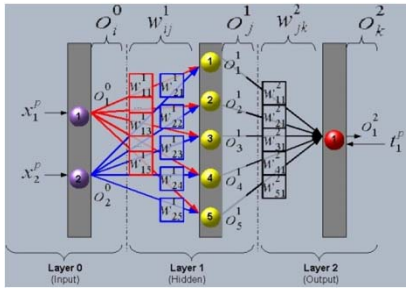
Probabilistic Graphical Models

Deep Learning and Graphical Models

Eric Xing

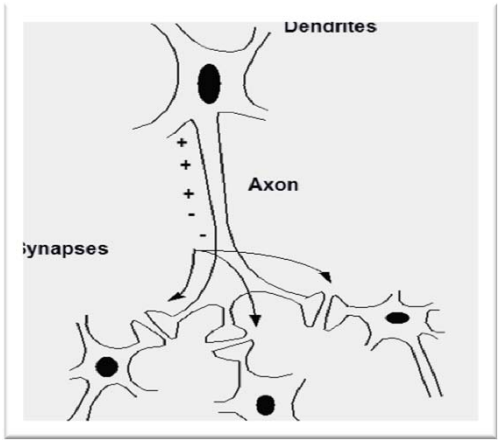


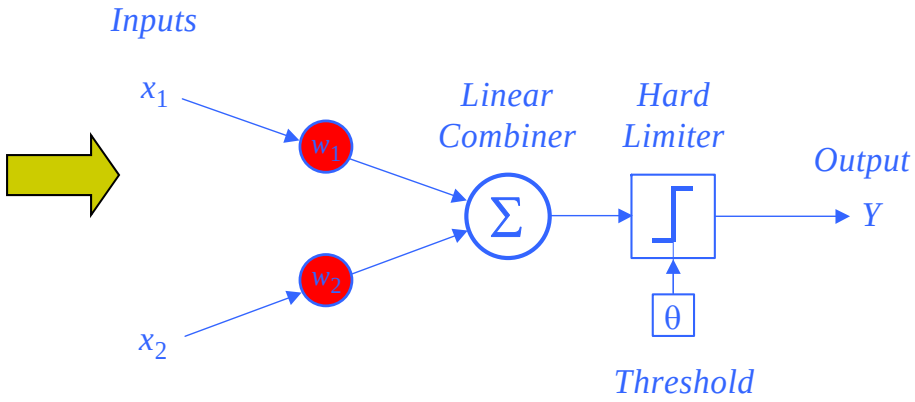
Lecture 25, April 15, 2015

Reading:

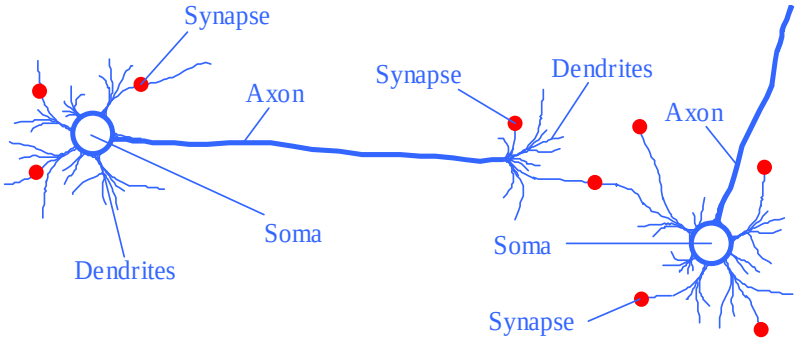
Perceptron and Neural Nets

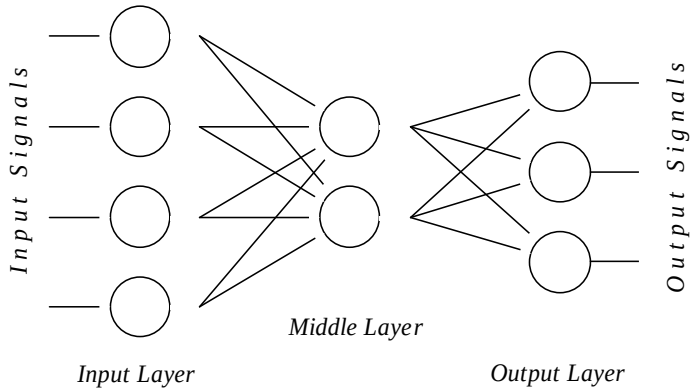
- From biological neuron to artificial neuron (perceptron)



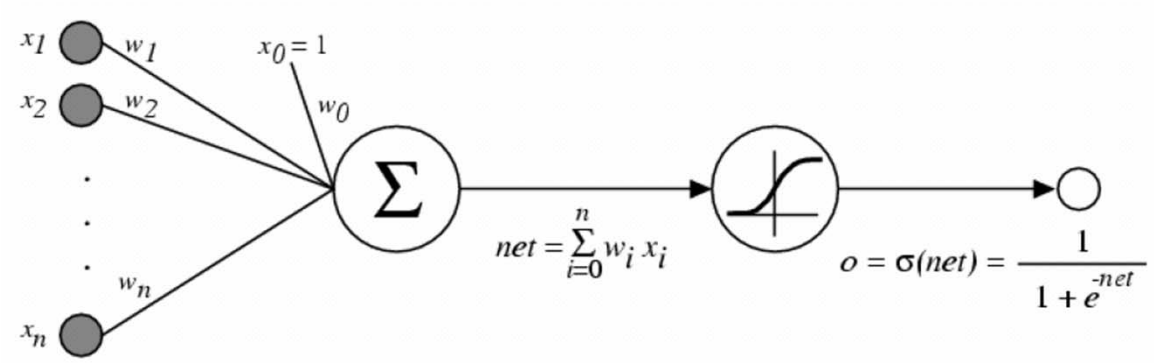


- From biological neuron network to artificial neuron networks





A perceptron learning algorithm



- Recall the nice property of sigmoid function $\frac{d\sigma}{dt} = \sigma(1-\sigma)$
- Consider regression problem f:X→Y , for scalar Y: $y = f(x) + \epsilon$
- We used to maximize the conditional data likelihood

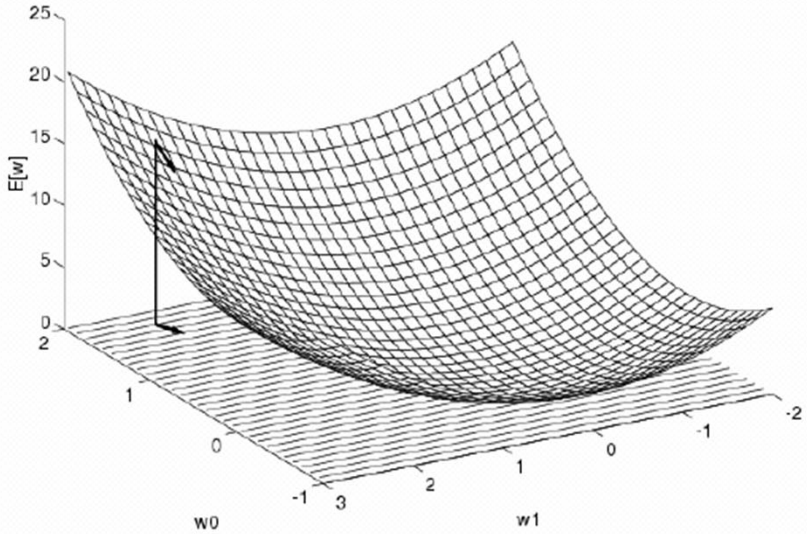
$$\vec{w} = \arg\max_{\vec{w}} \ln \prod_i P(y_i|x_i;\vec{w})$$

- Here …

$$\vec{w} = \arg\min_{\vec{w}} \sum_i \frac{1}{2}(y_i - \hat{f}(x_i;\vec{w}))^2$$

Gradient Descent

x_d = input
t_d = target output
o_d =observed unit output
w_i =weight i



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E[\vec{w}]}{\partial w_j} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum (t_d - o_d)^2$$
$$=$$

The perceptron learning rules

x_d = input
t_d = target output
o_d =observed unit output
w_i =weight i

$$\begin{aligned}
\frac{\partial E_D[\vec{w}])}{\partial w_j} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right) \\
&= -\sum_d (t_d - o_d)\frac{\partial o_d}{\partial net_i}\frac{\partial net_d}{\partial w_i} \\
&= -\sum_d (t_d - o_d) o_d(1 - o_d) x_d^i
\end{aligned}$$

Batch mode:

Do until converge:

1. compute gradient $\nabla E_D[w]$
2. $\vec{w} = \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode:

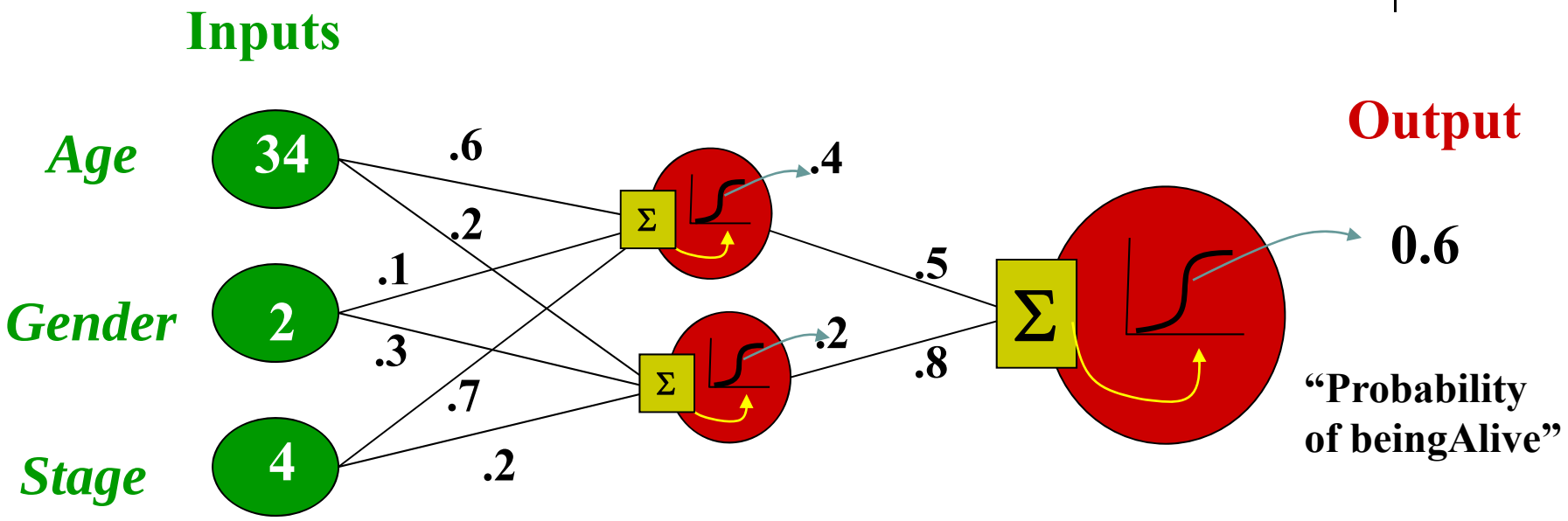
Do until converge:

- **For each training example *d* in *D***
  1. compute gradient $\nabla E_d[w]$
  2. $\vec{w} = \vec{w} - \eta \nabla E_d[\vec{w}]$

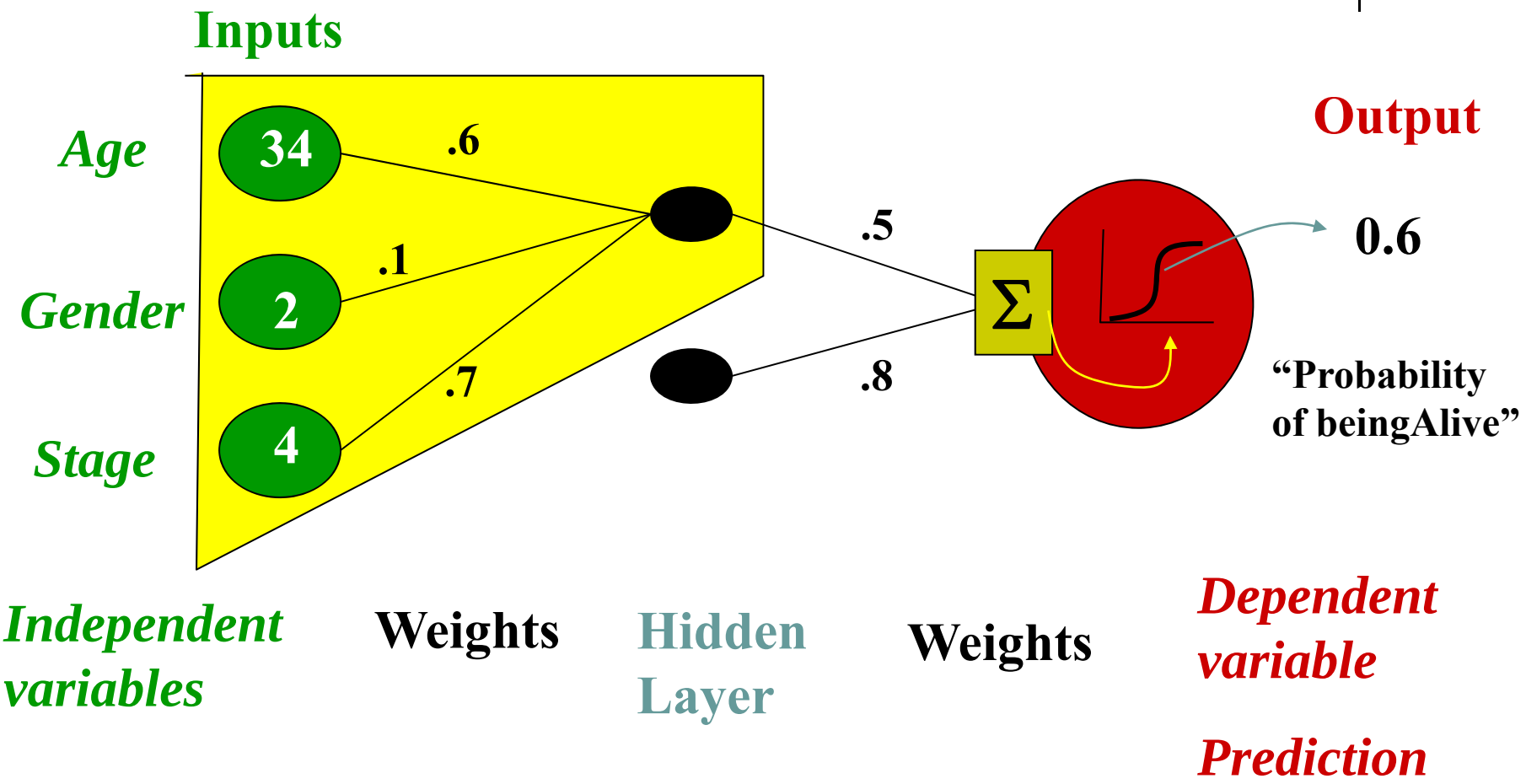
where

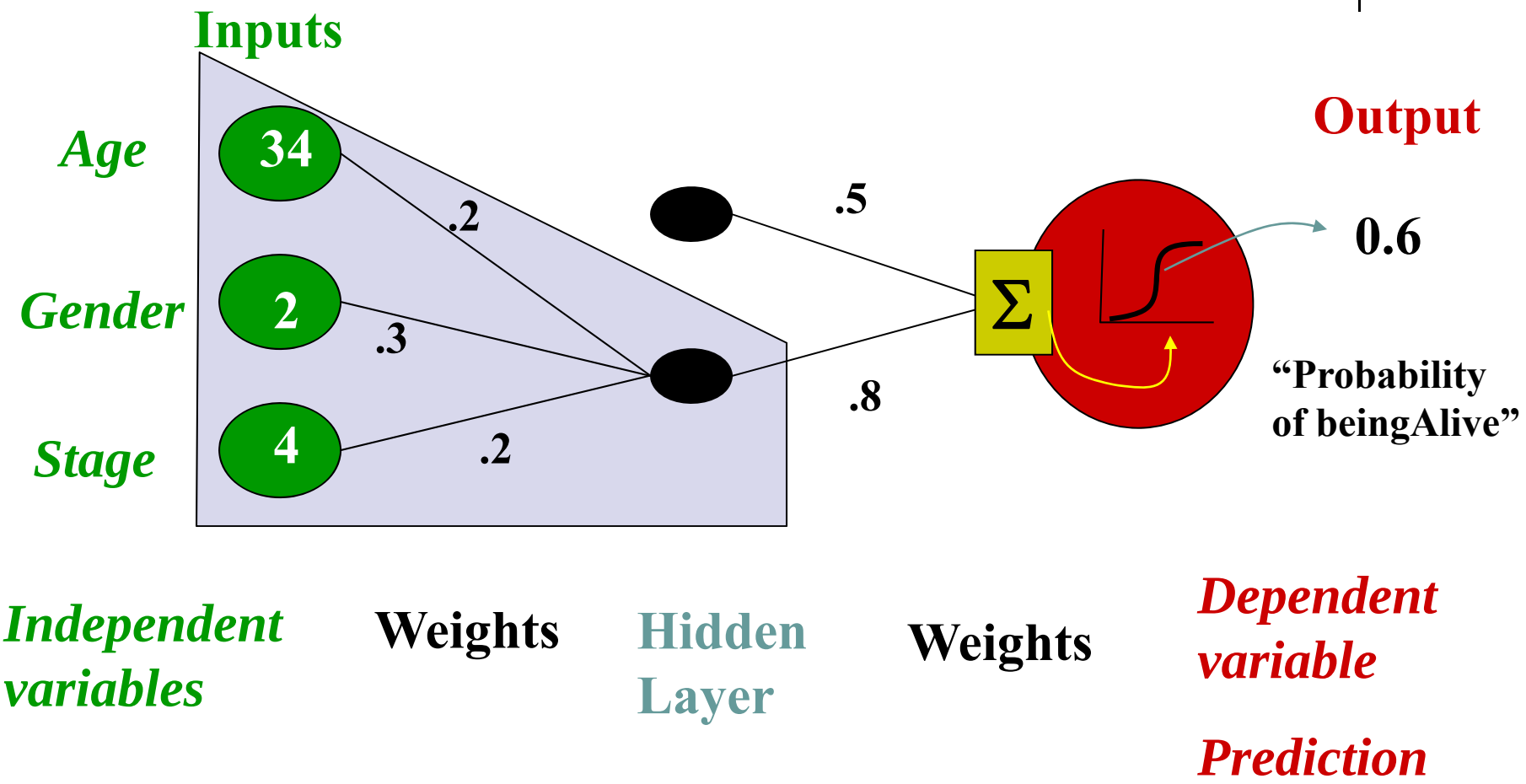
$$\nabla E_d[\vec{w}] = -(t_d - o_d) o_d(1 - o_d)\vec{x}_d$$

