



Deep Learning for Visual Computing

Features, Parametric Models, Loss Functions

Christopher Pramerdorfer
Computer Vision Lab, TU Wien

Topics

Feature Extraction

Parametric models

Linear models for classification

- ▶ Softmax classifiers

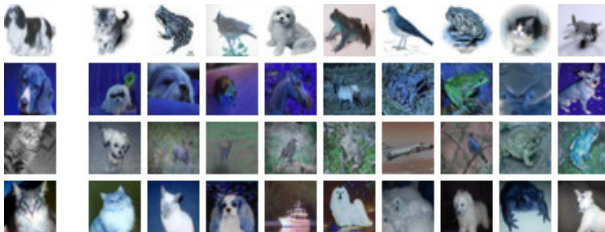
Loss functions

Feature Extraction

Motivation

Our k NN classifier for cats and dogs performs poorly

- ▶ On CIFAR10 dataset, test accuracy is about 40%
- ▶ Humans achieve about 94%, similar to CNNs



Feature Extraction

Motivation

Several reasons (later)

One is that we use raw images as **feature vector** \mathbf{x}

- ▶ But k NN has no understanding of images

Implications

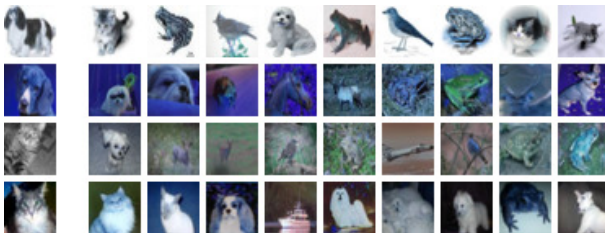
- ▶ Input features \mathbf{x} (pixel values) are indiscriminative
- ▶ D is large (**curse of dimensionality**)

Feature Extraction

Motivation

Classification based on overall image similarity

- ▶ No concept of objects
- ▶ Background has strong effect (many pixels)



Feature Extraction

Motivation

Raw pixel values are poor features

- ▶ A **feature** is certain property of data

Goal of **feature extraction**

- ▶ Extract discriminative features from images
- ▶ **Discriminative** features help distinguish between classes

Feature Extraction

HOG (Histogram of Oriented Gradients) Feature Extractor

Steps

- ▶ Compute gradient magnitude and orientation
- ▶ Divide image into overlapping cells
- ▶ Compute histogram of quantized orientations in every cell
- ▶ Combine cells to blocks, concatenate histograms
- ▶ Normalize blocks, concatenate to single vector

See [1] for details

Feature Extraction

HOG Feature Extractor

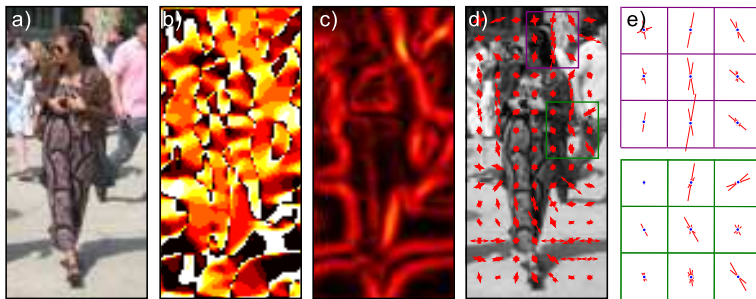


Image from [2]

Feature Extraction

Low-Level Features

HOG features improve performance by about 10%

- ▶ Still not close to human and CNN levels

Reasons

- ▶ k NN is a simple classifier
- ▶ HOG features are low-level

Feature Extraction

Low-Level Features

HOG features are **low-level**

- ▶ Capture directions of local brightness changes
- ▶ Generally useful but not task-specific

This applies to all manually designed feature extractors

- ▶ Impossible to design optimal task-specific extractors

Feature Extraction

High-Level Features

We want task-specific **high-level features**

- ▶ Features that carry semantics
- ▶ E.g. presence of certain part of a human face

We cannot design them, so they must be learned

- ▶ CNNs are able to do this in a robust way
- ▶ Main reason why CNNs are so powerful

Parametric Models

Motivation

k NN classifier has several limitations

- ▶ Must keep all training data for testing
- ▶ Testing (predicting class labels) is slow
- ▶ Not particularly powerful

