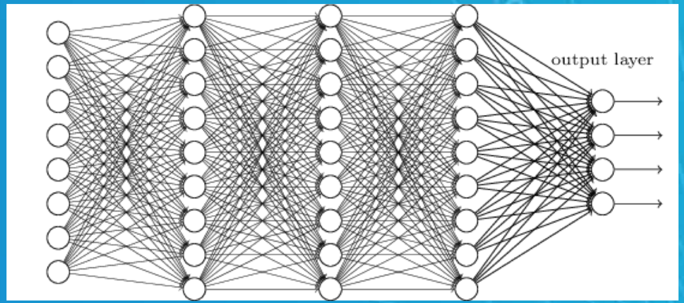
PETUUM

Carnegie Mellon University
School of Computer Science

Probabilistic Graphical Models

Statistical and Algorithmic Foundations of Deep Learning

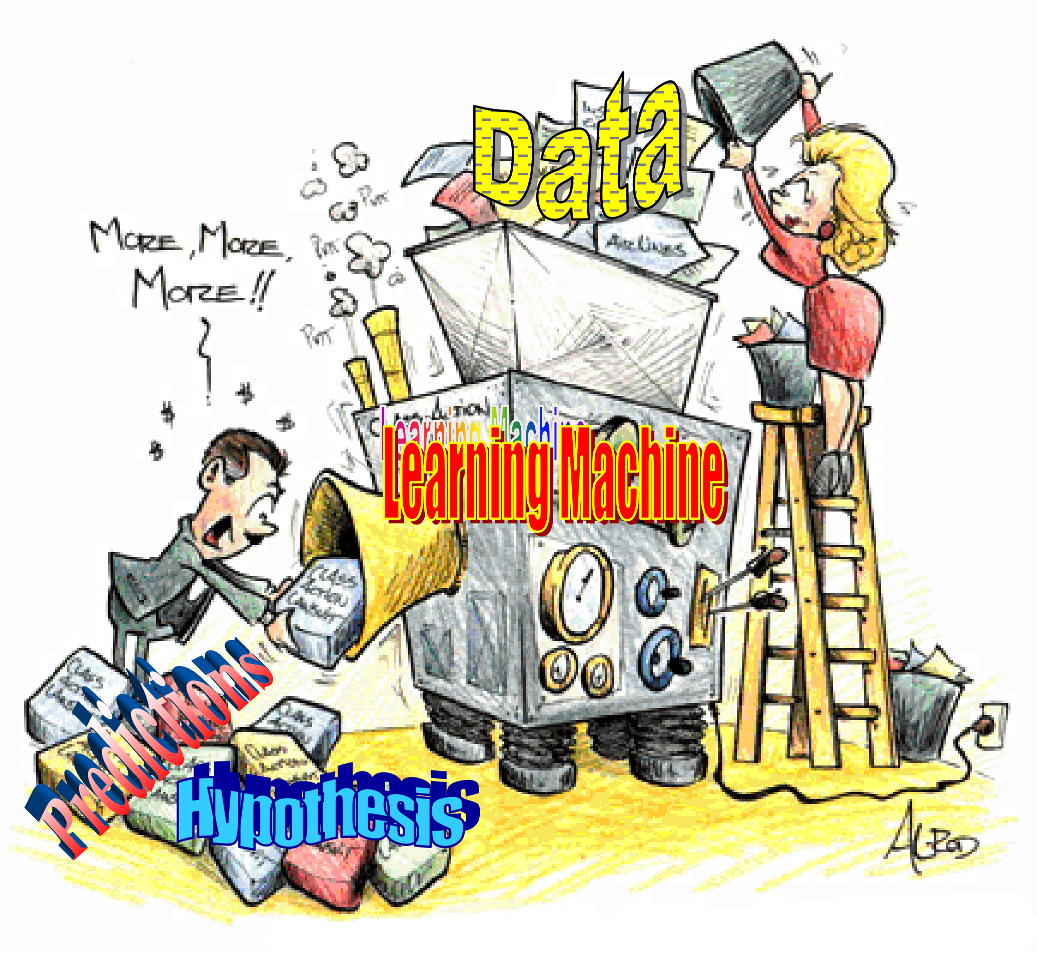


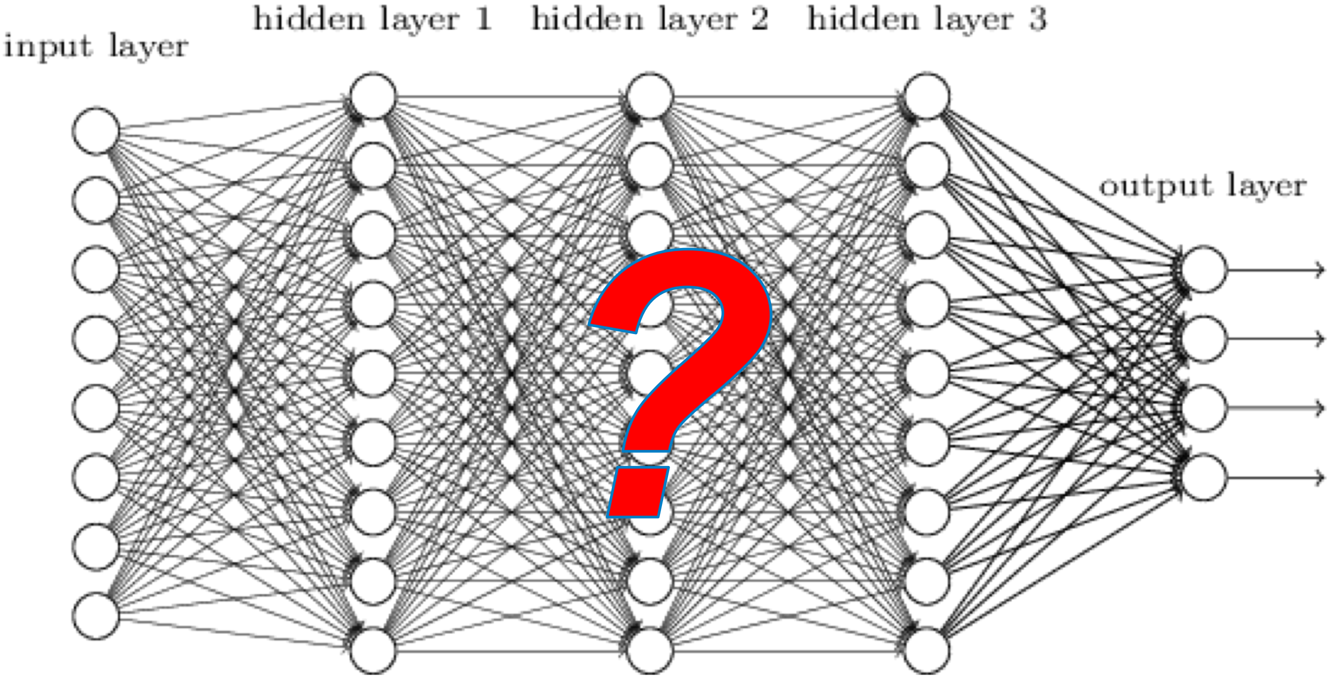
Eric Xing

Lecture 11, February 19, 2020

Reading: see class homepage

ML vs DL


Data
More, More, More!!
Learning Machine
Predictions
Hypothesis


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer
?

Outline

- An overview of 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)

- 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

- Bayesian Learning of NNs
  - Bayesian learning of NN parameters
  - Deep kernel learning

Outline

- An overview of 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)

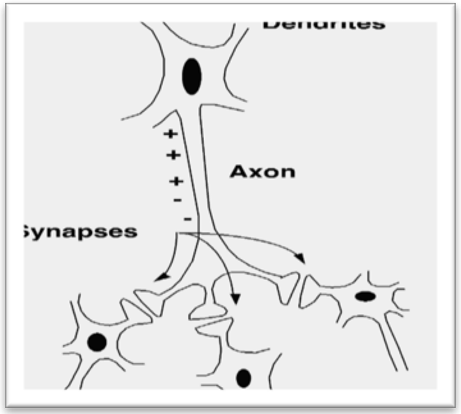
- Similarities and differences between GMs and NNs
  - Graphical models vs. computational graphs
  - Sigmoid Belief Networks as graphical models
  - Deep Belief Networks and Boltzmann Machines

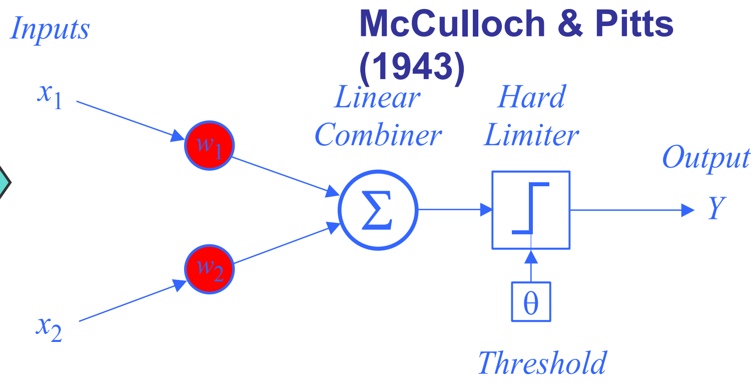
- Combining DL methods and GMs
  - Using outputs of NNs as inputs to GMs
  - GMs with potential functions represented by NNs
  - NNs with structured outputs

- Bayesian Learning of NNs
  - Bayesian learning of NN parameters
  - Deep kernel learning

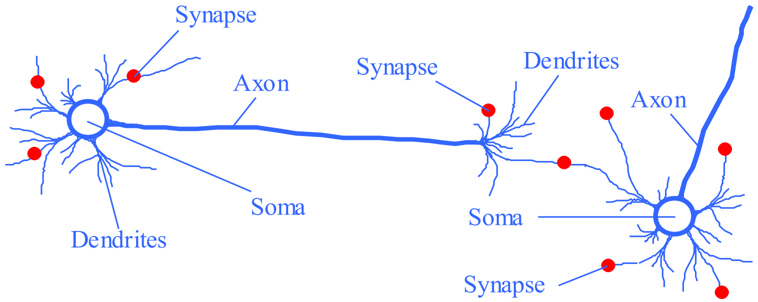
Perceptron and Neural Nets

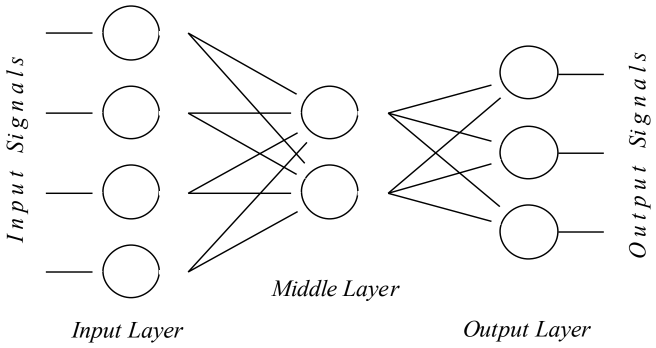
- From biological neuron to artificial neuron (perceptron)



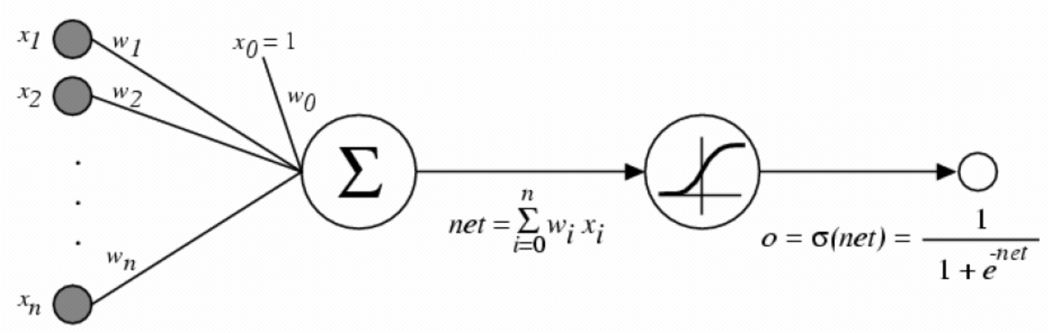

McCulloch & Pitts (1943)

- From biological neuron network to artificial neuron networks





The perceptron learning algorithm


x_1
w_1
$x_0 = 1$
w_0
x_2
w_2
w_n
x_n
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

- Recall the nice property of sigmoid function
- Consider regression problem f: X→Y, for scalar `

$$\frac{d\sigma}{dt} = \sigma(1-\sigma)$$

- We used to maximize the conditional data likelihood

$$y = f(x) + \epsilon$$

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

- 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

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

x_d = input

t_d = target output

o_d = observed output

w_i = weight i

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:

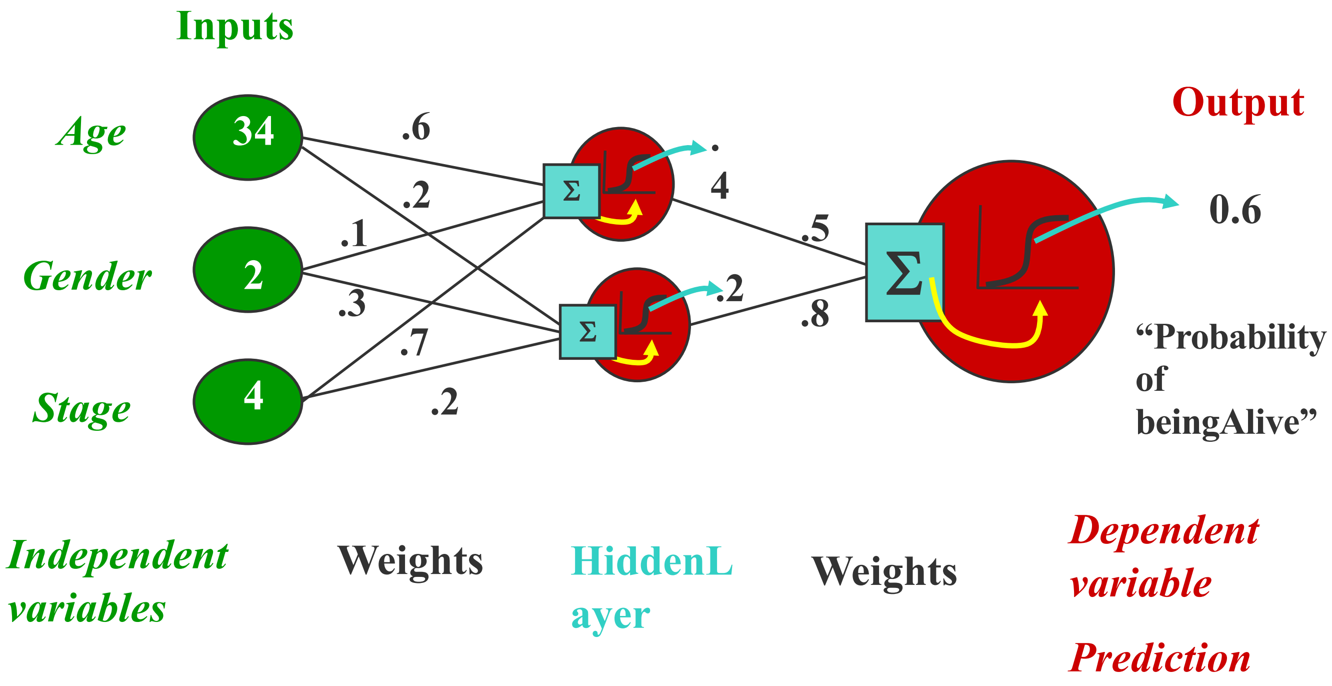
- For each training example *d* in *D*

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

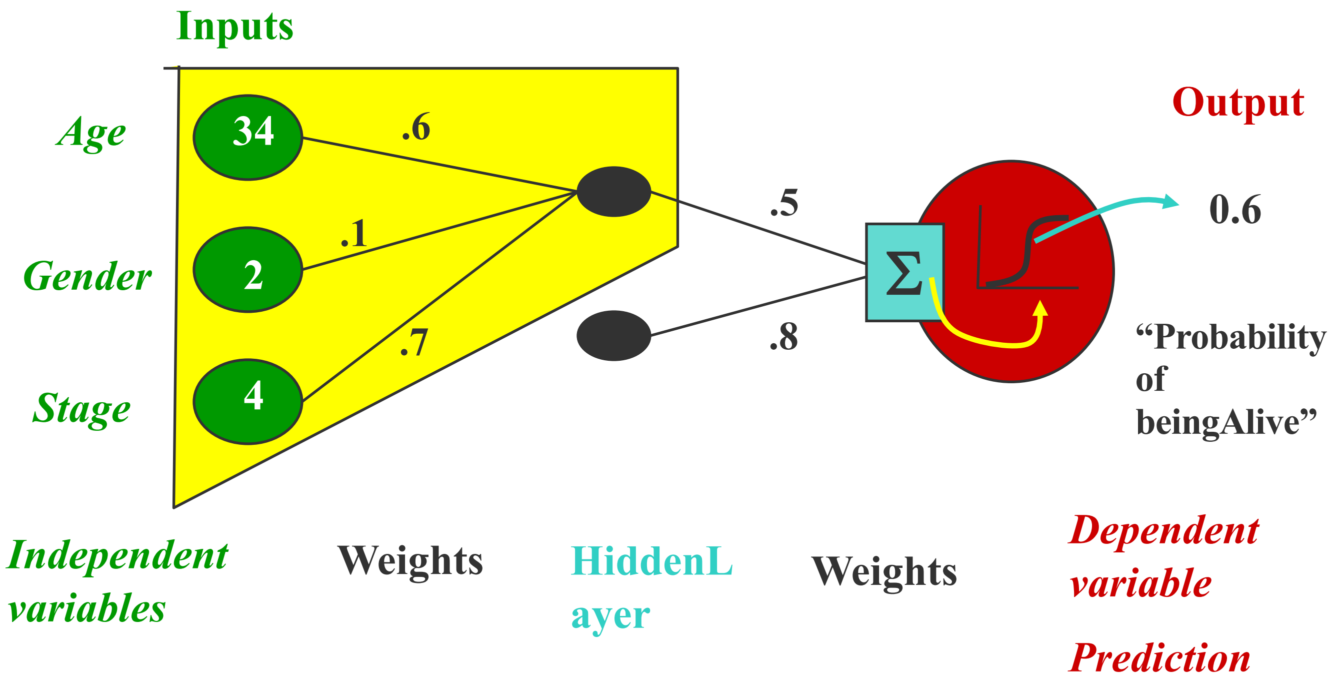
  where

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

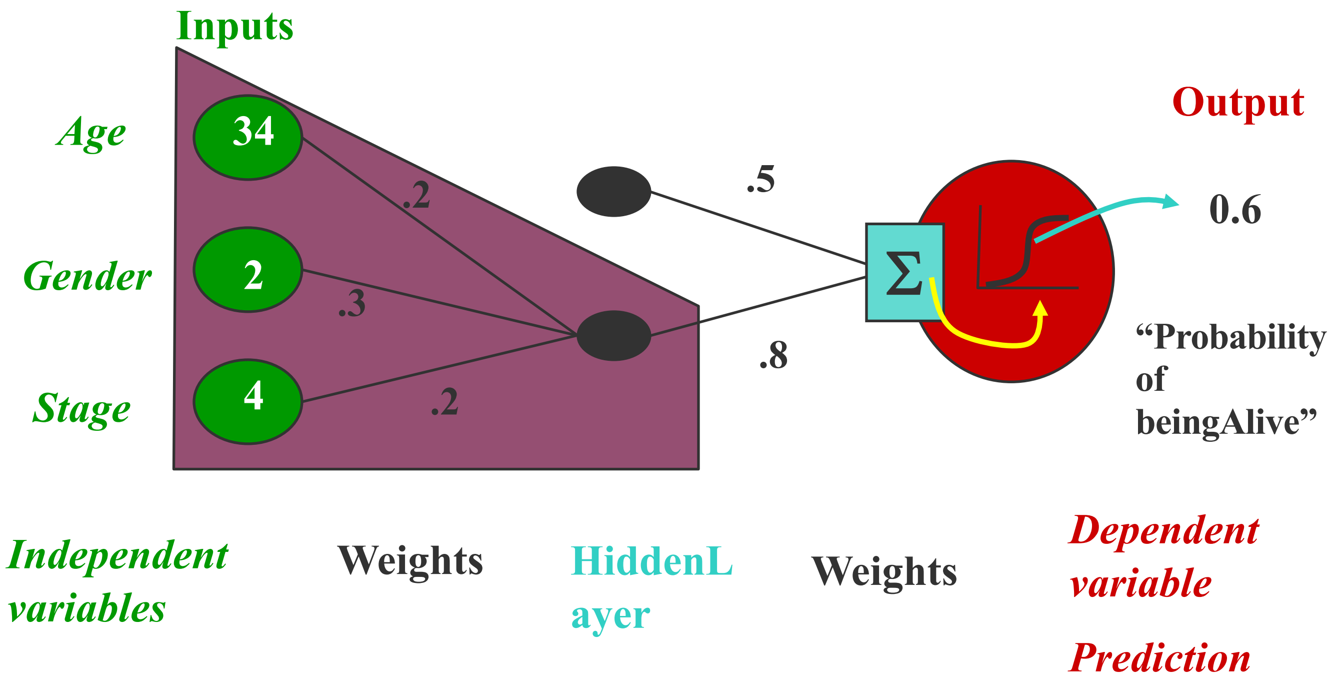
Neural Network Model



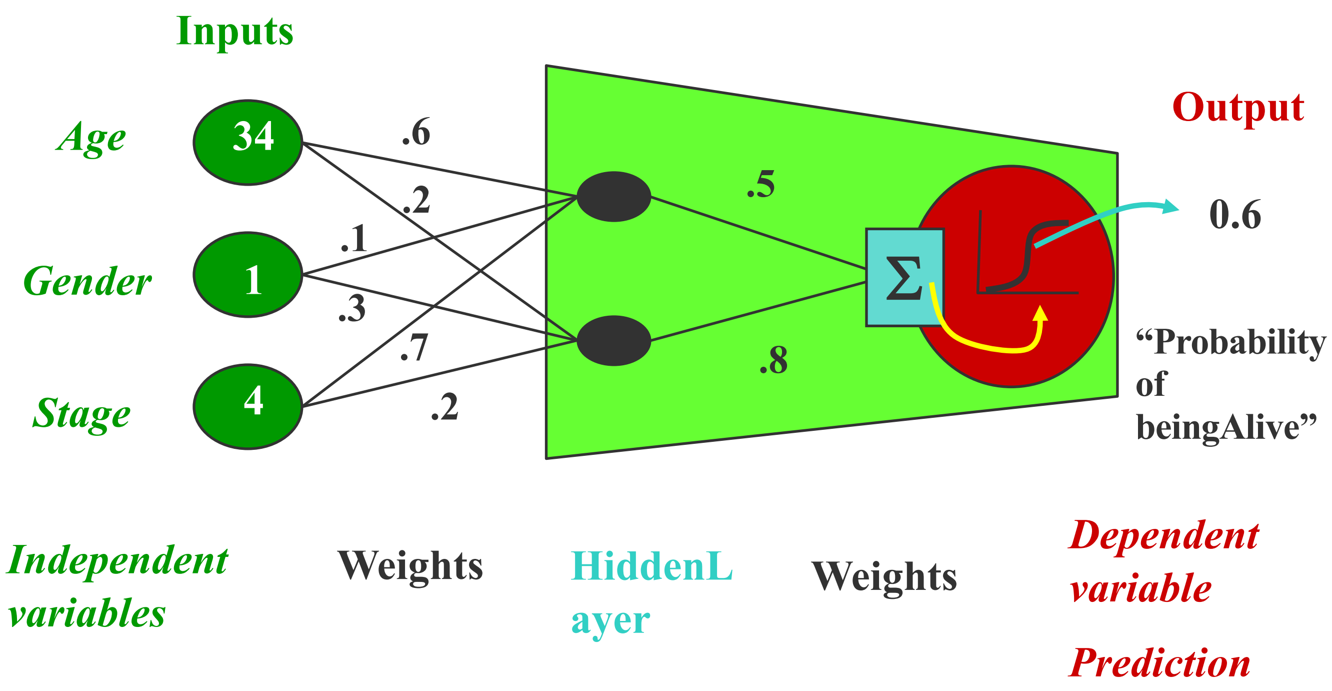
“Combined logistic models”



