Lecture 2:
Image Classification with Linear Classifiers

Administrative: Assignment 1

Will be out **TODAY 4/2, due 4/16 by 11:59 PM**

- K-Nearest Neighbor
- Linear classifiers: Softmax
- Two-layer neural network
- Image features
- Deep neural network and optimizers

Administrative: Course Project

Project proposal due 4/23 (Thursday) 11:59 pm. We will release proposal details on 4/16

Contact us on Ed, each project team will eventually have a TA assigned to them for future questions after the project proposal is due.

Use this Ed thread for finding project partners. We will also later post a Google Form to find project partners.

“Is X a valid project for 231n?” --- Ed private post / TA Office Hours

More info on the website

Administrative: Discussion Sections

This Friday 12:30 pm-1:20 pm, in person at NVIDIA Auditorium, remote on Zoom (recording will be made available)

Python / Numpy, Google Colab

Syllabus

Deep Learning Basics	Perceiving and Understanding the Visual World	Reconstructing and Interacting with the Visual World
Data-driven approaches	Transfer Learning	Style Transfer
Linear classification	Optimizers	Generative Models
K-Nearest Neighbor	Convolutions	Self-supervised Learning
Loss Functions	PyTorch	Image Generation
Optimization	RNNs / Attention / Transformers	Robotics and Embodied AI
Backpropagation	Normalization Layers	
Multi-layer Perceptrons	Architecture Design	
Neural Networks	Video Understanding	
Activation Functions	Vision and Language	Human-centered AI
Data Augmentation	3D Vision	Fairness & Ethics
	Object Detection and Segmentation	

Image Classification

A Core Task in Computer Vision

Today:

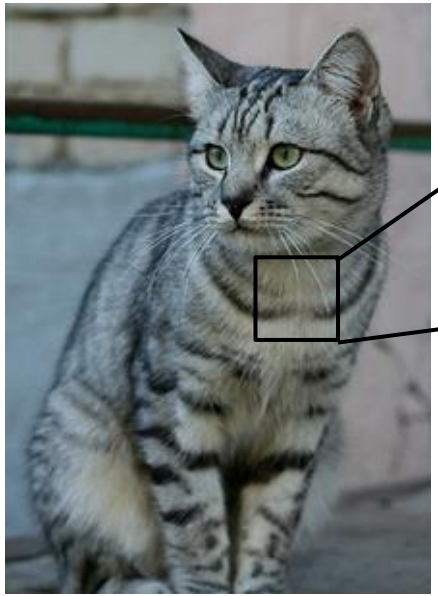
- The image classification task
- Two basic data-driven approaches to image classification
  - K-nearest neighbor and linear classifier

