EECS 442 Discussion 4

Intro to Machine Learning

Discussion Agenda

- L2 Distance
- k-NN
- Train/Test Split
- Histogram of Oriented Gradients (HOG)
- Linear Classification
- Loss Functions
  - Softmax
- Gradient Descent
  - Batch Gradient Descent
  - Stochastic Gradient Descent (SGD)

Image Classification

Input Image

Output Class

dog
cat
mountain
ship
.
.

dog
cat
mountain
ship
.
.

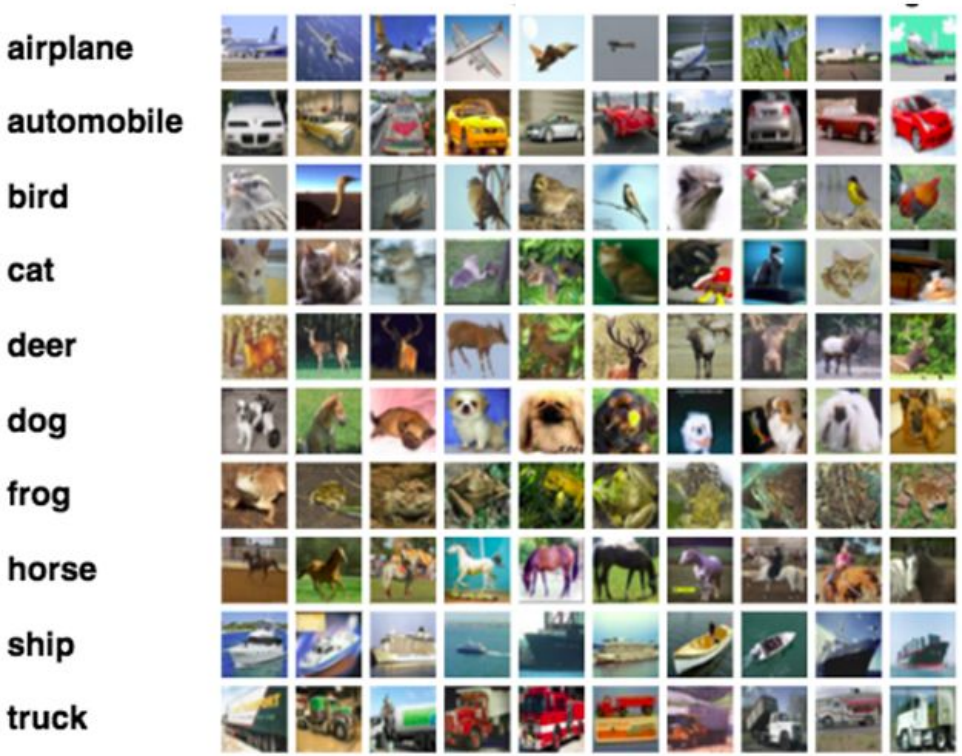
Example dataset: CIFAR10

Images: 32x32x3 (RGB)

10 classes

Training set: 5k images per class

Test set: 1k images per class



Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009

L2 Distance

How do we calculate the distance between two images?

$$\left\| \begin{matrix} 25 & 243 \\ 13 & 9 \end{matrix} \right. - \left. \begin{matrix} 10 & 190 \\ 20 & 52 \end{matrix} \right\|_2$$

