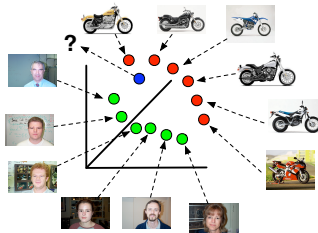


Lecture 2 - Learning Binary & Multi-class Classifiers from Labelled Training Data

DD2424

March 23, 2017

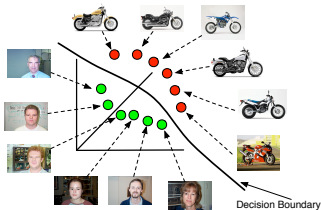
Have labelled training examples



Given a test example how do we decide its class?

High level solution

Technical description of the binary problem



Learn a decision boundary from the labelled training data.

Compare the test example to the decision boundary.

- 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

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

Usually

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

- Have to decide on

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

- Set up an optimization of the form (usually)

$$\arg \max_{\theta} \underbrace{\sum_{(\mathbf{x}, y) \in \mathcal{D}} l(y, f(\mathbf{x}; \theta))}_{\text{training error}} + \lambda \underbrace{R(\theta)}_{\text{regularization term}}$$

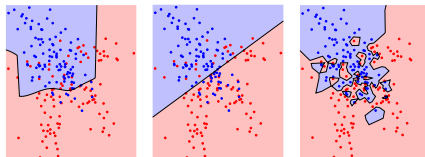
where

- $l(y, f(\mathbf{x}; \theta))$ 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

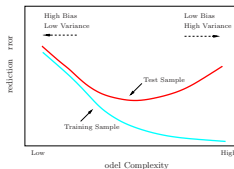
Example of Over and Under fitting



Bayes' Optimal

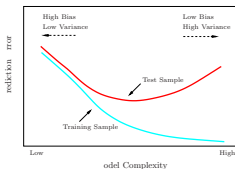
Under-fitting

Over-fitting



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

Overfitting



Linear Decision Boundaries

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

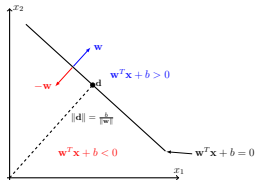
Linear discriminant functions

Pros & Cons of Linear classifiers

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

- Low variance classifier
- Easy to estimate.

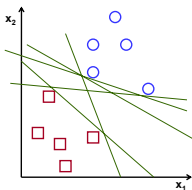
Frequently can set up training so that have an easy optimization problem.

- 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.



Given labelled training data:

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

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**.

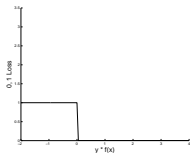
0, 1 Loss function

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

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

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

(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?

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.

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 absolute values.
- or want small absolute values in all entries of \mathbf{w} .

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 do estimate the classifier's parameters by optimizing relative to the above factors?

Squared error loss & no regularization

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

$$\begin{aligned} L(\mathcal{D}, \mathbf{w}, b) &= \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l_{sq}(y, f(\mathbf{x}; \mathbf{w}, b)) \\ &= \frac{1}{2} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \underbrace{((\mathbf{w}^T \mathbf{x} + b) - y)^2}_{\text{Squared error loss}} \end{aligned}$$

L is known as the **sum-of-squares** error function.

- The \mathbf{w}^*, b^* that minimizes $L(\mathcal{D}, \mathbf{w}, b)$ is known as the **Minimum Squared Error** solution.
- This minimum is found as follows....

Example: Squared Error loss

- Have a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that is $f(\mathbf{x}) = b$
- 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

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

Technical interlude: Matrix Calculus

Matrix Calculus

Matrix Calculus

- Have a function $f : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$ that is $f(X) = b$ with $X \in \mathbb{R}^{d \times d}$

- We use the notation

$$\frac{\partial f}{\partial X} = \begin{pmatrix} \frac{\partial f}{\partial x_{11}} & \frac{\partial f}{\partial x_{12}} & \cdots & \frac{\partial f}{\partial x_{1d}} \\ \frac{\partial f}{\partial x_{21}} & \frac{\partial f}{\partial x_{22}} & \cdots & \frac{\partial f}{\partial x_{2d}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{d1}} & \frac{\partial f}{\partial x_{d2}} & \cdots & \frac{\partial f}{\partial x_{dd}} \end{pmatrix}$$

