

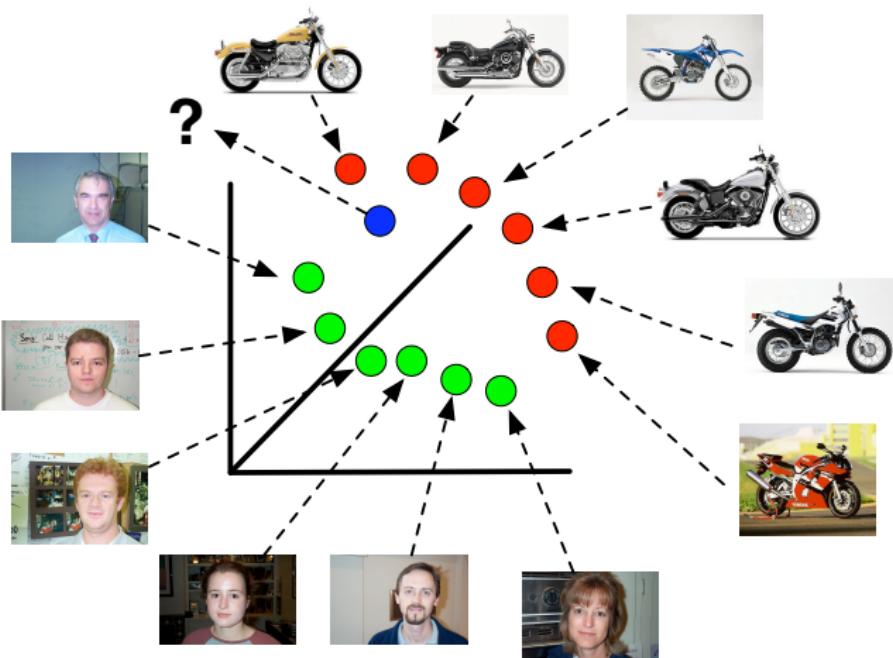
# Lecture 2 - Learning Binary & Multi-class Classifiers from Labelled Training Data

DD2424, Josephine Sullivan

March 19, 2025

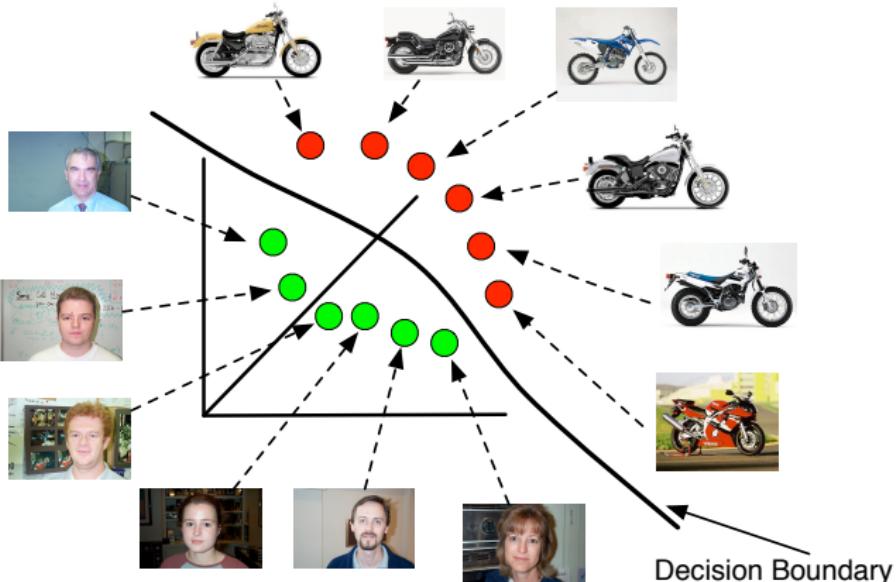
# Binary classification problem given labelled training data

Have labelled training examples



Given a test example how do we decide its class?

# High level solution



**Learn a decision boundary from the labelled training data.**

Compare the test example to the decision boundary.

# Technical description of the binary problem

- Have a set of labelled training examples

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\} \quad \text{with each } \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}.$$

- Want to learn from  $\mathcal{D}$  a classification function

$$g : \begin{matrix} \mathbb{R}^d \\ \uparrow \\ \text{input space} \end{matrix} \times \begin{matrix} \mathbb{R}^p \\ \uparrow \\ \text{parameter space} \end{matrix} \rightarrow \{-1, 1\}$$

Usually

$$g(\mathbf{x}; \boldsymbol{\theta}) = \text{sign}(f(\mathbf{x}; \boldsymbol{\theta})) \quad \text{where} \quad f : \mathbb{R}^d \times \mathbb{R}^p \rightarrow \mathbb{R}$$

- Have to decide on

1. Form of  $f$  (a hyperplane?) and
2. How to estimate  $f$ 's parameters,  $\hat{\boldsymbol{\theta}}$ , from  $\mathcal{D}$ .

# Learn decision boundary discriminatively

- Set up an optimization of the form (usually)

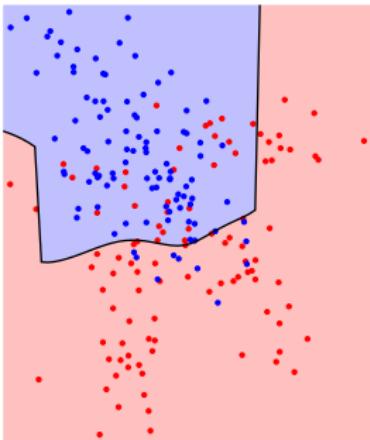
$$\arg \min_{\theta} \left[ \underbrace{\sum_{(\mathbf{x},y) \in \mathcal{D}} l(y, f(\mathbf{x}; \theta))}_{\text{loss function}} + \lambda \underbrace{R(\theta)}_{\text{regularization term}} \right]$$

where

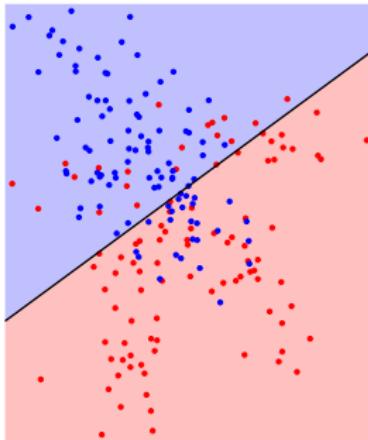
- $l(y, f(\mathbf{x}; \theta))$  is the **loss function** and measures how well (and robustly)  $f(\mathbf{x}; \theta)$  predicts the label  $y$ .
- The **training error** term measures how well and robustly the function  $f(\cdot; \theta)$  predicts the labels over all the training data.
- The **regularization** term measures the *complexity* of the function  $f(\cdot; \theta)$ .  
Usually want to learn simpler functions  $\implies$  less risk of over-fitting.

## **Comment on Over- and Under-fitting**

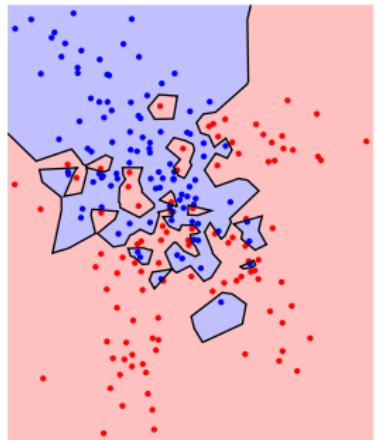
# Example of Over and Under fitting



Bayes' Optimal

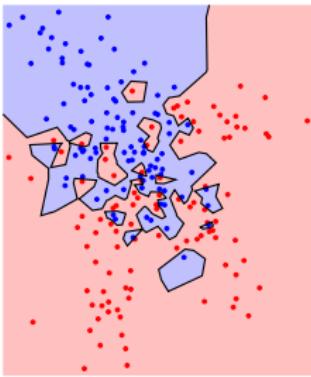


Under-fitting

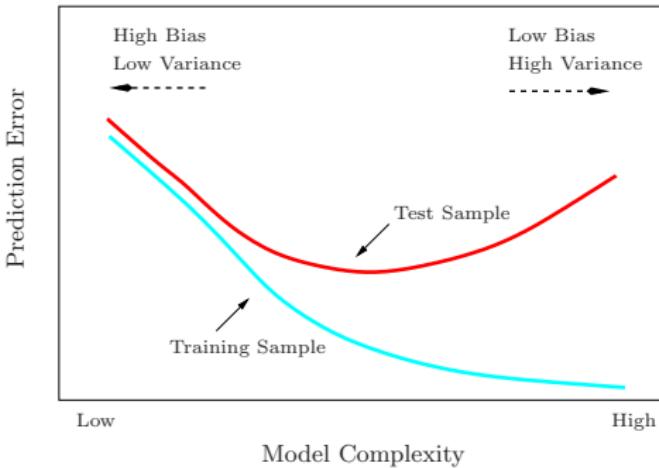


Over-fitting

## Over-fitting



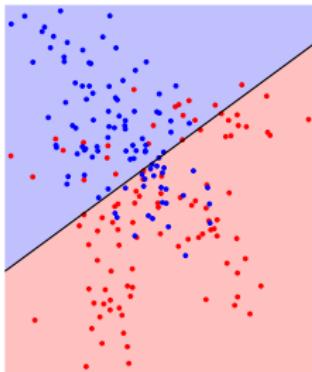
## Bias-Variance trade-off



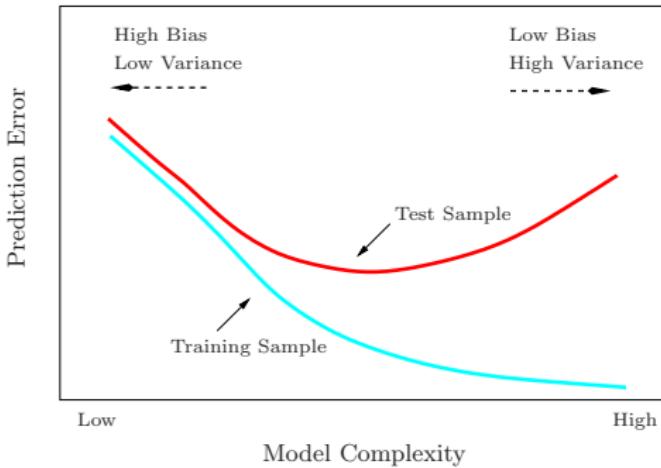
- Too much fitting  $\implies$  adapt too closely to the training data.
- Have a **high variance, low bias** predictor.
- This scenario is termed **overfitting**.
- In such cases predictor loses the ability to generalize.

# Under-fitting

## Under-fitting



## Bias-Variance trade-off



- Low complexity model  $\implies$  predictor may have high bias.
- Therefore predictor has poor generalization.