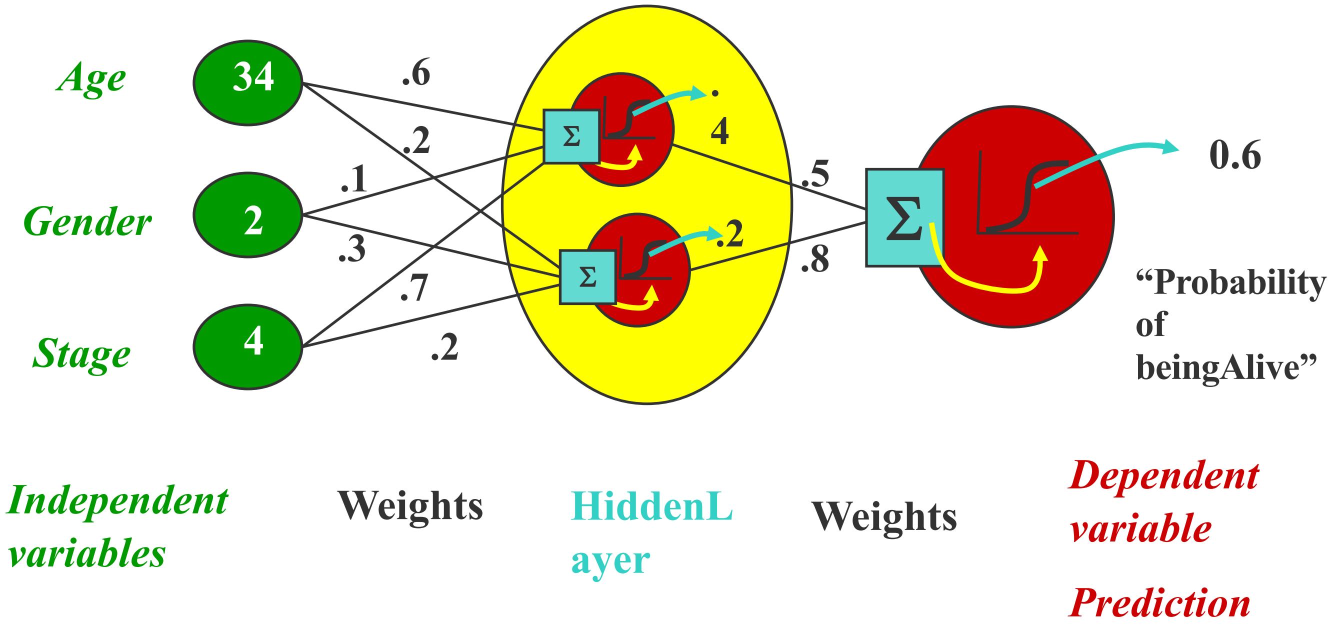
“Combined logistic models”



“Combined logistic models”

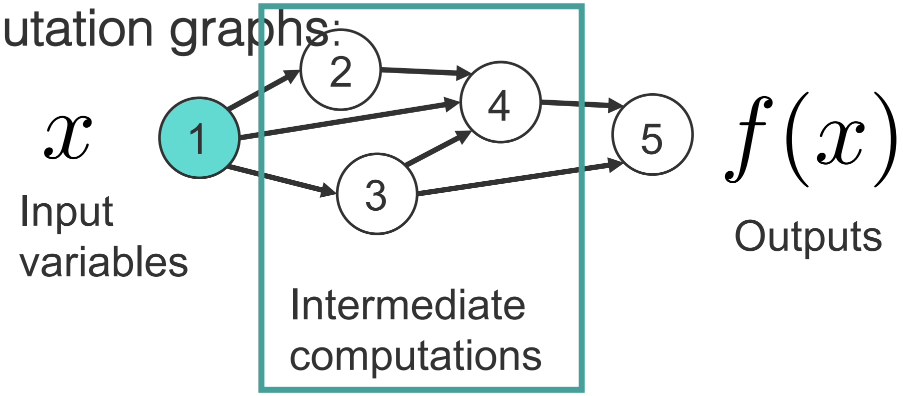


Not really, no target for hidden units...



Backpropagation: Reverse-mode differentiation

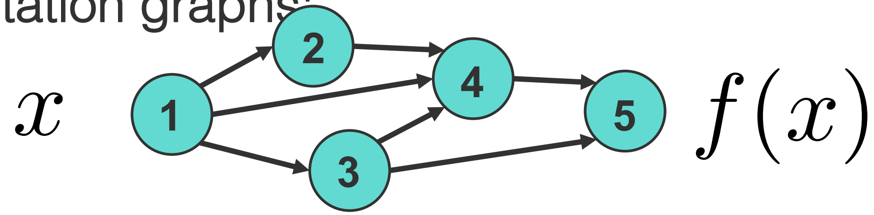
- Artificial neural networks are nothing more than complex functional compositions that can be represented by computation graphs:



$$\frac{\partial f_n}{\partial x} =$$

Backpropagation: Reverse-mode differentiation

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



$$\frac{\partial f_n}{\partial x} =$$

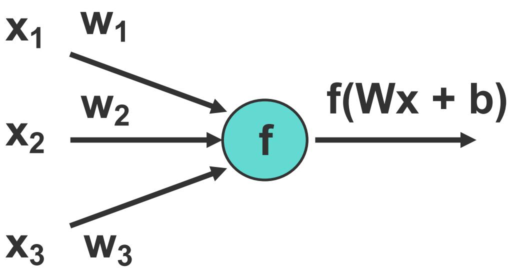
- By applying the chain rule and using reverse accumulation, we get

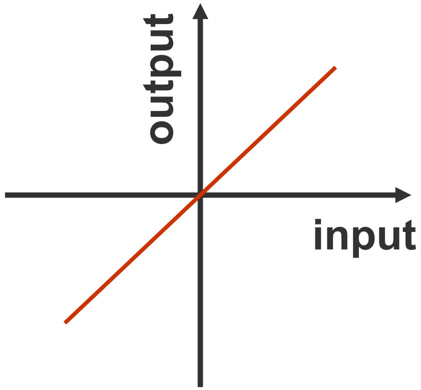
$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \frac{\partial f_{i_1}}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \sum_{i_2 \in \pi(i_1)} \frac{\partial f_{i_1}}{\partial f_{i_2}} \frac{\partial f_{i_1}}{\partial x} = \ldots$$

- 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 part)
- Modern packages can do this *automatically* (more later)

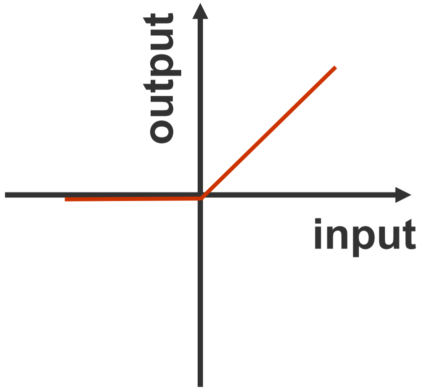
Modern building blocks of deep networks

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





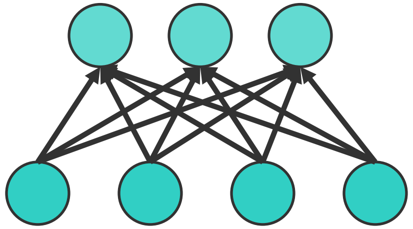
Linear



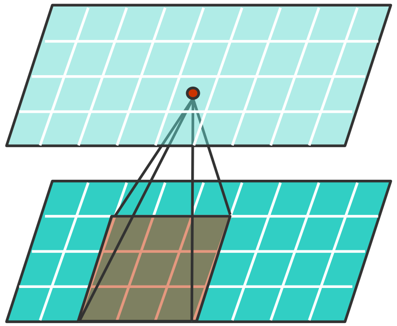
Rectified linear (ReLU)

Modern building blocks of deep networks

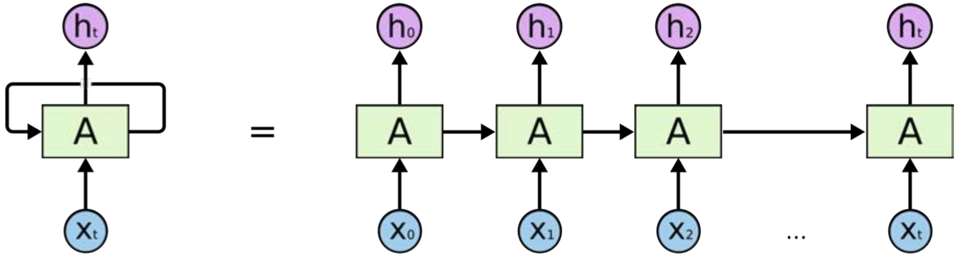
- Activation functions
  - Linear and ReLU
  - Sigmoid and tanh
  - Etc.
- Layers
  - Fully connected
  - Convolutional & pooling
  - Recurrent
  - ResNets
  - Etc.



fully connected

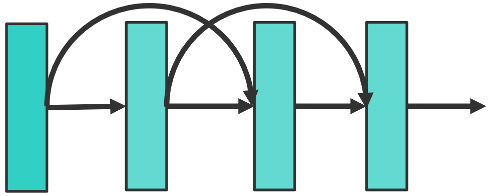


convolutional



recurrent

source: colah.github.io

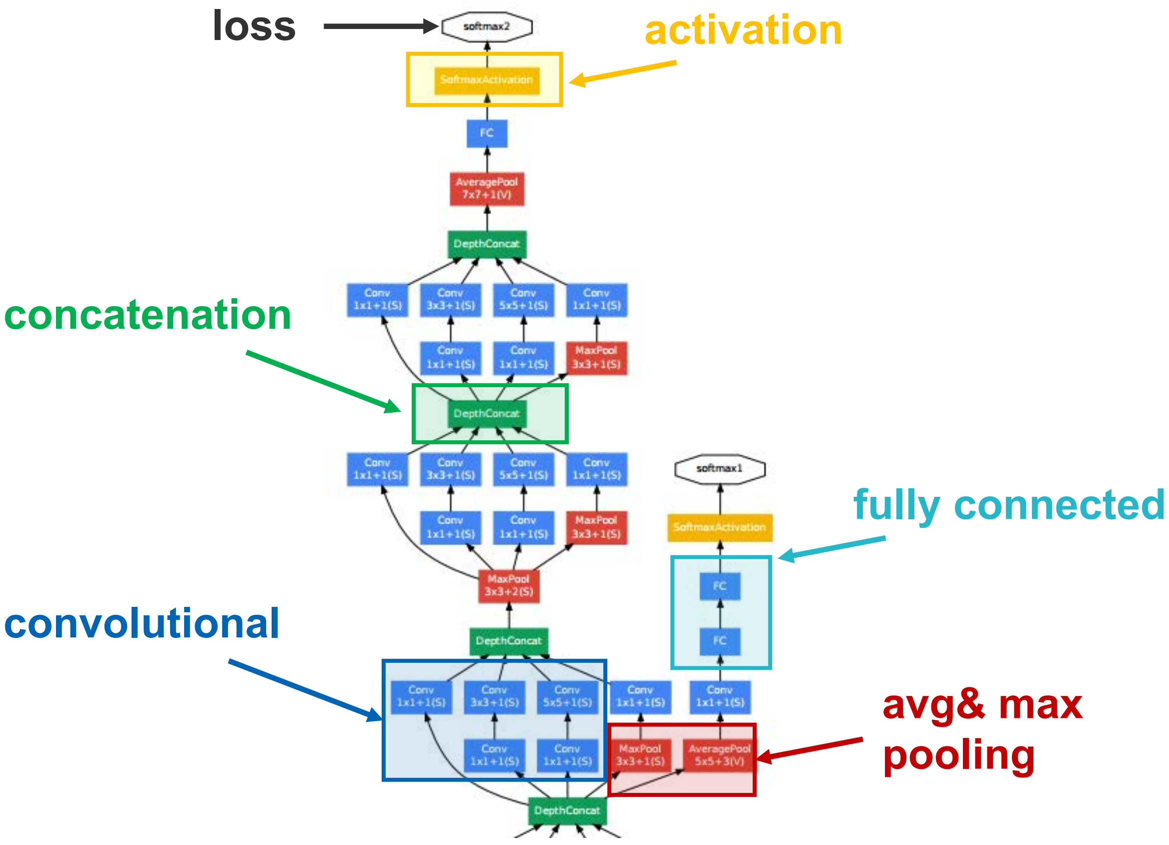


blocks with residual connections

Modern 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:



loss

activation

concatenation

fully connected

convolutional

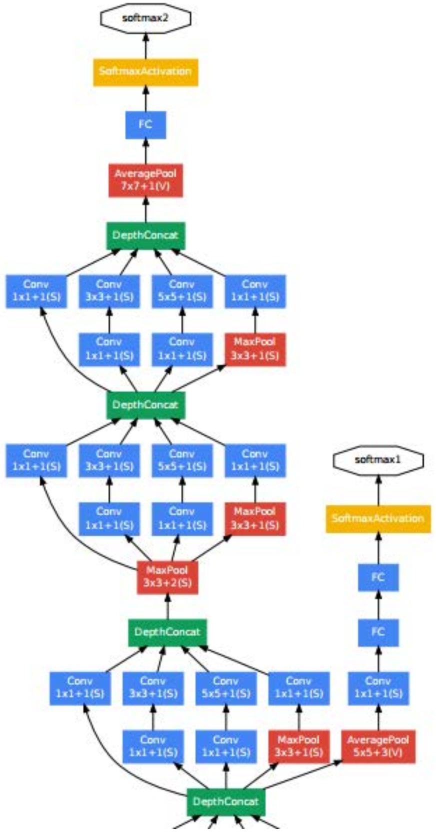
avg& max pooling

(a part of GoogleNet)

Modern 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:

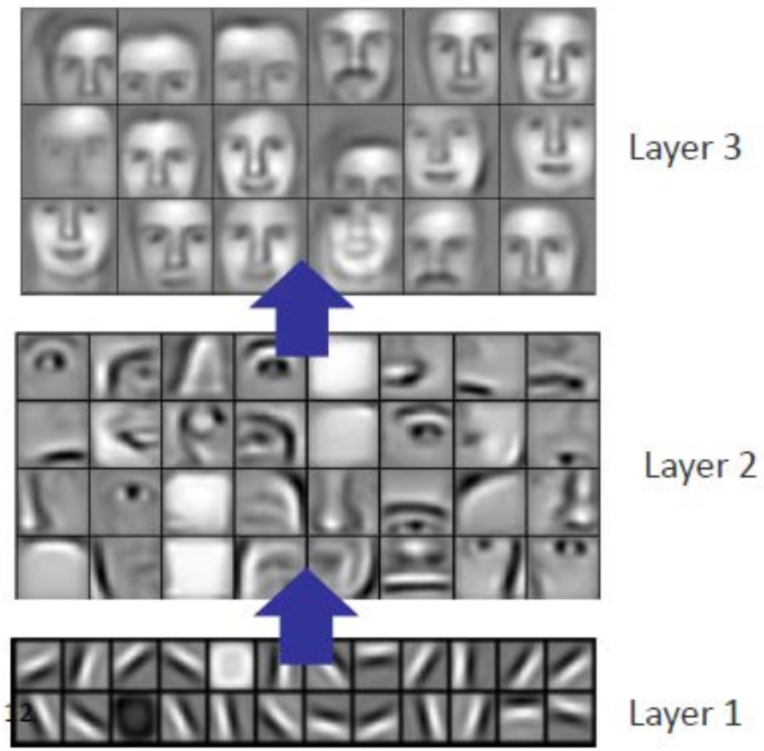


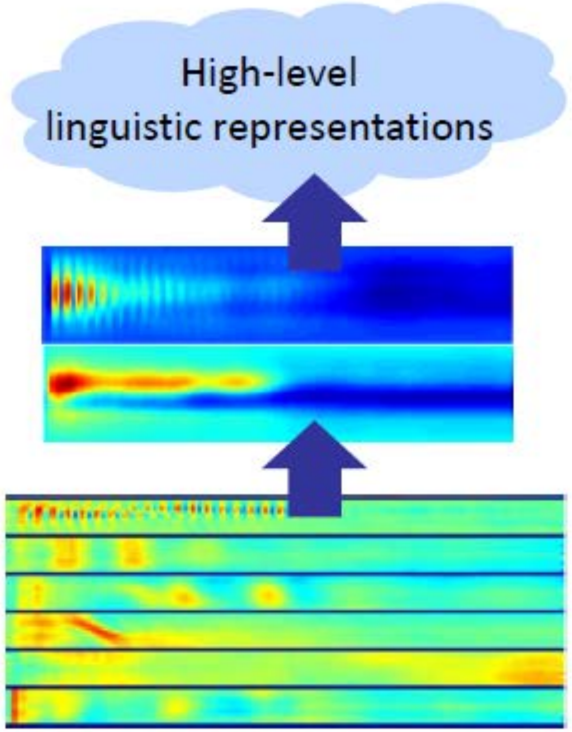
(a part of GoogleNet)

- **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.**

Feature learning

- Successful learning of intermediate representations [Lee et al ICML 2009, Lee et al NIPS 2009]





- 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.

