18-661 Introduction to Machine Learning

Neural Networks-I

Spring 2019

Prof. Gauri Joshi
Announcements

- **Mid-semester grades are available on S3.**
  - These grades are based on the score in the mid-term and first 3 HWs.
  - This weights the homework much more than your final grade will.
  - There is still plenty of time to improve!
  - Come talk to us if you are concerned about your grade.

- Homework 5 due date extended to Saturday by 11:59 pm.

- Homework 6 will be released later this week.

- Homework 7 will be a mini-project in TensorFlow/Pytorch. More details coming soon!

- My office hours today are canceled.

- There will be recitation this Friday.
1. Review of Ensemble Methods

2. Review of Boosting Methods: AdaBoost

3. Neural networks: Motivation

4. The Perceptron Algorithm

5. General Neural Network Architectures
Review of Ensemble Methods
Ensemble Methods

• Instead of learning a single (weak) classifier, learn many weak classifiers, preferably those that are good at different parts of the input spaces

• **Predicted Class:** (Weighted) Average or Majority of output of the classifiers

• Strength in Diversity!

\[
H: X \rightarrow Y \ (-1, 1) \\
H(X) = h_1(X) + h_2(X) \\
H(X) = \text{sign}(\sum_{t}^{\alpha_t h_t(X)})
\]
Next, we will learn the following ensemble methods:

- Bagging or Bootstrap Aggregation
- Random Forests
- AdaBoost

\[
H: X \rightarrow Y \{-1, 1\} \\
H(X) = h_1(X) + h_2(X) \\
H(X) = \text{sign}(\sum_{t} \alpha_t h_t(X))
\]
Bagging or Bootstrap Aggregating

To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

**Bagging Trees (Training Phase):**

- For $b = 1, 3, \ldots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $D$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

**Bagging Trees (Test Phase):**

- For a test unlabeled example $x$
To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

**Bagging Trees (Training Phase):**

- For $b = 1, 3, \cdots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $\mathcal{D}$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

**Bagging Trees (Test Phase):**

- For a test unlabeled example $\mathbf{x}$
- Find the decision from each of the $B$ trees
Bagging or Bootstrap Aggregating

To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

Bagging Trees (Training Phase):

- For $b = 1, 3, \cdots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $D$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

Bagging Trees (Test Phase):

- For a test unlabeled example $x$
  - Find the decision from each of the $B$ trees
  - Assign the majority label as the label for $x$
Bagging: Example

- We get different splits and thresholds for different $b$
- Predict the label assigned by majority of the $B$ trees
- Reduces variance without increasing bias, thus avoiding overfitting
Random Forests

- **Limitation of Bagging:** If one or more features are very informative, they will be selected by almost every tree in the bag, reducing the diversity.

- **Key Idea on Random Forests:** Reduces correlation between trees in the bag without increasing variance too much.

- Same as bagging in terms of sampling training data.

- Before each split, select \( m \leq d \) features at random as candidates for splitting \( m \sim \sqrt{d} \).

- Take majority vote of \( B \) such trees.
Increasing $m$, the number of splitting candidates chosen increases the correlation among the trees in the bag.
Increasing $m$ decreases the bias but increases variance in the ensemble classifier.
Random Forests
1. Review of Ensemble Methods

2. Review of Boosting Methods: AdaBoost

3. Neural networks: Motivation

4. The Perceptron Algorithm

5. General Neural Network Architectures
Review of Boosting Methods: AdaBoost
Limitations of Bagging and Random Forests

- Bagging: Significant correlation between trees that are learnt on different training datasets.
- Random Forests try to resolve this by doing "feature bagging" but some correlation still remains.
- All $B$ trees are given the same weight when taking the average.
Limitations of Bagging and Random Forests

- Bagging: Significant correlation between trees that are learnt on different training datasets
- Random Forests try to resolve this by doing "feature bagging" but some correlation still remains
- All $B$ trees are given the same weight when taking the average

