Warm up

Given $x_1, \ldots, x_n \in \mathbb{R}$

Find:
- $\arg \min_z \sum_{i=1}^{n} (x_i - z)^2 = \frac{1}{n} \sum_{i=1}^{n} x_i$
- $\arg \min_z \sum_{i=1}^{n} |x_i - z| = f(z) = \text{median}(x_1, \ldots, x_n)$

$\nabla f(z) = \sum_{i=1}^{n} \nabla |x_i - z| = 0$

$\frac{\partial}{\partial z} |x_i - z| = \begin{cases} 
-1 & z < x_i \\
0 & z = x_i \\
1 & z > x_i 
\end{cases}$

$n = \text{even}$

$n/2$ $n/2$ $\cdots$ $z$ $\cdots$

$n = \text{odd}$

$\nabla f(z) = 0 \quad z \in [x_{n+1}, x_{n+1}]$ sorted

$\nabla f(z) \quad z = x_{(n+1)/2}$
Warm up

\[(\cdot)^2\]

Least squares

\[\|\cdot\|\]

Absolute error

Huber

Sigma=0.05

True f(x)

Fitted \(\bar{f}(x)\)

Data

Data

Sigma=0.05

True f(x)

Fitted \(\bar{f}(x)\)

Data

Data

Sigma=0.05

True f(x)

Fitted \(\bar{f}(x)\)

Data

Data

©2018 Kevin Jamieson
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]
Backprop

\[ L(y, \hat{y}) = h_\ell (h_{\ell-1}(h_{\ell-2}( \cdots h_1(x))) \]  

\[ a^{(1)} = x \]

\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]

\[ a^{(2)} = g(z^{(2)}) \]

\[ \vdots \]

\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]

\[ a^{(l+1)} = g(z^{(l+1)}) \]

\[ \vdots \]

\[ \hat{y} = a^{(L+1)} \]

Gradient Descent

Loop: over examples \((x_n, y_n)\)

For all i,j,l

\[ \Theta^{(l)}_{i,j} \leftarrow \Theta^{(l)}_{i,j} - \gamma \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]

\[ \delta^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]
Backprop

\[
\begin{align*}
    a^{(1)} &= x \\
    z^{(2)} &= \Theta^{(1)} a^{(1)} \\
    a^{(2)} &= g(z^{(2)}) \\
    &\vdots \\
    z^{(l+1)} &= \Theta^{(l)} a^{(l)} \\
    a^{(l+1)} &= g(z^{(l+1)}) \\
    \hat{y} &= a^{(L+1)} \\
    g'(z) &= g(z)(1-g(z))
\end{align*}
\]

\[
\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta^{(l+1)} \cdot a_j^{(l)}
\]

\[
\begin{align*}
    \delta_i^{(l)} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \\
    &= \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} g'(z_i^{(l)}) \\
    &= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}
\end{align*}
\]

\[
L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})
\]

\[
g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}
\]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]
Backprop

\( a^{(1)} = x \)

\( z^{(2)} = \Theta^{(1)} a^{(1)} \)

\( a^{(2)} = g \left( z^{(2)} \right) \)

\( z^{(l+1)} = \Theta^{(l)} a^{(l)} \)

\( a^{(l+1)} = g \left( z^{(l+1)} \right) \)

\( \hat{y} = a^{(L+1)} \)

\[
\frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta^{(l+1)} \cdot a_j^{(l)}
\]

\[
\delta_i^{(l)} = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}
\]

\[
\delta_i^{(L+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[ y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)})) \right]
\]

\[
= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)})
\]

\[
= y - g(z^{(L+1)}) = y - a^{(L+1)}
\]

\[
L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})
\]

\[
g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}
\]

©Kevin Jamieson
Backprop

\[ a^{(1)} = x \]

\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]

\[ a^{(2)} = g \left( z^{(2)} \right) \]

\[ \vdots \]

\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]

\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]

\[ \vdots \]

\[ \hat{y} = a^{(L+1)} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i} \]

\[ \delta^{(L+1)} = y - a^{(L+1)} \]

Recursive Algorithm!

