Lecture 2:
Image Classification
Administrative: Assignment 1

Due 4/14 11:59pm

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

Project proposal due 4/24 (Monday) 11:59pm

Find your teammates on EdStem

Collaboration: EdStem

“Is X a valid project for 493G1?”
- Anything related to deep learning
- Maximum of 3 students per team
- Make a EdStem private post or come to TA Office Hours

More info on the website
Administrative: Fridays

This Friday 10:30-11:20 am (recording will be made available)

Room: SIG 134

Python / Numpy, Google Cloud Platform, Google Colab

Presenter: Sarah Pratt (TA)
# Syllabus

## Deep learning Fundamentals
- Data-driven approaches
- Linear classification & kNN
- Loss functions
- Optimization
- Backpropagation
- Multi-layer perceptrons
- Neural Networks
- Convolutions
- RNNs / LSTMs
- Transformers

## Practical training skills
- Pytorch 1.4 / Tensorflow 2.0
- Activation functions
- Batch normalization
- Transfer learning
- Data augmentation
- Momentum / RMSProp / Adam
- Architecture design

## Applications
- Image captioning
- Interpreting machine learning
- Generative AI
- Fairness & ethics
- Data-centric AI
- Deep reinforcement learning
- Self-supervised learning
- Diffusion
- LLMs
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)

Cat
Dog
Bird
Truck
Plane

This image by Nikita is licensed under CC-BY 2.0
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

RGB values are a function of surface materials, color, light source, etc.
Challenges: Background Clutter
Challenges: Occlusion
Challenges: Deformation
Challenges: Intraclass variation
Image classification is a building block for other tasks
Image classification is a building block for other tasks.

Image Captioning
Vinyals et al, 2015
Karpathy and Fei-Fei, 2015

A white teddy bear sitting in the grass
A man in a baseball uniform throwing a ball
A woman is holding a cat in her hand
A man riding a wave on top of a surfboard
A cat sitting on a suitcase on the floor
A woman standing on a beach holding a surfboard
Image classification is a building block for other tasks

Example: Playing Go

Where to play next?

(1, 1)
(1, 2)
...
(1, 19)
...
(19, 19)
Modern computer vision algorithms

Classifiers today take 1ms to classify images. And can handle thousands of categories.
An image classifier: can we implement this as a normal software function?

```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.
This is why expert systems in the 80s led to the AI winter.

Originally called heuristic programming project.
Attempts have been made

Find edges

Find corners

John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels

Example training set

<table>
<thead>
<tr>
<th>airplane</th>
</tr>
</thead>
<tbody>
<tr>
<td>automobile</td>
</tr>
<tr>
<td>bird</td>
</tr>
<tr>
<td>cat</td>
</tr>
<tr>
<td>deer</td>
</tr>
</tbody>
</table>
Example dataset: MNIST

10 classes: Digits 0 to 9
28x28 grayscale images
50k training images
10k test images
Example dataset: CIFAR10

10 classes
50k training images (5k per class)
10k testing images (1k per class)
32x32 RGB images

We will use this dataset for homework assignments
Example dataset: CIFAR100

100 classes
50k training images (500 per class)
10k testing images (100 per class)
32x32 RGB images

20 superclasses with 5 classes each:

Aquatic mammals: beaver, dolphin, otter, seal, whale
Trees: Maple, oak, palm, pine, willow
Example dataset: ImageNet (ILSVRC challenge)

1000 classes

~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)

Performance metric: **Top 5 accuracy**
Algorithm predicts 5 labels for each image; one of them needs to be right
Example dataset: MIT Places

365 classes of different scene types

~8M training images
18.25K val images (50 per class)
328.5K test images (900 per class)

Images have variable size, often resize to 256x256 for training
Example dataset: Omniglot

1623 categories: characters from 50 different alphabets

20 images per category

Meant to test few shot learning
Machine Learning: Data-Driven Approach

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

Example training set

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

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
Nearest Neighbor Classifier
First classifier: Nearest Neighbor

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

Memorize all data and labels

```python
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

Training data with labels

Distance Metric \( \rightarrow \mathbb{R} \)

What is a good distance metric?
Distance Metric to compare images

**L1 distance:**

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

<table>
<thead>
<tr>
<th>test image</th>
<th>training image</th>
<th>pixel-wise absolute value differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 32 10 18</td>
<td>10 20 24 17</td>
<td>46 12 14 1</td>
</tr>
<tr>
<td>90 23 128 133</td>
<td>8 10 89 100</td>
<td>82 13 39 33</td>
</tr>
<tr>
<td>24 26 178 200</td>
<td>12 16 178 170</td>
<td>12 10 0 30</td>
</tr>
<tr>
<td>2 0 255 220</td>
<td>4 32 233 112</td>
<td>2 32 22 108</td>
</tr>
</tbody>
</table>

\[ \text{add} \quad 456 \]
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-dimensional 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]
        # let's make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
Nearest Neighbor classifier

Memorize training data

```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
```
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimension of size N ""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Nearest Neighbor classifier

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

Many methods exist for fast / approximate nearest neighbor (beyond the scope of 231N!)