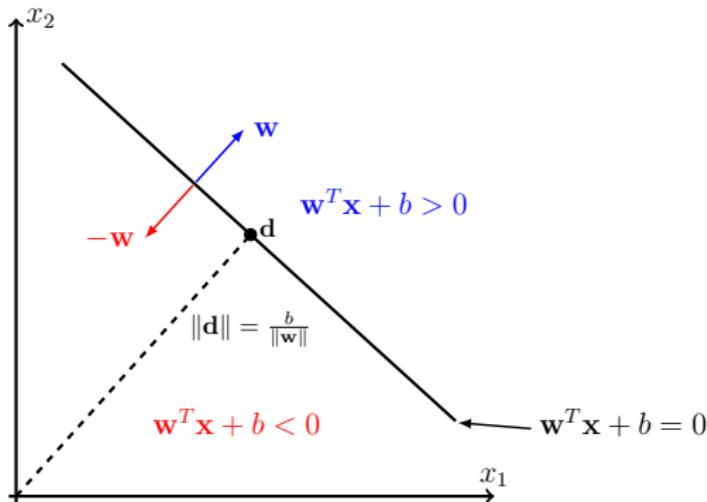
# **Linear Decision Boundaries**

# Linear discriminant functions

Linear function for the binary classification problem:

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b$$

where model parameters are  $\mathbf{w}$  the weight vector and  $b$  the bias.



$$g(\mathbf{x}; \mathbf{w}, b) = \begin{cases} 1 & \text{if } f(\mathbf{x}; \mathbf{w}, b) > 0 \\ -1 & \text{if } f(\mathbf{x}; \mathbf{w}, b) < 0 \end{cases}$$

# Pros & Cons of Linear classifiers

## Pros

- Low variance classifier
- Easy to estimate.

Frequently can set up training so that have an easy optimization problem.

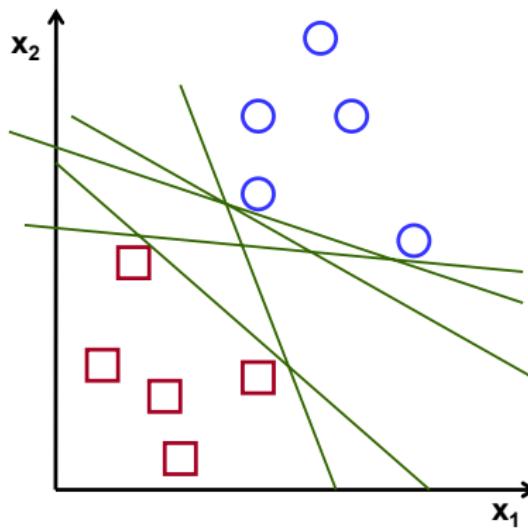
- For high dimensional input data a linear decision boundary can sometimes be sufficient.

## Cons

- High bias classifier

Often the decision boundary is not well-described by a linear classifier.

# How do we choose & learn the linear classifier?



Given labelled training data:

*How do we choose and learn the best **hyperplane** to separate the two classes?*

# Supervised learning of my classifier

Have a **linear classifier** next need to decide:

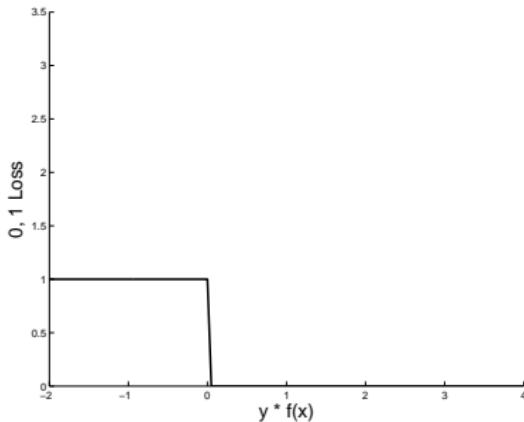
1. How to measure the quality of the classifier w.r.t. labelled training data?
  - Choose/Define a **loss function**.

# Most intuitive loss function

## 0-1 Loss function

For a single example  $(\mathbf{x}, y)$  the 0-1 loss is defined as

$$l(y, f(\mathbf{x}; \theta)) = \begin{cases} 0 & \text{if } y = \text{sgn}(f(\mathbf{x}; \theta)) \\ 1 & \text{if } y \neq \text{sgn}(f(\mathbf{x}; \theta)) \end{cases}$$



$$= \begin{cases} 0 & \text{if } y f(\mathbf{x}; \theta) > 0 \\ 1 & \text{if } y f(\mathbf{x}; \theta) \leq 0 \end{cases}$$

(assuming  $y \in \{-1, 1\}$ )

Applied to all training data  $\implies$  count the number of misclassifications.

Not really used in practice as has lots of problems! What are some?

# Supervised learning of my classifier

Have a linear classifier next need to decide:

1. How to measure the quality of the classifier w.r.t. labelled training data?
  - Choose/Define a **loss function**.
2. How to measure the complexity of the classifier?
  - Choose/Define a **regularization term**.

# Most common regularization function

## $L_2$ regularization

$$R(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i=1}^d w_i^2$$

Adding this form of regularization:

- Encourages  $\mathbf{w}$  not to contain entries with large squared values.

# Supervised learning of my classifier

Have a linear classifier next need to decide:

1. How to measure the quality of the classifier w.r.t. labelled training data?
  - Choose/Define a **loss function**.
2. How to measure the complexity of the classifier?
  - Choose/Define a **regularization term**.
3. How to estimate the classifier's parameters via optimization relative to the above factors?

**Example: Squared Error loss**

## Squared error loss & no regularization

- Learn  $\mathbf{w}, b$  from  $\mathcal{D}$ . Find the  $\mathbf{w}, b$  that minimizes:

$$L(\mathcal{D}, \mathbf{w}, b) = \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{sq}}(y, f(\mathbf{x}; \mathbf{w}, b))$$

where

$$l_{\text{sq}}(y, f(\mathbf{x}; \mathbf{w}, b)) = (f(\mathbf{x}; \mathbf{w}, b) - y)^2$$

- **Minimum Squared Error** solution:

$$\mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} L(\mathcal{D}, \mathbf{w}, b)$$

- This minimum is found as follows....

# Pseudo-Inverse solution

- First absorb the bias term into the dot-product

$$L(\mathcal{D}, \mathbf{w}, b) = \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}} (\mathbf{w}^T \mathbf{x} + b - y)^2 = \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}} (\mathbf{w}_1^T \mathbf{x}' - y)^2$$

where

$$\mathbf{x}' = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{pmatrix}, \quad \mathbf{w}_1 = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{pmatrix}$$

# Pseudo-Inverse solution

- Matrix formulation of cost function:

$$\begin{aligned} L(\mathcal{D}, \mathbf{w}_1) &= .5 \sum_{(\mathbf{x}, y) \in \mathcal{D}} (\mathbf{w}_1^T \mathbf{x}' - y)^2 \\ &= .5 \|X\mathbf{w}_1 - \mathbf{y}\|^2 \\ &= .5 (X\mathbf{w}_1 - \mathbf{y})^T (X\mathbf{w}_1 - \mathbf{y}) \\ &= .5 (\mathbf{w}_1^T X^T X \mathbf{w}_1 - 2\mathbf{y}^T X \mathbf{w}_1 + \mathbf{y}^T \mathbf{y}) \end{aligned}$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \underbrace{\begin{pmatrix} \leftarrow \mathbf{x}_1^T \rightarrow & 1 \\ \leftarrow \mathbf{x}_2^T \rightarrow & 1 \\ \vdots & \vdots \\ \leftarrow \mathbf{x}_n^T \rightarrow & 1 \end{pmatrix}}_{n \times (d+1)}, \quad \mathbf{w}_1 = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ w_{d+1} \end{pmatrix}$$

## **Technical interlude: Matrix Calculus**

- Have a function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  that is  $f(\mathbf{x}) = c$
- We use the notation

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{pmatrix}$$

- **Example:** If  $f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} = \sum_{i=1}^d a_i x_i$  then

$$\frac{\partial f}{\partial x_i} = a_i \quad \Rightarrow \quad \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} a_1 \\ \vdots \\ a_d \end{pmatrix} = \mathbf{a}$$

## Derivative of a linear function

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

**Derivative of a quadratic function**

$$\frac{\partial \mathbf{x}^T B \mathbf{x}}{\partial \mathbf{x}} = (B + B^T) \mathbf{x}$$

$$\frac{\partial (B\mathbf{x} + \mathbf{b})^T C (D\mathbf{x} + \mathbf{d})}{\partial \mathbf{x}} = B^T C (D\mathbf{x} + \mathbf{d}) + D^T C^T (B\mathbf{x} + \mathbf{b})$$

**End of Technical interlude**

## Pseudo-Inverse solution

- The gradient of  $L(\mathcal{D}, \mathbf{w}_1)$  w.r.t.  $\mathbf{w}_1$ :

$$\nabla_{\mathbf{w}_1} L(\mathcal{D}, \mathbf{w}_1) = X^T X \mathbf{w}_1 - X^T \mathbf{y}$$

- Setting this equal to zero yields  $X^T X \mathbf{w}_1 = X^T \mathbf{y}$  and

$$\mathbf{w}_1 = X^\dagger \mathbf{y}$$

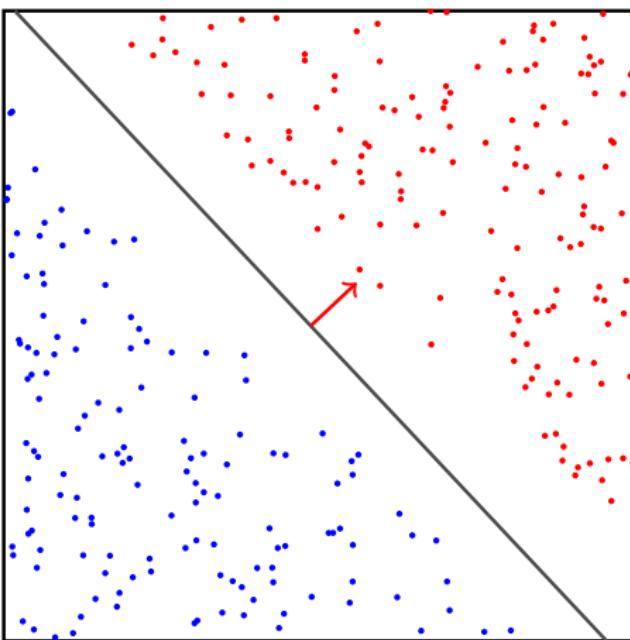
where

$$X^\dagger \equiv (X^T X)^{-1} X^T$$

if we assume  $n > d + 1$  and  $X$  has full column rank.

- $X^\dagger$  is called the **pseudo-inverse** of  $X$ . Note that  $X^\dagger X = I$  but in general  $XX^\dagger \neq I$ .

## Simple 2D Example



Decision boundary found by minimizing

$$L_{\text{squared error}}(\mathcal{D}, \mathbf{w}, b) = \sum_{(\mathbf{x}, y) \in \mathcal{D}} (y - (\mathbf{w}^T \mathbf{x} + b))^2$$

## Pseudo-Inverse solution

- The gradient of  $L(\mathcal{D}, \mathbf{w}_1)$  w.r.t.  $\mathbf{w}_1$ :

$$\nabla_{\mathbf{w}_1} L(\mathcal{D}, \mathbf{w}_1) = X^T X \mathbf{w}_1 - X^T \mathbf{y}$$

- Setting this equal to zero yields  $X^T X \mathbf{w}_1 = X^T \mathbf{y}$  and

$$\mathbf{w}_1 = X^\dagger \mathbf{y}$$

where

$$X^\dagger \equiv (X^T X)^{-1} X^T$$

if we assume  $n > d + 1$  and  $X$  has full column rank.