**Boosting methods:** Force classifiers to learn on different parts of the feature space, and take their weighted average
High-level idea: combine a lot of classifiers

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

\[
h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]
\]
**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 $\beta_t$ is the contribution of each weak classifier

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

**Adaboost Algorithm: Assumptions**

- Black-box routine for constructing a sequence of weak classifiers $h(\cdot)$
- The classifier needs to know how to optimize on *weighted* samples
The Adaboost Algorithm

- Given: $N$ samples $\{x_n, y_n\}$, 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 $h_t(x)$ using current weights $w_t(n)$, by minimizing

$$
\epsilon_t = \sum_n w_t(n)[y_n \neq h_t(x_n)]
$$

- Output the final classifier

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

- Given: \( N \) samples \( \{x_n, y_n\} \), 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 \( h_t(x) \) using current weights \( w_t(n) \), by minimizing
     \[
     \epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
     \]
  2. Compute contribution for this classifier
The Adaboost Algorithm

- Given: \( N \) samples \( \{x_n, y_n\} \), 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 \( h_t(x) \) using current weights \( w_t(n) \), by minimizing
     \[
     \epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
     \]
  2. Compute contribution for this classifier: \( \beta_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t} \)
  3. Update weights on training points
     \[
     w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(x_n)}
     \]
     and normalize them such that \( \sum_n w_{t+1}(n) = 1 \)
- Output the final classifier
  \[
  h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]
  \]
Example

10 data points and 2 features

- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers “+” or “-”)
10 data points and 2 features

- The data points are clearly not linearly 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: $t = 1$

$h_1$ with 3 misclassified data points:

$\epsilon_1 = 0$.

$\beta_1 = 0.42$.

Weights recomputed; the 3 misclassified data points receive larger weights.
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: \( t = 2 \)

- There are 3 misclassified data points (with circles).
  - \( \epsilon_2^2 = 0 \to \beta_2^2 = 0 \).
  - Note that \( \epsilon_2^2 \neq 0 \).

- Data points classified correctly in both rounds have small weights.

- 3 misclassified data points get larger weights.
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: $t = 3$

- $h_3$ misclassified (with circles): $\epsilon = 0.14 \rightarrow \beta = 0.92$.
- 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.
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

\[ H_{\text{final}} = \text{sign} \left( \begin{array}{c} 0.42 \\ + 0.65 \\ + 0.92 \end{array} \right) \]

- All data points are now classified correctly!
Why AdaBoost works?

It minimizes a loss function related to classification error.

**Classification loss**

- Suppose we want to have a classifier

\[ h(x) = \text{sign}[f(x)] = \begin{cases} 1 & \text{if } f(x) > 0 \\ -1 & \text{if } f(x) < 0 \end{cases} \]

- One seemingly natural loss function is 0-1 loss:

\[ \ell(h(x), y) = \begin{cases} 0 & \text{if } yf(x) > 0 \\ 1 & \text{if } yf(x) < 0 \end{cases} \]

Namely, the function \( f(x) \) and the target label \( y \) should have the same sign to avoid a loss of 1.
Surrogate loss

0 – 1 loss function $\ell(h(x), y)$ is non-convex and difficult to optimize.

We can instead use a surrogate loss – what are examples?
Surrogate loss

0 – 1 loss function $\ell(h(x), y)$ is non-convex and difficult to optimize. We can instead use a surrogate loss – what are examples?

**Exponential Loss**

$$\ell^{\text{EXP}}(h(x), y) = e^{-yf(x)}$$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

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

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does $\text{sign}[f_{t-1}(\cdot)]$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

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

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does sign $[f_{t-1}(\cdot)]$

How can we ‘optimally’ choose $h_t(x)$ and combination coefficient $\beta_t$?