Parametric models overcome these limitations

Parametric Models

Definition

Let $\mathbf{x} \in \mathbb{R}^D$ be input and $\mathbf{w} \in \mathbb{R}^T$ be output

A **model** describes family of functions from \mathbf{x} to \mathbf{w}

- ▶ Particular function $f : \mathbf{x} \mapsto \mathbf{w}$ learned during training

Model defines the **hypothesis space** of a ML algorithm

- ▶ Set of functions allowed as solution
- ▶ Extending family increases capacity

Parametric Models

Definition

In **parametric models** f depends on **parameters** θ

- ▶ We write $\mathbf{w} = f(\mathbf{x}; \theta)$
- ▶ Training entails finding good parameters
- ▶ Training set can be discarded after training

CNNs are parametric models

- ▶ Number of parameters can exceed 100 million (!)

Parametric Models

Linear Models

The most basic example are **linear models**

Hypothesis space comprises linear functions from \mathbb{R}^D to \mathbb{R}^T

- Formally $\mathbf{w} = f(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ with $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$

$\mathbf{W} \in \mathbb{R}^{T \times D}$ is called **weight matrix**

$\mathbf{b} \in \mathbb{R}^T$ is called **bias vector**

Parametric Models

Linear Models

Used in almost all CNNs ([linear layers](#))

- ▶ In the classifier stage
- ▶ For final mapping to class labels w

Linear Models for Classification

Definition of \mathbf{w}

We could directly estimate class label, $w \in \{1, \dots, T\}$

- ▶ Limited information, complicates training

Instead we let $\mathbf{w} = (w_1, \dots, w_T)^T$

- ▶ Want $\arg \max(\mathbf{w}) = c$ if image \mathbf{x} belongs to class c
- ▶ \mathbf{w} encodes **class scores** (confidences)

Linear Models for Classification

Example

A (unsuccessful) example prediction

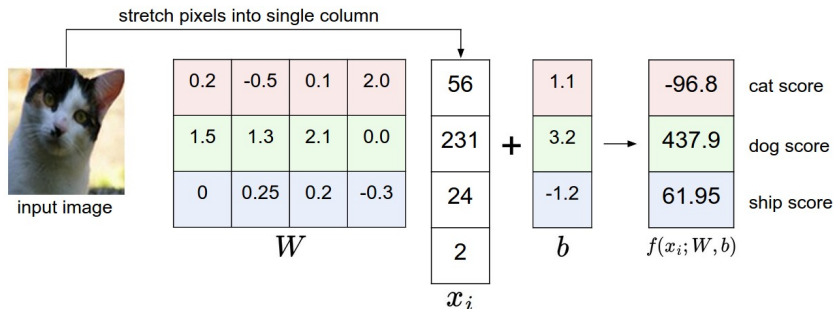


Image from [cs231n.github.io](https://github.com/cs231n)

Linear Models for Classification

Example

Can think of T independent classifiers, one per class

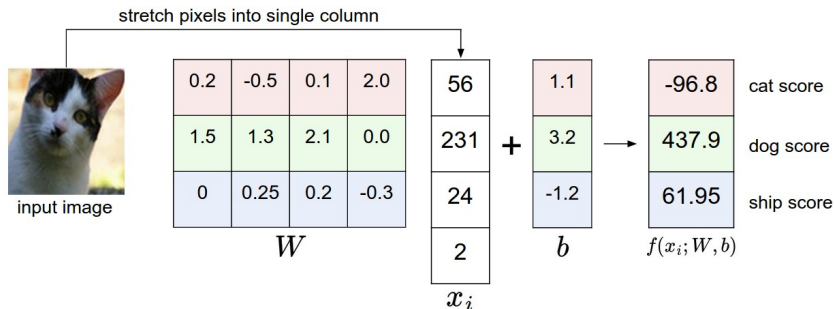


Image from [cs231n.github.io](https://github.com/cs231n)

Linear Models for Classification

Geometric Interpretation

T hyperplanes as decision boundaries in \mathbb{R}^D

- Weights define orientation, bias defines offset from 0

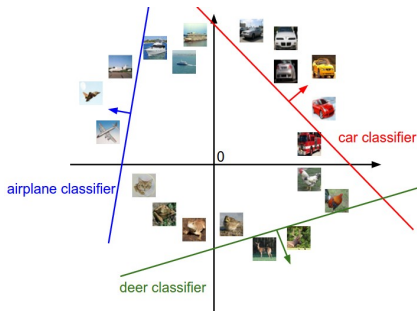


Image from [cs231n.github.io](https://github.com/cs231n)

Linear Models for Classification

Geometric Interpretation

Hyperplane c should answer “is x of class c ?”

