ECE 5582 Computer Vision
Lec 20: Part II Review - Subspace & Deep Neural Net Methods

Zhu Li
Dept of CSEE, UMKC
Office: FH560E, Email: lizhu@umkc.edu, Ph: x 2346.
http://l.web.umkc.edu/lizhu
Outline

- About Quiz-2
- Review Part II:
  - Linear Algebra Refresher
  - PCA and Eigenface
  - LDA and Fisherface
  - Graph Laplacian Embedding and Graph Fourier Transform
  - Sparse Signal Recovery and Dictionary Approach
  - Subspace Indexing on Grassmann Manifold
  - Direct Matrix Optimization on Stiefel and Grassmann Manifold
  - Hashing
Quiz-2/Project Signup

- **Quiz-2**
  - When? 4/22 (mon) in class
  - Close book, no cell phone.
  - Covers Part II Only
  - The same format as Quiz-1
  - 1-page hand written cheating sheet allowed/encouraged, will be graded
  - Relax 😊 only 10 pts.

- **Project Team sign up**
  - should be done now
  - for both project and paper review/presentation
Holistic Approach

• Identification
  – Given a set of training set $S$ of $n$ images $x_k$ and associated labels $y_k$
  – Design a function to predict its associated label, $f(x)$

• Why is it difficult?
  – When the number of unique labels, $m$, and training data $n$ are large....
Appearance Modeling – Finding a good $f()$

• Find a “good” $f()$
  – Such that after projecting the appearance onto the subspace, the data points belong to different classes are easily separable
  – This kind of “good” function is usually non-linear, hard to obtain from training data.
    – Linear Case: $y = AX$
    – non-linear, via kernel trick.

\[
y = f(x)
\]
Vector Products

- Inner Product

\[ x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i. \]

- Outer Product

\[ x y^T \in \mathbb{R}^{m \times n} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix}. \]
Matrix-Vector Product

- $y = Ax$

\[
y = Ax = \begin{bmatrix}
- & a_1^T & - \\
- & a_2^T & - \\
\vdots & \vdots & \vdots \\
- & a_m^T & -
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_m
\end{bmatrix}
= \begin{bmatrix}
a_1x \\
a_2x \\
\vdots \\
a_mx
\end{bmatrix}
\]

- So, $y$ is a linear combination of basis $\{a_k\}$ with weights from $x$

\[
y = Ax = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix} = \begin{bmatrix} a_1 \\
a_2 \\
\vdots \\
a_n
\end{bmatrix} x_1 + \begin{bmatrix} a_2 \\
a_2 \\
\vdots \\
a_n
\end{bmatrix} x_2 + \cdots + \begin{bmatrix} a_n \\
a_n \\
\vdots \\
a_n
\end{bmatrix} x_n
\]
Matrix Product

- \( C = AB \)

\[
C = AB = \begin{bmatrix}
- & a_1^T & - \\
- & a_2^T & - \\
\vdots & & \\
- & a_m^T & - \\
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_p \\
\end{bmatrix}
= \begin{bmatrix}
a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_p \\
a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_p \\
\vdots & \vdots & \ddots & \vdots \\
a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_p \\
\end{bmatrix}
\]

- **Associative:**
  - \( ABC = (AB)C = A(BC) \)

- **Distributive:**
  - \( A(B+C) = AB + AC \)
Matrix Transpose

**Transpose**

The *transpose* of a matrix results from “flipping” the rows and columns. Given a matrix $A \in \mathbb{R}^{m \times n}$, its transpose, written $A^T \in \mathbb{R}^{n \times m}$, is the $n \times m$ matrix whose entries are given by

$$(A^T)_{ij} = A_{ji}.$$

We have in fact already been using the transpose when describing row vectors, since the transpose of a column vector is naturally a row vector.

