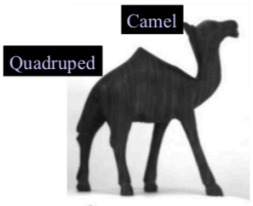
Recognition

Given a database of objects and an image determine what, if any of the objects are present in the image.

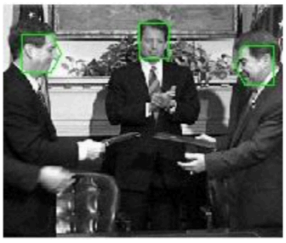


Problem:

Recognizing instances

Recognizing categories

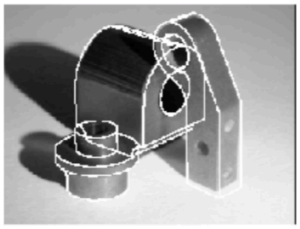
Recognition



Given a database of objects and an image determine what, if any of the objects are present in the image.

... and where are they and how many

Recognition

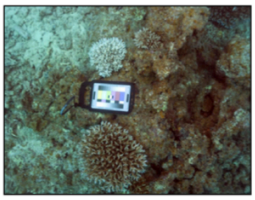


Given a database of objects and an image determine what, if any of the objects are present in the image.

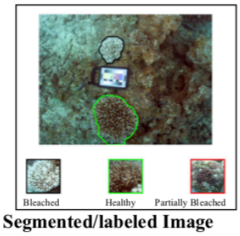
... and where are they and how many

... and where might be 3D pose

Where are the coral heads and which ones are healthy and which are bleached?



Input Image

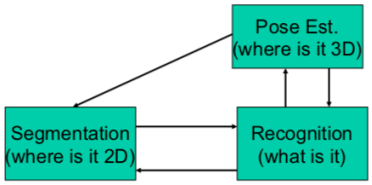


Segmented/labeled Image

Object Recognition: The Problem

Given: A database D of "known" objects and an image I:

1. Determine which (if any) objects in D appear in I
2. Determine the pose (rotation and translation) of the object



WHAT AND WHERE!!!

Recognition Challenges

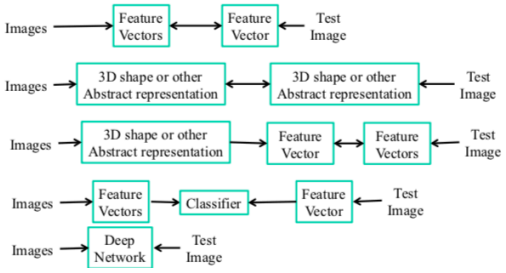
- Within-class variability
  - Different objects within the class have different shapes or different material characteristics
  - Deformable
  - Articulated
  - Compositional
- Pose variability:
  - 2-D Image transformation (translation, rotation, scale)
  - 3-D Pose Variability (perspective, orthographic projection)
- Lighting
  - Direction (multiple sources & type)
  - Color
  - Shadows
- Occlusion – partial occlusion, self occlusion
- Clutter in background -> false positives

Object Recognition System Design Considerations

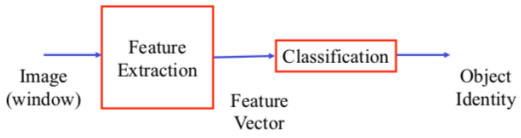
- How general is the problem?
  - 2D vs. 3D
  - range of viewing conditions
  - available context
  - segmentation cues
- What sort of data is best suited to the problem?
  - Whole images
  - Local 2D features (color, texture,
  - 3D (range) data
- What information do we have in the database?
  - Collection of images?
  - 3-D models?
  - Learned representation?
  - Learned classifiers?
- How many objects are involved?
  - small: brute force search
  - large: ??

The Recognition Progression

1 or more images per class for training

- Template matching
- 

Sketch of a Pattern Recognition Architecture



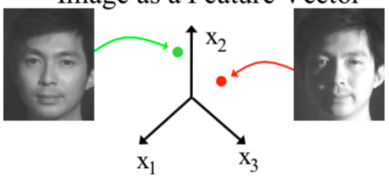
Example: Face Detection

- Scan window over image.
- Classify window as either:
  - Face
  - Non-face

See for example, Viola-Jones face detector in OpenCV

- So, what are the features?
- So, what is the classifier

Simplest feature: Image as a Feature Vector



- Consider an n-pixel image to be a point in an n-dimensional space, $\mathbf{x} \in \mathbf{R}^n$.
- Each pixel value is a coordinate of **x**.

More features

- Filtered image
- Filter with multiple filters (bank of filters
- Histogram of colors
- Histogram of Gradients (HOG)
- Haar wavelets
- Scale Invariant Feature Transform (SIFT)
- Speeded Up Robust Feature (SURF)

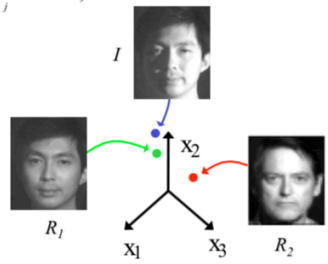
Pattern Classification Summary

- Supervised vs. Unsupervised: Do we have labels?
- Supervised
  - Nearest Neighbor
  - Bayesian
    - Plug in classifier
    - Distribution-based
    - Projection Methods (Fisher's, LDA)
  - Neural Network
  - Support Vector Machine
  - Kernel methods
- Unsupervised
  - Clustering
  - Reinforcement learning

Nearest Neighbor Classifier

$\{ R_j \}$ are set of training images.

$ID = \arg\min_j dist(R_j, I)$



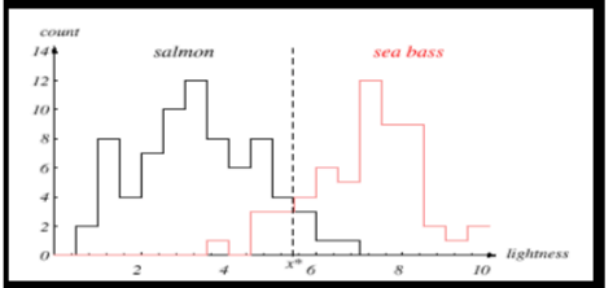
Comments

- Sometimes called "Template Matching"
- Variations on distance function (e.g. L_1, robust distances)
- Multiple templates per class- perhaps many training images per class
- Expensive to compute k distances, especially when each image is big (N dimensional)
- May not generalize well to unseen examples of class
- Will it apply well to other features (height, weight?)
- Some solutions:
  - Bayesian classification
  - Dimensionality reduction

24

An Example

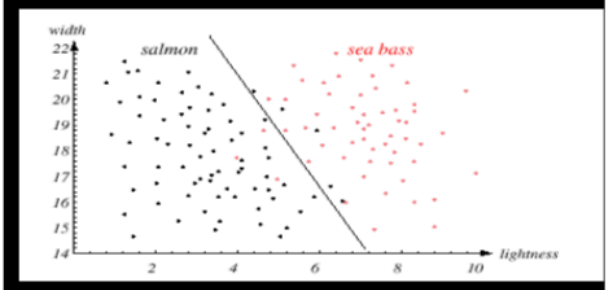
- "Sorting incoming Fish on a conveyor according to species using optical sensing"

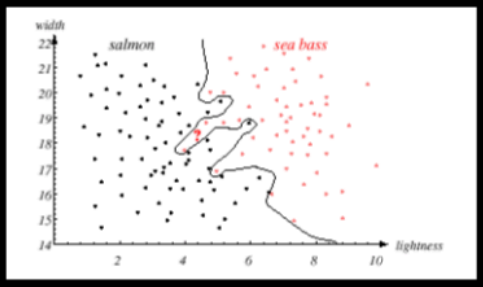


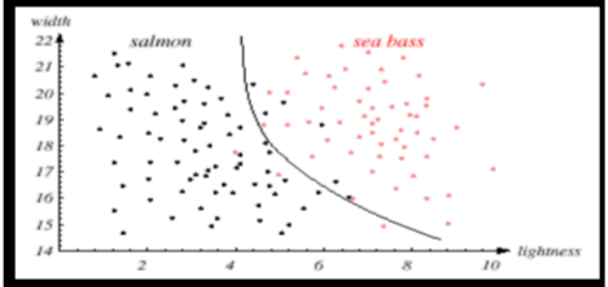
- Adopt the lightness and add the width of the fish

