University of Ljubljana, Faculty of Computer and Information Science

A brief revision of neural networks

Prof Dr Marko Robnik-Šikonja

Natural Language Processing, Edition 2024

Contents

- a gentle introduction to neural networks
- feed forward neural networks
- backpropagation
- convolutional neural networks
- attacks on neural networks

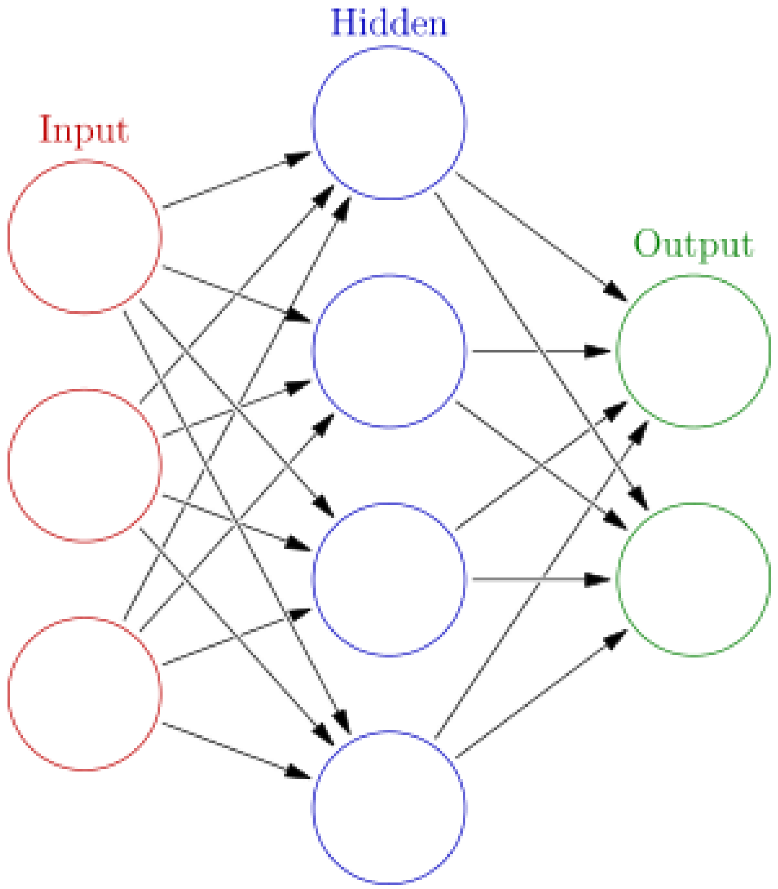
read Chapter 7 in Jurafsky & Martin, 3rd edition,

Sources

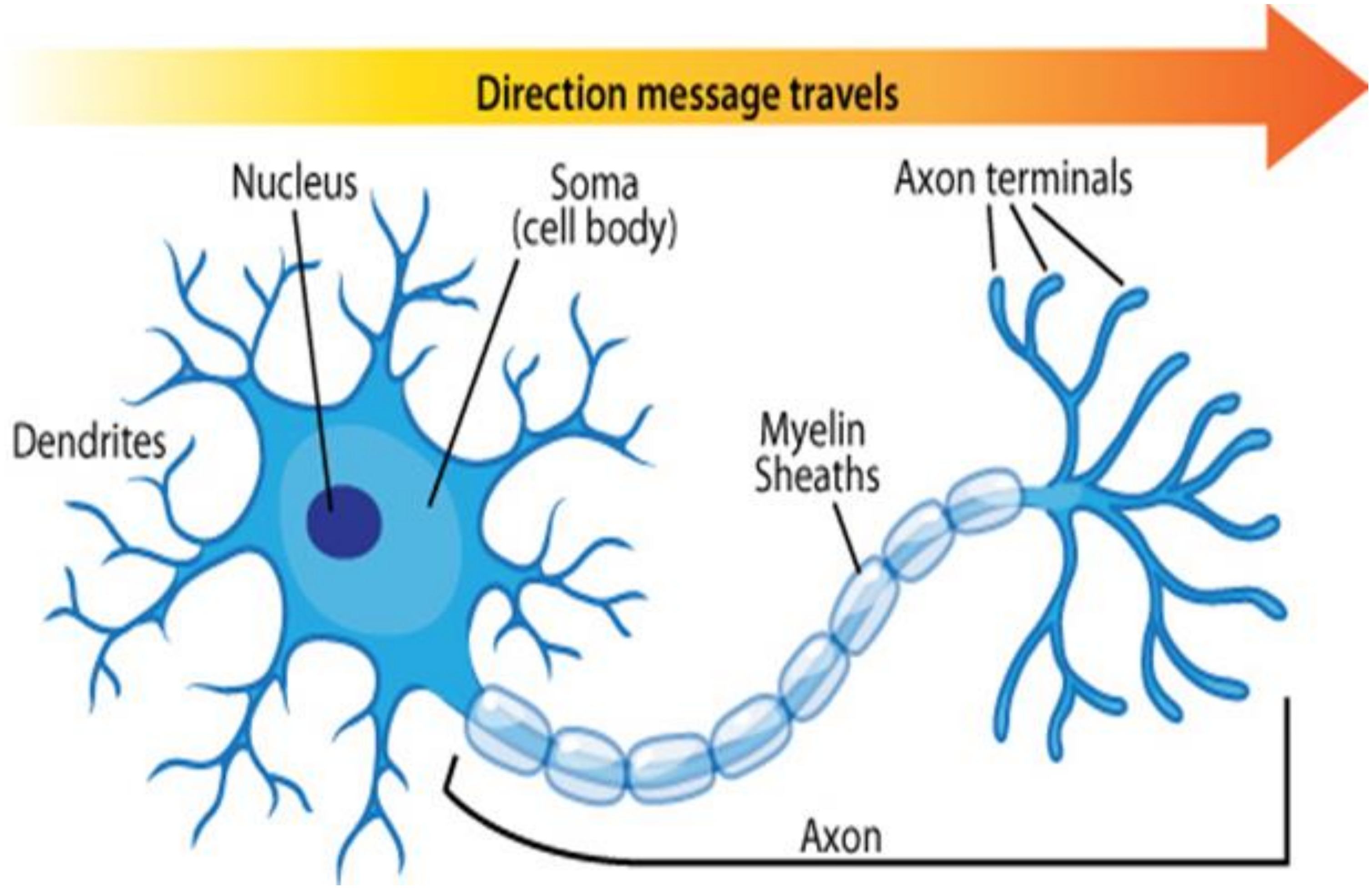
- Richard Socher: *Deep Learning for Natural Language Processing*. Coursera
- Ian Goodfellow and Yoshua Bengio and Aaron Courville: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org
- Yoav Goldberg: A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research* 57:345-420, 2016
- Keras library
- PyTorch

Artificial neural networks (ANN)

- universal function approximator
- intuition: neurons in successive layers encode useful features

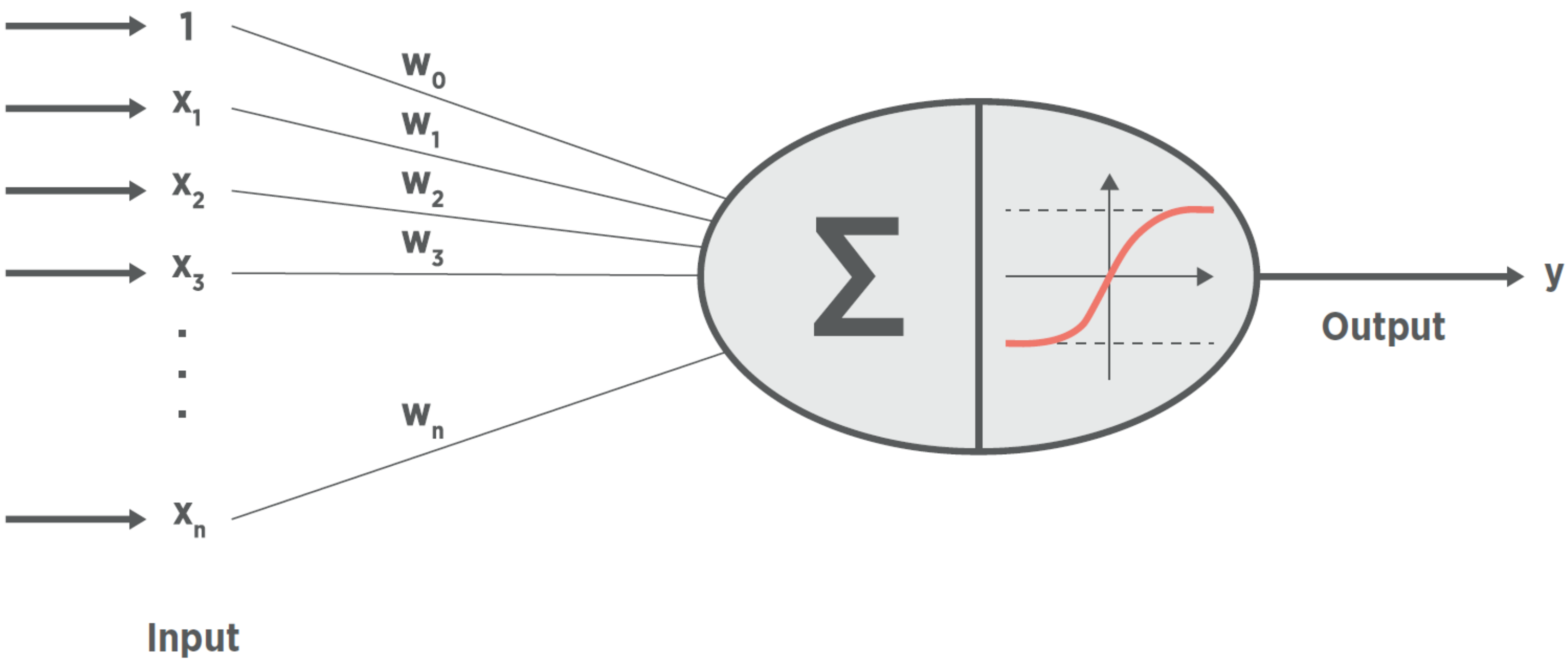


Artificial neural networks and brain analogy – a neuron



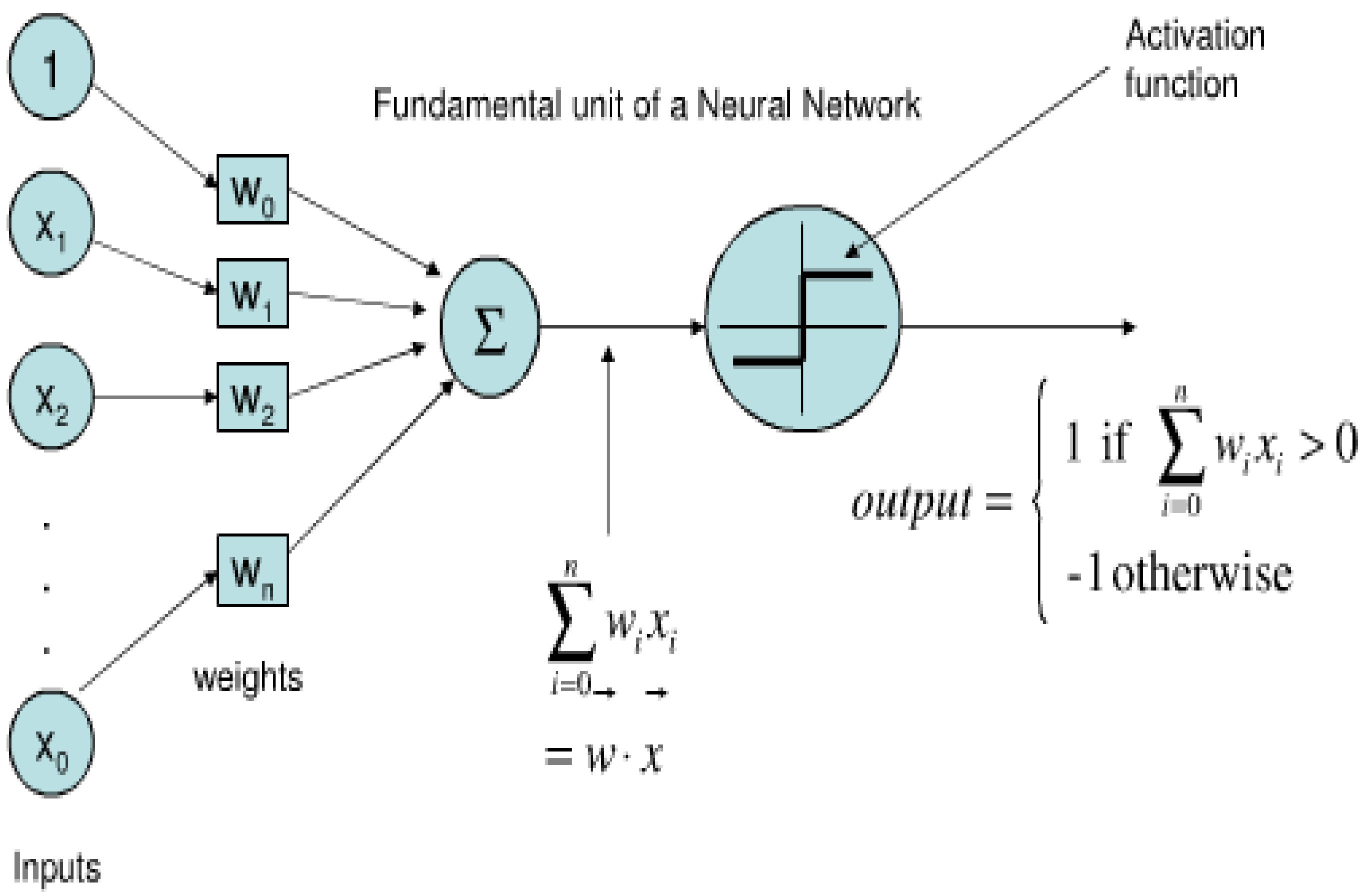
- more than a hundred types of neurons in brain

Artificial neuron



The brain analogy is far from realistic: a neuron cell is highly complex, and so are interconnections.

