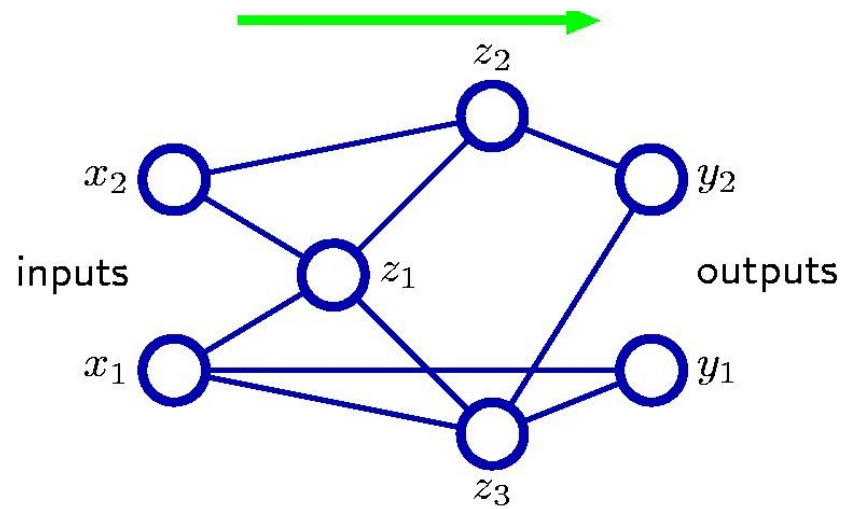


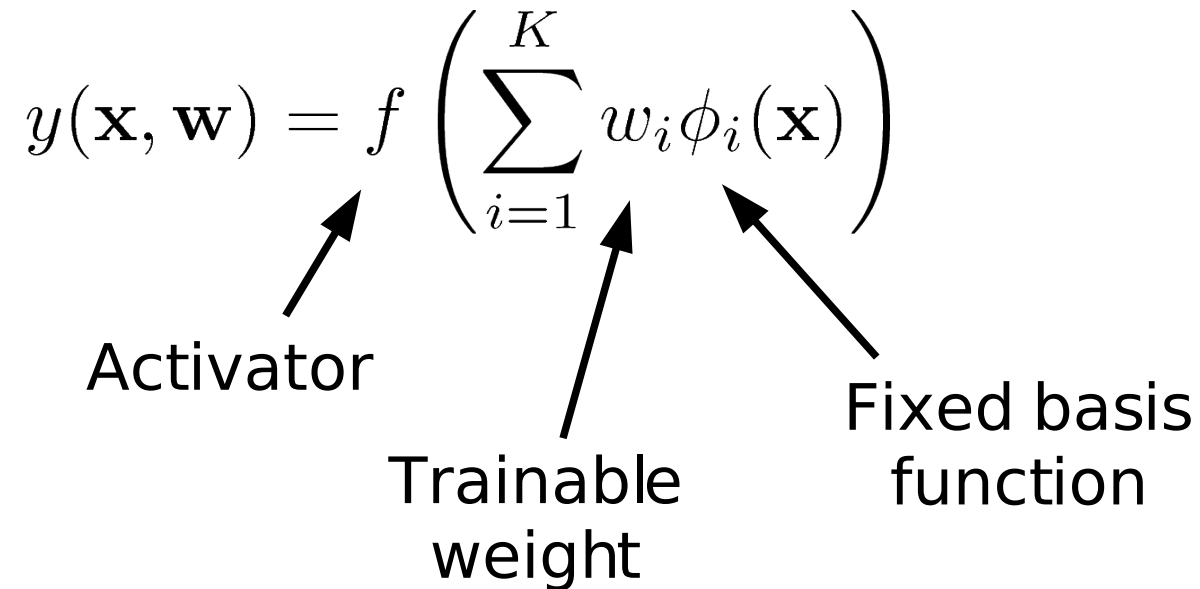
# Neural Networks



Machine Learning; Tue May 1, 2007

# Motivation

**Problem:** With our linear methods, we can train the weights but not the basis functions:

$$y(\mathbf{x}, \mathbf{w}) = f \left( \sum_{i=1}^K w_i \phi_i(\mathbf{x}) \right)$$


Activator

Trainable weight

Fixed basis function

# Motivation

**Problem:** With our linear methods, we can train the weights but not the basis functions:

$$y(\mathbf{x}, \mathbf{w}) = f \left( \sum_{i=1}^K w_i \phi_i(\mathbf{x}) \right)$$

Can we **learn** the basis functions?

# Motivation

**Problem:** With our linear methods, we can train the weights but not the basis functions:

$$y(\mathbf{x}, \mathbf{w}, \{\mathbf{w}^i\}) = f \left( \sum_{i=1}^K w_i \phi_i(\mathbf{x}, \mathbf{w}^i) \right)$$

Can we **learn** the basis functions?

Maybe reuse ideas from our previous technology? Adjustable weight vectors?

# Historical notes

This motivation is not how the technology came about!

Neural networks were developed in an attempt to achieve AI by modeling the brain (but ***artificial*** neural networks have very little to do with biological neural networks).

# Feed-forward neural network

Two levels of “linear regression” (or classification):

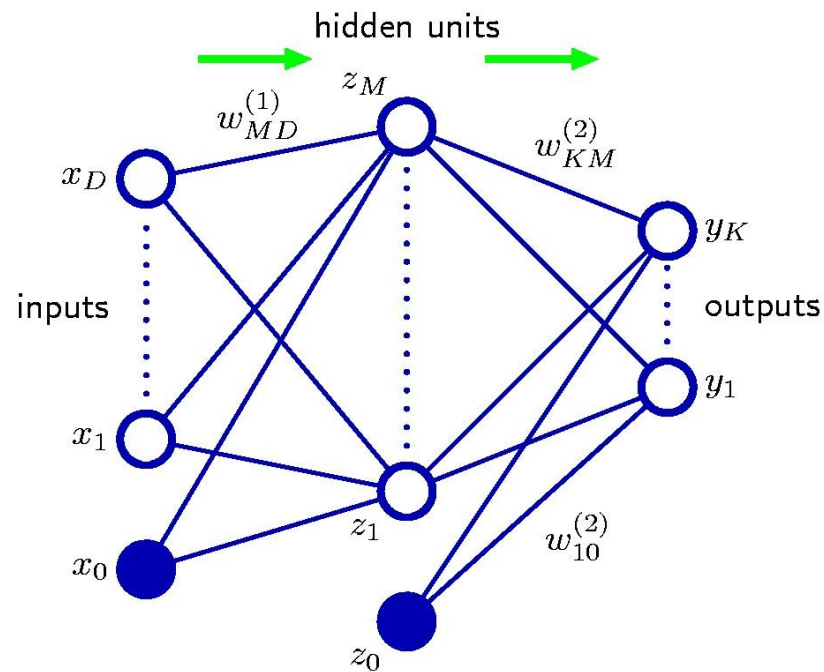
$$y(\mathbf{x}, \mathbf{w}^{(1)}, \mathbf{w}^{(2)}) = f \left( \sum_{i=1}^K w_i^{(2)} z_i \right)$$

