Lecture 2:

Image Classification with Linear Classifiers

Fei-Fei Li, Yunzhu Li, Ruohan Gao
2023

Image Classification

A Core Task in Computer Vision

Today:

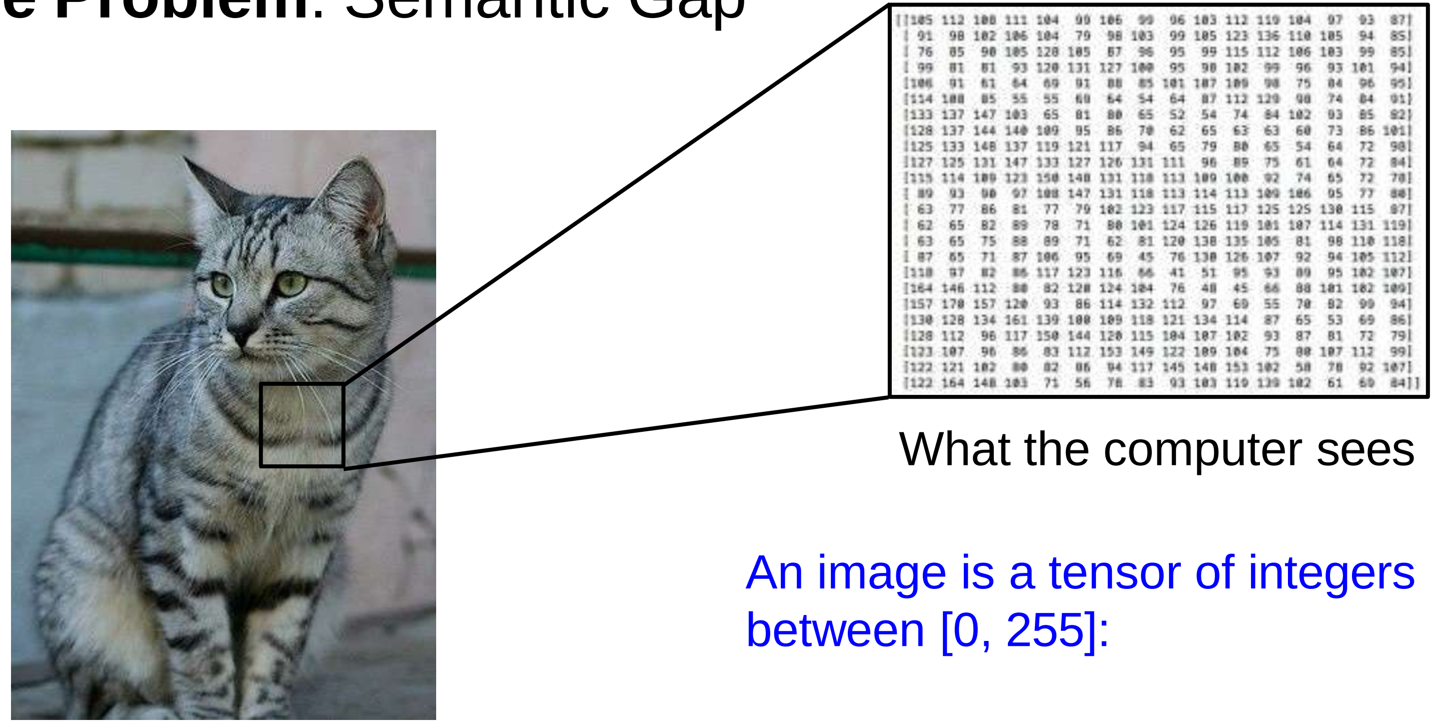
- The image classification task
- Two basic data-driven approaches to image classification
    - K-nearest neighbor and linear classifier

Image Classification: A core task in Computer Vision

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

→ cat

The Problem: Semantic Gap

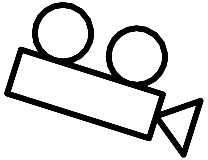


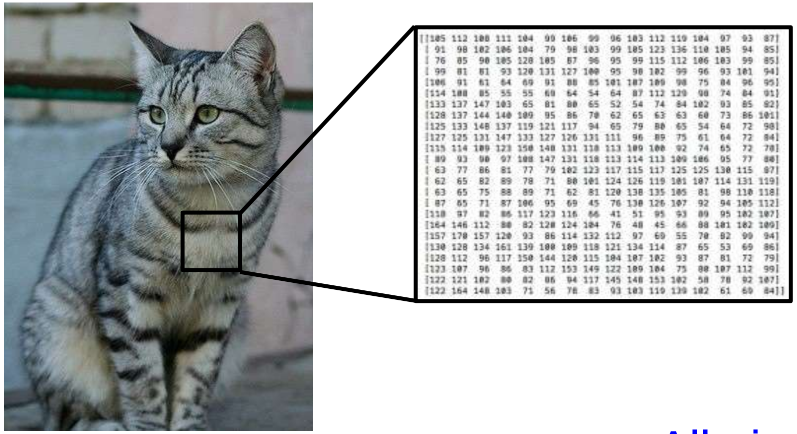
What the computer sees

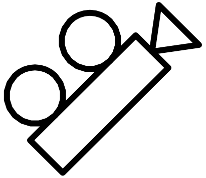
An image is a tensor of integers between [0, 255]:

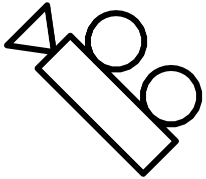
e.g. 800 x 600 x 3
(3 channels RGB)

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

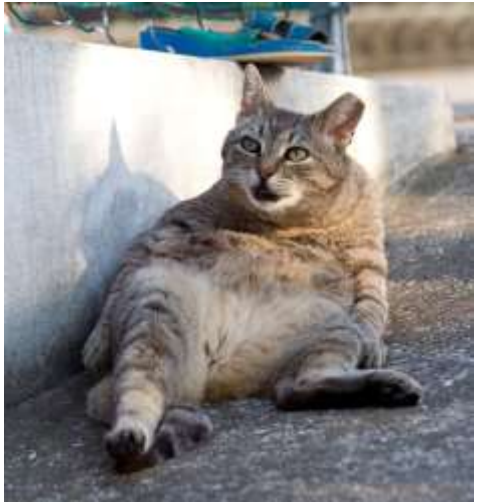
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