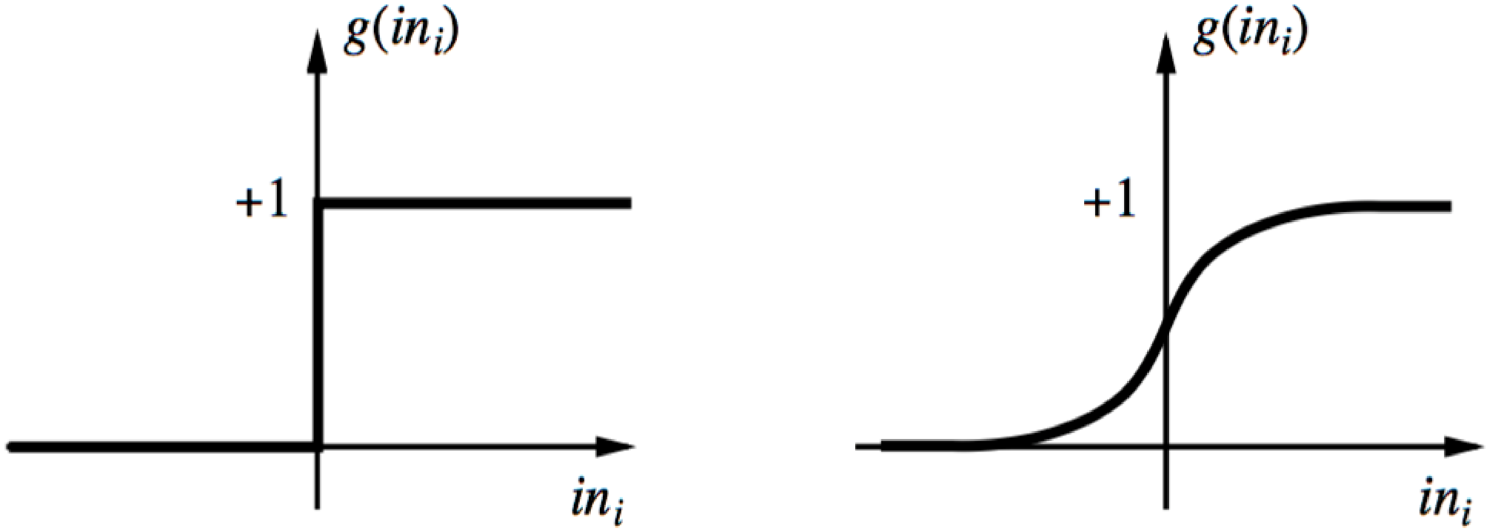
Perceptron



Activation
function
Fundamental unit of a Neural Network
1
x_1
x_2
x_0
w_0
w_1
w_2
w_n
weights
Inputs
Σ
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Activation functions

- examples: step function, sigmoid (logistic)



(a) (b)

Activation functions

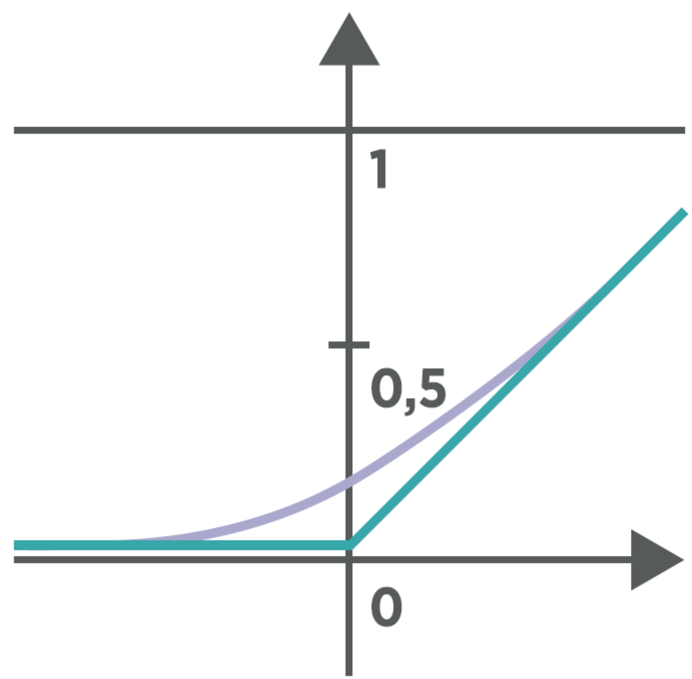
- ReLU (rectified linear unit)

  $f(x) = \max(0, x)$

- softplus / approximation of ReLU with continuous derivation

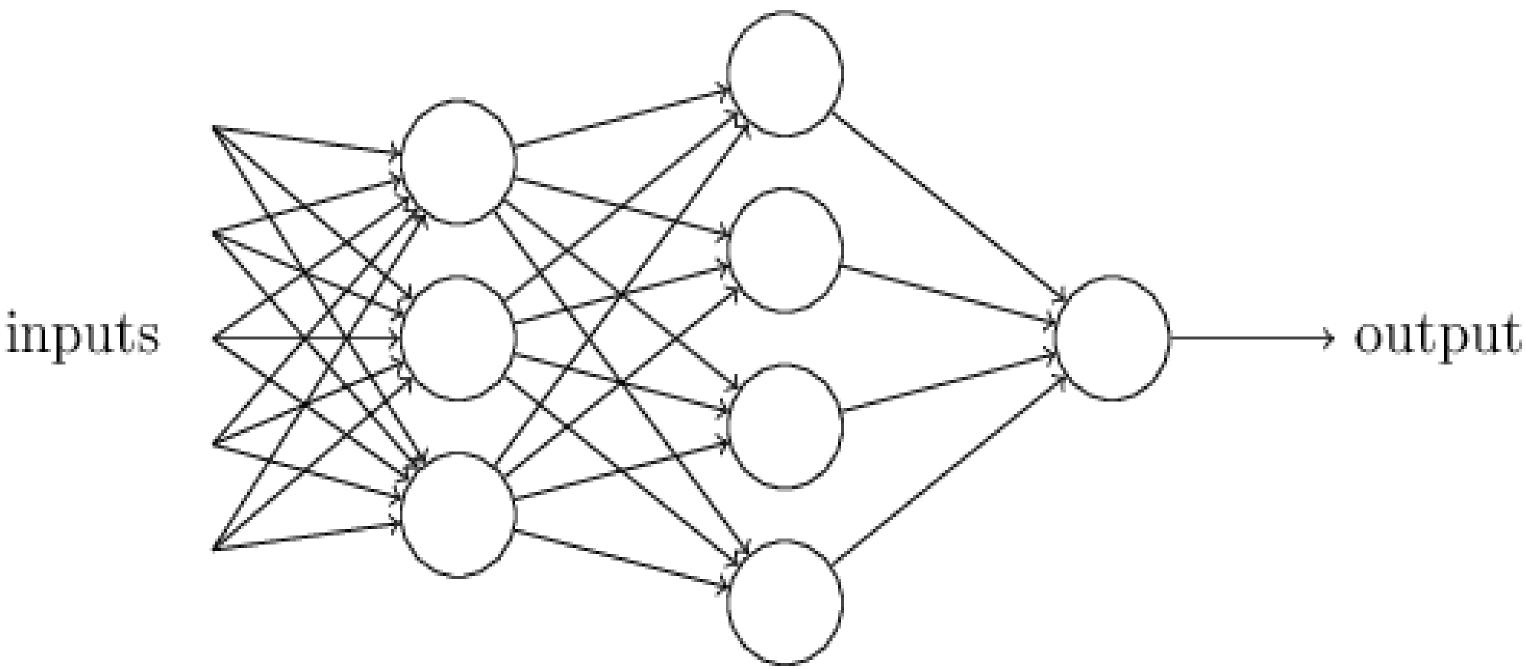
  $f(x) = \ln(1+e^{x})$

- many others



Learning: error backpropagation

- a single neuron is weak
- a network of neurons can approximate any continuous function
- deep neural network: more than one hidden level



- learning: error backpropagation

Backpropagation learning algorithm for NN

- Backpropagation: A neural network learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples
- Also referred to as connectionist learning due to the connections between units

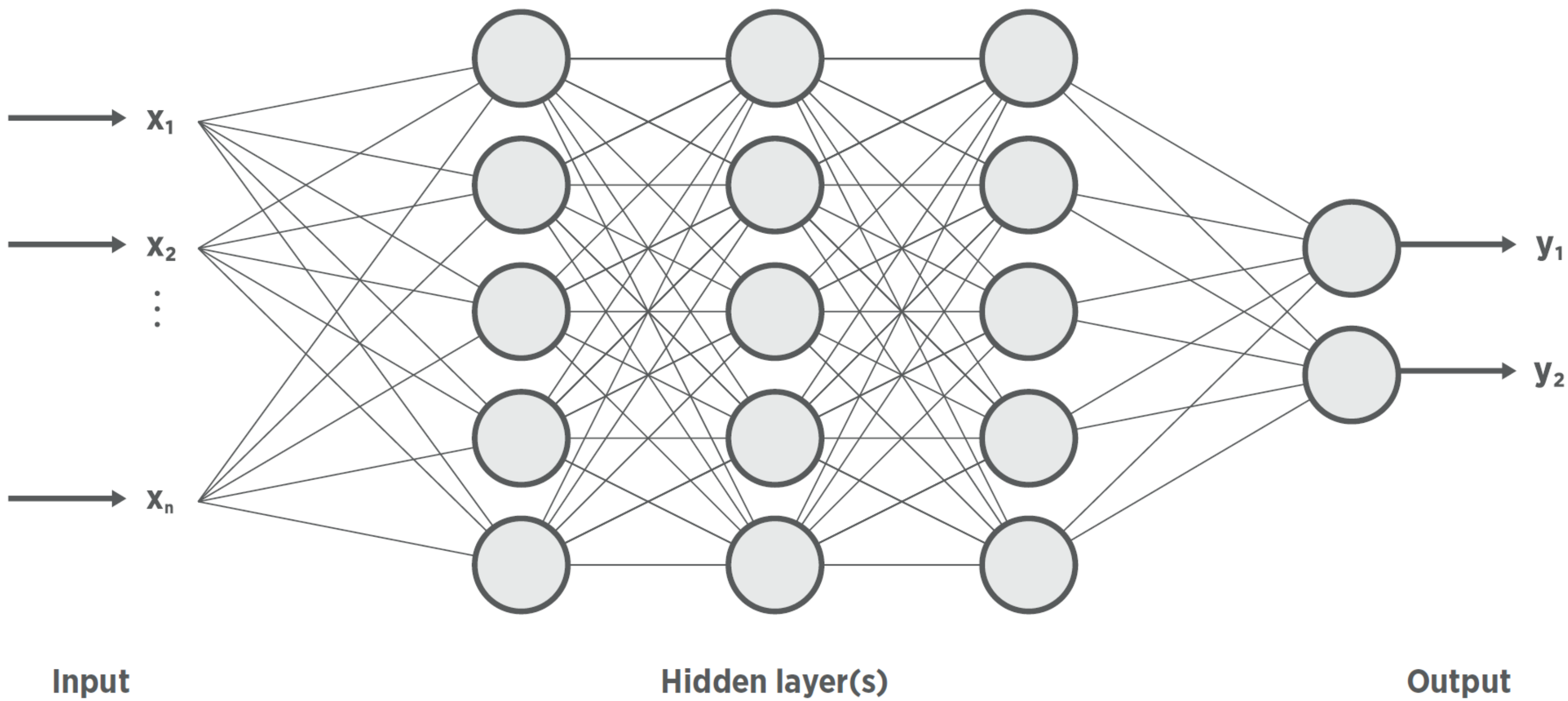
How a multi-layer feed-forward NN works?

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary; if more than 1 hidden layer is used, the network is called deep neural network
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward** if none of the weights cycles back to an input unit or to an output unit of a previous layer
- If we have backwards connections the network is called recurrent neural network
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

Deep learning =

deep neural networks + large data sets + GPU

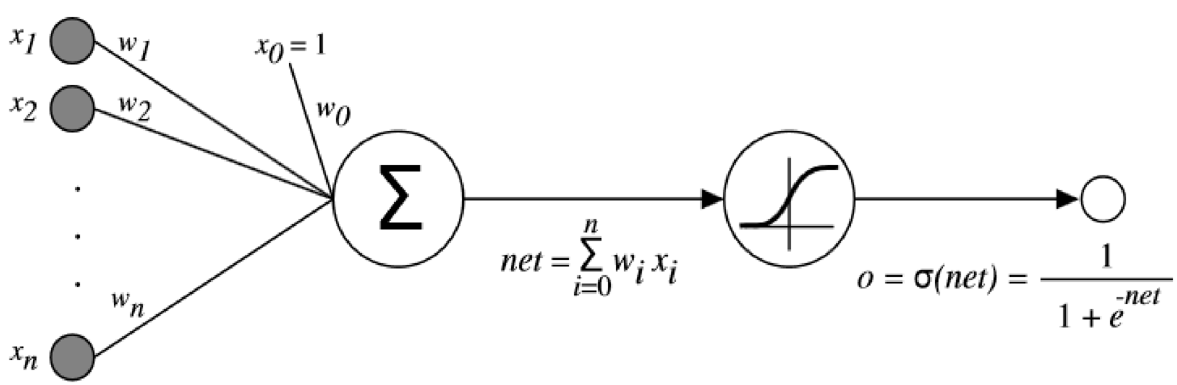
(+many new ideas)



Why nonlinear activation function?

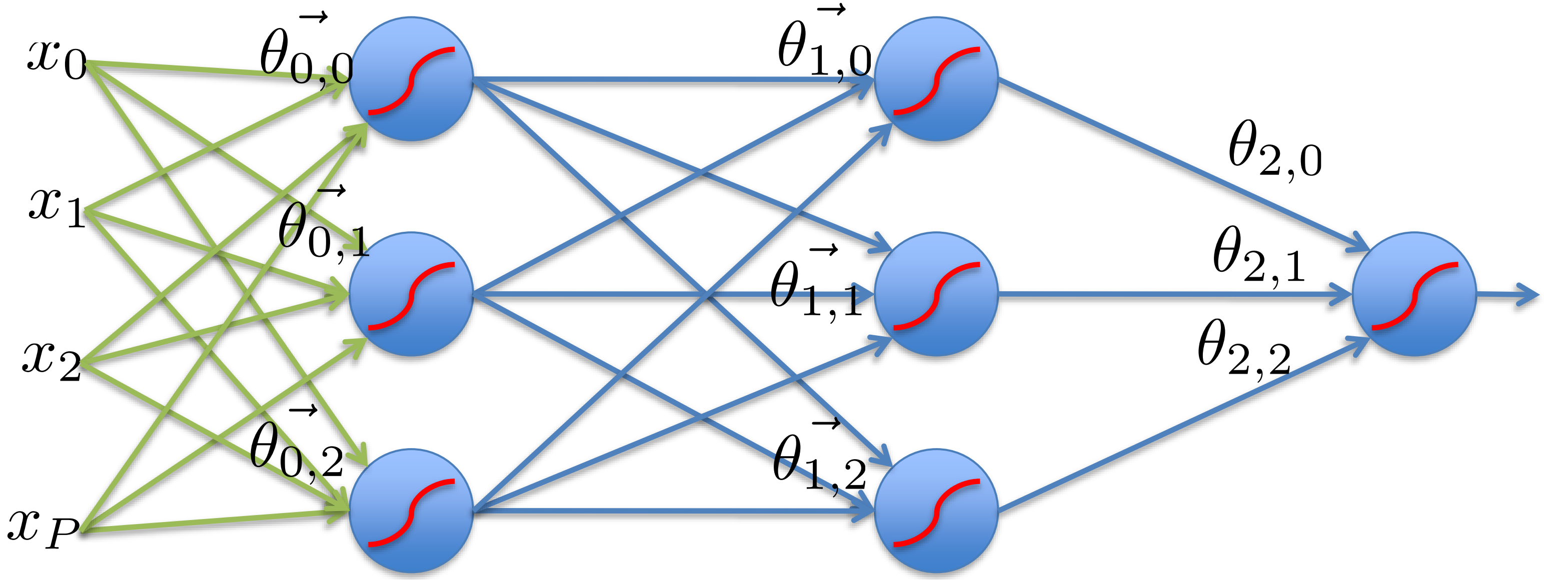
- The product of two linear transformations is itself a linear transformation.

- What is a derivative of a sigmoid?



