Lecture 15:
Introduction to Deep Learning

COS 429: Computer Vision

PRINCETON UNIVERSITY

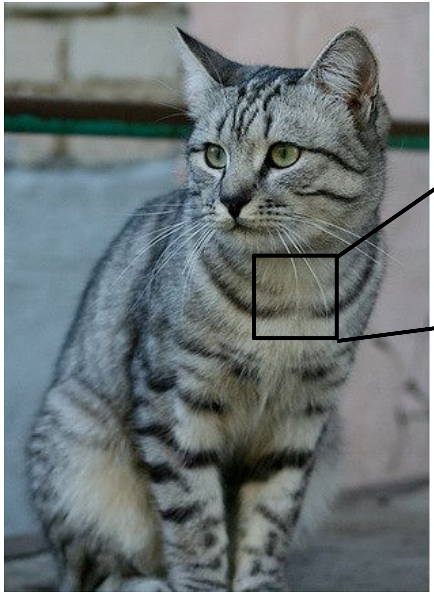
Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Image Classification: A core task in Computer Vision

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

$\longrightarrow$ cat

The Problem: Semantic Gap



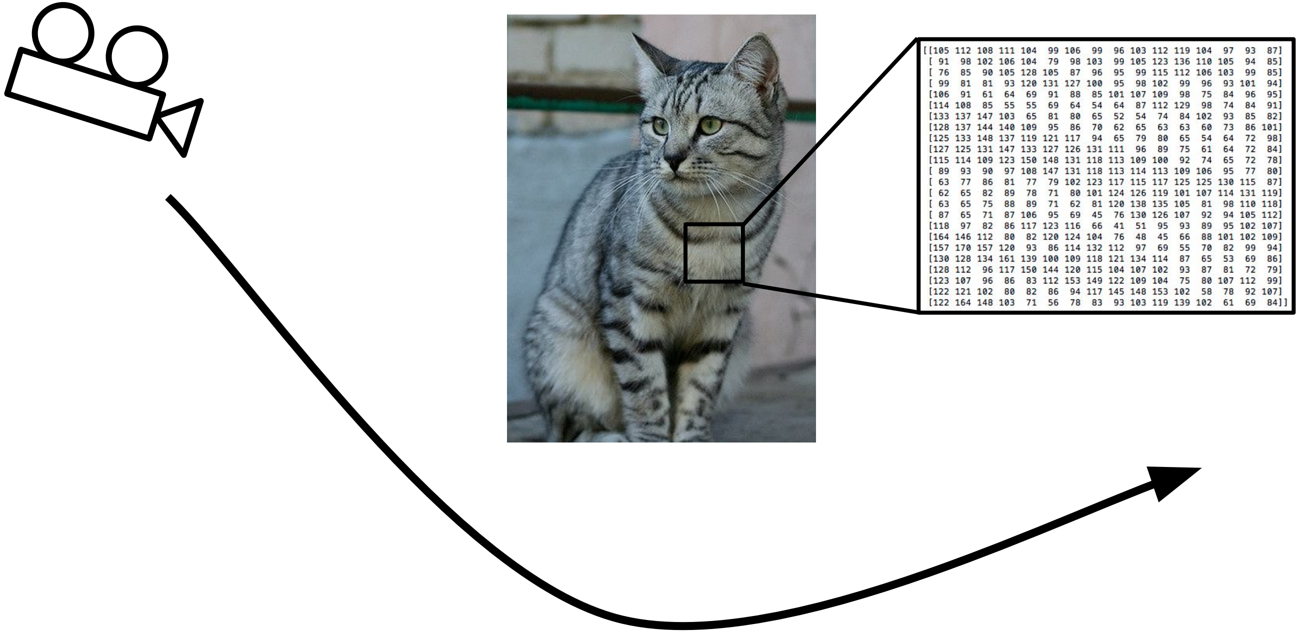

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  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  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

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

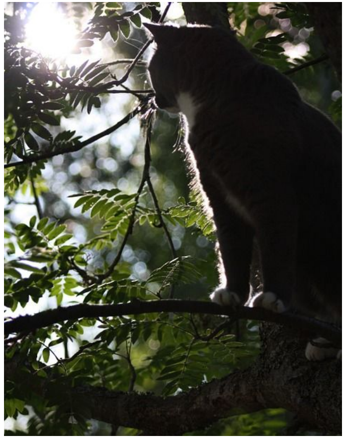
Challenges: Viewpoint variation



All pixels change when
the camera moves!

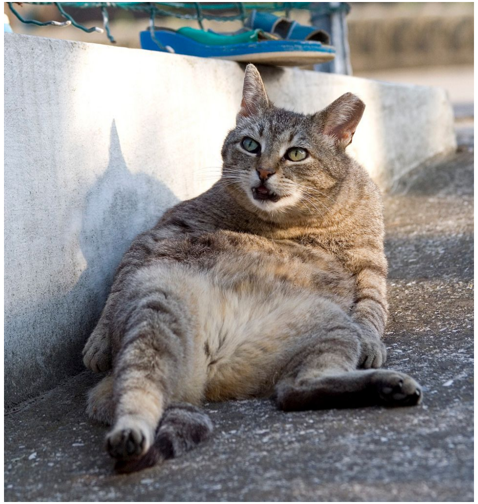
Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

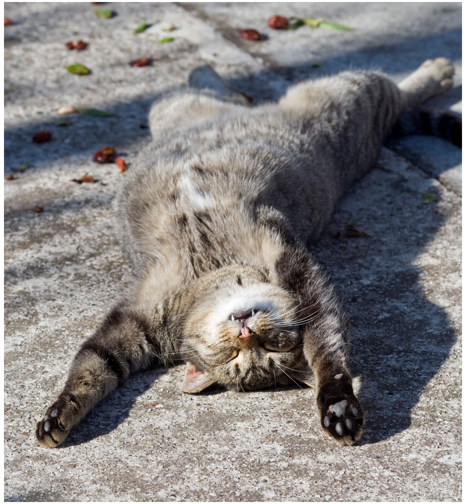
Challenges: Illumination

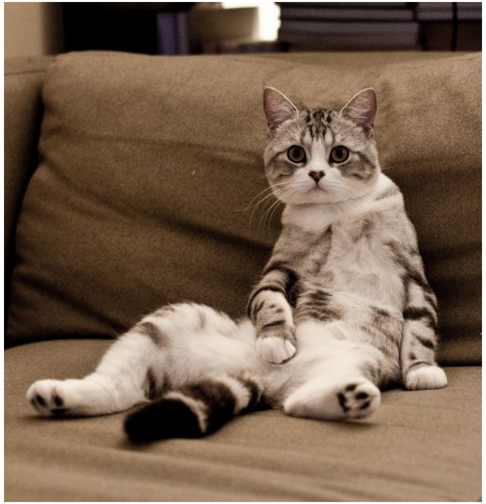


Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Challenges: Deformation







Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Challenges: Occlusion

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Challenges: Intraclass variation

This image is CC0 1.0 public domain

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

An image classifier

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

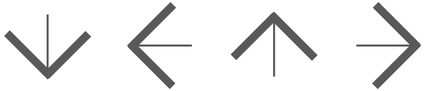
Unlike e.g. sorting a list of numbers,

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

Attempts have been made

Find edges

Find corners



?