Fish → $x^T = [x_1, x_2]$

Lightness Width







Bayesian Decision Theory
Continuous Features

Introduction

- The sea bass/salmon example
  - State of nature is a random variable, ω_i
    - ω_1 – *the fish is a salmon*
    - ω_2 – *the fish is a sea bass*
  - *Prior Probabilities*
    - $P(\omega_1)$, $P(\omega_2)$
    - $P(\omega_j) > 0$
    - $P(\omega_1) + P(\omega_2) = 1$ (exclusivity and exhaustivity)
  - *Example prior: bass & salmon are equally likely*
    - $P(\omega_1) = P(\omega_2) = ½$ (uniform priors)

- Decision rule with only the prior information
  - Decide ω_1 if $P(\omega_1) > P(\omega_2)$ otherwise decide ω_2
- Use of the class–conditional information
- *Let* $\mathbf{x}$ *be a vector of features*
- $P(\mathbf{x} \mid \omega_1)$ and $P(\mathbf{x} \mid \omega_2)$ describe the distribution in features (say lightness) between populations of sea-bass and salmon.

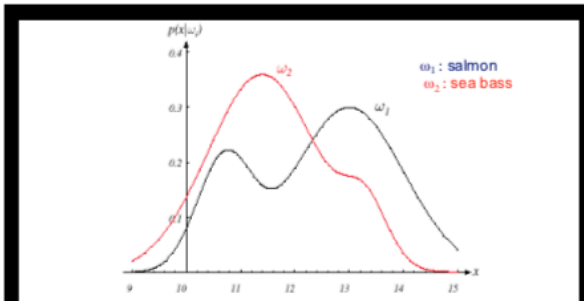


FIGURE 2.1. Hypothetical class-conditional probability density functions show the probability density of measuring a particular feature value x given the pattern is in category ω_i. If x represents the lightness of a fish, the two curves might describe the difference in lightness of populations of two types of fish. Density functions are normalized, and thus the area under each curve is 1.0. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- Posterior, likelihood, evidence
  - $$P(\omega_j \mid x) = \frac{P(x \mid \omega_j) P(\omega_j)}{P(x)}$$ **(BAYES RULE)**
  - In words, this can be said as:
    Posterior = (Likelihood * Prior) / Evidence
  - Where in case of two categories

$$P(x) = \sum_{j=1}^{j=2} P(x \mid \omega_j) P(\omega_j)$$

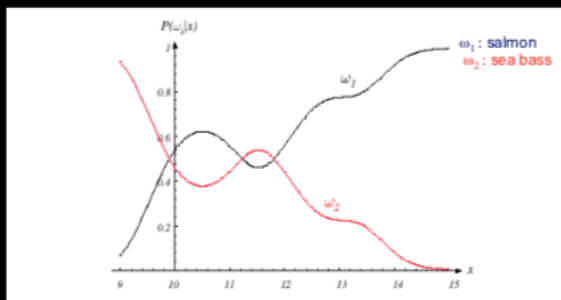
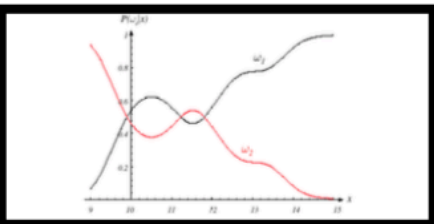


FIGURE 2.2. Posterior probabilities for the particular priors $P(\omega_1) = 2/3$ and $P(\omega_2) = 1/3$ for the class-conditional probability densities shown in Fig. 2.1. Thus in this case, given that a pattern is measured to have feature value $x = 14$, the probability it is in category ω_2 is roughly 0.08, and that it is in ω_1 is 0.92. At every x, the posteriors sum to 1.0. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.



- Intuitive decision rule given the posterior probabilities:
  Given x:
  if $P(\omega_1 \mid x) > P(\omega_2 \mid x)$ ➡ True state of nature = ω_1
  if $P(\omega_1 \mid x) < P(\omega_2 \mid x)$ ➡ True state of nature = ω_2

  Why do this?: Whenever we observe a particular x, the probability of error is :
  $P(\text{error} \mid x) = P(\omega_1 \mid x)$ *if we decide* ω_2
  $P(\text{error} \mid x) = P(\omega_2 \mid x)$ *if we decide* ω_1

Plug-in classifiers

- Assume that class conditional distributions $P(x|\omega_i)$ have some parametric form with parameters W
- Given training data, estimate parameters W

- Common parametric form:
  - Uniform over region
  - Normal distribution with shared covariance matrix, different means
  - Normal distribution but with different covariance matrices

If our feature space is one dimensional then the "boundary" that separates the area assigned to one class vs. another class is a point.

- But what happens as the dimensionality of our feature space increases?
- Or if there are more than two classes?

Let's think a classifier as set of scalar functions $g_i(\mathbf{x})$ — one for each class *i* — that assigns a score to the vector of feature values **x** and then choses the class *i* with the highest score

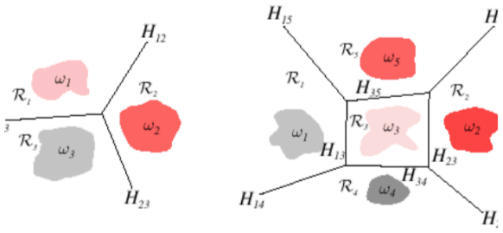
So, a classifier uses the following decision rule:
Choose class *i* if $g_i(x) > g_j(x) \quad \forall j, i \neq j$

So our Bayesian classifier assigns a score based on the *a posteriori* probabilities:

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x})$$

And, if our feature space is n-dimensional, i.e., $\mathbf{x} \in \Re^n$, then the boundaries separating regions that our classifier assigns to the same class is n-1 dimensional surface where $g_i(\mathbf{x})=g_j(\mathbf{x})$

Multiclass − 2D Feature Space



E 5.4. Decision boundaries produced by a linear machine for a three-class d a five-class problem. From: Richard O. Duda, Peter E. Hart, and David G. *Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

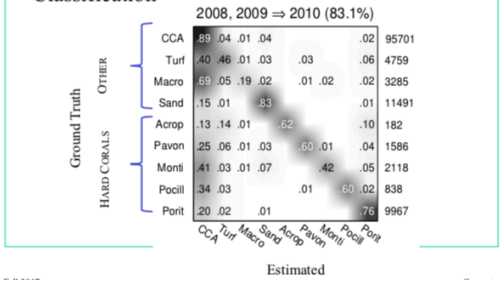
Evaluating Multi-class classifiers

CIFAR 10

60,000 32x32 color images in 10 classes

Evaluating Multi-class classifiers

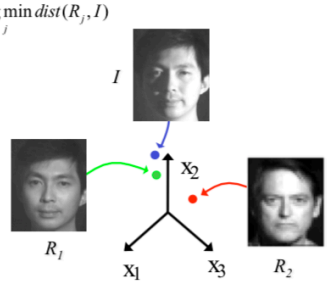
- Overall accuracy
- Confusion Matrix – Example from Coral Reef Classification



Nearest Neighbor Classifier

$\{ R_j \}$ are set of training images.

