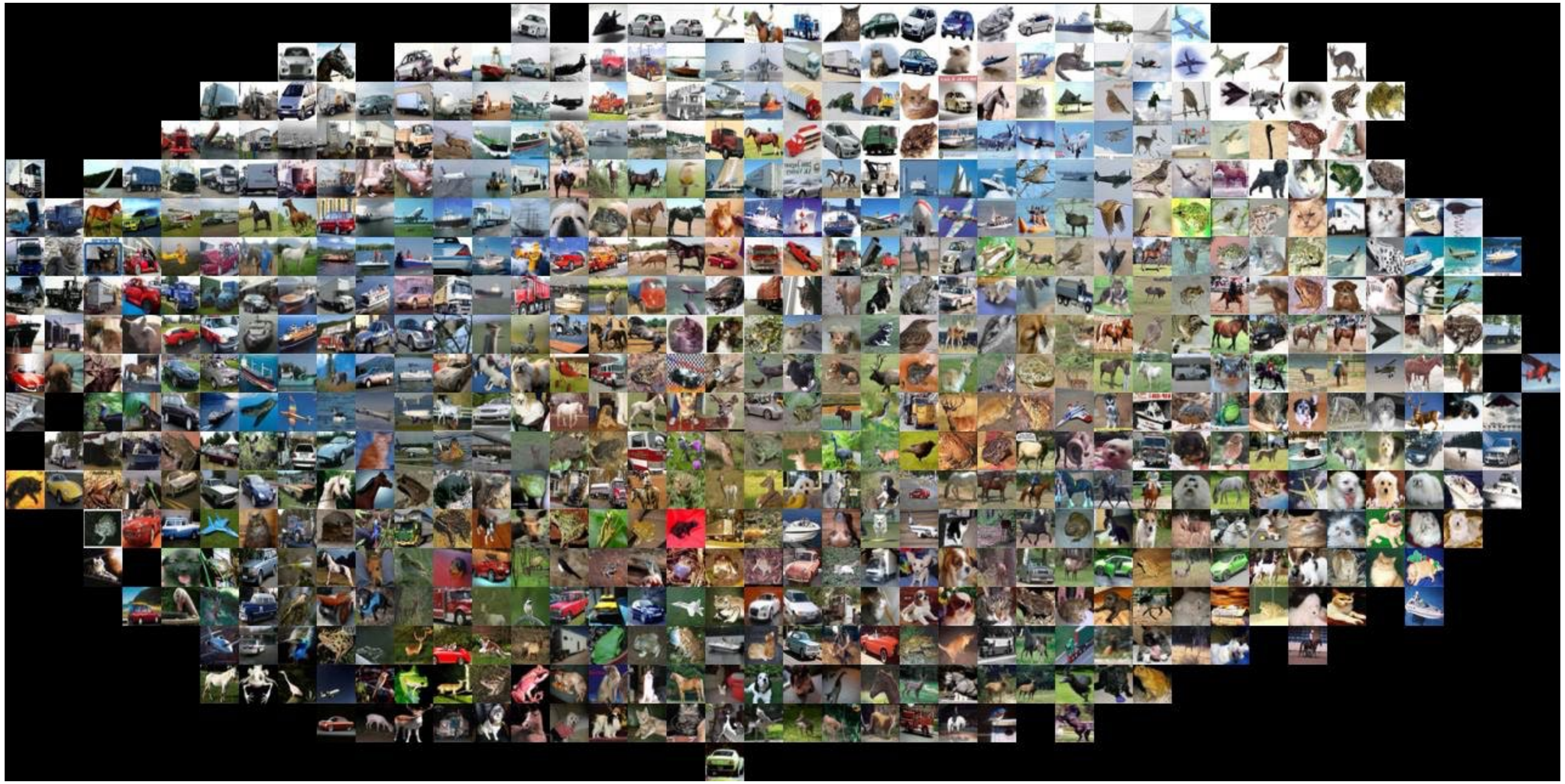
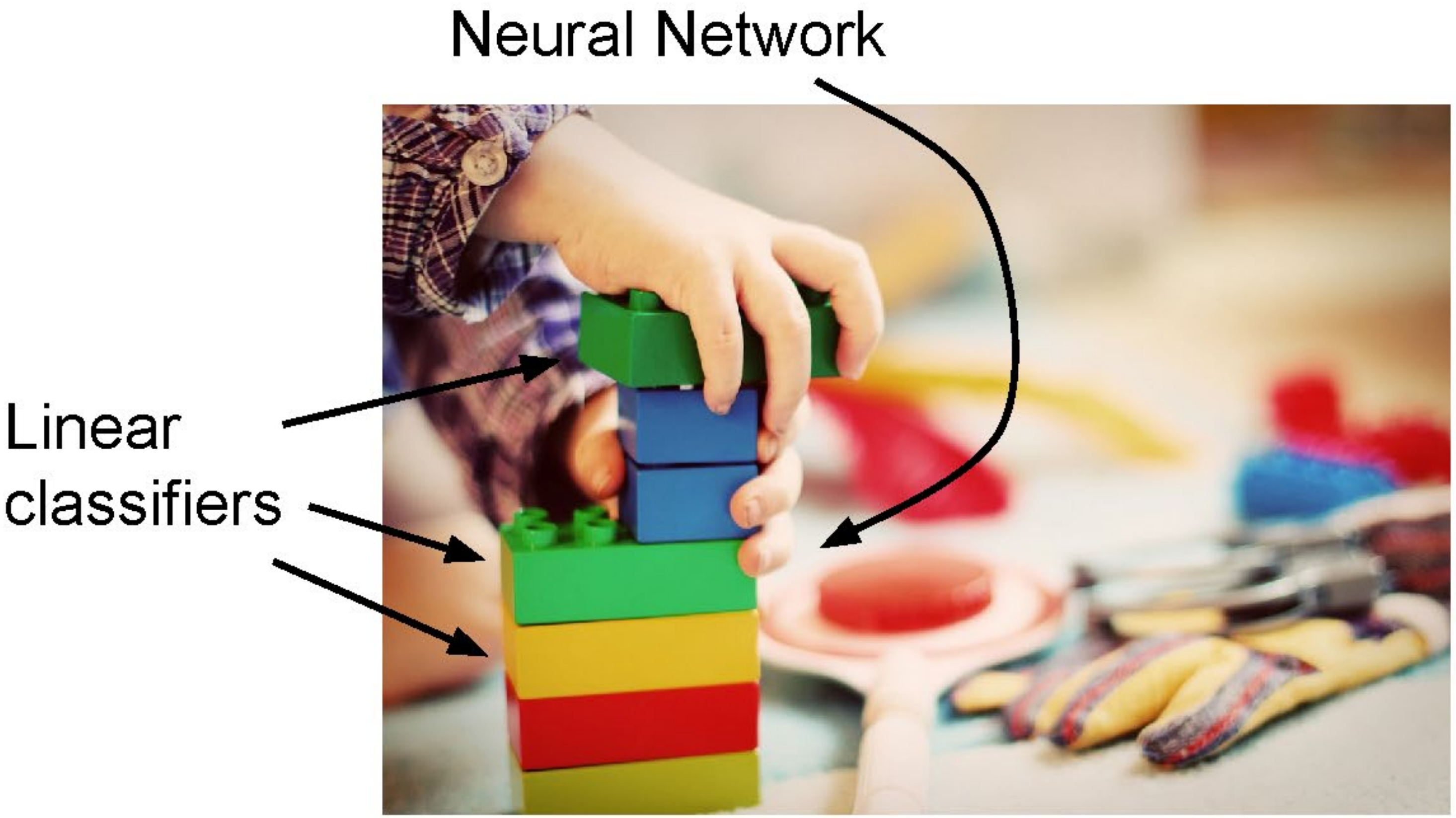
CS7GV1: Computer Vision

