Review: Supervised Learning

CS 6956: Deep Learning for NLP

THE UNIVERSITY OF UTAH

Previous lecture

- A broad overview of the class
  - Overview of why deep learning and NLP form an interesting intersection
  - Basically the syllabus of the class
- *Questions?*

Supervised learning, Binary classification

1. Linear classifiers
2. The Perceptron algorithm
3. Support vector machines and logistic regression
4. Learning as optimization
5. Loss functions for various tasks

Supervised learning: General setting

- Given: Training examples of the form <**x**, *f*(**x**)>
  - The function f is an unknown function

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

- Goal: Find a good approximation for *f*
- Different kinds of problems
  - Binary classification: $f(\mathbf{x}) \in \{-1,1\}$
  - Multiclass classification: $f(\mathbf{x}) \in \{1, 2, 3, \cdots, K\}$
  - Regression: $f(\mathbf{x}) \in \Re$

Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label y ∈ {-1, 1}

For now

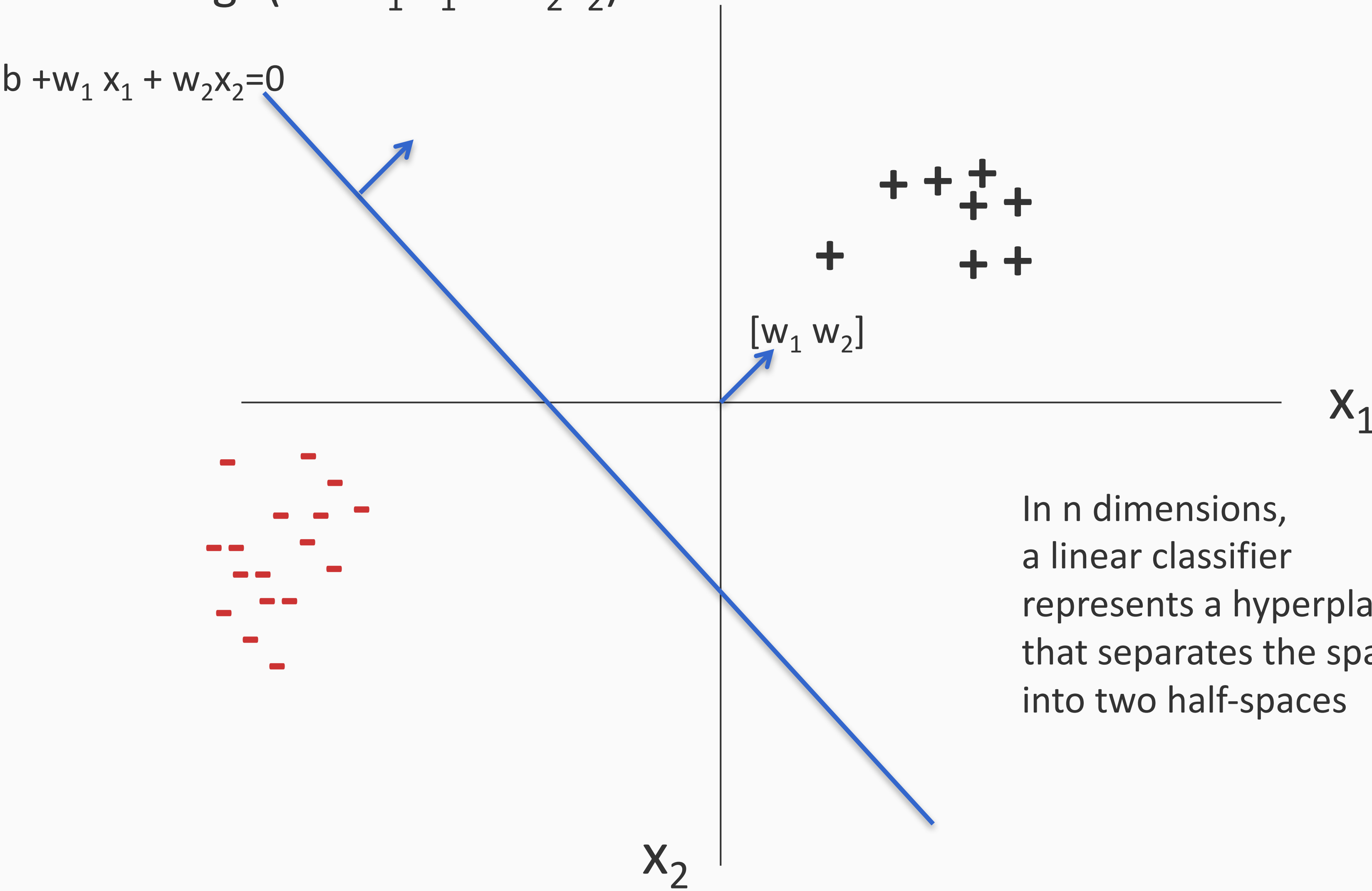
- Linear threshold units classify an example $\boldsymbol{x}$ using the classification rule