$ID = \arg\min_j dist(R_j, I)$



K-th Nearest Neighbor Classification

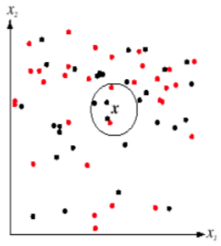


FIGURE 4.15. The k-nearest-neighbor query starts at the test point **x** and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point **x** would be labeled the category of the black points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

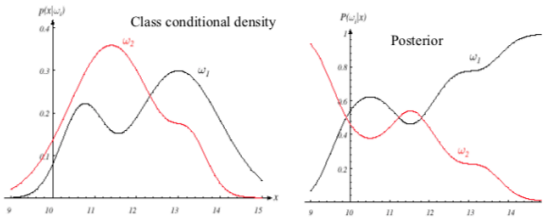
Maximum a posteriori classifier (MAP)

$$g_j(x) = P(\omega_j \mid x) = \frac{P(x \mid \omega_j) P(\omega_j)}{P(x)}$$

$P(x|\omega_j)$: Class conditional density

$P(\omega_j)$: Prior of class j

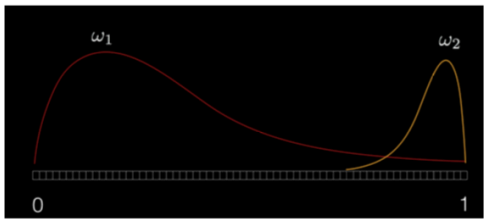
Classification: $\hat{j} = \arg\max_j g_j(x)$



Curse of Dimensionality

- If we want to build a minimum-error rate classifier then we need a very good estimate of $P(\omega_i | \mathbf{x})$
- How do we do this?

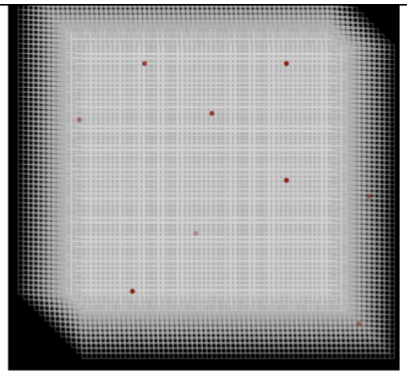
- Let's say our feature space is just 1-dimensional and our feature $\mathbf{x} \in [0,1]$
- And let's say we have **10,000 training samples** from which to estimate our *a posteriori* probabilities.
- We could estimate these probabilities using a histogram in which we divided the interval into 100 evenly spaced bins. .



- On average each bin would have 100 samples.
- We could estimate $P(\mathbf{x} \mid \omega_i)$ as the number of samples from class i that fall in the same bin that falls into divided by the total number of samples in that bin.

But this plan does not scale as we increase the dimensionality of the feature space!

- Let's say our feature space is just 3-D dimensional and our feature $\mathbf{x} \in [0,1]^3$
- Let's say we still have **10,000 training samples** from which to estimate our *a posteriori* probabilities.
- If we estimate these probabilities using a histogram in which we divide the volume into the same width bins as before…



On average each bin would only have 0.01 samples!
We're not going be able to estimate probabilities well

Dimensionality Reduction

Dimensionality reduction: linear projection

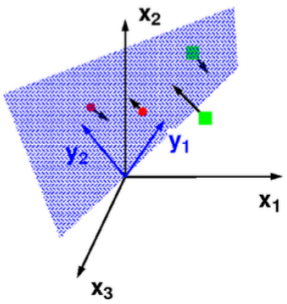
• An n-pixel image $\boldsymbol{x} \in \mathbf{R}^n$ can be projected to a low-dimensional feature space $\boldsymbol{y} \in \mathbf{R}^m$ by

$$\boldsymbol{y} = W^T \boldsymbol{x}$$

where W is an n by m matrix.

• Recognition is performed using nearest neighbor in $\mathbf{R}^m$.

• How do we choose a good W?



How do we choose a good W?

- Drop dimensions (feature selection)
- Random projects
- Principal component analysis
- Linear discriminant analysis
- Independent component analysis
- Or even non-linear dimensionality reduction

Eigenfaces: Principal Component Analysis (PCA)

Assume we have a set of n feature vectors $\boldsymbol{x}_i$ $(i = 1, \ldots, n)$ in $\mathbb{R}^d$. Write

$$\boldsymbol{\mu} = \frac{1}{n} \sum_i \boldsymbol{x}_i$$

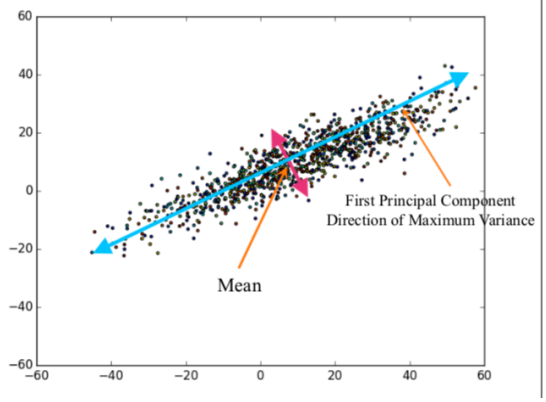
$$\Sigma = \frac{1}{n-1} \sum_i (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$$

The unit eigenvectors of Σ — which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue — give a set of features with the following properties:

- They are independent.
- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that preserves the most variance.

Algorithm 22.5: *Principal components analysis identifies a collection of linear features that are independent, and capture as much variance as possible from a dataset.*

Some details: Use Singular value decomposition, "trick" described in text to compute basis when n<<d



Singular Value Decomposition

- Any m by n matrix $\boldsymbol{A}$ may be factored such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

[m x n] = [m x m][m x n][n x n]

- $\boldsymbol{U}$: m by m, orthogonal matrix
  - Columns of **U** are the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$
- $\boldsymbol{V}$: n by n, orthogonal matrix,
  - columns are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$
- $\boldsymbol{\Sigma}$: m by n, diagonal with non-negative entries ($\sigma_1, \sigma_2, \ldots, \sigma_s$) with s=min(m,n) are called the called the singular values
  - **Singular values are the square roots of eigenvalues of both $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$ & *Columns of U are corresponding Eigenvectors***
  - *Result of SVD algorithm:* $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_s$

SVD Properties

- In Matlab [u s v] = svd(A), and you can verify that: A=u*s*v'
- r=Rank($\boldsymbol{A}$) = # of non-zero singular values.
- $\boldsymbol{U}$, $\boldsymbol{V}$ give us orthonormal bases for the subspaces of $\boldsymbol{A}$:
  - 1st r columns of $\boldsymbol{U}$: Column space of $\boldsymbol{A}$
  - Last m - r columns of $\boldsymbol{U}$: Left nullspace of $\boldsymbol{A}$
  - 1st r columns of $\boldsymbol{V}$: Row space of $\boldsymbol{A}$
  - 1st n - r columns of $\boldsymbol{V}$: Nullspace of $\boldsymbol{A}$
- *For $d \leq r$, the first d columns of U provide the best d-dimensional basis for columns of A in least squares sense.*

Performing PCA with SVD

- Singular values of A are the square roots of eigenvalues of both AA^T and A^TA & Columns of U are corresponding Eigenvectors
- And $\sum_{i=1}^{n} a_i a_i^T = [a_1 \quad a_2 \quad \cdots \quad a_n][a_1 \quad a_2 \quad \cdots \quad a_n]^T = AA^T$
- Covariance matrix is:

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T$$

- So, ignoring 1/n subtract mean image μ from each input image, create data matrix, and perform thin SVD on the data matrix.

Comment on images collections

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$$

[m x n] = [m x m][m x n][n x n]

- The matrix A is sometimes called the data matrix.
- Columns of A are vectorized images.
- So, we have m pixels and n images.
- For large images (e.g., 1kx1k), we often have more n > m. Using SVD is preferred.
- For CIFAR, we have m=3072, and we have n=60k and so explicit form with covariance matrix or SVD can work.

