Standard **Bag-of-Words** Pipeline (for image classification)

Dictionary Learning:
Learn Visual Words using clustering

Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train and test data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

K-Means Clustering

K-means clustering alternates between two steps:

1. Assume the cluster centers are known (fixed). Assign each point to the closest cluster center.

2. Assume the assignment of points to clusters is known (fixed). Compute the best center for each cluster, as the mean of the points assigned to the cluster.

The algorithm is initialized by choosing K random cluster centers

K-means converges to a local minimum of the objective function

— Results are initialization dependent

Expectation Maximization

A simpler version

The K-Means centers

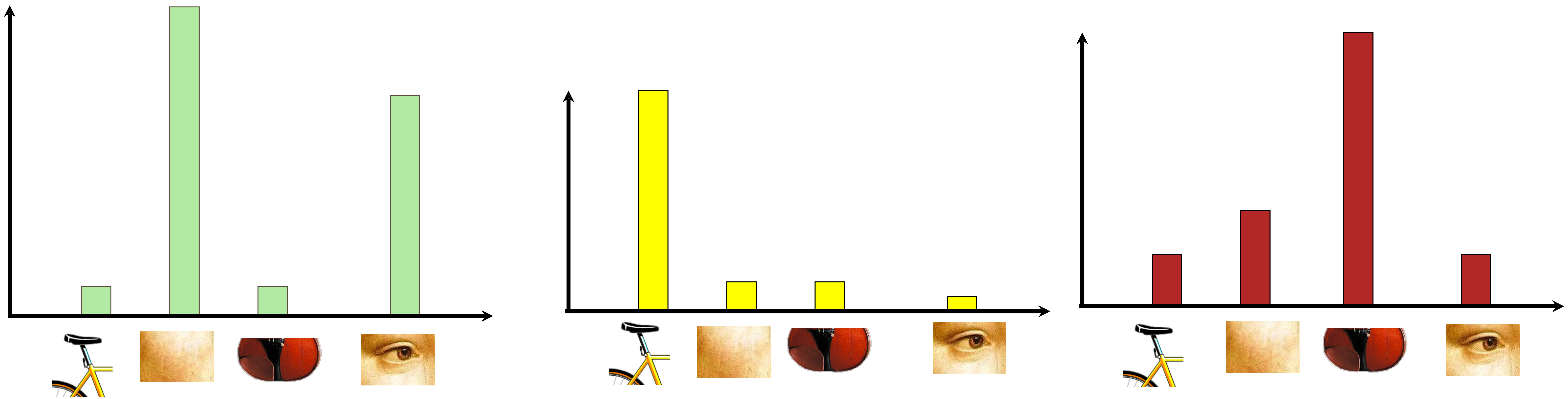
Given a model repeat

1. Create an “**expectation**” of the (log-)likelihood with the current hypothesis
2. Update the hypothesis to one that **maximizes** the expectation above

Not exactly the *hard* assignments of K-Means

2. Encode: build Bag-of-Words (BOW) vectors for each image

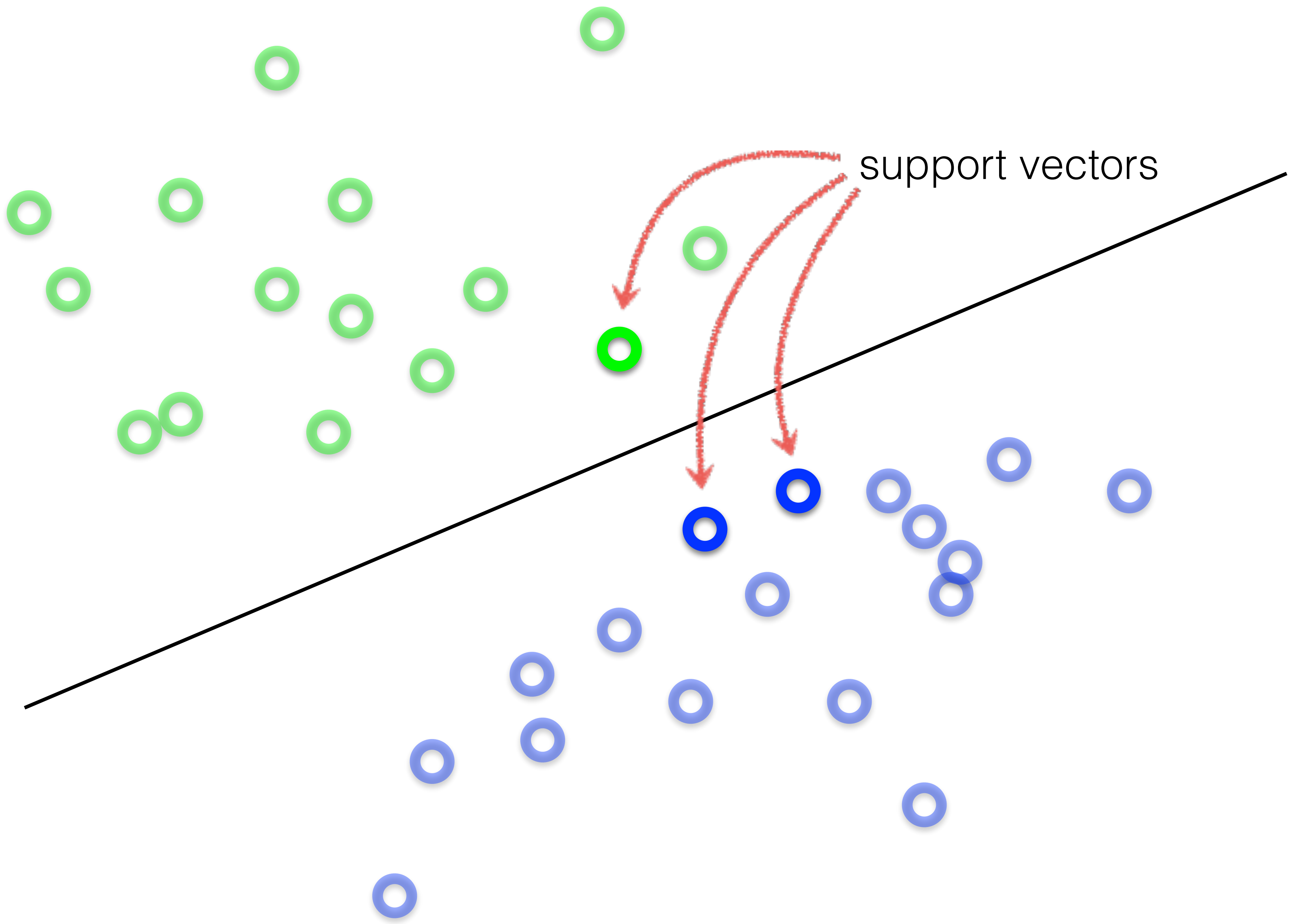
2. **Histogram**: count the number of visual word occurrences



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Support Vector Machines (SVM)

What's the best **w** ?

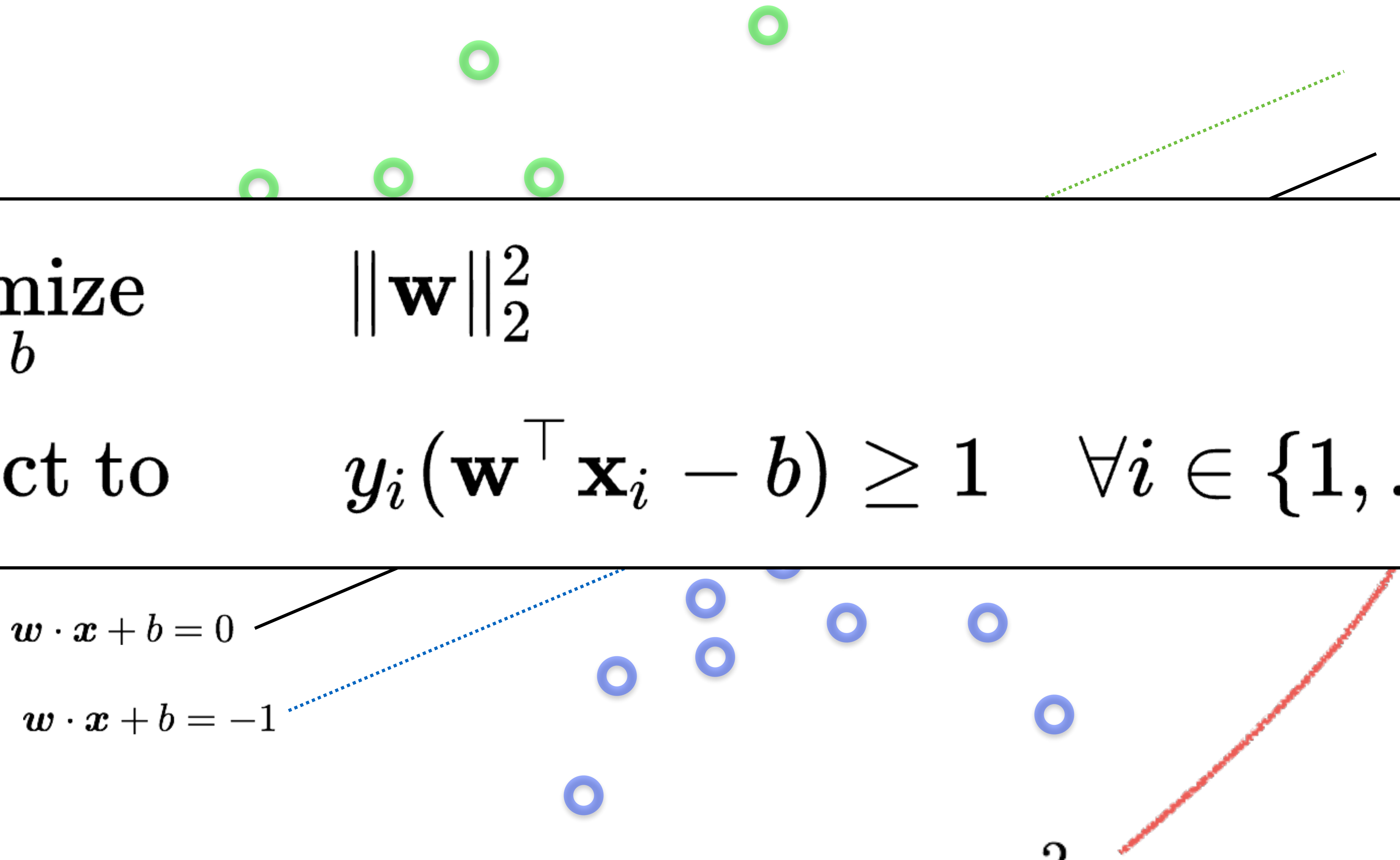


Want a hyperplane that is far away from 'inner points'

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Support Vector Machines (SVM)

Find hyperplane **w** such that …



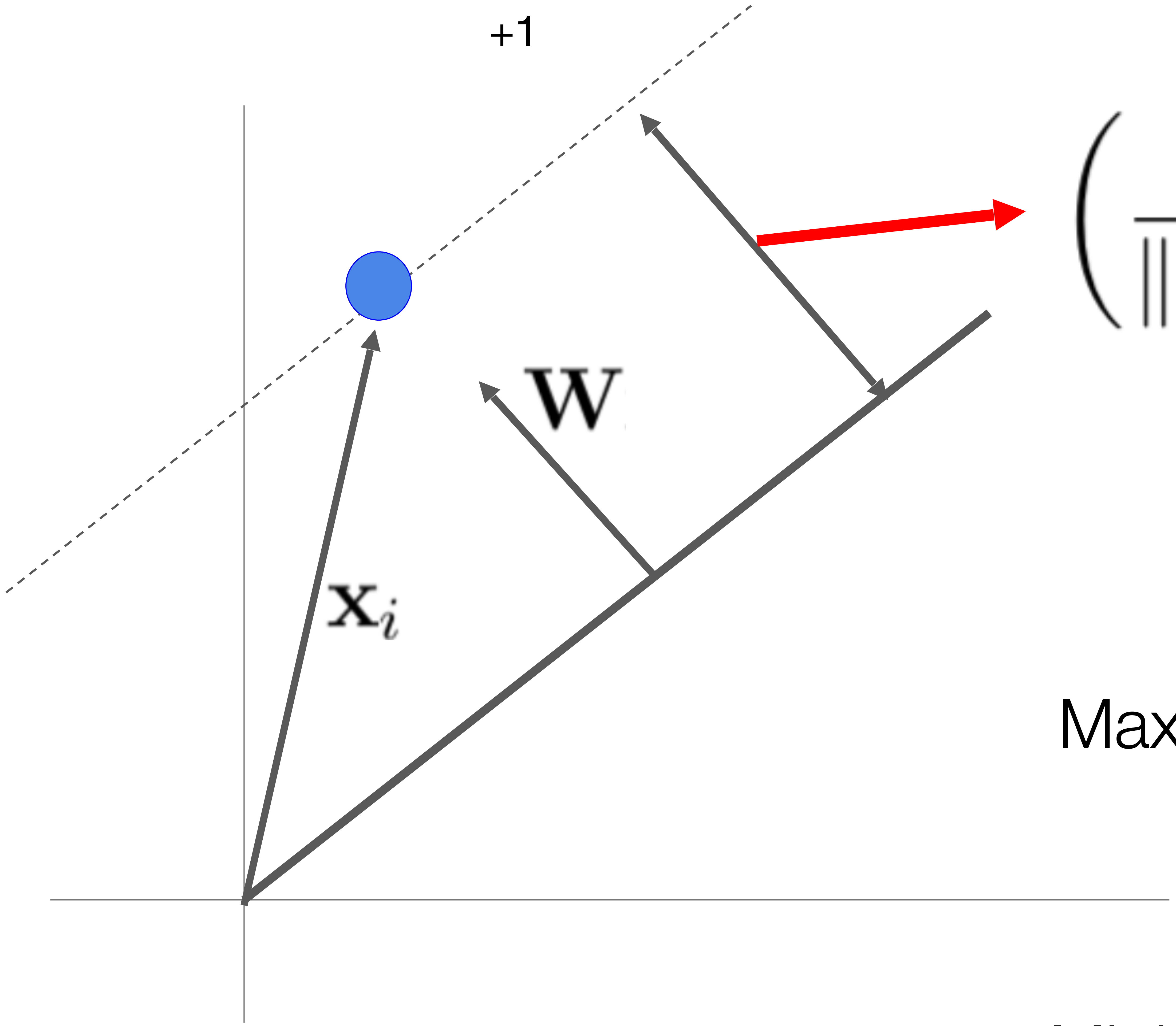
$$\underset{\mathbf{w},\ b}{\text{minimize}} \quad \|\mathbf{w}\|_2^2$$

$$\text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i \in \{1, \ldots, n\}$$

the gap between parallel hyperplanes $\frac{2}{\|\boldsymbol{w}\|}$ is maximized

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Distance to the **border**



Becomes 1 because it's the thing at the border (+1)

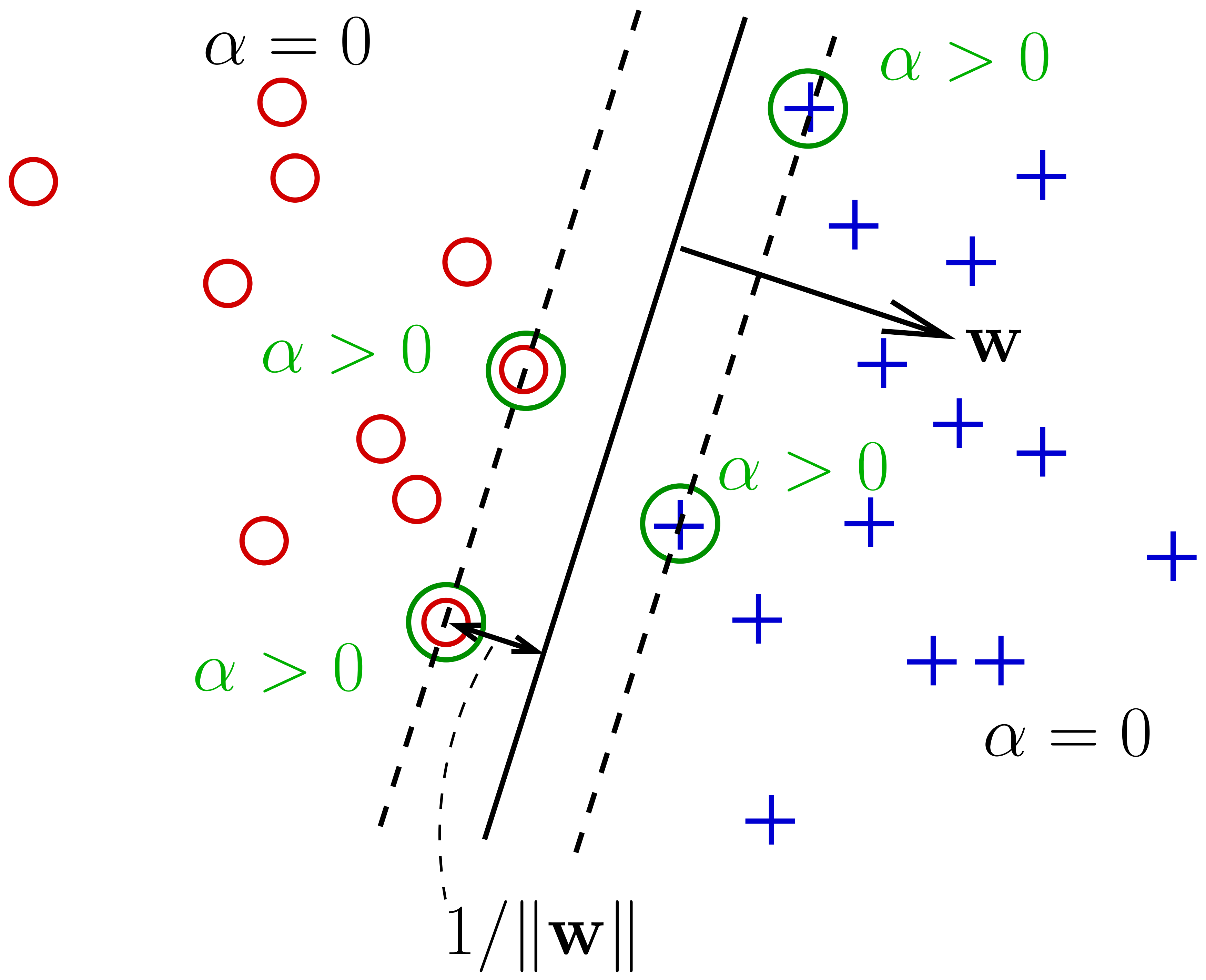
$$\left(\frac{\mathbf{W}}{\|\mathbf{W}\|_2}\right)^\top \mathbf{x}_i = \frac{1}{\|\mathbf{W}\|_2}$$

Maximize $\frac{1}{\|\mathbf{W}\|_2^2}$

Minimize $\|\mathbf{W}\|_2^2$

Support Vectors

- The active constraints are due to the data that define the classification boundary, these are called **support vectors**



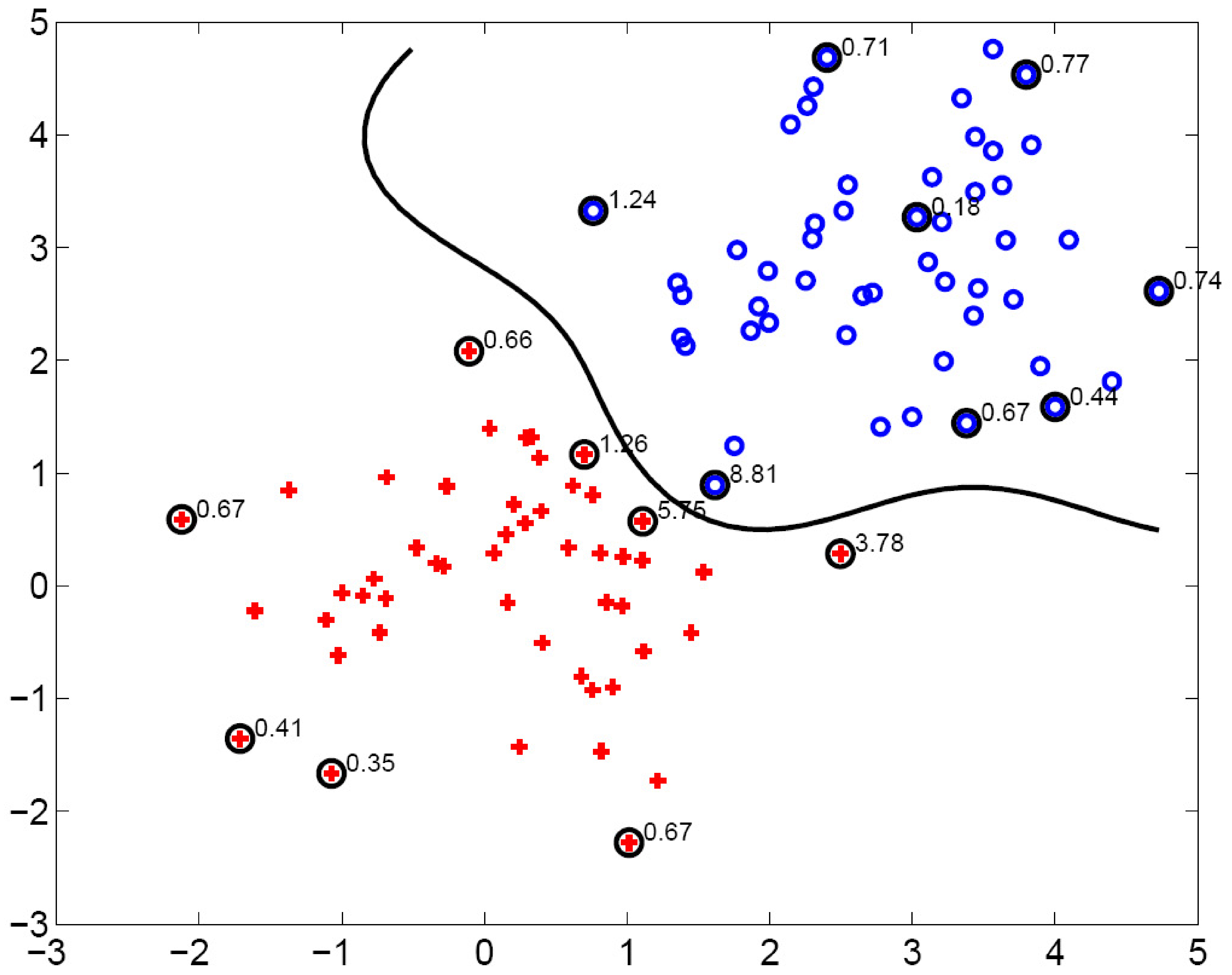
Final classifier can be written in terms of the support vectors:

$$\hat{y} = \text{sign}\left(\hat{w}_0 + \sum_{\alpha_i > 0} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}\right)$$

Non-Linear SVM

- Replace inner product with kernel

$$\mathbf{x}_i^T \mathbf{x} \rightarrow \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) \rightarrow k(\mathbf{x}_i, \mathbf{x})$$



- Data are (ideally) linearly separable in ϕ(x)
- But we don't need to know ϕ(x), we just specify k(x,y)
- Points with α>0 (circled) are support vectors
- Other data can be removed without affecting classifier

Bag-of-Words Representation

Algorithm:

Initialize an empty K -bin histogram, where K is the number of codewords
Extract local descriptors (e.g. SIFT) from the image
For each local descriptor **x**

Map (Quantize) **x** to its closest codeword → **c**(**x**)

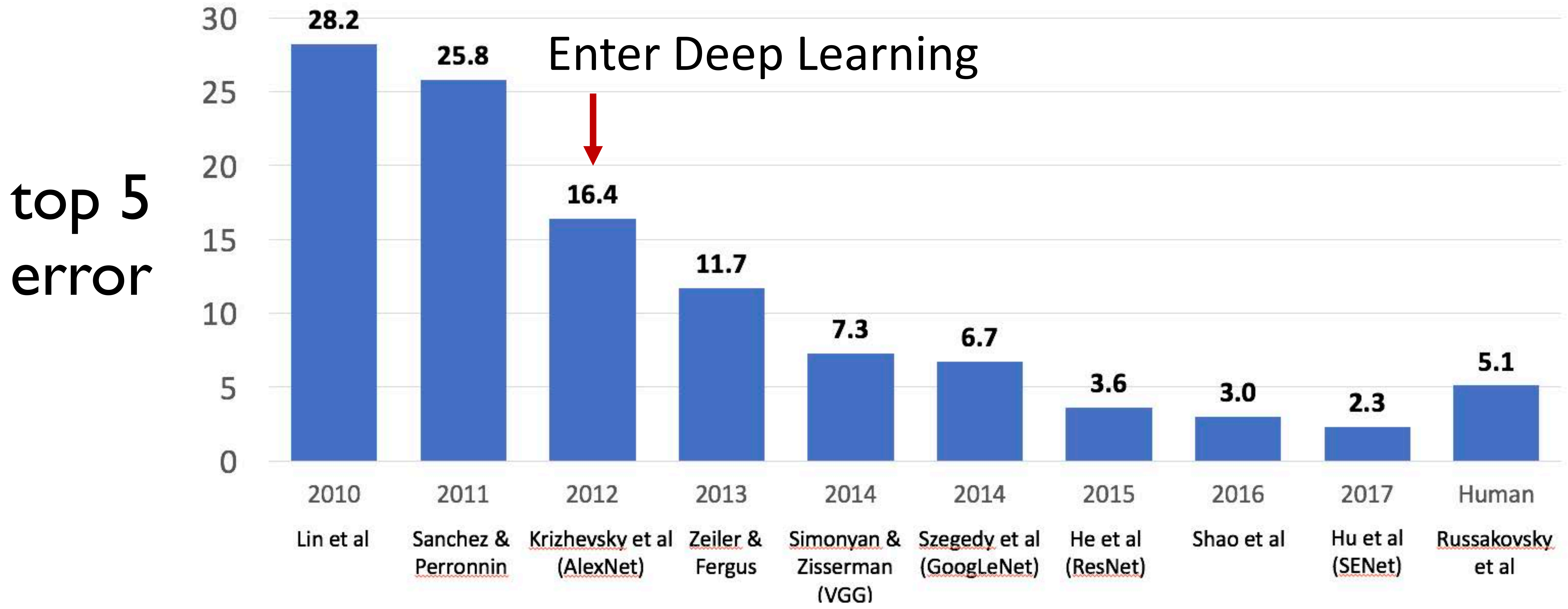
Increment the histogram bin for **c**(**x**)

Return histogram

We can then classify the histogram using a trained classifier, e.g. a support vector machine or k-Nearest Neighbor classifier

Alexnet

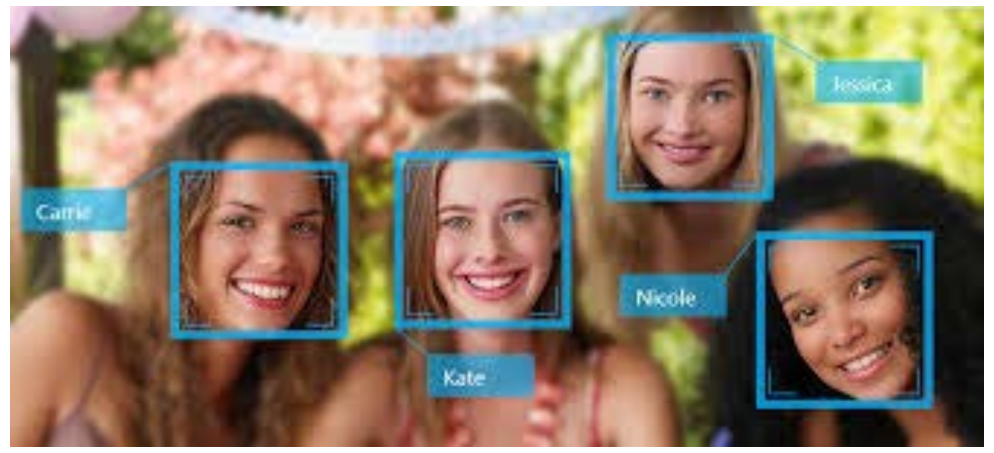
- Won the Imagenet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 by a large margin
- Some ingredients: Deep neural net (Alexnet), Large dataset (Imagenet), Lots of compute (2 GPU weeks), non-saturating activation functions (ReLU)

