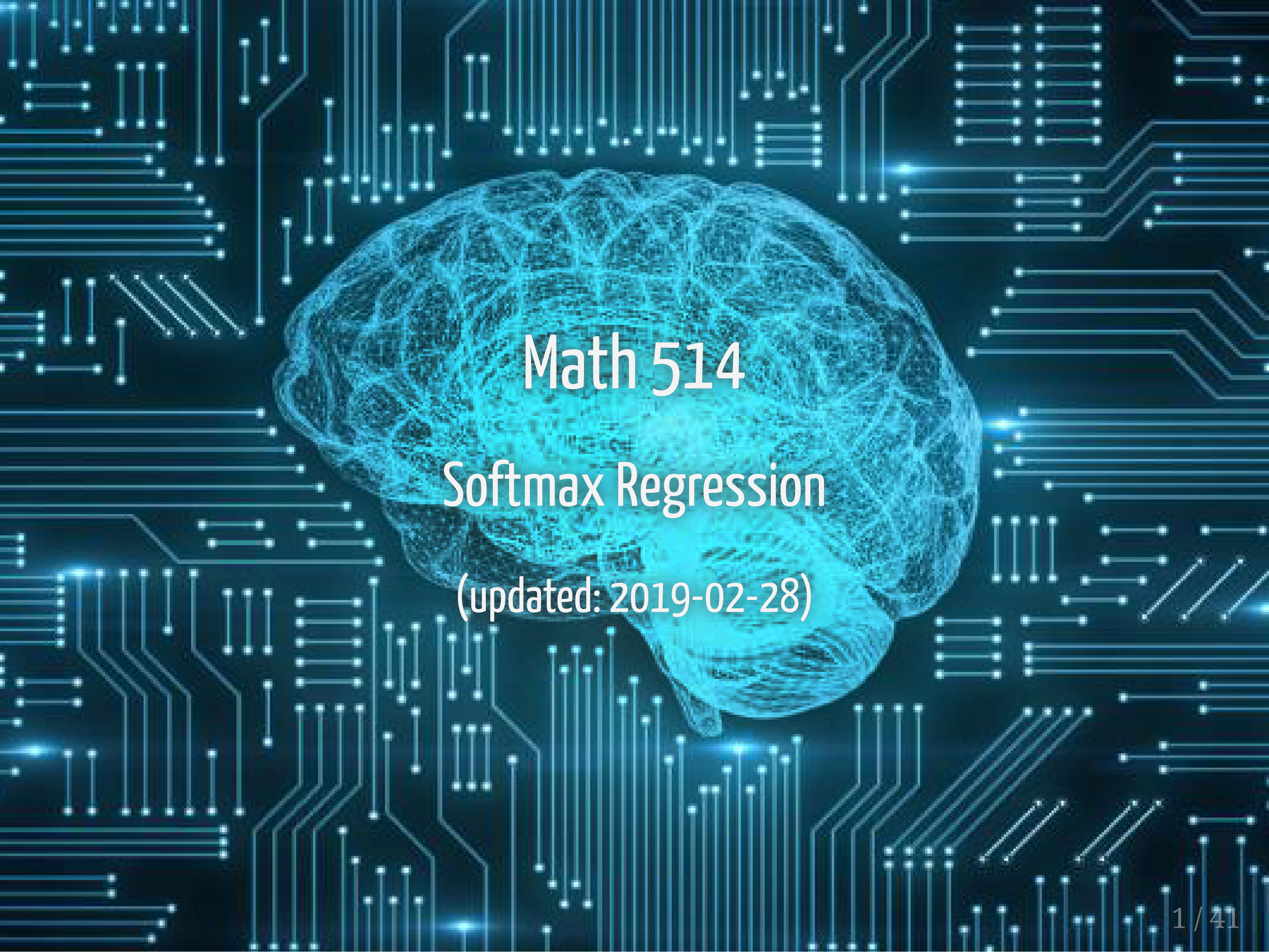


Math 514

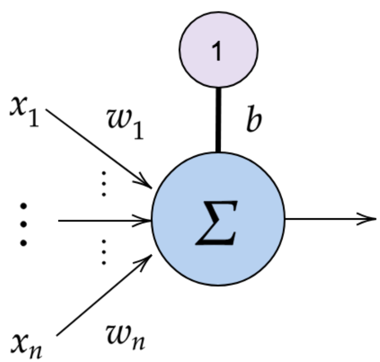
Softmax Regression

(updated: 2019-02-28)

Class 6

So far we've considered 3 networks and 3 cost functions

- **Perceptron** with error correcting algorithm
- **Linear regression** with squared error cost
- **Binary classification** with negative log likelihood cost



