

# Lecture 2: Image Classification with Linear Classifiers

# Administrative: Assignment 1

Out tomorrow, Due 4/15 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax
- Two-layer neural network
- Image features

# Administrative: Course Project

Project proposal due 4/18 (Monday) 11:59pm

Find your teammates on Ed (the pinned “Search for Teammates” post)

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

More info on the website

# Administrative: Discussion Sections

This Friday 1:30pm-2:30 pm (recording will be made available)

Python / Numpy, Google Colab

Presenter: Manasi Sharma (TA)

# Syllabus

## Deep Learning Basics

### **Data-driven approaches** **Linear classification & kNN**

Loss functions  
Optimization  
Backpropagation  
Multi-layer perceptrons  
Neural Networks

## Convolutional Neural Networks

Convolutions  
PyTorch / TensorFlow  
Activation functions  
Batch normalization  
Transfer learning  
Data augmentation  
Momentum / RMSProp / Adam  
Architecture design

## Computer Vision Applications

RNNs / Attention / Transformers  
Image captioning  
Object detection and segmentation  
Style transfer  
Video understanding  
Generative models  
Self-supervised learning  
3D vision  
Human-centered AI  
Fairness & ethics



# 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



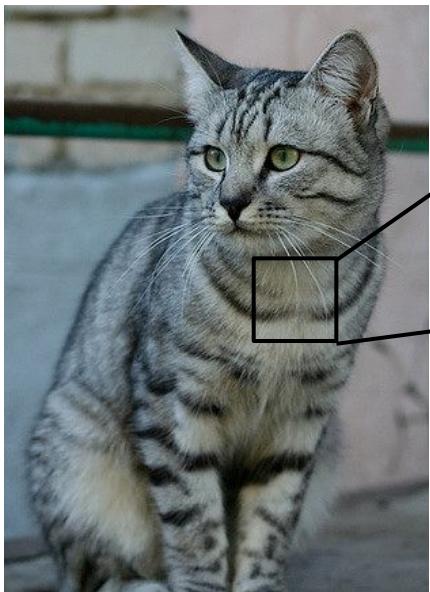
(assume given a set of possible labels)  
{dog, cat, truck, plane, ...}



cat

This image by [Nikita](#) is  
licensed under [CC-BY 2.0](#)

# The Problem: Semantic Gap



[ 105 112 108 111 104 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 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]
[114 108 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 97 108 147 131 118 113 114 113 109 106 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 81 128 138 135 105 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 88 82 120 124 104 76 48 45 66 88 101 102 109]
[157 170 157 128 93 86 114 132 112 97 69 55 70 82 99 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
[128 112 96 117 150 144 128 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
[122 121 102 88 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]]

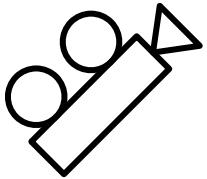
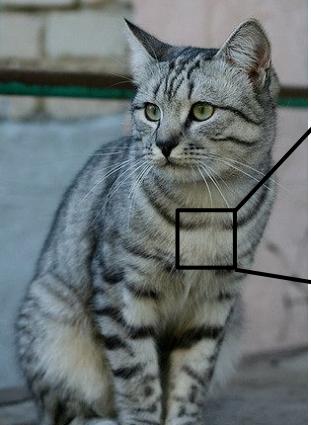
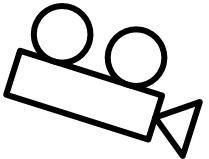
What the computer sees

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

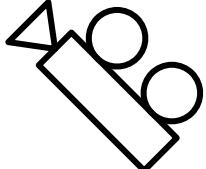
e.g. 800 x 600 x 3  
(3 channels RGB)

This image by Nikita is  
licensed under CC-BY 2.0

# Challenges: Viewpoint variation



```
[1185 112 188 111 184 99 186 99 96 183 112 119 184 97 93 87]  
[ 91 98 182 106 184 79 98 183 99 185 123 136 118 185 94 85]  
[ 76 85 98 185 128 105 87 96 95 99 115 112 106 183 99 85]  
[ 99 80 81 100 128 103 127 98 98 101 100 109 98 98 94 84]  
[104 91 86 84 60 91 68 85 101 100 109 98 75 84 96 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]  
[133 137 147 183 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 148 105 95 86 78 62 65 63 63 68 73 86 101]  
[102 125 131 147 133 127 116 131 111 98 89 75 61 64 72 80]  
[127 125 131 147 133 127 116 131 111 98 89 75 61 64 72 84]  
[115 111 189 123 150 148 131 118 113 109 108 92 74 65 72 78]  
[ 89 93 98 97 108 147 131 118 113 114 113 108 106 95 77 80]  
[ 63 77 86 81 77 79 182 123 137 115 111 125 125 130 115 87]  
[ 62 85 88 89 73 62 81 128 138 135 105 81 98 118 118]  
[ 63 65 75 88 89 73 62 81 128 138 135 105 81 98 118 118]  
[ 87 65 71 87 100 95 69 45 76 126 126 107 92 94 105 112]  
[118 97 82 86 117 123 116 66 41 51 95 93 89 89 95 102 107]  
[164 147 112 88 100 128 126 184 78 48 66 70 101 102 108]  
[157 98 100 118 103 92 86 104 100 109 108 95 70 89 91 94]  
[138 128 134 161 139 180 109 118 121 134 114 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 83 112 153 149 122 189 184 75 88 107 112 99]  
[122 121 102 88 82 86 94 117 145 148 153 105 58 78 92 107]  
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]
```



All pixels change when  
the camera moves!

This image by [Nikita](#) is  
licensed under [CC-BY 2.0](#)

# Challenges: Illumination



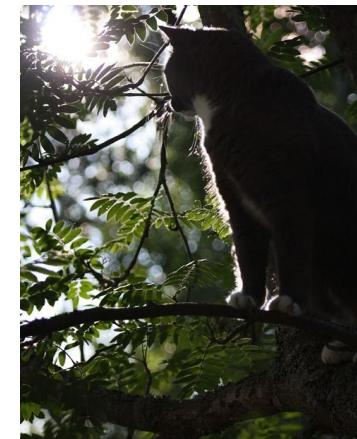
[This image](#) is CC0 1.0 public domain



[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

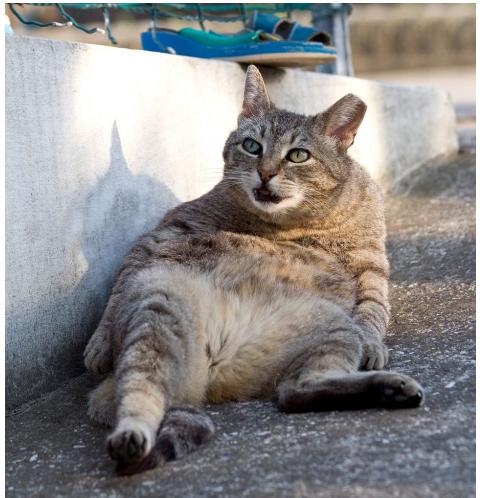


[This image](#) is [CC0 1.0](#) public domain



[This image](#) by [jonsson](#) is licensed  
under [CC-BY 2.0](#)

# Challenges: Deformation



[This image](#) by [Umberto Salvagnin](#)  
is licensed under [CC-BY 2.0](#)



[This image](#) by [Umberto Salvagnin](#)  
is licensed under [CC-BY 2.0](#)



[This image](#) by [sare bear](#) is  
licensed under [CC-BY 2.0](#)



[This image](#) by [Tom Thai](#) is  
licensed under [CC-BY 2.0](#)

