Garlic halved horizontally = nature's Voronoi diagram?

en.wikipedia.org/wiki/Voronoi_d...
Outline

• Linear Classifiers
• Gradient descent
• Cross-entropy
• Softmax
Linear classifier: Notation

• The observation \( x = [x_0, \ldots, x_{D-1}] \) is a real-valued vector (\( D \) is its dimension)

• The class label \( y \in \mathcal{Y} \) is drawn from some finite set of class labels.

• Usually the output vocabulary, \( \mathcal{Y} \), is some set of strings. For convenience, though, we usually map the class labels to a sequence of integers, \( \mathcal{Y} = \{0, \ldots, V - 1\} \), where \( V \) is the vocabulary size
Linear classifier: Definition

A linear classifier is defined by

\[ f(x) = \arg \max_k w_k @ x + b_k \]

- @ means matrix product or dot product, \( w_k @ x = \sum_{j=0}^{D-1} x_j w_{k,j} \)
- \( w_k, b_k \) are the **weight vector** and **bias** corresponding to **class** \( k \).
- There are a total of \( V(D + 1) \) trainable parameters:
  
  \[
  \text{(\# params)} = (\text{\# classes}) \times (\text{len}(w_k) + \text{len}(b_k)) \\
  = V(D + 1)
  \]
Example

Consider a two-class classification problem, with the biases $b_0 = b_1 = 0$, and

$$w_0 = [2,1]$$
$$w_1 = [1,2]$$
Example

Notice that in the two-class case, the equation

$$f(x) = \arg\max_{k} w_k \beta x + b_k$$

Simplifies to

$$f(x) = \begin{cases} 1 & w_1 \beta x + b_1 > w_0 \beta x + b_0 \\ 0 & w_1 \beta x + b_1 < w_0 \beta x + b_0 \end{cases}$$

The class boundary is the line whose equation is

$$(w_1 - w_0) \beta x + (b_1 - b_0) = 0$$
Multi-class linear classifier

In a general multi-class linear classifier,

\[ f(x) = \arg\max_k w_k @ x + b_k \]

The boundary between class \( k \) and class \( l \) is the line (or plane, or hyperplane) given by the equation

\[ (w_l - w_k) @ x + (b_l - b_k) = 0 \]
Voronoi regions

The classification regions in a linear classifier are called Voronoi regions.

A **Voronoi region** is a region that is

- Convex (if \( u \) and \( v \) are points in the region, then every point on the line segment \( uv \) connecting them is also in the region)
- Bounded by piece-wise linear boundaries
Outline

• Linear Classifiers
• Gradient descent
• Cross-entropy
• Softmax
Gradient descent

Suppose we have training tokens \((x_i, y_i)\), and we have some initial class vectors \(w_0\) and \(w_1\). We want to update them as

\[
\begin{align*}
w_0 &\leftarrow w_0 - \eta \nabla_{w_0} \mathcal{L} \\
w_1 &\leftarrow w_1 - \eta \nabla_{w_1} \mathcal{L}
\end{align*}
\]

...where \(\mathcal{L}\) is some loss function. What loss function makes sense?
Zero-one loss function

The most obvious loss function for a classifier is its classification error rate,

\[ \mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) \]

Where \( \ell(\hat{y}, y) \) is the zero-one loss function,

\[ \ell(\hat{y}, y) = \begin{cases} 0 & \hat{y} = y \\ 1 & \hat{y} \neq y \end{cases} \]
Non-differentiable!

The problem with the zero-one loss function is that it’s not differentiable:

\[
\nabla_{w_0} \ell(f(x), y) = \frac{\partial \ell(f(x), y)}{\partial f(x)} \nabla_{w_0} f(x)
\]

\[
= \begin{cases} 
0 & f(x) \neq y \\
+\infty & f(x) = y^+
\end{cases}
\]

\[
= \begin{cases} 
-\infty & f(x) = y^-
\end{cases}
\]

\[
\ell(f(x), y) = \begin{cases} 
0 & f(x) = y \\
1 & f(x) \neq y
\end{cases}
\]
Outline

• Linear Classifiers: multi-class and 2-class
• Gradient descent
• Cross-entropy
• Softmax
One-hot vectors

A **one-hot vector** is a binary vector in which all elements are 0 except for a single element that’s equal to 1.
Example: Binary classifier

Consider the classifier

\[ f(x_i) = [f_0(x_i), f_1(x_i)], \]

\[ f_c(x_i) = \begin{cases} 
1 & c = \arg\max_k w_k @ x + b_k \\
0 & \text{otherwise}
\end{cases} \]

... with two classes. Then the classification regions might look like this.
Consider the classifier
\[ f(x_i) = [f_0(x_i), \ldots, f_{V-1}(x_i)] \]
\[ f_c(x_i) = \begin{cases} 
1 & c = \text{argmax}_k w_k @ x + b_k \\
0 & \text{otherwise}
\end{cases} \]

... with 20 classes. Then some of the classifications might look like this.
Using one-hot vectors to calculate the loss

• Suppose that the output is a one-hot vector. Then the goal of the classifier is to set \( f_c(x_i) = 1 \) for the correct class, and \( f_c(x_i) \approx 0 \) for all others.

• We can measure this by a formula like:

\[
\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \log f_{y_i}(x_i)
\]

In words:

• choose the \( y_i \)th output of the classifier.
• If that output is \( f_{y_i}(x_i) = 1 \), then the loss is zero.
• If that output is \( f_{y_i}(x_i) < 1 \), then the loss is large (\( \infty \) if \( f_{y_i}(x_i) = 0 \)).
Cross-entropy

This loss function,
\[ \mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \log f_{y_i}(x_i), \]
is called cross-entropy. By measuring the negative log-probability of the correct class, we are measuring the extra uncertainty that is added to the system by classification errors.