Adaboost greedily minimizes the exponential loss function!

$$(h^*_t(x), \beta^*_t) = \text{argmin}_{(h_t(x), \beta_t)} \sum_n e^{-y_n f(x_n)}$$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

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

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does sign $[f_{t-1}(\cdot)]$

How can we ‘optimally’ choose $h_t(x)$ and combination coefficient $\beta_t$?

Adaboost greedily minimizes the exponential loss function!

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_nf(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n[f_{t-1}(x_n)+\beta_t h_t(x_n)]}$$
Choosing the $t$-th classifier

Suppose a classifier $f_{t-1}(x)$, and want to add a weak learner $h_t(x)$

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

note: $h_t(\cdot)$ outputs $-1$ or $1$, as does sign [$f_{t-1}(\cdot)$]

How can we ‘optimally’ choose $h_t(x)$ and combination coefficient $\beta_t$?

Adaboost greedily \textit{minimizes the exponential loss function!}

\begin{align*}
(h_t^*(x), \beta_t^*) &= \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_nf(x_n)} \\
&= \arg\min_{(h_t(x), \beta_t)} \sum_n e^{-y_n[f_{t-1}(x_n) + \beta_t h_t(x_n)]} \\
&= \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n) e^{-y_n\beta_t h_t(x_n)}
\end{align*}

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(x_n)}$
We can decompose the \textit{weighted} loss function into two parts

\[
\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)} = \sum_n w_t(n) e^{\beta_t \mathbb{I}[y_n \neq h_t(x_n)]} + \sum_n w_t(n) e^{-\beta_t \mathbb{I}[y_n = h_t(x_n)]}
\]
We can decompose the *weighted* loss function into two parts

\[
\sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}
\]

\[
= \sum_n w_t(n) e^{\beta_t [y_n \neq h_t(x_n)]} + \sum_n w_t(n) e^{-\beta_t [y_n = h_t(x_n)]}
\]

\[
= \sum_n w_t(n) e^{\beta_t [y_n \neq h_t(x_n)]} + \sum_n w_t(n) e^{-\beta_t (1 - [y_n \neq h_t(x_n)])}
\]
We can decompose the \textit{weighted} loss function into two parts

\[
\sum_{n} w_t(n) e^{-y_n \beta_t h_t(x_n)}
\]

\[
= \sum_{n} w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_{n} w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(x_n)]
\]

\[
= \sum_{n} w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(x_n)] + \sum_{n} w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(x_n)])
\]

\[
= (e^{\beta_t} - e^{-\beta_t}) \sum_{n} w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_{n} w_t(n)
\]

We have used the following properties to derive the above:

- $y_n h_t(x_n)$ is either 1 or -1 as $h_t(x_n)$ is the output of a binary classifier.
- The indicator function $\mathbb{I}[y_n = h_t(x_n)]$ is either 0 or 1, so it equals $1 - \mathbb{I}[y_n \neq h_t(x_n)]$. 
Finding the optimal weak learner

Summary

\[(h_t^*(x), \beta_t^*) = \arg\min_{h_t(x), \beta_t} \sum_n w_t(n)e^{-y_n \beta_t h_t(x_n)}\]

\[= \arg\min_{h_t(x), \beta_t} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(x_n)]\]

\[+ e^{-\beta_t} \sum_n w_t(n)\]

What term(s) must we optimize to choose \(h_t(x_n)\)?
Finding the optimal weak learner

Summary

\[(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}\]

\[= \arg\min_{(h_t(x), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) [y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)\]

What term(s) must we optimize to choose \(h_t(x_n)\)?

\[h^*_t(x) = \arg\min_{h_t(x)} \epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)]\]
Finding the optimal weak learner

Summary

\[
(h^*_t(x), \beta^*_t) = \arg\min_{h_t(x), \beta_t} \sum_n w_t(n)e^{-y_n\beta_t h_t(x_n)}
\]

\[
= \arg\min_{h_t(x), \beta_t} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n)[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)
\]

What term(s) must we optimize to choose \(h_t(x_n)\)?

\[
h^*_t(x) = \arg\min_{h_t(x)} \epsilon_t = \sum_n w_t(n)[y_n \neq h_t(x_n)]
\]

Minimize weighted classification error as noted in step 1 of Adaboost!
How to choose $\beta_t$?