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Data-Driven Approach

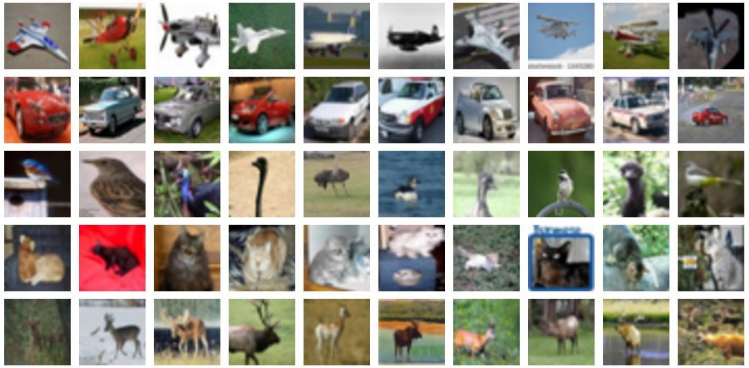
1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

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

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

Example training set

airplane
automobile
bird
cat
deer



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Example Dataset: **CIFAR10**

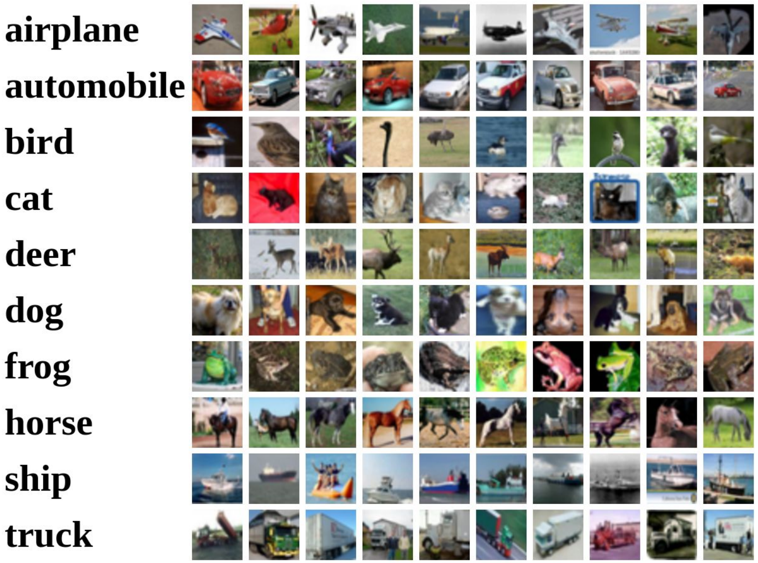
10 classes
50,000 training images
10,000 testing images

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

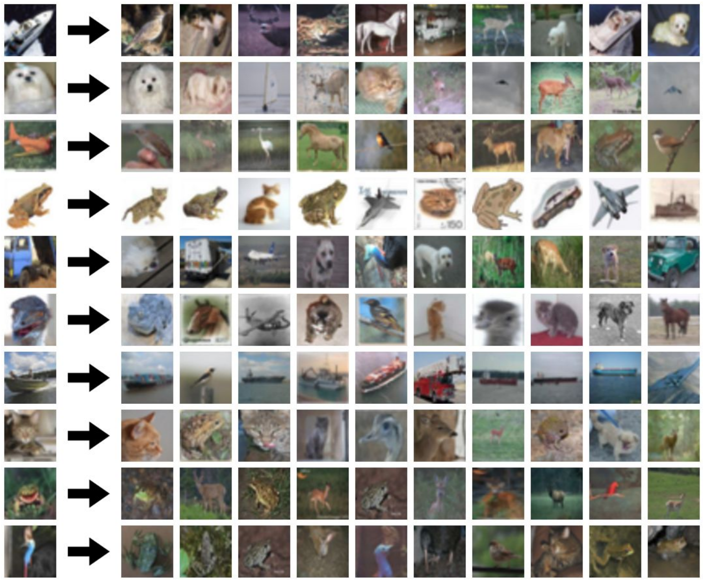
Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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.

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

$\xrightarrow{\text{add}}$ 456

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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
```

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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
```

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

```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?

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

```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?

