Deep Learning for Visual Computing

Machine Learning for Image Classification

Christopher Pramerdorfer

Computer Vision Lab, TU Wien

Image from reddit, created using midjourney

This Week in AI
Midjourney 5

Image from reddit

Topics

Machine learning for image classification (recap)

- Feature extraction
- Linear models
- Cross-entropy loss
- Gradient descent

Machine Learning for Image Classification

Recall that we want to build an image classifier

- Should support the classes {dog, cat}
- Using the CIFAR-10 dataset (6000 images per class)
- Manual approach failed

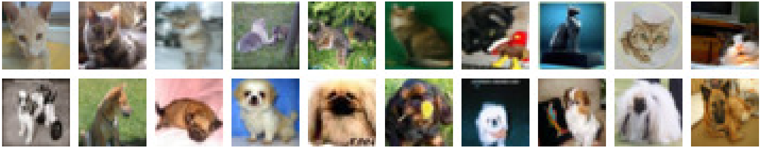


Image from cs.toronto.edu

Machine Learning for Image Classification

Perhaps machine learning (ML) can help us

- ML algorithms are able to learn from data
- Such that performance improves with experience

Approach task as supervised ML problem:

- Encode class labels as one-hot vectors $\mathbf{o}$
- Extract discriminative features $\mathbf{x}$ from images
- Train a ML model (classifier) to predict $\mathbf{o}$ from $\mathbf{x}$
- Ensure it generalizes to unseen data

One-Hot Encodings

One-hot encodings map categorical to numerical variables

- Most ML models expect numerical inputs

Represent each category by a binary vector $\mathbf{o}$

- In our case cat $:= (1, 0)$ and dog $:= (0, 1)$
- $\dim(\mathbf{o})$ equals number of categories

Note that $\mathbf{o}$ is valid probability mass function

- Will become handy during training and inference

Feature Extraction
Features

Extract feature vectors $\mathbf{x}$ from inputs

- We want to extract good features
- Better features improve model performance

Good features are usually high-level features

- Tell us something about the world depicted
- Task-specific (e.g. presence of pointy ears, whiskers)

We cannot write such high-level feature extractors

- But deep learning models can learn how to do so

Before deep learning there we were stuck with low-level features

- Manually designed feature extractors
- Encode properties about the image itself

Brightness changes at certain scale and/or orientation are popular

- Invariant to additive luminance changes
- Response at (certain) object borders, textured regions

This is where image processing comes in

Feature Extraction
Low-Level Features – Gabor Filters

Linear filters modeled after image processing in brain

- Respond to changes at certain frequency and orientation
- Deep learning models usually learn similar filters
- Applied via the convolution operation

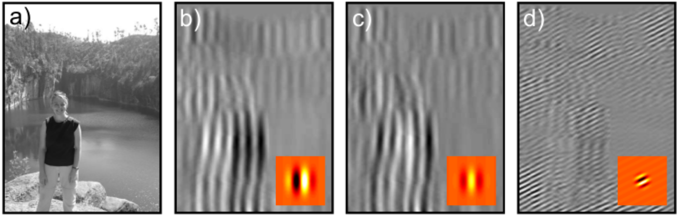


Image from [1]

Feature Extraction
Low-Level Features

Many other such feature extractors exist (e.g. SIFT, LBP)

- Suffer from same problems (extract low-level features)
- So we do not care about differences

All perform poorly on real-world tasks for this reason

- Reason why we need deep learning for computer vision
- Deep learning provides high-level features

But how can an algorithm learn?

Consider the following

- Each feature vector $\mathbf{x}$ is a point in D-dimensional space
- We want all cat images on one side of a learned hyperplane
- And all dog images on the other side

Assume a hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$

- $\mathbf{x}$ is on positive side if $o = \mathbf{w} \cdot \mathbf{x} + b \geq 0$
- Training entails adjusting $\mathbf{w}$ and b

Or more generally (works for $T > 2$)

- Learn T hyperplanes, one per class
- Want $o_t = \mathbf{w}_t \cdot \mathbf{x} + b_t \gg o_i \; \forall i \neq t$ if correct class is t

In other words we learn T binary classifiers

- Approach is called one vs all or one vs rest
- Most popular approach for multiclass classification

