Neural networks

Deep learning - motivation

NEURAL NETWORK

Topics: multilayer neural network

- Could have L hidden layers:
  - layer input activation for $k>0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

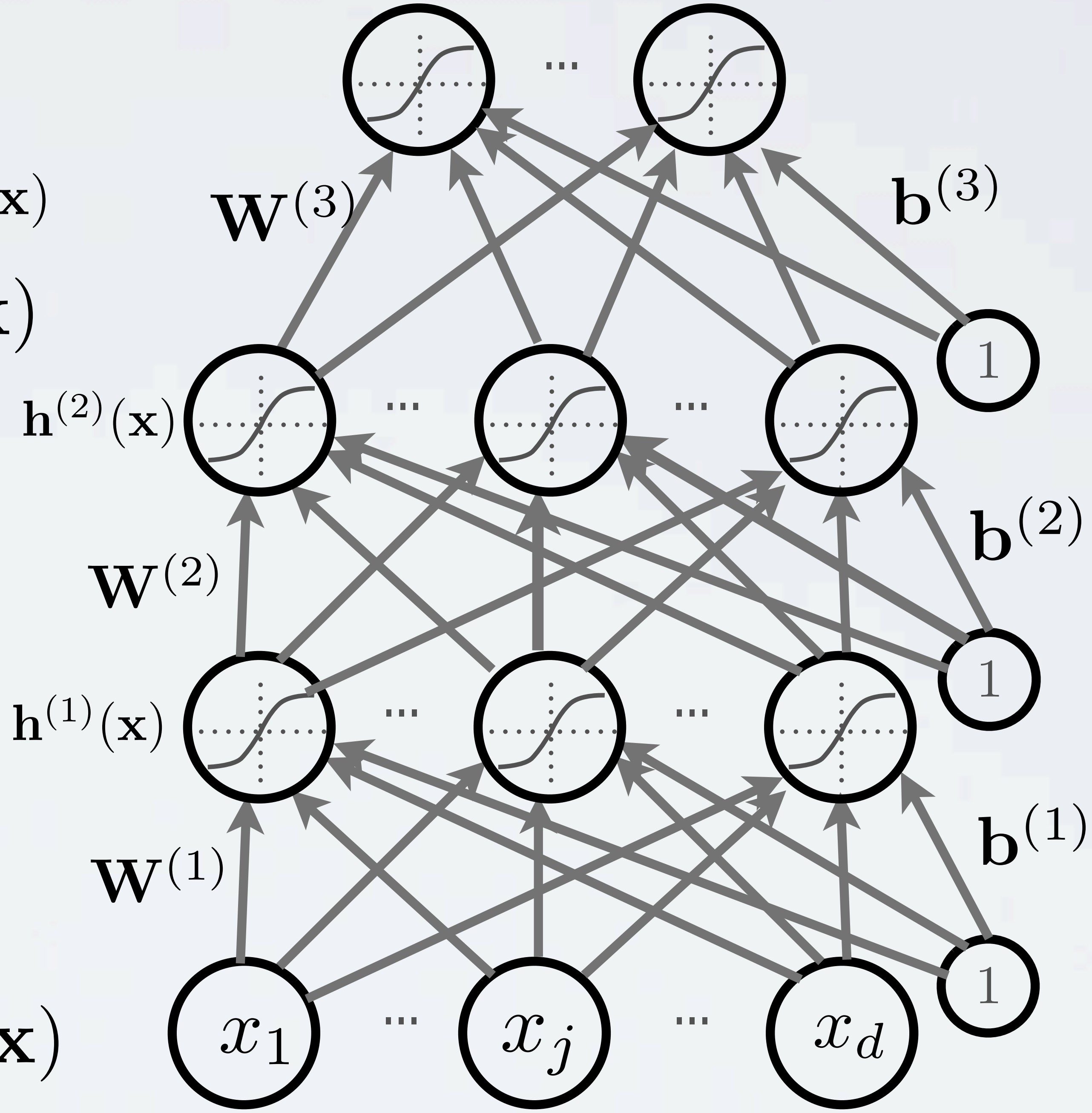
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation (k=L+1):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



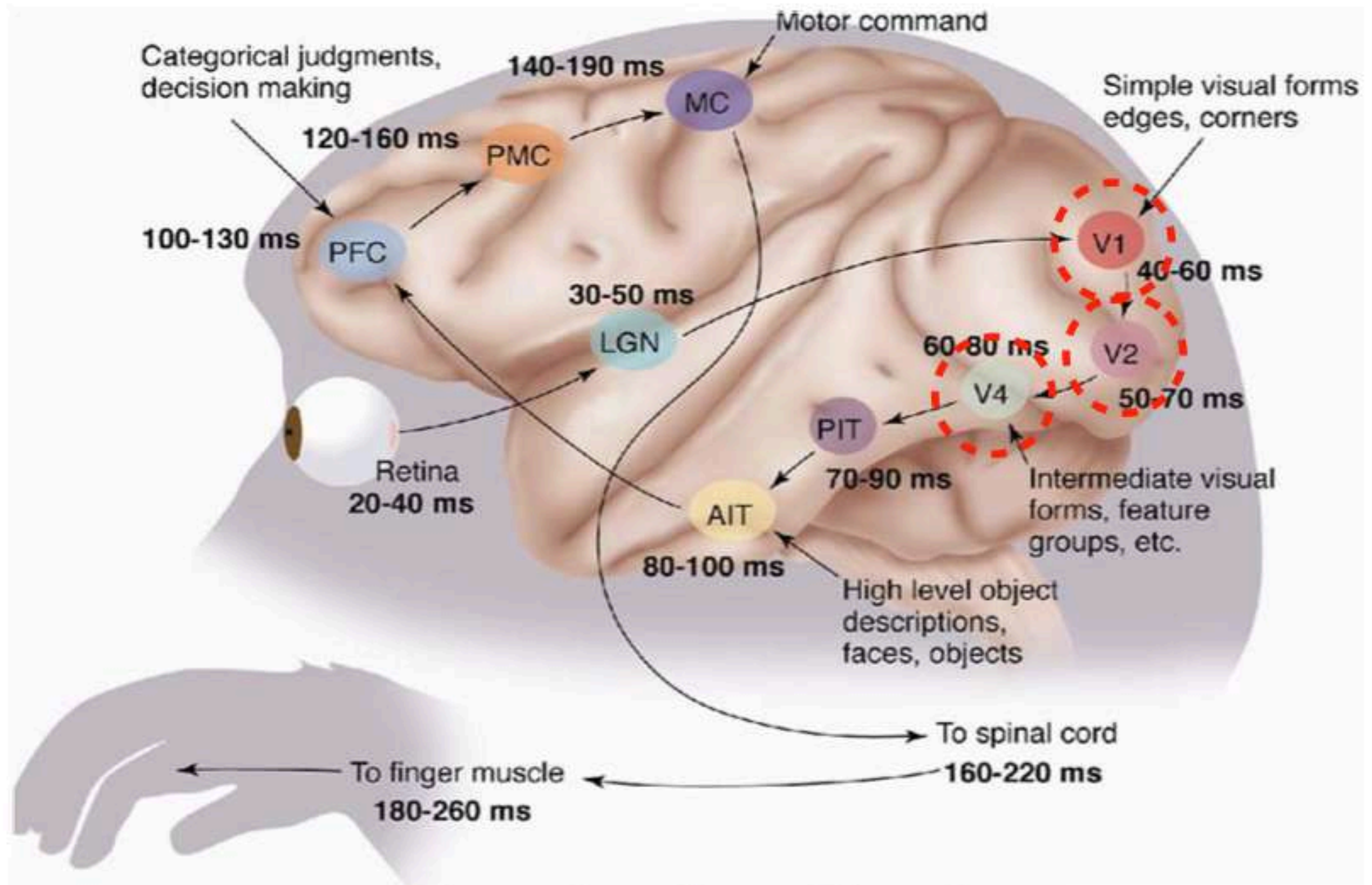
DEEP LEARNING

Topics: deep learning, distributed representation

- Deep learning is research on learning models with multilayer representations
  - multilayer (feed-forward) neural network
  - multilayer graphical model (deep belief network, deep Boltzmann machine)
- Each layer corresponds to a "distributed representation"
  - units in layer are not mutually exclusive
    - each unit is a separate feature of the input
    - two units can be "active" at the same time
  - they do not correspond to a partitioning (clustering) of the inputs
    - in clustering, an input can only belong to a single cluster

DEEP LEARNING

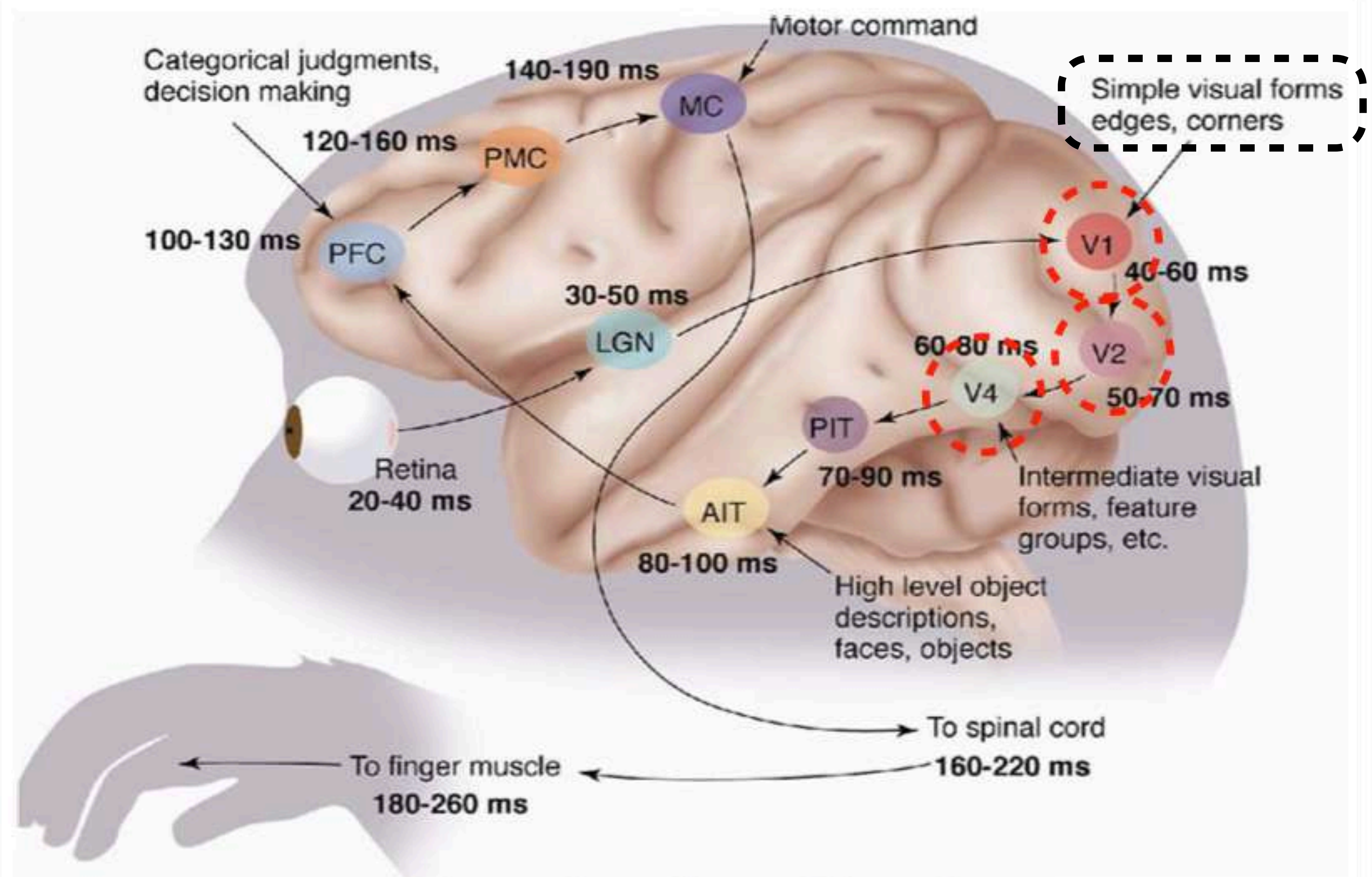
Topics: inspiration from visual cortex



[picture from Simon Thorpe]

DEEP LEARNING

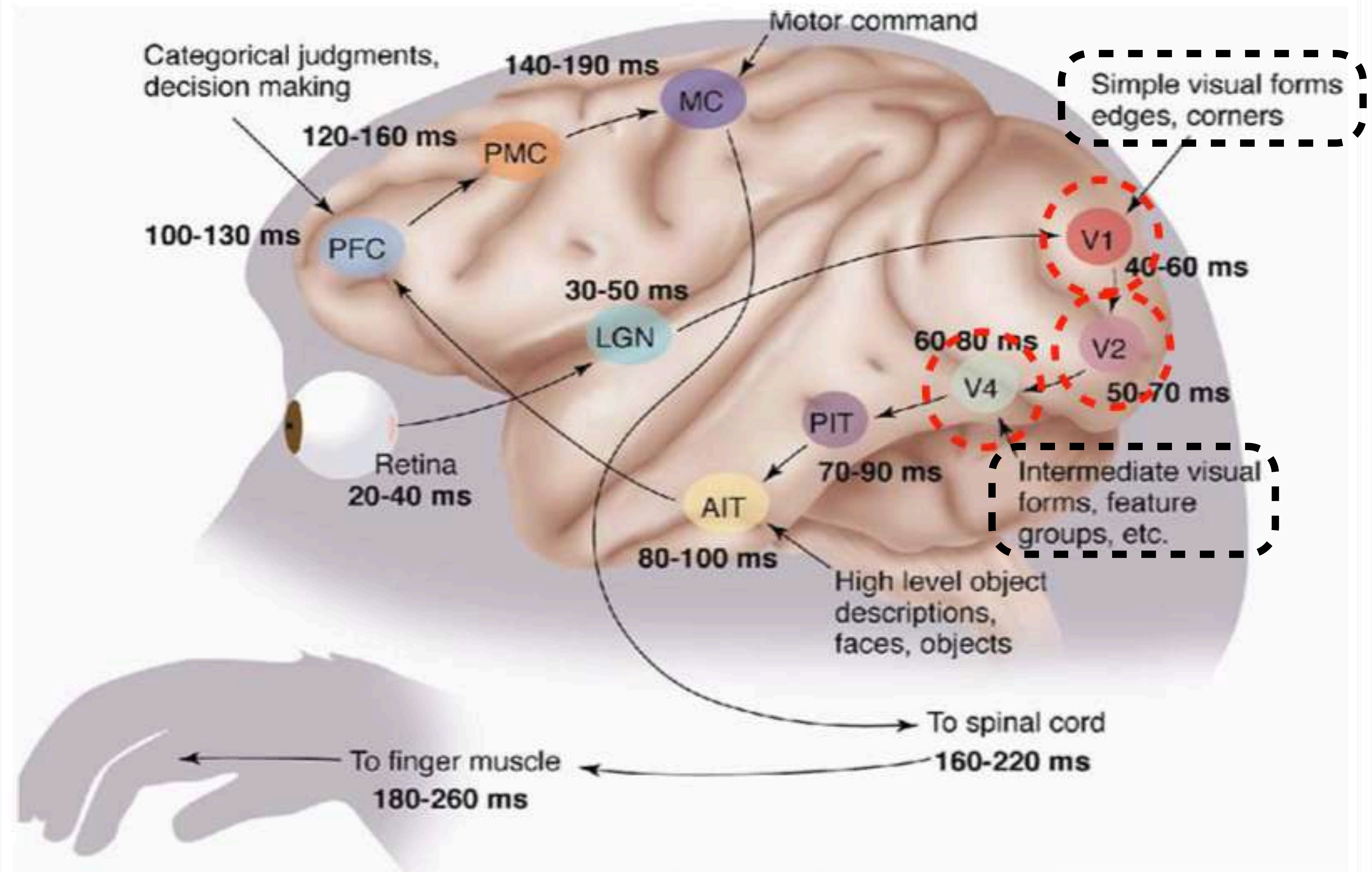
Topics: inspiration from visual cortex



[picture from Simon Thorpe]

DEEP LEARNING

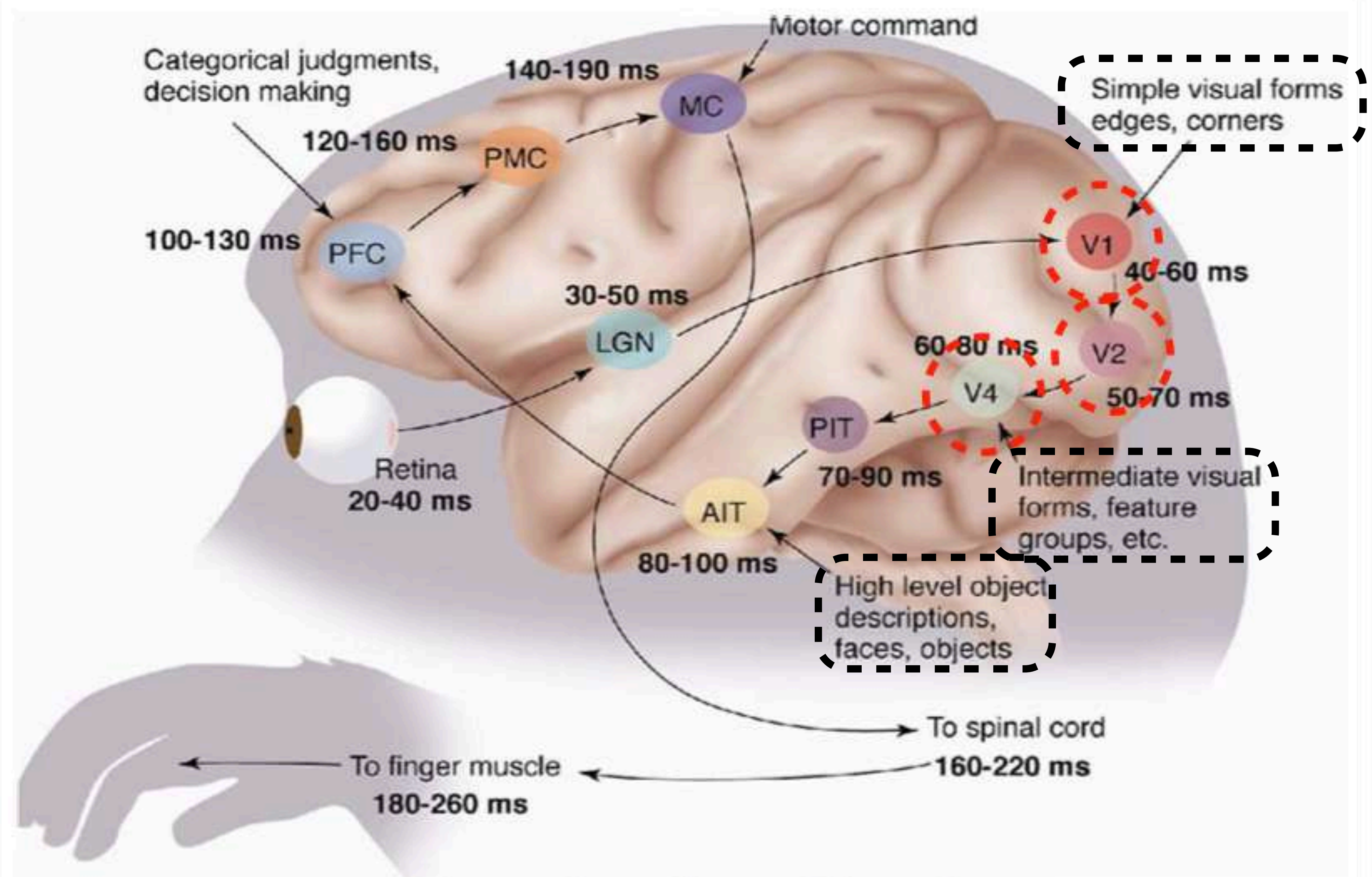
Topics: inspiration from visual cortex



[picture from Simon Thorpe]

