Perceptron Learning

Module 1, Tue. September 9, 2025

Overview of Material

- Tangent: Shortcut Definitions
- Neuroscience
  - A very brief history of neuroscience
  - What do we know about the brain?
  - Tangent: Our perception of early pioneers
- The Perceptron Learning Rule
- Homework Assignment:
  - Perceptron Learning Rule
  - Solving example problem
- Feature Space:
  - A very brief introduction
  - The Curse of Dimensionality
  - Discriminant Functions

Overview of Material

- Tangent: Shortcut Definitions
- Neuroscience
  - A very brief history of neuroscience
  - What do we know about the brain?
  - Tangent: Our perception of early pioneers
- The Perceptron Learning Rule
- Homework Assignment:
  - Perceptron Learning Rule
  - Solving example problem
- Feature Space:
  - A very brief introduction
  - The Curse of Dimensionality
  - Discriminant Functions

Building Blocks:

- Perceptron
- Perceptron Learning
- Dot-Product
- Decision Boundaries
- Feature Space
- Feature Vectors

Overview of Material

- Tangent: Shortcut Definitions
- Neuroscience
  - A very brief history of neuroscience
  - What do we know about the brain?
  - Tangent: Our perception of early pioneers
- The Perceptron Learning Rule
- Homework Assignment:
  - Perceptron Learning Rule
  - Solving example problem
- Feature Space:
  - A very brief introduction
  - The Curse of Dimensionality
  - Discriminant Functions

Building Blocks:

- Perceptron
- Perceptron Learning
- Dot-Product
- Decision Boundaries
- Feature Space
- Feature Vectors

Good News:
On a per-lecture basis, in the currency of LBBs (learned building blocks), you're getting more value today out of this one lecture than probably any other of the lectures in this course.

Overview of Material

- Tangent: Shortcut Definitions
- Neuroscience
  - A very brief history of neuroscience
  - What do we know about the brain?
  - Tangent: Our perception of early pioneers
- The Perceptron Learning Rule
- Homework Assignment:
  - Perceptron Learning Rule
  - Solving example problem
- Feature Space:
  - A very brief introduction
  - The Curse of Dimensionality
  - Discriminant Functions

Building Blocks:

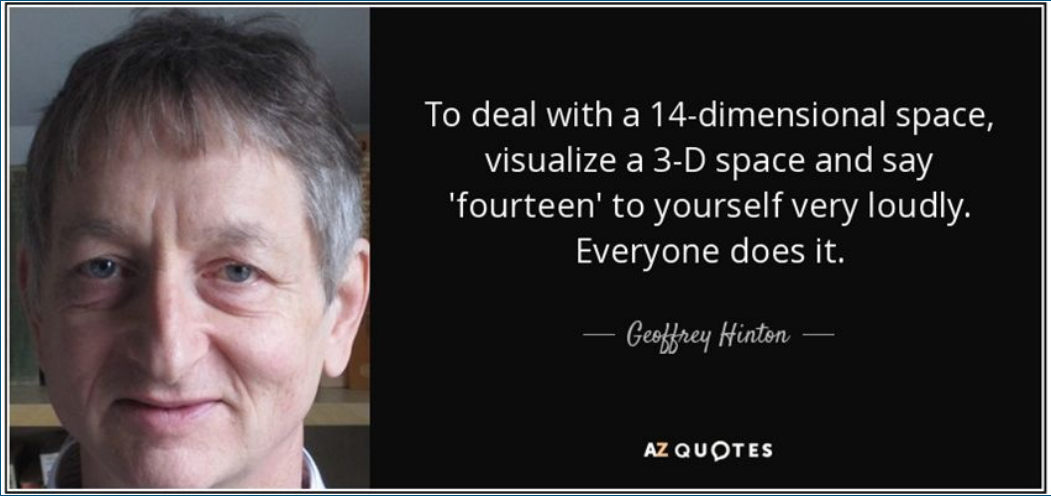
- Perceptron
- Perceptron Learning
- Dot-Product
- Decision Boundaries
- Feature Space
- Feature Vectors

Good News:
On a per-lecture basis, in the currency of LBBs (learned building blocks), you're getting more value today out of this one lecture than probably any other of the lectures in this course.

Bad News:
You're going to question your choice to take this class. You're going to wonder if you hate the instructor. You're going to wonder where this headache came from.

How to best make use (in the near-term) of today's concepts

Shortcut Definitions



I heard this quote a long time ago and found it very useful.

I think there are several similar *shortcut definitions* that might be useful when learning machine learning.

I started gathering them and putting them in my notebook.

My shortcut definitions aren't as useful or funny as Hinton's.

Shortcut Definitions

Machine Learning Shortcuts: Approximate Definitions of Complicated Concepts

When you see this ...		Think this ...
Cluster	→	Tendency to hang out together as a group
Component	→	One of the elements of a vector.
Conditioning on X	→	Acknowledging that X has occurred.
Decoder(Y)	→	A function that takes an encoded object, Y, and reverses whatever that cncoder did to produce it.
Discriminant	→	A classifier.
Embedding	→	A low-dimensional projection of a high-dimensional object that manages to retain most of the object's important information.
Encoder(X)	→	A function that compresses X, retaining the good stuff.
Estimator	→	An algorithmic guess.
Expected value, E[X]	→	Average(X)
Feature	→	A distinguishing characteristic: Charm
Feature Extractor	→	Process for converting something complicated into a few simple but descriptive characteristics.
Gradient	→	The direction of steepest descent.
Gradient Descent	→	An algorithm for optimization in which you descend down a surface hopefully arriving at a valley.
Heuristic	→	Rule that just seems to work.
Hyperplane	→	A dividing line in feature space
Hyperspace or N-dimensional feature space	→	3-dimensional space
Induction	→	Going from specific to general
Integral of a function	→	The accumulation or sum of the function's value
Kernel	→	A measure of similarity

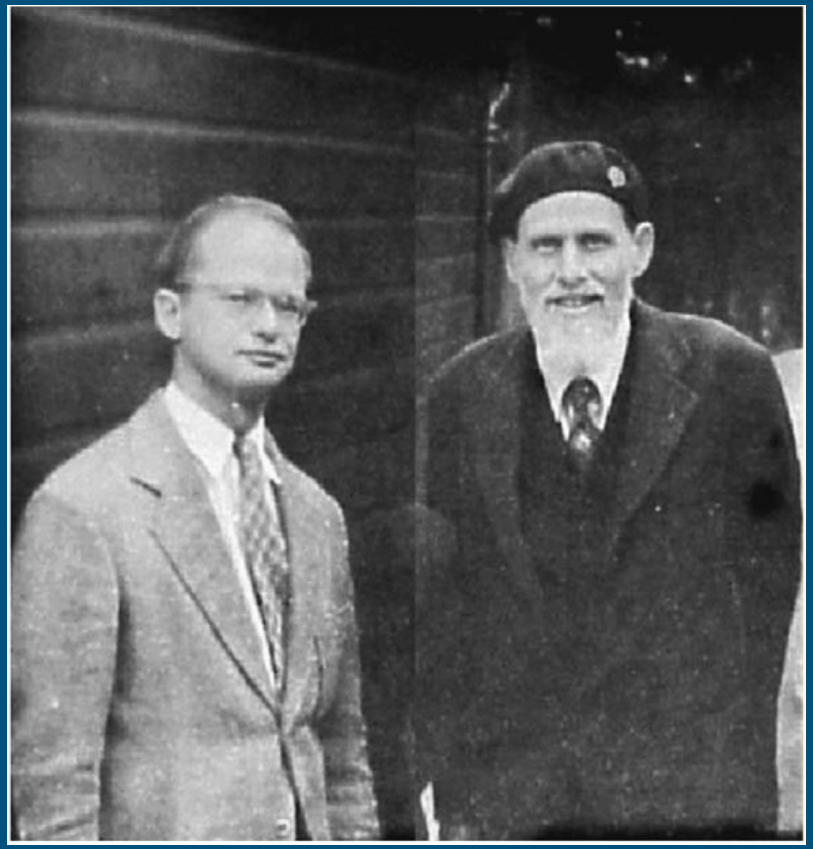
When you see this ...		Think this ...
Manifold	→	A low-dimensional space in which most of the relevant and useful information is preserved from a corresponding high-dimensionsal feature space.
Natural structure	→	Items seem to cluster
Odds	→	Ratio of number of chances for an event to happen vs. the number of chances for it *not* to happen. Rolling a 2: Odds are 1:5; probability is 1/6
Posterior	→	Probability after you account for your observations and priors.
Priors	→	Your existing beliefs (before you make any observations).
Probability	→	A measure (from 0% to 100%) of how likely it is for some event to happen.
Probability Density Function, p(X)	→	How are the X values distributed? Which values of X are more probable? Think 'histogram'.
Random variable, X	→	Variable, X
Stochastic	→	Random
Supervised	→	Ground-truth is available, so you know the right answer
Training vs. Learning	→	Training is from the perspective of a human looking at a model. Learning is from the perspective of a model. An algorithm trains a model. The model learns a dataset.
Vector Space	→	Feature Space: [Beauty, Smarts, Charm, Money, Personality]
Word Embedding	→	A vector representation of a word that encodes some of the word's meaning and semantics.

A Very Brief History of Neuroscience

Timeline

1936	:	Alan Turing proposed the Turing Machine as a model for universal computation.
1943	:	McCulloch and Pitts publish: *A Logical Calculus of the Ideas Immanent in Nervous Activity*. One of McP's intentions was to create a simple model of neurons that could be used to build a Turing Machine.
1949	:	Donald Hebb published "Organization of Behavior", which explains how neural connections are strengthened through experience (Hebbian learning).
1950	:	Alan Turing proposed Turing Test for Artificial Intelligence
1952	:	Hodgkin and Huxley measure and model membrane potentials with differential equations.
1957	:	Frank Rosenblatt proposes the Perceptron (based on McP's neuron)

McCulloch-Pitts Neuron (McP)

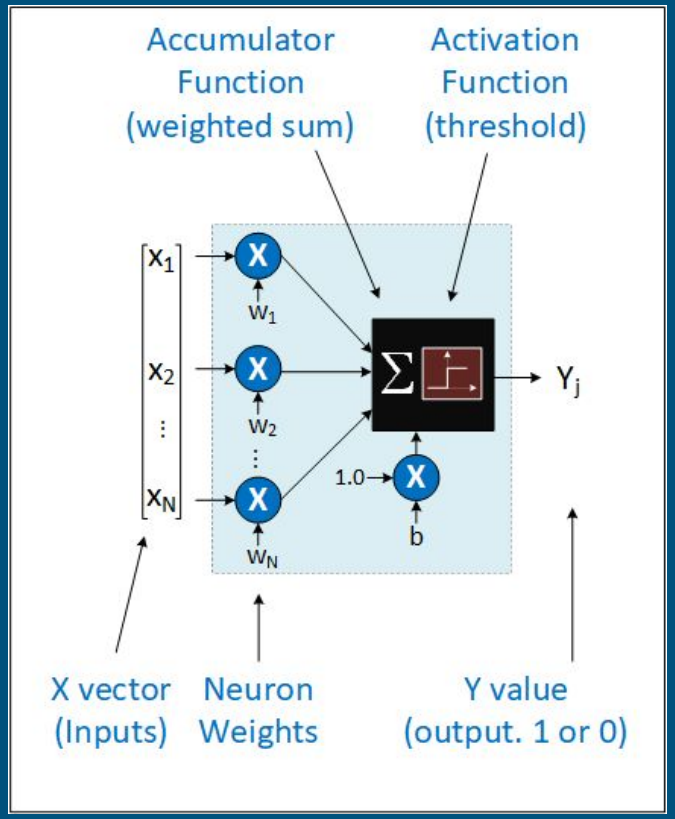


In 1943, McCulloch (neurologist) & Pitts (logician) published the first mathematical model of a neural network.

In their groundbreaking paper, McCulloch and Pitts start with 4 facts from the neuroscience known at the time:

1. The brain is composed of neurons
2. Neurons connect through synapses
3. Neurons can send either excitatory or inhibitory signals to each other
4. A neuron only fires when it reaches a certain threshold of excitatory input signals

McCulloch-Pitts Neuron (McP)



This is the model for the McP neuron.

It takes as input a vector, X.

It produces as output a scalar, Y.

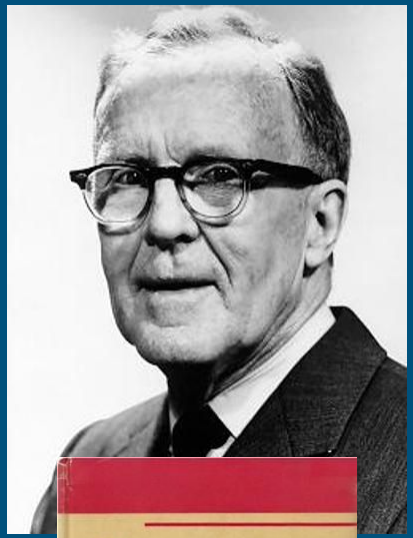
It computes two functions:

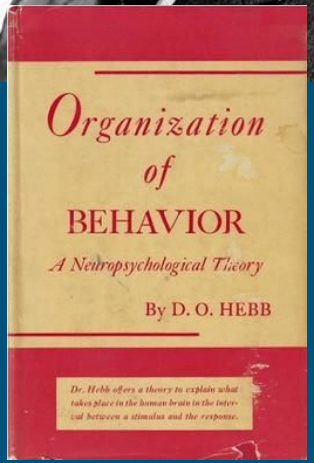
- Accumulator function (weighted sum)
- Activation function (threshold)

Timeline

1936	:	Alan Turing proposed the Turing Machine as a model for universal computation.
1943	:	McCulloch and Pitts publish: *A Logical Calculus of the Ideas Immanent in Nervous Activity*. One of McP's intentions was to create a simple model of neurons that could be used to build a Turing Machine.
1949	:	Donald Hebb published "Organization of Behavior", which explains how neural connections are strengthened through experience (Hebbian learning).
1950	:	Alan Turing proposed Turing Test for Artificial Intelligence
1952	:	Hodgkin and Huxley measure and model membrane potentials with differential equations.
1957	:	Frank Rosenblatt proposes the Perceptron (based on McP's neuron)

Donald Hebb: Hebbian Learning





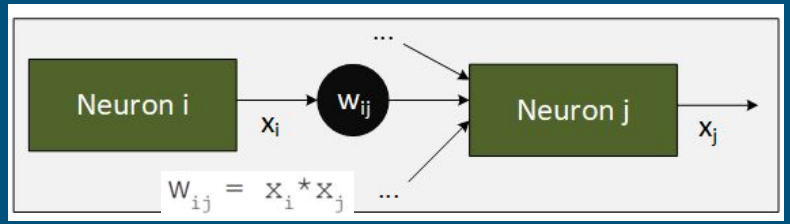
Donald Hebb proposed in 1949 the theory of Hebbian learning, explaining how neural connections are strengthened through experience, a process also known as synaptic plasticity.

Hebb's famous postulate is often summarized as "*neurons that fire together, wire together.*"

"*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased*".

