Dimensionality Reduction PCA

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

February 18, 2014

©Carlos Guestrin 2005-2014

Dimensionality reduction

- Input data may have thousands or millions of dimensions!
  - e.g., text data has

X

10,000 - 100 000 000 dims

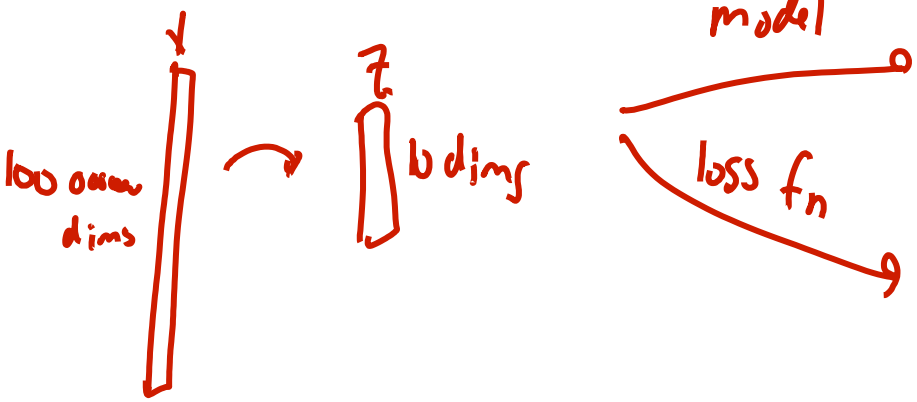
- **Dimensionality reduction**: represent data with fewer dimensions
  - easier learning – fewer parameters
  - visualization – hard to visualize more than 3D or 4D
  - discover "intrinsic dimensionality" of data
    - high dimensional data that is truly lower dimensional

Lower dimensional projections

- Rather than picking a subset of the features, we can new features that are combinations of existing features

$\rightarrow Z_7 = 2.5X_1 + 3.2X_2 - 4.7X_3 \ldots$

$Z = A\,X$

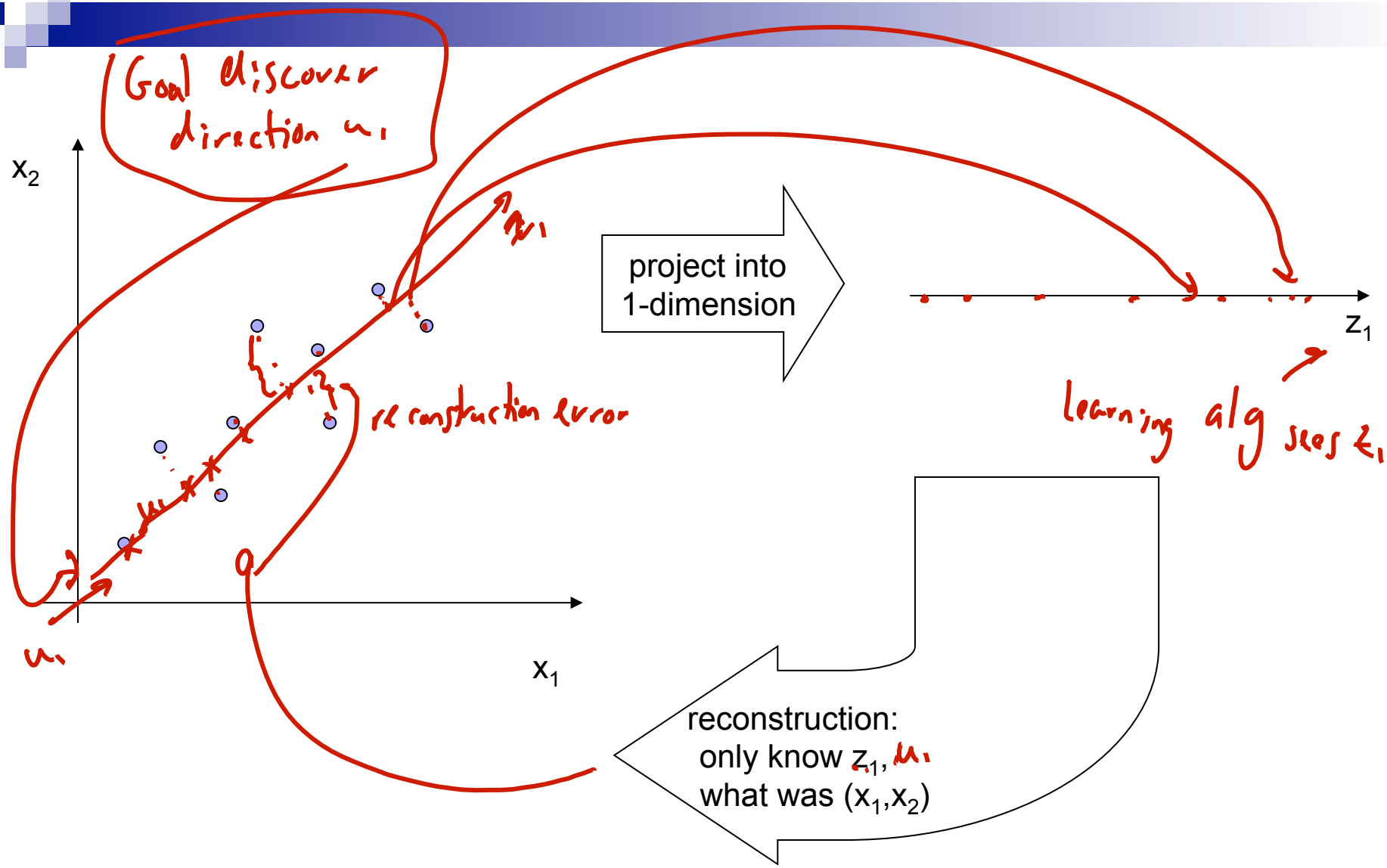


model → linear

loss fn → squared reconstruction error

- Let's see this in the unsupervised setting
  - just **X**, but no Y

Linear projection and reconstruction



Principal component analysis – basic idea

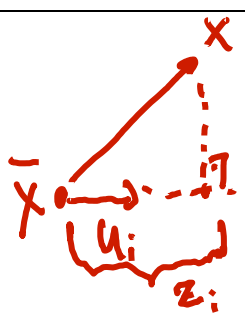
- Project n-dimensional data into k-dimensional space while preserving information:
  - e.g., project space of 10000 words into 3-dimensions
  - e.g., project 3-d into 2-d

- Choose projection with minimum reconstruction error

Linear projections, a review

$\|u_i\| = 1$

- Project a point into a (lower dimensional) space:
  - **point**: $\mathbf{x} = (x_1,\ldots,x_d)$ 1000 000
  - **select a basis** – set of basis vectors – $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$ 100
    - we consider orthonormal basis:
      - $\mathbf{u}_i \bullet \mathbf{u}_i = 1$, and $\mathbf{u}_i \bullet \mathbf{u}_j = 0$ for $i \neq j$
  - **select a center** – $\bar{\mathbf{x}}$, defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: $(z_1,\ldots,z_k)$, $z_i = (\mathbf{x}-\bar{\mathbf{x}}) \bullet \mathbf{u}_i$
    - minimum squared error



$$z_i = (x-\bar{x})\cdot u_i \iff z_i = \arg\min_z \underbrace{\left[\overset{\text{truth}}{(x-\bar{x})} - \overset{\text{approx}}{z\,u_i}\right]^2}_{\text{squared reconstruction error}}$$

PCA finds projection that minimizes reconstruction error

- Given N data points: $\mathbf{x}^i = (x_1^i,\ldots,x_d^i)$, i=1…N
- Will represent each point as a projection:
  - $\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$ where: $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^i$ and $z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$

k dims

desired directions

projection to each basis

approx

offset

coefficients of each basis for point i

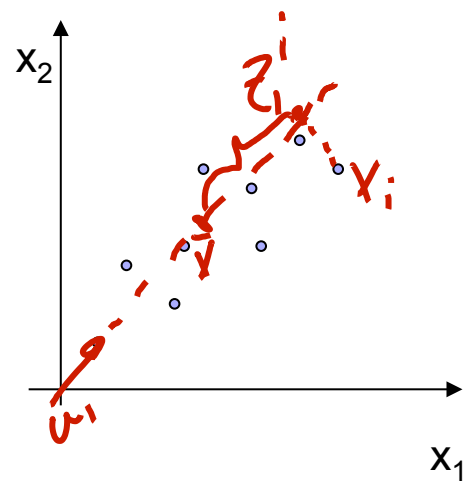
mean of data

- PCA:
  - Given k<<d, find $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$ minimizing reconstruction error:

$$error_k = \sum_{i=1}^{N} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

true point

approx



Understanding the reconstruction error

- Note that $\mathbf{x}^i$ can be represented exactly by d-dimensional projection:

$$\mathbf{x}^i = \bar{\mathbf{x}} + \sum_{j=1}^{d} z_j^i \mathbf{u}_j$$

cut short

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

$$z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$$

- Given k<<d, find ($\mathbf{u}_1,\ldots,\mathbf{u}_k$) minimizing reconstruction error:

$$error_k = \sum_{i=1}^{N} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

$u_1 \ldots u_d$ orthonormal

$$error_k = \sum_{i=1}^{N} (x^i - \hat{x}^i)^2 = \sum_{i=1}^{N} \left(\bar{x} + \sum_{j=1}^{d} z_j^i u_j - \left(\bar{x} + \sum_{j=1}^{k} z_j^i u_j \right) \right)^2$$

- Rewriting error:

$$= \sum_{i=1}^{N} \left(\sum_{j=k+1}^{d} z_j^i u_j \right)^2 = \sum_{i=1}^{N} \left[\sum_{j=k+1}^{d} z_j^i \cancelto{1}{u_j \cdot u_j} z_j^i + \sum_{j \neq l} z_j^i \cancelto{0}{u_j \cdot u_l} z_l^i \right]$$

$$= \sum_{i=1}^{N} \sum_{j=k+1}^{d} (z_j^i)^2 \leftarrow$$

minimizing reconstruction error $\equiv$ throwing dimensions with smallest z

Sum over z for ignored dims

Reconstruction error and covariance matrix

$\Sigma u = \lambda u$ ← eigen vectors

eigen values

want to min WRT $u_1 \dots u_d$

$$error_k = \sum_{i=1}^{N} \sum_{j=k+1}^{d} [\overbrace{\mathbf{u}_j \cdot (\mathbf{x}^i - \bar{\mathbf{x}})}^{z_j^i}]^2$$

$$= \sum_{i=1}^{N} \sum_{j=k+1}^{d} u_j^T (x^i - \bar{x})(x^i - \bar{x})^T u_j$$

$$= \sum_{j=k+1}^{d} u_j^T \underbrace{\left[\sum_{i=1}^{N} (x^i - \bar{x})(x^i - \bar{x})^T \right]}_{N\Sigma} u_j$$

want to min is
$u_1 \dots u_d$

$$error_k = N \sum_{j=k+1}^{d} u_j^T \Sigma u_j$$

$$\Sigma = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}^i - \bar{\mathbf{x}})(\mathbf{x}^i - \bar{\mathbf{x}})^T$$

$$\Sigma = \begin{pmatrix} \sigma_1^2 & & \sigma_{ij} \\ & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_d^2 \end{pmatrix}$$

$$\sigma_{uv} \overset{MLE}{=} \frac{1}{N} \sum_{i=1}^{N} (x_u^i - \bar{x}_u)(x_v^i - \bar{x}_v)$$

in vector form:

$$\Sigma_{MLE} = \frac{1}{N} \sum_{i=1}^{N} (x^i - \bar{x})(x^i - \bar{x})^T$$

Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis ($\mathbf{u}_1,\ldots,\mathbf{u}_d$) minimizing:

$$error_k = N \sum_{j=k+1}^{d} \mathbf{u}_j^T \Sigma \mathbf{u}_j$$

- Eigen vector:

$$\Sigma u = \lambda u$$

matrix, eigen value, eigen vector $\|u\|=1$

$$u^T \Sigma u = \lambda u^T u = \lambda$$

- Minimizing reconstruction error equivalent to picking ($\mathbf{u}_{k+1}$, …,$\mathbf{u}_d$) to be eigen vectors with smallest eigen values

