# CSC 2515 Lecture 4: Linear Models II

Roger Grosse

University of Toronto

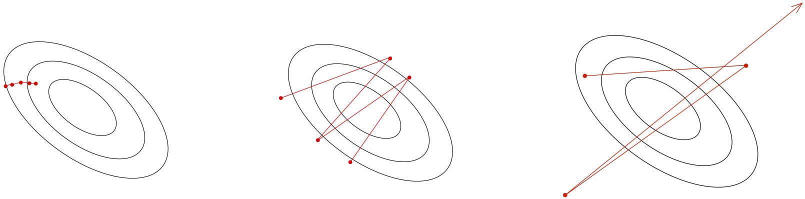
# Today's Agenda

Today's agenda:

- **Optimization**
  - **choice of learning rate**
  - **stochastic gradient descent**
- Multiclass classification
  - softmax regression
- $L^1$ regularization
- Support vector machines
- Boosting

# Learning Rate

- In gradient descent, the learning rate $\alpha$ is a hyperparameter we need to tune. Here are some things that can go wrong:



$\alpha$ too small: slow progress

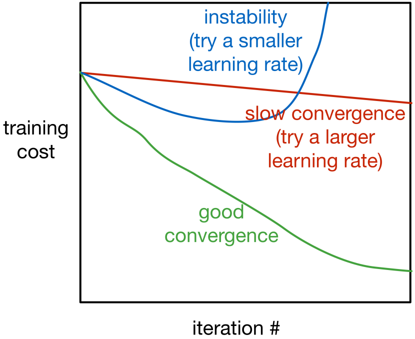
$\alpha$ too large: oscillations

$\alpha$ much too large: instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try $0.1, 0.03, 0.01, \ldots$).

# Training Curves

- To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.




- Warning: it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

# Stochastic Gradient Descent

- So far, the cost function $\mathcal{J}$ has been the average loss over the training examples:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}^{(i)} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y(\mathbf{x}^{(i)}, \boldsymbol{\theta}), t^{(i)}).$$

- By linearity,

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}.$$

- Computing the gradient requires summing over *all* of the training examples. This is known as batch training.
- Batch training is impractical if you have a large dataset (e.g. millions of training examples)!

# Stochastic Gradient Descent

- Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$$

- SGD can make significant progress before it has even looked at all the data!
- Mathematical justification: if you sample a training example at random, the stochastic gradient is an unbiased estimate of the batch gradient:

$$\mathbb{E}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}.$$

- Problem: if we only look at one training example at a time, we can't exploit efficient vectorized operations.

# Stochastic Gradient Descent

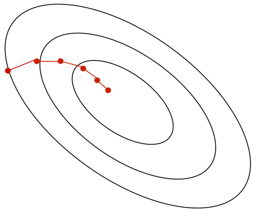
- Compromise approach: compute the gradients on a medium-sized set of training examples, called a mini-batch.
  - Conceptually, it's useful to think of mini-batches as sampled i.i..d. from the training set.
  - In practice, we typically go in order through the training set.
  - Each entire pass over the dataset is called an epoch.
- If mini-batches are independent, the stochastic gradients computed on larger mini-batches have smaller variance:

$$\mathrm{Var}\left[\frac{1}{S}\sum_{i=1}^{S}\frac{\partial \mathcal{L}^{(i)}}{\partial \theta_j}\right] = \frac{1}{S^2}\mathrm{Var}\left[\sum_{i=1}^{S}\frac{\partial \mathcal{L}^{(i)}}{\partial \theta_j}\right] = \frac{1}{S}\mathrm{Var}\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \theta_j}\right]$$

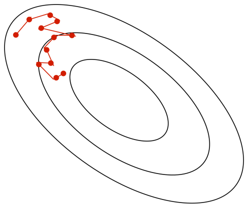
- The mini-batch size $S$ is a hyperparameter that needs to be set.
  - Too large: takes more memory to store the activations, and longer to compute each gradient update
  - Too small: can't exploit vectorization
  - A reasonable value might be $S = 100$.

# Stochastic Gradient Descent

- Batch gradient descent moves directly downhill. SGD takes steps in a noisy direction, but moves downhill on average.



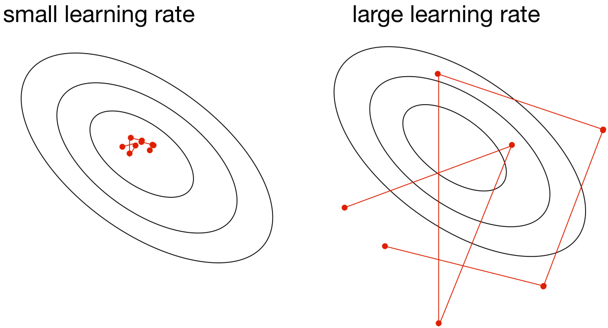
**batch gradient descent**



**stochastic gradient descent**

# SGD Learning Rate

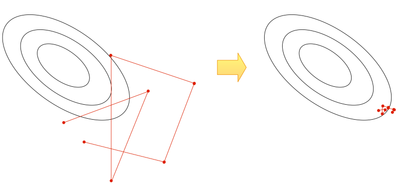
- In stochastic training, the learning rate also influences the fluctuations due to the stochasticity of the gradients.

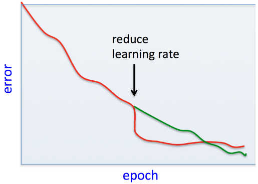



- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations

# SGD Learning Rate

- Warning: by reducing the learning rate, you reduce the fluctuations, which can appear to make the loss drop suddenly. But this can come at the expense of long-run performance.





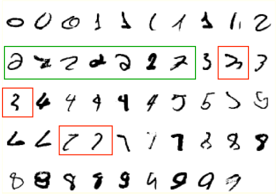
# Today's Agenda

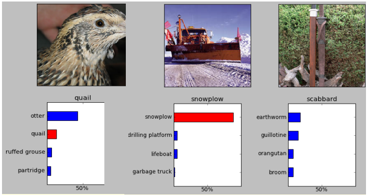
Today's agenda:

- Optimization
  - choice of learning rate
  - stochastic gradient descent
- **Multiclass classification**
  - **softmax regression**
- $L^1$ regularization
- Support vector machines
- Boosting

# Multiclass Classification

- What about classification tasks with more than two categories?





