Machine Learning

4771

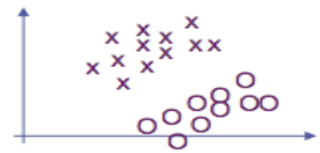
Instructors:
Adrian Weller and Ilia Vovsha

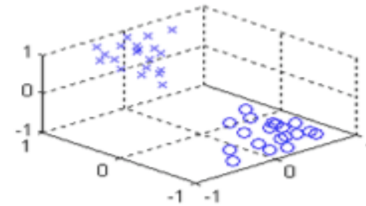
Lecture 6: Perceptron

- Linear decision surface
- Perceptron (Duda 5.1-5.5)
- Convergence proof
- Neural Networks (Bishop 5.1-5.3.2)

Linearly Separable 2-Class Problem

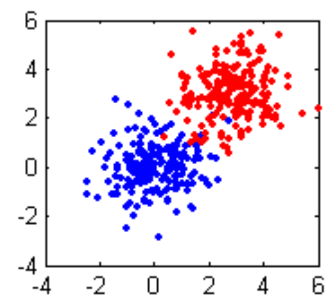
- Start with training dataset





$$\mathcal{X} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \quad x \in \mathbb{R}^D \quad y \in \{-1, 1\}$$

- Have N (vector, label {-1,1}) pairs
- Find a discriminant function f(x) to predict class (label) from x
- Assume there exists a weight vector **w** that classifies all samples correctly
    - Such **w** is called a solution vector
    - More than one (infinite #) **w**: solution region
    - We say the data is "linearly seperable"
    - Otherwise "non-separable" (example on the right)



- Symmetry: $\dfrac{w^T x_i \geq 0: \quad assign\ 1}{w^T x_i < 0: \quad assign\ -1} \Rightarrow y_i(w^T x_i) > 0$

Gradient Descent

- We have a set of linear inequalities, we want to find a solution vector

$$\forall i : y_i(w^T x_i) > 0$$

- Approach: define a loss function to minimize

$$L(y, f(x)) = h(-yf(x)) = step(-yf(x))$$

$$L(w) = h(-yw^T x) = step(-yw^T x)$$

- What if we can't get minimum in closed form?
  - ➢ Do gradient descent
  - ➢ Gradient points in direction of fastest increase
  - ➢ Take step in the opposite direction

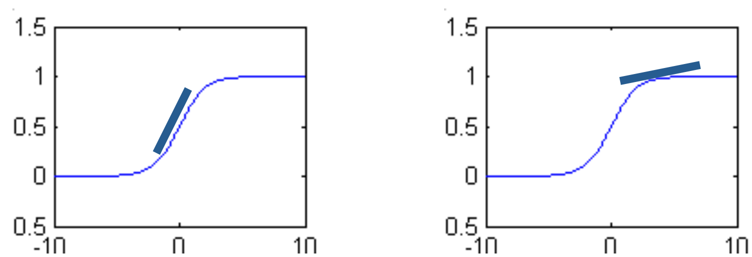
Gradient Descent Algorithm

- General Algorithm (any loss function)
    1. Fix step size η and threshold ε to some value
    2. Initialize: w^0 = random vector, k = 0 (counter)
    3. Update vector: $\mathbf{w}^{k+1} = \mathbf{w}^{k} - \eta \nabla R(\mathbf{w})$
    4. Increment counter: k = k+1
    5. If
    $$\left| R(\mathbf{w}^k) - R(\mathbf{w}^{k-1}) \right| > \varepsilon \quad OR \quad \left| \eta \nabla R(\mathbf{w}) \right| > \varepsilon$$
    go to step # 3.

- For appropriate learning rate η, guaranteed to converge to a local minimum

Learning Rate

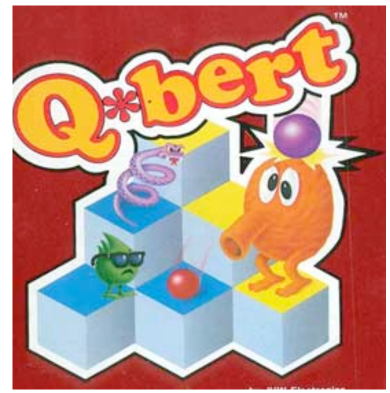
- Pick the step size scalar (learning rate) well so that each step reduces R(w)
- If step size is too small → slow. If too large → unstable
- Also, need to avoid flat regions in the space → slow



- Rate can be time (counter) dependent → large steps early on, small steps closer to the solution

Perceptron Criterion/Loss

- Recall: to do gradient descent, need reasonable gradients
- Currently have staircase-shaped (piece-wise constant) risk function
  - ➢ Hard to minimize
  - ➢ The gradient is zero except at edges when a label flips