$$z_i = h \left( \sum_{j=1}^M w_{ij}^{(1)} x_j \right)$$

Output layer

Hidden layer

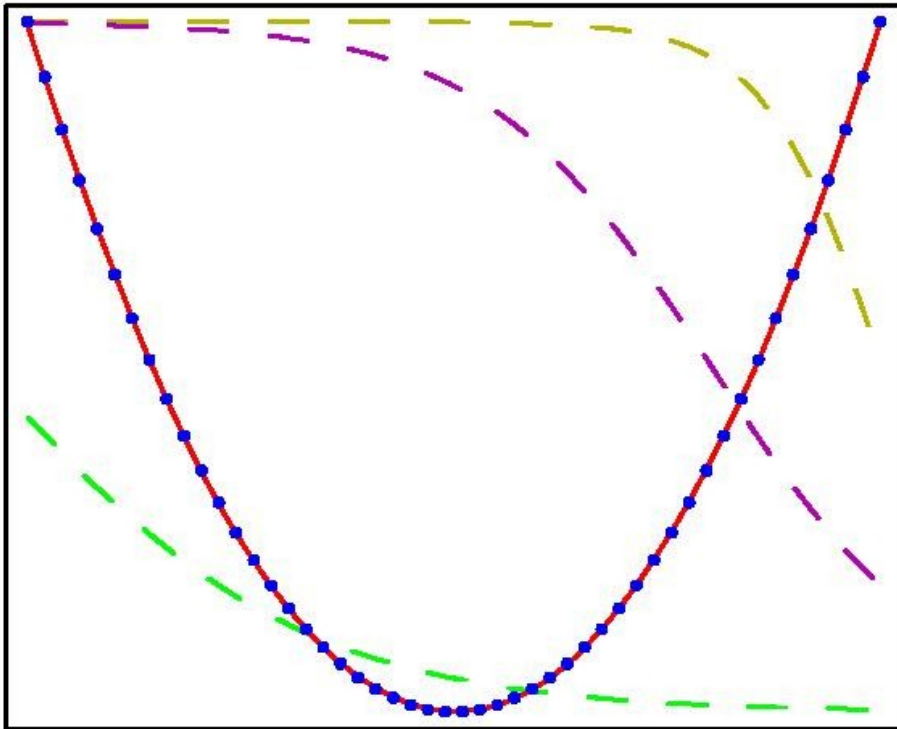
# Feed-forward neural network



$$y_K(\mathbf{x}, \mathbf{w}) = f \left( \sum_{j=0}^M w_{kj}^{(2)} \cdot h \left( \sum_{i=0}^D w_{ji}^{(1)} \cdot x_i \right) \right)$$

# Example

$$y(x) = x^2$$

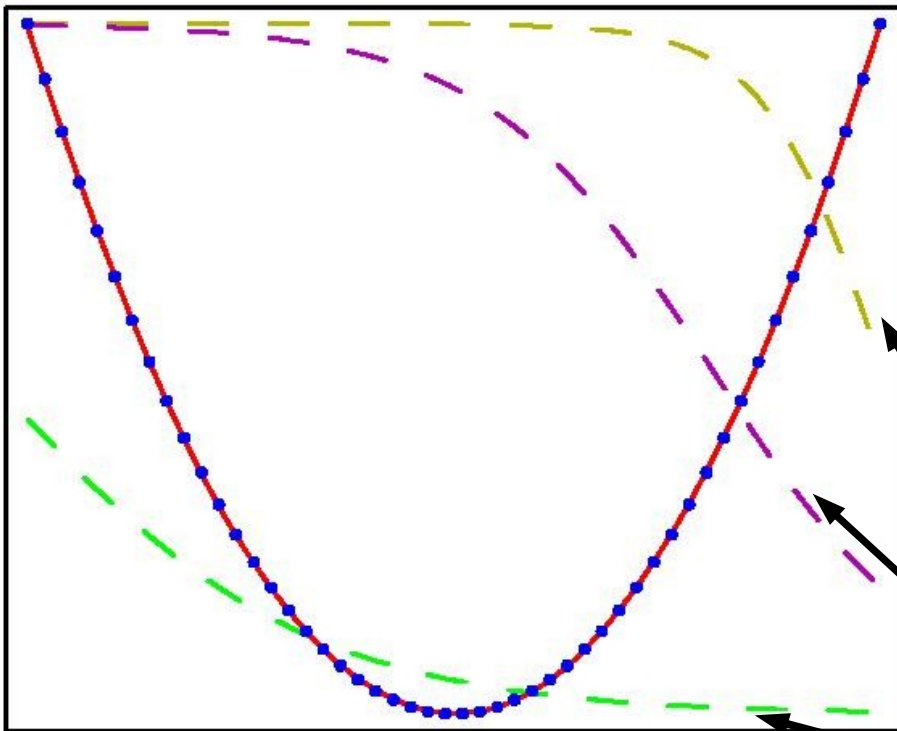


tanh activation function for  
3 hidden layers

Linear activation function  
for output layer

# Example

$$y(x) = x^2$$



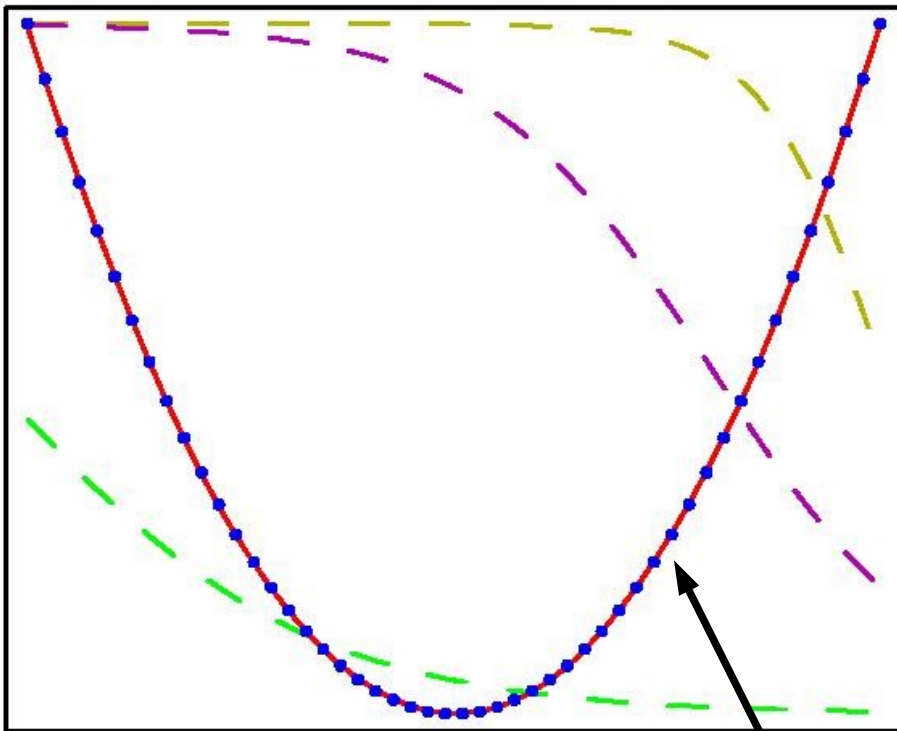
tanh activation function for  
3 hidden layers

