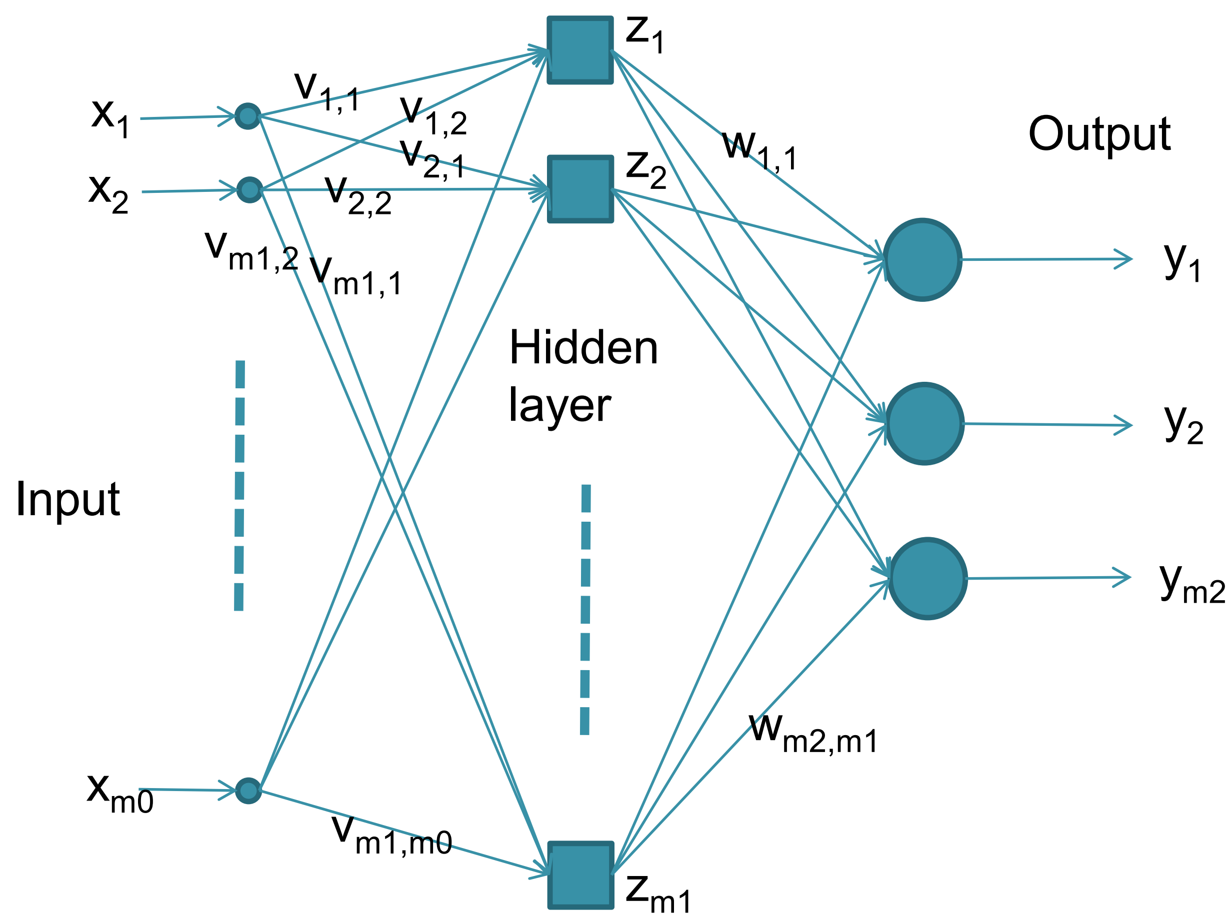
Multilayer perceptron (MLP)