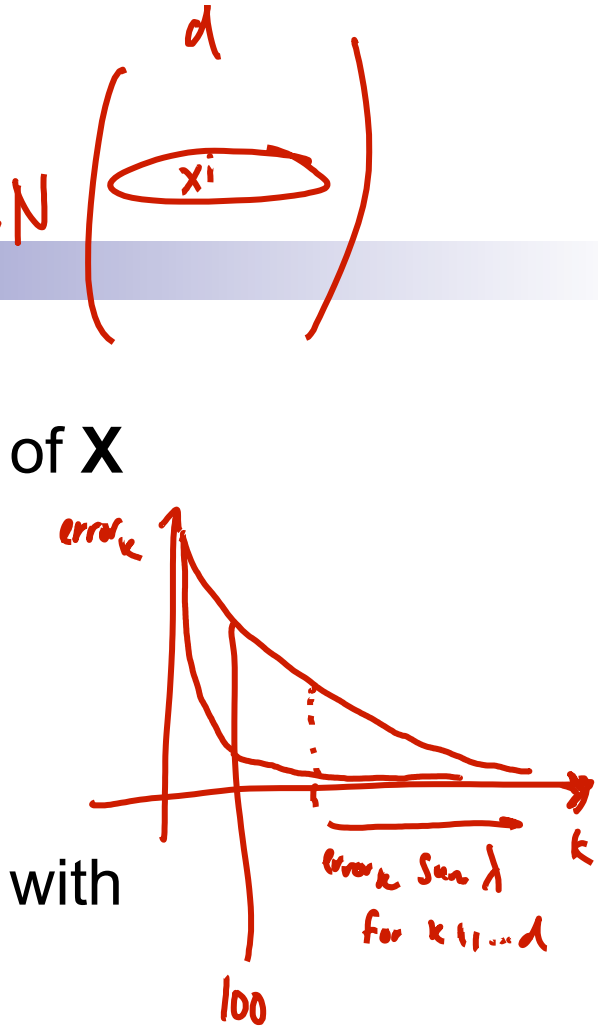
ignored directions

min $\underset{u_1 \ldots u_k}{\text{error}_k}$ $\equiv$ throw out $u_{k+1} \ldots u_d$ as eigen vectors with smallest eigen values

$\Rightarrow$ Keep eigen vectors with largest eigen values

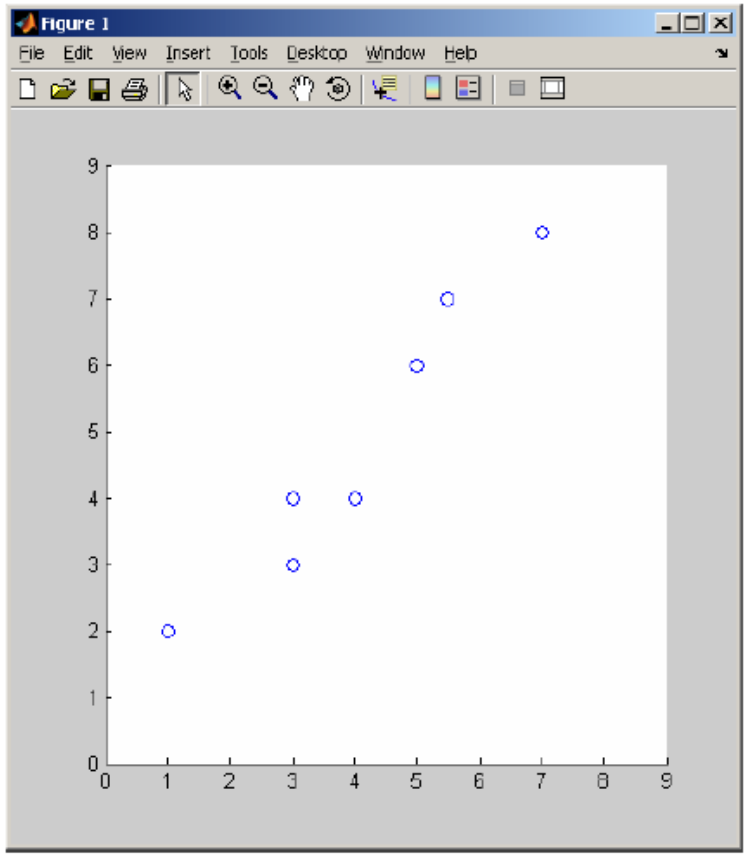
Basic PCA algoritm

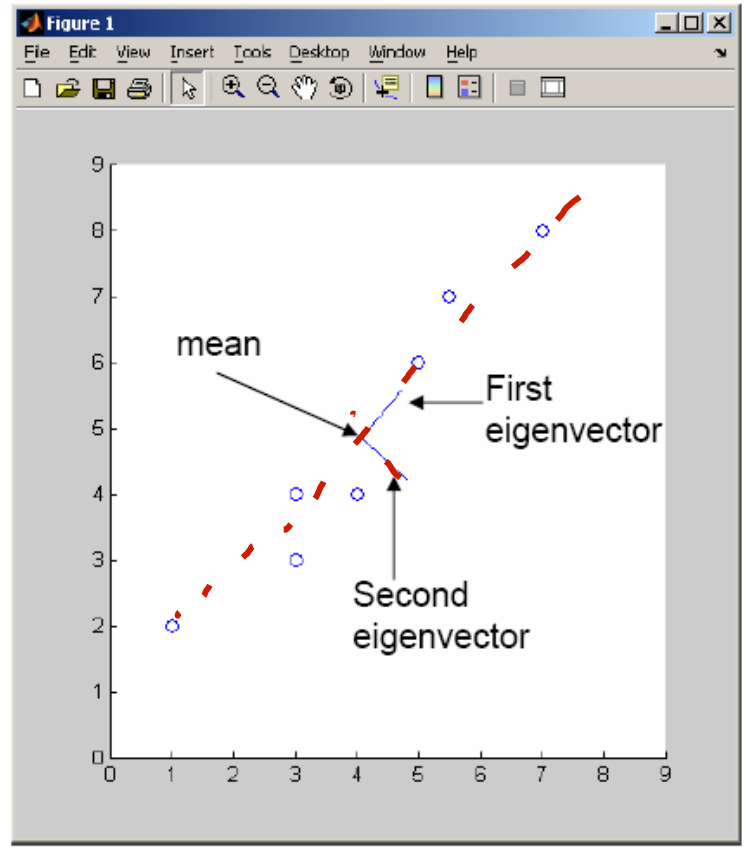
- Start from ~~m~~ N by ~~n~~ D data matrix **X**
- **Recenter**: subtract mean from each row of **X**
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$ ← $x_c = (x^i - \bar{x})$
- **Compute covariance matrix**:
  - $\Sigma \leftarrow 1/N\ \mathbf{X_c}^T \mathbf{X_c}$
- Find **eigen vectors and values** of Σ
- **Principal components:** k eigen vectors with highest eigen values



PCA example

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

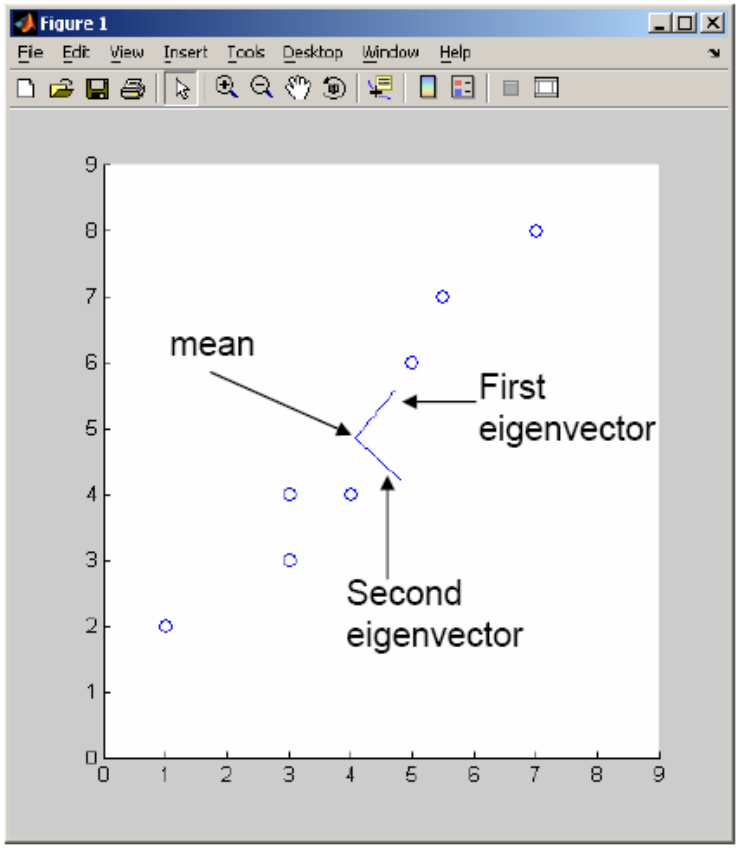


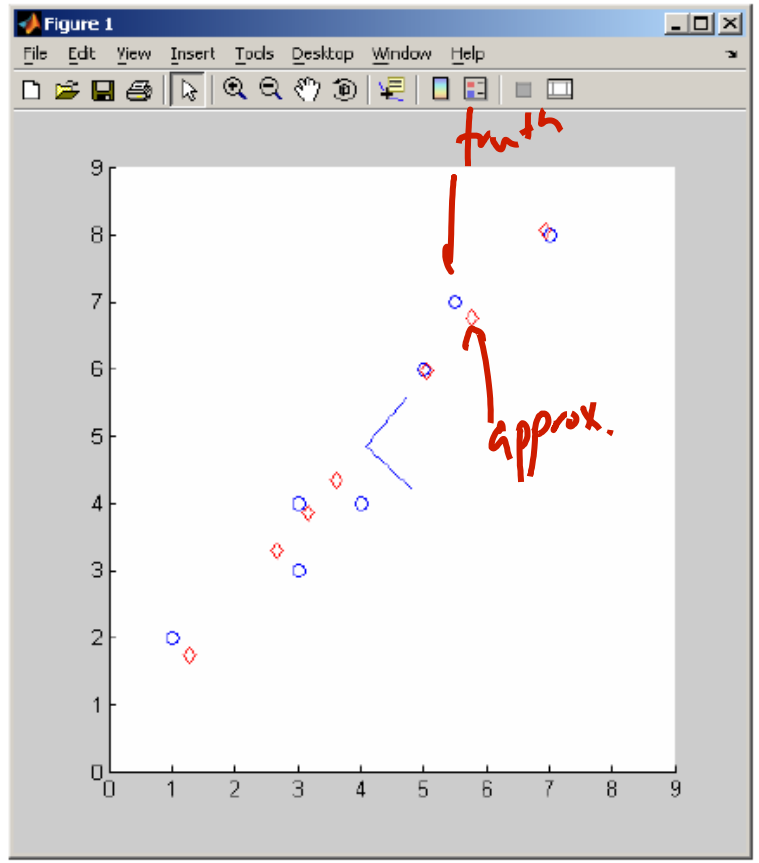


PCA example – reconstruction

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

only used first principal component





Eigenfaces [Turk, Pentland '91]

- Input images:

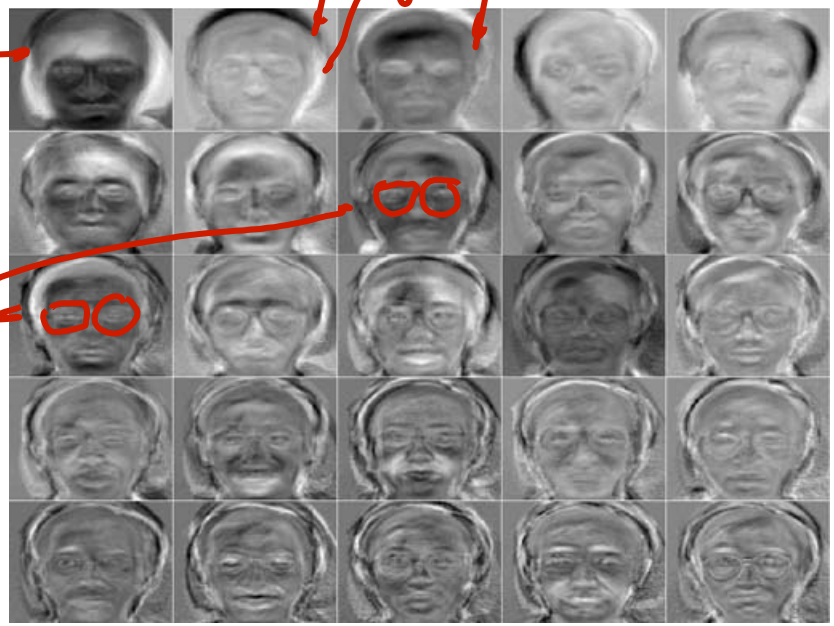
image pixels stacked into vector

- Principal components:

$\bar{x}$

u_1 u_2

glasses



Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:

k=64

truth

Scaling up

- Covariance matrix can be really big!
  - Σ is d by d
  - Say, only 10000 features
  - finding eigenvectors is very slow…

10 000 Σ 10 000

- Use singular value decomposition (SVD) ← directly on X matrix
  - finds top k eigenvectors
  - great implementations available, e.g., GraphLab, python, R, Matlab svd

SVD

- Write $\mathbf{X} = \mathbf{W}\,\mathbf{S}\,\mathbf{V}^T$
  - $\mathbf{X} \leftarrow$ data matrix, one row per datapoint
  - $\mathbf{W} \leftarrow$ weight matrix, one row per datapoint – coordinate of $\mathbf{x}^i$ in eigenspace
  - $\mathbf{S} \leftarrow$ singular value matrix, diagonal matrix
    - in our setting each entry is eigenvalue λ_j
  - $\mathbf{V}^T \leftarrow$ singular vector matrix
    - in our setting each row is eigenvector $\mathbf{v}_j$

©Carlos Guestrin 2005-2014

PCA using SVD algoritm

- Start from m by n data matrix **X**
- **Recenter**: subtract mean from each row of **X**
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- Call SVD algorithm on $\mathbf{X_c}$ – ask for k singular vectors
- **Principal components:** k singular vectors with highest singular values (rows of $\mathbf{V}^T$)
  - **Coefficients** become:

What you need to know

- Dimensionality reduction
  - why and when it's important
- Simple feature selection
- Principal component analysis
  - minimizing reconstruction error
  - relationship to covariance matrix and eigenvectors
  - using SVD

Neural Networks

Machine Learning – CSEP546

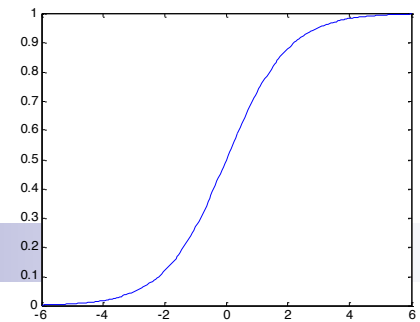
Carlos Guestrin

University of Washington

March 3, 2014

©Carlos Guestrin 2005-2014

Logistic regression



- P(Y|X) represented by:

$$P(Y=1 \mid x, W) = \frac{1}{1+e^{-(w_0+\sum_i w_i x_i)}} = g(w_0 + \sum_i w_i x_i)$$

- Learning rule – MLE:

$$\frac{\partial \ell(W)}{\partial w_i} = \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)]$$

$$= \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

step size

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_{j=1}^{N} x_i^j \delta^j}$$

truth

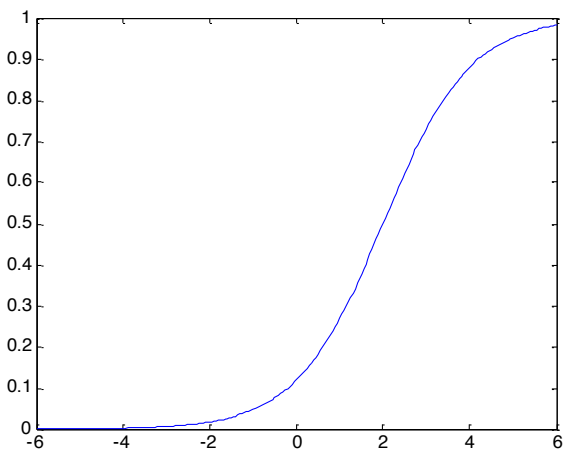
prediction

$$\delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)$$

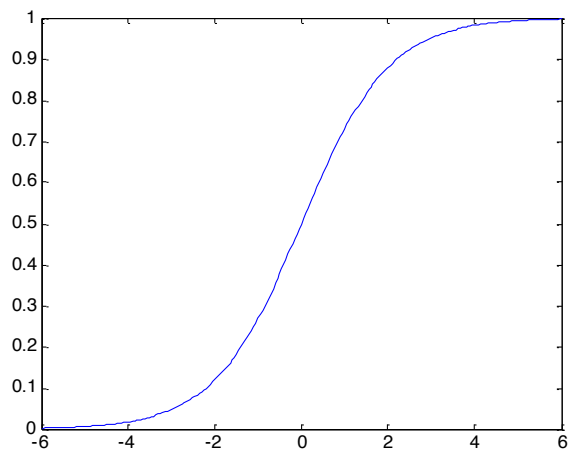
Sigmoid

$$g(w_0 + \sum_i w_i x_i) \quad = \quad \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

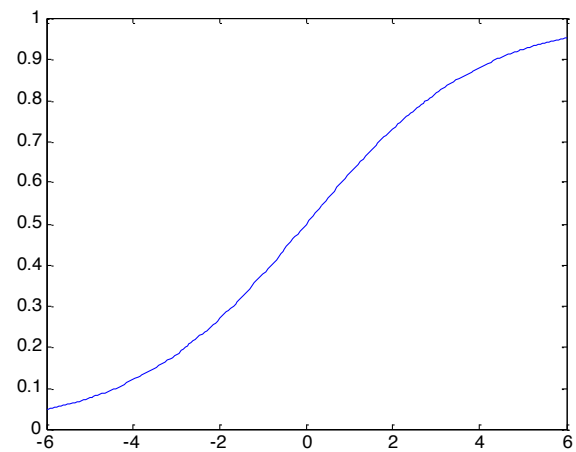
w_0=2, w_1=1



w_0=0, w_1=1



w_0=0, w_1=0.5



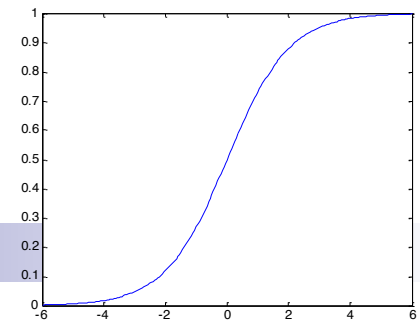
©Carlos Guestrin 2005-2014

Perceptron more generally

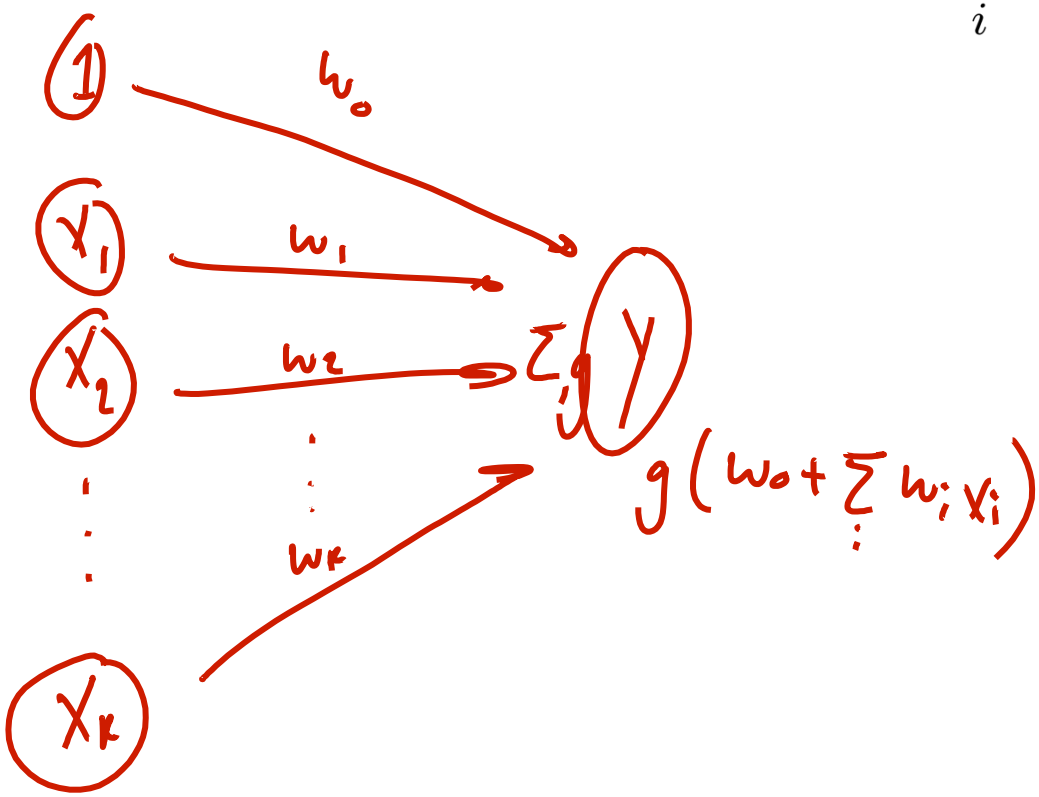
σ - sigmoid

- Linear function: $g(w_0 + \sum_i w_i x_i) = w_0 + \sum_i w_i x_i$
- Link function: $g(\cdot)$
- Output: $[0, 1]$
- Loss function: logistic loss
- Train via SGD:

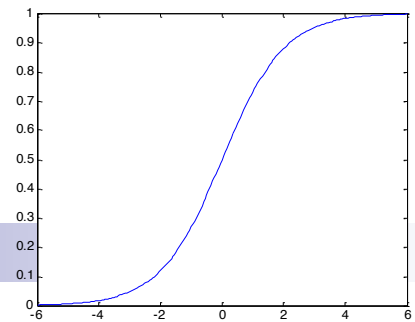
Perceptron as a graph



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

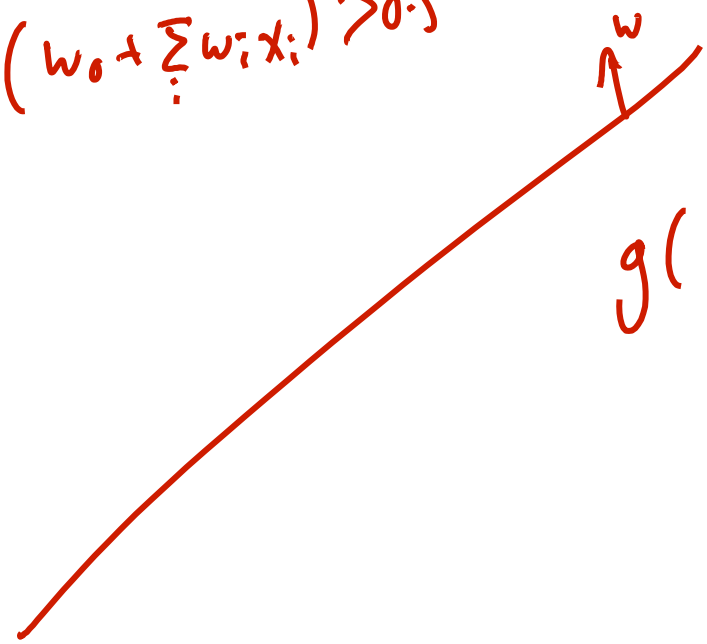


Linear perceptron classification region



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

$g(w_0 + \sum_i w_i x_i) > 0.5$



$g(w_0 + \sum_i w_i x_i) < 0.5$

Optimizing the perceptron

- Trained to minimize sum-squared error

$$\ell(W) = \frac{1}{2}\sum_j [y^j - \underbrace{g(w_0 + \sum_i w_i x_i^j)}_{\text{prediction}}]^2 \leftarrow \text{squared}$$

target

$$\frac{\partial \ell(w)}{\partial w_i} \rightarrow \text{grad descent or SGD}$$

The perceptron learning rule with logistic link and squared loss

step size feature deviation

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j \delta^j}$$

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)] g^j (1 - g^j)$$

$$g^j = g(w_0 + \sum_i w_i x_i^j)$$

Slightly diff δ^j because of squared loss

- Compare to MLE:

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j}$$

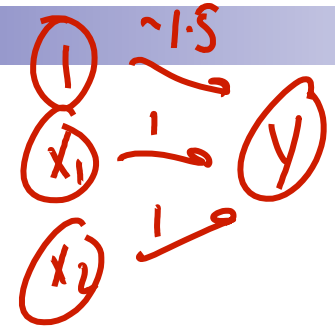
$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

Percepton, linear classification, Boolean functions

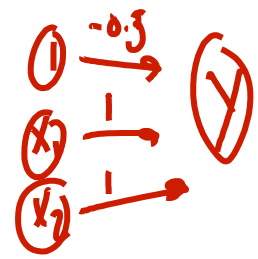
$x_1 \in \{0,1\}$ $x_2 \in \{0,1\}$

- Can learn x_1 AND x_2

output > 0 when x_1 AND x_2
< 0 otherwise

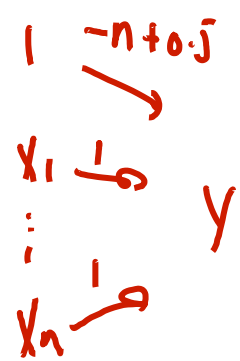


- Can learn x_1 OR x_2



- Can learn any conjunction or disjunction

x_1 AND x_2 AND x_3 ..... AND x_n



Percepton, linear classification, Boolean functions

- Can learn majority Yes

- Can perceptrons do everything? No: XOR

x_1 XOR x_2

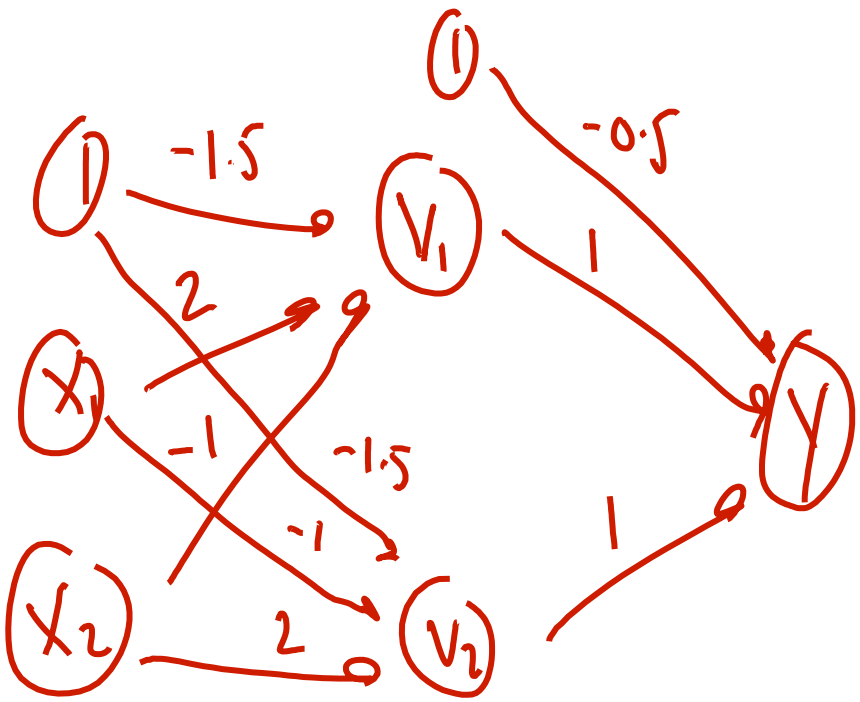
+ −

− +

Going beyond linear classification

- Solving the XOR problem

$$Y = X_1 \text{ XOR } X_2 = \underbrace{(X_1 \wedge \neg X_2)}_{V_1} \text{ OR } \underbrace{(\neg X_1 \wedge X_2)}_{V_2}$$

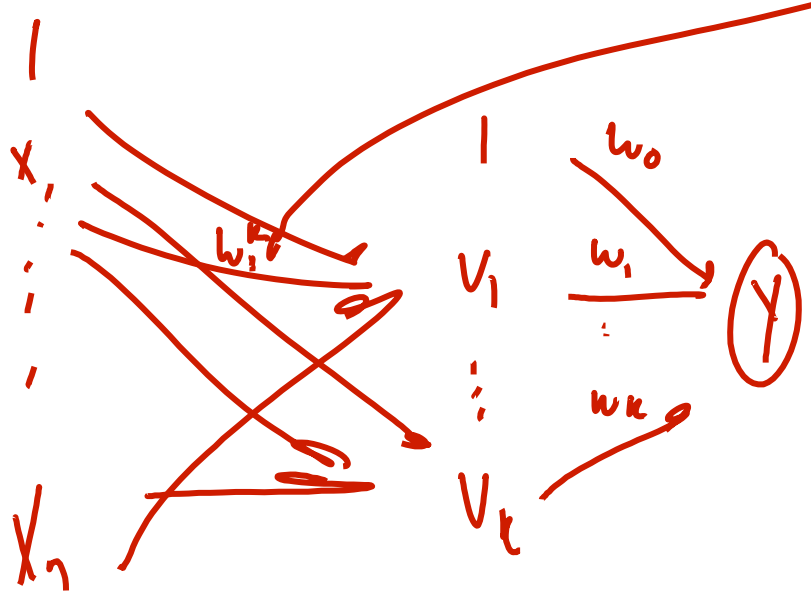


chained together perceptrons to generate non-linear classifier

Hidden layer

- Perceptron: $$out(\mathbf{x}) = g(w_0 + \sum_i w_i x_i)$$
- 1-hidden layer:

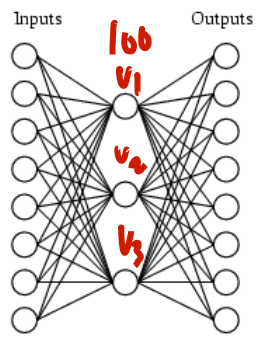
$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k \overbrace{g(w_0^k + \sum_i w_i^k x_i)}^{V_k}\right)$$



Example data for NN with hidden layer

100000 100000 ← using NN for dim reduction

X



X

Auto encoder

A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

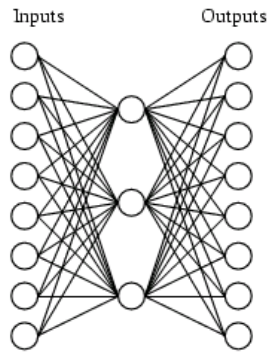
8 dim (→) but only need 3 bits/3 dim

Can this be learned??

©Carlos Guestrin 2005-2014

Learned weights for hidden layer

A network:

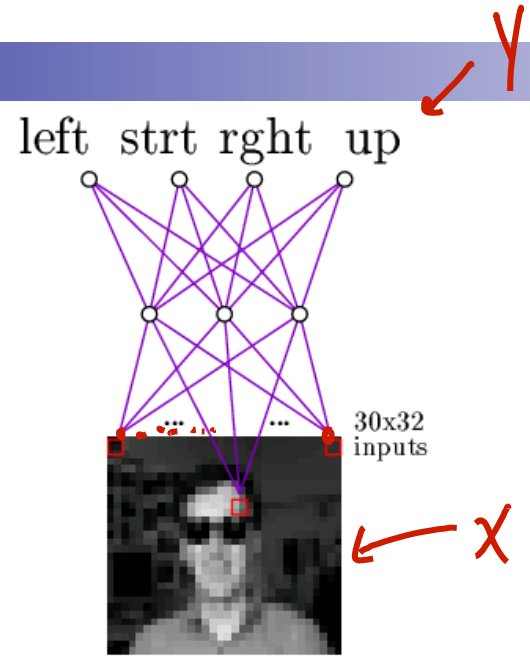


Binary encoding

Learned hidden layer representation:

Input		Hidden Values				Output
		v_1	v_2	v_3		
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

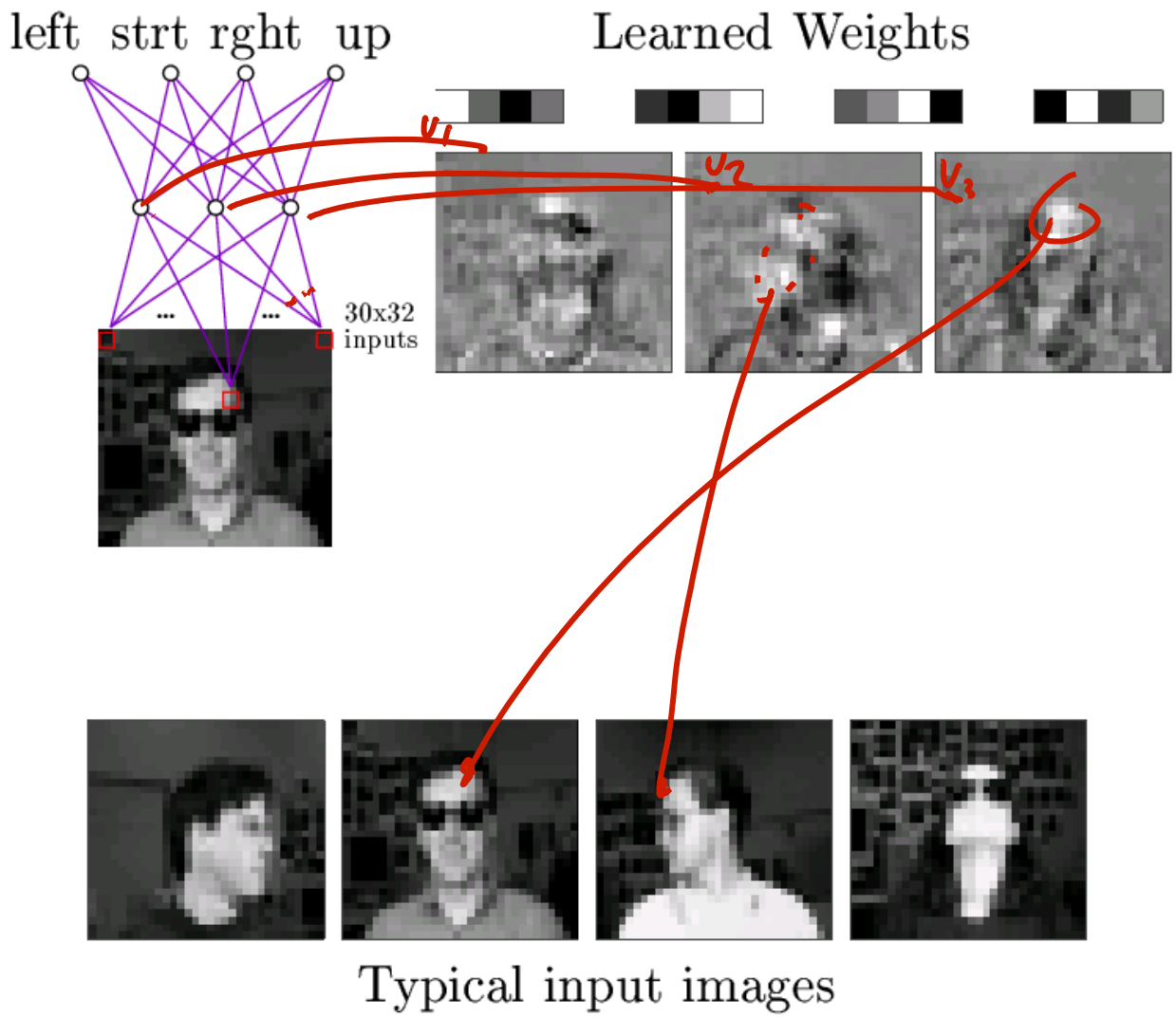
NN for images



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

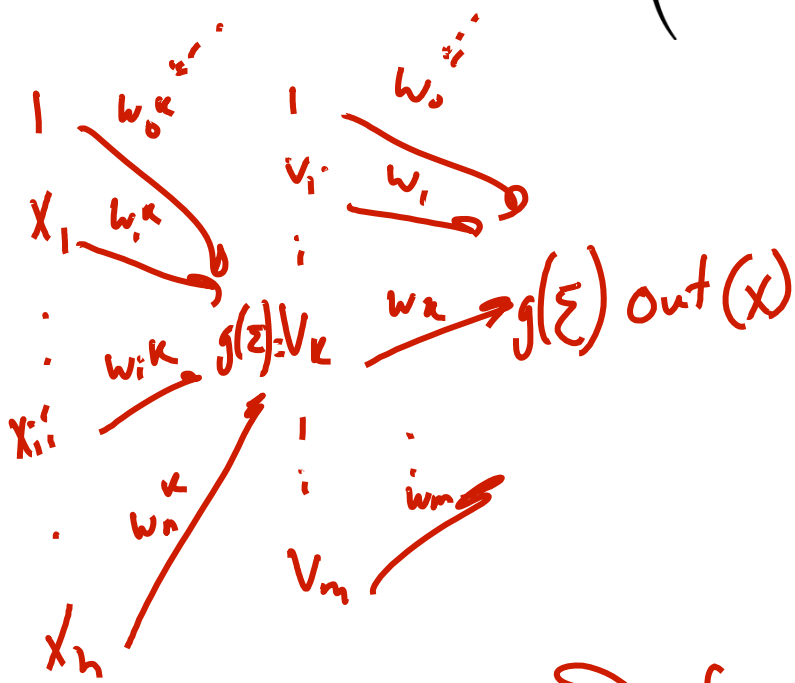
Weights in NN for images



Forward propagation for 1-hidden layer - Prediction

- 1-hidden layer:

$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k \overbrace{g(w_0^k + \sum_i w_i^k x_i)}^{V_k}\right)$$



forward propagation prediction

back prop compute gradients

Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_k}$

truth prediction

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g\left(\sum_{i'} w_{i'}^{k'} x_{i'}\right)\right)$$

$$\frac{\partial \ell(W)}{\partial w_k} = \sum_{j=1}^{m} -[y^j - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_k}$$

Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_i^k}$

Dropped w_0 to make derivation simpler

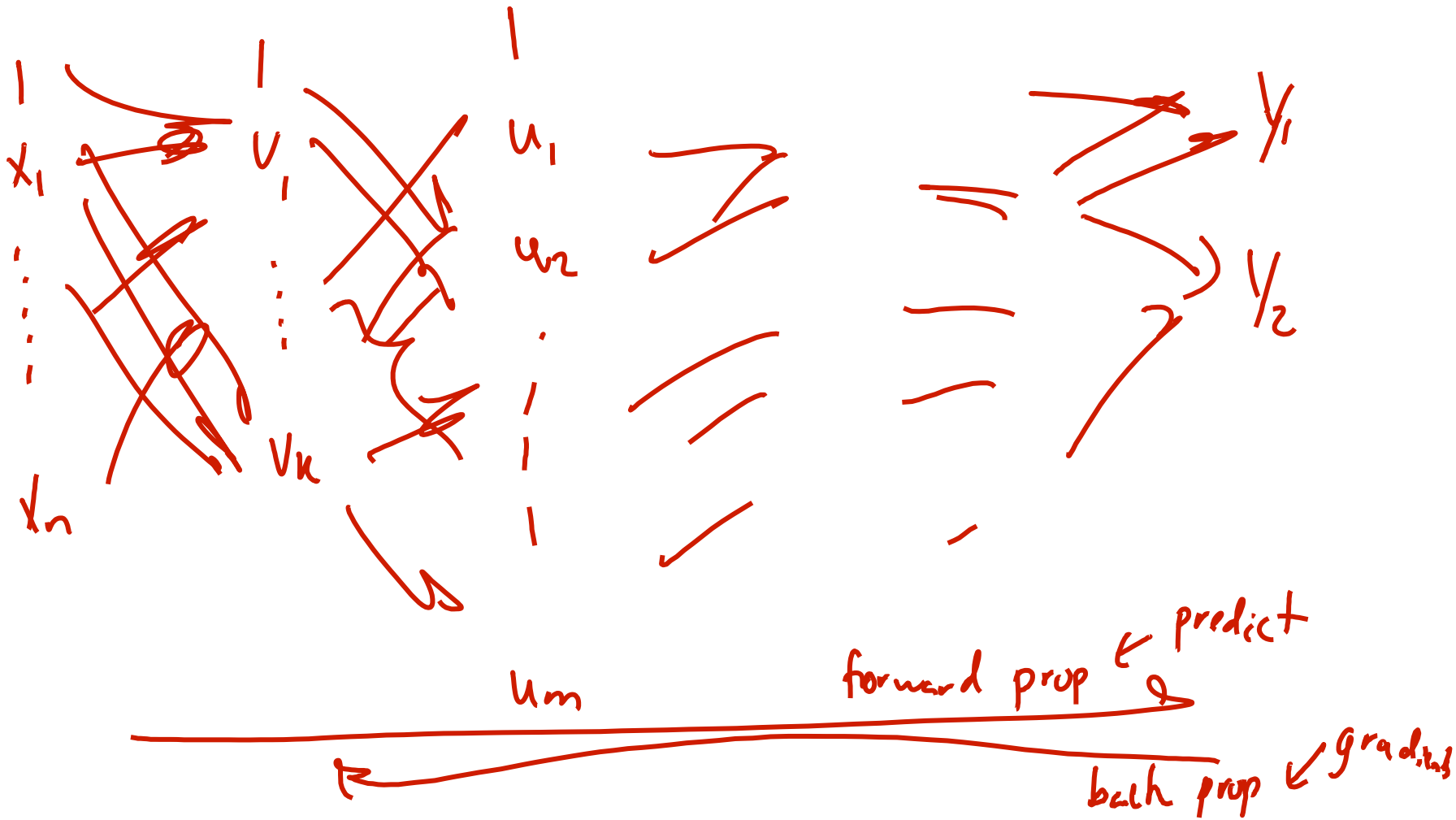
$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

$$\frac{\partial \ell(W)}{\partial w_i^k} = \sum_{j=1}^{m} -[y - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_i^k}$$

take gradient or SGD steps

Multilayer neural networks



Forward propagation – prediction

- Recursive algorithm
- Start from input layer
- Output of node V_k with parents $U_1, U_2, \ldots$:

$$V_k = g\left(\sum_i w_i^k U_i\right)$$

Back-propagation – learning

- Just stochastic gradient descent!!!
- Recursive algorithm for computing gradient
- For each example
  - Perform forward propagation
  - Start from output layer
  - Compute gradient of node V_k with parents $U_1, U_2, \ldots$
  - Update weight w_i^k

Many possible response/link functions

- Sigmoid
- Linear
- Exponential
- Gaussian
- Hinge
- Max
- …

Convolutional Neural Networks & Application to Computer Vision

Machine Learning – CSEP546

Carlos Guestrin

University of Washington

March 3, 2014

©Carlos Guestrin 2005-2014

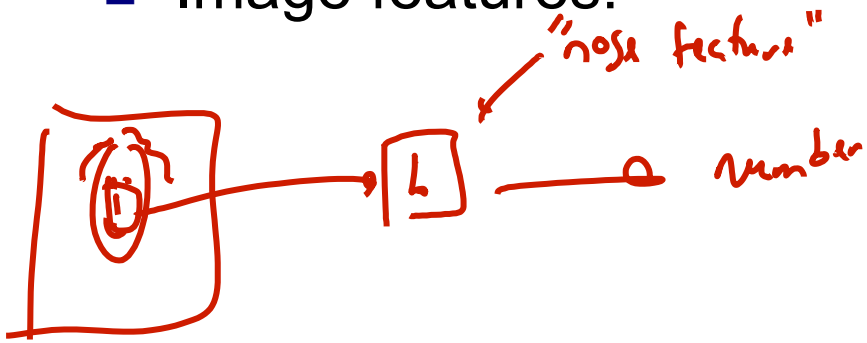
Contains slides from…

- LeCun & Ranzato
- Russ Salakhutdinov
- Honglak Lee

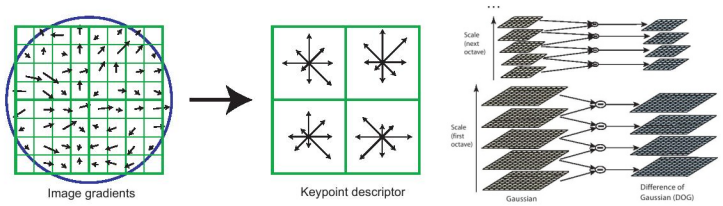
Neural Networks in Computer Vision

- Neural nets have made an amazing come back
  - Used to engineer high-level features of images

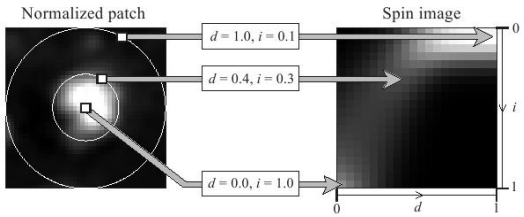
- Image features:



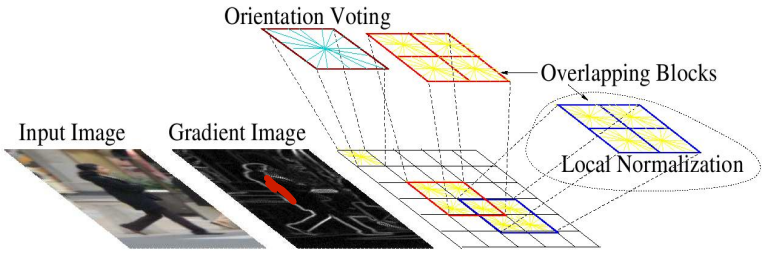
Some hand-created image features



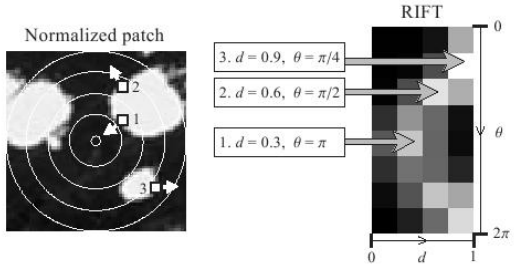
SIFT



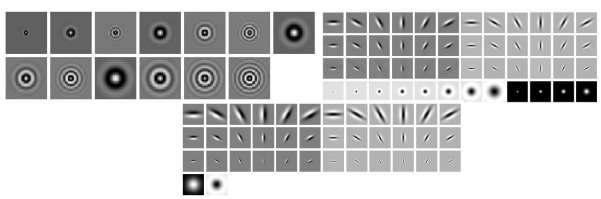
Spin image



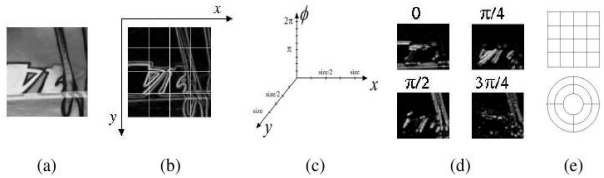
HoG



RIFT



Textons



GLOH

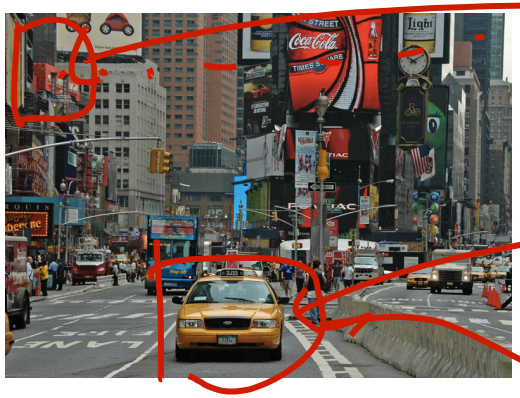
Slide Credit: Honglak Lee

Scanning an image with a detector

- Detector = Classifier from image patches:

Patch image → number
→ class
⋮

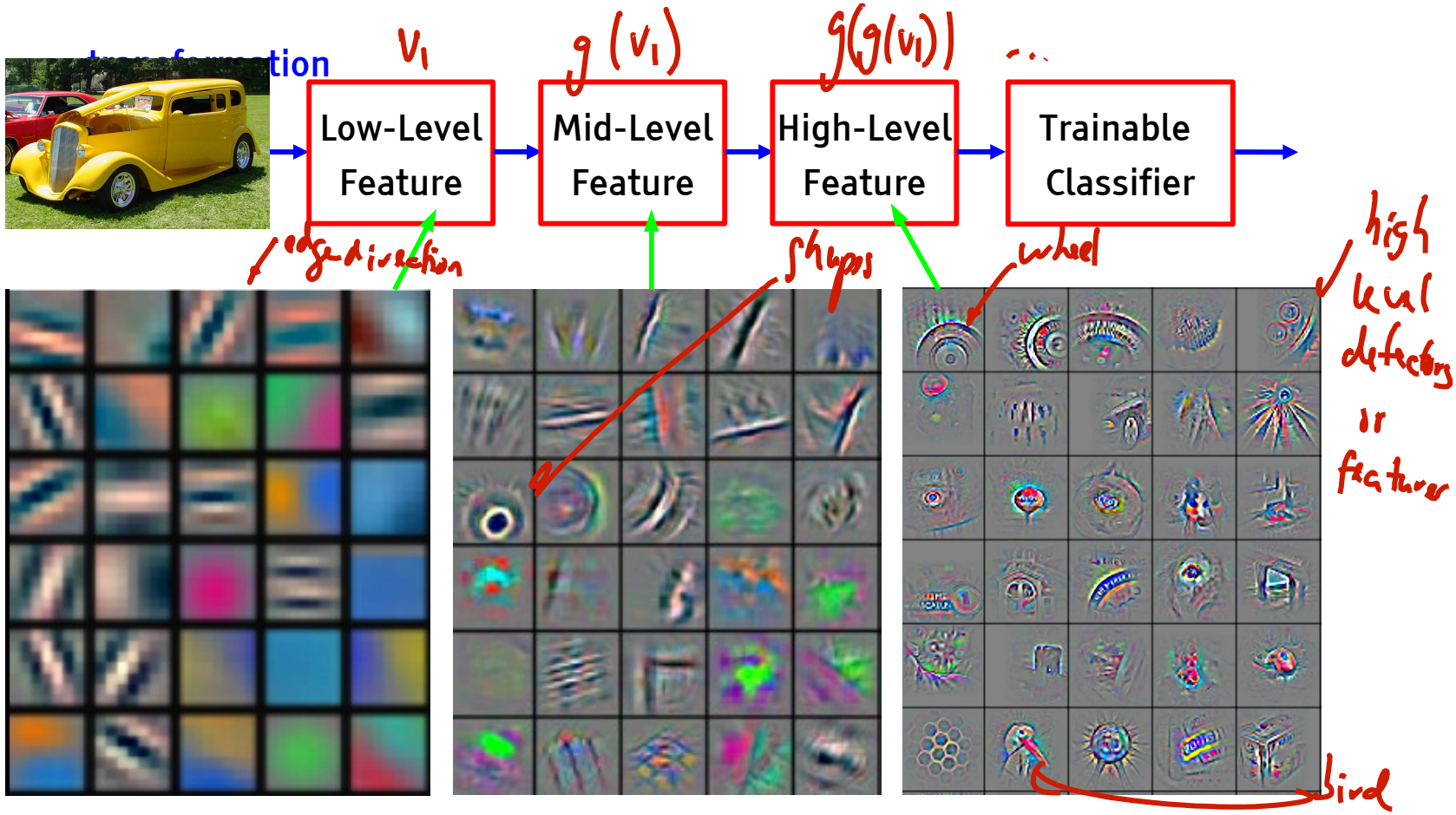
- Typically scan image with detector:



low output

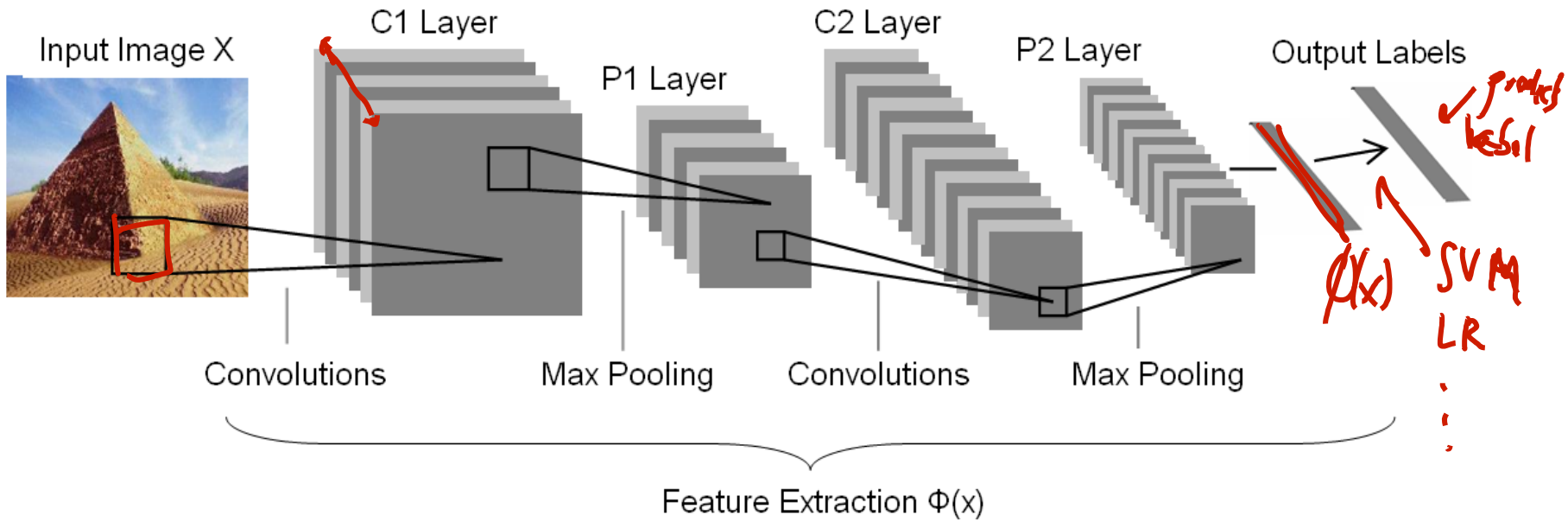
high output

Using neural nets to learn non-linear features



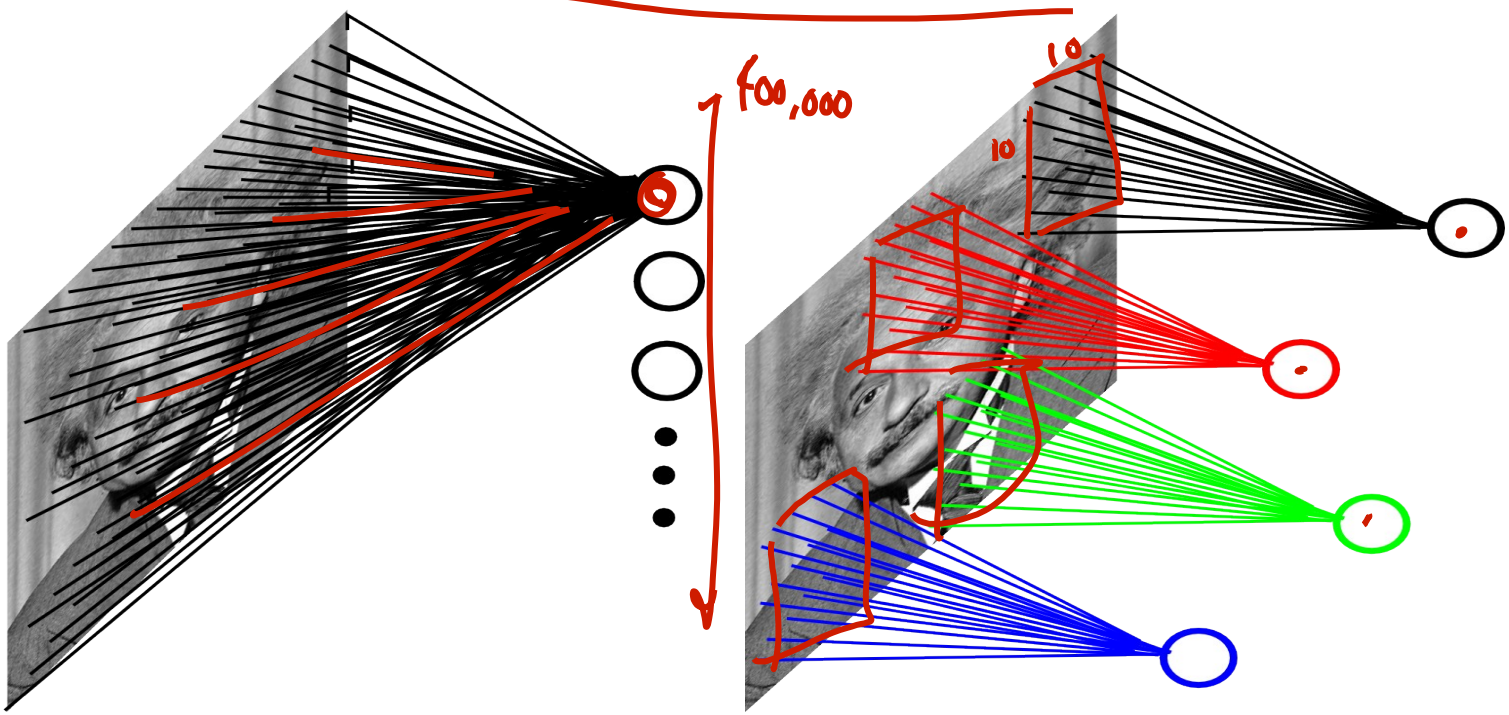
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

But, many tricks needed to work well…



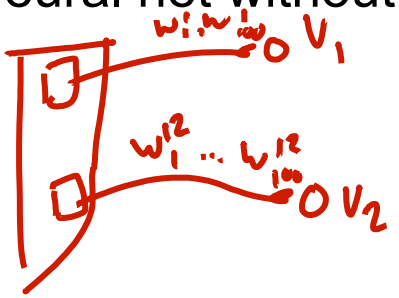
Convolution Layer

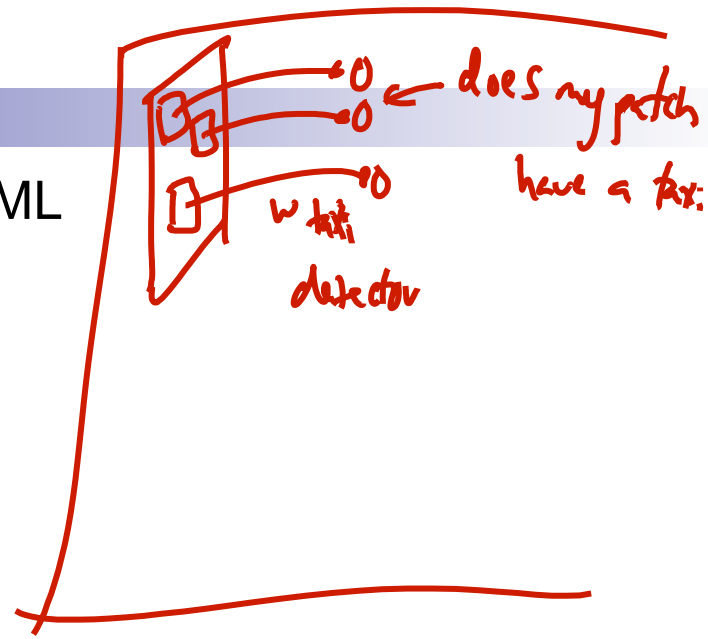
- Example: 200x200 image
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



Parameter sharing

- Fundamental technique used throughout ML
- Neural net without parameter sharing:





- Sharing parameters:

$$\left.\begin{array}{l} w_1^1 = w_1^{12} \\ \vdots \\ w_{100}^1 = w_{100}^{12} \end{array}\right\}$$

all convolutional units in layer use same parameters

Pooling/Subsampling

- Convolutions act like detectors:

only one taxi here

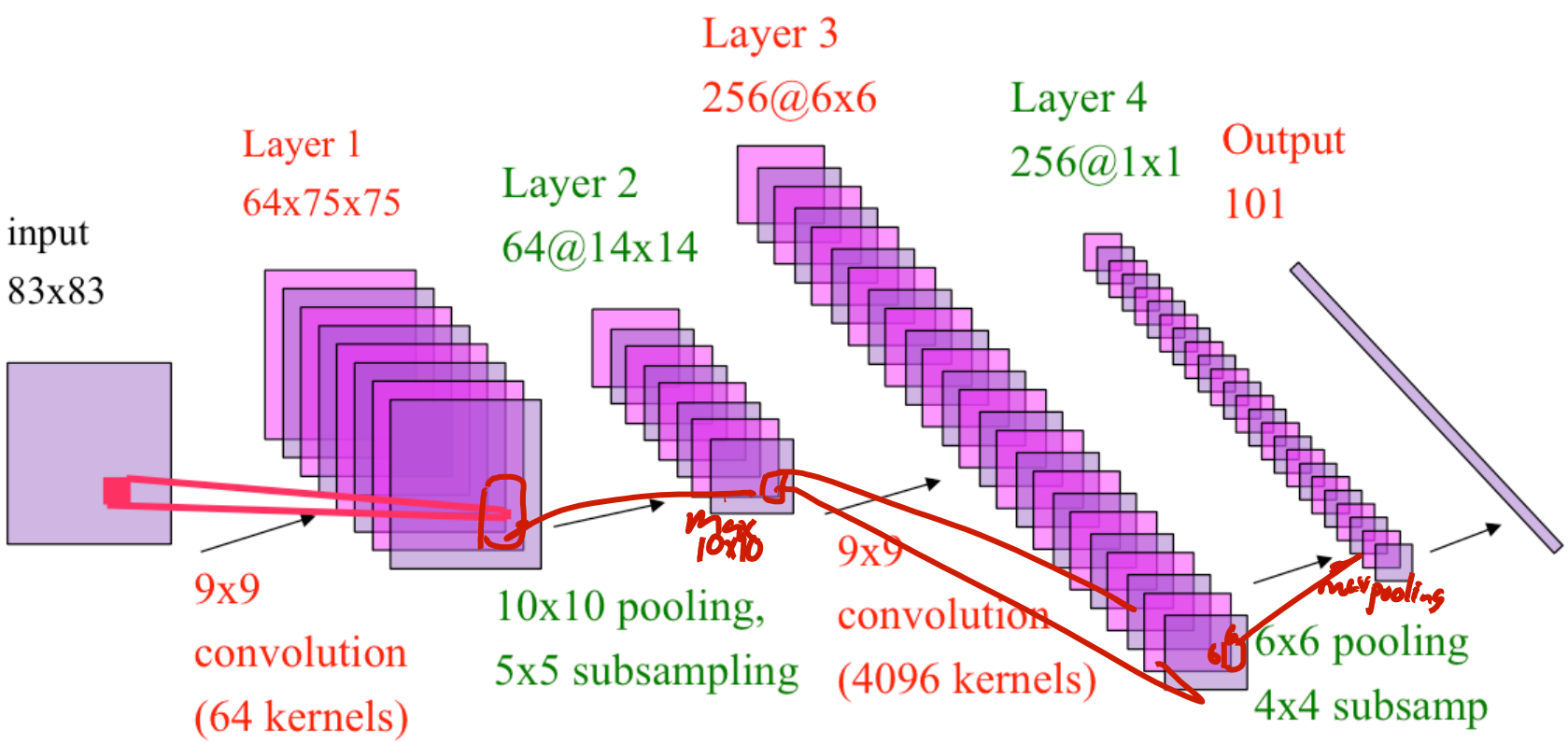
- But we don't expect true detections in every patch
- Pooling/subsampling nodes:

single number

max pooling

max of detector values for all patches in region

Example neural net architecture



Sample results

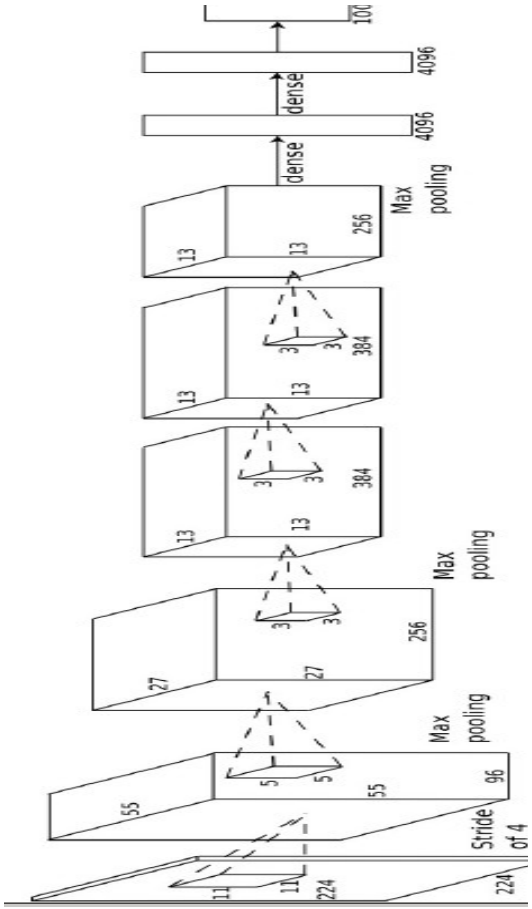
- Traffic Sign Recognition (GTSRB)
  - German Traffic Sign Reco Bench
  - 99.2% accuracy

- House Number Recognition (Google)
  - Street View House Numbers
  - 94.3 % accuracy

Example from Krizhevsky, Sutskever, Hinton 2012

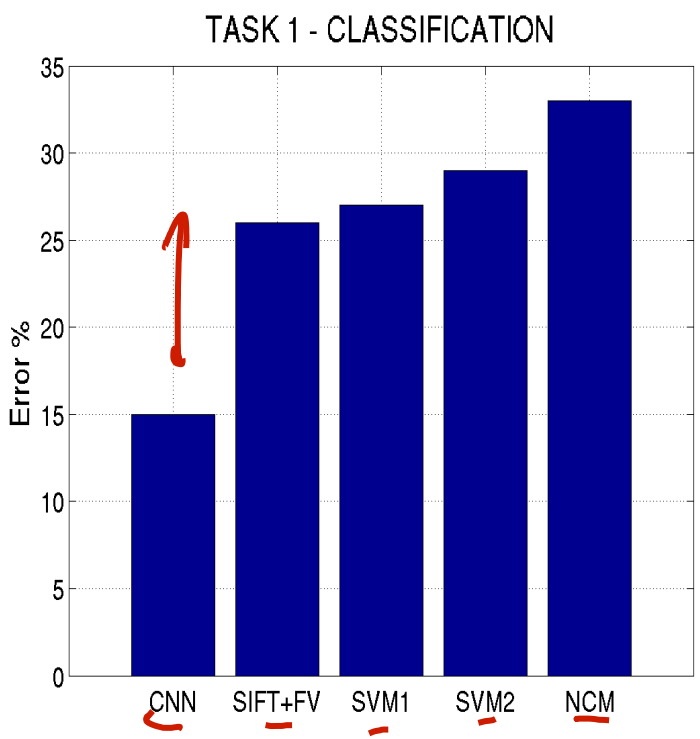
- Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops

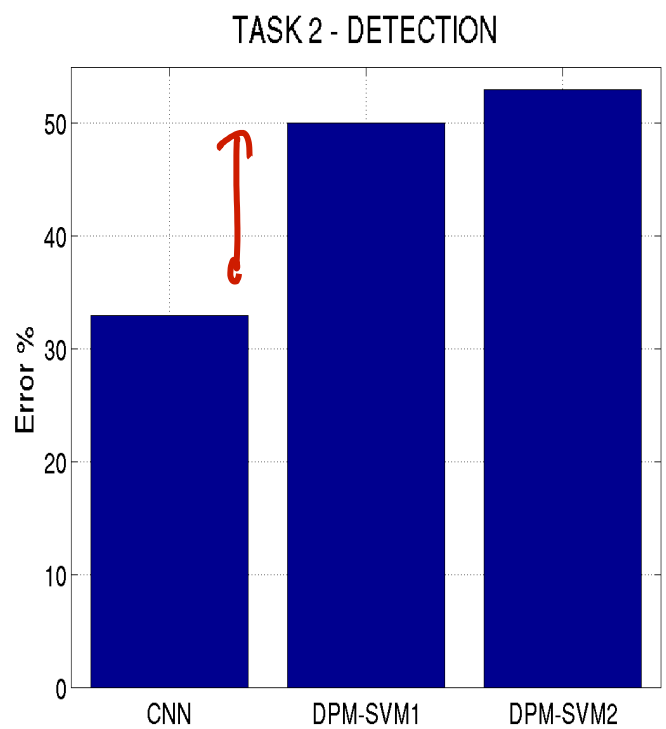
4M	FULL CONNECT	4Mflop
16M	FULL 4096/ReLU	16M
37M	FULL 4096/ReLU	37M
	MAX POOLING	
442K	CONV 3x3/ReLU 256fm	74M
1.3M	CONV 3x3ReLU 384fm	224M
884K	CONV 3x3/ReLU 384fm	149M
	MAX POOLING 2x2sub	
	LOCAL CONTRAST NORM	
307K	CONV 11x11/ReLU 256fm	223M
	MAX POOL 2x2sub	
	LOCAL CONTRAST NORM	
35K	CONV 11x11/ReLU 96fm	105M

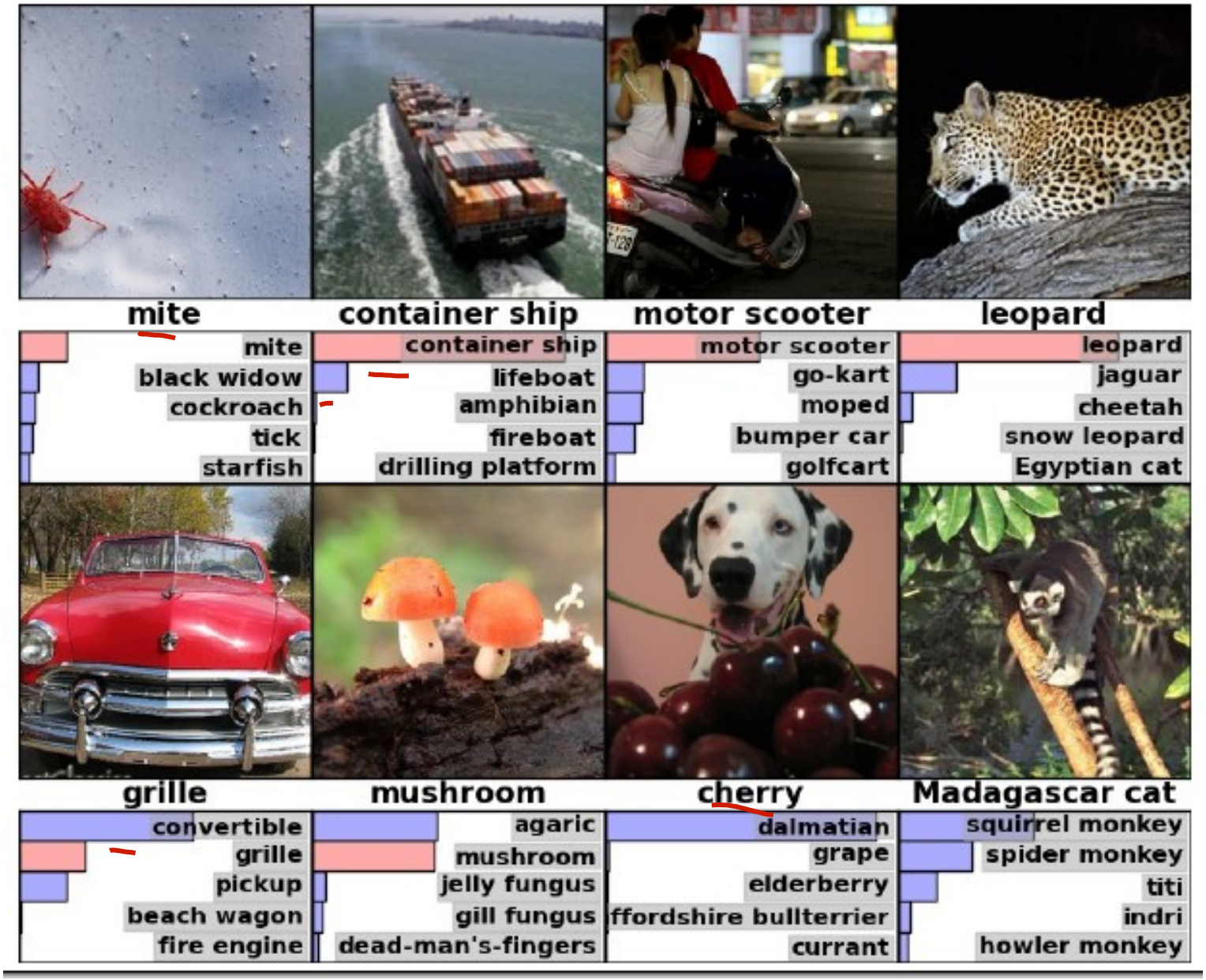


Results by Krizhevsky, Sutskever, Hinton 2012

- ImageNet Large Scale Visual Recognition Challenge
- 1000 categories, 1.5 Million labeled training samples