- ▶ Want x on positive side of hyperplane c iff of class c

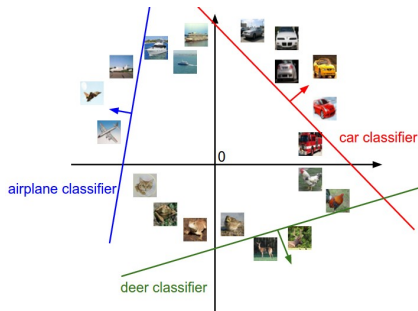


Image from cs231n.github.io

Linear Models for Classification

Geometric Interpretation

\mathbf{x} is on positive side if $\mathbf{w}_c \mathbf{x} + b_c \geq 0$

- ▶ \mathbf{w}_c is row c of \mathbf{W}

Left side equals score of class c , that is w_c

Signed distance to plane is $w_c / \|\mathbf{w}_c\|$

- ▶ Hence class score increases with distance/confidence

Linear Models for Classification

Geometric Interpretation

During training, make hyperplanes always answer correctly

- ▶ Only possible if classes are **linearly separable**

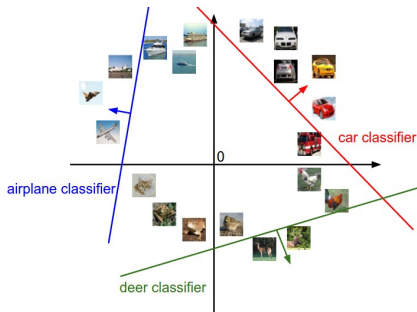


Image from cs231n.github.io

Linear Models for Classification

Geometric Interpretation

Hyperplanes do not work together

- ▶ Each hyperplane is an independent **binary** classifier

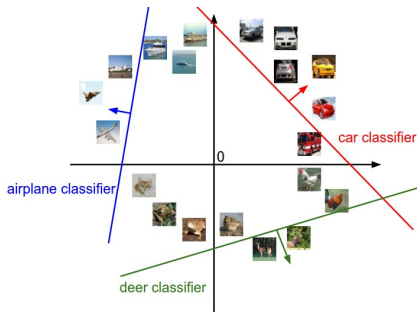


Image from cs231n.github.io

Linear Models for Classification

Template-Matching Interpretation

T learned **templates** that are matched with input images

- ▶ Each \mathbf{w}_c encodes a template
- ▶ Matching using inner product $\mathbf{w}_c \mathbf{x}$ (plus b_c)
- ▶ Class score increases with similarity of image to template



Image from [cs231n.github.io](https://github.com/cs231n)

Linear Models for Classification

Template-Matching Interpretation

Most templates have clear interpretation

- ▶ Horse template shows something horse-like
- ▶ Most cars training data seem to be red
- ▶ Background is (again) very dominant (sky, grass, water)



Image from [cs231n.github.io](https://github.com/cs231n)

Linear Models for Classification

Template-Matching Interpretation

Linear classifiers cannot properly model intraclass variation

- ▶ Templates merge modes of variation
- ▶ What about blue cars, planes on ground, gray horses?

CNNs solve this problem



Image from cs231n.github.io

Linear Models for Classification

Remarks

Much faster than k NN classifier

- ▶ Single matrix-vector multiplication

Number of parameters governed by D and T

- ▶ $\mathbf{W} \in \mathbb{R}^{T \times D}$ and $\mathbf{b} \in \mathbb{R}^T$
- ▶ CIFAR10 : $T = 10$ and $D = 3,072$, so 30,730 parameters

Linear Models for Classification

Remarks

Can apply the **bias trick** to simplify f to $f(\mathbf{x}) = \mathbf{W}\mathbf{x}$