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]

\[ \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]
\[ \delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l \]

Formally, \[ \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \]

where \[ \text{cost}(x_i) = y_i \log h_\Theta(x_i) + (1 - y_i) \log(1 - h_\Theta(x_i)) \]
Backpropagation Intuition

\[ \delta_j^{(l)} = \text{"error" of node } j \text{ in layer } l \]

Formally, \[ \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \]

where \[ \text{cost}(x_i) = y_i \log h_{\Theta}(x_i) + (1 - y_i) \log(1 - h_{\Theta}(x_i)) \]

\[ \delta^{(4)} = a_1^{(4)} - y \]

Based on slide by Andrew Ng
Backpropagation Intuition

\[ \delta_j^{(l)} = \text{“error” of node } j \text{ in layer } l \]

Formally, \[ \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \]

where \[ \text{cost}(x_i) = y_i \log h_{\Theta}(x_i) + (1 - y_i) \log(1 - h_{\Theta}(x_i)) \]
Backpropagation Intuition

\[ \delta_j^{(l)} = \text{"error" of node } j \text{ in layer } l \]

Formally, \[ \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \]

where \[ \text{cost}(x_i) = y_i \log h_\Theta(x_i) + (1 - y_i) \log(1 - h_\Theta(x_i)) \]

Based on slide by Andrew Ng
Backpropagation Intuition

$\delta_j^{(l)} = "error" \ of \ node \ j \ in \ layer \ l$

Formally, \( \delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(x_i) \)

where \( \text{cost}(x_i) = y_i \log h_\Theta(x_i) + (1 - y_i) \log(1 - h_\Theta(x_i)) \)

Based on slide by Andrew Ng
Backpropagation: Gradient Computation

Let $\delta_j^{(l)}$ = “error” of node $j$ in layer $l$

(#layers $L = 4$)

Backpropagation

- $\delta^{(4)} = a^{(4)} - y$
- $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \cdot g'(z^{(3)})$
- $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} \cdot g'(z^{(2)})$
- (No $\delta^{(1)}$)

Element-wise product

$g'(z^{(3)}) = a^{(3)} \cdot (1-a^{(3)})$
$g'(z^{(2)}) = a^{(2)} \cdot (1-a^{(2)})$

Based on slide by Andrew Ng
Backpropagation

Set $\Delta^{(l)}_{ij} = 0 \ \forall l, i, j$

For each training instance $(x_i, y_i)$:
- Set $a^{(1)} = x_i$
- Compute $\{a^{(2)}, \ldots, a^{(L)}\}$ via forward propagation
- Compute $\delta^{(L)} = a^{(L)} - y_i$
- Compute errors $\{\delta^{(L-1)}, \ldots, \delta^{(2)}\}$
- Compute gradients $\Delta^{(l)}_{ij} = \Delta^{(l)}_{ij} + a^{(l)}_j \delta^{(l+1)}_i$

Compute avg regularized gradient $D^{(l)}_{ij} = \begin{cases} \frac{1}{n} \Delta^{(l)}_{ij} + \lambda \Theta^{(l)}_{ij} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta^{(l)}_{ij} & \text{otherwise} \end{cases}$

$D^{(l)}$ is the matrix of partial derivatives of $J(\Theta)$

Based on slide by Andrew Ng
Training a Neural Network via Gradient Descent with Backprop

Given: training set \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \)

Initialize all \( \Theta^{(l)} \) randomly (NOT to 0!)

Loop // each iteration is called an epoch

- Set \( \Delta_{ij}^{(l)} = 0 \) \( \forall l, i, j \)  
  (Used to accumulate gradient)

- For each training instance \((x_i, y_i)\):
  - Set \( a_{\cdot 1}^{(1)} = x_i \)
  - Compute \( \{a_{\cdot 2}^{(2)}, \ldots, a_{\cdot L}^{(L)}\} \) via forward propagation
  - Compute \( \delta^{(L)} = a_{\cdot L}^{(L)} - y_i \)
  - Compute errors \( \{\delta^{(L-1)}, \ldots, \delta^{(2)}\} \)
  - Compute gradients \( \Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_{j}^{(l)} \delta^{(l+1)}_{i} \)

- Compute avg regularized gradient \( D_{ij}^{(l)} = \begin{cases} 
\frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\
\frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} 
\end{cases} \)

- Update weights via gradient step \( \Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)} \)

Until weights converge or max \#epochs is reached
Backprop for this simple network architecture is a special case of **reverse-mode auto-differentiation**:

\[ y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]

This is the special sauce in Tensorflow, PyTorch, Theano, …
Real networks

Modern networks have dozens of parameters to tune.

Data augmentation?  
Batch norm?