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

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

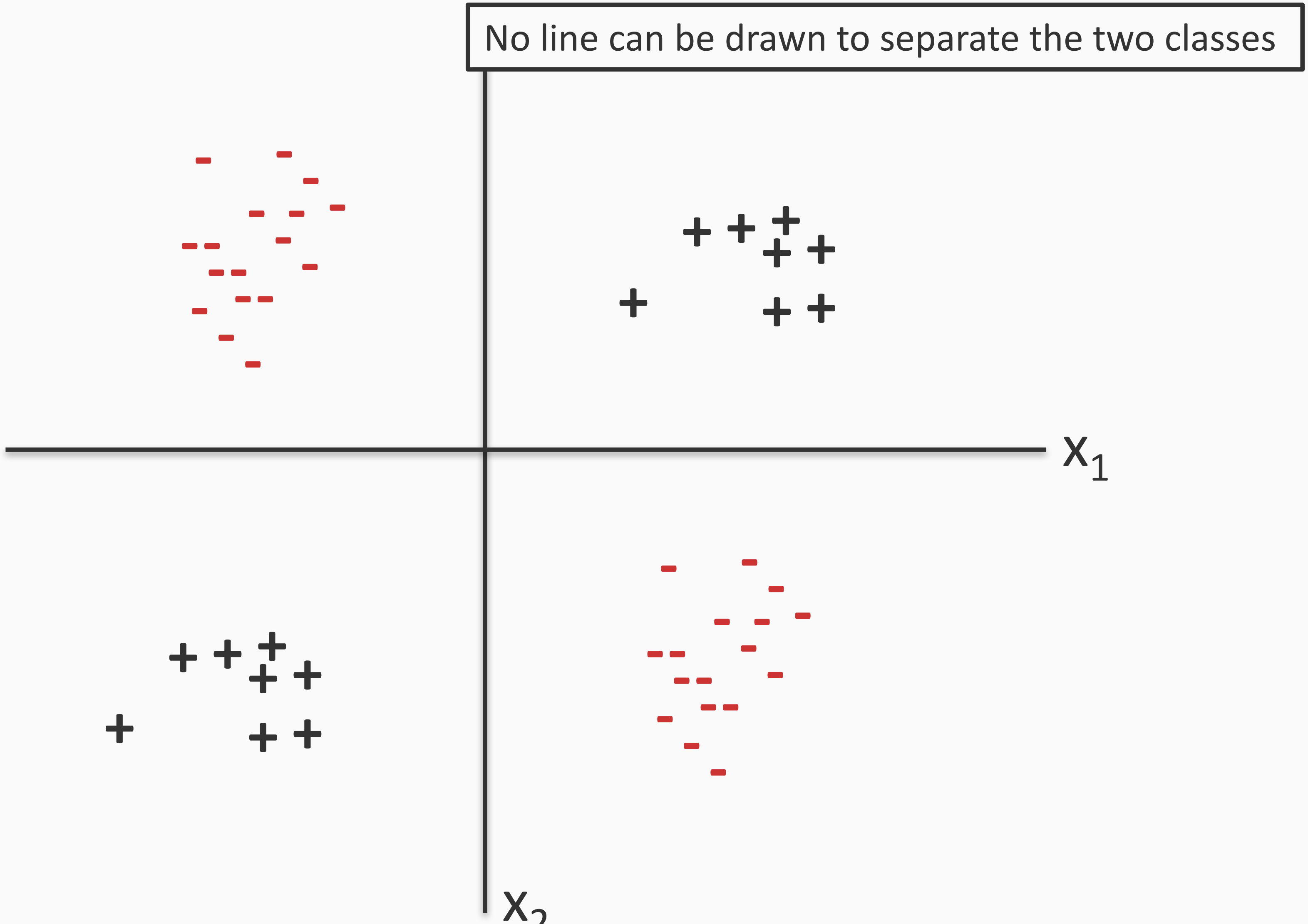
The geometry of a linear classifier

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



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

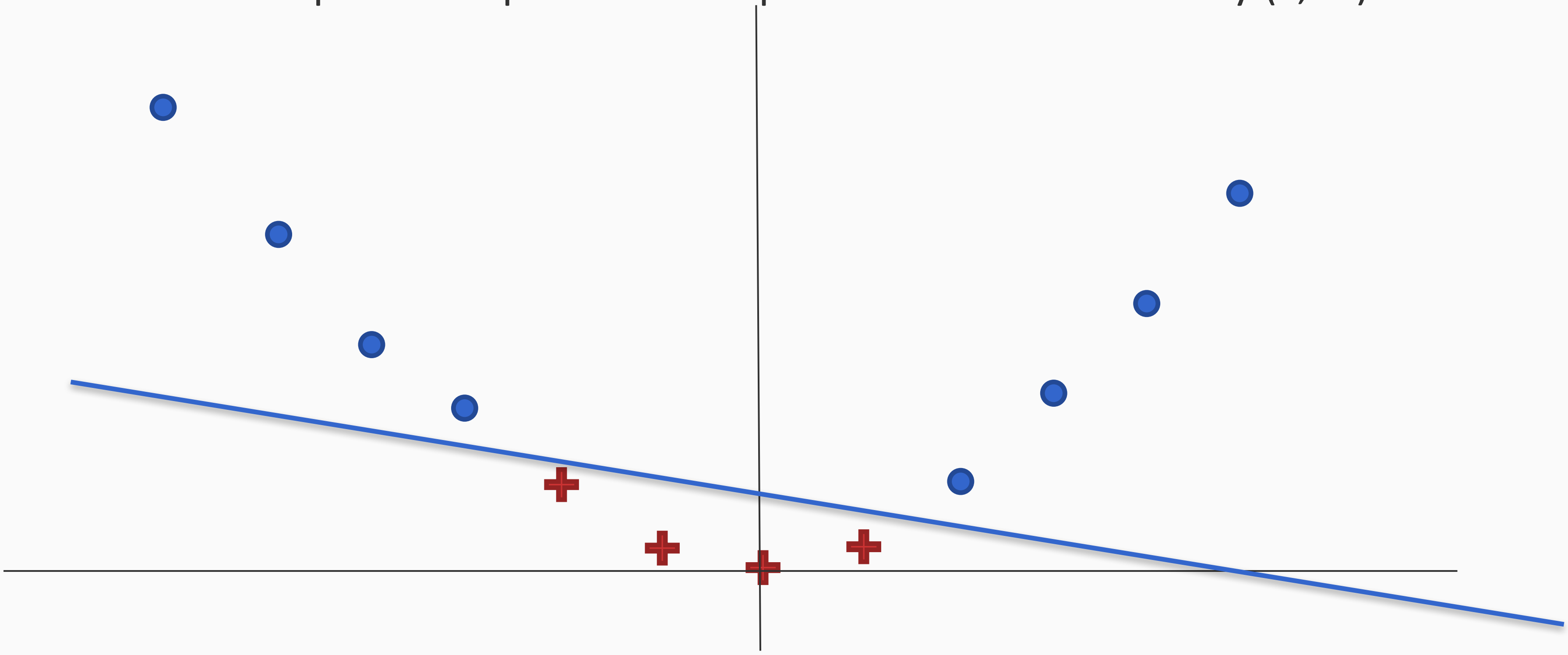
What is a one-dimensional line, by the way?