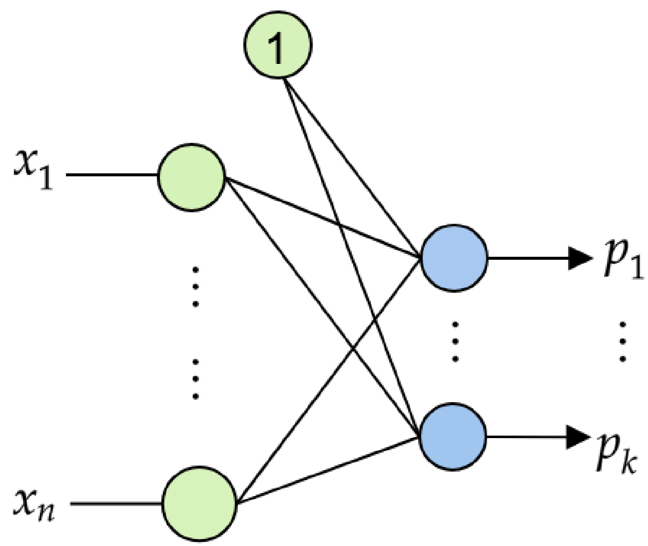
$$y = sign\left(b + w^T x\right)$$

$$y = b + w^T x$$

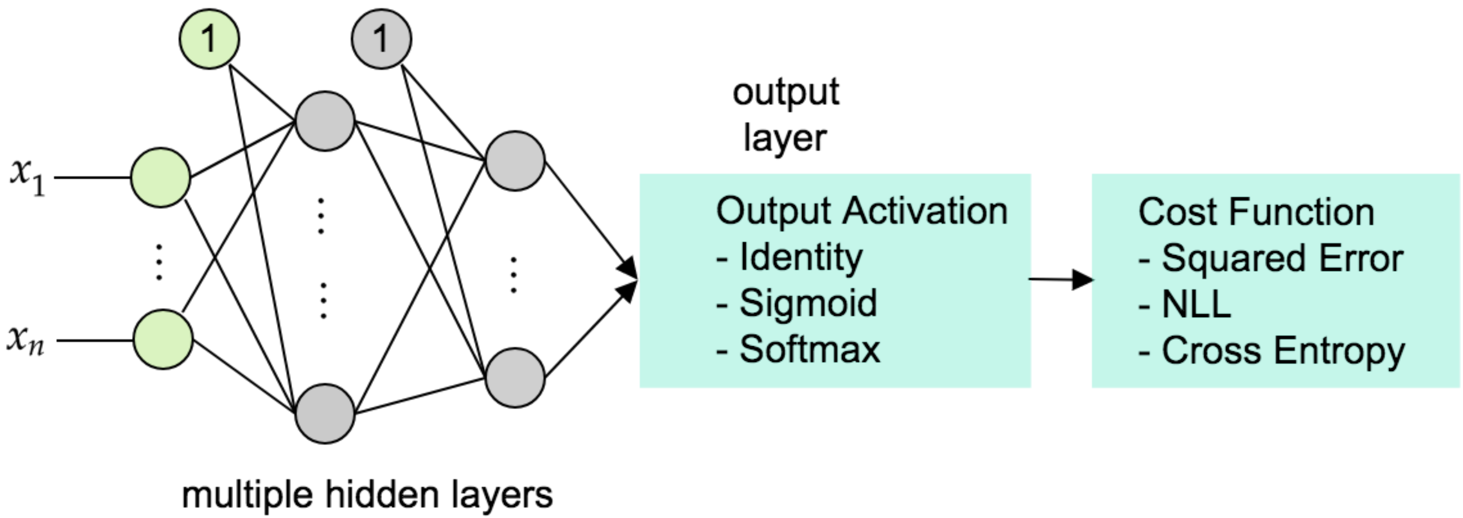
$$y = \sigma\left(b + w^T x\right)$$

Softmax Regression

Multi-category classification (Softmax) with cross entropy cost



Network Models



Softmax

- Introduces 3^{rd} cost function: Cross-Entropy
- Will need to cover some information theory
- Cross-Entropy is
    - Differentiable in adjustable parameters
    - A minimum when performance is optimal
    - Convex if no hidden layers
- Starting next week will write code that supports all 3 cost functions
    - Squared Error
    - Negative Log-Likelihood
    - Cross-Entropy

Information

Compare:

- Outcomes of a fair coin toss
- Outcomes of a fair 6-sided die toss

In each case the outcome removes all uncertainty

- More was learned from the roll of the die:

$$P_t(toss) = \frac{1}{2}$$
$$P_r(roll) = \frac{1}{6}$$

Want information

$$I(roll) > I(toss)$$

What about using surprise

$$I(event) = \frac{1}{p}?$$

Not additive for independent events:

$$\begin{aligned} I(toss, roll) &= \frac{1}{P(toss, roll)} \\ &= \frac{1}{P(toss)} \cdot \frac{1}{P(roll)} = 12 \\ &\neq I(toss) + I(roll) = 8 \end{aligned}$$

Information

Taking log gives desired additivity:

$$I(event) = \log(\frac{1}{P(event)}) = \log(\text{surprise})$$

Since log is monotone increasing, $\log(1/p)$ increases with decreasing p.

- Now:

$$\begin{aligned}
I(toss, roll) &= \log \frac{1}{P(toss, roll)} \\
&= \log \frac{1}{P(toss) \cdot P(roll)} \\
&= \log(\frac{1}{P(toss)}) + \log(\frac{1}{P(roll)}) \\
&= I(toss) + I(roll)
\end{aligned}$$

An event E that has probability $P(E)$ of occurrence has information content

$$I(E) = \log(\frac{1}{P(E)}) = -\log(P(E))$$

- If $\log$ is $\log_2$, information is in bits.
- If $\log$ is natural log, information is in nits.

Information

Examples

- Fair coin toss, $I = \log_2(\frac{1}{1/2}) = 1$
- Fair die roll, $I = \log_2(\frac{1}{1/6}) \approx 2.6$
- Sequence of k fair coin tosses has $P = (\frac{1}{2})^k, \quad I = \log(\frac{1}{1/2^k}) = k$

Entropy

Entropy is a measure of the average information of a distribution

$$\begin{aligned} H(\mathbf{x}) &= E[I(\mathbf{x}))] \\ &= E[\log(\frac{1}{P})] \\ &= -E[\log P] \\ &= -\sum_{\mathbf{x}_i} P(\mathbf{x}_i) \log P(\mathbf{x}_i) \end{aligned}$$