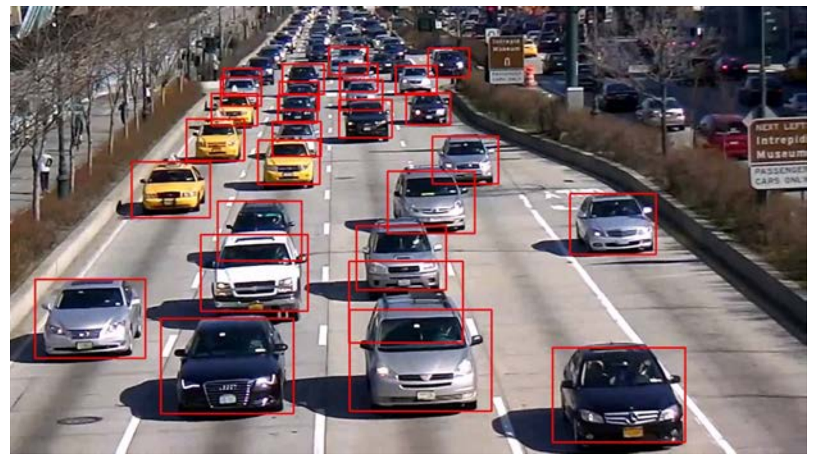


[J. Johnson]

THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision






Lecture 19: Visual Classification 2, Linear Classification

Menu for Today

Topics:

- **Linear Classification**
- Nearest Neighbour, nearest mean
- Bayesian classification

Readings:

- **Today's** Lecture: Szeliski 11.4, 12.3-12.4, 9.3, 5.1-5.2

Reminders:

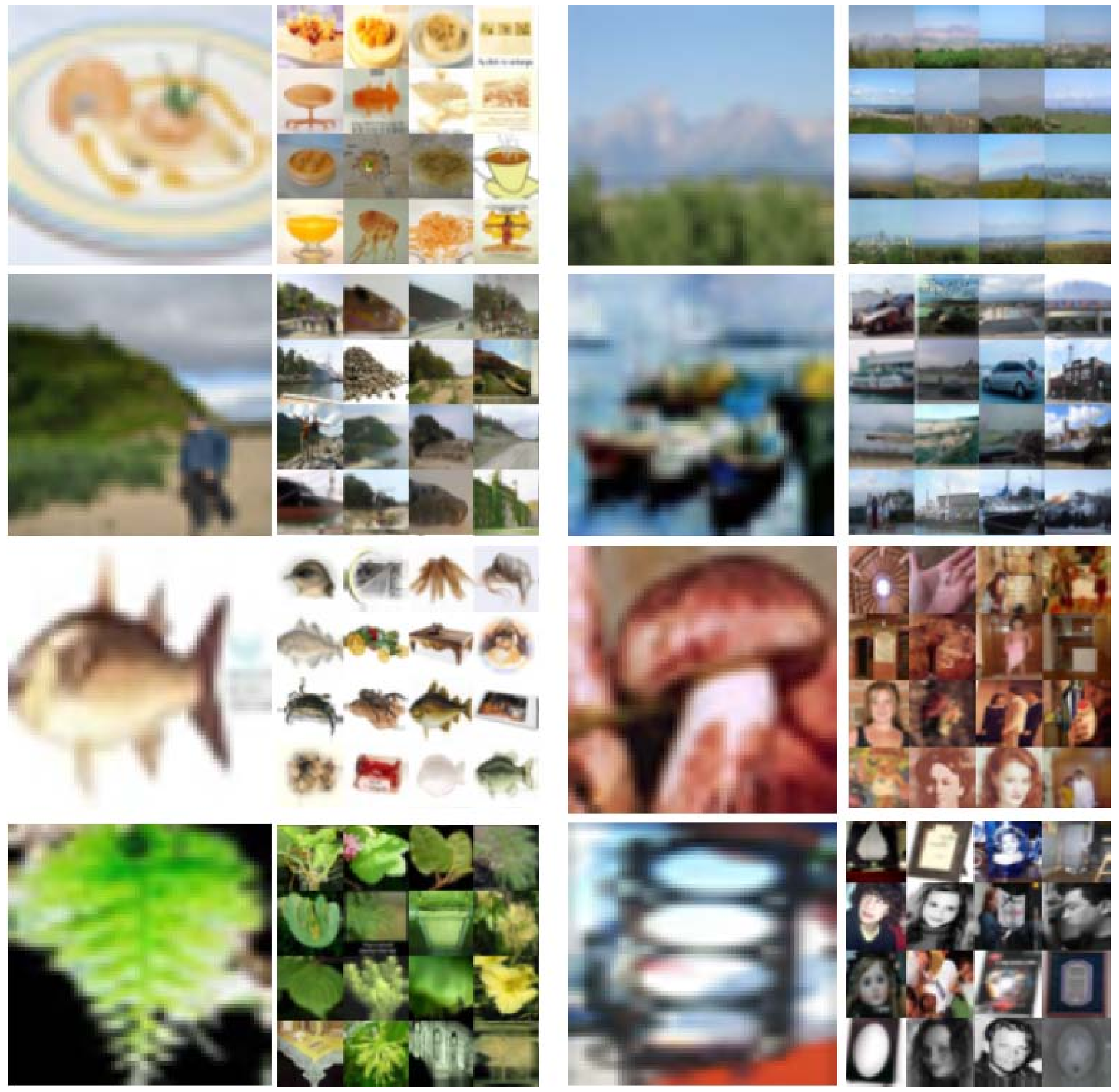
- **Assignment 5**: Scene Recognition with Bag of Words due **Apr 3rd**

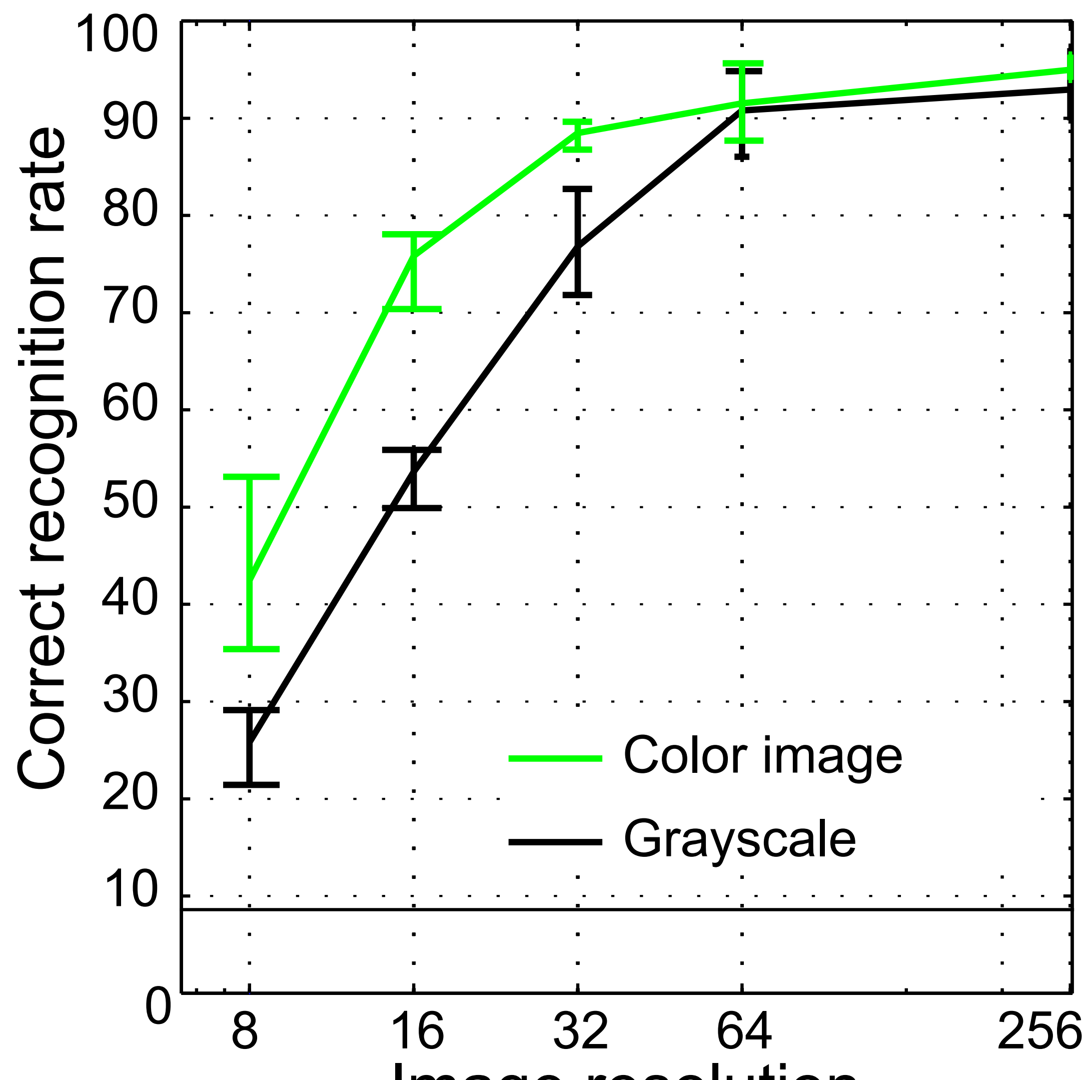
Visual Classification 2

- Linear classification, CIFAR10 case study
- Nearest neighbour, nearest mean
- Bayesian Classification, Gaussian distributions, priors

Tiny Image Dataset

- Precursor to ImageNet and CIFAR10/100
- 80 million images collected via image search using 75,062 noun synsets from WordNet (labels are noisy)
- Very small images (32x32xRGB) used to minimise storage
- Note human performance is still quite good at this scale!



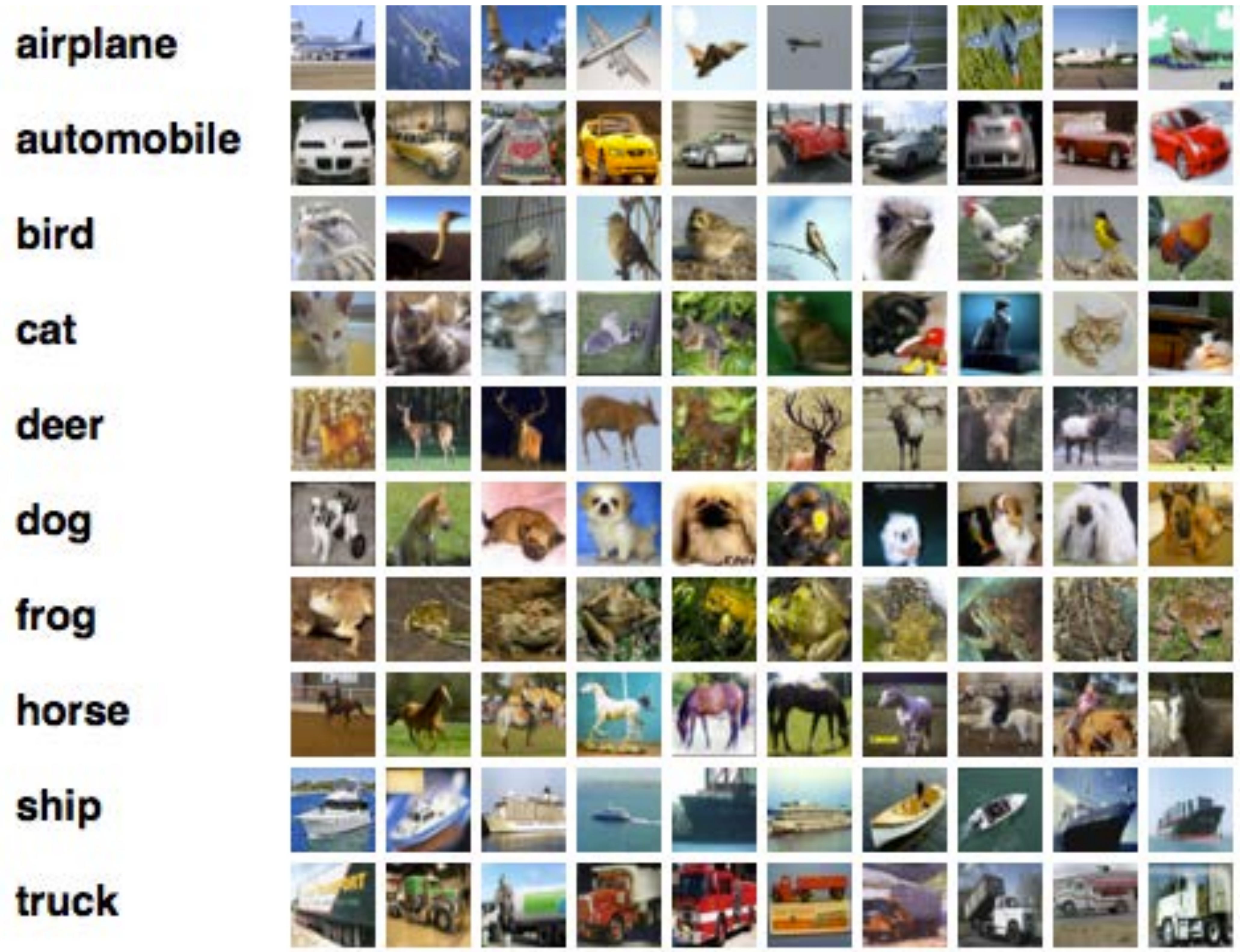


a) Scene recognition

[Torralba Freeman Fergus 2008]

CIFAR10 Dataset

- Hand labelled set of 10 categories from Tiny Images dataset
- 60,000 32x32 images in 10 classes (50k train, 10k test)

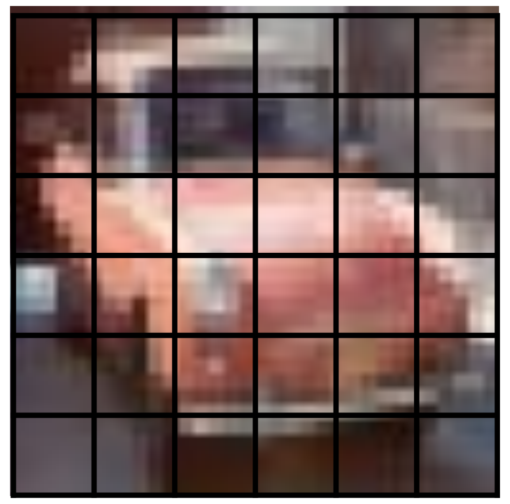


Good test set for visual recognition problems

CIFAR10 Classification

- Let's build an image classifier!

- Start by vectorizing the image data



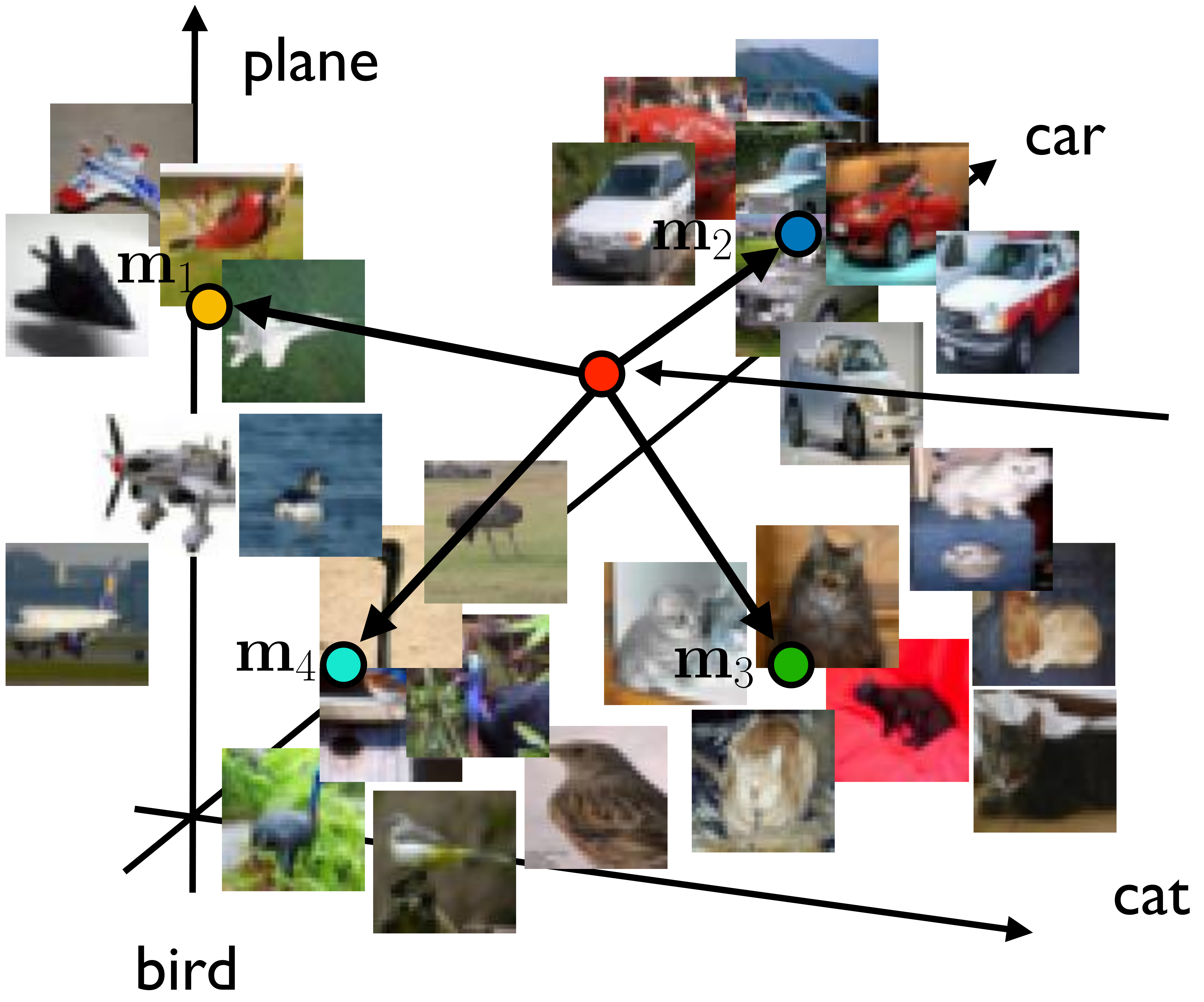
32 x 32 x RGB (8 bit) image →

x = [65 102 33 57 54 …]

- x = 3072 element vector of 0-255
- Note this throws away spatial structure, we'll bring it back later when we look at feature extraction and CNNs

Nearest Mean Classification

- How about a single template per class



Query:

$$\mathbf{x}_q =$$

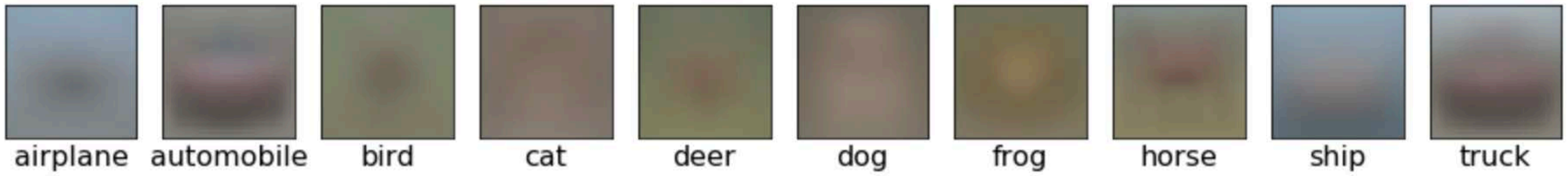
$$c_q = \ ?$$

Nearest Mean Classification

- Find nearest mean and assign class

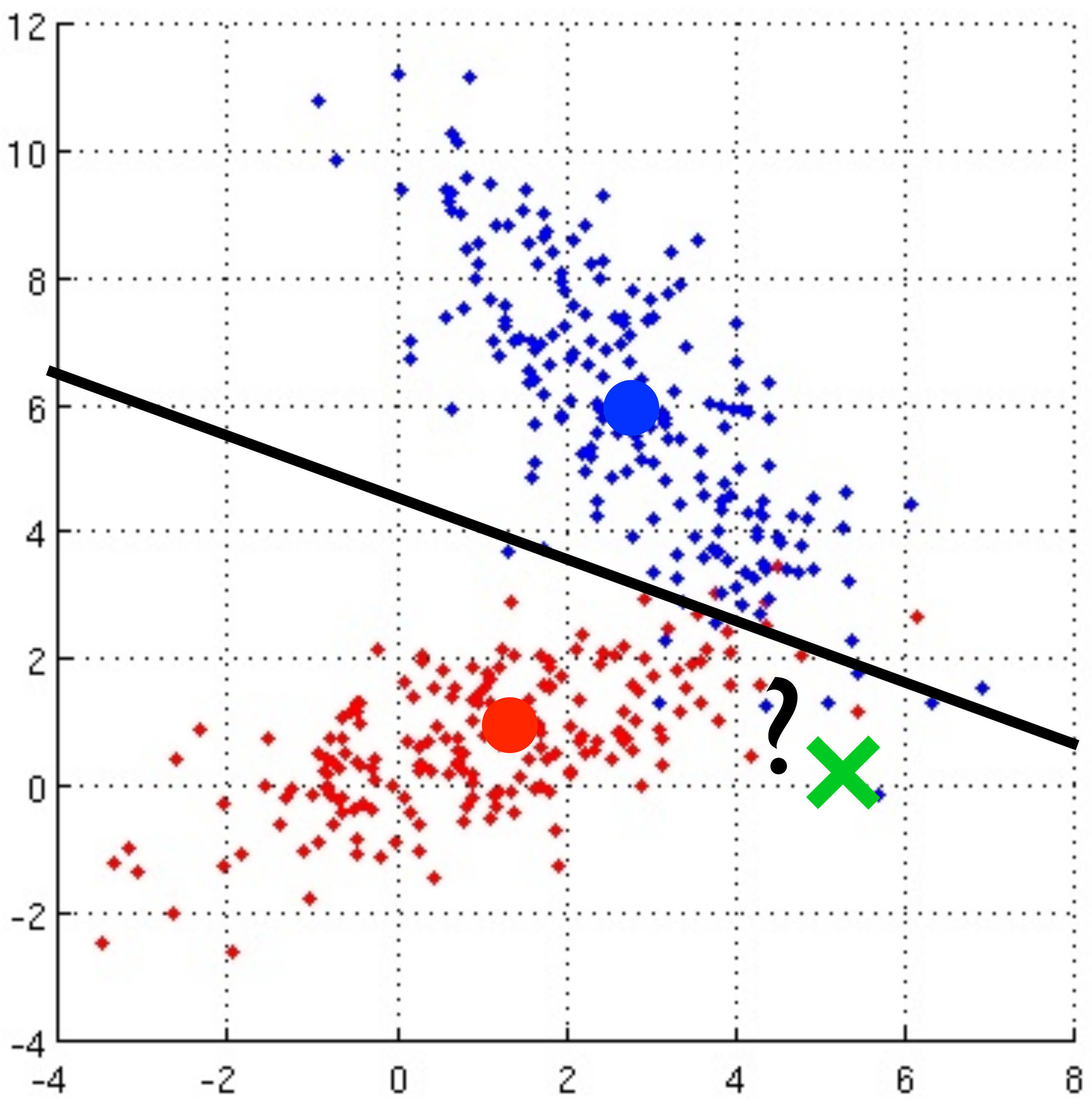
$$c_q = \arg\min_i |\mathbf{x}_q - \mathbf{m}_i|^2$$

- CIFAR 10 class means



Nearest Mean Classifier

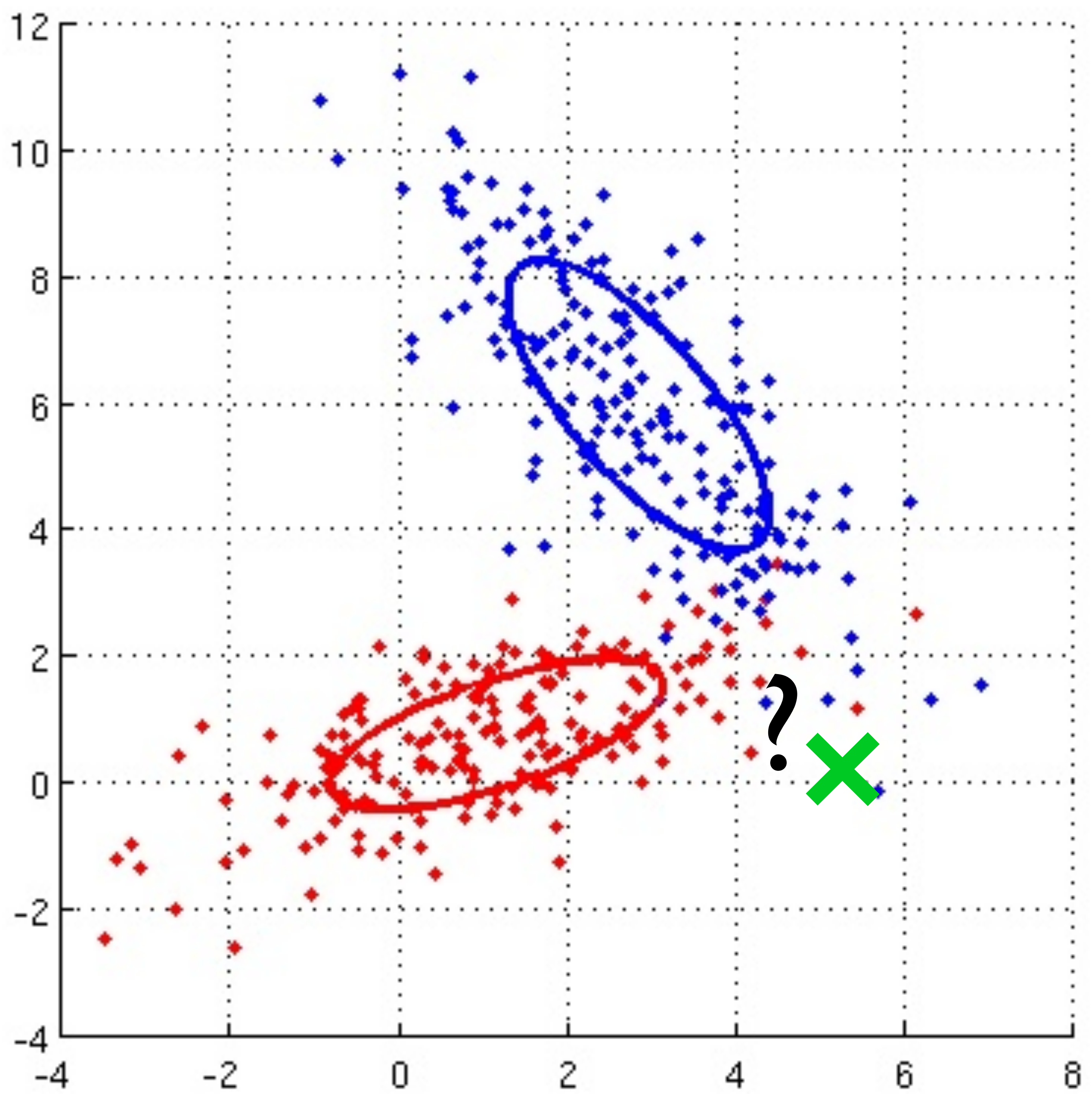
- Suppose we have 2 classes of 2-dimensional data that are not linearly separable



- A simple approach could be to assign to the class of the nearest mean
- Can we do better if we know about the data distribution?

