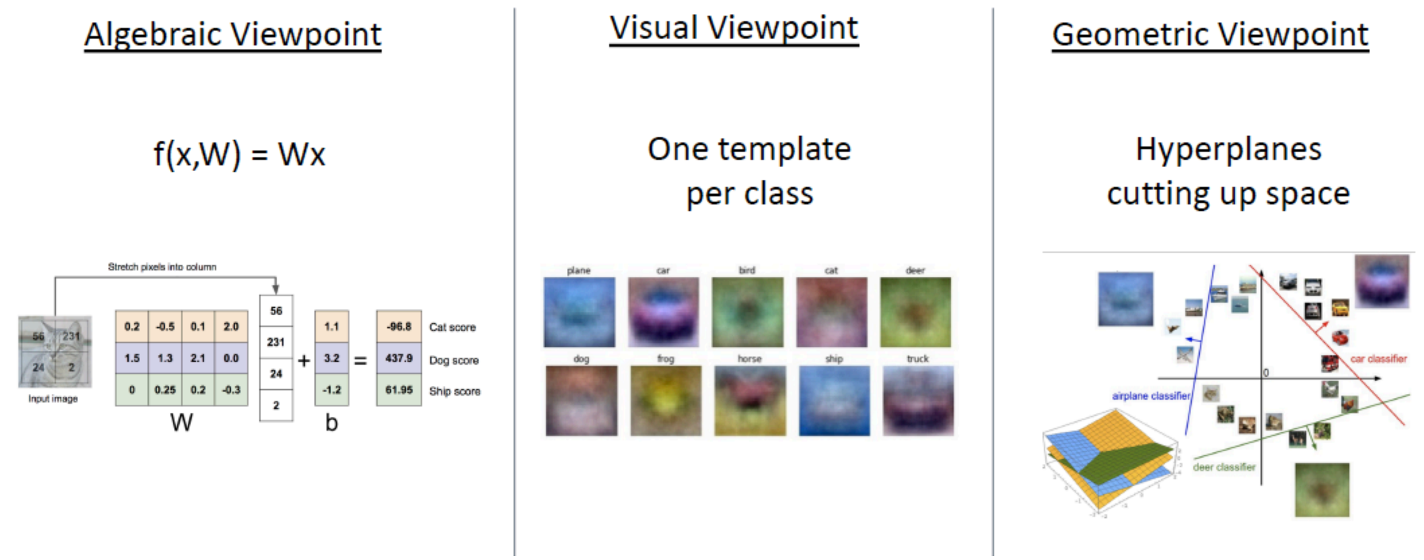
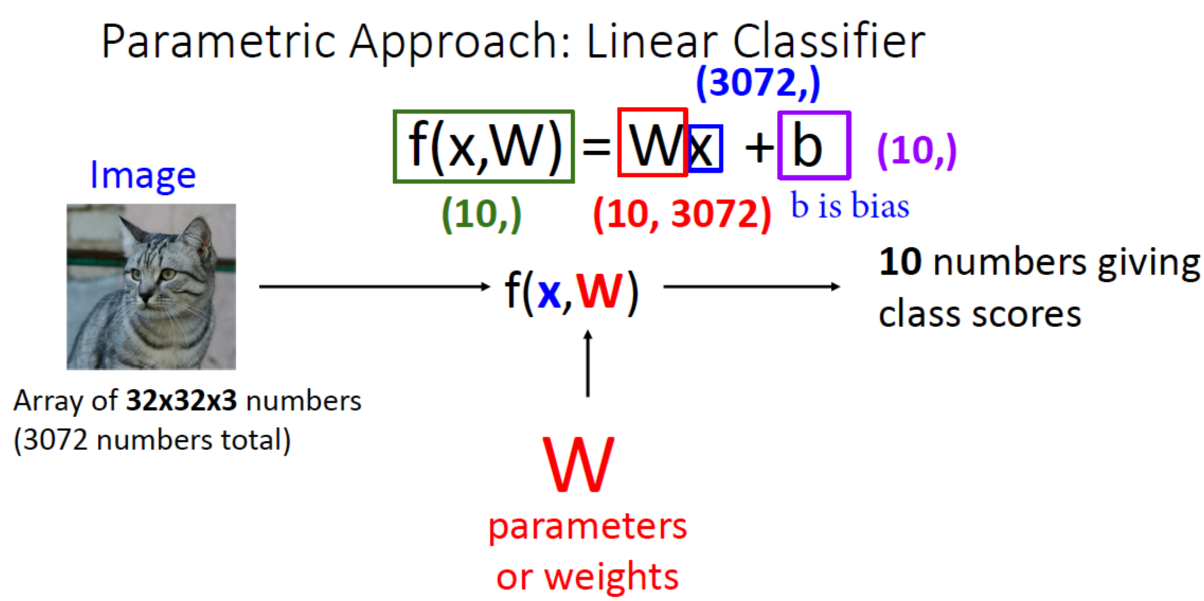
Lecture 3 - Linear Classifiers

