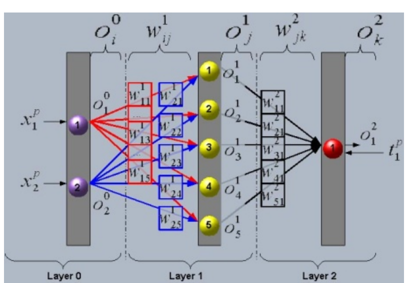
Probabilistic Graphical Models

Graphical Models and Deep Learning

Maruan Al-Shedivat

Lecture 18, March 27, 2017



Reading: see class website

DL module overview

- Lecture 18:
  - Historical remarks and an overview of DL building blocks
  - Similarities and differences between NNs and GMs
  - Ways to combine GMs and NNs
- Lecture 19:
  - Learning and inference in DL: VAEs and GANs
  - Ways to incorporate domain knowledge into NNs
  - NNs for NLP applications
- Lecture 20:
  - Convolutional and recurrent neural networks
  - Memory and attention mechanisms
  - Applications in computer vision

before 2015 → no DL lectures   2015 → 1 lecture   2016 → 2 lectures   2017 → 3 lectures

Outline

- An overview of the DL components
  - Historical remarks: early days of neural networks
  - Modern building blocks: units, layers, activations functions, loss functions, etc.
  - Reverse-mode automatic differentiation (aka backpropagation)
  - Distributed representations
- Similarities and differences between GMs and NNs
  - Graphical models vs. computational graphs
  - Sigmoid Belief Networks as graphical models
  - Deep Belief Networks and Boltzmann Machines
- Combining DL methods and GMs
  - Using outputs of NNs as inputs to GMs
  - GMs with potential functions represented by NNs
  - NNs with structured outputs

Outline

- An overview of the DL components
  - Historical remarks: early days of neural networks
  - Modern building blocks: units, layers, activations functions, loss functions, etc.
  - Reverse-mode automatic differentiation (aka backpropagation)
  - Distributed representations
- Similarities and differences between GMs and NNs
  - Graphical models vs. computational graphs
  - Sigmoid Belief Networks as graphical models
  - Deep Belief Networks and Boltzmann Machines
- Combining DL methods and GMs
  - Using outputs of NNs as inputs to GMs
  - GMs with potential functions represented by NNs
  - NNs with structured outputs

Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it…

Motivation

Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it…
  - DeepMind: Acquired by Google for **$400 million**
  - DNNResearch: **Three person startup** (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding **millions of VC dollars**

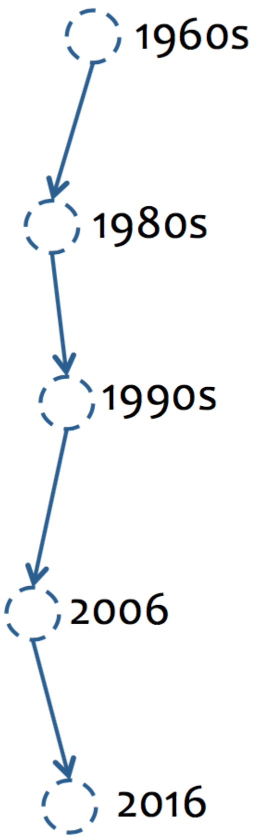
Motivation

Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it…
  - DeepMind: Acquired by Google for **$400 million**
  - DNNResearch: **Three person startup** (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding **millions of VC dollars**
- Because it made the **front page** of the New York Times

Motivation

Why is everyone talking about Deep Learning?



Deep learning:

- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn't always the case!

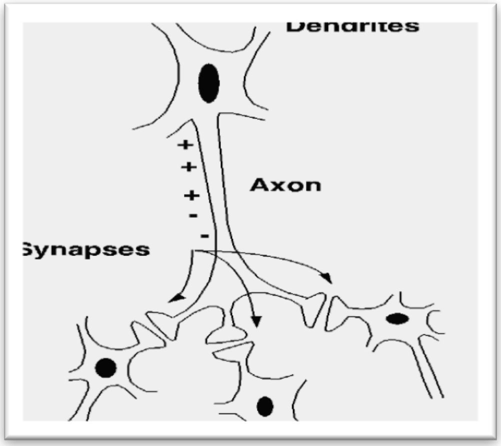
Since 1980s: Form of models hasn't changed much, but lots of new tricks…

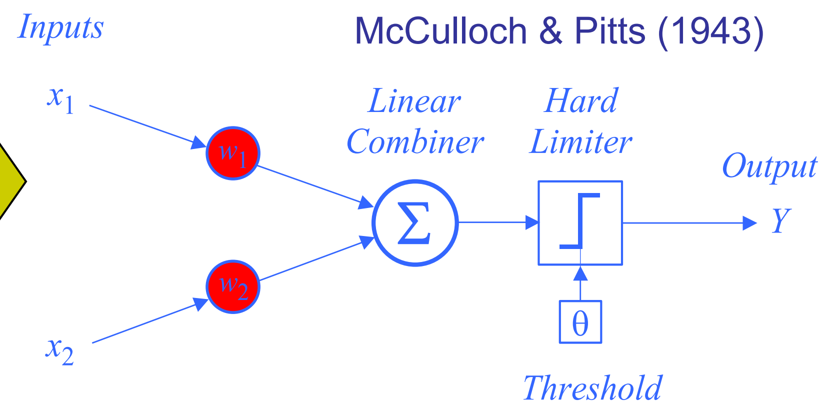
- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)

the slide is courtesy of Matt Gormley

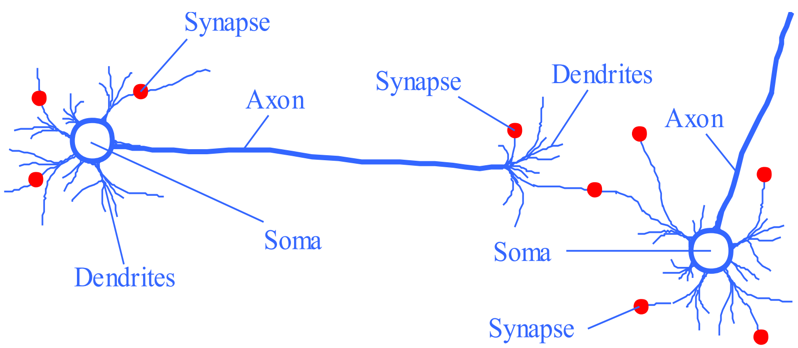
Perceptron and Neural Nets

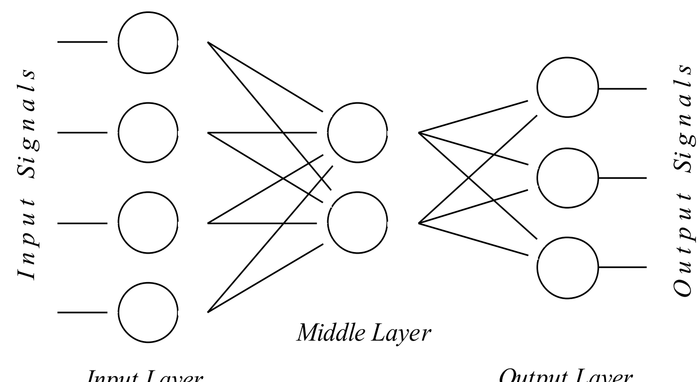
- From biological neuron to artificial neuron (perceptron)





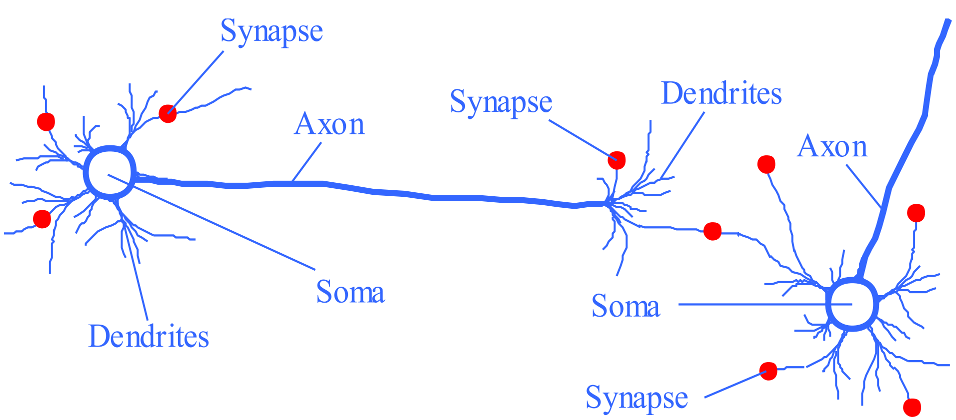
- From biological neuron network to artificial neuron networks

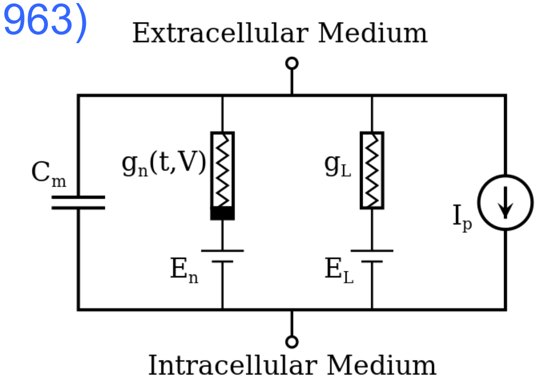


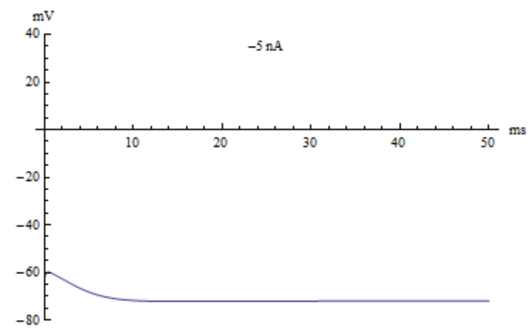


Remark:
Real biological neural networks

- The real biological neural networks are a *physical substrate*
  - Neural cells communicate using voltage pulses, aka *spikes*
  - Biological neural networks are completely analog, oscillating systems and can be described using systems of ordinary differential equations (ODEs)
  - E.g., the Hodgkin-Huxley (1952) model (Nobel Prize, 1963)





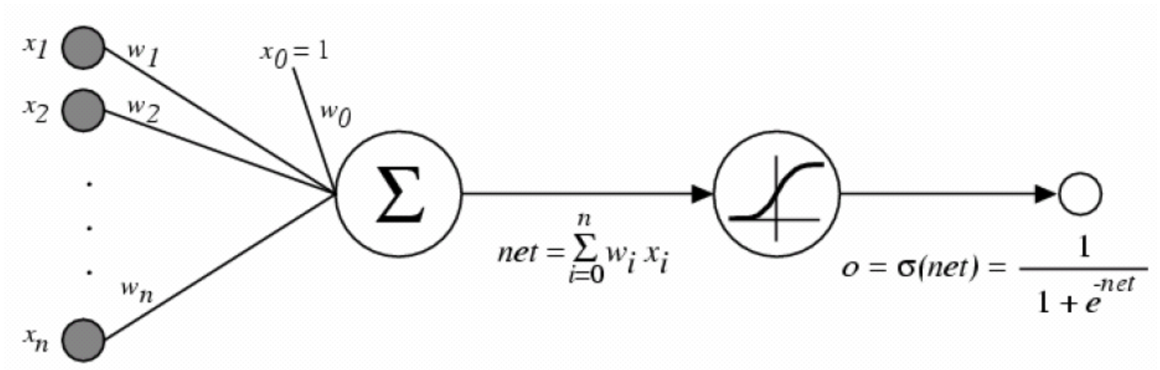


source: Wikipedia

Remark: Real biological neural networks

- The real biological neural networks are a *physical substrate*
  - Neural cells communicate using voltage pulses, aka *spikes*
  - Biological neural networks are completely analog, oscillating systems and can be described using systems of ordinary differential equations (ODEs)
  - E.g., the Hodgkin-Huxley (1952) model (Nobel Prize, 1963)
- "Computation" performed by real neural networks is still not understood well
  - Spiking models are often used for simulating brain activity, not for computing
  - There are various hypothesis of what exactly spiking networks compute:
    - E.g., the *neural sampling hypothesis* (Fiser et al., 2010, Trends in Cog. Sci.) states that chaotic neural behavior is used by the brain to approximate probability distributions, just like the Monte Carlo methods.
  - There is a research area termed "neuromorphic engineering" that aims to build efficient computing machines based on spiking neurons (e.g., TrueNorth chip)

The 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 …

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

The perceptron learning algorithm

x_d = input

t_d = target output

o_d = observed output

w_i =weight i

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

The perceptron learning algorithm

x_d = input

t_d = target output

o_d = observed 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:

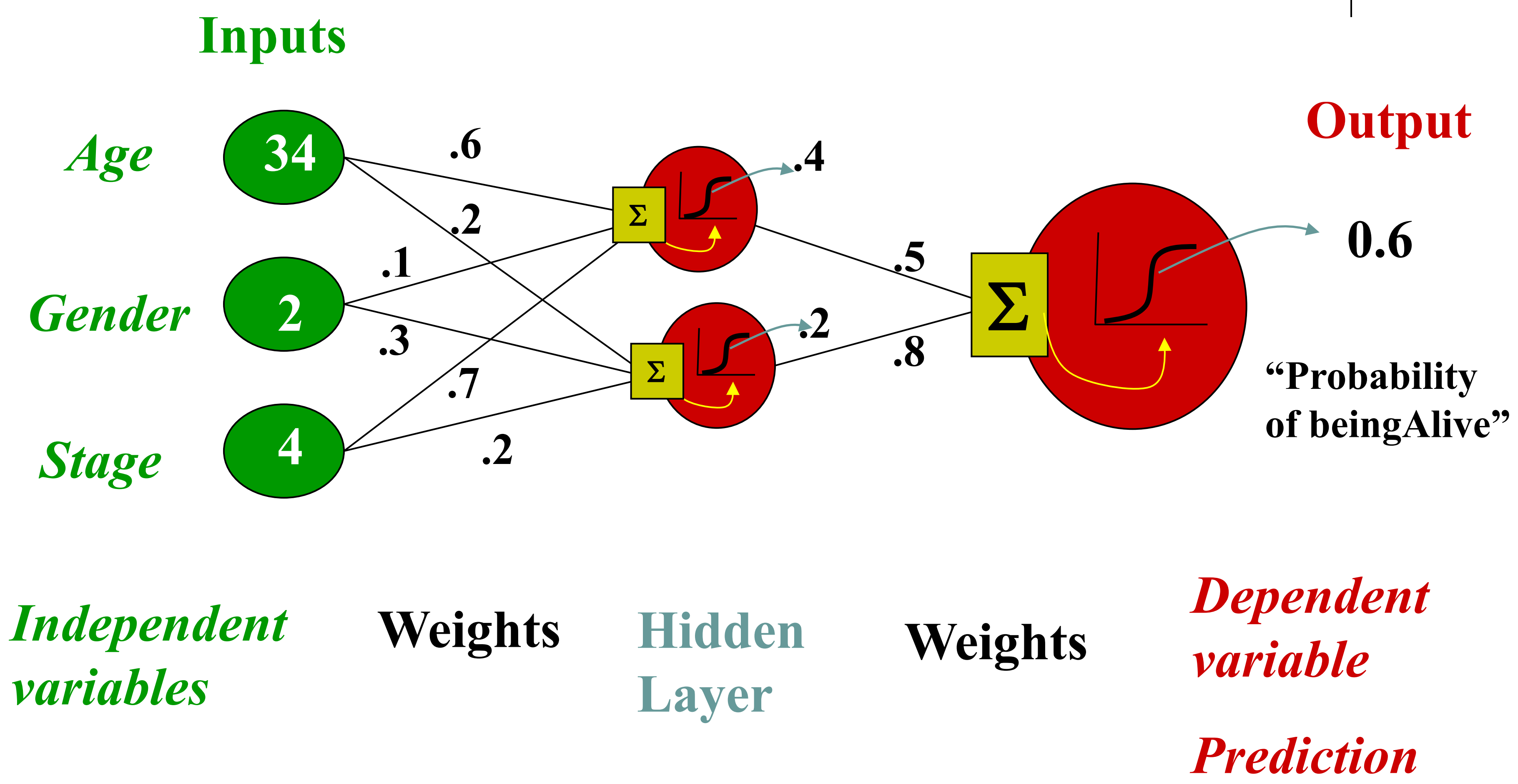
Do until converge:

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

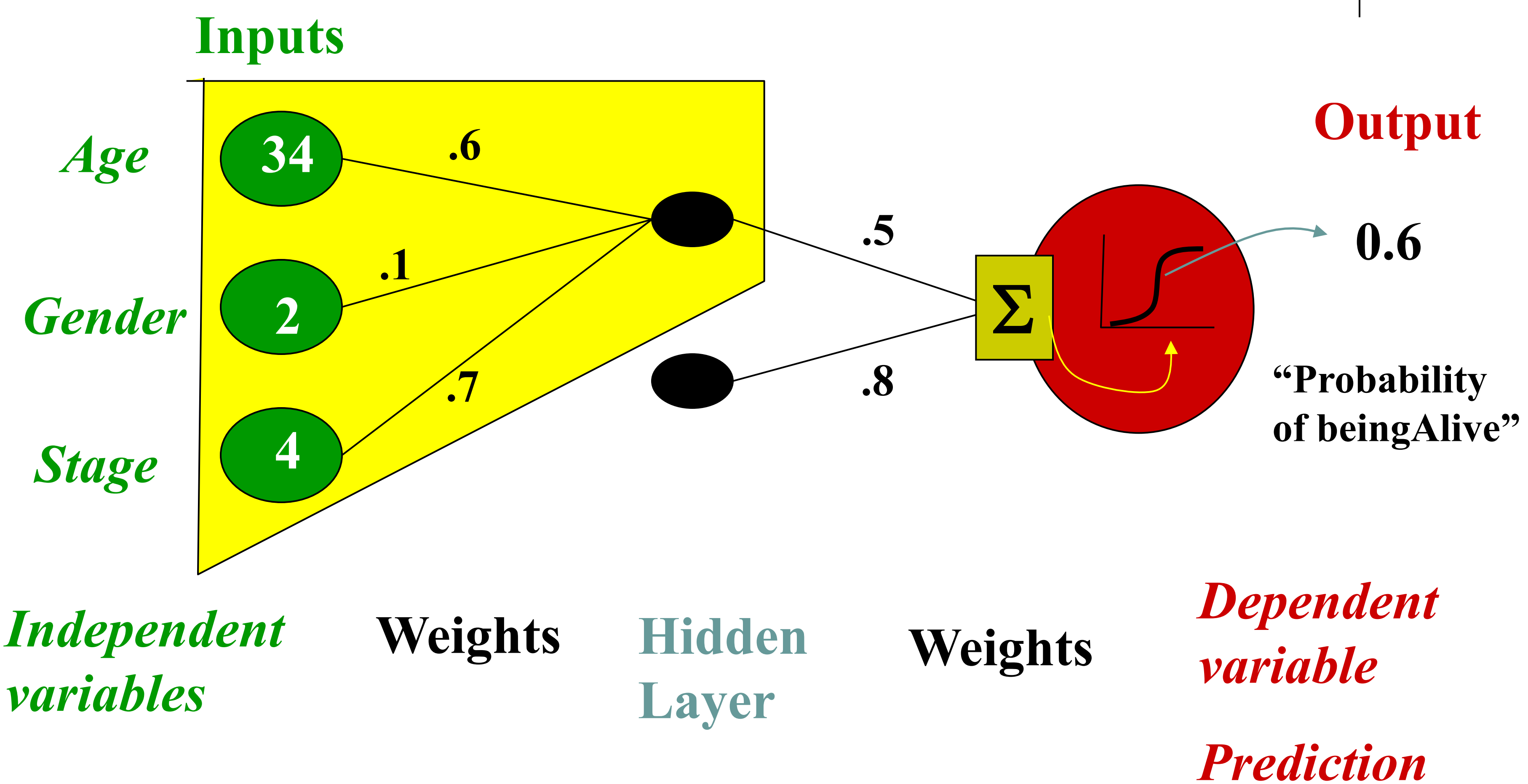
  where

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

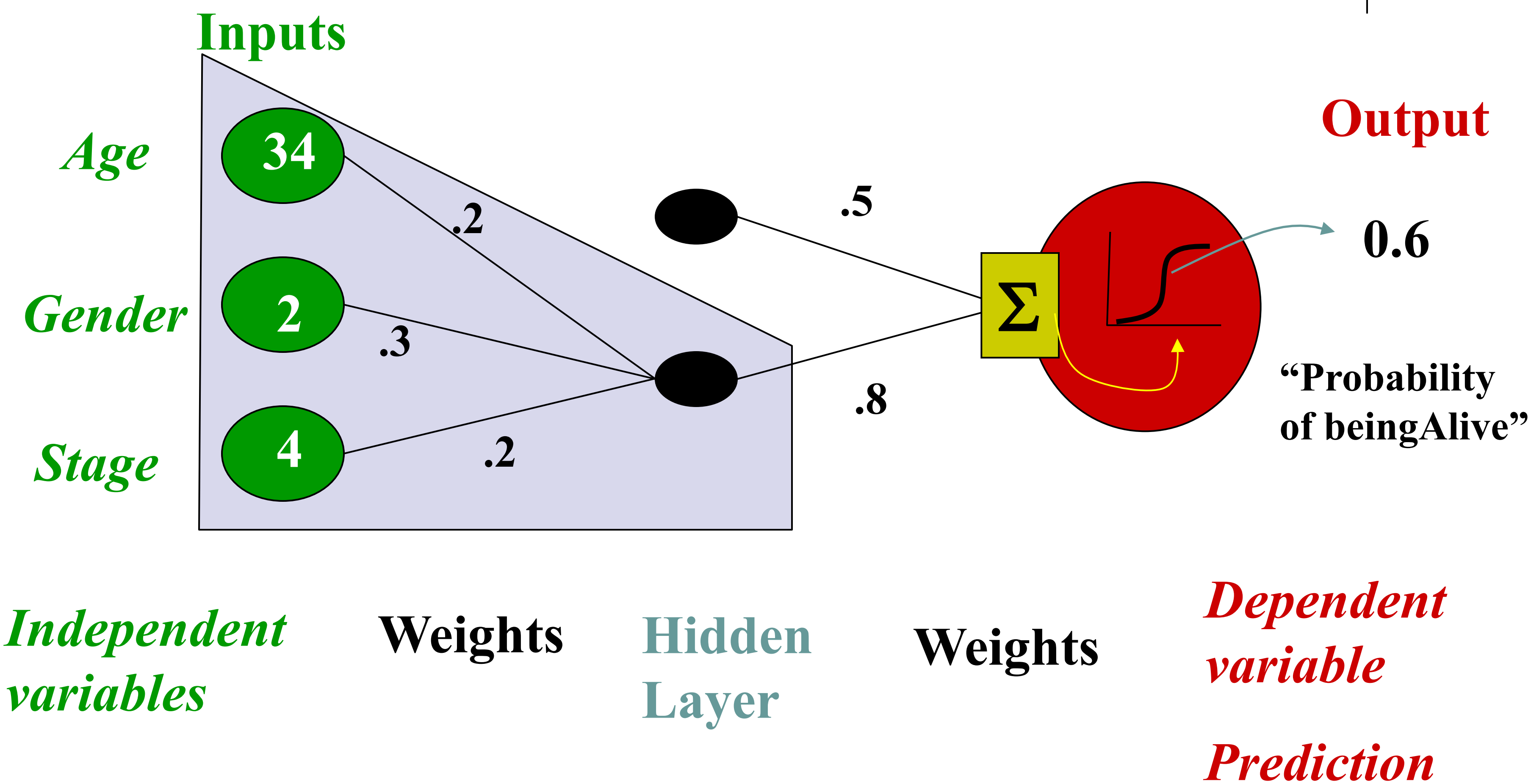
Neural Network Model



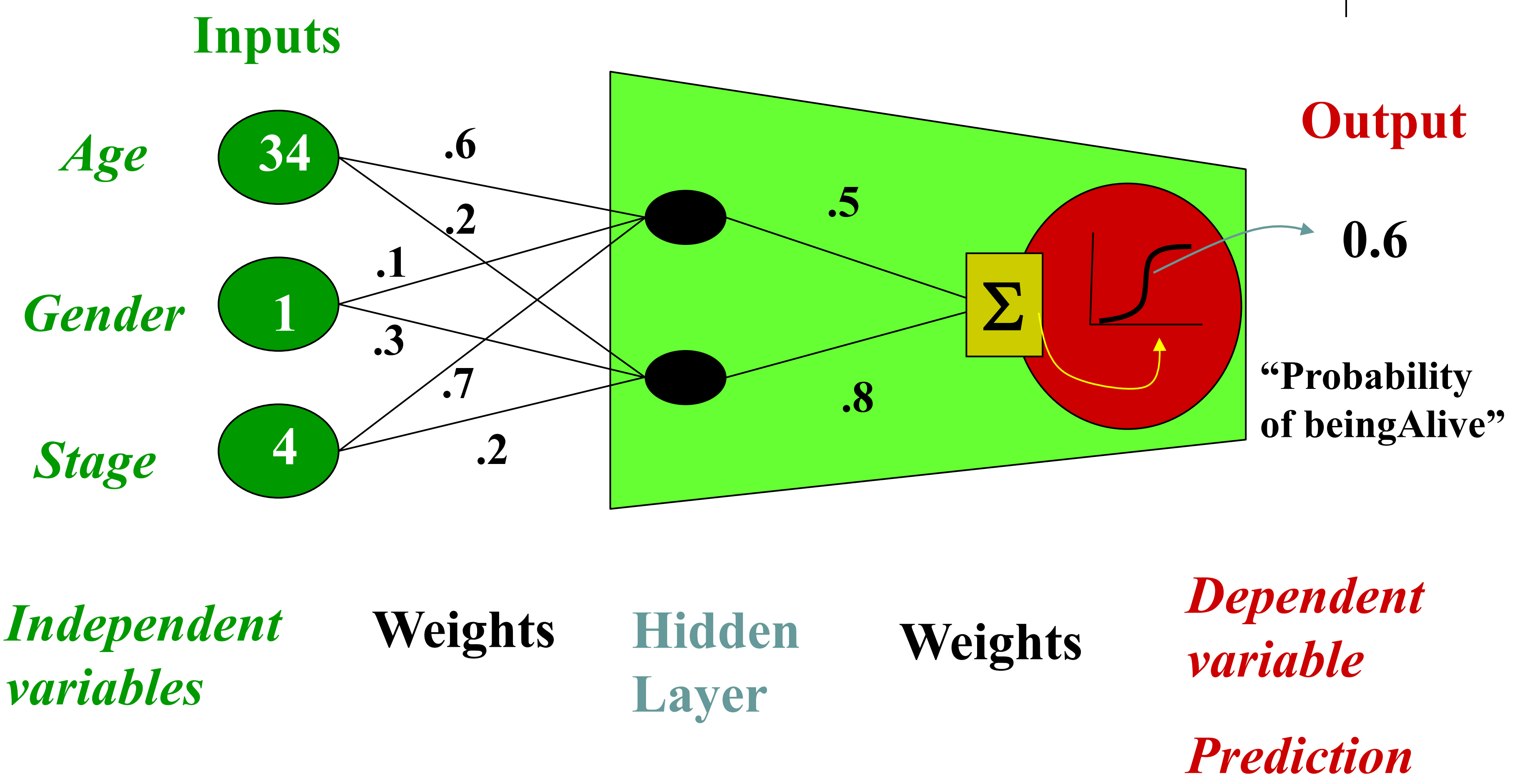
"Combined logistic models"



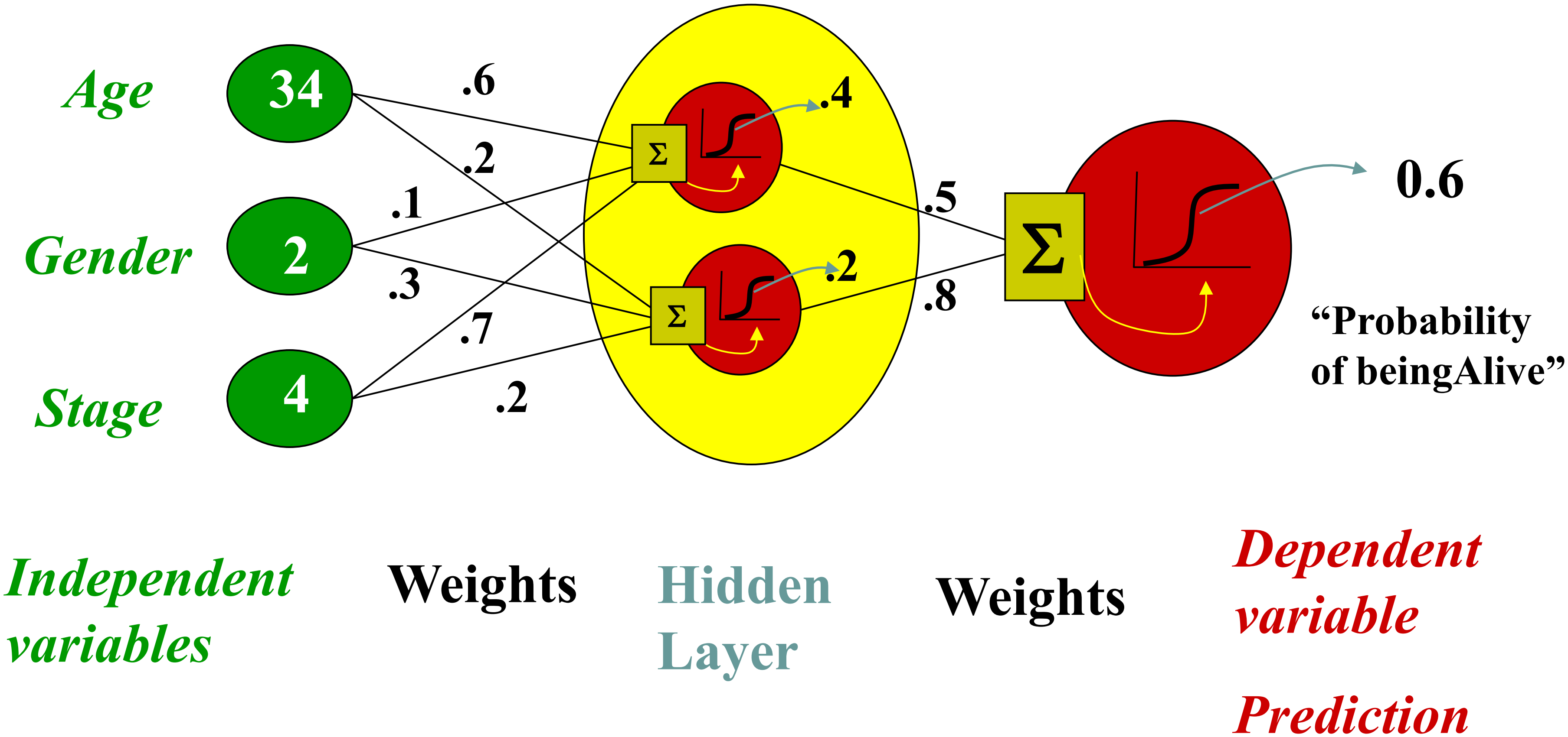
"Combined logistic models"



"Combined logistic models"

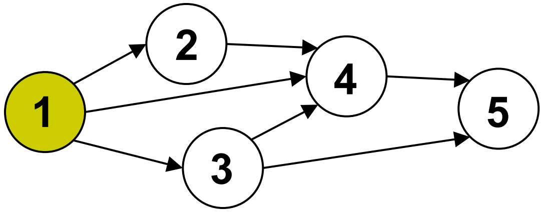


Not really, no target for hidden units...



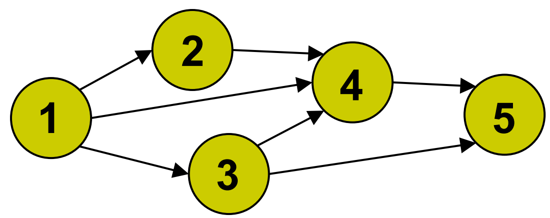
Backpropagation: Reverse-mode differentiation

- Artificial neural networks are nothing more than complex functional compositions that can be represented by computation graphs:



Backpropagation: Reverse-mode differentiation

- Artificial neural networks are nothing more than complex functional compositions that can be represented by computation graphs:



- By applying the chain rule and using reverse accumulation, we get

$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \frac{\partial f_{i_1}}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \sum_{i_2 \in \pi(i_1)} \frac{\partial f_{i_1}}{\partial f_{i_2}} \frac{\partial f_{i_1}}{\partial x} = \dots$$

- The algorithm is commonly known as backpropagation
- What if some of the functions are stochastic?
- Then use stochastic backpropagation!
  (to be covered in the next lecture)

Auto-reverse-mode differentiation

- A lot of engineering effort has been put into packages that can automatically compute derivatives for a given computation graph:

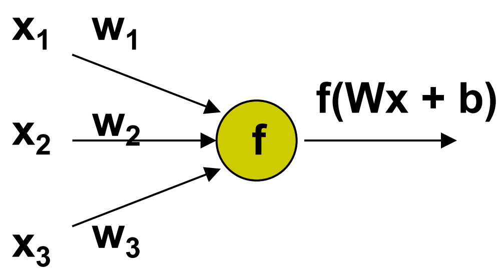
Caffe | Chainer | DL4J Deeplearning4j | KERAS | Microsoft CNTK | MatConvNet

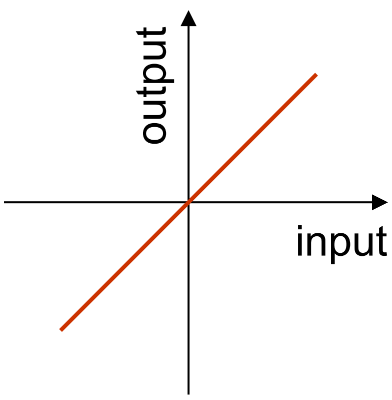
MINERVA | mxnet | Purine | TensorFlow | theano | torch

and the list is growing…

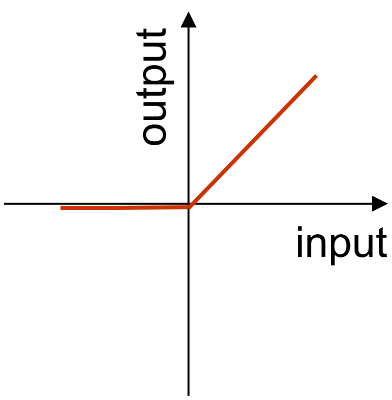
Modern building blocks

- Activation functions
  - Linear and ReLU





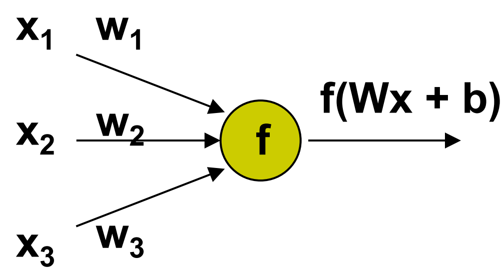
Linear

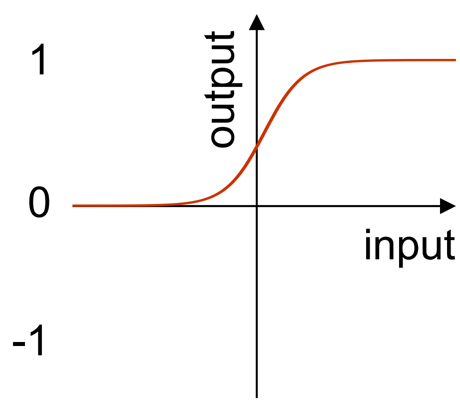


Rectified linear

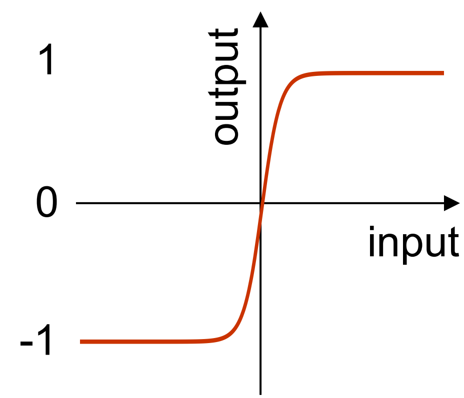
Building blocks of deep networks

- Activation functions
  - Linear and ReLU
  - Sigmoid and tanh
  - Etc.