- Append \mathbf{b} to \mathbf{W} , append 1 to \mathbf{x}

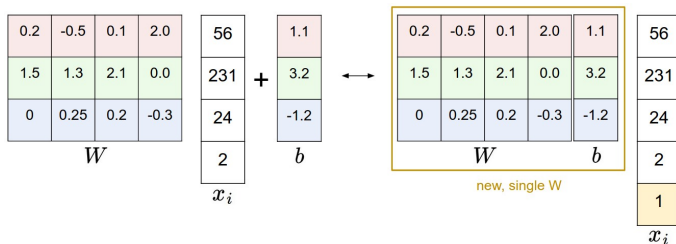


Image from cs231n.github.io

Loss Functions

For training any parametric model, we need

- ▶ A loss function
- ▶ An optimization algorithm

Loss Functions

Recall that parametric models have form $f(\mathbf{x}; \boldsymbol{\theta})$

A **loss function** $L(\boldsymbol{\theta})$ (or **cost** or **objective** function)

- ▶ Measures performance of $f(\cdot; \boldsymbol{\theta})$ (lower loss is better)
- ▶ On some dataset $\mathcal{D} = \{(\mathbf{x}_s, \mathbf{w}_s)\}_{s=1}^S$
- ▶ With respect to parameters $\boldsymbol{\theta}$

Choice of L depends on task

Goal during training is to change $\boldsymbol{\theta}$ to minimize $L(\boldsymbol{\theta})$

Loss Functions

Cross-Entropy Loss

For classification, most popular choice is **cross-entropy loss**

- ▶ Standard loss for training CNNs

Dissimilarity between probability distributions p and q

- ▶ In terms of **cross-entropy** $H(p,q) = -\sum_x (p(x) \log q(x))$

Loss Functions

Cross-Entropy Loss

p and q are discrete probability distributions

- ▶ So $p(x) \geq 0$ and $\sum_x p(x) = 1$ (same for q)

Thus $\log q(x)$ is always negative or 0, as are summands

- ▶ Cross-entropy is always positive
- ▶ Generally not 0 even if $p = q$ (but entropy of p)

Loss Functions

Cross-Entropy Loss

In our case

- ▶ $p \sim \mathbf{w}_s$, encoding the true class distribution
- ▶ $q \sim \mathbf{w}$, the predicted class distribution

Use **one-hot encoding** to obtain \mathbf{w}_s from single label w_s

- ▶ Vector of size T that is zero everywhere
- ▶ Except for element $c = 1$ if $w_s = c$
- ▶ \mathbf{w}_s is valid probability distribution

Loss Functions

Cross-Entropy Loss

Predictions \mathbf{w} are not valid probability distributions

- ▶ Class-scores are unbounded
- ▶ \mathbf{w} generally does not sum to 1

We solve this problem by

- ▶ Regarding \mathbf{w} as unnormalized log probabilities
- ▶ And using the [softmax function](#) for normalization

$$\text{softmax}_k(\mathbf{w}) = \frac{\exp(w_k)}{\sum_{t=1}^T \exp(w_t)}$$

Loss Functions

Cross-Entropy Loss

Softmax transforms any vector $\mathbf{w} \in \mathbb{R}^T$

- ▶ Such that $w_t \geq 0$ and $\sum_{t=1}^T w_t = 1$

$$\text{softmax}\left(\begin{pmatrix} -1 \\ 3 \\ 1 \end{pmatrix}\right) \approx \begin{pmatrix} 0.016 \\ 0.867 \\ 0.117 \end{pmatrix}$$

Loss Functions

Cross-Entropy Loss

We calculate the cross-entropy loss on \mathcal{D} as

$$L(\theta) = \frac{1}{S} \sum_{s=1}^S H(\mathbf{w}_s, \text{softmax}(f(\mathbf{x}_s; \theta)))$$

Minimize the average cross-entropy over whole dataset

- ▶ Corresponds to **maximum likelihood estimation (MLE)**

Loss Functions

Cross-Entropy Loss

MLE $\hat{\theta}$ is optimal in terms of data

- ▶ Choice under which observed data \mathcal{D} is most likely

Best we can do unless we have prior information about θ

- ▶ We'll talk about this later (regularization)

Loss Functions

Cross-Entropy Loss

Resulting classifier is called **softmax classifier**

- ▶ f still produces unnormalized log probabilities
- ▶ Must apply softmax to obtain probabilities

Loss Functions

Now that we have a good loss function, we can minimize it

- ▶ For this we need a suitable optimization algorithm
- ▶ Next lecture

Bibliography

- [1] N. Dalal and B. Triggs, *Histograms of oriented gradients for human detection*, CVPR.
- [2] Prince, *Computer Vision Models*. 2012.