Neural Networks

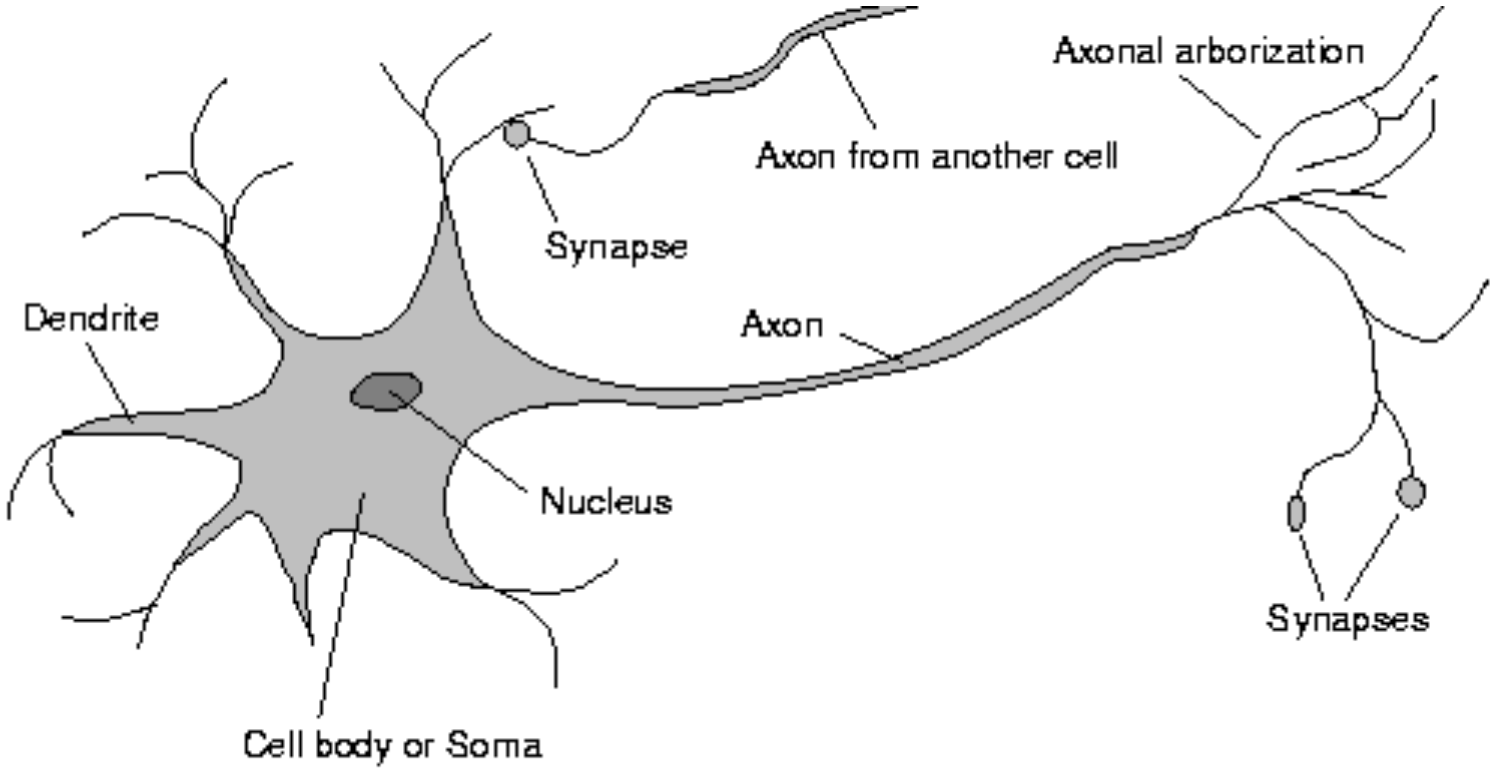
Adapted from slides by
Tim Finin and
Marie desJardins.

Some material adapted
from lecture notes by
Lise Getoor and Ron Parr

Neural function

- Brain function (thought) occurs as the result of the firing of **neurons**
- Neurons connect to each other through **synapses**, which propagate **action potential** (electrical impulses) by releasing **neurotransmitters**
- Synapses can be **excitatory** (potential-increasing) or **inhibitory** (potential-decreasing), and have varying **activation thresholds**
- Learning occurs as a result of the synapses' **plasticicity**: They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain!

