Polychotomizers: One-Hot Vectors, Softmax, and Cross-Entropy

Outline

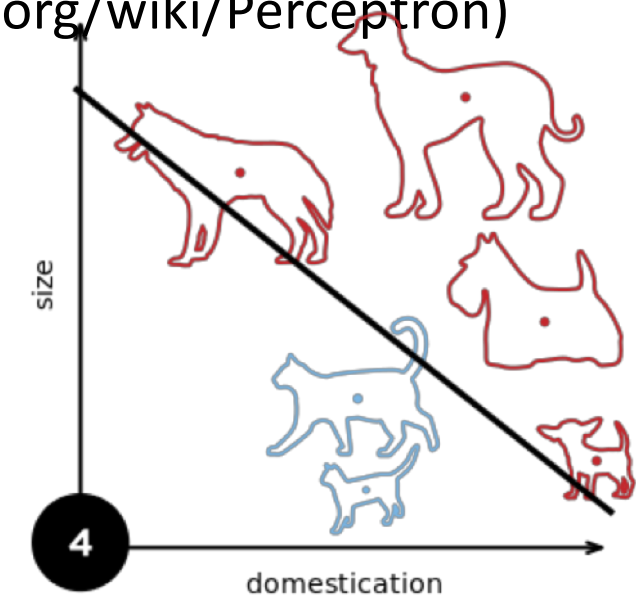
- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function
  - A differentiable approximate argmax
  - How to differentiate the softmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function
  - A differentiable approximate argmax
  - How to differentiate the softmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

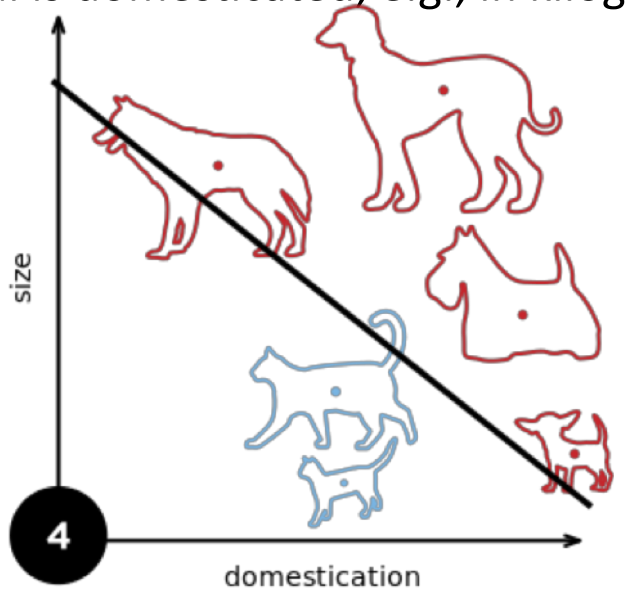
Dichotomizer: What is it?

- Dichotomizer = a two-class classifier
  - From the Greek, dichotomos = “cut in half”
  - First known use of this word, according to Merriam-Webster: 1606
- Example: a classifier that decides whether an animal is a dog or a cat (Elizabeth Goodspeed, 2015 https://en.wikipedia.org/wiki/Perceptron)



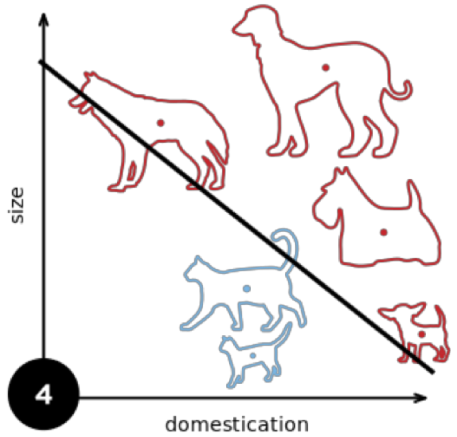
Dichotomizer: Example

- Dichotomizer = a two-class classifier
- Input to the dichotomizer: a feature vector, $\vec{f}$
- Example: $\vec{f} = [f_1, f_2]$
  - f_1 = degree to which the animal is domesticated, e.g., comes when called
  - f_2 = size of the animal is domesticated, e.g., in kilograms



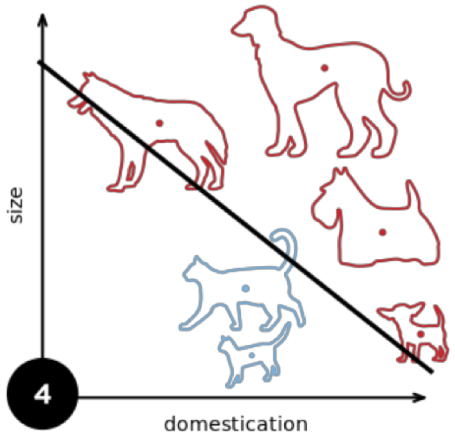
Dichotomizer: Example

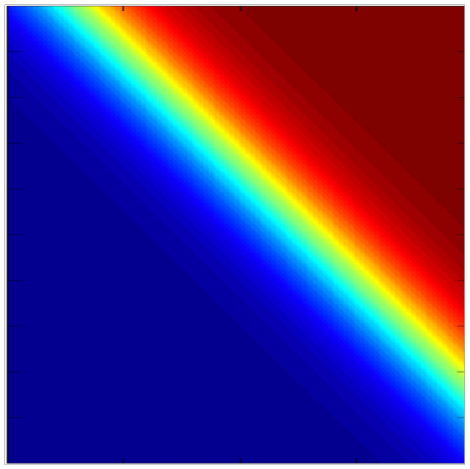
- Dichotomizer = a two-class classifier
- Input to the dichotomizer: a feature vector, $\vec{f}$
- Output of the dichotomizer: $\hat{y} = P(class\ 1|\ \vec{f}\ ), 0 \leq \hat{y} \leq 1$
  - For example, we could say class 1 = "dog"
  - Class 0 = "cat" (or we could call it class 2, or class -1, or whatever. Everybody agrees that one of the two classes is called "class 1," but nobody agrees on what to call the other class. Since there's only two classes, it doesn't really matter.



Linear Dichotomizer

- Dichotomizer = a two-class classifier
- Input to the dichotomizer: a feature vector, $\vec{f}$
- Output of the dichotomizer: $\hat{y} = P(class\ 1|\ \vec{f}\ ), 0 \leq \hat{y} \leq 1$
- A "linear dichotomizer" is one in which $\hat{y}$ varies along a straight line:





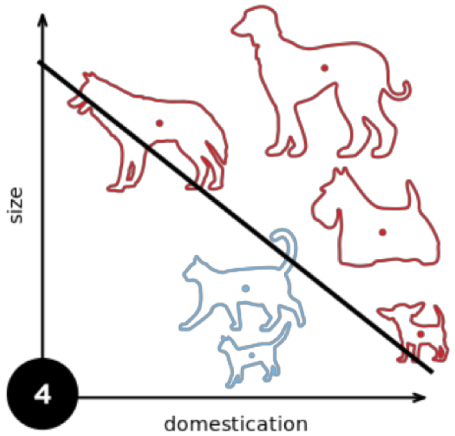
$\hat{y} = 1$
Up here

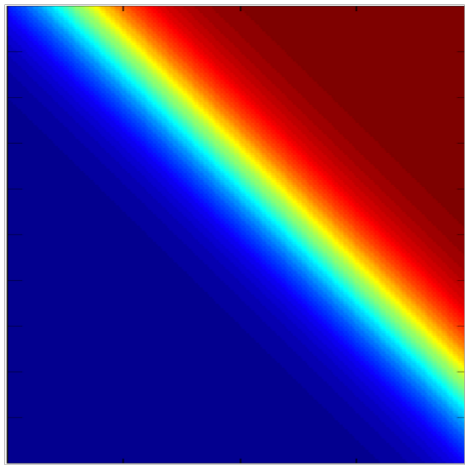
Along the middle:
$0 < \hat{y} < 1$

$\hat{y} = 0$
Down here

Training a Dichotomizer

- Training database = n training tokens
- Example: n=6 training examples





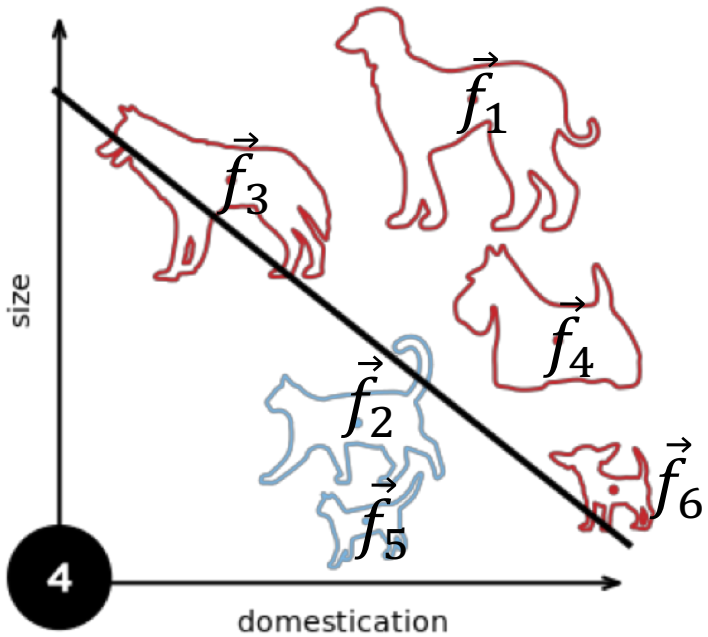
$\hat{y} = 1$
Up here

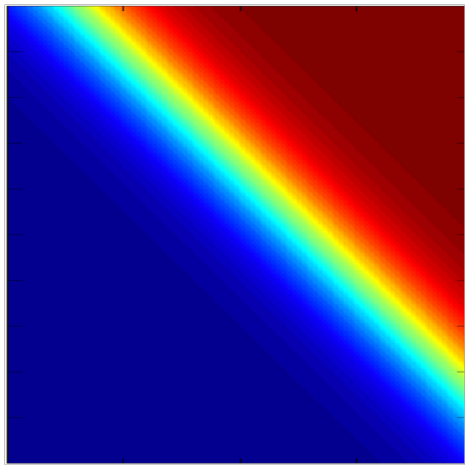
Along the middle:
$0 < \hat{y} < 1$

$\hat{y} = 0$
Down here

Training a Dichotomizer

- Training database = n training tokens
- n training feature vectors: $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_n\}$
- Each feature vector has d features: $\vec{f}_i = [f_{i1}, \ldots, f_{id}]$
  - Example: d=2 features per training example





