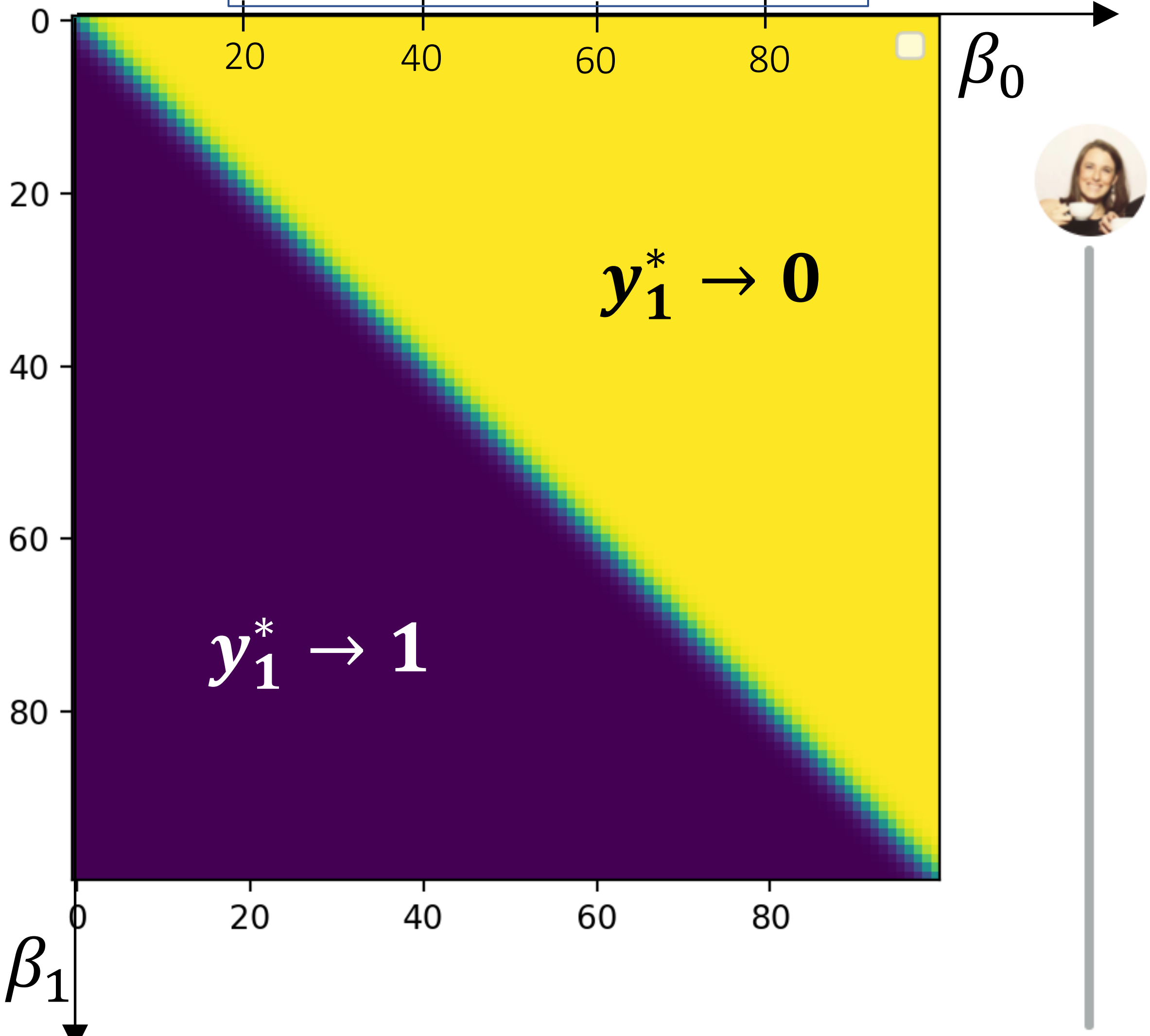
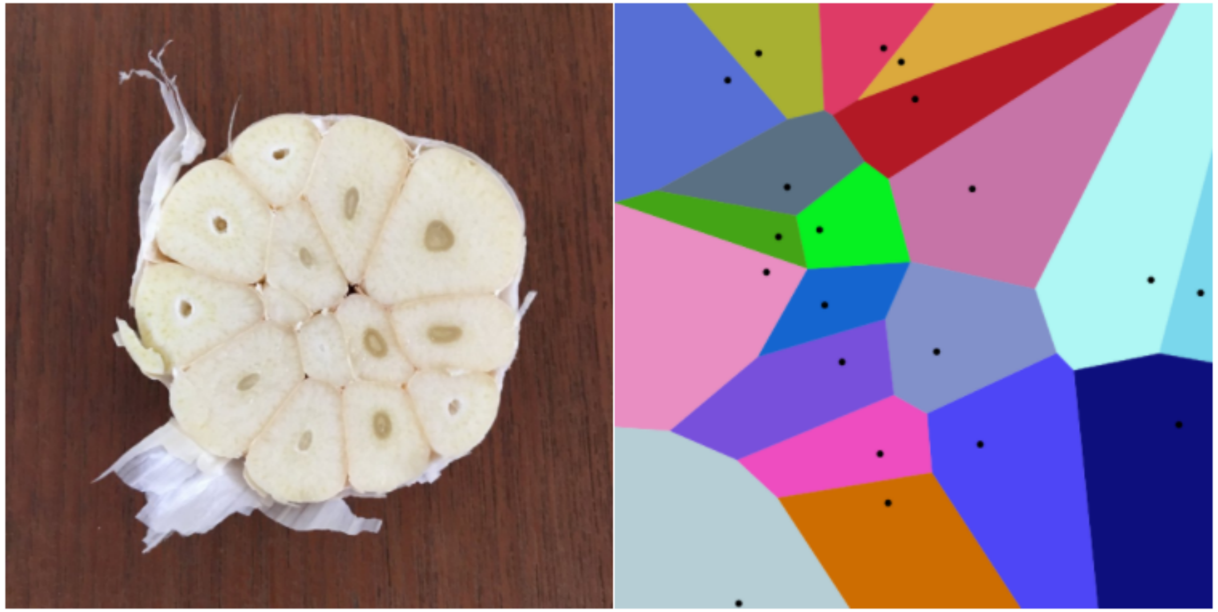
Multi-Class Linear Classifiers



Aliza Aufrichtig @alizauf · Mar 4

Garlic halved horizontally = nature's Voronoi diagram?

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



12 234 878

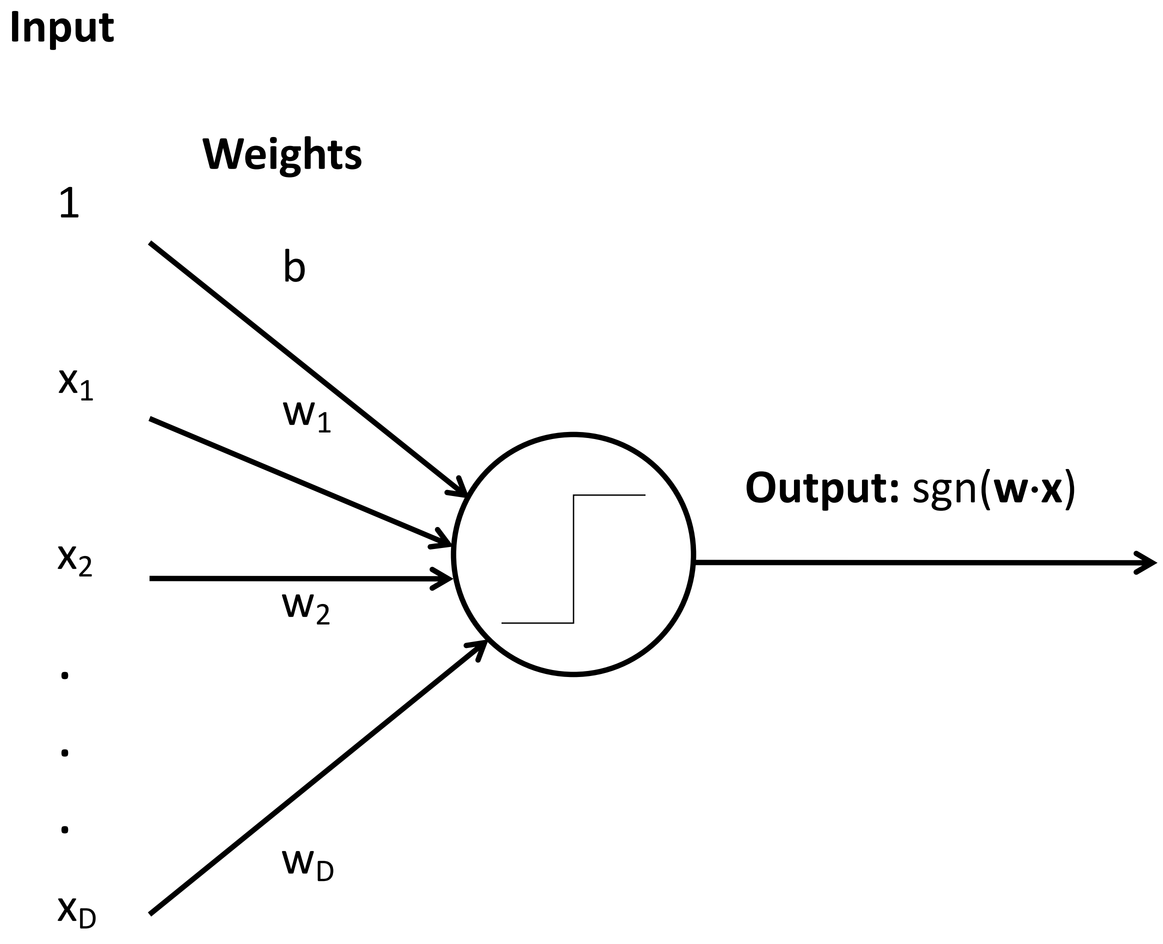
Outline

- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Outline

- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Review: Two-Class Perceptron



True class is $y \in \{-1,1\}$.

