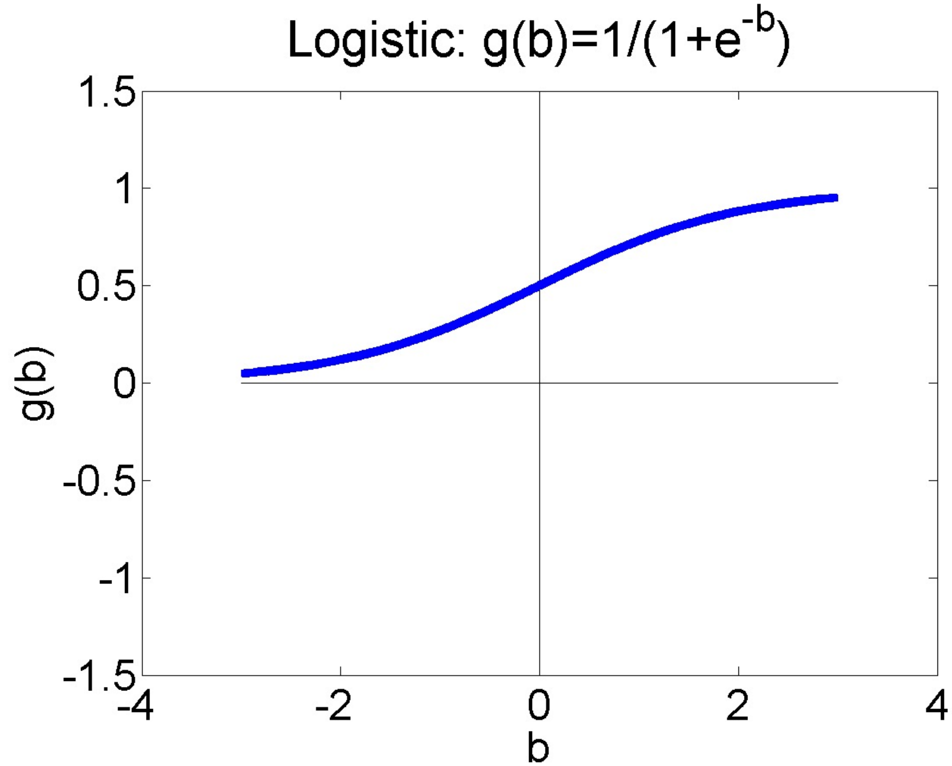
CS440/ECE448 Lecture 8: Logistic Regression

Mark Hasegawa-Johnson, 2/2022

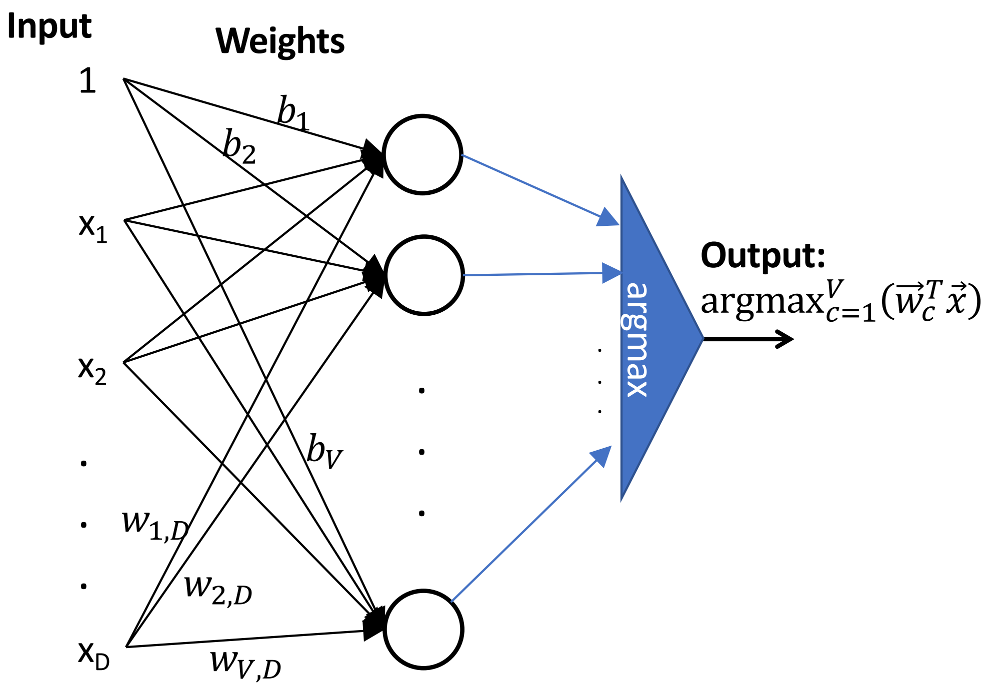


Outline

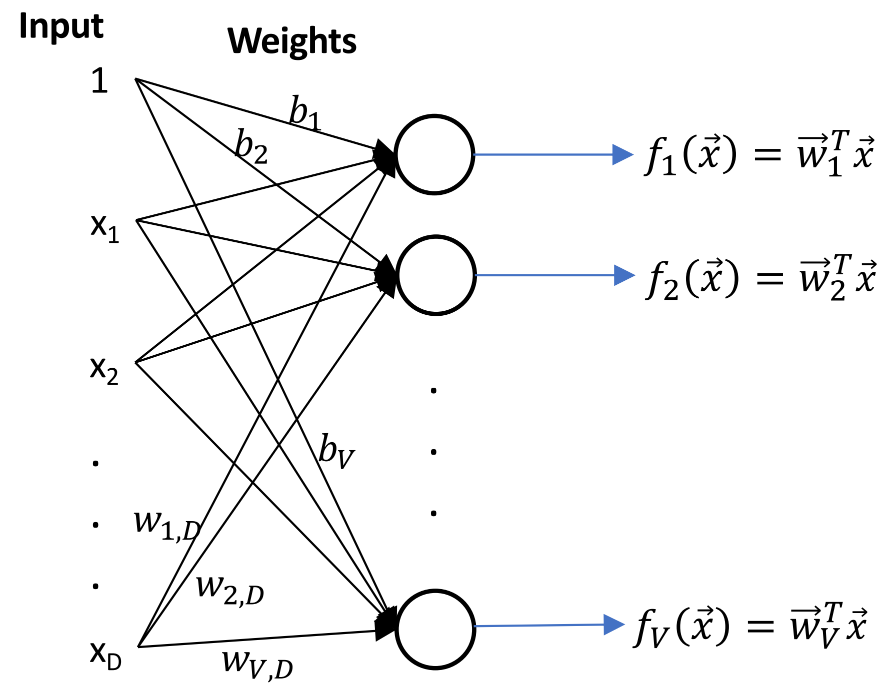
- One-hot vectors: rewriting the perceptron to look like linear regression
- Softmax: Soft category boundaries
- Cross-entropy = negative log probability of the training data
- Stochastic gradient descent for logistic regression

Comparison of Multi-Class Perceptron to Multiple Regression

Multi-Class Perceptron



Multiple Regression



Here's a weird question:

Can we come up with some new notation that can be used to write both the multi-class perceptron AND the linear regression algorithm?

New notation: Don't change the multi-class perceptron algorithm, but make it easier to write

- Instead of defining y_i as an integer, let's define $\vec{y}_i$ to be a vector:

$$\vec{y}_i = \begin{bmatrix} y_{i,1} \\ \vdots \\ y_{i,V} \end{bmatrix}$$

- For a multi-class perceptron, this only makes sense if $\vec{y}_i$ is what's called a **one-hot** vector:

$$y_{i,c} = \begin{cases} 1 & c = \text{true class label of the } i^{th} \text{ token} \\ 0 & \text{otherwise} \end{cases}$$

New notation: Don't change the multi-class perceptron algorithm, but make it easier to write

- Let's also define the output to be a one-hot vector:

$$f(\vec{x}_i) = \begin{bmatrix} f_1(\vec{x}_i) \\ \vdots \\ f_V(\vec{x}_i) \end{bmatrix}$$