Bayesian Classificaion

- A probabilistic view of classification models the likelihood of observing the data given a class/parameters

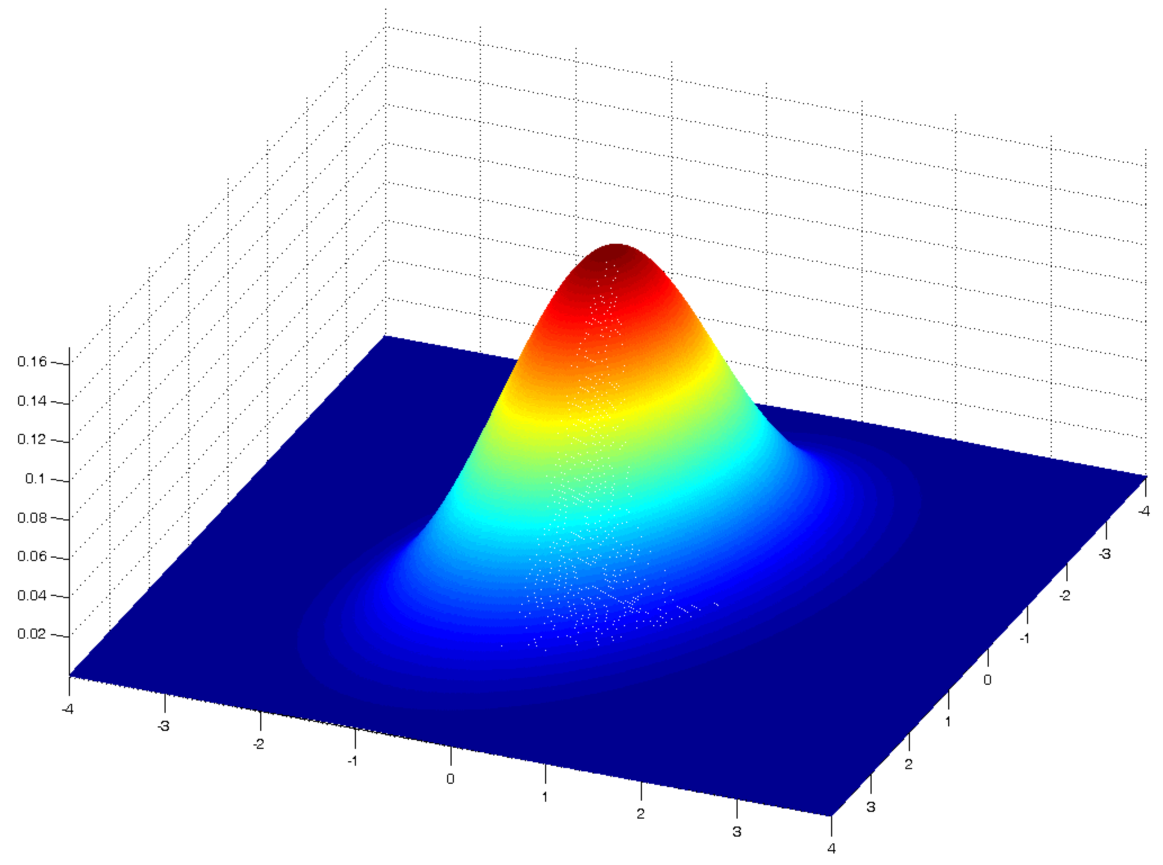


e.g., we might assume that the distribution of data given the class is Gaussian

Multi-dimensional Gaussian

- The Gaussian probability density is given by

$$p(\mathbf{x}|\mathbf{m}, \mathbf{\Sigma}) = \frac{1}{|2\pi\mathbf{\Sigma}|^{\frac{1}{2}}} \exp -\frac{1}{2}(\mathbf{x}-\mathbf{m})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{m})$$



- To estimate from data (x)

$$\hat{\mathbf{m}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$$

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

- These estimates maximise the probability of the data x given parameters m, Σ

2-Class Gaussian Classifier

- Simple classification rule: choose class #1 if

$$p(\mathbf{x}|c_1) > p(\mathbf{x}|c_2)$$

- taking -2 x ln of both sides (reverses sign)

$$-2\ln p(\mathbf{x}|c_1) < -2\ln p(\mathbf{x}|c_2)$$

- negative log of Gaussian density

$$-2\ln p(\mathbf{x}) = -2\ln \frac{1}{|2\pi\mathbf{\Sigma}|^{\frac{1}{2}}} \exp -\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{m})$$

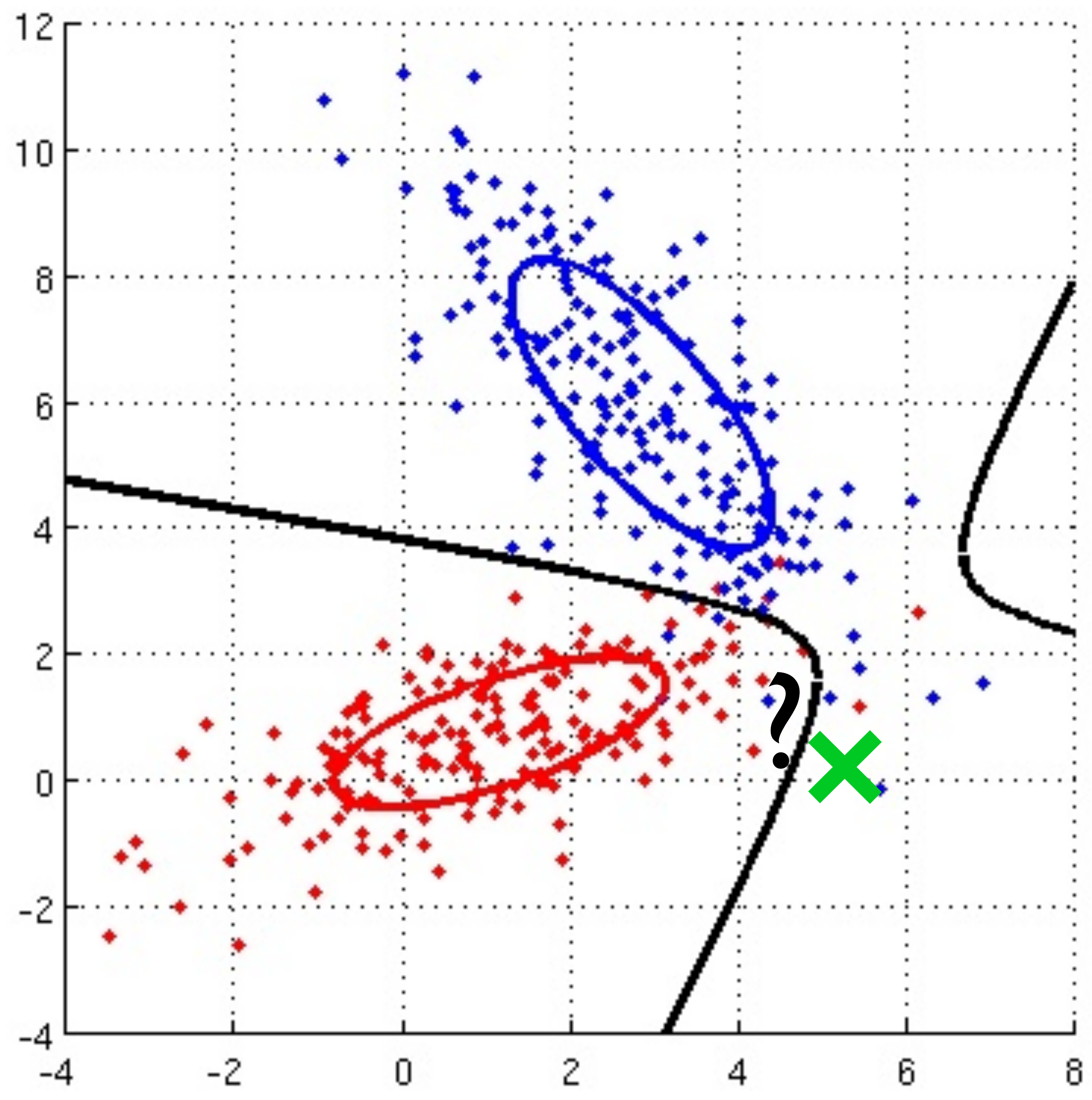
$$= \ln(2\pi^d) + \ln|\mathbf{\Sigma}| + (\mathbf{x}-\mathbf{m}^T)\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{m})$$

- decision rule becomes (class #1 if...)

$$\ln\mathbf{\Sigma}_1 + (\mathbf{x}-\mathbf{m}_1)^T\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\mathbf{m}_1) < \ln\mathbf{\Sigma}_2 + (\mathbf{x}-\mathbf{m}_2)^T\mathbf{\Sigma}_2^{-1}(\mathbf{x}-\mathbf{m}_2)$$

2-Class Gaussian Classifier

- Suppose we've modelled our 2 classes with Gaussian distributions



$$p(\mathbf{x}|c_1) = N(\mathbf{x}; \mathbf{m}_1, \mathbf{\Sigma}_1)$$
$$p(\mathbf{x}|c_2) = N(\mathbf{x}; \mathbf{m}_2, \mathbf{\Sigma}_2)$$

- Our decision rule, class #1 if

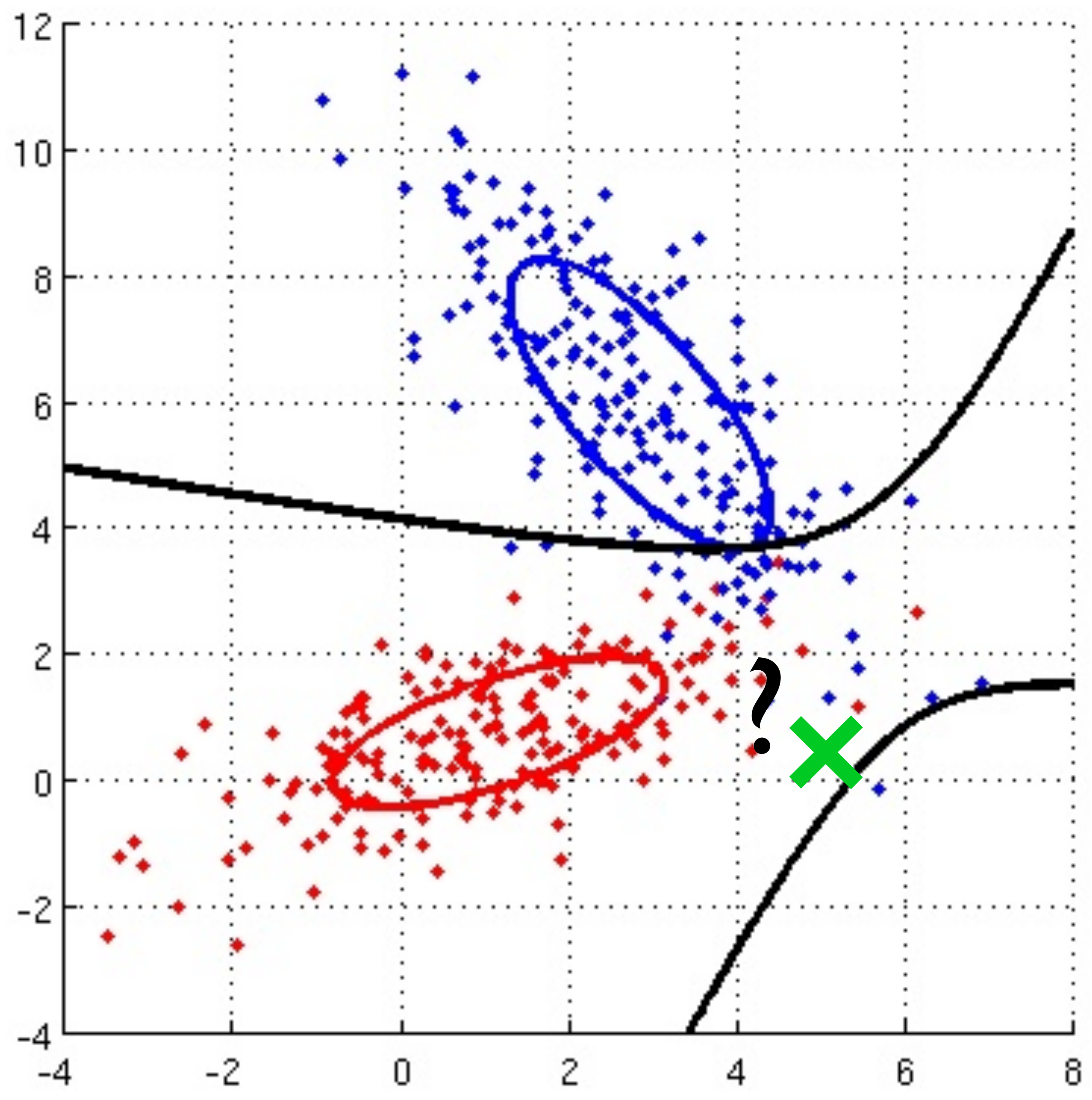
$$p(\mathbf{x}|c_1) > p(\mathbf{x}|c_2)$$

is called a maximum likelihood classifier

Incorporating Prior Knowledge

- What if red is more common than blue?
- Weight each likelihood by prior probabilities $p(c_1), p(c_2)$
- Decision rule (MAP classifier) choose class #1 if:

$$p(\mathbf{x}|c_1)p(c_1) > p(\mathbf{x}|c_2)p(c_2)$$



$p(c_1) = 0.$[illegible]

$p(c_2) = 0.$[illegible]

Nearest Neighbor Classifier

Given a new data point, assign the label of nearest training example in feature space.

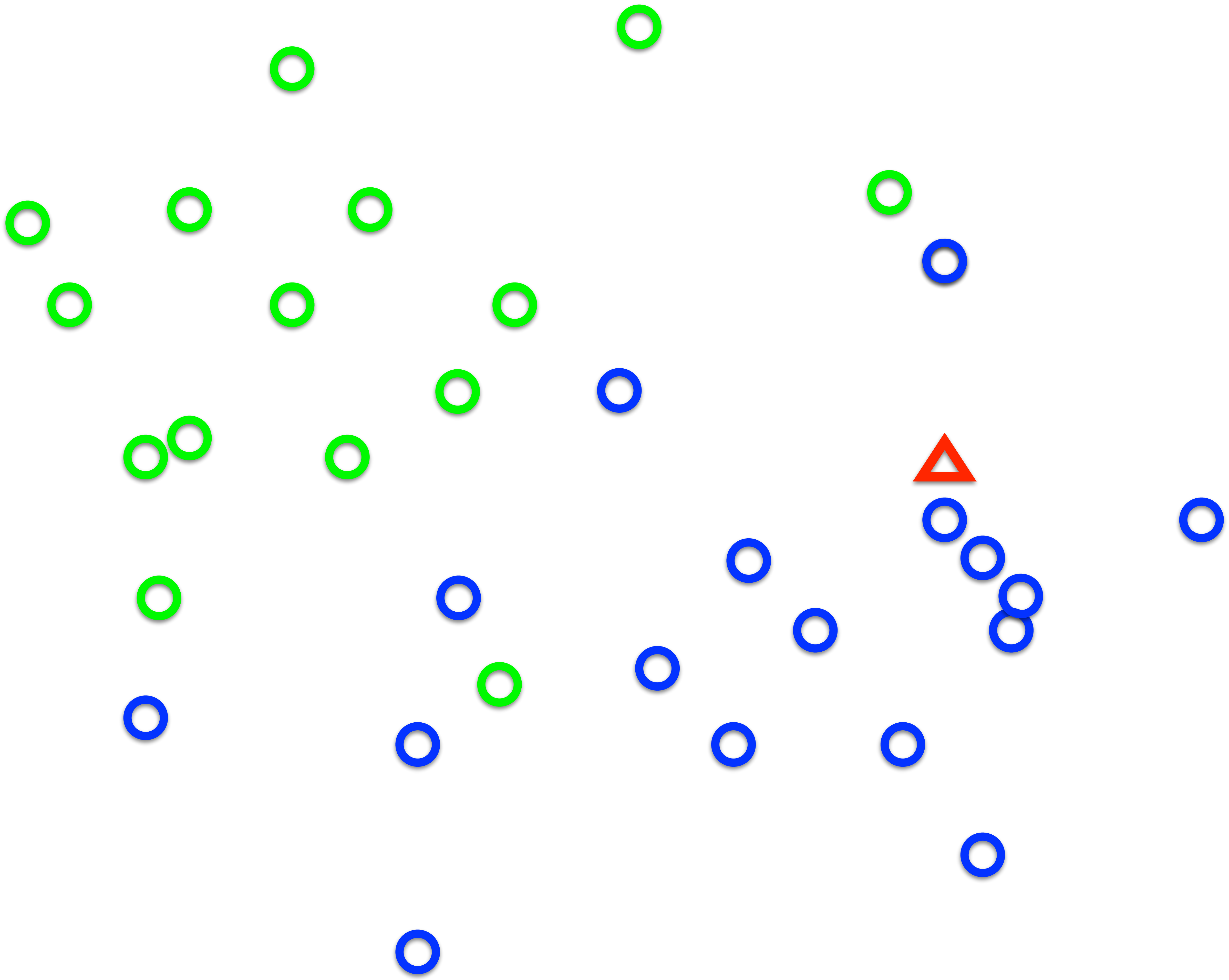


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Nearest Neighbor Classifier

Given a new data point, assign the label of nearest training example in feature space.

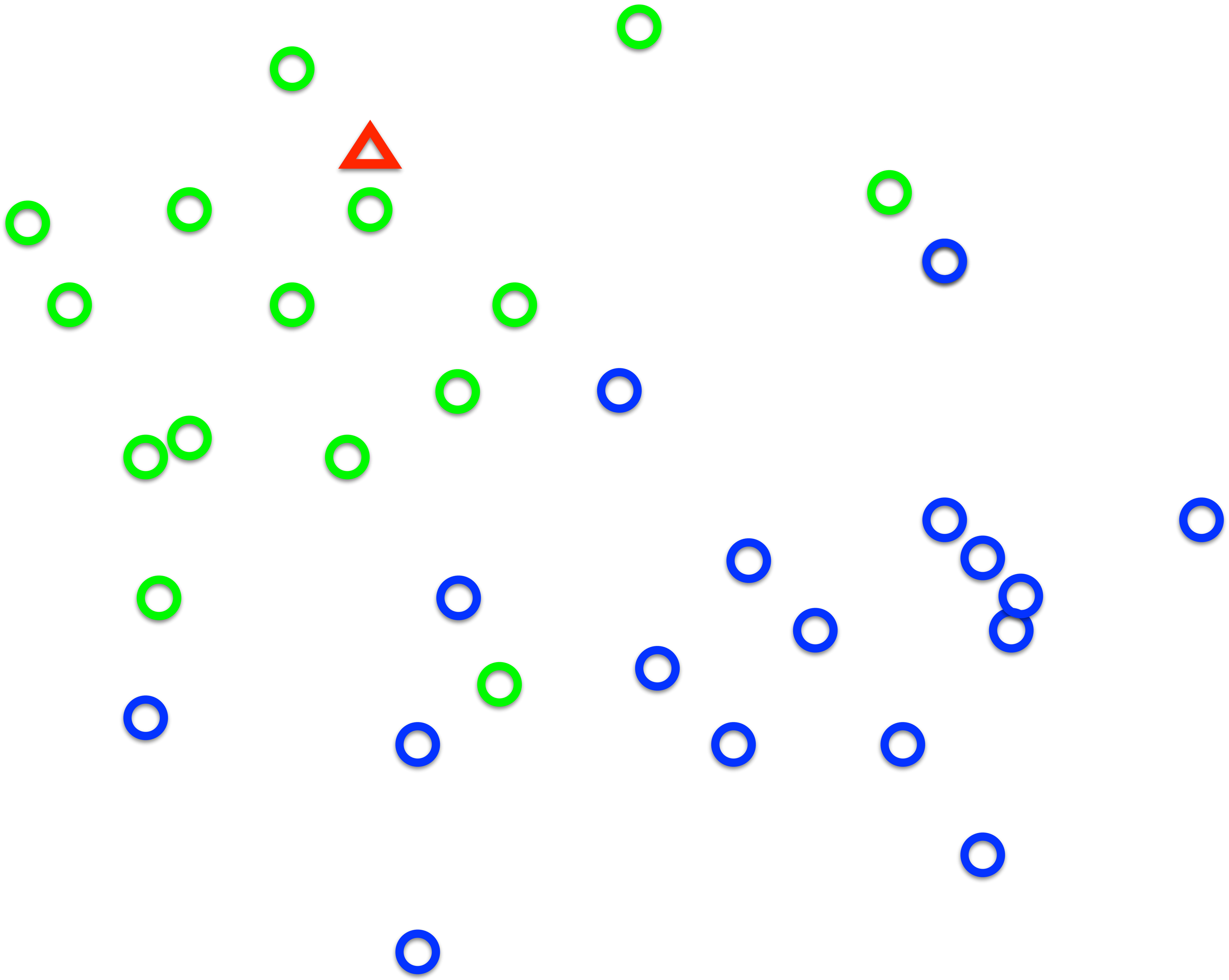


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Nearest Neighbour Classification

- Find nearest neighbour in training set

$$i_{NN} = \arg\min_i |\mathbf{x}_q - \mathbf{x}_i|$$

- Assign class to class of the nearest neighbour

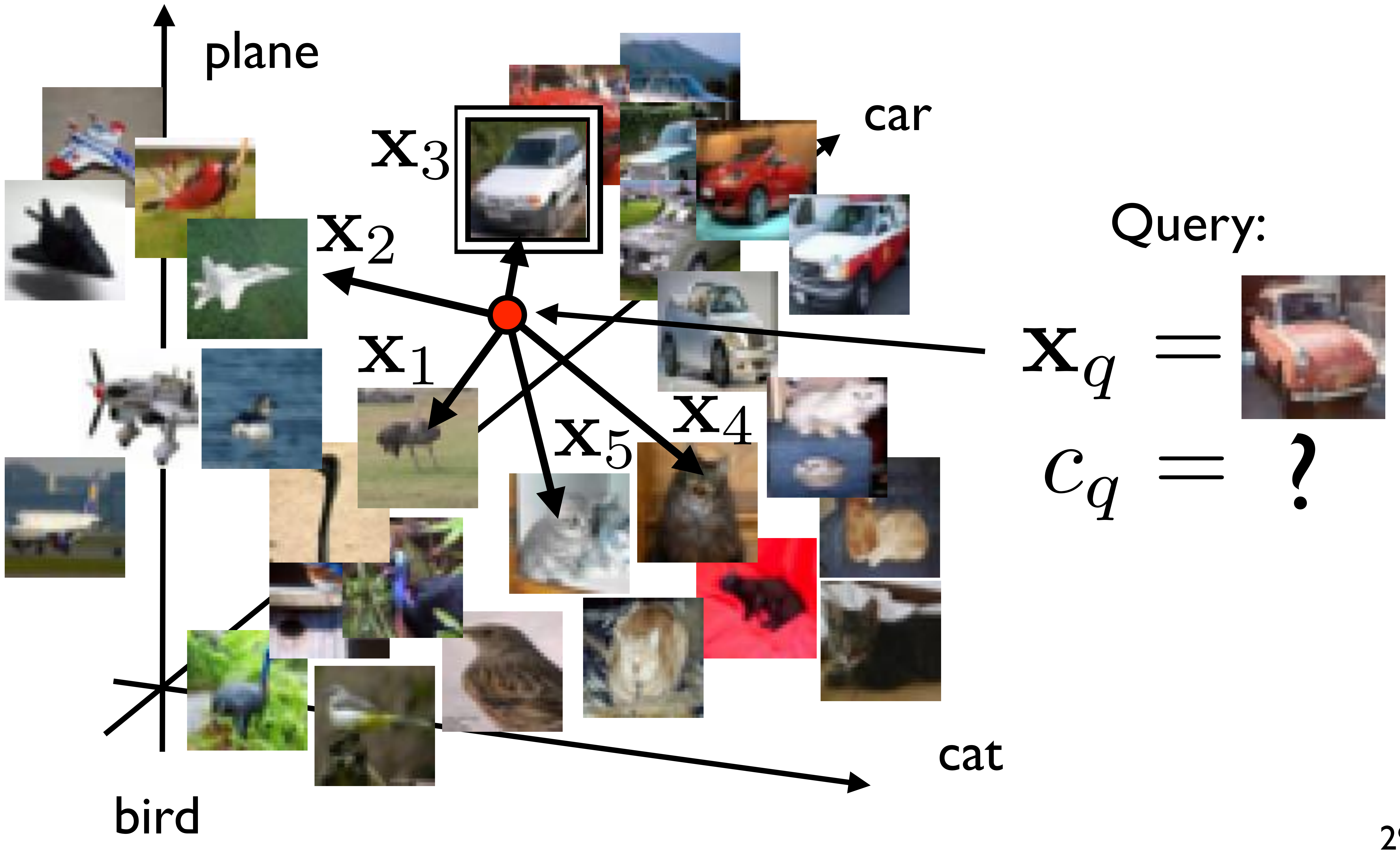
$$\hat{y}(\mathbf{x}_q) = y(\mathbf{x}_{i_{NN}})$$

Query $\mathbf{x}_q$

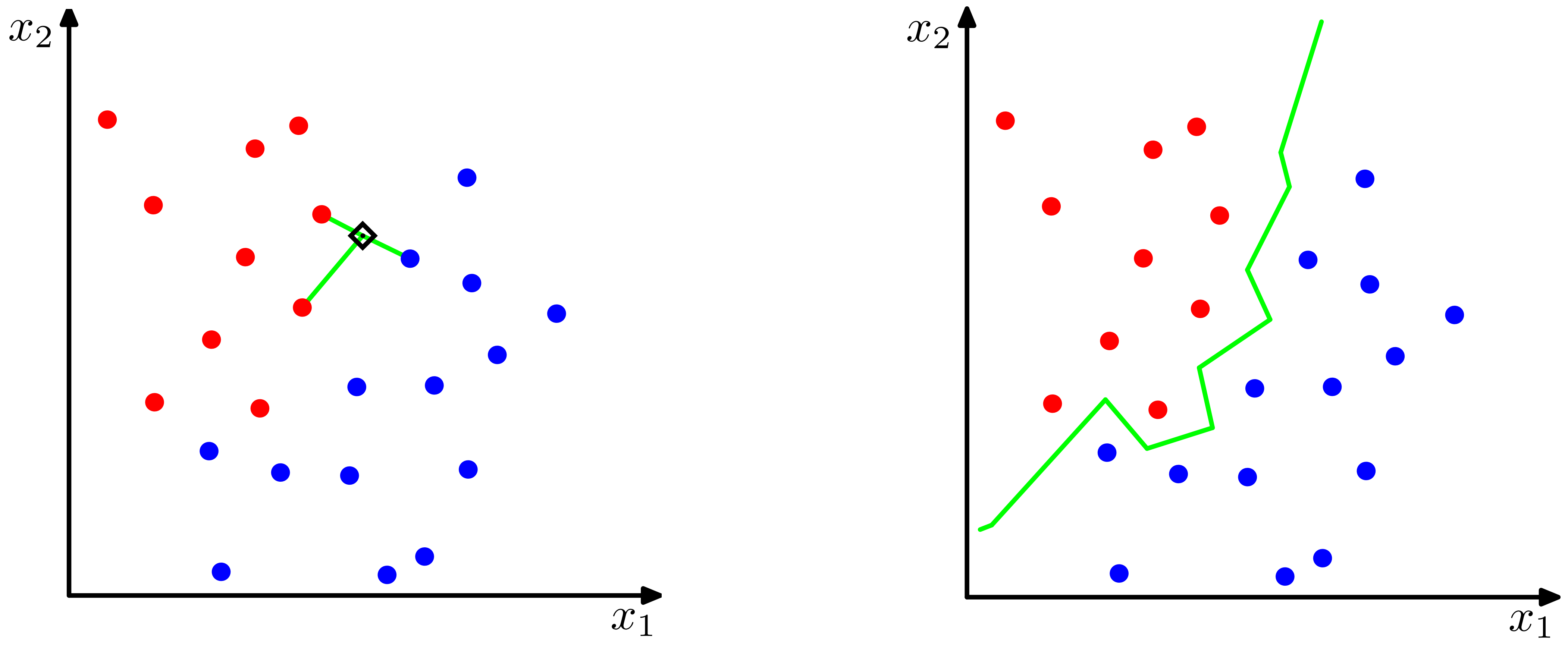
Calculate $|\mathbf{x}_q - \mathbf{x}_i|$ for all training data

Nearest Neighbour Classification

- We can view each image as a point in a high dimensional space



Nearest Neighbour Classifier



(a) (b)

- What is the decision boundary for a nearest-neighbour classifier?

