Backpropagation

LING 572

(Some slides are borrowed from from Yejin Choi's CSE 517)

Outline

- Partial derivative, gradient, and derivative chain rule
- Stochastic gradient descent (SGD)
- Backpropagation

Derivatives

- http://en.wikipedia.org/wiki/Derivative
- The derivative of the function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value).

- E.g. $f(x) = 5x^4+2x^3+5$
- $= 20x^3+6x^2$

Partial Derivatives

- http://en.wikipedia.org/wiki/Partial_Derivative
- A partial derivative of a function of several variables measures its derivative with respect one of those variables, with the others held constant.

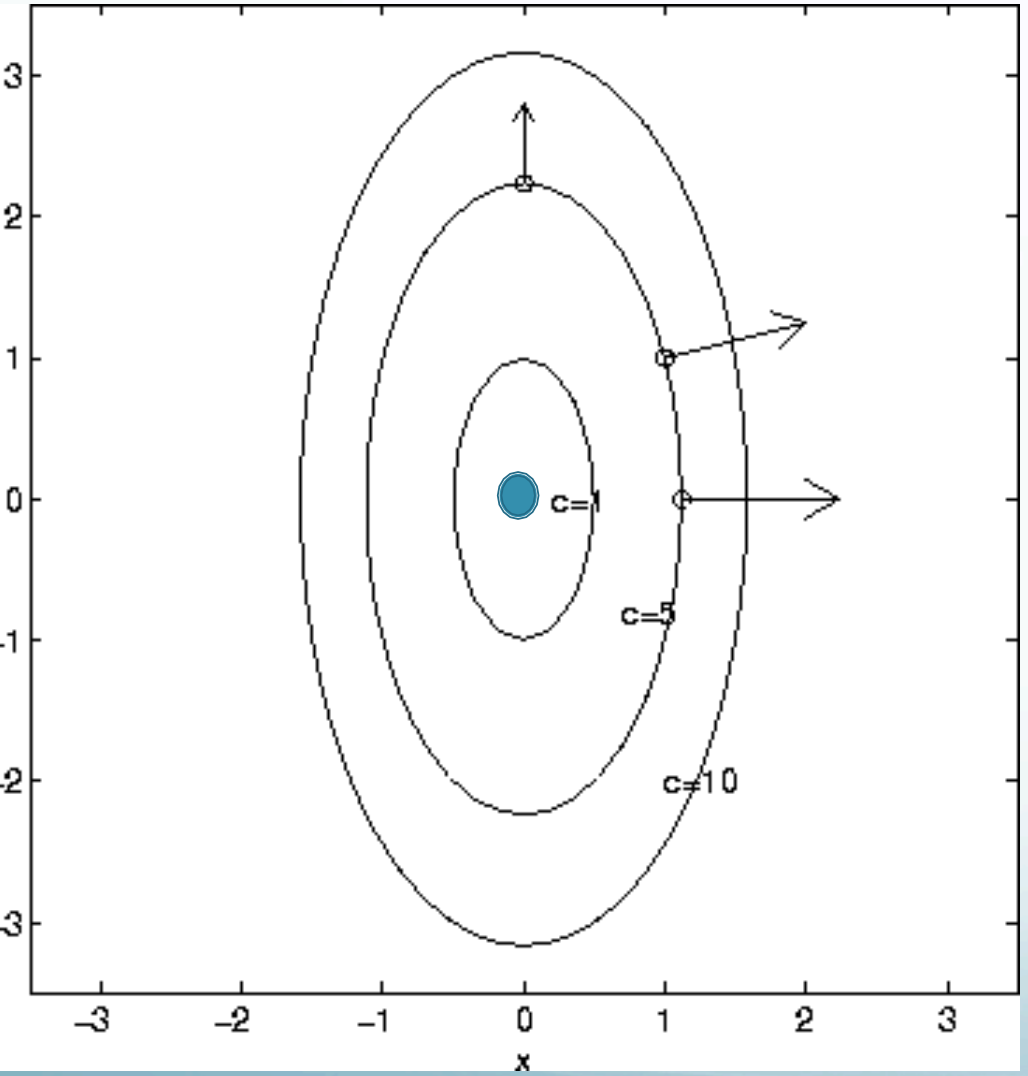
- E.g. $f(x,y) = 10x^3y^2+5xy^3+4x+y$
- $df/dx = 30x^2y^2+5y^3+4$
- $df/dy = 20x^3y+15xy^2+1$

Gradient

- The gradient of a function $f(x_1,x_2,\ldots x_n)$ is a vector function:
  - $<df/dx_1,df/dx_2,\ldots,df/dx_n>$

$$f(x,y) = 4x^2 + y^2$$
$$gradf = <8x, 2y>$$

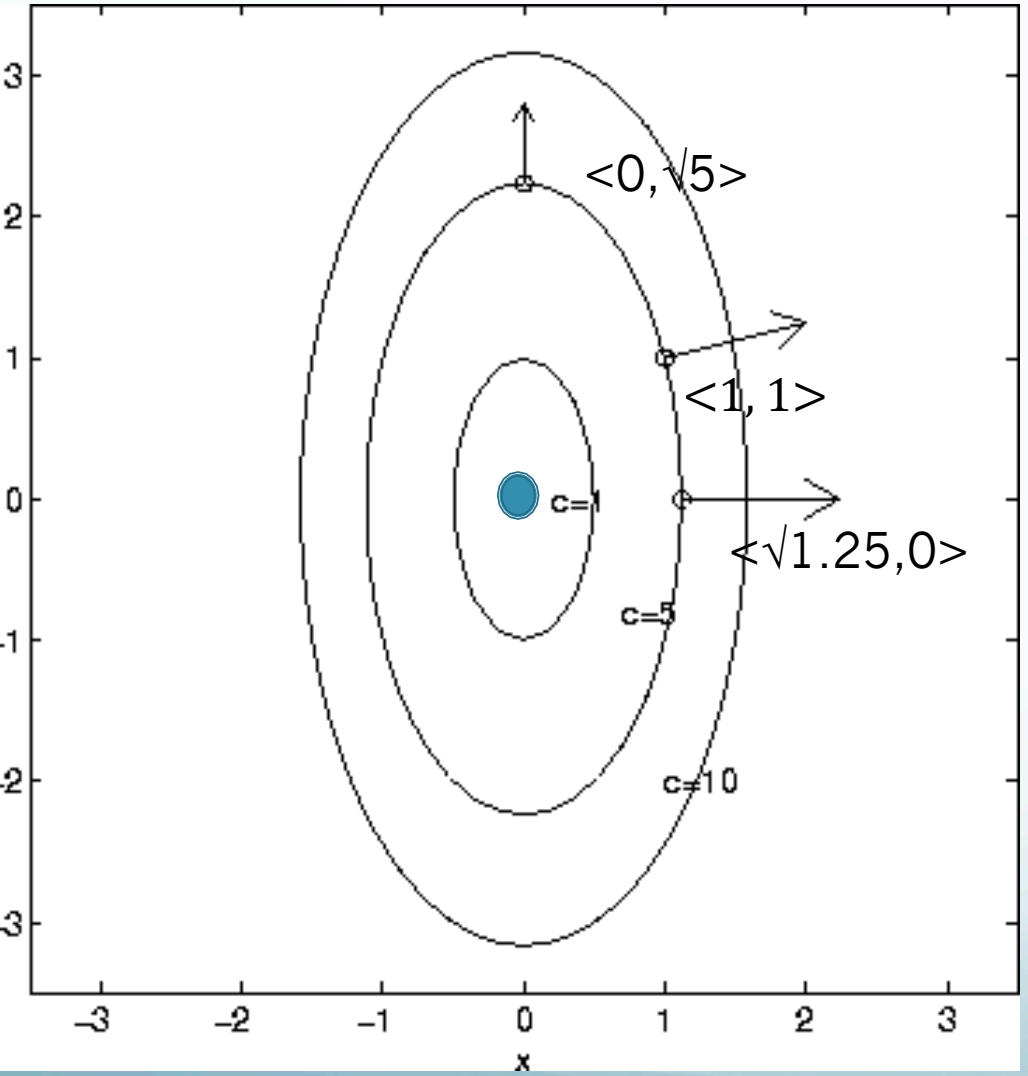
- The gradient vector at <x,y> is normal to level curve through (x,y)
- The gradient vector points in the direction of greatest rate of increase of the function



$$f(x,y) = 4x^2 + y^2$$

$$gradf = <8x, 2y>$$

Level curve: $4x^2+y^2=c$



$$f(x,y) = 4x^2 + y^2$$

$$gradf = <8x, 2y>$$

Level curve: $4x^2+y^2=c$

Derivative chain rule

$$z = f(y) \quad y = g(x)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

or

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$$

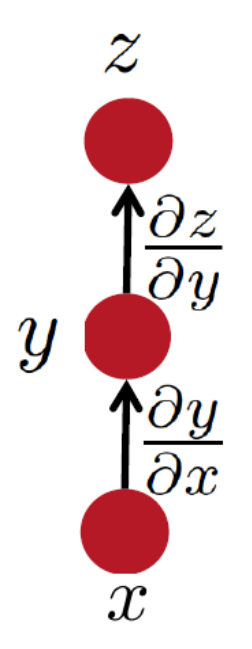
$z = (x^2+3)^4$

$dz/dx = 4(x^2+3)^3 * 2x$

$z = y^4 \quad y=x^2+3$

$dz/dy = 4y^3 \quad dy/dx=2x$

Simple chain rule



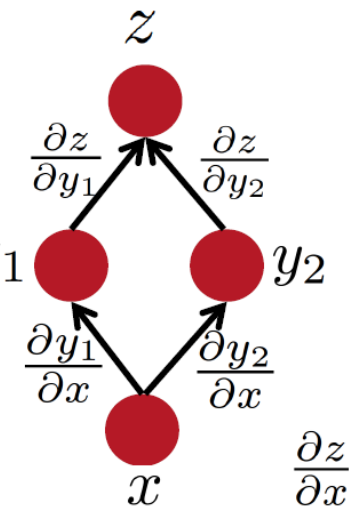
$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$