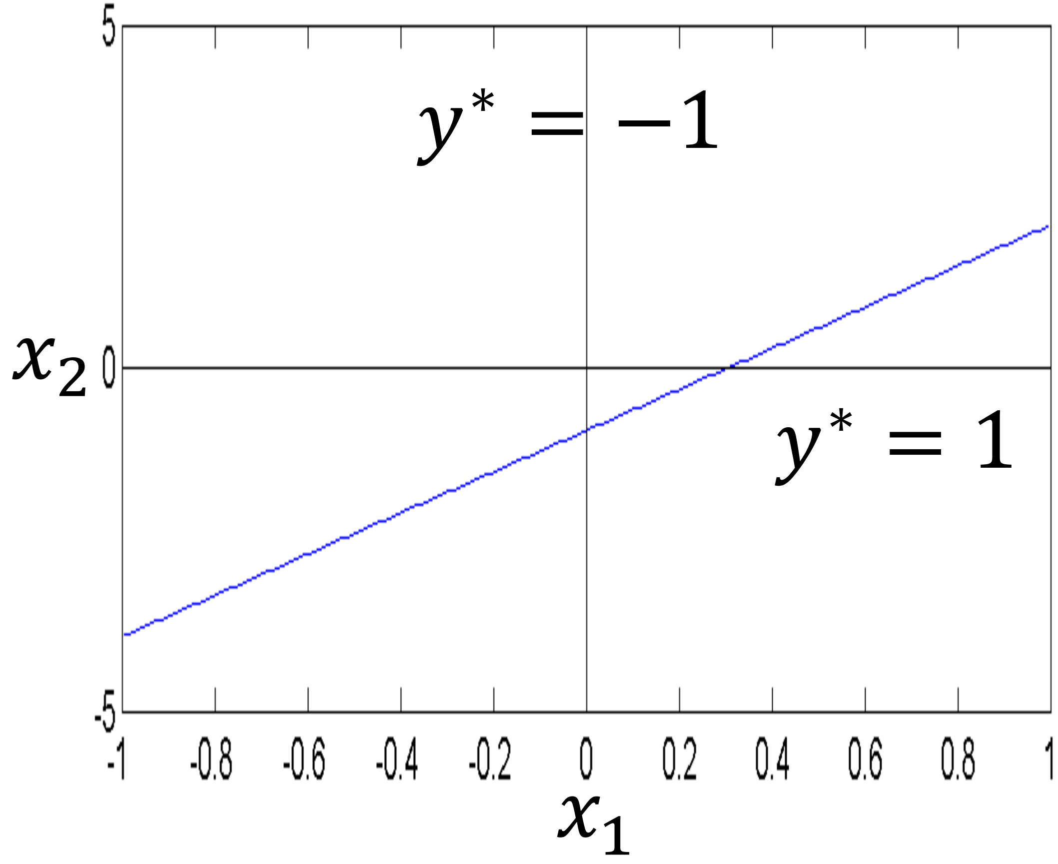
Classifier output is

$$y* = \text{sgn}(w_1 x_1 + \ldots + w_D x_D + b) = \text{sgn}(\vec{w}^T \vec{x}) \in \{-1,1\}$$

Where $\vec{w} = [w_1, \ldots, w_D, b]^T$

and $\vec{x} = [x_1, \ldots, x_D, 1]^T$

Review: Two-Class Perceptron



True class is $y \in \{-1,1\}$.

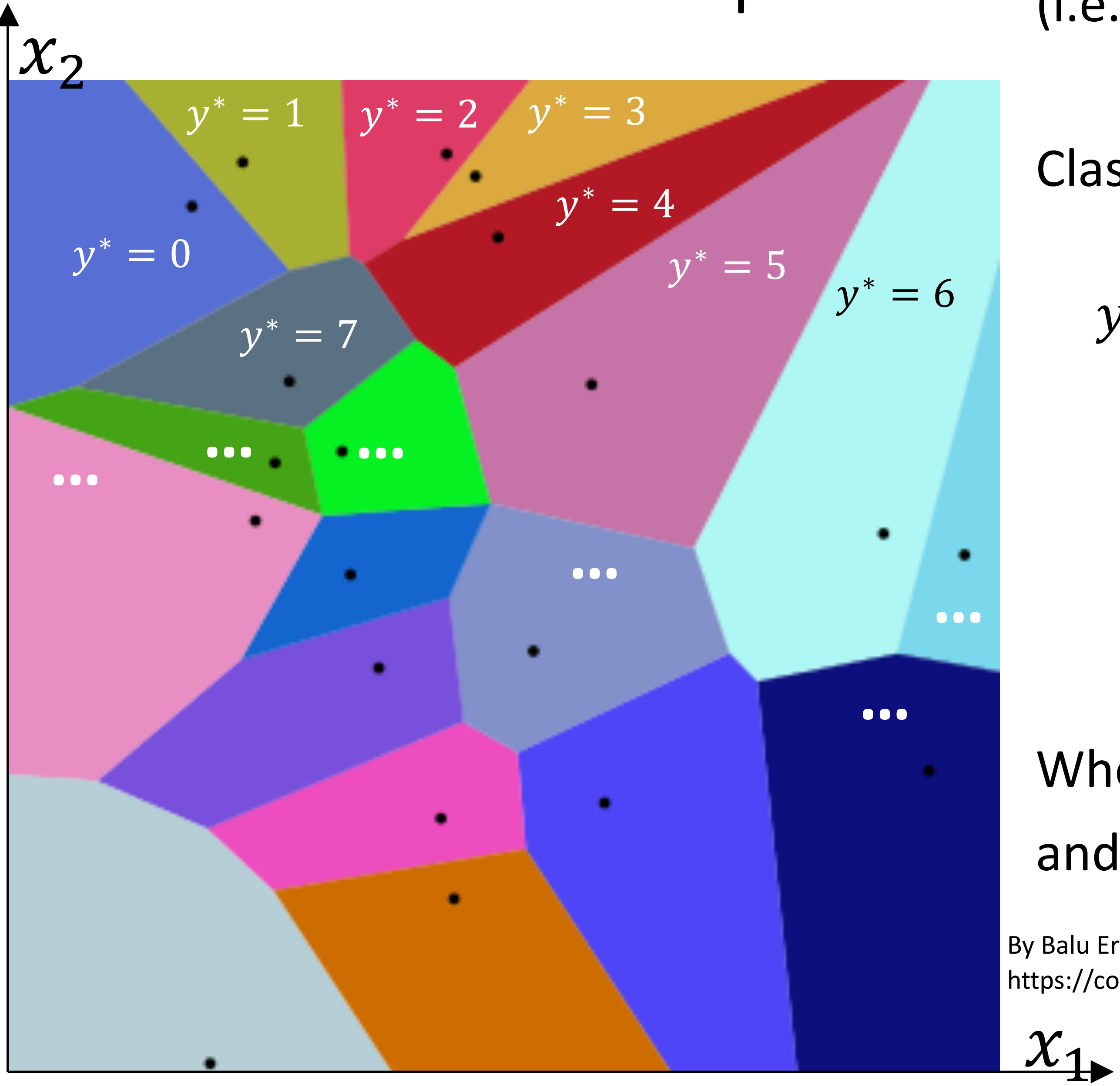
Classifier output is

$$y^* = \text{sgn}(w_1 x_1 + \ldots + w_D x_D + b)$$
$$= \text{sgn}(\vec{w}^T \vec{x})$$
$$\in \{-1,1\}$$

Where $\vec{w} = [w_1, \ldots, w_D, b]^T$

and $\vec{x} = [x_1, \ldots, x_D, 1]^T$

Multi-Class Perceptron



True class is $y \in \{0,1,2,\dots,V-1\}$
(i.e., V=vocabulary size = # of distinct classes).