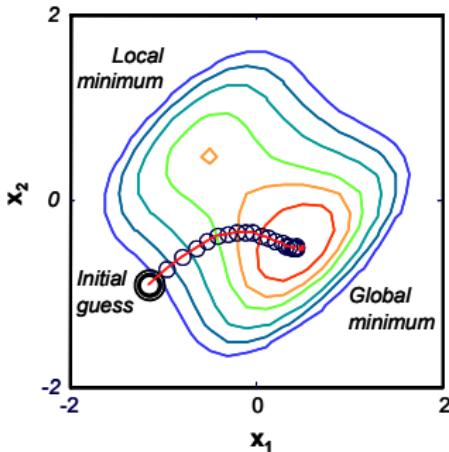
- $X^\dagger$  is called the **pseudo-inverse** of  $X$ . Note that  $X^\dagger X = I$  but in general  $XX^\dagger \neq I$ .
- If  $X^T X$  singular  $\implies$  no unique solution to  $X^T X \mathbf{w} = X^T \mathbf{y}$ .

## Technical interlude: Iterative Optimization

# Iterative Optimization

- Unconstrained optimization

$$\mathbf{x}^* = \arg \min_{\forall \mathbf{x}} f(\mathbf{x})$$



Common soln: **Iterative non-linear optimization**

- Start with an estimate  $\mathbf{x}^{(0)}$ .
- Improve it by finding successive new estimates  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \dots$  s.t.  $f(\mathbf{x}^{(1)}) \geq f(\mathbf{x}^{(2)}) \geq f(\mathbf{x}^{(3)}) \geq \dots$  until convergence.
- **To find a better estimate at each iteration:** Perform the search locally around the current estimate.
- Such iterative approaches will find a local minima.

Iterative optimization methods alternate between these two steps:

## **Decide search direction**

Choose a search direction based on the local properties of the cost function.

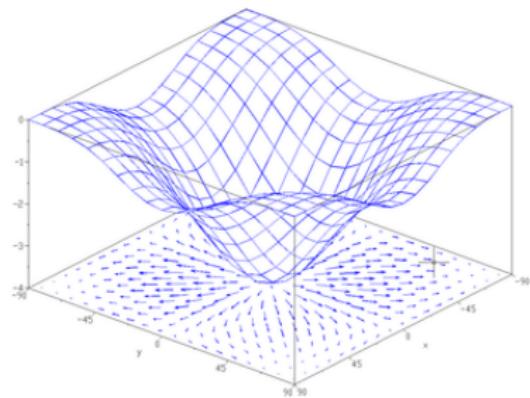
## **Line Search**

Perform an intensive search to find the minimum along the chosen direction.

# Choosing a search direction: The gradient

The gradient is defined as:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) \equiv \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{pmatrix}$$

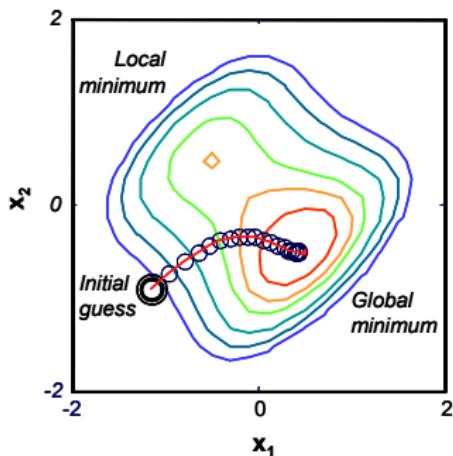


The gradient points in the direction of the greatest increase of  $f(\mathbf{x})$ .

# Gradient descent: Method for function minimization

Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent.

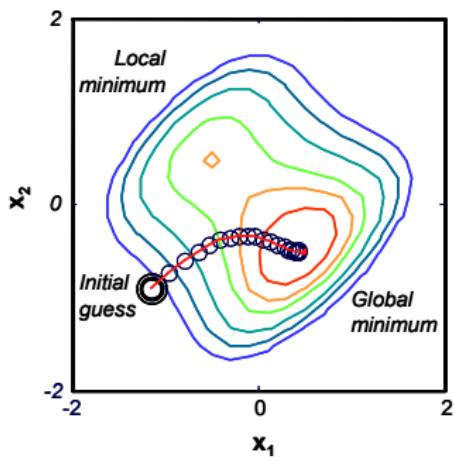
## Gradient Descent Minimization



1. Start with an arbitrary solution  $\mathbf{x}^{(0)}$ .
2. Compute the gradient  $\nabla_{\mathbf{x}} f(\mathbf{x}^{(k)})$ .
3. Move in the direction of steepest descent:
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \nabla_{\mathbf{x}} f(\mathbf{x}^{(k)}).$$
where  $\eta^{(k)}$  is the step size.
4. Go to 2 (until convergence).

# Gradient descent: Method for function minimization

Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent.



## Gradient Descent Minimization Properties

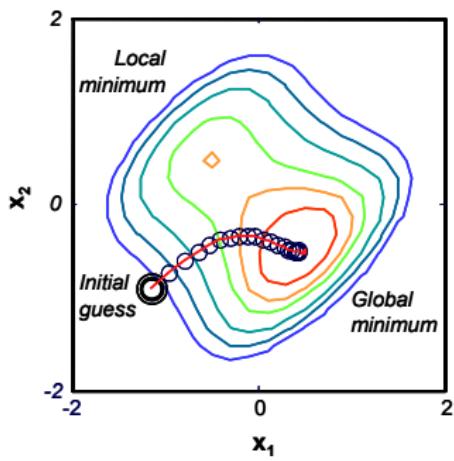
1. Will converge to a local minimum.
2. The local minimum found depends on the initialization  $\mathbf{x}^{(0)}$ .

But this is okay

1. For convex optimization problems: local minimum  $\equiv$  global minimum
2. For deep networks most parameter settings corresponding to a local minimum are fine.

# Gradient descent: Method for function minimization

Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent.



## Gradient Descent Minimization Properties

1. Will converge to a local minimum.
2. The local minimum found depends on the initialization  $\mathbf{x}^{(0)}$ .

But this is okay

1. For convex optimization problems: **local minimum  $\equiv$  global minimum**
2. For deep networks most parameter settings corresponding to a **local minimum** are fine.

**End of Technical interlude**

The error function  $L(\mathcal{D}, \mathbf{w}_1)$  could also be minimized w.r.t.  $\mathbf{w}_1$  by using a **gradient descent procedure**.

## Why?

- This avoids the numerical problems that arise when  $X^T X$  is (nearly) singular.
- It also avoids the need for inverting potentially a large matrix.

## How

1. Begin with an initial guess  $\mathbf{w}_1^{(0)}$  for  $\mathbf{w}_1$ .
2. Update the weight vector by moving a small distance in the direction  $-\nabla_{\mathbf{w}_1} L$ .

**Solution**

$$\mathbf{w}_1^{(t+1)} = \mathbf{w}_1^{(t)} - \eta^{(t)} X^T (X\mathbf{w}_1^{(t)} - \mathbf{y})$$

- If  $\eta^{(t)} = \eta_0/t$ , where  $\eta_0 > 0$ , then
- $\mathbf{w}_1^{(0)}, \mathbf{w}_1^{(1)}, \mathbf{w}_1^{(2)}, \dots$  converges to a solution of

$$X^T (X\mathbf{w}_1 - \mathbf{y}) = \mathbf{0}$$

- Irrespective of whether  $X^T X$  is singular or not.

## Stochastic gradient descent solution

- Increase the number of updates per computation: **Consider each training sample sequentially**

$$\mathbf{w}_1^{(t+1)} = \mathbf{w}_1^{(t)} - \eta^{(t)} (\mathbf{x}_i^T \mathbf{w}_1^{(t)} - y_i) \mathbf{x}_i$$

- This is known as the **Widrow-Hoff, least-mean-squares** (LMS) or **delta rule** [Mitchell, 1997].
- More generally this is an application of **Stochastic Gradient Descent**.

## Technical interlude: Stochastic Gradient Descent

# Common Optimization Problem in Machine Learning

- **Form of the optimization problem:**

$$J(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l(y, f(\mathbf{x}; \boldsymbol{\theta})) + \lambda R(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} J(\mathcal{D}, \boldsymbol{\theta})$$

- **Solution with gradient descent**

1. Start with a random guess  $\boldsymbol{\theta}^{(0)}$  for the parameters.
2. Then iterate until convergence

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}} J(\mathcal{D}, \boldsymbol{\theta})|_{\boldsymbol{\theta}^{(t)}}$$

$$J(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l(y, f(\mathbf{x}; \boldsymbol{\theta})) + \lambda R(\boldsymbol{\theta})$$

If  $|\mathcal{D}|$  is large

- $\implies$  computing  $\nabla_{\boldsymbol{\theta}} J(\mathcal{D}, \boldsymbol{\theta})|_{\boldsymbol{\theta}^{(t)}}$  is time consuming
- $\implies$  each update of  $\boldsymbol{\theta}^{(t)}$  takes lots of computations
- Gradient descent needs lots of iterations to converge as  $\eta$  usually small
- $\implies$  GD takes an age to find a local optimum.

## Work around: Stochastic Gradient Descent

- Start with a random solution  $\boldsymbol{\theta}^{(0)}$ .
- Until convergence for  $t = 1, \dots$ 
  1. Randomly select  $(\mathbf{x}, y) \in \mathcal{D}$ .
  2. Set  $\mathcal{D}^{(t)} = \{(\mathbf{x}, y)\}$ .
  3. Update parameter estimate with

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}} J(\mathcal{D}^{(t)}, \boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}^{(t)}}$$

- When  $|\mathcal{D}^{(t)}| = 1$ :  
 $\nabla_{\theta} J(\mathcal{D}^{(t)}, \theta)|_{\theta^{(t)}}$  a noisy estimate of  $\nabla_{\theta} J(\mathcal{D}, \theta)|_{\theta^{(t)}}$
- Therefore for the same computational effort  
 $|\mathcal{D}|$  **noisy** update steps in SGD  $\approx 1$  **correct** update step in GD.
- In practice SGD converges a lot faster (w.r.t. amount of computation) than GD.
- Given lots of labelled training data:  
**Quantity** of updates more important than **quality** of updates!

- **Preparing the data**
  - Randomly shuffle the training examples and zip sequentially through  $\mathcal{D}$ .
  - Use pre-conditioning techniques.
- **Monitoring and debugging**
  - Monitor both the training cost and the validation error.
  - Check the gradients using finite differences.
  - Experiment with learning rates  $\eta^{(t)}$  using a small sample of the training set.

