Linear Classification

Most of the materials are taken from here

Nastaran Okati

Outline

- Introduction to Linear Classification
- Score Functions
- Interpreting a Linear Classifier
- Loss Function

1.
Introduction

Linear Classification

- What are the disadvantages of KNN Classifier?
- The Classifier must remember all the training data (space inefficient)
- Classifying a test image is expensive

Score Function

Maps the pixel values of an image to confidence scores for each class.

- Training dataset: $x_i \in R^D$
- Labels: $y_i \in 1 \ldots K$
- The score function maps the raw image pixels to class scores $f : R^D \mapsto R^K$

Linear Classifier

The simplest possible function is a linear mapping:

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

We assume that our images are flattened out to a single column vector of shape [Dx1]

What are the shapes of W and b?

What are the exact shapes of x, W and b for the Cifar Dataset?

- The single matrix multiplication between W and X is effectively evaluating 10 separate classifiers in parallel and each classifier is a row of W.

- The input data is fixed but we have control over W and b and our goal is to set these in such way that the computed scores match the ground truth labels across the whole training set.

- Intuitively we wish that the correct class has a score that is higher than the scores of incorrect classes.

What are the advantages?

- The training data is used to learn the parameters W,b, but once the learning is complete we can discard the entire training set and only keep the learned parameters

- Classifying the test image involves a single matrix multiplication and addition, which is significantly faster than comparing a test image to all training images.