# Challenges: Intraclass variation



[This image](#) is [CC0 1.0](#) public domain

# Challenges: Context

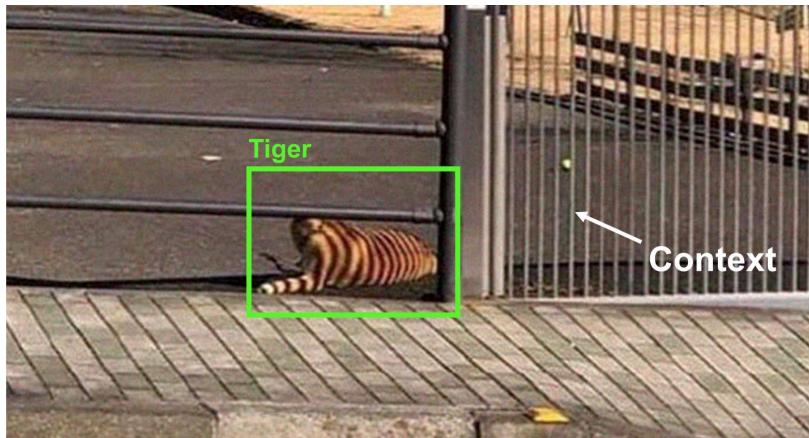
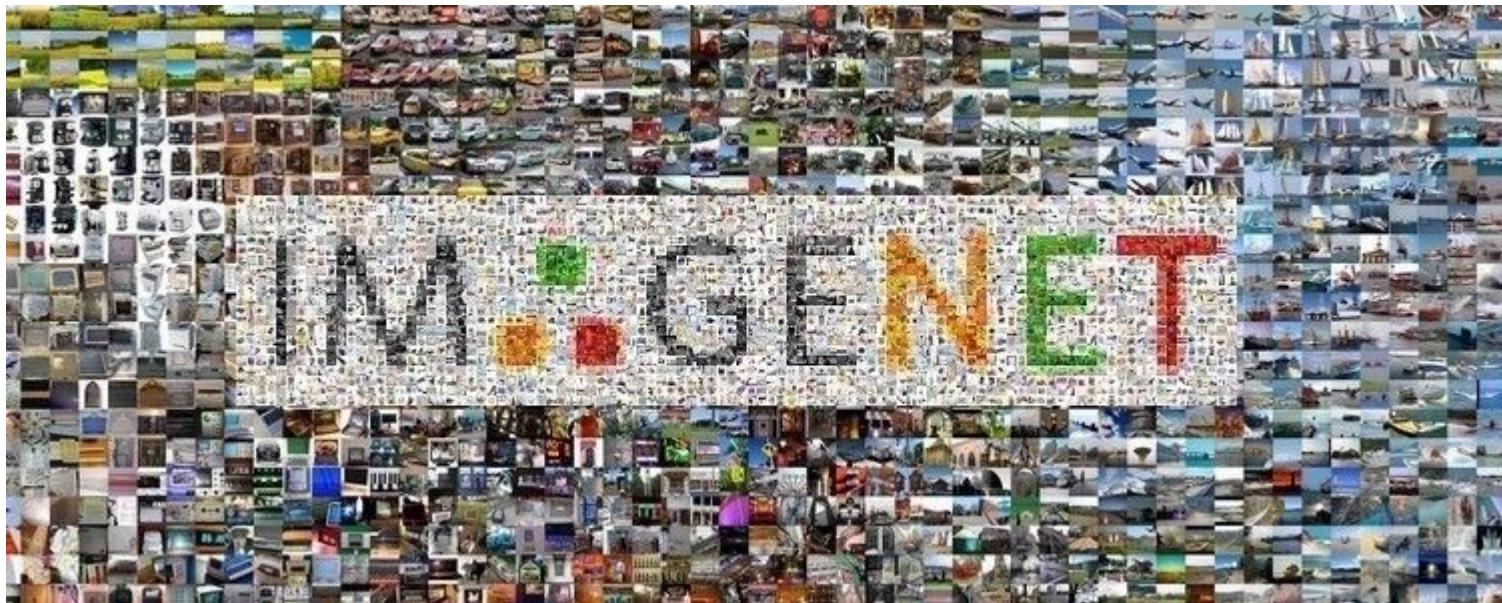


Image source:

[https://www.linkedin.com/posts/ralph-aboujaoude-diaz-40838313\\_technology-artificialintelligence-computervision-activity-6912446088364875776-h-lq/?utm\\_source=linkedin\\_share&utm\\_medium=member\\_desktop\\_web](https://www.linkedin.com/posts/ralph-aboujaoude-diaz-40838313_technology-artificialintelligence-computervision-activity-6912446088364875776-h-lq/?utm_source=linkedin_share&utm_medium=member_desktop_web)

# Modern computer vision algorithms



[This image](#) is [CC0 1.0](#) public domain

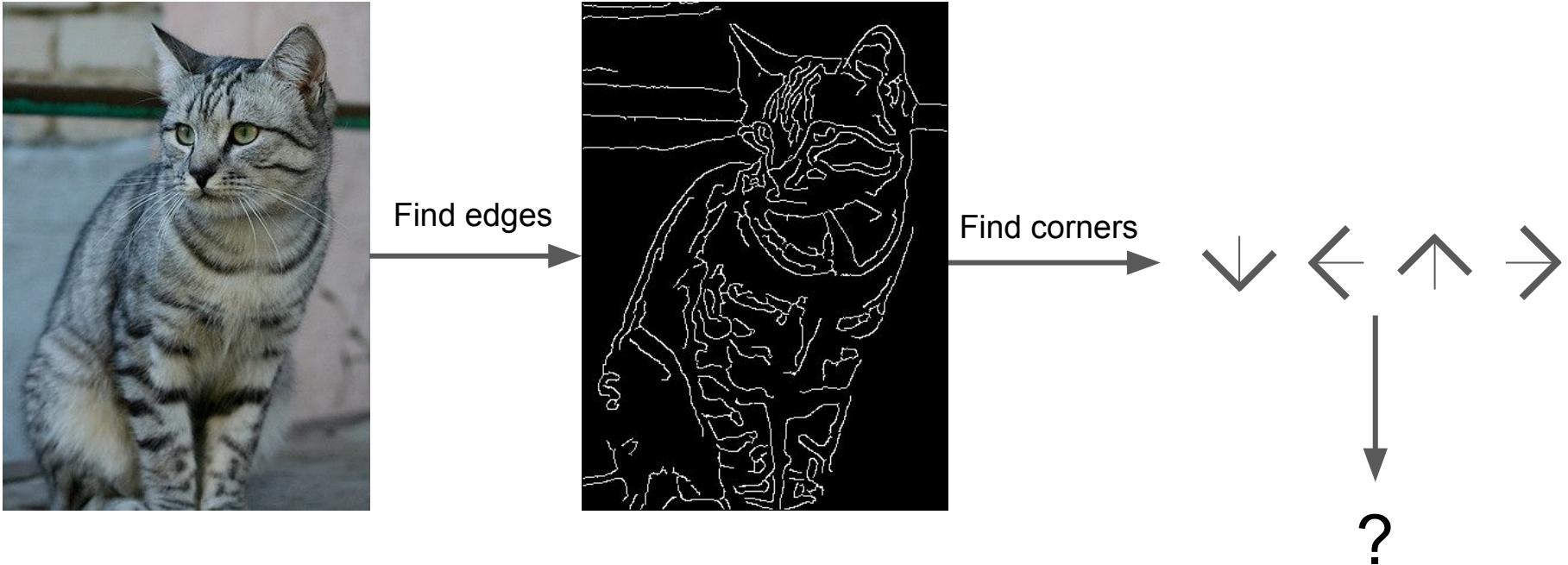
# An image classifier

```
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 have been made



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

Example training set

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

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

**airplane**



**automobile**



**bird**



**cat**



**deer**



# Nearest Neighbor Classifier

# First classifier: Nearest Neighbor

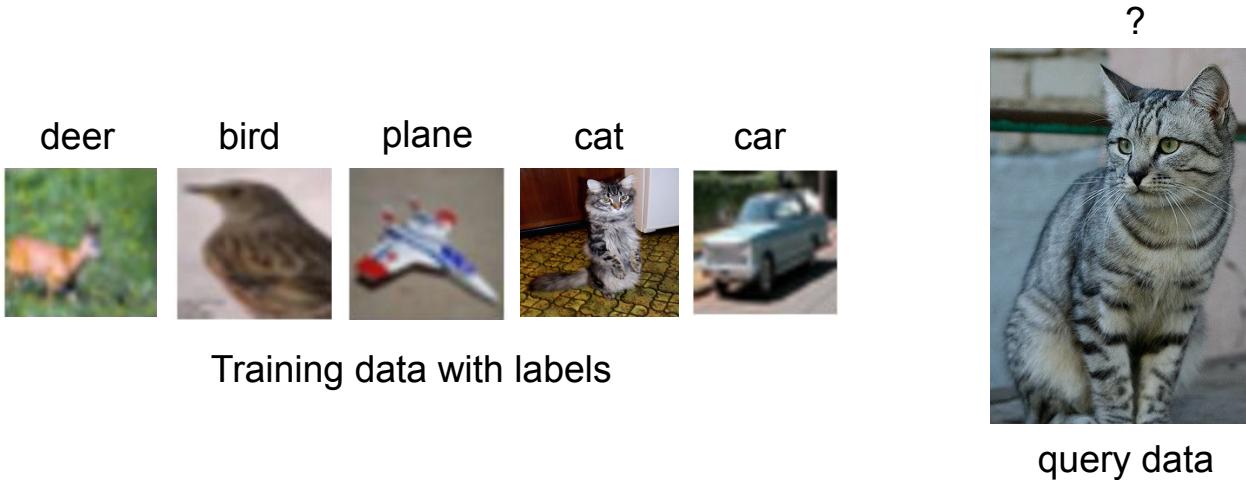
```
def train(images, labels):  
    # Machine learning!  
    return model
```

→ 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



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

# Distance Metric to compare images

L1 distance:

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

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

add  
→ 456

# Nearest Neighbor classifier

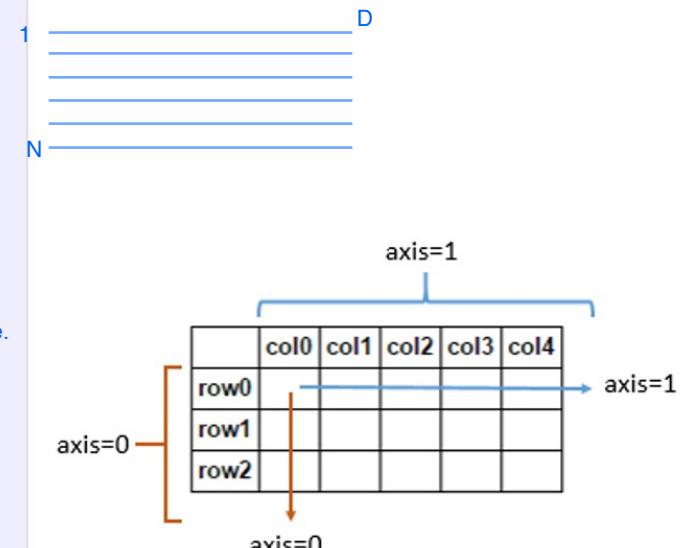
```
import numpy as np

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
```

subtract a 1d array from a 2d array where each item of 1d array subtracts from respective row in numpy



```

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

```

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

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

```

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

```

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