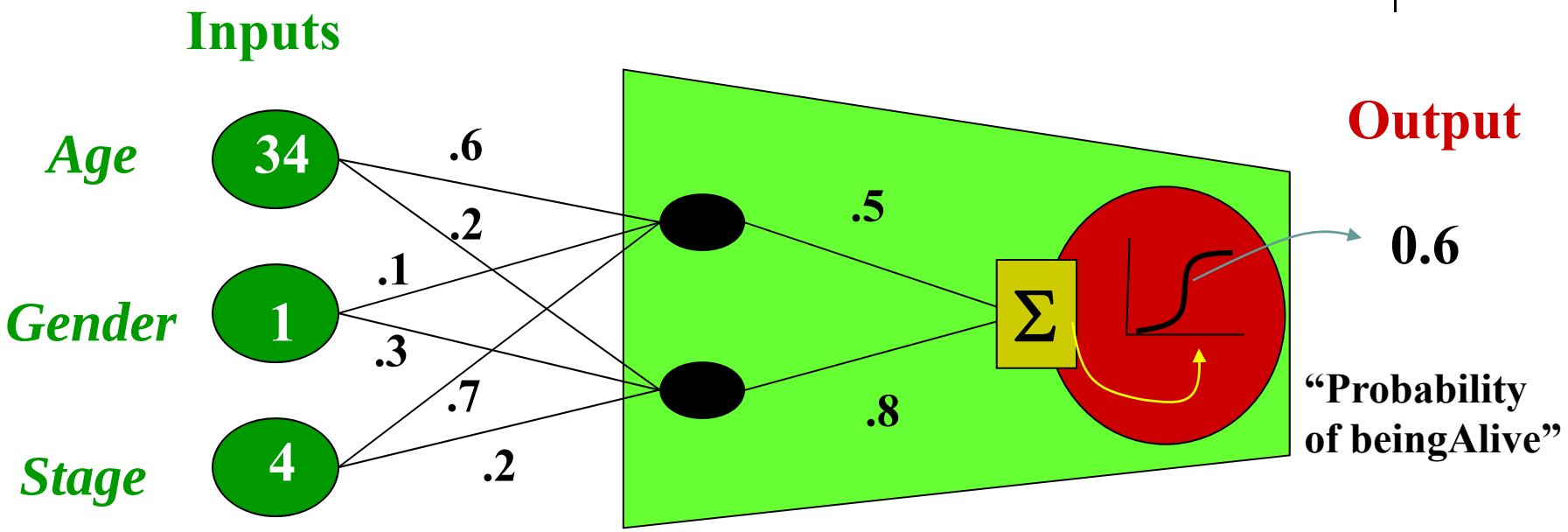
Neural Network Model



"Combined logistic models"







Independent variables

Weights

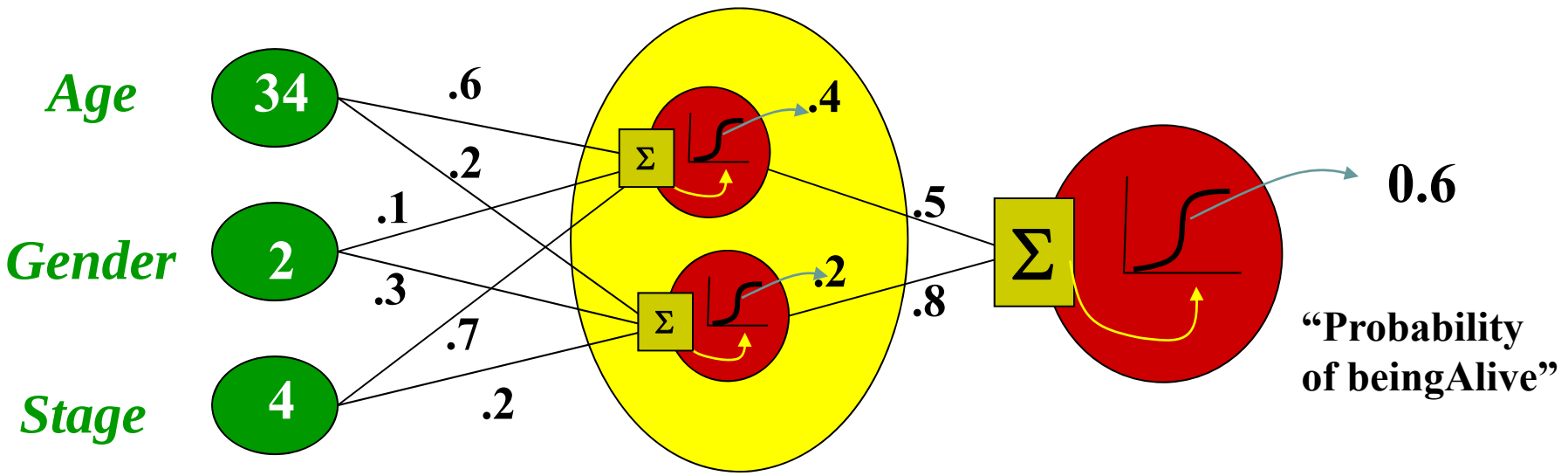
Hidden Layer

Weights

Dependent variable

Prediction

Not really,
no target for hidden units...



Independent variables

Weights

Hidden Layer

Weights

Dependent variable

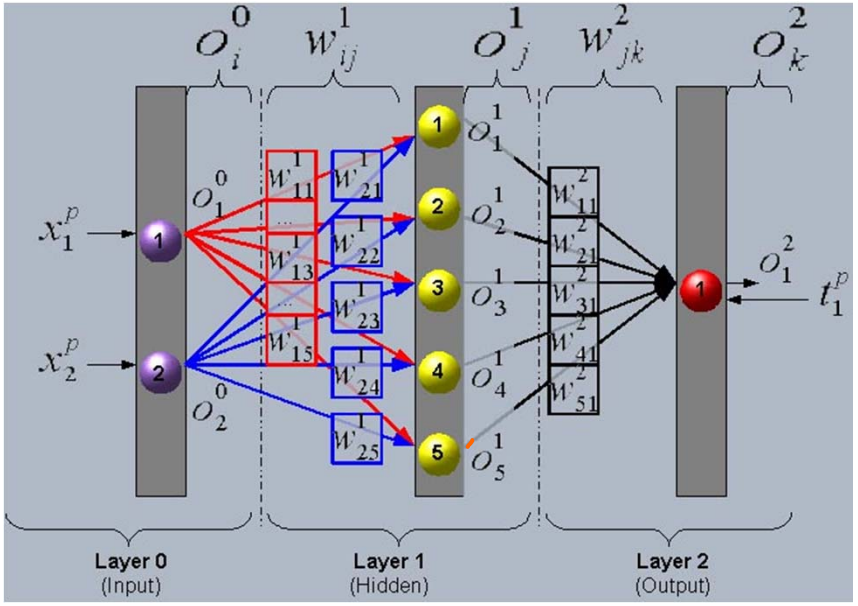
Prediction

Backpropagation Algorithm

x_d = input
t_d = target output
o_d =observed unit output
w_i =weight i

- Initialize all weights to small random numbers

  Until convergence, Do



1. Input the training example to the network and compute the network outputs

1. For each output unit k

$$\delta_k \leftarrow o_k^2(1-o_k^2)(t-o_k^2)$$

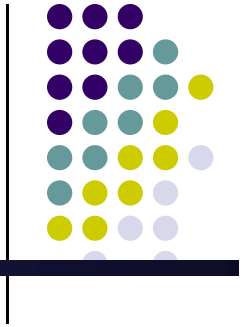
2. For each hidden unit h

$$\delta_h \leftarrow o_h^1(1-o_h^1) \sum_{k \in outputs} w_{h,k}\delta_k$$

3. Undate each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \quad \text{where} \quad \Delta w_{i,j} = \eta \delta_j x^j$$

More on Backpropatation

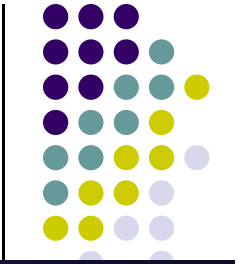


- It is doing gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

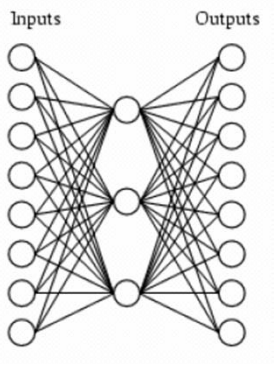
$$\Delta w_{i,j}(t) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(t-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent testing examples?
- Training can take thousands of iterations, → very slow!
- Using network after training is very fast

Learning Hidden Layer Representation



- A network:

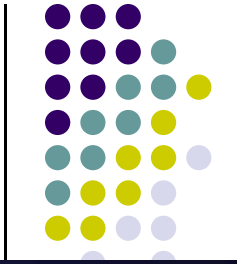


- A target function:

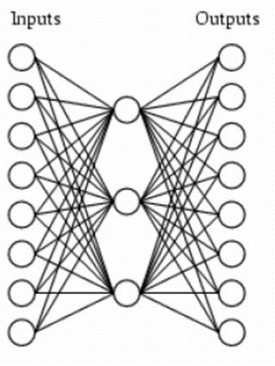
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

- Can this be learned?

Learning Hidden Layer Representation



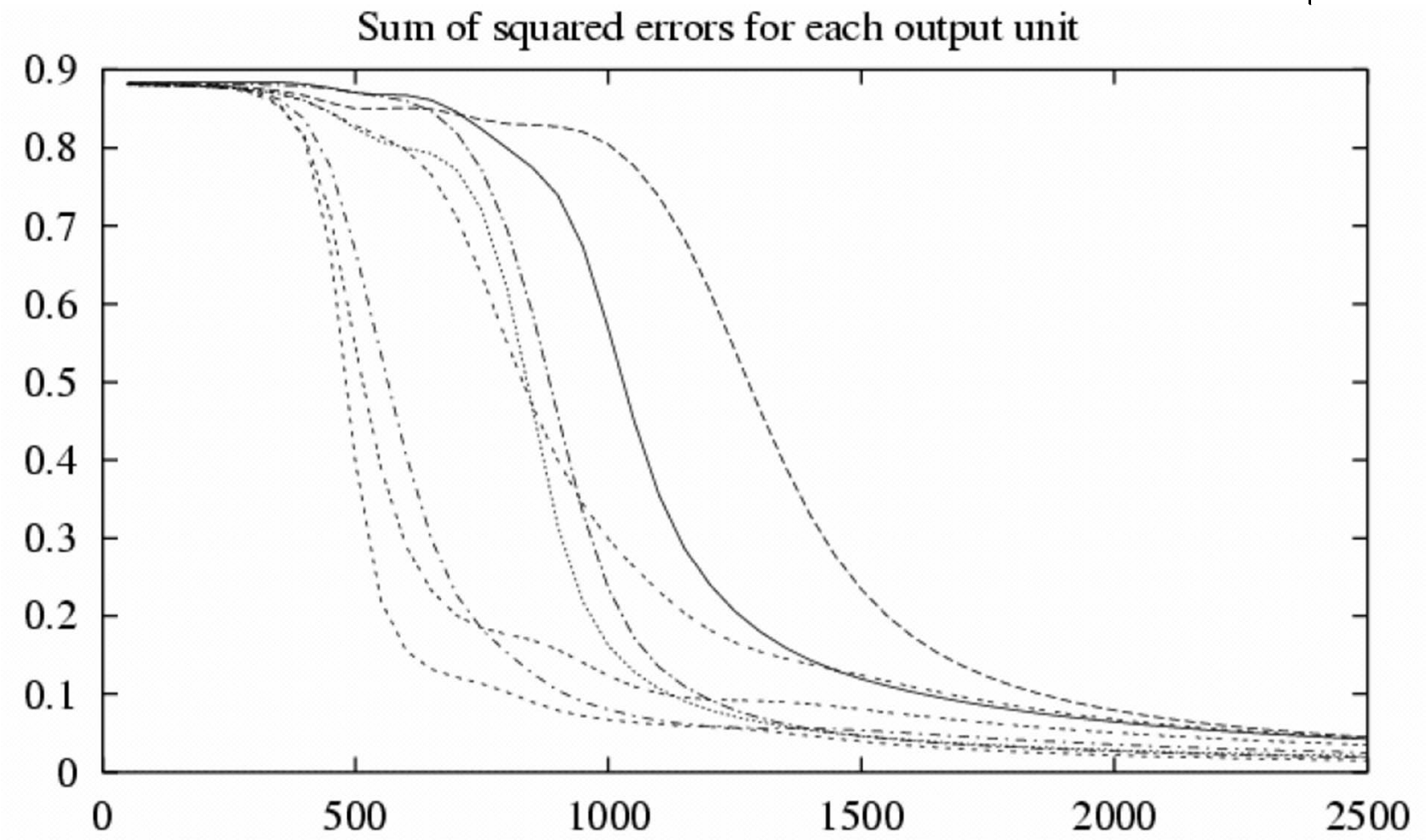
- A network:



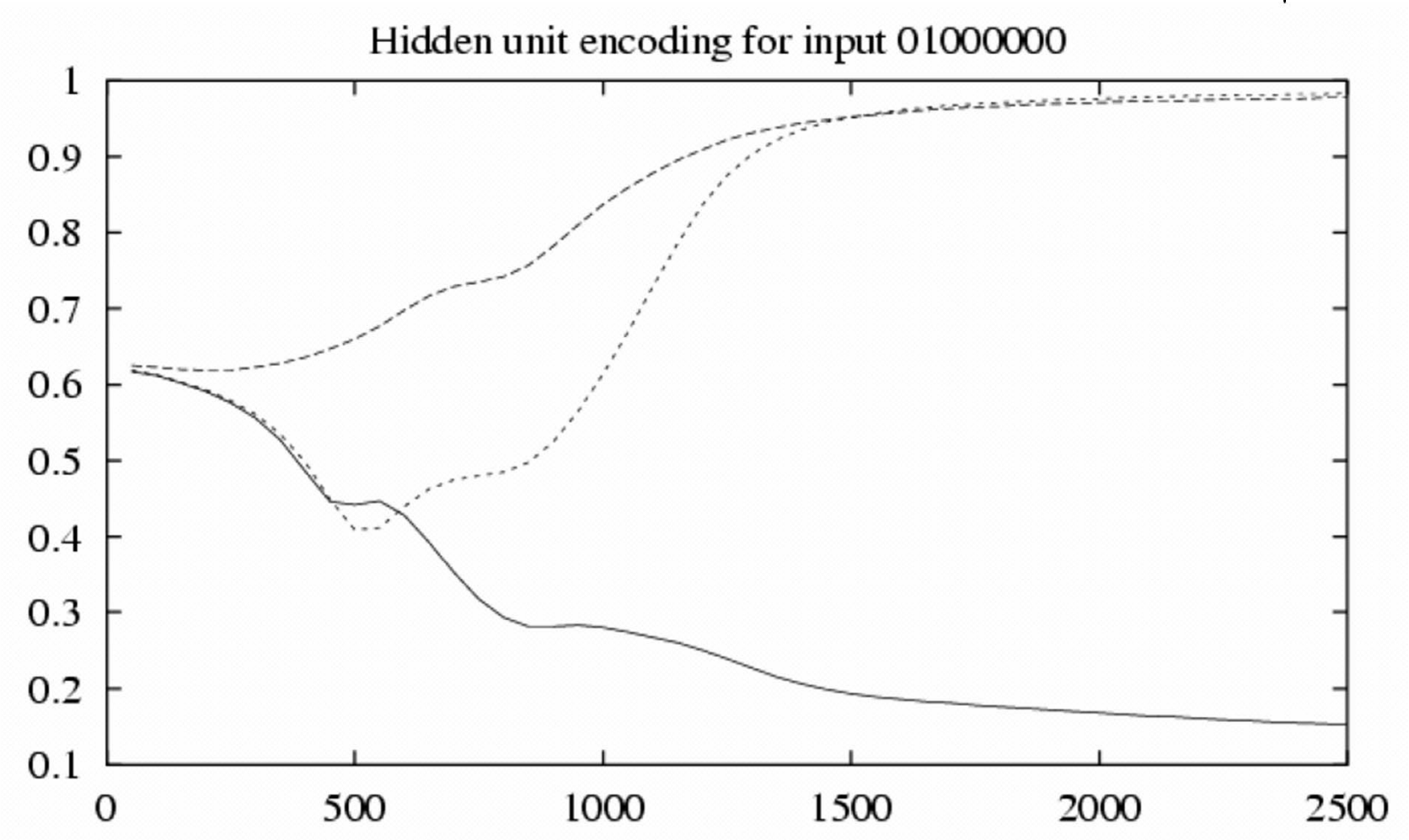
- Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

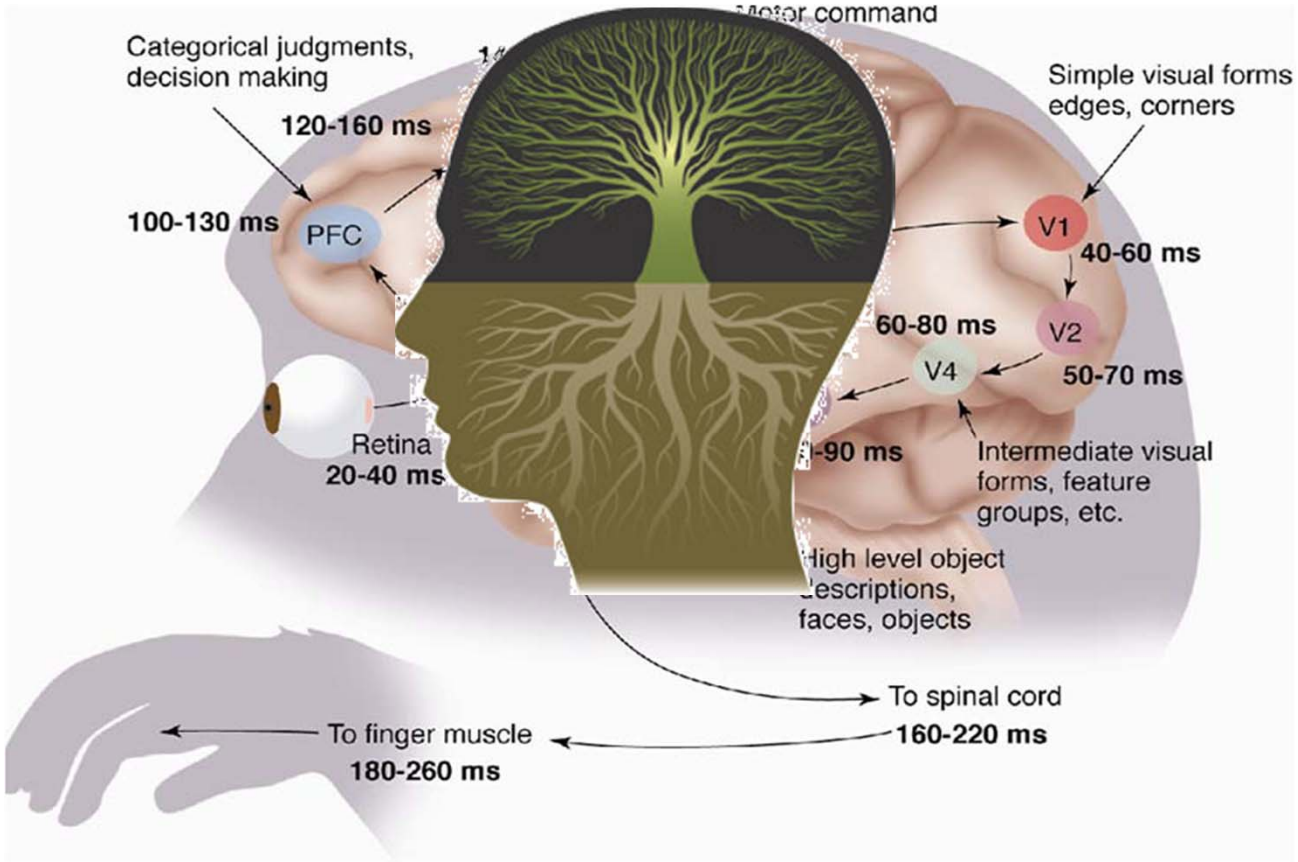
Training



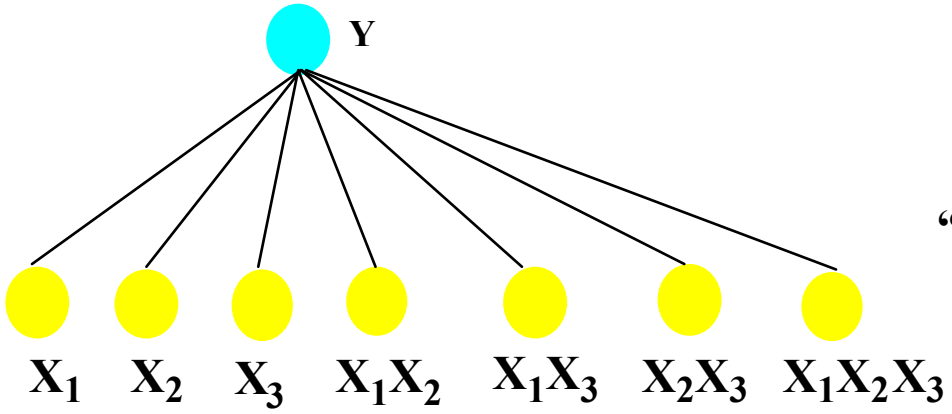
Training

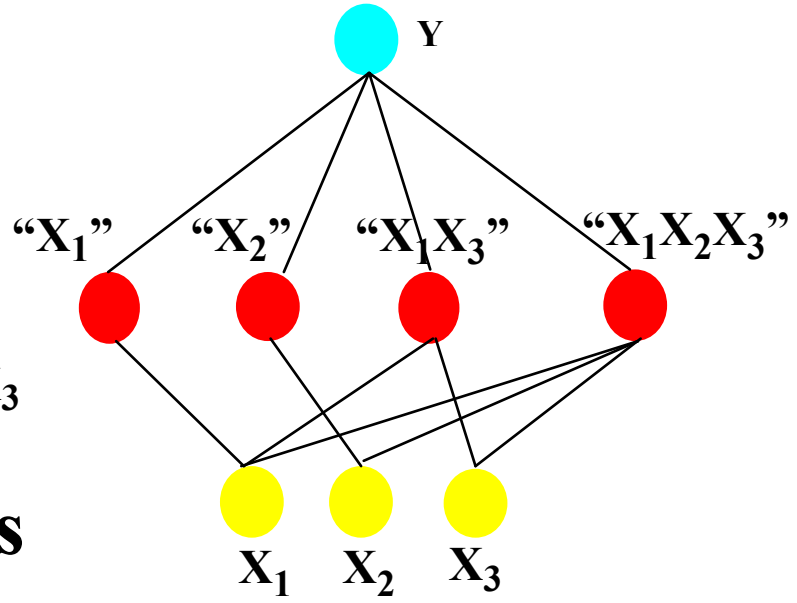


Modern ANN topics: "Deep" Learning



Non-linear LR vs. ANN

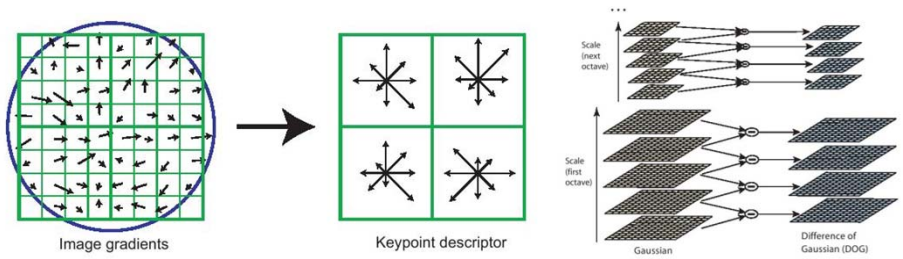




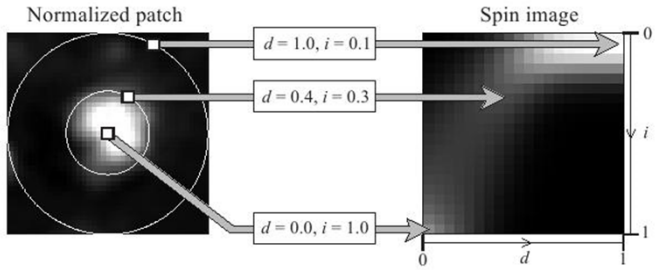
(2^3-1) possible combinations

$Y = a(X_1) + b(X_2) + c(X_3) + d(X_1X_2) + \ldots$

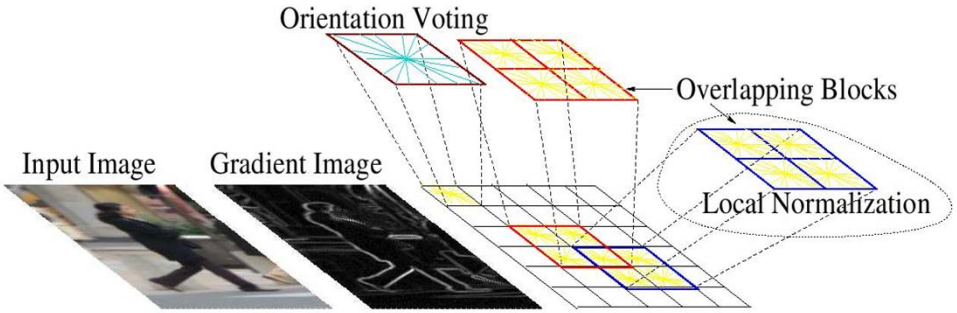
Computer vision features

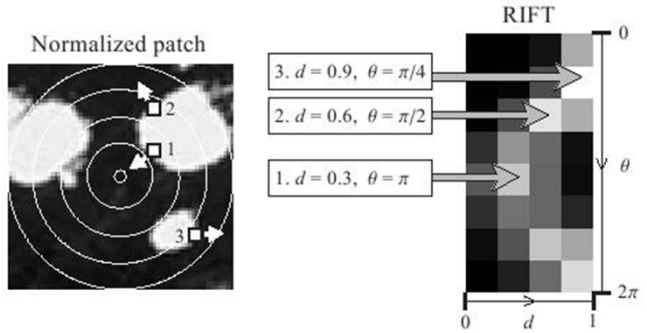


SIFT



Spin image

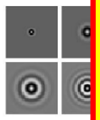


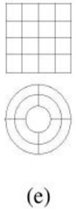


Drawbacks of feature engineering

1. Needs expert knowledge

2. Time consuming hand-tuning



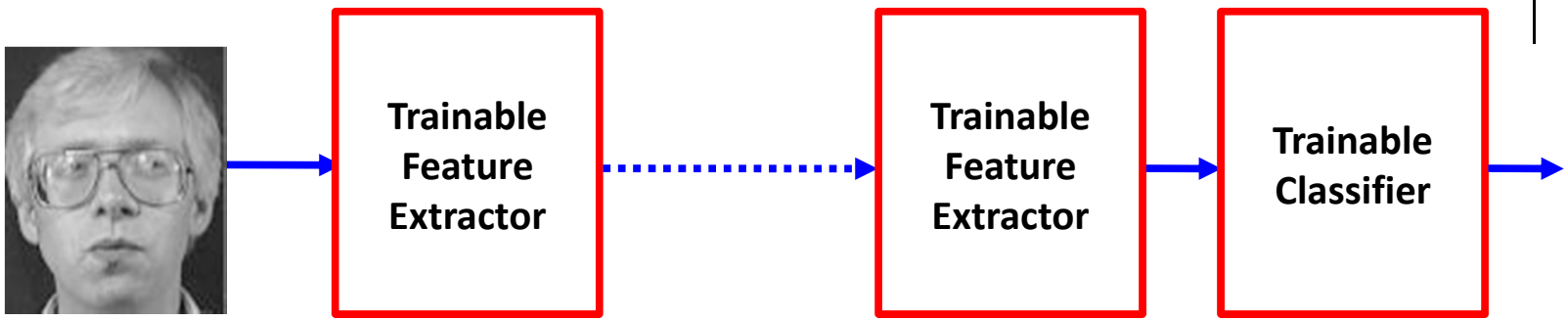


Textons

GLOH

e and Ng

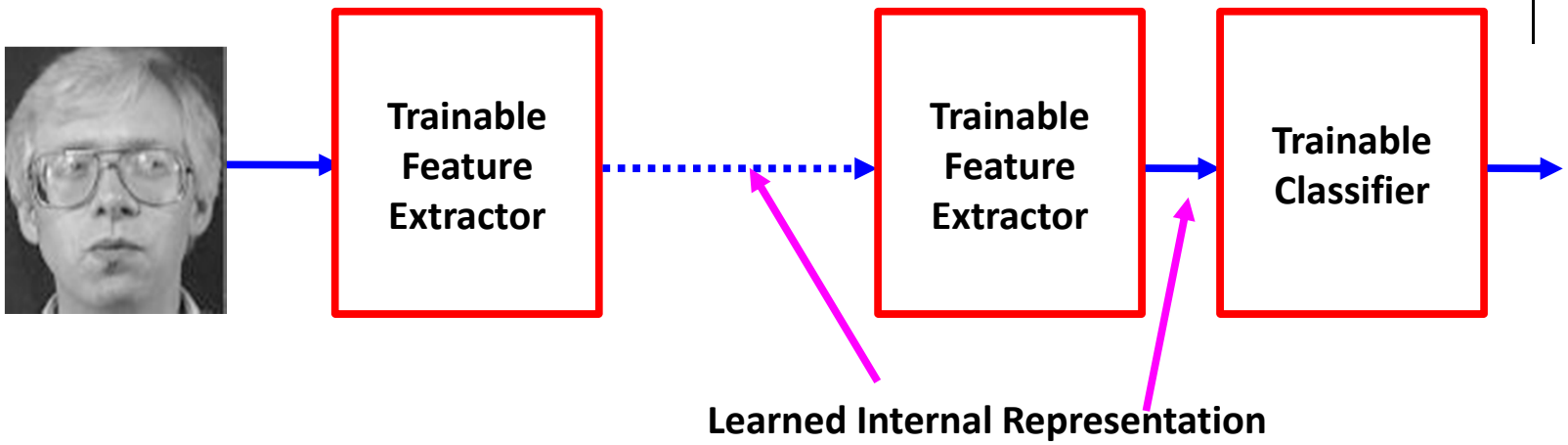
Using ANN to capture hierarchical representation



Good Representations are hierarchical

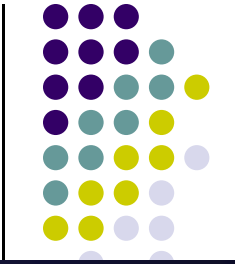
- In Language: hierarchy in syntax and semantics
  - Words->Parts of Speech->Sentences->Text
  - Objects,Actions,Attributes...-> Phrases -> Statements -> Stories
- In Vision: part-whole hierarchy
  - Pixels->Edges->Textons->Parts->Objects->Scenes