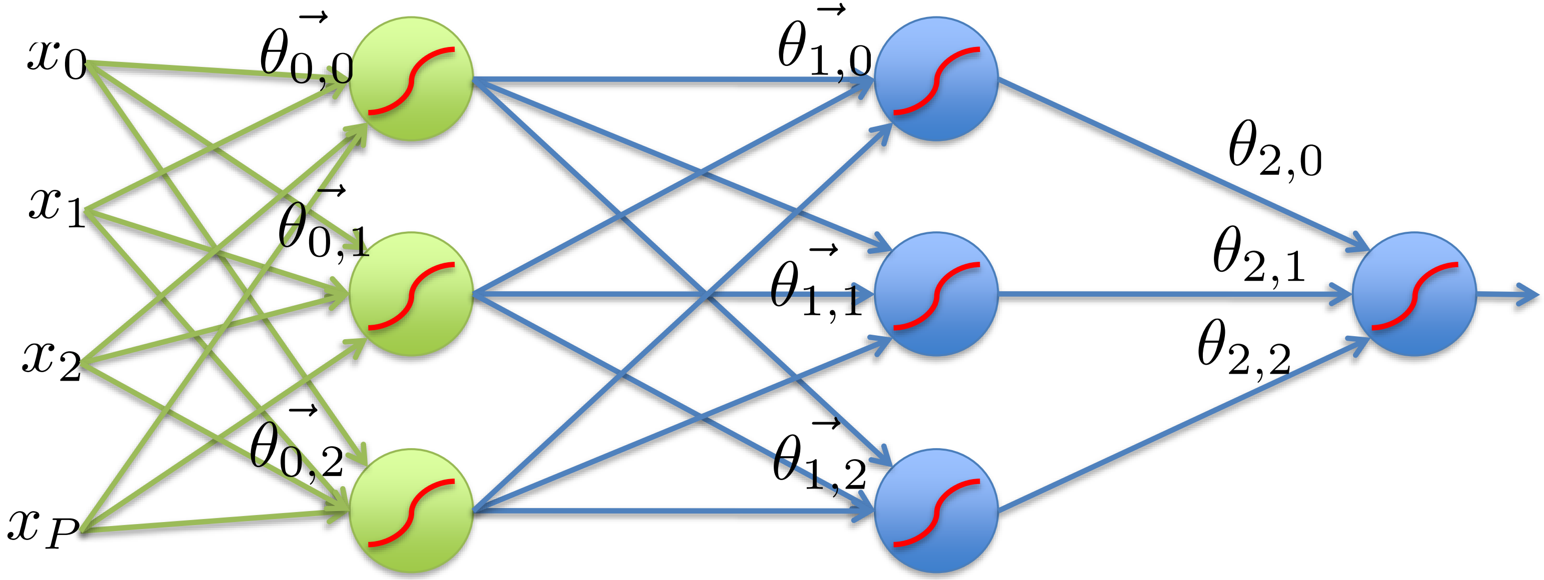
Feed-Forward Network

- Values are propagated from input through the network till the output layer which returns the prediction

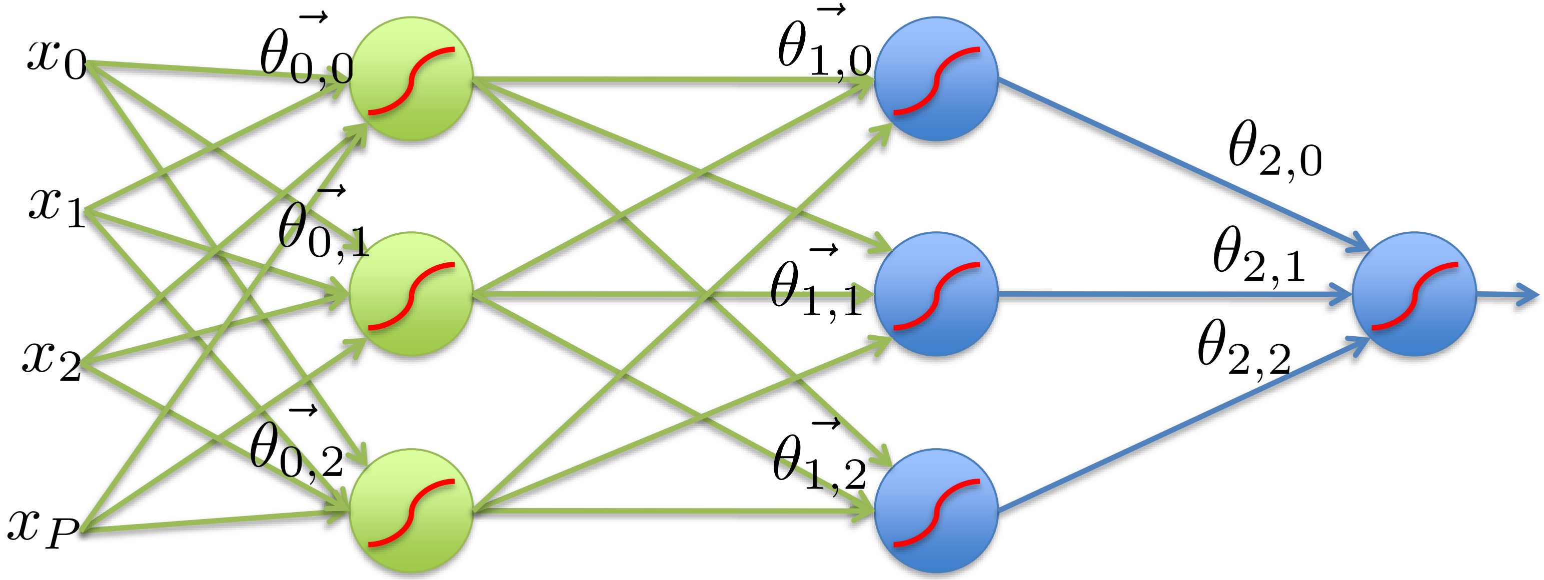


Next 6 slides by Andrew Rosenberg

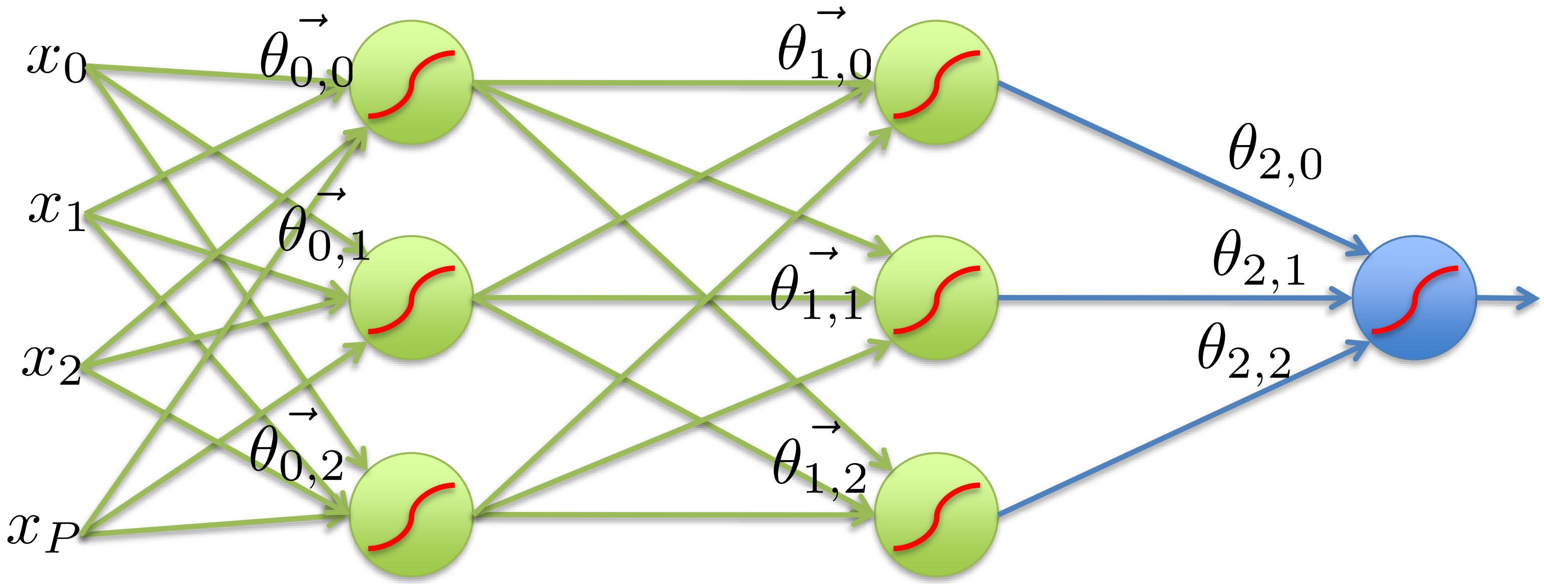
Feed-Forward Networks



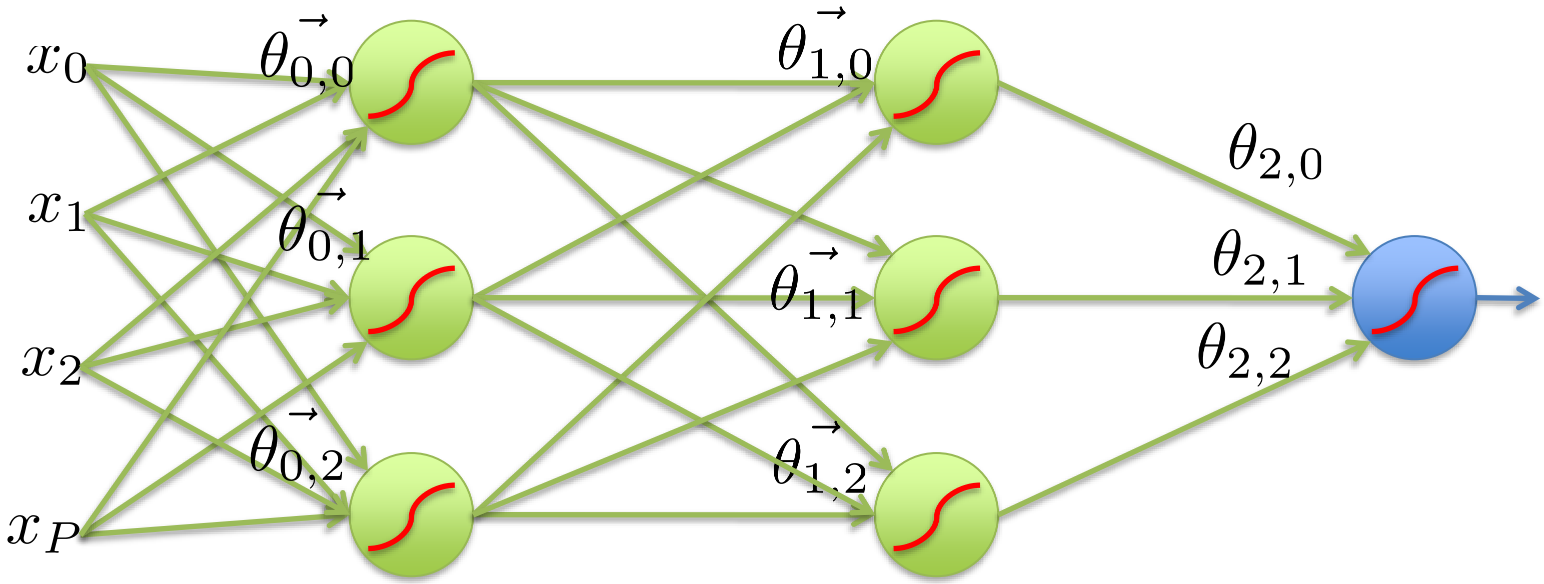
Feed-Forward Networks



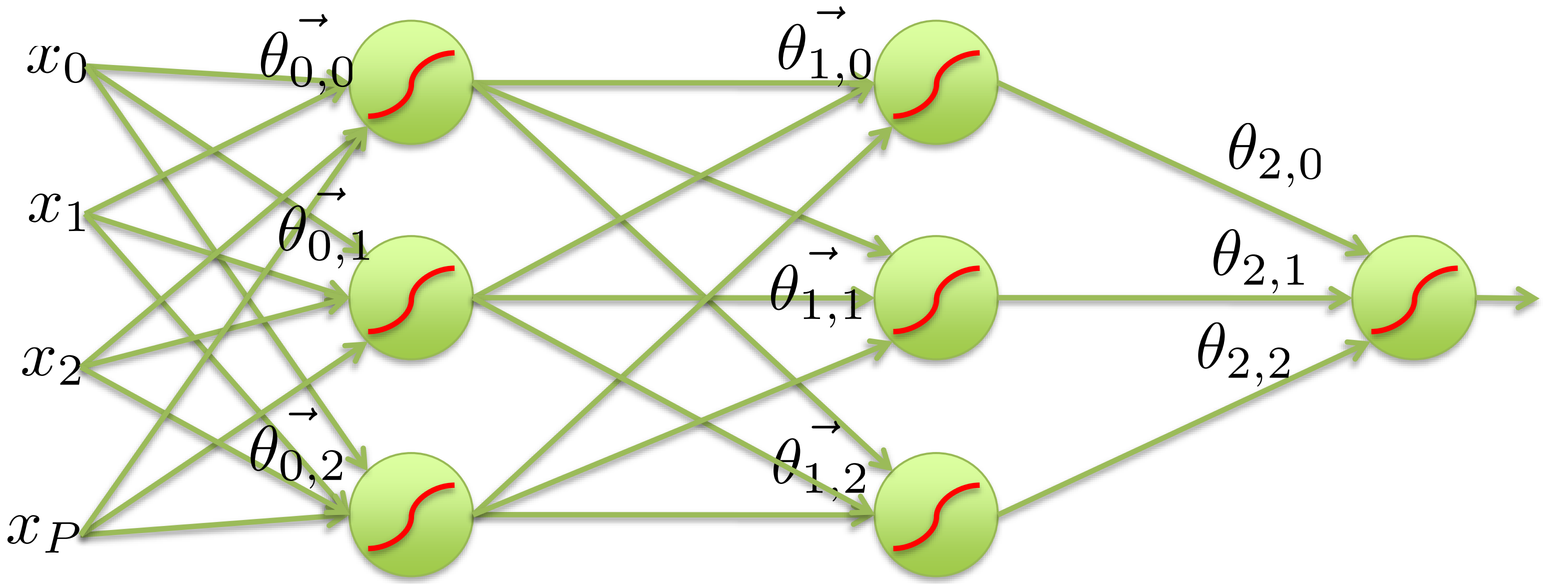
Feed-Forward Networks



Feed-Forward Networks



Feed-Forward Networks



Softmax

- normalizes the output scores to be a probability distribution (values between 0 and 1, the sum is 1)

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i \, (1 - y_i)$$

Criterion function

- together with softmax we frequently use cross entropy as cost function C

$$C = -\sum_j t_j \log y_j$$

target value

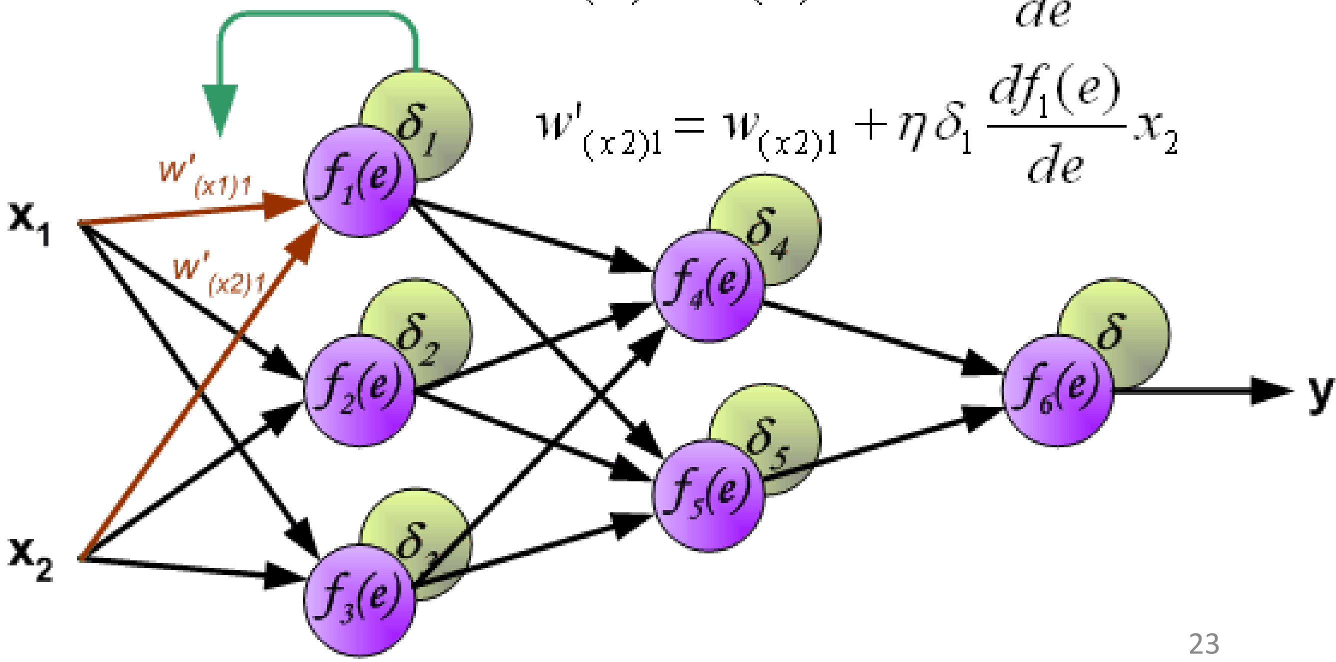
$$\frac{\partial C}{\partial z_i} = \sum_j \frac{\partial C}{\partial y_j} \frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Learning with error backpropagation

