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.
- Unsupervised learning:
  - Auto encoders.
  - Training auto encoders via back propagation.
  - Restricted Boltzmann machines.
- Convolutional networks.

Learning Problems can be Impossible without the Right Features

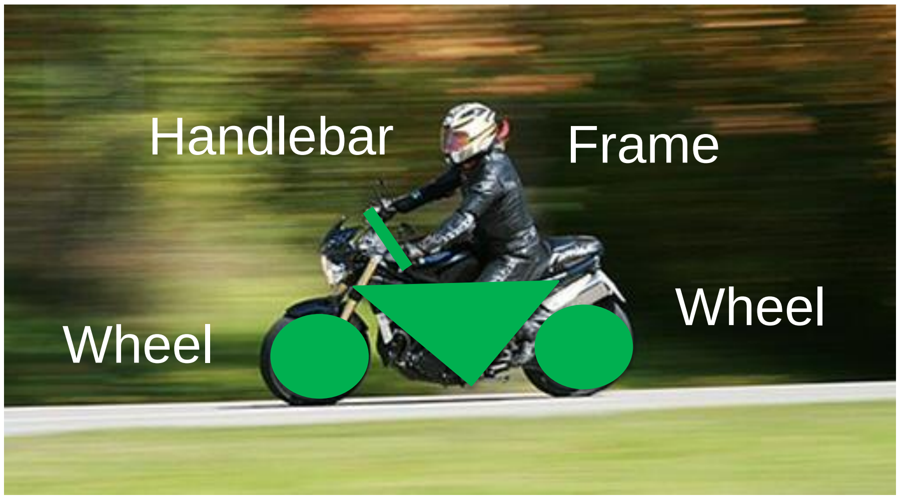
→ motorcycle

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

?

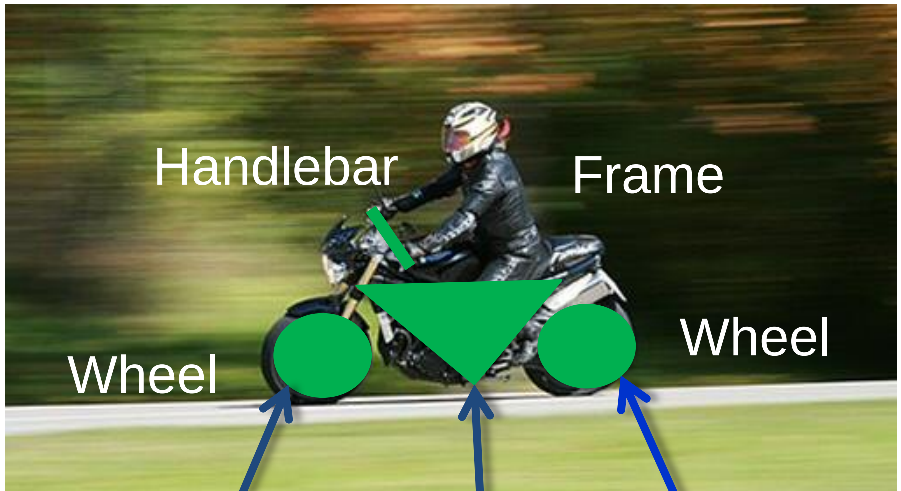
→ motorcycle

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

Neuronal 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)

Neuronal Networks

- Deep learning, unsupervised feature learning
  - Unsupervised discovery of features which can then be used for supervised learning
  - Implementation on GPU
  - Able to process vast amounts of data.
  - Seen as step towards AI

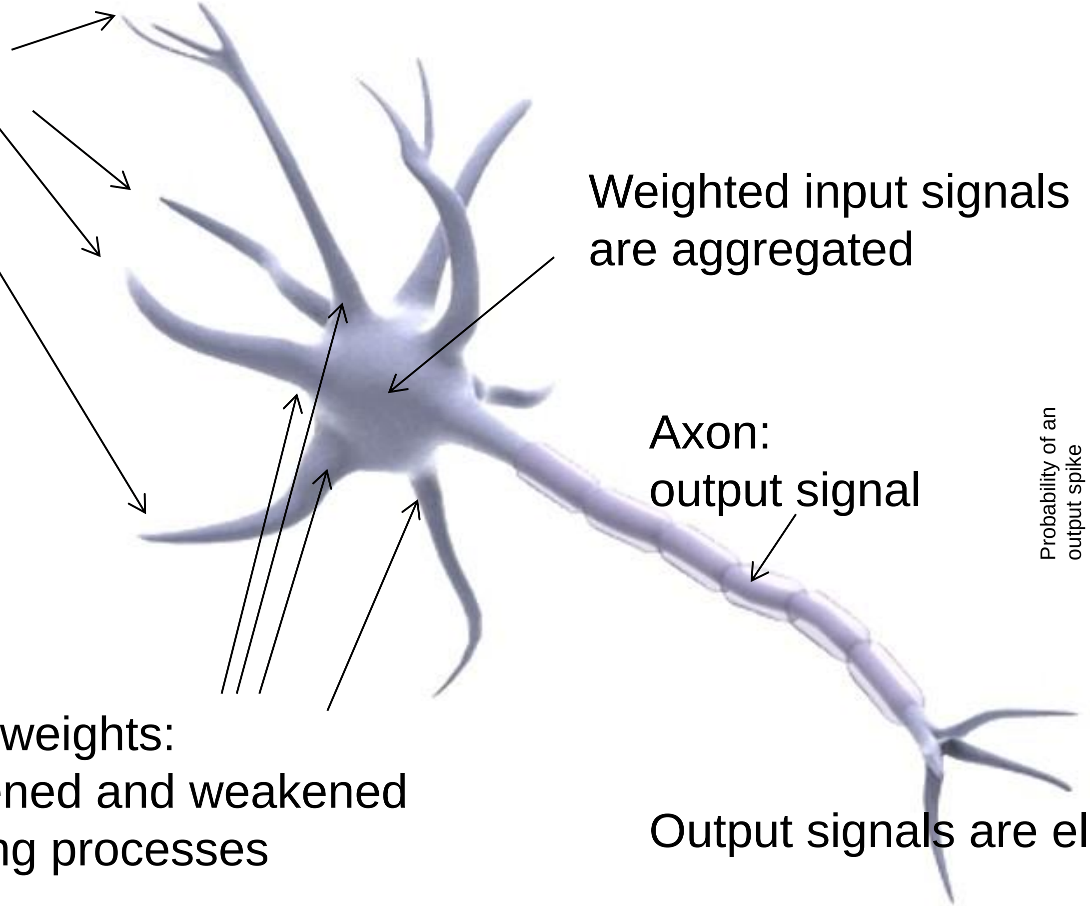
Deep Learning Records

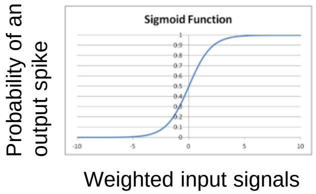
- Neural networks best-performing algorithms for
  - Object classification (CIFAR/NORB/PASCAL VOC-Benchmarks)
  - Video classification (various benchmarks)
  - Sentiment analysis (MR Benchmark)
  - Pedestrian detection
  - Speech recognition
  - Phychedelic art (Deep Dream)

Supervised and Unsupervised Learning

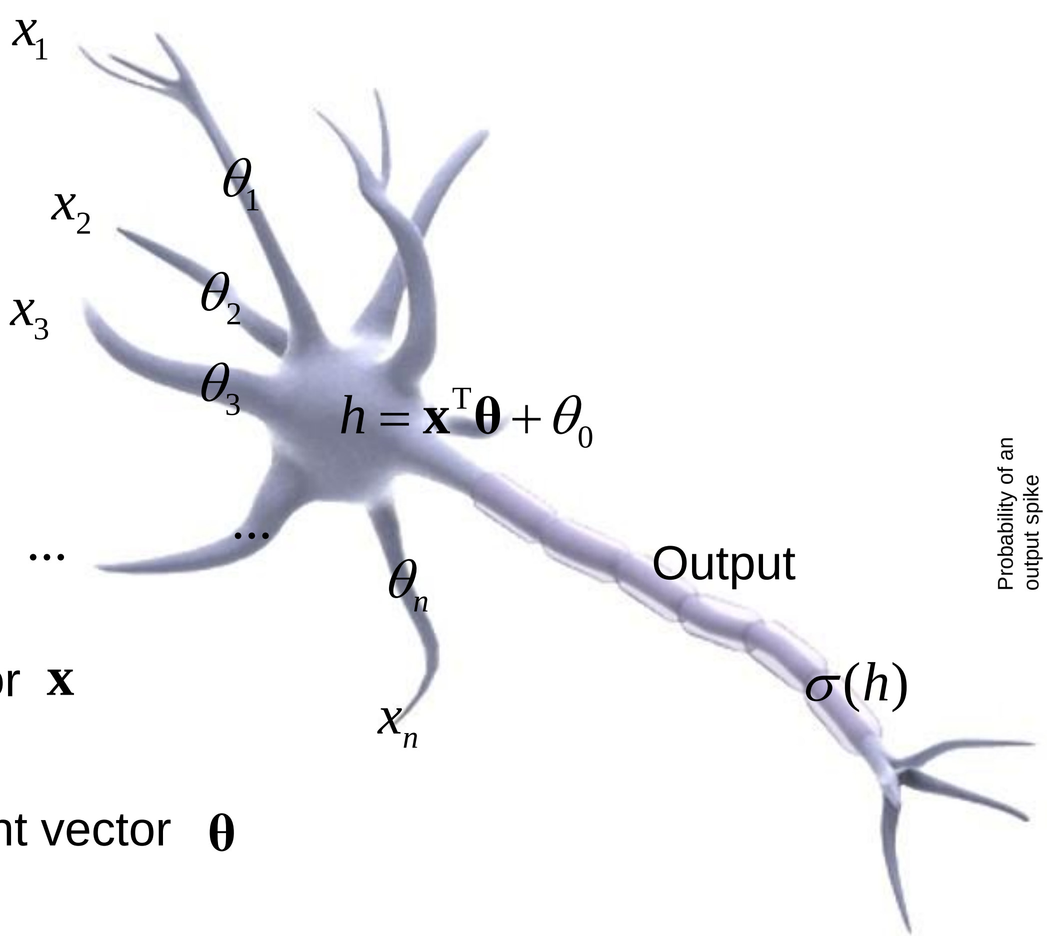
- Supervised learning
  - Entire network trained on labeled data.
- Unsupervised learning
  - Entire network trained on unlabeled data.
- Unsupervised pre-training + supervised learning
  - Network (except top-most layer) trained layer-wise on unsupervised data.
  - Then, entire network is trained on labeled data.
  - Good for many unlabeled + few labeled data.