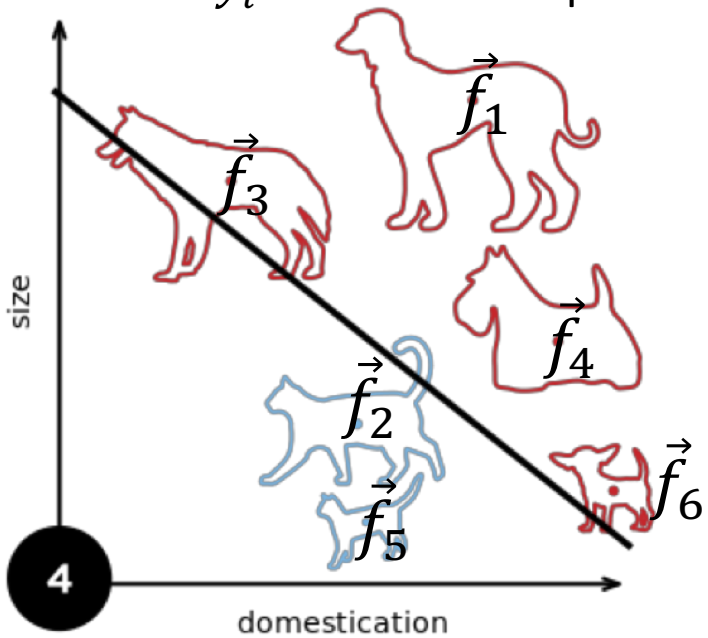
$\hat{y} = 1$
Up here

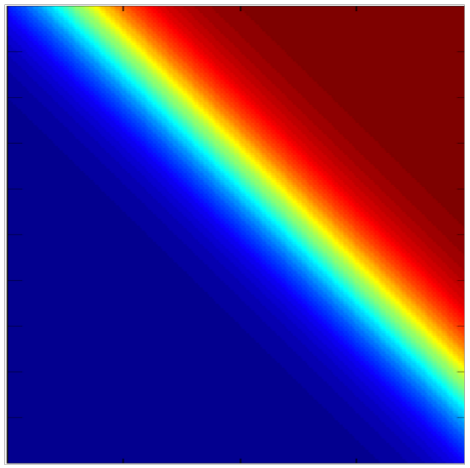
Along the middle:
$0 < \hat{y} < 1$

$\hat{y} = 0$
Down here

Training a Dichotomizer

- Training database = n training tokens
- n training feature vectors: $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_n\}, \quad \vec{f}_i = [f_{i1}, \ldots, f_{id}]$
- n "ground truth" labels: $\{y_1, y_2, \ldots, y_n\}$
  - $y_i = 1$ if i[th] example is from class 1
  - $y_i = 0$ if i[th] example is NOT from class 1





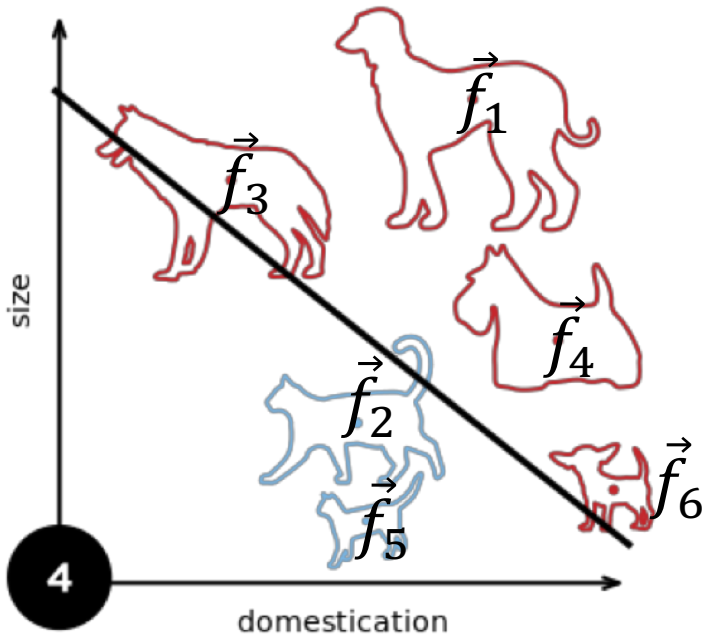
$\hat{y} = 1$
Up here

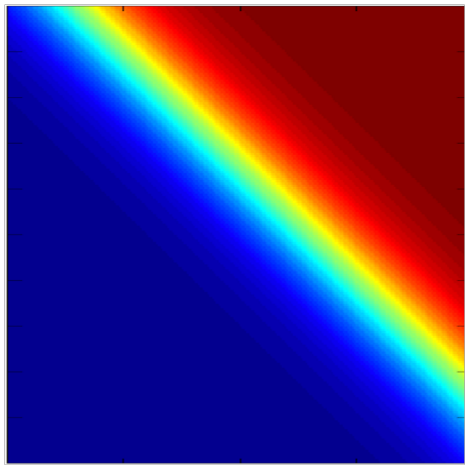
Along the middle:
$0 < \hat{y} < 1$

$\hat{y} = 0$
Down here

Training a Dichotomizer

- Training database = n training tokens
- n training feature vectors: $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_n\}, \quad \vec{f}_i = [f_{i1}, \ldots, f_{id}]$
- n "ground truth" labels: $\{y_1, y_2, \ldots, y_n\}$
- Example: $\{y_1, y_2, \ldots, y_n\} = \{1,0,1,1,0,1\}$





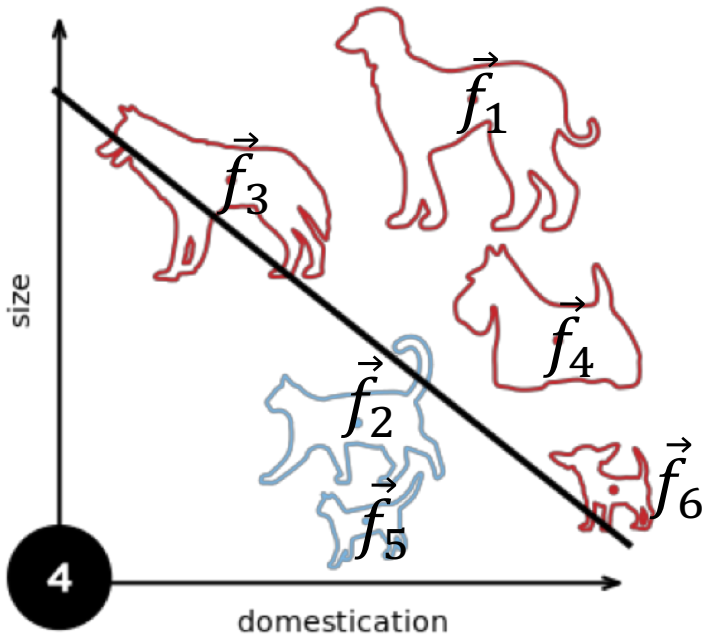
$\hat{y} = 1$
Up here

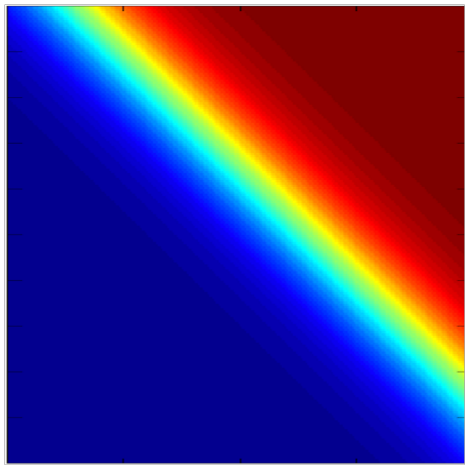
Along the middle:
$0 < \hat{y} < 1$

$\hat{y} = 0$
Down here

Training a Dichotomizer

- Training database: $\mathfrak{D} = \{\vec{f}_1, y_1, \vec{f}_2, y_2, \ldots, \vec{f}_n, y_n\}$
- n training feature vectors: $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_n\}, \quad \vec{f}_i = [f_{i1}, \ldots, f_{id}]$
- n "ground truth" labels: $\{y_1, y_2, \ldots, y_n\}$





$\hat{y} = 1$
Up here

Along the middle:
$0 < \hat{y} < 1$

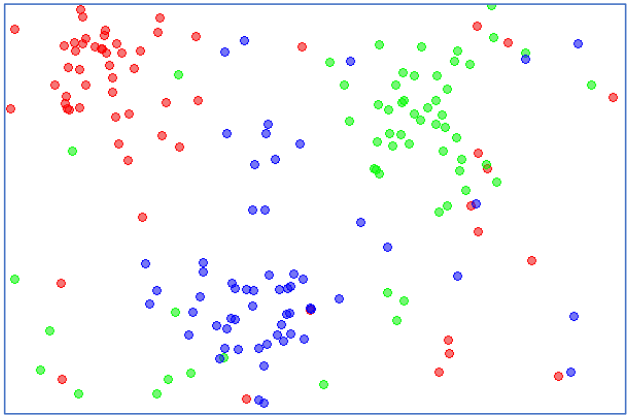
$\hat{y} = 0$
Down here

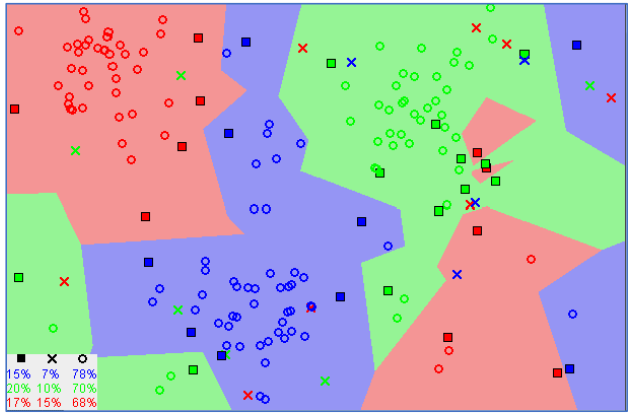
Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function
  - A differentiable approximate argmax
  - How to differentiate the softmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

Polychotomizer: What is it?