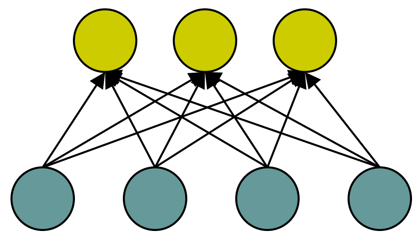
Sigmoid

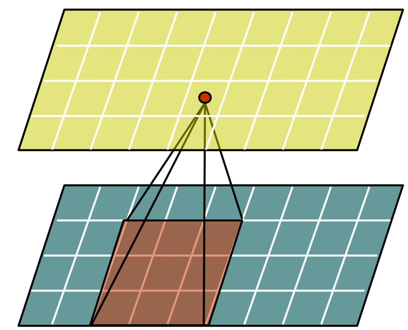


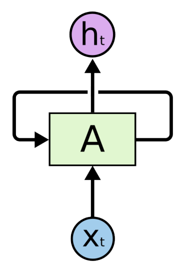
Hyperbolic tangent

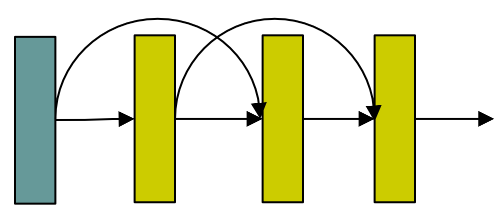
Building blocks of deep networks

- Activation functions
  - Linear and ReLU
  - Sigmoid and tanh
  - Etc.
- Layers
  - Fully connected
  - Convolutional & pooling
  - Recurrent
  - ResNets
  - Etc.









Building blocks of deep networks

- Activation functions
  - Linear and ReLU
  - Sigmoid and tanh
  - Etc.
- Layers
  - Fully connected
  - Convolutional & pooling
  - Recurrent
  - ResNets
  - Etc.
- Loss functions
  - Cross-entropy loss
  - Mean squared error
  - Etc.

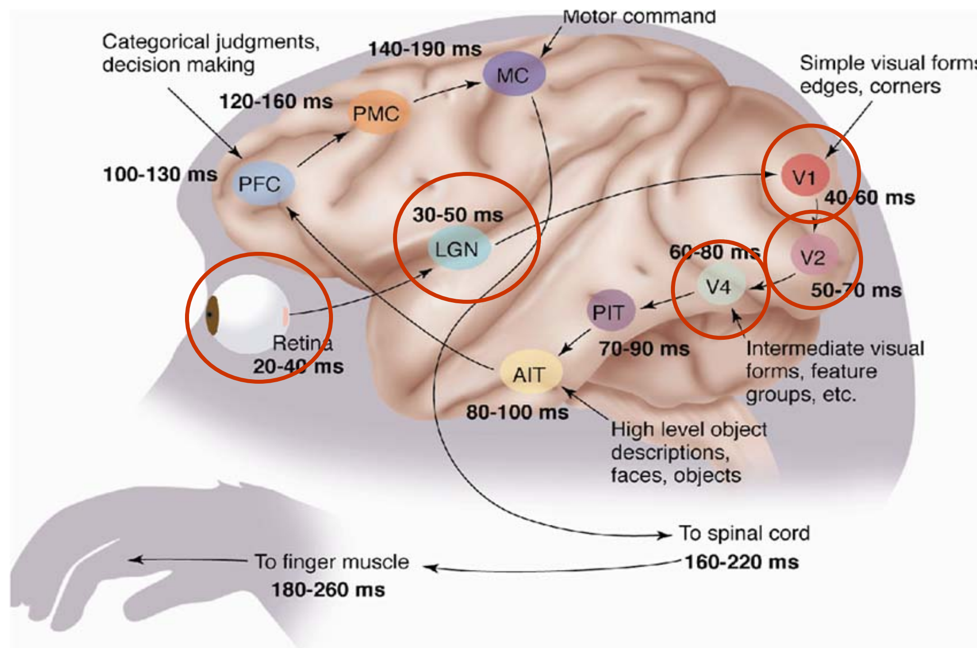
Putting things together:

(a part of GoogleNet)

- Arbitrary combinations of the basic building blocks
- Multiple loss functions – multi-target prediction, transfer learning, and more
- Given enough data, deeper architectures just keep improving
- Representation learning: the networks learn increasingly more abstract representations of the data that are "disentangled," i.e., amenable to linear separation.

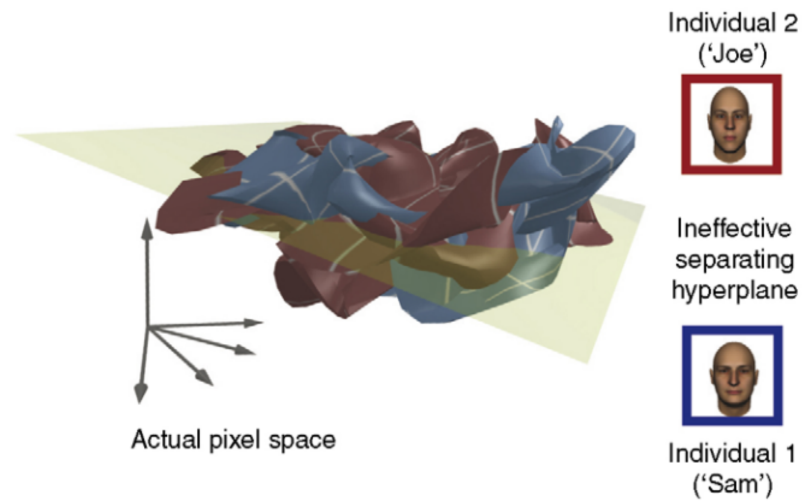
Inspiration: Signal processing in the brain

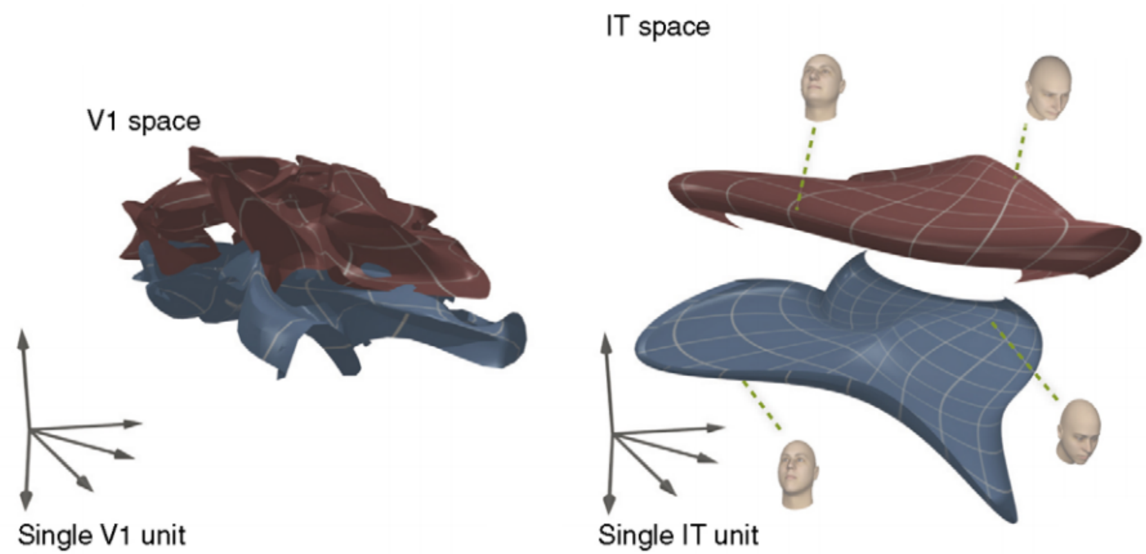
- "Computation" is hierarchical
- Hypothesis: the brain builds hierarchies of more and more abstract representations
- More abstract representation have nice invariant properties



Signal processing in the brain

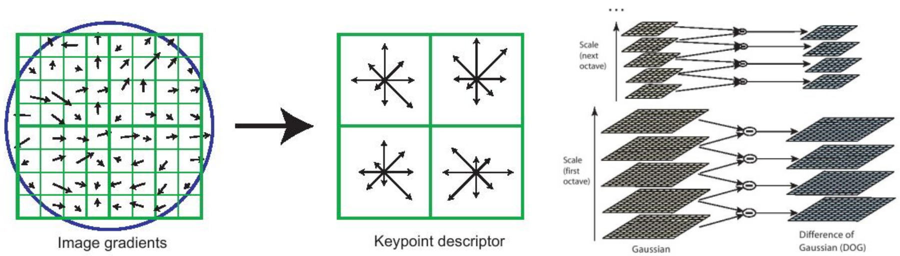
- "Computation" is hierarchical
- <u>Hypothesis:</u> the brain builds hierarchies of more and more abstract representations
- More abstract representation have nice invariant properties



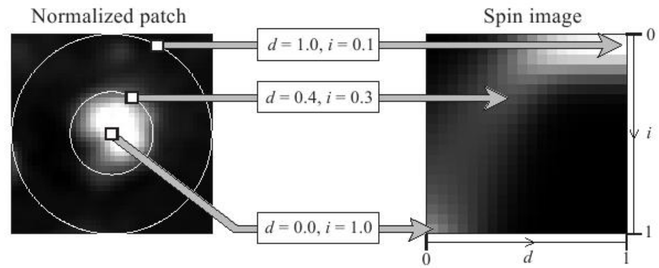


figures from DiCarlo & Cox, 2007

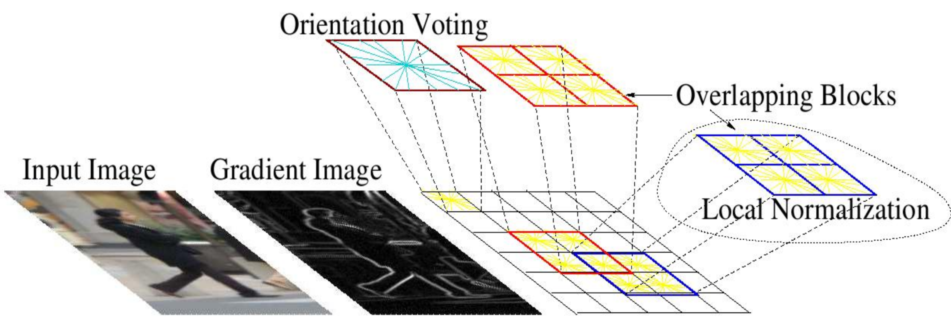
Hand-crafted features

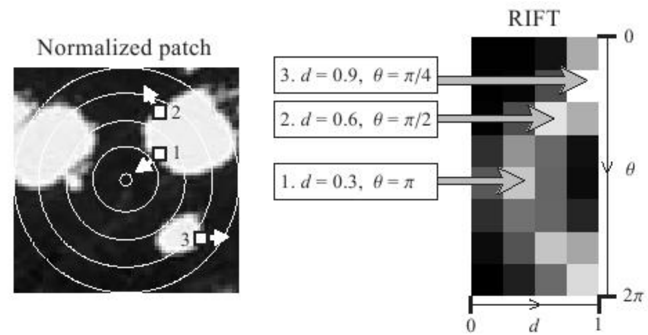


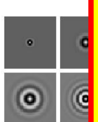
SIFT

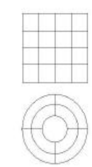


Spin image









(e)

Drawbacks of feature engineering

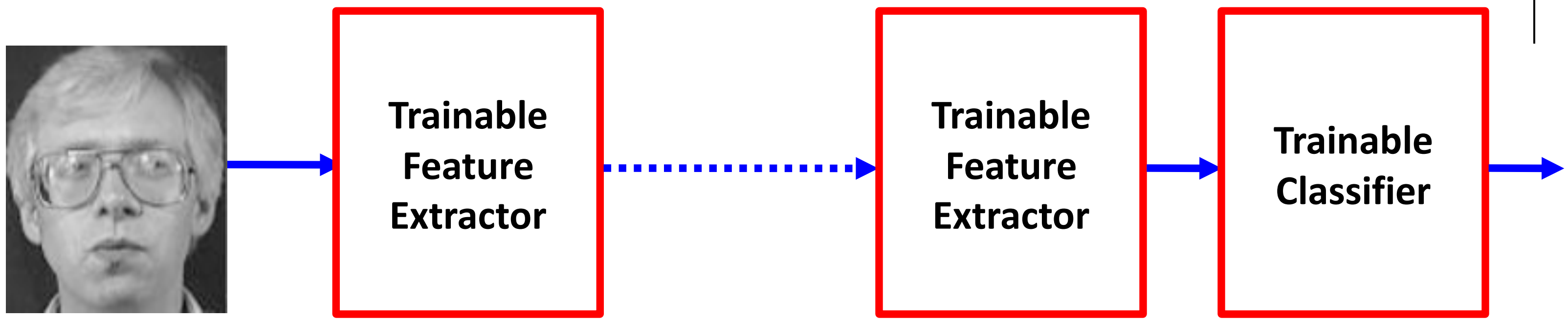
1. **Needs expert knowledge**
2. **Time consuming hand-tuning**
3. **Poor reproducibility**

Textons

GLOH

e and Ng

Using DNN to capture hierarchical representations



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

Outline

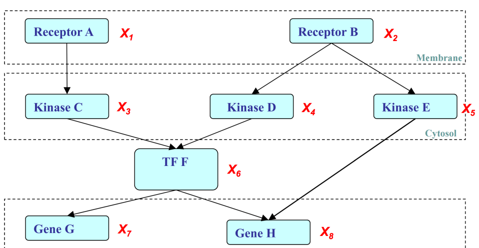
- An overview of the DL components
  - Historical remarks: early days of neural networks
  - Modern building blocks: units, layers, activations functions, loss functions, etc.
  - Reverse-mode automatic differentiation (aka backpropagation)
  - Distributed representations
- Similarities and differences between GMs and NNs
  - Graphical models vs. computational graphs
  - Sigmoid Belief Networks as graphical models
  - Deep Belief Networks and Deep Boltzmann Machines
- Combining DL methods and GMs
  - Using outputs of NNs as inputs to GMs
  - GMs with potential functions represented by NNs
  - NNs with structured outputs

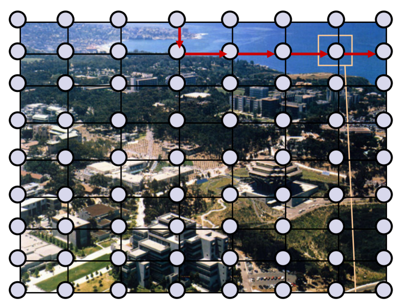
	DL	$\lesseqgtr$? ML (e.g., GM)
Empirical goal:	e.g., classification, feature learning	e.g., latent variable inference, transfer learning
Structure:	Graphical	Graphical
Objective:	Something aggregated from local functions	Something aggregated from local functions
Vocabulary:	Neuron, activation function, …	Variable, potential function, …
Algorithm:	A single, unchallenged, inference algorithm – Backpropagation (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 tricks	More or less standardized
Experiments:	Massive, real data (GT unknown)	Modest, often simulated data (GT known)

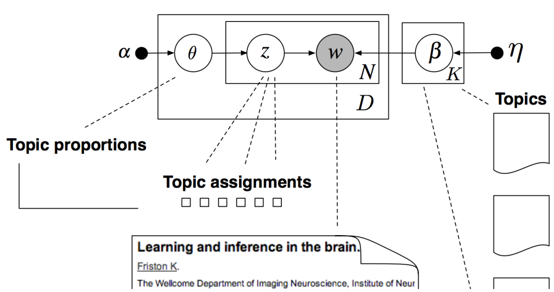
Graphical models vs. Deep nets

Graphical models

- Representation for encoding meaningful knowledge and the associated uncertainty in a graphical form

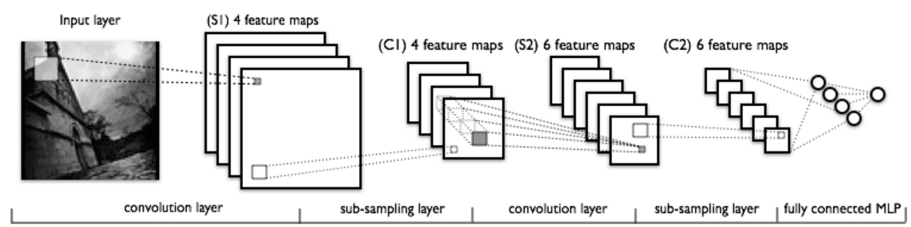


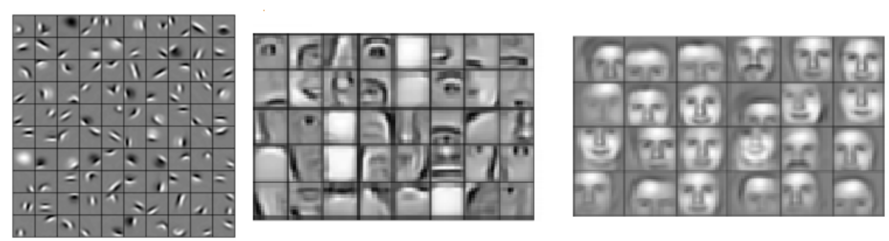




Deep neural networks

- Learn representations that facilitate computation and performance on the end-metric (intermediate representations may not be meaningful)





Graphical models vs. Deep nets

Graphical models

- Representation for encoding meaningful knowledge and the associated uncertainty in a graphical form
- Learning and inference are based on a rich toolbox of well-studied (structure-dependent) techniques (e.g., EM, message passing, VI, MCMC, etc.)
- Graphs represent models

Deep neural networks

- Learn representations that facilitate computation and performance on the end-metric (intermediate representations may not be meaningful)
- Learning is predominantly based on the gradient descent method (aka backpropagation); Inference is often trivial and done via a "forward pass"
- Graphs represent computation

Graphical models vs. Deep nets

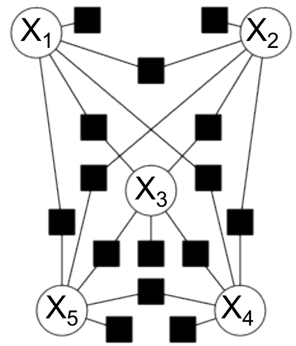
Graphical models

Utility of the graph

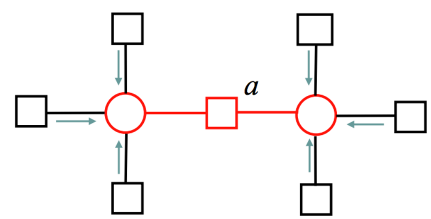
- A vehicle for synthesizing a global loss function from local structure
  - potential function, feature function, etc.
- A vehicle for designing sound and efficient inference algorithms
  - Sum-product, mean-field, etc.
- A vehicle to inspire approximation and penalization
  - Structured MF, Tree-approximation,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