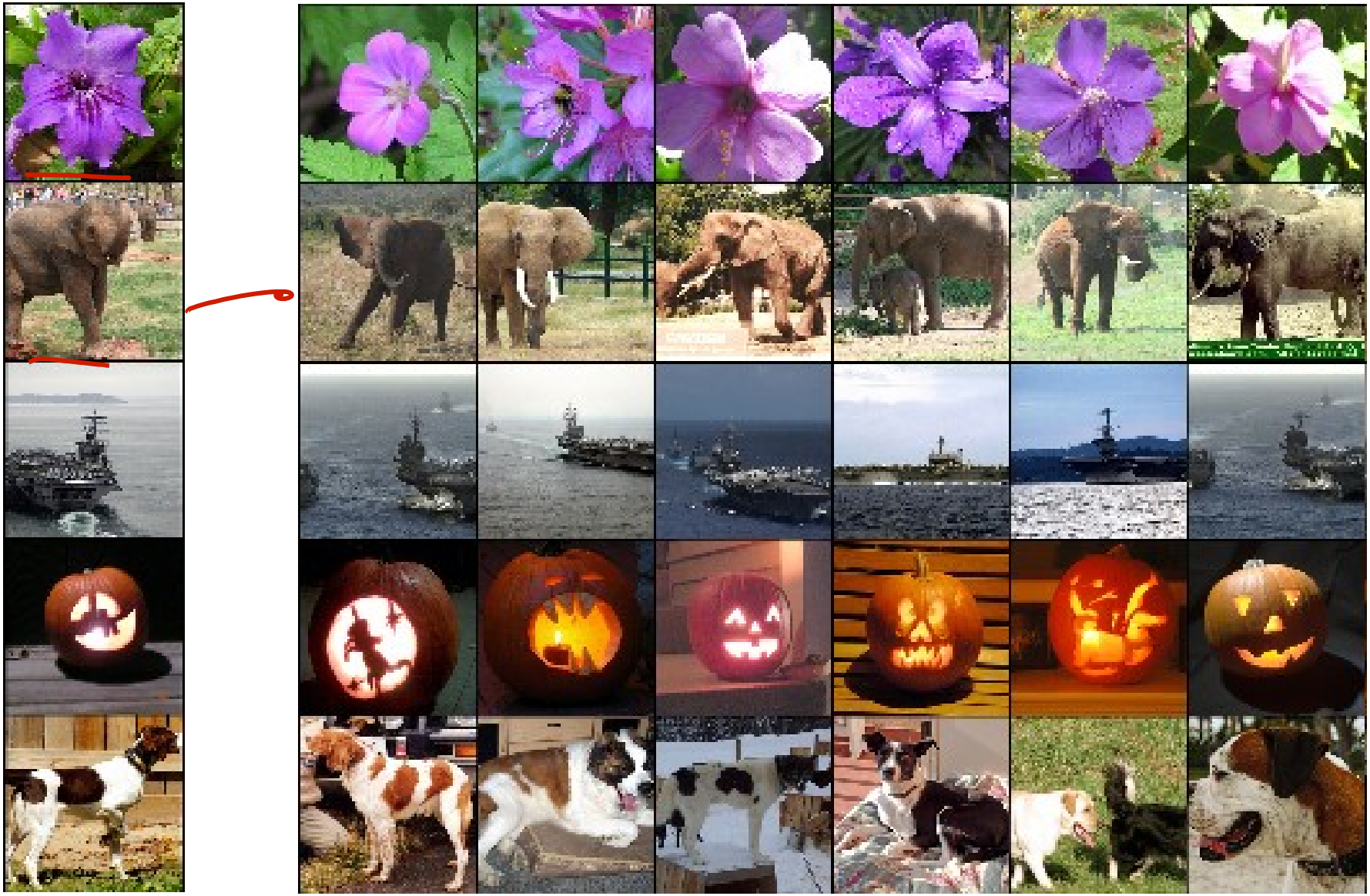




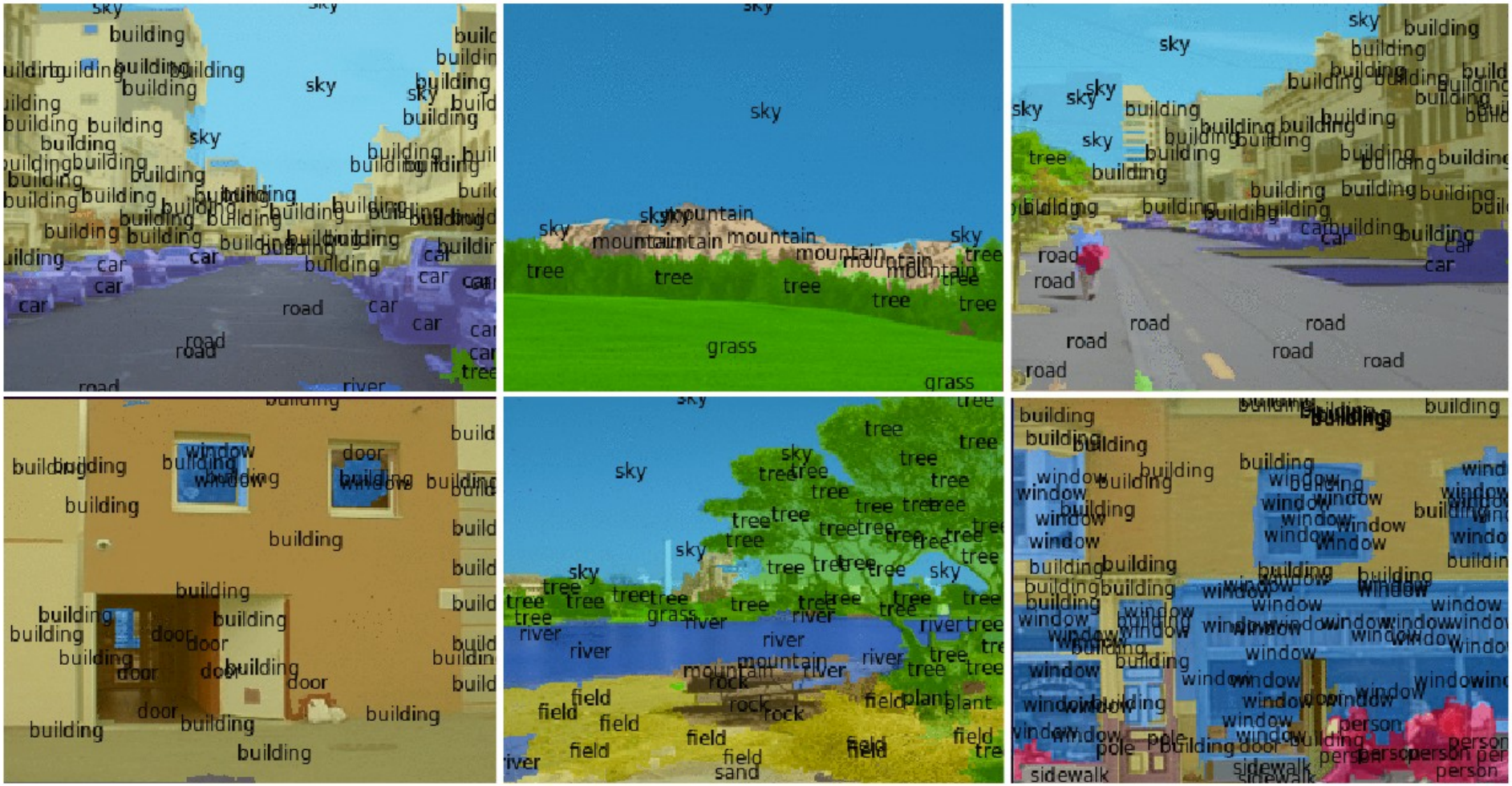

mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

TEST IMAGE

RETRIEVED IMAGES

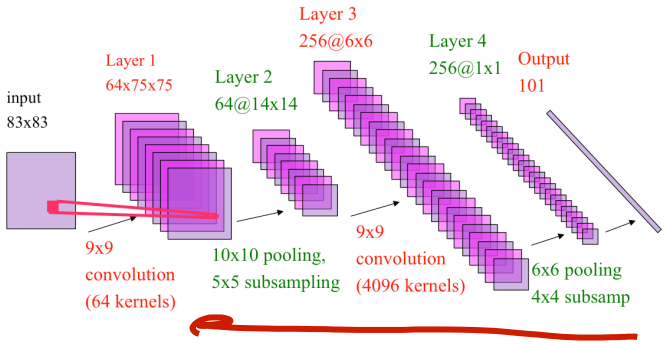


Application to scene parsing



[Farabet et al. ICML 2012, PAMI 2013]

Learning challenges for neural nets

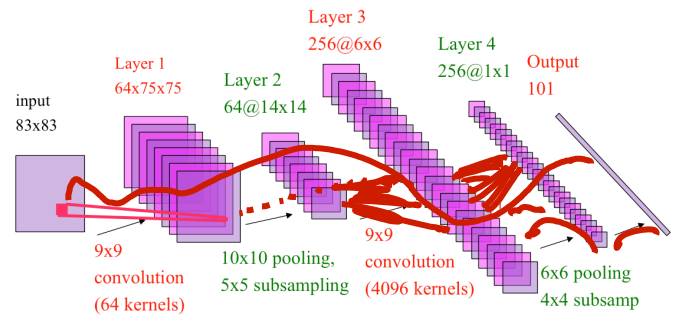


- Choosing architecture
- Slow per iteration and convergence

GPUs

- Gradient "diffusion" across layers ←
- Many local optima

Random dropouts

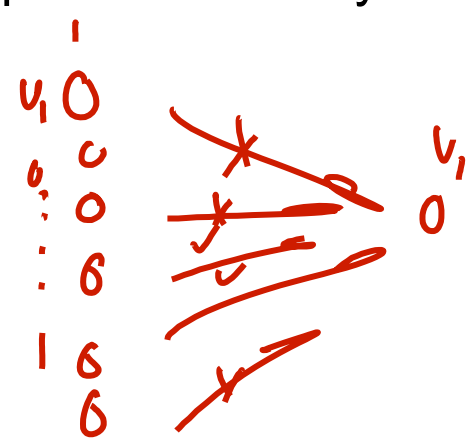


- Standard backprop:

$$w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j$$

gradient updates go to zero over layers

- Random dropouts: randomly choose edges not to update:



- Functions as a type of "regularization"… helps avoid "diffusion" of gradient

Revival of neural networks

- Neural networks fell into disfavor in mid 90s -early 2000s
  - Many methods have now been rediscovered ☺
- Exciting new results using modifications to optimization techniques and GPUs

- Challenges still remain:
  - Architecture selection feels like a black art
  - Optimization can be very sensitive to parameters
  - Requires a significant amount of expertise to get good results