Review: Supervised Learning

CS 6355: Structured Prediction

Previous lecture

- A broad overview of structured prediction
- The different aspects of the area
  - Basically the syllabus of the class
- *Questions?*

Supervised learning, Binary classification

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression

Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression

Supervised learning: General setting

- Given: Training examples of the form <$\mathbf{x}$, $f(\mathbf{x})$>
  - The function f is an unknown function

- The input $\mathbf{x}$ is represented in a *feature space*
  - Typically $\mathbf{x} \in \{0,1\}^n$ or $\mathbf{x} \in \Re^n$
- For a training example $\mathbf{x}$, the value of $f(\mathbf{x})$ is called its *label*

- *Goal: Find a good approximation for f*
- Different kinds of problems
  - Binary classification: $f(\mathbf{x}) \in \{-1, 1\}$
  - Multiclass classification*:* $f(\mathbf{x}) \in \{1, 2, \cdots, k\}$
  - Regression: $f(\mathbf{x}) \in \Re$

Nature of applications

- There is no human expert
  - Eg: Identify DNA binding sites
- Humans can perform a task, but can’t describe how they do it
  - Eg: Object detection in images
- The desired function is hard to obtain in closed form
  - Eg: Stock market

Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression

Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label $y \in \{-1, 1\}$ For now

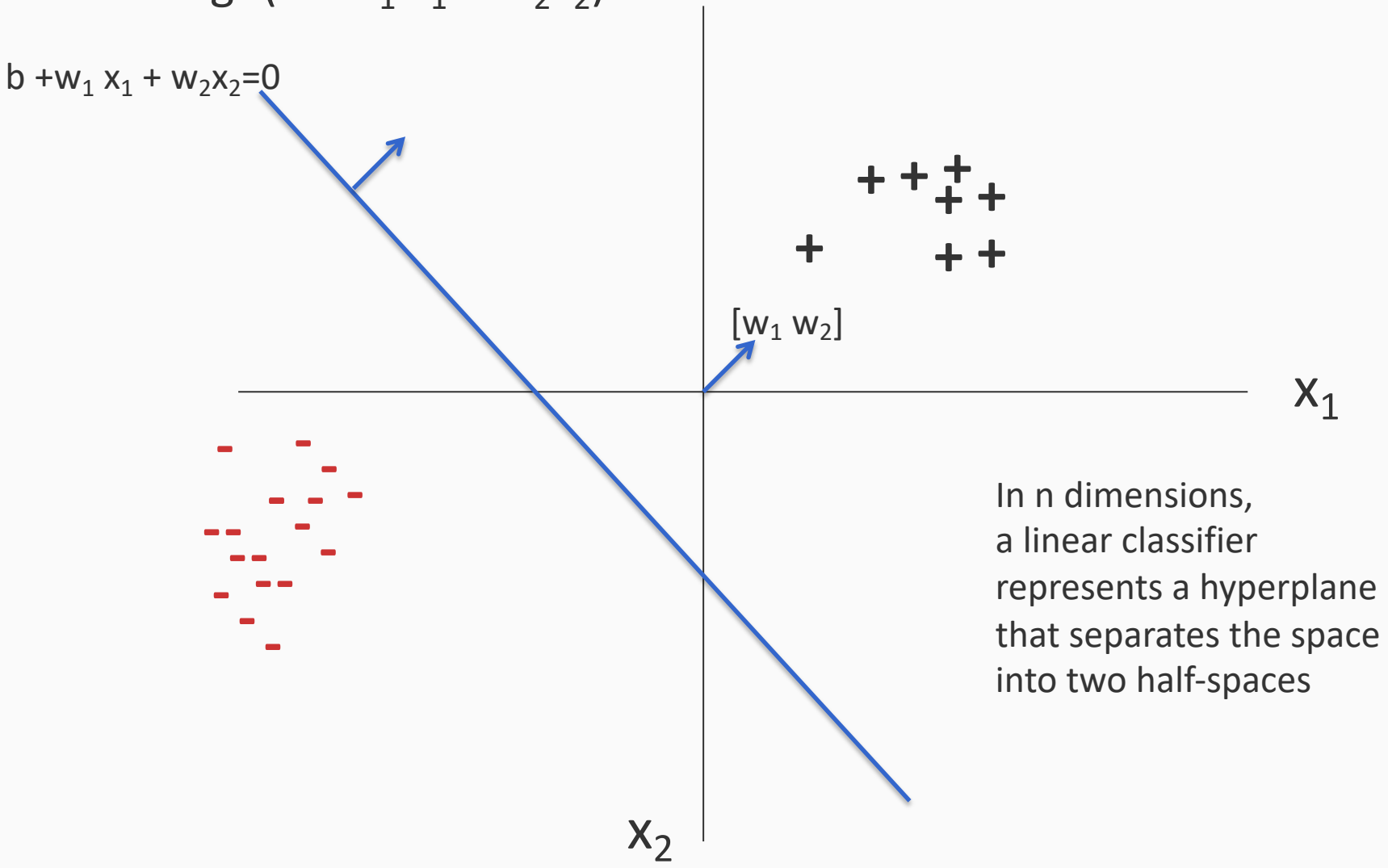
- Linear threshold units classify an example $\boldsymbol{x}$ using the classification rule

$$\text{sgn}(b + \boldsymbol{w}^T \boldsymbol{x}) = \text{sgn}(b + \sum_i w_i x_i)$$

  - $b + \boldsymbol{w}^T \boldsymbol{x} \geq 0$) Predict y = 1
  - $b + \boldsymbol{w}^T \boldsymbol{x} < 0$) Predict y = -1