18.1

k-**Nearest Neighbor** (kNN) Classifier

We can gain some robustness to noise by voting over **multiple** neighbours.

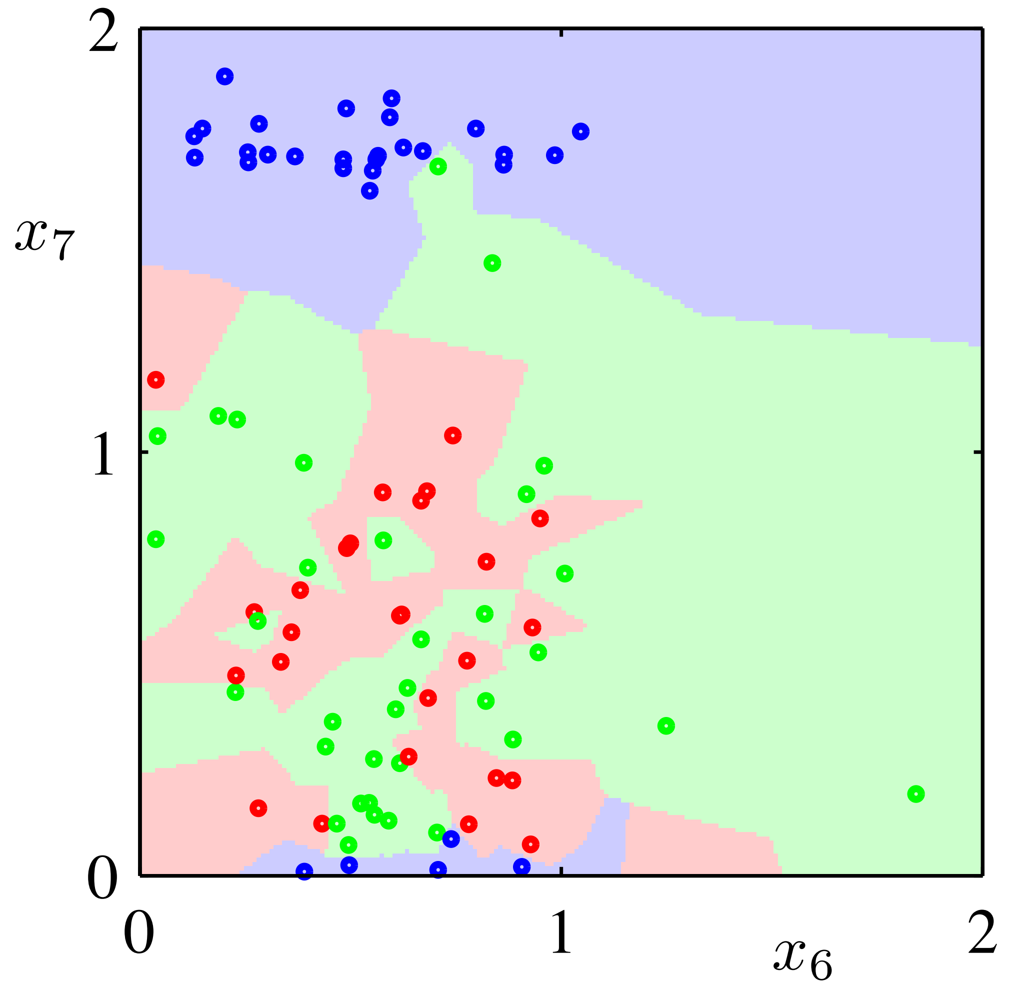
Given a **new** data point, find the k nearest training examples. Assign the label by **majority vote**.

Simple method that works well if the **distance measure** correctly weights the various dimensions

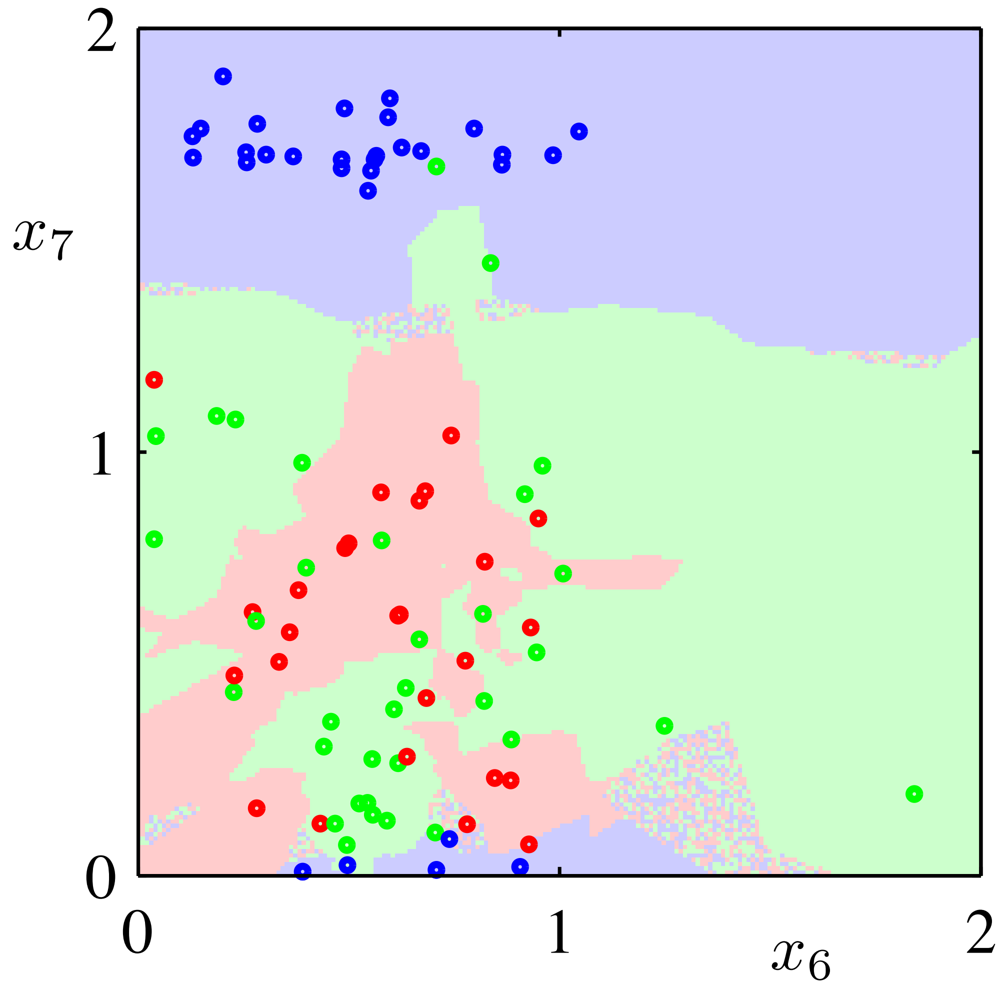
For **large data sets**, as k increases kNN approaches optimality in terms of minimizing probability of error

k-NN Classifier

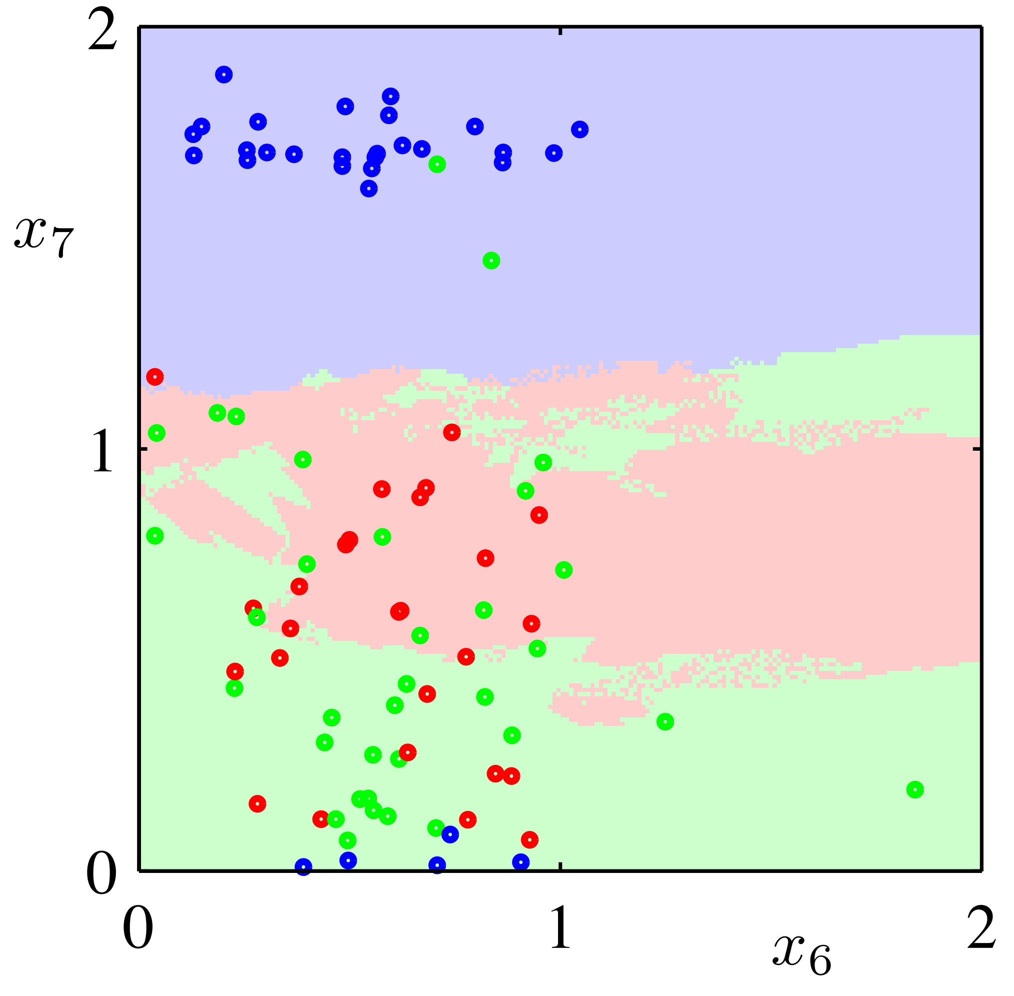
- Identify k nearest neighbours of the query
- Assign class as most common class in set
- k-NN decision boundaries:



k = 1



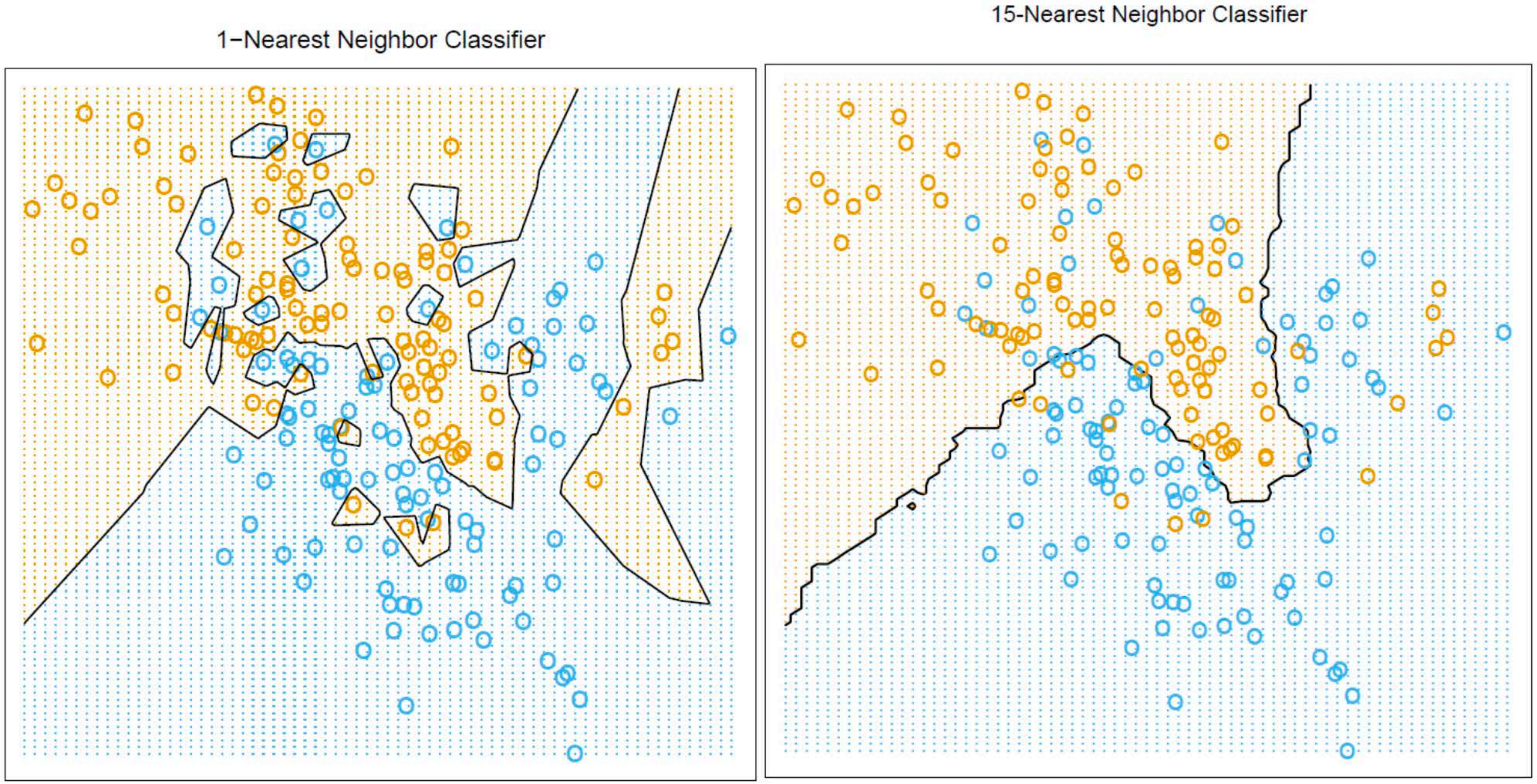
k = 3



k = 31

Good performance depends on suitable choice of k

k-**Nearest Neighbor** (kNN) Classifier



kNN decision boundaries respond to local clusters where one class dominates

Figure credit: Hastie, Tibshirani & Friedman (2nd ed.)

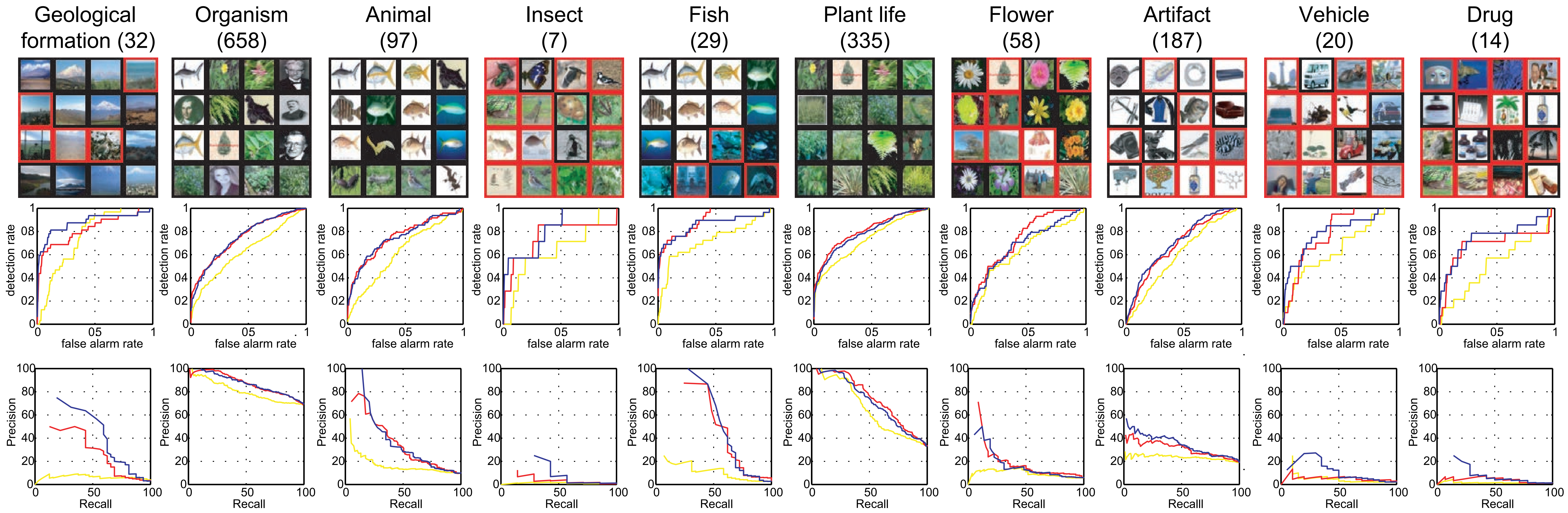
What do nearest neighbours look like with 80 million images?

[Torralba, Fergus, Freeman ‘08]

Query

Tiny Image Recognition

- Recognition performance (categories vary in semantic level)



yellow = 7900, red = 790,000, blue = 79,000,000

Nearest neighbour becomes increasingly accurate as N increases, but do we need to store a dataset of 80 million images?

Linear Classification

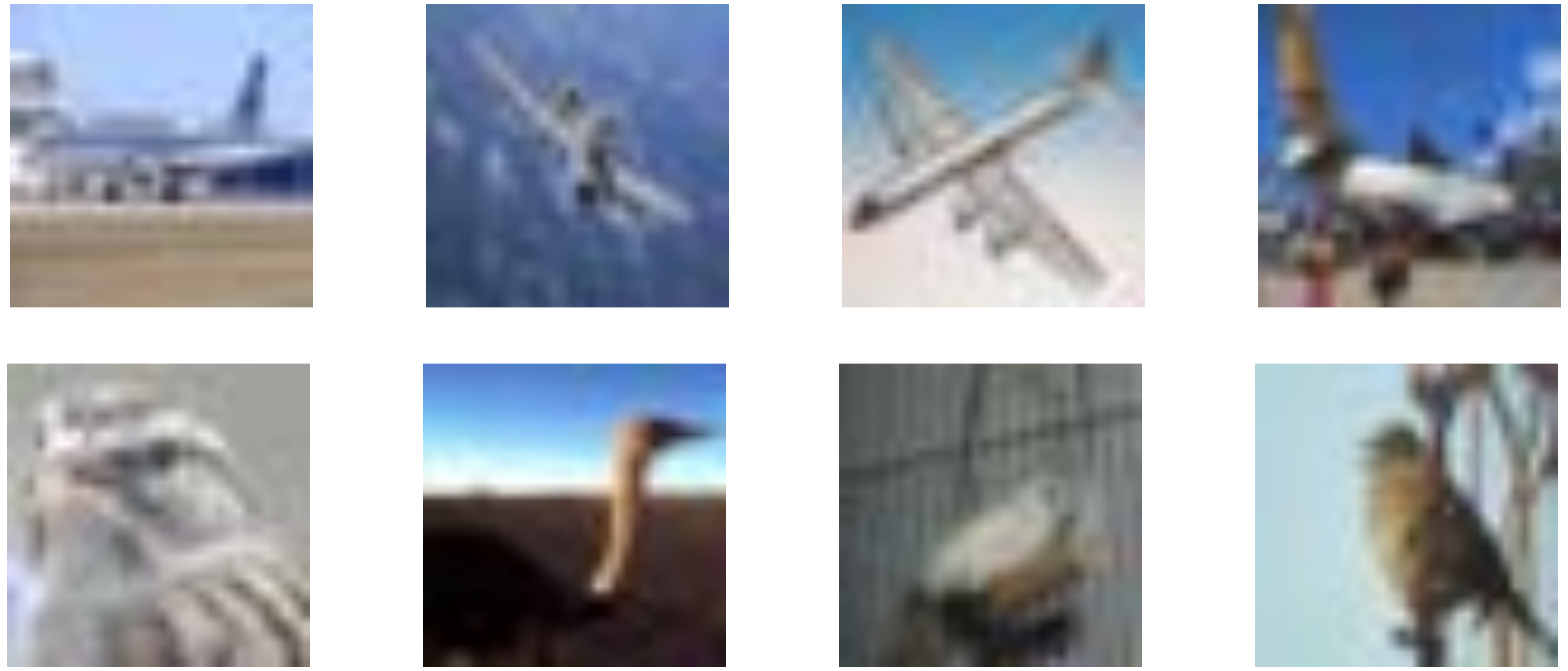
- Linear classification, 2-class, N-class
- Regularization, softmax, cross entropy
- SGD, learning rate, momentum

Linear Classification

- Let's start by using 2 classes, e.g., bird and plane
- Apply labels (y) to training set:

y = +1

y = -1

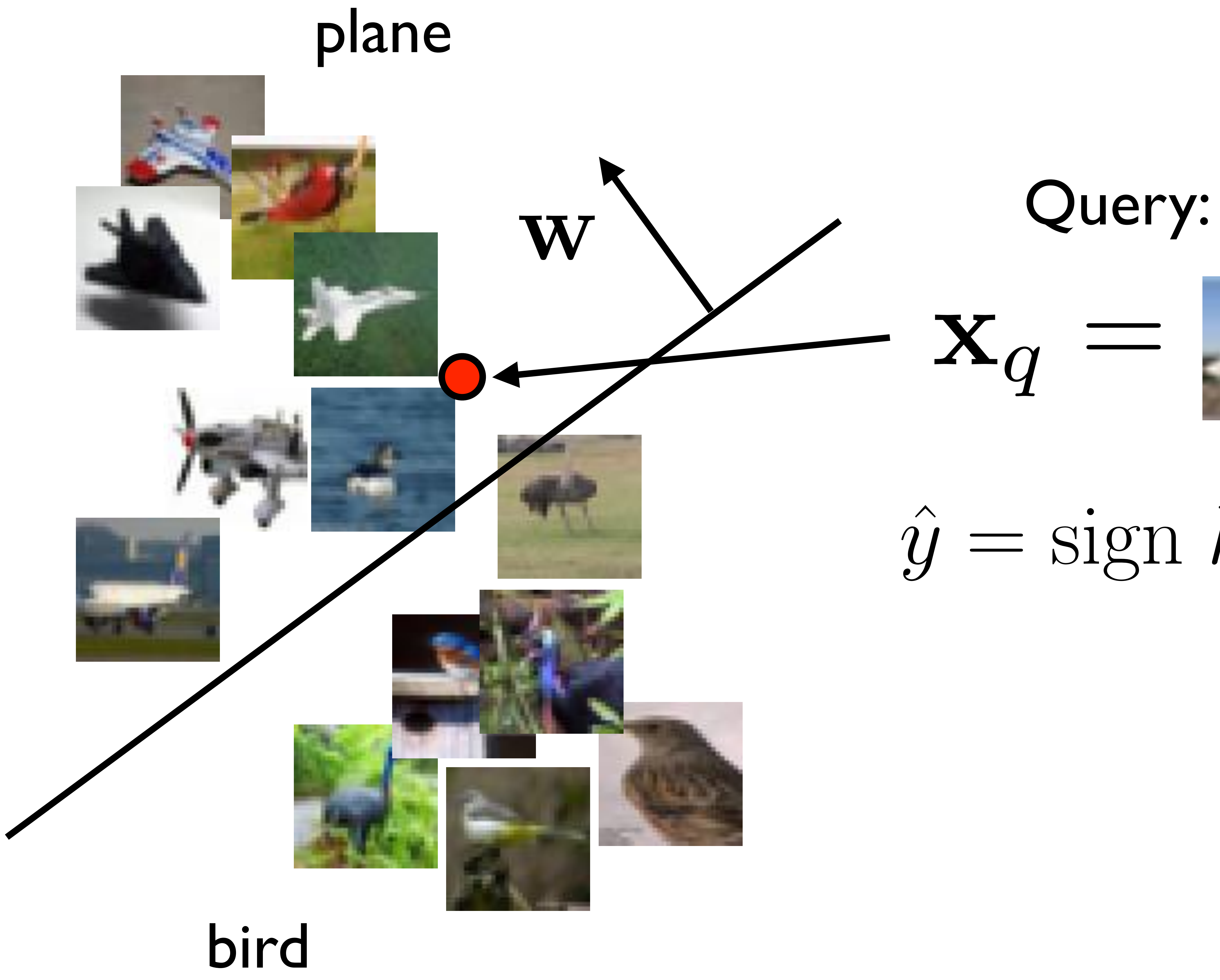


- Use a linear model to regress y from x

18.2

2-class Linear Classification

- Separating hyperplane, projection to a line defined by **w**

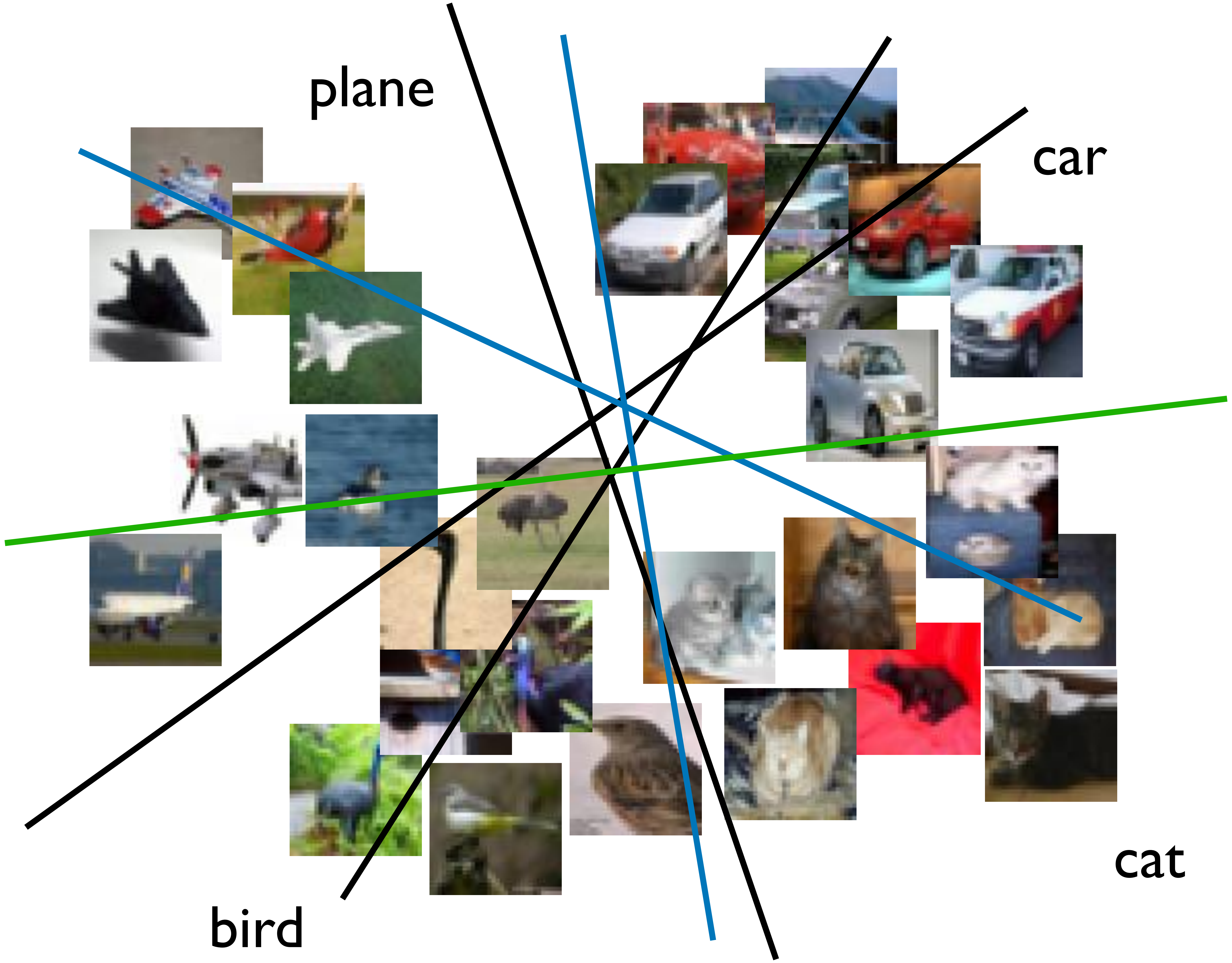


$$\mathbf{x}_q =$$

$$\hat{y} = \text{sign } h = \text{sign } \mathbf{w}^T \mathbf{x}_q$$

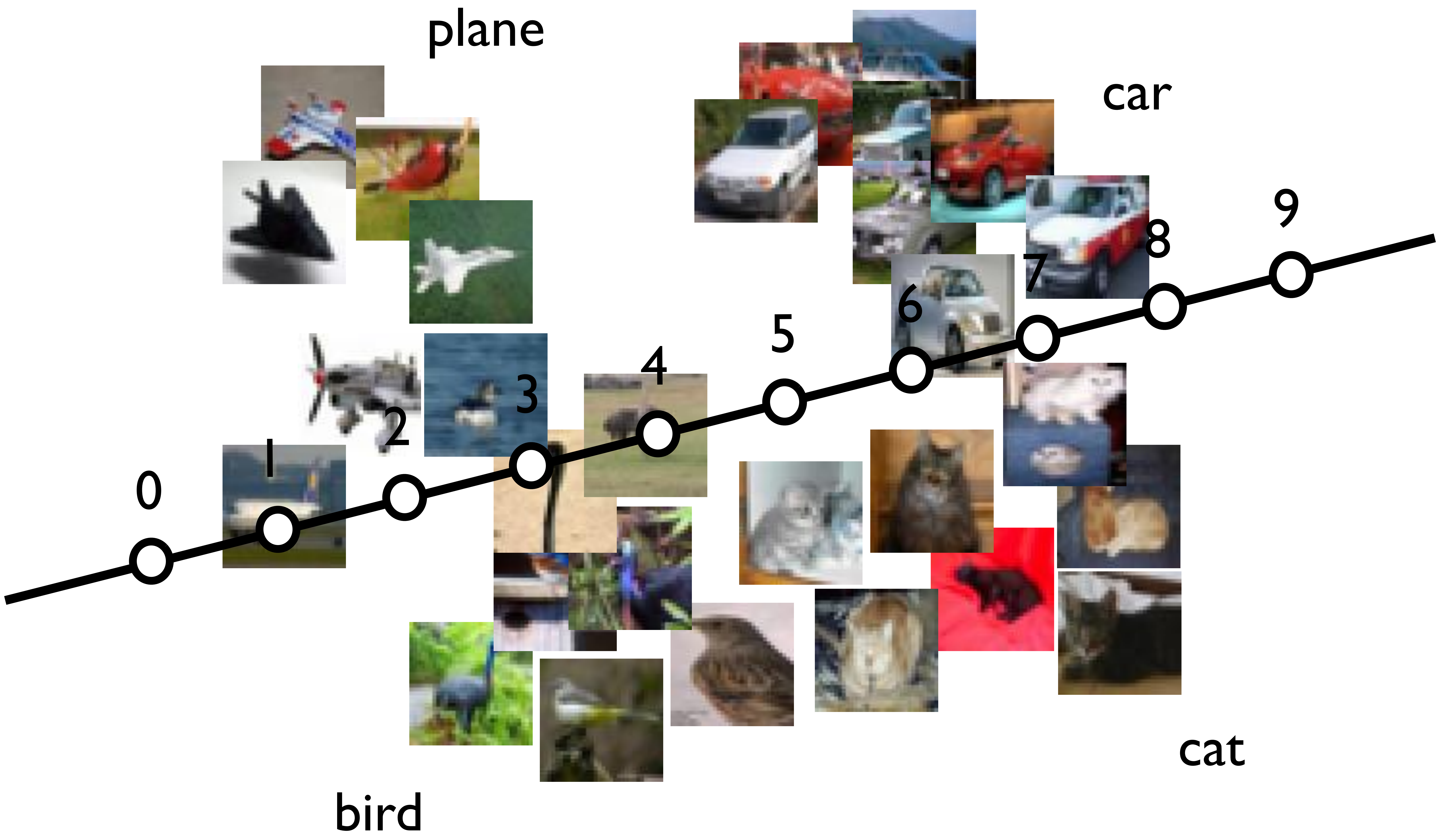
N-class Linear Classification

- We could construct $O(n^2)$ 1 vs 1 classifiers



N-class Linear Classification

- We could regress directly to integer class id, y = {0,1,2,3…9}



One-Hot Regression

- A better solution is to regress to one-hot targets = 1 vs all classifiers

$$\begin{bmatrix} & \mathbf{W}^T & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \cdots \end{bmatrix}$$

← class 2 = 'automobile'

$$\begin{bmatrix} & \mathbf{W}^T & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \cdots \end{bmatrix}$$

← class 4 = 'cat'

18.3

One-Hot Regression

- Stack into matrix form

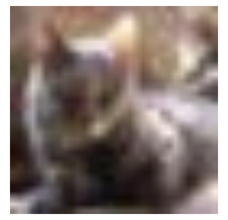
$$\begin{bmatrix} & \mathbf{W}^T & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix} \dots = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \dots \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \dots \end{bmatrix} \dots$$

class 2 = 'automobile'

class 4 = 'cat'

One-Hot Regression

- Transpose

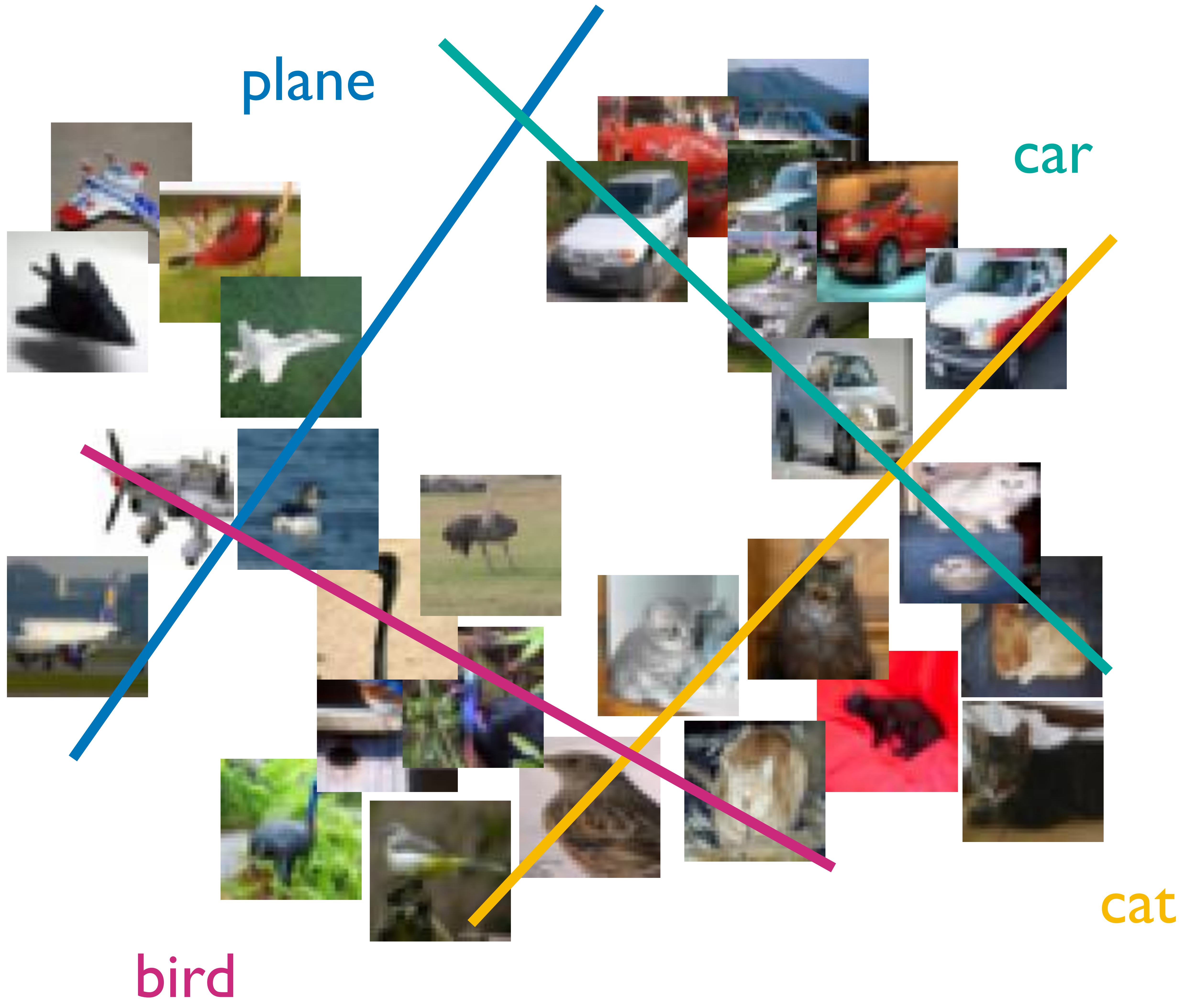
$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots \\ x_{21} & x_{22} & x_{23} & \dots \\ x_{31} & x_{32} & x_{33} & \dots \\ & \dots & & \end{bmatrix} \begin{bmatrix} \\ \mathbf{W} \\ \\ \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ & .. & .. & & \end{bmatrix} \begin{matrix} \text{auto} \\ \text{cat} \\ \\ \end{matrix}$$

$$\mathbf{XW} = \mathbf{T}$$

- Solve regression problem by Least Squares

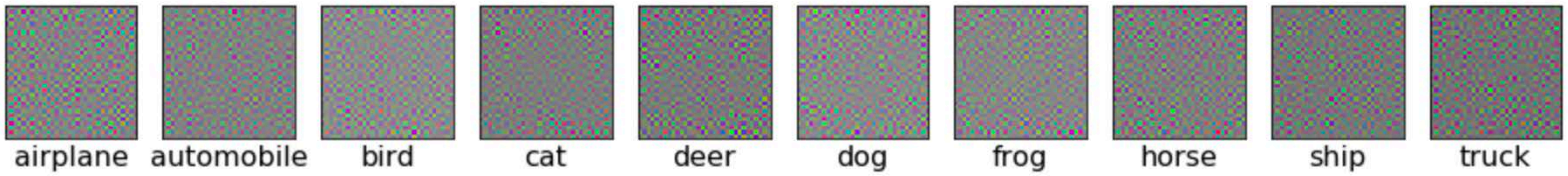
N-class Linear Classification

- One hot regression = 1 vs all classifiers



One-Hot Regression

- Visualise class templates for the least squares solution

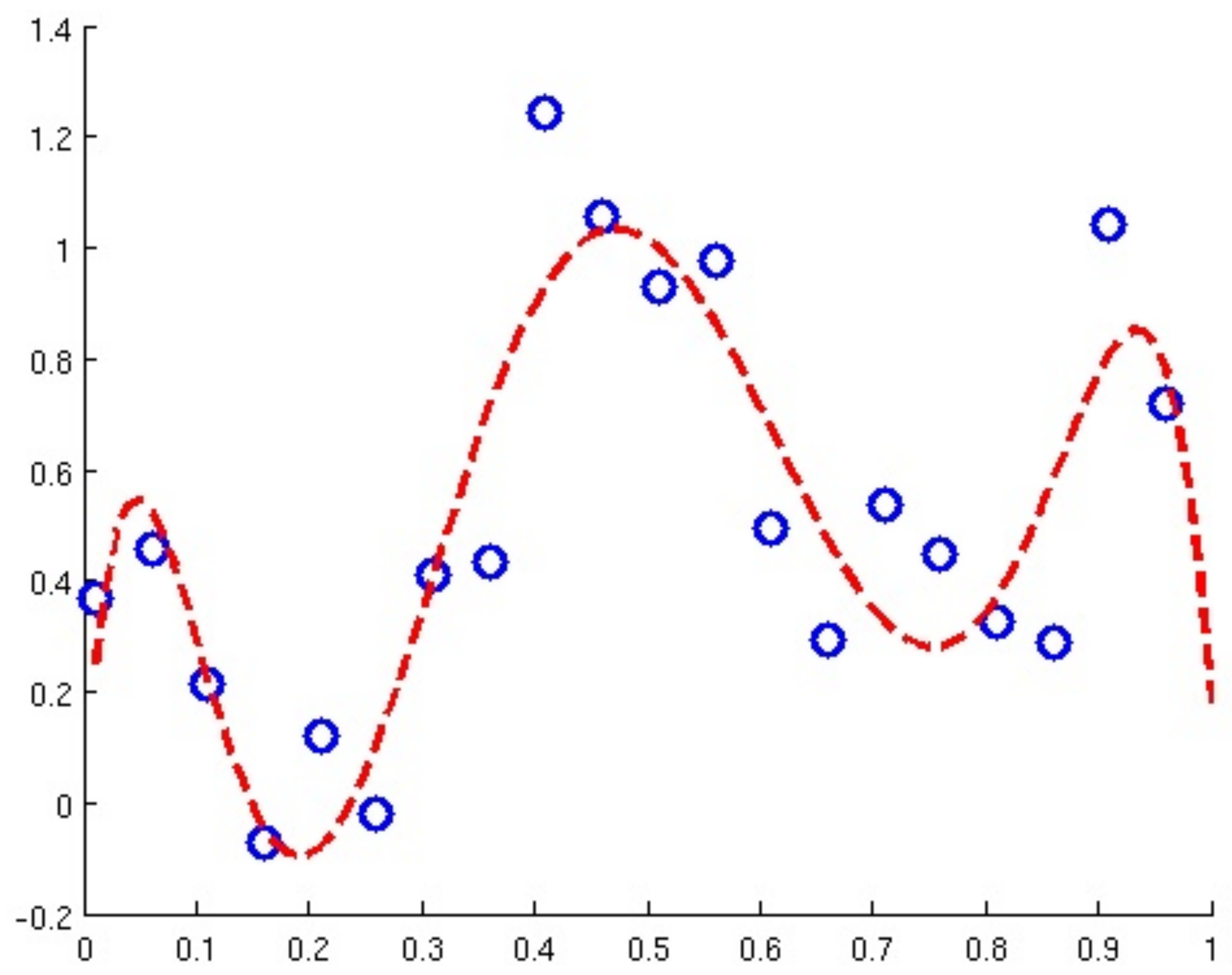


- Classifier accuracy = 35% (not bad, c.f., nearest mean = 27%)

What is happening here?

Polynomial Fitting

- Consider fitting a polynomial to some data by linear regression



18.4

Polynomial Fitting

- Multiple data points (y_i, x_i)

$$
\begin{aligned}
y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 \\
y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 \\
y_3 &= a_0 + a_1 x_3 + a_2 x_3^2 + a_3 x_3^3 \\
&\dots
\end{aligned}
$$

- In matrix form

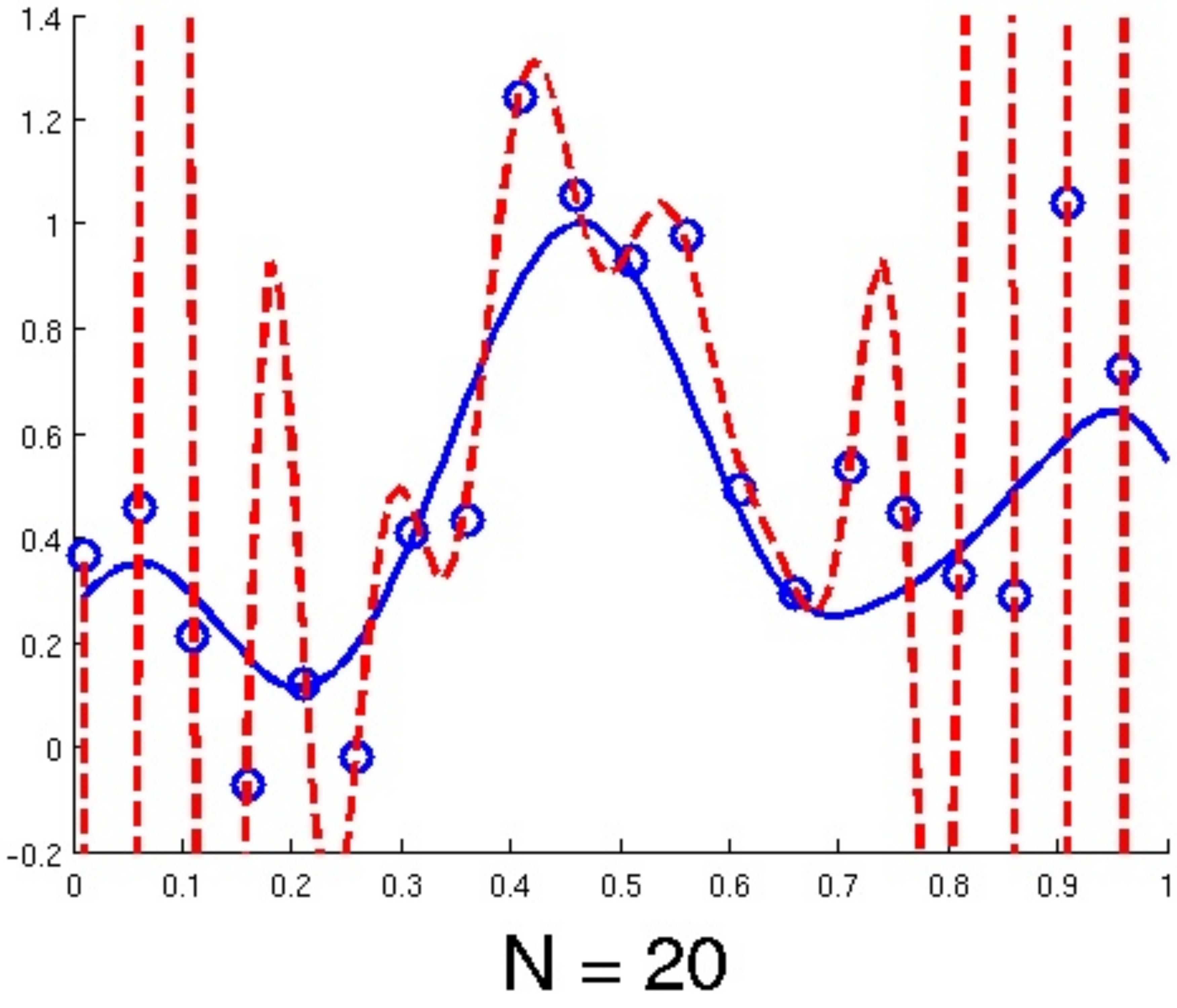
$$
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ & \dots & \dots & \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}
$$

$$
\mathbf{y} = \mathbf{M}\mathbf{a}
$$

- Solve linear system by Gaussian elimination (if square) or Least Squares (if overconstrained)

Polynomial Fitting

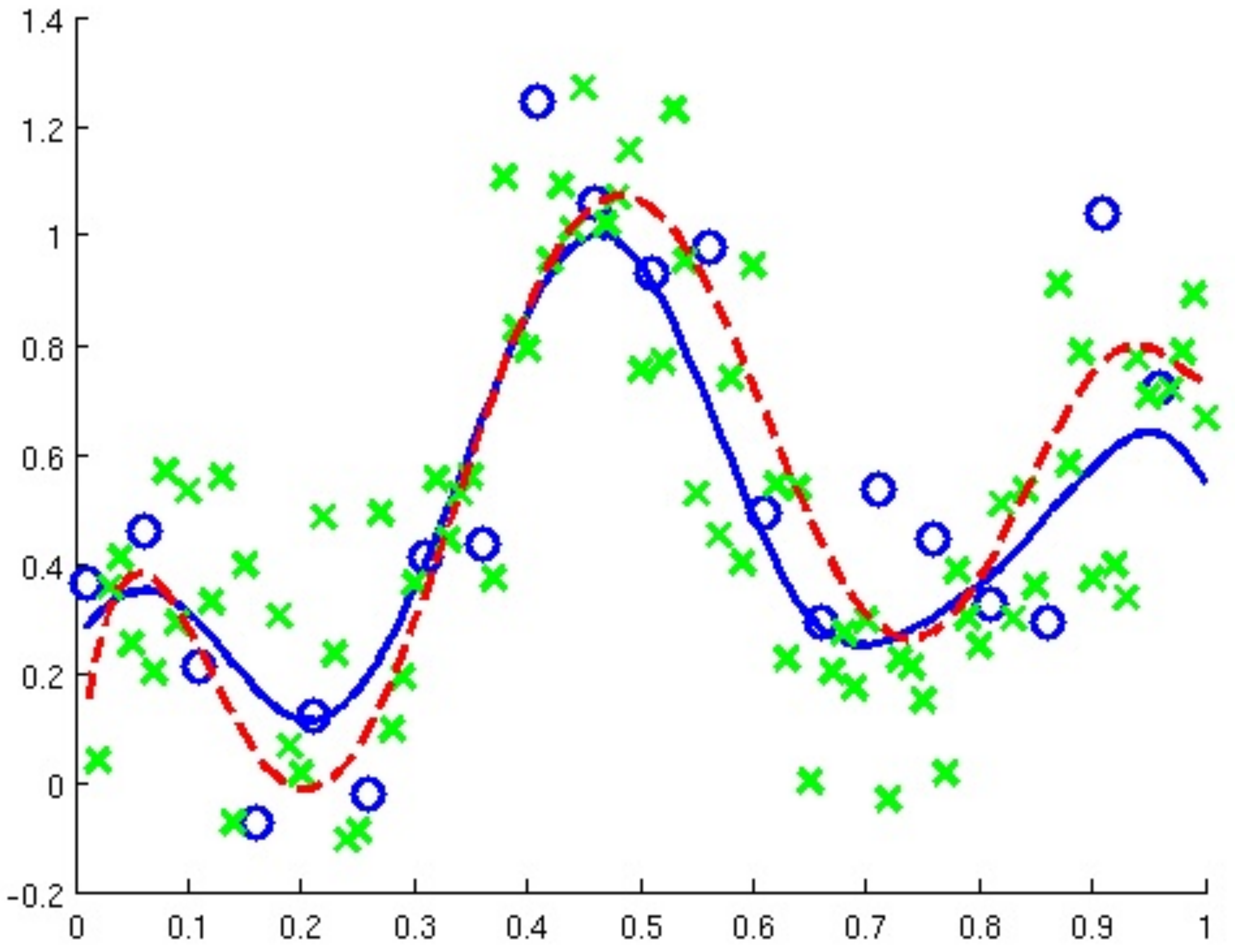
- Fit Nth order polynomial by least squares



- Overfitting

Cross Validation

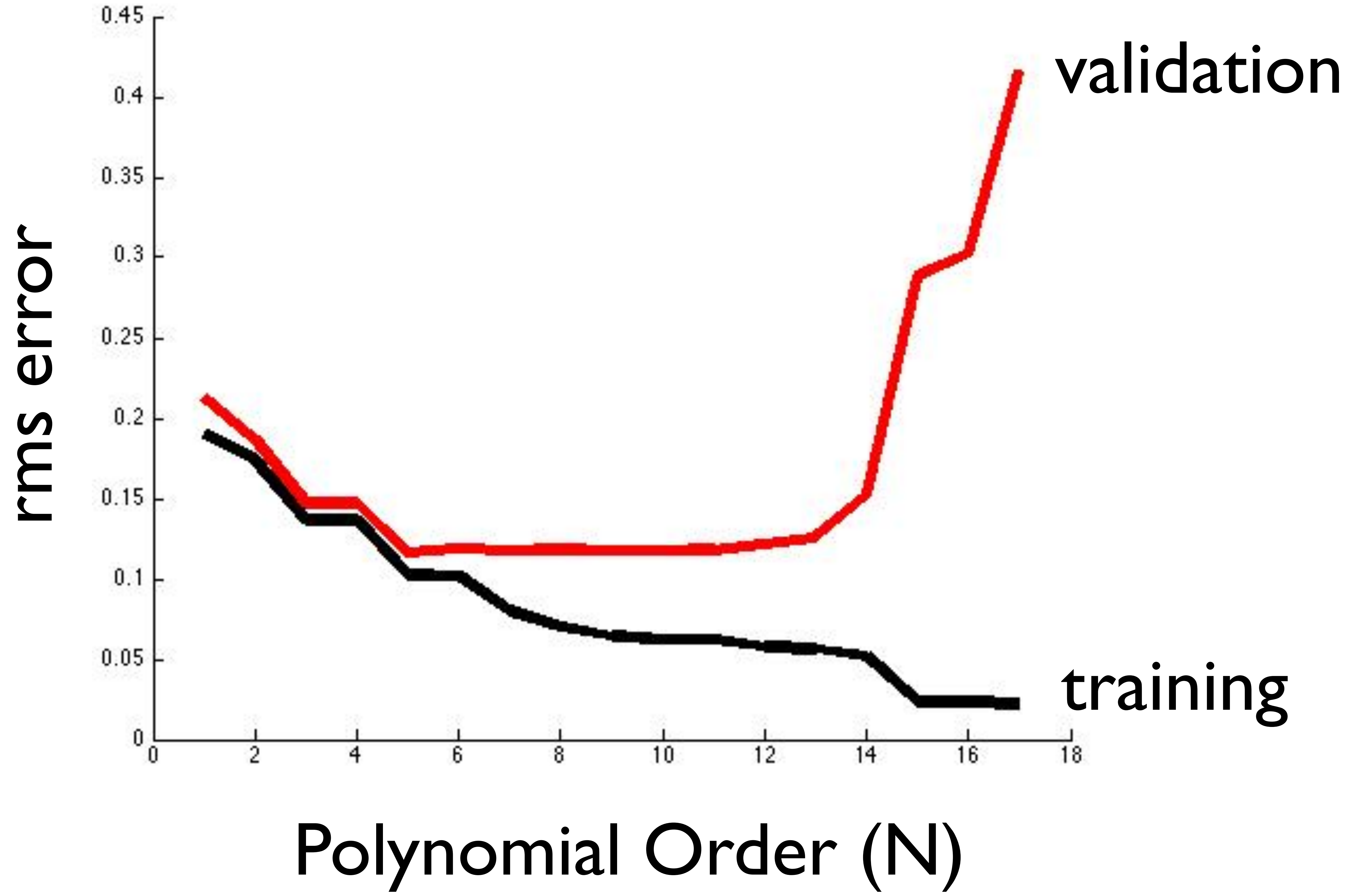
- Fit the model to a subset of data, and evaluate the fit on a held out **validation set**



- Calculate rms error $e_{rms} = \left(\frac{1}{N} \sum_i (y_i - \hat{y}_i)^2 \right)^{\frac{1}{2}}$

Cross Validation

- Training error always decreases, but validation error has a minimum for the best model order



Polynomial Fitting

- For large N, coefficients become HUGE!