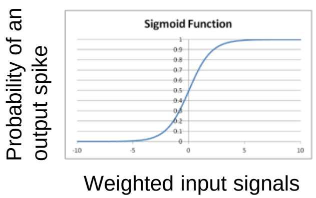
Neural Information Processing





Neural Information Processing: Model

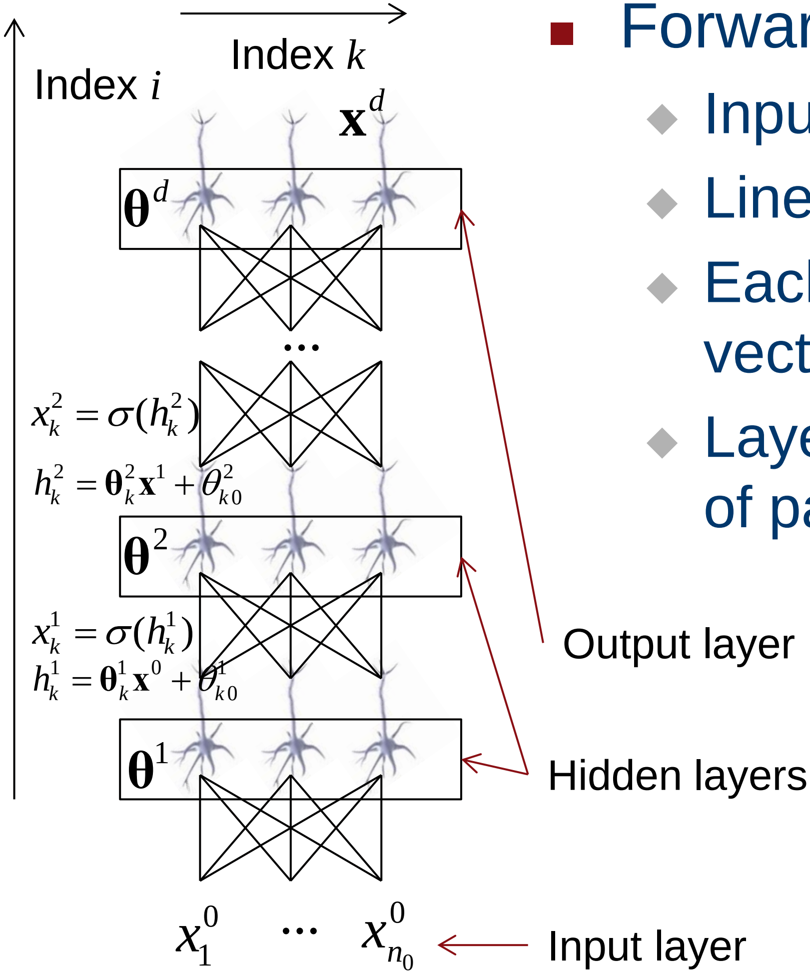




Input vector $\mathbf{x}$

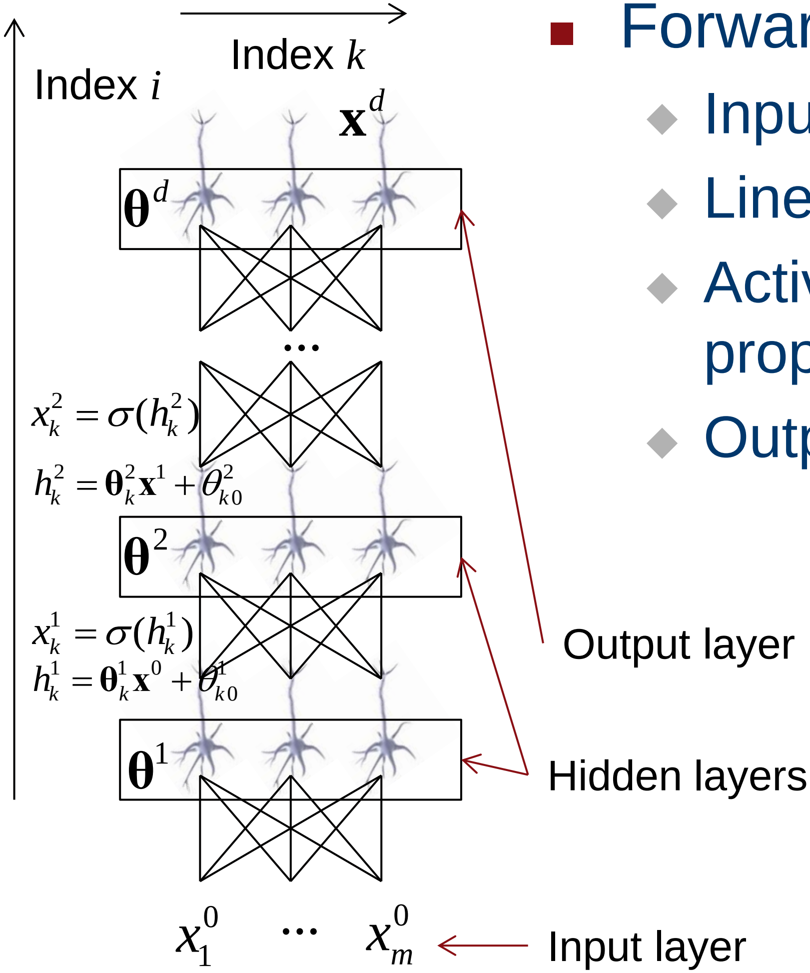
Weight vector $\boldsymbol{\theta}$

Feed Forward Networks



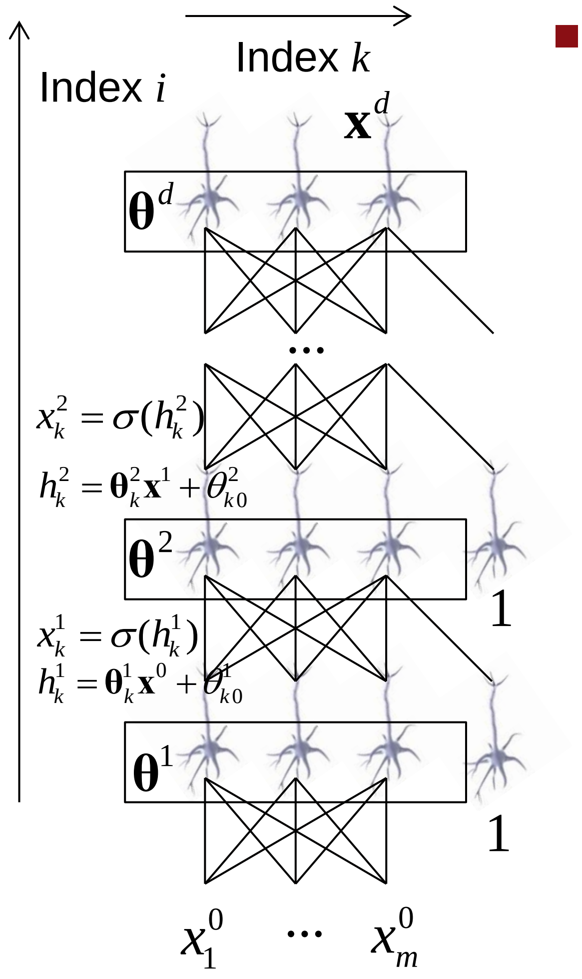
- Forward propagation:
  - Input vector $\mathbf{x}^0$
  - Linear model: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
  - Each unit has parameter vector $\boldsymbol{\theta}_k^i = \begin{pmatrix} \theta_1^i & \dots & \theta_{n_i}^i \end{pmatrix}$
  - Layer i has matrix of parameters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

Feed Forward Networks



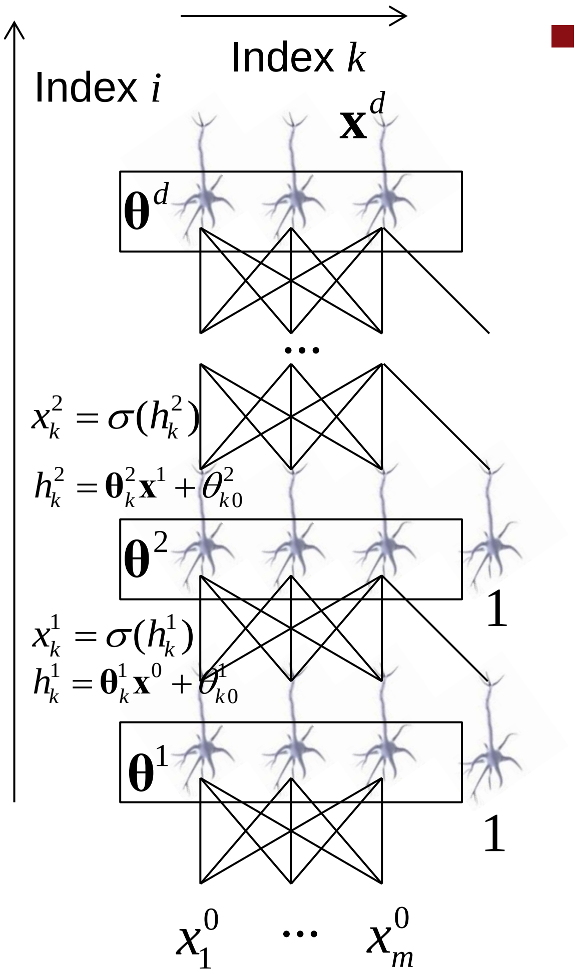
- Forward propagation:
  - Input vector $\mathbf{x}^0$
  - Linear model: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
  - Activation function and propagation: $\mathbf{x}^i = \sigma(\mathbf{h}^i)$
  - Output vector $\mathbf{x}^d$

Feed Forward Networks



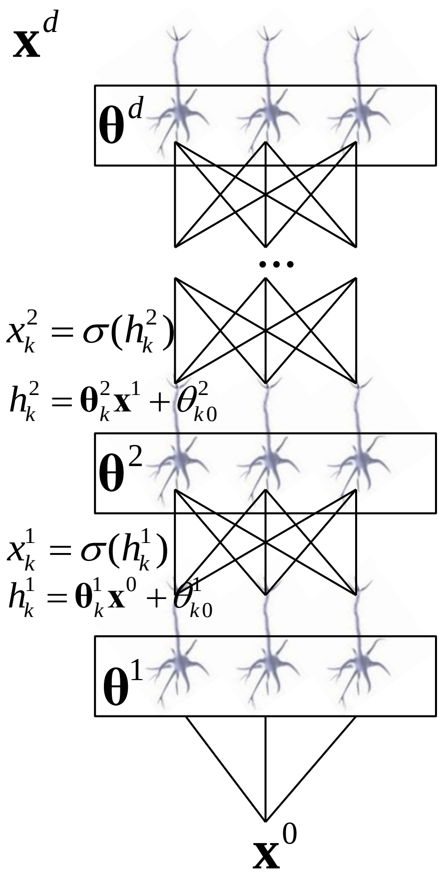
- Bias unit
  - Linear modell: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}^{i-1} + \theta_{k0}^i$
  - Constant element θ_{k0}^i is replaced by additional unit with constant output 1: $h_k^i = \boldsymbol{\theta}_k^i \mathbf{x}_{[1..n_k+1]}^{i-1}$

Feed Forward Networks



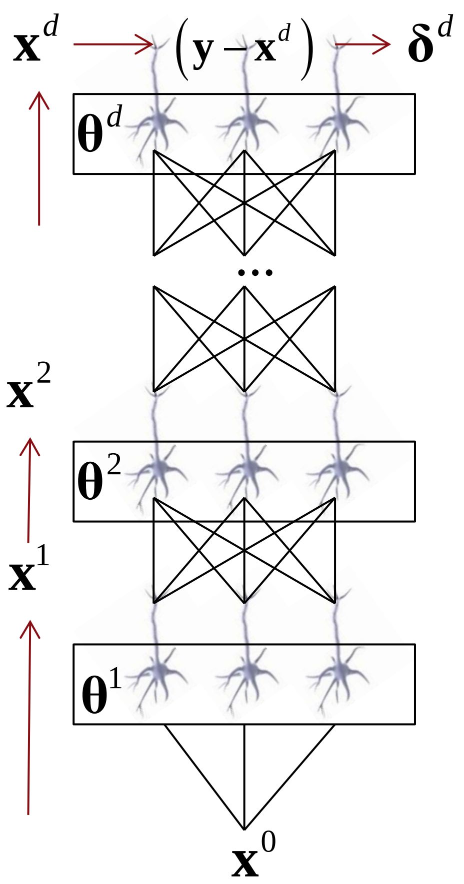
- Forward propagation per layer in vector notation:
  - $\mathbf{h}^i = \boldsymbol{\theta}^i \mathbf{x}^{i-1}$

Feed Forward Networks



- Stochastic gradient descent
- Squared loss:
  $$\hat{R}(\theta) = \frac{1}{2m} \sum_{j=1}^{m} \left(y_j - x_j^d\right)^2$$
- Gradient:
  - $$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla \hat{R}(\boldsymbol{\theta}) = \boldsymbol{\theta}' - \alpha \frac{\partial \hat{R}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$
    $$= \boldsymbol{\theta} - \alpha \frac{\partial \frac{1}{2m} \sum_j (\mathbf{y}_j - \mathbf{x}_j^d)^2}{\partial \boldsymbol{\theta}}$$
- Stochastic gradient for instance $\mathbf{x}$
  - $$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \frac{\partial \frac{1}{2} (\mathbf{y} \;\; - \mathbf{x}^d)^2}{\partial \boldsymbol{\theta}}$$

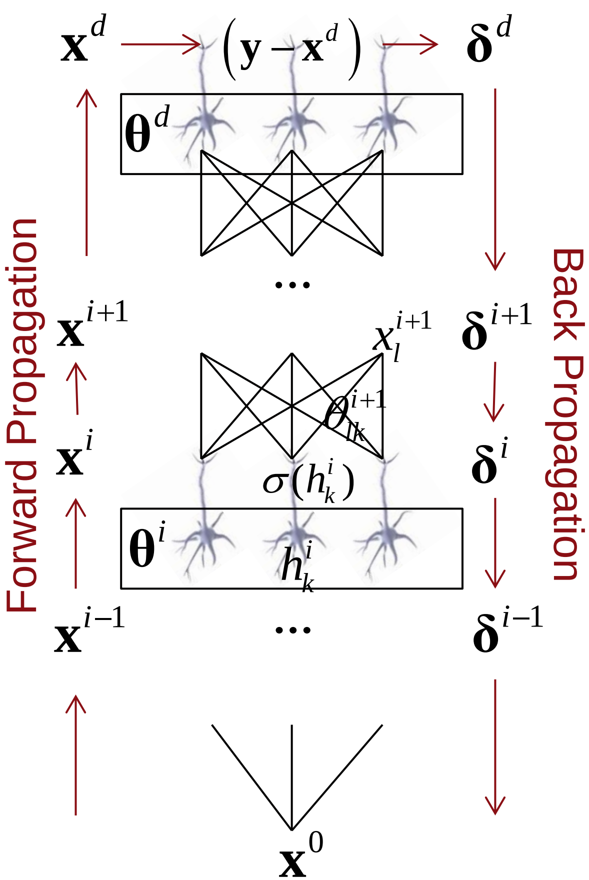
Feed Forward Nets: Back Propagation



- Stochastic gradient for instance $\mathbf{x}$
  - $$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \frac{\partial \frac{1}{2}(\mathbf{y}-\mathbf{x}^d)^2}{\partial \boldsymbol{\theta}}$$
- For top-level weights:
  - $$\begin{aligned}\frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial \boldsymbol{\theta}_k^d} &= \frac{\partial \frac{1}{2}(y_k - \sigma(\boldsymbol{\theta}_k^d \mathbf{x}^{d-1}))^2}{\partial \boldsymbol{\theta}_k^d} \\ &= \frac{\partial \frac{1}{2}(y_k - \sigma(\boldsymbol{\theta}_k^d \mathbf{x}^{d-1}))^2}{\partial \sigma(\boldsymbol{\theta}_k^d \mathbf{x}^{d-1})} \frac{\partial \sigma(\boldsymbol{\theta}_k^d \mathbf{x}^{d-1})}{\partial \boldsymbol{\theta}_k^d \mathbf{x}^{d-1}} \frac{\partial \boldsymbol{\theta}_k^d \mathbf{x}^{d-1}}{\partial \boldsymbol{\theta}_k^d} \\ &= (y_k - \sigma(\boldsymbol{\theta}_k^d \mathbf{x}^{d-1}))\sigma'(\boldsymbol{\theta}_k^d \mathbf{x}^{d-1})\mathbf{x}^{d-1} \\ &= (y_k - x_k^d)\sigma'(h_k^d)\mathbf{x}^{d-1} \\ &= \delta_k^d \mathbf{x}^{d-1}\end{aligned}$$
- with: $\delta_k^d = \frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial h_k^d}$

$$= \sigma'(h_k^d)(y_k - x_k^d)$$

Feed Forward Nets: Back Propagation



- For weights at layer i:
  - $$\frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial \boldsymbol{\theta}_k^i} = \frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial h_k^i} \frac{\partial h_k^i}{\partial \boldsymbol{\theta}_k^i}$$
    $$= \delta_k^i \mathbf{x}^{i-1}$$
- with
  - $$\delta_k^i = \frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial h_k^i}$$
    $$\frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial (x_1^{i+1}, \ldots, x_{n_{i+1}}^{i+1})} \frac{\partial (x_1^{i+1}, \ldots, x_{n_{i+1}}^{i+1})}{\partial h_k^i}$$
    $$= \sum_l \frac{\partial \frac{1}{2}(y_k - x_k^d)^2}{\partial h_l^{i+1}} \frac{\partial h_l^{i+1}}{\partial x_k^i} \frac{\partial x_k^i}{\partial h_k^i}$$
    $$= \sum_l \delta_l^{i+1} \theta_{lk}^{i+1} \sigma'(h_k^i)$$
    $$= \sigma'(h_k^i) \sum_l \delta_l^{i+1} \theta_{lk}^{i+1}$$

Activation Function

- Any differentiable sigmoidal function is suitable
- Examples:
  - $\sigma(h) = \frac{1}{1+e^{-h}}$
  - $\sigma'(h) = \sigma(h)(1-\sigma(h))$

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 $k=1\ldots n_i$: $\delta_k^d = \sigma'(h_k^d)(y_k - x_k^d)$

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

        $\boldsymbol{\theta}_k^i{}' = \boldsymbol{\theta}_k^i - \alpha \delta_k^i \mathbf{x}^{i-1}$
- 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:
  - Back propagation works well for few (1 or 2) hidden layers.

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: Stop training before convergence.
  - Delete units with small weights.
  - Dropout: During training, set some units‘ output to zero at random.
  - Normalize length of propagated vectors.

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 $x_k^i = \sigma(h_k^i)$.
- Stochastic-binary units calculate activation $\sigma(h_k^i)$,
  - 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.
- Biological nneurons behave like this.

Back Propagation: Tricks

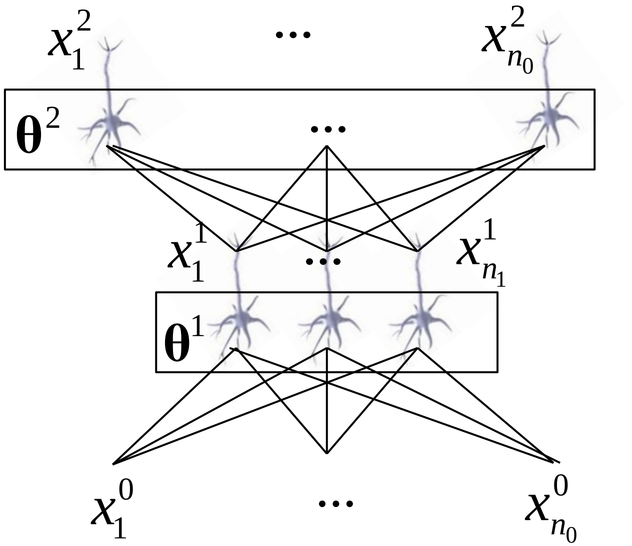
- Use cross-entropy as loss for classification
- Stochastic gradient on small batches.
- Permute training data at random.
- Decrease learning rate during optimization
- Initialize weights randomly (origin can be saddle point).
- Initialize weights via unsupervised pre-training.

