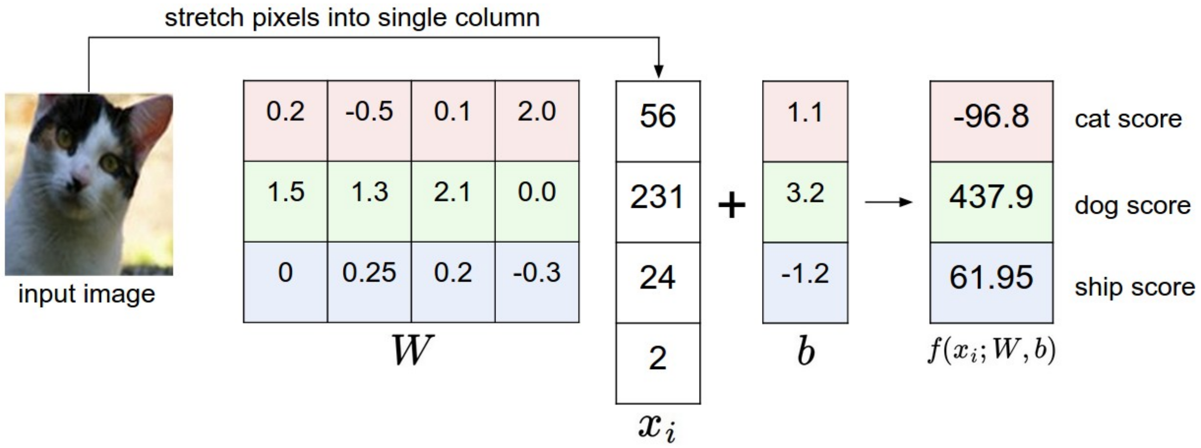
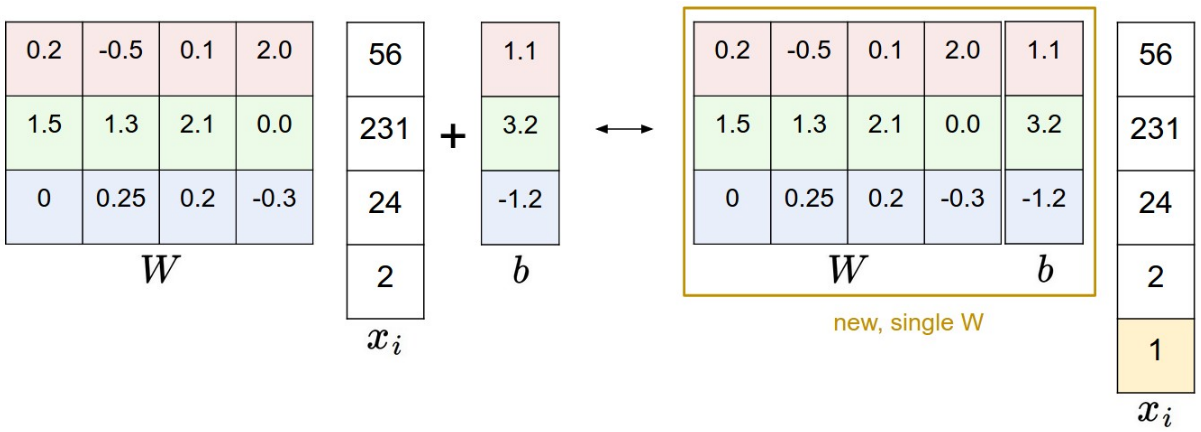
Linear Classification

Linear score function:

$$f(x_i, W, b) = Wx_i + b$$





Loss function

measure our unhappiness with outcomes

- **SVM loss（hinge loss）**

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

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

Example： Lets unpack this with an example to see how it works. Suppose that we have three classes that receive the scores $s=[13,-7,11]$ s=[13,-7,11], and that the first class is the true class (i.e. $y_i=0$yi=0). Also assume that ΔΔ (a hyperparameter we will go into more detail about soon) is 10. The expression above sums over all incorrect classes ($j \neq y_i$j≠yi), so we get two terms:

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

Multiclass Support Vector Machine "wants" the score of the correct class to be higher than all other scores by at least a margin of Δ.