$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$

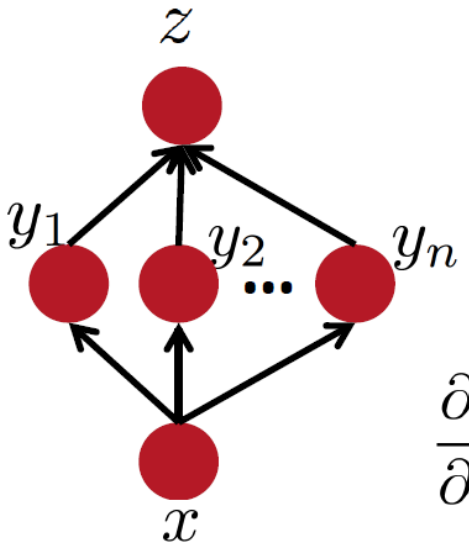
$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Multiple paths chain rule

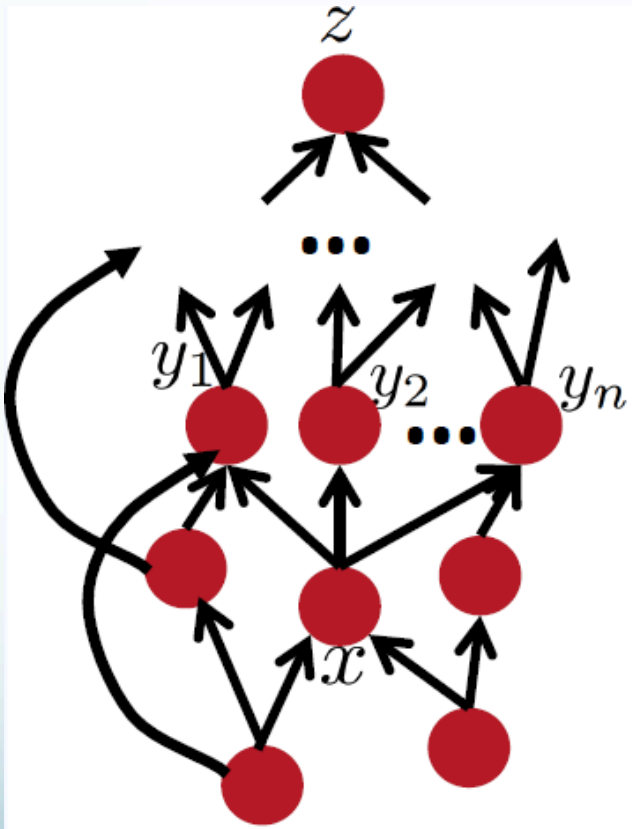


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

Chain rule in flow graph



Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

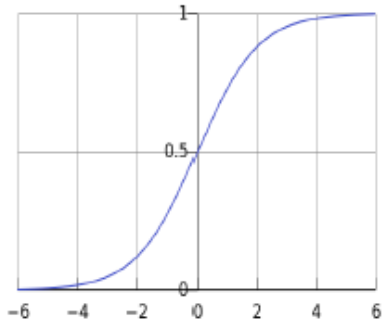
$\{y_1, y_2, \cdots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Common activation functions

logistic ("sigmoid")

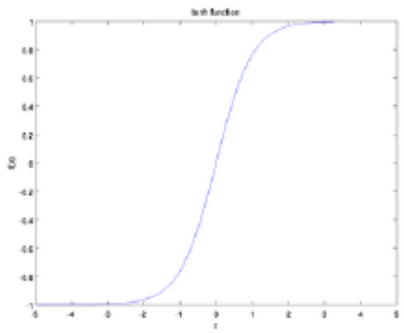
$$f(z) = \frac{1}{1 + \exp(-z)}.$$



$$f'(z) = f(z)(1 - f(z))$$

tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



$$f'(z) = 1 - f(z)^2$$

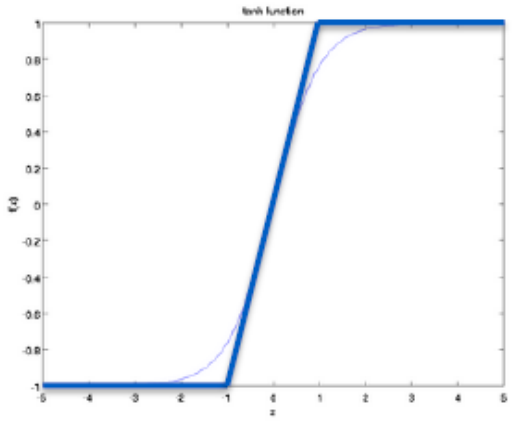
tanh is just a rescaled and shifted sigmoid $\tanh(z) = 2\text{logistic}(2z) - 1$

tanh is what is most used and often performs best for deep nets

Other choices

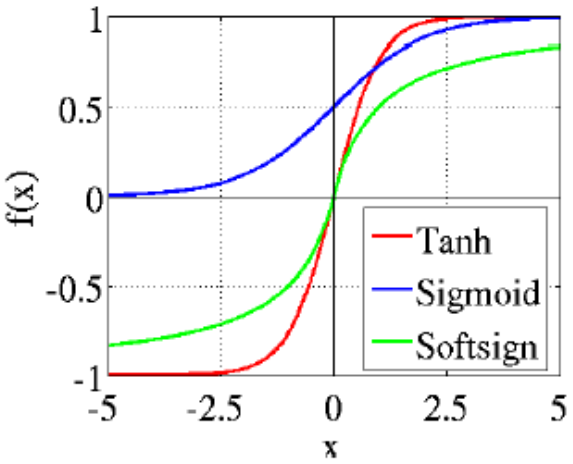
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



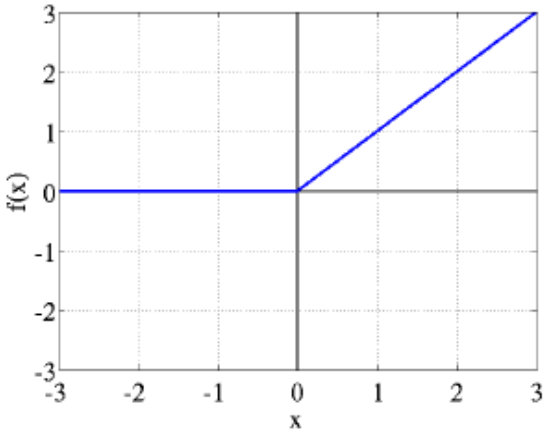
soft sign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$



rectifier

$$\text{rect}(z) = \max(z, 0)$$



Outline

- Partial derivative, gradient, and derivative chain rule
- Stochastic gradient descent (SGD)
- Backpropagation

Gradient descent

- See Chapter 1 of NN book, P23-P55

- Objective function: e.g., mean squared error (MSE)

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

- Goal: find w and b to minimize C(w, b)

- Approach: adjust w and b to make C(w, b) in each iteration

- Question: how do we adjust w and b to make C(w, b) smaller?

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2.$$

Let $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$ Then $\Delta C \approx \nabla C \cdot \Delta v$

Choose $\Delta v = -\eta \nabla C$ Then $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$ ≤ 0

Gradient descent: $v \rightarrow v' = v - \eta \nabla C$

When there are more than two variables

Let $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$

Let $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T$

Then $\Delta C \approx \nabla C \cdot \Delta v$

Choose $\Delta v = -\eta \nabla C$

$v \rightarrow v' = v - \eta \nabla C$

Apply this to NN

Let $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$

Let $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T$

Then $\Delta C \approx \nabla C \cdot \Delta v$

Choose $\Delta v = -\eta \nabla C$

$v \to v' = v - \eta \nabla C$

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

When there are multiple training instances

$$C = \frac{1}{n}\sum_x C_x$$

Where C_x is a training instance and we use a loss function such as

$$C_x \equiv \frac{\|y(x)-a\|^2}{2}$$

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x$$

Stochastic Gradient Descent (SGD)

- Instead of the whole training set, use a randomly selected sample (aka mini-batch).