At the time, this was a groundbreaking idea that connected observable psychological behaviors, such as learning and memory, with *a plausible neurological mechanism*.

The Hebb learning rule assumes that – If two neighbor neurons activated and deactivated at the same time. Then the weight connecting these neurons should increase. For neurons operating in the opposite phase, the weight between them should decrease. If there is no signal correlation, the weight should not change.



Timeline

1936	:	Alan Turing proposed the Turing Machine as a model for universal computation.
1943	:	McCulloch and Pitts publish: *A Logical Calculus of the Ideas Immanent in Nervous Activity*. One of McP's intentions was to create a simple model of neurons that could be used to build a Turing Machine.
1949	:	Donald Hebb published "Organization of Behavior", which explains how neural connections are strengthened through experience (Hebbian learning).
1950	:	Alan Turing proposed Turing Test for Artificial Intelligence
1952	:	Hodgkin and Huxley measure and model membrane potentials with differential equations.
1957	:	Frank Rosenblatt proposes the Perceptron (based on McP's neuron)

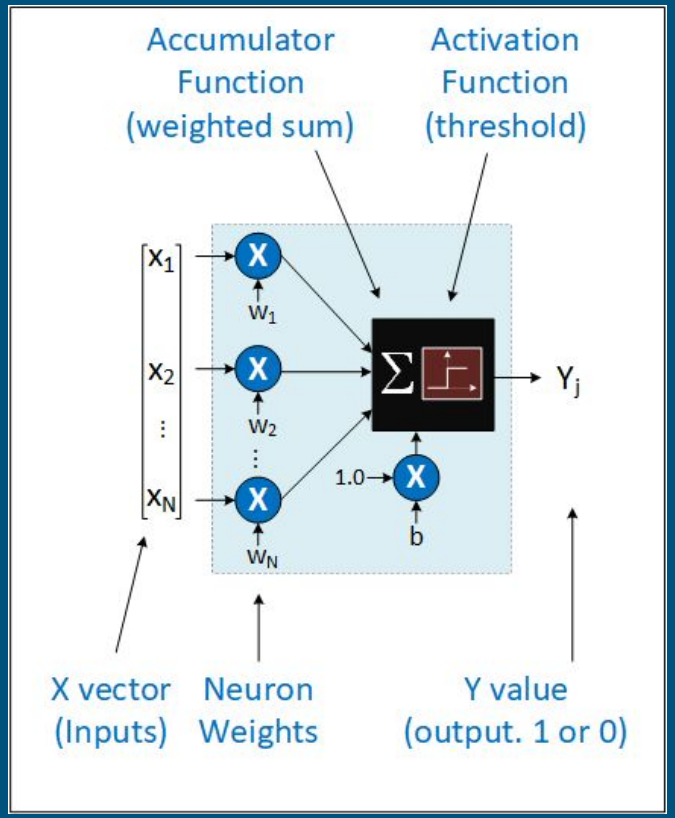
Rosenblatt and the Perceptron

The perceptron was first introduced by American psychologist, Frank Rosenblatt in 1957 at Cornell Aeronautical Laboratory (*shown here in his middle school class photo*).

Rosenblatt was heavily inspired by the biological neuron and its ability to learn.

Rosenblatt's perceptron consists of one or more inputs, a processor, and only one output.

Single-Node Perceptron



This is a single node in a Perceptron.

It takes as input a vector, X.

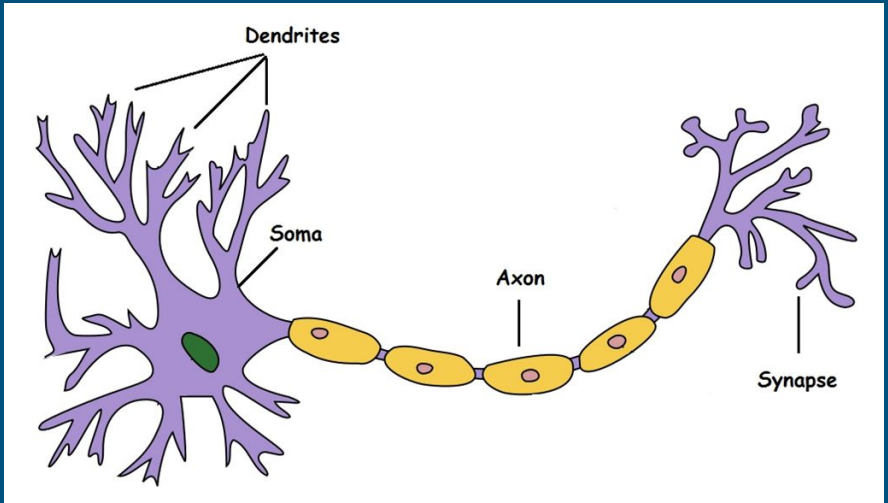
It produces as output a scalar, Y.

It computes two functions:

- Accumulator function (weighted sum)
- Activation function (threshold)

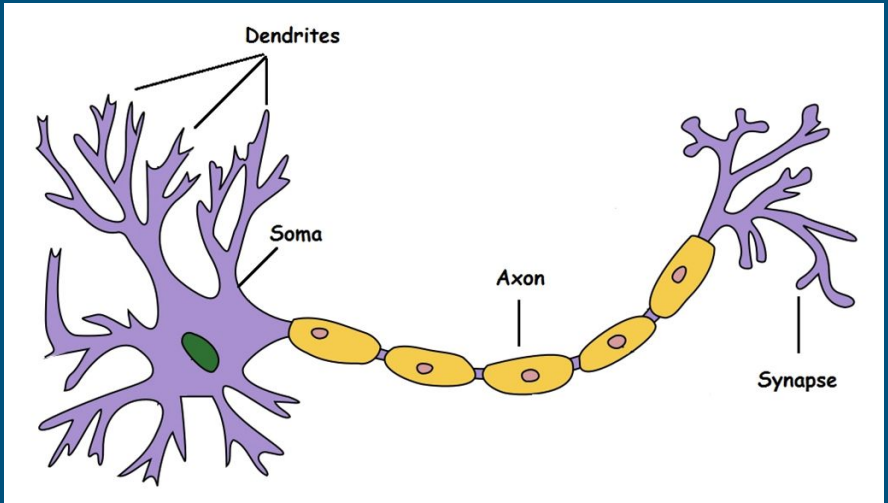
What do we know about the brain?

Neuroscience - Some basis things we know about the brain ...



- The brain has a lot of neurons (~ 100 billion).
- Neurons come in different types, depending on task.
- All neurons operate pretty much the same way:
  - Transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside body of neuron.
  - If this membrane potential reaches some threshold, neuron spikes or fires, and a pulse of fixed strength and duration is sent down the axon.

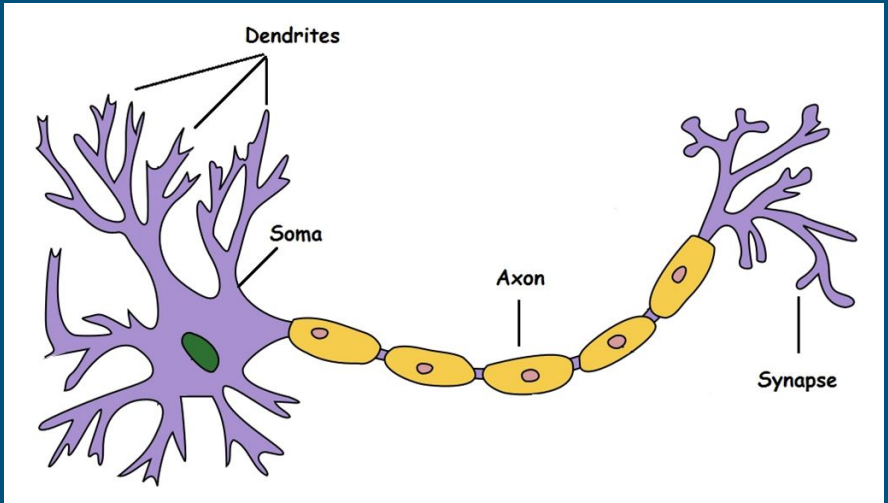
Neuroscience - Some basis things we know about the brain ...



- The brain has a lot of neurons (~ 100 billion).
- Neurons come in different types, depending on task.
- All neurons operate pretty much the same way:
  - Transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside body of neuron.
  - If this membrane potential reaches some threshold, neuron spikes or fires, and a pulse of fixed strength and duration is sent down the axon.

- The axons divide into connections to many other neurons, connecting to each of these neurons in a synapse.
- Each neuron is typically connected to thousands of other neurons. Estimate: There are about 100 trillion synapses within the brain.
- After firing, the neuron must wait for some time to recover its energy (the refractory period) before it can fire again.

Neuroscience - Some basis things we know about the brain ...

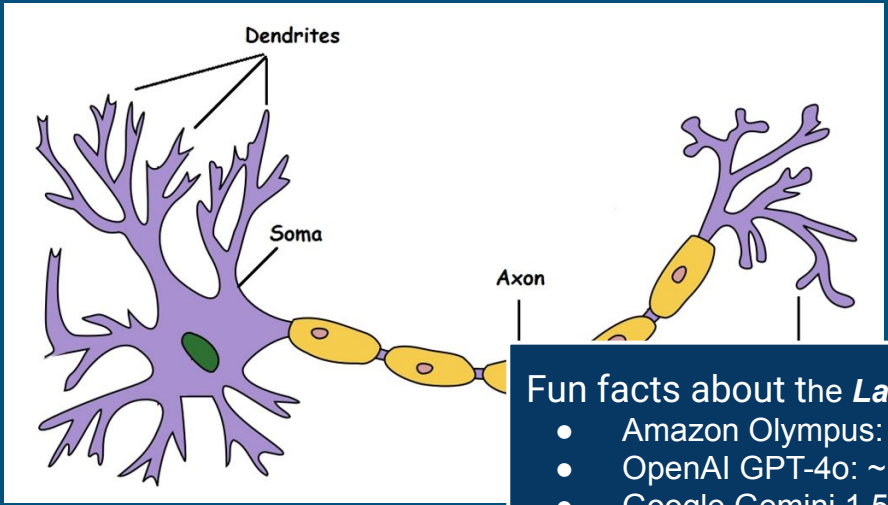


- The brain has a lot of neurons (~ 100 billion).
- Neurons come in different types, depending on task.
- All neurons operate pretty much the same way:
  - Transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside body of neuron.
  - If this membrane potential reaches some threshold, neuron spikes or fires, and a pulse of fixed strength and duration is sent down the axon.

- The axons divide into connections to many other neurons, connecting to each of these neurons in a synapse.
- Each neuron is typically connected to thousands of other neurons. Estimate: There are about 100 trillion synapses within the brain.
- After firing, the neuron must wait for some time to recover its energy (the refractory period) before it can fire again.

- Each neuron can be viewed as a separate processor, performing a very simple computation:
  - Decide whether or not to fire.
- This makes the brain a massively parallel computer made up of 100 billion processing elements.
- The "***capacity***" of the brain in terms of weights or parameters is ~ 100 trillion.

Neuroscience - Some basis things we know about the brain ...



- The brain has a lot of neurons (~ 100 billion).
- Neurons come in different types, depending on task.
- All neurons operate pretty much the same way:
  - Transmitter chemicals within the fluid of the brain raise or lower the electrical potential inside body of neuron.
  - If this membrane potential reaches some …kes or fires, and a pulse of …ration is sent down the

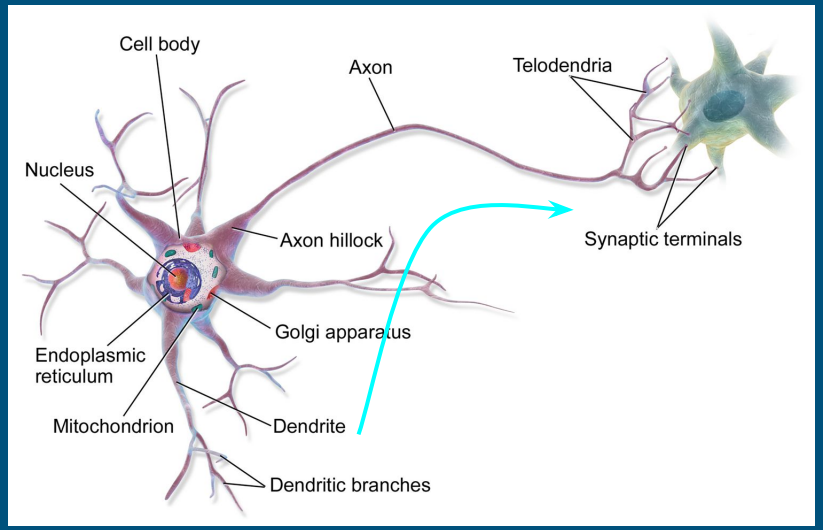
Fun facts about the ***Largest*** Large Language Models:

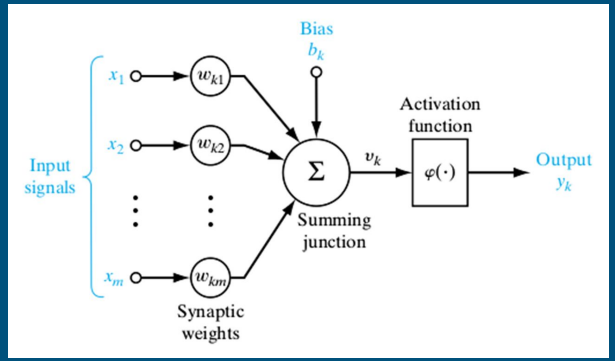
- Amazon Olympus: Trained with 2 trillion parameters.
- OpenAI GPT-4o: ~ 1.8 trillion parameters (rumored).
- Google Gemini 1.5 Pro: ~ 1.5 trillion params (rumored).

- The axons divide into connections to many other neurons, connecting to each of these neurons in a synapse.
- Each neuron is typically connected to thousands of other neurons. Estimate: There are about 100 trillion synapses within the brain.
- After firing, the neuron must wait for some time to recover its energy (the refractory period) before it can fire again.

- Each neuron can be viewed as a separate processor, performing a very simple computation:
  - Decide whether or not to fire.
- This makes the brain a massively parallel computer made up of 100 billion processing elements.
- The "***capacity***" of the brain in terms of weights or parameters is ~ 100 trillion.

Neuroscience - A simple model of a neuron





Neurons receive signals via the *dendrites* and soma and send out signals down the *axon*.

For most synapses, signals cross from the axon of one neuron to the dendrite of another.

The signaling process is partly electrical and partly chemical.