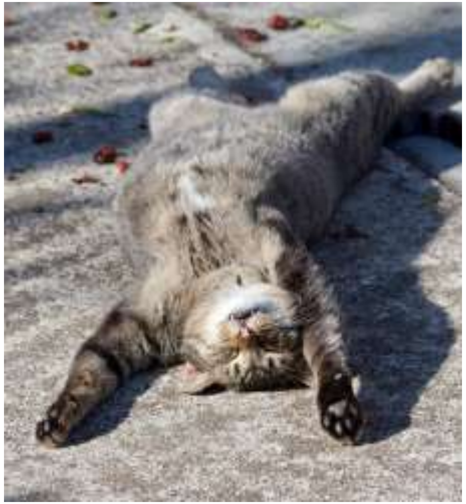
Challenges: Background Clutter

Challenges: Occlusion

Challenges: Deformation

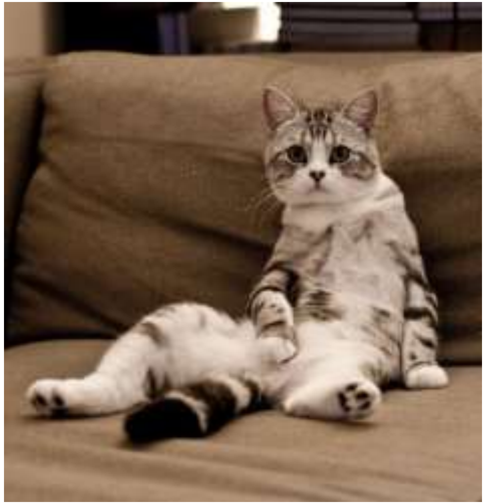


This image by Umberto Salvagnin is licensed under CC-BY 2.0



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

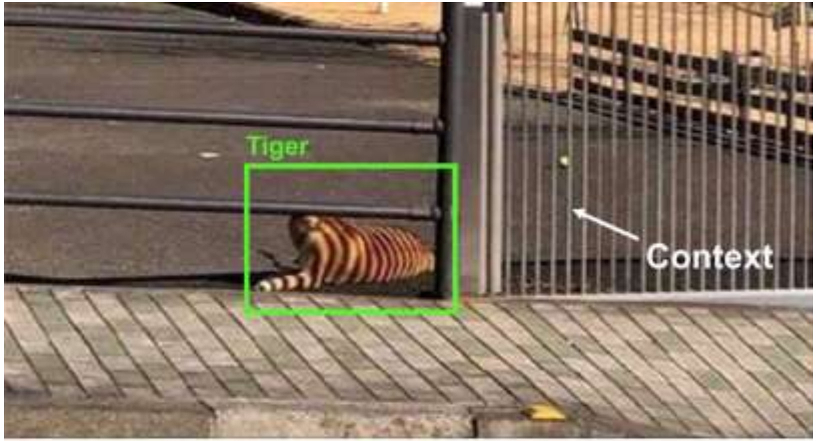
This image by sare bear is licensed under CC-BY 2.0



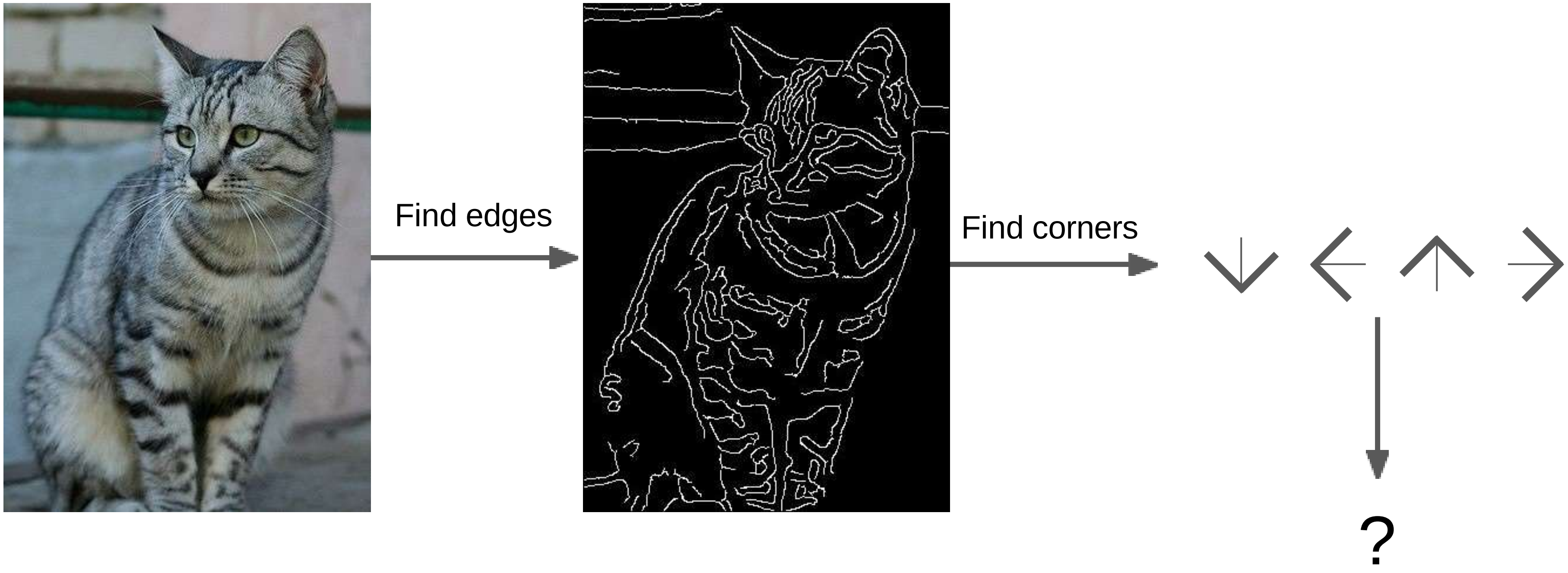
This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Intraclass variation