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)

- ❑ Similarities and differences between GMs and NNs
    - ❑ Graphical models vs. computational graphs
    - ❑ Sigmoid Belief Networks as graphical models
    - ❑ Deep Belief Networks and Boltzmann Machines

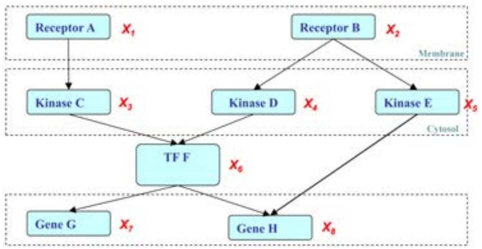
- ❑ Combining DL methods and GMs
    - ❑ Using outputs of NNs as inputs to GMs
    - ❑ GMs with potential functions represented by NNs
    - ❑ NNs with structured outputs

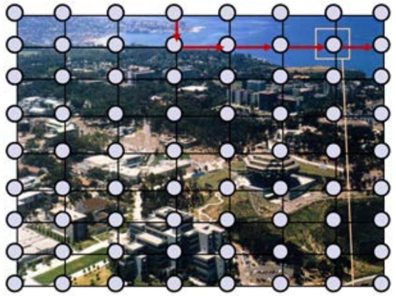
- ❑ Bayesian Learning of NNs
    - ❑ Bayesian learning of NN parameters
    - ❑ Deep kernel learning

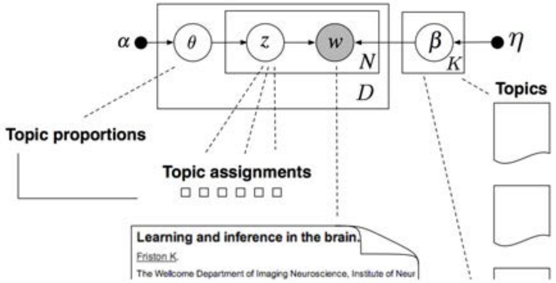
Graphical models vs. Deep nets

Graphical models

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

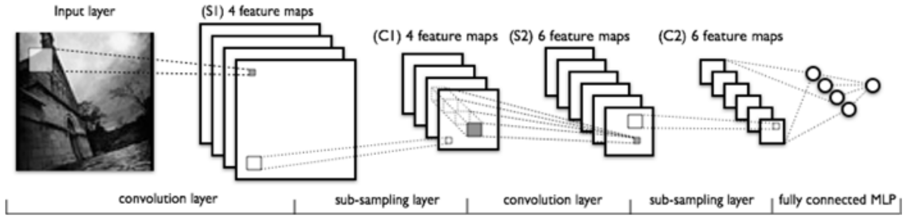


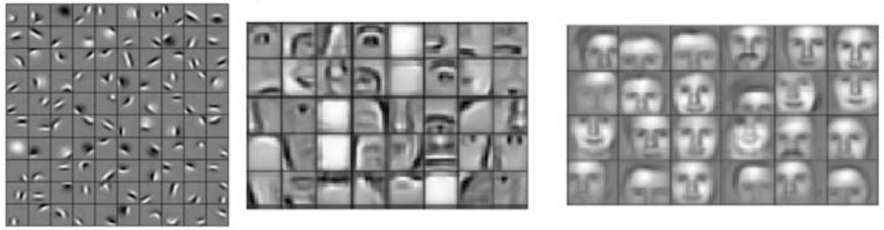



α
θ
z
w
N
D
β
K
η
Topics
Topic proportions
Topic assignments
Learning and inference in the brain.
Friston K.
The Wellcome Department of Imaging Neuroscience, Institute of Neur

Deep neural networks

- Learn representations that facilitate computation and performance on the end-metric (intermediate representations are not guaranteed to 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 are not guaranteed to 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

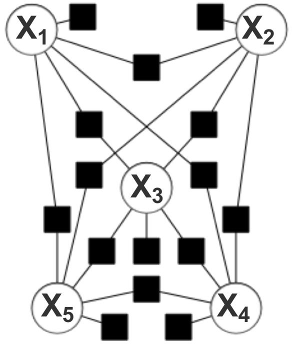
Graphical models

Utility of the graph

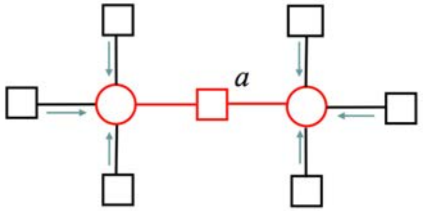
- A vehicle for synthesizing a global loss function from local structure
  - potential function, feature function, etc.
- A vehicle for designing sound and efficient inference algorithms
  - Sum-product, mean-field, etc.
- A vehicle to inspire approximation and penalization
  - Structured MF, Tree-approximation,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



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

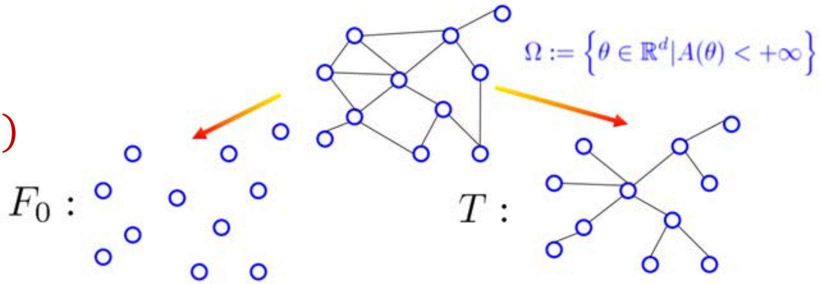


$$m_{i\to a}(x_i) = \prod_{c\in N(i)\backslash a} m_{c\to i}(x_i)$$

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

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

$Q(H) \sim P(H|V)$

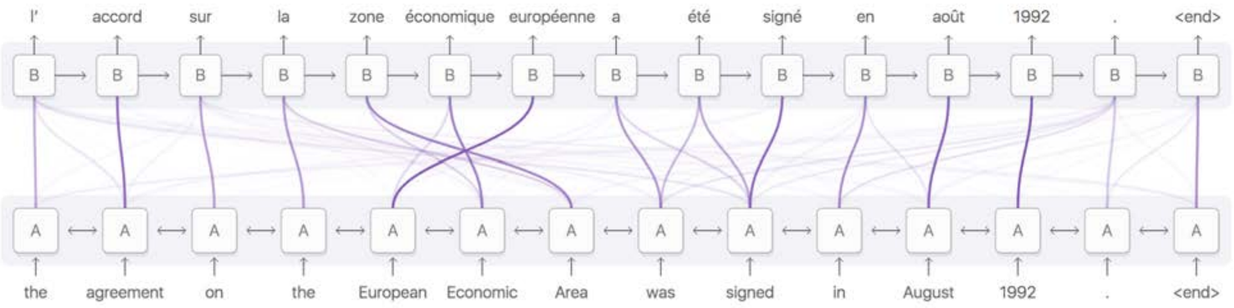


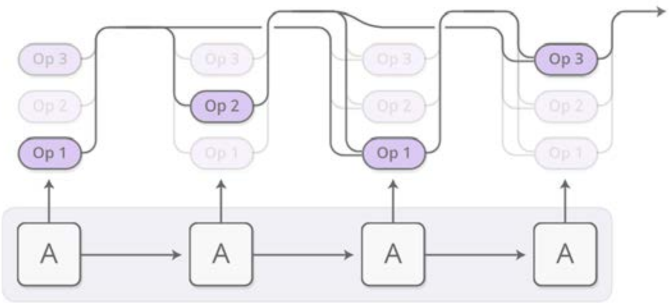
$\Omega := \{\theta \in \mathbb{R}^d | A(\theta) < +\infty\}$

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

$$\boldsymbol{\theta} = \mathrm{argmax}_{\boldsymbol{\theta}} P_{\theta}(V)$$

Graphical models vs. Deep nets





Images from Distill.pub

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 and 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

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 and 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...

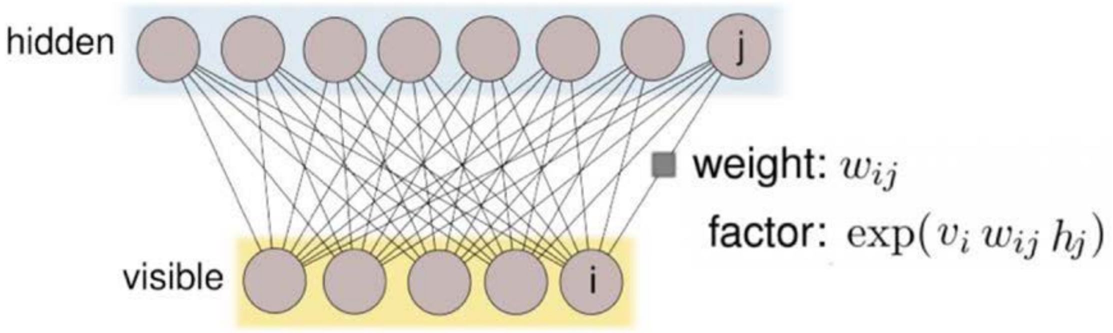
	DL ⋚ ?	ML (e.g., GM)
Empirical goal:	e.g., classification, feature learning	e.g., latent variable inference, transfer learning
Structure:	Graphical	Graphical
Objective:	Something aggregated from local functions	Something aggregated from local functions
Vocabulary:	Neuron, activation function, …	Variable, 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

- So far:
  - Graphical models are representations of probability distributions
  - Neural networks are function approximators (with no probabilistic meaning)
- Some of the neural nets are in fact proper graphical models (i.e., units/neurons represent 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

I: Restricted Boltzmann Machines

- RBM is a Markov random field represented with a bi-partite graph
- All nodes in one layer/part of the graph are connected to all in the other; no inter-layer connections



- Joint distribution:

$$P(v,h) = \frac{1}{Z}\exp\left\{\sum_{i,j} w_{ij}v_i h_i + \sum_i b_i v_i + \sum_j c_j h_j\right\}$$

Images from Marcus Frean, MLSS Tutorial 2010

I: Restricted Boltzmann Machines

- Log-likelihood of a single data point (unobservables marginalized out):

$$\log L(v) = \log \sum_{h} \exp\left\{\sum_{i,j} w_{ij} v_i h_i + \sum_{i} b_i v_i + \sum_{j} c_j h_j - \log(Z)\right\}$$

- Gradient of the log-likelihood w.r.t. the model parameters:

$$\frac{\partial}{\partial w_{ij}} \log L(v) = \sum_{h} P(h|v) \frac{\partial}{\partial w_{ij}} P(v,h) - \sum_{v,h} P(v,h) \frac{\partial}{\partial w_{ij}} P(v,h)$$

  - where we have averaging over the posterior and over the joint.

Images from Marcus Frean, MLSS Tutorial 2010

I: Restricted Boltzmann Machines

- Gradient of the log-likelihood w.r.t. the parameters (alternative form):

$$\frac{\partial}{\partial w_{ij}} \log L(v) = \mathbb{E}_{P(h|v)}\left[\frac{\partial}{\partial w_{ij}} P(v,h)\right] - \mathbb{E}_{P(v,h)}\left[\frac{\partial}{\partial w_{ij}} P(v,h)\right]$$

- Both expectations can be approximated via sampling
- Sampling from the posterior is exact (RBM factorizes over h given v)
- Sampling from the joint is done via MCMC (e.g., Gibbs sampling)
- In the neural networks literature:
  - computing the first term is called the clamped / wake / positive phase (the network is "awake" since it conditions on the visible variables)
  - Computing the second term is called the unclamped / sleep / free / negative phase (the network is "asleep" since it samples the visible variables from the joint; metaphorically, it is "dreaming" the visible inputs)

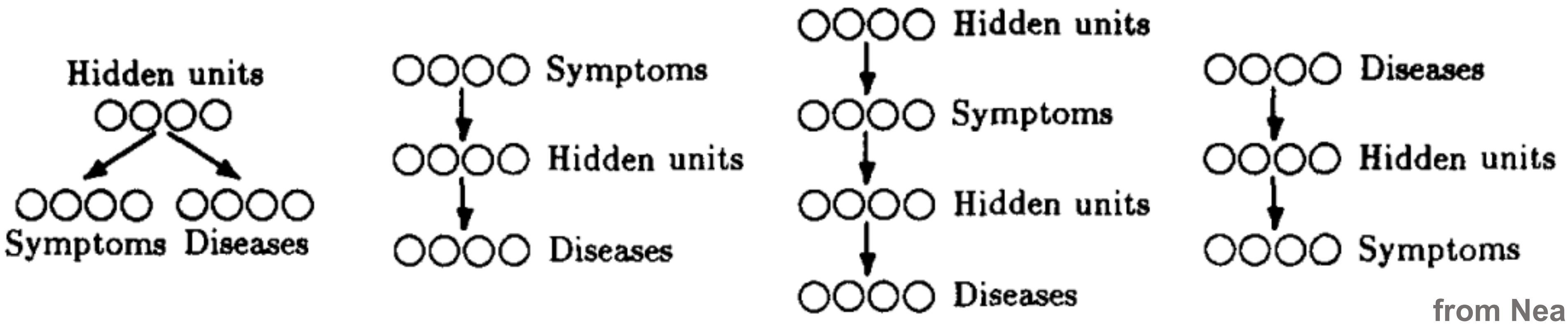
I: Restricted Boltzmann Machines

- Gradient of the log-likelihood w.r.t. the parameters (alternative form):

$$\frac{\partial}{\partial w_{ij}} \log L(v) = \mathbb{E}_{P(h|v)}\left[\frac{\partial}{\partial w_{ij}} P(v,h)\right] - \mathbb{E}_{P(v,h)}\left[\frac{\partial}{\partial w_{ij}} P(v,h)\right]$$

- Learning is done by optimizing the log-likelihood of the model for a given data via stochastic gradient descent (SGD)
- Estimation of the second term (the negative phase) heavily relies on the mixing properties of the Markov chain
- This often causes slow convergence and requires extra computation

II: Sigmoid Belief Networks

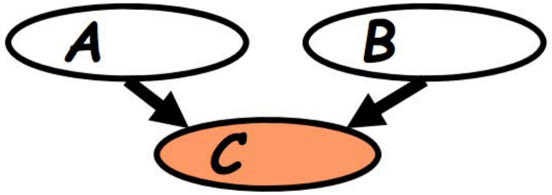


from Neal, 1992

- Sigmoid belief nets are simply Bayesian networks over binary variables with conditional probabilities represented by sigmoid functions:

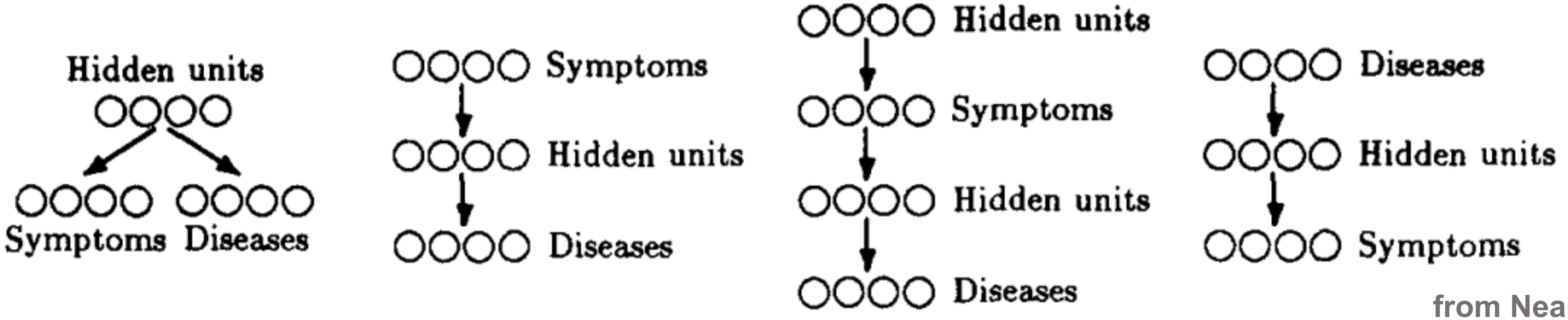
$$P(x_i|\pi(x_i)) = \sigma\left(x_i \sum_{x_j \in \pi(x_i)} w_{ij} x_j\right)$$

- Bayesian networks exhibit a phenomenon called “explain away effect”



If A correlates with C, then the chance of B correlating with C decreases. ⇒ A and B become correlated given C.

II: Sigmoid Belief Networks

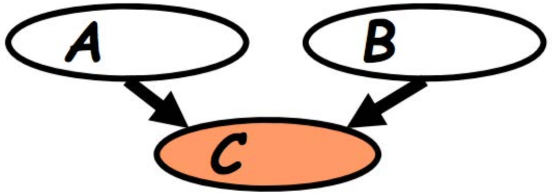


from Neal, 1992

- Sigmoid belief nets are simply Bayesian networks over binary variables with conditional probabilities represented by sigmoid functions:

$$P(x_i|\pi(x_i)) = \sigma\left(x_i \sum_{x_j \in \pi(x_i)} w_{ij}x_j\right)$$

- Bayesian networks exhibit a phenomenon called "explain away effect"



Note:
Due to the "explain away effect," when we condition on the visible layer in belief networks, hidden variables all become dependent.

Sigmoid Belief Networks: Learning and Inference

- Neal proposed Monte Carlo methods for learning and inference (Neal, 1992):

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

log derivative

prob. of the visibles via marginalization

Bayes rule + rearrange sums

Plug-in the actual sigmoid form of the conditional prob.

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…**

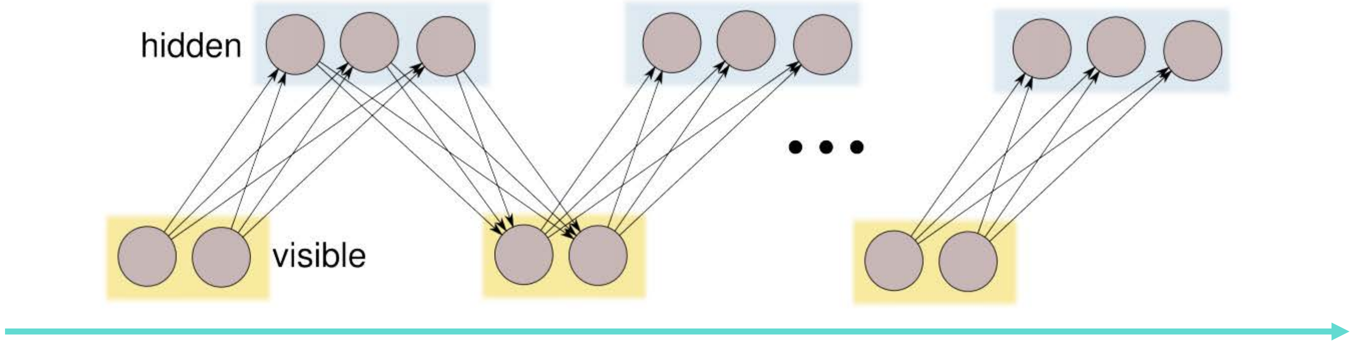
Equations from Neal, 1992

RBMs are infinite belief networks

- Recall the expression for the gradient of the log likelihood for RBM:

$$\frac{\partial}{\partial w_{ij}} \log L(v) = \mathbb{E}_{P(h|v)}\left[\frac{\partial}{\partial w_{ij}} P(v,h)\right] - \mathbb{E}_{P(v,h)}\left[\frac{\partial}{\partial w_{ij}} P(v,h)\right]$$

- To make a gradient update of the model parameters, we need compute the expectations via sampling.
  - We can sample exactly from the posterior in the first term
  - We run block Gibbs sampling to approximately sample from the joint distribution

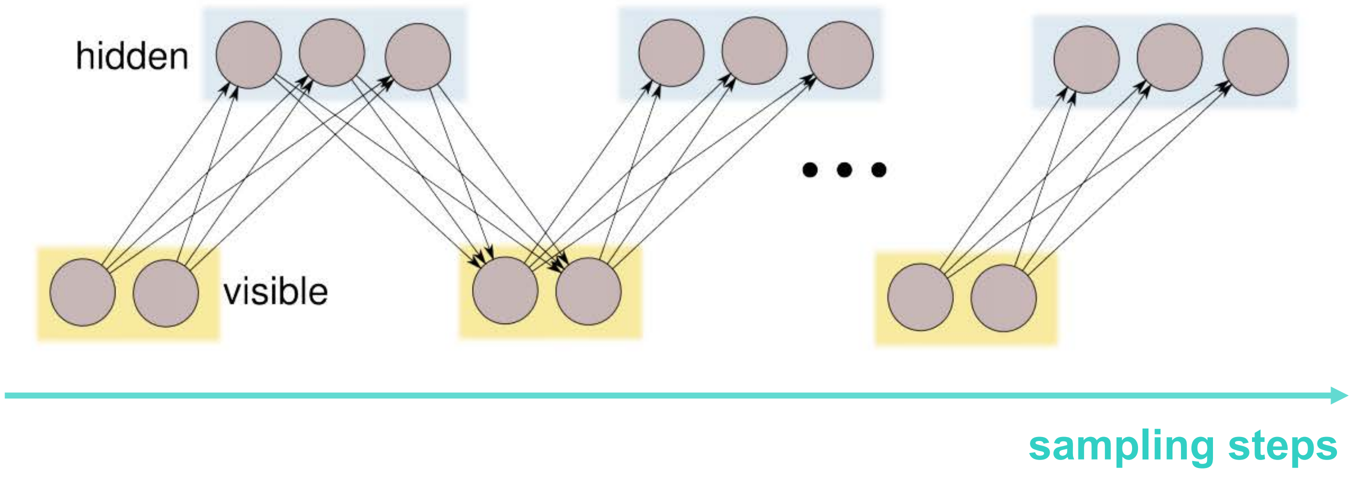


images from Marcus Frean, MLSS Tutorial 2010

sampling steps

RBMs are infinite belief networks

- Gibbs sampling: alternate between sampling hidden and visible variables

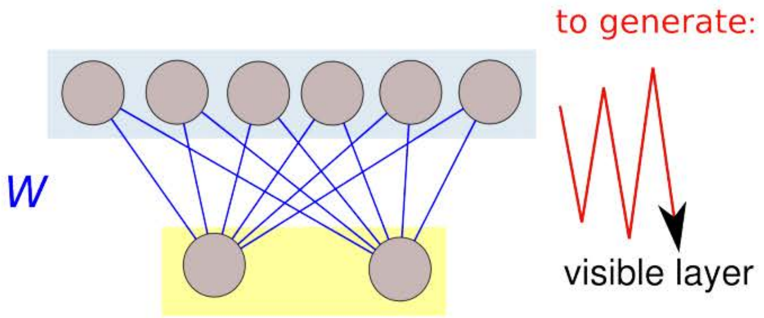


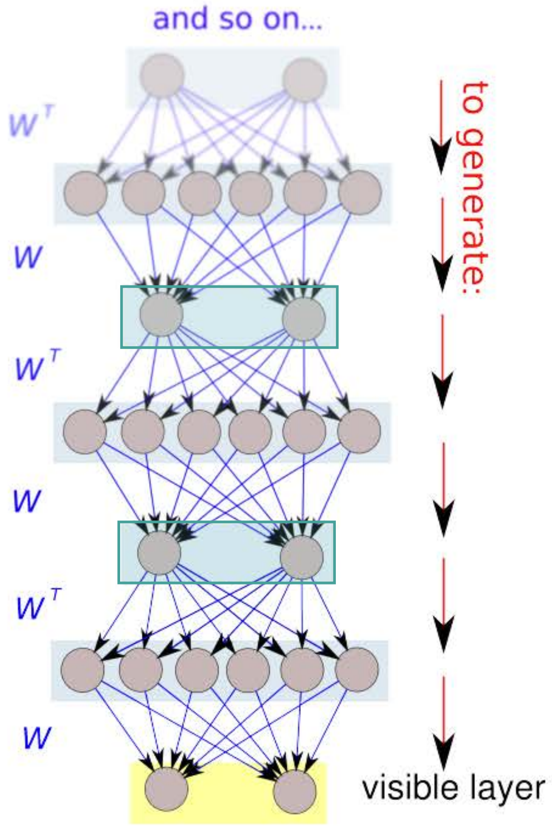
- Conditional distributions $P(v|h)$ and $P(h|v)$ are represented by sigmoids
- Thus, we can think of Gibbs sampling from the joint distribution represented by an RBM as a top-down propagation in an infinitely deep sigmoid belief network!

