Machine Learning for Image Processing and Synthesis

Julie Digne

LIRIS - CNRS
Equipe Origami
Teaser

From http://www.vision-ary.net/2015/03/boost-the-world-cat-faces/
File "/usr/lib/python3.9/multiprocessing/pool.py", line 771, in get
raise self._value
AssertionError: daemonic processes are not allowed to have children
1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data
Classical Vision Algorithms

- **Line detection:**
Classical Vision Algorithms

- Line detection: RANSAC/Hough for line parameter estimation.
Classical Vision Algorithms

- Line detection: RANSAC/Hough for line parameter estimation.
- Cat recognition: devise a model for a cat.
Classical Vision Algorithms

• Line detection: RANSAC/Hough for line parameter estimation.
• Cat recognition: devise a model for a cat.
  Example: two close-by ellipsoids
Classical Vision Algorithms

- Line detection: RANSAC/Hough for line parameter estimation.
- Cat recognition: devise a model for a cat.
  Example: two close-by ellipsoids
- Airplane recognition: devise a model for an airplane.
Classical Vision Algorithms

- Line detection: RANSAC/Hough for line parameter estimation.
- Cat recognition: devise a model for a cat.
  Example: two close-by ellipsoids
- Airplane recognition: devise a model for an airplane.
  Example one rectangle and two triangles?
Classical Vision Algorithms

- Line detection: RANSAC/Hough for line parameter estimation.
- Cat recognition: devise a model for a cat.
  Example: two close-by ellipsoids
- Airplane recognition: devise a model for an airplane.
  Example: one rectangle and two triangles?
- How about when it’s more complicated?
Classical Vision Algorithms

- Line detection: RANSAC/Hough for line parameter estimation.
- Cat recognition: devise a model for a cat.
  Example: two close-by ellipsoids
- Airplane recognition: devise a model for an airplane.
  Example one rectangle and two triangles?
- How about when it’s more complicated?

Problem

MS-COCO: 91 categories of objects. [Biederman 87]: around 10000 to 30000 common objects to model: a model for each of them Not doable in practice.
Machine learning and vision

- Recognition/detection
Machine learning and vision

- **Recognition/detection**
  - Recognize objects in an image/video.
Machine learning and vision

- **Recognition/detection**
  - Recognize objects in an image/video.
  - Locate an object in an image/video.
Machine learning and vision

- **Recognition/detection**
  - Recognize objects in an image/video.
  - Locate an object in an image/video
  - Recognize a behavior/an emotion in a video
Machine learning and vision

- **Recognition/detection**
  - Recognize objects in an image/video.
  - Locate an object in an image/video.
  - Recognize a behavior/an emotion in a video.

- **Synthesis**
Machine learning and vision

- **Recognition/detection**
  - Recognize objects in an image/video.
  - Locate an object in an image/video
  - Recognize a behavior/an emotion in a video

- **Synthesis**
  - Generate an image that looks like a set of examples
Machine learning and vision

**Recognition/detection**
- Recognize objects in an image/video.
- Locate an object in an image/video
- Recognize a behavior/an emotion in a video

**Synthesis**
- Generate an image that looks like a set of examples
- Generate an image from a sketch given by a user.
Supervised and Unsupervised learning

- *Supervised Learning* a set of data \((x_i)_i\) and associated labels (ex: cat, car, house...) \((l_i)_i\), learn a function \(\hat{f}\) such that \(\hat{f}(x_i) = l_i\).

- *Unsupervised Learning* a set of data \((x_i)_i\) without any label and learns from similarities between data.
Supervised and Unsupervised learning

- **Supervised Learning** a set of data \((x_i)_i\) and associated labels (ex: cat, car, house...) \((l_i)_i\), learn a function \(\hat{f}\) such that \(\hat{f}(x_i) = l_i\).

- **Unsupervised Learning** a set of data \((x_i)_i\) **without any label** and learns from *similarities* between data.
Some examples from previous classes

- Meanshift
Some examples from previous classes

- Meanshift
- K-means
Some examples from previous classes

- Meanshift
- K-means
- Expectation-Maximization
Some examples from previous classes

- Meanshift
- K-means
- Expectation-Maximization

Grouping problems

Unsupervised learning: no label provided for learning the classes.
Outline

1 Introduction

2 Object recognition

3 General Formulation

4 Support Vector Machine

5 On Trees and Forests

6 Boosting

7 Beyond classification: Dictionary Learning

8 Very small reminder on Convolutional Neural Networks

9 Generative problems

10 Generative Adversarial Networks (GAN)

11 Generative problems beyond images: Geometric data
Is this object in the image?

- **Recognition/Classification:** Is there a bicycle in this image?
- **Detection:** Where is the bicycle in the image if any?

**For now:** supervised learning setting.
Why is object recognition/detection difficult?

(a) (b) (c) (d)
Why is object recognition/detection difficult?
Why is object recognition/detection difficult?
Why is object recognition/detection difficult?
Why is object recognition/detection difficult?
Outline

1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data

General Formulation
General setup of a supervised machine learning problem

- Data: split into:
  - Training data
  - Evaluation data
  - Test data

- Given data and labels \((x_i, l_i)_i\), find \(f\) minimizing an objective function:

\[
\sum_i (f(x_i) - l_i)^2
\]
General setup of a supervised machine learning problem

- Data: split into:
  - Training data
  - Evaluation data
  - Test data
- Given data and labels \((x_i, l_i)_i\), find \(f\) minimizing an objective function:
  \[
  \sum_i (f(x_i) - l_i)^2
  \]
- This is the \(\ell^2\) loss but several objective functions exist (also called loss)
Underfitting and Overfitting

- **Under-fitting**: (too simple to explain the variance)
- **Appropriate-fitting**:  
- **Over-fitting**: (forcefitting – too good to be true)
Underfitting and Overfitting

Under-fitting

Appropriate-fitting

Over-fitting

(too simple to explain the variance)

(forcefitting – too good to be true)
Precision and Recall

- **Precision**: how accurate is the classifier in detecting a positive example and not misclassifying.

\[
\text{Precision} = \frac{\# \text{True positives}}{\# \text{True positives} + \# \text{False positives}}
\]

- **Recall**: how accurate is the classifier in correctly detecting a positive example.

\[
\text{Recall} = \frac{\# \text{True positives}}{\# \text{positive examples}}
\]

Precision and recall curves are usually drawn with respect to the number of training iterations.

**Other indicators**

- Bias, variance, confusion matrix...
Outline

1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data
Support Vector Machine

Large margin binary classifier

Find an hyperplane separating the two classes maximizing the *margin*.
Support Vector Machine

- Works well when the classes are linearly separable. What if it’s not the case?
Support Vector Machine

- Works well when the classes are linearly separable. What if it’s not the case?

**Kernel trick**

Find a function $\Phi$ such that the two classes of $(\phi(x_i), l_i)_i$ are linearly separable.
Support Vector Machine

- Works well when the classes are linearly separable. What if it’s not the case?

**Kernel trick**

Find a function $\Phi$ such that the two classes of $(\phi(x_i), l_i); i$ are linearly separable.

- Problem: how do we design $\Phi$? **Manually**
Support Vector Machine

- Works well when the classes are linearly separable. What if it’s not the case?

**Kernel trick**

Find a function \( \Phi \) such that the two classes of \( (\phi(x_i), l_i)_i \) are linearly separable.

- Problem: how do we design \( \Phi \)? Manually or that’s where Deep Learning methods come in handy.
Optimization

- Equation of the separating hyperplane: \( w^T x + b = 0 \)
- If \( w^T x_i + b > 0 \) then \( l_i = 1 \), and if \( w^T x_i + b < 0 \) then \( l_i = -1 \).
- Decision function: \( f(x) = \text{sign}(w^T x + b) \).

Maximal margin

Maximize the distance between hyperplanes \( w^T x + b = \pm 1 \). Decision function:
\( l_i = 1 \) if \( w^T x + b \geq 1 \), \( l_i = -1 \) if \( w^T x + b \leq 1 \).

What is the size of the margin between the two hyperplanes?
Optimization problem

\[
\text{Minimize}_{w, b} \frac{1}{2} w^T w
\]
subject to \( \forall i, l_i(w^T x_i + b) \geq 1 \)
Optimization problem

\[
\text{Minimize}_{w, b} \frac{1}{2} w^T w
\]

subject to \( \forall i, l_i(w^T x_i + b) \geq 1 \)

Allow for some training errors: samples that violates the margin condition

Reformulation

\[
\text{Minimize}_{w, b} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i
\]

subject to \( \forall i, l_i(w^T x_i + b) \geq 1 - \xi_i \) and \( \forall i, \xi_i \geq 0 \)
Notice that constraints are equivalent to setting $\xi_i = \max(0, 1 - l_i(w^T x_i + b))$.
Notice that constraints are equivalent to setting $\xi_i = \max(0, 1 - l_i(w^T x_i + b))$
set $x_i = [x_i; 1]$ and $w = [w; b]$ to simplify.
Optimization (continued)

- Notice that constraints are equivalent to setting $\xi_i = \max(0, 1 - l_i(w^T x_i + b))$
- Set $x_i = [x_i; 1]$ and $w = [w; b]$ to simplify.

Reformulation

$$\text{Minimize}_w J(w) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \max(0, 1 - l_i(w^T x_i))$$
Optimization (continued)

- Notice that constraints are equivalent to setting $\xi_i = \max(0, 1 - l_i(w^T x_i + b))$
- set $x_i = [x_i; 1]$ and $w = [w; b]$ to simplify.

**Reformulation**

Minimize $w J(w) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \max(0, 1 - l_i(w^T x_i))$

- Unconstrained optimization
Optimization (continued)

- Notice that constraints are equivalent to setting \( \xi_i = \max(0, 1 - l_i(w^T x_i + b)) \)
- set \( x_i = [x_i; 1] \) and \( w = [w; b] \) to simplify.