# Mini-Batch Gradient Descent

- Start with a random guess  $\boldsymbol{\theta}^{(0)}$  for the parameters.
- Until convergence for  $t = 1, \dots$ 
  1. Randomly select a subset  $\mathcal{D}^{(t)} \subset \mathcal{D}$  s.t.  $|\mathcal{D}^{(t)}| = n_b$  (typically  $n_b \approx 128, 64, 32, \dots$ )
  2. Update parameter estimate with

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}} J(\mathcal{D}^{(t)}, \boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}^{(t)}}$$

## Benefits of mini-batch gradient descent

- Obtain a more accurate estimate of  $\nabla_{\theta} J(\mathcal{D}, \theta)|_{\theta^{(t)}}$  than in SGD.
- Still get lots of updates per epoch (one iteration through all the training data).

- Issues with setting the learning rate  $\eta^{(t)}$ ?
  - Larger  $\eta$ 's  $\implies$  potentially faster learning but with the risk of less stable convergence.
  - Smaller  $\eta$ 's  $\implies$  slow learning but stable convergence.
- Strategies
  - Constant:  $\eta^{(t)} = .01$
  - Decreasing:  $\eta^{(t)} = 1/\sqrt{t}$
- Lots of recent algorithms dealing with this issue

Will describe these algorithms in later lectures.

**End of Technical interlude**

**Squared Error loss +  $L_2$  regularization**

## Add an $L_2$ regularization term (a.k.a. ridge regression)

- Add a regularization term to the loss function

$$J_{\text{ridge}}(\mathcal{D}, \mathbf{w}, b) = \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{sq}}(y, \mathbf{w}^T \mathbf{x} + b) + \lambda \|\mathbf{w}\|^2$$

where  $\lambda > 0$ .

- In matrix notation

$$J_{\text{ridge}}(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{2} \|X\mathbf{w} + b\mathbf{1} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$$

where  $X$  is the data matrix

$$X = \begin{pmatrix} \leftarrow & \mathbf{x}_1^T & \rightarrow \\ \leftarrow & \mathbf{x}_2^T & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_n^T & \rightarrow \end{pmatrix}$$

# Solving Ridge Regression: Centre the data to simplify

$$J_{\text{ridge}}(\mathcal{D}, \mathbf{w}, b) = \frac{1}{2} \|X\mathbf{w} + b\mathbf{1} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$$

- Centre the input data

$$X_c = \begin{pmatrix} \leftarrow & \mathbf{x}_{c,1}^T & \rightarrow \\ \leftarrow & \mathbf{x}_{c,2}^T & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_{c,n}^T & \rightarrow \end{pmatrix} \quad \text{where } \mathbf{x}_{c,i} = \mathbf{x}_i - \boldsymbol{\mu}_{\mathbf{x}}$$

$$\implies X_c^T \mathbf{1} = \mathbf{0}.$$

- Optimal bias with centered input  $X_c$  (does not depend on  $\mathbf{w}^*$ ) is:

$$\begin{aligned} \frac{\partial J_{\text{ridge}}}{\partial b} &= b\mathbf{1}^T \mathbf{1} + \mathbf{w}^T X_c^T \mathbf{1} - \mathbf{1}^T \mathbf{y} \\ &= b\mathbf{1}^T \mathbf{1} - \mathbf{1}^T \mathbf{y} \end{aligned}$$

$$\implies b^* = 1/n \sum_{i=1}^n y_i = \bar{y}.$$

## Solving ridge regression: Optimal weight vector

$$J_{\text{ridge}}(\mathcal{D}, \mathbf{w}) = \frac{1}{2} \|X_c \mathbf{w} + \bar{y} \mathbf{1} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$$

- Compute the gradient of  $J_{\text{ridge}}$  w.r.t.  $\mathbf{w}$

$$\frac{\partial J_{\text{ridge}}}{\partial \mathbf{w}} = (X_c^T X_c + \lambda I_d) \mathbf{w} - X_c^T \mathbf{y}$$

- Set to zero to get

$$\mathbf{w}^* = (X_c^T X_c + \lambda I_d)^{-1} X_c^T \mathbf{y}$$

- $(X_c^T X_c + \lambda I_d)$  has a unique inverse even if  $X_c^T X_c$  is singular.

## Solution for original data from centred solution

- For both the centred and original data:

$$\mathbf{w}^* = (X_c^T X_c + \lambda I_d)^{-1} X_c^T \mathbf{y}$$

- For the centred and original data:

$$b_c^* = \bar{y}$$

$$b^* = \bar{y} - \boldsymbol{\mu}_{\mathbf{x}}^T \mathbf{w}^*$$

- Distance from origin to the decision boundary is given by:

$$d = \frac{|b^*|}{\|\mathbf{w}^*\|}$$

## Solving ridge regression: Optimal weight vector

$$J_{\text{ridge}}(\mathcal{D}, \mathbf{w}, b) = \frac{1}{2} \|X\mathbf{w} + b\mathbf{1} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$$

- Compute the gradient of  $J_{\text{ridge}}$  w.r.t.  $\mathbf{w}$

$$\frac{\partial J_{\text{ridge}}}{\partial \mathbf{w}} = (X_c^T X_c + \lambda I_d) \mathbf{w} - X_c^T \mathbf{y}$$

- Set to zero to get

$$\mathbf{w}^* = (X_c^T X_c + \lambda I_d)^{-1} X_c^T \mathbf{y}$$

- $(X_c^T X_c + \lambda I_d)$  has a unique inverse even if  $X_c^T X_c$  is singular.
- If  $d$  is large  $\implies$  have to invert a very large matrix.

## Solving ridge regression: Iteratively

- The gradient-descent update step is

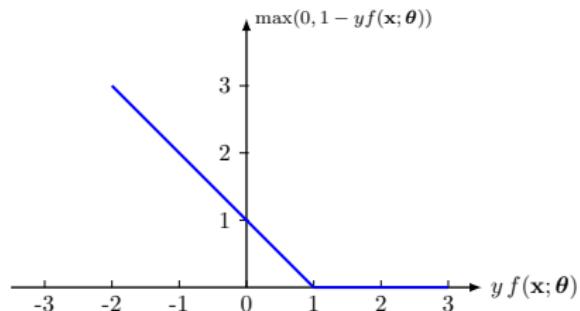
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left[ (X_c^T X_c + \lambda I_d) \mathbf{w}^{(t)} - X_c^T \mathbf{y} \right]$$

- The SGD update step for sample  $(\mathbf{x}, y)$  is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left[ ((\mathbf{x} - \boldsymbol{\mu}_x)(\mathbf{x} - \boldsymbol{\mu}_x)^T + \lambda I_d) \mathbf{w}^{(t)} - (\mathbf{x} - \boldsymbol{\mu}_x)y \right]$$

# Hinge Loss

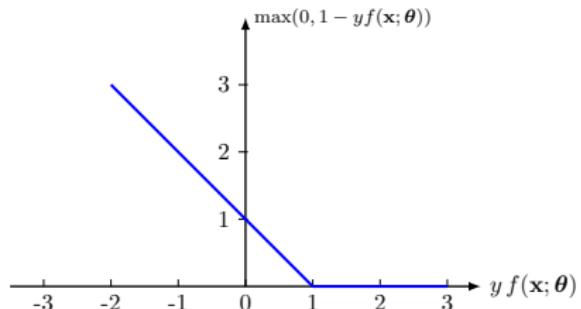
# The Hinge loss



Remember  $y \in \{-1, 1\}$  and classification decision given by  $\text{sign}(f(\mathbf{x}; \boldsymbol{\theta}))$  then

$$l_{\text{hinge}}(\mathbf{x}, y; \mathbf{w}, b) = \begin{cases} 1 - y f(\mathbf{x}; \boldsymbol{\theta}) & \text{if } y f(\mathbf{x}; \boldsymbol{\theta}) < 1 \\ 0 & \text{otherwise} \end{cases}$$

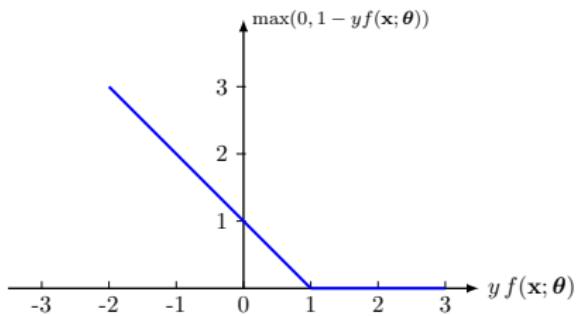
# The Hinge loss written compactly



Remember  $y \in \{-1, 1\}$  and classification decision given by  $\text{sign}(f(\mathbf{x}; \theta))$  then

$$l_{\text{hinge}}(\mathbf{x}, y; \mathbf{w}, b) = \max(0, 1 - y f(\mathbf{x}; \theta))$$

# The Hinge loss properties



- This loss is not differentiable but is convex.
- Correctly classified examples *sufficiently* far from the decision boundary have zero loss.
- $\implies$  have a way of choosing between classifiers that correctly classify all the training examples.

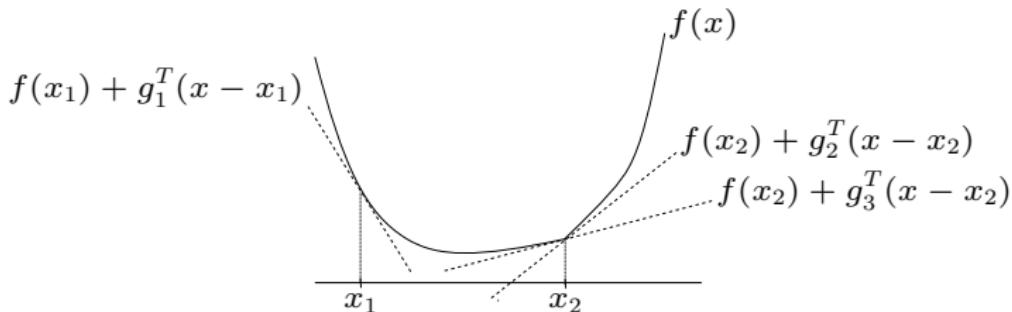
## Technical interlude: Sub-gradient

# Subgradient of a function

- $\mathbf{g}$  is a **subgradient** of  $f$  at  $\mathbf{x}$  if

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}) \quad \forall \mathbf{y}$$

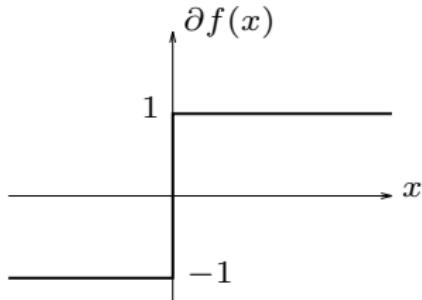
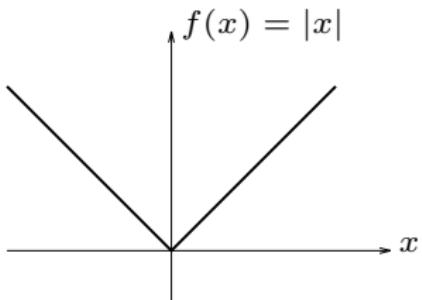
- **1D example:**



- $g_2, g_3$  are subgradients at  $x_2$ ;
- $g_1$  is a subgradient at  $x_1$ .