Classifier output is

$$y^* = \text{argmax}_{c=0}^{V-1}(w_{c1}x_1 + \cdots + w_{cD}x_D + b_c)$$

$$= \text{argmax}_{c=0}^{V-1}(\vec{w}_c^T\vec{x})$$

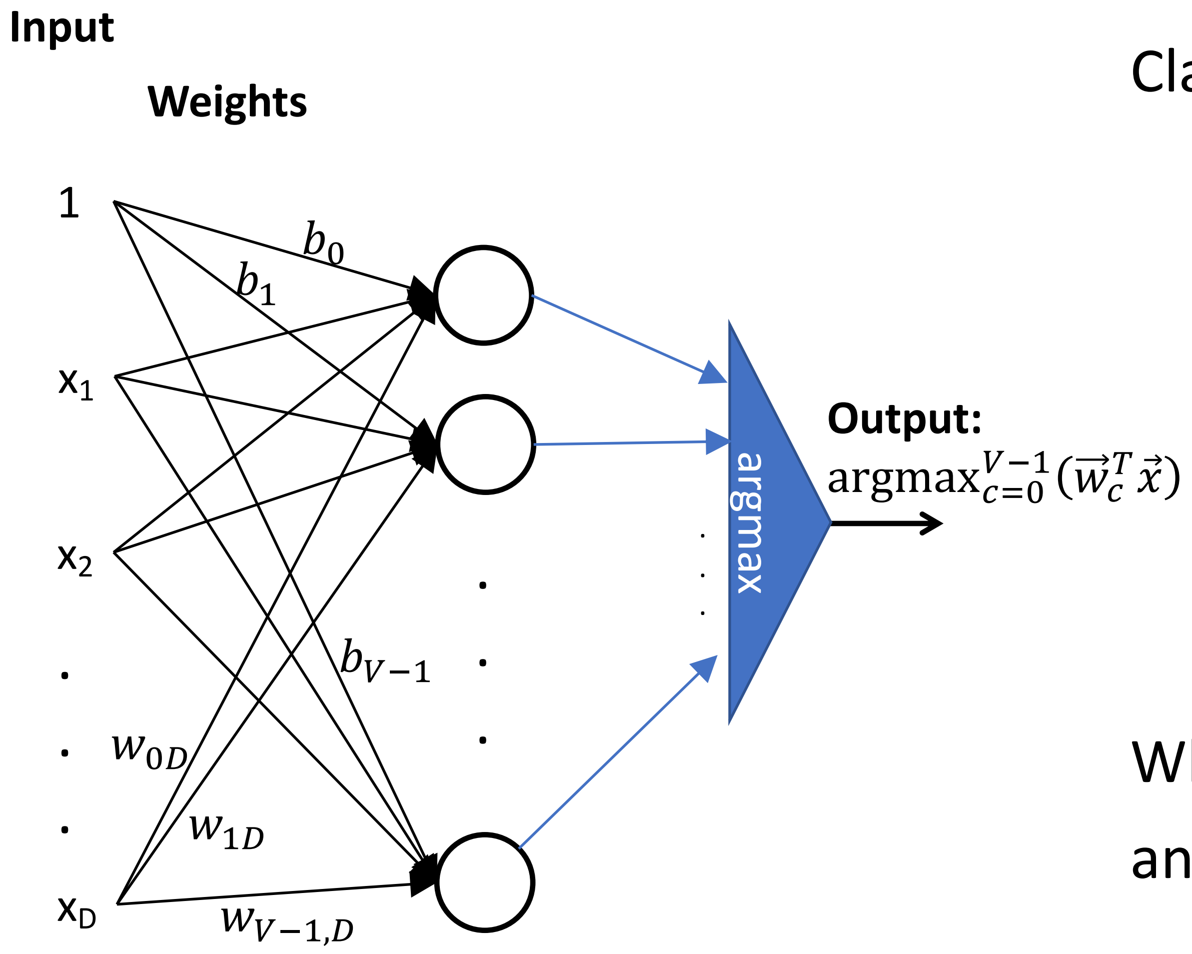
$$\in \{0,1,\dots,V-1\}$$

Where $\vec{w}_c = [w_{c1},\dots,w_{cD},b_c]^T$
and $\vec{x} = [x_1,\dots,x_D,1]^T$

Multi-Class Perceptron

True class is $y \in \{0,1,2,\ldots,V-1\}$

(i.e., V=vocabulary size = # of distinct classes).



Classifier output is

$$y^* = \text{argmax}_{c=0}^{V-1}(w_{c1}x_1 + \cdots + w_{cD}x_D + b_c)$$

$$= \text{argmax}_{c=0}^{V-1}(\vec{w}_c^T\vec{x})$$

$$\in \{0,1,\ldots,V-1\}$$

Where $\vec{w}_c = [w_{c1},\ldots,w_{cD},b_c]^T$

and $\vec{x} = [x_1,\ldots,x_D,1]^T$

Outline

- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Review: Training a Two-Class Perceptron

For each training instance $\vec{\boldsymbol{x}}$ w/ground truth label $y \in \{-1,1\}$:

- Classify with current weights: $y^* = \mathrm{sgn}(\vec{w}^T \vec{x})$
- Update weights:
  - if $y = y^*$ then do nothing
  - If $y \neq y^*$ then $\vec{w} = \vec{w} + \eta y \vec{x}$

Review: Training a Two-Class Perceptron

For each training instance $\vec{\boldsymbol{x}}$ w/ground truth label $y \in \{-1,1\}$:

- Classify with current weights: $y^* = \operatorname{sgn}(\vec{w}^T \vec{x})$
- Update weights:
  - if $y = y^*$ then do nothing
  - If $y \neq y^*$ then:
    - If $y = +1$ then set $\vec{w} = \vec{w} + \eta\vec{x}$
    - If $y = -1$ then set $\vec{w} = \vec{w} - \eta\vec{x}$

Training a Multi-Class Perceptron

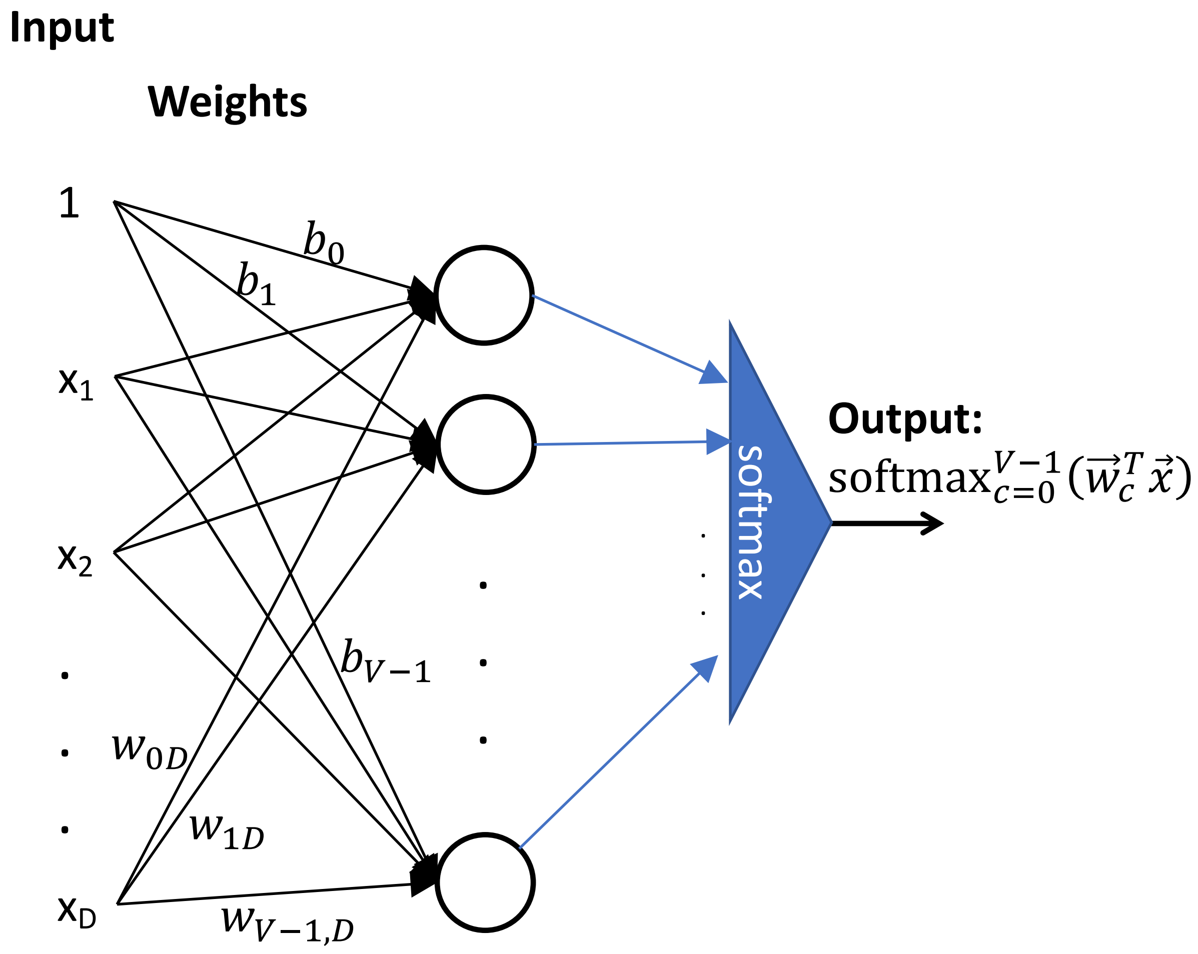
For each training instance $\vec{\boldsymbol{x}}$ w/ground truth label $y \in \{0,1, \ldots, V-1\}$:

- Classify with current weights: $y^* = \operatorname{argmax}_{c=0}^{V-1} (\vec{w}_c^T \vec{x})$
- Update weights:
  - if $y = y^*$ then do nothing
  - If $y \neq y^*$ then:
    - Update the correct-class vector as $\vec{w}_y = \vec{w}_y + \eta\vec{x}$
    - Update the wrong-class vector as $\vec{w}_{y^*} = \vec{w}_{y^*} - \eta\vec{x}$
    - **Don't change the vectors of any other class**

Outline

- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Multi-Class Logistic Regression



True class is $y \in \{0,1,2,\ldots,V-1\}$

(i.e., V=vocabulary size = # of distinct classes).

