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)


Linearily 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 “linearily separable”
  - Otherwise “non-separable” (example on the right)

- Symmetry:
  \[ \frac{w^T x_i \geq 0}{w^T x_i < 0} : \quad \begin{array}{l}
    \text{assign} \; 1 \\
    \text{assign} \; -1
  \end{array} \Rightarrow y_i(w^T x_i) > 0 \]
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 $\eta$ and threshold $\varepsilon$ to some value
2. Initialize: $w^0 = \text{random vector}, k = 0$ (counter)
3. Update vector: $w^{k+1} = w^k - \eta \nabla R(w)$
4. Increment counter: $k = k+1$
5. If
   \[
   \left| R(w^k) - R(w^{k-1}) \right| > \varepsilon \quad \text{OR} \quad |\eta \nabla R(w)| > \varepsilon
   \]
   go to step # 3.

• For appropriate learning rate $\eta$, 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 $\rightarrow$ slow. If too large $\rightarrow$ unstable
- Also, need to avoid flat regions in the space $\rightarrow$ slow

- Rate can be time (counter) dependent $\rightarrow$ 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

\[
L(w) = \text{step}( - yw^T x )
\]

\[
R(w) = \frac{1}{N} \sum_{i=1}^{N} \text{step}( - y_iw^T x_i )
\]

• Instead of misclassification count, consider Perceptron loss:

\[
R_{\text{per}}(w) = \sum_{i \in \text{misclassified}} (y_iw^T x_i)
\]

• Get smooth piece-wise linear risk:
Perceptron Update Rule

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

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

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

\[
w^{k+1} = w^k - \eta \nabla R^{per}(w)
\]

\[
= w^k + \eta \sum_{i \in M} y_i x_i
\]
Perceptron Algorithm

• Also known as “batch perceptron”

1. Fix step size $\eta$ and threshold $\varepsilon$ to some value
2. Initialize: $w^0 = \text{random vector}, k = 0$ (counter)
3. Update vector: $w^{k+1} = w^k + \eta \sum_{i \in M} y_i x_i$
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: $w^0 = \text{random vector}$, $t=1$, $k = 0$ (counters)
2. If $y_t$ is misclassified by $w^k$, update vector: $w^{k+1} = w^k + y_t x_t$
   Otherwise, no update: $w^k = w^k$
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 \((w^*)\) & current \((w^k)\) solution. Applying conditions \{1,2\} we can bound the norm of \(w^k\) & the dot product \( w^* \cdot w^k \). Algebraic manipulation then yields a finite upper bound on \( k \) (number of steps)

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

- **Step 1 (angle):** \( \cos(w^*, w^k) = \frac{(w^*)^T w^k}{\|w^*\| \|w^k\|} \leq 1 \)  
  \( r = \max_i \|x_i\| \)  

- **Step 2 (bound numerator & norm):**  
  \[
  \begin{align*}
  (w^*)^T w^k &= (w^*)^T w^{k-1} + y_i ((w^*)^T x_i) \\
  &\geq (w^*)^T w^{k-1} + \gamma \geq k\gamma \\
  \|w^k\|^2 &= \|w^{k-1} + y_i x_i\|^2 \\
  &= \|w^{k-1}\|^2 + 2y_i ((w^{k-1})^T x_i) + \|x_i\|^2 \\
  &\leq \|w^{k-1}\|^2 + r^2 \leq kr^2
  \end{align*}
  \]
Convergence Proof

• **Step 1 (angle):** \( \cos(w^*, w^k) = \frac{(w^*)^T w^k}{\|w^*\| \|w^k\|} \leq 1 \)

• **Step 2 (bound numerator & norm):**
  \[
  (w^*)^T w^k \geq k\gamma \\
  \|w^k\|^2 \leq kr^2
  \]

• **Step 3 (bound on k):**
  \[
  1 \geq \frac{(w^*)^T w^k}{\|w^*\| \|w^k\|} \geq \frac{k\gamma}{\|w^*\| \|w^k\|} \geq \frac{k\gamma}{\|w^*\| \sqrt{kr^2}}
  \]
  \[
  \sqrt{k} \leq \frac{\|w^*\| \sqrt{r^2}}{\gamma} \Rightarrow k \leq \frac{r^2}{\gamma^2 \|w^*\|^2}
  \]
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