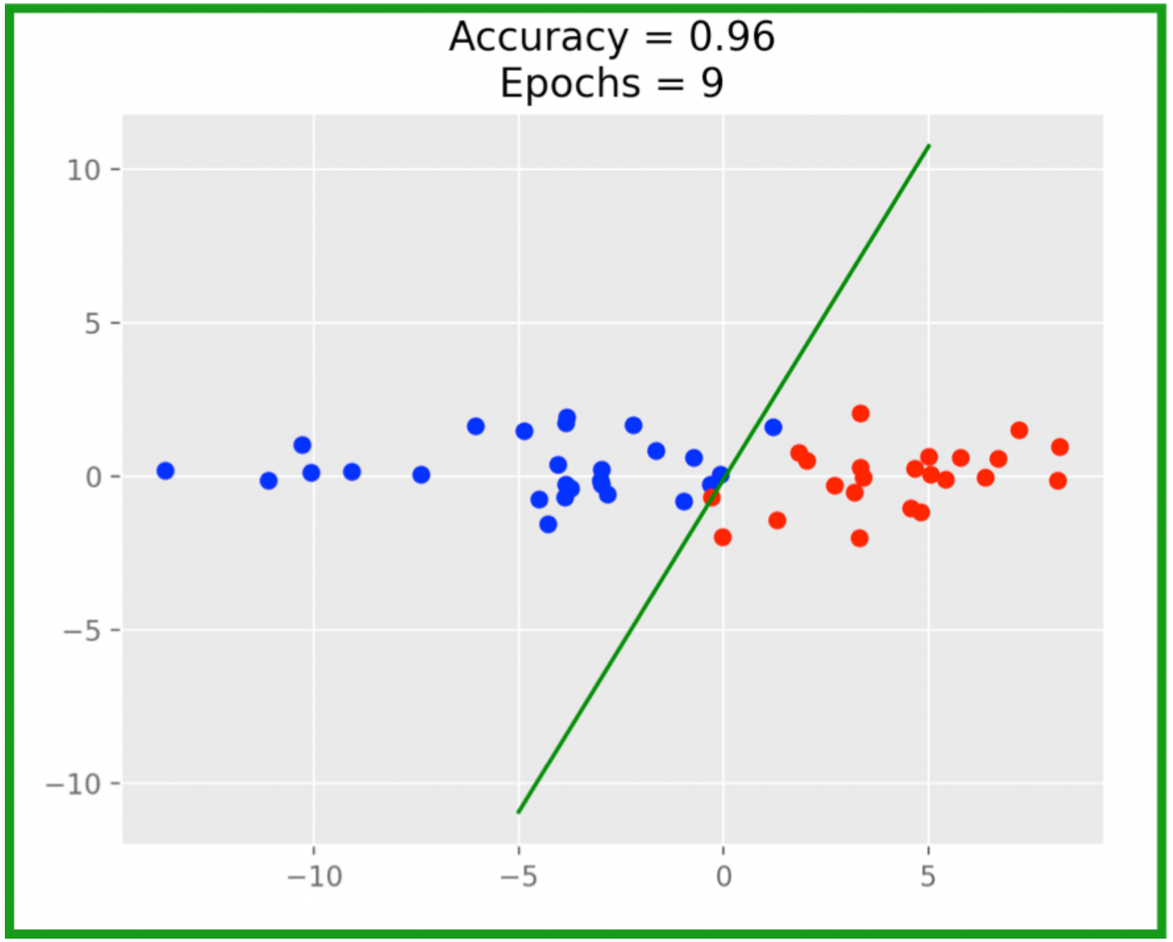
Machine Learning | Hyperplanes



Machine Learning | Hyperplanes



Machine Learning | Hyperplanes



Machine Learning | Hyperplanes

Our model will overfit our data **very quickly** when training on the same data for too many epochs

With hyperplanes, each dimension θ_i can grow very large while preserving the same linear discriminator (the plane dividing our 2D coordinate plane)

As $||\theta||$ grows, it becomes harder to train when introduced to new data

How do we dissuade this behavior?

Machine Learning | Hyperplanes

Many different strategies, but a popular one is a **regulating term**

$$J(\theta, \theta_0) = -\frac{1}{N}\sum_{i}^{N}\left(y_i(\theta^\top x_i + \theta_0)\right) + \frac{1}{2}\lambda\|\theta\|^2$$

In this case, we can continuously train our model to stay as **simple as possible** while correcting our hyperplane classifier if we misclassify a point.