Information

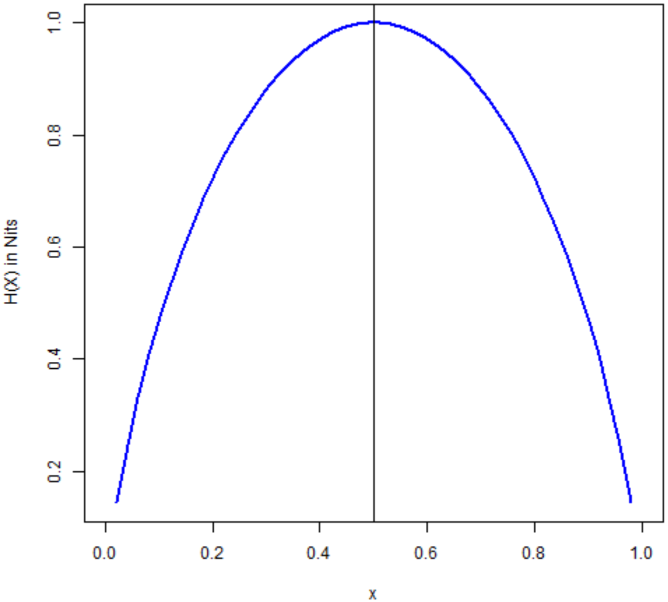
Example:

Let X model the outcome of a coin toss with probability of success (head) p.

$$H(X) = -\sum_{\mathbf{x}_i} P(\mathbf{x}_i) \log P(\mathbf{x}_i)$$
$$= -p \log p - (1-p) \log(1-p)$$

- Entropy is maximized when $p = \frac{1}{2}$
  - (Set $H^{'} = 0$ and solve)

- Entropy is zero when outcome is a certain value



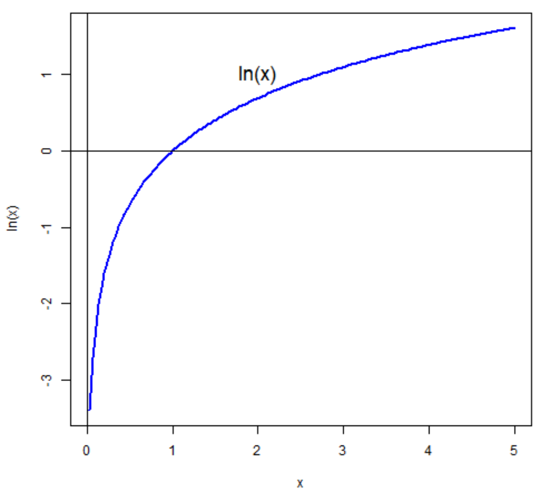
Information

- $\ln(x)$ is monotone increasing

$$\frac{d}{dx}\ln(x) = \frac{1}{x} > 0$$

- $\ln(x)$ is concave

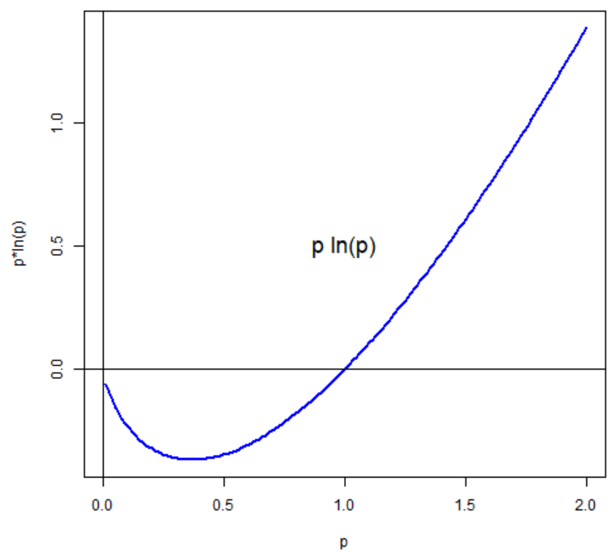
$$\frac{d^2}{dx^2}\ln(x) = -\frac{1}{x^2}$$



$$\lim_{p \to 0} p \cdot \log p = 0$$

Plot shows $\lim_{\mathbf{x} \downarrow 0} \mathbf{x} \log \mathbf{x} = 0$.

The missing point at the origin is due to R returning *NaN* at for $\ln(0)$



Information

Properties of Entropy

Entropy is completely defined given a set of probabilities

$$H(p_1, \cdots, p_k) = \sum_i p_i \log \frac{1}{p_i}$$

$$\sum p_i = 1$$

Entropy is non-negative

$$H(X) \geq 0 \qquad \text{because}$$

$$p \in [0,1], \frac{1}{p} \geq 1$$

$$\log \frac{1}{p} \geq 0$$

(1) Entropy H(p) is maximized when $p_i = \frac{1}{k};\ i = 1, \ldots, k$

(2) For any distribution $q_i = P(Y = x_i), q_i \neq p_i$ for some i

$$H(X) = \sum p_i \log \frac{1}{p_i} < \sum_i p_i \log \frac{1}{q_i}$$

(3) $H(X) \leq \log k$, with equality if $p_i = \frac{1}{k}$ for all i

(4) $H(X)$ decreases as X becomes less uniform

Jensen's Inequality

Convexity

A function f is convex provided

$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$

Jensen's Inequality (finite)

Generalizes the statement of convexity to n terms. If $p_i \in [0,1]$ and $\sum_i p_i = 1$ then Jensen's inequality states that for convex f

$$f(\sum_i p_i x_i) \leq \sum_i p_i f(x_i)$$

The definition of convexity shows relation is true fo $n = 2$.