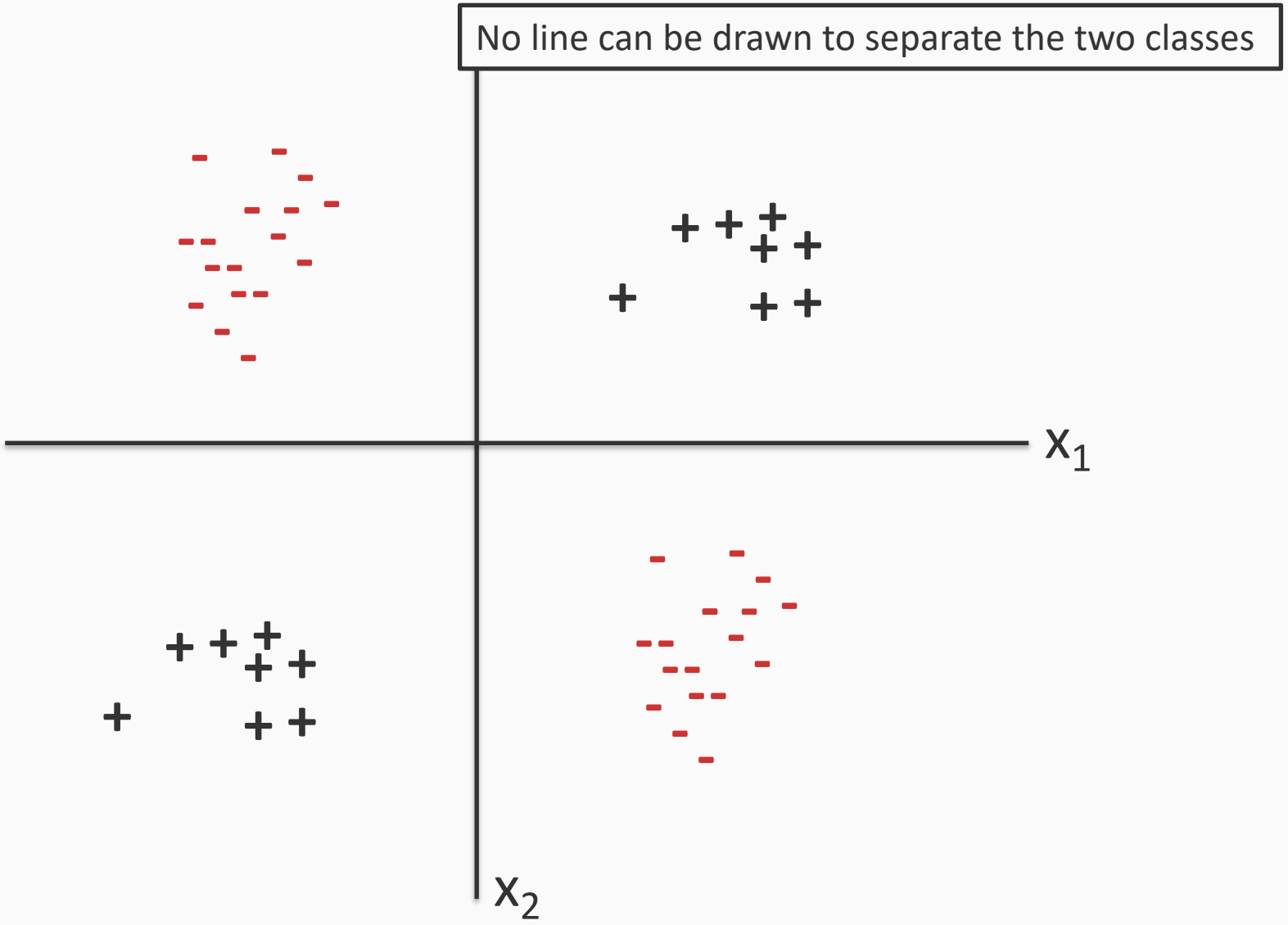
The geometry of a linear classifier

$\text{sgn}(b + w_1 x_1 + w_2 x_2)$



In n dimensions,
a linear classifier
represents a hyperplane
that separates the space
into two half-spaces

XOR is not linearly separable



Even these functions can be *made* linear

These points are not separable in 1-dimension by a line

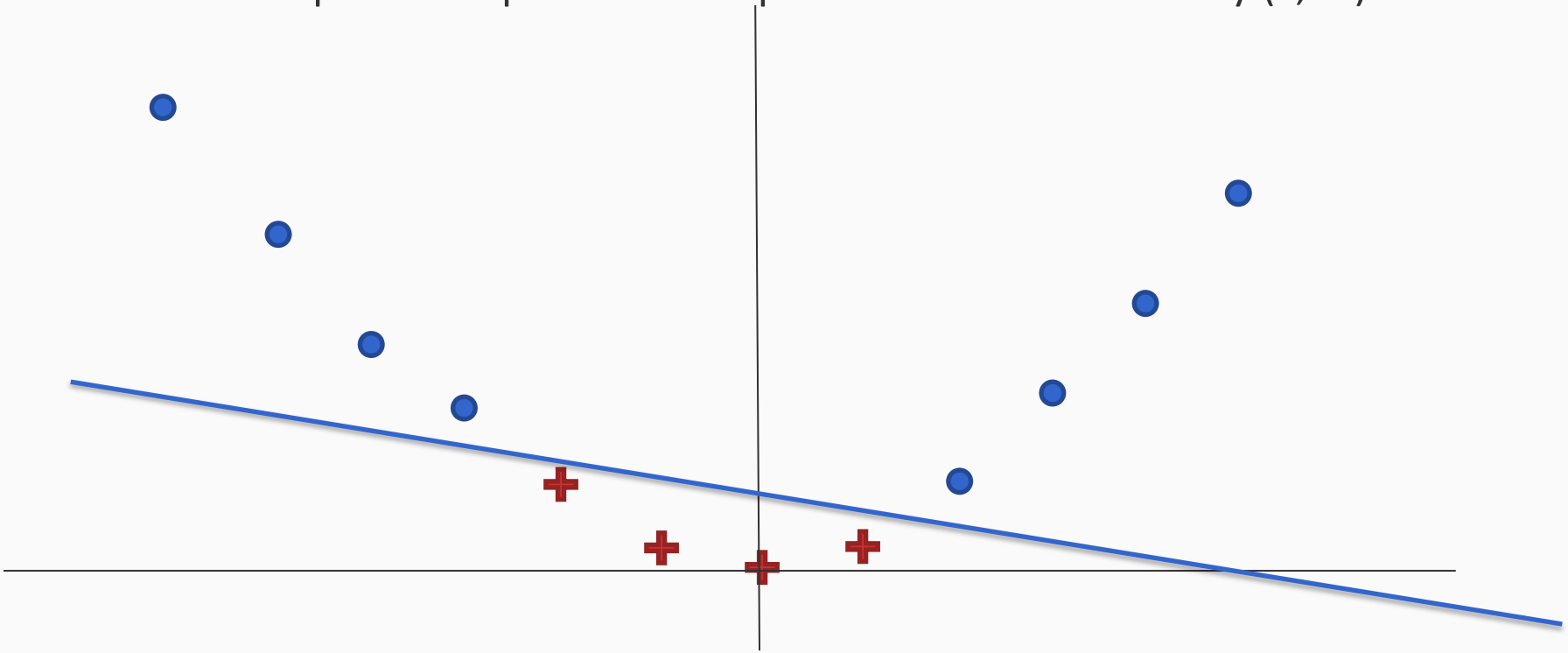
What is a one-dimensional line, by the way?

