



---

# Neural Networks

Tobias Scheffer

# Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - ◆ Convolutional neural networks,
  - ◆ Recurrent neural networks.

# Learning Problems can be Impossible without the Right Features



→ motorcycle

88	82	84	88	85	83	80	93	102	88	82	84	88	85	83	80	93	102
88	80	78	80	80	78	73	94	100	88	80	78	80	80	78	73	94	100
85	79	80	78	77	74	65	91	99	85	79	80	78	77	74	65	91	99
38	35	40	35	39	74	77	70	65	38	35	40	35	39	74	77	70	65
20	25	23	28	37	69	64	60	57	20	25	23	28	37	69	64	60	57
22	26	22	28	40	65	64	59	34	22	26	22	28	40	65	64	59	34
24	28	24	30	37	60	58	56	66	24	28	24	30	37	60	58	56	66
21	22	23	27	38	60	67	65	67	21	22	23	27	38	60	67	65	67
23	22	22	25	38	59	64	67	66	23	22	22	25	38	59	64	67	66

?  
→ motorcycle

# Learning Problems can be Impossible without the Right Features

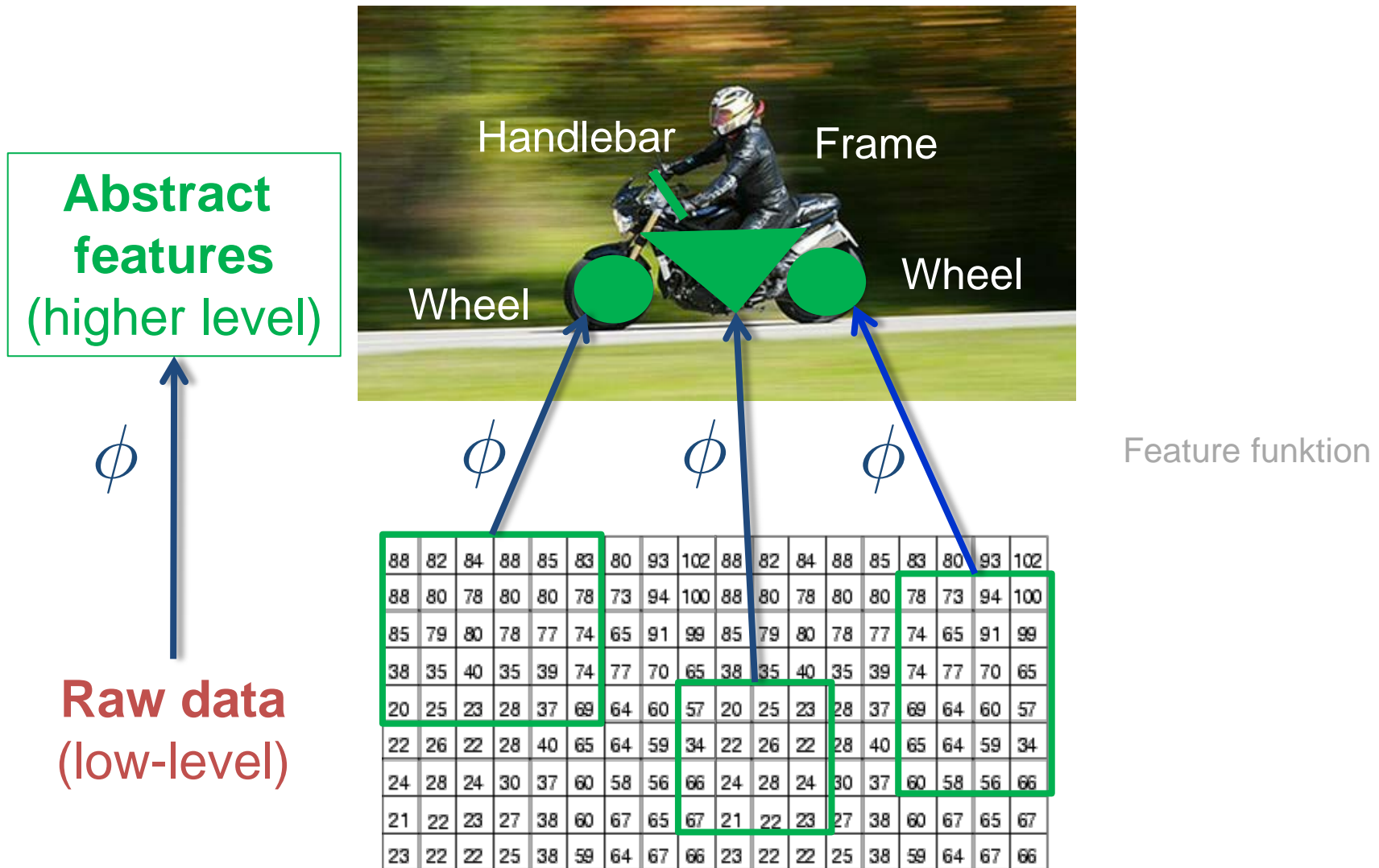


! → motorcycle

88	82	84	88	85	83	80	93	102	88	82	84	88	85	83	80	93	102
88	80	78	80	80	78	73	94	100	88	80	78	80	80	78	73	94	100
85	79	80	78	77	74	65	91	99	85	79	80	78	77	74	65	91	99
38	35	40	35	39	74	77	70	65	38	35	40	35	39	74	77	70	65
20	25	23	28	37	69	64	60	57	20	25	23	28	37	69	64	60	57
22	26	22	28	40	65	64	59	34	22	26	22	28	40	65	64	59	34
24	28	24	30	37	60	58	56	66	24	28	24	30	37	60	58	56	66
21	22	23	27	38	60	67	65	67	21	22	23	27	38	60	67	65	67
23	22	22	25	38	59	64	67	66	23	22	22	25	38	59	64	67	66

? → motorcycle

# Learning Problems can be Impossible without the Right Features



# Learning Problems can be Impossible without the Right Features

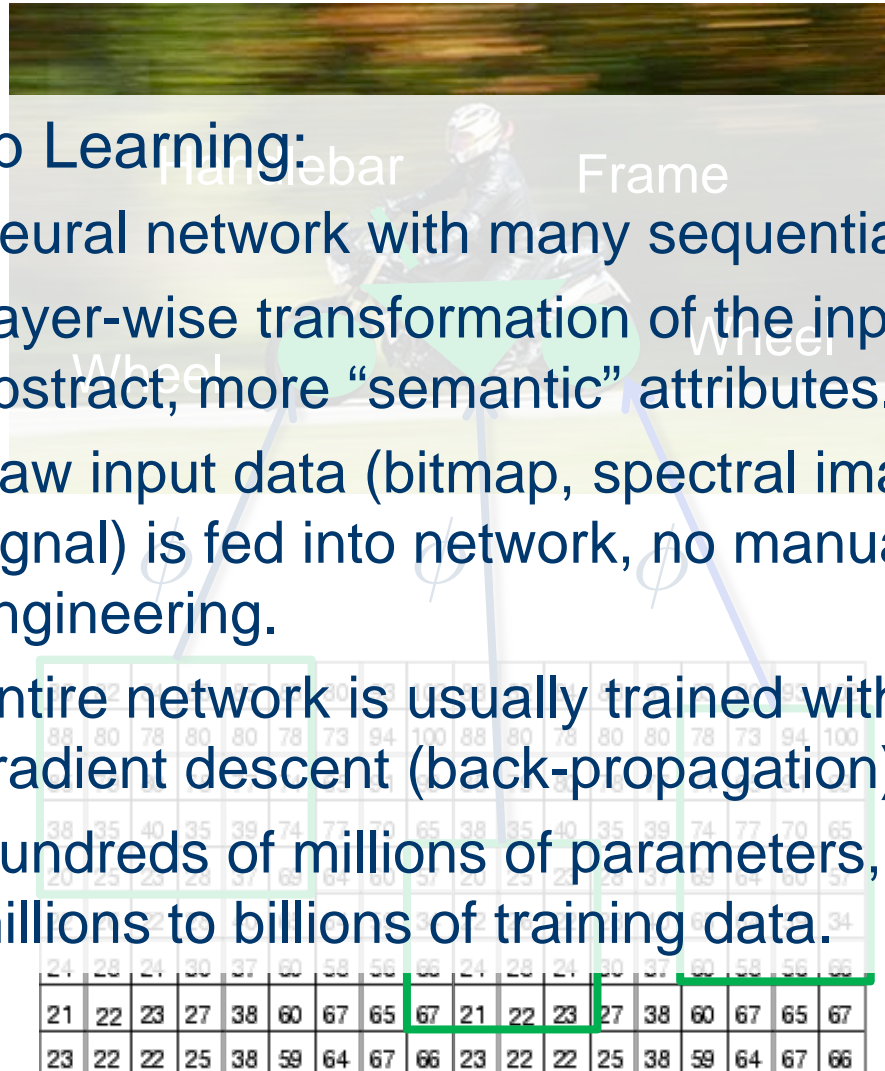
## ■ Deep Learning:

- Abstract features (higher level)** ♦ Neural network with many sequential layers.  
 ♦ Layer-wise transformation of the input into more abstract, more “semantic” attributes.

- ♦ Raw input data (bitmap, spectral image of audio signal) is fed into network, no manual feature engineering.

- ♦ Entire network is usually trained with stochastic gradient descent (back-propagation).

- Raw data (low-level)** ♦ Hundreds of millions of parameters, hundreds of millions to billions of training data.