CC-SA 4.0,
Cross-entropy of a one-hot vector is still not differentiable!

Consider the classifier

$$f(x_i) = [f_0(x_i), ..., f_{V-1}(x_i)]$$

$$f_c(x_i) = \begin{cases} 
1 & c = \text{argmax}_k w_k @ x + b_k \\
0 & \text{otherwise}
\end{cases}$$

Unfortunately, the cross-entropy of a one-hot vector is still not differentiable!

$$\mathcal{L} = -\log f_{y_i}(x_i) = \begin{cases} 
0 & f_{y_i}(x_i) = 1 \\
\infty & f_{y_i}(x_i) = 0
\end{cases}$$
Outline

• Linear Classifiers: multi-class and 2-class
• Gradient descent
• One-hot vectors
• Softmax
The problem with cross-entropy: $-\log 0 = \infty$

- Cross-entropy is a great loss function because $-\log 1 = 0$, so it measures no loss if the classifier has the right answer.
- The problem is that $-\log 0 = \infty$, so if the classifier has the wrong answer, the loss function is unmeasurably huge.
The solution: avoid 0-valued outputs

• The solution is to modify $f(x)$ so that it never outputs exactly 0
• Instead, we want $f(x)$ to approach 0 as the classifier gets more confident, but it should never actually reach zero
Argmax versus Softmax

The argmax version of the classifier is

\[ f(x_i) = [f_0(x_i), \ldots, f_{V-1}(x_i)], \quad f_c(x_i) = \begin{cases} 1 & c = \text{argmax}_k w_k @ x + b_k \\ 0 & \text{otherwise} \end{cases} \]

We can smooth it by using the softmax function, defined as

\[ f(x_i) = [f_0(x_i), \ldots, f_{V-1}(x_i)], \quad f_c(x_i) = \frac{\exp(w_c @ x + b_c)}{\sum_{k=0}^{V-1} \exp(w_k @ x + b_k)} \]
The softmax function

This is called the softmax function:

$$\text{softmax}(x_i) = [f_0(x_i), \ldots, f_{V-1}(x_i)]$$

$$\text{softmax}(w@x + b) = \frac{\exp(w_c@x + b_c)}{\sum_{k=0}^{V-1} \exp(w_k@x + b_k)}$$
Key features of the softmax

\[
\text{softmax}(w@x + b) = \frac{\exp(w_c@x + b_c)}{\sum_{k=0}^{V-1} \exp(w_k@x + b_k)}
\]

Notice that the softmax function is (1) smooth, and (2) behaves like a probability distribution:

- \(0 < \text{softmax}(w@x + b)_c < 1\)
- \(\sum_{c=0}^{V-1} \text{softmax}(w@x + b)_c = 1\)

Binary classifier with softmax output
Quiz

• Go to https://us.prairielearn.com/pl/course_instance/129874/assessment/2330383 and try the quiz
Gradient of the cross-entropy of the softmax

Consider the classifier

\[ f_c(x_i) = \frac{\exp(w_c@x + b_c)}{\sum_{k=0}^{V-1} \exp(w_k@x + b_k)} \]

The softmax is smooth, so its logarithm is differentiable:

\[ \mathcal{L} = -\log f_{y_i}(x_i) = -(w_{y_i}@x + b_{y_i}) + \log \sum_{k=0}^{V-1} \exp(w_k@x + b_k) \]

\[ \nabla_{w_c} \mathcal{L} = \begin{cases} (f_c(x_i) - 1)x_i & c = y_i \\ f_c(x_i)x_i & \text{otherwise} \end{cases} \]
...is the same as the gradient of MSE for linear regression!

For linear regression, we had

\[ \nabla_w \epsilon_i^2 = 2 \epsilon_i x_i \]

For the softmax classifier with cross-entropy loss, we have

\[ \nabla_{w_c} \mathcal{L} = \epsilon_{i,c} x_i \]

...where \( \epsilon_{i,c} \) is the error of the cth output of the classifier:

\[ \epsilon_{i,c} = \begin{cases} f_c(x_i) - 1 & c = y_i \text{ (output should be 1)} \\ f_c(x_i) - 0 & \text{otherwise (output should be 0)} \end{cases} \]
Stochastic gradient descent

Suppose we have a training token \((x_i, y_i)\), and we have some initial class vectors \(w_c\). Using softmax and cross-entropy loss, we can update the weight vectors as

\[
w_c \leftarrow w_c - \eta \epsilon_{i,c} x_i
\]

...where

\[
\epsilon_{i,c} = \begin{cases} 
  f_c(x_i) - 1 & c = y_i \\
  f_c(x_i) - 0 & \text{otherwise}
\end{cases}
\]

Training token \(x_i\) of class \(y_i\) = 0

\(w_1\) moves in the direction of \(x_i\)

\(w_0\) moves in the direction opposite \(x_i\)
Outline

• Linear Classifiers: \( f(x) = \arg\max_k w_k @ x + b_k \)

• Gradient descent: \( w_c \leftarrow w_c - \eta \nabla_{w_c} \mathcal{L} \)

• Cross-entropy: \( \mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \log f_{y_i}(x_i) \)

• Softmax: \( \text{softmax}(w @ x + b) = \frac{\exp(w_c @ x + b_c)}{\sum_{k=0}^{V-1} \exp(w_k @ x + b_k)} \)

• Gradient of the cross-entropy of the softmax:
  \[
  w_c \leftarrow w_c - \eta \varepsilon_{i,c} x_i, \quad \varepsilon_{i,c} = \begin{cases} 
    f_c(x_i) - 1 & c = y_i \text{ (output should be 1)} \\
    f_c(x_i) - 0 & \text{otherwise (output should be 0)}
  \end{cases}
  \]