DEEP LEARNING

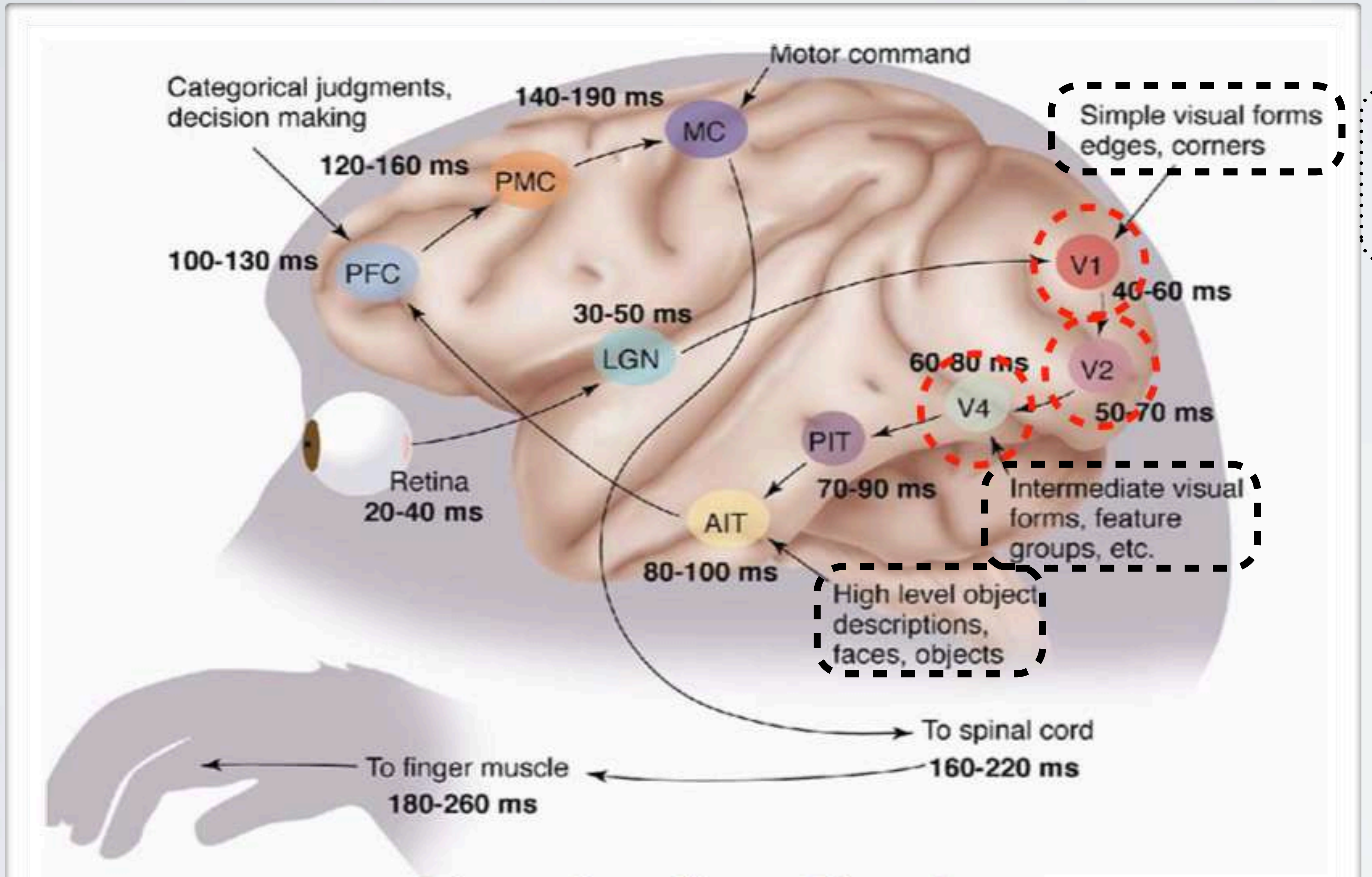
Topics: inspiration from visual cortex



[picture from Simon Thorpe]

DEEP LEARNING

Topics: inspiration from visual cortex



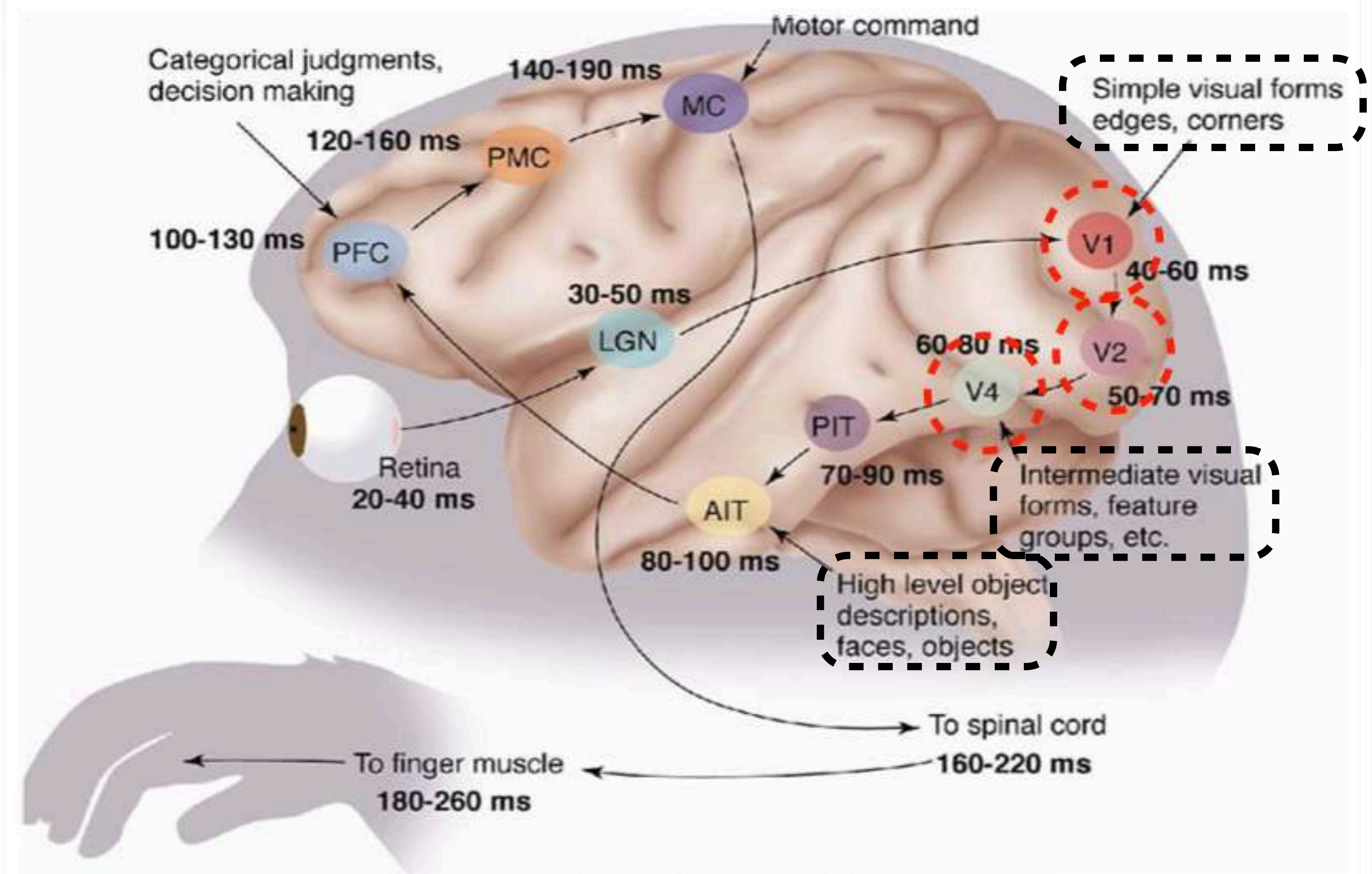
[picture from Simon Thorpe]

edges

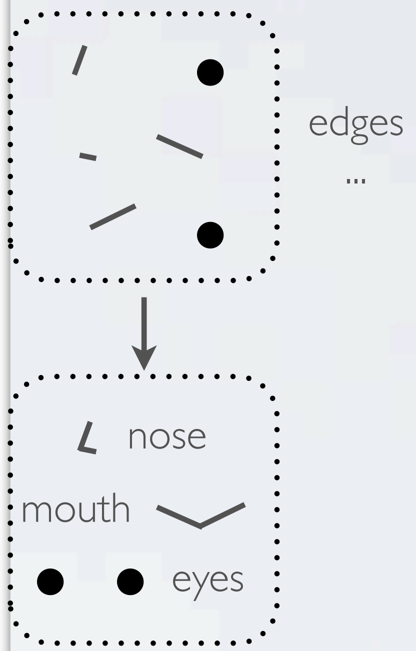
...

DEEP LEARNING

Topics: inspiration from visual cortex

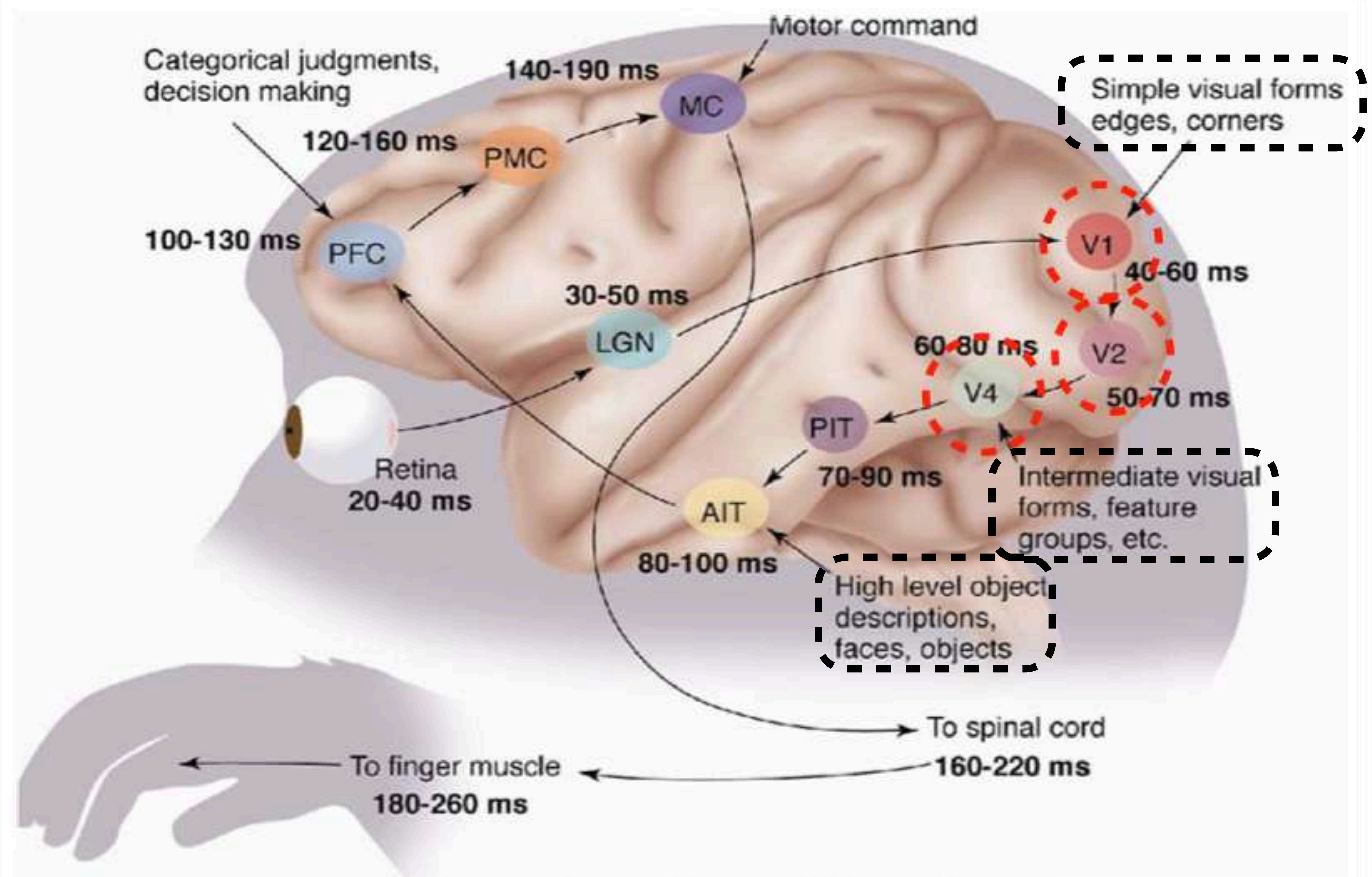


[picture from Simon Thorpe]

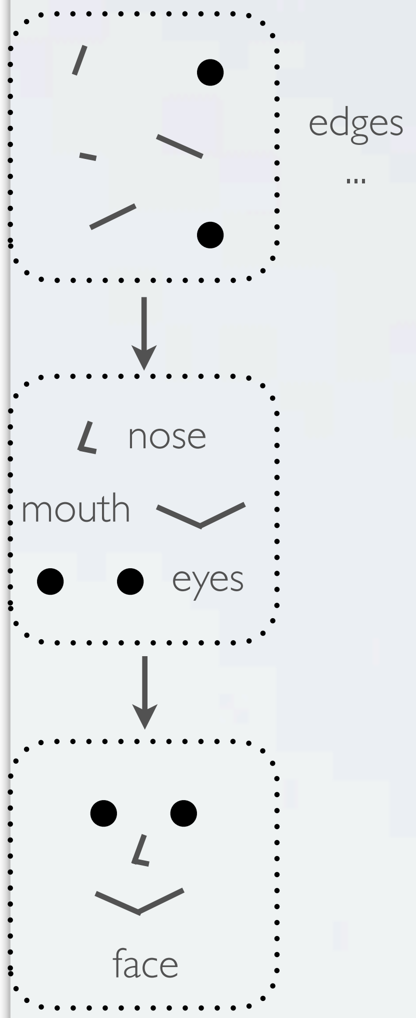


DEEP LEARNING

Topics: inspiration from visual cortex



[picture from Simon Thorpe]



DEEP LEARNING

Topics: theoretical justification

- A deep architecture can represent certain functions (exponentially) more compactly
- Example: Boolean functions
  - a Boolean circuit is a sort of feed-forward network where hidden units are logic gates (i.e. AND, OR or NOT functions of their arguments)
  - any Boolean function can be represented by a "single hidden layer" Boolean circuit
    - however, it might require an exponential number of hidden units
  - it can be shown that there are Boolean functions which
    - require an exponential number of hidden units in the single layer case
    - require a polynomial number of hidden units if we can adapt the number of layers
  - See "Exploring Strategies for Training Deep Neural Networks" for a discussion

DEEP LEARNING

Topics: success stories (Microsoft Research)

Microsoft Research

Search Microsoft Research

Home | Our Research | Connections | Careers | Hub

About Us | News | Media Resources | Events | Community

> News > Speech Recognition Leaps Forward

Speech Recognition Leaps Forward

By Janie Chang
August 29, 2011 12:01 AM PT

During Interspeech 2011, the 12th annual Conference of the International Speech Communication Association being held in Florence, Italy, from Aug. 28 to 31, researchers from Microsoft Research will present work that dramatically improves the potential of real-time, speaker-independent, automatic speech recognition.

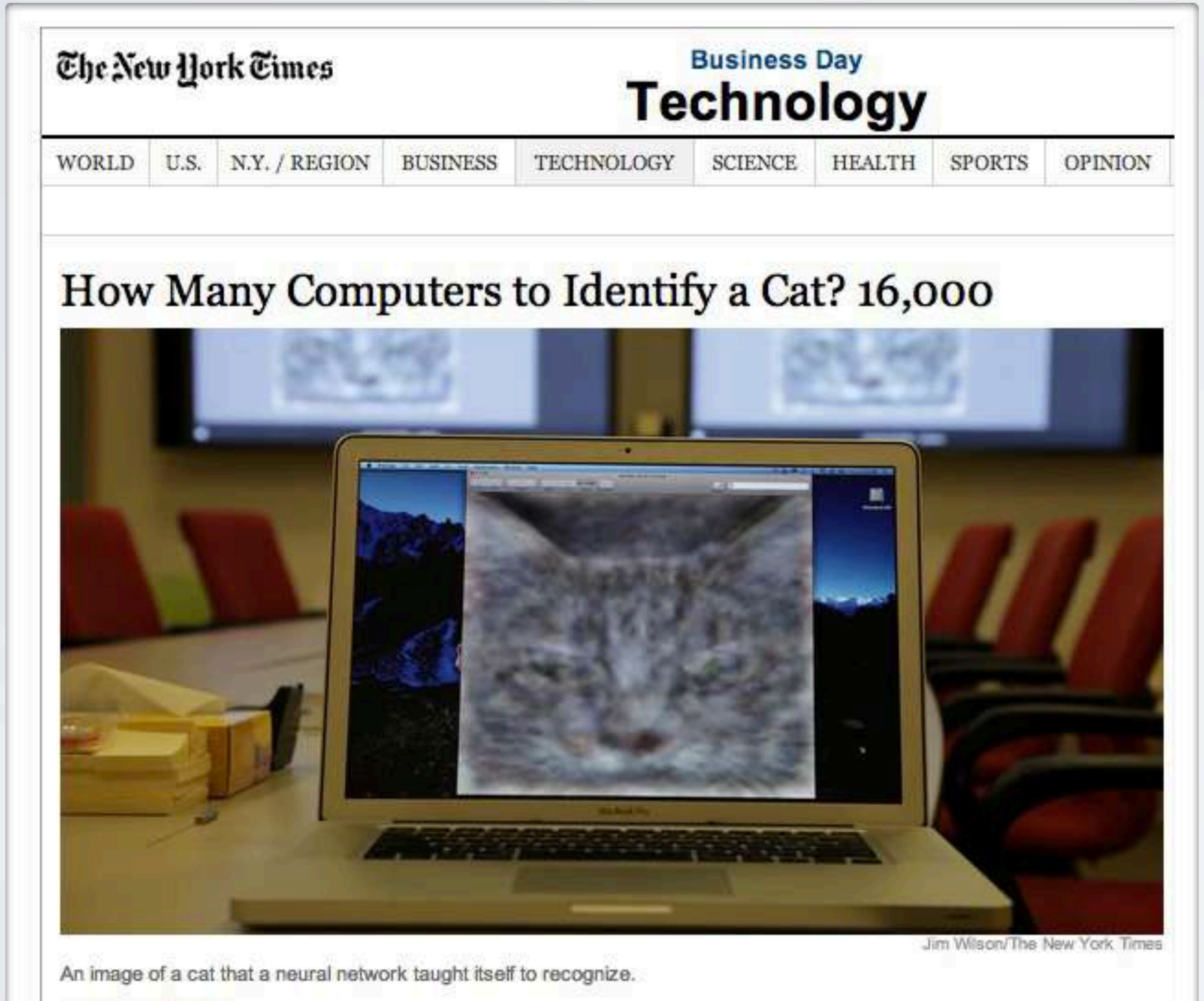
Dong Yu, researcher at Microsoft Research Redmond, and Frank Seide, senior researcher and research manager with Microsoft Research Asia, have been spearheading this work, and their teams have collaborated on what has developed into a research breakthrough in the use of artificial neural networks for large-vocabulary speech recognition.

The Holy Grail of Speech Recognition

Commercially available speech-recognition technology is behind applications such

DEEP LEARNING

Topics: success stories (Google)



The New York Times

Business Day

Technology

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION

How Many Computers to Identify a Cat? 16,000

Jim Wilson/The New York Times

An image of a cat that a neural network taught itself to recognize.