# Neural Networks

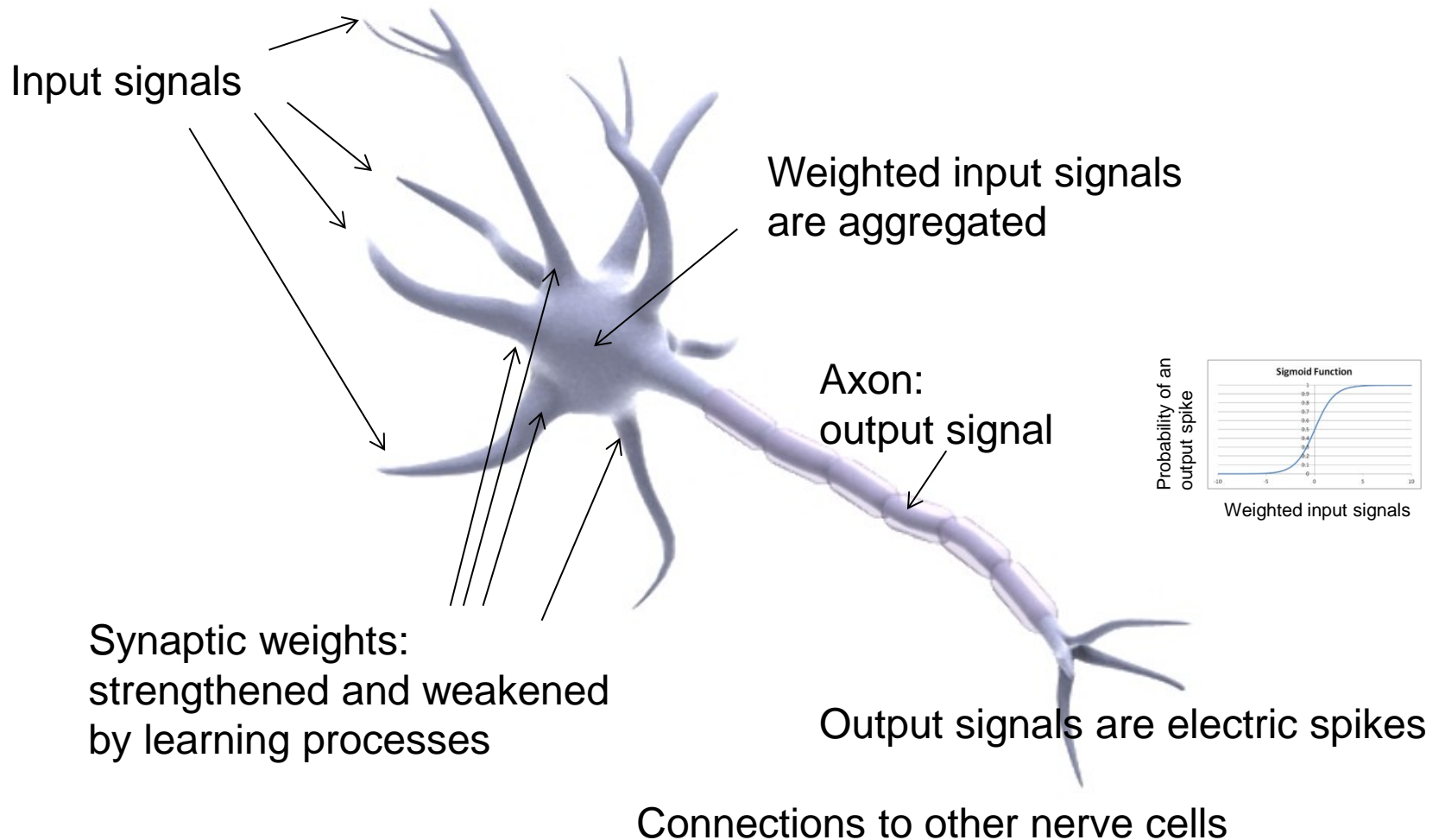
- Model of neural information processing
- Waves of popularity
  - ◆ ↑ Perceptron: Rosenblatt, 1960.
  - ◆ ↓ Perceptron only linear classifier (Minsky, Papert, 69).
  - ◆ ↑ Multilayer perceptrons (90s).
  - ◆ ↓ Popularity of SVMs (late 90s).
  - ◆ ↑ Deep learning (late 2000s).
  - ◆ Now state of the art for Speech Recognition image classification, face recognition and other problems.

# Deep Learning Records

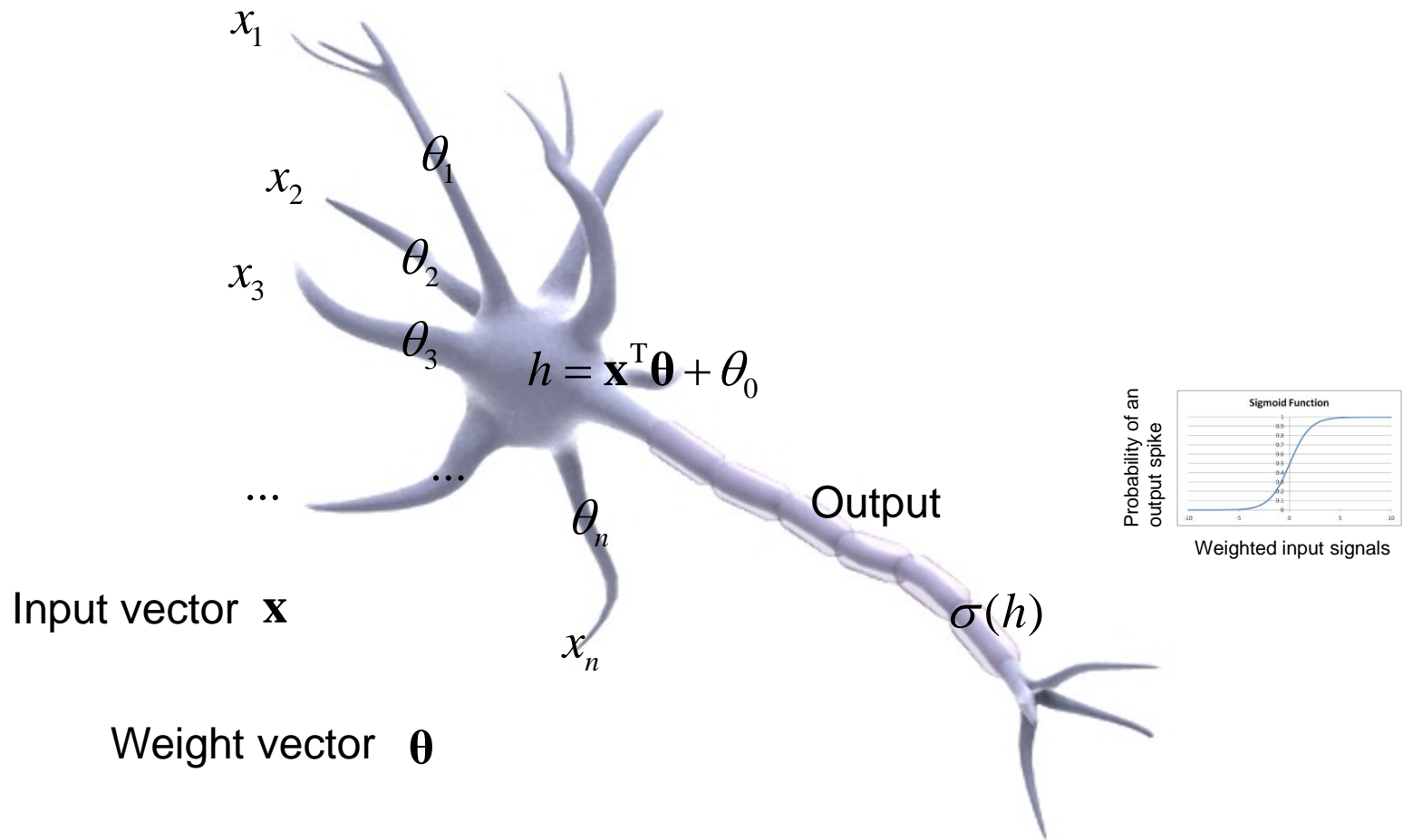
- Neural networks best-performing algorithms for
  - ◆ Object classification (CIFAR/NORB/PASCAL VOC-Benchmarks)
  - ◆ Video classification (various benchmarks)
  - ◆ Sentiment analysis (MR Benchmark)
  - ◆ Pedestrian detection
  - ◆ Face recognition
  - ◆ Speech recognition



# Neural Information Processing



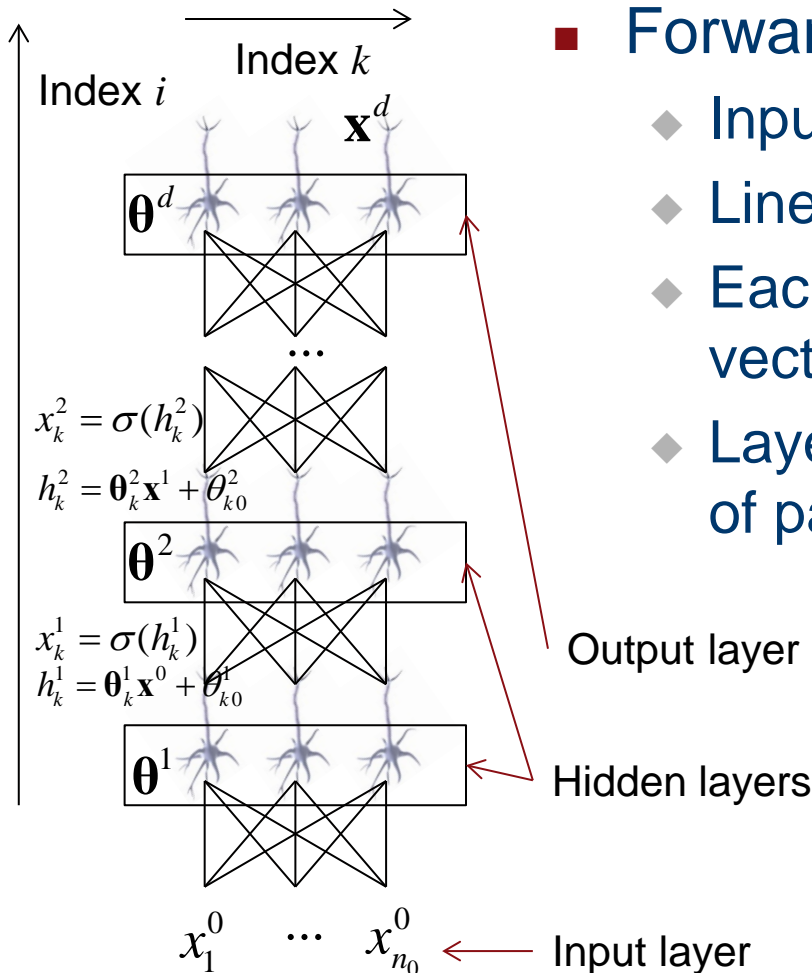
# Neural Information Processing: Model



# Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - ◆ Convolutional neural networks,
  - ◆ Recurrent neural networks.

