Statistical NLP for the Web

Neural Networks, Deep Belief Networks

Sameer Maskey

Week 8, October 24, 2012

*some slides from Andrew Rosenberg

Announcements

- Please ask HW2 related questions in courseworks
- HW2 due date has been moved to Oct 30 (next Tuesday)
- HW3 will be released next week

Student Projects

- Hashtag Recommendation for Twitter
- Reviews: How can the reviews help the restaurants improve more efficiently?
- Question Answering System dealing with factual questions in the field of Classical Music
- Automatic Summarization of Video Content
- Mood Sync: Text Mining for Mood Classification of Songs
- Web app for fashion item recognition
- TCoG
- Twitter Dedupe
- Unsupervised Medical Entity Recognition
- A Web App for Personalized Health News
- Twitter movie tweets sentiment analysis
- An intelligent newsreader service
- Legal Auto Assist

HW2

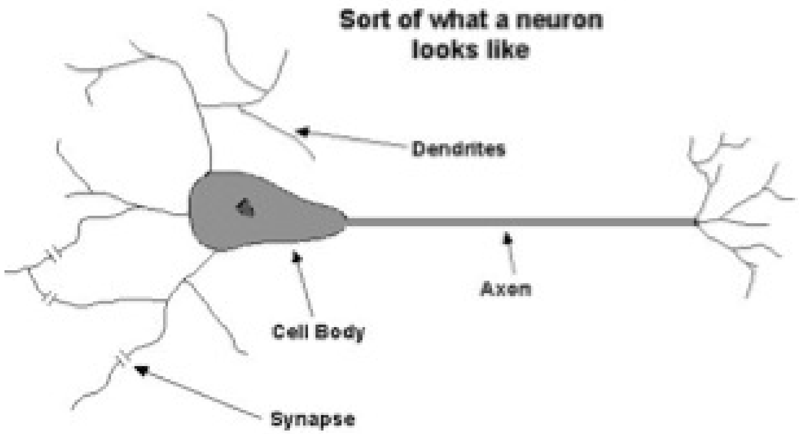
- How to do well in HW2?
  - Understand the concept clearly
  - Go through the animation of forward backward in the slides
  - Make sure you understand where each numbers are coming from
  - Also, take a look at Jason Eisner's excel sheet
  - You can make sure your algorithm is correct by first trying Eisner's example in the code
  - Make sure you do things in log probabilities

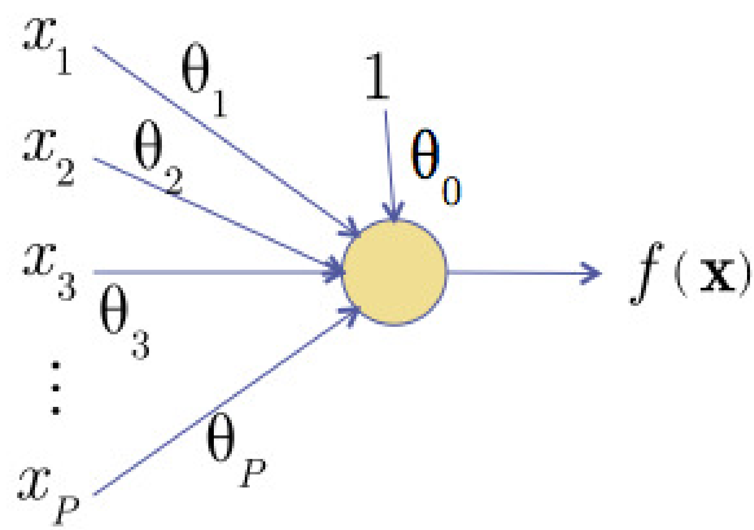
Topics for Today

- Neural Networks
- Deep Belief Networks

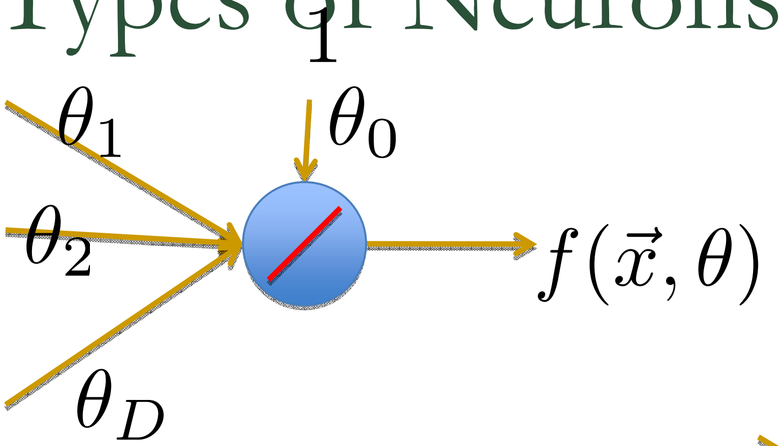
Neurons

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node

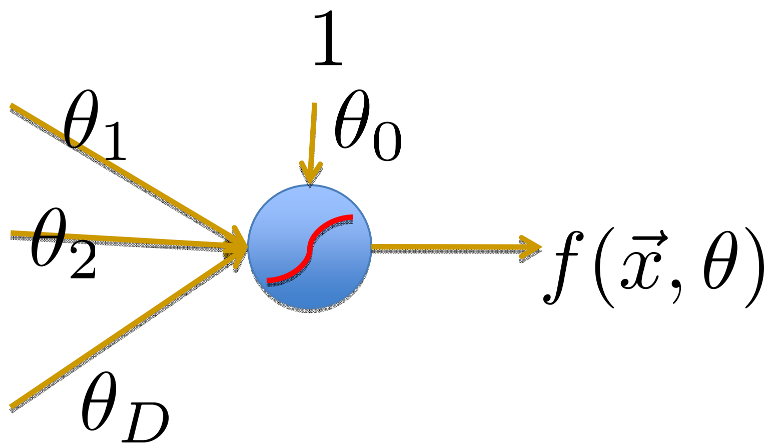




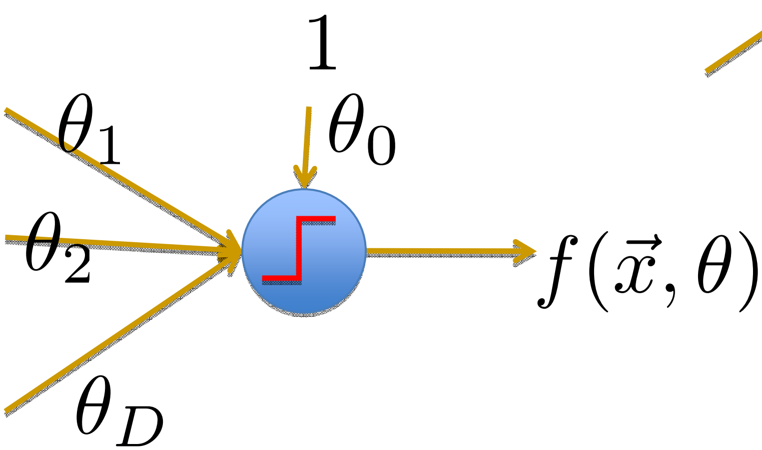
Types of Neurons



Linear Neuron



Logistic Neuron

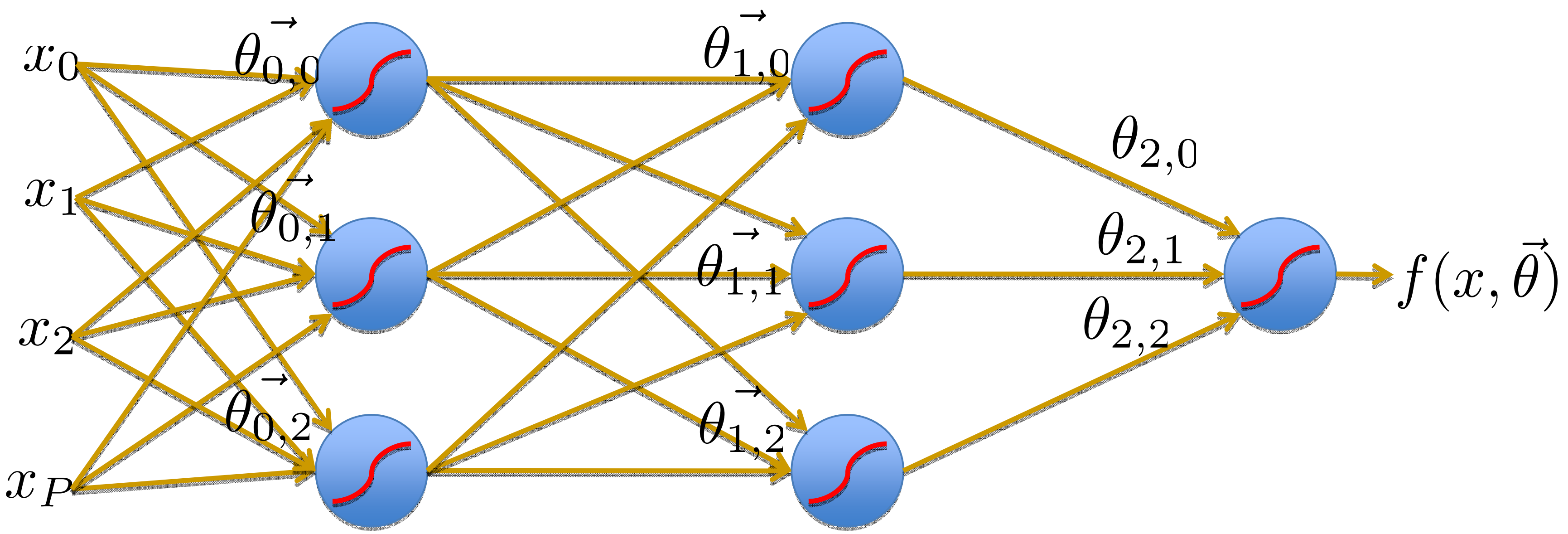


Perceptron

Potentially more. Require a convex loss function for gradient descent training.