… where …

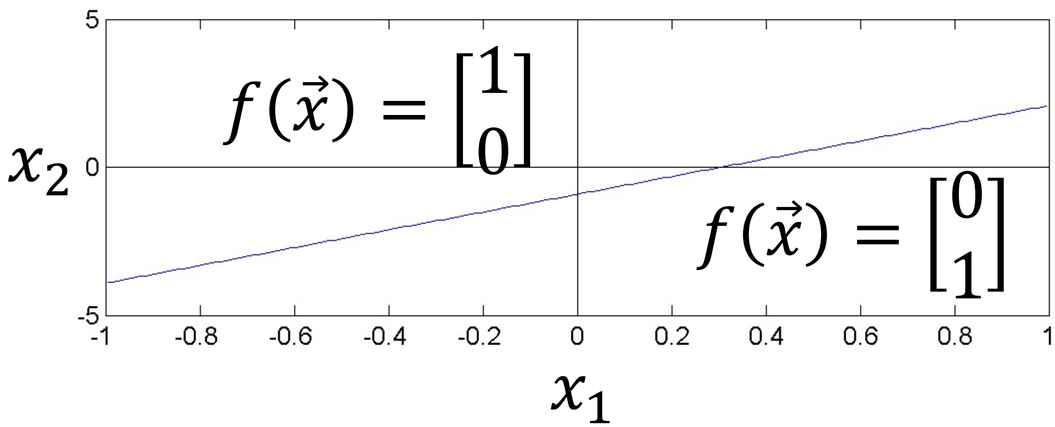
$$f_c(\vec{x}_i) = \begin{cases} 1 & c = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

Example: Binary classifier

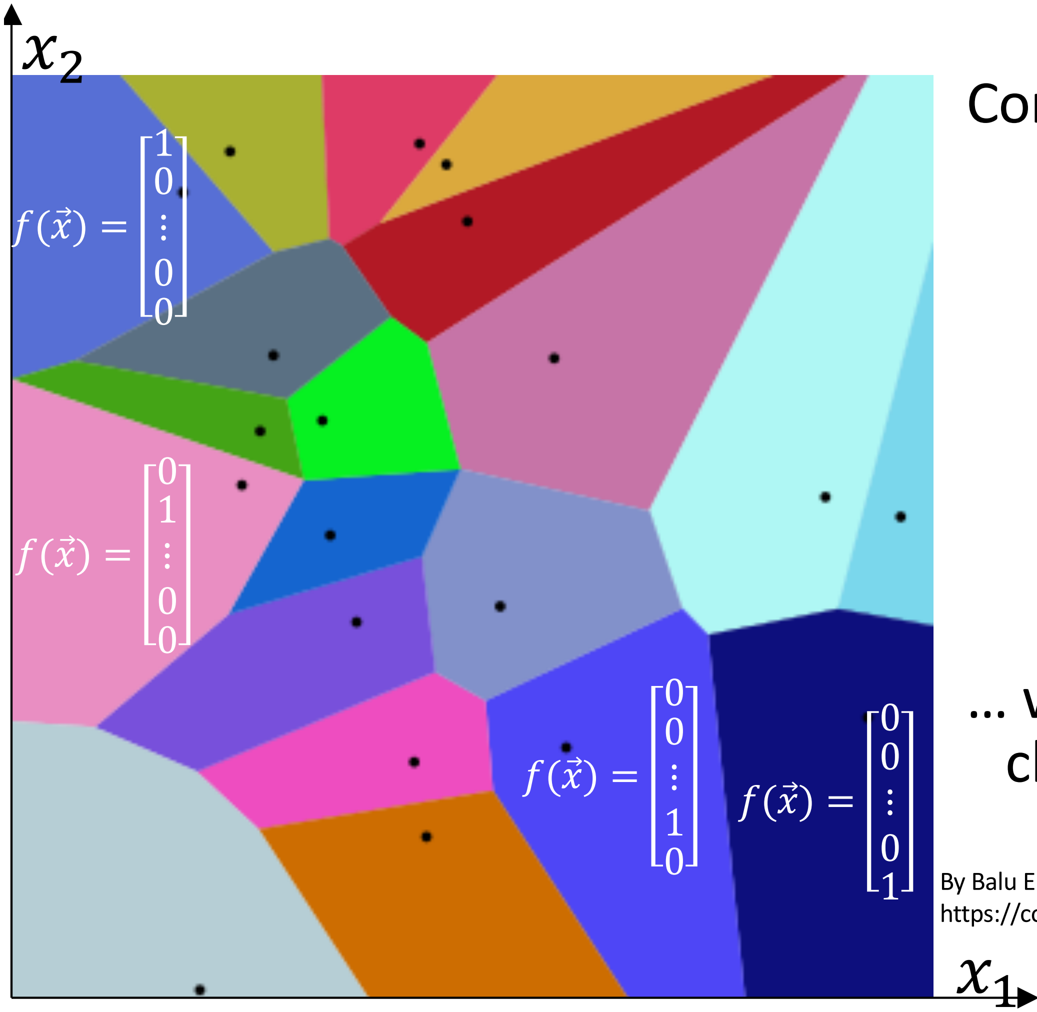
Consider the classifier

$$f(\vec{x}_i) = \begin{bmatrix} f_1(\vec{x}_i) \\ f_2(\vec{x}_i) \end{bmatrix}, \qquad f_c(\vec{x}_i) = \begin{cases} 1 & c = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

… with only two classes. Then the classification regions might look like this:



Multi-Class Linear Classifiers



Consider the classifier

$$f(\vec{x}_i) = \begin{bmatrix} f_1(\vec{x}_i) \\ \vdots \\ f_V(\vec{x}_i) \end{bmatrix},$$

$$f_c(\vec{x}_i) = \begin{cases} 1 & c = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

… with 20 classes. Then some of the classifications might look like this.

Now the perceptron has a vector error, just like linear regression

Now we can define an error term for every output:

$$\vec{\epsilon}_i = \begin{bmatrix} \epsilon_{i,1} \\ \vdots \\ \epsilon_{i,V} \end{bmatrix}, \qquad \epsilon_{i,c} = f_c(\vec{x}_i) - y_{i,c}$$

- If *c* was the correct class label ($y_{i,c} = 1$), but the network didn't get it right ($f_c(\vec{x}_i) = 0$), then it **undershot**:

$$\epsilon_{i,c} = -1$$

- If the network thought the correct answer was *c* ($f_c(\vec{x}_i) = 1$), but it wasn't ($y_{i,c} = 0$), then it **overershot**

$$\epsilon_{i,c} = +1$$

- Otherwise,

$$\epsilon_{i,c} = 0$$

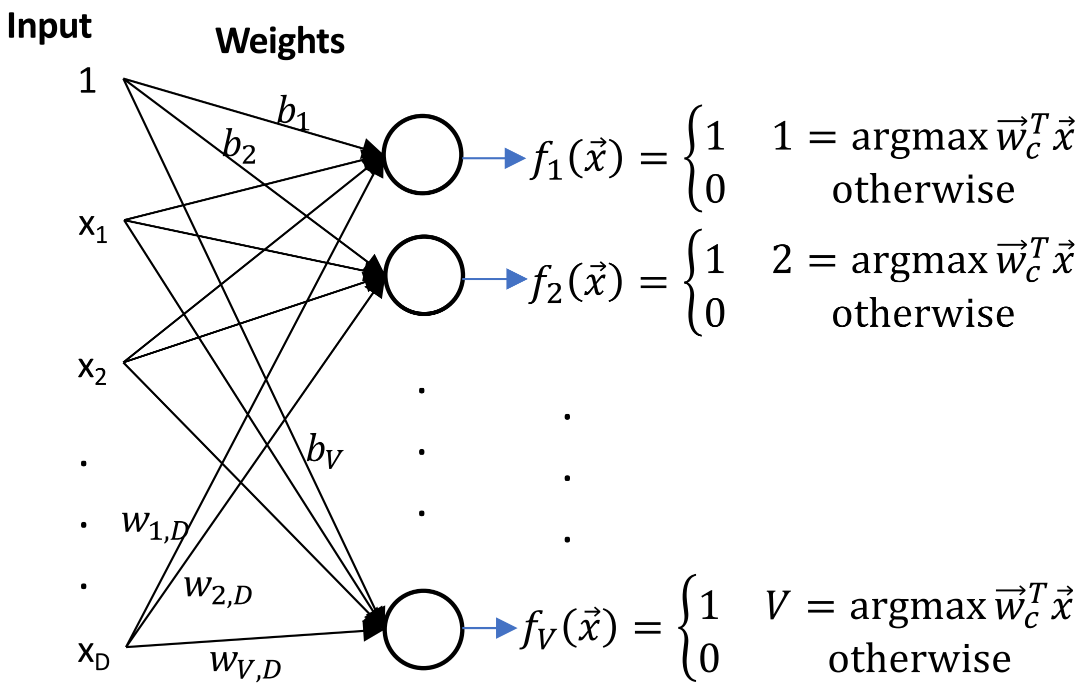
Multi-class perceptron, written in terms of one-hot vectors

But with this definition, we can write the perceptron update the same as the linear regression update:

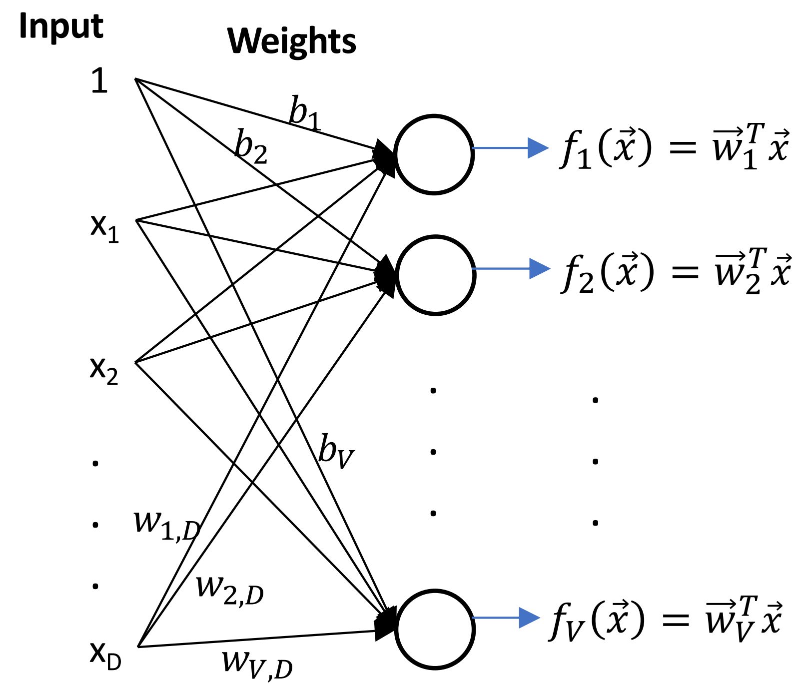
$$\vec{w}_c \leftarrow \vec{w}_c - \eta \epsilon_{i,c} \vec{x}_i = \begin{cases} \vec{w}_c + \eta \vec{x}_i & \epsilon_{i,c} = -1 \\ \vec{w}_c - \eta \vec{x}_i & \epsilon_{i,c} = +1 \\ \vec{w}_c & \epsilon_{i,c} = 0 \end{cases}$$

Comparison of Multi-Class Perceptron to Multiple Regression

Multi-Class Perceptron: One-hot output



Multiple Regression: Real-valued Output

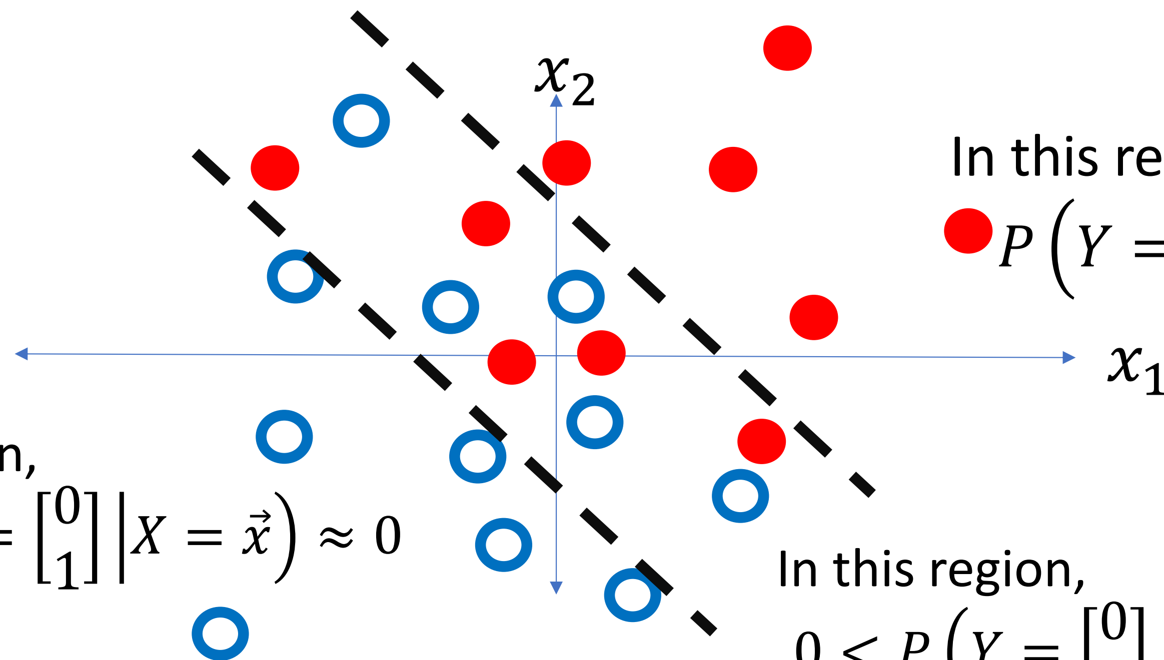


Outline

- One-hot vectors: rewriting the perceptron to look like linear regression
- Softmax: Soft category boundaries
- Cross-entropy = negative log probability of the training data
- Stochastic gradient descent for logistic regression

Probabilistic boundaries

Instead of trying to find the exact boundaries, logistic regression models the probability that token $\vec{x}$ belongs to class $\vec{y}$.



In this region,
$$P\left(Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \middle| X = \vec{x}\right) \approx 1$$

In this region,
$$P\left(Y = Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \middle| X = \vec{x}\right) \approx 0$$

In this region,
$$0 < P\left(Y = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \middle| X = \vec{x}\right) < 1$$

Perceptron versus logistic regression

Remember that for the perceptron, we have

$$f(\vec{x}_i) = \begin{bmatrix} f_1(\vec{x}_i) \\ \vdots \\ f_V(\vec{x}_i) \end{bmatrix}, \qquad f_c(\vec{x}_i) = \begin{cases} 1 & c = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

For logistic regression, we have

$$f(\vec{x}_i) = \begin{bmatrix} f_1(\vec{x}_i) \\ \vdots \\ f_V(\vec{x}_i) \end{bmatrix}, \qquad f_c(\vec{x}_i) = \frac{e^{\vec{w}_c^T \vec{x}}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}}}$$

The softmax function

- This is called the softmax function:

$$\text{softmax}(\vec{x}_i) = \begin{bmatrix} \underset{1}{\text{softmax}}(W^T\vec{x}) \\ \vdots \\ \underset{V}{\text{softmax}}(W^T\vec{x}) \end{bmatrix}, \qquad \underset{c}{\text{softmax}}(W^T\vec{x}) = \frac{e^{\vec{w}_c^T\vec{x}}}{\sum_{k=1}^{V} e^{\vec{w}_k^T\vec{x}}}$$

- …where the matrix W is defined to be

$$W = [\vec{w}_1, \dots, \vec{w}_V]$$

Argmax and Softmax

$$f_c(\vec{x}_i) = \begin{cases} 1 & c = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}, \qquad f_c(\vec{x}_i) = \frac{e^{\vec{w}_c^T \vec{x}}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}}}$$

In both cases, we have:

- $f_c(\vec{x}_i) \geq 0$
- $f_c(\vec{x}_i) \leq 1$
- $\sum_{c=1}^{V} f_c(\vec{x}_i) = 1$

Argmax and Softmax

$$f_c(\vec{x}_i) = \begin{cases} 1 & c = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}, \qquad f_c(\vec{x}_i) = \frac{e^{\vec{w}_c^T \vec{x}}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}}}$$

In both cases, we can interpret these as probabilities:

$$f_c(\vec{x}) = \mathrm{P}(\text{Class} = c | X = \vec{x})$$

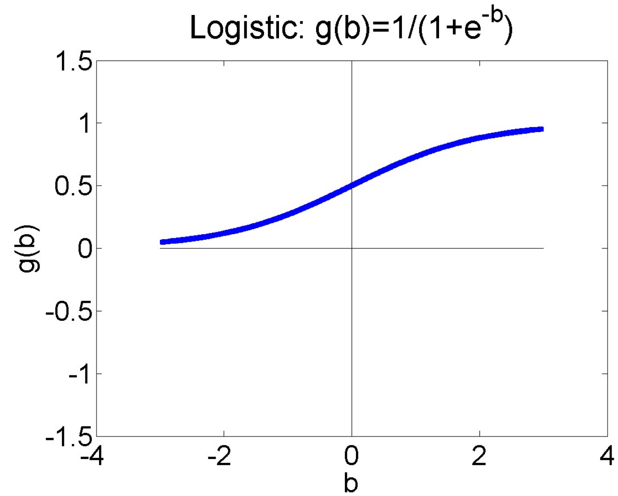
Some details: Logistic function

The probability $\mathrm{P}(\mathrm{Class} = 1 | X = x)$ in the two-class case has an interesting form. It's called the "logistic sigmoid" function:

$$\mathrm{P}(\mathrm{Class} = 1 | X = x) = \mathrm{softmax}(\vec{w}_1^T x) = \frac{e^{\vec{w}_1^T x}}{e^{\vec{w}_1^T x} + e^{\vec{w}_2^T x}} = \frac{1}{1 + e^{-\vec{w}^T x}}$$

where $\vec{w} = \vec{w}_1 - \vec{w}_2$.