Square of difference

225	2809
49	43

Add and take square root

$= \sqrt{225 + 2809 + 49 + 43}$

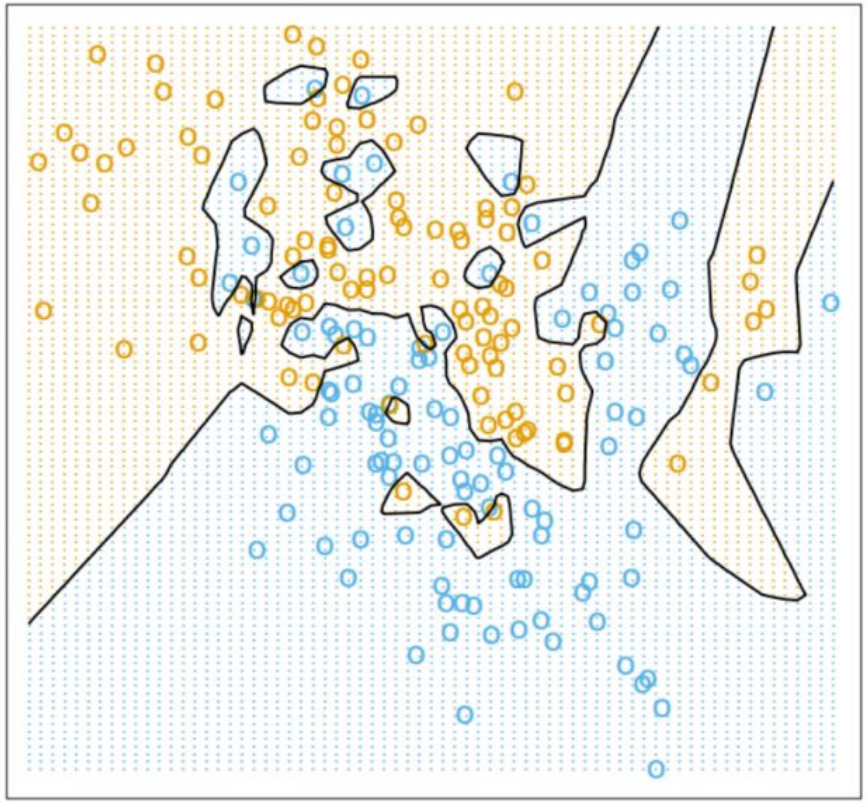
$= 55.91$

Called the L2 Distance between two images.

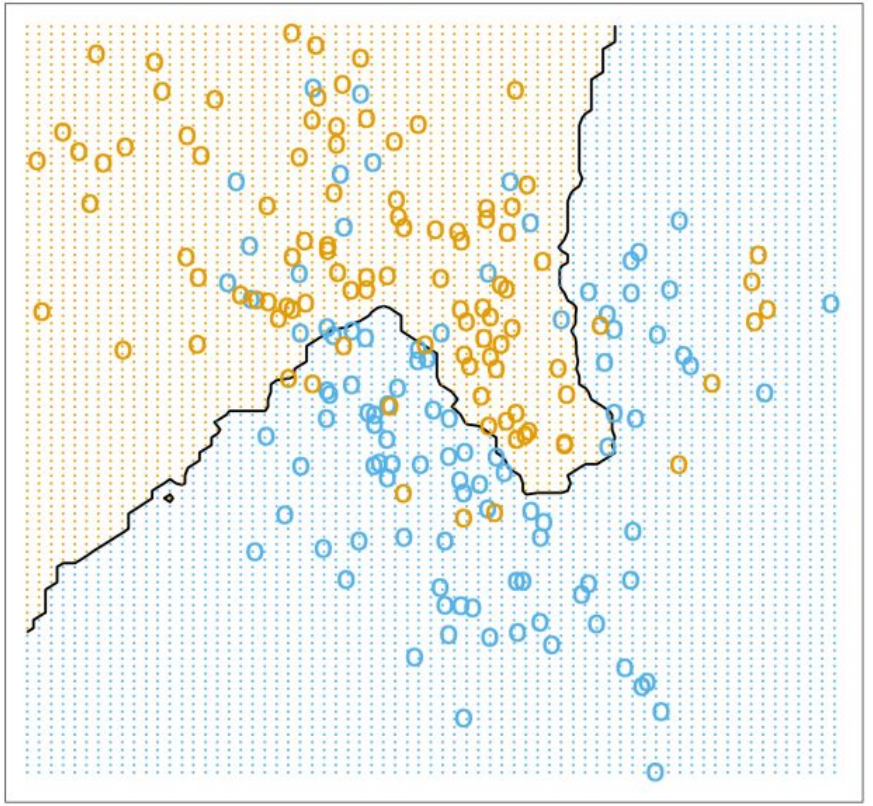
Hint for broadcasting: ||x - y||^2 = ||x||^2 + ||y||^2 - 2x^T y

k-NN

- For each point *x*
- Find the k points with the shortest L2 distance from *x*
- Those k points are called the k-Nearest Neighbors to the point *x*