Classifier output is

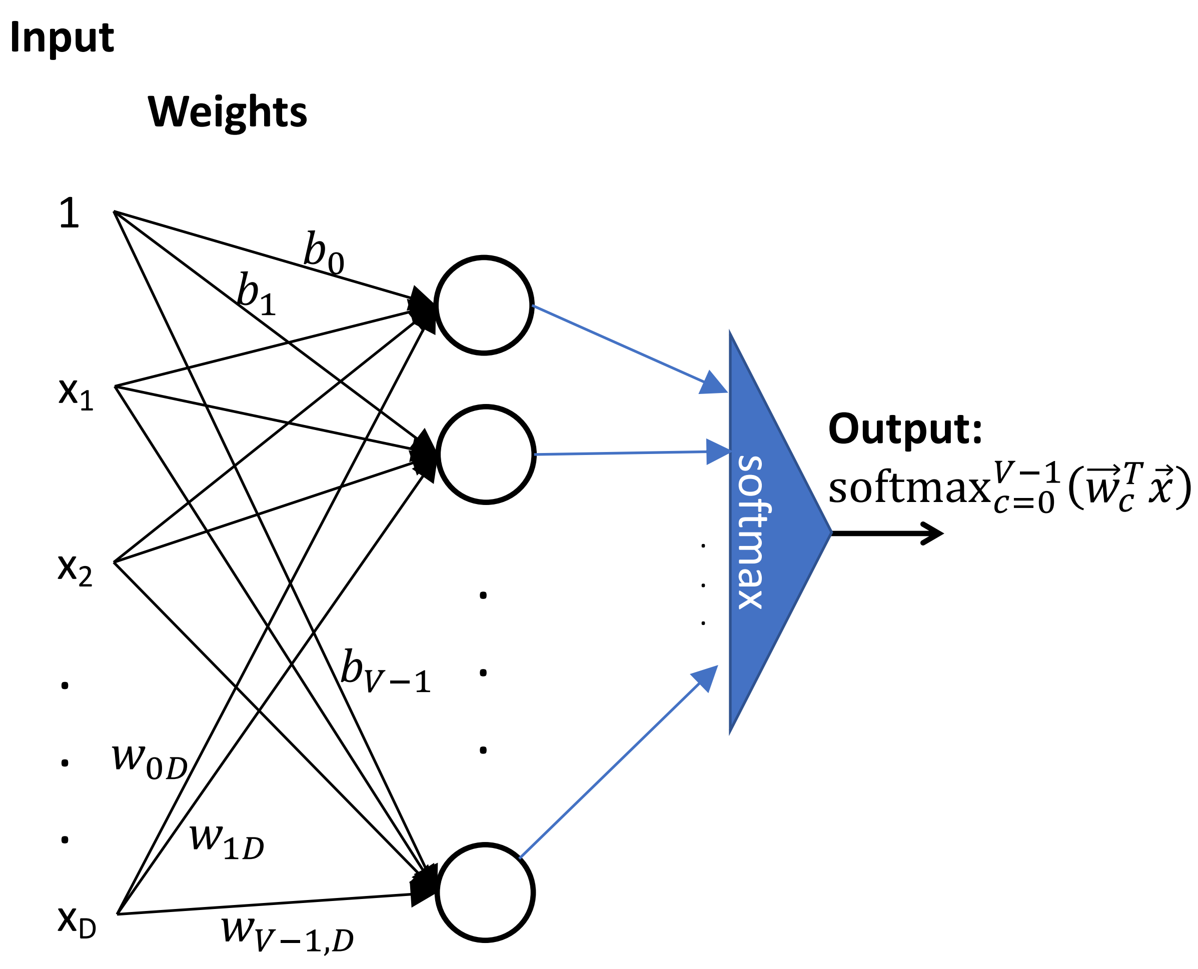
$$\vec{y}^* = \text{softmax}_{c=0}^{V-1}(\vec{w}_c^T\vec{x}) = [y_0^*,\ldots,y_{V-1}^*]$$

The “argmax” of perceptron is replaced by a “softmax.”

The “softmax” is a V-dimensional vector, each of whose elements is between 0 and 1.

If the classifier is working well, then the y^{th} element of this vector should be close to 1, and all other elements should be close to 0.

Multi-Class Logistic Regression



True class is $y \in \{0,1,2,\ldots,V-1\}$

(i.e., V=vocabulary size = # of distinct classes).

Classifier output is

$$\vec{y}^* = \text{softmax}_{c=0}^{V-1}(W\vec{x}) = [y_0^*,\ldots,y_{V-1}^*]$$

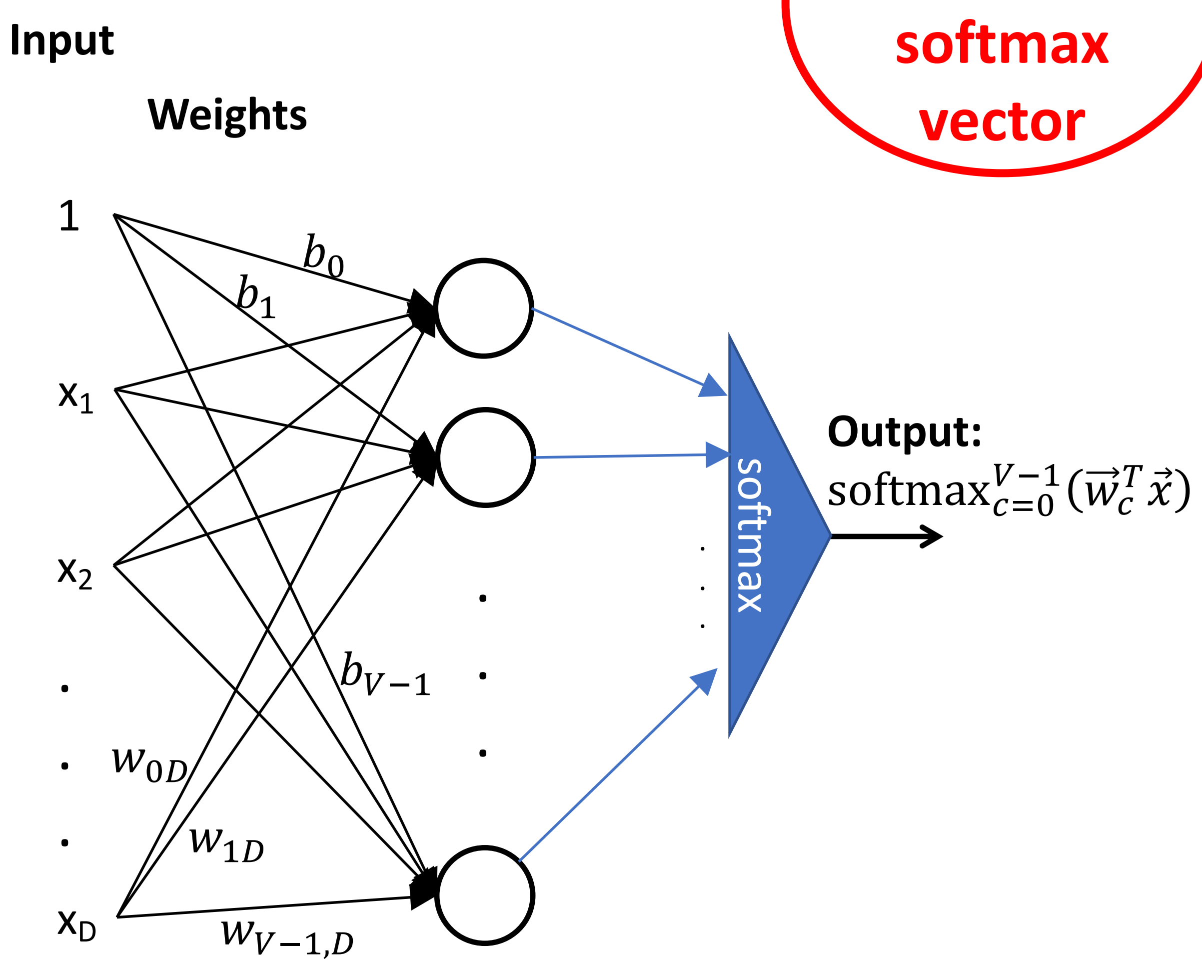
The "softmax" function is defined as follows:

$$\text{softmax}_c(W\vec{x}) = \frac{\exp(\vec{w}_c^T\vec{x})}{\sum_{j=0}^{V-1}\exp\left(\vec{w}_j^T\vec{x}\right)}$$

where W is the weight matrix whose $(c,d)^{th}$ element is w_{cd}.

The vector $\vec{w}_c = [w_{c1},\ldots,w_{cD},b_c]^T$

One-Hot Vectors



Classifier output: softmax vector

Let's redefine the "ground truth" label, so it's easier to train the softmax function.

The softmax output is

$$\vec{y}^* = [y_0^*, \dots, y_{V-1}^*]$$

where $0 < y_c^* < 1$ and $\sum_{j=0}^{V-1} y_j^* = 1$.

Let's redefine the "ground truth" label so it has the same format. Let's define

$$\vec{y} = [y_0, \dots, y_{V-1}]$$

where

- $y_j = 1$ if j is the correct class
- $y_j = 0$ otherwise

Ground truth: one-hot vector

This is called a ONE-HOT VECTOR.