Linear Classifiers



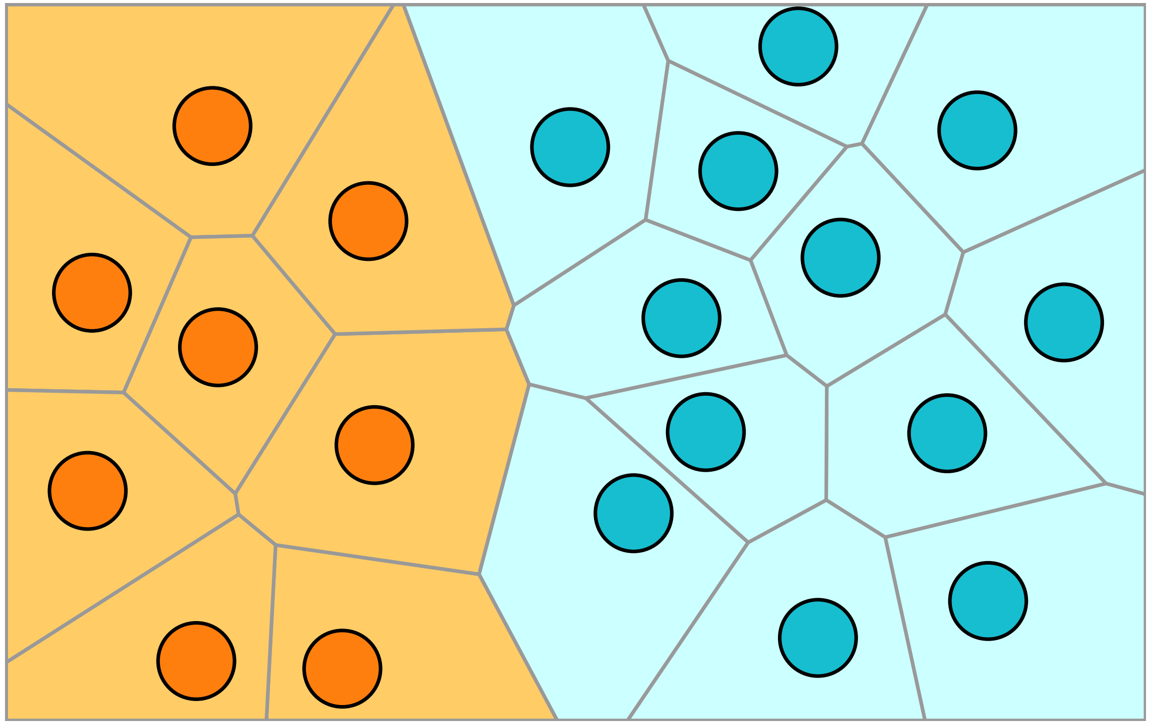
Credits: Some Slides from Noah Snavely and Abe Davis (Cornell)
and Fei-Fei Li, Justin Johnson, Serena Yeung (Stanford)
http://vision.stanford.edu/teaching/cs231n/

