Neural Networks

Professor Ameet Talwalkar

Outline

1. Administration
2. Review of last lecture
3. Neural Networks
4. Summary

Grade Policy and Final Exam

Upcoming Schedule

- Today: HW5 due, HW6 released
- Wednesday (3/8): Last day of class
- Next Monday (3/13): No class – I will hold office hours in my office (BH 4531F)
- Next Wednesday (3/15): Final Exam, HW6 due

Final Exam

- Cumulative but with more emphasis on new material

Outline

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where β_t is the contribution of each weak classifier

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where β_t is the contribution of each weak classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

$$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  2. Compute contribution for this classifier

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

     $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing
  $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$
  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points
  $$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

     $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

     $$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     and normalize them such that $\sum_n w_{t+1}(n) = 1$

Adaboost Algorithm

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
- For $t = 1$ to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ using current weights $w_t(n)$, by minimizing

     $$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] \quad \text{(the weighted classification error)}$$

  2. Compute contribution for this classifier: $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights on training points

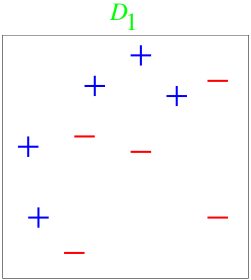
     $$w_{t+1}(n) \propto w_t(n)e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$

     and normalize them such that $\sum_n w_{t+1}(n) = 1$
- Output the final classifier

$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

Example

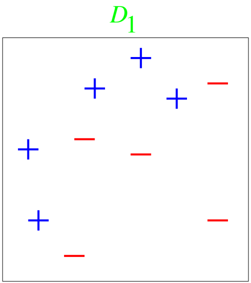
10 data points and 2 features



- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")

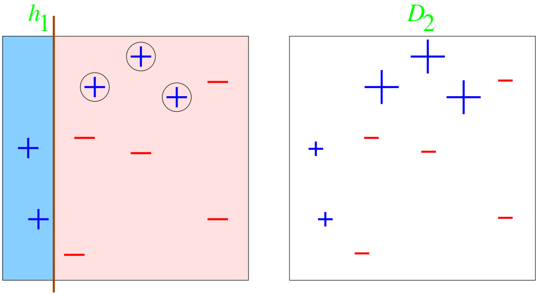
Example

10 data points and 2 features

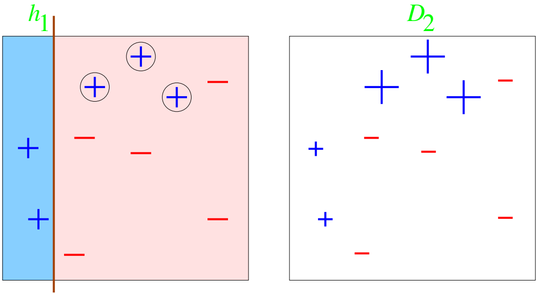


- The data points are clearly not linear separable
- In the beginning, all data points have equal weights (the size of the data markers "+" or "-")
- Base classifier $h(\cdot)$: horizontal or vertical lines ('decision stumps')
  - Depth-1 decision trees, i.e., classify data based on a single attribute