```

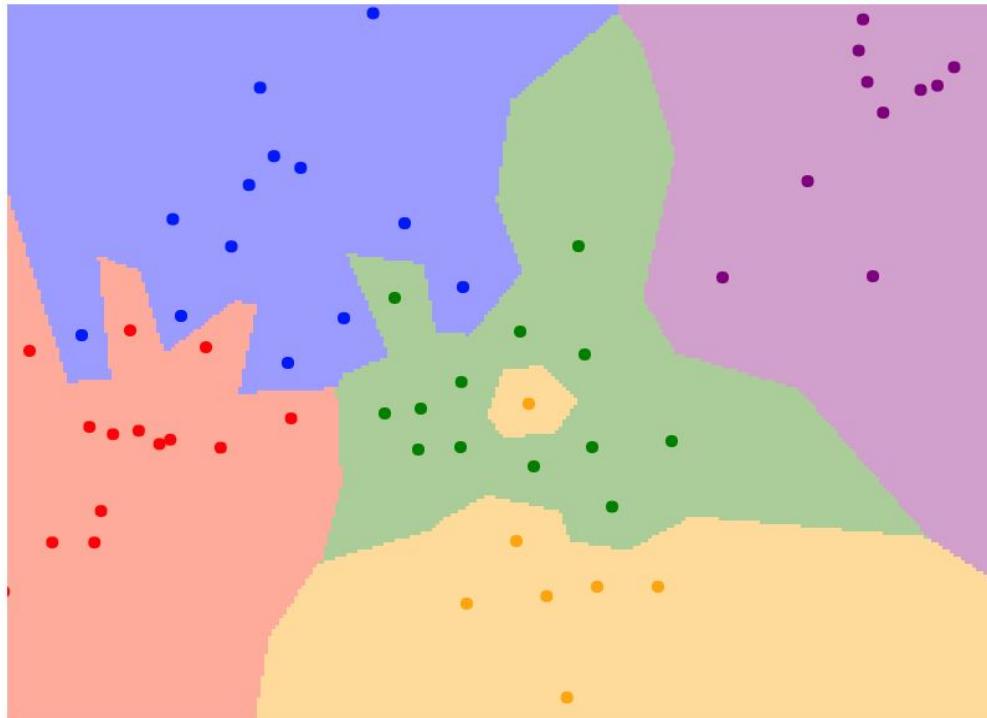
## Nearest Neighbor classifier

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

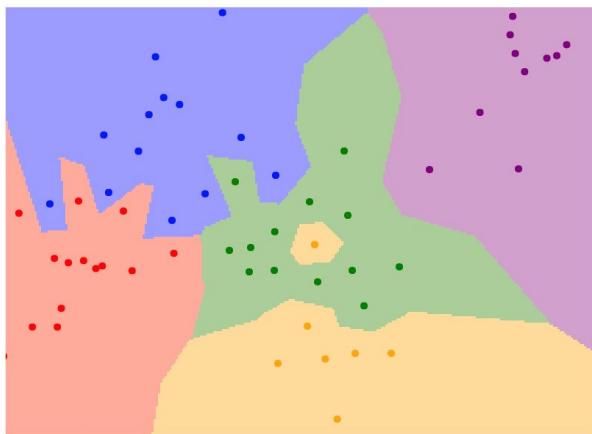
# What does this look like?



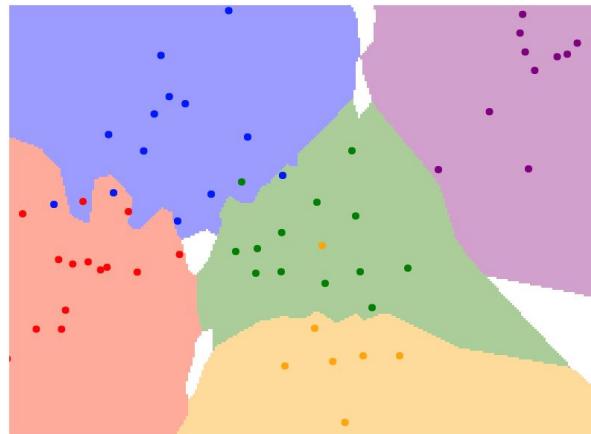
1-nearest neighbor

# K-Nearest Neighbors

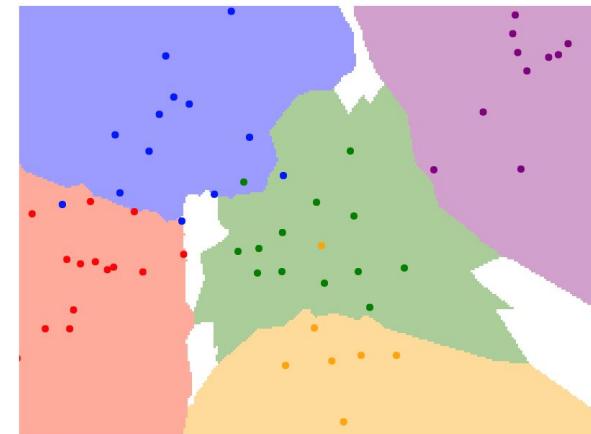
Instead of copying label from nearest neighbor,  
take **majority vote** from K closest points



$K = 1$



$K = 3$

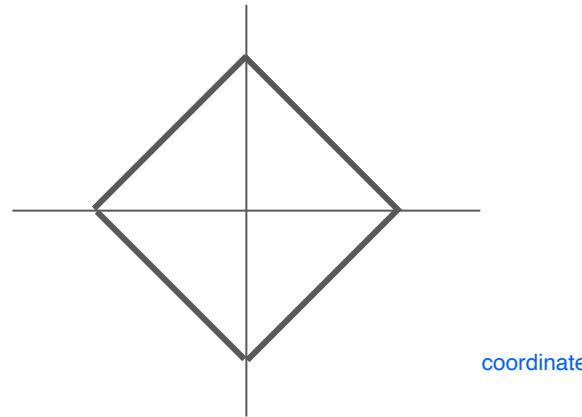


$K = 5$

# K-Nearest Neighbors: Distance Metric

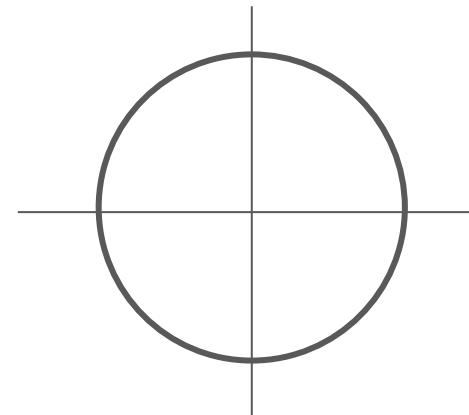
L1 (Manhattan) distance

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



L2 (Euclidean) distance

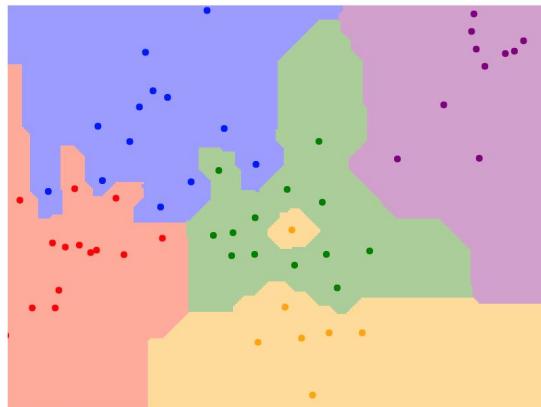
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



# K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

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



$K = 1$

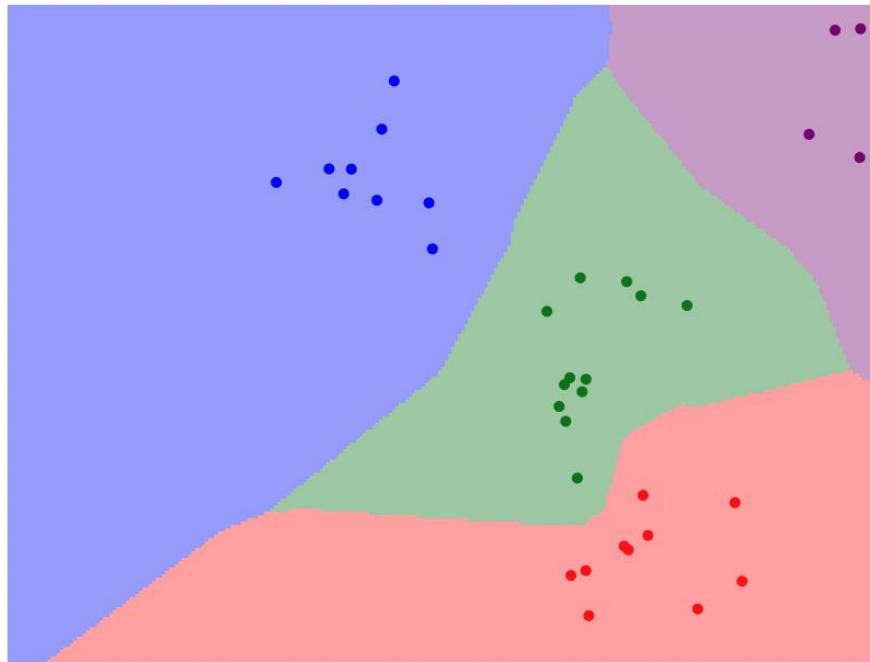
L2 (Euclidean) distance

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



$K = 1$

# 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.

we set rather than learn

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**

train

# Setting Hyperparameters

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

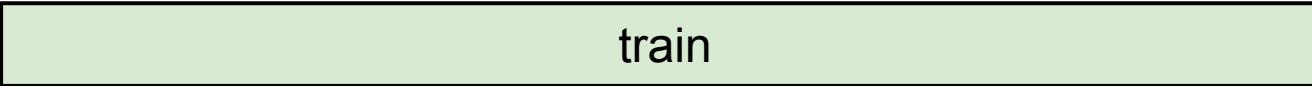
**BAD:**  $K = 1$  always works  
perfectly on training data

