Artificial Intelligence
Making your agent learn: ANN

Lecture 8
The classification problem ...

\[ x \quad f(x) \quad y \]

Hello,
Do you want free printr cartriges? Why pay more when you can get them ABSOLUTELY FREE! Just

\[
\begin{aligned}
\# \text{ free} & : 2 \\
\text{YOUR\_NAME} & : 0 \\
\text{MISSPELLED} & : 2 \\
\text{FROM\_FRIEND} & : 0 \\
\ldots
\end{aligned}
\]

SPAM or +

2

\[
\begin{aligned}
\text{PIXEL}-7,12 & : 1 \\
\text{PIXEL}-7,13 & : 0 \\
\ldots
\text{NUM\_LOOPS} & : 1 \\
\ldots
\end{aligned}
\]

“2”
Neural Networks (One Way)

- Very loose inspiration: human neurons
Linear Classifiers (Perceptron flavor)

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

activation\(_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)\)

- If the activation is:
  - Positive, output +1
  - Negative, output -1
How much should be the weights?

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

Dot product $w \cdot f$ positive means the positive class
Decision Making (Binary)

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to $Y=+1$
  - Other corresponds to $Y=-1$

$$w$$

BIAS : -3
free : 4
money : 2
...

$$f \cdot w = 0$$

+1 = SPAM
-1 = HAM

money
f
free
Learning the weights (weight vector)

- Start with weights = 0
- For each training instance:
  - Classify with current weights
    \[
    y = \begin{cases} 
    +1 & \text{if } w \cdot f(x) \geq 0 \\
    -1 & \text{if } w \cdot f(x) < 0 
    \end{cases}
    \]
  - If correct (i.e., \( y = y^* \)), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if \( y^* \) is -1.

\[
w = w + y^* \cdot f
\]
Perceptron Learning Rule: Details

\[ w_i \leftarrow w_i + \Delta w_i \]

where

\[ \Delta w_i = \eta(t - o)x_i \]

Where:

- \( t = c(\vec{x}) \) is target value
- \( o \) is perceptron output
- \( \eta \) is small constant (e.g., .1) called learning rate

\[ \theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \]
Will the rule work?

Can prove it will converge

• If training data is linearly separable
• and $\eta$ sufficiently small
Sigmoid Unit Vs. Linear Unit

Sigmoid Unit:

$$\text{net} = \sum_{i=0}^{n} w_i x_i$$

$$o = \sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}}$$

Linear Unit:

$$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$
Why Sigmoid?

\[ \sigma(x) \text{ is the sigmoid function} \]

\[ \frac{1}{1 + e^{-x}} \]

Nice property: \[ \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \]

We can derive gradient decent rules to train

- One sigmoid unit
- Multilayer networks of sigmoid units \(\rightarrow\) Backpropagation
What’s the LMS involved?

\[
\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2
\]

One training sample

\[
\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}
\]
What if my decision is not binary?
Multi-class Perceptron

- If we have multiple classes:
  - A weight vector for each class: $w_y$
  - Score (activation) of a class $y$: $w_y \cdot f(x)$
  - Prediction highest score wins

$$y = \text{arg max}_y \ w_y \cdot f(x)$$

*Binary = multiclass where the negative class has weight zero*
Learning: Multi-class Perceptron

- Start with all weights $= 0$
- Pick up training examples one by one
- Predict with current weights

\[ y = \arg \max_y w_y \cdot f(x) \]

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

\[ w_y = w_y - f(x) \]
\[ w_y^* = w_y^* + f(x) \]
Separability

- Separable Case
The troubling case ...

- Non-Separable Case
Perceptron properties

- Separability: some parameters get the training set perfectly correct.

- Convergence: if the training is separable, perceptron will eventually converge (binary case).

- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability.

\[ \text{mistakes} < \frac{k}{\delta^2} \]
How to deal with non-separability?
Strategy A: Multi-layer Perceptron
What is it’s power?
Backpropagation Algorithm

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit \( k \):
     \[
     \delta_k \leftarrow o_k(1-o_k)(t_k-o_k)
     \]
  3. For each hidden unit \( h \):
     \[
     \delta_h \leftarrow o_h(1-o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k
     \]
  4. Update each network weight \( w_{i,j} \):
     \[
     w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}
     \]
     where
     \[
     \Delta w_{i,j} = \eta \delta_j x_{i,j}
     \]