ReLU leakiness
slope

Learning rate schedule

$n_0$ layers of $f_0$ filters

$n_1$ layers of $f_1$ filters

$n_2$ layers of $f_2$ filters

$n_3$ layers of $f_3$ filters

Residual Network of  
[HeZhangRenSun’15]

Modern networks have dozens of parameters to tune.
Trees

\[ f(x) = \sum_{m=1}^{M} c_m I(x \in R_m). \]

Build a binary tree, splitting along axes.
Trees

Build a binary tree, splitting along axes

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

How do you split?

When do you stop?
Learning decision trees

- Start from empty decision tree
- Split on **next best attribute** (feature)
  - Use, for example, information gain to select attribute
  - Split on $\arg \max_i IG(X_i) = \arg \max_i H(Y) - H(Y \mid X_i)$
- Recurse
- Prune
Trees

- Trees
  - have low bias, high variance
  - deal with categorical variables well
  - intuitive, interpretable
  - good software exists
  - Some theoretical guarantees

\[ f(x) = \sum_{m=1}^{M} c_m I(x \in R_m). \]
Random Forests

Machine Learning – CSE546
Kevin Jamieson
University of Washington

November 29, 2018
Random Forests

Tree methods have **low bias** but **high variance**.

One way to reduce variance is to construct a lot of “lightly correlated” trees and average them:

“Bagging:” Bootstrap aggregating
**Algorithm 15.1** Random Forest for Regression or Classification.

1. For \( b = 1 \) to \( B \):
   
   (a) Draw a bootstrap sample \( Z^* \) of size \( N \) from the training data.
   
   (b) Grow a random-forest tree \( T_b \) to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size \( n_{min} \) is reached.
      
      i. Select \( m \) variables at random from the \( p \) variables.
      
      ii. Pick the best variable/split-point among the \( m \).
      
      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees \( \{T_b\}_1^B \).

To make a prediction at a new point \( x \):

**Regression:** \( \hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x) \). \( \text{m}\sim \frac{p}{3} \)

**Classification:** Let \( \hat{C}_b(x) \) be the class prediction of the \( b \)th random-forest tree. Then \( \hat{C}_{\text{rf}}^B(x) = \text{majority vote} \{\hat{C}_b(x)\}_1^B \). \( \text{m}\sim \sqrt{p} \)
The Kinect pose estimation pipeline

capture depth image & remove bg

infer body parts per pixel

cluster pixels to hypothesize body joint positions

fit model & track skeleton

Random Forests

Random forest

3 nearest neighbor
Random Forests

Given random variables $Y_1, Y_2, \ldots, Y_B$ with $
\mathbb{E}[Y_i] = y, \quad \mathbb{E}[(Y_i - y)^2] = \sigma^2, \quad \mathbb{E}[(Y_i - y)(Y_j - y)] = \rho\sigma^2
$

The $Y_i$'s are identically distributed but **not** independent

\[
\mathbb{E}[(\frac{1}{B} \sum_{i=1}^{B} Y_i - y)^2] = \frac{1}{\beta^2} \left( \sum_{i} \mathbb{E}[(Y_i - y)^2] + \sum_{i \neq j} \mathbb{E}[(Y_i - y)(Y_j - y)] \right)
\]

\[
= \frac{1}{\beta^2} \left( \beta \sigma^2 + B(\beta - 1) \rho \sigma^2 \right)
\]

\[
= \frac{1}{\beta} \sigma^2 + (1 - \frac{1}{B}) \rho \sigma^2
\]
Random Forests

• Random Forests
  • have low bias, low variance
  • deal with categorical variables well
  • not that intuitive or interpretable
  • Notion of confidence estimates
  • good software exists
  • Some theoretical guarantees
  • works well with default hyperparameters
Boosting

Machine Learning – CSE546
Kevin Jamieson
University of Washington

November 29, 2018
1988 Kearns and Valiant: “Can **weak learners** be combined to create a **strong learner**?”

**Weak learner definition (informal):**

An algorithm $A$ is a *weak learner* for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $A$ correctly classifies $h$ with error at most $1/2 - \gamma$.
Boosting

• 1988 Kearns and Valiant: “Can weak learners be combined to create a strong learner?”

Weak learner definition (informal):

An algorithm $A$ is a weak learner for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $A$ correctly classifies $h$ with error at most $1/2 - \gamma$

• 1990 Robert Schapire: “Yup!”