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

Where are we?

1. Linear classifiers
2. The Perceptron algorithm
3. Support vector machines and logistic regression
4. Learning as optimization
5. Loss functions for various tasks

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 = {($\mathbf{x}$,y)}, $x \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example ($\mathbf{x}$, y) $\in$ D:
      1. Predict $y' = \text{sgn}(\mathbf{w}^T\mathbf{x})$
      2. If $y \neq y'$, update $\mathbf{w} \leftarrow \mathbf{w} + y\,\mathbf{x}$
3. Return **w**

Prediction: $\text{sgn}(\mathbf{w}^T\mathbf{x})$

The algorithm

Given a training set D = {($\mathbf{x}$,y)}

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. For each training example ($\mathbf{x}$, y) $\in$ D:
        1. Predict y' = sgn($\mathbf{w}^T\mathbf{x}$)
        2. If y ≠ y', update $\mathbf{w} \leftarrow \mathbf{w} + y\,\mathbf{x}$
3. Return $\mathbf{w}$

T is a hyperparameter to the algorithm

In practice, good to shuffle D before the inner loop

Update only on an error. Perceptron is an mistake-driven algorithm.

Prediction: 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.

– [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

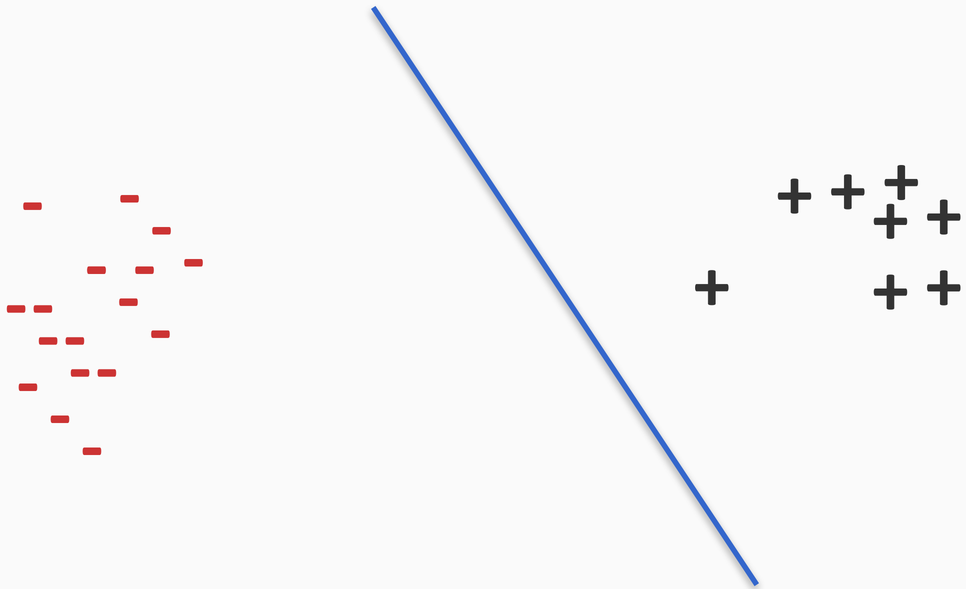
- The bad news: Real world is not linearly separable
  - Can’t expect to *never* make mistakes again
  - What can we do: more features, try to be linearly separable if you can

Where are we?

1. Linear classifiers
2. The Perceptron algorithm
3. Support vector machines and logistic regression
4. Learning as optimization
5. Loss functions for various tasks

Margin

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



Maximizing margin and minimizing loss

Find the linear separator that maximizes the margin

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

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
- Can be replaced with other regularization terms which impose other preferences

Empirical Loss:

- Hinge loss
- Penalizes weight vectors that make mistakes
- Can be replaced with other loss functions which impose other preferences

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
- Can be replaced with other regularization terms which impose other preferences

Empirical Loss:

- Hinge loss
- Penalizes weight vectors that make mistakes
- Can be replaced with other loss functions which impose other preferences

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

Regularized loss minimization: Logistic regression

- Learning: $\min_{f \in H} \ \mathrm{regularizer}(f) + C \sum_i L(y_i, f(\mathbf{x}_i))$
- 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:

$$\left.\begin{aligned} P(y=1|\mathbf{x},\mathbf{w}) &= \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}} = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}} \\ P(y=-1|\mathbf{x},\mathbf{w}) &= \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}} \end{aligned}\right\} \quad P(y|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp\left(-y\mathbf{w}^T\mathbf{x}\right)}$$