- **Example:** If $f(X) = \mathbf{a}^T X \mathbf{b} = \sum_{i=1}^d a_i \sum_{j=1}^d x_{ij} b_j$ then

$$\frac{\partial f}{\partial x_{ij}} = a_i b_j \implies \frac{\partial f}{\partial X} = \begin{pmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_d \\ \vdots & \vdots & \vdots & \vdots \\ a_d b_1 & a_d b_2 & \cdots & a_d b_d \end{pmatrix} = \mathbf{a} \mathbf{b}^T$$

Derivative of a linear function

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \mathbf{a} \quad (1)$$

$$\frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \quad (2)$$

$$\frac{\partial \mathbf{a}^T X \mathbf{b}}{\partial X} = \mathbf{a} \mathbf{b}^T \quad (3)$$

$$\frac{\partial \mathbf{a}^T X^T \mathbf{b}}{\partial X} = \mathbf{b} \mathbf{a}^T \quad (4)$$

Derivative of a quadratic function

$$\frac{\partial \mathbf{x}^T B \mathbf{x}}{\partial \mathbf{x}} = (B + B^T) \mathbf{x} \quad (5)$$

$$\frac{\partial \mathbf{b}^T X^T X \mathbf{c}}{\partial X} = X(\mathbf{b} \mathbf{c}^T + \mathbf{c} \mathbf{b}^T) \quad (6)$$

$$\frac{\partial (B\mathbf{x} + \mathbf{b})^T C (D\mathbf{x} + \mathbf{d})}{\partial \mathbf{x}} = B^T C (D\mathbf{x} + \mathbf{d}) + D^T C^T (B\mathbf{x} + \mathbf{b}) \quad (7)$$

$$\frac{\partial \mathbf{b}^T X^T D X \mathbf{c}}{\partial X} = D^T X \mathbf{b} \mathbf{c}^T + D X \mathbf{c} \mathbf{b}^T \quad (8)$$

End of Technical interlude

Pseudo-Inverse solution

Pseudo-Inverse solution

- Can write the cost function as

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

$$\text{where } \mathbf{x}' = (\mathbf{x}^T, 1)^T, \mathbf{w}_1 = (\mathbf{w}^T, b)^T$$

- Writing in matrix notation this becomes

$$\begin{aligned} L(\mathcal{D}, \mathbf{w}_1) &= \frac{1}{2} \|\mathbf{X} \mathbf{w}_1 - \mathbf{y}\|^2 = \frac{1}{2} (\mathbf{X} \mathbf{w}_1 - \mathbf{y})^T (\mathbf{X} \mathbf{w}_1 - \mathbf{y}) \\ &= \frac{1}{2} (\mathbf{w}_1^T \mathbf{X}^T \mathbf{X} \mathbf{w}_1 - 2 \mathbf{y}^T \mathbf{X} \mathbf{w}_1 + \mathbf{y}^T \mathbf{y}) \end{aligned}$$

where

$$\mathbf{y} = (y_1, \dots, y_n)^T, \quad \mathbf{w} = (w_1, \dots, w_{d+1})^T, \quad X = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^T & 1 \end{pmatrix}$$

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

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

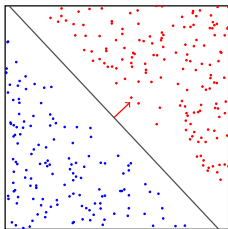
- Setting this equal to zero yields $\mathbf{X}^T \mathbf{X} \mathbf{w}_1 = \mathbf{X}^T \mathbf{y}$ and

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

where

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

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



Decision boundary found by minimizing

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

- 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

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

- 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

- Common approach to solving such unconstrained optimization problem is iterative non-linear optimization.

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

- Start with an estimate $\mathbf{x}^{(0)}$.
- Try to 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

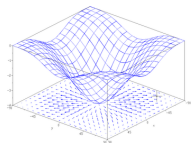
Choose a search direction based on the local properties of the cost function.

Line Search

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

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.

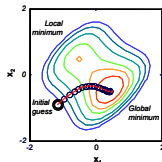
Gradient Descent Minimization

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

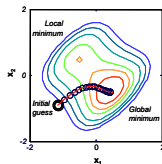
Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent.

Gradient Descent Minimization Properties

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

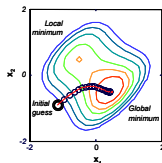
But this is okay

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

End of Technical interlude

Gradient descent solution

The error function $L(\mathcal{D}, \mathbf{w}_1)$ could also be minimized wrt \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 working with large matrices.

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.

Gradient descent solution

- Increase the number of updates per computation by considering 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

Given large scale data

- **Form of the optimization problem:**

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

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

- **Solution with gradient descent**

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

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

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

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

- Start with a random solution $\theta^{(0)}$.