Linear activation function  
for output layer

$$z_i = \tanh \left( w_{1i}^{(1)} x + w_{0i}^{(1)} \right)$$

# Example

$$y(x) = x^2$$



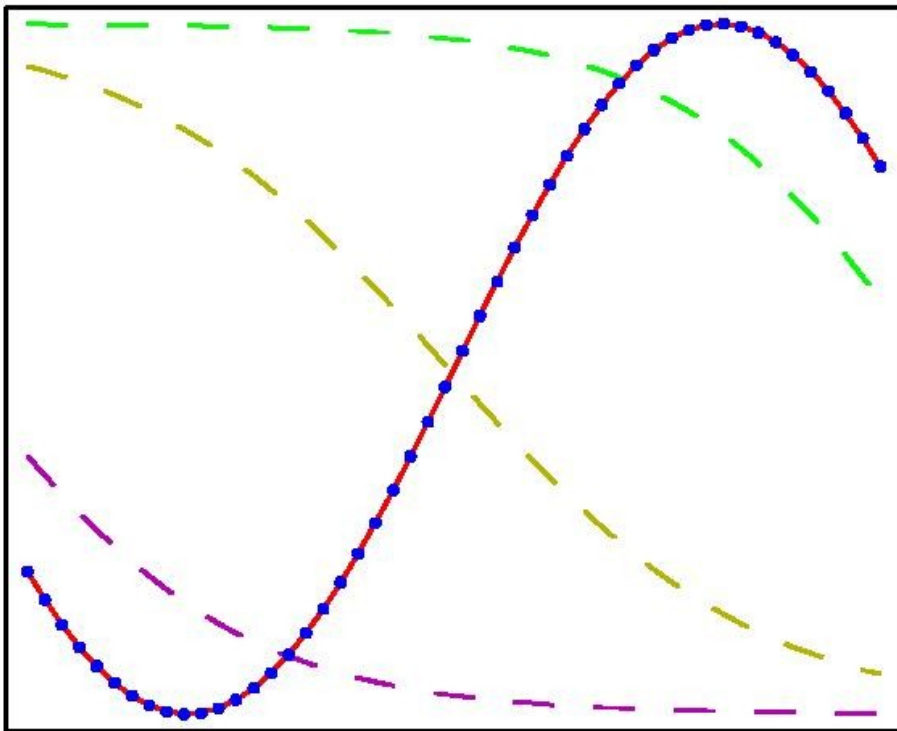
tanh activation function for  
3 hidden layers

Linear activation function  
for output layer

$$y = w_0^{(2)} + w_1^{(2)} \cdot z_1 + w_2^{(2)} \cdot z_2 + w_3^{(2)} \cdot z_3$$

# Example

$$y(x) = \sin(x)$$

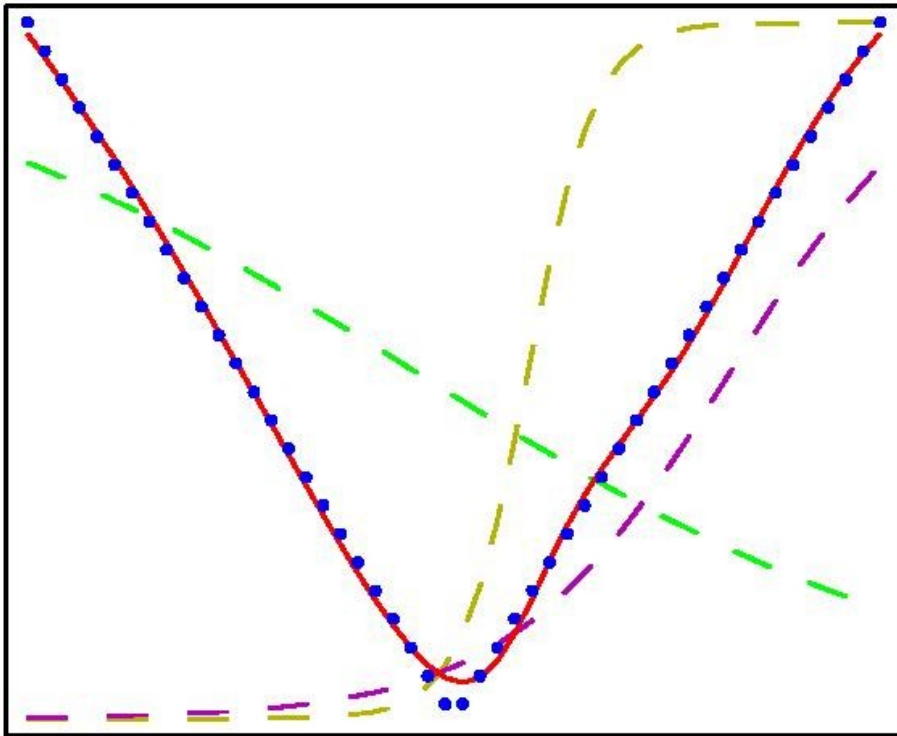


tanh activation function for  
3 hidden layers

Linear activation function  
for output layer

# Example

$$y(x) = |x|$$



tanh activation function for  
3 hidden layers

Linear activation function  
for output layer

# Network training

If we can give the network a probabilistic interpretation, we can use our “standard” estimation methods for training.

# Network training

If we can give the network a probabilistic interpretation, we can use our “standard” estimation methods for training.

Regression problems:

$$p(t \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(t \mid y(\mathbf{x}, \mathbf{w}), \sigma^2)$$

$$\hat{\mathbf{w}}_{\text{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}} p(t \mid \mathbf{x}, \mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

# Network training

If we can give the network a probabilistic interpretation, we can use our “standard” estimation methods for training.

Classification problems:

$$p(t \mid \mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$$

$$y = \sigma(a)$$

$$\hat{\mathbf{w}}_{\text{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}} p(t \mid \mathbf{x}, \mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

# Network training

If we can give the network a probabilistic interpretation, we can use our “standard” estimation methods for training.