- Backpropagation
- randomly initialize parameters (weights)
- compute error on the output
- compute contributions to error, δ_n, on each step backwards
- gradient
- step
- iteratively
- batch
- minibatch

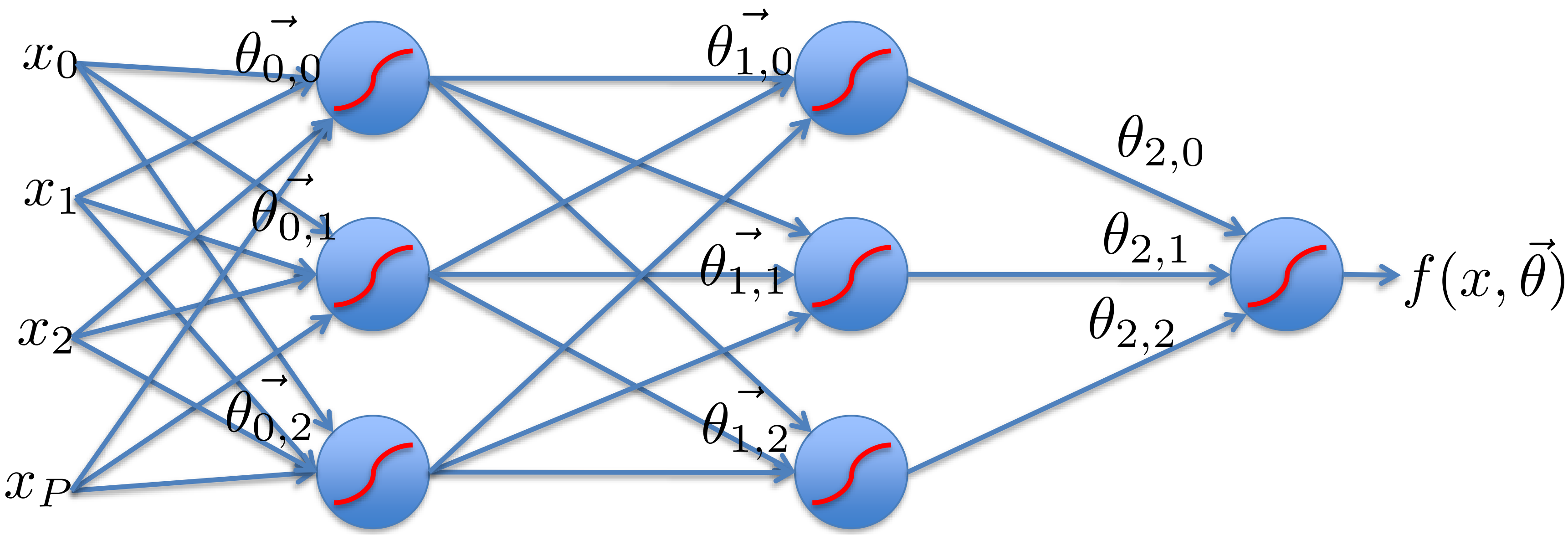
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Error Backpropagation

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

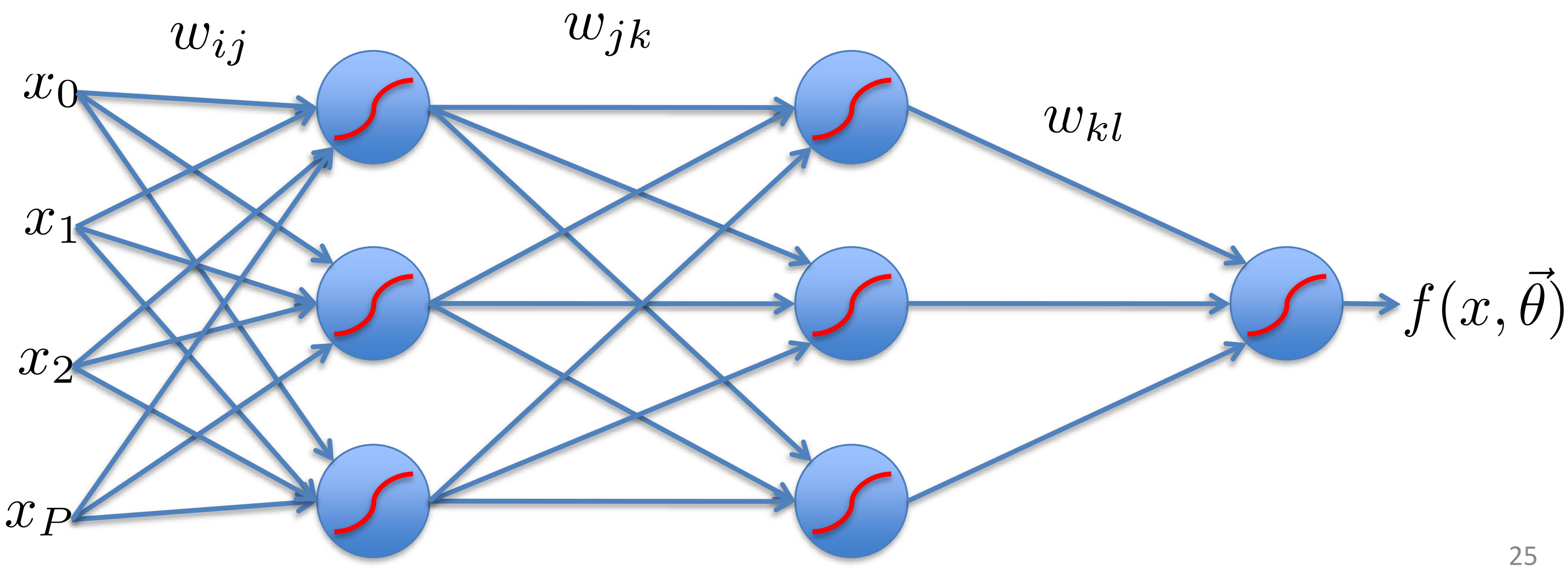


Next 18 slides by Andrew Rosenberg

Error Backpropagation

- Introduce variables over the neural network

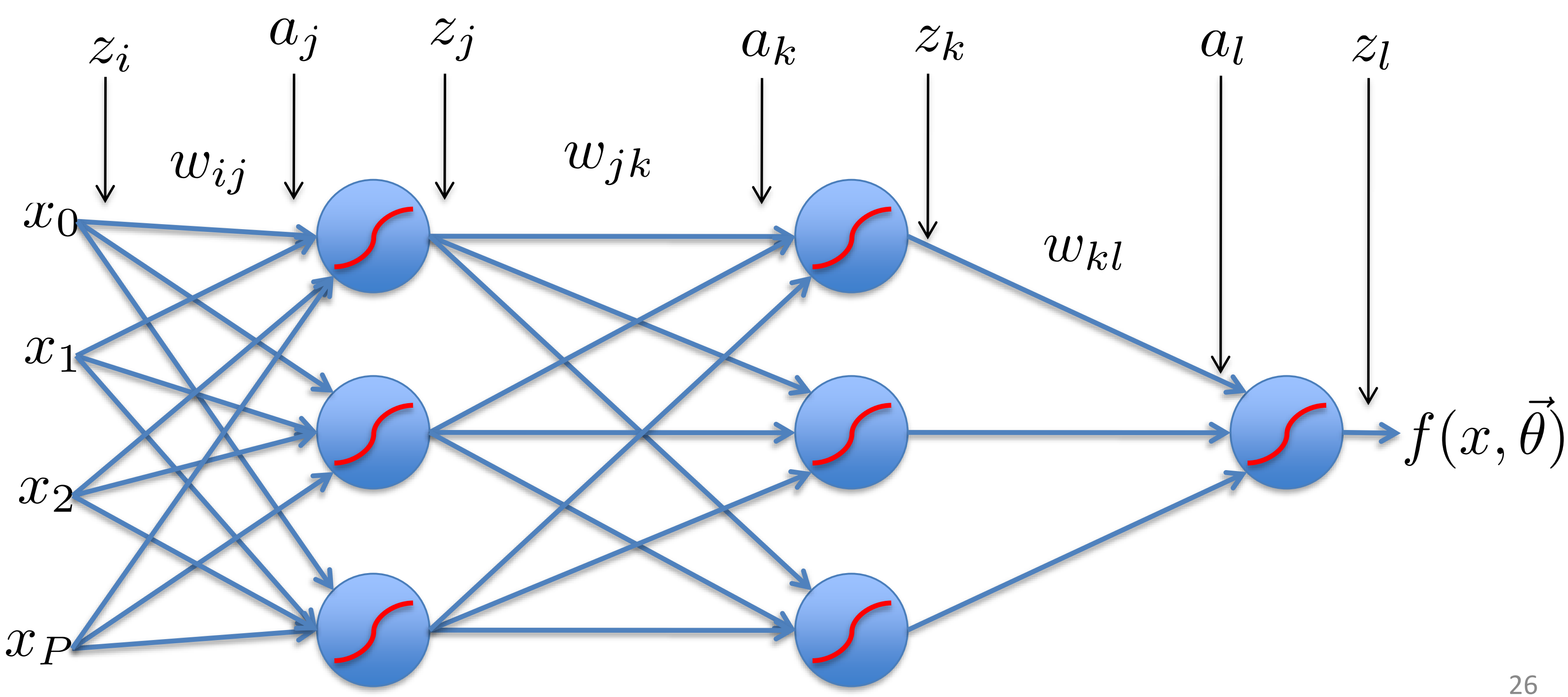
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$