Some details: Logistic function



This function,

$$P(\text{Class} = 1|X = x) = \frac{1}{1 + e^{-\vec{w}^T x}}$$

is called the "logistic sigmoid function."

- It's called "sigmoid" because it is S-shaped.
- It was first discovered by Verhulst in the 1830s, as a model of population growth. The idea was that the population grows exponentially until it runs up against resource limitations, and then starts to stagnate.

Logistic Regression

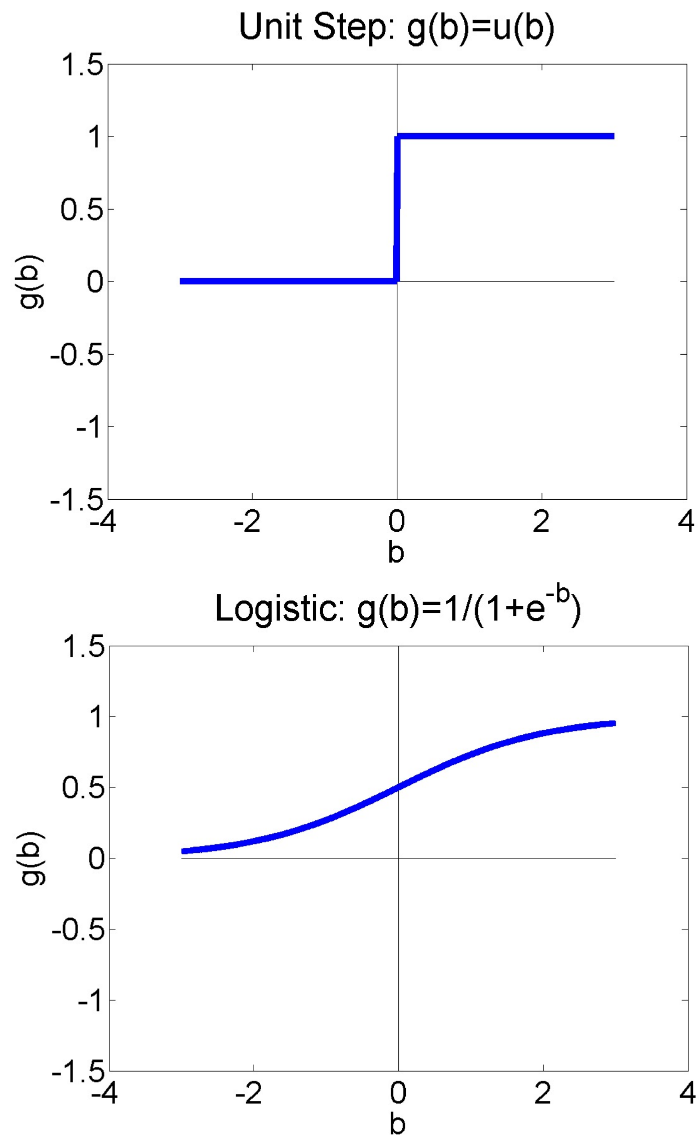
We can frame the basic idea of logistic regression in this way: replace the non-differentiable decision function

$$\hat{y} = \mathrm{u}(\vec{w}^T x)$$

with a differentiable decision function:

$$\hat{y} = \sigma(\vec{w}^T x) = \frac{1}{1 + e^{-\vec{w}^T x}}$$

…so that the classifier can be trained using gradient descent.

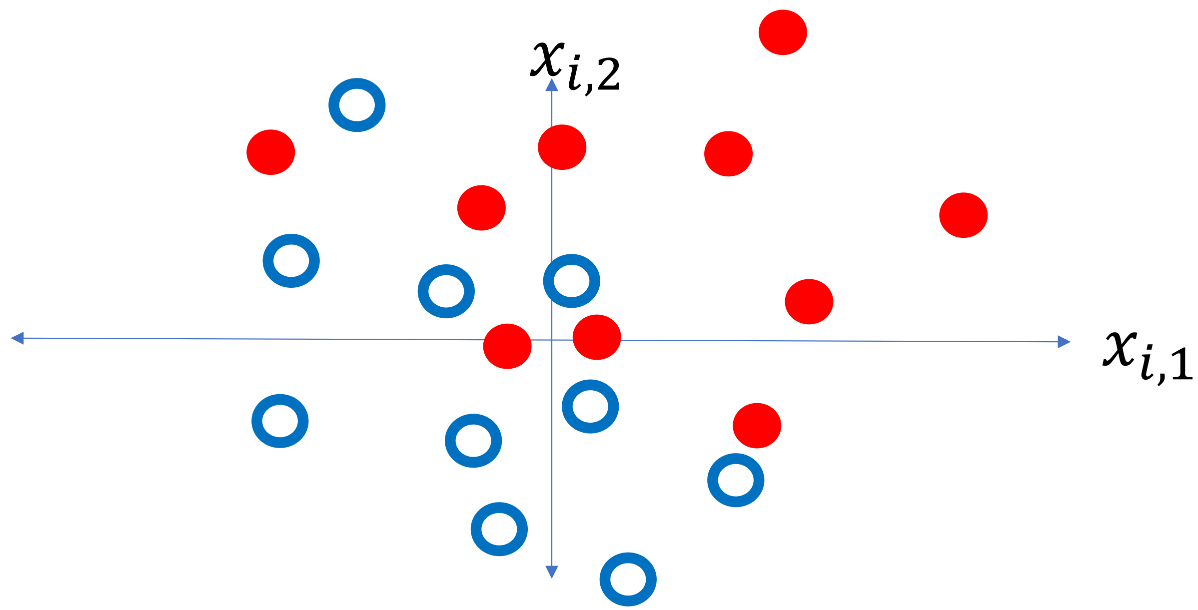


Outline

- One-hot vectors: rewriting the perceptron to look like linear regression
- Softmax: Soft category boundaries
- Cross-entropy = negative log probability of the training data
- Stochastic gradient descent for logistic regression

Learning logistic regression

- Suppose we have some data.
- We want to learn vectors $\vec{w}_c = \left[w_{c,1}, \ldots, w_{c,D}, b_c\right]^T$ so that $\mathrm{P}(\mathrm{Class} = c | X = \vec{x}) = \underset{c}{\mathrm{softmax}}(W^T \vec{x})$.

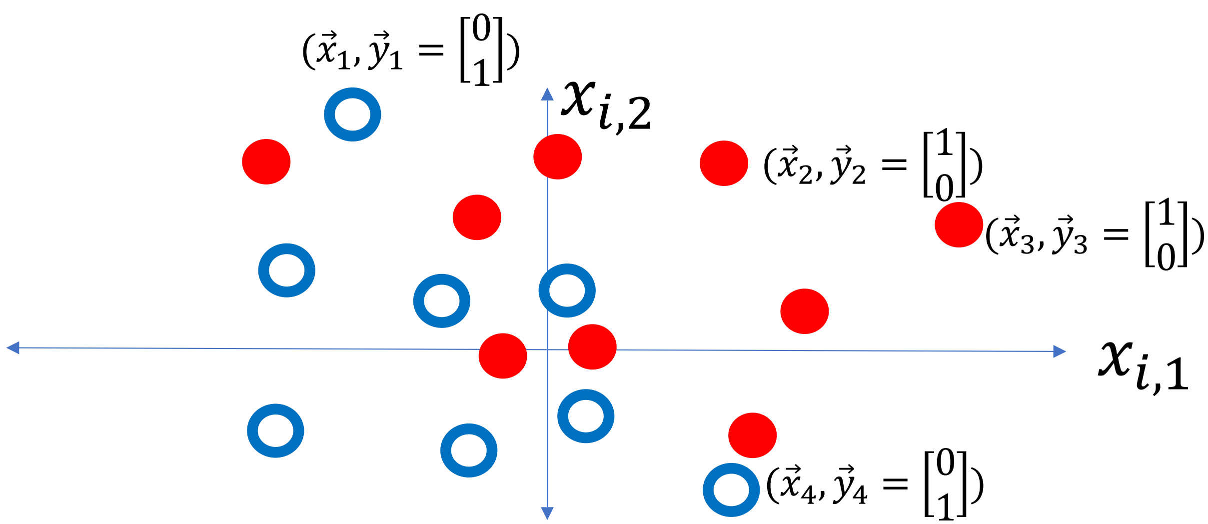


Learning logistic regression: Training data

Data:

$$\mathfrak{D} = \{(\vec{x}_1, c_1), (\vec{x}_2, c_2), \ldots, (\vec{x}_n, c_n)\}$$

where each $\vec{x}_i = \left[x_{i,1}, \ldots, x_{i,D}, 1\right]^T$ is a vector, and each $c_i \in \{1, \ldots, V\}$ is a integer encoding the true class label.