# Multiclass Classification

- Targets form a discrete set $\{1, \ldots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1}$$

# Multiclass Classification

- Now there are $D$ input dimensions and $K$ output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a $K$-dimensional vector $\mathbf{b}$ of biases.
- Linear predictions:

$$z_k = \sum_j w_{kj} x_j + b_k$$

- Vectorized:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

# Multiclass Classification

- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \mathrm{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

- The inputs $z_k$ are called the logits.
- Properties:
  - Outputs are positive and sum to 1 (so they can be interpreted as probabilities)
  - If one of the $z_k$'s is much larger than the others, $\mathrm{softmax}(\mathbf{z})$ is approximately the argmax. (So really it's more like "soft-argmax".)
  - **Exercise:** how does the case of $K = 2$ relate to the logistic function?
- Note: sometimes $\sigma(\mathbf{z})$ is used to denote the softmax function; in this class, it will denote the logistic function applied elementwise.

# Multiclass Classification

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\begin{aligned}\mathcal{L}_{\mathrm{CE}}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log y_k \\ &= -\mathbf{t}^\top (\log \mathbf{y}),\end{aligned}$$

  where the log is applied elementwise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

# Multiclass Classification

- Softmax regression:

$$\begin{aligned} \mathbf{z} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{y} &= \mathrm{softmax}(\mathbf{z}) \\ \mathcal{L}_{\mathrm{CE}} &= -\mathbf{t}^{\top}(\log \mathbf{y}) \end{aligned}$$

- Gradient descent updates are derived in the readings:

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial \mathbf{z}} = \mathbf{y} - \mathbf{t}$$

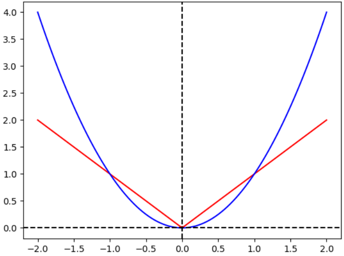
# Today's Agenda

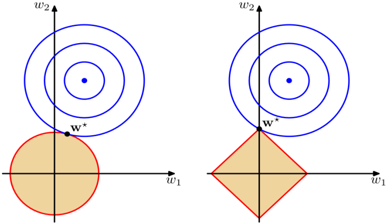
Today's agenda:

- Optimization
  - choice of learning rate
  - stochastic gradient descent
- Multiclass classification
  - softmax regression
- $L^1$ **regularization**
- Support vector machines
- Boosting

# $L^1$ vs. $L^2$ Regularization

- The $L^1$ norm, or sum of absolute values, is another regularizer that encourages weights to be exactly zero. (How can you tell?)
- We can design regularizers based on whatever property we'd like to encourage.
  - Which one will more strongly penalize very large weights?
  - Which one will try harder to push small weights towards zero?
- The derivative at a given value of $w_i$ determines how hard the regularizer "pushes."




L2 regularization

$$\mathcal{R} = \sum_i w_i^2$$

L1 regularization

$$\mathcal{R} = \sum_i |w_i|$$

— Bishop, *Pattern Recognition and Machine Learning*

# $L^1$ vs. $L^2$ Regularization

- $L^1$-regularized linear regression:

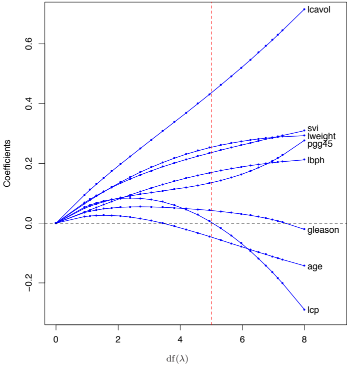
$$\mathcal{J}(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (\mathbf{w}^\top \mathbf{x}^{(i)} - t^{(i)})^2 \; + \; \lambda \sum_{j=1}^{D} |w_j|$$

- What happens when $\lambda$ is very large?
- In general, the optimal weight vector will be sparse, i.e. many of the weights will be exactly zero.
    - This is useful in situations where you have lots of features, but only a small fraction of them are likely to be relevant (e.g. genetics).
- The above cost function is a quadratic program, a more difficult optimization problem than for $L^2$ regularization.
    - What would go wrong if you just apply gradient descent?
    - Fast algorithms are implemented in frameworks like scikit-learn.

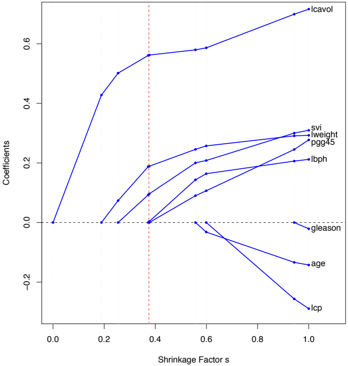
# $L^1$ vs. $L^2$ Regularization

- How the linear regression weights evolve for $L^2$ and $L^1$ regularization, as a function of the regularization parameter $\lambda$.
  - $\lambda$ *decreases* as you move to the right.

$L^2$ regularization




$L^1$ regularization




— *Elements of Statistical Learning*

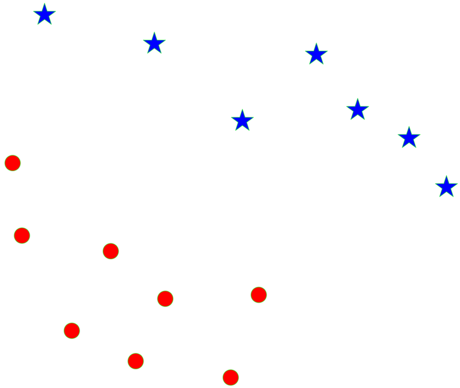
# Today's Agenda

Today's agenda:

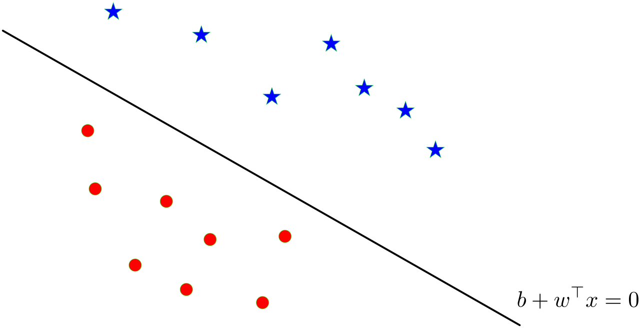
- Optimization
  - choice of learning rate
  - stochastic gradient descent
- Multiclass classification
  - softmax regression
- $L^1$ regularization
- **Support vector machines**
- Boosting

# Separating Hyperplanes

Suppose we are given these data points from two different classes and want to find a linear classifier that separates them.