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

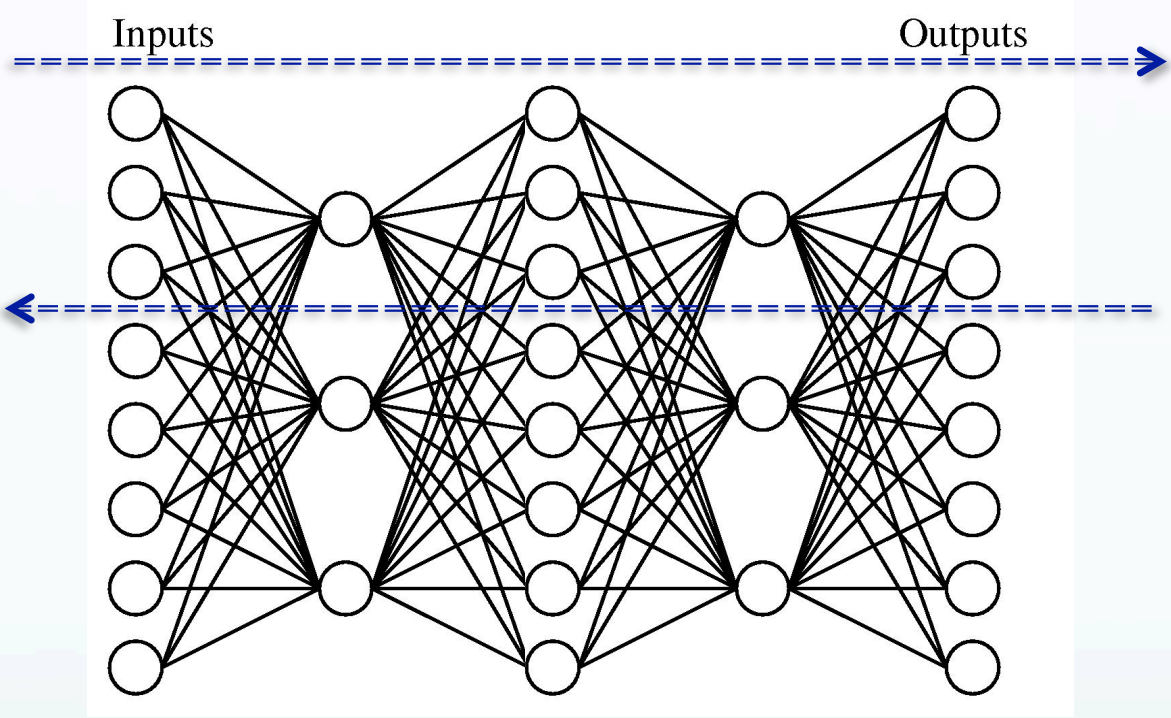
$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

SGD algorithm

```
for j in xrange(epochs):
    random.shuffle(training_data)
    mini_batches = [
        training_data[k:k+mini_batch_size]
        for k in xrange(0, n, mini_batch_size)]
    for mini_batch in mini_batches:
        self.update_mini_batch(mini_batch, eta)
```

Forward



Gradient

Learning: BackPropagation

Next 10 slides on back propagation are adapted from Andrew Rosenberg

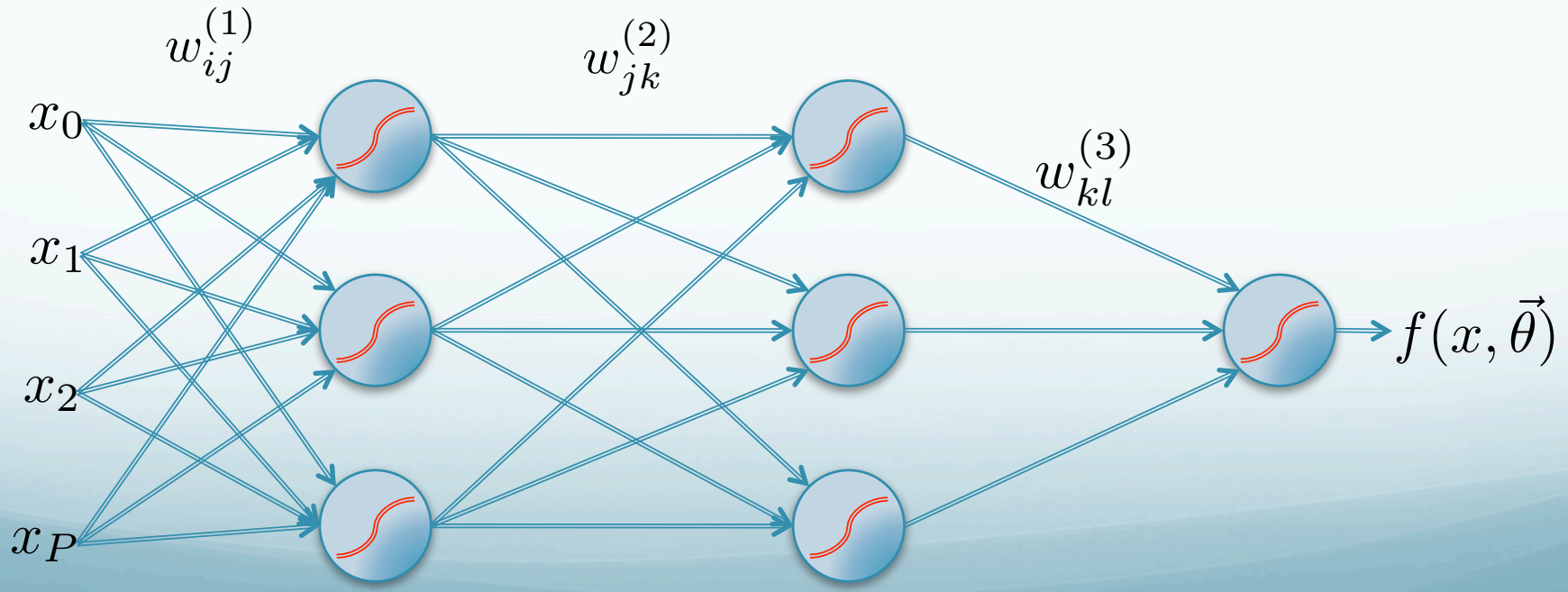
Error Backpropagation

- Model parameters:

$$\vec{\theta} = \{w_{ij}^{(1)}, w_{jk}^{(2)}, w_{kl}^{(3)}\}$$

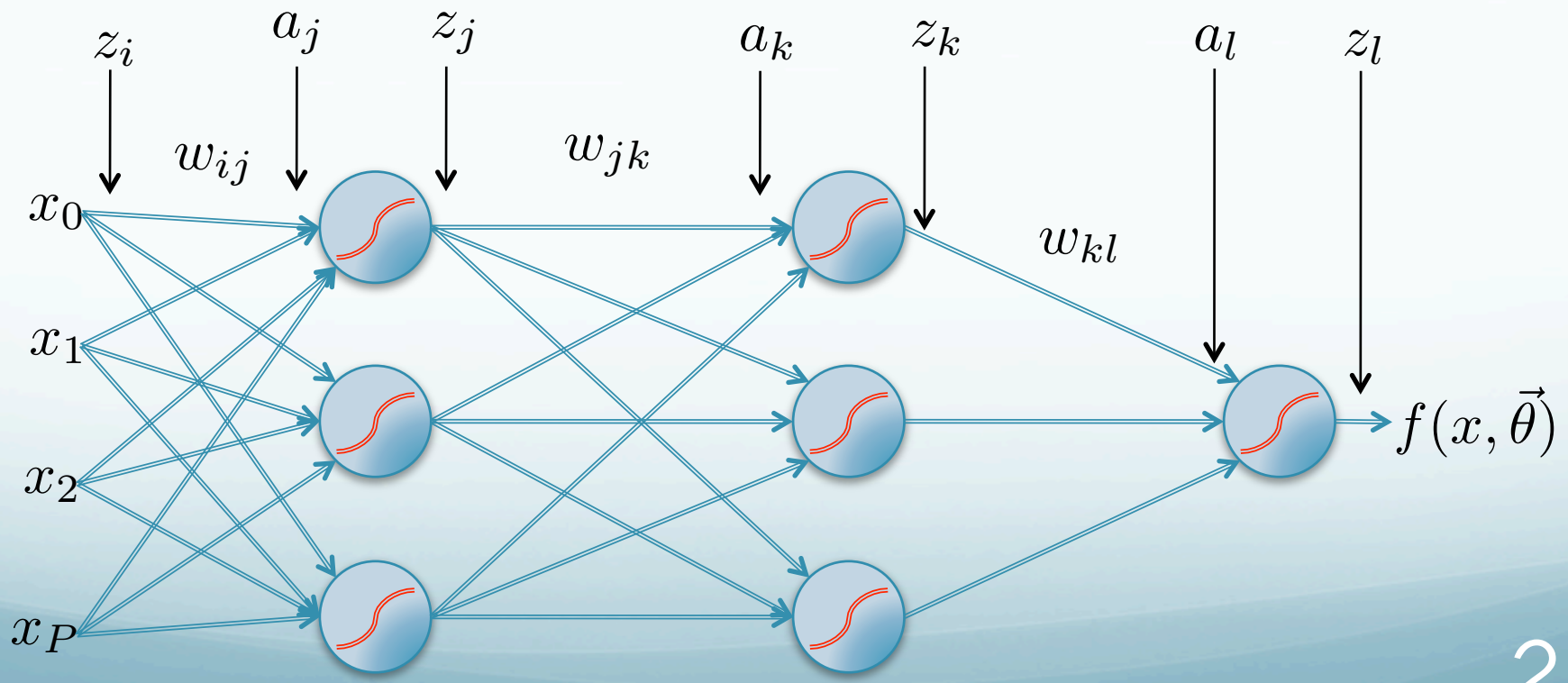
for brevity:

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



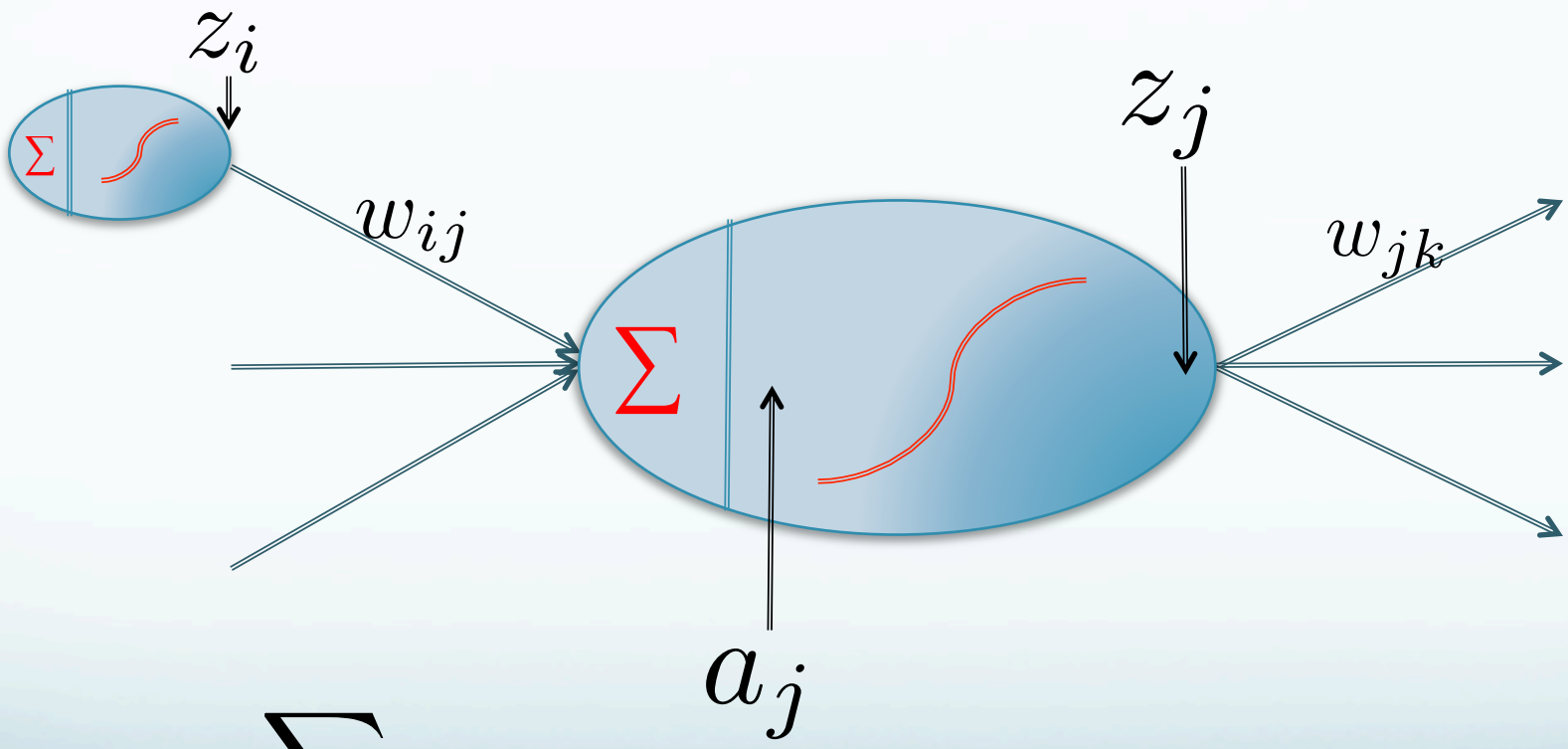
Error Backpropagation

- Model parameters: $\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$
- *Let a and z be the input and output of each node*



Error Backpropagation

$$z_j = g(a_j)$$

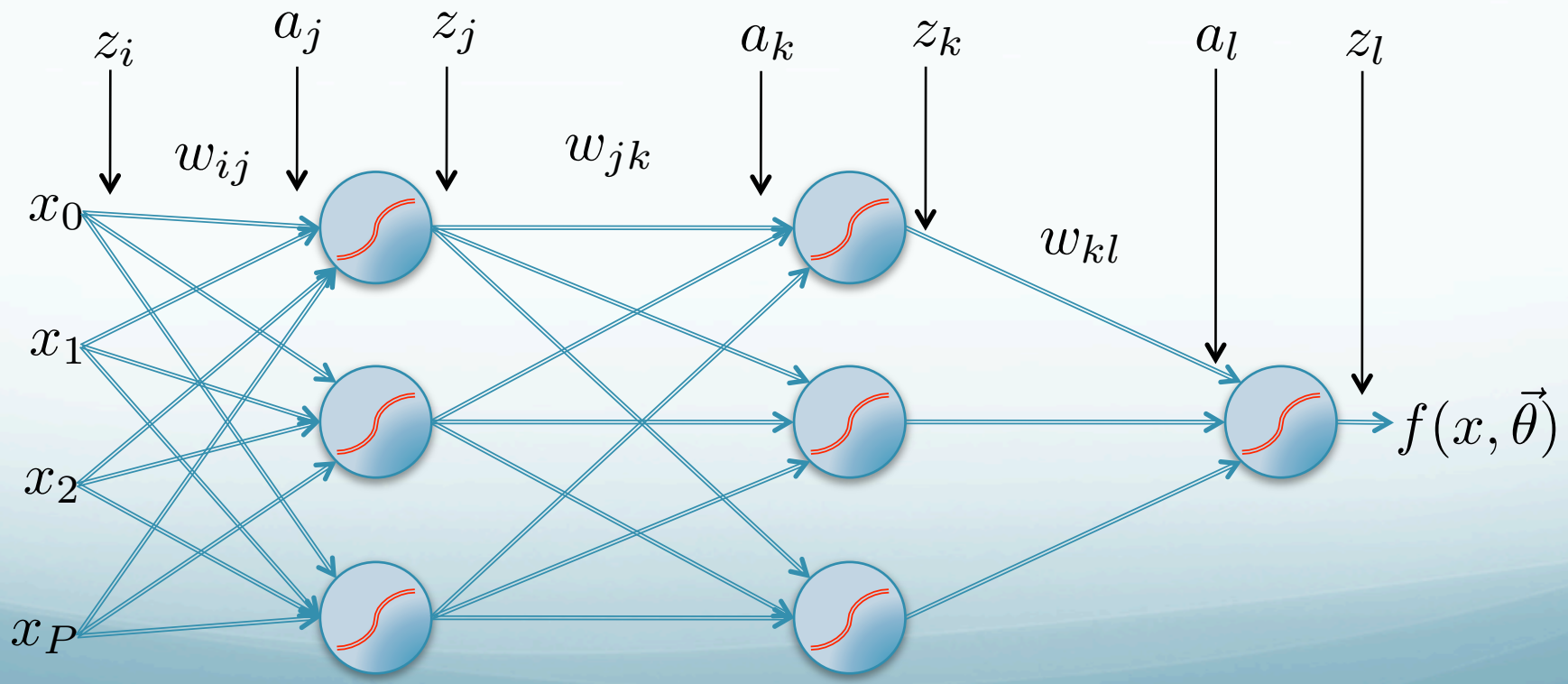


$$a_j = \sum_i w_{ij} z_i$$

- *Let a and z be the input and output of each node*

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \boxed{} \qquad a_l = \boxed{}$$

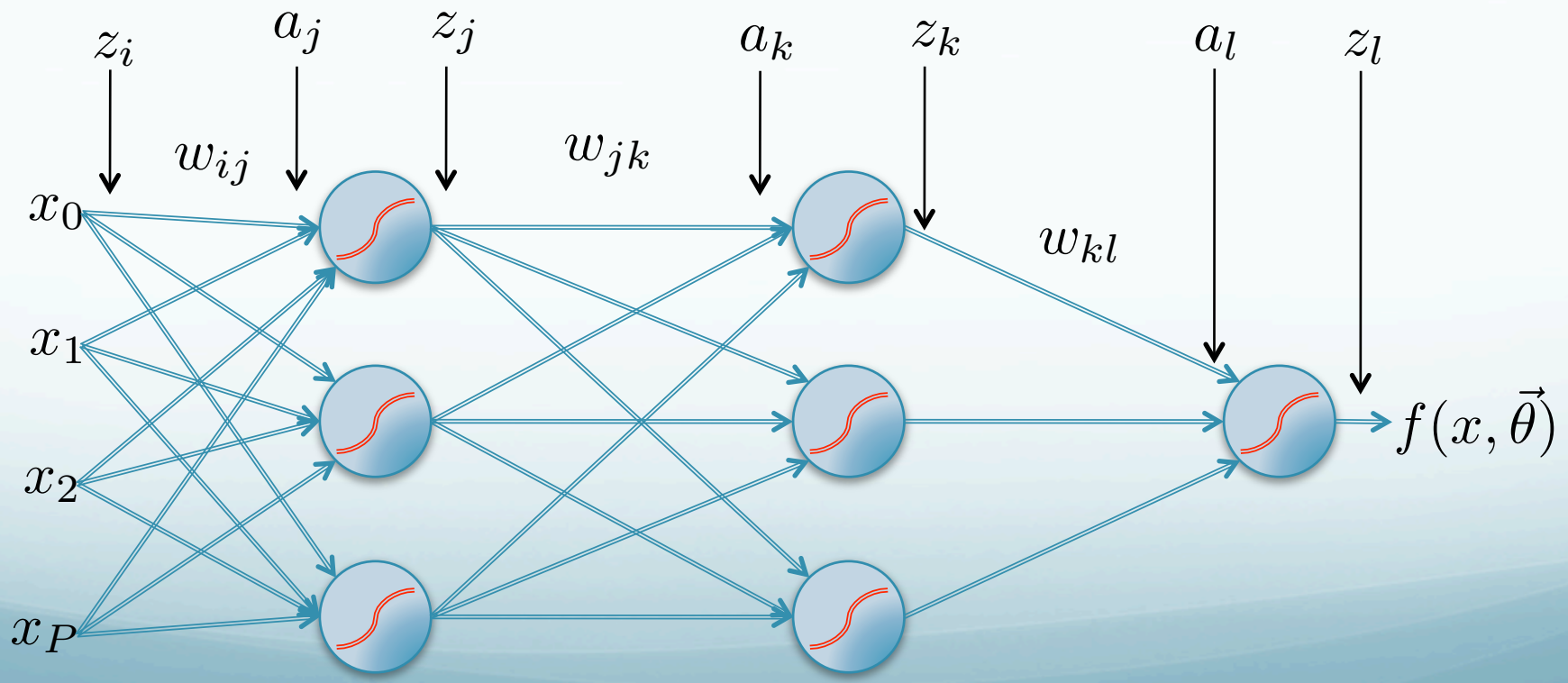
$$z_j = g(a_j) \qquad z_k = \boxed{} \qquad z_l = \boxed{}$$