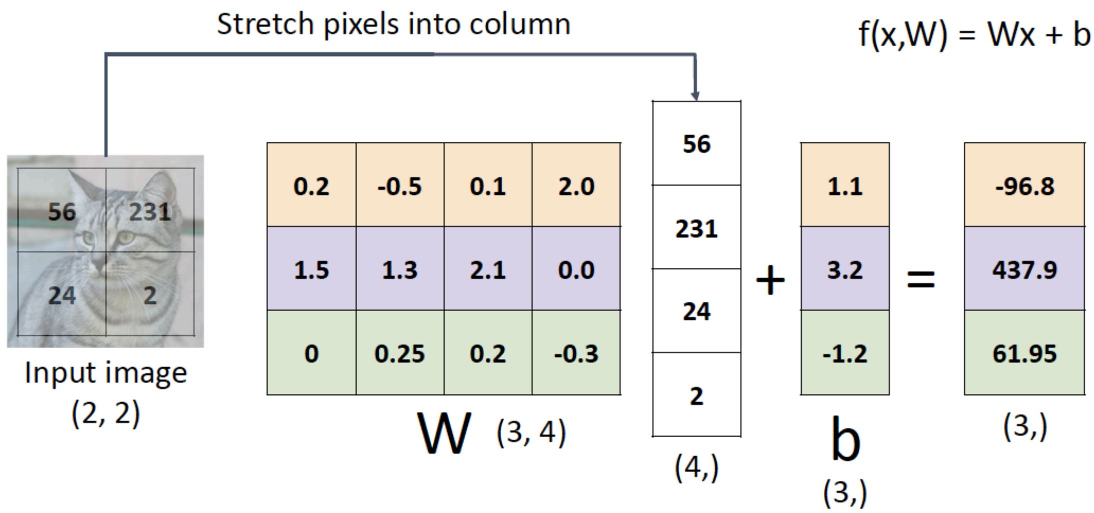
1. Linear Classifiers: Three Viewpoints



- **Algebraic Viewpoint / Parametric Approach**



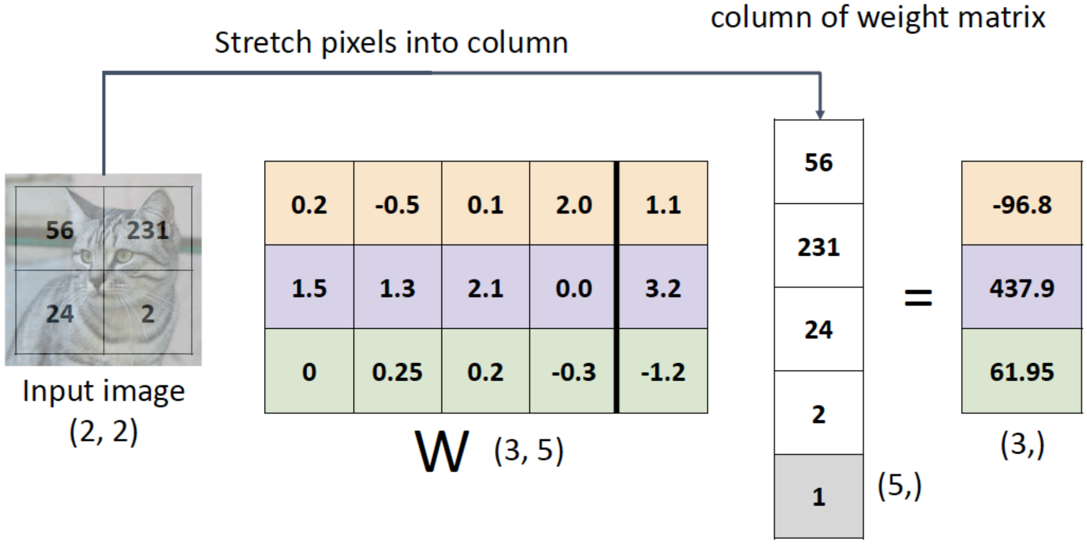
Example for 2x2 image, 3 classes (cat/dog/ship)