train

# Setting Hyperparameters

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

**BAD:**  $K = 1$  always works perfectly on training data



train

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



train

test

# Setting Hyperparameters

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

**BAD:**  $K = 1$  always works perfectly on training data



train

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

**BAD:** No idea how algorithm will perform on new data



train

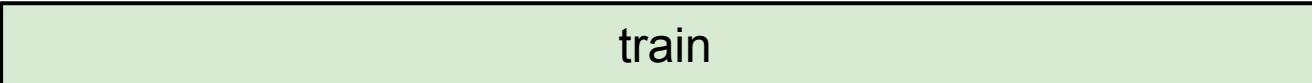
test

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



train

**Idea #2:** 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**; choose hyperparameters on val and evaluate on test

**Better!**



train

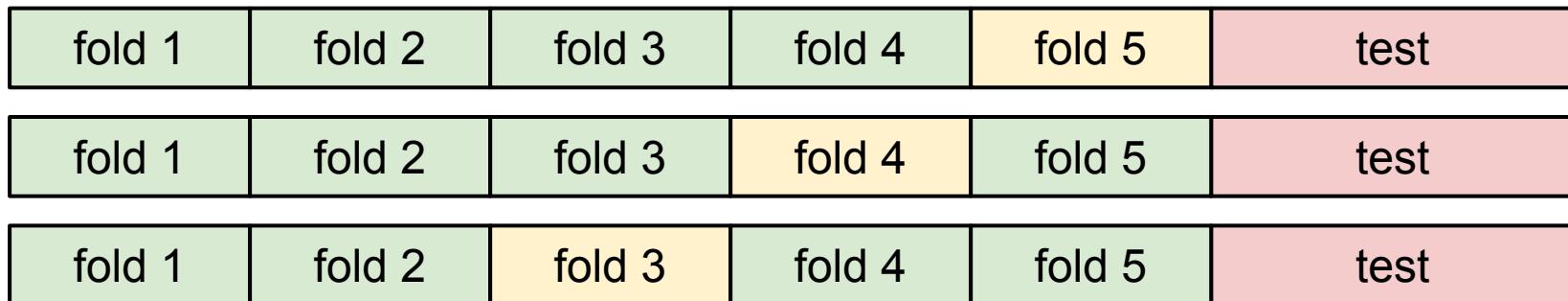
validation

test

# Setting Hyperparameters

train

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



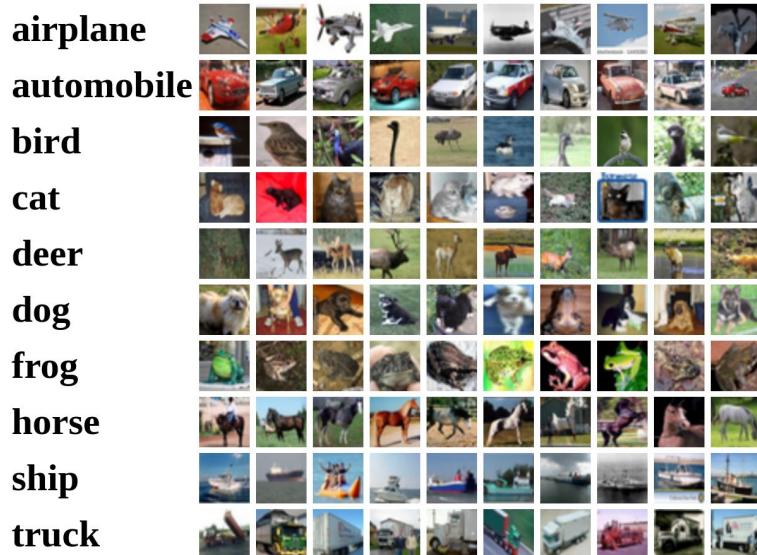
Useful for small datasets, but not used too frequently in deep learning

# Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images



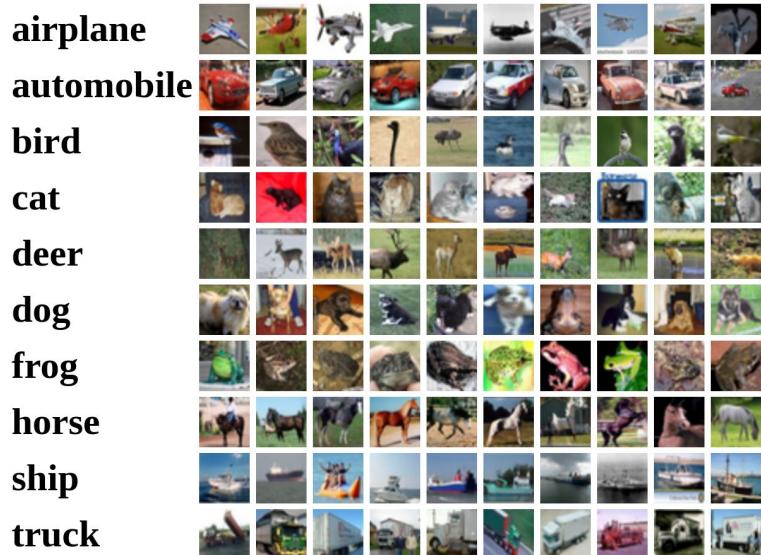
Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

# Example Dataset: CIFAR10

10 classes

50,000 training images

10,000 testing images

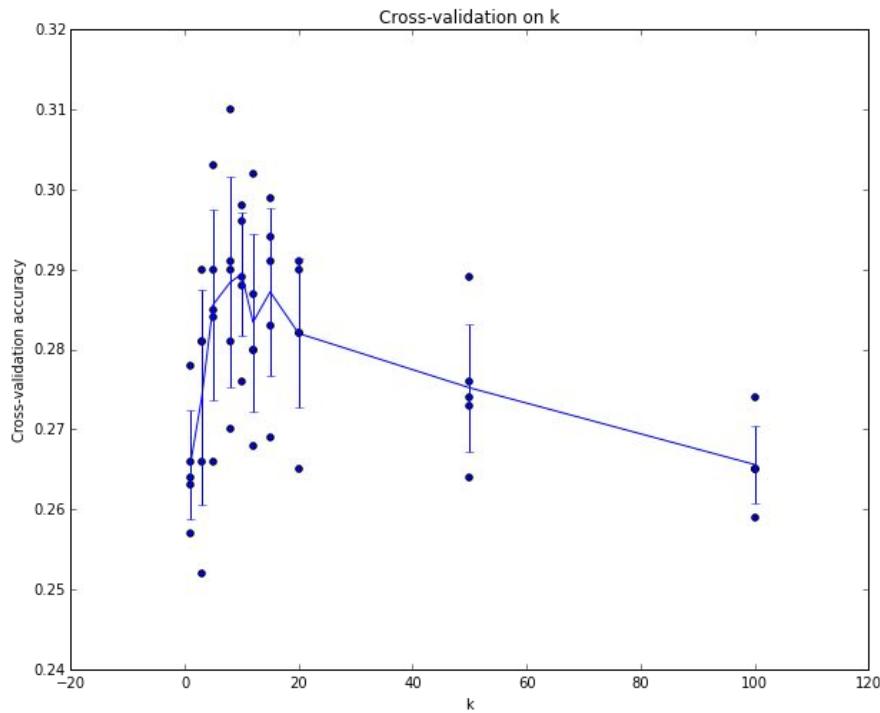


Test images and nearest neighbors



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  $k \approx 7$  works best  
for this data)

# 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

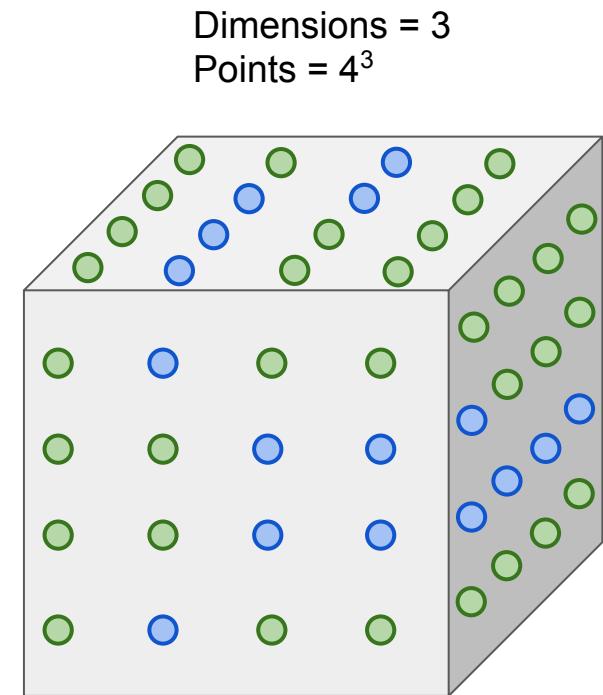
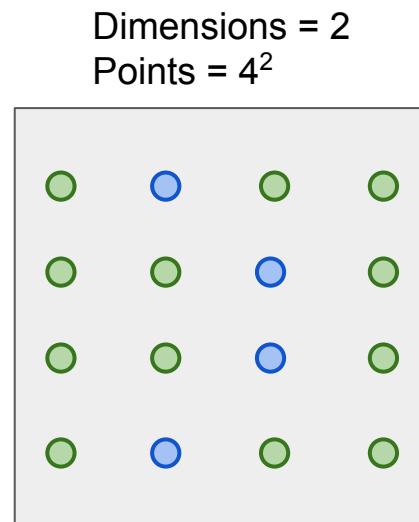
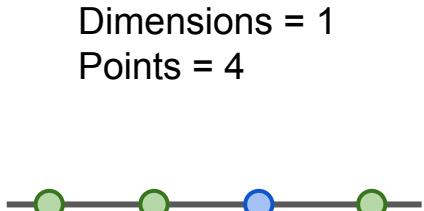
[Original image](#) is  
CC0 public domain



