RSNA Pneumonia Detection Challenge
Computer Vision 101

Santiago Hincapie

Universidad EAFIT

September 6, 2018

Pneumonia

- Specific Character Set
- SOP Class UIDS
- SOP Instance UID
- Study Date
- Study Time
- Accession Number
- Modality
- Conversion Type
- Referring Physician's Name
- Series Description
- Patient's Name
- Patient ID
- Patient's Birth Date
- Patient's Sex
- Patient's Age
- Study Instance UID
- Series Instance UID
- Study ID
- Series Number
- Instance Number
- Patient Orientation
- Samples per Pixel
- Photometric Interpretation
- Rows
- Columns
- Pixel Spacing
- Bits Allocated
- Bits Stored
- High Bit
- Pixel Representation
- Lossy Image Compression

Computer Vision

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Pre-requisite

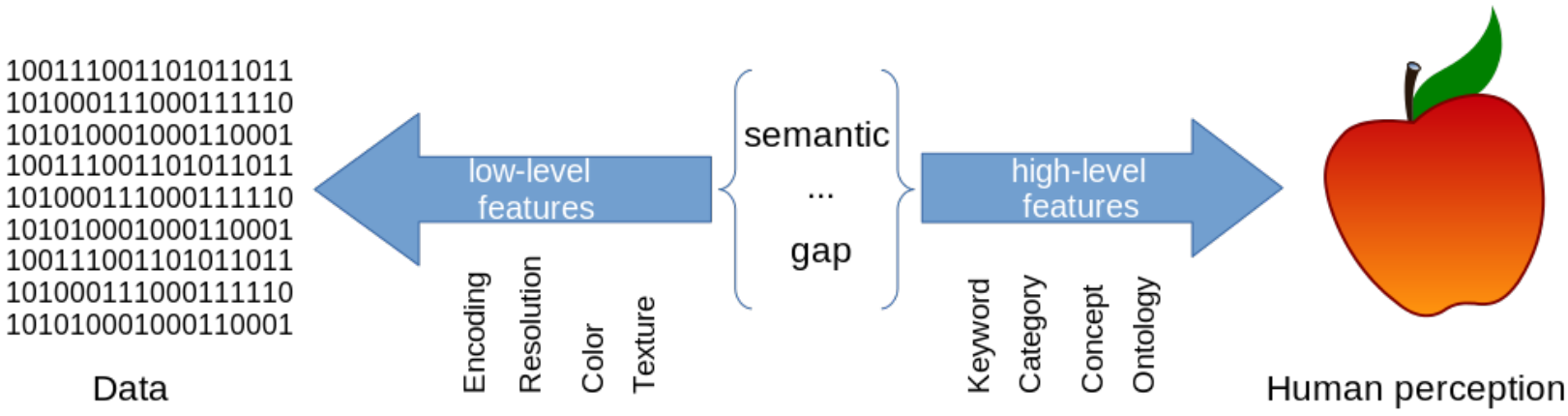
- Proficiency in Python and Numpy
- Linear Algebra
- Statistics
- Calculus Basics
- Algorithms

Image Classification

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

cat

The Problem: Semantic Gap

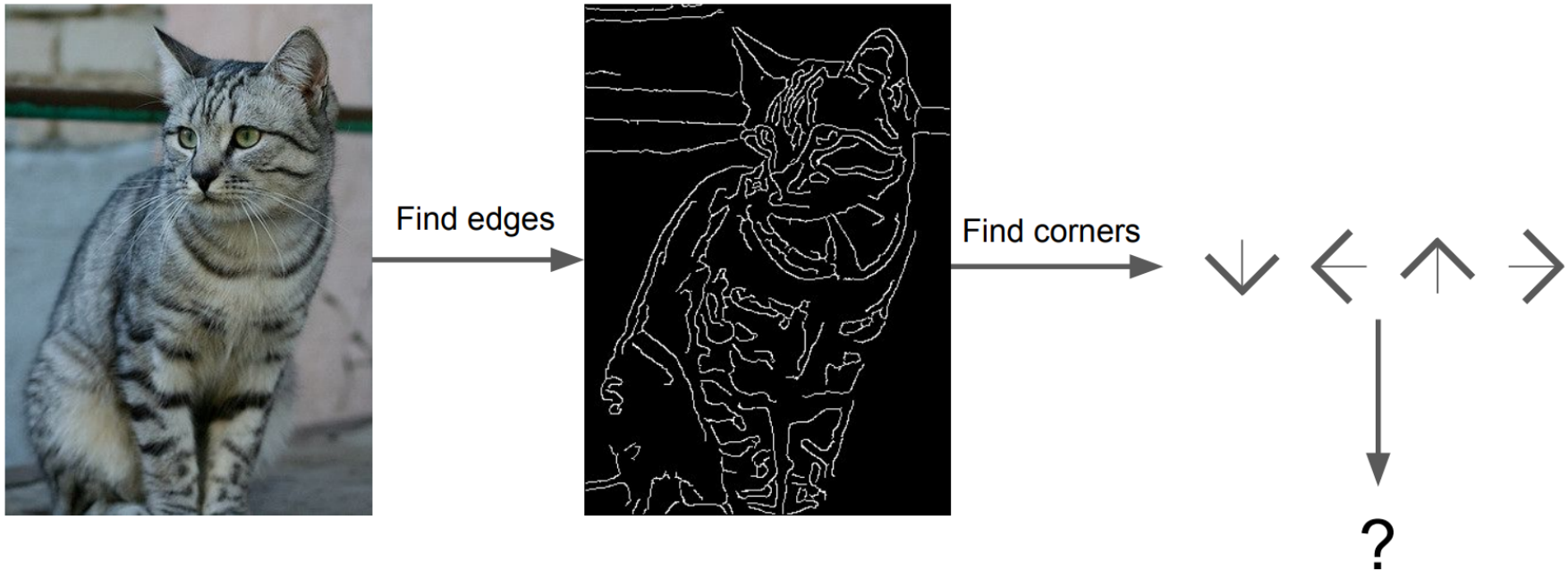


An Image Classifier

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

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

Attempts have been made



Data-Driven Approach

- Collect a dataset of images and labels
- Use Machine Learning to train a classifier
- Evaluate the classifier on new images

Data-Driven Approach

- Collect a dataset of images and labels
- Use Machine Learning to train a classifier
- Evaluate the classifier on new images

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

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

Machine Learning?