- *Let a and z be the input and output of each node*

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

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



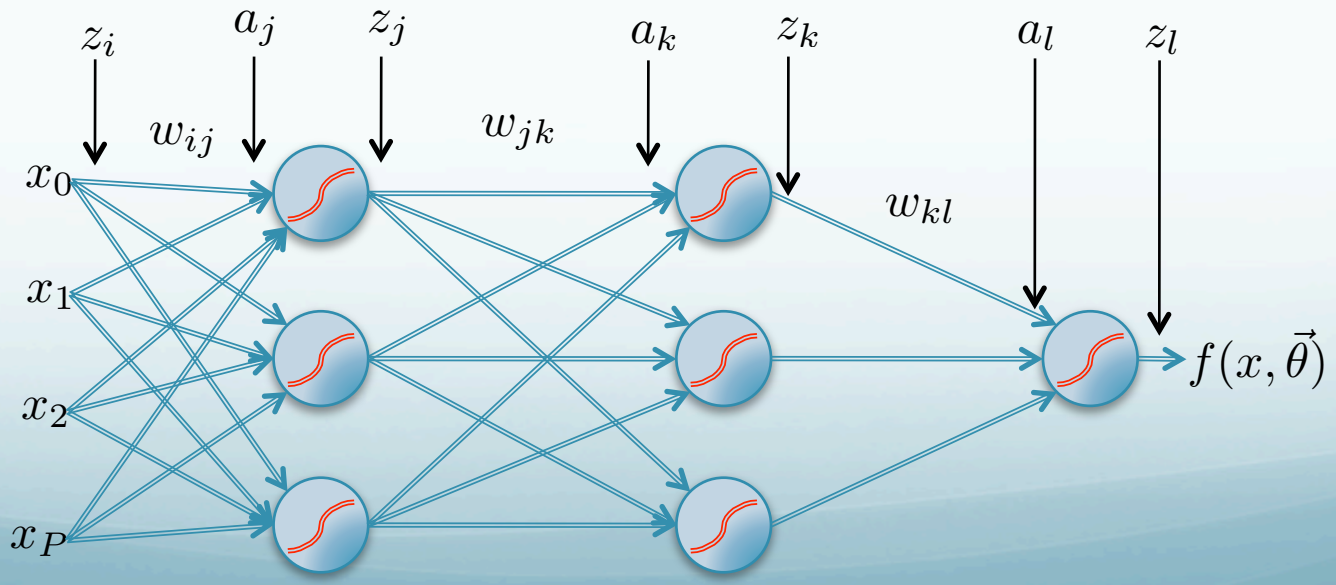
Training: minimize loss

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n))$$

Empirical Risk Function

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\quad g\left(\quad g\left(\quad x_{n,i}\right)\right)\right)\right)^2$$



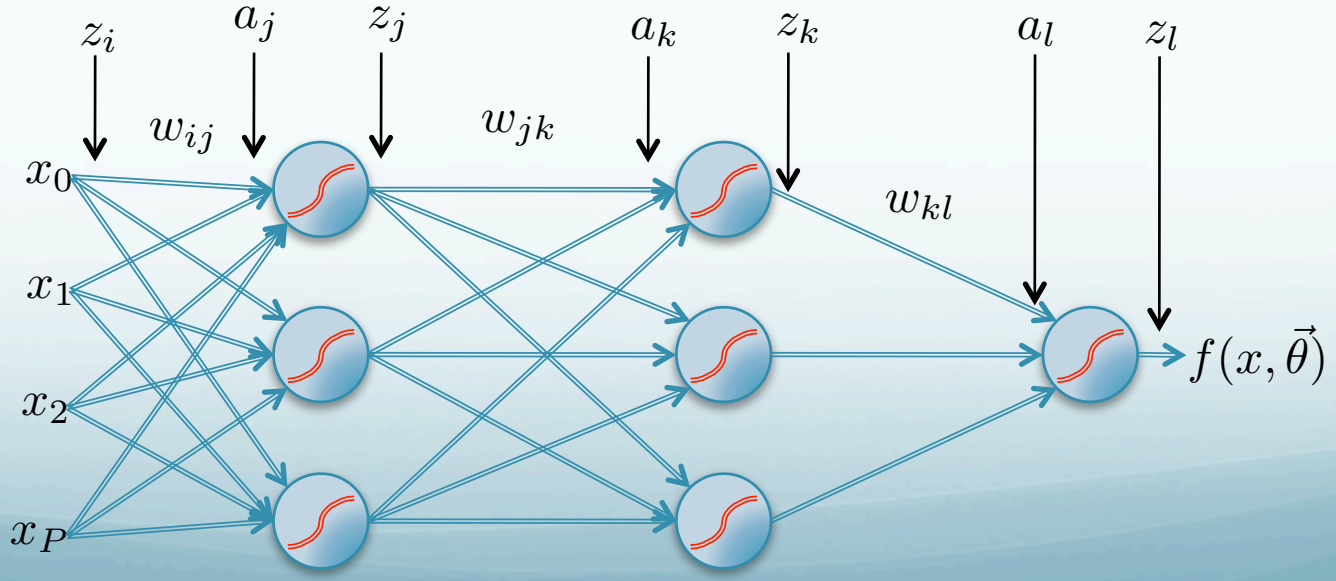
Training: minimize loss

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n))$$

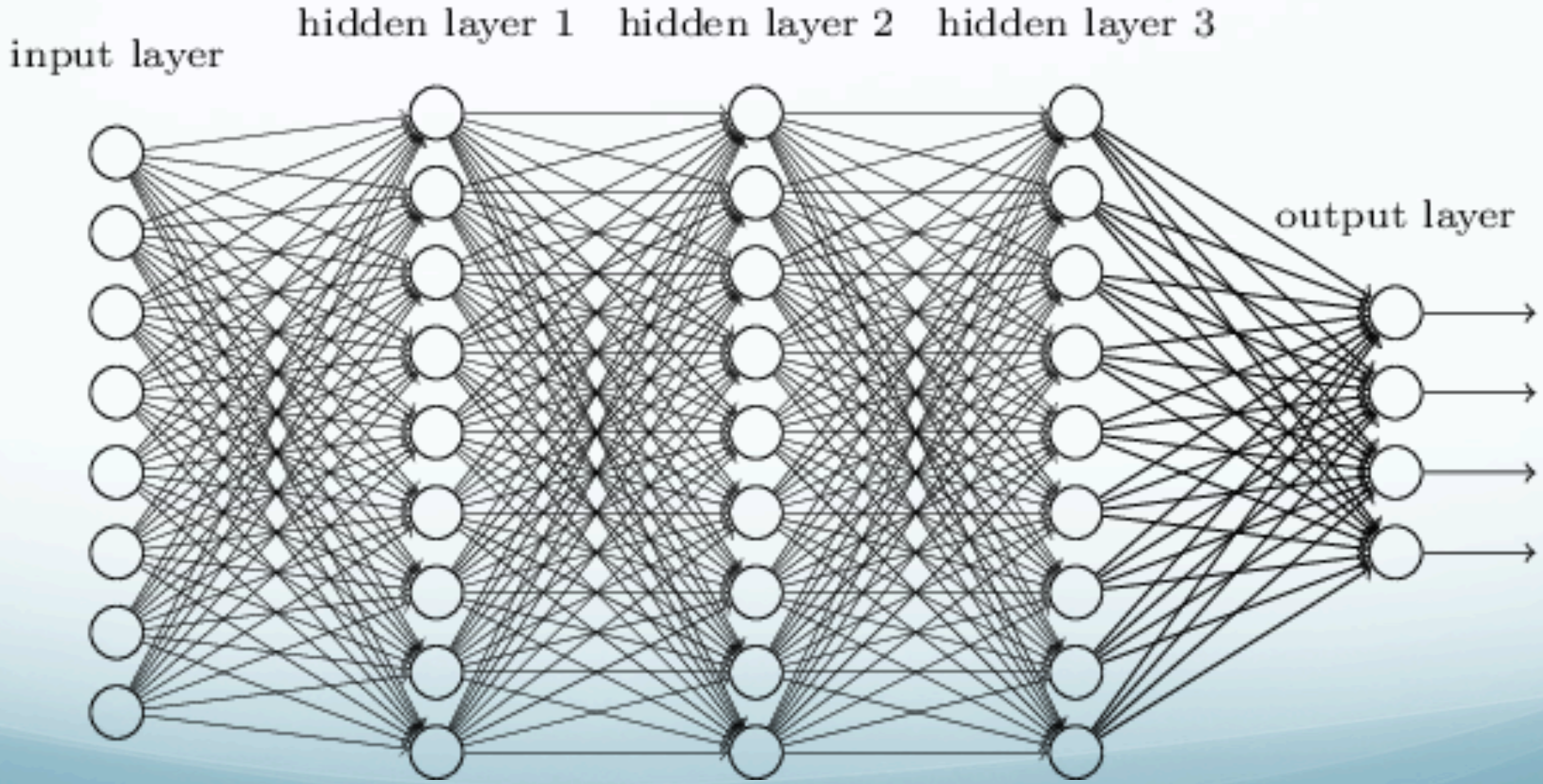
Empirical Risk Function

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$= \frac{1}{N}\sum_{n=0}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{kl} g\left(\sum_j w_{jk} g\left(\sum_i w_{ij} x_{n,i}\right)\right)\right)\right)^2$$



Taking Partial Derivatives...