# Feed Forward Networks

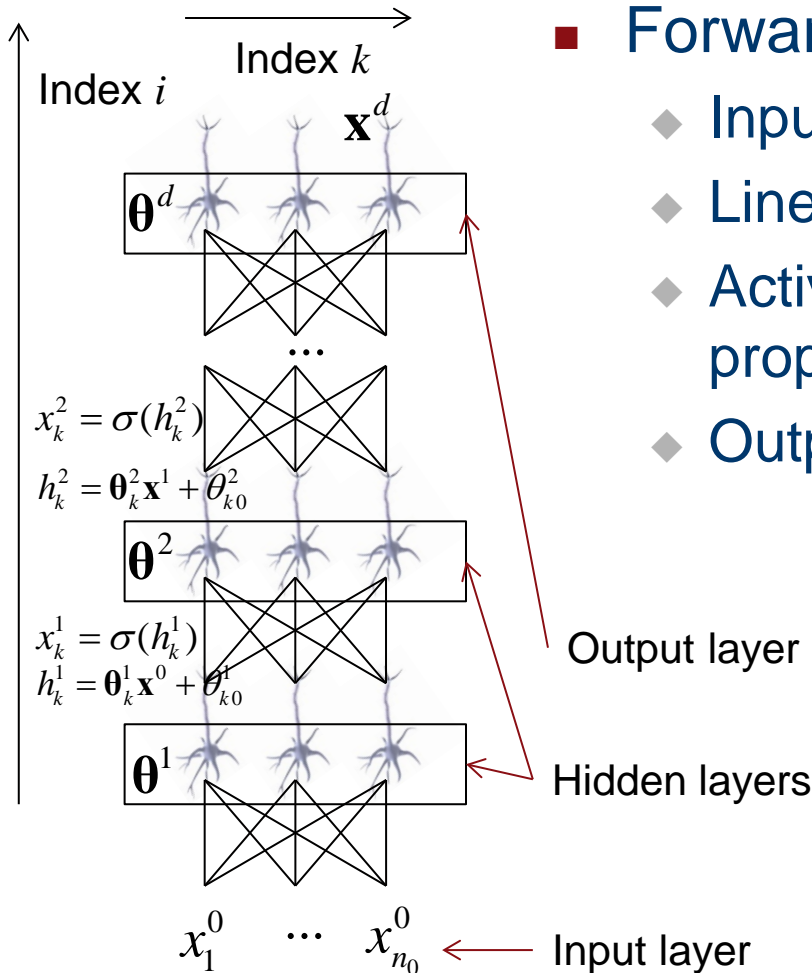


## ■ Forward propagation:

- ◆ Input vector  $\mathbf{x}^0$
- ◆ Linear model:  $h_k^i = \theta_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
- ◆ Each unit has parameter vector  $\theta_k^i = (\theta_{k1}^i \dots \theta_{kn_{i-1}}^i)$

- ◆ Layer  $i$  has matrix of parameters  $\theta^i = \begin{pmatrix} \theta_1^i \\ \vdots \\ \theta_{n_i}^i \end{pmatrix} = \begin{pmatrix} \theta_{11}^i & \dots & \theta_{1n_{i-1}}^i \\ \vdots & & \vdots \\ \theta_{n_i1}^i & \dots & \theta_{n_i n_{i-1}}^i \end{pmatrix}$

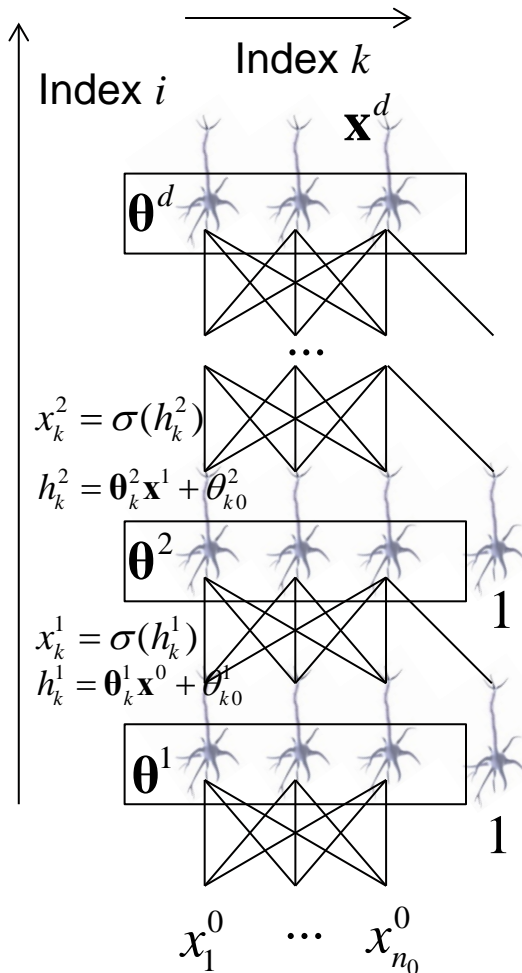
# Feed Forward Networks



## ■ Forward propagation:

- ◆ Input vector  $\mathbf{x}^0$
- ◆ Linear model:  $h_k^i = \theta_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
- ◆ Activation function and propagation:  $\mathbf{x}^i = \sigma(\mathbf{h}^i)$
- ◆ Output vector  $\mathbf{x}^d$

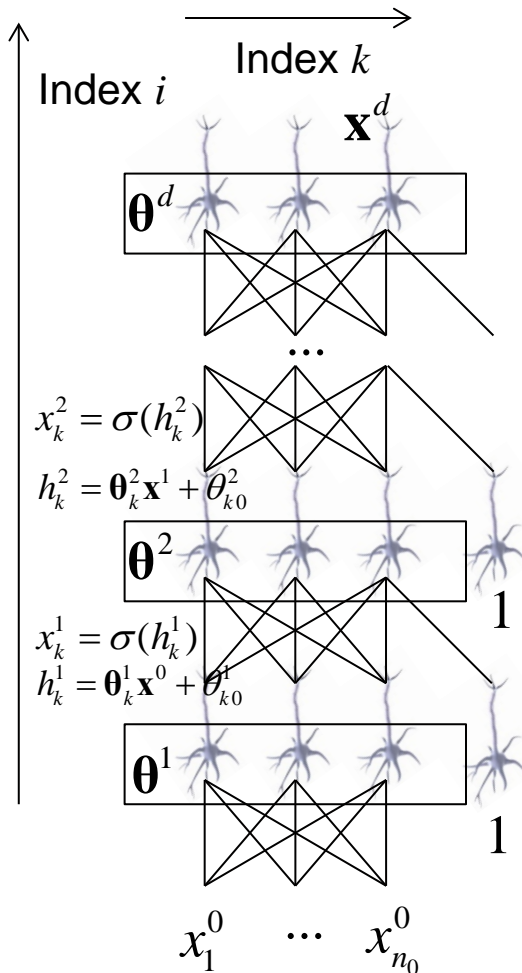
# Feed Forward Networks



## ■ Bias unit

- ◆ Linear model:  $h_k^i = \theta_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
- ◆ Constant element  $\theta_{k0}^i$  is replaced by additional unit with constant output 1:  $h_k^i = \theta_k^i \mathbf{x}_{[1..n_k+1]}^{i-1}$

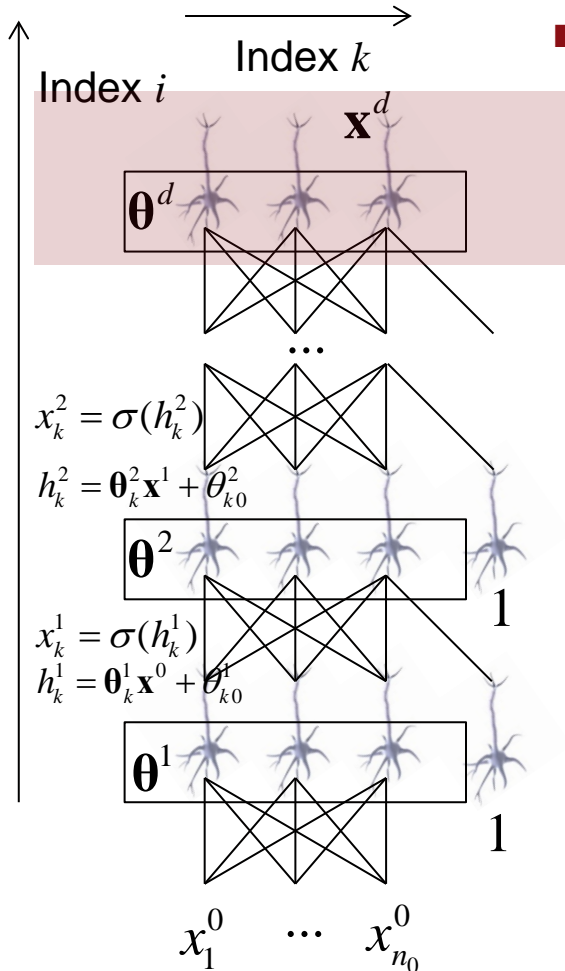
# Feed Forward Networks



- Forward propagation per layer in matrix notation:

- ◆ Linear model:  $\mathbf{h}^i = \boldsymbol{\theta}^i \mathbf{x}^{i-1}$
- ◆ Activation function:  $\mathbf{x}^i = \sigma(\mathbf{h}^i)$

