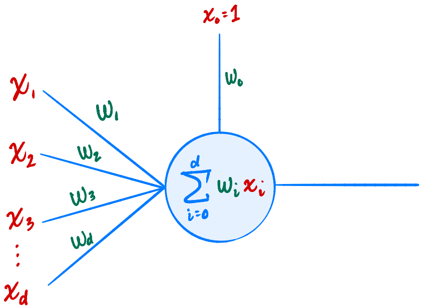


CSE 151A

Intro to Machine Learning

Lecture 11 – Part 01

Stochastic Gradient Descent

Midterm 01 Scores Posted

- Scores were high!
- Remember: redemption is available on final.
- Pass v. no pass.
  - Still a lot of plus points left to earn.

Recall: Gradient Descent

- Suppose you are minimizing a function $R(\vec{w})$.
- Gradient descent: on step i, update

$$\vec{w}^{(i)} = \vec{w}^{(i-1)} - \alpha \cdot \nabla R(\vec{w}^{(i-1)})$$

A Small Problem with Big Data

- In ML, our functions involve data.
  - MSE:

$$R(\vec{w}) = \sum_{i=1}^{n} (\operatorname{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2$$

  - Log Likelihood:

$$R(\vec{w}) = -\sum_{i=1}^{n} \log\left[1 + e^{-y_i \vec{w} \cdot \operatorname{Aug}(\vec{x}^{(i)})}\right]$$

- When n is large, gradient is **expensive** to compute.

Decomposability

- In ML, our functions are often **decomposable**:

$$R(\vec{w}) = \sum_{i=1}^{n} \ell(\vec{w}; \vec{x}^{(i)}, y_i)$$

- And so $\nabla R(\vec{w}) = \sum_{i=1}^{n} \nabla \ell(\vec{w}; \vec{x}^{(i)}, y_i)$

The Idea

- The **full gradient** is

$$\nabla R(\vec{w}) = \sum_{i=1}^{n} \nabla \ell(\vec{w}; \vec{x}^{(i)}, y_i)$$

- Instead, approximate with a **mini-batch**:

$$\nabla R(\vec{w}) \approx \sum_{i \in B} \nabla \ell(\vec{w}; \vec{x}^{(i)}, y_i)$$

Mini-Batch Stochastic Gradient Descent

To minimize a **decomposable** function $R(\vec{w}) = \sum_{i=1}^{n} \ell(\vec{w}; \vec{x}^{(i)}, y_i)$:

- Pick a starting guess, $\vec{w}^{(0)}$.
- On step k, randomly select a set B of $n' \leq n$ points, perform update:

$$\vec{w}^{(k)} = \vec{w}^{(k-1)} - \alpha \sum_{i \in B} \nabla \ell(\vec{w}^{(k-1)}; \vec{x}^{(i)}, y_i)$$

- Repeat until convergence

Stochastic Gradient Descent

To minimize a **decomposable** function $R(\vec{w}) = \sum_{i=1}^{n} \ell(\vec{w}; \vec{x}^{(i)}, y_i)$:

- Pick a starting guess, $\vec{w}^{(0)}$.
- Repeat until convergence:
  - Randomly shuffle the points.
  - Loop through all n points, one at a time, performing:

$$\vec{w}^{(k)} = \vec{w}^{(k-1)} - \alpha \nabla \ell(\vec{w}^{(k-1)}; \vec{x}^{(i)}, y_i)$$

Example: Logistic Regression

- **Prediction Rule**:

$$H_{\vec{w}}(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$$

- **Goal**: find $\vec{w}$ maximizing log likelihood

$$\log \mathcal{L}(\vec{w}) = -\sum_{i=1}^{n} \log\left[1 + e^{-y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)})}\right]$$

Observation

- The log likelihood is **decomposable**:

$$\log \mathcal{L}(\vec{w}) = -\sum_{i=1}^{n} \log\left[1 + e^{-y_i \vec{w}\cdot \text{Aug}(\vec{x}^{(i)})}\right]$$

$$= \sum_{i=1}^{n} \ell(\vec{w}; \vec{x}^{(i)}, y_i)$$

where $\ell(\vec{w}; \vec{x}^{(i)}, y_i) = -\log\left[1 + e^{-y_i \vec{w}\cdot \text{Aug}(\vec{x}^{(i)})}\right]$

Why SGD?

- SGD is computationally cheaper than GD.
- One step of GD with **full gradient**: $\Theta(nd)$.
- One step of SGD: $\Theta(d)$.

The Good and the Bad with SGD

- **Good**: fast(er) per step, guaranteed to converge if R convex, generalization.
- **Bad**: single point might result in noisy gradient.