Summary

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) [y_n \neq h_t(x_n)]$$

$$+ e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize?
How to choose $\beta_t$?

Summary

$$(h_t^*(x), \beta_t^*) = \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{1}[y_n \neq h_t(x_n)]$$

$$+ e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to $\beta_t$!
How to choose $\beta_t$?

Summary

$$(h^*_t(x), \beta^*_t) = \arg\min_{(h_t(x), \beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(x_n)}$$

$$= \arg\min_{(h_t(x), \beta_t)} (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(x_n)] + e^{-\beta_t} \sum_n w_t(n)$$

What term(s) must we optimize?

We need to minimize the entire objective function with respect to $\beta_t$!

We can do this by taking derivative with respect to $\beta_t$, setting to zero, and solving for $\beta_t$. After some calculation and using $\sum_n w_t(n) = 1$, we find:

$$\beta^*_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely step 2 of Adaboost! (Exercise – verify the solution)
Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t^* h_t^*(x) \]
Updating the weights

Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t^* h_t^*(x) \]

We then need to compute the weights for the above classifier as:

\[ w_{t+1}(n) = e^{-y_n f(x_n)} = e^{-y_n[f_{t-1}(x) + \beta_t^* h_t^*(x_n)]} \]
Updating the weights

Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta^*_t h^*_t(x) \]

We then need to compute the weights for the above classifier as:

\[ w_{t+1}(n) = e^{-y_nf(x_n)} = e^{-y_n[f_{t-1}(x) + \beta^*_t h^*_t(x_n)]} \]

\[ = w_t(n)e^{-y_n\beta^*_t h^*_t(x_n)} \]
Once we find the optimal weak learner we can update our classifier:

\[ f(x) = f_{t-1}(x) + \beta_t^* h_t^*(x) \]

We then need to compute the weights for the above classifier as:

\[
w_{t+1}(n) = e^{-y_n f(x_n)} = e^{-y_n [f_{t-1}(x) + \beta_t^* h_t^*(x_n)]}
\]

\[
= w_t(n) e^{-y_n \beta_t^* h_t^*(x_n)} = \begin{cases} 
  w_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(x_n) \\
  w_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(x_n) 
\end{cases}
\]

**Intuition** Misclassified data points will get their weights increased, while correctly classified data points will get their weight decreased.
Note that the AdaBoost algorithm itself never specifies how we would get $h^*_t(x)$ as long as it minimizes the weighted classification error

$$
\epsilon_t = \sum_n w_t(n) \mathbb{1}[y_n \neq h^*_t(x_n)]
$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any type of classifier
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data

- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness

  - Two common ensemble methods for decision trees
    - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees
    - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better

- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers
  - Increase the weight hard training points
  - Take weighted average of the outputs of the resulting classifiers

- Decision trees are commonly used as base classifiers in AdaBoost.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness
- Two common ensemble methods for decision trees
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
  - Increase the weight of hard training points.
  - Take weighted average of the outputs of the resulting classifiers.

Decision trees are commonly used as base classifiers in AdaBoost.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.

- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.

- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.