Image Classification: A core task in Computer Vision

(assume given a set of possible labels)
{dog, cat, truck, plane, ...}

cat

The Problem: Semantic Gap



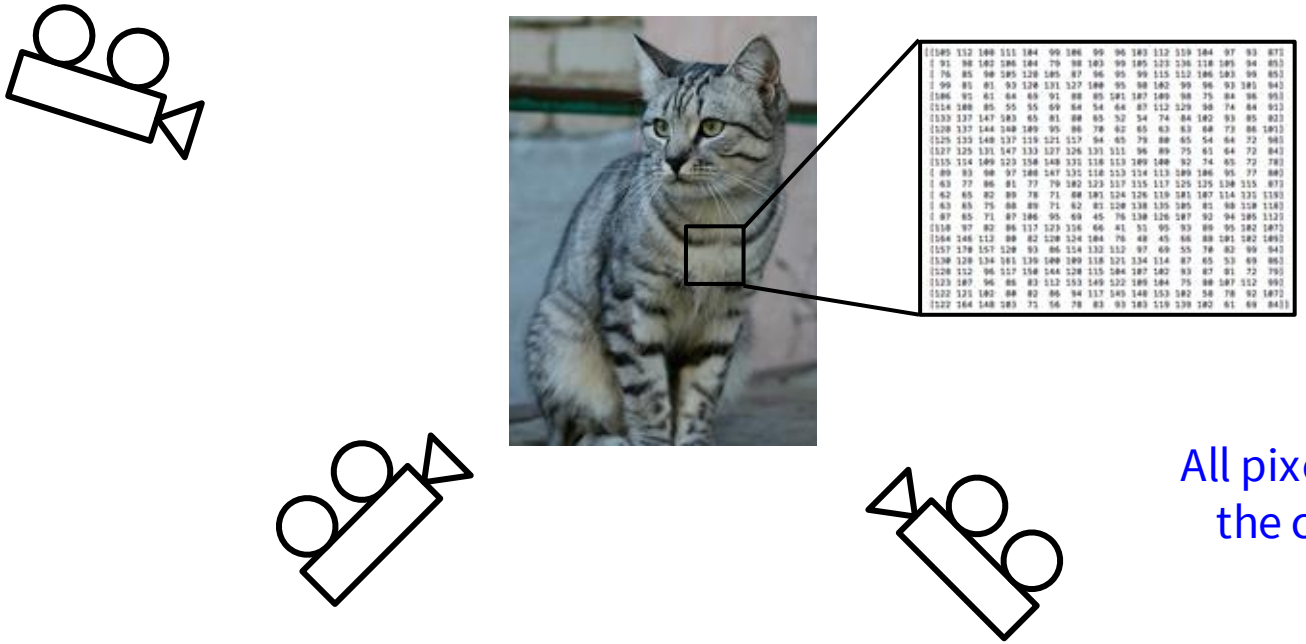

```
[[105 112 100 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 100  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 100 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  78  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is a tensor of integers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation



All pixels change when the camera moves!

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

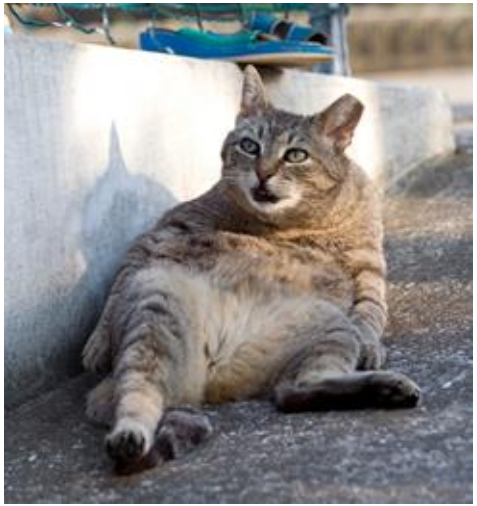
Challenges: Occlusion

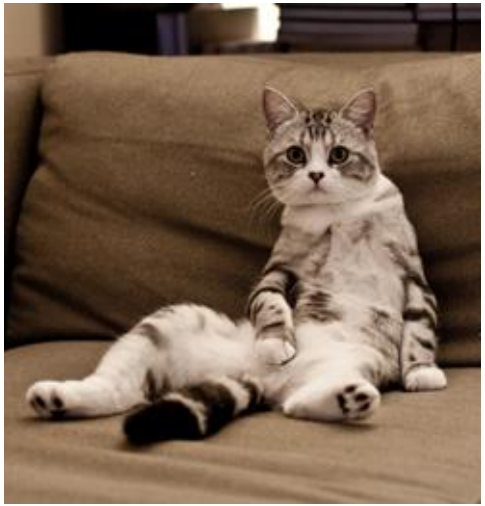
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

Challenges: Deformation





Challenges: Intraclass variation

Challenges: Context

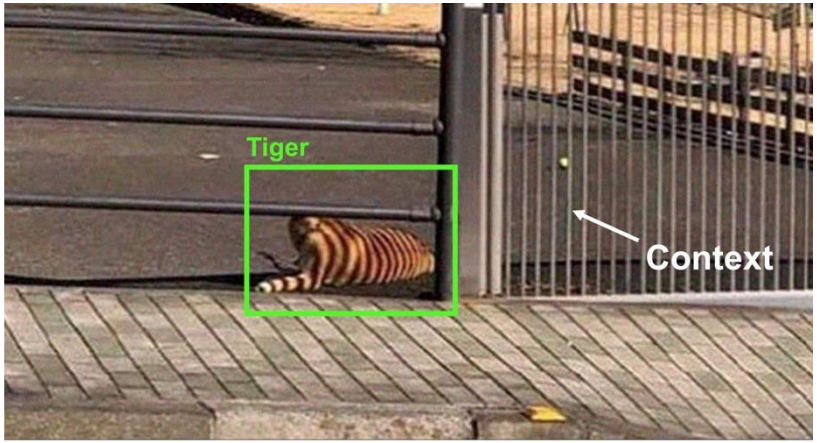


Image source: https://www.linkedin.com/posts/ralph-aboujaoude-diaz-40838313_technology-artificialintelligence-computervision-activity-6912446088364875776-h-Iq?utm_source=linkedin_share&utm_medium=member_desktop_web

Modern computer vision algorithms

An image classifier

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

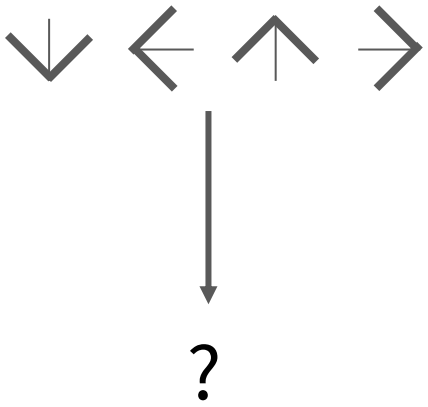
Unlike e.g. sorting a list of numbers,

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

Attempts have been made

Find edges

Find corners



?

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

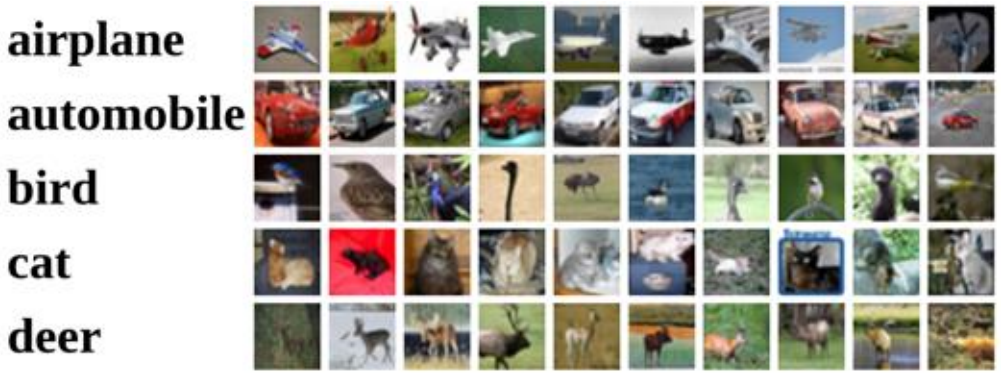
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning algorithms 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



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

Training data with labels

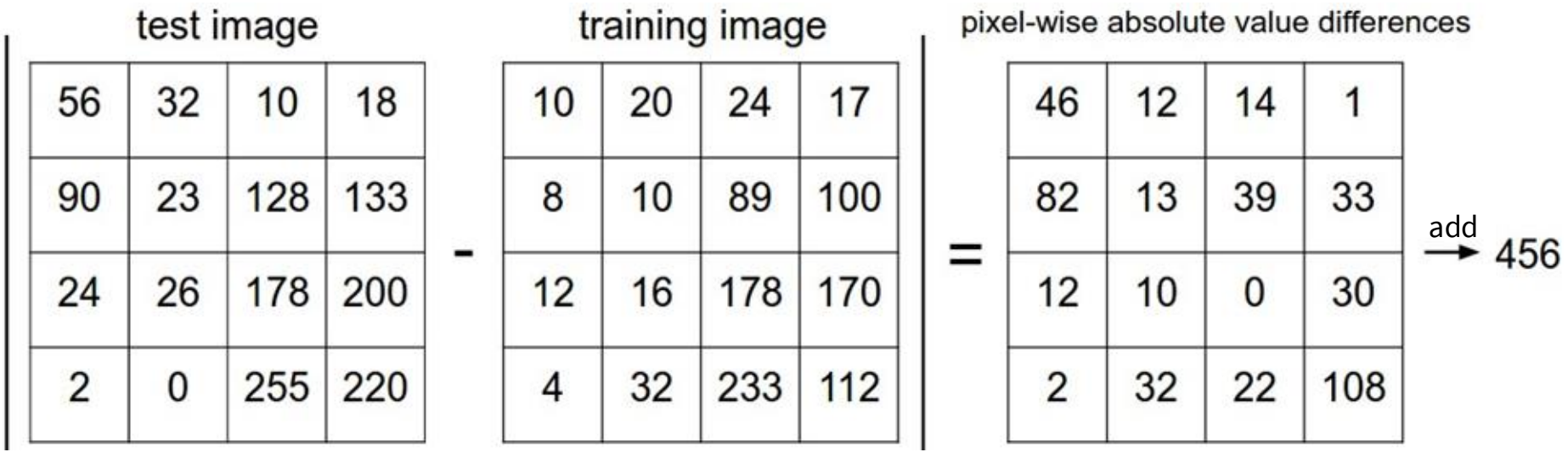
?

query data

Distance Metric $|\,,\,| \rightarrow \mathbb{R}$

Distance Metric to compare images

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



Nearest Neighbor classifier

```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
```

```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
```

Nearest Neighbor classifier

Memorize training data

Nearest Neighbor classifier

```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
```

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
```

Nearest Neighbor classifier

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

Ans: Train O(1),
predict O(N)

This is bad: we want classifiers that are fast at prediction; slow for training is ok

```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
```

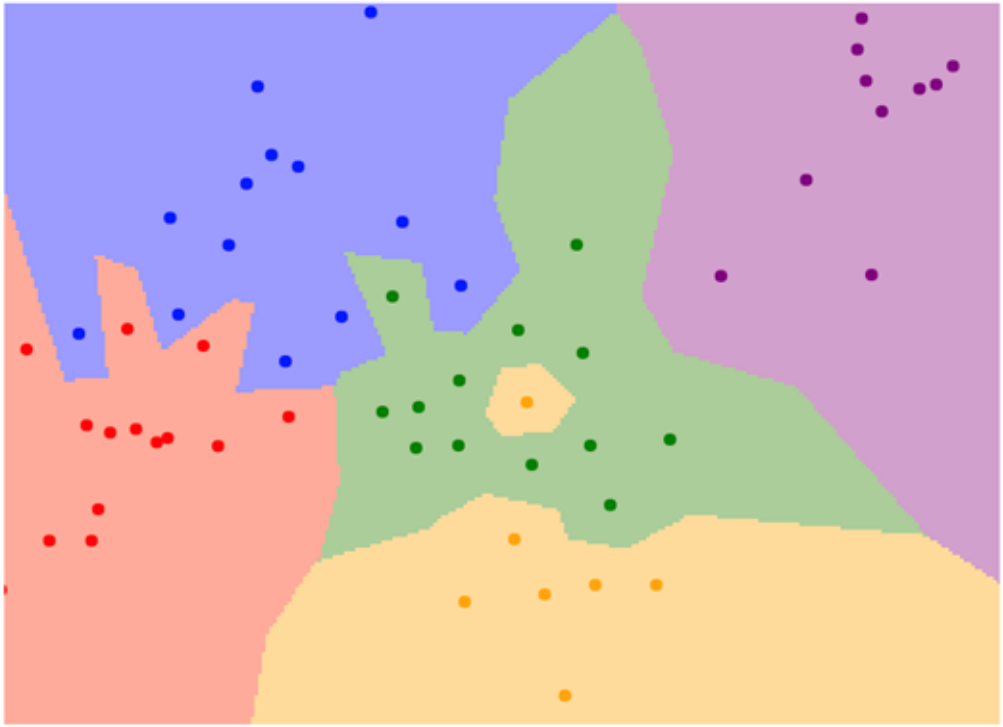
Nearest Neighbor classifier

Many methods exist for fast / approximate nearest neighbor (beyond the scope of 231N!)