One-Hot Vector

- Example: if the example is from class 1, then $\vec{y} = [0,1,0]$

$$y_j = \begin{cases} 1 & \text{example is from class j} \\ 0 & \text{example is NOT from class j} \end{cases}$$

Call y_j the **reference label**, and call y_j^* the **hypothesis**. Then notice that:

- $y_j =$ True value of $P(class = j \mid \vec{x})$, because the true probability is always either 1 or 0!
- $y_j^* =$ Estimated value of $P(class = j \mid \vec{x})$, $0 < y_j^* < 1$, $\sum_{j=1}^{c} y_j^* = 1$

Comparing the argmax and the softmax

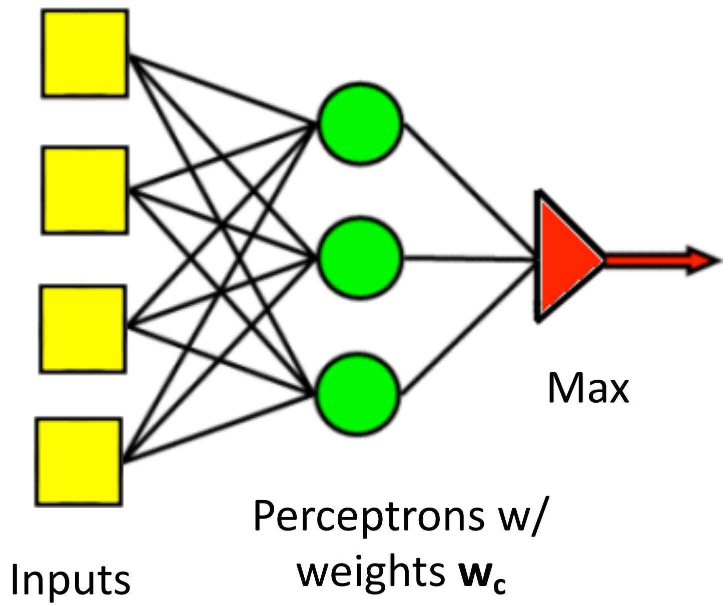
The multi-class perceptron calculates

$$y^* = \text{argmax}_{c=0}^{V-1}(\vec{w}_c^T\vec{x})$$

The multi-class logistic regression calculates

$$y_c^* = \text{softmax}_c(W\vec{x}) = \frac{\exp(\vec{w}_c^T\vec{x})}{\sum_{j=0}^{V-1}\exp\left(\vec{w}_j^T\vec{x}\right)}$$

How do these two things compare?

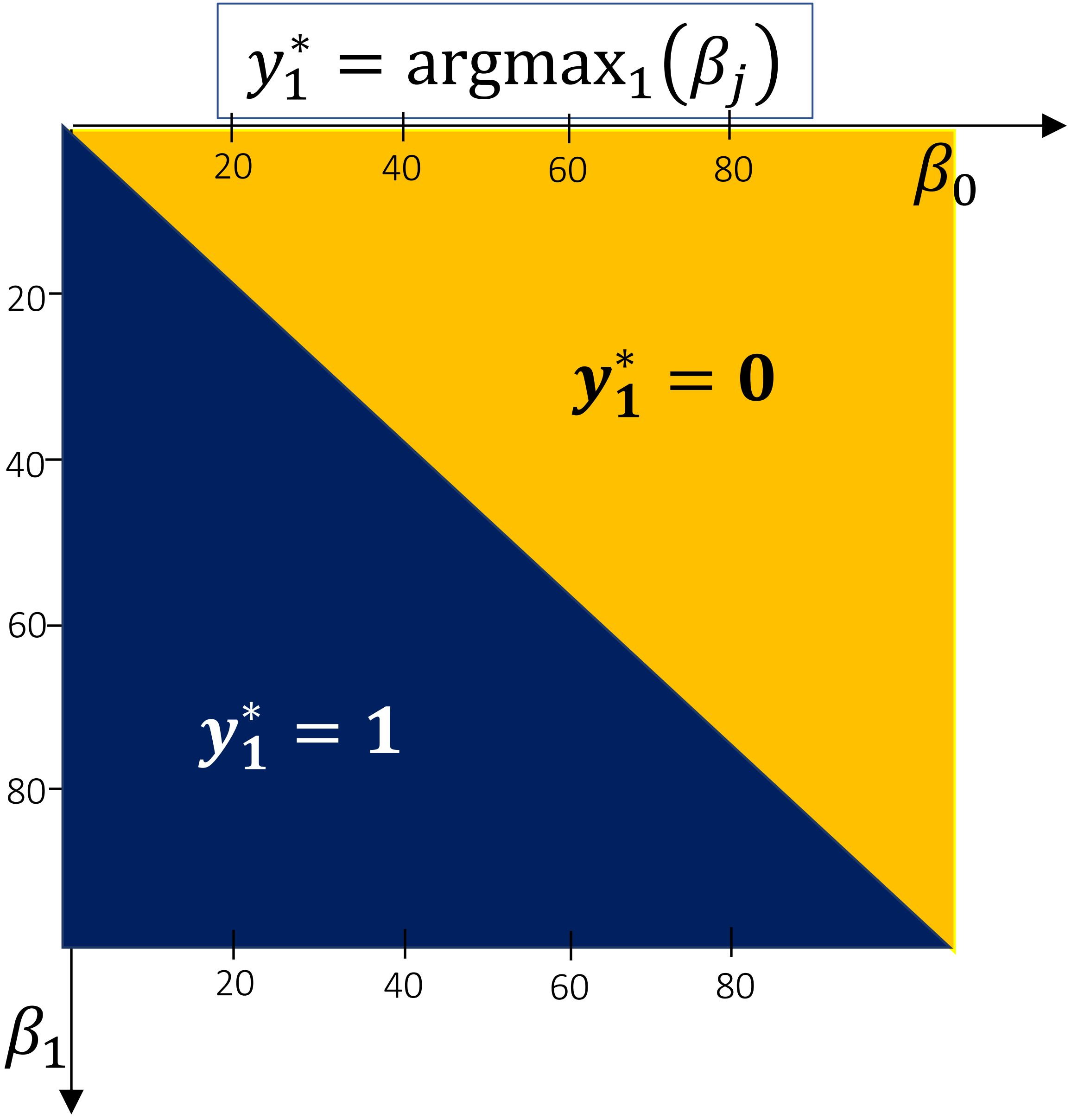


Comparing the argmax and softmax

Here's the second term in a two-class argmax function:

$$y_1^* = \begin{cases} 1 & \text{if } \beta_1 = \text{argmax}_{j=0}^{1}(\beta_j) \\ 0 & \text{otherwise} \end{cases}$$

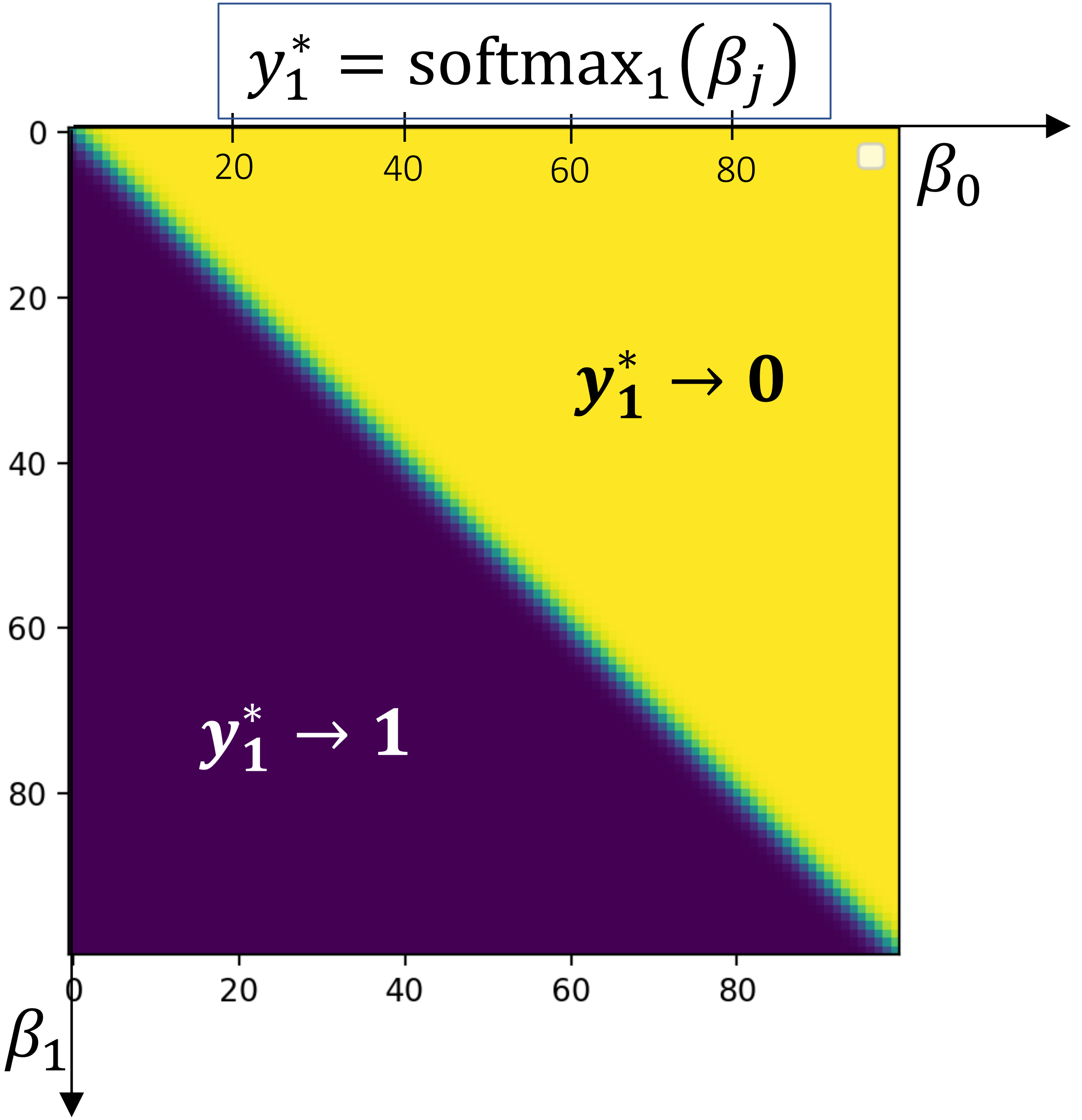
Where I'm using the abbreviation $\beta_j = \vec{w}_j^T \vec{x}$