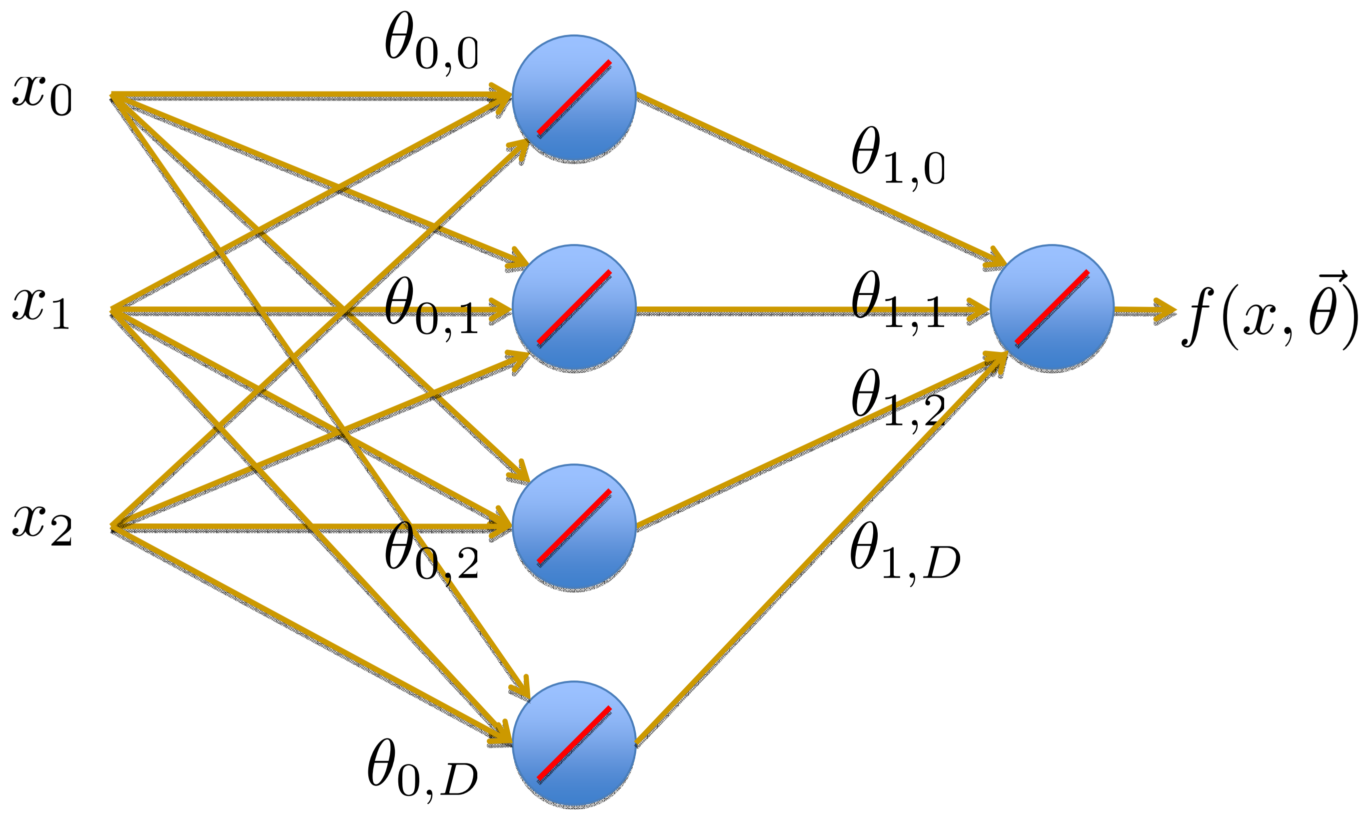
Multilayer Networks

- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights



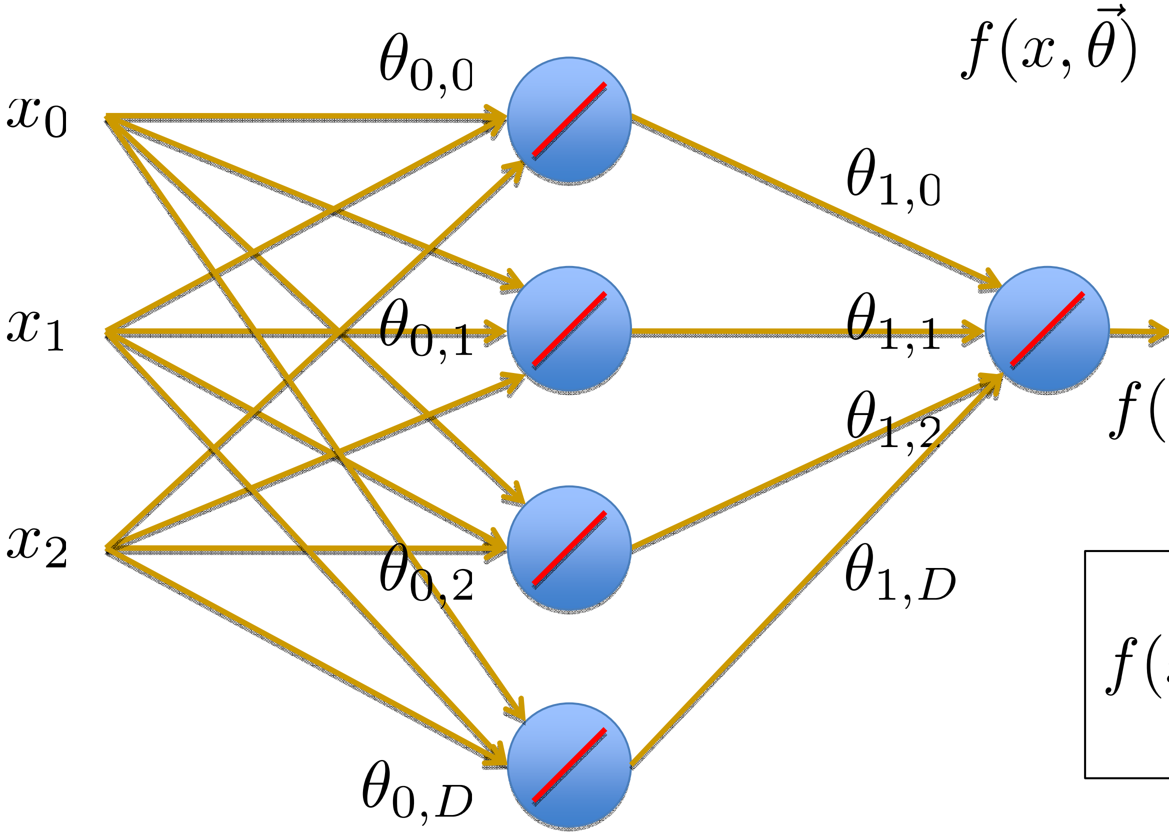
Linear Regression Neural Networks

- What happens when we arrange **linear neurons** in a multilayer network?



Linear Regression Neural Networks

- Nothing special happens.
  - The product of two linear transformations is itself a linear transformation.



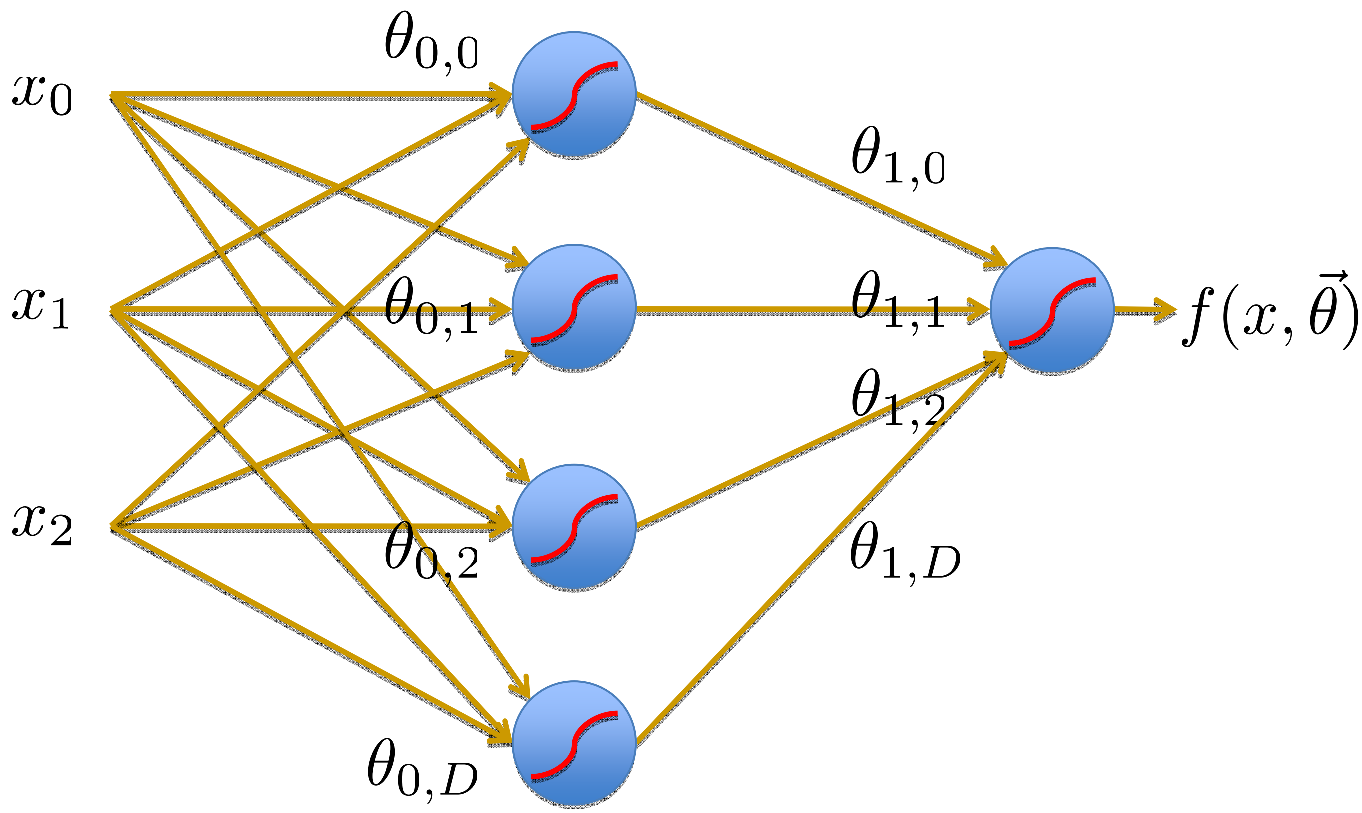
$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} \sum_{n=0}^{N-1} \theta_{0,i,n} x_n$$

$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} [\theta_{0,i}^T \vec{x}]$$

$$\boxed{f(x,\vec{\theta}) = \sum_{i=0}^{D} [\hat{\theta}_i^T \vec{x}]}$$

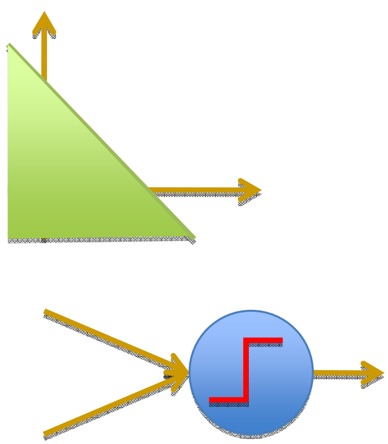
Neural Networks

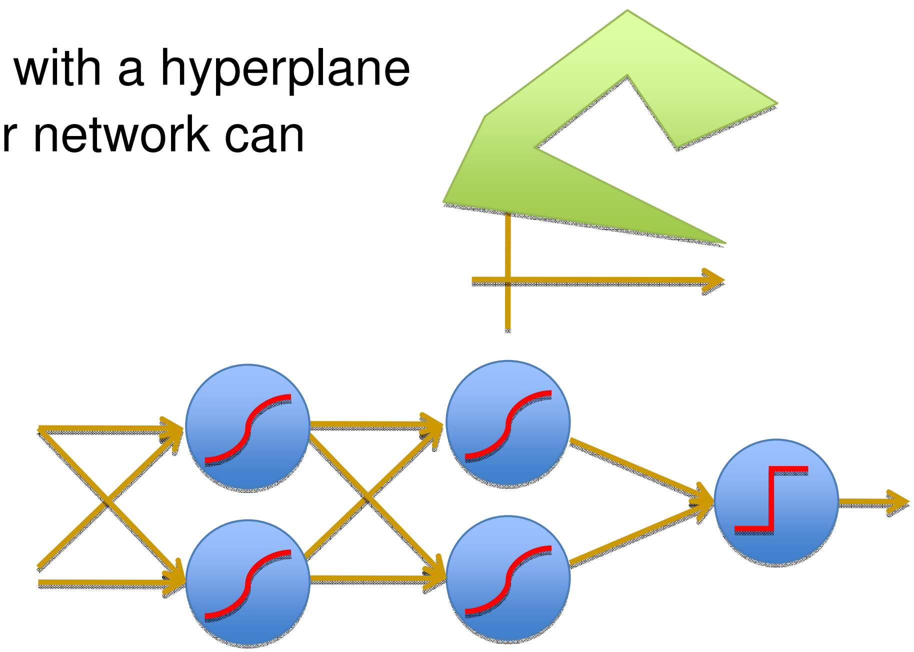
- We want to introduce non-linearities to the network.
  - Non-linearities allow a network to identify complex regions in space



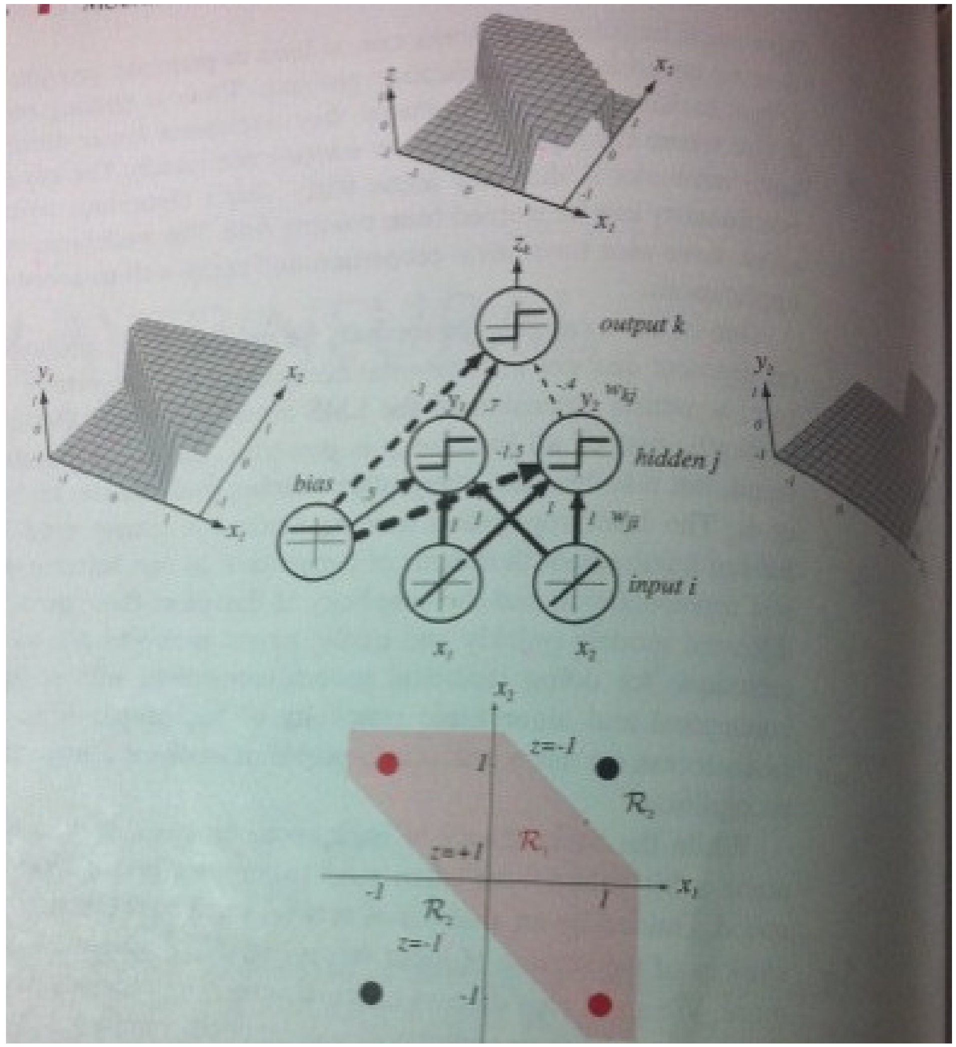
Linear Separability

- 1-layer cannot handle XOR
- More layers can handle more complicated spaces – but require more parameters
- Each node splits the feature space with a hyperplane
- If the second layer is AND a 2-layer network can represent any convex hull.