A good implementation:
https://github.com/facebookresearch/faiss

Johnson et al, “Billion-scale similarity search with GPUs”, arXiv 2017

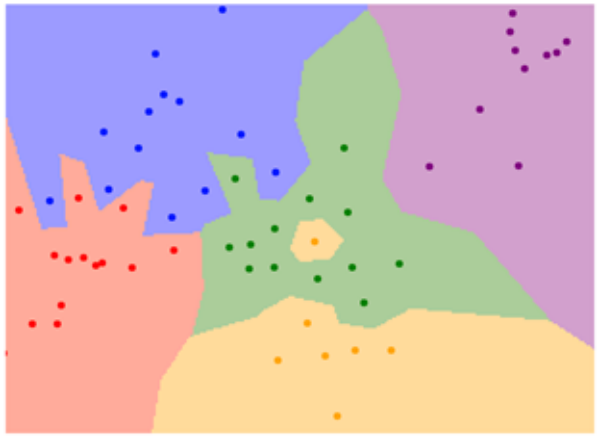
What does this look like?



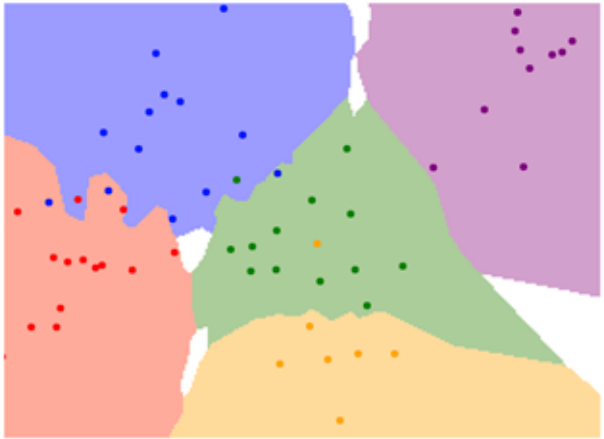
1-nearest neighbor

K-Nearest Neighbors

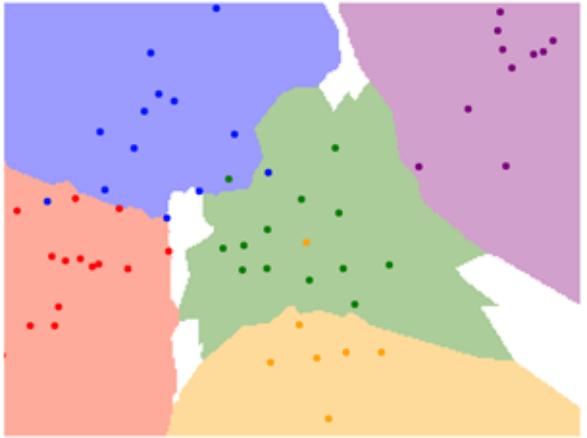
Instead of copying label from nearest neighbor,
take majority vote from K closest points



K = 1



K = 3

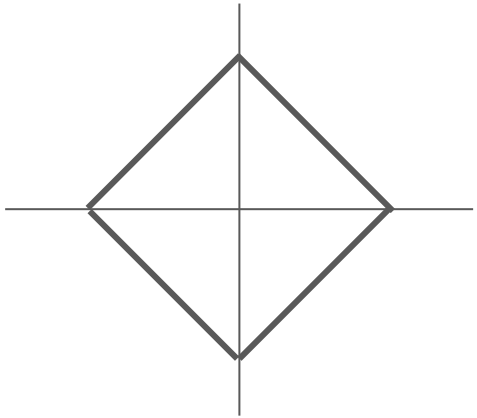


K = 5

K-Nearest Neighbors: Distance Metric

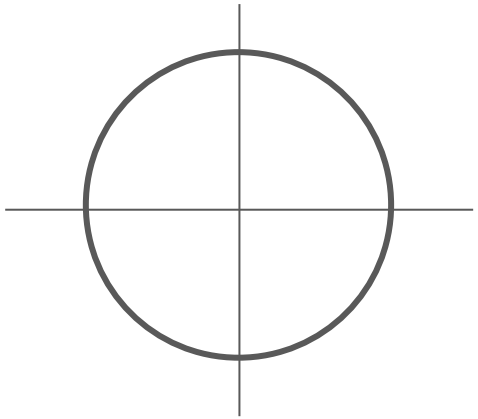
L1 (Manhattan) distance

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



L2 (Euclidean) distance

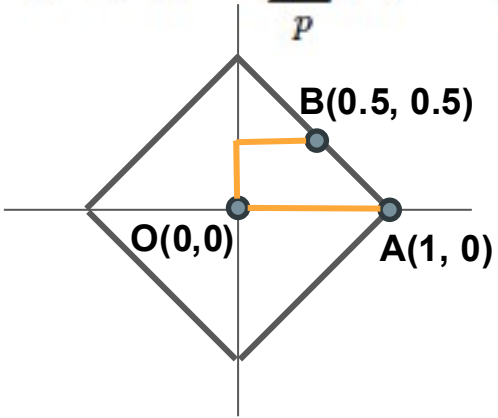
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



K-Nearest Neighbors: Distance Metric - Example

L1 Distance: Measures distance by moving along grid lines (like walking in a city with square blocks).

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



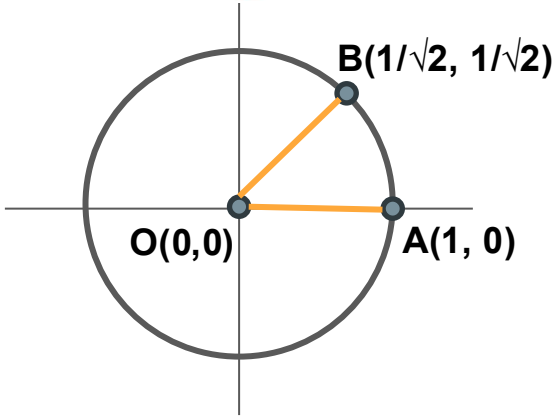
d_1(O,A) = |0-1| + |0-0| = 1

d_1(O,B) = |0-0.5| + |0-0.5| = 0.5 + 0.5 = 1

d_1(O,A) = d_1(O,B) = 1

L2 Distance: Measures the straight-line distance (as the crow flies).

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



d_2(O,A) = sqrt((0-1)^2 + (0-0)^2) = sqrt(1^2) = 1

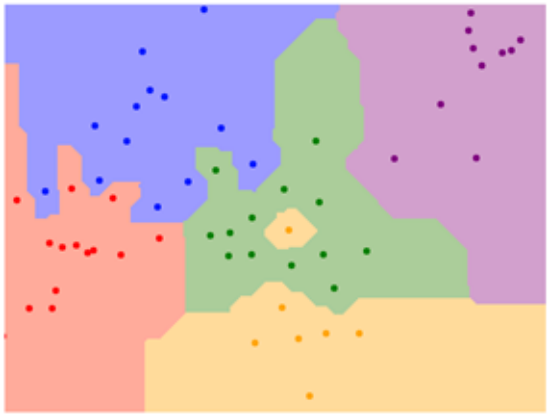
d_2(O,B) = sqrt((0-1/√2)^2 + (0-1/√2)^2) = sqrt(1/2+1/2) = sqrt(1) = 1

d_2(O,A) = d_2(O,A) = *1*

K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

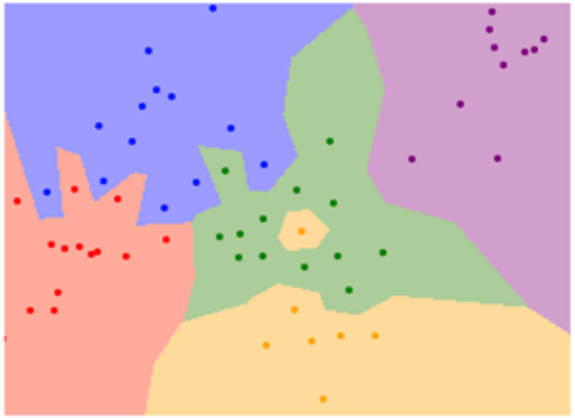
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

