Adaline

Nan Ye

School of Mathematics and Physics
The University of Queensland

Non-linearly Separable Data

- The perceptron learns a separating hyperplane when the data is linearly separable, but fails when the data is not linearly separable.
- Can we design an algorithm that works for non-linearly separable data? The answer is most likely no in general.
- Specifically, it is NP-hard to minimize the classification error for the perceptron, that is, given a training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, it is NP-hard to solve

$$\min_{\mathbf{w}} \frac{1}{n} I(\mathrm{sgn}(\mathbf{x}_i^\top \mathbf{w}) \neq y_i)$$

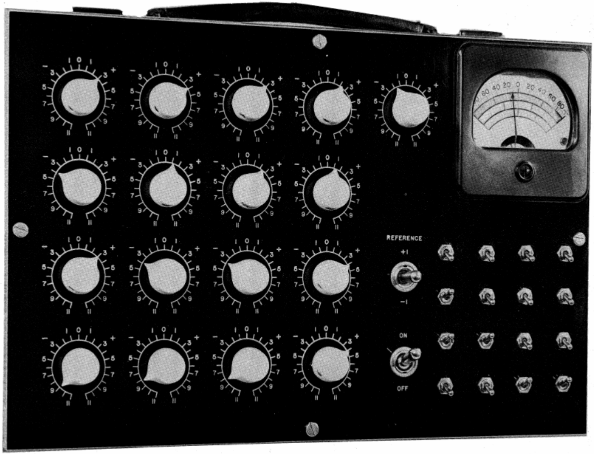
If you have not learned about computational complexity, NP-hardness basically means there is no efficient algorithm to solve the problem.

Dealing with Intractability

- Idea 1: hand-code powerful features
  - hand-coding good features are often hard
  - the hand-coded features may not make the data linearly separable
- Idea 2: optimize a surrogate objective
  - instead of trying to minimize the classification error, we minimize another objective function
  - this surrogate need to be computationally easy to minimize, and well-correlated with classification error

Adaline

- Widrow and Hoff (1960) developed an algorithm that is similar to the perceptron, but more stable than the perceptron when the data is non-linearly separable.
- The algorithm is known by various names: Adaline (Adaptive Linear), LMS (least mean square) rule,, Widrow-Hoff rule, or delta rule.
- They also built a learning device, also named as Adaline, to implement the Adaline learning rule.



The Adaline is a lunch box sized machine

Surrogate objective

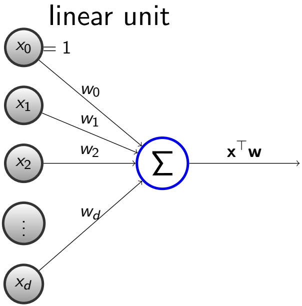
- Consider a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbf{R}^{d+1} \times \{-1, +1\}$.
- While the perceptron aims to minimize the classification error

$$\min_{\mathbf{w}} \sum_i I(y_i \neq \mathrm{sgn}(\mathbf{w}^\top \mathbf{x}_i)),$$

  Adaline aims to minimize

$$\min_{\mathbf{w}} \sum_i (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

- For classification, Adaline outputs $\mathrm{sgn}(\mathbf{w}^\top \mathbf{x})$, which is the same as the perceptron.



$$\min_{\mathbf{w}} \sum_i (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

Adaline essentially replaces the linear threshold unit by a linear unit, and minimizes the sum of squared error during learning.

