CS440/ECE448 Lecture 15: Linear Classifiers

Mark Hasegawa-Johnson, 3/2018

Including Slides by

Svetlana Lazebnik, 10/2016

Linear Classifiers

- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

Naïve Bayes/Bag-of-Words

- Model parameters: feature likelihoods P(word | class) and priors P(class)
  - How do we obtain the values of these parameters?
  - Need *training set* of labeled samples from both classes

$$\text{P(word | class)} = \frac{\text{\# of occurrences of this word in docs from this class}}{\text{total \# of words in docs from this class}}$$

  - This is the *maximum likelihood* (ML) estimate, or estimate that maximizes the likelihood of the training data:

$$\prod_{d=1}^{D} \prod_{i=1}^{n_d} P(w_{d,i} \mid class_{d,i})$$

d: index of training document, *i*: index of a word

Indexing in BoW: Types vs. Tokens

- Indexing the training dataset: TOKENS
  - i = document token index, $1 \leq i \leq m$ (there are n document tokens in the training dataset)
  - j = word token index, $1 \leq j \leq n$ (there are n word tokens in each document)
- Indexing the dictionary: TYPES
  - c = class type, $1 \leq c \leq C$ (there are a total of C different class types)
  - w = word type, $1 \leq w \leq V$ (there are a total of V words in the dictionary, i.e., V different word types)

Two Different BoW Algorithms

- **One bit per document**, per word type:
  - F_{iw} = 1 if word "w" occurs anywhere in the i'th document
  - F_{iw} = 0 otherwise
- **One bit per word** token, per word type:
  - F_{jw} = 1 if the j'th word token is "w"
  - F_{jw} = 0 otherwise

Example: "who saw who with who?"

$F_{i,"who"} = 1$

$F_{j,"who"} = \{1,0,1,0,1\}$

Feature = One Bit Per Document

- Features:
  - F_{iw} = 1 if word "w" occurs anywhere in the i'th document
- Parameters:
  - $\lambda_{cw} \equiv P(F_{iw} = 1 | C = c)$
  - Note this means that $P(F_{iw} = 0 | C = c) = 1 - \lambda_{cw}$
- Parameter Learning:

$$\lambda_{cw} = \frac{(1 + \#\text{ documents containing } w)}{(1 + \#\text{ documents containing } w) + (1 + \#\text{ documents NOT containing } w)}$$

Feature = One Bit Per Word Token

- Features:
  - F_{jw} = 1 if the j'th word token is word "w"
- Parameters:
  - $\lambda_{cw} \equiv P(F_{iw} = 1 | C = c) = P(W_j = w | C = c)$
  - Note this means that $P(F_{jw} = 0 | C = c) = \sum_{v \neq w} \lambda_{cv}$
- Parameter Learning:

$$\lambda_{cw} = \frac{(1 + \#\text{ tokens of } w \text{ in the training database})}{\sum_{v=1}^{V}(1 + \#\text{ tokens of } v \text{ in the training database})}$$

Feature = One Bit Per Document

Classification:

C* = argmax P(C=c|document)

= argmax P(C=c) P(Document|C=c)

$$= \arg\max_c \left(\pi_c \prod_{w:\, f_{cw}=1} \lambda_{cw} \prod_{w:\, f_{cw}=0} (1-\lambda_{cw}) \right)$$

Feature = One Bit Per Word Token

Classification:

C* = argmax P(C=c|document)

= argmax P(C=c) P(Document|C=c)

$$= \arg\max_{c} \left(\pi_c \prod_{j=1}^{n} \lambda_{cw_j} \right)$$

Feature = One Bit Per Document

Classification:

$$C^* = \arg\max_c \left(\pi_c \prod_{w=1}^{V} \left(\frac{\lambda_{cw}}{1-\lambda_{cw}} \right)^{f_{cw}} (1-\lambda_{cw}) \right)$$

$$C^* = \arg\max_c \left(\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} \right)$$

$$\alpha_{cw} = \log\left(\frac{\lambda_{cw}}{1-\lambda_{cw}} \right), \qquad \beta_c = \log\left(\pi_c \prod_{w=1}^{V} (1-\lambda_{cw}) \right)$$

Feature = One Bit Per Word Token

Classification:

$$C^* = \arg\max_c \left(\pi_c \prod_{w=1}^{V} {\lambda_{cw}}^{s_w} \right)$$

Where s_w = number of times w occurred in the document!! So…

$$C^* = \arg\max_c \left(\beta_c + \sum_{w=1}^{V} \alpha_{cw} s_{cw} \right)$$

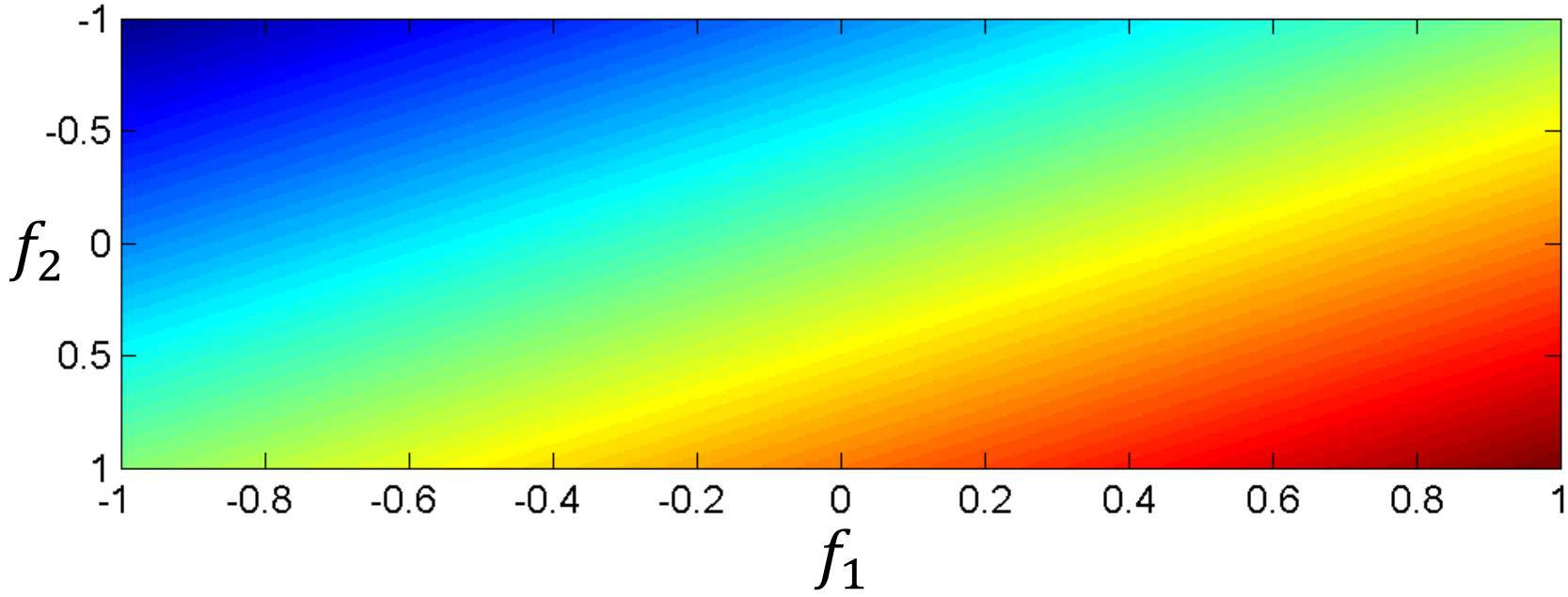
$$\alpha_{cw} = \log \lambda_{cw}\,, \qquad \beta_c = \log \pi_c$$

Linear Classifiers

- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

Linear Classifiers in General

The function $\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw}$ is an affine function of the features f_{cw}. That means that its contours are all straight lines. Here is an example of such a function, plotted as variations of color in a two-dimensional space f_1 by f_2:



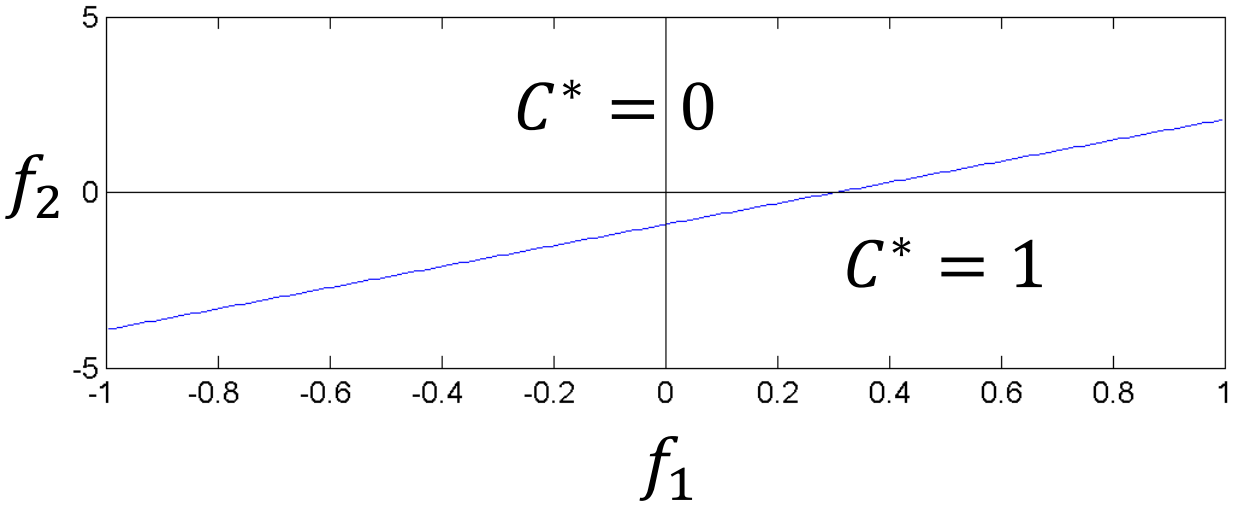
Linear Classifiers in General

Consider the classifier

$$C^* = 1 \quad \text{if} \quad \beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} > 0$$

$$C^* = 0 \quad \text{if} \quad \beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} < 0$$

This is called a "linear classifier" because the boundary between the two classes is a line. Here is an example of such a classifier, with its boundary plotted as a line in the two-dimensional space f_1 by f_2:

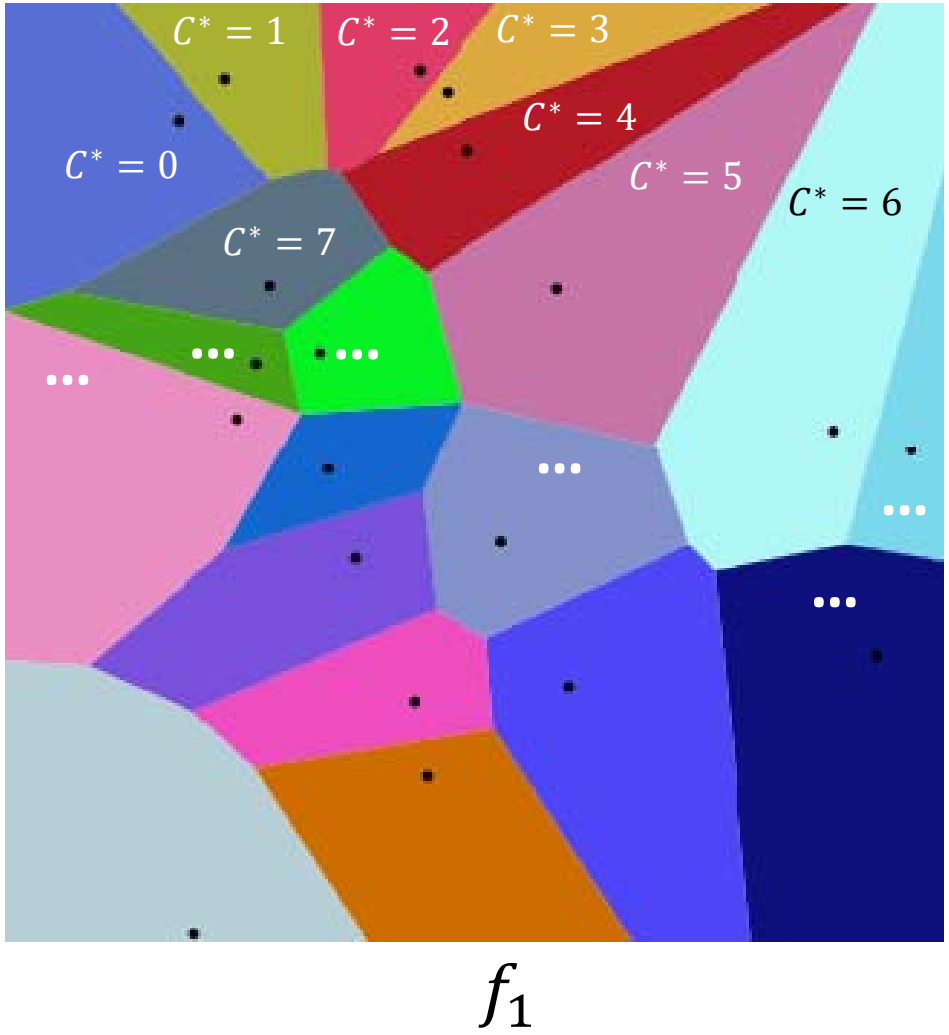


Linear Classifiers in General

Consider the classifier

$$C^* = \arg\max_c \left(\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} \right)$$

- This is called a "multi-class linear classifier."
- The regions $C^* = 0$, $C^* = 1$, $C^* = 2$ etc. are called "Voronoi regions."
- They are regions with piece-wise linear boundaries. Here is an example from Wikipedia of Voronoi regions plotted in the two-dimensional space f_1 by f_2:



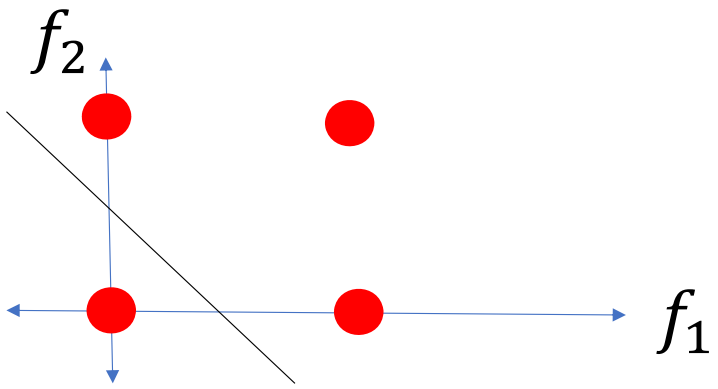
Linear Classifiers in General

When the features are binary ($f_w \in \{0,1\}$), many (but not all!) binary functions can be re-written as linear functions. For example, the function

$$C^* = (f_1 \vee f_2)$$

can be re-written as

C*=1 iff $f_1 + f_2 - 0.5 > 0$

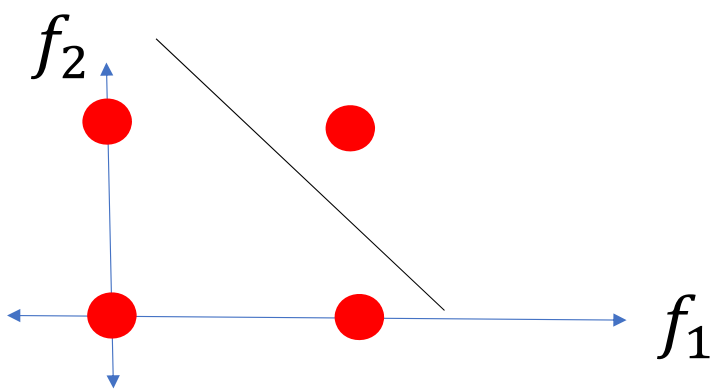


Similarly, the function

$$C^* = (f_1 \wedge f_2)$$

can be re-written as

C*=1 iff $f_1 + f_2 - 1.5 > 0$



Linear Classifiers in General

- Not all logical functions can be written as linear classifiers!
- Minsky and Papert wrote a book called *Perceptrons* in 1969. Although the book said many other things, the only thing most people remembered about the book was that:
- **"A linear classifier cannot learn an XOR function."**
- Because of that statement, most people gave up working on neural networks from about 1969 to about 2006.
- Minsky and Papert also proved that a two-layer neural net can learn an XOR function. But most people didn't notice.

Linear Classifiers

Classification:

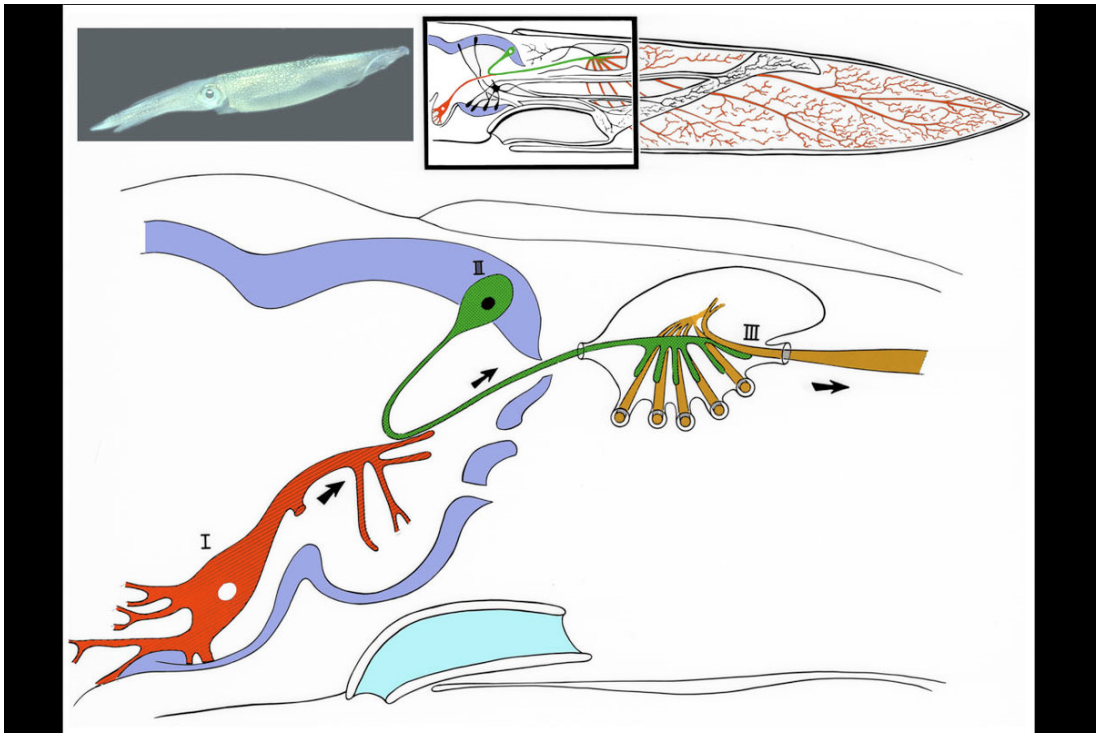
$$C^* = \arg\max_c \left(\beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} \right)$$

- Where f_{cw} are the features (binary, integer, or real), α_{cw} are the feature weights, and β_c is the offset

Linear Classifiers

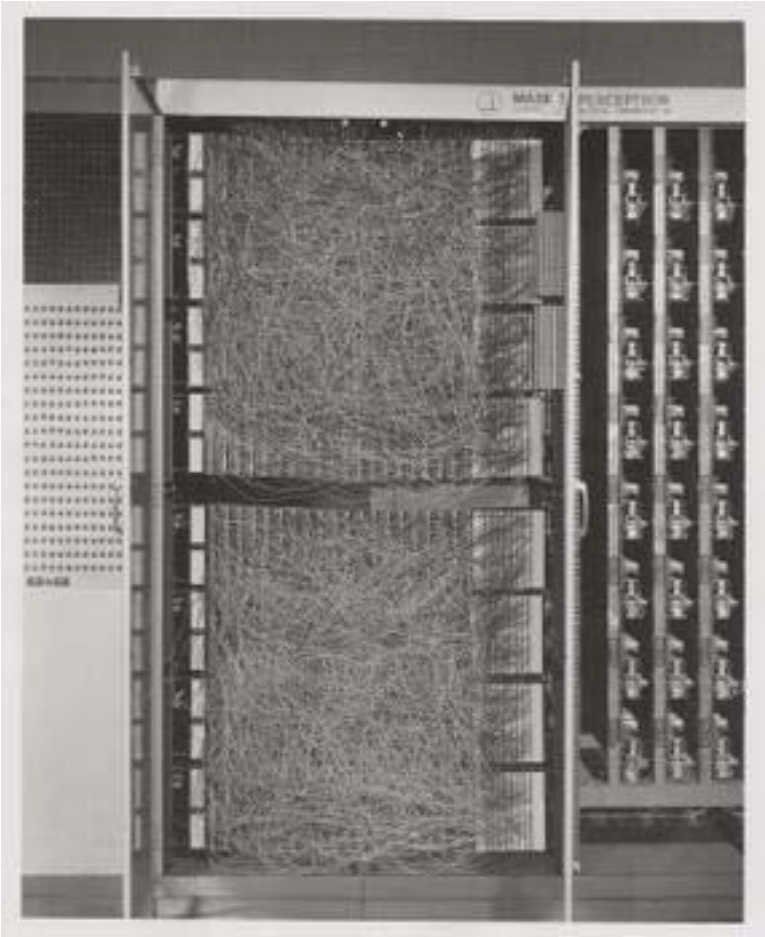
- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

The Giant Squid Axon



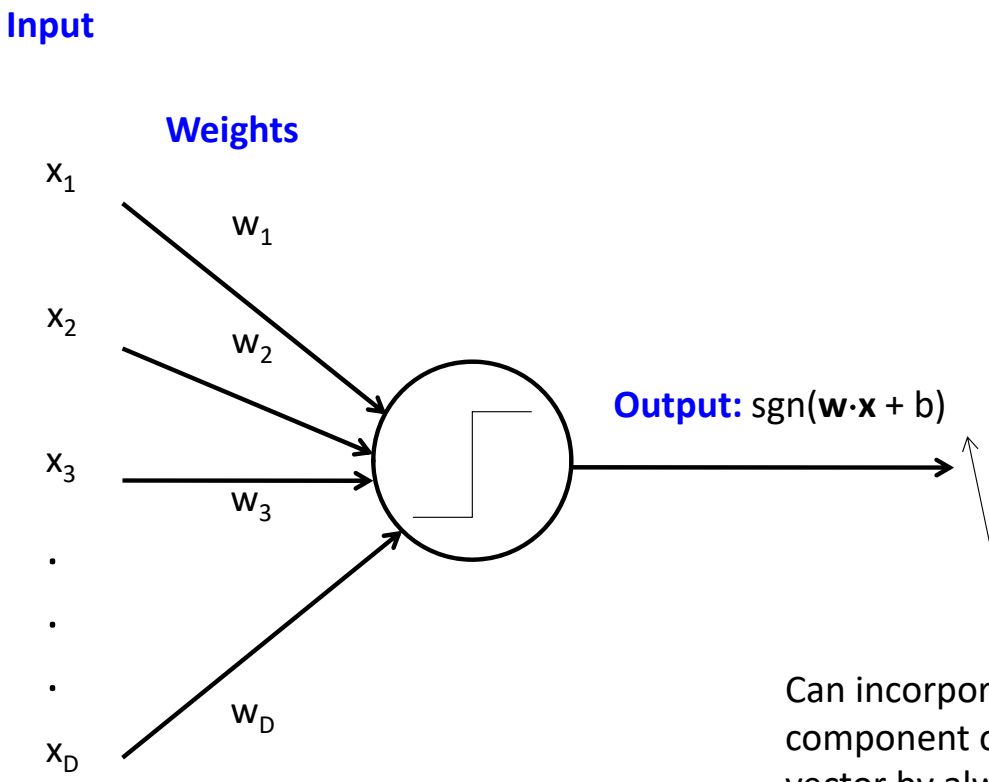
- 1909: Williams discovers that the giant squid has a giant neuron (axon 1mm thick)
- 1939: Young finds a giant synapse (fig. shown: Llinás, 1999, via Wikipedia). Hodgkin & Huxley put in voltage clamps.
- 1952: Hodgkin & Huxley publish an electrical current model for the generation of binary action potentials from real-valued inputs.