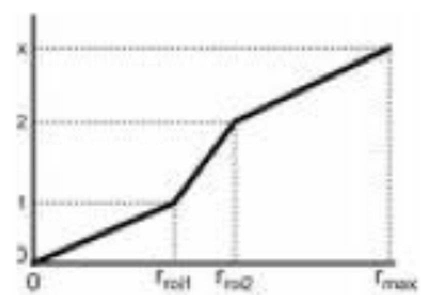
$$L(w) = step(-yw^T x)$$

$$R(w) = \frac{1}{N}\sum_{i=1}^{N} step(-y_i w^T x_i)$$

- Instead of misclassification count, consider Perceptron loss:

$$R^{per}(w) = -\sum_{i \in misclassified} (y_i w^T x_i)$$

- Get smooth piece-wise linear risk:



Perceptron Update Rule

- Obtain gradient for perceptron risk & plug in the general algorithm:

$$R^{per}(\mathbf{w}) = -\sum_{i \in misclassified} (y_i w^T x_i)$$

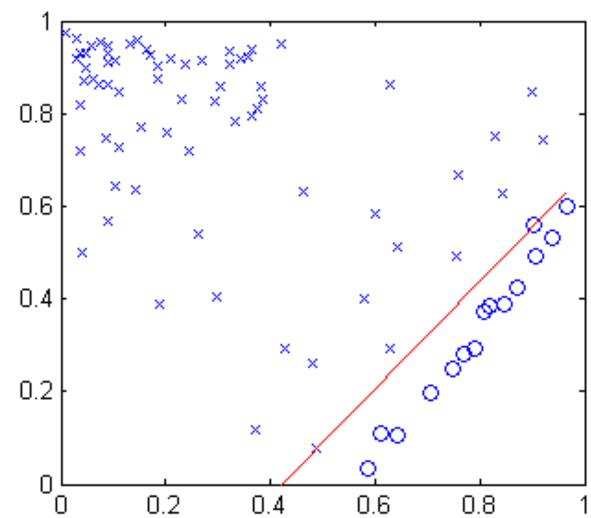
$$\nabla_w R^{per}(\mathbf{w}) = -\sum_{i \in M} y_i x_i$$

$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta \nabla R^{per}(\mathbf{w}) \\ &= \mathbf{w}^k + \eta \sum_{i \in M} y_i x_i \end{aligned}$$

Perceptron Algorithm

- Also known as "batch perceptron"
  1. Fix step size η and threshold ε to some value
  2. Initialize: w^0 = random vector, k = 0 (counter)
  3. Update vector: $\mathbf{w}^{k+1} = \mathbf{w}^{k} + \eta \sum_{i \in M} y_i x_i$
  4. Increment counter: k = k+1
  5. If $\left| \eta \sum_{i \in M} y_i x_i \right| > \varepsilon$

     go to step # 3.



Online Perceptron

- How good is the algorithm?
  - ➢ Convergence properties:
    1. Does it converge to a solution? (consistency)
    2. How fast does it converge? (rate of convergence)
- Idea: to simplify the proof of convergence, consider cycling through the examples one at a time (sequence instead of batch)
  - ➢ Update rule for each mis-classified point by itself
  - ➢ Skip correctly classified points (no update)
  - ➢ Stochastic Gradient Descent
  - ➢ Fix learning rate (w.l.o.g) $\eta=1$

Online Algorithm

- Also known as "single-sample perceptron"

1. Initialize: w^0 = random vector, t=1, k = 0 (counters)
2. If y_t is misclassified by w^k, update vector: $\mathbf{w}^{k+1} = \mathbf{w}^k + y_t x_t$

   Otherwise, no update: $\mathbf{w}^k = \mathbf{w}^k$
3. Increment counter: t = (t+1) mod N
4. If all examples are classified correctly, stop. Otherwise go back to step 2.

Convergence Proof

• ***Theorem***: assuming conditions {1,2} below are satisfied, the sequence of weight vectors determined by the online perceptron algorithm will converge to a solution vector in finite number of steps

1. Assume all data lies inside a sphere of radius r: $r = \max_i \|x_i\|$
2. Assume that the data is linearly separable: $\forall i : y_i((w^*)^T x_i) \geq \gamma > 0$

• ***Proof***: to show convergence we consider the angle between the optimal ($\mathbf{w}^*$) & current ($\mathbf{w}^k$) solution. Applying conditions {1,2} we can bound the norm of $\mathbf{w}^k$ & the dot product $\mathbf{w}^* \bullet \mathbf{w}^k$. Algebraic manipulation then yields a finite upper bound on k (number of steps)

1. Angle between optimal ($\mathbf{w}^*$) & current ($\mathbf{w}^k$) solution
2. Bound the dot product $\mathbf{w}^* \bullet \mathbf{w}^k$, and the norm of $\mathbf{w}^k$
3. Substitute and manipulate to get upper bound on k

Convergence Proof

- ***Step 1 (angle)***: $\cos\left(\mathbf{w}^*,\ \mathbf{w}^k\right) = \dfrac{\left(\mathbf{w}^*\right)^T \mathbf{w}^k}{\left\|\mathbf{w}^*\right\|\left\|\mathbf{w}^k\right\|} \le 1$