- Boosting Methods: AdaBoost
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
  - Increase the weight hard training points.
Mini-Summary

• Decision Trees have high variance and thus can change drastically with small changes in training data

• Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness

• Two common ensemble methods for decision trees
  • Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees
  • Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better

• Boosting Methods: AdaBoost
  • Sequentially add weak classifiers
  • Increase the weight hard training points
  • Take weighted average of the outputs of the resulting classifiers
Mini-Summary

- Decision Trees have high variance and thus can change drastically with small changes in training data.
- Ensemble Methods combine outputs of many weak classifiers in order to provide more robustness.
- Two common ensemble methods for decision trees:
  - Bagging – Train $B$ trees on random subsets of training data. Can lead to highly correlated trees.
  - Random Forests – In addition to bagging, randomly sample $m$ candidate features at each split. De-correlates the trees better.
- Boosting Methods: AdaBoost
  - Sequentially add weak classifiers.
  - Increase the weight hard training points.
  - Take weighted average of the outputs of the resulting classifiers.
- Decision trees are commonly used as base classifiers in AdaBoost.
Neural networks: Motivation
Logistic Regression: How to Handle Complex Boundaries?

- This data is not linear separable
- Use non-linear basis functions to add more features
Logistic Regression: How to Handle Complex Boundaries?

- This data is not linear separable
- Use non-linear basis functions to add more features
Adding polynomial features

- New feature vector is \( \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2] \)
- \( \Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2) \)
- If \( \mathbf{w} = [-1, 0, 0, 1, 1] \), the boundary is \(-1 + x_1^2 + x_2^2 = 0\)
  - If \(-1 + x_1^2 + x_2^2 \geq 0\) declare spam
  - If \(-1 + x_1^2 + x_2^2 < 0\) declare ham
Adding polynomial features

• New feature vector is \( \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2] \)
• \( \Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2) \)
• If \( \mathbf{w} = [-1, 0, 0, 1, 1] \), the boundary is \(-1 + x_1^2 + x_2^2 = 0\)
  • If \(-1 + x_1^2 + x_2^2 \geq 0\) declare spam
  • If \(-1 + x_1^2 + x_2^2 < 0\) declare ham

\[-1 + x_1^2 + x_2^2 = 0\]
But what if we had a large number of features?

Each feature $x_i$ is one pixel in an $100 \times 100$ input image

- Adding polynomial features would result in an enormous $\phi(x)$
But what if we had a large number of features?

Each feature $x_i$ is one pixel in an $100 \times 100$ input image

- Adding polynomial features would result in an enormous $\phi(x)$
- Can we somehow only retain the important features?
But what if we had a large number of features?

Each feature $x_i$ is one pixel in an $100 \times 100$ input image.

- Adding polynomial features would result in an enormous $\phi(x)$
- Can we somehow only retain the important features?
- We will need to carefully hand-pick them, which can be hard and tedious
But what if we had a large number of features?

Each feature \( x_i \) is one pixel in an 100 \( \times \) 100 input image

- Adding polynomial features would result in an enormous \( \phi(x) \)
- Can we somehow only retain the important features?
- We will need to carefully hand-pick them, which can be hard and tedious
- Neural networks automate this for us!
Neural Network Compress the Set of Features

- Start with feature vector $\mathbf{x}$ containing all pixels in the image
Neural Network Compress the Set of Features

- Start with feature vector $\mathbf{x}$ containing all pixels in the image
- Layer 1: distill the edges of the image
- Layer 2: distill triangles, circles, etc.
- Layer 3: recognize pointy ears, fur style etc.
- Layer 4: performs logistic regression on the features in layer 3

We cannot directly control what each layer learns; this depends on the training data.
Neural Network Compress the Set of Features

- Start with feature vector $\mathbf{x}$ containing all pixels in the image
- Layer 1: distill the edges of the image
- Layer 2: distill triangles, circles, etc.
- Layer 3: recognize pointy ears, fur style, etc.
- Layer 4: performs logistic regression on the features in layer 3

We cannot directly control what each layer learns; this depends on the training data.
Neural Network Compress the Set of Features

- Start with feature vector $\mathbf{x}$ containing all pixels in the image
- Layer 1: distill the edges of the image
- Layer 2: distill triangles, circles, etc.
- Layer 3: recognize pointy ears, fur style etc.
Neural Network Compress the Set of Features