# Subgradient of a function

- Set of all subgradients of  $f$  at  $\mathbf{x}$  is called the subdifferential of  $f$  at  $\mathbf{x}$ , written  $\partial f(\mathbf{x})$
- **1D example:**



- If  $f$  is convex and differentiable:  $\nabla f(\mathbf{x})$  a subgradient of  $f$  at  $\mathbf{x}$ .

**End of Technical interlude**

# The Hinge loss sub-gradients

- Find  $\mathbf{w}, b$  that minimize

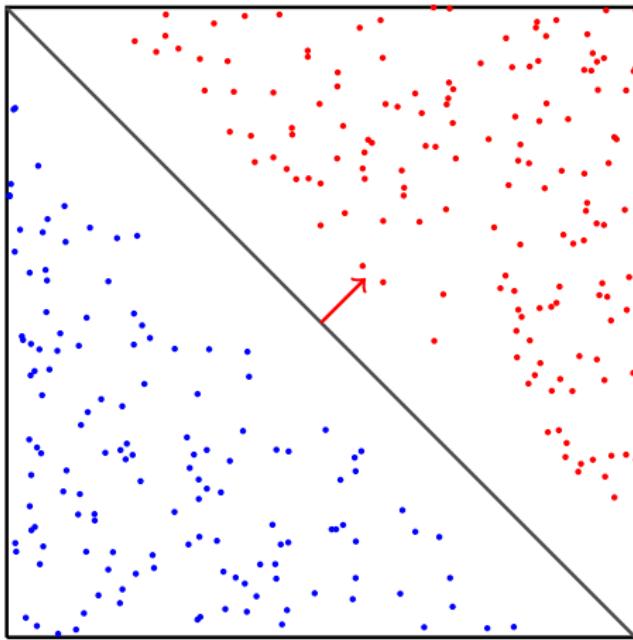
$$L_{\text{hinge}}(\mathcal{D}, \mathbf{w}, b) = \sum_{(\mathbf{x}, y) \in \mathcal{D}} \underbrace{\max(0, 1 - y(\mathbf{w}^T \mathbf{x} + b))}_{\text{Hinge Loss}}$$

- Can use stochastic gradient descent to do the optimization.
- The (sub-)gradients of the hinge-loss are

$$\nabla_{\mathbf{w}} l(\mathbf{x}, y; \mathbf{w}, b) = \begin{cases} -y \mathbf{x} & \text{if } y(\mathbf{w}^T \mathbf{x} + b) < 1 \\ 0 & \text{otherwise.} \end{cases}$$

$$\frac{\partial l(\mathbf{x}, y; \mathbf{w}, b)}{\partial b} = \begin{cases} -y & \text{if } y(\mathbf{w}^T \mathbf{x} + b) < 1 \\ 0 & \text{otherwise.} \end{cases}$$

## Example of decision boundary found



Decision boundary found by minimizing with SGD

$$L_{\text{hinge}}(\mathcal{D}, \mathbf{w}, b) = \sum_{(\mathbf{x}, y) \in \mathcal{D}} \max(0, 1 - y(\mathbf{w}^T \mathbf{x} + b))$$

$L_2$  Regularization + Hinge Loss

## $L_2$ regularization + Hinge loss

- Find  $\mathbf{w}, b$  that minimize

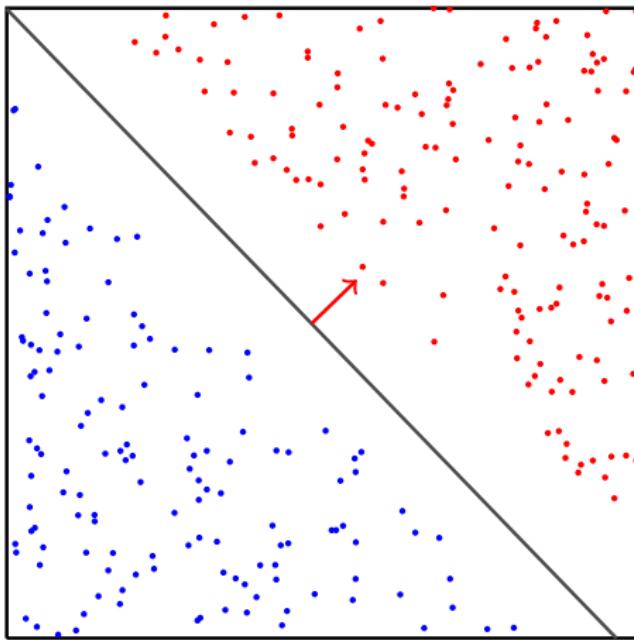
$$J_{\text{svm}}(\mathcal{D}, \mathbf{w}, b) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{(\mathbf{x}, y) \in \mathcal{D}} \underbrace{\max(0, 1 - y(\mathbf{w}^T \mathbf{x} + b))}_{\text{Hinge Loss}}$$

- Can use stochastic gradient descent to do the optimization.
- The sub-gradients of this cost function

$$\nabla_{\mathbf{w}} l(\mathbf{x}, y; \mathbf{w}, b) = \begin{cases} \lambda \mathbf{w} - y \mathbf{x} & \text{if } y(\mathbf{w}^T \mathbf{x} + b) < 1 \\ \lambda \mathbf{w} & \text{otherwise.} \end{cases}$$

$$\frac{\partial l(\mathbf{x}, y; \mathbf{w}, b)}{\partial b} = \begin{cases} -y & \text{if } y(\mathbf{w}^T \mathbf{x} + b) < 1 \\ 0 & \text{otherwise.} \end{cases}$$

## Example of decision boundary found

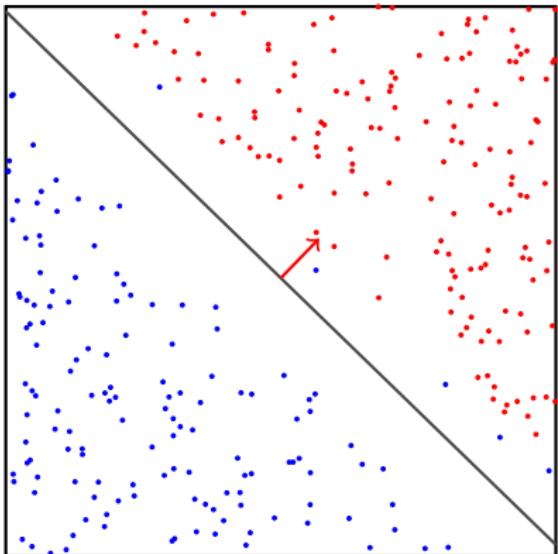


Decision boundary found with SGD by minimizing ( $\lambda = .01$ )

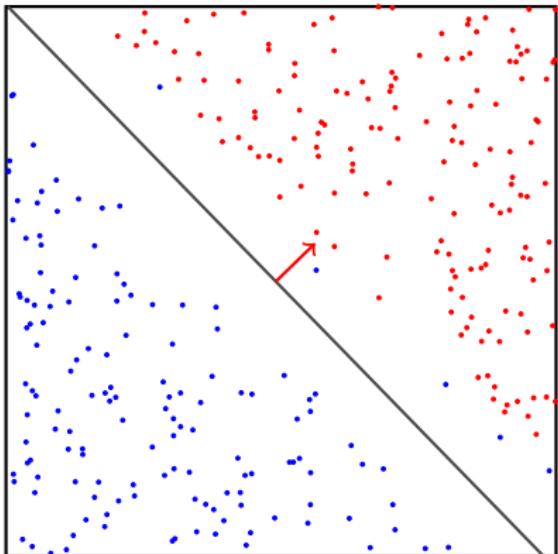
$$J_{\text{hinge}}(\mathcal{D}, \mathbf{w}, b) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{(\mathbf{x}, y) \in \mathcal{D}} \max(0, 1 - y(\mathbf{w}^T \mathbf{x} + b))$$

# Regularization reduces the influence of *outliers*

Decision boundaries found by minimizing



Hinge Loss



$L_2$  Regularization + Hinge Loss

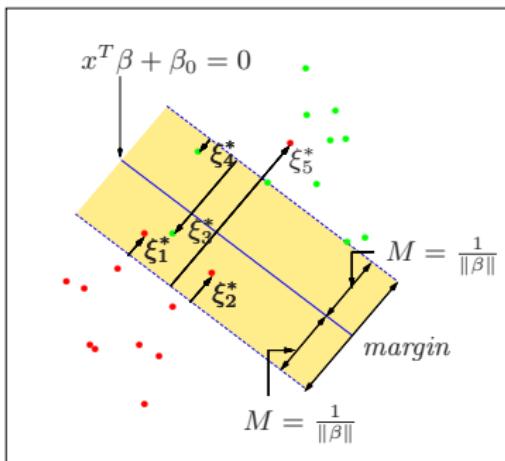
**“ $L_2$  Regularization + Hinge Loss”  $\equiv$  SVM**

# SVM's constrained optimization problem

SVM solves this constrained optimization problem:

$$\min_{\mathbf{w}, b} \left( \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^n \xi_i \right) \quad \text{subject to}$$

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \dots, n \quad \text{and}$$
$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, n.$$



# Alternative formulation of SVM optimization

SVM solves this constrained optimization problem:

$$\min_{\mathbf{w}, b} \left( \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^n \xi_i \right) \quad \text{subject to}$$

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \dots, n \quad \text{and}$$
$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, n.$$

- Let's look at the constraints:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \implies \xi_i \geq 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)$$

- But  $\xi_i \geq 0$  also, therefore

$$\xi_i \geq \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))$$

## Alternative formulation of the SVM optimization

Thus the original **constrained optimization problem** can be restated as an **unconstrained optimization problem**:

$$\min_{\mathbf{w}, b} \left( \underbrace{\frac{1}{2} \|\mathbf{w}\|^2}_{\text{Regularization term}} + C \sum_{i=1}^n \underbrace{\max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))}_{\text{Hinge loss}} \right)$$

and corresponds to the  $L_2$  **regularization + Hinge loss** formulation!

⇒ can train SVMs with SGD/mini-batch gradient descent.