$k = 1$

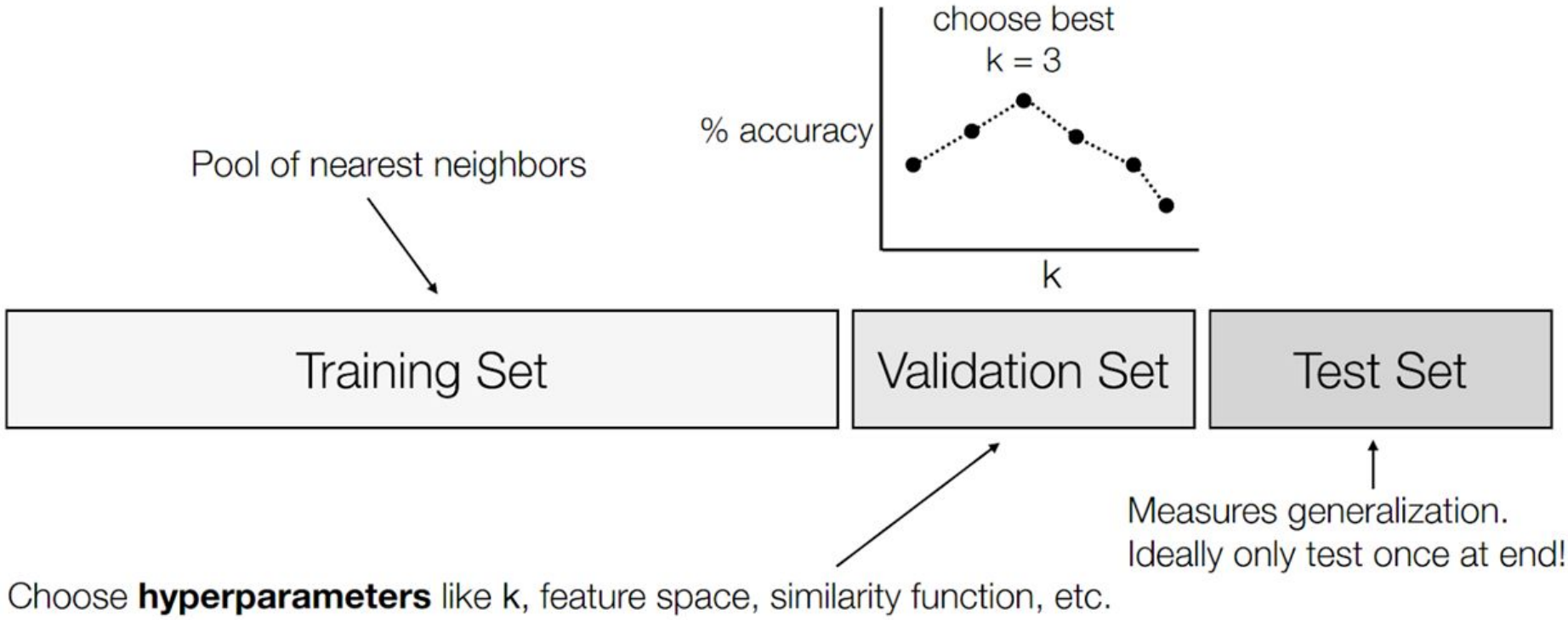


$k = 15$

Choosing *k*

- Machine learning: generalize in the wild by training on a dataset that is representative of the samples to which we want to apply our system
- How do we get a measure of how good our system will be?
  - Calculate its accuracy on a “test set” which is done only once at the end of training
- **How do we tune our hyperparameters (like *k*) then?**

Train/Test Split



Histogram of Oriented Gradients (HOG)

- Uniquely describe the features of images
- Edge and gradient based descriptors

1. Compute the orientations of the gradients
2. Create a histogram of edge orientation, with votes weighted by the gradient magnitudes
3. Perform block normalization across the histogram

Input image

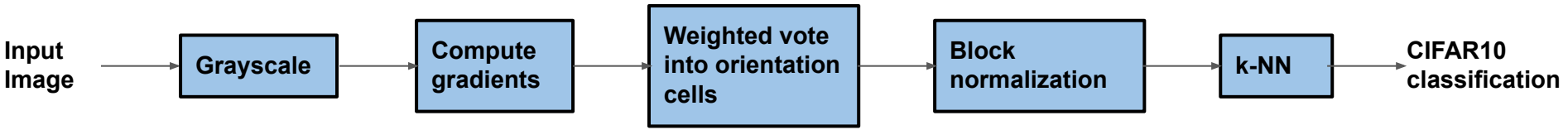
Histogram of Oriented Gradients

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection.