Linear Classifiers



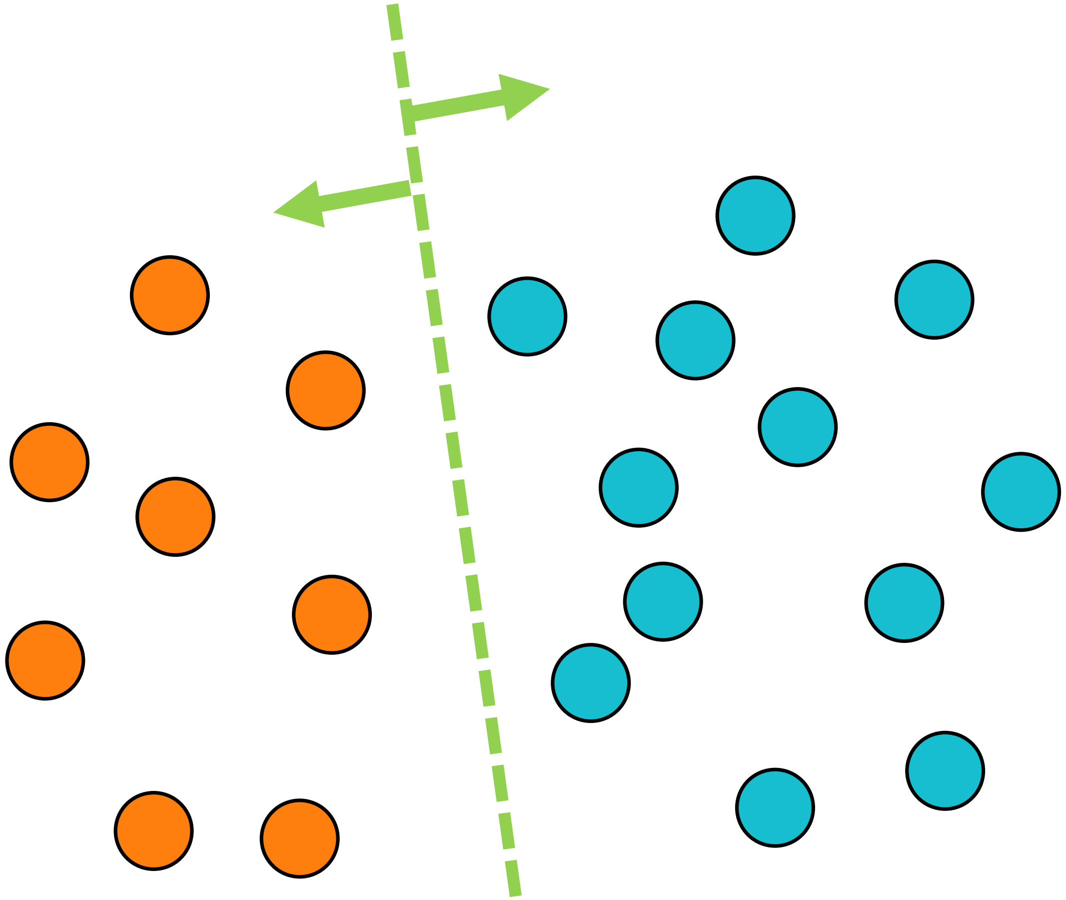
Linear Classification vs. Nearest Neighbors

- Nearest Neighbors
  - Store every image
  - Find nearest neighbors at test time, and assign same class



Linear Classification vs. Nearest Neighbors

- Nearest Neighbors
  - Store every image
  - Find nearest neighbors at test time, and assign same class
- Linear Classifier
  - Store hyperplanes that best separate different classes
  - We can compute continuous class score by calculating (signed) distance from hyperplane



We can interpret this as a linear "score function" for each class.

Score functions

class scores

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Parametric Approach

image parameters

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

[32x32x3] $= 3072$
array of numbers 0...1
(3072 numbers total)

10 numbers,
indicating class
scores

Parametric Approach: Linear Classifier

Image

3072x1

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

10x1 **10x3072**

10 numbers, indicating class scores

[32x32x3] $= 3072$
array of numbers 0...1

parameters, or “weights”