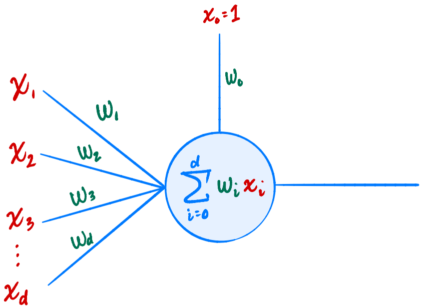
Example

More: Learning Rates

- How do we pick α?
- One strategy: line search.
- Decay: α_t instead of α.

More: Early Stopping

- To avoid overfitting, don’t run GD/SGD completely.
- Stop early, when validation error starts to go up.

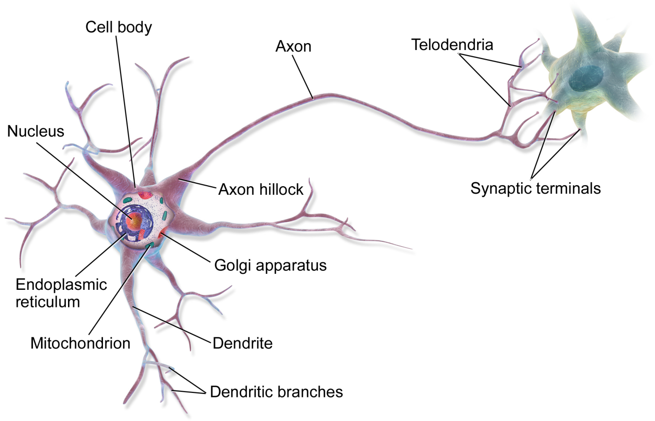


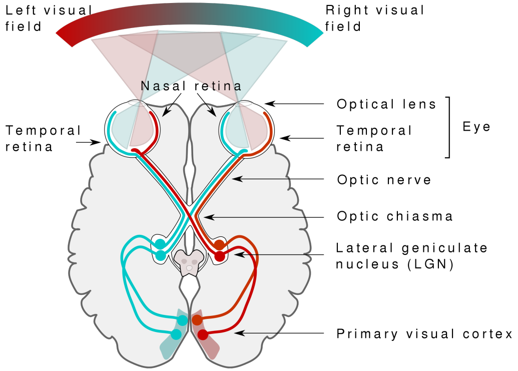
CSE 151A

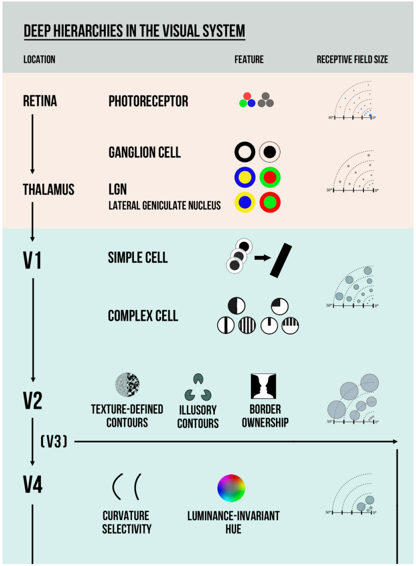
Intro to Machine Learning

Lecture 11 – Part 02

Perceptrons


Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic reticulum
Mitochondrion
Dendrite
Dendritic branches


Left visual field
Right visual field
Nasal retina
Temporal retina
Optical lens
Temporal retina
Eye
Optic nerve
Optic chiasma
Lateral geniculate nucleus (LGN)
Primary visual cortex

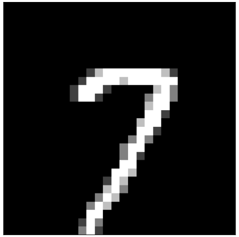


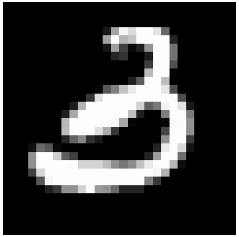
By Karlis.kanders - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=35206366

SALE

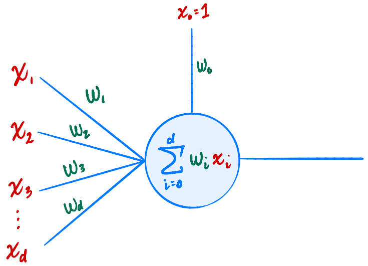
Today

- Design an artificial “neuron”, a **perceptron**.
- Train it to recognize images (handwritten digits).



