The Perceptron

HackCU Episode IV

Learn New Skills
Build Something Cool
Meet Awesome People
Discover Job Opportunities

February 24th and 25th 2018

HackCU.org

Administrivia

- **Reminder**: Homework 1 is due by 5pm Friday on Moodle
- **Friday** is a notebook day, so bring your laptops
- **Need a Note Taker for this course. CU Bookstore gift certificate at end of semester.**

Previously on CSCI 4622

In classification problems our outputs are discrete **classes** (eg. SPAM vs HAM)

Goal: Given features $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ predict which class $Y = k$, $\mathbf{x}$ belongs to

Instance-based methods like KNN are called **non-parametric models**

When we learn parameters we call the model a **parametric model**

Today we look at our first parametric model for classification

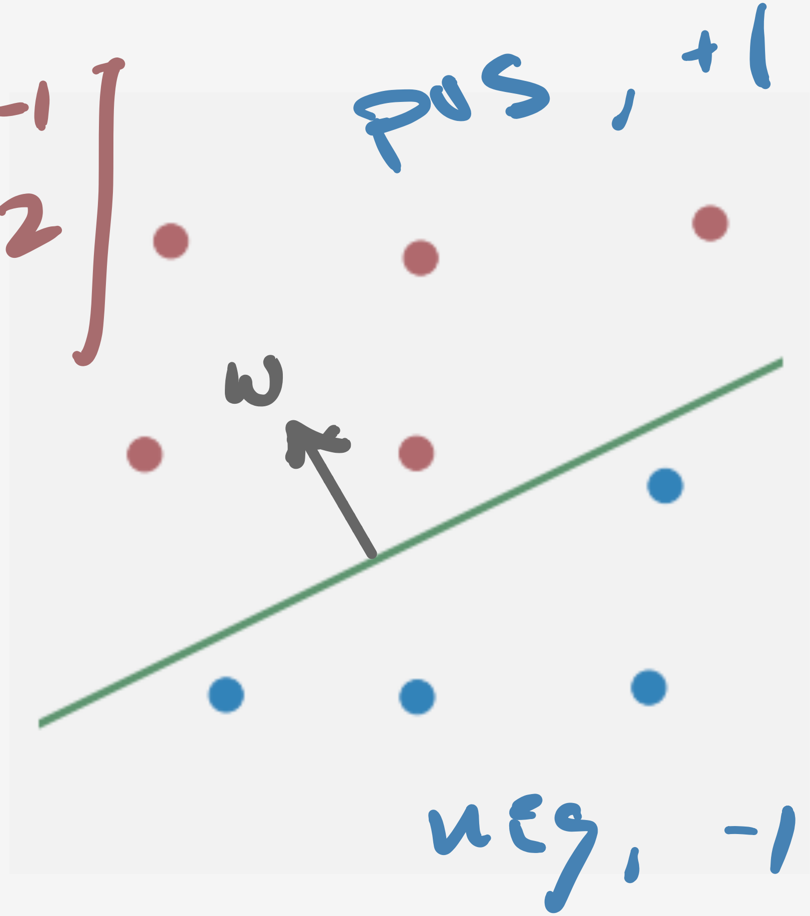
$\vec{x} = \langle x_1, x_2 \rangle$

Linear Classifier

- **Binary Classification**: Only two classes

$w = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

- A **linear classifier** draws a line through space separating the two classes.



- For two-features, a linear classifier takes form:

$$\hat{y} = a(\underline{w_1} x_1 + \underline{w_2} x_2 + \underline{b})$$

$w_2 > 0$

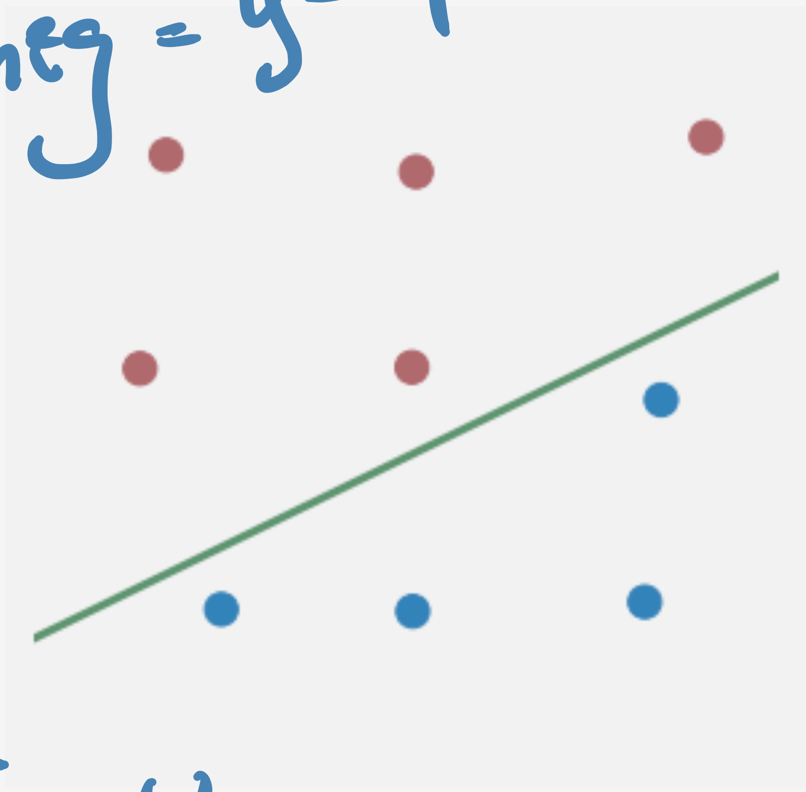
$$w_1 x_1 + w_2 x_2 + b = 0$$

$$x_2 = \frac{-(w_1 x_1 + b)}{w_2} = \underline{\underline{\frac{-b}{w_2}}} - \frac{w_1}{w_2} x_1$$

The Perceptron

- Assume that the labels are given by:

RED = pos $\Rightarrow y=1$ BLUE = neg = $y=-1$



- The perceptron activation function is given by:

$$a = \sum_{k=1}^{p} w_k x_k + b$$

DECISION RULE:

$$a \geq 0 \Rightarrow \hat{y} = +1$$

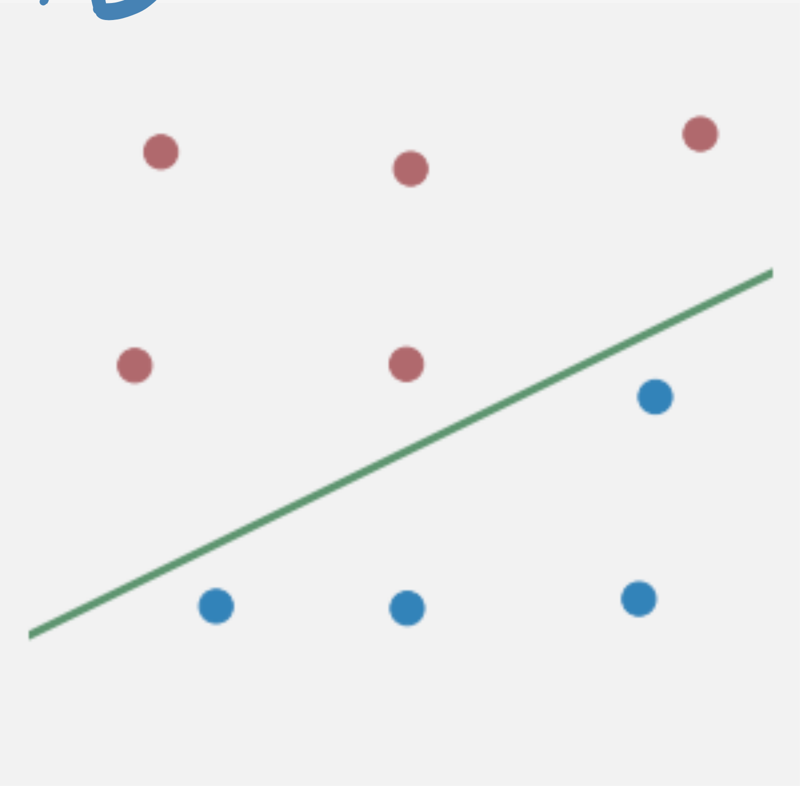
$$a < 0 \Rightarrow \hat{y} = -1$$

Perceptron Geometry

$$a = \sum_{k=1}^{p} w_k x_k + b \Rightarrow w^T x + b$$

- For two-features, a linear classifier takes form:

$$y = \mathrm{sign}\left(\underbrace{\mathbf{w}^T\mathbf{x} + b}_{a}\right) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$



The Perceptron Algorithm

initial guess for w & b

while **not converged**:

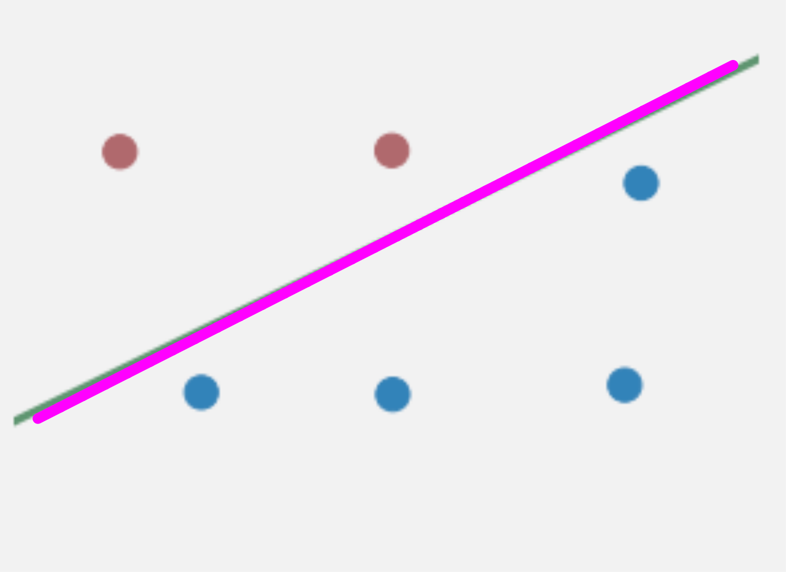
  for all $(\mathbf{x}, y)$ in training set: (randomize data)

    $a \leftarrow \sum_{k=1}^{p} w_k x_k + b = \underline{\mathbf{w}^T\mathbf{x} + b}$

    if $ya \leq 0$ then: (if not correct)

      $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$

      $b \leftarrow b + y$



Perceptron Algorithm Example

- Start with $\mathbf{w} = [1, 0]\,, b = 0$
- Process points in order:

$$(1,1), (0.5,-1), (-1,-1), (-1,1)$$

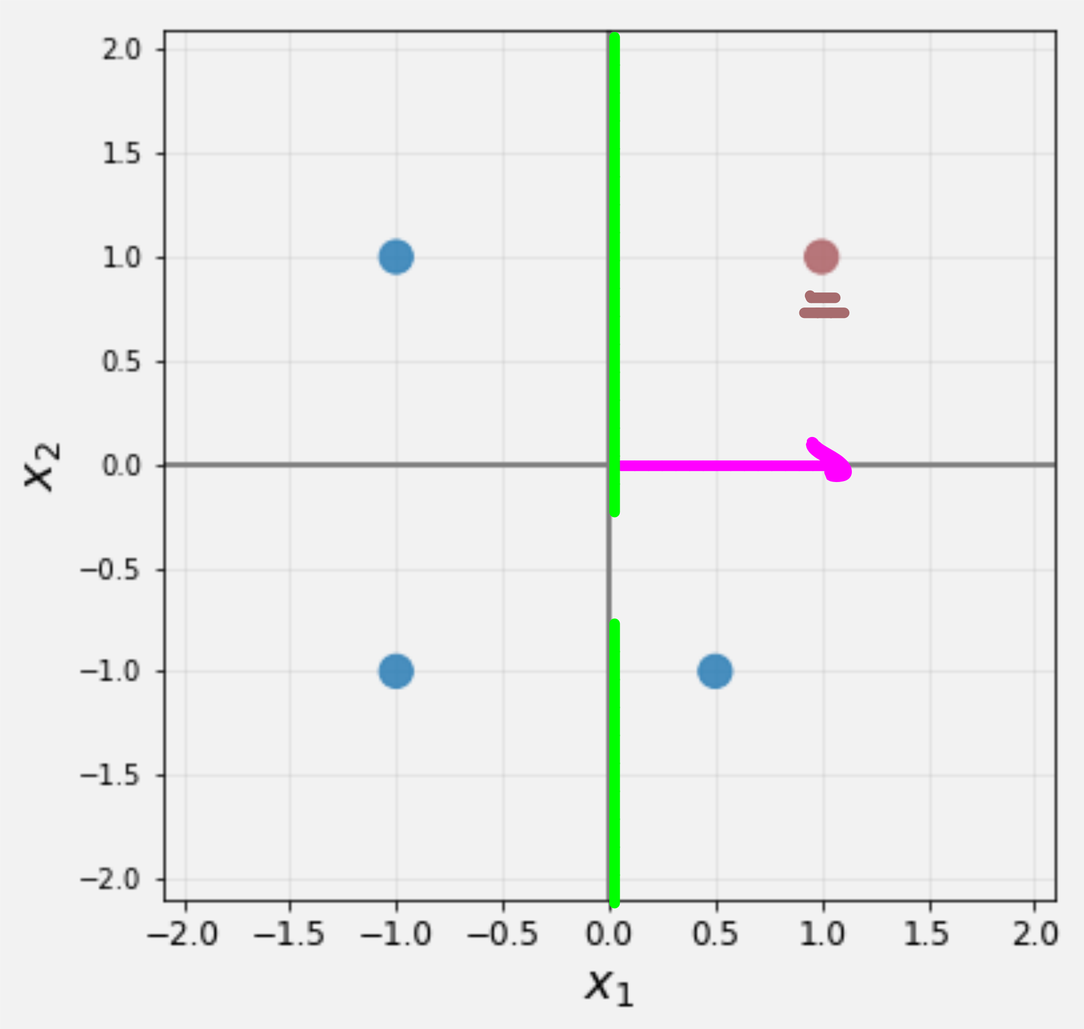
$\vec{x}_1 = (1,1)$

$a_1 = w^T \vec{x}_1 + b =$

$[1, 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0 = 1$

$a_1 \geq 0 \Rightarrow \hat{y}_1 = +1 = y_1$

$.5 + -1 = -.5$



Perceptron Algorithm Example

- Start with $\mathbf{w} = [1, 0], b = 0$
- Process points in order:

$$(1,1), (0.5,-1), (-1,-1), (-1,1)$$

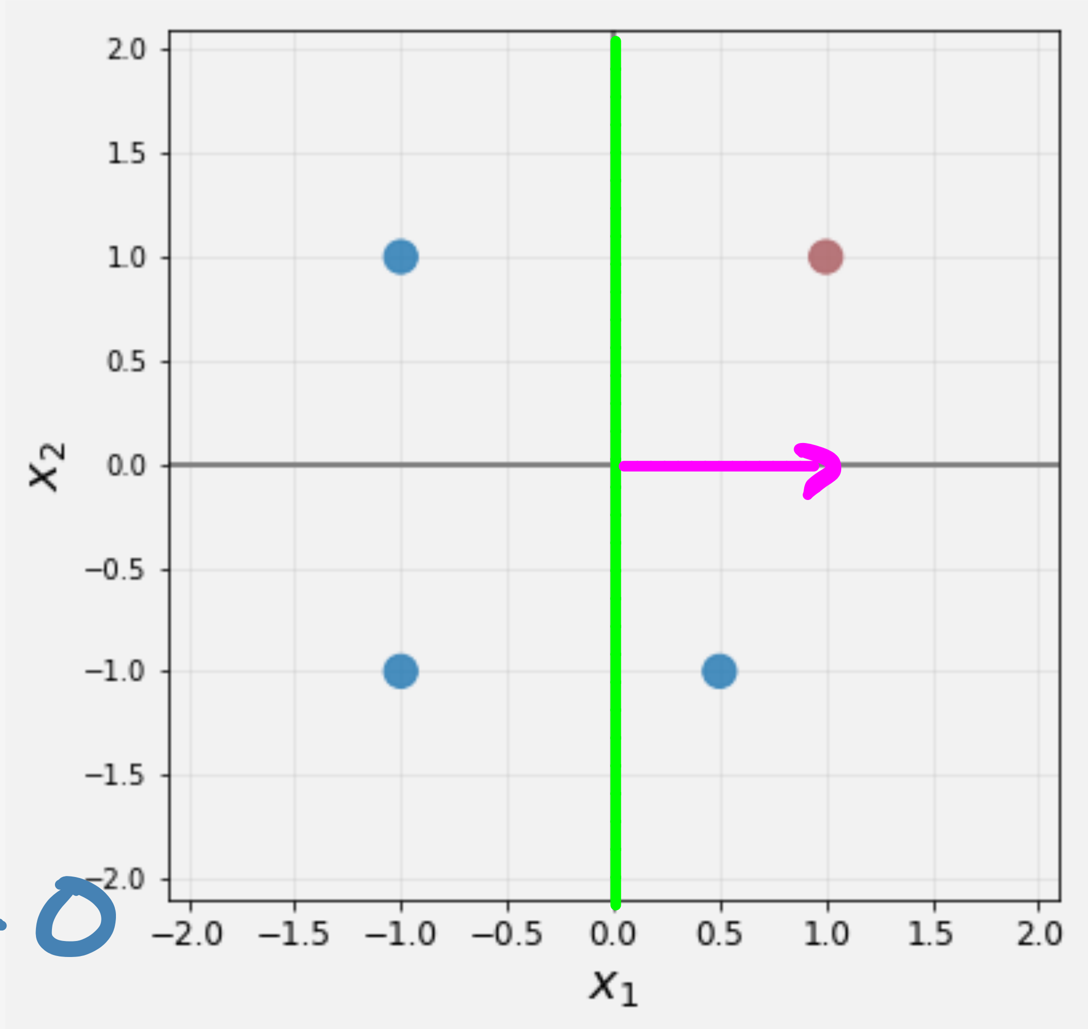
$\bar{x}_2 = \begin{bmatrix} .5 \\ -1 \end{bmatrix}$

