Image Classification:
K-NN and Linear Classifier

Xiaolong Wang

Today: Two basic methods

- Nearest Neighbors

- Linear Classifier

Image Classification

An image is a 300 x 500 x 3 Tensor.

Each bit has value in the range [0, 255]

Images with different background

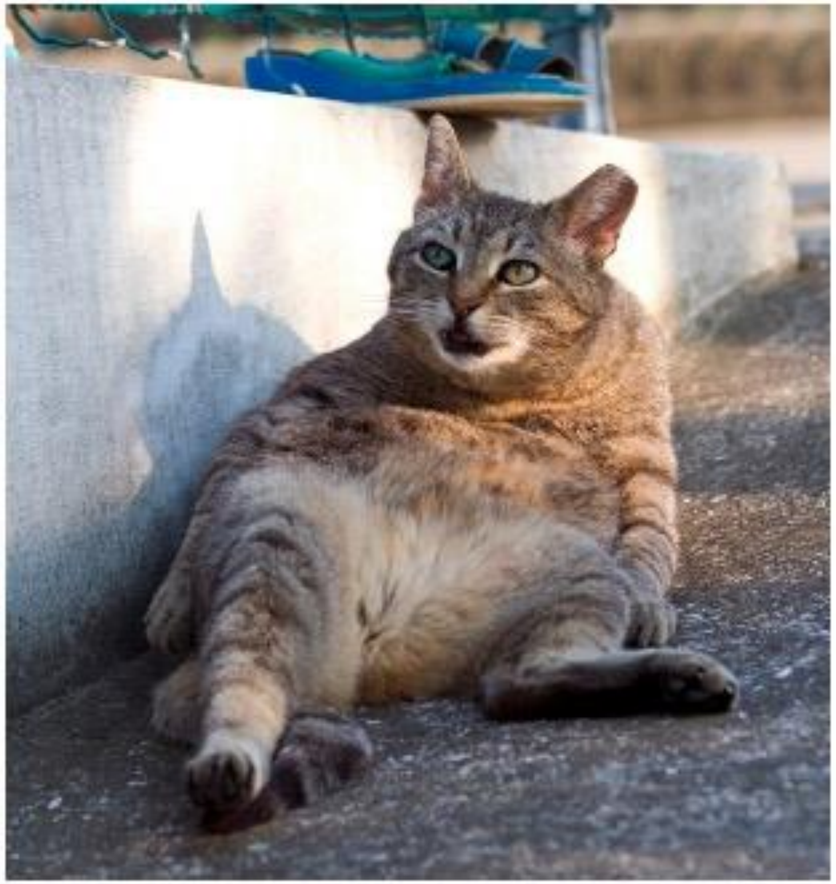
http://cs231n.stanford.edu/

Images with occlusion

Images with illumination



Images with Deformation



Nearest Neighbor Classifier

Nearest Neighbor

Training set:


Mushroom


Dog


Ant


Cat


Car

Testing: Compute the distance between a test image and training images

| ,  | $\rightarrow \mathbb{R}$

| ,  | $\rightarrow \mathbb{R}$

Nearest Neighbor

- What metric? What representation?
- Metric, L1 distance:

$$d(x_1, x_2) = \sum_{h,w} \left| x_1^{h,w} - x_2^{h,w} \right|$$

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

Recall Supervised Learning

$$y = f(x)$$

output label

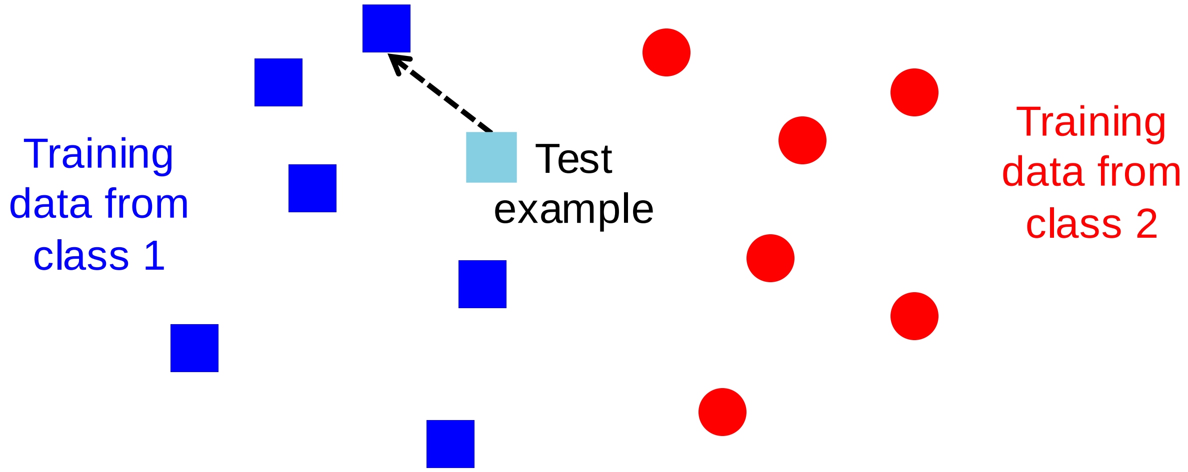
classifier

input image

- **Training** (or **learning**): given a *training set* of labeled examples $\{(x_1, y_1), \dots, (x_N, y_N)\}$, train a predictor f

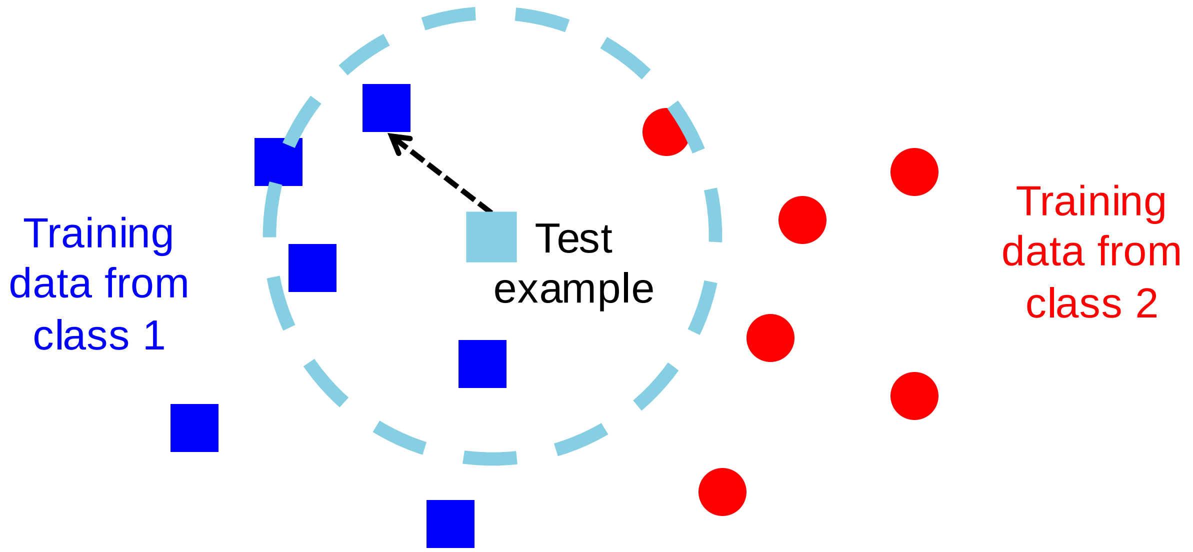
- **Testing** (or **inference**): apply predictor f to a new *test example* x and output the predicted value $y = f(x)$