- Each hidden node is a perceptron

$$z_j = \varphi\left(\sum_{i=1}^{m0} v_{j,i} x_i + b_j\right)$$

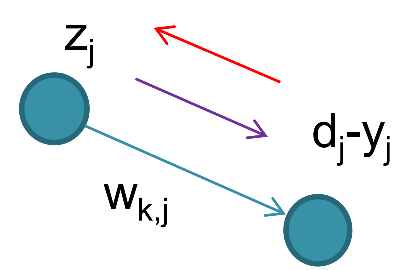
- Each output node is a perceptron

$$y_k = \varphi\left(\sum_{j=1}^{m1} w_{k,j} z_j + b_k\right)$$

Backpropagation (BP) algorithm

- In the SLP, we used gradient descent on the error function to perform error-correction learning via the update of the weights:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n) = \mathbf{w}(n) + \eta e(n)\mathbf{z}(n)$$
$$= \mathbf{w}(n) + \eta\big(d(n) - y(n)\big)\mathbf{z}(n)$$



or

$$w_{k,j}(n+1) = w_{k,j}(n) + \eta\big(d_k(n) - y_k(n)\big)z_j(n)$$

- Remark: errors/updates are local to the node, i.e., the change in the weight from node j to output k is controlled by the input that travels along the connection and the error signal from output k.

New issue due to hidden layer

- Since there is no direct error signal for the input layer in MLP, how are the weights from the input-to-hidden layers adjusted when the error is computed for output layer only?

gradient descent - chain rule - multi-layer/node

- **Credit assignment problem**

* asssigning "credit" or "blame' to individual elements involved in forming overall response of a learning systems (hidden nodes).
* deciding which weights should be altered, how much and in which direction, in MLP.
* specify how much a weight in the input layer contributes to the output and thus the error.

Solution to credit assignment

- Solution for MLP is provided by Rumelhart, Hinton, and Williams (1986) (actually invented earlier in a PhD thesis reltaing to economics)

prediction

- **Forward pass phase :** computes 'functional signal', feedforward propagation of input pattern signals through network.

- **Backward pass phase :** computes 'error signal', *propagates* the error *backwards* through network starting at output nodes (where the error is the difference between actual and desired output values).

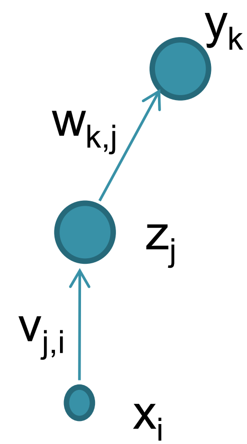
Note: a very different concept from 'feedback'

3-layer MLP Backpropagation

- **Forward Pass**

- Output of hidden node at time t

$$z_j(t) = \varphi\left(\sum_{i=0}^{m0} v_{j,i}(t)\, x_i(t)\right) = \varphi\left(u_j(t)\right)$$



- Output of output node at time t

$$y_k(t) = \varphi\left(\sum_{i=0}^{m1} w_{k,j}(t)\, z_j(t)\right) = \varphi\left(a_k(t)\right)$$

- **Backward Pass**

sum of squares error critrion for each training pattern is:

$$\varepsilon(t) = \frac{1}{2}\sum_{k=1}^{m2}\left(d_k(t) - y_k(t)\right)^2$$

where d_k is the target value for class k.

3-layer MLP Backpropagation

- Weight update using gradient descent

$$w_{k,j}(t+1) - w_{k,j}(t) \propto -\frac{\partial \varepsilon(t)}{\partial w_{k,j}(t)}$$

proportional to

- The partial derivative can be rewritten as product of two terms using chain rule for partial defferentiation:

$$\frac{\partial \varepsilon(t)}{\partial w_{k,j}(t)} = \frac{\partial \varepsilon(t)}{\partial a_k(t)} \cdot \frac{\partial a_k(t)}{\partial w_{k,j}(t)}$$

where the 1st term shows how error for pattern changes as function of change in network input to node k, and 2nd term shows how net input to node k changes as a function of change in weight $w_{k,j}$.

3-layer MLP Backpropagation

- Partial derivatives (2nd term)

$$\frac{\partial a_k(t)}{\partial w_{k,j}(t)} = z_j(t), \quad \frac{\partial u_j(t)}{\partial v_{j,i}(t)} = x_i(t)$$

input to node

- Partial derivatives (1st term) (error terms)

$$\Delta_k(t) = -\frac{\partial \varepsilon(t)}{\partial a_k(t)}, \quad \delta_j(t) = -\frac{\partial \varepsilon(t)}{\partial u_j(t)}$$

gradient

- For output nodes, we have:

$$\Delta_k(t) = -\frac{\partial \varepsilon(t)}{\partial a_k(t)} = -\frac{\partial \varepsilon(t)}{\partial y_k(t)} \cdot \frac{\partial y_k(t)}{\partial a_k(t)} = -\frac{\partial \varepsilon(t)}{\partial y_k(t)} \cdot \varphi'(a_k(t))$$

$$= -\frac{\partial \left[\frac{1}{2} \sum_{k=1}^{m2} \left(d_k(t) - y_k(t) \right)^2 \right]}{\partial y_k(t)} \cdot \varphi'(a_k(t)) = \varphi'(a_k(t)) \left(d_k(t) - y_k(t) \right)$$

single 'k'