“Deep” learning: learning hierarchical representations



- Deep Learning: learning a hierarchy of internal representations
- From low-level features to mid-level invariant representations, to object identities
- Representations are increasingly invariant as we go up the layers
- using multiple stages gets around the specificity/invariance dilemma

“Deep” models



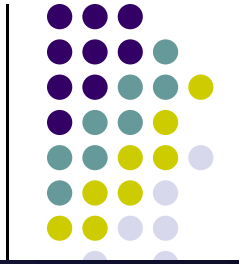
- Neural Networks: Feed-forward*
  - You have seen it
- “Deep” Restrictive Boltzmann Machines (RBM)
  - Probabilistic – Undirected: MRFs and RBMs*
- Autoencoders (multilayer NN with output = input)
  - Non-Probabilistic -Directed: PCA, Sparse Coding

Different loss function designs

- Recurrent Neural Networks*
- Convolutional Neural Nets

Different architecture designs

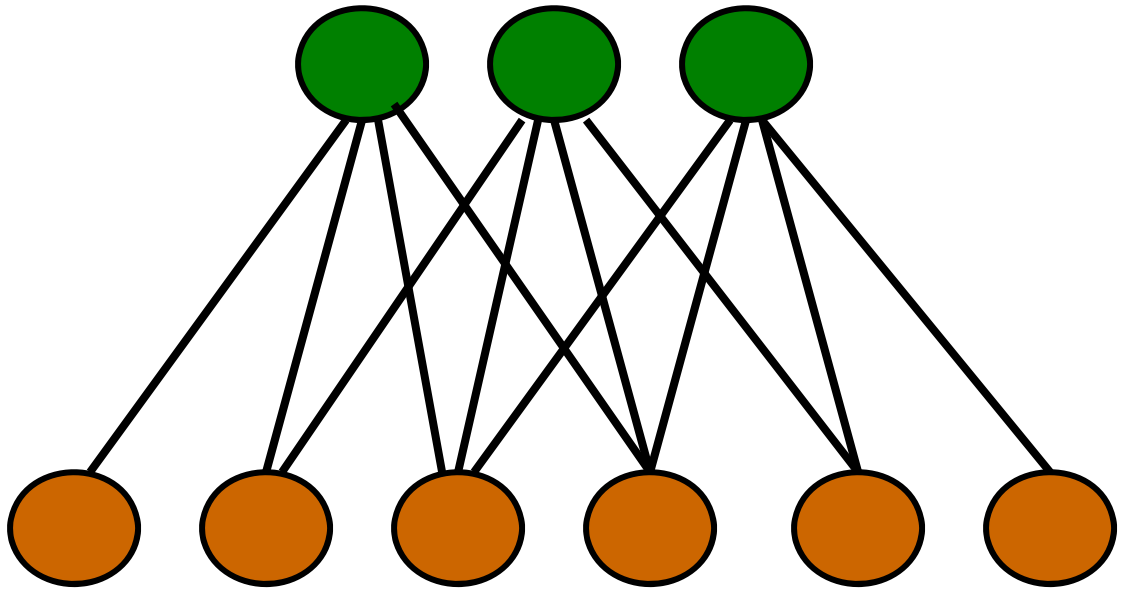
Example I: The Restrictive Boltzmann Machines, aka., the "Harmonium"



The Harmonium (Smolensky –'86)

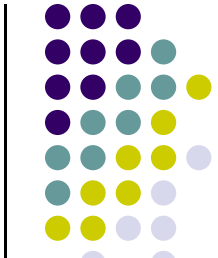
hidden units

visible units



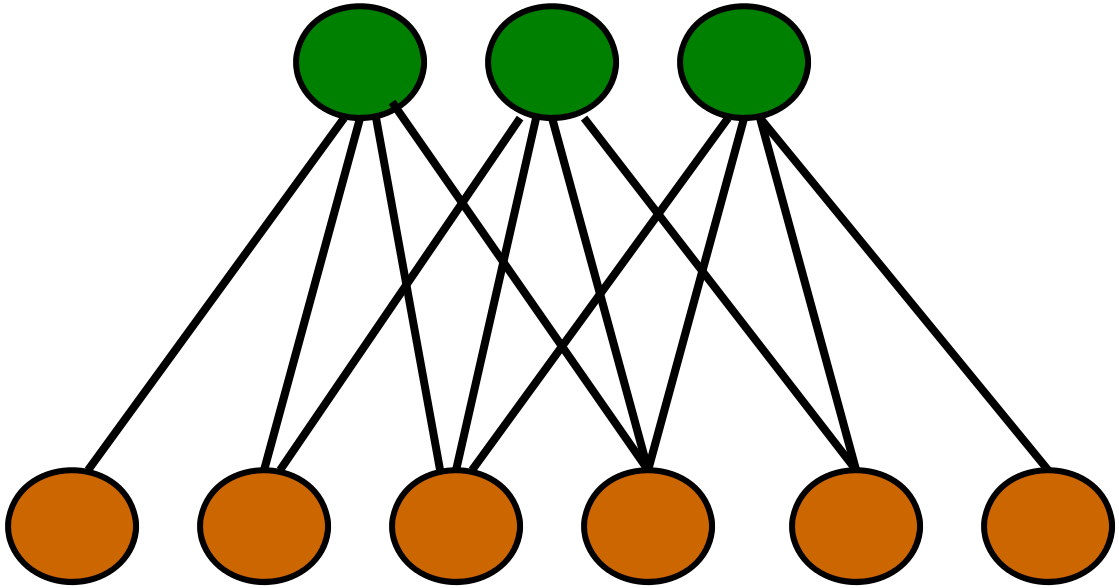
History:
Smolensky ('86), Proposed the architechture.
Freund & Haussler ('92), The "Combination Machine" (binary), learning with projection pursuit.
Hinton ('02), The "Restricted Boltzman Machine" (binary), learning with contrastive divergence.
Marks & Movellan ('02), Diffusion Networks (Gaussian).
Welling, Hinton, Osindero ('02), "Product of Student-T Distributions" (super-Gaussian)

A Two-layer MRFs



hidden units

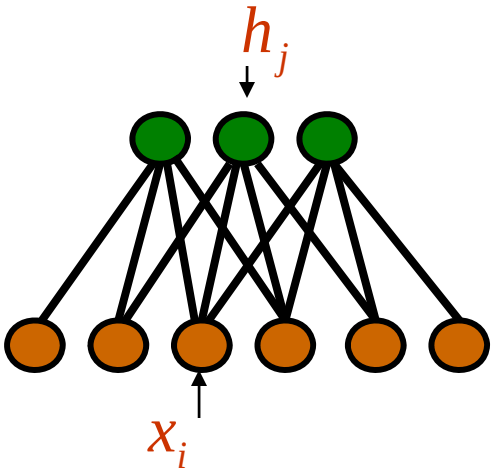
visible units



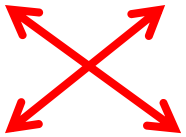
Boltzmann machines:

$$p(x,h\,|\,\theta) = \exp\Big\{ \sum_i \theta_i \phi_i(x_i) + \sum_j \theta_j \phi_j(h_j) + \sum_{i,j} \theta_{i,j} \phi_{i,j}(x_i,h_j) - A(\theta) \Big\}$$

A Constructive Definition



$$p_{\text{ind}}(\mathbf{h}) \propto \prod_j \exp\{\ \theta_j g_j(h_j)\ \}$$

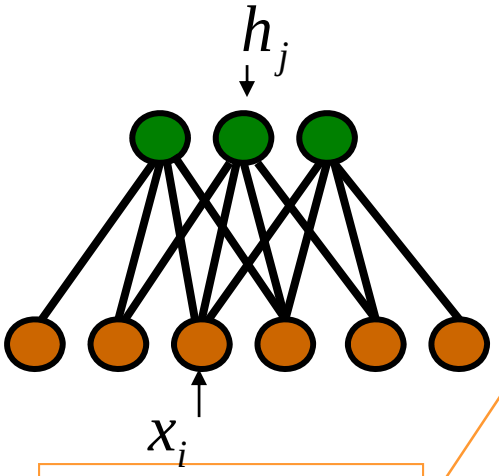


how do we couple them?

$$p_{\text{ind}}(\mathbf{x}) \propto \prod_i \exp\{\ \theta_i f_i(x_i)\ \}$$

$$p(x,h\,|\,\theta) = \exp\{\ \sum_i \vec{\theta}_i \vec{f}_i(x_i) + \sum_j \vec{\lambda}_j \vec{g}_j(h_j) + \sum_{i,j} \vec{f}_i^{\,T}(x_i)\mathbf{W}_{i,j}\vec{g}_j(h_j)\ \}$$

A Constructive Definition



coupling in the log-domain with **shifted parameters**

$$p(\mathbf{x}\mid\mathbf{h}) = \prod_i p(x_i\mid\mathbf{h}),$$

$$p(x_i\mid\mathbf{h}) = \exp\Big\{ \sum_a \hat{\theta}_{ia} f_{ia}(x_i) + A_i(\{\hat{\theta}_{ia}\}) \Big\}$$

$$\hat{\theta}_{ia} = \theta_{ia} + \sum_{jb} W_{ia}^{jb} g_{jb}(h_j) = \theta_{ia} + \sum_j \vec{W}_{ia}^{j} \vec{g}_j(h_j)$$

$$p(\mathbf{h}\mid\mathbf{x}) = \prod_j p(h_j\mid\mathbf{x})$$

$$p(h_j\mid\mathbf{x}) = \exp\Big\{ \sum_b \hat{\lambda}_{jb} g_{jb}(h_j) + B_j(\{\hat{\lambda}_{jb}\}) \Big\}$$

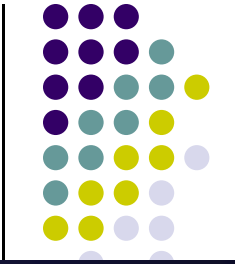
$$\hat{\lambda}_{jb} = \lambda_{jb} + \sum_{ia} W_{ia}^{jb} f_{ia}(x_i) = \lambda_{jb} + \sum_i \vec{W}_i^{jb} \vec{f}_i(x_i)$$

vector of local sufficient statistics (features)

They map to the harmonium random field:

$$p(x,h\mid\theta) = \exp\Big\{ \sum_i \vec{\theta}_i \vec{f}_i(x_i) + \sum_j \vec{\lambda}_j \vec{g}_j(h_j) + \sum_{i,j} \vec{f}_i^{T}(x_i)\mathbf{W}_{i,j}\vec{g}_j(h_j) \Big\}$$

The Computational Trade-off



Undirected model: Learning is hard, inference is easy.

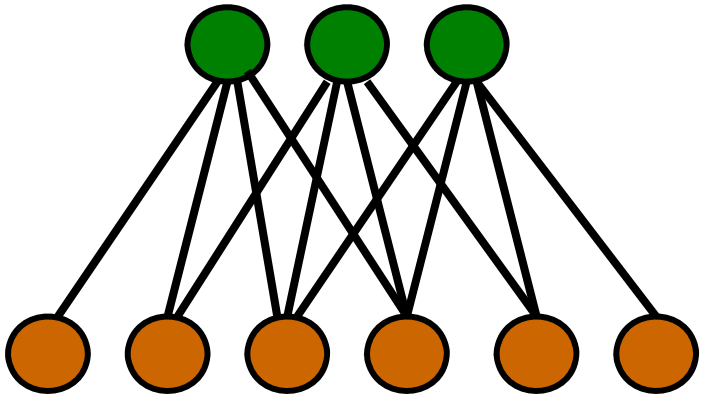
Directed Model: Learning is "easier", inference is hard.

Example: Document Retrieval.

Retrieval is based on comparing (posterior) topic distributions of documents.
- directed models: inference is slow. Learning is relatively “easy”.
- undirected model: inference is fast. Learning is slow but can be done offline.

An RMB for text/image

topics



words counts

EFH or DWH (welling et al 2005, Xing et al 2005):

$h_j = 3$: *topic j has strength 3*

$h_j \in \mathbf{R}, \qquad \langle h_j \rangle = \Sigma_i \vec{W}_{i,j} \bullet \vec{x}_i$

$x_{i,n} = 1$: *word i has count n*

$\vec{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,N_{\max}}], \; x_{i,n} \in \{0,1\}, \; \Sigma_n x_{i,n} = 1$

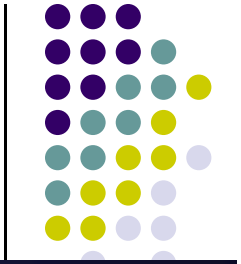
$$p(\mathbf{h} \mid \mathbf{x}) = \prod_j \text{Normal}_{h_j}\left[\sum_i \vec{W}_{ij} \vec{x}_i, 1 \right]$$

$$p(\mathbf{x} \mid \mathbf{h}) = \prod_i \text{Softmax}_{\vec{x}_i}\left[\vec{\alpha}_i + \sum_j \vec{W}_{ij} h_j \right]$$

$\text{Softmax}_{\vec{x}_i}[\vec{\alpha}_i + \Sigma_j \vec{W}_{ij} h_j] \propto \exp\{(\vec{\alpha}_i + \Sigma_j \vec{W}_{ij} h_j)^T \vec{x}_i\}$, note parameterization cost! ($\vec{W}_{ij} = [w_{i,j}^1, \ldots, w_{i,j}^{N_{\max}}], \forall i, j$

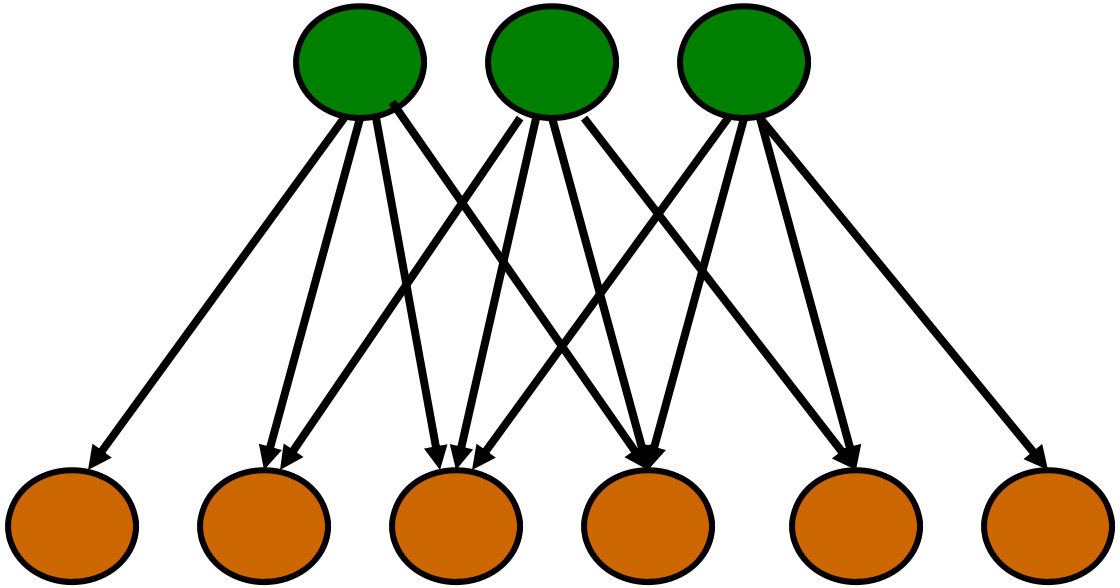
In practice, only 1-0 counting is used!!!