Nearest neighbor classifier



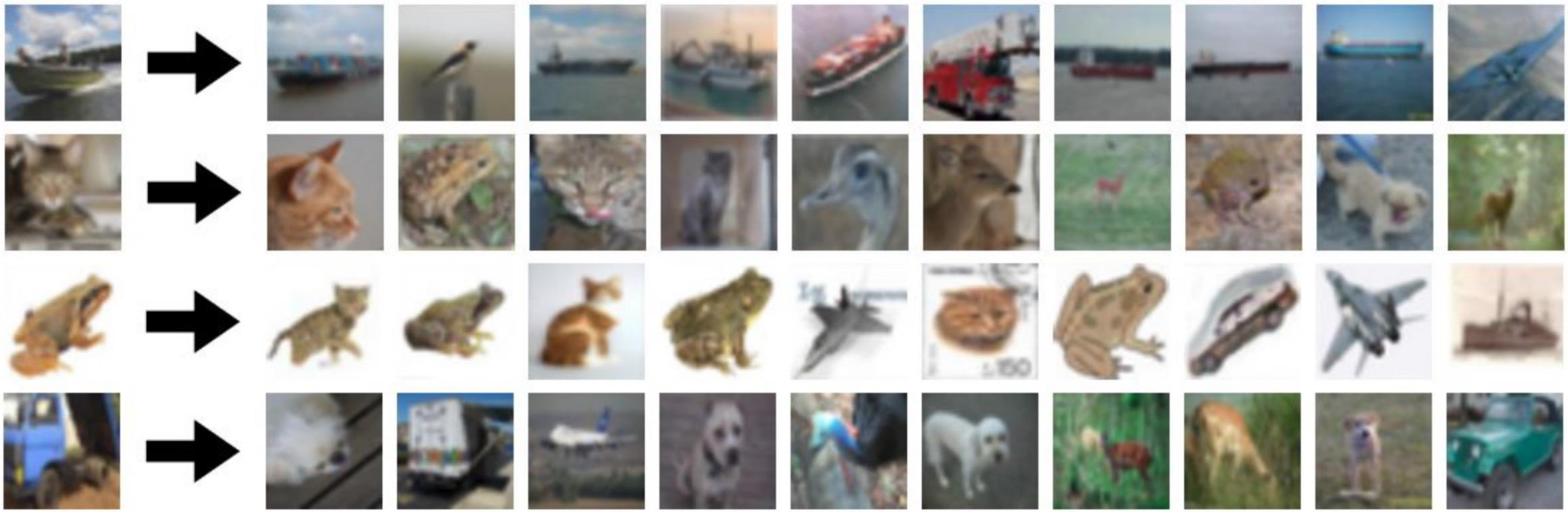
- $f(x)$ = the label of the closest example (computed via a distance metric)
- Store all the training data, search all data each test time given a test example

K-nearest neighbor classifier



- 1 example is sometimes not enough.
- K-NN, K=5: Find closest 5 examples instead of 1. Follow the label of the majority in the NN examples.

K-NN examples (K=10), based on pixel-wise difference



Goods and Bads of Nearest Neighbor

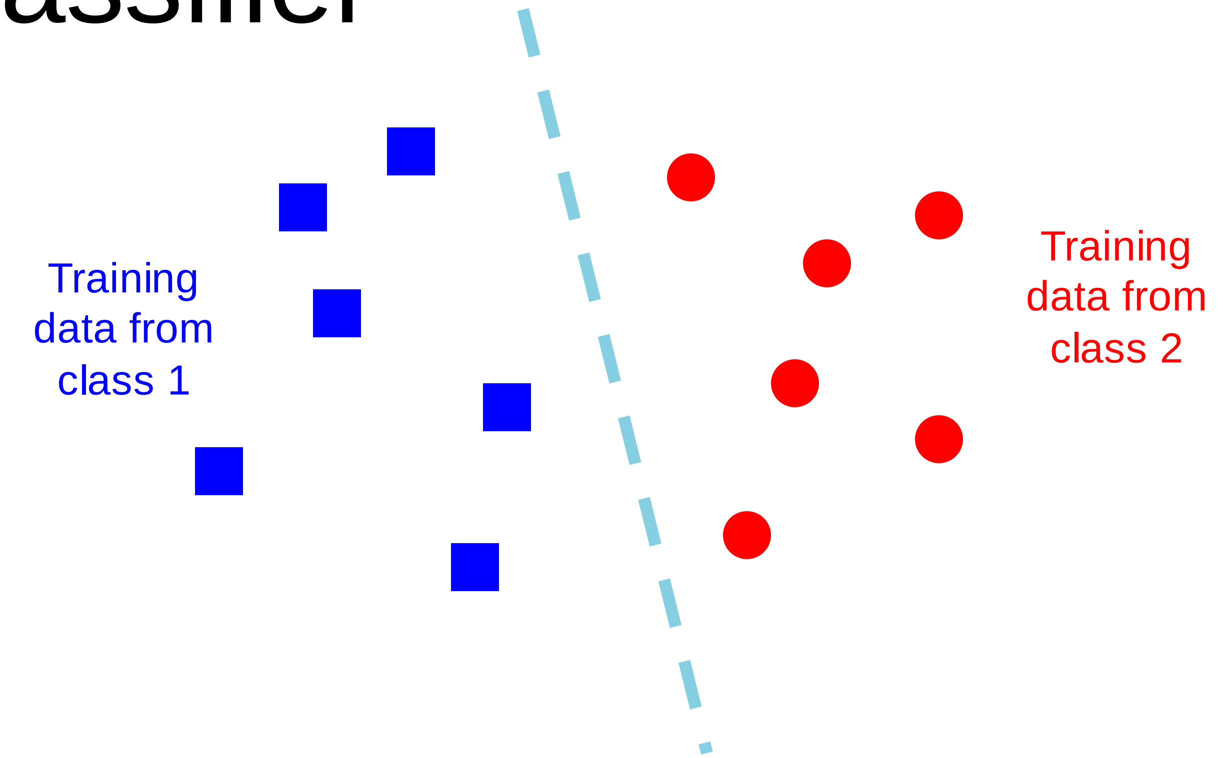
- Good:
  - Do not require training
  - Simple and robust to outliers

- Bad:
  - Storage: needs to store the whole dataset
  - Time: needs to go over each training data point, inference time grows linearly as the training data increases

- Can we *compress* the training samples to a set of weights?

