Softmax is a method to obtain probabilities from outputs. It basically is a generalization of the sigmoid (logistic) loss to more than two classes. We can use it for binary classification as well. The idea of softmax is to take the outputs in the final layer and convert them to probabilities. Suppose the outputs in the final layers are $z_1 = w_1^T x$ and $z_2 = w_2^T x$. Then the softmax conversion is $s_1 = \frac{e^{z_1}}{Z}$, $s_2 = \frac{e^{z_2}}{Z}$ where $Z = e^{z_1} + e^{z_2}$.

If we assume that s2 = 1-s1 then the above is the same as logistic regression.

Homework: can you prove the above statement?

To understand the origins of logistic and softmax see Section 10.7 in Introduction to Machine Learning by Alpaydin Second Edition. We provide a brief recap here from Alpaydin's textbook.

10.7 Logistic Discrimination

10.7.1 Two Classes

LOGISTIC DISCRIMINATION

In *logistic discrimination*, we do not model the class-conditional densities, $p(\boldsymbol{x}|C_i)$, but rather their ratio. Let us again start with two classes and assume that the log likelihood ratio is linear:

$$\log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} = \boldsymbol{w}^T \boldsymbol{x} + w_0^o \tag{10.18}$$

This indeed holds when the class-conditional densities are normal (equation 10.13). But logistic discrimination has a wider scope of applicability; for example, $\boldsymbol{x}$ may be composed of discrete attributes or may be a mixture of continuous and discrete attributes.

Using Bayes' rule, we have

$$\begin{aligned} \text{logit}(P(C_1|\boldsymbol{x})) &= \log \frac{P(C_1|\boldsymbol{x})}{1 - P(C_1|\boldsymbol{x})} \\ &= \log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} + \log \frac{P(C_1)}{P(C_2)} \\ &= \boldsymbol{w}^T \boldsymbol{x} + w_0 \end{aligned} \tag{10.19}$$

where

$$w_0 = w_0^o + \log \frac{P(C_1)}{P(C_2)} \tag{10.20}$$

Rearranging terms, we get the sigmoid function again:

$$y = \hat{P}(C_1|\boldsymbol{x}) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)]} \tag{10.21}$$

as our estimator of $P(C_1|\boldsymbol{x})$.

Let us see how we can learn $\boldsymbol{w}$ and w_0. We are given a sample of two classes, $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}$, where $r^t = 1$ if $\boldsymbol{x} \in C_1$ and $r^t = 0$ if $\boldsymbol{x} \in C_2$. We assume r^t, given $\boldsymbol{x}^t$, is Bernoulli with probability $y^t \equiv P(C_1|\boldsymbol{x}^t)$ as calculated in equation 10.21:

$$r^t|\boldsymbol{x}^t \sim \text{Bernoulli}(y^t)$$

Here, we see the difference from the likelihood-based methods where we modeled $p(\boldsymbol{x}|C_i)$; in the discriminant-based approach, we model directly $r|\boldsymbol{x}$. The sample likelihood is

$$l(\boldsymbol{w}, w_0|\mathcal{X}) = \prod_t (y^t)^{(r^t)}(1 - y^t)^{(1-r^t)} \tag{10.22}$$

We know that when we have a likelihood function to maximize, we can always turn it into an error function to be minimized as $E = -\log l$, and in our case, we have *cross-entropy*:

CROSS-ENTROPY

$$E(\boldsymbol{w}, w_0|\mathcal{X}) = -\sum_t r^t \log y^t + (1 - r^t)\log(1 - y^t) \tag{10.23}$$

Because of the nonlinearity of the sigmoid function, we cannot solve directly and we use gradient descent to minimize cross-entropy, equivalent to maximizing the likelihood or the log likelihood. If $y = \text{sigmoid}(a) = 1/(1 + \exp(-a))$, its derivative is given as