Goal of training: make hyperplanes always answer correctly

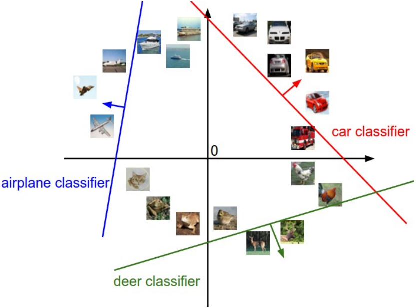


Image from cs231n.github.io

Can combine this to $\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$

- $\mathbf{W} \in \mathbb{R}^{T \times D}$ is called weight matrix
- $\mathbf{b} \in \mathbb{R}^{T}$ is called bias vector

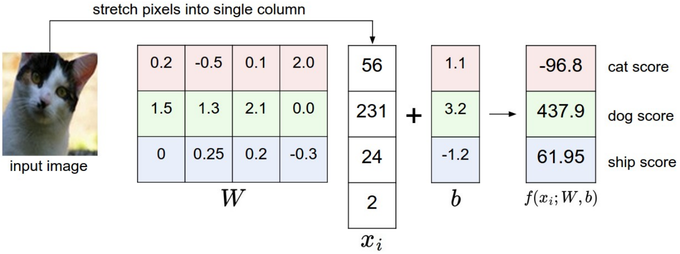


Image from cs231n.github.io

Model predicts T class scores $\mathbf{o} \in \mathbb{R}^T$

- Represent confidences (scaled distances from planes)

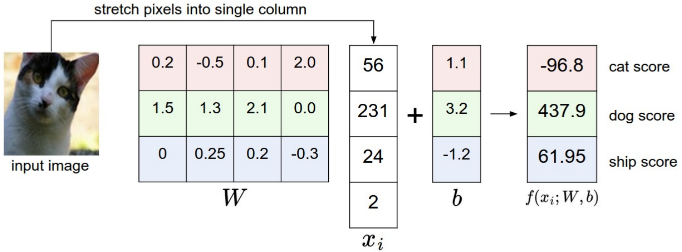


Image from cs231n.github.io

We have just derived linear models for classification

Heard that linear models are not useful in practice?

- They are the standard classifier in deep learning
- Sufficient because deep learning provides great features

A model describes family of functions from $\mathbf{x}$ to $\mathbf{o}$

- Particular function $f : \mathbf{x} \mapsto \mathbf{o}$ learned during training
- These functions are called trained models (or just models)

Model defines the hypothesis space

- Set of functions allowed as solution
- Extending family increases the model capacity (flexibility)

In parametric models f depends on parameters $\boldsymbol{\theta}$

- We write $\mathbf{o} = f(\mathbf{x}; \boldsymbol{\theta})$
- Training entails finding good parameters

Linear models are parametric models with $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$

- Number of parameters is $D \cdot T + T$
- Models in deep learning can have billions of parameters

T learned templates that are matched with input images

- Each $\mathbf{w}_t$ encodes a template
- Matching using inner product $\mathbf{w}_t \cdot \mathbf{x}$ (plus b_t)
- Result increases with similarity of image to template

Templates learned on CIFAR-10:

Image from cs231n.github.io

Most templates have clear interpretation

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

Image from cs231n.github.io

Linear Models

Limitations ($\mathbf{x}$ = image)

Classifier cannot properly model intraclass variation

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

What about our cats vs. dogs problem?

- Find out yourself during assignment 1

Image from cs231n.github.io

The Softmax Function
Motivation

Predicted class scores $\mathbf{o}$ are not optimal

- Unbound, can be negative
- Hard to interpret, scaling is not uniform

We want $\mathbf{o}$ to be a valid probability mass function

- $o_t \geq 0$ for all t and $\sum_t o_t = 1$

The Softmax Function
Definition

Most popular function for this purpose is softmax

$$\text{softmax}_t(\mathbf{o}) = \frac{\exp(o_t)}{\sum_t \exp(o_t)}$$

We obtain softmax$((1, 0, 4)) \approx (0.05, 0.02, 0.93)$

- Largest value is emphasized, small ones suppressed
- Softmax is not scale invariant

Loss Functions

But how can be learn $\mathbf{W}$ and $\mathbf{b}$?