Loss function (without regularization) implemented in Python

```
def L_i(x, y, W):
  """
  unvectorized version. Compute the multiclass svm loss for a single example (x,y)
  - x is a column vector representing an image (e.g. 3073 x 1 in CIFAR-10)
    with an appended bias dimension in the 3073-rd position (i.e. bias trick)
  - y is an integer giving index of correct class (e.g. between 0 and 9 in CIFAR-10)
  - W is the weight matrix (e.g. 10 x 3073 in CIFAR-10)
  """
  delta = 1.0 # see notes about delta later in this section
  scores = W.dot(x) # scores becomes of size 10 x 1, the scores for each class
  correct_class_score = scores[y]
  D = W.shape[0] # number of classes, e.g. 10
  loss_i = 0.0
  for j in xrange(D): # iterate over all wrong classes
    if j == y:
      # skip for the true class to only loop over incorrect classes
      continue
    # accumulate loss for the i-th example
    loss_i += max(0, scores[j] - correct_class_score + delta)
  return loss_i

def L_i_vectorized(x, y, W):
  """
  A faster half-vectorized implementation. half-vectorized
  refers to the fact that for a single example the implementation contains
  no for loops, but there is still one loop over the examples (outside this function)
  """
  delta = 1.0
  scores = W.dot(x)
  # compute the margins for all classes in one vector operation
```

```
margins = np.maximum(0, scores - scores[y] + delta)
# on y-th position scores[y] - scores[y] canceled and gave delta. We want
# to ignore the y-th position and only consider margin on max wrong class
margins[y] = 0
loss_i = np.sum(margins)
return loss_i
```

Loss function

measure our unhappiness with outcomes

- **Softmax classifier**

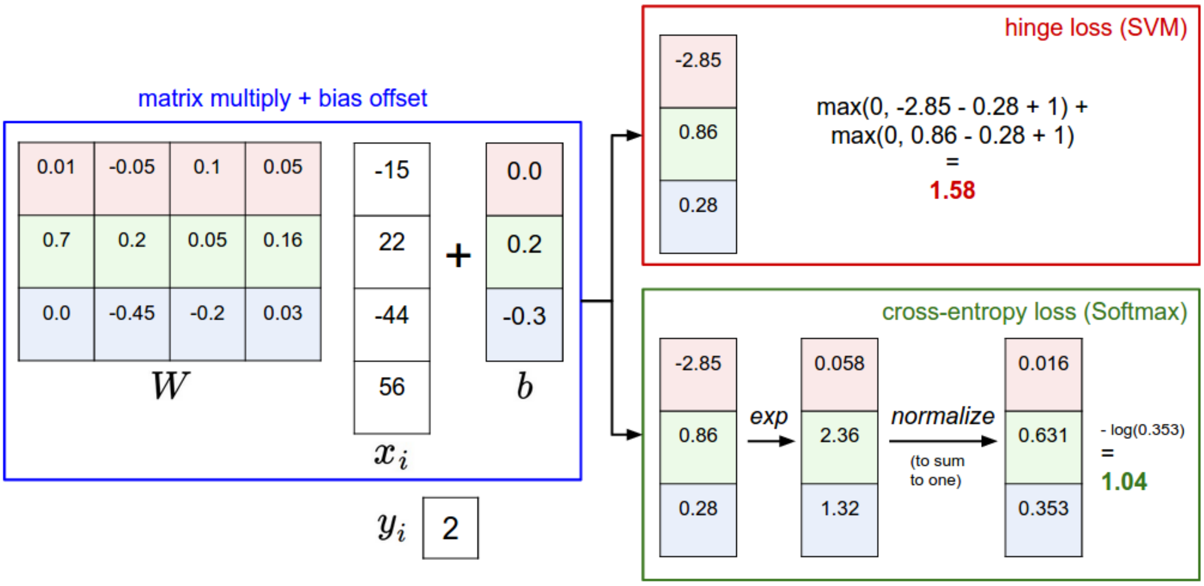
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

```
f = np.array([123, 456, 789]) # example with 3 classes and each having large scores
p = np.exp(f) / np.sum(np.exp(f)) # Bad: Numeric problem, potential blowup

# instead: first shift the values of f so that the highest number is 0:
f -= np.max(f) # f becomes [-666, -333, 0]
p = np.exp(f) / np.sum(np.exp(f)) # safe to do, gives the correct answer
```

SVM vs. Softmax

A picture might help clarify the distinction between the Softmax and SVM classifiers:



Regularization

有可能w1和w2在训练集上的表现一样好，但是w1只是在这个训练集表现好，而w2适用更广泛

光凭原来的Loss function比较不出w1和w2的优劣，加入**regularization**之后可以判断w2更好

regularization penalty:

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

$$R(W) = \text{ all the squared elements of } W$$

$$\text{如}w_1=[1,0,0,0]，\ \mathrm{R(w1)} = 1*1 + 0*0 + 0*0 + 0*0 = 1$$

Full Multiclass SVM:

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

For example, suppose that we have some input vector $x=[1,1,1,1]$x=[1,1,1,1]

and two weight vectors $w_1=[1,0,0,0]$, $w_2=[0.25,0.25,0.25,0.25]$.

Then $w_1^T x = w_2^T x = 1$ so both weight vectors lead to the same dot product, but the L2 penalty of w_1 is 1.0 while the L2 penalty of w_2 is only 0.25.

Therefore, according to the L2 penalty the weight vector w_2w2 would be preferred since it achieves a lower regularization loss.