"[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed."
–Arthur Samuel, 1959

Machine Learning?

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."–Tom Mitchell, 1997

First Classifier: Nearest Neighbor

`train` $\rightarrow$ Memorize all data and labels

`predict` $\rightarrow$ Predict the label of the most similar training image

Distance Metric

$$L_1 \text{ distance} : d_1(A, B) = \sum |A_{ij} - B_{ij}|$$

$$\left| \begin{bmatrix} 23 & 21 & 16 \\ 90 & 23 & 12 \\ 95 & 32 & 34 \end{bmatrix} - \begin{bmatrix} 34 & 31 & 56 \\ 30 & 63 & 14 \\ 65 & 22 & 23 \end{bmatrix} \right| = \begin{bmatrix} 11 & 10 & 40 \\ 60 & 40 & 2 \\ 30 & 10 & 11 \end{bmatrix} \xrightarrow{\text{add}} 214$$

```
import numpy as np
class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    num_test = X.shape[0]
    Ypred = np.zeros(num_test)
    for i in range(num_test):
      dis = np.sum(np.abs(self.Xtr - X[i,:]),
                   axis=1)
      min_i = np.argmin(dis)
      Ypred[i] = self.ytr[min_index]
      return test_labels
```

Nearest Neighbor Classifier

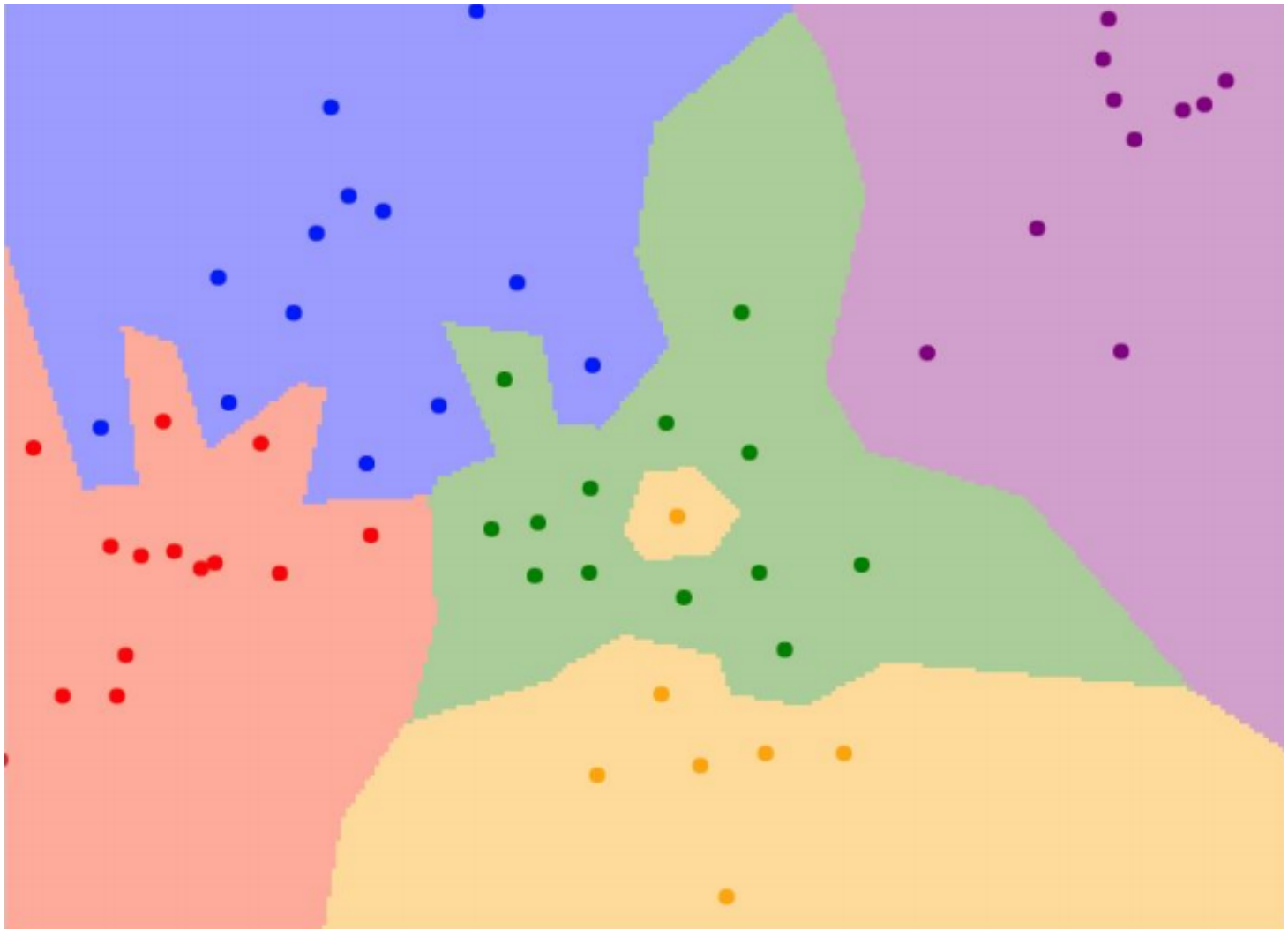
Q: With N examples, how fast are training and prediction?