Error Backpropagation

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

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

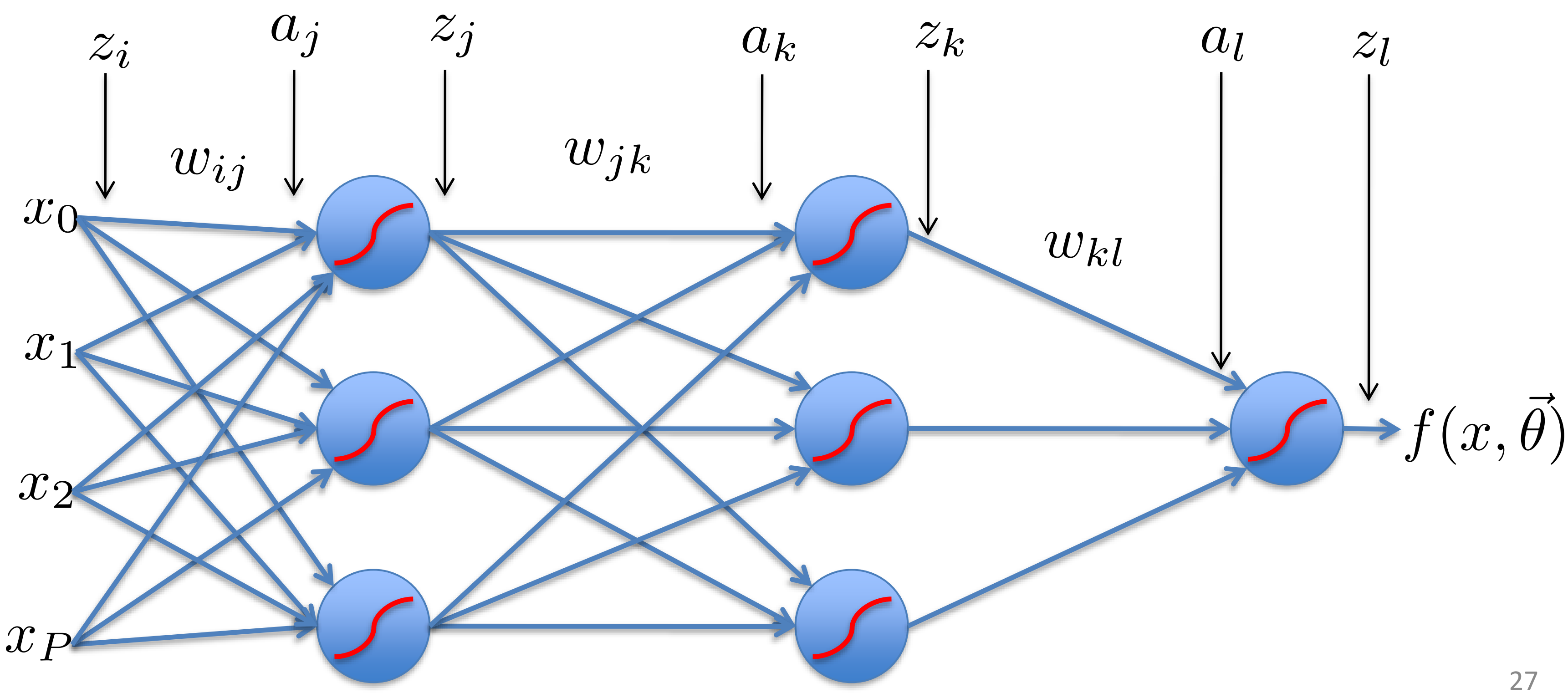


Error Backpropagation

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

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

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



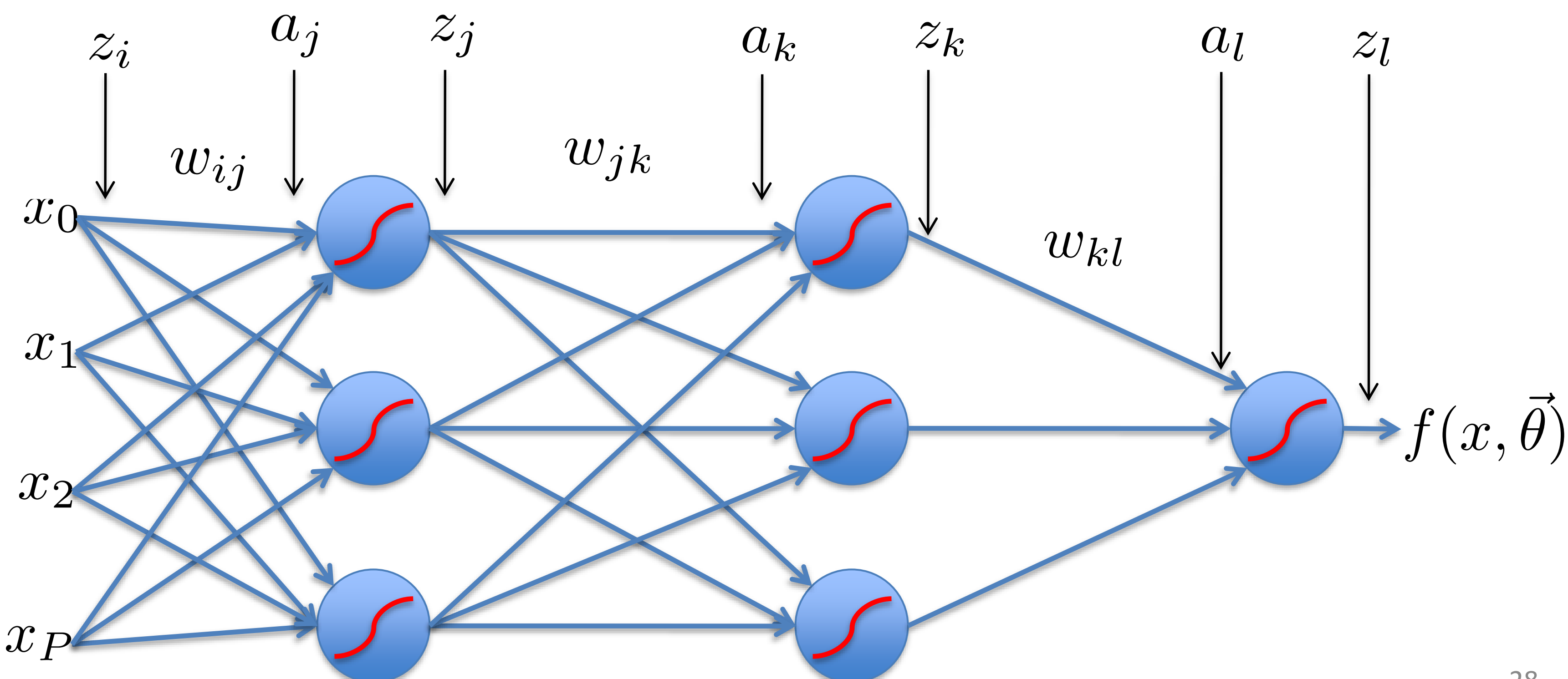
Error Backpropagation

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

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

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

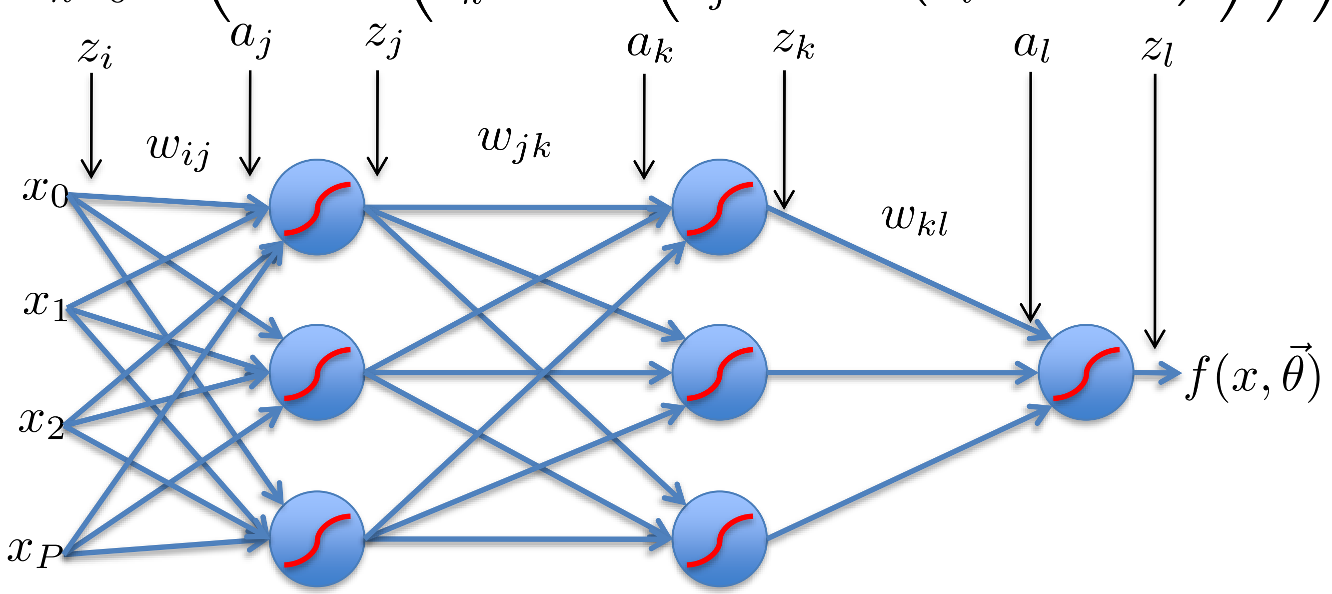
$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