Induction

Assume result true for n and show true for $n+1$

$$f(\sum_{i=1}^{n+1} p_i x_i) = f(p_1 x_1 + (1-p_1) \sum_{i=2}^{n+1} \frac{p_i}{1-p_1} x_i)$$

$$\leq p_1 f(x_1) + (1-p_1) f(\sum_{i=2}^{n+1} \frac{p_i}{1-p_1} x_i)$$

The result follows because the term on the right is a sum of n terms with coefficients summing to 1

$$\sum_{i=1}^{n+1} p_i = 1 \qquad \Longrightarrow \qquad \sum_{i=2}^{n+1} p_i = 1 - p_1$$

Properties of Entropy

If X is a random variable that takes on k values, then

$$H(X) \leq \ln k$$

The result follows from Jensen's inequality for concave functions. $\ln$ is concave, so

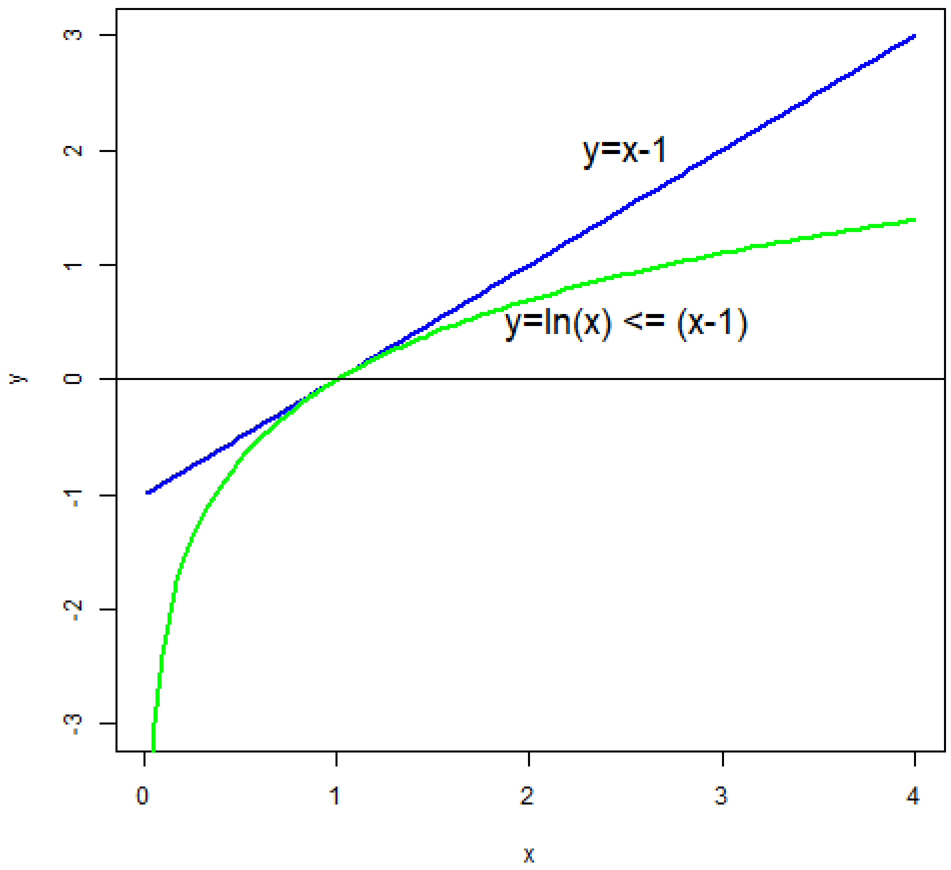
$$\begin{aligned} H(X) &= \sum_{i=1}^{k} p_i \ln \frac{1}{p_i} \\ &\leq \ln(\sum_{i=1}^{k} p_i \cdot \frac{1}{p_i}) \\ &= \ln k \end{aligned}$$

When X is uniform, $p_i = k^{-1}, i = 1, \ldots, k$

$$\begin{aligned} H(\frac{1}{k}, \ldots, \frac{1}{k}) &= \sum_{i=1}^{k} \frac{1}{k} \ln k \\ &= \ln k \end{aligned}$$

It follows that entropy is maximized when X is uniform

Bound on ln(x)



Kullback-Liebler Divergence

KL Divergence is used as measure of similarity between distributions. For $X \sim p(x_i)$

$$D_{KL}(p||q) = \sum_{x_i} p(x_i) \ln \frac{p(x_i)}{q(x_i)} = -\sum_{x_i} p(x_i) \ln \frac{q(x_i)}{p(x_i)}$$

Note that if $p(x_i) = q(x_i)$ for all i then $D_{KL}(p||q) = 0$. Using $-\ln x \geq -(x-1)$ gives

$$\begin{aligned} D_{KL}(p||q) &\geq -\sum_i p_i(\frac{q_i}{p_i} - 1) \\ &= \sum_i p_i - \sum_i q_i \\ &= 1 - \sum_i q_i \\ &\geq 0 \end{aligned}$$

The lower bound will be 0 if q has the same support as p

Since KL Divergence is non-negative and equal to zero when the distributions match, it can function as a distance between distributions. It is not called a distance because in general

$$D_{KL}(p||q) \neq D_{KL}(q||p)$$

Cross Entropy

When training a model want the generated distribution to match the empirical data distribution. It makes sense to use the KL Divergence as a cost function.

If $X \sim p(x_i)$ is empirical data distribution and $Y \sim q(x_i)$ is generated by model training

$$\begin{aligned} D_{KL}(p||q) &= \sum_{x_i} p(x_i) \ln \frac{p(x_i)}{q(x_i)} \\ &= \sum_{x_i} p(x_i) \ln p(x_i) \\ &\qquad - \sum_{x_i} p(x_i) \ln q(x_i) \\ &= H(p,q) - H(p) \end{aligned}$$

$$H(p,q) = -\sum_{x_i} p(x_i) \ln q(x_i)$$

The new symbol $H(p,q)$ is called the **cross-entropy**.

$$H(p,q) = H(p) + D_{KL}(p||q)$$

$$D_{KL}(p||q) \geq 0 \quad \Rightarrow \quad H(p) \leq H(p,q)$$

Want to train the model to make $Y \sim X$. Because $H(p)$ is not affected by changing model parameters, minimizing $D_{KL}(p||q)$ is equivalent to minimizing the cross-entropy $H(p,q)$.