Neural networks

Deep learning - difficulty of training

NEURAL NETWORK

Topics: multilayer neural network

- Could have L hidden layers:
  - layer input activation for $k{>}0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

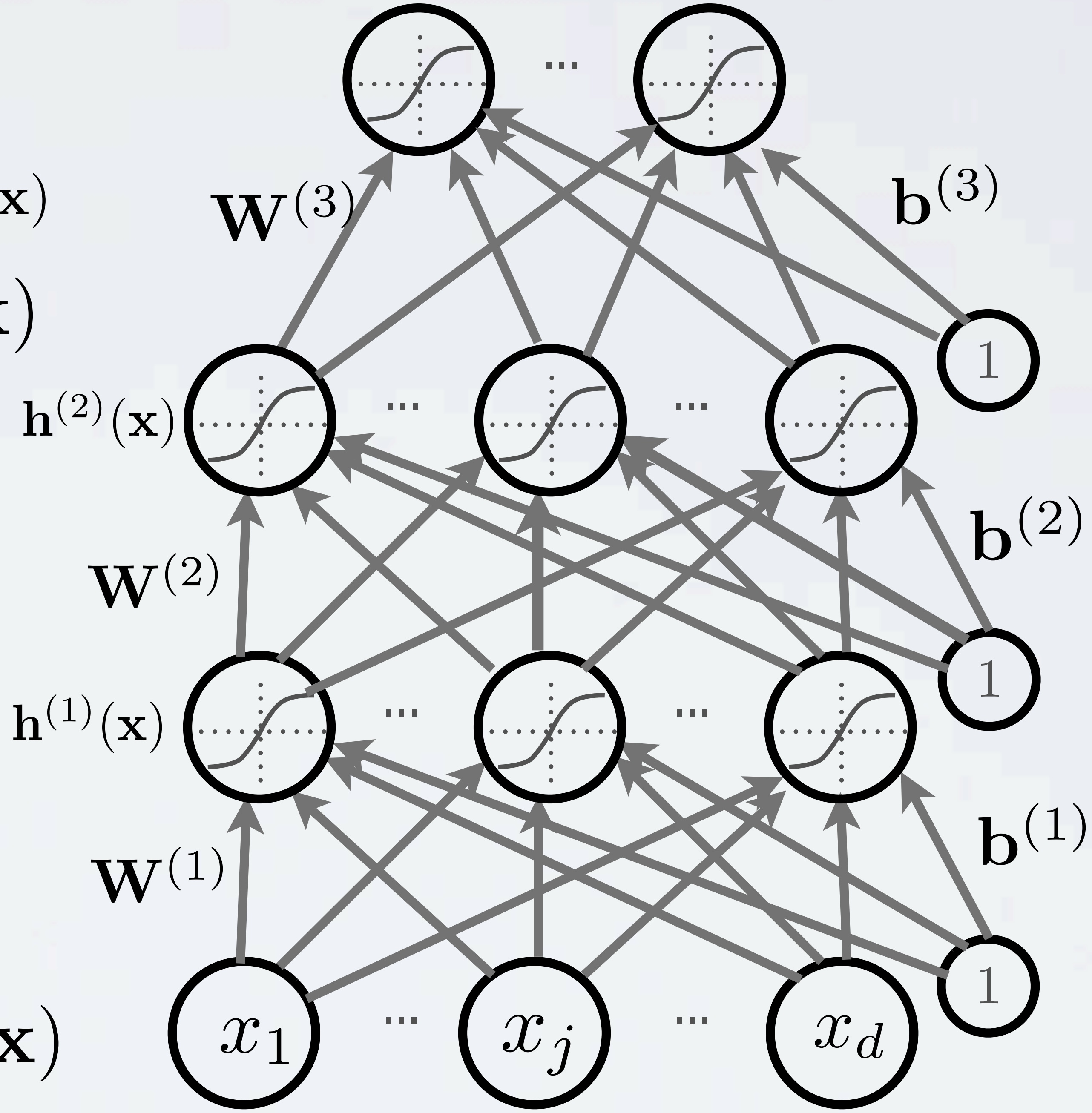
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation ($k{=}L{+}1$):

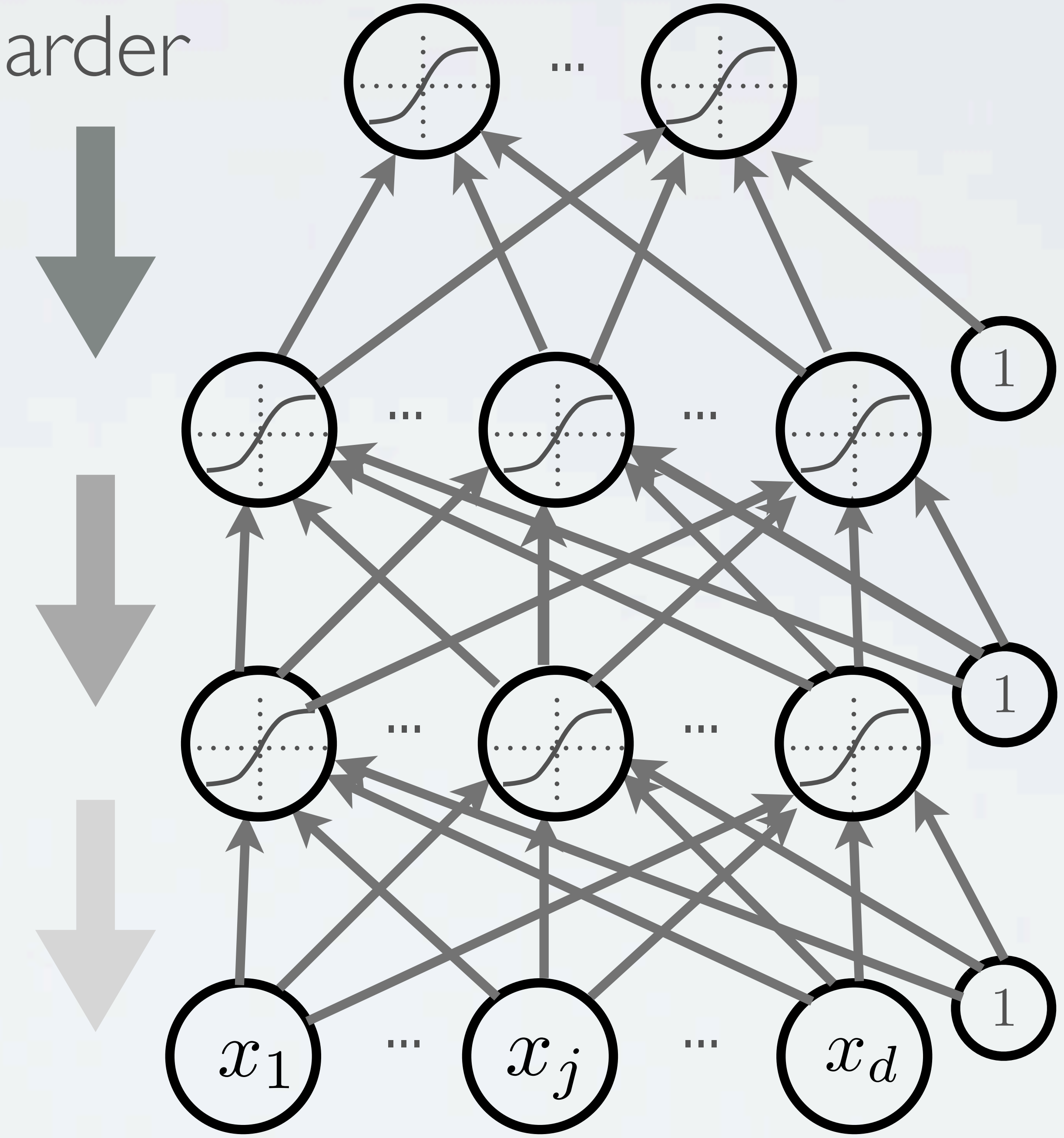
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



DEEP LEARNING

Topics: why training is hard

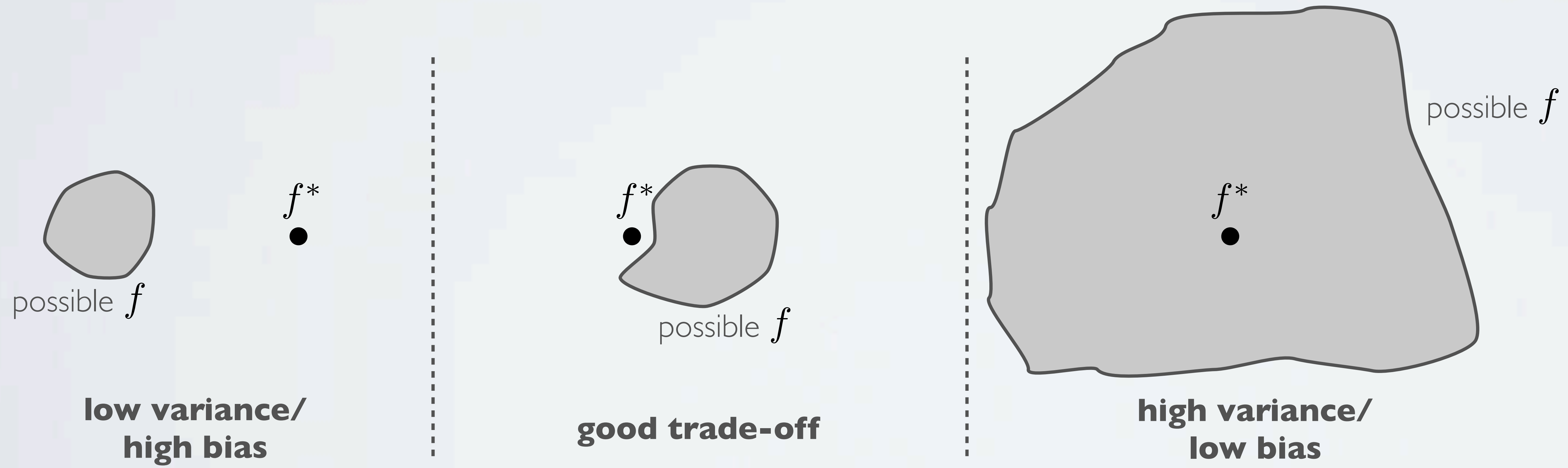
- First hypothesis: optimization is harder (underfitting)
    - vanishing gradient problem
    - saturated units block gradient propagation
- This is a well known problem in recurrent neural networks



DEEP LEARNING

Topics: why training is hard

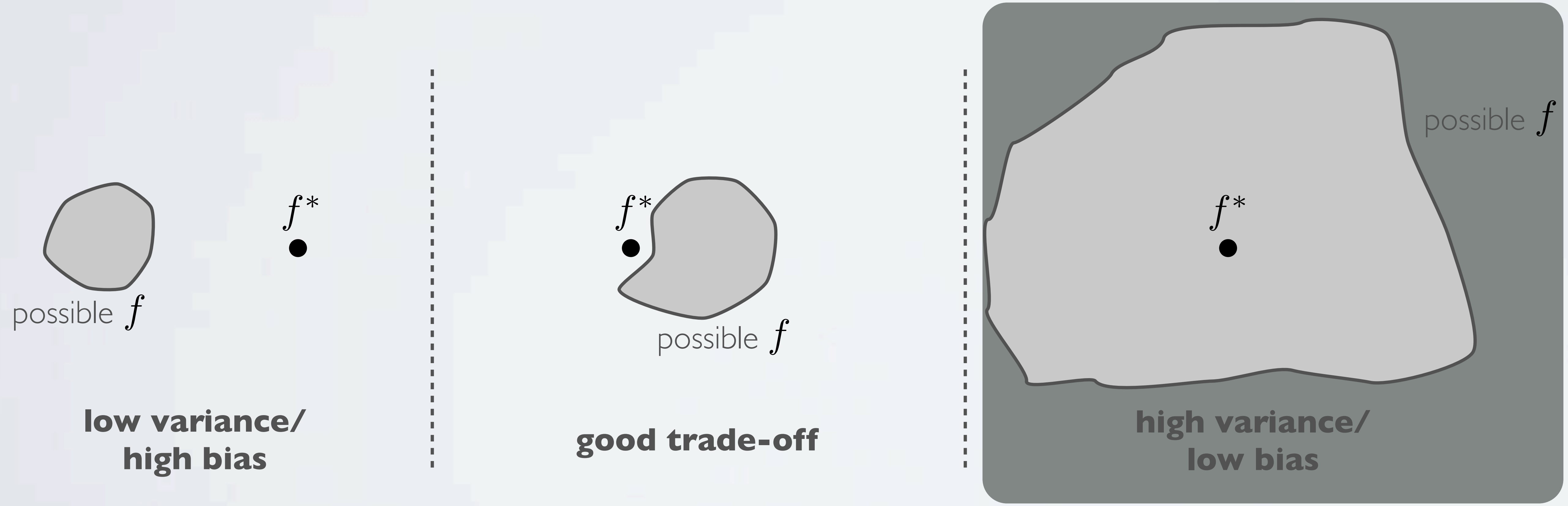
- Second hypothesis: overfitting
  - we are exploring a space of complex functions
  - deep nets usually have lots of parameters
- Might be in a high variance / low bias situation



DEEP LEARNING

Topics: why training is hard

- Second hypothesis: overfitting
  - we are exploring a space of complex functions
  - deep nets usually have lots of parameters
- Might be in a high variance / low bias situation



DEEP LEARNING

Topics: why training is hard

- Depending on the problem, one or the other situation will tend to dominate

- If first hypothesis (underfitting): use better optimization
    - this is an active area of research

- If second hypothesis (overfitting): use better regularization
    - unsupervised learning
    - stochastic «dropout» training

Neural networks

Deep learning - unsupervised pre-training

DEEP LEARNING

Topics: why training is hard

- Depending on the problem, one or the other situation will tend to prevail

- If first hypothesis (underfitting): use better optimization
  - this is an active area of research

- If second hypothesis (overfitting): use better regularization
  - unsupervised learning
  - stochastic «dropout» training

UNSUPERVISED PRE-TRAINING

Topics: unsupervised pre-training

- Solution: initialize hidden layers using unsupervised learning
  - force network to represent latent structure of input distribution

character image

random image

  - encourage hidden layers to encode that structure

UNSUPERVISED PRE-TRAINING

Topics: unsupervised pre-training

- Solution: initialize hidden layers using unsupervised learning
  - force network to represent latent structure of input distribution

character image

Why is one a character and the other is not ?

random image

  - encourage hidden layers to encode that structure

UNSUPERVISED PRE-TRAINING

Topics: unsupervised pre-training

- Solution: initialize hidden layers using unsupervised learning
  - this is a harder task than supervised learning (classification)

Why is one
a character
and the other
is not ?

character image

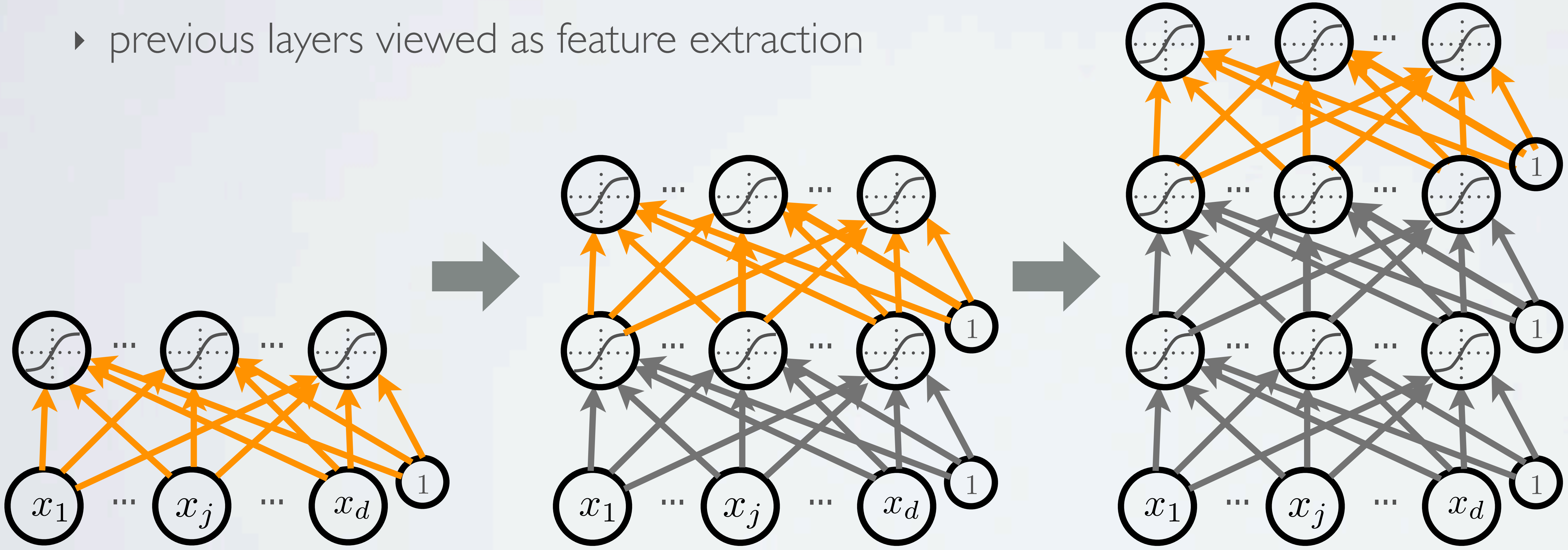
random image

  - hence we expect less overfitting

UNSUPERVISED PRE-TRAINING

Topics: unsupervised pre-training

- We will use a greedy, layer-wise procedure
  - train one layer at a time, from first to last, with unsupervised criterion
  - fix the parameters of previous hidden layers
  - previous layers viewed as feature extraction



UNSUPERVISED PRE-TRAINING

Topics: unsupervised pre-training

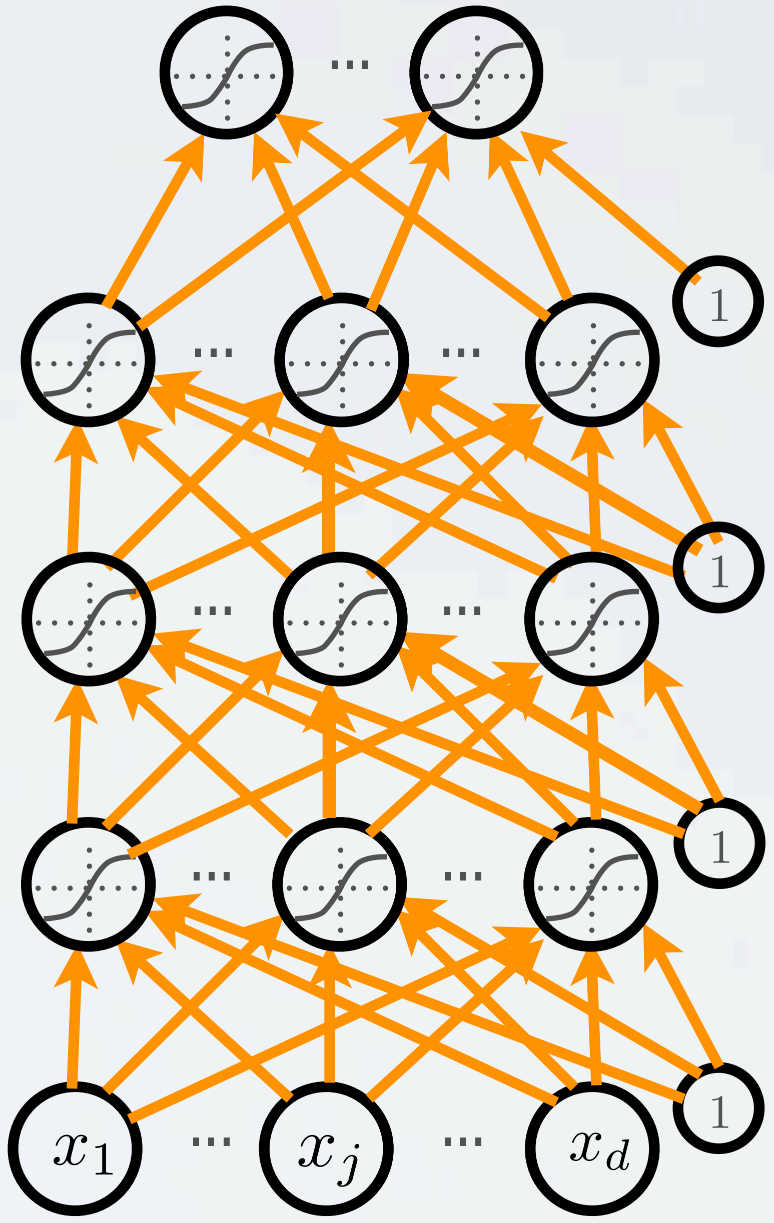
- We call this procedure unsupervised pre-training
  - **first layer:** find hidden unit features that are more common in training inputs than in random inputs
  - **second layer:** find *combinations* of hidden unit features that are more common than random hidden unit features
  - **third layer:** find *combinations of combinations* of ...
  - etc.

- Pre-training initializes the parameters in a region such that the near local optima overfit less the data

FINE-TUNING

Topics: fine-tuning

- Once all layers are pre-trained
  - add output layer
  - train the whole network using supervised learning
- Supervised learning is performed as in a regular feed-forward network
  - forward propagation, backpropagation and update
- We call this last phase fine-tuning
  - all parameters are "tuned" for the supervised task at hand
  - representation is adjusted to be more discriminative



DEEP LEARNING

Topics: pseudocode

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{\mathbf{h}^{(l-1)}(\mathbf{x}^{(t)})\right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$
- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

DEEP LEARNING

Topics: pseudocode

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{ \mathbf{h}^{(l-1)}(\mathbf{x}^{(t)}) \right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$

pre-training

- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

DEEP LEARNING

Topics: pseudocode

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{\mathbf{h}^{(l-1)}(\mathbf{x}^{(t)})\right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$

pre-training

- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

fine-tuning

WHAT KIND OF UNSUPERVISED LEARNING ?

Topics: stacked RBMs, stacked autoencoders

- Stacked restricted Boltzmann machines:
  - Hinton, Teh and Osindero suggested this procedure with RBMs
    - A fast learning algorithm for deep belief nets.
      Hinton, Teh, Osindero., 2006.
    - To recognize shapes, first learn to generate images.
      Hinton, 2006.
- Stacked autoencoders:
  - Bengio, Lamblin, Popovici and Larochelle studied and generalized the procedure to autoencoders
    - Greedy Layer-Wise Training of Deep Networks.
      Bengio, Lamblin, Popovici and Larochelle, 2007.
  - Ranzato, Poultney, Chopra and LeCun also generalized it to sparse autoencoders
    - Efficient Learning of Sparse Representations with an Energy-Based Model.
      Ranzato, Poultney, Chopra and LeCun, 2007.