## Alternative formulation of the SVM optimization

Thus the original **constrained optimization problem** can be restated as an **unconstrained optimization problem**:

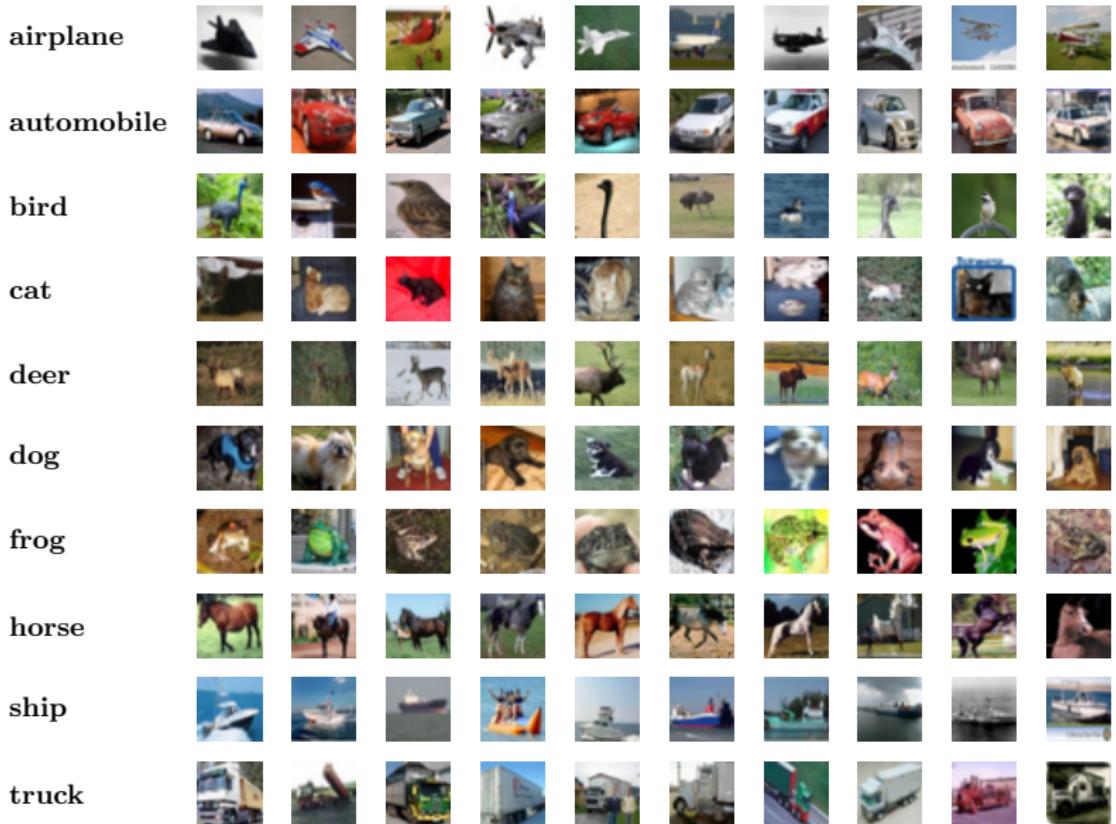
$$\min_{\mathbf{w}, b} \left( \underbrace{\frac{1}{2} \|\mathbf{w}\|^2}_{\text{Regularization term}} + C \sum_{i=1}^n \underbrace{\max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i + b))}_{\text{Hinge loss}} \right)$$

and corresponds to the  $L_2$  **regularization + Hinge loss** formulation!

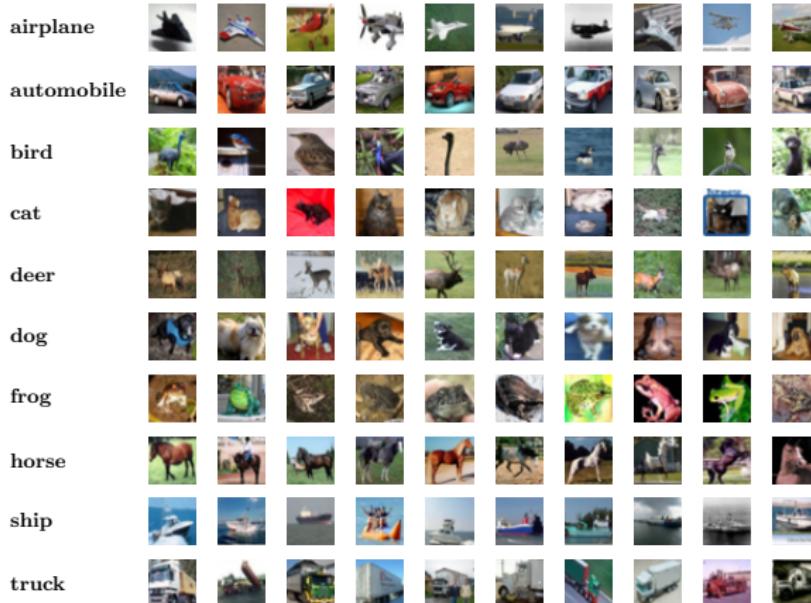
⇒ can train SVMs with SGD/mini-batch gradient descent.

**From binary to multi-class classification**

# Example dataset: CIFAR-10



# Example dataset: CIFAR-10



## CIFAR-10

- 10 classes
- 50,000 training images
- 10,000 test images
- Each image has size  $32 \times 32 \times 3$

# Technical description of the multi-class problem

- Have a set of labelled training examples

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\} \quad \text{with each } \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{1, \dots, C\}.$$

- Want to learn from  $\mathcal{D}$  a classification function

$$g : \begin{matrix} \mathbb{R}^d \\ \uparrow \\ \text{input space} \end{matrix} \times \begin{matrix} \mathbb{R}^P \\ \uparrow \\ \text{parameter space} \end{matrix} \rightarrow \{1, \dots, C\}$$

Usually

$$g(\mathbf{x}; \boldsymbol{\Theta}) = \arg \max_{1 \leq j \leq C} f_j(\mathbf{x}; \boldsymbol{\theta}_j)$$

where for  $j = 1, \dots, C$ :

$$f_j : \mathbb{R}^d \times \mathbb{R}^p \rightarrow \mathbb{R}$$

and  $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_C)$ .

## Multi-class linear classifier

- Let each  $f_j$  be a linear function that is

$$f_j(\mathbf{x}; \boldsymbol{\theta}_j) = h(\mathbf{x}; \boldsymbol{\theta}_j) = \mathbf{w}_j^T \mathbf{x} + b_j$$

- Define

$$f(\mathbf{x}; \boldsymbol{\Theta}) = \begin{pmatrix} h(\mathbf{x}; \boldsymbol{\theta}_1) \\ \vdots \\ h(\mathbf{x}; \boldsymbol{\theta}_C) \end{pmatrix}$$

then

$$f(\mathbf{x}; \boldsymbol{\Theta}) = f(\mathbf{x}; W, \mathbf{b}) = W\mathbf{x} + \mathbf{b}$$

where

$$W = \begin{pmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_C^T \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_C \end{pmatrix}$$

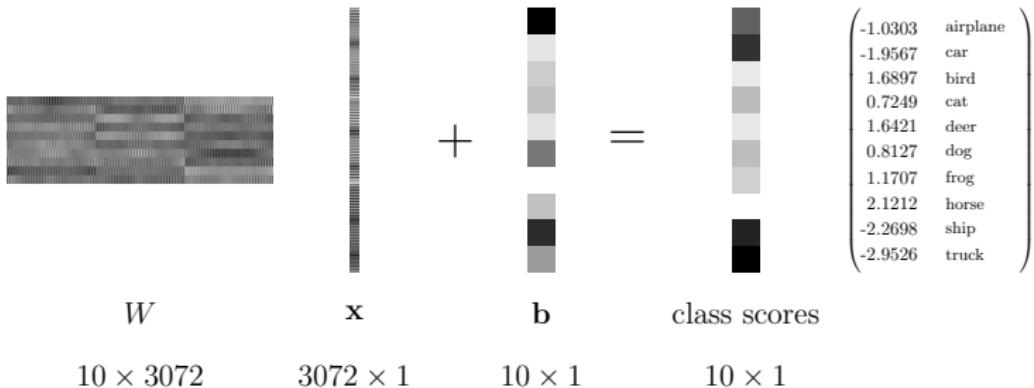
- Note  $W$  has size  $C \times d$  and  $\mathbf{b}$  is  $C \times 1$ .

# Apply a multi-class linear classifier to an image

- Have a 2D colour image but can flatten it into a 1D vector  $\mathbf{x}$



- Apply classifier:  $W\mathbf{x} + \mathbf{b}$  to get a score for each class.



# Interpreting a multi-class linear classifier

- Learn  $W, \mathbf{b}$  to classify the images in a dataset.
- Can interpret each row,  $\mathbf{w}_j^T$ , of  $W$  as a template for class  $j$ .
- Below is the visualization of each learnt  $\mathbf{w}_j$  for CIFAR-10

airplane



car



bird



cat



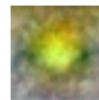
deer



dog



frog



horse



ship

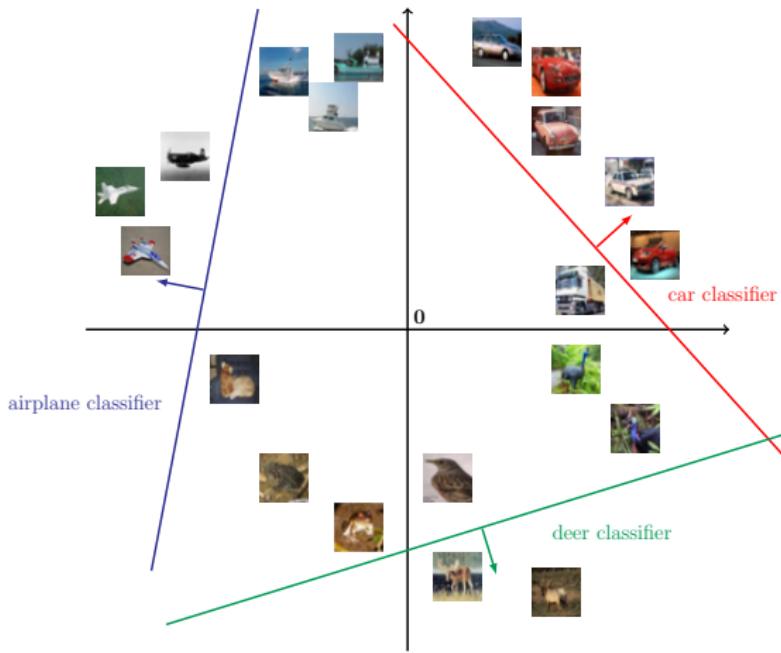


truck



# Interpreting a multi-class linear classifier

- Each  $\mathbf{w}_j^T \mathbf{x} + b_j = 0$  corresponds to a hyperplane,  $H_j$ , in  $\mathbb{R}^d$ .
- $\text{sign}(\mathbf{w}_j^T \mathbf{x} + b_j)$  tells us which side of  $H_j$  the point  $\mathbf{x}$  lies.
- The score  $|\mathbf{w}_j^T \mathbf{x} + b_j| \propto$  the distance of  $\mathbf{x}$  to  $H_j$ .



## How do we learn $W$ and $b$ ?

As before need to

- Specify a loss function (+ a regularization term).
- Set up the optimization problem.
- Perform the optimization.

# How do we learn $W$ and $b$ ?

As before need to

- Specify a loss function
  - must quantify the quality of all the class scores across all the training data.
- Set up the optimization problem.
- Perform the optimization.

## Multi-class loss functions

# Multi-class SVM Loss

- Have training data

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$$

with each  $\mathbf{x}_j \in \mathbb{R}^d$ ,  $y_j \in \{1, \dots, C\}$ .