Error Backpropagation

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

Empirical Risk Function



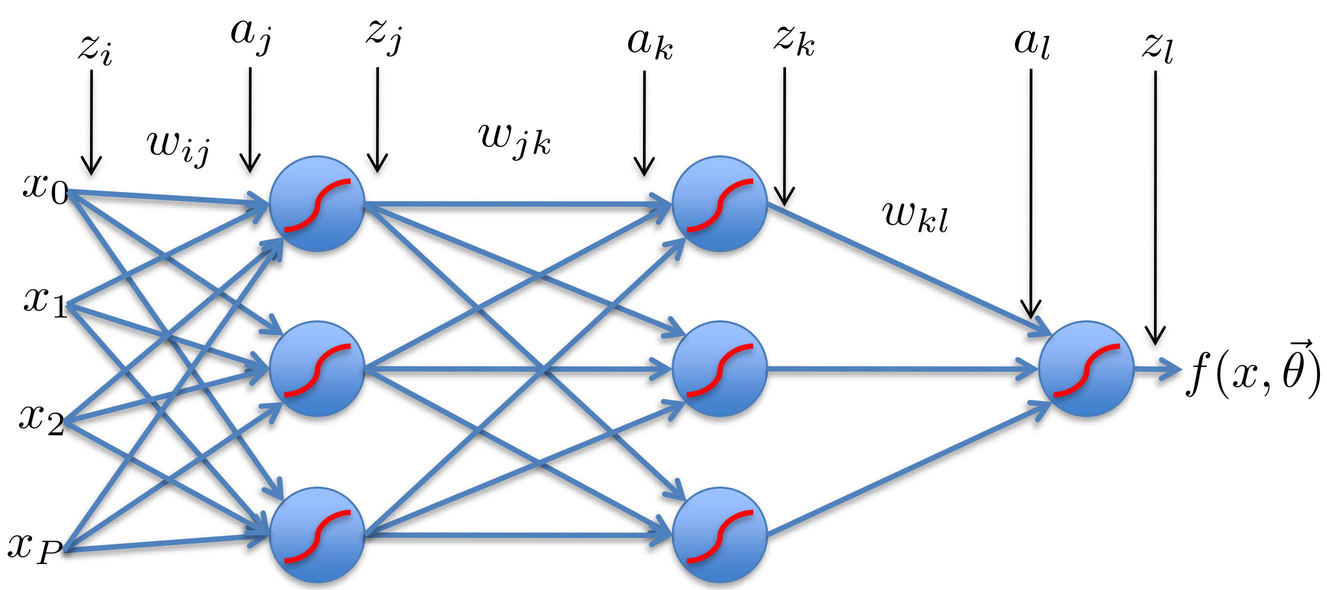
Error Backpropagation

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



Error Backpropagation

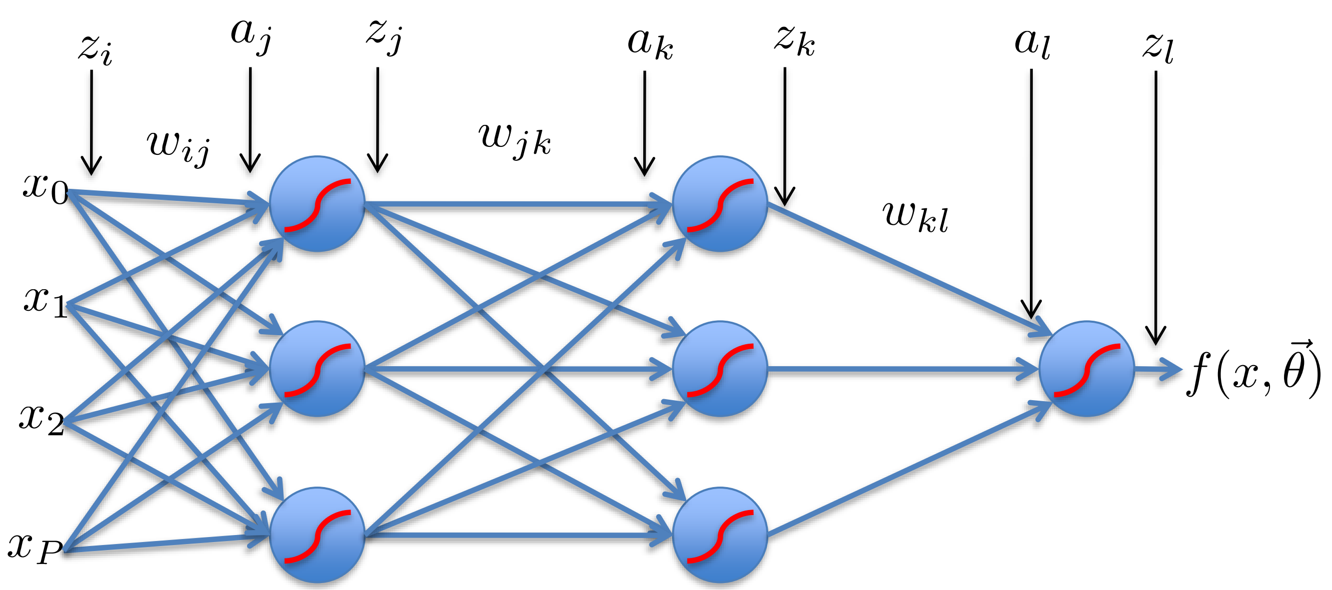
Optimize last layer weights w_{kl}

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

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

Calculus chain rule

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



Error Backpropagation

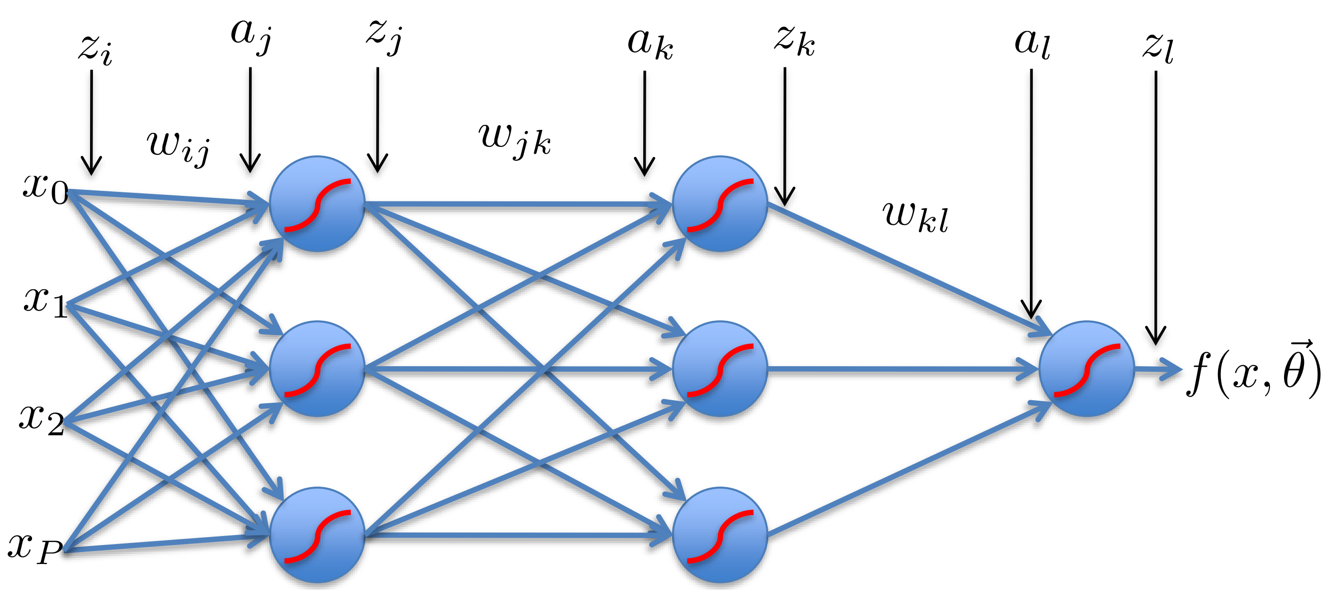
Optimize last layer weights w_{kl}

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

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

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

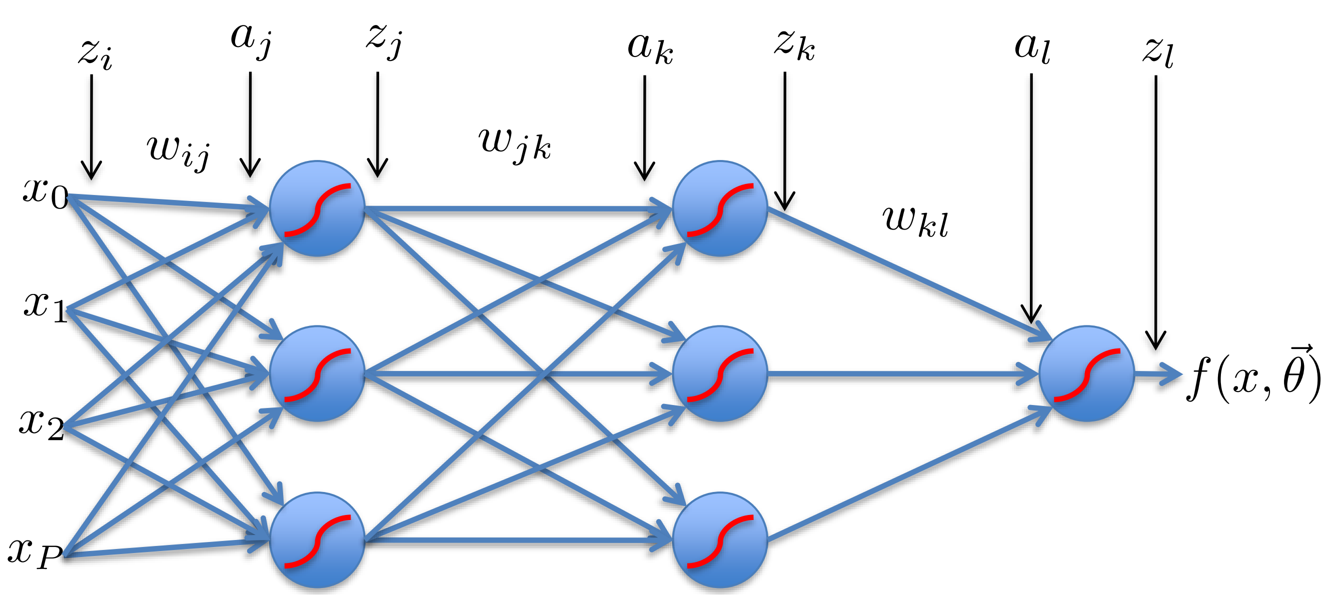
Optimize last layer weights w_{kl}

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

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

Calculus chain rule

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



Error Backpropagation

Optimize last layer weights w_{kl}

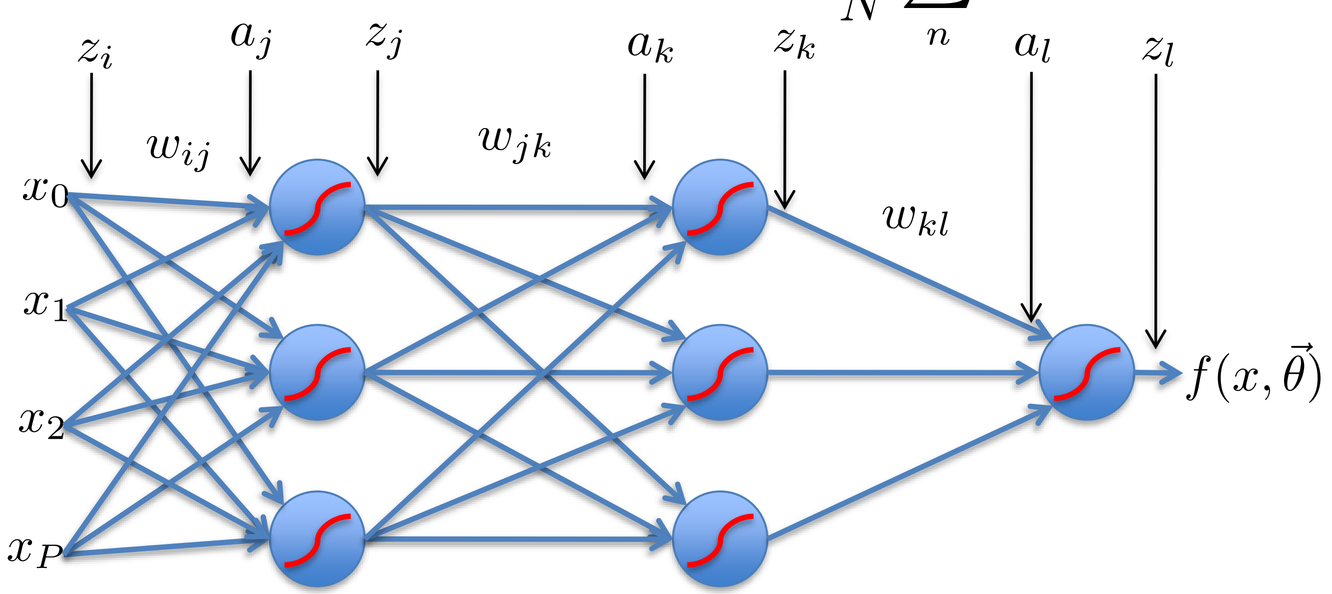
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

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

Calculus chain rule

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

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

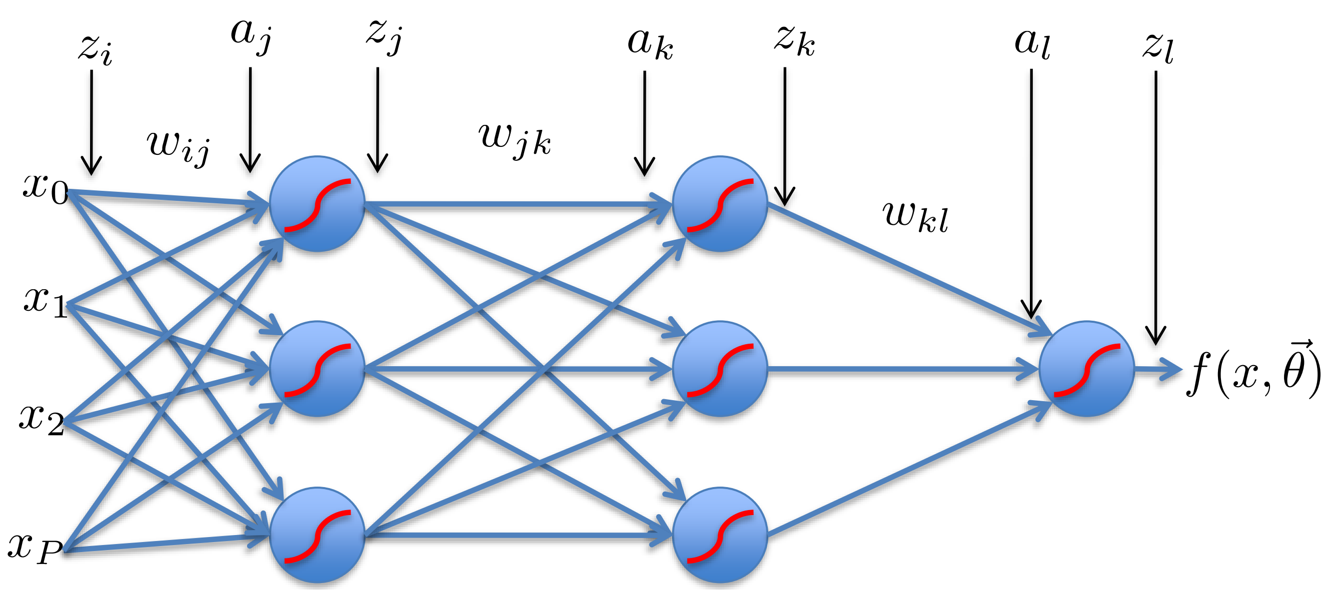


Error Backpropagation

Optimize last hidden weights w_{jk}

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

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



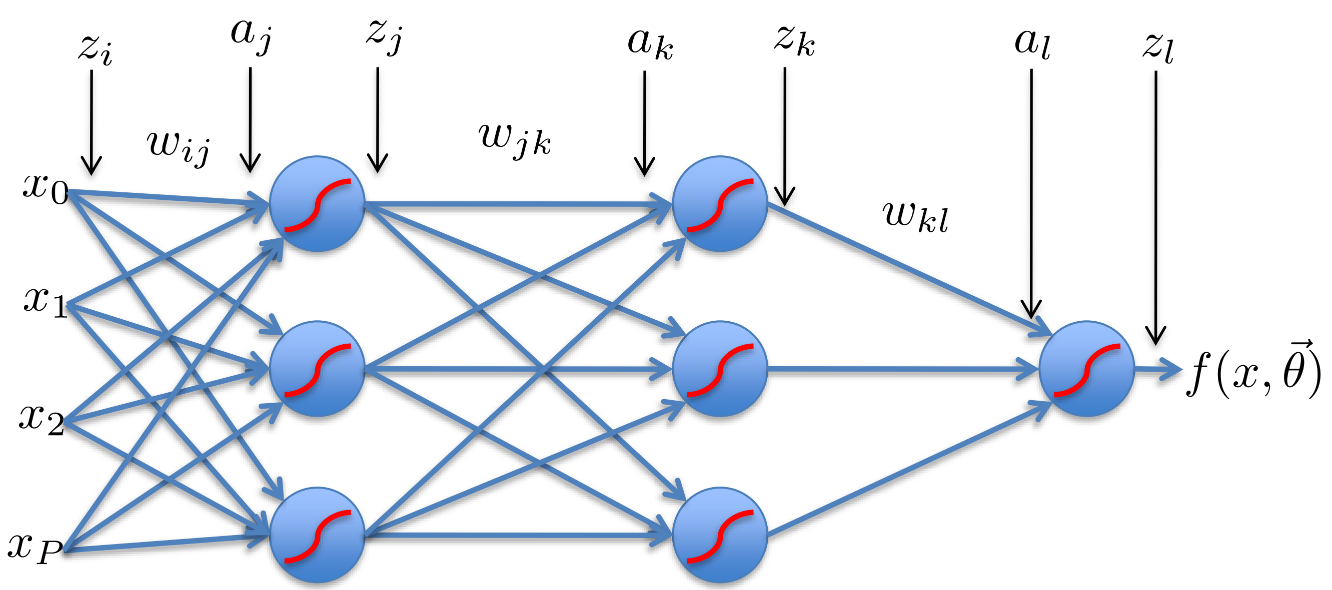
Error Backpropagation

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

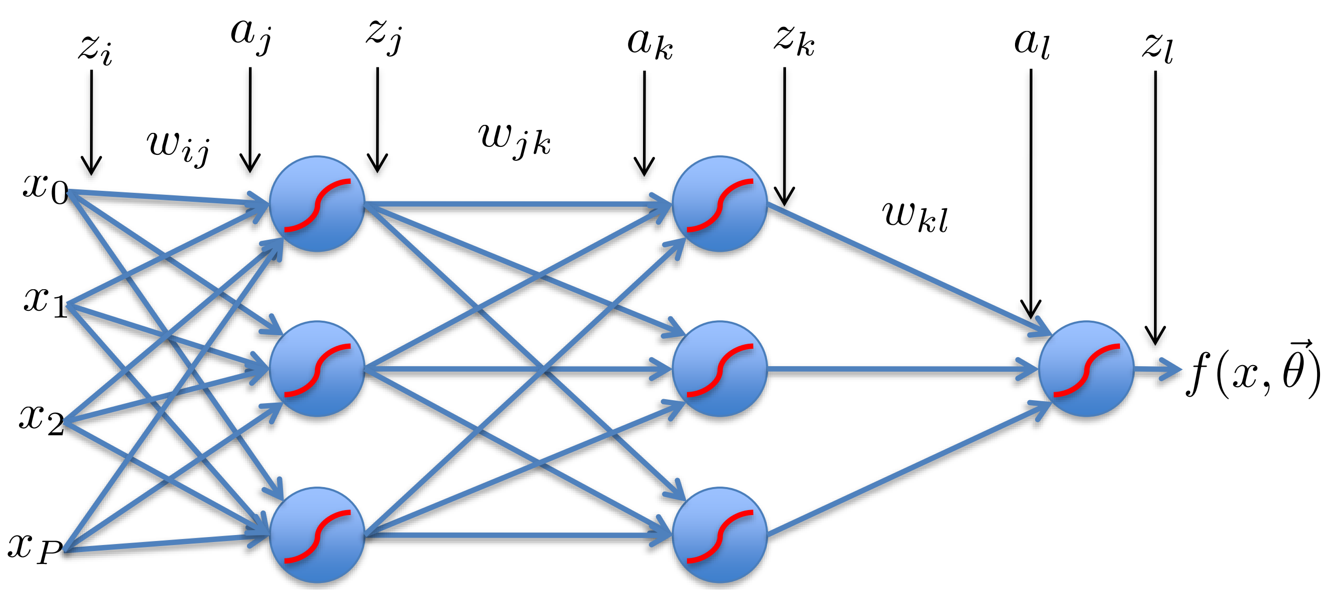
Optimize last hidden weights w_{jk}

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

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

Multivariate chain rule

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



Error Backpropagation

Optimize last hidden weights w_{jk}

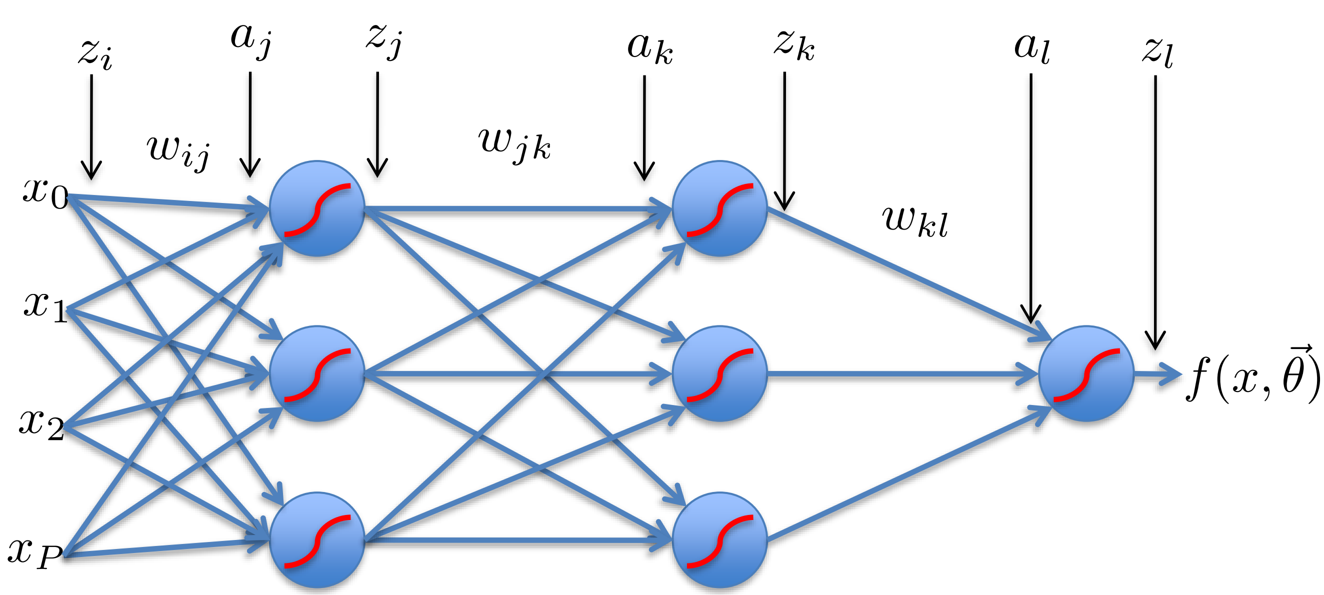
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

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

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}

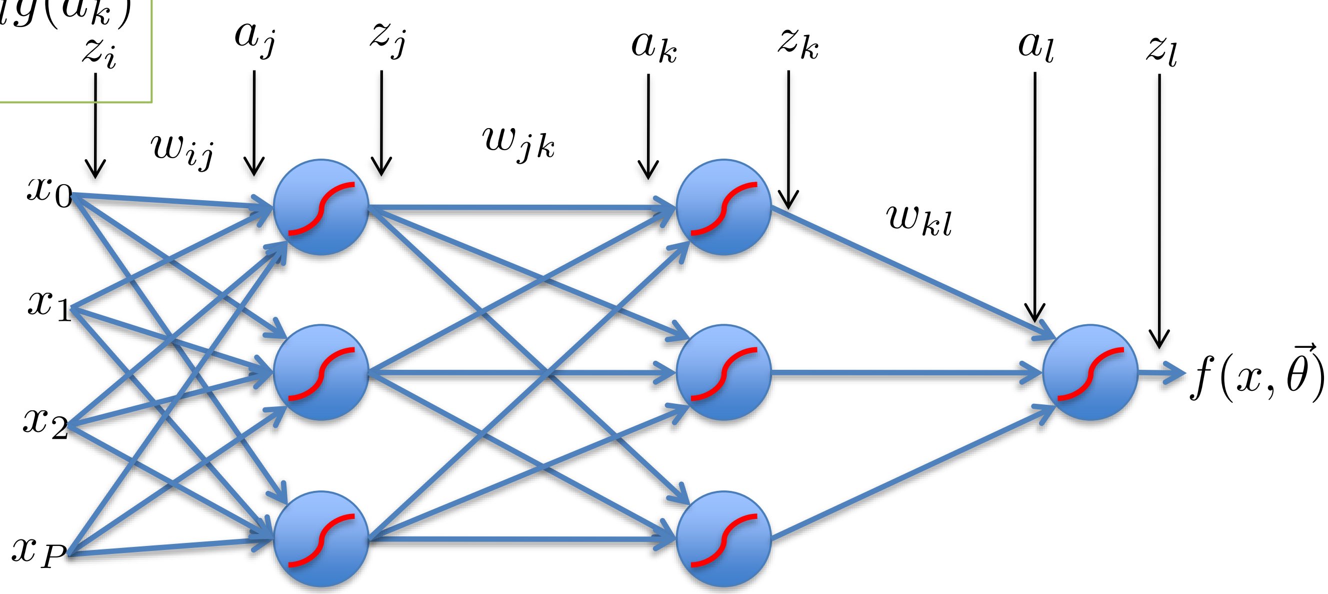
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