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

– Equivalent to predicting 1 if $\mathbf{w}^T\mathbf{x} \geq 0$

The probabilistic interpretation

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

$$\left.\begin{aligned} P(y=1|\mathbf{x},\mathbf{w}) &= \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}} = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}} \\ P(y=-1|\mathbf{x},\mathbf{w}) &= \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}} \end{aligned}\right\} \quad P(y|\mathbf{x},\mathbf{w}) = \frac{1}{1+\exp\left(-y\mathbf{w}^T\mathbf{x}\right)}$$

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

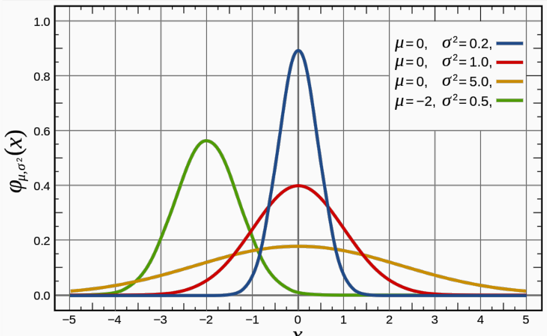
$$\log P(D|\mathbf{w}) = -\sum_i \log\left(1+\exp(-y\mathbf{w}^T\mathbf{x}\right)$$

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 a weight vector **w** being the true one for a dataset D = {<$\mathbf{x}_i$, y_i>}?

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

Regularized logistic regression

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

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

Learning: Find weight vector by maximizing the posterior distribution P(**w** | D)

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

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?

Where are we?

1. Linear classifiers
2. The Perceptron algorithm
3. Support vector machines and logistic regression
4. Learning as optimization
5. Loss functions for various tasks

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
  - Regularizer, but with neural networks, dropout is common
- Choose a loss function that penalizes incorrect preditions
  - Eg: negative log-likelihood, hinge loss
- Minimize the expected loss
  - Eg: Set derivative to zero, but more often procedural, usually gradient-based

Learning as loss minimization

- The setup
  - Examples $\mathbf{x}$ 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 $\mathbf{x}$ 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 $\in$ H to minimize expected loss

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

But distribution D is unknown

Learning as loss minimization

- The setup
  - Examples $\mathbf{x}$ 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 $\in$ 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

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(x_i), y_i)$$

Empirical loss minimization

Learning = minimize *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(x_i), y_i)$$

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 labels as F(x, w)
  - 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 labels as F(x, w)
  - 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 F and descend along that direction to minimize the loss

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

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

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

3. Return w

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

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

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

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

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

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

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

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

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

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

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

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

$$\min_{\boldsymbol{w}} \sum_i L(F(x_i, w), y_i)$$

Stochastic gradient descent

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

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

If the objective is not convex, initialization can be important

γ_t: learning rate, many tweaks possible

In practice…

- There are many variants of this idea

- We will encounter named learning algorithms as we go on
  - AdaGrad, 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

Where are we?

1. Linear classifiers
2. The Perceptron algorithm
3. Support vector machines and logistic regression
4. Learning as optimization
5. Loss functions for various tasks