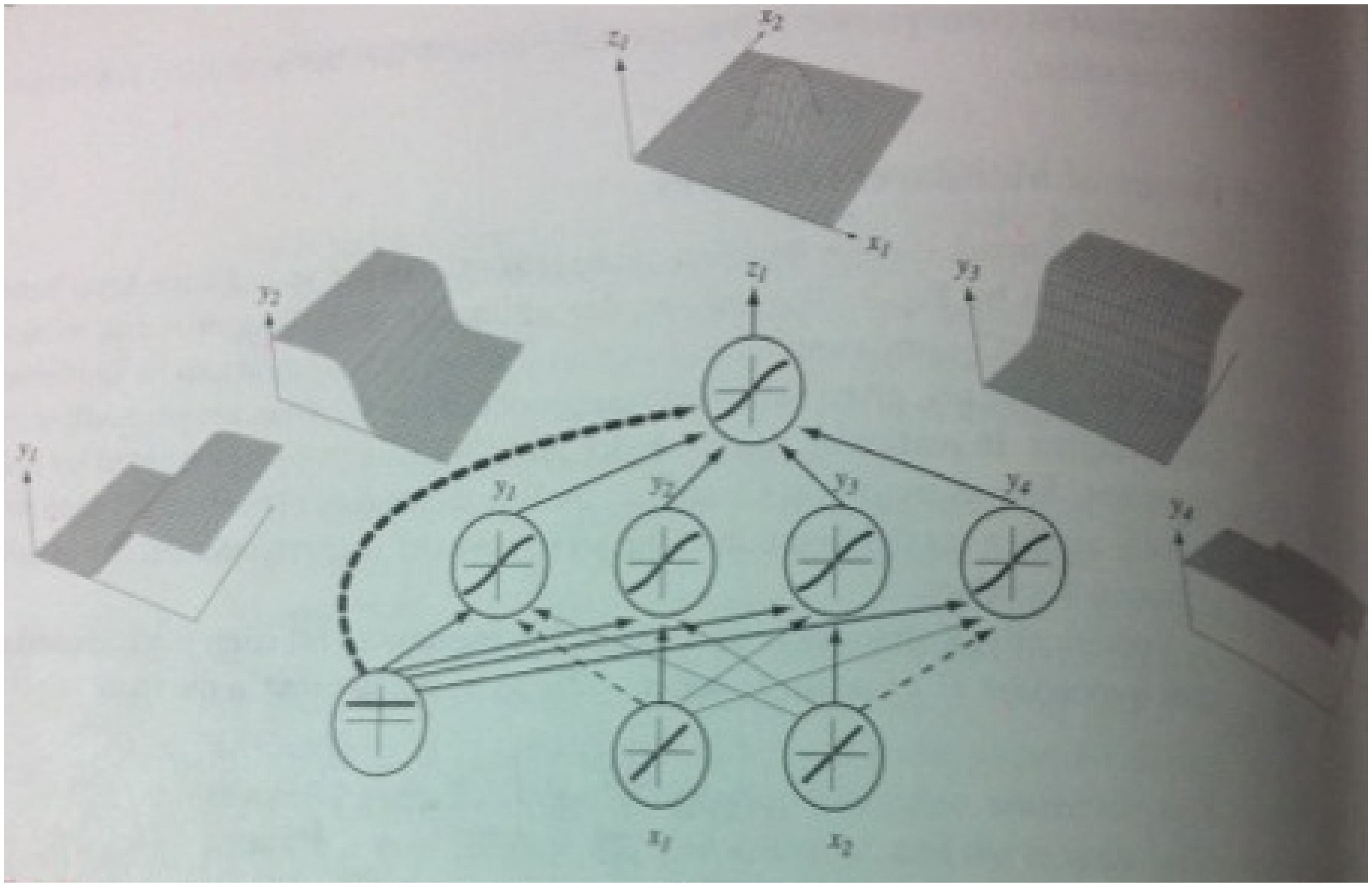


XOR Problem and Neural Net Solution



Picture from [1]

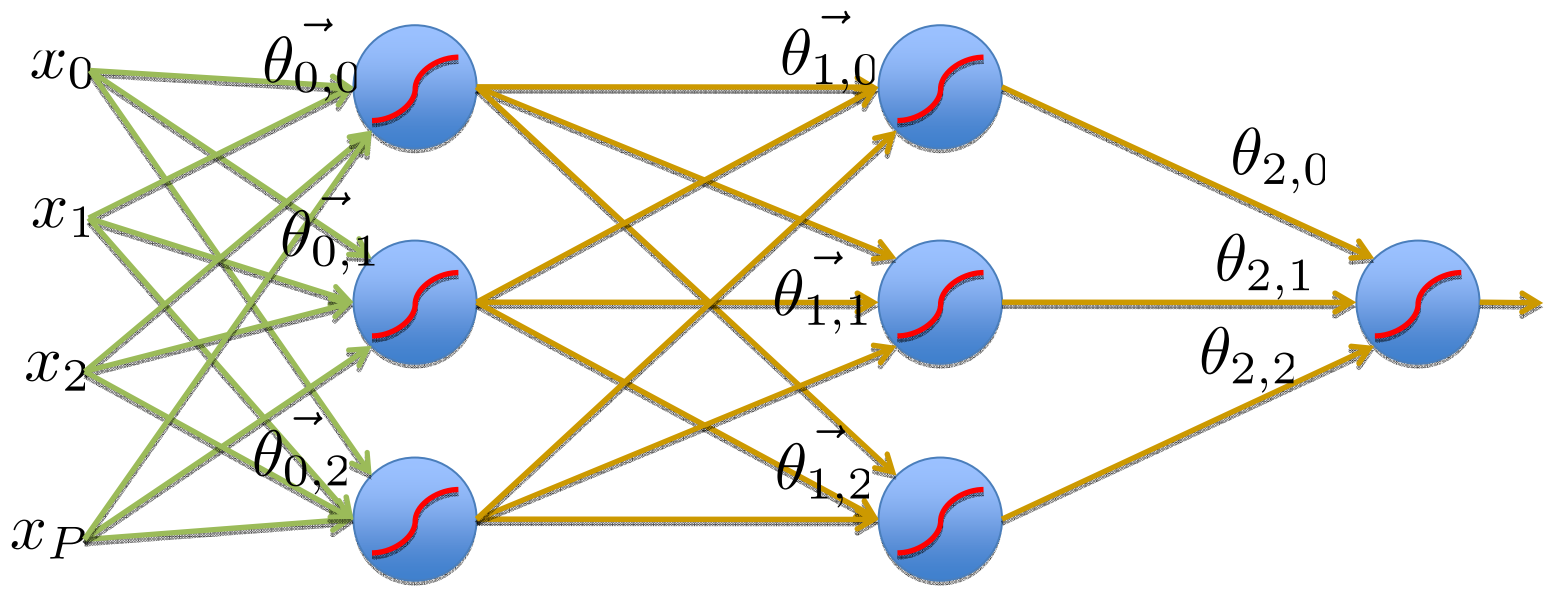
Neural Net



Picture from [1]