Training by minimizing an ***error function***.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

# Network training

If we can give the network a probabilistic interpretation, we can use our “standard” estimation methods for training.

Training by minimizing an ***error function***.

Typically, we cannot minimize analytically but need numerical optimization algorithms.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

$$E(\mathbf{w}) = - \sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

# Network training

If we can give the network a probabilistic interpretation, we can use our “standard” estimation methods for training.

Training by minimizing an ***error function***.

Typically, we cannot minimize analytically but need numerical optimization algorithms.

**Notice:** There will typically be more than one minimum of the error function (due to symmetries).

**Notice:** At best, we can find local minima.

# Parameter optimization

Numerical optimization is beyond the scope of this class – you can (should) get away with just using appropriate libraries such as GSL.

$$\hat{\mathbf{w}} = \text{minimize}(E, \mathbf{w}_0)$$

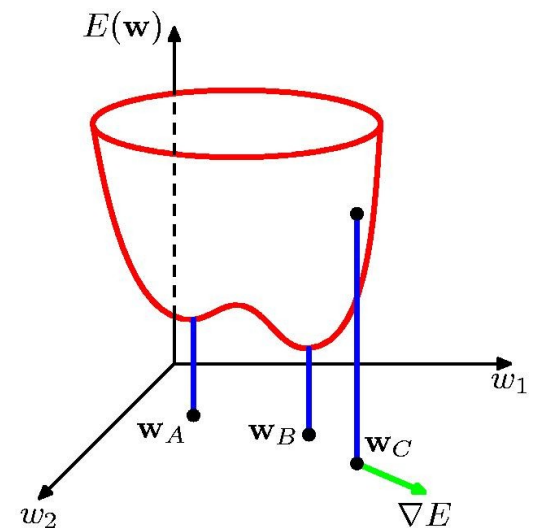
# Parameter optimization

Numerical optimization is beyond the scope of this class – you can (should) get away with just using appropriate libraries such as GSL.

These can typically find (local) minima of general functions, but the **most efficient** algorithms also need the **gradient** of the function to minimize.

$$\hat{\mathbf{w}} = \text{minimize}(E, \mathbf{w}_0)$$

$$\hat{\mathbf{w}} = \text{minimize}(E, \nabla E, \mathbf{w}_0)$$



# Back propagation

***Back propagation*** is an efficient algorithm for computing both the error function and the gradient.

# Back propagation

**Back propagation** is an efficient algorithm for computing both the error function and the gradient.

For iid data:

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

$$\nabla E(\mathbf{w}) = \sum_{k=1}^K \frac{\partial E}{\partial w_k} = \sum_{k=1}^K \left( \sum_{n=1}^N \frac{\partial E_n}{\partial w_k} \right)$$

# Back propagation

**Back propagation** is an efficient algorithm for computing both the error function and the gradient.

For iid data:

$$E(\mathbf{w}) = \sum_{n=1}^N E_n(\mathbf{w})$$

$$\nabla E(\mathbf{w}) = \sum_{k=1}^K \frac{\partial E}{\partial w_k} = \sum_{k=1}^K \left( \sum_{n=1}^N \frac{\partial E_n}{\partial w_k} \right)$$

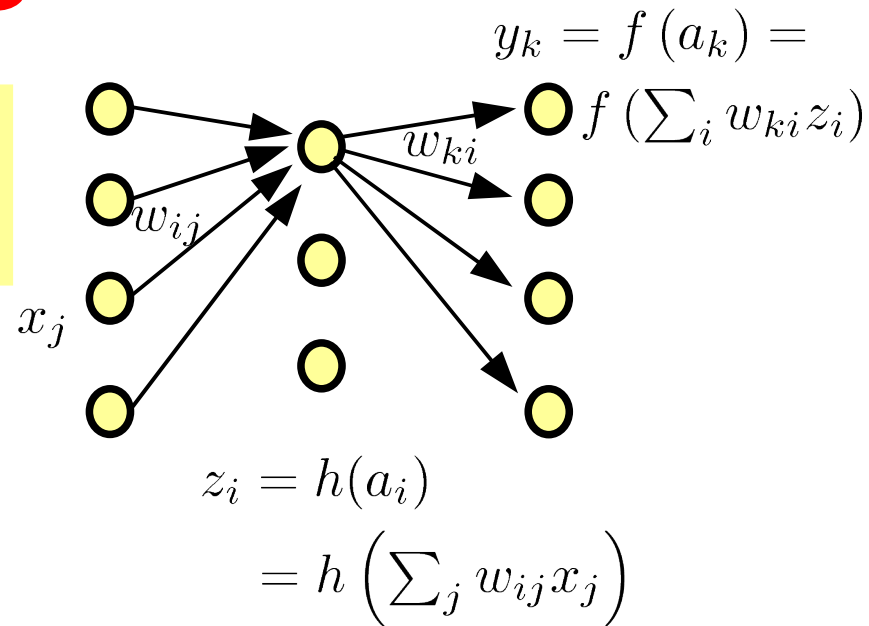
↖  
We just focus  
on this guy

# Back propagation

After running the feed-forward algorithm, we know the values  $z_i$  and  $y_k$ .

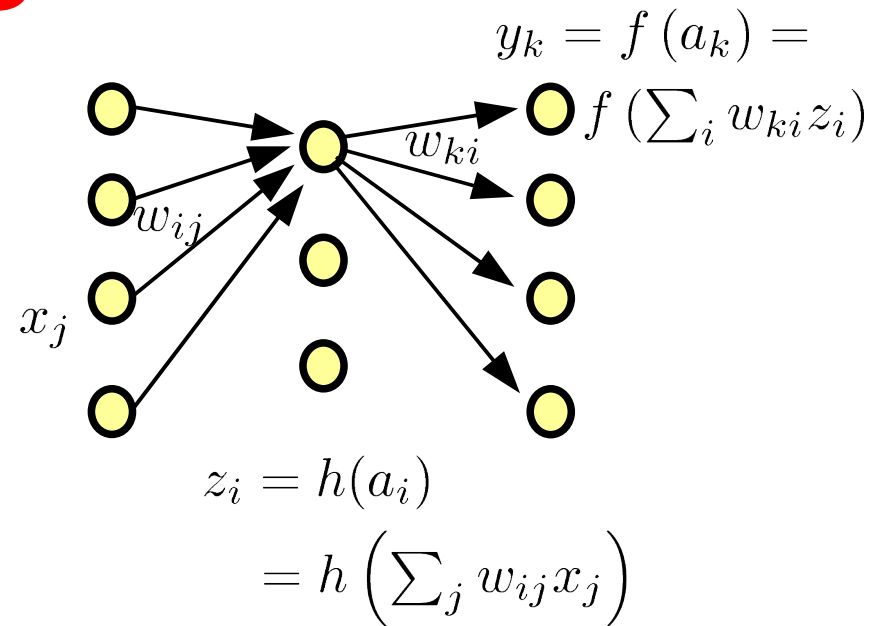
$$\begin{aligned}\frac{\partial E_n}{\partial w_{ki}} &= \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{ki}} \\ &= \delta_k z_i\end{aligned}$$

we need to compute the  $\delta$ s (called *errors*).