3-layer MLP Backpropagation

- For hidden nodes, we would need to use the chain rule:

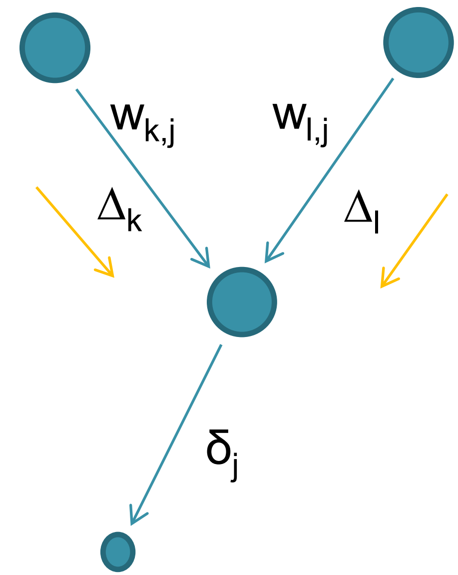
sum 'k'

$$\delta_j(t) = -\frac{\partial \varepsilon(t)}{\partial u_j(t)} = -\sum_{k=1}^{m2} \frac{\partial \varepsilon(t)}{\partial a_k(t)} \cdot \frac{\partial a_k(t)}{\partial u_j(t)}$$

$$= \sum_{k=1}^{m2} \Delta_k(t) \cdot \frac{\partial a_k(t)}{\partial u_j(t)}$$

$$= \sum_{k=1}^{m2} \Delta_k(t) \cdot \frac{\partial a_k(t)}{\partial z_j(t)} \cdot \frac{\partial z_j(t)}{\partial u_j(t)}$$

$$= \varphi'\left(u_j(t)\right) \sum_{k=1}^{m2} w_{k,j} \cdot \Delta_k(t)$$



weights here can be viewed as providing degree of 'credit' or 'blame' to hidden nodes.

3-layer MLP Backpropagation

- Combining the two terms, we have

$$-\frac{\partial \varepsilon(t)}{\partial w_{k,j}(t)} = -\frac{\partial \varepsilon(t)}{\partial a_k(t)} \cdot \frac{\partial a_k(t)}{\partial w_{k,j}(t)} = \Delta_k(t) z_j(t)$$

$$-\frac{\partial \varepsilon(t)}{\partial v_{j,i}(t)} = -\frac{\partial \varepsilon(t)}{\partial u_j(t)} \cdot \frac{\partial u_j(t)}{\partial v_{j,i}(t)} = \delta_j(t) x_i(t) = \varphi'\left(u_j(t)\right) \sum_{k=1}^{m2} w_{k,j} \Delta_k(t) x_i(t)$$

- To achieve gradient descent in ε, weights are updated by:

$$w_{k,j}(t+1) = w_{k,j}(t) + \eta \Delta_k(t) z_j(t)$$

$$v_{j,i}(t+1) = v_{j,i}(t) + \eta \delta_j(t) x_i(t)$$

where η is the learning rate parameter ($0 < \eta \leq 1$).

3-layer MLP Backpropagation

- In BP learning algorithm, weight updates are "local":

For output nodes

$$w_{k,j}(t+1) = w_{k,j}(t) + \eta \Delta_k(t) z_j(t)$$
$$= w_{k,j}(t) + \eta \varphi'(a_k(t))(d_k(t) - y_k(t)) z_j(t)$$

For hidden nodes

$$v_{j,i}(t+1) = v_{j,i}(t) + \eta \delta_j(t) x_i(t)$$
$$= v_{j,i}(t) + \eta \varphi'(u_j(t)) \sum_{k=1}^{m2} w_{k,j} \Delta_k(t) x_i(t)$$