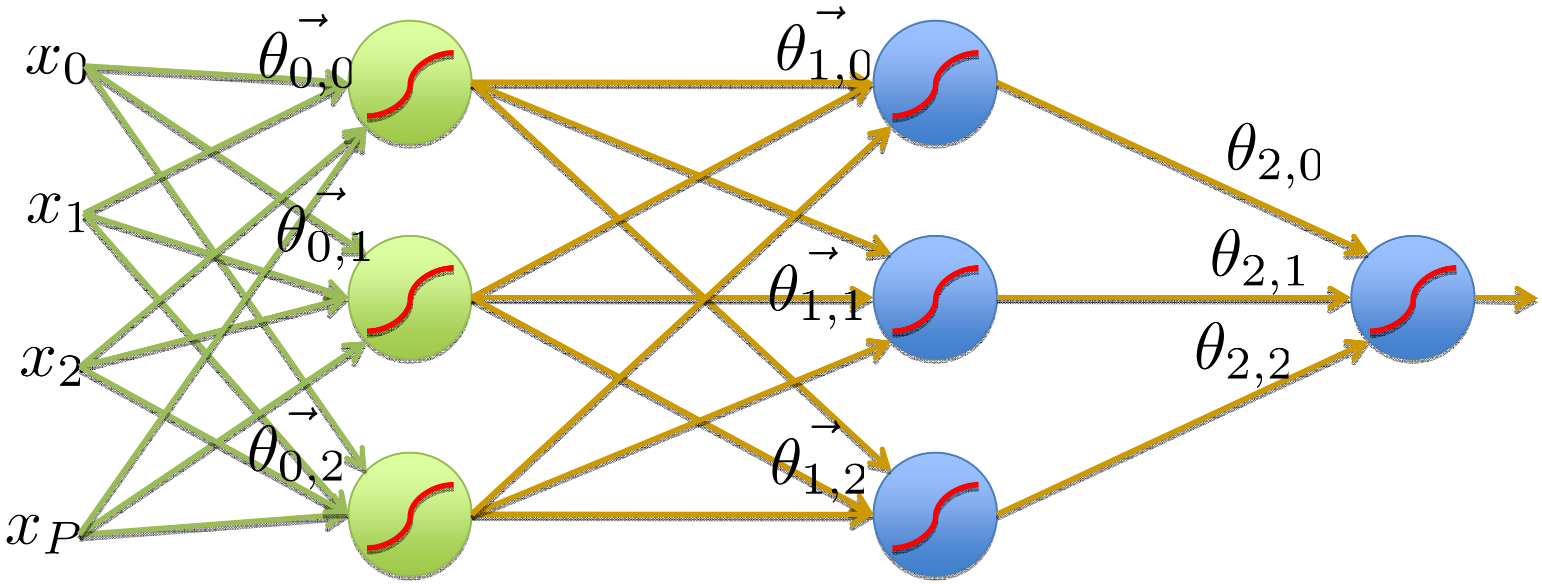
Feed-Forward Networks

- Predictions are fed forward through the network to classify



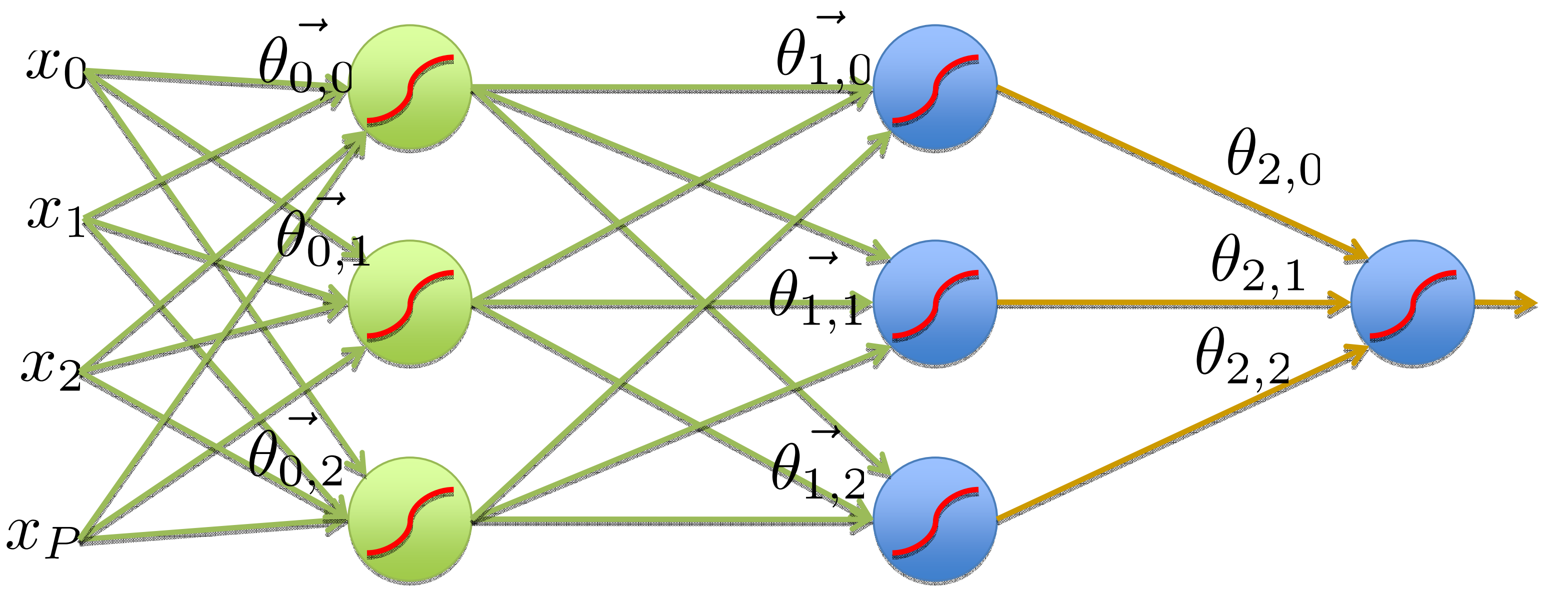
Feed-Forward Networks

- Predictions are fed forward through the network to classify



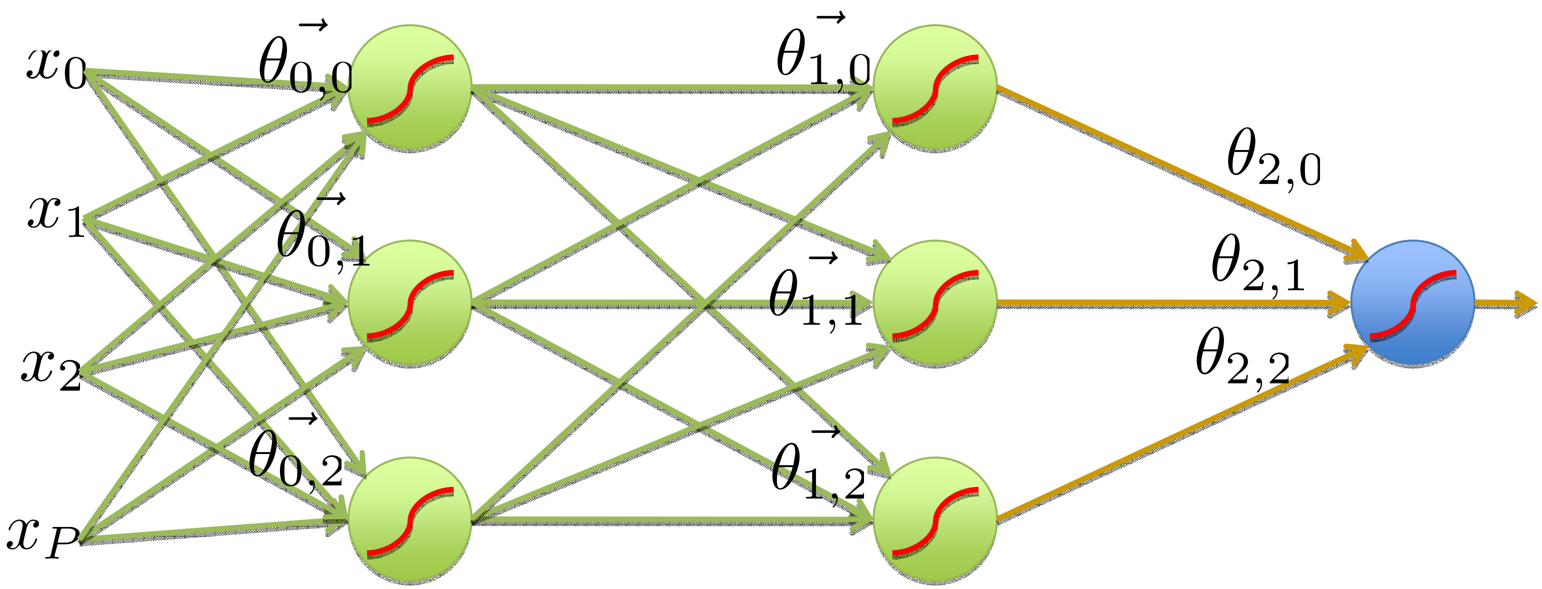
Feed-Forward Networks

- Predictions are fed forward through the network to classify



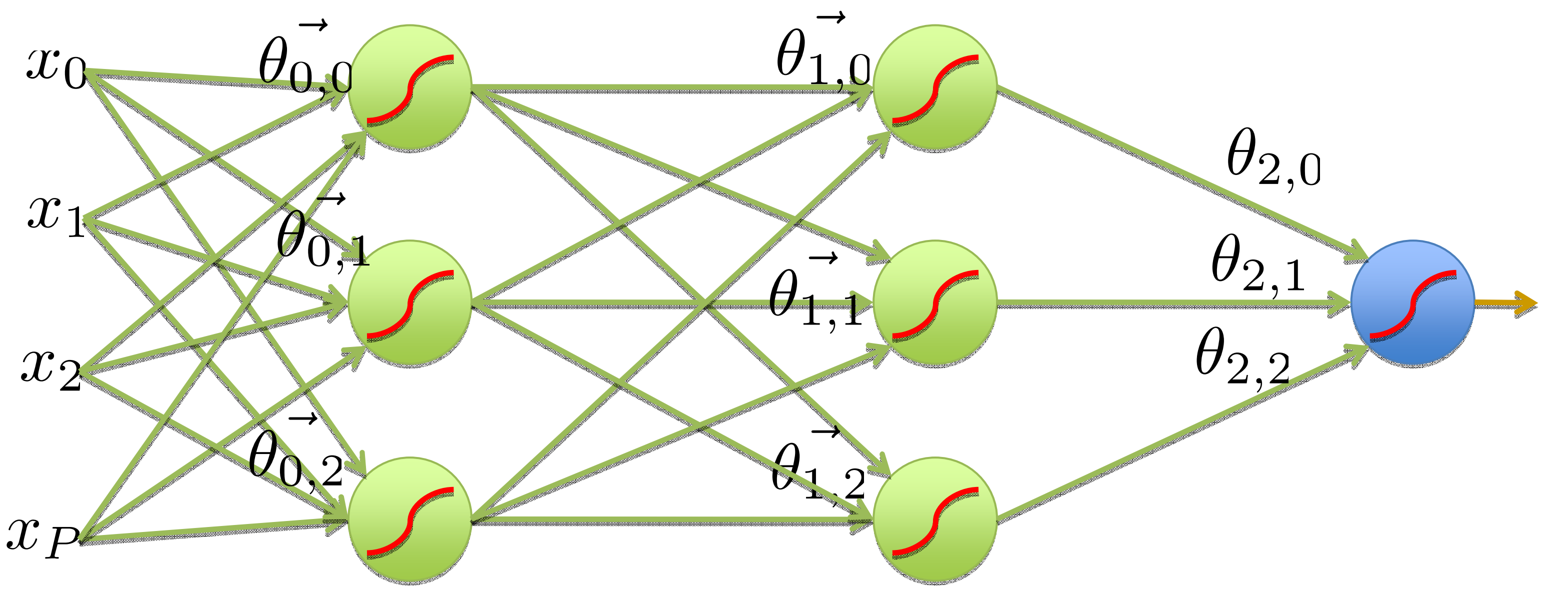
Feed-Forward Networks

- Predictions are fed forward through the network to classify



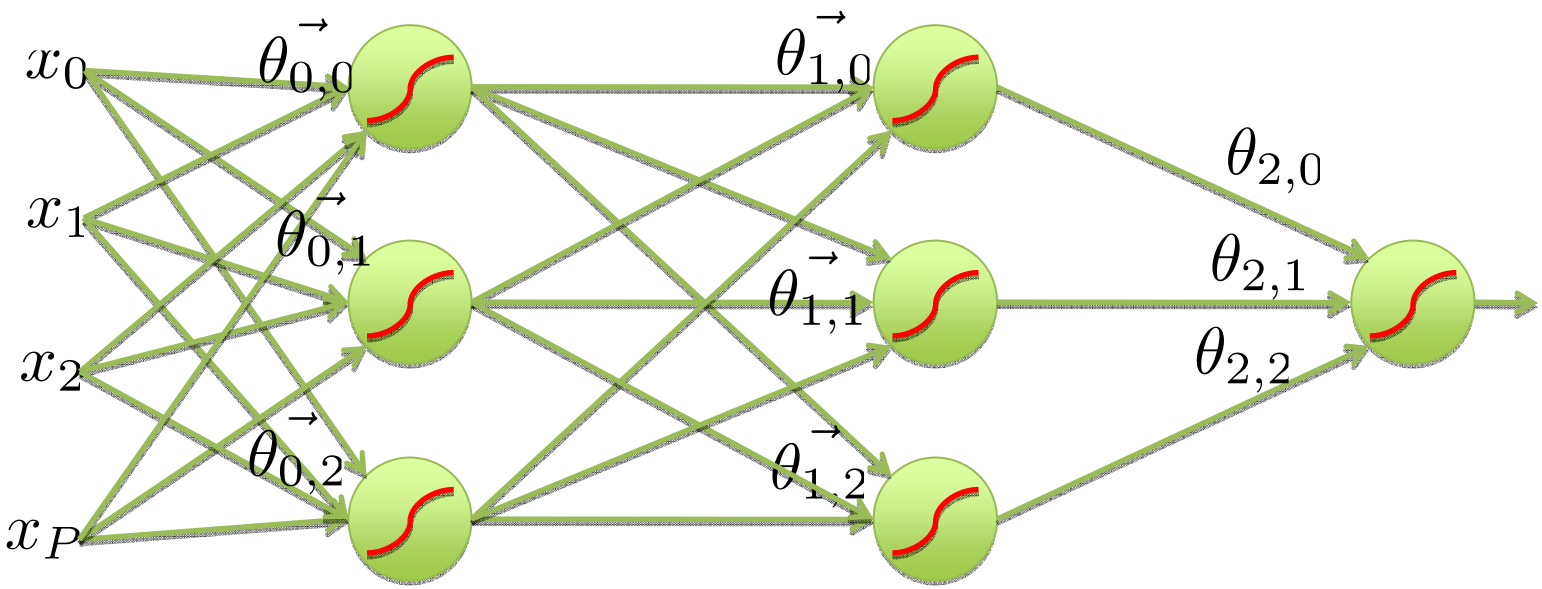
Feed-Forward Networks

- Predictions are fed forward through the network to classify



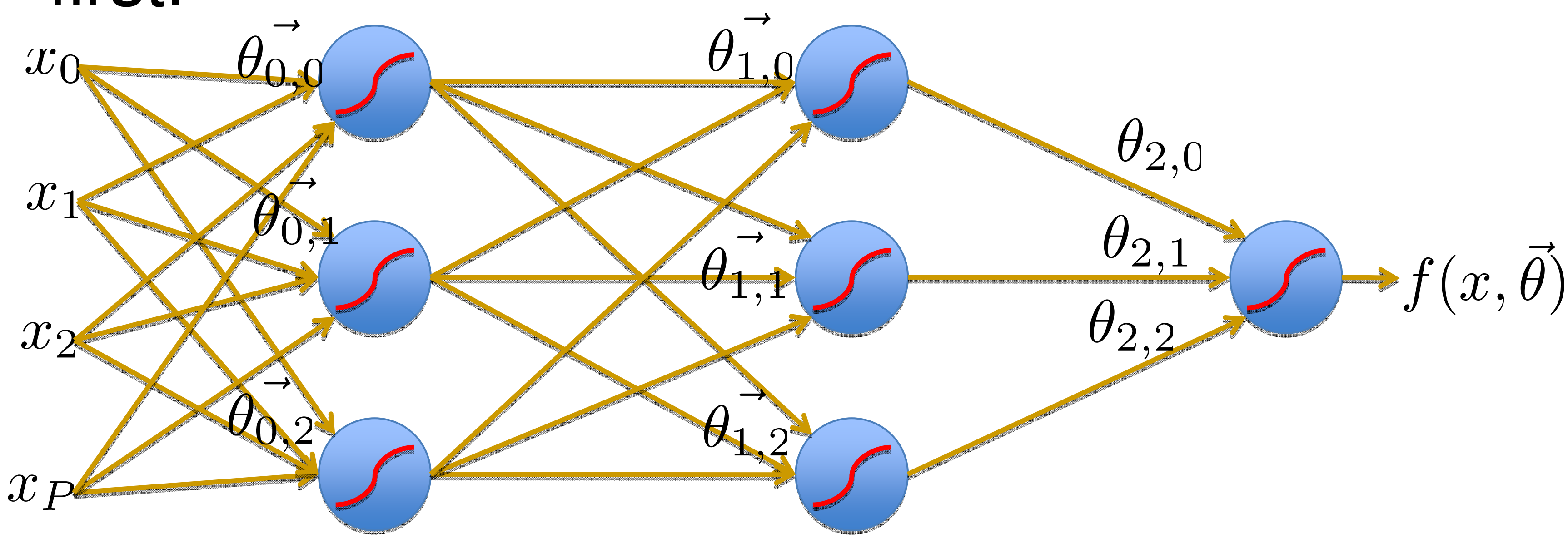
Feed-Forward Networks

- Predictions are fed forward through the network to classify



Error Backpropagation

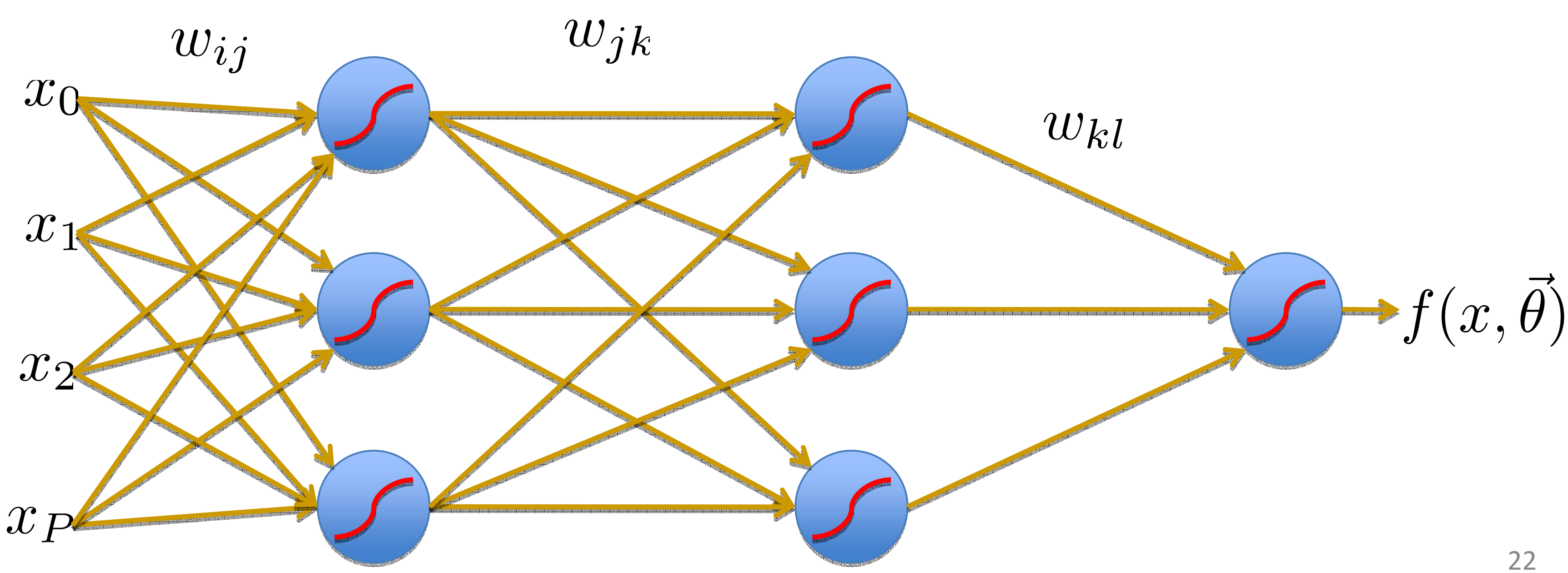
- We will do gradient descent on the whole network.
- Training will proceed from the last layer to the first.



Error Backpropagation

- Introduce variables over the neural network

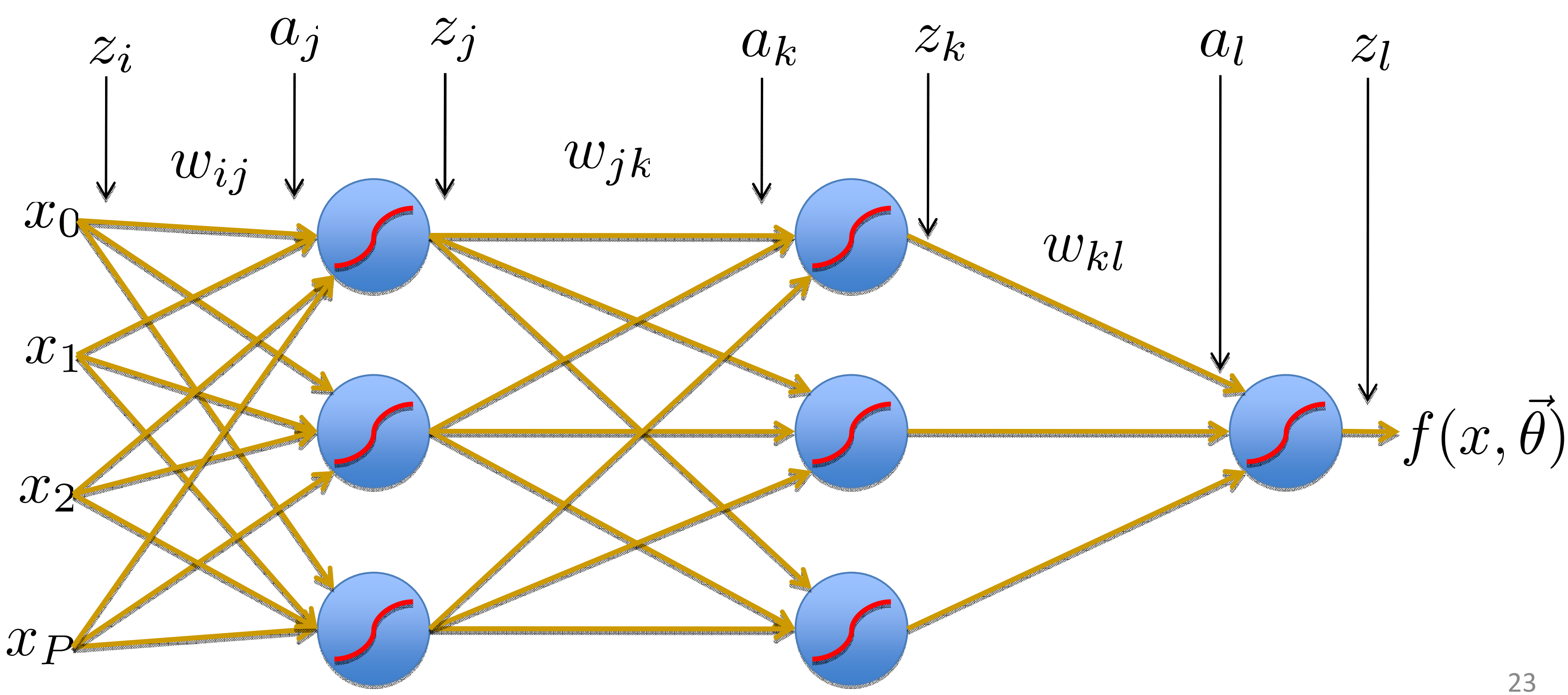
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