Utility of the loss function

- A major measure of quality of the learning algorithm and the model



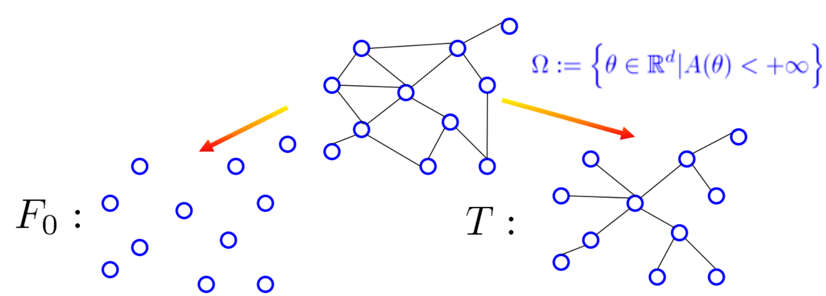
$$\log P(X) = \sum_i \log \phi(x_i) + \sum_{i,j} \log \psi(x_i, x_j)$$



$$m_{i\to a}(x_i) = \prod_{c\in N(i)\backslash a} m_{c\to i}(x_i)$$

$$b_a(X_a) \propto f_a(X_a) \prod_{i\in N(a)} m_{i\to a}(x_i)$$

$$m_{a\to i}(x_i) = \sum_{X_a\backslash x_i} f_a(X_a) \prod_{j\in N(a)\backslash i} m_{j\to a}(x_j)$$



$\Omega(F_0) := \{\theta \in \Omega \mid \theta_{(s,t)} = 0 \, \forall \, (s,t) \in E\}$. $\Omega(T) := \{\theta \in \Omega \mid \theta_{(s,t)} = 0 \;\; \forall \; (s,t) \notin E(T)\}$

Graphical models vs. Deep nets

Graphical models

Utility of the graph

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

Utility of the loss function

- A major measure of quality of the learning algorithm and the model

Deep neural networks

Utility of the network

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

Utility of the Loss Function

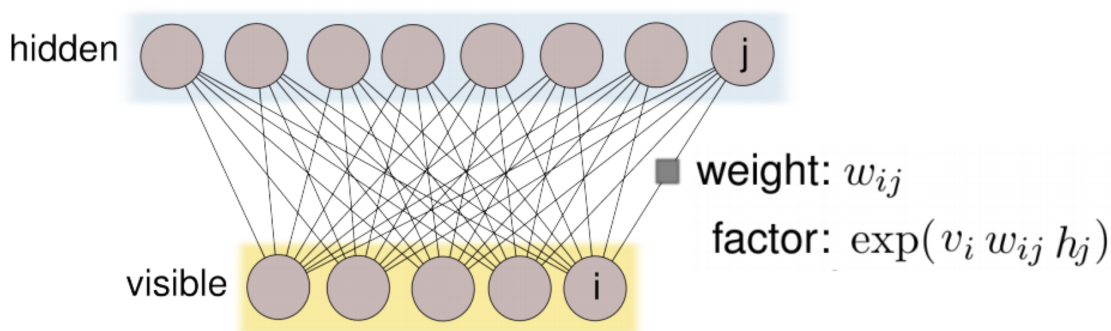
- Global loss? Well it is complex and non-convex...

Graphical models vs. Deep nets

- So far: neural nets are flexible function approximators
- Some of the neural nets are in fact proper graphical models (i.e., units/neurons represent proper random variables):
  - Boltzmann machines (Hinton & Sejnowsky, 1983)
  - Restricted Boltzmann machines (Smolensky, 1986)
  - Learning and Inference in sigmoid belief networks (Neal, 1992)
  - Fast learning in deep belief networks (Hinton, Osindero, Teh, 2006)
  - Deep Boltzmann machines (Salakhutdinov and Hinton, 2009)
- Let's go through these models one-by-one

Restricted Boltzmann Machines

- Assume visible units are one layer, and hidden units are another.
- Throw out all the connections within each layer.



Slide from Marcus Frean, MLSS Tutorial 2010

75

Restricted Boltzmann Machines: Learning and Inference

$$\frac{\partial}{\partial w} \log L \quad \propto$$

$$\underbrace{\frac{1}{N} \sum_{\mathbf{v} \in \mathcal{D}}}_{\text{data}} \underbrace{\sum_{\mathbf{h}} P(\mathbf{h} \mid \mathbf{v})}_{\text{av. over posterior}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \quad - \underbrace{\sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v}, \mathbf{h})}_{\text{av. over joint}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$$

Both terms involve averaging over $\frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$.

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzmann Machines: Learning and Inference

$$\frac{\partial}{\partial w} \log L \quad \propto$$

$$\underbrace{\frac{1}{N}\sum_{\mathbf{v}\in\mathcal{D}}}_{\text{data}} \underbrace{\sum_{\mathbf{h}} P(\mathbf{h} \mid \mathbf{v})}_{\text{av. over posterior}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \;-\; \underbrace{\sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v},\mathbf{h})}_{\text{av. over joint}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$$

Both terms involve averaging over $\frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$.

Another way to write it:

$$\left\langle \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \right\rangle_{\mathbf{v}\in\mathcal{D},\;\; \mathbf{h}\sim P(\mathbf{h}|\mathbf{v})} \quad - \quad \left\langle \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \right\rangle_{\mathbf{x}\sim P(\mathbf{x})}$$

clamped / wake phase	unclamped / sleep / free phase
↑↑↑ conditioned hypotheses	↓↓↓ random fantasies

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzmann Machines: Learning and Inference

$$\frac{\partial}{\partial w}\log L \quad \propto$$

$$\underbrace{\frac{1}{N}\sum_{\mathbf{v}\in\mathcal{D}}}_{\text{data}} \underbrace{\sum_{\mathbf{h}} P(\mathbf{h}\mid\mathbf{v})}_{\text{av. over posterior}} \frac{\partial}{\partial w}\log P^\star(\mathbf{x}) \;-\; \underbrace{\sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v},\mathbf{h})}_{\text{av. over joint}} \frac{\partial}{\partial w}\log P^\star(\mathbf{x})$$

Contrastive Divergence estimates the second term with a Monte Carlo estimate from 1-step of a Gibbs sampler!

Both terms involve averaging over $\frac{\partial}{\partial w}\log P^\star(\mathbf{x})$.

Another way to write it:

$$\left\langle \frac{\partial}{\partial w}\log P^\star(\mathbf{x}) \right\rangle_{\mathbf{v}\in\mathcal{D},\ \mathbf{h}\sim P(\mathbf{h}|\mathbf{v})} \;-\; \left\langle \frac{\partial}{\partial w}\log P^\star(\mathbf{x}) \right\rangle_{\mathbf{x}\sim P(\mathbf{x})}$$

clamped / wake phase

↑↑↑ conditioned hypotheses

unclamped / sleep / free phase

↓↓↓ random fantasies

Slide from Marcus Frean, MLSS Tutorial 2010

Restricted Boltzmann Machines: Learning and Inference

$$\frac{\partial}{\partial w} \log L \quad \propto$$

$$\underbrace{\frac{1}{N}\sum_{\mathbf{v}\in\mathcal{D}}}_{\text{data}} \underbrace{\sum_{\mathbf{h}} P(\mathbf{h} \mid \mathbf{v})}_{\text{av. over posterior}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \;-\; \underbrace{\sum_{\mathbf{v},\mathbf{h}} P(\mathbf{v},\mathbf{h})}_{\text{av. over joint}} \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$$

Contrastive Divergence estimates the second term with a Monte Carlo estimate from 1-step of a Gibbs sampler!

Both terms involve averaging over $\frac{\partial}{\partial w} \log P^{\star}(\mathbf{x})$.

Another way to write it:

$$\left\langle \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \right\rangle_{\mathbf{v}\in\mathcal{D},\ \mathbf{h}\sim P(\mathbf{h}|\mathbf{v})} \;-\; \boxed{\left\langle \frac{\partial}{\partial w} \log P^{\star}(\mathbf{x}) \right\rangle_{\mathbf{x}\sim P(\mathbf{x})}}$$

clamped / wake phase

↑↑↑ conditioned hypotheses

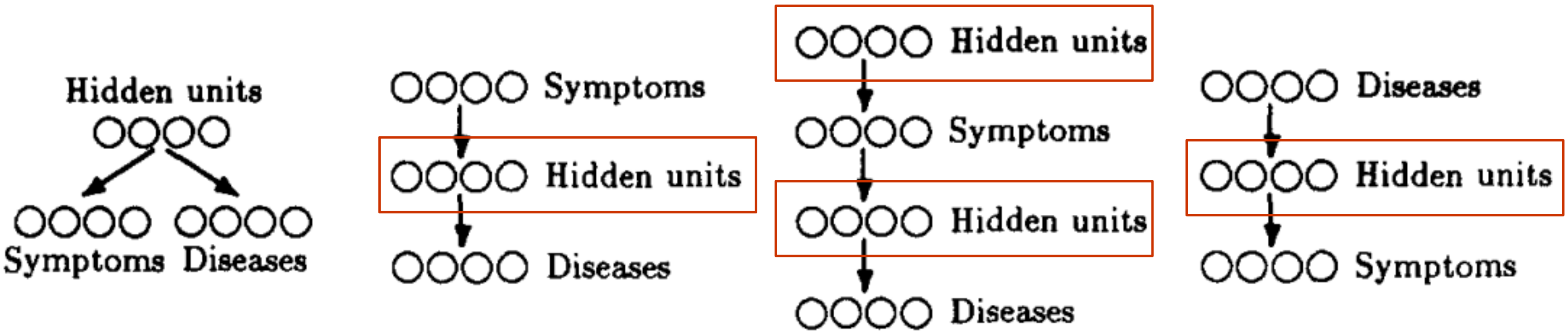
unclamped / sleep / free phase

↓↓↓ random fantasies

Slide from Marcus Frean, MLSS Tutorial 2010

The negative phase is problematic:
- slow convergence & extra computation

Sigmoid Belief Networks



- Sigmoid belief nets are simply Bayes networks with conditionals represented in a particular form:

$$
\begin{aligned}
P(S_i = x \mid S_j = s_j : j \neq i) \\
\quad \propto P(S_i = x \mid S_j = s_j : j < i) \\
\qquad \cdot \prod_{j>i} P(S_j = s_j \mid S_i = x,\ S_k = s_k : k < j,\ k \neq i)
\end{aligned}
$$

$$
P(S_i = s_i \mid S_j = s_j : j < i) \;=\; \sigma\left(s_i^* \sum_{j<i} s_j w_{ij}\right)
$$

from Neal, 1992

Sigmoid Belief Networks: Learning and Inference

- Radford Neal proposed to use Monte Carlo methods to do inference (Neal, 1992):

$$
\begin{aligned}
\frac{\partial L}{\partial w_{ij}} &= \sum_{\tilde{v}\in\mathcal{T}} \frac{1}{P(\tilde{V}=\tilde{v})} \frac{\partial P(\tilde{V}=\tilde{v})}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \frac{1}{P(\tilde{V}=\tilde{v})} \sum_{\tilde{h}} \frac{\partial P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle)}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{h}} P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle \mid \tilde{V}=\tilde{v}) \cdot \frac{1}{P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle)} \frac{\partial P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle)}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{s}} P(\tilde{S}=\tilde{s} \mid \tilde{V}=\tilde{v}) \frac{1}{P(\tilde{S}=\tilde{s})} \frac{\partial P(\tilde{S}=\tilde{s})}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{s}} P(\tilde{S}=\tilde{s} \mid \tilde{V}=\tilde{v}) \cdot \frac{1}{\sigma\left(s_i^* \sum_{k<i} s_k w_{ik}\right)} \frac{\partial \sigma\left(s_i^* \sum_{k<i} s_k w_{ik}\right)}{\partial w_{ij}} \\
&= \boxed{\sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{s}} P(\tilde{S}=\tilde{s} \mid \tilde{V}=\tilde{v})\, s_i^* s_j \sigma\Big(-s_i^* \sum_{k<i} s_k w_{ik}\Big).}
\end{aligned}
$$

Approximated with Gibbs sampling

- Conditional distributions:

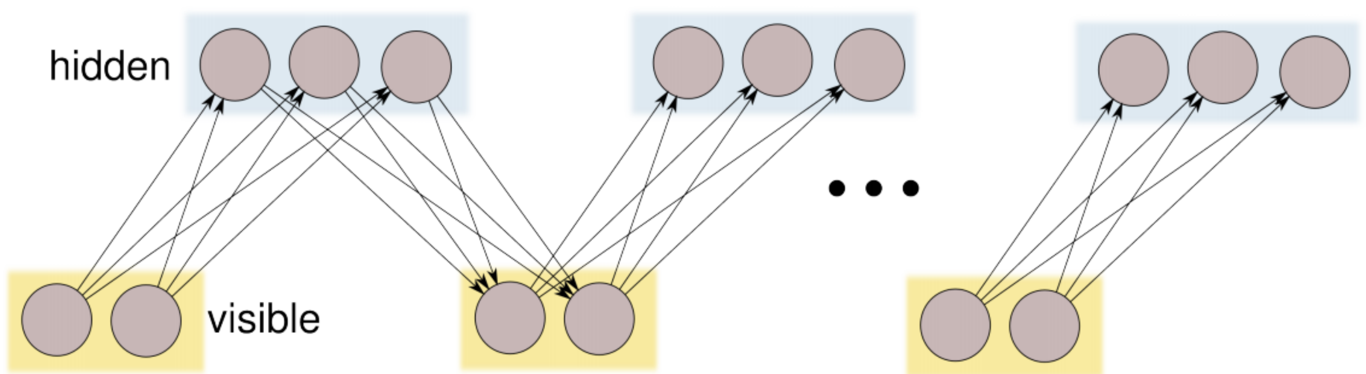
$$
P(S_i = x \mid S_j = s_j : j \neq i) \propto \sigma\Big(x^* \sum_{j<i} s_j w_{ij}\Big) \prod_{j>i} \sigma\Big(s_j^* \Big(x w_{ji} + \sum_{k<j,\,k\neq i} s_k w_{jk}\Big)\Big)
$$

- No negative phase as in RBM!
- Convergence is very slow, especially for large belief nets, due to the intricate “explain-away” effects…

RBMs are infinite belief networks

Alternating Gibbs sampling

Since none of the units within a layer are interconnected, we can do Gibbs sampling by updating the whole layer at a time.

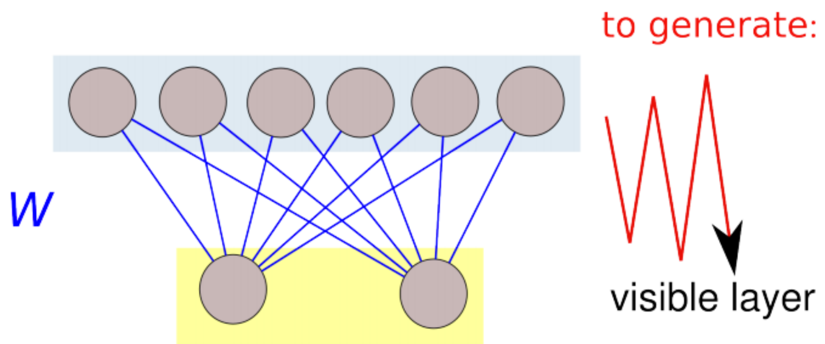


(with time running from left $\longrightarrow$ right)

Slide from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

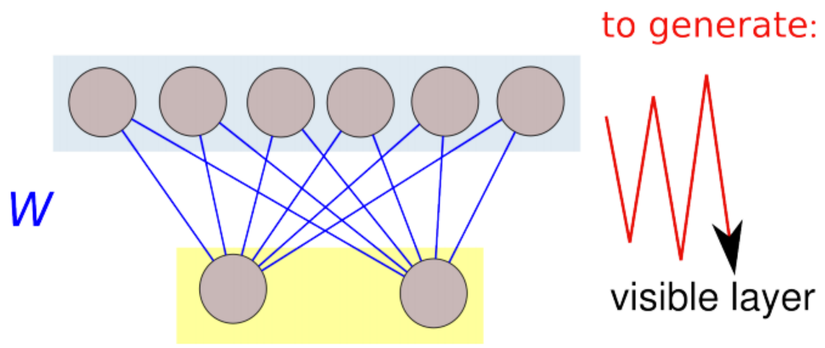
RBMs are equivalent to infinitely deep belief networks

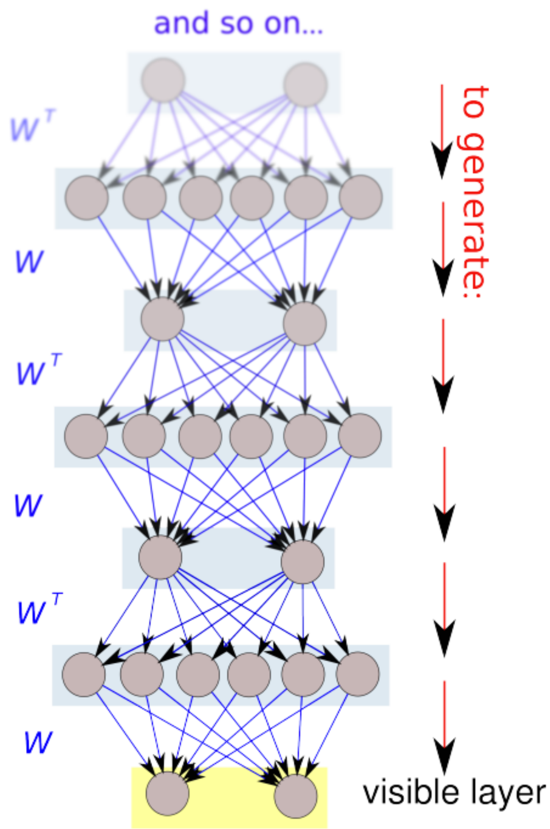


Slide from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

RBMs are equivalent to infinitely deep belief networks



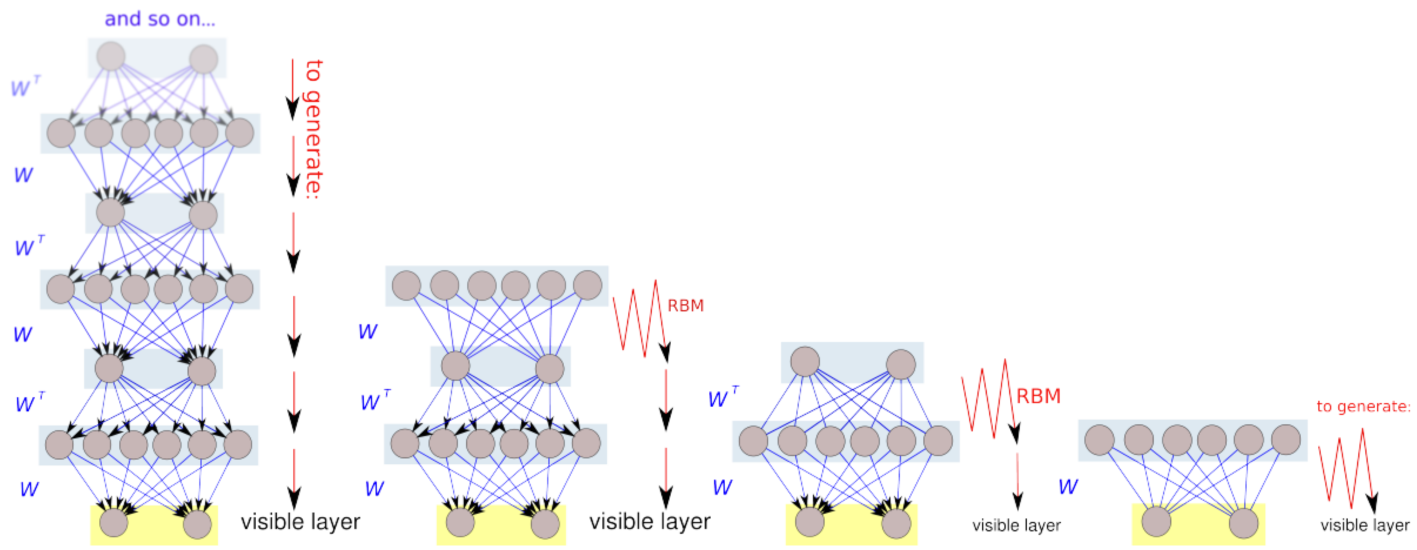


sampling from this is the same as sampling from the network on the right.