Challenges: Context



Attempts have been made



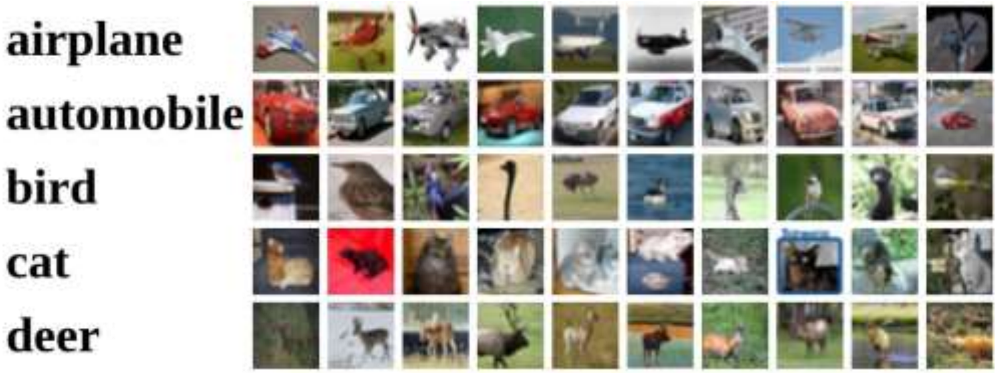
Machine Learning: Data-Driven Approach

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

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

```python
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**

deer bird plane cat car

Training data with labels

?

query data

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

Distance Metric to compare images

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

$$\left| \underbrace{\begin{array}{|c|c|c|c|} \hline 56 & 32 & 10 & 18 \\ \hline 90 & 23 & 128 & 133 \\ \hline 24 & 26 & 178 & 200 \\ \hline 2 & 0 & 255 & 220 \\ \hline \end{array}}_{\text{test image}} - \underbrace{\begin{array}{|c|c|c|c|} \hline 10 & 20 & 24 & 17 \\ \hline 8 & 10 & 89 & 100 \\ \hline 12 & 16 & 178 & 170 \\ \hline 4 & 32 & 233 & 112 \\ \hline \end{array}}_{\text{training image}} \right| = \underbrace{\begin{array}{|c|c|c|c|} \hline 46 & 12 & 14 & 1 \\ \hline 82 & 13 & 39 & 33 \\ \hline 12 & 10 & 0 & 30 \\ \hline 2 & 32 & 22 & 108 \\ \hline \end{array}}_{\text{pixel-wise absolute value differences}} \xrightarrow{\text{add}} 456$$

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
```

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
```

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
```

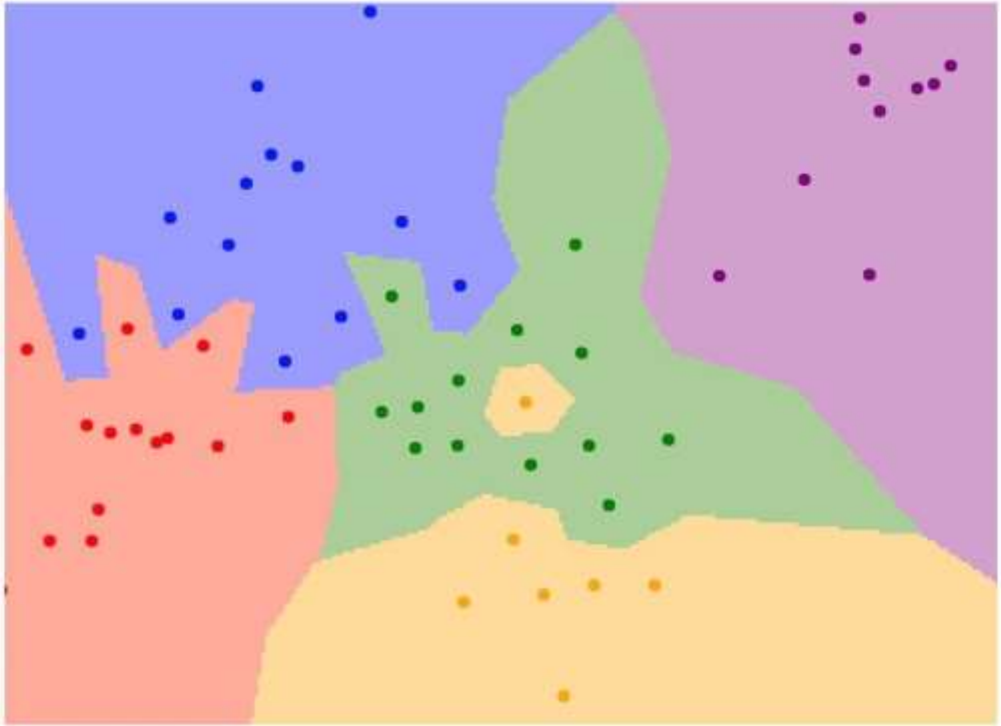
Nearest Neighbor classifier

Many methods exist for fast / approximate nearest neighbor

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

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

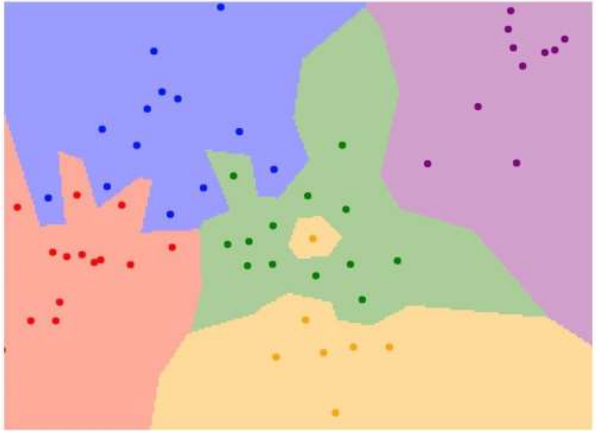
What does this look like?



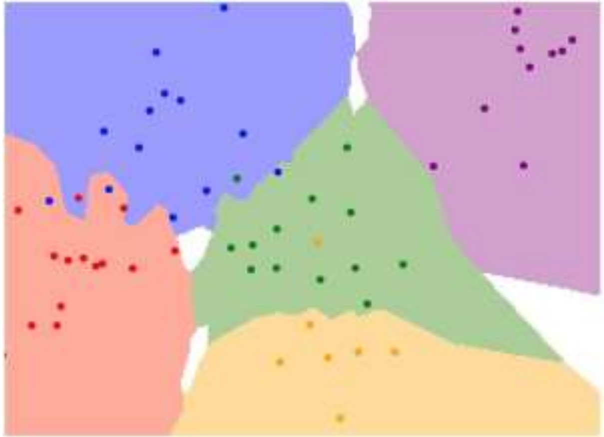
1-nearest neighbor

K-Nearest Neighbors

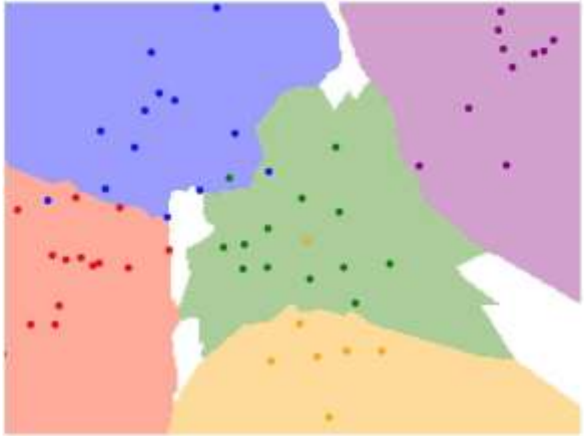
Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