- Introduce variables over the neural network
  - Distinguish the input and output of each node

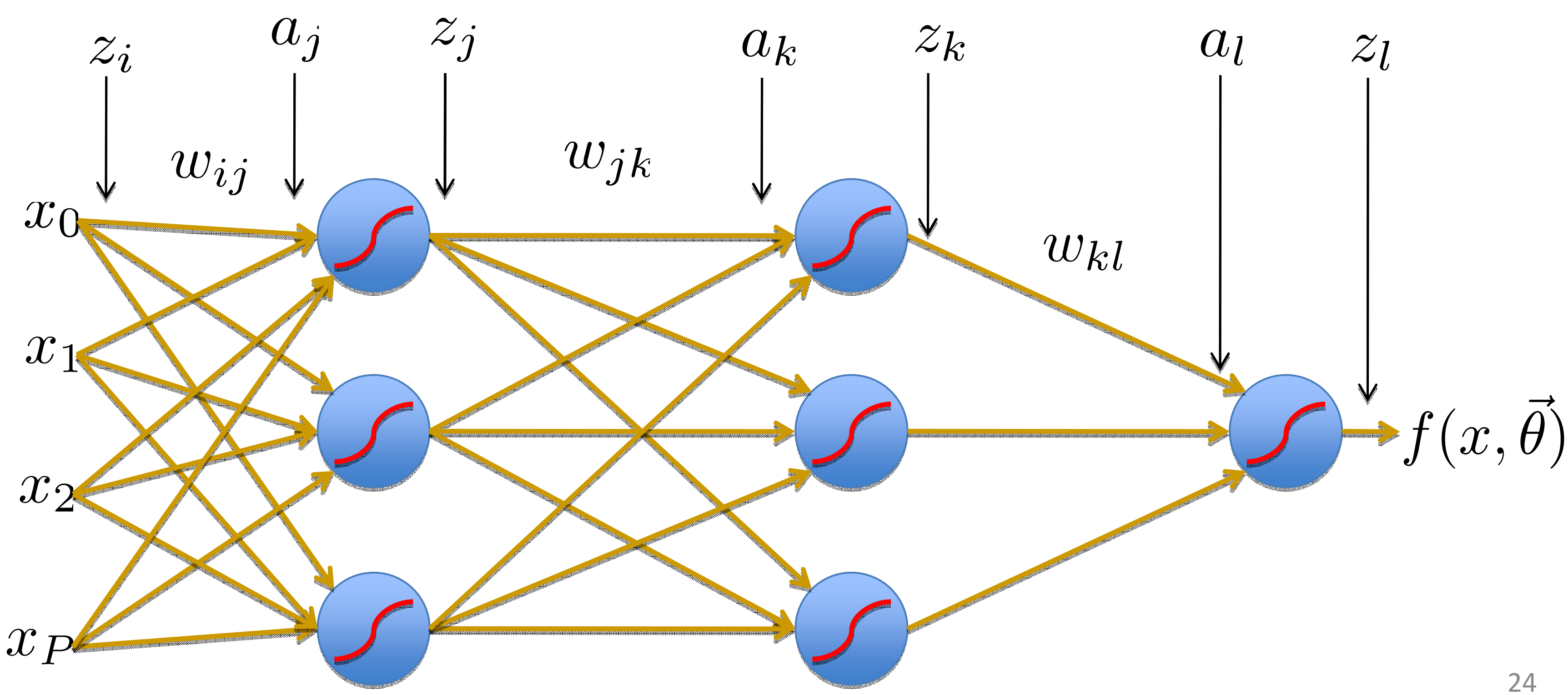


Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \sum_j w_{jk} z_j \qquad a_l = \sum_k w_{kl} z_k$$

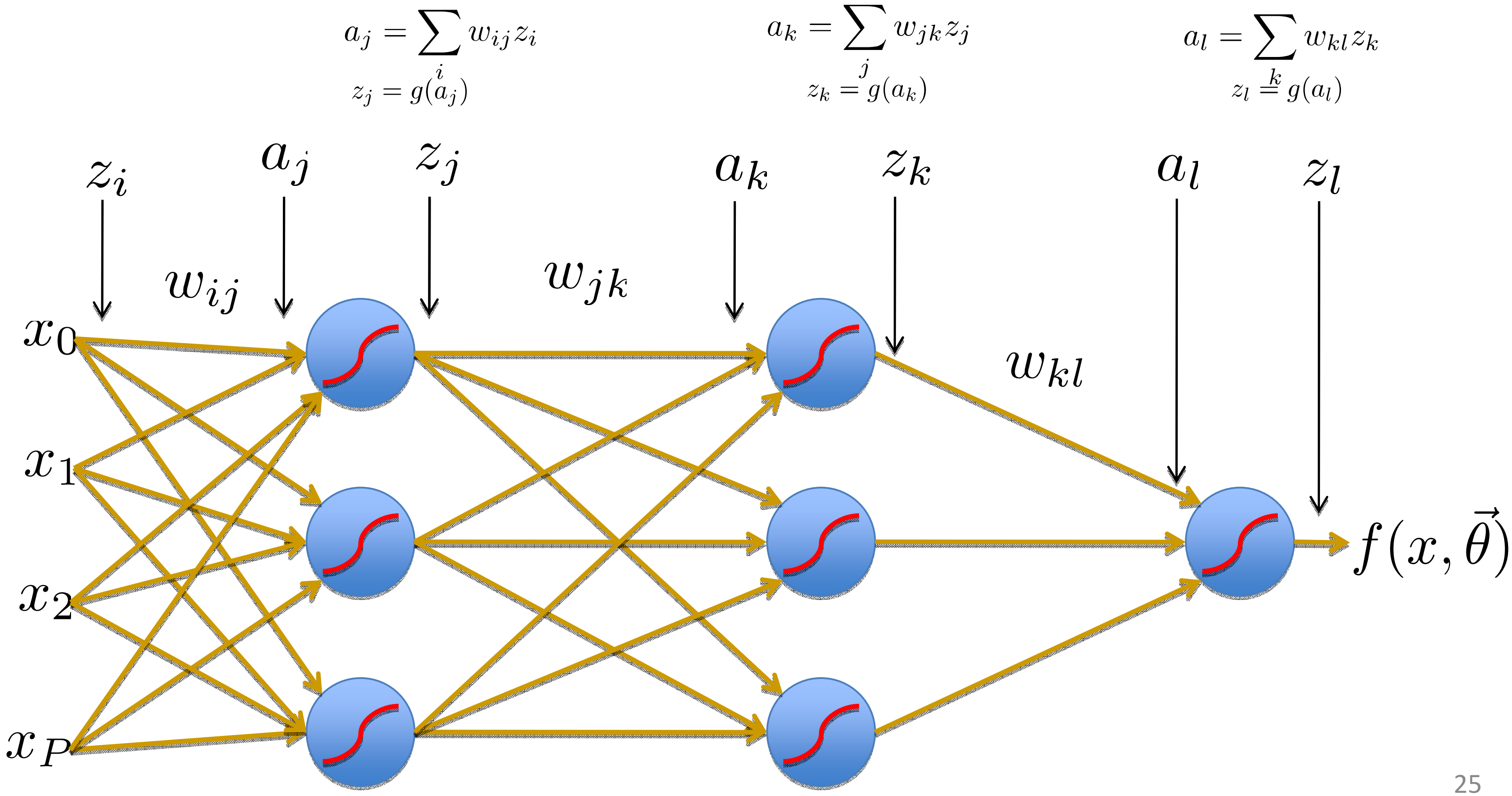
$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

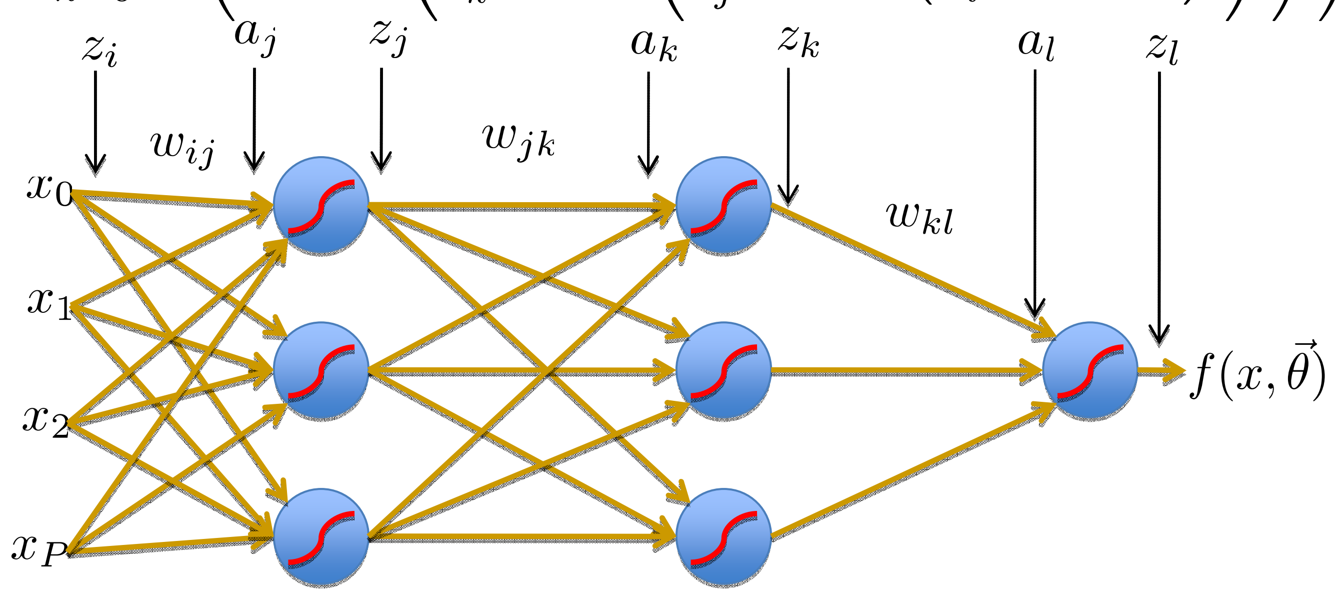
Training: Take the gradient of the last component and iterate backwards



Error Backpropagation

$$
\begin{aligned}
R(\theta) &= \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n)) \\
&= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - f(x_n)\right)^2 \\
&= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{kl}\, g\left(\sum_j w_{jk}\, g\left(\sum_i w_{ij} x_{n,i}\right)\right)\right)\right)^2
\end{aligned}
$$

Empirical Risk Function



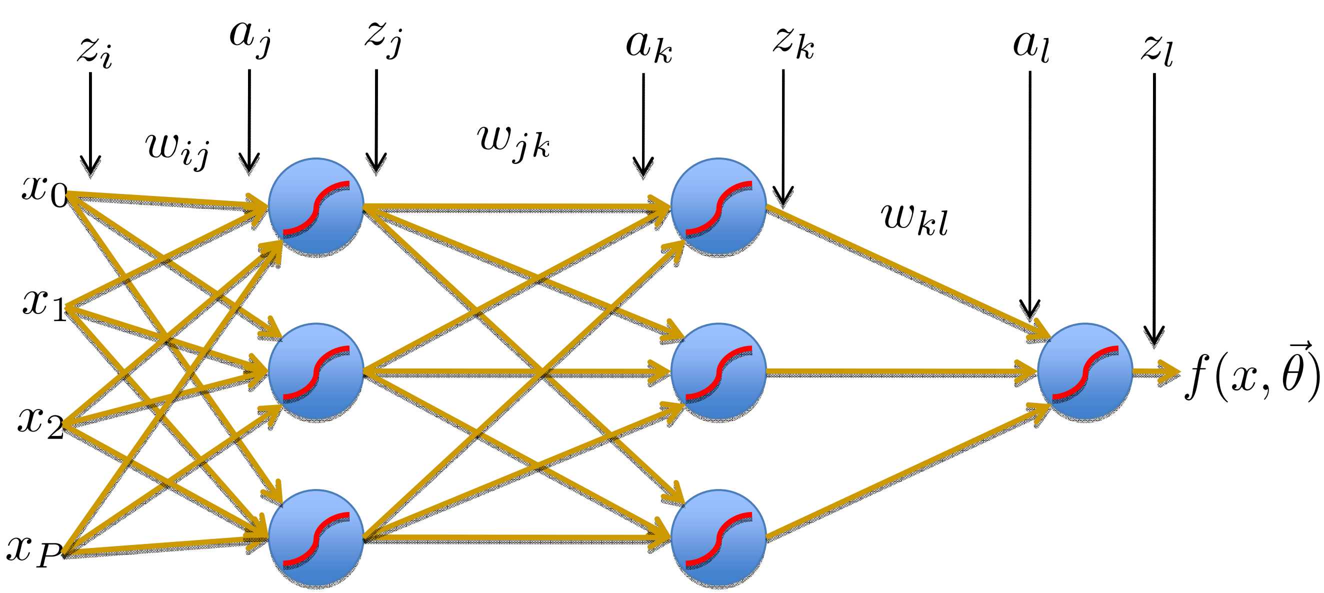
Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule



Chain Rule

- What is chain rule saying?
  - If we want to know how error changes when the weights change we can think of it as
    - See how error changes when the input to the weight changes
    - Multiply it with a factor that shows how the input changes when the weight changes