Linear Classifier

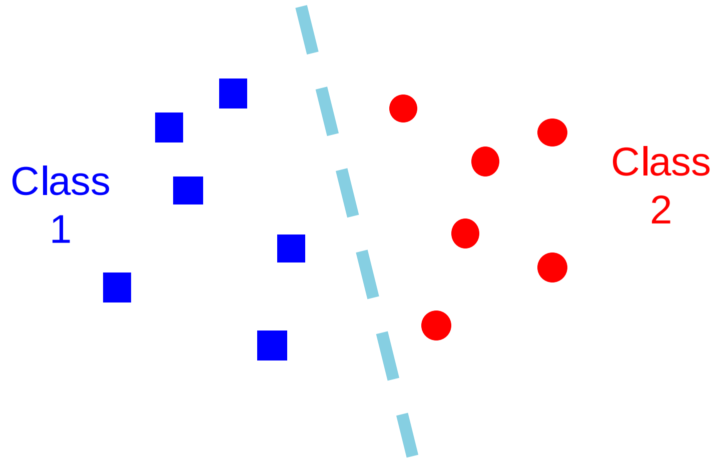
Linear Classifier



- Goal: Learn a d-dimentional vector of parameters $W \in \mathbb{R}^d$, given a set of d-dimentional data

- Prediction: $f(x) = W_1 x_1 + W_2 x_2 + \ ... + W_d x_d = Wx$

Linear Classifier

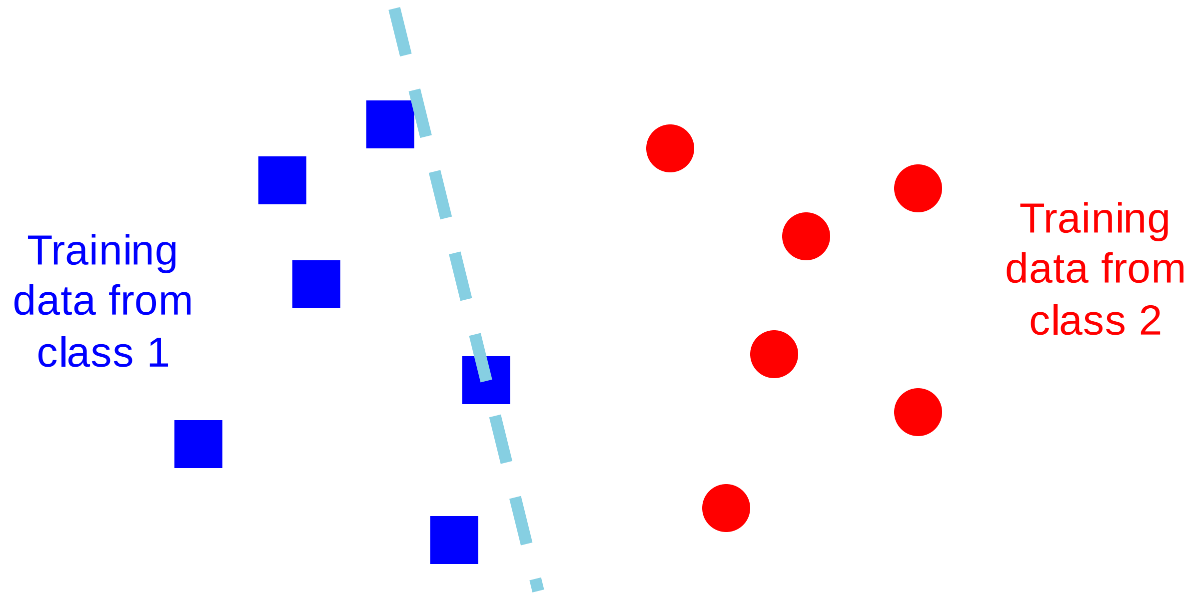


- Prediction: $f(x) = W_1 x_1 + W_2 x_2 + \ldots + W_d x_d = Wx$

- If $f(x) > 0$, x belongs to class 1, if $f(x) < 0$, x belongs to class 2.

- See W as the compression of the whole training dataset, and we only need to compute 1 multiplication for obtaining the label.

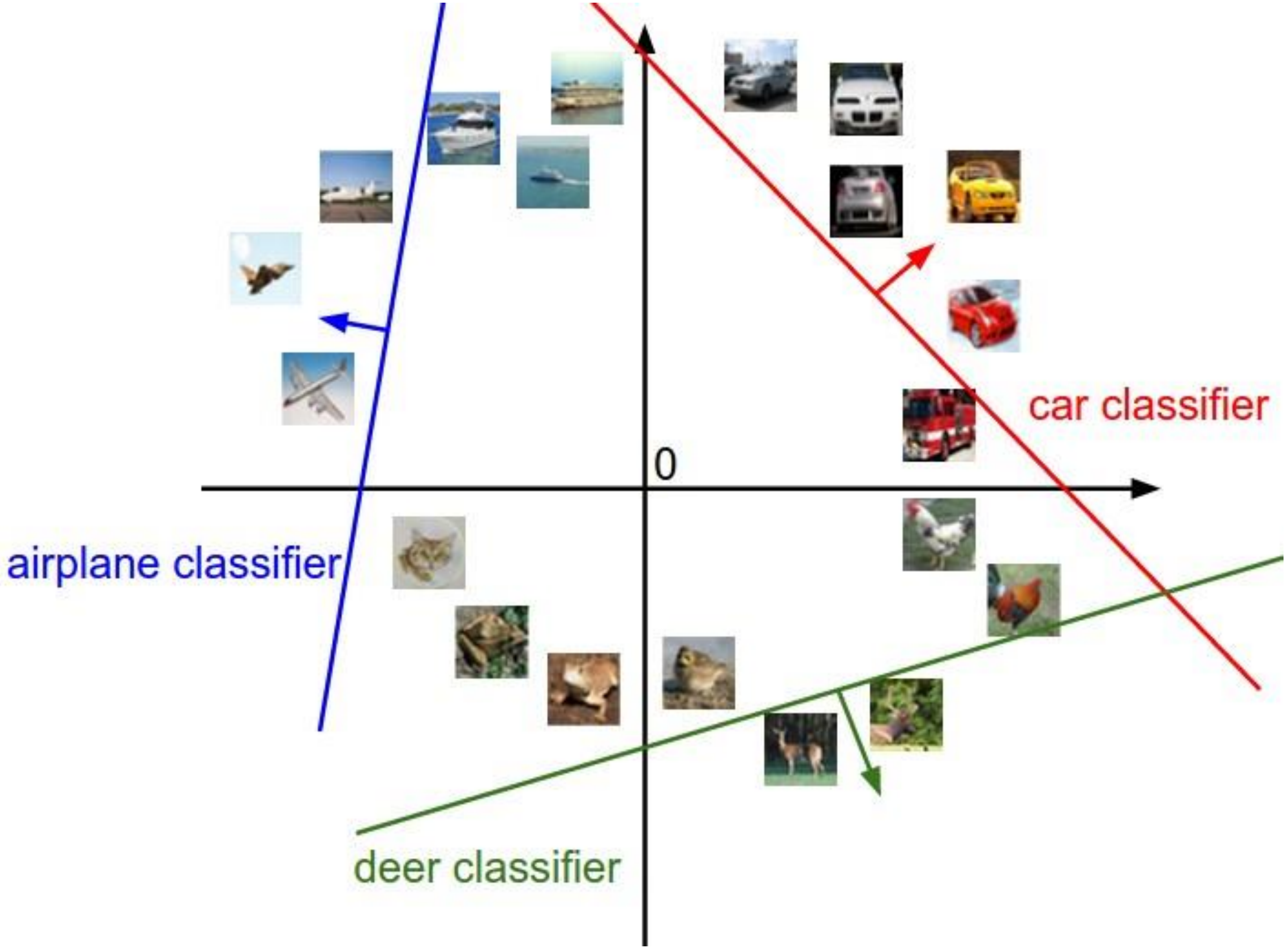
Linear Classifier: adding bias



- Prediction: $f(x) = W_1 x_1 + W_2 x_2 + \ \ldots + W_d x_d + b = Wx + b$
- $b \in \mathbb{R}^1$, b is only a 1-dimentional digit for 2-class classification

