Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Neural Networks

Tobias Scheffer

Overview

- Neural information processing.
- Feed-forward networks.
- Training feed-forward networks, back propagation.
- Recurrent neural networks.
- LSTM networks.

Learning Problems can be Impossible without the Right Features

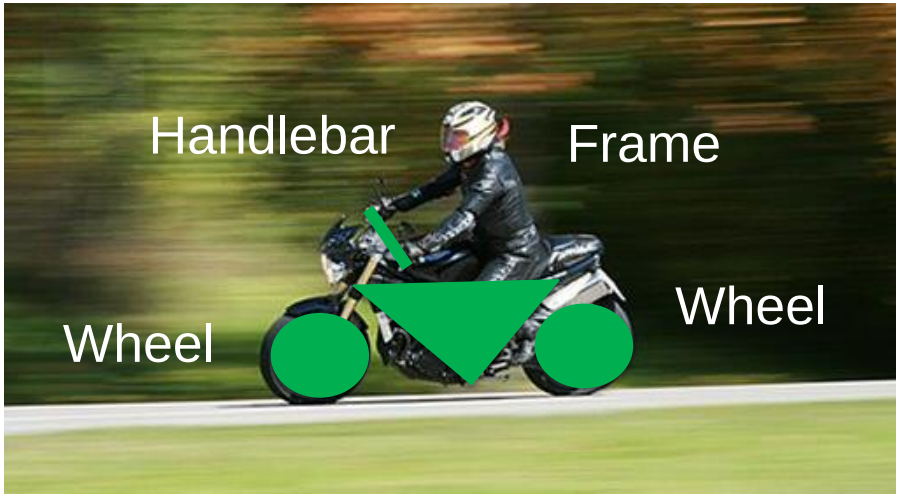
→ motorcycle

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

?

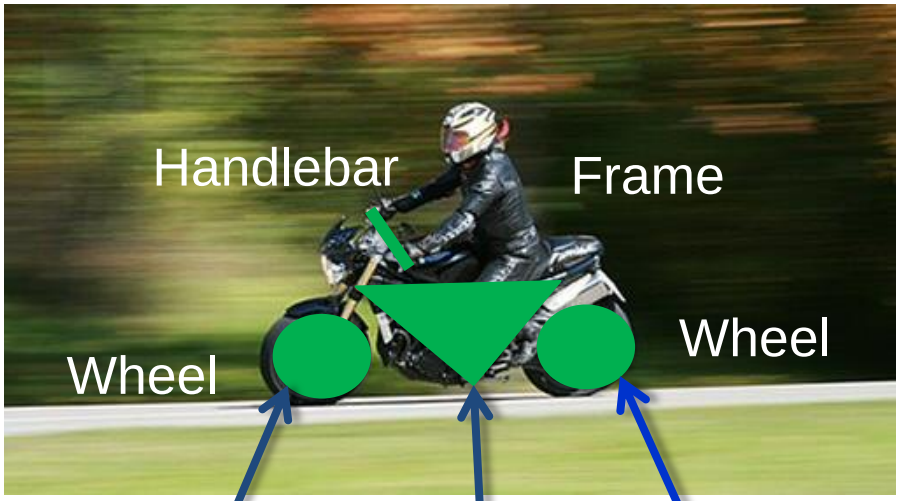
→ motorcycle

Learning Problems can be Impossible without the Right Features

Abstract features (higher level)

ϕ

Raw data (low-level)



ϕ ϕ ϕ

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

Feature funktion

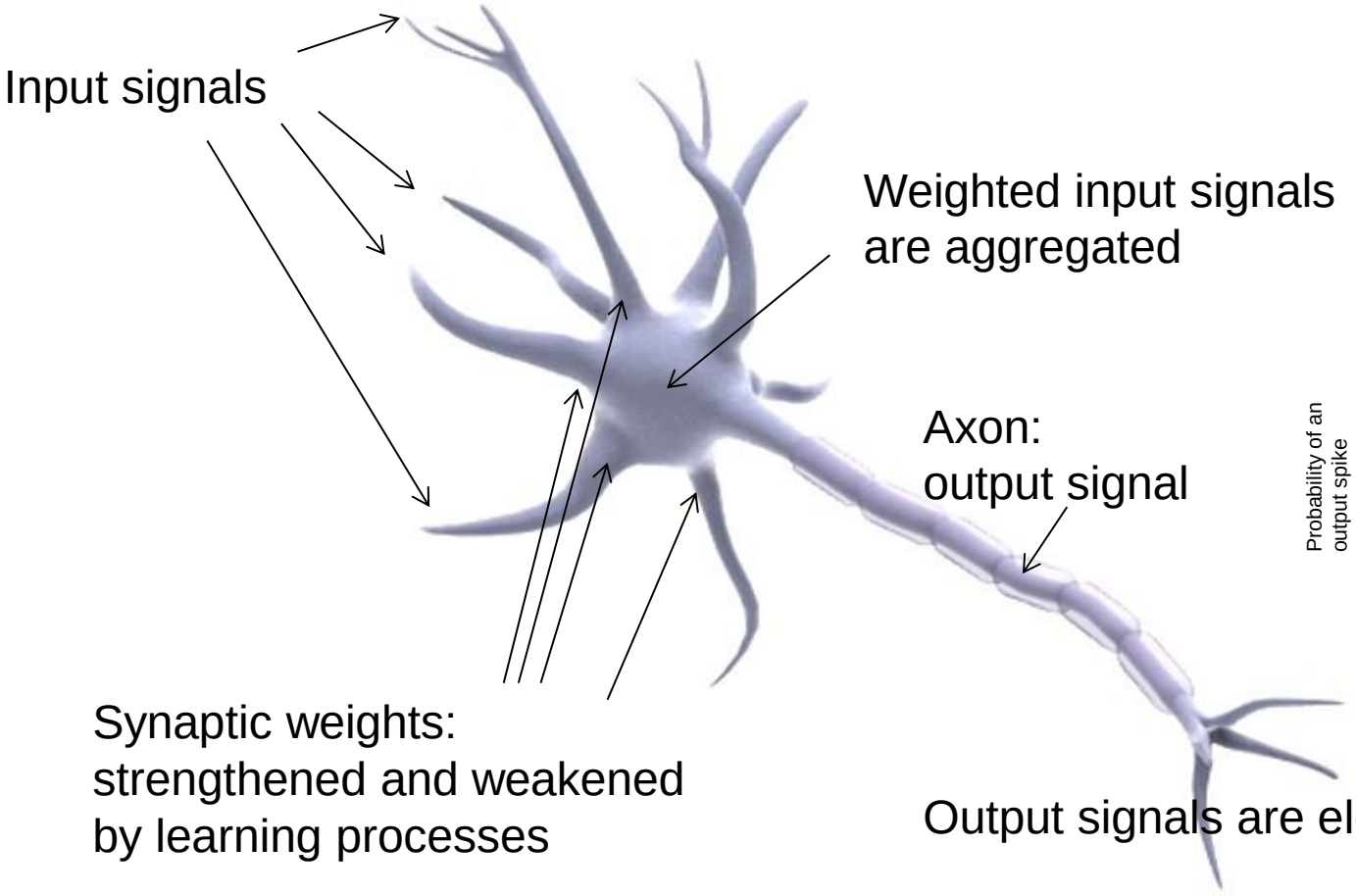
Neural Networks

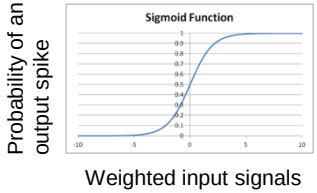
- Model of neural information processing
- Waves of popularity
  - ↑ Perceptron: Rosenblatt, 1960.
  - ↓ Perceptron only linear classifier (Minsky, Papert, 69).
  - ↑ Multilayer perceptrons (90s).
  - ↓ Popularity of SVMs (late 90s).
  - ↑ Deep learning (late 2000s).
  - Now state of the art for Voice Recognition (Google DeepMind), Face Recognition (Deep Face, 2014)