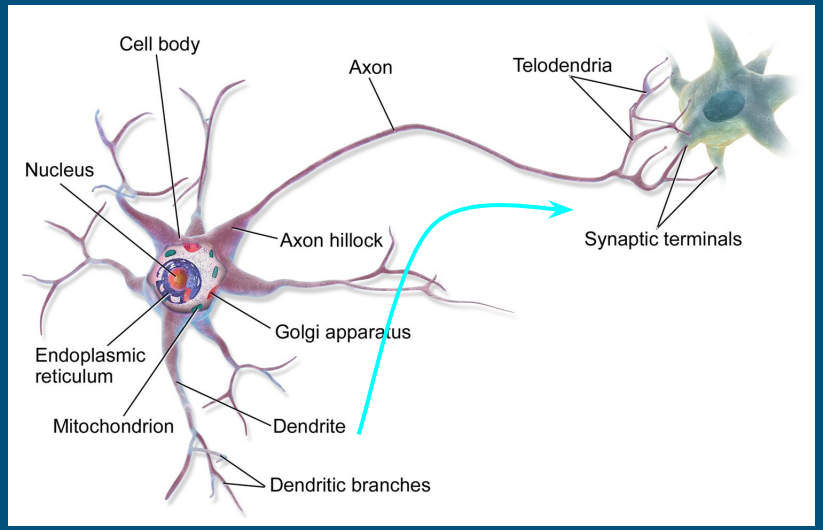
Neurons are electrically excitable, due to the maintenance of voltage gradients across their membranes.

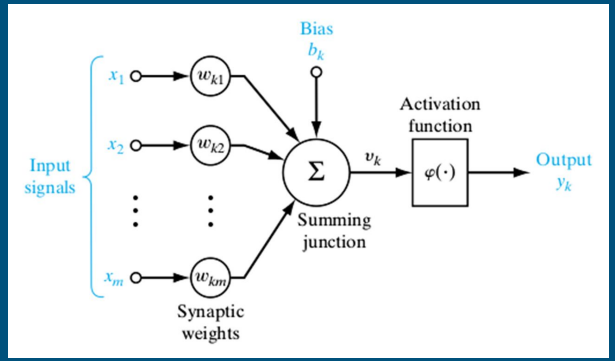
If the voltage changes by a large enough amount over a short interval, the neuron generates an all-or-nothing electrochemical pulse called an action potential.

This potential travels rapidly along the axon and activates synaptic connections as it reaches them.

Synaptic signals may be excitatory or inhibitory, increasing or reducing the net voltage that reaches the soma.

Neuroscience - A simple model of a neuron





Synaptic Efficiency:

If you increase the “strength” of the synaptic connection between two neurons, you are making the connection more efficient.

This is modeled as a scalar weight.

A larger weight means more synaptic strength or efficiency.

A positive weight is an excitatory or amplifying connection, making neuron more likely to fire.

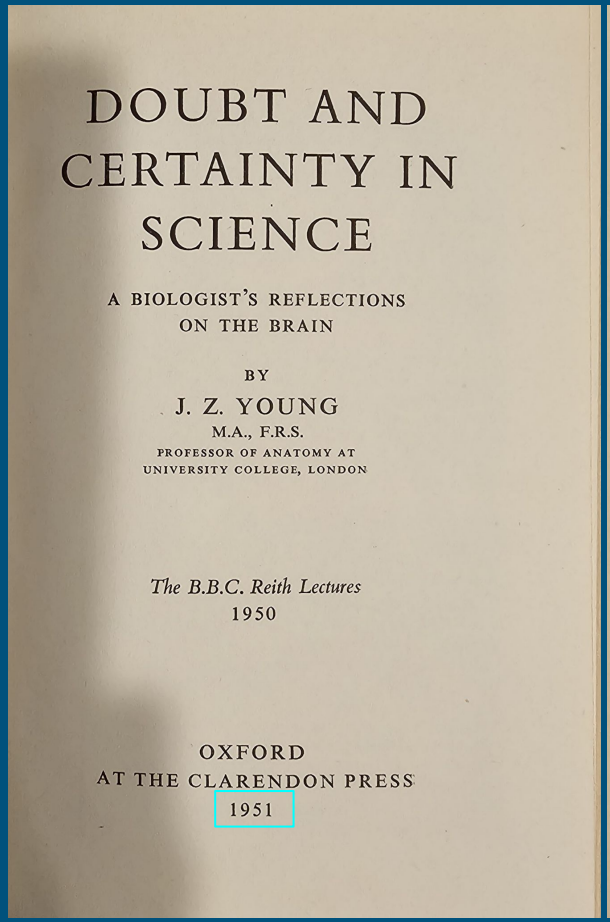
A negative weight is an inhibitory or suppressing connection, making neuron less likely to fire.

Decision to Fire:

Is the total membrane potential (the weighted activations from the source neurons) above some threshold?

The Early Pioneers: They knew a lot more than we give them credit.

What did they know and when did they know it?



DOUBT AND CERTAINTY IN SCIENCE

A BIOLOGIST'S REFLECTIONS ON THE BRAIN

BY

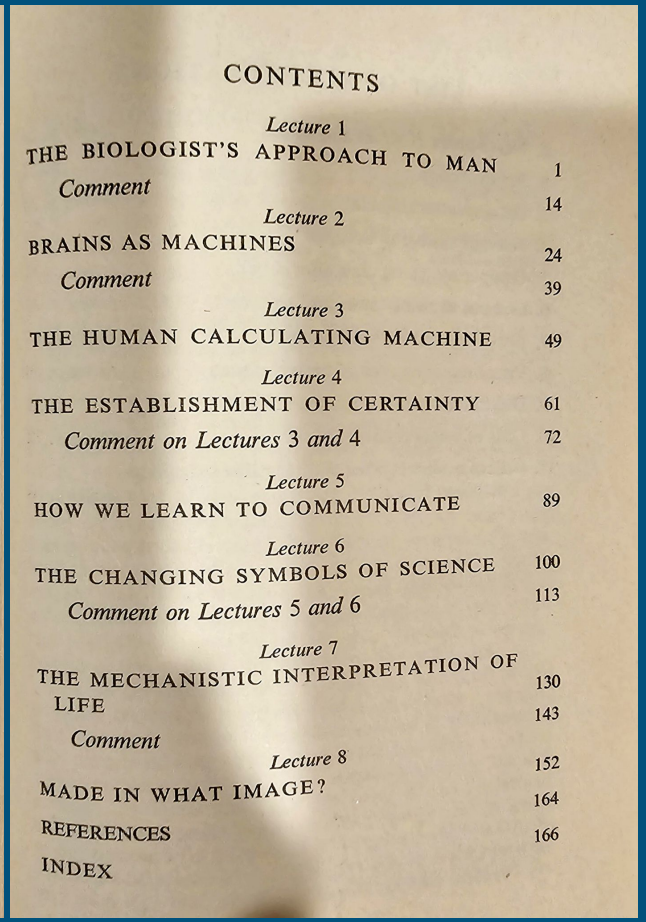
J. Z. YOUNG

M.A., F.R.S.

PROFESSOR OF ANATOMY AT UNIVERSITY COLLEGE, LONDON

The B.B.C. Reith Lectures
1950

OXFORD
AT THE CLARENDON PRESS
1951

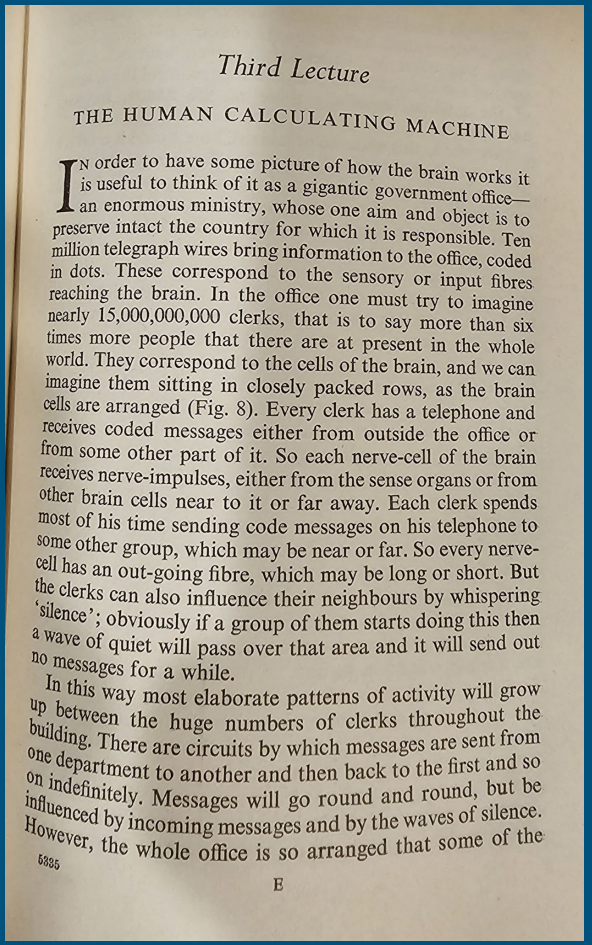


CONTENTS

These are pictures from a book I bought in a used book store a long time ago.

They show that the scientific community had a much better grasp of brains and neural networks than we give them credit.

What did they know and when did they know it?



Third Lecture

THE HUMAN CALCULATING MACHINE

IN order to have some picture of how the brain works it is useful to think of it as a gigantic government office—an enormous ministry, whose one aim and object is to preserve intact the country for which it is responsible. Ten million telegraph wires bring information to the office, coded in dots. These correspond to the sensory or input fibres reaching the brain. In the office one must try to imagine nearly 15,000,000,000 clerks, that is to say more than six times more people that there are at present in the whole world. They correspond to the cells of the brain, and we can imagine them sitting in closely packed rows, as the brain cells are arranged (Fig. 8). Every clerk has a telephone and receives coded messages either from outside the office or from some other part of it. So each nerve-cell of the brain receives nerve-impulses, either from the sense organs or from other brain cells near to it or far away. Each clerk spends most of his time sending code messages on his telephone to some other group, which may be near or far. So every nerve-cell has an out-going fibre, which may be long or short. But the clerks can also influence their neighbours by whispering 'silence'; obviously if a group of them starts doing this then a wave of quiet will pass over that area and it will send out no messages for a while.

In this way most elaborate patterns of activity will grow up between the huge numbers of clerks throughout the building. There are circuits by which messages are sent from one department to another and then back to the first and so on indefinitely. Messages will go round and round, but be influenced by incoming messages and by the waves of silence. However, the whole office is so arranged that some of the

5835

E

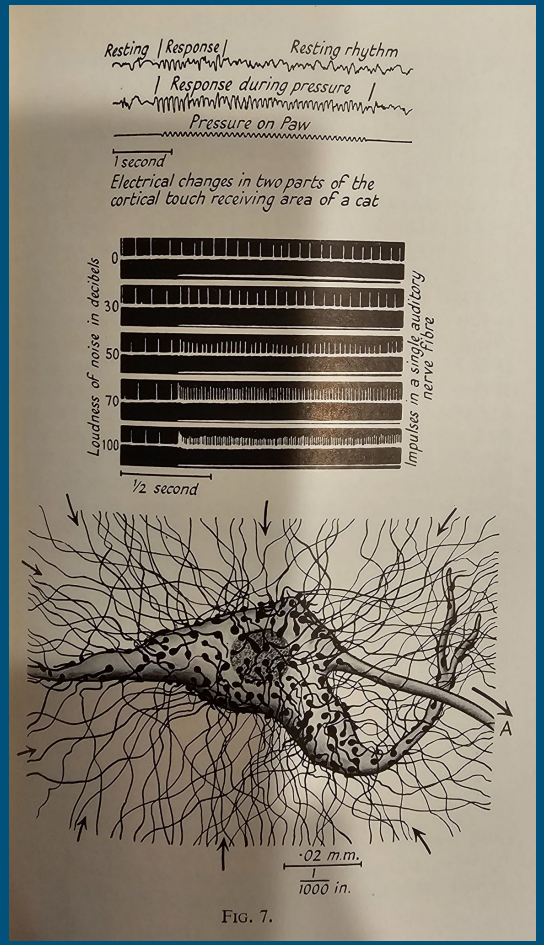


FIG. 7.

These are pictures from a book I bought in a used book store a long time ago.

They show that the scientific community had a much better grasp of brains and neural networks than we give them credit.

What did they know and when did they know it?

FIG. 10. Cells of the cerebral cortex of the cat stained to show the receiving dendrites (D) and the axons (A) along which the output is sent to other regions. The method of staining colours only a selection of the cells (about one in 50); if all those shown in Fig. 9 were fully stained the network would be very much more dense.

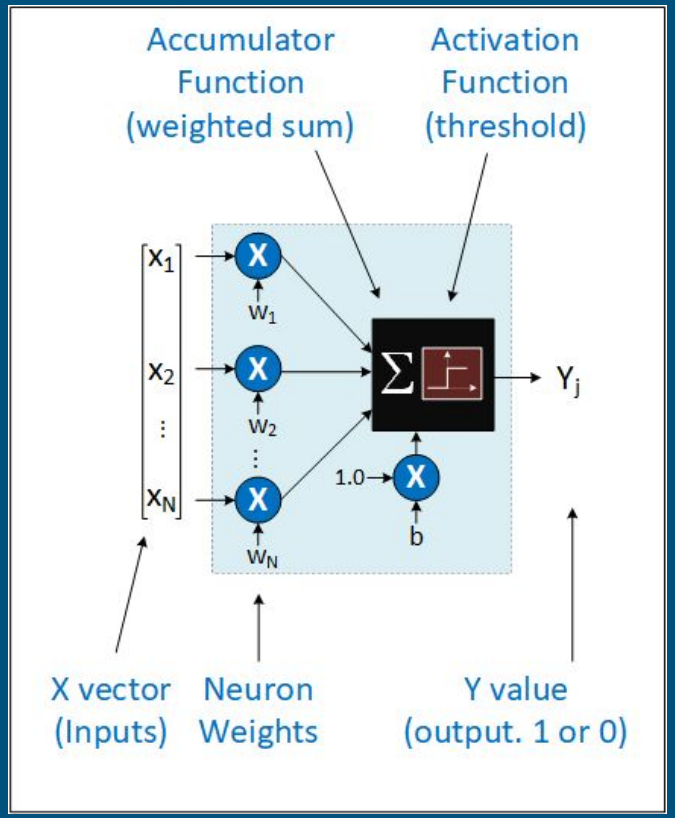
FIG. 11. Cells of the cerebral cortex of the cat stained as in Fig. 10, but shown with a smaller magnification. The interlacing network of receiving dendrites is well seen towards the inner side of the cortex, but would be 50 times more dense if all the cells were stained. The input fibres are not stained.

These are pictures from a book I bought in a used book store a long time ago.

They show that the scientific community had a much better grasp of brains and neural networks than we give them credit.

The Perceptron Learning Rule

Single-Node Perceptron



This is a single node in a Perceptron.

