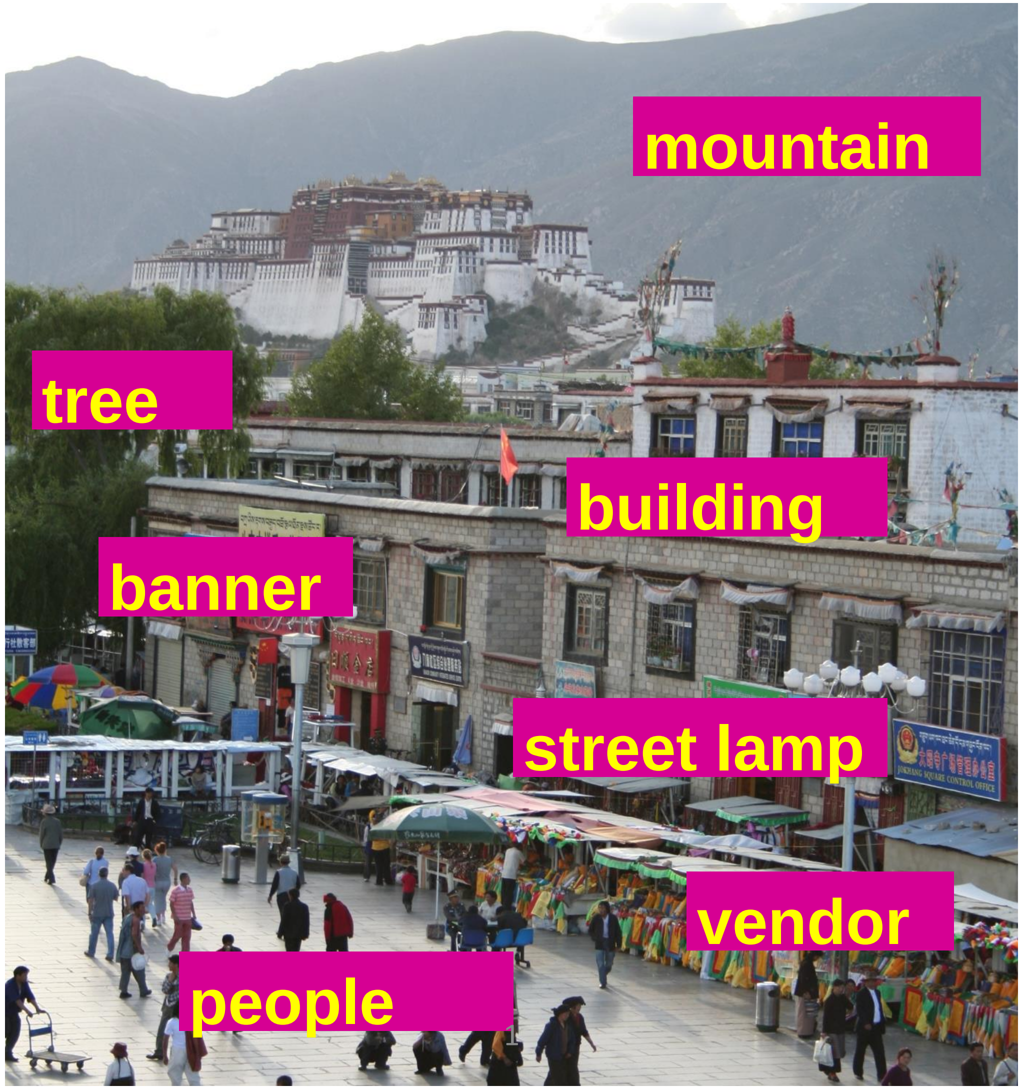
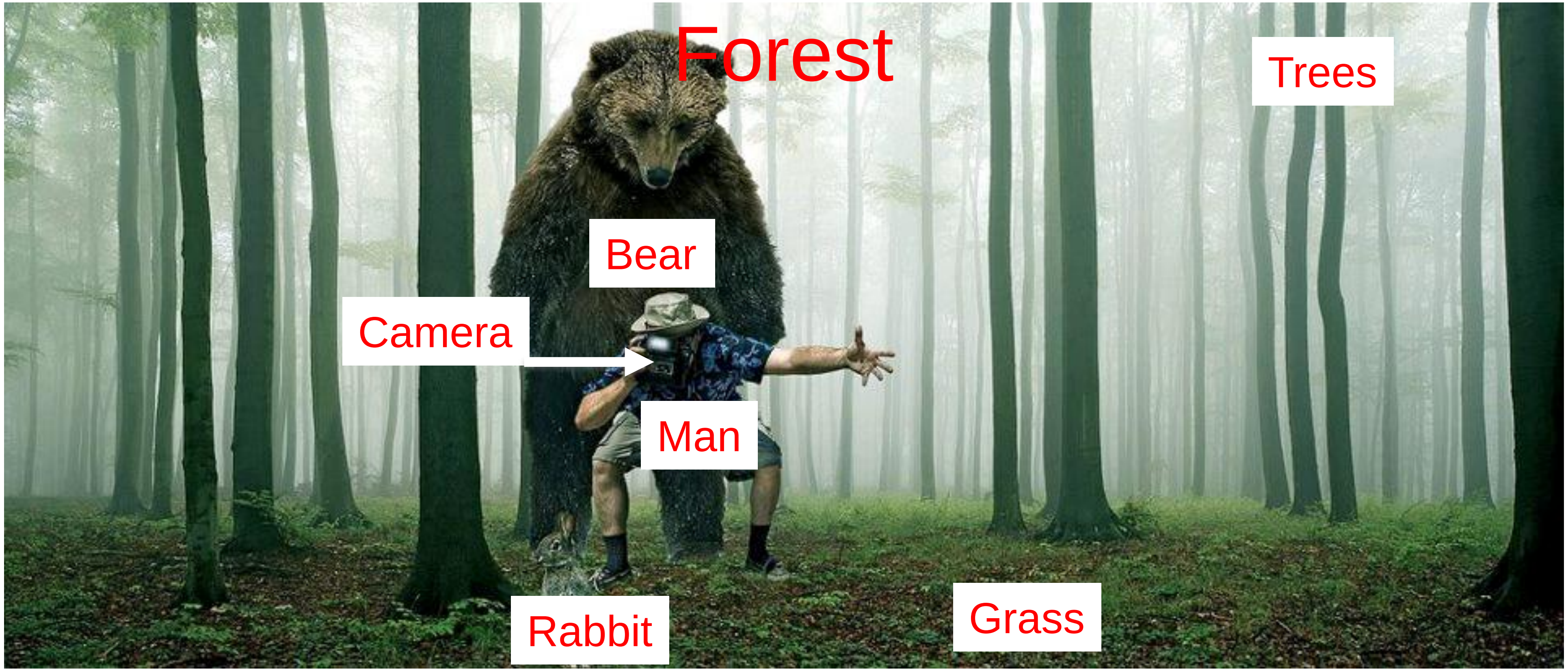
15. Classification



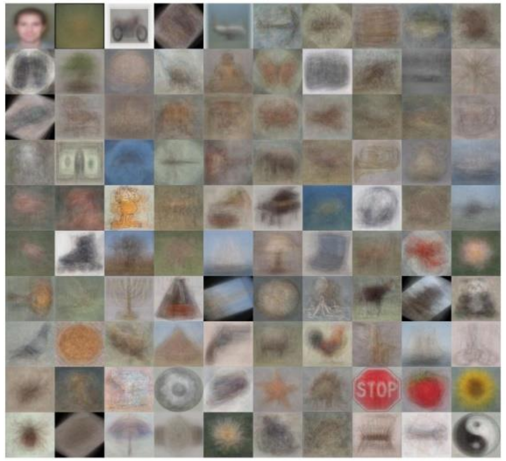
Classification

- Classification allows us understand, predict, and interact with the surrounding environment
  - Is it dangerous? How fast does it run? Can I poke with it?