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

```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

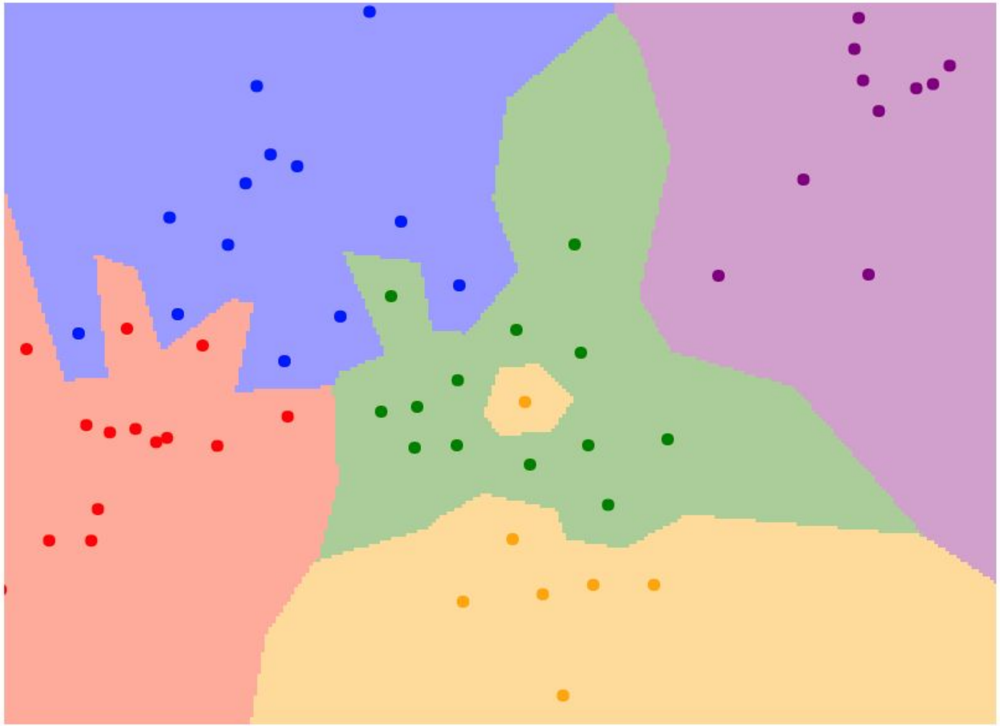
Q: With N examples, how fast are training and prediction?

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

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

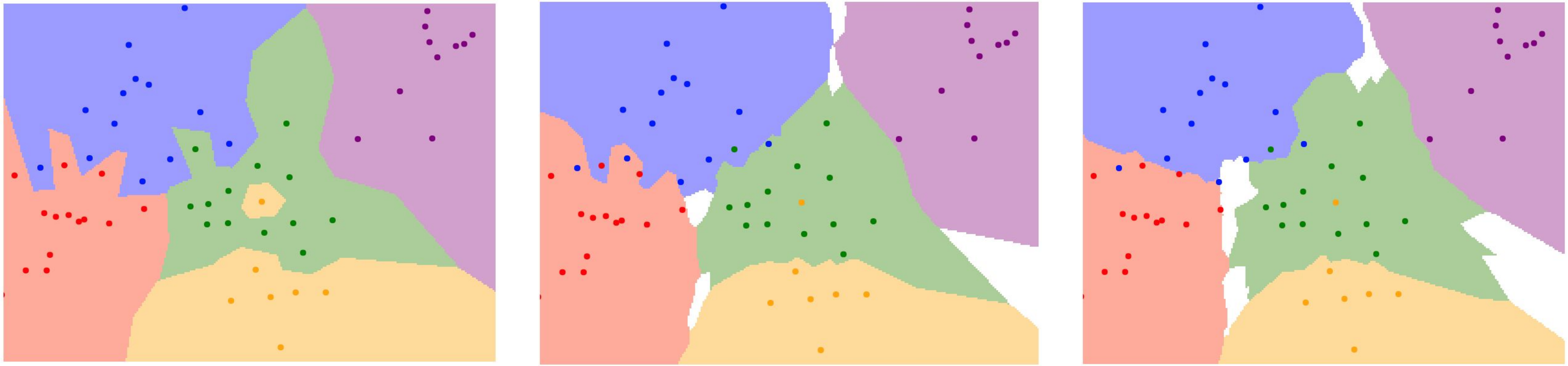
What does this look like?



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

K-Nearest Neighbors

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



K = 1 K = 3 K = 5

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Hyperparameters

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

These are **hyperparameters**: choices about the algorithm that we set rather than learn

Hyperparameters

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

These are **hyperparameters**: choices about the algorithm that we set rather than learn

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

Setting Hyperparameters

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

Your Dataset

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

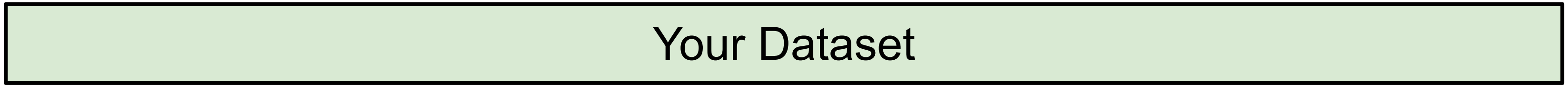
train	test

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train | test

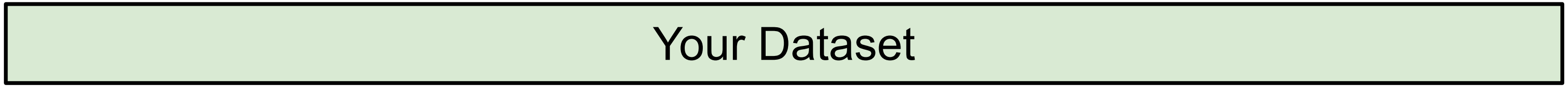
Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

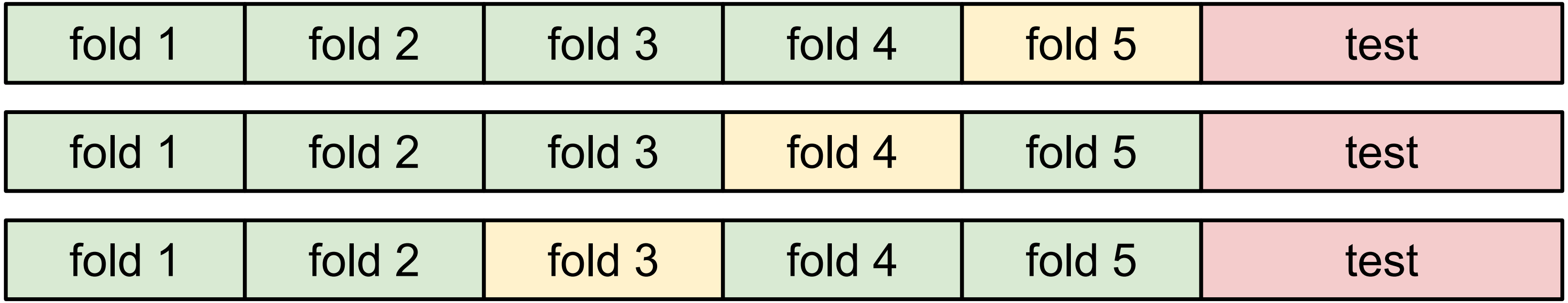
train | validation | test

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Setting Hyperparameters



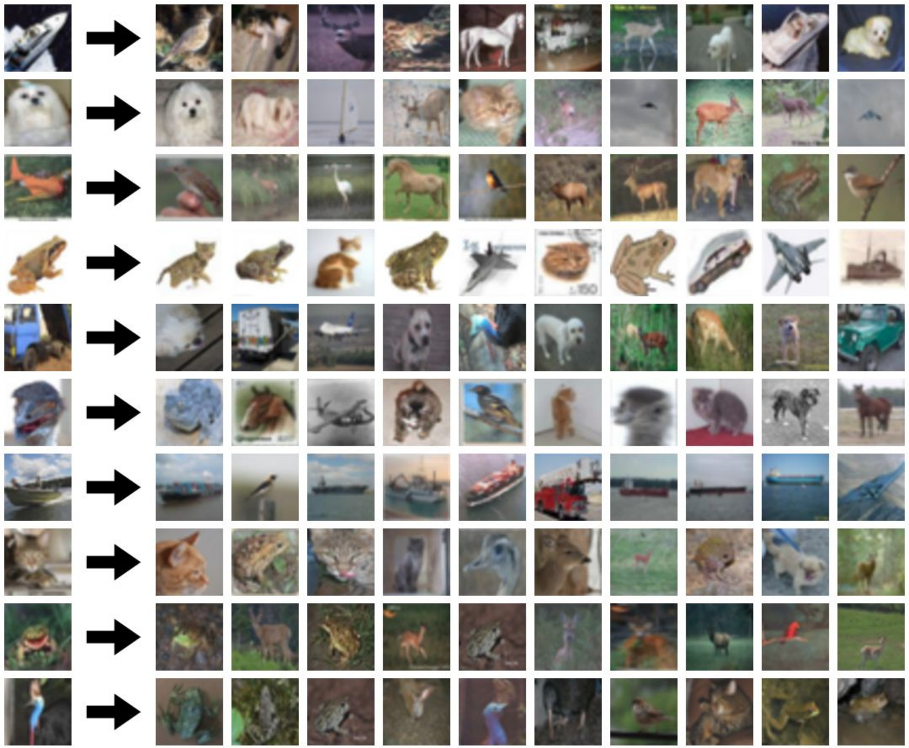
Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results



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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

What does this look like?



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

What does this look like?

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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 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 Classification

Recall CIFAR10