Overview

- Neural information processing.
- Feed-forward networks.
- Training feed-forward networks, back propagation.
- Unsupervised learning:
  - Auto encoders.
  - Training auto encoders via back propagation.
  - Restricted Boltzmann machines.
- Convolutional networks.

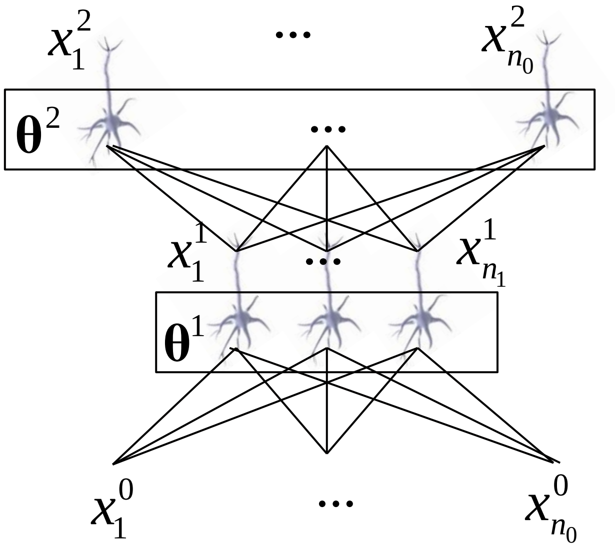
Auto Encoders

- Auto encoders learn the identity function.
- n_0 input units to n_1 hidden units to n_0 output units, with $n_0 > n_1$.
- On the hidden layer, the input has to be compressed.
- Learning algorithm derives a representation that preserves the information from the input.



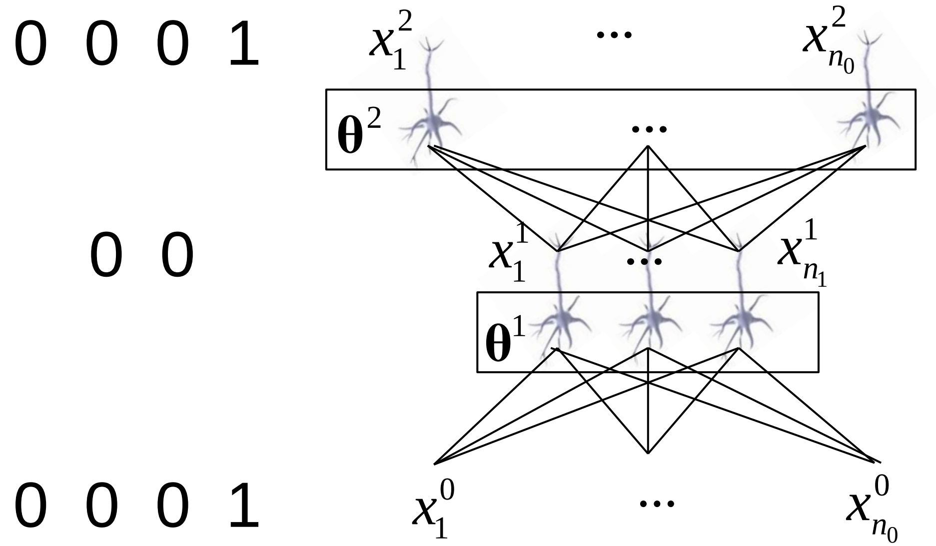
Auto Encoders: Example

- Input: binary vectors with a single 1.
- 4 input units, 2 hidden units, 4 output units
- Inputs:
  - 0,0,0,1
  - 0,0,1,0
  - 0,1,0,0
  - 1,0,0,0



Auto Encoders: Example

- Possible activation of the hidden units after training



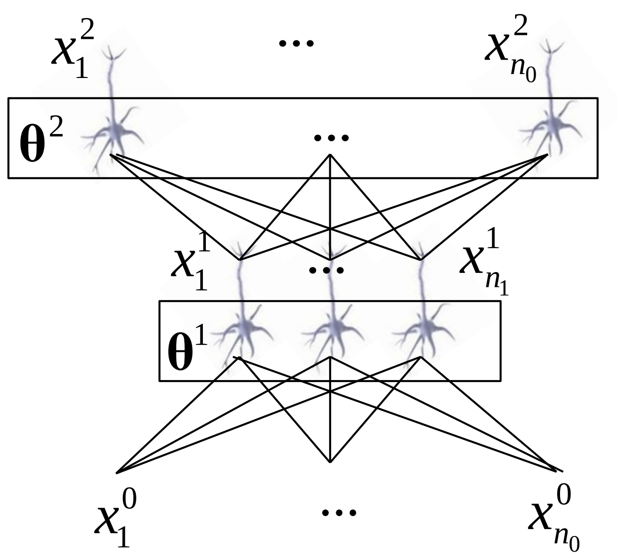
Auto Encoders: Example

- Possible activation of the hidden units after training

0 0 1 0

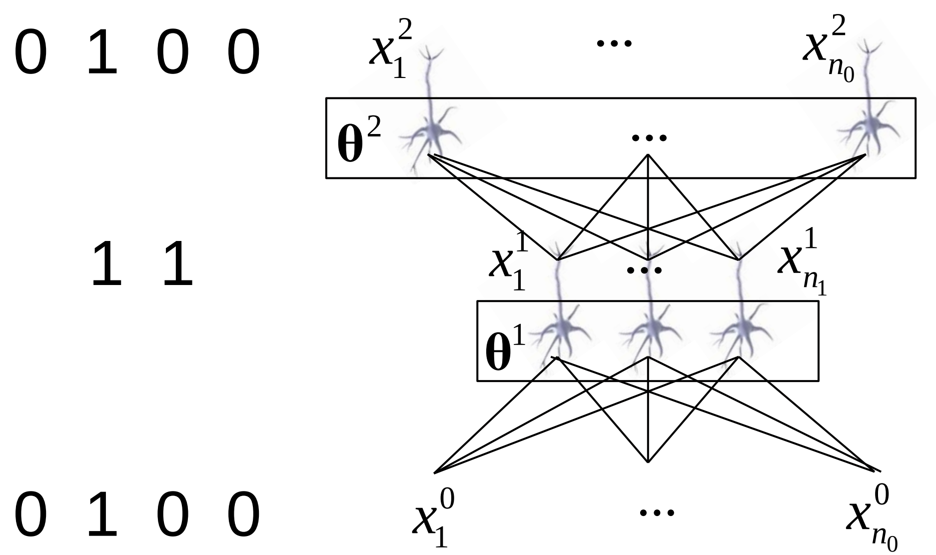
1 0

0 0 1 0



Auto Encoders: Example

- Possible activation of the hidden units after training



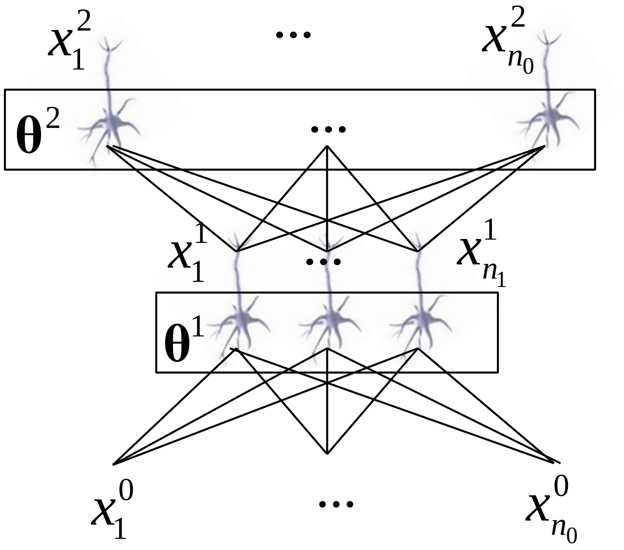
Auto Encoders: Example

- Possible activation of the hidden units after training

1 0 0 0

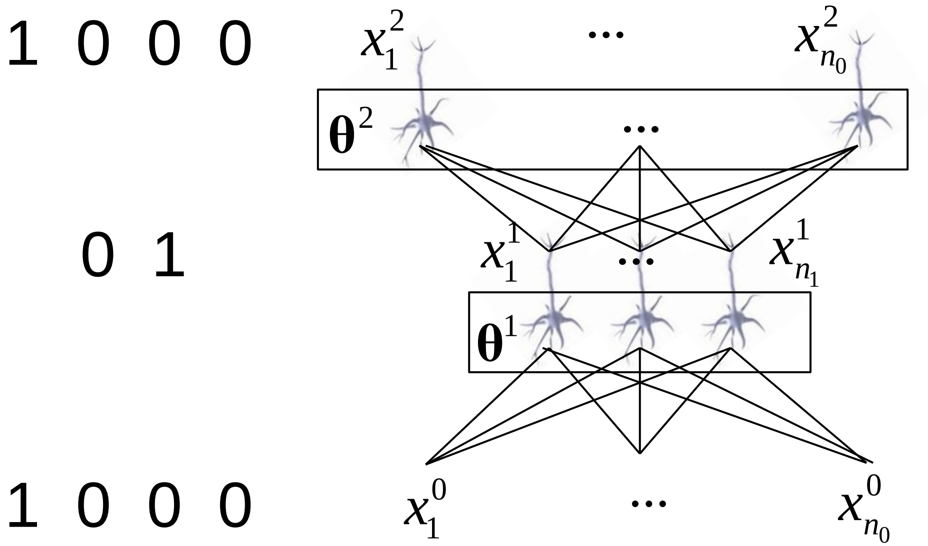
0 1

1 0 0 0



Auto Encoders: Example

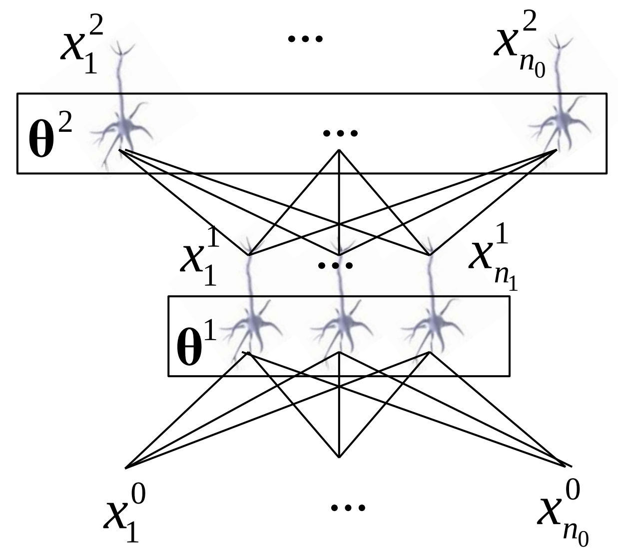
- There are several local minima of the loss functions (how many?)



Auto Encoders: Example

- Input: 256×256 units
  - Each unit represents the grey value of a pixel.
- Hidden layer: k units
- Output: 256×256 units

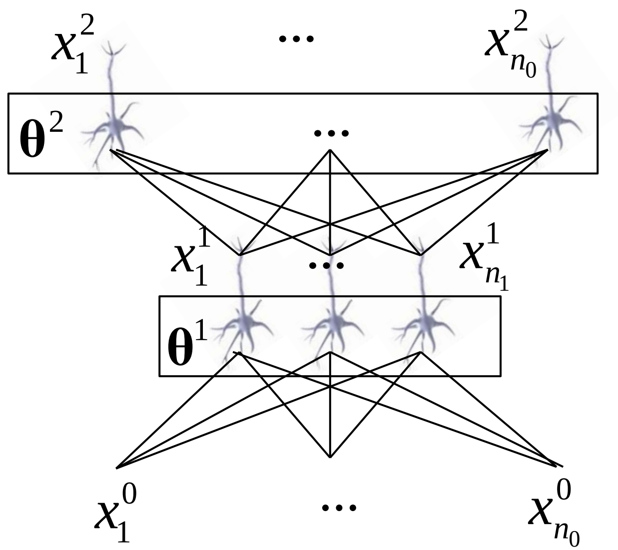
0.23 0.18 0.87 0.43



Auto Encoders: Example

- Each of the hidden units is a detector for a “base face”
- The weights from one hidden unit to the output units encode the image of the base face.
- Input faces are represented as a combination of these base faces.

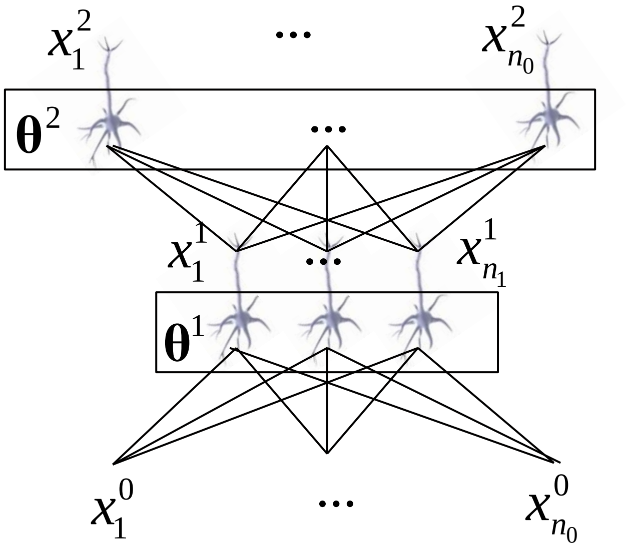
0.23 0.18 0.87 0.43



Auto Encoders: Example

- The weights from one hidden unit to the output units encode the image of the base face.

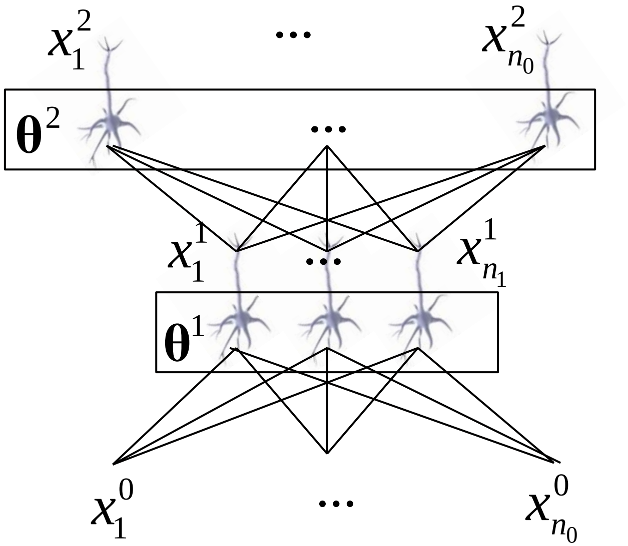
1 0 0 0



Auto Encoders: Example

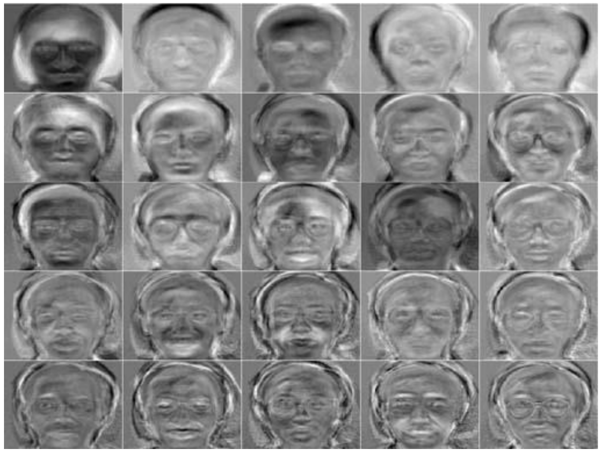
- Feeding an input of 1 into one of the hidden units produces the base face that the hidden unit represents.