It is standard to use the cross-entropy cost function for multi-category classification.

Sigmoid (Logistic) Function

Recall that the Sigmoid function maps (squashes) any set of real numbers to the interval $[0, 1]$

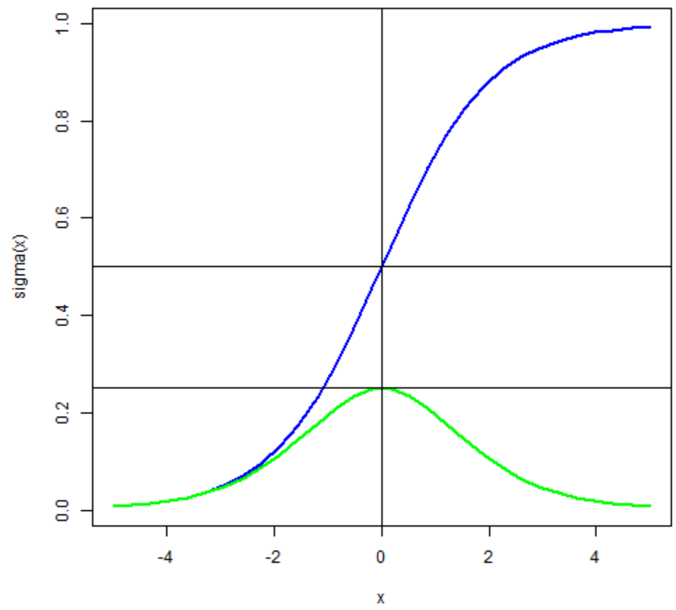
Standard Sigmoid Function

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma(0) = \frac{1}{2}$$

$$\sigma' = \sigma(1-\sigma), \quad \sigma'(0) = \frac{1}{4}$$

$$\lim_{x \uparrow +\infty} \sigma(x) = 1, \lim_{x \downarrow -\infty} \sigma(x) = 0$$

$$\lim_{x \uparrow +\infty} \sigma'(x) = 0, \lim_{x \downarrow -\infty} \sigma'(x) = 0$$



The sigmoid function is the CDF of the logistic distribution

Softmax Function

The softmax function is a vector valued generalization of $\sigma(x)$. Given a vector $\mathbf{x} \in \mathbb{R}^k$, the softmax of $\mathbf{x}$ is

$$S(\mathbf{x}) = \frac{1}{\sum_{i=1}^{k} e^{\mathbf{x}_i}} \begin{bmatrix} e^{\mathbf{x_1}} \\ \vdots \\ e^{\mathbf{x_k}} \end{bmatrix} = \begin{bmatrix} S_1 \\ \vdots \\ S_k \end{bmatrix}$$

Two properties are obvious:

$$\frac{e^{\mathbf{x}_i}}{\sum_j e^{\mathbf{x}_j}} > 0$$

$$\sum_{i=1}^{k} S_i(\mathbf{x}) = 1$$

So the components of S satisfy conditions for a probability distribution

Softmax Function

The sigmoid function resulted naturally from inverting the logit.

$\text{logit}(p_i) = \ln(p_i/(1 - p_i) = w_0 + \mathbf{w}_1 \cdot \mathbf{x}$

Gave:

$$\begin{aligned} p_i &= \text{logit}^{-1}(w_0 + \mathbf{w}_1 \cdot \mathbf{x}) \\ &= \sigma(w_0 + \mathbf{w}_1 \cdot \mathbf{x}) \\ &= \frac{1}{1 + e^{-(w_0 + \mathbf{w}_1 \cdot \mathbf{x})}} \end{aligned}$$

For K classes, assume each class probability can be linearly related to the data. $\ln z$ will serve as a normalizing factor

$$\ln P(y^{(i)} = 1) = \mathbf{w}_1^T \mathbf{x}^{(i)} - \ln z$$

$$\ln P(y^{(i)} = K) = \mathbf{w}_K^T \mathbf{x}^{(i)} - \ln z$$

Softmax Function

Using the constraint:

$$\sum_{k=1}^{K} P(y^{(i)} = k) = 1$$
$$\sum_{k=1}^{K} e^{\mathbf{w}_k^T \mathbf{x}^{(i)} - \ln z} = 1$$

Implies

$$z = \sum_{k=1}^{K} e^{\mathbf{w}_k^T \mathbf{x}^{(i)}}$$

$$P(y^{(j)} = i) = \frac{e^{\mathbf{w}_i^T \mathbf{x}^{(j)}}}{\sum_{k=1}^{K} e^{\mathbf{w}_k^T \mathbf{x}^{(j)}}}$$

Softmax Numerical Stability

Numerical Stability

Double precision floating point numbers use an 11-bit exponent giving a maximum value around $2^{1023} \sim 10^{308}$. It is relatively easy for the exponentials in the softmax to exceed this. R will produce a non-numeric "Inf" result on overflow.

Normalizing Softmax

$$
\begin{aligned}
S_j &= \frac{e^{\mathbf{x}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}_k}} = \frac{C e^{\mathbf{x}_j}}{C \sum_{k=1}^{K} e^{\mathbf{x}_k}} \\
&= \frac{e^{\mathbf{x}_j + \ln C}}{\sum_{k=1}^{K} e^{\mathbf{x}_k + \ln C}} \\
&= \frac{e^{\mathbf{x}_j + b}}{\sum_{k=1}^{K} e^{\mathbf{x}_k + b}}
\end{aligned}
$$

Where $b = \ln C$

Softmax Numerical Stability

Taking

$$b = -\max_k(\mathbf{x}_k)$$

shifts all exponents to be ≤ 0 . Large negative exponents will generate 0, so they don't cause a problem.