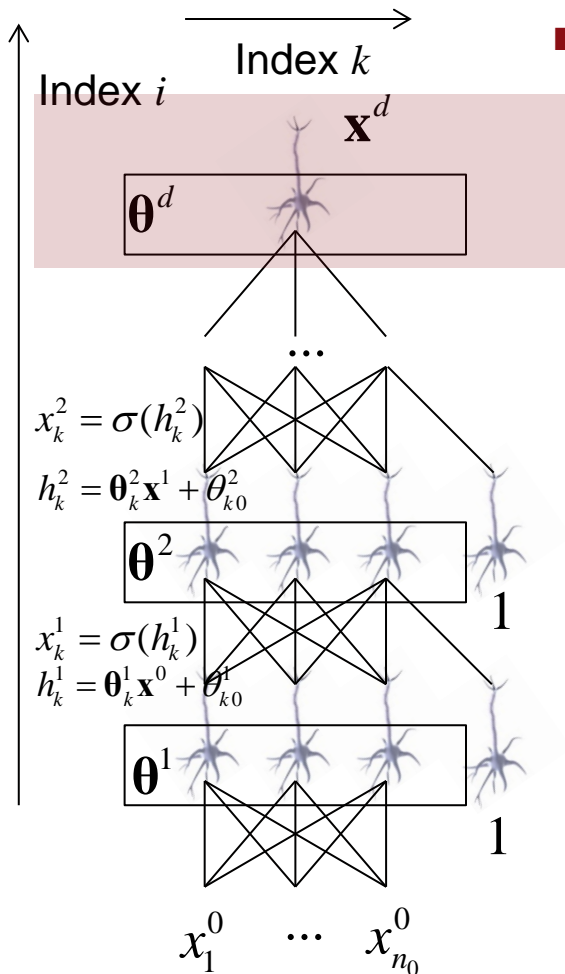
# Classification: Softmax Layer



■ One output unit per class:

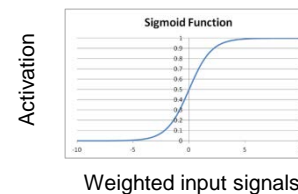
- ◆  $x_k^d = \sigma_{sm}(h_k^d) = \frac{e^{h_k^d}}{\sum_{k'} e^{h_{k'}^d}}$
- ◆  $x_k^d$ : predicted probability for class  $k$ .

# Regression: Sigmoidal Activation

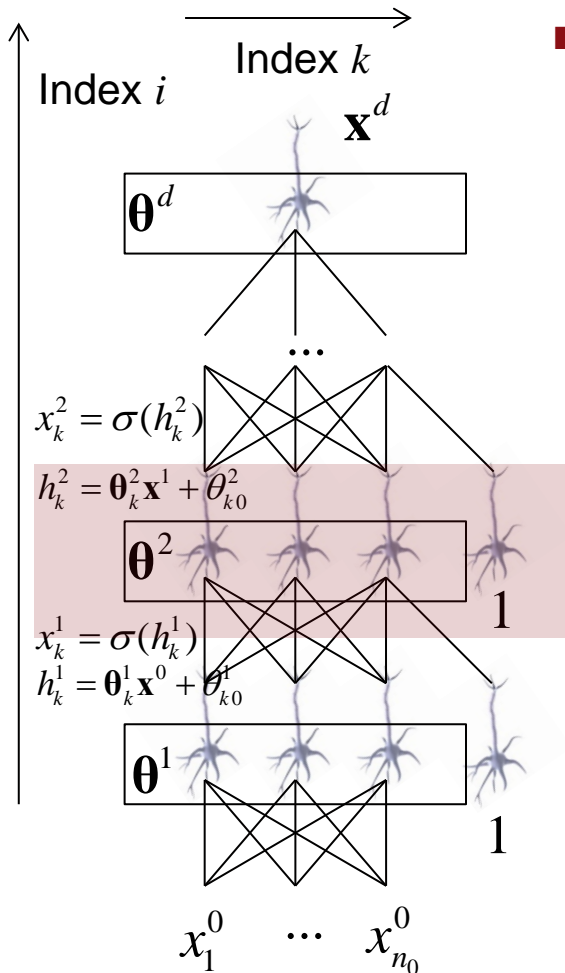


■ For target variable  $k$ :

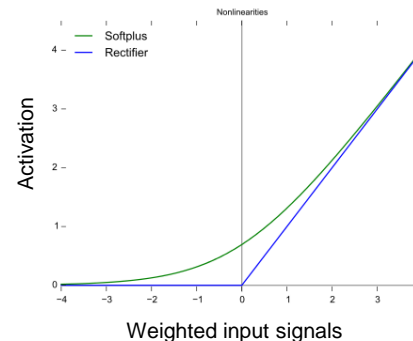
- ◆  $x_k^d = \sigma_s(h_k^d) = \frac{1}{1+e^{-h_k^d}}$
- ◆ Logistic activation function.



# Internal Units: Rectified Linear Units



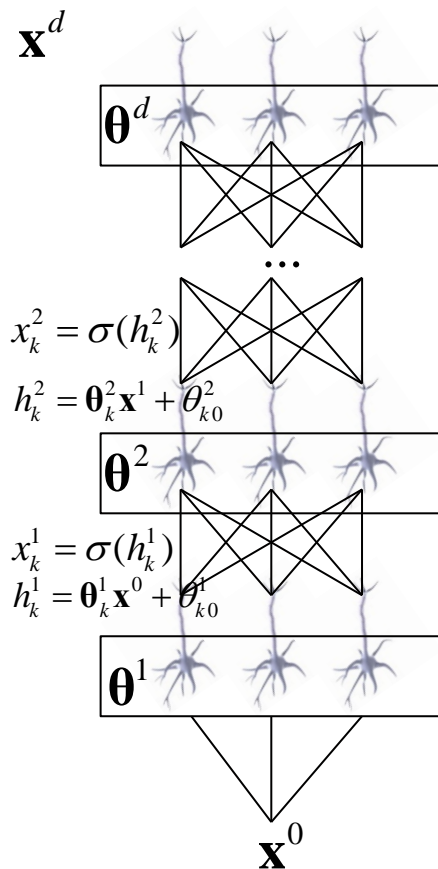
- For internal unit  $(i, k)$ :
  - ◆  $x_k^i = \sigma_{ReLU}(h_k^i) = \max(0, h_k^i)$
  - ◆ Leads to sparse activations and prevents the gradient from vanishing for deep networks.



# Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - ◆ Convolutional neural networks,
  - ◆ Recurrent neural networks.

# Feed Forward Networks: Learning



- Stochastic gradient descent

- Loss function

$$\hat{R}(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{j=1}^m \ell(\mathbf{y}_j, \mathbf{x}^d)$$

- Gradient descent:

$$\blacklozenge \boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla \hat{R}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta})$$

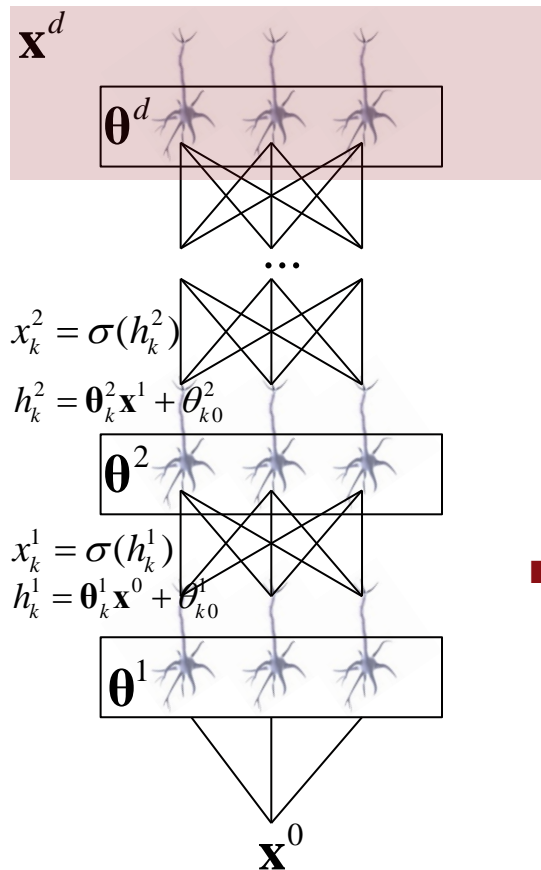
$$= \boldsymbol{\theta} - \frac{\alpha}{2m} \sum_{j=1}^m \frac{\partial}{\partial \boldsymbol{\theta}} \ell(\mathbf{y}_j, \mathbf{x}^d)$$

- Stochastic gradient for instance  $\mathbf{x}_j$ :

$$\blacklozenge \boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla_{\mathbf{x}_j} \hat{R}(\boldsymbol{\theta})$$

$$= \boldsymbol{\theta} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \ell(\mathbf{y}_j, \mathbf{x}^d)$$