images from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

- RBMs are equivalent to infinitely deep belief networks



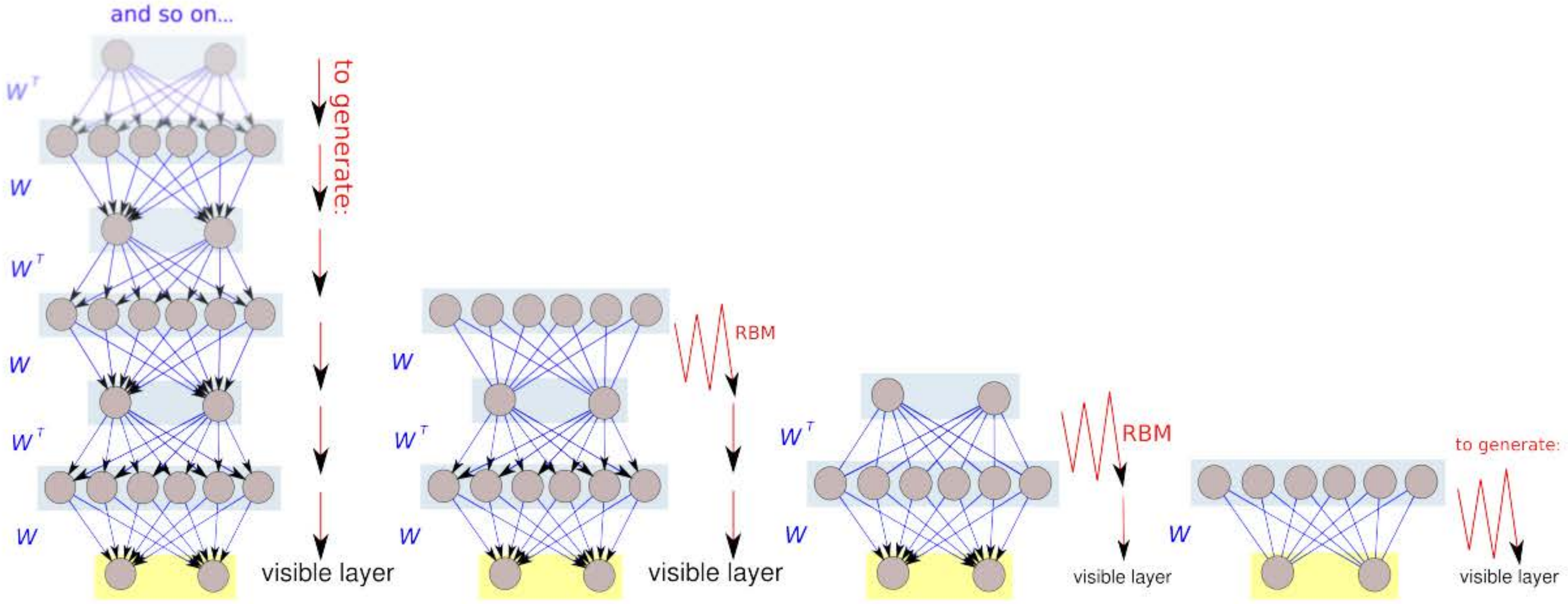


- Sampling from this is the same as sampling from the network on the right

images from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

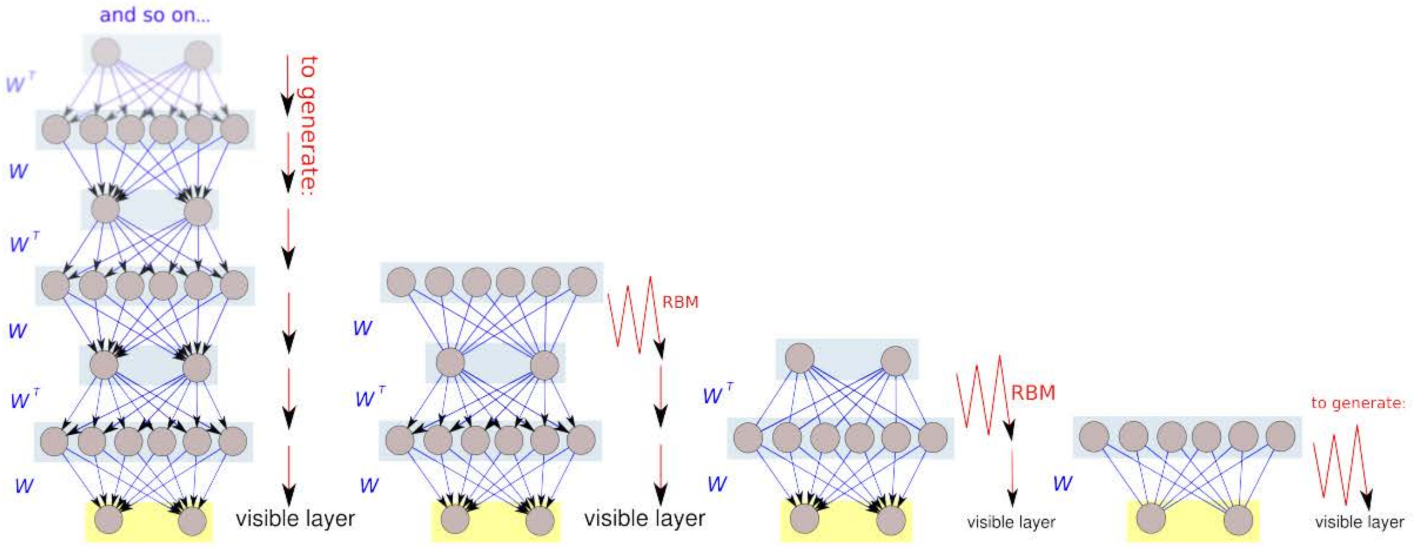
- RBMs are equivalent to infinitely deep belief networks



images from Marcus Frean, MLSS Tutorial 2010

RBMs are infinite belief networks

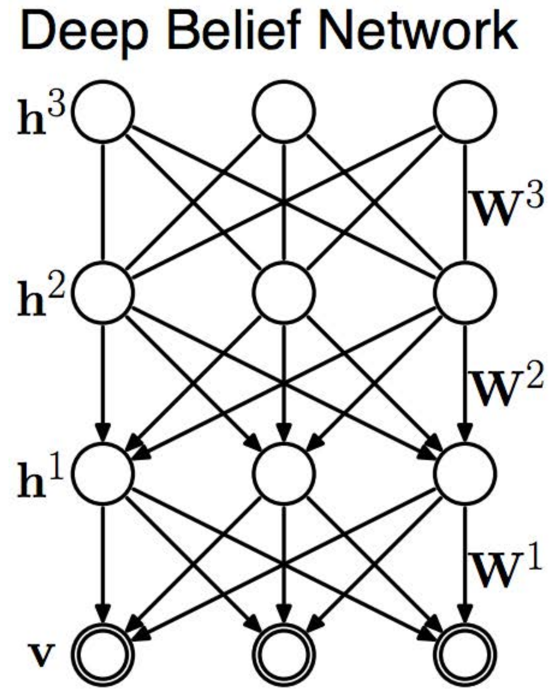
- RBMs are equivalent to infinitely deep belief networks



- When we train an RBM, we are really training an infinitely deep brief net!
- It is just that the weights of all layers are tied.
- If the weights are “untied” to some extent, we get a Deep Belief Network.

images from Marcus Frean, MLSS Tutorial 2010

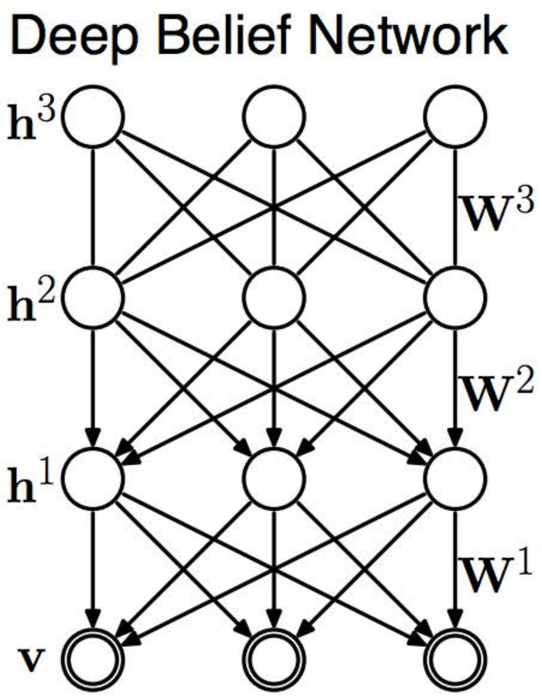
III: Deep Belief Nets



Now weights are untied!

- DBNs are hybrid graphical models (chain graphs):
  - Exact 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 Belief Networks



- DBNs represent a joint probability distribution

$$P(v,h^1,h^2,h^3) = P(h^2,h^3)P(h^1|h^2)P(v|h^1)$$

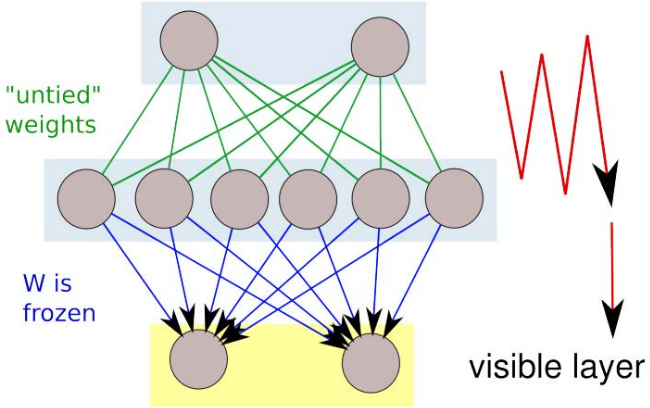
- Note that $P(h^2,h^3)$ is an RBM and the conditionals $P(h^1|h^2)$ and $P(v|h^1)$ are represented in the sigmoid form
- The model is trained by optimizing the log likelihood for a given data $\log P(v)$

Challenges:

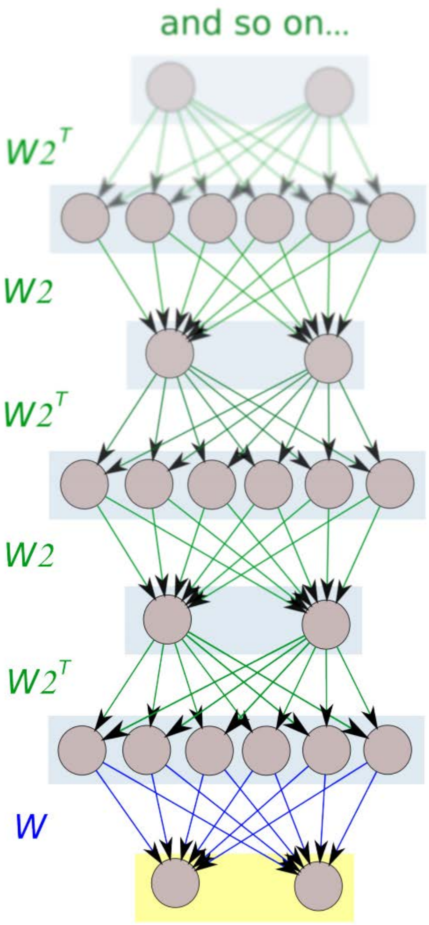
- Exact inference in DBNs is problematic due to explain away effect
- Training is done in two stages:
  - greedy pre-training + ad-hoc fine-tuning; no proper joint training
- Approximate inference is feed-forward (bottom-up)

DBN: Layer-wise pre-training

- Pre-train and freeze the 1st RBM
- Stack another RBM on top and train it



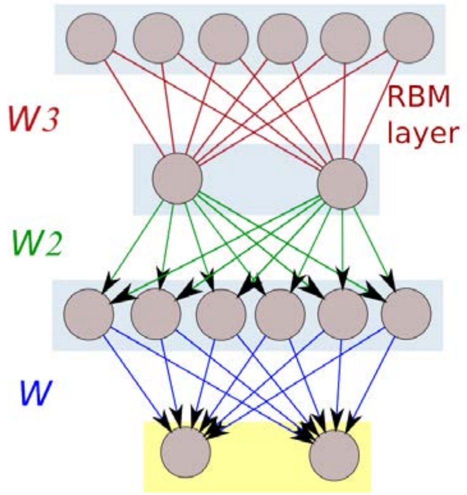
- The weights weights 2+ layers remain tied
- We repeat this procedure: pre-train and untie the weights layer-by-layer…

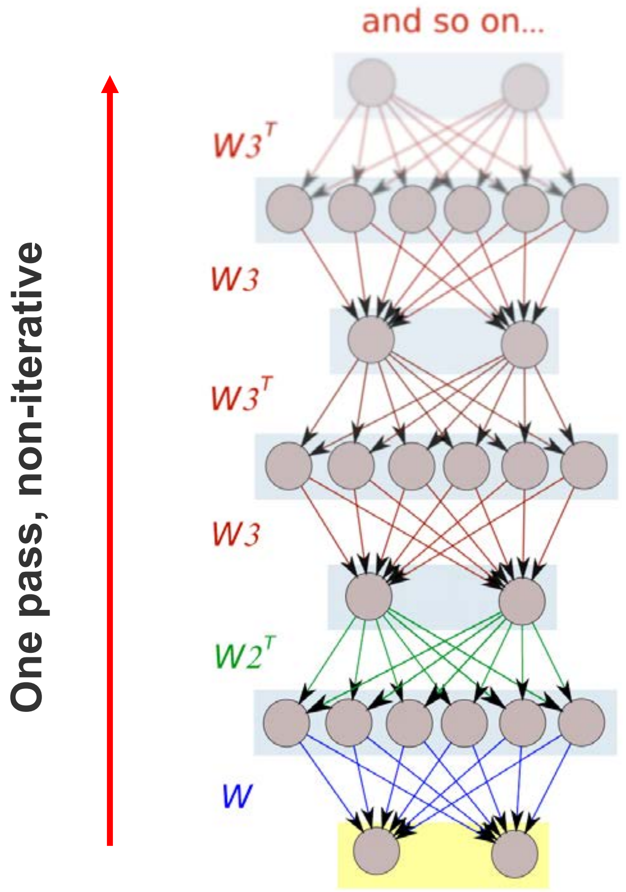


images from Marcus Frean, MLSS Tutorial 2010

DBN: Layer-wise pre-training

- We repeat this procedure: pre-train and untie the weights layer-by-layer:
- The weights of 3+ layers remain tied





One pass, non-iterative

- and so forth
- *From the optimization perspective, this procedure loosely corresponds to an approximate block-coordinate accent on the log-likelihood*

images from Marcus Frean, MLSS Tutorial 2010

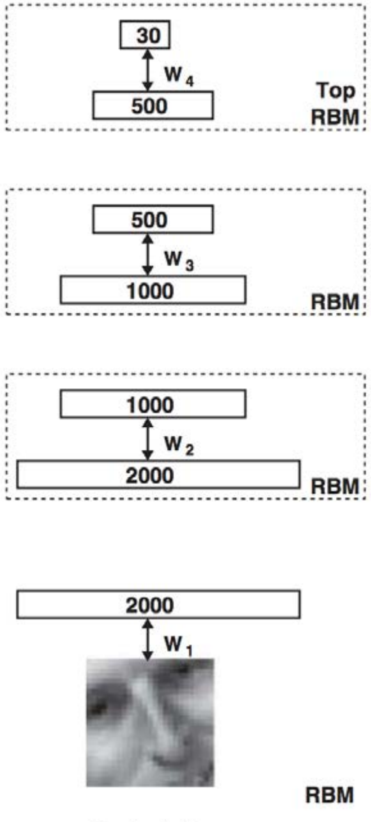
DBN: Fine-tuning

- Pre-training is quite ad-hoc and is unlikely to lead to a good probabilistic model *per se*
- However, the layers of representations could perhaps be useful for some other downstream tasks!

- We can further “fine-tune” a pre-trained DBN for some other task

<u>Setting A:</u> Unsupervised learning (DBN → autoencoder)

1. Pre-train a stack of RBMs in a greedy layer-wise fashion
2. “Unroll” the RBMs to create an autoencoder
3. Fine-tune the parameters by optimizing the reconstruction error



Pretraining

images from Hinton & Salakhutdinov, 2006

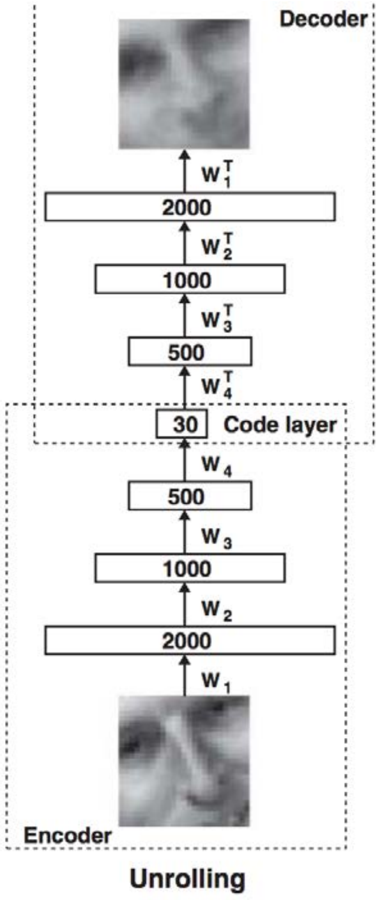
DBN: Fine-tuning

- Pre-training is quite ad-hoc and is unlikely to lead to a good probabilistic model *per se*
- However, the layers of representations could perhaps be useful for some other downstream tasks!

- We can further "fine-tune" a pre-trained DBN for some other task

Setting A: Unsupervised learning (DBN → autoencoder)

1. Pre-train a stack of RBMs in a greedy layer-wise fashion
2. "Unroll" the RBMs to create an autoencoder
3. Fine-tune the parameters by optimizing the reconstruction error



images from Hinton & Salakhutdinov, 2006

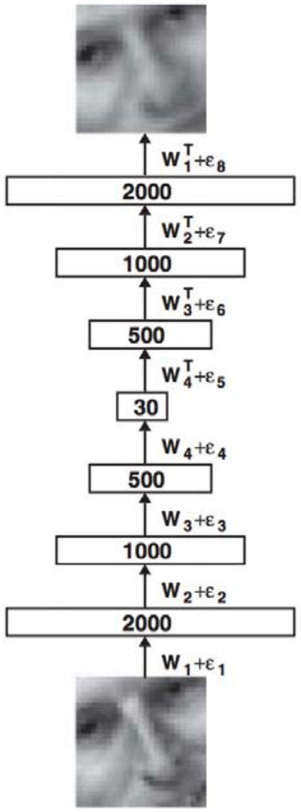
DBN: Fine-tuning

- Pre-training is quite ad-hoc and is unlikely to lead to a good probabilistic model *per se*
- However, the layers of representations could perhaps be useful for some other downstream tasks!