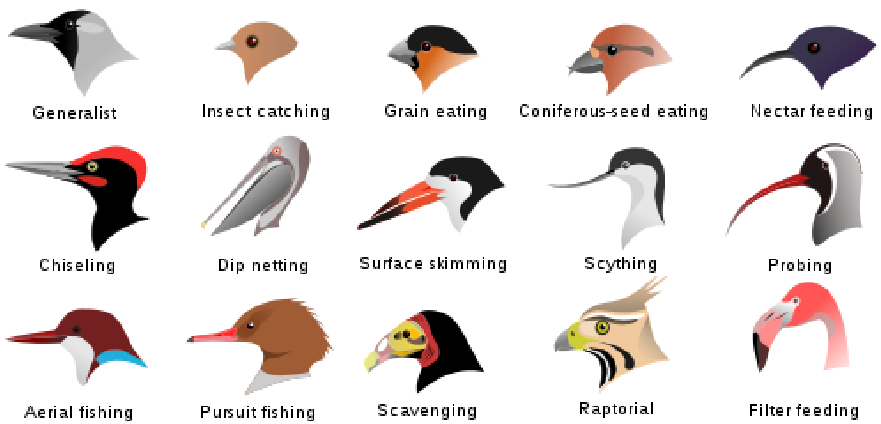


Sample Classification Problems

- Object recognition
- Place recognition
- Fine-grained recognition



Caltech 101 Average Object Images



Places Database [Zhou et al. NIPS 2014]

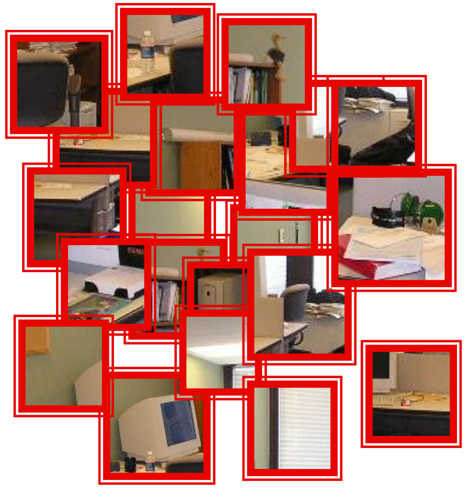
Object Detection by Classification

Object detection can be formulated as a classification problem.

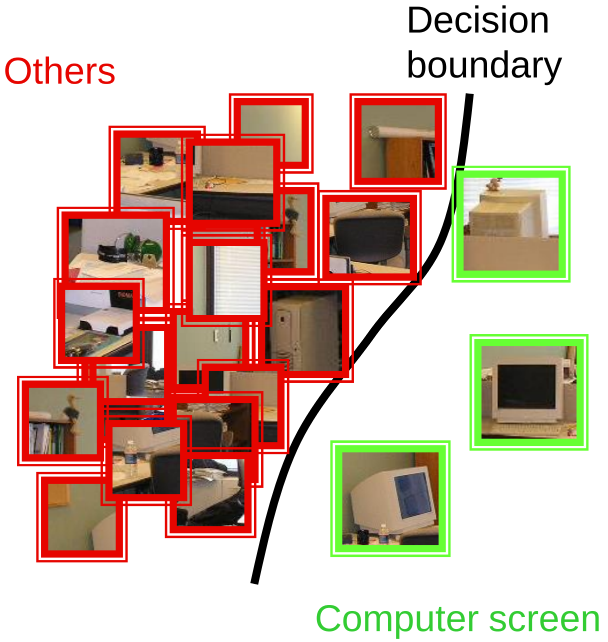
The image is partitioned into a set of overlapping windows

… and a decision is made at each window about if it contains a target object or not.

Where are the screens?