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 $sgn(F(x, w))$

- 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

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

The loss function zoo

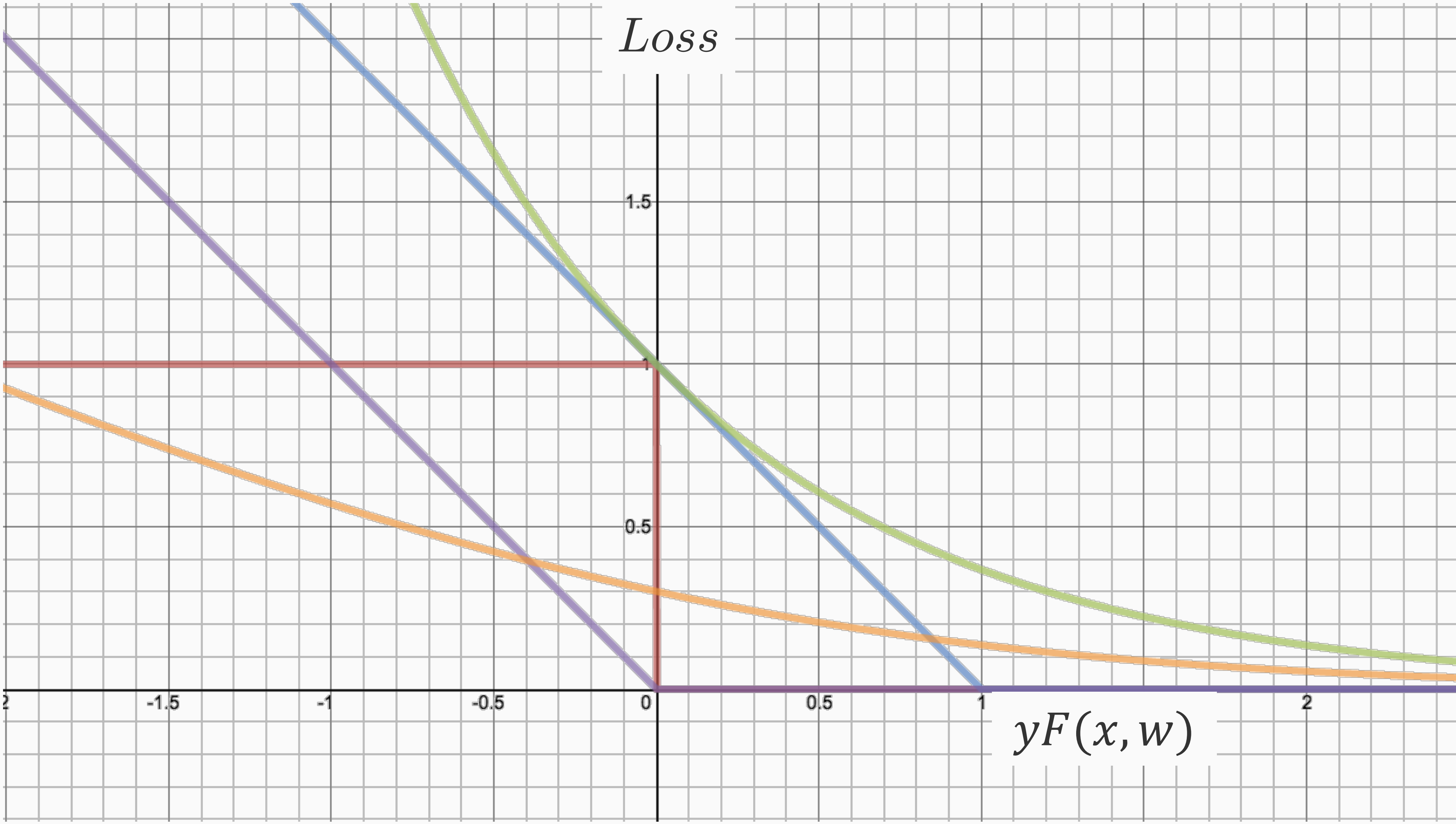
For binary classification

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

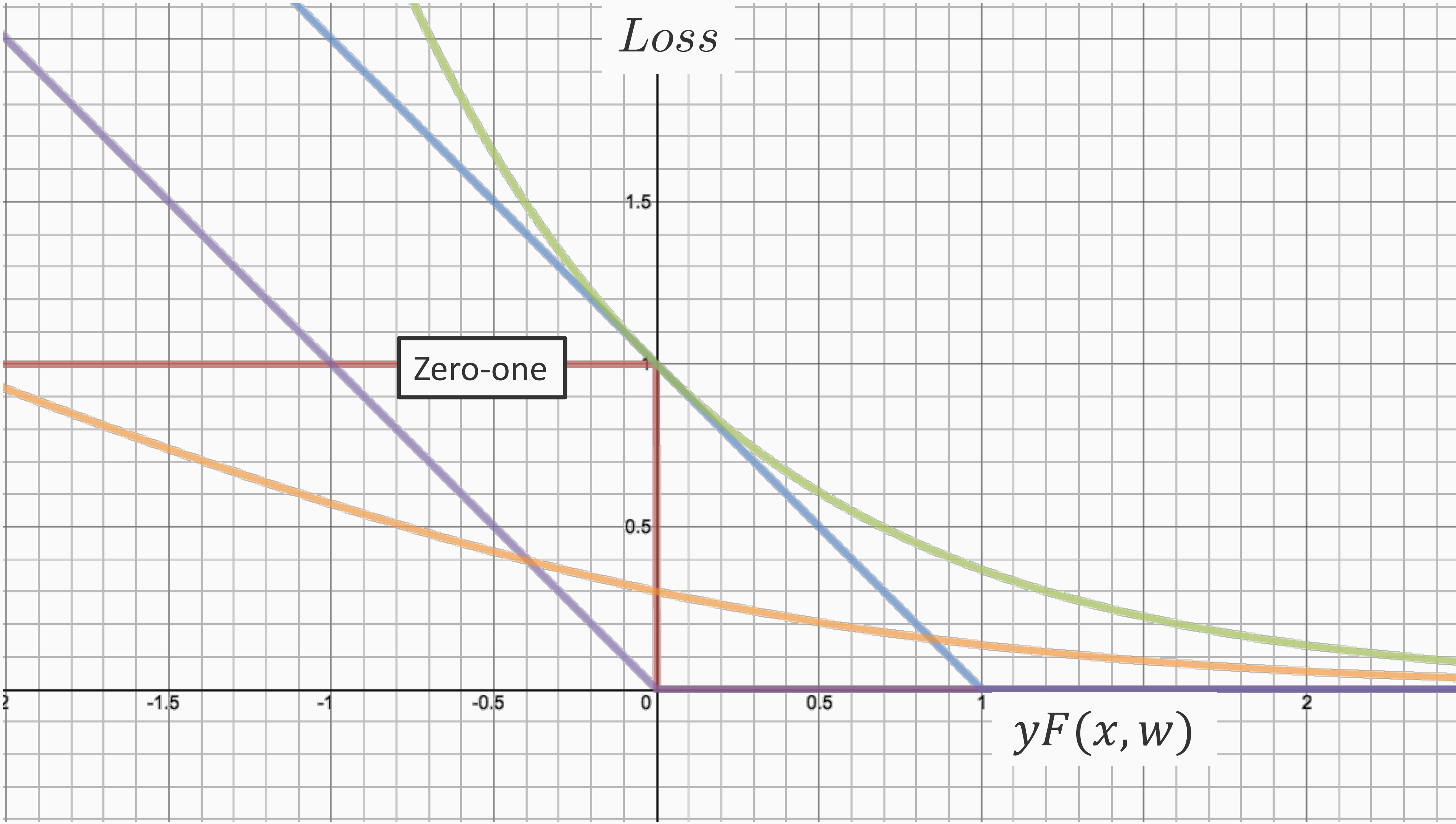
$$\min_{h \in \mathrm{H}} \text{regularizer}(\mathrm{w}) + \mathrm{C}\frac{1}{m}\sum_i L(F(x_i, w), y_i)$$