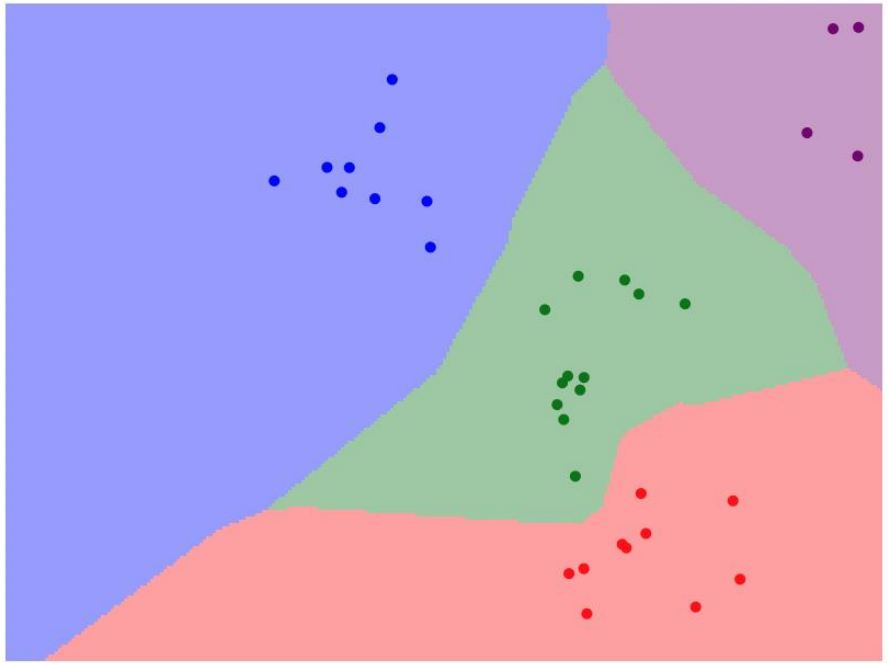
L2 (Euclidean) distance

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



K = 1

K-Nearest Neighbors: try it yourself!



http://vision.stanford.edu/teaching/cs231n-demos/knn/

Hyperparameters

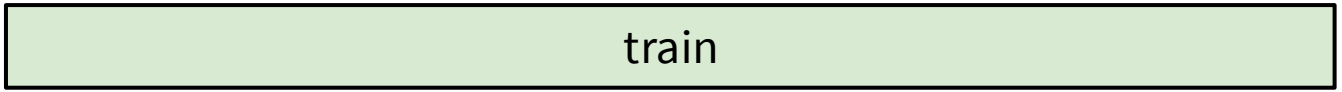
What is the best value of k to use?
What is the best distance to use?

These are hyperparameters: choices about the algorithms themselves.

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

Setting Hyperparameters

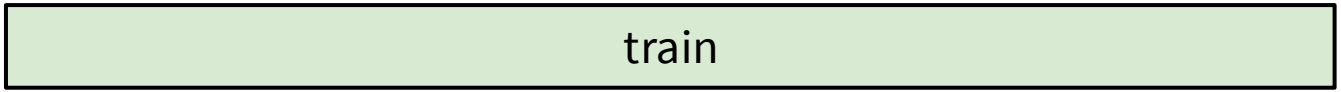
Idea #1: Choose hyperparameters that work best on the training data



Setting Hyperparameters

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

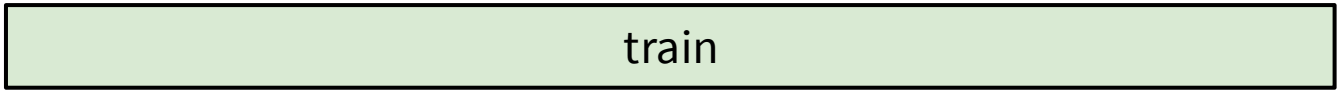
BAD: K = 1 always works perfectly on training data



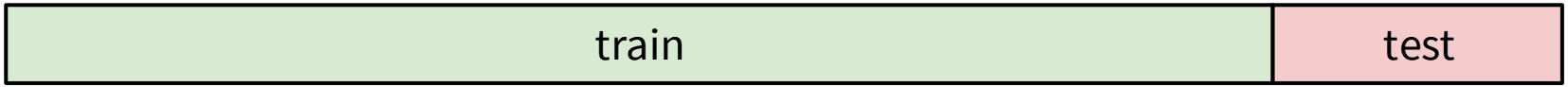
Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data



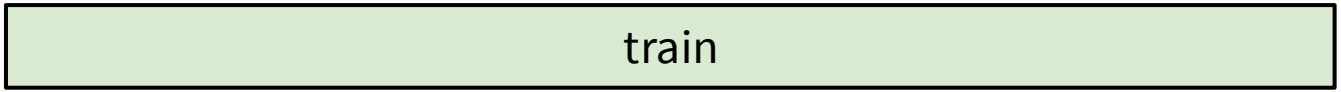
Idea #2: choose hyperparameters that work best on test data



Setting Hyperparameters

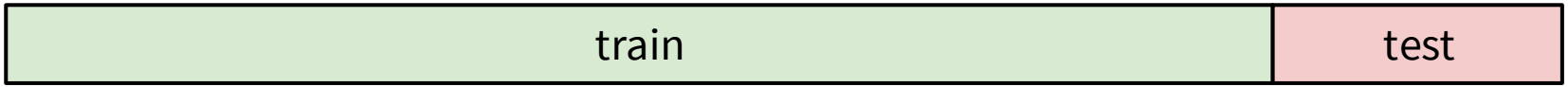
Idea #1: Choose hyperparameters that work best on the training data

BAD: K = 1 always works perfectly on training data



Idea #2: choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

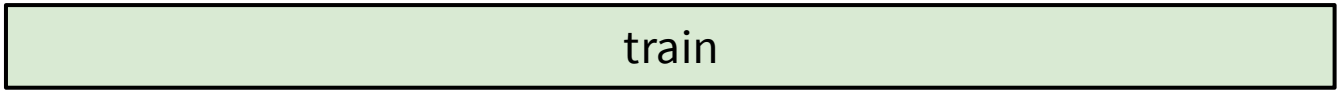


Never do this!

Setting Hyperparameters

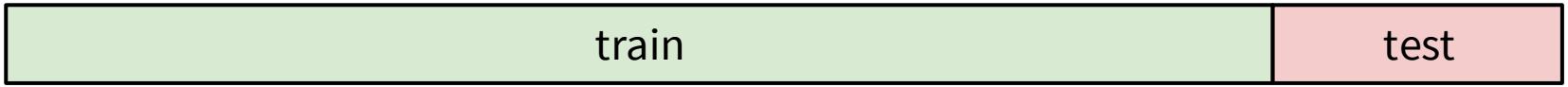
Idea #1: Choose hyperparameters that work best on the training data

BAD: K = 1 always works perfectly on training data



Idea #2: choose hyperparameters that work best on test data

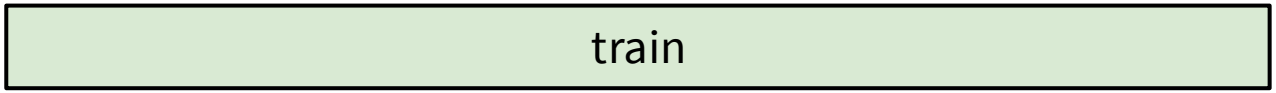
BAD: No idea how algorithm will perform on new data



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

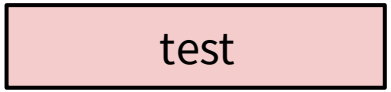
Better!