- 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

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

- When $|\mathcal{D}^{(t)}| = 1$:

$$\nabla_{\theta} J(\mathcal{D}^{(t)}, \theta) \Big|_{\theta^{(t)}} \text{ a noisy estimate of } \nabla_{\theta} J(\mathcal{D}, \theta) \Big|_{\theta^{(t)}}$$

- Therefore

$|\mathcal{D}|$ **noisy** update steps in SGD \approx 1 **correct** update step in GD.

- In practice SGD converges a lot faster than GD.

- Given lots of labelled training data:

Quantity of updates more important than **quality** of updates!

Best practices for SGD

Mini-Batch Gradient Descent

- **Preparing the data**

- Randomly shuffle the training examples and zip sequentially through \mathcal{D} .
- Use preconditioning 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.

- Start with a random guess $\theta^{(0)}$ 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 $n_b \approx 150$.)

2. Update parameter estimate with

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

- 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 η 's \implies potentially faster learning but with the risk of less stable convergence.
- Smaller η '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 the near future.

End of Technical interlude

Squared Error loss + L_2 regularization

- Add a regularization term to the loss function

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

$$= \frac{1}{2} \|\mathbf{X}\mathbf{w} + b\mathbf{1} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$$

where $\lambda > 0$ and small and \mathbf{X} is the data matrix

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

- Add a regularization term to the loss function

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

- Let's centre the input data

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

$$\Rightarrow \mathbf{X}_c^T \mathbf{1} = \mathbf{0}.$$

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

$$\frac{\partial J_{\text{ridge}}}{\partial b} = b\mathbf{1}^T \mathbf{1} + \mathbf{w}^T \mathbf{X}_c^T \mathbf{1} - \mathbf{1}^T \mathbf{y}$$

$$= b\mathbf{1}^T \mathbf{1} - \mathbf{1}^T \mathbf{y}$$

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

Solving ridge regression: Optimal weight vector

Simple 2D Example

- Add a regularization term to the loss function

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

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

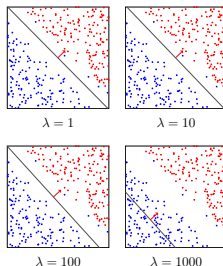
$$\frac{\partial J_{\text{ridge}}}{\partial \mathbf{w}} = (\mathbf{X}_c^T \mathbf{X}_c + \lambda \mathbf{I}_d) \mathbf{w} - \mathbf{X}_c^T \mathbf{y}$$

- Set to zero to get

$$\mathbf{w}^* = (\mathbf{X}_c^T \mathbf{X}_c + \lambda \mathbf{I}_d)^{-1} \mathbf{X}_c^T \mathbf{y}$$

- $(\mathbf{X}_c^T \mathbf{X}_c + \lambda \mathbf{I}_d)$ has a unique inverse even if $\mathbf{X}_c^T \mathbf{X}_c$ is singular.

Ridge Regression decision boundaries as λ is varied



- Add a regularization term to the loss function

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

- Compute the gradient of J_{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.

Hinge Loss

- The gradient-descent update step is

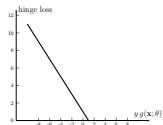
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left[(X_c^T X_c + \lambda I_d) \mathbf{w}^{(t)} - X_c^T \mathbf{y} \right]$$

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

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

The Hinge loss

$$l(\mathbf{x}, y; \mathbf{w}, b) = \max \{0, 1 - y(\mathbf{w}^T \mathbf{x} + b)\}$$



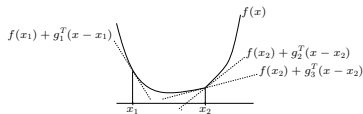
- This loss is not differentiable but is convex.
- Correctly classified examples *sufficiently* far from the decision boundary have zero loss.
- \implies have a way of choosing between classifiers that correctly classify all the training examples.

- g is a **subgradient** of f at x if

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

Technical interlude: Sub-gradient

- 1D example:

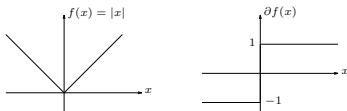


- g_2, g_3 are subgradients at x_2 ;
- g_1 is a subgradient at x_1 .