$a_2 = [1, 0] \begin{bmatrix} .5 \\ -1 \end{bmatrix} + 0 = .5$

$a_2 = .5 \geq 0 \to$

$a_2 y_2 = .5(-1) = -.5 \leq 0$

wrong



Perceptron Algorithm Example

$\frac{-.5}{1} = -.5$

$x_2 = \frac{-b}{w_2} - \frac{w_1}{w_2} x_1$

$\frac{-(-1)}{1} = 1$

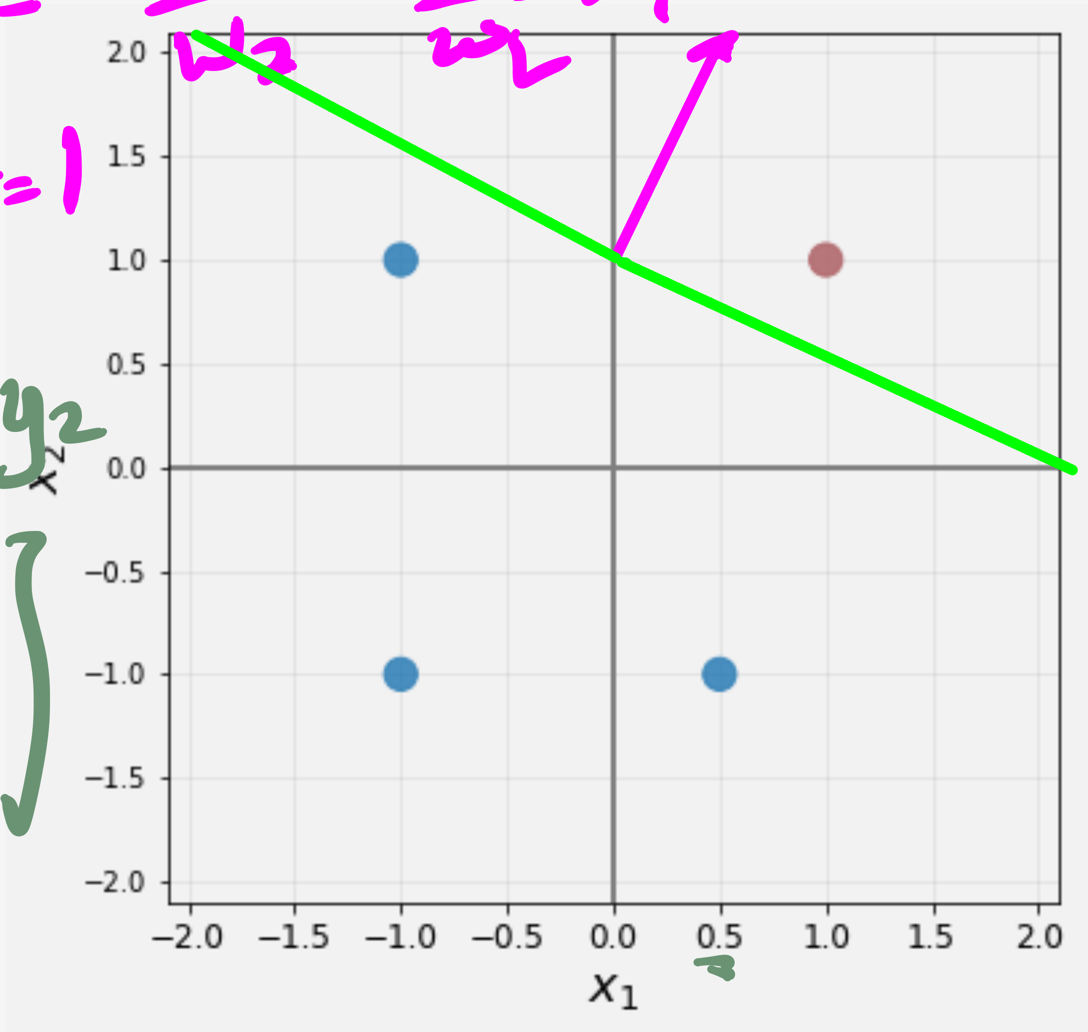
- Start with $\mathbf{w} = [1, 0]\,, b = 0$
- Process points in order:

$$(1,1), (0.5,-1), (-1,-1), (-1,1)$$

$w \leftarrow w + y_2 \vec{x}_2 \quad b \leftarrow b + y_2$

$w = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + (-1) \begin{bmatrix} .5 \\ -1 \end{bmatrix} = \begin{bmatrix} .5 \\ 1 \end{bmatrix}$