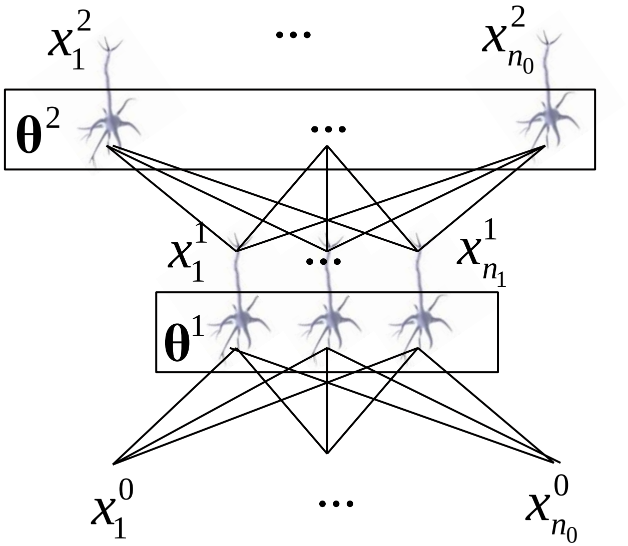
0 1 0 0



Auto Encoders: Example

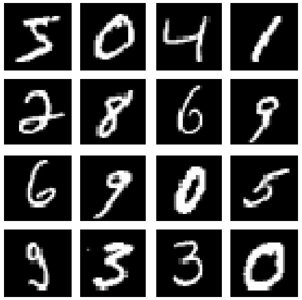
- The weights from one hidden unit to the output units encode the image of the base face.
- Weights from all hidden units to output units after training with a set of aligned faces:

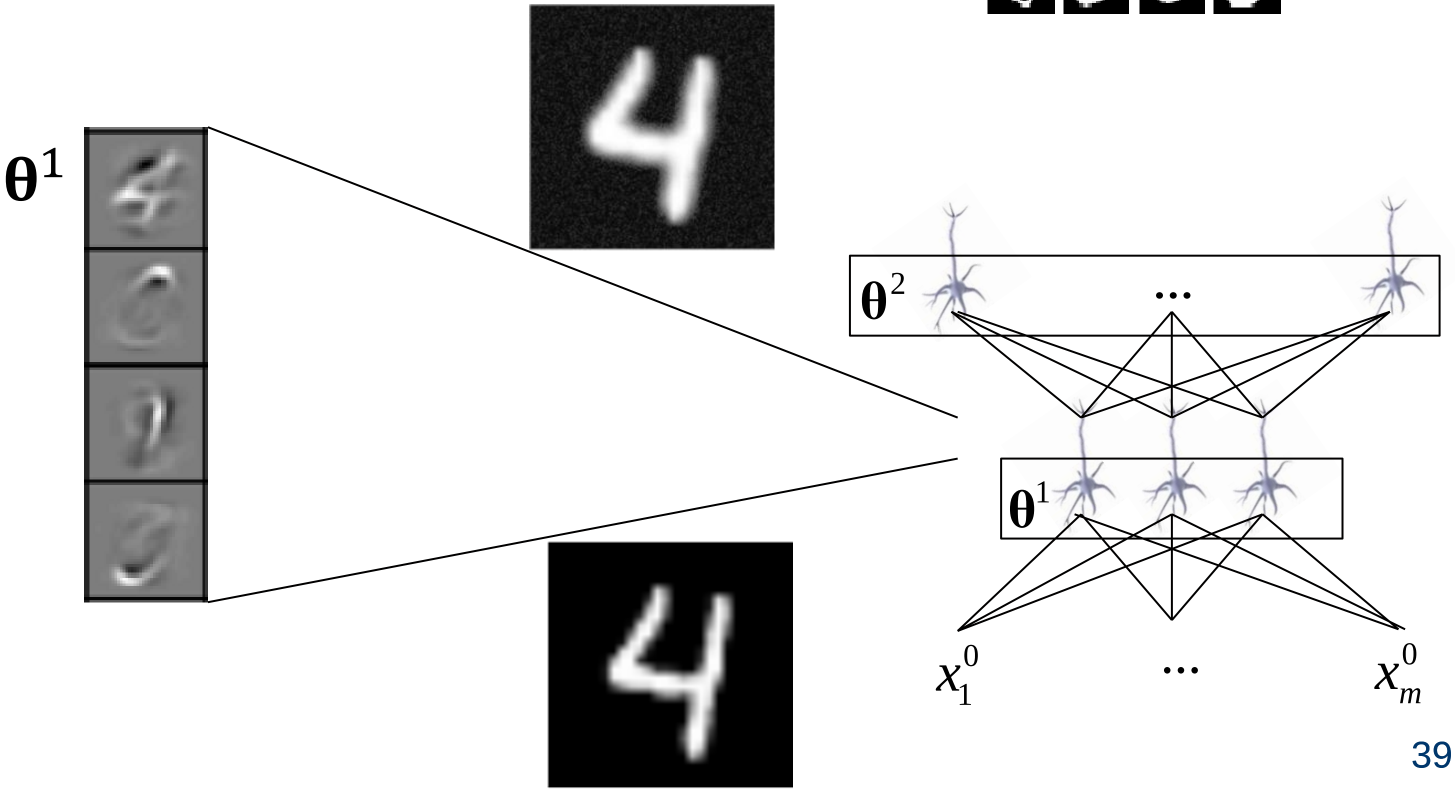




Auto Encoders: Example

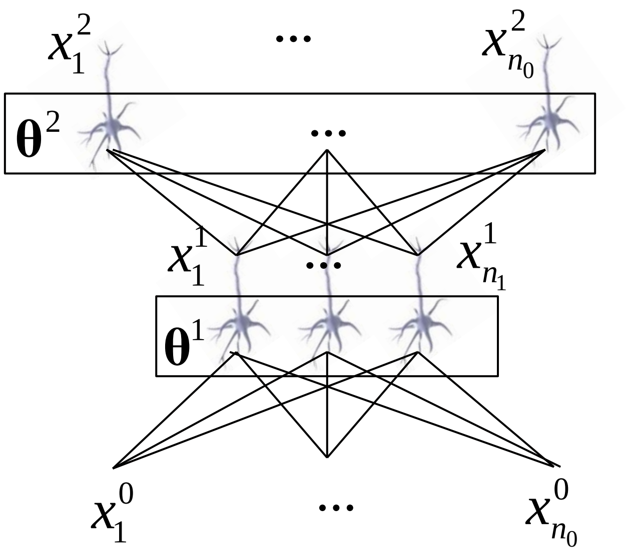
- After training on hand-written digits





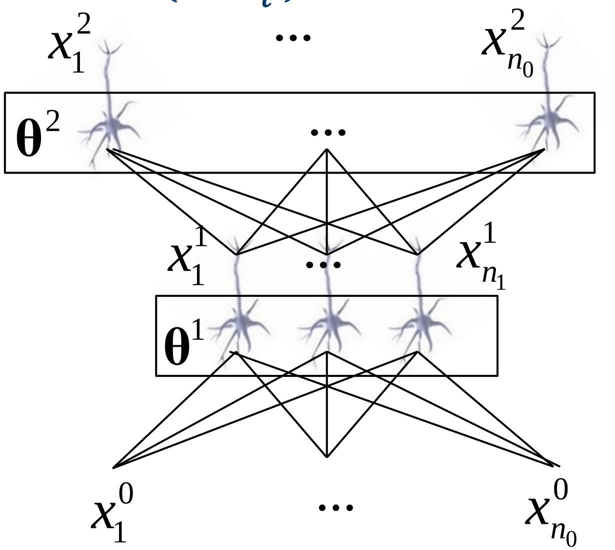
Auto Encoders via Backpropagation

- Desired output: $y_j = x_j^0$.
- Empirical risk: $\hat{R}(\theta) = \frac{1}{2m}\sum_{j=1}^{m}\left(\mathbf{x}_j^0 - \mathbf{x}_j^1\right)^2$
- Train with standard back propagation, using the input as target output values.



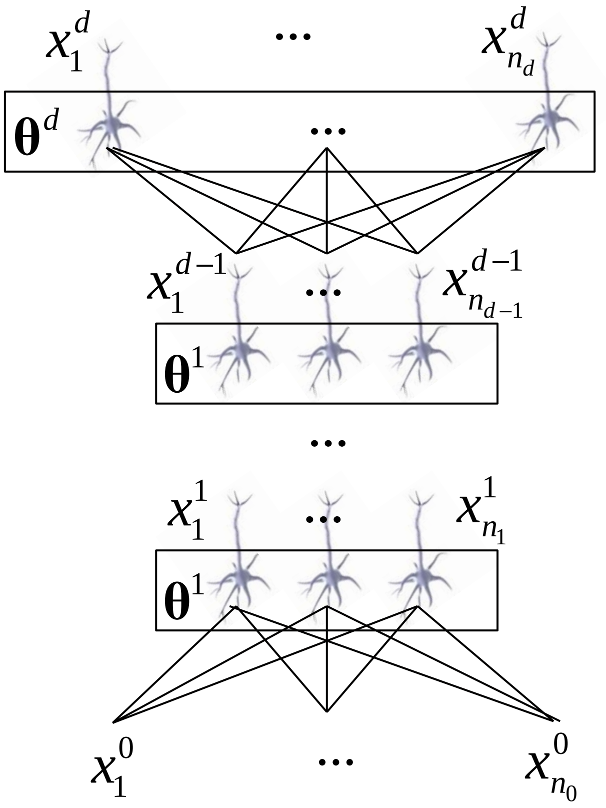
Auto Encoders via Backpropagation

- Additional regularization: hidden units should be sparse (i.e., have activation 0 most of the time).
- Minimize KL divergence between $\boldsymbol{\rho} = (\rho, \ldots, \rho)$ and activation of hidden units.
  - $KL(\boldsymbol{\rho} || \mathbf{x}^1) = \sum_{i=1}^{n_1} \rho \log \frac{\rho}{x_i^1} + (1-\rho) \log \frac{(1-\rho)}{(1-x_i^1)}$
- Modified backprop update:
  - $\delta_k^2 = \sigma'(h_k^2) \sum_l \delta_l^3 \theta_{lk}^3 + \beta \left(\frac{-\rho}{x_k^1} + \frac{1-\rho}{1-x_k^1} \right)$



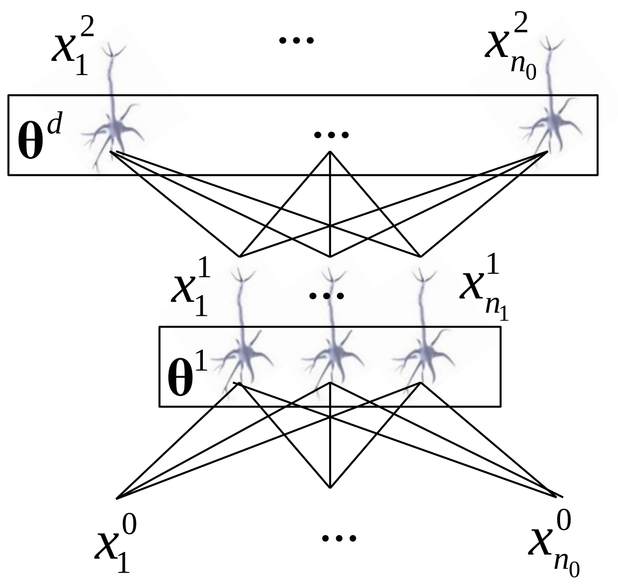
Deep Learning: Stacked Autoencoders

- Multiple hidden layers, each layer has fewer units.
- Autoencoder has to reproduce the input vector.
- $\hat{R}(\theta) = \frac{1}{2m}\sum_{j=1}^{m}\left(\mathbf{x}_j^0 - \mathbf{x}_j^d\right)^2$
- $n_0 > n_1 > \cdots > n_{d-1}$,
- $n_0 = n_d$.



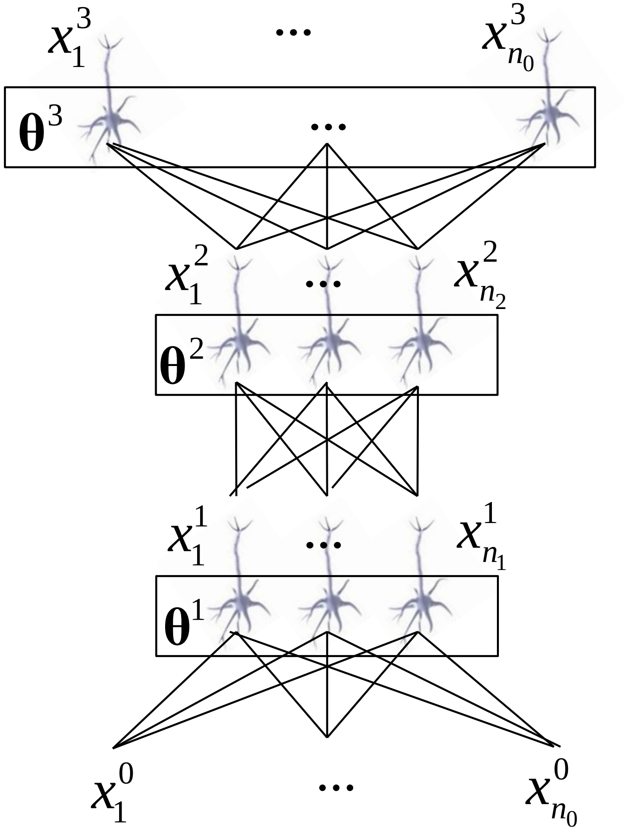
Stacked Autoencoders: Learning

- Step 1: Learn autoencoder using back propagation
  - Run back propagation until convergence.
- $\hat{R}(\theta) = \frac{1}{2m}\sum_{j=1}^{m}\left(\mathbf{x}_j^0 - \mathbf{x}_j^2\right)^2$



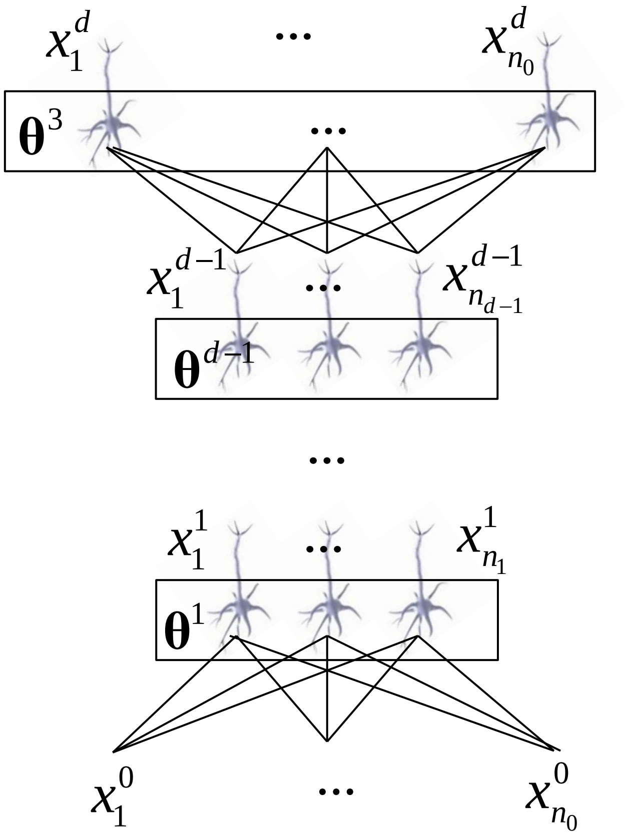
Stacked Autoencoders: Learning

- Step 2: Freeze $\boldsymbol{\theta}^1$, add another layer.
  - Train $\boldsymbol{\theta}^2$ and $\boldsymbol{\theta}^3$ using back propagation
- $\hat{R}(\theta) = \frac{1}{2m}\sum_{j=1}^{m}\left(\mathbf{x}_j^0 - \mathbf{x}_j^3\right)^2$



