Lecture 1 Overview of Computer Vision: Image Classification (Some basic concepts of ML/DL)

Tian Sheuan Chang

Source: CS231n: Convolutional Neural Networks for Visual Recognition

Outline

- Overview of image classification
- Two simple classifier
  - K nearest neighbor classifier
  - Linear classifier
- Loss function
  - SVM (support vector machine) loss
  - Softmax loss (Cross entropy loss)
- Optimization
  - Gradient descent

Source: CS231n

A 50 Years' Summer Project

- 1966: Minsky assigns computer vision as an undergrad summer project
  - linking a camera to a computer and getting the computer to describe what it saw

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Group July 7, 1966

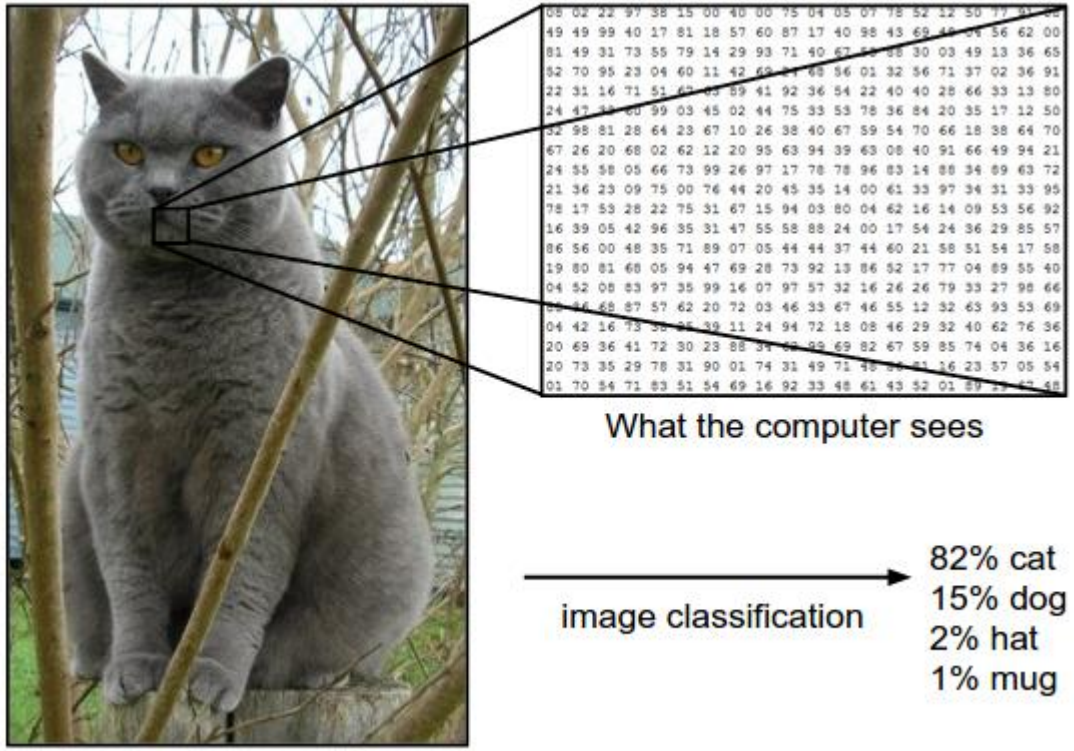
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

Image Classification: A core task in Computer Vision

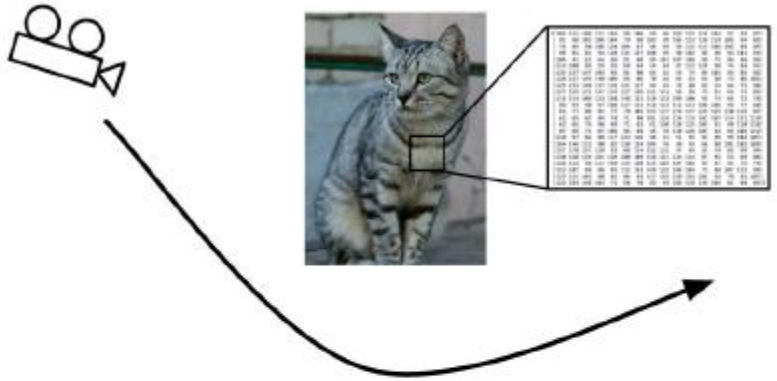
- What the computer sees
  - An image is just a big grid of numbers between [0, 255]
  - e.g. 800 x 600 x 3 (3 channels RGB)



- Problem: Semantic Gap

Why Computer Vision is So Challenging?

Viewpoint



Illumination

This image is CC0 1.0 public domain

Deformation

This image by Umberto Salvagnin is licensed under CC-BY 2.0

Occlusion

This image by jonsson is licensed under CC-BY 2.0

Clutter

This image is CC0 1.0 public domain

Intraclass Variation

This image is CC0 1.0 public domain

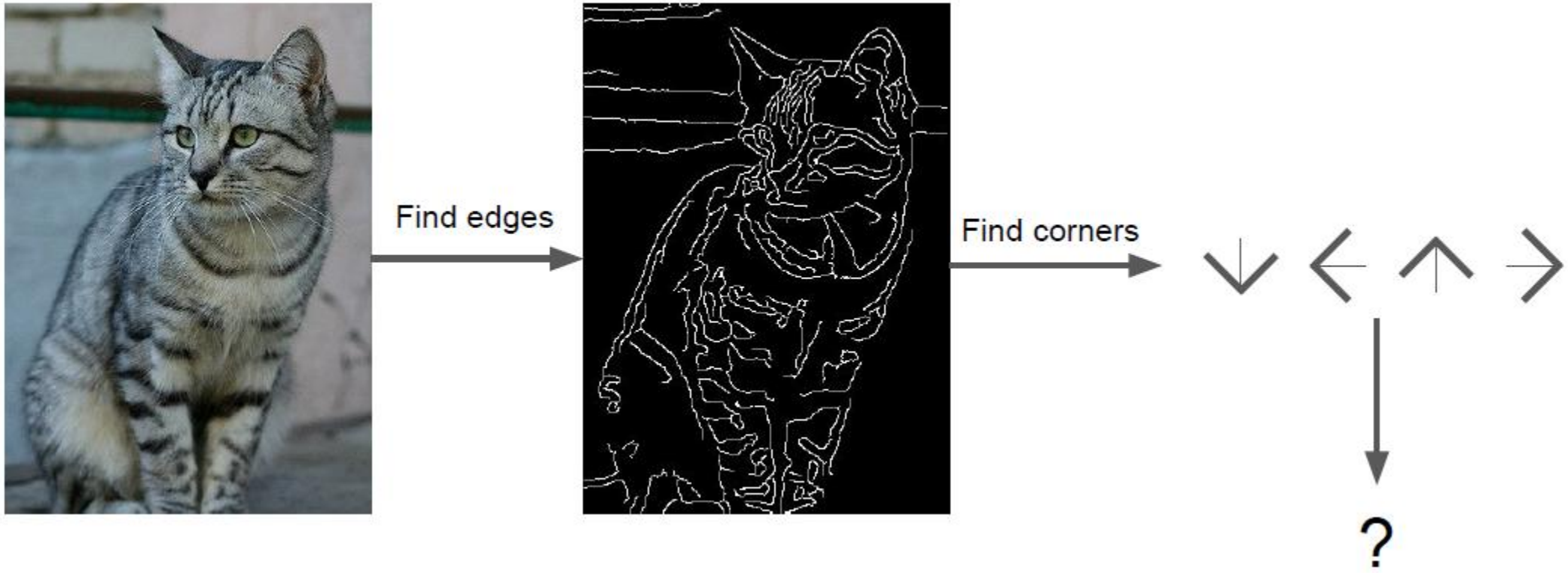
An Image Classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Attempts has been made



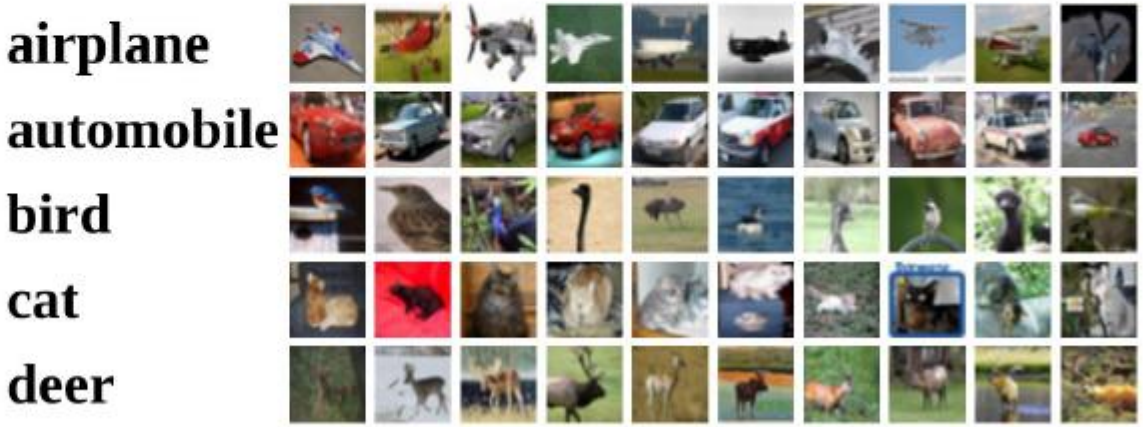
Data Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set



K NEAREST NEIGHBOR CLASSIFIER

First Classifier: Nearest Neighbor

```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

First classifier: Nearest Neighbor

deer bird plane cat car

Training data withlabels

?

query data

Distance Metric $\left| \text{ }, \text{ } \right| \rightarrow \mathbb{R}$

Loss Function to Compare Images

- L1 distance $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

Python Implementation of Nearest Neighbor

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Memorize training data

For each test image:
Find closest train image
Predict label of nearest image

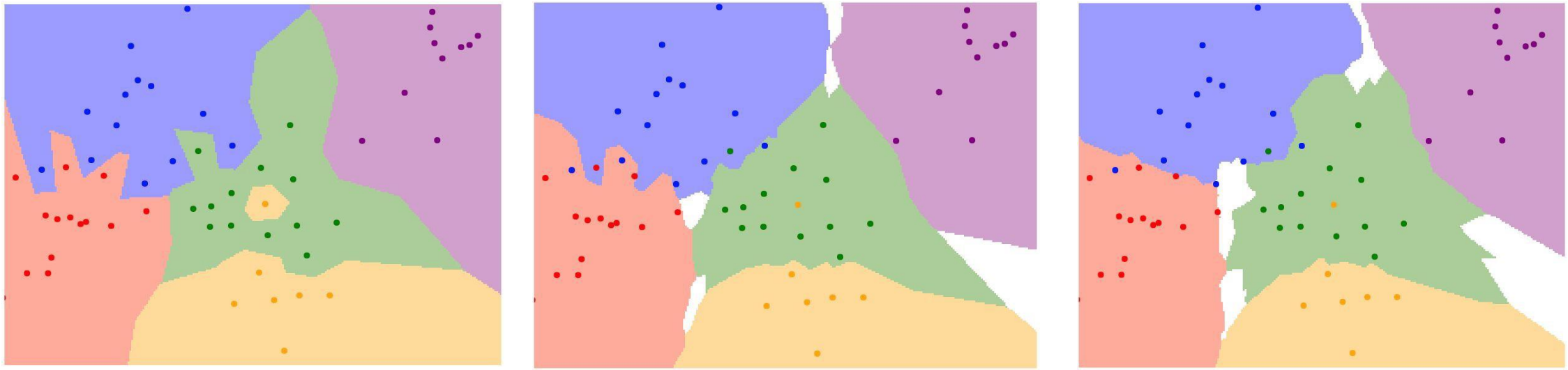
```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N examples, how fast are training and prediction?