- Polychotomizer = a multi-class classifier
  - From the Greek, poly = “many”
- Example: classify dots as being purple, red, or green (E.M. Mirkes, KNN and Potential Energy applet, 2011, CC-BY 3.0, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)

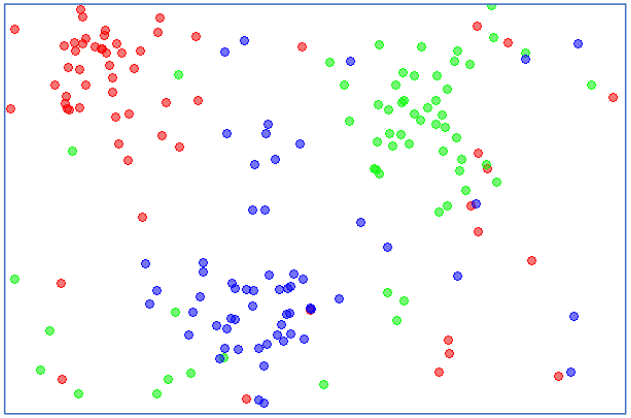


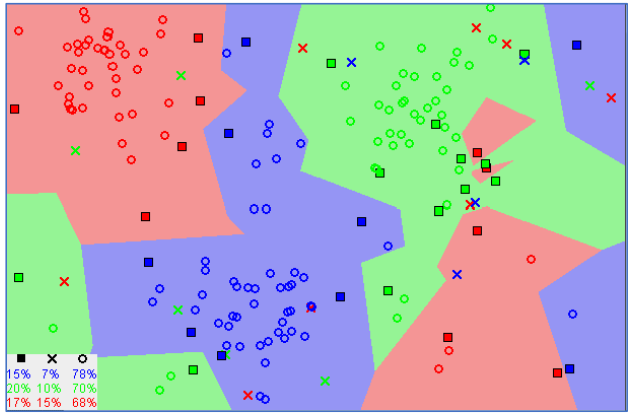


Polychotomizer: What is it?

- Polychotomizer = a multi-class classifier
- Input to the dichotomizer: a feature vector, $\vec{f} = [f_1, \ldots, f_d]$
- Output: a label vector, $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_c]$
  - $\hat{y}_j = P(\ class\ j \mid \vec{f})$
  - Example: c=3 possible class labels, so you could define

$\hat{y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3] = [P(purple|\vec{f}), P(red|\vec{f}), P(green|\vec{f})]$

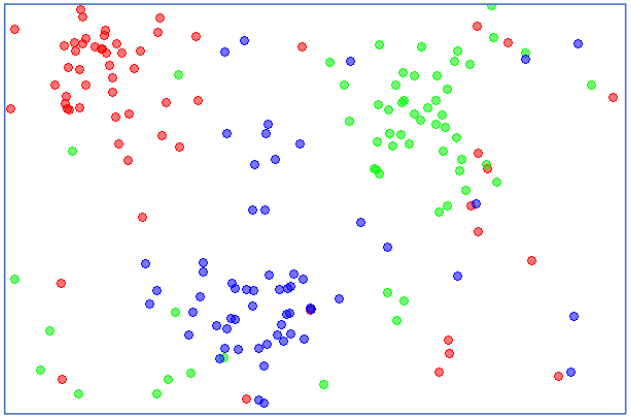


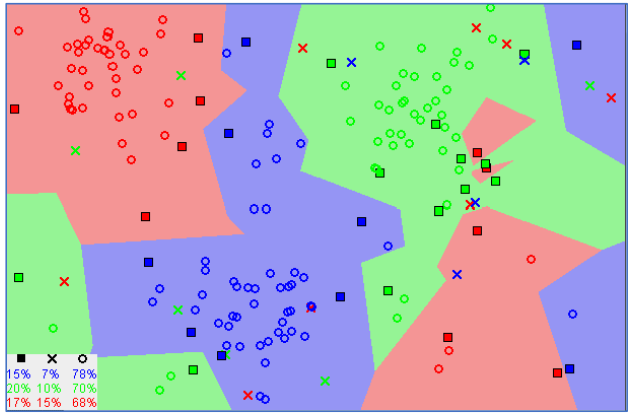


Polychotomizer: What is it?

- Polychotomizer = a multi-class classifier
- Input to the dichotomizer: a feature vector, $\vec{f} = [f_1, \ldots, f_d]$
- Output: a label vector, $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_c]$

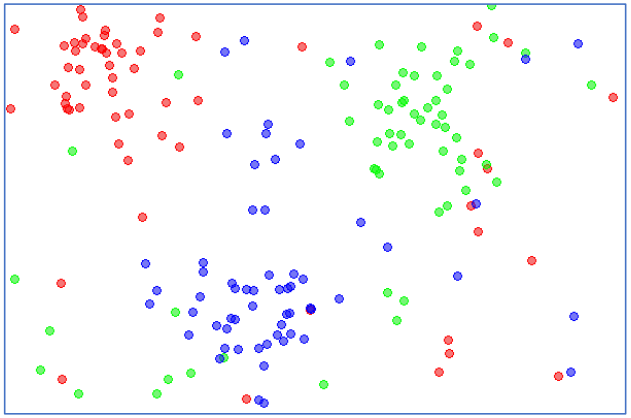
$$0 \leq \hat{y}_j \leq 1, \qquad \sum_{j=1}^{c} \hat{y}_j = 1$$

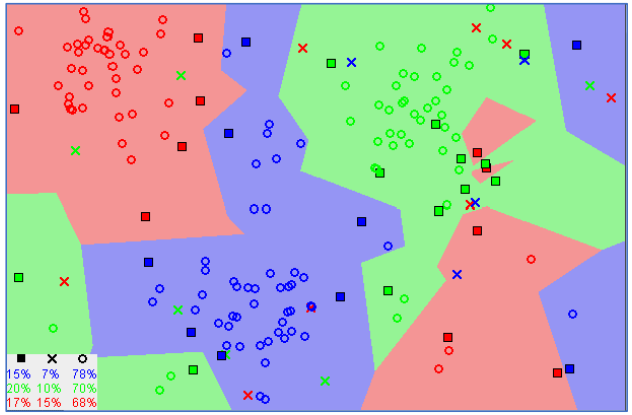




Training a Polychotomizer

- Training database = n training tokens, $\mathfrak{D} = \{\vec{f}_1, \vec{y}_1, \vec{f}_2, \vec{y}_2, \ldots, \vec{f}_n, \vec{y}_n\}$
- n training feature vectors: $\{\vec{f}_1, \vec{f}_2, \ldots, \vec{f}_n\}, \quad \vec{f}_i = [f_{i1}, \ldots, f_{id}]$
- n ground truth labels: $\{\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_n\}, \qquad \vec{y}_i = [y_{i1}, \ldots, y_{ic}]$
- $y_{ij} = 1$ if ith example is from class j
- $y_{ij} = 0$ if ith example is NOT from class j
- Example: if the first example is from class 2 (red), then $\vec{y}_1 = [0,1,0]$





Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function
  - A differentiable approximate argmax
  - How to differentiate the softmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

One-Hot Vector

- Example: if the first example is from class 2 (red), then $\vec{y}_1 = [0,1,0]$

$$y_{ij} = \begin{cases} 1 & \text{i}^{\text{th}} \text{ example is from class j} \\ 0 & \text{i}^{\text{th}} \text{ example is NOT from class j} \end{cases}$$

Call y_{ij} the **reference label**, and call $\hat{y}_{ij}$ the **hypothesis**. Then notice that:

- y_{ij} = True value of $P(class\ j\ |\vec{f}_i)$, because the true probability is always either 1 or 0!
- $\hat{y}_{ij}$ = Estimated value of $P(class\ j\ |\vec{f}_i)$, $0 \leq \hat{y}_j \leq 1$, $\sum_{j=1}^{c} \hat{y}_j = 1$

Wait. Dichotomizer is just a Special Case of Polychotomizer, isn't it?

Yes. Yes, it is.

- Polychotomizer: $\vec{y}_i = [y_{i1}, \ldots, y_{ic}],\ y_{ij} = P(class\ j\ |\vec{f}_i).$
- Dichotomizer: $y_i = P(class\ 1\ |\vec{f}_i)$
- That's all you need, because if there are only two classes, then $P(other\ class\ |\vec{f}_i) = 1 - y_i$
- (One of the two classes in a dichotomizer is always called "class 1." The other might be called "class 2," or "class 0," or "class -1"…. Who cares. They all mean "the class that is not class 1.")

Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function
  - A differentiable approximate argmax
  - How to differentiate the softmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

OK, now we know what the polychotomizer should compute. How do we compute it?

Now you know that

- y_{ij} = reference label = True value of $P(class\ j\ |\vec{f}_i)$, given to you with the training database.
- $\hat{y}_{ij}$ = hypothesis = value of $P(class\ j\ |\vec{f}_i)$ estimated by the neural net.

How can we do that estimation?

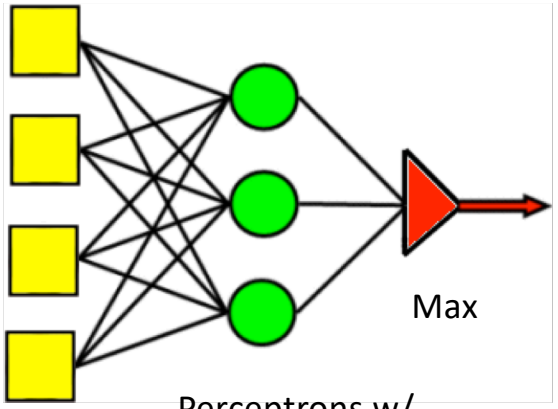
OK, now we know what the polychotomizer should compute. How do we compute it?

$\hat{y}_{ij}$ = value of $P(class\ j\ |\vec{f}_i)$ estimated by the neural net.

How can we do that estimation?

Multi-class perceptron example:

$$\hat{y}_{ij} = \begin{cases} 1 & \text{if } j = \underset{1 \le \ell \le c}{\text{argmax}}\, \vec{w}_\ell \cdot \vec{f}_i \\ 0 & \text{otherwise} \end{cases}$$



Differentiable perceptron: we need a differentiable approximation of the argmax function.