Comparing the argmax and softmax

Here's the second term in a two-class softmax function:

$$y_1^* = \frac{\exp(\beta_1)}{\sum_{j=0}^{1} \exp(\beta_j)}$$



Outline

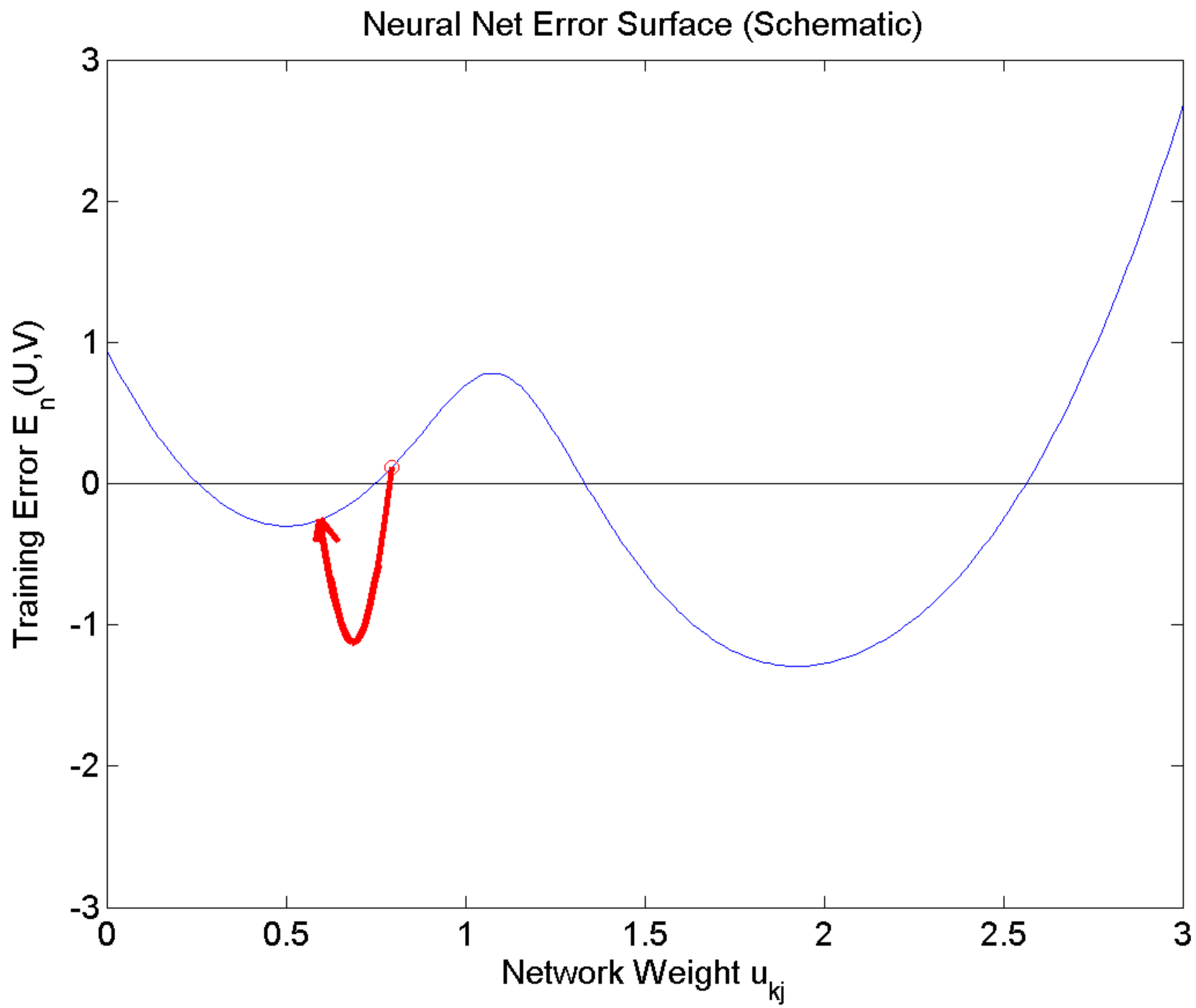
- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Training a Softmax Neural Network

We want to train the neural network to represent a training database as well as possible. If we can define the training error to be some function L, then we want to update the weights according to

$$w_{cd} = w_{cd} - \eta \frac{dL}{dw_{cd}}$$

So what is L?



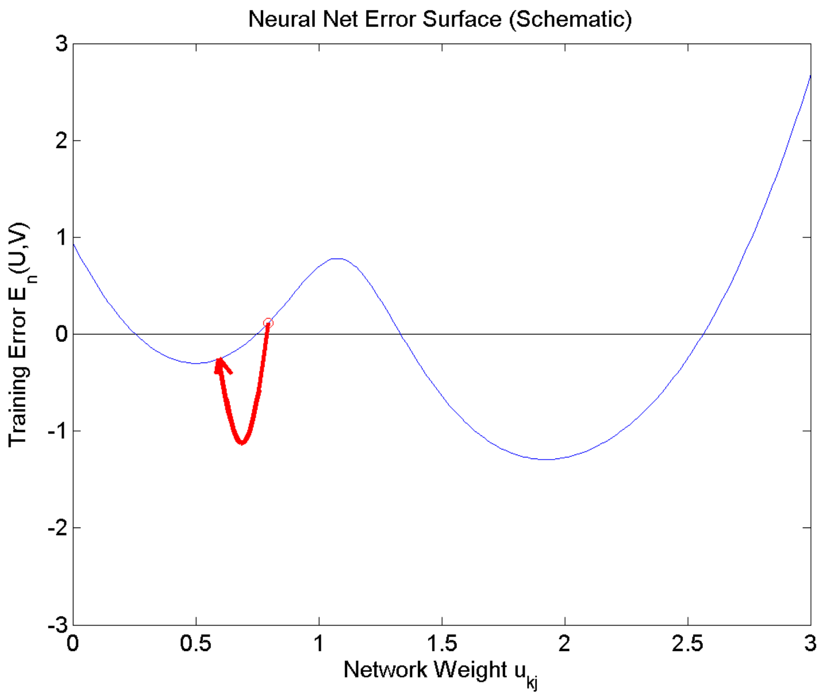
Training: Maximize the probability of the training data

Remember, the whole point of that denominator in the softmax function is that it allows us to use softmax as

$$y_j^* = \text{Estimated value of } P(\text{class} = j \mid \vec{x})$$

Suppose we decide to estimate the network weights w_{cd} in order to maximize the probability of the training database, in the sense of

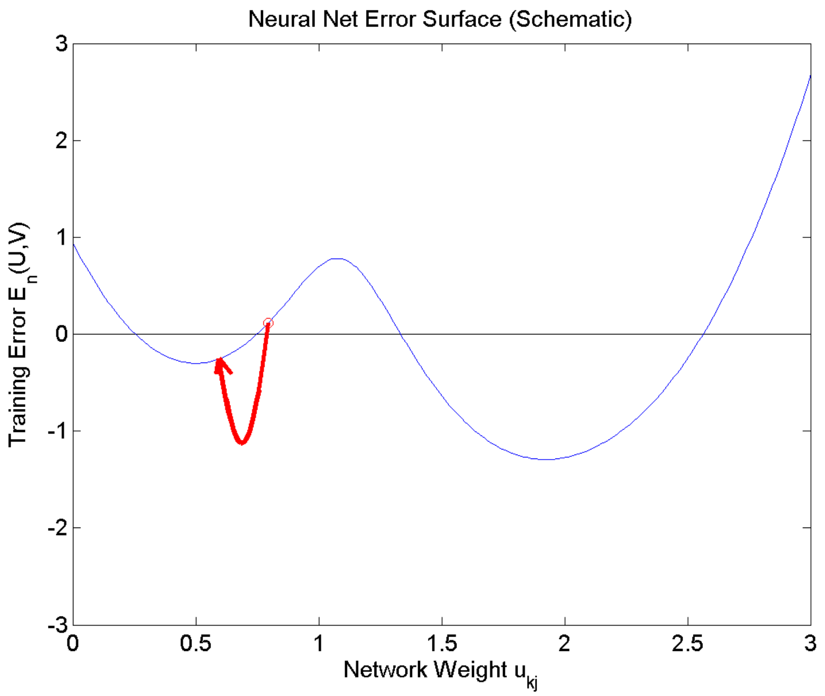
$$w_{cd} = \underset{W}{\operatorname{argmax}}\, P(\text{training labels} \mid \text{training feature vectors})$$



Training: Maximize the probability of the training data

Remember, the whole point of that denominator in the softmax function is that it allows us to use softmax as

$$y_j^* = \text{Estimated value of } P(\text{class} = j \mid \vec{x})$$



If we assume the training tokens are independent, this is:

$$w_{cd} = \underset{W}{\operatorname{argmax}} \prod_{i=1}^{n} P(\text{reference label of the } i^{th} \text{token} \mid i^{th} \text{feature vector})$$

