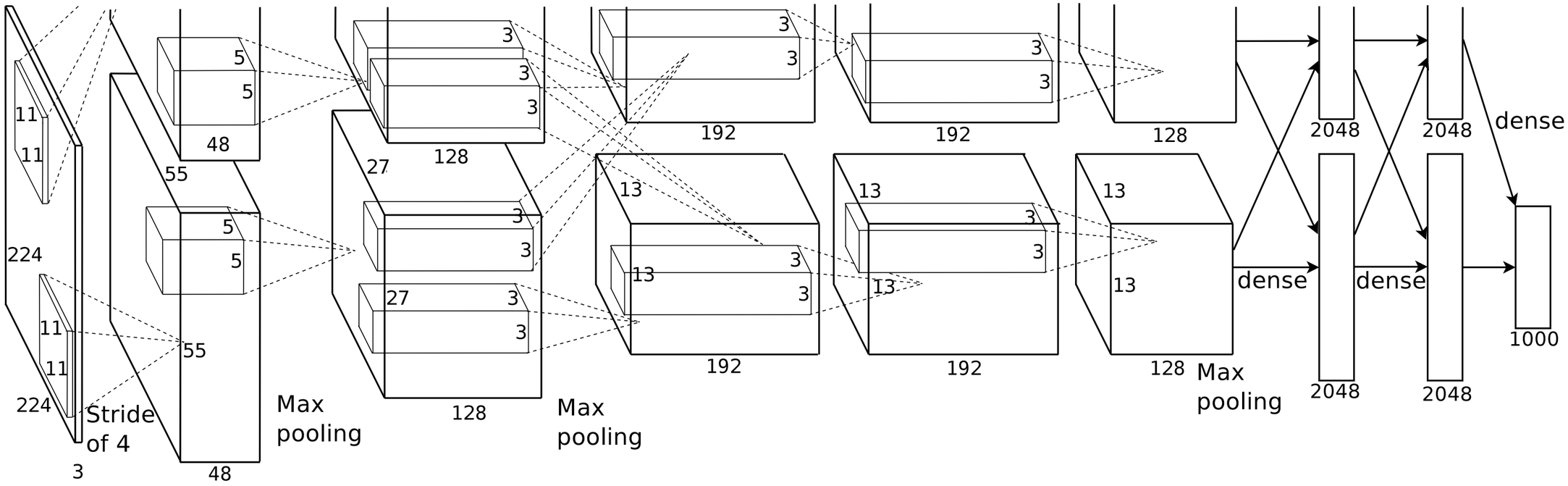
UNSUPERVISED LEARNING

GRAHAM TAYLOR
SCHOOL OF ENGINEERING
UNIVERSITY OF GUELPH

Deep Learning for Computer Vision Tutorial @ CVPR 2014
Columbus, OH

Motivation

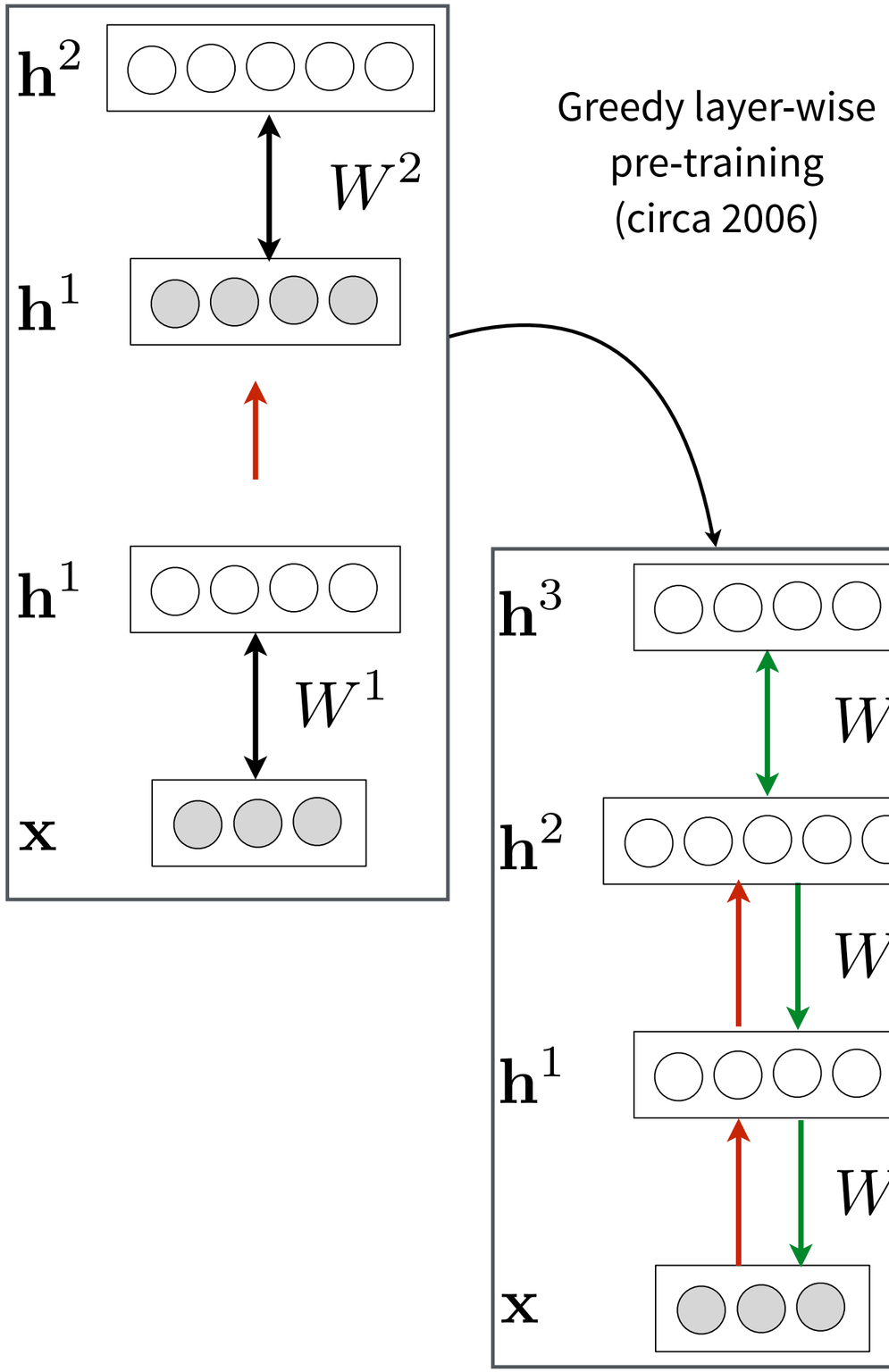


- Most impressive results in deep learning have been obtained with purely supervised learning methods (see previous talk)
- In vision, typically classification (e.g. object recognition)
- Though progress has been slower, it is likely that unsupervised learning will be important to future advances in DL

Image: Krizhevsky (2012) - AlexNet, the “hammer” of DL

An Interesting Historical Fact

- Unsupervised learning was the catalyst for the present DL revolution that started around 2006
- Now we can train deep supervised neural nets without “pre-training”, thanks to
  - Algorithms (nonlinearities, regularization)
  - More data
  - Better computers (e.g. GPUs)
- Should we still care about unsupervised learning?



Why Unsupervised Learning?

<u>Reason 1:</u>

We can exploit unlabelled data; much more readily available and often free.

Why Unsupervised Learning?

Reason 2:

We can capture enough information about the observed variables so as to ask new questions about them; questions that were not anticipated at training time.

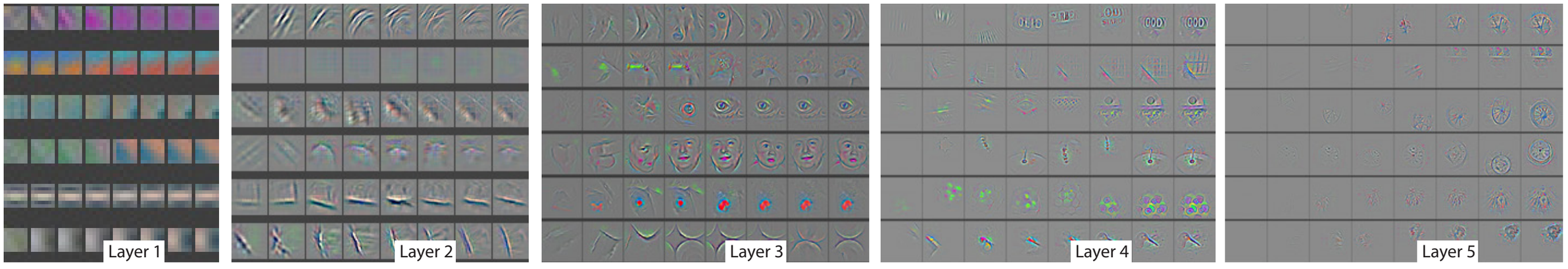


Image: Features from a convolutional net (Zeiler and Fergus, 2013)

Why Unsupervised Learning?

Reason 3:

Unsupervised learning has been shown to be a good regularizer for supervised learning; it helps generalize.

This advantage shows up in practical applications:

- transfer learning, domain adaptation
- unbalanced classes
- zero-shot, one-shot learning

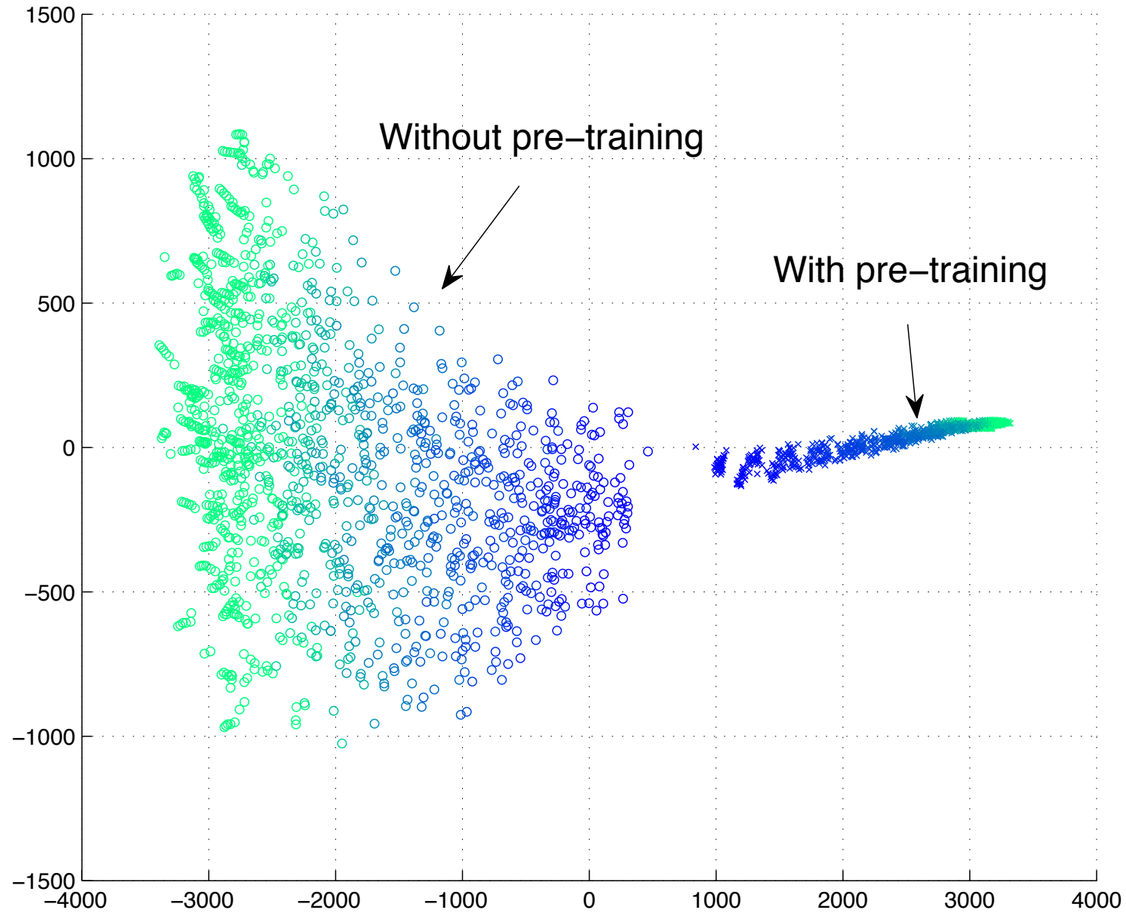
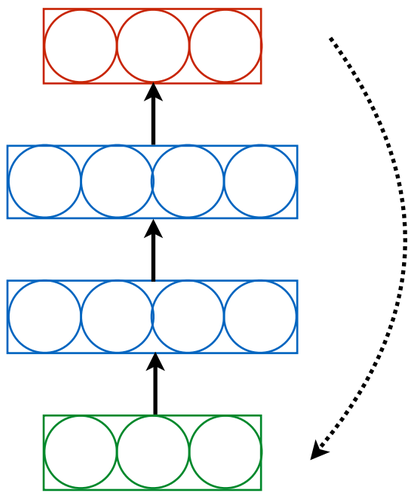


Image: ISOMAP embedding of functions represented by 50 networks w and w/o pre training (Erhan et al., 2010)

Why Unsupervised Learning?

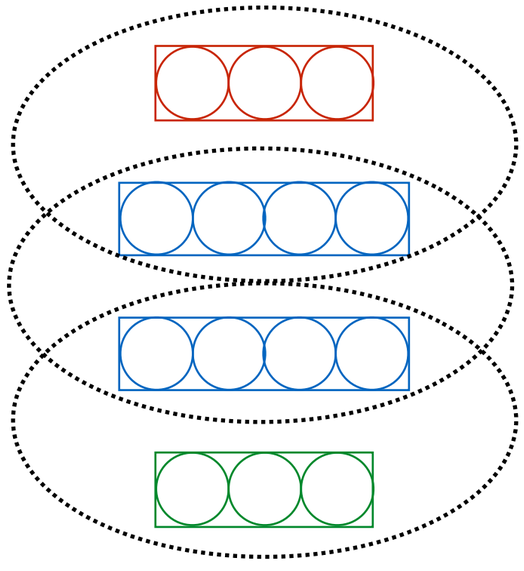
Reason 4:

There is evidence that unsupervised learning can be achieved mainly through a level-local training signal; compare this to supervised learning where the only signal driving parameter updates is available at the output and gets backpropagated.