K = 1



K = 3

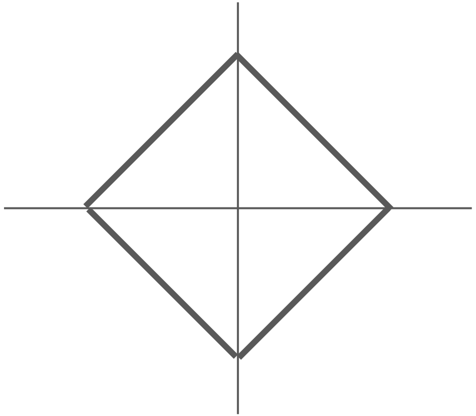


K = 5

K-Nearest Neighbors: Distance Metric

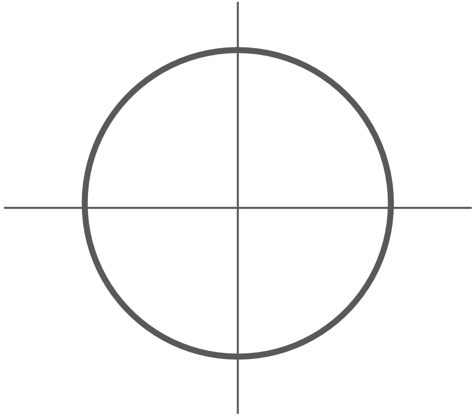
L1 (Manhattan) distance

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



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

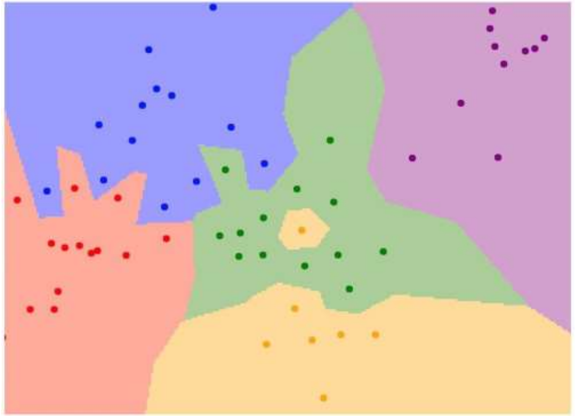
L1 (Manhattan) distance

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

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

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

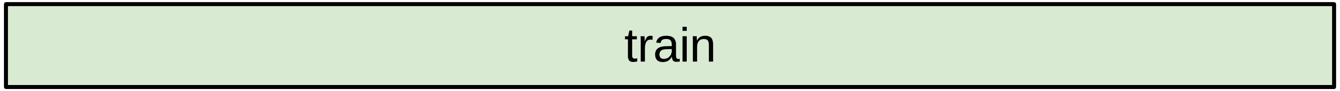
Idea #1: Choose hyperparameters that work best on the **training data**

train

Setting Hyperparameters

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

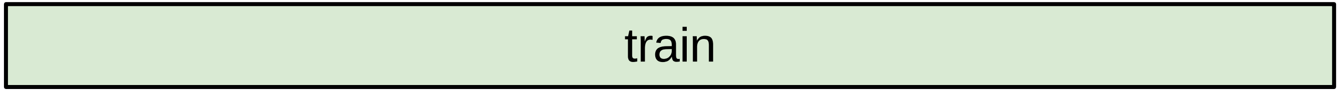
BAD: K = 1 always works perfectly on training data



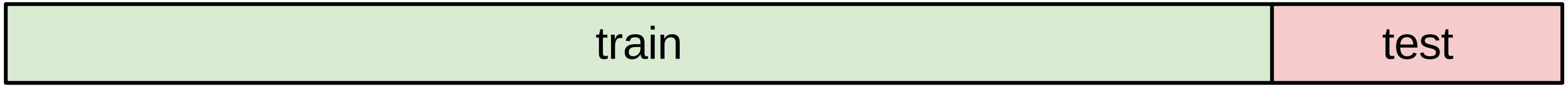
Setting Hyperparameters

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

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

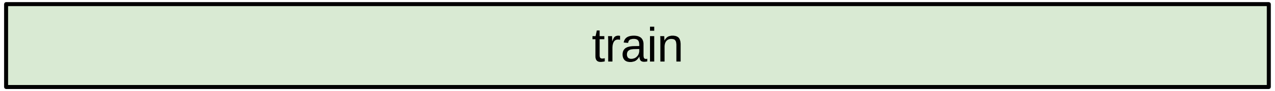
train	test

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

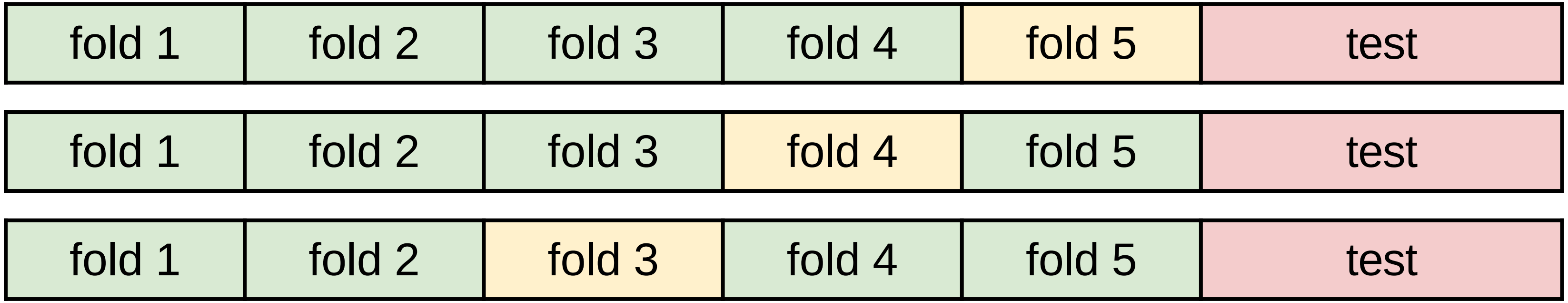
Better!