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

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

10,000 test images.

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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)

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

10 numbers giving class scores

W

parameters or weights

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{\text{10x1}} = \underbrace{\boxed{W}}_{\text{10x3072}}\overbrace{\boxed{x}}^{\text{3072x1}} + \boxed{b}\ \ \text{10x1}$$

Image

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

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

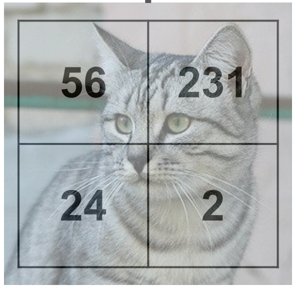
$\uparrow$

W
parameters
or weights

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

Stretch pixels into column



Input image

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Interpreting a Linear Classifier

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

What is this thing doing?

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

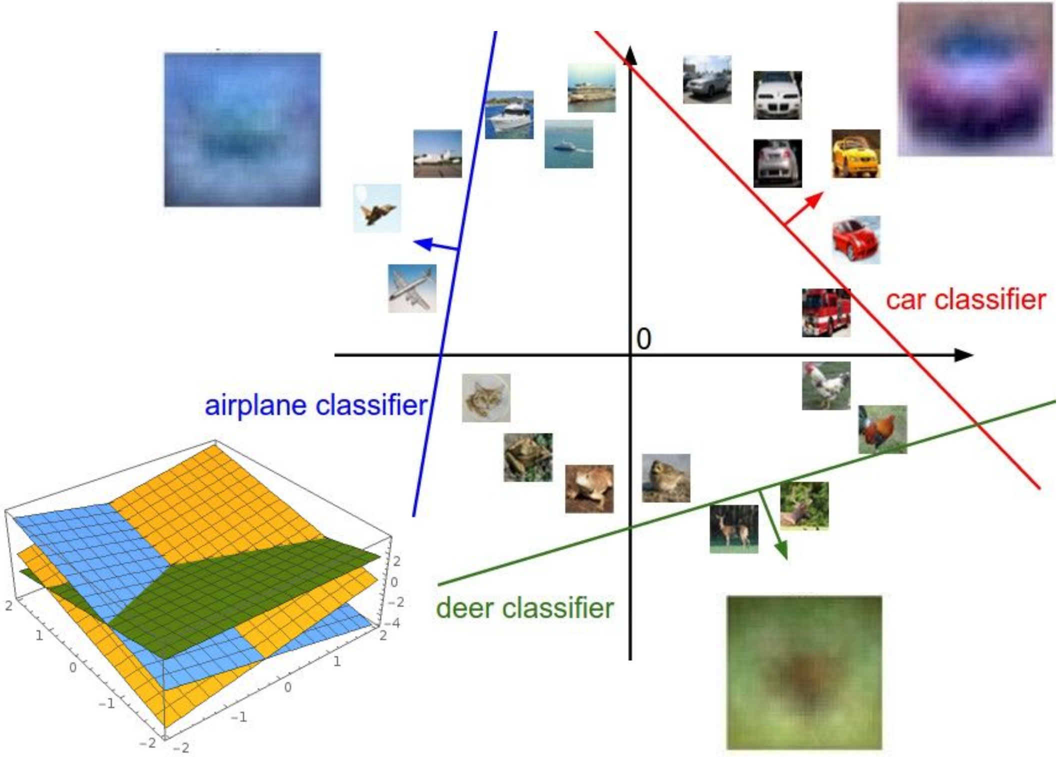
Interpreting a Linear Classifier

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

Example trained weights of a linear classifier trained on CIFAR-10:

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Interpreting a Linear Classifier



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

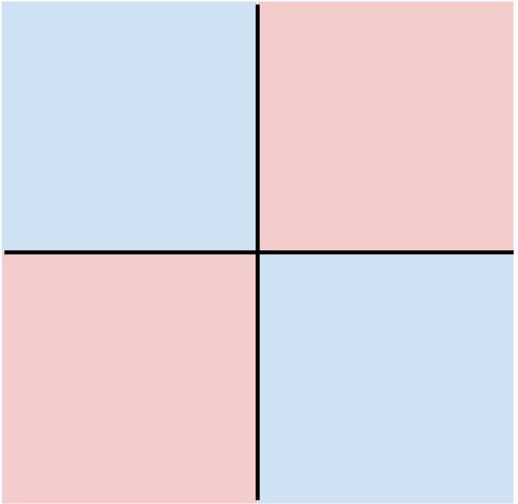
Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Hard cases for a linear classifier

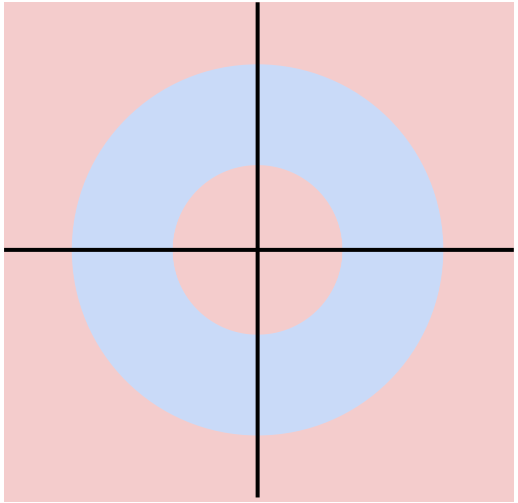
Class 1:
number of pixels > 0 odd

Class 2:
number of pixels > 0 even



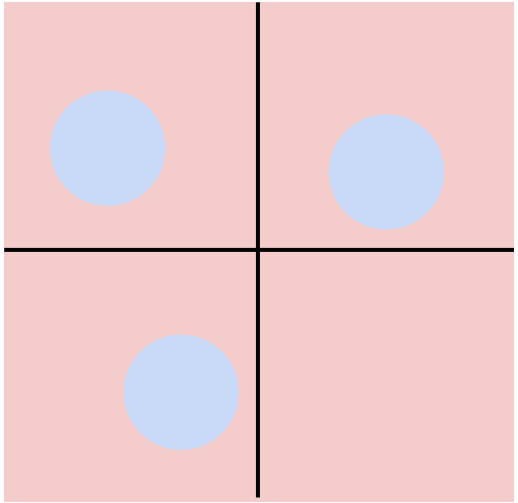
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

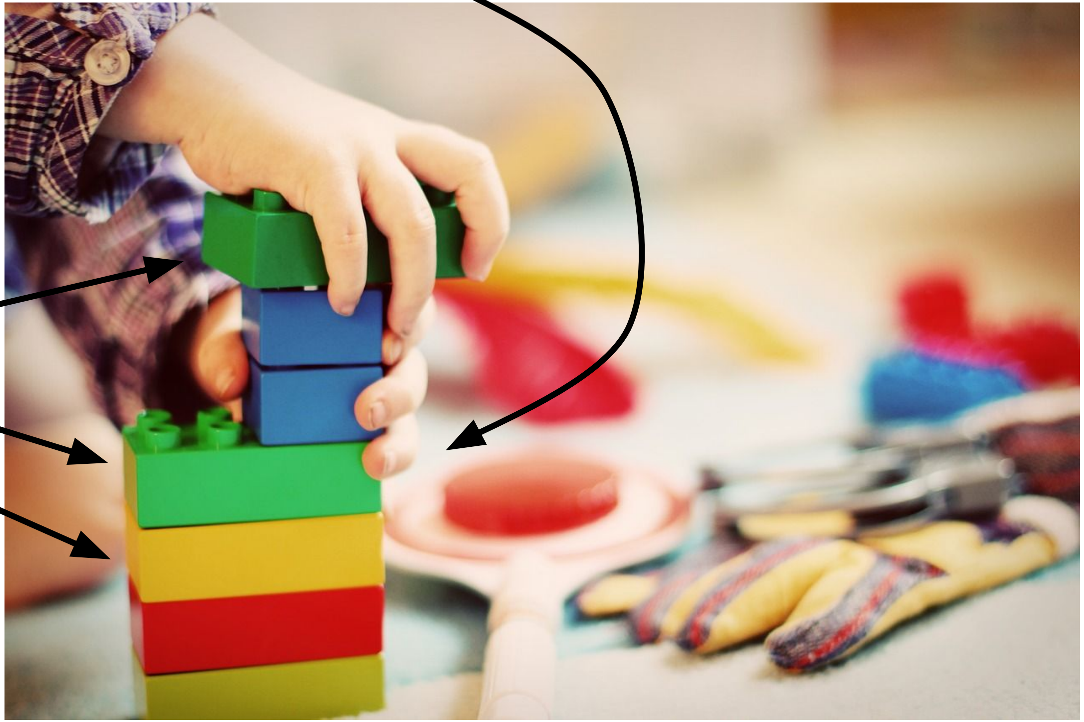
Class 2:
Everything else



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Neural Network

Linear classifiers



This image is CC0 1.0 public domain

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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:

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

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**

Previous losses:

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: $s = f(x_i, W)$

Least-squares regression has the form

$$L_i = \sum_i ||s_i - y_i||^2$$

The SVM loss has the form:

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

cat	**3.2**
car	5.1
frog	-1.7

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

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

cat	**3.2**
car	5.1
frog	-1.7

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s=f(x_i;W)$$

cat	**3.2**
car	5.1
frog	-1.7

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad s = f(x_i; W)$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

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

cat	**3.2**
car	5.1
frog	-1.7

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

cat	**3.2**
car	5.1
frog	-1.7

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

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

---

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

Softmax Classifier (Multinomial Logistic Regression)

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

cat	**3.2**
car	5.1
frog	-1.7

unnormalized log probabilities

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

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

unnormalized log probabilities

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

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

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

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

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

unnormalized probabilities

	unnormalized log probabilities		unnormalized probabilities		probabilities	
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

unnormalized log probabilities

probabilities

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

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

Q: What is the min/max possible loss L_i?

unnormalized probabilities

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

→ L_i = -log(0.13)
= **0.89**

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Softmax Classifier (Multinomial Logistic Regression)

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

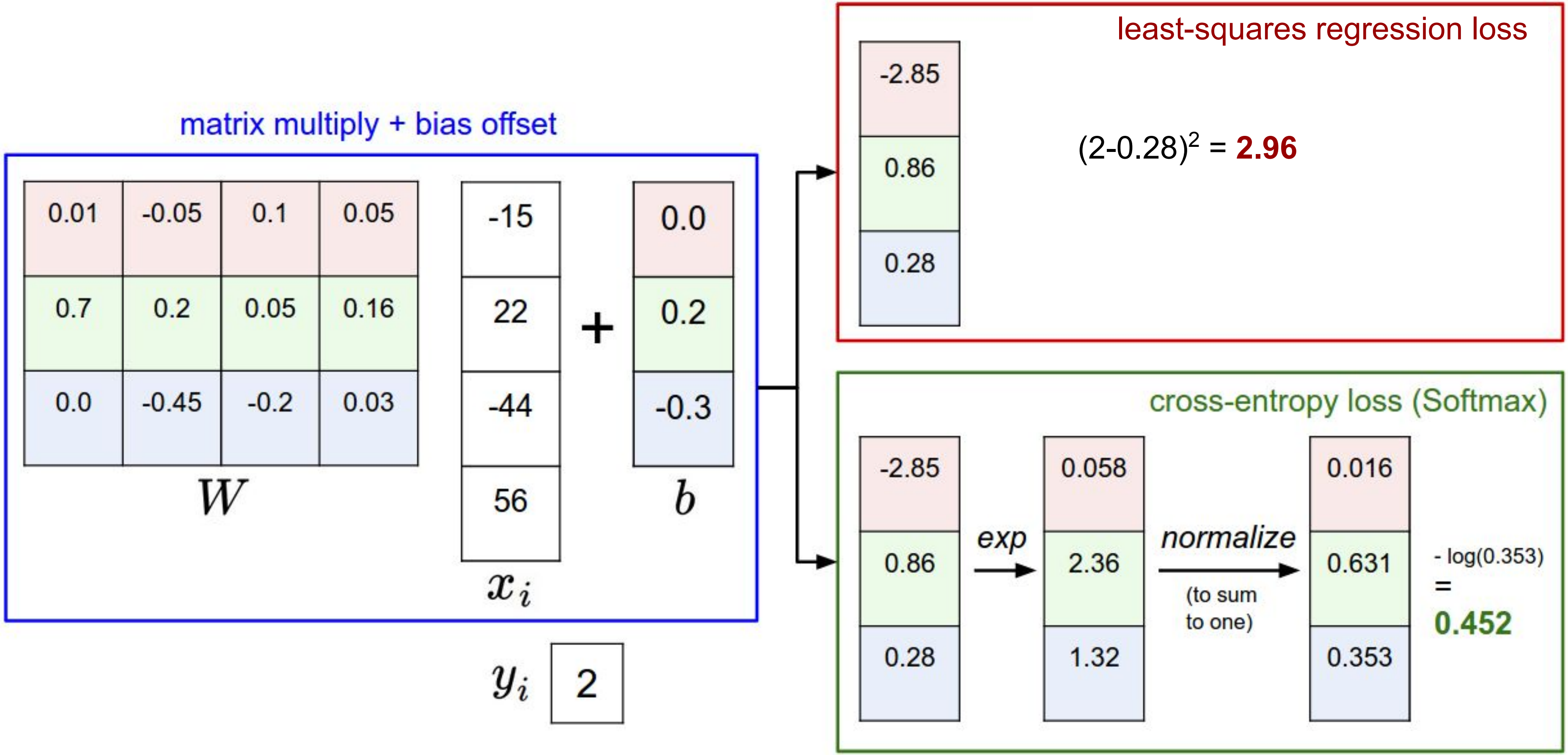
unnormalized probabilities

Q2: Usually at initialization W is small so all s ≈ 0.
What is the loss?

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ L_i = -log(0.13)
= **0.89**

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung



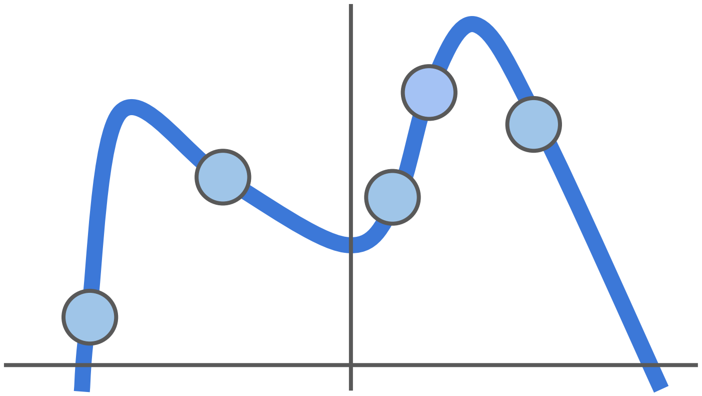
Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{}$$

Data loss: Model predictions should match training data

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

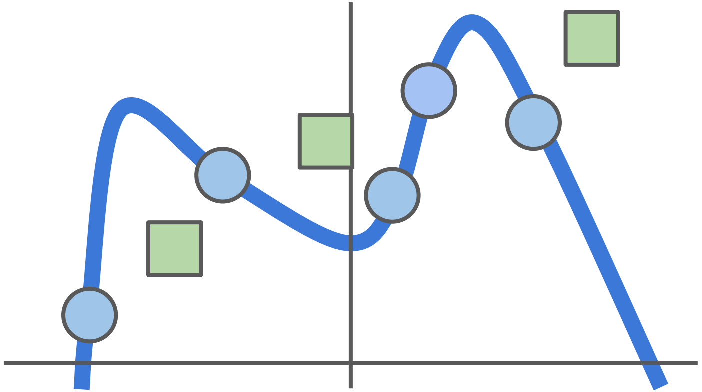
Data loss: Model predictions should match training data



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

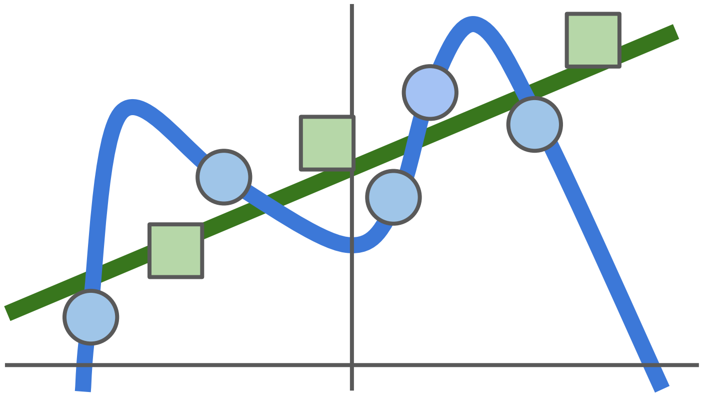
Data loss: Model predictions should match training data



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions should match training data

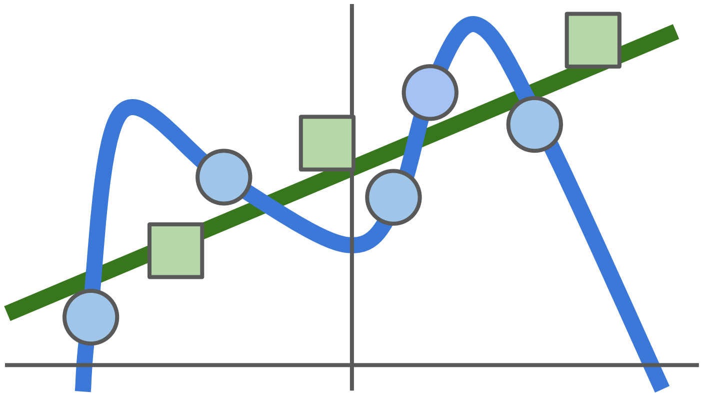


Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

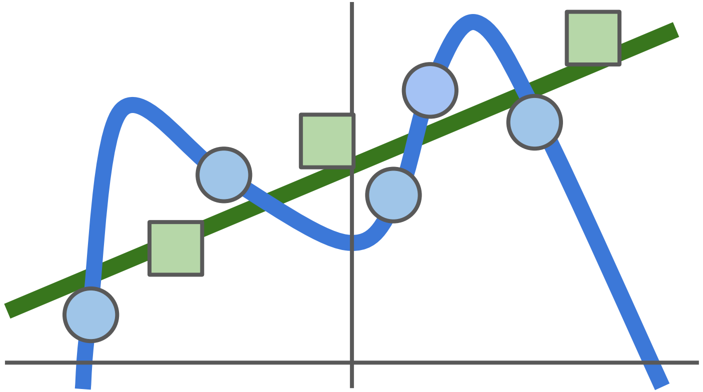
Regularization: Model should be “simple”, so it works on test data



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data



Regularization: Model should be “simple”, so it works on test data

Occam’s Razor:
“Among competing hypotheses, the simplest is the best”
William of Ockham, 1285 - 1347

L2 Regularization (Weight Decay)

$$x = [1, 1, 1, 1]$$

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Recap

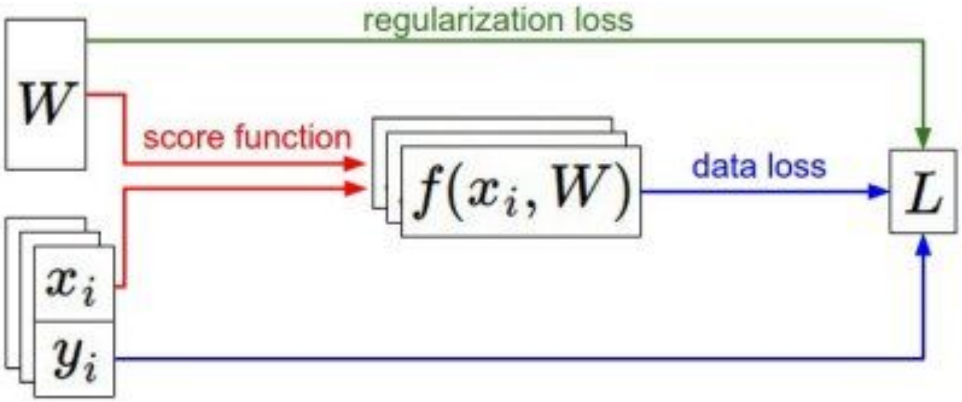
- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

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

Full loss

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Optimization