HOG Classification Workflow

Original HOG

Our simplified HOG



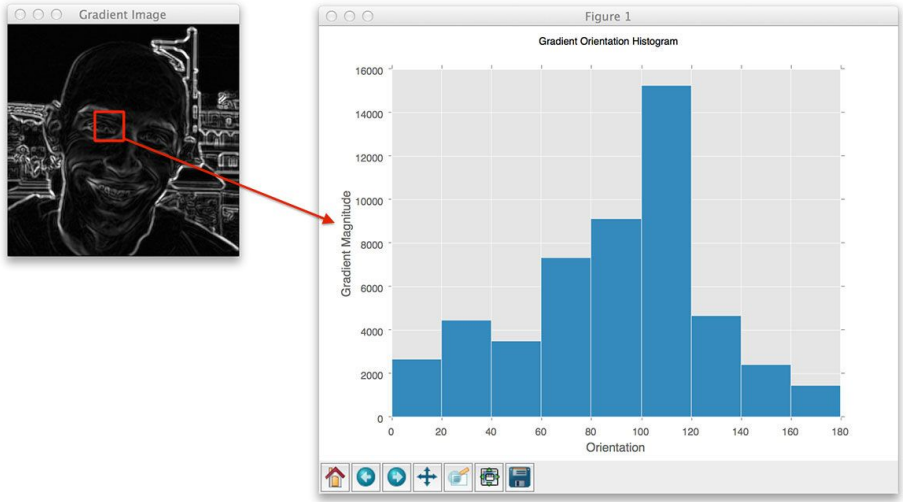
N. Dalal and B. Triggs. Histograms of oriented gradients for human detection.

Computing Orientations of the Image Gradients

- Compute gradients of an image along the horizontal and vertical directions
  - Convolve with dx = [1 -1] and dy = $[1\ -1]^T$ from ps1
  - or convolve with a Sobel filter
- Compute magnitude of each gradient
  - magnitude = sqrt(Gx^2 + Gy^2)
- Compute orientation/angle of each gradient
  - orientation = $\tan^{-1}(Gy, Gx)$
  - Convert angles to degrees in range [0, 180 deg]
    - Why [0, 180 deg] instead of [0, 360 deg]?

Create Histogram from Orientations and Magnitudes

- Iterate through each pixel in every cell
- Calculate vote for a bin based on the orientation of the gradient element centred on it (with linear interpolation) vote = magnitude * weight
- Weight vote based on magnitude and place into bins



N. Dalal and B. Triggs. Histograms of oriented gradients for human detection.

[Histogram of Oriented Gradients (and car logo recognition) – PyImageSearch]

Linear Interpolation

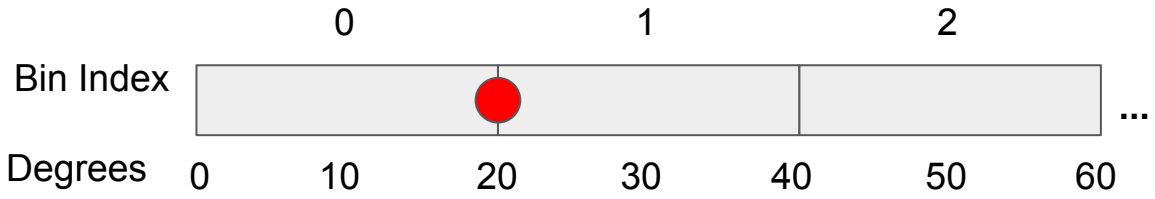
- Weigh a magnitude by how far away it is from the center of a bin
- weight = dist_from_bin_center / step_size

i.e. weight for 20 degrees (between bin 0 and 1)