The loss function zoo



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

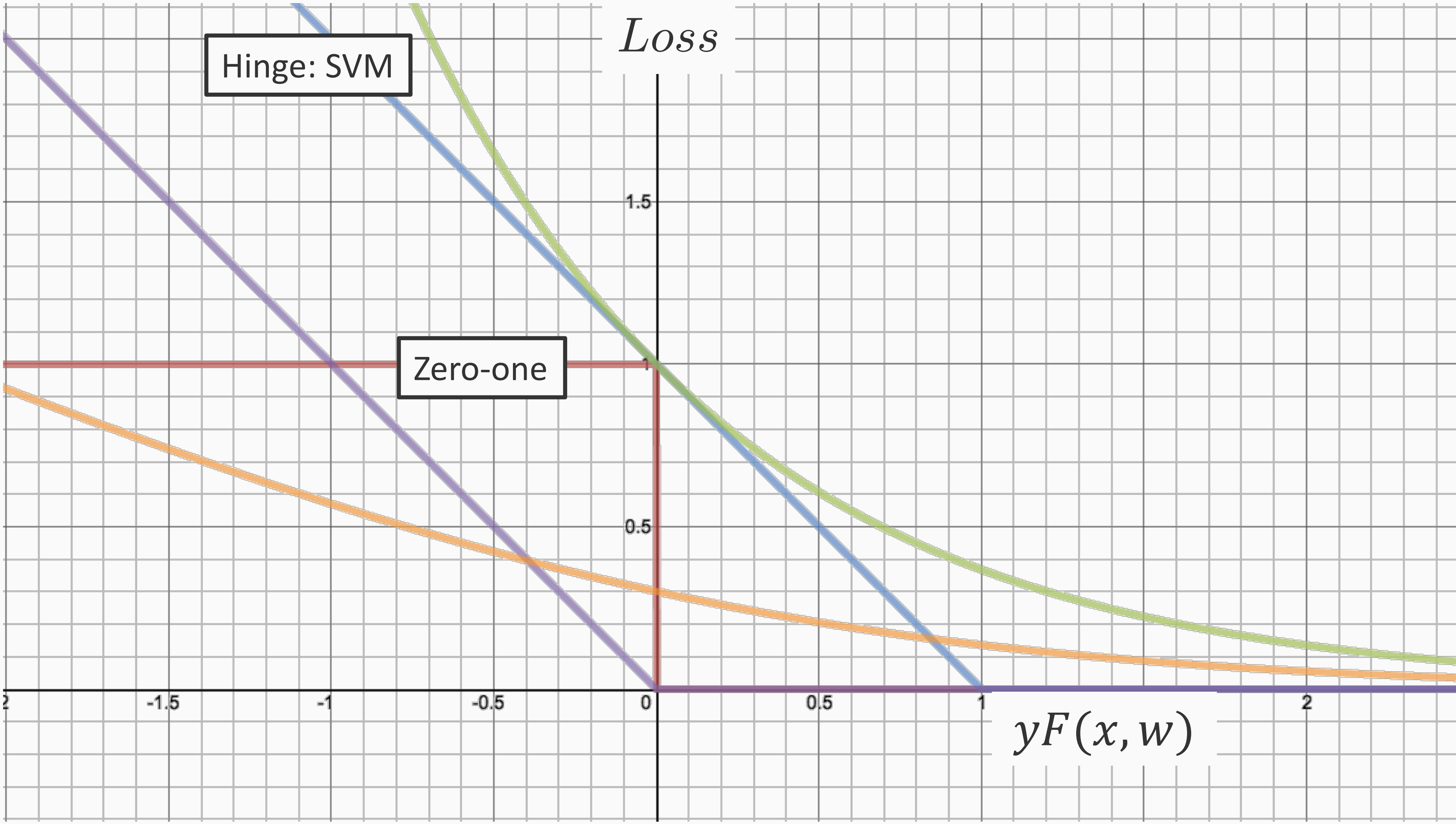
The loss function zoo



The loss function zoo

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

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

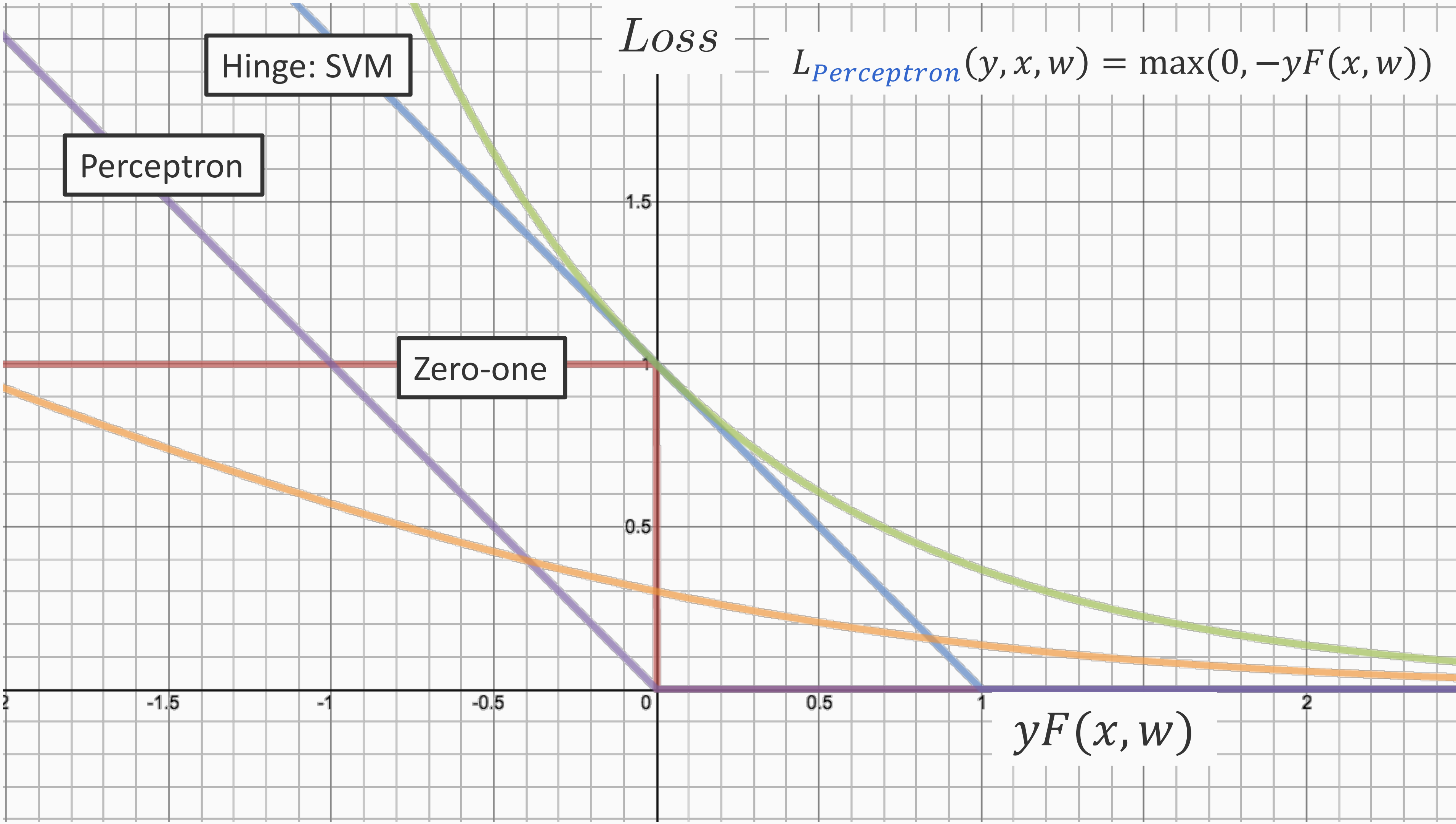