Nearest Neighbor Classifier

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

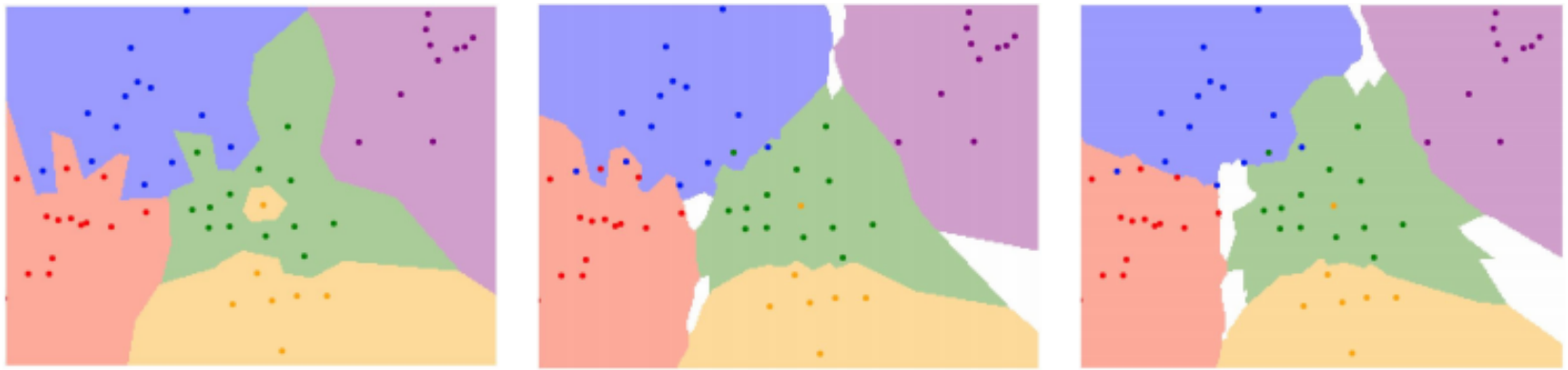
A: Train $O(1)$, predict $O(N)$

What Does This Look Like?



K-Nearest Neighbors

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



Hyperparamters

What is the best value of **K** to use?
What is the best value of **distance** to use?
These are hyperparameters: choices about the algorithm that we set rather than learn.

Whiteboard

k-Nearest Neighbor on images never used

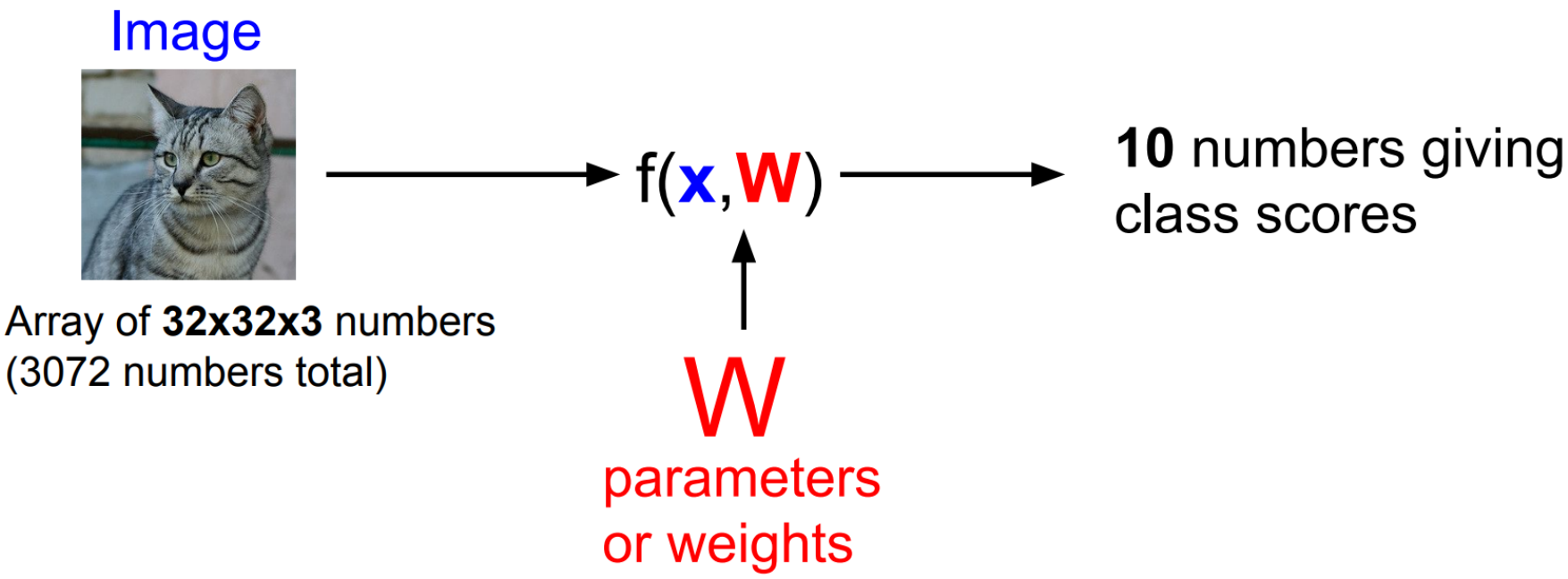
- Very slow at test time
- Distance metric on pixels are not informative