lower_bin_index = floor(angle / step_size - 0.5) = 0

weight for bin 0: (angle - (lower_index + 0.5) * step_size) / step_size = 0.5

weight for bin 1: ((lower_index + 1.5) * step_size - angle) / step_size = 0.5



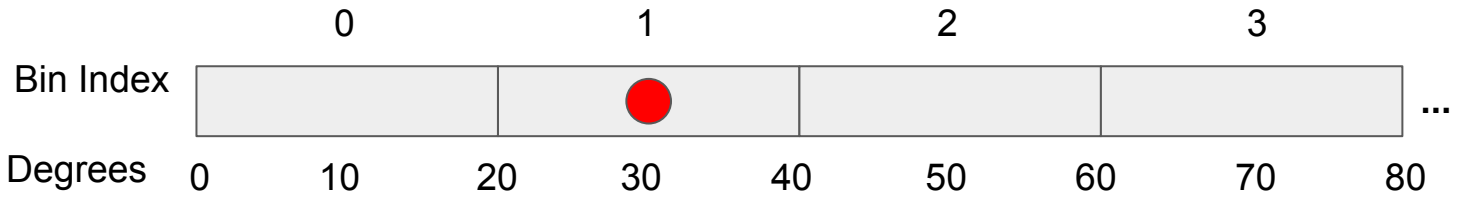
More Examples of Linear Interpolation

Weight for 30 degrees (exactly at center of bin 1)

lower_bin_index = floor(angle / step_size - 0.5) = 1

weight for bin 1: (angle - (lower_index + 0.5) * step_size) / step_size = 0

weight for bin 2: ((lower_index + 1.5) * step_size - angle) / step_size = 1



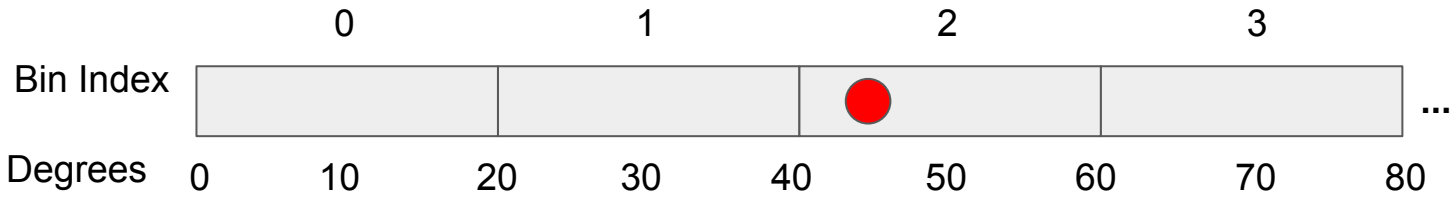
More Examples of Linear Interpolation

Weight for 45 degrees (between bins 1 and 2)

lower_bin_index = floor(angle / step_size - 0.5) = 1

weight for bin 1: (angle - (lower_index + 0.5) * step_size) / step_size = 0.75

weight for bin 2: ((lower_index + 1.5) * step_size - angle) / step_size = 0.25



Block Normalization

- Each block consists of nxn cells
- Divide everything in a block by the square root of the sum of the squares
  - Plus an epsilon term to avoid division by zero
- Keep regions of image at the same intensity/brightness

$$L2\text{-}norm,\ \mathbf{v} \rightarrow \mathbf{v}/\sqrt{||\mathbf{v}||_2^2 + \epsilon^2}$$

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection.

Linear classifiers

x : (32x32x3)

Flatten image

x : (3072,)

Linear Classifier
f(x) = Wx + b

W: (10, 3072)
b: (10,)