A good implementation:

https://github.com/facebookresearch/faiss

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

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
Example outputs from a NN classifier on CIFAR:
Example outputs from a NN classifier on CIFAR:
Assume each dot is a training image.

Assume all images are two dimensional.

What does this classifier look like?

1-nearest neighbor
Decision boundary is the boundary between two classification regions.

1-nearest neighbor
Yellow point in the middle of green might be mislabeled.

1-NN is not robust to label noise.
K-Nearest Neighbors

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

K = 1

K = 3
K-Nearest Neighbors

Using more neighbors helps smooth out rough decision boundaries

K = 1

K = 3
K-Nearest Neighbors

Find more labels near uncertain white regions

K = 1

K = 3
K-Nearest Neighbors

Larger K smooths boundaries more and leads to more uncertain regions

K = 1

K = 3

K = 5
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I^p_1 - I^p_2|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I^p_1 - I^p_2)^2}$$
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

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

L2 (Euclidean) distance

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

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

BAD: No idea how algorithm will perform on new data

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

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

BAD: No idea how algorithm will perform on new data

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

Better!
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

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Example Dataset: **CIFAR10**

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

Test images and nearest neighbors

Example of 5-fold cross-validation for the value of $k$.

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation.

(Seems that $k \approx 7$ works best for this data)
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!

(*) Note: The notation (*) is used to indicate that there are conditions or restrictions on the function representation, but these are not specified in the text.

![Graph showing K-Nearest Neighbor approximation to a complex function](image)
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!

![Graph showing nearest neighbor approximation of a function with 10 training points.](image)
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!

![Diagram showing 20 training points, true function, and nearest neighbor function.]
K-Nearest Neighbor: Universal Approximation

As the number of training samples goes to infinity, nearest neighbor can represent any(*) function!
**Problem**: curse of dimensionality

**Curse of dimensionality**: For uniform coverage of space, number of training points needed grows exponentially with dimension.

- **Dimensions = 1**
  - Points = 4

- **Dimensions = 2**
  - Points = $4^2$

- **Dimensions = 3**
  - Points = $4^3$
Problem: curse of dimensionality

Curse of dimensionality: For uniform coverage of space, number of training points needed grows exponentially with dimension

Number of possible 32x32 binary images:

\[ 2^{32 \times 32} = 10^{308} \]

Number of elementary particles in the visible universe:

\[ 10^{97} \]
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!
k-Nearest Neighbor with pixel distance *never used*.

- Distance metrics on pixels are not informative

(All three images on the right have the same pixel distances to the one on the left)
Linear Classifier
Parametric Approach

Image

Array of 32x32x3 numbers (3072 numbers total)

\( f(x, W) \)

10 numbers giving class scores

\( W \)

parameters or weights
Parametric Approach: Linear Classifier

\[ f(x, W) = Wx \]

Array of \(32 \times 32 \times 3\) numbers (3072 numbers total)

Image

\(W\) parameters or weights

10 numbers giving class scores
Parametric Approach: Linear Classifier

Image parameters or weights $W$

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

$f(x, W) = Wx$

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

$W$ parameters or weights
Parametric Approach: Linear Classifier

\[ f(x, W) = Wx + b \]

- **Image**: Array of 32x32x3 numbers (3072 numbers total)
- **\( W \)**: Parameters or weights
- **\( f(x, W) \)**: 10x3072
- **\( Wx \)**: 10x1
- **\( b \)**: 10x1
- **Class scores**: 10 numbers giving class scores
Neural Network

Linear classifiers

This image is in the public domain.
Linear layers

[Krizhevsky et al. 2012]

[He et al. 2015]
Recall CIFAR10

50,000 training images
each image is $32 \times 32 \times 3$

10,000 test images.
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

\[ W \begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix} + b = \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} = \begin{bmatrix} 1.1 \\ 3.2 \\ 24 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \]

Cat score
Dog score
Ship score
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