# Back propagation

The choice of error function and output activator typically makes it simple to deal with the output layer



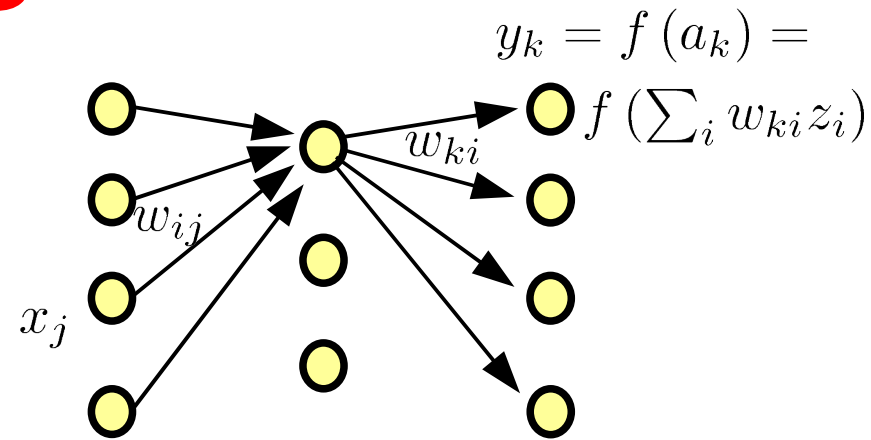
$$y_k = f(a_k) = a_k$$

$$E_n(\mathbf{w}) = \frac{1}{2} \{y_n - t_n\}^2$$

$$\delta_k = \frac{\partial E_n}{\partial a_k} = (y_{kn} - t_n) \cdot 1$$

# Back propagation

For hidden layers, we apply the chain rule

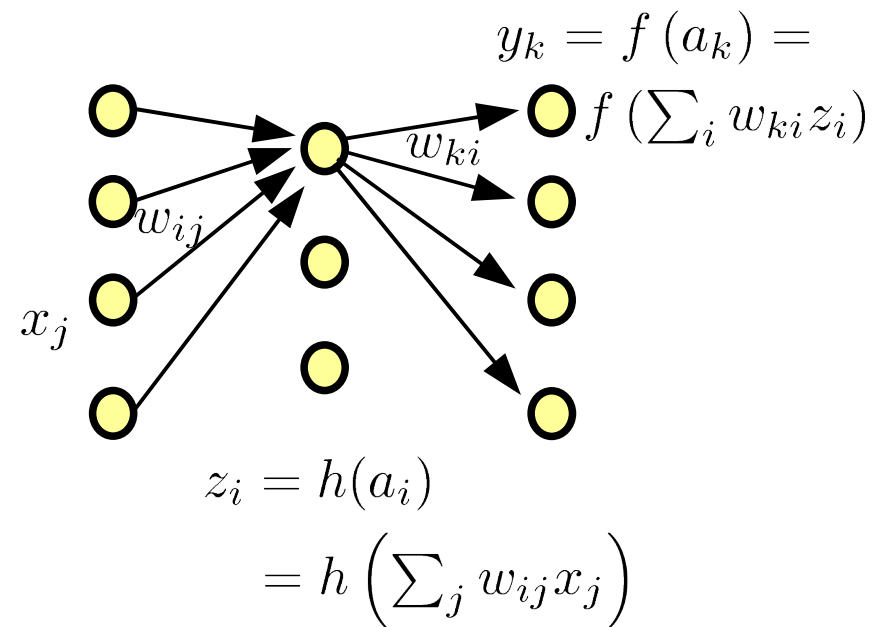


$$\begin{aligned}\delta_i &= \frac{\partial E_n}{\partial a_i} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_i} \\ &= \sum_k \delta_k \frac{\partial a_k}{\partial a_i} = \sum_k \delta_k \frac{\partial a_k}{\partial z_i} \frac{\partial z_i}{\partial a_i} \\ &= \sum_k \delta_k w_{ki} \frac{\partial z_i}{\partial a_i} = h'(a_i) \sum_k w_{ki} \delta_k\end{aligned}$$

# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.
2. Calculate output errors and propagate errors back to get the gradient.
3. Iterate ad lib.



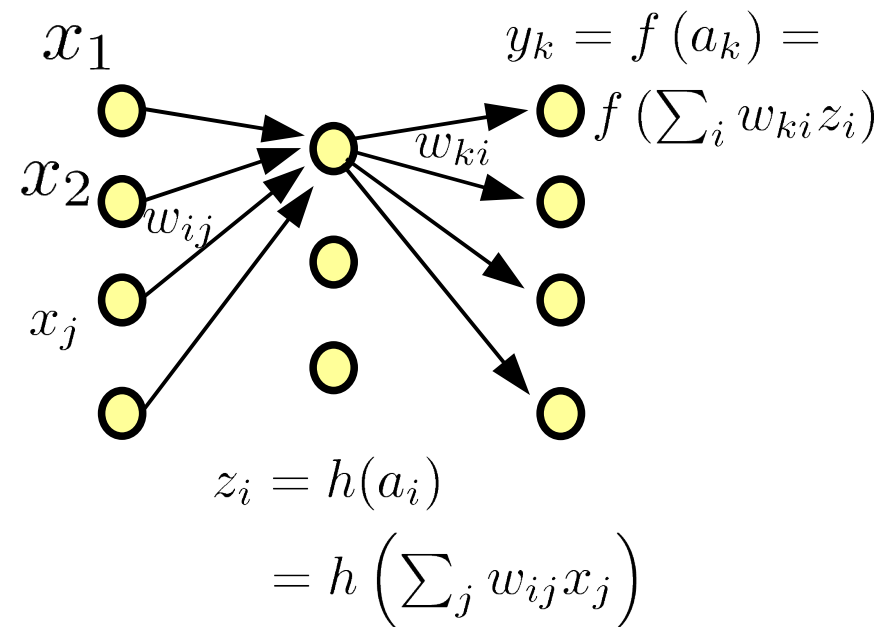
# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.