Subgradient of a function

- Set of all subgradients of f at x is called the subdifferential of f at x , written $\partial f(x)$

- 1D example:



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

End of Technical interlude

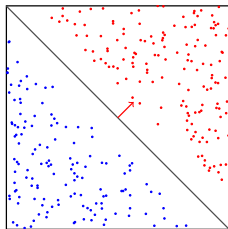
- Find \mathbf{w}, b that minimize

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

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

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

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



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

- Find \mathbf{w}, b that minimize

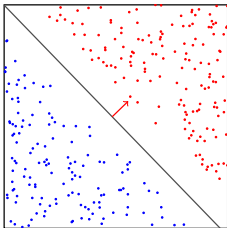
$$J_{\text{sum}}(\mathcal{D}, \mathbf{w}, b) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{(\mathbf{x}, y) \in \mathcal{D}} \underbrace{\max\{0, 1 - y(\mathbf{w}^T \mathbf{x} + b)\}}_{\text{Hinge Loss}}$$

L_2 Regularization + Hinge Loss

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

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

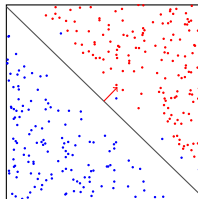
$$\frac{\partial l(\mathbf{x}, y; \mathbf{w}, b)}{\partial b} = \begin{cases} -y & \text{if } y(\mathbf{w}^T \mathbf{x} + b) > 1 \\ 0 & \text{otherwise.} \end{cases}$$



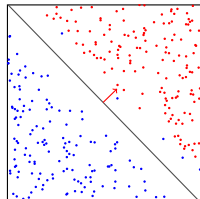
Decision boundary found with SGD by minimizing ($\lambda = .01$)

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

Decision boundaries found by minimizing



Hinge Loss



L_2 Regularization + Hinge Loss

“ L_2 Regularization + Hinge Loss” \equiv SVM

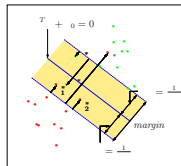
SVM's constrained optimization problem

SVM solves this constrained optimization problem:

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

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

$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, n.$$



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:

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

- 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!

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

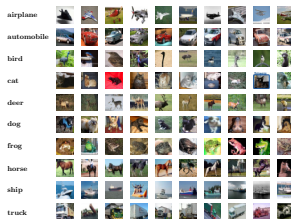
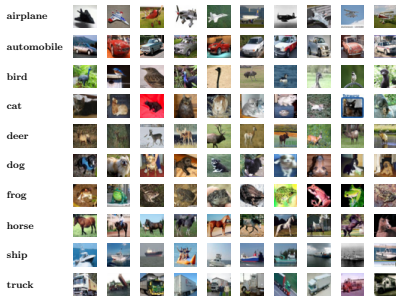
Thus the original **constrained optimization problem** can be restated as an **unconstrained optimization problem**:

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

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

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

From binary to multi-class classification

**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

Multi-class linear classifier

- 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

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

Usually

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

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

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

and $\Theta = (\theta_1, \theta_2, \dots, \theta_C)$.

- Let each f_j be a linear function that is

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

- Define

$$f(\mathbf{x}; \Theta) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_C(\mathbf{x}) \end{pmatrix}$$

then

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

where

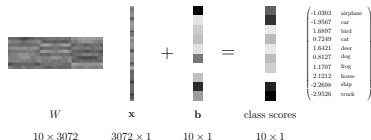
$$W = \begin{pmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_C^T \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_C \end{pmatrix}$$

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

- 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.



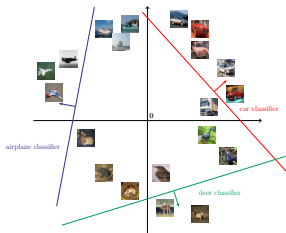
- Learn W, \mathbf{b} to classify the images in a dataset.
- Can interpret each row, \mathbf{w}_j , of W as a template for class j .
- Below is the visualization of each learnt \mathbf{w}_j for CIFAR-10



Interpreting a multi-class linear classifier

How do we learn W and \mathbf{b} ?

- 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 .



As before need to

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

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

- Remember have training data

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

- Let s_j be the score of function f_j applied to \mathbf{x}

$$s_j = f_j(\mathbf{x}; \mathbf{w}_j, b_j) = \mathbf{w}_j^T \mathbf{x} + b_j$$

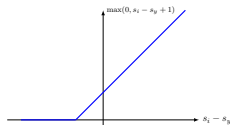
- The SVM loss for training example \mathbf{x} with label y is