### Reformulation

\[
\text{Minimize}_w J(w) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \max(0, 1 - l_i(w^T x_i))
\]

- Unconstrained optimization
- Gradient descent \( w_{t+1} = w_t - \nu_t \nabla_w J(w_t) \)
Optimization (continued)

**Stochastic gradient descent**

A gradient computed per sample:

\[
J(w, x_i, l_i) = \frac{1}{2} w^T w + C \max(0, 1 - l_i(w^T x_i))
\]

- initialization \( w_0 = 0 \)
- While not converged
  - For each training sample \((x_i, l_i)\)
  - Compute \( \nabla_w J(w_t, x_i, l_i) \)
  - \( w_{t+1} = w_t - \nu_t \nabla_w J(w_t, x_i, l_i) \)
- Return \( w \)
Optimization (continued)

Problem

\( J \) is not differentiable!

Strategy:

\[ \nabla J = w \text{ if } \max(0, 1 - l_i w^T x_i) = 0 \]

\[ \nabla J = w - Cl_i x_i \text{ otherwise} \]

Initialization

\( w_0 = 0 \)

While not converged

▶ For each training sample \((x_i, l_i)\)

▶ If \( l_i w^T x_i \leq 1 \), \( w^{t+1} = (1 - \nu_t) w^t + \nu_t Cl_i x_i \)

▶ Otherwise \( w^{t+1} = (1 - \nu_t) w^t \)

Return \( w \)

Stochastic Gradient Descent

Shue the training set before picking an example

What's wrong in the above derivation?
Optimization (continued)

Problem

\( J \) is not differentiable! Strategy:

1. \( \nabla J = w \) if \( \max(0, 1 - l_i w^T x_i) = 0 \)
2. \( \nabla J = w - C_l i x_i \) otherwise
Optimization (continued)

**Problem**

*J* is not differentiable! Strategy:

1. \( \nabla J = w \) if \( \max(0, 1 - l_i w^T x_i) = 0 \)
2. \( \nabla J = w - C l_i x_i \) otherwise

- Initialization \( w_0 = 0 \)
- While not converged
  - For each training sample \((x_i, l_i)\)
    - If \( l_i w_t^T x_i \leq 1 \), \( w_{t+1} = (1 - \nu_t) w_t + \nu_t C l_i x_i \)
    - Otherwise \( w_{t+1} = (1 - \nu_t) w_t \)
- Return \( w \)
Optimization (continued)

Problem

\( J \) is not differentiable! Strategy:

- \( \nabla J = w \) if \( \max(0, 1 - l_i w^T x_i) = 0 \)
- \( \nabla J = w - C l_i x_i \) otherwise

- Initialization \( w_0 = 0 \)
- While not converged
  - For each training sample \( (x_i, l_i) \)
    - If \( l_i w^T x_i \leq 1 \), \( w_{t+1} = (1 - \nu_t)w_t + \nu_t C l_i x_i \)
    - Otherwise \( w_{t+1} = (1 - \nu_t)w_t \)

- Return \( w \)

Stochastic Gradient Descent

Shuffle the training set before picking an example

What’s wrong in the above derivation?
Pedestrian detection (Dalal & Triggs 2005)

- Descriptor of each image: Histogram of oriented gradients
- Classified using a linear SVM (soft: allows for some margin violation during training).
Outline

1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data
Decision trees

**Principle**

Partition the feature space into a set of regions on which the decision will be uniform.

- How to build a decision tree:
  - Given a set of training data with associated labels \((x_i, l_i)_i\)
Decision trees

Principle
Partition the feature space into a set of regions on which the decision will be uniform.

- How to build a decision tree:
  - Given a set of training data with associated labels \((x_i, l_i)\);
  - At each step, find the subdivision of the space that best enforces the homogeneity of the labels inside each domain using a (usually weak) classifier.
Decision trees

**Principle**
Partition the feature space into a set of regions on which the decision will be uniform.

- How to build a decision tree:
  - Given a set of training data with associated labels \((x_i, l_i)\);
  - At each step, find the subdivision of the space that best enforces the homogeneity of the labels inside each domain using a (usually weak) classifier.

**Homogeneity measure**
Example: Shannon Entropy. \( L \) random variable of the label of the samples inside a domain \( \Omega \):

$$H(X) = - \sum_{l} P(X = l) \log P(X = l)$$

The entropy should be as low as possible inside each partition cell.
Random Forests [Breiman, Cutler 2001]

**Principle**

Aggregate the result of several decision trees.

**Advantage:** Less overfitting.

**Training Algorithm**

Create $B$ training sets by drawing $N$ samples from the initial training set. On each of the $B$ training sets train a decision tree.

**Decision Algorithm**

Apply the $B$ trees to a data to classify and record the $B$ decisions. Majority vote for the final decision.
Random Forests [Breiman, Cutler 2001]

**Principle**
Aggregate the result of several decision trees.

- Advantage: less overfitting.
Random Forests [Breiman, Cutler 2001]

**Principle**
Aggregate the result of several decision trees.

- Advantage: less overfitting.

**Training Algorithm**
- Create B training sets by drawing N samples from the initial training set.
Random Forests [Breiman, Cutler 2001]

### Principle

Aggregate the result of several decision trees.

- Advantage: less overfitting.

### Training Algorithm

- Create B training sets by drawing N samples from the initial training set.
- On each of the B training sets train a decision tree.
Random Forests [Breiman, Cutler 2001]

**Principle**
Aggregate the result of several decision trees.

- Advantage: less overfitting.

**Training Algorithm**
- Create $B$ training sets by drawing $N$ samples from the initial training set.
- On each of the $B$ training sets train a decision tree.

**Decision Algorithm**
- Apply the $B$ trees to a data to classify and record the $B$ decisions.
## Random Forests [Breiman, Cutler 2001]

### Principle

Aggregate the result of several decision trees.

- Advantage: less overfitting.

### Training Algorithm

- Create $B$ training sets by drawing $N$ samples from the initial training set.
- On each of the $B$ training sets train a decision tree.

### Decision Algorithm

- Apply the $B$ trees to a data to classify and record the $B$ decisions
- Majority vote for the final decision.
Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, Andrew Blake

CVPR 2011
Body part recognition

- No temporal information
  - frame-by-frame

- Local pose estimate of parts
  - each pixel & each body joint treated independently

- Very fast
  - simple depth image features
  - parallel decision forest classifier
The Kinect pose estimation pipeline

capture depth image & remove bg

infer body parts per pixel

cluster pixels to hypothesize body joint positions

fit model & track skeleton
Synthetic training data

Record mocap
500k frames
distilled to 100k poses

Retarget to several models

Render (depth, body parts) pairs

Train invariance to:
Synthetic vs. real data

synthetic
(train & test)

real
(test)
Fast depth image features

- Depth comparisons
  - very fast to compute

\[ f(I, x) = d_I(x) - d_I(x + \Delta) \]
\[ \Delta = v/d_I(x) \]

scales inversely with depth

Background pixels
\( d = \text{large constant} \)
Depth of trees

input depth

ground truth parts

inferred parts (soft)

depth 18
Depth of trees

Average per-class accuracy vs. Depth of trees for synthetic test data and real test data.

- **Synthetic test data**
  - Red triangle line
  - Green cross line

- **Real test data**
  - Red triangle line
  - Green cross line

- **900k training images**
- **15k training images**
Number of trees

<table>
<thead>
<tr>
<th>Number of trees</th>
<th>Average per-class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40%</td>
</tr>
<tr>
<td>2</td>
<td>45%</td>
</tr>
<tr>
<td>3</td>
<td>50%</td>
</tr>
<tr>
<td>4</td>
<td>55%</td>
</tr>
</tbody>
</table>

- **Ground truth**: 6 trees
- **Inferred body parts (most likely)**: 1 tree, 3 trees, 6 trees
input depth  
inferred body parts  

front view  
side view  
top view  
inferred joint positions  
no tracking or smoothing
input depth
inferred body parts

front view
side view
top view

inferred joint positions
no tracking or smoothing
From proposals to skeleton

- Use...
  - 3D joint hypotheses
  - kinematic constraints
  - temporal coherence

- ... to give
  - full skeleton
  - higher accuracy
  - invisible joints
  - multi-player
Outline

1 Introduction
2 Object recognition
3 General Formulation
4 Support Vector Machine
5 On Trees and Forests
6 Boosting
7 Beyond classification: Dictionary Learning
8 Very small reminder on Convolutional Neural Networks
9 Generative problems
10 Generative Adversarial Networks (GAN)
11 Generative problems beyond images: Geometric data
Boosting

**Principle [Rojas 2009]**

Combine $N$ *weak* classifiers $f_n$ to get a better classifier $g = \sum_{i=1}^{N} \alpha_i f_i$. Initially all training samples have the same weight. At each iteration:

- Find the classifier with the lowest training error.
- Raise the weights of the training examples misclassified by this classifier.

Final classifier: linear combination of the classifiers (weights proportional to the classifier accuracy).

Examples

AdaBoost, LogitBoost: difference lies in the way the weights of the samples are updated.
## Principle [Rojas 2009]

Combine $N$ weak classifiers $f_n$ to get a better classifier $g = \sum_{i=1}^{N} \alpha_i f_i$.

- Initially all training samples have the same weight.
Boosting

Principle [Rojas 2009]

Combine $N$ weak classifiers $f_n$ to get a better classifier $g = \sum_{i=1}^{N} \alpha_i f_i$.

- Initially all training samples have the same weight.
- At each iteration:
  - Find the classifier with the lowest training error.
Principle [Rojas 2009]

Combine $N$ weak classifiers $f_n$ to get a better classifier $g = \sum_{i=1}^{N} \alpha_i f_i$.

- Initially all training samples have the same weight.
- At each iteration:
  - Find the classifier with the lowest training error.
  - Raise the weights of the training examples misclassified by this classifier.
Boothing

Principle [Rojas 2009]

Combine $N$ weak classifiers $f_n$ to get a better classifier $g = \sum_{i=1}^{N} \alpha_if_i$.

- Initially all training samples have the same weight.
- At each iteration:
  - Find the classifier with the lowest training error.
  - Raise the weights of the training examples misclassified by this classifier.
- Final classifier: linear combination of the classifiers (weights proportional to the classifier accuracy).
Boosting

**Principle [Rojas 2009]**

Combine \( N \) *weak* classifiers \( f_n \) to get a better classifier \( g = \sum_{i=1}^{N} \alpha_i f_i \).

- Initially all training samples have the same weight.
- At each iteration:
  - Find the classifier with the lowest training error.
  - Raise the weights of the training examples misclassified by this classifier.
- Final classifier: linear combination of the classifiers (weights proportional to the classifier accuracy).