- A: Train O(1), predict O(N)
- This is bad: we want classifiers that are fast at prediction; slow for training is ok

K-Nearest Neighbors

- Instead of copying label from nearest neighbor, take majority vote from K closest points
  - http://vision.stanford.edu/teaching/cs231n-demos/knn/



K = 1 K = 3 K = 5

Classification Results

- 38.6% on CIFAR-10 with KNN
- Better than random
  - 10% for 10 classes
- Human performance
  - ~94%
- State of the art CNN
  - 95%

- L2 distance does not help

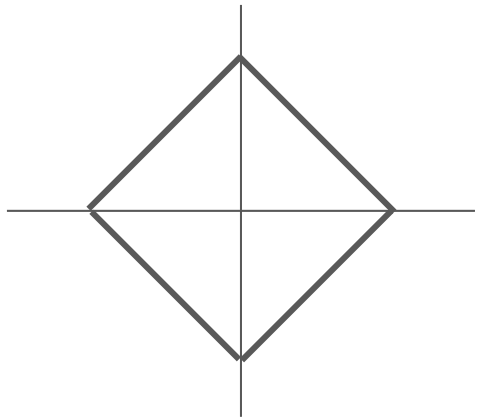
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

- top 10 nearest neighbors in the training set according to pixel-wise difference

Distance Metric

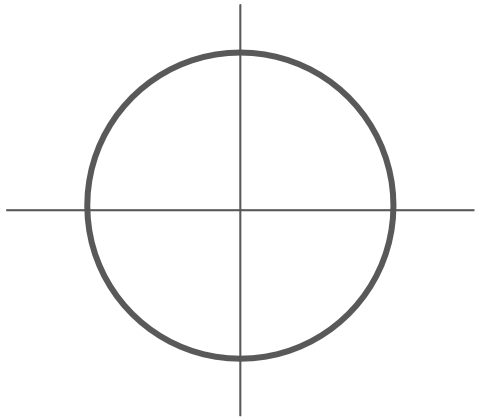
- L1 distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



- L2 distance

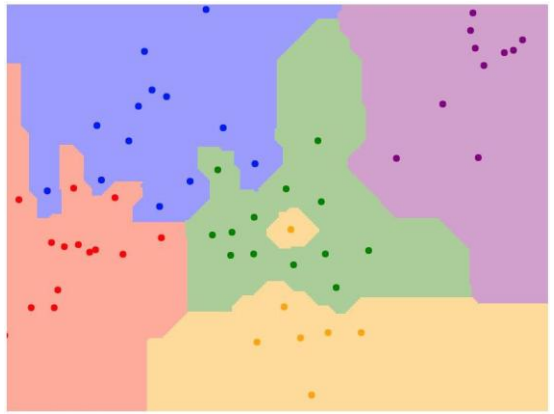
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



Distance Metric: K = 1

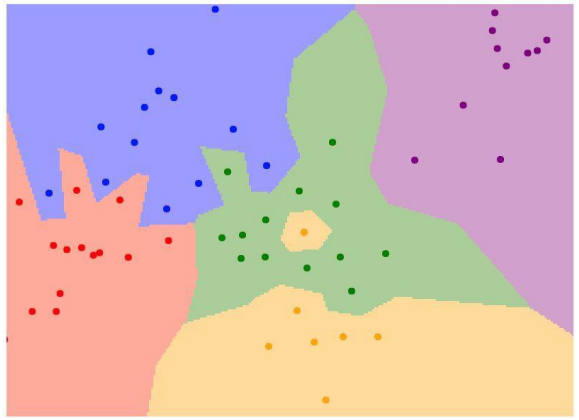
- L1 distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



- L2 distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



Hyperparameters

- What is the best value of k to use?
- What is the best distance to use?

- These are hyperparameters: choices about the algorithm that we set rather than learn

- Very problem-dependent. Must try them all out and see what works best.

Setting Hyperparameters: K and Loss Function

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train	test

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

train	validation	test

Setting Hyperparameters

Your Dataset

Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

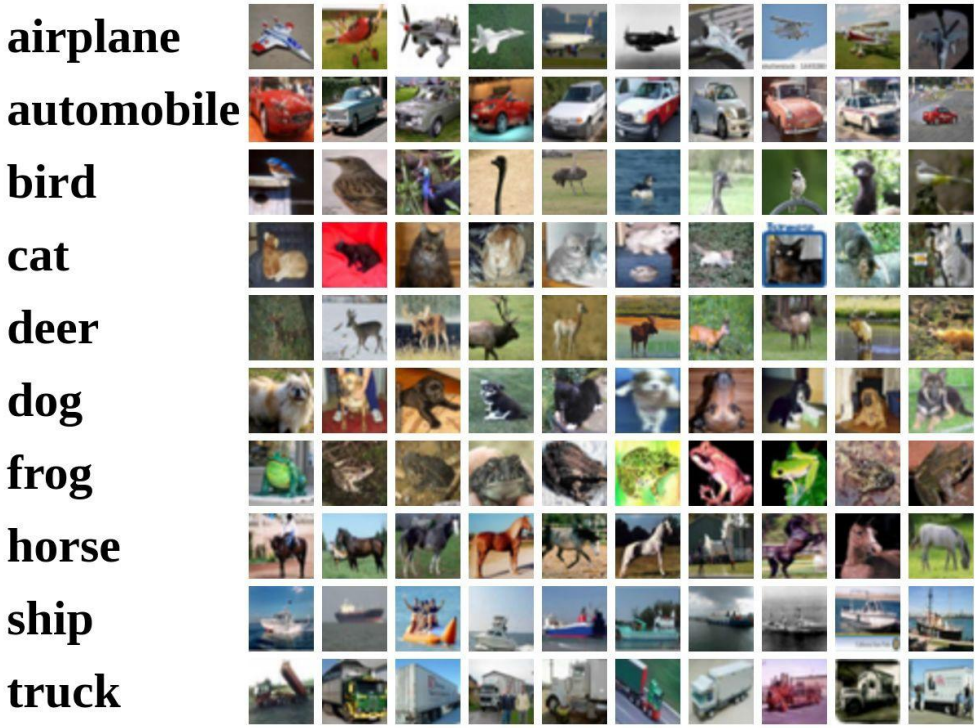
Useful for small datasets, but not used too frequently in deep learning

Terminology: Data set

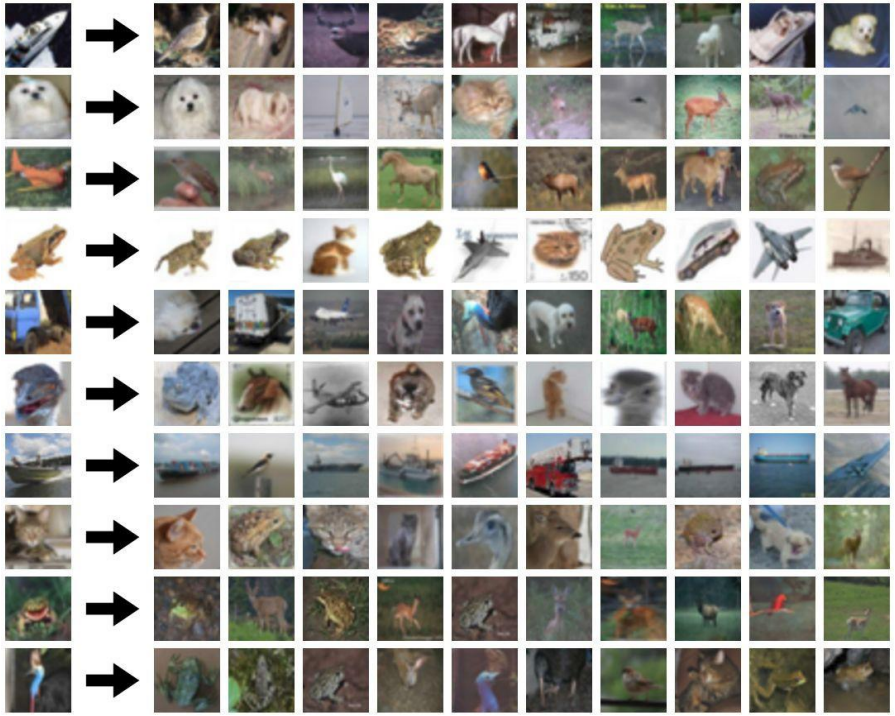
- Training set: contains inputs and labels
  - to train the models
  - Run many times
- Test set: use model to predict labels on the test set
  - Run only once at the end
- Choose hyperparameters using the validation set; only run on the test set once at the very end

Example Dataset: CIFAR-10

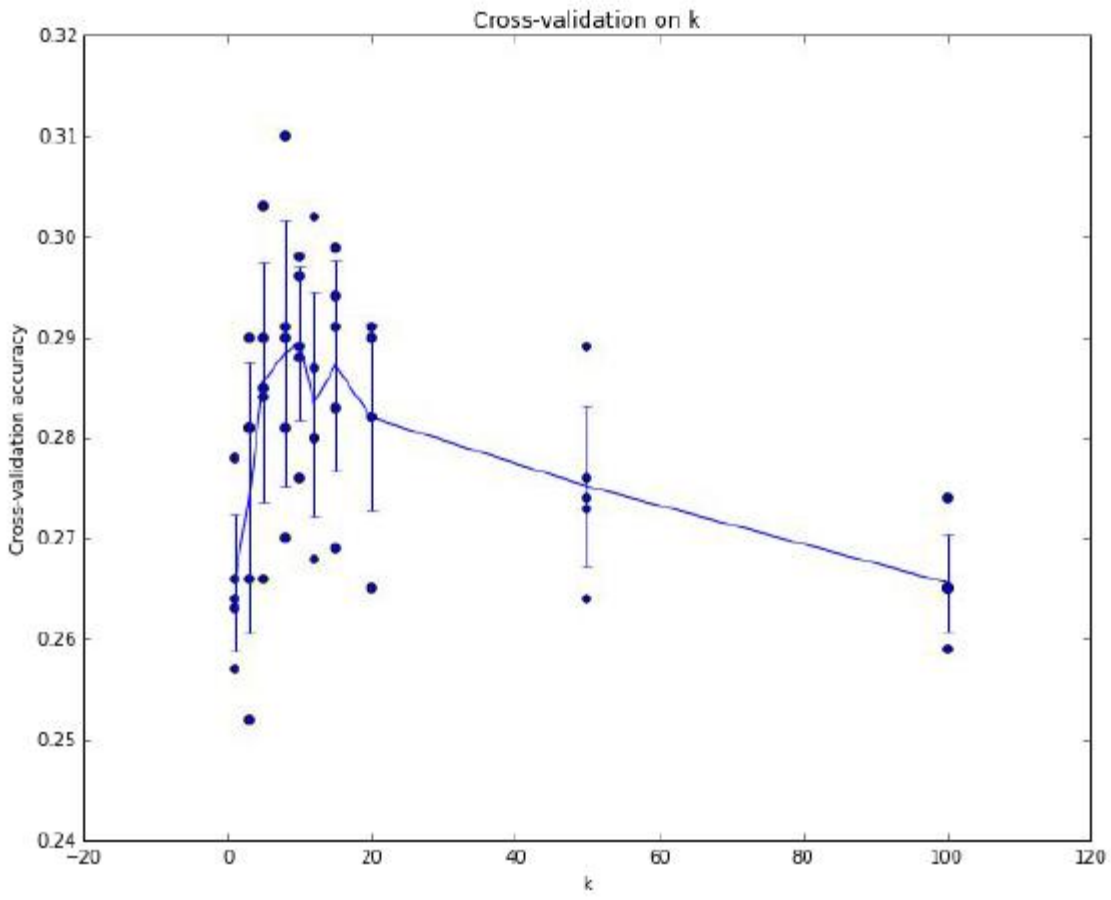
- 10 classes
- 50,000 training images
- 10,000 testing images