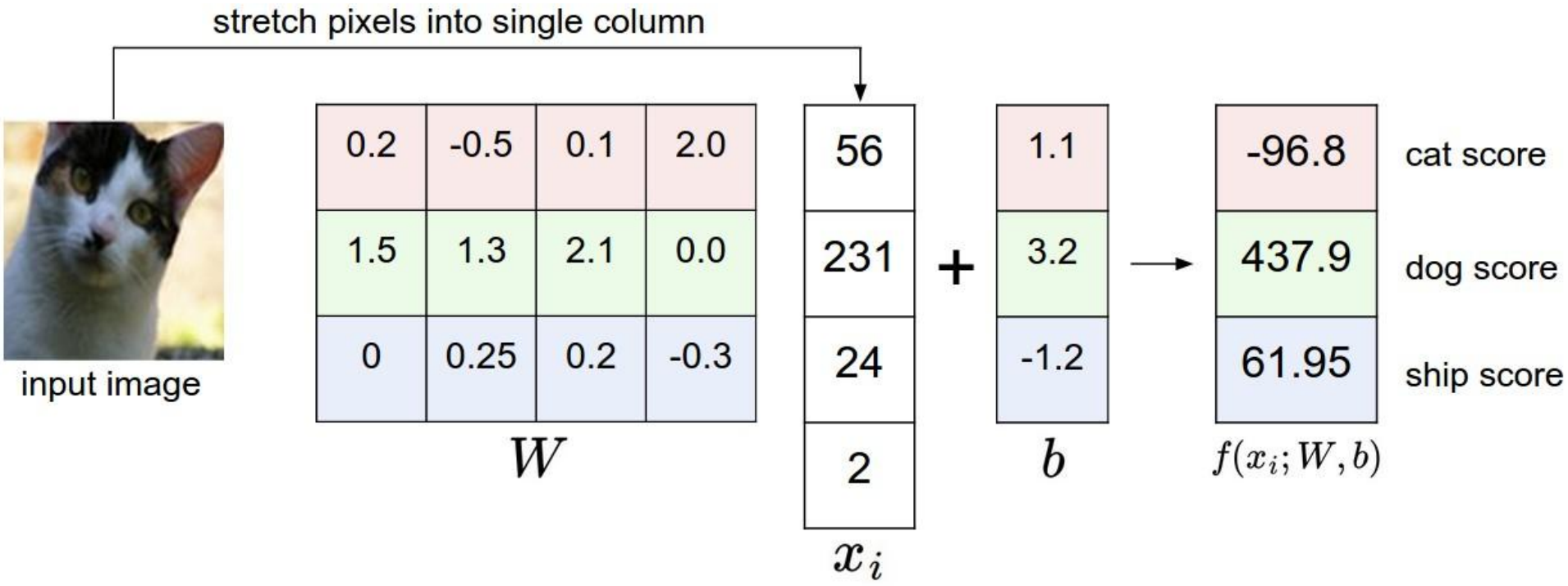
Interpreting a linear classifier

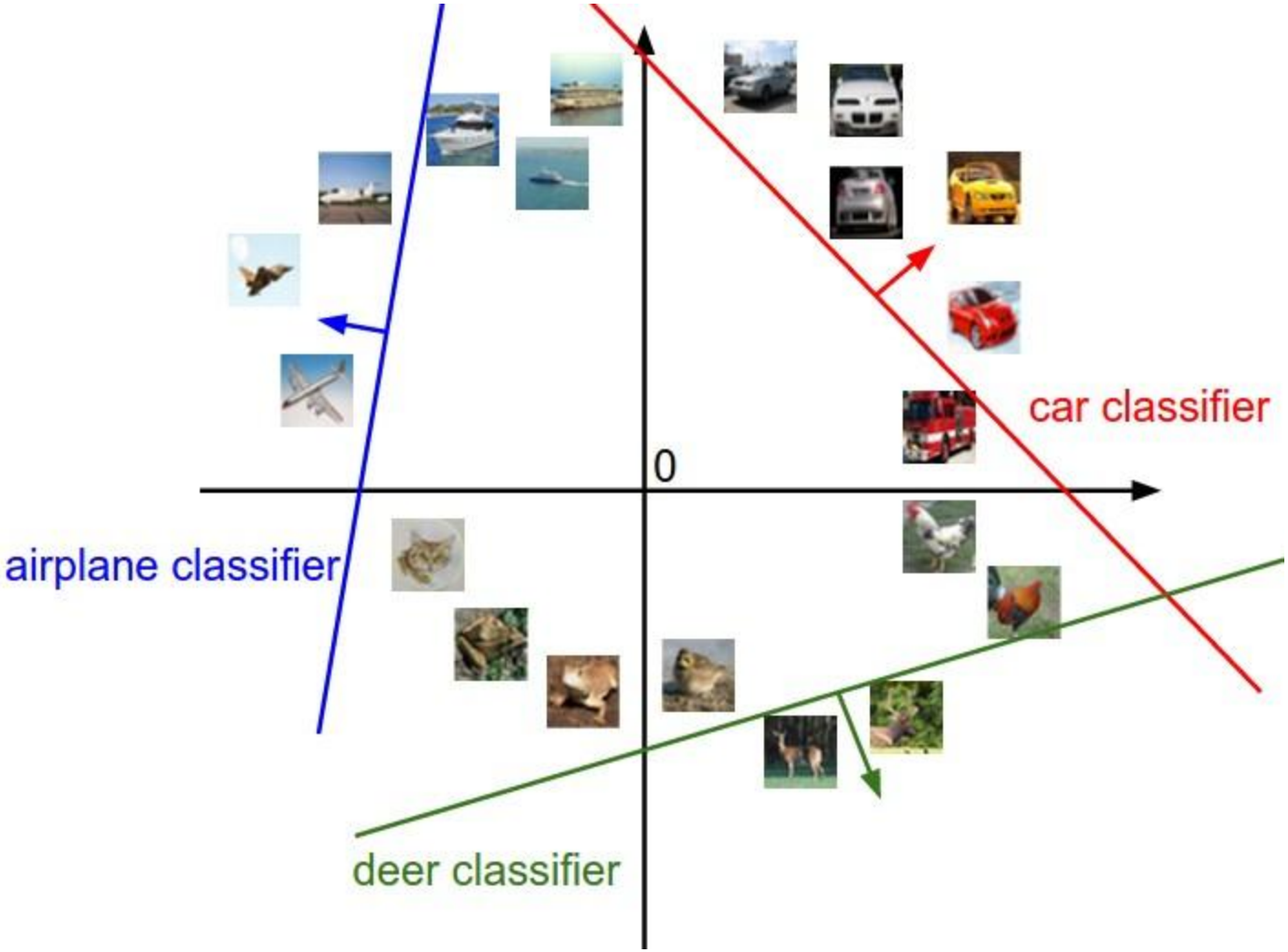
- A linear classifier computes the score of a class as a weighted sum of all of its pixel values across all 3 of its color channels

- The function has the capacity to like or dislike (depending on the sign of each weight) certain colors at certain positions

Example

- 4-pixel image
- 3 classes
- Is it a good one?




car classifier
0
airplane classifier
deer classifier

- Every row of W is a classifier for one of the classes. The geometric interpretation of these numbers is that as we change one of the rows of W, the corresponding line in the pixel space will rotate in different directions.