Machine Learning | Hyperplanes

In this case, we can continuously train our model to stay as **simple as possible** while correcting our hyperplane classifier if we misclassify a point:

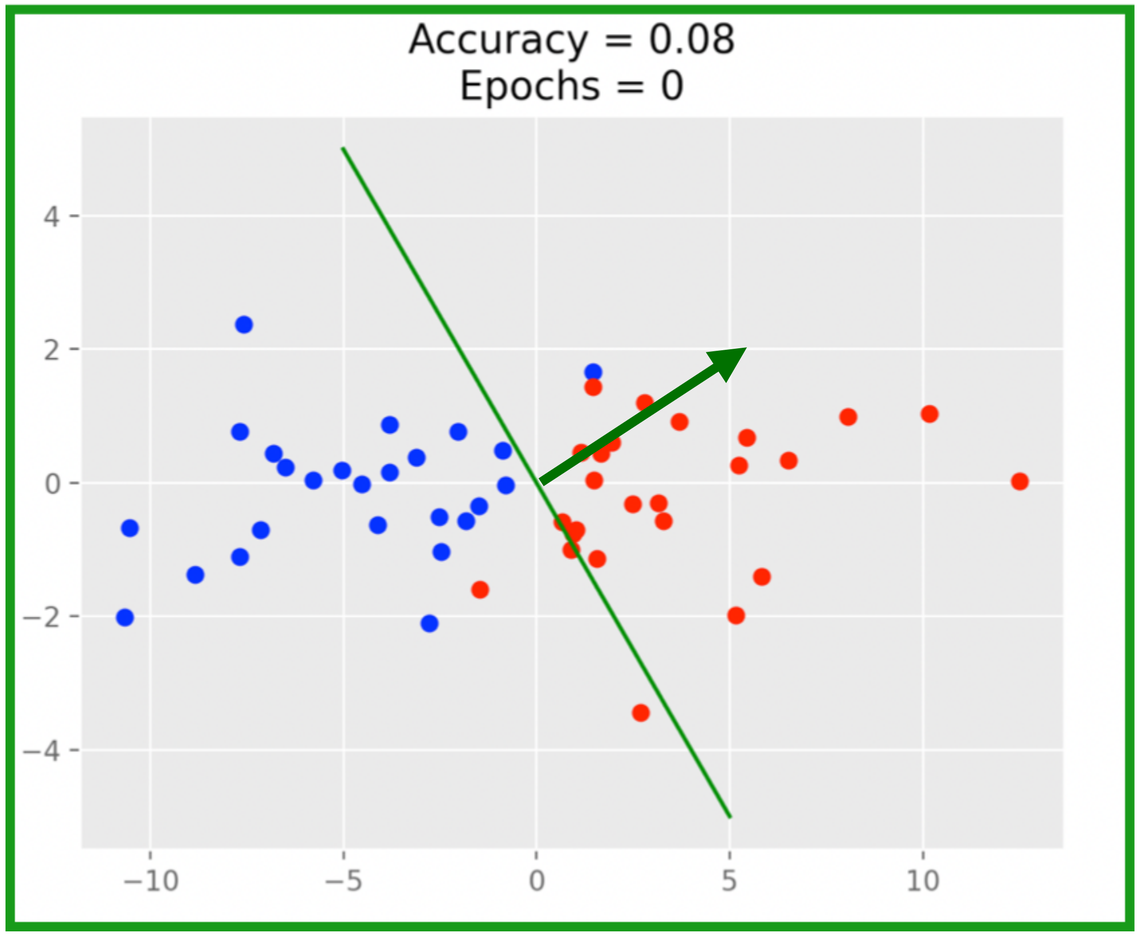
$$\theta_2 := \theta_2 + \left[\frac{1}{N}\sum_{j}^{R}(y_j)(x_j[2])\right] - \lambda\theta_2$$

$$\theta_1 := \theta_1 + \left[\frac{1}{N}\sum_{j}^{R}(y_j)(x_j[1])\right] - \lambda\theta_1$$