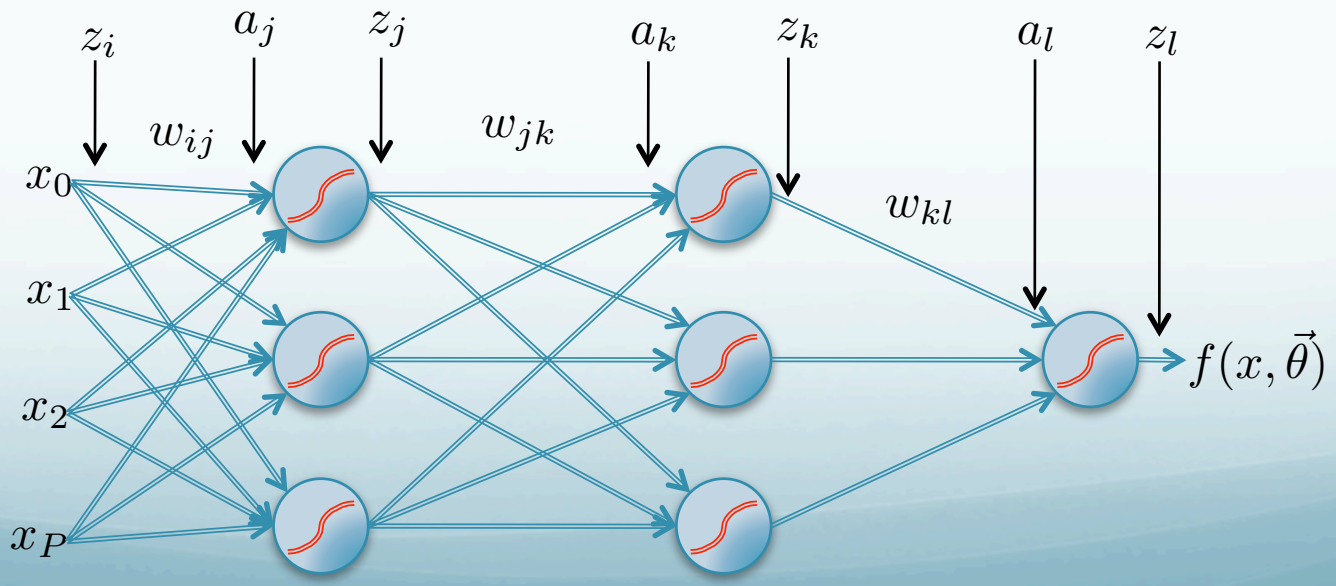
Error Backpropagation

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



Error Backpropagation

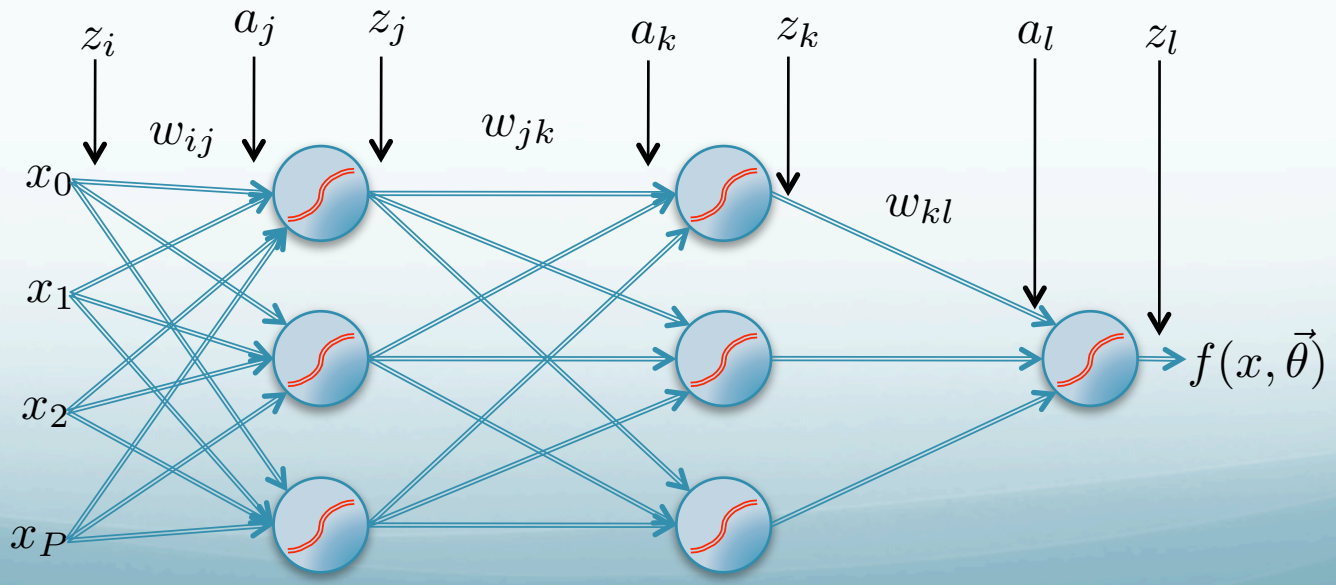
Optimize last layer weights w_{kl}

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

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

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

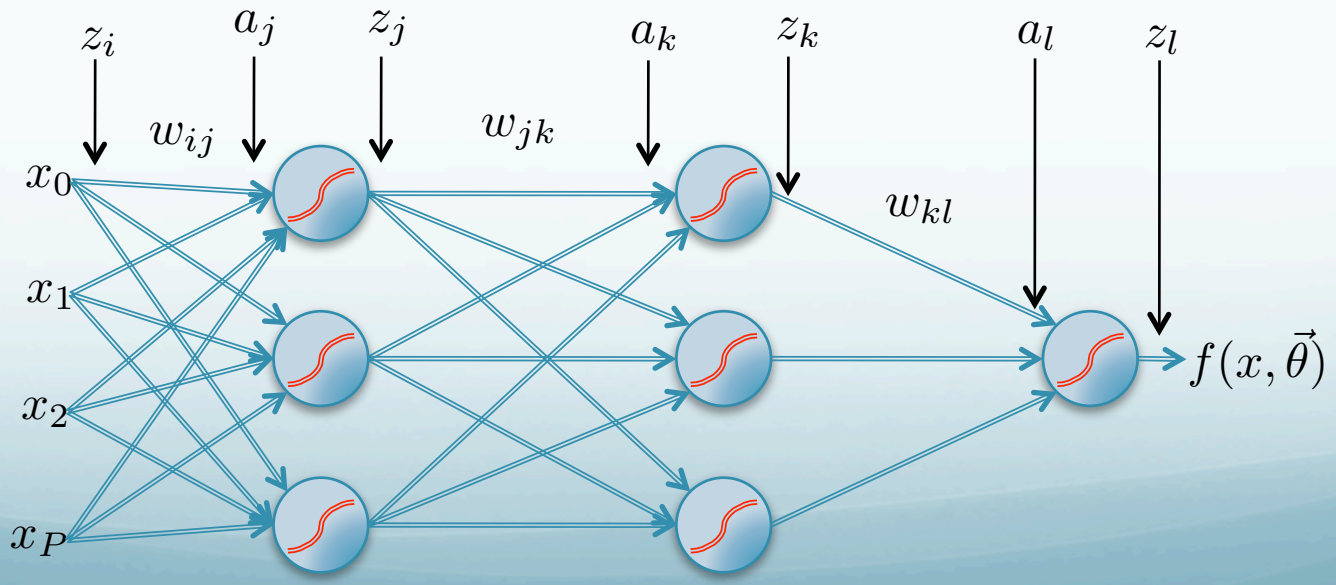
Optimize last layer weights w_{kl}

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

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

Calculus chain rule

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



Error Backpropagation

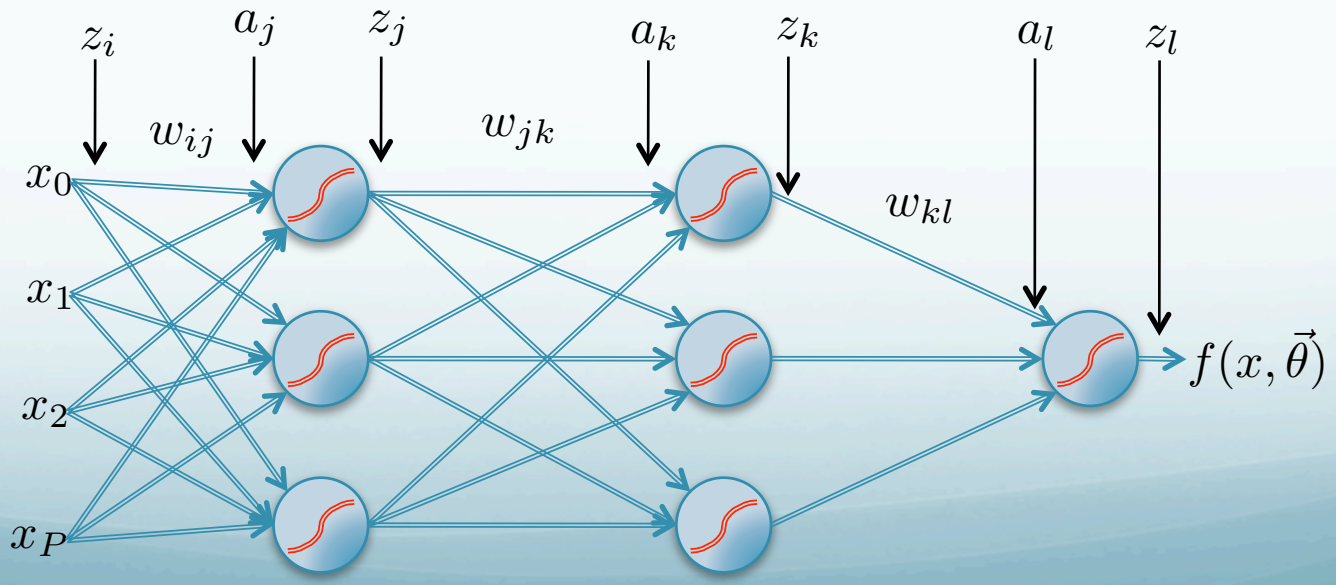
Optimize last layer weights w_{kl}