**Examples**

Adaboost, Logiboost: difference lies in the way the weights of the samples are updated.
Example

Weak Classifier 1
Example

Weights Increased
Example

Weak Classifier 2

From David Filliat (ENSTA)
Example

Weights Increased

From David Filliat (ENSTA)
Example

From David Filliat (ENSTA).

Boosting 38/167
Example

Final classifier is a combination of weak classifiers

From David Filliat (ENSTA)
Robust Object Detection [Viola, Jones 2001]

- Work on simple features: rectangle integration of images
- Optimize the classifier parameters
Adaboost for combining the classifiers

- Given example images \((x_1, y_1), \ldots, (x_n, y_n)\) where 
  \(y_i = 0, 1\) for negative and positive examples respectively.
- Initialize weights \(w_{1,i} = \frac{1}{nm} \cdot \frac{1}{p_i}\) for \(y_i = 0, 1\) respectively, where \(m\) and \(l\) are the number of negatives and positives respectively.
- For \(t = 1, \ldots, T\):
  1. Normalize the weights,
     \[ w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{p_i} w_{t,j}} \]
     so that \(w_t\) is a probability distribution.
  2. For each feature, \(j\), train a classifier \(h_j\) which
     is restricted to using a single feature. The
     error is evaluated with respect to \(w_t\), 
     \(e_j = \sum_i w_t |h_j(x_i) - y_i|\).
  3. Choose the classifier, \(h_{\hat{e}}\), with the lowest error \(e_{\hat{e}}\).
  4. Update the weights:
     \[ w_{t+1,i} = w_{t,i} \beta_t^{1-e_{\hat{e}}} \]
     where \(e_{\hat{e}} = 0\) if example \(x_t\) is classified
correctly, \(e_{\hat{e}} = 1\) otherwise, and 
     \(\beta_t = \frac{1}{1-e_{\hat{e}}}\).
- The final strong classifier is:
  \[
  h(x) = \begin{cases} 
    1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\
    0 & \text{otherwise}
  \end{cases}
  \]
  where \(\alpha_t = \log \frac{1}{\beta_t}\).
Outline

1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data
Sparse processing of signals

Signal Processing aims to decompose complex signals using elementary functions which are then easier to manipulate

\[ x(t) = \sum_{i=-\infty}^{\infty} \alpha_i \varphi_i(t) \]
Sparse processing of signals

\[ \text{signal} = \text{signal}_1 + \text{signal}_2 + \text{signal}_3 \]
Sparse processing of signals

- Between two representations of a signal pick the ones with the higher number of zero coefficients.
Patch-based approaches for images and surfaces

- Texture synthesis [Efros 99], Non local means [Buades et al. 2005].

**Compressive sensing theory [Candès et al. 2006]**

There exists spaces, in which the signals would be sparsely represented, that are especially well suited for processing the signals.

- Sparse regularization for image analysis, inpainting... [Elad et al. 2006], [Mairal 2009] The K-SVD algorithm
Norm definition

Let $E$ be a vector space over a subfield $K$, a norm on $E$ is an application with nonnegative values $\|\| : E \to \mathbb{R}$ such that for all $\alpha \in K$ and $u, v \in E$:

- $\|\alpha v\| = |\alpha|\|v\|$ (positive homogeneity)
- $\|u + v\| \leq \|u\| + \|v\|$ (subadditivity)
- $\|u\| = 0_K \iff u = 0_E$ (separation)
Sparse Coding: A brief reminder on norms

**Norm definition**

Let $E$ be a vector space over a subfield $K$, a norm on $E$ is an application with nonnegative values $\|\| : E \rightarrow R$ such that for all $\alpha \in K$ and $u, v \in E$:

- $\|\alpha v\| = |\alpha| \|v\|$ (positive homogeneity)
- $\|u + v\| \leq \|u\| + \|v\|$ (subadditivity)
- $\|u\| = 0_K \Leftrightarrow u = 0_E$ (separation)

- The $\ell^2$ norm is also called the euclidean norm. Let $x$ be a vector in $\mathbb{R}^n$ with coordinates $(x_1, \cdots, x_n)$ in the canonical basis, the $\ell^2$ norm writes:

$$\|x\|_2 = \sqrt{x \cdot x^T} = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$$
Norm Examples on vectors of $\mathbb{R}^n$

- $\ell^1$ Norm (Manhattan)

\[ \|x\|_1 = (\sum_{i=1}^{n} |x_i|) \]
Norm Examples on vectors of $\mathbb{R}^n$

- $\ell^1$ Norm (Manhattan)
  \[ \|x\|_1 = \left( \sum_{i=1}^{n} |x_i| \right) \]

- $\ell^3$
  \[ \|x\|_3 = \left( \sum_{i=1}^{n} |x_i|^3 \right)^{\frac{1}{3}} \]

Exercice: Prove that $\ell^\infty$ is indeed a norm?
Norm Examples on vectors of $\mathbb{R}^n$

- $\ell^1$ Norm (Manhattan)

\[ \|x\|_1 = \left( \sum_{i=1}^{n} |x_i| \right) \]

- $\ell^3$

\[ \|x\|_3 = \left( \sum_{i=1}^{n} |x_i|^3 \right)^{\frac{1}{3}} \]

- $\ell^{2.1}$

\[ \|x\|_{2.1} = \left( \sum_{i=1}^{n} x_i^{2.1} \right)^{\frac{1}{2.1}} \]
Norm Examples on vectors of $\mathbb{R}^n$

- $\ell^1$ Norm (Manhattan)
  \[ \|x\|_1 = \left(\sum_{i=1}^{n} |x_i|\right) \]

- $\ell^3$
  \[ \|x\|_3 = \left(\sum_{i=1}^{n} |x_i|^3\right)^{\frac{1}{3}} \]

- $\ell^{2.1}$:
  \[ \|x\|_{2.1} = \left(\sum_{i=1}^{n} x_i^{2.1}\right)^{\frac{1}{2.1}} \]

- $\ell^p$ pour $p \geq 1$
  \[ \|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}} \]
Norm Examples on vectors of $\mathbb{R}^n$

- $\ell^1$ Norm (Manhattan)

$$
\|x\|_1 = \left( \sum_{i=1}^{n} |x_i| \right)
$$

- $\ell^3$

$$
\|x\|_3 = \left( \sum_{i=1}^{n} |x_i|^3 \right)^{\frac{1}{3}}
$$

- $\ell^{2.1}$

$$
\|x\|_{2.1} = \left( \sum_{i=1}^{n} x_i^{2.1} \right)^{\frac{1}{2.1}}
$$

- $\ell^p$ pour $p \geq 1$

$$
\|x\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}
$$

- $\ell^\infty$

$$
\|x\|_\infty = \max_{i=1\ldots n} |x_i|
$$

Exercice: Prove that $\ell^\infty$ is indeed a norm?
The ball of radius 1 for norms $\ell^p$ with $p \geq 2$
The ball of radius 1 for norms $\ell^p$ with $p \geq 2$
The ball of radius 1 for norms $\ell^p$ with $p \geq 2$
The ball of radius 1 for norms $\ell^p$ with $p \geq 2$
The ball of radius 1 for norms $\ell^p$ with $p \leq 2$
The ball of radius 1 for norms $\ell^p$ with $p \leq 2$
The ball of radius 1 for norms $\ell^p$ with $p \leq 2$
The ball of radius 1 for norms $\ell^p$ with $p \leq 2$
The ball of radius 1 with norms and quasi-norms $\ell^p$
Norm and sparsity

**Sparsity definition**

A vector \( x \in \mathbb{R}^N \) is said to be \( s \)-sparse if at most \( s \) of its entries are non-zero, i.e.

\[
\text{card support}(x) \leq s
\]

where \( \text{support}(x) = \{i | x_i \neq 0\} \).

We note \( \|x\|_0 = \text{card support}(x) \) and call it \( \ell^0 \).
Norm and sparsity

**Sparsity definition**

A vector \( x \in \mathbb{R}^N \) is said to be \( s \)-sparse if at most \( s \) of its entries are non-zero, i.e.

\[
\text{card support}(x) \leq s
\]

where \( \text{support}(x) = \{i | x_i \neq 0\} \).

We note \( \|x\|_0 = \text{card support}(x) \) and call it \( \ell^0 \).

- Is \( \ell^0 \) a norm?
Sparse Coding with the $\ell^0$ norm

Problem statement

Let $A \in \mathbb{R}^{m \times n}$ and $\alpha$ a $s$-sparse vector in $\mathbb{R}^n$. Let $x \in \mathbb{R}^m$ such that $x = A\alpha$. Assume only $x$ and $A$ are known and we want to recover $\alpha$. If $m < n$, the system is underdetermined.
Sparse Coding with the $\ell^0$ norm

Problem statement
Let $A \in \mathbb{R}^{m \times n}$ and $\alpha$ a $s$-sparse vector in $\mathbb{R}^n$ Let $x \in \mathbb{R}^m$ such that $x = A\alpha$. Assume only $x$ and $A$ are known and we want to recover $\alpha$. If $m < n$, the system is underdetermined.