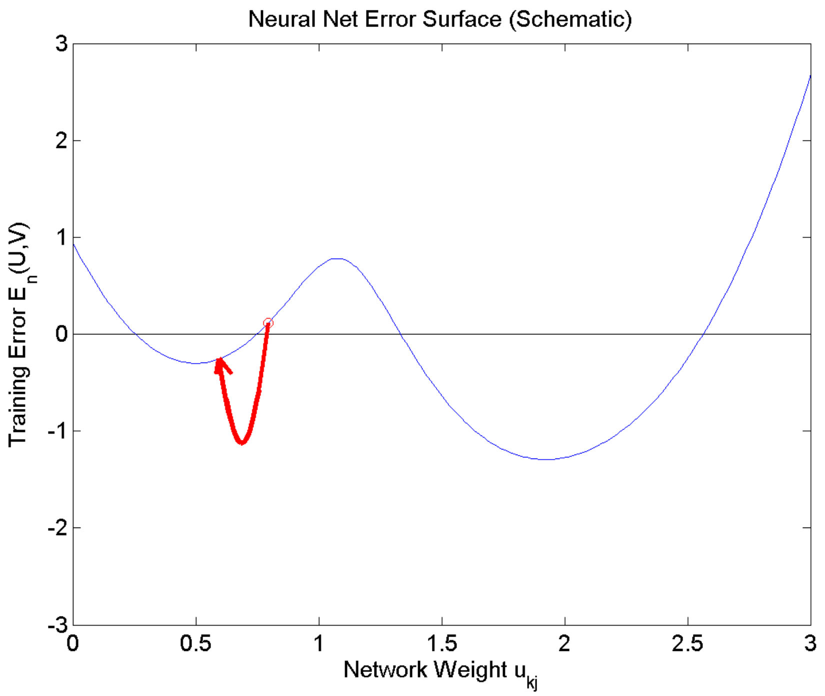
Training: Maximize the probability of the training data

Remember, the whole point of that denominator in the softmax function is that it allows us to use softmax as

$$y_j^* = \text{Estimated value of } P(\text{class} = j \mid \vec{x})$$

OK. We need to create some notation to mean "the reference label for the i^{th} token." Let's call it $j(i)$.

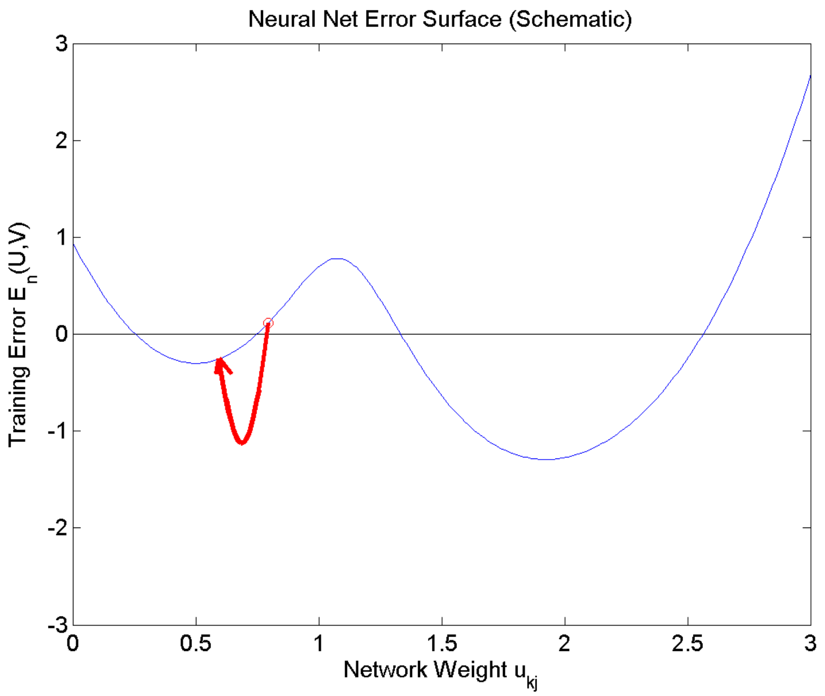
$$w_{cd} = \underset{W}{\operatorname{argmax}} \prod_{i=1}^{n} P(\text{class} = j(i) \mid \vec{x})$$



Training: Maximize the probability of the training data

Wow, Cool!! So we can maximize the probability of the training data by just picking the softmax output corresponding to the **<u>correct class</u>** $j(i)$, for each token, and then multiplying them all together:

$$w_{cd} = \underset{W}{\operatorname{argmax}} \prod_{i=1}^{n} y^*_{j(i)}$$



So, hey, let's take the logarithm, to get rid of that nasty product operation.

$$w_{cd} = \underset{W}{\operatorname{argmax}} \sum_{i=1}^{n} \ln y^*_{j(i)}$$

Training: Minimizing the negative log probability

Softmax neural networks are almost always trained in order to minimize the negative log probability of the training data:

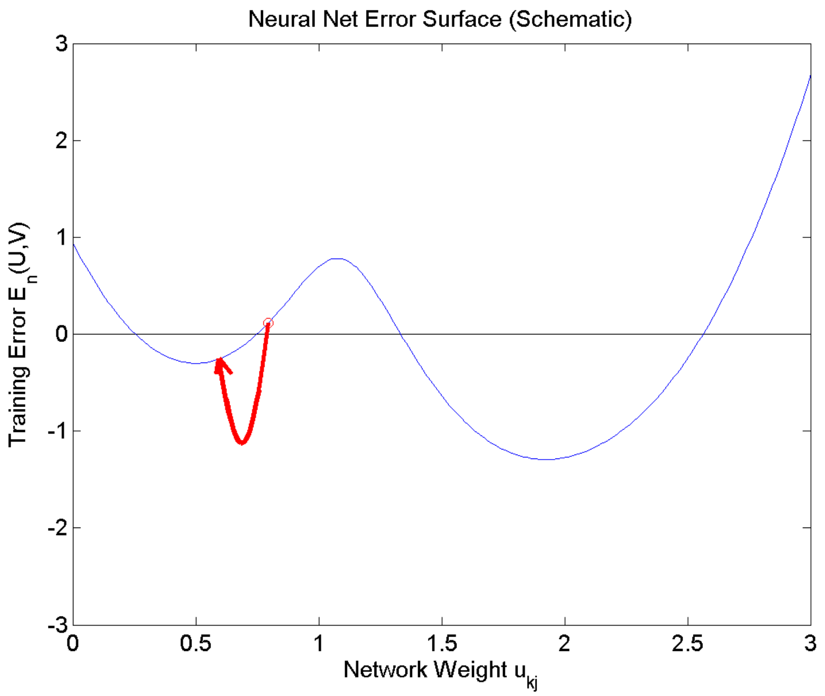
$$w_{cd} = \underset{W}{\operatorname{argmin}} L$$

$$L = \sum_{i=1}^{n} -\ln y^*_{j(i)}$$

This loss function is called the **cross-entropy loss**. Cross-entropy is a measure of dissimilarity between two probability distributions. In this case, we're minimizing the dissimilarity between the true and estimated classes:

$$y_j = \text{True } P(class = j \mid \vec{x}) = \begin{cases} 1 & j = j(i) \\ 0 & \text{otherwise} \end{cases}$$

$$y_j^* = \text{Estimated } P(class = j \mid \vec{x}) = \frac{\exp\left(\vec{w}_j^T \vec{x}\right)}{\sum_{k=0}^{V-1} \exp\left(\vec{w}_k^T \vec{x}\right)}$$



Outline

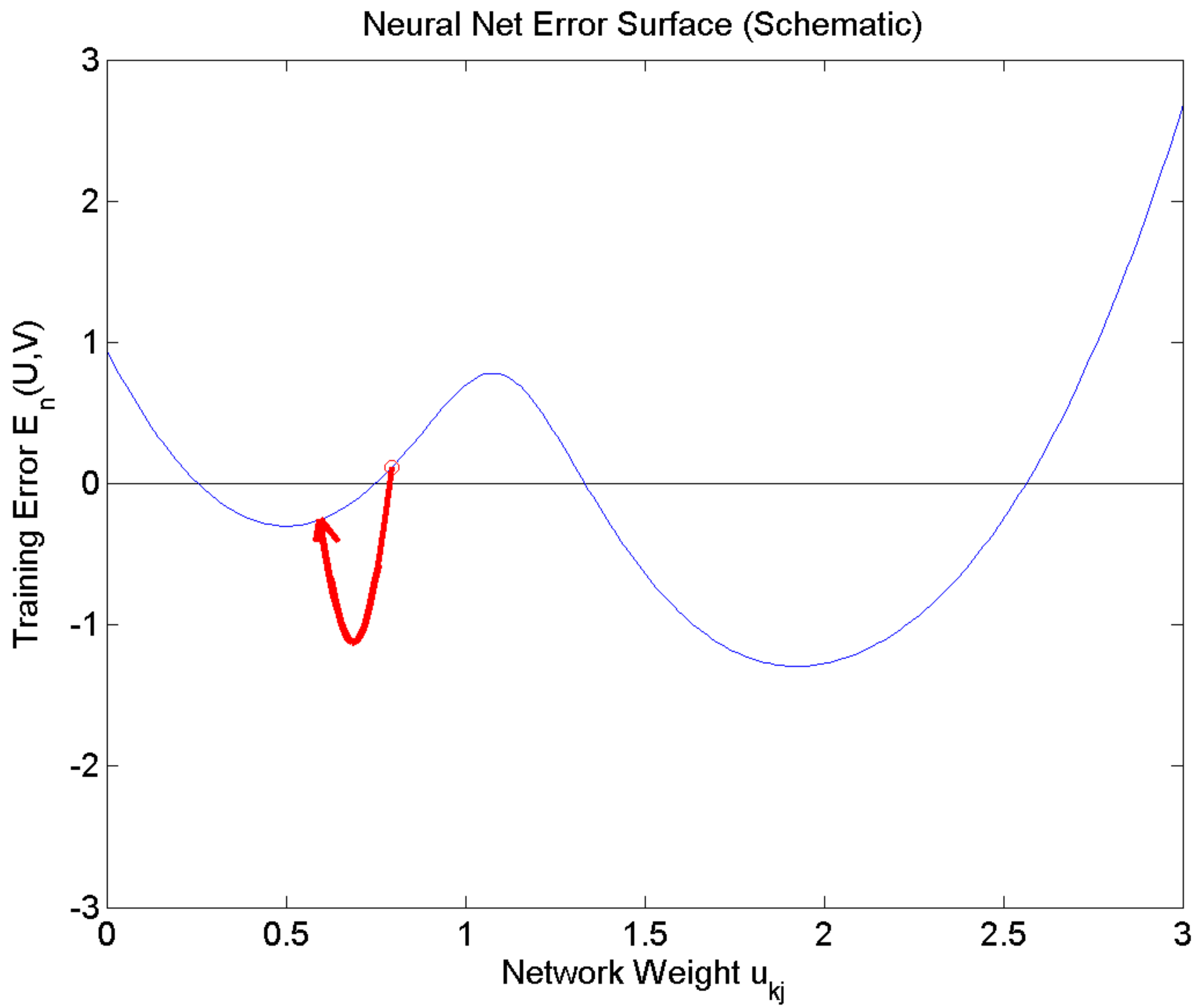
- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Training a Softmax Neural Network