The trick: Change the representation

Even these functions can be *made* linear

The trick: Use feature *conjunctions*

Transform points: Represent each point x in 2 dimensions by (x, x^2)



Now the data is linearly separable in this space!

Linear classifiers are an expressive hypothesis class

- Many functions are linear
  - Conjunctions, disjunctions
  - At least m-of-n functions
- Often a good guess for a hypothesis space
  - *If we know a good feature representation*
- Some functions are not linear
  - The XOR function
  - Non-trivial Boolean functions

We will see in the coming weeks that structured predictors can also be defined via linear functions.

Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression

The Perceptron algorithm

- Rosenblatt 1958
- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one
- An online algorithm
  - Processes one example at a time
- Several variants exist

The algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^d, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^d$
2. For epoch in $1 \cdots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
            - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Prediction on a new example with features $\mathbf{x}$: $\text{sgn}(\mathbf{w}^T \mathbf{x})$

The algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^d, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^d$
2. For epoch in $1 \cdots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
        - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

T is a hyper-parameter to the algorithm

Another way of writing that there is an error

Prediction on a new example with features $\mathbf{x}$: $\text{sgn}(\mathbf{w}^T \mathbf{x})$

Convergence theorem

If there exist a set of weights that are consistent with the data (i.e. the data is linearly separable), the perceptron algorithm will converge after a finite number of updates.

- [Novikoff 1962]

Beyond the separable case

- The good news
  - Perceptron makes no assumption about data distribution
  - Even adversarial
  - After a fixed number of mistakes, you are done. Don't even need to see any more data

- The bad news: Real world is not linearly separable
  - Can't expect to *never* make mistakes again
  - What can we do: more features, try to be linearly separable if you can

Variants of the algorithm

- The original version: Return the final weight vector

- Averaged perceptron
  - Returns the average weight vector from the entire training time (i.e longer surviving weight vectors get more say)
  - Widely used
  - A practical approximation of the Voted Perceptron

Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
    1. The general idea
    2. Stochastic gradient descent
    3. Loss functions
5. Support vector machines
6. Logistic Regression

Learning as loss minimization

- Collect some annotated data. More is generally better
- Pick a hypothesis class (also called model)
  - Eg: linear classifiers, deep neural networks
  - Also, decide on how to impose a preference over hypotheses
- Choose a loss function
  - Eg: negative log-likelihood, hinge loss
  - Decide on how to penalize incorrect decisions
- Minimize the expected loss
  - Eg: Set derivative to zero and solve on paper, typically a more complex algorithm

Learning as loss minimization

- The setup
  - Examples **x** drawn from a fixed, unknown distribution D
  - Hidden oracle classifier f labels examples
  - We wish to find a hypothesis h that mimics f

Learning as loss minimization

- The setup
  - Examples **x** drawn from a fixed, unknown distribution D
  - Hidden oracle classifier f labels examples
  - We wish to find a hypothesis h that mimics f
- The ideal situation
  - Define a function L that penalizes bad hypotheses
  - **Learning:** Pick a function h 2 H to minimize expected loss

$$\min_{h \in H} E_{\mathbf{x} \sim D} \left[L\left(h(\mathbf{x}), f(\mathbf{x})\right) \right]$$

But distribution D is unknown

Learning as loss minimization

- The setup
  - Examples **x** drawn from a fixed, unknown distribution D
  - Hidden oracle classifier f labels examples
  - We wish to find a hypothesis h that mimics f
- The ideal situation
  - Define a function L that penalizes bad hypotheses
  - **Learning:** Pick a function h 2 H to minimize expected loss

$$\min_{h \in H} E_{\mathbf{x} \sim D} \left[L\left(h(\mathbf{x}), f(\mathbf{x})\right) \right]$$

But distribution D is unknown

- Instead, minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_{i} L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Empirical loss minimization

Learning = minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i))$$

Is there a problem here? Overfitting!

We need something that biases the learner towards simpler hypotheses

- Achieved using a regularizer, which penalizes complex hypotheses
- Capacity control for better generalization

Regularized loss minimization

- Learning: $\min_{h \in \mathrm{H}} \mathrm{regularizer(w)} + \mathrm{C} \frac{1}{m} \sum_i L(h(x_i), y_i)$
- With L2 regularization: $\min_w \frac{1}{2} w^T w + C \sum_i L(F(x_i, w), y_i)$

Regularized loss minimization

- Learning: $\min_{h \in \mathrm{H}} \mathrm{regularizer(w)} + \mathrm{C} \frac{1}{m} \sum_i L(h(x_i), y_i)$
- With L2 regularization: $\min_w \frac{1}{2} w^T w + C \sum_i L(F(x_i, w), y_i)$
- What is a loss function?
  - Loss functions should penalize mistakes
  - We are minimizing average loss over the training data

How do we train in such a regime?

- Suppose we have a predictor F that maps inputs x to a score F(x, w) that is thresholded to get a label
  - Here w are the parameters that define the function
  - Say F is a differentiable function

- How do we use a labeled training set to learn the weights i.e. solve this minimization problem?

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

How do we train in such a regime?

- Suppose we have a predictor F that maps inputs x to a score F(x, w) that is thresholded to get a label
  - Here w are the parameters that define the function
  - Say F is a differentiable function

- How do we use a labeled training set to learn the weights i.e. solve this minimization problem?

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

- We could compute the gradient of the loss and descend along that direction to minimize

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:

3. Return w

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
    1. Shuffle the training set

3. Return w

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
    1. Shuffle the training set
    2. For each training example $(\mathbf{x}_i, y_i) \in S$:

3. Return w

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
    1. Shuffle the training set
    2. For each training example $(\mathbf{x}_i, y_i) \in S$:
        - Treat this example as the entire dataset
          Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
3. Return w

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
    1. Shuffle the training set
    2. For each training example $(\mathbf{x}_i, y_i) \in S$:
        - Treat this example as the entire dataset
          Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
        - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in S$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

γ_t: learning rate, many tweaks possible

$$\min_{w} \sum_{i} L(F(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in S$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(F(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

The objective is not convex.
Initialization can be important

γ_t: learning rate,
many tweaks possible

A more general form

Suppose we want to minimize a function that is the sum of other functions

$$f(x) = \sum_{i=1}^{n} f_i(x)$$

- Initialize x
- Loop till convergence:
  - Pick i randomly from $\{1, 2, \cdots, n\}$
  - Update $x \leftarrow x - stepsize \cdot \nabla f_i(x)$
- Return x

In practice…

- There are many variants of this idea
- Several named learning algorithms
  - AdaGrad, AdaDelta, RMSProp, Adam
- But the key components are the same. We need to…
  1. …sample a tiny subset of the data at each step
  2. …compute the gradient of the loss using this subset
  3. …take a step in the negative direction of the gradient