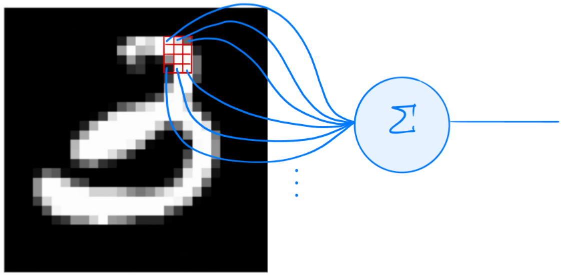


Modeling a Neuron



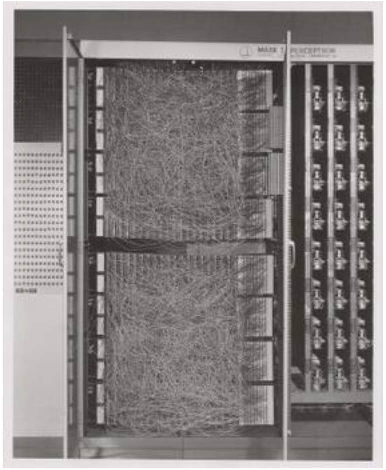
- Input: $\vec{x} = (x_1, \dots, x_d)^T$
- **Synapse weights**: $\vec{w} = (w_0, w_1, \dots, w_d)^T$
- Output: $\sum_{i=0}^{d} w_i x_i = \text{Aug}(\vec{x}) \cdot \vec{w}$
- This model is called a **perceptron**.

Example: Image Recognition

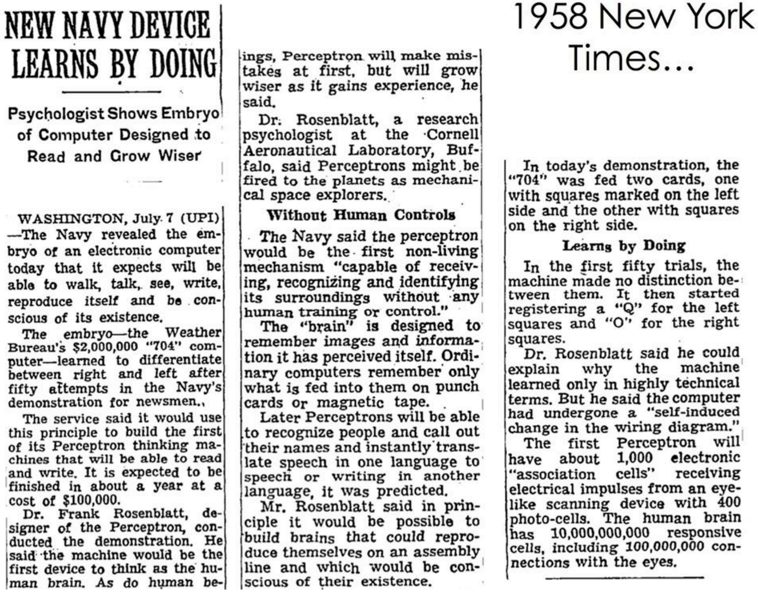


- Binary classifier:
  - If output > 0, predicted digit is a three
  - If output < 0, predicted digit is not a three

Rosenblatt’s Perceptron (1958)



1958 New York Times…



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.,

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

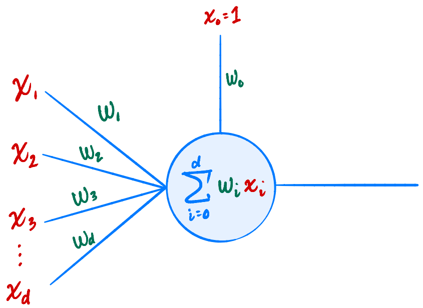
Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Training a Perceptron



- A perceptron is trained by adjusting its weights, $w_0, \ldots, w_d$.

Example: Predicting Survival

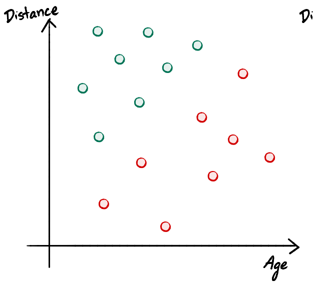
- We have a data set of hurricane survivors:

Age	Distance From Coast	Survived
21	30	1 (Yes)
84	2	-1 (No)
47	10	-1 (No)
30	15	1 (Yes)
⋮	⋮	⋮

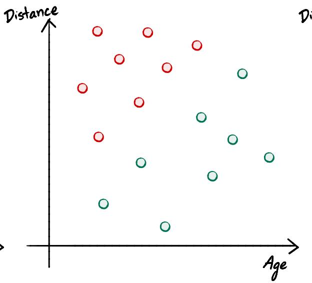
- Goal: train perceptron to predict if someone will survive.
  - If output is positive, prediction is “yes”
  - If output is negative, prediction is “no”

Example

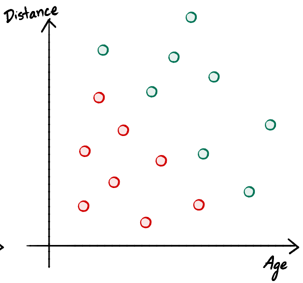
Suppose we plot the data, with green points for people who survived, and red points for those who did not. Which do we see?