Setting Hyperparameters



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

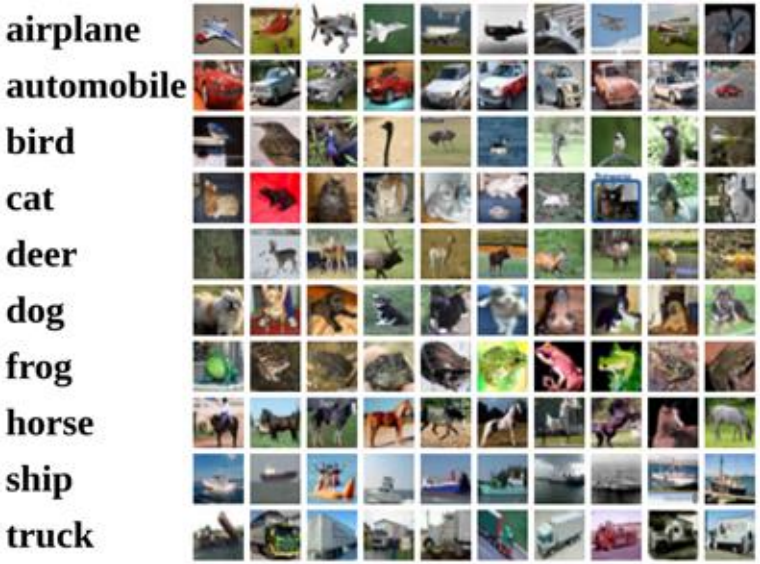
fold 1	fold 2	fold 3	fold 4	fold 5
fold 1	fold 2	fold 3	fold 4	fold 5
fold 1	fold 2	fold 3	fold 4	fold 5
fold 1	fold 2	fold 3	fold 4	fold 5
fold 1	fold 2	fold 3	fold 4	fold 5



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

Example Dataset: CIFAR10

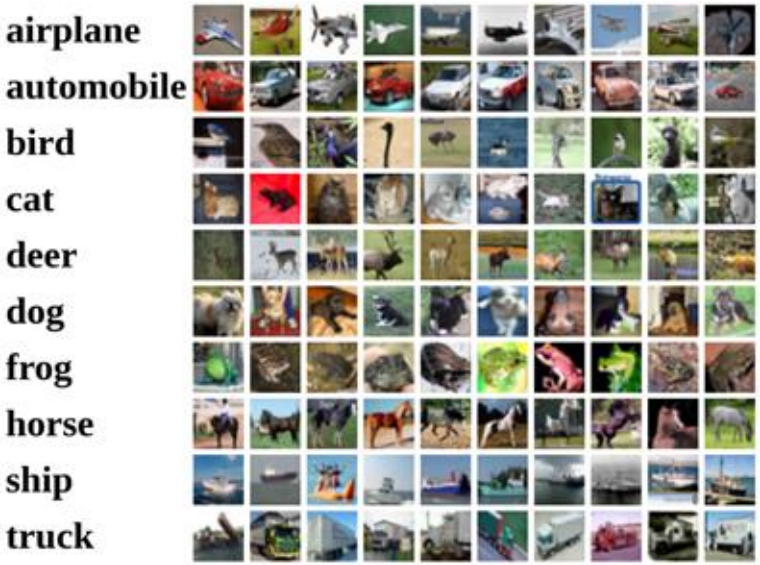
10 classes
50,000 training images
10,000 testing images



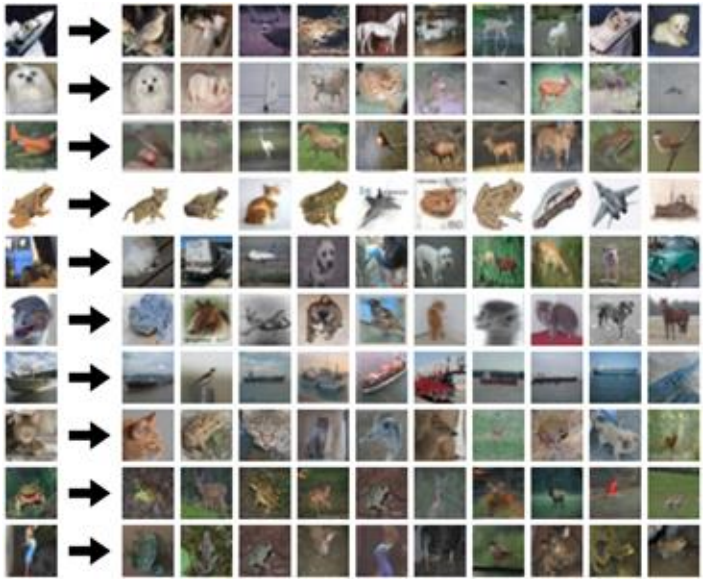
Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Example Dataset: CIFAR10

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

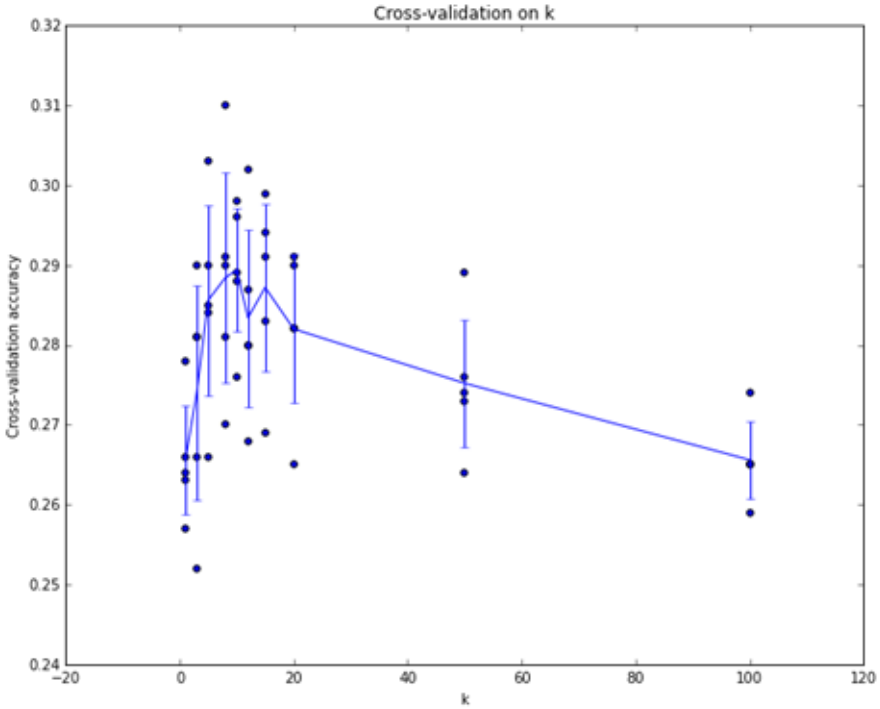


Test images and nearest neighbors



Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Setting Hyperparameters



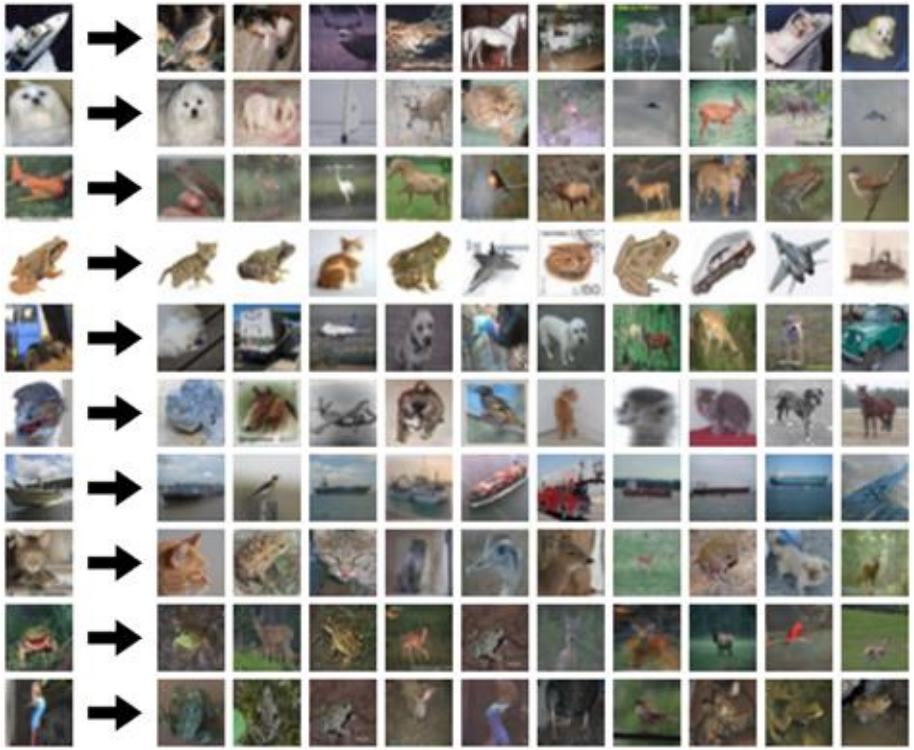
Example of
5-fold cross-validation
for the value of k.

Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that k ~= 7 works best
for this data)

What does this look like?



What does this look like?

k-Nearest Neighbor with pixel distance is never used.

- Distance metrics on pixels are not informative

Original

Occluded

Shifted (1 pixel)

Tinted

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

K-Nearest Neighbors: Summary

In image classification we start with a training set of images and labels, and must predict labels on the test set

The K-Nearest Neighbors classifier predicts labels based on the **K nearest training examples**

Distance metric and K are hyperparameters

Choose hyperparameters using the validation set

Only run on the test set once at the very end!

Linear Classifier

Parametric Approach

Image

Array of 32x32x3 numbers
(3072 numbers total)

f(x,W)

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

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

Image

Array of 32x32x3 numbers
(3072 numbers total)

f(x,W)

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

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

3072x1

10x1

10x3072

Image

Array of 32x32x3 numbers
(3072 numbers total)

f(x,W)

10 numbers giving class scores

W

parameters or weights

Parametric Approach: Linear Classifier

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

3072x1 (x)

10x1 (b)

10x1 (f(x,W))

10x3072 (W)

Image

Array of 32x32x3 numbers
(3072 numbers total)

f(x,W)

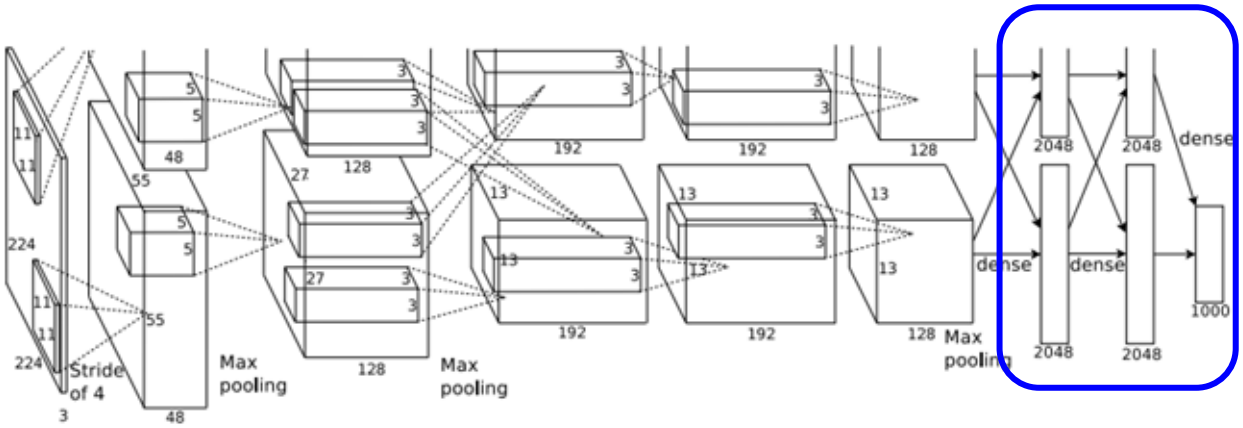
10 numbers giving class scores

W

parameters or weights

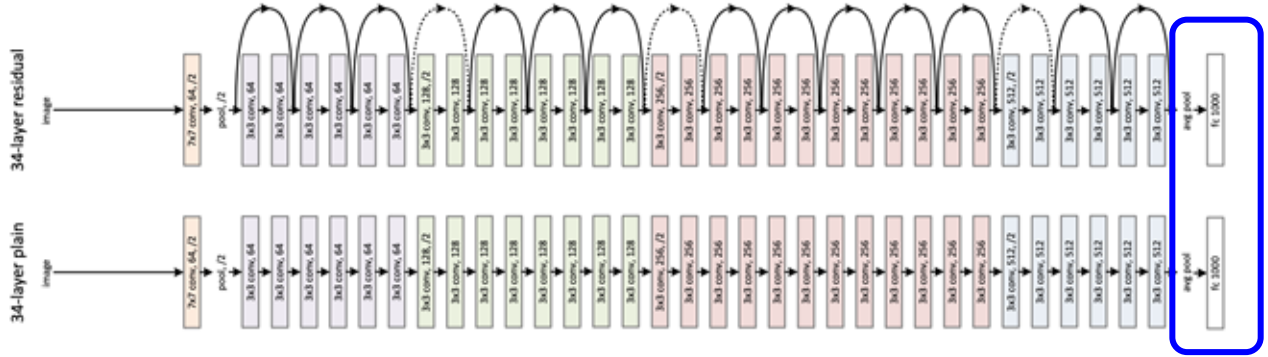
Neural Network

Linear classifiers



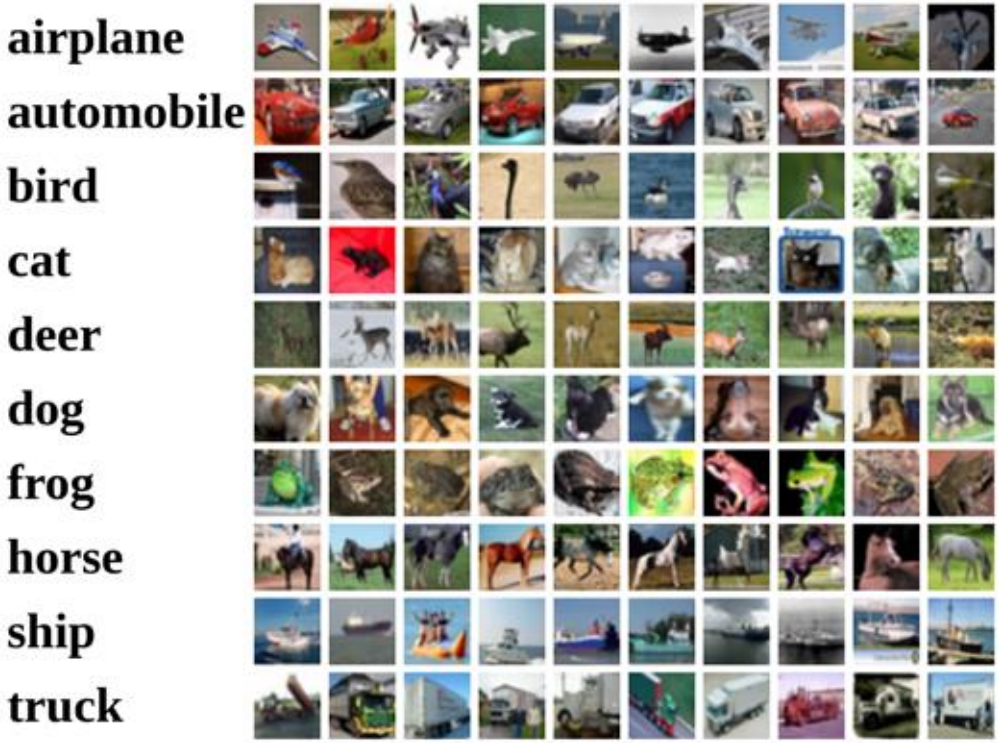
[Krizhevsky et al. 2012]

Linear layers



[He et al. 2015]