Standard loss functions

We need to think about the problem we have at hand

Is it a…

1. Binary classification problem?
2. Regression problem?
3. Multi-class classification problem?
4. Or something else?

Each case is naturally paired with a different loss function

The ideal case for binary classification: The 0-1 loss

Penalize classification mistakes between true label y and prediction y'

$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } y \neq y', \\ 0 & \text{if } y = y'. \end{cases}$$

More generally, suppose we have a prediction function of the form $\text{sgn}(F(x, w))$

- Note that F need not be linear

$$L_{0-1}(y, y') = \begin{cases} 1 & \text{if } yF(x, w) \leq 0, \\ 0 & \text{if } yF(x, w) > 0. \end{cases}$$

Minimizing 0-1 loss is intractable. Need surrogates

$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

The loss function zoo

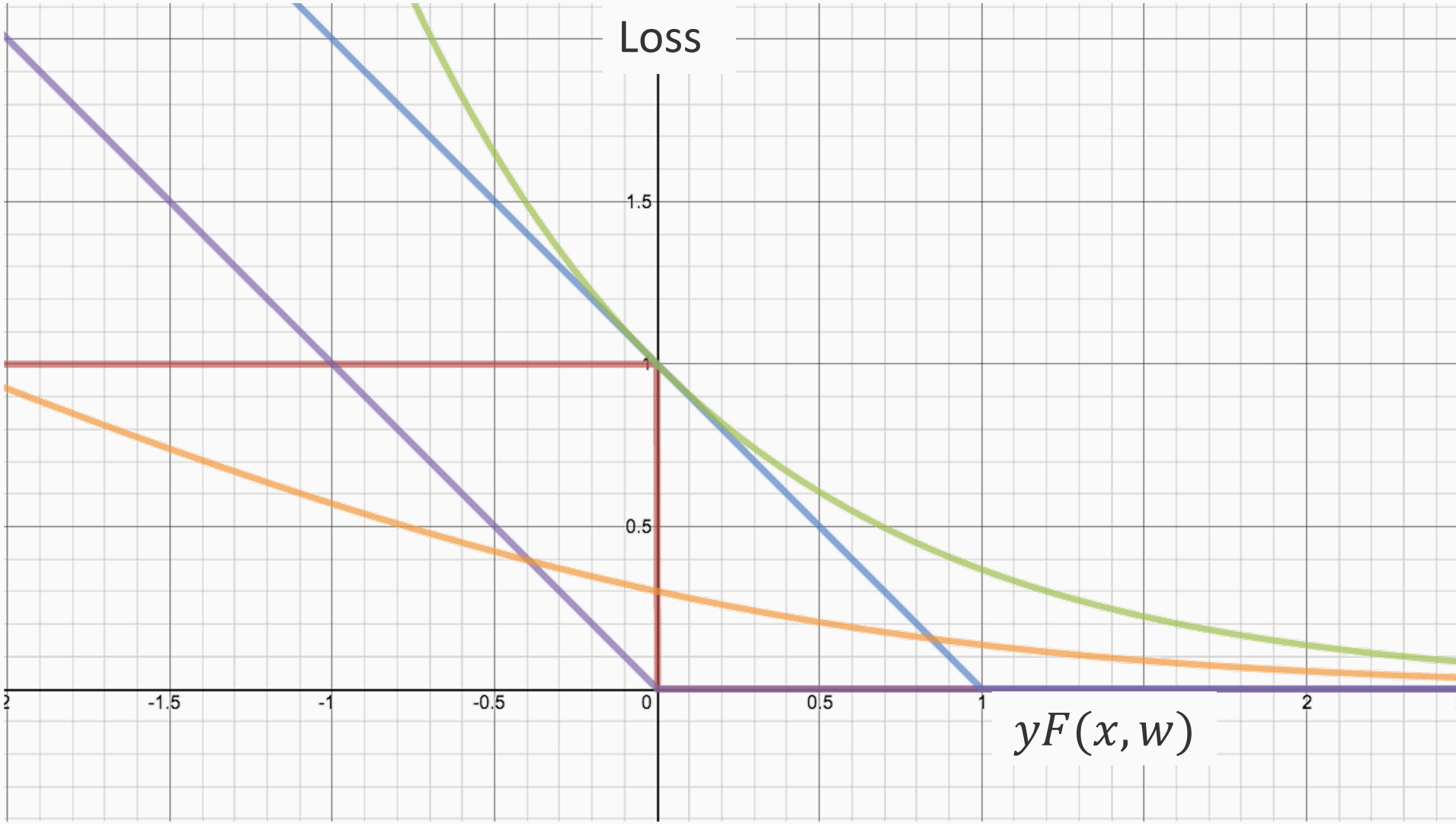
For binary classification

Many loss functions exist

Perceptron	$L_{Perceptron}(y, x, w) = \max(0, -yF(x, w))$
Hinge (SVM)	$L_{Hinge}(y, x, w) = \max(0, 1 - yF(x, w))$
Exponential (Adaboost)	$L_{Exponential}(y, x, w) = e^{-yF(x,w)}$
Logistic loss	$L_{Logistic}(y, x, w) = \log(1 + e^{-yF(x,w))})$