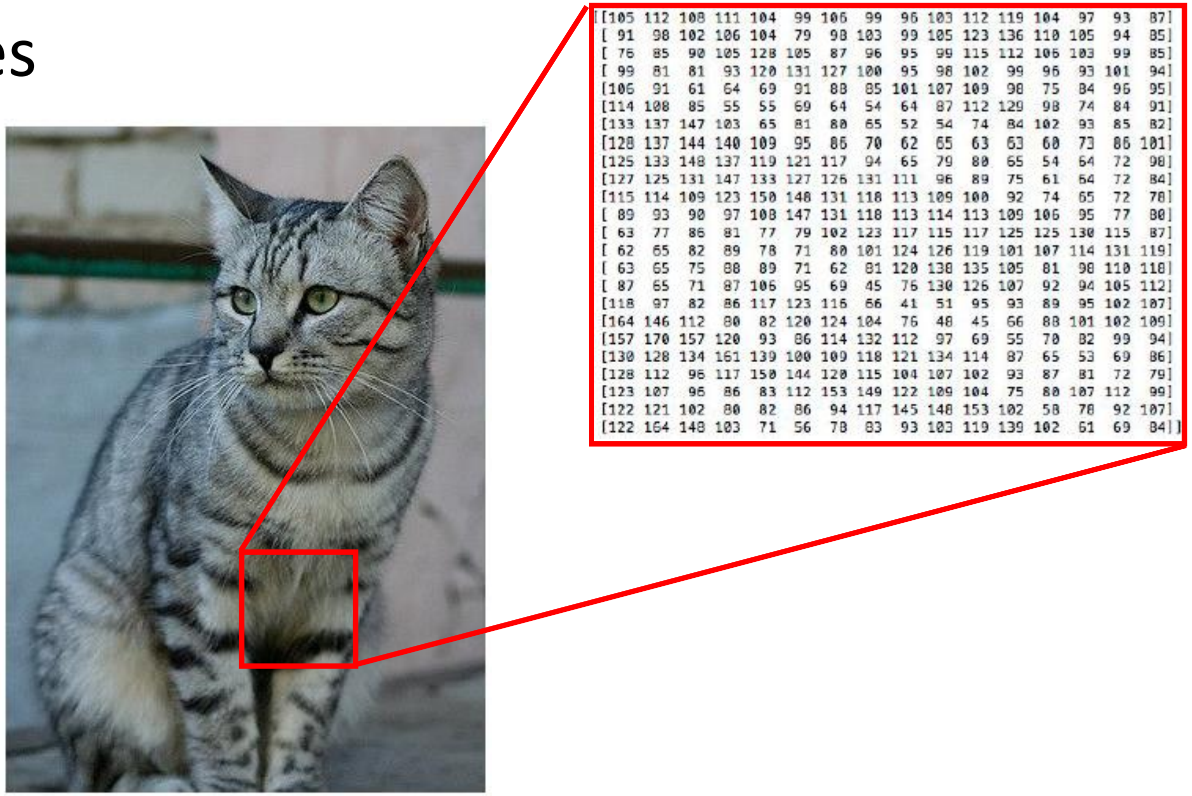
overlapping image patches



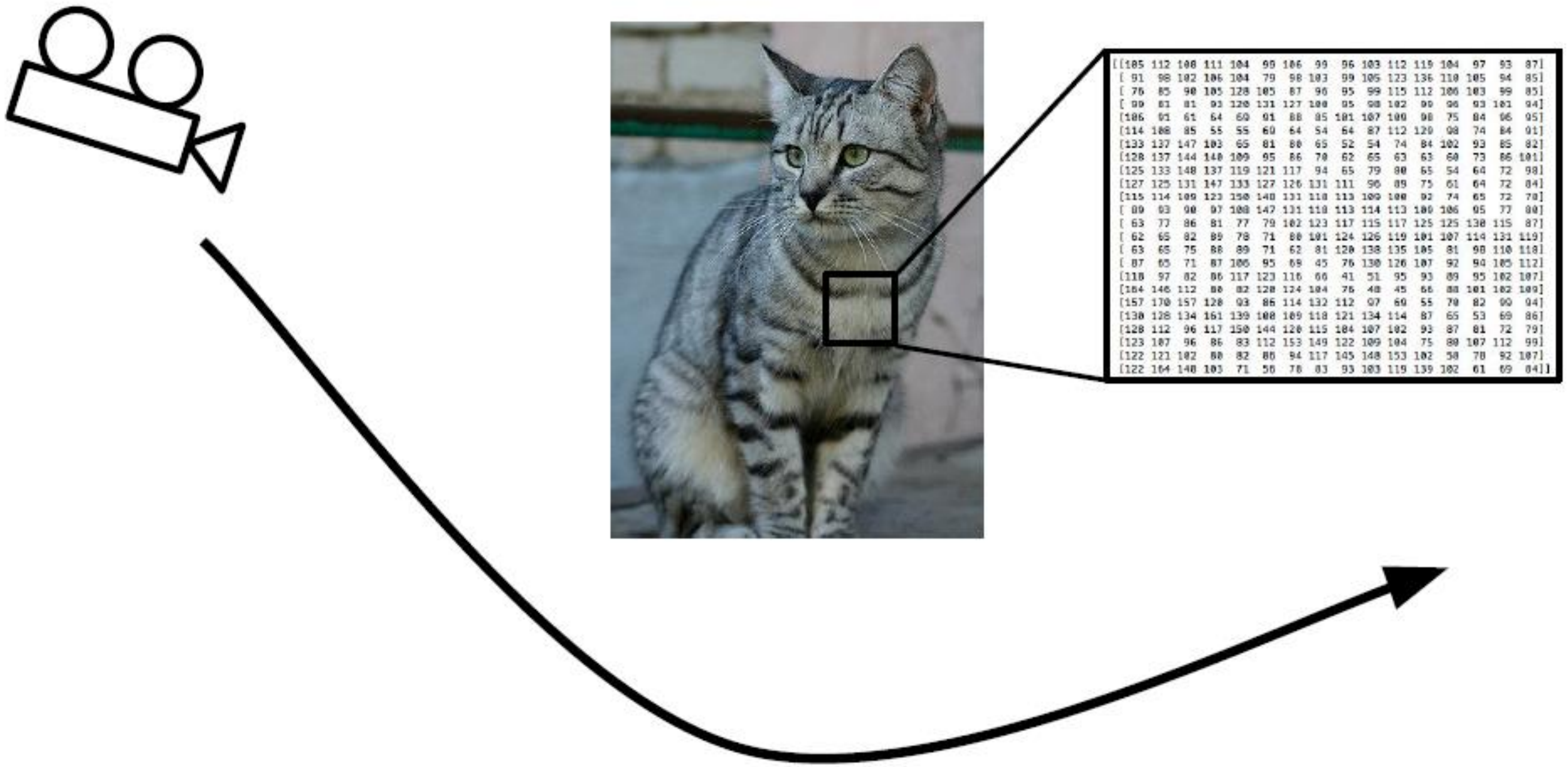
In some feature space

The Problem: Semantic Gap

- Images are represented by RGB values
- The computer does not ‘see’ the semantics of the image!
  - No easy way to write a program to recognize a cat
  - Unlike sorting a set of numbers



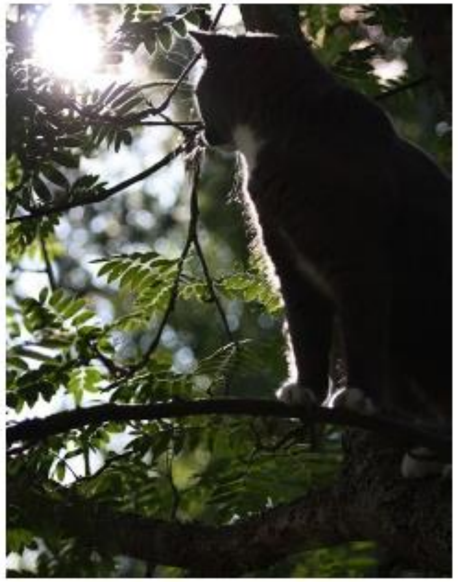
Challenges: Viewpoint variation



All pixels change when
the camera moves!