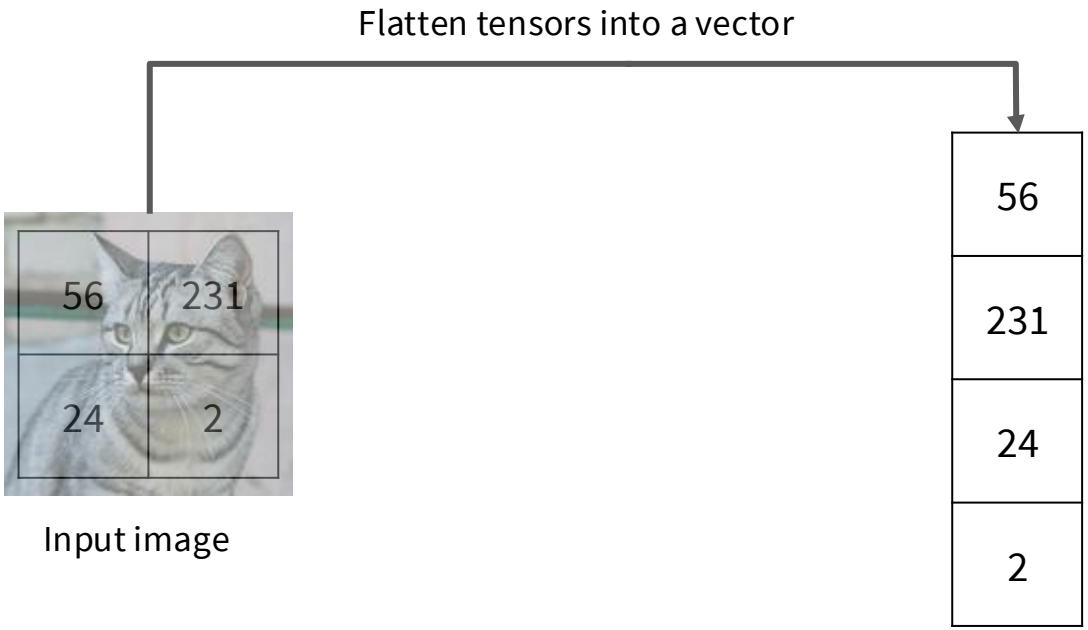
Recall CIFAR10



50,000 training images
each image is 32x32x3

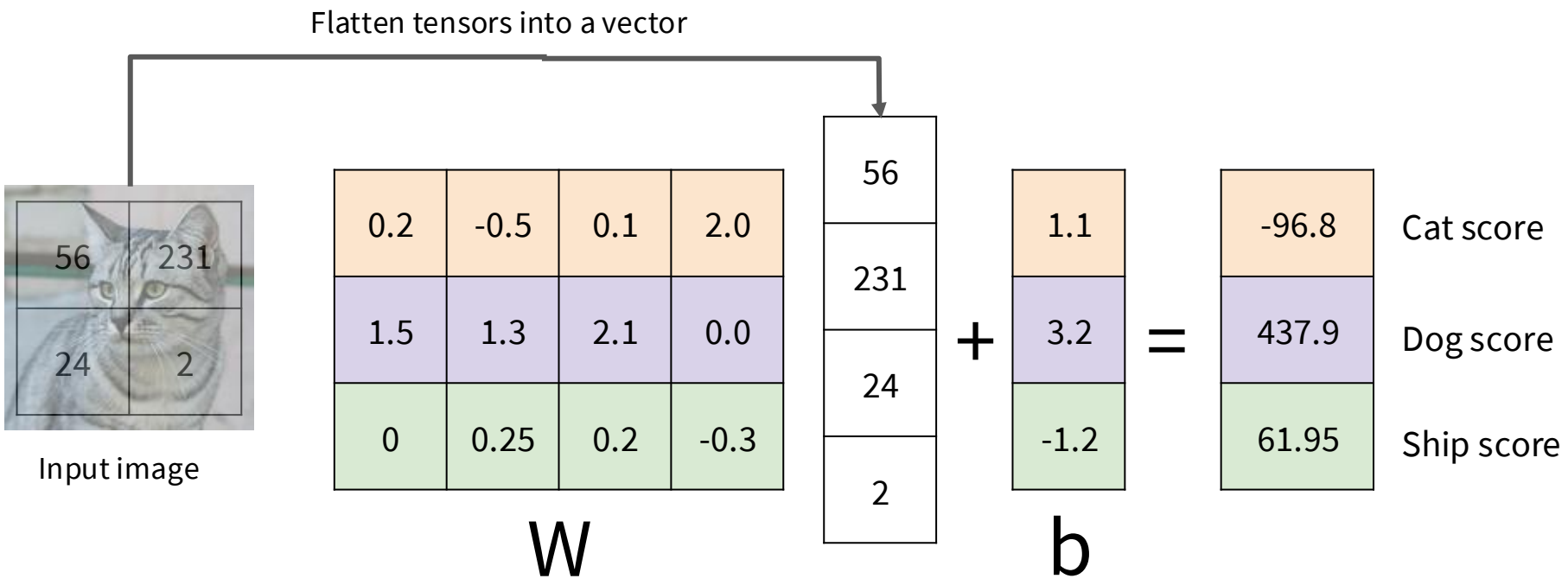
10,000 test images.