- Test images and nearest neighbors



Setting Hyperparameters

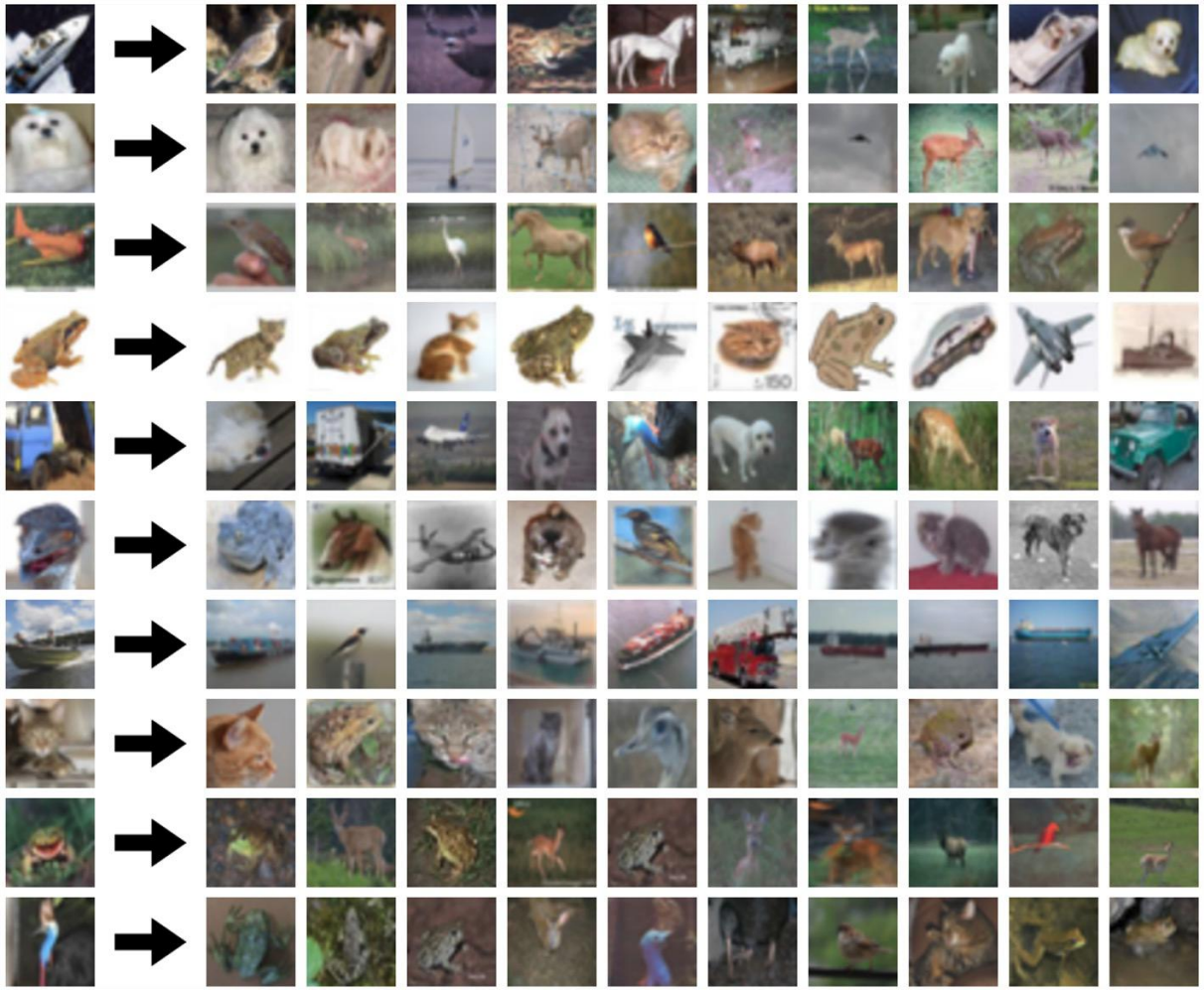


- Example of 5-fold cross-validation for the value of k.
- Each point: single outcome.
- The line goes through the mean, bars indicated standard deviation
- (Seems that k ~= 7 works best for this data)

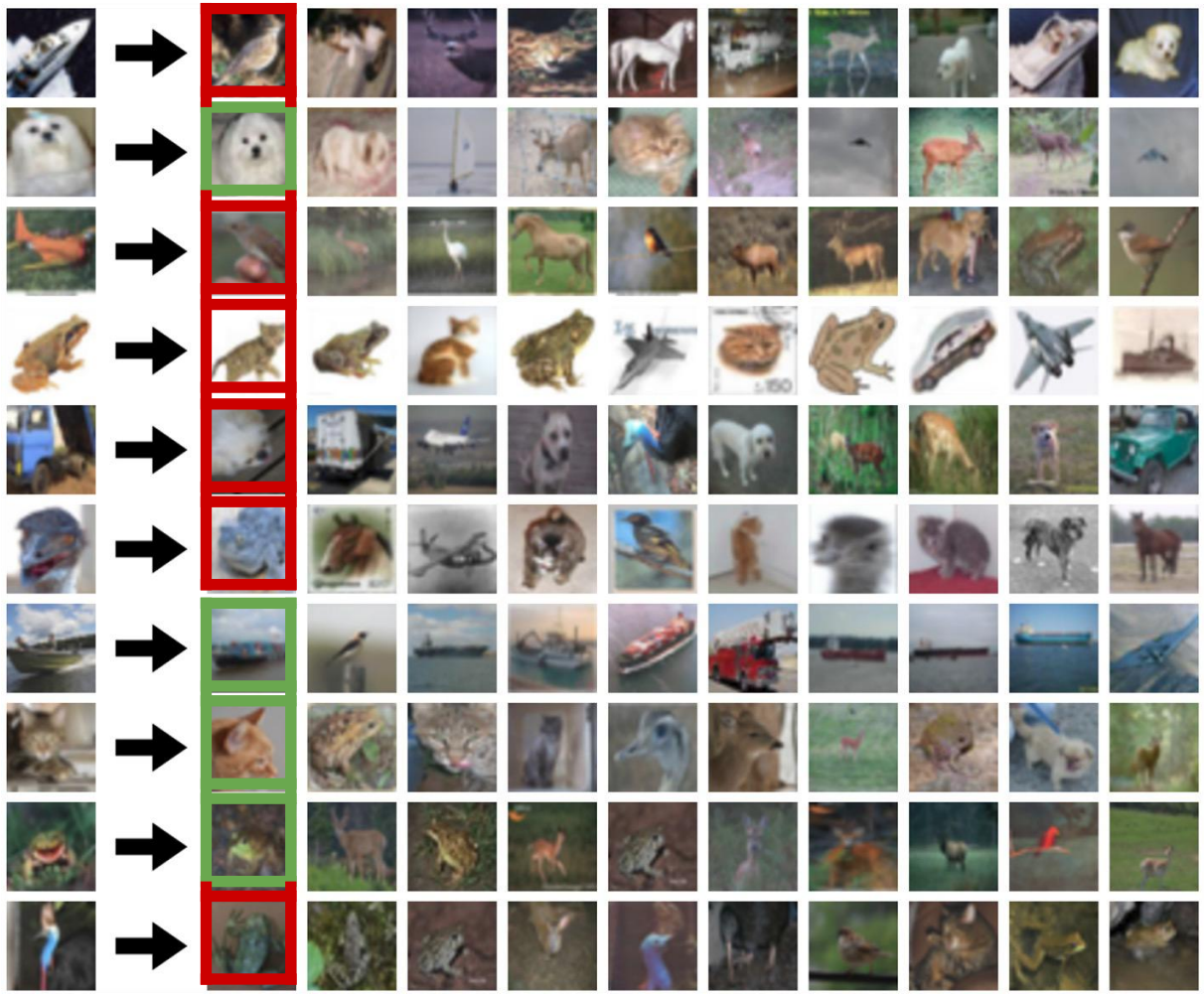
Summary: Data Driven Approach

- Image classification problem
  - Given training set, predict its category for a new set of image
- K-Nearest neighbor classifier
  - No obvious way to choose the hyperparameters
- Correct way to choose hyperparameters
  - Split training data into
    - training set: to train the model
    - Validation set: to tune the parameters of model
    - Test set

What does this look like?



What does this look like?



k-Nearest Neighbor with pixel distance never used.

- Distance metrics on pixels are not informative

Original image is CC0 public domain

Original

Occluded

Shifted (1 pixel)

Tinted

(All three images on the right have the same pixel distances to the one on the left)

LINEAR CLASSIFIER

Parametric Approach: Linear Classifiers

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(x,W)$

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifiers

$$\boxed{f(x,W)} = \boxed{W}\boxed{x}$$

3072x1

10x1 10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(\mathbf{x},\mathbf{W})$

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifiers

$$\underbrace{f(x,W)}_{10\times 1} = \underbrace{W}_{10\times 3072}\overbrace{x}^{3072\times 1} + \underbrace{b}_{10\times 1}$$