Numerically Stable Softmax

$$b = -\max_k(\mathbf{x}_k)$$

$$\tilde{\mathbf{x}} = \mathbf{x} + b$$

$$(S(\mathbf{x}))_i = (S(\tilde{\mathbf{x}}))_i = \frac{e^{\tilde{\mathbf{x}}_i}}{\sum_{k=1}^{K} e^{\tilde{\mathbf{x}}_k}}$$

Encoding

For many modeling algorithms, data must be converted to numeric values

- Images are arrays of pixel values, so already numeric
- Text data must be encoded -In cases where data has a natural order, simple integer encoding can be used
    - sometimes called label encoding

- For text(or other non-numeric data) which are unordered, use one-hot encoding.

text	code
"small"	1
"medium"	2
"large"	3

Label encoding for data with natural order

Encoding

There are 3 common ways to encode text for ML:

One-hot encoding

- sparse encoding of text

TF-IDF: Term frequency-Inverse Document Frequency

- many variants

`

$$\mathrm{tf}(w, d) \cdot \log \frac{N}{|\{d \in D : w \in d\}|}$$

` **Word Embedding**

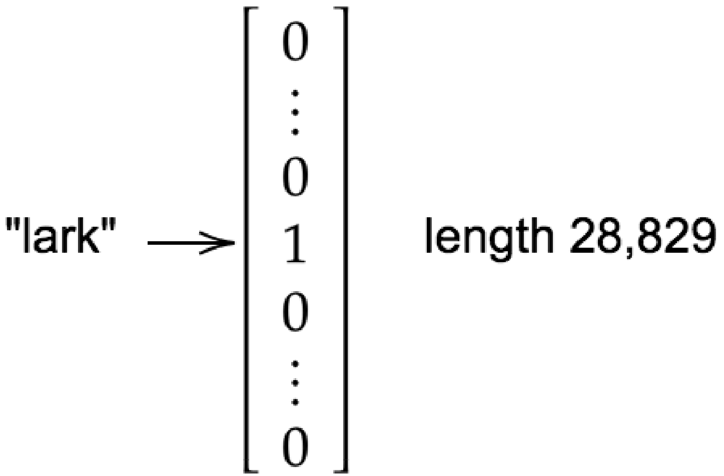
- technique to generate dense encodings from one-hot encoding

One-Hot Encoding

The Shakespeare corpus contains 28,829 unique words, 884,421 total words.

Example:

One-hot encoding of text data from Shakespeare corpus



One-Hot Encoding

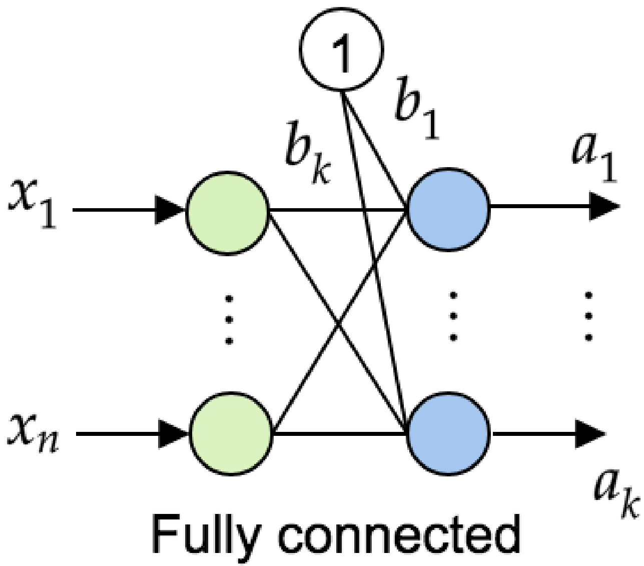
Equivalent to using the dummy variables:

	is.dog	is.cat	is.bird	is.other
"dog"	1	0	0	0
"cat"	0	1	0	0
"bird"	0	0	1	0
"other"	0	0	0	1

Special techniques are needed for NLP when the number of words is large (over 50,000)

- Hierarchical softmax regression
  - Output layer is structured as a binary tree

Softmax Gradient Descent



Softmax regression has K output activations generated by the softmax function. Let X be the full data matrix with m data samples $\mathbf{x}^{(i)}, i = 1, \cdots, m$ arranged by column.

$$X = \begin{bmatrix} \mathbf{x}_1^{(1)} & \cdots & \mathbf{x}_1^{(m)} \\ \vdots & & \vdots \\ \mathbf{x}_n^{(1)} & \cdots & \mathbf{x}_n^{(m)} \end{bmatrix}$$

Softmax Gradient Descent

The weight matrix W maps n-dimensional input vectors to K-dimensional outputs $W : \mathbb{R}^n \to \mathbb{R}^K$

So W is a $K \times n$ matrix. To simplify, let Z be the input to the softmax function:

$$Z = \mathbf{b} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}^T + WX$$

There is a bias for each output node, so $\mathbf{b} \in \mathbb{R}^K$.

$$Z = \mathbf{b}\underbrace{[1, \cdots, 1]}_{m} + \underbrace{W}_{k \text{ x } n} \underbrace{X}_{n \text{ x } m}$$

The result Z is $K \times m$ and the softmax function computes probabilities from each column $\mathbf{z^{(i)}}$ of Z

Softmax Gradient Descent

The predicted responses are $H(X; \mathbf{b}, W) = S(Z)$

Both Z and $H(X)$ are $K \times m$.

$$H(X) = \begin{bmatrix} \vdots & & \vdots \\ \frac{e^{z_i^{(1)}}}{\sum_j e^{z_j^{(1)}}} & \cdots & \frac{e^{z_i^{(m)}}}{\sum_j e^{z_j^{(m)}}} \\ \vdots & & \vdots \end{bmatrix}$$

$$= \begin{bmatrix} \vdots & & \vdots \\ p_i^{(1)} & \cdots & p_i^{(m)} \\ \vdots & & \vdots \end{bmatrix}$$

Softmax Gradient Descent

Let $\mathbf{y} \in \mathbb{R}^m$ where each $y_i \in \{1, 2, \cdots, K\}$ be the training data class labels

One-hot encode $\mathbf{y}$ to get m length K one-hot vectors

$$T = \left.\begin{bmatrix} \vdots & & \vdots \\ t_i^{(1)} & \cdots & t_i^{(m)} \\ \vdots & & \vdots \end{bmatrix}\right\} \quad K \times m$$

Where each $t^{(i)}$ has a single element 1, and all other elements 0.

