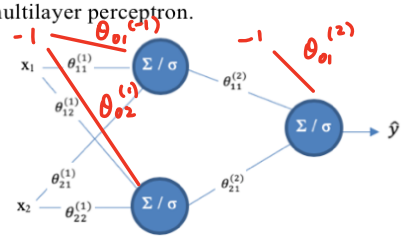
1. Consider the following multilayer perceptron.



The network should implement the XOR function. Perform one epoch of *backpropagation* as introduced in the lecture on multilayer perceptrons.

Notes:

- The activation function f for a perceptron is the *sigmoid function*:

$$f(x) = \frac{1}{1 + e^{-x}}$$

bias bias

- The thresholds are not shown in the network. The threshold nodes are set to -1.
- Use the following initial parameter values:

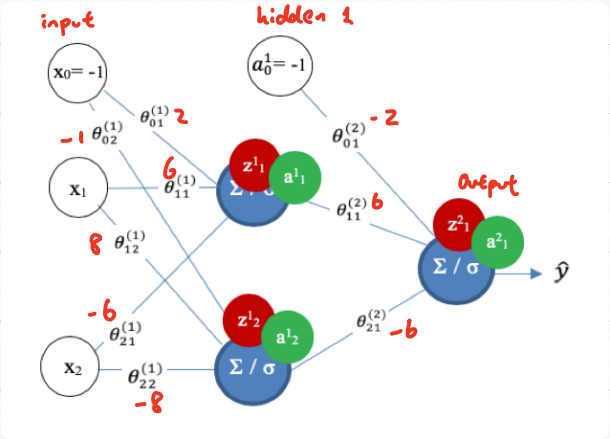
$$\theta_{01}^{(1)} = 2 \qquad \theta_{02}^{(1)} = -1 \qquad \theta_{01}^{(2)} = -2$$
$$\theta_{11}^{(1)} = 6 \qquad \theta_{12}^{(1)} = 8 \qquad \theta_{11}^{(2)} = 6$$
$$\theta_{21}^{(1)} = -6 \qquad \theta_{22}^{(1)} = -8 \qquad \theta_{21}^{(2)} = -6$$

- The learning rate is set to $\eta = 0.7$

i. Compute the activations of the hidden and output neurons. $a_1^{(1)}, a_2^{(1)}$ (level 1), $a_1^{(2)}$ (level 2)

ii. Compute the error of the network.

iii. Backpropagate the error to determine $\Delta\theta_{ij}$ for all weights θ_{ij} and updates the weight θ_{ij}.



(i) For level 1 :

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \sigma(2(-1) + 6x_1 - 6x_2) = \sigma(-2 + 6x_1 - 6x_2)$$

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \sigma(-1(-1) + 8x_1 - 8x_2) = \sigma(1 + 8x_1 - 8x_2)$$

For level 2 :

(Output $\hat{y}$ =) $a_1^{(2)} = \sigma(z_1^{(2)}) = \sigma(-2(-1) + 6a_1^{(1)} - 6a_2^{(1)}) = \sigma(2 + 6a_1^{(1)} - 6a_2^{(1)})$

(ii) Error:

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - a_i^{(2)})^2$$

(iii) Backpropagation (efficient way of computing partial derivatives of the error of MLP wrt. each weight $\theta_{jk}^{(l)}$. $\frac{\partial E}{\partial \theta_{jk}^{(l)}}$

1. Foward pass 2. Compute error 3. Backward pass (propagate error terms)

4. Update all θ (at once) $\Delta\theta_i$

GD: $\theta_i \leftarrow \theta_i - \eta \frac{\partial E}{\partial \theta_i}$ ← By back propagation

In MLP:

$$\theta_{jk}^{(l)} \leftarrow \theta_{jk}^{(l)} + \Delta\theta_{jk}^{(l)}$$

$$\Delta\theta_{jk}^{(l)} = -\eta \frac{\partial E}{\partial \theta_{jk}^{(l)}} = \eta\, \delta_k^{(l)} a_j^{(l-1)}$$ ← prev layer

error term: $\delta_k^{(l)} = (y - a_k^{(l)})\, \sigma'(z_k^{(l)})$ (lecture)

(last layer) $= (y - a_k^{(l)})\, \sigma(z_k^{(l)})(1 - \sigma(z_k^{(l)}))$

Chain Rule:

$$E = \frac{1}{2}(y - a_k^{(l)})^2$$

$$\frac{\partial E}{\partial \theta_{jk}^{(l)}} = \frac{\partial E}{\partial a_k^{(l)}} \cdot \frac{\partial a_k^{(l)}}{\partial z_k^{(l)}} \cdot \frac{\partial z_k^{(l)}}{\partial \theta_{jk}^{(l)}}$$