Propagate credit

Supervised learning

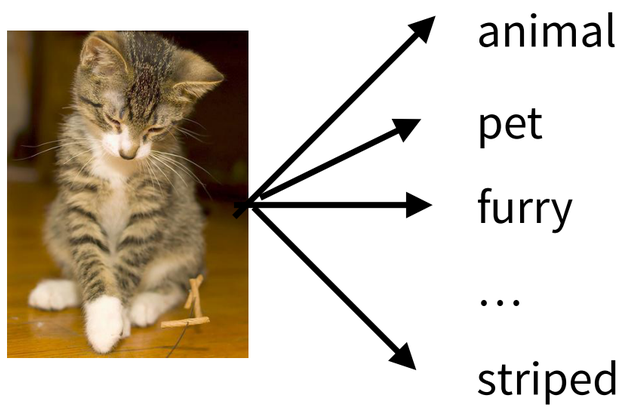


Local learning

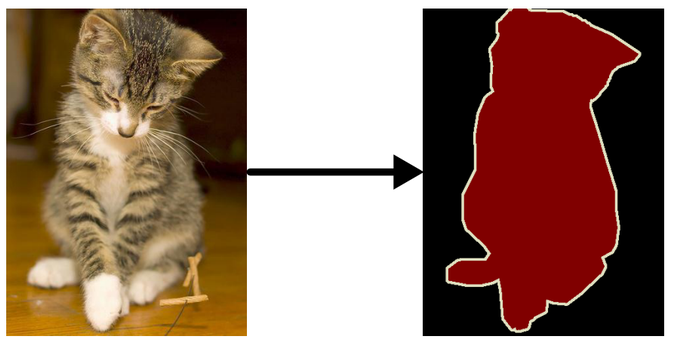
Why Unsupervised Learning?

Reason 5:

A recent trend in machine learning is to consider problems where the output is high-dimensional and has a complex, possibly multi-modal joint distribution. Unsupervised learning can be used in these "structured output" problems.



Attribute Prediction



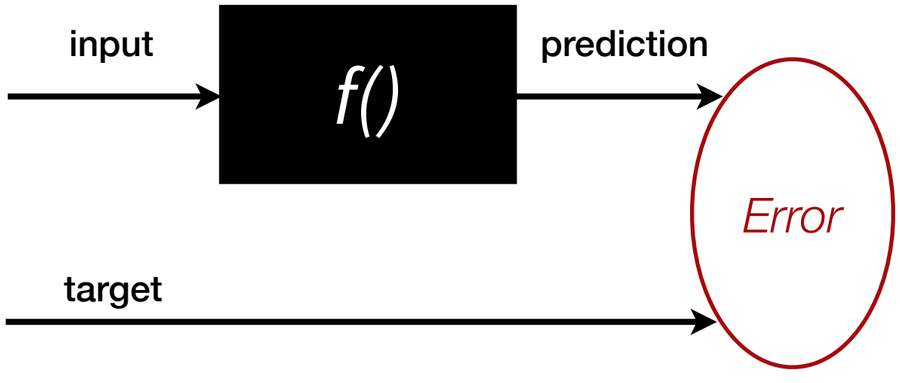
Segmentation

Learning Representations

- "Concepts" or "Abstractions" that help us make sense of the variability in data
- Often hand-designed to have desirable properties: e.g. sensitive to variables we want to predict, less sensitive to other factors explaining variability
- DL has leveraged the ability to learn representations
  - these can be task-specific or task-agnostic

Supervised Learning of Representations

- Learn a representation with the objective of selecting one that is best suited for predicting targets given input



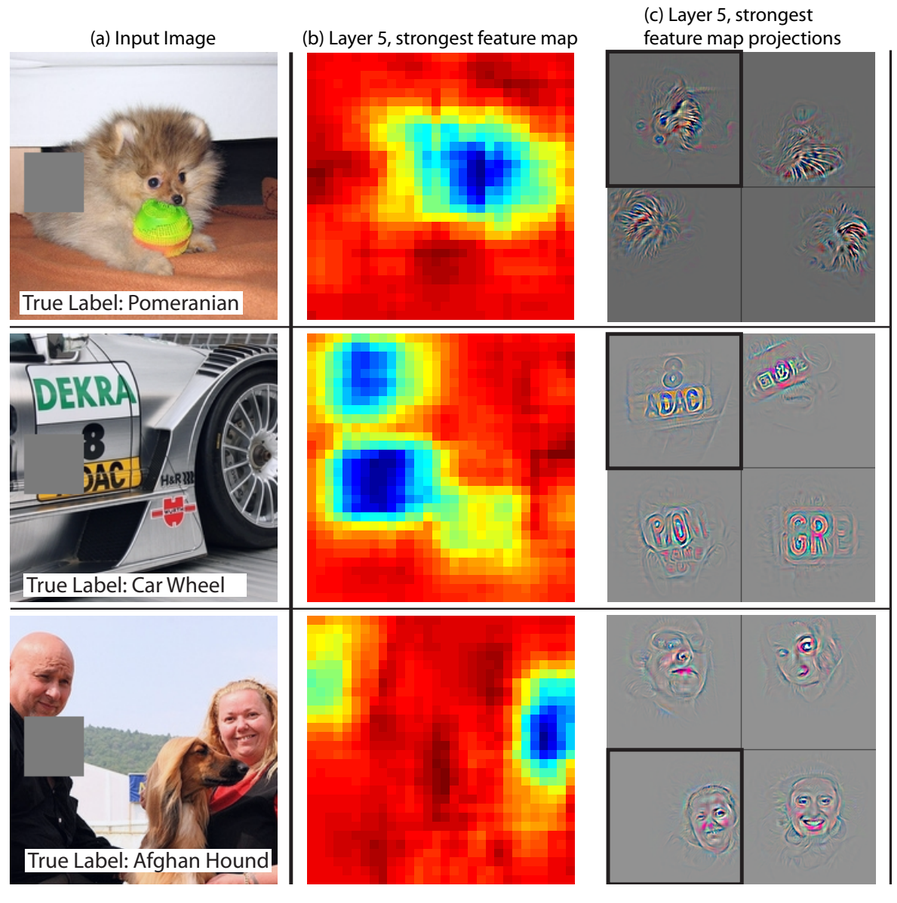
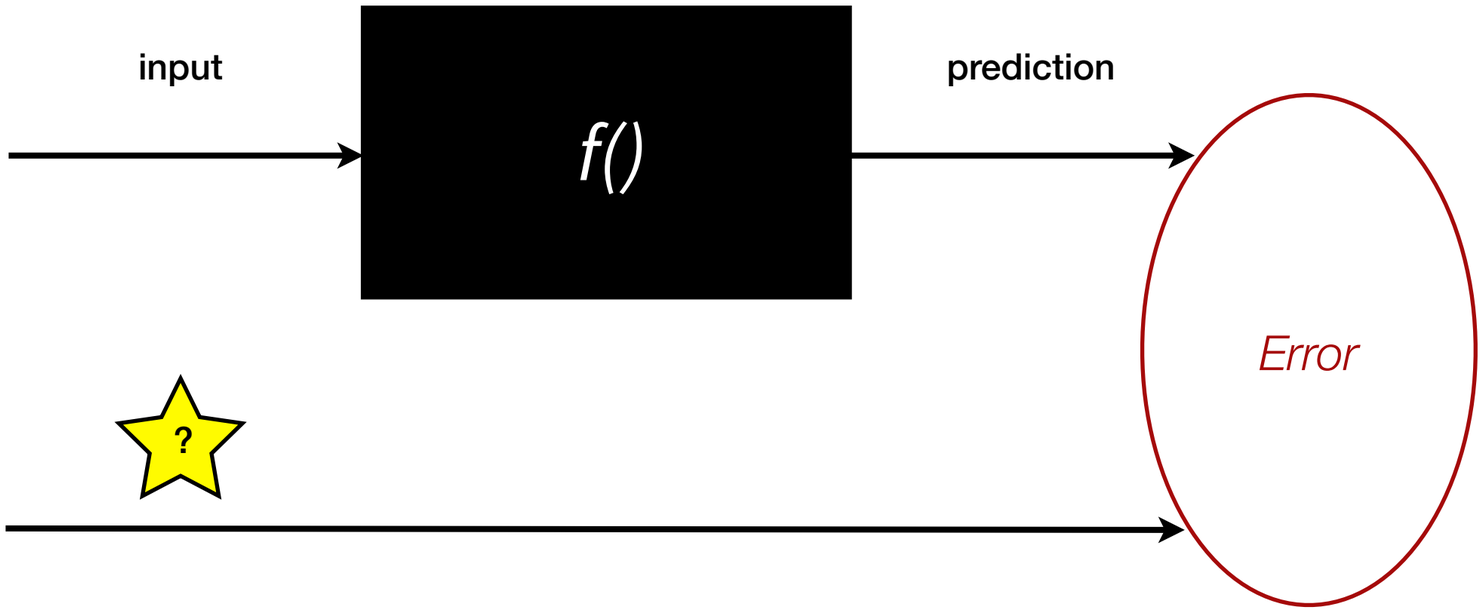


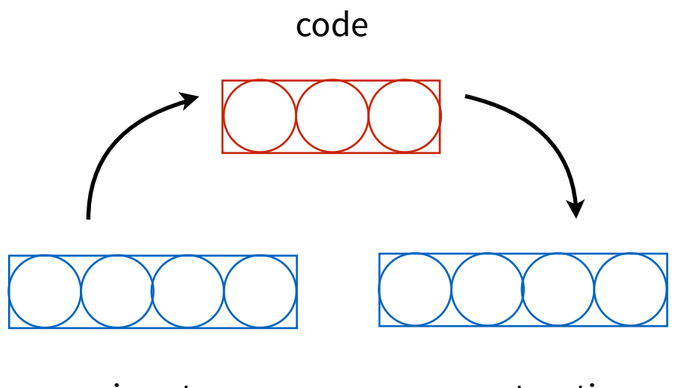
Image: Features from a convolutional net (Zeiler and Fergus, 2013)

Unsupervised Learning of Representations



Unsupervised learning of representations

- What is the objective?
  - reconstruction error?
  - maximum likelihood?
  - disentangle factors of variation?



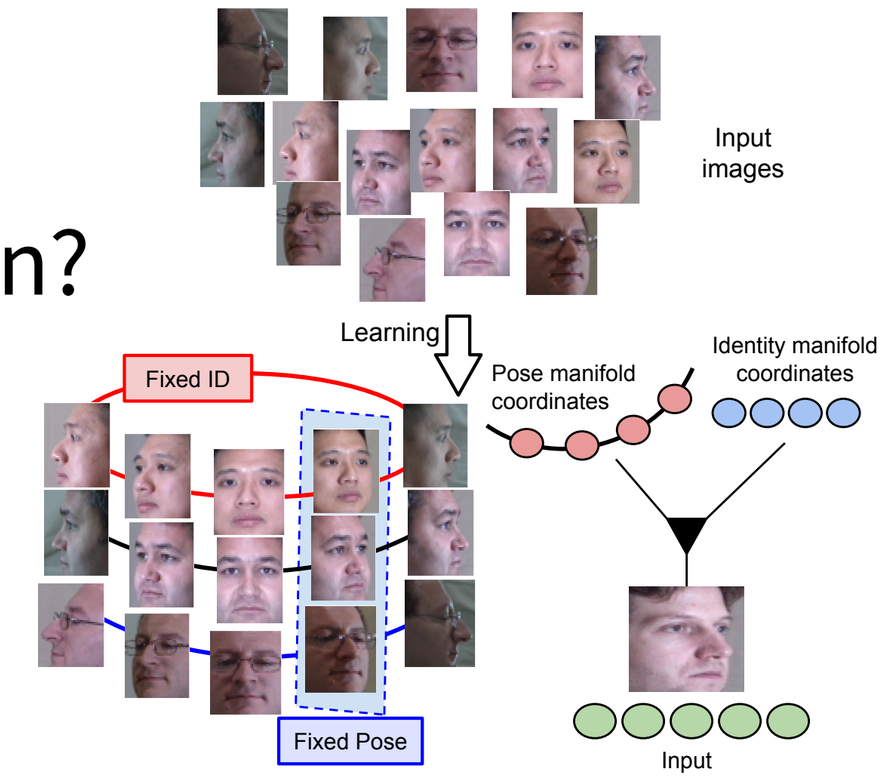


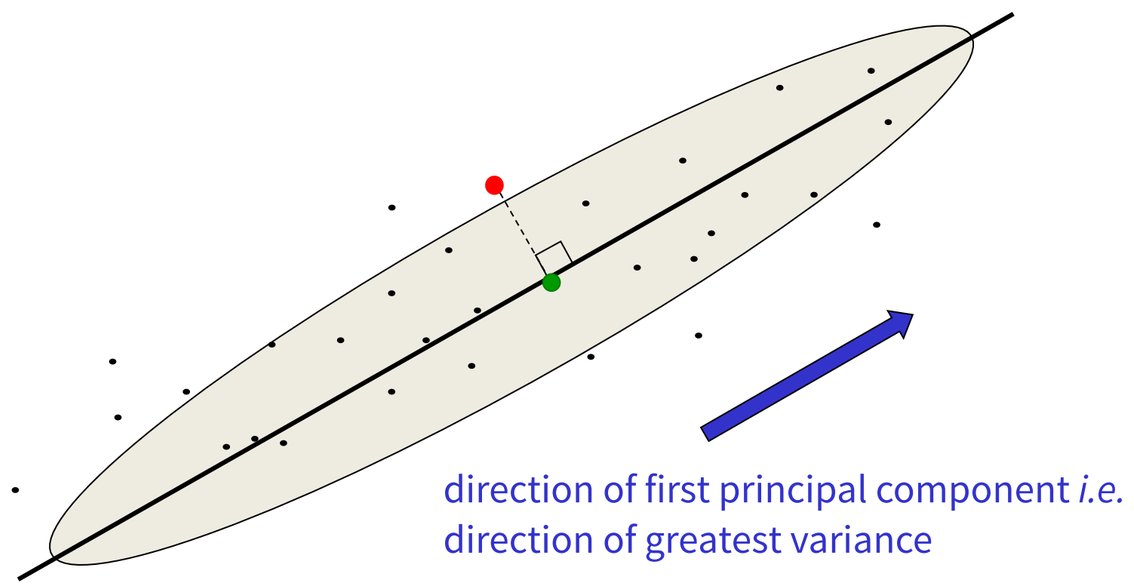
Image: Lee et al. 2014

Overview (for the remainder of the talk)