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

# k-Nearest Neighbor with pixel distance **never used**.

- Curse of dimensionality



# 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)

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

**W**  
parameters  
or weights

10 numbers giving  
class scores

# Parametric Approach: Linear Classifier

Image



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

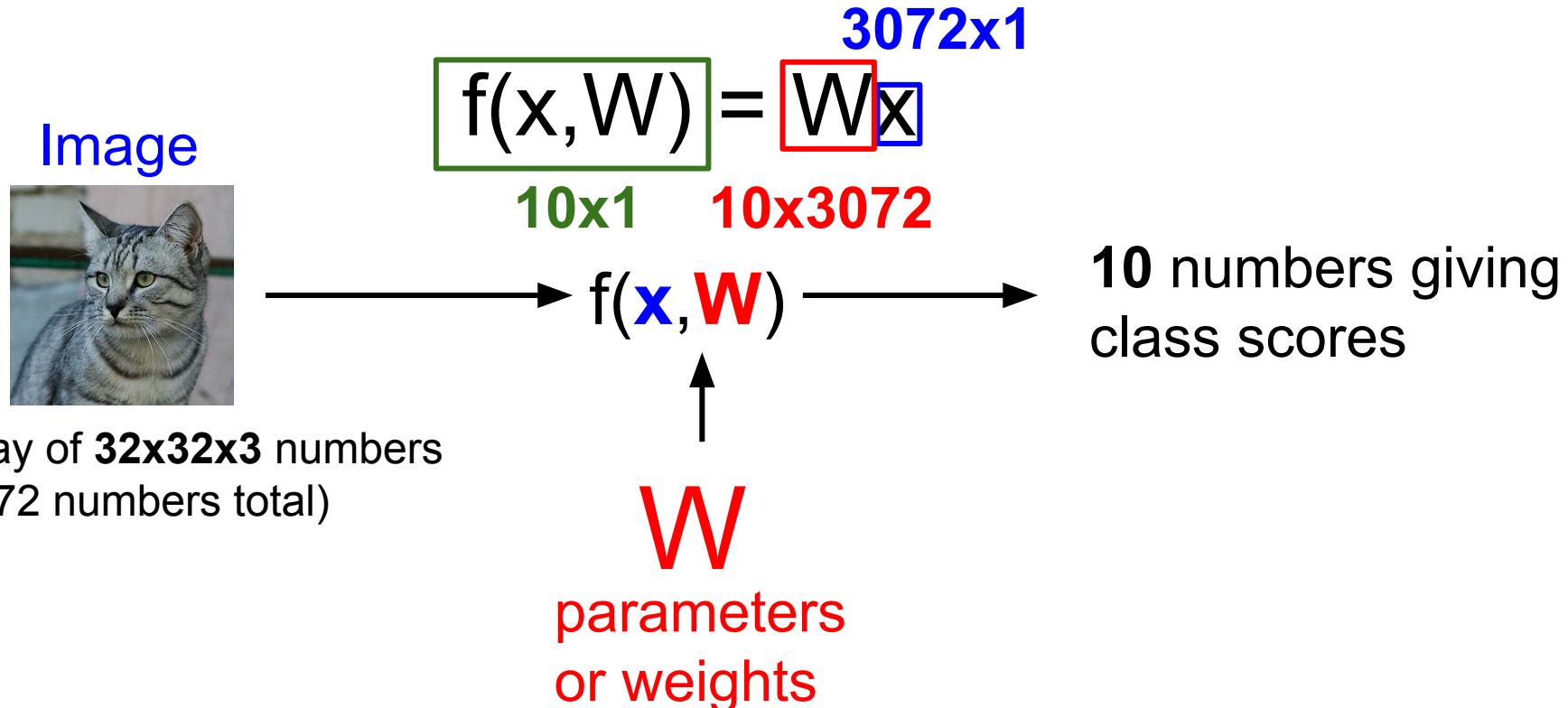
Array of **32x32x3** numbers  
(3072 numbers total)

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

10 numbers giving  
class scores

**W**  
parameters  
or weights

# Parametric Approach: Linear Classifier



# Parametric Approach: Linear Classifier



Image

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

$$f(x, W) = \boxed{W} \underset{10 \times 1}{\boxed{x}} + \boxed{b} \underset{10 \times 1}{}$$

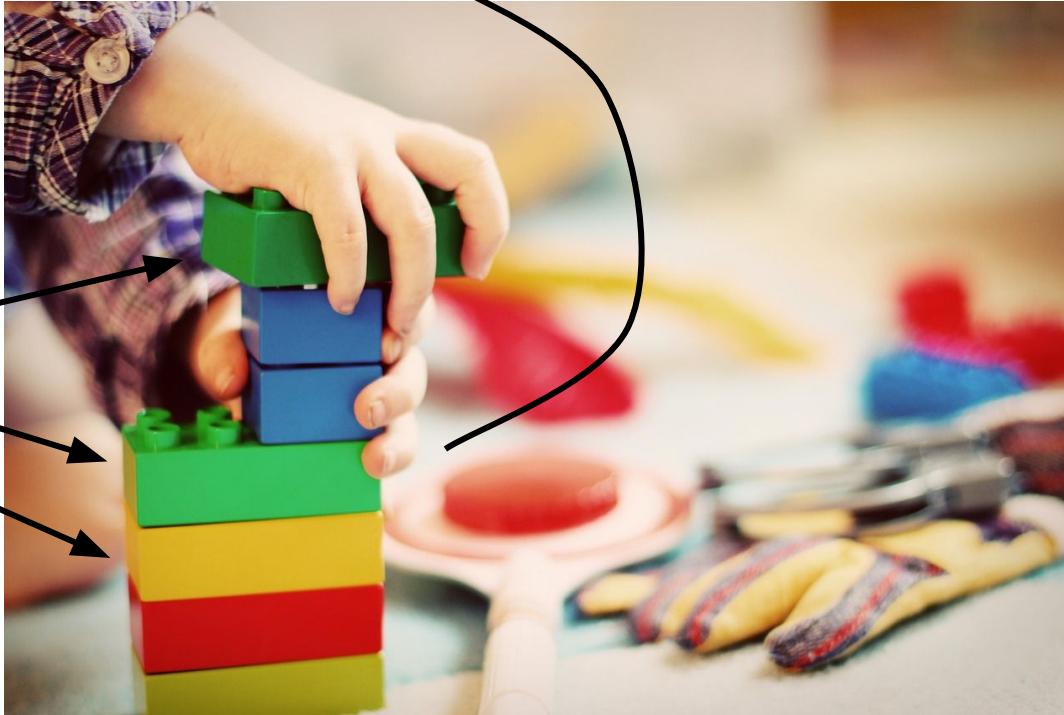
**10x1**      **10x3072**

10 numbers giving  
class scores

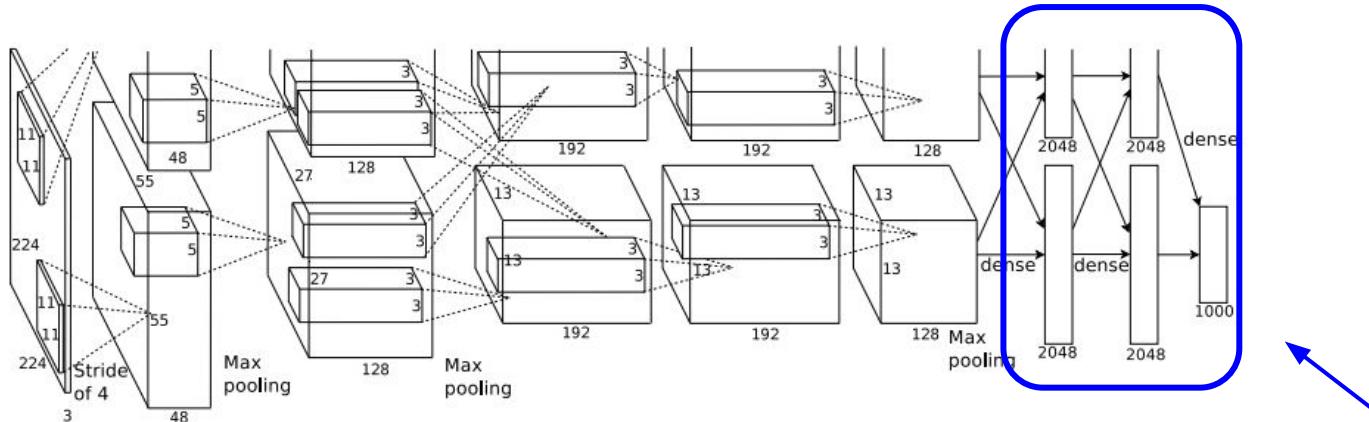
**W**  
parameters  
or weights

# Neural Network

Linear  
classifiers

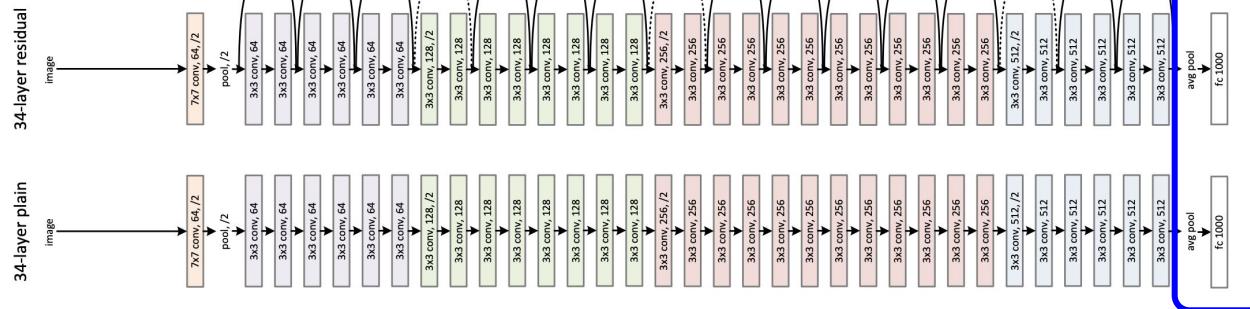


[This image](#) is [CC0 1.0](#) public domain



[Krizhevsky et al. 2012]