# Learning: Back Propagation



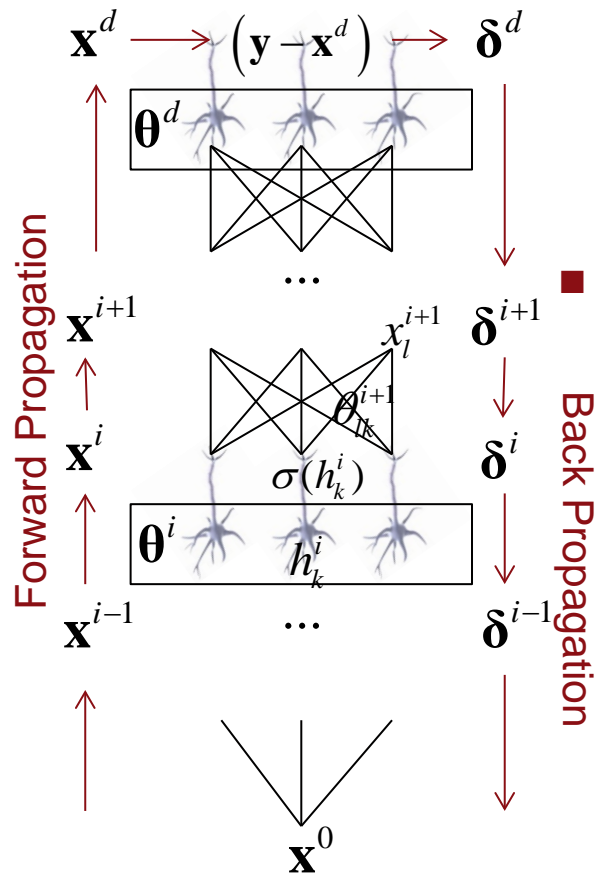
- Stochastic gradient for output units for instance  $\mathbf{x}_j$ :

$$\begin{aligned} \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial \theta_k^d} &= \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial \mathbf{x}^d} \frac{\partial \mathbf{x}^d}{\partial h_k^d} \frac{\partial h_k^d}{\partial \theta_k^d} \\ &= \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial \mathbf{x}^d} \frac{\partial \sigma(h_k^d)}{\partial h_k^d} \mathbf{x}^{d-1} \\ &= \delta_k^d \mathbf{x}^{d-1} \end{aligned}$$

- With

$$\delta_k^d = \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial \mathbf{x}^d} \frac{\partial \sigma(h_k^d)}{\partial h_k^d}$$

# Learning: Back Propagation



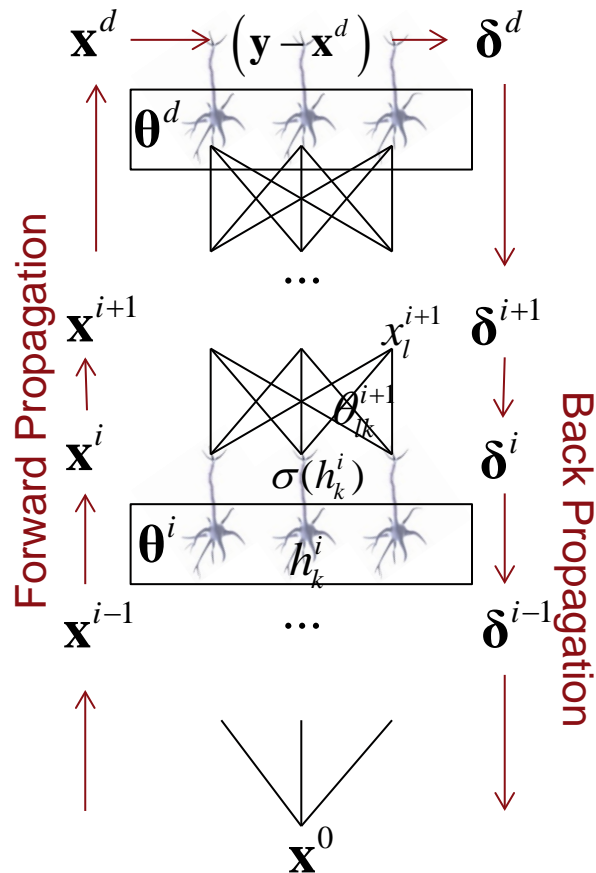
- Stochastic gradient for hidden units for instance  $\mathbf{x}_j$ :

$$\frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial \theta_k^i} = \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial h_k^i} \frac{\partial h_k^i}{\partial \theta_k^i} = \delta_k^i \mathbf{x}^{i-1}$$

- With

$$\begin{aligned} \delta_k^i &= \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial h_k^i} \\ &= \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial (\mathbf{x}_1^{i+1}, \dots, \mathbf{x}_{n_{i+1}}^{i+1})} \frac{\partial (\mathbf{x}_1^{i+1}, \dots, \mathbf{x}_{n_{i+1}}^{i+1})}{\partial h_k^i} \\ &= \sum_{l=1}^{n_{i+1}} \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial h_l^{i+1}} \frac{\partial h_l^{i+1}}{\partial x_k^i} \frac{\partial x_k^i}{\partial h_k^i} \\ &= \sum_{l=1}^{n_{i+1}} \delta_l^{i+1} \theta_{lk}^{i+1} \frac{\partial \sigma(h_k^i)}{\partial h_k^i} \end{aligned}$$

# Learning: Back Propagation



- Derivative of the loss function for classification.

- Softmax activation function:

$$\diamond x_k^d = \sigma_{sm}(h_k^d) = \frac{e^{h_k^d}}{\sum_{k'} e^{h_{k'}^d}}$$

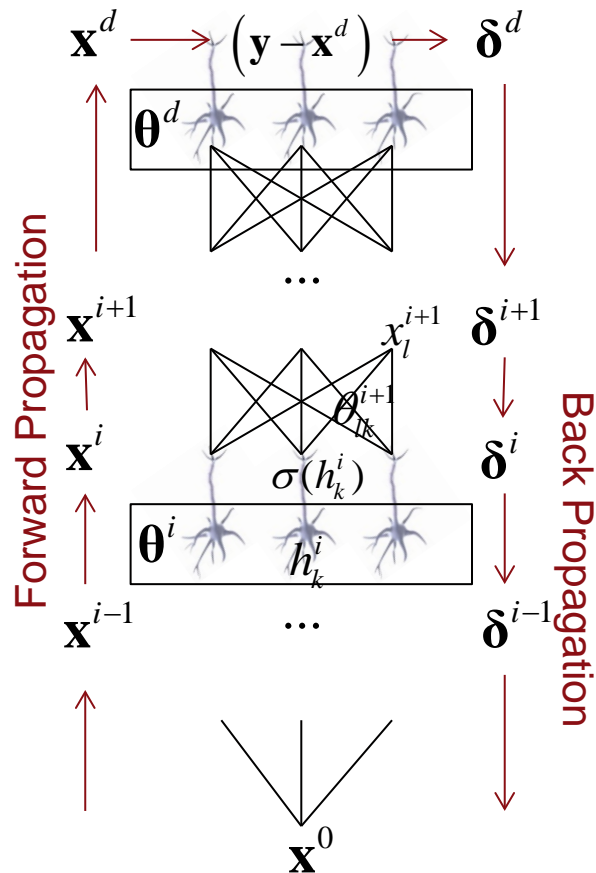
$$\diamond \frac{\partial \sigma_{sm}(h_k^d)}{\partial h_k^d} = \sigma_{sm}(h_k^d)(1 - \sigma_{sm}(h_k^d))$$

- Cost function:

$$\diamond \ell(\mathbf{y}, \mathbf{x}^d) = \sum_k y_k \log x_k^d$$

$$\diamond \frac{\partial \ell(\mathbf{y}, \mathbf{x}^d)}{\partial h_k^d} = x_k^d - y_k$$

# Learning: Back Propagation



- Derivative of the loss function for regression.

- Sigmoidal activation function:

- ◆  $x_k^d = \sigma_s(h_k^d) = \frac{1}{1+e^{-h_k^d}}$

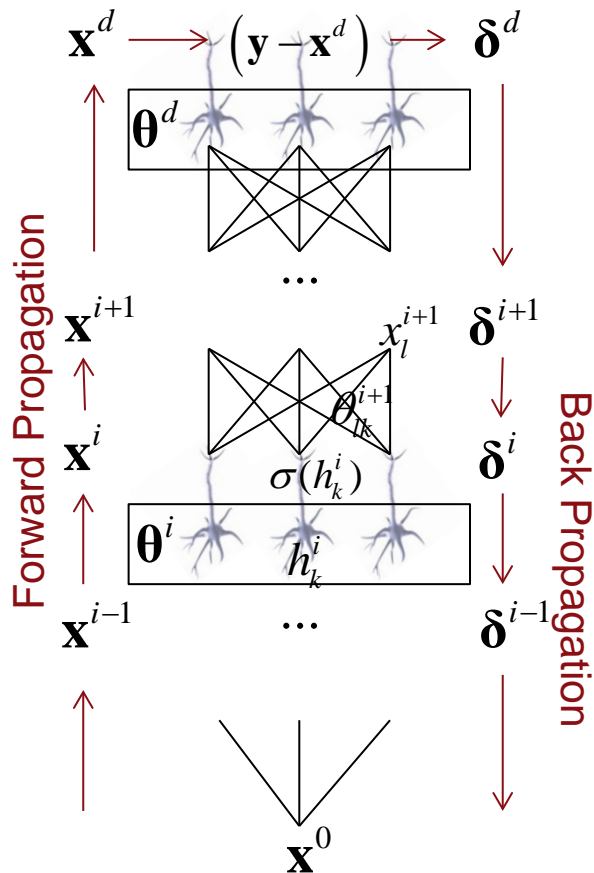
- ◆  $\frac{\partial \sigma_s(h_k^d)}{\partial h_k^d} = \sigma_s(h_k^d)(1 - \sigma_s(h_k^d))$

- Cost function:

- ◆  $\ell(\mathbf{y}, \mathbf{x}^d) = \frac{1}{2} \sum_k (x_k^d - y_k)^2$

- ◆  $\frac{\partial \ell(\mathbf{y}, \mathbf{x}^d)}{\partial x_k^d} = x_k^d - y_k$

# Learning: Back Propagation



- Derivative of the activation function for internal units.