WHAT KIND OF UNSUPERVISED LEARNING ?

Topics: stacked RBMs, stacked autoencoders

- Stacked denoising autoencoders:
  - proposed by Vincent, Larochelle, Bengio and Manzagol
    - Extracting and Composing Robust Features with Denoising Autoencoders,
      Vincent, Larochelle, Bengio and Manzagol, 2008.
- And more:
  - stacked semi-supervised embeddings
    - Deep Learning via Semi-Supervised Embedding. Weston, Ratle and Collobert, 2008.
  - stacked kernel PCA
    - Kernel Methods for Deep Learning.
      Cho and Saul, 2009.
  - stacked independent subspace analysis
    - Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis.
      Le, Zou, Yeung and Ng, 2011.

Neural networks

Deep learning - example

DEEP LEARNING

Topics: pseudocode

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{ \mathbf{h}^{(l-1)}(\mathbf{x}^{(t)}) \right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$
- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

DEEP LEARNING

Topics: pseudocode

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{\mathbf{h}^{(l-1)}(\mathbf{x}^{(t)})\right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$

pre-training

- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

DEEP LEARNING

Topics: pseudocode

- for l=1 to L
  - build unsupervised training set (with $\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$):

$$\mathcal{D} = \left\{\mathbf{h}^{(l-1)}(\mathbf{x}^{(t)})\right\}_{t=1}^{T}$$

  - train "greedy module" (RBM, autoencoder) on $\mathcal{D}$
  - use hidden layer weights and biases of greedy module to initialize the deep network parameters $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$

pre-training

- Initialize $\mathbf{W}^{(L+1)}$, $\mathbf{b}^{(L+1)}$ randomly (as usual)
- Train the whole neural network using (supervised) stochastic gradient descent (with backprop)

fine-tuning

DEEP LEARNING

Topics: datasets

- Datasets generated with varying number of factors of variations

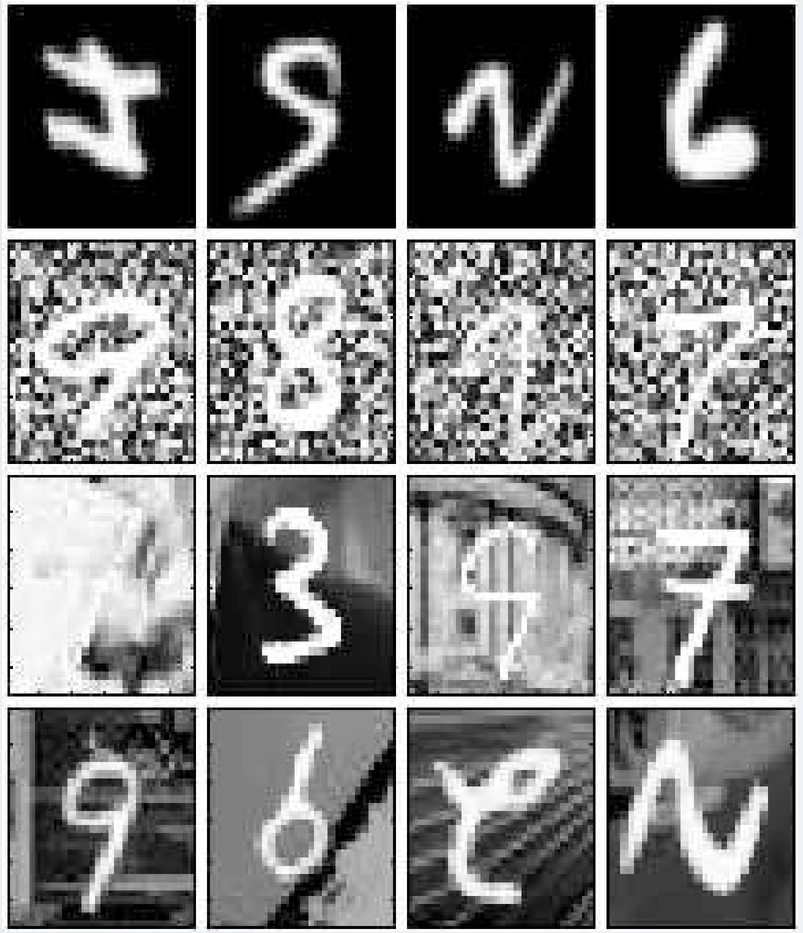
Variations on MNIST

MNIST-rotation

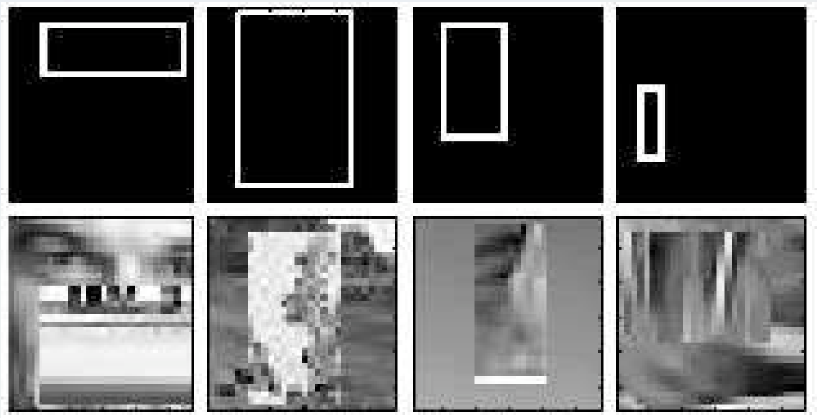
MNIST-random-background

MNIST-image-background

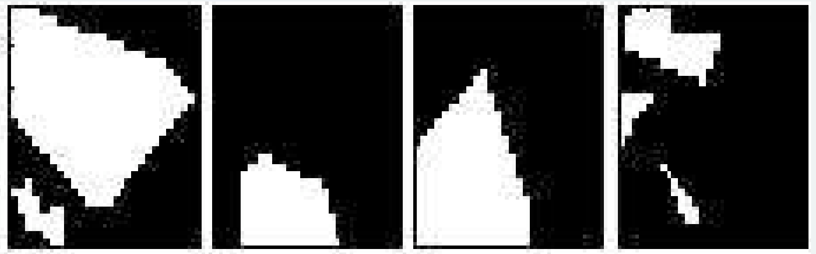
MNIST-background-rotation



Tall or wide?



Convex shape or not?



An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation
Larochelle, Erhan, Courville, Bergstra and Bengio, 2007

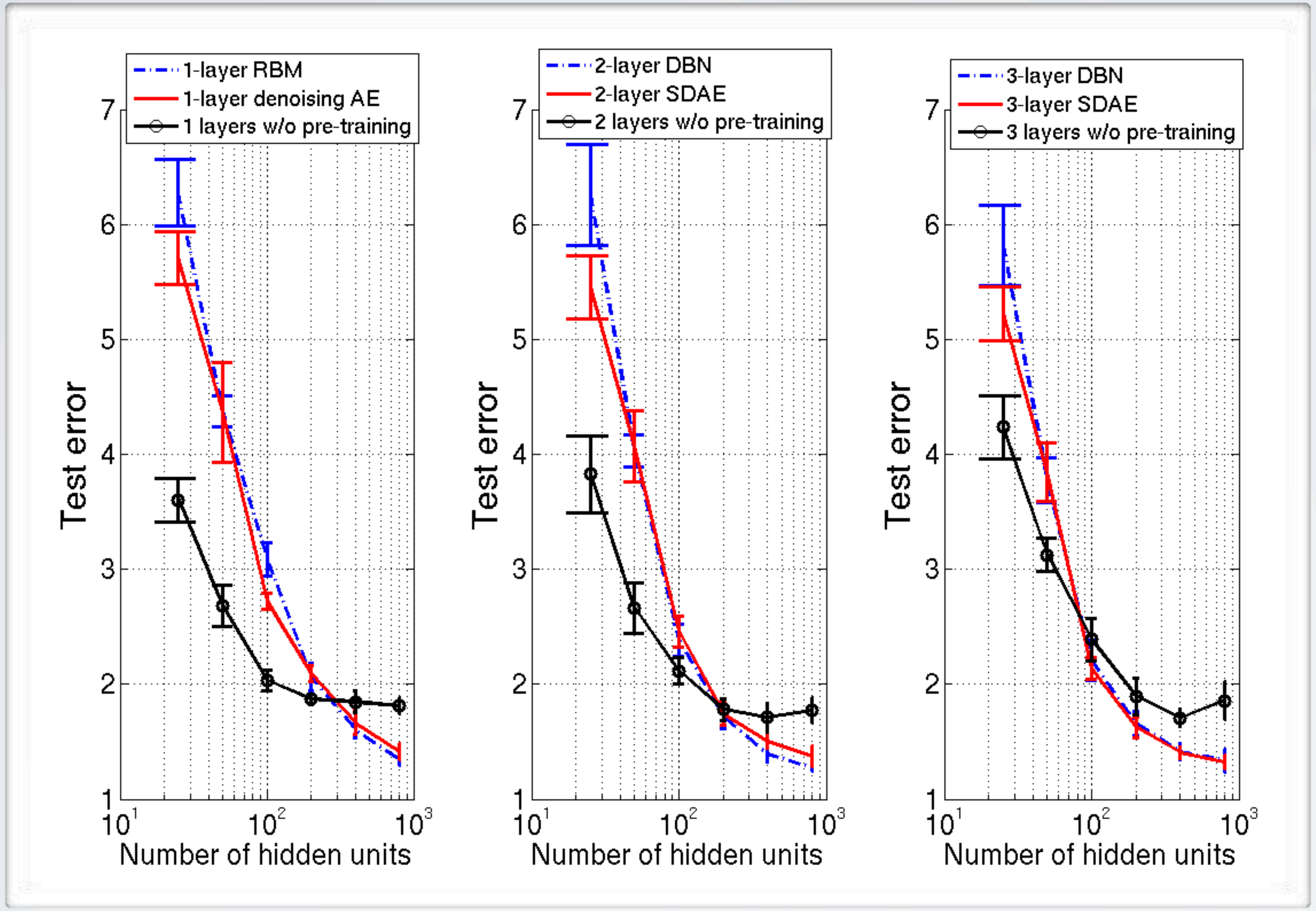
DEEP LEARNING

Topics: impact of initialization

Network		MNIST-small classif. test error	MNIST-rotation classif. test error
Type	Depth		
Deep net	1	**4.14** % $\pm$ 0.17	15.22 % $\pm$ 0.31
	2	**4.03** % $\pm$ 0.17	**10.63** % $\pm$ 0.27
	3	**4.24** % $\pm$ 0.18	11.98 % $\pm$ 0.28
	4	4.47 % $\pm$ 0.18	11.73 % $\pm$ 0.29
Deep net + autoencoder	1	3.87 % $\pm$ 0.17	11.43% $\pm$ 0.28
	2	**3.38** % $\pm$ 0.16	9.88 % $\pm$ 0.26
	3	**3.37** % $\pm$ 0.16	**9.22** % $\pm$ 0.25
	4	**3.39** % $\pm$ 0.16	**9.20** % $\pm$ 0.25
Deep net + RBM	1	3.17 % $\pm$ 0.15	10.47 % $\pm$ 0.27
	2	**2.74** % $\pm$ 0.14	9.54 % $\pm$ 0.26
	3	**2.71** % $\pm$ 0.14	**8.80** % $\pm$ 0.25
	4	**2.72** % $\pm$ 0.14	**8.83** % $\pm$ 0.24

DEEP LEARNING

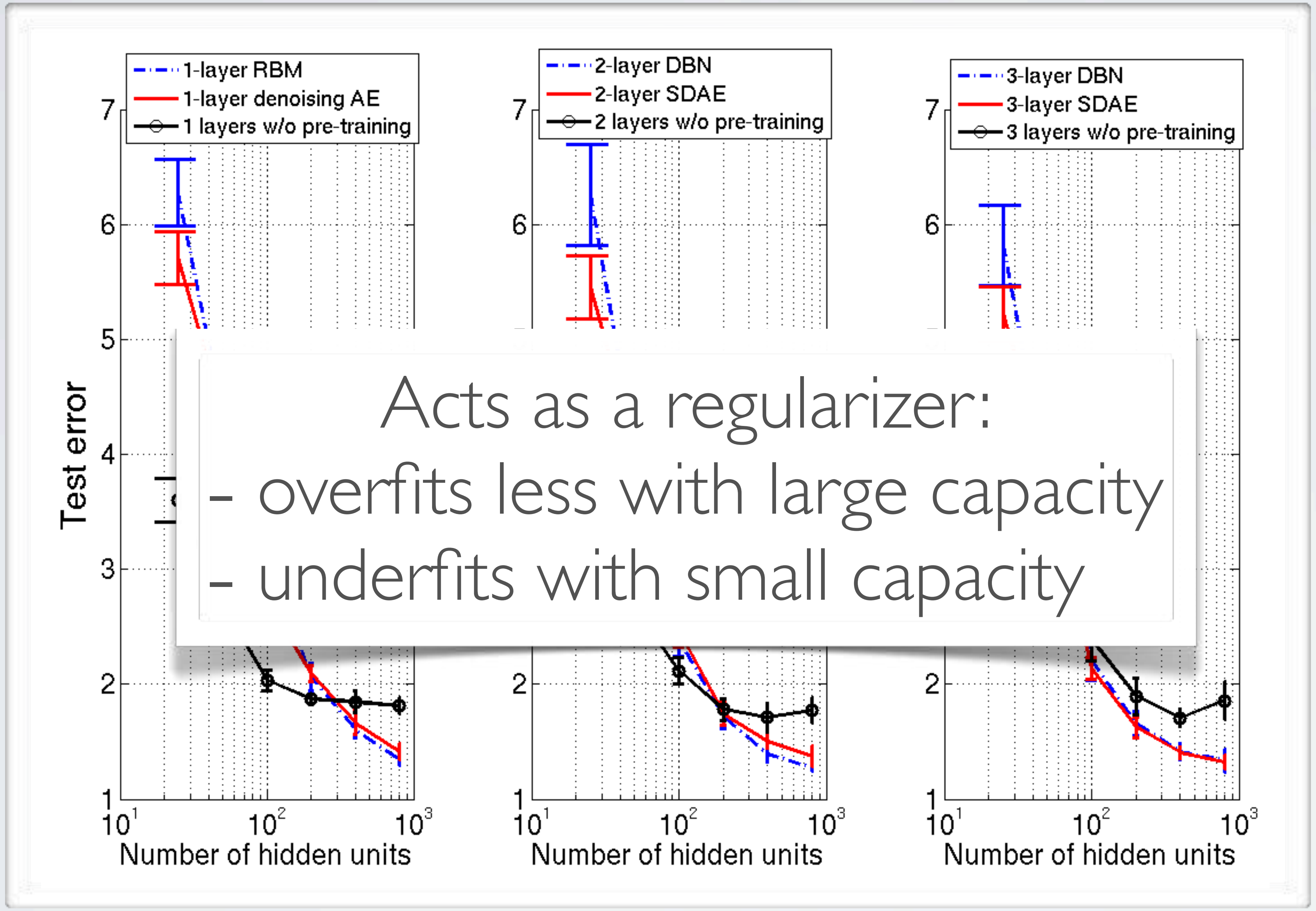
Topics: impact of initialization



Why Does Unsupervised Pre-training Help Deep Learning?
Erhan, Bengio, Courville, Manzagol, Vincent and Bengio, 2011

DEEP LEARNING

Topics: impact of initialization



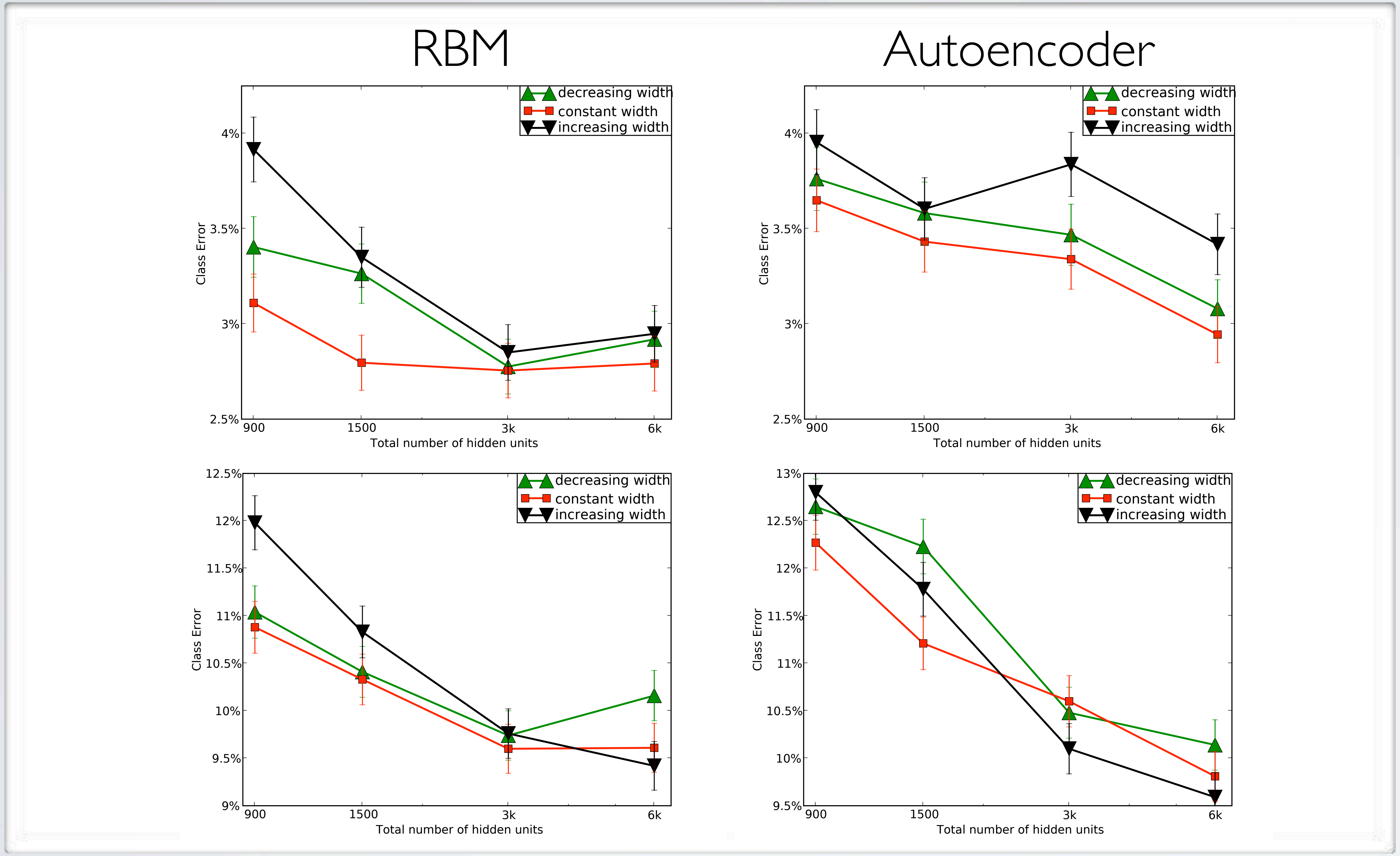
Acts as a regularizer:

- overfits less with large capacity
- underfits with small capacity

Why Does Unsupervised Pre-training Help Deep Learning?
Erhan, Bengio, Courville, Manzagol, Vincent and Bengio, 2011

DEEP LEARNING

Topics: choice of hidden layer size



DEEP LEARNING

Topics: performance on different datasets

Stacked Autoencoders | Stacked RBMS | Stacked Denoising Autoencoders

Dataset	**SVM**$_{rbf}$	**SAA-3**	**DBN-3**	**SdA-3** (ν)
basic	**3.03±0.15**	3.46±0.16	3.11±0.15	**2.80±0.14** (10%)
rot	11.11±0.28	**10.30±0.27**	**10.30±0.27**	**10.29±0.27** (10%)
bg-rand	14.58±0.31	11.28±0.28	**6.73±0.22**	10.38±0.27 (40%)
bg-img	22.61±0.37	23.00±0.37	**16.31±0.32**	**16.68±0.33** (25%)
rot-bg-img	55.18±0.44	51.93±0.44	47.39±0.44	**44.49±0.44** (25%)
rect	**2.15±0.13**	2.41±0.13	2.60±0.14	**1.99±0.12** (10%)
rect-img	24.04±0.37	24.05±0.37	22.50±0.37	**21.59±0.36** (25%)
convex	19.13±0.34	**18.41±0.34**	**18.63±0.34**	**19.06±0.34** (10%)

Extracting and Composing Robust Features with Denoising Autoencoders,
Vincent, Larochelle, Bengio and Manzagol, 2008.