The following properties of transposes are easily verified:

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A + B)^T = A^T + B^T$
Matrix Trace and Determinant

- **Trace**: $\text{Tr}(A)$: only for $n \times n$ square matrix
  \[
  \text{tr}A = \sum_{i=1}^{n} A_{ii}.
  \]
  \[
  \text{tr}AB = \sum_{i=1}^{m} (AB)_{ii} = \sum_{i=1}^{m} \left( \sum_{j=1}^{n} A_{ij} B_{ji} \right)
  = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} B_{ji} = \sum_{j=1}^{n} \sum_{i=1}^{m} B_{ji} A_{ij}
  = \sum_{j=1}^{n} \left( \sum_{i=1}^{m} B_{ji} A_{ij} \right) = \sum_{j=1}^{n} (BA)_{jj} = \text{tr}BA.
  \]

- **Determinant**: $\text{Det}(A)$:
  - The size of volumes spanned by $A$,
  - All possible linear combinations of $a_1$ and $a_2$

  $A = \begin{bmatrix} 1 & 3 \\ 3 & 2 \end{bmatrix}$

  \[
  \text{Det}(A) = |2-9| = 7;
  \]
Eigen Values and Eigen Vectors

- Definition: for nxn matrix A:
  
  All non-zero vectors \( \mathbf{x} \) for which there is a \( \lambda \in \mathbb{R} \) so that
  
  \[ \mathbf{A}\mathbf{x} = \lambda \mathbf{x} \]

  are called **eigenvectors** of \( \mathbf{A} \). \( \lambda \) are the associated **eigenvalues**.

  If \( \mathbf{e} \) is an eigenvector of \( \mathbf{A} \), then also \( c \cdot \mathbf{e} \) with \( c \neq 0 \).

  Label eigenvalues \( \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \) with their eigenvectors \( \mathbf{e}_1, \ldots, \mathbf{e}_n \) (assumed to be unit vectors).

- In Matlab:
  
  \[ [\mathbf{P}, \mathbf{V}] = \text{eig}(\mathbf{A}); \]
Eigen Vectors of Symmetric Matrix

- If square matrix A: nxn is symmetric
  \[ A = A^T \]

Then its Eigen Values are real, and Eigen Vectors are orthonormal:

\[ A = USU^T \]

where S is a diagonal matrix with eigen values of A.

\[ x^T Ax = x^T U \Lambda U^T x = y^T \Lambda y = \sum_{i=1}^{n} \lambda_i y_i^2 \quad \text{where} \quad y = U^T x \]

- Application: solution to the Quadratic form maximization:

\[
\max_{x \in \mathbb{R}^n} x^T Ax \quad \text{subject to} \quad \|x\|_2^2 = 1
\]

will be the largest eigen value, and \(x^*\) will be the corresponding eigen vector of A.
for non square matrix: $A_{mxn}$:

Suppose $A \in \mathbb{R}^{m \times n}$. Then a $\lambda \geq 0$ is called a singular value of $A$, if there exist $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ such that

$$Av = \lambda u \quad \text{and} \quad A^T u = \lambda v$$

We can decompose any matrix $A \in \mathbb{R}^{m \times n}$ as

$$A = U\Sigma V^T,$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthonormal and $\Sigma$ is a diagonal matrix of the singular values.
The Singular Value Decomposition (SVD) of an nxm matrix A, is,

\[ A = U S V^T = \sum_i \sigma_i u_i v_i^T \]

- Where the diagonal of S are the eigen values of AA^T, \([\sigma_1, \sigma_2, \ldots, \sigma_n]\), called “singular values”
- U are eigenvectors of AA^T, and V are eigen vectors of A^TA, the outer product of \(u_i v_i^T\), are basis of A in reconstruction:

The 1st order SVD approx. of A is:

\[ \sigma_1 \cdot U(:, 1) \cdot V(:, 1)^T \]
SVD approximation of an image

Very easy...

function \([x] = \text{svd\_approx}(x0, k)\)

dbg = 0;

if dbg
    x0 = fix(100 * randn(4, 6));
    k = 2;
end

[u, s, v] = svd(x0);
[m, n] = size(s);
x = zeros(m, n);

sgm = diag(s);

for j = 1:k
    x = x + sgm(j) * u(:, j) * v(:, j)';
end
Given 2 subspace models $A_1, A_2$, what is their distance?