• 1995 Schapire and Freund: “Practical for 0/1 loss” AdaBoost
Boosting

- 1988 Kearns and Valiant: “Can weak learners be combined to create a strong learner?”

Weak learner definition (informal):

An algorithm $A$ is a weak learner for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $A$ correctly classifies $h$ with error at most $1/2 - \gamma$

- 1990 Robert Schapire: “Yup!”
- 1995 Schapire and Freund: “Practical for 0/1 loss” AdaBoost
- 2001 Friedman: “Practical for arbitrary losses”
Boosting

• 1988 Kearns and Valiant: “Can weak learners be combined to create a strong learner?”

Weak learner definition (informal):

An algorithm $A$ is a weak learner for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $A$ correctly classifies $h$ with error at most $1/2 - \gamma$

• 1990 Robert Schapire: “Yup!”

• 1995 Schapire and Freund: “Practical for 0/1 loss” AdaBoost

• 2001 Friedman: “Practical for arbitrary losses”

• 2014 Tianqi Chen: “Scale it up!” XGBoost
Boosting and Additive Models

Machine Learning – CSE546
Kevin Jamieson
University of Washington

November 29, 2018
Additive models

- Consider the first algorithm we used to get good classification for MNIST. Given: \( \{(x_i, y_i)\}_{i=1}^{n} \) \( x_i \in \mathbb{R}^d, y_i \in \{-1, 1\} \)

- Generate random functions: \( \phi_t : \mathbb{R}^d \rightarrow \mathbb{R} \) \( t = 1, \ldots, p \)

- Learn some weights: \( \hat{w} = \arg \min_w \sum_{i=1}^{n} \text{Loss} \left( y_i, \sum_{t=1}^{p} w_t \phi_t(x_i) \right) \)

- Classify new data: \( f(x) = \text{sign} \left( \sum_{t=1}^{p} \hat{w}_t \phi_t(x) \right) \)
• Consider the first algorithm we used to get good classification for MNIST. Given: \( \{(x_i, y_i)\}_{i=1}^{n} \) \( x_i \in \mathbb{R}^d, y_i \in \{-1, 1\} \)

• Generate **random** functions: \( \phi_t : \mathbb{R}^d \rightarrow \mathbb{R} \) \( t = 1, \ldots, p \)

• Learn some weights: \( \hat{w} = \arg \min_w \sum_{i=1}^{n} \text{Loss} \left( y_i, \sum_{t=1}^{p} w_t \phi_t(x_i) \right) \)

• Classify new data: \( f(x) = \text{sign} \left( \sum_{t=1}^{p} \hat{w}_t \phi_t(x) \right) \)

**An interpretation:**
Each \( \phi_t(x) \) is a classification rule that we are assigning some weight \( \hat{w}_t \)
Additive models

• Consider the first algorithm we used to get good classification for MNIST. Given: \(\{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\)

• Generate random functions: \(\phi_t : \mathbb{R}^d \rightarrow \mathbb{R} \quad t = 1, \ldots, p\)

• Learn some weights: \(\hat{w} = \text{arg min}_w \sum_{i=1}^{n} \text{Loss} \left( y_i, \sum_{t=1}^{p} w_t \phi_t(x_i) \right)\)

• Classify new data: \(f(x) = \text{sign} \left( \sum_{t=1}^{p} \hat{w}_t \phi_t(x) \right)\)

An interpretation:
Each \(\phi_t(x)\) is a classification rule that we are assigning some weight \(\hat{w}_t\)

\(\hat{w}, \hat{\phi}_1, \ldots, \hat{\phi}_t = \text{arg min}_{w, \phi_1, \ldots, \phi_p} \sum_{i=1}^{n} \text{Loss} \left( y_i, \sum_{t=1}^{p} w_t \phi_t(x_i) \right)\)

is in general computationally hard
Forward Stagewise Additive models

\[ b(x, \gamma) \] is a function with parameters \( \gamma \)

Examples:

\[ b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}} \]

\[ b(x, \gamma) = \gamma_1 1\{x_3 \leq \gamma_2\} \]

Algorithm 10.2 Forward Stagewise Additive Modeling.

1. Initialize \( f_0(x) = 0 \).
2. For \( m = 1 \) to \( M \):
   
   (a) Compute

   \[
   (\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).
   \]

   (b) Set \( f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \).