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

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

Calculus chain rule

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



Error Backpropagation

Optimize last layer weights w_{kl}

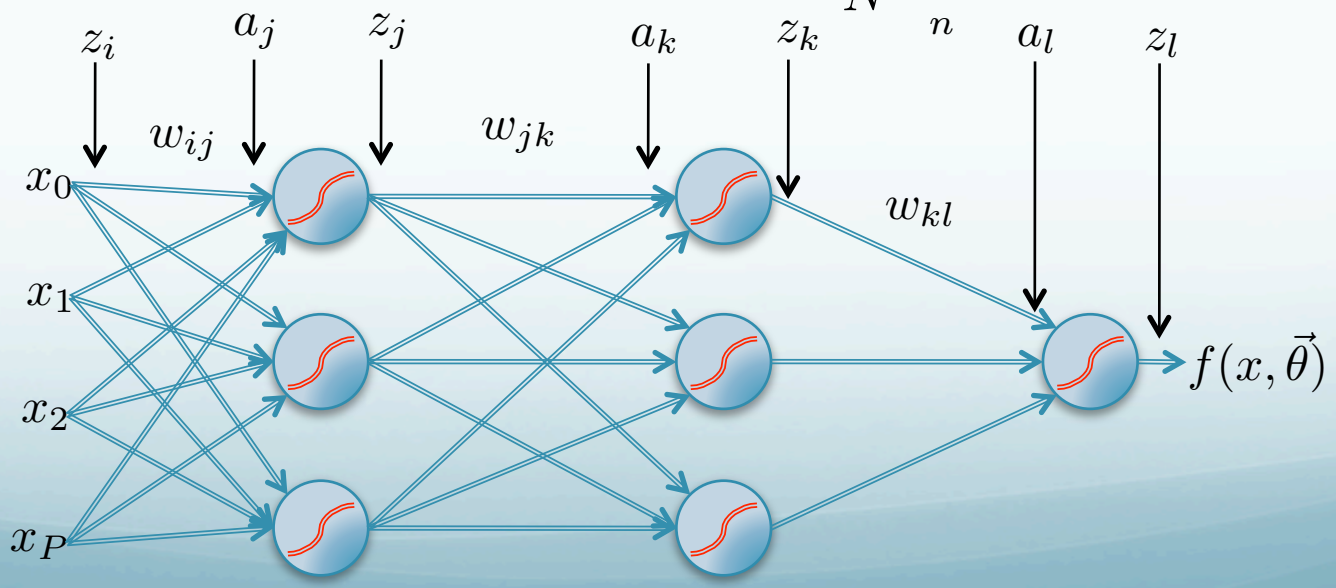
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

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

Calculus chain rule

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

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



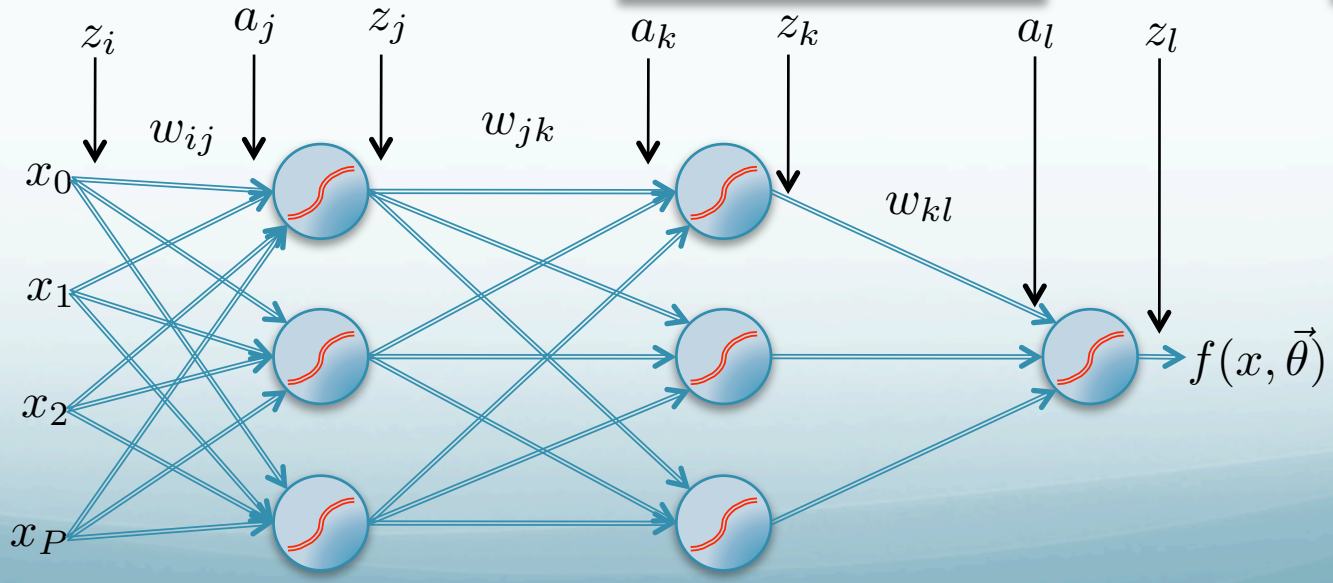
Error Backpropagation

Repeat for all previous layers

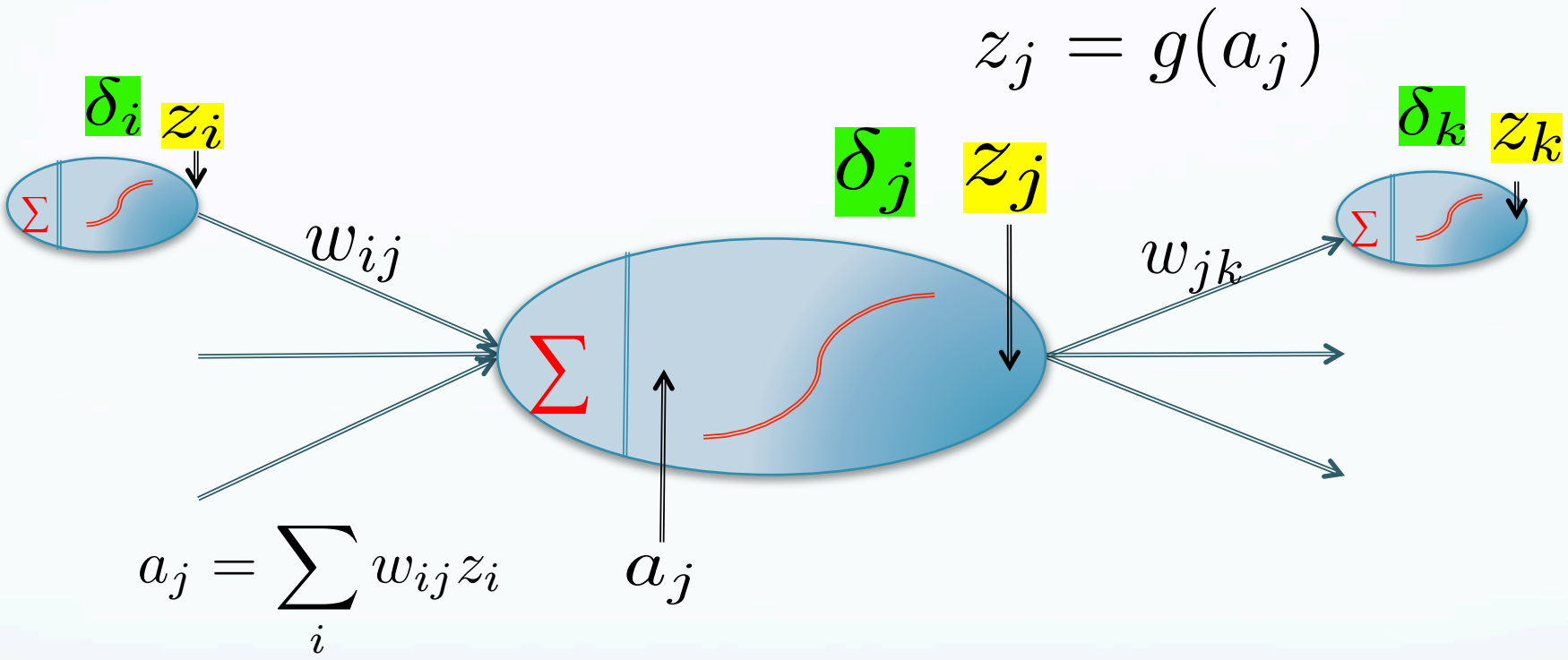
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