- Principal angles:

\[
\cos(\theta_k) = \max_{u_k \in \text{span}(A_1), v_k \in \text{span}(A_2)} u_k^T v_k
\]

subject to

\[
\begin{align*}
u_k^T u_k &= 1, & v_k^T v_k &= 1 \\
u_k^T u_i &= 0, & v_k^T v_i &= 0
\end{align*}
\]

- SVD to compute principal angle:

\[
[U, S, V] = \text{svd}(A_1^T A_2)
\]

\[
\theta_k = \cos^{-1}(s_{k,k})
\]
Norm

Vector Norm: Length of the vector

- Euclidean Norm (L2 Norm): \( \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \)

- \( L_p \) norm:
  \[
  \|x\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}
  \]

Matrix Norm: Forbenius Norm

\[
\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}^2} = \sqrt{\text{tr}(A^T A)}.
\]
Quadratic Form

Quadratic form $f(x) = x^T A x$ in $\mathbb{R}$:

Given a square matrix $A \in \mathbb{R}^{n \times n}$ and a vector $x \in \mathbb{R}^n$, the scalar value $x^T A x$ is called a **quadratic form**. Written explicitly, we see that

$$x^T A x = \sum_{i=1}^{n} x_i (A x)_i = \sum_{i=1}^{n} x_i \left( \sum_{j=1}^{n} A_{ij} x_j \right) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j .$$

Note that,

$$x^T A x = (x^T A x)^T = x^T A^T x = x^T \left( \frac{1}{2} A + \frac{1}{2} A^T \right) x,$$

Positive Definite (PD):
- For non-zero $x$, $x^T A x > 0$

Positive Semi-Definite (PSD):
- For non-zero $x$, $x^T A x \geq 0$

Indefinite:
- Exists $x_1, x_2$ non zero, but $x_1^T A x_1 > 0$, while $x_2^T A x_2 < 0$;
Matrix Calculus

- Gradient of $f(A)$: $f : \mathbb{R}^{m \times n} \to \mathbb{R}$

$$\nabla_A f(A) \in \mathbb{R}^{m \times n} = \begin{bmatrix}
\frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\
\frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}}
\end{bmatrix}$$

- Matrix Gradient Properties

  - $\nabla_x (f(x) + g(x)) = \nabla_x f(x) + \nabla_x g(x)$.
  - For $t \in \mathbb{R}$, $\nabla_x (t \cdot f(x)) = t \nabla_x f(x)$. 
Hessian of $f(X)$

- For function: $f: R^n \rightarrow R$

\[ \nabla^2_x f(x) \in \mathbb{R}^{n \times n} = \begin{bmatrix}
\frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \ldots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\
\frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_2} & \ldots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \ldots & \frac{\partial^2 f(x)}{\partial x_n \partial x_n}
\end{bmatrix} \]

- Gradient & Hessian of Quadratic Form: $f(x) = x^T A x$

  - $\nabla_x b^T x = b$
  - $\nabla_x x^T A x = 2A x$ (if $A$ symmetric)
  - $\nabla^2_x x^T A x = 2A$ (if $A$ symmetric)
Data points scatter/covariance

- Scatter of the all the data, aka, total scatter/covariance
  \[ S_T = E\{(X - \bar{X})(X - \bar{X})^T\} = \sum_i (x_i - \bar{x})(x_i - \bar{x})^T \]

- Intra-class scatter: assuming we have \( k=1..m \) class, each with mean \( \mu_k \)
  \[ S_W = E_{x \in \text{class } k}\{(X - \mu_k)(X - \mu_k)^T\} = \sum_k \sum_{x_i \in \text{class } k} (x_i - \mu_k)(x_i - \mu_k)^T \]

- Inter class scatter: class mean’s scatter
  \[ S_B = n_k \sum_k (\mu_k - \bar{X})(\mu_k - \bar{X})^T \]
PCA – Maximizing Total Scatter Interpretation

- PCA is to find a projection that maximizes total scatter

$$A = \arg \max_W W^TX^TXW, \text{ s.t. } W^TW = I$$

$$A = \arg \max_W W^TS_TW, \text{ s.t. } W^TW = I$$

- Solution:
  - By Lagrangian relaxation & KKT Condition
    $$S_TW = \lambda W$$