Parametric Approach: Linear Classifier

"Bias" vector

Image

3072x1

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

10x1

10x3072

10x1

[32x32x3] $= 3072$

array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Linear Classifier

define a **score function**

data (image)

$$f(x_i, W, b) = W x_i + b$$

class scores

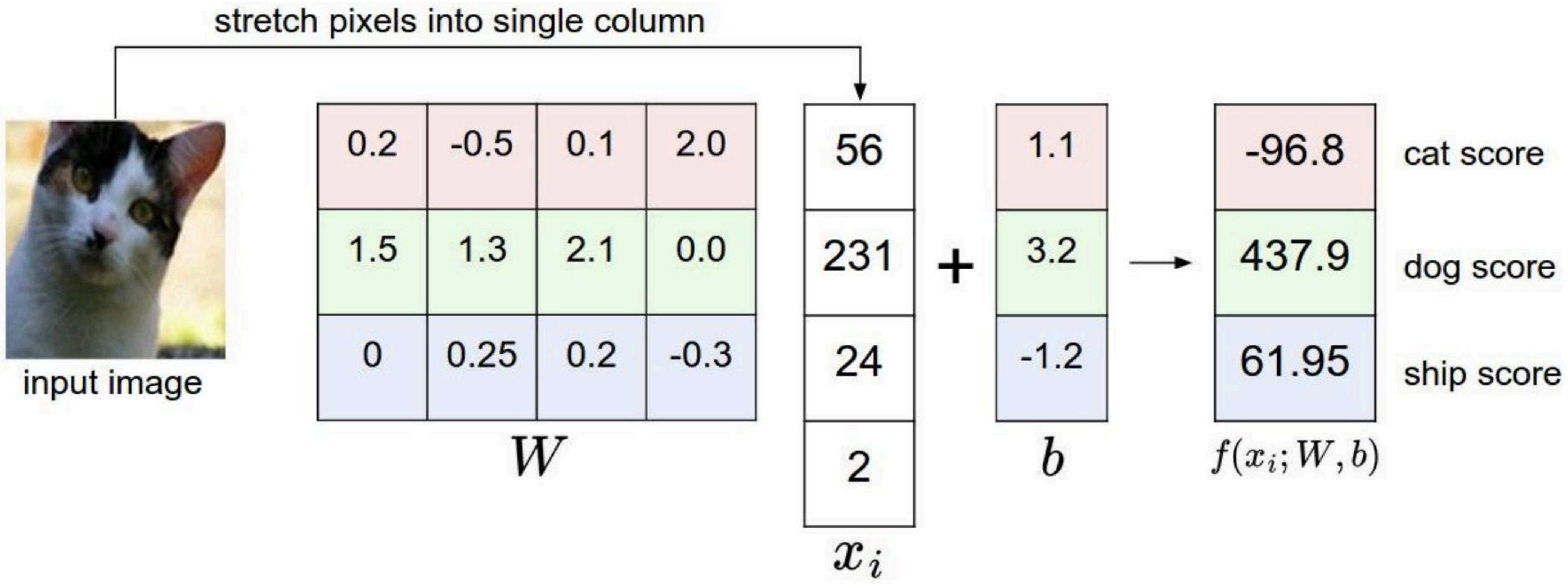
"weights"

"bias vector"

"parameters"

Interpretation: Algebraic

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



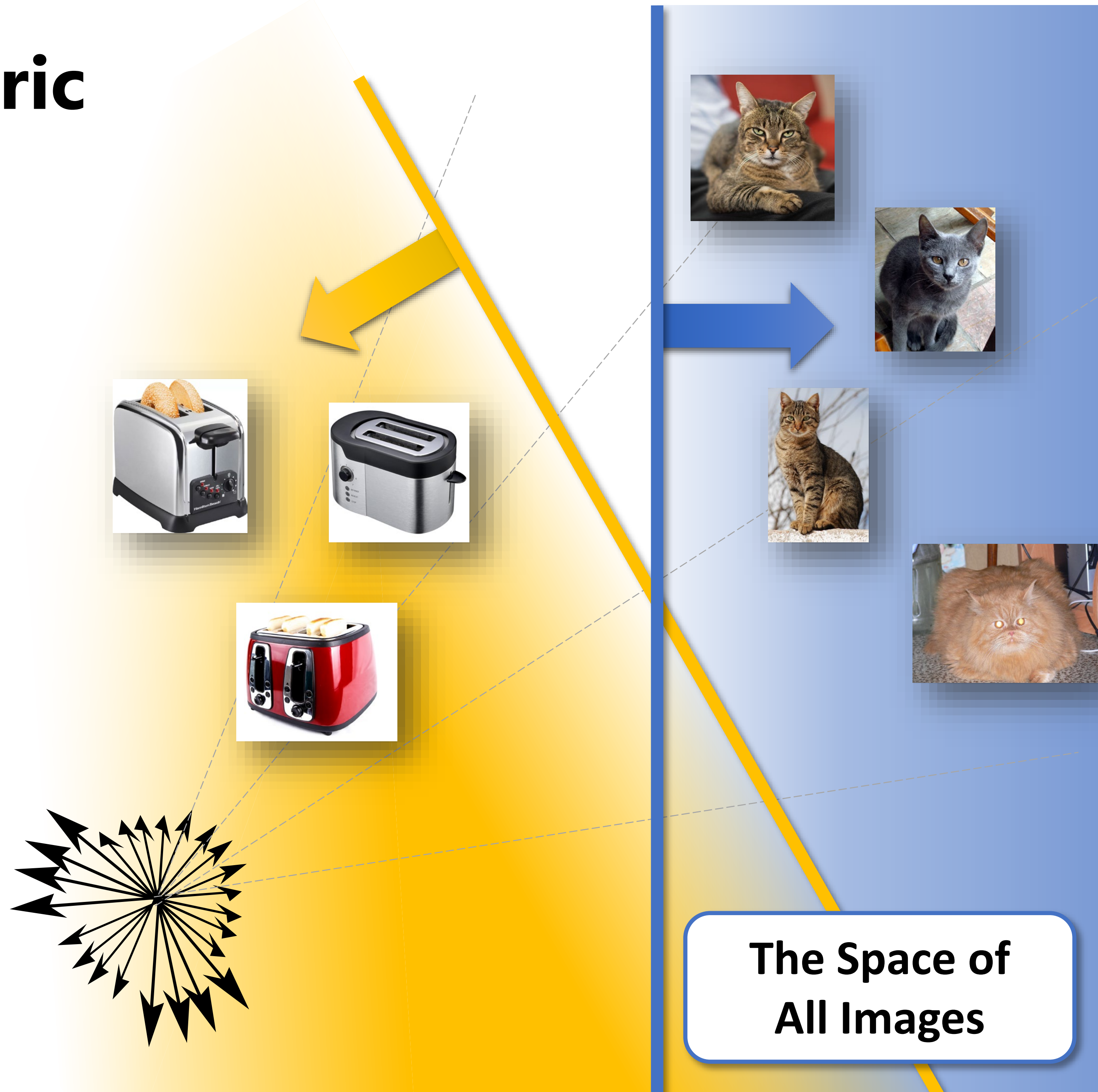
Slide adapted from Andrej Karpathy and Fei-Fei Li http://vision.stanford.edu/teaching/cs231n/