Sparsity hypothesis
Identifying the solution $\alpha$ under the $s$-sparsity hypothesis is easier.
Sparse Coding with the $\ell^0$ norm

**Optimization problem**

Given a measurement matrix $A \in \mathbb{R}^{m \times n}$ and $x$ a vector in $\mathbb{R}^n$, under the $s$-sparse assumption, the vector $\alpha$ can be reconstructed as the solution of:

$$\begin{align*}
\text{Minimize} & \quad ||\alpha||_0 \\
\text{s.t.} & \quad x = A\alpha
\end{align*} \quad (P_0)$$
Sparse Coding with the $\ell^0$ norm

**Optimization problem**

Given a measurement matrix $A \in \mathbb{R}^{m \times n}$ and $x$ a vector in $\mathbb{R}^n$, under the $s$-sparse assumption, the vector $\alpha$ can be reconstructed as the solution of:

$$\begin{align*}
\text{Minimize} & \quad ||\alpha||_0 \\
\text{s.t.} & \quad x = A\alpha
\end{align*} \quad (P_0)$$

- Nonconvex optimization problem
Problem ($P_0$) is a NP-hard problem

- Reformulate the problem as

\[
\begin{align*}
\text{Minimize} & \| \alpha \|_0 \\
\text{s.t.} & \| x - A \alpha \|_2 \leq \eta
\end{align*}
\]

($P_0, \eta$)

---

**Theorem**

Problem ($P_0, \eta$) is a NP-hard problem

- NP-hardness: all problems for which a solving algorithm could be turned in polynomial time into a solving algorithm for any NP-problem.
- Proof: demonstrate that using Problem ($P_0, \eta$) one can solve for the exact cover 3-set problem.
- Reminder: Given a collection $S$ of 3-subsets of a set $X$, an exact cover of $X$ is a subcollection $S_{\text{sub}}$ of $S$ such that the intersection of two distinct elements of $S_{\text{sub}}$ is empty and the union of all elements of $S_{\text{sub}}$ cover $X$. 
Sparse decomposition algorithm

Sparse Decomposition

Given a dictionary \( D \in \mathbb{R}^{m,n} \) whose columns have norm 1 and a signal \( x \in \mathbb{R}^n \) find a vector \( \alpha \) whose sparsity is \( s \) minimizing \( \|x - D\alpha\|_2^2 \)

- Efficient greedy algorithms have been proposed to find an approximate solution.
Matching Pursuit

Matching Pursuit Algorithm [Mallat & Zhang 1993]

- Set $k = 0$, $\alpha = 0_{\mathbb{R}^n}$
- While $k < s$ and $\|x - D\alpha\| > 0$ do:
  - Select index $j$ maximizing $|D_j^T \cdot (x - D\alpha)|$
  - Update coefficients $\alpha(j) = \alpha(j) + D_j^T \cdot (x - D\alpha)$
Matching Pursuit

Matching Pursuit Algorithm [Mallat & Zhang 1993]

- Set \( k = 0, \alpha = 0_{\mathbb{R}^n} \)
- While \( k < s \) and \( \| x - D\alpha \| > 0 \) do:
  - Select index \( j \) maximizing \( |D_j^T \cdot (x - D\alpha)| \)
  - Update coefficients \( \alpha(j) = \alpha(j) + D_j^T \cdot (x - D\alpha) \)

- At each step the algorithm finds the atom that best represents the residual \( r = x - D\alpha \)
### Matching Pursuit Algorithm [Mallat & Zhang 1993]

- Set $k = 0$, $\alpha = 0_{\mathbb{R}^n}$
- While $k < s$ and $\|x - D\alpha\| > 0$ do:
  - Select index $j$ maximizing $|D_j^T \cdot (x - D\alpha)|$
  - Update coefficients $\alpha(j) = \alpha(j) + D_j^T \cdot (x - D\alpha)$

- At each step the algorithm finds the atom that best represents the residual $r = x - D\alpha$
- The residual monotonically decreases until the residual is orthogonal to all $D_j$
Matching Pursuit

Matching Pursuit Algorithm [Mallat & Zhang 1993]

- Set $k = 0, \alpha = 0_{\mathbb{R}^n}$
- While $k < s$ and $\|x - D\alpha\| > 0$ do:
  - Select index $j$ maximizing $|D_j^T \cdot (x - D\alpha)|$
  - Update coefficients $\alpha(j) = \alpha(j) + D_j^T \cdot (x - D\alpha)$

- At each step the algorithm finds the atom that best represents the residual $r = x - D\alpha$
- The residual monotonically decreases until the residual is orthogonal to all $D_j$

How does the sparsity behave?
Matching Pursuit

Matching Pursuit Algorithm [Mallat & Zhang 1993]

- Set $k = 0$, $\alpha = 0_{\mathbb{R}^n}$
- While $k < s$ and $\|x - D\alpha\| > 0$ do:
  - Select index $j$ maximizing $|D_j^T \cdot (x - D\alpha)|$
  - Update coefficients $\alpha(j) = \alpha(j) + D_j^T \cdot (x - D\alpha)$

- At each step the algorithm finds the atom that best represents the residual $r = x - D\alpha$
- The residual monotonically decreases until the residual is orthogonal to all $D_j$

How does the sparsity behave? nondecreasing
Orthogonal Matching Pursuit

- Goal: The sparsity should increase at each step.
Orthogonal Matching Pursuit

- Goal: The sparsity should increase at each step.
  - How?
Orthogonal Matching Pursuit

- **Goal:** The sparsity should increase at each step.

- **How?** Render the residual orthogonal to all selected atoms.
Orthogonal Matching Pursuit

- Goal: The sparsity should increase at each step.

- How? Render the residual orthogonal to all selected atoms.
Orthogonal Matching Pursuit

- **Goal:** The sparsity should increase at each step.
- **How?** Render the residual orthogonal to all selected atoms.

### Orthogonal Matching Pursuit (OMP)

<table>
<thead>
<tr>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set $k = 0$, $\alpha = 0_{\mathbb{R}^n}$, $\Gamma = \emptyset$</td>
</tr>
<tr>
<td>While $k &lt; s$ and $|x - D\alpha| &gt; 0$ do:</td>
</tr>
<tr>
<td>Select index $j$ maximizing $</td>
</tr>
<tr>
<td>Update the active set $\Gamma = \Gamma \cup {j}$</td>
</tr>
<tr>
<td>Recompute $\alpha_\Gamma$ minimizing $x - D_\Gamma \alpha_\Gamma$</td>
</tr>
<tr>
<td>Set $\alpha_\Gamma = 0$</td>
</tr>
</tbody>
</table>

**Remark:** $D_\Gamma$, $\alpha_\Gamma$: matrix (resp. vector) composed of the columns (resp. elements) of $D$ (resp. $\alpha$) whose indices are in $\Gamma$. 

---

Beyond classification: Dictionary Learning
What can we prove about OMP?

- The index selection is guided by finding the one that makes the error decrease most.
What can we prove about OMP?

- The index selection is guided by finding the one that makes the error decrease most.
- What about the case where a vector is a linear combination of 3 vectors.
What can we prove about OMP?

- The index selection is guided by finding the one that makes the error decrease most.
- What about the case where a vector is a linear combination of 3 vectors.
- Tropp and Gilbert (2007): OMP is able to reliably recover a sparse vector from random measurements.
What can we prove about OMP?

- The index selection is guided by finding the one that makes the error decrease most.
- What about the case where a vector is a linear combination of 3 vectors.
- Tropp and Gilbert (2007): OMP is able to reliably recover a sparse vector from random measurements.
- OMP is slower than MP
How can we make OMP faster?

Which step is computationally intensive?

- Computing the best index means computing $D_f^T(x - D_\Gamma \alpha_\Gamma)$ and taking the index of the smallest coefficient
How can we make OMP faster?

Which step is computationally intensive?

- Computing the best index means computing $D_f^T(x - D_{\Gamma}\alpha_{\Gamma})$ and taking the index of the smallest coefficient
- Then we compute $\alpha_{\Gamma'}$ as $\alpha$ minimizing $\|x - D_{\Gamma'}\alpha\|^2$
How can we make OMP faster?

Which step is computationally intensive?

- Computing the best index means computing $D_f^T(x - D_\Gamma \alpha_\Gamma)$ and taking the index of the smallest coefficient
- Then we compute $\alpha_{\Gamma'}$ as $\alpha$ minimizing $\|x - D_{\Gamma'} \alpha\|^2$
- Closed form solution:

$$\alpha_{\Gamma'} = (D_{\Gamma'}^T D_{\Gamma'})^{-1} D_{\Gamma'} x$$
How can we make OMP faster?

Which step is computationally intensive?

- Computing the best index means computing $D_f^T(x - D_\Gamma \alpha_\Gamma)$ and taking the index of the smallest coefficient.
- Then we compute $\alpha_{\Gamma'}$ as $\alpha$ minimizing $\|x - D_{\Gamma'} \alpha\|^2$.
- Closed form solution: 
  \[ \alpha_{\Gamma'} = (D_{\Gamma'}^T D_{\Gamma'})^{-1} D_{\Gamma'} x \]

Making OMP faster

Invert quickly $D_{\Gamma'}^T D_{\Gamma'}$, knowing the inverse of $D_{\Gamma}^T D_{\Gamma}$.
Update of the inverse of $D^T D$ when appending a column $d$

- $u_1 \leftarrow D^T d$
- $u_2 \leftarrow (D^T D)^{-1} u_1$
- $u_3 \leftarrow d u_2$
- $A \leftarrow (X^T X)^{-1} + d u_2^T u_2$
- $s \leftarrow \frac{1}{d^T d - u_1^T u_2}$

Updated inverse: $(A \quad -u_3 \quad -u_3^T s)$
An application of OMP: synthesizing terrains based on examples [Guérin et al. 2016]

A terrain is seen as a set of blended patches
An application of OMP: synthesizing terrains based on examples [Guérin et al. 2016]

- Build a dictionary by decomposing a real-world elevation map into patches
- Decompose patches to synthesize on it
Terrain "Amplification"

Remark: Works because terrains are height fields, much more complicated with images or textures!
Terrain “Amplification”

Remark
Works because terrains are heightfields, much more complicated with images or textures!
Problem
In the context of Image Processing and Synthesis, we only have access to a set of signals for which we want to build a dictionary.
Dictionary learning

Problem

In the context of Image Processing and Synthesis, we only have access to a set of signals for which we want to build a dictionary.