Softmax = differentiable approximation of the argmax function

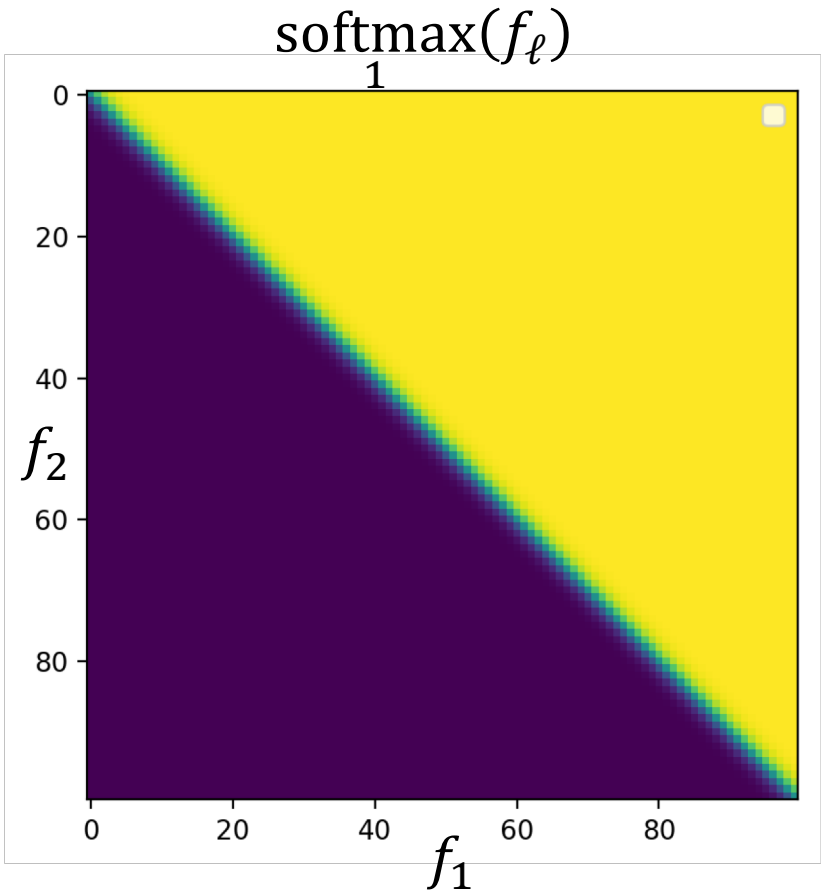
The softmax function is defined as:

$$\hat{y}_{ij} = \underset{j}{\text{softmax}}\left(\vec{w}_\ell \cdot \vec{f}_i\right) = \frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}$$

For example, the figure to the right shows

$$\hat{y}_1 = \underset{1}{\text{softmax}}(f_\ell) = \frac{e^{f_1}}{\sum_{\ell=1}^{2} e^{f_\ell}}$$

Notice that it's close to 1 (yellow) when $f_1 = \max f_\ell$, and close to zero (blue) otherwise, with a smooth transition zone in between.



Softmax = differentiable approximation of the argmax function

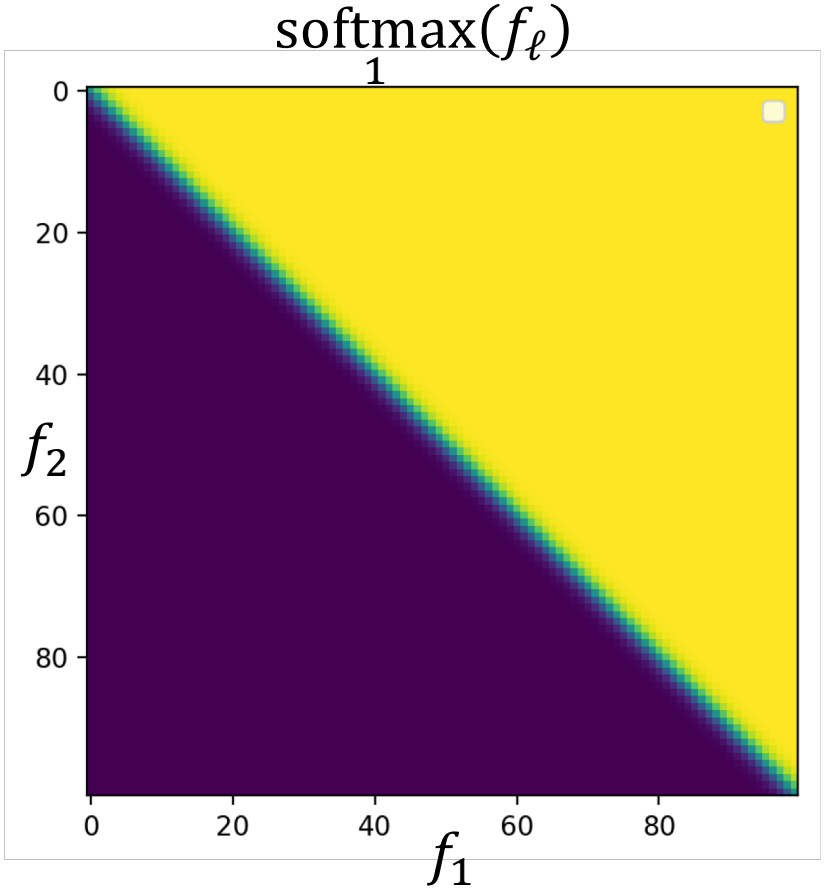
The softmax function is defined as:

$$\hat{y}_{ij} = \underset{j}{\text{softmax}}\left(\vec{w}_\ell \cdot \vec{f}_i\right) = \frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}$$

Notice that this gives us

$$0 \leq \hat{y}_{ij} \leq 1, \qquad \sum_{j=1}^{c} \hat{y}_{ij} = 1$$

Therefore we can interpret $\hat{y}_{ij}$ as an estimate of $P(class\ j\ |\vec{f}_i)$.



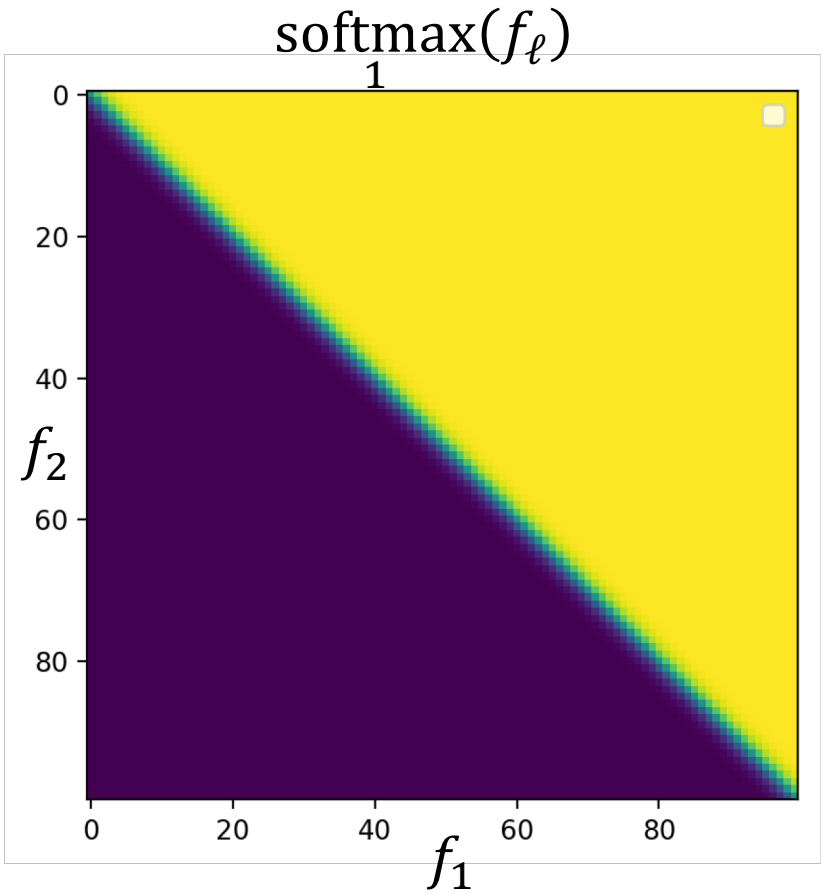
Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function
  - A differentiable approximate argmax
  - How to differentiate the softmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

How to differentiate the softmax: 3 steps

Unlike argmax, the softmax function is differentiable. All we need is the chain rule, plus three rules from calculus:

1. $\frac{\partial}{\partial w}\left(\frac{a}{b}\right) = \left(\frac{1}{b}\right)\frac{\partial a}{\partial w} - \left(\frac{a}{b^2}\right)\frac{\partial b}{\partial w}$
2. $\frac{\partial}{\partial w}(e^a) = (e^a)\frac{\partial a}{\partial w}$
3. $\frac{\partial}{\partial w}(wf) = f$



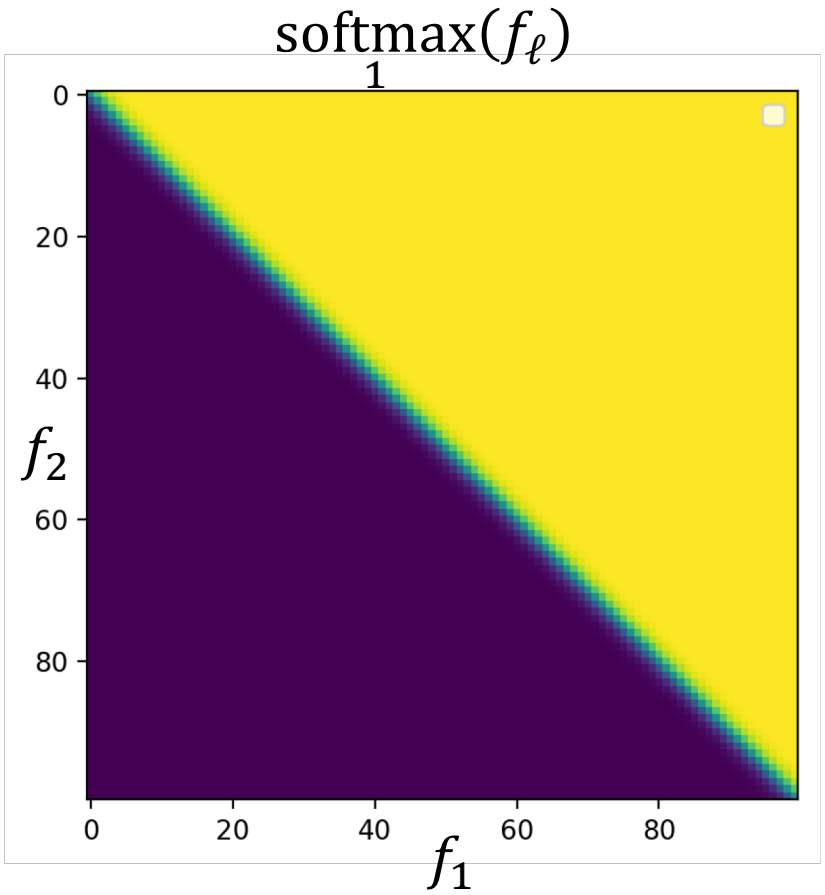
How to differentiate the softmax: step 1