Thin SVD

- Any m by n matrix $\boldsymbol{A}$ may be factored such that

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$$

[m x n] = [m x m][m x n][n x n]

- If m>n, then one can view $\boldsymbol{\Sigma}$ as:

$$\begin{bmatrix} \Sigma' \\ 0 \end{bmatrix}$$

- Where $\boldsymbol{\Sigma}'$=diag(σ_1, σ_2, …, σ_s) with s=min(m,n), and lower matrix is (n-m by m) of zeros.
- Alternatively, you can write:

$$\boldsymbol{A} = \boldsymbol{U'\Sigma' V}^T$$

- In Matlab, thin SVD is: [U S V] = svds(A)

PCA for recognition (Eigenfaces)

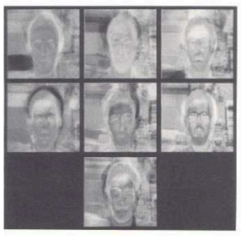
- Modeling
  1. Given a collection of n labeled training images,
  2. Compute mean image and covariance matrix.
  3. Compute k Eigenvectors (note that these are images) of covariance matrix corresponding to k largest Eigenvalues. (Or perform using SVD!!)
  4. Project the training images to the k-dimensional Eigenspace.
- Recognition
  1. Given a test image, project to Eigenspace.
  2. Perform classification to the projected training images.

Eigenfaces: Training Images

[Turk, Pentland 01

Eigenfaces

Mean Image



Basis Images

Accuracy of PCA + K-NN

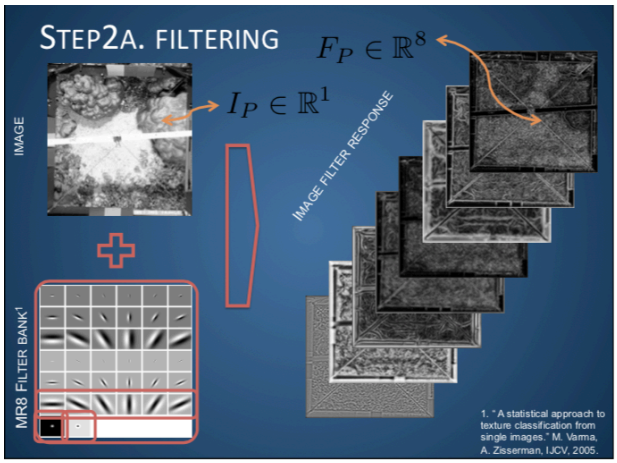
KNN + 3,072 Features	33.86
KNN + 200 PCA Comp.	36.54
KNN + 75 PCA Comp.	39.77
KNN + 50 PCA Comp.	40.12
KNN + 40 PCA Comp.	40.93
KNN + 30 PCA Comp.	**41.78**
KNN + 25 PCA Comp.	41.57
KNN + 15 PCA Comp.	38.75
KNN + 10 PCA Comp.	34.93

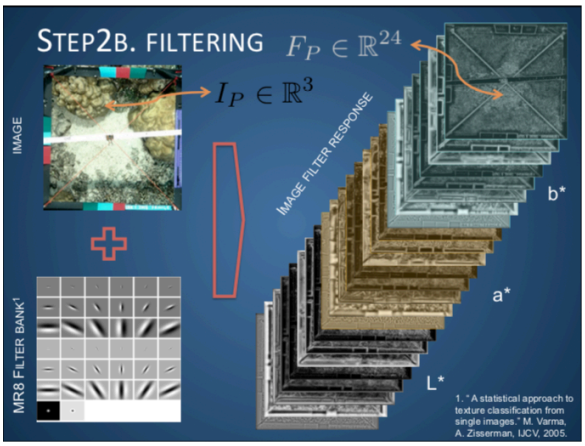
Difficulties with PCA

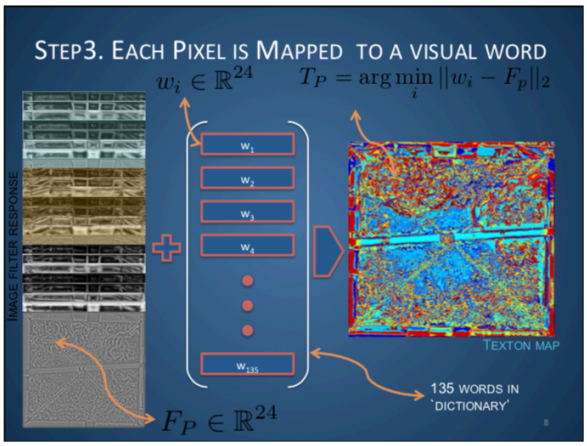
- Projection may suppress important detail
  - smallest variance directions may not be unimportant
- Method does not take discriminative task into account
  - typically, we wish to compute features that allow good discrimination
  - not the same as largest variance or minimizing reconstruction error.

Example of Feature Extraction

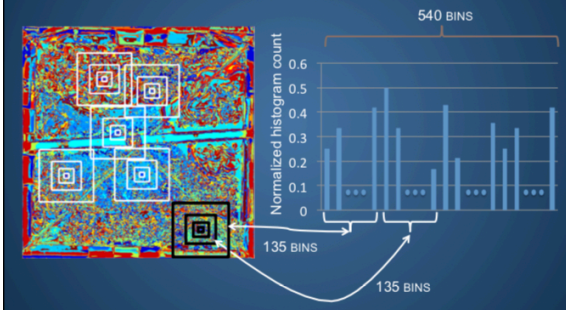
Example taken from Beijbom, Edmunds, Kline, Mitchell, Kriegman, "Automated Annotation of Coral Reef Survey Images", CVPR, 2012 and used on CoralNet alpha web site, coralnet.ucsd.edu







STEP4B. HISTOGRAMS AT MULTIPLE SCALES



Neural Networks

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

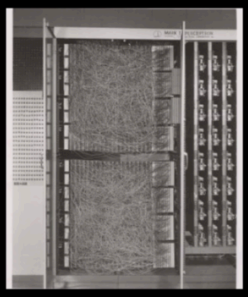
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

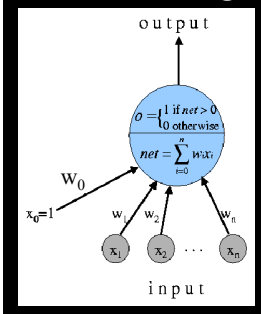
Mark I Perceptron machine

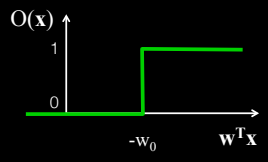
- The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used **20×20 cadmium sulfide photocells to produce a 400-pixel image**. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays of potentiometers that implemented the adaptive weights



[From Wikipedia]

Perceptron





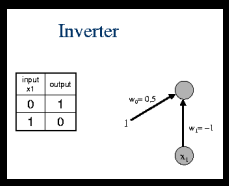
Note: For $x=(x_1,\ldots,x_2)$, x_i can be binary or a real number

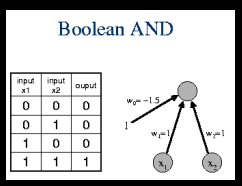
$$o(x_1,\ldots,x_n)=\begin{cases}1 \text{ if } w_0+w_1x_1+\cdots+w_nx_n>0\\ 0 \text{ otherwise.}\end{cases}$$

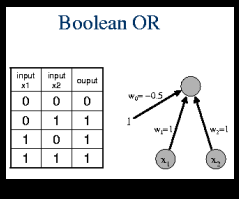
For a Network, even as simple as a single perceptron, we an ask questions:

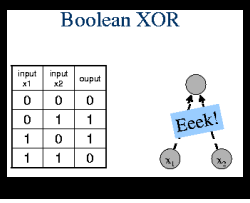
1. What can be represented with it
2. How do we evaluate it?
3. How do we train it?