Biology of a neuron



Brain structure

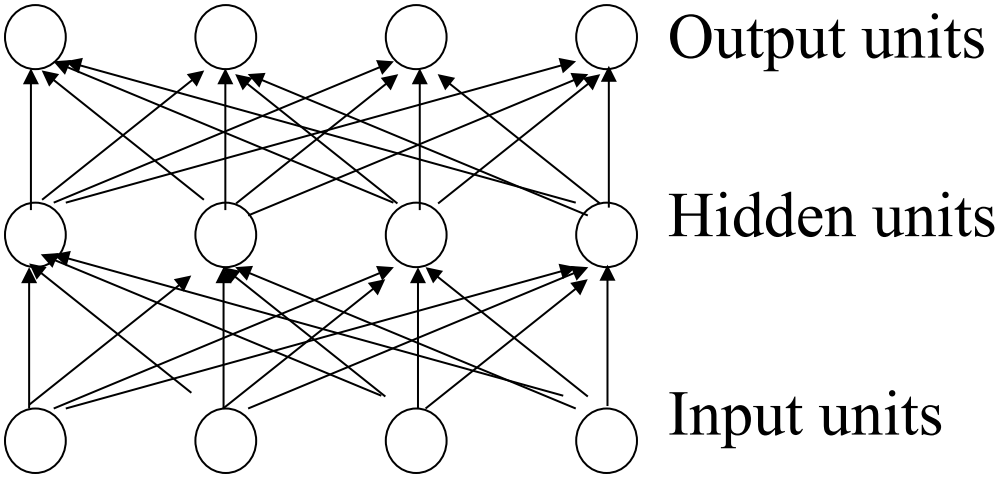
- Different areas of the brain have different functions
  - Some areas seem to have the same function in all humans (e.g., Broca's region for motor speech); the overall layout is generally consistent
  - Some areas are more plastic, and vary in their function; also, the lower-level structure and function vary greatly
- We don't know how different functions are "assigned" or acquired
  - Partly the result of the physical layout / connection to inputs (sensors) and outputs (effectors)
  - Partly the result of experience (learning)
- We *really* don't understand how this neural structure leads to what we perceive as "consciousness" or "thought"
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

Comparison of computing power

INFORMATION CIRCA 2012	Computer	Human Brain
Computation Units	10-core Xeon: 10^9 Gates	10^{11} Neurons
Storage Units	10^9 bits RAM, 10^{12} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-9} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec

- Computers are way faster than neurons…
- But there are a lot more neurons than we can reasonably model in modern digital computers, and they all fire in parallel
- Neural networks are designed to be massively parallel
- The brain is effectively a billion times faster

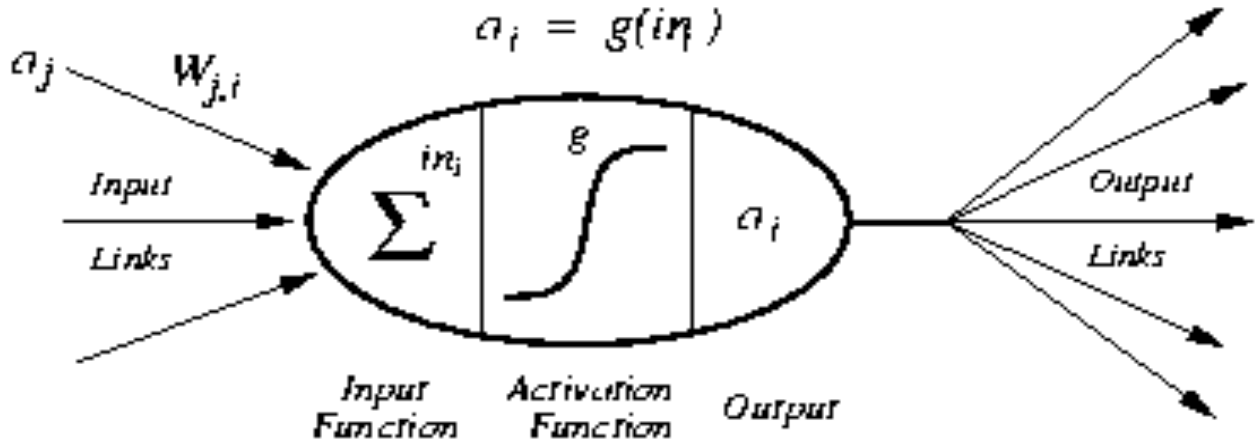
Neural networks



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Model of a neuron



- Neuron modeled as a unit i
- weights on input unit j to i, w_{ji}
- net input to unit i is:

$$in_i = \sum_j w_{ji} \cdot o_j$$

- Activation function g() determines the neuron's output
  - g() is typically a sigmoid
  - output is either 0 or 1 (no partial activation)

"Executing" neural networks

- Input units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Learning rules

- Rosenblatt (1959) suggested that if a target output value is provided for a single neuron with fixed inputs, can incrementally change weights to learn to produce these outputs using the *perceptron learning rule*
  - assumes binary valued input/outputs
  - assumes a single linear threshold unit