Recall
Properties of Directed Networks



hidden units

visible units



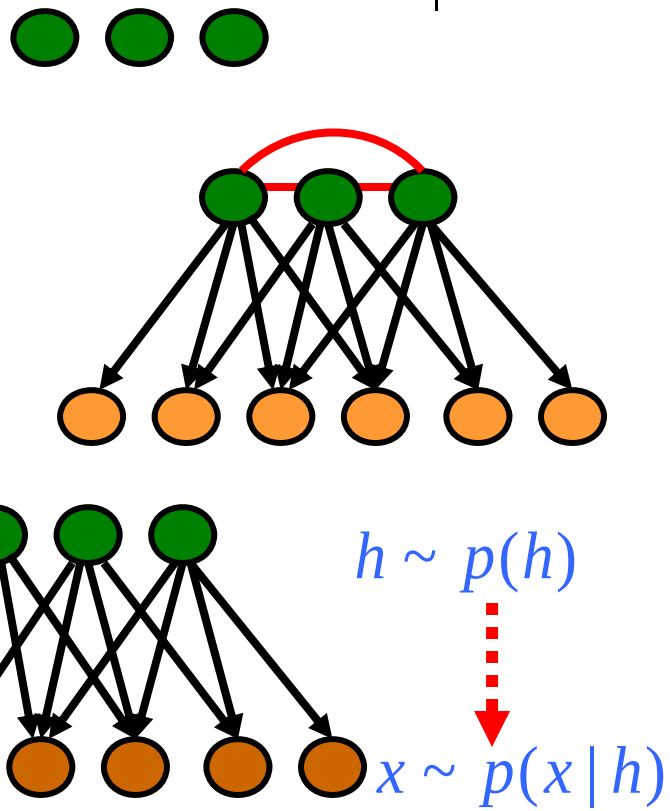
- Semantic naturalness:
  - intuitive causal structural, easy to design, comprehend and manipulate
- Bayesian extensions:
  - straightforward to set up, conjugate ..., but hyper-para fitting is non-trivial
- Computational properties:

Properties of Directed Networks

- Factors are marginally *independent.*
- Factors are conditionally *dependent* given observations on the visible nodes.

$$P(\ell \mid \mathbf{w}) = \frac{P(\mathbf{w} \mid \ell) P(\ell)}{P(\mathbf{w})}$$

- Easy ancestral sampling.
- Learning with (variational) EM

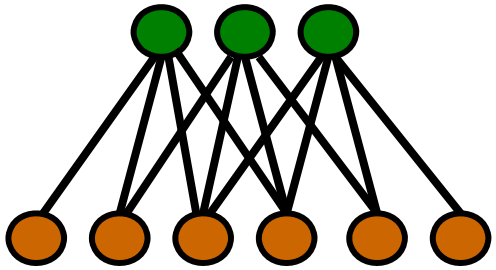


Properties of RBMs

- Factors are marginally *dependent*.
- Factors are conditionally *independent* given observations on the visible nodes.

$$P(\ell \mid \mathbf{w}) = \prod_i P(\ell_i \mid \mathbf{w})$$

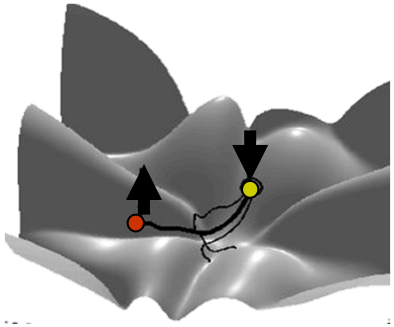
- Iterative Gibbs sampling.



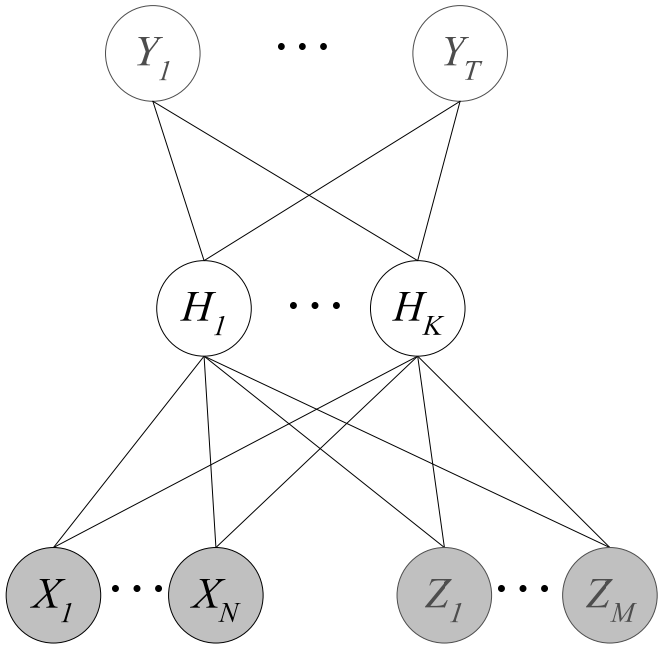
$$h \sim p(h \mid x)$$

$$x \sim p(x \mid h)$$

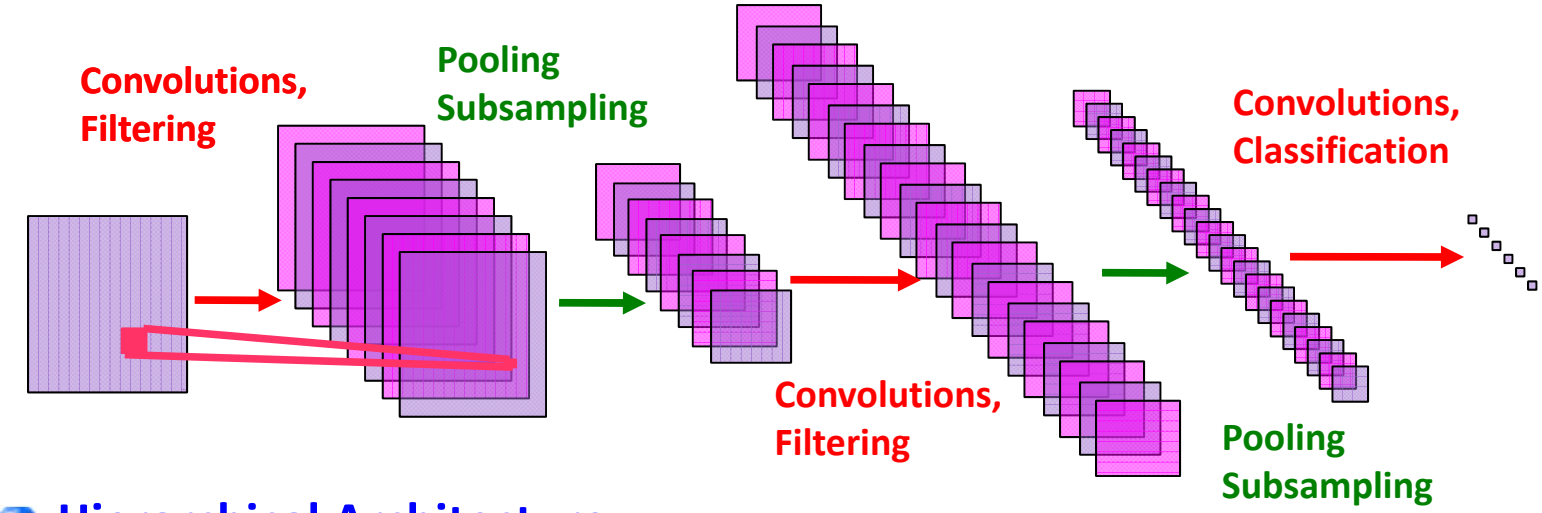
- Learning with contrastive divergence



Deep RBMs

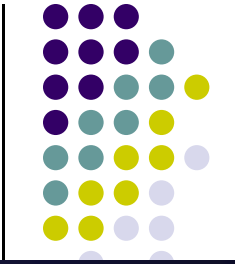


Example II: Convolutional Networks

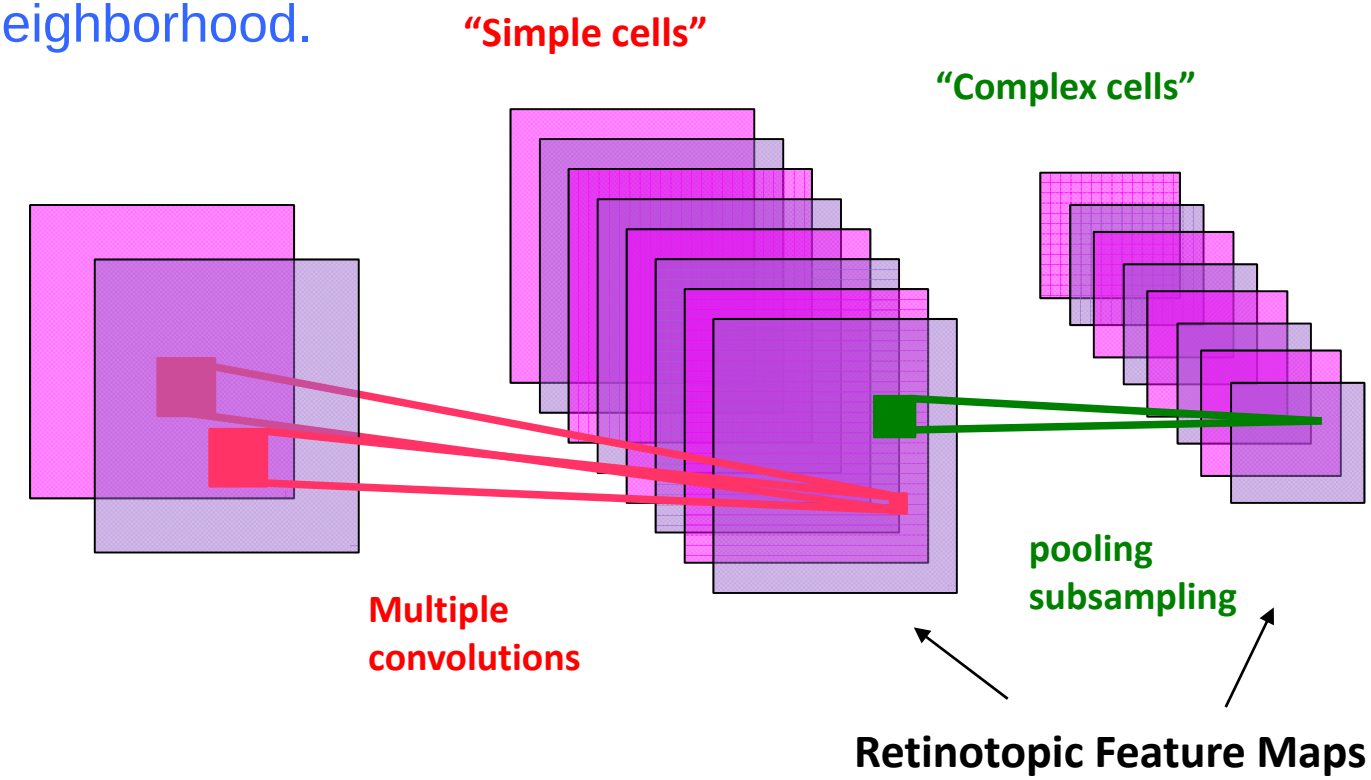


- **Hierarchical Architecture**
  - Representations are more global, more invariant, and more abstract as we go up the layers
- **Alternated Layers of Filtering and Spatial Pooling**
  - Filtering detects conjunctions of features
  - Pooling computes local disjunctions of features
- **Fully Trainable**
  - All the layers are trainable

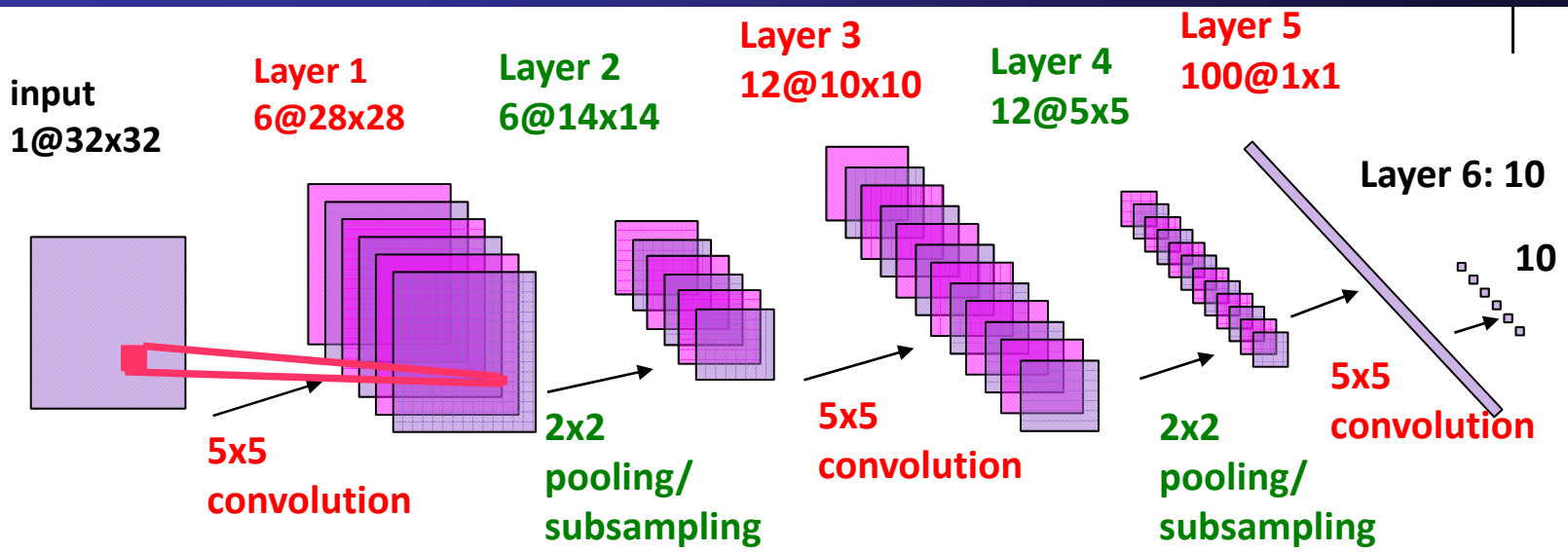
Filtering + NonLinearity + Pooling = 1 stage of a Convolutional Net



- [Hubel & Wiesel 1962]:
  - simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.



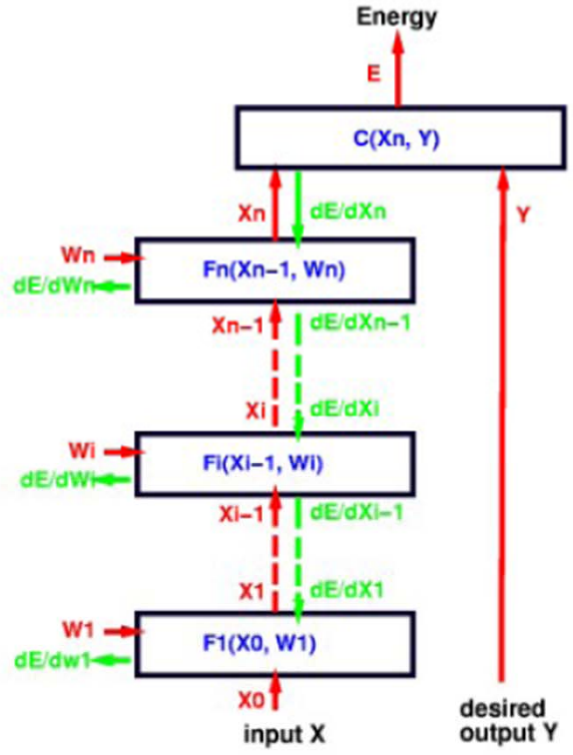
Convolutional Net Architecture for Hand-writing recognition



- Convolutional net for handwriting recognition (400,000 synapses)
  - Convolutional layers (simple cells): all units in a feature plane share the same weights
  - Pooling/subsampling layers (complex cells): for invariance to small distortions.
  - Supervised gradient-descent learning using back-propagation
  - The entire network is trained end-to-end. All the layers are trained simultaneously.
  - [LeCun et al. Proc IEEE, 1998]

How to train?

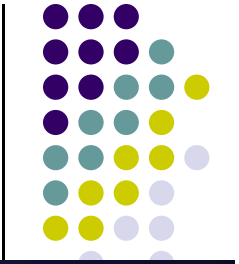
To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$



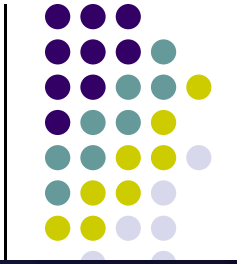
- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

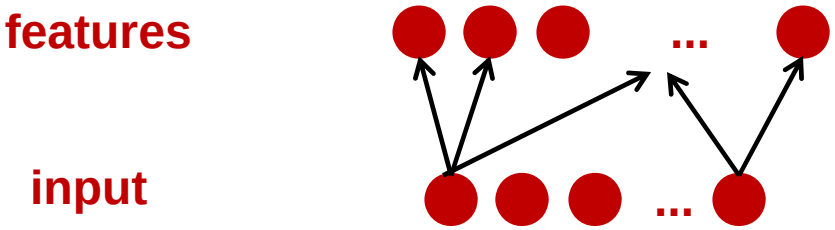
But this is very slow !!!