train	validation	test

Setting Hyperparameters



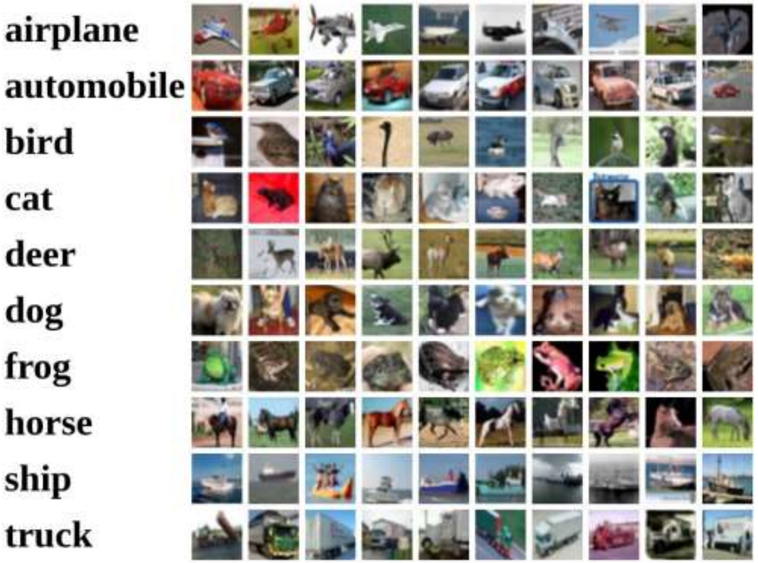
Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results



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

Example Dataset: **CIFAR10**

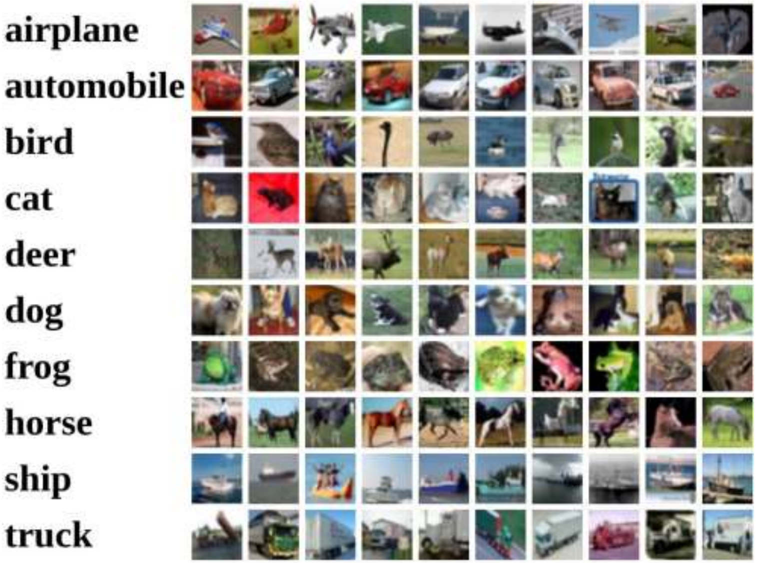
10 classes
50,000 training images
10,000 testing images



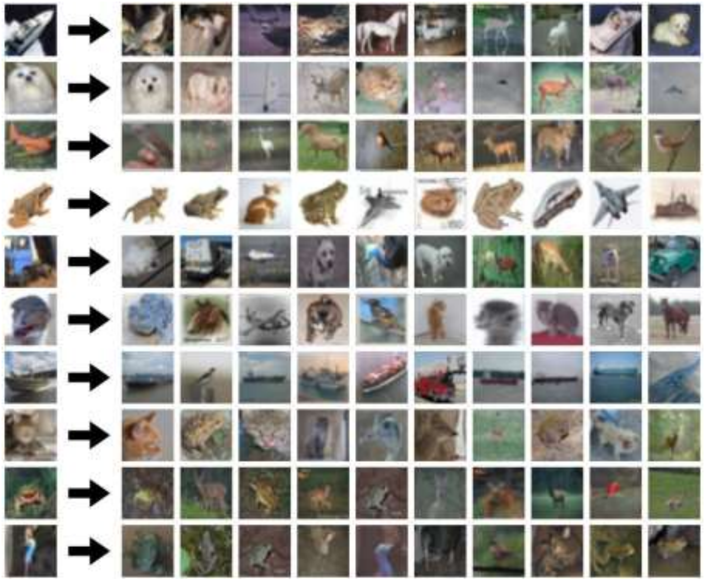
Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Example Dataset: **CIFAR10**

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

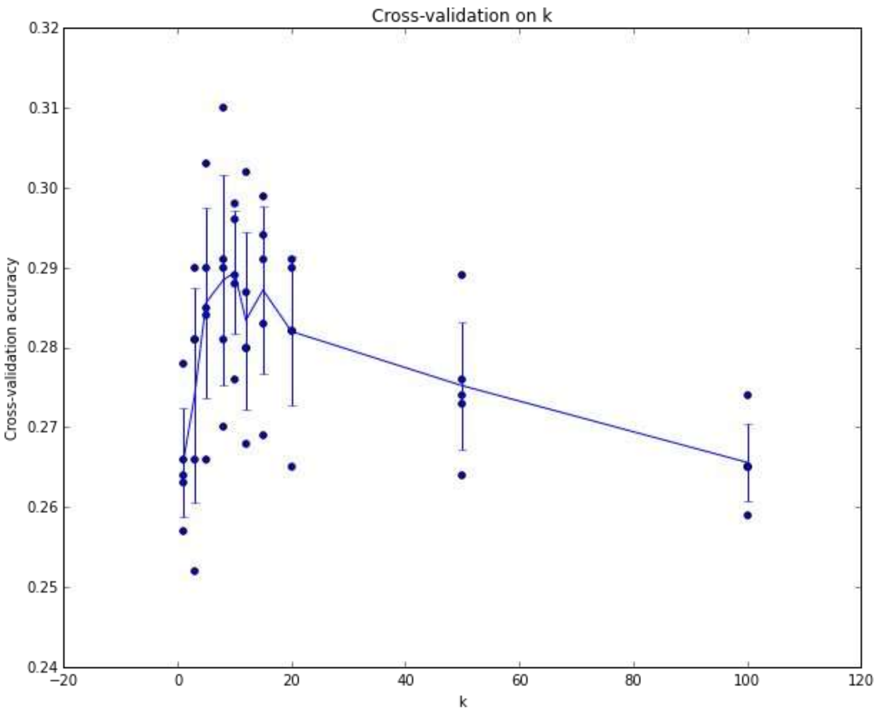


Test images and nearest neighbors



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 ~= 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

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)

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

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**)

W

parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier

3072x1

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

10x1 10x3072

Image

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

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

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

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

10x1

10x1 10x3072

Image

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

$\longrightarrow$ f(**x**,**W**) $\longrightarrow$ **10** numbers giving class scores

$\uparrow$

W

parameters
or weights

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)