Idea: greedily add one function at a time
Forward Stagewise Additive models

\[ b(x, \gamma) \text{ is a function with parameters } \gamma \]

**Examples:**

\[ b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}} \]

\[ b(x, \gamma) = \gamma_1 1\{x_3 \leq \gamma_2\} \]

**Algorithm 10.2 Forward Stagewise Additive Modeling.**

1. Initialize \( f_0(x) = 0 \).
2. For \( m = 1 \) to \( M \):
   
   (a) Compute
   
   \[ (\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)). \]
   
   (b) Set \( f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \).

**Idea:** greedily add one function at a time

**AdaBoost:** \( b(x, \gamma) \): classifiers to \( \{-1, 1\} \)

\[ L(y, f(x)) = \exp(-y f(x)) \]
Forward Stagewise Additive models

\[ b(x, \gamma) \text{ is a function with parameters } \gamma \]

Examples:

\[ b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}} \]

\[ b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\} \]

Algorithm 10.2 Forward Stagewise Additive Modeling.

1. Initialize \( f_0(x) = 0 \).
2. For \( m = 1 \) to \( M \):
   
   (a) Compute
   \[
   (\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).
   \]
   
   (b) Set \( f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \).

Idea: greedily add one function at a time

Boosted Regression Trees:

\[ L(y, f(x)) = (y - f(x))^2 \]

\( b(x, \gamma) \): regression trees
Forward Stagewise Additive models

\[ b(x, \gamma) \] is a function with parameters \( \gamma \)

**Examples:**

\[ b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}} \]

\[ b(x, \gamma) = \gamma_1 1\{x_3 \leq \gamma_2\} \]

**Algorithm 10.2 Forward Stagewise Additive Modeling.**

1. Initialize \( f_0(x) = 0 \).
2. For \( m = 1 \) to \( M \):
   a. Compute
   \[
   (\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).
   \]
   b. Set \( f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \).

Idea: greedily add one function at a time

**Boosted Regression Trees:**

\[ L(y, f(x)) = (y - f(x))^2 \]

\[
L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2
= (r_{im} - \beta b(x_i; \gamma))^2, \quad r_{im} = y_i - f_{m-1}(x_i)
\]

Efficient: No harder than learning regression trees!
Forward Stagewise Additive models

\[ b(x, \gamma) \text{ is a function with parameters } \gamma \]

Examples:

\[ b(x, \gamma) = \frac{1}{1 + e^{-\gamma^T x}} \]

\[ b(x, \gamma) = \gamma_1 1\{x_3 \leq \gamma_2\} \]

**Algorithm 10.2 Forward Stagewise Additive Modeling.**

1. Initialize \( f_0(x) = 0 \).
2. For \( m = 1 \) to \( M \):
   
   (a) Compute
   
   \[ (\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)). \]
   
   (b) Set \( f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m) \).

Idea: greedily add one function at a time

**Boosted Logistic Trees:**

\[ L(y, f(x)) = y \log(f(x)) + (1 - y) \log(1 - f(x)) \]

\( b(x, \gamma) \): regression trees

Computationally hard to update
Gradient Boosting

Least squares, exponential loss easy. But what about cross entropy? Huber?

**Algorithm 10.3** Gradient Tree Boosting Algorithm.

1. Initialize $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$ r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}. $$
   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$ \gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma). $$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$. 

LS fit regression tree to n-dimensional gradient, take a step in that direction
Gradient Boosting

Least squares, 0/1 loss easy. But what about cross entropy? Huber?

Adaboost uses 0/1 loss, all other trees are minimizing binominal deviance
Additive models

- Boosting is popular at parties: Invented by theorists, heavily adopted by practitioners.
Additive models

• Boosting is popular at parties: Invented by theorists, heavily adopted by practitioners.

• Computationally efficient with “weak” learners. But can also use trees! Boosting can scale.

• Kind of like sparsity?
Additive models

• Boosting is popular at parties: Invented by theorists, heavily adopted by practitioners.

• Computationally efficient with “weak” learners. But can also use trees! Boosting can scale.

• Kind of like sparsity?

• Gradient boosting generalization with good software packages (e.g., XGBoost). Effective on Kaggle

• Robust to overfitting and can be dealt with with “shrinkage” and “sampling”
Bagging versus Boosting

- Bagging *averages* many *low-bias, lightly dependent* classifiers to reduce the variance.
- Boosting *learns* linear combination of *high-bias, highly dependent* classifiers to reduce error.
- Empirically, boosting appears to outperform bagging.