The loss function zoo

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

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

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



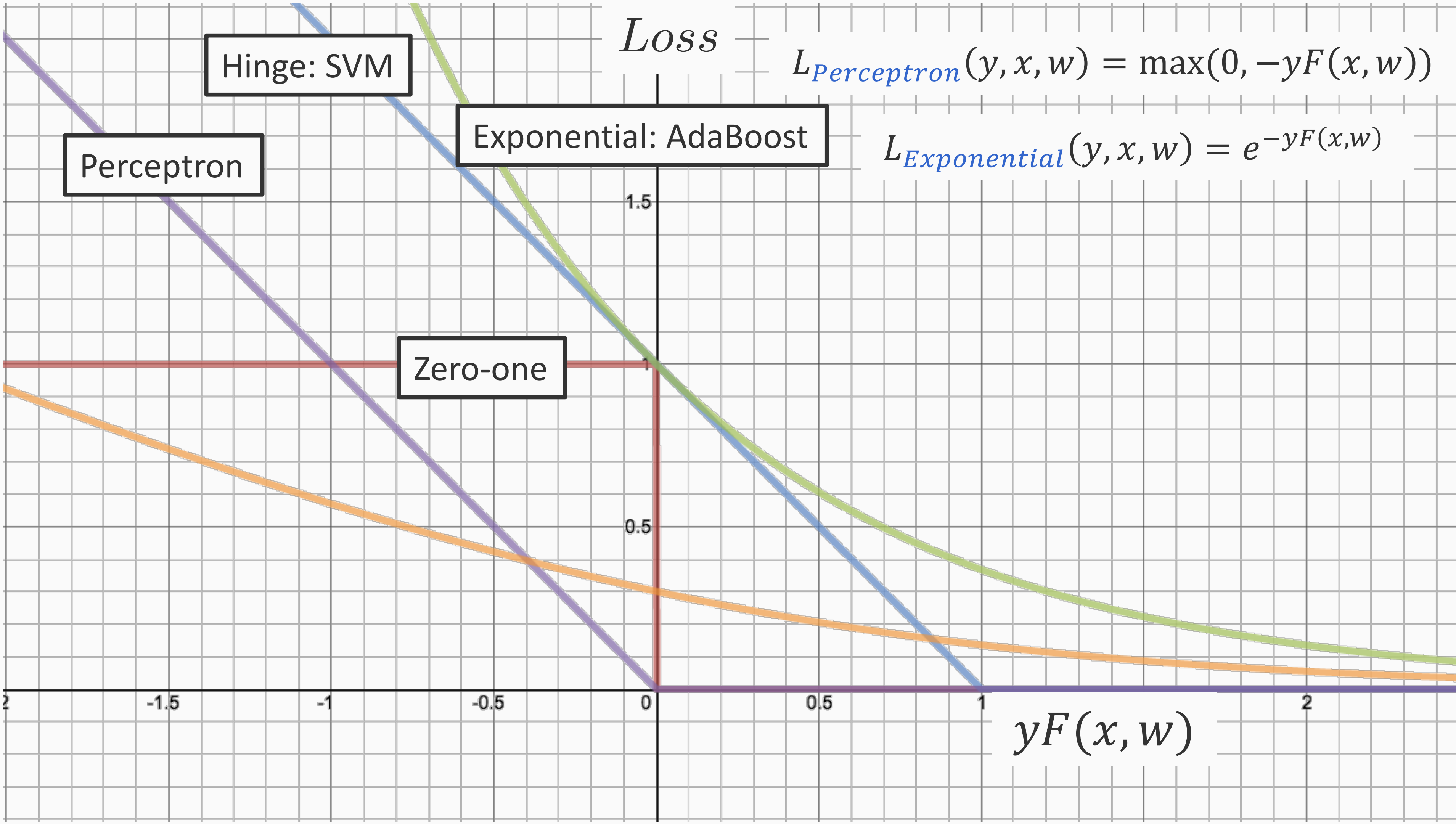
$$\min_{h \in H} \text{regularizer}(w) + C\frac{1}{m}\sum_i L(F(x_i, w), y_i)$$