W: weight matrix (row number: number of categories; column number: number of pixel values)

Linear Classifier: Bias Trick

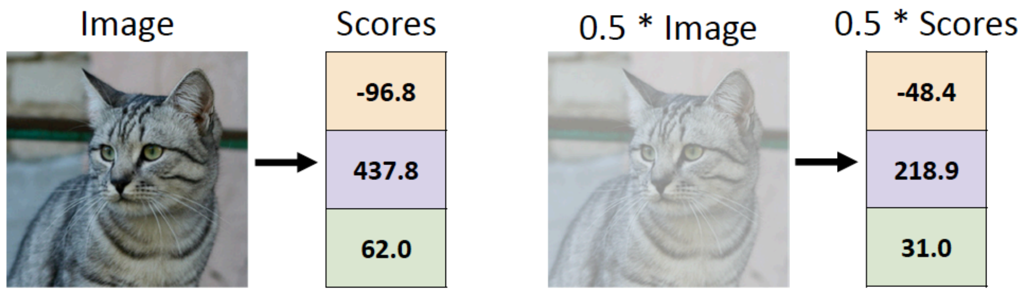
Add extra one to data vector; bias is absorbed into last column of weight matrix



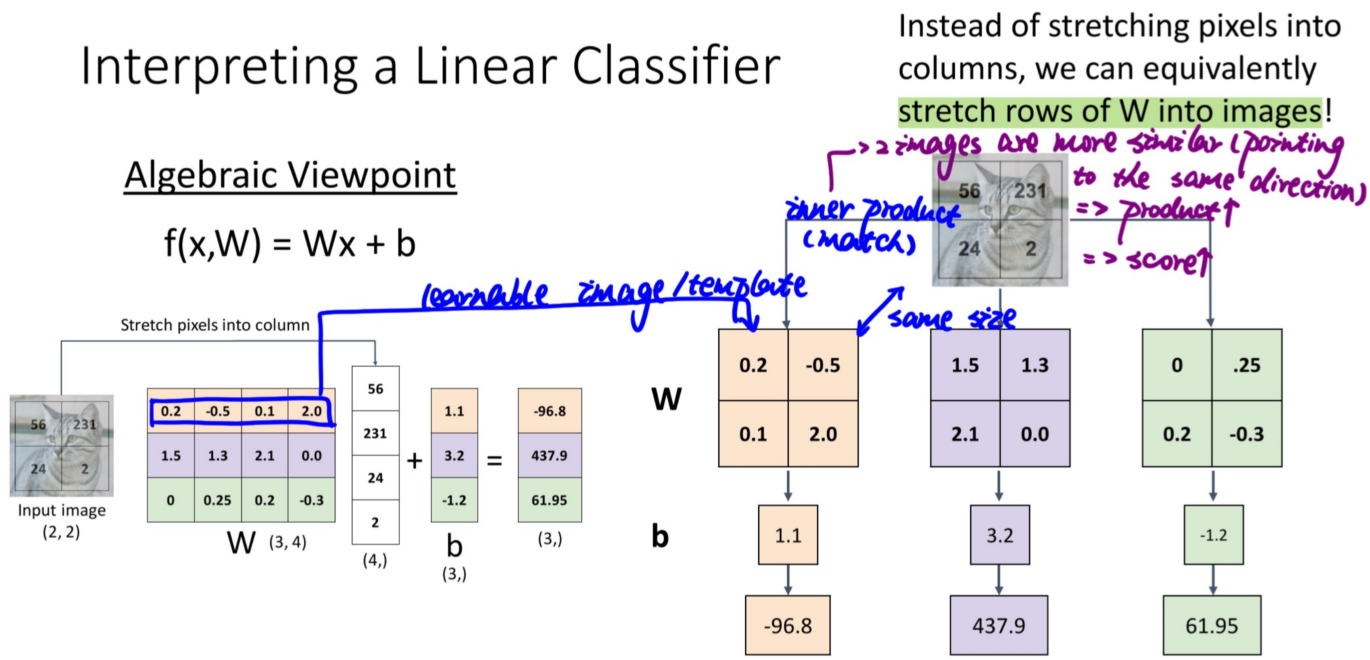
- Predictions are linear!