- Unsupervised building blocks of Deep Learning
  - Auto-encoders
  - Restricted Boltzmann Machines
- Their use in Deep Architectures (how/why?)
- Practical considerations

Principal Components Analysis

- PCA works well when the data is near a linear manifold in high-dimensional space
- Project the data onto this subspace spanned by principal components
- In dimensions orthogonal to the subspace the data has low variance

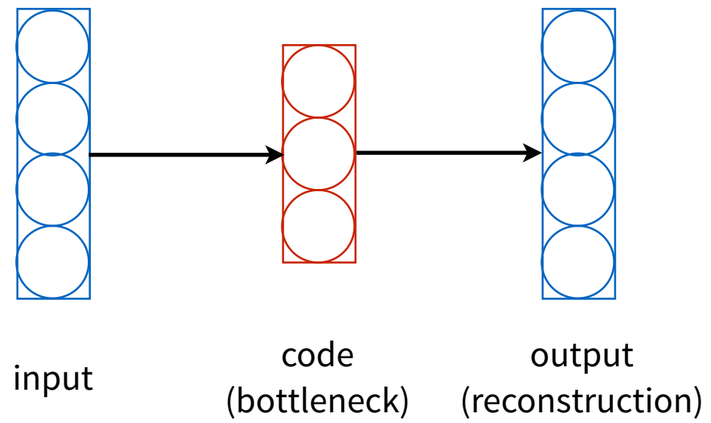


Credit: Geoff Hinton

An inefficient way to fit PCA

- Train a neural network with a “bottleneck” hidden layer

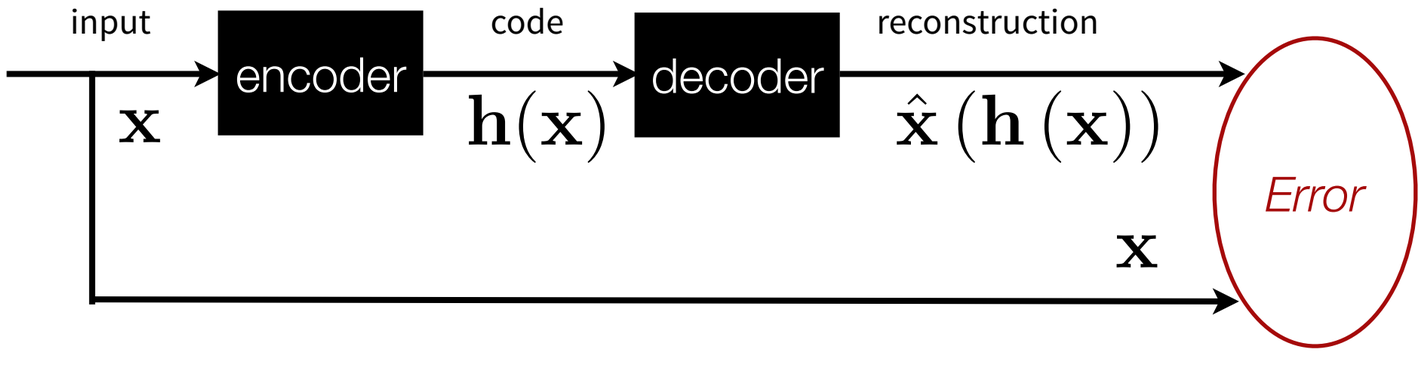
- Try to make the output the same as the input



- If the hidden and output layers are linear, and we minimize squared reconstruction error:
  - The M hidden units will span the same space as the first M principal components
  - But their weight vectors will not be orthogonal
  - And they will have approximately equal variance

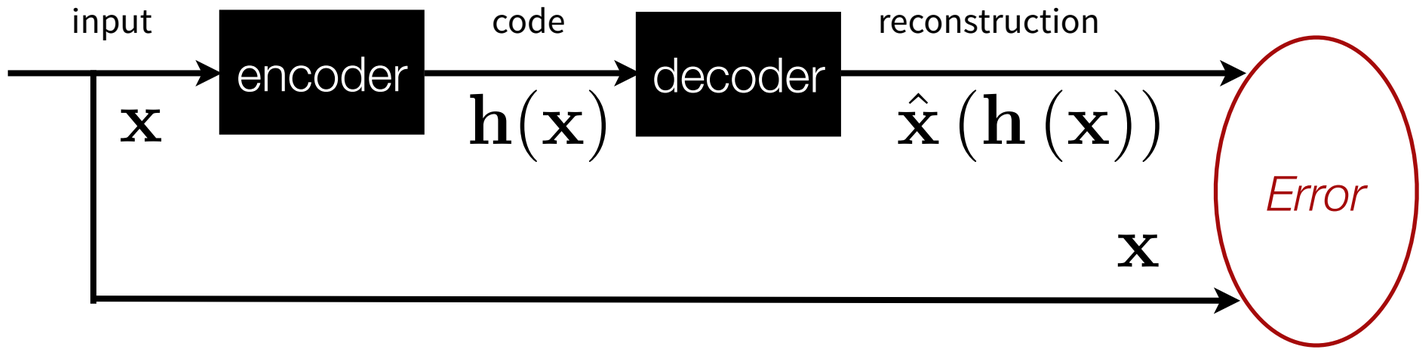
Credit: Geoff Hinton

Why fit PCA inefficiently?



- With nonlinear layers before and after the code, it should be possible to represent data that lies on or near a nonlinear manifold
  - the encoder maps from data space to co-ordinates on the manifold
  - the decoder does the inverse transformation
- The encoder/decoder can be rich, multi-layer functions

Auto-encoder

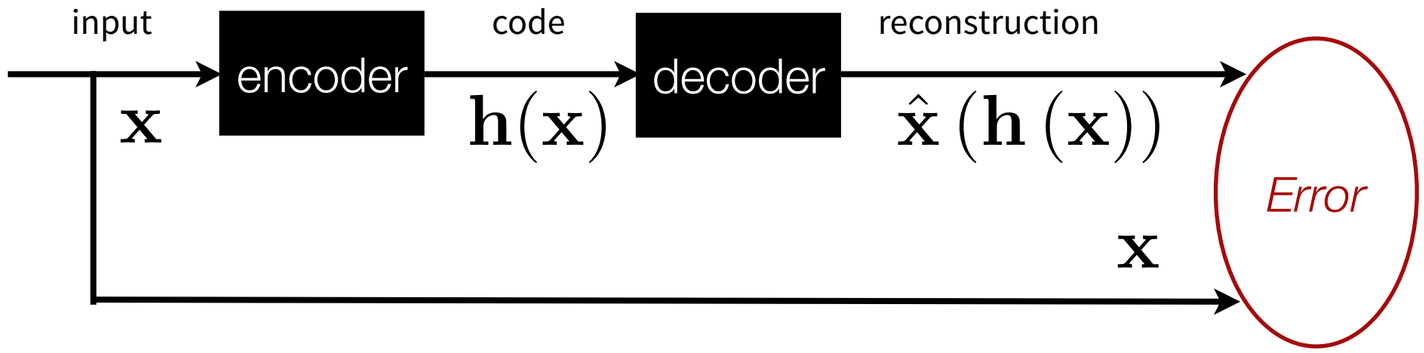


- Feed-forward architecture

- Trained to minimize reconstruction error

  - bottleneck or regularization essential

Auto-encoder



- Feed-forward architecture
- Trained to minimize reconstruction error
  - bottleneck or regularization essential

Example: real-valued data

Encoder

$$h_j(\mathbf{x}) = \sigma\left(\sum_i w_{ji} x_i\right)$$

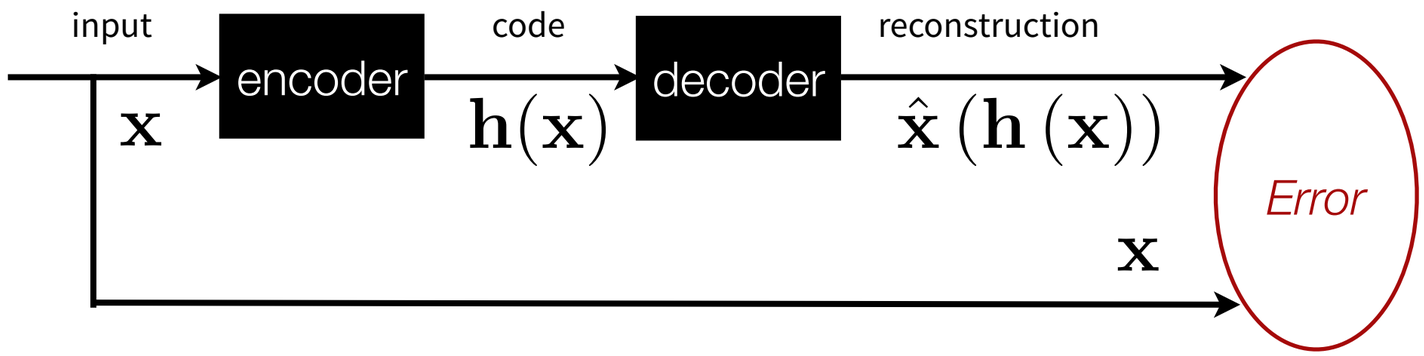
Decoder

$$\hat{x}_i(\mathbf{h}(\mathbf{x})) = \sum_j w_{ji} h_j(\mathbf{x})$$

Error

$$E = \sum_\alpha \left(\hat{\mathbf{x}}(\mathbf{h}(\mathbf{x}^\alpha)) - \mathbf{x}^\alpha\right)^2$$

Regularized Auto-encoders



- Permit code to be higher-dimensional than the input

- Capture structure of the training distribution due to predictive opposition b/w reconstruction distribution and regularizer

- Regularizer tries to make enc/dec as simple as possible

Simple?

Simple?

- Reconstruct the input from the code and make the code compact
  (PCA, auto-encoder with bottleneck)

Simple?

- Reconstruct the input from the code and make the code compact
  (PCA, auto-encoder with bottleneck)
- Reconstruct the input from the code and make the code sparse
  (sparse auto-encoders)

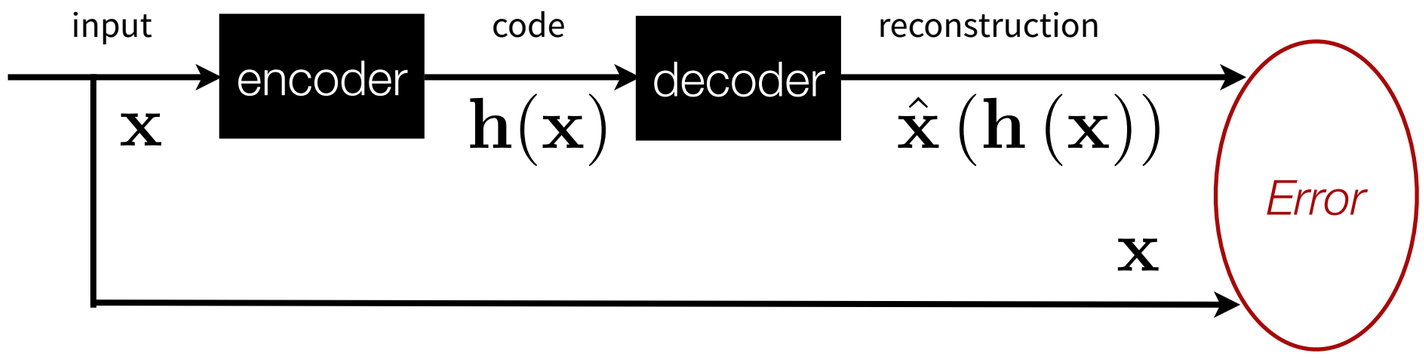
Simple?

- Reconstruct the input from the code and make the code compact
  (PCA, auto-encoder with bottleneck)
- Reconstruct the input from the code and make the code sparse
  (sparse auto-encoders)
- Add noise to the input or code and reconstruct the cleaned-up version
  (denoising auto-encoders)

Simple?

- Reconstruct the input from the code and make the code compact
  (PCA, auto-encoder with bottleneck)
- Reconstruct the input from the code and make the code sparse
  (sparse auto-encoders)
- Add noise to the input or code and reconstruct the cleaned-up version
  (denoising auto-encoders)
- Reconstruct the input from the code and make the code insensitive to the input (contractive auto-encoders)

Sparse Auto-encoders



$$\mathcal{L}_{\mathrm{SAE}} = \mathbb{E}[l(\mathbf{x}, \hat{\mathbf{x}}(\mathbf{h}(\mathbf{x})))] + \beta \sum_j \mathrm{KL}(\rho || \hat{\rho}_j)$$

$$\hat{\rho}_j = \frac{1}{N} \sum_i^N h_j(\mathbf{x}_i) \text{ : mean activation}$$

ρ : target activation (small)