- Computational:
  - Compute the covariance of data (mean removed), $S_T$,
  - Compute the Eigen Vectors of $S_T$, that would be the Principal Component of $X$.
  - Matlab: $[A, s, \text{ Eigv}]=\text{princomp}(X)$;
The PCA dimension reduction of SIFT data

PCA on face data/ Eigenface
LDA and Fisherface

- **PCA (Eigenfaces)**
  \[ W_{PCA} = \arg \max_W |W^T S_T W| \]
  Maximizes projected total scatter

- **Fisher’s Linear Discriminant**
  \[ W_{fld} = \arg \max_W \frac{|W^T S_B W|}{|W^T S_W W|} \]
  Maximizes ratio of projected between-class to projected within-class scatter, solved by the generalized Eigen problem:
  \[ S_B W = \lambda S_W W \]
Matlab:

% compute class mean
mx = mean(x);
ids = unique(y); m = length(ids); Sb = zeros(kd, kd);
for k=1:m
    indx = find(y==ids(k)); nk(k) = length(indx);
    % class mean
    m_cx(k,:) = mean(x(indx, :));
    % between class scatter
    Sb = Sb + nk(k)*(m_cx(k,:) - mx)'*(m_cx(k,:) - mx);
end

% compute intra-class scatter
Sw = zeros(kd, kd);
for k=1:m
    indx = find(y==ids(k)); nk(k) = length(indx);
    % remove mean
    xk = x(indx, :) - repmat(m_cx(k,:), [nk(k), 1]);
    % adding up
    Sw = Sw + (xk'*xk);
end

% solve by generalized eigen problem
[A, v]=eigs(Sb, Sw);
PCA vs LDA performance

- myLDA implementation: with pca kd=32
Find a projection that best preserves the graph affinity matrix

\[
\min_W \sum_i \sum_j S_{i,j} (y_i - y_j)^2, \quad s.t. \ y = Wx
\]

\[
\min_W \sum_i \sum_j S_{i,j} (Wx_i - Wx_j)^2
\]

\[
s.t. S_{i,j} = \begin{cases} 
-\exp\left(\frac{|x_j - x_i|^2}{h}\right), & \text{if } |x_j - x_i| \leq \theta \\
0, & \text{else}
\end{cases}
\]
Graph Laplacian Embedding Solution

- **Formulation**

\[
\begin{align*}
\text{arg min} & \quad w^T X L X^T w \\
\text{s.t.} & \quad w^T X D X^T w = 1
\end{align*}
\]

- Graph Laplacian: \( L = D - S \):
- Diagonal matrix, entry \( D_{ii} \) = sum of col/row affinity
- The larger the value, the more important data point is

- **Lagrangian Relaxation:**

\[
L(w, \lambda) = w^T X L X^T w - \lambda (w^T X D X w - 1)
\]

- **Generalized Eigen**

\[
X L X^T w = \lambda X D X^T w
\]
Connection to PCA, LDA

- Affinity graph $S$ determines the embedding subspace $W$ via
  \[ XLX^T w = \lambda XDX^T w \]

- PCA and $s_{i,j} = 1/n$
  - PCA:
    \[ S_{i,j} = \begin{cases} 
    \frac{1}{n_k}, & \text{if } x_i, x_j \in C_k \\
    0, & \text{else}
    \end{cases} \]
  - LDA
  - LPP
    \[ S_{i,j} = \begin{cases} 
    -\exp\left(\frac{|x_j - x_i|^2}{h}\right), & \text{if } |x_j - x_i| \leq \theta \\
    0, & \text{else}
    \end{cases} \]
Basis Functions

- **Eigenface vs Fisherface vs Laplacianface**

### Eigenface

- `eigface = eye(400)*A1(:,1:kd);`

### Fisherface

- `lapface = eye(400)*A1(:,1:kd)*A2;`

### Laplacian Face

- `for k=1:8`
  - `figure(36); subplot(2,4,k); imagesc(reshape(eigface(:,k),[20, 20])); colormap('gray'); title(sprintf('eigf_%d', k));`
  - `figure(37); subplot(2,4,k); imagesc(reshape(lapface(:,k),[20, 20])); colormap('gray'); title(sprintf('lapf_%d', k));`
- `end`
Laplacian vs Eigenface