Slide from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

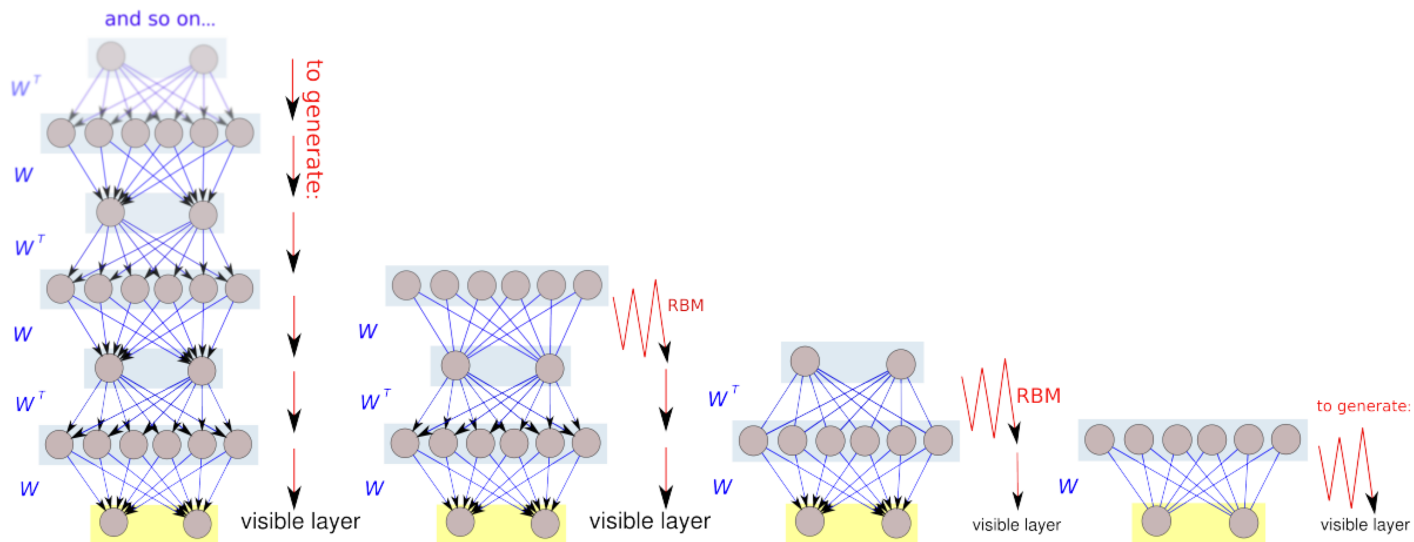
RBMs are equivalent to infinitely deep belief networks



Slide from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

RBMs are equivalent to infinitely deep belief networks



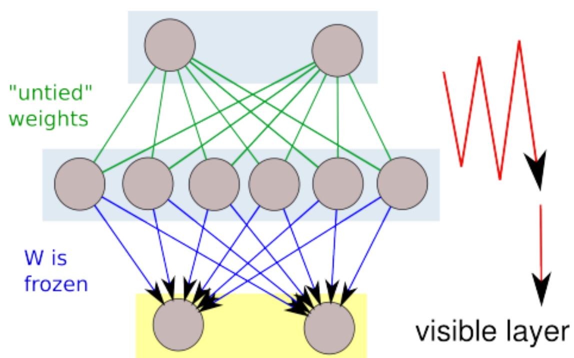
- So when we train an RBM, we're really training an ∞^{ly} deep sigmoid belief net!
- It's just that the weights of all layers are tied.

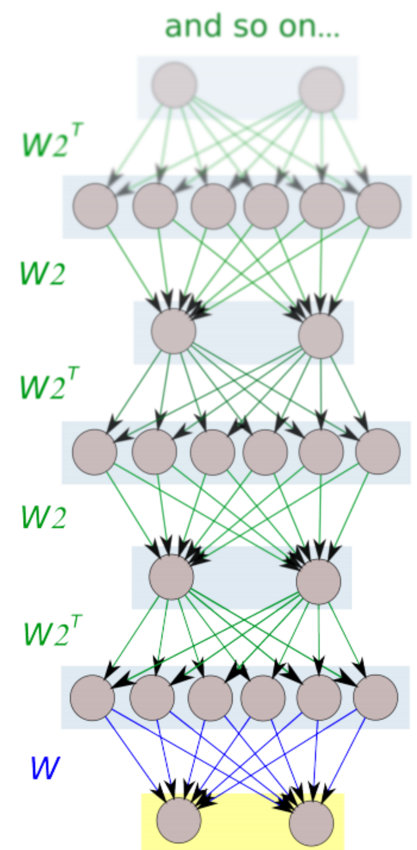
Slide from Marcus Frean, MLSS Tutorial 2010

Deep belief networks: Layer-wise pre-training

Un-tie the weights from layers 2 to infinity

If we freeze the first RBM, and then train another RBM atop it, we are untying the weights of layers 2+ in the ∞ net (which remain tied together).



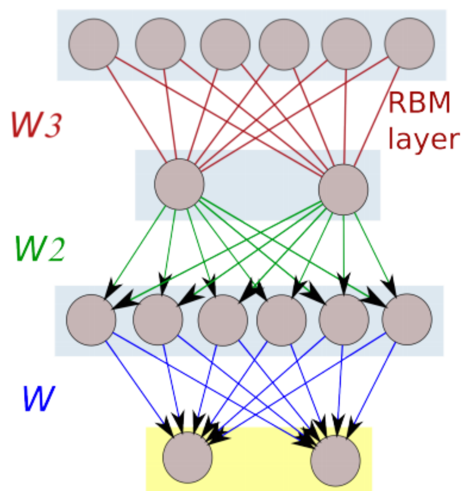


Slide from Marcus Frean, MLSS Tutorial 2010

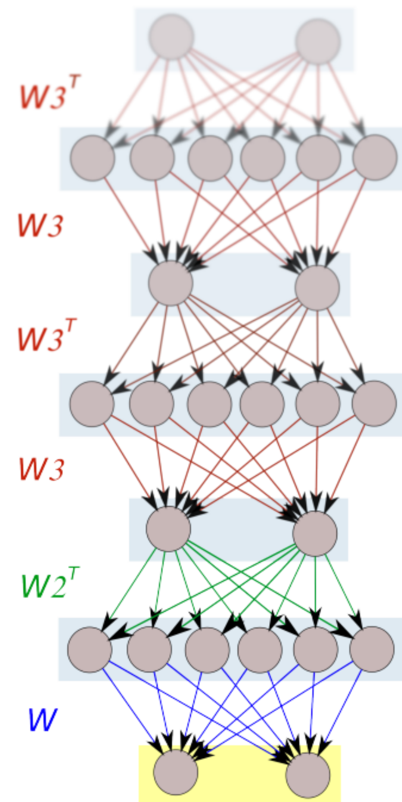
Deep belief networks: Layer-wise pre-training

Un-tie the weights from layers 3 to infinity

and ditto for the 3rd layer...



and so on...



Slide from Marcus Frean, MLSS Tutorial 2010

Deep belief networks: Pre-training and finetuning

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

Figure from (Hinton & Salakhutinov, 2006)

Deep belief networks: Pre-training and finetuning

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

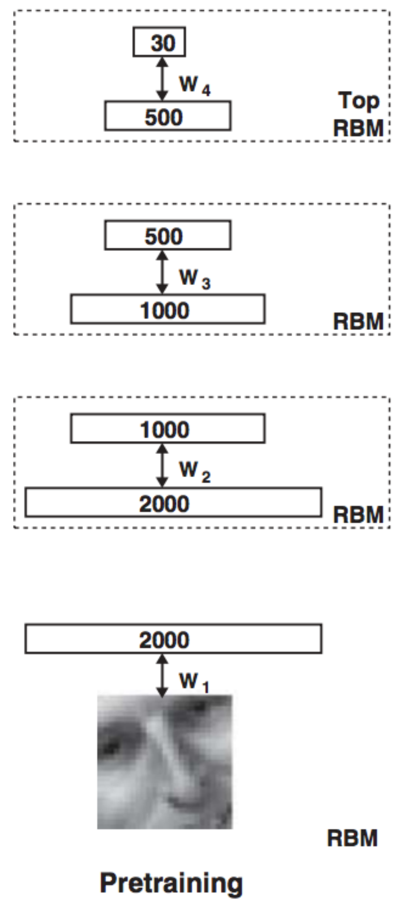


Figure from (Hinton & Salakhutinov, 2006)

Deep belief networks: Pre-training and finetuning

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

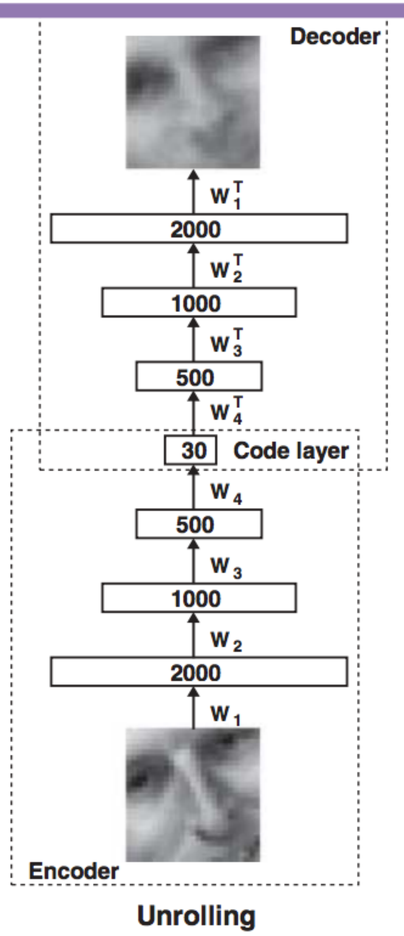


Figure from (Hinton & Salakhutinov, 2006)

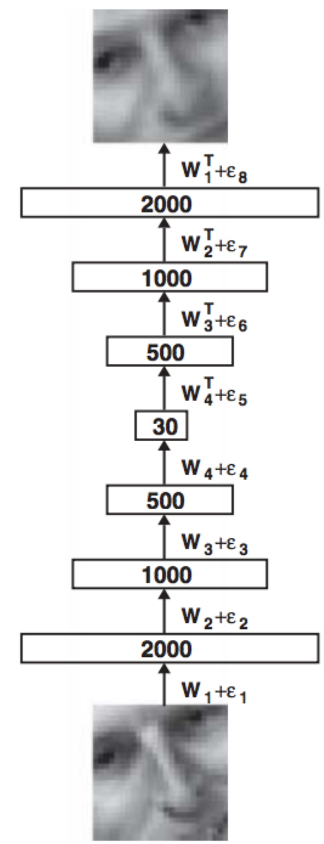
Deep belief networks: Pre-training and finetuning

Setting A: DBN Autoencoder

I. Pre-train a stack of RBMs in greedy layerwise fashion

II. Unroll the RBMs to create an autoencoder (i.e. bottom-up and top-down weights are untied)

III. Fine-tune the parameters using backpropagation

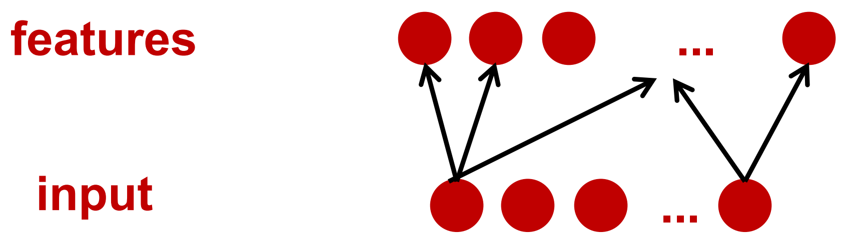


Fine-tuning

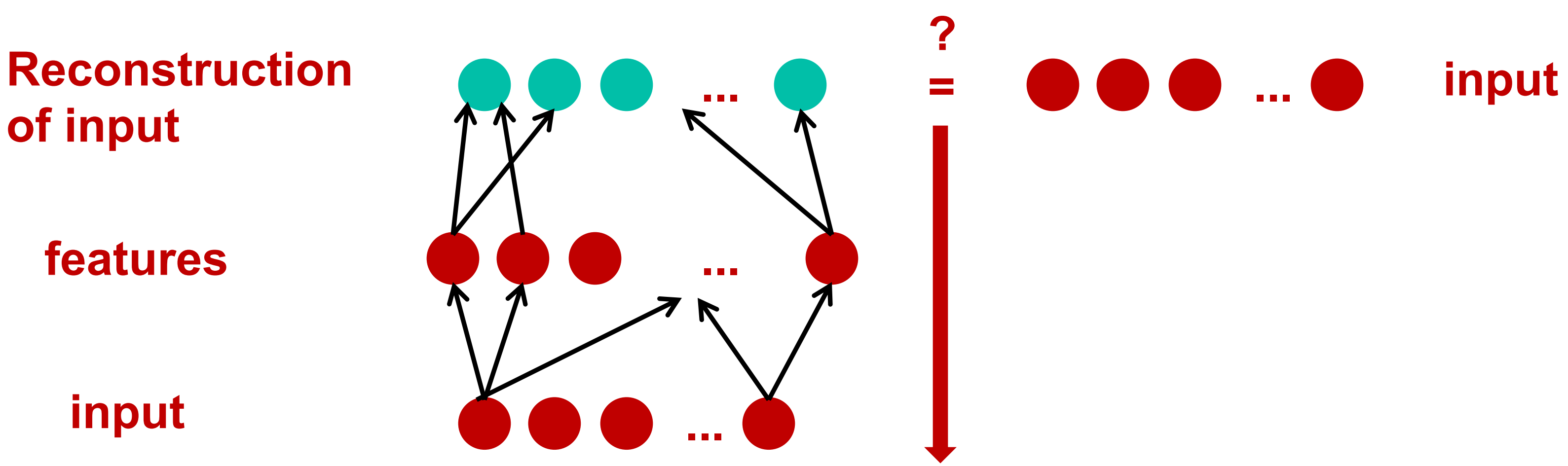
Figure from (Hinton & Salakhutinov, 2006)