Neural networks

Deep learning - dropout

DEEP LEARNING

Topics: why training is hard

- Depending on the problem, one or the other situation will tend to prevail

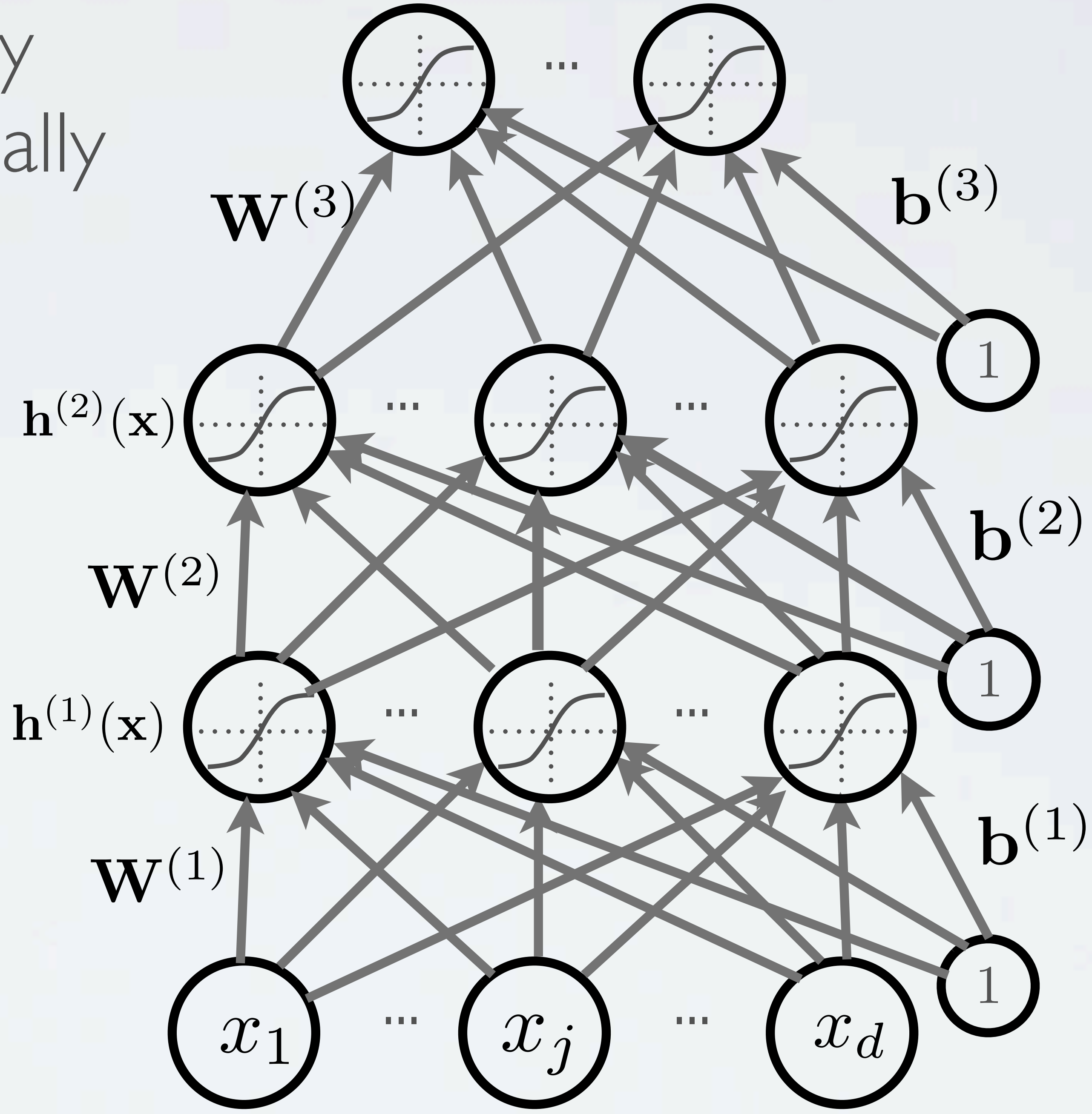
- If first hypothesis (underfitting): use better optimization
  - this is an active area of research

- If second hypothesis (overfitting): use better regularization
  - unsupervised learning
  - stochastic «dropout» training

DROPOUT

Topics: dropout

- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful
- Could use a different dropout probability, but 0.5 usually works well

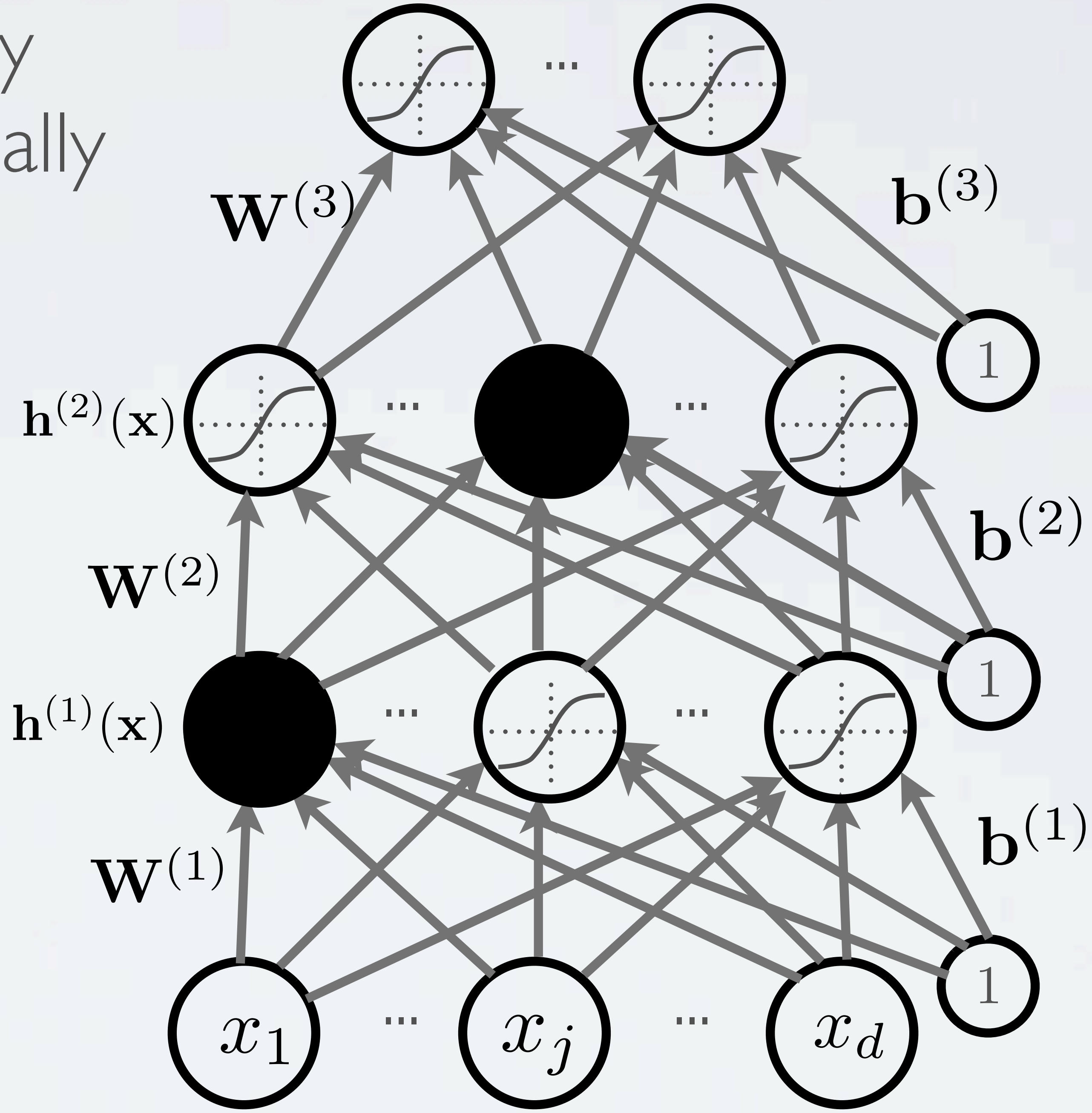


DROPOUT

Topics: dropout

- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful

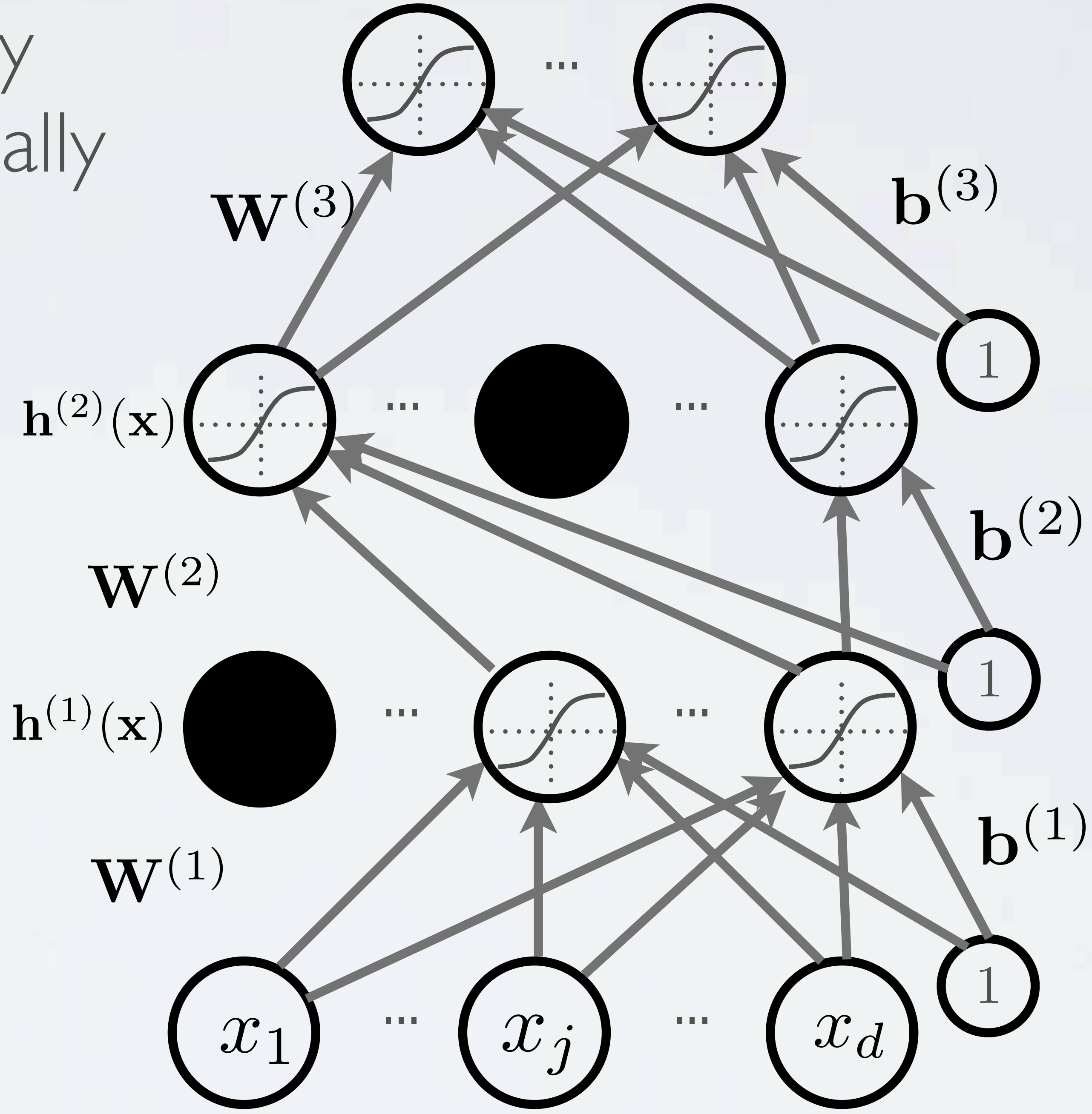
- Could use a different dropout probability, but 0.5 usually works well



DROPOUT

Topics: dropout

- Idea: «cripple» neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful
- Could use a different dropout probability, but 0.5 usually works well



DROPOUT

Topics: dropout

- Use random binary masks $\mathbf{m}^{(k)}$
  - layer pre-activation for $k{>}0$ $(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

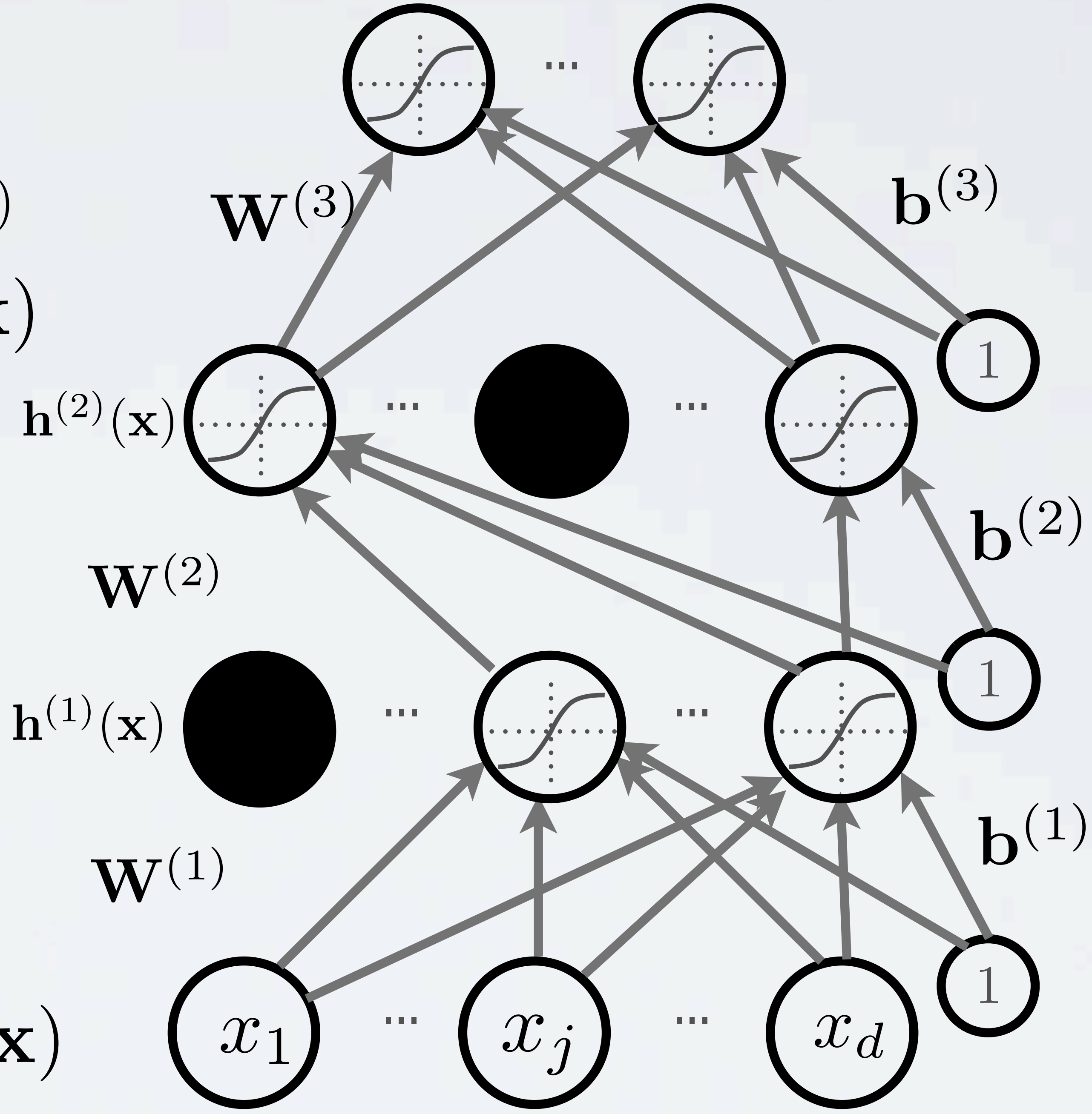
    $$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

    $$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation ($k{=}L{+}1$):

    $$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



DROPOUT

Topics: dropout

- Use random binary masks $\mathbf{m}^{(k)}$
  - layer pre-activation for $k>0$ $\quad (\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

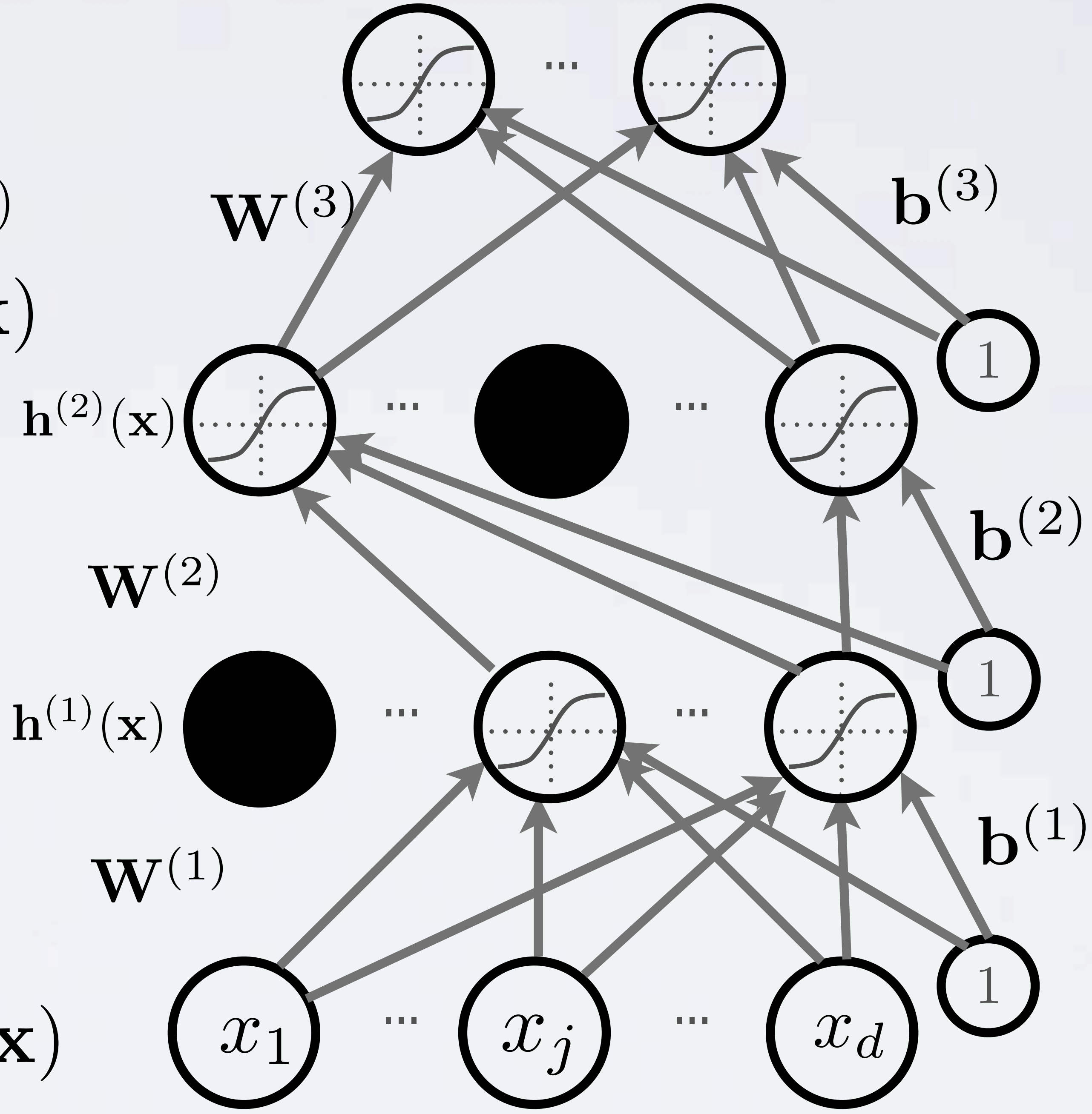
    $$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k from 1 to L):

    $$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x})) \odot \mathbf{m}^{(k)}$$

  - output layer activation (k=L+1):

    $$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



DROPOUT

Topics: dropout backpropagation

- This assumes a forward propagation has been made before
  - compute output gradient (before activation)

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

  - for k from $L+1$ to 1
    - compute gradients of hidden layer parameter

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \iff \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \; \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \iff \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

    - compute gradient of hidden layer below

$$\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff \mathbf{W}^{(k)}{}^\top \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right)$$

    - compute gradient of hidden layer below (before activation)

$$\nabla_{\mathbf{a}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff \left(\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \odot [\ldots, g'(a^{(k-1)}(\mathbf{x})_j), \ldots]$$

DROPOUT

Topics: dropout backpropagation

- This assumes a forward propagation has been made before
  - compute output gradient (before activation)

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff -\left(\mathbf{e}(y) - \mathbf{f}(\mathbf{x})\right)$$

  - for k from $L{+}1$ to 1

includes the mask $\mathbf{m}^{(k-1)}$

    - compute gradients of hidden layer parameter

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \iff \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \ \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \iff \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

    - compute gradient of hidden layer below

$$\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff \mathbf{W}^{(k)^\top} \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right)$$

    - compute gradient of hidden layer below (before activation)

$$\nabla_{\mathbf{a}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff \left(\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \odot \left[\ldots, g'(a^{(k-1)}(\mathbf{x})_j), \ldots\right] \odot \mathbf{m}^{(k-1)}$$

DROPOUT

Topics: test time classification

- At test time, we replace the masks by their expectation
  - this is simply the constant vector 0.5 if dropout probability is 0.5
  - for single hidden layer, can show this is equivalent to taking the geometric average of all neural networks, with all possible binary masks

- Can be combined with unsupervised pre-training

- Beats regular backpropagation on many datasets
  - Improving neural networks by preventing co-adaptation of feature detectors. Hinton, Srivastava, Krizhevsky, Sutskever and Salakhutdinov, 2012.