- 1200 faces, 144 subjects
Limit of Subspace Approach

- Project face images to certain face model A space
  - Matlab: $x = \text{faces} \cdot A(:,1:kd)$

- DoF!
  - Limited to $wxh$. 

$$eigf_1 = 10.9^* + 0.4^* + 4.7^*$$
Subspace Indexing on Grassmann Manifold

- Data partition via kd-tree
  - Local data balancing off the DoF of a given model, vs that of number of non-zero edges in affinity graph

- Build local subspace models, and VQ on Grassmann to get a model hierarchical tree

Subspace DoF

- $G(p, d)$ identifies $p$-dimensional subspaces in $d$-dimensional space, it consists of othonormal bases in $d$-dimensional space (Stiefel Manifold) and an equivalence constraint on the rotation of the basis functions:

$$A_1 = A_2, \text{ if } \text{span}(A_1) = \text{span}(A_2)$$

- $G(p, d)$ is the quotient space of $S(p, d)/O(p)$, i.e., $A_1 = A_2$, if exists $R \in R^{p \times p}$, s.t., $RA_1 = A_2$, $R \in O_p$

- The DoF of subspaces on Grassmann manifold

$$\text{DoF}(G(p, d)) = pd - p^2$$
Subspace Distance Metric

- **Distance between 2 subspaces**
  - **Principal angles**
    
    \[
    \cos(\theta_k) = \max_{u_k \in \text{span}(A_1), v_k \in \text{span}(A_2)} u_k^T v_k
    \]
    
    s.t. \[
    \begin{align*}
    u_k^T u_k &= 1, v_k^T v_k = 1 \\
    u_k^T u_i &= 0, v_k^T v_i = 0
    \end{align*}
    \]

  - **Compute Principle angles**
    - \([u, s, v] = \text{svd}(A_1^T A_2)\)
    - The arc \(\cos(\text{diag}(s))\), would be the angles

  - **Distance**
    - The distance, \(d(A_1, A_2) = \sqrt{\sum_k \theta_k^2}\)
Grassmann Distance Metrics:

- **Projection Distance**
  
  **Def:**
  \[ d_{prj}(A_1, A_2) = \left( \sum_{i=1}^{p} \sin^2 \theta_i \right)^{1/2} \]

  **Computing:**
  \[ d^2_{prj}(A_1, A_2) = p - \sum_{i=1}^{p} \cos^2 \theta_i = m - \|A'_1 A_2\|_F^2 \]

- **Binet-Cauchy Distance**
  
  **Def:**
  \[ d_{bc}(A_1, A_2) = \left( 1 - \prod_i \cos^2 \theta_i \right)^{1/2} \]

  **Computing:**
  \[ d^2_{bc}(A_1, A_2) = 1 - \prod_i \cos^2 \theta_i = 1 - \det^2(A'_1 A_2) \]
2.5.3. The Gradient of a Function (Grassmann). We must compute the gradient of a function $F(Y)$ defined on the Grassmann manifold. Similarly to §2.4.4, the gradient of $F$ at $[Y]$ is defined to be the tangent vector $\nabla F$ such that

\[
\text{tr } F_Y^T \Delta = g_c(\nabla F, \Delta) \equiv \text{tr}(\nabla F)^T \Delta
\]

for all tangent vectors $\Delta$ at $Y$, where $F_Y$ is defined by Eq. (2.52). Solving Eq. (2.69) for $\nabla F$ such that $Y^T(\nabla F') = 0$ yields

\[
\nabla F = F_Y - YY^T F_Y.
\]
Hessian on Grassmann Manifold

\[ \text{Hessian:} \]