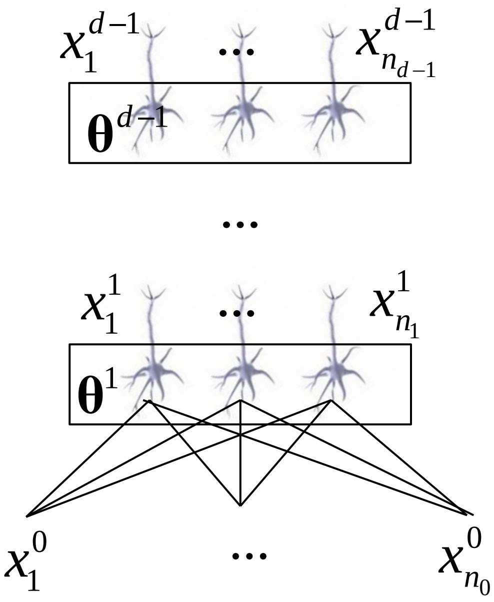
Stacked Autoencoders: Learning

- Step d: Freeze $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{d-2}$, add another layer.
  - Train $\boldsymbol{\theta}^{d-1}$ and $\boldsymbol{\theta}^d$ using back propagation
- $\hat{R}(\theta) = \frac{1}{2m} \sum_{j=1}^{m} \left(\mathbf{x}_j^0 - \mathbf{x}_j^d\right)^2$



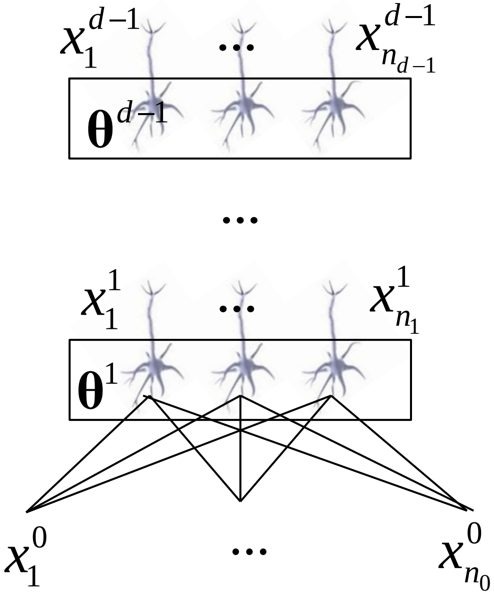
Stacked Autoencoders: Learning

- Step $d + 1$: Remove layer d.
- The result is a hierarchical feature representation.
- These features can be very useful for supervised learning, in particular for convolutional networks.



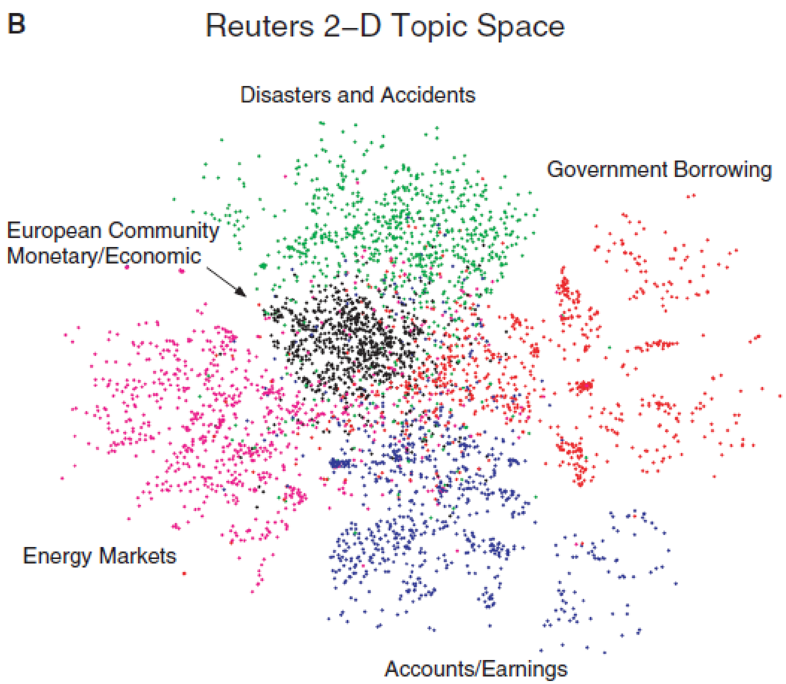
Denoising Autoencoders

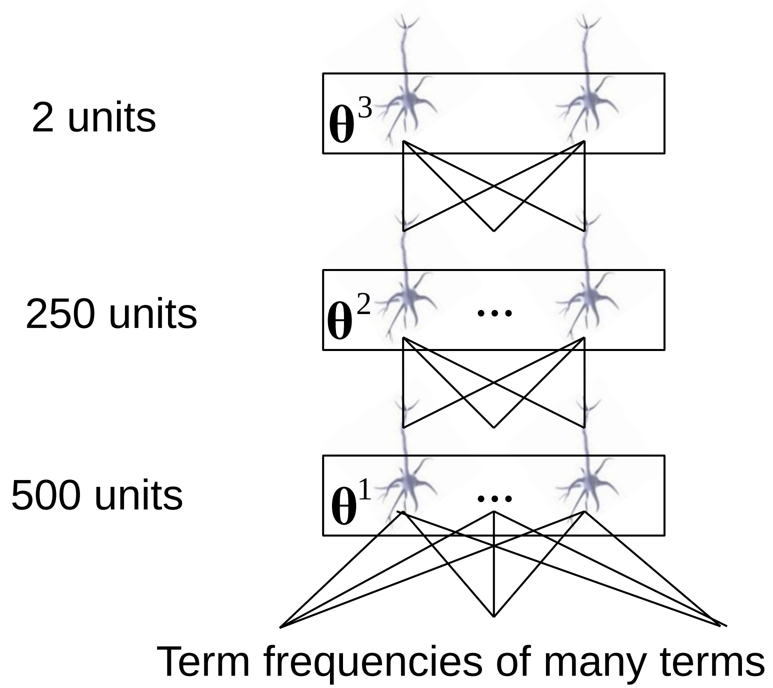
- Additional regularization: Reconstruct input from corrupted version of the input.
- Randomly set a fraction of the input to zero; use uncorrupted input as target for loss function.
- Tends to lead to more robust representations.



Stacked Autoencoder: Example

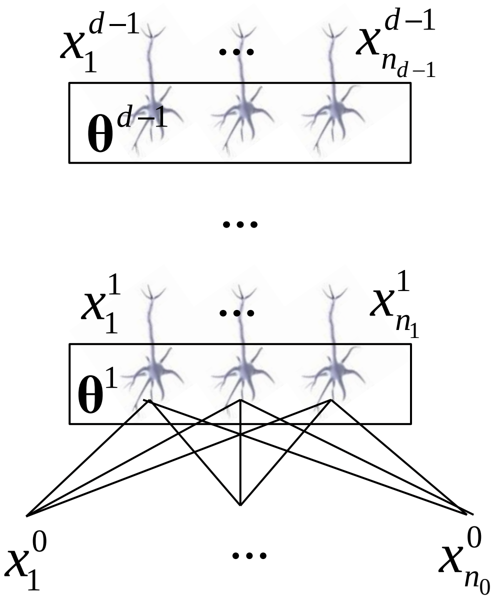
- 2D visualization of a document corpus
- TF vector → 500 hidden units → 250 hidden units → 2 hidden units (dimensions)





Auto Encoders: What are they Good for?

- Hidden units represent a clustering of the inputs.
- Hidden units are features that can now be used for a classification task.
  - For instance, face identification, hand-written letter recognition.



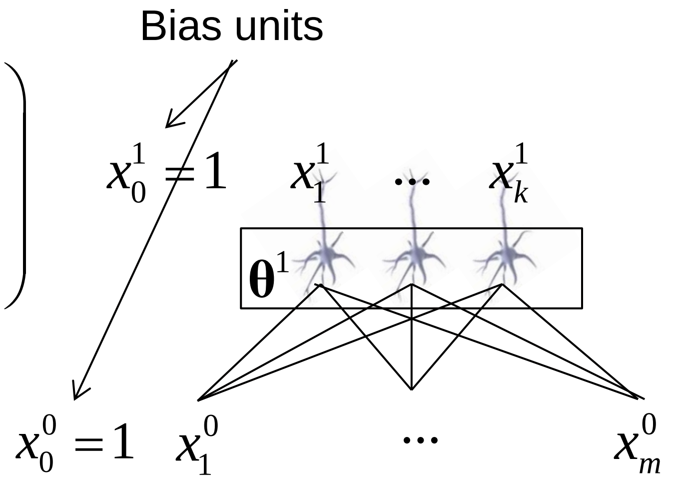
Intelligent Data Analysis II

Restricted Boltzmann Machine

- Unsupervised learning.
- Input layer and hidden layer.
- Binary stochastic units, one bias unit per layer.
- Generative, probabilistic model.
- Energy function:
  - $E(\mathbf{x}^0, \mathbf{x}^1) = -(\boldsymbol{\theta}^1 \mathbf{x}^0)^{\mathrm{T}} \mathbf{x}^1$

$$= -\left(\begin{pmatrix} \theta^1_{00} & \cdots & \theta^1_{0n_0} \\ \vdots & \ddots & \vdots \\ \theta^1_{n_1 0} & \cdots & \theta^1_{n_1 n_0} \end{pmatrix} \begin{pmatrix} x^0_0 \\ \vdots \\ x^0_{n_0} \end{pmatrix} \right)^{\mathrm{T}} \begin{pmatrix} x^1_0 \\ \vdots \\ x^1_{n_1} \end{pmatrix}$$

=0 (Bias units
Are not connected)

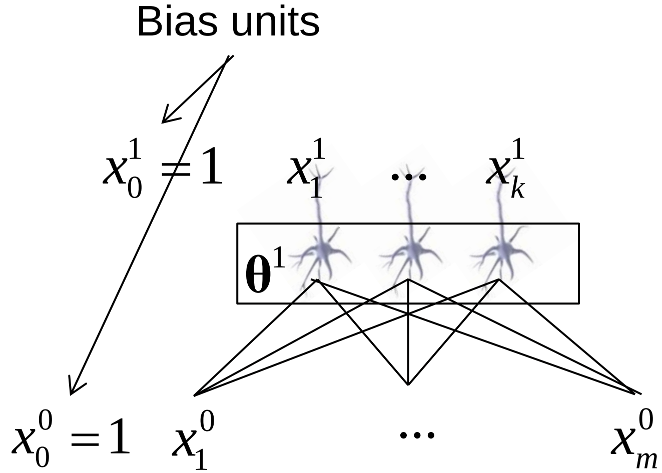


Restricted Boltzmann Machine

- Energy function:
  - $E(\mathbf{x}^0, \mathbf{x}^1) = -(\boldsymbol{\theta}^1 \mathbf{x}^0)^\mathrm{T} \mathbf{x}^1$
- Energy function $\sim -\log P(\text{activation})$
  - $P(\mathbf{x}^0, \mathbf{x}^1) = \frac{1}{Z} e^{-E(\mathbf{x}^0, \mathbf{x}^1)}$
  - $P(\mathbf{x}^0) = \sum_{\mathbf{x}^1} \frac{1}{Z} e^{-E(\mathbf{x}^0, \mathbf{x}^1)}$
  - $P(\mathbf{x}^1 \mid \mathbf{x}^0) = \dfrac{P(\mathbf{x}^0, \mathbf{x}^1)}{P(\mathbf{x}^0)}$

$$= \frac{\frac{1}{Z} e^{-E(\mathbf{x}^0, \mathbf{x}^1)}}{\sum_{\mathbf{x}^1} \frac{1}{Z} e^{-E(\mathbf{x}^0, \mathbf{x}^1)}} = \frac{1}{1 + e^{\boldsymbol{\theta}\mathbf{x}^0}}$$

- Z is normalization constant

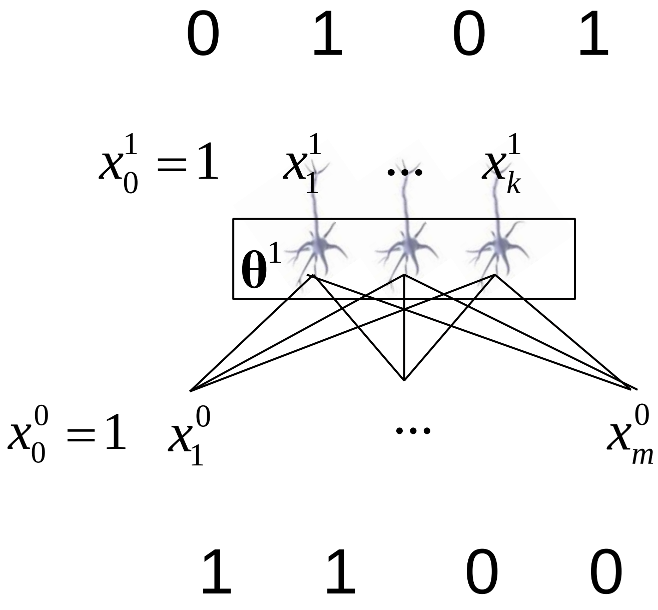


Inference in RBM

- RMB is a generative model; generates states like a Bayesian network.
- Inference by Markov chain monte carlo:
  - Iterate over units, alternate between input and hidden units.
  - Draw unit activation given activations of neighboring units.
- After burn-in phase, Markov chain of activations is governed by distribution of the encoded RBM.