- Start with feature vector $x$ containing all pixels in the image
- Layer 1: distill the edges of the image
- Layer 2: distill triangles, circles, etc.
- Layer 3: recognize pointy ears, fur style etc.
- Layer 4: performs logistic regression on the features in layer 3
Neural Network Compress the Set of Features

- Start with feature vector \( \mathbf{x} \) containing all pixels in the image
- Layer 1: distill the edges of the image
- Layer 2: distill triangles, circles, etc.
- Layer 3: recognize pointy ears, fur style etc.
- Layer 4: performs logistic regression on the features in layer 3
Neural Network Compress the Set of Features

- Start with feature vector \( \mathbf{x} \) containing all pixels in the image
- Layer 1: distill the edges of the image
- Layer 2: distill triangles, circles, etc.
- Layer 3: recognize pointy ears, fur style etc.
- Layer 4: performs logistic regression on the features in layer 3

We cannot directly control what each layer learns; this depends on the training data
Inspiration from Biology: How does our brain work?

Each feature $x_i$ is one pixel in an $100 \times 100$ input image

- Humans easily perform such complex image or speech recognition tasks
- We cannot exactly describe a set of rules by which we distinguish cats vs. dogs, but we almost always know the correct answers when a new image is presented to us
- How do our brains learn these complex tasks?
Neurons in the Brain

- Each neuron is a non-linear computing unit
- It collects input signals from neighboring neurons
- Output of its computation is transmitted through the axon
- Other neurons use this output as the input signal

An average human brain has \( \sim 100 \text{ billion neurons!} \)
Plasticity of Neurons in the Brain

- Different parts of our brain are specialized for different tasks
- But they can be re-wired to adapt to different inputs
- The auditory cortex can be taught to see and the visual cortex can be taught to listen!
- Based on these observations, scientists concluded that all the neurons are structurally homogeneous
Based on the biological insights, a mathematical model for an ‘artificial’ neuron was developed

- Each input $x_i$ is multiplied by weight $w_i$
- Add a $+1$ input neuron which is multiplied by the bias $b$
- Apply a non-linear function $g$ to the weighted combination of the inputs, $\mathbf{w}^T \mathbf{x} + b$
- Different candidates for $g$: heaviside function, sigmoid, tanh, rectified linear unit, etc.
1. Review of Ensemble Methods

2. Review of Boosting Methods: AdaBoost

3. Neural networks: Motivation

4. The Perceptron Algorithm

5. General Neural Network Architectures
The Perceptron Algorithm
The perception is a single-unit neural network with the heavyside activation function or $\text{sign}(x)$.
• The perception is a single-unit neural network with the heavyside activation function or $\text{sign}(x)$.
• It considers a linear binary classification problem to distinguish between two classes $\{-1, +1\}$.

$$g(w^T x + b), \text{Linear comb. of features}$$
Perceptron: Rosenblatt (1957)

- The perception is a single-unit neural network with the heavyside activation function or $\text{sign}(x)$.
- It considers a linear binary classification problem to distinguish between two classes $\{-1, +1\}$.

![Perceptron diagram]

- Assign label $\text{sign}(\mathbf{w}^T \mathbf{x} + b)$ to a new sample.
The perceptron is a single-unit neural network with the heaviside activation function or \( \text{sign}(x) \). It considers a linear binary classification problem to distinguish between two classes \([-1, +1]\).

\[
g(w^T x + b) = \begin{cases} 
1 & \text{if } w^T x + b > 0 \\
0 & \text{otherwise}
\end{cases}
\]

- Assign label \( \text{sign}(w^T x + b) \) to a new sample
- Notation change: Merge \( b \) into the vector \( w \) and append 1 to the vector \( x \)
How to learn the weights $w$?

The objective is to learn $w$ that minimizes the number of errors on the training dataset. That is, minimize

$$
\varepsilon = \sum_n \mathbb{I}[y_n \neq \text{sign}(w^\top x_n)]
$$

**Algorithm**: For a randomly chosen data point $(x_n, y_n)$ make small changes to $w$ so that

$$
y_n = \text{sign}(w^\top x_n)
$$
How to learn the weights $w$?