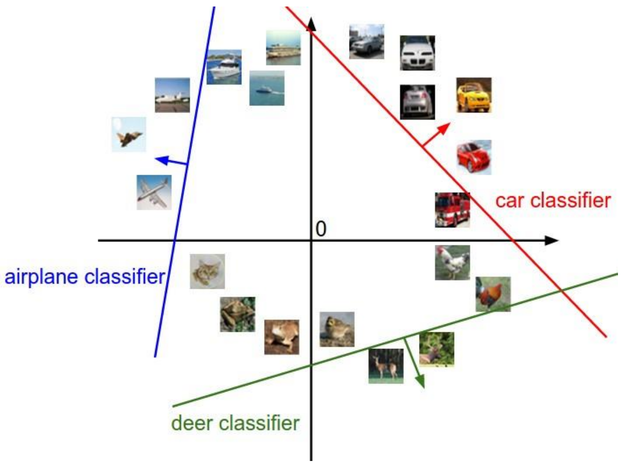
Linear Classifier: Multiple Class

- 1 plane is not enough
- Multiple planes



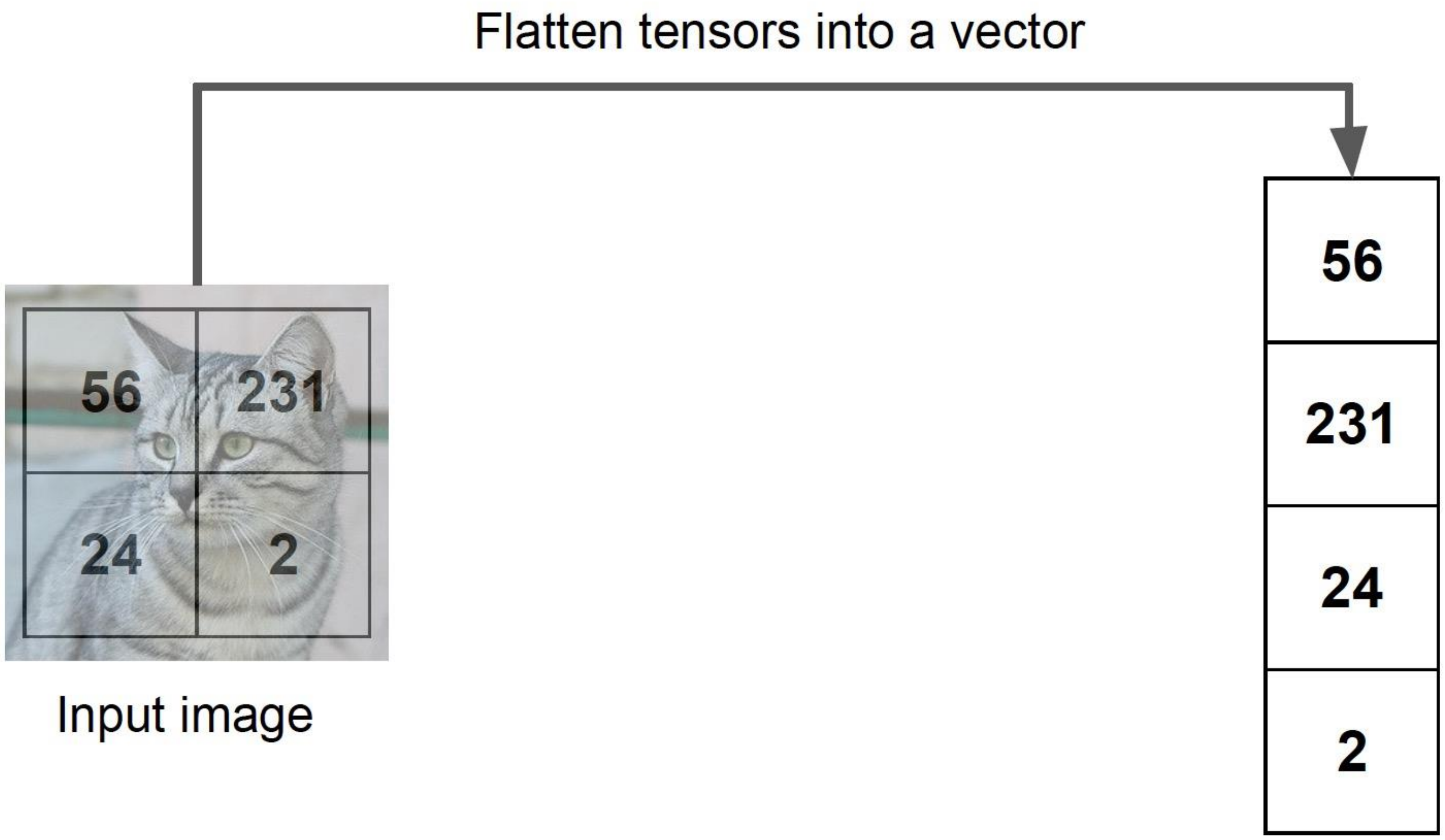
Source: Andrej Karpathy, http://cs231n.github.io/linear-classify/