Input image

\[ W \]

\[ b \]

\[
\begin{pmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
56 \\
231 \\
24 \\
2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{pmatrix}
\]

Cat score
Dog score
Ship score

Flatten tensors into a vector

\[(2,2)\]

\[(3,4)\]

\[(4,)\]

\[(3,)\]
**Algebraic viewpoint:** Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

Input image

\[
\begin{align*}
W &= \begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{bmatrix} \\
&= \begin{bmatrix}
56 \\
231 \\
24 \\
2 \\
\end{bmatrix} \\
&= \begin{bmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{bmatrix} \\
&= \begin{bmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{bmatrix} \\
\end{align*}
\]

Likelihood of being a cat

Cat score

Dog score

Ship score
Algebraic viewpoint: Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

Input image

Cat template

\[
\begin{align*}
W &= \begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{bmatrix} \\
b &= \begin{bmatrix}
56 \\
231 \\
24 \\
2 \\
\end{bmatrix} \\
\text{Cat score} &= -96.8 \\
\text{Dog score} &= 437.9 \\
\text{Ship score} &= 61.95
\end{align*}
\]
Algebraic viewpoint: Bias trick to simply computation

Flatten tensors into a vector

Input image

\[
\begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 & 1.1 \\
1.5 & 1.3 & 2.1 & 0.0 & 3.2 \\
0 & 0.25 & 0.2 & -0.3 & -1.2 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
56 \\
231 \\
24 \\
2 \\
1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{bmatrix}
\]
Visual Viewpoint: learning templates

Algebraic viewpoint:

\[
\begin{align*}
\text{Input image} & \quad \begin{bmatrix}
56 & 231 \\
24 & 2 \\
\end{bmatrix} \\
\text{Stretch pixels into column} & \quad W & \quad \begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{bmatrix} \\
\text{} & \quad + & \quad \begin{bmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{bmatrix} & \quad = & \quad \begin{bmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{bmatrix} \\
\text{Score} & \quad \begin{bmatrix}
\text{W} \\
\text{b} \\
\end{bmatrix} & \quad \begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{bmatrix} & \quad \begin{bmatrix}
1.1 \\
3.2 \\
-1.2 \\
\end{bmatrix} & \quad \begin{bmatrix}
-96.8 \\
437.9 \\
61.95 \\
\end{bmatrix}
\end{align*}
\]
Visual Viewpoint: learning templates

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

Input image

\[
\begin{align*}
W &= \begin{bmatrix} 0.2 & -0.5 \\ 0.1 & 2.0 \end{bmatrix} \\
b &= \begin{bmatrix} 1.1 \\ 3.2 \end{bmatrix} \\
\text{Score} &= \begin{bmatrix} -96.8 \\ 437.9 \end{bmatrix}
\end{align*}
\]
Visual Viewpoint: learning templates
Visual Viewpoint: learning templates

airplane  automobile  bird  cat  deer  dog  frog  horse  ship  truck

Input image

W
0.2 -0.5
0.1 2.0

1.5 1.3
2.1 0.0

0 0.25
0.2 -0.3

Score
-96.8
437.9
61.95

Ranjay Krishna, Aditya Kusupati  Lecture 2 - 85  March 30, 2023
Visual Viewpoint: learning templates

The diagram shows a visual viewpoint for different categories such as airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each category is represented with a set of images, illustrating the learning templates for each class.

The right side of the diagram includes a neural network representation with input images and corresponding weights and biases. The score for each class is calculated, showing the classification outcomes for the input images.
Geometric Viewpoint: linear decision boundaries

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

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)
Geometric Viewpoint: linear decision boundaries

\[ f(x, W) = Wx + b \]

Array of \(32 \times 32 \times 3\) numbers
(3072 numbers total)
Geometric Viewpoint: linear decision boundaries

\[ f(x, W) = Wx + b \]

Array of $32 \times 32 \times 3$ numbers
(3072 numbers total)
Hard cases for a linear classifier

Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants

Class 1:
1 ≤ L2 norm ≤ 2

Class 2:
Everything else

Class 1:
Three modes

Class 2:
Everything else
Recall the Minsky report 1969 from last lecture

Unable to learn the XNOR function

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>F(x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Three viewpoints for interpreting linear classifiers

**Algebraic Viewpoint**

\[ f(x,W) = Wx \]

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space
Coming up:
- Loss function
- Optimization
- ConvNets!

\[ f(x,W) = Wx + b \]

(quantifying what it means to have a “good” \( W \))

(start with random \( W \) and find a \( W \) that minimizes the loss)

(tweak the functional form of \( f \))