Deep Learning Records

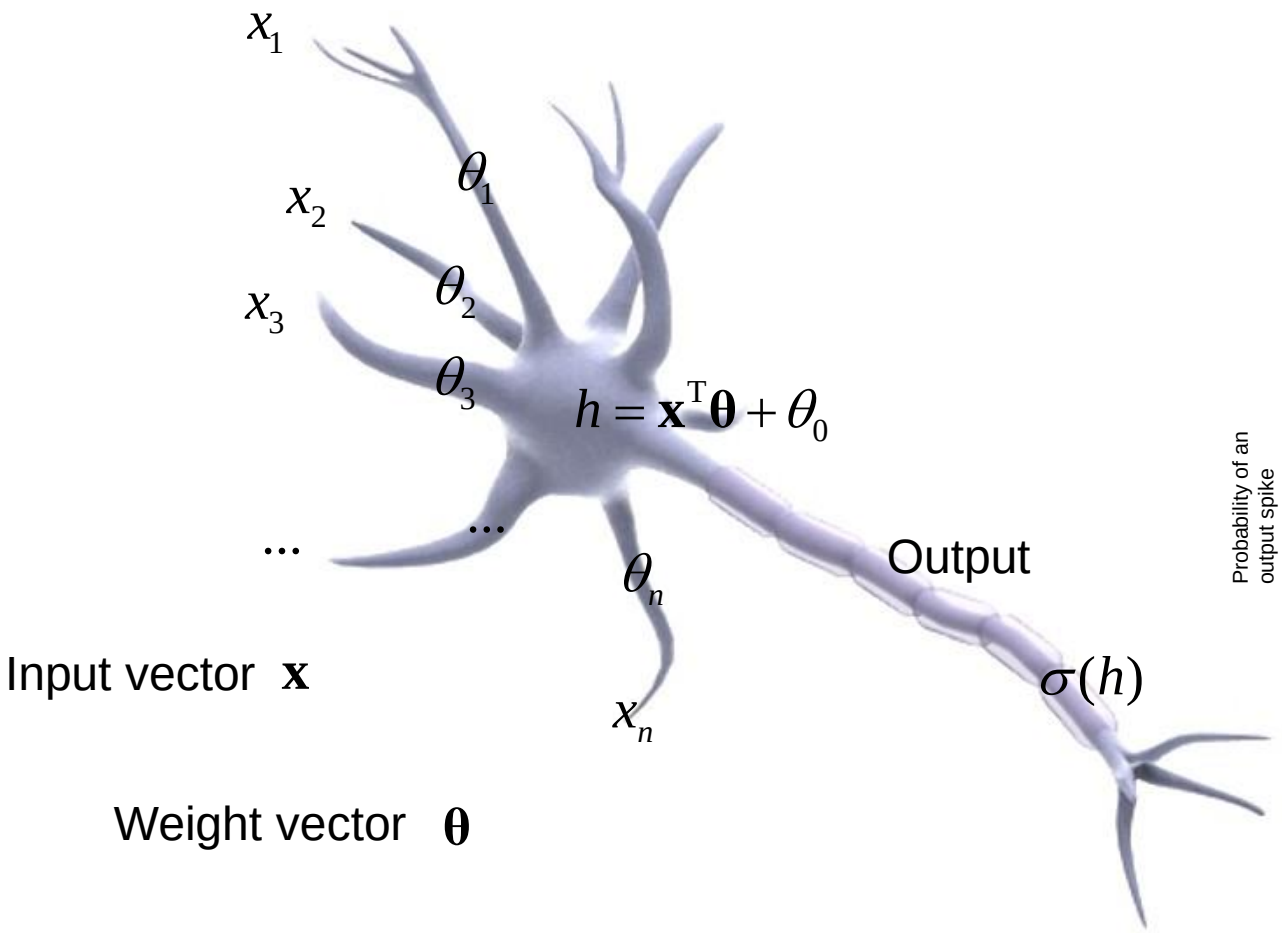
- Neural networks best-performing algorithms for
  - Object classification (CIFAR/NORB/PASCAL VOC-Benchmarks)
  - Video classification (various benchmarks)
  - Sentiment analysis (MR Benchmark)
  - Pedestrian detection
  - Speech recognition
  - Phychedelic art (Deep Dream)