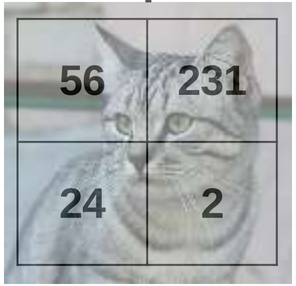
Flatten tensors into a vector

Input image

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

Algebraic Viewpoint

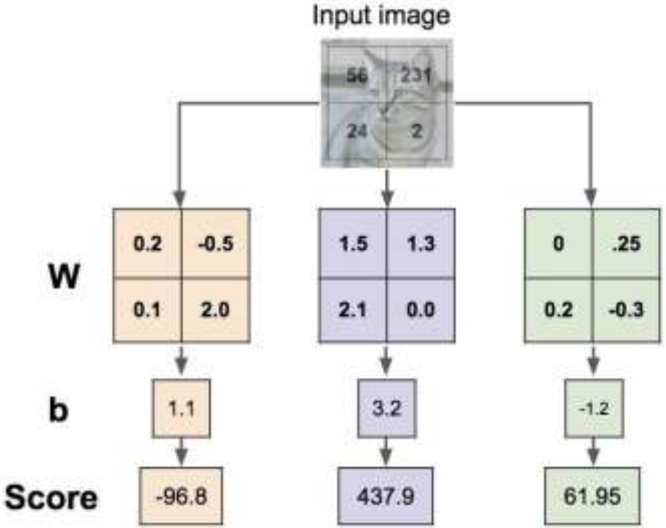
Flatten tensors into a vector



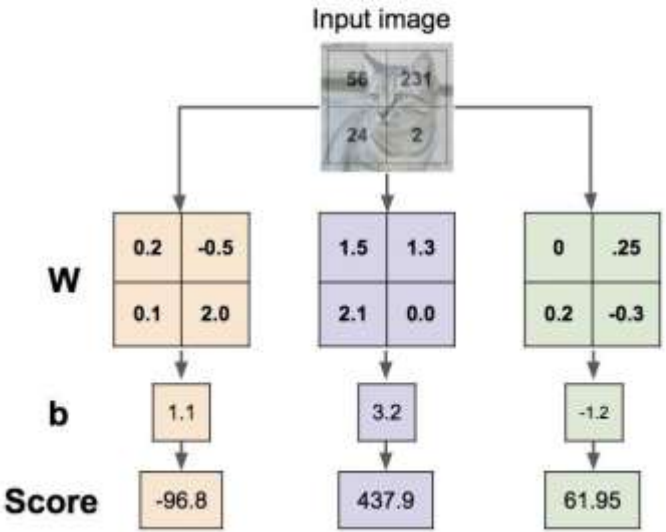
Input image

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

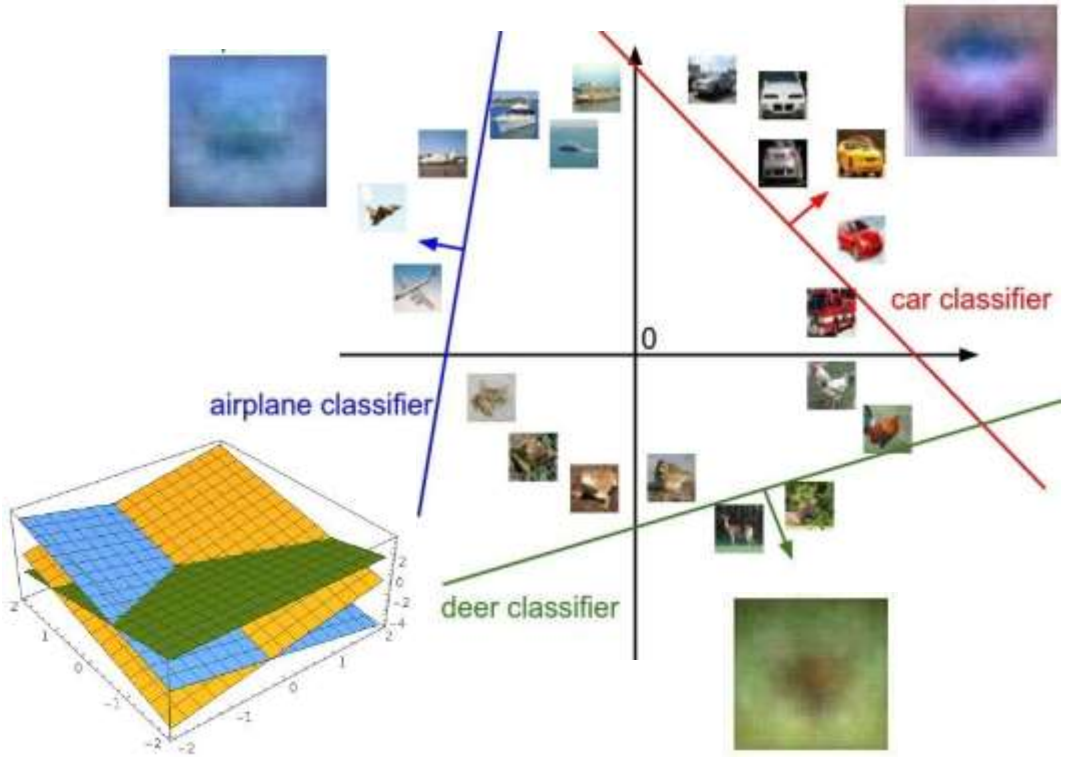
Interpreting a Linear Classifier



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

Bias trick

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

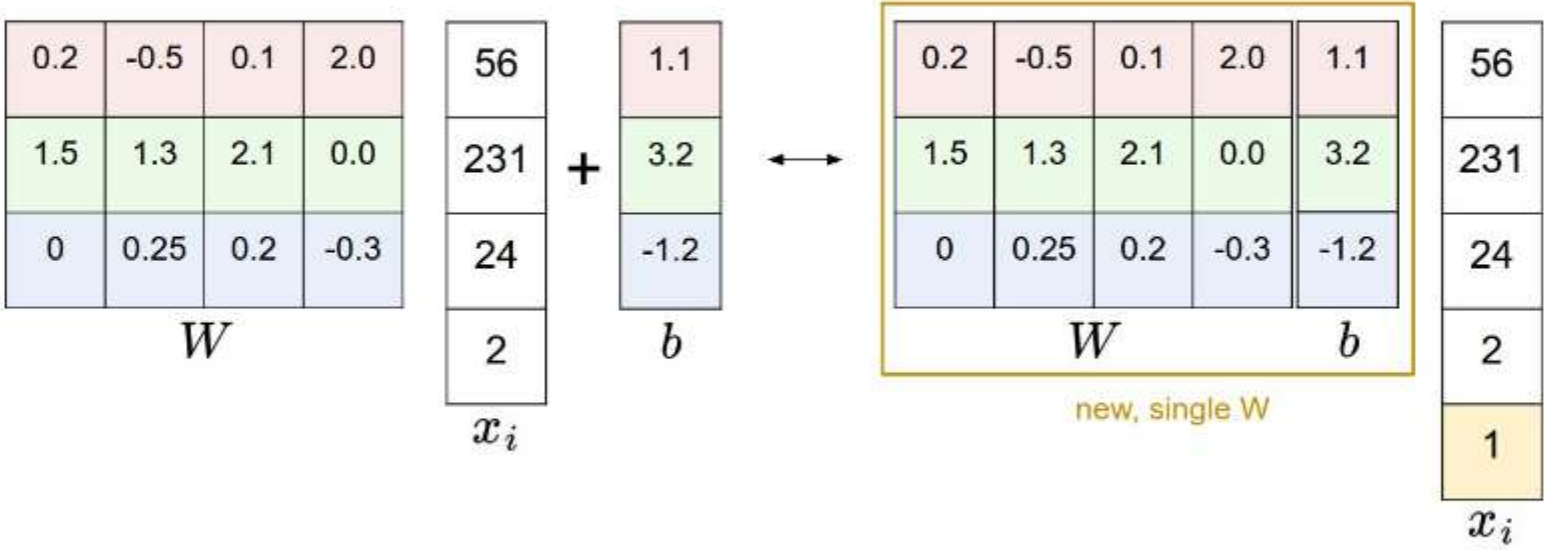
$$\downarrow$$

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

Bias trick

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

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



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

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)**

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**

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

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

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

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:

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

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

The SVM loss is set up so that the SVM “wants” the correct class for each image to a have a score higher than the incorrect classes by some fixed margin Δ.

SVM

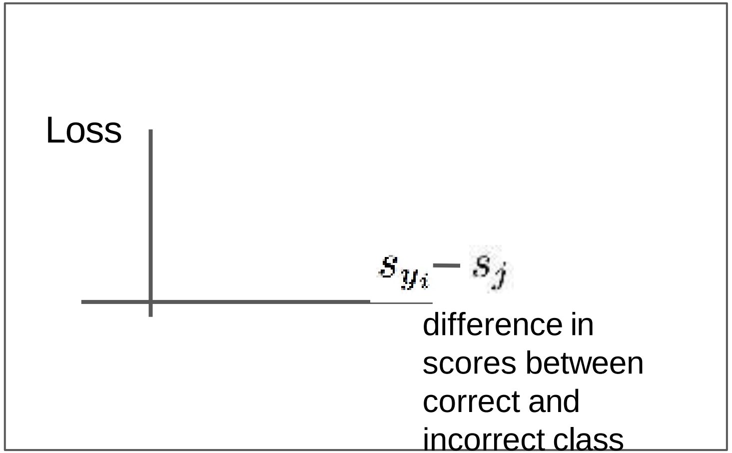
> *The loss function quantifies our unhappiness with predictions on the training set*

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



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

Interpreting Multiclass SVM loss:



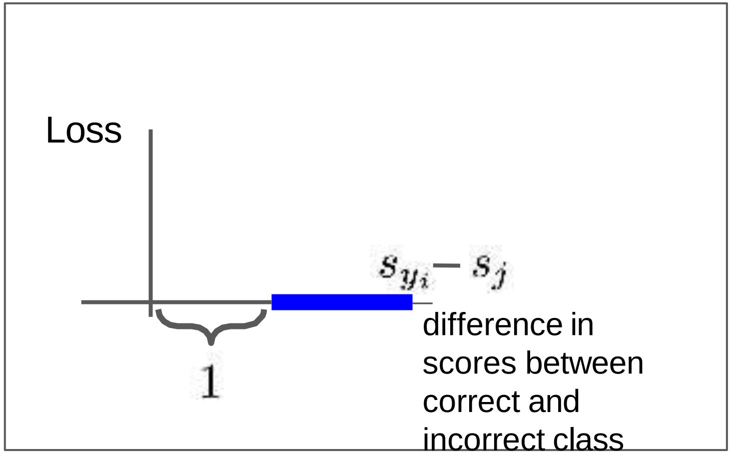
$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



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

Interpreting Multiclass SVM loss:



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

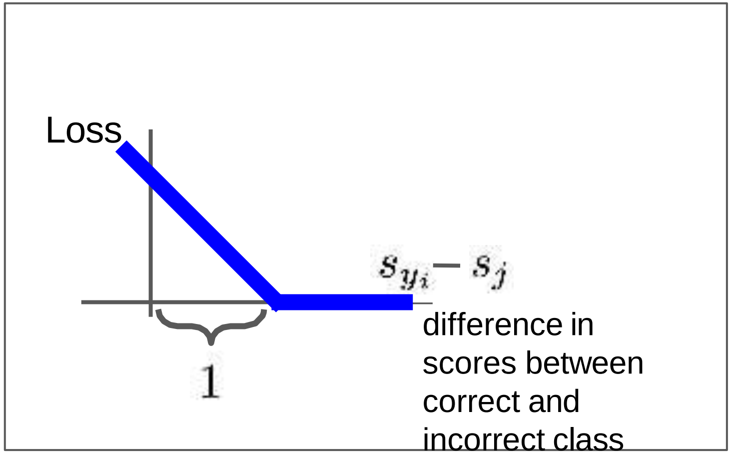
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



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

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

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:

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

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



cat	$\boxed{\textbf{3.2}}$	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

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

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

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:

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

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

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



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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

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

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



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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

Loss over full dataset is average:

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

L = (2.9 + 0 + 12.9)/3
= **5.27**

Suppose that we have three classes that receive the scores s=[13,−7,11] and that the first class is the true class (i.e. y_i=0). Also assume that Δ is 10.

Calculate Loss?

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

Suppose that we have three classes that receive the scores s=[13,−7,11] and that the first class is the true class (i.e. y_i=0). Also assume that Δ is 10.

Calculate Loss?

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

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

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

Multiclass SVM loss:

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

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 ≈ 0. What is the loss L_i, assuming N examples and C classes?

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



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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

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

Multiclass SVM loss:

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

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

the SVM loss has the form:

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

Q6: What if we used

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

What will happen if we use the max of squared scores?

- The model would focus on the most misclassified example (the one with the highest loss value) and try to correct it first.

It could be useful in specific situations, such as:

Robustness to outliers: By focusing on the most misclassified example, the model might become more robust to outliers.

Imbalanced datasets: In datasets with class imbalance, this approach could help the model focus on the minority class.

Will it be convex/non-convex loss function?

Will it be convex/non-convex loss function?