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

Multivariate chain rule

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

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



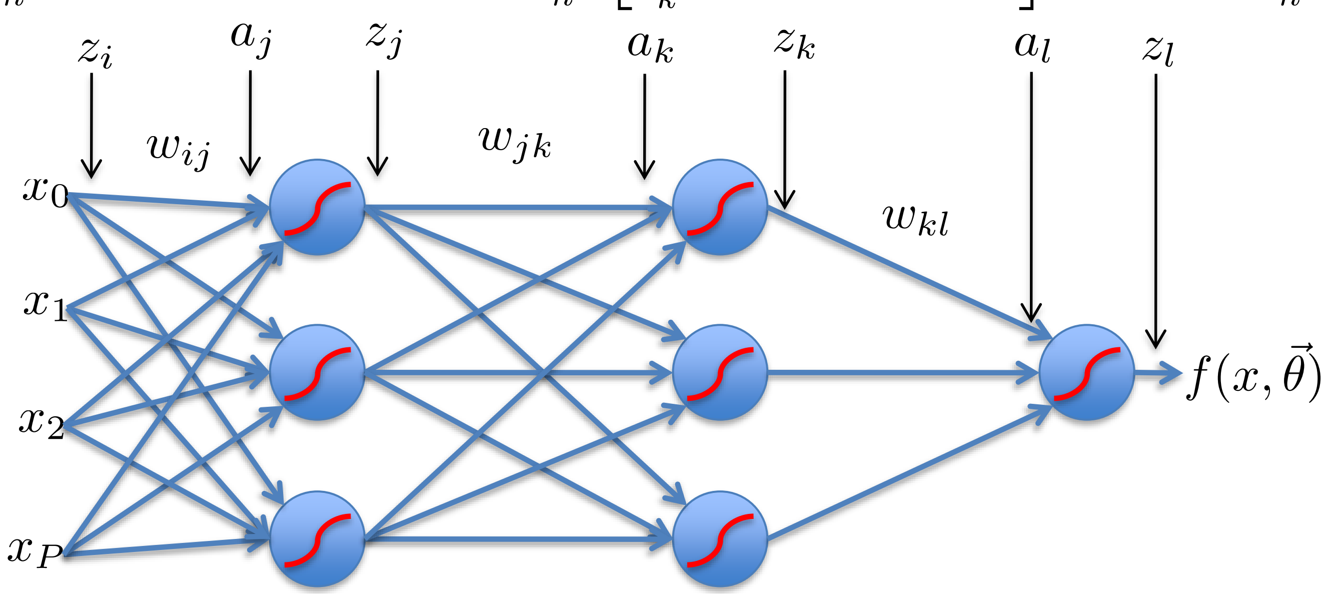
Error Backpropagation

Repeat for all previous layers

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

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

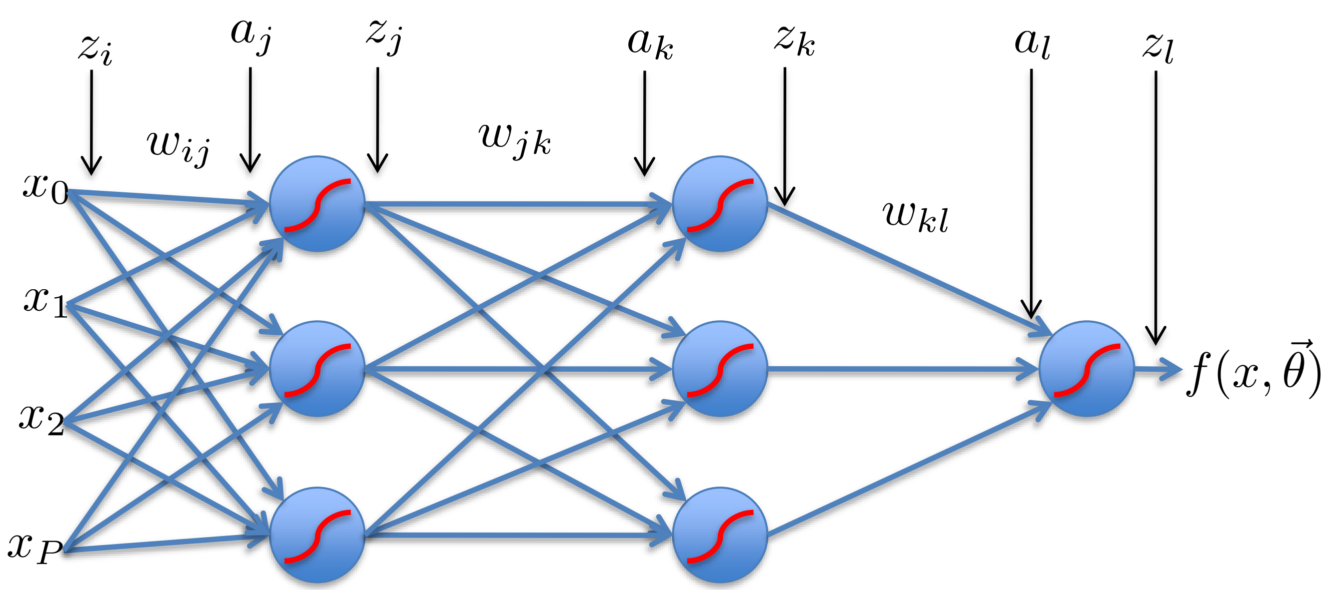
$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$



Error Backpropagation

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

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



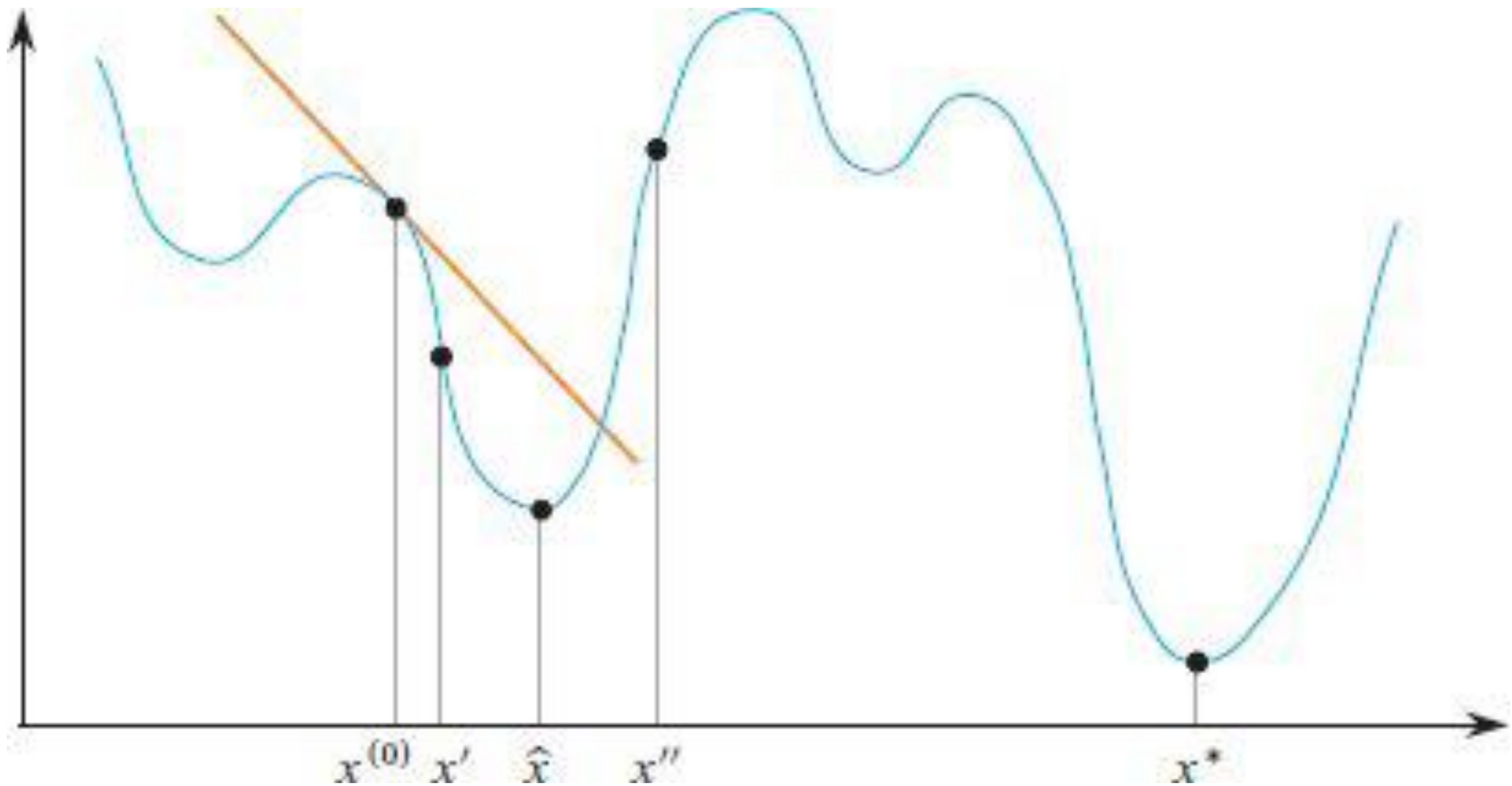
Gradient descent (GD)

- Gradient descent is an efficient local optimization in $\mathbb{R}^n$
- Local minimum of function *f:* $\mathbb{R}^n \rightarrow \mathbb{R}$ is a point ***x*** for which *f(x) ≤ f(x')* for all ***x'*** that are "near" ***x***
- Gradient $\nabla f(x)$ is a function ∇f: $\mathbb{R}^n \rightarrow \mathbb{R}^n$ comprising *n* partial derivatives:

$$\nabla f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n})$$

- The GD optimization moves in the direction of -$\nabla f(x)$

Ilustration of GD



GD algorithm

```
GRADIENT-DESCENT(f, x0, γ, T) {
  // function f,  initial value x0, fixed step size γ, number of steps T
  x_best = x = x0 ; // n-dimensional vectors, initially set to the initial value
  f_best = f_x = f(x_best) ;
  for t = 0 to T – 1 do {
    x_next = x – γ ▪ ∇f(x);  // ∇f(x), x, and x_next are n-dimensional
    f_next = f(x_next)
    if (f_next < f_x)
      x_best = x_next ;
    x = x_next ;
    f_x = f_next ;
  }
  return x_best ;
}
```

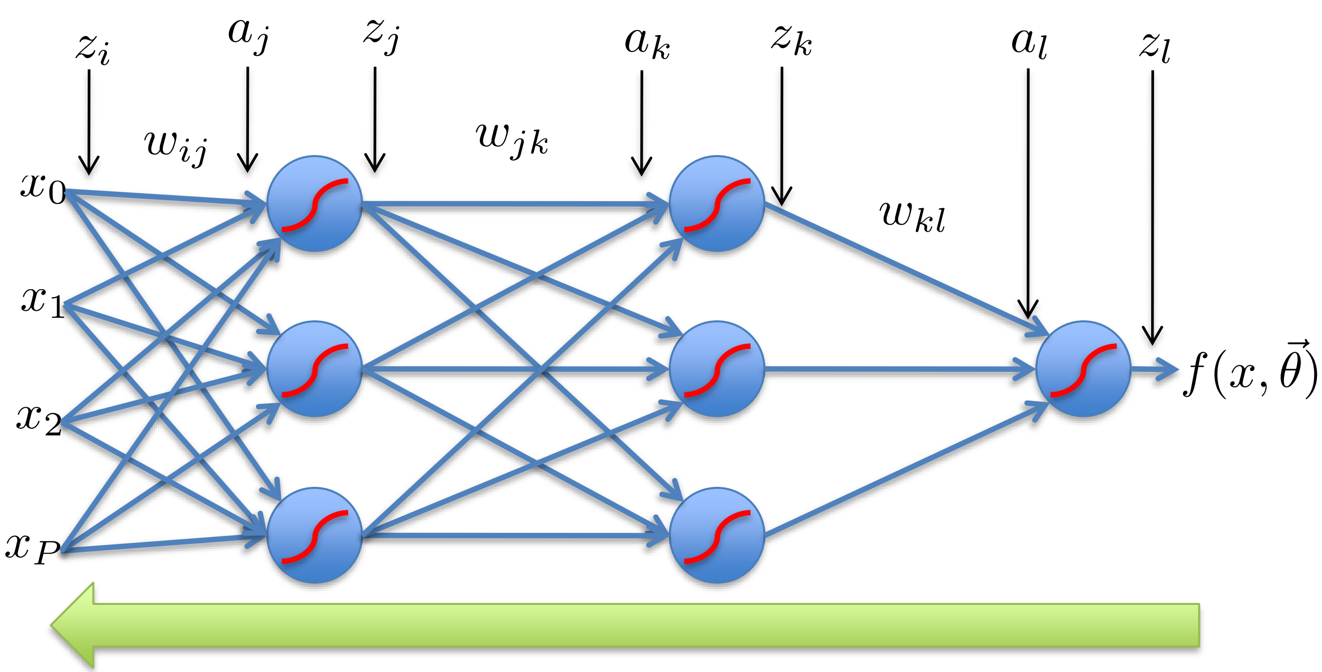
Chain rule of derivation

- In a network, the output of each neuron is a function of activation function and all its inputs, where the inputs may again be composite functions of neurons in previous layers
- To compute a gradient of a composite function, we use the chain rule of derivation

$$f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$$

Error Back-propagation

- Error backpropagation 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.