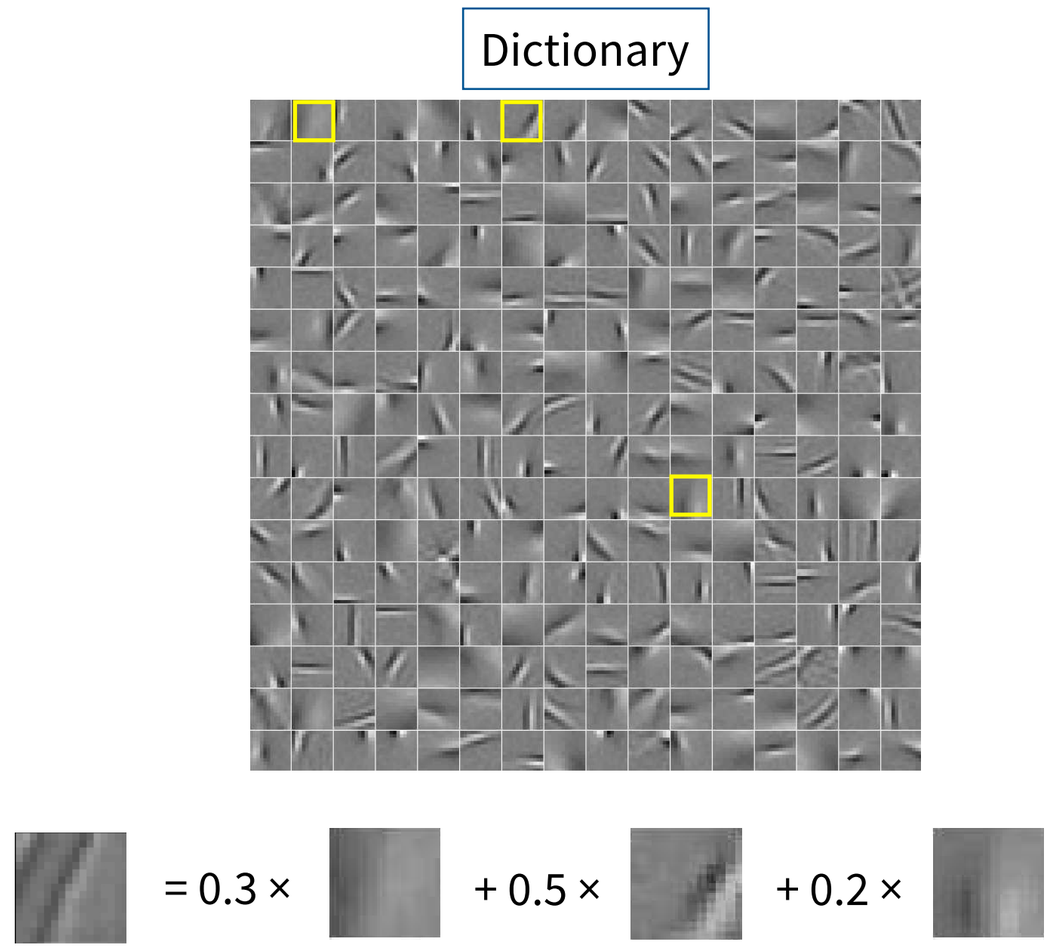
- Apply a sparsity penalty to the hidden activations
- Also see Predictive Sparse Decomposition (Kavukcuoglu et al. 2008)

Diversion: Sparse Coding

- Inputs as sparse linear combinations of basis elements

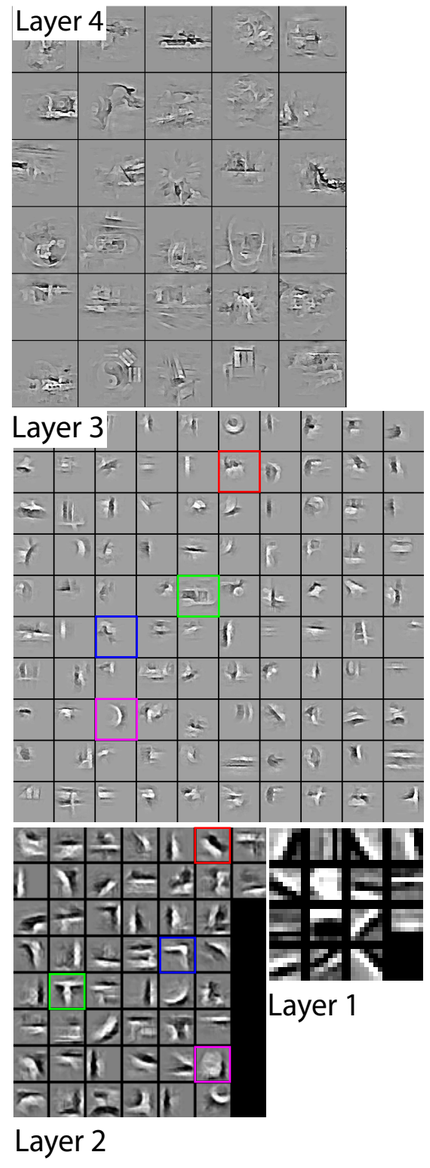
- Linear decoder, no encoder

  - relies on optimization for inference



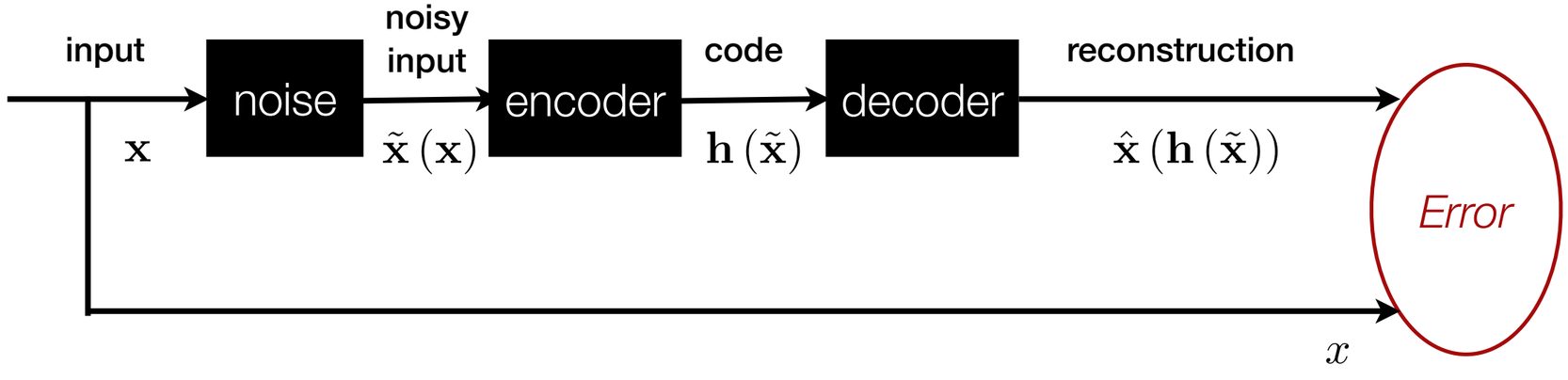
Deconvolutional Networks

- Deep convolutional sparse coding
- Trained to reconstruct the input from any layer
- Fast approximate inference
- Recently used to visualize features learned by convolutional nets (Zeiler and Fergus 2013)



(Vincent et al. 2008)

Denoising Auto-encoders



$$\mathcal{L}_{\mathrm{DAE}} = \mathbb{E}\left[l\left(\mathbf{x}, \hat{\mathbf{x}}\left(\mathbf{h}\left(\tilde{\mathbf{x}}\right)\right)\right)\right]$$

$$\tilde{\mathbf{x}}(\mathbf{x}) = \mathbf{x} + \epsilon$$

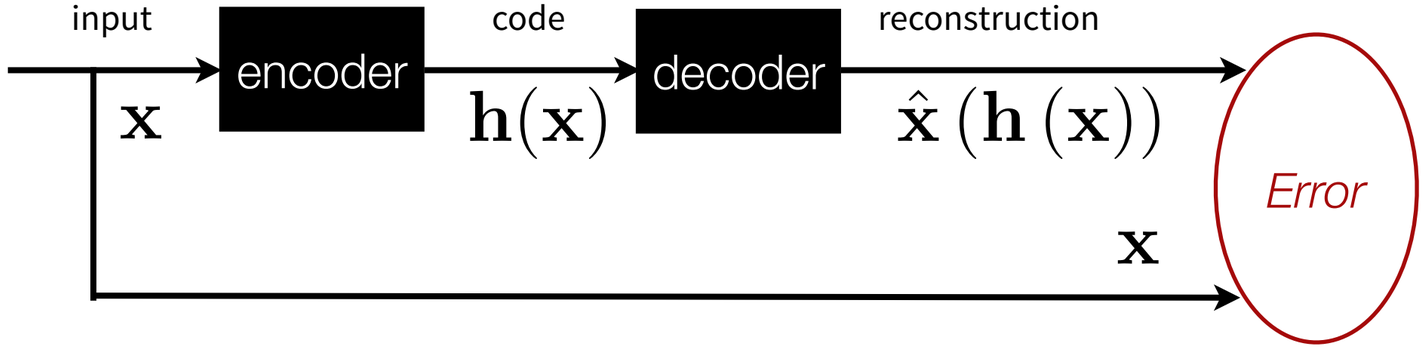
$$\epsilon \sim \mathcal{N}\left(\mathbf{0}, \sigma^2 I\right)$$

only one possible choice of noise model

- The code can be viewed as a lossy compression of the input
- Learning drives it to be a good compressor for training examples (and hopefully others as well) but not arbitrary inputs

(Rifai et al. 2011)

Contractive Auto-encoders



$$\mathcal{L}_{\mathrm{CAE}} = \mathbb{E}\left[l\left(\mathbf{x}, \hat{\mathbf{x}}\left(\mathbf{h}\left(\mathbf{x}\right)\right)\right) + \lambda \left|\left| \frac{\partial \mathbf{h}\left(\mathbf{x}\right)}{\partial \mathbf{x}} \right|\right|^2 \right]$$

$$\mathbf{h}\left(\mathbf{x}\right) = \mathrm{sigmoid}\left(W\mathbf{x} + b\right)$$

$$\hat{\mathbf{x}}\left(\mathbf{h}\left(\mathbf{x}\right)\right) = \mathrm{sigmoid}\left(W^T\mathbf{h} + c\right)$$

- Learn good models of high-dimensional data (Bengio et al. 2013)
- Can obtain good representations for classification
- Can produce good quality samples by a random walk near the manifold of high density (Rifai et al. 2012)

What do Denoising Auto-encoders Learn?

What do Denoising Auto-encoders Learn?

- The reconstruction function locally characterizes the data generating density (Alain and Bengio 2013)
    - derivative of the log-density (score) with respect to the input
    - second derivative of the density
    - other local properties

What do Denoising Auto-encoders Learn?

- The reconstruction function locally characterizes the data generating density (Alain and Bengio 2013)
  - derivative of the log-density (score) with respect to the input
  - second derivative of the density
  - other local properties
- Bengio et al. (2013) generalized this result to arbitrary variables (discrete, continuous, or both), arbitrary corruption, arbitrary loss function

Advanced Autoencoders

Advanced Autoencoders

- (Bengio et al. 2013) also showed a way to sample from an autoencoder by running a Markov chain that alternately adds noise and denoises

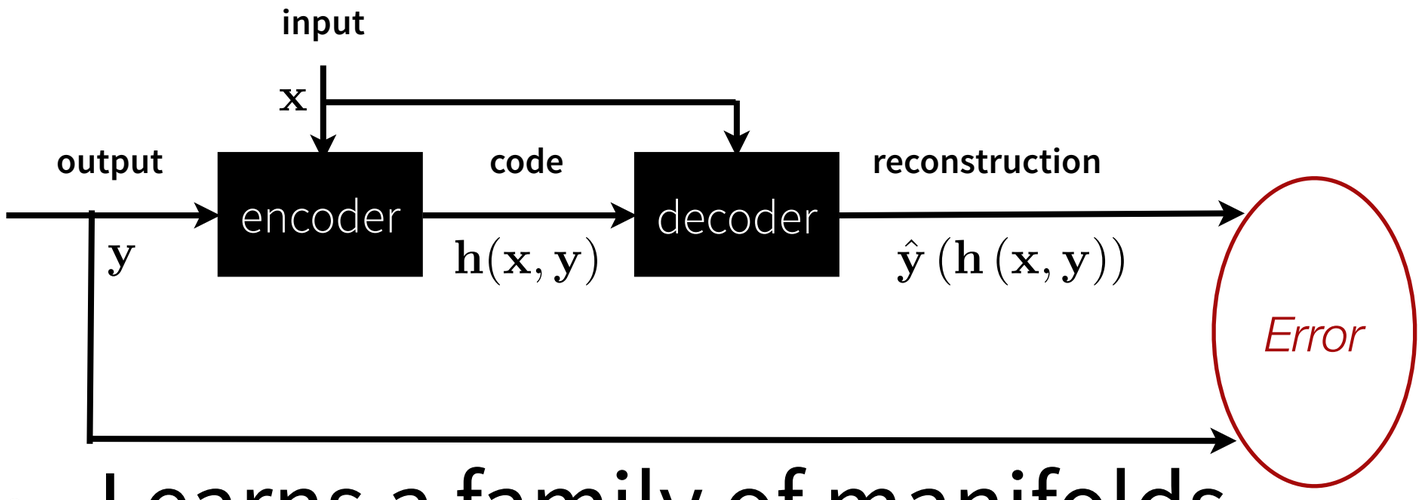
Advanced Autoencoders

- (Bengio et al. 2013) also showed a way to sample from an autoencoder by running a Markov chain that alternately adds noise and denoises
- (Kamyshanska and Memisivec 2013) demonstrate a way to score data under an autoencoder

Advanced Autoencoders

- (Bengio et al. 2013) also showed a way to sample from an autoencoder by running a Markov chain that alternately adds noise and denoises
- (Kamyshanska and Memisivec 2013) demonstrate a way to score data under an autoencoder
- Generative Stochastic Networks (Bengio et al. 2013) are an intriguing recent generalization of DAEs

Relational Autoencoders

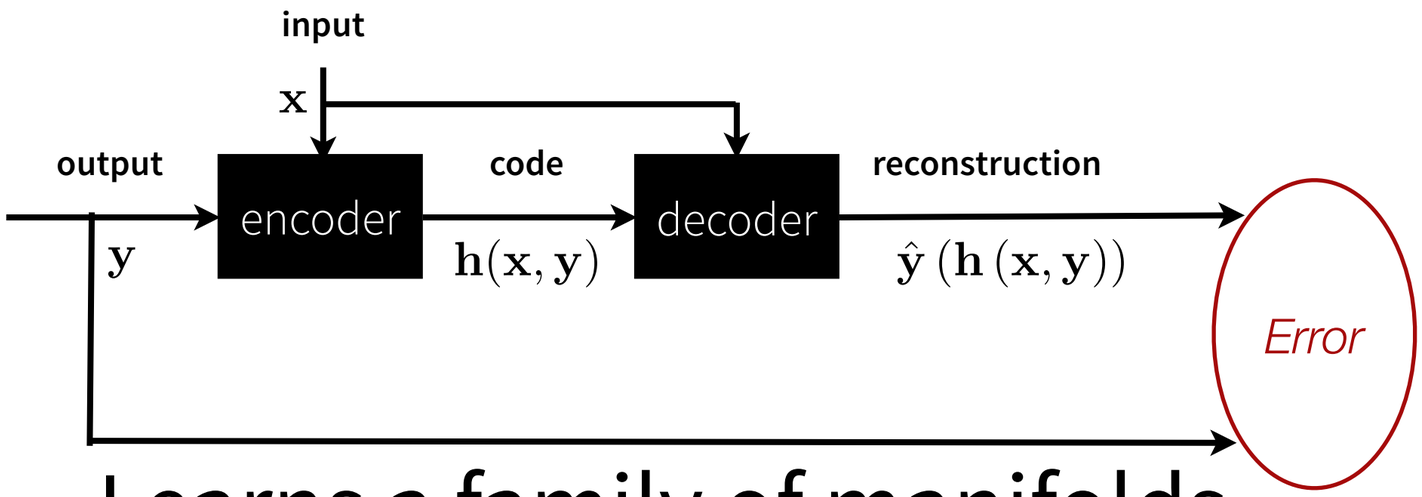


- Learns a family of manifolds (Memisevic 2011)