How powerful is a perceptron?

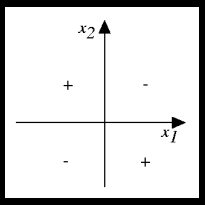


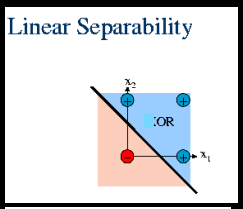


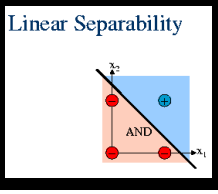


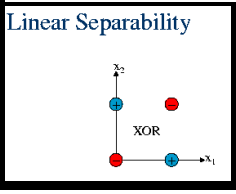


Concept Space & Linear Separability

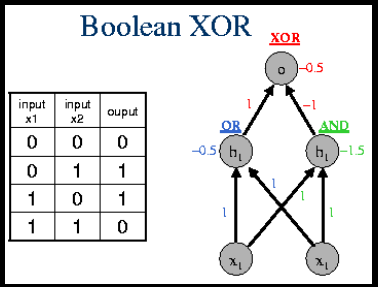








Increasing Expressiveness: Multi-Layer Neural Networks

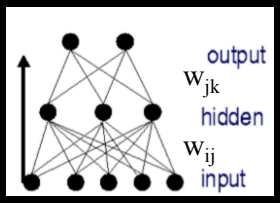


2-layer Perceptron Net

But where did those weights come from?

Stay tuned

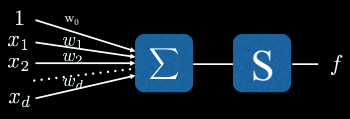
Two Layer Network



- Fully connected network
- Nodes are nonlinear function of weighted sum inputs:

$$f(\mathbf{x};\mathbf{w}) = S(\mathbf{w}^T\mathbf{x}+w_0)$$

The nodes of multilayered network

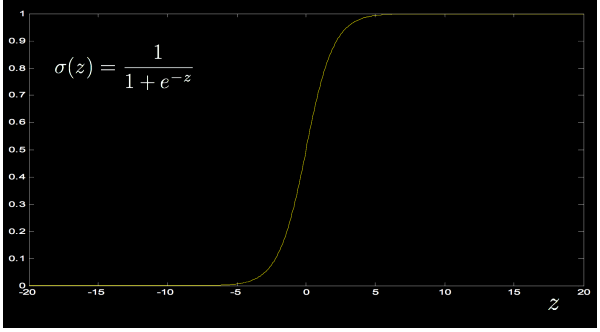


$$f(\mathbf{x};\mathbf{w}) = S(\mathbf{w}^T\mathbf{x}+w_0)$$

Nonlinearities: S(x)

- Threshold (perceptron)
- Sigmoid
- Rectified Linear Unit (ReLU)

Sigmoid Function



Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

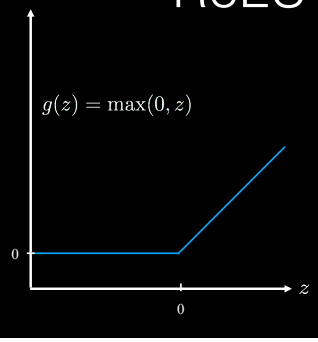
- As z goes from $-\infty$ to ∞, $\sigma(z)$goes from 0 to 1
- It has a "sigmoid" or S-like shape

$\sigma(0) = 0.5$

Using the sigmoid as a classifier

$$\text{if } \sigma(f(\mathbf{x})) \geq 0.5 \rightarrow +1$$
$$< 0.5 \rightarrow -1$$

Rectified Linear Unit ReLU



Feedforward Networks

- Let $y = f^*(\mathbf{x})$ be some function we are trying to approximate
- This function could be assignment of an input to a category as in a classifier
- Let a feedforward network approximate this mapping $y=f(x; w)$ by learning parameters w

Feedforward Networks

- Feedforward networks have **NO** feedback
- These networks can be represented as directed acyclic graphs describing the composition of functions
- These networks are composed of functions represented as **"layers"** $f(\mathbf{x}) = l^3(l^2(l^1(\mathbf{x})))$
- The length of the chain of compositions gives the **"depth"** of the network

Feedforward Networks

- The functions defining the layers have been influenced by neuroscience
- Our training dictates the values to be produced output layer and the weights are chosen accordingly
- The weights for intermediate or **"hidden"** layers are learned and not specified directly
- You can think of the network as mapping the raw input space $\mathbf{x}$ to some transformed feature space $\phi(\mathbf{x})$ where the samples are ideally linearly separable

Universal Approximation Theorem

- tldr: if we have enough hidden units we can approximate "any" function! … but we may not be able to train it.
- **Universal Approximation Theorem**: A feedforward neural network with a linear output layer and one or more hidden layers with ReLU **[Leshno et al. '93]**, or sigmoid or some other "squashing" activation function **[Hornik et al. '89, Cybenko '89]** can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$. This holds for functions mapping finite dimensional discrete spaces as well.

Universal Approximation Theorem: Caveats

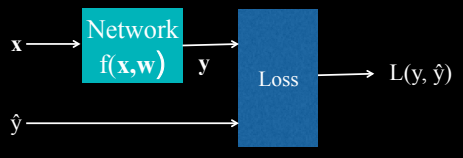
- Optimization may fail to find the parameters needed represent the desired function.
- Training might choose the wrong function due to overfitting.
- The network required to approximate this function might be so large as to be infeasible.

Universal Approximation Theorem: Caveats

- So even though "any" function can be approximated with a network as described with single hidden layer, the network may fail to train, fail to generalize, or require so many hidden units as to be infeasible.
- This is both encouraging and discouraging!
- However, **[Montufar et al. 2014]** showed that **deeper networks are more efficient** in that a deep rectified net can represent functions that would require an exponential number of hidden units in a shallow one hidden layer network.
- Deep networks composed on many rectified hidden layers are good at approximating functions that can be composed from simpler functions. And lots of tasks such as image classification may fit nicely into this space.

High level view of evaluation and training

- Training data: $\{(\mathbf{x}_i, y_i) : 1 \le i \le n\}$



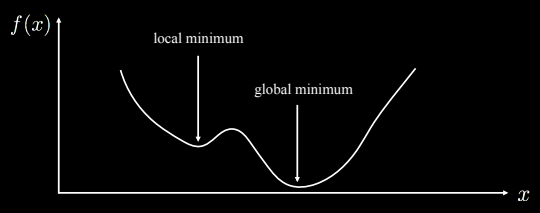
- Total Loss: $\sum_{i=1}^{n} L\left(f\left(\mathbf{x}_i; \mathbf{w}\right), \hat{y}\right)$
- Training: Find $\mathbf{w}$ that minimizes the total loss.

The loss function

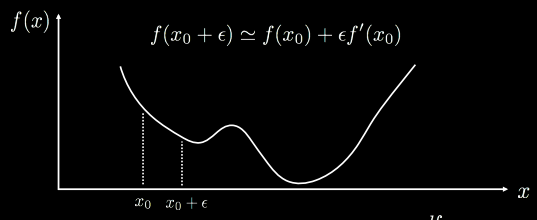
- The loss function is really important. It determines how we compare what the network produces to our labels.
- Common ones:
  - Regression problems:
    - Distance : $\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^p$, usually p = 1 or 2
  - Classification
    - Cross entropy (See homework)
    - Softmax

$$P(y = i|\mathbf{x}) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Gradient-Based Optimization

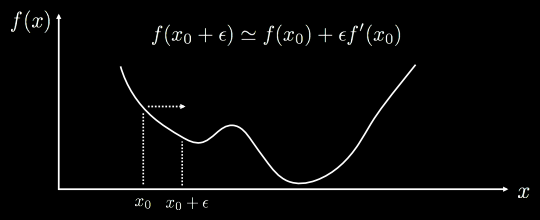


Gradient-Based Optimization



$$f' = \frac{df}{dx}$$

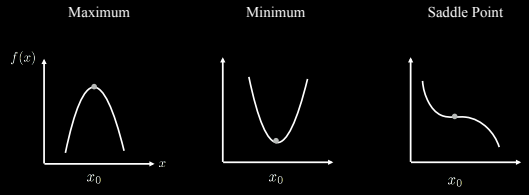
Gradient Descent



Note that f' is negative, so going in positive direction decreases the function.