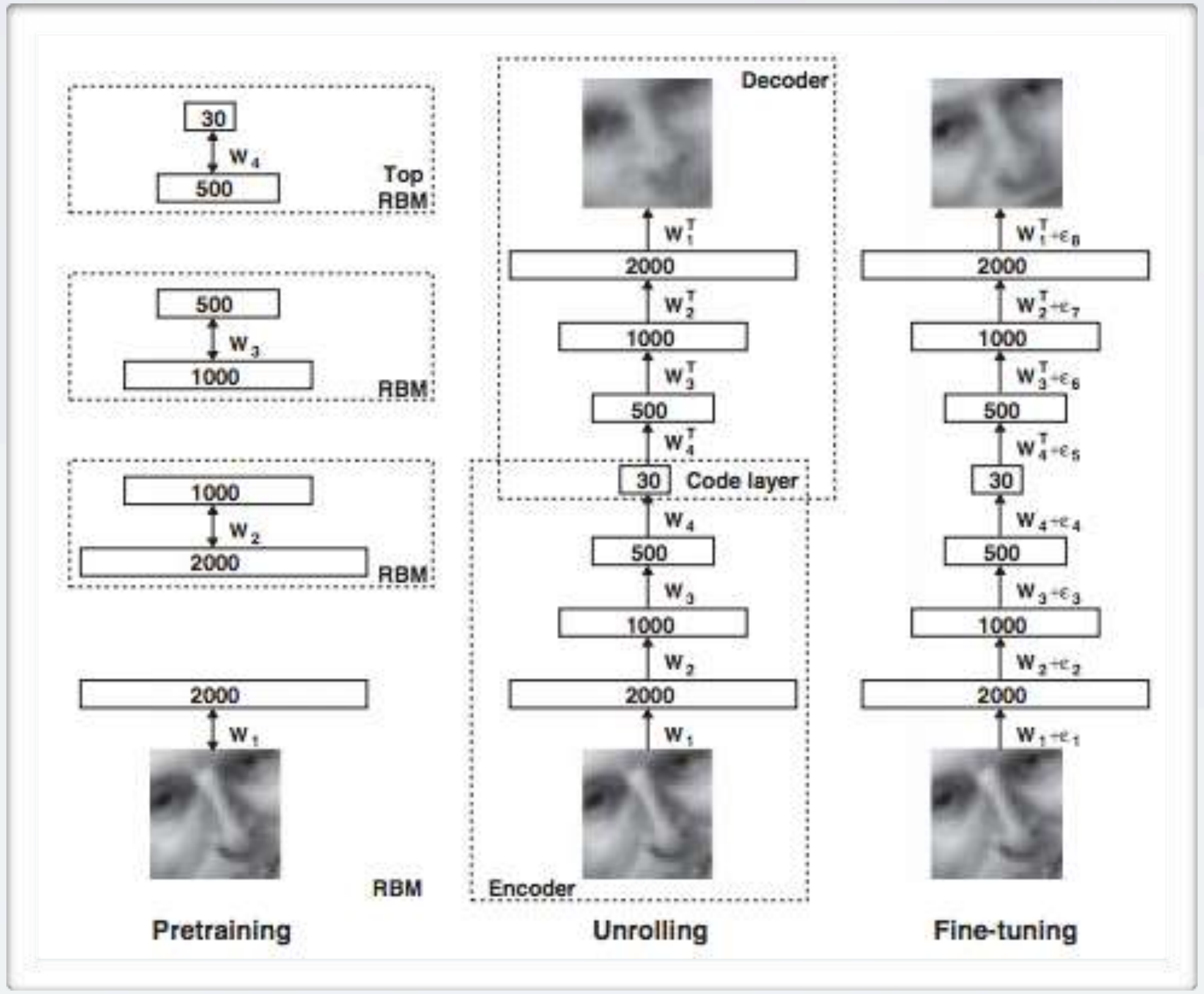
Neural networks

Deep learning - deep autoencoder

DEEP AUTOENCODER

Topics: deep autoencoder

- Pre-training can be used to initialize a deep autoencoder
  - This is an example of a situation where underfitting is an issue
    - perhaps surprisingly, pre-training initializes the optimization problem in a region with better local optima of training objective
  - Each RBM used to initialize parameters both in encoder and decoder ("unrolling")
  - Better optimization algorithms can also help
    - Deep learning via Hessian-free optimization. James Martens, 2010



From Hinton and Salakhutdinov, Science, 2006

DEEP AUTOENCODER

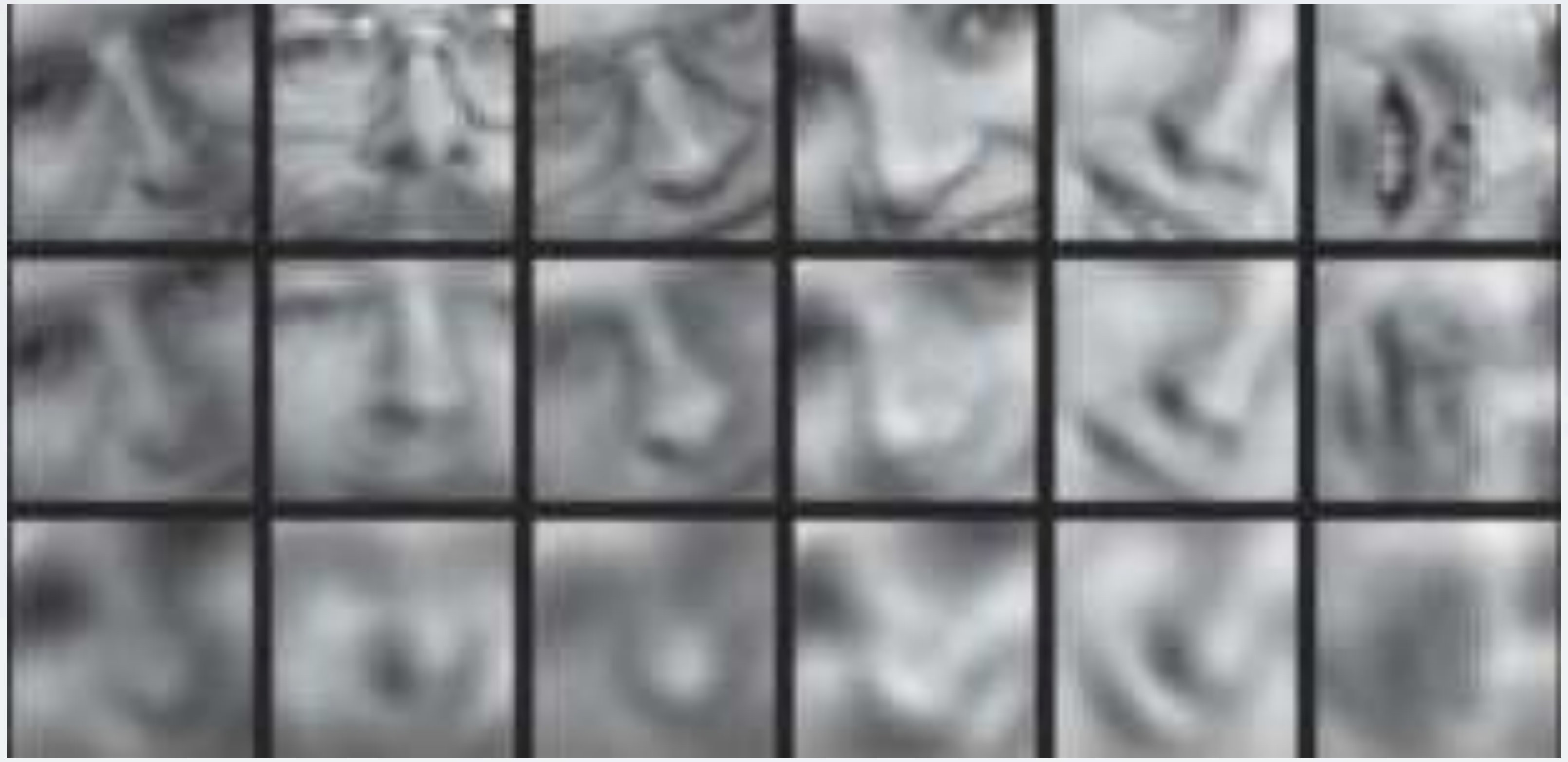
Topics: deep autoencoder

- Can be used to reduce the dimensionality of the data
  - will have better reconstruction than a single layer network (i.e. PCA)

Original data

Deep autoencoder reconstruction

PCA reconstruction

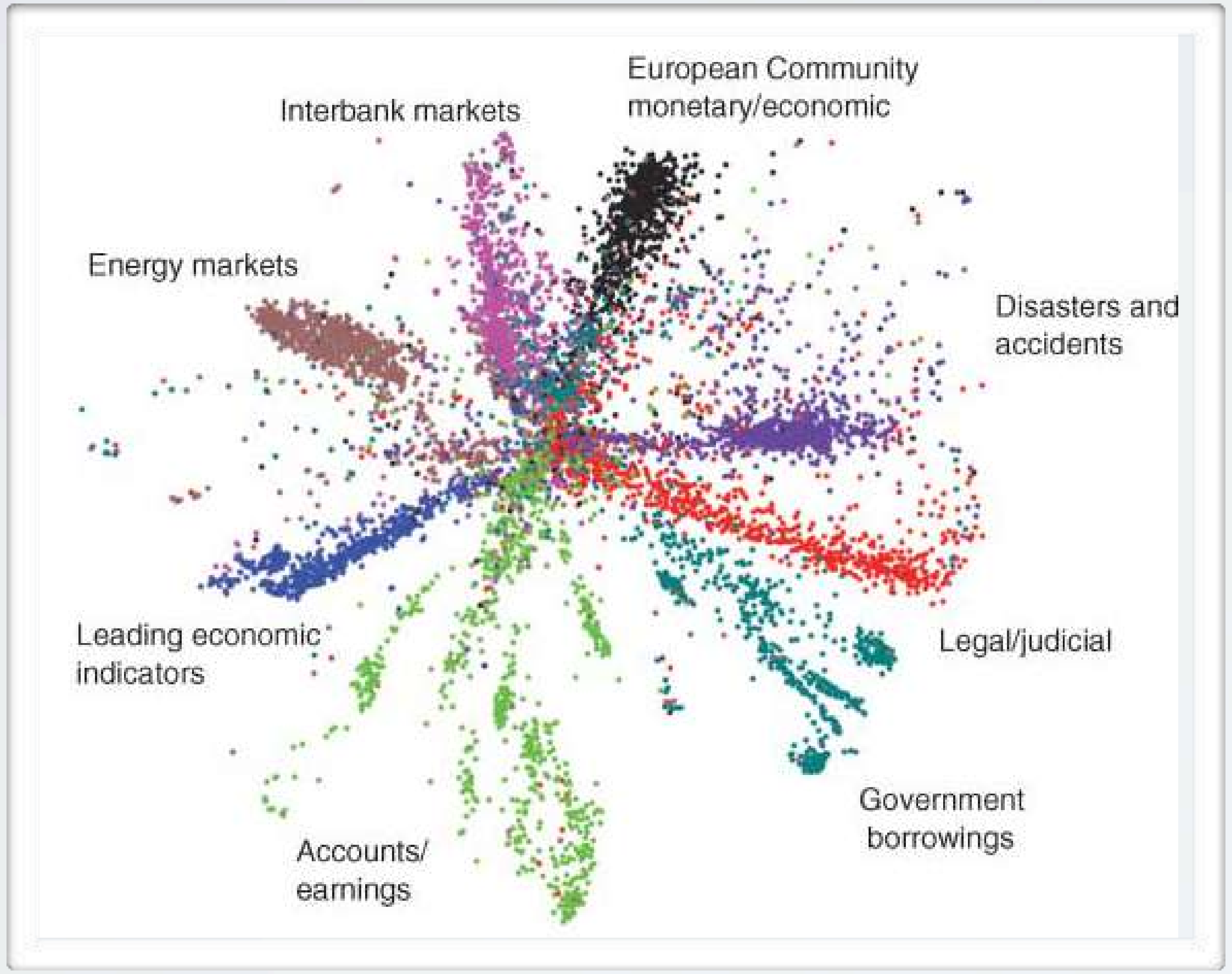


From Hinton and Salakhutdinov, Science, 2006

DEEP AUTOENCODER

Topics: deep autoencoder

- If we reduce to 2D, we can visualize the data (e.g. a collection of document)



From Hinton and Salakhutdinov, Science, 2006

Neural networks

Deep learning - deep belief network

DEEP BELIEF NETWORK

Topics: deep belief network

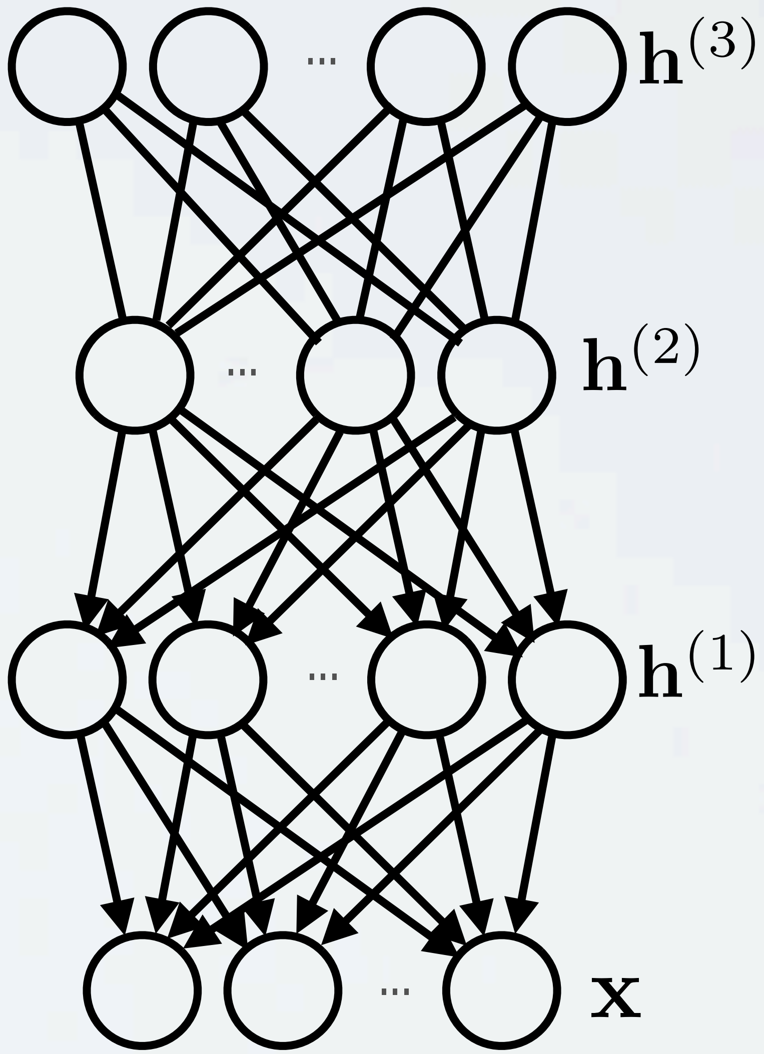
- The idea of pre-training came from work on deep belief networks (DBNs)
  - it is a generative model that mixes undirected and directed connections between variables
  - top 2 layers' distribution $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$ is an RBM
  - other layers form a Bayesian network:
    - the conditional distributions of a layers given the one above it are

$$p(h_j^{(1)} = 1 | \mathbf{h}^{(2)}) = \mathrm{sigm}(\mathbf{b}^{(1)} + \mathbf{W}^{(2)^\top} \mathbf{h}^{(2)})$$

$$p(x_i = 1 | \mathbf{h}^{(1)}) = \mathrm{sigm}(\mathbf{b}^{(0)} + \mathbf{W}^{(1)^\top} \mathbf{h}^{(1)})$$

    - this is referred to as a **sigmoid belief network** (SBN)
  - a DBN **is not** a feed-forward network

DBN's graphical model



DEEP BELIEF NETWORK

Topics: deep belief network

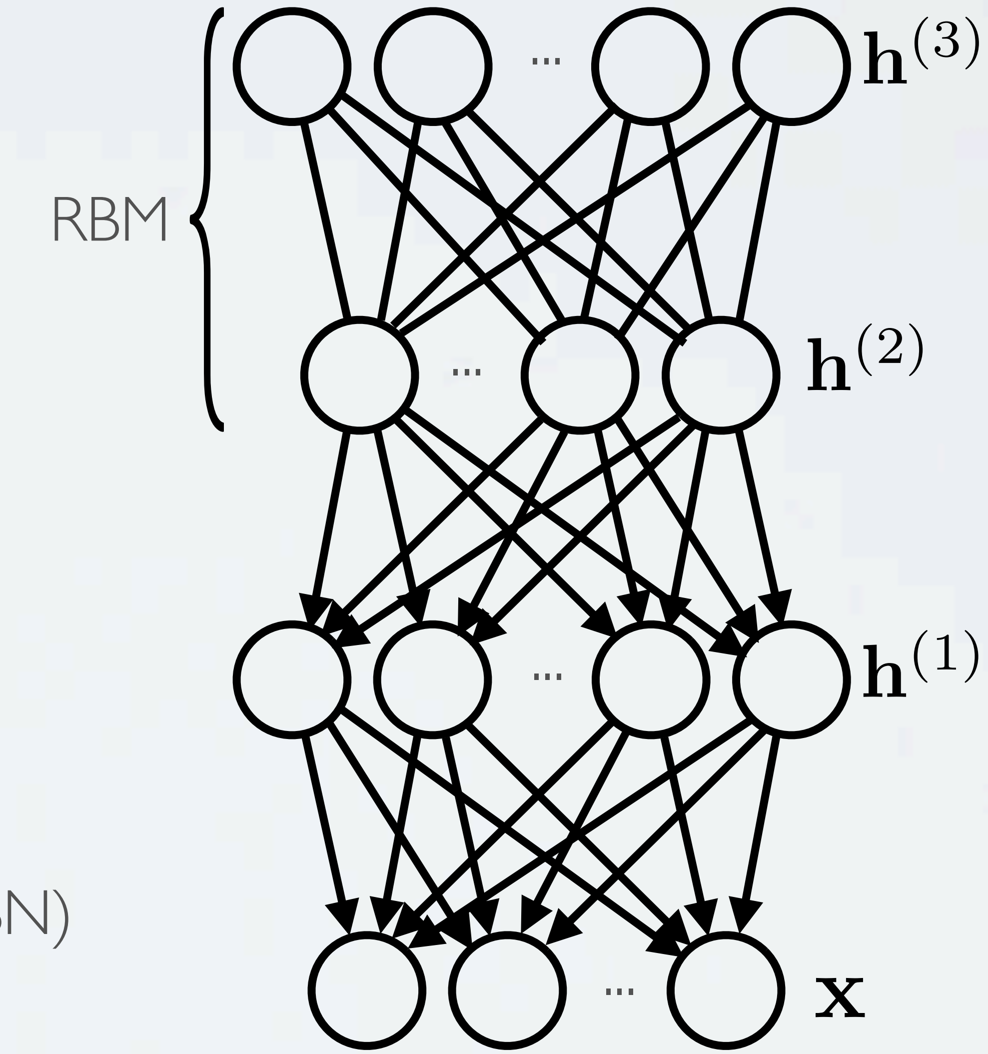
- The idea of pre-training came from work on deep belief networks (DBNs)
  - it is a generative model that mixes undirected and directed connections between variables
  - top 2 layers' distribution $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$ is an RBM
  - other layers form a Bayesian network:
    - the conditional distributions of a layers given the one above it are

$$p(h_j^{(1)} = 1|\mathbf{h}^{(2)}) = \mathrm{sigm}(\mathbf{b}^{(1)} + \mathbf{W}^{(2)^\top}\mathbf{h}^{(2)})$$

$$p(x_i = 1|\mathbf{h}^{(1)}) = \mathrm{sigm}(\mathbf{b}^{(0)} + \mathbf{W}^{(1)^\top}\mathbf{h}^{(1)})$$