- Can be viewed as AE whose weights are modulated by input vector $w_{kj}(\mathbf{x}) = \sum_i \hat{w}^i_{kj} x_i$

- Used for modelling image transformations, extracting spatio-temporal features

Relational Autoencoders



- Learns a family of manifolds (Memisevic 2011)

Example: real-valued data

Encoder

$$h_k(\mathbf{x};\mathbf{y}) = \sigma\left(\sum_{ij} \hat{w}^i_{kj} x_i y_j\right)$$

Decoder

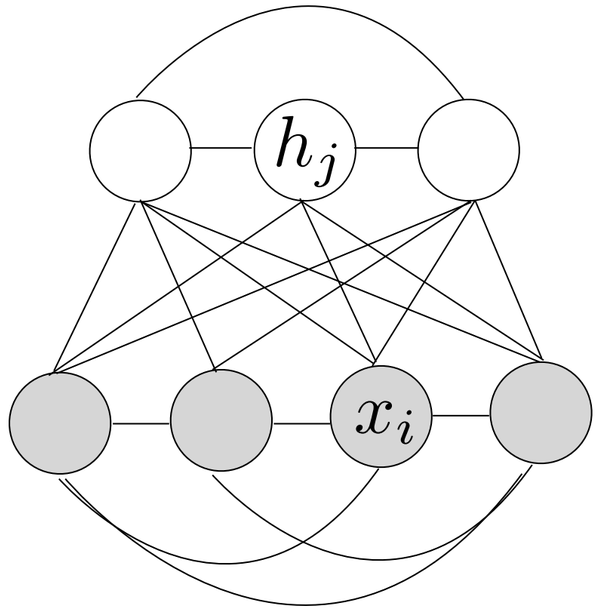
$$\hat{y}_j\left(\mathbf{h}\left(\mathbf{x};\mathbf{y}\right)\right) = \sum_{ki} \hat{w}^i_{kj} x_i h_k(\mathbf{x};\mathbf{y})$$

- Can be viewed as AE whose weights are modulated by input vector $w_{kj}\left(\mathbf{x}\right) = \sum_i \hat{w}^i_{kj} x_i$

- Used for modelling image transformations, extracting spatio-temporal features

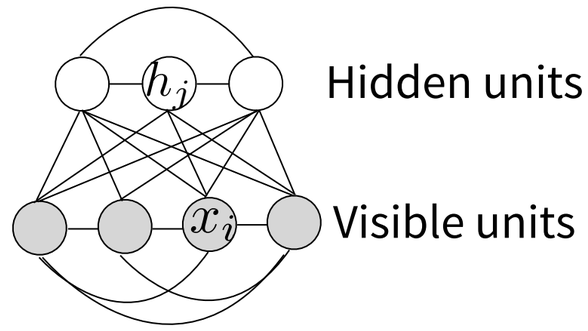
Boltzmann Machines

- Stochastic Hopfield Networks with hidden units
- Both visible and hidden units are binary
- Make the states of the hidden units form interpretations of the perceptual input presented at the visible units



Energy Function

- Energy-based model: negative energy assigns a "goodness" to every joint configuration



$E(\mathbf{x}, \mathbf{h})$ Energy function

- Can convert negative energies to probabilities by normalizing

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{x}, \mathbf{h})}$$

$$p(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}$$

Inference in a Boltzmann Machine

- The binary stochastic units make biased random decisions
- Probability of activating is a function of an “energy gap”

$$p(h_k = 1|\mathbf{x}, \{h_l\}\forall l \neq k) = \frac{1}{1 + e^{-\Delta E_k}}$$

- The number of possible hidden configurations is exponential so we need MCMC to sample from the posterior (this is slow!)

$$\Delta E_k = E(h_k = 0) - E(h_k = 1) = b_k + \sum_i x_i w_{ik} + \sum_l h_l w_{kl}$$

Learning in a Boltzmann Machine

- Goal: maximize the product of the probabilities that the Boltzmann machine assigns to the binary vectors in the training set
- Everything that one weight needs to know about the other weights and the data is contained in the difference of two correlations

$$\frac{\partial \log p(\mathbf{x})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Credit: Geoff Hinton

Learning in a Boltzmann Machine

- Goal: maximize the product of the probabilities that the Boltzmann machine assigns to the binary vectors in the training set
- Everything that one weight needs to know about the other weights and the data is contained in the difference of two correlations

$$\frac{\partial \log p(\mathbf{x})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Derivative of log prob of one training vector, **v**, under the model

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Credit: Geoff Hinton

Learning in a Boltzmann Machine

- Goal: maximize the product of the probabilities that the Boltzmann machine assigns to the binary vectors in the training set
- Everything that one weight needs to know about the other weights and the data is contained in the difference of two correlations

$$\frac{\partial \log p(\mathbf{x})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Derivative of log prob of one training vector, **v**, under the model

Expected value of product of states at thermal equilibrium when **v** is clamped on the visible units (positive phase)

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Credit: Geoff Hinton

Learning in a Boltzmann Machine

- Goal: maximize the product of the probabilities that the Boltzmann machine assigns to the binary vectors in the training set
- Everything that one weight needs to know about the other weights and the data is contained in the difference of two correlations

$$\frac{\partial \log p(\mathbf{x})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Derivative of log prob of one training vector, **v**, under the model

Expected value of product of states at thermal equilibrium when **v** is clamped on the visible units (positive phase)

Expected value of product of states at thermal equilibrium with no clamping (negative phase)

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{\mathbf{x}} - \langle s_i s_j \rangle_{\text{model}}$$

Credit: Geoff Hinton

Why do we need a negative phase?

$$p(\mathbf{x}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}}{\sum_{\mathbf{u}} \sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

Credit: Geoff Hinton

Why do we need a negative phase?

The positive phase finds hidden configurations that work well with **x** and lowers their energies

$$p(\mathbf{x}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}}{\sum_{\mathbf{u}} \sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

Credit: Geoff Hinton

Why do we need a negative phase?

$$p(\mathbf{x}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}}{\sum_{\mathbf{u}} \sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

The positive phase finds hidden configurations that work well with **x** and lowers their energies

The negative phase finds the joint configurations that are the best competitors and raises their energies

Credit: Geoff Hinton

How to inefficiently collect stats

Positive phase

- Clamp a data vector on the visible units and set the hidden units to random binary states

  - Update the hidden units one at a time until the network reaches “thermal equilibrium”

  - Sample $\langle s_i s_j \rangle$ for every connected pair of units

  - Repeat for all data vectors in the training set and average

Credit: Geoff Hinton

How to inefficiently collect stats

Positive phase

- Clamp a data vector on the visible units and set the hidden units to random binary states
  - Update the hidden units one at a time until the network reaches “thermal equilibrium”
  - Sample $\langle s_i s_j \rangle$ for every connected pair of units
  - Repeat for all data vectors in the training set and average

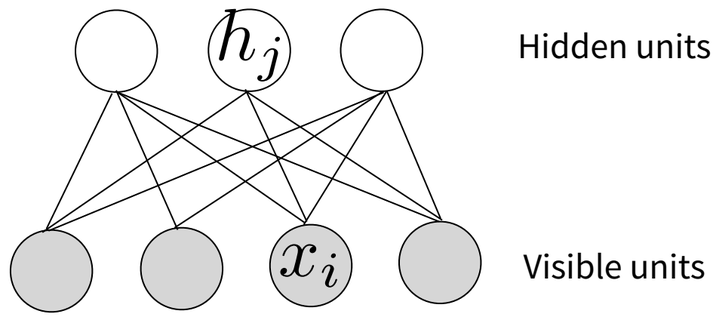
Negative phase

- Set all the units to random states
  - Update all the units one at a time until the network reaches thermal equilibrium
  - Sample $\langle s_i s_j \rangle$ for every connected pair of units
  - Repeat many times and average to get good estimates

Credit: Geoff Hinton

Restricted Boltzmann Machines

- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.

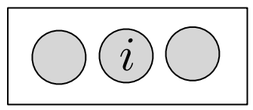


- In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped
  - So we can quickly get the exact value of $\langle x_i h_j \rangle_{\mathbf{x}}$

$$p(h_j = 1|\mathbf{x}) = \frac{1}{1 + e^{-(b_j + \sum_{i \in \text{vis}} x_i w_{ij})}}$$

Credit: Geoff Hinton

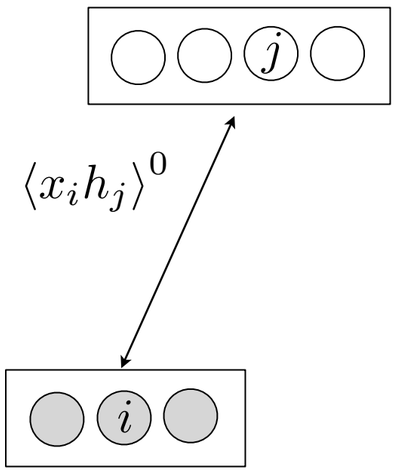
The Boltzmann Machine Learning Algorithm - RBMs



t=0

Credit: Geoff Hinton

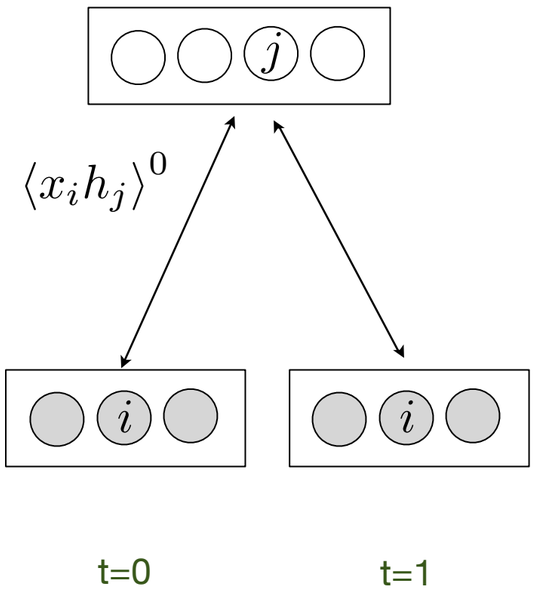
The Boltzmann Machine Learning Algorithm - RBMs



t=0

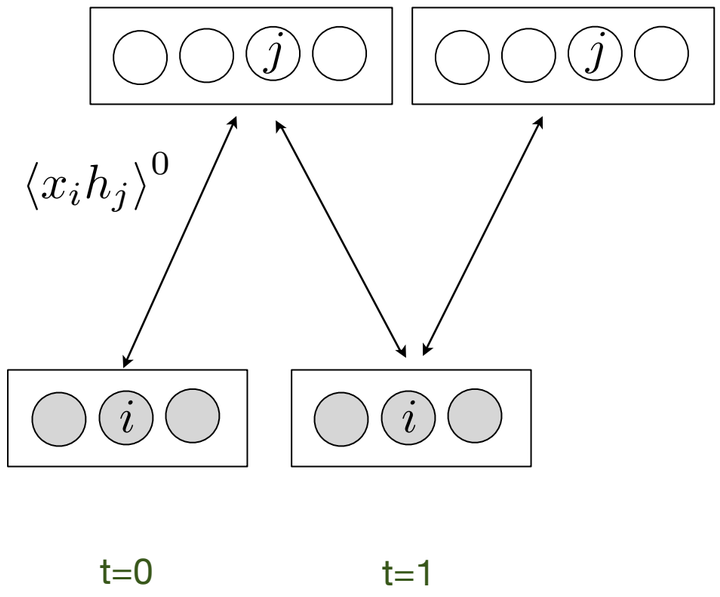
Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



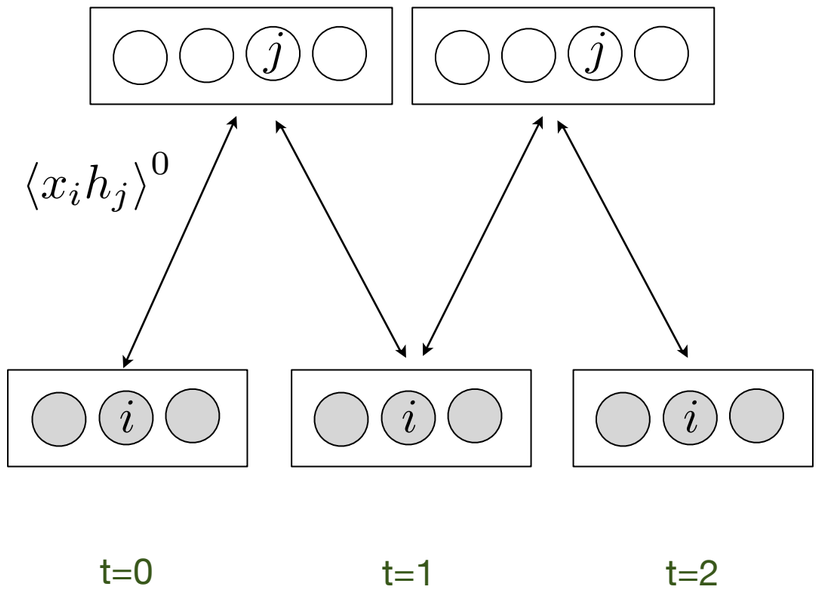
Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