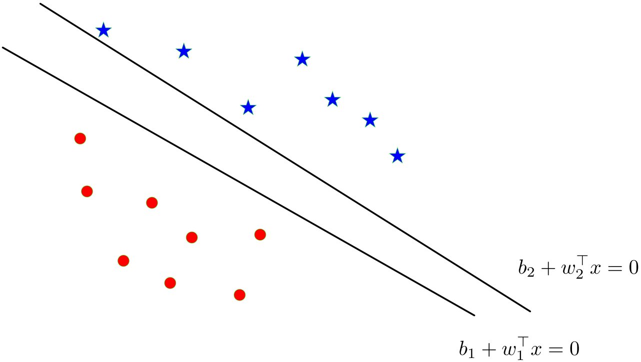


# Separating Hyperplanes



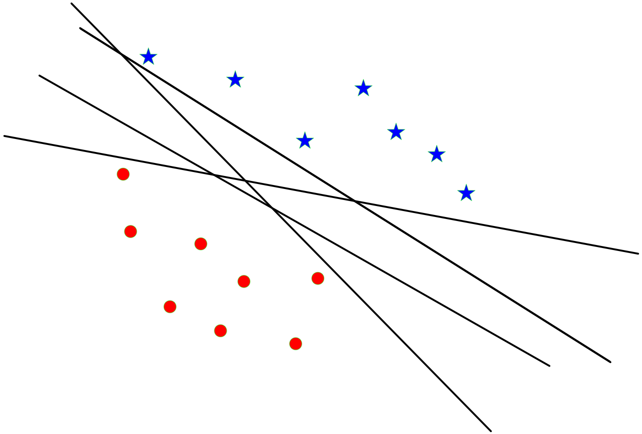

- The decision boundary looks like a line because $\mathbf{x} \in \mathbb{R}^2$, but think about it as a $D - 1$ dimensional hyperplane.
- Recall that a hyperplane is described by points $\mathbf{x} \in \mathbb{R}^D$ such that $f(\mathbf{x}) = \mathbf{w}^\top x + b = 0$.

# Separating Hyperplanes



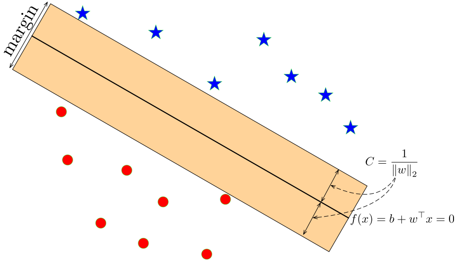

- There are multiple separating hyperplanes, described by different parameters $(\mathbf{w}, b)$.

# Separating Hyperplanes



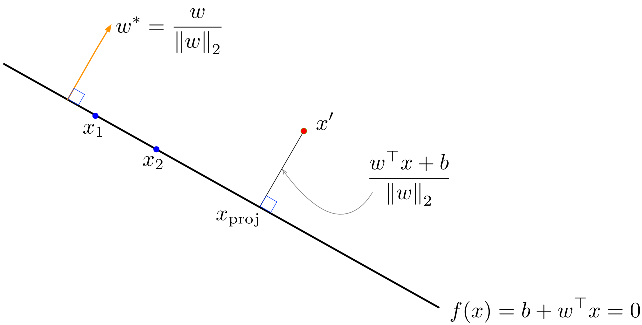
# Optimal Separating Hyperplane

Optimal Separating Hyperplane: A hyperplane that separates two classes and maximizes the distance to the closest point from either class, i.e., maximize the margin of the classifier.



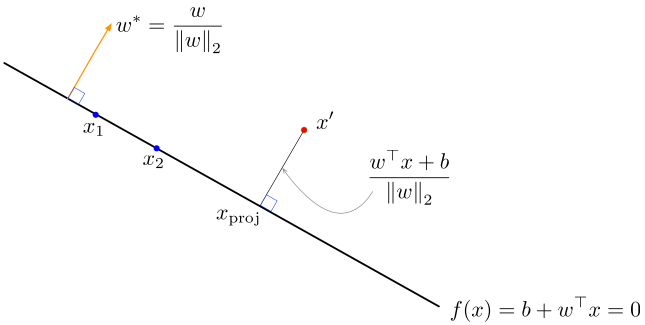

Intuitively, ensuring that a classifier is not too close to any data points leads to better generalization on the test data.

# Geometry of Points and Planes




- Recall that the decision hyperplane is orthogonal (perpendicular) to $\mathbf{w}$.
- The vector $\mathbf{w}^* = \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ is a unit vector pointing in the same direction as $\mathbf{w}$.
- The same hyperplane could equivalently be defined in terms of $\mathbf{w}^*$.