$b \leftarrow 0 + (-1) = -1$



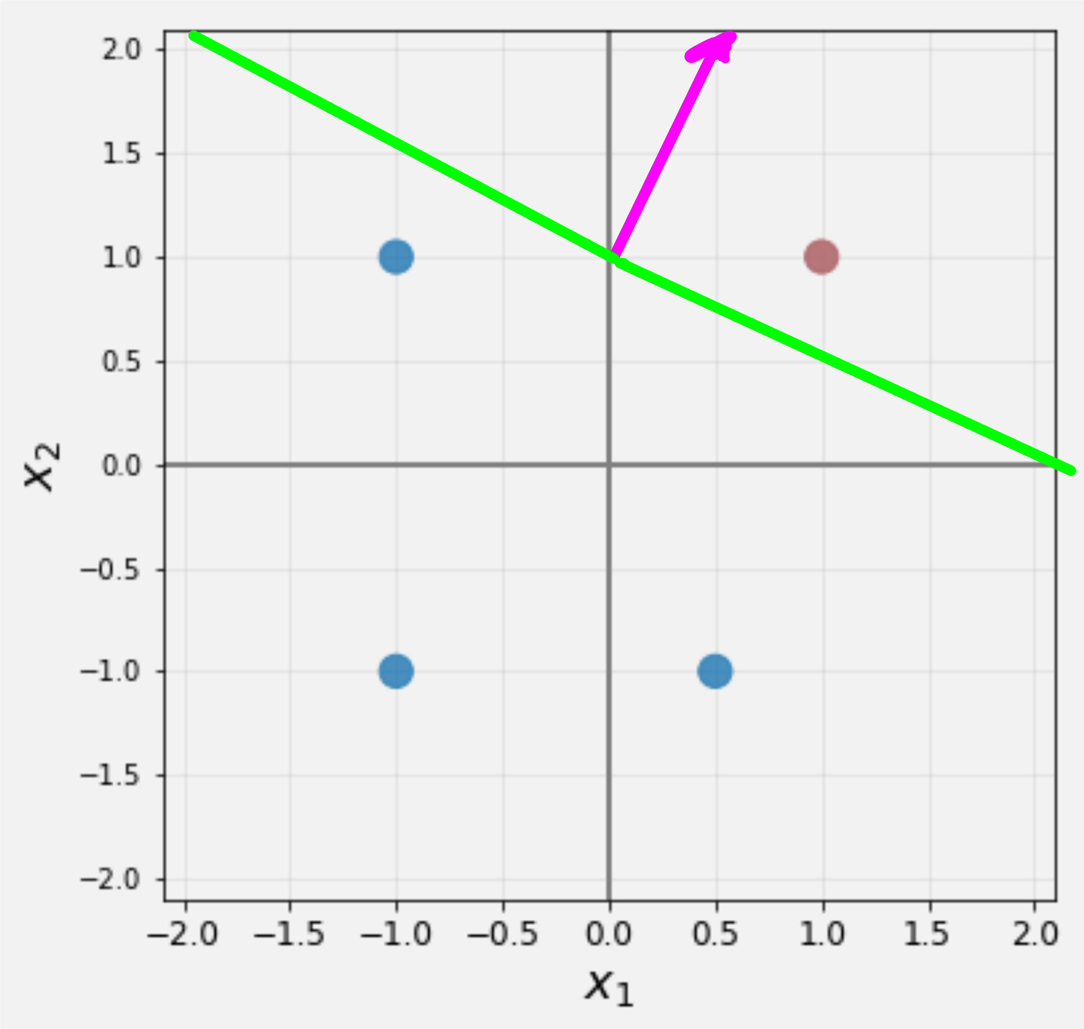
Perceptron Algorithm Example

- Start with $\mathbf{w} = [1, 0]\, , b = 0$
- Process points in order:

$$(1,1), (0.5,-1), (-1,-1), (-1,1)$$

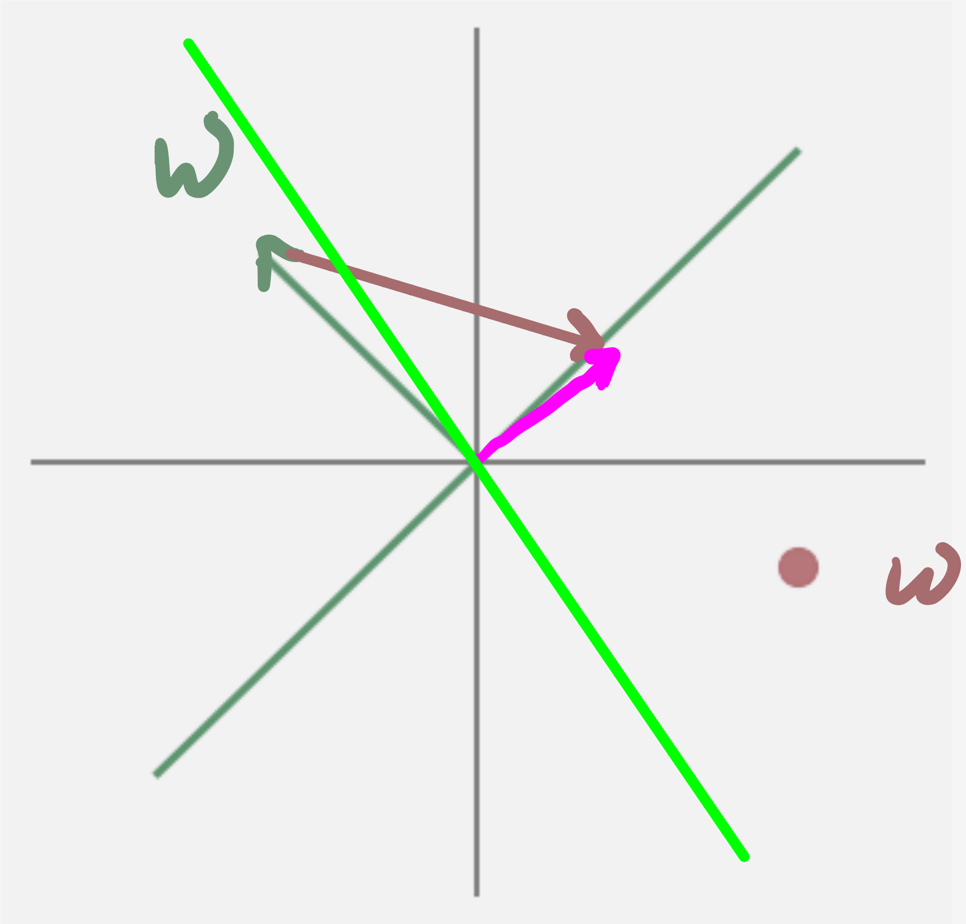
MODEL will classify all points correctly from now on

$w = \begin{bmatrix} .5 \\ 1 \end{bmatrix} \quad b = -1$



Why Does This Work (Geometry Edition)

$w \leftarrow w + y\vec{x}$



- wrong

Why Does This Work (Algebra Edition)

- Assume we have a current set of weights $w_1, w_2, \ldots, w_p$
- Assume we've just misclassified a point $(\mathbf{x}, y)$ with $y = +1$
- We compute activation $a < 0$ and update the weights and bias

$$
\begin{aligned}
a' &= \left[\sum_{k=1}^{p} w'_k x_k\right] + b' \\
&= \sum_{k=1}^{p} (w_k + x_k) x_k + (b+1) \\
&= \left[\sum_{k=1}^{p} (w_k x_k) + b\right] + \left[\sum_{k=1}^{p} x_k^2 + 1\right] \\
&= a + (\|x\|^2 + 1) \leftarrow \text{ALWAYS} > a
\end{aligned}
$$

Why Does This Work (Algebra Edition)

- Assume we have a current set of weights $w_1, w_2, \ldots, w_p$
- Assume we've just misclassified a point $(\mathbf{x}, y)$ with $y = -1$
- We compute activation $a > 0$ and update the weights and bias

$$
\begin{aligned}
a' &= \textstyle\sum_{k=1}^{p} w'_k x_k + b' \\
&= \\
&= \\
&=
\end{aligned}
$$

Perceptron Weight Interpretation

- Remember that we classify points according to

$$y = \operatorname{sign}\left(\mathbf{w}^T\mathbf{x} + b\right) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

- How sensitive is the final classification to changes in individual features?