- The biases b allow our classifiers to translate the lines. Without the bias terms, plugging in xi=0 would always give score of zero regardless of the weights, so all lines would be forced to cross the origin.

Interpretation of W as template matching

- Each row of W corresponds to a template for one class

- The score of each class for an image is then obtained by comparing each template with the image using an inner product one by one to find the one that “fits” best.

Another terminology...

we are still effectively doing Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class (although we will learn it, and it does not necessarily have to be one of the images in the training set), and we use the (negative) inner product as the distance instead of the L1 or L2 distance.

Example learned weights at the end of learning for CIFAR-10

Look at the ship, horse and car template!

Bias Trick

- Cumbersome to keep track of two sets of parameters separately
- Combine the two sets of parameters into a single matrix that holds both of them by extending the vector x_i with one additional dimension that always holds the constant 1 - a default bias dimension.

Cifar-10 Example

What are the new matrix dimensions?

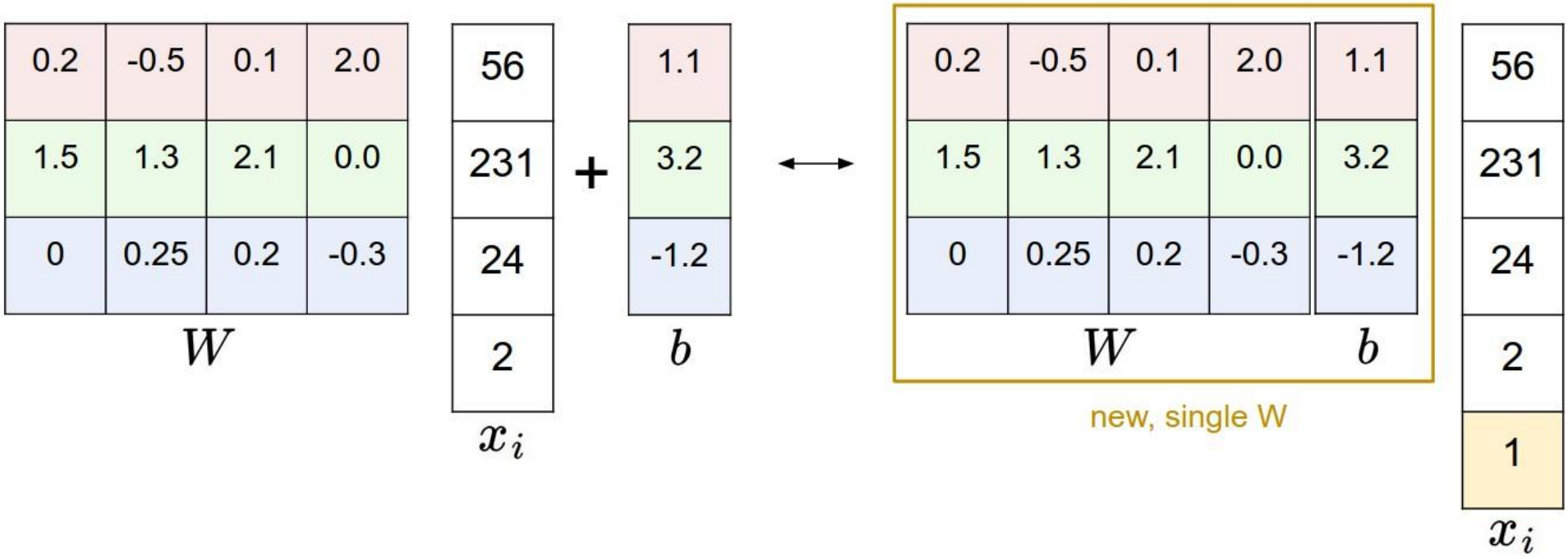


Image data preprocessing

- In Machine Learning, it is a very common practice to always perform normalization of your input features
- Important to center your data by subtracting the mean from every feature
- Further common preprocessing is to scale each input feature so that its values range from [-1, 1]