Challenges: Illumination



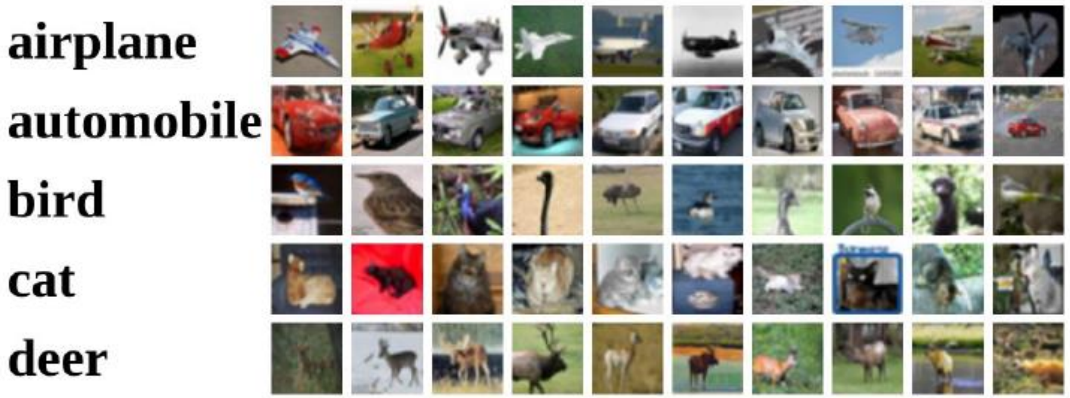



Challenges: Deformation

Challenges: Occlusion

Learned Classifier

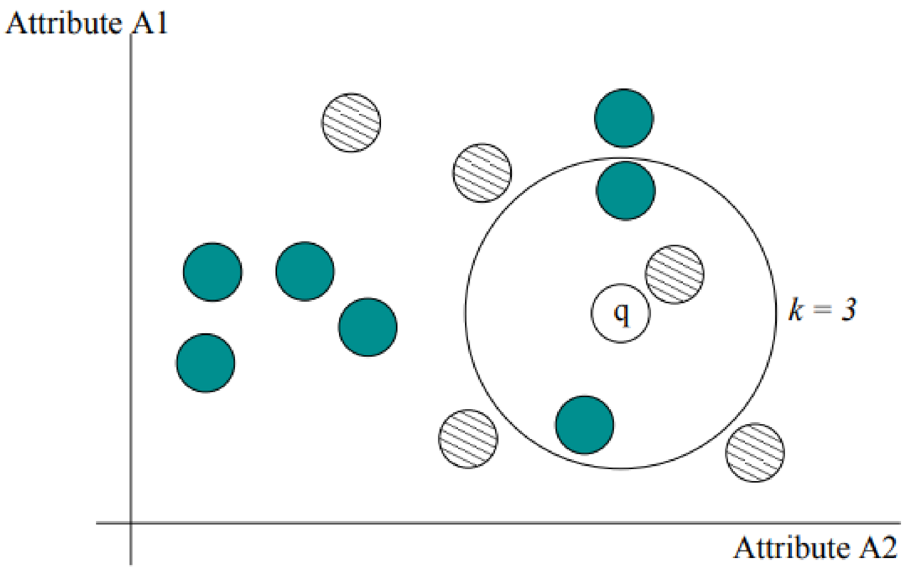
- Given training images with labels
- Learn a “classifier”
- Evaluate new images with the learned classifier



example of training images

Example: Nearest Neighbor Classifier

- Given training images with labels
- Learn a “classifier”: memorize all data (images + labels)
- Evaluate new images: find the most similar training image

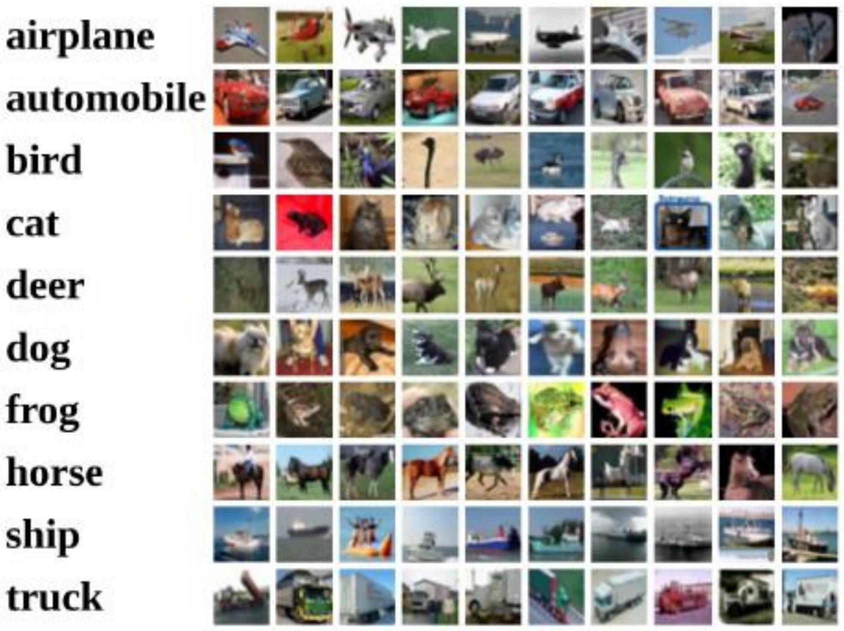


Might be generalized to k-Nearest Neighbors:
Finding the k most similar training images, and let them vote

Example: KNN with CIFAR10

- 10 classes
- 50,000 training images
- 10,000 testing images

training data



testing data with nearest neighbors

Distance between Images?

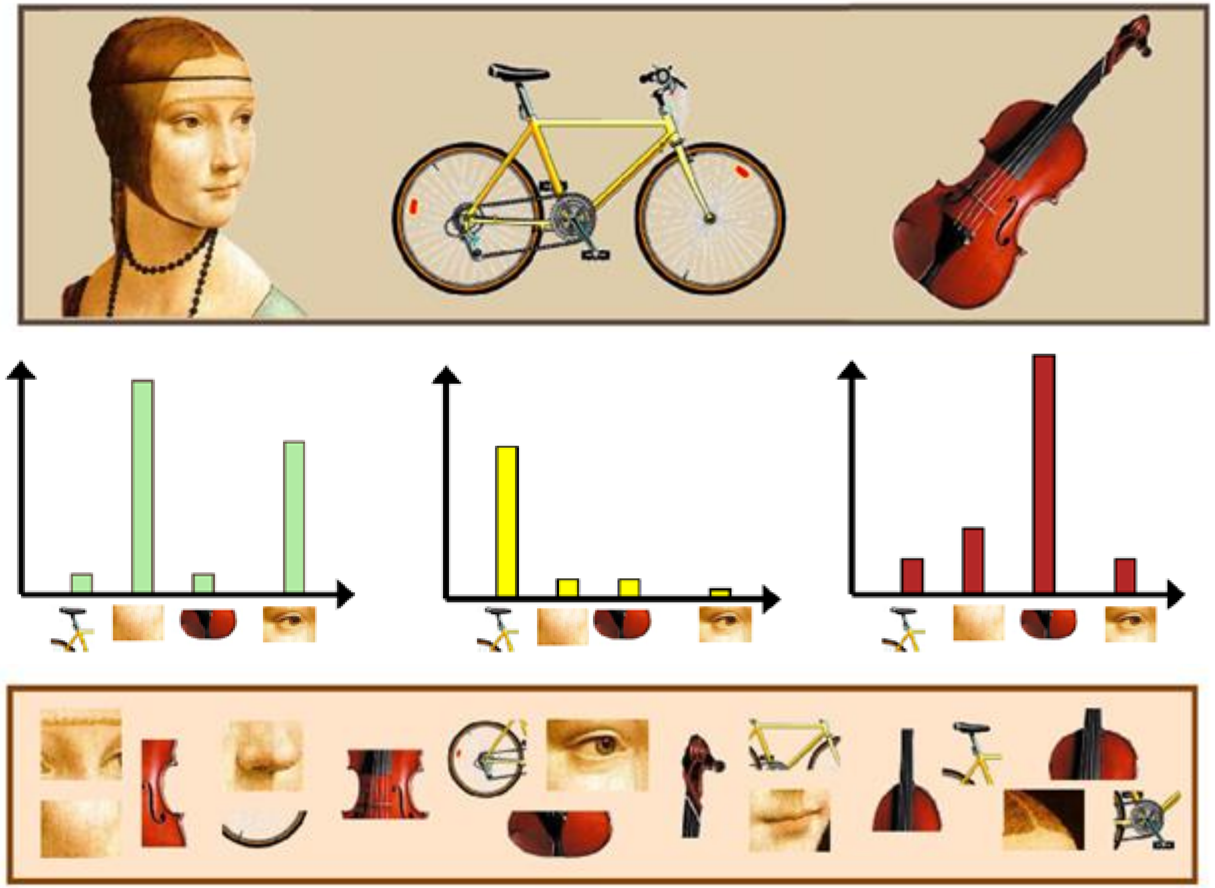
- Take each image as a matrix, and compute matrix distances (this would never work)
    - L1 distance: $d_1(I_1, I_2) = \sum_x |I_1(x) - I_2(x)|$
    - L2 distance: $d_2(I_1, I_2) = \sqrt{\sum_x |I_1(x) - I_2(x)|^2}$
- Better to compute a 'feature vector' for each image
    - And compute distance between 'feature vectors', e.g. L_1, L_2, etc.