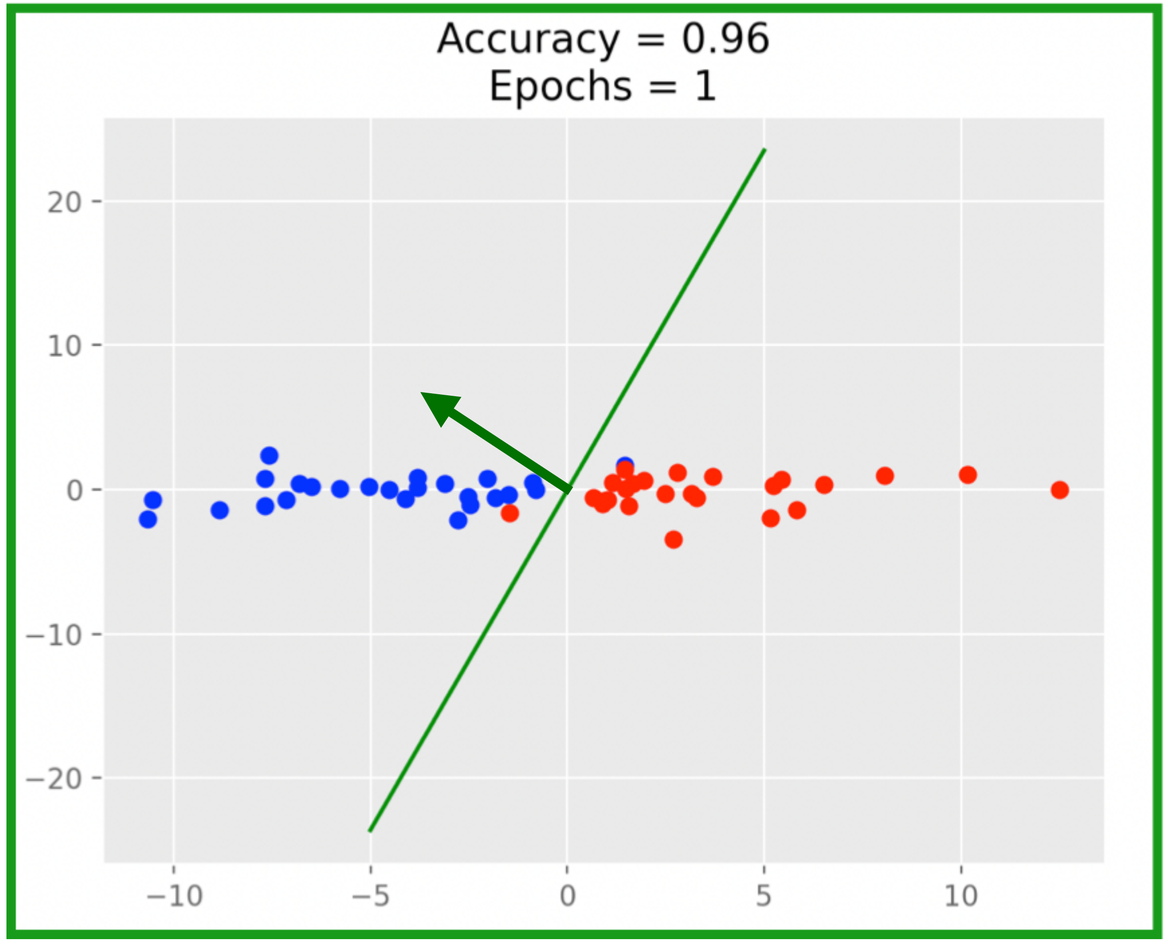
$$\theta_0 := \theta_0 + \frac{1}{N}\sum_{j}^{R} y_j$$

We don't regularize the bias because data that isn't centered at 0 may need a large bias to avoid underfitting.