For training any parametric model, we need

- A loss function
- An optimization algorithm

Loss Functions

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

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

Choice of L depends on task

- Most popular classification loss is cross-entropy

Recall from information theory that

- Given a mass function $\mathbf{u} = (u_1, \dots, u_T)$
- The entropy of $\mathbf{u}$ is $H(\mathbf{u}) = -\sum_{t=1}^{T} u_t \log_b u_t$
- In information theory $b = 2$, here it does not matter

$H(\mathbf{u})$ is average information gain when sampling from $\mathbf{u}$

- Biased coin with $\mathbf{u} = (1, 0)$ has entropy 0 ($\log_b 1 = 0$)
- Unbiased coin with $\mathbf{u} = (0.5, 0.5)$ has entropy $>$ 0

Loss Functions
Cross-Entropy

Given two probability mass functions $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{R}^T$

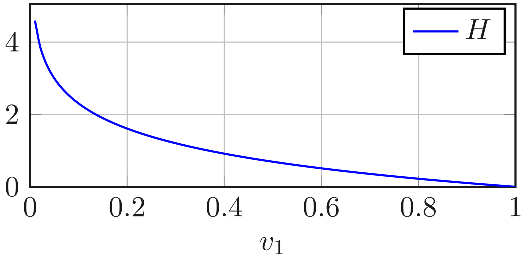
- $\mathbf{u} = (u_1, \ldots, u_T)$ and $\mathbf{v} = (v_1, \ldots, v_T)$

The cross-entropy between $\mathbf{u}$ and $\mathbf{v}$ is

$$H(\mathbf{u}, \mathbf{v}) = -\sum_{t=1}^{T} u_t \ln v_t$$

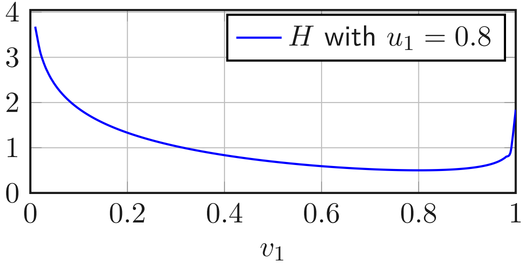
Example with $T = 2$ and $u_1 = 1$

- The more different $\mathbf{u}$ and $\mathbf{v}$ the higher H
- H measures the dissimilarity between $\mathbf{u}$ and $\mathbf{v}$



Note that H can reach 0 only if any $u_t = 1$

- In general $H(\mathbf{u}, \mathbf{v}) = H(\mathbf{u})$ if $\mathbf{u} = \mathbf{v}$



To utilize the cross-entropy for classifier training we

- Let $\mathbf{u}$ be our one-hot encoded class labels
- Let $\mathbf{v}$ be the predicted softmax class scores

H measures how dissimilar true and predicted probabilities are

- How well the classifier performs on a single sample

Note that $H(\mathbf{u}, \mathbf{v})$ depends only on u_t and v_t

- $u_t = 1$ so all other u_k are 0

On this basis we calculate the cross-entropy loss on $\mathcal{D}$ as

$$L(\boldsymbol{\theta}) = \frac{1}{S} \sum_{s=1}^{S} H(\mathbf{o}^s, \mathsf{softmax}(f(\mathbf{x}^s; \boldsymbol{\theta})))$$

Average cross-entropy over some dataset $\mathcal{D}$

- How good our classifier performs on $\mathcal{D}$ on average

Gradient Descent
Motivation

We now know how to compute $L(\boldsymbol{\theta})$ for classification

Need a way to minimize $L(\boldsymbol{\theta})$

- Maximizes the training set classification performance
- And hopefully also validation/test performance (more later)

$L(\boldsymbol{\theta})$ is not linear in $\boldsymbol{\theta}$

- Need a nonlinear optimization algorithm
- Gradient descent is popular choice in deep learning

Assume terrain corresponds to $L(\boldsymbol{\theta})$ with $\dim(\boldsymbol{\theta}) = 2$

Gradient Descent
Introduction

How do I get from location $\boldsymbol{\theta}$ to location of minimum $\hat{\boldsymbol{\theta}}$?

Without actually seeing $L(\boldsymbol{\theta})$?

