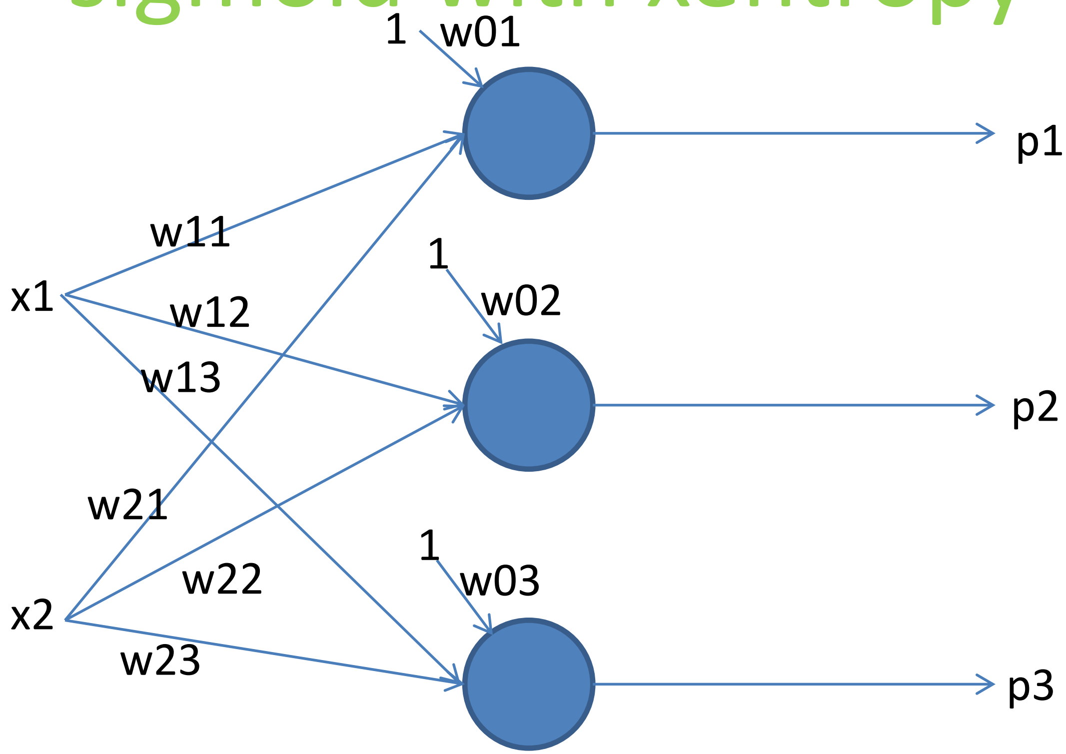
Perceptron Layer

Issues of Perceptron-III

- Can only perform binary linear classification: The objective based perceptrons does find better linear decision boundary than traditional perceptron but still can only be useful for binary classification task.

  Solution: Use single layer of perceptrons for multi-class classification task with either sigmoid perceptrons and cross entropy objective. Encode the output of train data using one-hot encoding technique.

Single layer perceptron network: sigmoid with xentropy



- k sigmoid perceptrons are used for k-class classification problem and cross entropy objective is used for learning

cross-entropy objective function

$$E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k} \; y_k^{(n)} \log o_k + (1 - y_k)\log(1 - o_k)$$

Intuitive meaning

- Since the output is always between 0 to 1, the above error is always positive
- The cost tends to be close to zero if the output is approaching with actual value

Single layer perceptron learning with sigmoid AF + Xentropy

- Assume some random weights and random bias for perceptron

- Repeat the following until the error is below threshold or maximum number of epochs reached:

  Shuffle the train data and repeat the following until the end of epoch

  a. Pick a training smaple($x^{(n)}$, $y^{(n)}$) and compute the output of each perceptron, $o^{(n)}$

  b. Update each weight, w_{ij}, of the perceptron as follows:

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

$$w_{ij} = w_{ij} + \eta \left(y_j^{(n)} - o_j^{(n)}\right) x_i^{(n)}$$

Note: keep $x_i^{(n)} = 1$ for w_{oj} (bias input)

Error derivatives of weights

Compute error gradient on single example :

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial \sum_k w_{kj} x_k}{\partial w_{ij}} = x_i$$

$$\frac{\partial o_j}{\partial a_j} = g'(a_j) = g(a_j)\ (1 - g(a_j))$$

$$= o_j\,(1 - o_j))$$

Error derivatives of weights

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \sum_k y_k \log o_k + (1 - y_k) \log(1 - o_k)$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} y_j \log o_j + (1 - y_j) \log(1 - o_j)$$

$$\frac{\partial E}{\partial o_j} = -\left(\frac{y_j}{o_j}\right) - \left(\frac{1-y_j}{1-o_j}\right)$$

$$\frac{\partial E}{\partial o_j} = \left(\frac{o_j - y_j}{o_j(1-o_j)}\right)$$

Error derivatives of weights

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

$$= \left(\frac{o_j - y_j}{o_j(1 - o_j)}\right) o_j(1 - o_j)\, x_i$$

$$= \left(o_j - y_j\right) x_i$$

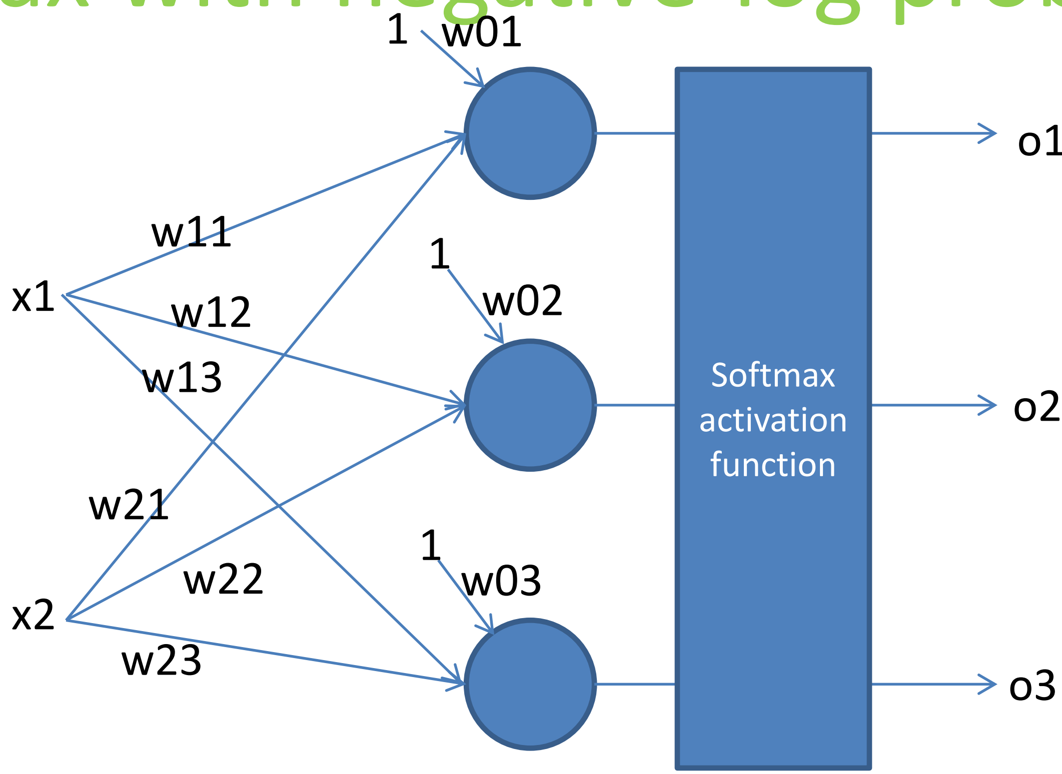
Whats wrong with sigmoid?