	N=1	N=2	N=4	N=10
a_0	0.90	2.03	-2.88	48.50
a_1		-1.54	29.76	-1294.90
a_2			-57.43	14891.41
a_3			31.86	-95161.10
a_4				367736.84
a_5				-885436.68
a_6				1331063.41
a_7				-1212056.89
a_8				610930.32
a_9				-130727.39

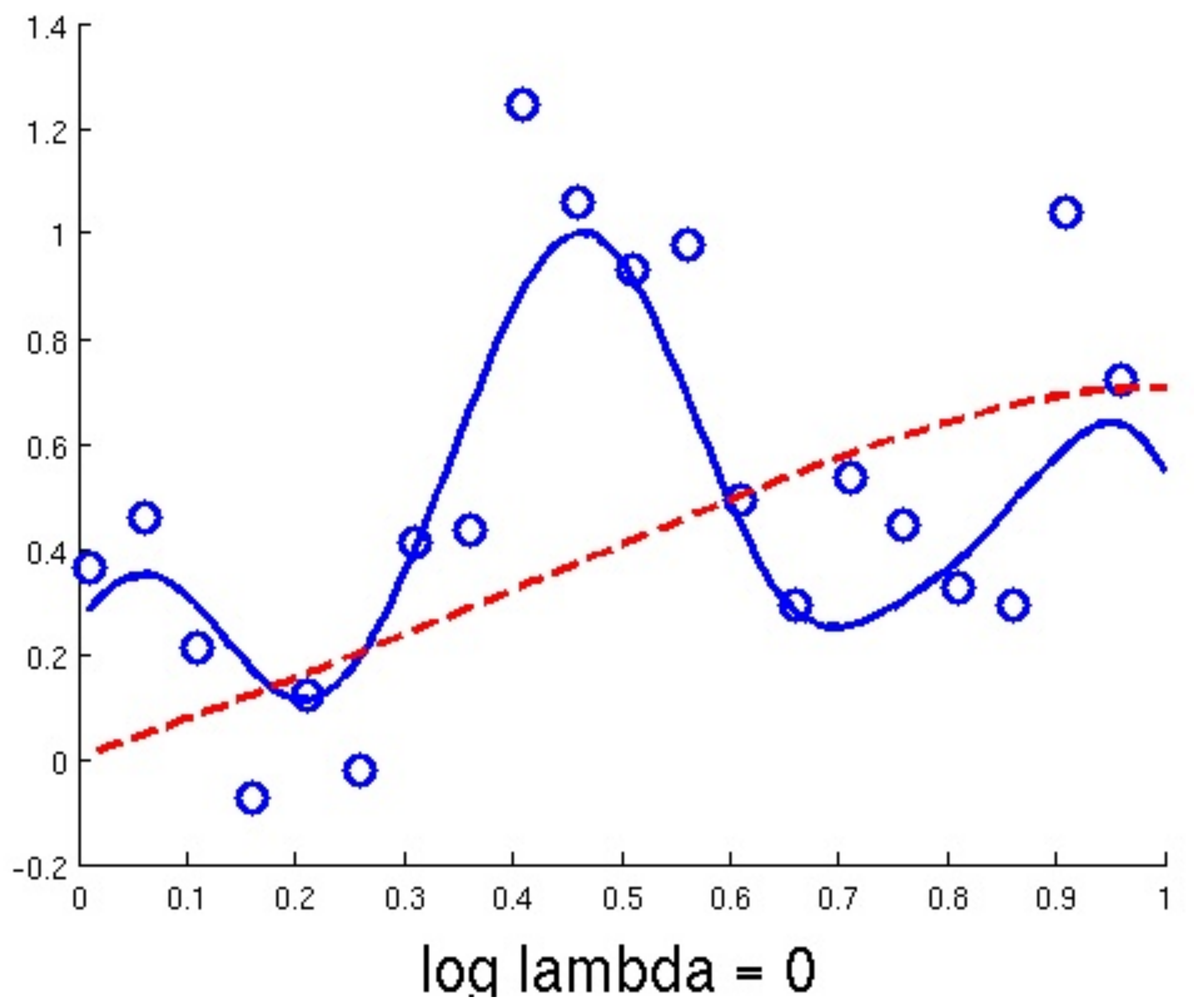
Regularization

- L2 penalty on polynomial coefficients

18.5

Regularized Linear Regression

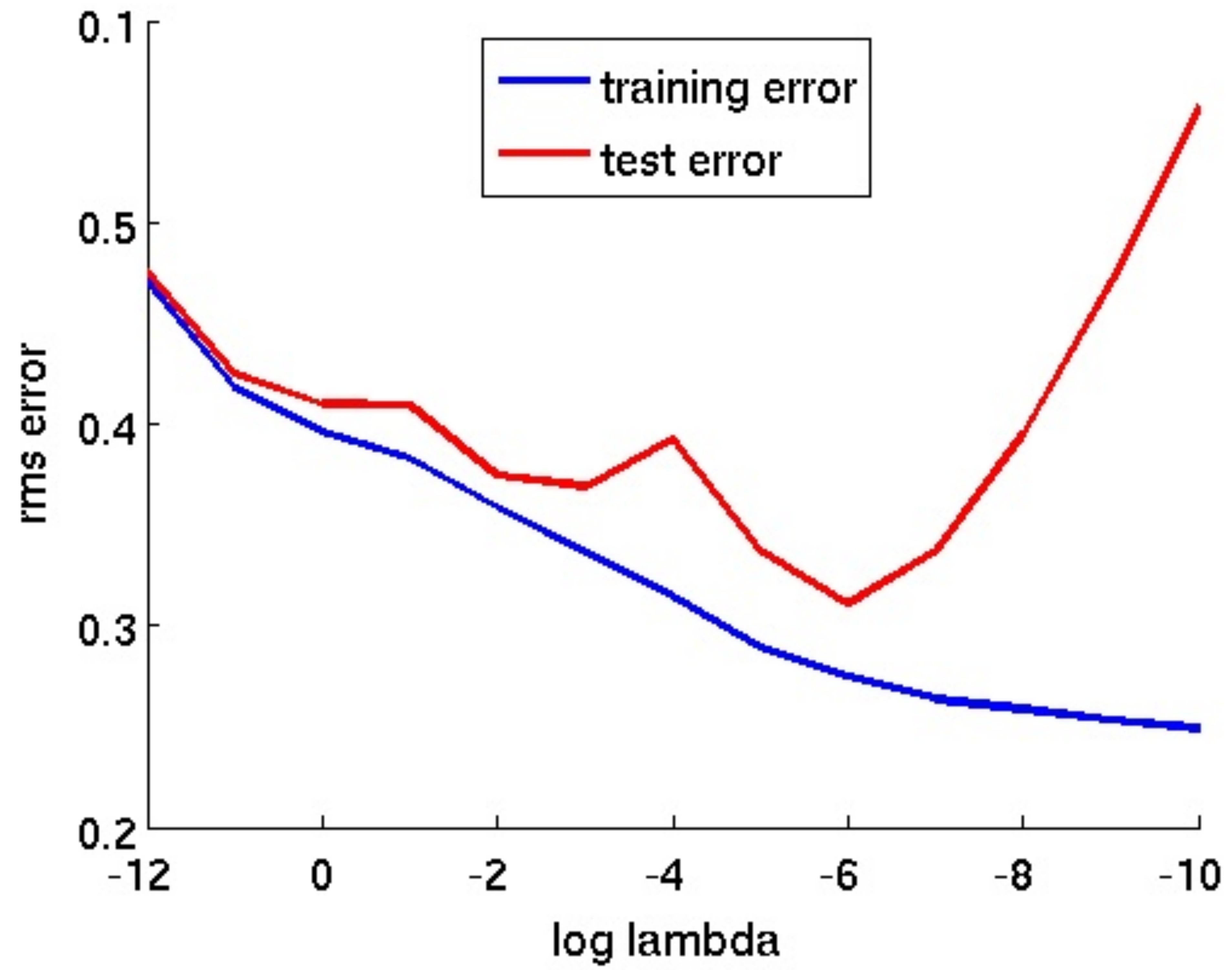
- 10th order polynomial, prior on the coefficients weight λ



- Over-smoothing...

Under/Overfitting

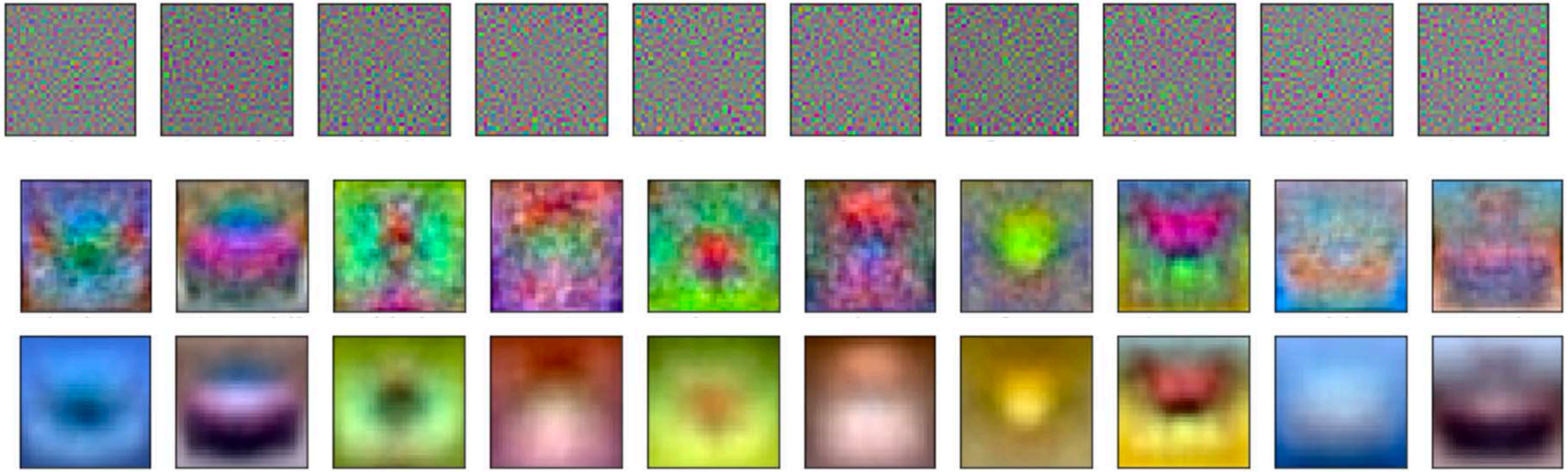
- Test error vs lambda



- Training error always decreases as lambda is reduced
- Test error reaches a minimum, then increases ⇒ overfitting

Regularized Classification

- Add regularization to CIFAR10 linear classifier



- Row 1 = overfitting, Row 3 = oversmoothing?

Non-Linear Optimisation

- With a linear predictor and L2 loss, we have a closed form solution for model weights W
- How about this (non-linear) function

$$\mathbf{h} = \mathbf{W}_2 \max(0, \mathbf{W}_1 \mathbf{x})$$

- Previously (e.g., bundle adjustment), we locally linearised the error function and iteratively solved linear problems

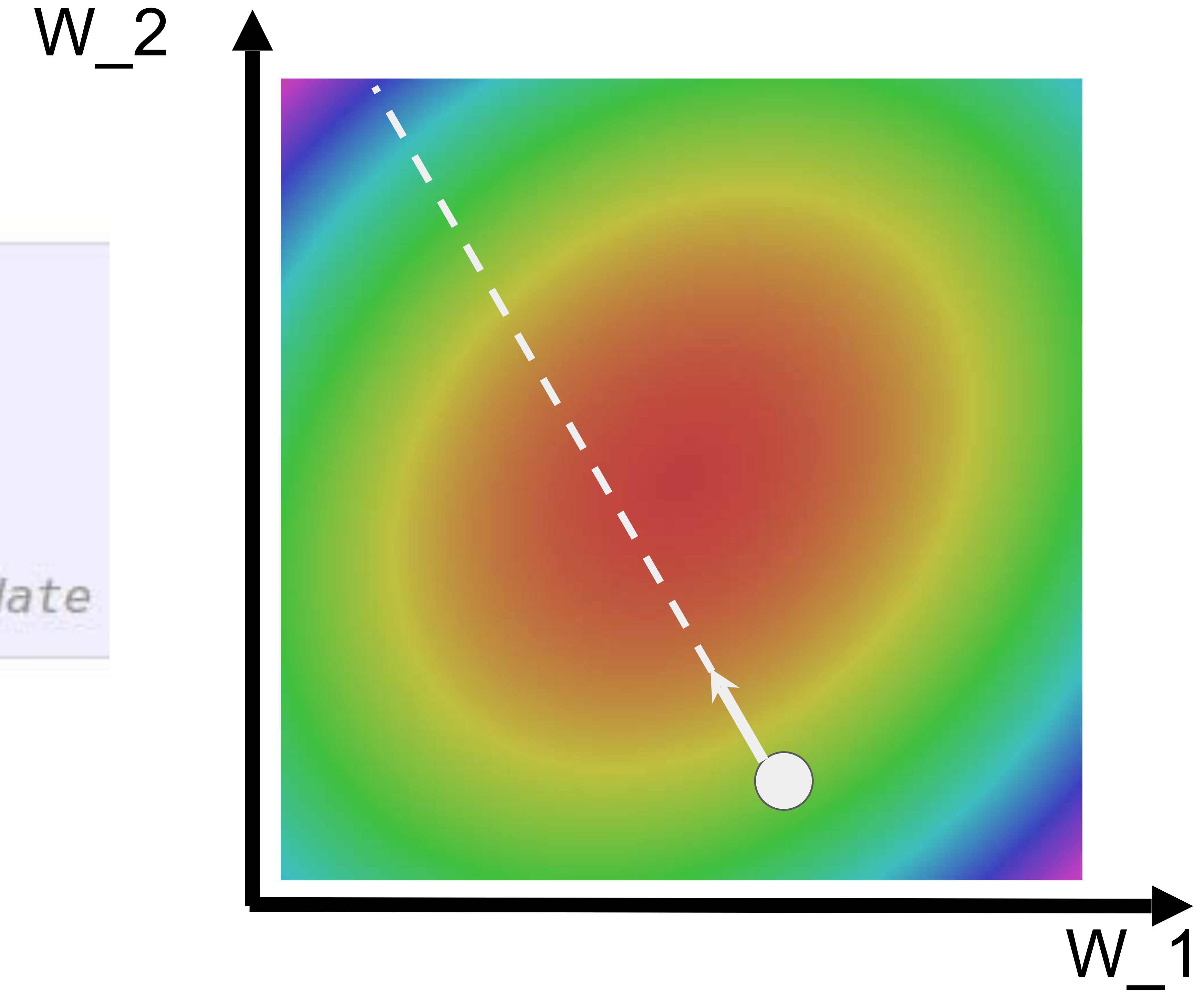
$$e = \sum_i |\mathbf{h}_i - \mathbf{t}_i|^2 \approx |\mathbf{J}\Delta\mathbf{W} + \mathbf{r}|^2$$

$$\Delta\mathbf{W} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}$$

Does this look like a promising approach?

Vanilla Gradient Descent

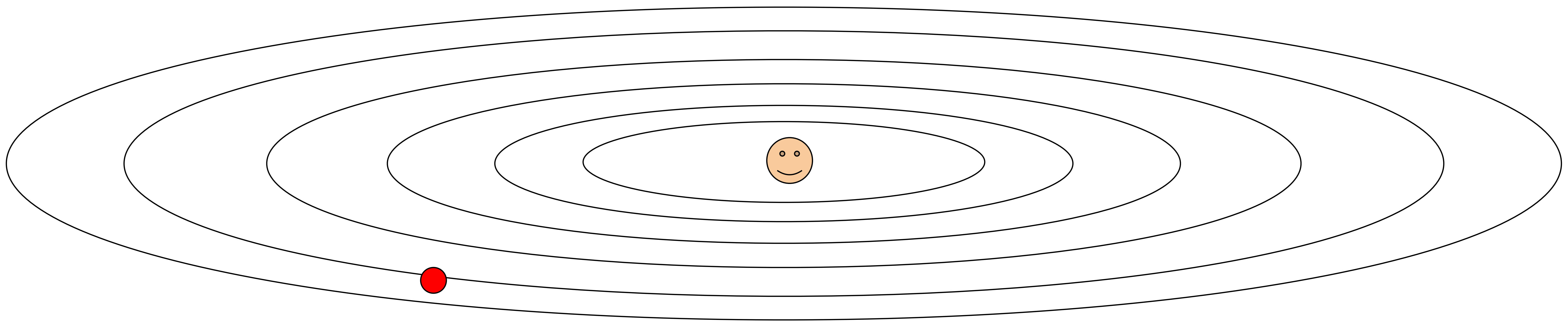
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission

Problem with vanilla GD

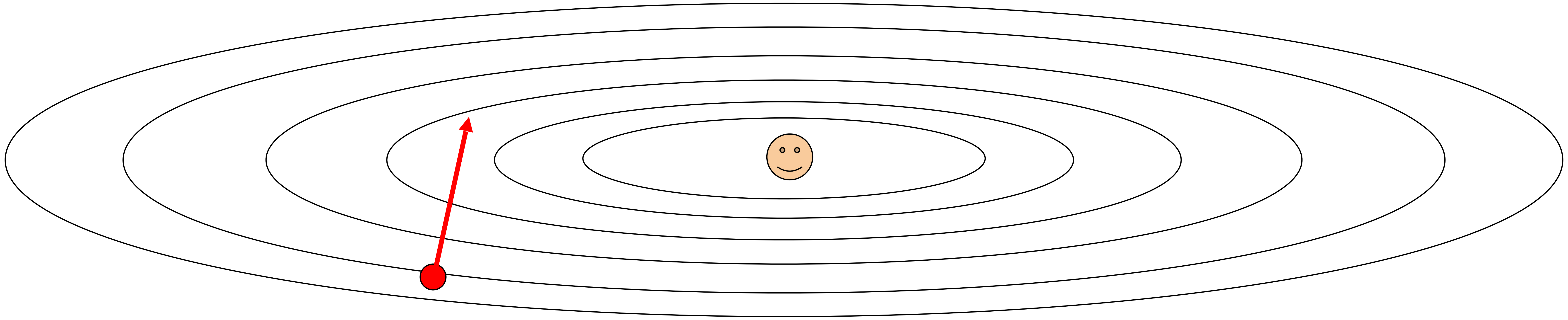
What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problem with vanilla GD

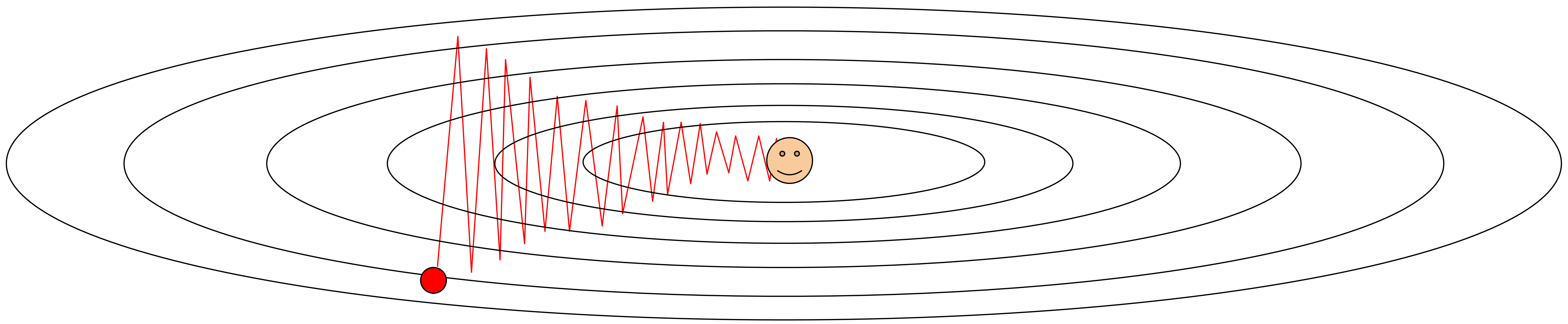
What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problem with vanilla GD

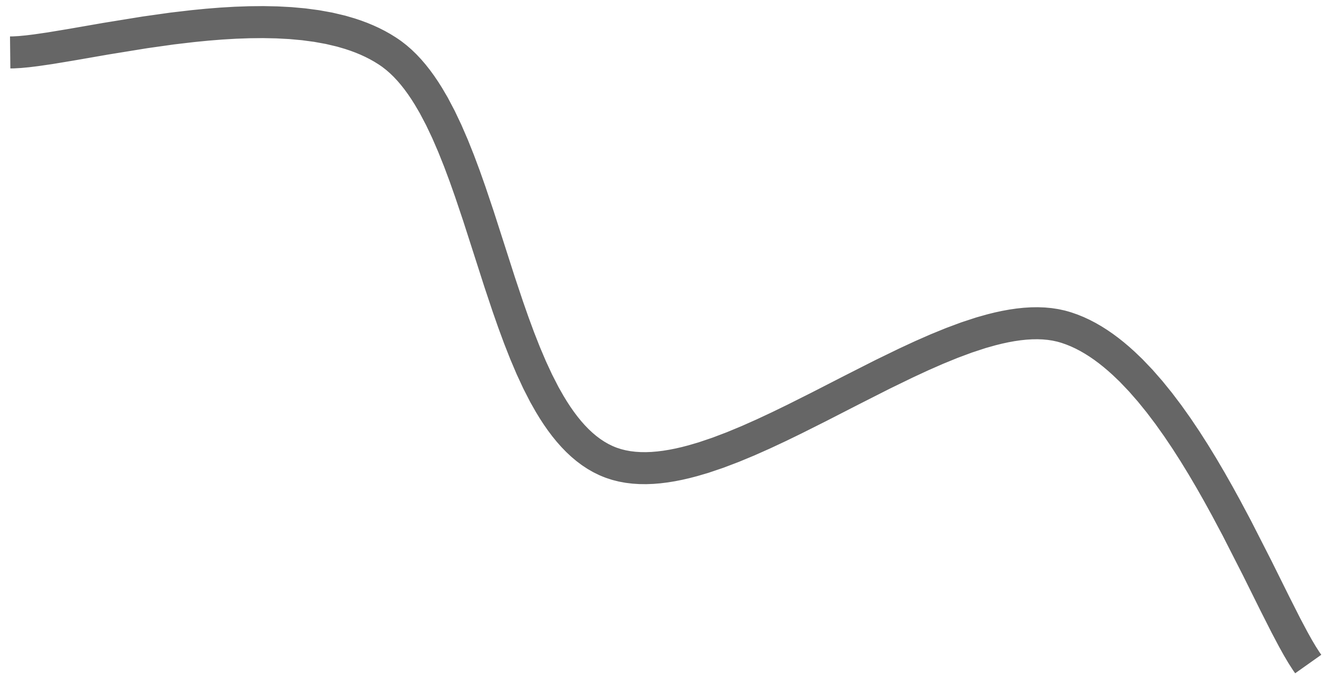
What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