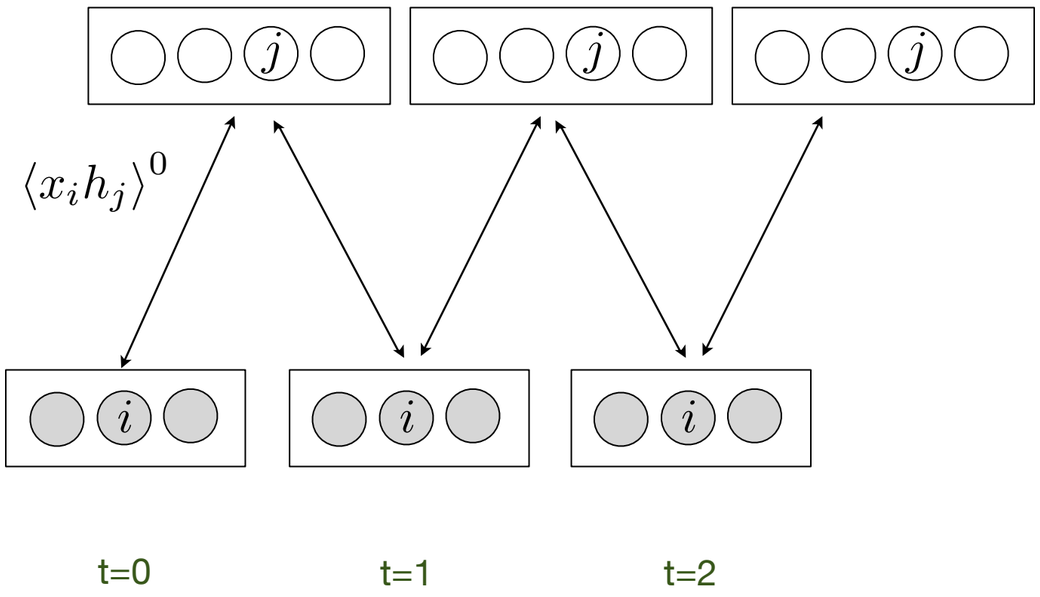
Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



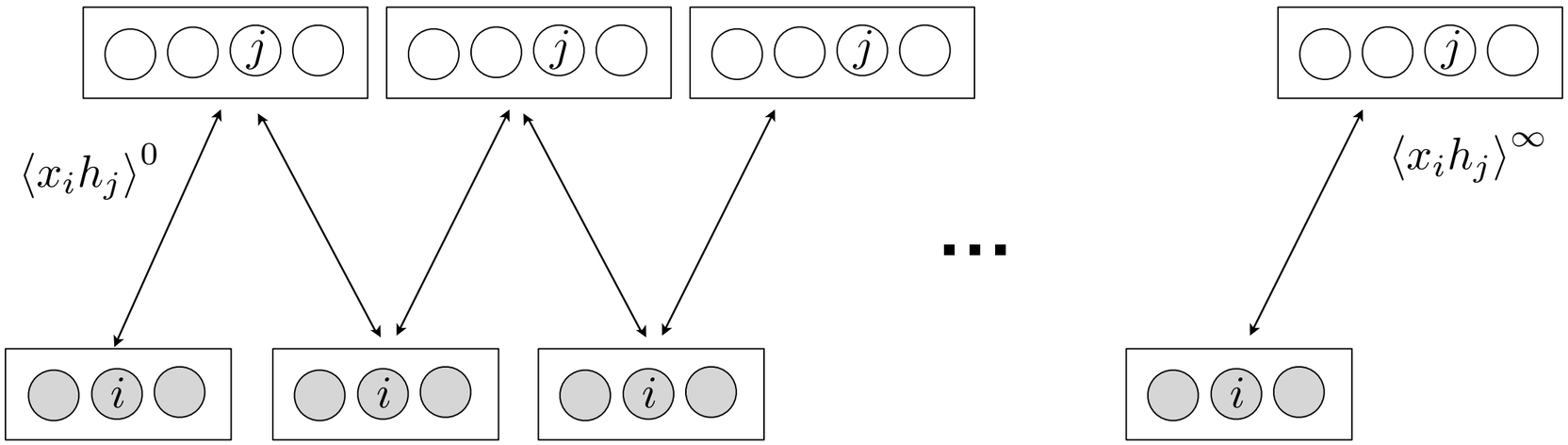
Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



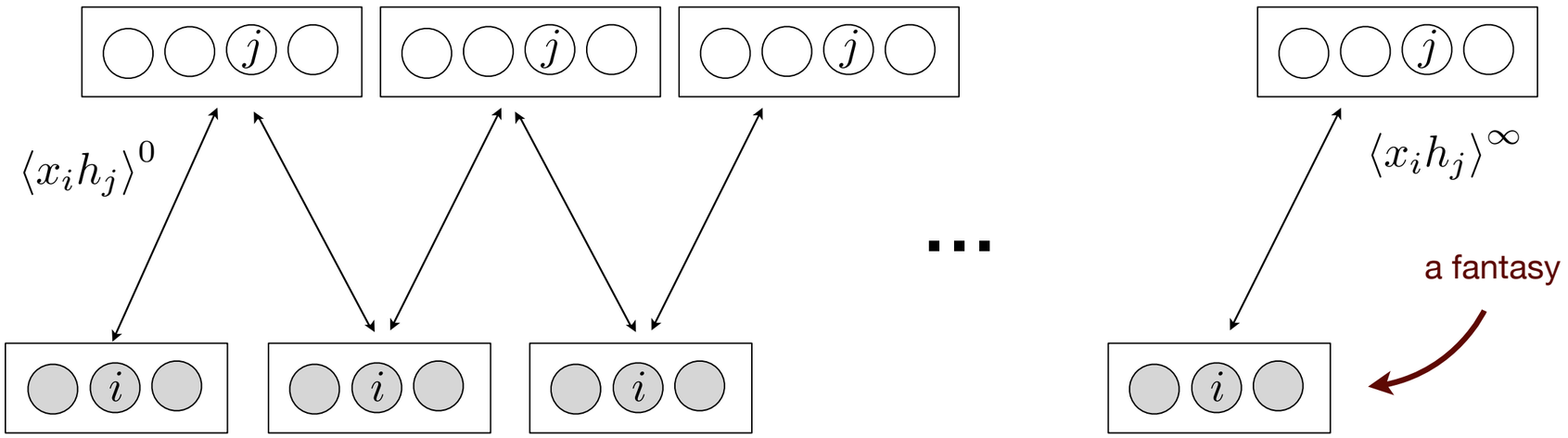
Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



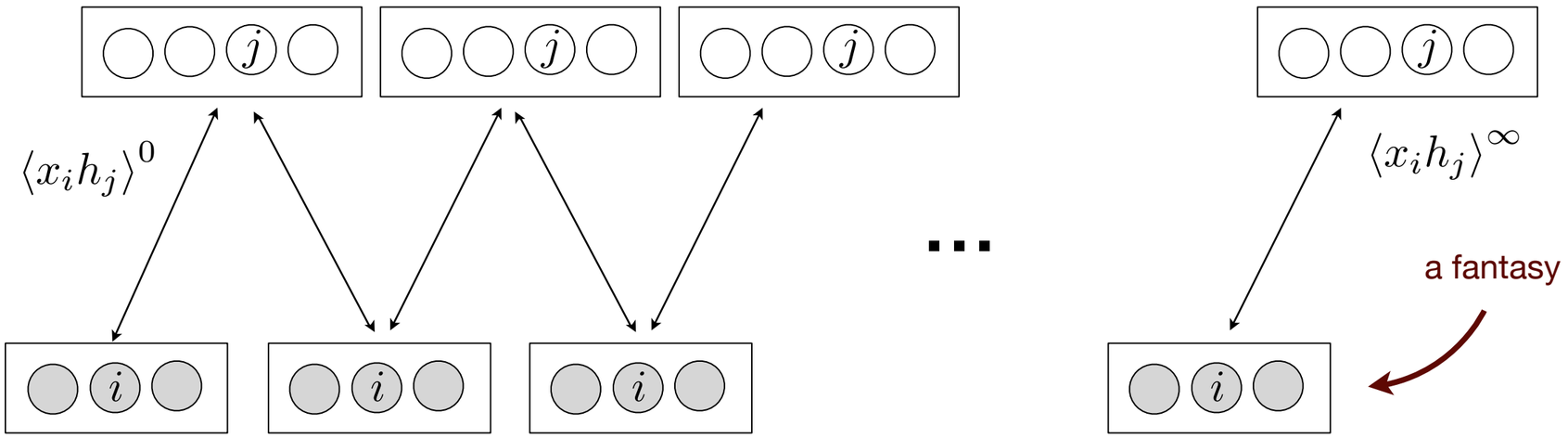
Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



Credit: Geoff Hinton

The Boltzmann Machine Learning Algorithm - RBMs



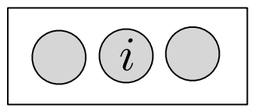
$$\Delta w_{ij} = \epsilon \left(\langle x_i h_j \rangle^0 - \langle x_i h_j \rangle^\infty \right)$$

$$\langle x_i h_j \rangle^0 = \langle x_i h_j \rangle_{\mathbf{x}}$$
$$\langle x_i h_j \rangle^\infty = \langle x_i h_j \rangle_{\text{model}}$$

Credit: Geoff Hinton

Contrastive Divergence

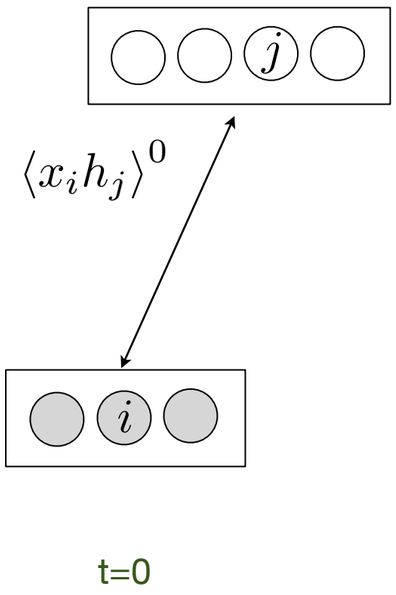
Instead of running the Markov chain to equilibrium, run for just one (or a few) steps!



t=0

Credit: Geoff Hinton

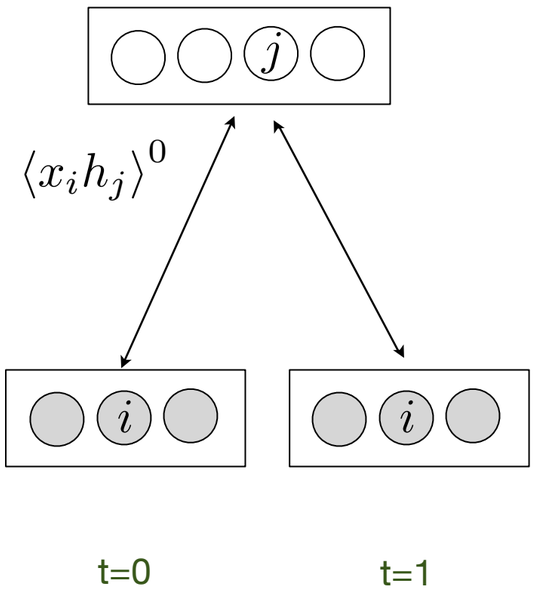
Contrastive Divergence



Instead of running the Markov chain to equilibrium, run for just one (or a few) steps!

Credit: Geoff Hinton

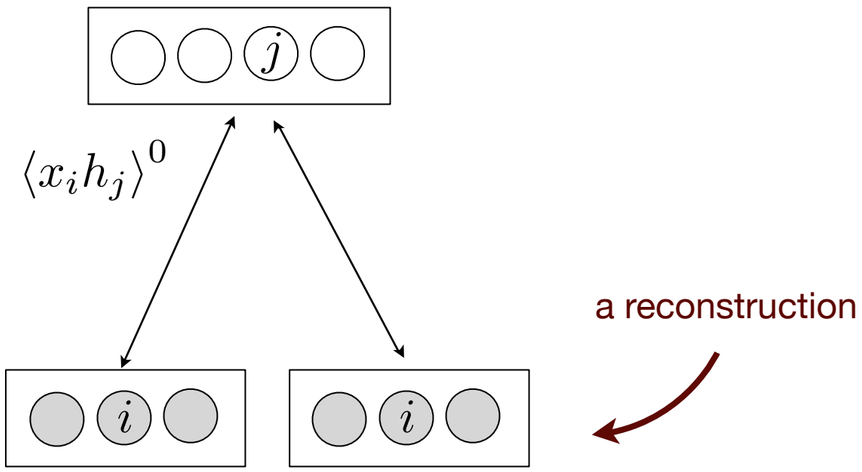
Contrastive Divergence



Instead of running the Markov chain to equilibrium, run for just one (or a few) steps!

Credit: Geoff Hinton

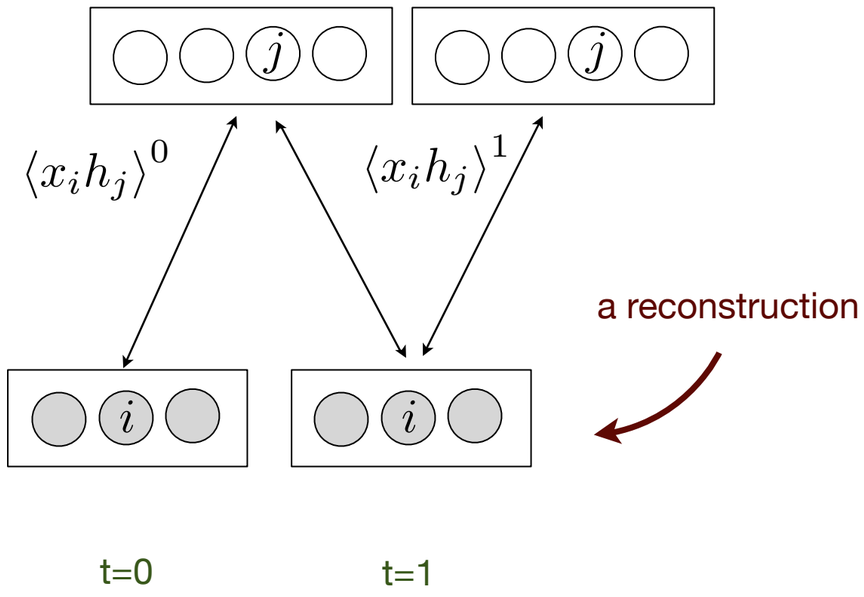
Contrastive Divergence



Instead of running the Markov chain to equilibrium, run for just one (or a few) steps!