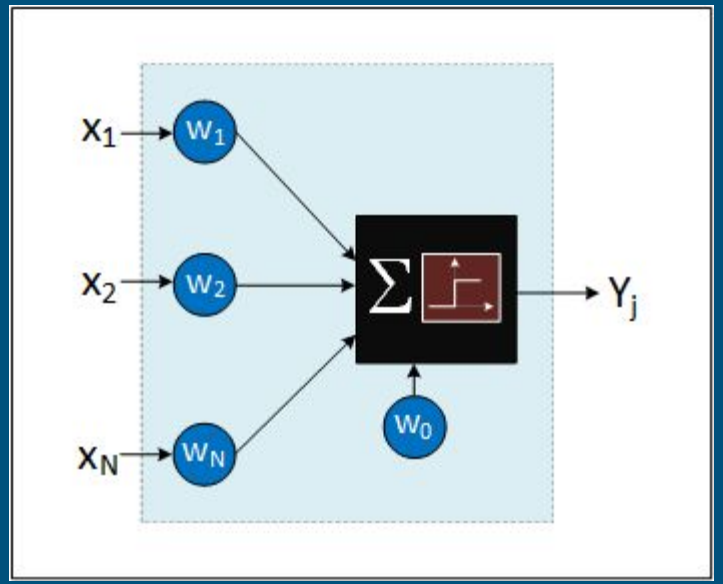
It takes as input a vector, X.

It produces as output a scalar, Y.

It computes two functions:

- Accumulator function (weighted sum)
- Activation function (threshold)

Single-Node Perceptron

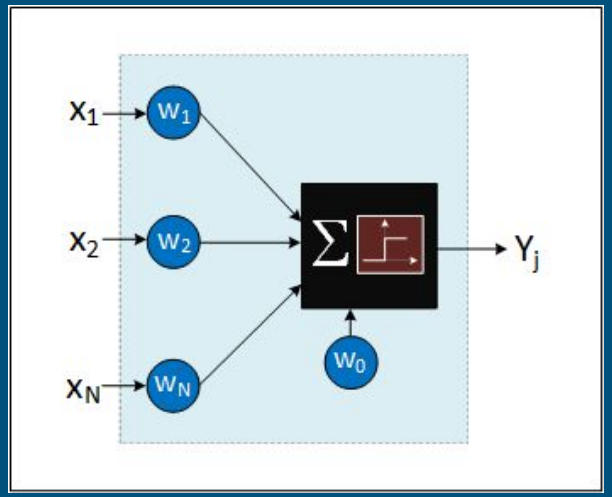


- We're going to simplify the diagram.
- But it's still the same thing, just less formal.

Perceptron Learning Rule: *Without motivating where it came from*

$$\Delta w_i = c(t - y)\, x_i$$

- w_i is the weight from input i to the perceptron node,
- c is the learning rate,
- t is the target for the current instance,
- y is the current output,
- x_i is i^{th} input

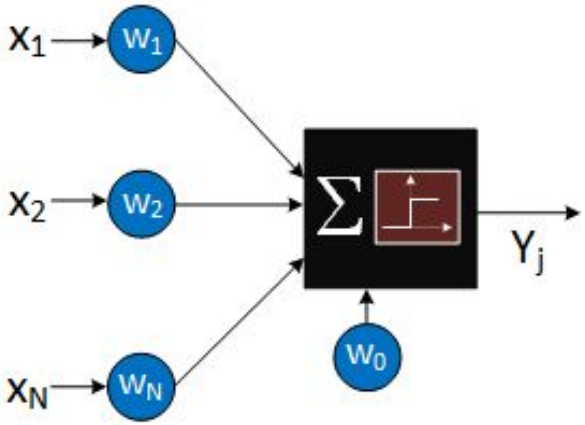


Perceptron Learning Rule: *Without motivating where it came from*

Least perturbation principle:

- Only change weights if there is an error
- Use a small learning rate (to make a small correction), rather than making a large change to the weight that would make current pattern correct
- Scale by input value x_i

Single-Node Perceptron: An *error* is when Y does not match the target value.



$$\varepsilon = Y_j - T_j$$

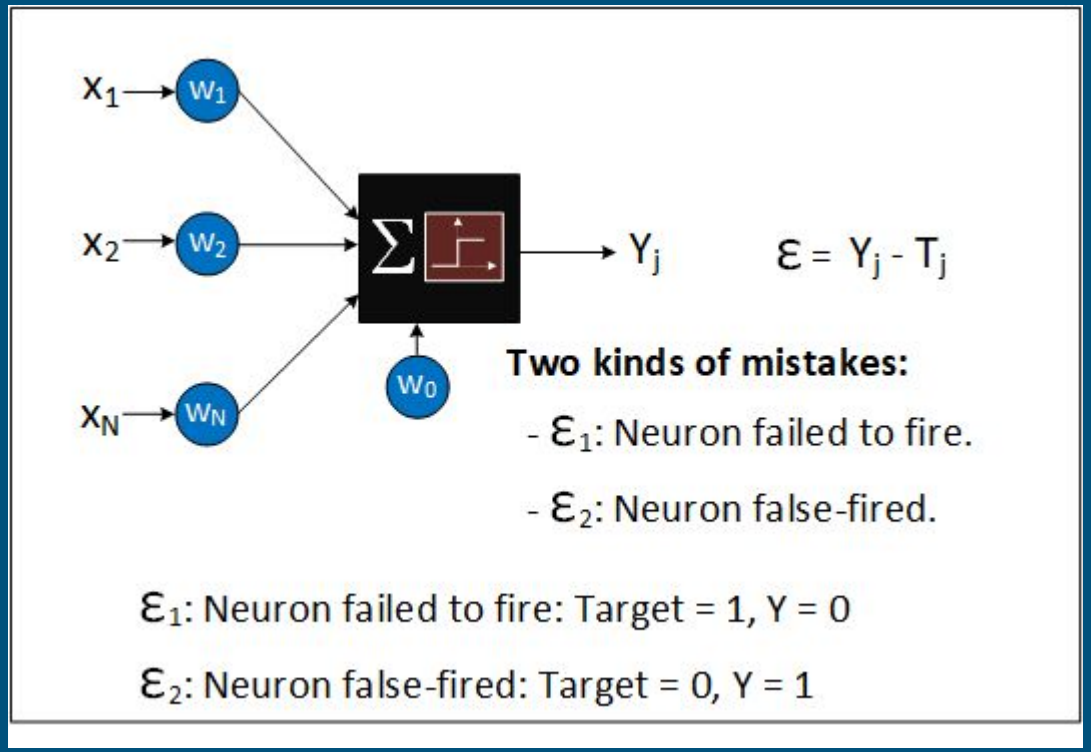
0 → Didn't fire.
1 → Fired.

0 → Not supposed to fire.
1 → Supposed to fire.

Four Possibilities:

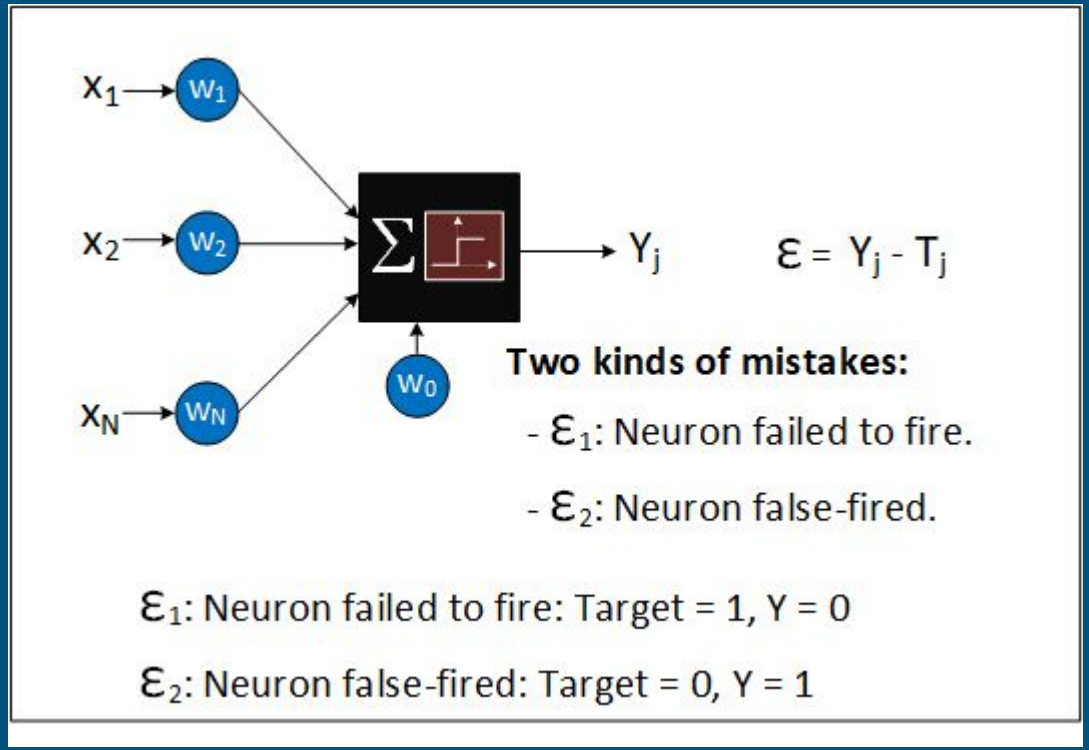
0 - 1 = -1 : Failed to fire
1 – 0 = 1 : False-fire
1 – 1 = 0 : Correctly fired
0 – 0 = 0 : Correctly didn't fire

Single-Node Perceptron: There are two kinds of errors



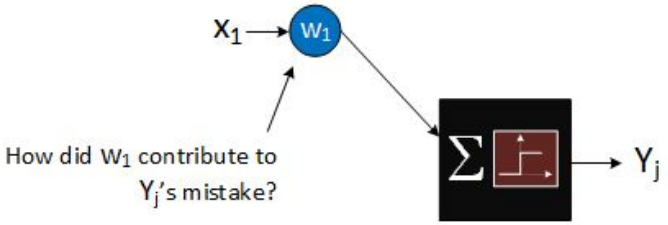
Single-Node Perceptron: Who can we blame?

Who can we blame for these two kinds of errors?!



Single-Node Perceptron: Who can we blame? Answer: The weights.

We blame the weights. They have the wrong values.



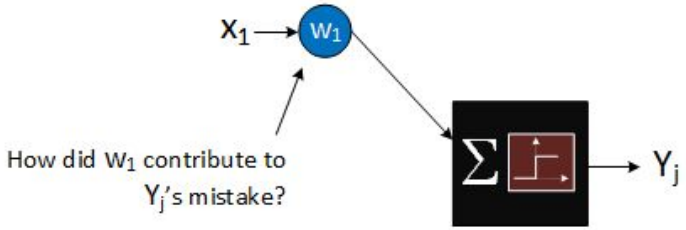
ε_1: Neuron failed to fire: Target = 1, Y = 0

If neuron failed to fire, then weighted sum didn't exceed threshold, which means w_1 inhibited x_1, which means w_1 needs to be larger.

ε_1: Neuron false-fired: Target = 0, Y = 1

If neuron false-fired, then weighted sum exceeded threshold (and shouldn't have), which means w_1 was too excitatory for x_1, which means w_1 needs to be smaller.

Single-Node Perceptron: How can we change w_1 to prevent an error?



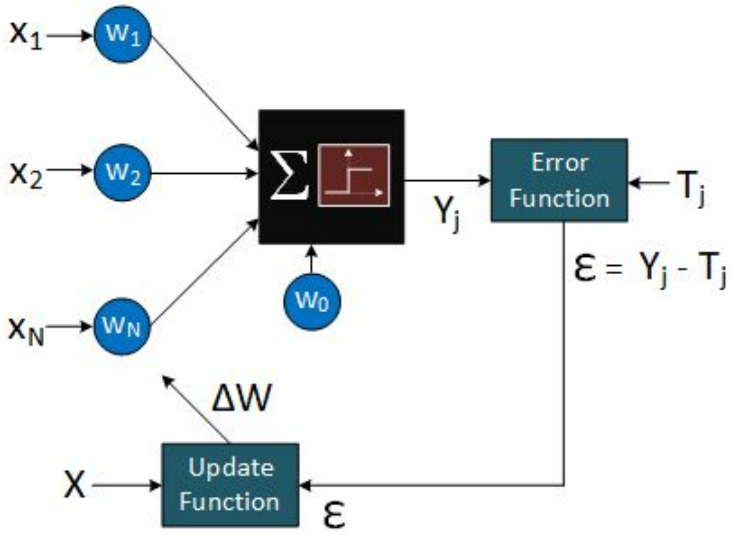
ε_1: Neuron failed to fire: Target = 1, Y = 0

If neuron failed to fire, then weighted sum didn't exceed threshold, which means w_1 inhibited x_1, which means w_1 needs to be larger.

ε_1: Neuron false-fired: Target = 0, Y = 1

If neuron false-fired, then weighted sum exceeded threshold (and shouldn't have), which means w_1 was too excitatory for x_1, which means w_1 needs to be smaller.

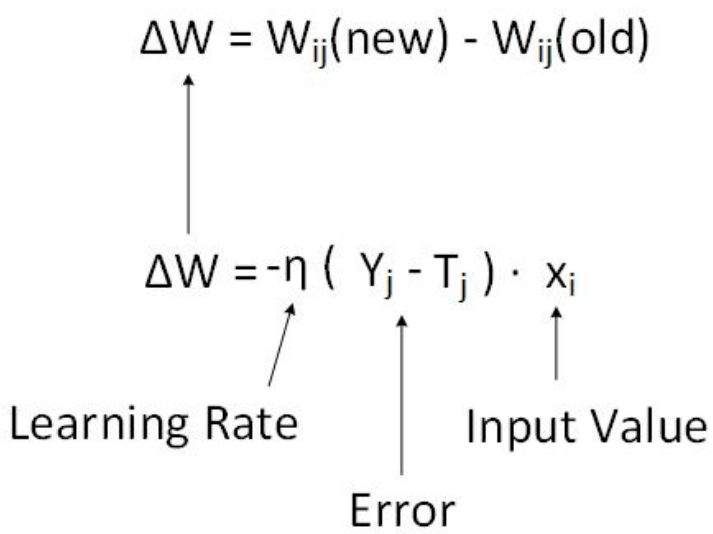
Single-Node Perceptron: There are two kinds of errors



ΔW is the change we make to W to "fix" the error.

Note: We can't ignore x1. It shares some of the blame for the error. But we can't change x1. But its value needs to be considered when we calculate how to change W.

Single-Node Perceptron: Learning Rule



This is the Percepton learning rule.