The objective is to learn $w$ that minimizes the number of errors on the training dataset. That is, minimize

$$\varepsilon = \sum_n \mathbb{1}[y_n \neq \text{sign}(w^\top x_n)]$$

**Algorithm**: For a randomly chosen data point $(x_n, y_n)$ make small changes to $w$ so that

$$y_n = \text{sign}(w^\top x_n)$$

**Two cases**

- If $y_n = \text{sign}(w^\top x_n)$, do nothing.
- If $y_n \neq \text{sign}(w^\top x_n)$,

$$w^{\text{NEW}} \leftarrow w^{\text{OLD}} + y_n x_n$$
If \( y_n \neq \text{sign}(w^\top x_n) \), then

\[ y_n(w^\top x_n) < 0 \]
Why would it work?

If $y_n \neq \text{sign}(w^\top x_n)$, then

$$y_n(w^\top x_n) < 0$$

What would happen if we change to new $w^{\text{NEW}} = w + y_n x_n$?

$$y_n[(w + y_n x_n)^\top x_n] = y_n w^\top x_n + y_n^2 x_n^\top x_n$$
Why would it work?

If \( y_n \neq \text{sign}(w^\top x_n) \), then

\[
y_n(w^\top x_n) < 0
\]

What would happen if we change to new \( w^{\text{NEW}} = w + y_nx_n \)?

\[
y_n[(w + y_nx_n)^\top x_n] = y_n w^\top x_n + y_n^2 x_n^\top x_n
\]

We are adding a positive number, so it is possible that

\[
y_n(w^{\text{NEW}}^\top x_n) > 0
\]

i.e., we are more likely to classify correctly
Iteratively solving one case at a time

- REPEAT
- Pick a data point $x_n$ (can be a fixed order of the training instances)
- Make a prediction $y = \text{sign}(w^T x_n)$ using the current $w$. 
Iteratively solving one case at a time

- REPEAT
- Pick a data point $x_n$ (can be a fixed order of the training instances)
- Make a prediction $y = \text{sign}(\mathbf{w}^\top \mathbf{x}_n)$ using the current $\mathbf{w}$.
- If $y = y_n$, do nothing.
  Else,
  \[ \mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n. \]
- UNTIL converged.
Properties

• This is an online algorithm (works when data is arriving sequentially as a stream)
• If the training data is linearly separable, the algorithm stops in a finite number of steps.
• The parameter vector is always a linear combination of training instances (requires initialization of $w_0 = 0$).
• We don’t need to set a learning rate

The perceptron algorithm was used in old times to train $w$ by hand, without a computer.
Outline

1. Review of Ensemble Methods
2. Review of Boosting Methods: AdaBoost
3. Neural networks: Motivation
4. The Perceptron Algorithm
5. General Neural Network Architectures
General Neural Network Architectures
• Suppose $g$ is the sigmoid function $\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$

• We can find a linear decision boundary separating two classes. The output is the probability of $x$ belonging to class 1.
Binary Logistic Regression

• Suppose $g$ is the sigmoid function $\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$

• We can find a linear decision boundary separating two classes. The output is the probability of $x$ belonging to class 1.