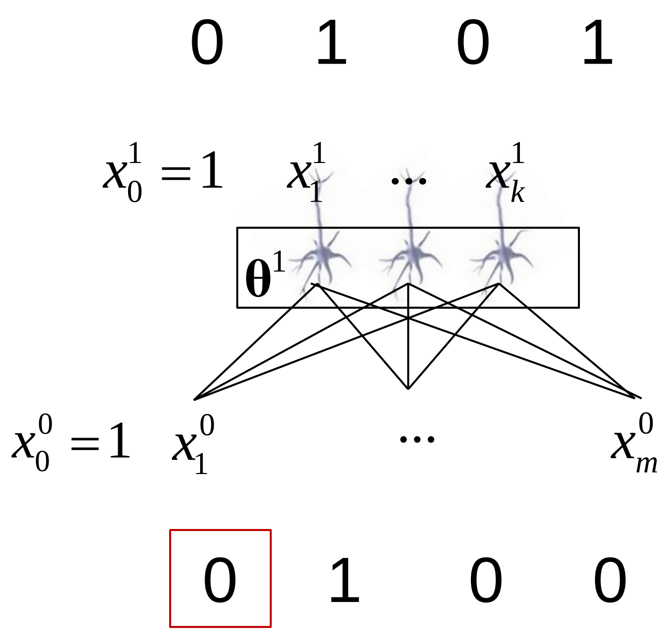
RBM: Sampling of States

- Initialize states at random



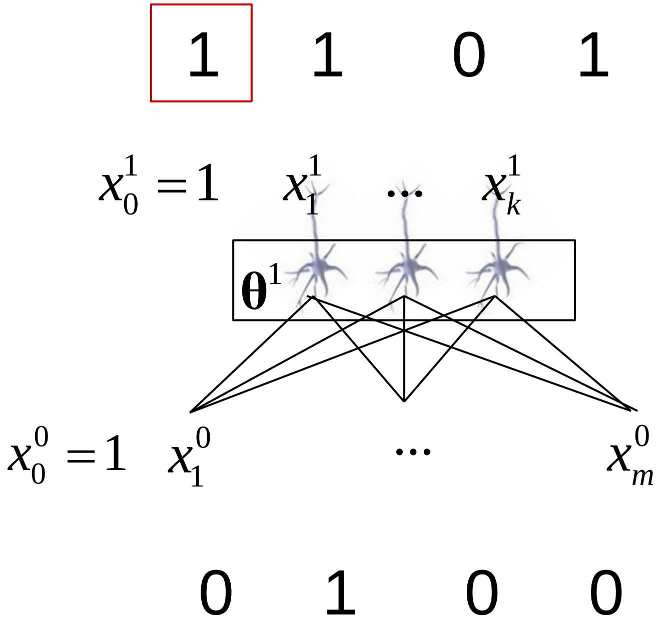
RBM: Sampling of States

- Initialize states at random
- $P(x_1^0 | \mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$



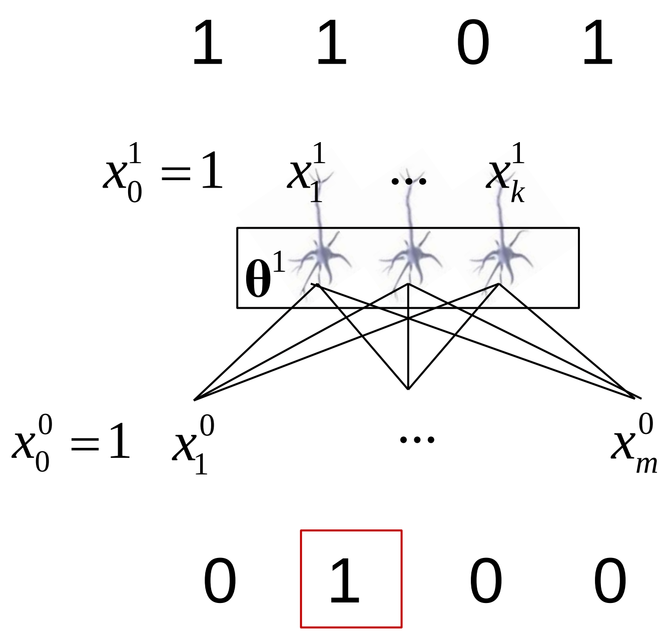
RBM: Sampling of States

- Initialize states at random
- $P(x_1^0 | \mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$
- $P(x_1^1 | \mathbf{x}^0) = \frac{1}{1+e^{\theta_0^1 \mathbf{x}^0}}$



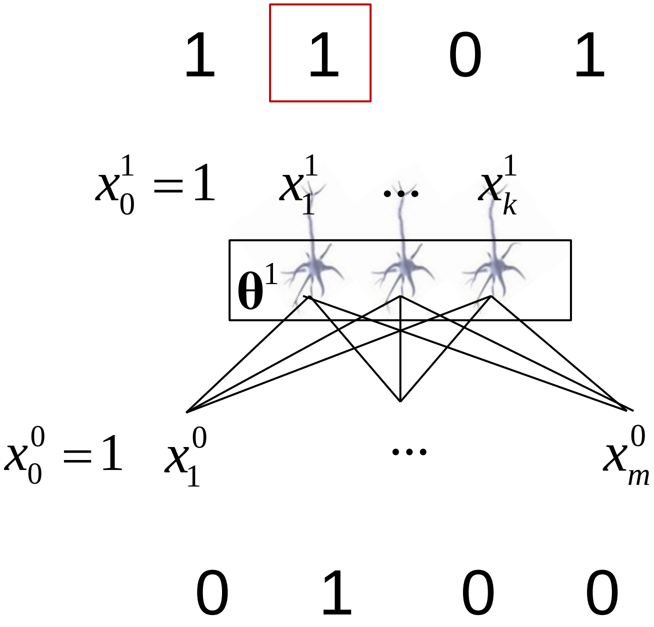
RBM: Sampling of States

- Initialize states at random
- $P(x_1^0 | \mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$
- $P(x_1^1 | \mathbf{x}^0) = \frac{1}{1+e^{\theta_0^1 \mathbf{x}^0}}$
- $P(x_2^0 | \mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$



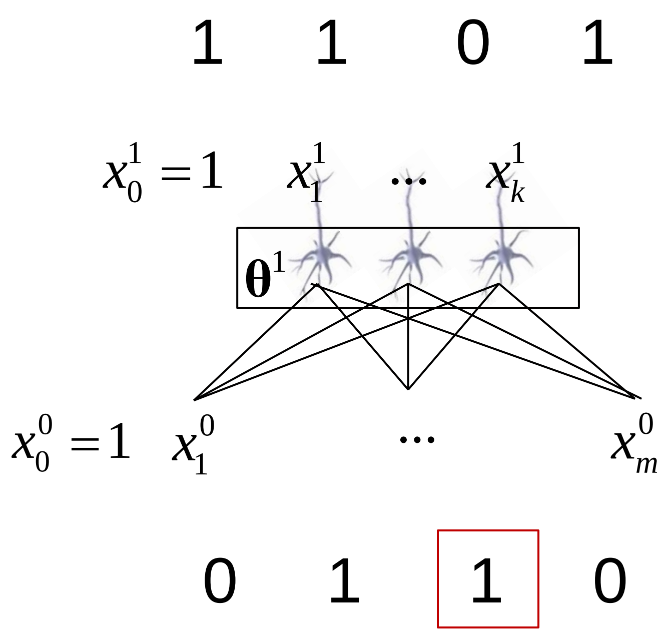
RBM: Sampling of States

- Initialize states at random
- $P(x_1^0 | \mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$
- $P(x_1^1 | \mathbf{x}^0) = \frac{1}{1+e^{\theta_0^1 \mathbf{x}^0}}$
- $P(x_2^0 | \mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$
- $P(x_2^1 | \mathbf{x}^0) = \frac{1}{1+e^{\theta_0^1 \mathbf{x}^0}}$



RBM: Sampling of States

- Initialize states at random
- $P(x_1^0|\mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$
- $P(x_1^1|\mathbf{x}^0) = \frac{1}{1+e^{\theta_0^1 \mathbf{x}^0}}$
- $P(x_2^0|\mathbf{x}^1) = \frac{1}{1+e^{\theta_0^0 \mathbf{x}^1}}$
- $P(x_2^1|\mathbf{x}^0) = \frac{1}{1+e^{\theta_0^1 \mathbf{x}^0}}$
- ...



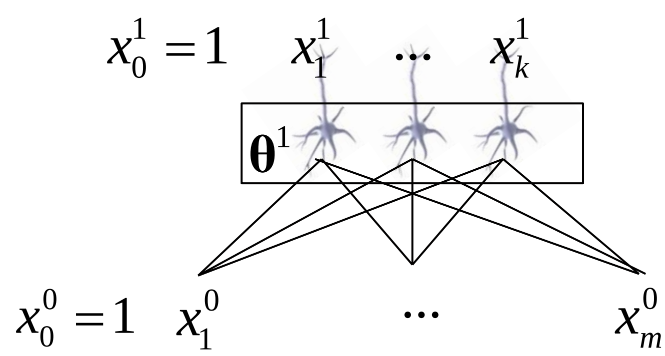
Restricted Boltzmann Machine: Learning

- Learning: maximize log-likelihood of the input vectors.
  - $\arg\max_{\boldsymbol{\theta}^1} -\log P(\mathbf{x}^0)$
- Gradient:

Energy gradient for observed input

Marginal energy gradient

  - $$-\frac{\partial \log p(\mathbf{x}^0)}{\partial \theta_{ji}^1}$$
  $$= \sum_{\mathbf{x}^1} p(\mathbf{x}^1 \mid \mathbf{x}^0) \frac{\partial E(\mathbf{x}^0, \mathbf{x}^1)}{\partial \theta_{ji}^1}$$
  $$- \sum_{\mathbf{x}^0, \mathbf{x}^1} p(\mathbf{x}^1, \mathbf{x}^0) \frac{\partial E(\mathbf{x}^0, \mathbf{x}^1)}{\partial \theta_{ji}^1}$$
- Energy gradient:
  - $$\frac{\partial E(\mathbf{x}^0, \mathbf{x}^1)}{\partial \theta_{ji}^1} = \frac{\partial -(\boldsymbol{\theta}^1 \mathbf{x}^0)^{\mathrm{T}} \mathbf{x}^1}{\partial \theta_{ji}^1} = -x_i^0 x_j^1$$



Restricted Boltzmann Machine: Learning

- Gradient:
  - $$-\frac{\partial \log p(\mathbf{x}^0)}{\partial \theta_{ji}^1} = \sum_{\mathbf{x}^1} p(\mathbf{x}^1 \mid \mathbf{x}^0) \frac{\partial E(\mathbf{x}^0, \mathbf{x}^1)}{\partial \theta_{ji}^1} - \sum_{\mathbf{x}^0, \mathbf{x}^1} p(\mathbf{x}^1 \mid \mathbf{x}^0) \frac{\partial E(\mathbf{x}^0, \mathbf{x}^1)}{\partial \theta_{ji}^1}$$
  - Mit $\frac{\partial E(\mathbf{x}^0, \mathbf{x}^1)}{\partial \theta_{ji}^1} = \frac{\partial -(\boldsymbol{\theta}^1 \mathbf{x}^0)^{\mathrm{T}} \mathbf{x}^1}{\partial \theta_{ji}^1} = -x_i^0 x_j^1$

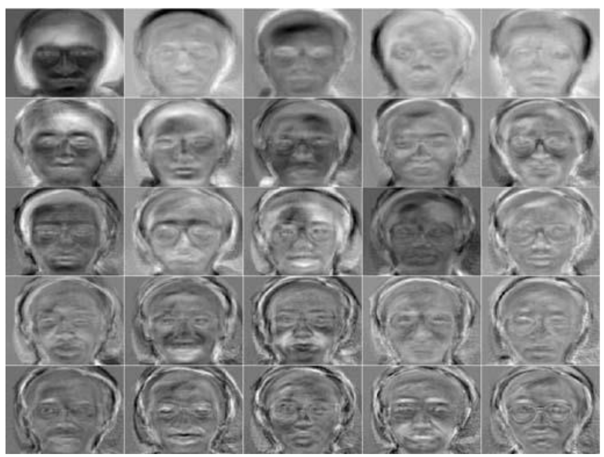
- Weight update
  - $\theta_{ji}^1{}' = \theta_{ji}^1 + \alpha(x_i^0 h_j^1 - x^0 h_j^1)$

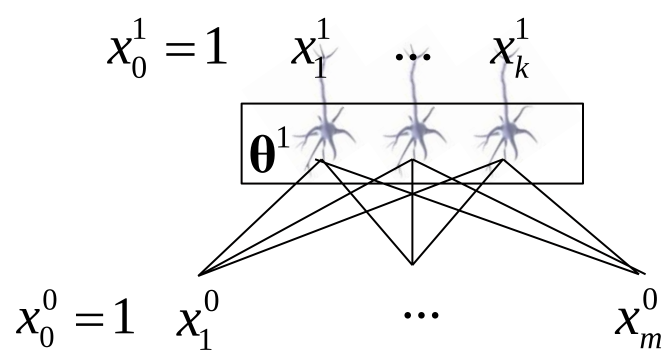
Observed input

Input inferred in an MCMC step

Restricted Boltzman Machine: Example

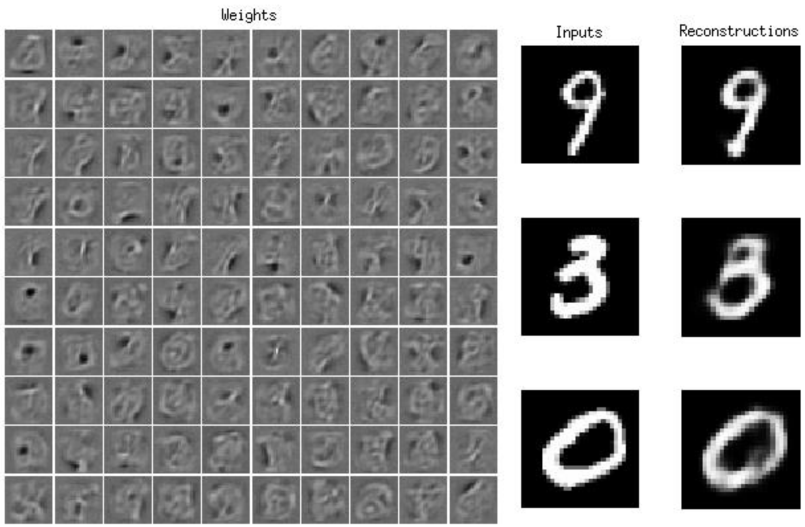
- Unsupervised learning of a representation, similar to hidden layer of an autoencoder.
- 25 weight vectors $\boldsymbol{\theta}_i^1$ after training with a set of aligned faces:

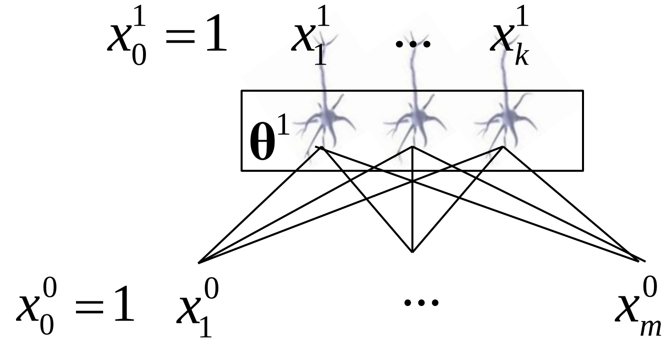




Restricted Boltzman Machine: Example

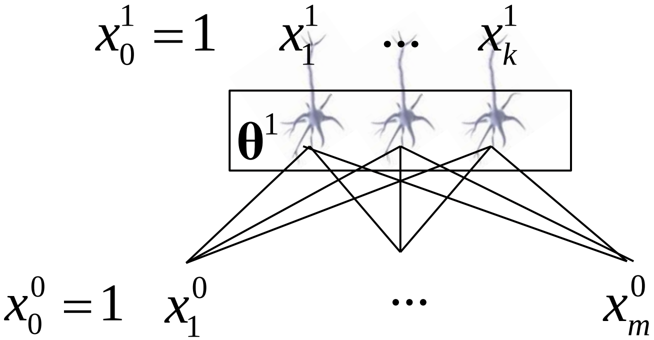
- Unsupervised learning of a representation, similar to hidden layer of an autoencoder.
- Weights $\boldsymbol{\theta}_i^1$ after training with a set of hand-written digits:





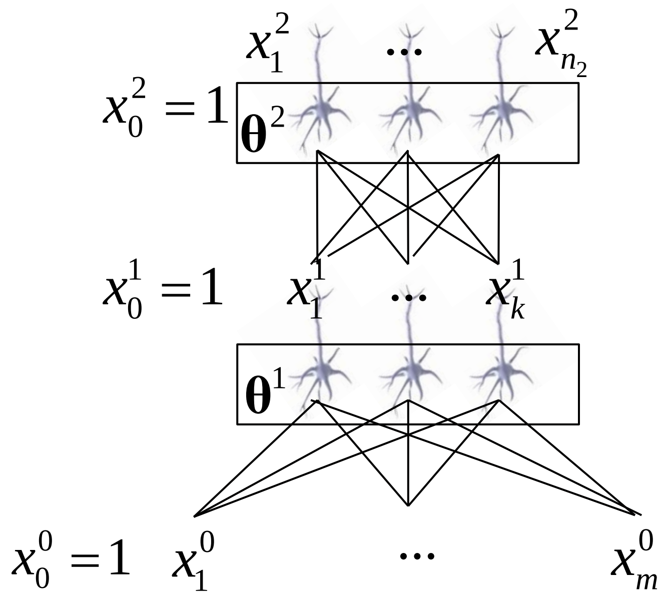
Stacked RBM Learning

- Stacked RBMs analogous to stacked backpropagation autoencoders.
- Step 1: $\text{argmax}_{\boldsymbol{\theta}^1}\{-\log P(\mathbf{x}^0)\}$



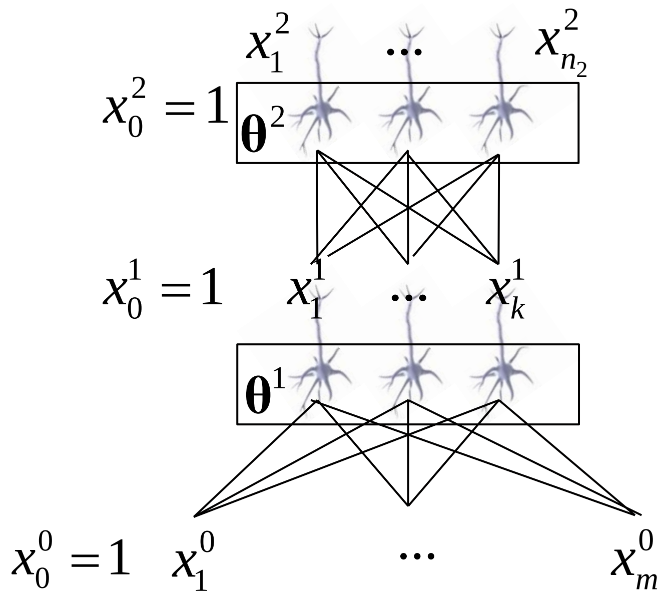
Stacked RBM Learning

- Stacked RBMs analogous to stacked backpropagation autoencoders.
- Step 1: $\text{argmax}_{\boldsymbol{\theta}^1}\{-\log P(\mathbf{x}^0)\}$
- Step 2: $\text{argmax}_{\boldsymbol{\theta}^2}\{-\log P(\mathbf{x}^0)\}$
- …



RBM: What are they Good for?

- Hidden units represent a clustering of the inputs.
- Hidden units are features that can then be used for supervised learning.



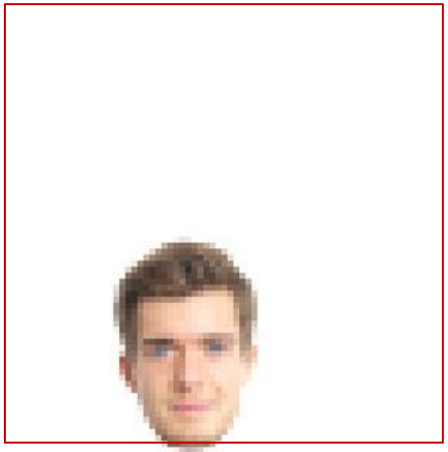
Overview

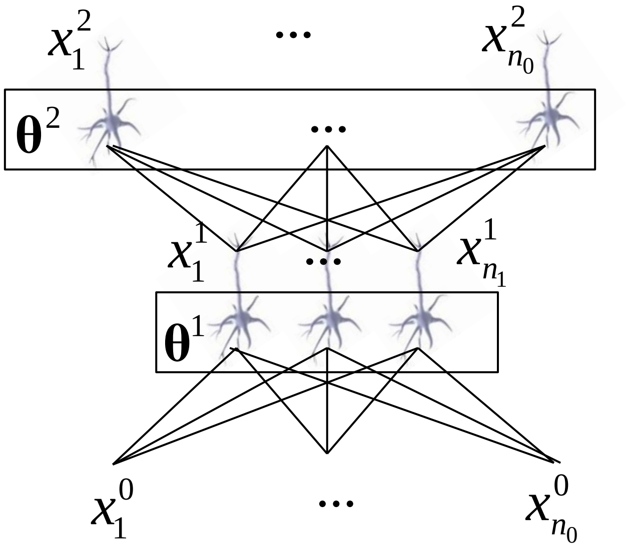
- Neural information processing.
- Feed-forward networks.
- Training feed-forward networks, back propagation.
- Unsupervised learning:
  - Auto encoders.
  - Training auto encoders via back propagation.
  - Restricted Boltzmann machines.
- Convolutional networks.

Convolutional Networks

- What happens when an autoencoder is trained with unaligned images of faces?

Training images

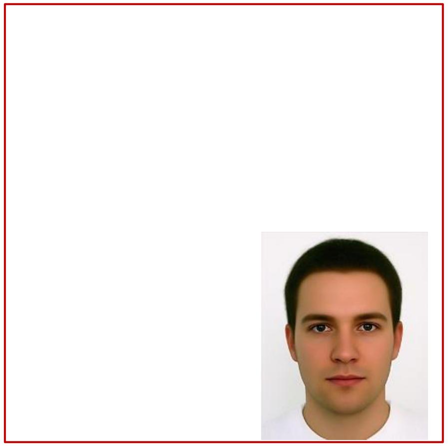


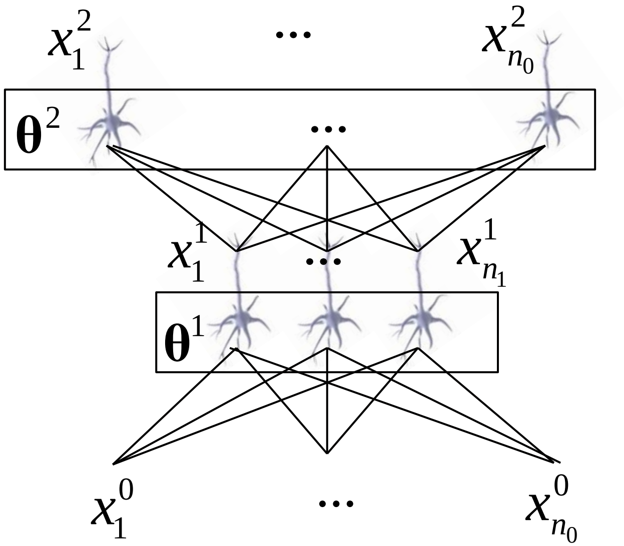


Convolutional Networks

- What happens when an autoencoder is trained with unaligned images of faces?

Training images

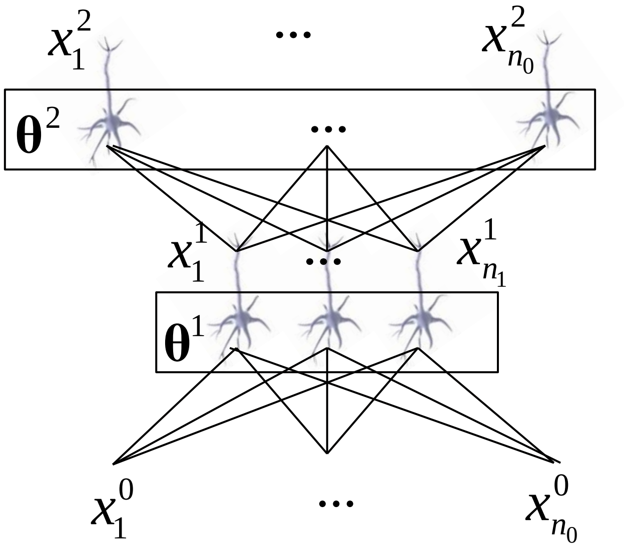




Convolutional Networks

- What happens when an autoencoder is trained with unaligned images of faces?
- Weights $\boldsymbol{\theta}^1$ become blurry; hidden units tend to represent different face positions rather than different looks of faces.

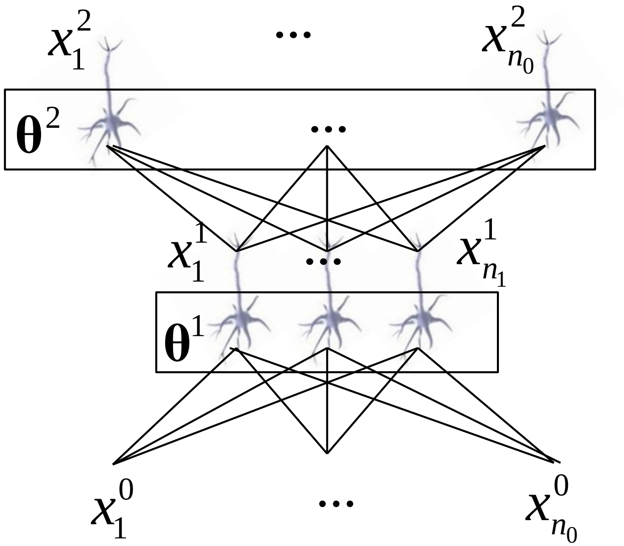
Training images



Convolutional Networks

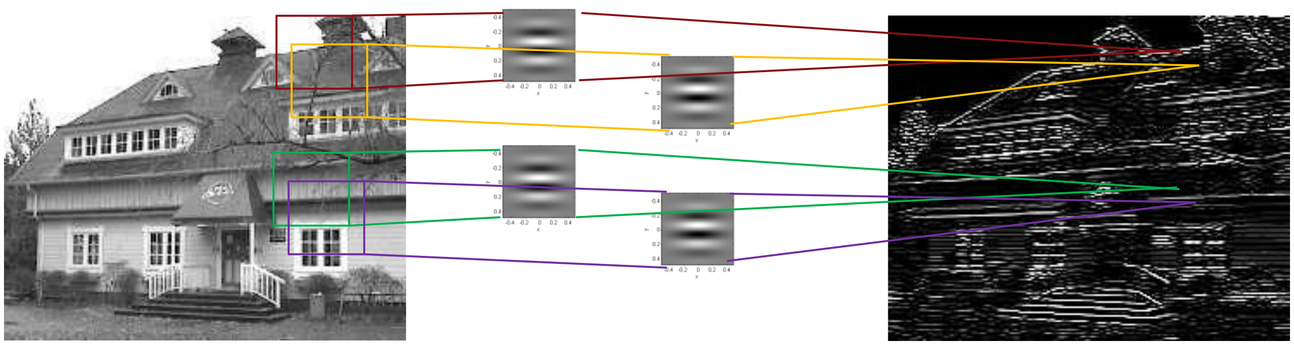
- Idea: Have detectors find common patterns at different positions of the input image and at different scales.

Training images



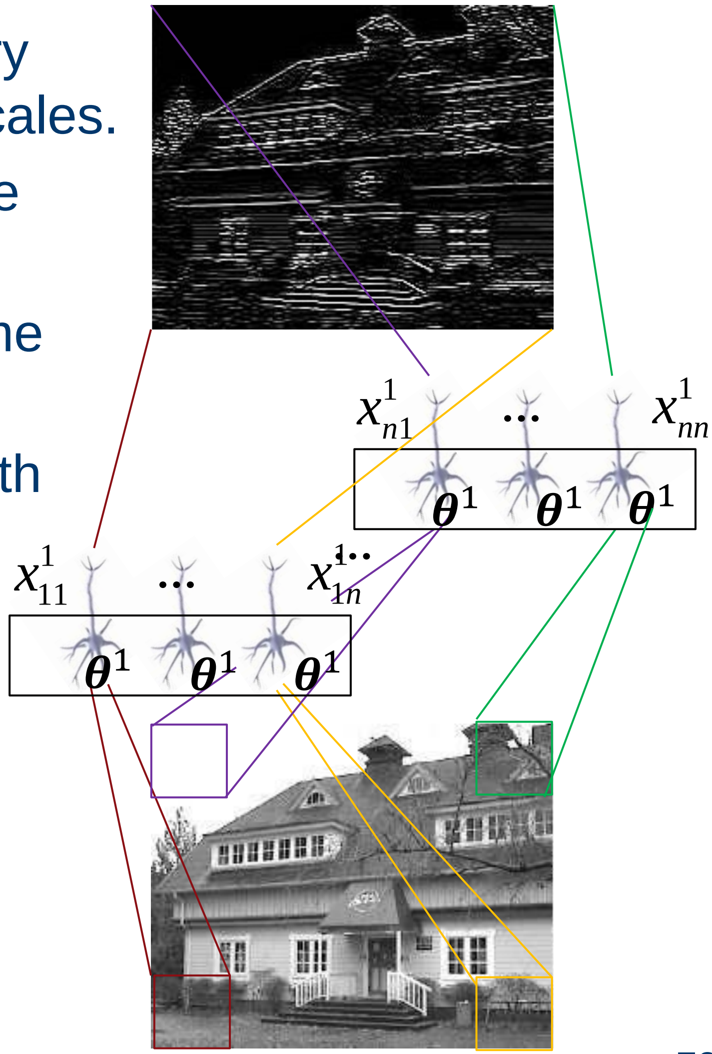
Convolution

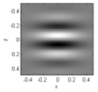
- Multiplication of a filter with an area of the input gives intensity of the filter signal at the position.
  - $x_{ij}^{1} = \sum_{k=-n}^{n} \sum_{l=-n}^{n} x_{i+k,\,j+l}^{0} \theta_{kl}$
- Pixel in result image $x_{[1\ldots n][1\ldots n]}^{1}$ is the result of a convolution.
- Used for images, audio signals.
  - E.g., detection of edges (greyvalue gradients).