Error Backpropagation

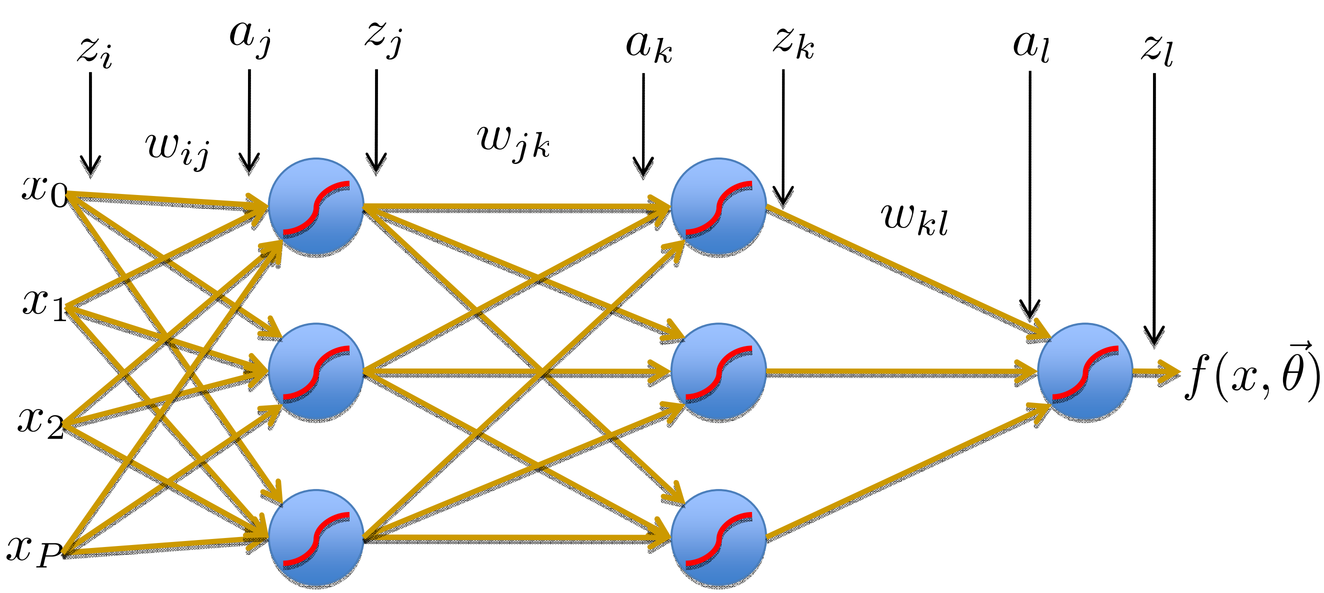
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$



Error Backpropagation

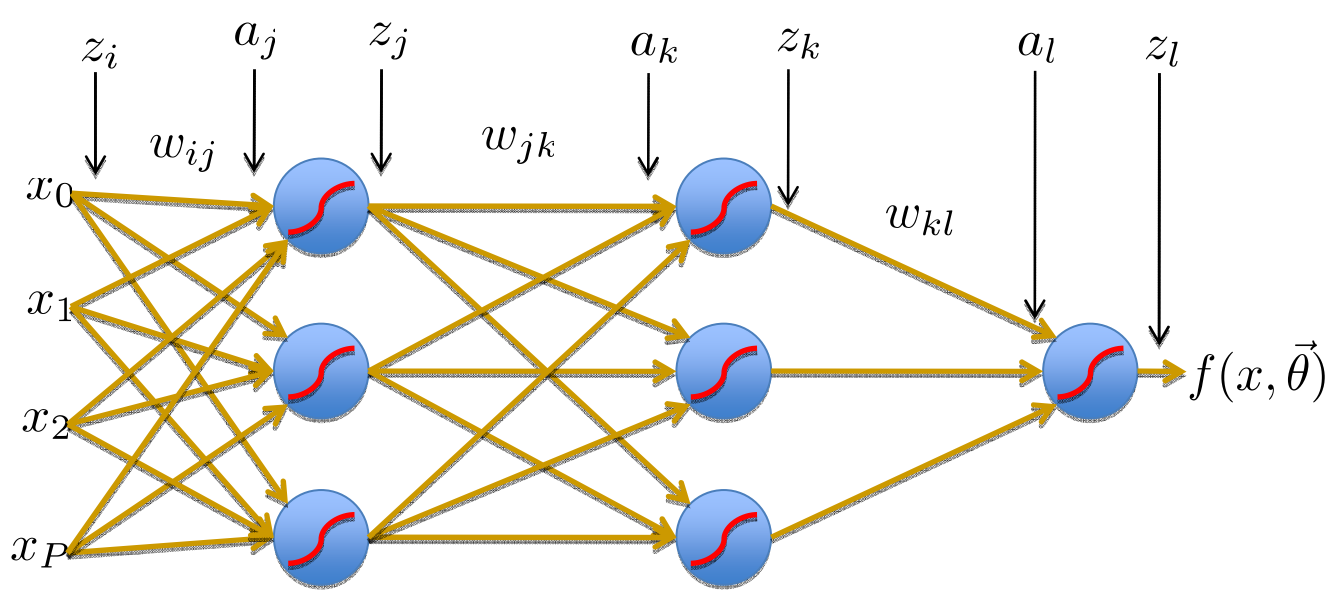
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right]$$



- Remember

$$\frac{\pm}{\pm w_{ik}}\left(t_k - \sum_j w_{jk} x_j\right) = -x_i \qquad \text{when i=j}$$

Only part of the sum that is function of w_{ik} is when $i = j$

Error Backpropagation

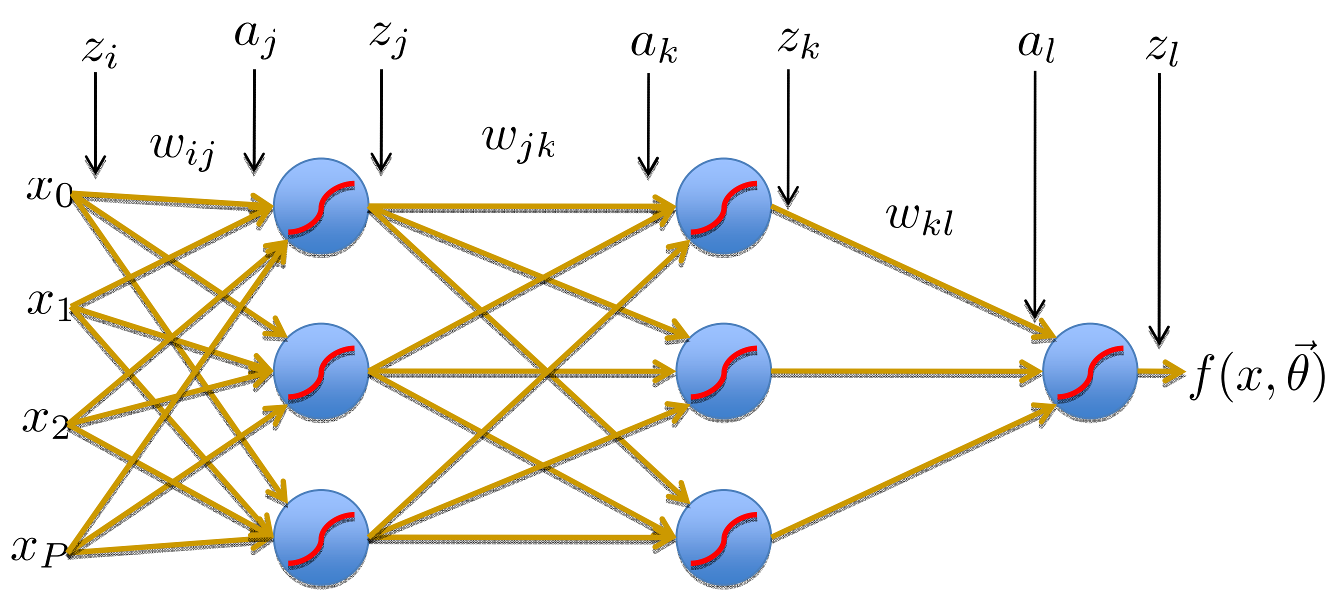
Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \boxed{\left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]} \boxed{\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right]} = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$



Error Backpropagation

Optimize last layer weights w_{kl}

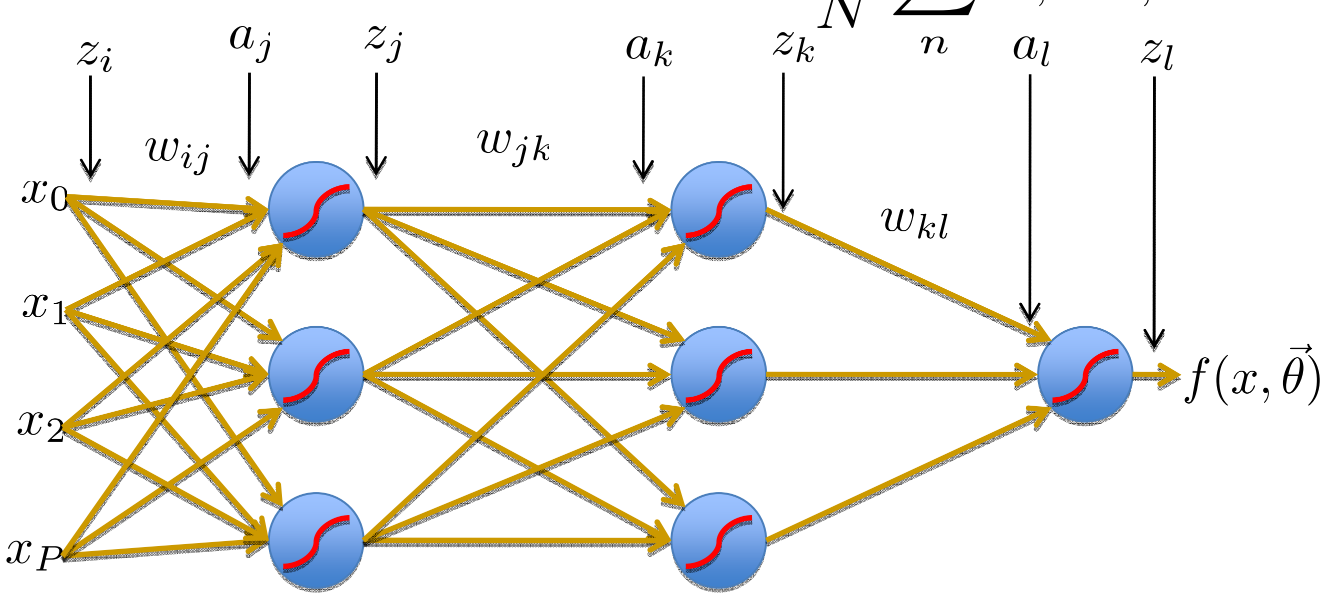
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \boxed{\left[-(y_n - z_{l,n})g'(a_{l,n})\right]} z_{k,n}$$

$$= \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

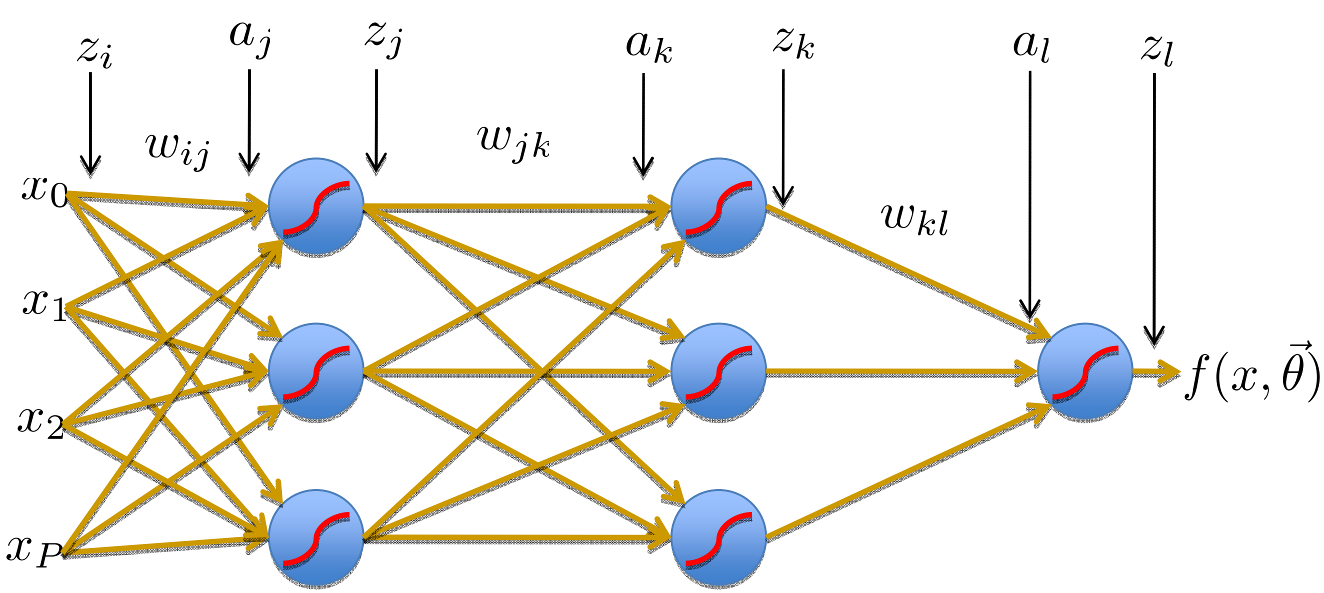


Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$



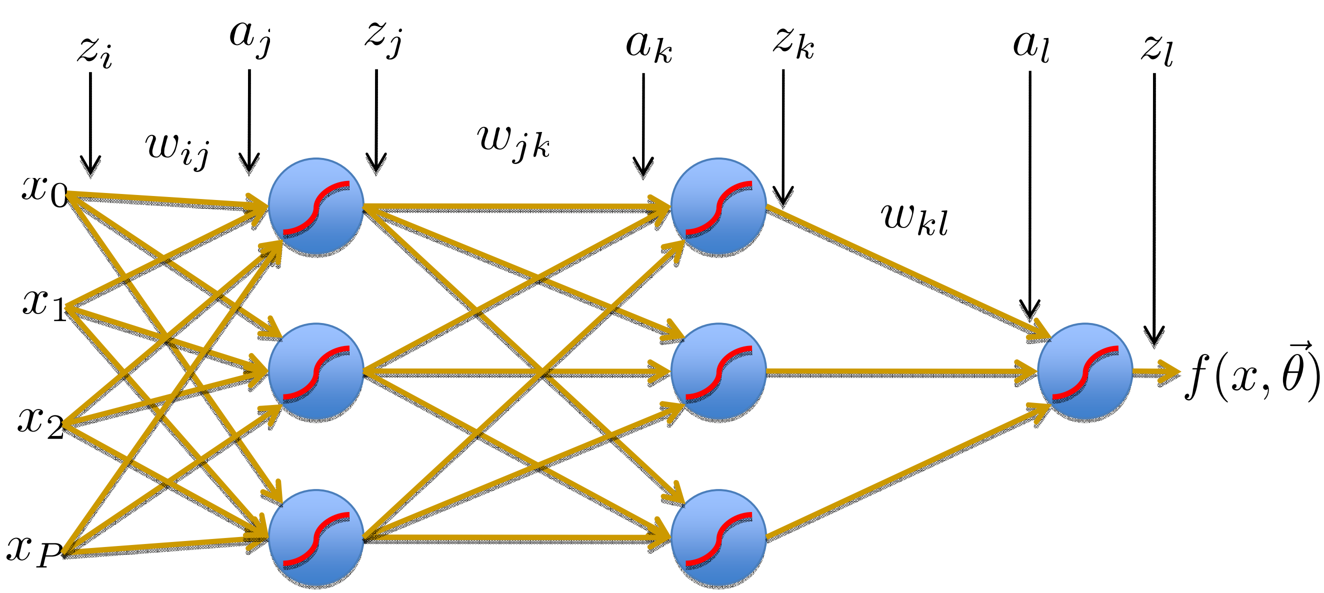
Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule



Error Backpropagation

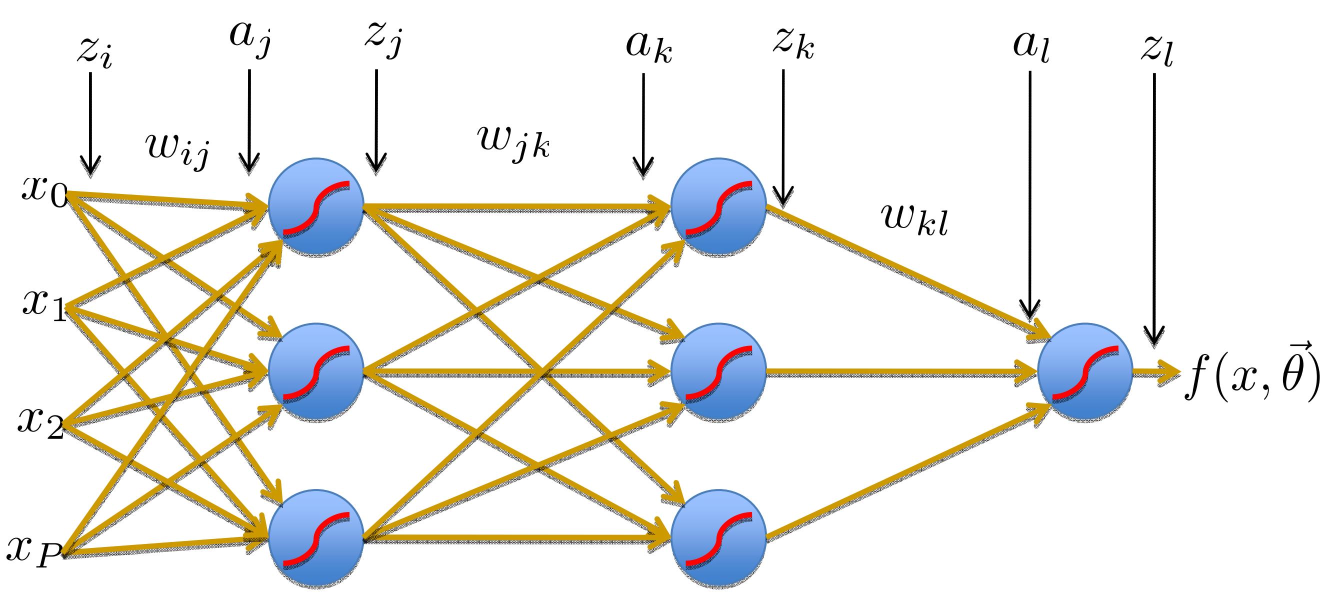
Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] [z_{j,n}]$$



Error Backpropagation

Optimize last hidden weights w_{jk}

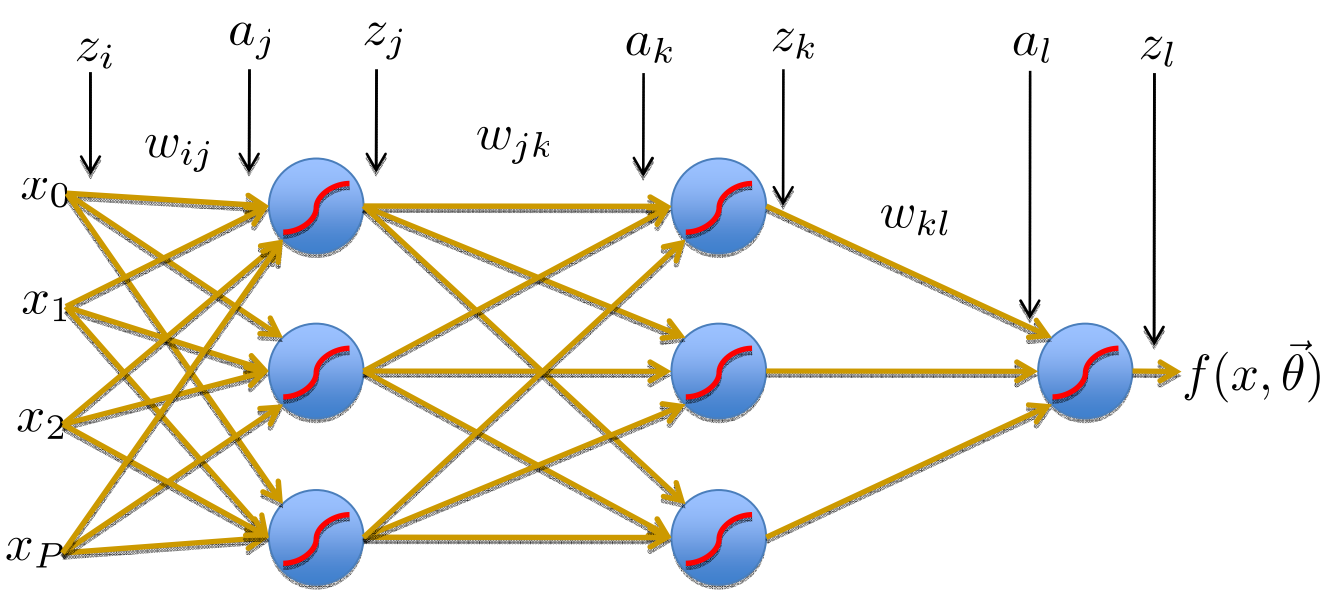
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l \frac{\partial a_{l,n}}{\partial a_{k,n}}\right][z_{j,n}]$$

$$a_l = \sum_k w_{kl} g(a_k)$$



Error Backpropagation

Optimize last hidden weights w_{jk}

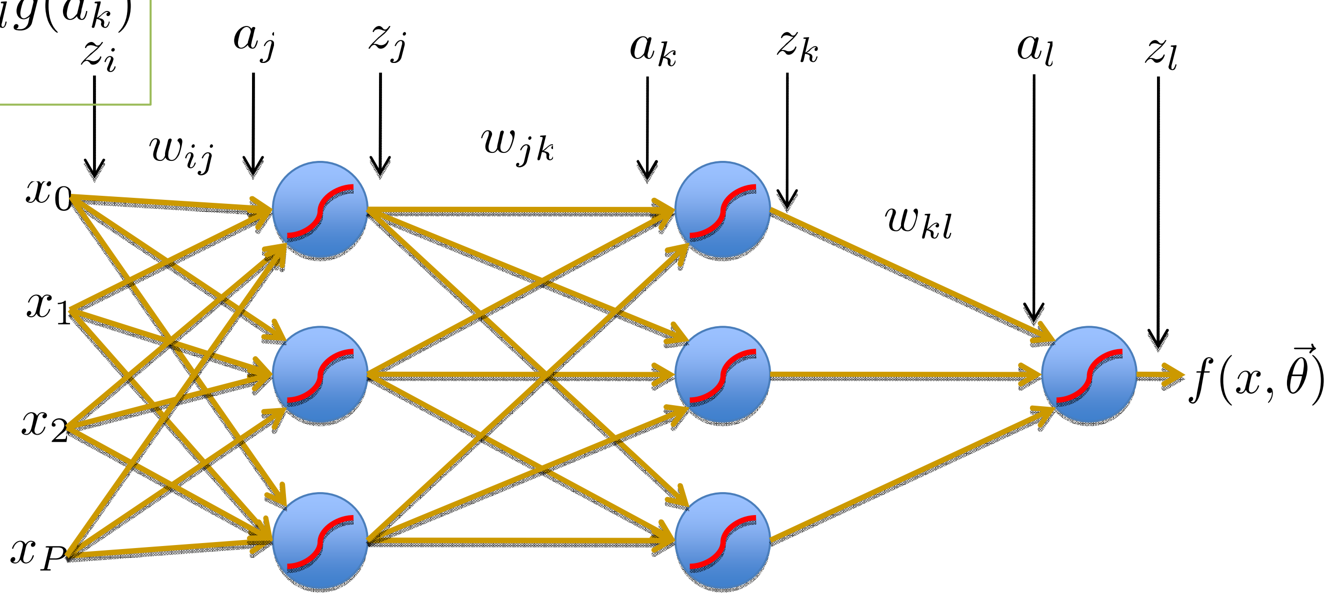
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\quad\right][z_{j,n}] = \frac{1}{N}\sum_n [\delta_{k,n}]\,[z_{j,n}$$

$$a_l = \sum_k w_{kl}\, g(a_k)$$



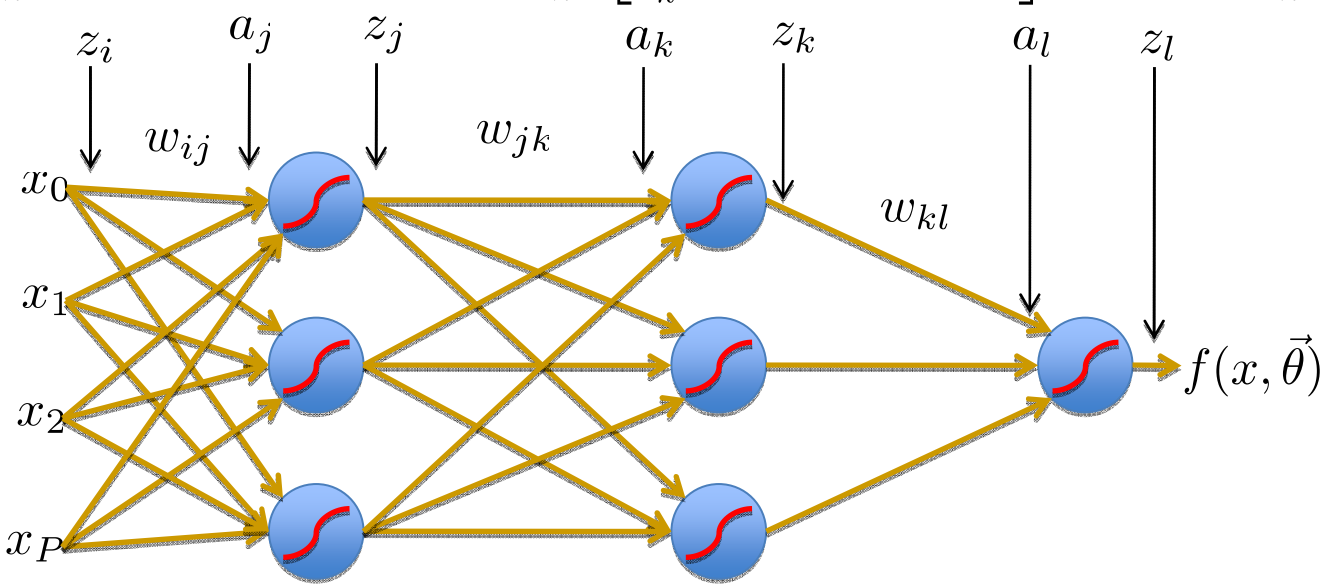
Error Backpropagation

Repeat for all previous layers

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

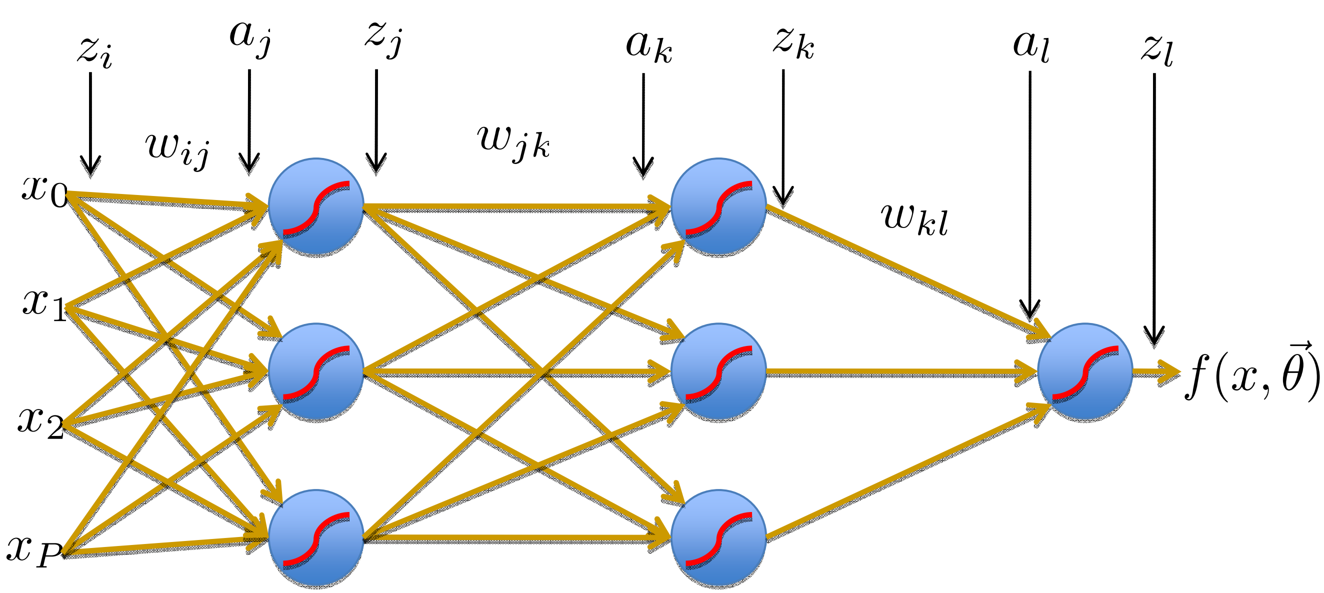
$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$