We want to train the neural network to represent a training database as well as possible. If we can define the training error to be some function L, then we want to update the weights according to

$$w_{cd} = w_{cd} - \eta \frac{dL}{dw_{cd}}$$

So what is $\frac{dL}{dw_{cd}}$?



Differentiating the cross-entropy

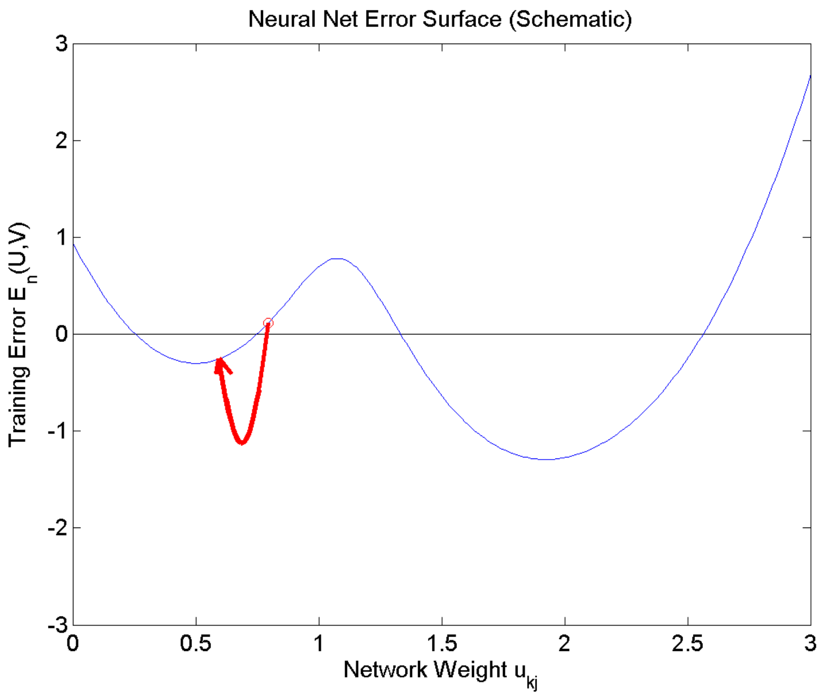
The cross-entropy loss function is:

$$L = \sum_{i=1}^{n} -\ln y^*_{j(i)}$$

Let's try to differentiate it:

$$\frac{dL}{dw_{cd}} = \sum_{i=1}^{n} -\left(\frac{1}{y^*_{j(i)}}\right)\frac{dy^*_{j(i)}}{dw_{cd}}$$

So what is $\frac{dy^*_{j(i)}}{dw_{cd}}$?



Differentiating the cross-entropy

The cross-entropy loss function is:

$$y^*_{j(i)} = \text{softmax}_j(\beta_k) = \frac{\exp(\beta_{j(i)})}{\sum_{k=0}^{V-1}\exp(\beta_k)}$$

Let's try to differentiate it:

$$\frac{dy^*_{j(i)}}{dw_{cd}} = \left(\frac{1}{\sum_{k=0}^{V-1}\exp(\beta_k)}\right)\left(\frac{d\exp(\beta_{j(i)})}{dw_{cd}}\right) - \left(\frac{\exp(\beta_{j(i)})}{(\sum_{k=0}^{V-1}\exp(\beta_k))^2}\right)\left(\frac{d\left(\sum_{l=0}^{V-1}\exp(\beta_l)\right)}{dw_{cd}}\right)$$

$$= \left(\frac{\exp(\beta_{j(i)})}{\sum_{j=0}^{V-1}\exp(\beta_k)}\right)\left(\frac{d\beta_{j(i)}}{dw_{cd}}\right) - \sum_{l=0}^{V-1}\left(\frac{\exp\left(\beta_{j(i)}\right)\exp(\beta_l)}{(\sum_{k=0}^{V-1}\exp(\beta_k))^2}\right)\left(\frac{d\beta_l}{dw_{cd}}\right)$$

$$= y^*_{j(i)}\left(\frac{d\beta_{j(i)}}{dw_{cd}}\right) - \sum_{l=0}^{V-1} y^*_{j(i)} y^*_l \left(\frac{d\beta_l}{dw_{cd}}\right) = (y^*_{j(i)} y_c - y^*_{j(i)} y^*_c) x_d$$

Where the last line uses $\beta_j = \vec{w}_j^T \vec{x}$, and therefore $\frac{d\beta_j}{dw_{cd}} = \begin{cases} x_d & \text{if } j = c \\ 0 & \text{otherwise} \end{cases}$

Putting it all together…

$$w_{cd} = w_{cd} - \eta \frac{dL}{dw_{cd}}$$

$$= w_{cd} + \eta \sum_{i=1}^{n} \left(\frac{1}{y_{j(i)}^*} \right) \frac{dy_{j(i)}^*}{dw_{cd}}$$

$$= w_{cd} + \eta \sum_{i=1}^{n} (y_c - y_c^*) x_d$$

where

$$y_c = \text{True } P(class = c \mid \vec{x}) = \begin{cases} 1 & c = j(i) \\ 0 & \text{otherwise} \end{cases}$$

$$y_c^* = \text{Estimated } P(class = c \mid \vec{x}) = \frac{\exp(\vec{w}_c^T \vec{x})}{\sum_{k=0}^{V-1} \exp(\vec{w}_k^T \vec{x})}$$

Training Multi-Class Logistic Regression

Putting it all together, we wind up with a surprisingly simple result:

$$\vec{w}_c = \vec{w}_c + \eta \sum_{i=1}^{n} (y_c - y_c^*)\, \vec{x}$$

where $y_c = 1$ if and only if the i'th token is of class c. In other words,

- If c is the correct class, but $y_c^* \approx 0$, then $\vec{w}_c = \vec{w}_c + \eta\vec{x}$
- If $y_c^* \approx 1$, but c is the wrong class, then $\vec{w}_c = \vec{w}_c - \eta\vec{x}$

Outline

- Multi-Class Perceptron
  - Testing
  - Training
- Multi-Class Logistic Regression
  - Testing: softmax function
  - Training: cross-entropy training criterion
  - Training: how to differentiate the softmax
- Comparing Multi-Class Perceptron and Logistic Regression

Training a Multi-Class Perceptron

For each training instance $\vec{\boldsymbol{x}}$ w/ground truth label $y \in \{0,1,\ldots,V-1\}$:

- Classify with current weights: $y^* = \text{argmax}_{c=0}^{V-1}(\vec{w}_c^T\vec{x})$
- Update weights:
  - if $y = y^*$ then do nothing
  - If $y \neq y^*$ then:
    - Update the correct-class vector as $\vec{w}_y = \vec{w}_y + \eta\vec{x}$
    - Update the wrong-class vector as $\vec{w}_{y^*} = \vec{w}_{y^*} - \eta\vec{x}$

Training Multi-Class Logistic Regression

Putting it all together, we wind up with a surprisingly simple result:

$$\vec{w}_c = \vec{w}_c + \eta \sum_{i=1}^{n} (y_c - y_c^*)\, \vec{x}$$

where $y_c = 1$ if and only if the i'th token is of class c. In other words,

- If c is the correct class, but $y_c^* \approx 0$, then $\vec{w}_c = \vec{w}_c + \eta\vec{x}$
- If $y_c^* \approx 1$, but c is the wrong class, then $\vec{w}_c = \vec{w}_c - \eta\vec{x}$

Conclusion: Comparing Multi-Class Perceptron and Logistic Regression

Perceptron: If classifier output is incorrect, then:

- Update the correct-class, y, as $\vec{w}_y = \vec{w}_y + \eta\vec{x}$
- Update the wrong-class, y^*, as $\vec{w}_{y^*} = \vec{w}_{y^*} - \eta\vec{x}$

Logistic Regression: for **every** class c,

$$\vec{w}_c = \vec{w}_c + \eta(y_c - y_c^*)\vec{x}$$

$$\approx \begin{cases} \vec{w}_c + \eta\vec{x} & \text{if } c \text{ is the correct class } (y_c = 1) \text{ and } y_c^* \approx 0 \\ \vec{w}_c - \eta\vec{x} & \text{if } c \text{ is the wrong class } (y_c = 0) \text{ and } y_c^* \approx 1 \end{cases}$$

Conclusion: they're almost exactly the same thing! The main difference is that, for logistic regression, $0 < y_c^* < 1$: it's never exactly equal to either 0 or 1.