It tells us how to change one weight, which is:

> *Change W to help make Y match T by computing the error value (Y - T) and then multiplying it by input value, x.*

Single-Node Perceptron: Learning Rule Reality Check

$$\Delta W = -\eta \left(Y_j - T_j \right) \cdot x_i$$

Scenario	Example Values	Output, Y vs. Target, T	Error	What Caused This Error?	sign(X)	Required Change to W	Sign of: - (Yj - Tj) · xi	Update Rule with η = 0.01 ΔW = - η · (Yj - Tj) · xi
Neuron failed to fire	w = 0.1 **x = 1** Y = 0 T = 1	Y - T 0 - 1 -1	Negative (-1)	W, a small *positive* value, is too small. (W inhibits +x too much).	Positive	Add pos number to W to make it a bigger positive number	Positive	0.1 - (0.01) (0 - 1) x 1 0.1 + 0.01 = 0.11
	w = -0.1 **x = -1** Y = 0 T = 1			W, a small *negative* value, is too small. (W inhibits -x too much).	Negative	Add neg number to W to make it a bigger negative number	Negative	-0.1 - (0.01) (0 - 1) x (-1) -0.1 - 0.01 = -0.11
Neuron false-fired	w = 0.9 **x = 1** Y = 1 T = 0	Y - T 1 - 0 1	Positive (1)	W, a large *positive* value, is too large. (W amplifies +x too much).	Positive	Add neg number to W to make it a smaller positive number	Negative	0.9 - (0.01) (1 - 0) x 1 0.9 - 0.01 = 0.89
	w = -0.9 **x = -1** Y = 1 T = 0			W, a large *negative* value, is too large. (W amplifies -x too much).	Negative	Add pos number to W to make it a smaller negative number.	Positive	-0.9 - (0.01) (1 - 0) x (-1) -0.9 + 0.01 = -0.89

Homework Assignment: **Perceptron Learning**

Perceptron Homework Assignment: Fill in the cells to get the weights after step 5

Index	Pattern				Target	Weight Vector				Net Output	Z	LR	Error	Multiplier (Target - Z) x LR	ΔW				Wnew			
	x1	x2	x3	bias		w1	w2	w3	bias						w1	w2	w3	bias	w1	w2	w3	bias
0	1	0.1	0.1	1	0	0.35	-0.2	-0.1	0.25	0.57	1	0.25										
1	0.9	0.1	0.5	1	1							0.25										
2	0.1	0.1	0.1	1	0							0.25										
3	0.1	0.1	0.5	1	1							0.25										
4	0.1	1	0.2	1	0							0.25										
5	0.1	0.9	0.6	1	1							0.25										

Perceptron Homework Assignment: Fill in the cells to get the weights after step 5

Index	Pattern				Target	Weight Vector				Net Output	Z	LR	Error	Multiplier (Target - Z) x LR	ΔW				Wnew			
	x1	x2	x3	bias		w1	w2	w3	bias						w1	w2	w3	bias	w1	w2	w3	bias
0	1	0.1	0.1	1	0	0.35	-0.2	-0.1	0.25	0.57	1	0.25										
1	0.9	0.1	0.5	1	1							0.25										
2	0.1	0.1	0.1	1	0							0.25										
3	0.1	0.1	0.5	1	1							0.25										
4	0.1	1	0.2	1	0							0.25										
5	0.1	0.9	0.6	1	1							0.25										

Single-Node Perceptron:

Index	Pattern				Target	Weight Vector				Net Output	Z	LR	Error	Multiplier (Target - Z) x LR	ΔW				Wnew			
	x1	x2	x3	bias		w1	w2	w3	bias						w1	w2	w3	bias	w1	w2	w3	bias
0	1	1	1	1	0	0	0	0	0	0	0	1	FALSE	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0														
2	0	0	0	1	0	1	0	0														
3	0	1	0	1	1	1	0	0														
4	1	1	0	1	0	1	1	0														
5	0	0	1	1	1	0	0	0														
6	0	1	1	1	0	0	0	1														
7	1	0	1	1	0	0	-1	0														

```
Index 0:
X = [1, 1, 1, Bias=1], W = [0, 0, 0, 0]

Wnew = Wold - LR * (Z - T) * X
     Input1: w1 = 0 - 1 * (0 - 0) * 1 → 0
     Input2: w2 = 0 - 1 * (0 - 0) * 1 → 0
     Input3: w3 = 0 - 1 * (0 - 0) * 1 → 0
     Bias:   b  = 0 - 1 * (0 - 0) * 1 → 0

Wnew = [0, 0, 0, 0]
```

Single-Node Perceptron:

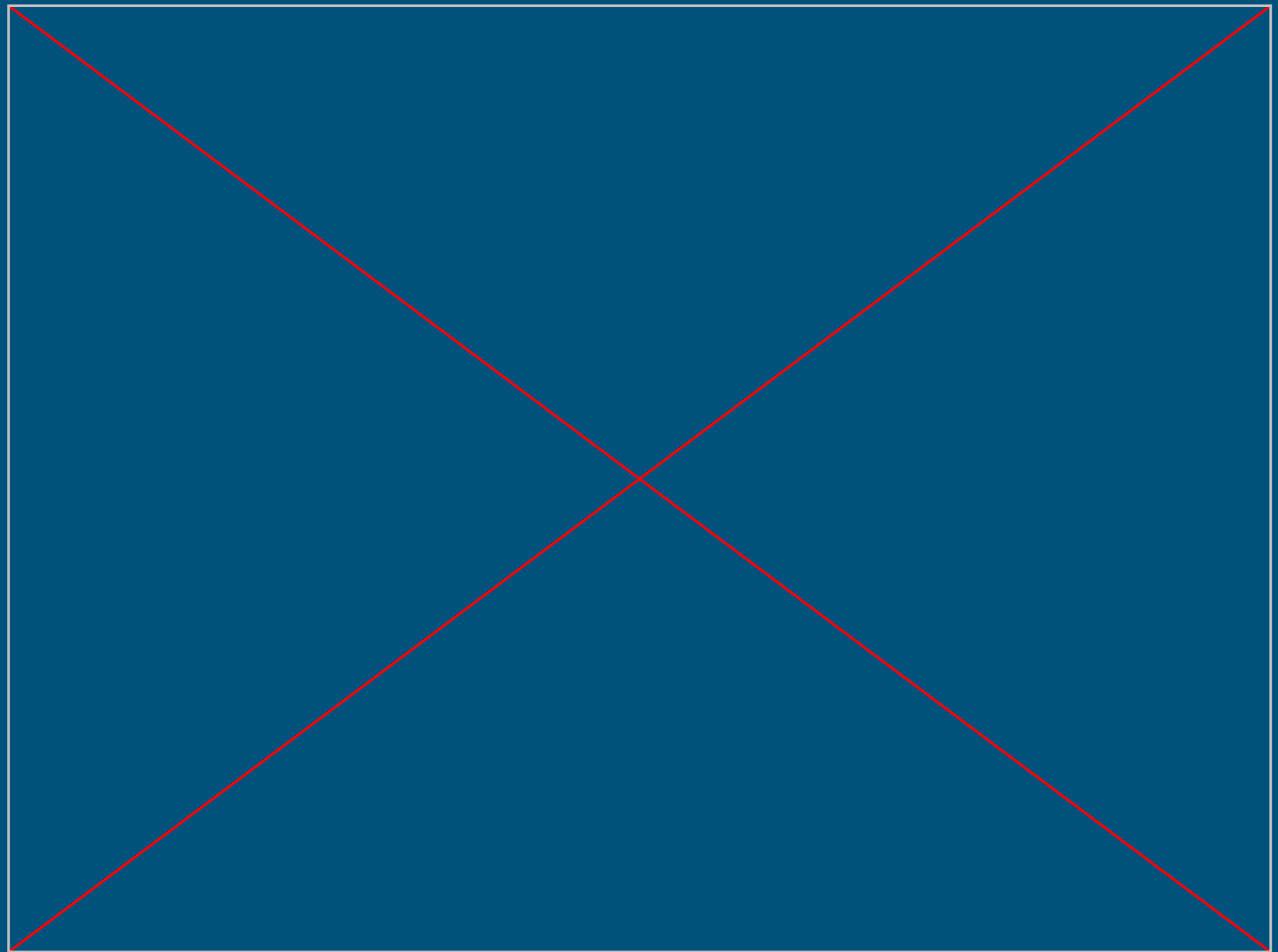
Index	Pattern				Target	Weight Vector				Net Output	Z	LR	Error	Multiplier (Target - Z) x LR	ΔW				Wnew			
	x1	x2	x3	bias		w1	w2	w3	bias						w1	w2	w3	bias	w1	w2	w3	bias
0	1	1	1	1	0	0	0	0	0	0	0	1	FALSE	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0	0	1	TRUE	1	1	0	0	1	1	0	0	1
2	0	0	0	1	0	1	0	0														
3	0	1	0	1	1	1	0	0														
4	1	1	0	1	0	1	1	0														
5	0	0	1	1	1	0	0	0														
6	0	1	1	1	0	0	0	1														
7	1	0	1	1	0	0	-1	0														

```
Index 1:
X = [1, 0, 0, Bias=1], W = [0, 0, 0, 0]

Wnew = Wold - LR * (Z - T) * X
     Input1: w1 = 0 - 1 * (0 - 1) * 1 → 1
     Input2: w2 = 0 - 1 * (0 - 1) * 0 → 0
     Input3: w3 = 0 - 1 * (0 - 1) * 0 → 0
     Bias:   b  = 0 - 1 * (0 - 1) * 1 → 1

Wnew = [1, 0, 0, 1]
```

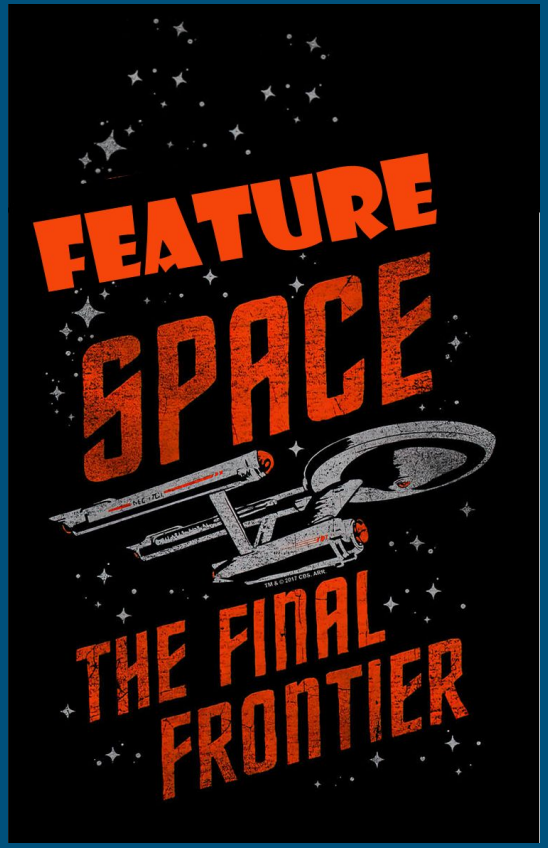
Feature Space

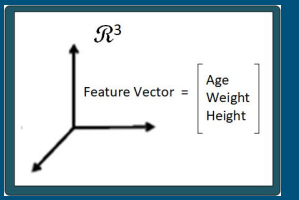
- Feature Vectors
- The Curse of Dimensionality
- Discriminant Surfaces in Feature Space

Feature Space



Feature Space





Feature Space is one of our “building blocks”.

It’s more of a *conceptual* building block.

Feature space is easily the most important building block in our whole set.

Where does feature space come from?

- Feature space is where feature vectors live
- Feature vectors are produced by a feature extractor.
- Almost everything we touch is related to a feature vector.

Feature Space

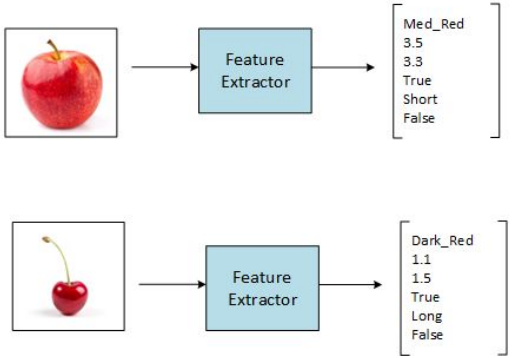
We need a way to encode or numerically represent a piece of fruit
(*so we can classify images of fruit*)

We decide we will encode each piece of fruit with six features.

Feature Vector = [Color, Horiz_Diameter, Vert_Diameter, Presence_of_Stem, Length_of_Stem, Leaf_on_Stem]

Each dimension corresponds to one feature

$\mathcal{R}^6$: A feature space to characterize or encode a piece of fruit with six features

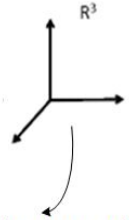


Feature Space

We need a way to encode or numerically represent a person
(*so we can create profiles to deny insurance coverage*)

We decide we will encode each person by their age, weight, and height

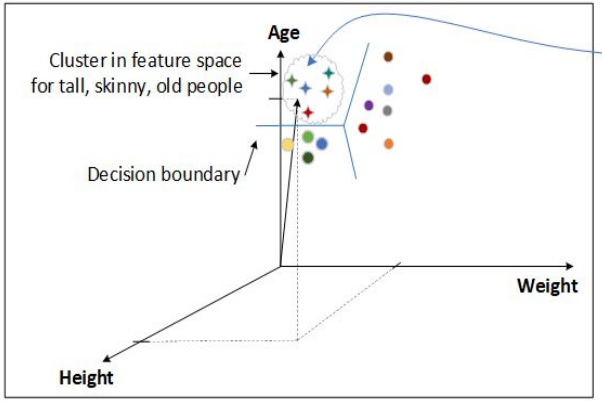
Feature Vector = $\begin{bmatrix} \text{Age} \\ \text{Weight} \\ \text{Height} \end{bmatrix}$

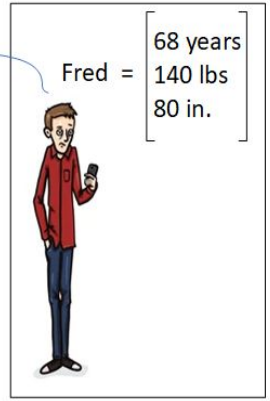


Each of the three dimensions corresponds to one feature

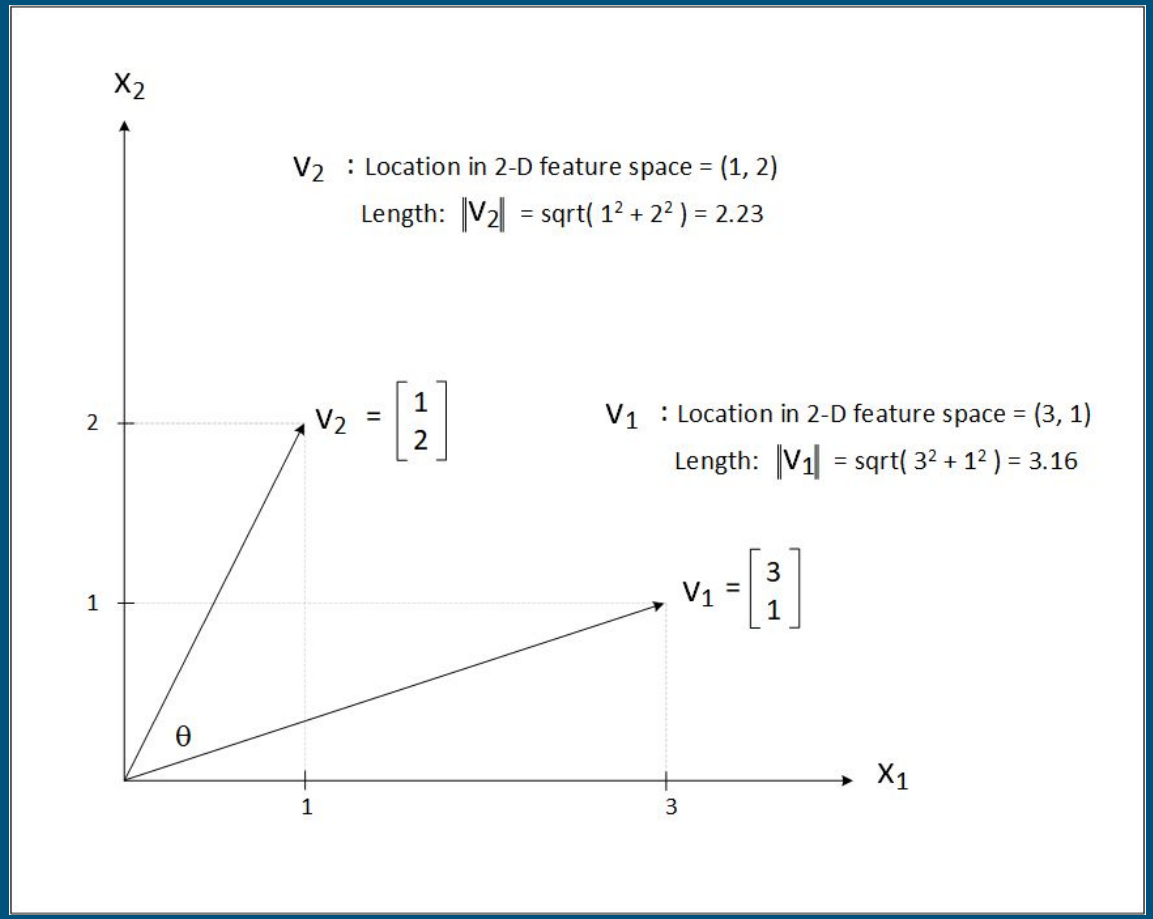
A "point" in this feature space corresponds to one person