Perceptron Weight Interpretation

- Remember that we classify points according to

$$y = \operatorname{sign}\left(\mathbf{w}^T\mathbf{x} + b\right) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

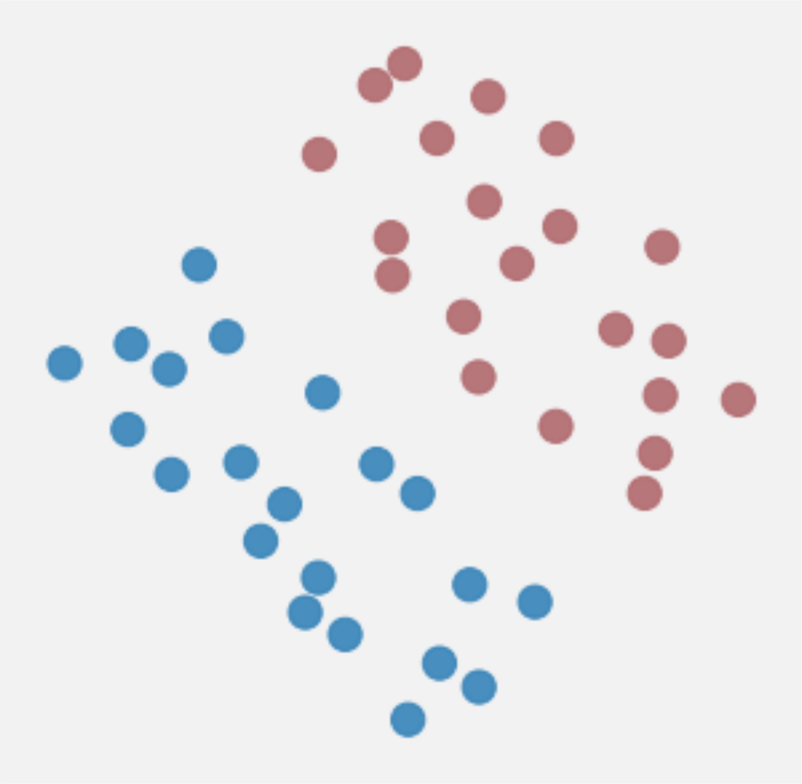
- How sensitive is the final classification to changes in individual features?

$$\frac{\partial}{\partial w_k}\left(\mathbf{w}^T\mathbf{x} + b\right) = \frac{\partial}{\partial w_k}\left(\sum_{k=1}^{p} w_k x_k + b\right) = \quad w_k$$

- If **features are similar size** then large weights indicate important features

Convergence of the Perceptron Algorithm

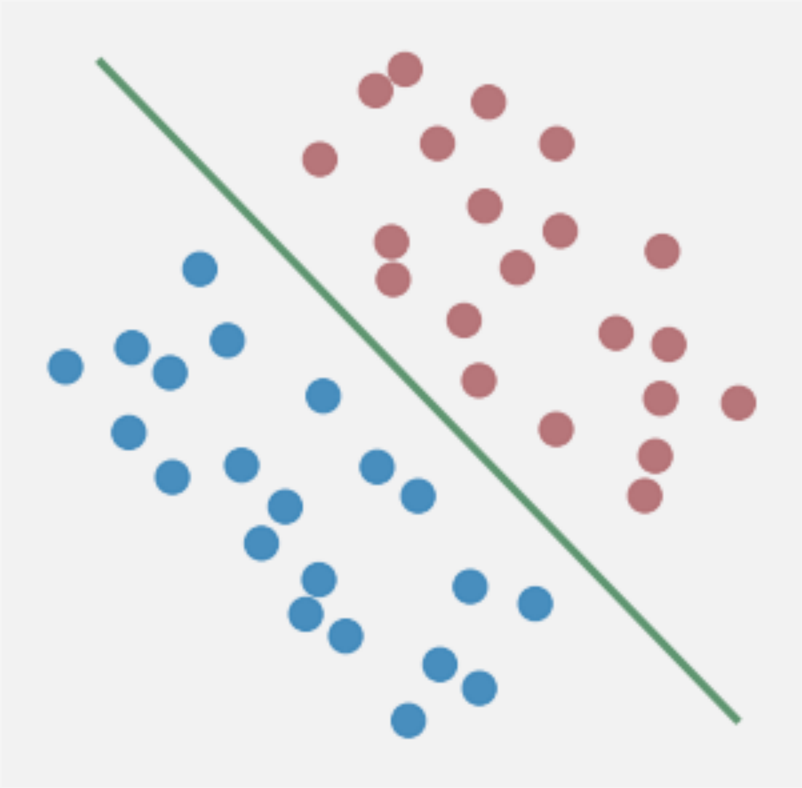
- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called **linearly separable**