- Rectified linear activation function:

- ◆  $x_k^i = \sigma_{ReLU}(h_k^i) = \max(0, h_k^i)$

- ◆  $\frac{\partial \sigma_{ReLU}(h_k^i)}{\partial h_k^i} = \begin{cases} 1 & \text{if } h_k^i > 0 \\ 0 & \text{otherwise} \end{cases}$

# Back Propagation: Algorithm

- Iterate over training instances  $(\mathbf{x}, \mathbf{y})$ :

- ◆ Forward propagation: for  $i=0\dots d$ :

- ★ For  $k=1\dots n_i$ :  $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$

- ★  $\mathbf{x}^i = \sigma(\mathbf{h}^i)$

- ◆ Back propagation:

- ★ For  $k=1\dots n_i$ :  $\delta_k^d = \frac{\partial}{\partial h_k^d} \sigma(h_k^d) \frac{\partial}{\partial x_k^d} \ell(y_k, x_k^d)$

- $\boldsymbol{\theta}_k^{d'} = \boldsymbol{\theta}_k^d - \alpha \delta_k^d \mathbf{x}^{d-1}$

- ★ For  $i=d-1\dots 1$ :

- For  $k=1\dots n_i$ :  $\delta_k^i = \sigma'(h_k^i) \sum_l \delta_l^{i+1} \theta_{lk}^{i+1}$

- $\boldsymbol{\theta}_k^{i'} = \boldsymbol{\theta}_k^i - \alpha \delta_k^i \mathbf{x}^{i-1}$

- Until convergence

# Back Propagation

- Loss function is not convex
  - ◆ Each permutation of hidden units is a local minimum.
  - ◆ Learned features (hidden units) may be ok, but not usually globally optimal.
- But:
  - ◆ Local minima can still be arbitrarily good.
  - ◆ Many local minima can be equally good.
  - ◆ Supervised learning often works with hundreds of layers and millions of training instances.

# Regularization

- L2-regularized loss

- ◆  $\hat{R}_2(\boldsymbol{\theta}) = \frac{1}{2m} \sum_j (\mathbf{y}_j - \mathbf{x}_j^d)^2 + \frac{\eta}{2} \boldsymbol{\theta}^T \boldsymbol{\theta}$

- ◆ Corresponds to normal prior on parameters.

- Gradient:  $\nabla \hat{R}_2(\boldsymbol{\theta}^i) = \frac{1}{m} \sum_j \delta_j^i \mathbf{x}^i + \eta \boldsymbol{\theta}$

- Update:  $\boldsymbol{\theta}' = \boldsymbol{\theta} - \delta_j \mathbf{x} - \eta \boldsymbol{\theta}$

- Called *weight decay*.

- Additional regularization schemes:

- ◆ Early stopping (outdated): Stop before convergence.

- ◆ Delete units with small weights.

- ◆ Dropout: During training, set some units' output to zero at random.

# Regularization: Dropout

- In complex networks, complex co-adaptation relationships can form between units.
  - ◆ Not robust for new data.
- Dropout: In each training set, draw a fraction of units at random and set their output to zero.
- At application time, use all units.
- Improves overall robustness: each unit has to function within varying combinations of units.

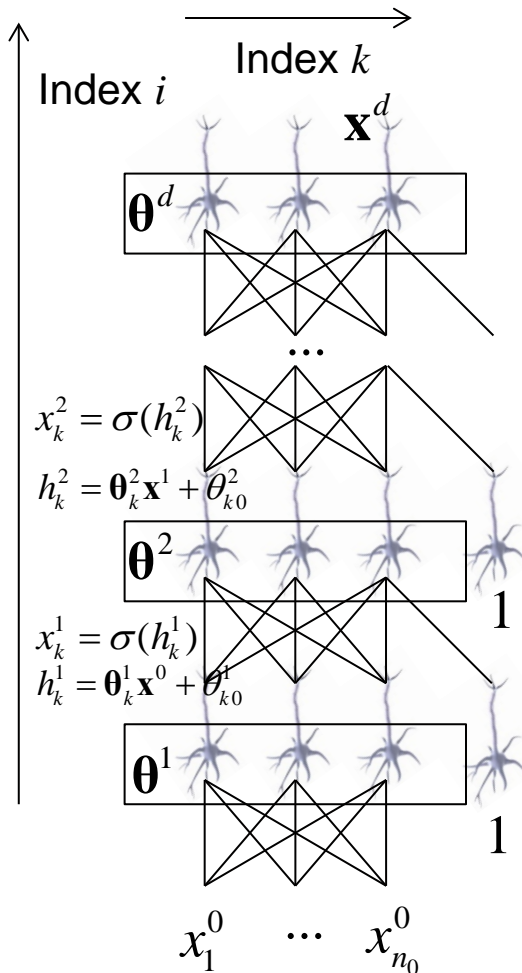
# Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - ◆ Convolutional neural networks,
  - ◆ Recurrent neural networks.

# Parallel Inference

- Both forward and backward propagation can be made much faster by parallel computation.
- GPUs are particularly suited.
- Pipelining in single-core CPU can be exploited.
- Forward- and backward-propagation can be written as matrix multiplications.
- Columns of the weight matrix can be processed in parallel.

# Parallel Inference



- Forward propagation per layer in matrix notation:

$$\mathbf{h}^i = \boldsymbol{\theta}^i \mathbf{x}^{i-1}$$

$$\begin{bmatrix} h_1^i \\ \vdots \\ h_{n_i}^i \end{bmatrix} = \begin{bmatrix} \theta_{11}^i & \dots & \theta_{1n_{i-1}}^i \\ \vdots & & \vdots \\ \theta_{n_i1}^i & \dots & \theta_{n_i n_{i-1}}^i \end{bmatrix} \begin{bmatrix} x_1^{i-1} \\ \vdots \\ x_{n_{i-1}}^{i-1} \end{bmatrix}$$

- Use vector coprocessor / GPU row-by-column multiplications.
- Split up rows of  $\boldsymbol{\theta}$  between multiple cores.

# Software Packages


- Caffe: allows to easily apply deep learning with standard units, loss functions, learning techniques.
- Deep learning frameworks that allow development of new architectures and learning techniques.
  - ◆ Torch,
  - ◆ Theano (+Lasagne),
  - ◆ TensorFlow (+Keras),
  - ◆ CNTK.

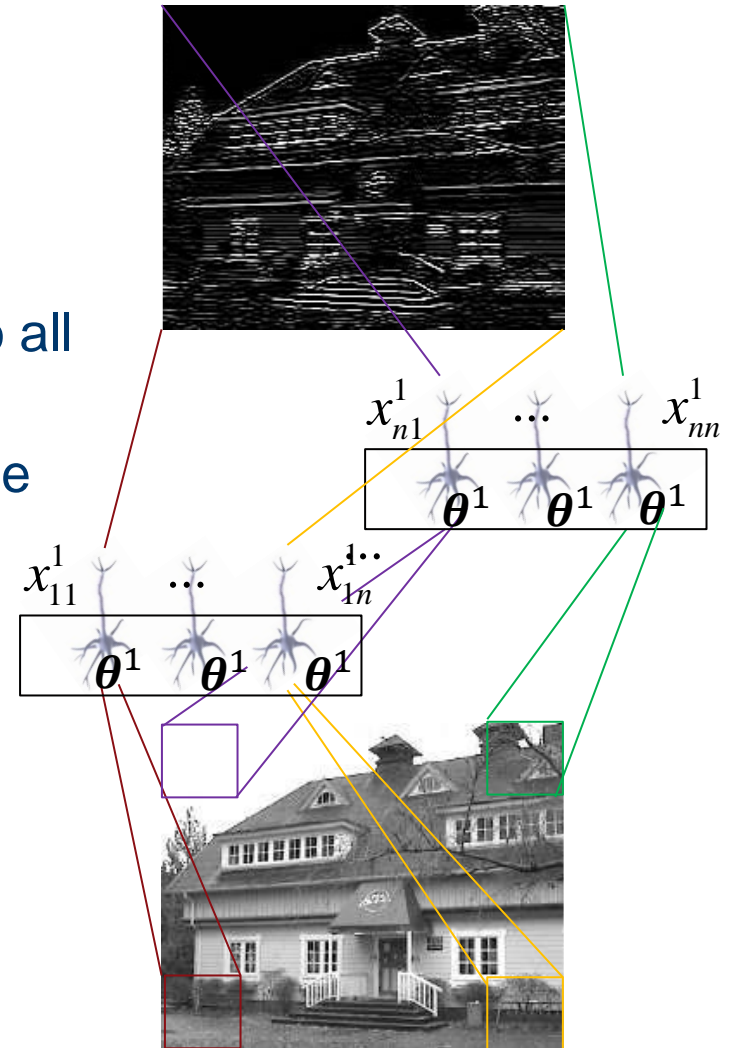
# Overview

- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - ◆ Convolutional neural networks,
  - ◆ Recurrent neural networks.

# Convolution

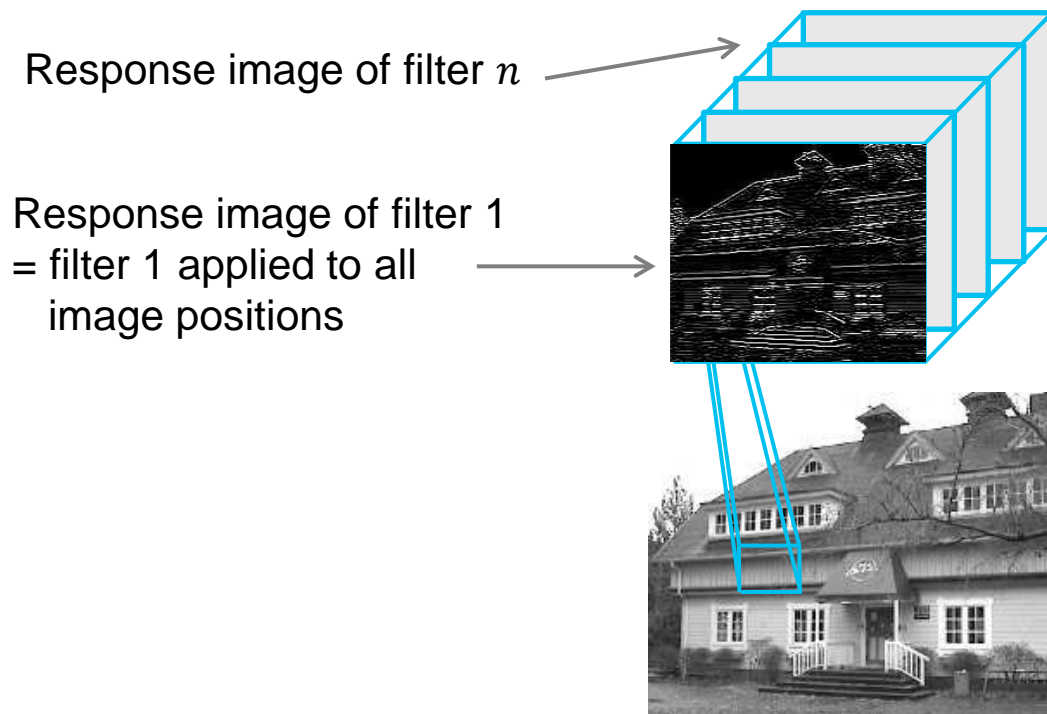
- Filter unit takes window of input, produces response.
- For instance, a filter can detect a specific type of patterns.
- Same filter is applied in parallel to all window positions of the input.
- Each unit produces output of same filter for different position.
- This produces a response image.