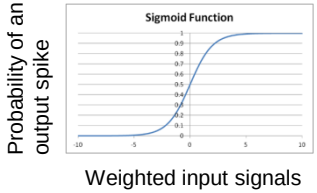
Neural Information Processing



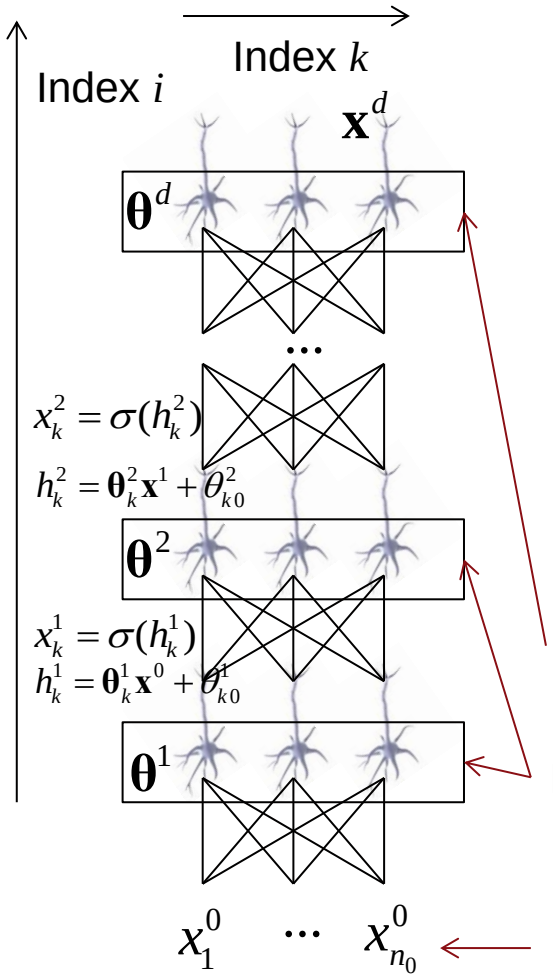


Neural Information Processing: Model





Feed Forward Networks



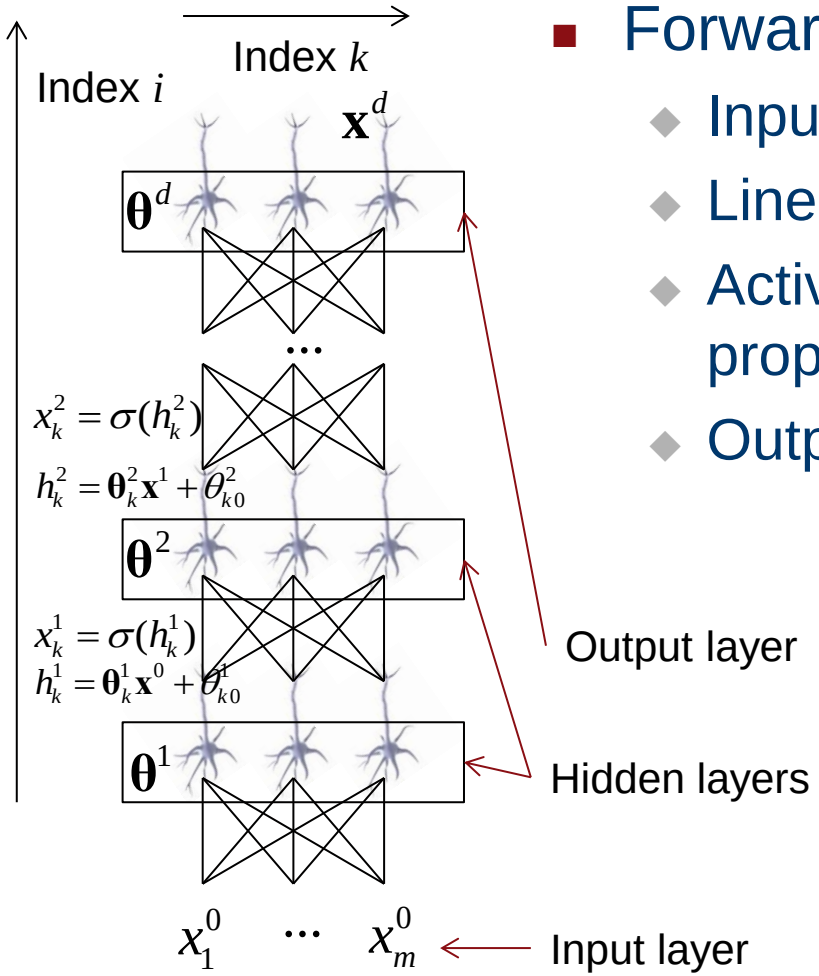
Output layer

Hidden layers

Input layer

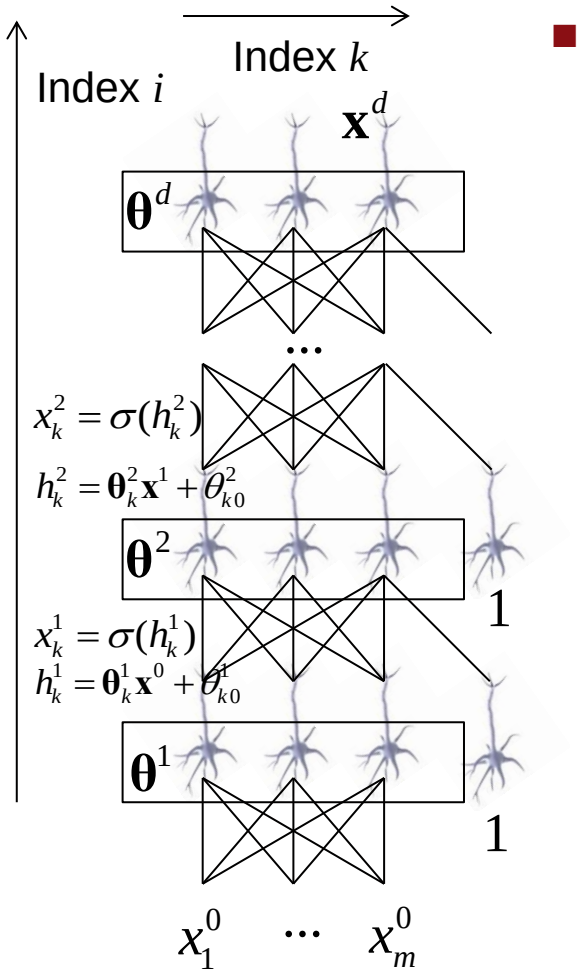
- Forward propagation:
  - Input vector $\mathbf{x}^0$
  - Linear model: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
  - Each unit has parameter vector $\boldsymbol{\theta}_k^i = \begin{pmatrix} \theta_1^i & \dots & \theta_{n_i}^i \end{pmatrix}$
  - Layer i has matrix of parameters $\boldsymbol{\theta}^i = \begin{pmatrix} \boldsymbol{\theta}_1^i \\ \\ \boldsymbol{\theta}_{n_i}^i \end{pmatrix} = \begin{pmatrix} \theta_{11}^i & & \theta_{1n_{i-1}}^i \\ \\ \theta_{n_i 1}^i & & \theta_{n_i n_{i-1}}^i \end{pmatrix}$

Feed Forward Networks



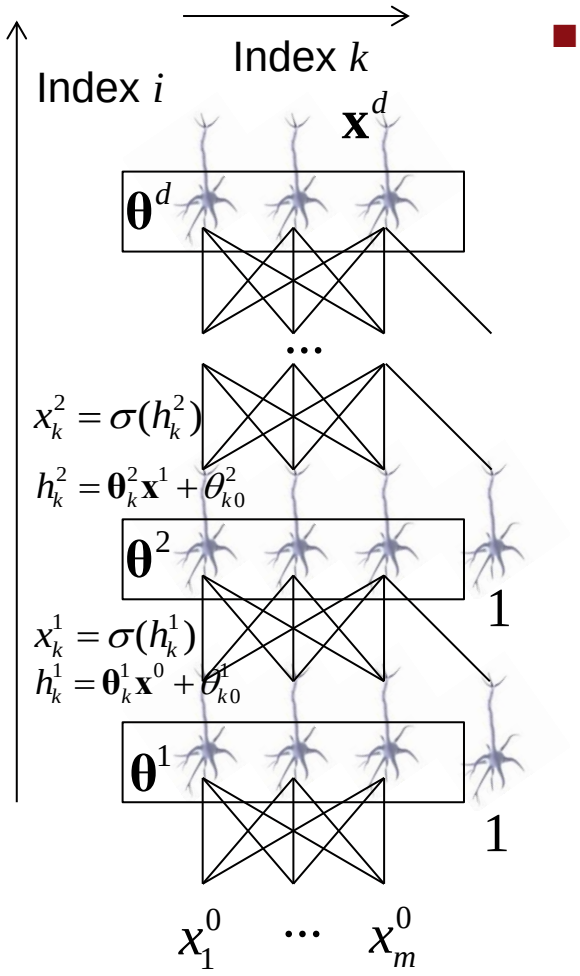
- Forward propagation:
  - Input vector $\mathbf{x}^0$
  - Linear model: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
  - Activation function and propagation: $\mathbf{x}^i = \sigma(\mathbf{h}^i)$
  - Output vector $\mathbf{x}^d$

Feed Forward Networks



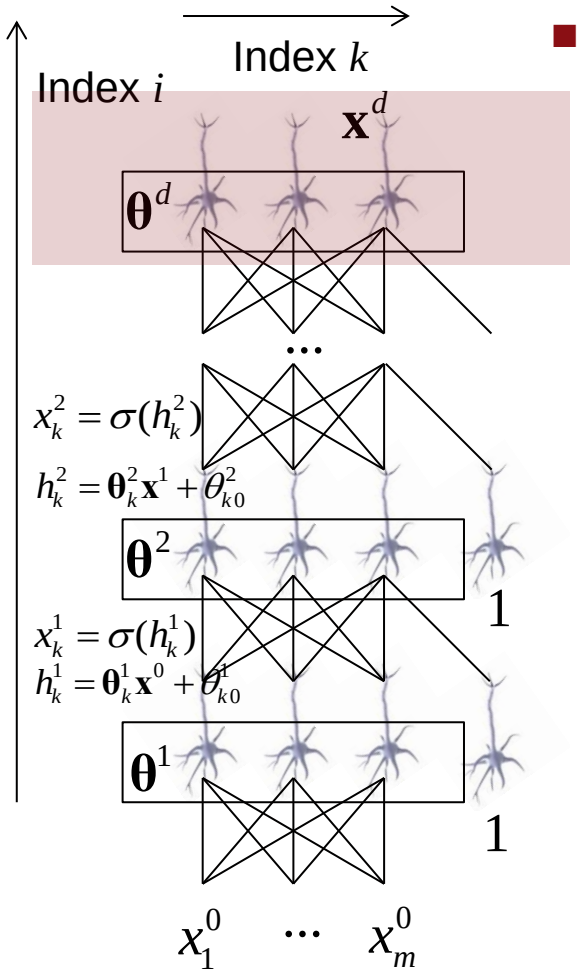
- Bias unit
  - Linear modell: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
  - Constant element θ_{k0}^i is replaced by additional unit with constant output 1: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}_{[1..n_k+1]}^{i-1}$