(a)

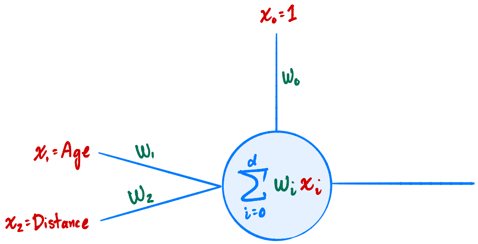


(b)



(c)

Example: Predicting Survival

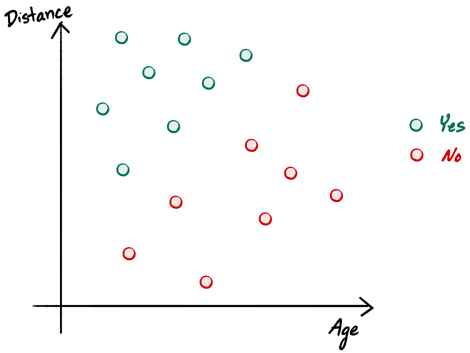


- Prediction rule:

$$H(\text{Age}, \text{Distance}) = w_0 + w_1 \times \text{Age} + w_2 \times \text{Distance}$$

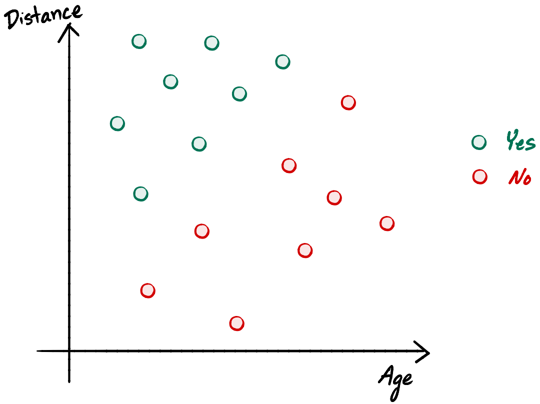
- If $H(x_1, x_2) > 0$, prediction is “yes”
- If $H(x_1, x_2) < 0$, prediction is “no”

The Decision Boundary



- The **decision boundary** is where $H = 0$.
- On one side of boundary, $H > 0$; prediction is **yes**.
- On other side, $H < 0$; prediction is **no**.
- **Important**: $|H(\vec{x})| \propto$ distance from boundary.

A Good Decision Boundary



Learning a Linear Decision Boundary

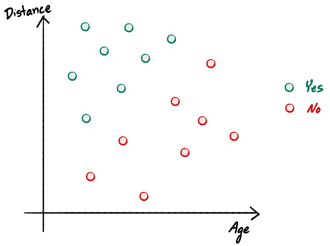
- Given feature vectors $\vec{x}^{(1)}, \ldots \vec{x}^{(n)}$ and labels $y_1, \ldots, y_n$.
- Goal: find $\vec{w}$ such that each point is **classified** correctly; i.e., it is on the correct side of boundary.
- We'll use empirical risk minimization.
- What is an appropriate objective function?

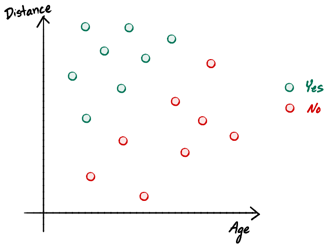
Risk #1: The 0-1 Loss

- The **0-1 risk** is then:

$$R_{01}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}1, & \operatorname{sign}(H(\vec{x})) \neq \operatorname{sign}(y)\\ 0, & \operatorname{sign}(H(\vec{x})) = \operatorname{sign}(y)\end{cases}$$

- Proportion of points which are misclassified.
- Example:





Goal: Minimize the 0-1 Risk

- Find $\vec{w}$ which results in fewest # of misclassified points.