2. Calculate output errors and propagate errors back to get the gradient.

3. Iterate ad lib.



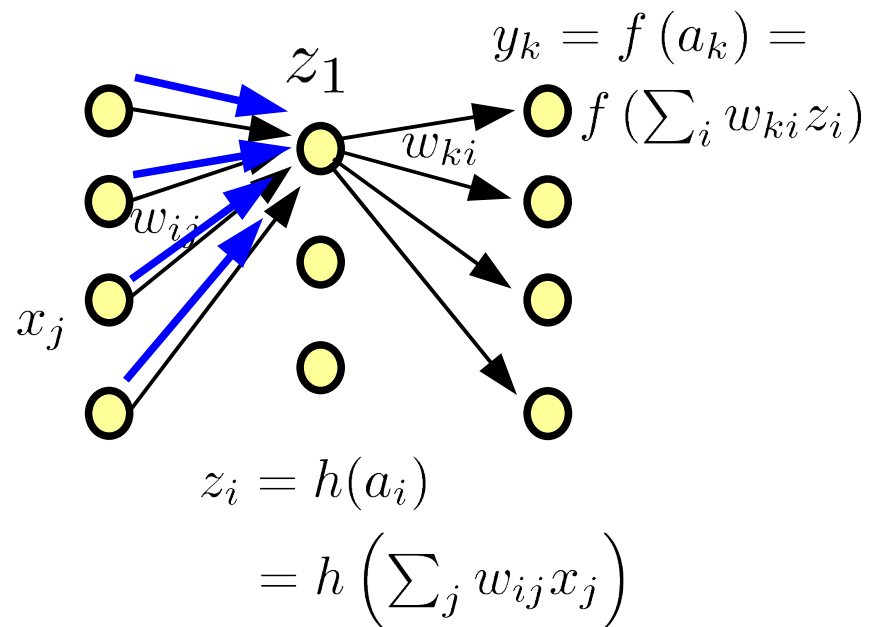
# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.

2. Calculate output errors and propagate errors back to get the gradient.

3. Iterate ad lib.



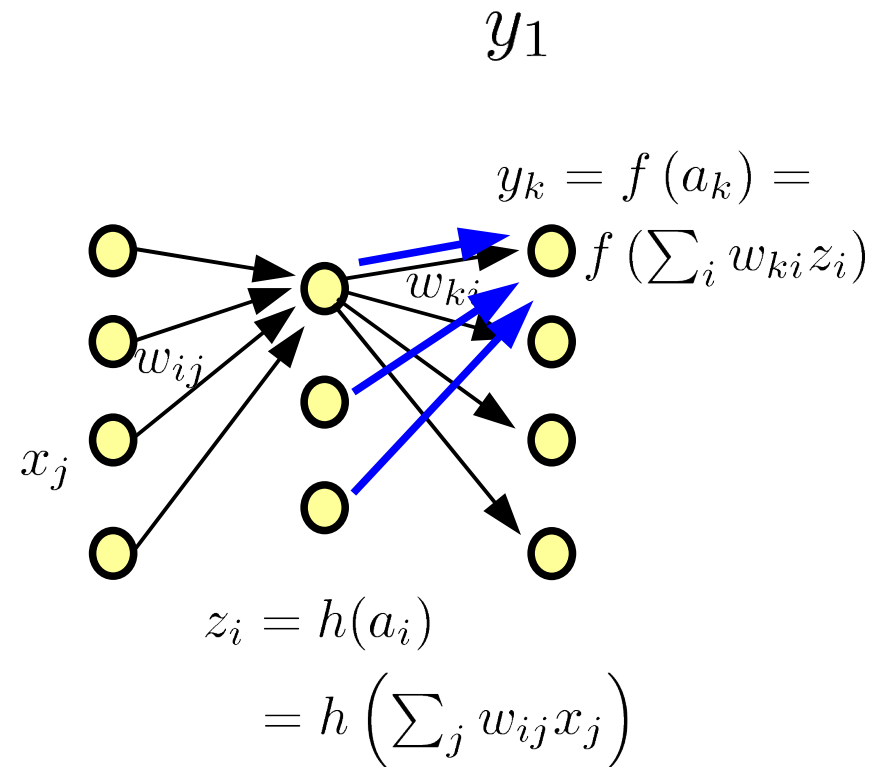
# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.

2. Calculate output errors and propagate errors back to get the gradient.

3. Iterate ad lib.



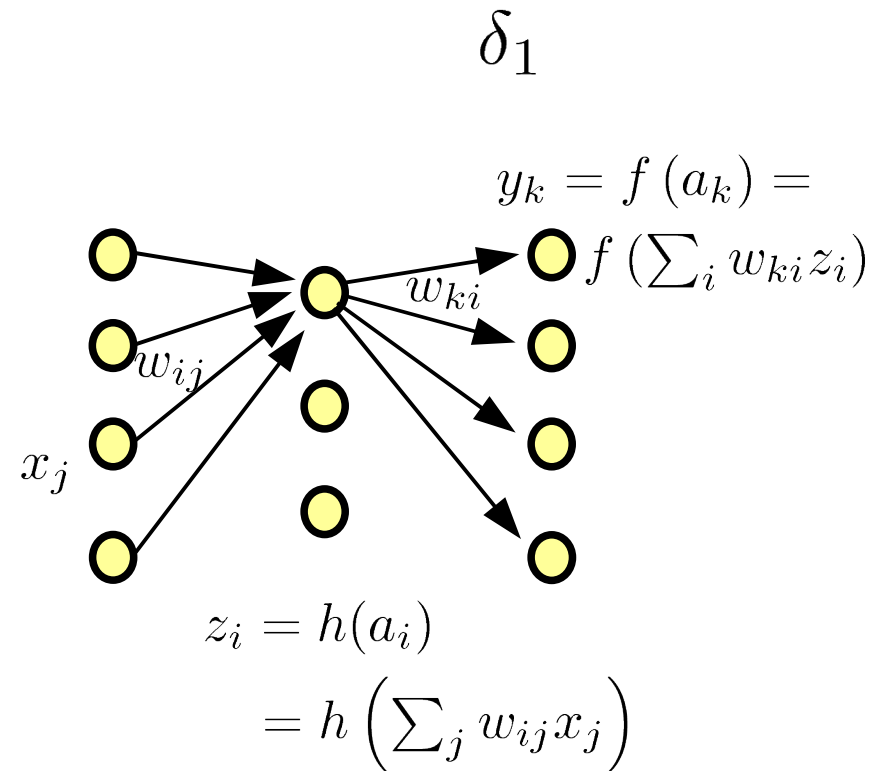
# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.