Linear Classifier: Multiple Class

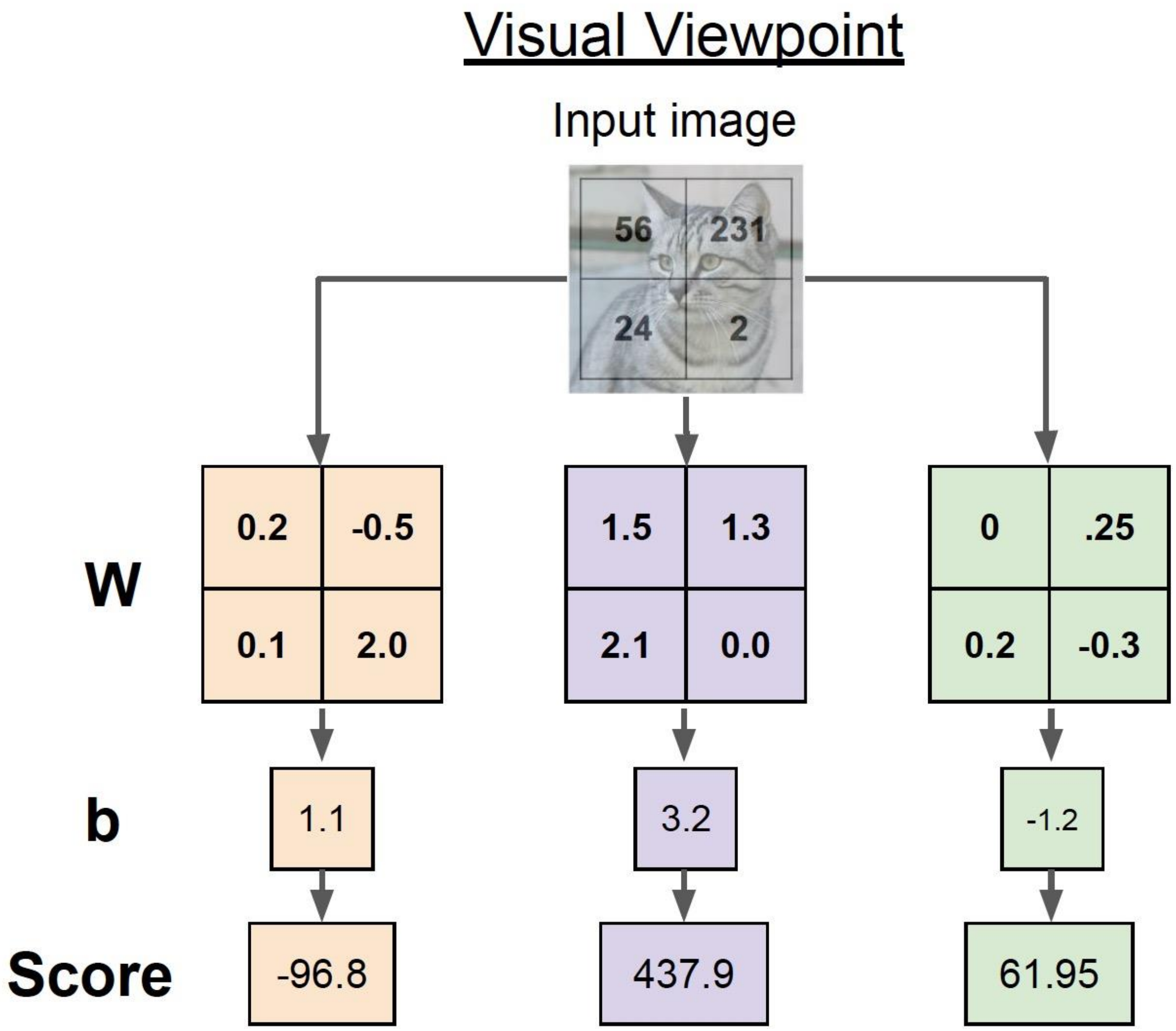


- Instead of learning one vector of weights, we will need to learn one vector of weights for each category:
  - A dog classifier: $f_1(x) = W^1x + b^1$
  - A cat classifier: $f_2(x) = W^2x + b^2$
  - A ship classifier: $f_3(x) = W^3x + b^3$
- Select the class with the max classification score

Example: Represent an image with 4 pixels



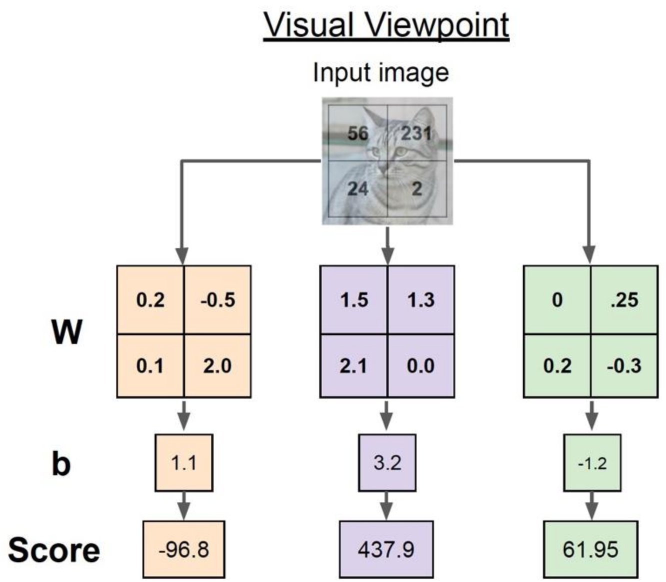
Example: Represent an image with 4 pixels



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

$$x \in \mathbb{R}^{3072}\ (32 \times 32 \times 3)$$
$$W \in \mathbb{R}^{3072}$$
$$b \in \mathbb{R}^{1}$$

Example: Represent an image with 4 pixels



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

$$x \in \mathbb{R}^{3072} \ (32 \times 32 \times 3)$$
$$W \in \mathbb{R}^{3072}$$
$$b \in \mathbb{R}^{1}$$

Visualizing W in 10 different classes:

Training the Linear Classifier

- Linear regression
- Logistic regression

Training with Linear Regression

- Given the training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, drawn from distribution D.

- Find predictor $f(x)$ so that it performs well on test (unseen) data drawn from the same distribution D.

- Potential problem: What if the data is not taken from the same distribution D?

How to evaluate “performs well”?

- Define an expected loss as,

$$\mathbb{E}_{(x,y)\backsim D}[l(f,x,y)]$$

- To approximate the loss using N examples $\{(x_1,y_1),\dots,(x_N,y_N)\}$,

$$\frac{1}{N}\sum_{i=1}^{N} l(f,x_i,y_i)$$

Linear Regression

- Loss: Using L2 distance:

$$l(f, x_i, y_i) = (f(x_i) - y_i)^2 = (Wx_i + b - y_i)^2$$

- Average through all the examples

$$\frac{1}{N}\sum_{i=1}^{N}(Wx_i + b - y_i)^2$$

Linear Regression

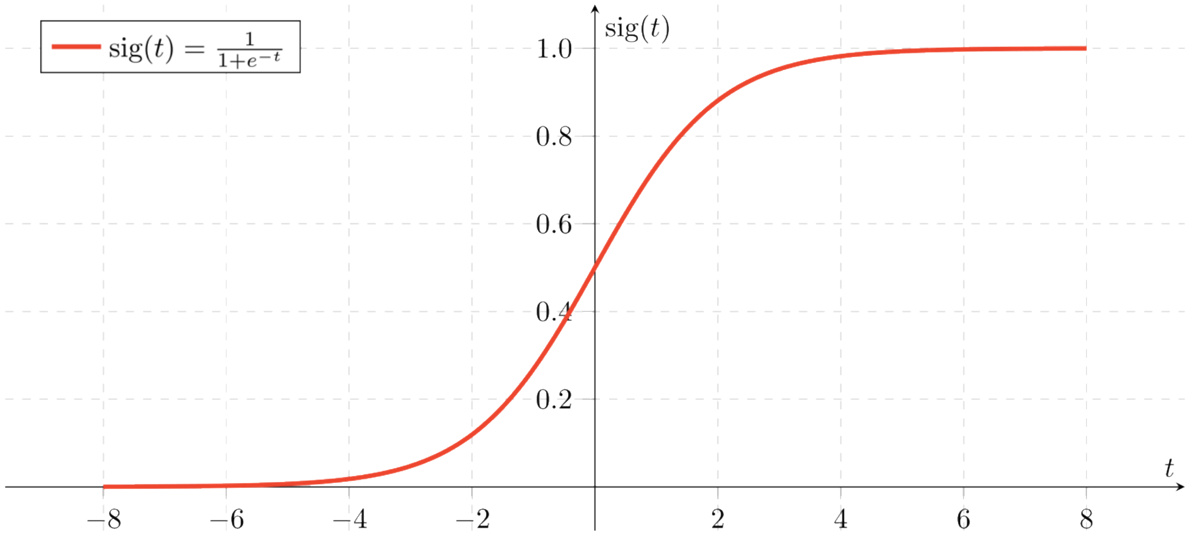
$$\frac{1}{N}\sum_{i=1}^{N}(Wx_i + b - y_i)^2$$