Convergence of the Perceptron Algorithm

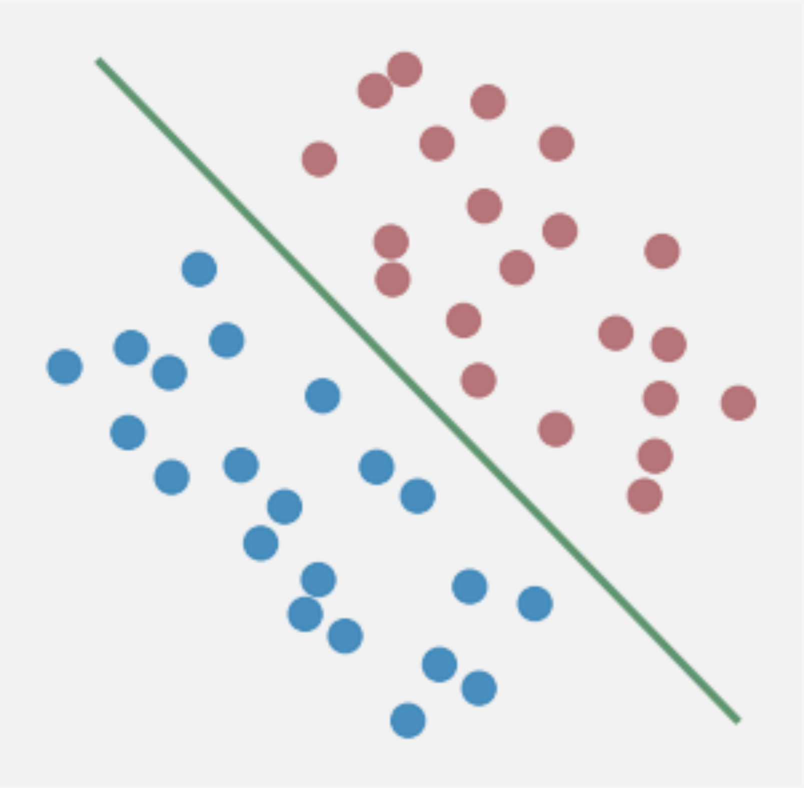
- If possible for a linear classifier to separate data, Perceptron will find it

- Such training sets are called **linearly separable**



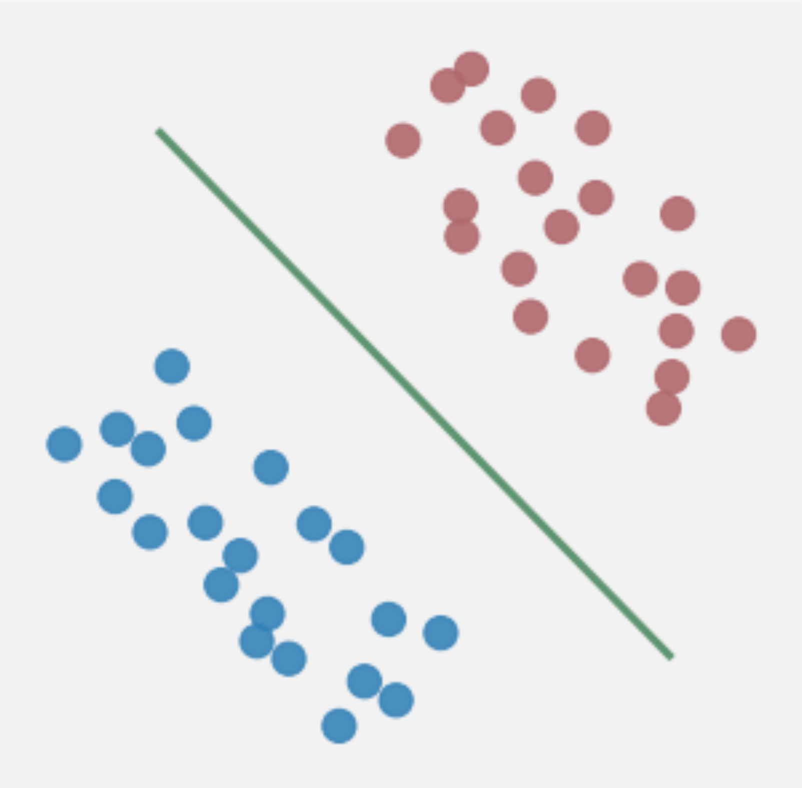
Convergence of the Perceptron Algorithm

- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called **linearly separable**
- How long it takes depends on depends on data



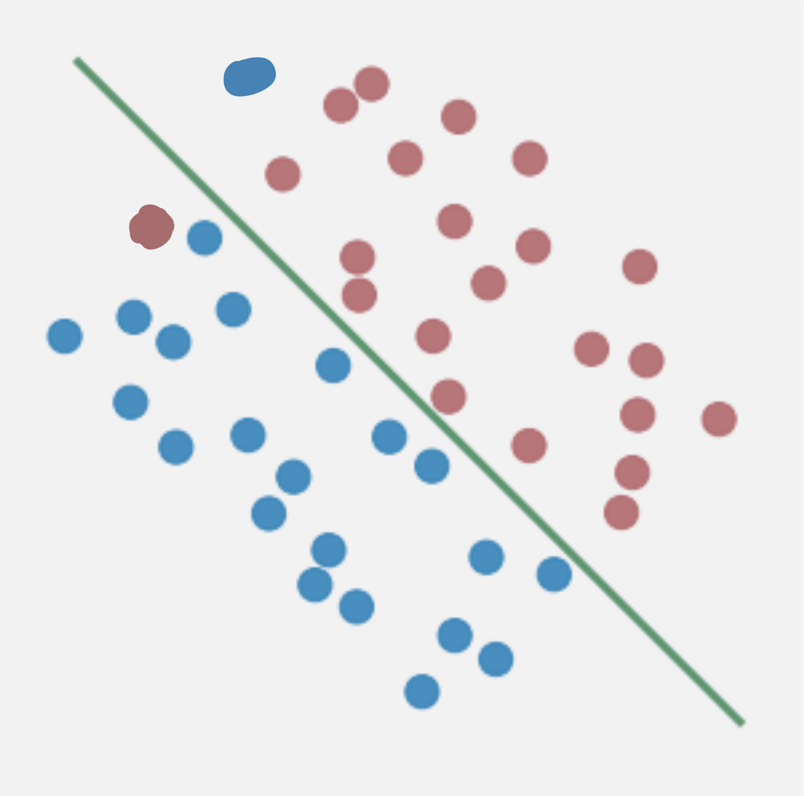
Convergence of the Perceptron Algorithm

- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called **linearly separable**
- How long it takes depends on depends on data