Critical Points

$$f'(x_0) = 0$$



$f''(x_0) < 0$ $f''(x_0) > 0$ $f''(x_0) = 0$

What if our input is a vector?

- Let $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$
- The ***directional derivative*** is the slope of the function in direction $\mathbf{u}$
- We can find this as $\frac{\partial}{\partial \eta} f(\mathbf{x} + \eta \mathbf{u})$ at $\eta = 0$
- …or after the chain rule yields $\nabla_{\mathbf{x}} f(\mathbf{x})^T \mathbf{u}$

PLEASE DON'T FORGET

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Gradient Descent

- So if we move in direction $\mathbf{u}$ the slope is $\nabla_{\mathbf{x}} f(\mathbf{x})^T \mathbf{u}$
- So in what direction is the slope most negative?
- Clearly in the **OPPOSITE** direction of the gradient!
- And if we traverse the fcn this way then we are doing ***steepest descent*** or ***gradient descent***.

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla_{\mathbf{x}} f(\mathbf{x}_t)$$

Optimization for Deep Nets

- Deep learning optimization is a type global optimization where the optimization is usually expressed as a loss summed over all the training samples.
- Our goal is not so much find the parameters (or weights) that minimize the loss but rather to find parameters that produce a network with the desired behavior.
- Note that there are LOTS of solutions to which our optimization could converge to—with very different values for the weights—but each producing a model with very similar behavior on our sample data.
- For example, consider all the permutations of the weights in a hidden layer that produce the same outputs.

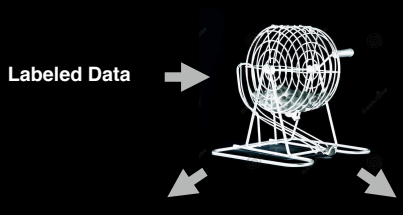
Optimization for Deep Nets

- Although there is a seemingly endless literature on global optimization, here we consider only gradient descent-based methods.
- Our optimizations for deep learning are typically done in very high dimensional spaces, were the parameters we are optimizing can run into the millions.
- And for these optimizations, when starting the training from scratch (i.e., some random initialization of the weights) we will needs LOTS of labeled training data.
- The process of learning our model from this labeled data is referred to as ***supervised learning***. Although, supervised learning is more general than the deep learning algorithms we will consider.

Back propagation

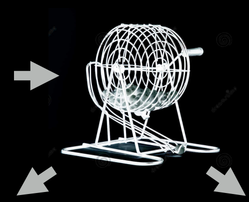
- Basically another name for gradient descent
- Because of nature of network $l_3(l_2(l_1(x;w_1);w_2);w_3)$, gradients with respect to w_i are determined by chain rule
- Can be thought of has "propagating" from loss function to input.
- You'll use adaptive step size method called ADAM.

Training and Validation Sets



Training and Validation Sets

Labeled Data



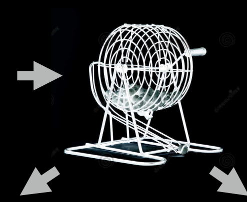
Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training and Validation Sets

Labeled Data



Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training and Validation Sets

Labeled Data

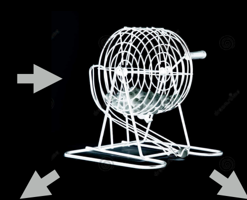
Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training and Validation Sets

Labeled Data



Training Data

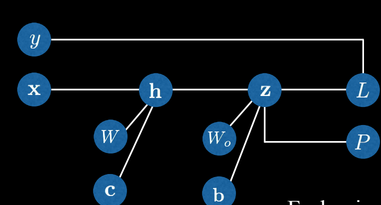
Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Regularization

The goal of ***regularization*** is to prevent **overfitting** the training data with the hope that this improves **generalization**, *i.e.*, the ability to correctly handle data that the network has not trained on.

Regularization for MLP



Frobenius norm

$$L = -z_i + \log \sum_j e^{z_j} + \frac{\lambda}{2}(||W||^2 + ||W_o||^2)$$

loss regularization

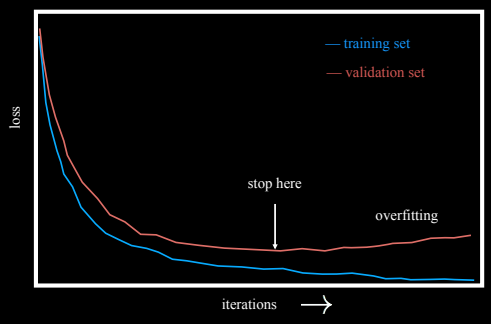
Dropout [Srivastava et al., 2014]

- For every training batch through the network, ***dropout*** 0.5 of hidden units and 0.2 of input units. You can choose the probabilities as you like…
- Train as you normally would using SGD, but each time you impose a random ***dropout*** that essentially trains for that batch on a random sub-network.
- When you are done training, you use for your model the complete network with all its learned weights, except multiply the weight by the probability of including its parent unit.
- This is called the ***weight scaling inference rule***. [Hinton et al., 2012]

Early Stopping

- Typical deep neural networks have millions and millions of weights!
- With so many parameters at their disposal how to we prevent them from overfitting?
- Clearly we can use some of the other regularization techniques that have been mentioned…
- …but given enough training time, our network will eventually start to overfit the data.

Early Stopping



Deterministic vs. Stochastic Methods

- If we performed our gradient descent optimization using all the training samples to compute each step in our parameter updates, then our optimization would be ***deterministic***.
- Confusingly, ***deterministic*** gradient descent algorithms are sometimes referred to as ***batch*** algorithms
- In contrast, when we use a subset of randomly selected training samples to compute each update, we call this ***stochastic gradient descent*** and refer to the subset of samples as a ***mini-batch***.
- And even more confusingly, we often call this mini-batch the "batch" and refer to its size as the "batch size."

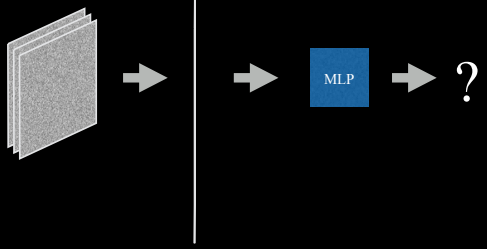
Stochastic Gradient Descent

The SGD algorithm could not be any simpler:

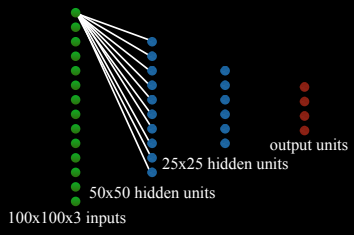
1. Choose a learning rate schedule η_t.
2. Choose stopping criterion.
3. Choose batch size m.
4. Randomly select mini-batch $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(m)}\}$
5. Forward and backpropagation
6. Update $\theta_{t+1} = \theta_t - \eta_t g \qquad \mathbf{g} = \frac{1}{m} \sum_i^m \nabla_\theta L(\mathbf{x}^{(i)}, y^i)$
7. Repeat 4, 5, 6 until the stopping criterion is satisfied.

Finally, we get to images…

What if we just vectorized images and stuffed these into a MLP?