# Geometry of Points and Planes




The signed distance of a point $\mathbf{x}'$ to the hyperplane is

$$\frac{\mathbf{w}^\top \mathbf{x}' + b}{\|\mathbf{w}\|_2}$$

# Maximizing Margin as an Optimization Problem

- Recall: the classification for the $i$-th data point is correct when

$$\text{sign}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) = t^{(i)}$$

- This can be rewritten as

$$t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) > 0$$

- Enforcing a margin of $C$:

$$t^{(i)} \cdot \underbrace{\frac{(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2}}_{\text{signed distance}} \geq C$$

# Maximizing Margin as an Optimization Problem

Max-margin objective:

$$\max_{\mathbf{w},b} C$$
$$\text{s.t.} \quad \frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C \qquad i = 1, \ldots, N$$

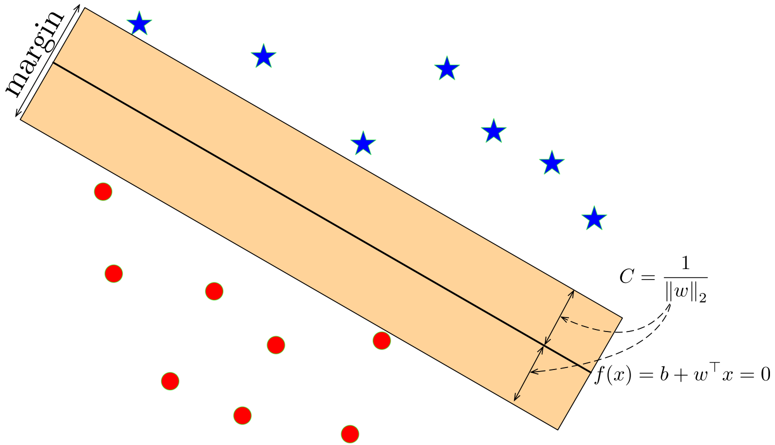
Plug in $C = 1/\|\mathbf{w}\|_2$ and simplify:

$$\underbrace{\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq \frac{1}{\|\mathbf{w}\|_2}}_{\text{geometric margin constraint}} \qquad \iff \qquad \underbrace{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1}_{\text{algebraic margin constraint}}$$

Equivalent optimization objective:

$$\min \|\mathbf{w}\|_2^2$$
$$\text{s.t.} \quad t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \qquad i = 1, \ldots, N$$

# Maximizing Margin as an Optimization Problem

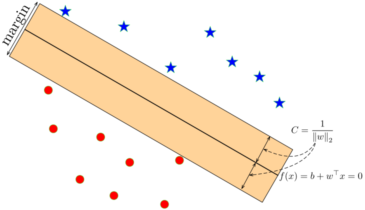


# Maximizing Margin as an Optimization Problem

Algebraic max-margin objective:

$$\min_{\mathbf{w},b} \|\mathbf{w}\|_2^2$$

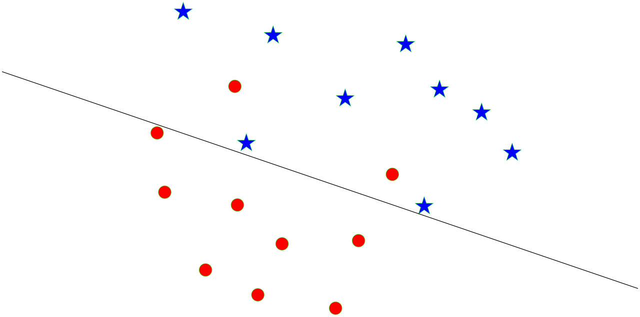
$$\text{s.t. } t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \qquad i = 1, \ldots, N$$



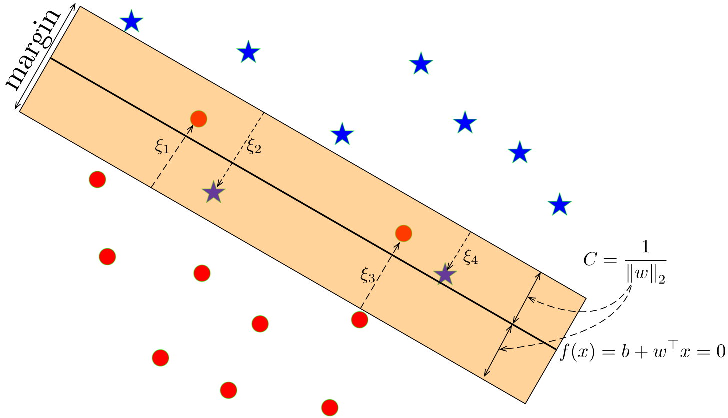

- Observe: if the margin constraint is not tight for $\mathbf{x}^{(i)}$, we could remove it from the training set and the optimal $\mathbf{w}$ would be the same.
- The important training examples are the ones with algebraic margin 1, and are called support vectors.
- Hence, this algorithm is called the (hard) Support Vector Machine (SVM) (or Support Vector Classifier).
- SVM-like algorithms are often called max-margin or large-margin.

# Non-Separable Data Points

How can we apply the max-margin principle if the data are **not** linearly separable?



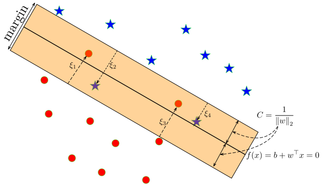
# Maximizing Margin for Non-Separable Data Points




Main Idea:

- Allow some points to be within the margin or even be misclassified; we represent this with slack variables $\xi_i$.
- But constrain or penalize the total amount of slack.

# Maximizing Margin for Non-Separable Data Points




- Soft margin constraint:

$$\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|_2} \geq C(1 - \xi_i),$$

  for $\xi_i \geq 0$.

- Penalize $\sum_i \xi_i$

# Maximizing Margin for Non-Separable Data Points

Soft-margin SVM objective:

$$
\begin{aligned}
\min_{\mathbf{w},b,\xi} \; & \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{N} \xi_i \\
\text{s.t.} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi_i \qquad & i = 1, \ldots, N \\
& \xi_i \geq 0 \qquad & i = 1, \ldots, N
\end{aligned}
$$

- $\gamma$ is a hyperparameter that trades off the margin with the amount of slack.
  - For $\gamma = 0$, we'll get $\mathbf{w} = 0$. (Why?)
  - As $\gamma \to \infty$ we get the hard-margin objective.
- Note: it is also possible to constrain $\sum_i \xi_i$ instead of penalizing it.