Dictionary Learning Problem

Given a set of signals $x_i$ for $i = 1 \cdots N$ in $\mathbb{R}^n$ we want to build a matrix $D \in \mathbb{R}^{n \times m}$ and coefficients $\alpha_i \in \mathbb{R}^m$ for $i = 1 \cdots n$ solving:

\[
\text{Minimize} \quad \sum_{i=1}^{N} \| x_i - D \cdot \alpha_i \|_2^2 \\
\text{subject to} \quad \| D \|_2 \leq 1 \\
\forall i = 1 \cdots N, \alpha_i \in \mathbb{R}^m, \\
\forall i = 1 \cdots N, \| \alpha_i \|_0 = s
\]

($P_{D,\alpha,0}$)
Dictionary learning problems

- Still a nonconvex problem
- Common approach: alternate minimization
  - Fix the dictionary $D$ and compute the sparse decomposition $\alpha$
  - Fix the sparse decomposition $\alpha$ and compute $D$
Method of Optimal Directions (MOD)

First introduced by Engan et al. [1999]
Method of Optimal Directions (MOD)

- First introduced by Engan et al. [1999]
- Step 1: Compute the sparse codes?
Method of Optimal Directions (MOD)

- First introduced by Engan et al. [1999]
- Step 1: Compute the sparse codes? MP, OMP, Iterative Hard Thresholding
Method of Optimal Directions (MOD)

- First introduced by Engan et al. [1999]
- Step 1: Compute the sparse codes? MP, OMP, Iterative Hard Thresholding
- Step 2: Update the dictionary
Assume all coefficients $\alpha_i$ are fixed, Problem $(P_{D,\alpha,0})$ becomes

$$
\text{Minimize } \sum_{D \in \mathbb{R}^{n \times m}, \|D_i\|_2 \leq 1}^{N} \|x_i - D \cdot \alpha_i\|_2^2
$$
Assume all coefficients $\alpha_i$ are fixed, Problem $(P_{D,\alpha,0})$ becomes

$$\text{Minimize} \sum_{i=1}^{N} \|x_i - D \cdot \alpha_i\|_2^2$$

$$\text{subject to } \|D_i\|_2 \leq 1$$

This problem is a convex problem on a convex set.
MOD: Dictionary Update

- Assume all coefficients $\alpha_i$ are fixed, Problem $(P_{D,\alpha,0})$ becomes

$$\text{Minimize} \sum_{D \in \mathbb{R}^{n \times m}, \|D_i\|_2 \leq 1}^N \|x_i - D \cdot \alpha_i\|_2^2$$

- This problem is a convex problem on a convex set
- Discarding the unit norm constraint yields a least squares objective
Assume all coefficients $\alpha_i$ are fixed, Problem $(P_{D,\alpha,0})$ becomes

$$\text{Minimize} \sum_{i=1}^{N} \left\| x_i - D \cdot \alpha_i \right\|^2_2$$

This problem is a convex problem on a convex set

Discarding the unit norm constraint yields a least squares objective

Idea: Solve the least squares problem and project the solution onto the convex set of admissible solutions
Assume all coefficients $\alpha_i$ are fixed, Problem $(P_{D,\alpha,0})$ becomes

$$\text{Minimize} \sum_{i=1}^{N} \|x_i - D \cdot \alpha_i\|_2^2$$

This problem is a convex problem on a convex set

Discarding the unit norm constraint yields a least squares objective

Idea: Solve the least squares problem and project the solution onto the convex set of admissible solutions

Bertsekas 1999

In general, solving the general problem and projecting the solution on the convex constraints set yields a poor solution
Assume all coefficients $\alpha_i$ are fixed, Problem $(P_{D,\alpha,0})$ becomes

$$\text{Minimize } \sum_{i=1}^{N} \| x_i - D \cdot \alpha_i \|^2_2$$

This problem is a convex problem on a convex set.

Discarding the unit norm constraint yields a least squares objective.

Idea: Solve the least squares problem and project the solution onto the convex set of admissible solutions.

Bertsekas 1999

Since the $\ell^0$ norm remains constant when a vector undergoes a nonzero rescaling, the projection is valid.
Dictionary Update

Least Squares Problem

Solve for $D$ in:

$$\text{Minimize} \sum_{i=1}^{N} \|x_i - D\alpha_i\|_2^2$$
Dictionary Update

Least Squares Problem

Solve for $D$ in:

$$\text{Minimize} \sum_{i=1}^{N} \| x_i - D\alpha_i \|^2_2$$

- Compute the gradient and set to 0:
  $$\sum_{i=1}^{N} (x_i - D\alpha_i)\alpha_i^T = 0$$
Dictionary Update

Least Squares Problem

Solve for $D$ in:

$$\text{Minimize } \sum_{i=1}^{N} \|x_i - D\alpha_i\|_2^2$$

- Compute the gradient and set to 0: $\sum_{i=1}^{N} (x_i - D\alpha_i)\alpha_i^T = 0$
- $D = (\sum_{i=1}^{N} x_i\alpha_i^T)(\sum_{i=1}^{N} (\alpha_i\alpha_i^T)^{-1}$
Dictionary Update

Least Squares Problem

Solve for $D$ in:

$$\text{Minimize } \sum_{i=1}^{N} \| x_i - D \alpha_i \|^2_2$$

- Compute the gradient+set to 0: $\sum_{i=1}^{N} (x_i - D \alpha_i) \alpha_i^T = 0$
- $D = (\sum_{i=1}^{N} x_i \alpha_i^T)(\sum_{i=1}^{N} (\alpha_i \alpha_i^T)^{-1}$
- Setting $A = (\alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_N), \; X = (x_1 \mid x_2 \mid \cdots \mid x_N)$
  one has: $D = X A^T (A A^T)^{-1}$

Projection on the constraint set: no rmalizing each column of $D$ if its norm is above 1.
Dictionary Update

Least Squares Problem

Solve for $D$ in:

$$\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{N} \|x_i - D\alpha_i\|_2^2 \\
\text{subject to} & \quad D \in \mathbb{R}^{m \times n}
\end{align*}$$

- Compute the gradient and set to 0: $$\sum_{i=1}^{N} (x_i - D\alpha_i)\alpha_i^T = 0$$
- $$D = (\sum_{i=1}^{N} x_i\alpha_i^T)(\sum_{i=1}^{N} (\alpha_i\alpha_i^T)^{-1})$$
- Setting $A = (\alpha_1 | \alpha_2 | \cdots | \alpha_N)$, $X = (x_1 | x_2 | \cdots | x_N)$ one has: $D = XA^T(AA^T)^{-1}$
- Projection on the constraint set: normalizing each column of $D$ if its norm is above 1.
K-SVD algorithm

- Still an alternating direction minimization method
K-SVD algorithm

- Still an alternating direction minimization method
- Goal: Incorporate the sparsity constraint also in the dictionary update step
Goal

- A set of training signals \( \{x_i\}_{i=1}^N \in \mathbb{R}^n \)
- Design a dictionary \( D \in \mathbb{R}^{n \times K} \) such that there exists \( \alpha \in \mathbb{R}^k \) such that either \( x = D\alpha \)
  or \( \|x - D\alpha\|_p \leq \varepsilon \)
- if \( n < K \) and \( D \) is full-ranked the solution must be constrained
  - \( \min_\alpha \|\alpha\|_0 \) s.t. \( x = D\alpha \)
  - \( \min_\alpha \|\alpha\|_0 \) s.t. \( \|x - D\alpha\|_2 \leq \varepsilon \)
- Design \( D \) in order to best fit the sparsity model imposed
An extension of K-means

- K-means search for the best possible representative enforcing that each representation uses a single atom with coefficient 1.
- K-SVD solves \( \min_{D,A} \|X - DA\|^2_F \) s.t. \( \forall i \|\alpha_i\|_0 \leq T_0 \)
- An iterative approach that alternates between two steps
  - **Sparse coding** of the examples based on the current dictionary
  - **Update of the dictionary** so as to better fit the data
- \( X \in \mathbb{R}^{n \times N} \): training samples, \( A \in \mathbb{R}^{K \times N} \) matrix of coefficients
Sparse Coding stage

- D is fixed, compute the best representation $\alpha_i$ of sample $x_i$
- Find $\alpha_i$ minimizing $\|x_i - D\alpha_i\|_2^2$ s.t. $\|\alpha_i\|_0 \leq T_0$
- Can be done using a pursuit algorithm (e.g. Orthogonal Matching Pursuit)
Dictionary Update stage

- The update will be done atom by atom.
- \[ \|X - DA\|_F^2 = \|X - \sum_{j=1}^{N} d_j \alpha_j^T\|_F^2 = \|X - \sum_{j=1,j\neq k}^{N} d_j \alpha_j^T - d_k \alpha_k^T\|_F^2 \]
- \( E_k = X - \sum_{j=1,j\neq k}^{N} d_j \alpha_j^T \) error obtained by omitting atom \( d_k \) in the decomposition
- Finally solve for:
  \[ \|E_k - d_k \alpha_k^T\|_F \text{ w.r.t. } d_k, \alpha_k^T \]
- Solve using SVD? if so sparsity not enforced.
Trick to enforce the sparsity

- $\omega_k = \{i | 1 \leq i \leq K, \alpha^k_T(i) \neq 0\}$
- Restrict $E_k$ and $\alpha^k_T$ to $E^R_k$ and $\alpha^k_R$ by selecting only the columns of indices included in the support of $\alpha^k_T$
- Use SVD to decompose $E^R_k = U\Delta V^T$
- Set $d_k$ to be the first column of $U$
- Set $\alpha^k_R$ to be the first column of $V$ multiplied by $\Delta(1,1)$
- The columns of $D$ remain normalized and the support of the representations can not increase
Application to the denoising of images

- noisy input image $x$
- Build $\hat{D}$ and $(\hat{\alpha}_i)_i$ the dictionary and representations of all patches of image $x$
- $(P_i(y))_i$ the set of all image $y$ patches.
- $\hat{D}\hat{\alpha}_i$ is the representation of patch $P_i(x)$
- Find $y$ minimizing $\lambda \|x - y\|_2^2 + \sum_i \|\hat{D}\hat{\alpha}_i - P_i(y)\|_2^2$

\[ \text{fidelity term} + \text{proximity of the reconstructed patches to the denoised patches} \]
Can be tested on IPOL http://www.ipol.im/pub/algo/llm_ksvd
Learned dictionary

Dictionary learned from face patches
Train time 9.0s on 94500 patches
Learned Color dictionary
Denoising via dictionary learning

noisy image
Denoising via dictionary learning

Denoising each channel separately
Denoising via dictionary learning

Denoising all channels simultaneously
Denoising via dictionary learning

Denoising each channel separately (left) vs globally (right)
Comparison to NL-means
Comparison to NL-means

Dictionary learning
Comparison to NL-means
Application: Fluid Simulation [Bai et al. 2021]

- Accurate high-resolution Fluid Simulation are expansive.
- Simulate low-res only and synthesize the high res by dictionary learning
Application: Fluid Simulation [Bai et al. 2021]
Application: Fluid Simulation [Bai et al. 2021]

Comparison with TempoGAN [Xie 2018]
(Input/TempoGAN result/Dictionary Learning Result)
Application: Point Cloud Compression


Two samplings of the same shape
Pipeline

Original Seeds and patches Parameterization
Patch descriptions Coefficients Dictionary
1 2 n-1 n

Beyond classification: Dictionary Learning
Working assumptions

- **Topological condition:** Surface covered by a set of topological disks centered around seeds.