- We can further “fine-tune” a pre-trained DBN for some other task

Setting A: Unsupervised learning (DBN → autoencoder)

1. Pre-train a stack of RBMs in a greedy layer-wise fashion
2. “Unroll” the RBMs to create an autoencoder
3. Fine-tune the parameters by optimizing the reconstruction error



Fine-tuning

images from Hinton & Salakhutdinov, 2006

DBN: Fine-tuning

- Pre-training is quite ad-hoc and is unlikely to lead to a good probabilistic model *per se*
- However, the layers of representations could perhaps be useful for some other downstream tasks!

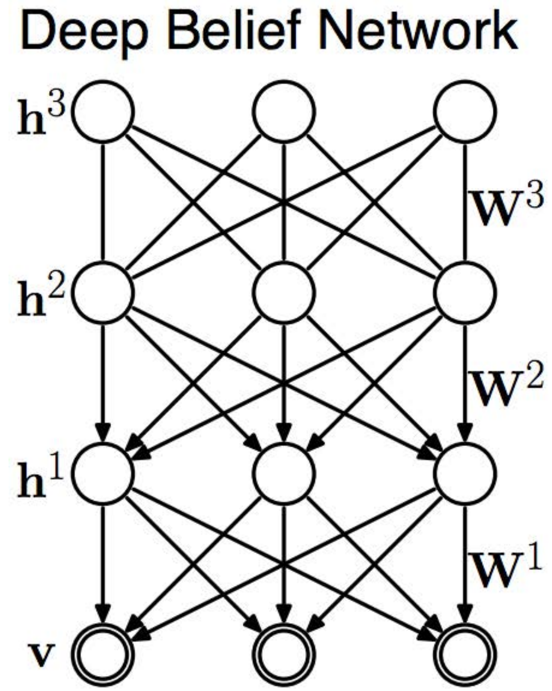
- We can further “fine-tune” a pre-trained DBN for some other task

<u>Setting B:</u> Supervised learning (DBN → classifier)

1. Pre-train a stack of RBMs in a greedy layer-wise fashion
2. “Unroll” the RBMs to create a feedforward classifier
3. Fine-tune the parameters by optimizing the reconstruction error

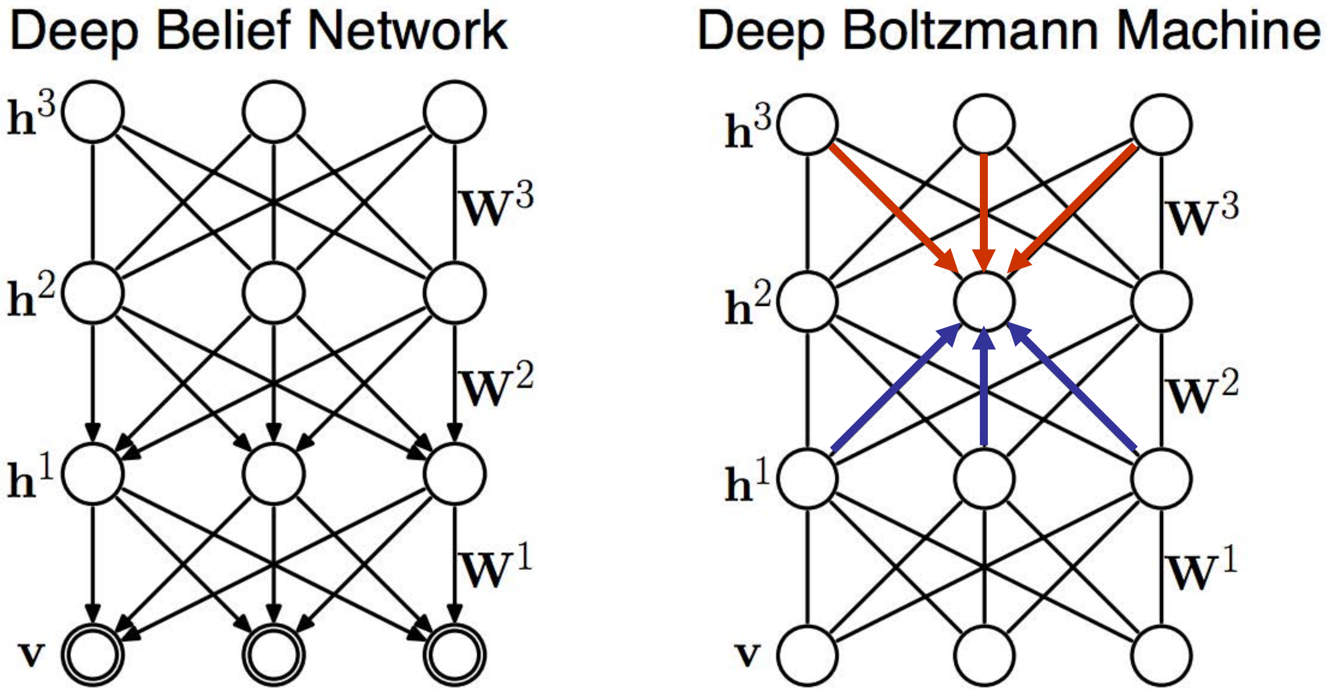
Some intuitions about how pre-training works:
Erhan et al.: Why Does Unsupervised Pre-training Help Deep Learning? JMLR, 2010

Deep Belief Nets and 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 Belief Nets and Boltzmann Machines



- DBMs are fully un-directed models (Markov random fields):
  - Can be trained similarly as RBMs via MCMC (Hinton & Sejnowski, 1983)
  - Use a variational approximation of the data distribution for faster training (Salakhutdinov & Hinton, 2009)
  - Similarly, can be used to initialize other networks for downstream tasks

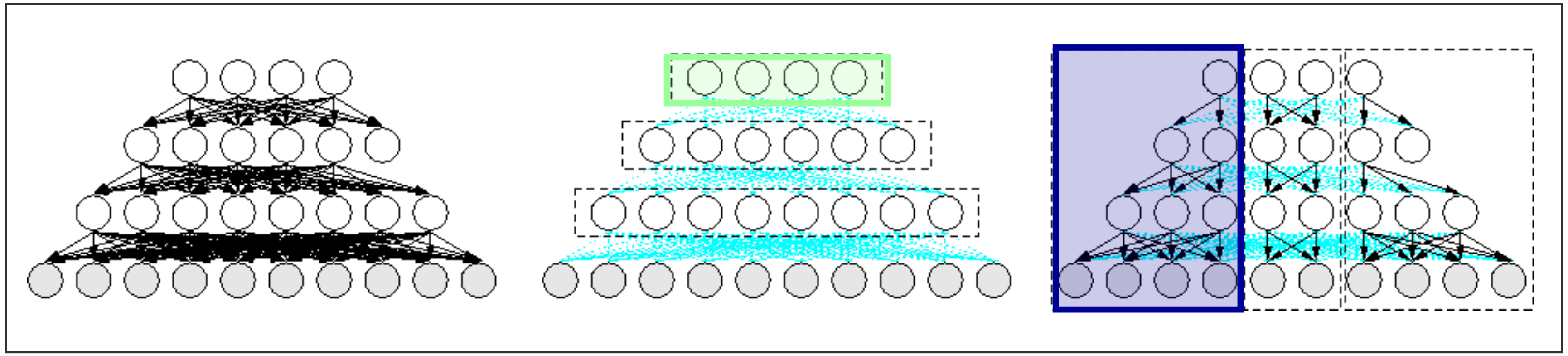
Graphical models vs. Deep networks

- A few critical points to note about all these models:
  - The primary goal of deep generative models is to represent the distribution of the observable variables. Adding layers of hidden variables allows to represent increasingly more complex distributions.
  - Hidden variables are secondary (auxiliary) elements used to facilitate learning of complex dependencies between the observables.
  - Training of the model is ad-hoc, but what matters is the quality of learned hidden representations.
  - Representations are judged by their usefulness on a downstream task (the probabilistic meaning of the model is often discarded at the end).
- In contrast, classical graphical models are often concerned with the correctness of learning and inference of all variables

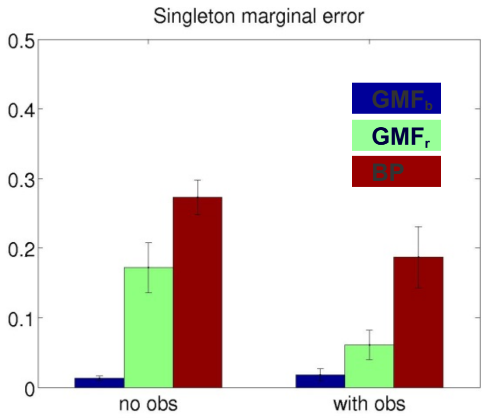
An old study of belief networks from the GM standpoint

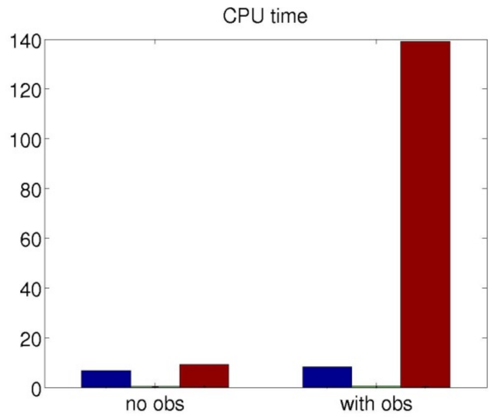
[Xing, Russell, Jordan, UAI 2003]

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



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





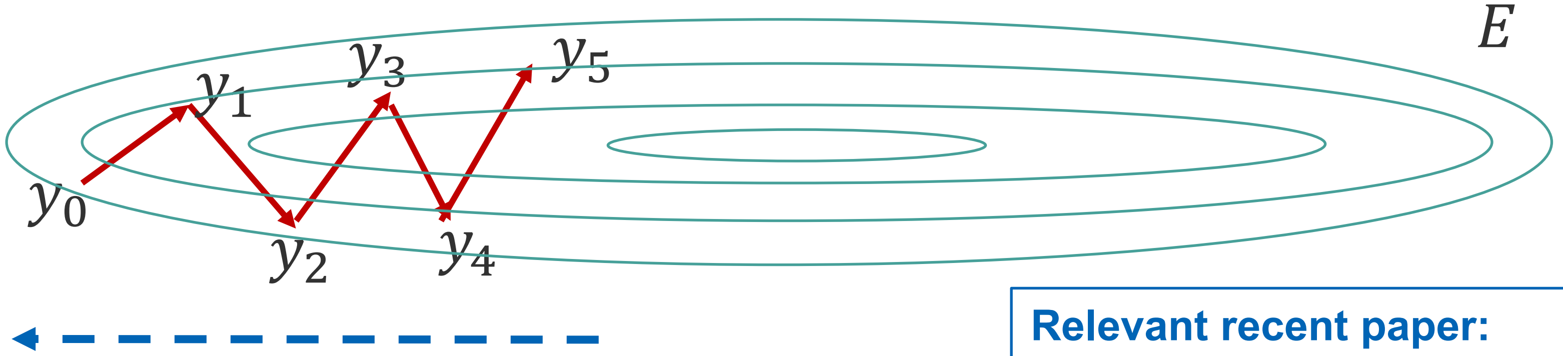
“Optimize” how to optimize via truncation & re-opt

- 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)

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



We can backprop through the optimization steps since they are just a sequence of computations

Relevant recent paper:
Anrychowicz et al.: *Learning to learn by gradient descent by gradient descent*. 2016.

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)

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

- We can think of y* as some non-linear differentiable function of the inputs and weights → impose some loss and optimize it as any other 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)

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 (models are driven by evaluating empirical performance on a downstream tasks)
  - DL models are good at learning robust hierarchical representations from the data and suitable for simple reasoning (call it "low-level cognition")
- GM: the effort is directed towards improving inference accuracy and convergence speed
  - GMs are best for provably correct inference and suitable for high-level complex reasoning tasks (call it "high-level cognition")
- Convergence of both fields is very promising!
  - Next part: a unified view of deep generative models in the GM interpretation

PETUUM

Supplementary

Outline

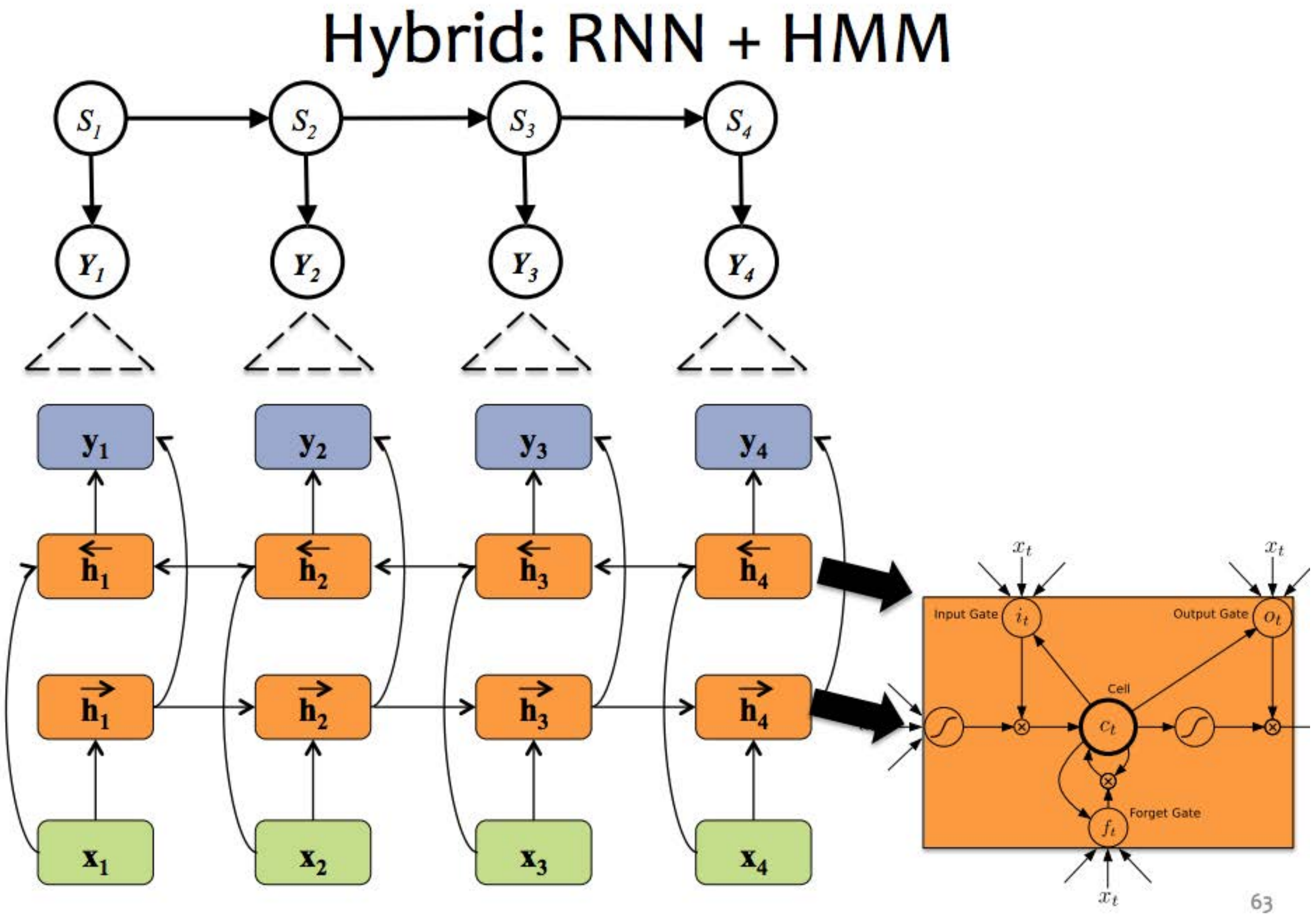
- An overview of DL components
  - Historical remarks: early days of neural networks
  - Modern building blocks: units, layers, activations functions, loss functions, etc.
  - Reverse-mode automatic differentiation (aka backpropagation)

- 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

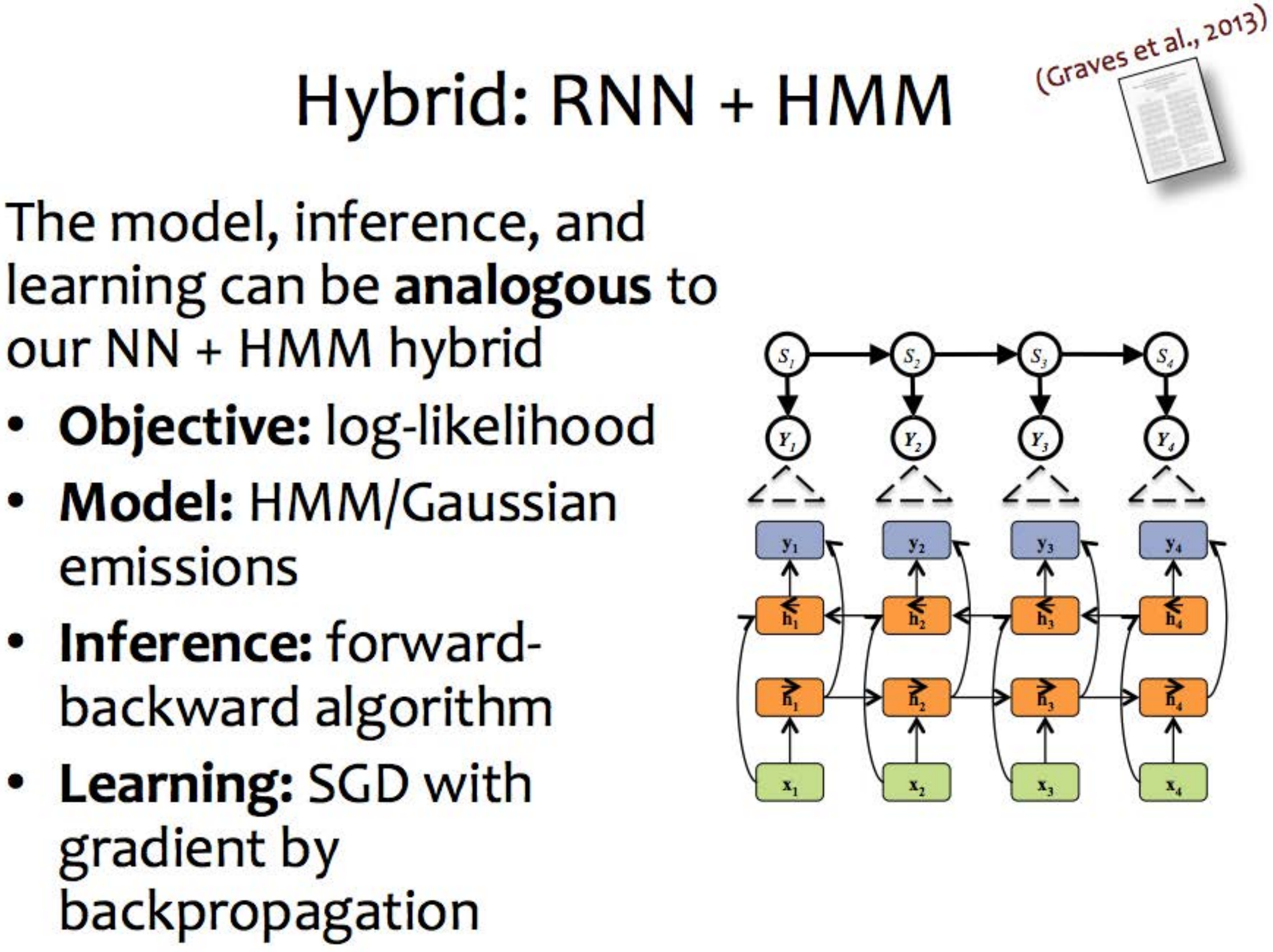
- Bayesian Learning of NNs
  - Bayesian learning of NN parameters
  - Deep kernel learning