    - this is referred to as a **sigmoid belief network** (SBN)
  - a DBN **is not** a feed-forward network

DBN's graphical model



DEEP BELIEF NETWORK

Topics: deep belief network

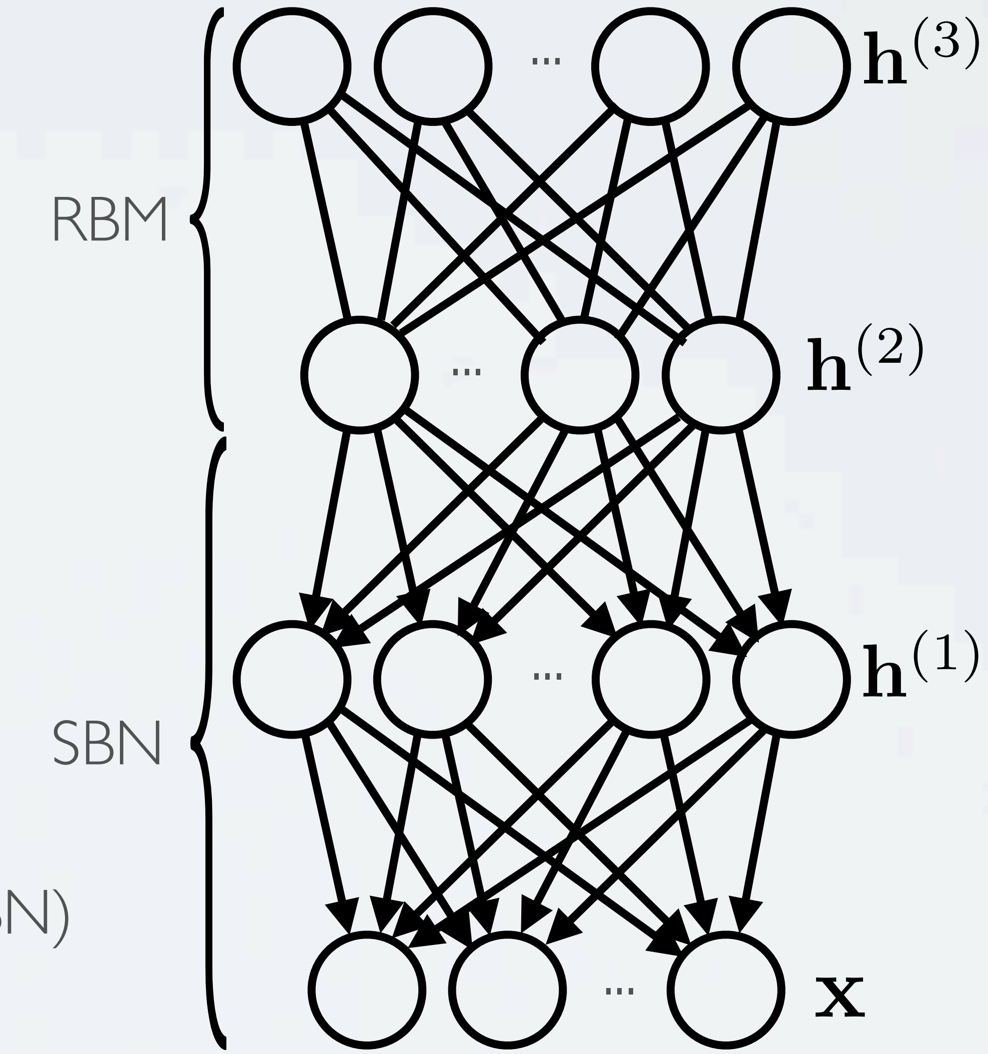
- The idea of pre-training came from work on deep belief networks (DBNs)
  - it is a generative model that mixes undirected and directed connections between variables
  - top 2 layers' distribution $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$ is an RBM
  - other layers form a Bayesian network:
    - the conditional distributions of a layers given the one above it are

$$p(h_j^{(1)} = 1|\mathbf{h}^{(2)}) = \mathrm{sigm}(\mathbf{b}^{(1)} + \mathbf{W}^{(2)^\top}\mathbf{h}^{(2)})$$

$$p(x_i = 1|\mathbf{h}^{(1)}) = \mathrm{sigm}(\mathbf{b}^{(0)} + \mathbf{W}^{(1)^\top}\mathbf{h}^{(1)})$$

    - this is referred to as a **sigmoid belief network** (SBN)
  - a DBN **is not** a feed-forward network

DBN's graphical model



DEEP BELIEF NETWORK

Topics: deep belief network

- The full distribution of a DBN is as follows

$$p(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})\, p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)})\, p(\mathbf{x}|\mathbf{h}^{(1)})$$

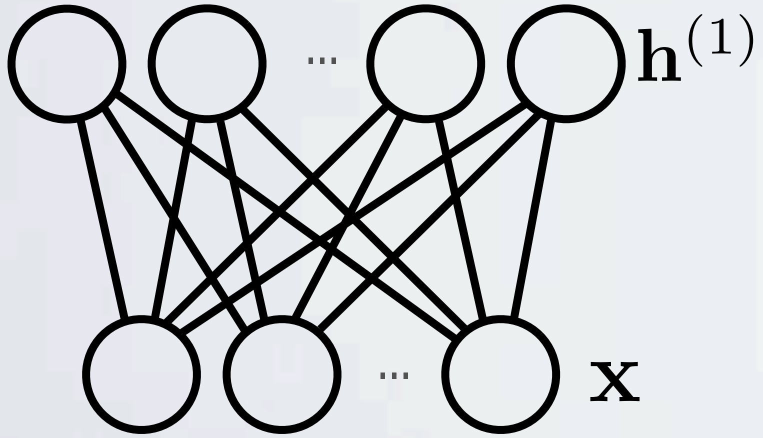
  - where:
    - $p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \exp\left(\mathbf{h}^{(2)^\top}\mathbf{W}^{(3)}\mathbf{h}^{(3)} + \mathbf{b}^{(2)^\top}\mathbf{h}^{(2)} + \mathbf{b}^{(3)^\top}\mathbf{h}^{(3)}\right)/Z$
    - $p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) = \prod_j p(h_j^{(1)}|\mathbf{h}^{(2)})$
    - $p(\mathbf{x}|\mathbf{h}^{(1)}) = \prod_i p(x_i|\mathbf{h}^{(1)})$
- To observe a DBN trained on MNIST in action:
  - http://www.cs.toronto.edu/~hinton/adi/index.htm
- As in a deep feed-forward network, training a DBN is hard
  - initialization will play a crucial role on the results

DEEP BELIEF NETWORK

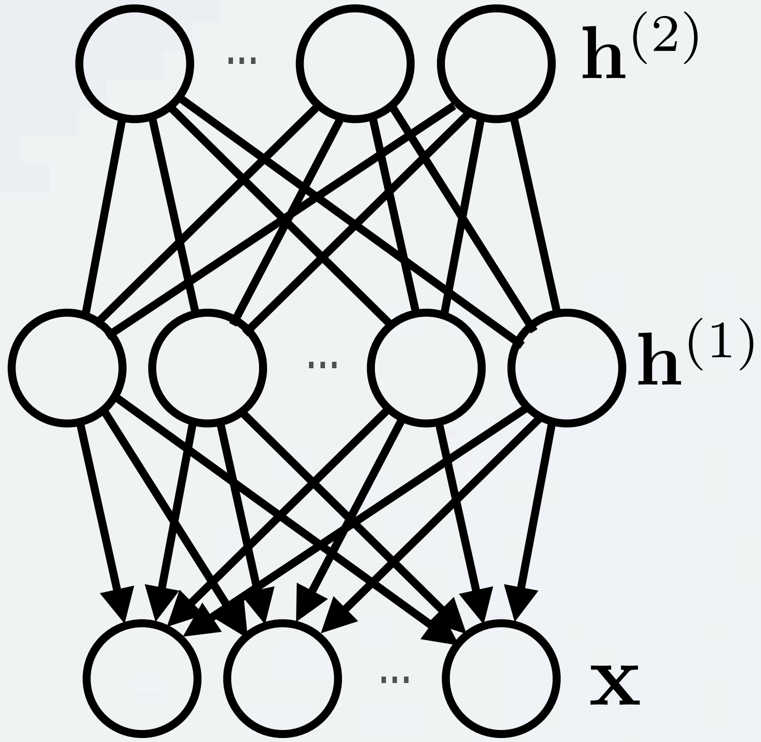
Topics: deep belief network

- This is where the RBM stacking procedure comes from
  - idea: improve prior on last layer by adding another hidden layer
  - how do we train these additional layers?

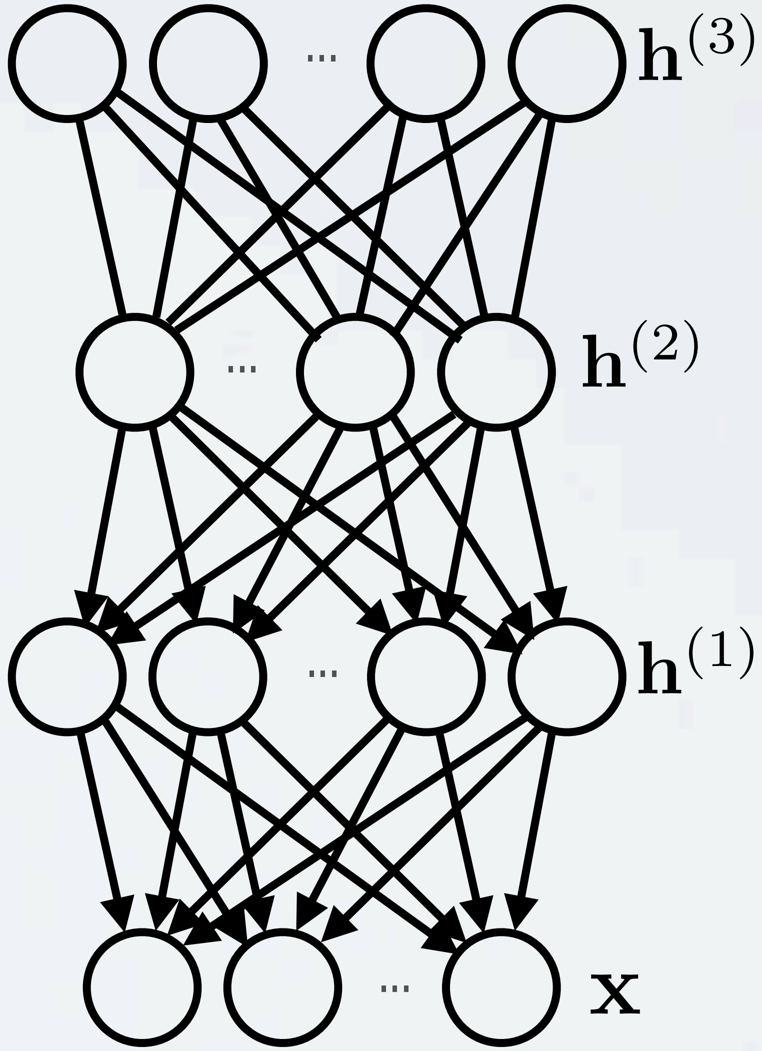
$$p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$$



$$p(\mathbf{x}, \mathbf{h}^{(1)}) = p(\mathbf{x}|\mathbf{h}^{(1)}) \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$$



$$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \sum_{\mathbf{h}^{(3)}} p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$$



DEEP BELIEF NETWORK

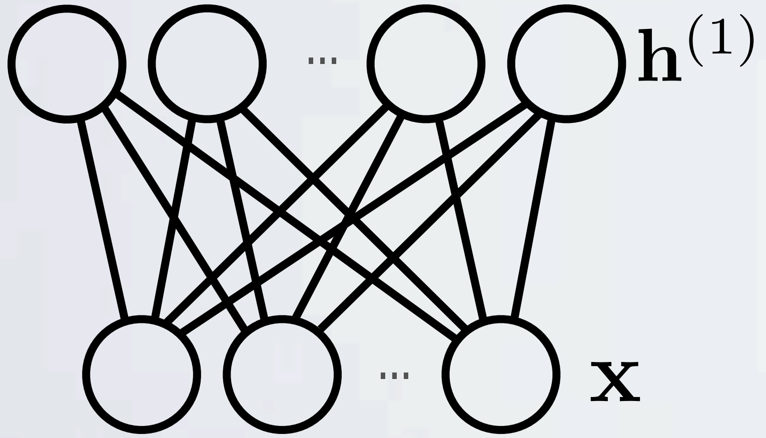
Topics: deep belief network

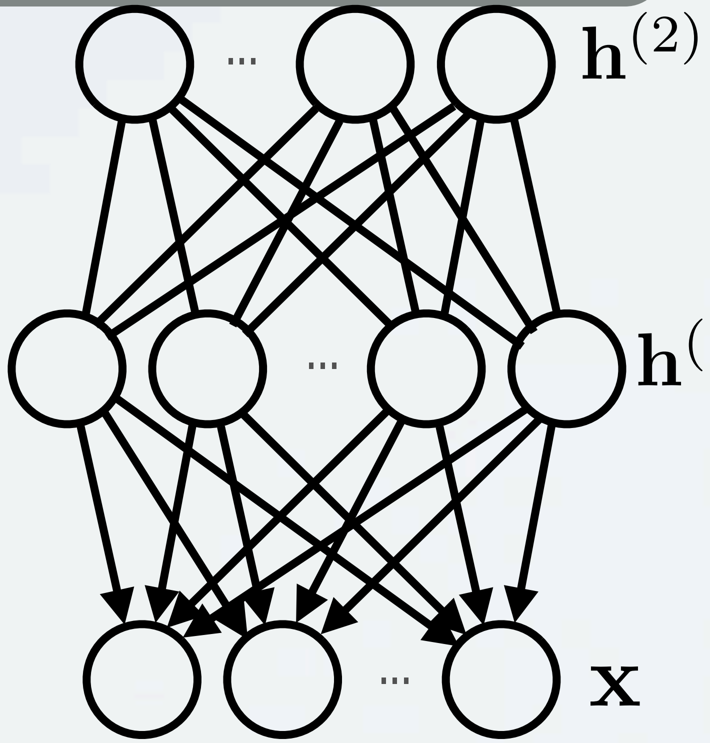
- This is where the RBM stacking procedure comes from
  - idea: improve prior on last layer by adding another hidden layer
  - how do we train these additional layers?

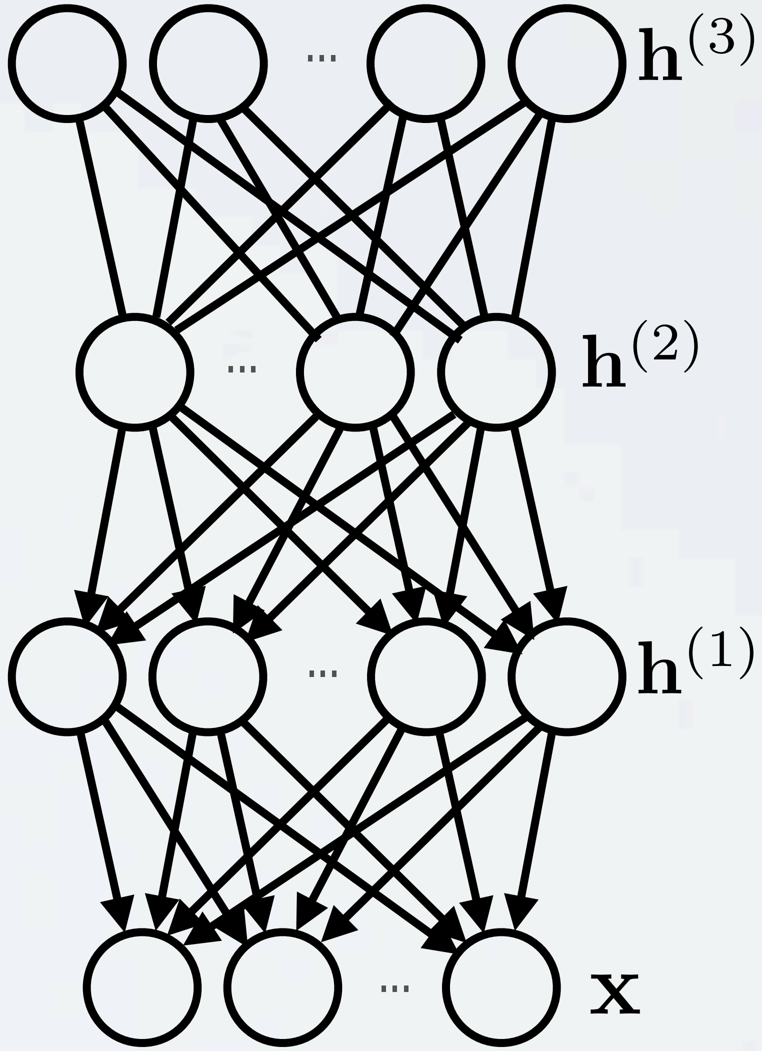
$p(\mathbf{x}, \mathbf{h}^{(1)}) = p(\mathbf{x}|\mathbf{h}^{(1)}) \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$

$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \sum_{\mathbf{h}^{(3)}} p(\mathbf{h}^{(2)}, \mathbf{h}^{(3)})$

$p(\mathbf{x}) = \sum_{\mathbf{h}^{(1)}} p(\mathbf{x}, \mathbf{h}^{(1)})$



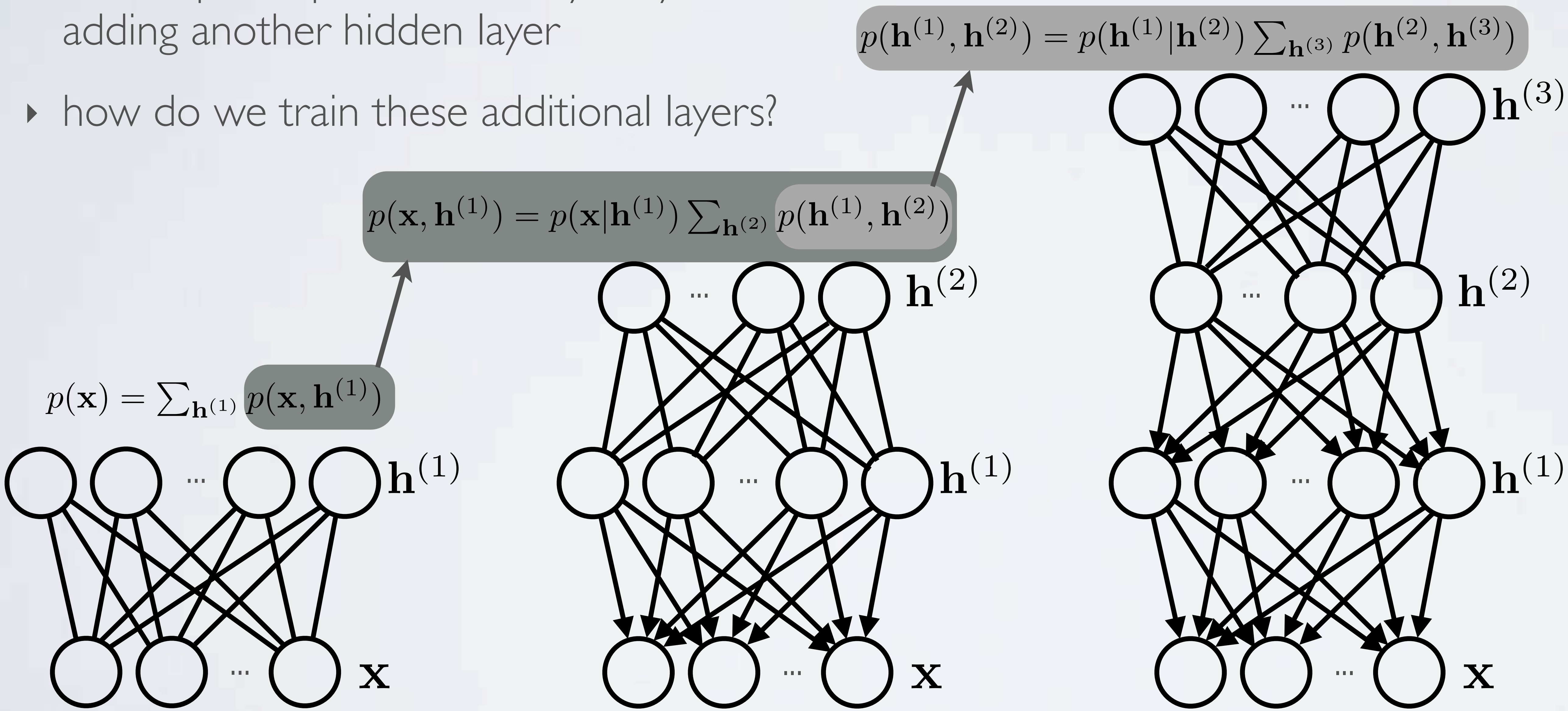




DEEP BELIEF NETWORK

Topics: deep belief network

- This is where the RBM stacking procedure comes from
    - idea: improve prior on last layer by adding another hidden layer
    - how do we train these additional layers?