Image

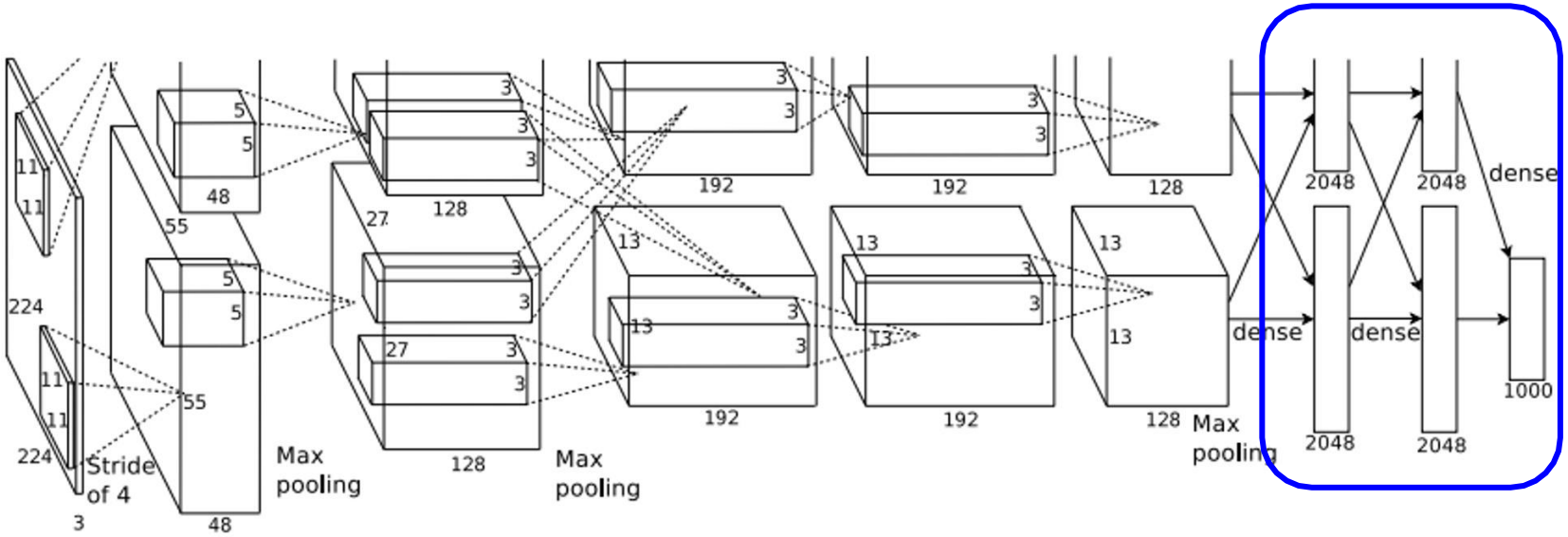
Array of **32x32x3** numbers
(3072 numbers total)

$$\longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow$$

10 numbers giving
class scores

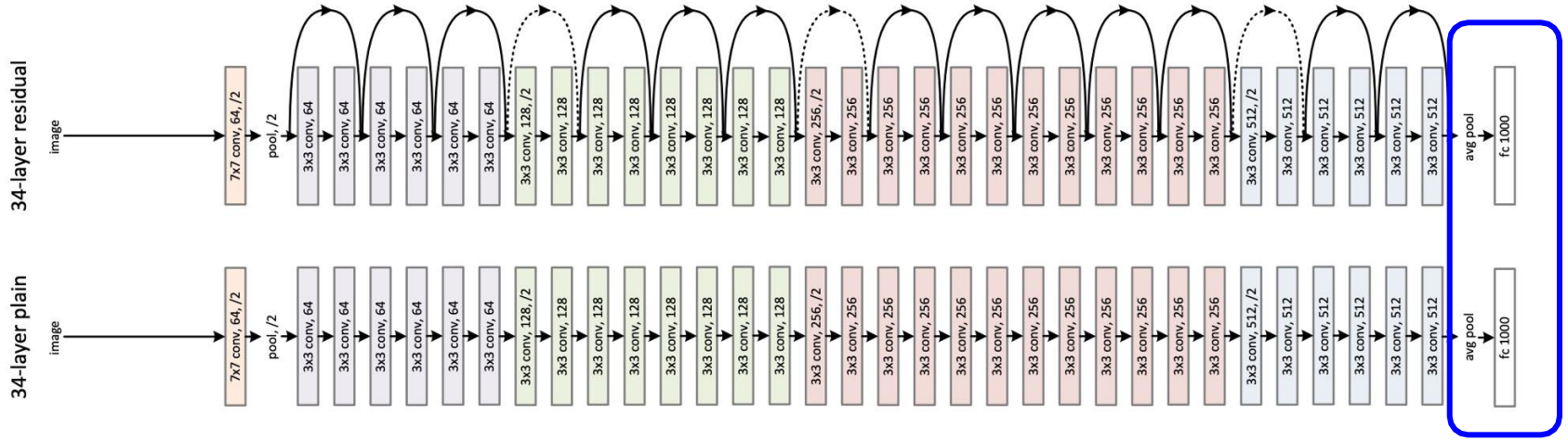
W
parameters
or weights

Long training time,
But short test time



[Krizhevsky et al. 2012]

Linear layers



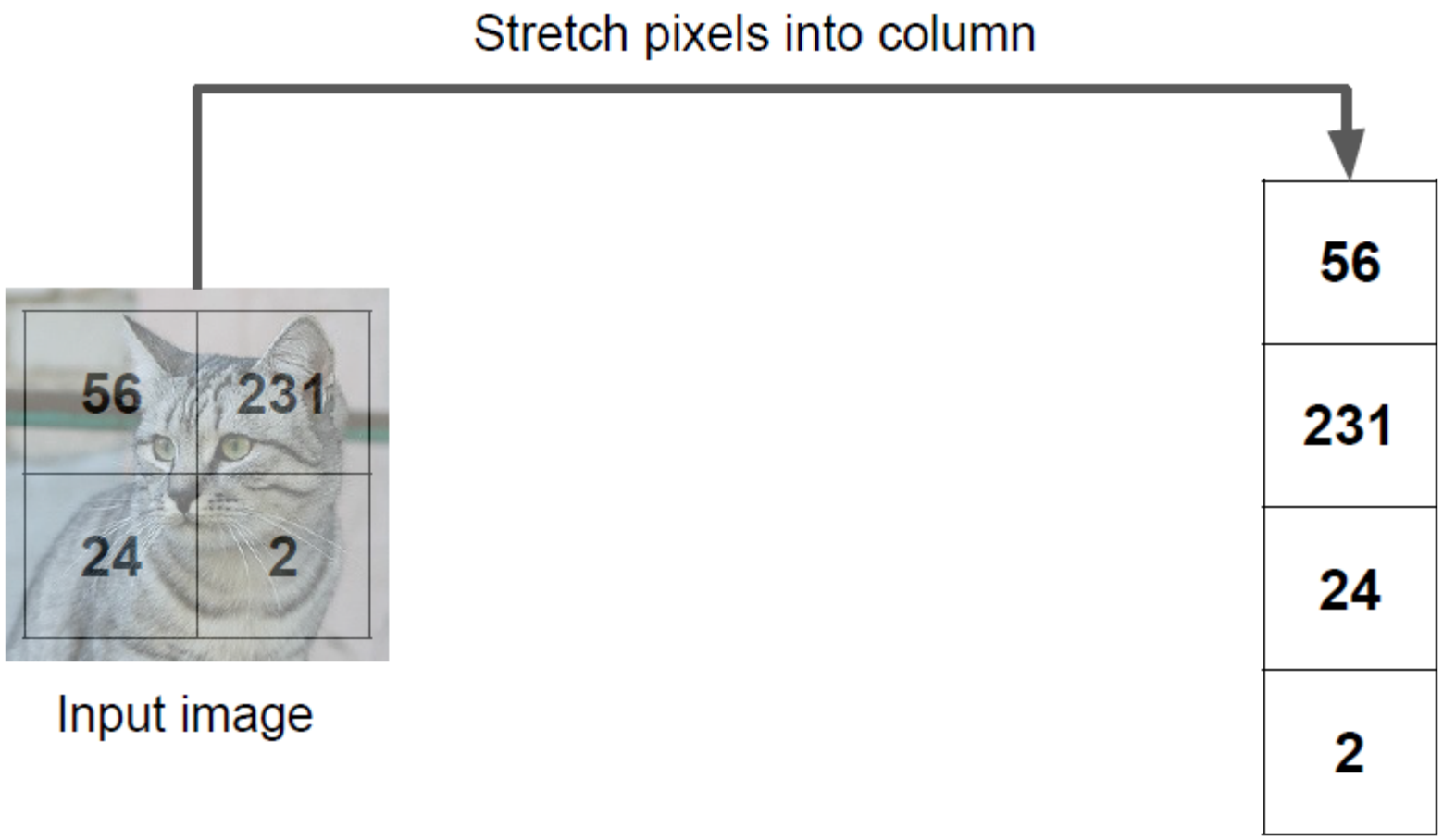
[He et al. 2015]

Recall CIFAR10

50,000 training images
each image is 32x32x3

10,000 test images.

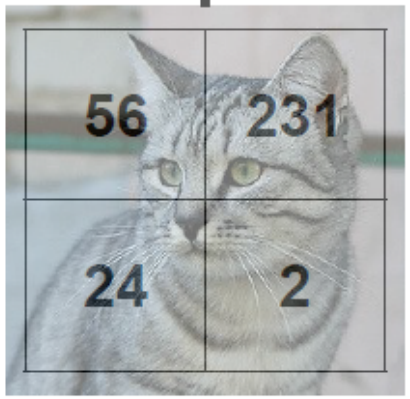
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



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

Stretch pixels into column

A bad weight: a dog?