- Let  $s_i$  be the score of function  $f_i$  applied to  $\mathbf{x}$

$$s_i = f_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + b_i$$

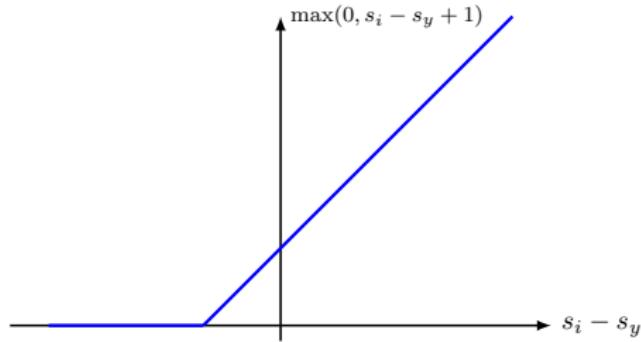
- The SVM loss for labelled example  $(\mathbf{x}, y)$  is

$$l = \sum_{\substack{i=1 \\ i \neq y}}^C \max(0, s_i - s_y + 1)$$

# Multi-class SVM Loss

- $s_i$  is the score of function  $f_i$  applied to  $\mathbf{x}$

$$s_i = f_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + b_i$$

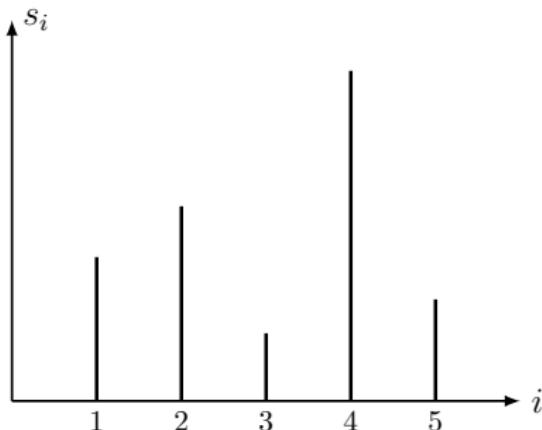


- SVM loss for training example  $\mathbf{x}$  with label  $y$  is

$$l = \sum_{\substack{i=1 \\ i \neq y}}^C \max(0, s_i - s_y + 1)$$

# Multi-class SVM Loss- How it works

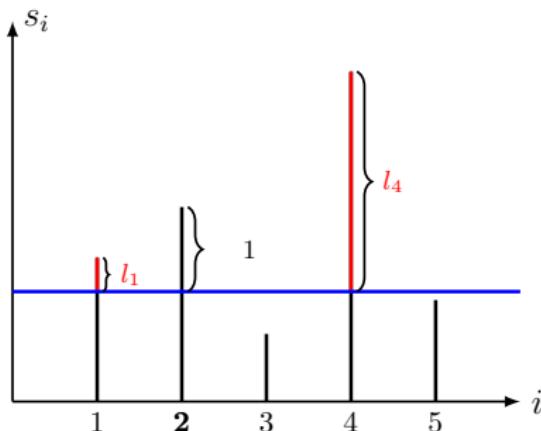
## Simple Example



- Have 5 classes
- Each class has score  $s_i$  for  $i = 1, \dots, 5$

# Multi-class SVM Loss- How it works

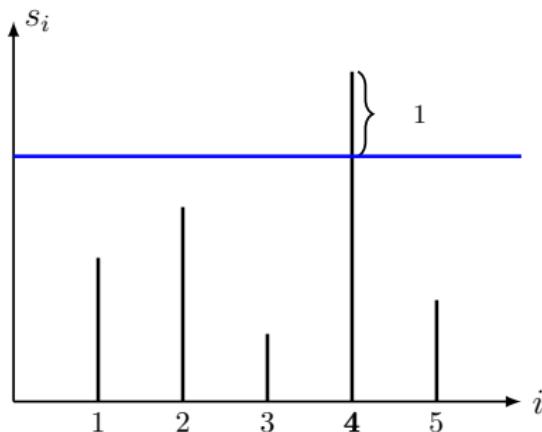
## Simple Example



- Have 5 classes
- Each class has score  $s_i$  for  $i = 1, \dots, 5$
- Say class 2 is the ground truth label.
- Multi-class loss is:  $l = l_1 + l_4$

# Multi-class SVM Loss- How it works

## Simple Example



- Have 5 classes
- Each class gets score  $s_i$  for  $i = 1, \dots, 5$
- Say class 4 is the ground truth label.
- Multi-class loss is:  $l = 0$

# Calculate the multi-class SVM loss for a CIFAR image

input:  $\mathbf{x}$



output

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

label

$$y = 8$$

loss

$$l = \sum_{\substack{i=1 \\ i \neq y}}^{10} \max(0, s_i - s_y + 1)$$

Scores

airplane	-0.3166
car	-0.6609
bird	0.7058
cat	0.8538
deer	0.6525
dog	0.1874
frog	0.6072
horse	<b>0.5134</b>
ship	-1.3490
truck	-1.2225

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

# Calculate the multi-class SVM loss for a CIFAR image

input:  $\mathbf{x}$



output

label

loss

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

$$y = 8$$

$$l = \sum_{\substack{i=1 \\ i \neq y}}^{10} \max(0, s_i - s_y + 1)$$

Scores

Compare to horse score

airplane	-0.3166	0.1701
car	-0.6609	-0.1743
bird	0.7058	1.1925
cat	0.8538	1.3405
deer	0.6525	1.1392
dog	0.1874	0.6741
frog	0.6072	1.0938
horse	<b>0.5134</b>	<b>1.0000</b>
ship	-1.3490	-0.8624
truck	-1.2225	-0.7359

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

$$\mathbf{s} - s_8 + 1$$

# Calculate the multi-class SVM loss for a CIFAR image

input:  $\mathbf{x}$



output

label

loss

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

$$y = 8$$

$$l = \sum_{\substack{i=1 \\ i \neq y}}^{10} \max(0, s_i - s_y + 1)$$

Scores

Compare to horse score

Keep badly performing classes

airplane	-0.3166	0.1701	0.1701
car	-0.6609	-0.1743	0
bird	0.7058	1.1925	1.1925
cat	0.8538	1.3405	1.3405
deer	0.6525	1.1392	1.1392
dog	0.1874	0.6741	0.6741
frog	0.6072	1.0938	1.0938
horse	<b>0.5134</b>	<b>1.0000</b>	<b>1.0000</b>
ship	-1.3490	-0.8624	0
truck	-1.2225	-0.7359	0

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

$$\mathbf{s} - s_8 + 1$$

$$\max(0, \mathbf{s} - s_8 + 1)$$

**Loss for  $\mathbf{x}$ :** 5.4723

# Problem with the SVM loss

Given  $W$  and  $\mathbf{b}$  then

- Response for one training example

$$f(\mathbf{x}; W, \mathbf{b}) = W\mathbf{x} + \mathbf{b} = \mathbf{s}$$

- **Loss for  $(\mathbf{x}, y)$**

$$l(\mathbf{x}, y, W, \mathbf{b}) = \sum_{\substack{i=1 \\ i \neq y}}^C \max(0, s_i - s_y + 1)$$

- **Loss over all the training data**

$$L(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l(\mathbf{x}, y, W, \mathbf{b})$$

Say have found a  $W$  s.t.  $L = 0$ . Is this  $W$  unique?

Let  $W_1 = \alpha W$  and  $\mathbf{b}_1 = \alpha \mathbf{b}$  where  $\alpha > 1$  then

- Response for one training example

$$f(\mathbf{x}; W_1, \mathbf{b}_1) = W_1 \mathbf{x} + \mathbf{b}_1 = \mathbf{s}' = \alpha(W \mathbf{x} + \mathbf{b})$$

- **Loss for**  $(\mathbf{x}, y)$  w.r.t.  $W_1$  and  $\mathbf{b}_1$

$$\begin{aligned} l(\mathbf{x}, y, W_1, \mathbf{b}_1) &= \sum_{\substack{i=1 \\ i \neq y}}^C \max(0, s'_i - s'_y + 1) \\ &= \max(0, \alpha(\mathbf{w}_i^T \mathbf{x} + b_i - \mathbf{w}_y^T \mathbf{x} - b_y) + 1) \\ &= \max(0, \alpha(s_i - s_y) + 1) \\ &= 0 \quad \text{as have assumed } s_i - s_y < -1 \text{ and } \alpha > 1 \end{aligned}$$

$\implies$  Total loss  $L(\mathcal{D}, W_1, \mathbf{b}_1)$  is 0.

# Solution: Weight regularization

$$L(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \sum_{\substack{j=1 \\ j \neq y}}^C \max(0, f_j(\mathbf{x}; W, \mathbf{b}) - f_y(\mathbf{x}; W, \mathbf{b}) + 1) + \lambda R(W)$$

## Commonly used Regularization

Name of regularization	Mathematical def. of $R(W)$
$L_2$	$\sum_k \sum_l W_{k,l}^2$
$L_1$	$\sum_k \sum_l  W_{k,l} $
Elastic Net	$\sum_k \sum_l (\beta W_{k,l}^2 +  W_{k,l} )$

# Cross-entropy loss

# Probabilistic interpretation of scores

Let  $p_i$  be the probability that input  $\mathbf{x}$  has label  $i$ :

$$P_{Y|\mathbf{X}}(i \mid \mathbf{x}) = p_i$$

- For  $\mathbf{x}$  our linear classifier outputs scores for each class:

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

- Can interpret scores,  $\mathbf{s}$ , as:  
**unnormalized log probability for each class.**

$\implies$

$$s_i = \log p'_i \quad \text{with} \quad p_i = p'_i / \sum_k p'_k$$

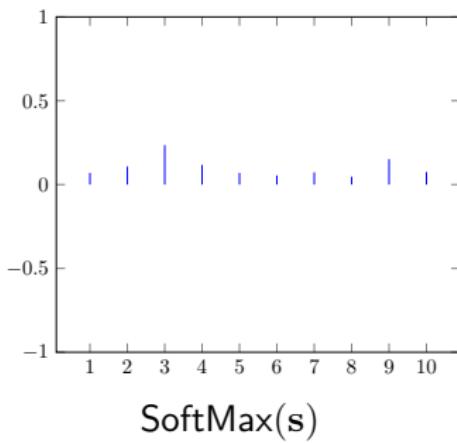
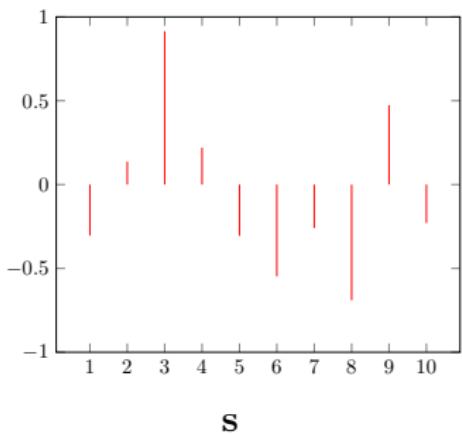
$\implies$

$$P_{Y|\mathbf{X}}(i \mid \mathbf{x}) = p_i = \frac{\exp(s_i)}{\sum_k \exp(s_k)}$$