$$\theta^1 =$$




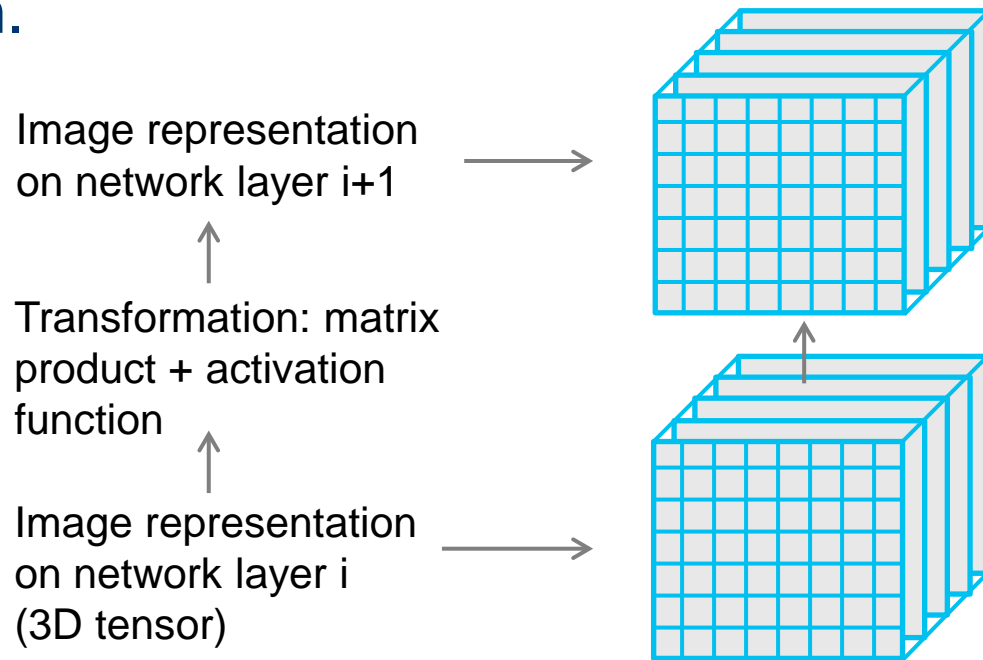
# Convolutional Layer

- Applies multiple filters to all positions of input from layer below.
- Creates a bank of response images.



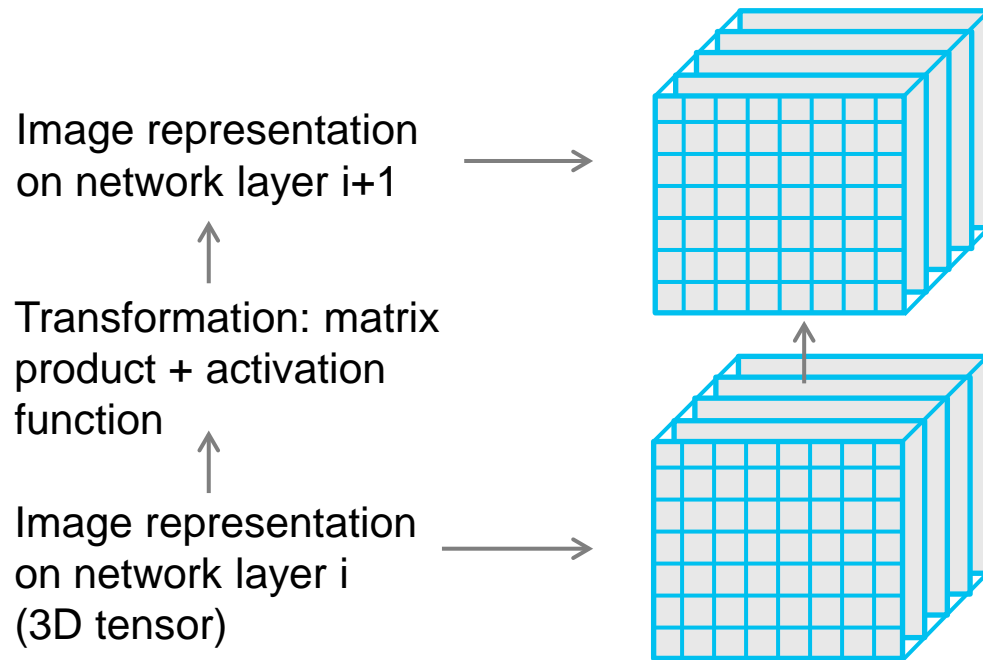
# Convolutional Neural Networks

- Each network layer transforms the image into a more abstract feature representation.
- Top-most layer can be softmax units for classification.



# Convolutional Neural Networks

- Widely used in image processing; e.g.,
  - ◆ ImageNet (general image classification),
  - ◆ DeepFace (face recognition).



# Overview

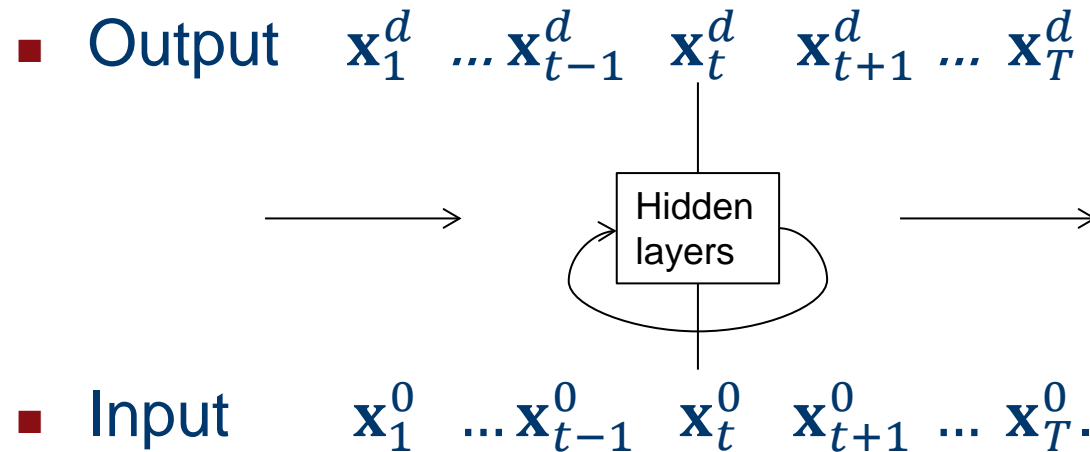
- Neural information processing.
- Deep learning.
- Feed-forward networks.
- Training feed-forward networks: back propagation.
- Parallel inference on GPUs.
- Outlook:
  - ◆ Convolutional neural networks,
  - ◆ Recurrent neural networks.

# Recurrent Neural Networks

- Input: time series  $\mathbf{x}_1^0, \dots, \mathbf{x}_T^0$ .
- Output can be:
  - ◆ One output (vector) for entire time series:  $\mathbf{x}^d$ .
  - ◆ One output at each time step:  $\mathbf{x}_1^d, \dots, \mathbf{x}_T^d$ .

# Recurrent Neural Networks

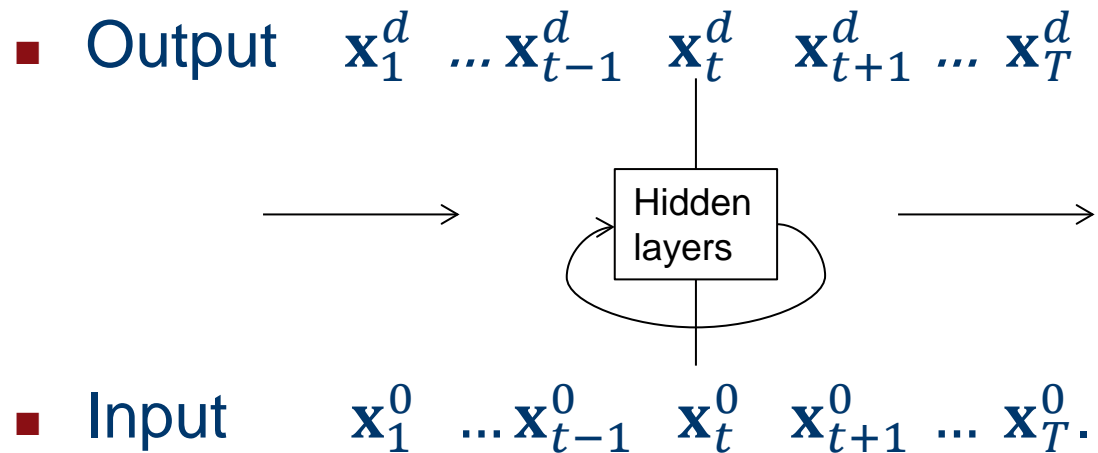
- Hidden layer propagates information to itself.
- Hidden layer activation stores context information.