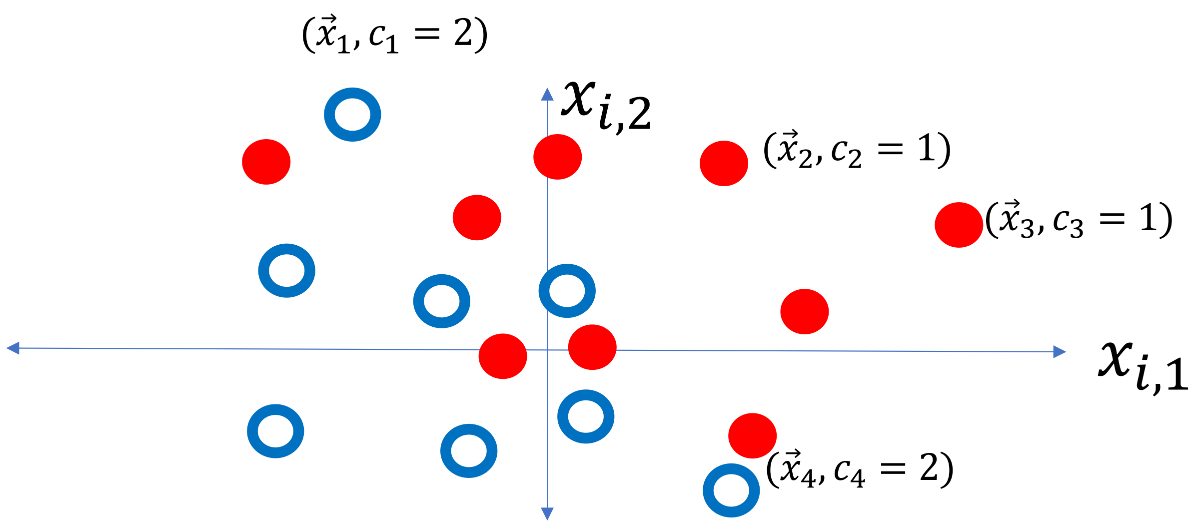
Learning logistic regression: Model parameters

We want to learn the model parameters

$$W = [\vec{w}_1, \dots, \vec{w}_V]$$

so that

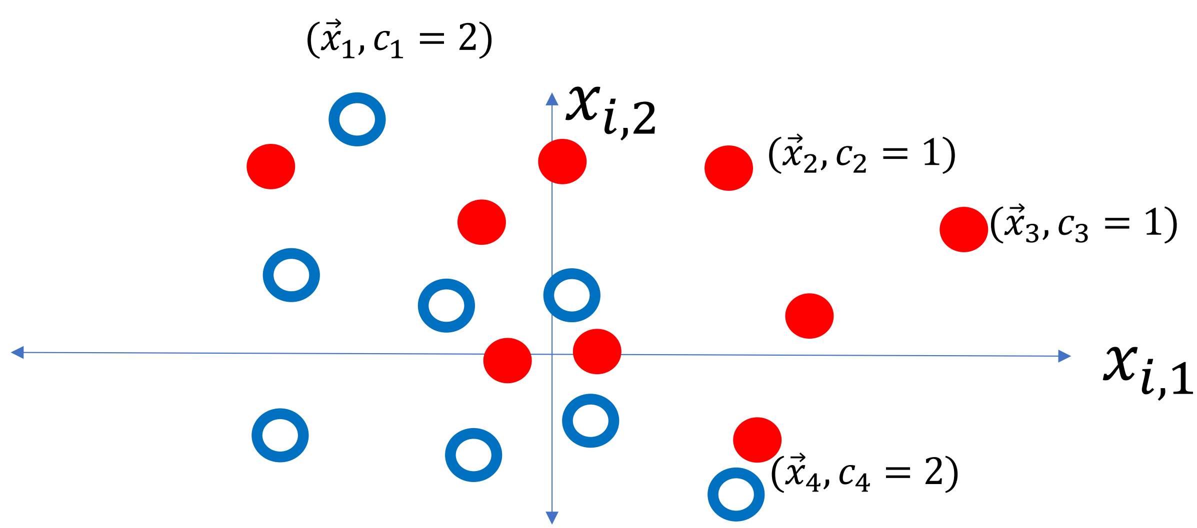
$$\mathrm{P}(\mathrm{C} = c_i | X = \vec{x}_i) = \underset{c_i}{\mathrm{softmax}}(W^T \vec{x}_i)$$



Learning logistic regression: Training criterion

We want to learn the model parameters, $W = [\vec{w}_1, \dots, \vec{w}_V]$, in order to maximize the probability of the observed data:

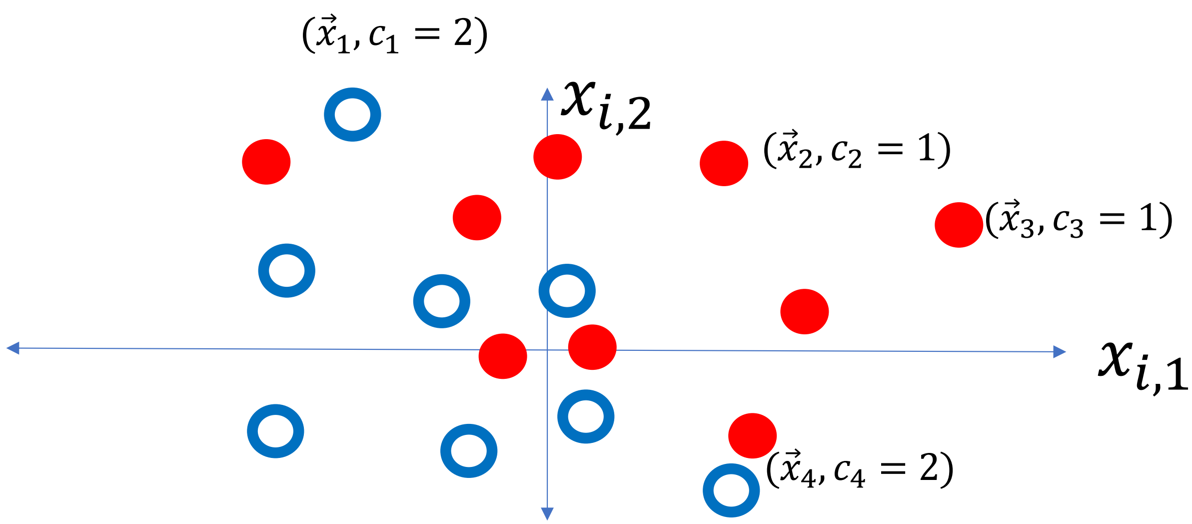
$$P(\mathfrak{D}|W) = \prod_{i=1}^{n} \mathrm{P}(\mathrm{C} = c_i | X = \vec{x}_i)$$



Learning logistic regression

We want to learn the model parameters, $W = [\vec{w}_1, \ldots, \vec{w}_V]$, in order to maximize the probability of the observed data:

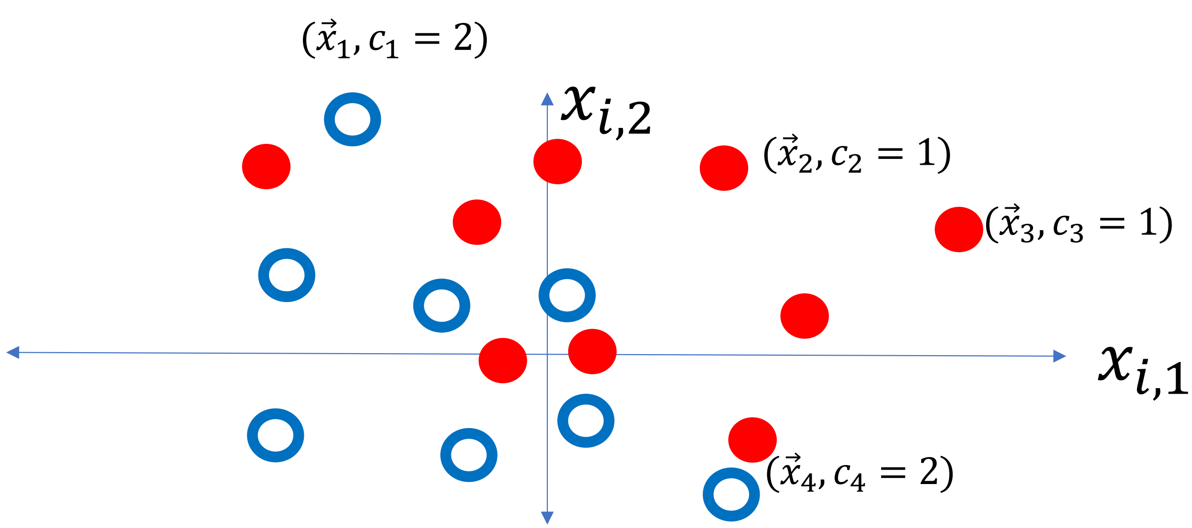
$$P(\mathfrak{D}|W) = \prod_{i=1}^{n} \underset{c_i}{\text{softmax}}(W^T \vec{x}_i)$$



Learning logistic regression

We want to learn the model parameters, $W = [\vec{w}_1, \ldots, \vec{w}_V]$, in order to maximize the probability of the observed data:

$$P(\mathfrak{D}|W) = \prod_{i=1}^{n} \frac{e^{\vec{w}_{c_i}^T \vec{x}_i}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i}}$$



How do you maximize a function?