Feed Forward Networks



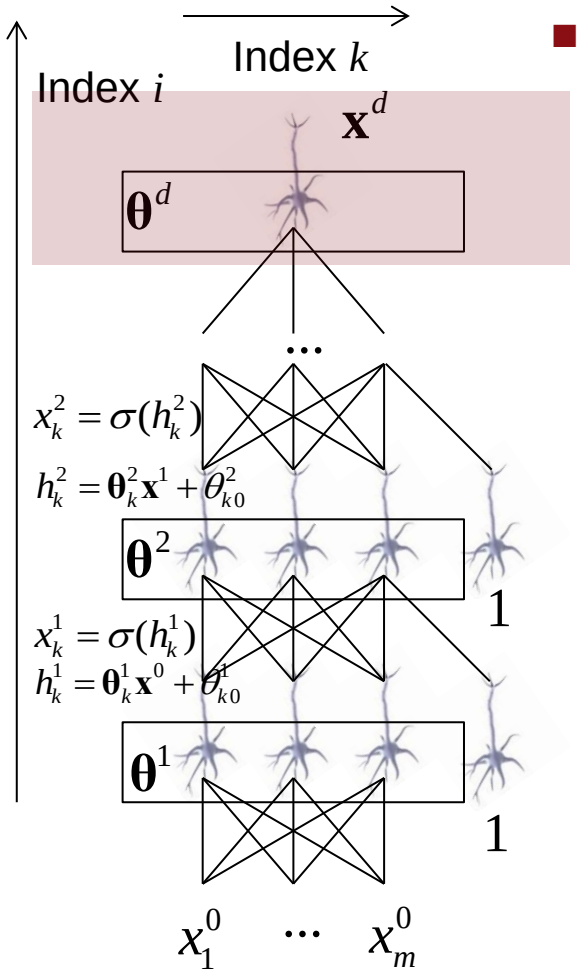
- Forward propagation per layer in matrix notation:
  - Linear model: $\mathbf{h}^i = \boldsymbol{\theta}^i \mathbf{x}^{i-1}$
  - Activation function: $\mathbf{x}^i = \sigma(\mathbf{h}^i)$

Classification: Softmax Layer

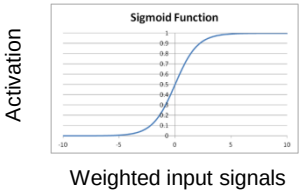


- One output unit per class:
  - $x_k^d = \sigma_{sm}(h_k^d) = \frac{e^{h_k^d}}{\sum_{k'} e^{h_{k'}^d}}$
  - x_k^d: predicted probability for class k.

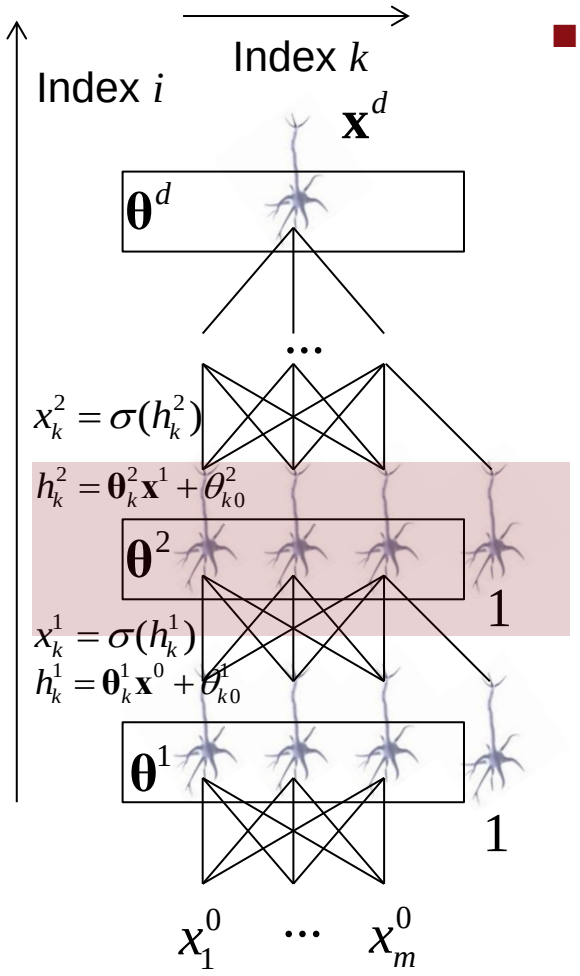
Regression: Sigmoidal Activation



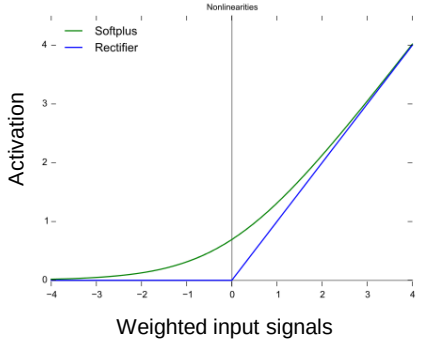
- For target variable k:
  - $x_k^d = \sigma_s(h_k^d) = \frac{1}{1+e^{-h_k^d}}$
  - Sigmoidal activation function.



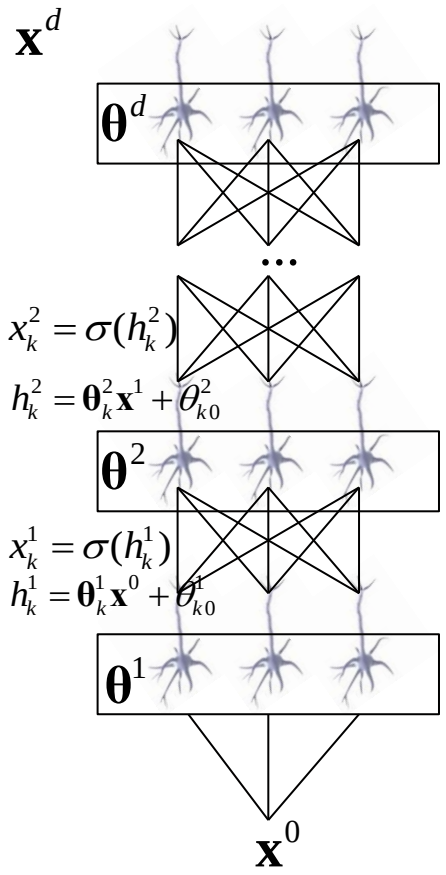
Internal Units: Rectified Linear Units



- 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.