$\mathcal{R}^3$: A feature space to characterize or encode a person with three features

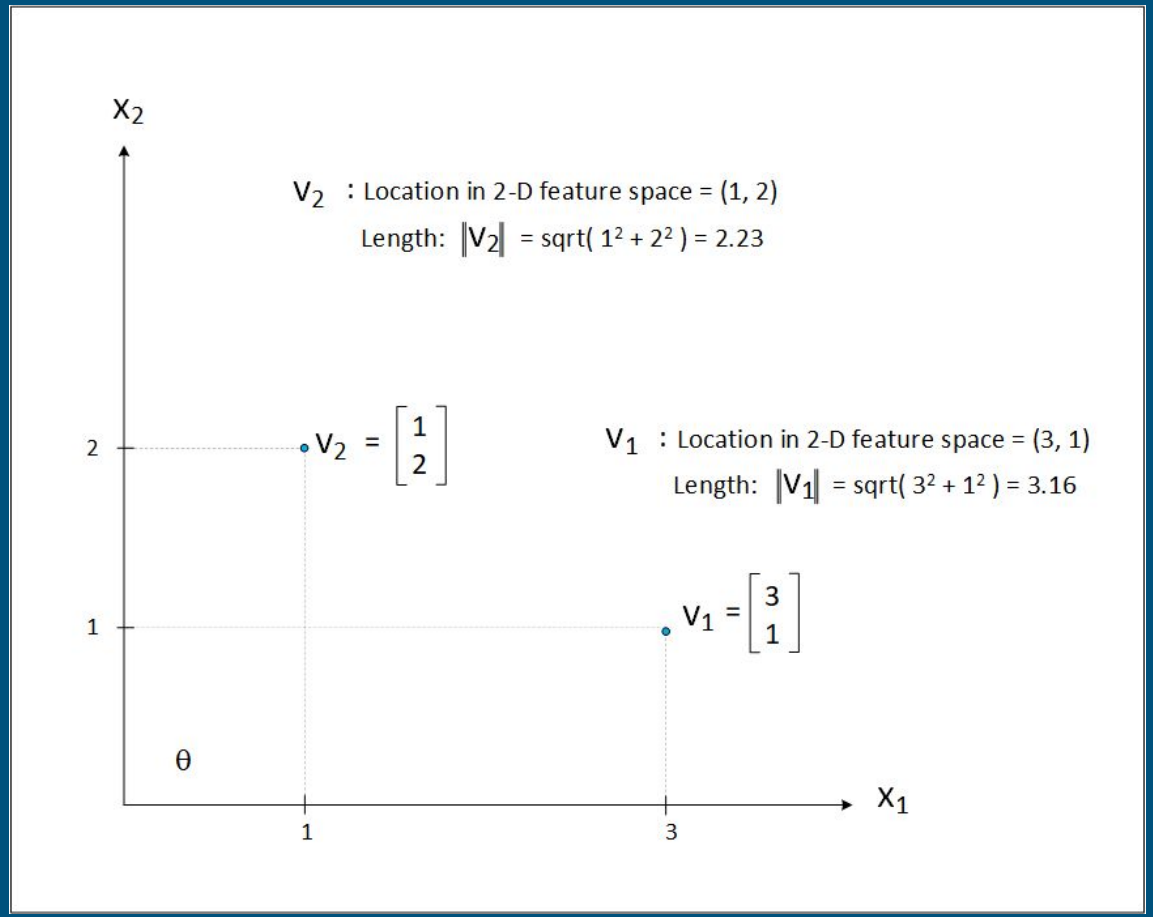




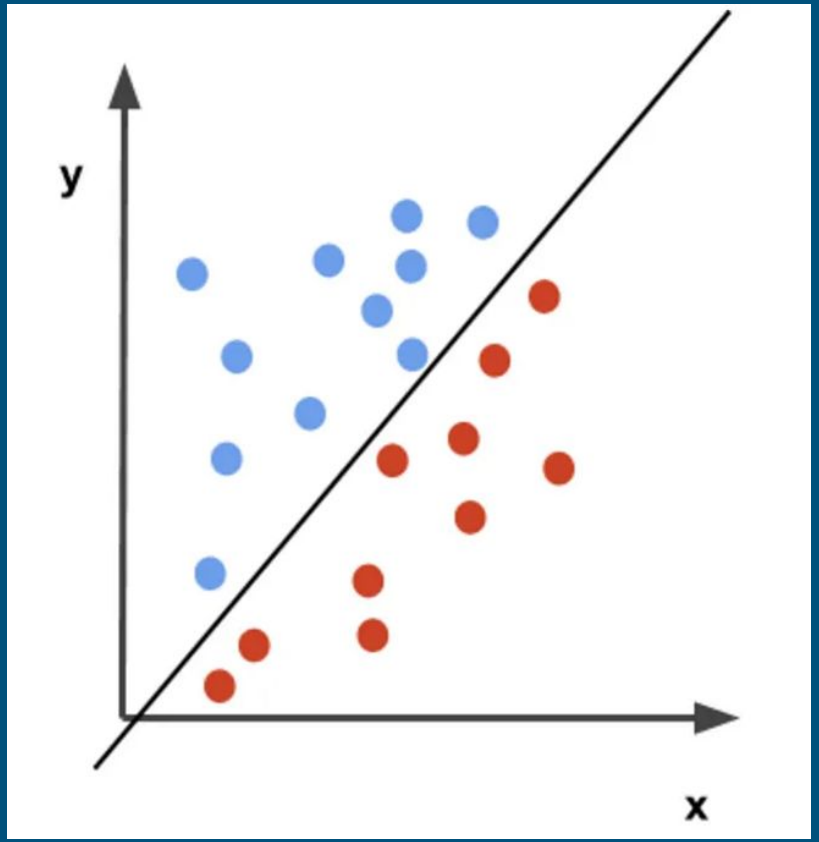
Feature Space



Feature Space



Feature Space



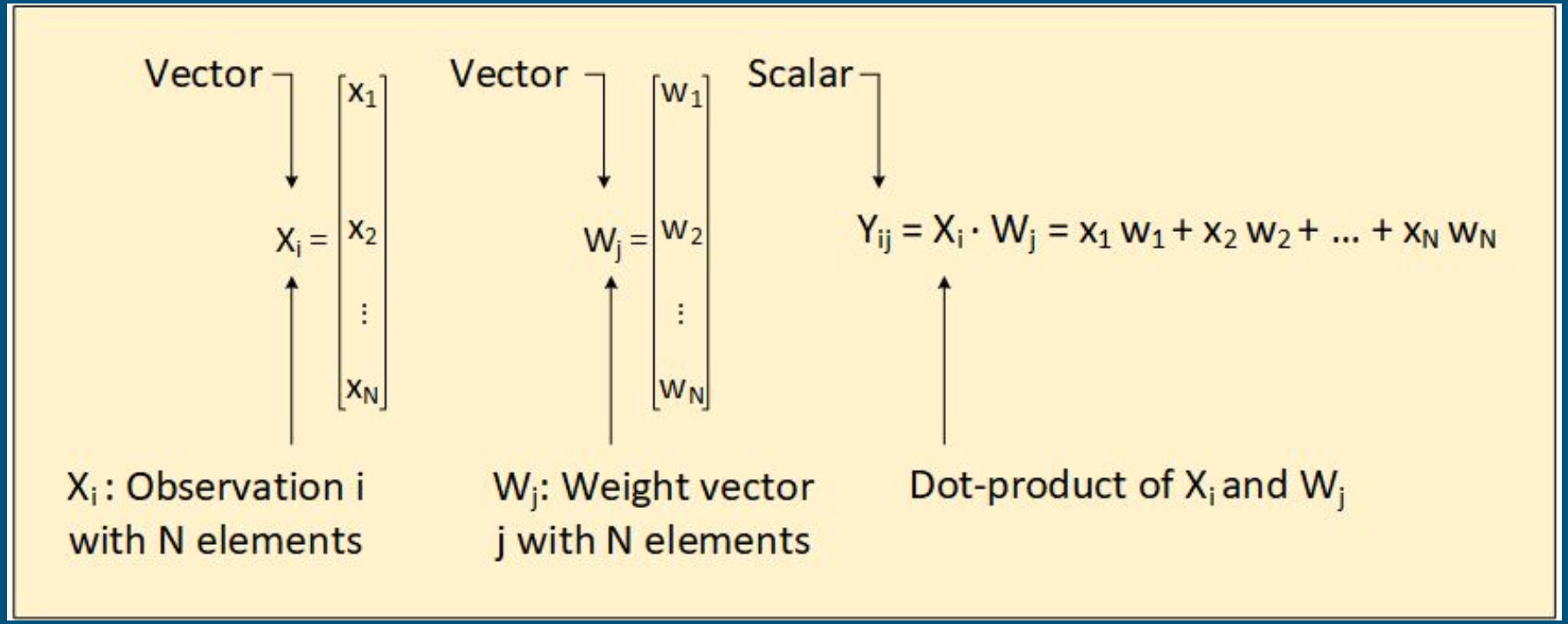
Here's the key:

We want to project vectors into feature space so that …

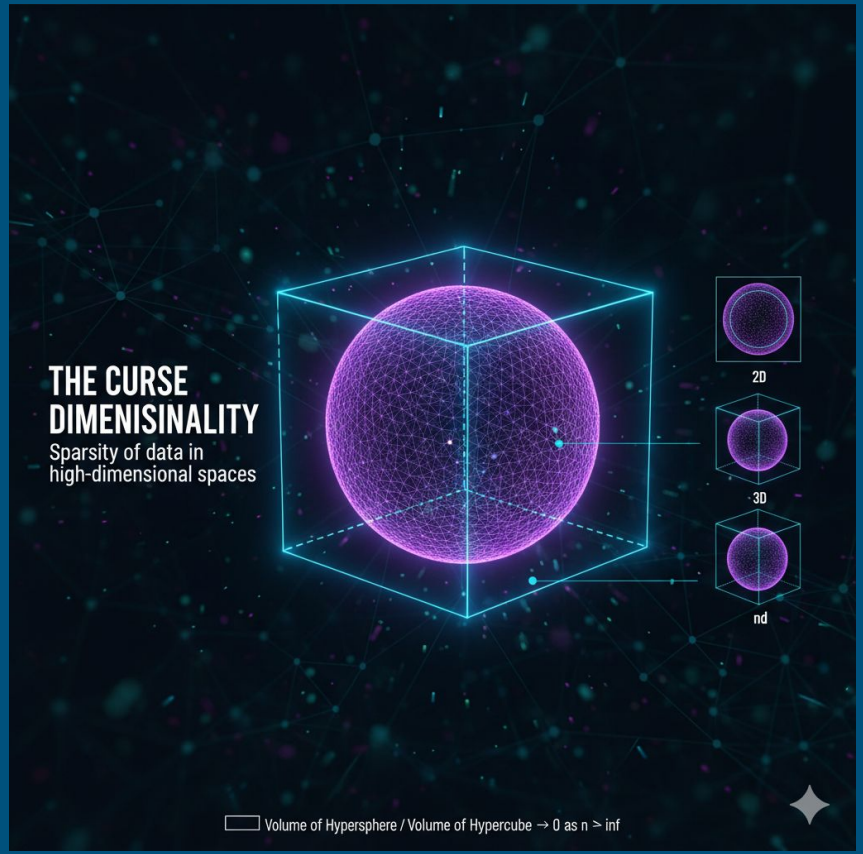
- Items that belong together are spatially close together,
- Items that do not belong together a spatially distant.

This allows us to use a hyperplane to separate things into classes.

Feature Space: The Dot-Product of Two Vectors, a nd b



The Curse of Dimensionality



Keep this image in your head:

- The sphere is called a *hypersphere*.
- The cube is called a *hypercube*.

The Curse of Dimensionality

- In machine learning, feature space is a multi-dimensional vector space.
- Each of the D features serves as one of the dimensions in this space.
- Thus, a feature vector with ten elements lives in a feature space with ten dimensions.
- Increasing the number of dimensions comes with a cost.
- Having a very high-dimensional feature space will cause a serious problem in machine learning that Richard Bellman called the "*curse of dimensionality*".
- Why it is called a curse is explained in the following.

The Curse of Dimensionality

Hyperspheres:

- Starting with a vector space of dimension, D, if we take all of the D-dimensional vectors that start at the origin and have a length of one, their endpoints form a surface or region that we generically refer to as a sphere.
- For D = 2, the endpoints form a circle; for D = 3, the endpoints form a sphere.
- For higher dimensions, the endpoints of all of the unit vectors form a hypersphere.

Hypercubes:

- Similarly, if we create an enclosing (or bounding) cube, centered on the origin, the circle/sphere will touch the edges of this cube.
- Brian Hayes wrote that the "surface of the (hypersphere) kisses the center of each face of the cube".
- Note that the circle/sphere doesn't extend out to touch the corners (vertices) of the cube. The regions of the cube that are left vacant by the sphere are the corners.