Backpropagation algorithm

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Defining a network topology

- Decide the network topology: Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One input unit per domain value, each initialized to 0
- Output, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is still unacceptable, repeat the training process with a *different network topology* or a *different set of initial weights*

Neural network as a classifier

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs *and outputs*
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques exist for the extraction of explanations from trained neural networks

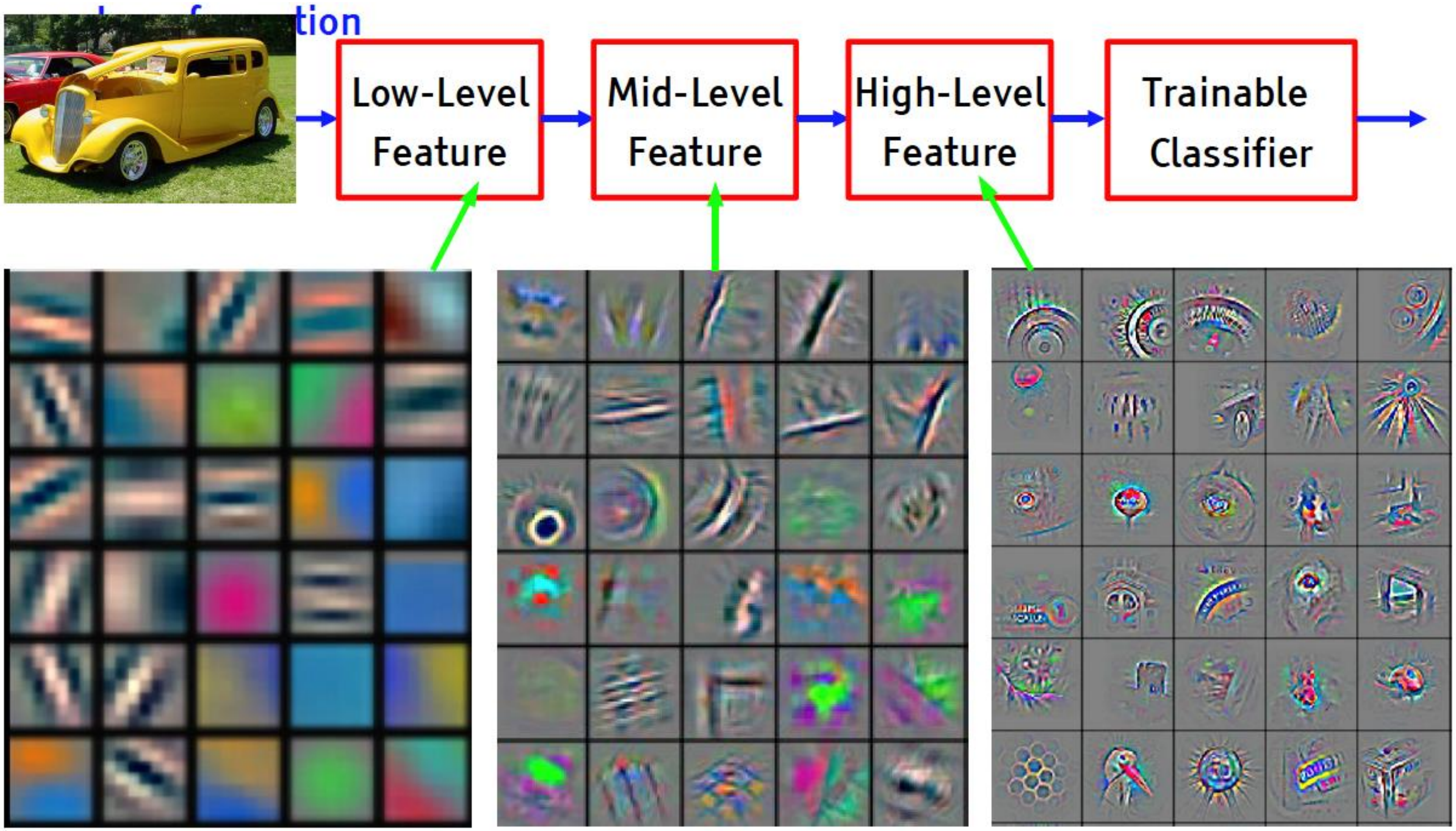
Efficiency

- Efficiency of backpropagation: Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be large (e..g, exponential to n, the number of inputs), in worst case

Interpretation of hidden layers

- What are the hidden layers doing?!
- Feature Extraction
- The non-linearities in the feature extraction can make interpretation of the hidden layers very difficult.
- This leads to Neural Networks being treated as **black boxes**.

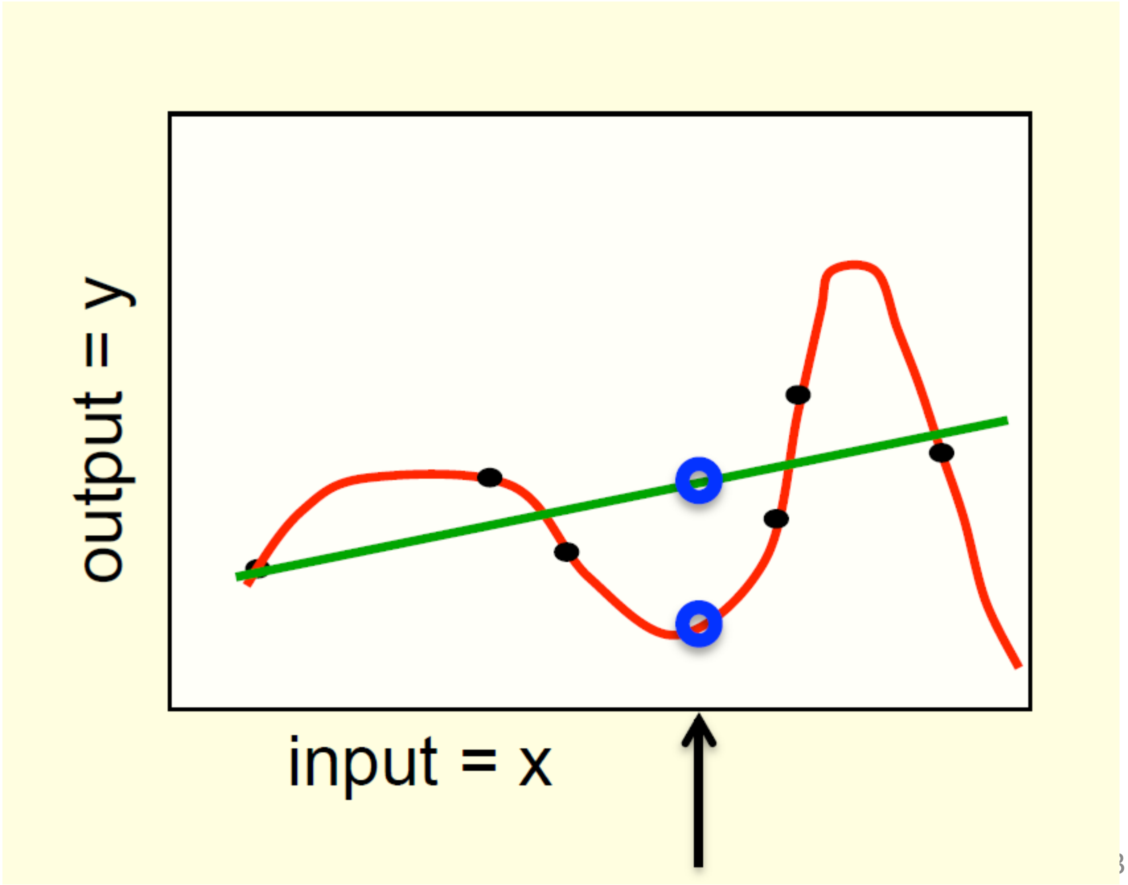
Deep learning = learning of hierarchical representation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Overfitting and model complexity

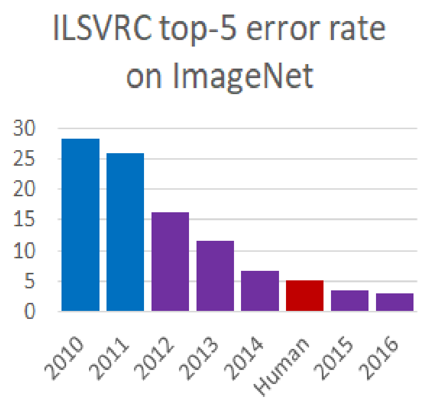
- which curve is more plausible given the data?
- overfitting
- neural nets are especially prone to overfitting
- why?

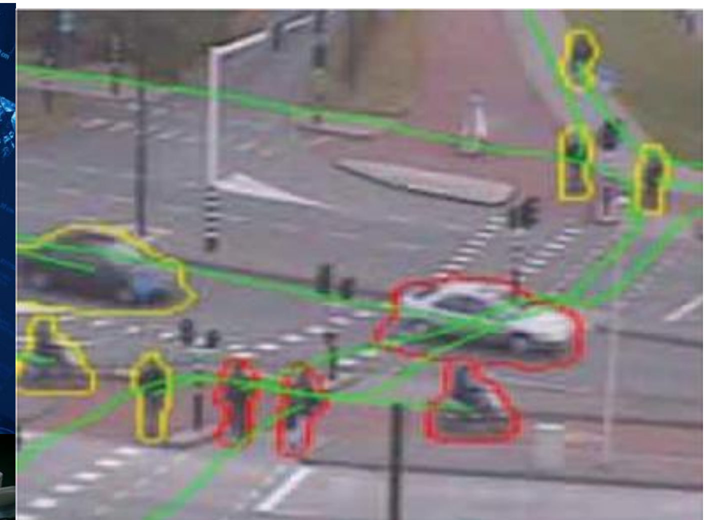


Approaches to prevent overfitting

- Weight-decay
- Weight-sharing
- Early stopping
- Model averaging
- Bayesian fitting of neural nets
- Dropout
- Generative pre-training
- etc.

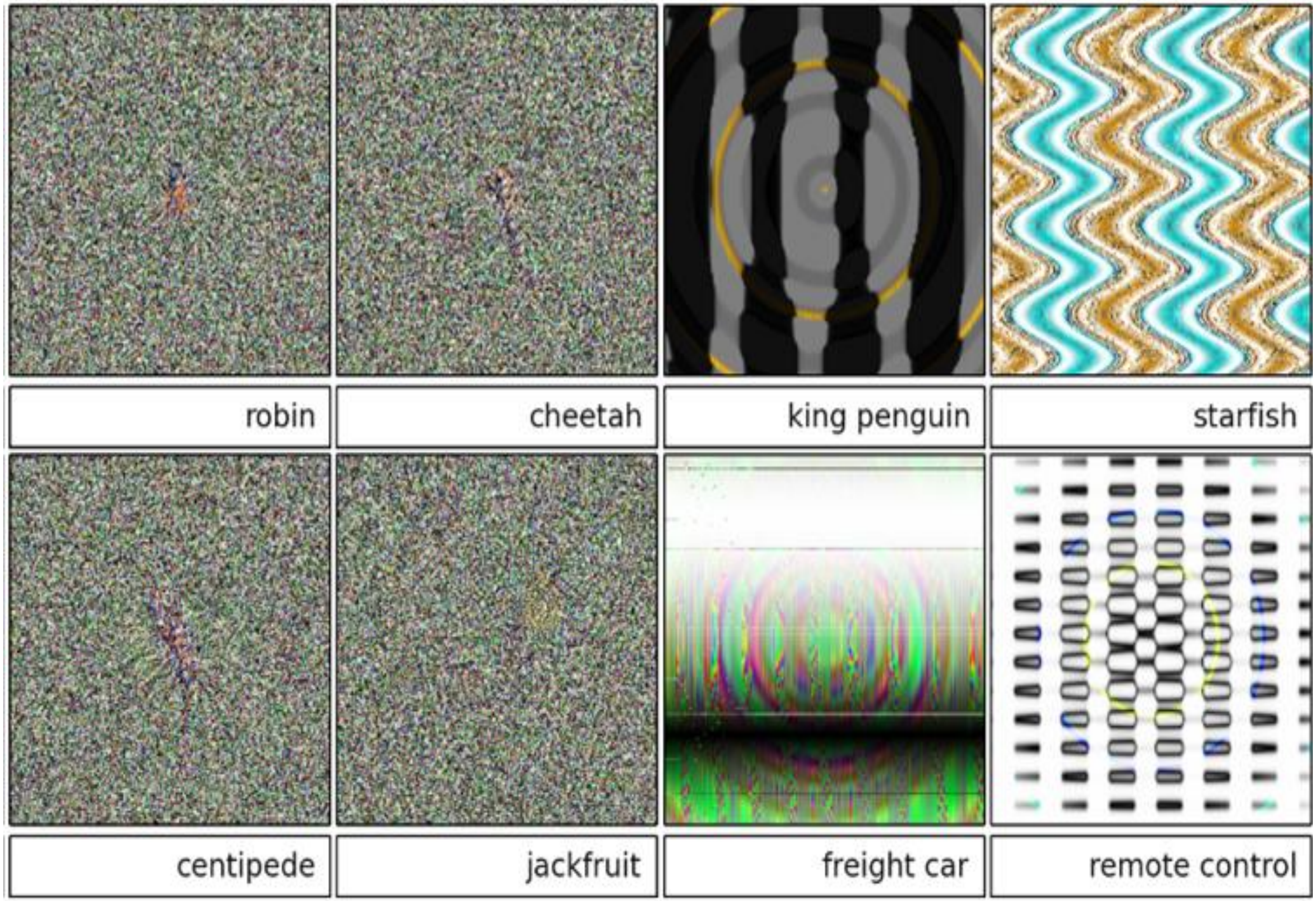
Deep learning successes



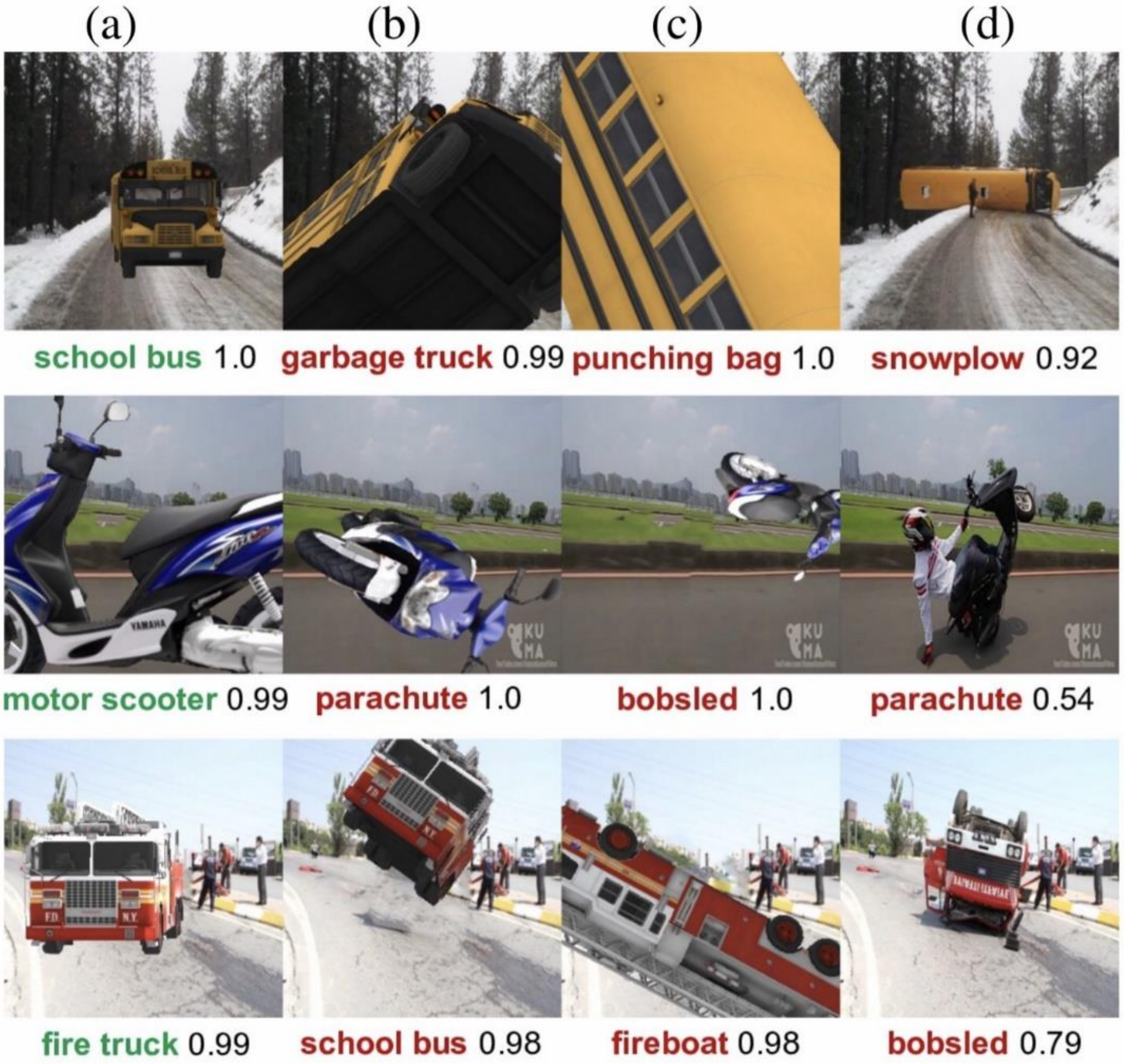


Microsoft claims new speech recognition record, achieving a super-human 5.1% error rate

Weaknesses of deep learning



Failures on out-of-distribution examples



Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, Anh Nguyen (2018):
Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects. arXiv:1811.11553

Attacks on neural networks