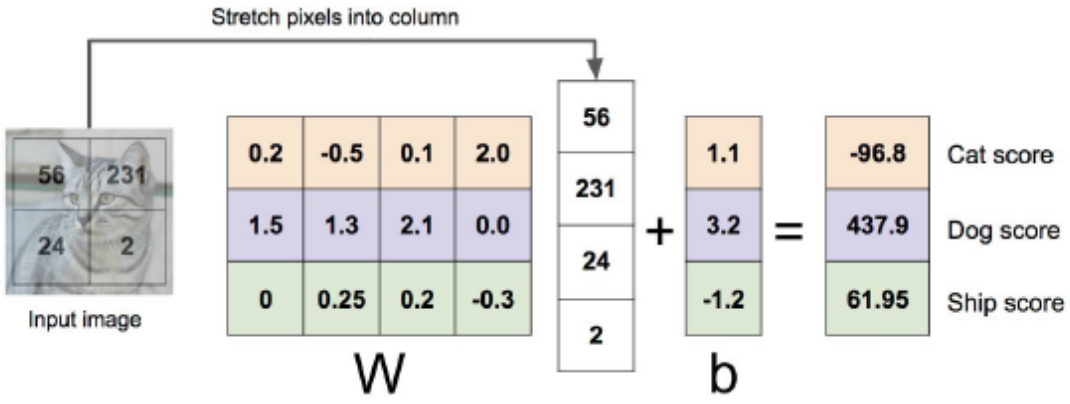
Input image

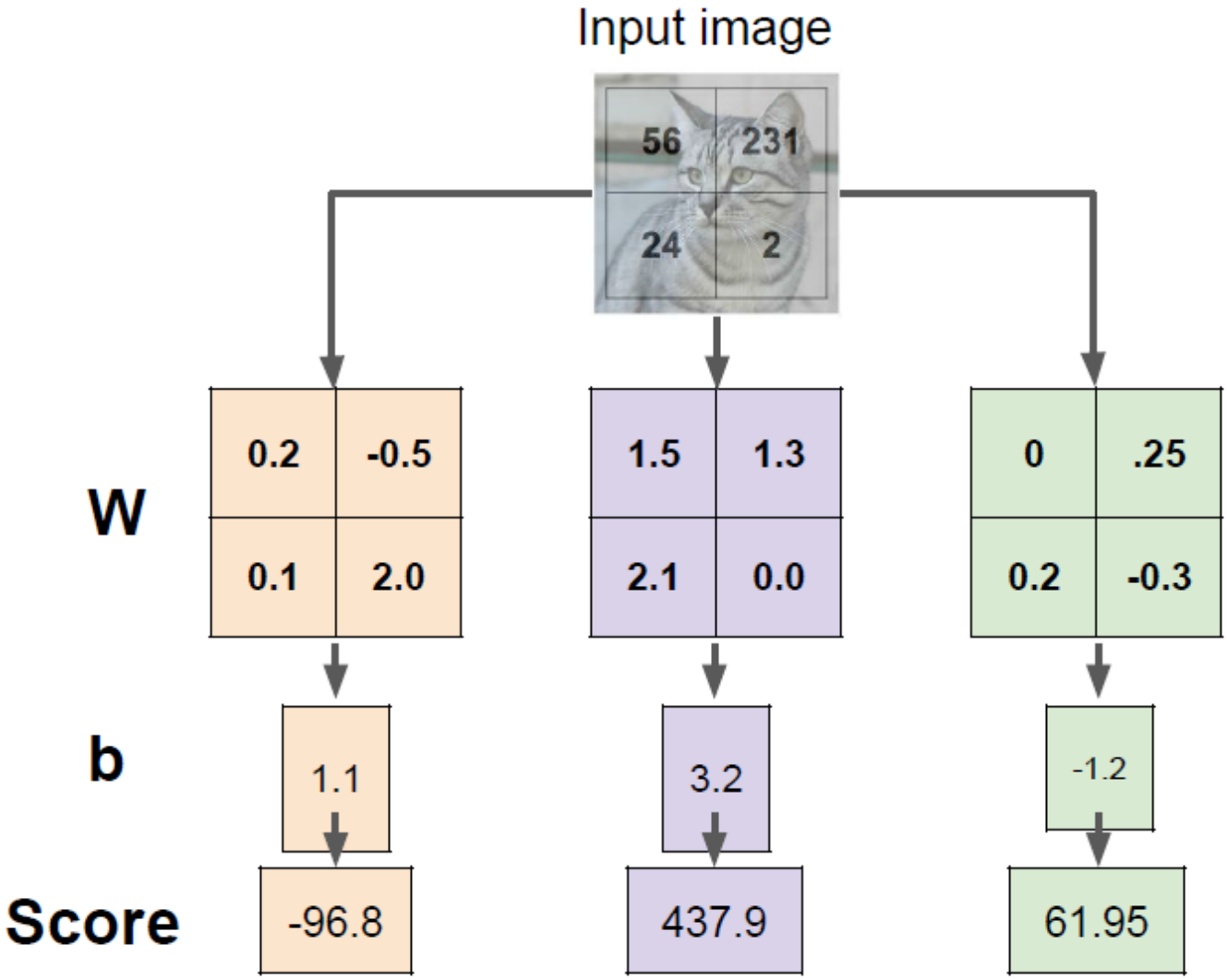
$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \begin{matrix} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{matrix}$$

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

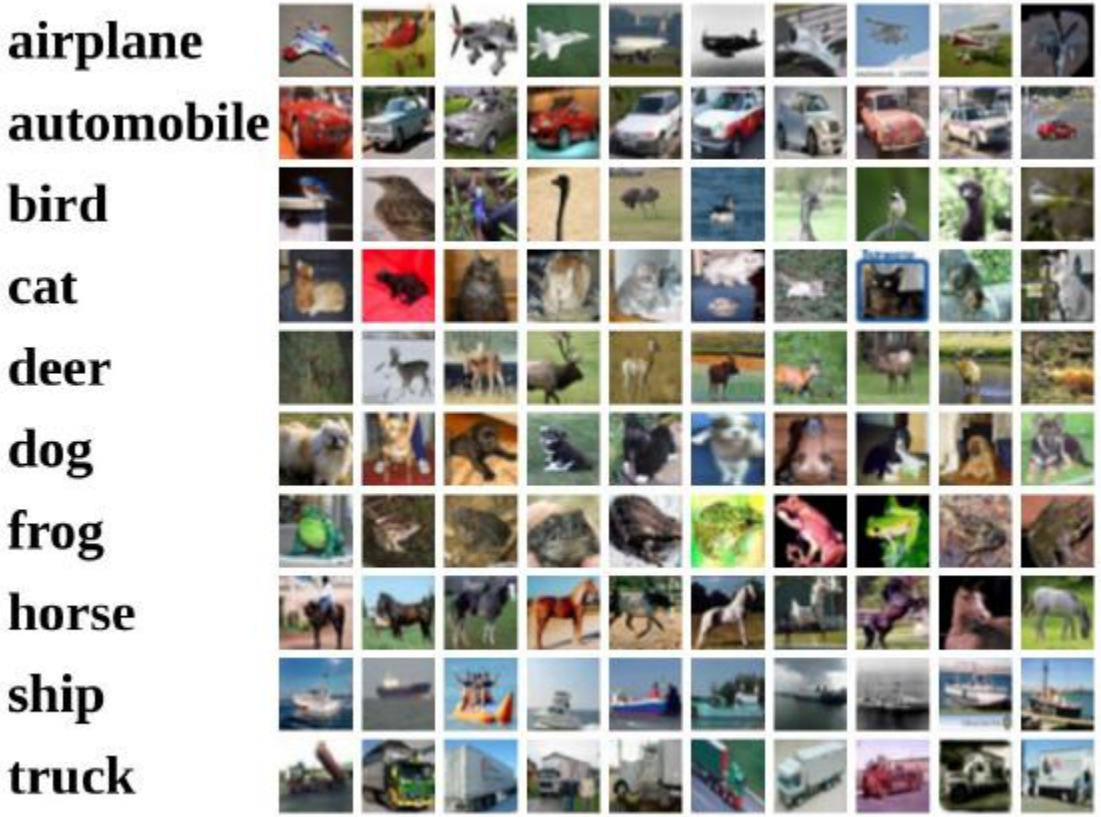
Algebraic Viewpoint

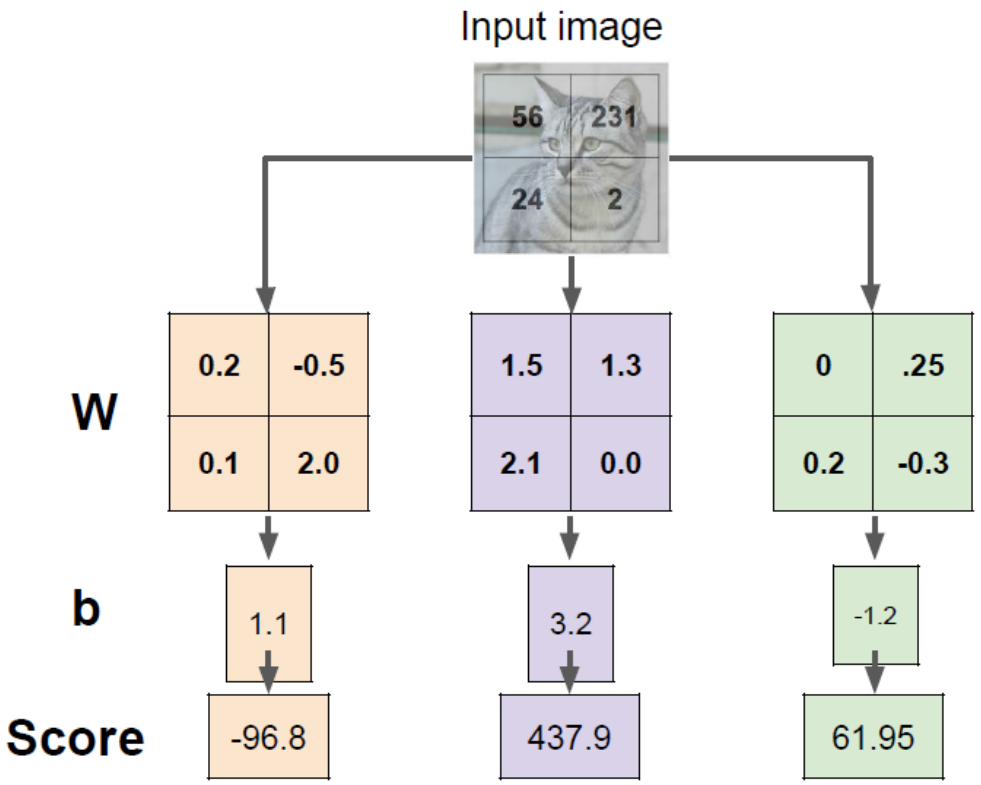
$$f(x,W) = Wx$$





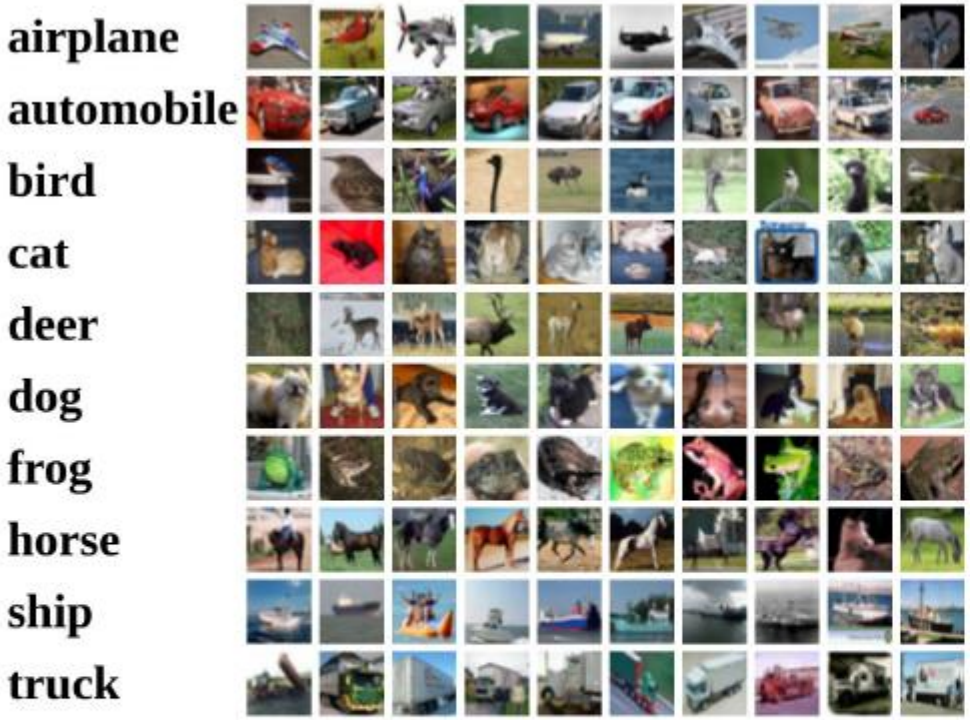
Interpreting a Linear Classifier: Visual Viewpoint