Round 1: $t = 1$

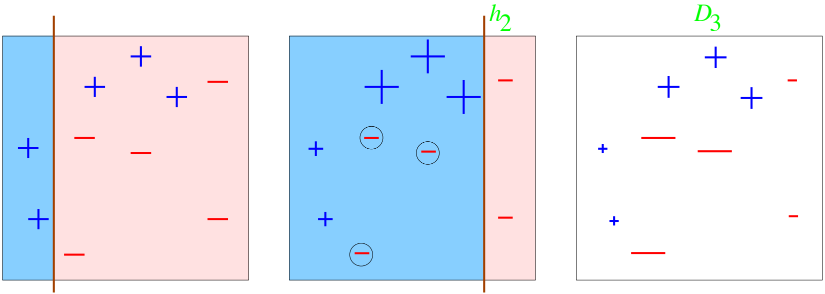


Round 1: $t = 1$

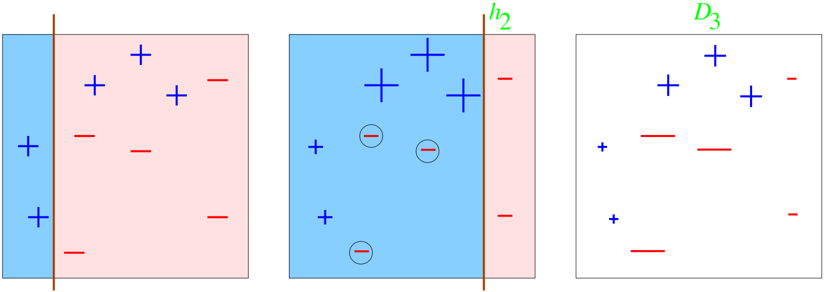


- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$

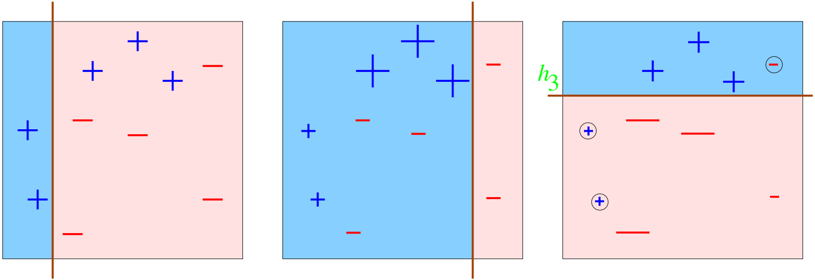


Round 2: $t = 2$

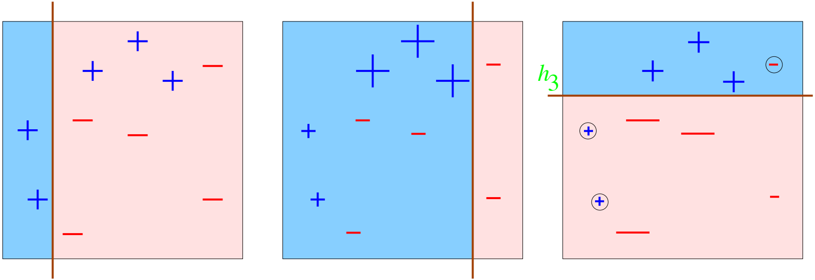


- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  Note that $\epsilon_2 \neq 0.3$ as those 3 data points have weights less than $1/10$
- 3 misclassified data points get larger weights
- Data points classified correctly in both rounds have small weights

Round 3: $t = 3$

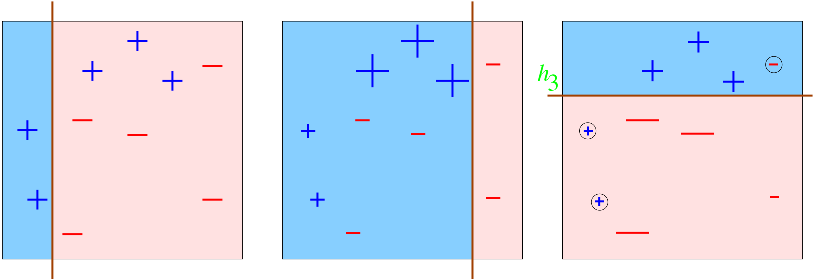


Round 3: $t = 3$



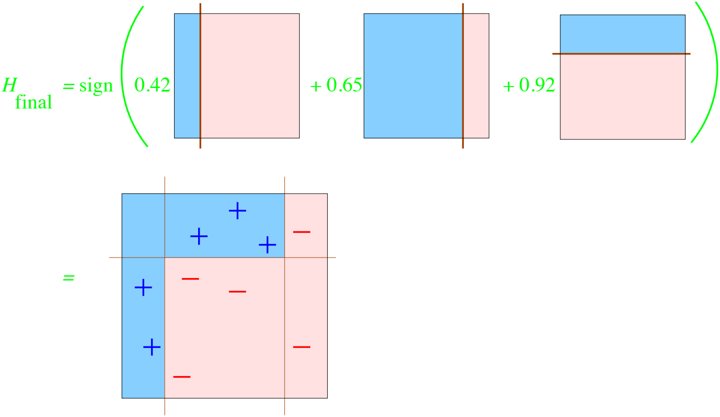
- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?

Round 3: $t = 3$



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?
  - Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction

Final classifier: combining 3 classifiers



- All data points are now classified correctly!