Feed Forward Networks: Learning



- Stochastic gradient descent
- Loss function

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

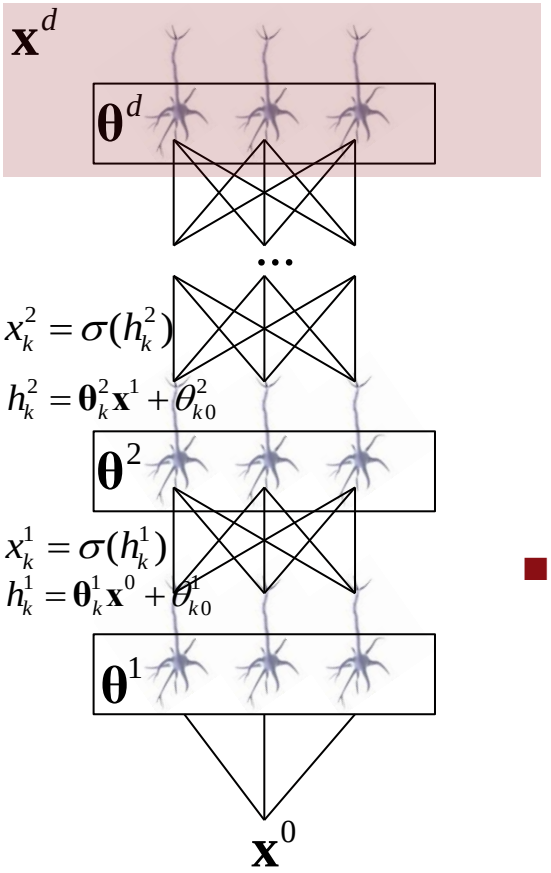
- Gradient descent:
  - $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla \hat{R}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \hat{R}(\boldsymbol{\theta})$

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

- Stochastic gradient for instance $\mathbf{x}_j$:
  - $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla_{\mathbf{x}_j} \hat{R}(\boldsymbol{\theta})$

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

Learning: Back Propagation



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

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

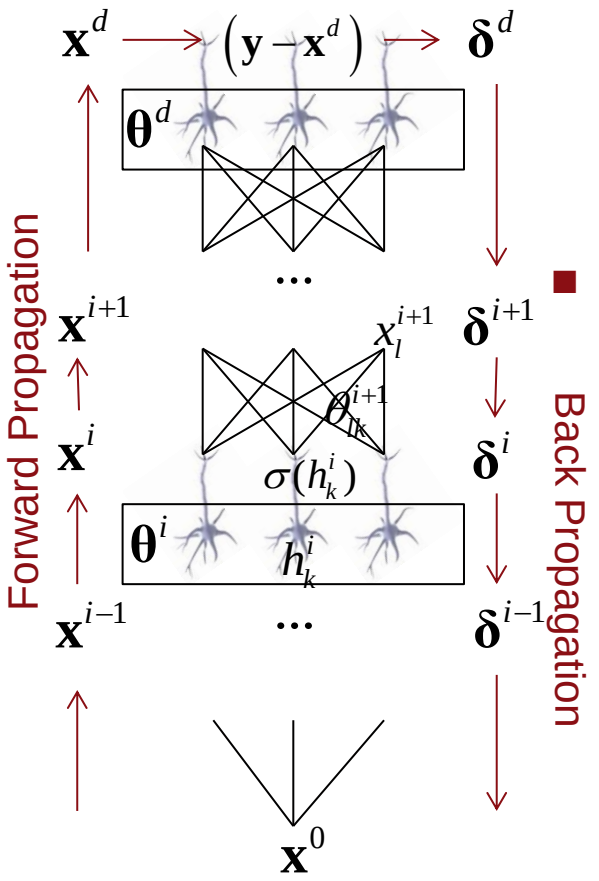
$$= \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial \mathbf{x}^d} \frac{\partial \sigma(h_k^d)}{\partial h_k^d} \mathbf{x}^{d-1}$$

$$= \delta_k^d \mathbf{x}^{d-1}$$

- 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 $\mathbf{x}_j$:

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

- With

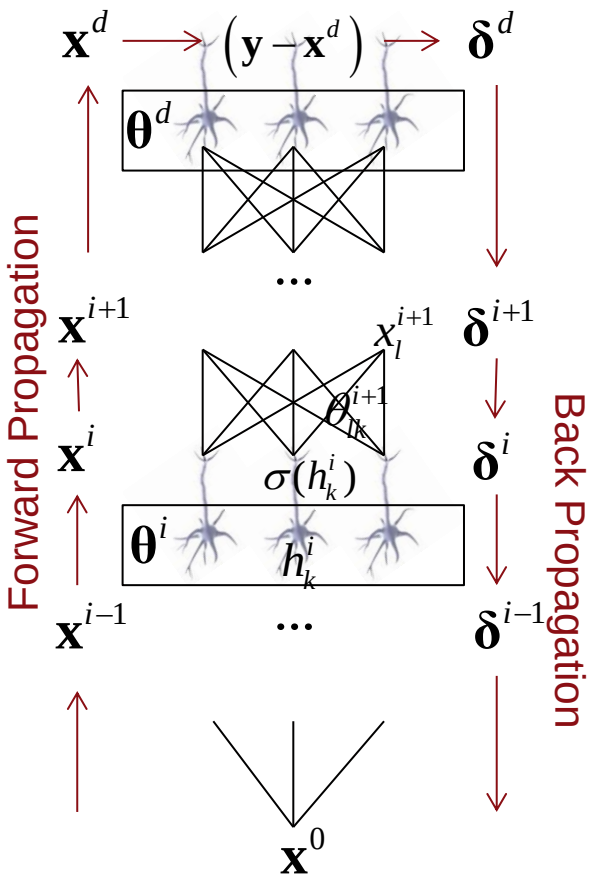
$$\delta_k^d = \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial h_k^i}$$

$$= \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial(\mathbf{x}_1^{i+1}, \ldots, \mathbf{x}_{n_{i+1}}^{i+1})} \frac{\partial(\mathbf{x}_1^{i+1}, \ldots, \mathbf{x}_{n_{i+1}}^{i+1})}{\partial h_k^i}$$

$$= \sum_{l=1}^{n_i+1} \frac{\partial \ell(\mathbf{y}_j, \mathbf{x}^d)}{\partial h_l^{i+1}} \frac{\partial h_l^{i+1}}{\partial x_k^i} \frac{\partial x_l^{i+1}}{\partial h_k^i}$$

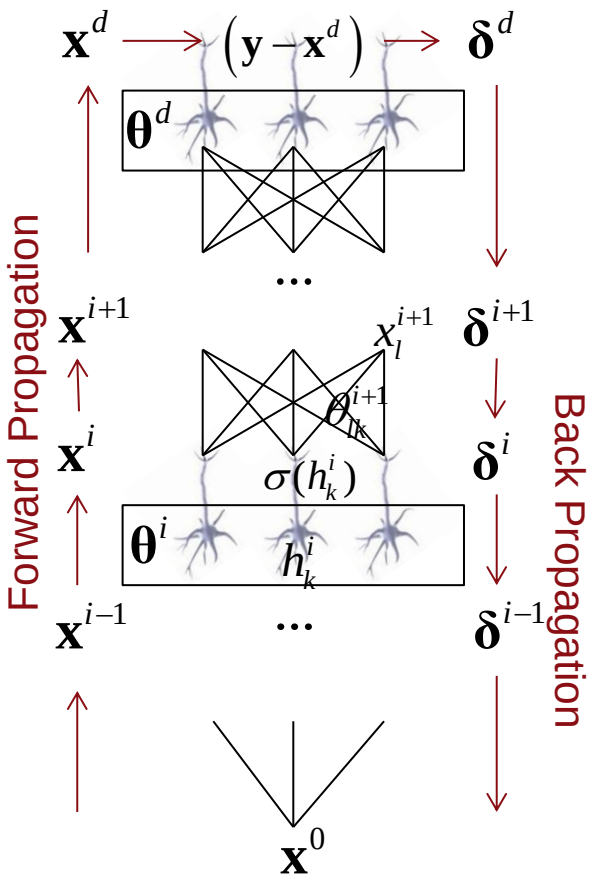
$$= \sum_{l=1}^{n_i+1} \delta_l^{i+1} \theta_{lk}^{i+1} \frac{\partial \sigma(h_k^i)}{\partial h_k^i}$$