# SoftMax operation

- This transformation is known as

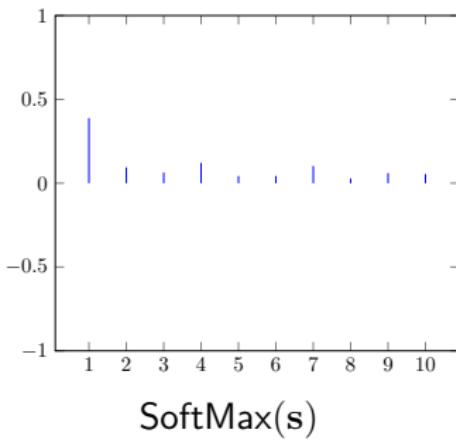
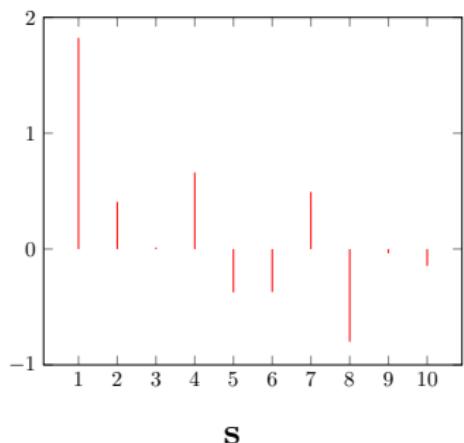
$$\text{SoftMax}(\mathbf{s}) = \frac{\exp(\mathbf{s})}{\sum_k \exp(s_k)}$$



# SoftMax operation

- This transformation is known as

$$\text{SoftMax}(\mathbf{s}) = \frac{\exp(\mathbf{s})}{\sum_k \exp(s_k)}$$



## SoftMax classifier: Log likelihood of the training data

- **Have probabilistic model:** Estimate its parameters by maximizing the log-likelihood of the training data.

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg \max_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log P_{Y|\mathbf{X}}(y \mid \mathbf{x}; \boldsymbol{\theta}) \\ &= \arg \min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log P_{Y|\mathbf{X}}(y \mid \mathbf{x}; \boldsymbol{\theta})\end{aligned}$$

- Given probabilistic interpretation of our classifier:  $\mathbf{s} = W\mathbf{x} + \mathbf{b}$
- The negative log-likelihood of the training data is

$$-\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log \left( \frac{\exp(s_y)}{\sum_{k=1}^C \exp(s_k)} \right)$$

## SoftMax classifier: Log likelihood of the training data

- **Have probabilistic model:** Estimate its parameters by maximizing the log-likelihood of the training data.

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg \max_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log P_{Y|\mathbf{X}}(y \mid \mathbf{x}; \boldsymbol{\theta}) \\ &= \arg \min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log P_{Y|\mathbf{X}}(y \mid \mathbf{x}; \boldsymbol{\theta})\end{aligned}$$

- Given probabilistic interpretation of our classifier:  $\mathbf{s} = W\mathbf{x} + \mathbf{b}$
- The negative log-likelihood of the training data is

$$-\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log \left( \frac{\exp(s_y)}{\sum_{k=1}^C \exp(s_k)} \right)$$

## SoftMax classifier + cross-entropy loss

- Have probabilistic interpretation of classifier:

$$L(\mathcal{D}, W, \mathbf{b}) = -\frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \log \left( \frac{\exp(s_y)}{\sum_{k=1}^C \exp(s_k)} \right)$$

where  $\mathbf{s} = W\mathbf{x} + \mathbf{b}$ .

- Can interpret this log-likelihood as **cross-entropy loss**:

$$L(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{ce}} (\text{SoftMax}(W\mathbf{x} + \mathbf{b}), y)$$

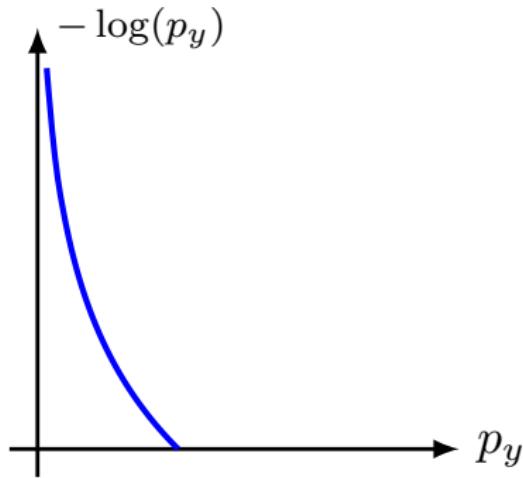
where

$$l_{\text{ce}}(\mathbf{p}, y) = -\log(p_y)$$

# Cross-entropy loss

- $\mathbf{p}$  the probability vector the *network* assigns to  $\mathbf{x}$  for each class

$$\mathbf{p} = \text{SoftMax}(W\mathbf{x} + \mathbf{b})$$



- Cross-entropy loss for training example  $\mathbf{x}$  with label  $y$  is

$$l = -\log(p_y)$$

# Calculate the cross-entropy loss for a CIFAR image

input:  $\mathbf{x}$



output

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

label

$$y = 8$$

loss

$$l = -\log \left( \frac{\exp(s_y)}{\sum \exp(s_k)} \right)$$

Scores

airplane	-0.3166
car	-0.6609
bird	0.7058
cat	0.8538
deer	0.6525
dog	0.1874
frog	0.6072
horse	0.5134
ship	-1.3490
truck	-1.2225

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

# Calculate the cross-entropy loss for a CIFAR image

input:  $\mathbf{x}$



output

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

label

$$y = 8$$

loss

$$l = -\log \left( \frac{\exp(s_y)}{\sum \exp(s_k)} \right)$$

	Scores	$\exp(\text{Scores})$
airplane	-0.3166	0.7354
car	-0.6609	0.5328
bird	0.7058	2.0203
cat	0.8538	2.3583
deer	0.6525	1.9303
dog	0.1874	1.2080
frog	0.6072	1.8319
horse	<b>0.5134</b>	<b>1.7141</b>
ship	-1.3490	0.2585
truck	-1.2225	0.2945

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

$\exp(\mathbf{s})$

# Calculate the cross-entropy loss for a CIFAR image

input:  $\mathbf{x}$



output

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

label

$$y = 8$$

loss

$$l = -\log \left( \frac{\exp(s_y)}{\sum \exp(s_k)} \right)$$

	Scores	$\exp(\text{Scores})$	Normalized scores
--	--------	-----------------------	-------------------

airplane	-0.3166	0.7354	0.0571
car	-0.6609	0.5328	0.0414
bird	0.7058	2.0203	0.1568
cat	0.8538	2.3583	0.1830
deer	0.6525	1.9303	0.1498
dog	0.1874	1.2080	0.0938
frog	0.6072	1.8319	0.1422
horse	0.5134	1.7141	0.1330
ship	-1.3490	0.2585	0.0201
truck	-1.2225	0.2945	0.0229

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

$$\exp(\mathbf{s})$$

$$\frac{\exp(\mathbf{s})}{\sum_k \exp(s_k)}$$

**Loss for  $\mathbf{x}$ :** 2.0171

## Cross-entropy loss

$$l(\mathbf{x}, y, W, \mathbf{b}) = -\log \left( \frac{\exp(s_y)}{\sum_{k=1}^C \exp(s_k)} \right)$$

## Questions

- What is the minimum possible value of  $l(\mathbf{x}, y, W, \mathbf{b})$ ?
- What is the max possible value of  $l(\mathbf{x}, y, W, \mathbf{b})$ ?
- The log of zero is not defined. Could this be a problem?
- Given a random initialization of  $W$  and  $\mathbf{b}$ , what is the loss?
- A training point's input value is changed slightly. Does this have much effect on the loss in some cases?

## Learning the parameters: $W, \mathbf{b}$

- Have training data  $\mathcal{D}$ .
- Have scoring function:

$$\mathbf{s} = f(\mathbf{x}; W, \mathbf{b}) = W\mathbf{x} + \mathbf{b}$$

- We have a choice of loss functions

$$l_{\text{ce}}(\mathbf{x}, y, W, \mathbf{b}) = -\log \left( \frac{\exp(s_y)}{\sum_{k=1}^C \exp(s_k)} \right)$$

$$l_{\text{svm}}(\mathbf{x}, y, W, \mathbf{b}) = \sum_{\substack{i=1 \\ i \neq y}}^C \max(0, s_i - s_y + 1)$$

- Complete training cost

$$J(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{ce}(\text{svm})}(\mathbf{x}, y, W, \mathbf{b}) + \lambda R(W)$$

## Learning the parameters: $W, \mathbf{b}$

- Learning  $W, \mathbf{b}$  corresponds to solving the optimization problem

$$W^*, \mathbf{b}^* = \arg \min_{W, \mathbf{b}} J(\mathcal{D}, W, \mathbf{b})$$

where

$$J(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{ce(svm)}}(\mathbf{x}, y, W, \mathbf{b}) + \lambda R(W)$$

- Know how to solve this! **Mini-batch gradient descent.**
- To implement mini-batch gradient descent need
  - to compute gradient of the loss  $l_{\text{ce(svm)}}(\mathbf{x}, y, W, \mathbf{b})$  and  $R(W)$
  - Set the hyper-parameters of the mini-batch gradient descent procedure.

## Learning the parameters: $W, \mathbf{b}$

- Learning  $W, \mathbf{b}$  corresponds to solving the optimization problem

$$W^*, \mathbf{b}^* = \arg \min_{W, \mathbf{b}} J(\mathcal{D}, W, \mathbf{b})$$

where

$$J(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{ce(svm)}}(\mathbf{x}, y, W, \mathbf{b}) + \lambda R(W)$$

- Know how to solve this! **Mini-batch gradient descent.**
- To implement mini-batch gradient descent need
  - to compute gradient of the loss  $l_{\text{ce(svm)}}(\mathbf{x}, y, W, \mathbf{b})$  and  $R(W)$
  - Set the hyper-parameters of the mini-batch gradient descent procedure.

## Learning the parameters: $W, \mathbf{b}$

- Learning  $W, \mathbf{b}$  corresponds to solving the optimization problem

$$W^*, \mathbf{b}^* = \arg \min_{W, \mathbf{b}} J(\mathcal{D}, W, \mathbf{b})$$

where

$$J(\mathcal{D}, W, \mathbf{b}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{\text{ce(svm)}}(\mathbf{x}, y, W, \mathbf{b}) + \lambda R(W)$$

- Know how to solve this! **Mini-batch gradient descent.**
- To implement mini-batch gradient descent need
  - to compute gradient of the loss  $l_{\text{ce(svm)}}(\mathbf{x}, y, W, \mathbf{b})$  and  $R(W)$
  - Set the hyper-parameters of the mini-batch gradient descent procedure.

We will cover how to compute these gradients using back-propagation.