The Perceptron Algorithm

- Classification setting: \( y \in \{-1,+1\} \)
- Linear model
  - Prediction: \( g = \text{Sign}(w \cdot x) \)

- Training:
  - Initialize weight vector: \( w(0) = 0 \) or something similar
  - At each time step:
    - Observe features: \( x^{(t)} \)
    - Make prediction: \( g^{(t)} = \text{Sign}(w^{(t)} \cdot x^{(t)}) \)
    - Observe true class: \( y^{(t)} \) is true label
    - Update model:
      - If prediction is not equal to truth:
        - \( w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)} \)
      - \( y^{(t)} \) makes mistake
        - \( y^{(t)} = -1 \)

\[ \text{Rosenblatt '58, '62} \]
What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
    \[ p(y | x, w) \]

- When we discussed the Perceptron:
  - Started from description of an algorithm

- What is the Perceptron optimizing????

Perceptron Prediction: Margin of Confidence
Hinge Loss

- Perceptron prediction: \( \text{Sign}(w \cdot x) \)
- Makes a mistake when: \( y(w \cdot x) < 0 \) →
- Hinge loss (same as maximizing the margin used by SVMs)

Minimizing hinge loss in Batch Setting

- Given a dataset: \( (x_1, y_1), \ldots, (x_n, y_n) \)
- Minimize average hinge loss:
  \[
  \min_w \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 
  0 & \text{if } y_i(w \cdot x_i) > 0 \\
  -y_i(w \cdot x_i) & \text{otherwise}
  \end{cases} 
  \]
- How do we compute the gradient?
Subgradients of Convex Functions

- Gradients lower bound convex functions:
  \[ F(w') \geq F(w) + \nabla F(w)^	op (w' - w) \]

- Gradients are unique at \( w \) iff function differentiable at \( w \)

- Subgradients: Generalize gradients to non-differentiable points:
  - Any plane that lower bounds function:
    \[ F(w') \geq F(w) + \nabla F(w)^	op (w' - w) \]
  
  \[ \text{For } |w| \to 0: \quad V \in [-1, 1] \quad \text{iff} \quad F(w') \geq F(w) + V (w' - w) \]

Subgradient of Hinge

- Hinge loss:

- Subgradient of hinge loss:
  - If \( y^{(i)} (w \cdot x^{(i)}) > 0 \): \[ \ell(w, x) = 0 \]
  - If \( y^{(i)} (w \cdot x^{(i)}) < 0 \): \[ \ell(w, x) = -y x \]
  - If \( y^{(i)} (w \cdot x^{(i)}) = 0 \): \[ \ell(w, x) = [-y x, 0] \quad \text{e.g. } -y x \]
  - In one line:
    \[ \ell(w, x) = \max \{ y (w \cdot x) \leq 0, -y x \} \]
    \[ \text{indicators of } m \text{ mistake} \]
Subgradient Descent for Hinge Minimization

- Given data: \((x^i, y^i) \ldots (x^n, y^n)\)

- Want to minimize:
  \[
  \frac{1}{N} \sum_{i=1}^{N} l(w, x^i) = \frac{1}{N} \sum_{i=1}^{N} (-y^i (w \cdot x^i))
  \]

- Subgradient descent works the same as gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:
    \[
    w^{(t+1)} = w^{(t)} - \eta \sum_{i=1}^{N} \mathbb{1}\{l(w, x^i) \neq 0\} \left(-y^i x^i\right)
    \]

Perceptron Revisited

- Perceptron update:
  \[
  w^{(t+1)} \leftarrow w^{(t)} + \mathbb{1}\left[y^{(t)} (w^{(t)} \cdot x^{(t)}) \leq 0\right] y^{(t)} x^{(t)}
  \]

- Batch hinge minimization update:
  \[
  w^{(t+1)} \leftarrow w^{(t)} + \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left[y^{(i)} (w^{(t)} \cdot x^{(i)}) \leq 0\right] y^{(i)} x^{(i)}
  \]

- Difference?
  - Perceptron update is a stochastic gradient descent algorithm for hinge loss minimization with a fixed step size (\(\eta > 0\))
What you need to know

- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective
Linear Separability: More formally, Using Margin

- Data linearly separable, if there exists
  - a vector \( \exists w \), \( \| w \| = 1 \)
  - a margin \( \gamma > 0 \)

  Such that

  \[
  \forall x \in \text{dataset}, \quad y > 0 \quad \text{or} \quad y \leq 0 \quad \text{or} \quad w^T x < -\gamma \quad \text{or} \quad w^T x > \gamma \\
  \text{for all points with } w^T x = 0 \text{ and } y \neq 0 \\
  \text{linearly separable, margin } \gamma
  \]

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: \( (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(T)}, y^{(T)}) \)
  - Each feature vector has bounded norm:
    \( \forall t, \| x^{(t)} \| \leq R \)
  - If dataset is linearly separable:
    \( \exists w, \| w \| = 1 \)
    \( \forall t, \quad y^{(t)} w^T x^{(t)} > 0 \) \( \text{for } y > 0 \)

  Then the number of mistakes made by the online perceptron on this sequence is bounded by

  \[
  \left( \frac{R}{\gamma} \right)^2 \leq \text{constant, independent of } T, \text{ dimensionality of } x
  \]
Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it’s done for ever!
    - Even if you see infinite data

- However, real world not linearly separable
  - Can’t expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire
  - Converges, but ultimately may not give good accuracy (make many many mistakes)

What if the data is not linearly separable?

- Use features of features of features....

\[ \Phi(x) : \mathbb{R}^{\infty} \rightarrow F \]

Feature space can get really large really quickly!
Higher order polynomials

\[ \text{num. terms} = \binom{d + m - 1}{d} = \frac{(d + m - 1)!}{d!(m - 1)!} \]

- \( m \) – input features
- \( d \) – degree of polynomial

Even though the dimensions of \( \phi(x) \) are huge, the model very quickly grows fast!

\( d = 6, m = 100 \) about 1.6 billion terms