Interpreting a Linear Classifier

What is this thing doing?: Template Matching

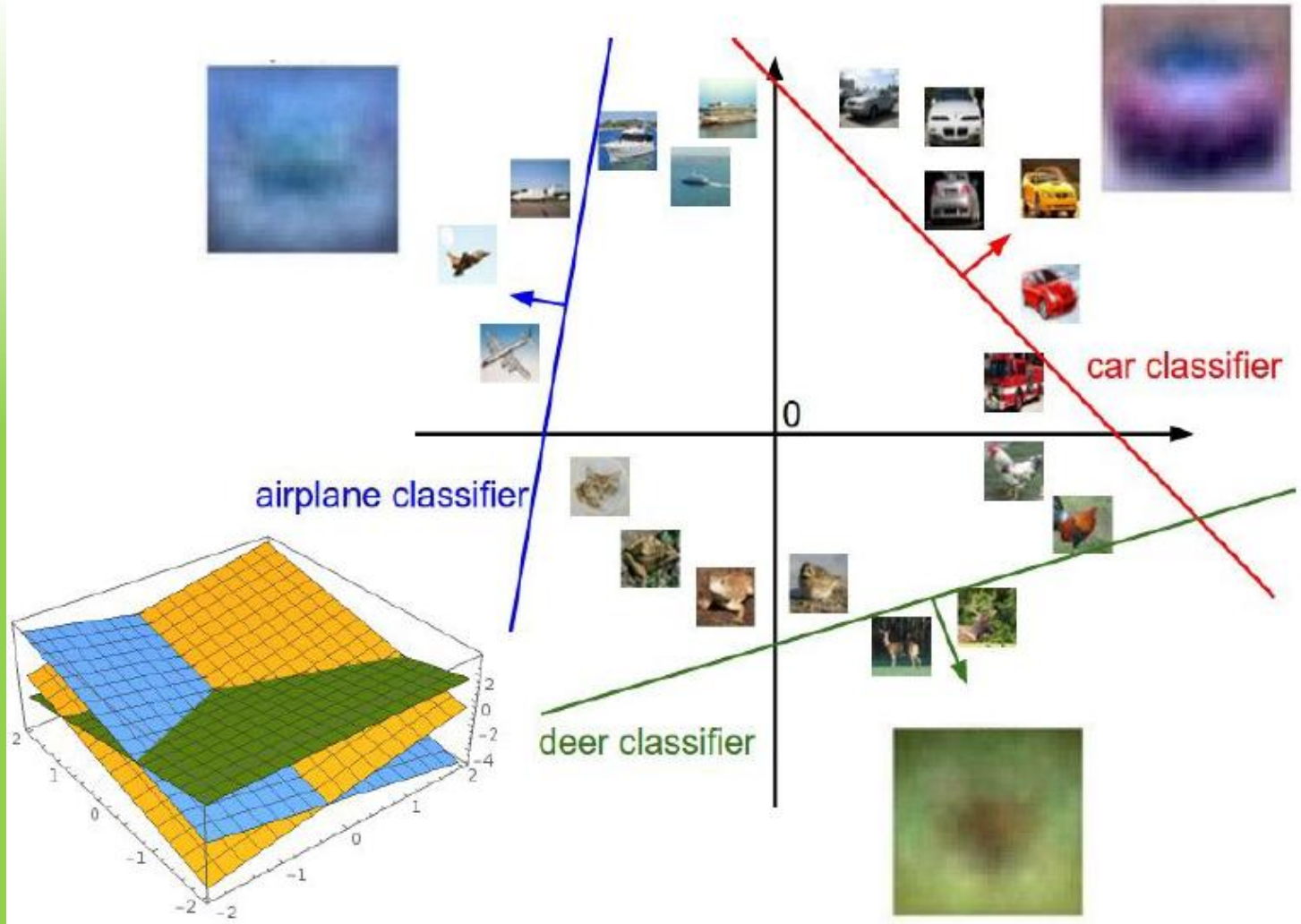


Inner product

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

Example trained weights of a linear classifier trained on CIFAR-10:

Interpreting a Linear Classifier: Geometric Viewpoint



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

Array of **32x32x3** numbers
(3072 numbers total)

Bias Trick: Simplified Representation

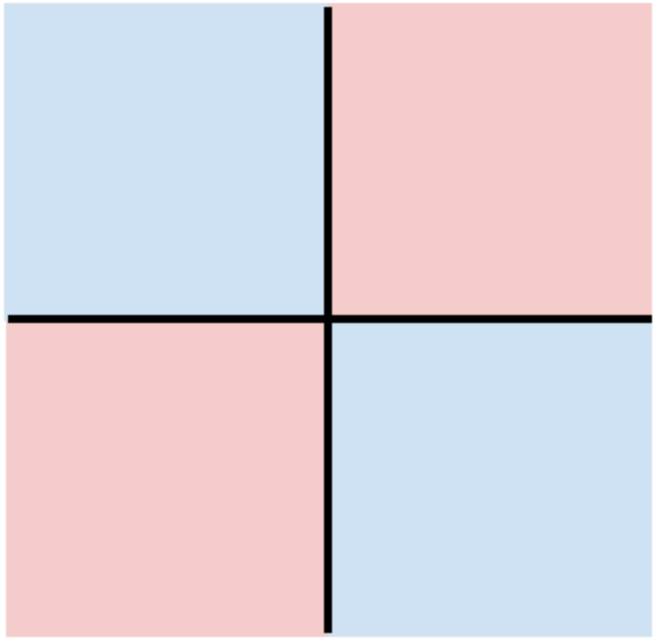
- extending the vector x_i with one additional dimension that always holds the constant 1 - a default bias dimension.
  - One matrix instead of two matrix for matrix multiplication

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longleftrightarrow \underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 & 1.1 \\ 1.5 & 1.3 & 2.1 & 0.0 & 3.2 \\ 0 & 0.25 & 0.2 & -0.3 & -1.2 \end{bmatrix}}_{\text{new, single } W} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \\ 1 \end{bmatrix}$$

Hard cases for a linear classifier

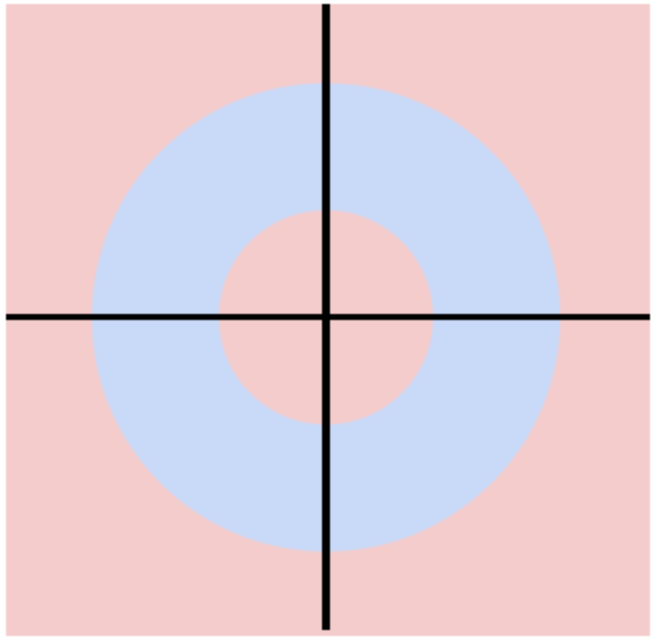
Class 1:
number of pixels > 0 odd

Class 2:
number of pixels > 0 even



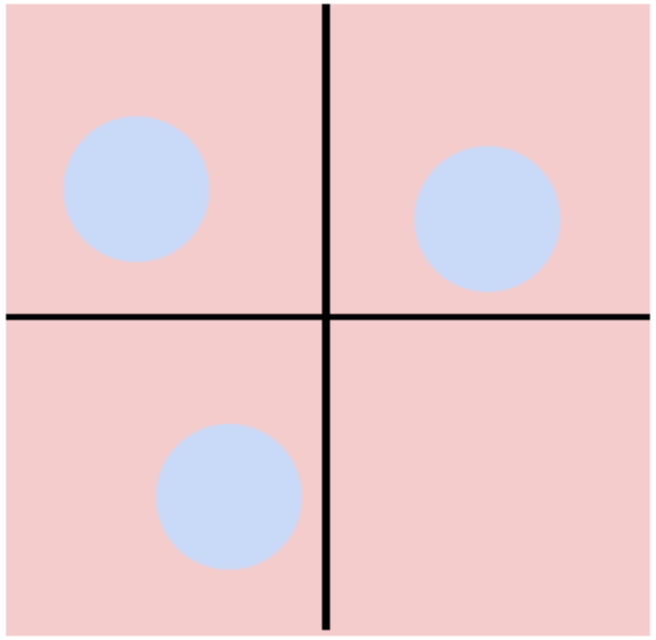
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



So Far: Defined a (linear) score function

$f(x. W) = Wx + b$

Example class scores for 3 images for some W:

How can we tell whether this W is good or bad?

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

ge by Nikita is licensed under CC-BY 2.0
ge is CC0 1.0 public domain
age is in the public domain

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

t image by Nikita is licensed under CC-BY 2.0; Car image is CC0 1.0 public domain; Frog image is in the public domain

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

LOSS FUNCTION

Loss Function

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

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

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Loss Function (Cost Function/Objective)

- The loss function quantifies our unhappiness with predictions on the training set

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

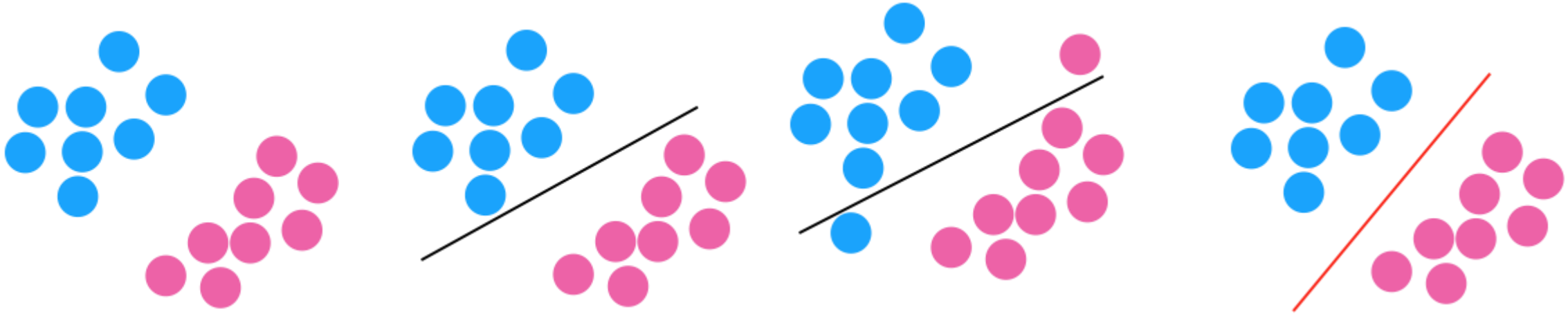
and using the shorthand for the scores vector: $s = f(x_i, W)$

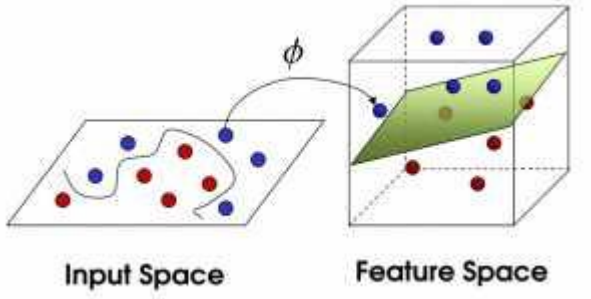
the SVM loss has the form:

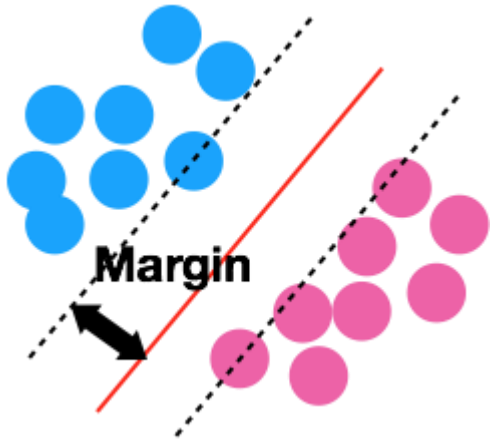
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

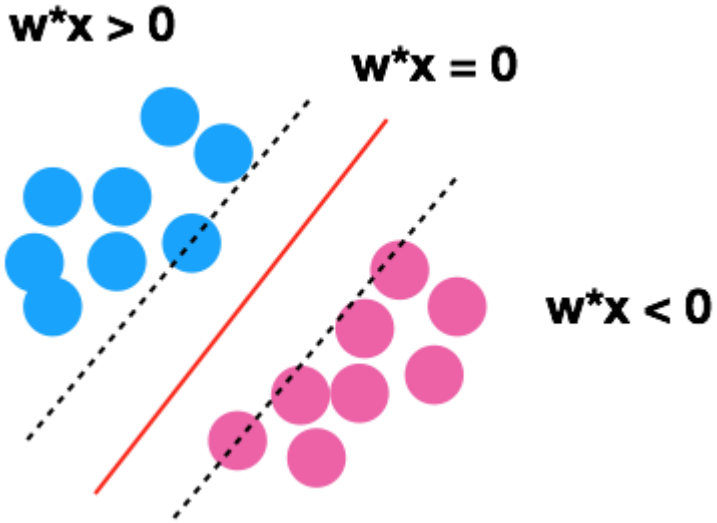
= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

SVM









Bug in Above Loss Function: W is not Unique, Choose W or 2W or 3W?