Bag-of-Words Features

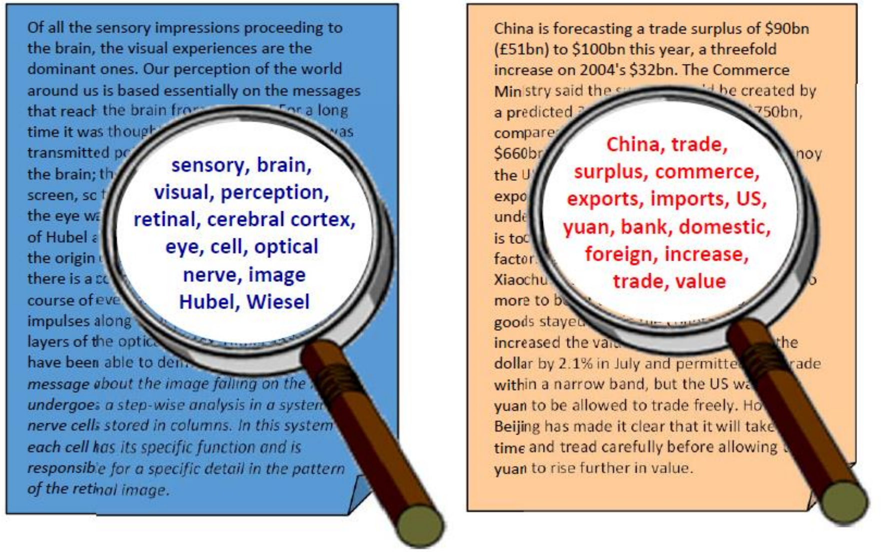
images

feature vectors: the frequency of visual words

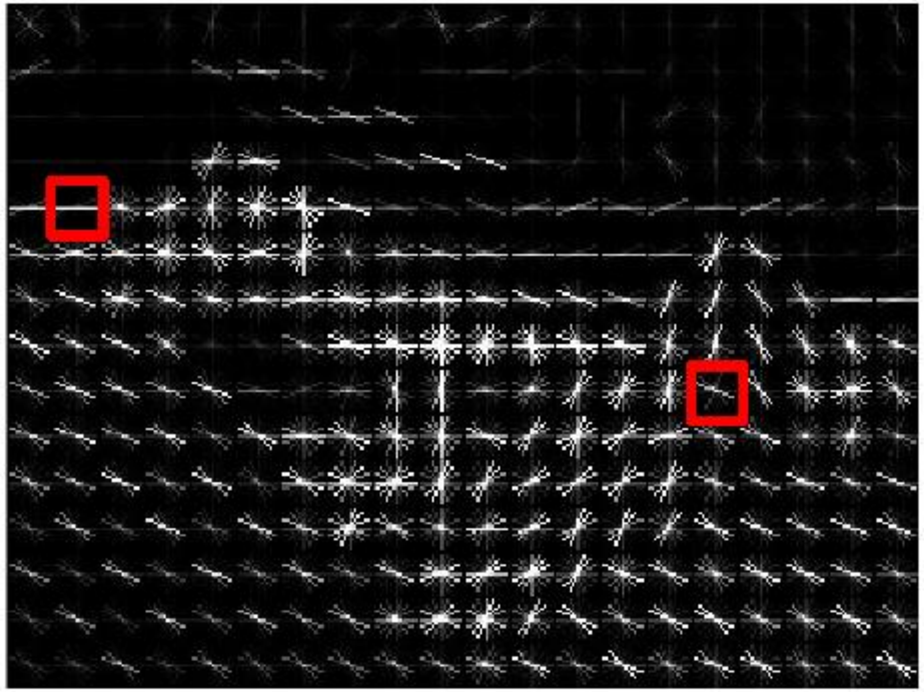
'visual words'



Analogy to documents features



HOG Features



Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Recap: Classification

- Given training images with labels
- Compute a feature vector (e.g. BoW, HOG, etc) for each image
- Learn a classifier (e.g. K-NN)
- Apply the classifier to unseen testing images

Questions?

Parametric Classifiers

- A function (with parameters) to predict the category label

$$\hat{y}_i = f(x_i, W)$$

the classifier   an input image   parameters

- Training:
  - Given a set of training data $\{(x_i, y_i)\}$
  - Estimate parameters W, such that $\hat{y}_i = y_i$ on the training data

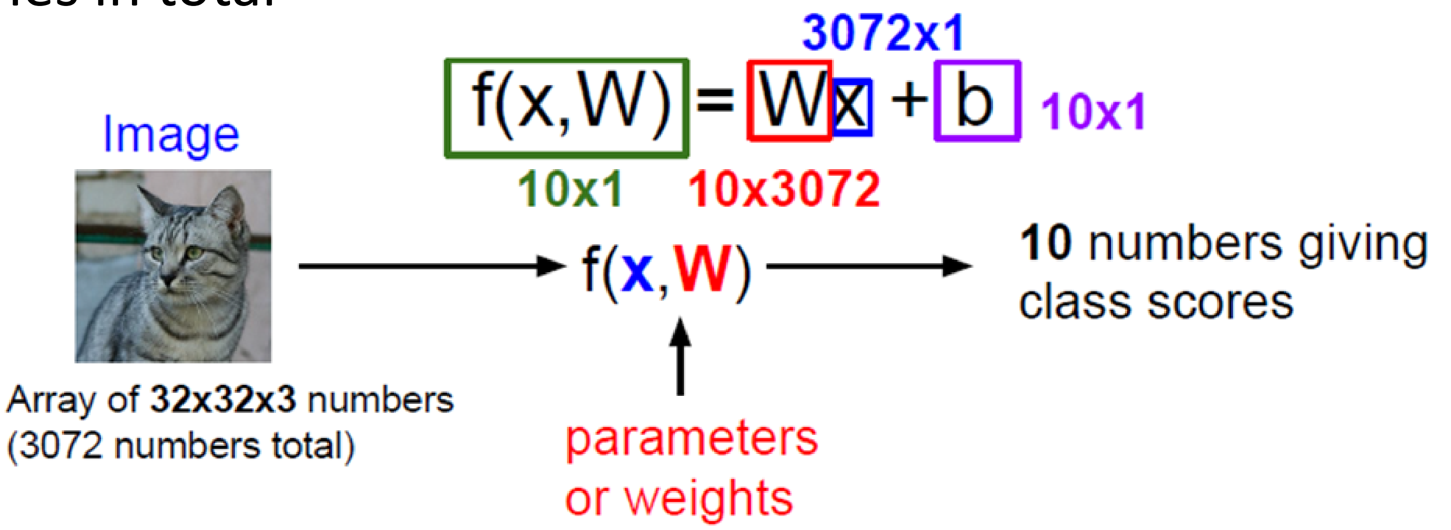
- There are different choices for **function f**, **parameter estimation method**, etc.