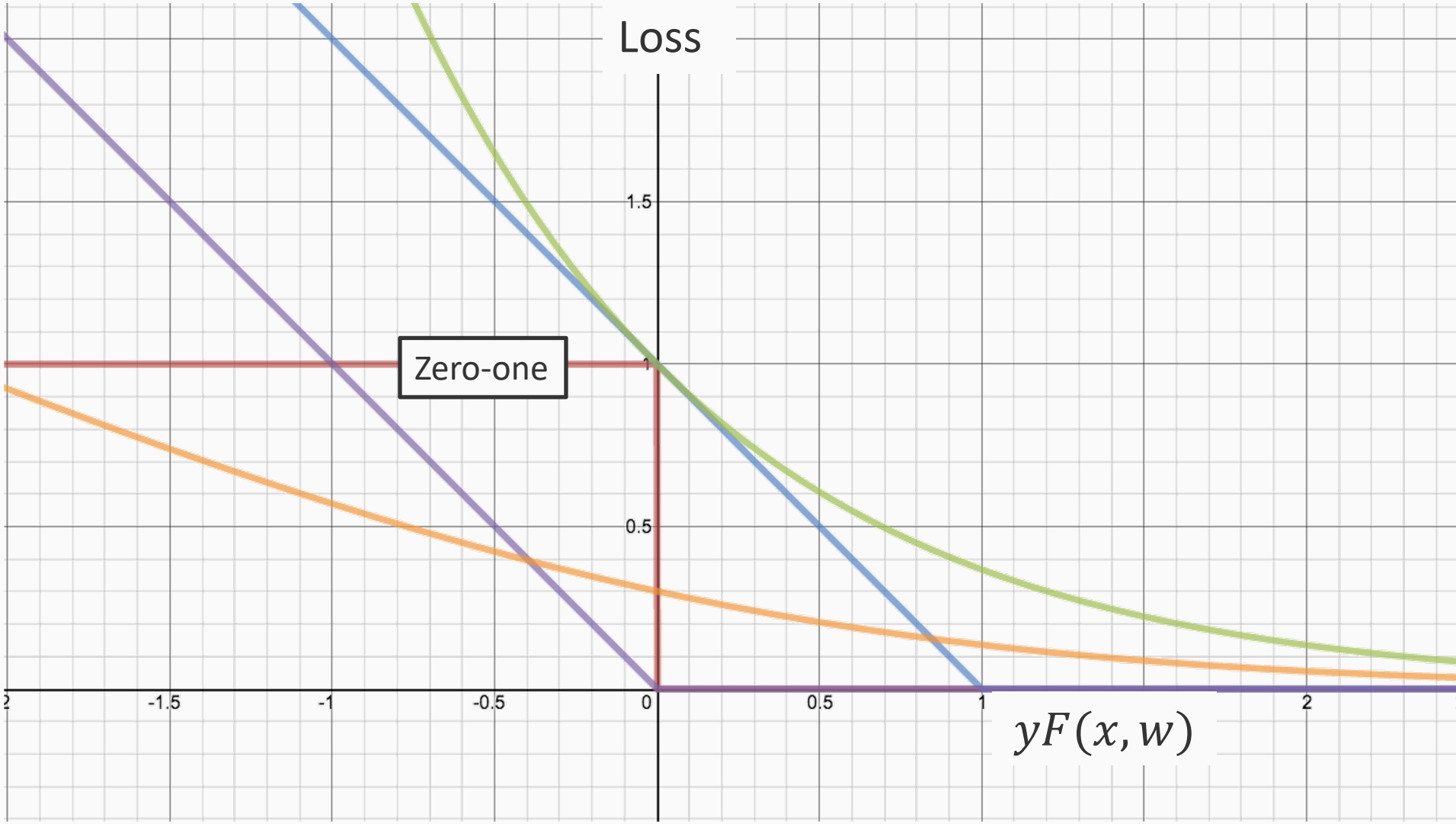
$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