First, we use the rule for $\frac{\partial}{\partial w}\left(\frac{a}{b}\right) = \left(\frac{1}{b}\right)\frac{\partial a}{\partial w} - \left(\frac{a}{b^2}\right)\frac{\partial b}{\partial w}$:

$$\hat{y}_{ij} = \underset{j}{\text{softmax}}\left(\vec{w}_\ell \cdot \vec{f}_i\right) = \frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}$$

$$\frac{\partial \hat{y}_{ij}}{\partial w_{mk}} = \left(\frac{1}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}\right)\left(\frac{\partial e^{\vec{w}_j \cdot \vec{f}_i}}{\partial w_{mk}}\right) - \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)}{\partial w_{mk}}\right)$$

$$= \begin{cases} \left(\frac{1}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}\right)\left(\frac{\partial e^{\vec{w}_j \cdot \vec{f}_i}}{\partial w_{mk}}\right) - \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)}{\partial w_{mk}}\right) & m = j \\ -\left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)}{\partial w_{mk}}\right) & m \neq j \end{cases}$$



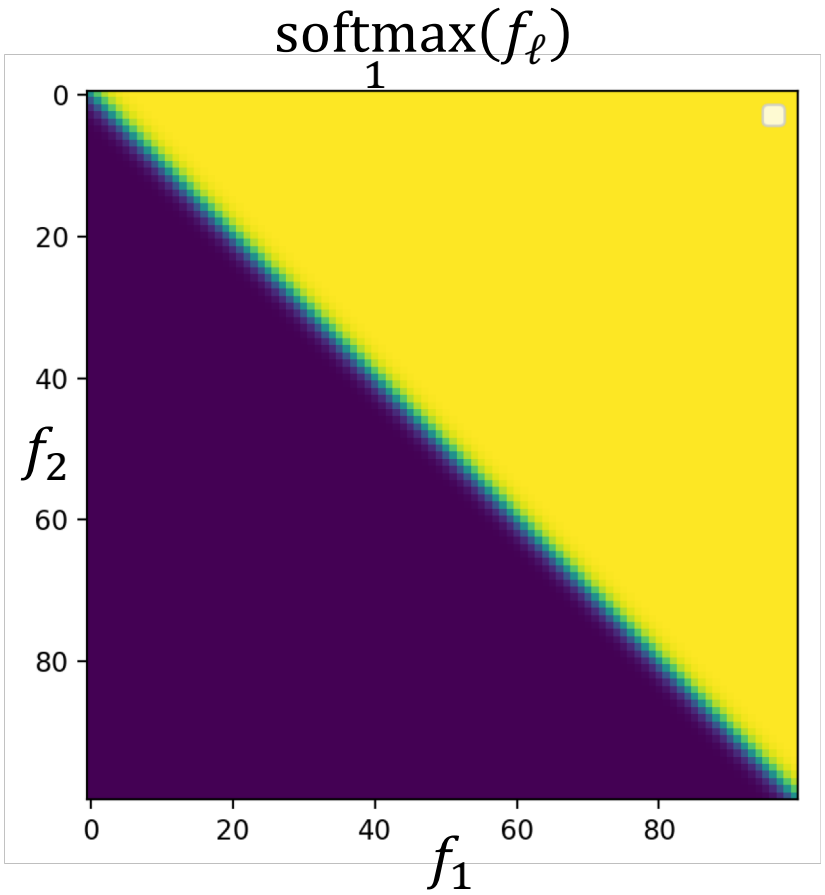
How to differentiate the softmax: step 2

Next, we use the rule $\frac{\partial}{\partial w}(e^a) = (e^a)\frac{\partial a}{\partial w}$:

$$\frac{\partial \hat{y}_{ij}}{\partial w_{mk}} =$$

$$\begin{cases} \left(\frac{1}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}\right)\left(\frac{\partial e^{\vec{w}_j \cdot \vec{f}_i}}{\partial w_{mk}}\right) - \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial \left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)}{\partial w_{mk}}\right) & m = j \\ -\left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial \left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)}{\partial w_{mk}}\right) & m \neq j \end{cases}$$

$$= \begin{cases} \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}} - \frac{\left(e^{\vec{w}_j \cdot \vec{f}_i}\right)^2}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial (\vec{w}_m \cdot \vec{f}_i)}{\partial w_{mk}}\right) & m = j \\ \left(-\frac{e^{\vec{w}_j \cdot \vec{f}_i} e^{\vec{w}_m \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2}\right)\left(\frac{\partial (\vec{w}_m \cdot \vec{f}_i)}{\partial w_{mk}}\right) & m \neq j \end{cases}$$

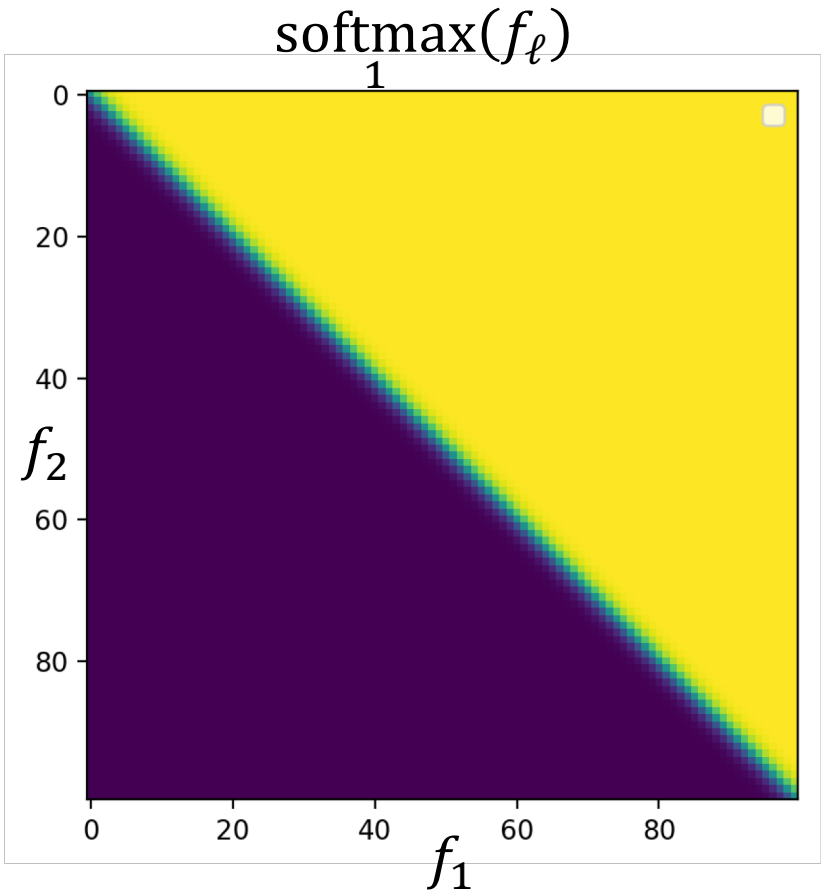


How to differentiate the softmax: step 3

Next, we use the rule $\frac{\partial}{\partial w}(wf) = f$:

$$\frac{\partial \hat{y}_{ij}}{\partial w_{mk}} = \begin{cases} \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}} - \frac{\left(e^{\vec{w}_j \cdot \vec{f}_i}\right)^2}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2} \right) \left(\frac{\partial (\vec{w}_m \cdot \vec{f}_i)}{\partial w_{mk}} \right) & m = j \\ \left(-\frac{e^{\vec{w}_j \cdot \vec{f}_i} e^{\vec{w}_m \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2} \right) \left(\frac{\partial (\vec{w}_m \cdot \vec{f}_i)}{\partial w_{mk}} \right) & m \neq j \end{cases}$$

$$= \begin{cases} \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}} - \frac{\left(e^{\vec{w}_j \cdot \vec{f}_i}\right)^2}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2} \right) f_{ik} & m = j \\ \left(-\frac{e^{\vec{w}_j \cdot \vec{f}_i} e^{\vec{w}_m \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2} \right) f_{ik} & m \neq j \end{cases}$$

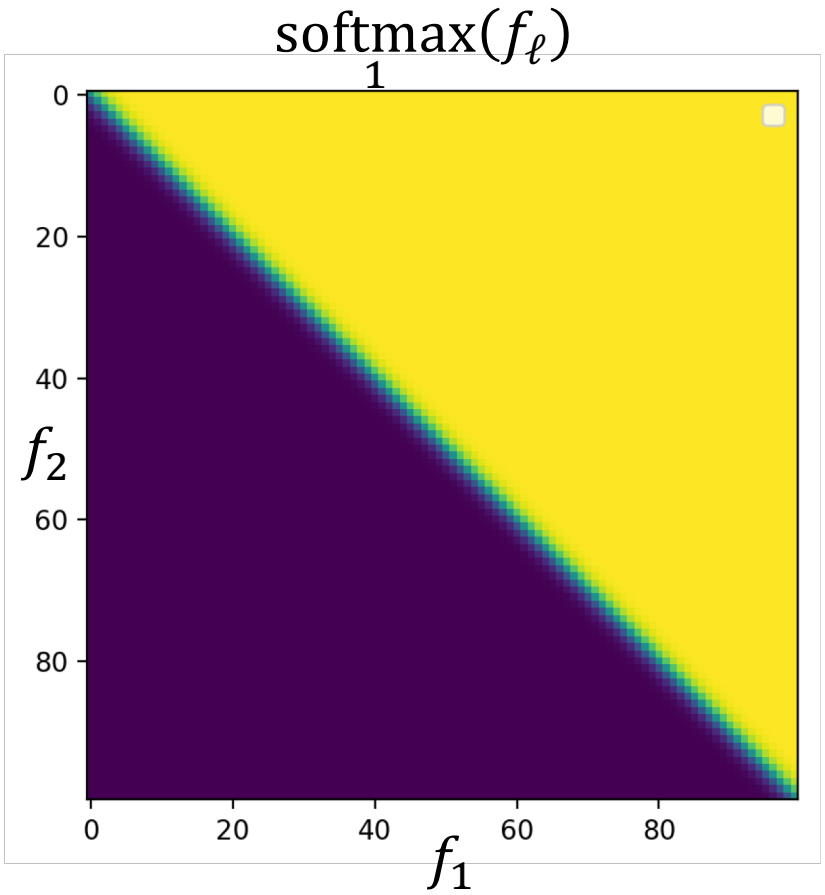


Differentiating the softmax

… and, simplify.

$$\frac{\partial \hat{y}_{ij}}{\partial w_{mk}} = \begin{cases} \left(\frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}} - \frac{\left(e^{\vec{w}_j \cdot \vec{f}_i}\right)^2}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2} \right) f_{ik} & m = j \\ \left(- \frac{e^{\vec{w}_j \cdot \vec{f}_i} e^{\vec{w}_m \cdot \vec{f}_i}}{\left(\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}\right)^2} \right) f_{ik} & m \neq j \end{cases}$$

$$\frac{\partial \hat{y}_{ij}}{\partial w_{mk}} = \begin{cases} \left(\hat{y}_{ij} - \hat{y}_{ij}{}^2\right) f_{ik} & m = j \\ -\hat{y}_{ij} \hat{y}_{im} f_{ik} & m \neq j \end{cases}$$