- ***Step 2 (bound numerator & norm)***:

$$\left(\mathbf{w}^*\right)^T \mathbf{w}^k = \left(\mathbf{w}^*\right)^T \mathbf{w}^{k-1} + y_i\left(\left(\mathbf{w}^*\right)^T x_i\right)$$

$$\ge \left(\mathbf{w}^*\right)^T \mathbf{w}^{k-1} + \gamma \ge k\gamma$$

$$\left\|\mathbf{w}^k\right\|^2 = \left\|\mathbf{w}^{k-1} + y_i x_i\right\|^2$$

$$= \left\|\mathbf{w}^{k-1}\right\|^2 + 2y_i\left(\left(\mathbf{w}^{k-1}\right)^T x_i\right) + \left\|x_i\right\|^2$$

$$\le \left\|\mathbf{w}^{k-1}\right\|^2 + r^2 \le kr^2$$

(1) $r = \max_i \left\|x_i\right\|$

(2) $\forall i : y_i((w^*)^T x_i) \ge \gamma$

(3) $\mathbf{w}^k = \mathbf{w}^{k-1} + y_i x_i$

Convergence Proof

- ***Step 1 (angle)***: $\cos\left(\mathbf{w}^*,\ \mathbf{w}^k\right) = \dfrac{\left(\mathbf{w}^*\right)^T \mathbf{w}^k}{\left\|\mathbf{w}^*\right\|\left\|\mathbf{w}^k\right\|} \leq 1$

- ***Step 2 (bound numerator & norm)***:

$$\left(\mathbf{w}^*\right)^T \mathbf{w}^k \geq k\gamma$$

$$\left\|\mathbf{w}^k\right\|^2 \leq kr^2$$

- ***Step 3 (bound on k)***: $1 \geq \dfrac{\left(\mathbf{w}^*\right)^T \mathbf{w}^k}{\left\|\mathbf{w}^*\right\|\left\|\mathbf{w}^k\right\|} \geq \dfrac{k\gamma}{\left\|\mathbf{w}^*\right\|\left\|\mathbf{w}^k\right\|} \geq \dfrac{k\gamma}{\left\|\mathbf{w}^*\right\|\sqrt{kr^2}}$

$$\sqrt{k} \leq \frac{\left\|\mathbf{w}^*\right\|\sqrt{r^2}}{\gamma} \Rightarrow k \leq \frac{r^2}{\gamma^2}\left\|\mathbf{w}^*\right\|^2$$

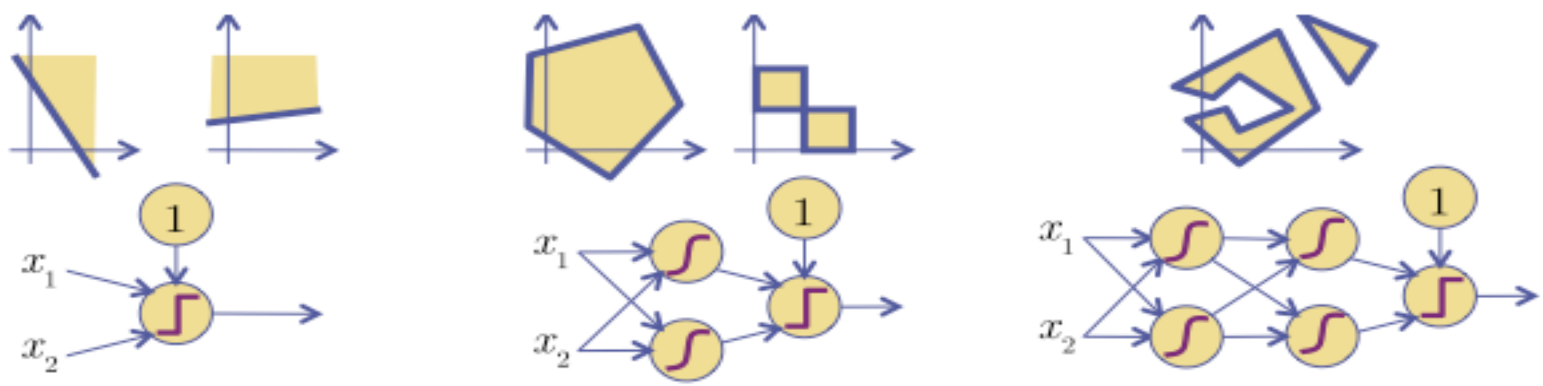
Perceptron Deficiencies

- Many Deficiencies!

1. Multiple (infinite #) solutions, which is best?
2. Actual solution depends on initialization
3. Data is not linearly-separable? Algorithm doesn't converge!
4. Slow convergence in practice
5. Algorithm lacks straight-forward generalization to multi-class problems
6. Can't solve the XOR problem (more generally nonlinear problems)

Multi-Layer Neural Network (idea)

- 1-layer (perceptron): can't even handle XOR!
- What if we consider cascading multiple layers of network?
- Each output layer is input to the next layer
- Each layer has its own weights (parameters)
- Each layer adds more flexibility (but more parameters!)
- Each node splits its input space with linear hyperplane



- Multi-Layer Network can handle more complex decisions
- Note: Without loss of generality, we can use augmented vectors