$$f(x, W) = Wx \quad \text{(ignore bias)}$$

$$f(cx, W) = W(cx) = c * f(x, W)$$



- **Visual Viewpoint**



将权重矩阵的每一行reshape成与input data一样的格式，然后求二者的inner product

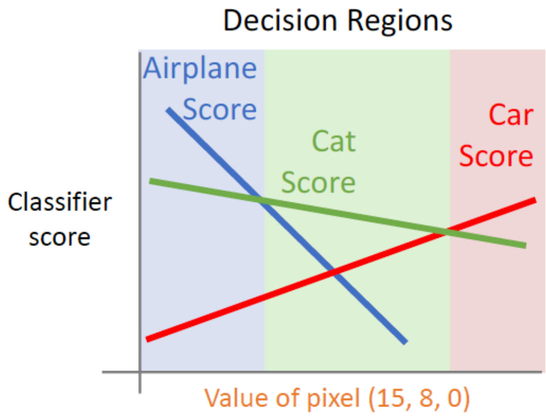
- Linear classifier has **one "template"** per category --> the "template" represents all appearances of a category (from different angles/directions...)

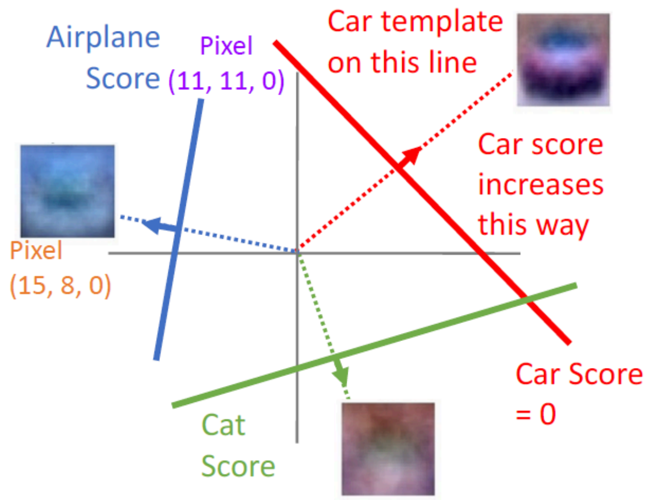
  [A single template cannot capture multiple modes of the data!] --> Eg. a horse template has 2 heads (left + right) 因为每一类只有一个模板，而马的头可能是朝左或右的，所以马的模板是由左右头的马叠加的

- **Geometric Viewpoint**

Score funtion: $s = f(x, W) = Wx + b$, which is shown as each line below

--> the movement of each line is the change of b





2. Loss function

- **Goal:** choose a good W
  - Todo 1: Use a **loss function** to quantify how good a value of W is
  - Todo 2: Find a W that minimizes the loss function (**optimization**)
- **Loss Function / Objective Function / Cost Function**

  --> tells how good our current classifier is

  - Low loss = good classifier; High loss = bad classifier
  - Negative loss function: reward function, profit function, utility function, fitness function, etc

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss for a single example is

$$L_i(f(x_i, W), y_i)$$

Loss for the dataset is average of per-example losses:

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

- where $f(x_i, W)$ is the prediction while y_i is the ground-truth

- **Loss Function 1: Cross-Entropy Loss (Multinomial Logistic Regression)**

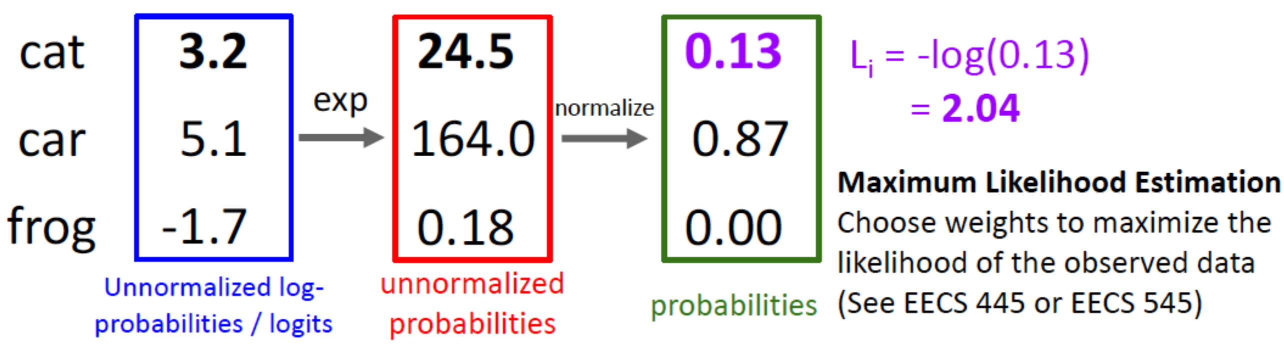
  --> use **softmax function** and interpret raw classifier scores as **probabilities**