• This is binary logistic regression, which we already know.
We can construct many common functions using just a single neuron.
Example: Logic Gates

We can construct many common functions using just a single neuron.

\[ \sigma(w^T x + b) \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
We can construct many common functions using just a single neuron:

\[
\sigma(x_1, x_2, +1) = \sigma(\mathbf{w}^T \mathbf{x} + b)
\]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

This is the OR gate.
Example: Logic Gates

We can construct many common functions using just a single neuron

\[
x_1 \quad \sigma \quad 20 \\
x_2 \quad 20 \\
+1 \quad -30 \\
\sigma \left( w^T x + b \right)
\]

This is the AND gate
Example: Logic Gates

We can construct many common functions using just a single neuron

\[ \sigma(\mathbf{w}^T \mathbf{x} + b) \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Logic Gates

We can construct many common functions using just a single neuron

\[
\sigma(\mathbf{w}^T \mathbf{x} + b)
\]

This is the AND gate
Learning weights $w$ using SGD

Cross-entropy Error Function

$$\mathcal{E}(w) = -\sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \}$$

Gradient descent for logistic regression

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function

$$w(t+1) \leftarrow w(t) - \eta \sum_n \{ \sigma(w^\top x_n) - y_n \} x_n$$
Learning weights $w$ using SGD

Cross-entropy Error Function

$$\mathcal{E}(w) = -\sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)]\}$$

Gradients

$$\frac{\partial \mathcal{E}(w)}{\partial w} = \sum_n \{\sigma(w^\top x_n) - y_n\} x_n$$

Gradient descent for logistic regression

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \sum_n \{\sigma(w^\top x_n) - y_n\} x_n$$
Can we build an XOR Gate?

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- No, because this data is not linearly separable.
- We can create a combination of other logic gates $(x_1 + x_2)(\overline{x_1} + \overline{x_2})$
- Equivalent to creating a multi-layer neural network
Multi-layer Neural Network

- $w_{ij}$: weights connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.
Multi-layer Neural Network

- $w_{ij}$: weights connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.
- $b_j$, $b_k$: bias for nodes $j$ and $k$.
- $u_j$, $u_k$: inputs to nodes $j$ and $k$ (where $u_j = b_j + \sum_i x_i w_{ij}$).

Nodes in the hidden layer

Apply $g_j$ to $u_j$
Multi-layer Neural Network

- $w_{ij}$: **weights** connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.
- $b_j$, $b_k$: **bias** for nodes $j$ and $k$.
- $u_j$, $u_k$: **inputs to nodes $j$ and $k** (where $u_j = b_j + \sum_i x_i w_{ij}$).
- $g_j$, $g_k$: **activation function** for node $j$ (applied to $u_j$) and node $k$.
- $y_j = g_j(u_j)$, $z_k = g_k(u_k)$: **output/activation** of nodes $j$ and $k$. 

Apply $g_j$ to $u_j$ 

Nodes in the hidden layer 

50
Multi-layer Neural Network

- $w_{ij}$: weights connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.
- $b_j, b_k$: bias for nodes $j$ and $k$.
- $u_j, u_k$: inputs to nodes $j$ and $k$ (where $u_j = b_j + \sum_i x_i w_{ij}$).
- $g_j, g_k$: activation function for node $j$ (applied to $u_j$) and node $k$.
- $y_j = g_j(u_j), z_k = g_k(u_k)$: output/activation of nodes $j$ and $k$.
- $t_k$: target value for node $k$ in the output layer.
Neural Networks are Very Powerful

- With enough neurons and layers we can represent very complex input-output relationships
- Can be used for regression, classification, embedding, and many other ML applications
Expressing outputs $z$ in terms of inputs $x$ is called forward-propagation. This is the operation that is performed when doing inference with a trained neural network.

Exercise: Perform forward propagation for the 1-hidden layer neural network shown above. Assume that we are using the sigmoid activation function.
• Outputs of the hidden layer are
Exercise: Forward-Propagation

- Outputs of the hidden layer are $\sigma(0.5x + 1)$ and $\sigma(x + 0.5)$
- Input to the last layer is
• Outputs of the hidden layer are $\sigma(0.5x + 1)$ and $\sigma(x + 0.5)$
• Input to the last layer is $0.5\sigma(0.5x + 1) + 0.5\sigma(x + 0.5) + 0.25$
• $z = \sigma(0.5\sigma(0.5x + 1) + 0.5\sigma(x + 0.5) + 0.25)$