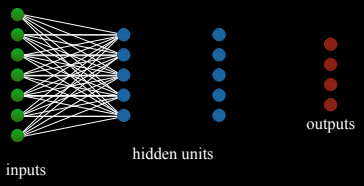
Too many weights and connections!



- This fully connected hidden layer might have 75 million weights!
- And this is just for a thumbnail image and a two layer net.

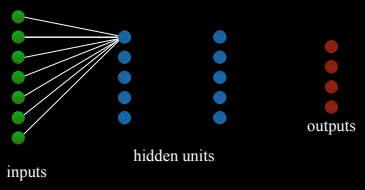
Convolutional Neural Networks

Fully Connected (FC) Layer



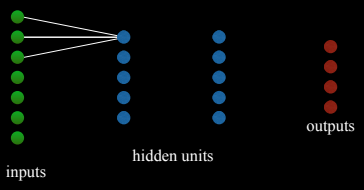
Every input unit is connected to every output unit.

Fully Connected (FC) Layer



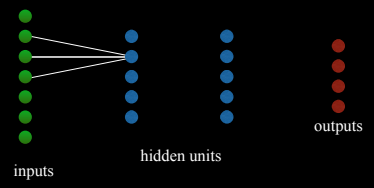
Consider a hidden unit: it connects to all units from the previous layer.

Convolutional Layer: Local Connections



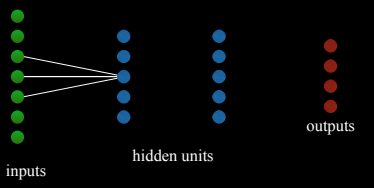
Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Local Connections



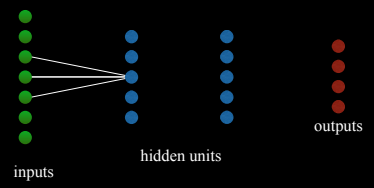
Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Local Connections



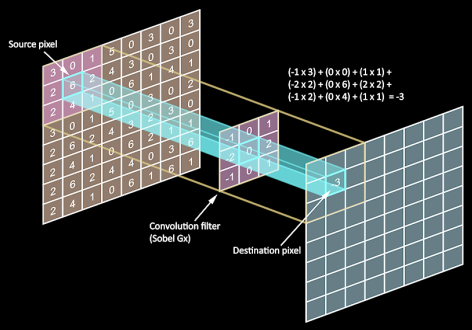
Here the connections are spatially local and governed by the kernel size.

Convolutional Layer: Shared Weights

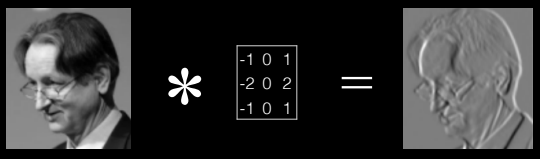


The weights are shared. They are the same for each position.
So, this is like a convolution kernel.

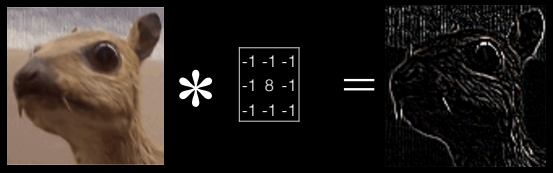
Convolution in 2D



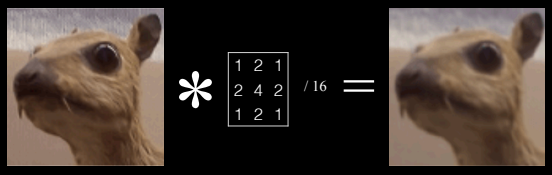
Convolution with 2D Kernel



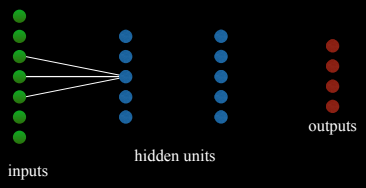
Convolution with 2D Kernel



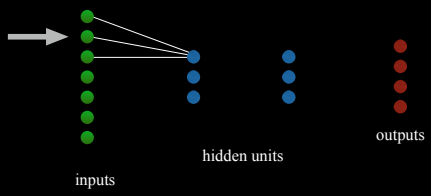
Convolution with 2D Kernel



Convolutional Layer: Shared Weights

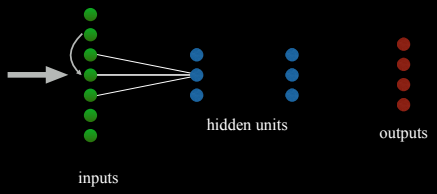


Convolutional Layer: Stride



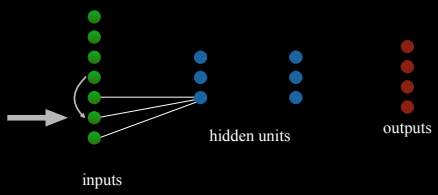
We can skip input pixels by choosing a *stride* > 1.

Convolutional Layer: Stride



We can skip input pixels by choosing a *stride* > 1.

Convolutional Layer: Stride



The output dim = (input dim - kernel size) / stride + 1.

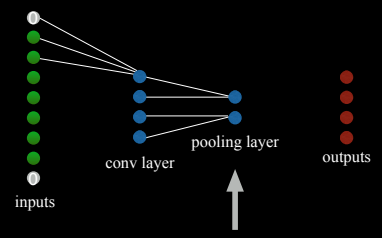
ReLU used with ConvNets

- Just like with our fully connected layers, for our convolutional layers we will follow the linear operation (convolution) with with a non-linear squashing function.
- Again the fcn to use for now is ReLU.
- But we are not done…there's one more thing!

Pooling

- We can spatially pool the output from the ReLU to reduce the size of subsequent layers in our network.
- This reduces both computation and the number of parameters that need to be fit and helps prevent overfitting.
- The pooling operation is often the **max** value in the region, but it could be **average**, or **median**, etc.
- The pooling has a stride associated with it that determines the downsampling of the input.

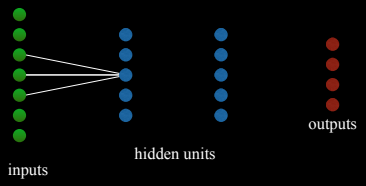
Pooling Layer



The pooling layer pools values in regions of the conv layer.

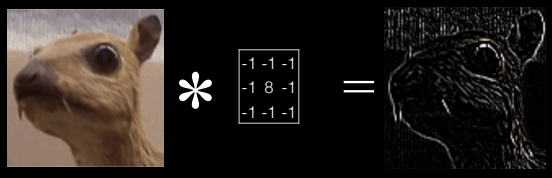
Oops. Just one more thing…Recall

Convolutional Layer: Shared Weights



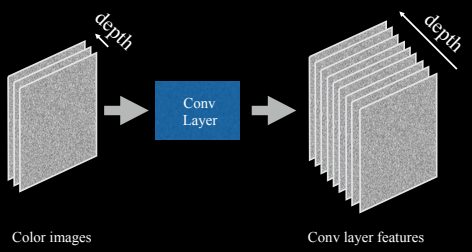
The weights for the kernel are shared. They are the same for each position.

Each kernel finds just one type of feature.



If a kernel shares weights then it can only extract one type of feature.

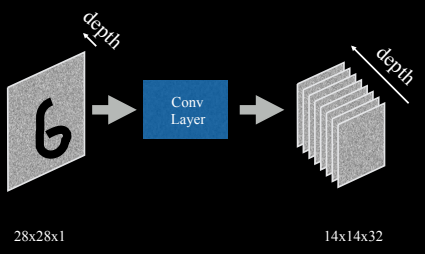
Why not allow for many kernels and many features!