Linear layers



[He et al. 2015]

# Recall CIFAR10

airplane



automobile



bird



cat



deer



dog



frog



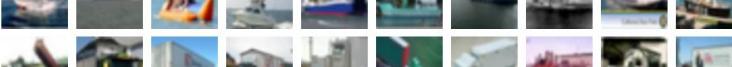
horse



ship



truck

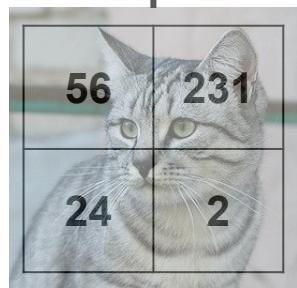


**50,000** training images  
each image is **32x32x3**

**10,000** test images.

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

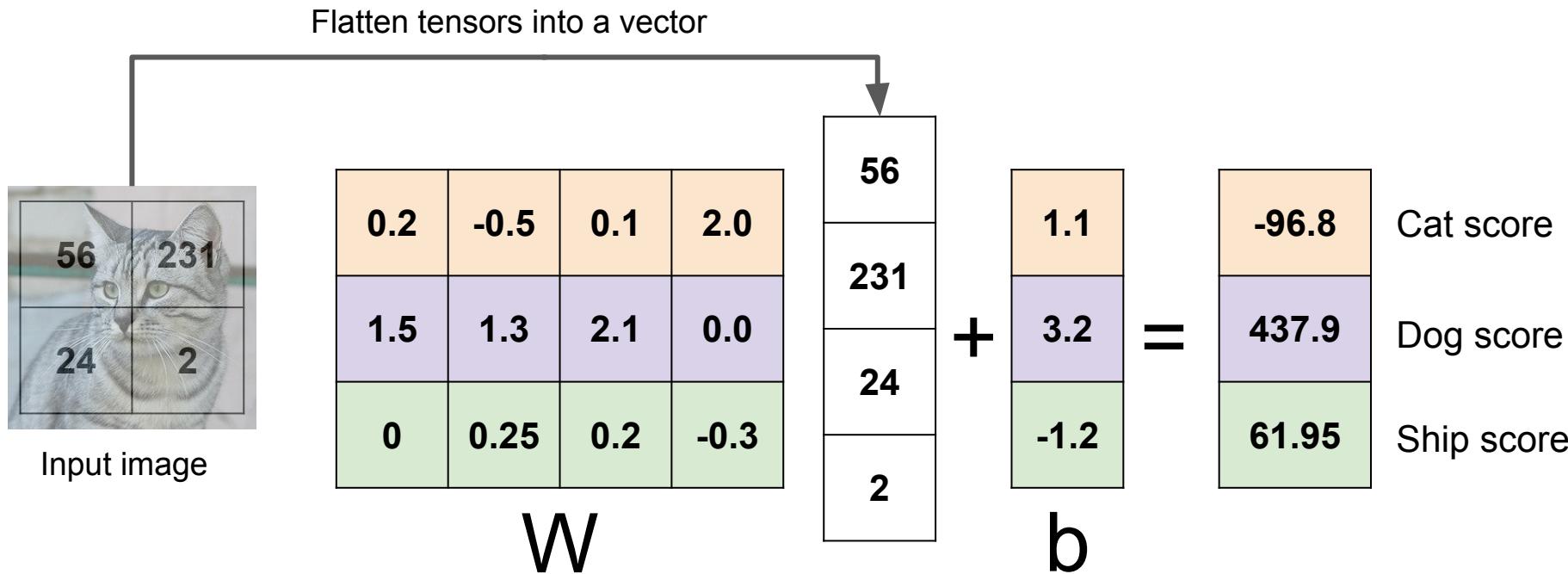
Flatten tensors into a vector



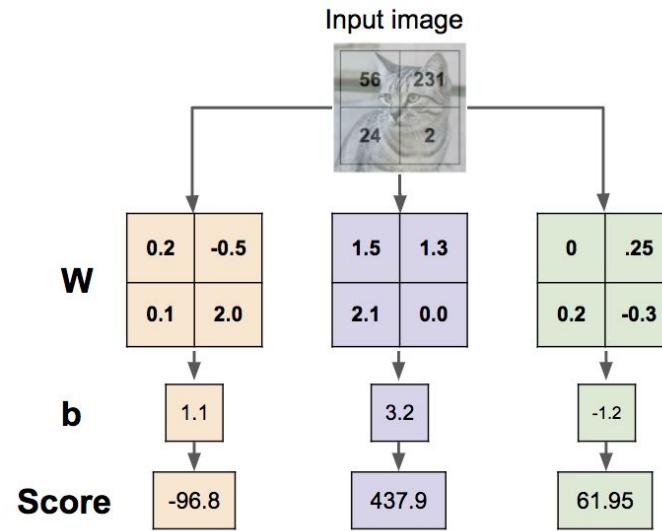
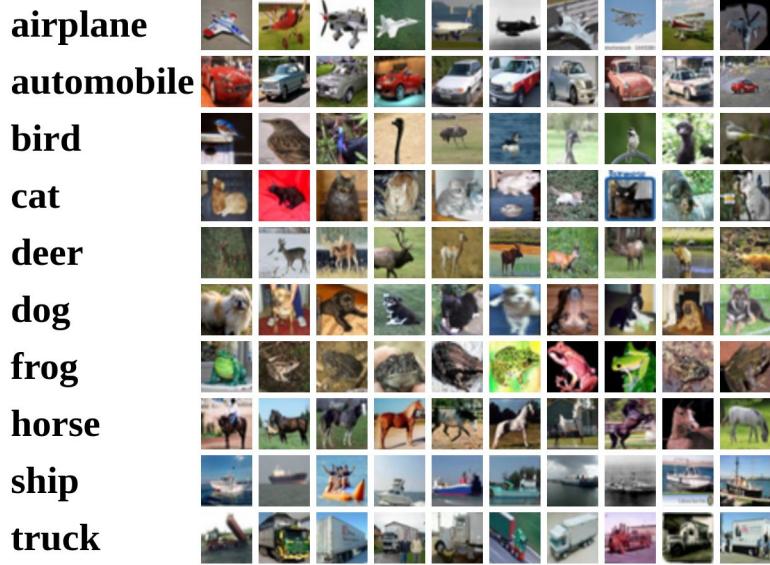
Input image



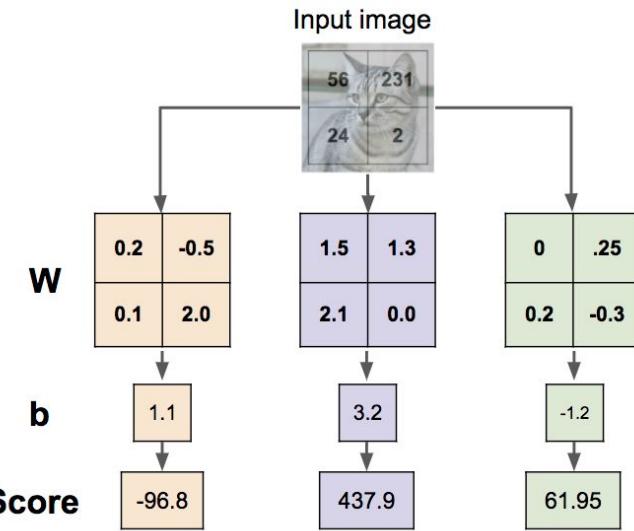
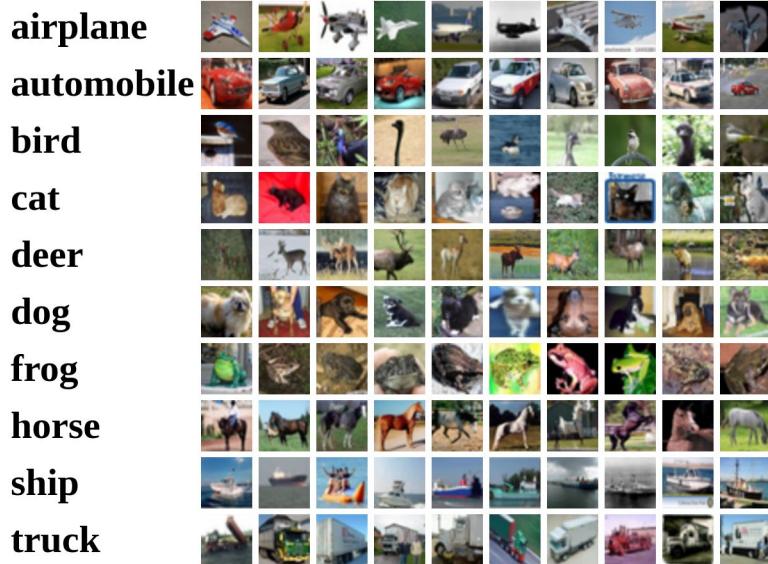
# Example with an image with 4 pixels, and 3 classes (cat/dog/ship) Algebraic Viewpoint