Remark:
Layer-wise pre-training for FNN

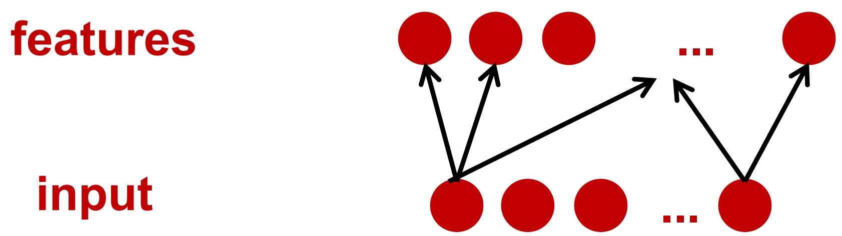
Remark: Layer-wise pre-training for FNN



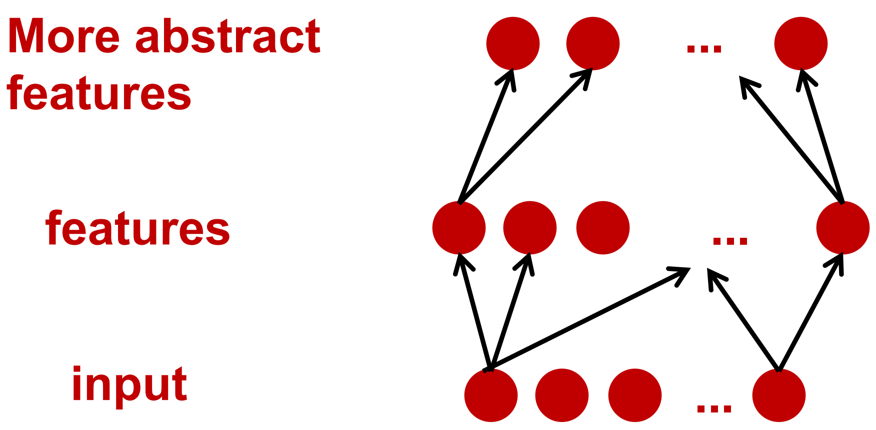
Remark:
Layer-wise pre-training for FNN



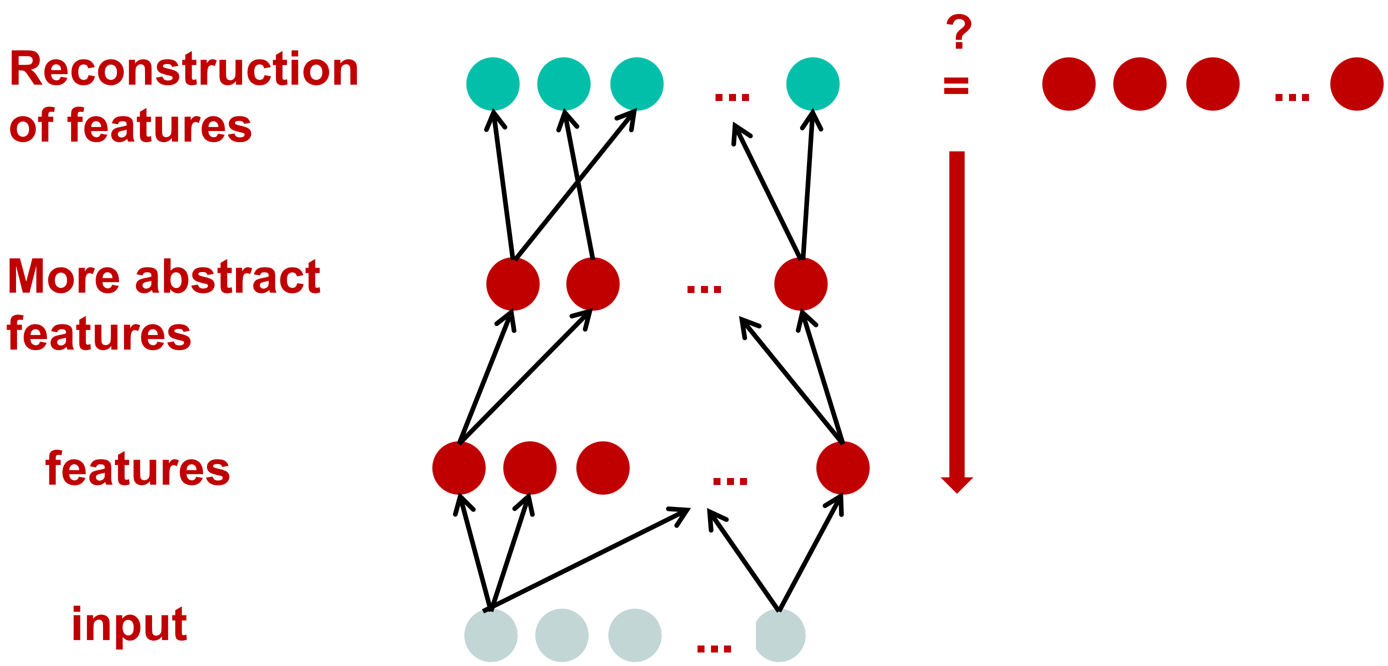
Remark:
Layer-wise pre-training for FNN



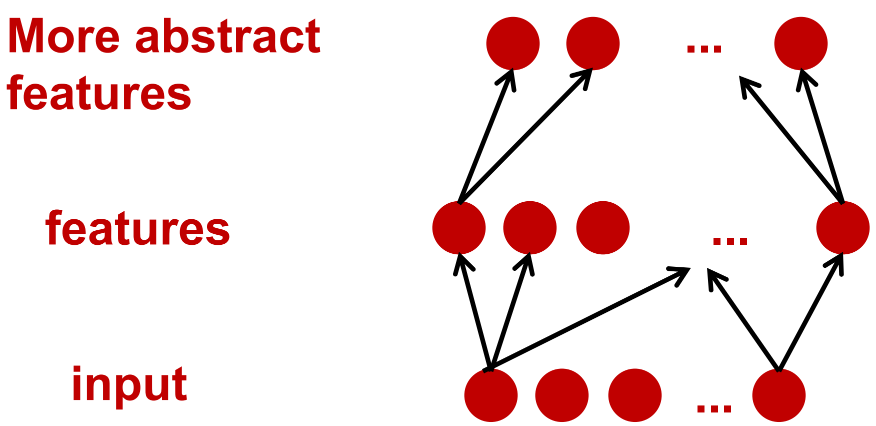
Deep Belief Networks: Layer-wise pre-training



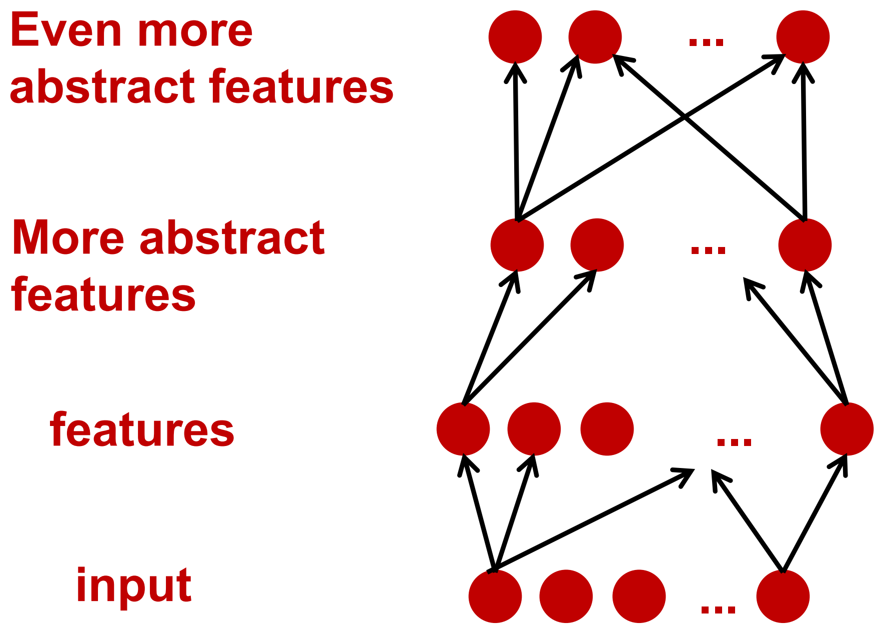
Remark:
Layer-wise pre-training for FNN



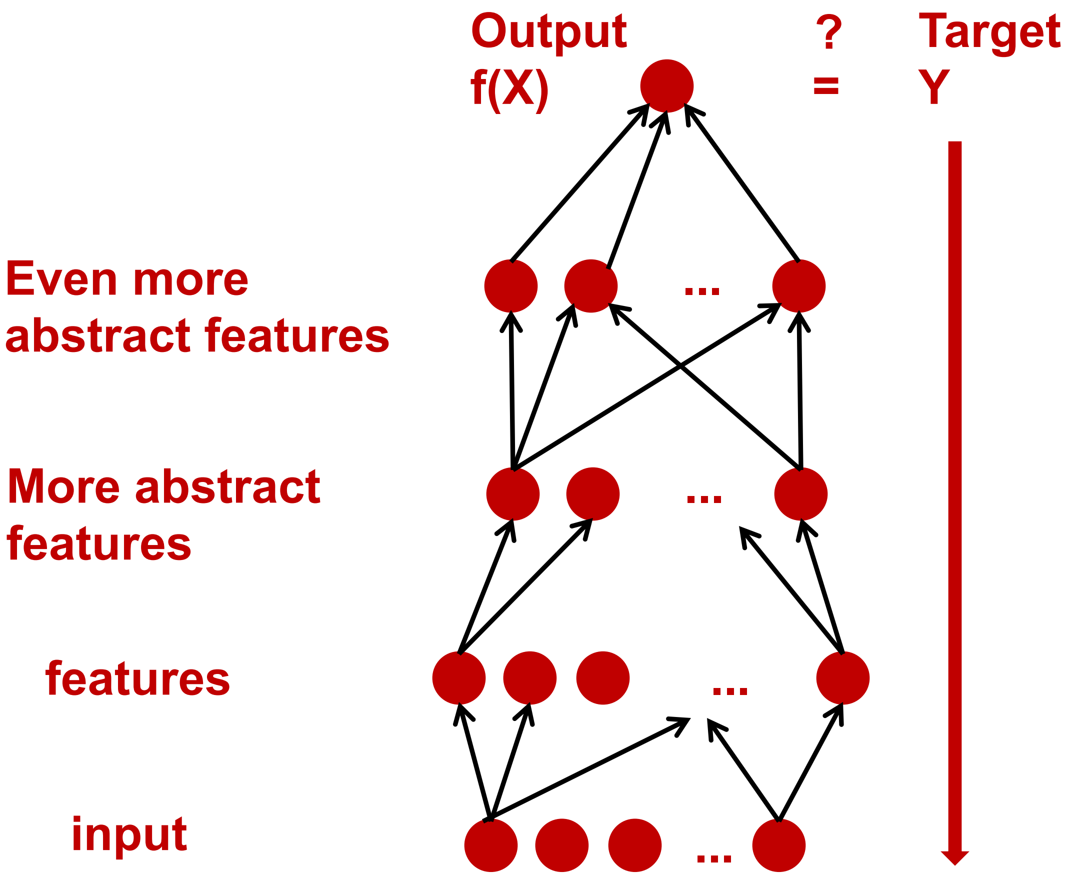
Remark: Layer-wise pre-training for FNN



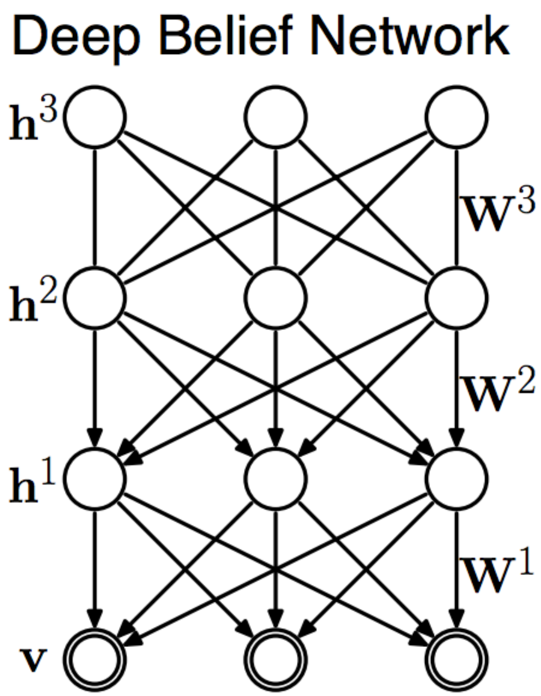
Remark:
Layer-wise pre-training for FNN



Remark:
Layer-wise pre-training for FNN

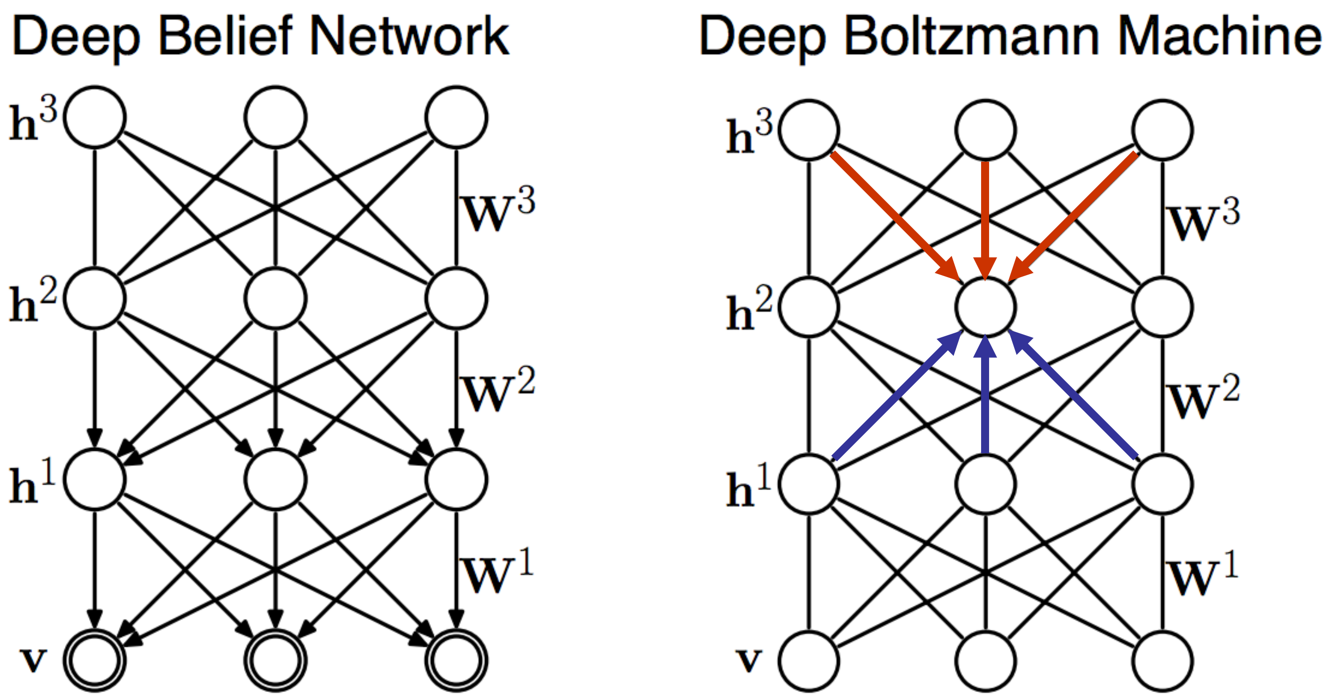


Deep Boltzmann Machines



- DBNs are hybrid graphical models (chain graphs):
  - Inference in DBNs is problematic due to explaining away effect
  - Training: greedy pre-training + ad-hoc fine-tuning; no proper joint training
  - Approximate inference is feed-forward

Deep Boltzmann Machines



- DBNs are hybrid graphical models (chain graphs):
  - Inference in DBNs is problematic due to explaining away effect
  - Training: greedy pre-training + ad-hoc fine-tuning; no proper joint training
  - Approximate inference is feed-forward

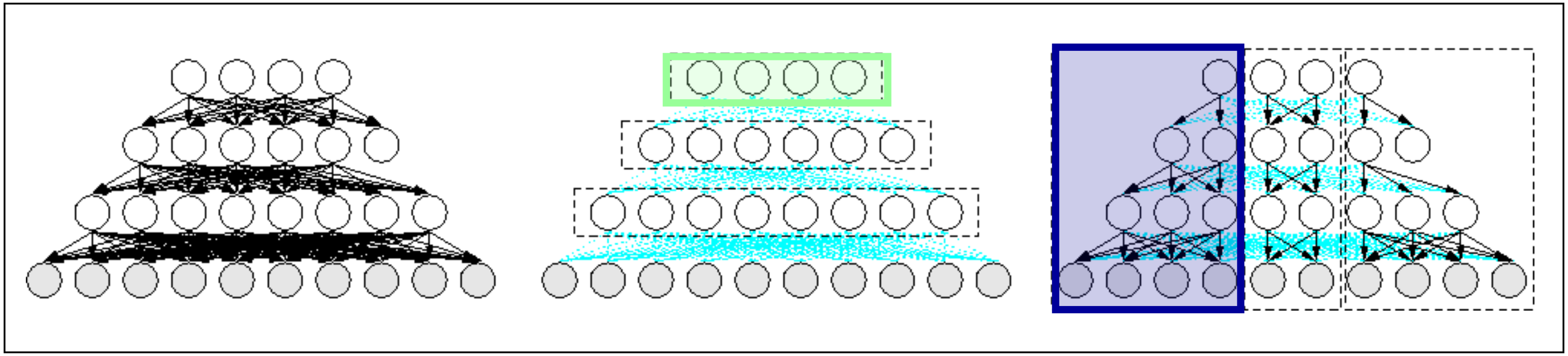
Graphical models vs. Deep nets

- So far: neural nets are flexible function approximators
- Some of the neural nets are in fact proper graphical models (i.e., units/neurons represent proper random variables):
  - Boltzmann machines (Hinton & Sejnowsky, 1983)
  - Restricted Boltzmann machines (Smolensky, 1986)
  - Learning and Inference in sigmoid belief networks (Neal, 1992)
  - Fast learning in deep belief networks (Hinton, Osindero, Teh, 2006)
  - Deep Boltzmann machines (Salakhutdinov and Hinton, 2009)
- Note that in all these models:
  - The primary goal is to represent the distribution of the observables
  - Hidden variables are secondary (auxiliary) elements used to facilitate learning of complex dependencies between the observables
  - The quality of hidden representations is judged by the marginal likelihood
- In contrast, graphical models are often concerned with the correctness of learning and inference of all variables

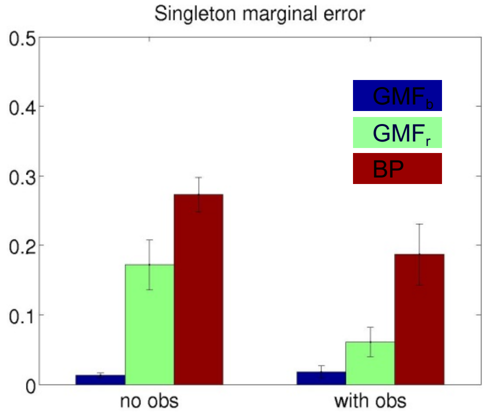
An old study of belief networks from the GM standpoint

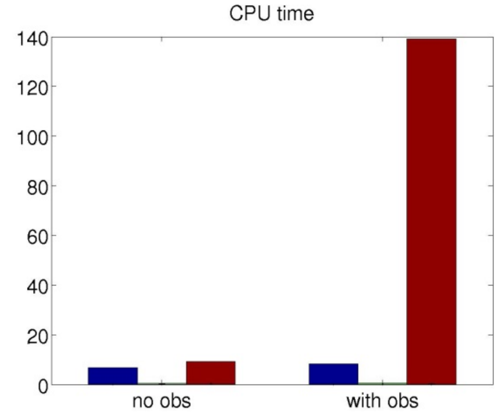
[Xing, Russell, Jordan, UAI 2003]