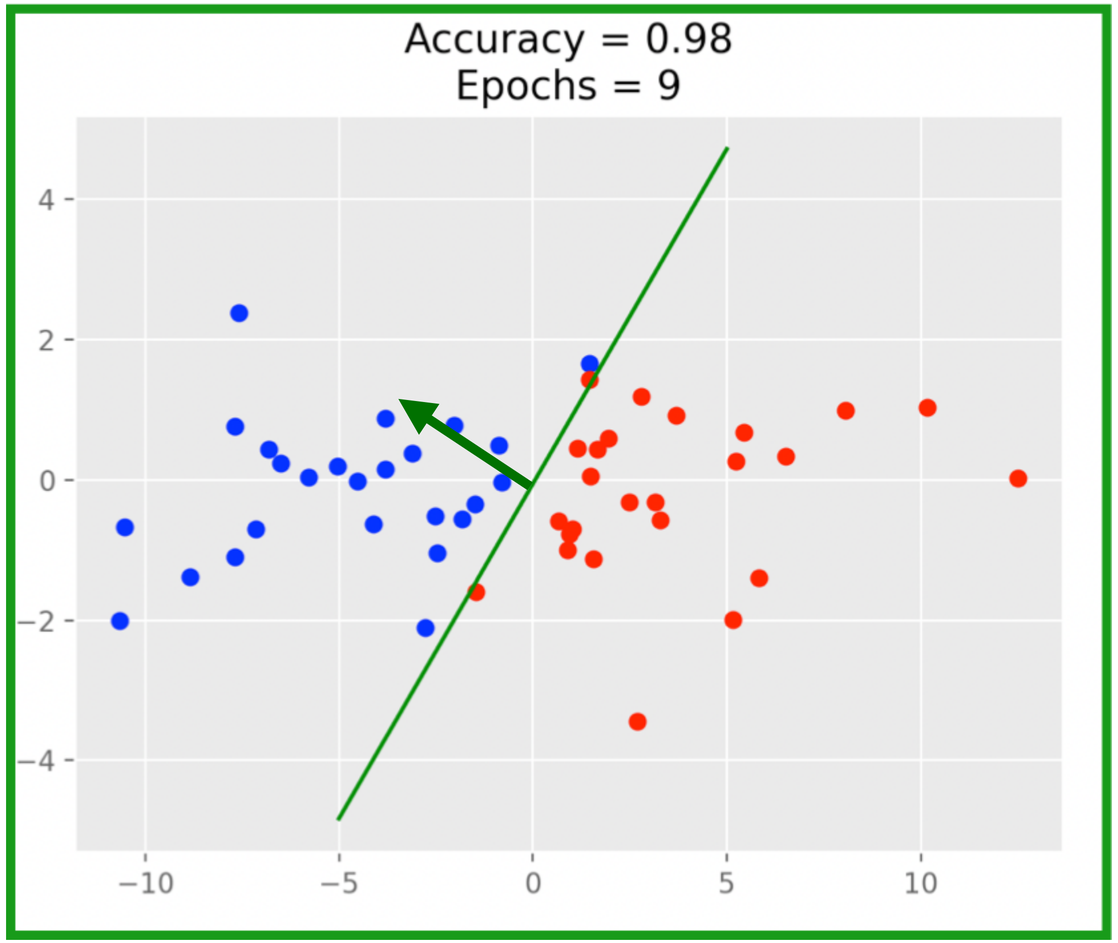
Machine Learning | Hyperplanes



Machine Learning | Hyperplanes



Machine Learning | Hyperplanes



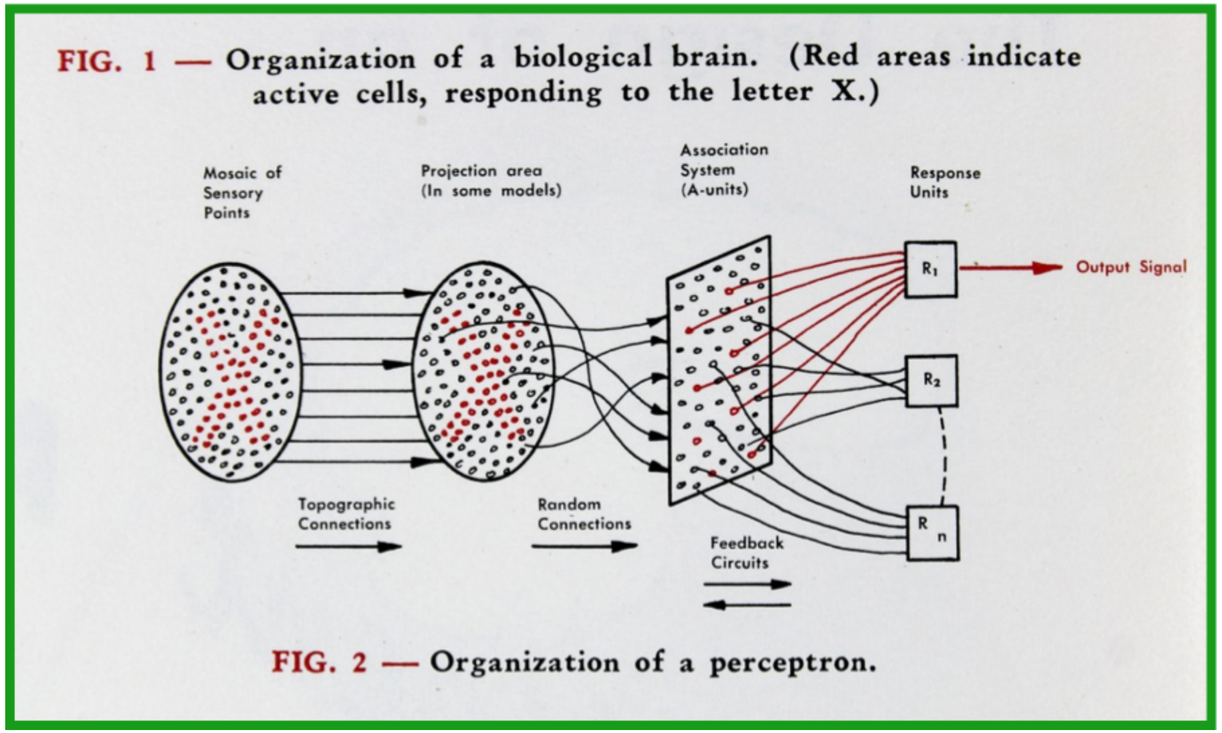
Machine Learning | Hyperplanes

With our new regulating term in our objective function, controlled by $\lambda \in [0,1]$, our final model after 9 epochs has a **smaller magnitude** θ!