Interpretation: Geometric

- Parameters define a hyperplane for each class:

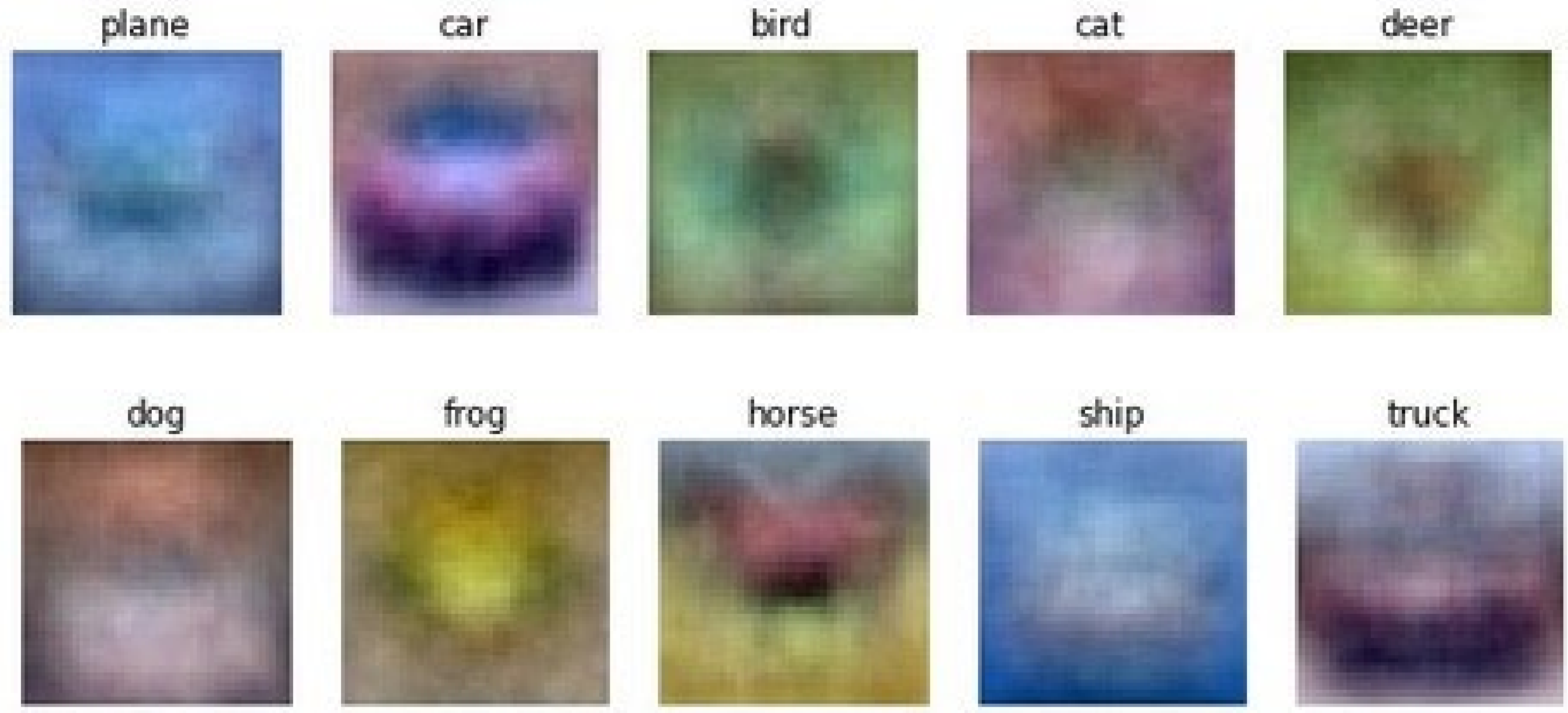
$$f(x_i, W, b) = Wx_i + b$$

- We can think of each class score as defining a distribution that is proportional to distance from the corresponding hyperplane
- Weight and Bias Effect:
  - The effect of changing the weight will change the line angle, while changing the bias, will move the line left/right