Feel slope with feet, step in direction that feels steepest

- Again and again until ground feels flat

Gradient Descent
Definition

Iterative Optimization algorithm

In every iteration we

- Compute gradient $\boldsymbol{\theta}' = \nabla L(\boldsymbol{\theta})$
- Update parameters $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha\boldsymbol{\theta}'$

Hyperparameter $\alpha > 0$ is called learning rate

- Final step size is $\alpha \left\|\boldsymbol{\theta}'\right\|$

Let $f(x_1, \ldots, x_n)$ be a differentiable, real-valued function

The partial derivative f_{x_i} of f with respect to x_i

- Is also a real-valued function $f_{x_i}(x_1, \ldots, x_n)$

$f_{x_i}(\mathbf{x})$ encodes

- How fast f changes with argument x_i
- At some location $\mathbf{x}$

Gradient ∇f is vector of all partial derivatives of f

- $\nabla f = (f_{x_1}, \dots, f_{x_n})$
- Vector-valued function $\mathbb{R}^n \mapsto \mathbb{R}^n$

$\nabla f(\mathbf{x}) = (f_{x_1}(\mathbf{x}), \dots, f_{x_n}(\mathbf{x}))$ encodes

- How fast f changes with all arguments $x_1 \cdots x_n$
- At some location $\mathbf{x}$

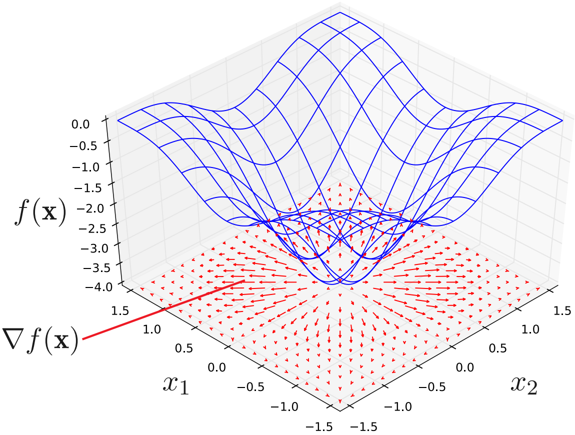


Image adapted from wikimedia.org

Gradient Descent
Gradients

$\nabla f(\mathbf{x})$ specifies how f changes locally at $\mathbf{x}$

- Points in direction of greatest increase
- Norm equals magnitude of increase

Exactly what we need to minimize L

- Compute direction of greatest increase $\nabla L(\boldsymbol{\theta})$
- Move in the opposite direction

Must stop if $\nabla L(\boldsymbol{\theta}) \approx \mathbf{0}$ (if norm is close to 0)

- No information where to go next
- L is flat at current location
- The case if we are at $\hat{\boldsymbol{\theta}}$ (but not only then)

Gradient Descent
Remarks

Simple and general algorithm

- Requires only that f is differentiable and real-valued
- Efficient (requires only first derivatives)

Several (possible) limitations

- Performs poorly for many f
- But works remarkably well with deep learning models

In a later lecture we will cover

- Various improvements to gradient descent training
- How to compute ∇L efficiently

Machine Learning for Image Classification

Demo

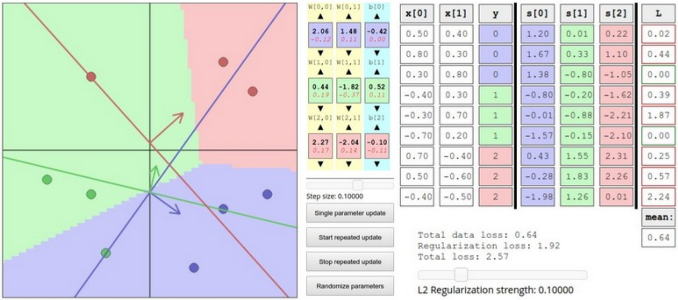


Image from cs231n.github.io

We factorized machine learning into basic blocks

- Output shape, model, loss function, optimizer

Flexible approach that (hopefully) makes things clearer

- Different problem? Change output shape and loss function
- Want more capacity? Change the model
- Optimization not going well? Change optimizer

Bibliography

[1] Prince. Computer Vision Models. 2012.