Combining sequential NNs and GMs



slide courtesy: Matt Gormley

Combining sequential NNs and GMs



Hybrid: RNN + HMM

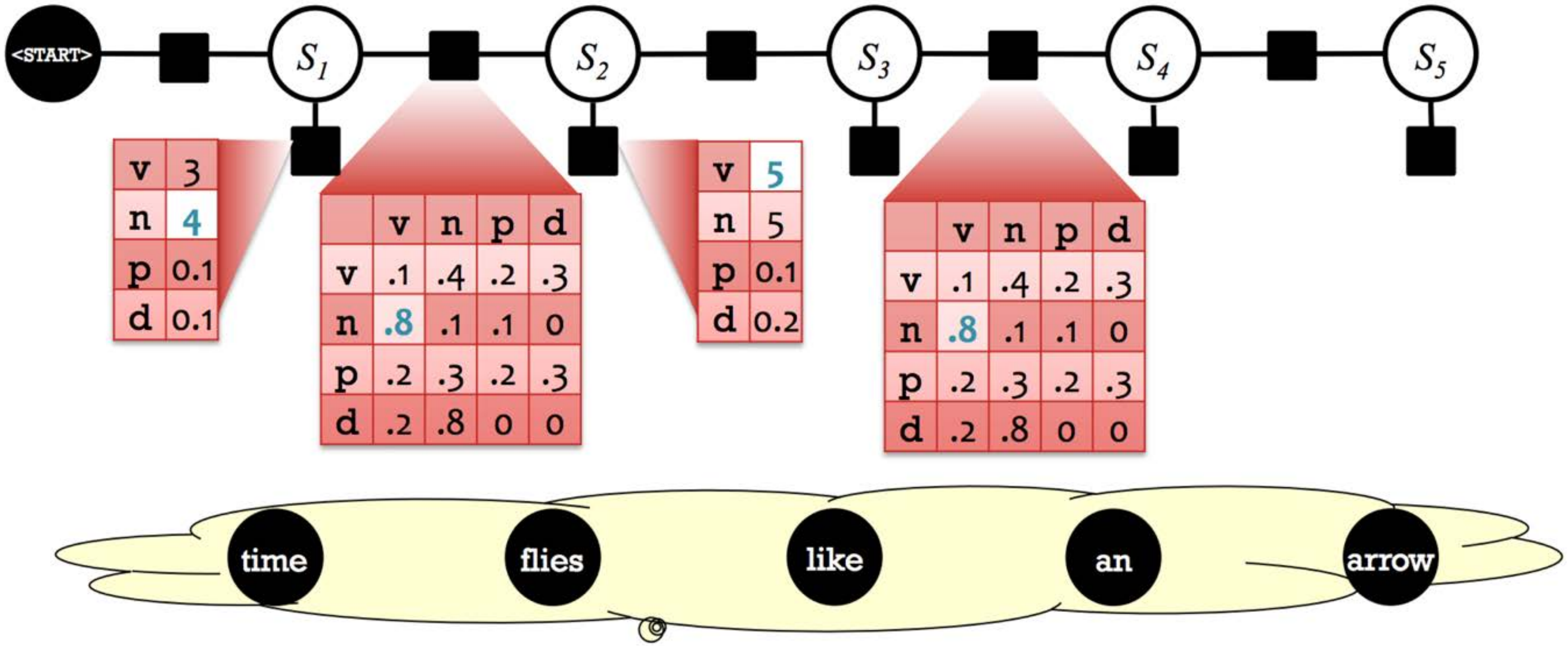
(Graves et al., 2013)

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

slide courtesy: Matt Gormley

Hybrid NNs + conditional GMs



v	3
n	4
p	0.1
d	0.1

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

v	5
n	5
p	0.1
d	0.2

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

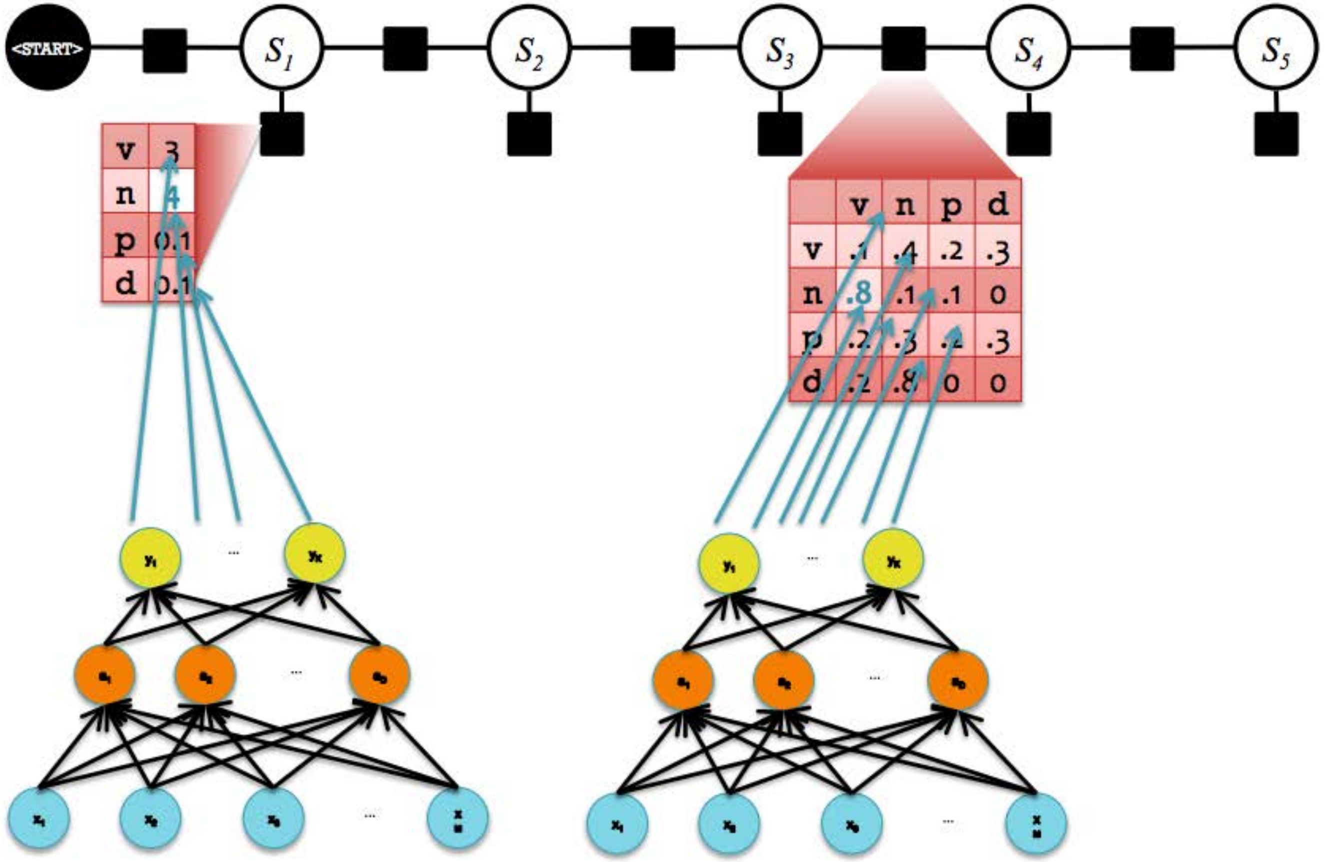
- In a standard CRF, each of the factor cells is a parameter.
- In a hybrid model, these values are computed by a neural network.

slide courtesy: Matt Gormley

Hybrid NNs + conditional GMs

Hybrid: Neural Net + CRF

Forward computation



v	3
n	4
p	0.1
d	0.1

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

slide courtesy: Matt Gormley

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)

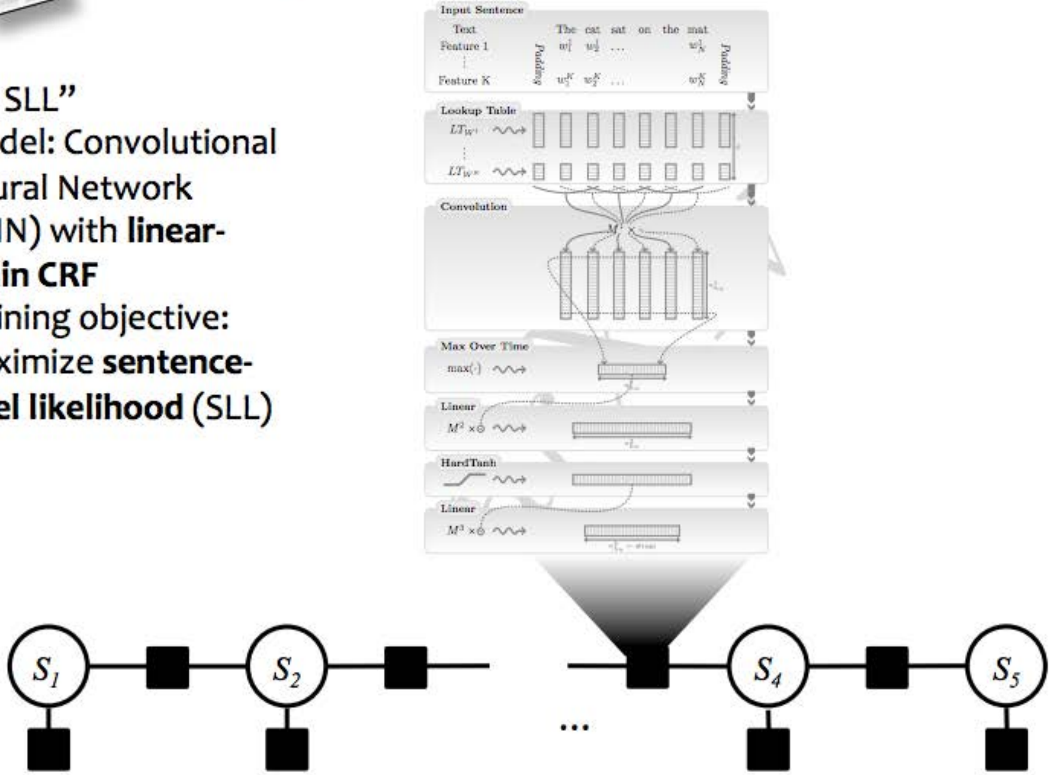


Figure from (Collobert & Weston, 2011)

slide courtesy: Matt Gormley

Using GMs as Prediction Explanations

Satellite imagery

Meaningful attributes

01 Nightlight intensity

02 Is urban

03 Has electricity

04 Has generator

05 Avg. temperature

06 Avg. percipitation

07 Vegetation

08 Avg. vegetation inc.

09 Avg. vegetation dec.

10 Avg. dist. to market

11 Avg. dist. to road

12 Num. of rooms

13 Dist. to water src.

14 Water usage p/ day

15 Is water payed

16 HH type: BQ

Al-Shedivat, Dubey, Xing, arXiv, 2017

Using GMs as Prediction Explanations

How do we build a powerful predictive model whose predictions we can interpret in terms of semantically meaningful features?

Satellite imagery

Area attributes

01 Nightlight intensity

02 Is urban

03 Has electricity

04 Has generator

05 Avg. temperature

06 Avg. percipitation

07 Vegetation

08 Avg. vegetation inc.

09 Avg. vegetation dec.

10 Avg. dist. to market

11 Avg. dist. to road

12 Num. of rooms

13 Dist. to water src.

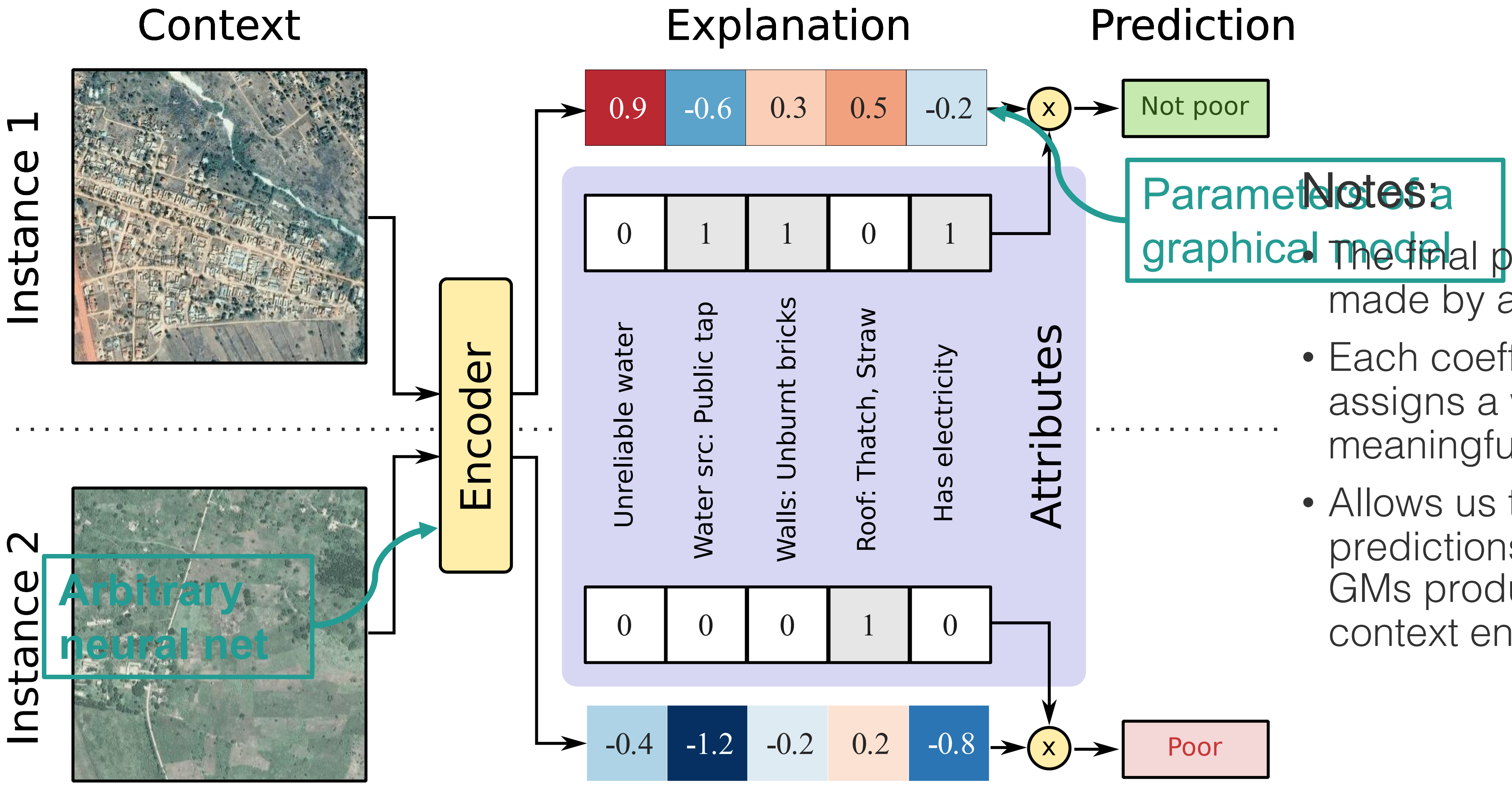
14 Water usage p/ day

15 Is water payed

16 HH type: BQ

Al-Shedivat, Dubey, Xing, arXiv, 2017

Contextual Explanation Networks (CENs)

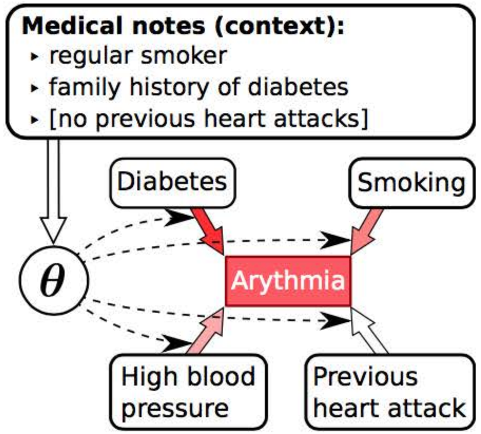


Notes:

- The final prediction is made by a linear GM.
- Each coefficient assigns a weight to a meaningful attribute.
- Allows us to judge predictions in terms of GMs produced by the context encoder.

Al-Shedivat, Dubey, Xing, arXiv, 2017

Contextual Explanation Networks (CENs)

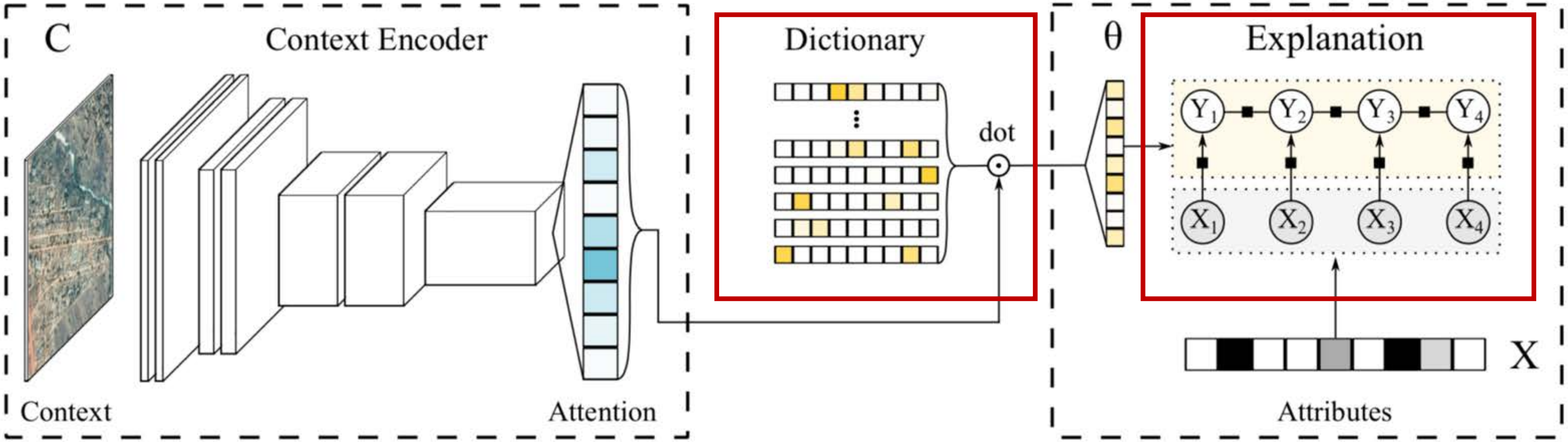


(a)

- General idea: Use deep neural nets to **generate parameters** for graphical models applicable in a given context (e.g., for a given patient).
- Produced GMs are used to make the final prediction ⇒ 100% fidelity and consistency.
- GMs are built on top of semantically meaningful variables (not deep embeddings!) and can be used as **explanations** for each prediction.

Al-Shedivat, Dubey, Xing, arXiv, 2017

CEN: Implementation Details

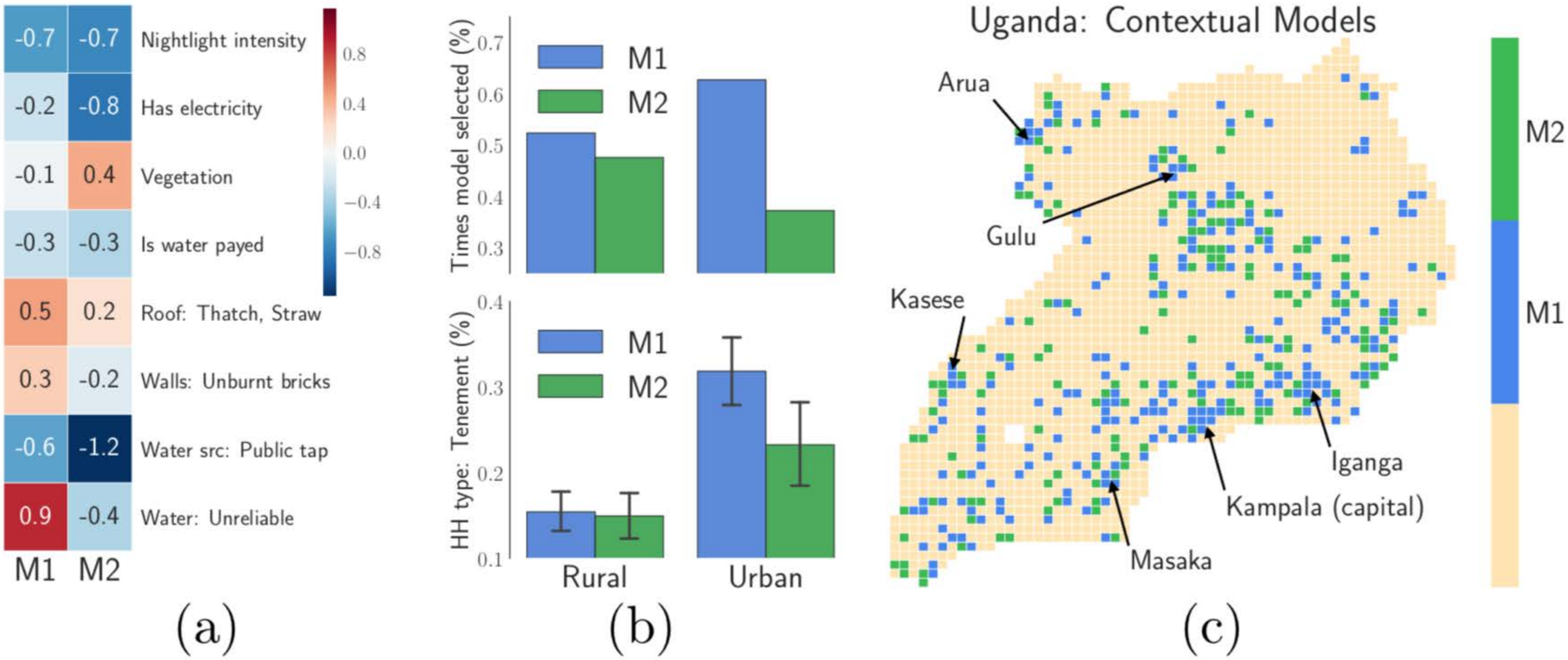


Workflow:

- Maintain a (sparse) dictionary of GM parameters.
- Process complex inputs (images, text, time series, etc.) using deep nets; use soft attention to either select or combine models from the dictionary.
- Use constructed GMs (e.g., CRFs) to make predictions.
- Inspect GM parameters to understand the reasoning behind predictions.