Derivation of the AdaBoost

Minimize exponential loss

$$\ell^{\text{EXP}}(h(\boldsymbol{x}), y) = e^{-yf(\boldsymbol{x})}$$

Greedily (sequentially) find the best classifier to optimize the loss

A classifier $f_{t-1}(\boldsymbol{x})$ is improved by adding a new classifier $h_t(\boldsymbol{x})$

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\ &= \arg\min_{(h_t(\boldsymbol{x}), \beta_t)} \sum_n e^{-y_n [f_{t-1}(\boldsymbol{x}_n) + \beta_t h_t(\boldsymbol{x}_n)]}\end{aligned}$$

Nonlinear basis learned by boosting

Two-stage process

- Get $\textsc{sign}[f_1(\boldsymbol{x})]$, $\textsc{sign}[f_2(\boldsymbol{x})]$,$\cdots$,
- Combine into a linear classification model

$$y = \textsc{sign}\left\{\sum_t \beta_t \textsc{sign}[f_t(\boldsymbol{x})]\right\}$$

Equivalently, each stage learns a nonlinear basis $\phi_t(\boldsymbol{x}) = \textsc{sign}[f_t(\boldsymbol{x})]$.

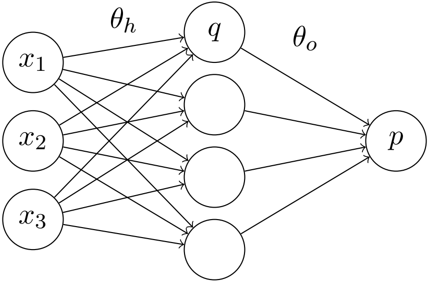
One thought is then, why not learning the basis functions and the classifier at the same time?

Outline

Derivation of backpropagation

- Calculate the feed-forward signals $\boldsymbol{a}^{(l)}$ from the input to the output
- Calculate output error based on the predictions $\boldsymbol{a}^{(L)}$ and the label
- Backpropagate the error by computing the $\boldsymbol{\delta}^{(l)}$ values
- Calculating the gradients $\frac{\partial J}{\partial \theta_{ij}}$ via $\boldsymbol{a}^{(l)}$ and $\boldsymbol{\delta}^{(l)}$
- Updates the parameters via $\theta_{ij} \leftarrow \theta_{ij} - \eta \frac{\partial J}{\partial \theta_{ij}}$

Illustrative example



- p, q: outputs at the indicated nodes
- θ_h: edge weight from x_1 to hidden node q
- θ_o: edge weight from hidden node q to output node p
- b_q: bias associated with node q
- z_q: input to node q, i.e., $z_q = b_q + \theta_h x_1 + \ldots$
- z_p: input to node p, i.e., $z_p = b_p + \theta_o q + \ldots$
- g: activation function (e.g., sigmoid)
- t: target value for output node

Illustrative example (cont'd)

- Assume cross entropy loss between target t and network output p

$$E = t \log p + (1-t)(1-\log p)$$

- Gradients for the output layer (assuming t = 1)

$$\begin{aligned}
\frac{\partial E}{\partial \theta_o} &= \frac{\partial E}{\partial g}\frac{\partial g}{\partial z_q}\frac{\partial z_q}{\partial \theta_o} \\
&= \frac{1}{p}\frac{\partial}{\partial \theta_o} p = \frac{1}{p}\frac{\partial}{\partial \theta_o}\sigma(z_p) \\
&= \frac{1}{p} p(1-p)\frac{\partial}{\partial \theta_o} z_p = (1-p)\frac{\partial}{\partial \theta_o}(b_p + \theta_o q + \ldots) \\
&= (1-p)q
\end{aligned}$$

- Gradients for hidden layer

$$\frac{\partial E}{\partial \theta_h} = \frac{\partial E}{\partial g}\frac{\partial g}{\partial z_q}\frac{\partial z_q}{\partial \theta_h}$$

Outline

Summary of the course so far: a short list of important concepts

Supervised learning has been our focus

- Setup: given a training dataset $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, we learn a function $h(\boldsymbol{x})$ to predict $\boldsymbol{x}$'s true value y (i.e., regression or classification)
- Linear vs. nonlinear features
  1. Linear: $h(\boldsymbol{x})$ depends on $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$
  2. Nonlinear: $h(\boldsymbol{x})$ depends on $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$, where ϕ is either explicit or depends on a kernel function $k(\boldsymbol{x}_m, \boldsymbol{x}_n) = \boldsymbol{\phi}(\boldsymbol{x}_m)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n)$,
- Loss function
  1. Squared loss: least square for regression (minimizing residual sum of errors)
  2. Logistic loss: logistic regression
  3. Exponential loss: AdaBoost
  4. Margin-based loss: support vector machines
- Principles of estimation
  1. Point estimate: maximum likelihood, regularized likelihood

cont'd

- Optimization
  1. Methods: gradient descent, Newton method
  2. Convex optimization: global optimum vs. local optimum
  3. Lagrange duality: primal and dual formulation
- Learning theory
  1. Difference between training error and generalization error
  2. Overfitting, bias and variance tradeoff
  3. Regularization: various regularized models