Learning: Back Propagation



- Derivative of the loss function for classification.
- Softmax activation function:
  - $x_k^d = \sigma_{sm}(h_k^d) = \frac{e^{h_k^d}}{\sum_{k'} e^{h_{k'}^d}}$
  - $\frac{\partial \sigma_{sm}(h_k^d)}{\partial h_k^d} = \sigma_{sm}(h_k^d)(1 - \sigma_{sm}(h_k^d))$
- Cost function:
  - $\ell(\mathbf{y}, \mathbf{x}^d) = \sum_k y_k \log x_k^d$
  - $\frac{\partial \ell(\mathbf{y}, \mathbf{x}^d)}{\partial h_k^d} = x_k^d - y_k$

Learning: Back Propagation



- Derivative of the loss function for regression.
- Logistic activation function:
  - $x_k^d = \sigma_{sm}(h_k^d) = \frac{1}{e^{h_k^d}}$
  - $\frac{\partial \sigma_{sm}(h_k^d)}{\partial h_k^d} = \sigma_{sm}(h_k^d)(1 - \sigma_{sm}(h_k^d))$
- Cost function:
  - $\ell(\mathbf{y}, \mathbf{x}^d) = \frac{1}{2}\sum_k (x_k^d - y_k)^2$
  - $\frac{\partial \ell(\mathbf{y}, \mathbf{x}^d)}{\partial x_k^d} = x_k^d - y_k$

Back Propagation: Algorithm

- Iterate over training instances $(\mathbf{x}, \mathbf{y})$:
  - Forward propagation: for $i=0 \ldots d$:
    - For $k=1 \ldots n_i$: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
    - $\mathbf{x}^i = \sigma(\mathbf{h}^i)$
  - Back propagation:
    - For $k=1 \ldots n_i$: $\delta_k^d = \frac{\partial}{\partial h_k^d} \sigma(h_k^d) \frac{\partial}{\partial x_k^d} \ell(y_k, x_k^d)$

      $\boldsymbol{\theta}_k^{d\,\prime} = \boldsymbol{\theta}_k^d - \alpha \delta_k^d \mathbf{x}^{d-1}$
    - For $i=d\text{-}1 \ldots 1$:
      - For $k=1 \ldots n_i$: $\delta_k^i = \sigma'(h_k^i) \sum_l \delta_l^{i+1} \theta_{lk}^{i+1}$

        $\boldsymbol{\theta}_k^{i\,\prime} = \boldsymbol{\theta}_k^i - \alpha \delta_k^i \mathbf{x}^{i-1}$
- Until concergence

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.
- Hope:
  - Local minima can still be arbitrarily good.
  - Many local minima can be equally good.
- Reality:
  - Supervised learning often works with hundreds of layers and millions of training instances.

Regularization

- L2-regularized loss
  - $\hat{R}_2(\boldsymbol{\theta}) = \frac{1}{2m} \sum_j (\mathbf{y}_j - \mathbf{x}_j^d)^2 + \frac{\eta}{2} \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\theta}$
  - Corresponds to normal prior on parameters.
- Gradient: $\nabla \hat{R}_2(\boldsymbol{\theta}^i) = \frac{1}{m} \sum_j \boldsymbol{\delta}_j^i \mathbf{x}^i + \eta \boldsymbol{\theta}$
- Update: $\boldsymbol{\theta}' = \boldsymbol{\theta} - \boldsymbol{\delta}_j \mathbf{x} - \eta \boldsymbol{\theta}$
- 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.

Regularization: Stochastic Binary Units

- Deterministic units propagate $x_k^i = \sigma\left(h_k^i\right)$.
- Stochastic-binary units calculate activation $\sigma\left(h_k^i\right)$,
  - Then propagate $x_k^i = 1$ with probability $\sigma(h_k^i)$
  - $x_k^i = 0$ otherwise.
- Similar to dropout: with some probability, each unit does not produce output. x_k^i
- Biological neurons behave like this.

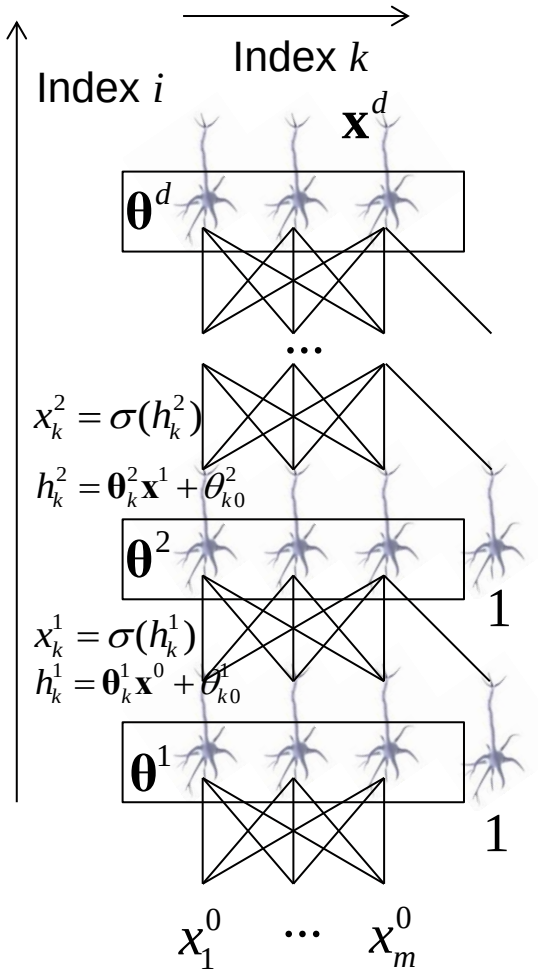
Back Propagation: Tricks

- Stochastic gradient on small batches.
- Permute training data at random.
- Decrease learning rate during optimization
- Initialize weights randomly (origin can be saddle point).
- Initialize weights via unsupervised pre-training.

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:

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

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

- Use vector coprocessor / GPU roy-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.
- Torch, Theano (+Lasagne), TensorFlow (+Keras). Deep learning libraries that allow development of new architectures and learning techniques.

Overview

- Neural information processing.
- Feed-forward networks.
- Training feed-forward networks, back propagation.
- Recurrent neural networks.
- LSTM networks.

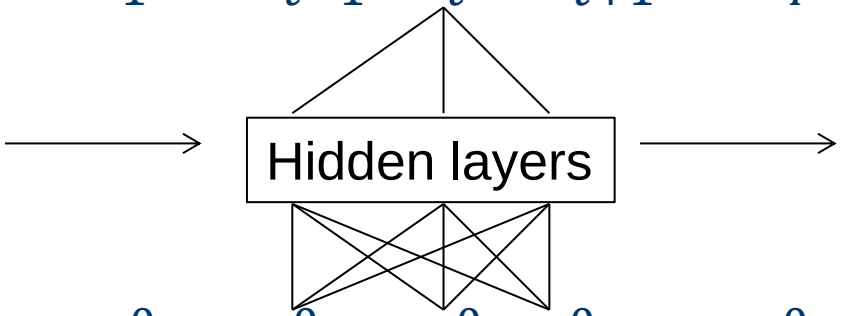
Recurrent Neural Networks

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

Convolutional Neural Network

- Network moves over sequence of input vectors.
- Window of k vectors serves as input
- Long-term dependencies not captured by model.