Al-Shedivat, Dubey, Xing, arXiv, 2017

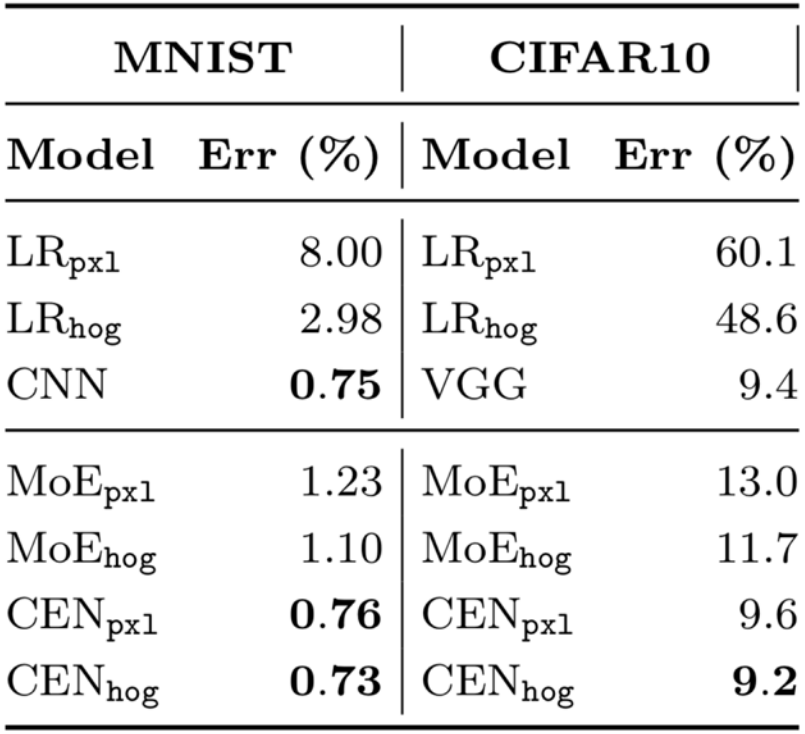
Results: imagery as context



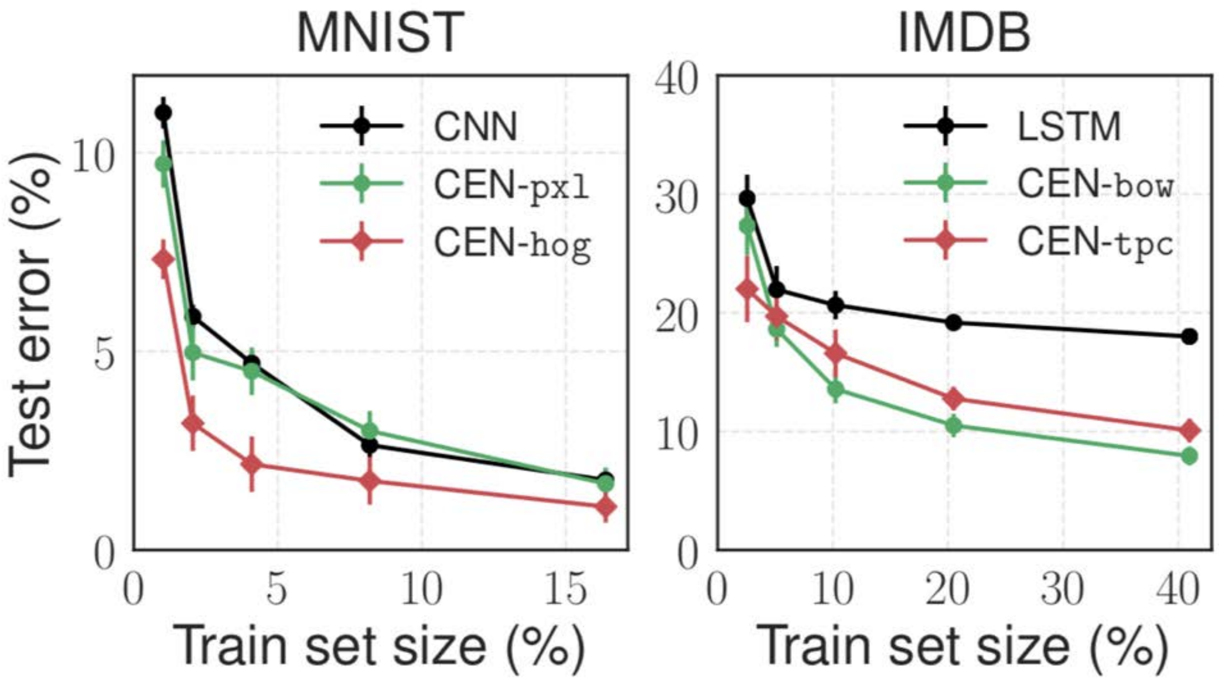
Based on the imagery, CEN learns to select different models for urban and rural areas

Al-Shedivat, Dubey, Xing, arXiv, 2017

Results: classical image & text datasets



MNIST		CIFAR10	
Model	**Err (%)**	**Model**	**Err (%)**
LR$_{\text{pxl}}$	8.00	LR$_{\text{pxl}}$	60.1
LR$_{\text{hog}}$	2.98	LR$_{\text{hog}}$	48.6
CNN	**0.75**	VGG	9.4
MoE$_{\text{pxl}}$	1.23	MoE$_{\text{pxl}}$	13.0
MoE$_{\text{hog}}$	1.10	MoE$_{\text{hog}}$	11.7
CEN$_{\text{pxl}}$	**0.76**	CEN$_{\text{pxl}}$	9.6
CEN$_{\text{hog}}$	**0.73**	CEN$_{\text{hog}}$	**9.2**

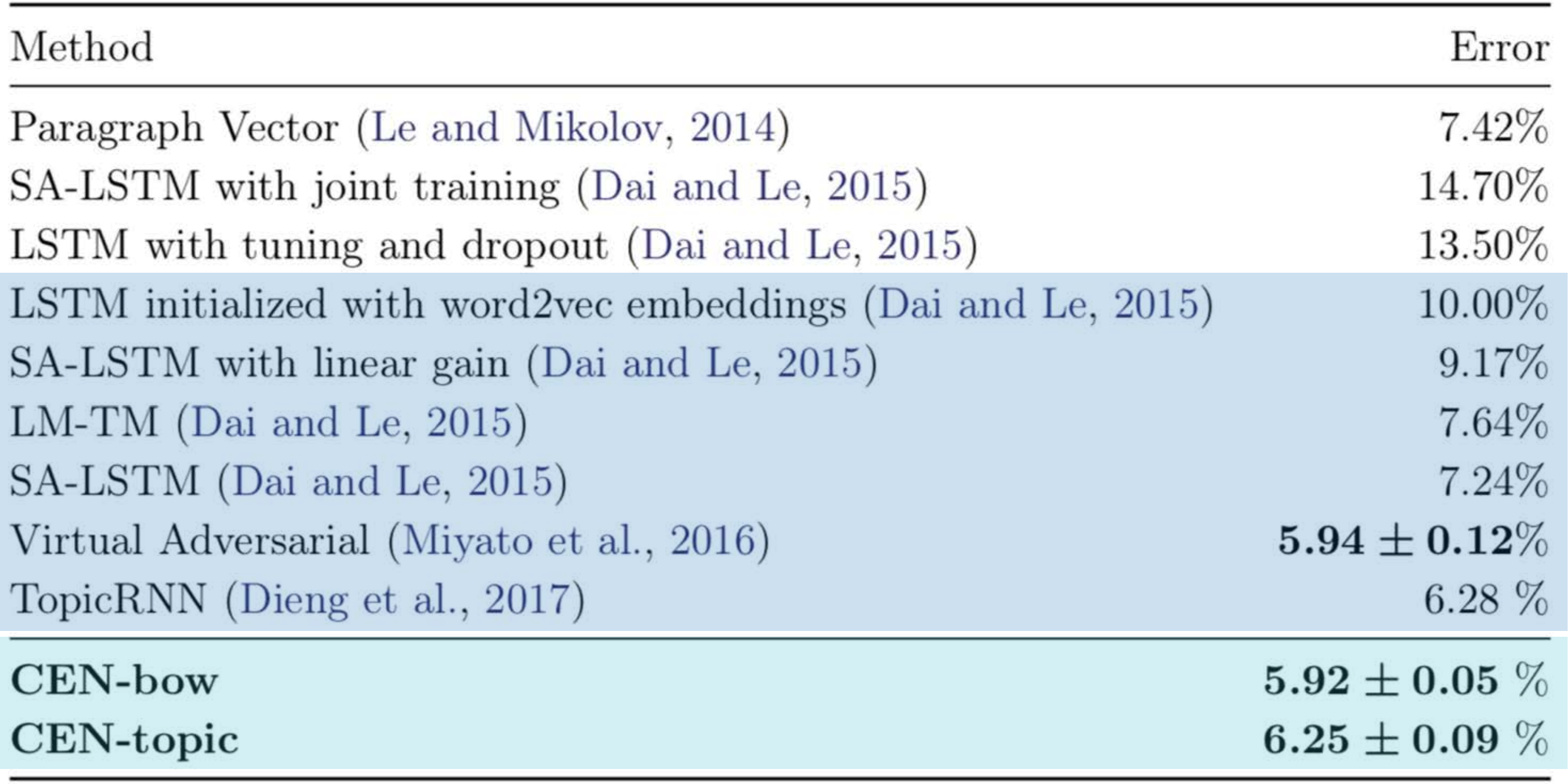


Same performance as vanilla deep networks; no compute overhead.

Predicting via explanation regularizes the model when there is not enough data.

Al-Shedivat, Dubey, Xing, arXiv, 2017

Results: classical image & text datasets



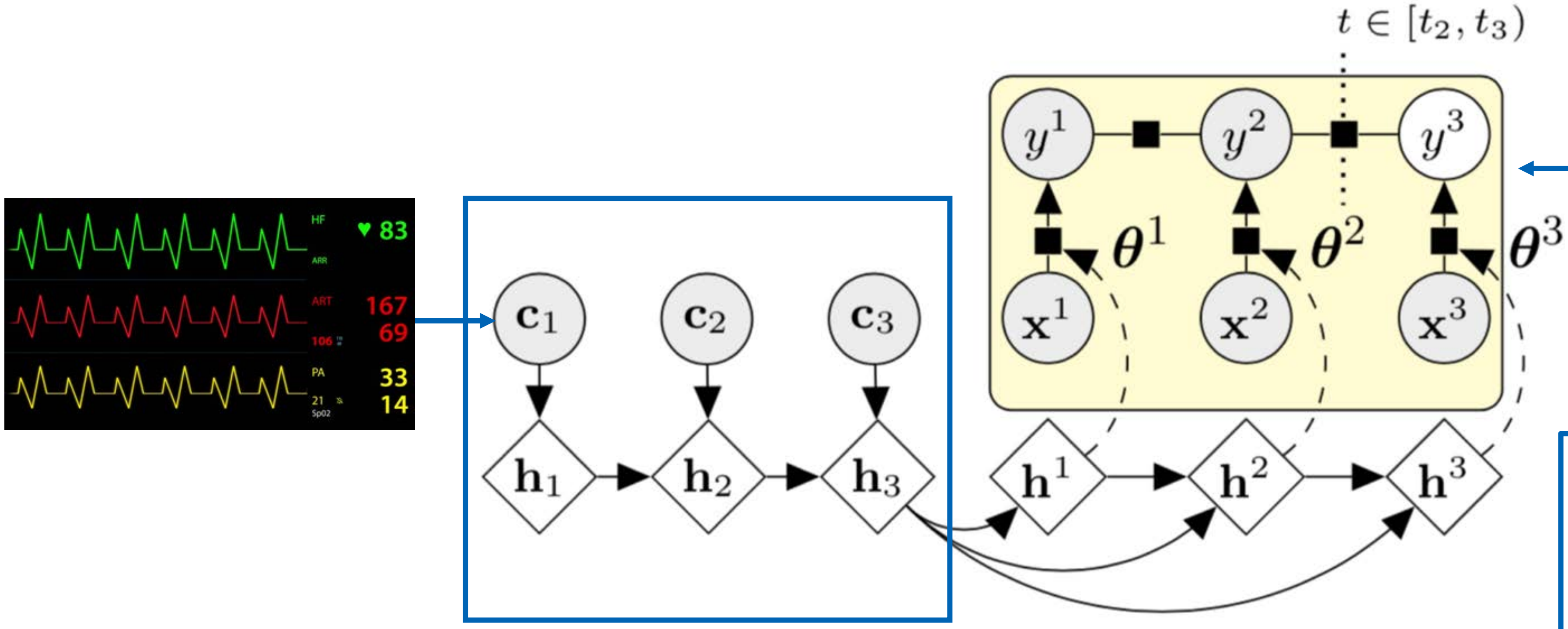
Method	Error
Paragraph Vector (Le and Mikolov, 2014)	7.42%
SA-LSTM with joint training (Dai and Le, 2015)	14.70%
LSTM with tuning and dropout (Dai and Le, 2015)	13.50%
LSTM initialized with word2vec embeddings (Dai and Le, 2015)	10.00%
SA-LSTM with linear gain (Dai and Le, 2015)	9.17%
LM-TM (Dai and Le, 2015)	7.64%
SA-LSTM (Dai and Le, 2015)	7.24%
Virtual Adversarial (Miyato et al., 2016)	**5.94 ± 0.12%**
TopicRNN (Dieng et al., 2017)	6.28 %
CEN-bow	**5.92 ± 0.05** %
CEN-topic	**6.25 ± 0.09** %

Semi-supervised training

Only supervised training (!!)

Al-Shedivat, Dubey, Xing, arXiv, 2017

CEN architectures for survival analysis



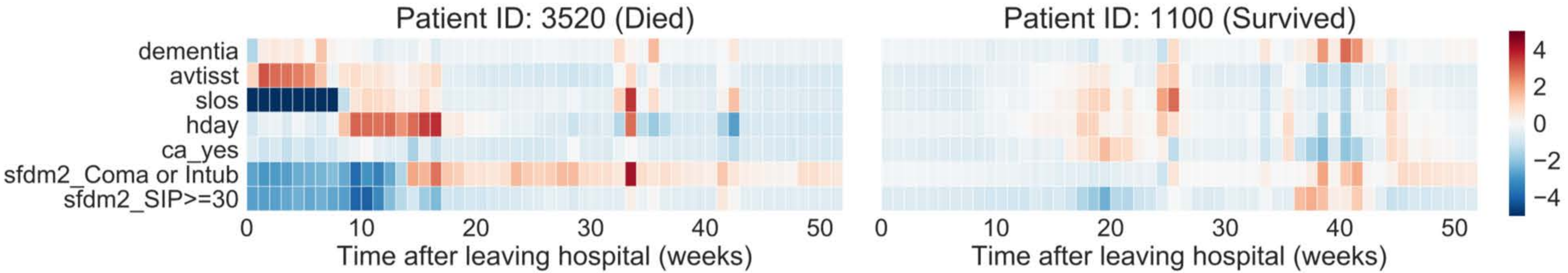
Produce a CRF that predicts a sequence of survival indicators over future time intervals.

Encode a sequence of observations for a patient (e.g., vitals/tests measured in ICU).

Al-Shedivat, Dubey, Xing, arXiv, 2017

Results: survival analysis

SUPPORT2					PhysioNet Challenge 2012				
Model	**Acc@25**	**Acc@50**	**Acc@75**	**RAE**	**Model**	**Acc@25**	**Acc@50**	**Acc@75**	**RAE**
Cox	84.1	73.7	47.6	0.90	Cox	93.0	69.6	49.1	0.24
Aalen	87.1	66.2	45.8	0.98	Aalen	93.3	78.7	57.1	0.31
CRF	84.4	89.3	79.2	0.59	CRF	93.2	85.1	65.6	0.14
MLP-CRF	**87.7**	89.6	80.1	0.62	LSTM-CRF	93.9	86.3	68.1	**0.11**
MLP-CEN	85.5	**90.8**	**81.9**	**0.56**	LSTM-CEN	**94.8**	**87.5**	**70.1**	**0.09**



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)
- ❑ 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
- ❑ Bayesian Learning of NNs
  - ❑ Bayesian learning of NN parameters
  - ❑ Deep kernel learning

Bayesian learning of NNs

- A neural network as a probabilistic model:
  - Likelihood: $p(y|x,\boldsymbol{\theta})$
    - Categorical distribution for classification ⇒ cross-entropy loss
    - Gaussian distribution for regression ⇒ squared loss
  - Prior on parameters: $p(\boldsymbol{\theta})$
- Maximum a posteriori (MAP) solution:
  - $\boldsymbol{\theta}^{MAP} = \mathrm{argmax}_{\theta} \log p(y|x,\boldsymbol{\theta})p(\boldsymbol{\theta})$
  - Gaussian prior ⇒ L2 regularization
  - Laplace prior ⇒ L1 regularization
- Bayesian learning [MacKay 1992, Neal 1996, de Freitas 2003]
  - Posterior: $p(\boldsymbol{\theta}|x,y)$
  - Variational inference with approximate posterior $q(\boldsymbol{\theta})$

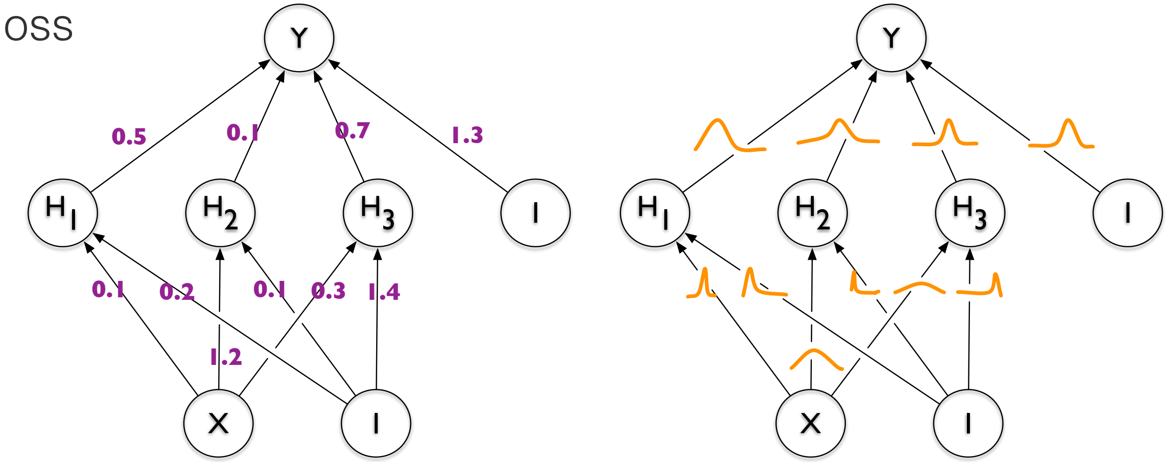


Figure courtesy: Blundell et al, 2016

Bayesian learning of NNs

- Variational inference (in a nutshell):

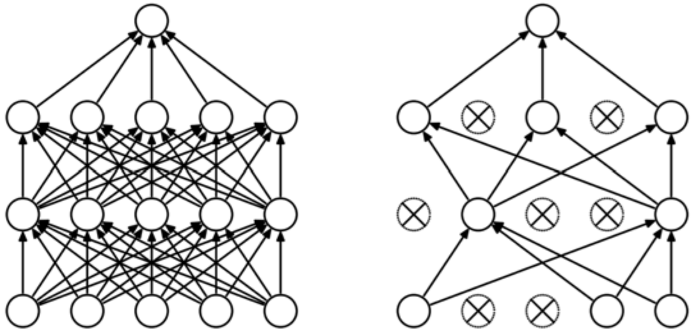
$$\min_q F(D,\boldsymbol{\theta}) = \mathrm{KL}\big(q(\boldsymbol{\theta})\,||\,p(\boldsymbol{\theta}|D)\big) - \mathrm{E}_{q(\boldsymbol{\theta})}[\log p(D|\boldsymbol{\theta})]$$

$$\min_q F(D,\boldsymbol{\theta}) = \mathrm{KL}\big(q(\boldsymbol{\theta})\,||\,p(\boldsymbol{\theta}|D)\big) - \sum_i \log p(D|\boldsymbol{\theta}_i)$$

  where $\boldsymbol{\theta}_i \sim q(\boldsymbol{\theta})$; KL term can be approximated similarly

- We can define $q(\boldsymbol{\theta})$ as a diagonal Gaussian or full-covariance Gaussian
- Alternatively, $q(\boldsymbol{\theta})$ can be defined implicitly, e.g. via dropout [Gal & Ghahramani, 2016]

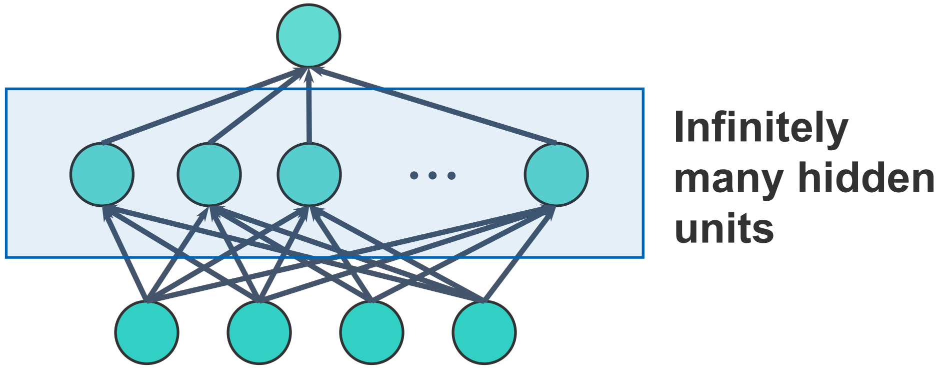
$$\boldsymbol{\theta} = \boldsymbol{M} \cdot \mathrm{diag}(\boldsymbol{z}),$$
$$\boldsymbol{z} \sim \mathrm{Bernoulli}(p)$$



- Dropping out neurons is equivalent to zeroing out columns of the parameter matrices (i.e., weights)
- $z_i = 0$ corresponds to i-th column of $\boldsymbol{M}$ being dropped out
  ⇒ the procedure is equivalent to dropout of unit $\boldsymbol{i}$ [Hinton et al., 2012]
- Variational parameters are $\{\boldsymbol{M}, \boldsymbol{p}\}$

“Infinitely Wide” Deep Models

- We have seen that an ”infinitely deep” network can be explained by a proper GM, How about an “infinitely wide” one?
- Consider a neural network with a Gaussian prior on its weights an infinitely many hidden neurons in the intermediate layer.
- Turns out, if we have a certain Gaussian prior on the weights of such infinite network, it will be equivalent to a Gaussian process [Neal 1996].