This prevents overfitting, and makes our model simpler to achieve this feature

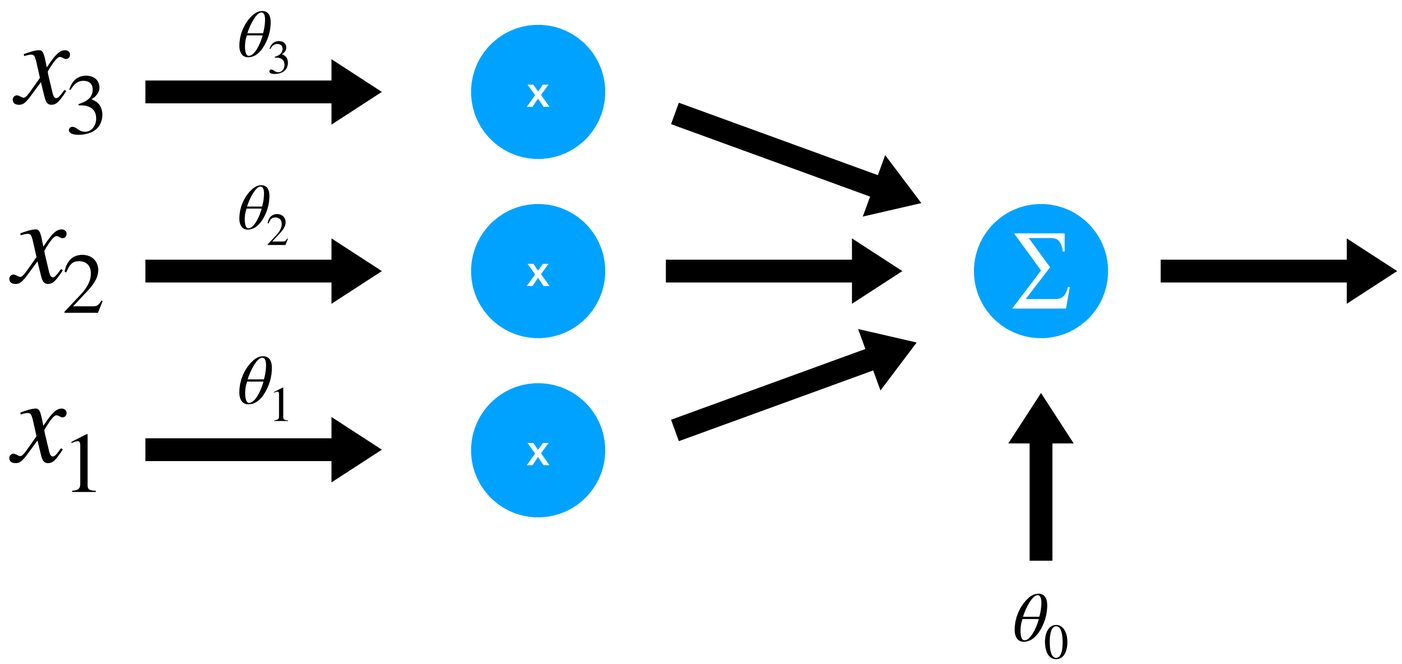
Machine Learning | Perceptron

What we've just implemented is known as **Perceptron**, one of the earliest linear classifiers, and the precursor to Neural Networks