The Curse of Dimensionality

The problem is this:

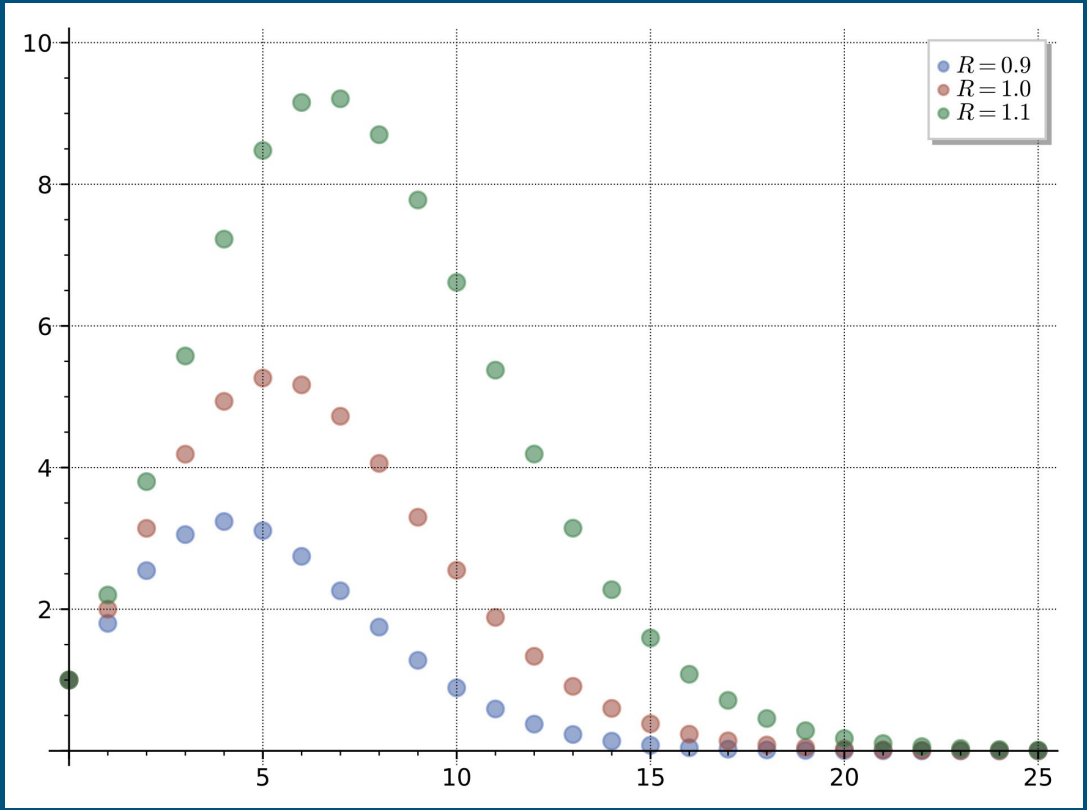
- The volume of the hypersphere and the hypercube both grow as you increase D.
- But the hypercube expands faster. Much faster.
- Each time D increases by 1, the number of corners of the hypercube doubles, so we can expect ever more hypercube volume to migrate into the nooks and crannies near the cube's vertices.
- In high dimensions (D = 100), the fraction of the hypercube that the hypersphere fills is smaller than the volume of an atom in relation to the volume of the earth.
- The volume of the hypersphere, relative to the hypercube, is so small, it has effectively vanished.
- If you were to select a trillion points at random from the interior of the cube, you'd have almost no chance of landing on even one point that is also inside the ball.

The Curse of Dimensionality

Here is what is so strange:

- The hypersphere is still the largest one that could possibly be stuffed into the hypercube.
- We are not talking about a pea rattling around loose inside a refrigerator carton.
- The ball’s diameter is still equal to the side length of the cube.
- The surface of the ball touches every face of the cube.
- But in terms of volume measure, the ball is nearly crushed out of existence.

The Curse of Dimensionality



Volumes of balls in dimensions 0 through 25; unit ball in red.

Beyond the fifth dimension, the volume of a unit n-ball decreases as n increases

The Curse of Dimensionality

The key insight is that the hypersphere/hypercube analogy is a proxy for understanding how data sparsity emerges in high dimensions.

Here's how it connects to training data coverage:

- Data Doesn't Fill the Hypersphere Uniformly:
  - Even if your data lies within a hypersphere, it doesn't mean the data is uniformly distributed there.
  - In high dimensions, the effective volume where data actually resides (the "support" of the distribution) is often a thin shell or a lower-dimensional manifold within the hypersphere.
  - This is analogous to the hypersphere becoming negligible compared to the hypercube.

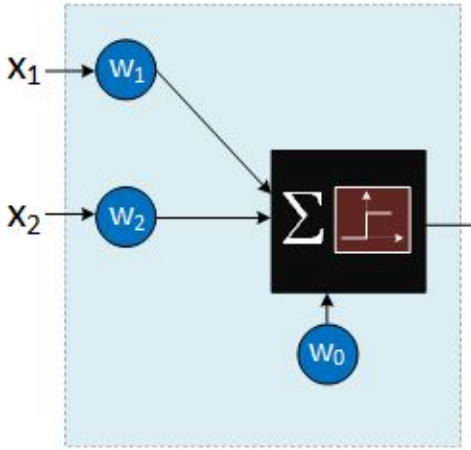
The Curse of Dimensionality

- Distance Concentration:
  - In high dimensions, distances between points become similar (all pairs of points are roughly equidistant).
  - This means that local neighborhoods (which are critical for methods like k-NN or kernel-based models) become ill-defined because "nearby" points are still very far apart in absolute terms.
  - The hypersphere/hypercube analogy hints at this: most points are concentrated in the corners (far apart), while the hypersphere (representing "close" points) is vanishingly small.

- Coverage Requires Exponentially More Data:
  - Suppose your data lies in a D-dimensional hypersphere. To "cover" this space with a grid of points spaced every epsilon along each dimension, you need O((1/epsilon)D) points.
  - This exponential growth is the core of the curse, and the hypersphere/hypercube analogy is a way to show how volume (and thus required data) explodes with D

Discriminant Surfaces in Feature Space

Feature Space



$D(X) = W^T X + w_0$

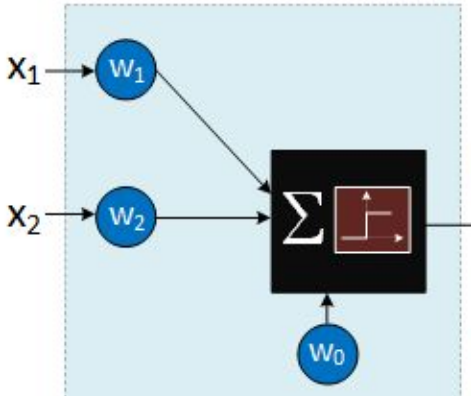
D(X) can be used as a *discriminant function* that decides if X is assigned to class C1 or C2.

Class membership for X is determined by W, because W defines a decision surface.

W is perpendicular to $W^T X + w_0$

Feature Space

This should look familiar!
It's a single node of a perceptron.
Actually, it's a dot-product of X and W.



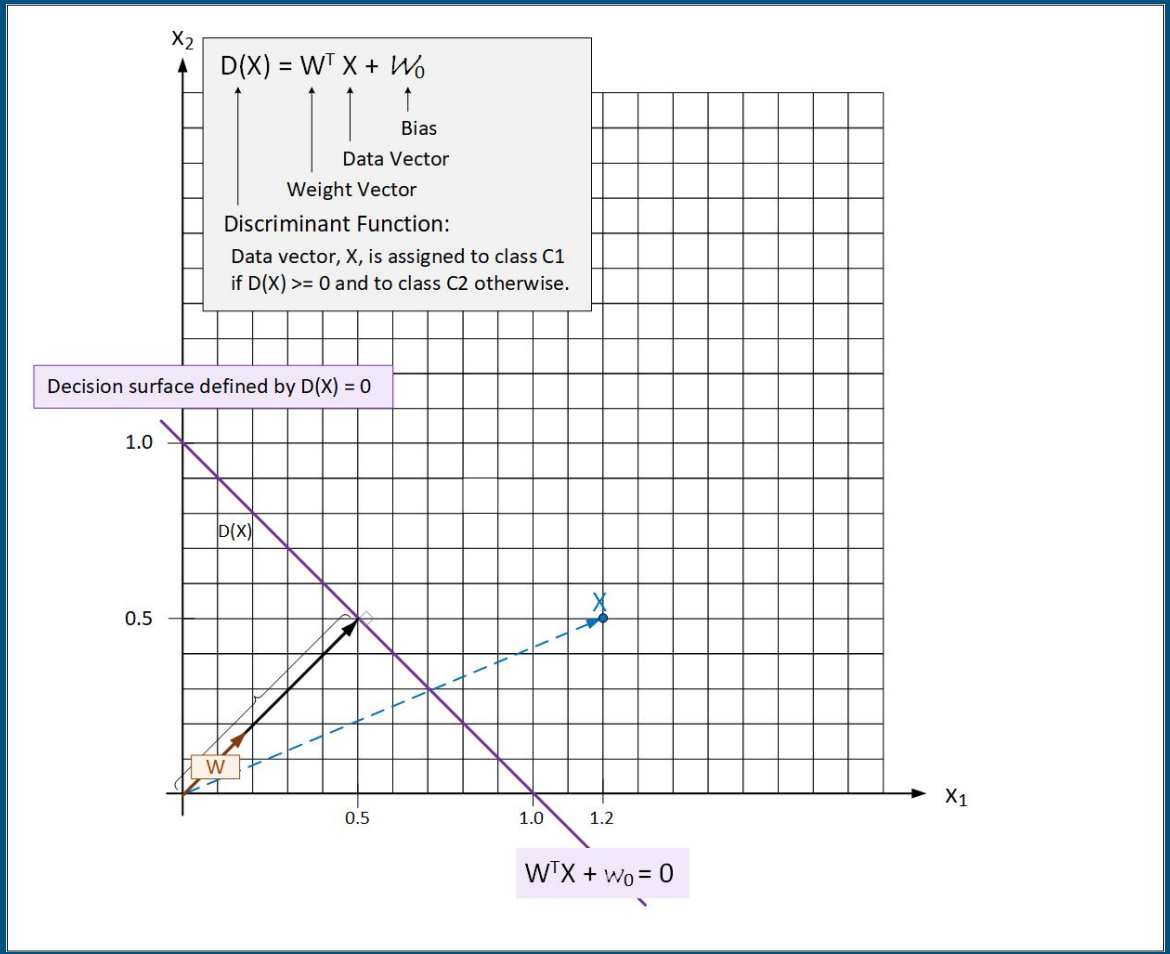
$D(X) = W^T X + w_0$

D(X) can be used as a *discriminant function* that decides if X is assigned to class C1 or C2.

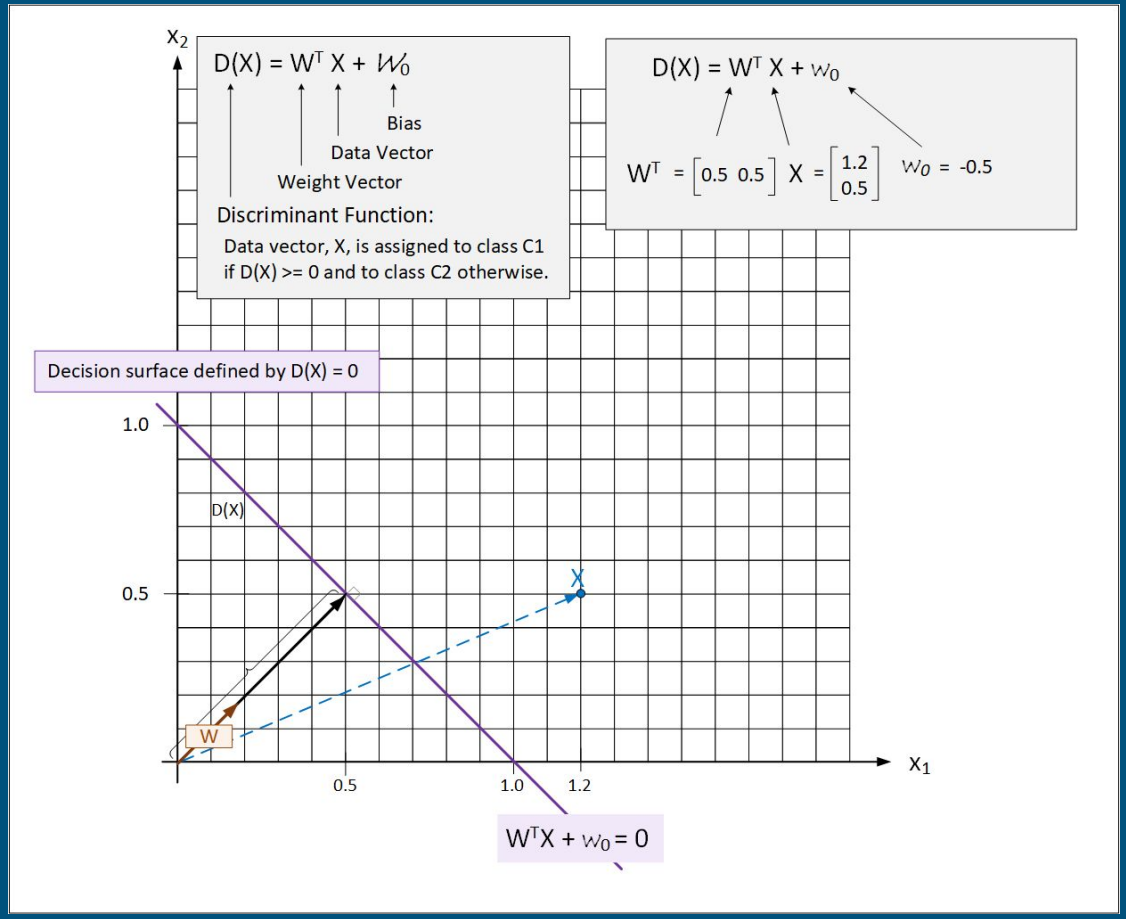
Class membership for X is determined by W, because W defines a decision surface.