Layer-wise Unsupervised Pre-training

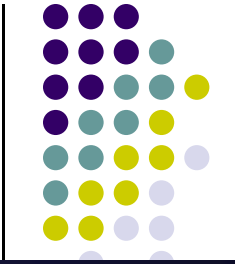


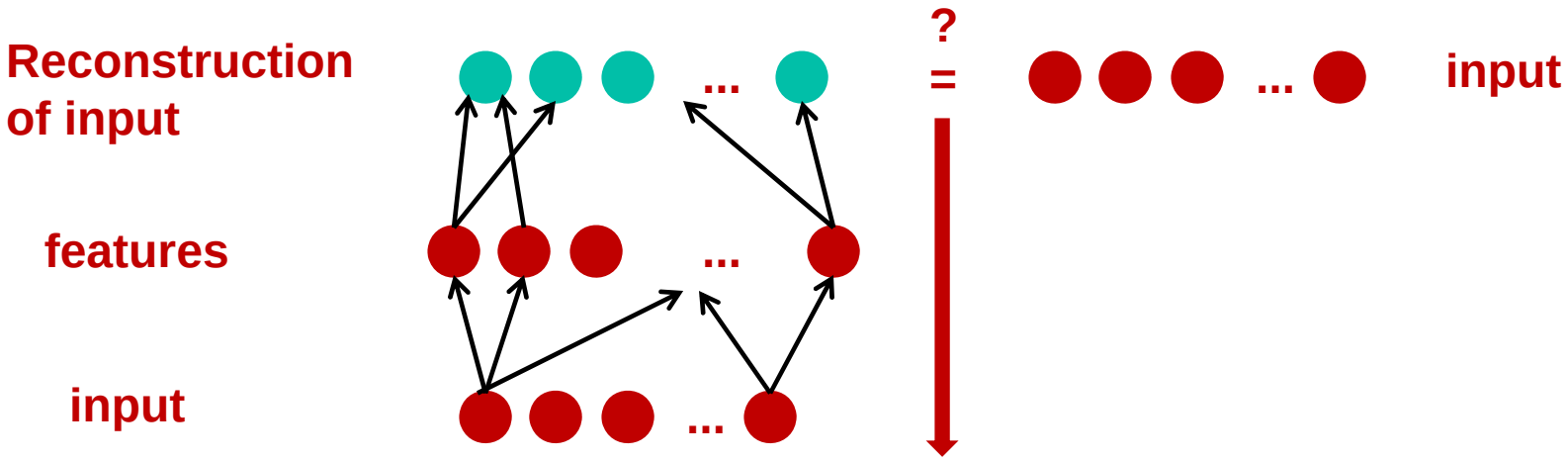
Layer-wise Unsupervised Pre-training



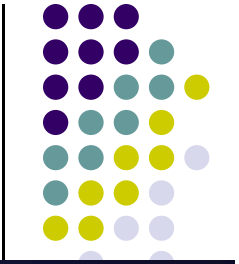


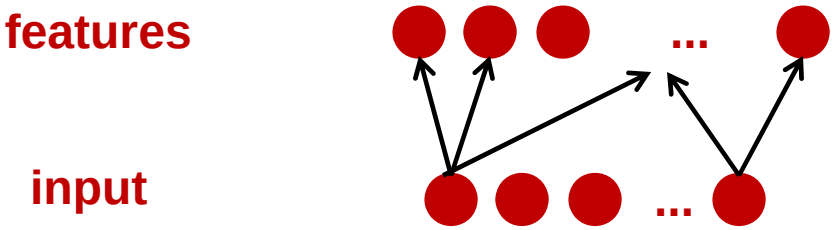
Layer-wise Unsupervised Pre-training



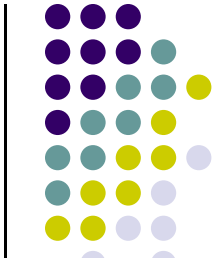


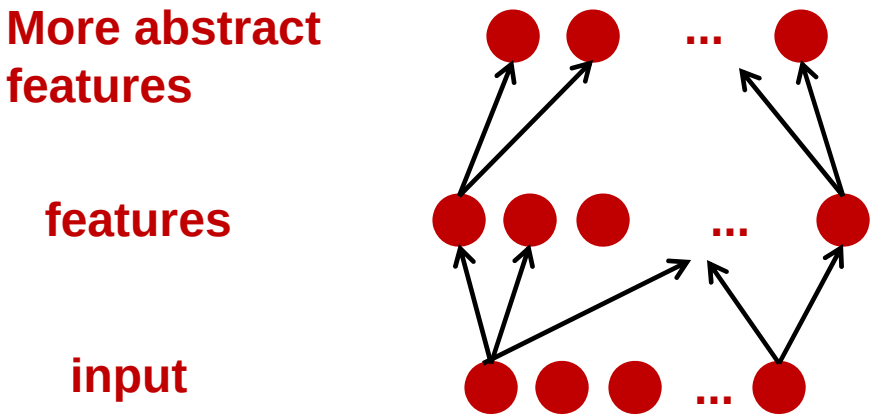
Layer-wise Unsupervised Pre-training



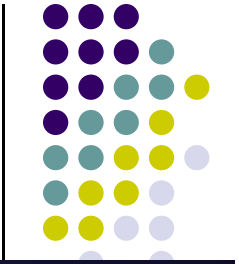


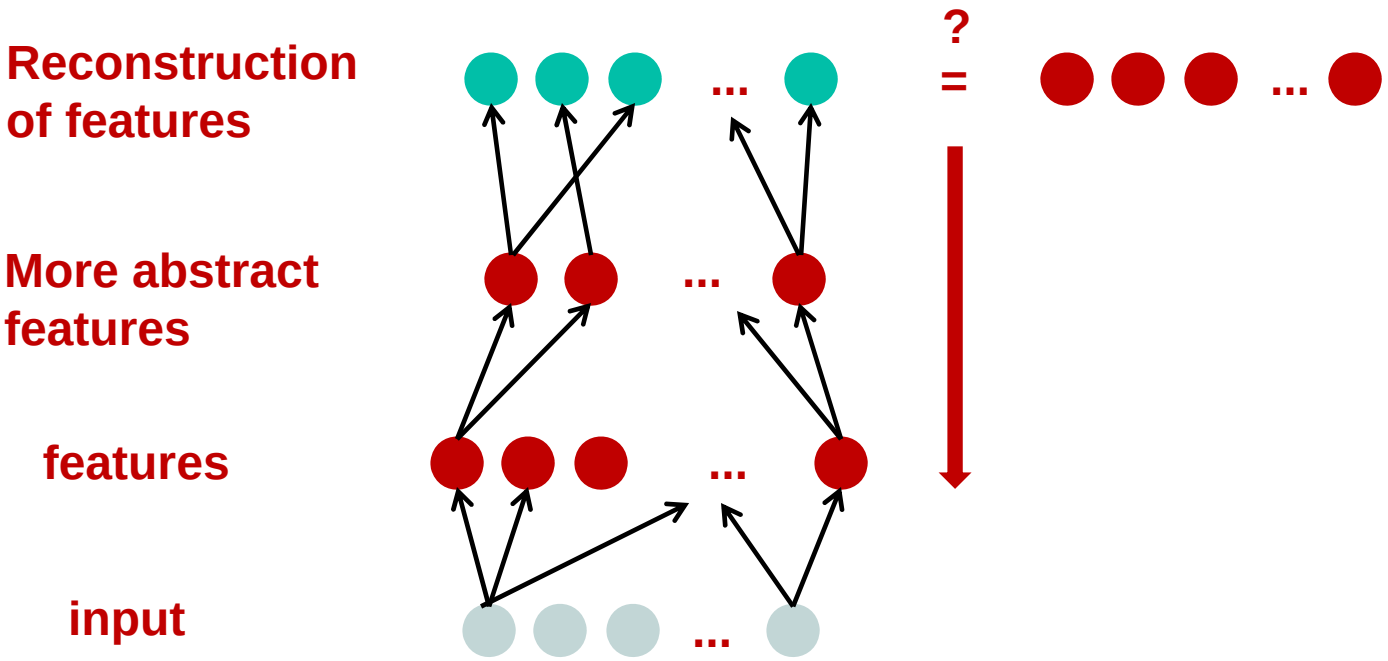
Layer-wise Unsupervised Pre-training



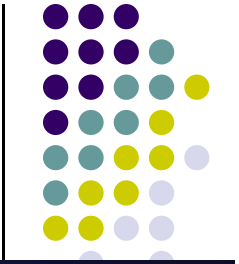


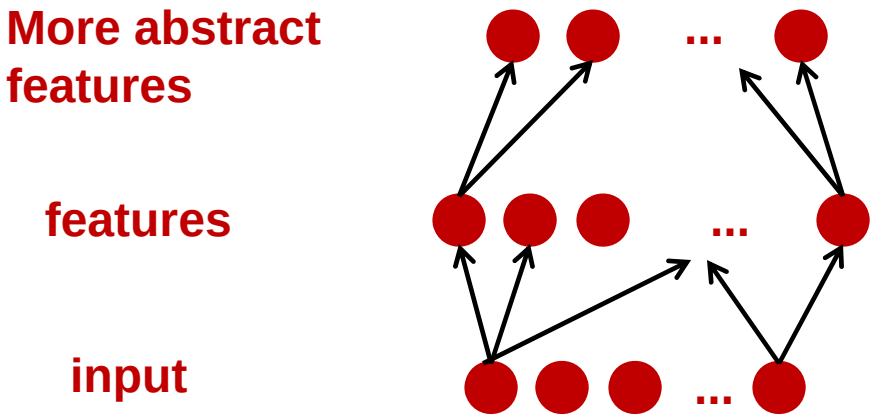
Layer-wise Unsupervised Pre-training



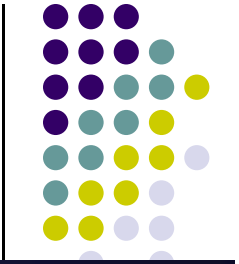


Layer-wise Unsupervised Pre-training





Layer-wise Unsupervised Pre-training

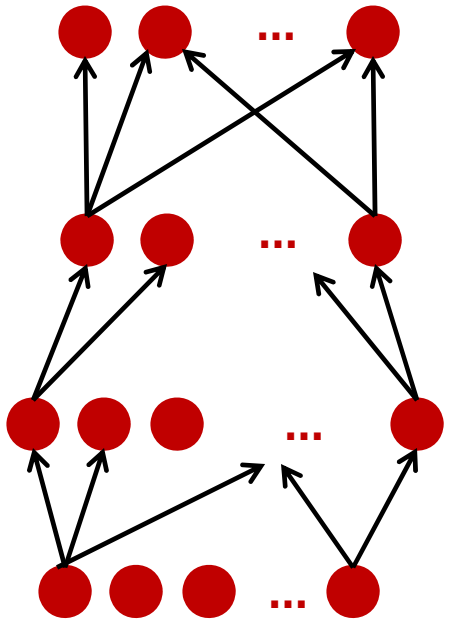


Even more abstract features

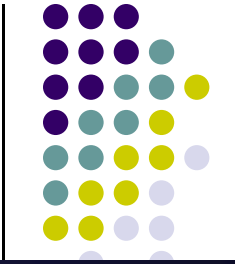
More abstract features

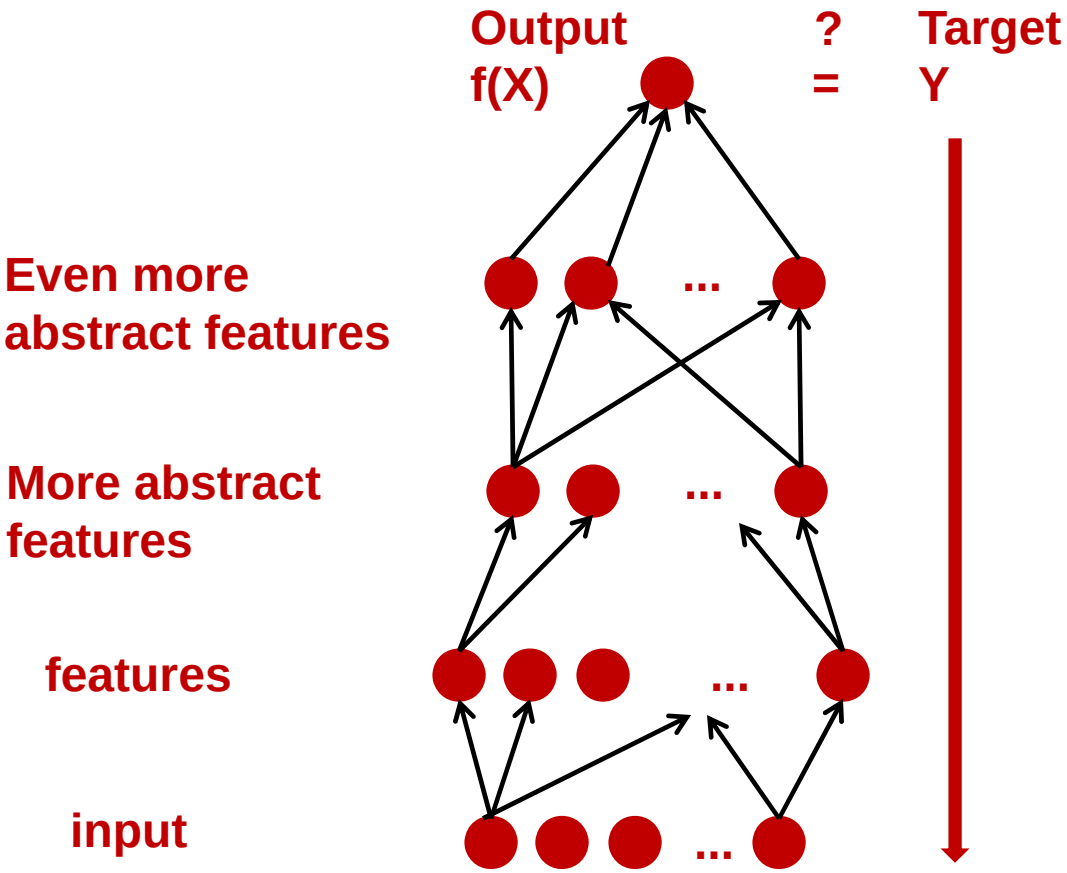
features

input



Layer-wise Unsupervised Pre-training

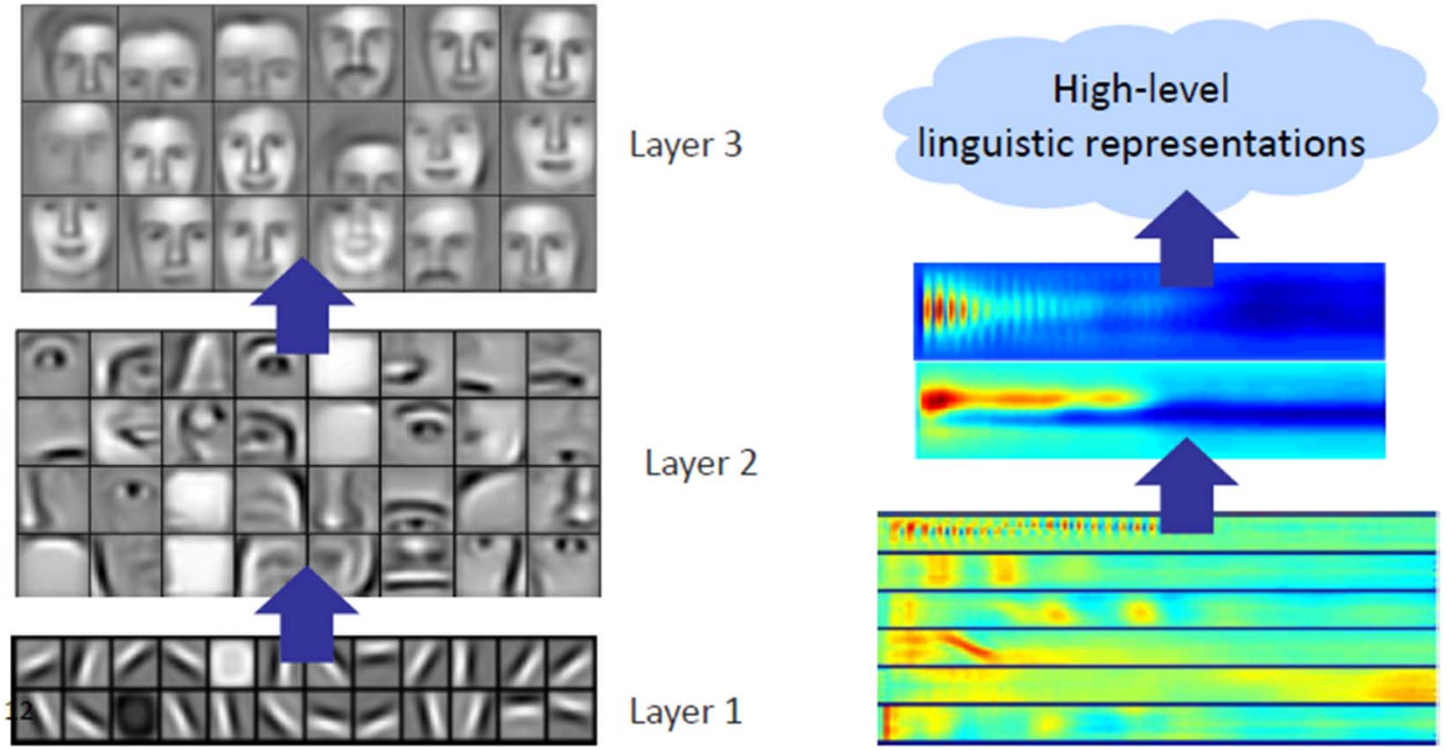




Feature learning

- Successful learning of intermediate representations

  [Lee et al ICML 2009, Lee et al NIPS 2009]

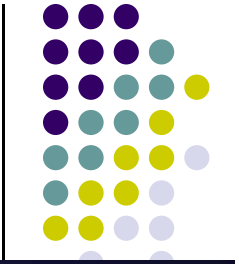


Deep Learning is Amazing!!!