- That is, minimize

$$R_{01}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1, & \operatorname{sign}(H(\vec{x}^{(i)})) \neq \operatorname{sign}(y_i) \\ 0, & \operatorname{sign}(H(\vec{x}^{(i)})) = \operatorname{sign}(y_i) \end{cases}$$

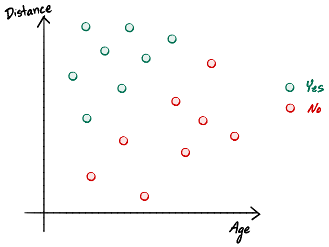
- Gradient descent?

Analyzing the 0-1 Risk

Is

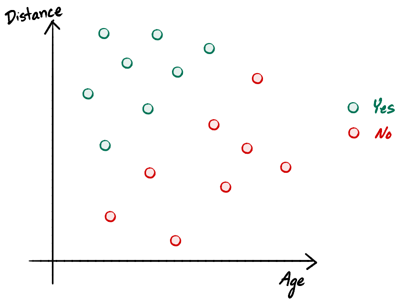
$$R_{01}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1, & \operatorname{sign}(H(\vec{x}^{(i)})) \neq \operatorname{sign}(y_i) \\ 0, & \operatorname{sign}(H(\vec{x}^{(i)})) = \operatorname{sign}(y_i) \end{cases}$$

continuous? Differentiable? Convex?



The Problem

- R_{01} is **flat**.
- The gradient gives no information.



Loss #2: The Perceptron Loss

- We need a loss function that isn't flat.

- If prediction is wrong, size of $|H(\vec{x})|$ measures how wrong.

- The **perceptron loss**:

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

Loss #2: The Perceptron Loss

- The **perceptron risk**:

$$R_{\text{tron}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}0, & \text{sign}(H(\vec{x}^{(i)})) = \text{sign}(y_i)\\ |H(\vec{x}^{(i)})|, & \text{sign}(H(\vec{x}^{(i)})) \neq \text{sign}(y_i)\end{cases}$$

$$=$$

where M is the set of misclassified points.

- **Continuous**, **not differentiable**, **not flat**.

Stochastic Gradient Descent for Perceptron Learning

- Compute "gradient" of

$$\ell(\vec{w}; \vec{x}^{(i)}, y_i) = \begin{cases} 0, & \text{sign}(H(\vec{x}^{(i)})) = \text{sign}(y_i) \\ |H(\vec{x}^{(i)})|, & \text{sign}(H(\vec{x}^{(i)})) \neq \text{sign}(y_i) \end{cases}$$

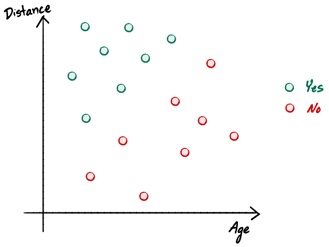
The Perceptron Algorithm

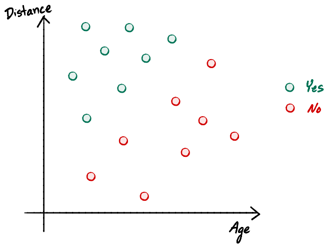
- Pick an initial $\vec{w}^{(0)}$.
- Repeat until convergence:
  - Construct set M of misclassified points using $\vec{w}^{(t-1)}$
  - If M is empty, break (no points misclassified).
  - Otherwise, loop over misclassified points $i \in M$, performing update:

$$\vec{w}^{(t)} = \vec{w}^{(t-1)} - \alpha \begin{cases} \operatorname{Aug}(\vec{x}^{(i)}), & \vec{w}^{(t-1)} \cdot \operatorname{Aug}(\vec{x}^{(i)}) \geq 0 \\ -\operatorname{Aug}(\vec{x}^{(i)}), & \vec{w}^{(t-1)} \cdot \operatorname{Aug}(\vec{x}^{(i)}) < 0 \end{cases}$$

The Perceptron Algorithm

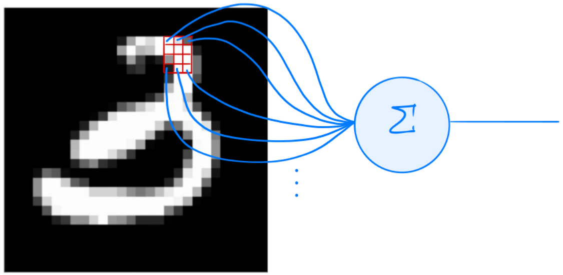
- Data is **linearly separable** if classes can be separated by a line (plane):





- If linearly separable, perceptron algorithm will terminate; classify all points correctly.

Example: Image Recognition



- Binary classifier:
    - If output > 0, predicted digit is a three
    - If output < 0, predicted digit is not a three

(DEMO)