Machine Learning (CSE 446): Learning as Minimizing Loss

Noah Smith

University of Washington
nasmith@cs.washington.edu

October 23, 2017

Sorry! No office hour for me today. Wednesday is as usual.

Perceptron

A model and an algorithm, rolled into one.

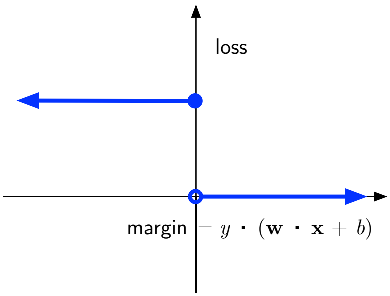
Model: $f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$, known as **linear**, visualized by a (hopefully) separating hyperplane in feature-space.

Algorithm: $\textsc{PerceptronTrain}$, an error-driven, iterative updating algorithm.

A Different View of PerceptronTrain: Optimization

"Minimize training-set error rate":

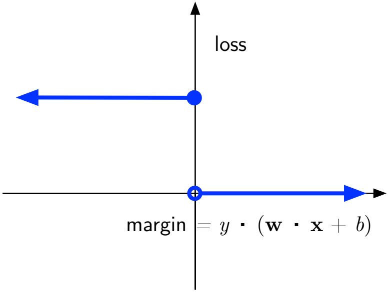
$$\min_{\mathbf{w},b} \underbrace{\frac{1}{N}\sum_{n=1}^{N}[\![y_n \cdot (\mathbf{w}\cdot\mathbf{x}+b) \le 0]\!]}_{\epsilon^{\text{train}} \equiv \text{ zero-one loss}}$$



A Different View of PerceptronTrain: Optimization

"Minimize training-set error rate":

$$\underbrace{\min_{\mathbf{w},b} \frac{1}{N} \sum_{n=1}^{N} [\![y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b) \leq 0]\!]}_{\epsilon^{\text{train}} \equiv \text{ zero-one loss}}$$



This problem is NP-hard; even solving it approximately (i.e., getting within a small constant factor of the optimal value) is NP-hard!

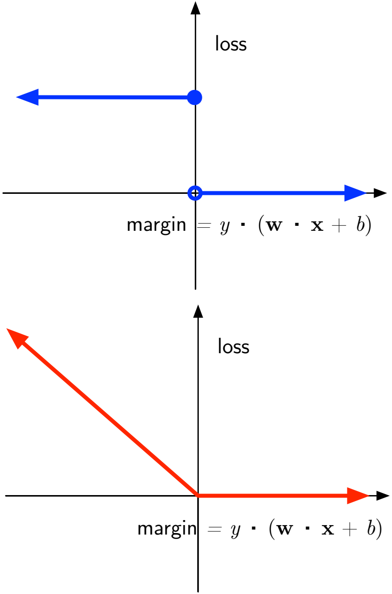
A Different View of PerceptronTrain: Optimization

"Minimize training-set error rate":

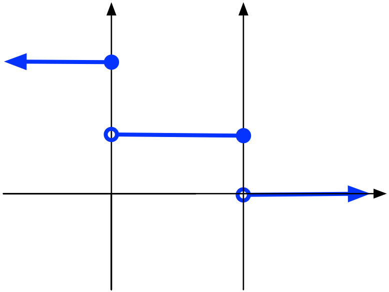
$$\underbrace{\min_{\mathbf{w},b} \frac{1}{N}\sum_{n=1}^{N} [\![y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b) \leq 0]\!]}_{\epsilon^{\text{train}} \equiv \text{ zero-one loss}}$$

What the perceptron does:

$$\min_{\mathbf{w},b} \frac{1}{N}\sum_{n=1}^{N} \underbrace{\max(-y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b), \quad 0)}_{\text{perceptron loss}}$$

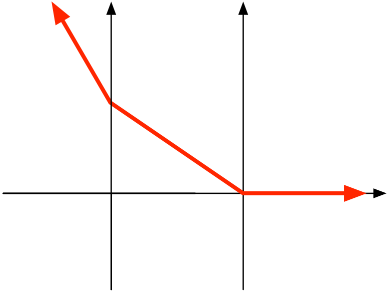


A Different View of PerceptronTrain: Optimization



"Minimize training-set error rate":

$$\underbrace{\min_{\mathbf{w},b} \frac{1}{N}\sum_{n=1}^{N} [\![y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b) \leq 0]\!]}_{\epsilon^{\text{train}} \equiv \text{ zero-one loss}}$$