Recap: how to differentiate the softmax

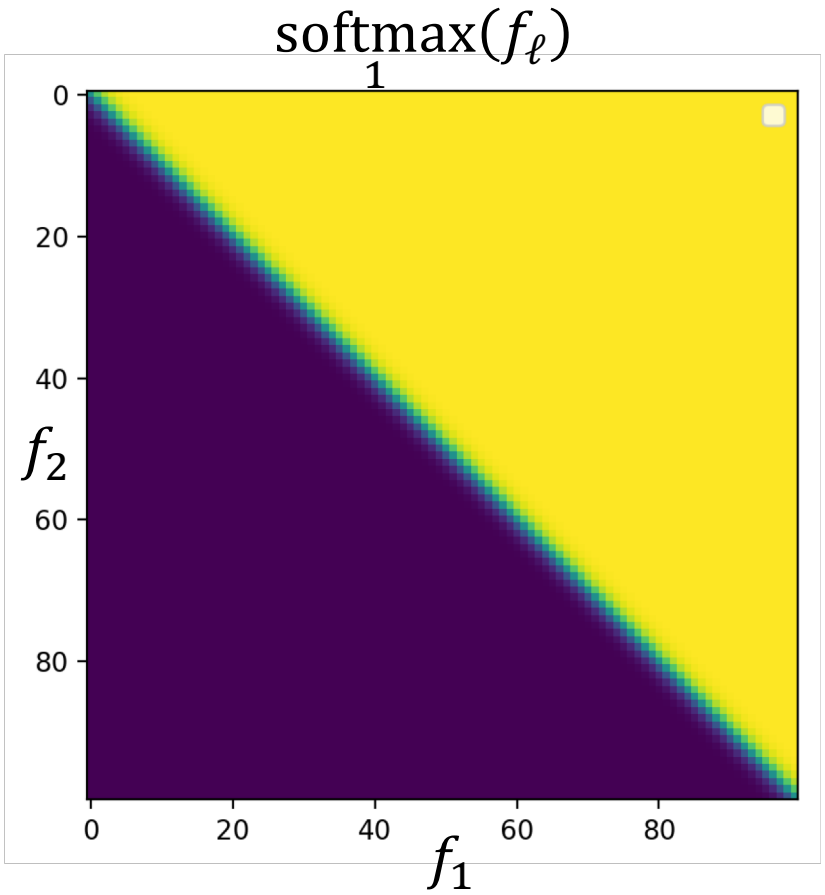
- $\hat{y}_{ij}$ is the probability of the j^{th} class, estimated by the neural net, in response to the i^{th} training token
- w_{mk} is the network weight that connects the k^{th} input feature to the m^{th} class label

The dependence of $\hat{y}_{ij}$ on w_{mk} for $m \neq j$ is weird, and people who are learning this for the first time often forget about it. It comes from the denominator of the softmax.

$$\hat{y}_{ij} = \underset{j}{\text{softmax}}\left(\vec{w}_\ell \cdot \vec{f}_i\right) = \frac{e^{\vec{w}_j \cdot \vec{f}_i}}{\sum_{\ell=1}^{c} e^{\vec{w}_\ell \cdot \vec{f}_i}}$$

$$\frac{\partial \hat{y}_{ij}}{\partial w_{mk}} = \begin{cases} \left(\hat{y}_{ij} - \hat{y}_{ij}{}^2\right) f_{ik} & m = j \\ -\hat{y}_{ij}\hat{y}_{im} f_{ik} & m \neq j \end{cases}$$

- $\hat{y}_{im}$ is the probability of the m^{th} class for the i^{th} training token
- f_{ik} is the value of the k^{th} input feature for the i^{th} training token



Outline

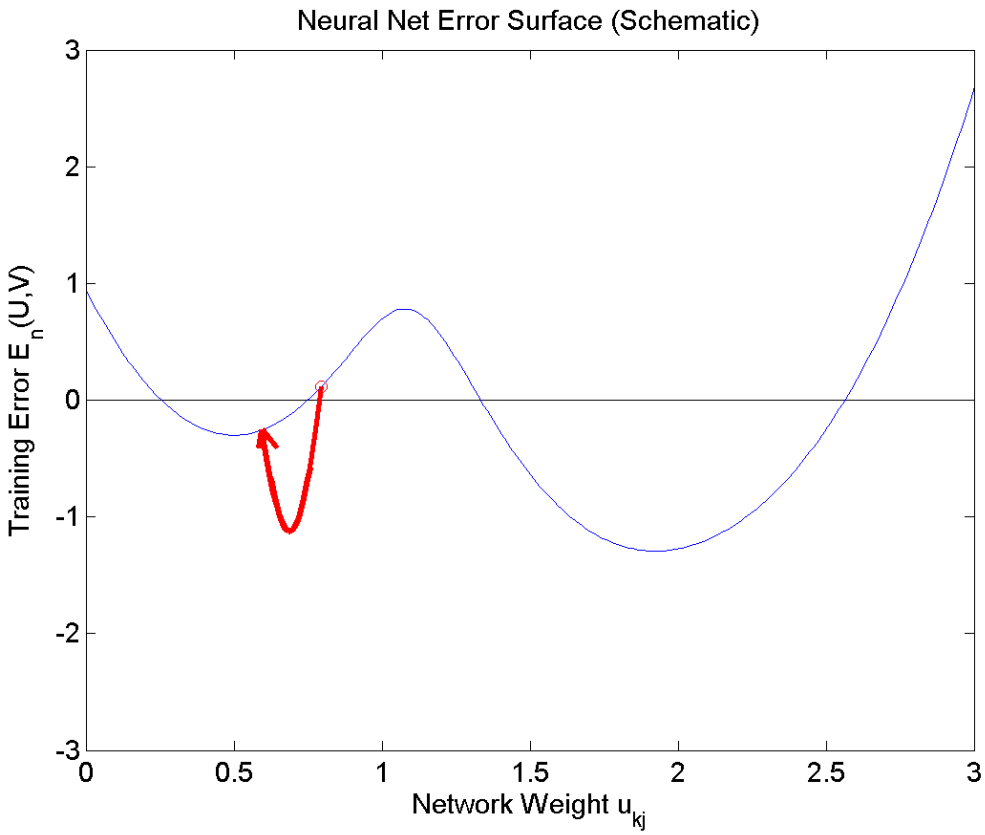
- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function: A differentiable approximate argmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

Training a Softmax Neural Network

All of that differentiation is useful because 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

$$w_{mk} = w_{mk} - \eta \frac{\partial L}{\partial w_{mk}}$$

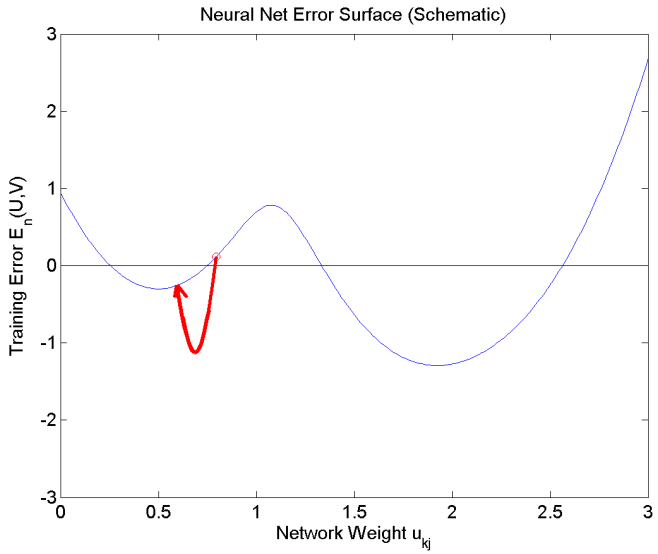
So what is L?



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

$$\hat{y}_{ij} = \text{ Estimated value of } P(\text{class } j \mid \vec{f}_i)$$



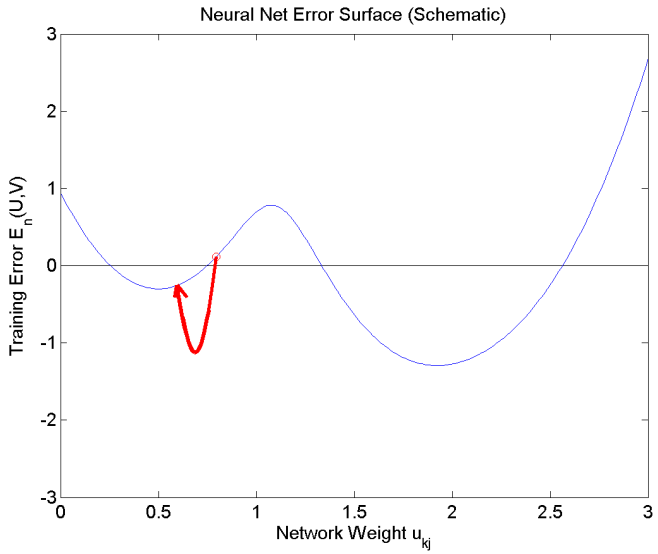
Suppose we decide to estimate the network weights w_{mk} in order to maximize the probability of the training database, in the sense of

$$w_{mk} = \underset{w}{\text{argmax}}\, P(\text{training labels} \mid \text{training feature vectors})$$

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

$$\hat{y}_{ij} = \text{ Estimated value of } P(\text{class } j \mid \vec{f}_i)$$



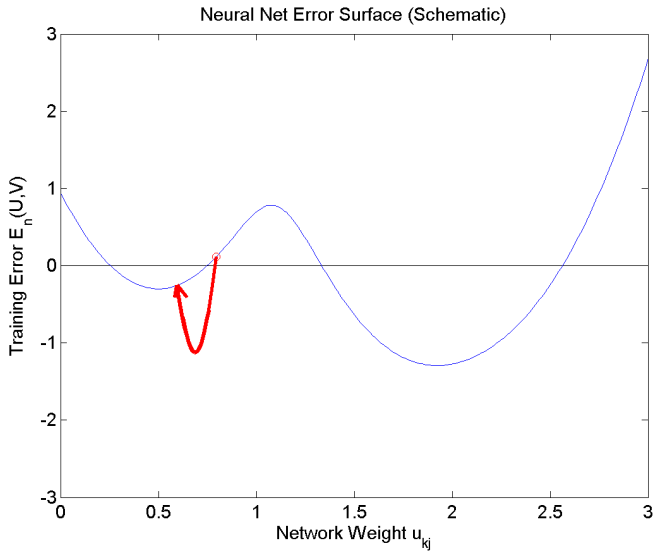
If we assume the training tokens are independent, this is:

$$w_{mk} = \underset{w}{\text{argmax}} \prod_{i=1}^{n} P\left(\text{reference label of the } i^{th} \text{token} \,\middle|\, i^{th} \text{feature vector}\right)$$