Linear Classification

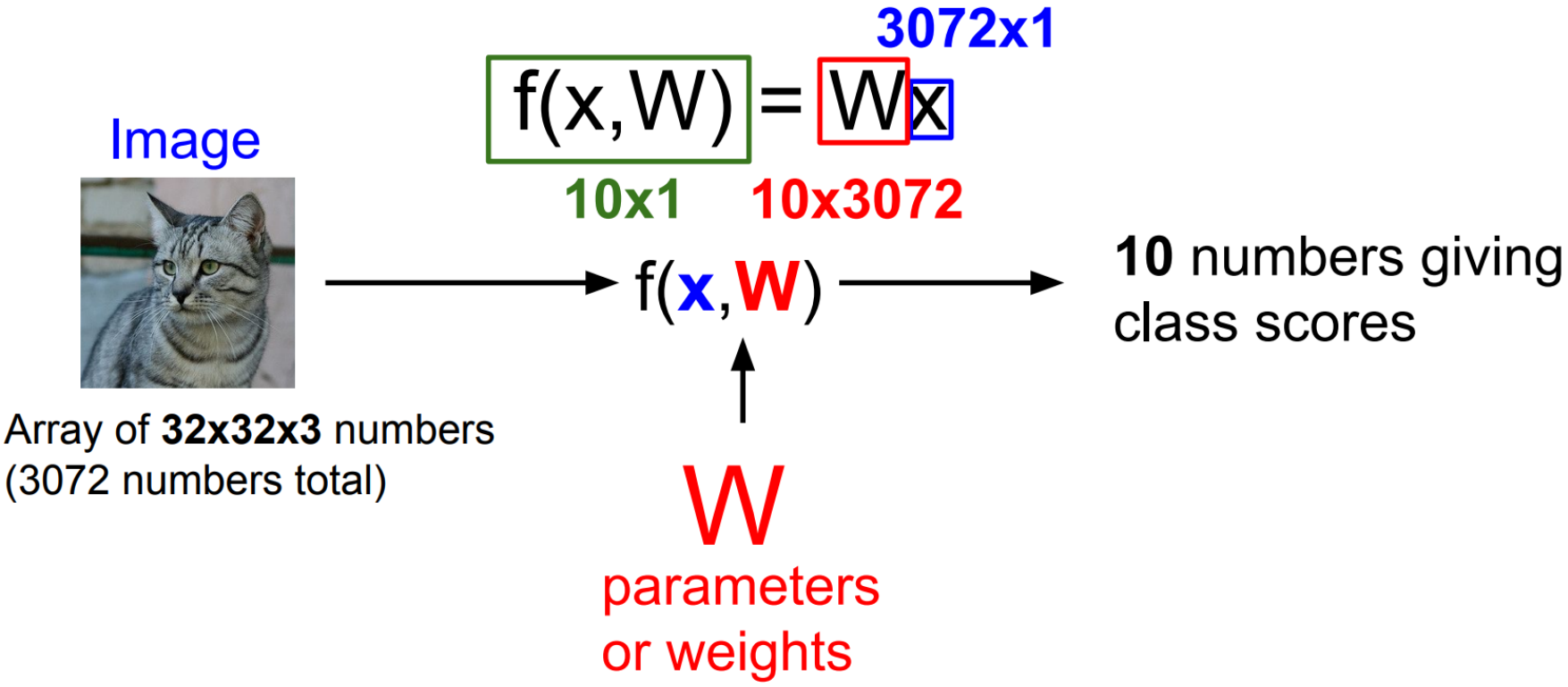
Neural Network

Linear classifiers

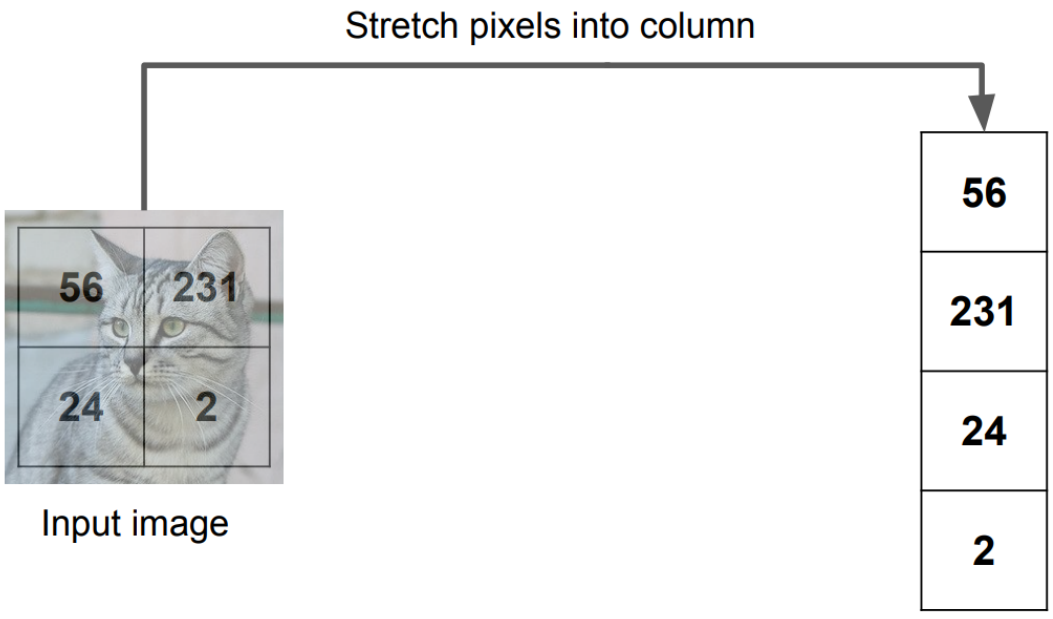
Parametric Approach



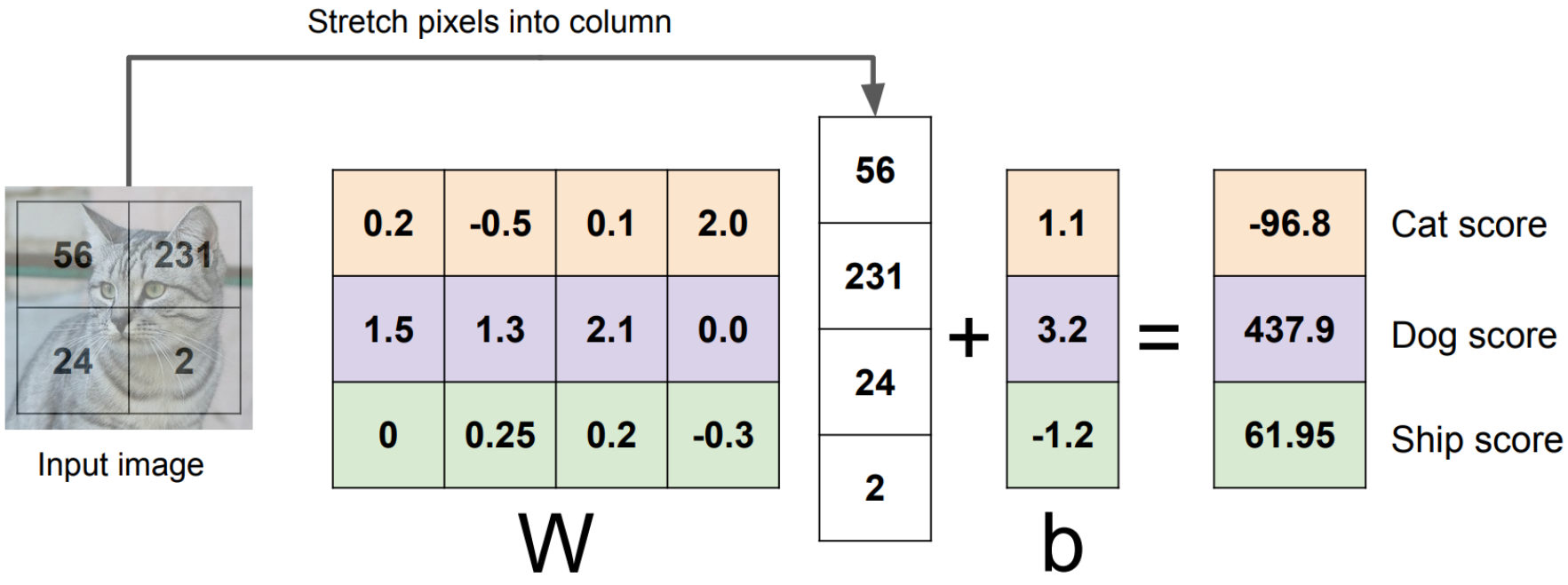
Parametric Approach: Linear Classifier



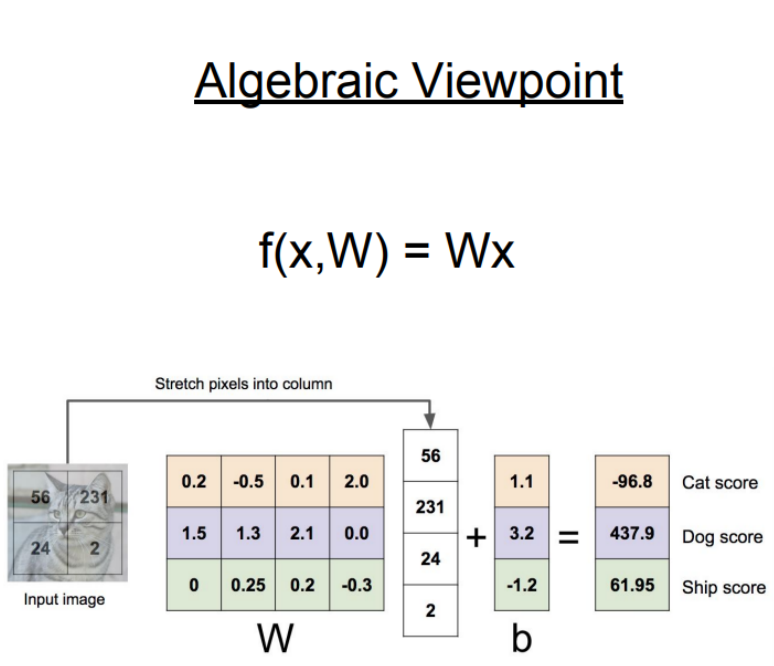
Example with an image with 4 pixels, and 3 classes