- **Sampling condition:** $R$-neighborhood of a seed containing enough points.

- **Noise level:** Noise magnitude strictly below radius $R$.

- **Seeds selection:** anchors to define local patches
Self-similarity compression

- Seeds selection
- Local patches represented in a comparable way

- Patches decomposed upon a dictionary found by the K-SVD algorithm
- Final data: a set of seeds with local frames, a small dictionary and the (sparse) coefficients for each patch.
Further compression

- **Seeds**: kd-tree compression [Gandoin and Devillers, 2002].
- **Local parameterization** (3 Euler angles): predictive coding
- **Dictionary**: lossless compression.
- **Coefficients**: scalar quantization (increases sparsity) followed by entropy coding.
Controlling the error

Two types of errors are introduced:

- **Resampling error**
  
  ⇒ *Increasing the accuracy of the resampling pattern*

- **Compression error**
  
  ⇒ *Increasing the number of atoms in the dictionary*
Decompression

1. Decompress
   - seed positions
   - euler angles
   - dictionary $D$
   - coefficients $A$

2. Reconstruct the patches:
   \[ P_{\text{rec}} = D \ast A \]

3. Consolidate the reconstructed point cloud in overlapping areas.
Anubis (9, 9M pts) compressed to 0.96bpp; error = 0.01mm (0.003%)
St Matthew (93.5M pts) compressed to 0.83bpp; error = 0.05cm (0.002%)
Comparison with kd-tree coding. 4.83bpp against 0.6bpp in our method.
Breaking the working assumptions

Church (69, 9M pts) compressed to 0.76bpp; error = 1.48cm (0.005%)
Beyond classification: Dictionary Learning

rate/distortion performance compared to previous works
Some results

(a) Original

(b) Noisy ($RMSE = 0.124$)

(c) APSS ($RMSE = 0.085$)

(d) RIMLS ($RMSE = 0.081$)

(e) Bilateral ($RMSE = 0.071$)

(f) WLOP ($RMSE = 0.051$)

(g) LPF ($RMSE = 0.017$)
Outline

1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data
Each connection has a weight $w$
Each neuron has a bias $b$ and an activation function $s$ (e.g. sigmoid).
Output of a neuron $s(wx + b)$
Each connection has a weight $w$
Each neuron has a bias $b$ and an activation function $s$ (e.g. sigmoid).
Output of a neuron $s(wx + b)$

For images
Each pixel is an input to the net.
Training a neural network

Output

In classification cases, the neural network outputs a class the input image supposedly belongs to.
Training a neural network

Output
In classification cases, the neural network outputs a class the input image supposedly belongs to.

Cost function
For training samples, we evaluate how well the neural net performed via a cost function $C$: Mean Square error, Cross-Entropy...
Training a neural network

To optimize the cost $C$

Gradient descent with respect to weight $w_i$ and bias $b_i$ for each neuron $i$. 
To optimize the cost $C$

Gradient descent with respect to weight $w_i$ and bias $b_i$ for each neuron $i$.

**Back-Propagation**

The gradient can be propagated back from the output to the input (chain rule).
Back-propagation example

**Toy model**

Compute the gradient of the cost with respect to each parameter.

\[ a^1 = s_1(w_1 x + b_1) \]
\[ a^2 = s_2(w_2 x + b_2) \]
Back-propagation example

**Toy model**

Compute the gradient of the cost with respect to each parameter.

\[
\begin{align*}
a^1 &= s_1(w_1 x + b_1) \\
a^2 &= s_2(w_2 x + b_2)
\end{align*}
\]

- In practice start with random weights and bias.
Convolutional Neural Networks

**Shared parameters**

Dropping fully connected layers, CNN use convolutions by kernels with weights independent of the image location. These weights are optimized during training.
Convolutional Neural Networks

**Shared parameters**

Dropping fully connected layers, CNN use convolutions by kernels with weights independent of the image location. These weights are optimized during training.
Convolution layer parameters

- **Kernel size**: controls the locality of the kernel
- **Padding**: increases the size of the input
- **Dilatation**: aggregates values from every $n$ pixels where $n$ is the dilatation. (eq to set some weights in the kernel to 0).
- **Stride**: performs the convolution centered every $n$ pixels where $n$ is the stride.

**Visualization**

Outline

1. Introduction
2. Object recognition
3. General Formulation
4. Support Vector Machine
5. On Trees and Forests
6. Boosting
7. Beyond classification: Dictionary Learning
8. Very small reminder on Convolutional Neural Networks
9. Generative problems
10. Generative Adversarial Networks (GAN)
11. Generative problems beyond images: Geometric data
Generative problems
Generative Problems

Goal

Given a set of samples $x_1, x_2, \ldots, x_n$ (images, signals, animations...) learn a model $p_\theta(x)$ of the true underlying distribution $p(x)$. 
Generative Problems

**Goal**

Given a set of samples $x_1, x_2, \cdots, x_n$ (images, signals, animations...) learn a model $p_\theta(x)$ of the true underlying distribution $p(x)$.

- In practice, we use some prior knowledge of the problem to model $p_\theta$. 
Generative Problems

Goal

Given a set of samples $x_1, x_2, \cdots, x_n$ (images, signals, animations...) learn a model $p_\theta(x)$ of the true underlying distribution $p(x)$.

- In practice, we use some prior knowledge of the problem to model $p_\theta$.
- Optimize $\theta$, to minimize the difference between $p$ and $p_\theta$. 

An almost-training-free approach

Idea

Use a pretrained CNN (ImageNet) and make the features resemble those of the target image (using gradient descent)

- Texture synthesis [Gatys et al. 2015]
- Style transfer [Gatys et al. 2016].
Autoregressive maximum likelihood methods (PixelRNN, PixelCNN) [Van der Oord et al. 2016]

**Idea**
Find the model with the highest likelihood to have generated the data.
Autoregressive maximum likelihood methods (PixelRNN, PixelCNN) [Van der Oord et al. 2016]

Idea

Find the model with the highest likelihood to have generated the data.

- Pixels $x_1, x_2, \cdots x_n$
Autoregressive maximum likelihood methods (PixelRNN, PixelCNN) [Van der Oord et al. 2016]

**Idea**

Find the model with the highest likelihood to have generated the data.

- **Pixels** \( x_1, x_2, \ldots, x_n \)
- \( p_\theta(x_i) = \prod_j p(x_i | x_{i-1}, \ldots, x_n) \)
Autoregressive maximum likelihood methods (PixelRNN, PixelCNN) [Van der Oord et al. 2016]

**Idea**
Find the model with the highest likelihood to have generated the data.

- Pixels $x_1, x_2, \cdots x_n$
- $p_\theta(x_i) = \prod_j p(x_i|x_i-1, \cdots x_n)$

**Process**
Generate pixels sequentially starting from a corner. Dependency on the previous pixels modeled by a Recurrent Neural Network (PixelRNN) or a Convolutional Neural Network (PixelCNN).
Samples trained on ImageNet, 64x64 images.

Pros and Cons

Pros: explicit model of $p_\theta$, Good evaluation metric
Cons: slow because of sequential generation
**Auto-encoders**

**Goal**

Given input data $x$ produce $z$ smaller than $x$ that sums up $x$. 

Image copyright Arden Dertat.
Auto-encoders

**Goal**

Given input data $x$ produce $z$ smaller than $x$ that sums up $x$.

- **Training done by encoding $x$ into $z$, decoding $z$ into $\hat{x}$ and minimizing $\|x - \hat{x}\|^2$.**
**Goal**

Given input data $x$ produce $z$ smaller than $x$ that sums up $x$.

- **Training** done by encoding $x$ into $z$, decoding $z$ into $\hat{x}$ and minimizing $\|x - \hat{x}\|^2$.
- **Latent space** capture data variations Generate new data from a sample in the latent space.
Auto-encoders

**Goal**

Given input data \( x \) produce \( z \) smaller than \( x \) that sums up \( x \).

- **Training** done by encoding \( x \) into \( z \), decoding \( z \) into \( \hat{x} \) and minimizing \( \| x - \hat{x} \|^2 \).
- **Latent space** capture data variations Generate new data from a sample in the latent space.
Auto-encoders

Goal

Given input data \( x \) produce \( z \) smaller than \( x \) that sums up \( x \).

- **Training** done by encoding \( x \) into \( z \), decoding \( z \) into \( \hat{x} \) and minimizing \( \|x - \hat{x}\|^2 \).
- **Latent space** capture data variations. Generate new data from a sample in the latent space.
- If the goal is to compute an **embedding**: after training throw away the decoder. For generating new data, we keep only the decoder.
**Goal**

Given input data $x$ produce $z$ smaller than $x$ that sums up $x$.

- **Training** done by encoding $x$ into $z$, decoding $z$ into $\hat{x}$ and minimizing $\|x - \hat{x}\|^2$.
- **Latent space** capture data variations Generate new data from a sample in the latent space
- If the goal is to compute **an embedding**: after training throw away the decoder. For generating new data, we keep only the decoder.
Latent space

Problem
The data are not spread in the latent space and well clustered.
Variational Auto-encoder [Kingma and Welling 2016]

**Idea**

Ensure that the data spreads well in the latent space.
Variational Auto-encoder [Kingma and Welling 2016]

Idea

Ensure that the data spreads well in the latent space. Add some noise to embeddings in latent space and decode: the output should still be “valid”.
In practice

Instead of learning a vector embedding, the encoder outputs a covariance and mean.