Error Backpropagation

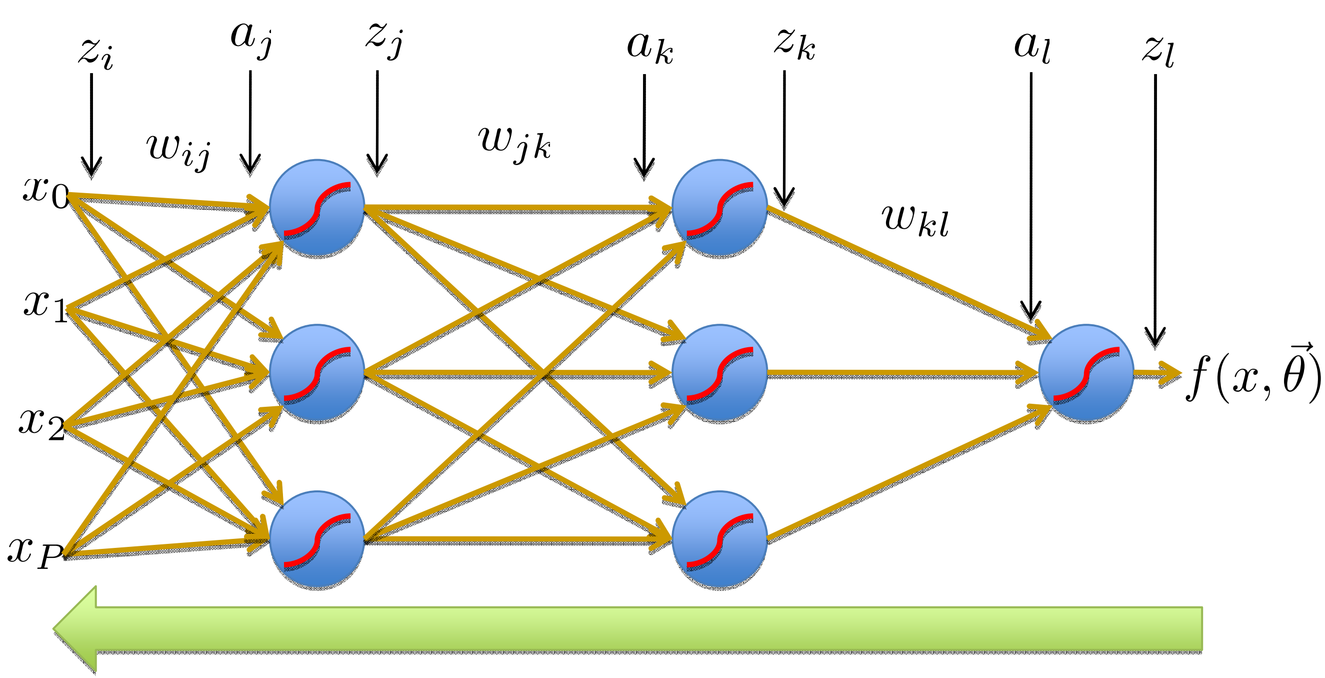
Now that we have well defined gradients for each parameter, update using Gradient Descent

$$
\begin{aligned}
w_{ij}^{t+1} &= w_{ij}^{t} - \eta \frac{\partial R}{w_{ij}} \\
w_{jk}^{t+1} &= w_{jk}^{t} - \eta \frac{\partial R}{w_{kl}} \\
w_{kl}^{t+1} &= w_{kl}^{t} - \eta \frac{\partial R}{w_{kl}}
\end{aligned}
$$

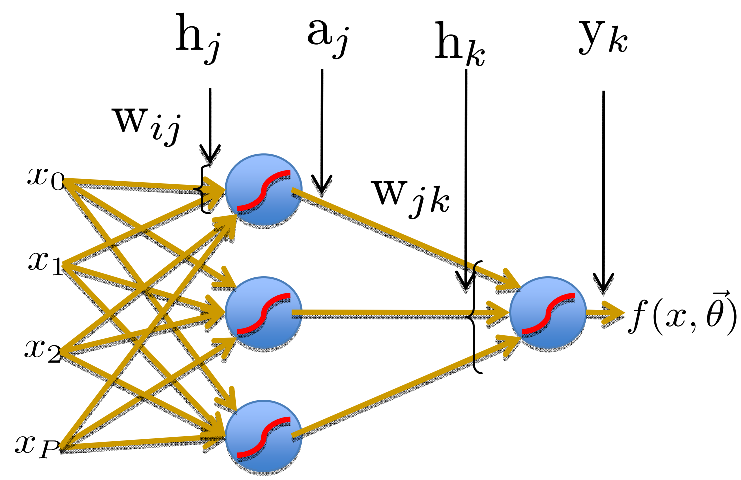


Error Back-propagation

- Error backprop unravels the multivariate chain rule and solves the gradient for each partial component separately.
- The target values for each layer come from the next layer.
- This feeds the errors back along the network.



Neural Net Algorithm : Forward Phase



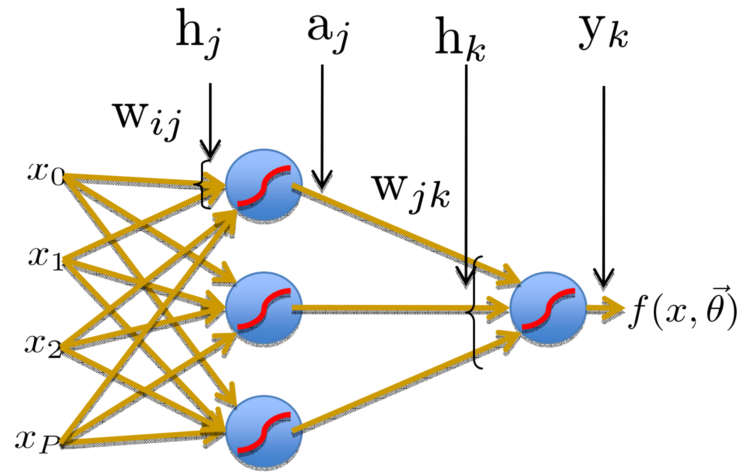
$$h_j = \sum_i x_i w_{ij}$$

$$a_j = g(h_j) = 1/(1 + e^{-\beta h_j})$$

$$\mathrm{h}_k = \sum_j a_j w_{jk}$$

$$\mathrm{y}_k = g(h_k) = 1/(1 + e^{-\beta h_k})$$

Neural Networks : Backward Phase



$$\delta_{ok} = (t_k - y_k)y_k(1 - y_k)$$

$$\delta_{hj} = a_j(1 - a_j)\sum_k w_{jk}\delta_{ok}$$

$$w_{jk} \leftarrow w_{jk} + \eta\delta_{ok}a_j$$

$$w_{ij} \leftarrow +\eta\delta_{hj}x_i$$

Deriving Backprop Again

- Remember

$$\frac{\delta}{\delta w_{ik}}(t_k - \sum_j w_{jk} x_j) = -x_i$$ when i=j

Only part of the sum that is function of w_{ik} is when $i = j$

Also Derivative of Activation Function

$$g(h) = \frac{1}{1+e^{-\beta h}}$$

$$\frac{dg}{dh} = \frac{d}{dh}\frac{1}{1+e^{-\beta h}}$$

$$= \beta g(h)(1 - g(h))$$

Backpropagation of Error

$$\frac{\delta E}{\delta w_{jk}} = \frac{\delta E}{\delta h_k}\frac{\delta h_k}{\delta w_{jk}}$$

$$\frac{\delta E}{\delta w_{jk}} = \left(\frac{\delta E}{\delta y_k}\frac{\delta y_k}{\delta h_k}\right)\frac{\delta h_k}{\delta w_{jk}}$$

$$\frac{\delta}{\delta y_k}\frac{1}{2}\sum_k (y_k - t_k)^2$$

$$\frac{\delta h_k}{\delta w_{jk}} = \frac{\delta \sum_l w_{lk} a_l}{\delta w_{jk}}$$

$$= \sum_l \frac{\delta w_{lk} a_l}{\delta w_{jk}}$$

$$\underbrace{(y_k - t_k) \qquad y_k(1 - y_k)} \qquad \mathrm{a}_j$$

$$\mathrm{w}_{jk} \leftarrow w_{jk} + \eta \delta_{ok} a_j$$

Problems with Neural Networks

- Neural Networks can easily overfit
  - Many parameters to estimate
- It's hard to interpret the numbers produced by hidden layer

Types of Neural Networks

- Convolutional Networks
- Multiple Outputs
- Skip Layer Network
- Recurrent Neural Networks

What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.

Backpropagation Problems

- Backpropagation does not scale well with many hidden layer
- Requires a lot of data
- Easily stuck in poor local minima
- Use similar gradient method to adjust weights but maximize the likelihood of data given the model
  - Deep Belief Networks

Deep Belief Network in NLP and Speech

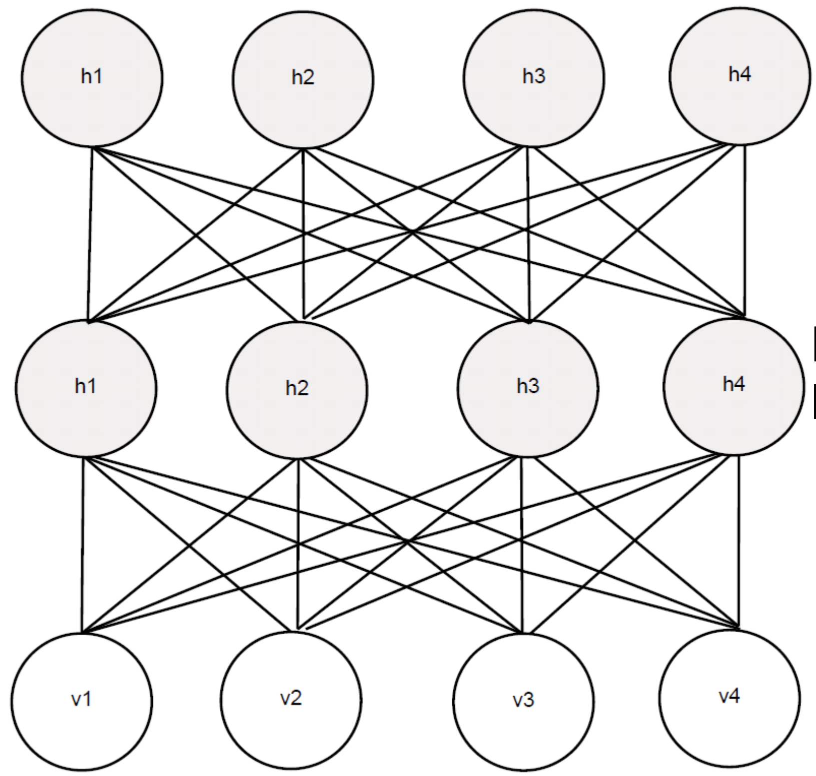
- Deep Networks used in variety of NLP and Speech processing tasks

- [Colbert and Weston, 2008] Tagging, Chunking
  - Words into features
- [Mohamed et. al, 2009] ASR
  - Phone recognition
- [Dealaers et. al, 2007] Machine Transliteration

Deep Networks

$$p(\mathrm{v},\, \mathrm{h}^1, h^2, h^3, ..., h^l)$$

join distribution factored into conditionals across layers such as $\mathrm{p}(\mathrm{h}^1|h^2)$



Hidden Nodes

Visible Nodes

Conditional Distributions of Layers

- Conditionals are given by

$$p(h^k|h^{k+1}) = \prod_i p(h_i^k|h^k + 1)$$

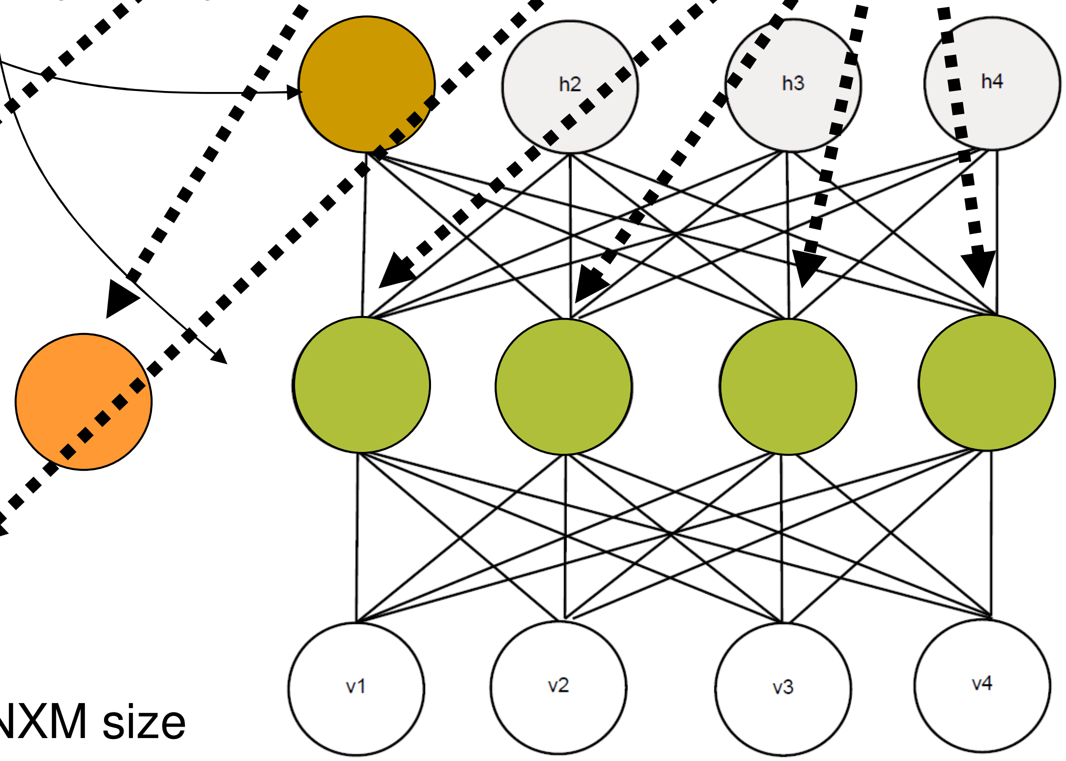
where

$$\mathrm{p}(\mathrm{h}_i^k|h^k + 1) = sig(b_i^k + \sum_j W_{ij}^k h_j^{k+1})$$

Conditional Distribution per Node

$$\mathrm{p}(\mathrm{h}_i^k | h^k + 1) = sig(b_i^k + \textstyle\sum_j W_{ij}^k h_j^{k+1})$$

- This is basically saying



Sigmoid function

$$\mathrm{W}_{ik}$$

Weight matrix if NXM size

Reference

- [1] Duda, Hart, and Stock, “Pattern Classification”