Linear Classifier

- A simple example is linear classifier, where $f(\cdot,\cdot)$ is a linear function
- For example:

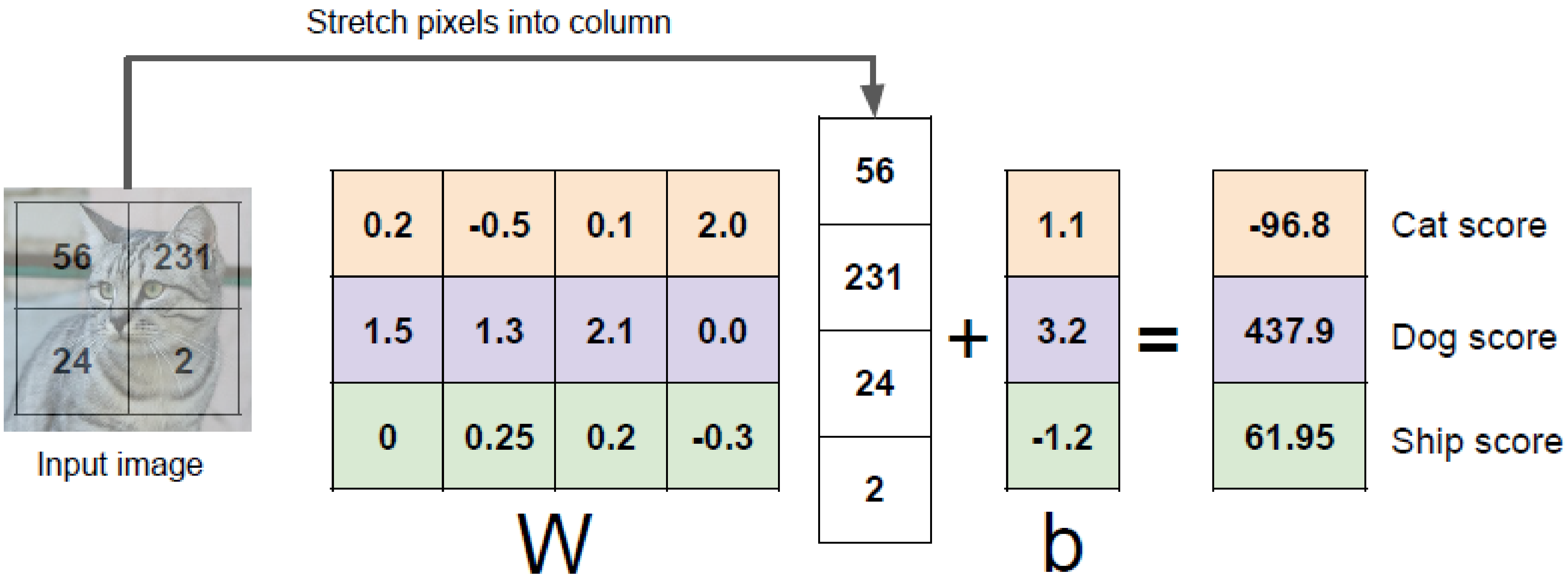
$$s = f(x, W) = Wx + b$$

  - Input: x is a vector formed by concatenating all pixel values
  - Output: s is a vector giving class scores, e.g., 10×1 vector if there are 10 categories in total



Linear Classifier: An Example

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Parameter Estimation

- To estimate parameters W from training data $\{(x_i, y_i)\}$
- Need to measure the consistency between prediction and label
  - by a loss function $L_i(y_i, s_i)$ – larger when y_i is inconsistent with $s_i = f(x_i, W)$

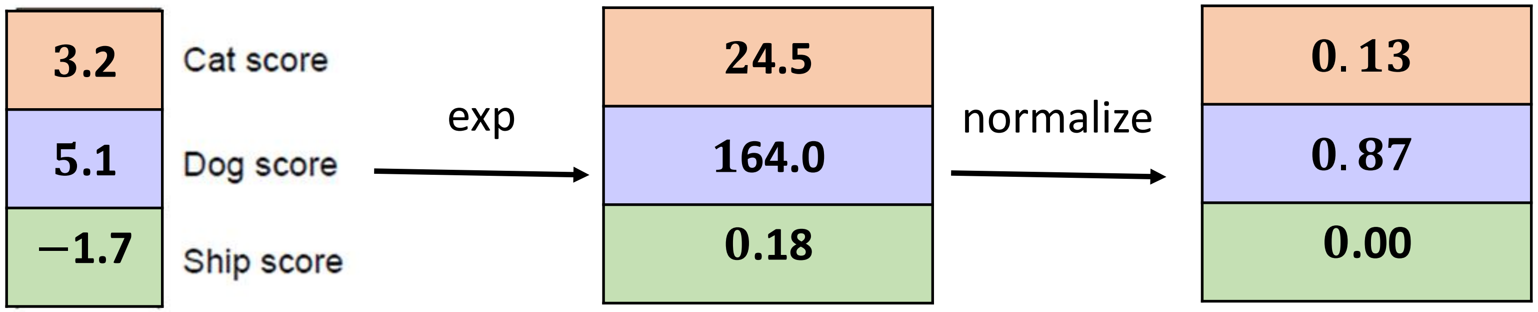
Note s_i is a vector contain scores for all categories

- The parameters are estimated by minimizing the loss

$$W^* = \arg\min_{\mathrm{W}} \sum_i L_i(y_i, s_i)$$

SoftMax Loss

- We can interpret the category score s_i as probabilities
- That requires some manipulations of the original scores
  - $P(Y = k|X = x_i) = \frac{e^{s_i(k)}}{\sum_j e^{s_i(j)}}$ (called softmax function)



Probabilities must be non-negative

Probabilities must add to 1

SoftMax Loss

1
0
0

- Evaluates how well the current s_i is

$$L_i(y_i, s_i) = -\log s_i(y_i)$$

the probability vector

the ground truth class label

- It maximizes the likelihood of the truth class

3.2	Cat score
5.1	Dog score
−1.7	Ship score

exp →

24.5
164.0
0.18

Probabilities must be non-negative

normalize →

0.13
0.87
0.00

Probabilities must add to 1

→

$$L_i = -\log P(Y = y_i | X = x_i) = -\log 0.13 = 0.89$$

Maximum Likelihood Estimation

Maximizing the likelihood of the correct label.