- In two-class classification: $y \in \{-1,1\}$. However, there is no regulation to constrain the output range.
- In multiple-class case, for each class we perform two-class classification: $y \in \{-1,1\}$.
- Not convenient for classification

The Sigmoid Function (2-class)

- Squash the linear response of the classifier to the interval [0,1] to represent the prediction probability:

$$\sigma(Wx) = \frac{1}{1+\exp(-Wx)}$$



The Sigmoid Function (2-class)

- Thus we let $P(y = 1|x) = \sigma(Wx) = \frac{1}{1+\exp(-Wx)}$

- For the other category:

$$P(y = -1|x) = 1 - P(y = 1|x) = 1 - \sigma(Wx)$$

$$= 1 - \frac{1}{1 + \exp(-Wx)} = \frac{\exp(-Wx)}{1 + \exp(-Wx)}$$

$$= \frac{1}{\exp(Wx) + 1} = \sigma(-Wx)$$

The sigmoid function is *symmetric:* $1 - \sigma(Wx) = \sigma(-Wx)$

Logistic regression: Training Objective

- Given: $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1,1\}$

$$\hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|x_i)$$

$$= -\frac{1}{N}\sum_{i:y_i=1} \log \sigma(Wx_i) - \frac{1}{N}\sum_{i:y_i=-1} \log[1 - \sigma(Wx_i)]$$

$$= -\frac{1}{N}\sum_{i:y_i=1} \log \sigma(Wx_i) - \frac{1}{N}\sum_{i:y_i=-1} \log[\sigma(-Wx_i)]$$

$$= -\frac{1}{N}\sum_{i} \log \sigma(y_i W x_i)$$

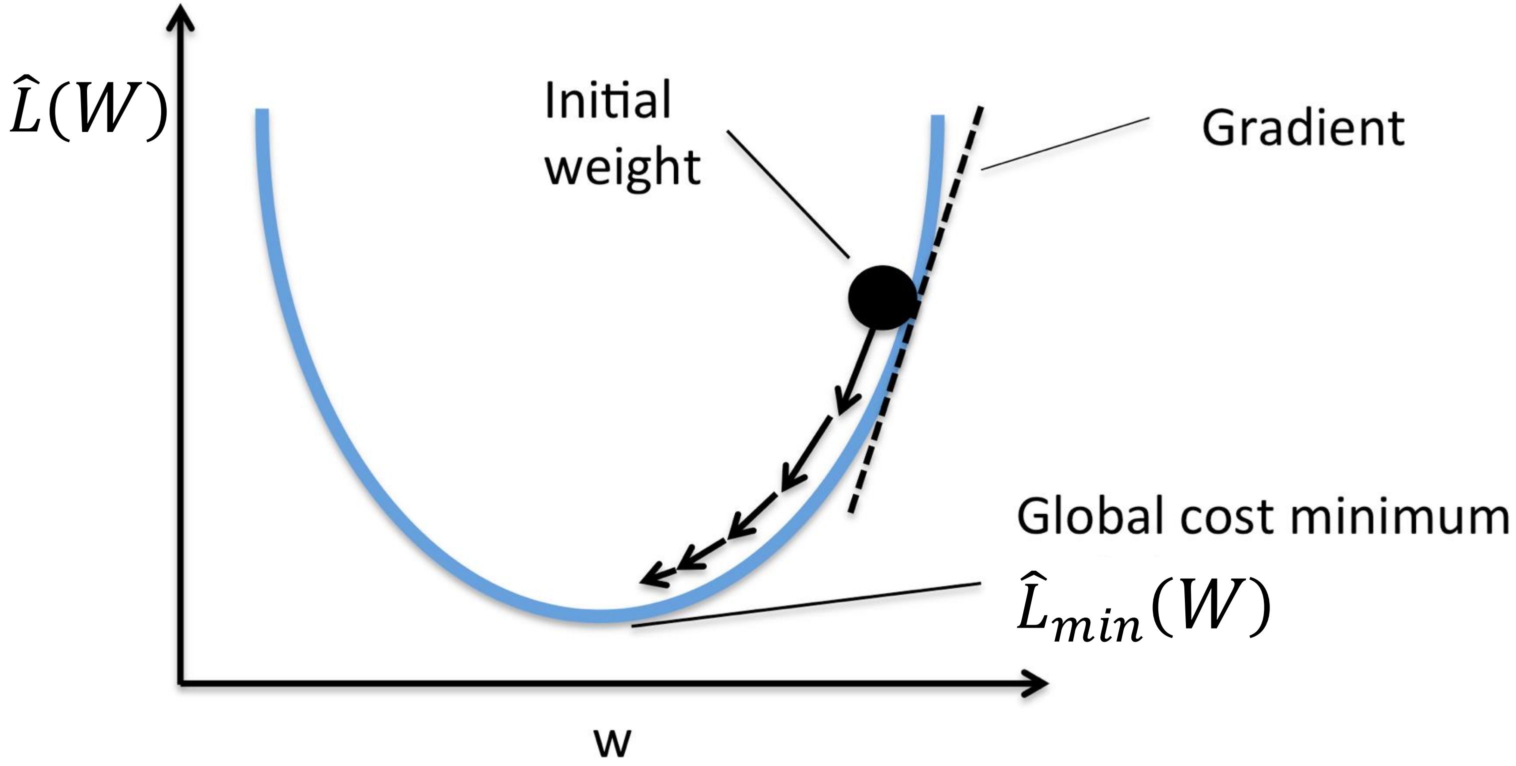
Optimization

Gradient descent

- Start with some initial estimate of W.

- At each step, compute the gradient $\nabla \hat{L}(W)$.