W is perpendicular to $W^T X + w_0$

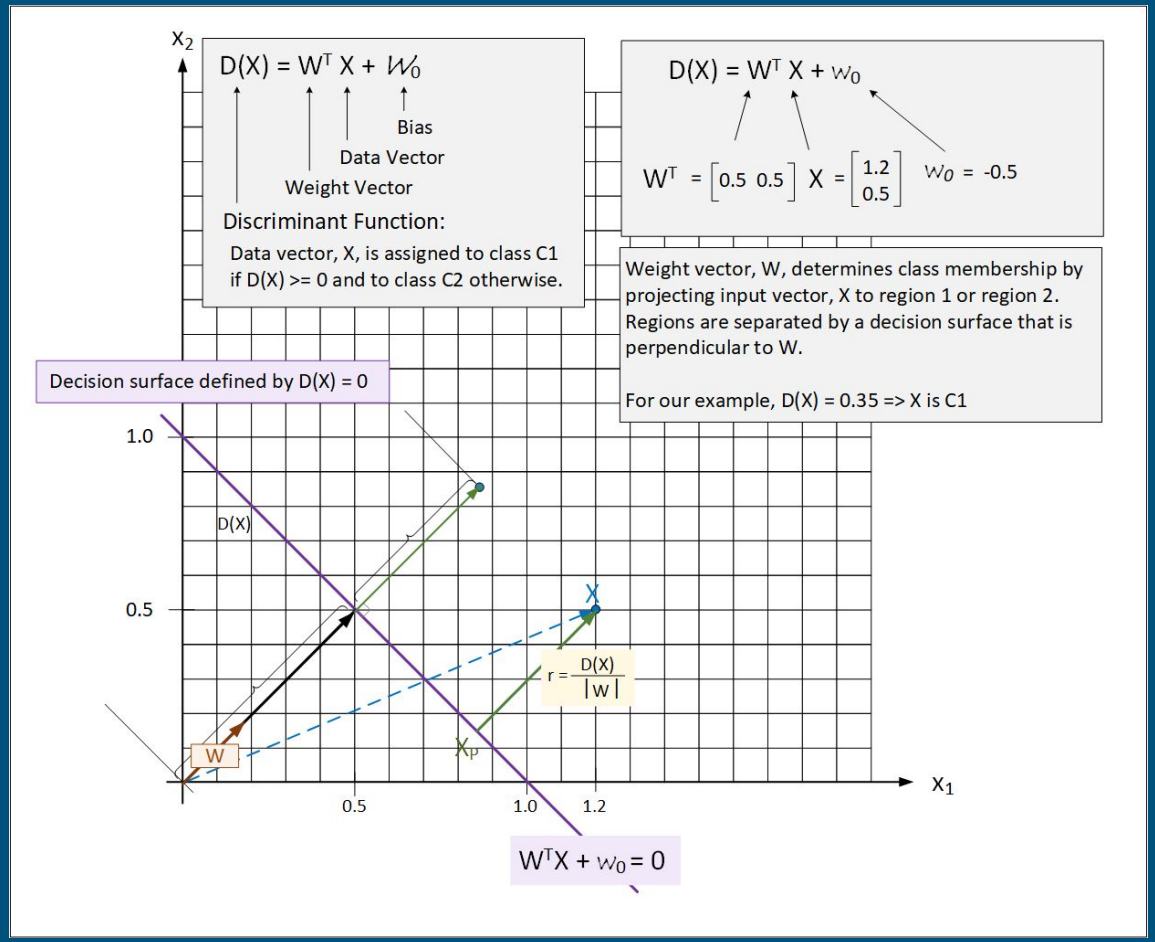
Linear Discriminants



Linear Discriminants



Linear Discriminants



Linear Discriminants

Consider two points, X_A & X_B that are on the decision surface, $D(X)$

Since X_A, X_B are on the surface, we know ...

$$D(X_A) = D(X_B) = 0$$

By def'n, $D(X_A) := W^T X_A = 0$

and $D(X_B) := W^T X_B = 0$

They're both 0, so we can equate them:

$$W^T X_A = W^T X_B$$

$$W^T X_A - W^T X_B = 0$$

$$W^T (X_A - X_B) = 0$$

This is a dot-product of two vectors

Since the dot-product is 0, we know they are orthogonal.

Linear Discriminants

Recall ...

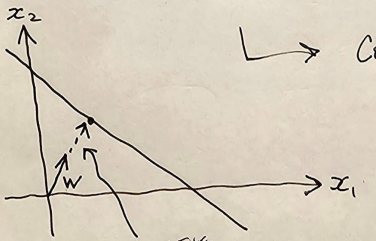
1. $W^T \cdot X = 0$
   $\Rightarrow W \perp X$

2. $W \parallel X \Rightarrow \cos\theta = 1$

3. W only defines the direction of decision surface.
   ω_0 is a constant that defines the displacement or distance or offset from origin.

4. The def'n for cosine similarity:

$$\cos\theta := \frac{X \cdot W}{|X||W|}$$



X_N is the normal vector. We want its length

$$\rightarrow \cos\theta := \frac{X_N \cdot W}{|X_N||W|} = 1$$

(*)

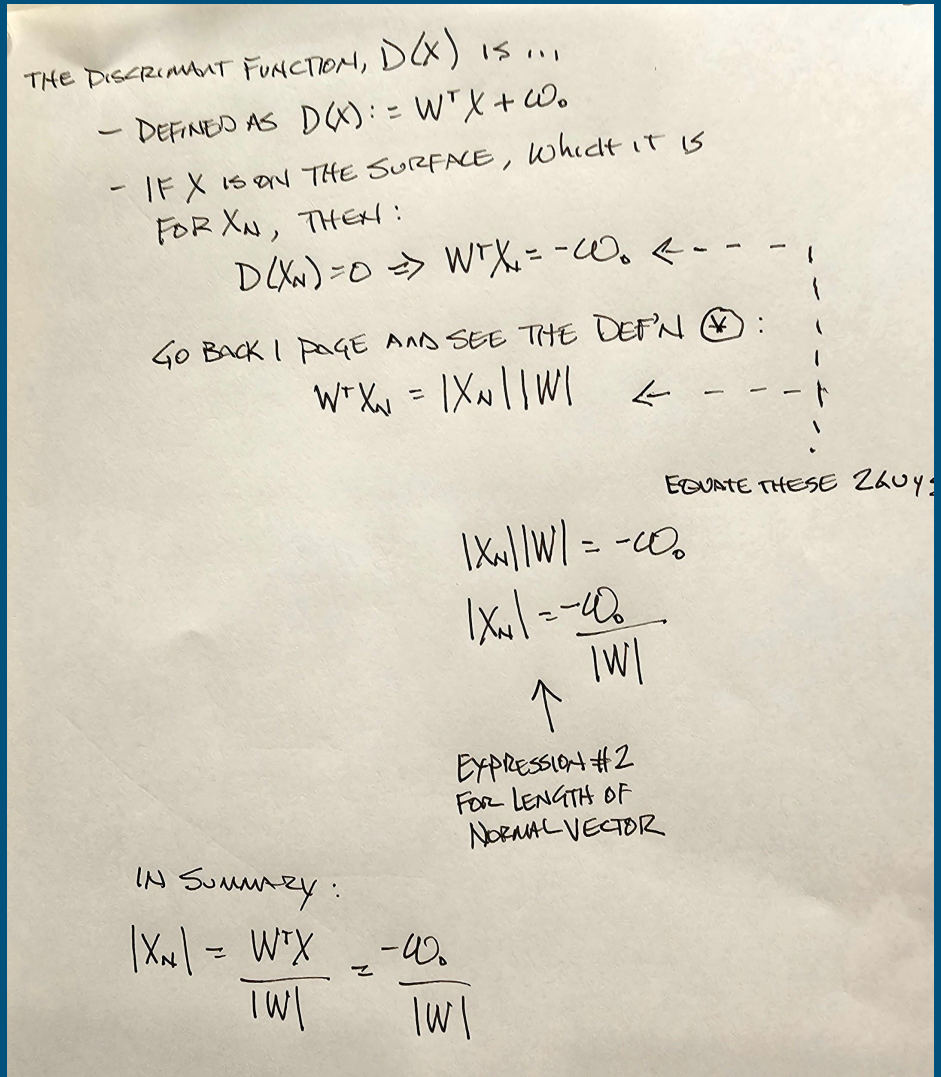
$$X_N \cdot W = |X_N||W|$$

(this is how we get it!)

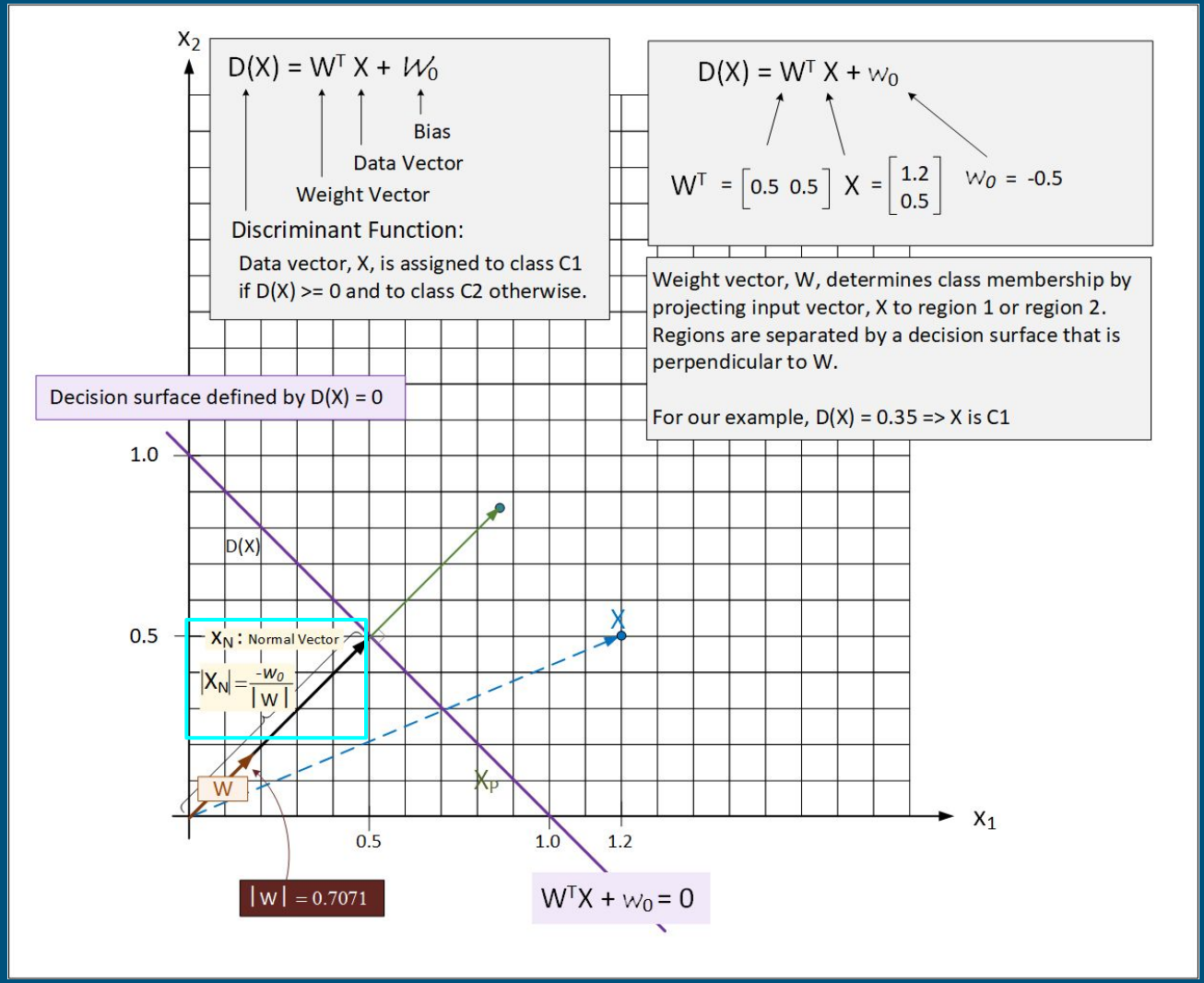
$$|X_N| = \frac{W^T X}{|W|}$$

Expression #1 for length of normal vector

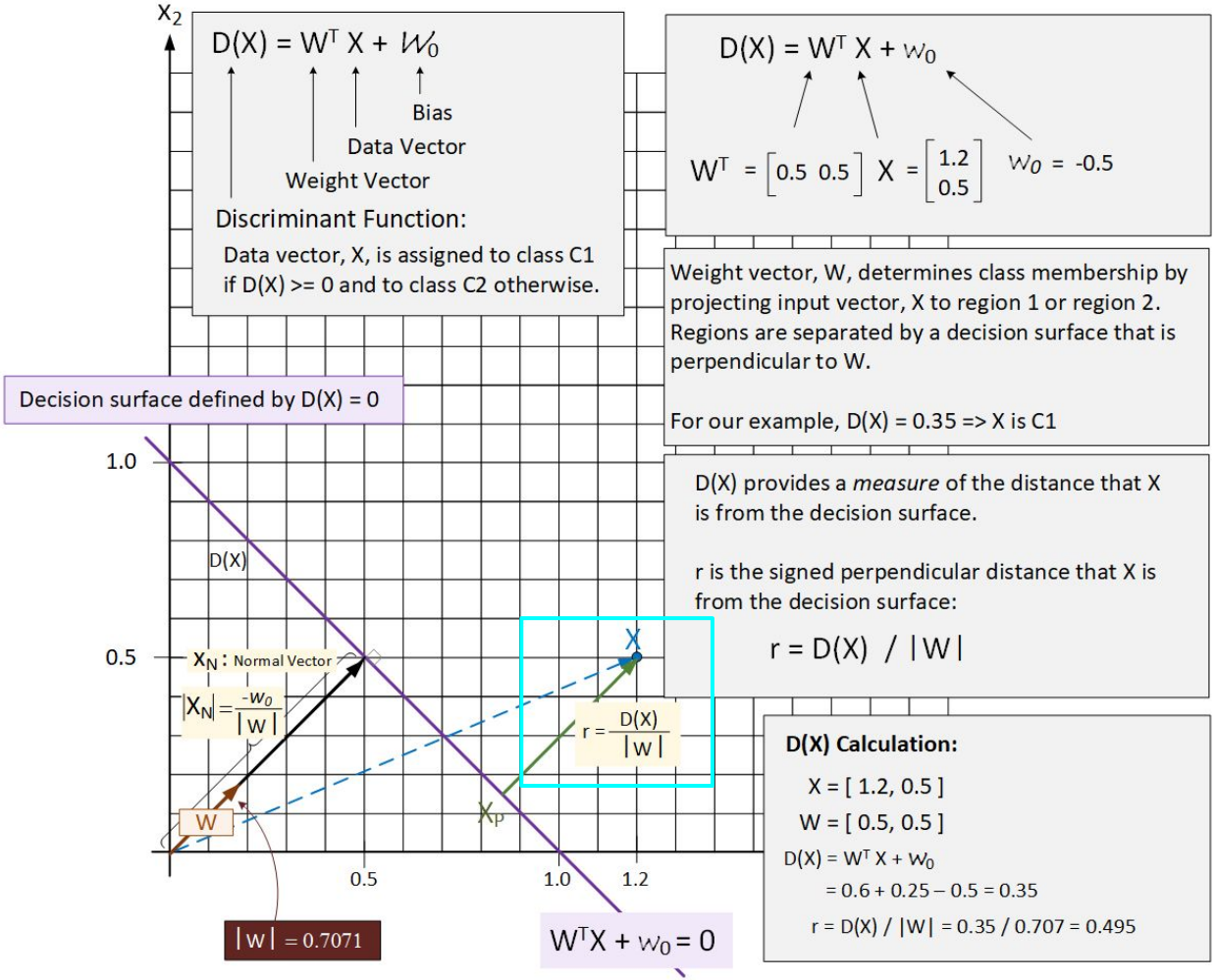
Linear Discriminants



Linear Discriminants



Linear Discriminants



Linear Discriminants

What do we learn from the geometry of the linear discriminant function?

- What is the orientation of the decision surface, W?
- What is the displacement of the decision surface, w0?
- Where does the discriminant project X?
- Where is X relative to the decision surface?