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

$$\hat{y}_{ij} = \text{ Estimated value of } P(\text{class } j \mid \vec{f}_i)$$

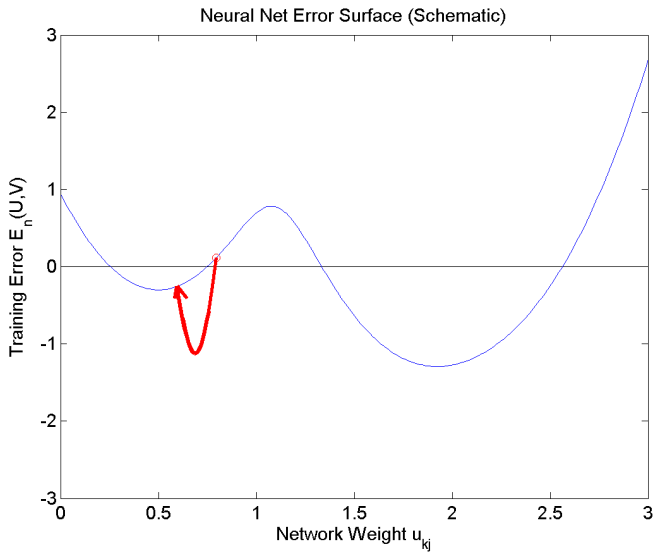


OK. We need to create some notation to mean "the reference label for the i^{th} token." Let's call it $j(i)$.

$$w_{mk} = \underset{w}{\text{argmax}} \prod_{i=1}^{n} P(\text{class } j(i) \mid \vec{f})$$

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 **correct class** $j(i)$, for each token, and then multiplying them all together:



$$w_{mk} = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} \hat{y}_{i,j(i)}$$

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

$$w_{mk} = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \ln \hat{y}_{i,j(i)}$$

Training: Minimizing the negative log probability

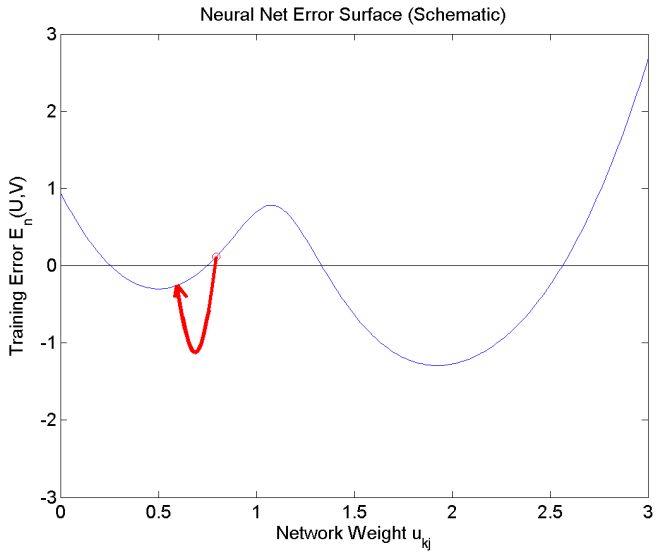
So, to maximize the probability of the training data given the model, we need:

$$w_{mk} = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \ln \hat{y}_{i,j(i)}$$

If we just multiply by (-1), that will turn the max into a min. It's kind of a stupid thing to do---who cares whether you're minimizing L or maximizing $-L$, same thing, right? But it's standard, so what the heck.

$$w_{mk} = \underset{w}{\operatorname{argmin}} L$$

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

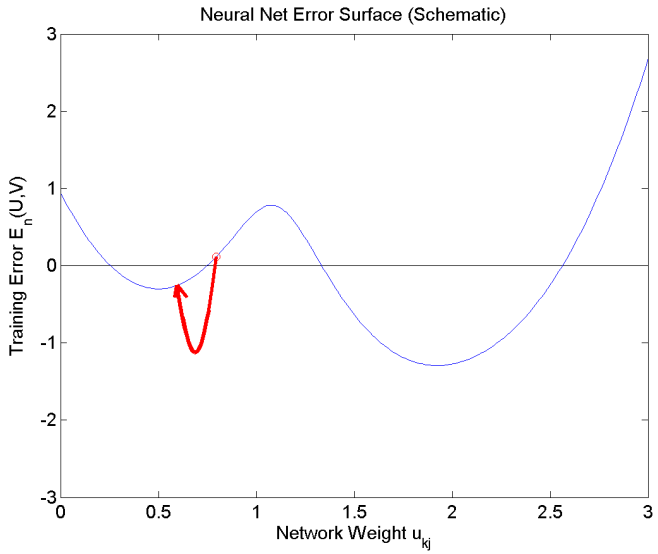


Training: Minimizing the negative log probability

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

$$w_{mk} = \underset{w}{\operatorname{argmin}}\, L$$

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



This loss function, defined above, is called the **cross-entropy loss**. The reasons for that name are very cool, and very far beyond the scope of this course. Take CS 446 (Machine Learning) and/or ECE 563 (Information Theory) to learn more.

Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function: A differentiable approximate argmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

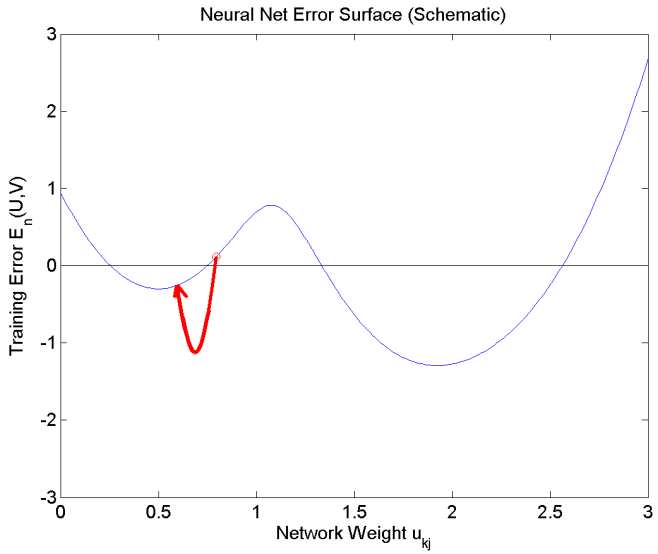
Differentiating the cross-entropy

The cross-entropy loss function is:

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

Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} -\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}}$$



Differentiating the cross-entropy

The cross-entropy loss function is:

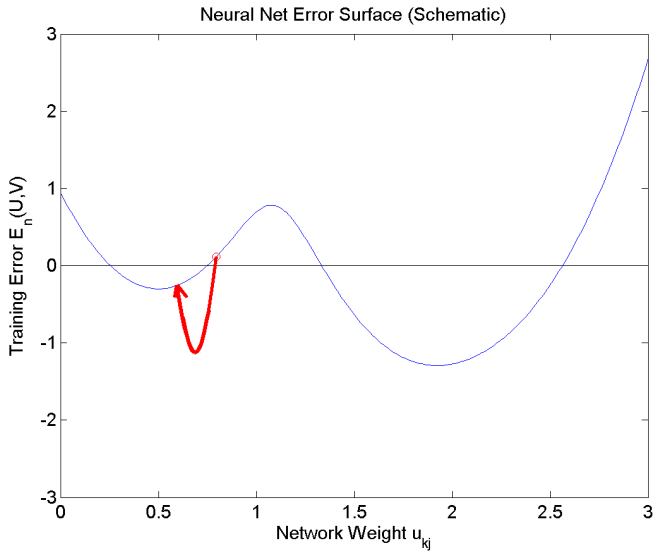
$$L = \sum_{i=1}^{n} -\ln \hat{y}_{i,j(i)}$$

Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} -\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}}$$

…and then…

$$\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}} = \begin{cases} (1-\hat{y}_{im})f_{ik} & m = j(i) \\ -\hat{y}_{im}f_{ik} & m \neq j(i) \end{cases}$$



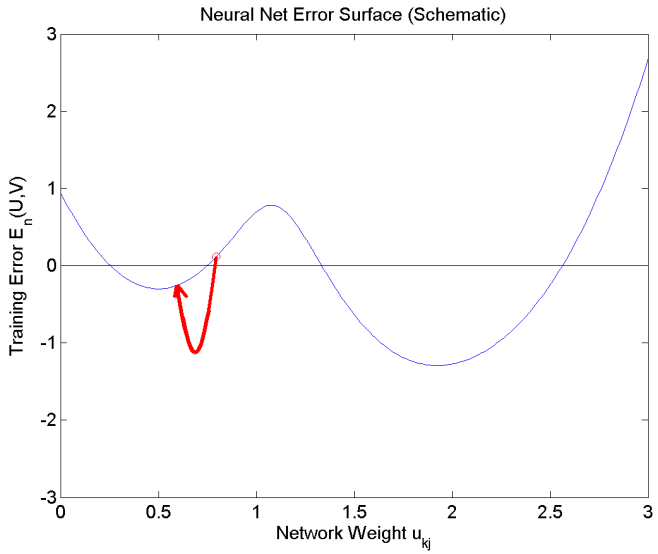
Differentiating the cross-entropy

Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} -\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}}$$

…and then…

$$\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}} = \begin{cases} (1-\hat{y}_{im})f_{ik} & m = j(i) \\ -\hat{y}_{im}f_{ik} & m \neq j(i) \end{cases}$$



… but remember our reference labels:

$$y_{ij} = \begin{cases} 1 & \text{i}^{\text{th}} \text{ example is from class j} \\ 0 & \text{i}^{\text{th}} \text{ example is NOT from class j} \end{cases}$$

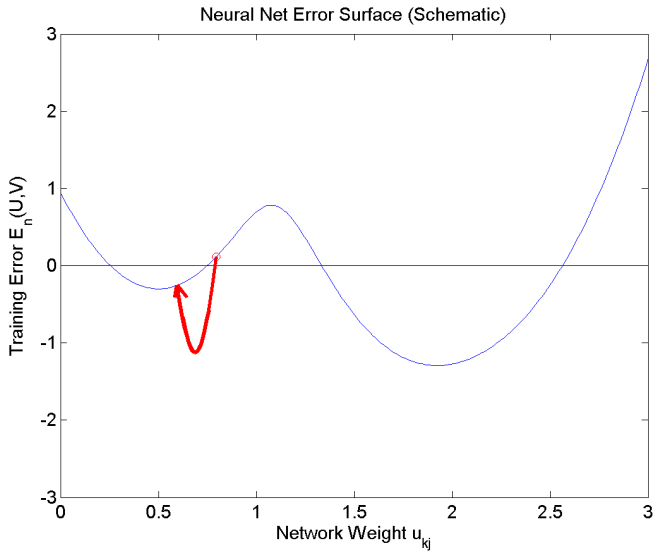
Differentiating the cross-entropy

Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} -\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}}$$