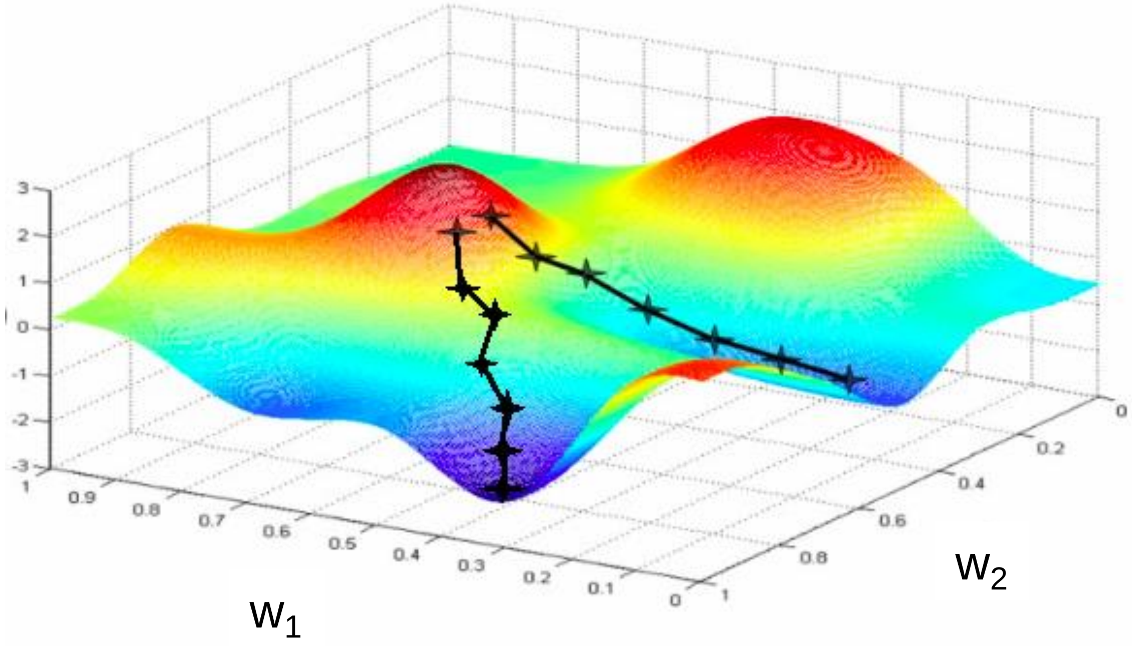
- Move in the opposite direction of the gradient



2D Example

Take a small step in the *opposite* direction, using learning rate α:

$$W \leftarrow W - \alpha \nabla \hat{L}(W)$$



Source: Svetlana Lazebnik

Gradients for linear regression

$$\hat{L}(W) = \frac{1}{N}\sum_{i=1}^{N}(Wx_i - y_i)^2$$

Compute the gradient:

$$\nabla\hat{L}(W) = \frac{1}{N}\sum_{i=1}^{N}\nabla_W(Wx_i - y_i)^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}2(Wx_i - y_i)\,x_i$$

Gradients for linear regression

Update rule:

$$W \leftarrow W - \alpha\, \nabla \hat{L}(W)$$

$$\nabla \hat{L}(W) = \frac{1}{N} \sum_{i=1}^{N} 2(W x_i - y_i)\, x_i$$

Combine both:

$$W \leftarrow W - \alpha \frac{1}{N} \sum_{i=1}^{N} 2(W x_i - y_i)\, x_i$$

We update the parameters iteratively, compute the gradient over all examples each gradient step

Gradient descent

$$W \leftarrow W - \alpha \, \nabla \hat{L}(W)$$

- We can set $\alpha = 0.1$ or other smaller number if the parameters diverge.

- However, it might be too slow to perform one update by calculating the gradients over all the training examples.

- Can we approximate the gradients more efficiently?

Stochastic gradient descent (SGD)

- We approximate the gradient of the whole dataset $\nabla\hat{L}(W)$ by using only ONE example (x_i, y_i) as $\nabla L(W, x_i, y_i)$

- Instead of

$$W \leftarrow W + \alpha \frac{1}{N} \sum_{i=1}^{N} \sigma(-y_i W x_i) y_i x_i$$

- Use

$$W \leftarrow W + \alpha\, \sigma(-y_i W x_i) y_i x_i$$

- Since gradient on each example is unstable, it is “stochastic”

Stochastic gradient descent (SGD)

- Instead of using only one example, or the whole dataset, we can try something in between.

- Sample a batch of examples (e.g., $B = 128$ examples) to compute the gradients for update

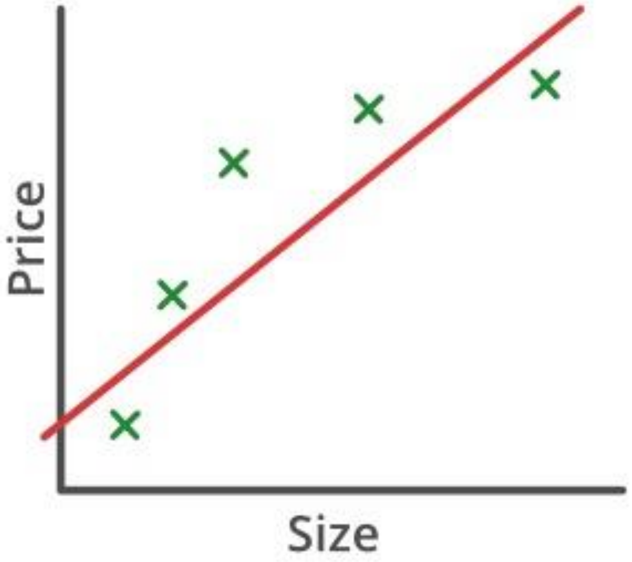
$$W \leftarrow W + \alpha \frac{1}{B} \sum_{i=1}^{B} \sigma(-y_i W x_i) y_i x_i$$

- batch size: A trade off between accurate gradient approximation and efficiency

Regularization

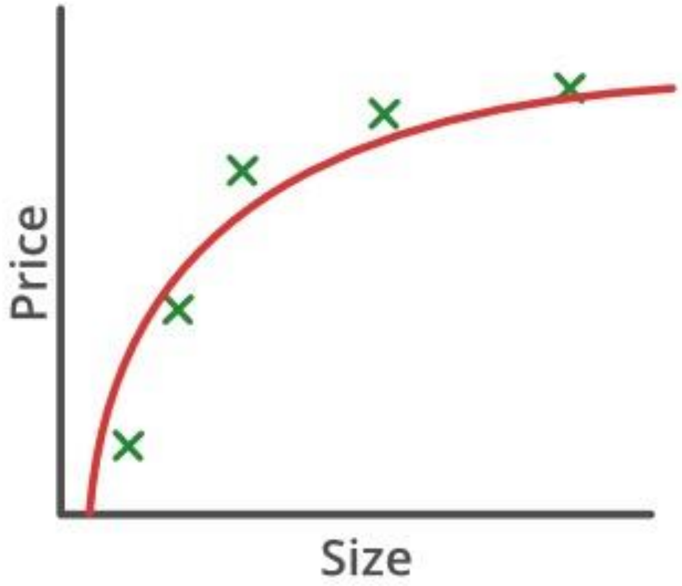
Overfitting

We want to estimate a function to fit the green data points.