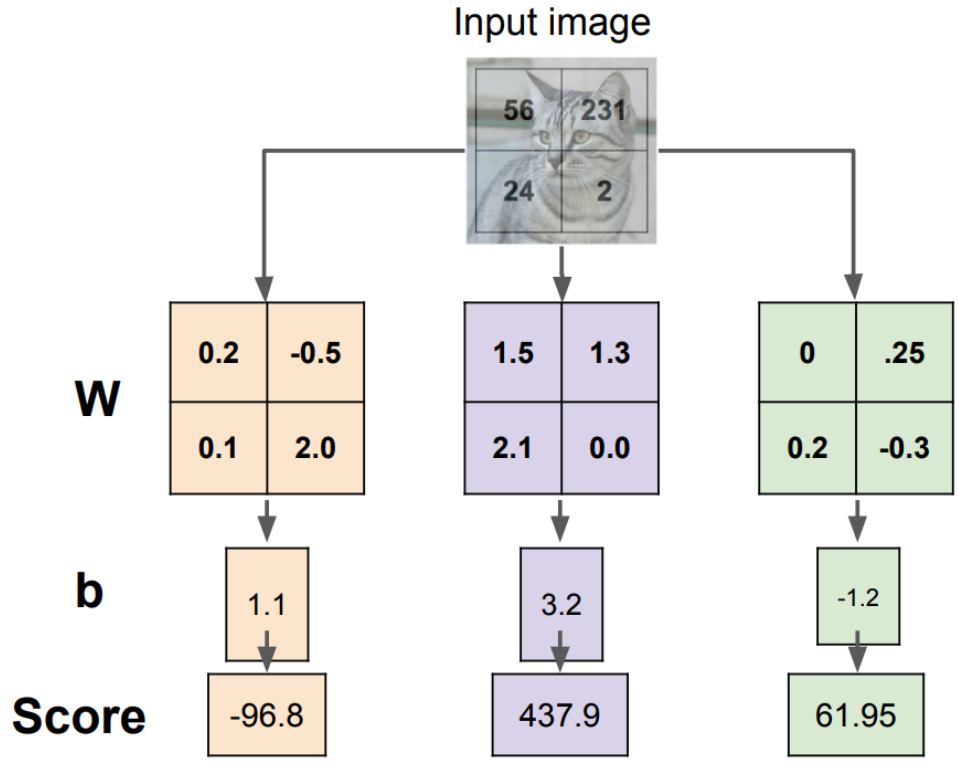


Example with an image with 4 pixels, and 3 classes



Interpreting a Linear Classifier





Example

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**

Example

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**

- How bad is this W?
- There is a best W?