Update rule

- Each time Adaline sees an example $(\mathbf{x}, y)$ it updates current $\mathbf{w}$ to

$$\mathbf{w}' = \mathbf{w} + \Delta\mathbf{w} = \mathbf{w} + \eta(y - \mathbf{w}^\top\mathbf{x})\mathbf{x}.$$

- Usually the example is randomly chosen from the training set.
- This can be seen as a way to reduce the error on $(\mathbf{x}, y)$ as follows.
  - The first order approximation of the error $f(\mathbf{w}) = (\mathbf{w}^\top\mathbf{x} - y)^2$ is

$$\begin{aligned} f(\mathbf{w} + \Delta\mathbf{w}) &\approx f(\mathbf{w}) + \nabla f(\mathbf{w})^\top \Delta\mathbf{w} \\ &= f(\mathbf{w}) + (-2(y - \mathbf{w}^\top\mathbf{x})\mathbf{x})^\top(\eta(y - \mathbf{w}^\top\mathbf{x})\mathbf{x}) \\ &= f(\mathbf{w}) - 2\eta(y - \mathbf{w}^\top\mathbf{x})^2\|\mathbf{x}\|_2^2. \end{aligned}$$

  - When $\Delta\mathbf{w}$ is small, updating $\mathbf{w}$ to $\mathbf{w} + \Delta\mathbf{w}$ decreases the error.
- In fact, $\Delta\mathbf{w} = -\eta\nabla f(\mathbf{w})$, and Adaline is a special case of stochastic gradient descent (more on this later in the course).

When to stop

- In practice, we can stop the algorithm after a fixed number of iterations.
- Alternatively, we can monitor the losses on the chosen examples over last few iterations, and stop when we don't see much progress.

Convergence

- We need to choose a suitable value of the learning rate η.
  - If η is too small, we cannot reduce the error by much.
  - If η is too large, we are not guaranteed to reduce the error.
- With a suitable η, the algorithm will eventually find a $\mathbf{w}$ that minimizes the sum of squared error, if we run the algorithm forever.
- If two input dimensions are highly correlated, the algorithm may convergence very slowly.

Mini-batch and batch versions

- The mini-batch version of Adaline uses the average of the correction computed over a small random subset S of examples, that is

$$\Delta \mathbf{w} = \frac{1}{|S|} \sum_{(\mathbf{x}, y) \in S} \eta (y - \mathbf{w}^\top \mathbf{x}) \mathbf{x}.$$

- In the batch version, S is the whole training set, and this can be computationally expensive for large datasets.
- Using a suitable mini-batch size can accelerate convergence, and at the same time maintaining efficiency.

Classification performance guarantee

- While Adaline can be used to produce a classification rule for linearly non-separable data, it may not find a separating hyperplane even when there is one.
- In fact, it may give an error arbitrary close to 0.5 when there is a separating hyperplane — in practice, this is usually not that bad.

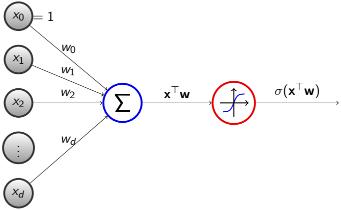
Adaline vs. Perceptron

- Perceptron and Adaline minimize different error functions, but both are error correction rules that tries to minimize the error on a chosen example.
- Perceptron is guaranteed to find a separating hyperplane when there is one, but Adaline may not.
- Perceptron never converges on non-linearly separable data, but Adaline generally converges.

Logistic Approximation

- Besides using the quadratic loss as a surrogate loss for the $0/1$ loss, there are other surrogate losses.
- The logistic approximation computes a conditional distribution $p(y = 1 \mid \mathbf{x}, \mathbf{w})$ and aims to minimize the log-loss (equivalently, maximize the log-likelihood) of the data.

- Consider a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbf{R}^{d+1} \times \{0,1\}$ (note that we are using $\{0,1\}$ instead of $\{-1,1\}$ to encode the labels).
- We use a sigmoid unit as shown below, where $\sigma(u) = \frac{1}{1+e^{-u}}$ squashes $u \in (-\infty, +\infty)$ to be in the range $[0,1]$



- $\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}} = \frac{e^{\mathbf{w}^\top \mathbf{x}}}{1+e^{\mathbf{w}^\top \mathbf{x}}}$ is the probability that $\mathbf{x}$ is positive, and thus $\frac{1}{1+e^{\mathbf{w}^\top \mathbf{x}}}$ is the probability that $\mathbf{x}$ is negative.
- We can write down the class distribution in a compact form as

$$p(y \mid \mathbf{x}, \mathbf{w}) = \frac{e^{y\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}}.$$

- The objective is to minimize the log-loss

$$\min_{\mathbf{w}} \sum_i - \ln \frac{e^{y_i \mathbf{w}^\top \mathbf{x}_i}}{1 + e^{\mathbf{w}^\top \mathbf{x}_i}}$$

- This is just a binary logistic regression model.