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

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



Backprop Recursion

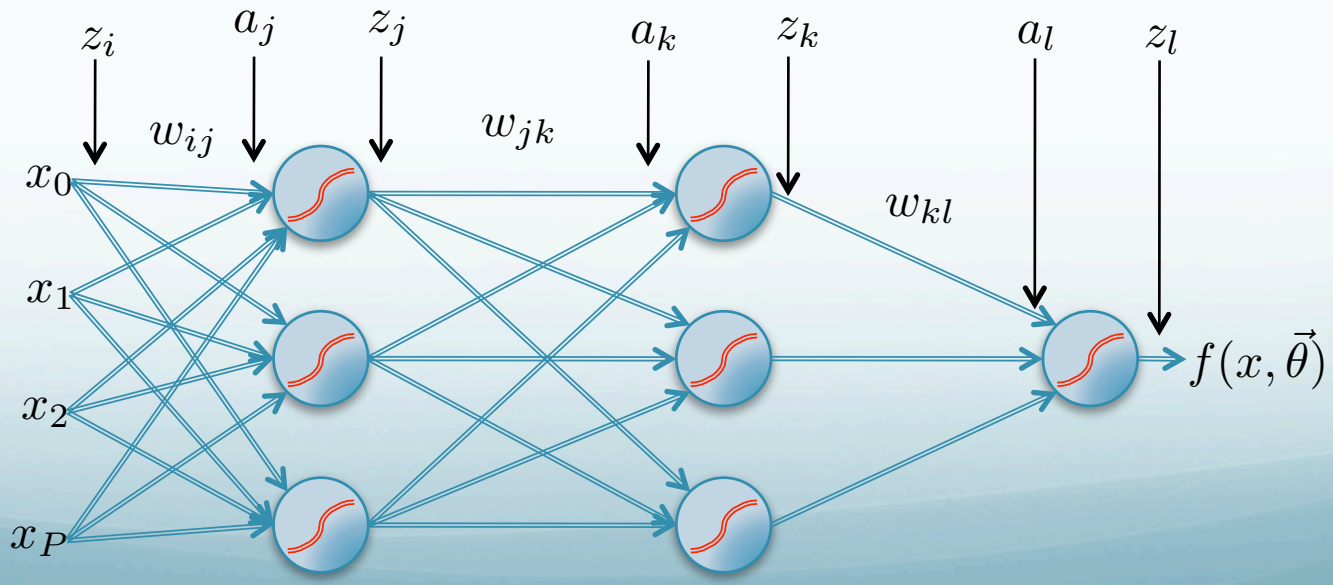


$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

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

Learning: Gradient Descent

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



Backpropagation

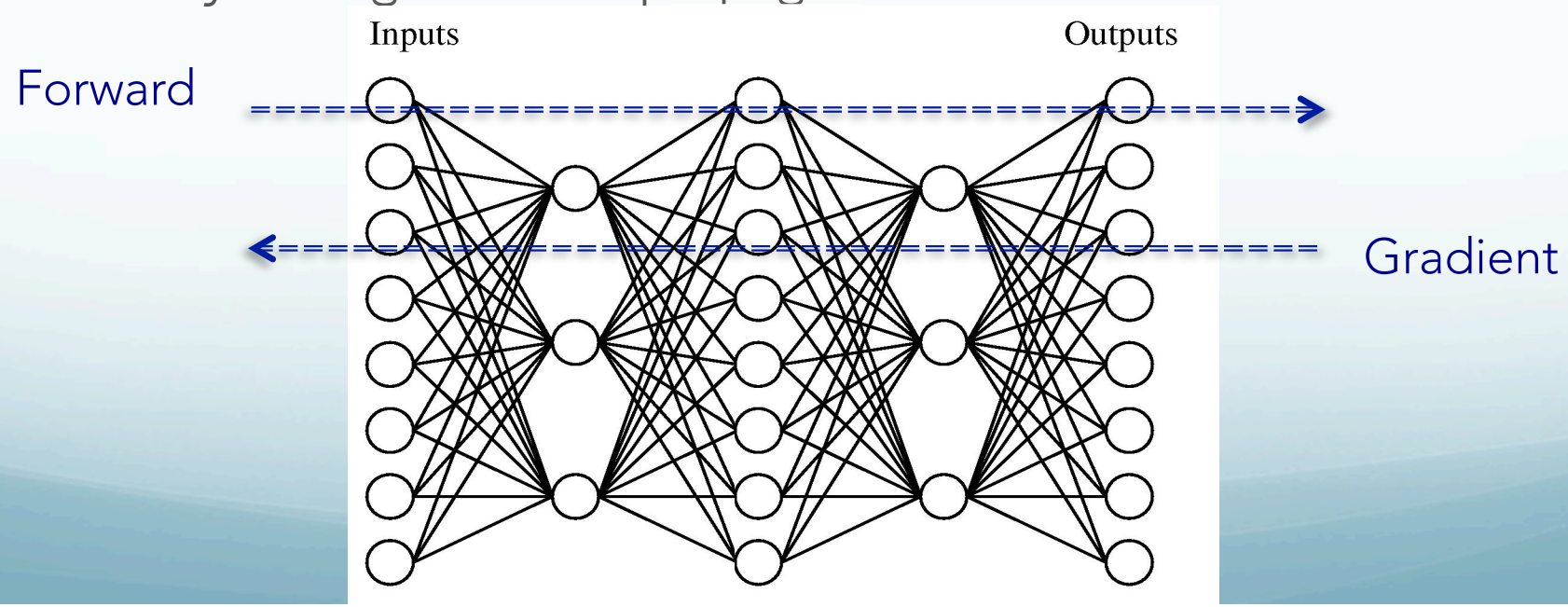
Starts with a forward sweep to compute all the intermediate function values z_i

Through backprop, computes the partial derivatives recursively δ_j $\frac{\partial R}{\partial w_{ij}}$

A form of dynamic programming

- Instead of considering exponentially many paths between a weight w_ij and the final loss (risk), store and reuse intermediate results.

A type of automatic differentiation. (there are other variants e.g., recursive differentiation only through forward propagation.

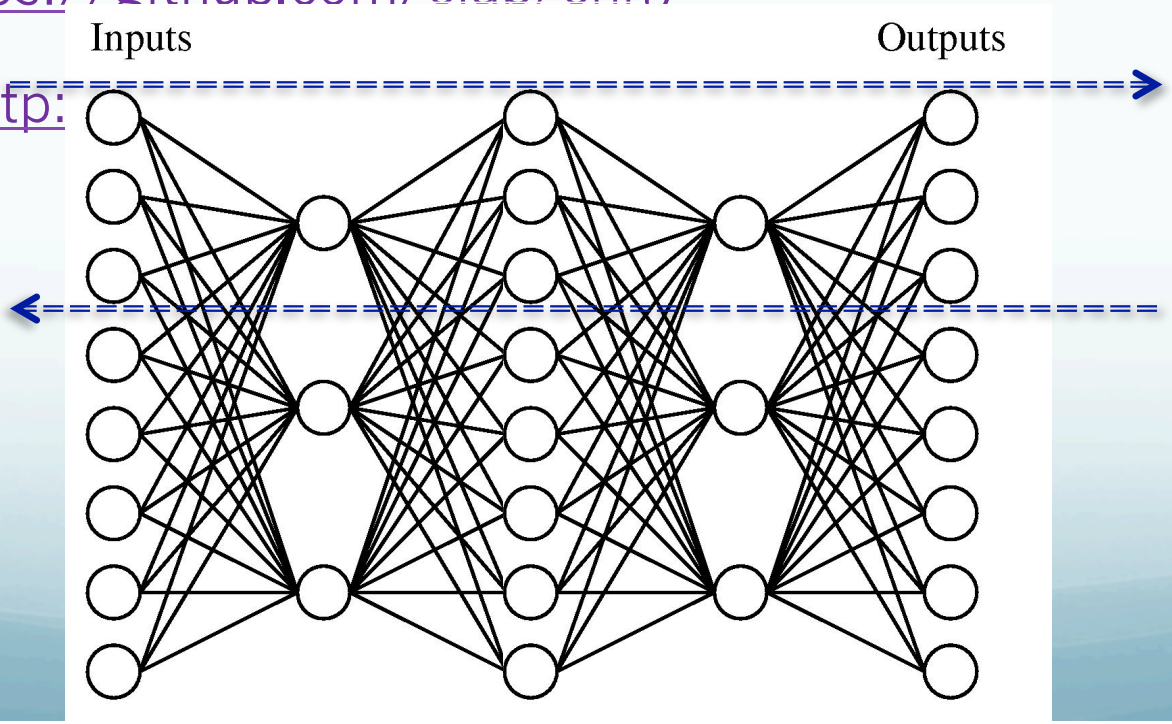


Backpropagation

- TensorFlow (https://www.tensorflow.org/)
- Torch (http://torch.ch/)
- Theano (http://deeplearning.net/software/theano/)
- CNTK (https://github.com/Microsoft/CNTK)
- cnn (https://github.com/clab/cnn)
- Caffe (http:

Primary Interface Language:

- Python
- Lua
- Python
- C++
- C++
- C++



Cross Entropy Loss (aka log loss, logistic loss)

- Cross Entropy $$H(p,q) = -\sum_y p(y)\ \log\ q(y)$$

  Predicted prob

  True prob

- Related quantities
  - Entropy $$H(p) = \sum_y p(y) \log\ p(y)$$
  - KL divergence (the distance between two distributions p and q)

$$D_{KL}(p||q) = \sum_y p(y)\ \log\ \frac{p(y)}{q(y)}$$

$$H(p,q) = E_p[-\log\ q] = H(p) + D_{KL}(p||q)$$

- Use Cross Entropy for models that should have more probabilistic flavor (e.g., language models)
- Use Mean Squared Error loss for models that focus on correct/ incorrect predictions

$$\text{MSE} = \frac{1}{2}(y - f(x))^2$$