Loss Function

Loss Function

A loss function tells how good our current classifier is
Given a dataset of examples $\{(x_i, y_i)\}_{i=0}^{N}$ where x_i is a image and y_i is a label. Loss of the data set is a sum of loss over examples:

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

Multiclass SVM Loss

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

$$L_i = \sum_{j \neq y_i} \max\left(0, f(x_j, W) - f(x_{y_i}, W) + 1\right)$$

Multiclass SVM Loss: Example (Cat)



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

$$
\begin{aligned}
L_i &= \sum_{j \neq y_i} \max\left(0, f(x_j, W) - f(x_{y_i}, W) + 1\right) \\
&= \max\left(0, 5.1 - 3.2 + 1\right) + \max\left(0, -1.7 - 3.2 + 1\right) \\
&= \max\left(0, 2.9\right) + \max\left(0, -3.9\right) \\
&= 2.9 + 0 \\
&= 2.9
\end{aligned}
$$

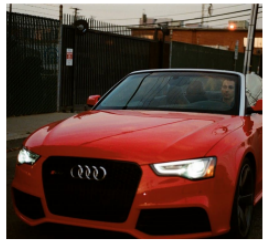
Multiclass SVM Loss: Example (Car)



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

$$
\begin{aligned}
L_i &= \sum_{j \neq y_i} \max\left(0, f(x_j, W) - f(x_{y_i}, W) + 1\right) \\
&= \max\left(0, 1.3 - 4.9 + 1\right) + \max\left(0, 2.0 - 4.9 + 1\right) \\
&= \max\left(0, -2.6\right) + \max\left(0, -1.9\right) \\
&= 0 + 0 \\
&= 0
\end{aligned}
$$

Multiclass SVM Loss: Example (Frog)


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

$$
\begin{aligned}
L_i &= \sum_{j \neq y_i} \max\left(0, f(x_j, W) - f(x_{y_i}, W) + 1\right) \\
&= \max\left(0, 2.2 - (-3.1) + 1\right) + \max\left(0, 2.5 - (-3.1) + 1\right) \\
&= \max\left(0, 6.3\right) + \max\left(0, 6.6\right) \\
&= 6.3 + 6.6 \\
&= 12.9
\end{aligned}
$$

Multiclass SVM Loss: Example



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

$$L = \frac{1}{N}\sum_{i=0}^{N} L_i$$

$$L = \frac{1}{3}(2.9 + 0 + 12.9)$$

$$L = \mathbf{5.27}$$

Loss function

Q: What happens to loss if car scores change a bit?

Loss function

Q: What happens to loss if car scores change a bit?

Q: what is the min/max possible loss?

Loss function

Q: What happens to loss if car scores change a bit?

Q: what is the min/max possible loss?

Q: At initialization W is small so all $s \approx 0$ What is the loss?

Loss function

Q: What happens to loss if car scores change a bit?

Q: what is the min/max possible loss?

Q: At initialization W is small so all $s \approx 0$ What is the loss?

Q: What if the sum was over all classes?

Loss function

Q: What happens to loss if car scores change a bit?

Q: what is the min/max possible loss?

Q: At initialization W is small so all $s \approx 0$ What is the loss?

Q: What if the sum was over all classes?

Q: What if we used mean instead of sum?

Loss function

Q: What happens to loss if car scores change a bit?

Q: what is the min/max possible loss?

Q: At initialization W is small so all $s \approx 0$ What is the loss?

Q: What if the sum was over all classes?

Q: What if we used mean instead of sum?

Q: What if we used
$L_i = \sum_{j \neq y_i} \max(0, f(x_j, W) - f(x_{y_i}, W) + 1)^2$

Loss function

Q: What happens to loss if car scores change a bit?

Q: what is the min/max possible loss?

Q: At initialization W is small so all $s \approx 0$ What is the loss?

Q: What if the sum was over all classes?

Q: What if we used mean instead of sum?

Q: What if we used
$L_i = \sum_{j \neq y_i} \max(0, f(x_j, W) - f(x_{y_i}, W) + 1)^2$

Q: Suppose that we found a W such that $L = 0$. Is this W unique?

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}}$$

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Data Loss: Model prediction should match training data

Regularization: Prevent the model from doing too well on training data

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Data Loss: Model prediction should match training data
Regularization: Prevent the model from doing too well on training data

Why?

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Data Loss: Model prediction should match training data
Regularization: Prevent the model from doing too well on training data

Why?
Occam's razor

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Where $\lambda =$ regularization strength (hyperparameter)

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Where λ = regularization strength (hyperparameter)

In common use:

- L_2 **regularization**: $R(W) = \sum_k \sum_l W_{k,l}^2$
- L_1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$
- Elastic net ($L_1 + L_2$): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$
- Max norm regularization
- Dropout.
- Fancier: Batch normalization, Stochastic depth, etc.

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Why?