The loss function zoo

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

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

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



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

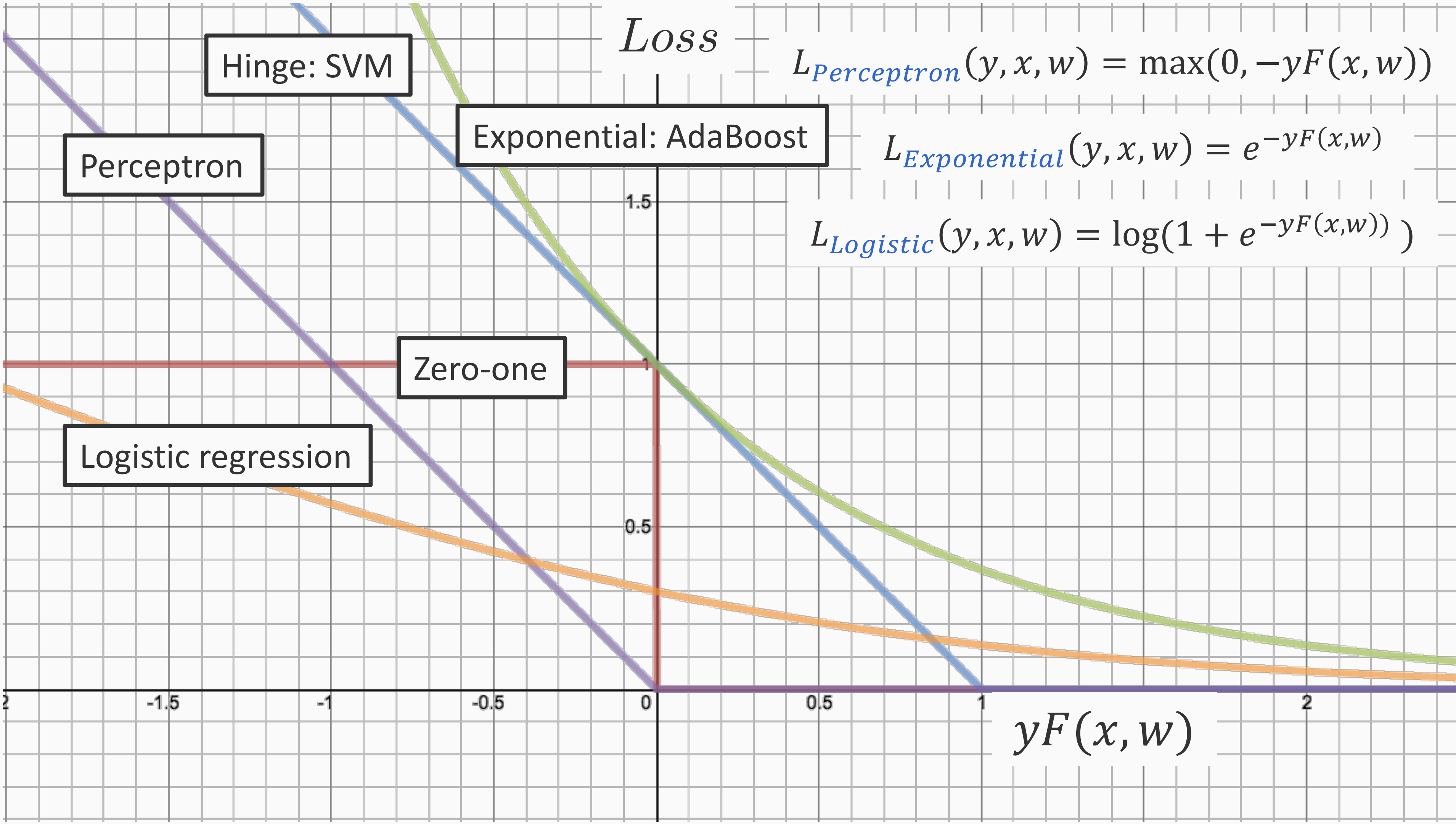
The loss function zoo

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

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

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

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



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

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

What if we have more than two labels?

What is multiclass classification?

- An input can belong to one of K classes

- Training data: Input associated with class label (a number from 1 to K)
- Prediction: Given a new input, predict the class label

Each input belongs to exactly one class. Not more, not less.

- Otherwise, the problem is not multiclass classification
- If an input can be assigned multiple labels (think tags for emails rather than folders), it is called *multi-label classification*

Example applications: Images

– *Input*: hand-written character; *Output*: which character?

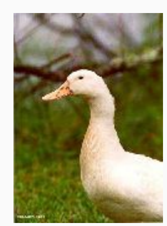
all map to the letter A

– *Input*: a photograph of an object; *Output*: which of a set of categories of objects is it?

  • Eg: the Caltech 256 dataset

Car tire

Car tire



Duck

laptop

Example applications: Language

- *Input*: a news article
- *Output*: Which section of the newspaper should be be in

- *Input*: an email
- *Output*: which folder should an email be placed into

- *Input*: an audio command given to a car
- *Output*: which of a set of actions should be executed

Coming up…

- Multi-class classification
  - Modeling multiple classes
  - Natural loss functions for this situation

- The computation graph abstraction