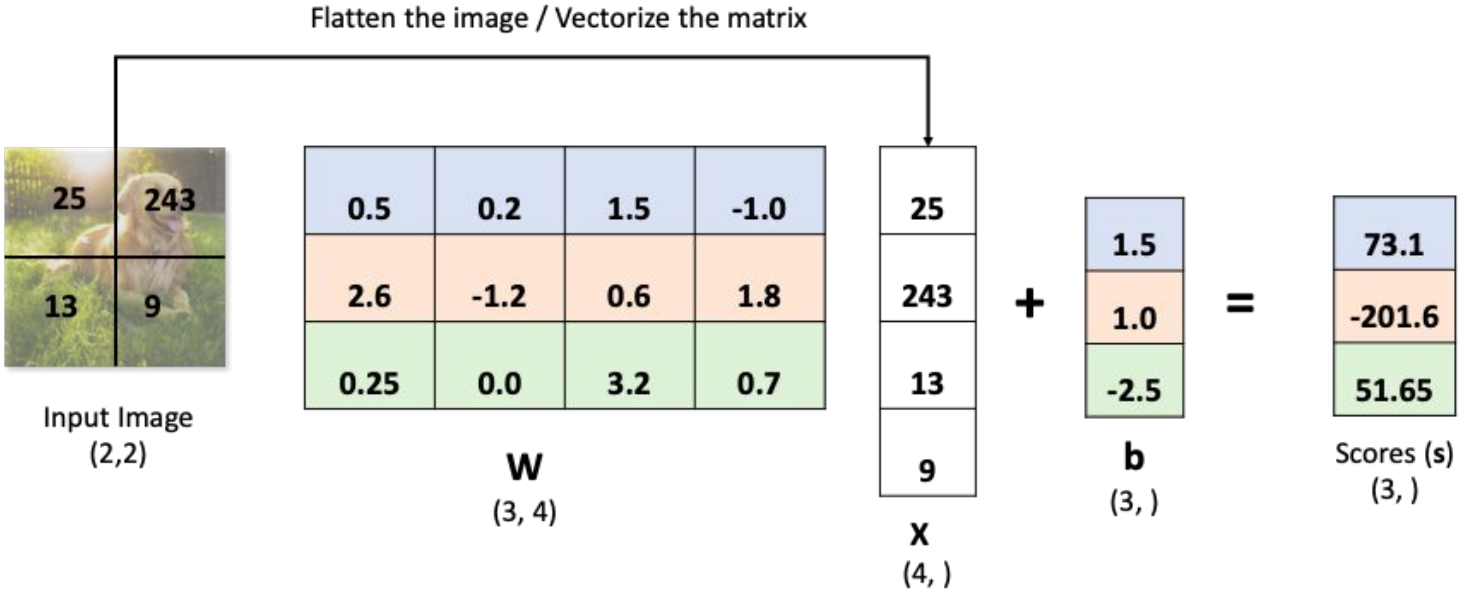
250.75	dog
30.5	cat
67.325	mountain
-20.25	ship
.	.
.	.