Interpretation: Template matching

- We can think of the rows in W as templates for each class

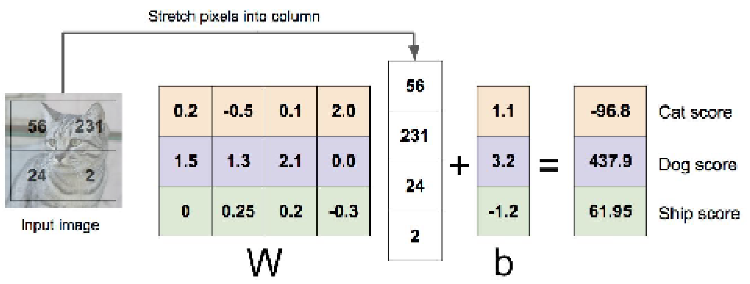


Rows of W in $f(x_i, W, b) = Wx_i + b$

Linear Classifier: Three Viewpoints

<u>Algebraic Viewpoint</u>

$f(x,W) = Wx$

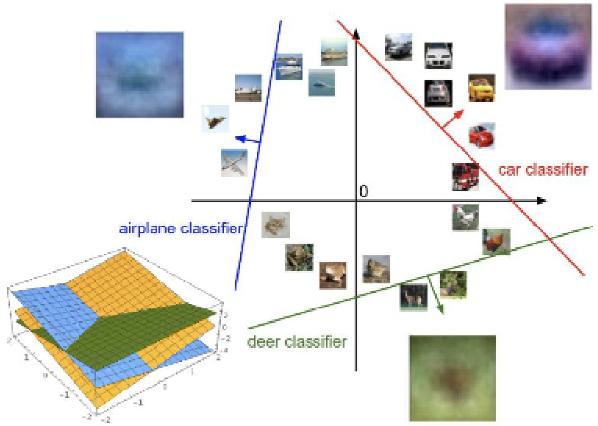


<u>Visual Viewpoint</u>

One template
per class

<u>Geometric Viewpoint</u>

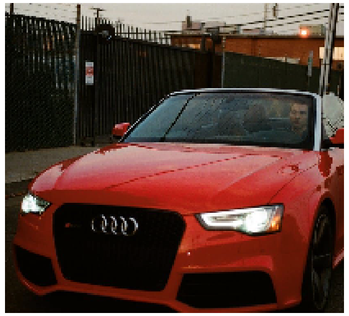
Hyperplanes
cutting up space



So far: Defined a (linear) score function $f(x,W) = Wx + b$

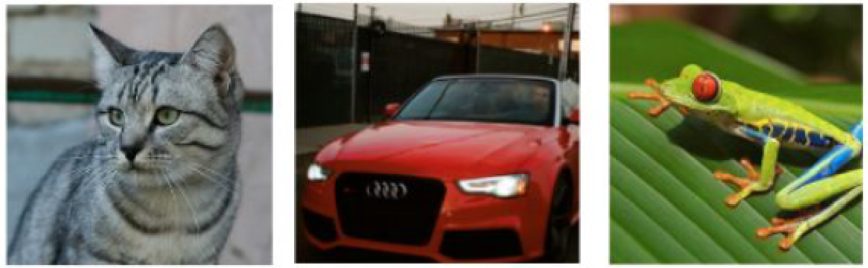
Example class scores for 3 images for some W:

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

Linear classification



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

Output scores

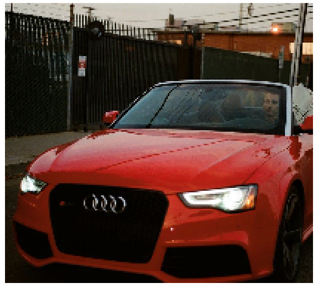
TODO:

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 functions

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/objective function

- Given ground truth labels (y_i), scores $f(x_i, \mathbf{W})$
  - how unhappy are we with the scores?

- Loss function or objective/cost function measures unhappiness

- During training, **want to find the parameters W that minimize the loss function**