2.
Loss Functions

Loss Function

We are going to measure our unhappiness with outcomes with a loss function (cost function or objective)

The loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

Multiclass Support Vector Machine loss

The SVM loss is set up so that the SVM “wants” the correct class for each image to a have a score higher than the incorrect classes by some fixed margin Δ

The SVM “wants” a certain outcome in the sense that the outcome would yield a lower loss

SVM Loss

$$s_j = f(x_i, W)_j$$

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

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

Example

3 Classes

$$s = [13, -7, 11]$$

$$y_i = 0$$

$$\Delta = 10$$

Compute the SVM loss!

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

Regularization

Suppose that we have a dataset and a set of parameters W that correctly classify every example.

The issue is that this set of W is not necessarily unique.

Some examples of other functions?

Regularization Penalty

The most common regularization penalty is the L2 norm that discourages large weights through an elementwise quadratic penalty over all parameters:

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

Full SVM Loss

- Data loss + regularization loss

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

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

- Penalizing large weights tends to improve generalization, because it means that no input dimension can have a very large influence on the scores all by itself.

- The classifier is encouraged to take into account all input dimensions to small amounts rather than a few input dimensions and very strongly

Example

Which one has higher regularization loss? Which one is prefered?

W = [1,0,0,0]

W’ = [0.25,0.25,0.25.0.25]

Setting Delta

The hyperparameters Δ and λ both control the same tradeoff: The tradeoff between the data loss and the regularization loss in the objective.

The exact value of the margin between the scores (e.g. $\Delta=1$, or $\Delta=100$) is in some sense meaningless because the weights can shrink or stretch the differences arbitrarily.

Coding time!

Implement the SVM Loss function!

Softmax Classifier

In the Softmax classifier, the function mapping stays unchanged, but we now interpret these scores as the unnormalized log probabilities for each class and replace the hinge loss with a cross-entropy loss that has the form:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

As before, the full loss for the dataset is the mean of Li over all training examples together with a regularization term R(W).

Softmax Function

Takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Probabilistic interpretation

The Softmax classifier interprets the scores inside the output vector f as the unnormalized log probabilities

Exponentiating these quantities gives the (unnormalized) probabilities, and the division performs the normalization so that the probabilities sum to one

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

Numeric stability

- The intermediate terms may be very large due to the exponentials

- Dividing large numbers can be numerically unstable, so it is important to use a normalization trick

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

What is a good choice for C?