The squared function (x^2) is convex

The max function is also convex

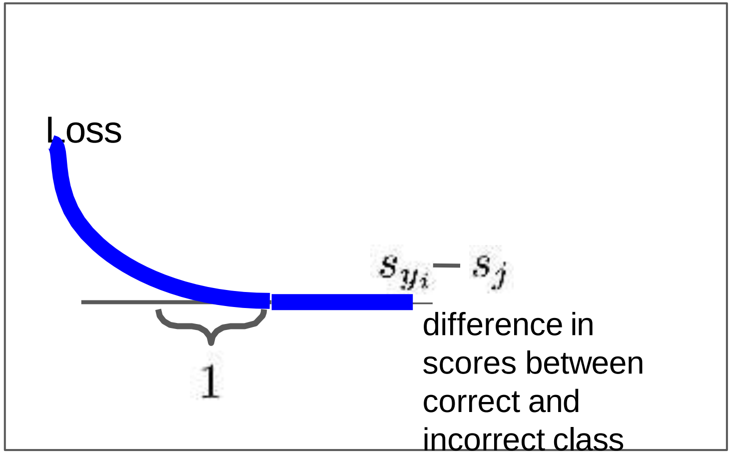
The composition of convex functions is convex,
so the resulting loss function:

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



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

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)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

First calculate scores
Then calculate the margins s_j - s_{yi} + 1
only sum j is not y_i, so when j = y_i, set to zero.
sum across all j

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**

$$\boxed{s = f(x_i; W)}$$

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

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**		**24.5**
car	5.1	$\xrightarrow{\text{exp}}$	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

		exp	Probabilities must be >= 0	normalize	Probabilities must sum to 1
cat	**3.2**		**24.5**		**0.13**
car	5.1	→	164.0	→	0.87
frog	-1.7		0.18		0.00

unnormalized log-probabilities / logits → unnormalized probabilities → 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

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

Unnormalized log-probabilities / logits

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

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

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

→ 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

Probabilities must be >= 0

Probabilities must sum to 1

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

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

→ $L_i = -\log(0.13)$
$= \mathbf{2.04}$

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	$\xrightarrow{\text{exp}}$ unnormalized probabilities	$\xrightarrow{\text{normalize}}$ probabilities	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

probabilities → compare ← 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

Probabilities must be >= 0

Probabilities must sum to 1

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

	Unnormalized log-probabilities / logits	unnormalized probabilities	probabilities	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

exp → normalize → probabilities → compare ← Correct probs

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**

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

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

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

Cross Entropy

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

Softmax Classifier (Multinomial Logistic Regression)

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

$$\boxed{s = f(x_i; W)}$$

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

Softmax Function

Maximize probability of correct class

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

Putting it all together:

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

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

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

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

Putting it all together:

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

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

cat	**3.2**
car	5.1
frog	-1.7

Maximize probability of correct class

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

Putting it all together:

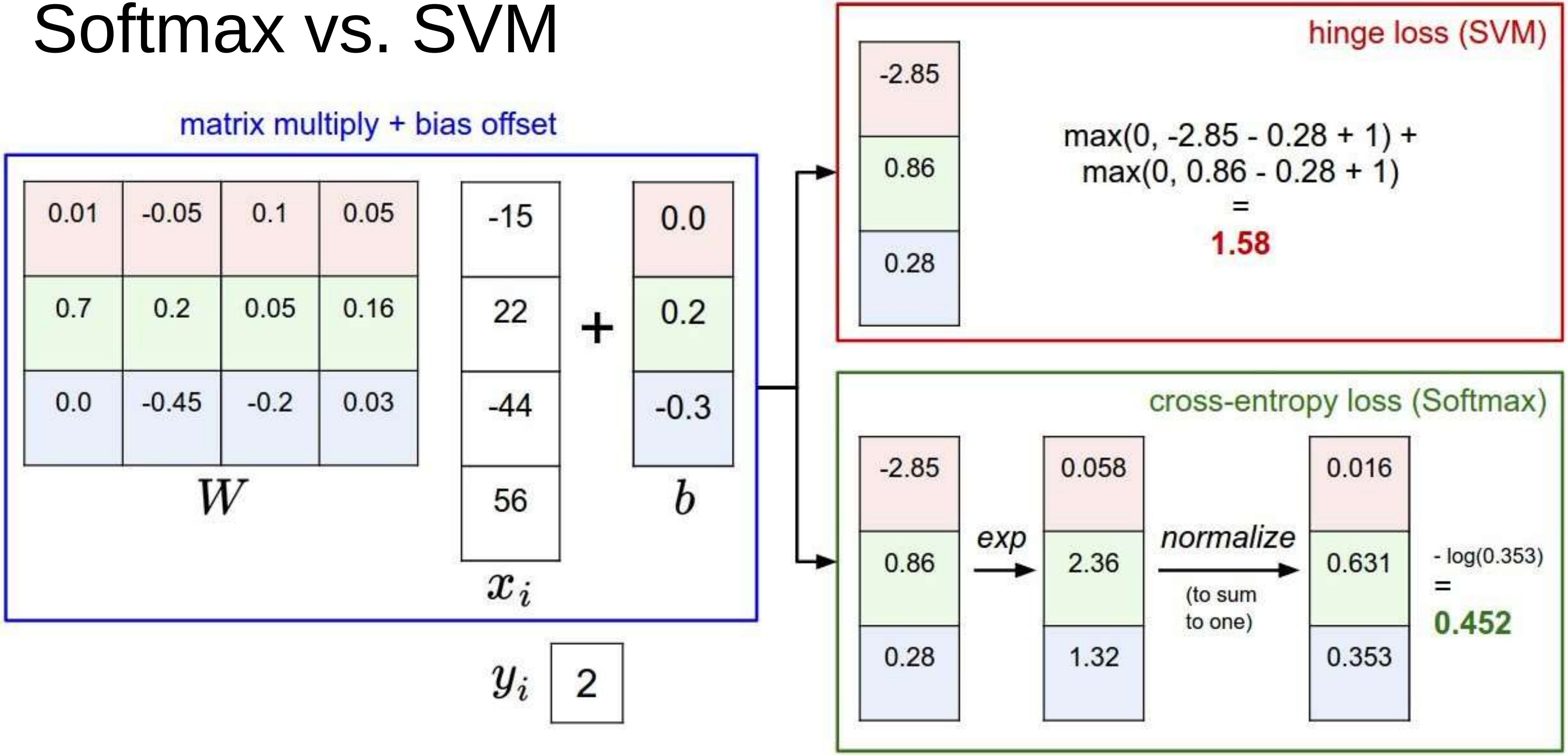
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

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) ≈ 2.3

Softmax vs. SVM



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 $\boxed{y_i = 0}$

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 $\boxed{y_i = 0}$

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