- Sigmoids with xentropy provides the faster learning

- The outputs of sigmoid perceptrons provides the confidence of being respective class but the output probabilities may not sum to 1. Hence, interpretation is not straight.

- Soft max activation instead of sigmoid activation provides the probability distribution which is more interpretable.

Single layer perceptron network: softmax with negative-log probability



- k perceptrons are required for k-class classification problem and softmax activation is used across perceptrons

Softmax function: sigmoid for multi-classes

- Softmax output of perceptron i, $o_j = g(a_j) = \frac{e^{a_j}}{\sum_k e^{a_k}}$

- The properties of softmax function:
  - $0 < o_j < 1$
  - $\sum_k o_k = 1$

- Individual sigmoid units also provides value between 0 and 1 but the summation of output sigmoid units may not equal to 1.

- The learning algorithm is same as that of sigmoidAF because the partial derivative of softmax is: $g(a_j)\,(1 - g(a_j))$

Single layer perceptron learning with softmax AF + Xentropy objective

- Assume some random weights and random bias for perceptron

- Repeat the following until the error is below threshold or maximum number of epochs reached:

  Shuffle the train data and repeat the following until the end of epoch

  a. Pick a training smaple($x^{(n)}$, $y^{(n)}$) and compute the output of each perceptron, $o^{(n)}$

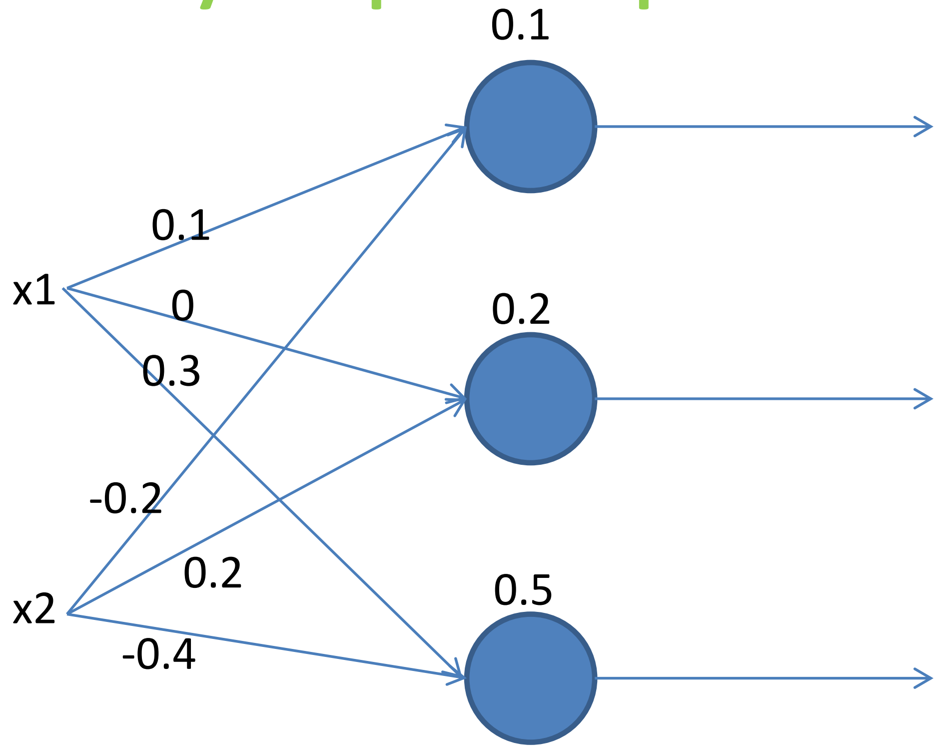
  b. Update each weight, w_{ij}, of the perceptron as follows:

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

$$w_{ij} = w_{ij} + \eta \left(y_j^{(n)} - o_j^{(n)}\right) x_i^{(n)}$$

Note: keep $x_i^{(n)} = 1$ *for* w_{oj} *(bias input)*

Single layer perceptron network



x1	x2	Output
1	2	c1
-1	2	c2
0	-1	c3