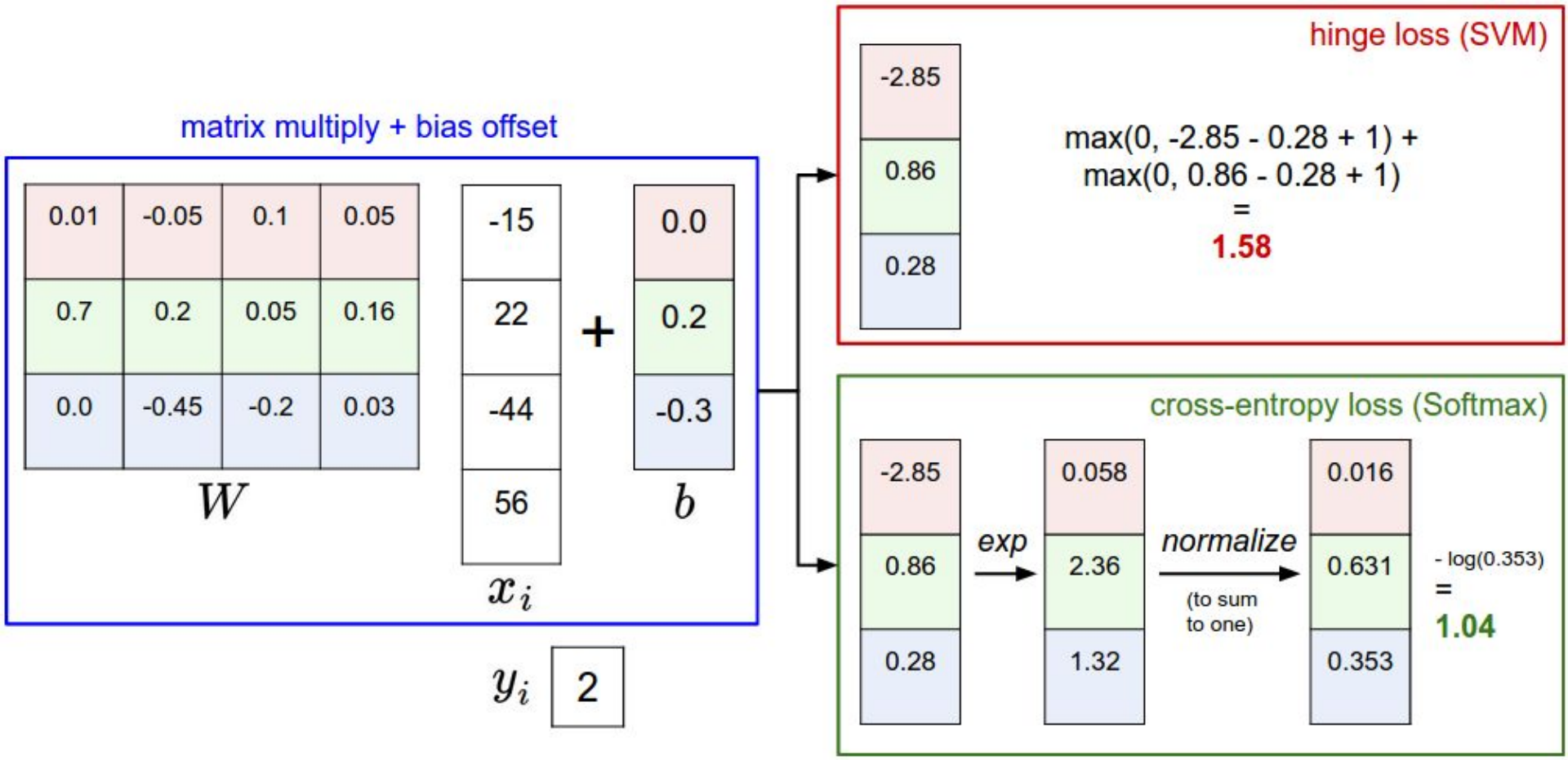
$$\log C = -\max_j f_j$$

This simply states that we should shift the values inside the vector f so that the highest value is zero

SVM vs Softmax

The SVM classifier uses the hinge loss, or also sometimes called the max-margin loss.

The Softmax classifier uses the cross-entropy loss


matrix multiply + bias offset
0.01
-0.05
0.1
0.05
0.7
0.2
0.05
0.16
0.0
-0.45
-0.2
0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
y_i
2
hinge loss (SVM)
-2.85
0.86
0.28
max(0, -2.85 - 0.28 + 1) +
max(0, 0.86 - 0.28 + 1)
=
1.58
cross-entropy loss (Softmax)
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum
to one)
0.016
0.631
0.353
- log(0.353)
=
1.04

Unlike the SVM which computes uncalibrated and not easy to interpret scores for all classes, the Softmax classifier allows us to compute “probabilities” for all labels

How peaky or diffuse these probabilities are depends directly on the regularization strength λ

$$[1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26]$$

$$[0.5, -1, 0] \rightarrow [e^{0.5}, e^{-1}, e^0] = [1.65, 0.37, 1] \rightarrow [0.55, 0.12, 0.33]$$

SVM or Softmax

Softmax is never fully happy with the scores it produces: the correct class could always have a higher probability and the incorrect classes always a lower probability and the loss would always get better.

SVM is happy once the margins are satisfied and it does not micromanage the exact scores beyond this constraint.

Give and example!

How do we efficiently determine the parameters that give the best (lowest) loss?

Wait until next session! 

Thanks!

Any questions?