SoftMax Loss

- Evaluates how well the current s_i is

$$L_i(y_i, s_i) = -\log s_i(y_i)$$

the probability vector

the ground truth class label

- It maximizes the likelihood of the truth class

3.2	Cat score
5.1	Dog score
−1.7	Ship score

$\xrightarrow{\text{exp}}$

24.5
164.0
0.18

Probabilities must be non-negative

$\xrightarrow{\text{normalize}}$

0.13
0.87
0.00

Probabilities must add to 1

Only when $S_i(y_i) = 1$, L_i will be 0.

1
0
0

The ground truth probability

SVM Loss

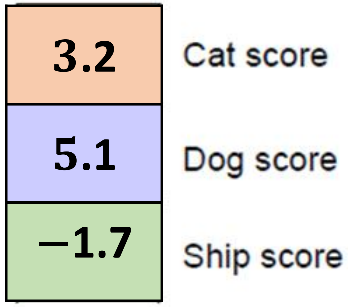
- Instead of normalizing the original scores to probabilities
- It requires the score of the true category to be the largest
- So a loss function can be defined as (for a cat training image):

max(0, Dog Score + 1 – Cat Score) + max(0, Ship Score + 1 – Cat Score)

make sure the cat score is larger for a sufficient margin

- Formally, the loss is:

$$L_i(y_i, s_i) = \sum_{j \neq y_i} max(0, s_i(j) + 1 - s_i(y_i))$$



max(0, 5.1+1 -3.2) + max(0, -1.7 + 1 – 3.2)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0 = 2.9

Sum Over All Samples

- The final loss is summed over all training data

$$\sum_i L_i(y_i, s_i)$$

- An example: suppose we have the following class scores $s_i = Wx_i$

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Evaluate the loss (say the SVM loss) for each sample

$$L_i = \sum_{j \neq y_i} \max(0, s_i(j) + 1 - s_i(y_i))$$

And sum over all samples

Cat: max(0, 5.1 + 1 - 3.2) + max(0, -1.7 + 1 - 3.2) = 2.9

Car: max(0, 1.3 + 1 – 4.9) + max(0, 2.0 + 1 – 4.9) = 0

Frog: max(0, 2.2 + 1 – (-3.1)) + max(0, 2.5 + 1 – (-3.1)) = 12.9

Total Loss = 2.9 + 0 + 12.9 = 15.8

Regularizer

- Suppose a coefficient W makes all loss to be 0
- Is W unique? Usually no!
  - For example, the SVM loss, comparing the relative scores of different classes
  - If we scale W, scores of all classes will be scaled the same way
  - If W makes $L = 0$, $2W$ also makes $L = 0$
- The optimization algorithm will waggle between solutions
  - We need regularizers to favor some solution than the others
  - So that the optimizer won't be confused

Regularizer

$$W^* = \underset{W}{\arg\min} \sum_i \underbrace{L_i(y_i, s_i)}_{\text{data loss}} + \underbrace{\lambda}_{\text{regularization strength}} \underbrace{R(W)}_{\text{regularizer}}$$

- Some common regularizers:
  - L2 regularization: $R(W) = \sum_{k,l} W_{k,l}^2$
  - L1 regularization: $R(W) = \sum_{k,l} |W_{k,l}|$

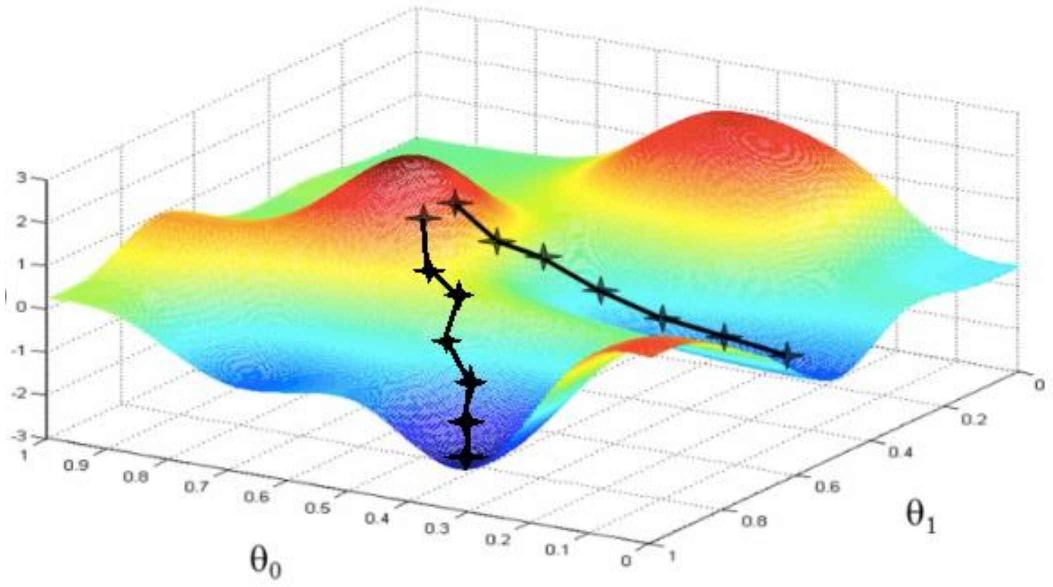
Questions?

Parameter Estimation

- Estimate parameters from training data

$$W^* = \arg\min_{W} \sum_i L_i(y_i, s_i) + \lambda R(W)$$

- Energy minimization by gradient descendent

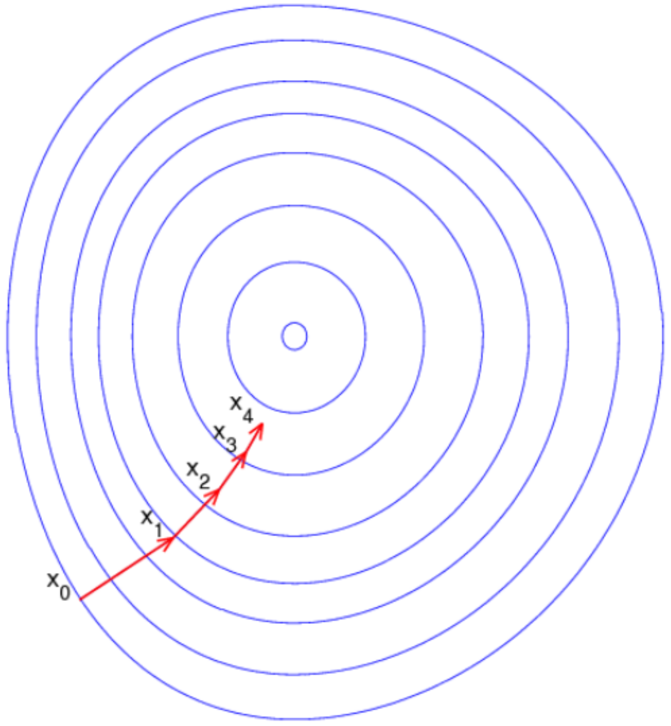


Gradient Descendent

- Start from some initial parameters W_0
- Iterate the following two steps:
  - Compute the gradient vector $\frac{\partial L}{\partial W}$
  - Update the parameters

$$W_{n+1} = W_n + \alpha \frac{\partial L}{\partial W}$$

α is called learning rate



Gradient of Multi-Variable Functions

- In 1D, the derivative of a function is:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- In multiple dimensions, the gradient is a vector of partial derivatives along each dimension

$$\frac{\partial f}{\partial \boldsymbol{x}} = \begin{pmatrix} df/dx_1 \\ df/dx_2 \\ \vdots \\ df/dx_n \end{pmatrix}$$