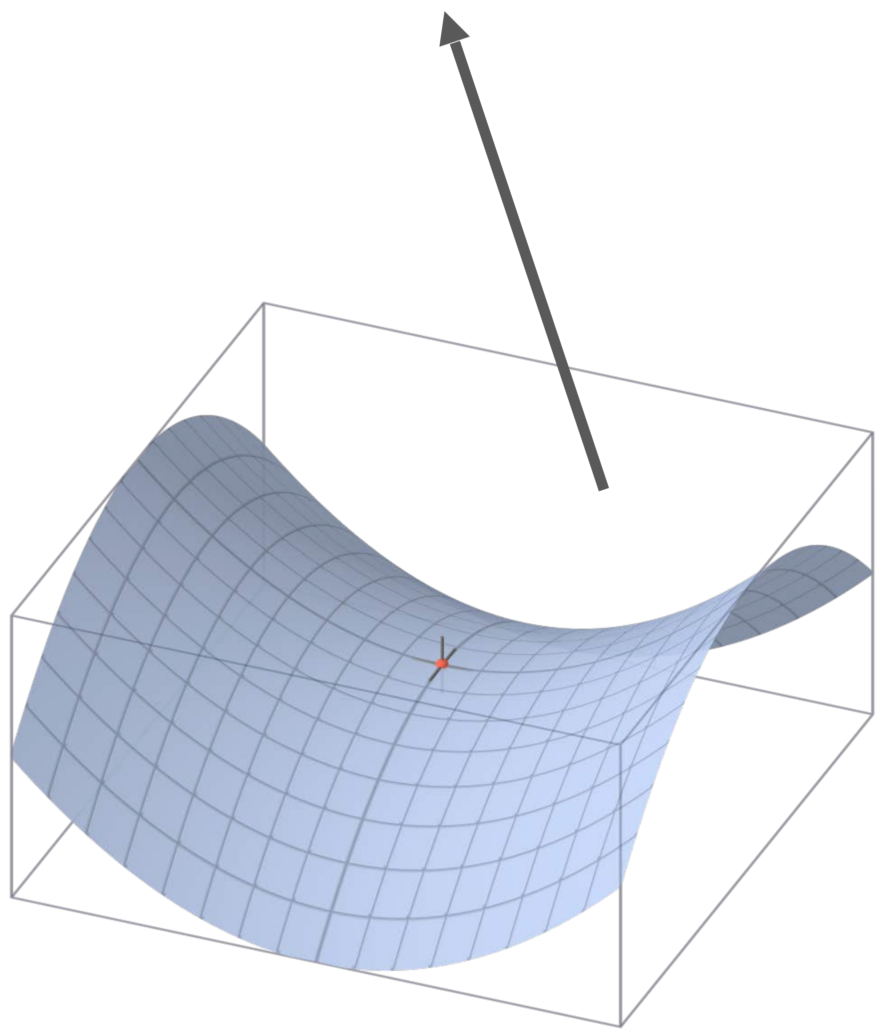
Optimization: problem with SGD

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

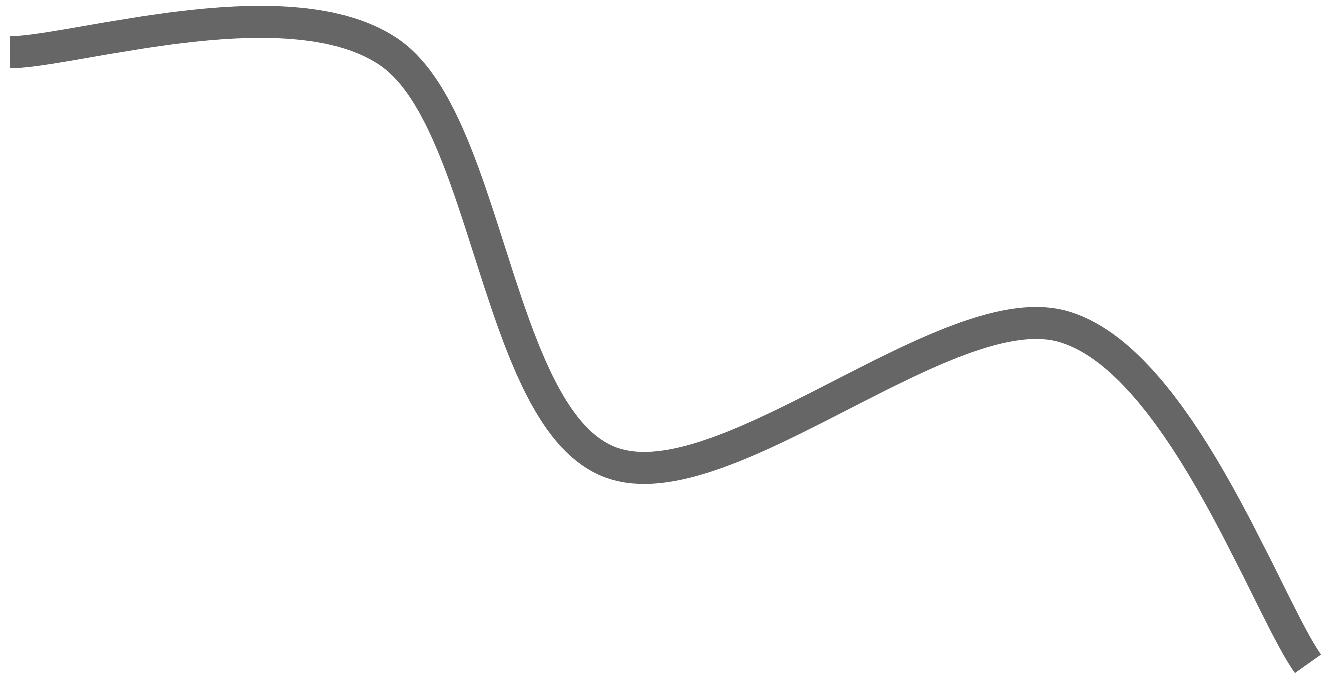


Optimization: problem with SGD

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

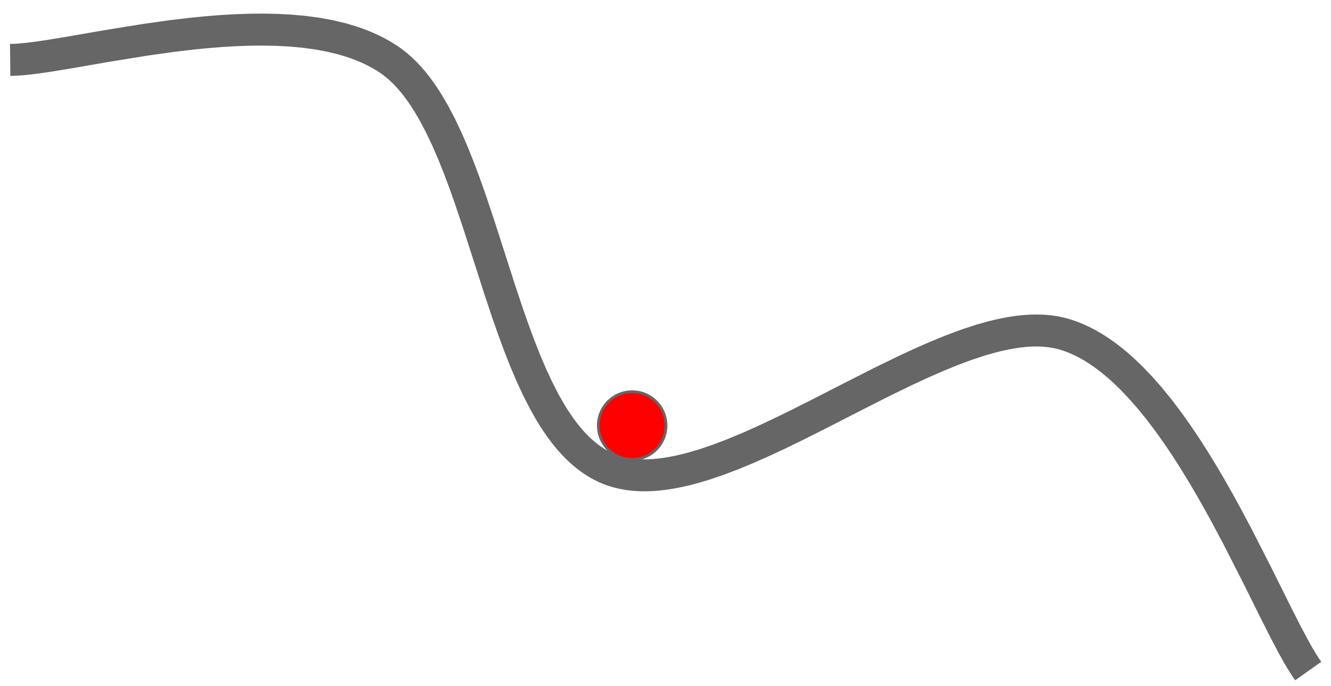


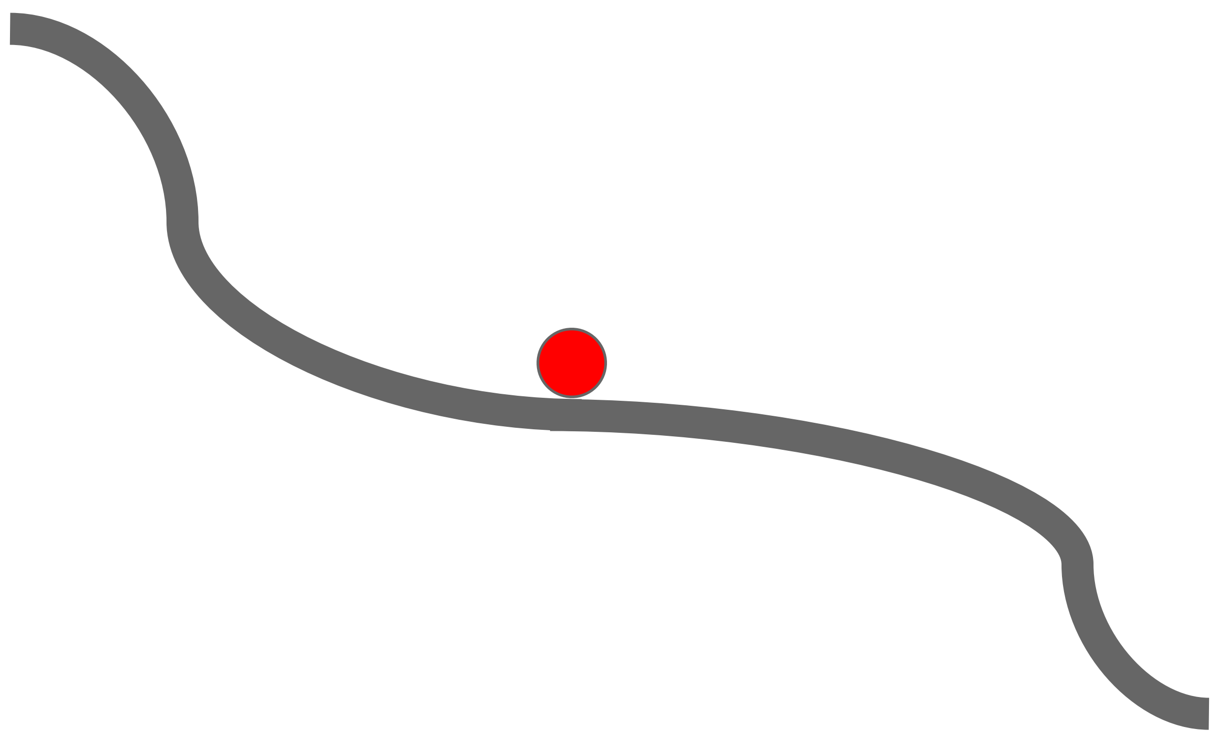
[Image](#) by [Oleg Alexandrov](#) is in the public domain



Optimization: problem with SGD

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

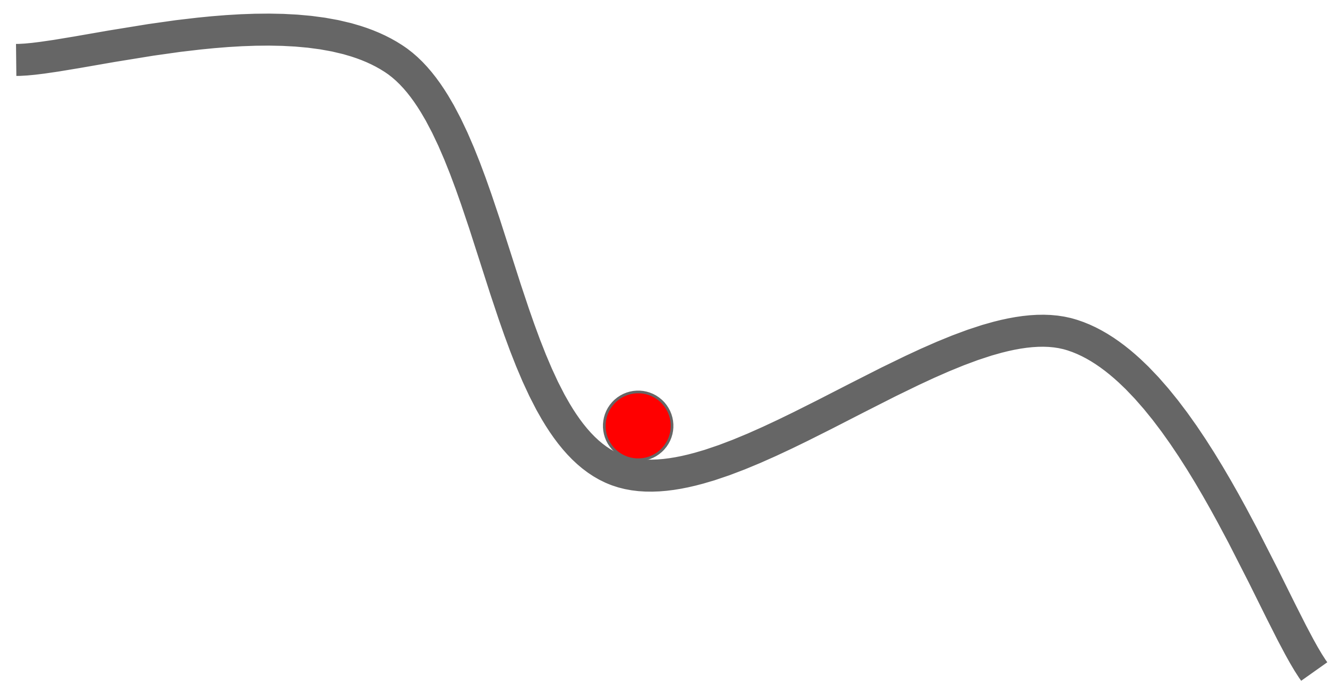


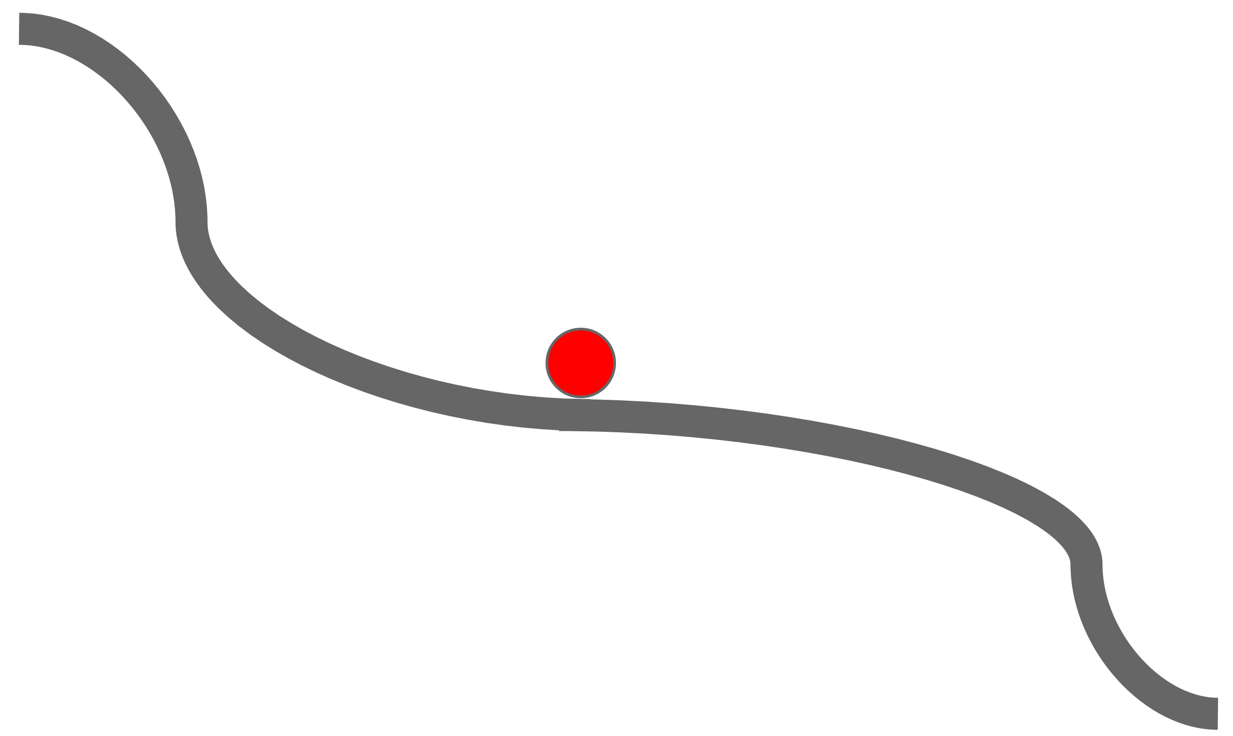


Optimization: problem with SGD

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

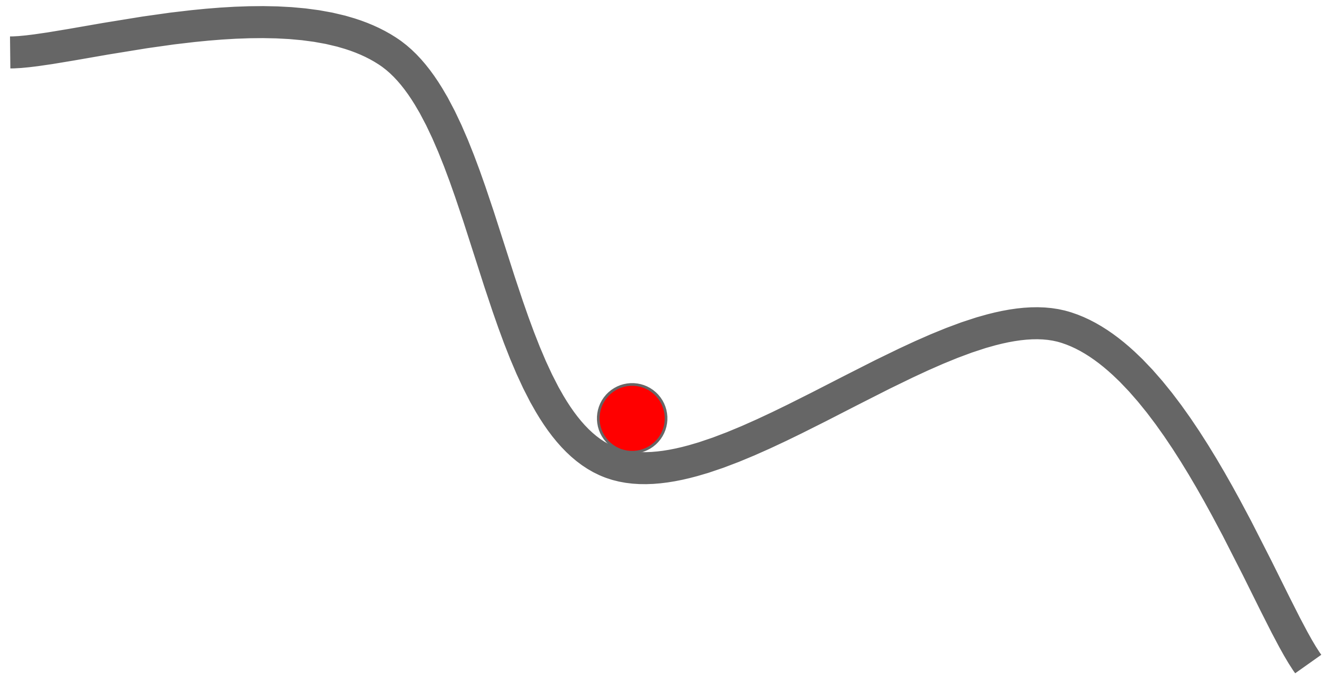
Zero gradient, gradient descent gets stuck



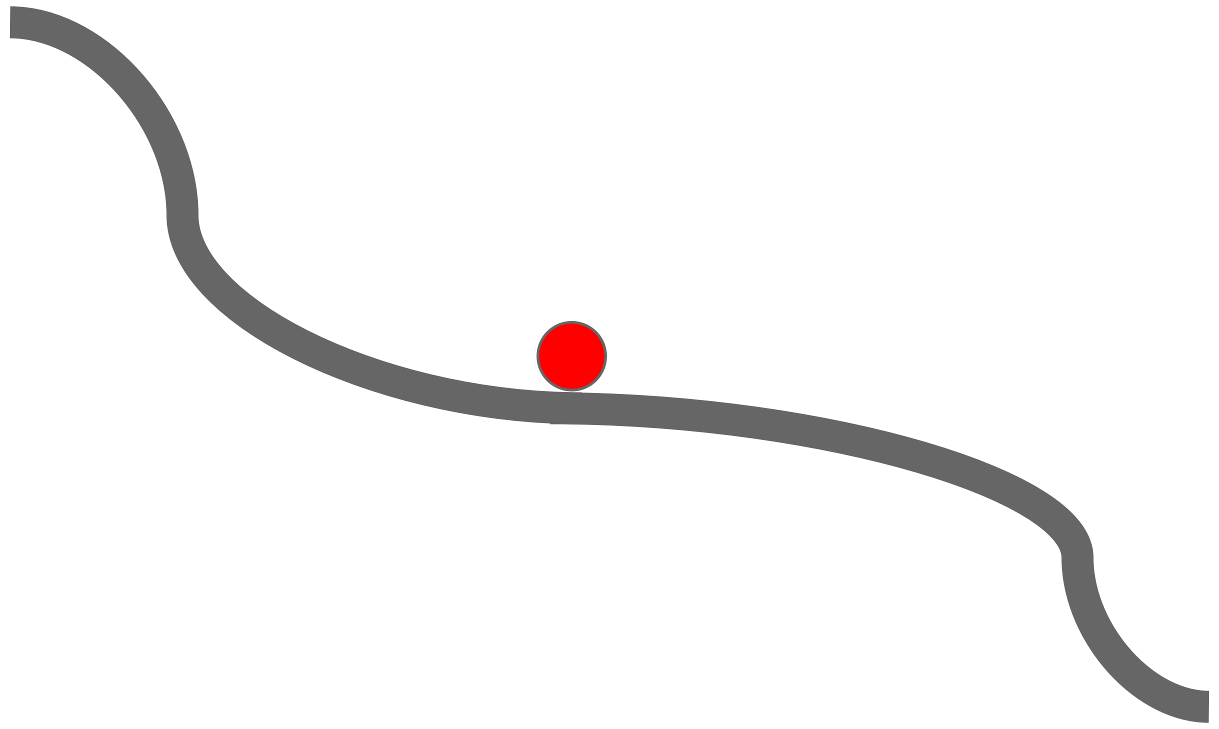


Optimization: problem with SGD

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



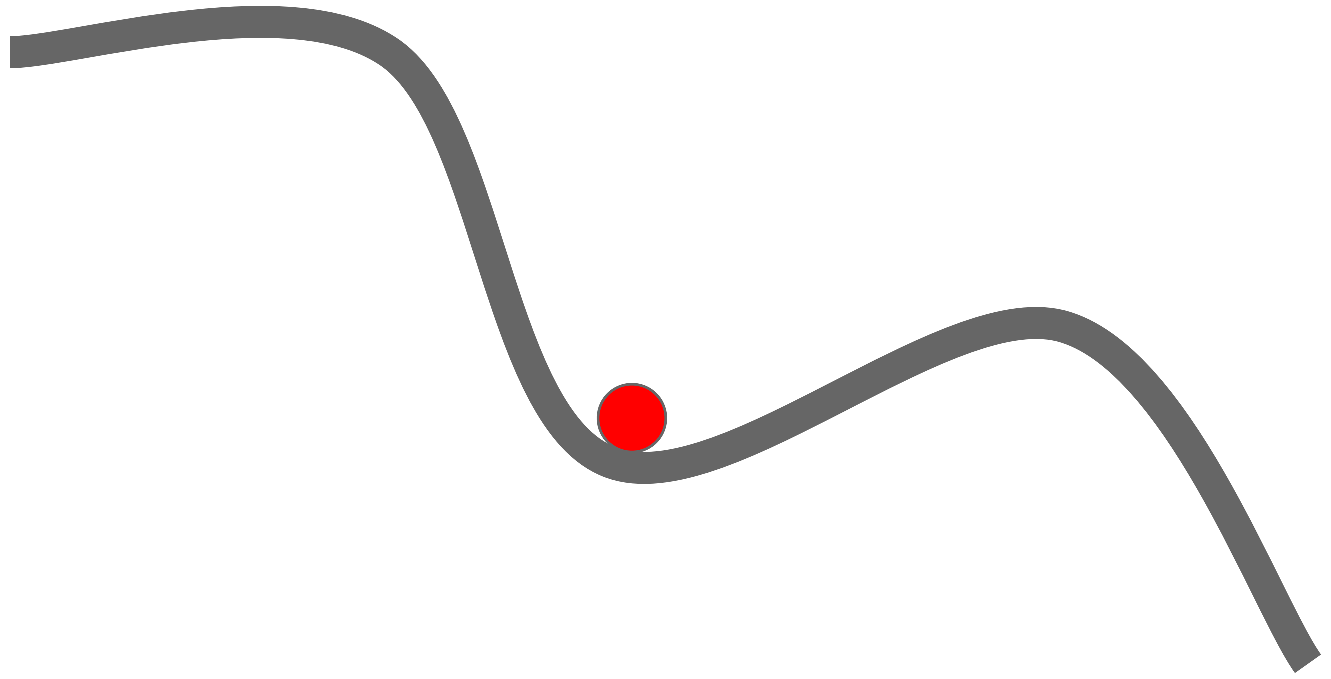
Saddle points much more common in high dimension



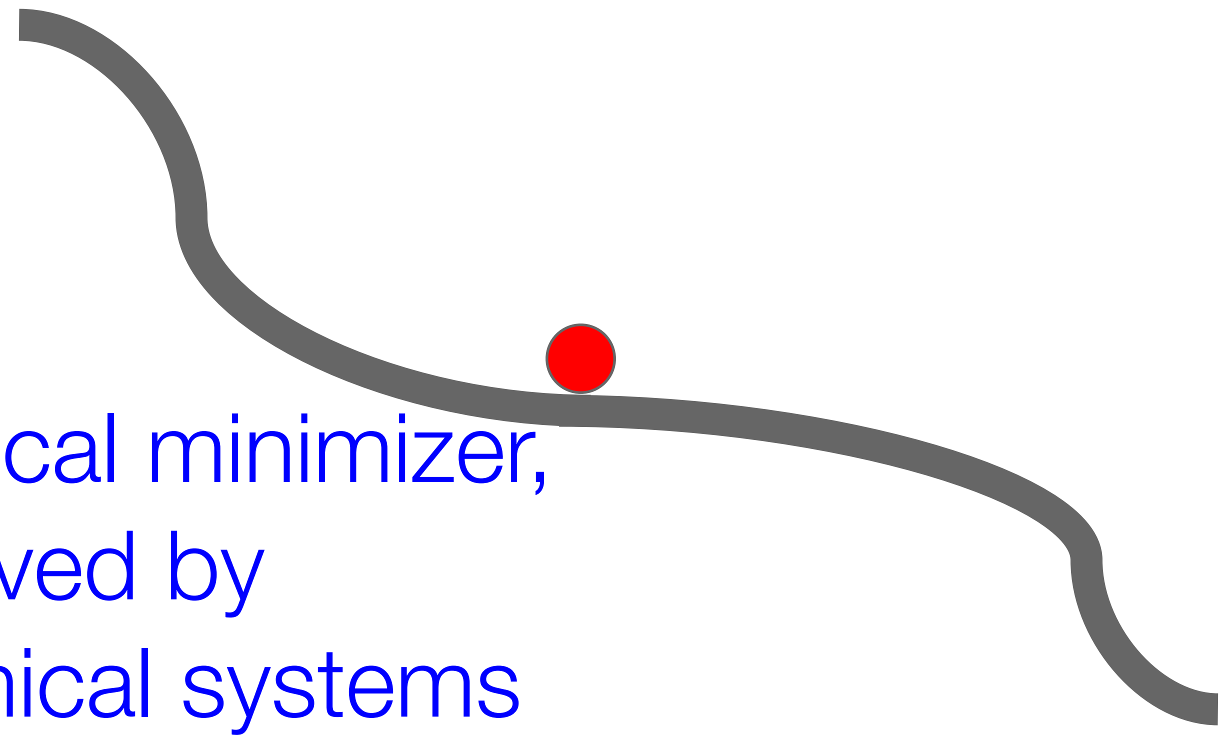
Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

Optimization: problem with SGD

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



Or not?



"We show that gradient descent converges to a local minimizer, almost surely with random initialization. This is proved by applying the Stable Manifold Theorem from dynamical systems theory."