Convolutional Networks

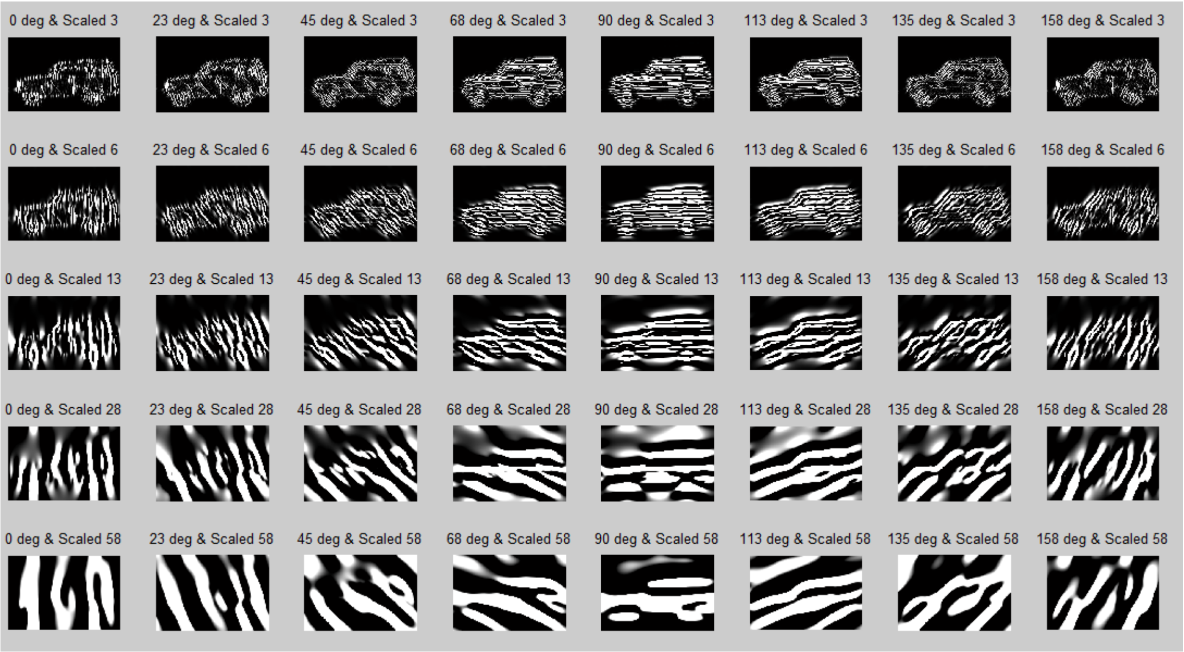
- Apply one or several filters to every position and possibly at several scales.
- Each unit produces output of same filter for different position.
- All units for one filter have the same weight.
- Example convolutional network with fixed scale and a single filter.



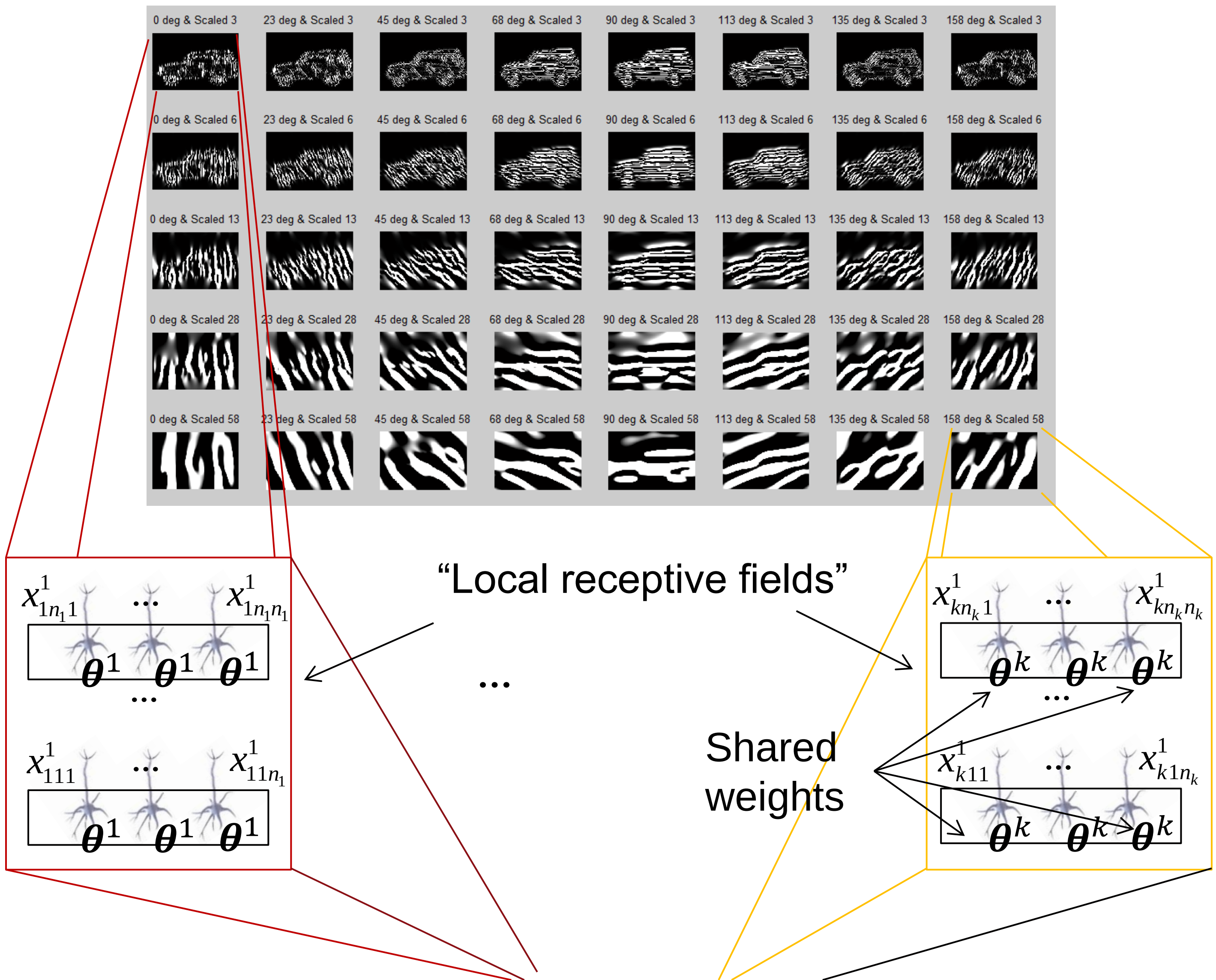
$\theta^1=$ 

Multiple Convolutions

- Multiple detectors per location and scale
  - E.g., edges of varying orientation
  - Edges of varying scale
- Results in an array of convoluted result images.

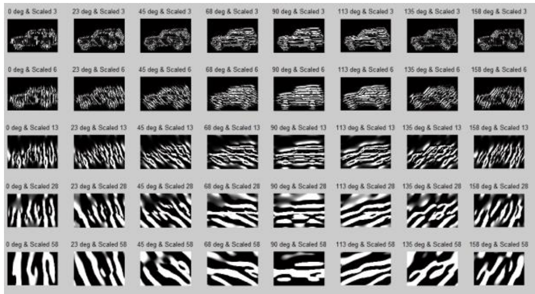


Convolutional Networks: Example



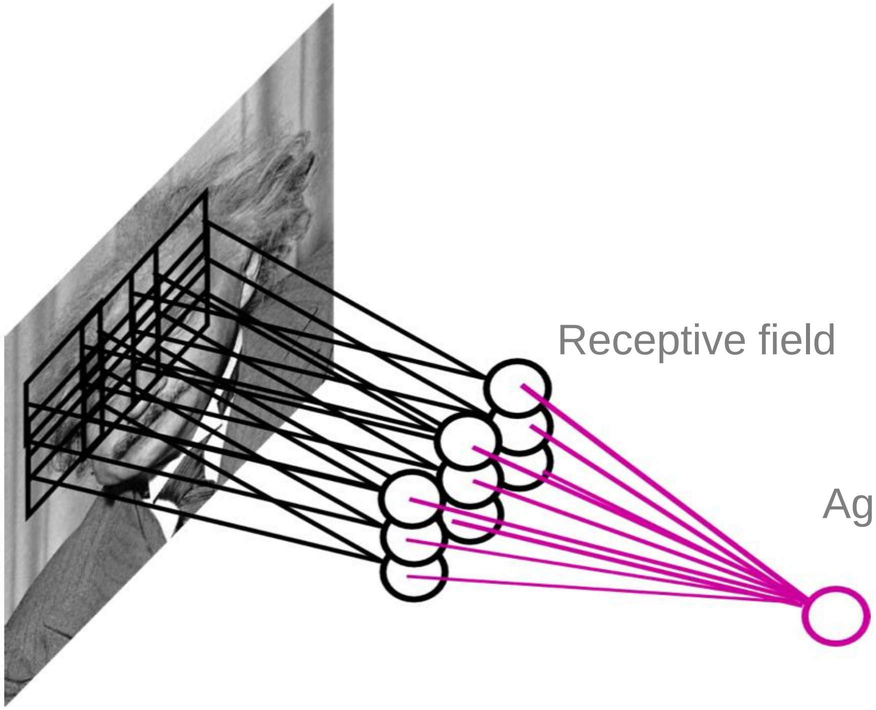
Convolutional Networks: Example

- Convolutional layer with k filters takes an input image and produces k images.
- k images have to be aggregated by pooling layers.



Convolutional Networks: Pooling Layers

- MaxPooling-Layer:
  - Split layer into non-overlapping areas.
  - For each area, pass on maximal value to next layer.



Stacked Convolutional Networks

- Local receptive fields are trained layer-wise restricted Boltzmann machine.
- Weight coupling has to be observed in the implementation (one parameter vector per filter, applied to all areas and at all scales).
- Iteratively, train next layer on output of previous pooling layer.

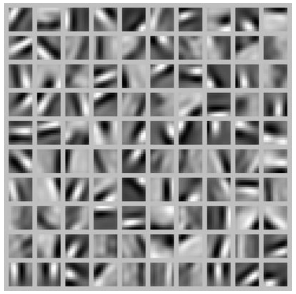
Stacked Convolutional RBM

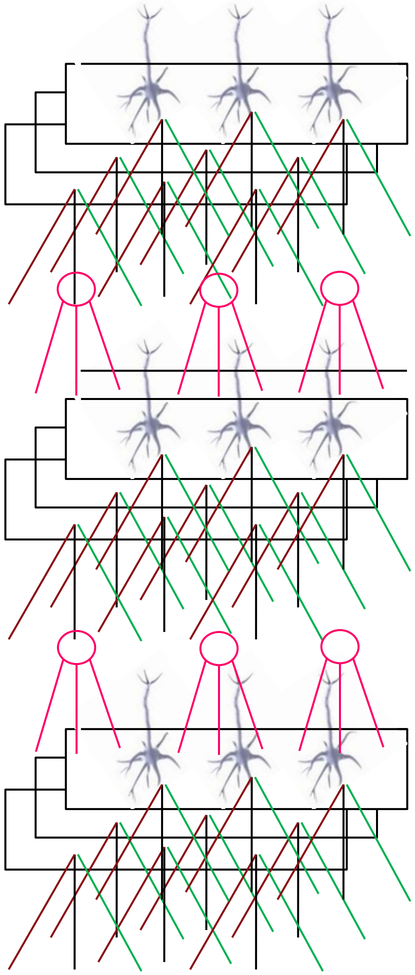
- Trained on faces, cars, motorbikes, airplanes

$\boldsymbol{\theta}^3$

$\boldsymbol{\theta}^2$

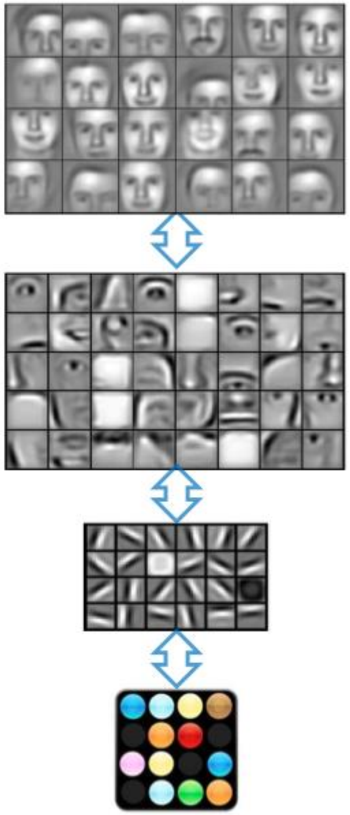
$\boldsymbol{\theta}^1$

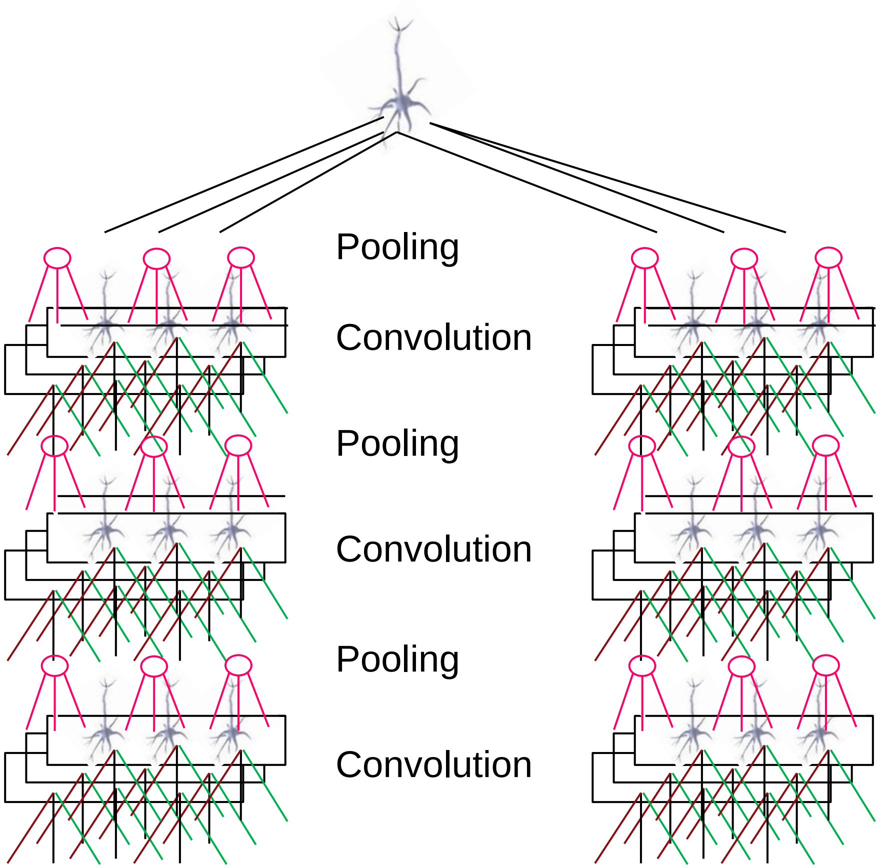




DeepFace: Face Identification

Discriminative layer: same person or not? Layer is trained on labeled data.





Convolutiopnal layers trained with unlabeled data

GPU Training

- GPUs are suitable to parallelize neural network training
  - Matrix multiplications, convolutions, element-wise operations
- GPUs oftware
  - CUDA: NVIDIA C-API
  - OPENCL: not specific to NVIDIA
  - PyCUDA: Python-API
  - PyOPENCL: not specific to NVIDIA

Deep Learning

- Step-wise transformation of the input space into more abstract feature spaces.
- Features emanate as solution of an optimization problem.

- Image processing
  - Pixels $\rightarrow$ local grey-value gradients $\rightarrow$ object parts $\rightarrow$ objects.
- Natural-language text
  - Characters $\rightarrow$ words $\rightarrow$ chunks $\rightarrow$ clauses $\rightarrow$ sentences.
- Spoken language
  - Signal $\rightarrow$ spectral band $\rightarrow$ phone $\rightarrow$ phoneme $\rightarrow$ word $\rightarrow$ …

Summary

- Supervised neural network learning
  - Stochastic gradient: back propagation.
- Unsupervised learning:
  - Autoencoders learn features that preserve the iunformation in the training instances
  - Stacked autoencoders.
  - (Stacked) restricted Boltzmann machines: generative models; inference by sampling
- (Stacked) convolutional networks:
  - Learn filters, apply filters to different regions, scales
  - Aggregate filter banks by pooling layers.
  - Increasingly abstract detectors applied to regions.

Seminar Lecture on Neural Networks?

- 3 x 30 minutes lecture incl. some time for questions.

- Natural-language description of images.
- Word2vec.
- Speech recognition.