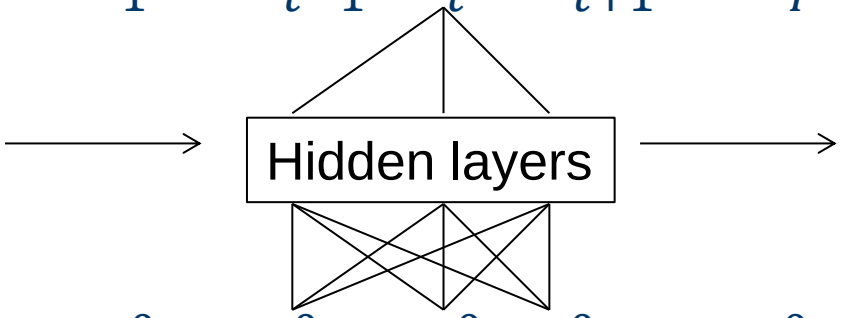
- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$



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

Convolutional Neural Network

- This architecture is used in neural language models.
- Here, each $\mathbf{x}_t$ is a one-hot coded word vector, $\mathbf{x}^d$ is the next word.
- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$

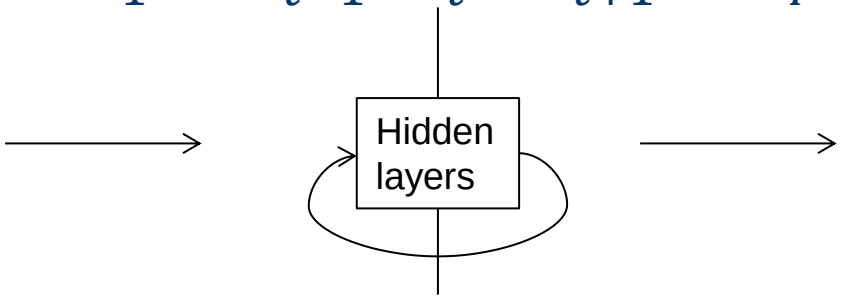


- Input $\mathbf{x}_1^0 \ \ldots \mathbf{x}_{t-1}^0 \ \mathbf{x}_t^0 \ \mathbf{x}_{t+1}^0 \ \ldots \ \mathbf{x}_T^0$.

Recurrent Neural Networks

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

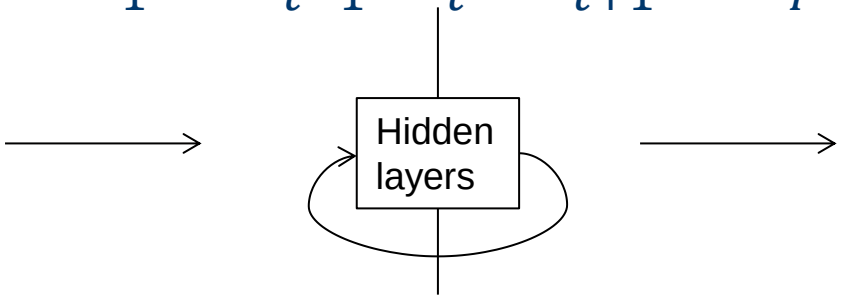
- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$



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

Recurrent Neural Networks

- Propagation:
  - $\mathbf{h}_t^1 = \boldsymbol{\theta}^1 \begin{pmatrix} \mathbf{x}_t^0 \\ \mathbf{x}_{t-1}^1 \end{pmatrix}; \mathbf{x}_t^1 = \sigma(\mathbf{h}_t^1)$
- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$

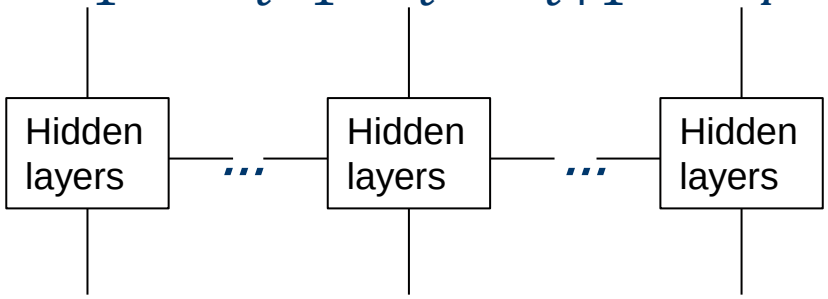


- Input $\mathbf{x}_1^0 \ \ldots \mathbf{x}_{t-1}^0 \ \mathbf{x}_t^0 \ \mathbf{x}_{t+1}^0 \ \ldots \ \mathbf{x}_T^0$.

Recurrent Neural Networks

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

- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$

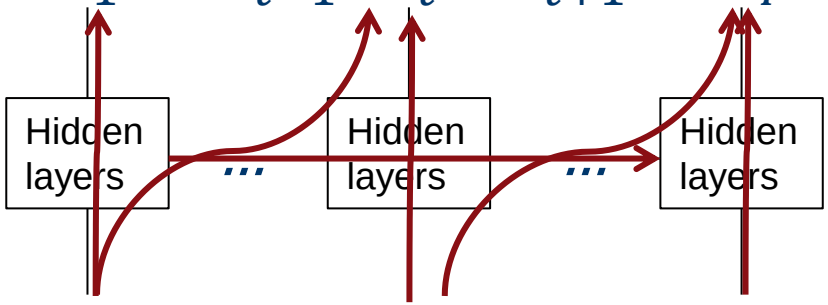


- Input $\mathbf{x}_1^0 \ \ldots \mathbf{x}_{t-1}^0 \ \mathbf{x}_t^0 \ \mathbf{x}_{t+1}^0 \ \ldots \ \mathbf{x}_T^0$.

Recurrent Neural Networks

- Forward propagation

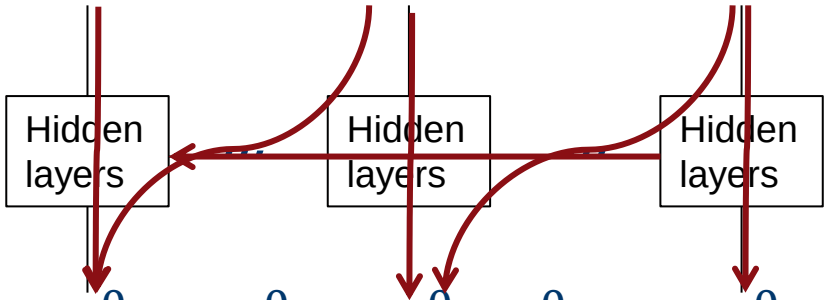
- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$



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

Recurrent Neural Networks

- Back propagation through time.
- A copy of each model parameter exists for each time step.
- Gradient is averaged over all time steps
- Output $\mathbf{x}_1^d \ \ldots \mathbf{x}_{t-1}^d \ \mathbf{x}_t^d \ \mathbf{x}_{t+1}^d \ \ldots \ \mathbf{x}_T^d$



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

Memory in Recurrent Neural Networks

- Memory state at time t:
  - Activation function of (weights x memory state at time $t-1$, weights x input at time t).
- Memory state is almost always changed.
- Memory state has the same impact on the outcome for almost al inputs.
- Idea: Learn when to access, modify, reset memory state → LSTM units.