**2. Calculate output errors and propagate errors back to get the gradient.**

3. Iterate ad lib.



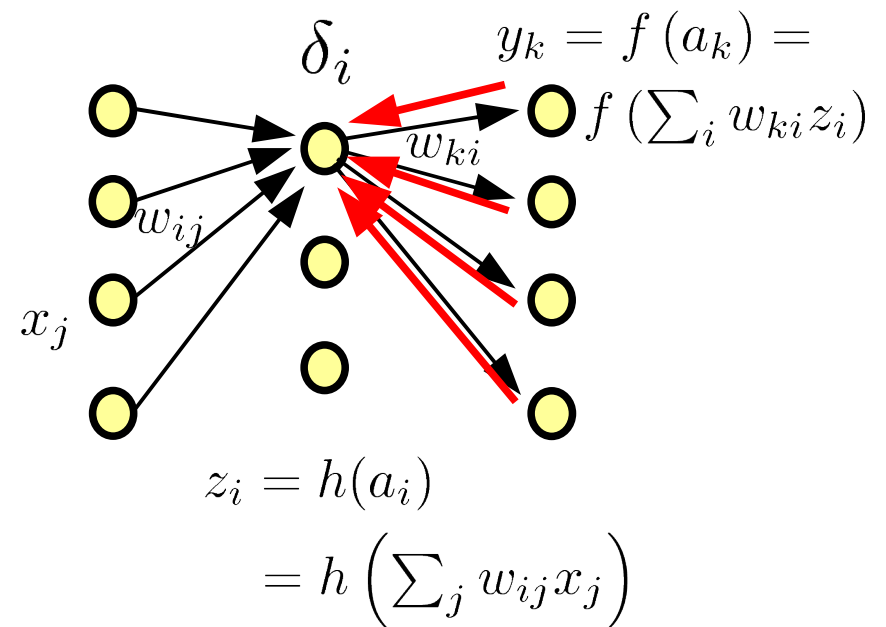
# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.

**2. Calculate output errors and propagate errors back to get the gradient.**

3. Iterate ad lib.



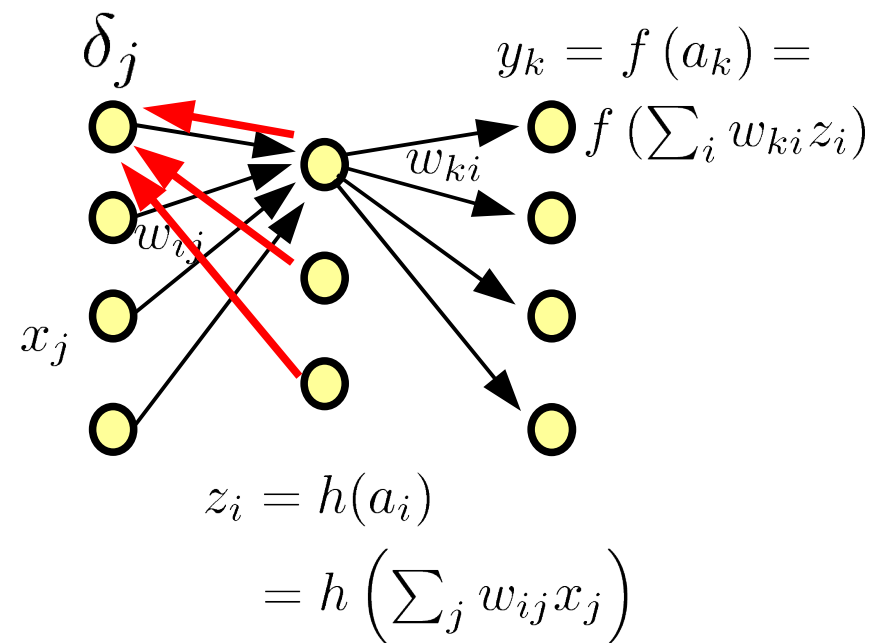
# Calculating error and gradient

## Algorithm

1. Apply feed forward to calculate hidden and output variables and then the error.

**2. Calculate output errors and propagate errors back to get the gradient.**

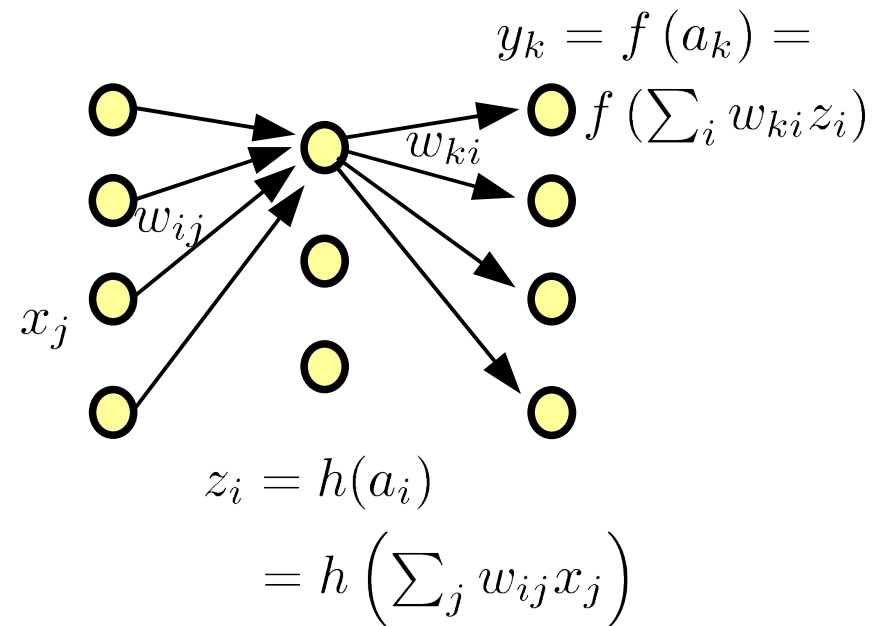
3. Iterate ad lib.