2.5.4. The Hessian of a Function (Grassmann). Applying the definition for the Hessian of \( F(Y) \) given by Eq. (2.54) in the context of the Grassmann manifold yields the formula

\[
\operatorname{Hess} F(\Delta_1, \Delta_2) = F_{YY}(\Delta_1, \Delta_2) - \text{tr}(\Delta_1^T \Delta_2 Y^T F_Y),
\]

where \( F_Y \) and \( F_{YY} \) are defined in §2.4.5. For Newton’s method, we must determine \( \Delta = - \operatorname{Hess}^{-1} G \) satisfying Eq. (2.58), which for the Grassmann manifold is expressed as the linear problem

\[
F_{YY}(\Delta) - \Delta(Y^T F_Y) = -G,
\]

\( Y^T \Delta = 0 \), where \( F_{YY}(\Delta) \) denotes the unique tangent vector satisfying Eq. (2.60) for the Grassmann manifold’s canonical metric.

- \( F_Y = nxp \) 1st order differentiation
- \( F_{YY} = 2nd \) order differentiation along \( Y \)
Newton’s Method on Grassmann Manifold

Overall framework

Newton’s Method for Minimizing \( F(Y) \) on the Grassmann Manifold

- Given \( Y \) such that \( Y^T Y = I_p \),
  - Compute \( G = F_Y - YY^T F_Y \).
  - Compute \( \Delta = -\text{Hess}^{-1} G \) such that \( Y^T \Delta = 0 \) and
    \[
    F_{YY}(\Delta) - \Delta(Y^T F_Y) = -G.
    \]
- Move from \( Y \) in direction \( \Delta \) to \( Y(1) \) using the geodesic formula
  \[
  Y(t) = YV \cos(\Sigma t)V^T + U \sin(\Sigma t)V^T
  \]
  where \( U\Sigma V^T \) is the compact singular value decomposition of \( \Delta \) (meaning \( U \) is \( n \)-by-\( p \) and both \( \Sigma \) and \( V \) are \( p \)-by-\( p \)).
- Repeat.
Hashing

- **Locality Sensitive Hash**
  - Random projections that generate hash bits
  - Sufficient number of projections will preserve its distance in hamming distance, as $d(p,q)$ nearness is always preserved in projection.
  - Not very efficient though (see Complementary Hashing)

- **Grassmann Hash**
  - Allow flexible multiple dimension projection and bucket design
  - Penalizing the projections with Grassmann metric
Deeper is wider: final conv feature go thru linear FC

- Deeper feature, has larger receptive field, i.e., how many pixels it derives from
CNN key processing elements

- **Conv filters**
  - input features $h \times w \times k$, 16 5x5 filters, what would be the output feature map size?

- **Padding**

- **Pooling & Stride**
  - stride 2, 3x3 max pooling, what is the output feature map size?

- **ReLu**
  - Only non-linear part

- **Batch Normalization**
  - normalize feature map to $[0, 1]$, e.g.
Loss Function

- **Classification:** Softmax cross entropy loss

\[ y_k = \zeta(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{C} e^{z_j}} \quad \text{for } k = 1 \cdots C \]

\[ \xi(T, Y) = \sum_{i=1}^{n} \xi(t_i, y_i) = -\sum_{i=1}^{n} \sum_{i=c}^{C} t_{ic} \cdot \log(y_{ic}) \]

- **Identification:** triplet loss

\[ \sum_{i}^{N} \left[ \| f(x_i^a) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha \right]_+ \]
Gradient Back Propagation

Vector case

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k) (1_{k=i}x_j)$$

$$= 2q_i x_j$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)^2_i$

$$\nabla_W f = 2q \cdot x^T$$

$$q = W \cdot x = \left( \begin{array}{c} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{array} \right)$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\nabla_w f_{i,j} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k) (1_{k=i}x_j)$$

$$= 2q_i x_j$$
Part II: Holistic Approach in Image Analysis

- **Math Basis:**
  - Linear Algebra, Prob & Stats, Numerical Optimization
- **PCA:** un-supervised
- **LDA:** supervised
- **LPP:** Graph Embedding formulation, unifying theory
- **Subspace Indexing on Grassmann manifold**
- **Direct matrix optimization on Grassmann/Stiefel manifold**
- **Hashing**
- **Deep Learning**