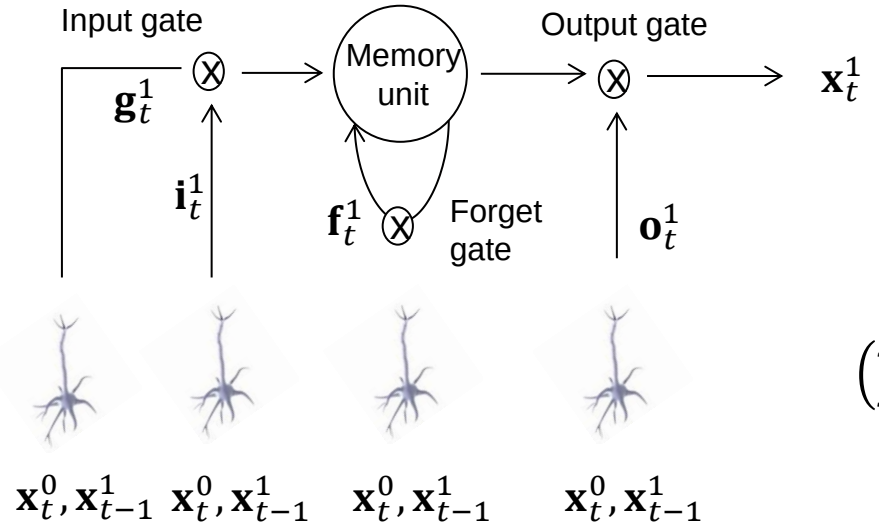
Overview

- Neural information processing.
- Feed-forward networks.
- Training feed-forward networks, back propagation.
- Recurrent neural networks.
- LSTM networks.

LSTM Units

- Input gate scales input to memory.
- Forget gate scales old memory value.
- Output gate scales output.

$$\mathbf{c}_t^1 = \mathbf{f}_t^1 \odot \mathbf{c}_{t-1}^1 + \mathbf{i}_t^1 \odot \mathbf{g}_t^i$$

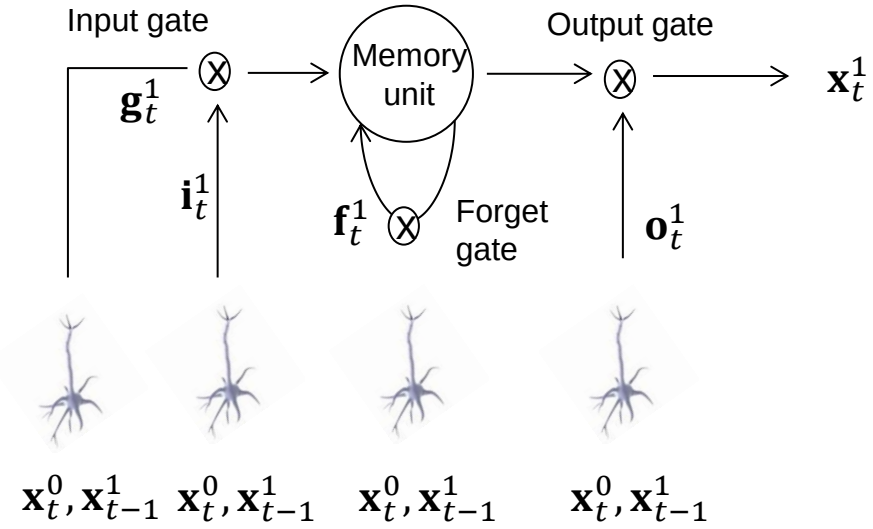


$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \odot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 y_1 \\ x_2 y_2 \end{pmatrix}$$

LSTM Units

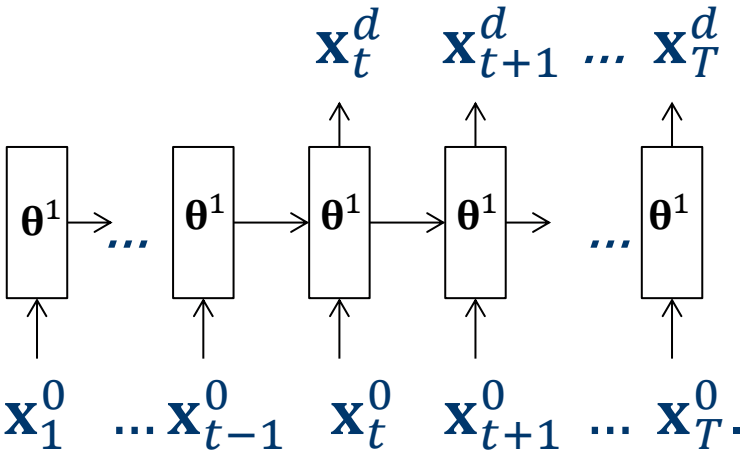
- Gradient propagated through gates:
  - $\mathbf{z} = \mathbf{x}_1 \odot \mathbf{x}_2$
  - $\frac{\partial E}{\partial \mathbf{x}_1} = \frac{\partial E}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial E}{\partial \mathbf{z}} \odot \mathbf{x}_2$

$$\mathbf{c}_t^1 = \mathbf{f}_t^1 \odot \mathbf{c}_{t-1}^1 + \mathbf{i}_t^1 \odot \mathbf{g}_t^i$$



Sequence Prediction with LSTMs

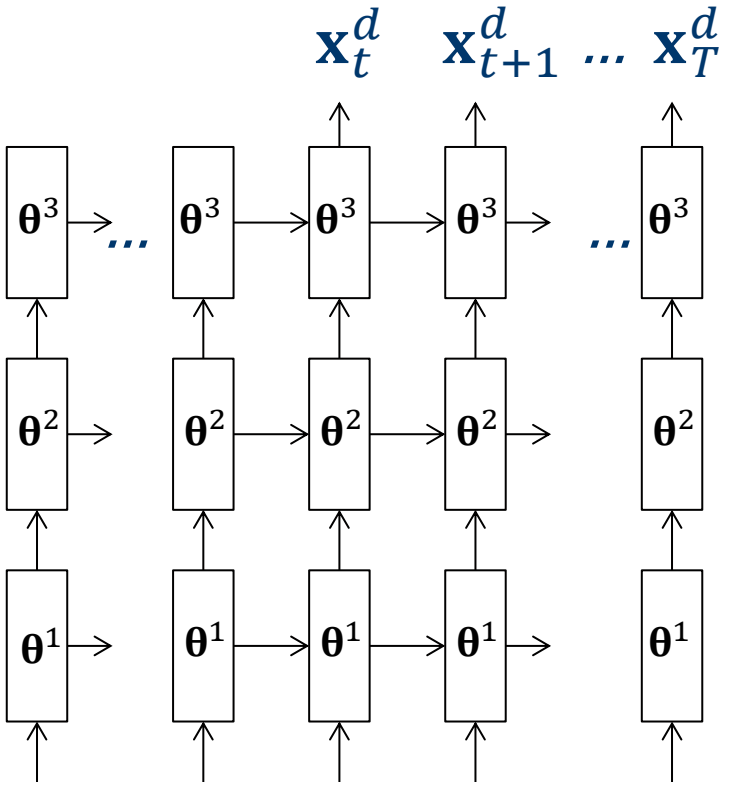
- Output
- Input



- Input sequence mapped to output sequence; sequences may differ in length:
- For instance, sentence in source language → sentence in target language.

Deep LSTM Networks

- Output



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

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

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.
- Recurrent neural networks: state information passed on to next time step.
- LSTM networks: gating units decide whether to update, delete memory, produce output.