…and then…

$$\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}} = \begin{cases} (y_{im} - \hat{y}_{im}) f_{ik} & m = j(i) \\ (y_{im} - \hat{y}_{im}) f_{ik} & m \neq j(i) \end{cases}$$



… but remember our reference labels:

$$y_{ij} = \begin{cases} 1 & \text{i}^{\text{th}} \text{ example is from class j} \\ 0 & \text{i}^{\text{th}} \text{ example is NOT from class j} \end{cases}$$

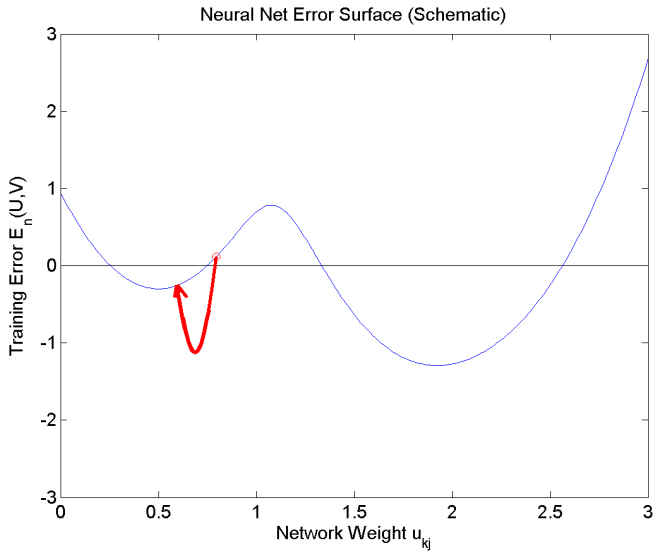
Differentiating the cross-entropy

Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} -\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}}$$

…and then…

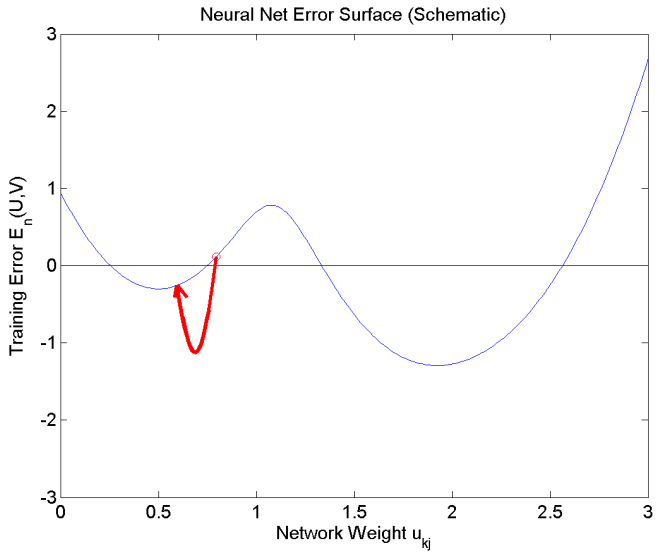
$$\left(\frac{1}{\hat{y}_{i,j(i)}}\right)\frac{\partial \hat{y}_{i,j(i)}}{\partial w_{mk}} = (y_{im} - \hat{y}_{im})f_{ik}$$



Differentiating the cross-entropy

Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} (\hat{y}_{im} - y_{im}) f_{ik}$$



Differentiating the cross-entropy

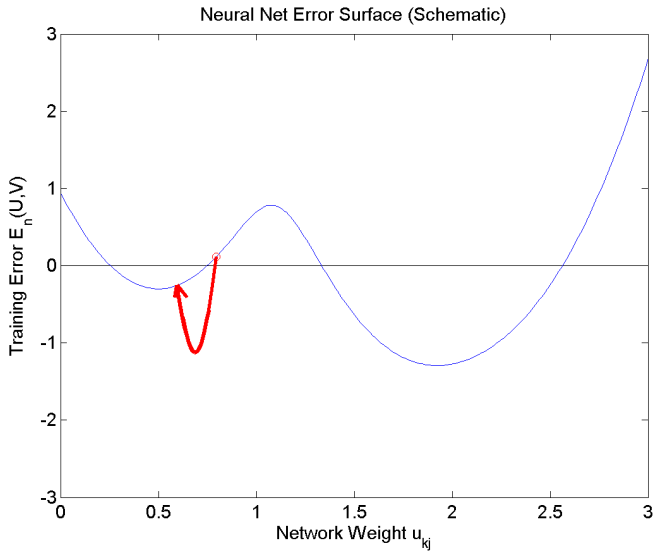
Let's try to differentiate it:

$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} (\hat{y}_{im} - y_{im}) f_{ik}$$

Interpretation:

Increasing w_{mk} will make the error worse if

- $\hat{y}_{im}$ is already too large, and f_{ik} is positive
- $\hat{y}_{im}$ is already too small, and f_{ik} is negative



Differentiating the cross-entropy

Let's try to differentiate it:

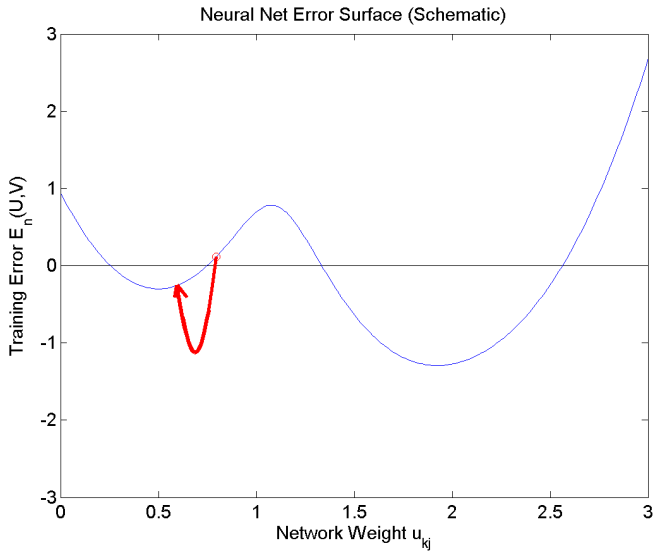
$$\frac{\partial L}{\partial w_{mk}} = \sum_{i=1}^{n} (\hat{y}_{im} - y_{im}) f_{ik}$$

Interpretation:

Our goal is to make the error as small as possible. So if

- $\hat{y}_{im}$ is already too large, then we want to make $w_{mk} f_{ik}$ smaller
- $\hat{y}_{im}$ is already too small , then we want to make $w_{mk} f_{ik}$ larger

$$w_{mk} = w_{mk} - \eta \frac{\partial L}{\partial w_{mk}}$$



Outline

- Dichotomizers and Polychotomizers
  - Dichotomizer: what it is; how to train it
  - Polychotomizer: what it is; how to train it
- One-Hot Vectors: Training targets for the polychotomizer
- Softmax Function: A differentiable approximate argmax
- Cross-Entropy
  - Cross-entropy = negative log probability of training labels
  - Derivative of cross-entropy w.r.t. network weights
- Putting it all together: a one-layer softmax neural net

Summary: Training Algorithms You Know

1. Naïve Bayes with Laplace Smoothing:

$$P(f_k = x|\text{class } j) = \frac{(\#\text{tokens of class } j \text{ with } f_k = x) + 1}{(\#\text{tokens of class } j) + (\#\text{possible values of } f_k)}$$

2. Multi-Class Perceptron: If token $\vec{f_i}$ of class j is misclassified as class m, then

$$\vec{w_j} = \vec{w_j} + \eta\vec{f_i}$$
$$\vec{w_m} = \vec{w_m} - \eta\vec{f_i}$$

3. Softmax Neural Net: for all weight vectors (correct or incorrect),

$$\vec{w_m} = \vec{w_m} - \eta\nabla_{\vec{w_m}}L$$
$$= \vec{w_m} - \eta(\hat{y}_{im} - y_{im})\vec{f_i}$$

Summary: Perceptron versus Softmax

Softmax Neural Net: for all weight vectors (correct or incorrect),

$$\vec{w}_m = \vec{w}_m - \eta(\hat{y}_{im} - y_{im})\vec{f}_i$$

Notice that, if the network were adjusted so that

$$\hat{y}_{im} = \begin{cases} 1 & \text{network thinks the correct class is } m \\ 0 & \text{otherwise} \end{cases}$$

Then we'd have

$$(\hat{y}_{im} - y_{im}) = \begin{cases} -2 & \text{correct class is } m\text{, but network is wrong} \\ 2 & \text{network guesses } m\text{, but it's wrong} \\ 0 & \text{otherwise} \end{cases}$$

Summary: Perceptron versus Softmax

Softmax Neural Net: for all weight vectors (correct or incorrect),

$$\vec{w}_m = \vec{w}_m - \eta(\hat{y}_{im} - y_{im})\vec{f}_i$$

Notice that, if the network were adjusted so that

$$\hat{y}_{im} = \begin{cases} 1 & \text{network thinks the correct class is } m \\ 0 & \text{otherwise} \end{cases}$$

Then we get the perceptron update rule back again (multiplied by 2, which doesn't matter):

$$\vec{w}_m = \begin{cases} \vec{w}_m + 2\eta\vec{f}_i & \text{correct class is } m\text{, but network is wrong} \\ \vec{w}_m - 2\eta\vec{f}_i & \text{network guesses } m\text{, but it's wrong} \\ \vec{w}_m & \text{otherwise} \end{cases}$$

Summary: Perceptron versus Softmax

So the key difference between perceptron and softmax is that, for a perceptron,

$$\hat{y}_{ij} = \begin{cases} 1 & \text{network thinks the correct class is } j \\ 0 & \text{otherwise} \end{cases}$$

Whereas, for a softmax,

$$0 \leq \hat{y}_{ij} \leq 1, \qquad \sum_{j=1}^{c} \hat{y}_{ij} = 1$$

Summary: Perceptron versus Softmax

…or, to put it another way, for a perceptron,

$$\hat{y}_{ij} = \begin{cases} 1 & \text{if } j = \underset{1 \le \ell \le c}{\operatorname{argmax}} \vec{w}_\ell \cdot \vec{f}_i \\ 0 & \text{otherwise} \end{cases}$$

Whereas, for a softmax network,

$$\hat{y}_{ij} = \underset{j}{\operatorname{softmax}}\left(\vec{w}_\ell \cdot \vec{f}_i\right)$$