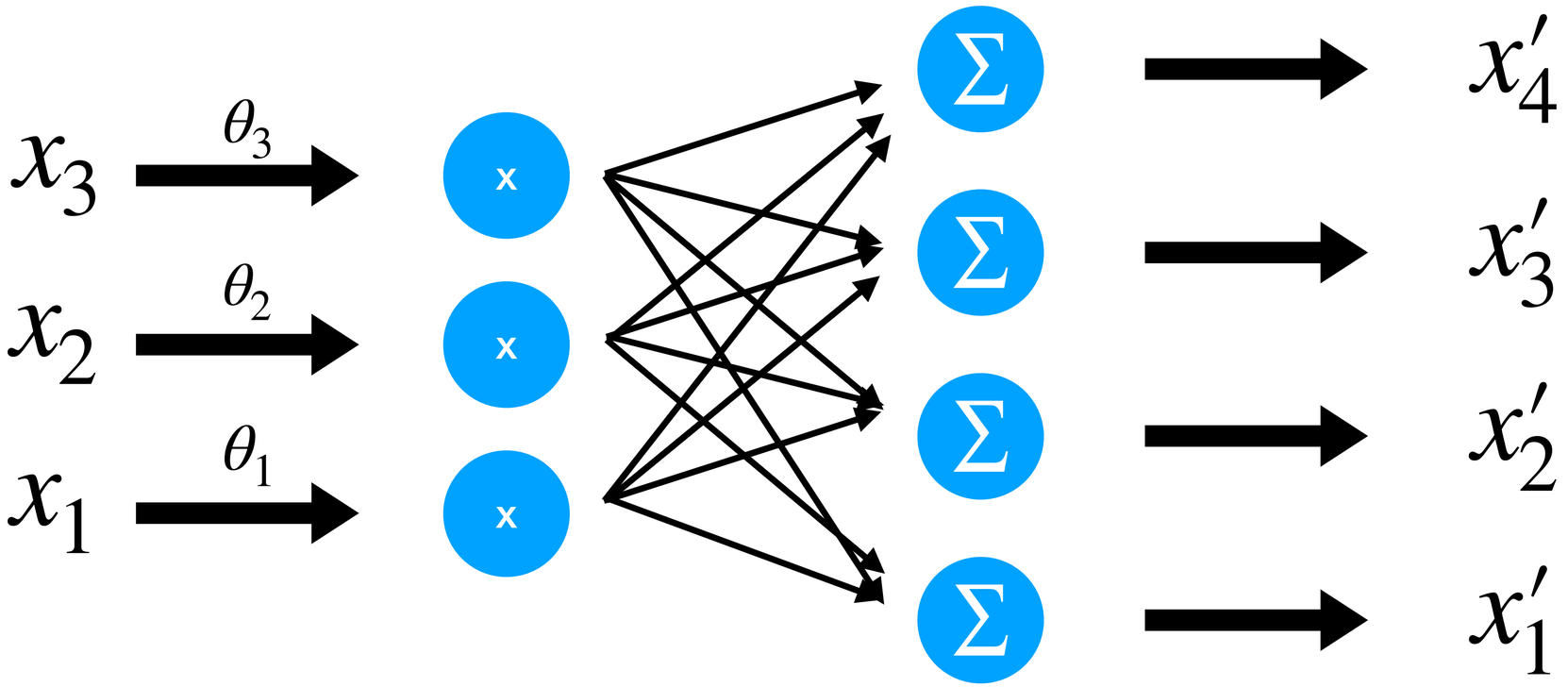
Machine Learning | Perceptron

The **Perceptron** has a certain **input dimensionality**, as well as an **output dimensionality**