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

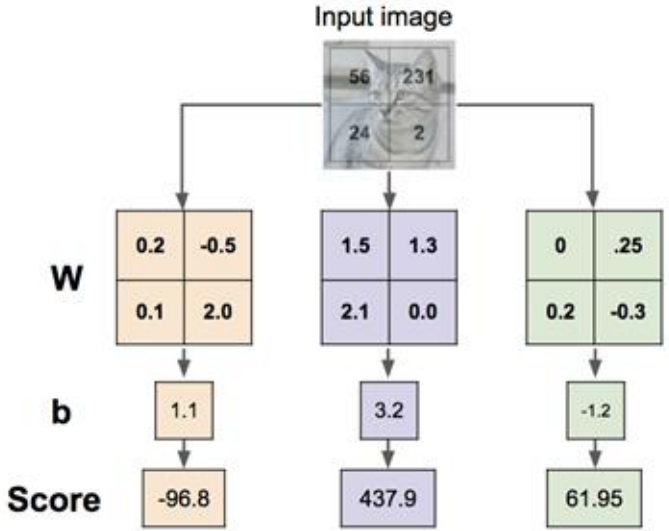


Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

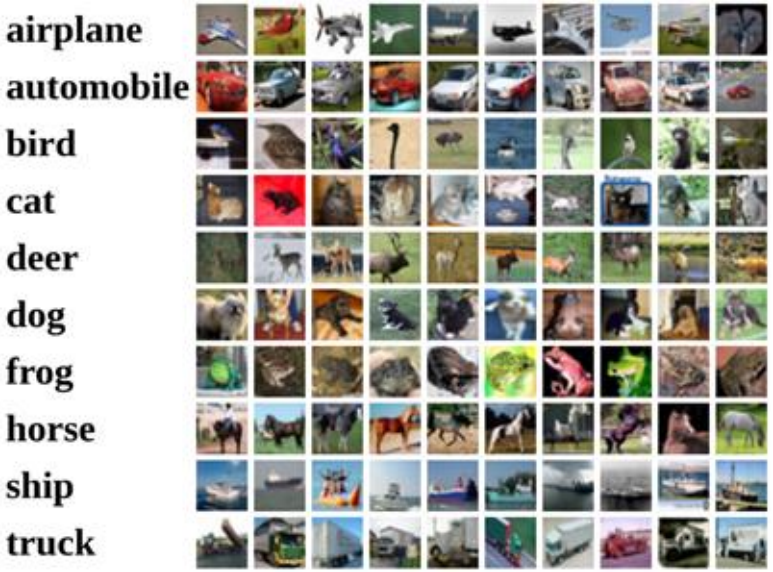
Algebraic Viewpoint

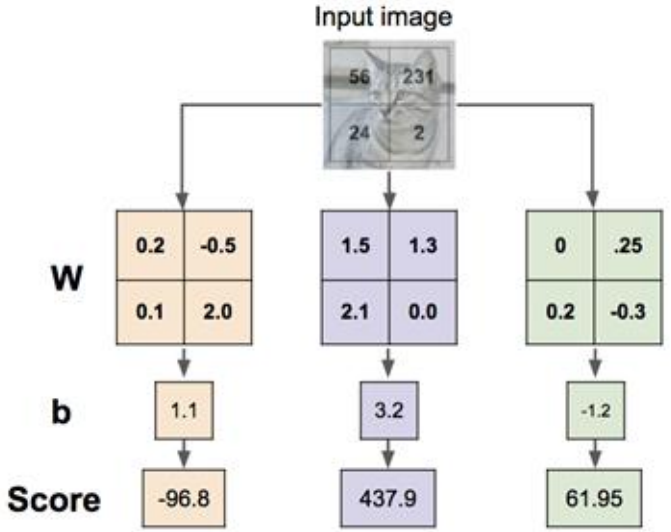


Interpreting a Linear Classifier

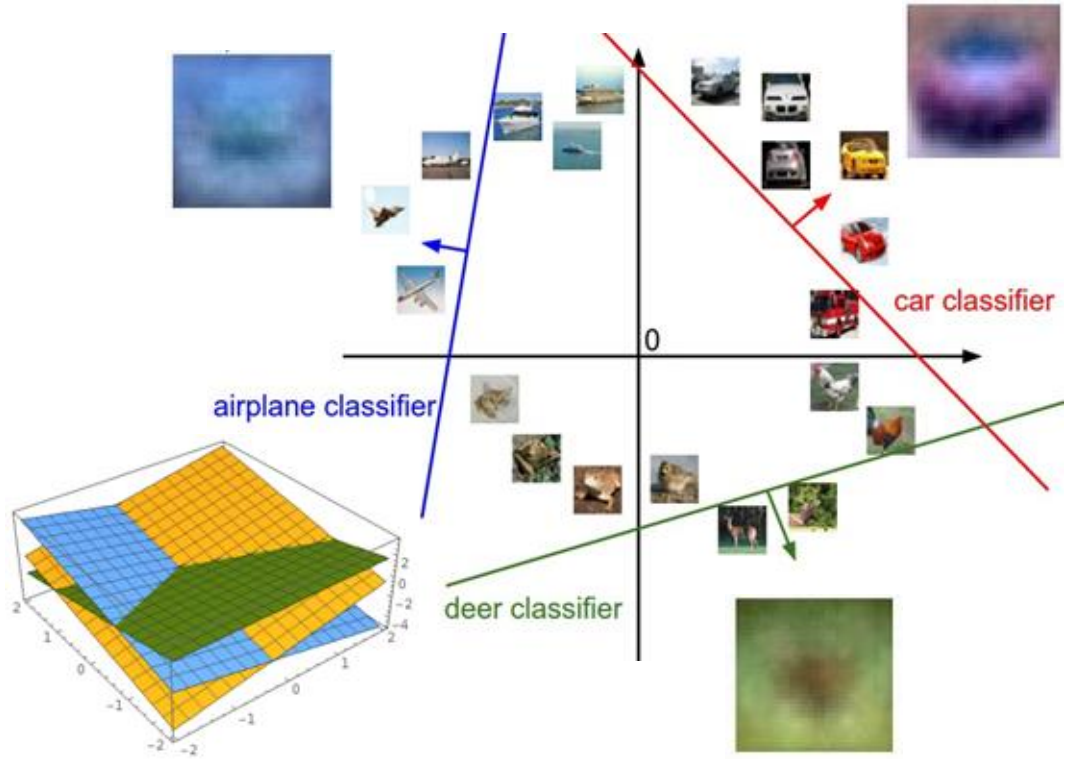


Interpreting a Linear Classifier: Visual Viewpoint





Interpreting a Linear Classifier: Geometric Viewpoint



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

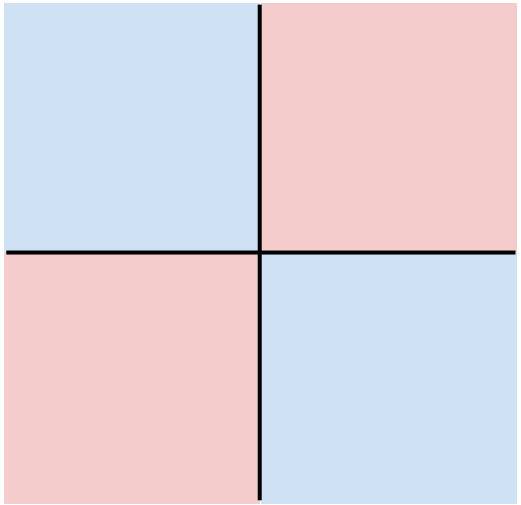
Array of 32x32x3 numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Hard cases for a linear classifier

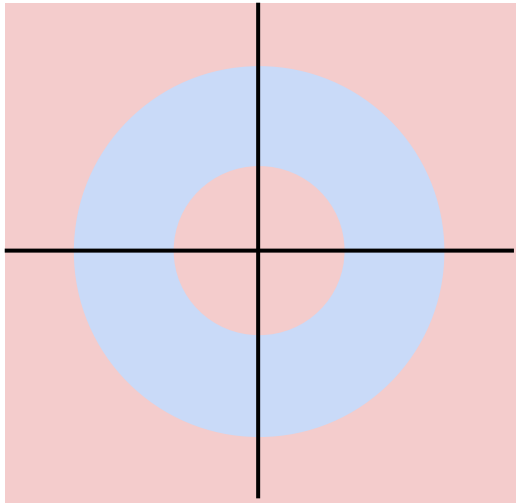
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



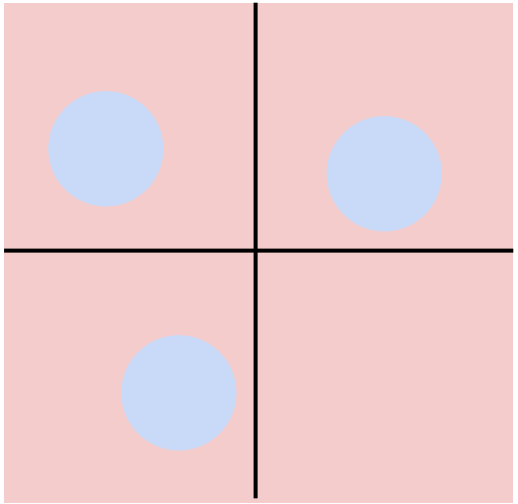
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



Linear Classifier – Choose a good W

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

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)

Linear Classifier – Choose a good W

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

A loss function tells how good our current classifier is



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

Linear Classifier – Choose a good W

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



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

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

Linear Classifier – Choose a good W

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

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

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 average of loss over examples:

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

Softmax classifier

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

cat	3.2
car	5.1
frog	-1.7

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

cat	3.2
car	5.1
frog	-1.7

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

cat	3.2	exp →	24.5
car	5.1		164.0
frog	-1.7		0.18

unnormalized probabilities

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

		exp		normalize	
cat	3.2	→	24.5	→	0.13
car	5.1	→	164.0	→	0.87
frog	-1.7	→	0.18	→	0.00

unnormalized probabilities

probabilities

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

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

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	
cat	3.2		24.5		0.13	→ L_i = -log(0.13) = 2.04
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

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

→ L_i = -log(0.13) = 2.04

Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data
(See CS 229 for details)

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

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

Maximize probability of correct class

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

Putting it all together:

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

cat	3.2
car	5.1
frog	-1.7

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

Maximize probability of correct class

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

Putting it all together:

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

cat	3.2
car	5.1
frog	-1.7

Q1: What is the min/max possible softmax loss L_i?

Q2: At initialization all s_j will be approximately equal; what is the softmax loss L_i, assuming C classes?

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

Maximize probability of correct class

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

Putting it all together:

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

cat	3.2
car	5.1
frog	-1.7

Q2: At initialization all s will be approximately equal; what is the loss?
A: -log(1/C) = log(C),
If C = 10, then L_i = log(10) ≈ 2.3

Coming up:

- Regularization
- Optimization

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