# Calculating error and gradient

## Algorithm

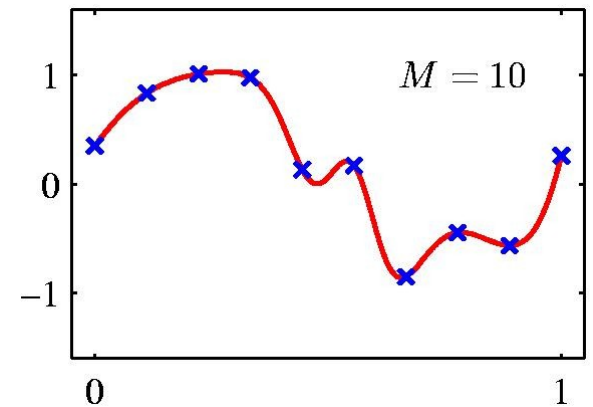
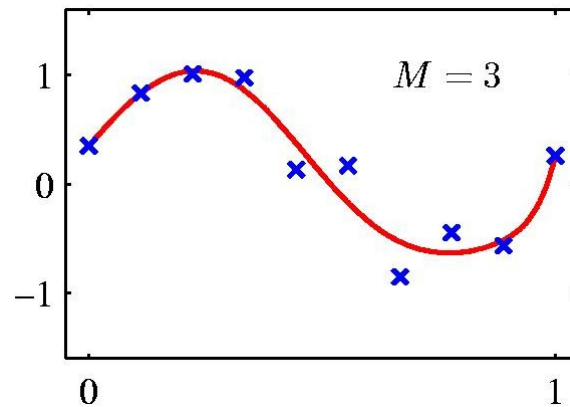
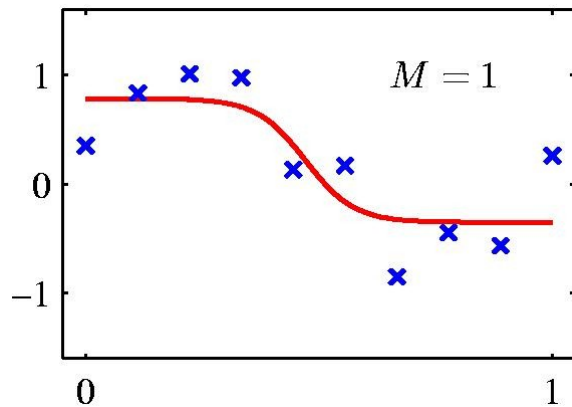
1. Apply feed forward to calculate hidden and output variables and then the error.
2. Calculate output errors and propagate errors back to get the gradient.
3. Iterate ad lib.



# Overfitting

The complexity of the network is determined by the hidden layer – and overfitting is still a problem.

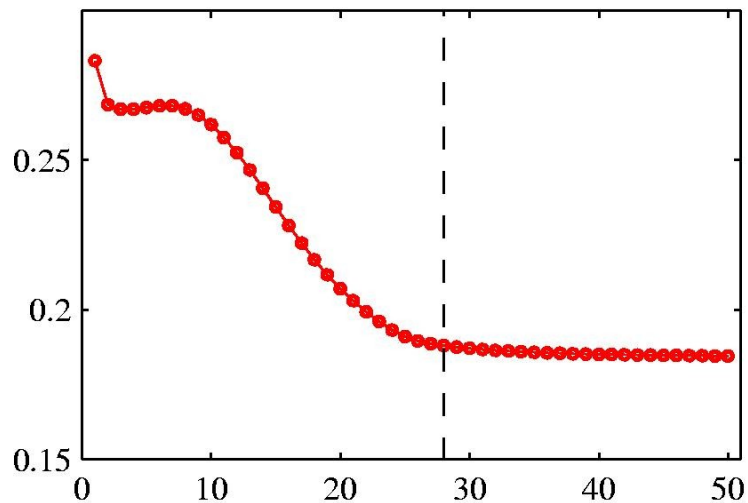
Use e.g. test datasets to alleviate this problem.



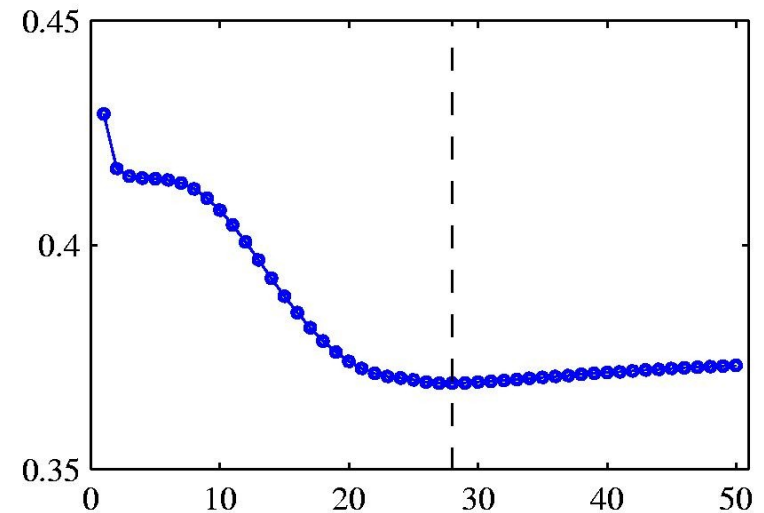
# Early stopping

...or just stop the training when overfitting starts to occur.

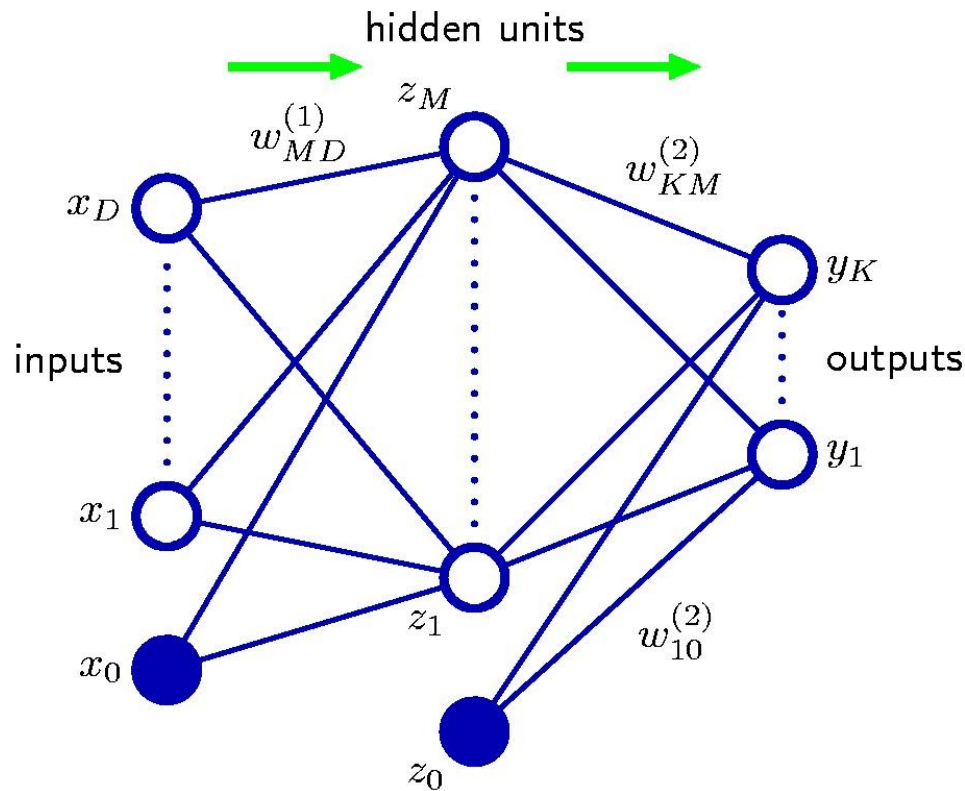
Training data



Test data



# Summary



- Neural network as general regression/classification models
- Feed-forward evaluation
- Maximum-likelihood framework
- Error back-propagation to obtain gradient