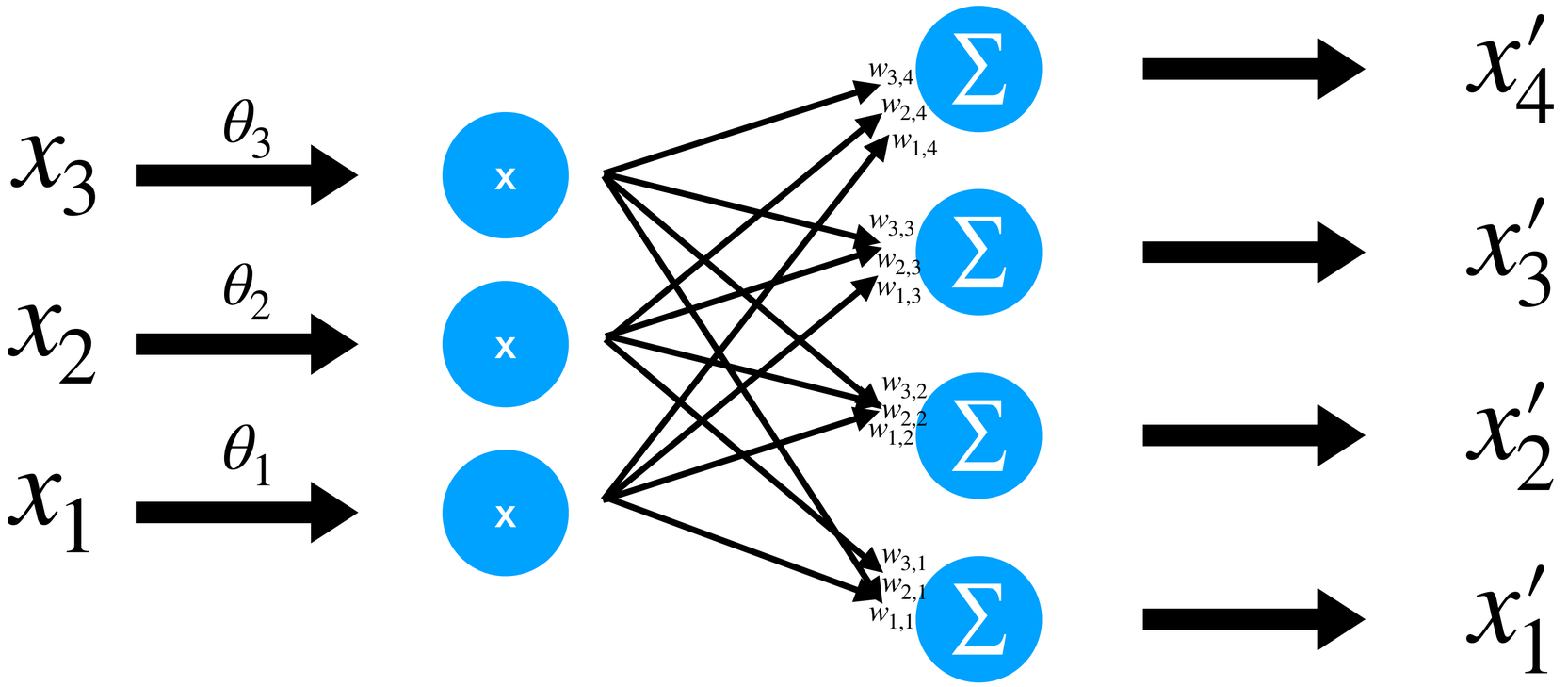
Machine Learning | Neural Networks

Neural **Layers** in networks can produce a more complex **output dimensionality** than the input as well:



Machine Learning | Neural Networks

Each connection has its own **weight**, leading to the requirement of **several** different model attributes or **weights**, not just θ