Perceptron



- 1959: Rosenblatt is granted a patent for the “perceptron,” an electrical circuit model of a neuron.

Perceptron



Perceptron model: action potential = signum(affine function of the features)

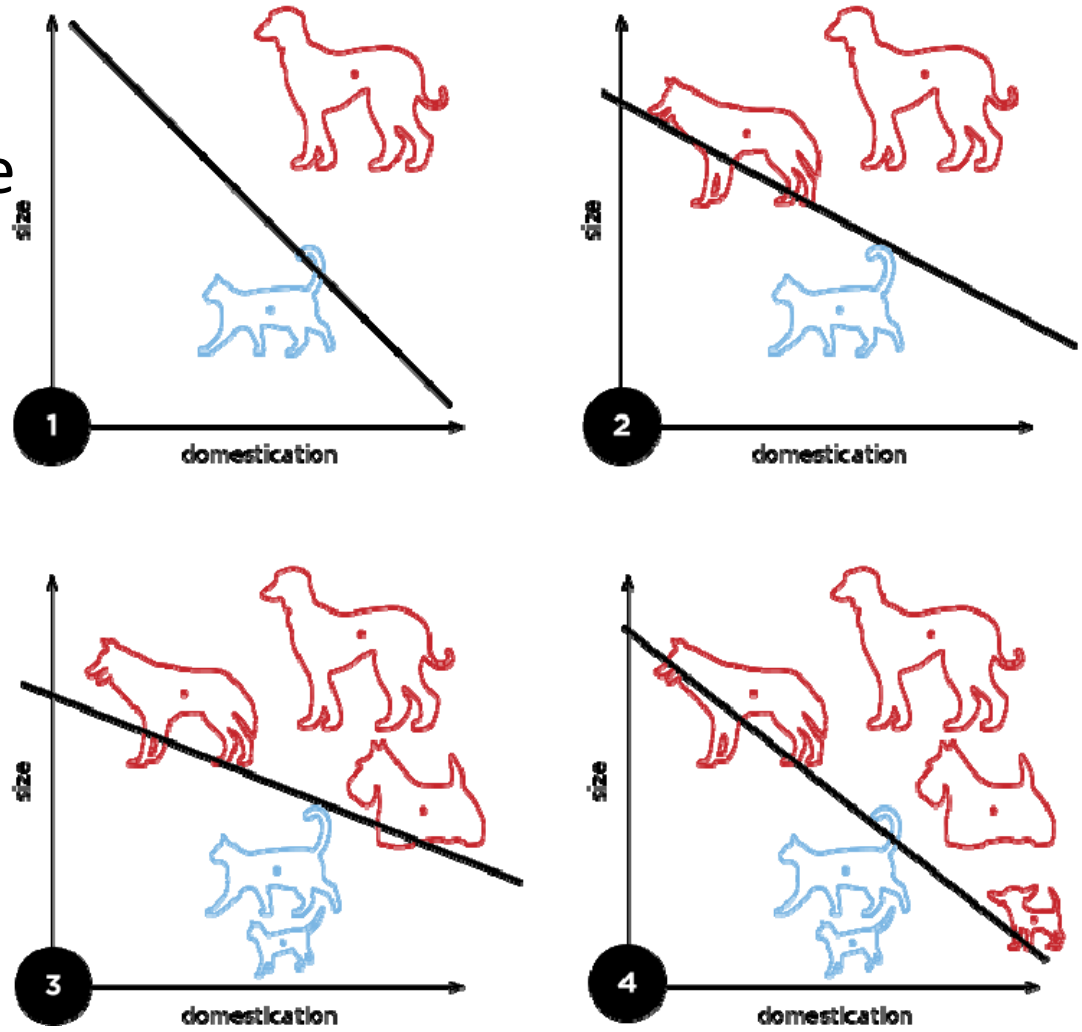
$C^* = \text{sgn}(\alpha_1 f_1 + \alpha_2 f_2 + \ldots + \alpha_V f_V + \beta)$
$= \text{sgn}(\vec{w}^T \vec{x})$

Where $\vec{w} = [\alpha_1, \ldots, \alpha_V, \beta]^T$
and $\vec{x} = [f_1, \ldots, f_V, 1]^T$

Perceptron

Rosenblatt's big innovation: the perceptron learns from examples.

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example:
  - If classified correctly, do nothing
  - If classified incorrectly, update weights



Perceptron

For each training instance $\vec{x}$ with label $y \in \{-1,1\}$:

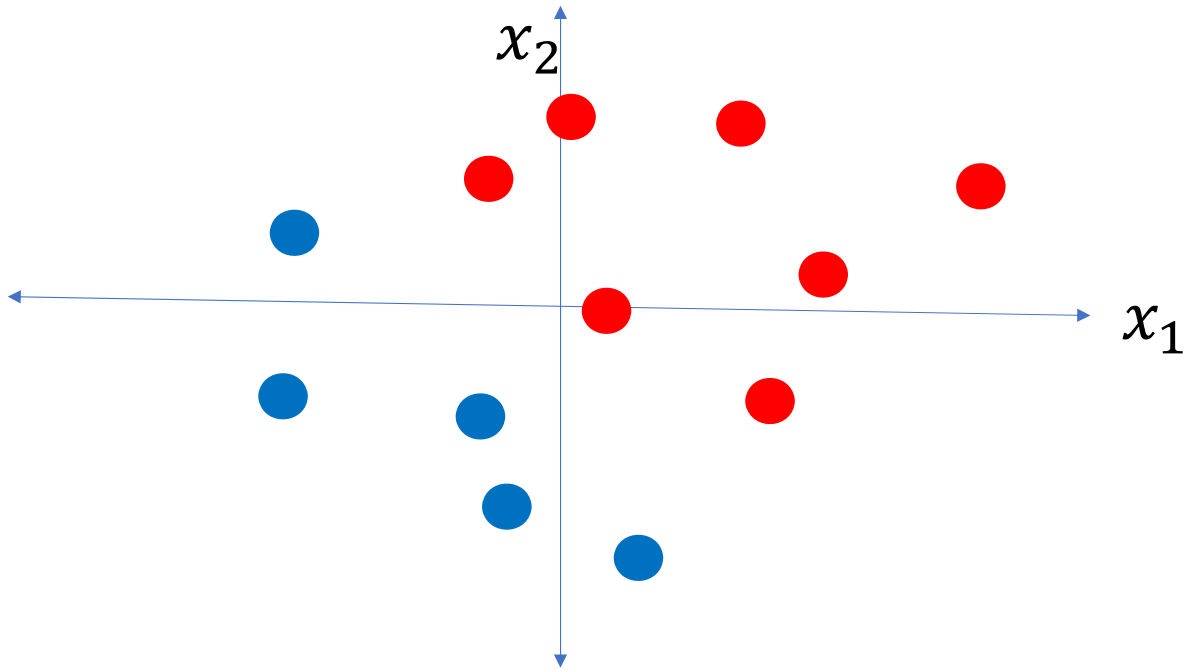
- Classify with current weights: $y' = \text{sgn}(\vec{w}^T \vec{x})$
  - Notice $y' \in \{-1,1\}$ too.
- Update weights:
  - if $y = y'$ then do nothing
  - if $y \neq y'$ then $\vec{w} = \vec{w} + \eta\, y\, \vec{x}$
  - η (eta) is a "learning rate." More about that later.

Perceptron: Proof of Convergence

- If the data are linearly separable (if there exists a $\vec{w}$ vector such that the true label is given by $y' = \mathrm{sgn}(\vec{w}^T \vec{x})$), then the perceptron algorithm is guarantee to converge, even with a constant learning rate, even $\eta=1$.
- In fact, training a perceptron is often the fastest way to find out if the data are linearly separable. If $\vec{w}$ converges, then the data are separable; if $\vec{w}$ diverges toward infinity, then no.
- If the data are not linearly separable, then perceptron converges iff the learning rate decreases, e.g., $\eta=1/n$ for the n'th training sample.

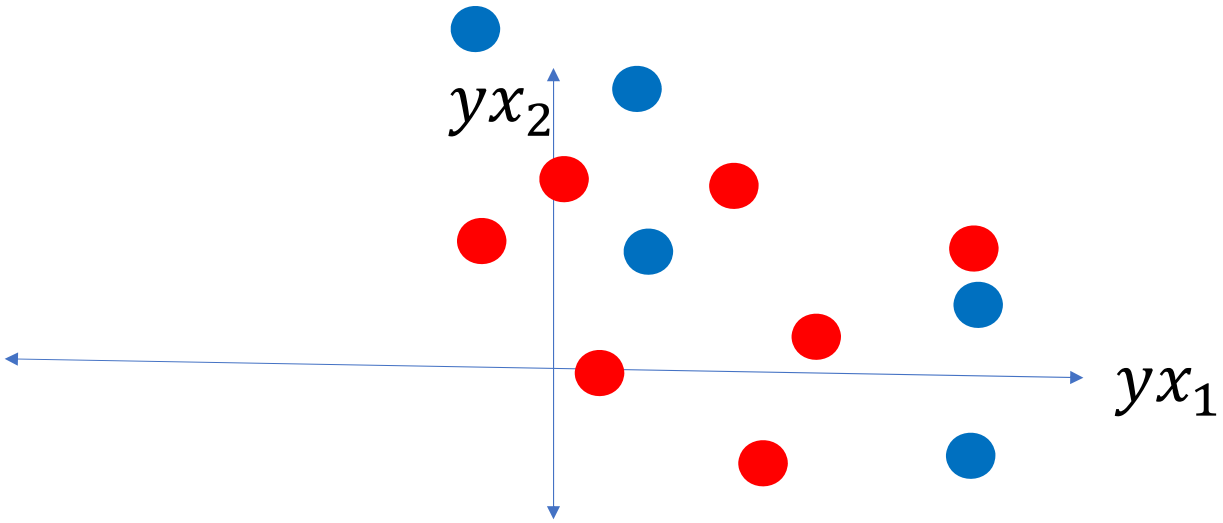
Perceptron: Proof of Convergence

Suppose the data are linearly separable. For example, suppose red dots are the class y=1, and blue dots are the class y=-1:



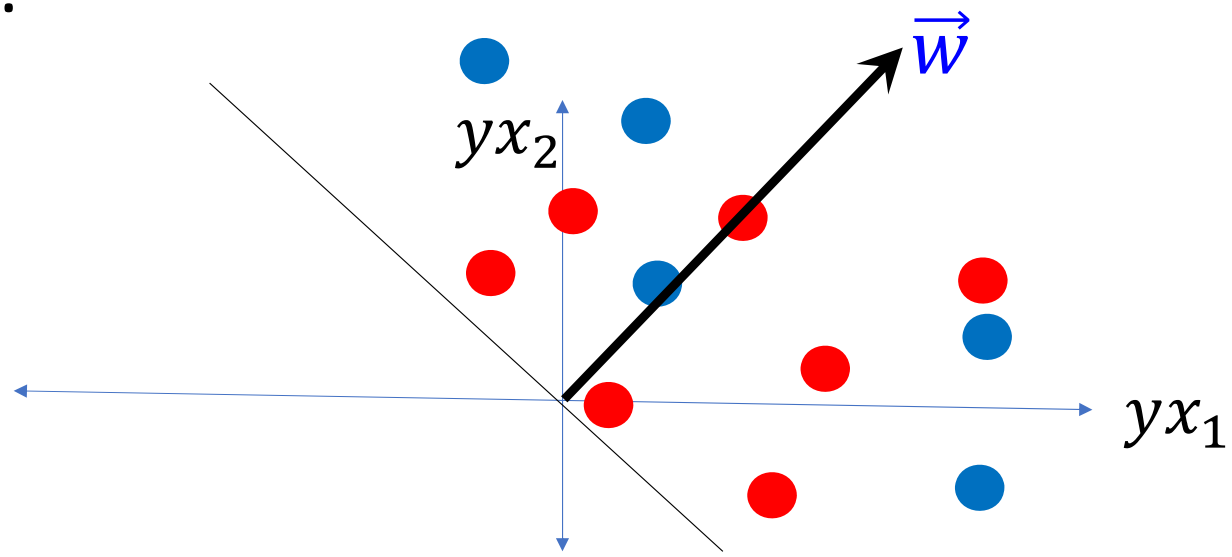
Perceptron: Proof of Convergence

- Instead of plotting $\vec{x}$, plot $y \times \vec{x}$. The red dots are unchanged; the blue dots are multiplied by -1.
- Since the original data were linearly separable, the new data are all in the same half of the feature space.



Perceptron: Proof of Convergence

- Remember the perceptron training rule: if any example is misclassified, then we use it to update $\vec{w} = \vec{w} + y\,\vec{x}$.
- So eventually, $\vec{w}$ becomes just a weighted average of $y\,\vec{x}$.
- … and the perpendicular line, $\vec{w}^T\vec{x} = 0$, is the classifier boundary.



Perceptron: Proof of Convergence

- If the data are not linearly separable, then perceptron converges iff the learning rate decreases, e.g., $\eta=1/n$ for the n'th training sample.

Implementation details

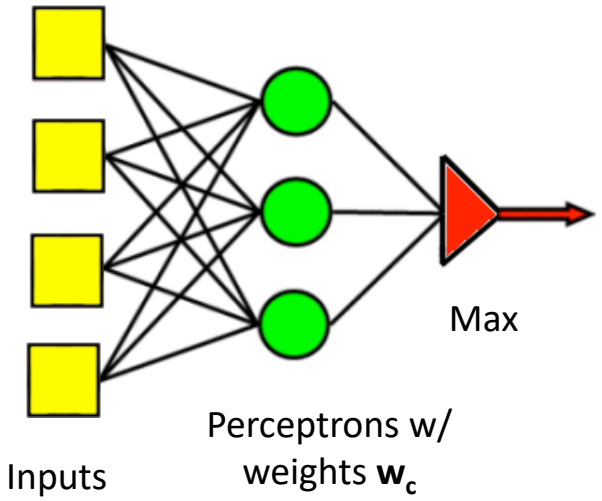
- Bias (add feature dimension with value fixed to 1) vs. no bias
- Initialization of weights: all zeros vs. random
- Learning rate decay function
- Number of epochs (passes through the training data)
- Order of cycling through training examples (random)

Multi-class perceptrons

- *One-vs-others* framework: Need to keep a weight vector $\mathbf{w}_c$ for each class c
- Decision rule: $c = \text{argmax}_c \, \mathbf{w}_c \cdot \mathbf{x}$
- Update rule: suppose example from class c gets misclassified as c'
  - Update for c: $\mathbf{w}_c \leftarrow \mathbf{w}_c + \eta \mathbf{x}$
  - Update for c': $\mathbf{w}_{c'} \leftarrow \mathbf{w}_{c'} - \eta \mathbf{x}$
  - Update for all classes other than c and c': no change

Review: Multi-class perceptrons

- *One-vs-others* framework: Need to keep a weight vector $\mathbf{w}_c$ for each class c
- Decision rule: $c = \text{argmax}_c \, \mathbf{w}_c \cdot \mathbf{x}$



Linear Classifiers

- Naïve Bayes/BoW classifiers
- Linear Classifiers in General
- Perceptron
- Differential Perceptron/Neural Net

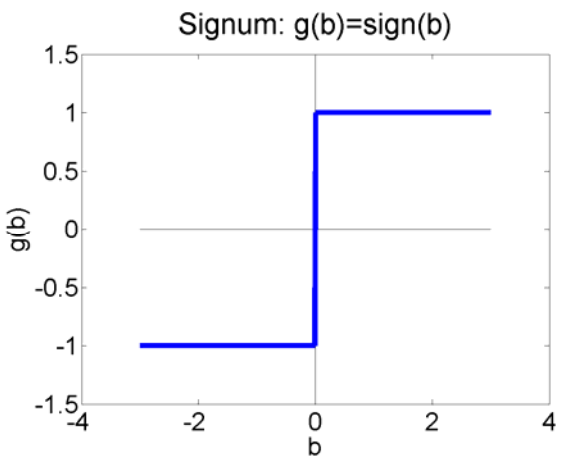
Differential Perceptron

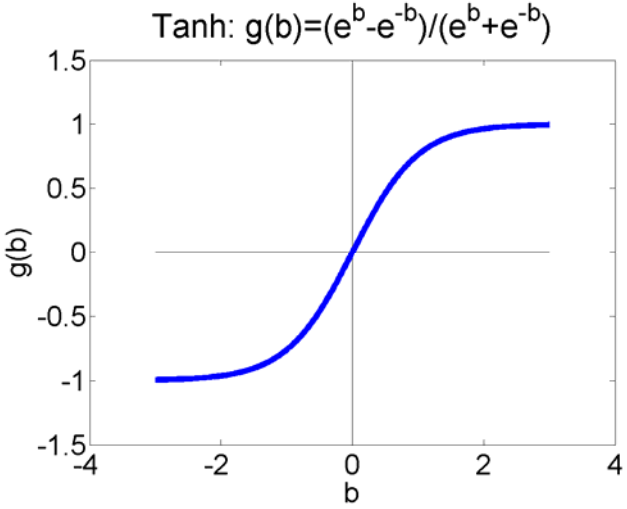
- Also known as a “one-layer feedforward neural network,” also known as “logistic regression.” Has been re-invented many times by many different people.
- Basic idea: replace the non-differentiable decision function