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

Multi-class SVM Loss

- s_j is the score of function f_j applied to \mathbf{x}


$$s_j = f_j(\mathbf{x}; \mathbf{w}_j, b_j) = \mathbf{w}_j^T \mathbf{x} + b_j$$




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

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


Calculate the multi-class SVM loss for a CIFAR image

input: \mathbf{x}	output	label	loss
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$y = 8$	$l = \sum_{j=1}^{10} \max(0, s_j - s_y + 1)$
	Scores		
airplane	-0.3166		
car	-0.6609		
bird	0.7058		
cat	0.8538		
deer	0.6525		
dog	0.1874		
frog	0.6072		
horse	0.5134		
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
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$y = 8$	$l = \sum_{j=1}^{10} \max(0, s_j - s_y + 1)$
	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	0.5134	1.0000	
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
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$y = 8$	$l = \sum_{j=1}^{10} \max(0, s_j - s_y + 1)$
	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	0.5134	1.0000	1.0000
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}

$$l(\mathbf{x}, y, W, \mathbf{b}) = \sum_{j=1}^C \max(0, s_j - s_y + 1)$$

- Loss over all the training data

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

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

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

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

$$\begin{aligned} l(\mathbf{x}, y, W_1, \mathbf{b}_1) &= \sum_{\substack{j=1 \\ j \neq y}}^C \max(0, s'_j - s'_y + 1) \\ &= \max(0, \alpha(\mathbf{w}_j^T \mathbf{x} + b_j - \mathbf{w}_y^T \mathbf{x} - b_y) + 1) \\ &= \max(0, \alpha(s_j - s_y) + 1) \\ &= 0 \quad \text{as by definition } s_j - s_y < -1 \text{ and } \alpha > 1 \end{aligned}$$

- Thus the total loss $L(\mathcal{D}, W_1, \mathbf{b}_1)$ is 0.

Cross-entropy Loss

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

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	$\sum_k \sum_l (\beta W_{k,l}^2 + W_{k,l})$

Probabilistic interpretation of scores

Let p_j be the probability that input \mathbf{x} has label j :

$$P_{Y|\mathbf{X}}(j | \mathbf{x}) = p_j$$

- 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

$$s_j = \log p'_j$$

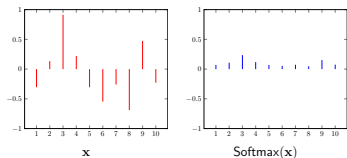
where $\alpha p'_j = p_j$ and $\alpha = \sum p'_j$.

\implies

$$P_{Y|\mathbf{X}}(j | \mathbf{x}) = p_j = \frac{\exp(s_j)}{\sum_k \exp(s_k)}$$

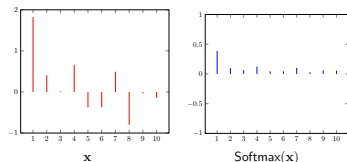
- This transformation is known as

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



- This transformation is known as

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



Softmax classifier: Log likelihood of the training data

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

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

- Given probabilistic interpretation of our classifier, the negative log-likelihood of the training data is

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

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

Softmax classifier: Log likelihood of the training data

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

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

- Given probabilistic interpretation of our classifier, the negative log-likelihood of the training data is

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

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

- Given the probabilistic interpretation of our classifier, the negative log-likelihood of the training data is

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

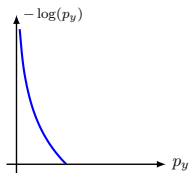
where $\mathbf{s} = W\mathbf{x} + \mathbf{b}$.

- Can also interpret this in terms of the cross-entropy loss:

$$\begin{aligned} L(\mathcal{D}, W, \mathbf{b}) &= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \underbrace{-\log \left(\frac{\exp(s_y)}{\sum_{k=1}^C \exp(s_k)} \right)}_{\text{cross-entropy loss for } (\mathbf{x}, y)} \\ &= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}} l(\mathbf{x}, y, W, \mathbf{b}) \end{aligned}$$

- \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	label	loss
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$y = 8$	$l = -\log \left(\frac{\exp(s_y)}{\sum \exp(s_k)} \right)$

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	label	loss
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$y = 8$	$l = -\log \left(\frac{\exp(s_y)}{\sum \exp(s_k)} \right)$


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	0.5134	1.7141
ship	-1.3490	0.2585
truck	-1.2225	0.2945

$\exp(\text{Scores})$

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

$\exp(\mathbf{s})$

input: \mathbf{x}	output	label	loss
	$\mathbf{s} = W\mathbf{x} + \mathbf{b}$	$y = 8$	$l = -\log\left(\frac{\exp(s_y)}{\sum \exp(s_k)}\right)$

	Scores	exp(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 $l(\mathbf{x}, y, W, \mathbf{b})$?
- At initialization all the entries of W are small \implies all $s_k \neq 0$. What is the loss?
- A training point's input value is changed slightly. What happens to the loss?
- The log of zero is not defined. Could this be a problem?

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

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

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

- Complete training loss

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

Learning the parameters: W, \mathbf{b}

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

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

where

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

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

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

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

where

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

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

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

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

where

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

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

Next Lecture

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