# From Margin Violation to Hinge Loss

Let's simplify the soft margin constraint by eliminating $\xi_i$. Recall:

$$\begin{aligned} t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) &\geq 1 - \xi_i & i = 1, \ldots, N \\ \xi_i &\geq 0 & i = 1, \ldots, N \end{aligned}$$

- Rewrite as $\xi_i \geq 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$.
- **Case 1:** $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \leq 0$
  - The smallest non-negative $\xi_i$ that satisfies the constraint is $\xi_i = 0$.
- **Case 2:** $1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) > 0$
  - The smallest $\xi_i$ that satisfies the constraint is $\xi_i = 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)$.
- Hence, $\xi_i = \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}$.
- Therefore, the slack penalty can be written as

$$\sum_{i=1}^{N} \xi_i = \sum_{i=1}^{N} \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}.$$

- We sometimes write $\max\{0, y\} = (y)_+$

# From Margin Violation to Hinge Loss

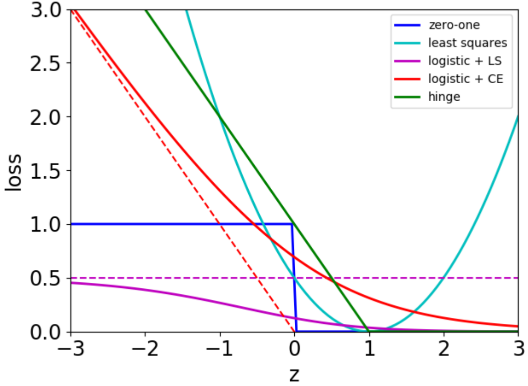
If we write $y^{(i)}(\mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$, then the optimization problem can be written as

$$\min_{\mathbf{w},b,\xi} \sum_{i=1}^{N} \left(1 - t^{(i)} y^{(i)}(\mathbf{w}, b)\right)_+ + \frac{1}{2\gamma} \|\mathbf{w}\|_2^2$$

- The loss function $\mathcal{L}_\mathrm{H}(y, t) = (1 - ty)_+$ is called the hinge loss.
- The second term is the $L_2$-norm of the weights.
- Hence, the soft-margin SVM can be seen as a linear classifier with hinge loss and an $L_2$ regularizer.

# Revisiting Loss Functions for Classification

Hinge loss compared with other loss functions



# SVMs: What we Left Out

What we left out:

- How to fit $\mathbf{w}$:
  - One option: gradient descent
  - Can reformulate with the Lagrange dual
- The "kernel trick" converts it into a powerful nonlinear classifier. This is covered in CSC2506 and CSC2547.
- Classic results from learning theory show that a large margin implies good generalization.

# Today's Agenda

Today's agenda:

- Optimization
  - choice of learning rate
  - stochastic gradient descent
- Multiclass classification
  - softmax regression
- $L^1$ regularization
- Support vector machines
- **Boosting**

# Boosting

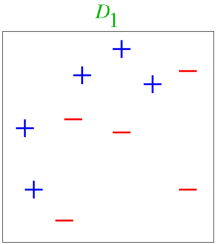
- Recall that an *ensemble* is a set of predictors whose individual decisions are combined in some way to classify new examples.
- (Lecture 2) **Bagging**: Train classifiers independently on random subsets of the training data.
- (This lecture) **Boosting**: Train classifiers sequentially, each time focusing on training data points that were previously misclassified.
- Let us start with the concept of weak learner/classifier (or base classifiers).

# Weak Learner/Classifier

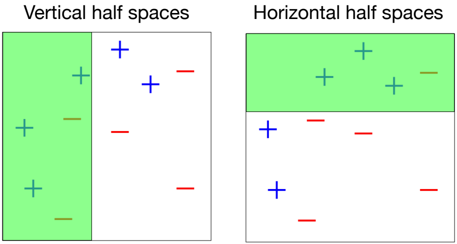
- (Informal) Weak learner is a learning algorithm that outputs a hypothesis (e.g., a classifier) that performs slightly better than chance, e.g., it predicts the correct label with probability 0.6.
- We are interested in weak learners that are *computationally* efficient.
  - Decision trees
  - Even simpler: Decision Stump: A decision tree with only a single split

[Formal definition of weak learnability has quantifies such as "for any distribution over data" and the requirement that its guarantee holds only probabilistically.]

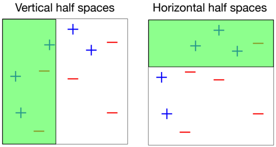
# Weak Classifiers




These weak classifiers, which are decision stumps, consist of the set of horizontal and vertical half spaces.



# Weak Classifiers




- A *single* weak classifier is not capable of making the training error very small. It only perform slightly better than chance, i.e., the error of classifier $h$ according to the given weights $\mathbf{w} = (w_1, \dots, w_N)$ (with $\sum_{i=1}^{N} w_i = 1$ and $w_i \geq 0$)

$$\text{err} = \sum_{i=1}^{N} w_i \mathbb{I}\{h(\mathbf{x}_i) \neq y_i\}$$

  is at most $\frac{1}{2} - \gamma$ for some $\gamma > 0$.

- Can we combine a set of weak classifiers in order to make a better ensemble of classifiers?

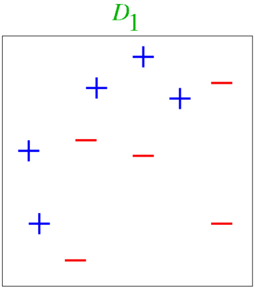
- Boosting: Train classifiers sequentially, each time focusing on training data points that were previously misclassified.

# AdaBoost (Adaptive Boosting)

- Key steps of AdaBoost:
  1. At each iteration we re-weight the training samples by assigning larger weights to samples (i.e., data points) that were classified incorrectly.
  2. We train a new weak classifier based on the re-weighted samples.
  3. We add this weak classifier to the ensemble of classifiers. This is our new classifier.
  4. We repeat the process many times.
- The weak learner needs to minimize weighted error.
- AdaBoost reduces bias by making each classifier focus on previous mistakes.