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

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

Before:

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

With W twice as large:

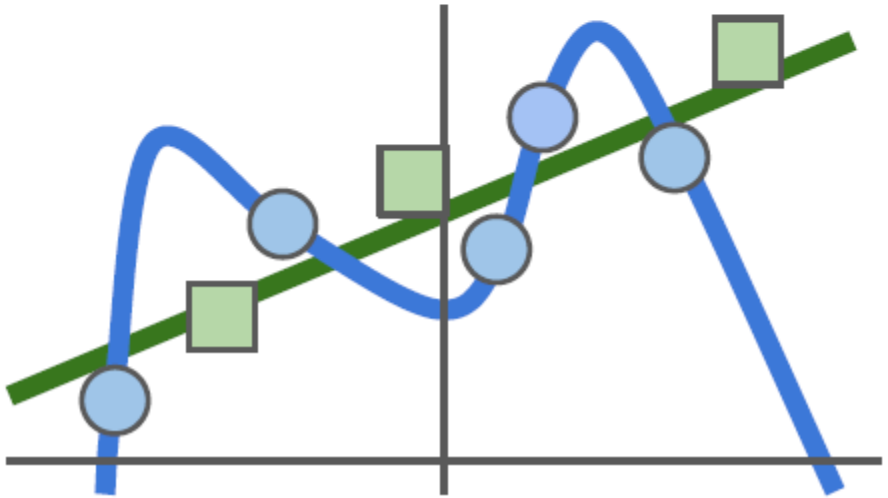
= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

Regularization

λ = regularization strength (hyperparameter)

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Max norm regularization (might see later)

Dropout (will see later)

Fancier: Batch normalization, stochastic depth

L2 Regularization (Weight Decay)

- Strong penalty than L1 (prefers smaller and more diffuse weight)
  - penalizing large weights tends to improve generalization
  - no input dimension can have a very large influence on the scores all by itself

$$x = [1, 1, 1, 1]$$

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Softmax Classifier (Multinomial Logistic Regression)

- Want to interpret raw classifier scores as probabilities

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

exp → normalize →

unnormalized log probabilities

probabilities

L_i = -log(0.13)
= **0.89**

Maximize probability of correct class

Softmax Classifier: Cross Entropy Loss

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

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

	Unnormalized log-probabilities / logits	exp →	unnormalized probabilities	normalize →	probabilities	compare	Correct probs
cat	3.2		24.5		0.13		1.00
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Cross Entropy

$$H(P, Q) = H(p) + D_{KL}(P \| Q)$$

Cross Entropy Loss with Softmax

- Cross entropy loss $L = \frac{1}{n}\sum_{i=1}^{n}(-\sum_{j=1}^{C} y_{i,j} log p_{i,j})$

- For single sample $L_1 = -\sum_{j=1}^{C} y_{1,j} log p_{1,j}$

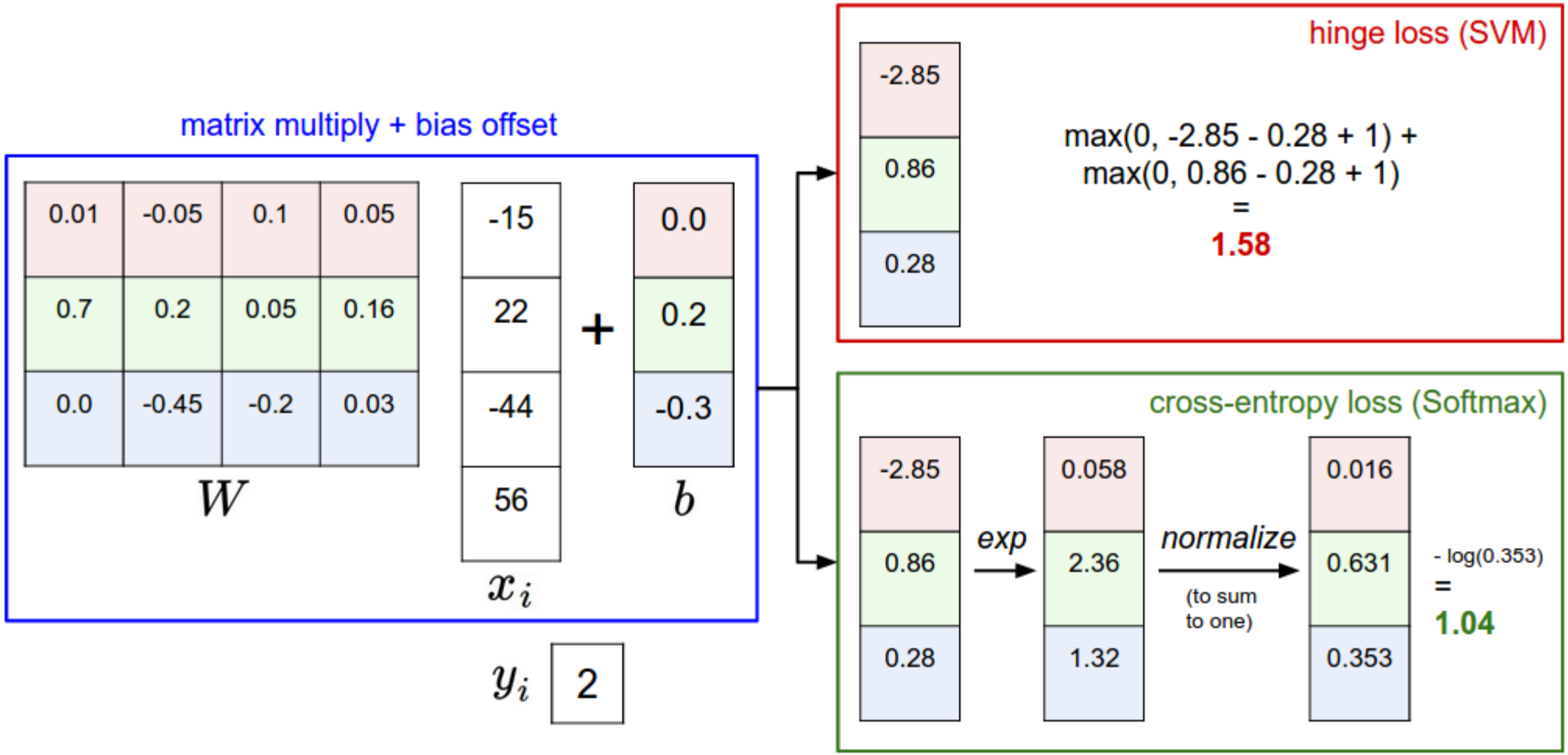
  – Assume $y_1 = (y_{1,1}, y_{1,2}) = (1, 0)$ $p_1 = (p_{1,1}, p_{1,2}) = (0.9, 0.1)$

$$L = -(1 * log p_{1,1} + 0 * log p_{1,2}) = -log p_{1,1} = -log 0.9$$

  – 只有當前樣本對應的類計算出來是有值的
  – 公式可以簡化為

$$L = -log p_{i,Y(i)}$$

SVM v.s. Softmax



Optimization a.k.a. How Do We Learn?

- How do we find the best W?

  - We have some dataset of (x,y)
  - We have a **score function:** $s = f(x;W) \overset{\text{e.g.}}{=} Wx$
  - We have a **loss function**:

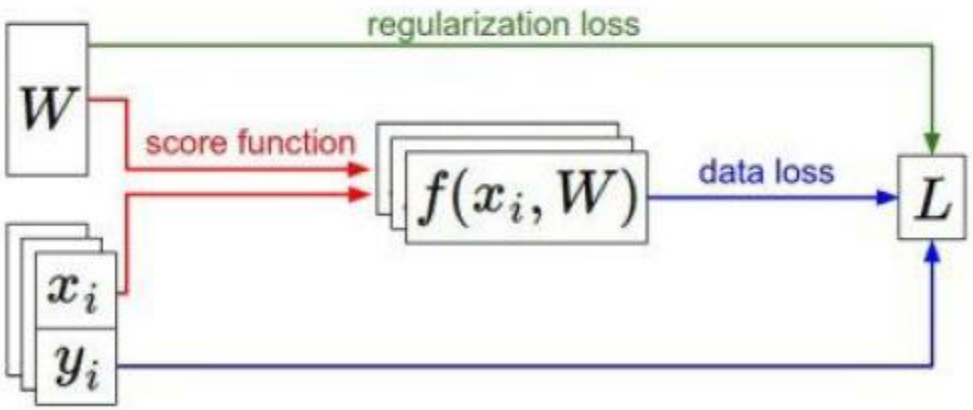
Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

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

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



Loss Function of Multiclass SVM (without regularization)

- Assume a simple dataset that contains three 1-dimensional points and three classes. The full SVM loss