A Convolutional Net

- Let's assume we have 28x28 grayscale images as input to our conv net. So we will input 28x28x1 samples into the net.
- Let's fix our kernel size at 5x5 and, to make this simple, pad our images with zeros and use a stride = 1.
- Let's use max pooling on the output, with a 2x2 pooling region and a stride of 2.
- Let's extract 32 features after the first layer.
- So the output from this layer will be 14x14x32.

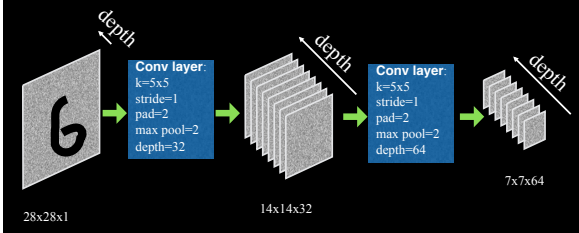
A Convolutional Net



A Convolutional Net

- Now let's make a second layer, also convolutional.
- Let's fix our kernel size at 5x5, pad our images with zeros and use a stride = 1.
- Let's use max pooling on the output again, with a 2x2 pooling region and a stride of 2.
- Let's extract 64 features after the second layer.
- So the output from this layer will be 7x7x64.

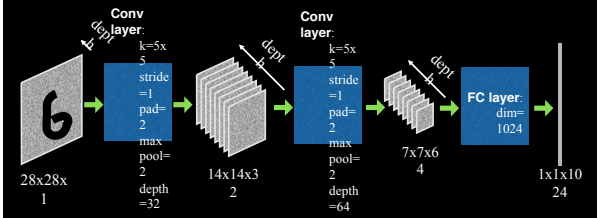
A Convolutional Net



A Convolutional Net

- Our third layer will be a fully connected layer mapping our convolutional features to a 1024 dimensional feature space.
- This layer is just like any of the hidden layers you have created before. It is a linear transformation followed by ReLU.
- So the output from this layer will be 1x1x1024.

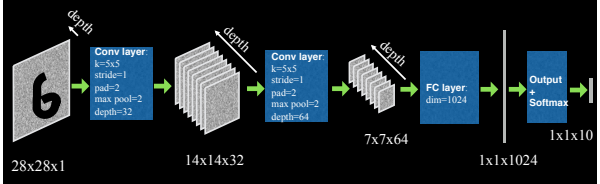
A Convolutional Net



A Convolutional Net

- Finally, will map this feature space to a 10 class output space and use a softmax with a MLE/ cross entropy loss function.
- And…we're done!

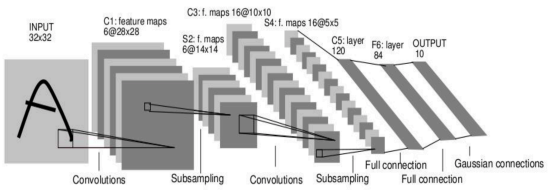
A Convolutional Net



Parameters = (5x5x1x32+32) + (5x5x32x64+64) + (7x7x64x1024+1024) + (1024x10+10)

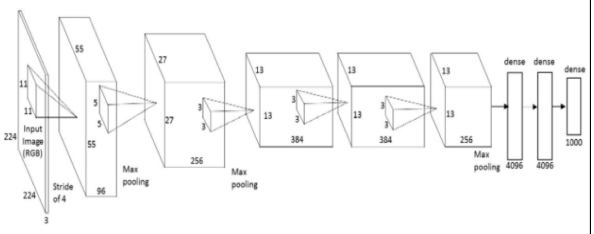
Some Famous Deep Nets and Data sets

LeNet 5

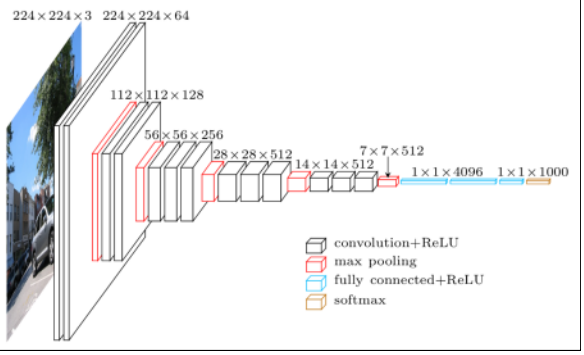


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner,
Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

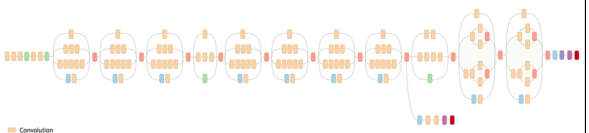
AlexNet [Krizhevsky et al., 2012]



VGG16 [Simonyan and Zisserman, 2014]

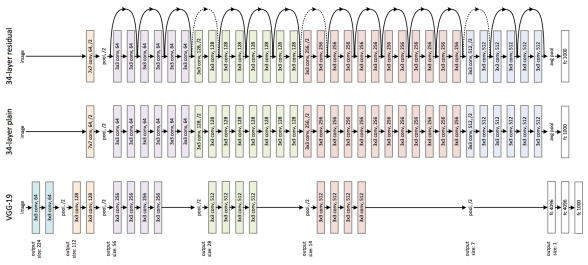


GoogLeNet [Szegedy et al., 2014]

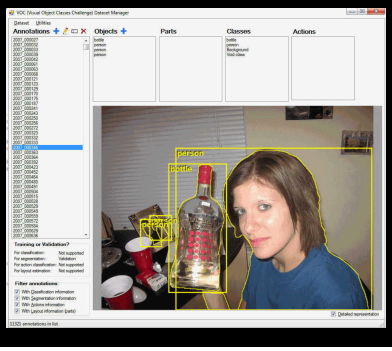


Another view of GoogLeNet's architecture.

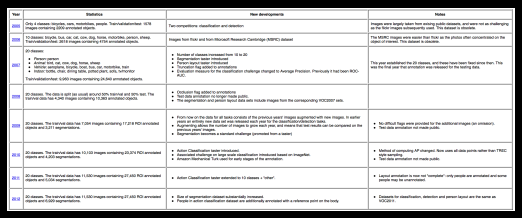
ResNet [He et al., 2016]



PASCAL VOC



PASCAL VOC



20 categories in 11,530 images with 27,450 ROIs and 6,929 segmentations

[Everingham et al. 2005—2012]

Object Detection: PASCAL VOC mean Average Precision (mAP)

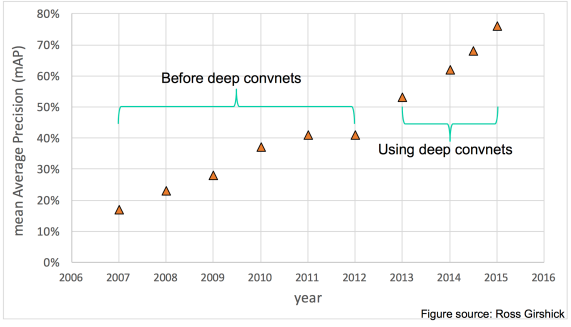


Figure source: Ross Girshick

ImageNet

[Deng et al. 2009]

20,000+ categories x ~1000 instances = 14,000,

Classification: ImageNet Challenge top-5 error

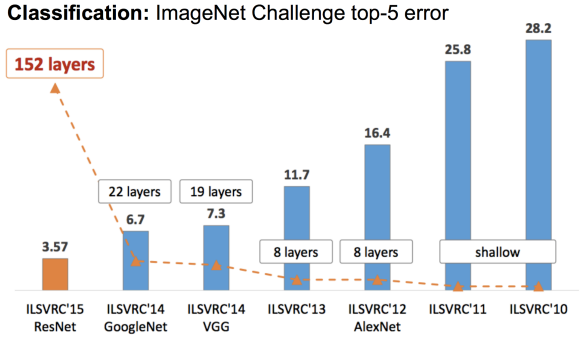


Figure source: Kaiming He