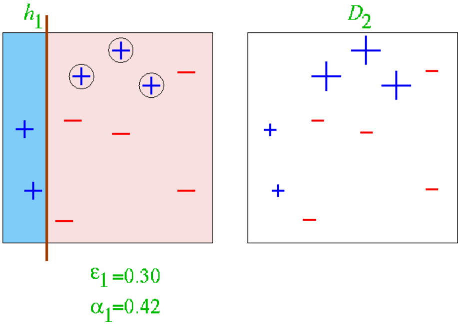
# AdaBoost Example

- Training data




[Slide credit: Verma & Thrun]

- Round 1

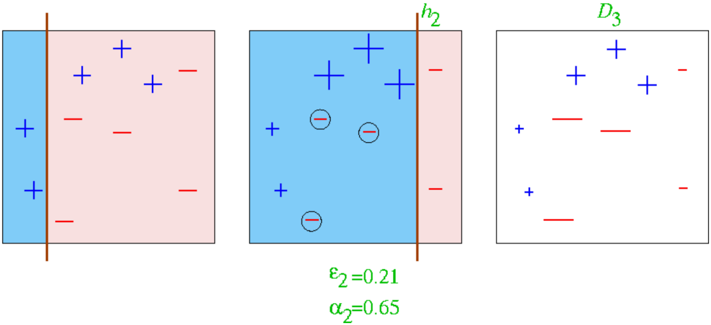



$$\mathbf{w} = \left(\frac{1}{10}, \ldots, \frac{1}{10}\right) \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_1 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_1(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = \frac{3}{10}$$

$$\Rightarrow \alpha_1 = \frac{1}{2} \log \frac{1 - \text{err}_1}{\text{err}_1} = \frac{1}{2} \log(\frac{1}{0.3} - 1) \approx 0.42 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x})\right)$$

[Slide credit: Verma & Thrun]

- Round 2

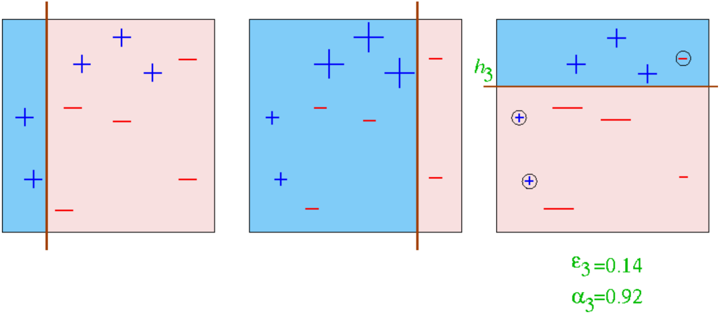



$$\mathbf{w} = \text{updated weights} \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_2 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_2(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = 0.21$$

$$\Rightarrow \alpha_2 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log\left(\frac{1}{0.21} - 1\right) \approx 0.66 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x})\right)$$

# AdaBoost Example

- Round 3



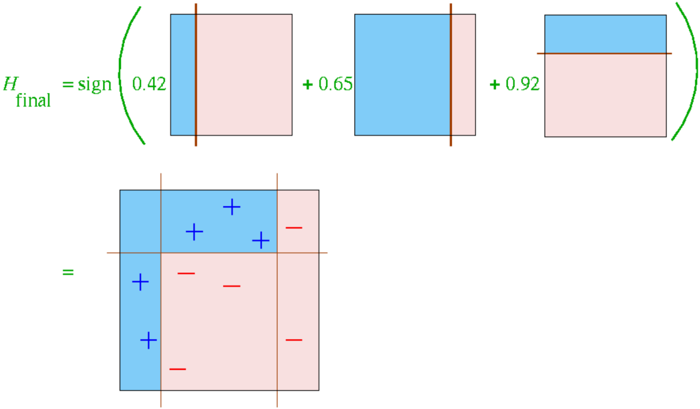

$$\mathbf{w} = \text{updated weights} \Rightarrow \text{Train a classifier (using } \mathbf{w}) \Rightarrow \text{err}_3 = \frac{\sum_{i=1}^{10} w_i \mathbb{I}\{h_3(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i} = 0.14$$

$$\Rightarrow \alpha_3 = \frac{1}{2} \log \frac{1 - \text{err}_3}{\text{err}_3} = \frac{1}{2} \log(\frac{1}{0.14} - 1) \approx 0.91 \Rightarrow H(\mathbf{x}) = \text{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \alpha_3 h_3(\mathbf{x})\right)$$

[Slide credit: Verma & Thrun]

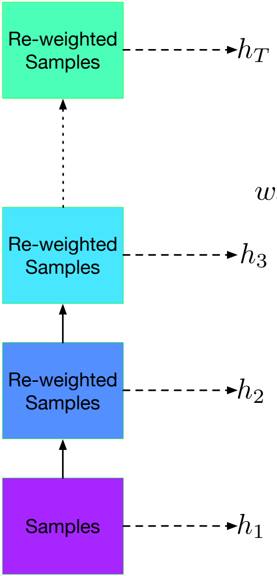
# AdaBoost Example

- Final classifier




[Slide credit: Verma & Thrun]

# AdaBoost Algorithm




$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

$$w_i \leftarrow w_i \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(\mathbf{i})}) \neq \mathbf{t}^{(\mathbf{i})}\}\right)$$

$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\text{err}_t}{\text{err}_t}\right)$$

$$\text{err}_t = \frac{\sum_{i=1}^{N} w_i \mathbb{I}\{h_t(\mathbf{x}^{(\mathbf{i})}) \neq \mathbf{t}^{(\mathbf{i})}\}}{\sum_{i=1}^{N} w_i}$$

# AdaBoost Algorithm

- Input: Data $\mathcal{D}_N = \{\mathbf{x}^{(i)}, t^{(i)}\}_{i=1}^N$, weak classifier WeakLearn (a classification procedure that return a classifier from base hypothesis space $\mathcal{H}$ with $h : \mathbf{x} \to \{-1, +1\}$ for $h \in \mathcal{H}$), number of iterations $T$
- Output: Classifier $H(x)$
- Initialize sample weights: $w_i = \frac{1}{N}$ for $i = 1, \ldots, N$
- For $t = 1, \ldots, T$
  - Fit a classifier to data using weighted samples ($h_t \leftarrow WeakLearn(\mathcal{D}_N, \mathbf{w})$), e.g.,