Mean-field partitions of a sigmoid belief network for subsequent GMF inference



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

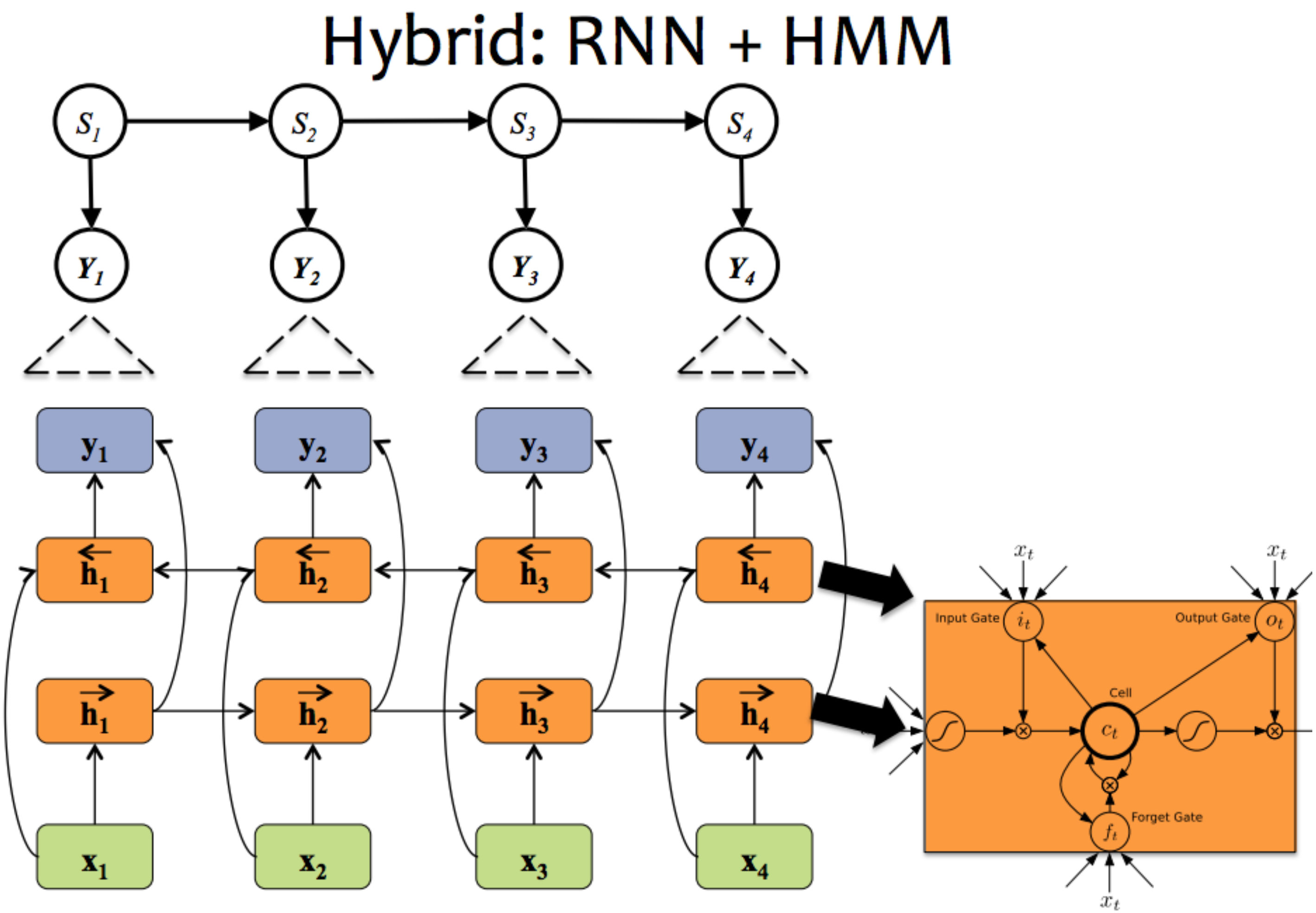




Outline

- An overview of the DL components
  - Historical remarks: early days of neural networks
  - Modern building blocks: units, layers, activations functions, loss functions, etc.
  - Reverse-mode automatic differentiation (aka backpropagation)
  - Distributed representations
- Similarities and differences between GMs and NNs
  - Graphical models vs. computational graphs
  - Sigmoid Belief Networks as graphical models
  - Deep Belief Networks and Boltzmann Machines
- Combining DL methods and GMs
  - Using outputs of NNs as inputs to GMs
  - GMs with potential functions represented by NNs
  - NNs with structured outputs

Combining sequential NNs and GMs



the slide is courtesy of Matt Gormley

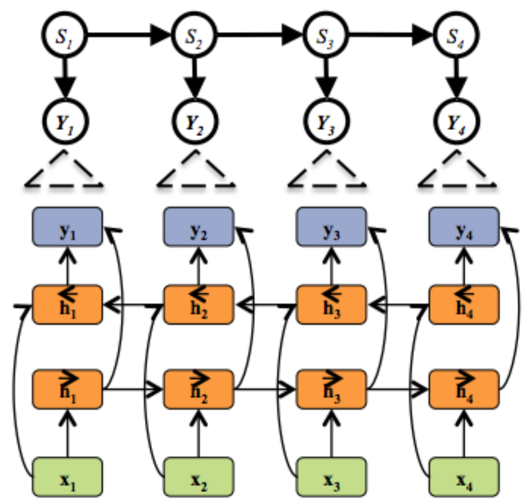
Combining sequential NNs and GMs

Hybrid: RNN + HMM

(Graves et al., 2013)

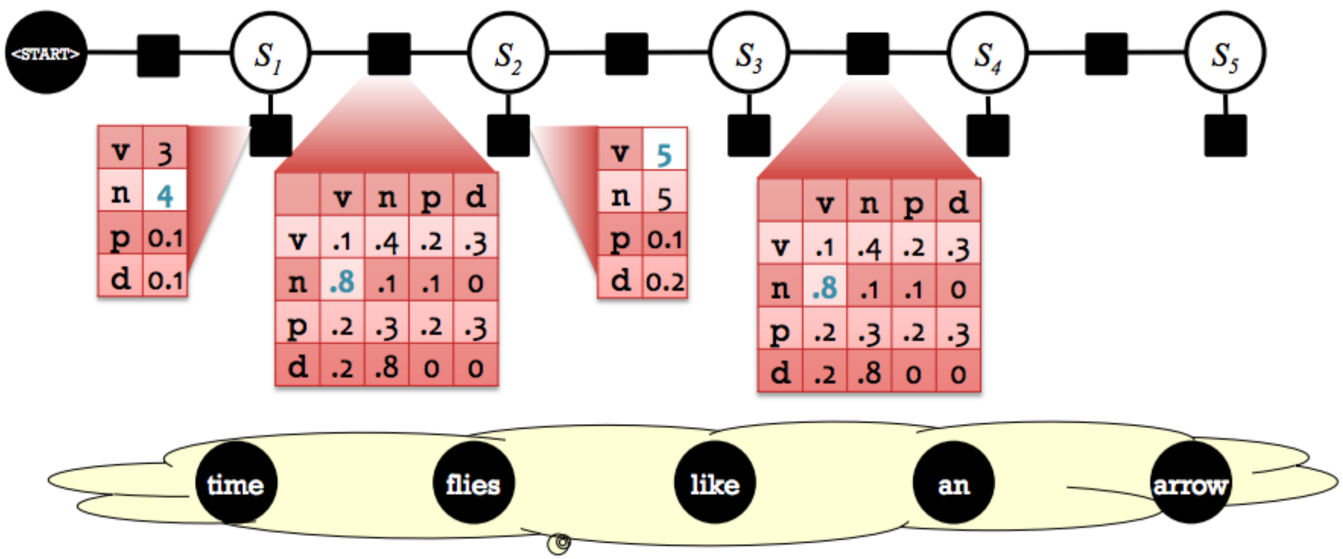
The model, inference, and learning can be **analogous** to our NN + HMM hybrid

- **Objective:** log-likelihood
- **Model:** HMM/Gaussian emissions
- **Inference:** forward-backward algorithm
- **Learning:** SGD with gradient by backpropagation



Hybrid NNs + conditional GMs

Hybrid: Neural Net + CRF



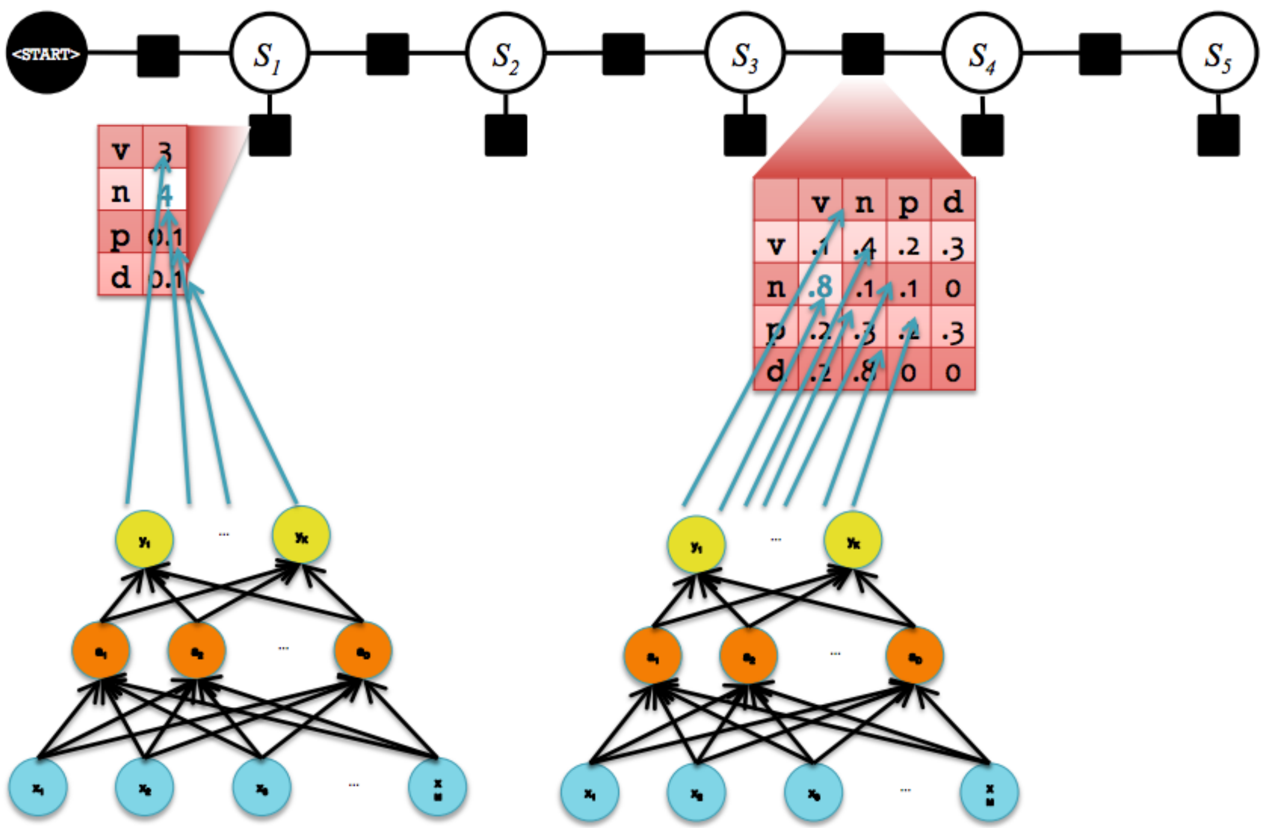
- In a standard CRF, each of the factor cells is a parameter (e.g. transition or emission)
- In the hybrid model, these values are computed by a neural network with its own parameters

the slide is courtesy of Matt Gormley

Hybrid NNs + conditional GMs

Hybrid: Neural Net + CRF

Forward computation



Hybrid NNs + conditional GMs

(Collobert & Weston, 2011)

Hybrid: CNN + CRF

"NN + SLL"
- Model: Convolutional Neural Network (CNN) with **linear-chain CRF**
- Training objective: maximize **sentence-level likelihood** (SLL)

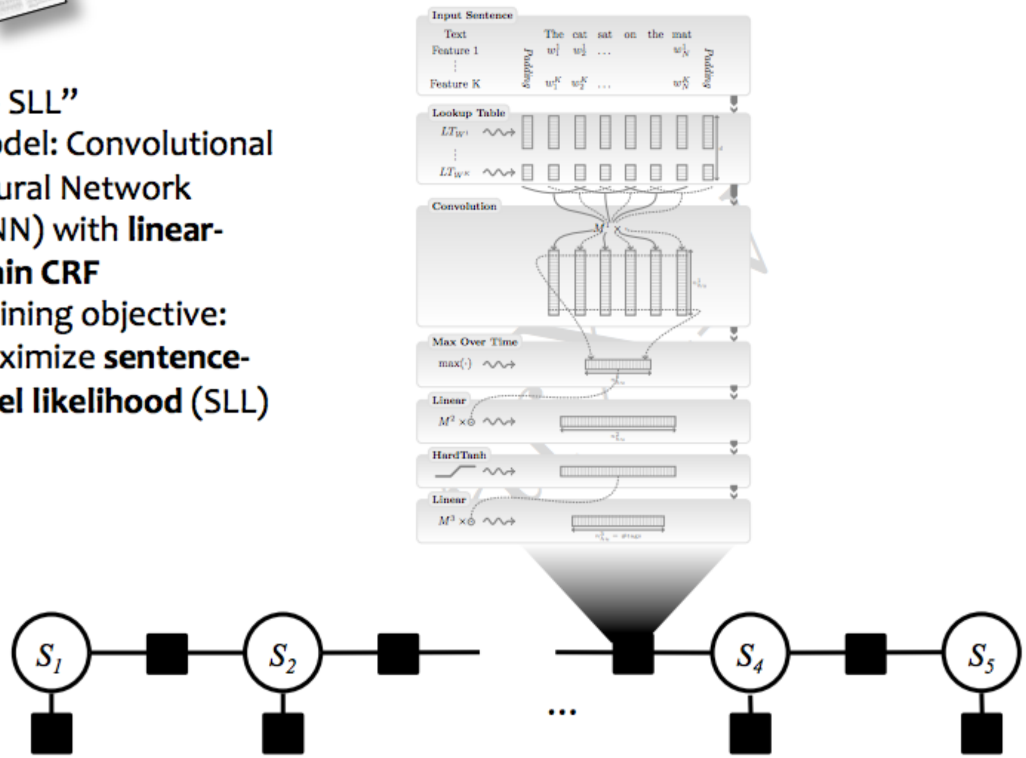


Figure from (Collobert & Weston, 2011)

Dealing with structured prediction

- Energy-based modeling of the structured output (CRF)

$$\mathbf{y}^*(\mathbf{x}; \mathbf{w}) := \arg\min_{\mathbf{y}} E(\mathbf{y}, \mathbf{x}; \mathbf{w})$$

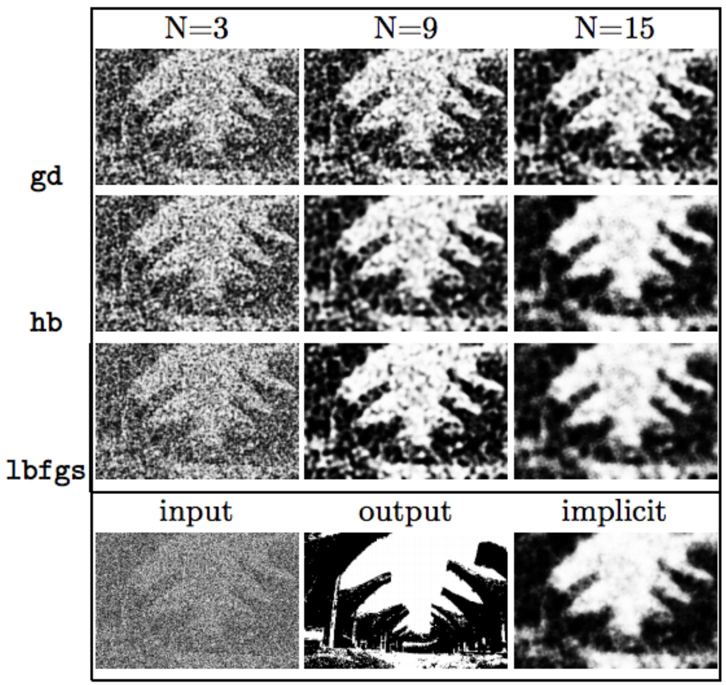
- Unroll the optimization algorithm for a fixed number of steps (Domke, 2012)

$$\mathbf{y}^*(\mathbf{x}; \mathbf{w}) = \operatorname*{opt\text{-}alg}_{\mathbf{y}} E(\mathbf{y}, \mathbf{x}; \mathbf{w})$$

- Now, y* is some non-linear differentiable function of the inputs and weights → impose some loss and optimize it as the standard computation graph using backprop!
- Similarly, message passing based inference algorithms can be truncated and converted into computational graphs (Domke, 2011; Stoyanov et al., 2011)

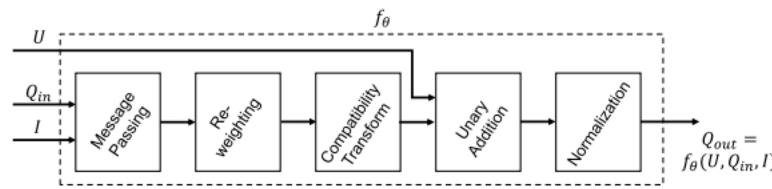
Dealing with structured prediction

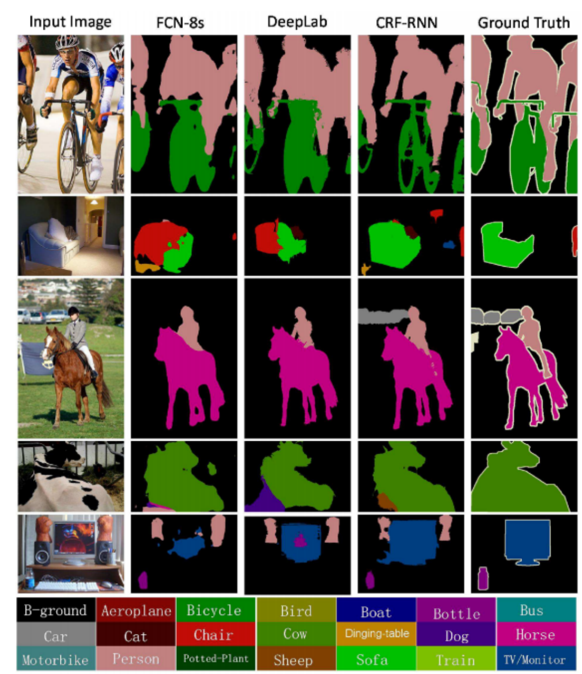
(Domke, 2012)



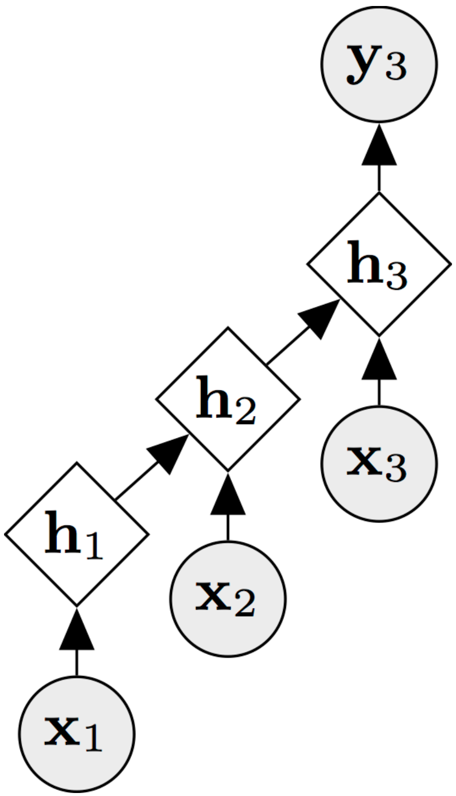
(Zheng et al., CVPR 2015)