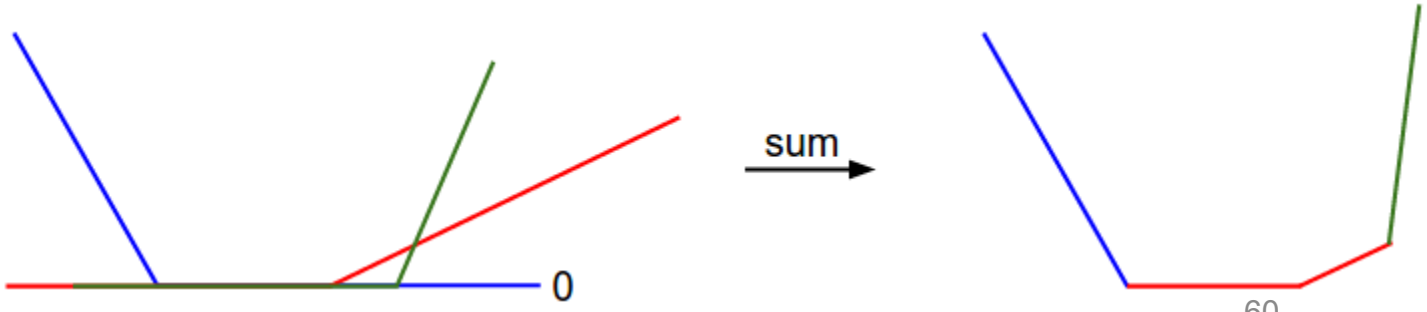
$$L_0 = max(0, w_1^T x_0 - w_0^T x_0 + 1) + max(0, w_2^T x_0 - w_0^T x_0 + 1)$$

$$L_1 = max(0, w_0^T x_1 - w_1^T x_1 + 1) + max(0, w_2^T x_1 - w_1^T x_1 + 1)$$

$$L_2 = max(0, w_0^T x_2 - w_2^T x_2 + 1) + max(0, w_1^T x_2 - w_2^T x_2 + 1)$$

$$L = (L_0 + L_1 + L_2)/3$$

- A convex function
  - Non differential at kinks
  - Easy to optimize
  - Non-convex in NN



Reach the Bottom As soon As Possible

- In the example of CIFAR-10, the hills are 30,730-dimensional, since the dimensions of W are 10 x 3073. At every point on the hill we achieve a particular loss (the height of the terrain)

Walking man image is CC0 1.0 public domain

Strategy #1: A first very bad idea solution: Random search

- start with random weights and iteratively refine them over time to get lower loss
  - Just keep the best
  - 15.5 accuracy
  - Better than random (10%)

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```

Strategy 2: Follow the Slope

- best direction to change weight
  - the direction of the steepest descend
    - negative gradient
  - Gradient of the loss function
- For 1-D

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

Gradient Descent (梯度下降法) $W(i+1) = W(i) - \eta \frac{\partial L}{\partial W}$

Numerical Gradients

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (first dim):

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

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

?,
?,…]

This is silly. The loss is just a function of W:

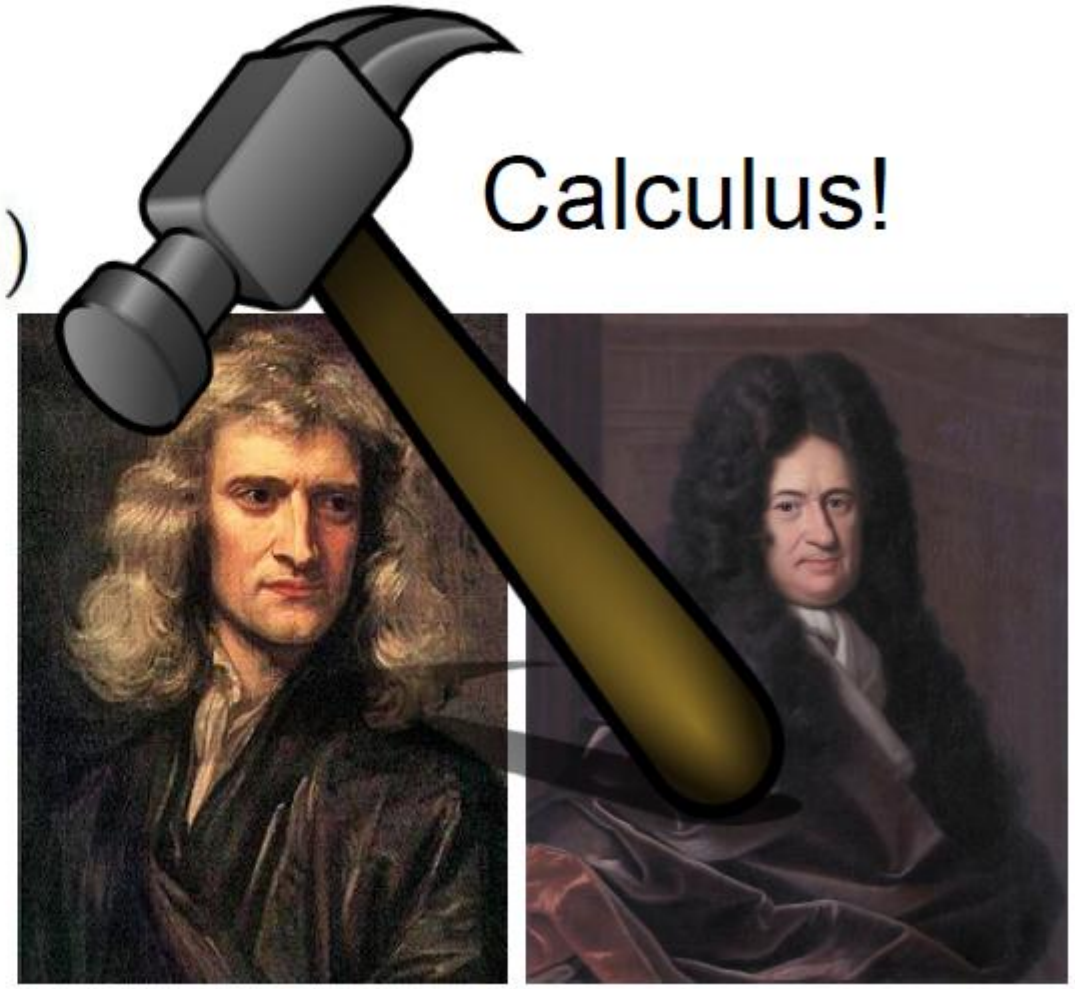
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

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

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Use calculus to compute an **analytic gradient**



This image is in the public domain

This image is in the public domain

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

dW = …
(some function
data and W)

$$L_i = \sum_{j \neq y_i} \left[max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1\left(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0\right)\right) x_i$$

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

Summary

- Numerical gradient:
  - approximate, slow, easy to write
- Analytic gradient:
  - exact, fast, error-prone

- In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a gradient check.

Gradient Descent/Stochastic Gradient Descent

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

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

- Gradient descent (full set)

```
# Vanilla Gradient Descent

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

- Full sum expensive when N is large!
- Approximate sum using a minibatch of examples 32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Summary

- Intuition of the loss function as a high-dimensional optimization landscape
- Optimizing the loss function with iterative refinement
- Gradient of a function gives the steepest ascent direction
  - simple but inefficient numerical gradient
  - Analytical gradient
  - Gradient check
- Gradient descent algorithm to update parameters
- Modern approach
  - **Automatic differentiation**

- ability to compute the gradient of a loss function with respect to its weights
  - (and have some intuitive understanding of it) is the most important skill needed to design, train and understand neural networks

DL/ML Framework

Image Recognition:

$f($ $) =$ "cat"

A set of function

Model

$f_1, f_2 \cdots$

$f_1($ $) =$ "cat"

$f_2($ $) =$ "money"

$f_1($ $) =$ "dog"

$f_2($ $) =$ "snake"

Framework

Image Recognition:

$f($ $) =$ "cat"

A set of function

Model

$f_1, f_2 \cdots$

Goodness of function f

Training Data

$f_1($ $) =$ "cat"

$f_2($ $) =$ "money"

Better!

$f_1($ $) =$ "dog"

$f_2($ $) =$ "snake"

function input:

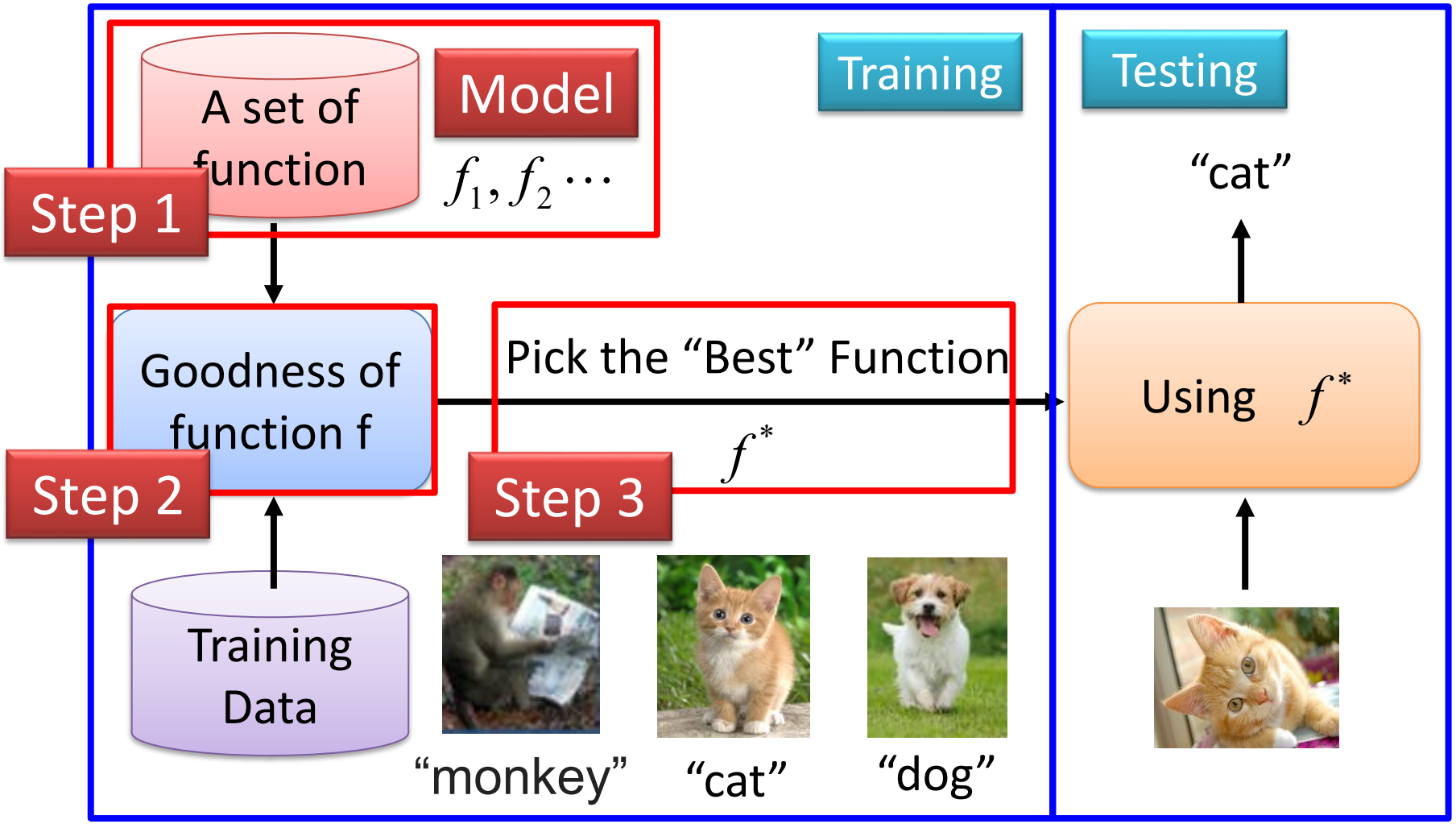
function output: "monkey" "cat" "dog"

Source: http://speech.ee.ntu.edu.tw/~tlkagk/courses_MLDS17.html

Framework

Image Recognition:

$f($ $) =$ "cat"



Source: http://speech.ee.ntu.edu.tw/~tlkagk/courses_MLDS17.html