The loss function zoo



$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

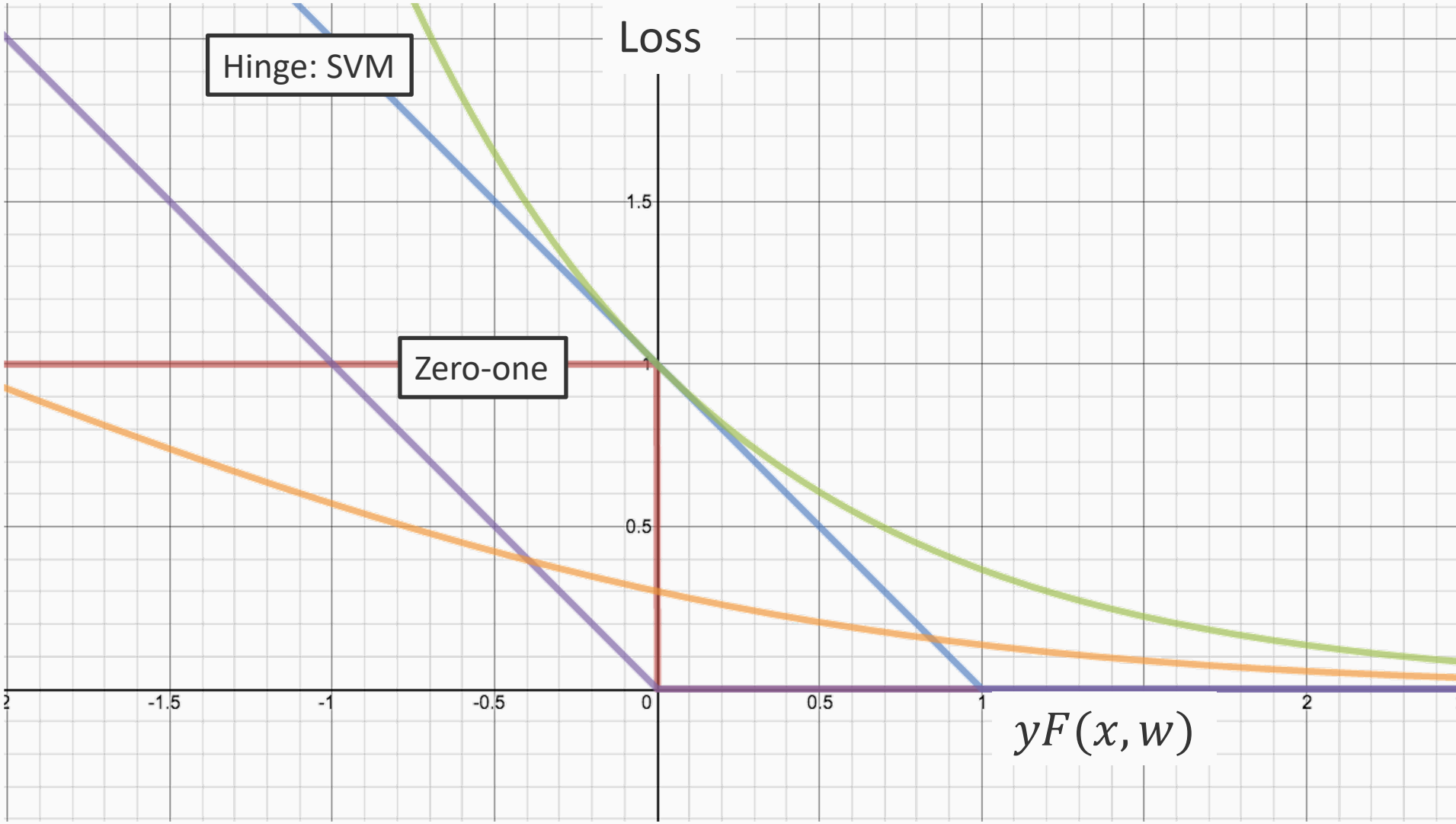
The loss function zoo



$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

The loss function zoo

$$L_{Hinge}(y, x, w) = \max(0, 1 - yF(x, w))$$

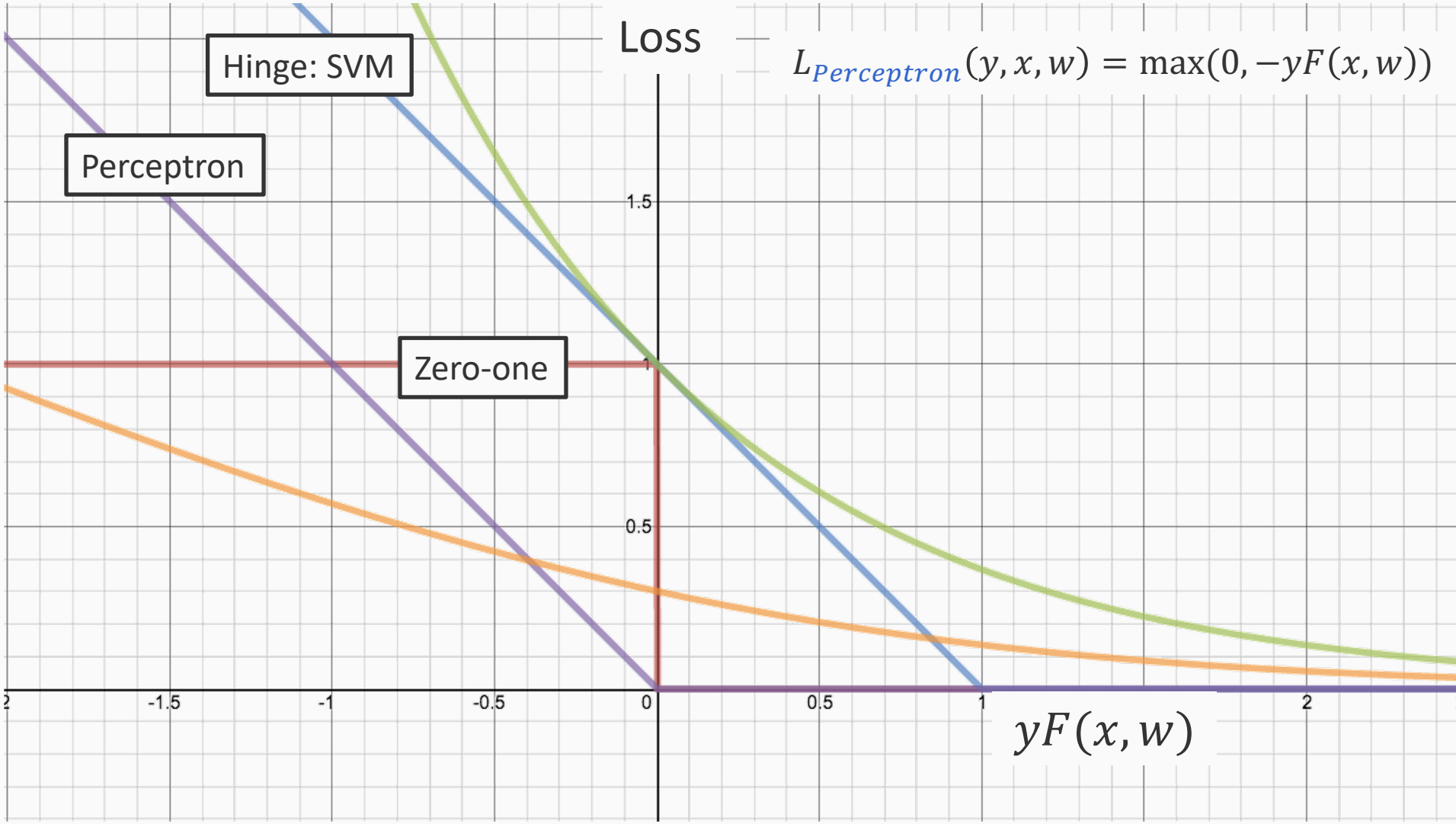


The loss function zoo

$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

$$L_{Hinge}(y, x, w) = \max(0, 1 - yF(x, w))$$

$$L_{Perceptron}(y, x, w) = \max(0, -yF(x, w))$$



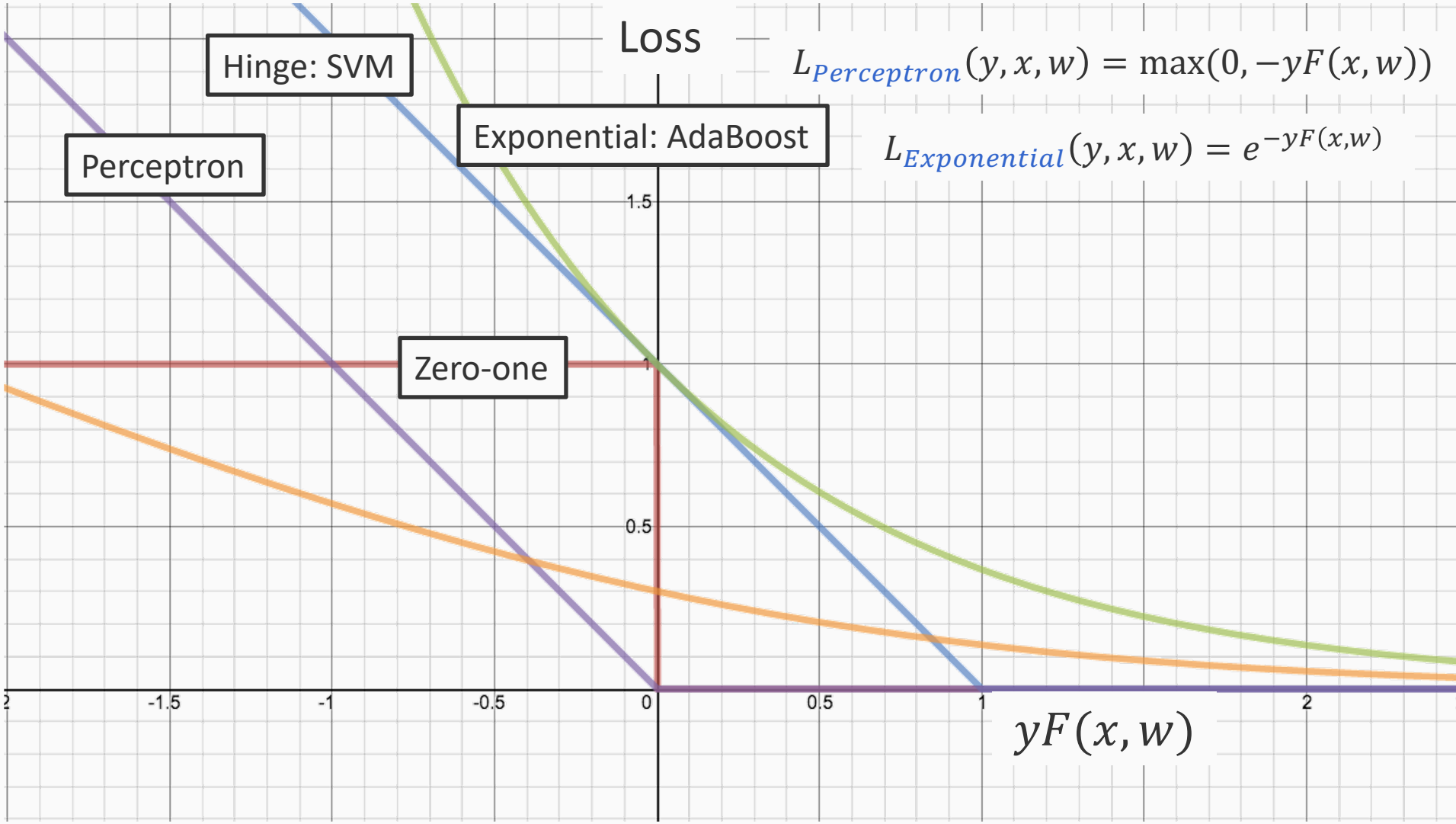
The loss function zoo

$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

$$L_{Hinge}(y, x, w) = \max(0, 1 - yF(x, w))$$