Credit: Geoff Hinton

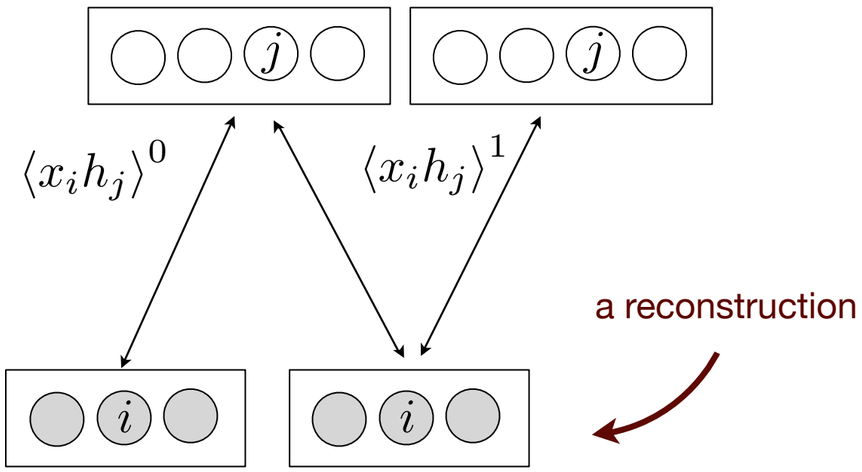
Contrastive Divergence



Instead of running the Markov chain to equilibrium, run for just one (or a few) steps!

Credit: Geoff Hinton

Contrastive Divergence

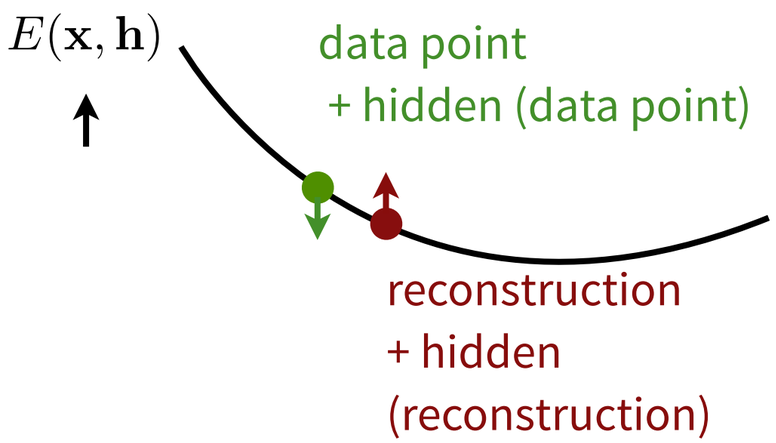


Instead of running the Markov chain to equilibrium, run for just one (or a few) steps!

$$\Delta w_{ij} = \epsilon \left(\langle x_i h_j \rangle^0 - \langle x_i h_j \rangle^1 \right)$$

Credit: Geoff Hinton

Contrastive Divergence (A picture)

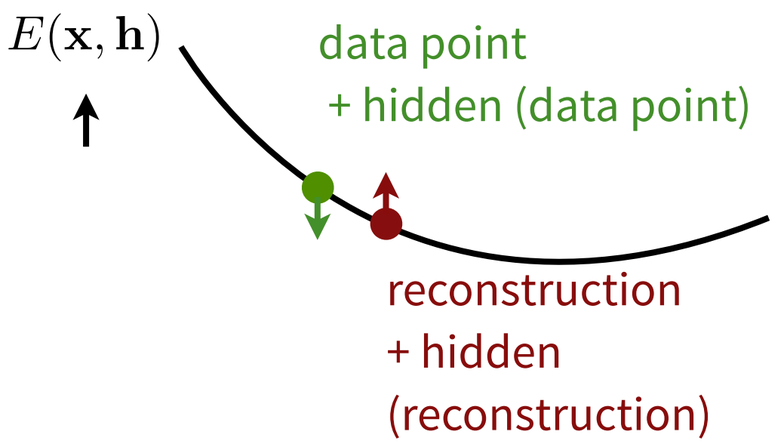


Change the weights to pull the energy down at the data point

Change the weights to pull the energy up at the reconstruction

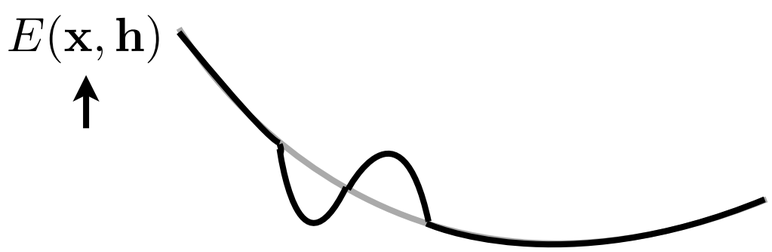
Credit: Geoff Hinton

Contrastive Divergence (A picture)



Change the weights to pull the energy down at the data point

Change the weights to pull the energy up at the reconstruction



Credit: Geoff Hinton

Alternatives to CD

- Persistent CD a.k.a. Stochastic Maximum Likelihood (Tieleman 2008)
  - don’t reset the Markov chain at the data for every point
- Score Matching/Ratio Matching (Hyvarinen 2005, 2007)
  - minimize the expected distance b/w model and data “score function”
- Minimum Probability Flow (Sohl-Dickstein et al. 2011)
  - establish dynamics that would transform the observed data distribution into the model distribution
  - minimize the KL divergence b/w the data distribution and the distribution produced by running the dynamics for an infinitesimal time

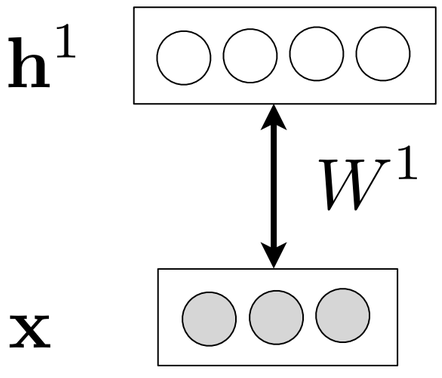
For a comparison, see *Inductive Principles for Restricted Boltzmann Machine Learning,* Marlin et al. 2010

Stacking to Build Deep Models

- Greedy layer-wise training can be used to build deep models
- It is most popular to use RBMs, but other architectures (regularized autoencoders, ICA, even k-means) can be stacked

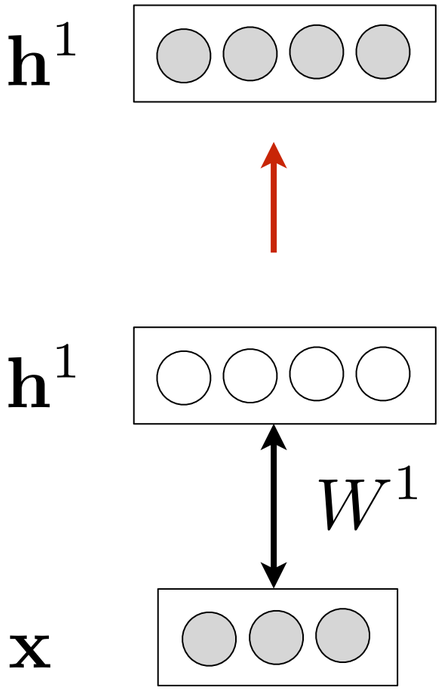
Stacking RBMs: Procedure

Stacking RBMs: Procedure



① Train an RBM

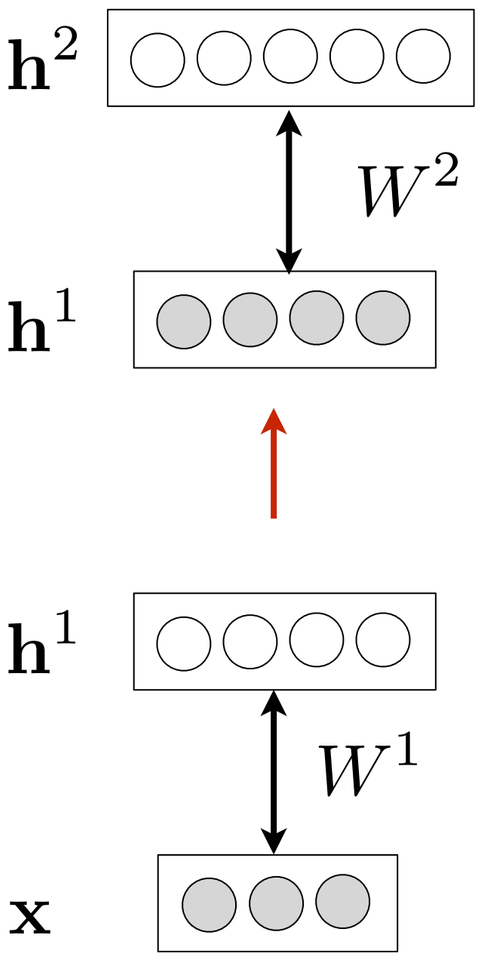
Stacking RBMs: Procedure



② Run your data through the model to generate a dataset of hidden activations

① Train an RBM

Stacking RBMs: Procedure

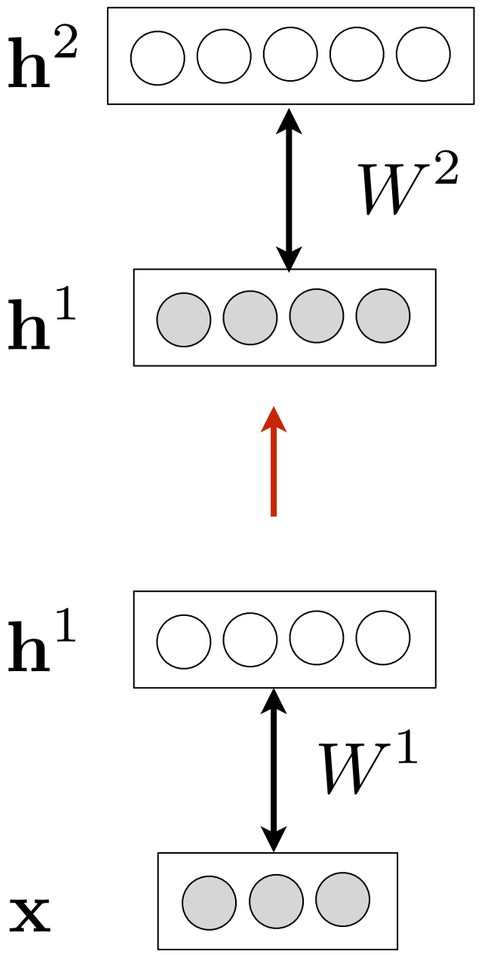


③ Treat the hiddens like data, train another RBM

② Run your data through the model to generate a dataset of hidden activations

① Train an RBM

Stacking RBMs: Procedure

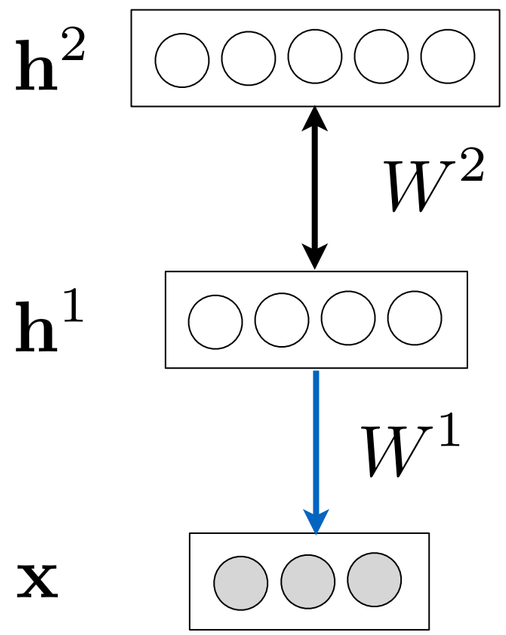


③ Treat the hiddens like data, train another RBM

② Run your data through the model to generate a dataset of hidden activations

① Train an RBM

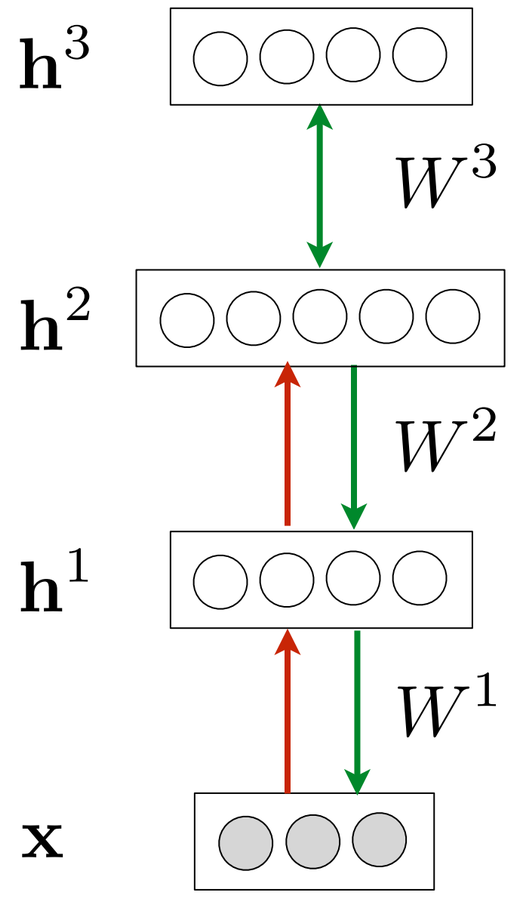
④ Compose the two models



(Hinton et al. 2006)

Deep Belief Networks

- The resulting model is called a Deep Belief Network
- Generate by alternating Gibbs sampling between the top two layers followed by a down-pass
- The lower level bottom-up connections are not part of the generative model, they are used only for inference