Lee et al, "Gradient Descent Only Converges to Minimizers", JLMR Workshop and Conference Proceedings, 2016

Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

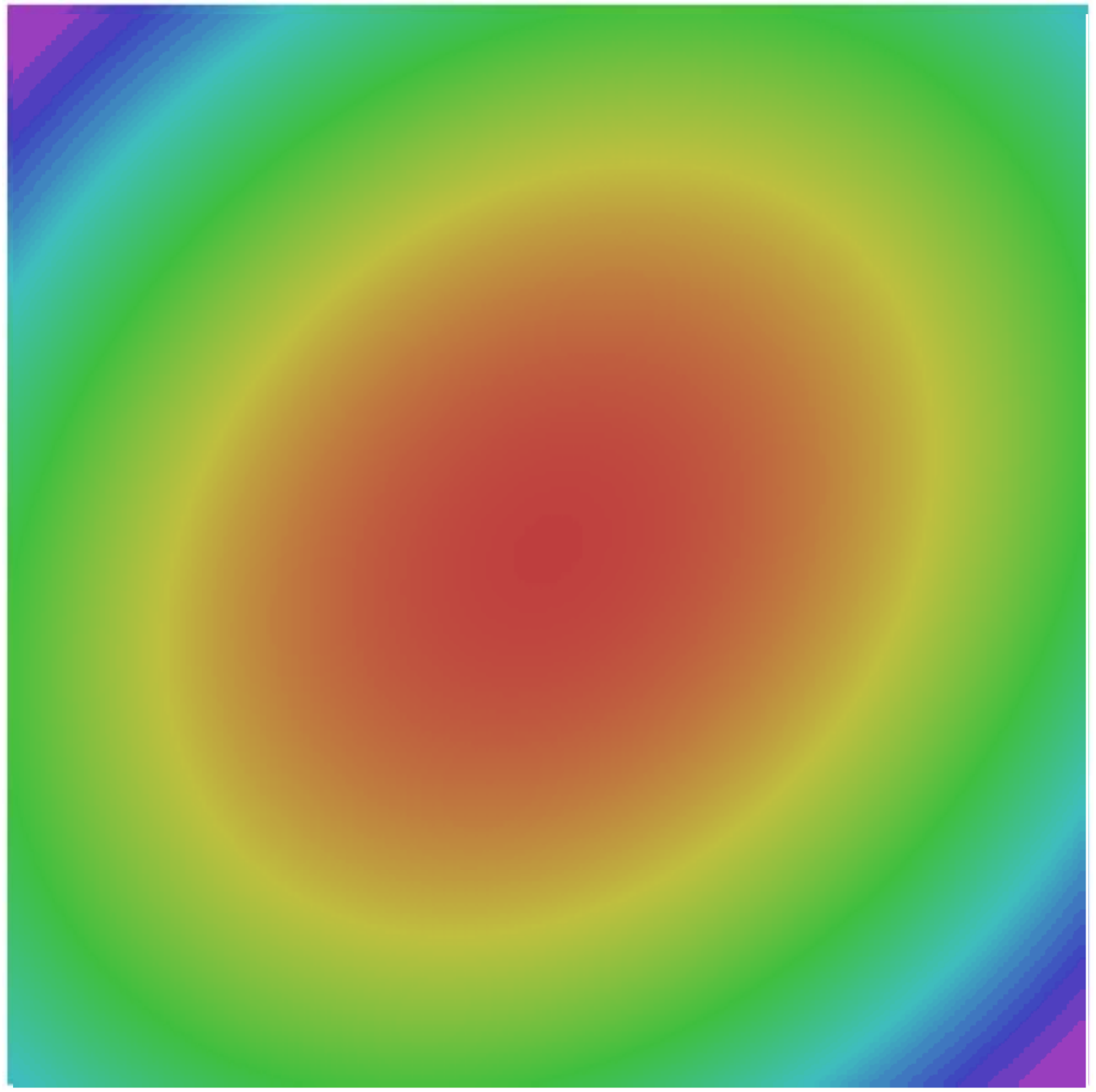
Stochastic gradient descent

Minibatches

Our gradients come from mini-batches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



Q: How would you remove the noise?

SGD + Momentum

SGD

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

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

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

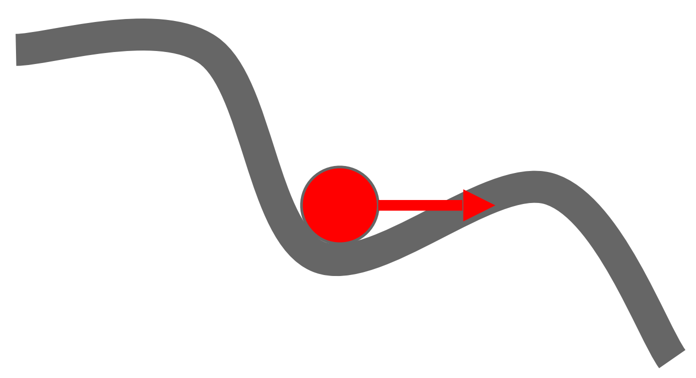
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

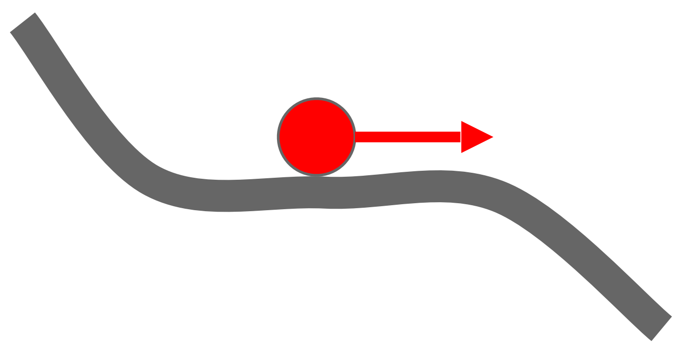
SGD + Momentum

Gradient Noise

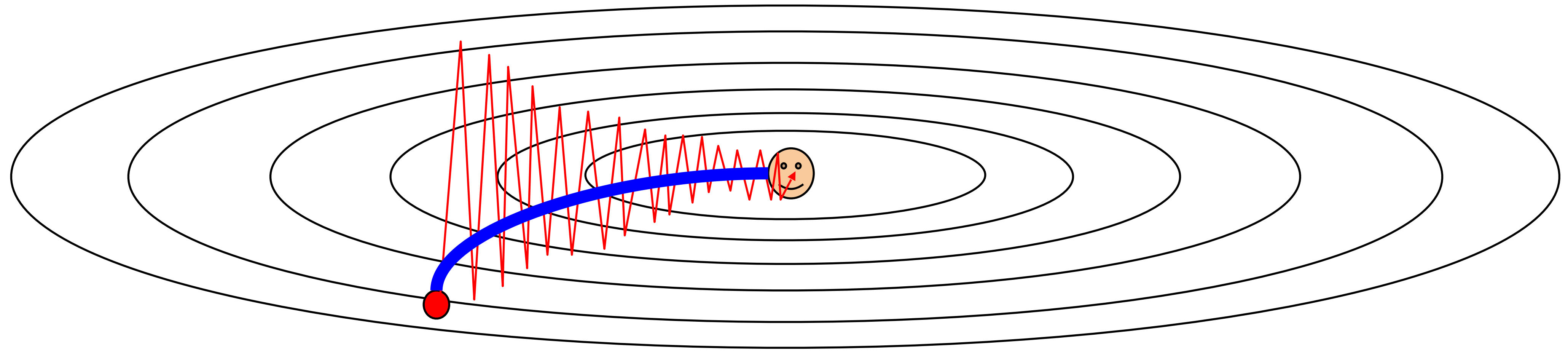
Local Minima



Saddle points

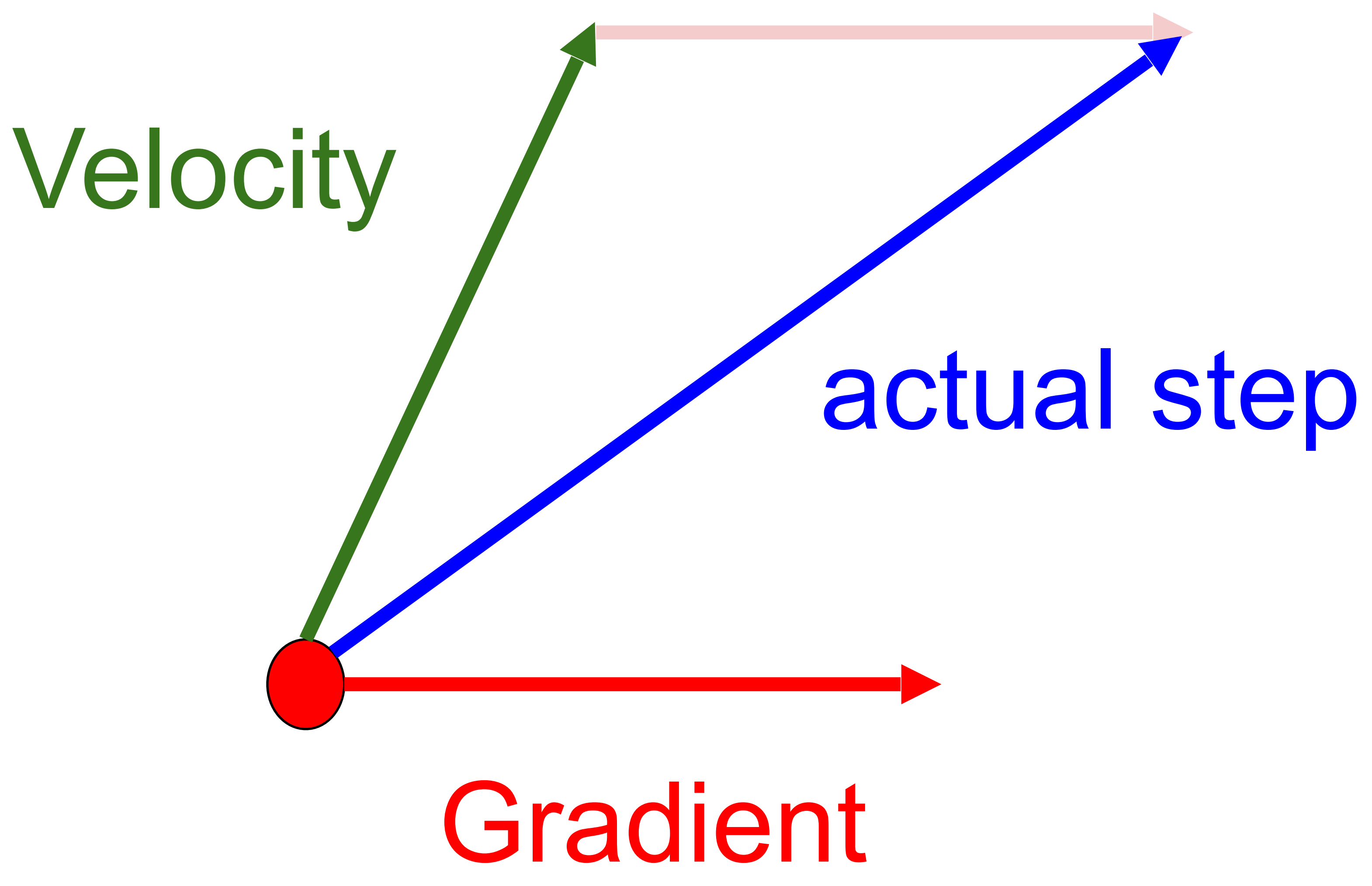


Poor Conditioning



SGD + Momentum

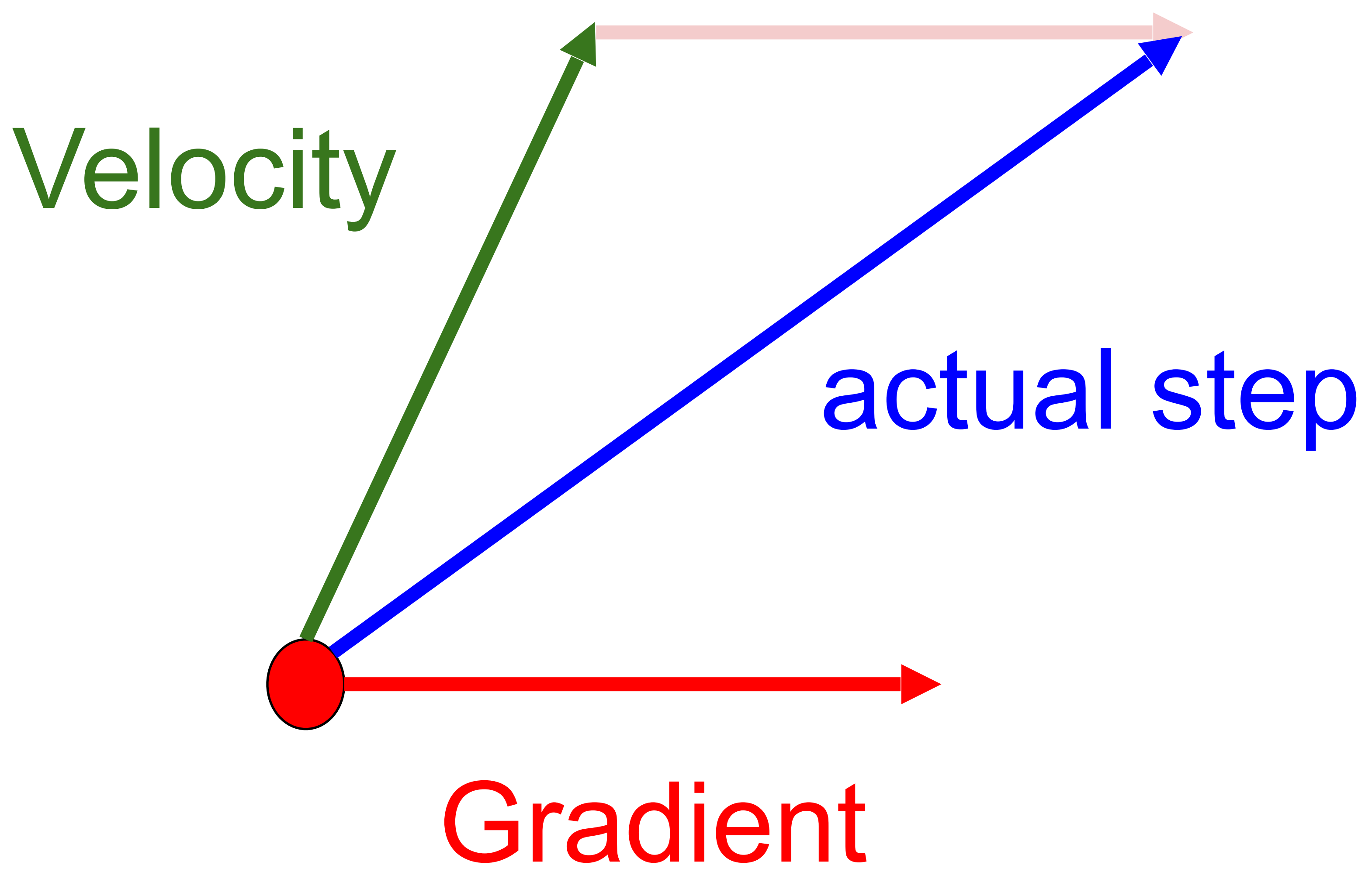
Momentum update:



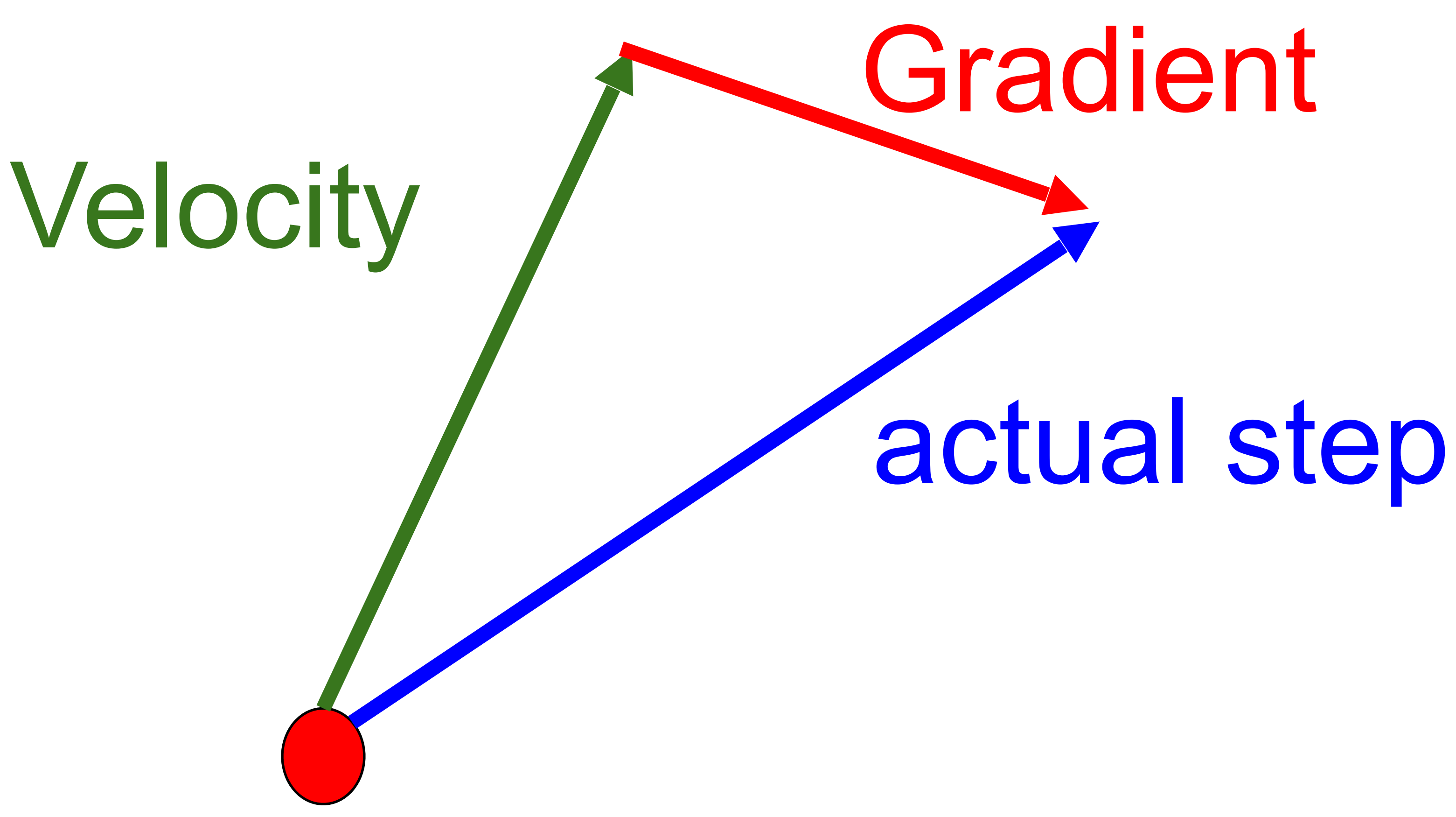
Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k^2)”, 1983
Nesterov, “Introductory lectures on convex optimization: a basic course”, 2004
Sutskever et al, “On the importance of initialization and momentum in deel learning”, ICML 2013

SGD + Momentum

Momentum update:



Nesterov Momentum



Nesterov, "A method of solving a convex programming problem with convergence rate O(1/k^2)", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deel learning", ICML 2013

Nesterov Momentum

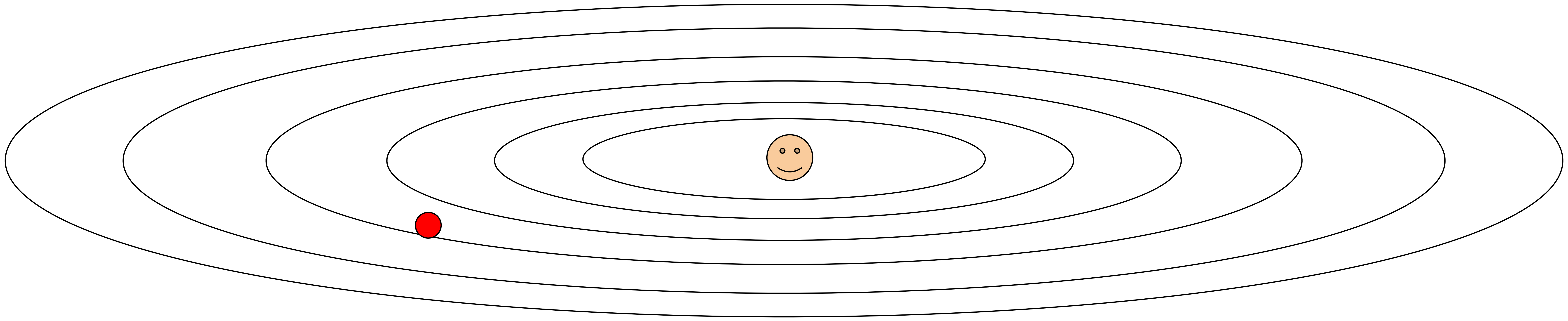
SGD

SGD+Momentum

Nesterov

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Q: What happens with RMSProp?

Tieleman and Hinton, 2012

RMSProp

- SGD
- SGD+Momentum
- RMSProp

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

RMSProp

RMSProp with momentum

Q: What happens at first the timestep?

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

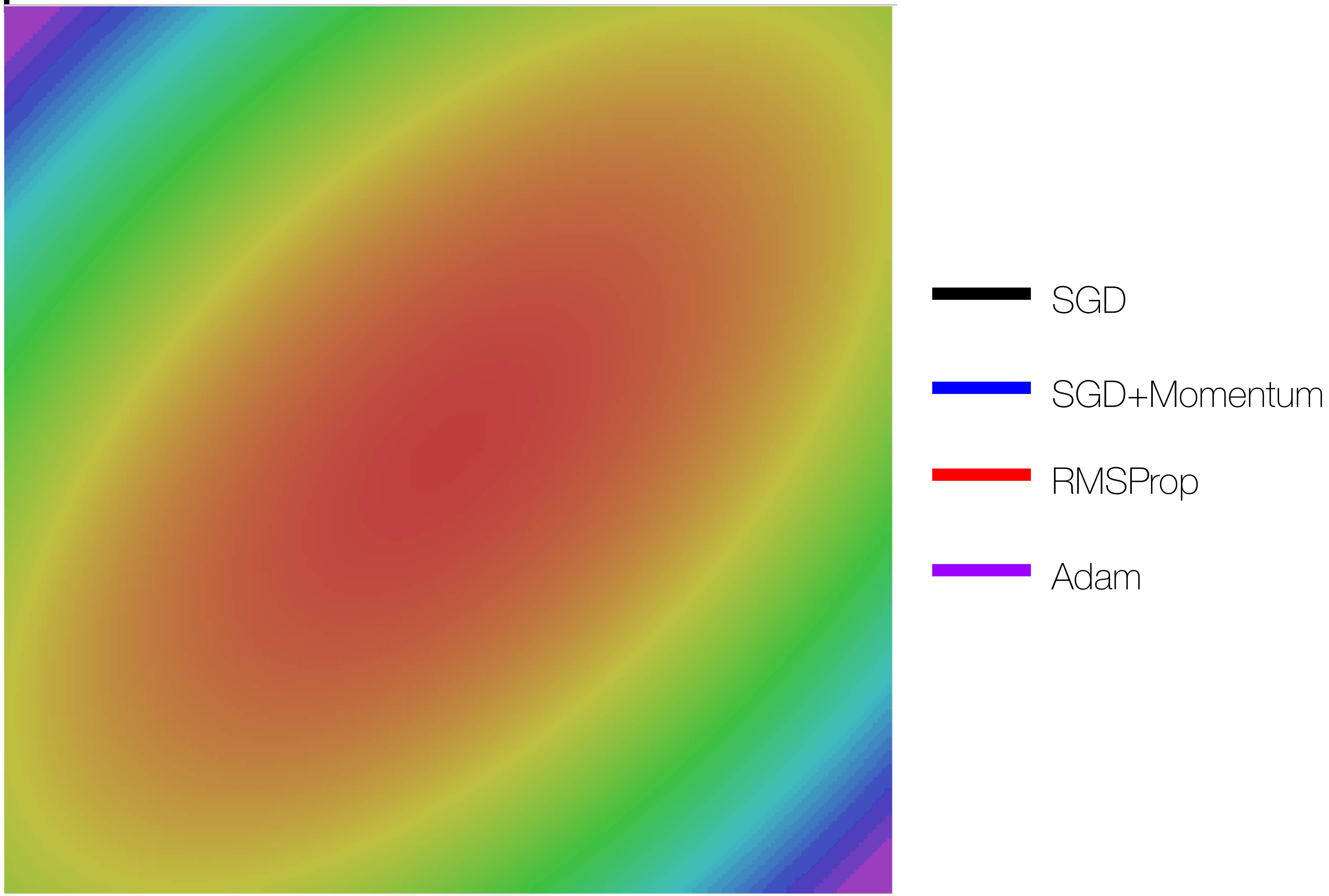
AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-4 is a great starting point for many models!

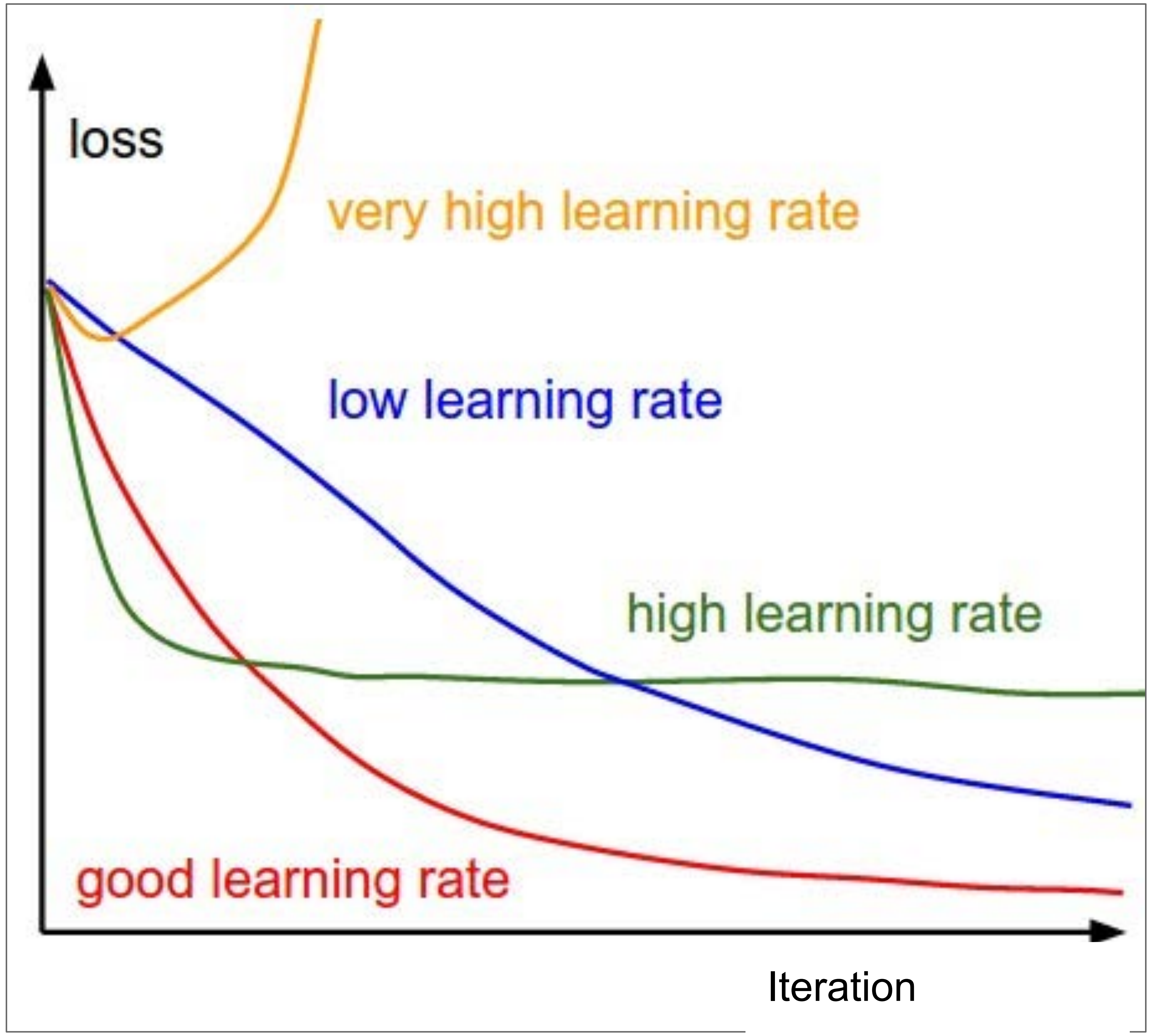
Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Adam



Learning rate: hyperparameter

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter



Based on slides for Stanford cs231n by Li, Jonson, and Young. Modified and reused with permission