Stochastic Gradient Descendent (SGD)

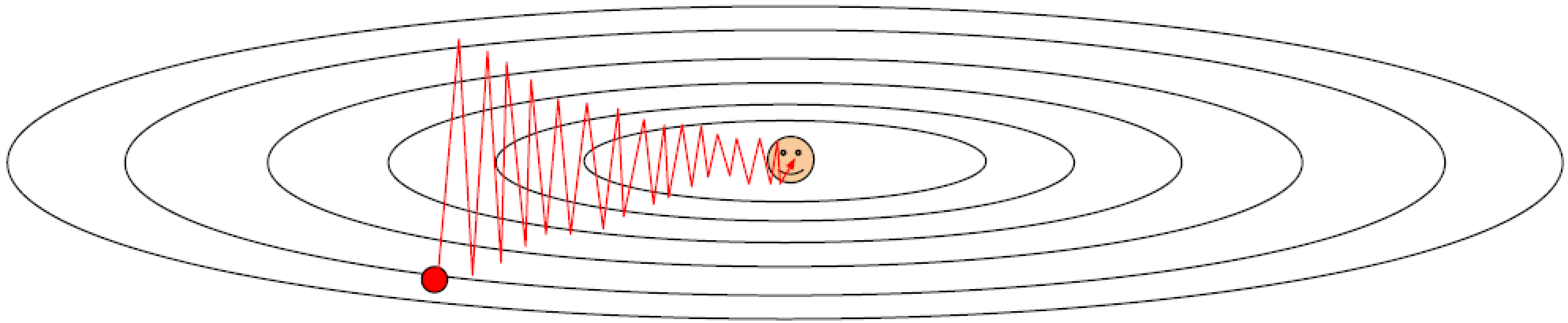
- Evaluating the Loss (and gradient) over all training data is expensive

$$W^* = \arg\min_{\mathrm{W}} \sum_i L_i(y_i, s_i) + \lambda R(W)$$

- Approximate sum using a mini-batch of training data, e.g. 32/64/128

Problems in SGD

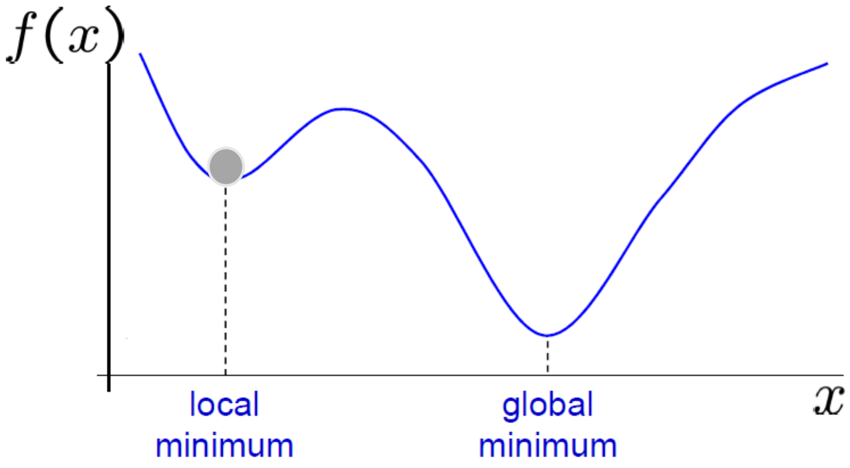
- What if the loss function changes quickly in one dimension but slowly in another?
- What does gradient descent do?
- Very slow progress along shallow dimension, jitter along steep direction

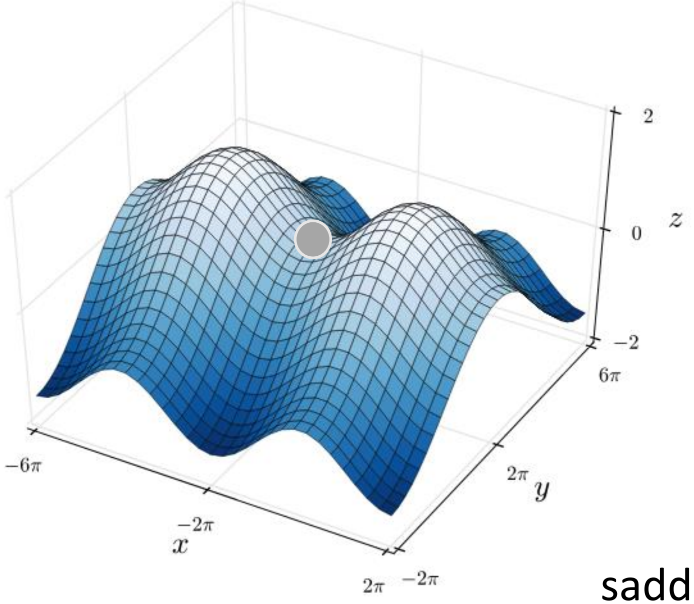


The gradient direction is always perpendicular to the local contours

Problems in SGD

- What if the loss function has a local minima or saddle point?

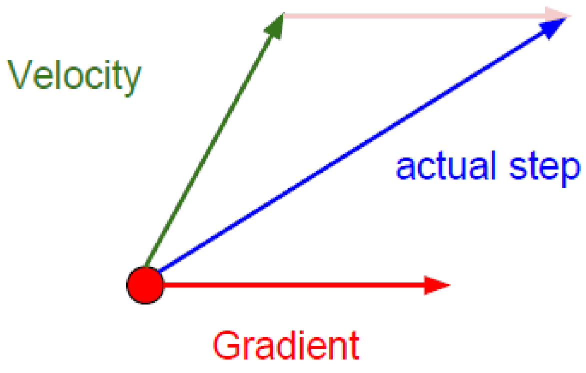




saddle point

Momentum

- Build up velocity as a running mean of gradient
- Compute the update vector as combination of gradient and velocity



$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

Recap: Learning Parametric Classifiers

- Choose a parametric function $s_i = f(x_i, W)$
- Choose a loss function $L(y_i, s_i)$
- Minimize the loss over all training data to learn W

$$W^* = \arg\min_{\mathrm{W}} \sum_i L_i(y_i, s_i)$$

- Regularizers are helpful
- Minimize by gradient descendent
- Use stochastic gradient descendent
- Use momentum

Questions?