- To decode, we sample from a Gaussian distribution with predicted covariance and mean and compare the distributions using Kullback-Leibler divergence.
Variational Auto-Encoder (VAE)

- Add an additional encoder $q_\phi(z|x)$ approximating $p_\theta(z|x)$
Variational Auto-Encoder (VAE)

- Add an additional encoder $q_\phi(z|x)$ approximating $p_\theta(z|x)$
- Encoder Network: $q_\phi(z|x)$, gaussian model: $\mu_{z|x}, \Sigma_{z|x}$, can sample $z|x$ (probabilistic encoder).
Variational Auto-Encoder (VAE)

- Add an additional encoder $q_\phi(z|x)$ approximating $p_\theta(z|x)$
- Encoder Network: $q_\phi(z|x)$, gaussian model: $\mu_{z|x}, \Sigma_{z|x}$, can sample $z|x$ (probabilistic encoder).
- Decoder Network: $p_\theta(x|z)$, gaussian model: $\mu_{x|z}, \Sigma_{x|z}$, can sample $x|z$ (probabilistic decoder).

[Li, Johnson, Yeung, 2017]
Variational Auto-Encoder (VAE)

- Add an additional encoder $q_\phi(z|x)$ approximating $p_\theta(z|x)$
- Encoder Network: $q_\phi(z|x)$, gaussian model: $\mu_{z|x}, \Sigma_{z|x}$, can sample $z|x$ (probabilistic encoder).
- Decoder Network: $p_\theta(x|z)$, gaussian model: $\mu_{x|z}, \Sigma_{x|z}$, can sample $x|z$ (probabilistic decoder).

[Li, Johnson, Yeung, 2017]
Objective function in a VAE

Minimization

Computing parameters $\theta, \phi$ maximizing:
\[
\mathcal{L}(x_i, \theta, \phi) = \log p_\theta(x_i) \geq E_{z \sim q_\phi(z|x_i)} \left[ \log p_\theta(x_i|z) \right] - D_{KL}(q_\phi(z|x_i) || p_\theta(z))
\]
Objective function in a VAE

Minimization

Computing parameters $\theta$, $\phi$ maximizing:

$$
\mathcal{L}(x_i, \theta, \phi) = \log p_\theta(x_i) \geq \mathbb{E}_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{KL}(q_\phi(z|x_i)||p_\theta(z))
$$

- $\mathcal{L}(x_i, \theta, \phi)$ is a lower bound of $p_\theta(x_i)$
Image generation using VAE

- Sample $z$ from gaussian prior (diagonal covariance).

Remark: Diagonal covariance for $z$ yields independent latent variables corresponding to interpretable factors of variation.

Pros & Cons:
- Pros: Interpolation possible in latent space. Latent variables can be interpretable.
- Cons: Maximizes a lower bound of the likelihood, blurry results.
Image generation using VAE

- Sample $z$ from gaussian prior (diagonal covariance).
- Run $z$ through the decoder, yielding $\mu_x|z$ and $\Sigma_x|z$.

**Pros & Cons**

**Pros:** Interpolation possible in latent space. Latent variables can be interpretable.

**Cons:** Maximizes a lower bound of the likelihood, blurry results.

Generative problems
Image generation using VAE

- Sample $z$ from gaussian prior (diagonal covariance).
- Run $z$ through the decoder, yielding $\mu_x|z$ and $\Sigma_x|z$.
- Sample $\hat{x}$ from $\mathcal{N}(\mu_x|z, \Sigma_x|z)$. 

Remark: Diagonal covariance for $z$ yields independent latent variables corresponding to interpretable factors of variation.

Pros & Cons

Pros: Interpolation possible in latent space. Latent variables can be interpretable.

Cons: Maximizes a lower bound of the likelihood, blurry results.
Image generation using VAE

- Sample $z$ from gaussian prior (diagonal covariance).
- Run $z$ through the decoder, yielding $\mu_x|z$ and $\Sigma_x|z$.
- Sample $\hat{x}$ from $\mathcal{N}(\mu_x|z, \Sigma_x|z)$.

Remark

Diagonal covariance for $z$ yields independent latent variables corresponding to interpretable factors of variation.
Image generation using VAE

- Sample $z$ from gaussian prior (diagonal covariance).
- Run $z$ through the decoder, yielding $\mu_x|z$ and $\Sigma_x|z$.
- Sample $\hat{x}$ from $\mathcal{N}(\mu_x|z, \Sigma_x|z)$.

**Remark**
Diagonal covariance for $z$ yields independent latent variables corresponding to interpretable factors of variation.

**Pros & Cons**

**Pros:** Interpolation possible in latent space. Latent variables can be interpretable.

**Cons:** Maximizes a lower bound of the likelihood, blurry results.
VAE Applications

$\alpha=0 \rightarrow \alpha=1$

- add smiling vector
- subtract smiling vector
- add sunglass vector
- add sunglass vector
- subtract sunglass vector

$\alpha=0 \rightarrow \alpha=1$

$z \sim \mathcal{N}(0, 1)$
Outline

1 Introduction
2 Object recognition
3 General Formulation
4 Support Vector Machine
5 On Trees and Forests
6 Boosting
7 Beyond classification: Dictionary Learning
8 Very small reminder on Convolutional Neural Networks
9 Generative problems
10 Generative Adversarial Networks (GAN)
11 Generative problems beyond images: Geometric data
GAN Principle

- We are not going to model explicitly the density $p_\theta(x)$
- But we will be able to sample from it!
- Sample from a simple distribution and learn the transform to the training distribution.
GAN Principle

- We are not going to model explicitly the density $p_\theta(x)$
- But we will be able to sample from it!
- Sample from a simple distribution and learn the transform to the training distribution.

**Generative Adversarial training**

Admit you have an oracle $D$ that rates if an image $I$ looks *real* ($D(I) = 1$) or *unreal* ($D(I) = 0$). If you want to synthesize an image, you want this oracle to judge the synthesized image as *real*. 
GAN Principle

- We are not going to model explicitly the density $p_\theta(x)$
- But we will be able to sample from it!
- Sample from a simple distribution and learn the transform to the training distribution.

Generative Adversarial training

Admit you have an oracle $D$ that rates if an image $I$ looks real ($D(I) = 1$) or unreal ($D(I) = 0$). If you want to synthesize an image, you want this oracle to judge the synthesized image as real.

- Sadly, we have no oracle $D$ available.
2 players Game

$G$ tries to synthesize images that will fool $D$ and $D$ tries to distinguish between real images and fake images synthesized by $G$. 
G tries to synthesize images that will fool $D$ and $D$ tries to distinguish between real images and fake images synthesized by $G$.

**Objective Function**

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{x \sim p_{data}(x)}[\log D_{\theta_D}(x)] + \mathbb{E}_{z \sim p_{prior}(z)}[\log(1 - D_{\theta_D}(G_{\theta_G}(z)))]$$

Where $\theta_D$ (resp. $\theta_G$) are the parameters of the discriminator (resp. generator).
GAN training

Alternate optimization

Alternate between

1. Optimize parameters $\theta_D$ by gradient ascent ($\theta_G$ fixed).
2. Optimize parameters $\theta_G$ by gradient descent ($\theta_D$ fixed).
GAN training

Alternate optimization

Alternate between

1. Optimize parameters $\theta_D$ by gradient ascent ($\theta_G$ fixed).
2. Optimize parameters $\theta_G$ by gradient descent ($\theta_D$ fixed).

Do we need all the terms of the objective functions for the two steps?
GAN training

Alternate optimization

Alternate between

1. Optimize parameters $\theta_D$ by gradient ascent ($\theta_G$ fixed).
2. Optimize parameters $\theta_G$ by gradient descent ($\theta_D$ fixed).

Do we need all the terms of the objective functions for the two steps?

Problem

In practice hard to optimize! Alternative:

1. Optimize parameters $\theta_D$ by gradient ascent ($\theta_G$ fixed).
2. Optimize parameters $\theta_G$ by gradient ascent ($\theta_D$ fixed) with objective:

$$\max_{\theta_G} \mathbb{E}_{z \sim p_{\text{prior}}(z)} \log D_{\theta_D}(G_{\theta_G}(z))$$
Algorithm 1: Training

for \( j = 1 \cdots N \) do
  for \( k = 1 \cdots K \) do
    Sample a minibatch of \( m \) samples \( z_i \);
    Sample a minibatch of \( m \) real samples \( x_i \);
    Update \( \theta_D \):
    \[
    \theta_D = \theta_D + \nu \nabla \theta_D \left( \sum_{i=1}^{m} \log D_{\theta_D}(x_i) + \log(1 - D_{\theta_D}(G_{\theta_G}(z_i))) \right)
    \]
  
  Sample a minibatch of \( m \) samples \( z_i \);
  Update \( \theta_G \):
  \[
  \theta_G = \theta_G + \nu \nabla \theta_G \left( \sum_{i=1}^{m} \log(D_{\theta_D}(G_{\theta_G}(z_i))) \right)
  \]
Training Algorithm

Algorithm 2: Training

1. for $j = 1 \cdots N$ do
2.     for $k = 1 \cdots K$ do
3.         Sample a minibatch of $m$ samples $z_i$;
4.         Sample a minibatch of $m$ real samples $x_i$;
5.         Update $\theta_D$:
6.             $\theta_D = \theta_D + \nu \nabla_{\theta_D} \left( \sum_{i=1}^{m} \log D_{\theta_D}(x_i) + \log(1 - D_{\theta_D}(G_{\theta_G}(z_i))) \right)$
7.         Sample a minibatch of $m$ samples $z_i$;
8.         Update $\theta_G$:
9.             $\theta_G = \theta_G + \nu \nabla_{\theta_G} \left( \sum_{i=1}^{m} \log(D_{\theta_D}(G_{\theta_G}(z_i))) \right)$

Generation

Sample $z$ and generate $\hat{x} = G(z)$. 

Generative Adversarial Networks (GAN)
Training Algorithm

Algorithm 3: Training

for \( j = 1 \cdots N \) do
  for \( k = 1 \cdots K \) do
    Sample a minibatch of \( m \) samples \( z_i \);
    Sample a minibatch of \( m \) real samples \( x_i \);
    Update \( \theta_D \):
    \[
    \theta_D = \theta_D + \nu \nabla_{\theta_D} \left( \sum_{i=1}^{m} \log D_{\theta_D}(x_i) + \log(1 - D_{\theta_D}(G_{\theta_G}(z_i))) \right)
    \]
  Sample a minibatch of \( m \) samples \( z_i \);
  Update \( \theta_G \):
  \[
  \theta_G = \theta_G + \nu \nabla_{\theta_G} \left( \sum_{i=1}^{m} \log(D_{\theta_D}(G_{\theta_G}(z_i))) \right)
  \]

Generation

Sample \( z \) and generate \( \hat{x} = G(z) \). \( D \) is not needed.
Results

[Goodfellow et al. 2014]
What are $D$ and $G$?