- Handwriting recognition MNIST (many), Arabic HWX (IDSIA)
- OCR in the Wild [2011]: StreetView House Numbers (NYU and others)
- Traffic sign recognition [2011] GTSRB competition (IDSIA, NYU)
- Pedestrian Detection [2013]: INRIA datasets and others (NYU)
- Volumetric brain image segmentation [2009] connectomics (IDSIA, MIT)
- Human Action recognition [2011] Hollywood II dataset (Stanford)
- Object Recognition [2012] ImageNet competition
- Scene Parsing [2012] Stanford background, SiftFlow, Barcelona (NYU)
- Scene parsing from depth images [2013] NYU RGB-D dataset (NYU)
- Speech Recognition [2012] Acoustic modeling (IBM and Google)
- Breast cancer cell mitosis detection [2011] MITOS (IDSIA)

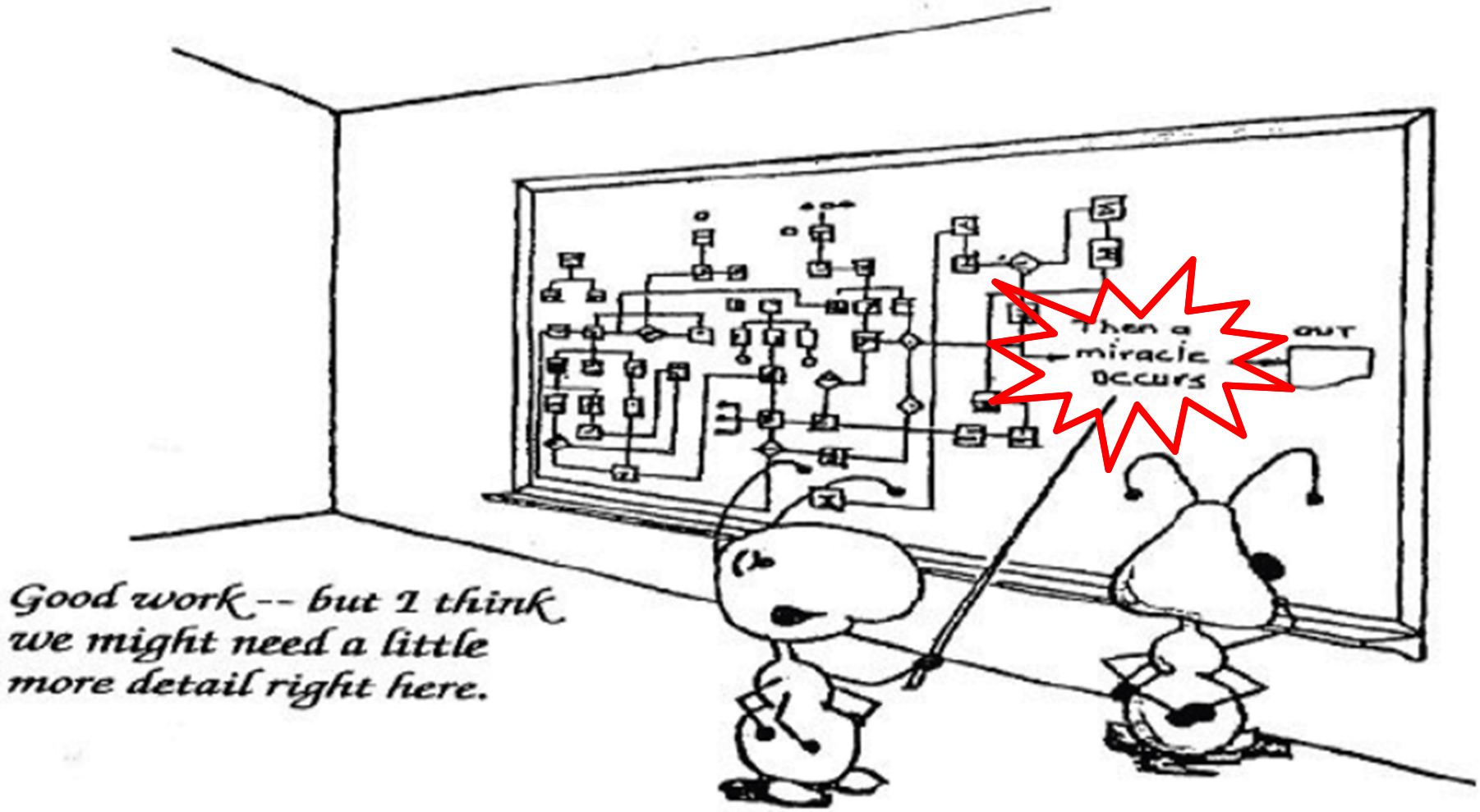
Wow! ☺

- The list of perceptual tasks for which ConvNets hold the record is growing.
- Most of these tasks (but not all) use purely supervised convnets.



A Few Thoughts on How We May Want to Further Study DNN

What makes it work? Why?



An MLer's View of the World

Loss functions
(likelihood, reconstruction, margin, …)

Structures
(Graphical, group, chain, tree, iid, …)

Constraints
(normality, sparsity, label, prior, KL, sum, …)

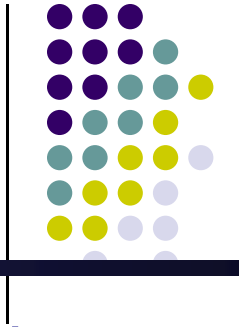
Algorithms
MC (MCMC, Importance), Opt (gradient, IP), …

Stopping criteria
Change in objective, change in update …

Empirical Performances?

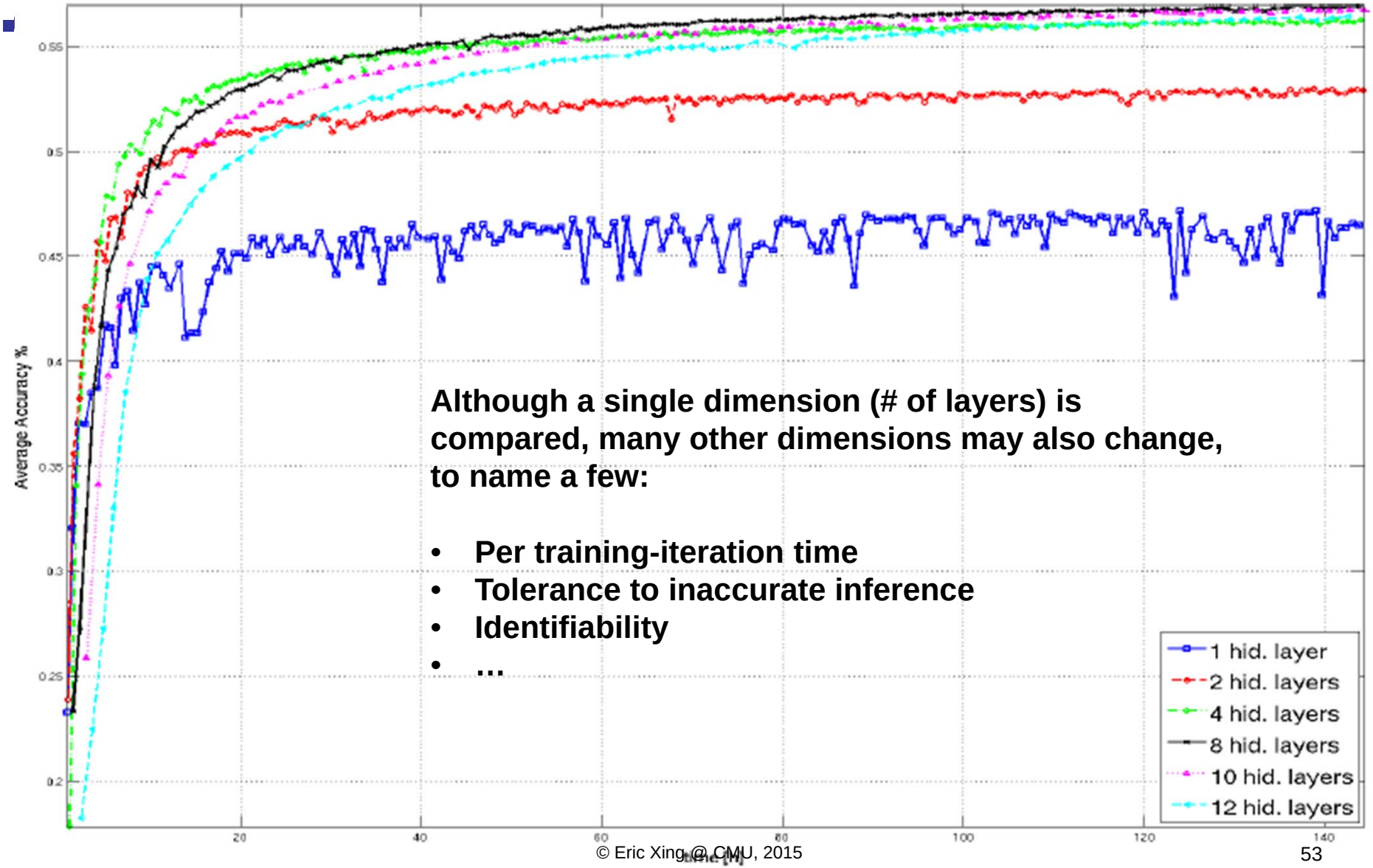
	DL	vs ?	ML (e.g., GM)
Empirical goal:	e.g., classification, feature learning		e.g., transfer learning, latent variable inference
Structure:	Graphical		Graphical
Objective:	Something aggregated from local functions		Something aggregated from local functions
Vocabulary:	Neuron, activation/gate function …		Variables, potential function
Algorithm:	A single, unchallenged, inference algorithm -- BP		A major focus of open research, many algorithms, and more to come
Evaluation:	On a black-box score -- end performance		On almost every intermediate quantity
Implementation:	Many untold-tricks		More or less standardized
Experiments:	Massive, real data (GT unknown)		Modest, often simulated data (GT known)

A slippery slope to mythology?



- How to conclusively determine what an improve in performance could come from:
  - Better model (architecture, activation, loss, size)?
  - Better algorithm (more accurate, faster convergence)?
  - Better training data?
- Current research in DL seem to get everything above mixed by evaluating on a black-box "performance score" that is not directly reflecting
  - Correctness of inference
  - Achievability/usefulness of model
  - Variance due to stochasticity

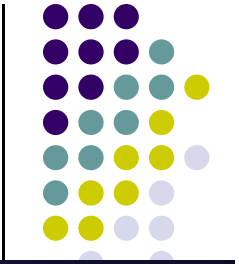
An Example



Although a single dimension (# of layers) is compared, many other dimensions may also change, to name a few:

- **Per training-iteration time**
- **Tolerance to inaccurate inference**
- **Identifiability**
- **…**

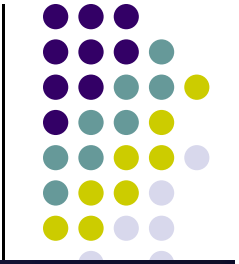
Inference quality



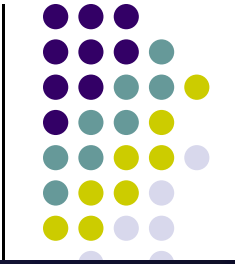
- Training error is the old concept of a classifier with no hidden states, no **inference** is involved, and thus inference accuracy is not an issue
- But a DNN is not just a classifier, some DNNs are not even fully supervised, there are MANY **hidden states**, why their inference quality is not taken seriously?

- In DNN, inference accuracy = visualizing features
  - Study of inference accuracy is badly discouraged
  - Loss/accuracy is not monitored

Inference/Learning Algorithm, and their evaluation



Learning a GM with Hidden Variables – the thought process

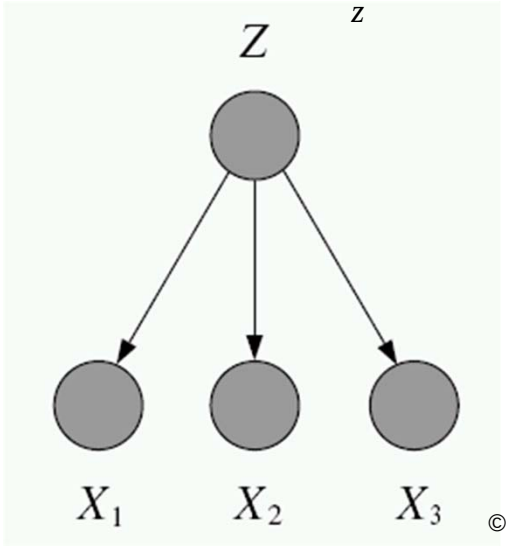


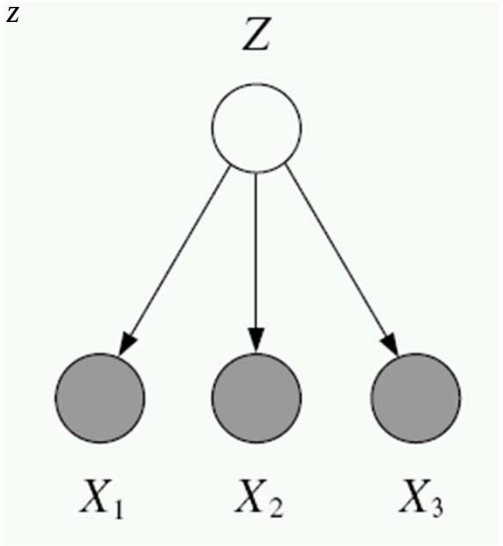
- In fully observed iid settings, the log likelihood decomposes into a sum of local terms (at least for directed models).

$$\ell_c(\theta; D) = \log p(x, z \mid \theta) = \log p(z \mid \theta_z) + \log p(x \mid z, \theta_x)$$

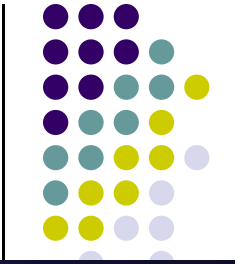
- With latent variables, all the parameters become coupled together via marginalization

$$\ell_c(\theta; D) = \log \sum_z p(x, z \mid \theta) = \log \sum_z p(z \mid \theta_z)\, p(x \mid z, \theta_x)$$





Gradient Learning for mixture models



- We can learn mixture densities using gradient descent on the log likelihood. The gradients are quite interesting:

$$l(\theta) = \log p(\mathrm{x} \mid \theta) = \log \sum_k \pi_k p_k(\mathrm{x} \mid \theta_k)$$

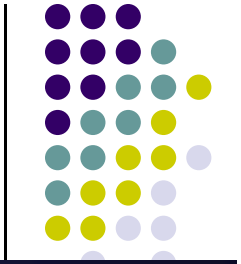
$$\frac{\partial l}{\partial \theta_k} = \frac{1}{p(\mathrm{x} \mid \theta)} \sum_k \pi_k \frac{\partial p_k(\mathrm{x} \mid \theta_k)}{\partial \theta_k}$$

$$= \sum_k \frac{\pi_k}{p(\mathrm{x} \mid \theta)} p_k(\mathrm{x} \mid \theta_k) \frac{\partial \log p_k(\mathrm{x} \mid \theta_k)}{\partial \theta_k}$$

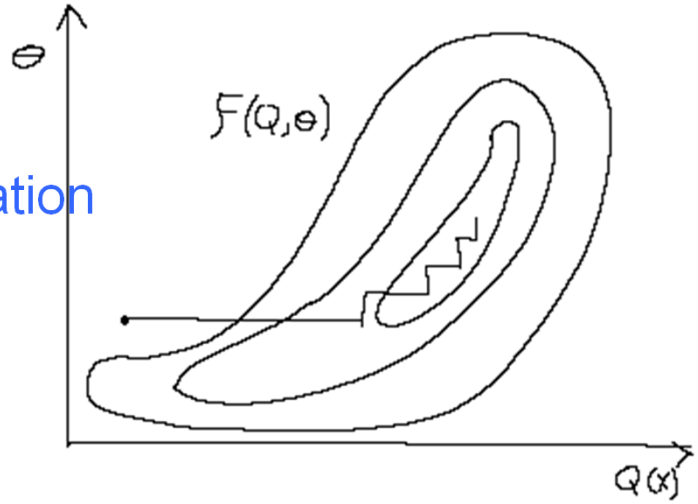
$$= \sum_k \pi_k \frac{p_k(\mathrm{x} \mid \theta_k)}{p(\mathrm{x} \mid \theta)} \frac{\partial \log p_k(\mathrm{x} \mid \theta_k)}{\partial \theta_k} = \sum_k r_k \frac{\partial l_k}{\partial \theta_k}$$