$$L_{Perceptron}(y, x, w) = \max(0, -yF(x, w))$$

$$L_{Exponential}(y, x, w) = e^{-yF(x,w)}$$



The loss function zoo

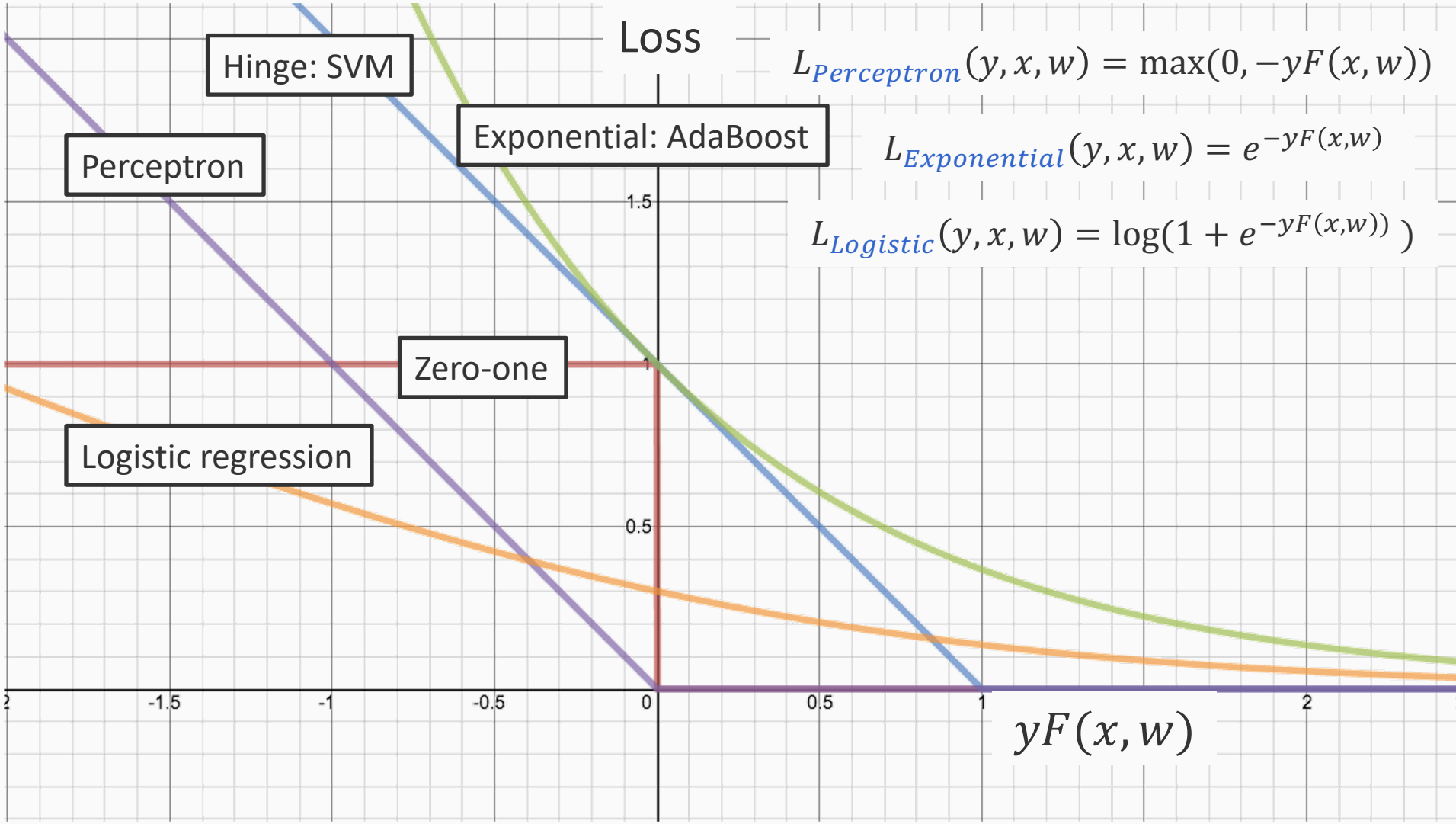
$$\min_{h \in H} \text{regularizer}(w) + C \frac{1}{m} \sum_i L(F(x_i, w), y_i)$$

$$L_{Hinge}(y, x, w) = \max(0, 1 - yF(x, w))$$

$$L_{Perceptron}(y, x, w) = \max(0, -yF(x, w))$$

$$L_{Exponential}(y, x, w) = e^{-yF(x,w)}$$

$$L_{Logistic}(y, x, w) = \log(1 + e^{-yF(x,w)})$$



What if we have a regression task

Real valued outputs

- That is, our model is a function F(x, w) that maps inputs x to a real number
- Parameterized by w
- The ground truth y is also a real number