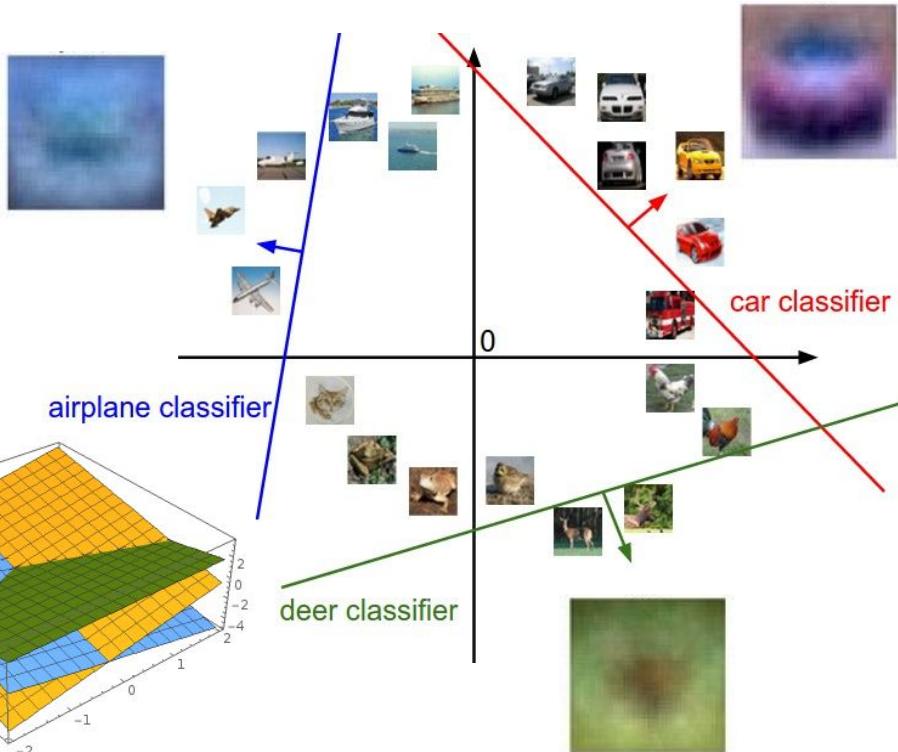
# Interpreting a Linear Classifier



# 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](#)

[Cat image](#) by [Nikita](#) is licensed under [CC-BY 2.0](#)

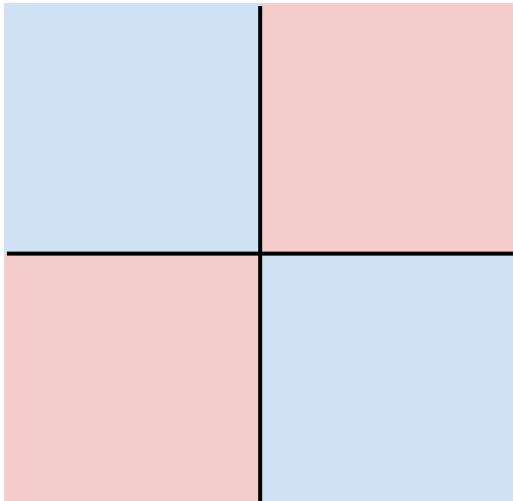
# Hard cases for a linear classifier

**Class 1:**

First and third quadrants

**Class 2:**

Second and fourth quadrants

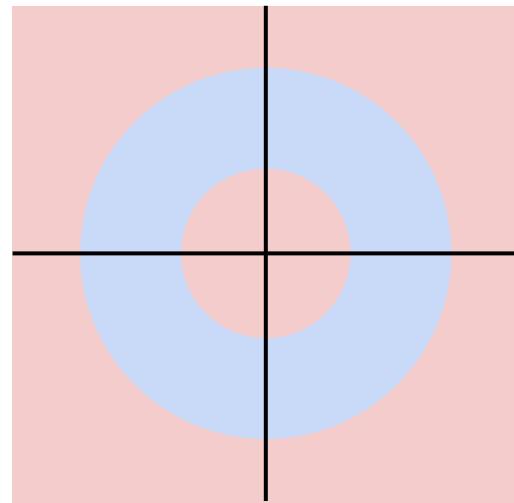


**Class 1:**

$1 \leq L_2 \text{ norm} \leq 2$

**Class 2:**

Everything else

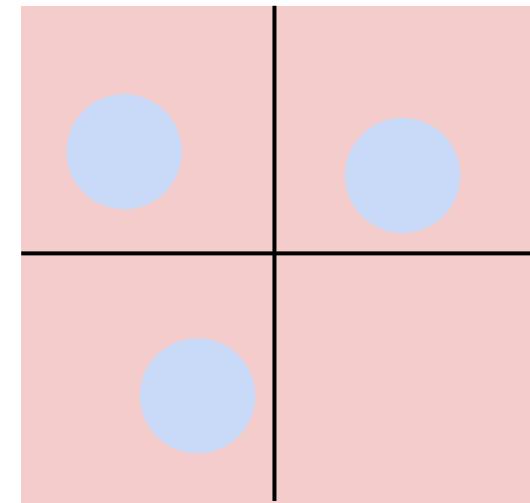


**Class 1:**

Three modes

**Class 2:**

Everything else



# Linear Classifier – Choose a good W



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

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**)

[Cat 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

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:



cat	<b>3.2</b>	1.3	2.2
car	<b>5.1</b>	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

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

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



cat	<b>3.2</b>	1.3	2.2
car	<b>5.1</b>	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

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

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

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:

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

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## 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:

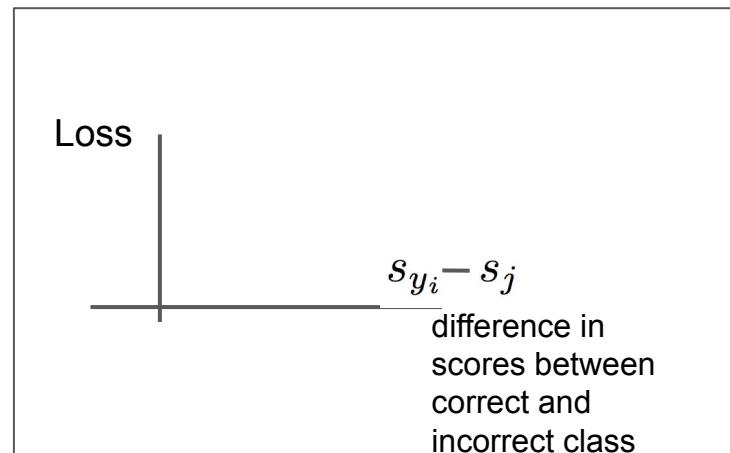
$$\begin{aligned} L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\ &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \end{aligned}$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## Interpreting Multiclass SVM loss:



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

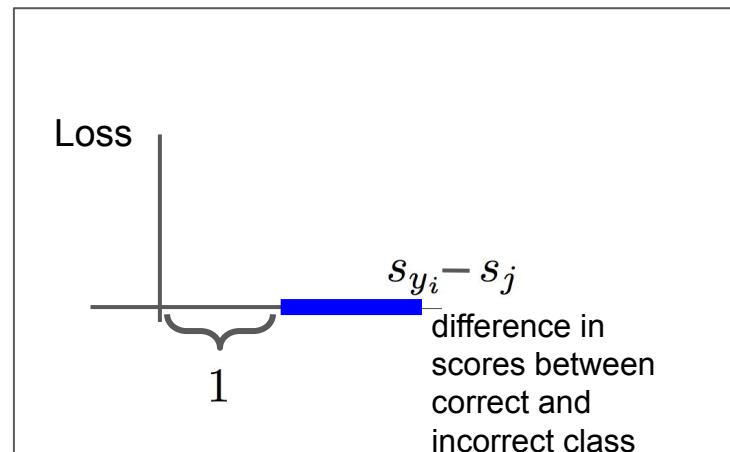
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## Interpreting Multiclass SVM loss:



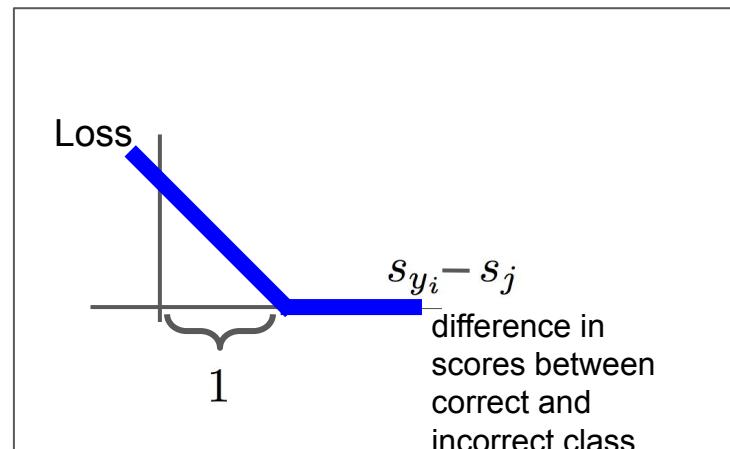
$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\
 &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
 \end{aligned}$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## Interpreting Multiclass SVM loss:



$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\
 &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
 \end{aligned}$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>

## 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:

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

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



cat	<b>3.2</b>	1.3	2.2
car	<b>5.1</b>	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>		

## 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:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 5.1 - 3.2 + 1) \\
 &\quad + \max(0, -1.7 - 3.2 + 1) \\
 &= \max(0, 2.9) + \max(0, -3.9) \\
 &= 2.9 + 0 \\
 &= 2.9
 \end{aligned}$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	2.9	0	

## 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:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 1.3 - 4.9 + 1) \\
 &\quad + \max(0, 2.0 - 4.9 + 1) \\
 &= \max(0, -2.6) + \max(0, -1.9) \\
 &= 0 + 0 \\
 &= 0
 \end{aligned}$$

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



cat	<b>3.2</b>	1.3	<b>2.2</b>
car	<b>5.1</b>	<b>4.9</b>	<b>2.5</b>
frog	-1.7	2.0	<b>-3.1</b>
Losses:	2.9	0	<b>12.9</b>

## 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:

$$\begin{aligned}
 L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\
 &= \max(0, 2.2 - (-3.1) + 1) \\
 &\quad + \max(0, 2.5 - (-3.1) + 1) \\
 &= \max(0, 6.3) + \max(0, 6.6) \\
 &= 6.3 + 6.6 \\
 &= 12.9
 \end{aligned}$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## 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:

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

Loss over full dataset is average:

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$\begin{aligned} L &= (2.9 + 0 + 12.9)/3 \\ &= 5.27 \end{aligned}$$

Suppose: 3 training examples, 3 classes.

With some  $W$  the scores  $f(x, W) = Wx$  are:

### Multiclass SVM loss:

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



cat	1.3
car	4.9
frog	2.0
Losses:	0

Q1: What happens to loss if car scores decrease by 0.5 for this training example?

Q2: what is the min/max possible SVM loss  $L_i$ ?

Q3: At initialization  $W$  is small so all  $s \approx 0$ . What is the loss  $L_i$ , assuming  $N$  examples and  $C$  classes?

1-0; debugging

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## 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:

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

Q4: What if the sum was over all classes?  
(including  $j = y_i$ )

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## 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:

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

Q5: What if we used mean instead of sum?

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## 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:

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

**Q6: What if we used**

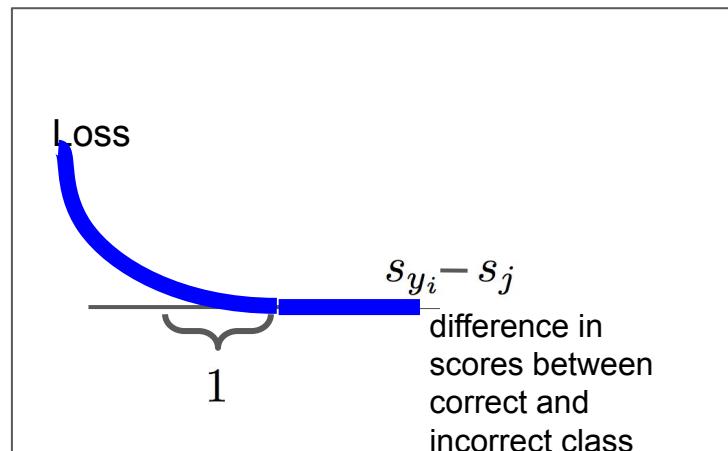
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

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



cat	<b>3.2</b>	1.3	2.2
car	5.1	<b>4.9</b>	2.5
frog	-1.7	2.0	<b>-3.1</b>
Losses:	<b>2.9</b>	0	<b>12.9</b>

## Multiclass SVM loss:



Q6: What if we used

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

# Multiclass SVM Loss: Example code

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

```
def L_i_vectorized(x, y, W):
    scores = W.dot(x)                                     # First calculate scores
    margins = np.maximum(0, scores - scores[y] + 1)      # Then calculate the margins  $s_j - s_{y_i} + 1$ 
    margins[y] = 0                                         # only sum j is not  $y_i$ , so when  $j = y_i$ , set to zero.
    loss_i = np.sum(margins)                               # sum across all j
    return loss_i
```

# Softmax classifier

# Softmax Classifier (Multinomial Logistic Regression)

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



cat	<b>3.2</b>
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	<b>3.2</b>
car	5.1
frog	-1.7

# Softmax Classifier (Multinomial Logistic Regression)

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



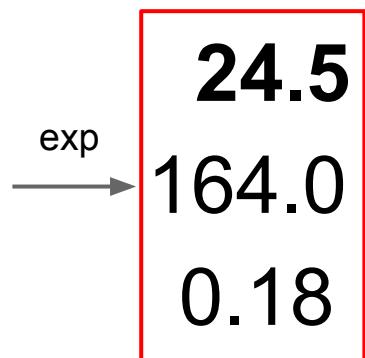
$$s = f(x_i; W)$$

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

Softmax  
Function

Probabilities  
must be  $\geq 0$

cat	3.2
car	5.1
frog	-1.7



unnormalized  
probabilities

# Softmax Classifier (Multinomial Logistic Regression)

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



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

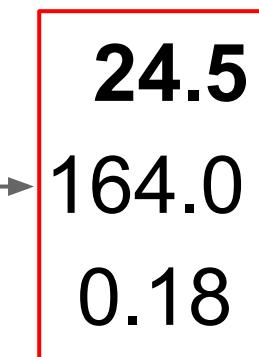
Probabilities  
must be  $\geq 0$

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

Softmax  
Function

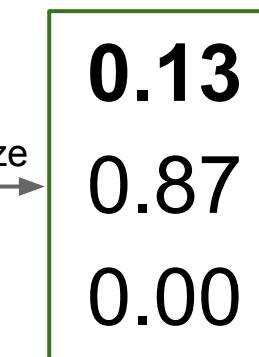
cat	3.2
car	5.1
frog	-1.7

exp



unnormalized  
probabilities

normalize



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

cat  
car  
frog

3.2
5.1
-1.7

Unnormalized  
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized  
probabilities

normalize

0.13
0.87
0.00

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  $\geq 0$

Probabilities  
must sum to 1

cat  
car  
frog

3.2
5.1
-1.7

Unnormalized  
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized  
probabilities

normalize

0.13
0.87
0.00

probabilities

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

$$\rightarrow L_i = -\log(0.13) \\ = 2.04$$

# 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  
car  
frog

3.2
5.1
-1.7

Unnormalized  
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized  
probabilities

normalize

0.13
0.87
0.00

probabilities

Probabilities  
must sum to 1

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

$$\rightarrow L_i = -\log(0.13) \\ = 2.04$$

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**



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

Probabilities  
must be  $\geq 0$

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

Softmax  
Function

cat  
car  
frog

3.2
5.1
-1.7

Unnormalized  
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized  
probabilities

normalize

0.13
0.87
0.00

probabilities

compare

1.00
0.00
0.00

Correct  
probs

# 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  
car  
frog

3.2
5.1
-1.7

Unnormalized  
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized  
probabilities

normalize

0.13
0.87
0.00

probabilities

$$D_{KL}(P\|Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

1.00

0.00

0.00

Correct  
probs

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

Probabilities  
must sum to 1

compare

Kullback–Leibler  
divergence

$$D_{KL}(P\|Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

# Softmax Classifier (Multinomial Logistic Regression)



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

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

Probabilities  
must be  $\geq 0$

cat  
car  
frog

3.2
5.1
-1.7

Unnormalized  
log-probabilities / logits

exp

24.5
164.0
0.18

unnormalized  
probabilities

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

Softmax  
Function

Probabilities  
must sum to 1

0.13
0.87
0.00

probabilities

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

compare

1.00

0.00

0.00

Cross Entropy

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

Correct  
probs

# 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

cat	3.2
car	5.1
frog	-1.7

Putting it all together:

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

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

# 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

Putting it all together:

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

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

cat	3.2
car	5.1
frog	-1.7

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

(0, infinity)

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

cat	3.2
car	5.1
frog	-1.7

Putting it all together:

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

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

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

# Softmax vs. SVM

matrix multiply + bias offset

0.01	-0.05	0.1	0.05
0.7	0.2	0.05	0.16
0.0	-0.45	-0.2	0.03

$W$

-15
22
-44
56

$x_i$

$y_i$

2

+

0.0
0.2
-0.3

$b$

-2.85
0.86
0.28

hinge loss (SVM)

$$\begin{aligned} & \max(0, -2.85 - 0.28 + 1) + \\ & \max(0, 0.86 - 0.28 + 1) \\ & = \\ & \mathbf{1.58} \end{aligned}$$

cross-entropy loss (Softmax)

-2.85
0.86
0.28

$\exp$

0.058
2.36
1.32

normalize  
(to sum  
to one)

→

0.016
0.631
0.353

$$\begin{aligned} & -\log(0.353) \\ & = \\ & \mathbf{0.452} \end{aligned}$$

# Softmax vs. SVM

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

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

# Softmax vs. SVM

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

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

assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and  $y_i = 0$

Q: What is the **softmax loss** and the **SVM** loss?

# Softmax vs. SVM

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

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

assume scores:

[20, -2, 3]

[20, 9, 9]

[20, -100, -100]

and

$$y_i = 0$$

Q: What is the **softmax loss** and the **SVM loss if I double the correct class score from 10 -> 20?**

# Coming up:

- Regularization
- Optimization

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