Our goal is to find $W = [\vec{w}_1, \ldots, \vec{w}_V]$ in order to maximize

$$P(\mathfrak{D}|W) = \prod_{i=1}^{n} \frac{e^{\vec{w}_{c_i}^T \vec{x}_i}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i}}$$

Here are some useful things to know:

1. Logarithm turns products into sums
2. Maximizing $f(W)$ is the same thing as minimizing $-f(W)$

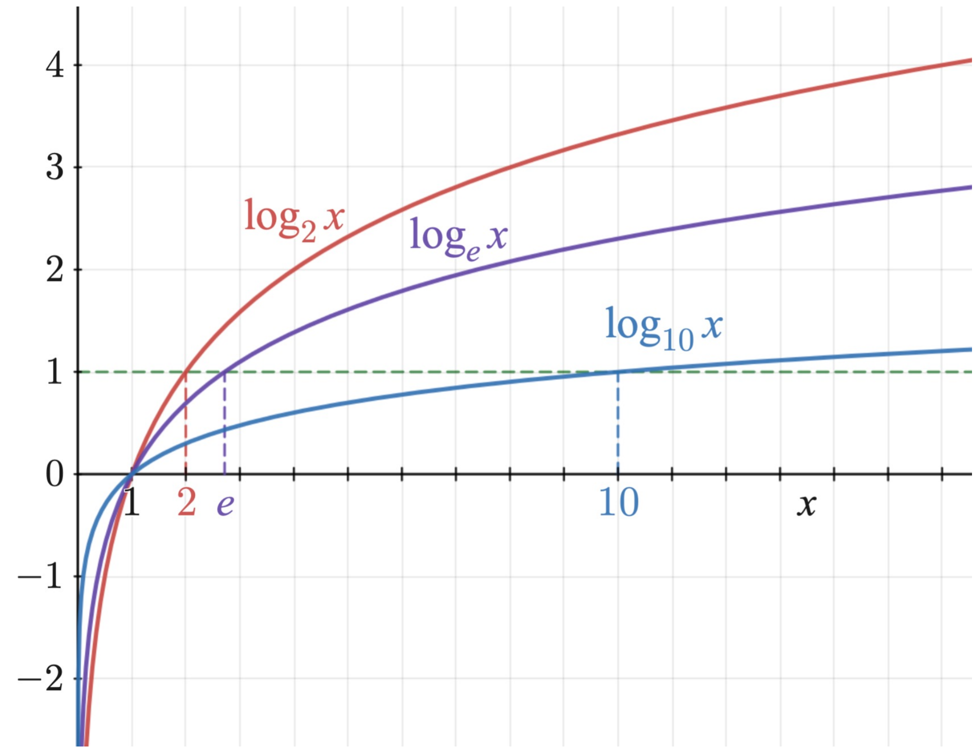
1. Logarithms turn products into sums

$\ln x$ (the natural logarithm of x, shown as $\log_e x$ in the plot at right) is a monotonically increasing function of x.

Since it's monotonically increasing,

$$\underset{W}{\operatorname{argmax}} P(\mathfrak{D}|W) = \underset{W}{\operatorname{argmax}} \ln P(\mathfrak{D}|W)$$

Almost always, maximizing the log probability is easier than maximizing the probability, because logarithms turn products into sums.



Logarithm_plots.png, CC-SA 3.0, Richard F. Lyon, 2011

1. Logarithms turn products into sums

Our goal is to find $W = [\vec{w}_1, \dots, \vec{w}_V]$ in order to maximize

$$\ln P(\mathfrak{D}|W) = \sum_{i=1}^{n} \ln \frac{e^{\vec{w}_{c_i}^T \vec{x}_i}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i}} = \sum_{i=1}^{n} \left(\vec{w}_{c_i}^T \vec{x}_i - \ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} \right)$$

2. Maximizing $f(W)$ is the same thing as minimizing $-f(W)$.

Our goal is to find $W = [\vec{w}_1, \ldots, \vec{w}_V]$ in order to maximize

$$\ln P(\mathfrak{D}|W) = \sum_{i=1}^{n} \left(\vec{w}_{c_i}^T \vec{x}_i - \ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} \right)$$

Choosing W to maximizing $\vec{w}_{c_i}^T \vec{x}_i$ is kind of obvious: just set $\vec{w}_{c_i} = A\vec{x}_i$, where A is a scalar that's as big as possible. Maximizing $-\ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i}$, is not obvious.

2. Maximizing $f(W)$ is the same thing as minimizing $-f(W)$.

To emphasize the hard part of the problem, there is a convention that, instead of maximizing $\ln P(\mathfrak{D}|W)$, we minimize $-\ln P(\mathfrak{D}|W)$:

Our goal is to find $W = [\vec{w}_1, \ldots, \vec{w}_V]$ in order to minimize

$$\mathfrak{L} = -\ln P(\mathfrak{D}|W) = \sum_{i=1}^{n} \left(\ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} - \vec{w}_{c_i}^T \vec{x}_i \right)$$

The curly $\mathfrak{L}$ is a symbol we use to denote a "loss function". A loss function is something you want to minimize.

Some details: Cross entropy

- The loss function is called "cross entropy," because it is similar in some ways to the entropy of a thermodynamic system in physics.
- When you implement this in software, it's a good idea to normalize by the number of training tokens, so that the scale is easier to understand:

$$\mathfrak{L} = -\frac{1}{n}\log P(\mathfrak{D}|W) = -\frac{1}{n}\sum_{i=1}^{n}\log \mathrm{P}(\mathrm{C} = c_i|X = \vec{x}_i)$$

Outline

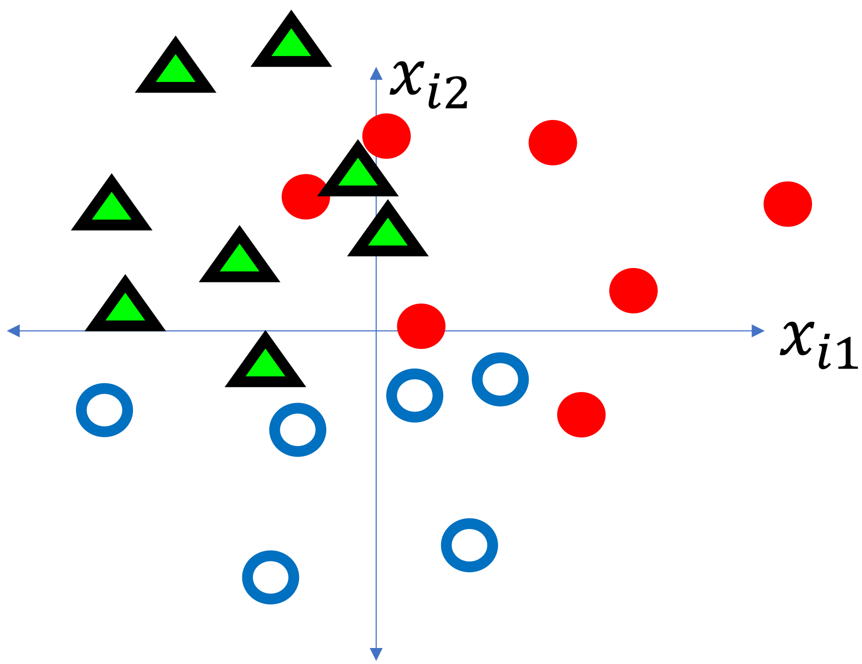
- One-hot vectors: rewriting the perceptron to look like linear regression
- Softmax: Soft category boundaries
- Cross-entropy = negative log probability of the training data
- Stochastic gradient descent for logistic regression

Logistic regression training

- In each iteration, present a batch of training data, $\mathfrak{D} = \{(x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)\}$.
  - If the batch contains all the data, this is called "gradient descent"
  - If the batch contains a randomly chosen subset of the data, this is called "stochastic gradient descent"
- Calculate $\mathrm{P}(\mathrm{Class} = c | X = \vec{x}_i) = \underset{c}{\mathrm{softmax}}(W^T\vec{x})$ for each training token $\vec{x}_i$, for each class c.
- Update all the weight vectors using stochastic gradient descent.