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=0}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data Loss}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}}$$

Why?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

Regularization: Expressing Preferences

$$x = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$
$$w_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$
$$w_2 = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

Now

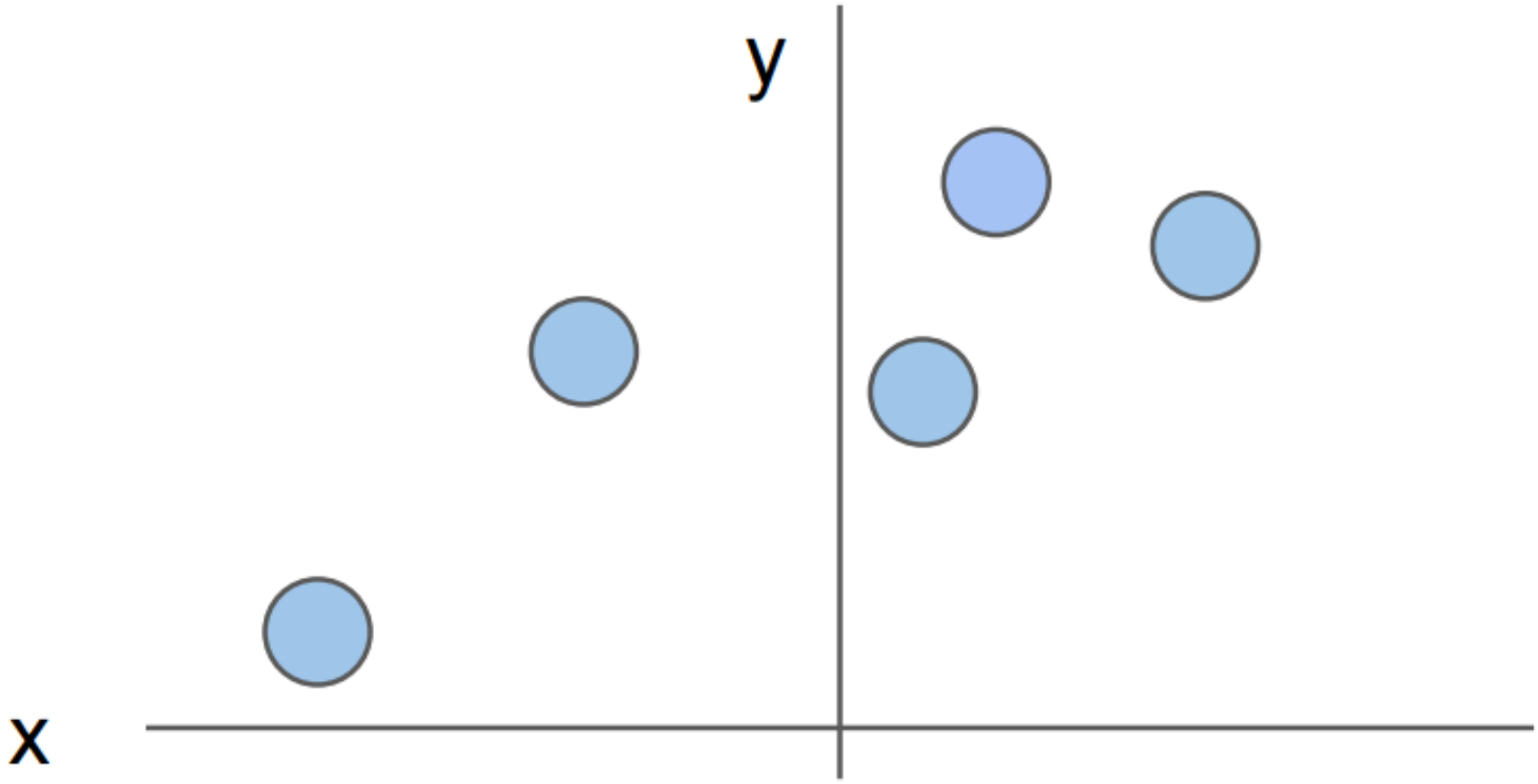
$$w_1^T x = w_2^T x = 1$$

But L_2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

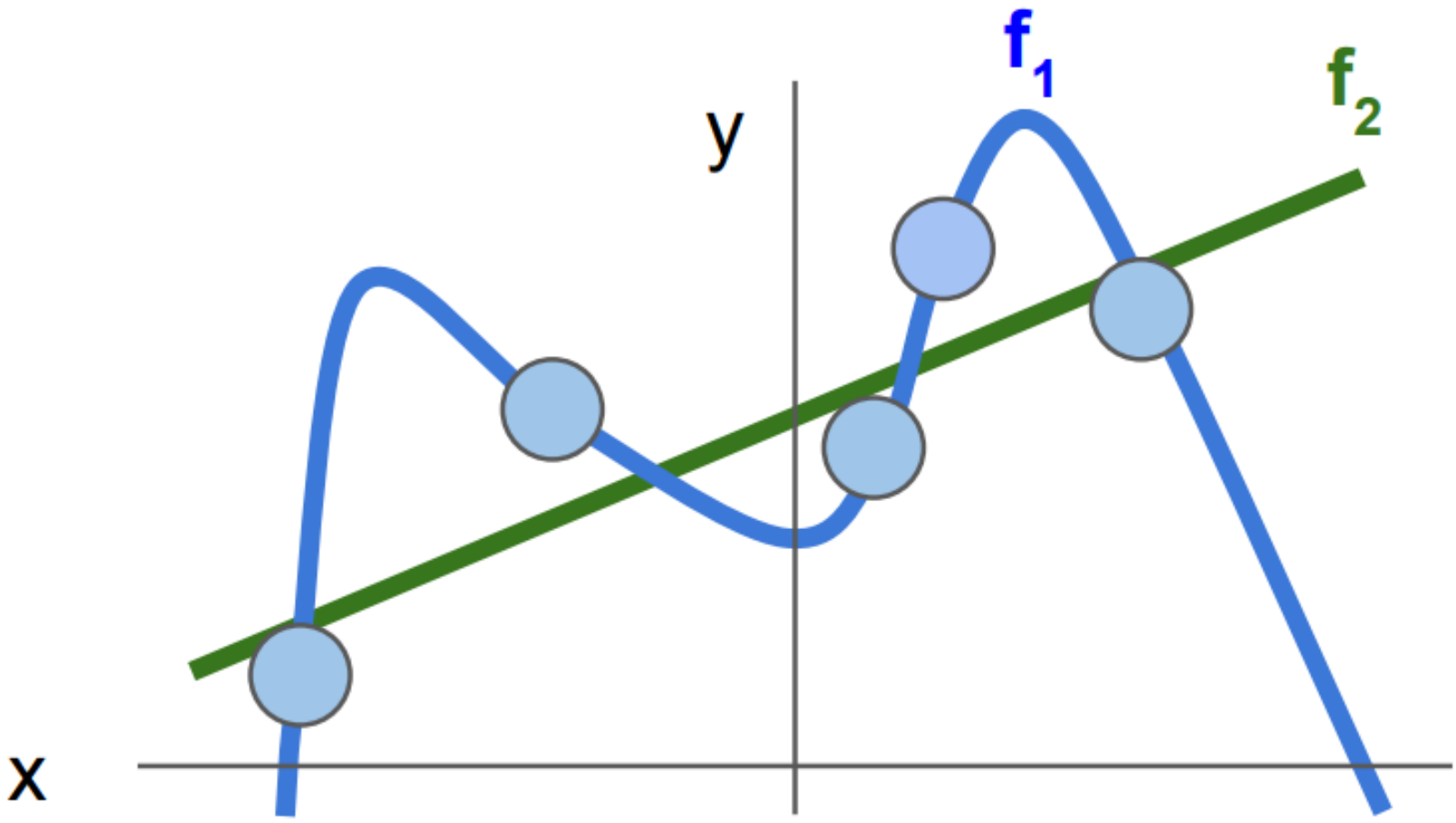
$$R(w_1) = 1 \quad R(w_2) = 0.25$$

L_2 regularization likes to "spread out" the weights

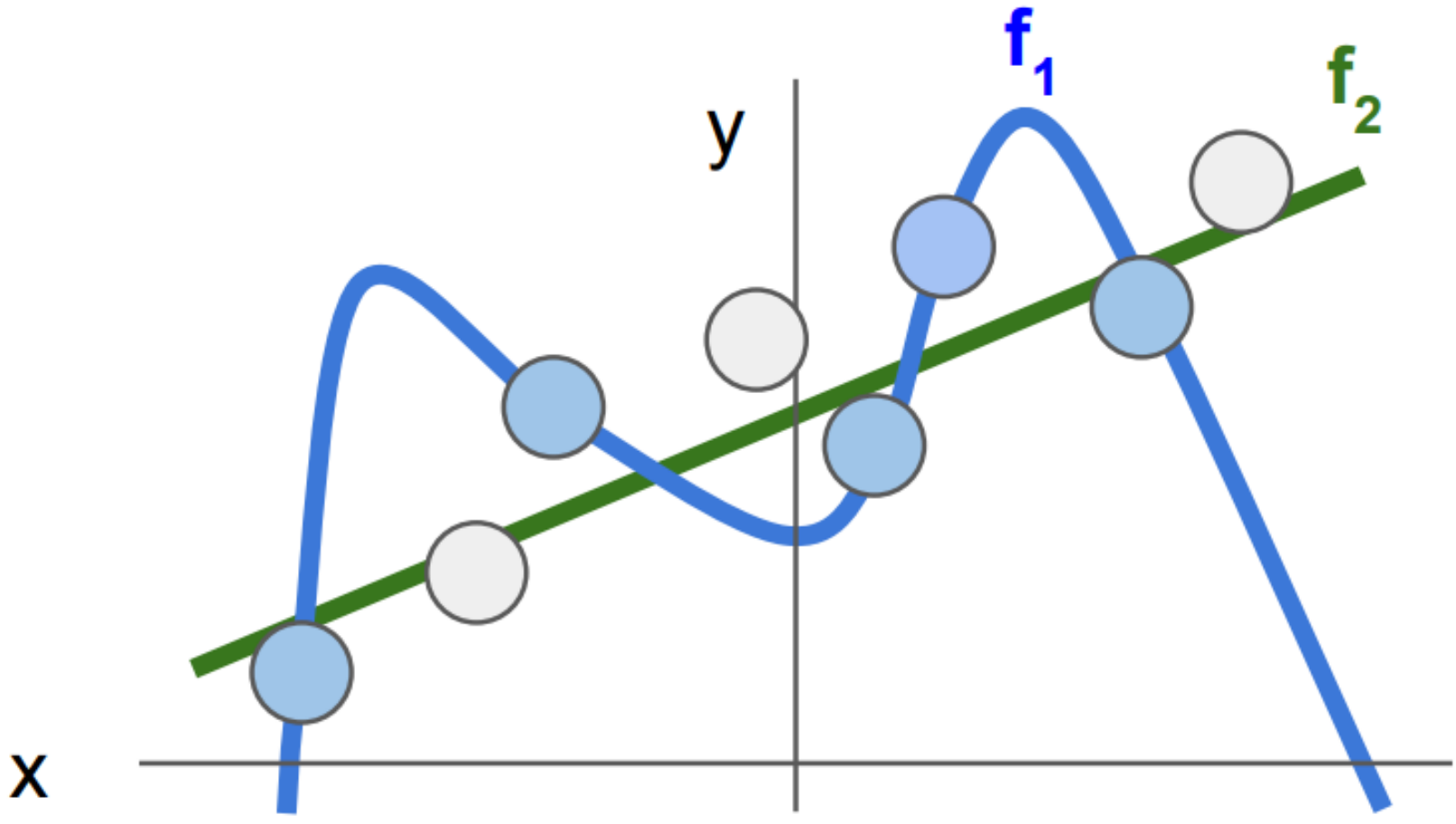
Regularization: Prefer Simpler Models



Regularization: Prefer Simpler Models



Regularization: Prefer Simpler Models



How do we actually find this W that minimize the loss?

Next meeting.