Stacking RBMs: Intuition

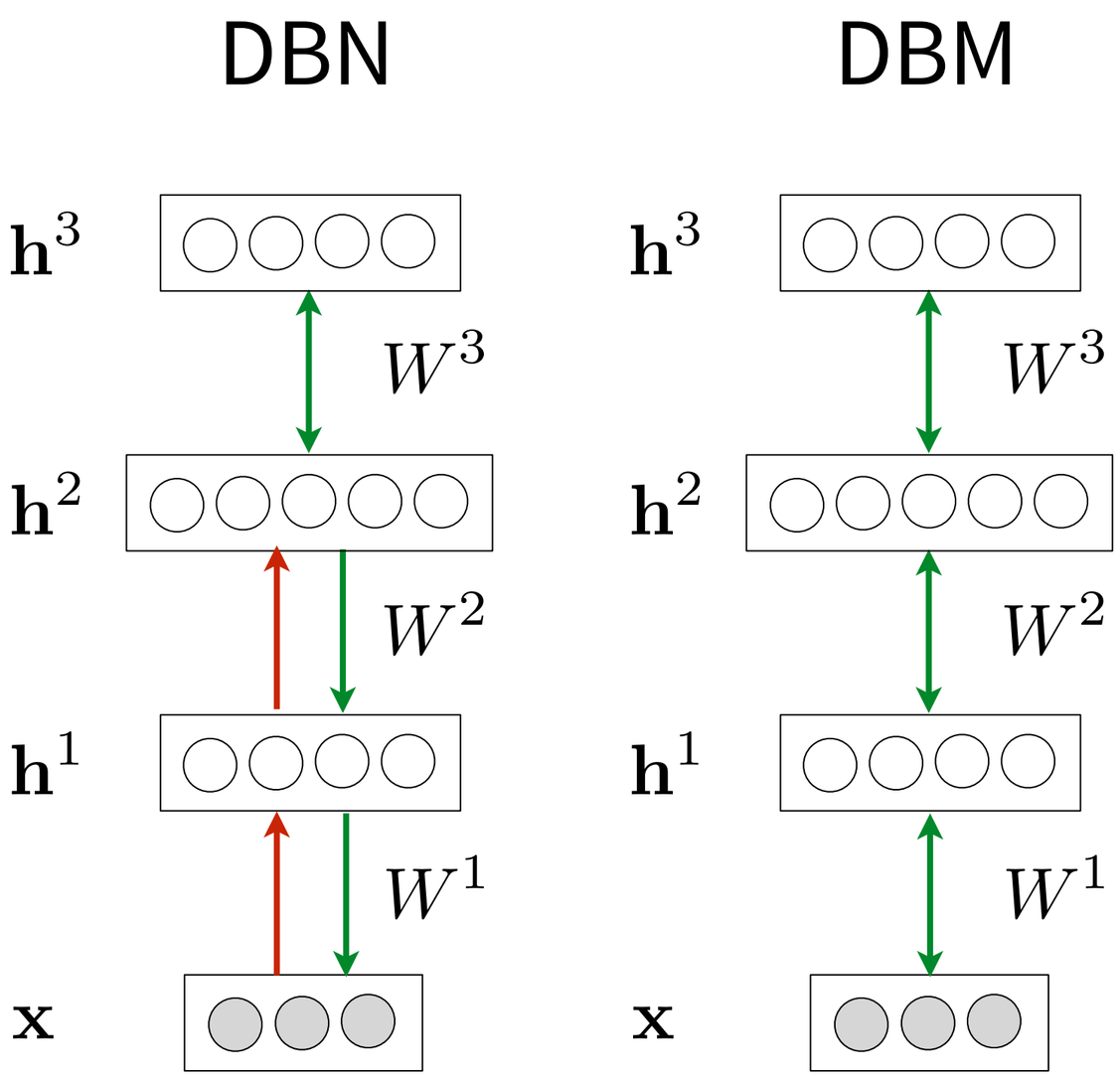
- The weights in the bottom-most RBM define many different distributions: $p(\mathbf{x}, \mathbf{h}), p(\mathbf{x}|\mathbf{h}), p(\mathbf{h}|\mathbf{x}), p(\mathbf{x}), p(\mathbf{h})$
- We can express the RBM as: $p(\mathbf{x}) = \sum_h p(\mathbf{h}) p(\mathbf{x}|\mathbf{h})$
- If we leave $p(\mathbf{x}|\mathbf{h})$ as-is and improve $p(\mathbf{h})$, we improve $p(\mathbf{x})$
- To improve $p(\mathbf{h})$ we need it to be better than $p(\mathbf{h}; W^1)$ at modeling the aggregated posterior over hidden vectors produced by applying the RBM to the data

Credit: Geoff Hinton

(Salakhutdinov and Hinton 2009)

Deep Boltzmann Machines

- DBN is a hybrid directed graphical model
  - maintains a set of “feed-forward” connections for inference
- DBN is an undirected graphical model
  - feedback is important
- Both take different approaches to dealing with intractable p(h|x)



Training DBMs

Training DBMs

- Standard DBM training procedure:

Training DBMs

- Standard DBM training procedure:
  - Greedy-wise pre-training of RBMs

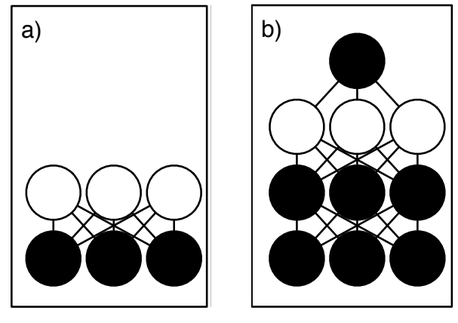


Image: (Goodfellow et al. 2013)

Training DBMs

- Standard DBM training procedure:
  - Greedy-wise pre-training of RBMs
  - Stitch the RBMs into a DBM and train with variational approximation to log-likelihood

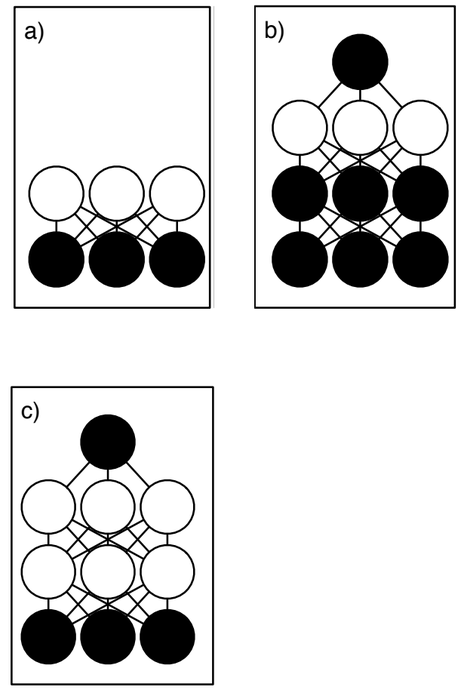


Image: (Goodfellow et al. 2013)

Training DBMs

- Standard DBM training procedure:
  - Greedy-wise pre-training of RBMs
  - Stitch the RBMs into a DBM and train with variational approximation to log-likelihood
  - Discriminative fine-tuning (DBM used as feature learner)

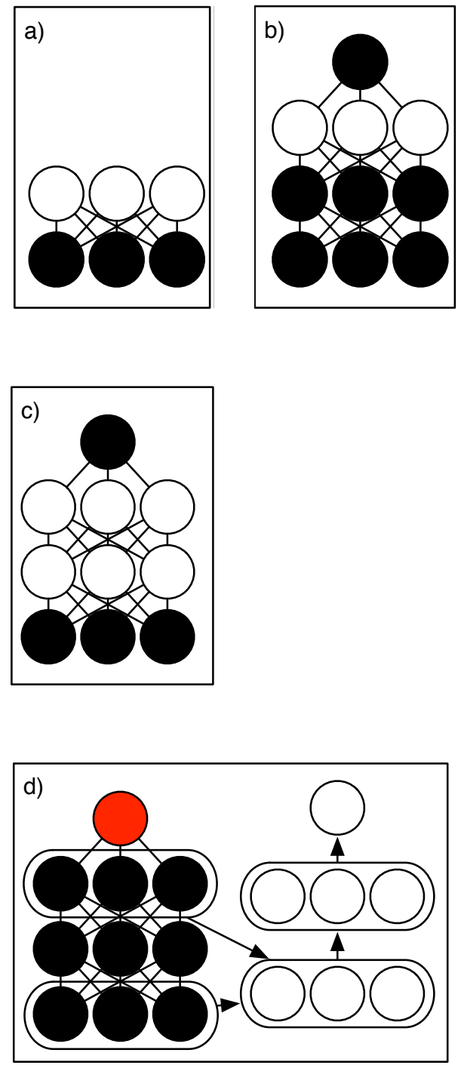


Image: (Goodfellow et al. 2013)

(Goodfellow et al. 2013)

Multi-prediction DBMs

(Goodfellow et al. 2013)

Multi-prediction DBMs

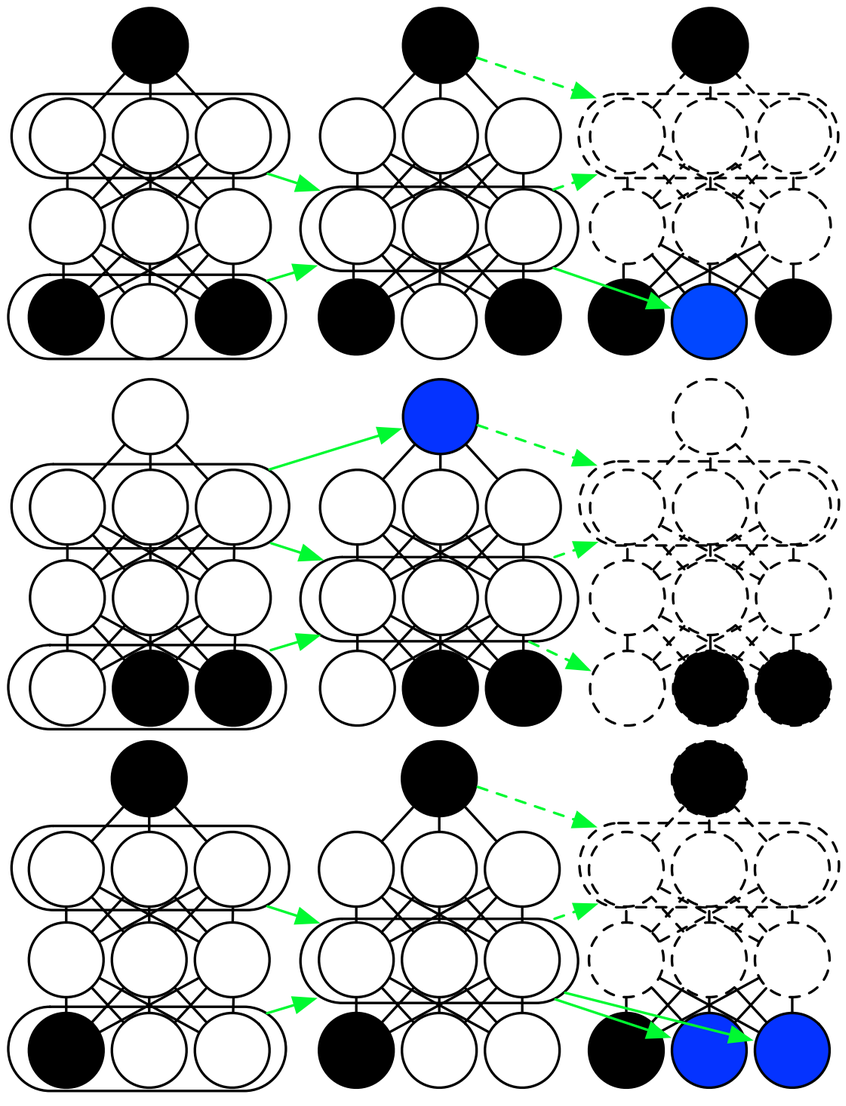
- Greedy pre-training is suboptimal
  - training procedure for each layer should account for the influence of deeper layers
  - one model for all tasks can use inference for arbitrary queries
  - needing to implement multiple models and stages makes DBMs cumbersome

(Goodfellow et al. 2013)

Multi-prediction DBMs

- Greedy pre-training is suboptimal
  - training procedure for each layer should account for the influence of deeper layers
  - one model for all tasks can use inference for arbitrary queries
  - needing to implement multiple models and stages makes DBMs cumbersome
- Joint “multi-prediction” training (Goodfellow et al. 2013)
  - Train DBM to predict any subset of vars given the complement of that subset

Multi-prediction training for classification



Black - variables net is allowed to observe
Blue - prediction targets

Conclusions and Challenges

- Most of the vision community's attention has gone towards supervised deep learning, however unsupervised learning will be key to future success
- Single-layer unsupervised learners well developed but joint unsupervised training of deep models remains difficult
- Can we train deep structured output models?

Resources

- Online courses
  - Andrew Ng's Machine Learning (Coursera)
  - Geoff Hinton's Neural Networks (Coursera)
- Websites
  - deeplearning.net
  - http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial

Surveys and Reviews

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35(8):1798–1828, Aug 2013.

Y. Bengio. Deep learning of representations: Looking forward. In Statistical Language and Speech Processing, pages 1–37. Springer, 2013.

Y. Bengio, I. Goodfellow, and A. Courville. Deep Learning. 2014. Draft available at http://www.iro.umontreal.ca/~bengioy/dlbook/

J. Schmidhuber. Deep learning in neural networks: An overview. arXiv preprint arXiv:1404.7828, 2014.

Y. Bengio. Learning deep architectures for ai. Foundations and trends in Machine Learning, 2(1):1–127, 2009.

Papers in this Tutorial

D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? The Journal of Machine Learning Research, 11:625–660, 2010.

K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. arXiv preprint arXiv:1010.3467, 2010.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM, 2008.

S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto- encoders: Explicit invariance during feature extraction. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pages 833–840, 2011.

G. Alain and Y. Bengio. What regularized auto-encoders learn from the data generating distribution. arXiv preprint arXiv:1211.4246, 2012.

Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto- encoders as generative models. In Advances in Neural Information Processing Systems, pages 899–907, 2013.

1H. Kamyshanska and R. Memisevic. On autoencoder scoring. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pages 720–728, 2013.

B. M. Marlin, K. Swersky, B. Chen, and N. D. Freitas. Inductive principles for restricted boltzmann machine learning. In International Conference on Artificial Intelligence and Statistics, pages 509–516, 2010.

G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527–1554, 2006.

R. Salakhutdinov and G. E. Hinton. Deep boltzmann machines. In Inter- national Conference on Artificial Intelligence and Statistics, pages 448–455, 2009.

Recent Work

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. arXiv preprint arXiv:1311.2901, 2013.

Y. Bengio and E. Thibodeau-Laufer. Deep generative stochastic networks trainable by backprop. arXiv preprint arXiv:1306.1091, 2013.

I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio. Multi-prediction deep boltzmann machines. In Advances in Neural Information Processing Systems, pages 548–556, 2013.

Y. He, K. Kavukcuoglu, Y. Wang, A. Szlam, and Y. Qi. Unsupervised feature learning by deep sparse coding. In ICLR, 2014.

Practical Tips

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In Neural Networks: Tricks of the Trade, pages 437–478. Springer, 2012.

G. E. Hinton. A practical guide to training restricted boltzmann machines. In Neural Networks: Tricks of the Trade, pages 599–619. Springer, 2012.