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Walking man image is CC0 1.0 public domain

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Strategy: **Follow the slope**

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Strategy: **Follow the slope**

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[**-2.5**,
?,
?,
?,
?,…]

(1.25322 - 1.25347)/0.0001
= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

current W:	**W + h** (second dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (second dim):

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,

(1.25353 - 1.25347)/0.0001
= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (third dim)**:**

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

gradient dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,…]

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (third dim):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?,…]

(1.25347 - 1.25347)/0.0001
= 0

$$\boxed{\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

This is silly. The loss is just a function of W:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

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

$$s = f(x; W) = Wx$$

want $\nabla_W L$

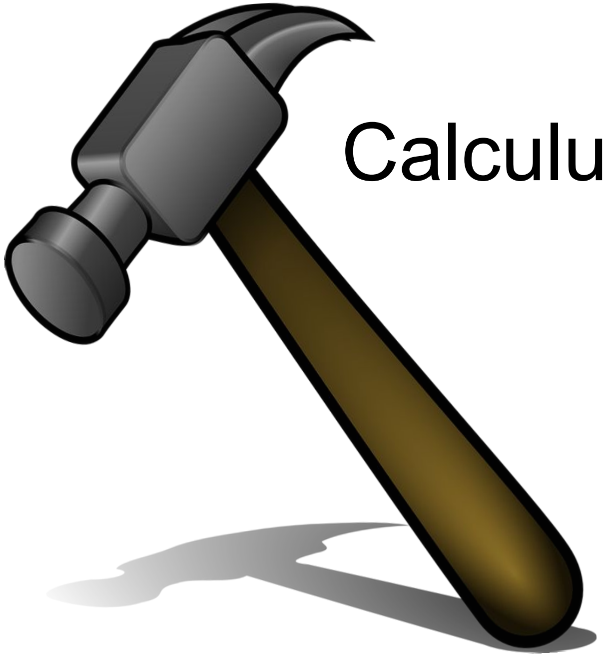
This is silly. The loss is just a function of W:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

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

$$s = f(x; W) = Wx$$

want $\nabla_W L$



Use calculus to compute an **analytic gradient**

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

dW = ...
(some function
data and W)

gradient dW:

[-2.5,
0.6,
0,
0.2,
0.7,
-0.5,
1.1,
1.3,
-2.1,…]

In summary:

- Numerical gradient: approximate, slow, easy to write

- Analytic gradient: exact, fast, error-prone

=>

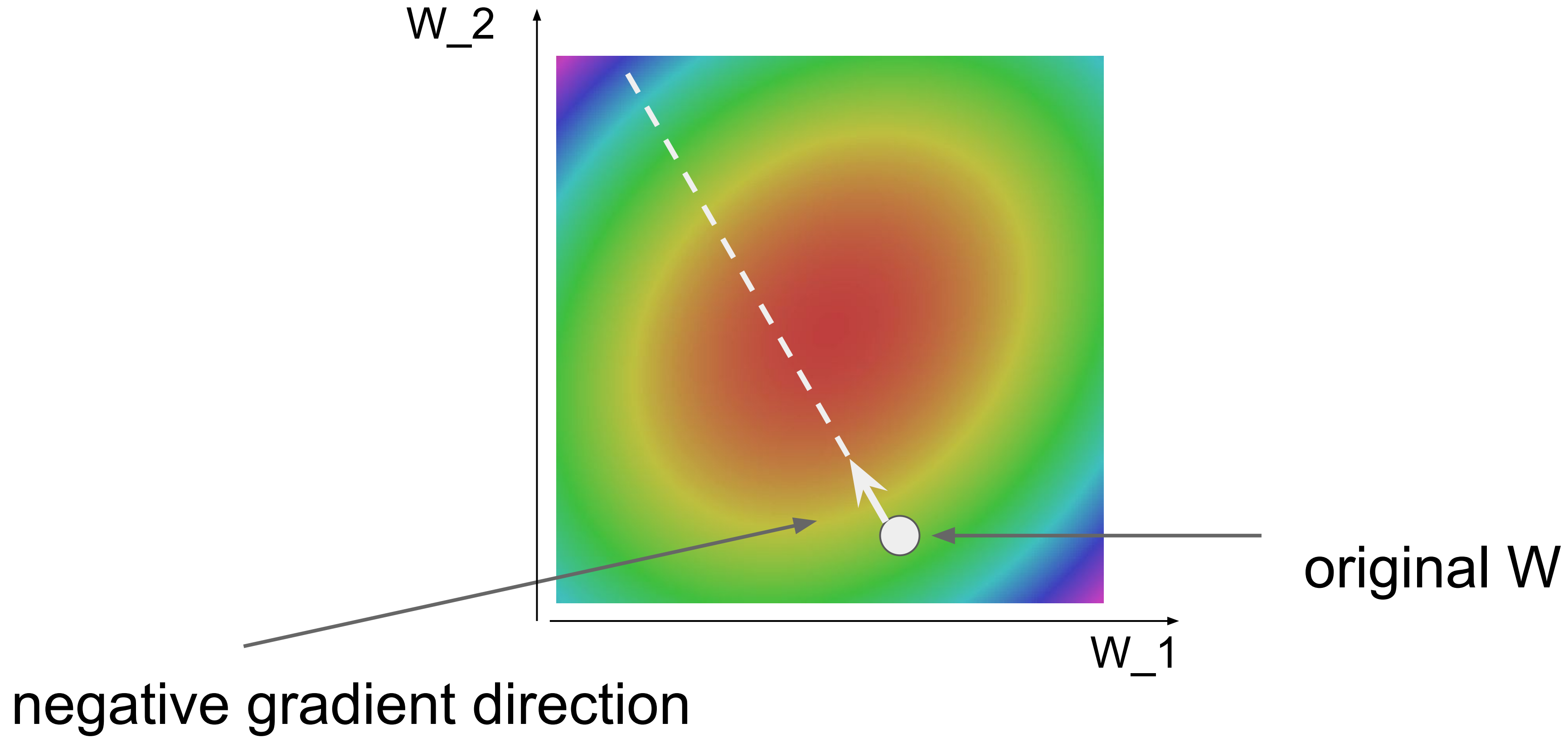
<u>In practice:</u> Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check.**

Gradient Descent

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

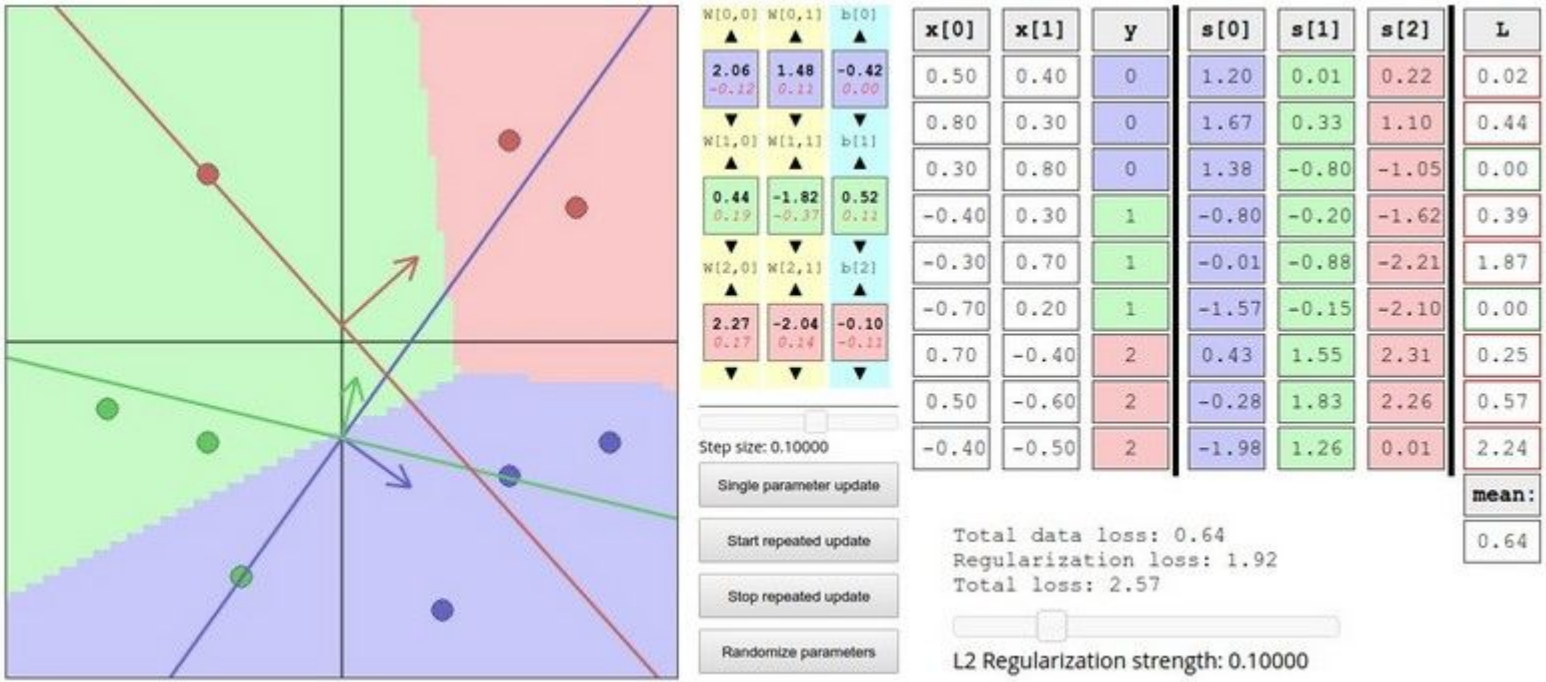
Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Interactive Web Demo

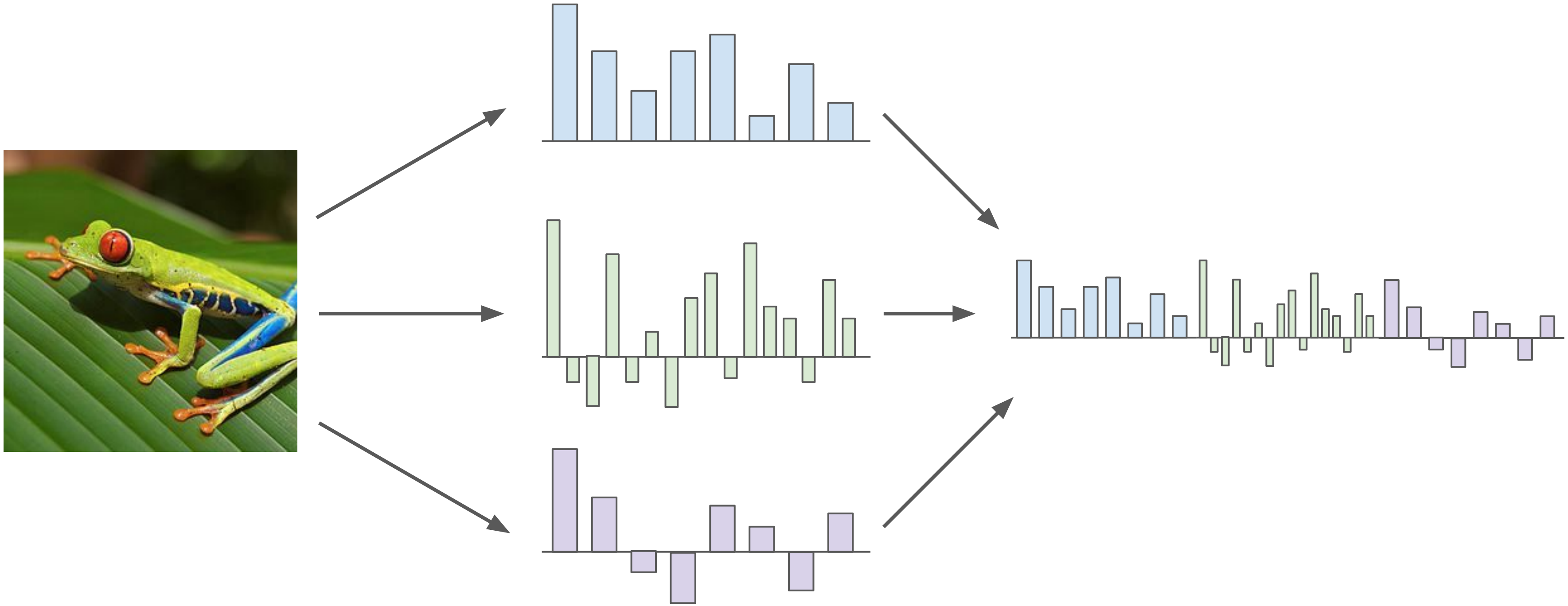


http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

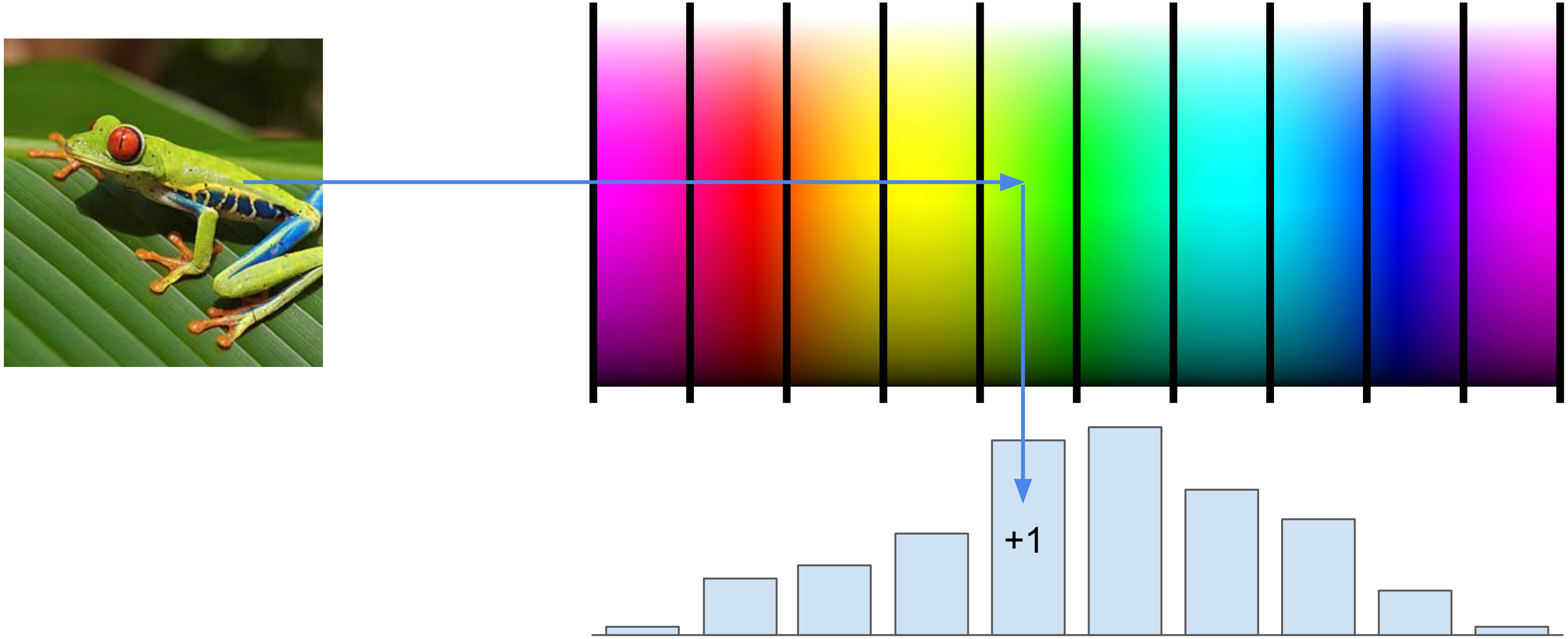
Image features

Image Features



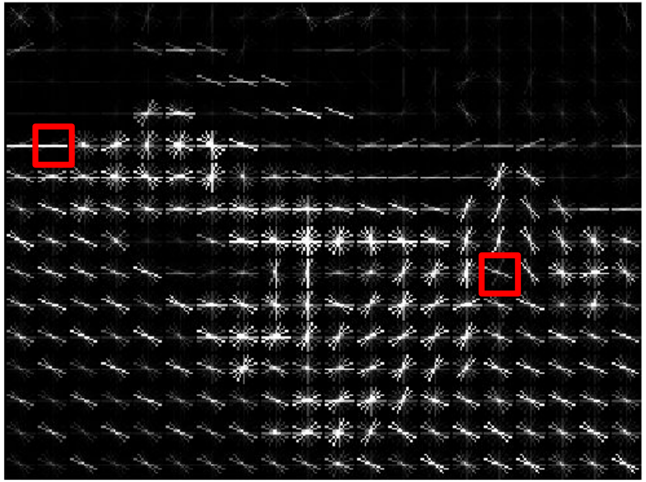
Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Example: Color Histogram



Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Example: Histogram of Oriented Gradients (HoG)



Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

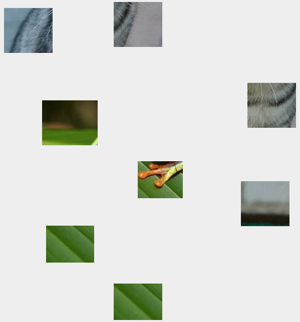
Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Example: Bag of Words

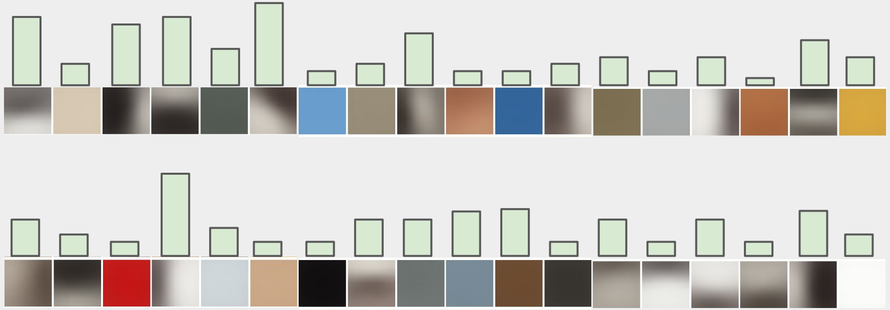
Step 1: Build codebook

Extract random patches



Cluster patches to form “codebook” of “visual words”

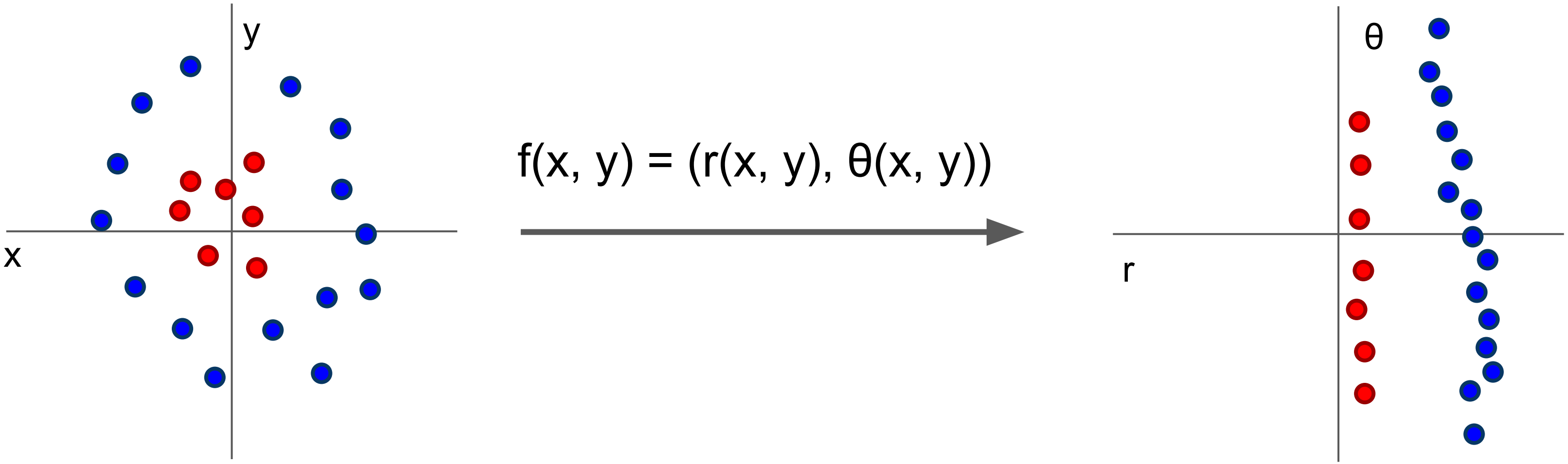
Step 2: Encode images



Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Image Features: Motivation



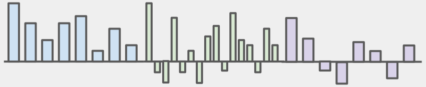
$$f(x, y) = (r(x, y), \theta(x, y))$$

Cannot separate red and blue points with linear classifier

After applying feature transform, points can be separated by linear classifier

Image features vs ConvNets

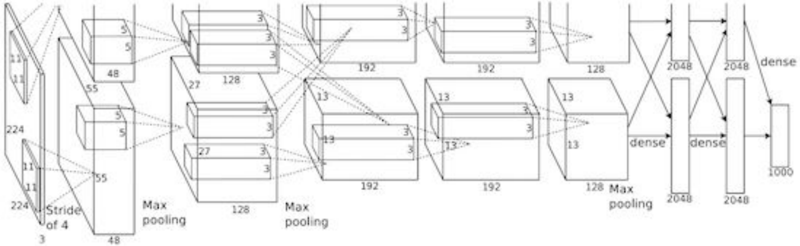
Feature Extraction



f

10 numbers giving scores for classes

training



Krizhevsky, Sutskever, and Hinton, “Imagenet classification with deep convolutional neural networks”, NIPS 2012.
Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

training

Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Neural networks

(**Before**) Linear score function: $f = Wx$

Neural networks

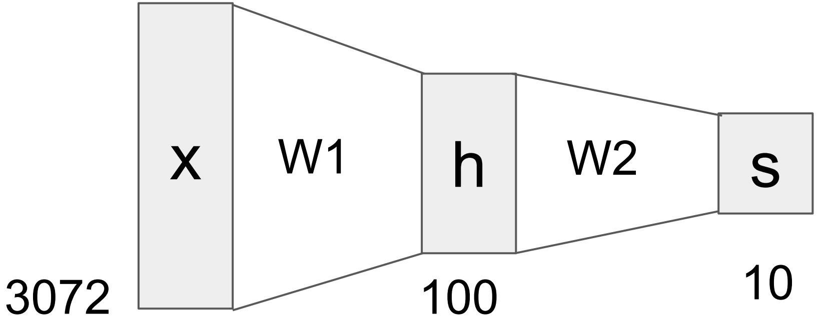
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

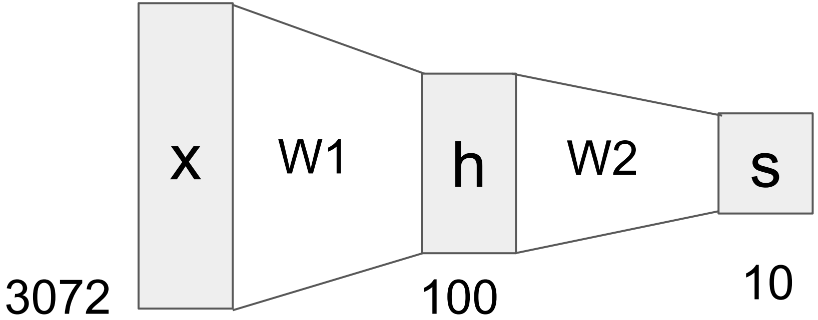


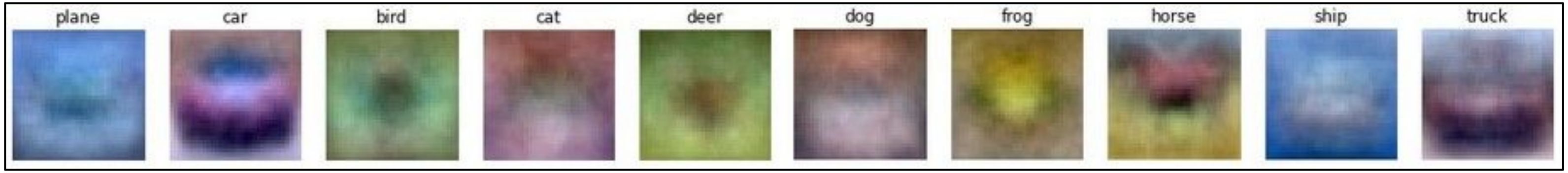
Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$





Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

Neural networks

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Next time:

Backpropagation