Deep convolutional GANs
GAN analysis

Pros and Cons

Pros: State-of-the-art results, difficult to quantify the quality of the results.
Cons: Difficult to train, cannot produce the explicit density.

Architecture guidelines for stable Deep Convolutional GANs
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

[Radford et al. 2016]
Latent space arithmetic

[Radford et al. 2016]
Latent space arithmetic
Comparison: pixel space arithmetic
**Conditional GANs**

**cGAN idea**

Condition $G$ and $D$ on some additional variable $y$. Feed $y$ to both $G$ and $D$.

---

**Objective Function**

$$
\min_G \max_D E_{x \sim p_{\text{data}}} \left[ \log D(x|y) \right] + E_{z \sim p_{\text{prior}}} \left[ (1 - \log D(G(z)|y)) \right]
$$
Conditional GANs

cGAN idea

Condition $G$ and $D$ on some additional variable $y$. Feed $y$ to both $G$ and $D$.

Objective Function

$$\min_G \max_D \mathbb{E}_x \sim p_{data} \left[ \log D(x|y) \right] + \mathbb{E}_{z \sim p_{prior}} \left[ (1 - \log D(G(z)|y)) \right]$$
Results of conditional GAN

<table>
<thead>
<tr>
<th>User tags + annotations</th>
<th>Generated tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>montanha, trem, inverno, frio, people, male, plant life, tree, structures, transport, car</td>
<td>taxi, passenger, line, transportation, railway station, passengers, railways, signals, rail, rails</td>
</tr>
<tr>
<td>food, raspberry, delicious, homemade</td>
<td>chicken, fattening, cooked, peanut, cream, cookie, house made, bread, biscuit, bakes</td>
</tr>
<tr>
<td>water, river</td>
<td>creek, lake, along, near, river, rocky, treeline, valley, woods, waters</td>
</tr>
<tr>
<td>people, portrait, female, baby, indoor</td>
<td>love, people, posing, girl, young, strangers, pretty, women, happy, life</td>
</tr>
</tbody>
</table>
Conditioning on images

Image-to-Image Translation [Isola et al. 2017]

$y$ is now an image we want to transform (sketch to object, day to night, B/W to color...). Other formulation:

$$
\min_G \max_D E_{x,y \sim p_{data}}[\log D(x, y)] + E_{y \sim p_{data}, z \sim p_{prior}}[\log (1 - D(G(z, y)|y))] \\
+ \lambda E_{x,y,z}[\|x - G(z, y)\|_1]
$$

- Additional term favors resemblance to true result and produces better results [Pathak et al. 2014]
Conditioning on images
Denoising Diffusion for Image synthesis: **Dall-E 2**

![A still of Homer Simpson in The Blair Witch Project](image)

**Growing field**

Dall-E 2 [Ramesh et al. 2022] - ~ April; Stable Diffusion [Rombach et al. 2022] ~ September (but also: Imagen...).
Dates back to: [Sohl-Dickstein et al. 2015] [Ho et al. 2020]
Principle

2 stages:
- Learn a CLIP (text+image) embedding for a caption
- Generate an image from the image embedding
CLIP [Radford et al. 2021]

1. Contrastive pre-training

Learns which caption goes with which image.
To build $P(x|y)$

- Learns a prior $P(z_i|y)$ that produces CLIP image embeddings $z_i$ conditioned on captions $y$.
- Learns a decoder $P(x|z_i)$ or $P(x|z_i, y)$

**Key Ingredient**

Diffusion-based data generation
Diffusion-based data generation

Blur an image until you get a noisy image, learn the reverse process.
Diffusion Process

- Given $x_0 \sim q(x_0)$, generate a Markov chain by adding noise
  \[ p(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1 - \sqrt{\alpha_t})I) \]
- If the noise is large enough $x_T$ can be sampled using $\mathcal{N}(0, I)$
- Iteratively remove the noise by learning a model $\mathcal{N}((\mu(x_t), \Sigma(x_t))$ approximating the true posterior $p(x_{t-1}|x_t)$
- Better: predict the added noise minimizing

\[
L_{simple} = \mathbb{E}_{t \sim [0, T], x_0 \sim q(x_0), \varepsilon \sim \mathcal{N}(0, I)} \left[ \| \varepsilon - \varepsilon_\theta(x_t, t) \|^2 \right]
\]
Some more details

- Generate 64x64 images
- Upsampling through two Diffusion-based upsampler models (256x256, 1024x1024).
Results

- Vibrant portrait painting of Salvador Dalí with a robotic half face.
- A Shiba Inu wearing a beret and black turtleneck.
- A close-up of a hand palm with leaves growing from it.
- An espresso machine that makes coffee from human souls, Aristotelian.
- A panda mad scientist mixing sparkling chemicals, Aristotelian.
- A Corgi’s head depicted as an explosion of a nebula.
- A dolphin in an astronaut suit on Saturn, Aristotelian.
- A propaganda poster depicting a cat dressed as French Emperor Napoleon holding a piece of cheese.
- A teddy bear on a skateboard in Times Square.

Generative Adversarial Networks (GAN)
Decoding an input image
Interpolation

[Image of dog and Starry Night paintings]

Generative Adversarial Networks (GAN)
Text differences

- A photo of a cat → an anime drawing of a super saiyan cat, Artstation
- A photo of a Victorian house → a photo of a modern house
- A photo of an adult lion → a photo of a lion cub
- A photo of a landscape in winter → a photo of a landscape in fall
Outline

1 Introduction
2 Object recognition
3 General Formulation
4 Support Vector Machine
5 On Trees and Forests
6 Boosting
7 Beyond classification: Dictionary Learning
8 Very small reminder on Convolutional Neural Networks
9 Generative problems
10 Generative Adversarial Networks (GAN)
11 Generative problems beyond images: Geometric data
Geometric Deep Learning

- No image-like grid structure
- What is a good representation for working on geometric data?
- Various representations: Meshes, Point sets... → Networks adapted to this kind of data
An example for generating shapes [GRASS, Li et al. 2017]

- Input data: set of shapes with a semantic segmentation into parts.
Algorithm

- **Step 1:** Learn a code representing an arrangement of boxes.
- **Step 2:** Train a GAN for generating a new structure.
- **Step 3:** Use voxelization in each box to synthesize the local geometry.

![Diagram](image-url)
Step 1: Learn a code

Key idea

Shape components are commonly arranged or perceived to be arranged hierarchically. Goal of the code: encode this hierarchy of parts.

Recursive auto-encoder for binary trees: encode the structure into a code; decode and compare the recovered structure.

Recursively merge parts that are either adjacent or symmetric (rotational, translational, reflectional).

Training: generate plausible hierarchies for each shape (sample the space of plausible part groupings).

Adjacency and Symmetry encoder/decoder (transform a code into another encodes the symmetry and the generator).

Additionally: Box encoder/Node classifier.

Fig. 3. Merging criteria used by our model demonstrated with 3D shapes represented by part bounding boxes (relevant parts highlighted in red). From left: (a) two adjacent parts, (b) translational symmetry, (c) rotational symmetry, and (d) reflective symmetry.

[Li et al. 2017]
Learned hierarchies

In a nutshell

Transform a binary tree into a meaningful hierarchy while minimizing the loss (sum of bounding boxes distances)
Etape 2: encoder-decoder model for generation

- Idea: adversarial training: the generator tries to fool the discriminator which in turns tries to detect generated pairs.
- Prior structure for the input to the generator (sample from the set of input and generated output hierarchies for the auto-encoder + other tricks)
Etape 3: geometry synthesis

- **Goal**: synthesize a coherent voxel grid for each bounding box representing the fine-grained geometry
- **Take into account both the geometry of the bounding box and its context**
Contextual description and final synthesis

Generative problems beyond images: Geometric data
Application: interpolation

[Li et al. 2017]

Generative problems beyond images: Geometric data
### Application: shape query

<table>
<thead>
<tr>
<th>Query</th>
<th>Top ranked box structures</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Chair" /></td>
<td><img src="image2" alt="Box Structure 1" /></td>
</tr>
<tr>
<td><img src="image3" alt="Chair" /></td>
<td><img src="image4" alt="Box Structure 2" /></td>
</tr>
<tr>
<td><img src="image5" alt="Chair" /></td>
<td><img src="image6" alt="Box Structure 3" /></td>
</tr>
<tr>
<td><img src="image7" alt="Chair" /></td>
<td><img src="image8" alt="Box Structure 4" /></td>
</tr>
</tbody>
</table>

[Li et al. 2017](#)
Neural Radiance Field (Nerf [Mildenhall et al. 2020])

- Goal: Generate a new view from a set of views

Input Images → Optimize NeRF → Render new views
Principle

Neural network takes as input a 3D coordinate and viewing direction and outputs the volume density and view-dependent emitted radiance at this location and direction.

- Cameras are calibrated (ie we know their positions, orientations and intrinsic parameters)
Neural net $F_\Theta : (x, y, z, \theta, \phi) \rightarrow (R, G, B, \sigma)$: Fully connected layers

• Volume rendering by querying along viewing directions.
• Sampling along the rays to estimate the rendering integral
• Comparison with the ground truth color on the target image
More tricks

- Add a positional encoding to improve high resolution details
- View-dependent radiance is what allows to capture mirror reflections
Results

Video: https://www.matthewtancik.com/nerf
Training time

The optimization for a single scene typically take around 100–300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days).

Video: https://www.matthewtancik.com/nerf
Video: https://www.matthewtancik.com/nerf

Training time

The optimization for a single scene typically take around 100–300k iterations to converge on a single NVIDIA V100 GPU (about 1–2 days). \textit{(Faster variants released since - Plenoxels, InstantNGP...)}
Conclusion

- Many more applications of Machine Learning to investigate
- Machine Learning:
  - Irregularly sampled data? Geometry?
- Several open questions on Deep Learning
  - Given a problem, find a rule to design the network?
  - Metaparameters tuning?
Some reading