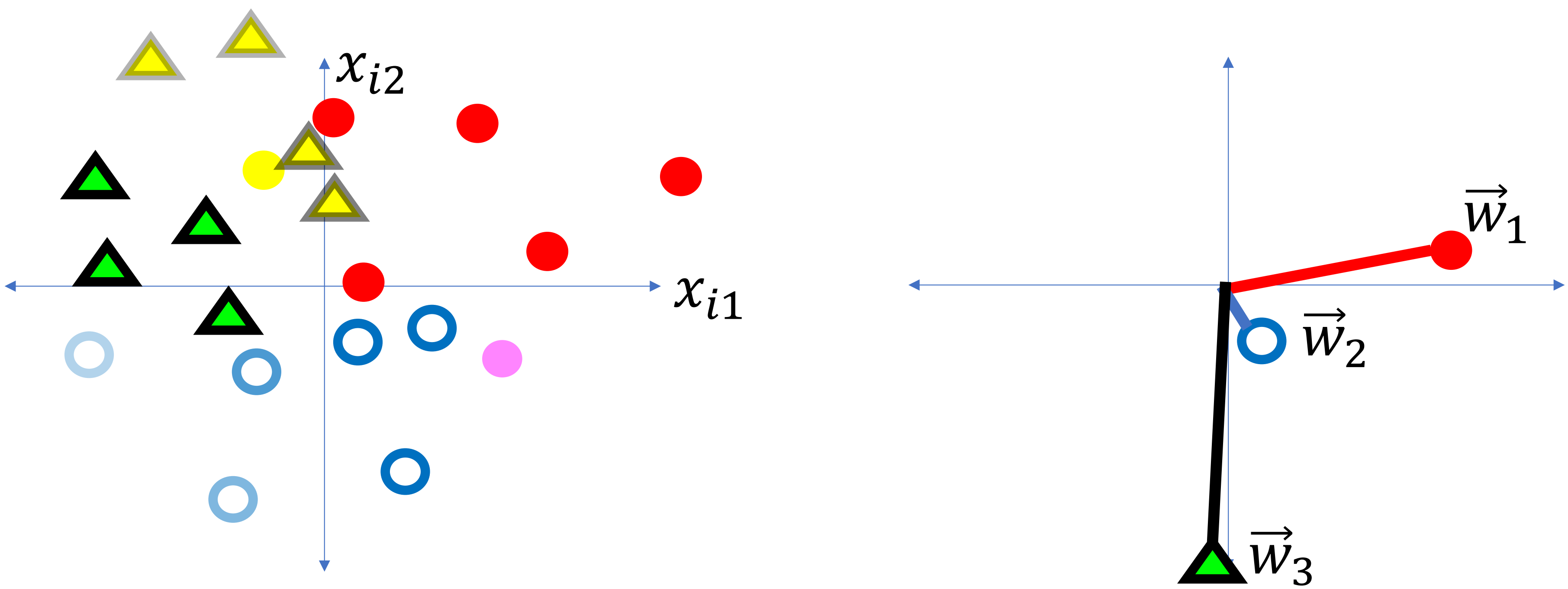
Logistic regression training example

Start with the given dataset $\mathfrak{D}$. Here the true class is indicated by both color and shape.



Logistic regression training example

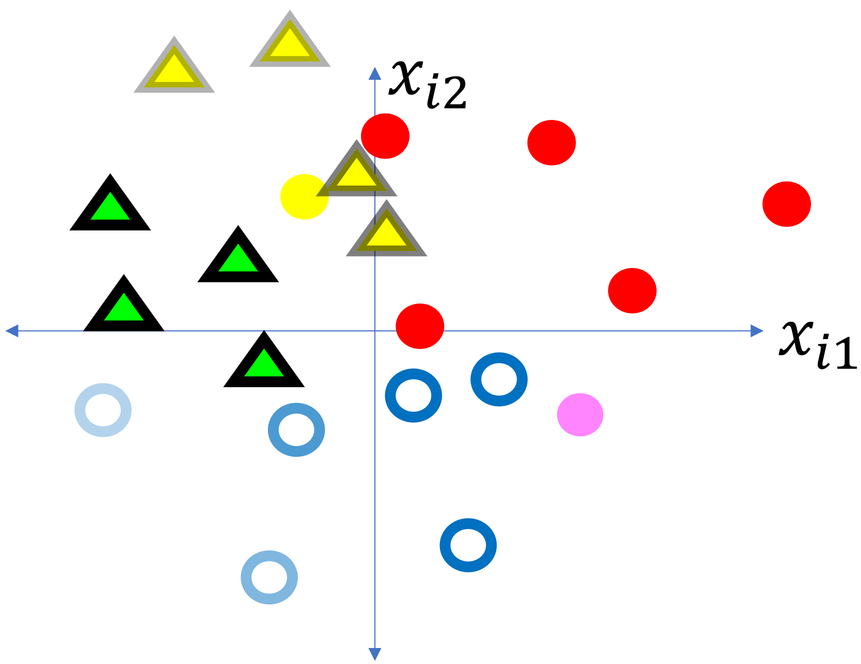
Randomly initialize the weight vectors, and then calculate the probabilities $\mathrm{P}(\mathrm{Y} = c|X = x_i)$ for every class c, for every training token (shown as transparency and color change, left side)