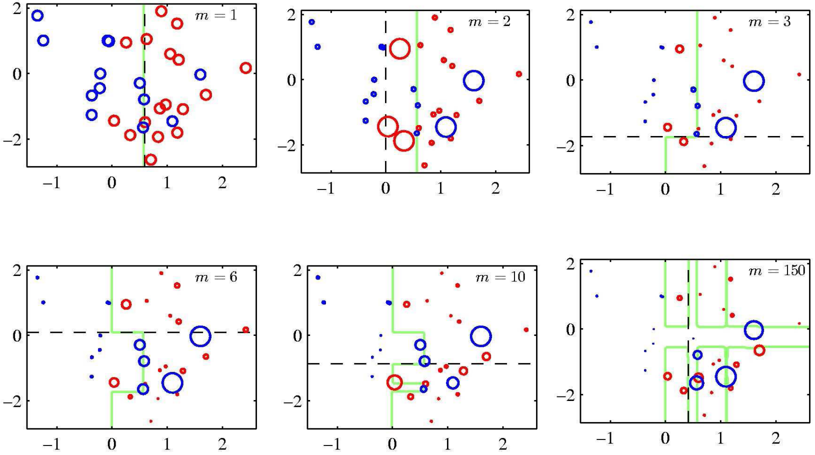
$$h_t \leftarrow \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{N} w_i \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\}$$

  - Compute weighted error $\text{err}_t = \frac{\sum_{i=1}^N w_i \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i}$
  - Compute classifier coefficient $\alpha_t = \frac{1}{2} \log \frac{1 - \text{err}_t}{\text{err}_t}$
  - Update data weights

$$w_i \leftarrow w_i \exp\left(-\alpha_t t^{(i)} h_t(\mathbf{x}^{(i)})\right) \left[\equiv w_i \exp\left(2\alpha_t \mathbb{I}\{h_t(\mathbf{x}^{(i)}) \neq t^{(i)}\}\right)\right]$$

- Return $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x})\right)$

# AdaBoost Example




- Each figure shows the number *m* of base learners trained so far, the decision of the most recent learner (dashed black), and the boundary of the ensemble (green)

# AdaBoost Minimizes the Training Error

**Theorem**

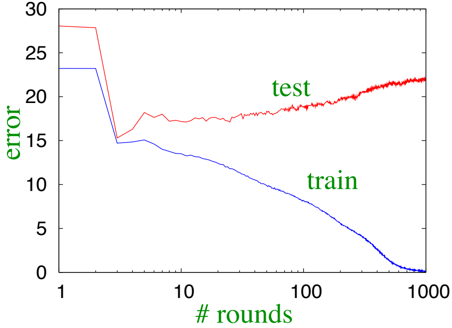
Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $\mathrm{err}_t \leq \frac{1}{2} - \gamma$ for all $t = 1, \ldots, T$ with $\gamma > 0$. The training error of the output hypothesis $H(\mathbf{x}) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$ is at most

$$L_N(H) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{H(\mathbf{x}^{(i)}) \neq t^{(i)})\} \leq \exp\left(-2\gamma^2 T\right).$$

- This is under the simplifying assumption that each weak learner is $\gamma$-better than a random predictor.
- Maybe this assumption is less innocuous than it seems.

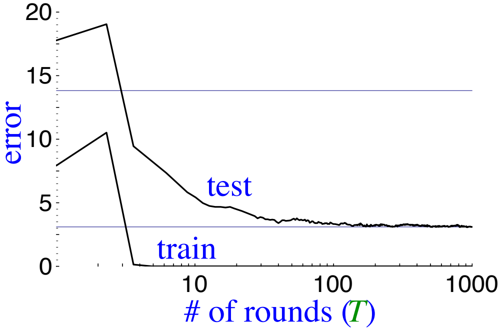
# Generalization Error of AdaBoost

- AdaBoost's training error (loss) converges to zero. What about the test error of $H$?
- As we add more weak classifiers, the overall classifier $H$ becomes more "complex".
- We expect more complex classifiers overfit.
- If one runs AdaBoost long enough, it can in fact overfit.



# Generalization Error of AdaBoost

- But often it does not!
- Sometimes the test error decreases even after the training error is zero!




[Slide credit: Robert Shapire's Slides, `http://www.cs.princeton.edu/courses/archive/spring12/cos598A/schedule.html` ]

# Additive Models

- Consider a hypothesis class $\mathcal{H}$ with each $h_i : \mathbf{x} \mapsto \{-1, +1\}$ within $\mathcal{H}$, i.e., $h_i \in \mathcal{H}$. These are the "weak learners", and in this context they're also called bases.
- An additive model with $m$ terms is given by

$$H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}),$$

  where $(\alpha_1, \cdots, \alpha_m) \in \mathbb{R}^m$.
- Observe that we're taking a linear combination of base classifiers, just like in boosting.
- We'll now interpret AdaBoost as a way of fitting an additive model.

# Stagewise Training of Additive Models

A greedy approach to fitting additive models, known as stagewise training:

1. Initialize $H_0(x) = 0$
2. For $m = 1$ to $T$:
   - Compute the $m$-th hypothesis and its coefficient

$$(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}\left(H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)}), t^{(i)}\right)$$

   - Add it to the additive model

$$H_m = H_{m-1} + \alpha_m h_m$$

# Additive Models with Exponential Loss

Consider the exponential loss

$$\mathcal{L}_{\mathrm{E}}(y, t) = \exp(-ty).$$

We want to see how the stagewise training of additive models can be done.

$$\begin{aligned}
(h_m, \alpha_m) \leftarrow \operatorname*{argmin}_{h \in \mathcal{H}, \alpha} \sum_{i=1}^{N} \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha h(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\
= \sum_{i=1}^{N} \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)} - \alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \\
= \sum_{i=1}^{N} \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right) \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \\
= \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right).
\end{aligned}$$

Here we defined $w_i^{(m)} \triangleq \exp\left(-H_{m-1}(\mathbf{x}^{(i)}) t^{(i)}\right)$.