Convergence of the Perceptron Algorithm

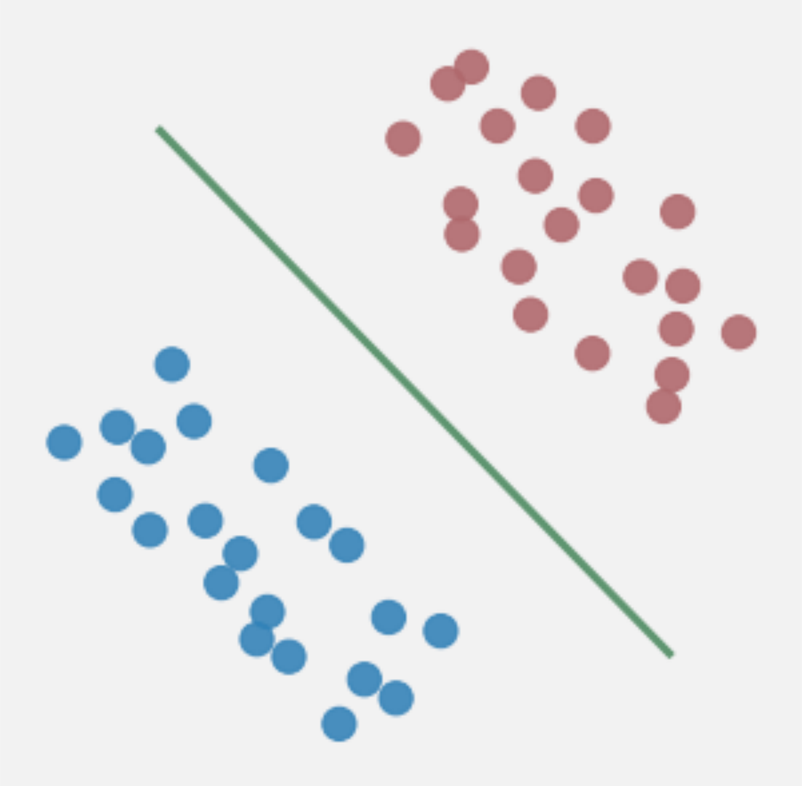
- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called **linearly separable**
- How long it takes depends on depends on data



Convergence of the Perceptron Algorithm

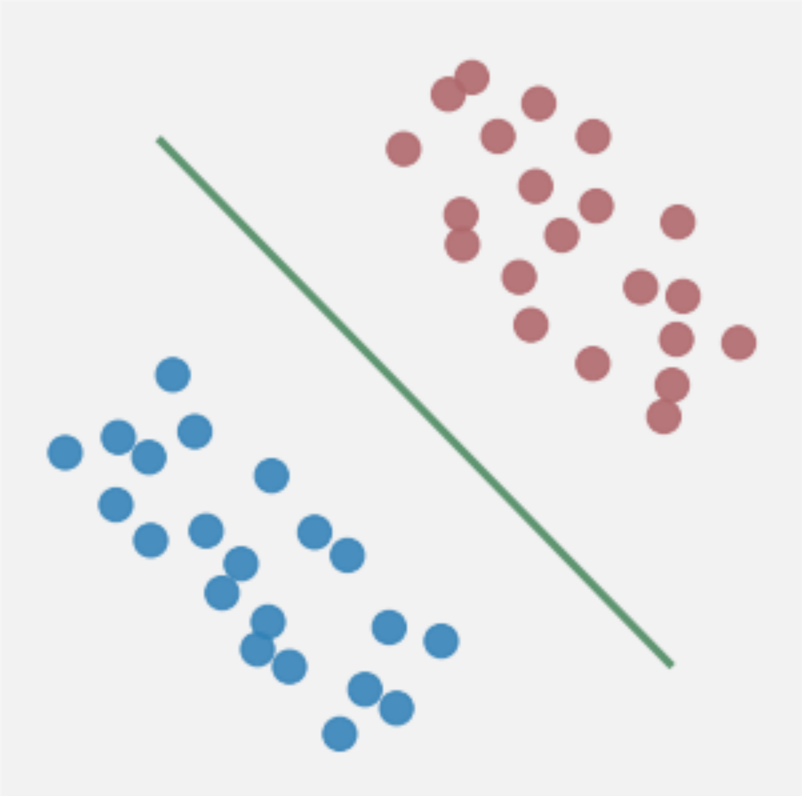
- If possible for a linear classifier to separate data, Perceptron will find it
- Such training sets are called **linearly separable**
- How long it takes depends on depends on data

Def: The margin of a classifier is the distance between decision boundary and nearest point.



Convergence of the Perceptron Algorithm

Perceptron Convergence Theorem: Suppose you train a perceptron on a **linearly separable** training set with margin $M > 0$. If $\|\mathbf{x}\| \leq 1$ then the algorithm will converge after at most $1/M^2$ updates.



Perceptron Wrap-Up

- The perceptron is a simple classifier that occasionally works very well
- Linear classifiers in general will pop up again and again
- Logistic Regression (next week) is pretty similar
- The idea of margins will show up again when we talk Support Vector Machines
- Neural Networks are essentially generalizations of the perceptron

If-Time Bonus: The Voting Perceptron

Suppose you have a data set with 10,000 training examples

Suppose that after 100 examples it's learned a really good set of weights

So good that for the next 9,899 examples it doesn't make any mistakes

And then on 10,000th example it misclassifies and totally changes the weights

Idea: Give more vote to weights that persist for a long time

If-Time Bonus: The Voting Perceptron

- Train as usual, save weights $(\mathbf{w}, b)^{(1)}, \ldots, (\mathbf{w}, b)^{(k)}$ and steps they persist $c^{(1)}, \ldots, c^{(k)}$

- Then predict using a weighted activation:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}(\mathbf{w}^{(k)^T}\mathbf{x} + b^{(k)})\right)$$

If-Time Bonus: The Voting Perceptron

- Train as usual, save weights $(\mathbf{w}, b)^{(1)}, \ldots, (\mathbf{w}, b)^{(k)}$ and steps they persist $c^{(1)}, \ldots, c^{(k)}$

- Then predict using a weighted activation:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}({\mathbf{w}^{(k)}}^T \mathbf{x} + b^{(k)})\right)$$

- A more efficient method is the **Averaged Perceptron**

$$\hat{y} = \text{sign}\left(\left(\sum_{k=1}^{K} c^{(k)} \mathbf{w}^{(k)}\right) \cdot \mathbf{x} + \sum_{k=1}^{K} c^{(k)} b^{(k)}\right)$$