10 possible classes in total

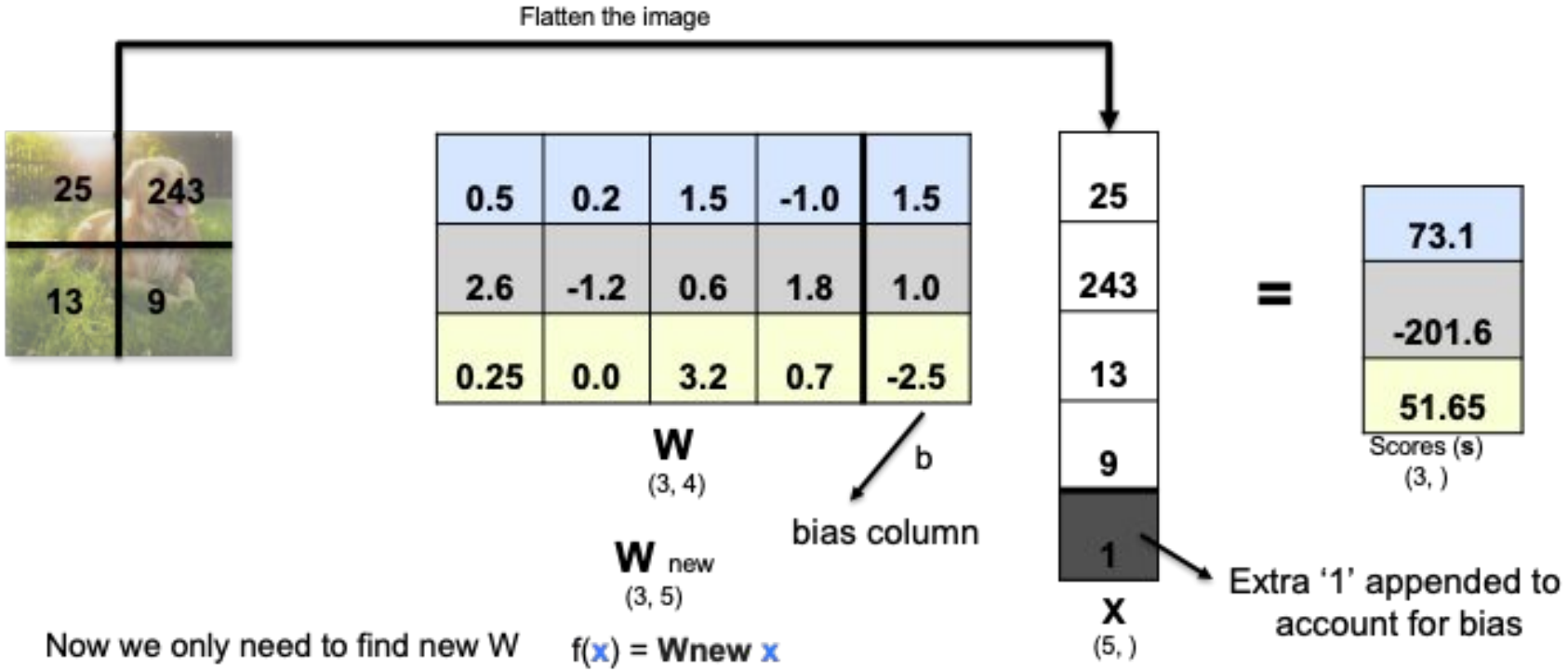
img = img.reshape(img.shape[0], -1)

Linear classifiers

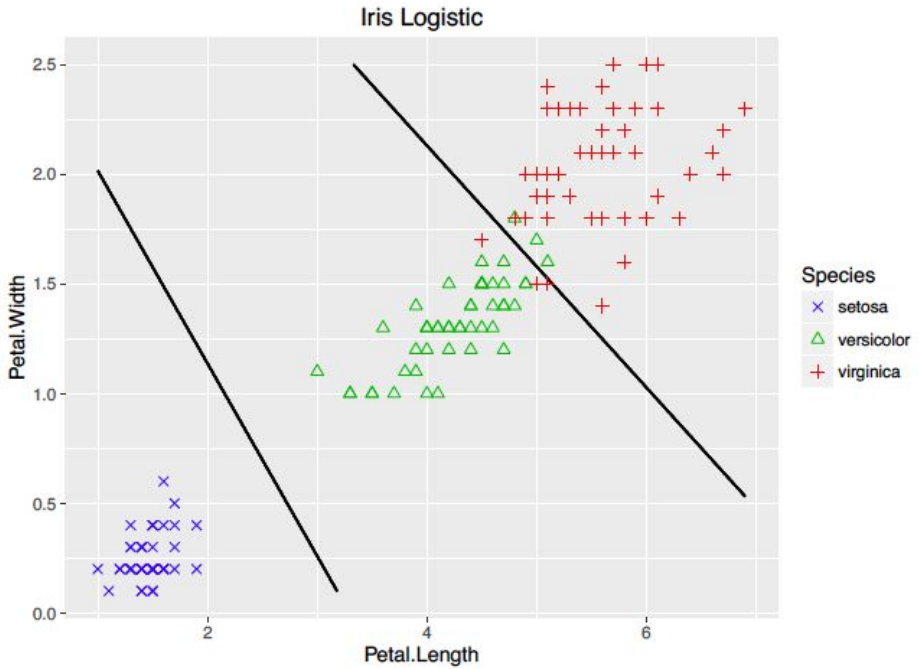
Example 2x2 image, 3 classes (dog, cat, mountain)



Bias Trick


Flatten the image
25
243
13
9
0.5
0.2
1.5
-1.0
1.5
2.6
-1.2
0.6
1.8
1.0
0.25
0.0
3.2
0.7
-2.5
W
(3, 4)
b
bias column
W new
(3, 5)
25
243
13
9
1
X
(5,)
Extra '1' appended to account for bias
=
73.1
-201.6
51.65
Scores (s)
(3,)
Now we only need to find new W
f(x) = Wnew x