# Additive Models with Exponential Loss

We want to solve the following minimization problem:

$$(h_m, \alpha_m) \leftarrow \underset{h \in \mathcal{H}, \alpha}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right).$$

- If $h(\mathbf{x}^{(i)}) = t^{(i)}$, we have $\exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) = \exp(-\alpha)$.
- If $h(\mathbf{x}^{(i)}) \neq t^{(i)}$, we have $\exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) = \exp(+\alpha)$.

(recall that we are in the binary classification case with $\{-1, +1\}$ output values).
We can divide the summation to two parts:

$$\begin{aligned}
\sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) =& e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) = t_i\} + e^{\alpha} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t_i\} \\
=& (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t_i\} + \\
& e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)} \left[\mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t_i\} + \mathbb{I}\{h(\mathbf{x}^{(i)}) = t_i\}\right]
\end{aligned}$$

# Additive Models with Exponential Loss

$$\begin{aligned}
\sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) =& (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)} \neq t_i\} + \\
& e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)} \left[\mathbb{I}\{h(\mathbf{x}^{(i)} \neq t_i\} + \mathbb{I}\{h(\mathbf{x}^{(i)}) = t_i\}\right] \\
=& (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t_i\} + e^{-\alpha} \sum_{i=1}^{N} w_i^{(m)}.
\end{aligned}$$

Let us first optimize $h$:
The second term on the RHS does not depend on $h$. So we get

$$h_m \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) \equiv \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t_i\}.$$

This means that $h_m$ is the minimizer of the weighted 0/1-loss.

# Additive Models with Exponential Loss

Now that we obtained $h_m$, we want to find $\alpha$: Define the weighted classification error:

$$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i^{(m)}}$$

With this definition and
$\min_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) = \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t_i\}$, we have

$$\min_{\alpha} \min_{h \in \mathcal{H}} \sum_{i=1}^N w_i^{(m)} \exp\left(-\alpha h(\mathbf{x}^{(i)}) t^{(i)}\right) =$$

$$\min_{\alpha} \left\{ (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^N w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t_i\} + e^{-\alpha} \sum_{i=1}^N w_i^{(m)} \right\}$$

$$= \min_{\alpha} \left\{ (e^{\alpha} - e^{-\alpha}) \mathrm{err}_m \left( \sum_{i=1}^N w_i^{(m)} \right) + e^{-\alpha} \left( \sum_{i=1}^N w_i^{(m)} \right) \right\}$$

Take derivative w.r.t. $\alpha$ and set it to zero. We get that

$$e^{2\alpha} = \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \Rightarrow \alpha = \frac{1}{2} \log \left( \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \right).$$

# Additive Models with Exponential Loss

The updated weights for the next iteration is

$$
\begin{aligned}
w_i^{(m+1)} &= \exp\left(-H_m(\mathbf{x}^{(i)})t^{(i)}\right) \\
&= \exp\left(-\left[H_{m-1}(\mathbf{x}^{(i)}) + \alpha_m h_m(\mathbf{x}^{(i)})\right] t^{(i)}\right) \\
&= \exp\left(-H_{m-1}(\mathbf{x}^{(i)})t^{(i)}\right) \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right) \\
&= w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right) \\
&= w_i^{(m)} \exp\left(-\alpha_m \left(2\mathbb{I}\{h_m(\mathbf{x}^{(i)}) = t^{(i)}\} - 1\right)\right) \\
&= \exp(\alpha_m) w_i^{(m)} \exp\left(-2\alpha_m \mathbb{I}\{h_m(\mathbf{x}^{(i)}) = t^{(i)}\}\right).
\end{aligned}
$$

The term $\exp(\alpha_m)$ multiplies the weight corresponding to all samples, so it does not affect the minimization of $h_{m+1}$ or $\alpha_{m+1}$.

# Additive Models with Exponential Loss

To summarize, we obtain the additive model $H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x})$ with

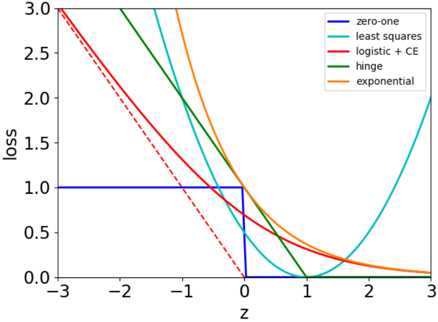
$$h_m \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t_i\},$$

$$\alpha = \frac{1}{2} \log \left( \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \right), \qquad \text{where } \mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^{N} w_i^{(m)}},$$

$$w_i^{(m+1)} = w_i^{(m)} \exp\left(-\alpha_m h_m(\mathbf{x}^{(i)}) t^{(i)}\right).$$

We derived the AdaBoost algorithm!

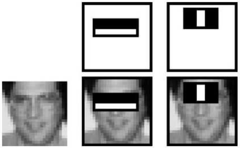
# Revisiting Loss Functions for Classification




- If AdaBoost is minimizing exponential loss, what does that say about its behavior (compared to, say, logistic regression)?
- This interpretation allows boosting to be generalized to lots of other loss functions!

# AdaBoost for Face Recognition

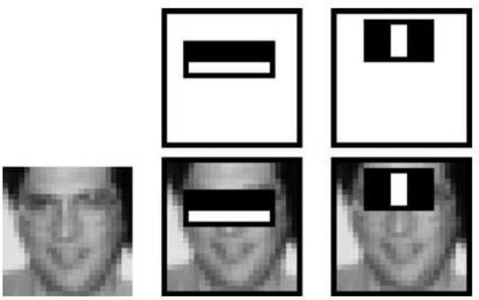
- Viola and Jones (2001) created a very fast face detector that can be scanned across a large image to find the faces.



- The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image.
    - There is a neat trick for computing the total intensity in a rectangle in a few operations.
        - So it is easy to evaluate a huge number of base classifiers and they are very fast at runtime.
    - The algorithm adds classifiers greedily based on their quality on the weighted training cases.

# AdaBoost for Face Detection

- A few twists on standard algorithm
  - Pre-define weak classifiers, so optimization=selection
  - Change loss function for weak learners: false positives less costly than misses
  - Smart way to do inference in real-time (in 2001 hardware)



# AdaBoost Face Detection Results