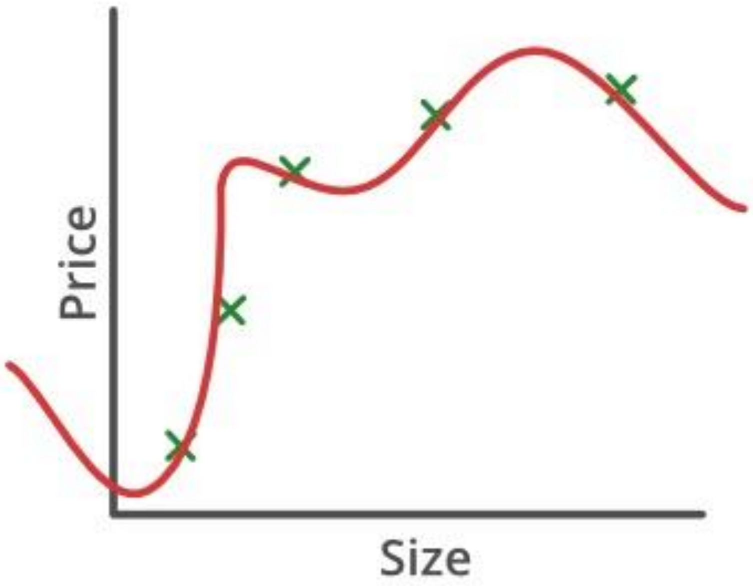
$\theta_0 + \theta_1 x$

Underfit



$\theta_0 + \theta_1 x + \theta_2 x^2$

Ideal fit

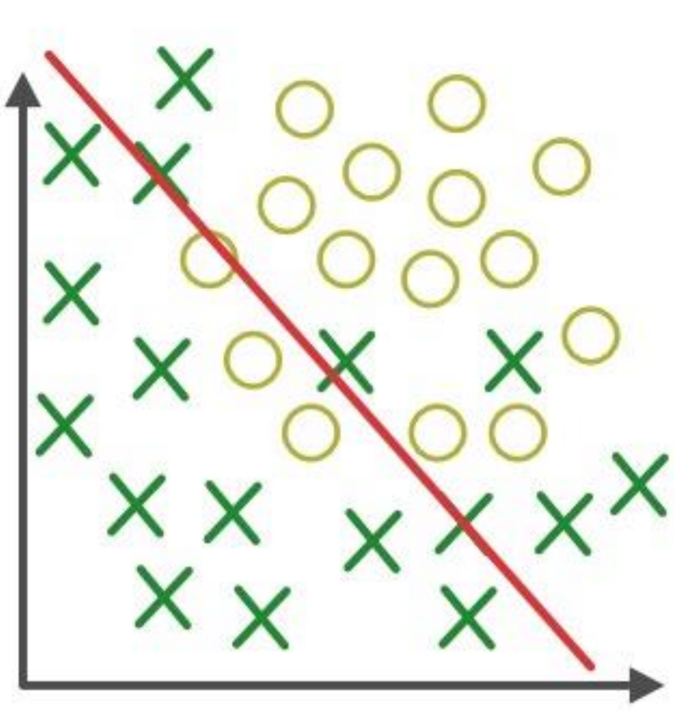


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_2 x^2 + \theta_2 x^2$

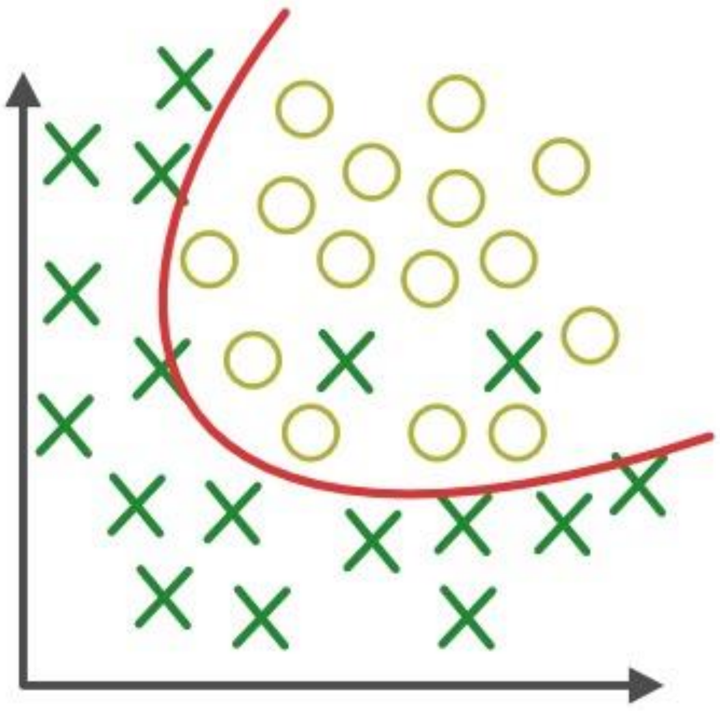
Overfit

Overfitting

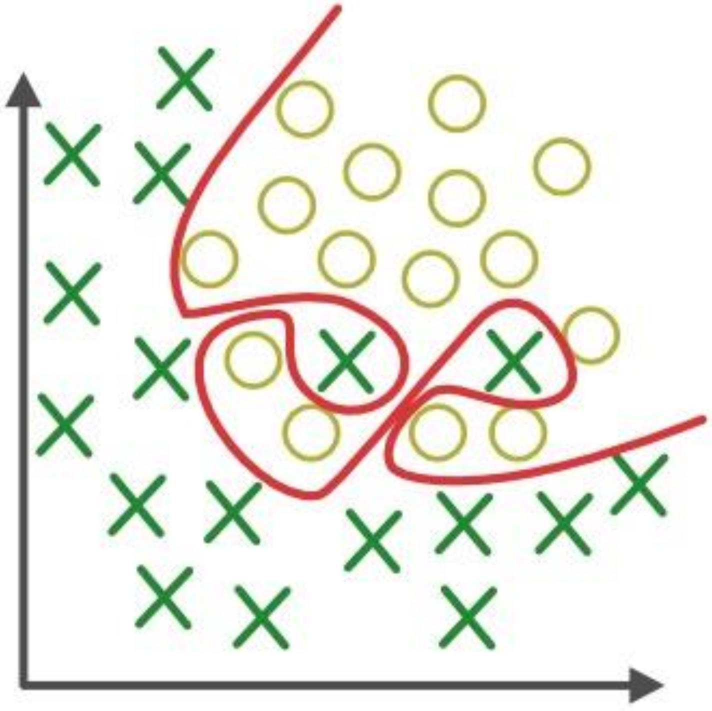
We want to estimate a classifier to separate two types of data.



Underfit



Ideal fit



Overfit

One trick to prevent overfitting

- Adding regularization in training objective, L2 regularization:

$$\hat{L}(W) = \underbrace{\frac{\lambda}{2}||W||^2}_{\text{L2 regularization}} + \underbrace{\frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)}_{\text{Loss from data}}$$

$$W \leftarrow W - \underline{\alpha\, (\lambda\, W} + \nabla_W \frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)\,)$$

To prevent overfitting

$$W \leftarrow W - \underline{\alpha \, (\lambda \, W} + \nabla_W \frac{1}{n} \sum_{i=1}^{n} L(W, x_i, y_i) \,)$$

Gradients from
L2 regularization

Also called weight decay

We usually set $\lambda = 0.00005$ in neural networks

Compare K-NN and Linear classifier

- Do not need training
- Time consuming in test time
- Non-parametric, explicitly search through data
- More robust to outliers, using larger K

- Need training
- Time efficient in test time
- Parametric, use parameters to ”memorize” the dataset
- Can be sensitive to outliers

Next class

- Training Multi-Layer Perceptrons

- Back-propagation