# Recurrent Neural Networks

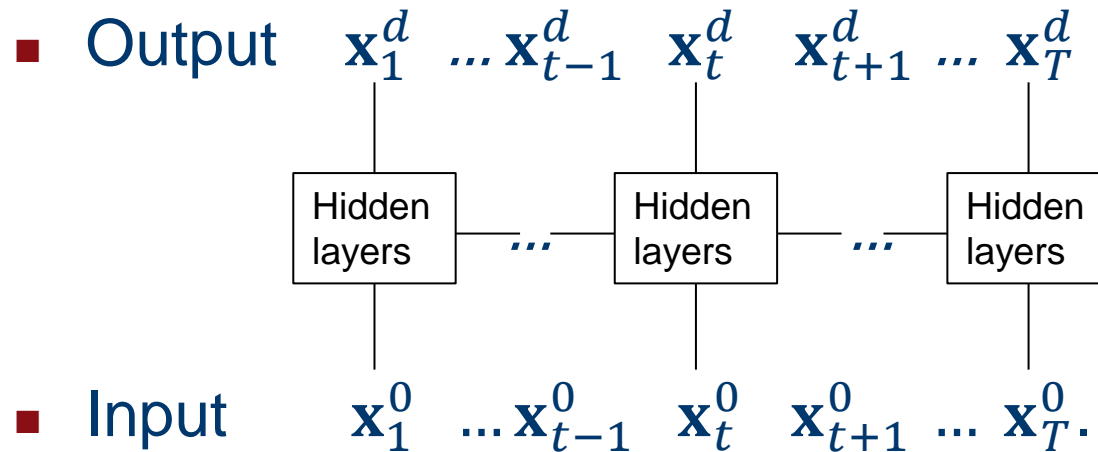
- Propagation:

- ◆  $\mathbf{h}_t^1 = \theta^1 \begin{pmatrix} \mathbf{x}_t^0 \\ \mathbf{x}_{t-1}^1 \end{pmatrix}; \mathbf{x}_t^1 = \sigma(\mathbf{h}_t^1)$



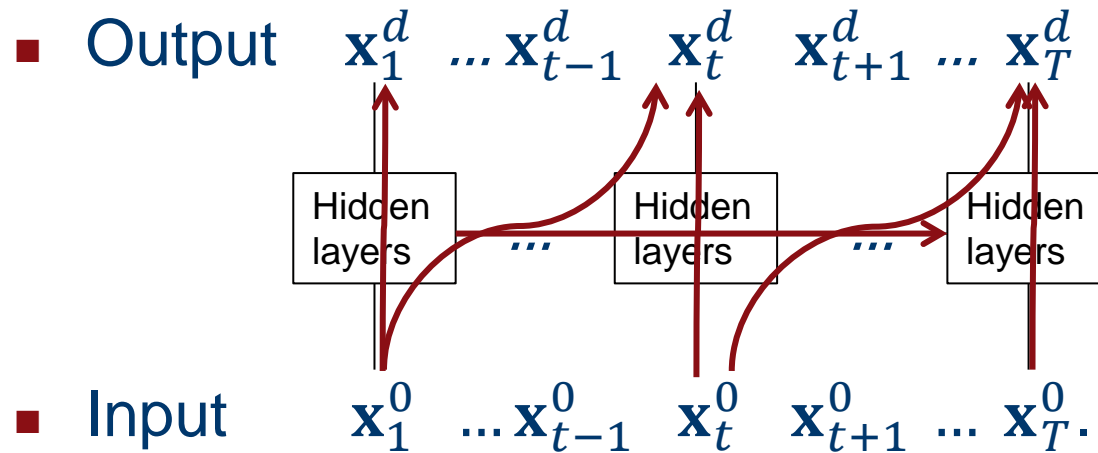
# Recurrent Neural Networks

- Identical network “unfolded” in time.
- Units on hidden layer propagate to the right.
- Hidden layer activation stores context information.



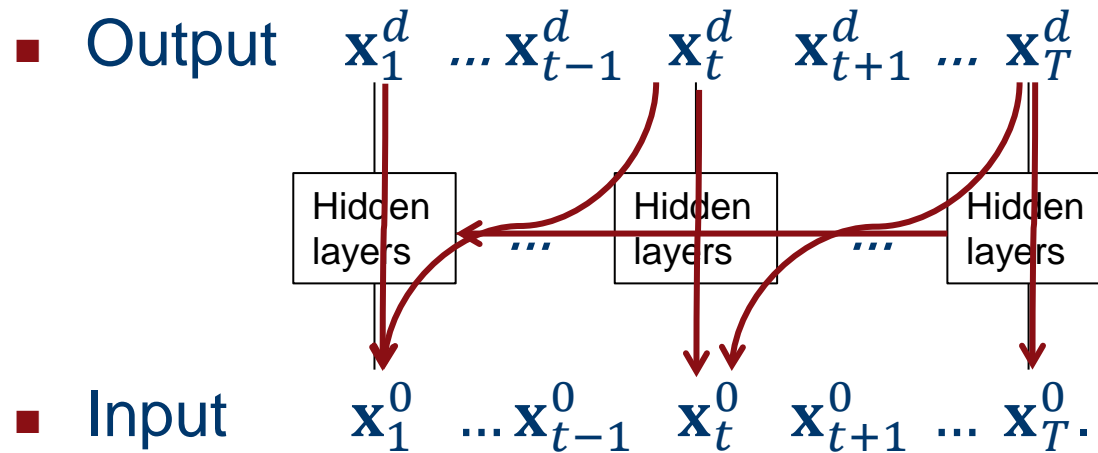
# Recurrent Neural Networks

- Forward propagation



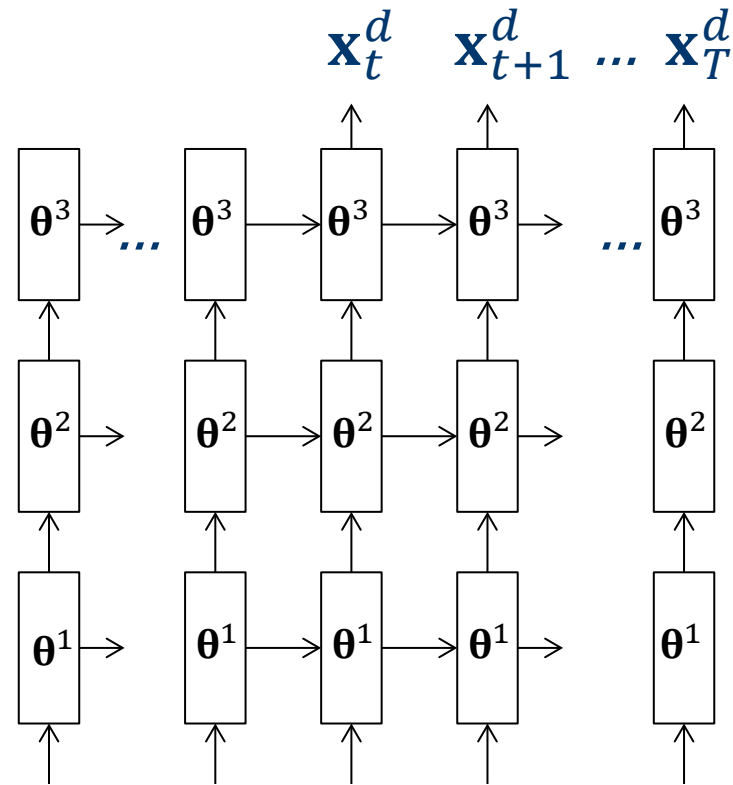
# Recurrent Neural Networks

- Back propagation through time.



# Deep Recurrent Neural Networks

## ■ Output



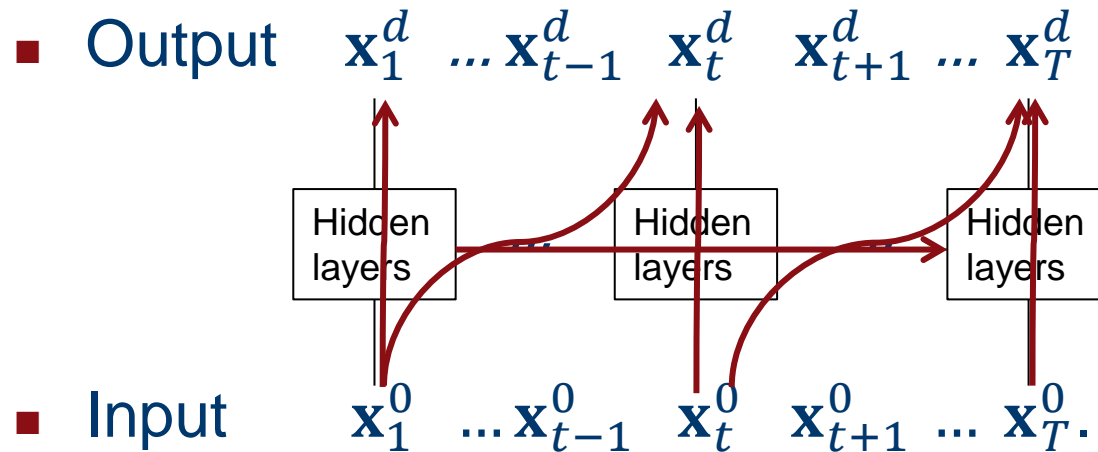
Many paths through which long-term dependencies can be propagated through the network

## ■ Input

$\mathbf{x}_1^0 \dots \mathbf{x}_{t-1}^0 \mathbf{x}_t^0 \mathbf{x}_{t+1}^0 \dots \mathbf{x}_T^0$

# Recurrent Neural Networks

- Widely used in language processing:
  - ◆ Speech recognition,
  - ◆ Machine translation,



# Summary

- Computational model of neural information processing.
- Feed-forward networks: layer-wise matrix multiplication + activation function.
- Back propagation: stochastic gradient descent. Gradient computation by layer-wise matrix multiplication + derivative of activation function.
- Convolutional neural networks: layer-wise application of local filters.
- Recurrent neural networks: state information passed on to next time step.