Softmax Gradient Descent

Softmax regression commonly uses the cross-entropy cost function. For a single data sample:

$$C^{(i)} = -\sum_{k=1}^{K} t_k^{(i)} \ln p_k^{(i)}$$

If sample (i) is class k^*, then $C^{(i)} = -\ln p_{k^*}^{(i)}$

For the full data set:

$$C = \frac{1}{m} \sum_{i=1}^{m} C^{(i)} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} t_k^{(i)} \ln p_k^{(i)}$$

Now need to compute the gradient of C w.r.t the vector $\mathbf{b}$ and the matrix W.

Softmax Gradient Descent

Since

$$\nabla C = \frac{1}{m} \sum_i \nabla C^{(i)}$$

will focus just on one data vector at a time and drop the superscript (i):

$$\frac{\partial C}{\partial z_i} = -\sum_{k=1}^{K} \frac{t_k}{p_k} \frac{\partial p_k}{\partial z_i}$$

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

As homework show that

$$\frac{1}{p_k} \frac{\partial p_k}{\partial z_i} = \delta_{ik} - p_i$$

$$\frac{\partial C}{\partial z_i} = -\sum_{k=1}^{K} t_k(\delta_{ik} - p_i)$$

Softmax Gradient Descent

$$\frac{\partial C}{\partial z_i} = -\sum_{k=1}^{K} t_k(\delta_{ik} - p_i)$$

Every term in this summation is zero unless k is this sample's label class. There are two possibilities. Let k^* be the sample's class. The sum reduces to

$$\frac{\partial C}{\partial z_i} = -t_{k^*}(\delta_{i,k^*} - p_i) = p_i - \delta_{i,k^*}$$

- $i = k^*, (t_i = 1)\,, \frac{\partial C}{\partial z_i} = (p_i - 1) = p_i - t_i$
- $i \neq k^*, (t_i = 0)\,, \frac{\partial C}{\partial z_i} = p_i = p_i - 0 = p_i - t_i$

As before, the Z-gradient is the error:

$$\frac{\partial C}{\partial z} = (\mathbf{p} - \mathbf{t})$$

Just like logistic regression but now the gradient is a vector for each sample.

Softmax Gradient Descent

To implement gradient descent will need gradient w.r.t the adjustable parameters $\mathbf{b}$ and W. For softmax the bias $\mathbf{b}$ is a vector with an element for each of the K class output probabilities. Applying the chain rule gives

$$\begin{aligned}\frac{\partial C}{\partial b_i} &= \sum_j \frac{\partial C}{\partial z_j}\frac{\partial z_j}{\partial b_i} \\ &= \sum_j \frac{\partial C}{\partial z_j}\delta_{ij} \\ &= \frac{\partial C}{\partial z_i}\end{aligned}$$

So, for a single sample vector, the length K $\mathbf{b}$ partial of C is

$$\frac{\partial C}{\partial \mathbf{b}} = \frac{\partial C}{\partial \mathbf{z}}$$

Softmax Gradient Descent

Two steps left:

- Compute $\frac{\partial C}{\partial W}$
- Write equations for full batch X, not just a single sample $\mathbf{x}^{(i)}$

$$\frac{\partial C}{\partial W_{ij}} = \sum_k \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial W_{ij}}$$

$$z_k = b_k + \sum_p W_{kp}\mathbf{x}_p \qquad \text{for one } \mathbf{x}^{(i)}$$

$\frac{\partial z_k}{\partial W_{ij}}$ looks like a 3-dimensional tensor. Fortunately, since z_k only depends on row k of W, all $\frac{\partial z_k}{\partial W_{ij}}$ for $i \neq k$ are zero and can be ignored (they contribute nothing to $\sum$ above.)

$$\frac{\partial z_k}{\partial W_{kj}} = \frac{\partial}{\partial W_{kj}}(b_k + W_{k1}\mathbf{x}_1 + \cdots + W_{kn}\mathbf{x}_n) = \mathbf{x}_j$$

$$\frac{\partial z_k}{\partial W_{i,j}} = \begin{cases} 0 & \text{if } k \neq i \\ x_j & \text{o.w.} \end{cases}$$

Softmax Gradient Descent

So:

$$\begin{aligned}\frac{\partial C}{\partial W_{i,j}} &= \sum_k \frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial W_{i,j}} \\ &= \sum_k \frac{\partial C}{\partial z_k} \delta_{i,k} x_j \\ &= \frac{\partial C}{\partial z_i} \mathbf{x}_j\end{aligned}$$

This can be written in vector notation as

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial \mathbf{z}} \mathbf{x}^T$$

$$= \begin{bmatrix} \frac{\partial C}{\partial z_1} \\ \vdots \\ \frac{\partial C}{\partial z_k} \end{bmatrix} \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}$$

Giving $\frac{\partial C}{\partial W}$ as a $k \times n$ matrix.

Softmax Gradient Descent

For the full data set, each data point $\mathbf{x}^{(i)}$ contributes to the gradient. A good approach is to compute the gradient for all m data points using matrix operations, and then average.