$$score : s = f(x_i, W)$$

$$softmax\ function : P(Y = k|X = x_i) = \frac{exp(s_k)}{\sum_j exp(s_j)}(normalize)$$

$$loss\ function : L_i = -logP(Y = y_i|X = x_i) = -log\frac{exp(s_k)}{\sum_j exp(s_j)}$$

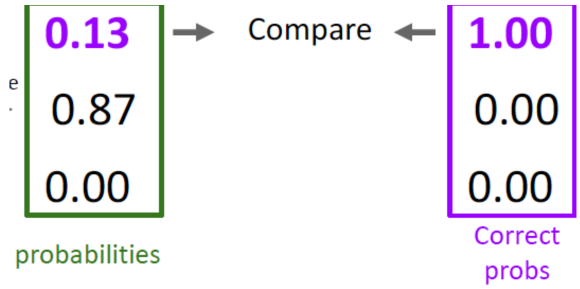
  - Eg. Input an image of a cat



  --> To compare:

  - **Kullback-Leibler Divergence**: $D_{KL}(P||Q) = \sum_y P(y)log\frac{P(y)}{Q(y)}$

    Cross Entopy: $H(P, Q) = H(P) + D_{KL}(P||Q)$

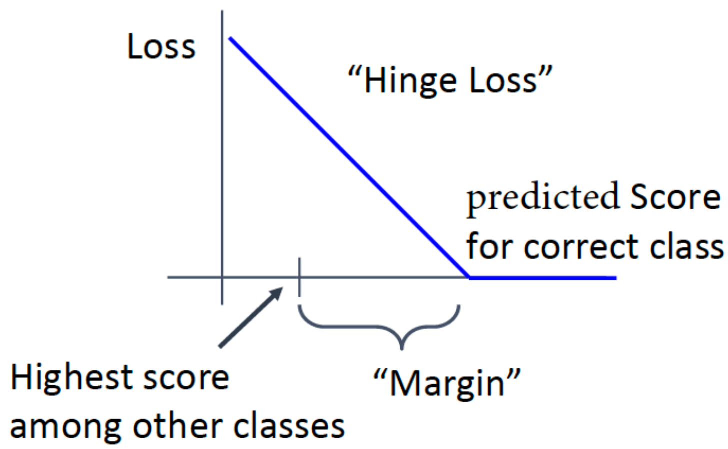


- Possible loss L_i: min: 0; max: infinity
- If all scores are small **random** value, the loss is: $L_i = -log(\frac{1}{C})$, where C is the number of classes

  For CIFAR-10: $L_i = -log(1/10) = log(10) \approx 2.3$

- **Loss Function 2: Multiclass SVM Loss**

  The score of the correct class should be higher than all the other scores



Given an example (x_i, y_i)
(x_i is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

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

Loss over the dataset: $L = \frac{\sum_j L_i}{C}$

, where y_i is the correct category and 1 is the margin

Given an example (x_i, y_i)

Eg.	cat	**3.2**	1.3	2.2
	car	5.1	**4.9**	2.5
	frog	-1.7	2.0	**-3.1**
	Loss	2.9		

Given an example (x_i, y_i)
(x_i is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

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

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

--> If the scores for the car image change a bit --> no influence: because 4.9 is much larger than other two scores, so the loss will still be 0 --> **loss remain the same** loss will change in cross-entropy loss

- Possible loss:
  - min: 0
  - max: infinity (when the largest score is not the corrected class while the corrected class has small score)
- If all the scores were random:

  $s_j \approx s_{y_i} \rightarrow L_i = 0 + (C-1) \times 1$
- If the sum was over all classes (including $i = y_i$) --> add a **constant loss offset** of 1 to the loss (not change the optimal weights)
- If the loss used a mean instead of a sum --> **scale** the loss --> not change the result
- If we used this loss instead:

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

  --> very different from SVM loss (change the entire geometry of the optimal landscape --> change optimal weights)

- **Summary of the Two Losses**
  - Cross-entropy loss > 0 --> always assign a nonzero loss

    SVM loss = 0
  - Softmax funtion is more commonly used in training neural networks