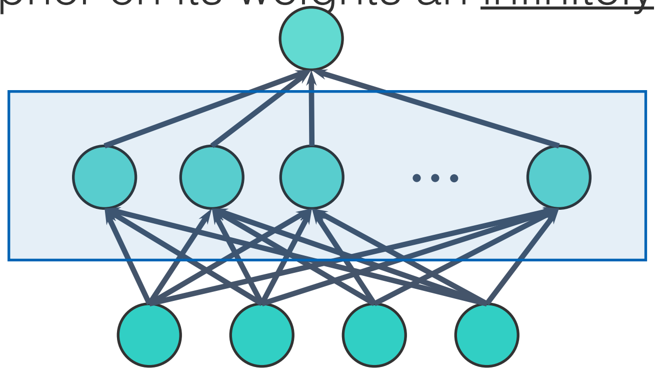
- Gaussian process (GP) is a distribution over functions:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x},\mathbf{x}') &= \mathbb{E}[(f(\mathbf{x})-m(\mathbf{x}))(f(\mathbf{x}')-m(\mathbf{x}'))], \\ f(\mathbf{x}) &\sim \mathcal{GP}\big(m(\mathbf{x}),k(\mathbf{x},\mathbf{x}')\big). \end{aligned}$$

- When used for prediction, GPs account for correlations between the data points and can output well-calibrated predictive uncertainty estimates.

Gaussian Process and Deep Kernel Learning

- Consider a neural network with a Gaussian prior on its weights an infinitely many hidden neurons in the intermediate layer.



Infinitely many hidden units

- Certain classes of Gaussian priors for neural networks with infinitely many hidden units converge to Gaussian processes [Neal 1996]
- Deep kernel [Wilson et al., 2016]
  - Combines the inductive biases of deep model architectures with the non-parametric flexibility of Gaussian processes

$$k\left(x_i, x_j \middle| \phi\right) \rightarrow k(g(x_i, \theta), g(x_j, \theta)|\phi, \theta) \quad \text{where } K_{ij} = k(x_i, x_j)$$

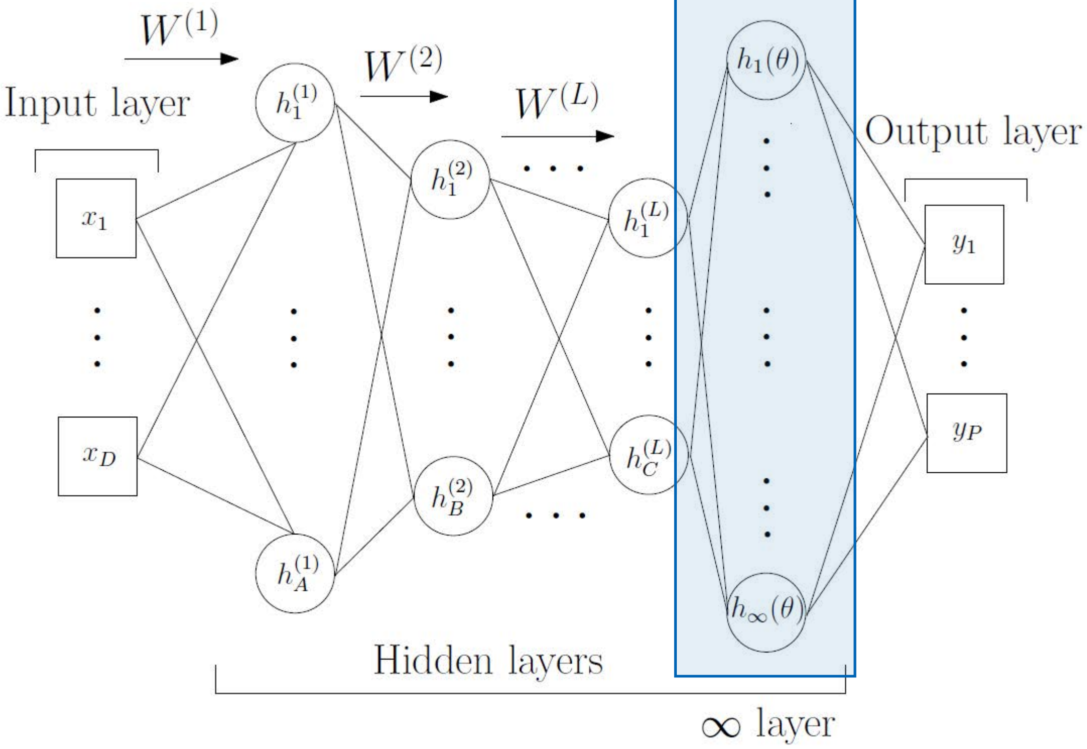
  - Starting from a base kernel $k(x_i, x_j|\phi)$, transform the inputs x as

$$p(f|\phi) = \mathcal{N}(f|m(x), K)$$
$$p(y|f) = \mathcal{N}(y|f, \beta^{-1})$$

  - Learn both kernel and neural parameters $\{\phi, \theta\}$ jointly by optimizing marginal log-likelihood (or its variational lower-bound).
  - Fast learning and inference with local kernel interpolation, structured inducing points, and Monte Carlo approximations

Gaussian Process and Deep Kernel Learning

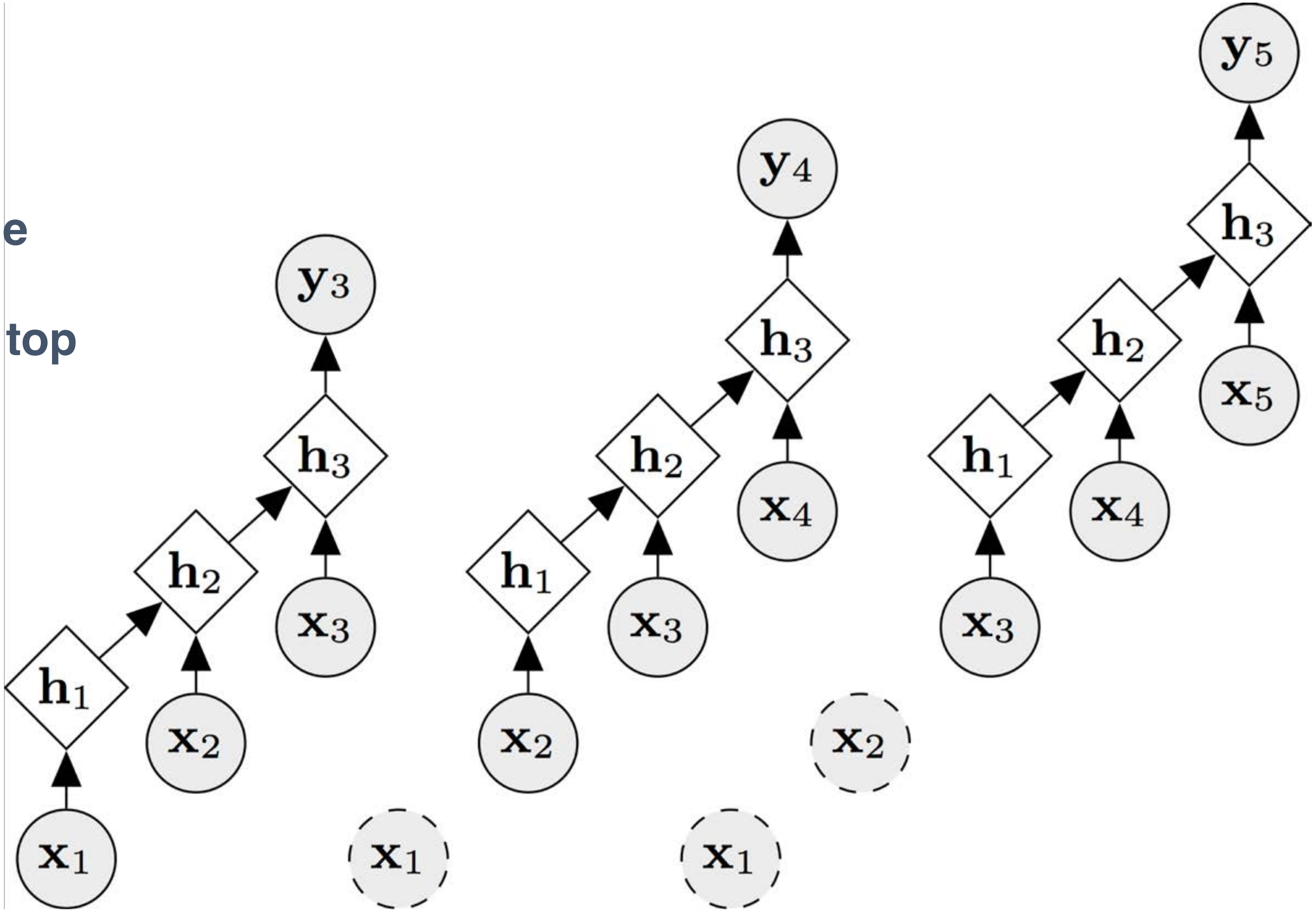
- By adding GP as a layer to a deep neural net, we can think of it as adding an infinite hidden layer with a particular prior on the weights
- Deep kernel learning [Wilson et al., 2016]
  - Combines the inductive biases of deep models with the non-parametric flexibility of Gaussian processes
  - GPs add powerful regularization to the network
  - Additionally, they provide predictive uncertainty estimates



Deep kernel learning on sequential data

What if we have data of sequential nature?

Can we still apply the same reasoning and build rich nonparametric models on top recurrent nets?

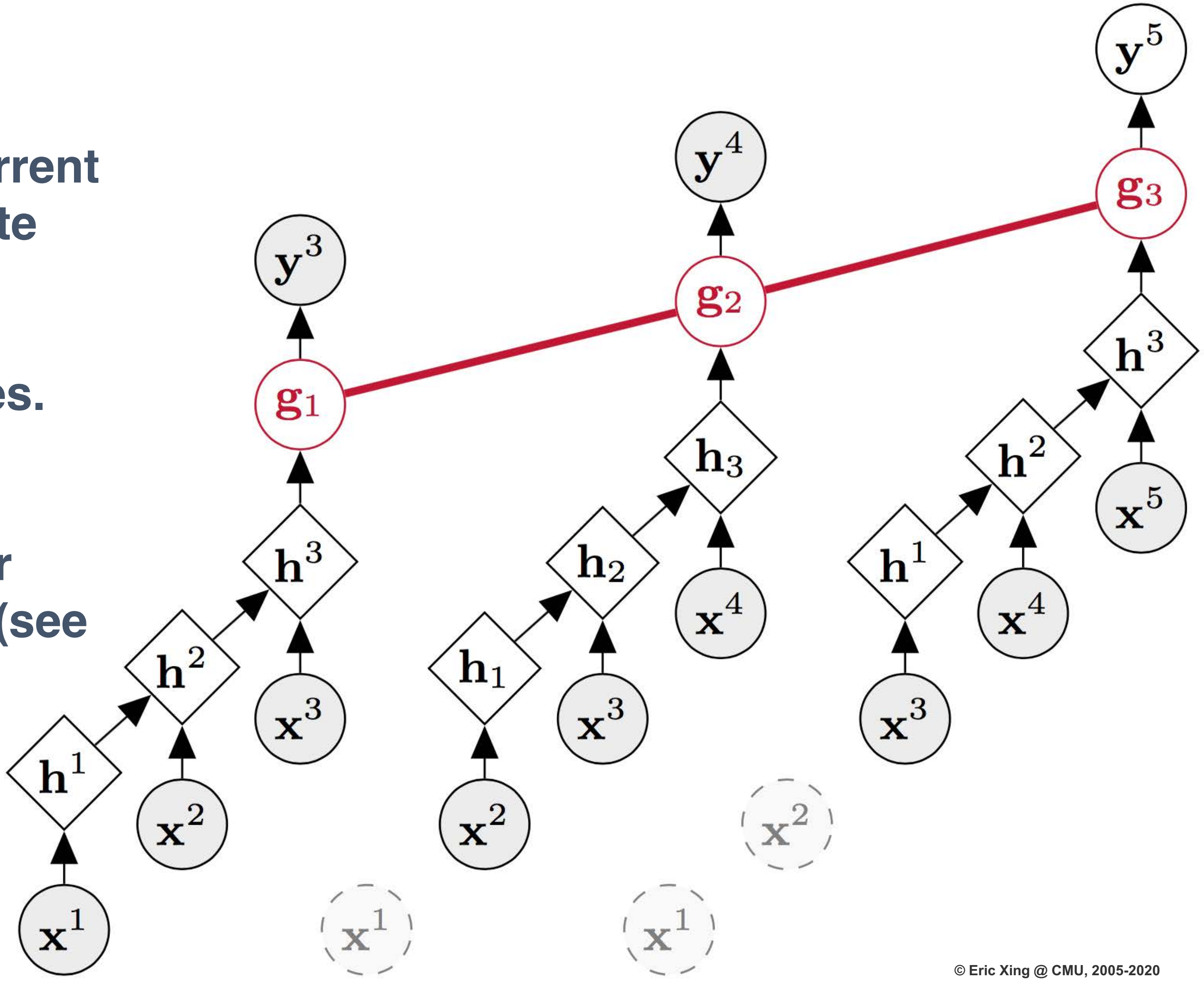


Deep kernel learning on sequential data

The answer is YES!

By adding a GP layer to a recurrent network, we effectively correlate samples across time and get predictions along with well calibrated uncertainty estimates.

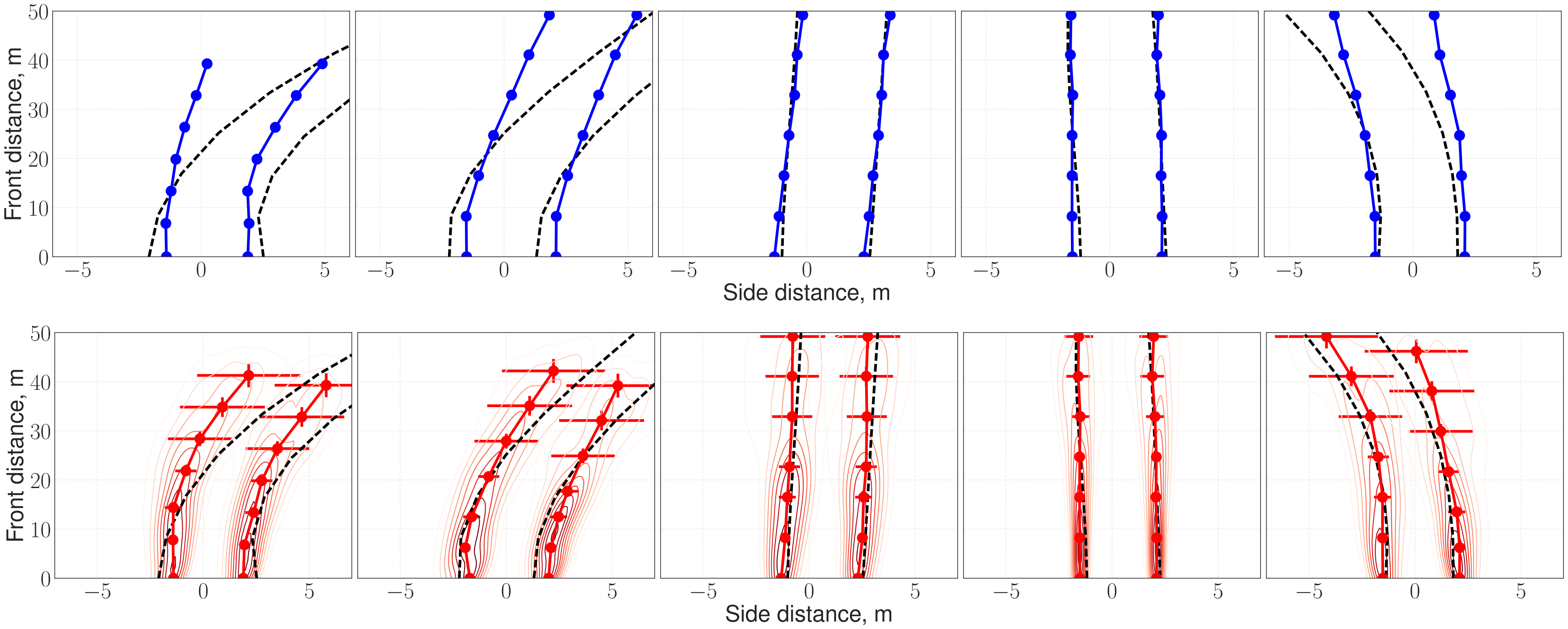
To train such model using stochastic techniques however requires some additional care (see our paper).



Al-Shedivat et al., JMLR, 2017

Deep kernel learning on sequential data

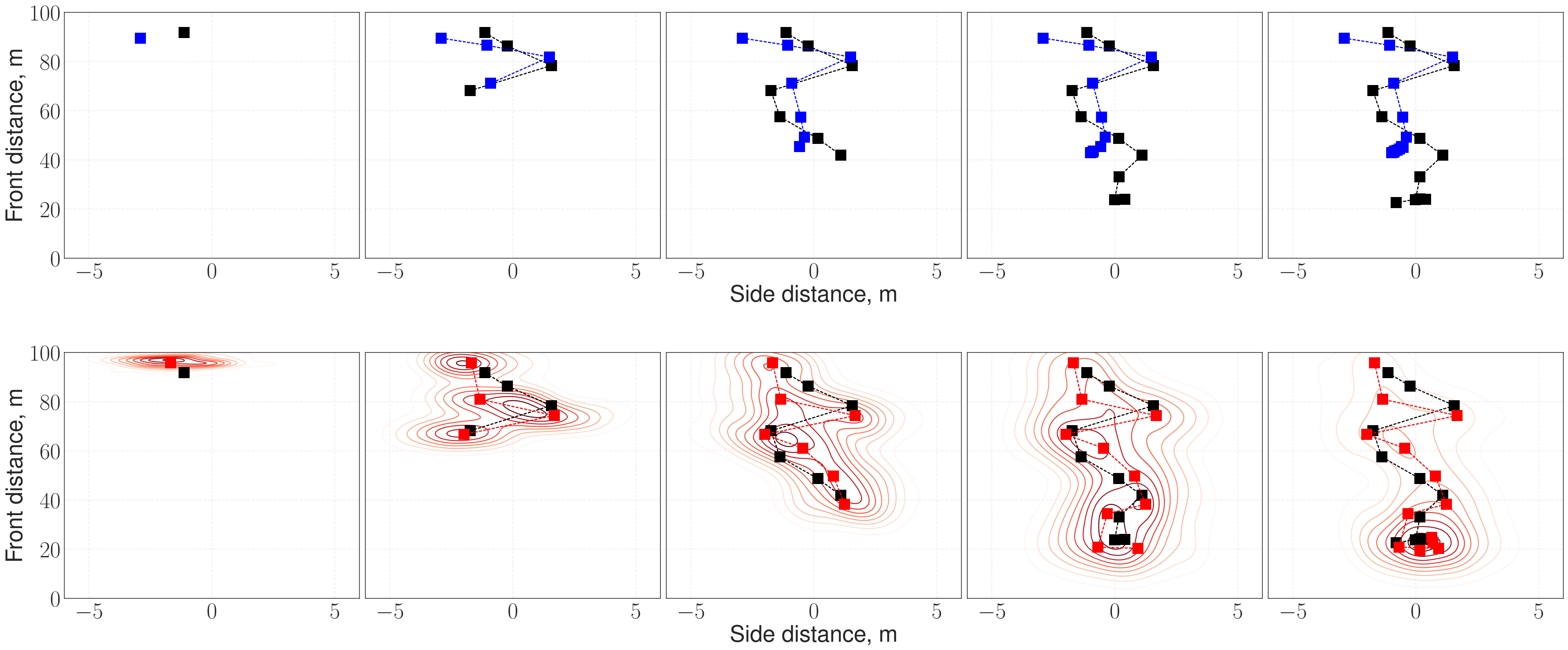
Lane prediction: LSTM vs GP-LSTM



Al-Shedivat et al., JMLR, 2017

Deep kernel learning on sequential data

Lead vehicle prediction: LSTM vs GP-LSTM



Al-Shedivat et al., JMLR, 2017