The m $\frac{\partial C}{\partial \mathbf{b}}$ gradients are

$$\begin{bmatrix} \vdots & & \vdots \\ \frac{\partial C(x^{(i)})}{\partial \mathbf{b}} & \cdots & \frac{\partial C(x^{(m)})}{\partial \mathbf{b}} \\ \vdots & & \vdots \end{bmatrix}$$

$$= \frac{\partial C}{\partial Z}$$

$$= S(Z) - T$$

Where $S(Z) = S(\mathbf{b} + WX)$ is the softmax on the entire input data matrix.

Softmax Gradient Descent

For homework, show that sum of the $\frac{\partial C}{\partial W}$ matrices is as shown:

$$
\begin{aligned}
\sum_{i=1}^{m} \frac{\partial C^{(i)}}{\partial W} &= \sum_{i=1}^{m} \frac{\partial C^{(i)}}{\partial \mathbf{z}^{(i)}} (\mathbf{x}^{(i)})^T \\
&= \frac{\partial C}{\partial Z} X^T \\
&= (S(Z) - T) X^T
\end{aligned}
$$

Now compute averages over all samples

$$
\langle \frac{\partial C}{\partial b} \rangle = \frac{1}{m} (S(Z) - T) \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}
$$

$$
\langle \frac{\partial C}{\partial W} \rangle = \frac{1}{m} (S(Z) - T) X^T
$$

Check dimensions: $S(Z) - T$ is $K \times m$, so the row sums of $S(Z) - T$ form a length K vector matching the length of $\mathbf{b}$

Check dimensions: $S(Z) - T$ is $K \times m$, X is $n \times m$ so $(S(Z) - T)X^T$ is $K \times n$ matching the dimension of W

Softmax Regression

Softmax Algorithm

Select a learning rate parameter α
Initialize $\mathbf{b} \in \mathbb{R}^k$ and $W \in \mathbb{R}^{k \times n}$

for $i = 1, \cdots, \text{max\_iterations}$ do
  Compute error $E = S(Z) - T$
  update parameters
    $b = b - \alpha < \frac{\partial C}{\partial \mathbf{b}} >$
    $W = W - \alpha < \frac{\partial C}{\partial W} >$
  save relevant information
end

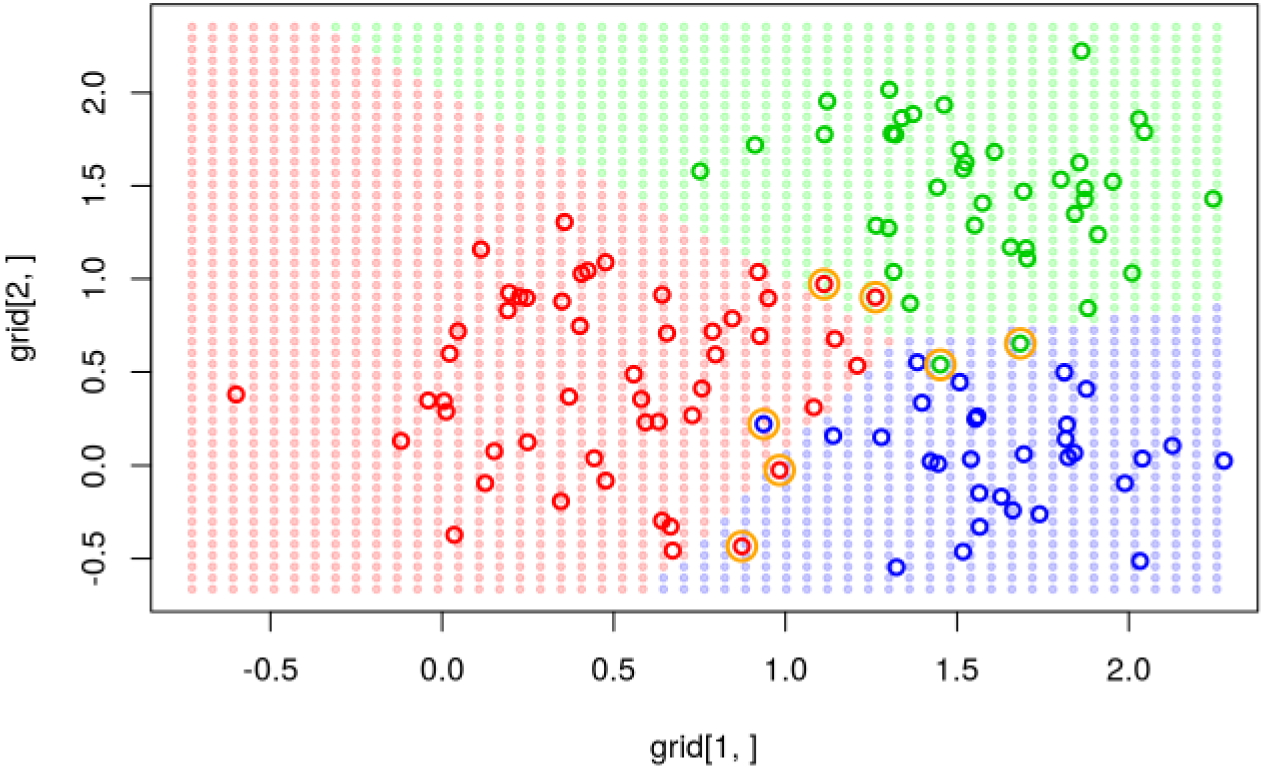
Where $< \cdot >$ means averaged over entire dataset

Binary or Softmax?

Softmax vs. K Binary Classifiers

- If only 1 class possible
  - use Softmax
- If more than one class (characteristic) possible
  - use K binary classifiers and select highest probabilities

Softmax Decision Boundary



MNIST Digits