Linear classifier boundary



Loss Function

Computes how much prediction from current model deviates from target

Cross entropy

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

← easy to optimize, good approximation

Adapted from: Isola, Torralba, Freeman

Softmax loss / Multinomial logistic loss

S: score, y: one of the labels

For a single image, **x** with label 'y'

$$L(s,y) = -\log \frac{e^{s_{y_i} - \max(s_j)}}{\sum_{j=1}^{C} e^{s_j - \max(s_j)}}$$

	Scores, s = Wx	exp	e^{s_y}	normalize	Probabilities
Dog	2.5		12.18		0.06
Cat	-1.3		0.27		0.0
Mountain	5.2		181.27		0.94

y = 1, 2, 3
1: Dog
2: Cat
3: Mountain

$$\frac{e^{s_y}}{\sum_{j=1}^{3} e^{s_j}}$$

In this case, class y = 1 (Dog)

$$L(s,1) = -\log 0.06 = 2.81$$

Softmax loss properties

$$L(s, y) = -\log \frac{e^{s_{y_i} - \max(s_j)}}{\sum_{j=1}^{C} e^{s_j - \max(s_j)}}$$

C: Number of classes

What is the min/max possible Value of L(s, y) ? → Min 0, Max +infinity

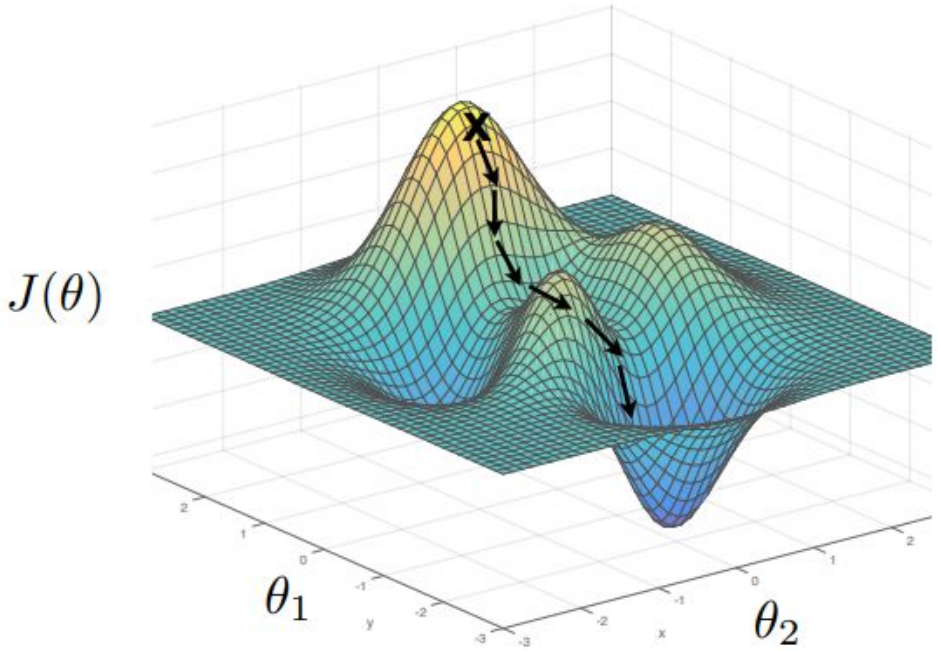
If all scores were really small and thus approximately the same, What would be the value of L(s, y)? → log(C)

Gradient Descent

- Want to optimize some objective function **J**

$$\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}_{J(\theta)}$$

Gradient Descent



Gradient Descent

One iteration of gradient descent:

$$\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$$

learning rate

Batch Gradient Descent

Loss function:

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

Its gradient is the **sum of gradients** for each example:

$$\nabla J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla L(x_i, y_i, \theta)$$

Requires iterating over every training example each gradient step!

Can we speed this up?

Stochastic Gradient Descent

- Sample a point instead of looping over all training examples

$$\nabla J(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla L(x_i, y_i, \theta)$$

where *B* is a minibatch: a random subset of examples