Machine Learning | Neural Networks

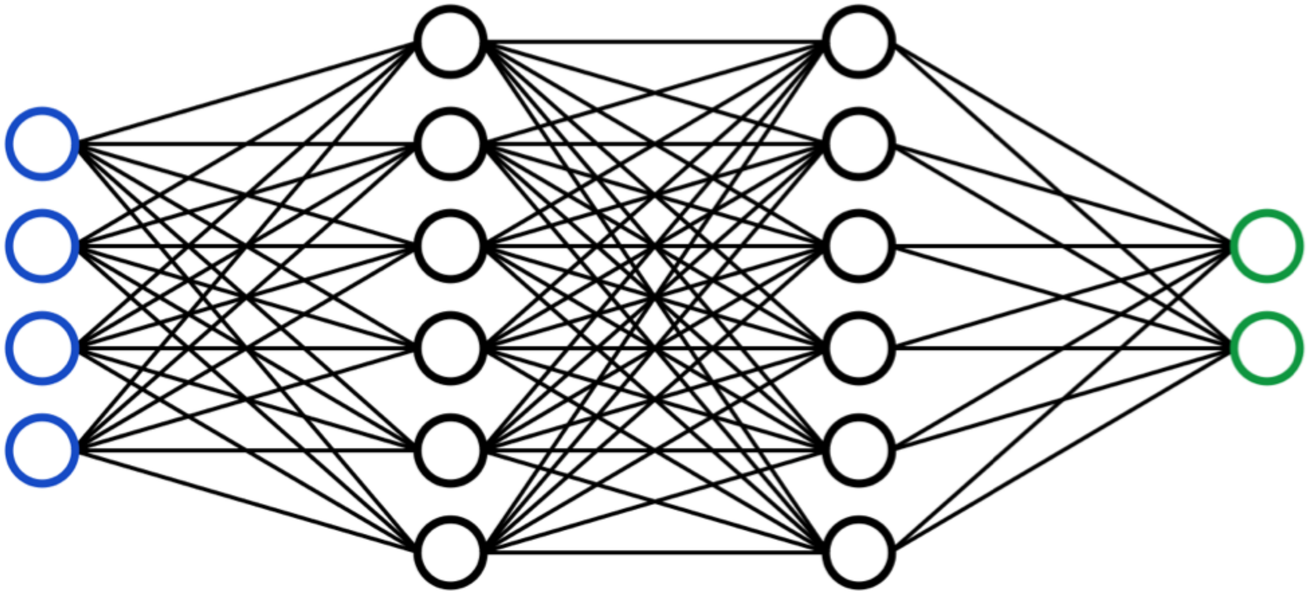
These computations end up being most easily computed using **matrix multiplication**, storing the weights of each layer in a weight matrix mapping input dimensionality to output dimensionality $W_{i \times o}$

For an input dimensionality of 3 and an output dimensionality of 1, our weight matrix:

$$W_{3\times1} = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$ for perceptron

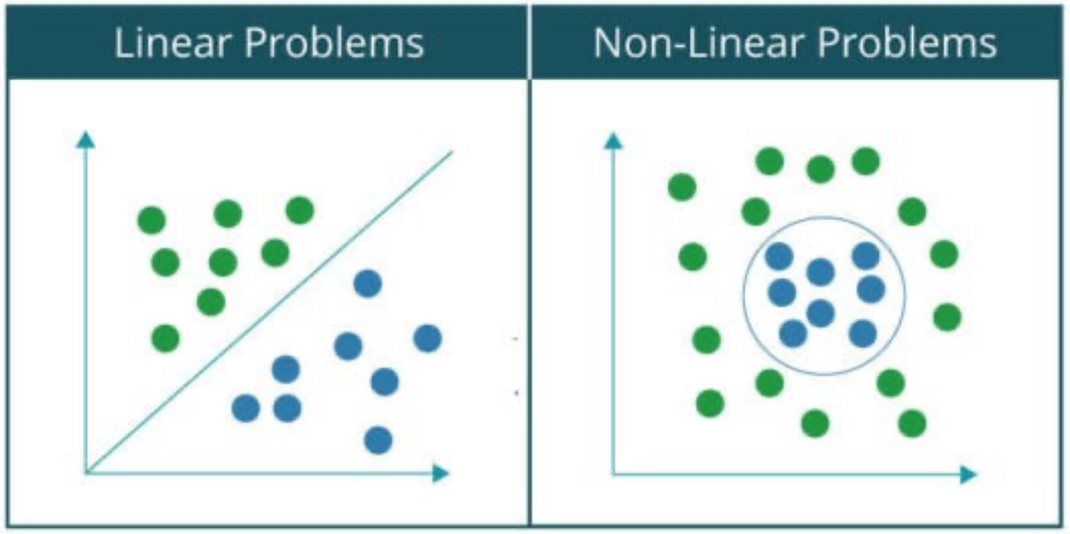
Machine Learning | Neural Networks

Neural Networks end up being several **layers** of modified Perceptrons, with an output layer that can **classify items** for us!



Machine Learning | Clustering

Though Neural Networks can modify the dimensionality of our information between layers, there are certain cases in which we'd like to classify **non-linearly discriminable data** using models with less complexity



Machine Learning | Clustering

KNN or (**K-Nearest-Neighbors**) is a means of classification in which we predict a data point's classification based on the classification of **K** data points that are spatially closest to it

To determine what we can classify a point x_i as, we need to rank our other datapoints by distance to x_i and choose the majority label