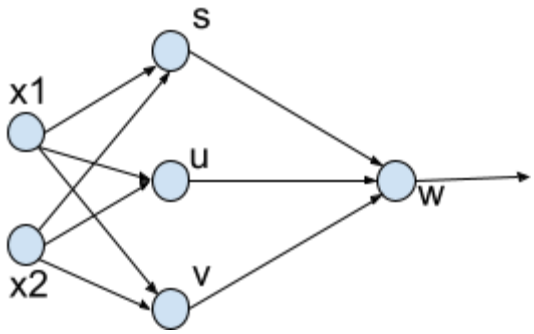
$$\frac{dy}{da} = y(1 - y)$$

and we get the following update equations:

$$\begin{aligned} \Delta w_j &= -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t \left(\frac{r^t}{y^t} - \frac{1 - r^t}{1 - y^t} \right) y^t(1 - y^t)x_j^t \\ &= \eta \sum_t (r^t - y^t)x_j^t, j = 1, \ldots, d \\ \Delta w_0 &= -\eta \frac{\partial E}{\partial w_0} = \eta \sum_t (r^t - y^t) \end{aligned} \tag{10.24}$$

For multiple classes the logistic loss becomes softmax.

Now that we have reviewed logistic loss let us write out the objective of a single layer neural network shown below.



The least squares loss is given by $f = ((w_1, w_2, w_3)^T(\sigma(s^T x), \sigma(u^T x), \sigma(v^T x)) - y)^2$ where **σ**(x) is an activation function such as sigmoid or relu. In this document we let **σ**(x) be the sigmoid activation: $\sigma(x) = 1/(1 + e^{-x})$.

Instead of minimizing the difference between true and predicted values we can try to maximize the probability of the output. First we have to define this probability. Fortunately the logistic loss gives us a straightforward way to convert outputs to probabilities. Thus we write the loss as $f = p = 1/(1 + e^{-z})$ where $z = (w_1, w_2, w_3)^T(\sigma(s^T x), \sigma(u^T x), \sigma(v^T x))$. Now the loss is a probability and so we want to maximize this if the label y = 1 and maximize $f = 1 - p = 1 - (1/(1 + e^{-z}))$ if the label y = 0. We can write the objective as one function:

$$f' = p^y(1 - p)^{(1-y)}$$

The empirical loss can be written as the likelihood which is a product of the above probabilities as shown in equation 10.22 above.

$$emploss = \Pi_i p^{y_i}(1 - p)^{(1-y_i)}$$

By definition the loss function is to be minimized and so we take the negative log to convert this to a machine learning loss function. We can now write the loss as

$f = -log(f') = -ylog(p) - (1 - y)log(1 - p) = -ylog((1/(1 + e^{-z}))) - (1 - y)log((1 - 1/(1 + e^{-$
(see equations 10.22 and 10.23).

Let's rewrite this succinctly as

$f = -\ ylog(s)\ -\ (1\ -\ y)log((1\ -\ s))$where $s\ =\ 1/(1\ +\ e^{-z})$.

We now proceed to derive the update equations for the final layer and intermediate layer weights.

$df/dw_1\ =\ df/ds\ ds/dz\ dz/dw_1$ where $df/ds\ =-\ y/s\ +\ (1\ -\ y)(1/(1\ -\ s))$, $ds/dz\ =\ s(1\ -\ s)$, and $dz/dw_1\ =\sigma(s^T x)$

Similarly we get the weight updates for w_2, w_3.

Now we do the inner layer weights:

$df/ds_1\ =\ df/ds\ ds/dz\ dz/ds_1$(can also be written as $df/ds_1\ =\ df/ds\ ds/dz\ dz/d\sigma\ \ d\sigma/ds_1$)where $dz/ds_1\ =\ w_1\sigma(s^T x)(1\ -\ \sigma(s^T x))x_1$and the other derivatives are the same as above.

Similarly we derive weights for s_2, u_1, u_2, v_1, v_2.