$$y' = \text{sgn}(\vec{w}^T \vec{x})$$

with a differentiable decision function

$$y' = \tanh(\vec{w}^T \vec{x})$$





Differential Perceptron

Suppose we have n training vectors, $\vec{x}_1$ through $\vec{x}_n$. Each one has an associated label $y_i \in \{-1,1\}$. Then we replace the true error,

$$E = \frac{1}{4}\sum_{i=1}^{n}(y_i - \mathrm{sgn}(\vec{w}^T\vec{x}_i))^2$$

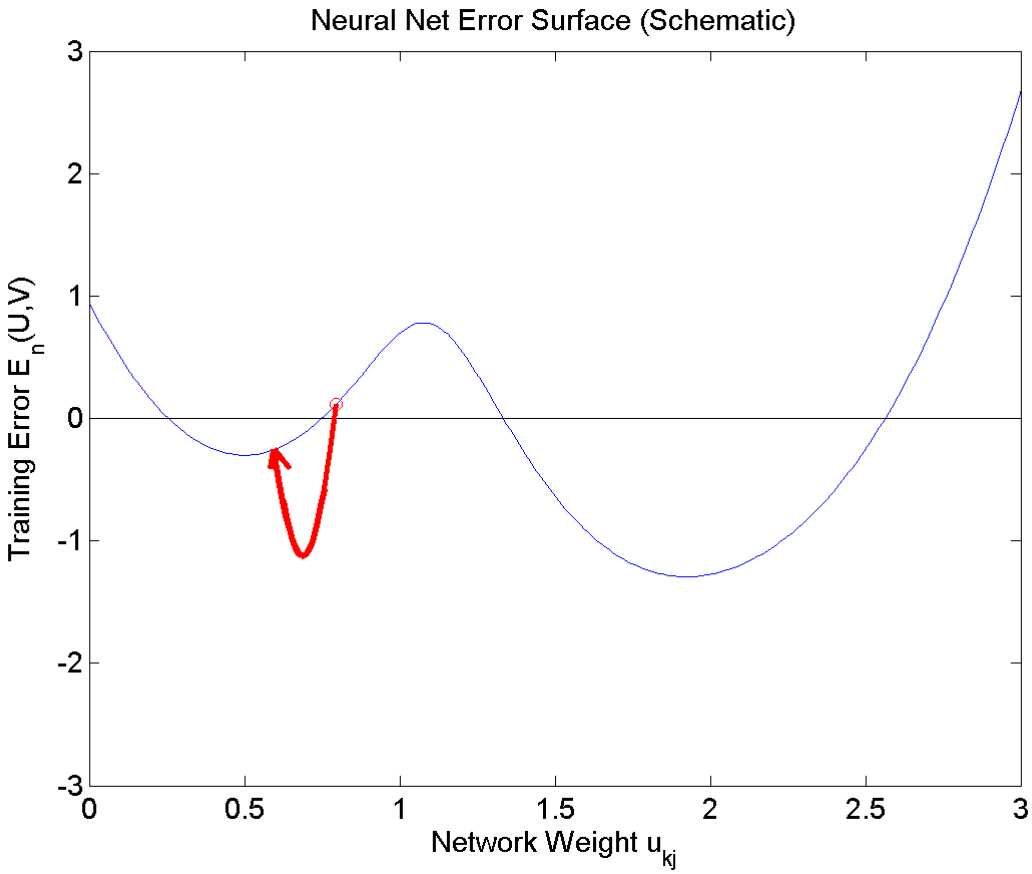
with a differentiable error

$$E = \frac{1}{4}\sum_{i=1}^{n}(y_i - \tanh(\vec{w}^T\vec{x}_i))^2$$

Differential Perceptron

And then the weights get updated according to

$$w_k = w_k - \eta \frac{\partial E}{\partial w_k}$$

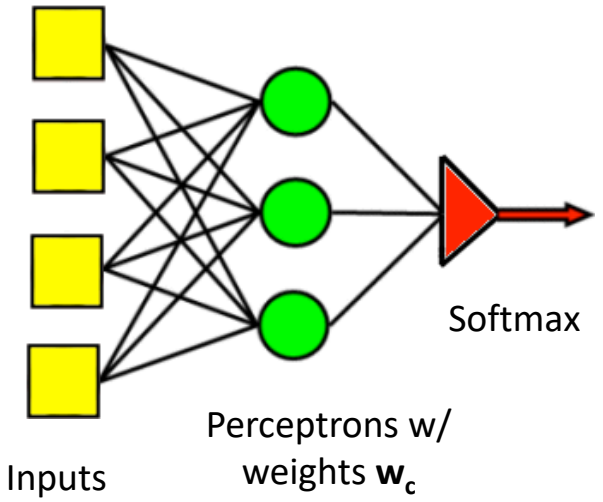


Differentiable Multi-class perceptrons

Same idea works for multi-class perceptrons. We replace the non-differentiable decision rule $c = \text{argmax}_c \mathbf{w}_c \cdot \mathbf{x}$ with the differentiable decision rule $c = \text{softmax}_c \mathbf{w}_c \cdot \mathbf{x}$, where the softmax function is defined as

Softmax:

$$P(c \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x})}{\sum_{k=1}^{C} \exp(\mathbf{w}_k \cdot \mathbf{x})}$$



Summary

You now know SEVEN!! different types of linear classifiers:

- One bit per document Naïve Bayes
- One bit per word token Naïve Bayes
- Linear classifier can implement some logical functions, like AND and OR, but not others, like XOR
- Perceptron
- Multi-class Perceptron
- Differentiable Perceptron
- Differentiable Multi-class perceptron