CRFs as recurrent neural networks

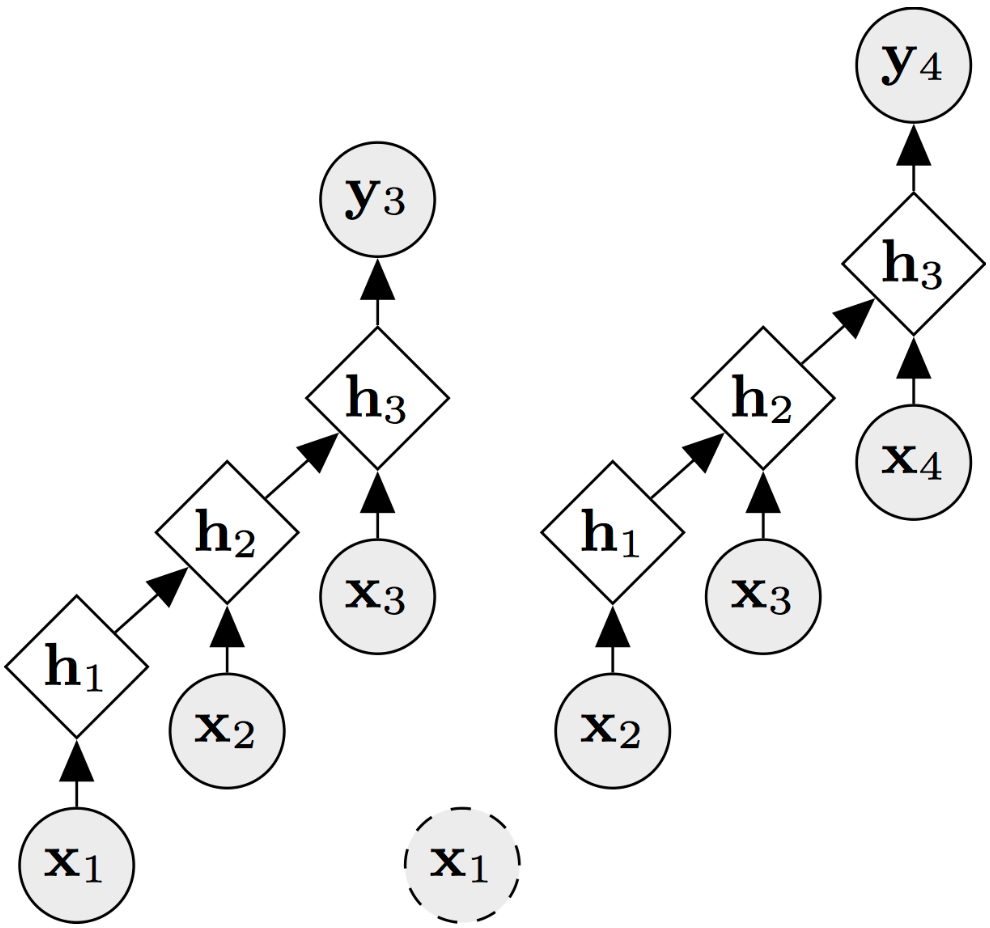




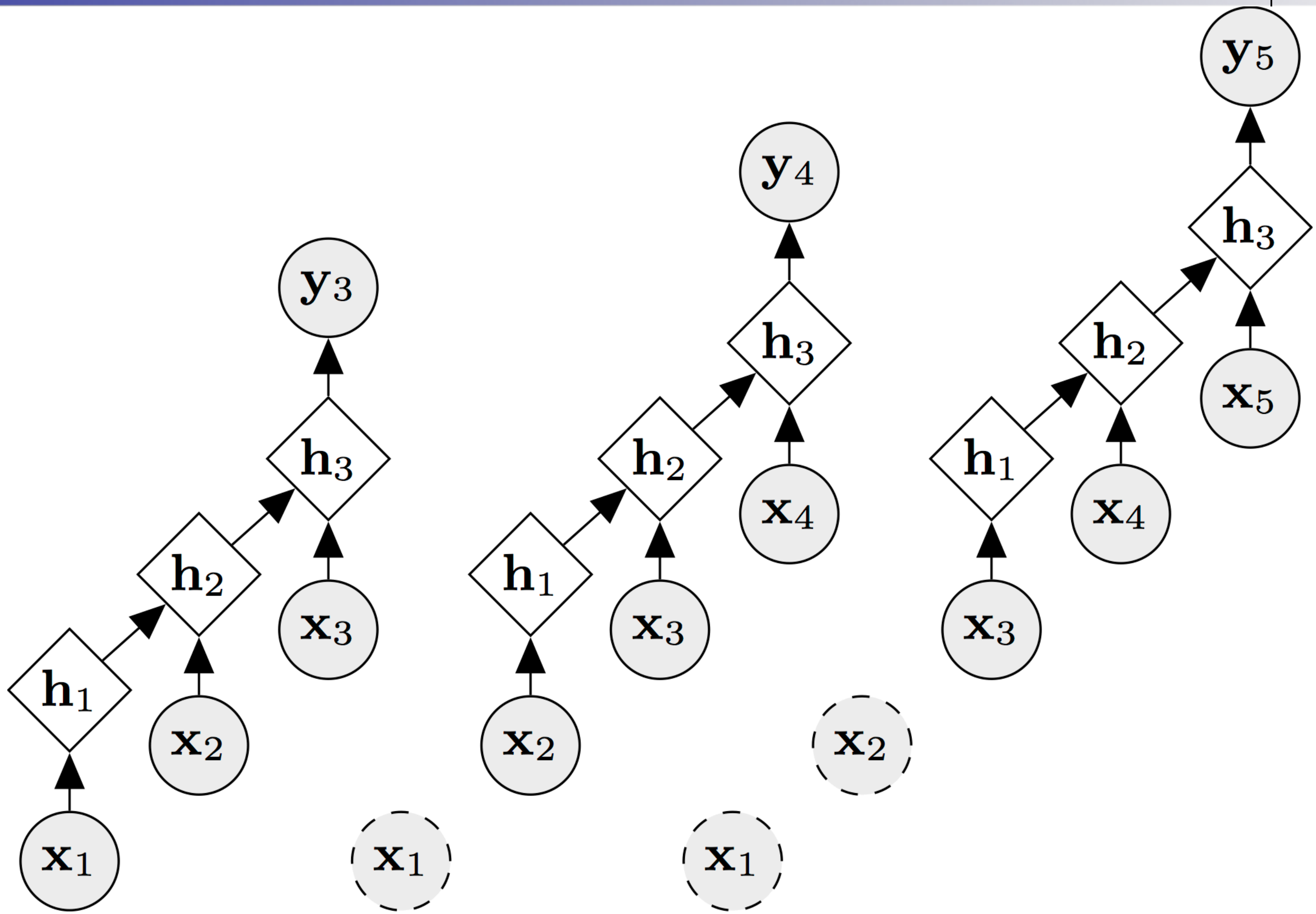
NNs + nonparametric GMs



NNs + nonparametric GMs

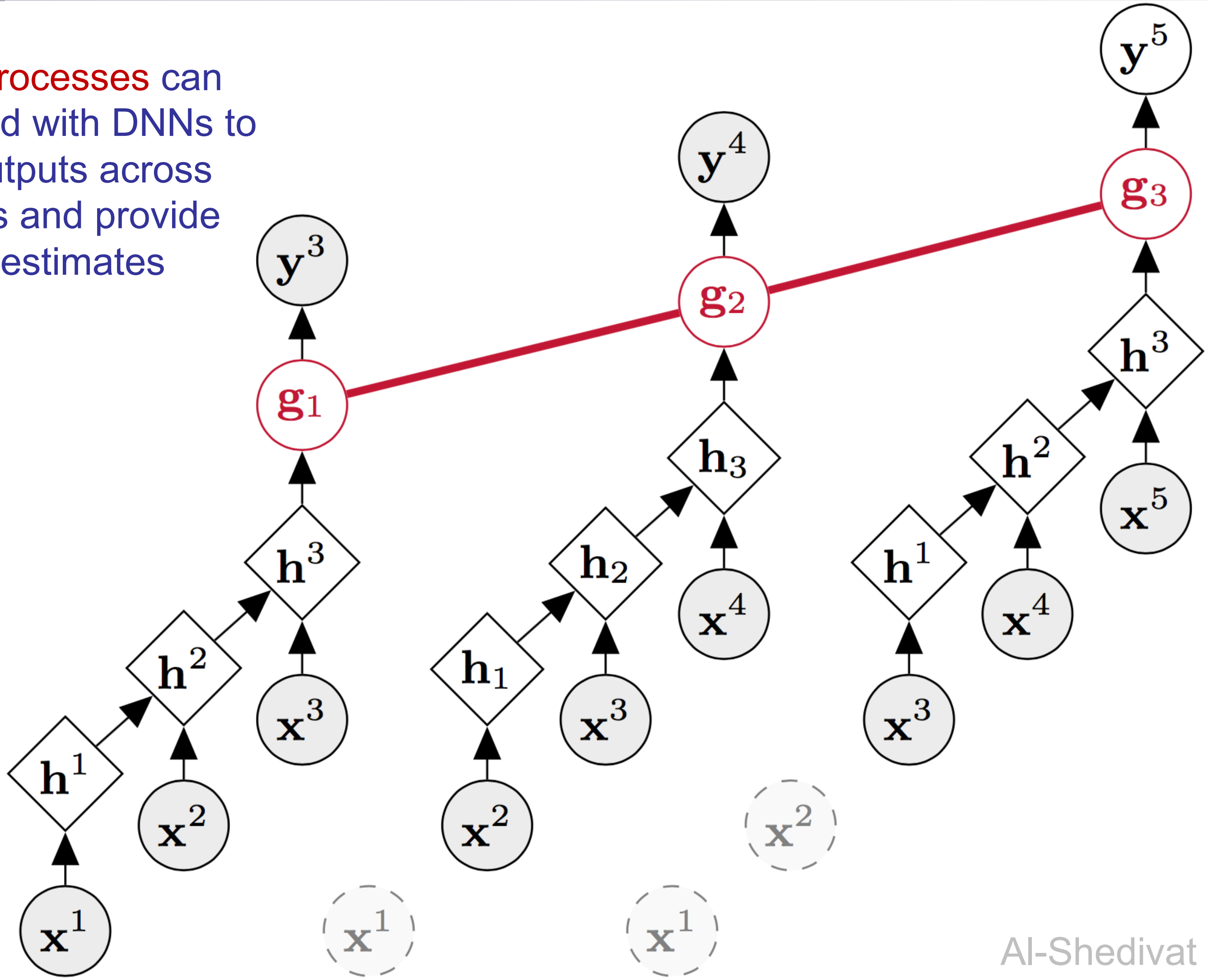


NNs + nonparametric GMs



NNs + nonparametric GMs

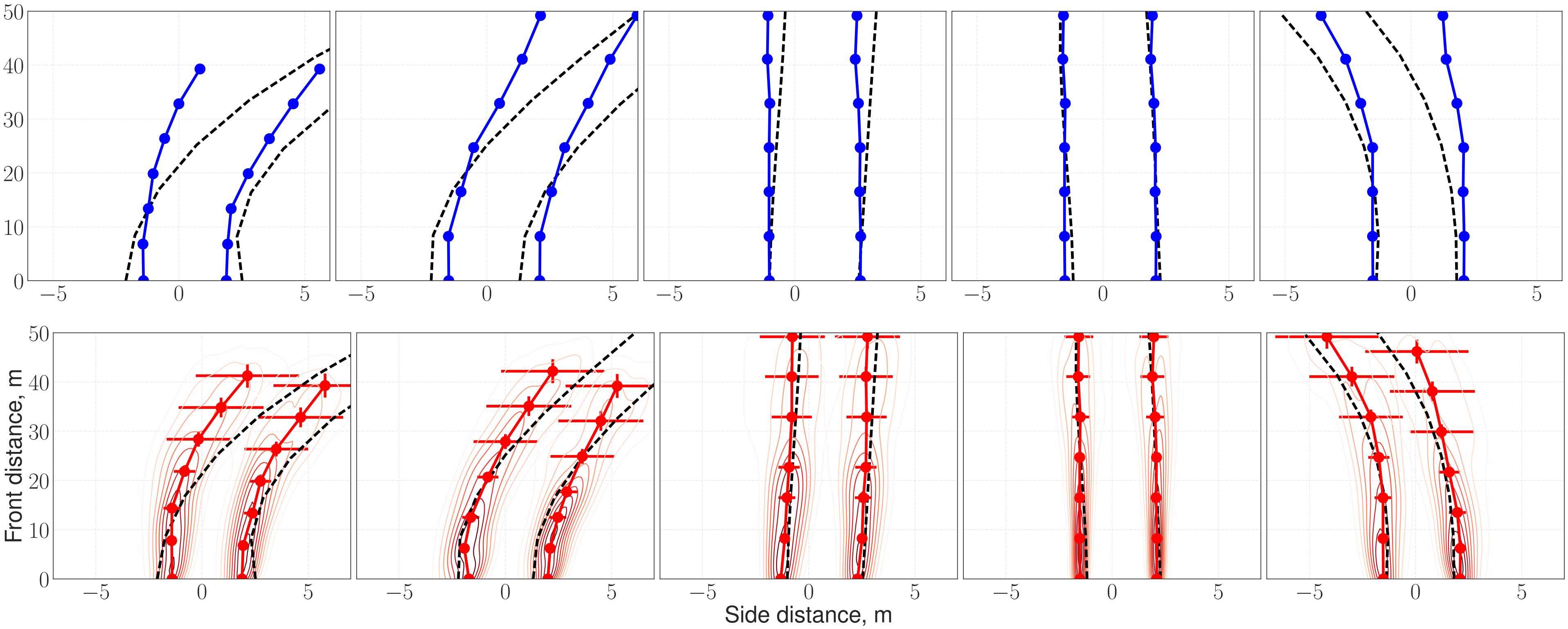
Gaussian processes can be combined with DNNs to correlate outputs across the samples and provide uncertainty estimates



Al-Shedivat et al., 2016

NNs + nonparametric GMs

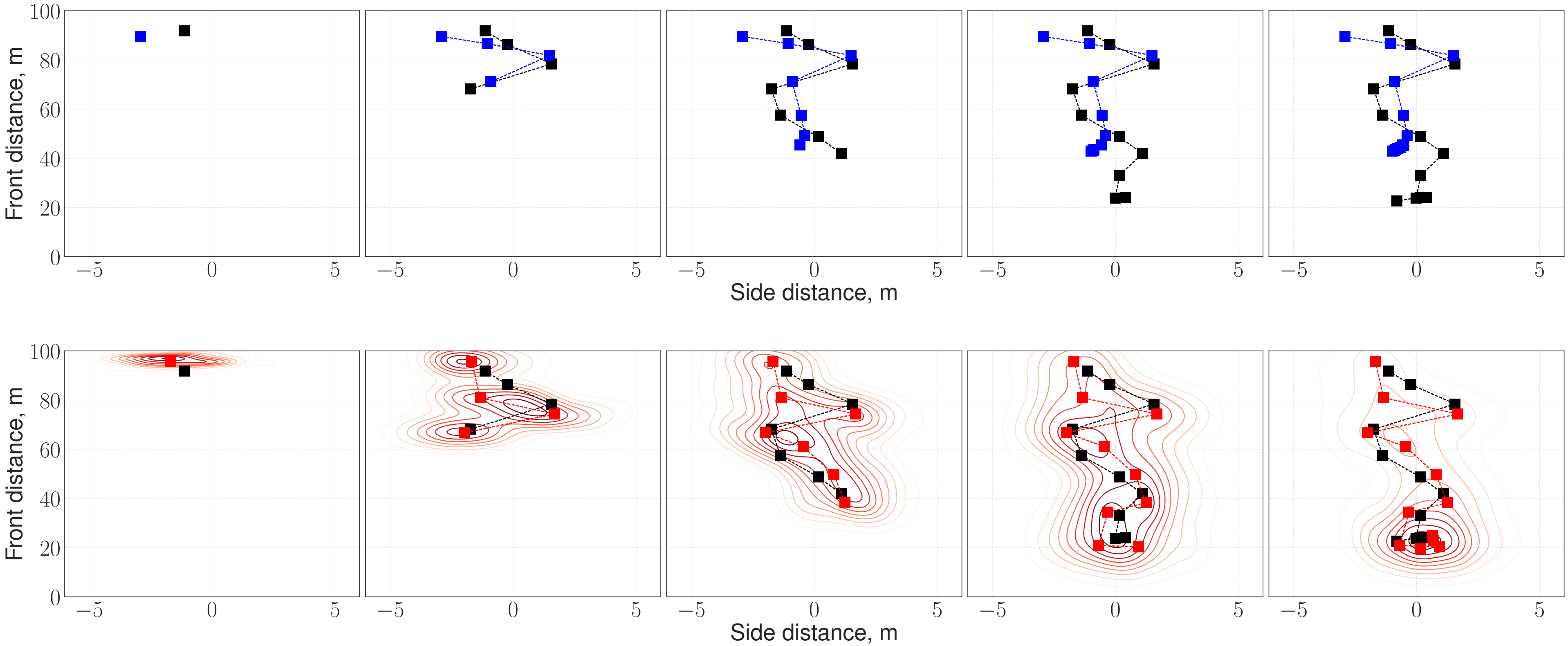
Lane prediction: LSTM vs GP-LSTM



Al-Shedivat et al., 2016

NNs + nonparametric GMs

Lead vehicle prediction: LSTM vs GP-LSTM



Al-Shedivat et al., 2016

Conclusion

- DL & GM: the fields are similar in the beginning (structure, energy, etc.), and then diverge to their own signature pipelines
- DL: most effort is directed to comparing different architectures and their components (based on empirical performance on a downstream task)
  - DL models are good at learning robust hierarchical representations from the data and suitable for simple reasoning (“low-level cognition”)
- GM: lots of efforts are directed to improving inference accuracy and convergence speed
  - GMs are best for provably correct inference and suitable for high-level complex reasoning tasks (“high-level cognition”)
- Convergence of both fields is very promising!