What the perceptron does:

$$\min_{\mathbf{w},b} \frac{1}{N}\sum_{n=1}^{N} \underbrace{\max(-y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b), \quad 0)}_{\text{perceptron loss}}$$

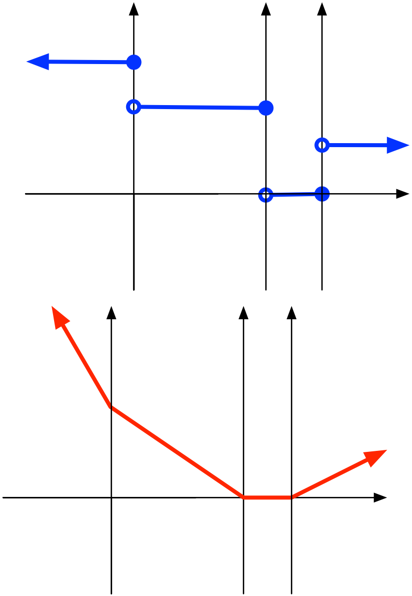
A Different View of PerceptronTrain: Optimization

"Minimize training-set error rate":

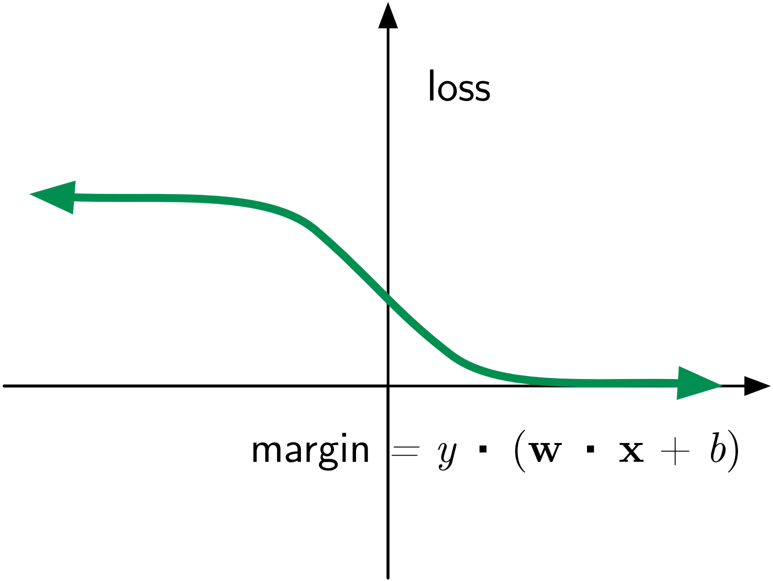
$$\min_{\mathbf{w},b} \underbrace{\frac{1}{N}\sum_{n=1}^{N} [\![y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b) \leq 0]\!]}_{\epsilon^{\text{train}} \equiv \text{ zero-one loss}}$$

What the perceptron does:

$$\min_{\mathbf{w},b} \frac{1}{N}\sum_{n=1}^{N} \underbrace{\max(-y_n \cdot (\mathbf{w} \cdot \mathbf{x} + b), \quad 0)}_{\text{perceptron loss}}$$

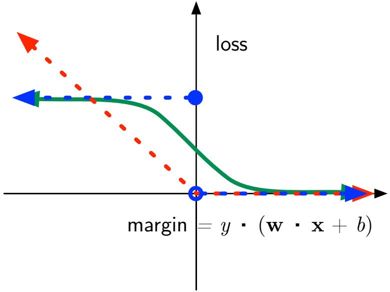


Squash (Sigmoid) Loss?



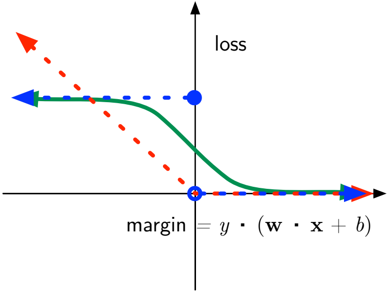
Different Kinds of Objective Functions

- Continuous (perceptron loss, squash loss) vs. discrete (zero-one loss)



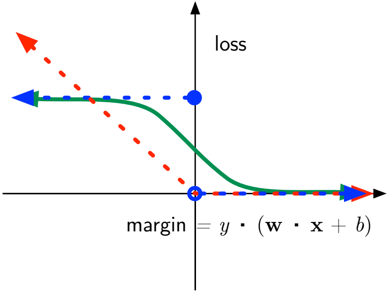
Different Kinds of Objective Functions

- Continuous (perceptron loss, squash loss) vs. discrete (zero-one loss)
- Convex (perceptron loss) vs. nonconvex (zero-one loss, squash loss)