Simpler example: binary classification

- Two classes (e.g., “cat” and “not cat”)
  - AKA “positive” and “negative” classes

cat

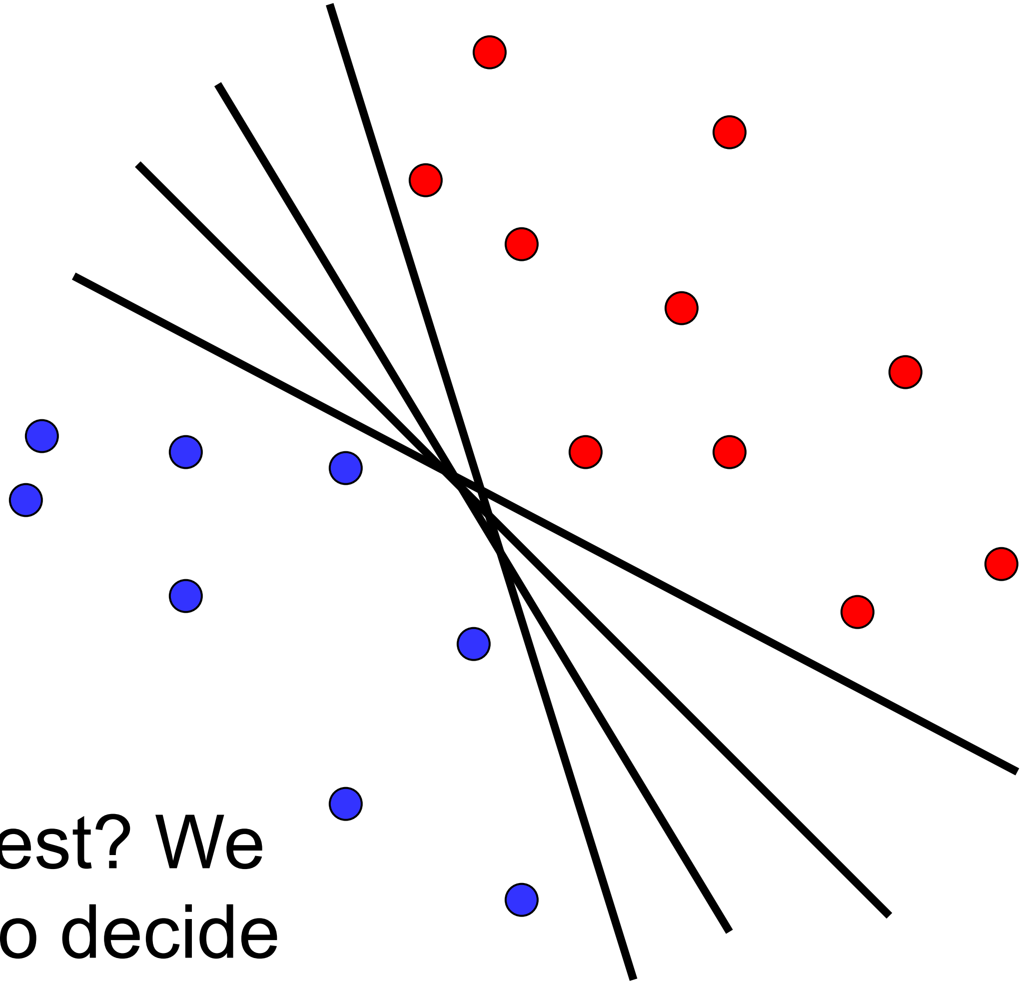
not cat

Linear classifiers

- Find linear function (*hyperplane*) to separate positive and negative examples

$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

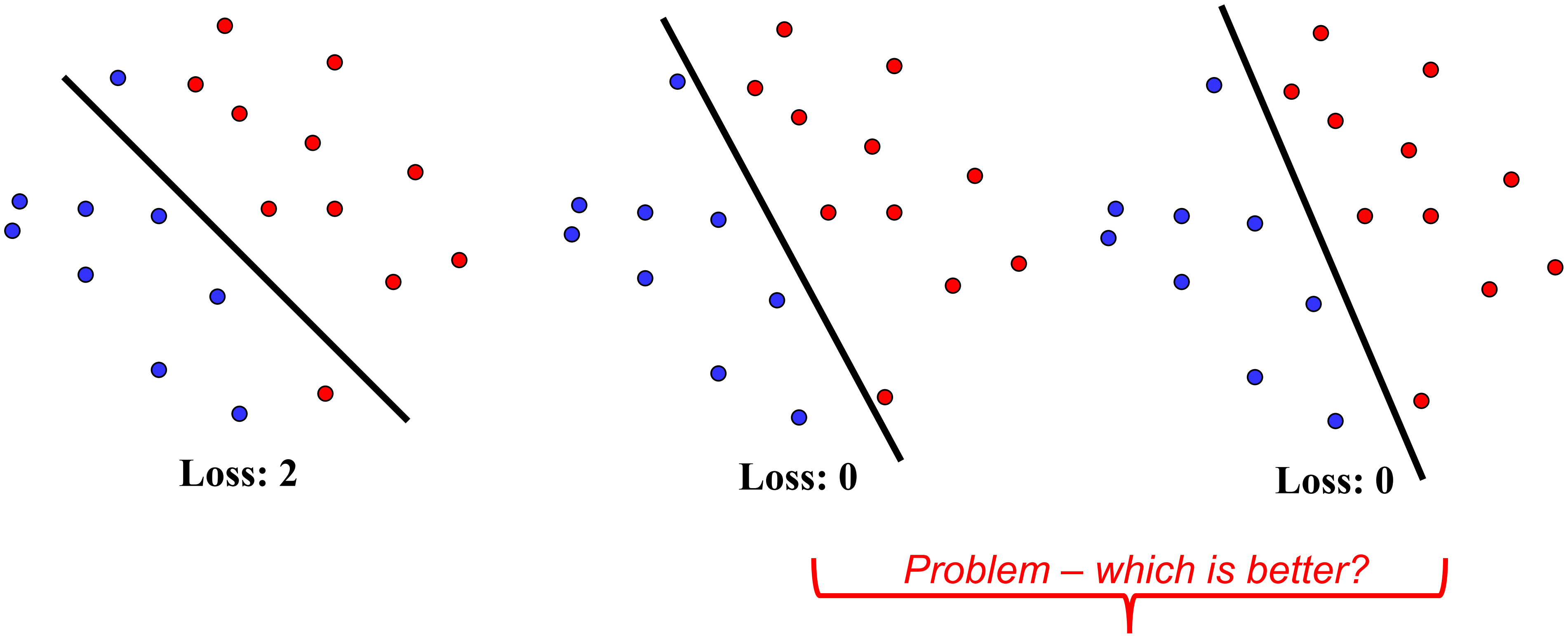
$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$