Update rule

- Each time we see an example $(\mathbf{x}, y)$, we update current $\mathbf{w}$ to

$$\mathbf{w}' = \mathbf{w} + \Delta\mathbf{w} = \mathbf{w} + \eta(y - \sigma(\mathbf{w}^\top \mathbf{x}))\mathbf{x}.$$

  This has the same form as the perceptron, except that we have a different scaling factor for $\mathbf{x}$.
- Usually the example is randomly chosen from the training set.

Perceptron, Adaline, Logistic Regression

- Each time we see a random example $(\mathbf{x}, y)$, we update current $\mathbf{w}$ to

$$\begin{aligned}
&\text{(Perceptron)} && \mathbf{w}' = \mathbf{w} + \eta(y - \operatorname{sgn}(\mathbf{w}^\top \mathbf{x}))\mathbf{x}, \\
&\text{(Adaline)} && \mathbf{w}' = \mathbf{w} + \eta(y - \mathbf{w}^\top \mathbf{x})\mathbf{x}, \\
&\text{(Logistic)} && \mathbf{w}' = \mathbf{w} + \eta(y - \sigma(\mathbf{w}^\top \mathbf{x}))\mathbf{x}.
\end{aligned}$$

- All three algorithms adjust current $\mathbf{w}$ by an amount of $\eta c \mathbf{w}$, where c is a correction factor specific to each algorithm.

Demo

A linearly separable dataset

```
import numpy as np
from scipy.special import expit

# generate a random dataset consisting of 200 examples
n = 200
d = 10
X = np.random.rand(n, d) - 0.5
beta = np.ones(d)
Y = np.sign(X @ beta)
```

- The true classifier is $f(\mathbf{x}; \beta) = \text{sgn}(\mathbf{w}^\top \mathbf{x})$, with each component of β randomly drawn from $[0, 1]$.
- Each $\mathbf{x} \in \mathbf{R}^{10}$ has its coordinate randomly drawn from $[-0.5, 0.5]$.

6 lines implementation

```
w = np.zeros(d)
for s in range(1000):
  i = np.random.randint(n)
  if Y[i] != np.sign(w @ X[i,]):
    w += 0.2 * Y[i] * X[i,]
print('Perceptron error:', sum(np.sign(X @ w) != Y)/n)

w = np.zeros(d)
for s in range(1000):
  i = np.random.randint(n)
  w += 0.2 * (Y[i] - w @ X[i,]) * X[i,]
print('Adaline error:', sum(np.sign(X @ w) != Y)/n)

Y[Y == -1] = 0
w = np.zeros(d)
for s in range(1000):
  i = np.random.randint(n)
  w += 0.2 * X[i,] * (Y[i] - expit(w @ X[i,]))
print('Logistic regression error:', sum(np.abs(np.around(expit(X @
   w)) != Y))/*n)
```

We only need 5 or 6 lines for learning and testing for each algorithm.

Sample output

```
Perceptron error: 0.075
Adaline error: 0.06
Logistic regression error: 0.025
```

Note that for the perceptron, we can get a zero error by always choosing a misclassified example, instead of randomly choosing an example.

Your turn

Which of the following statement is correct? (Multiple choice)

(a) There is a well-known efficient algorithm to find a perceptron with minimum classification error on any dataset.

(b) Adaline solves a classification problem by solving a regression problem.

(c) Adaline always finds a linear decision boundary with minimum classification error.

What You Need to Know

- Two approaches to train linear classifiers for non-linearly separable data
- Adaline (using quadratic loss as a surrogate loss for 0/1 loss)
- Logistic approximation (using log-loss as a surrogate loss for 0/1 loss)