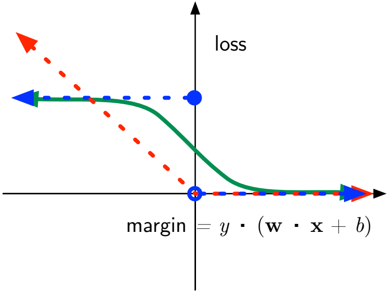
Different Kinds of Objective Functions

- Continuous (perceptron loss, squash loss) vs. discrete (zero-one loss)
- Convex (perceptron loss) vs. nonconvex (zero-one loss, squash loss)
- Differentiable (squash loss) vs. nondifferentiable (zero-one loss, perceptron loss)



Different Kinds of Objective Functions

- Continuous (perceptron loss, squash loss) vs. discrete (zero-one loss) (The sum of two continuous functions is also continuous.)
- Convex (perceptron loss) vs. nonconvex (zero-one loss, squash loss) (The sum of two convex functions is also convex.)
- Differentiable (squash loss) vs. nondifferentiable (zero-one loss, perceptron loss) (The sum of two differentiable functions is also differentiable.)



Regularization

Choose your loss function L. To fit the training data:

$$\min_{\mathbf{w},b} \frac{1}{N} \sum_{n=1}^{N} L\left(y_n \cdot (\mathbf{w} \cdot \mathbf{x}_n + b)\right)$$

Regularization

Choose your loss function L. To fit the training data:

$$\min_{\mathbf{w},b} \frac{1}{N} \sum_{n=1}^{N} L\left(y_n \cdot (\mathbf{w} \cdot \mathbf{x}_n + b)\right) + R(\mathbf{w}, b)$$

Regularization: add a penalty to the objective function to encourage generalization.

Regularization

Choose your loss function L. To fit the training data:

$$\min_{\mathbf{w},b} \frac{1}{N} \sum_{n=1}^{N} L\left(y_n \cdot (\mathbf{w} \cdot \mathbf{x}_n + b)\right) + R(\mathbf{w}, b)$$

Regularization: add a penalty to the objective function to encourage generalization.

Most common: $R(\mathbf{w}, b) = \lambda \|\mathbf{w}\|_2^2$.

Regularization

Choose your loss function L. To fit the training data:

$$\min_{\mathbf{w},b} \frac{1}{N} \sum_{n=1}^{N} L\left(y_n \cdot (\mathbf{w} \cdot \mathbf{x}_n + b)\right) + R(\mathbf{w}, b)$$

Regularization: add a penalty to the objective function to encourage generalization.

Most common: $R(\mathbf{w}, b) = \lambda \|\mathbf{w}\|_2^2$.

- Note that this term is convex and differentiable.

Your new hobby: blindfolded mountain escape

Convex Optimization 101

Assume we are minimizing a function $F : \mathbb{R}^d \rightarrow \mathbb{R}$ that is continuous, convex, and differentiable with respect to its input, $\mathbf{z}$.

$$\min_{\mathbf{z}} F(\mathbf{z})$$

At a given point $\mathbf{z}_0$, the direction of steepest descent is the negative gradient:

$$-\mathbf{g}(\mathbf{z}_0) = -\nabla_{\mathbf{z}} F(\mathbf{z}_0) = -\begin{bmatrix} \frac{\partial F}{\partial \mathbf{z}[1]}(\mathbf{z}_0) \\ \frac{\partial F}{\partial \mathbf{z}[2]}(\mathbf{z}_0) \\ \vdots \frac{\partial F}{\partial \mathbf{z}[d]}(\mathbf{z}_0) \end{bmatrix}$$

Note that $\mathbf{g} : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

Gradient Descent

Data: function $F : \mathbb{R}^d \to \mathbb{R}$, number of iterations K, step sizes $\langle \eta^{(1)}, \ldots, \eta^{(K)} \rangle$
Result: $\mathbf{z} \in \mathbb{R}^d$
initialize: $\mathbf{z}^{(0)} = \mathbf{0}$;
for $k \in \{1, \ldots, K\}$ **do**
    $\mathbf{g}^{(k)} = \nabla_{\mathbf{z}} F(\mathbf{z}^{(k-1)})$;
    $\mathbf{z}^{(k)} = \mathbf{z}^{(k-1)} - \eta^{(k)} \cdot \mathbf{g}^{(k)}$;
end
return $\mathbf{z}^{(K)}$;

Algorithm 1: GradientDescent

Gradient Descent