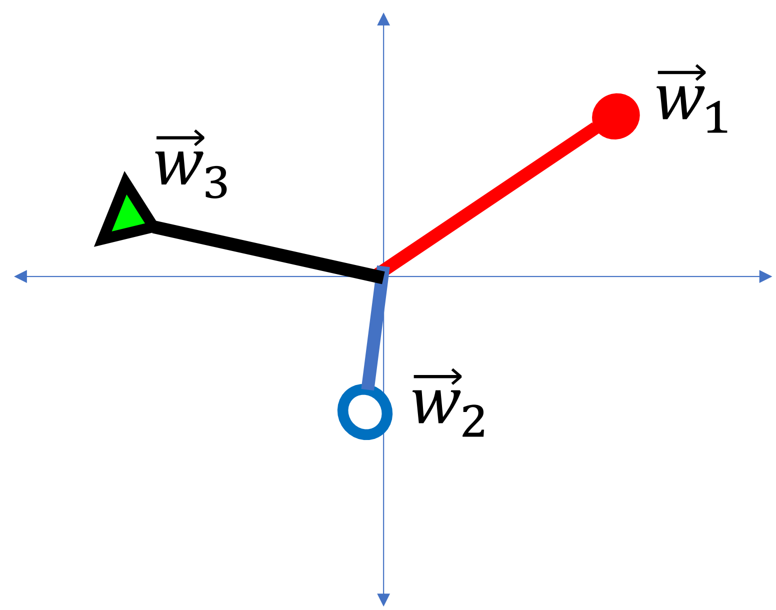


Logistic regression training example

Modify the weight vectors to reduce the loss function, as

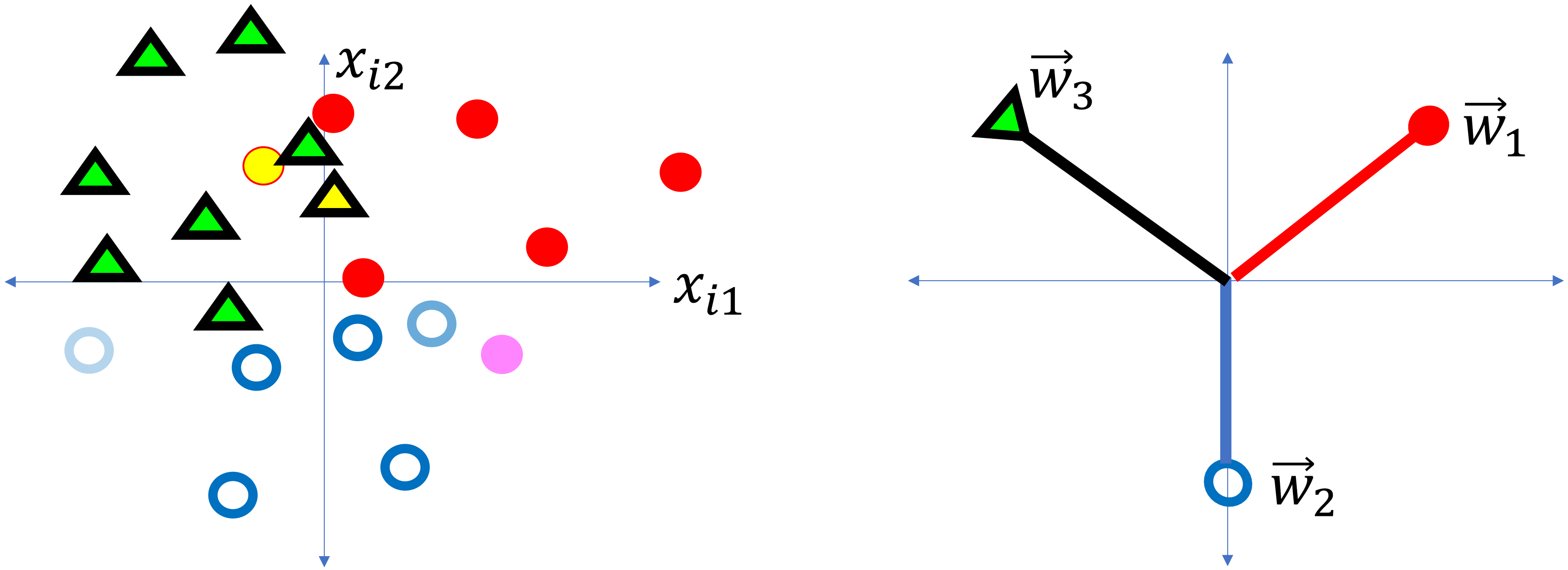
$$\vec{w}_c \leftarrow \vec{w}_c - \eta \nabla_{\vec{w}_c} \mathfrak{L}$$





Logistic regression training example

Repeat until the loss stops decreasing: $\vec{w}_c \leftarrow \vec{w}_c - \eta \nabla_{\vec{w}_c} \mathfrak{L}$



Stochastic gradient descent

Our goal is to find $W = [\vec{w}_1, \ldots, \vec{w}_V]$ in order to minimize

$$\mathfrak{L} = -\ln P(\mathfrak{D}|W) = \sum_{i=1}^{n} \left(\ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} - \vec{w}_{c_i}^T \vec{x}_i \right)$$

Just like in linear regression, let's do that one token at a time. Choose a training token $(\vec{x}_i, c_i)$, and try to minimize

$$\mathfrak{L}_i = -\ln P(C = c_i | X = \vec{x}_i) = \ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} - \vec{w}_{c_i}^T \vec{x}_i$$

Stochastic gradient descent

Our goal is to find $W = [\vec{w}_1, \ldots, \vec{w}_V]$ in order to minimize

$$\mathfrak{L}_i = \ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} - \vec{w}_{c_i}^T \vec{x}_i$$

We do that by adjusting $\vec{w}_c \leftarrow \vec{w}_c - \eta \nabla_{\vec{w}_c} \mathfrak{L}_i$, where

- η is called the learning rate. Typically $\eta \approx 0.001$, but it's very hard to know in advance what learning rate will work for a particular problem; you need to experiment to see what works.
- $\nabla_{\vec{w}_c} \mathfrak{L}_i$ is the gradient of the loss with respect to $\vec{w}_c$.

The gradient of the cross-entropy of a softmax

Now, let's calculate that gradient.

$$\nabla_{\vec{w}_c}\mathfrak{L}_i = \nabla_{\vec{w}_c}\left(\ln\sum_{k=1}^{V} e^{\vec{w}_k^T\vec{x}_i}\right) - \nabla_{\vec{w}_c}\left(\vec{w}_{c_i}^T\vec{x}_i\right)$$

$$= \nabla_{\vec{w}_c}\left(\ln\sum_{k=1}^{V} e^{\vec{w}_k^T\vec{x}_i}\right) - y_{i,c}\vec{x}_i$$

…where $y_{i,c}$ is our old friend the one-hot vector:

$$y_{i,c} = \begin{cases} 1 & c_i = c \\ 0 & \text{otherwise} \end{cases}$$

The gradient of the cross-entropy of a softmax

$$\nabla_{\vec{w}_c} \mathfrak{L}_i = \nabla_{\vec{w}_c} \left(\ln \sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i} \right) - y_{i,c} \vec{x}_i$$

$$= \frac{e^{\vec{w}_c^T \vec{x}_i}}{\sum_{k=1}^{V} e^{\vec{w}_k^T \vec{x}_i}} \vec{x}_i - y_{i,c} \vec{x}_i$$

$$= \left(f_c(\vec{x}_i) - y_{i,c} \right) \vec{x}_i$$

$$= \epsilon_{i,c} \vec{x}_i$$

Conclusion

- Perceptron:

$$\epsilon_{i,c} = f_c(\vec{x}_i) - y_{i,c}, \qquad \vec{w}_c \leftarrow \vec{w}_c - \eta\epsilon_{i,c}\vec{x}_i$$

- Linear Regression:

$$\epsilon_{i,c} = f_c(\vec{x}_i) - y_{i,c}, \qquad \vec{w}_c \leftarrow \vec{w}_c - \eta\epsilon_{i,c}\vec{x}_i$$

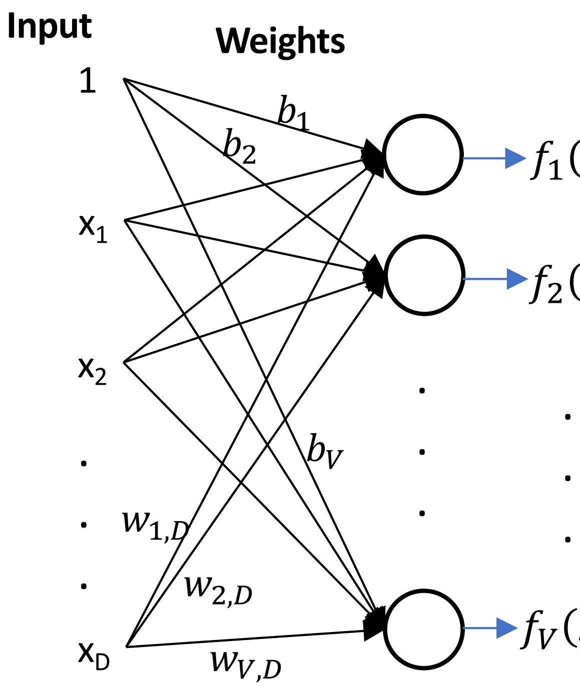
- Logistic Regression:

$$\epsilon_{i,c} = f_c(\vec{x}_i) - y_{i,c}, \qquad \vec{w}_c \leftarrow \vec{w}_c - \eta\epsilon_{i,c}\vec{x}_i$$

The only difference is how you define the network output (argmax, linear, or softmax).

Comparison of Multi-Class Perceptron to Multiple Regression

Multi-Class Perceptron: One-hot output

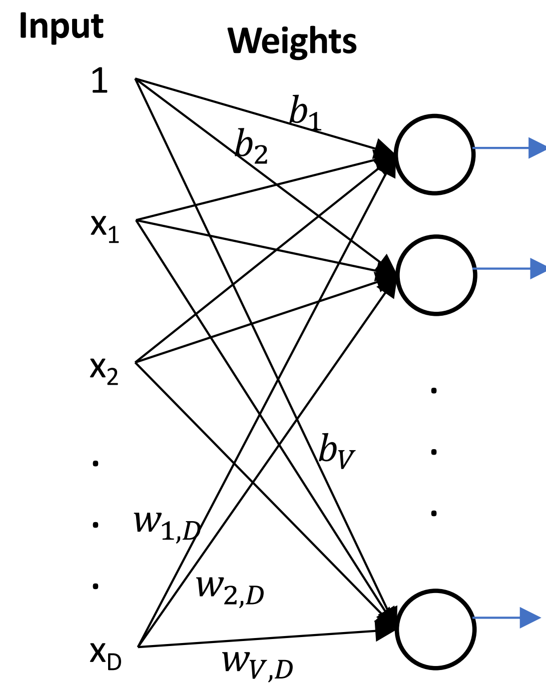


$$f_1(\vec{x}) = \begin{cases} 1 & 1 = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(\vec{x}) = \begin{cases} 1 & 2 = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

$$f_V(\vec{x}) = \begin{cases} 1 & V = \operatorname{argmax} \vec{w}_c^T \vec{x} \\ 0 & \text{otherwise} \end{cases}$$

Multiple Regression: Real-valued Output



$$f_1(\vec{x}) = \vec{w}_1^T \vec{x}$$

$$f_2(\vec{x}) = \vec{w}_2^T \vec{x}$$

$$f_V(\vec{x}) = \vec{w}_V^T \vec{x}$$

Logistic Regression

Logistic Regression:
Output is a vector of probabilities