- In other words, the gradient is aggregated from many other intermediate states
  - Implication: costly iteration, heavy coupling between parameters
- Other issues: imposing constraints, identifiability …

Then Alternative Approaches Were Proposed



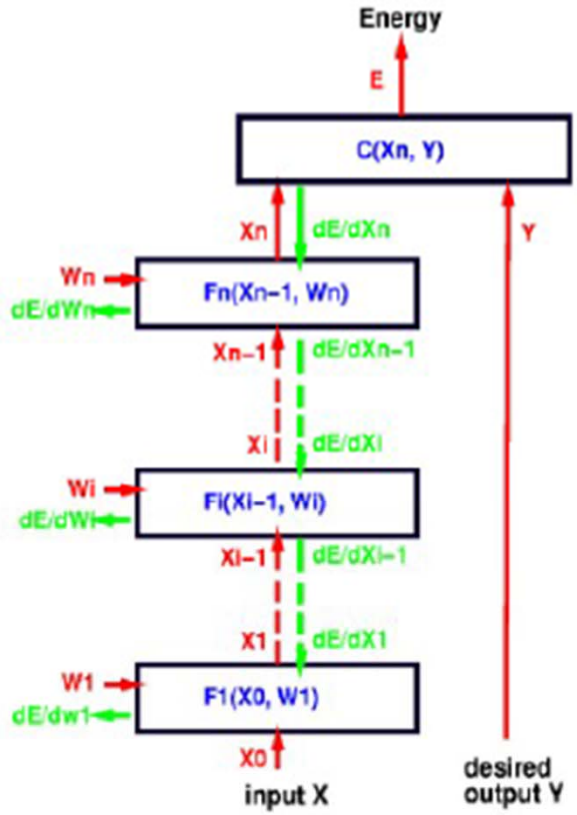
- The EM algorithm
  - M: a convex problem
  - E: approximate constrained optimization
    - Mean field
    - BP/LBP
    - Marginal polytope



- Spectrum algorithm:
  - redefine intermediate states, convexify the original problem

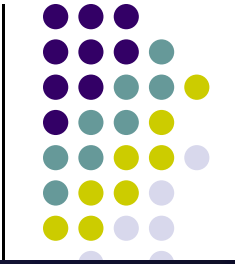
Learning a DNN

To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$



- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

Learning a DNN

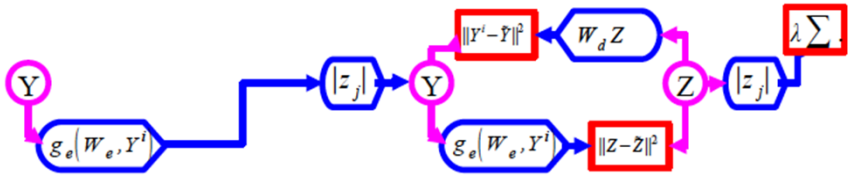


- In a nutshell, sequentially, and recursively apply:

$$w_{j,i}^{t+1} = w_{j,i}^{t} - \eta_t \delta_j z_i$$

$$\delta_i = h'(a_i) \sum_j \delta_j w_{j,i}$$

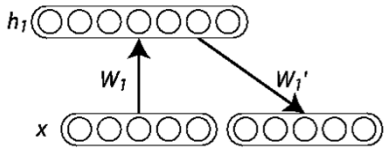
- Things can getting hairy when locally defined losses are introduced, e.g., auto-encoder, which breaks a loss-driven global optimization formulation

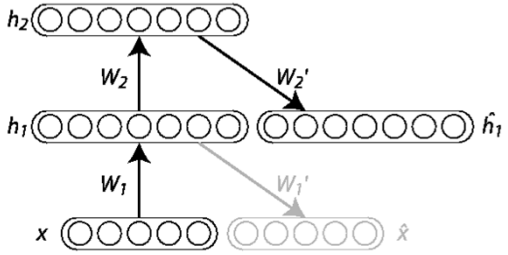


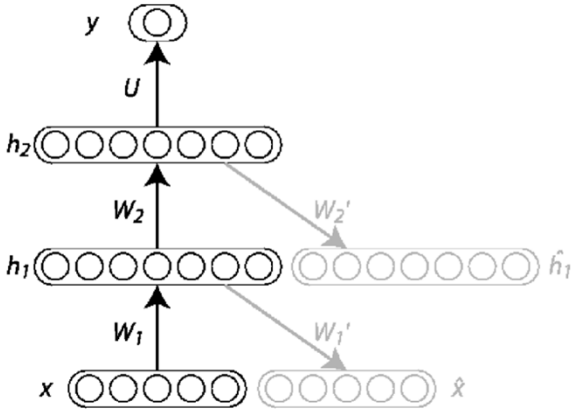
- Depending on starting point, BP converge or diverge with probability 1
  - A serious problem in Large-Scale DNN

Some new ideas to speed up

- Stacking from smaller building blocks
  - Layers
  - Blocks





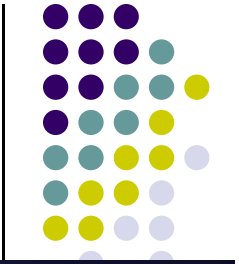


- Approximate Inference
  - Undirected connections for all layers (Markov net) [Related work: Salakhutdinov and Hinton, 2009]
  - Block Gibbs sampling or mean-field
  - Hierarchical probabilistic inference
- Layer-wise Unsupervised Learning

Backprop in Practice

Y LeCun
MA Ranzato

- Use ReLU non-linearities (tanh and logistic are falling out of favor)
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it's best to turn it on after a couple of epochs
- Use "dropout" for regularization
  - Hinton et al 2012 http://arxiv.org/abs/1207.0580
- Lots more in [LeCun et al. "Efficient Backprop" 1998]
- Lots, lots more in "Neural Networks, Tricks of the Trade" (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)



DL

Utility of the network

- A vehicle to conceptually synthesize complex decision hypothesis
  - stage-wise projection and aggregation
- A vehicle for organizing computing operations
  - stage-wise update of latent states
- A vehicle for designing processing steps/computing modules
  - Layer-wise parallization
- No obvious utility in evaluating DL algorithms

Utility of the Loss Function

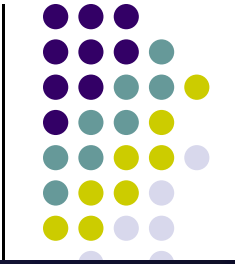
- Global loss? Well it is non-convex anyway, why bother ?

GM

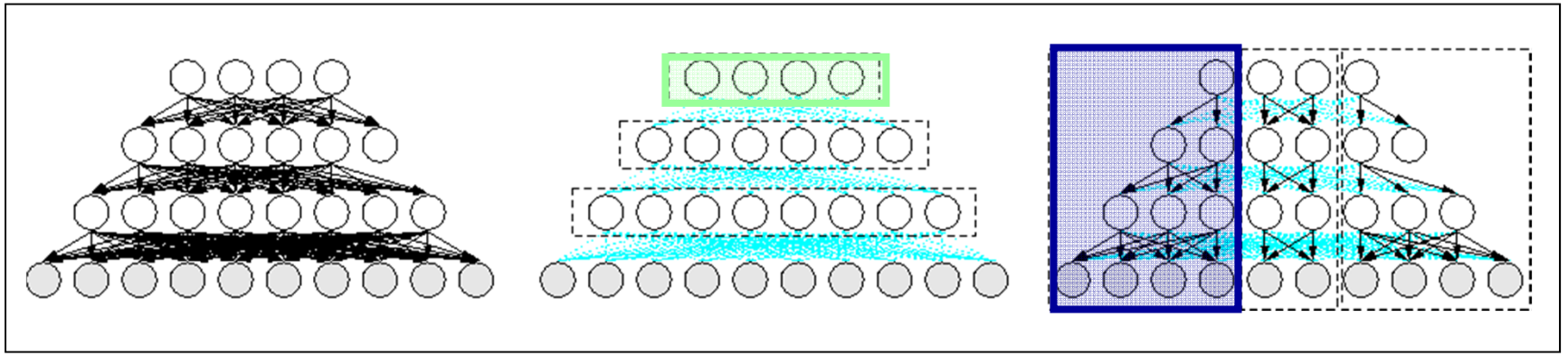
- A vehicle for synthesizing a global loss function from local structure
  - potential function, feature function
- A vehicle for designing sound and efficient inference algorithms
  - Sum-product, mean-field
- A vehicle to inspire approximation and penalization
  - Structured MF, Tree-approx
- A vehicle for monitoring theoretical and empirical behavior and accuracy of inference

- A major measure of quality of algorithm and model

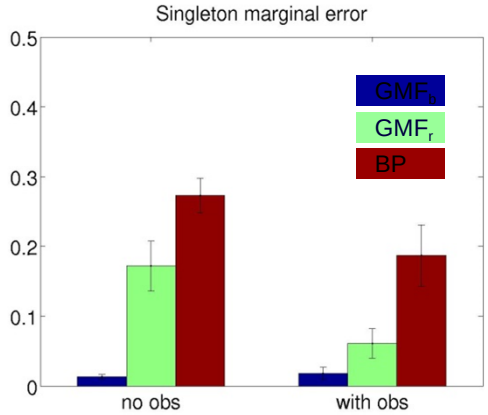
An Old Study of DL as GM Learning [Xing, Russell, Jordan, UAI 2003]

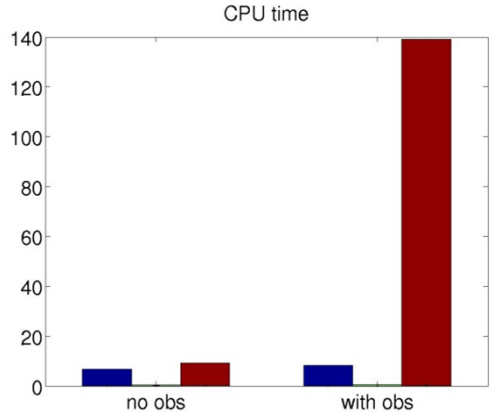


A sigmoid belief network at a GM, and mean-field partitions



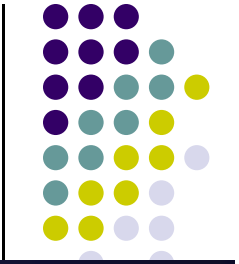
Study focused on only inference/learning accuracy, speed, and partition





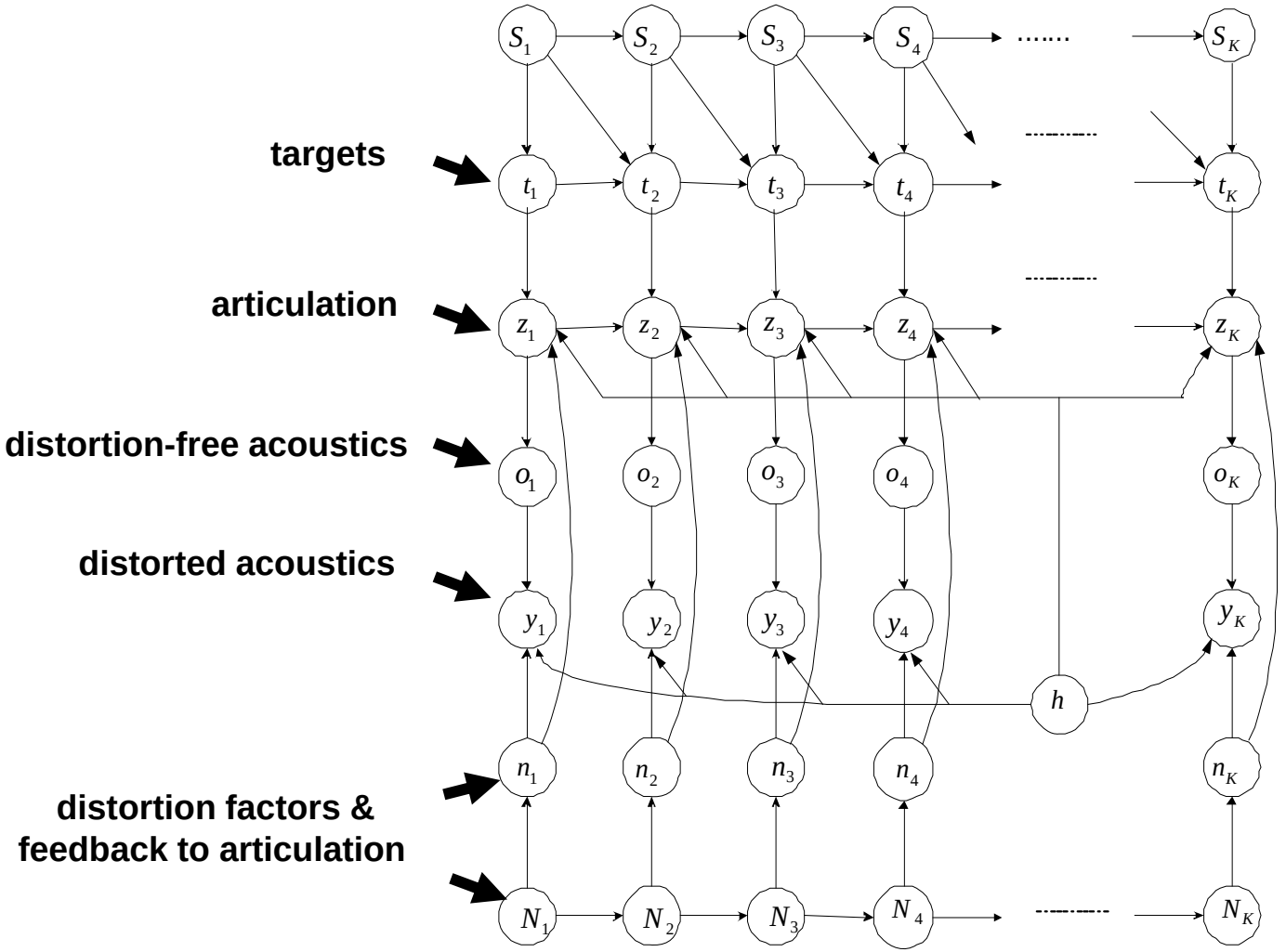
Now we can ask, with a correctly learned DN, is it doing will on the desired task?

Why A Graphical Model formulation of DL might be fruitful?



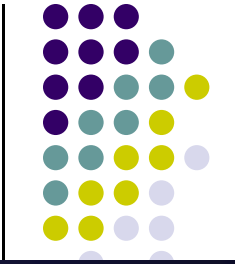
- **Modular design**: easy to incorporate knowledge and interpret, easy to integrate feature learning with high level tasks, easy to built on existing (partial) solutions
- Defines an **explicit and natural objective**
- Guilds strategies for **systematic study** of inference, parallelization, evaluation, and theoretical analysis
- A clear path to further **upgrade**:
  - structured prediction
  - Integration of multiple data modality
  - Modeling complex: time series, missing data, online data …
- Big DL on **distributed architectures**, where things can get messy everywhere due to incorrect parallel computations

Easy to incorporate knowledge and interpret



Slides Courtesy: Li Deng

Easy to integrate feature learning with high level tasks



Hidden Markov Model

+

Gaussian Mixture Model

Jointly trained, but shallow

Hidden Markov Model

+

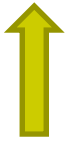
Deep Neural Network

Deep, but separately trained

Hidden Markov Model

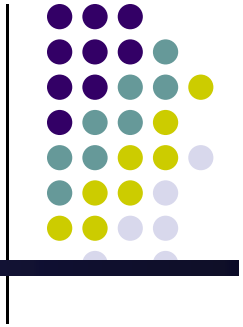
+

Deep Graphical Models



Jointly trained and deep

Conclusion



- In GM: lots of efforts are directed to improving inference accuracy and convergence speed
  - An advanced tutorial would survey dozen's of inference algorithms/theories, but few use cases on empirical tasks
- In DL: most effort is directed to comparing different architectures and gate functions (based on empirical performance on a downstream task)
  - An advanced tutorial typically consist of a list of all designs of nets, many use cases, but a single name of algorithm: back prop of SGD
- The two fields are similar at the beginning (energy, structure, etc.), and soon diverge to their own signature pipelines
- A convergence might be necessary and fruitful