$$= -(y - a_k^{(l)}) \cdot \sigma'(z_k^{(l)}) \cdot a_j^{(l-1)}$$

$$= -\delta_k^{(l)} a_j^{(l-1)}$$

prev layer: $(l-1)$

$$\text{If } \frac{\partial E}{\partial \theta_{jk}^{(l-1)}} = \underbrace{\frac{\partial E}{\partial a_k^{(l)}} \cdot \frac{\partial a_k^{(l)}}{\partial z_k^{(l)}}}_{\delta_k^{(l)}} \cdot \underbrace{\frac{\partial z_k^{(l)}}{\partial a_k^{(l-1)}}}_{\theta_{jk}^{(l)}} \cdot \underbrace{\frac{\partial a_k^{(l-1)}}{\partial z_k^{(l-1)}}}_{\sigma'(z_k^{(l-1)})} \cdot \underbrace{\frac{\partial z_k^{(l-1)}}{\partial \theta_{jk}^{(l-1)}}}_{a_{jk}^{(l-2)}}$$

$$\text{new } \delta_k^{(l-1)} \text{ at prev layer}$$

For training instance $\langle 1, 0 \rangle$: $y = 1$

$$a_1^{(1)} = \sigma(-2 + 6x_1 - 6x_2) = \sigma(4) = 0.982$$

$$a_2^{(1)} = \sigma(1 + 8x_1 - 8x_2) = \sigma(9) = 0.999$$

Output: $a_1^{(2)} = \sigma(2 + 6a_1^{(1)} - 6a_2^{(1)}) = \sigma(2 + 6 \times 0.982 - 6 \times 0.999) = \sigma(1.898) = 0.8691$

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1 - 0.8691) = 0.0086$$

Back propagation: (right to left)

$$\delta_1^{(2)} = (y - a_1^{(2)})\, \sigma(z_1^{(2)})\, (1 - \sigma(z_1^{(2)}))$$

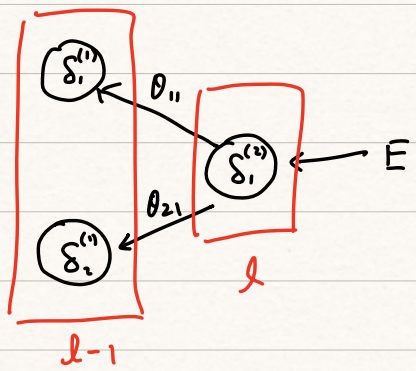
$$= (1 - 0.8691)\, \sigma(1.898)\, [1 - \sigma(1.898)]$$

$$= 0.0149$$

$$\delta_1^{(1)} = \theta_{11}^{(2)}\, \delta_1^{(2)}\, \sigma(z_1^{(1)})\, [1 - \sigma(z_1^{(1)})]$$

$$= 6 \times 0.0149 \times \sigma(4) \times [1 - \sigma(4)]$$

$$= 0.0016$$



$$\delta_2^{(1)} = \theta_{12}^{(2)}\, \delta_1^{(2)}\, \sigma(z_2^{(1)})\, [1 - \sigma(z_2^{(1)})]$$

$$= -6 \times 0.0149 \times \sigma(9) \times [1 - \sigma(9)]$$

$$= -0.00000103$$

$\eta = 0.7$, calculate $\Delta\theta$: (at once)

bias

$$\Delta\theta_{01}^{(2)} = \eta\, \delta_1^{(2)}\, a_0^{(1)} = 0.7 \times 0.0149 \times (-1) = -0.0104$$

$$\Delta\theta_{11}^{(2)} = \eta\, \delta_1^{(2)}\, a_1^{(1)} = 0.7 \times 0.0149 \times 0.982 = 0.0102$$

...

...

Update θ:

$$\theta_{01}^{(2)} \leftarrow \theta_{01}^{(2)} + \Delta\theta_{01}^{(2)}$$

$$= -2 + (-0.0104) = -2.0104$$

$$\theta_{11}^{(2)} \leftarrow \theta_{11}^{(2)} + \Delta\theta_{11}^{(2)}$$

$$= 6 + 0.0102 = 6.0102$$

...

This is only one training instance ⇒ do this for all training instances to finish one epoch.

2. How is Logistic Regression similar to Naive Bayes and how is it different? In what circumstances would the former be preferable, and in what circumstances would the latter?

Similar:

1. classification methods ⇒ predict class Y for instance X.
2. Compute $P(Y|X)$ ⇒ predict class with highese prob.
   ⇒ probabilistic ML methods (prediction based on probs.)
3. Have params → maximise likelihood

Different:

NB	LR
Generative model	Discriminative model
⇒ maximise joint $P(x,y)$	⇒ model $P(y\|x)$ directly
assume conditional independence (among features)	No independence assumption
Trying to learn what dogs & cats look like.	Trying to learn to "distinguish" between classes without actually learning much about them

E.g. If features x_1, x_2 are highly correlated (OR $x_1 = x_2$)

NB: Multiply them both in ($P(x_1|y) P(x_2|y) \cdots$)

LR: Assign part of the weight to x_1 and part to x_2.

=> LR might get a more accurate probability & work better on larger datasets.

Prefer:

Generally, LR outperform NB (NB can outperform LR when little data available)

NB: simpler, easier to implement, param estimated in closed-form.

LR: adopt iterative optimisation method => time-consuming for large data.