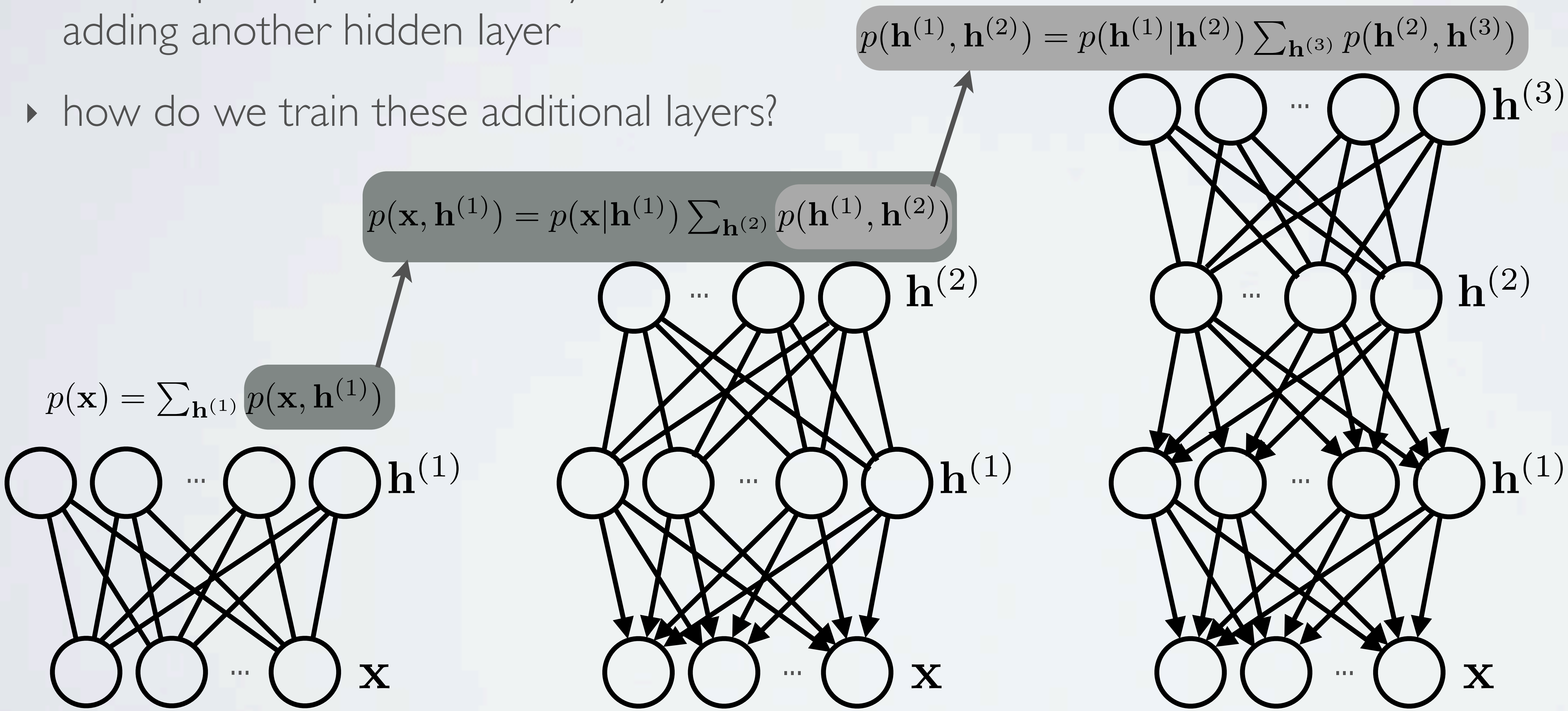
Neural networks

Deep learning - variational bound

DEEP BELIEF NETWORK

Topics: deep belief network

- This is where the RBM stacking procedure comes from
  - idea: improve prior on last layer by adding another hidden layer
  - how do we train these additional layers?

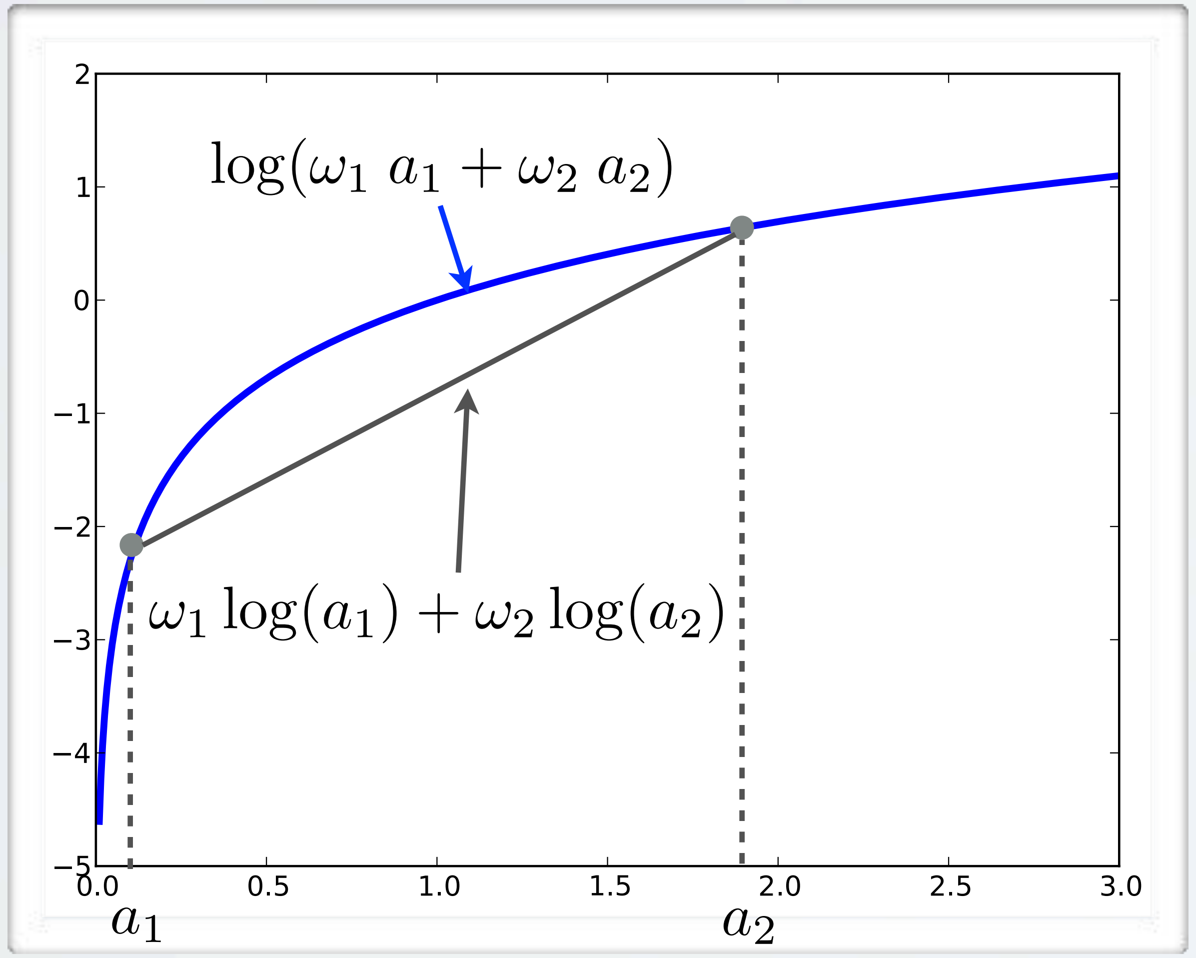


DEEP BELIEF NETWORK

Topics: concavity

- We will use the fact that the logarithm function is concave:

$$\log(\textstyle\sum_i \omega_i\; a_i) \geq \sum_i \omega_i \log(a_i) \quad \text{(where } \sum_i \omega_i = 1 \text{ and } \omega_i \geq 0\text{)}$$



DEEP BELIEF NETWORK

Topics: variational bound

- For any model $p(\mathbf{x}, \mathbf{h}^{(1)})$ with latent variables $\mathbf{h}^{(1)}$ we can write:

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \left(\sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left(\frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&= \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) \\
&\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})
\end{aligned}
$$

- $q(\mathbf{h}^{(1)}|\mathbf{x})$ is any approximation of $p(\mathbf{h}^{(1)}|\mathbf{x})$

DEEP BELIEF NETWORK

Topics: variational bound

- For any model $p(\mathbf{x}, \mathbf{h}^{(1)})$ with latent variables $\mathbf{h}^{(1)}$ we can write:

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \left(\sum_{\mathbf{h}^{(1)}} \overbrace{q(\mathbf{h}^{(1)}|\mathbf{x})}^{\omega_i} \frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left(\frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&= \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) \\
&\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})
\end{aligned}
$$

- $q(\mathbf{h}^{(1)}|\mathbf{x})$ is any approximation of $p(\mathbf{h}^{(1)}|\mathbf{x})$

DEEP BELIEF NETWORK

Topics: variational bound

- For any model $p(\mathbf{x}, \mathbf{h}^{(1)})$ with latent variables $\mathbf{h}^{(1)}$ we can write:

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \left(\sum_{\mathbf{h}^{(1)}} \overbrace{q(\mathbf{h}^{(1)}|\mathbf{x})}^{\omega_i} \overbrace{\frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})}}^{a_i} \right) \\
&\geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left(\frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \\
&= \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) \\
&\quad - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})
\end{aligned}$$

- $q(\mathbf{h}^{(1)}|\mathbf{x})$ is any approximation of $p(\mathbf{h}^{(1)}|\mathbf{x})$

DEEP BELIEF NETWORK

Topics: variational bound

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

  - if $q(\mathbf{h}^{(1)}|\mathbf{x})$ is equal to the true conditional $p(\mathbf{h}^{(1)}|\mathbf{x})$, then we have an equality
  - the more $q(\mathbf{h}^{(1)}|\mathbf{x})$ is different from $p(\mathbf{h}^{(1)}|\mathbf{x})$ the less tight the bound is
  - in fact, the difference between the left and right terms is the KL divergence between $q(\mathbf{h}^{(1)}|\mathbf{x})$ and $p(\mathbf{h}^{(1)}|\mathbf{x})$:

$$\mathrm{KL}(q||p) = \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left(\frac{q(\mathbf{h}^{(1)}|\mathbf{x})}{p(\mathbf{h}^{(1)}|\mathbf{x})} \right)$$

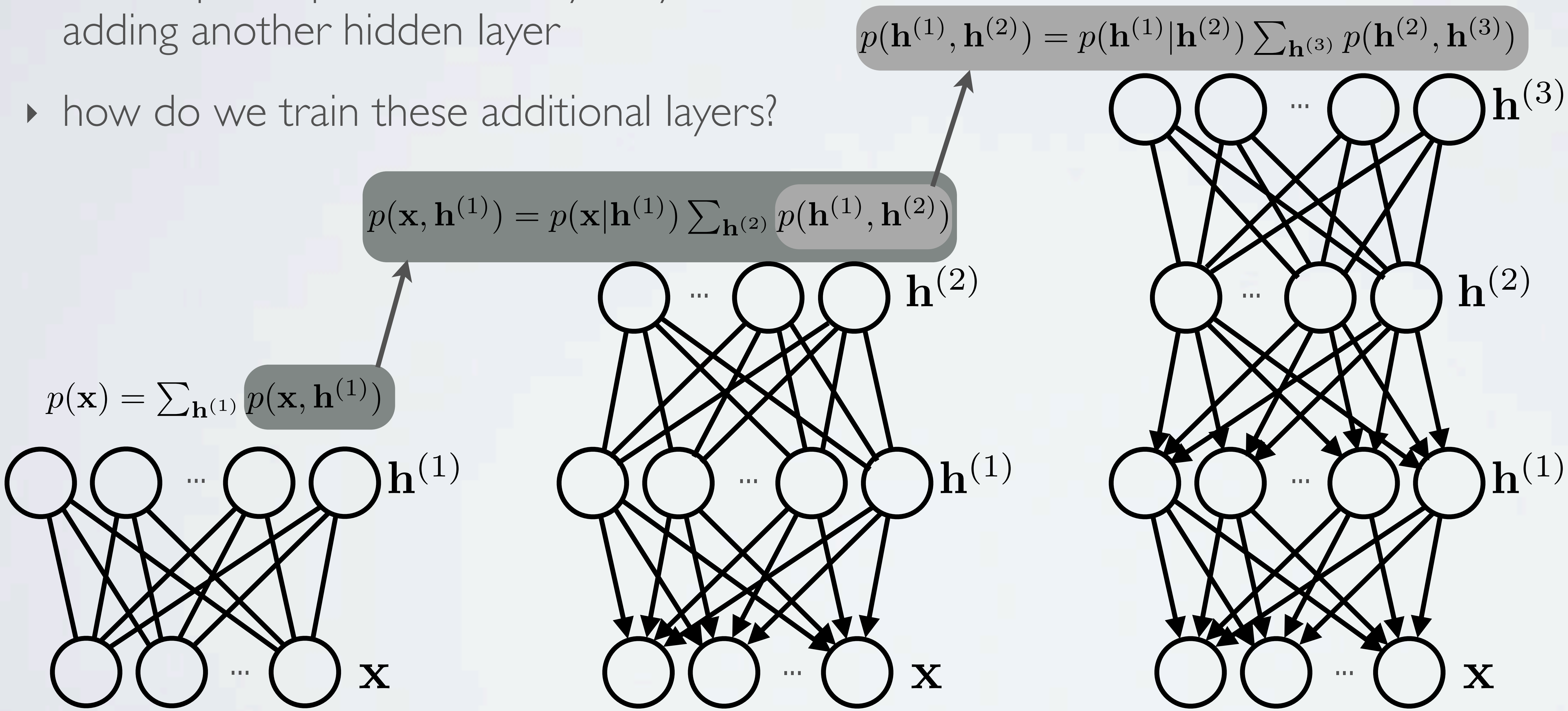
Neural networks

Deep learning - DBN pretraining

DEEP BELIEF NETWORK

Topics: deep belief network

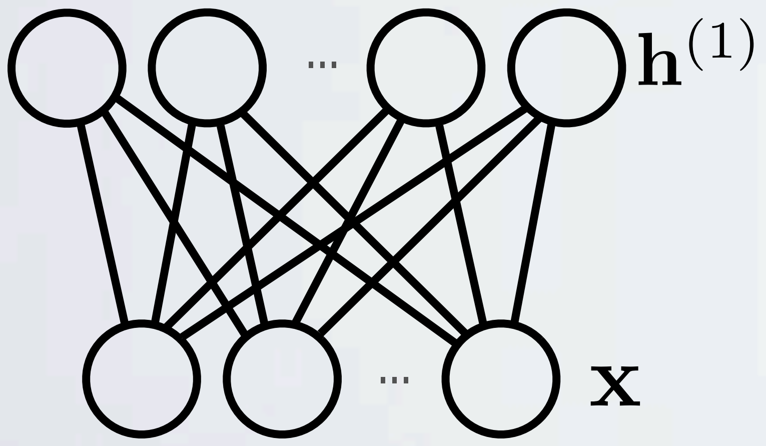
- This is where the RBM stacking procedure comes from
  - idea: improve prior on last layer by adding another hidden layer
  - how do we train these additional layers?

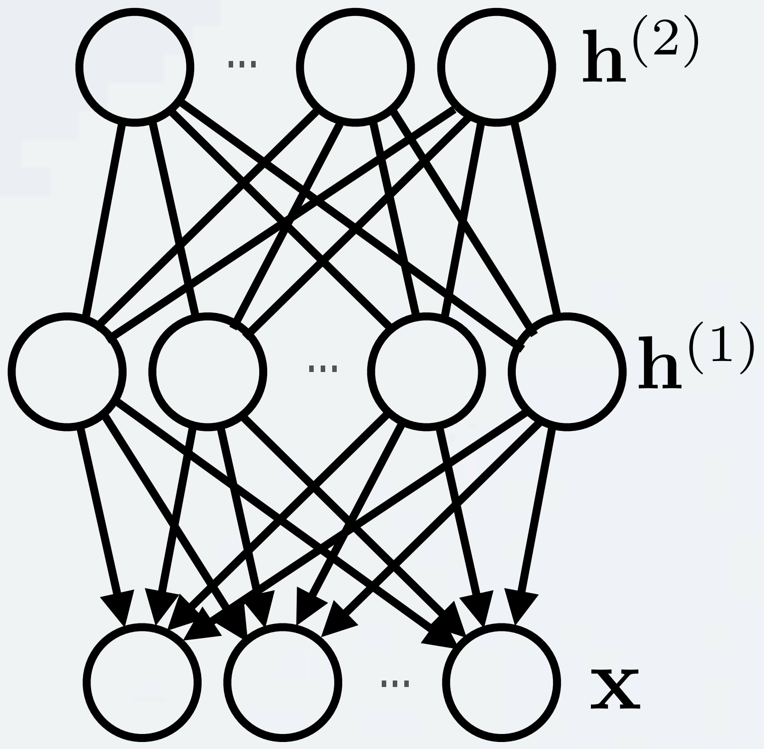


DEEP BELIEF NETWORK

Topics: deep belief network

- This is where the RBM stacking procedure comes from
  - idea: improve prior on last layer by adding another hidden layer
  - how do we train these additional layers?





DEEP BELIEF NETWORK

Topics: variational bound

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

  - if $q(\mathbf{h}^{(1)}|\mathbf{x})$ is equal to the true conditional $p(\mathbf{h}^{(1)}|\mathbf{x})$, then we have an equality
  - the more $q(\mathbf{h}^{(1)}|\mathbf{x})$ is different from $p(\mathbf{h}^{(1)}|\mathbf{x})$ the less tight the bound is
  - in fact, the difference between the left and right terms is the KL divergence between $q(\mathbf{h}^{(1)}|\mathbf{x})$ and $p(\mathbf{h}^{(1)}|\mathbf{x})$:

$$\mathrm{KL}(q||p) = \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log\left(\frac{q(\mathbf{h}^{(1)}|\mathbf{x})}{p(\mathbf{h}^{(1)}|\mathbf{x})}\right)$$

DEEP BELIEF NETWORK

Topics: variational bound

- This is called a variational bound

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)})\right) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

  - for a single hidden layer DBN (i.e. an RBM), both $p(\mathbf{x}|\mathbf{h}^{(1)})$ and $p(\mathbf{h}^{(1)})$ depend on the parameters of the first layer
  - when adding a second layer, we model $p(\mathbf{h}^{(1)})$ using a separate set of parameters
    - they are the parameters of the RBM involving $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$
    - $p(\mathbf{h}^{(1)})$ is now the marginalization of the second hidden layer $p(\mathbf{h}^{(1)}) = \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)})$

DEEP BELIEF NETWORK

Topics: variational bound

- This is called a variational bound

adding 2nd layer means untying the parameters in

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right)$$
$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

- we can train the parameters of the new second layer by maximizing the bound
  - this is equivalent to minimizing the following, since the other terms are constant:

$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{h}^{(1)})$$

  - this is like training an RBM on data generated from $q(\mathbf{h}^{(1)}|\mathbf{x})$!

DEEP BELIEF NETWORK

Topics: variational bound

- This is called a variational bound

adding 2nd layer means
untying the parameters in

$$\log p(\mathbf{x}) \quad \geq \quad \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \left(\log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \right)$$
$$- \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log q(\mathbf{h}^{(1)}|\mathbf{x})$$

- for $q(\mathbf{h}^{(1)}|\mathbf{x})$ we use the posterior of the first layer RBM
  - equivalent to a feed-forward (sigmoidal) layer, followed by sampling
- by initializing the weights of the second layer RBM as the transpose of the first layer weights, the bound is initially tight
  - a 2 layer DBN with tied weights is equivalent to a 1 layer RBM

DEEP BELIEF NETWORK

Topics: variational bound

- This process of adding layers can be repeated recursively
  - we obtain the greedy layer-wise pre-training procedure for neural networks
- We now see that this procedure corresponds to maximizing a bound on the likelihood of the data in a DBN
  - in theory, if our approximation $q(\mathbf{h}^{(1)}|\mathbf{x})$ is very far from the true posterior, the bound might be very loose
  - this only means we might not be improving the true likelihood
  - we might still be extracting better features!
- Fine-tuning is done by the Up-Down algorithm
  - A fast learning algorithm for deep belief nets. Hinton, Teh, Osindero, 2006.