Which hyperplane is best? We need a **loss function** to decide

What is a good loss function?

- One possibility: Number of misclassified examples
  - Problems: discrete, can't break ties
  - We want the loss to lead to *good generalization*
  - We want the loss to work for more than 2 classes



Softmax classifier

- Interpret Scores as unnormalized log probabilities of classes

$f(x_i, W) = Wx_i$ (score function)

$\boxed{\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}}$ softmax function

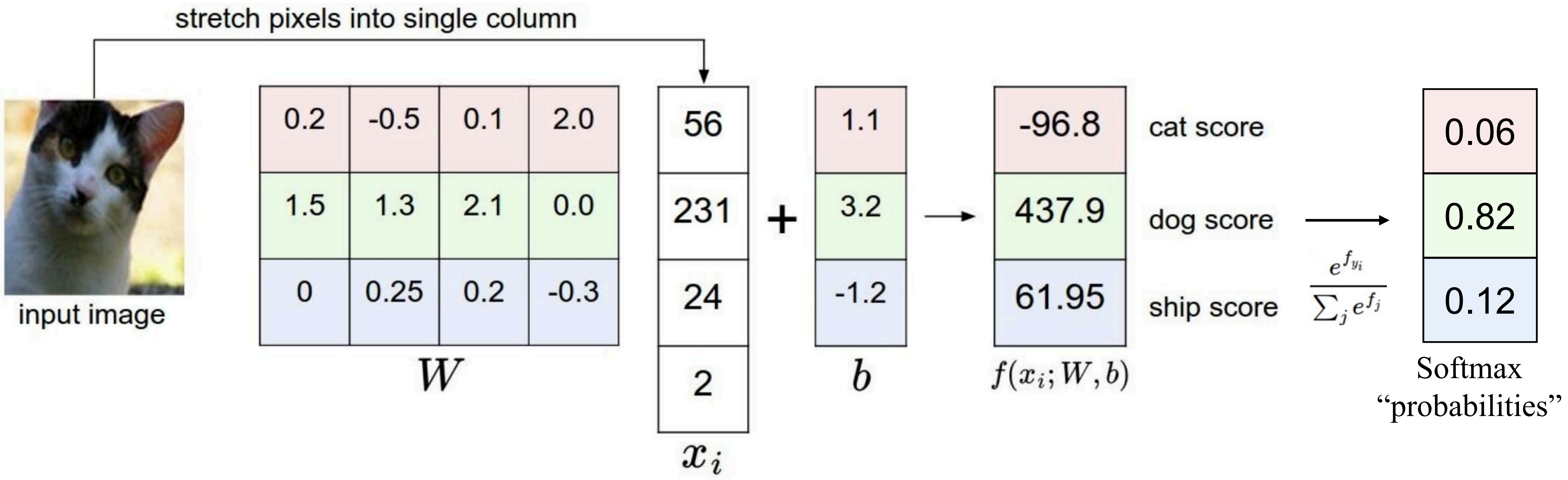
Squashes values into *probabilities* ranging from 0 to 1 $P(y_i \mid x_i; W)$

Example with three classes:

$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

Softmax classifier

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



Cross-entropy loss

$$f(x_i, W) = Wx_i$$ (score function)

Cross-entropy loss

$f(x_i, W) = Wx_i$ (score function)

f_{y_i} : score of correct class

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

We call L_i *cross-entropy loss*

Cross-entropy loss

$f(x_i, W) = Wx_i$ (score function)

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

$$\boxed{P(y_i \mid x_i; W)}$$

$$L_i = -f_{y_i} + \log \sum_j e^{f_j}$$

We call L_i *cross-entropy loss*

i.e. we're minimizing the negative log likelihood.

Losses

- Cross-entropy loss is just one possible loss function
  - One nice property is that it reinterprets scores as probabilities, which have a natural meaning
- SVM (max-margin) loss functions also used to be popular
  - But currently, cross-entropy is the most common classification loss

Summary

- Have score function and loss function
  - Currently, score function is based on linear classifier
  - Next, will generalize to convolutional neural networks
- Find W and b to minimize loss

$$L = \underbrace{\frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)}_{\text{Average of cross-entropy loss over all training examples}} + \lambda \sum_k \sum_l W_{k,l}^2$$

Regularization term (Won't talk about this now)