A natural loss function for this situation is the squared loss

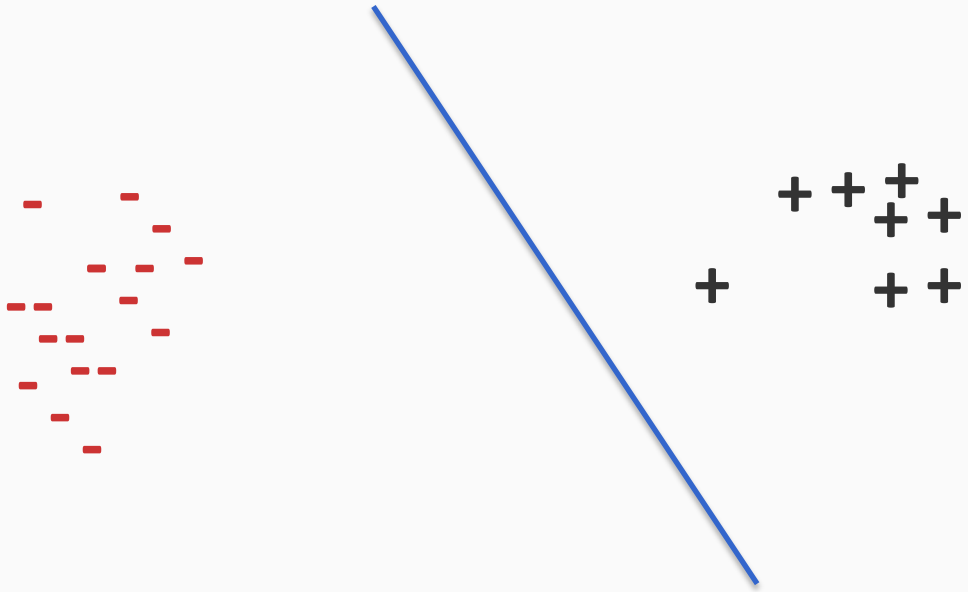
$$L(x, y, w) = \big(y - F(x, w)\big)^2$$

Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression

Margin

The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Learning strategy

Find the linear separator that maximizes the margin

Maximizing margin and minimizing loss

Find the linear separator that maximizes the margin

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

Maximize margin

Penalty for the prediction:
The Hinge loss

SVM objective function

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

Regularization term:

- Maximize the margin
- Imposes a preference over the hypothesis space and pushes for better generalization

Empirical Loss:

- Hinge loss
- Penalizes weight vectors that make mistakes

A hyper-parameter that controls the tradeoff between a large margin and a small hinge-loss

Where are we?

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Learning as optimization
5. Support vector machines
6. Logistic Regression

Regularized loss minimization: Logistic regression

- Learning: $\min_{f \in H} \ \text{regularizer}(f) + C \sum_i L(y_i, f(\mathbf{x}_i))$
- With linear classifiers: $\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \sum_i L(y_i, \mathbf{x}_i, \mathbf{w})$
- SVM uses the hinge loss
- Another loss function: The logistic loss

$$L_{logistic}(y, \mathbf{x}, \mathbf{w}) = \log(1 + e^{-y\mathbf{w}^T\mathbf{x}})$$

The probabilistic interpretation

Suppose we believe that the labels are distributed as follows given the input:

$$P(y=1|\mathbf{x},\mathbf{w}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}} = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$P(y=-1|\mathbf{x},\mathbf{w}) = \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}}$$

$$\Big\} \quad P(y|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp\left(-y\mathbf{w}^T\mathbf{x}\right)}$$

Predict label = 1 if P(1 | **x**, **w**) > P(-1 | **x**, **w**)

- Equivalent to predicting 1 if $\mathbf{w}^T\mathbf{x} > 0$

The probabilistic interpretation

Suppose we believe that the labels are distributed as follows given the input:

$$P(y=1|\mathbf{x},\mathbf{w}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}} = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$P(y=-1|\mathbf{x},\mathbf{w}) = \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}}$$

$$\Big\} \quad P(y|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp\left(-y\mathbf{w}^T\mathbf{x}\right)}$$

The log-likelihood of seeing a dataset $D = \{(\mathbf{x}_i, y_i)\}$ if the true weight vector was $\mathbf{w}$:

$$\log P(D|\mathbf{w}) = -\sum_i \log\left(1+\exp(-y\mathbf{w}^T\mathbf{x}\right)$$

Regularized logistic regression

What is the probability of weights **w** being the true ones for a dataset D = {<$\mathbf{x}_i$, y_i>}?

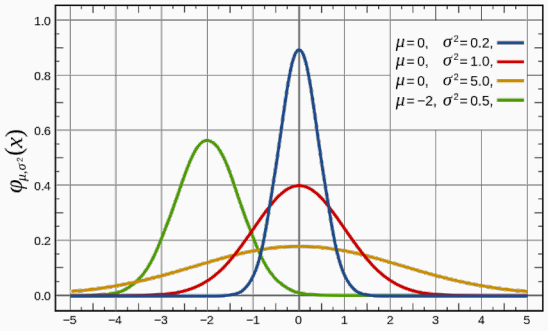
$$P(\boldsymbol{w} \mid D) \propto P(\mathbf{w}, D) = P(\ D \mid \boldsymbol{w})P(\ \boldsymbol{w})$$

Prior distribution over the weight vectors

A prior balances the tradeoff between the likelihood of the data and existing belief about the parameters

- Suppose each weight w_i is drawn independently from the normal distribution centered at zero with variance σ^2
  - Bias towards smaller weights

$$P(w_i) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right)$$



Source: Wikipedia

- Probability of the entire weight vector:

$$\log P(\mathbf{w}) = -\frac{1}{2\sigma^2}\mathbf{w}^T\mathbf{w} + \text{constant terms}$$

Regularized logistic regression

What is the probability of weights **w** being the true ones for a dataset D = {<$\mathbf{x}_i$, y_i>}?

$$P(\boldsymbol{w} \mid D) \propto P(\mathbf{w}, D) = P(\ D \mid \boldsymbol{w}\ )P(\ \boldsymbol{w})$$

Learning: Find weights by maximizing the posterior distribution $P(\ \mathbf{w} \mid D\ )$

$$-\log P(\ \mathbf{w} \mid D\ ) = \frac{1}{2\sigma^2}\mathbf{w}^T\mathbf{w} + \sum_i \log(1 + \exp(-y\mathbf{w}^T\mathbf{x})) + \text{constants}$$

Once again, regularized loss minimization! This is the Bayesian interpretation of regularization

Regularized loss minimization

Learning objective for both SVM & logistic regression:

"loss over training data + regularizer"

- Different loss functions
  - Hinge loss vs. logistic loss
- Same regularizer, but different interpretation
  - Margin vs prior
- Hyper-parameter controls tradeoff between the loss and regularizer
- Other regularizers/loss functions also possible

Questions?

Review of supervised binary classification

1. Supervised learning: The general setting
2. Linear classifiers
3. The Perceptron algorithm
4. Support vector machine
5. Learning as optimization
6. Logistic Regression

What if we have more than two labels?

Reading for next lecture:

- Erin L. Allwein, Robert E. Schapire, Yoram Singer, Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers, ICML 2000.