Perceptron learning rule

- If the target output for unit i is t_i

$$w_{ji} = w_{ji} + \eta(t_i - o_i)o_j$$

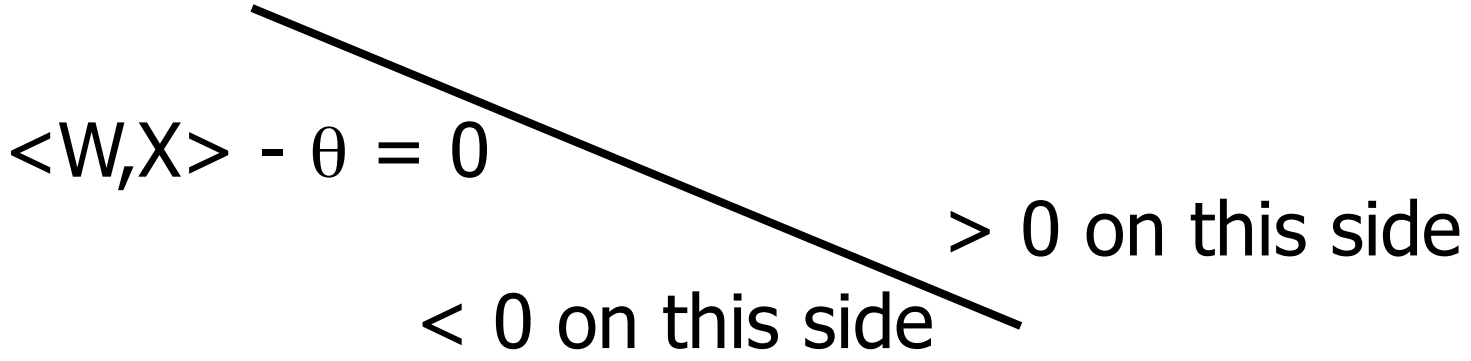
- Equivalent to the intuitive rules:
  - If output is correct, don't change the weights
  - If output is low (o_i=0, t_i=1), increment weights for all the inputs which are 1
  - If output is high (o_i=1, t_i=0), decrement weights for all inputs which are 1
  - Must also adjust threshold. Or equivalently assume there is a weight w_{0i} for an extra input unit that has an output of 1.

Perceptron learning algorithm

- Repeatedly iterate through examples adjusting weights according to the perceptron learning rule until all outputs are correct
  - Initialize the weights to all zero (or random)
  - Until outputs for all training examples are correct
    - for each training example e do
      - compute the current output o_j
      - compare it to the target t_j and update weights
- Each execution of outer loop is called an ***epoch***
- For multiple category problems, learn a separate perceptron for each category and assign to the class whose perceptron most exceeds its threshold

Representation limitations of a perceptron

- Perceptrons can only represent linear threshold functions and can therefore only learn functions which linearly separate the data.
  - i.e., the positive and negative examples are separable by a hyperplane in n-dimensional space



Perceptron learnability

- **Perceptron Convergence Theorem**: If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge (Minicksy & Papert, 1969)
- Unfortunately, many functions (like parity) cannot be represented by LTU

Learning: Backpropagation

- Similar to perceptron learning algorithm, we cycle through our examples
  - if the output of the network is correct, no changes are made
  - if there is an error, the weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

Output layer

- As in perceptron learning algorithm, we want to minimize difference between target output and the output actually computed

$$W_{ji} = W_{ji} + \alpha \times a_j \times Err_i \times g'(in_i)$$

activation of hidden unit j

$(T_i - O_i)$

derivative of activation function

$$\Delta_i = Err_i \times g'(in_i)$$

$$W_{ji} = W_{ji} + \alpha \times a_j \times \Delta_i$$

Hidden layers

- Need to define error; we do error backpropagation.
- Intuition: Each hidden node j is "responsible" for some fraction of the error Δ_l in each of the output nodes to which it connects.
- Δ_l divided according to the strength of the connection between hidden node and the output node and propagated back to provide the Δ_j values for the hidde

$$\Delta_j = g'(in_j)\sum_j W_{ji}\Delta_i$$

update rule: $W_{kj} = W_{kj} + \alpha \times I_k \times \Delta_j$

Backprogation algorithm

- Compute the Δ values for the output units using the observed error
- Starting with output layer, repeat the following for each layer in the network, until earliest hidden layer is reached:
  - propagate the Δ values back to the previous layer
  - update the weights between the two layers

Backprop issues

- "Backprop is the cockroach of machine learning. It's ugly, and annoying, but you just can't get rid of it."

  -Geoff Hinton

- Problems:
  - black box
  - local minima

Restricted Boltzmann Machines and Deep Belief Networks

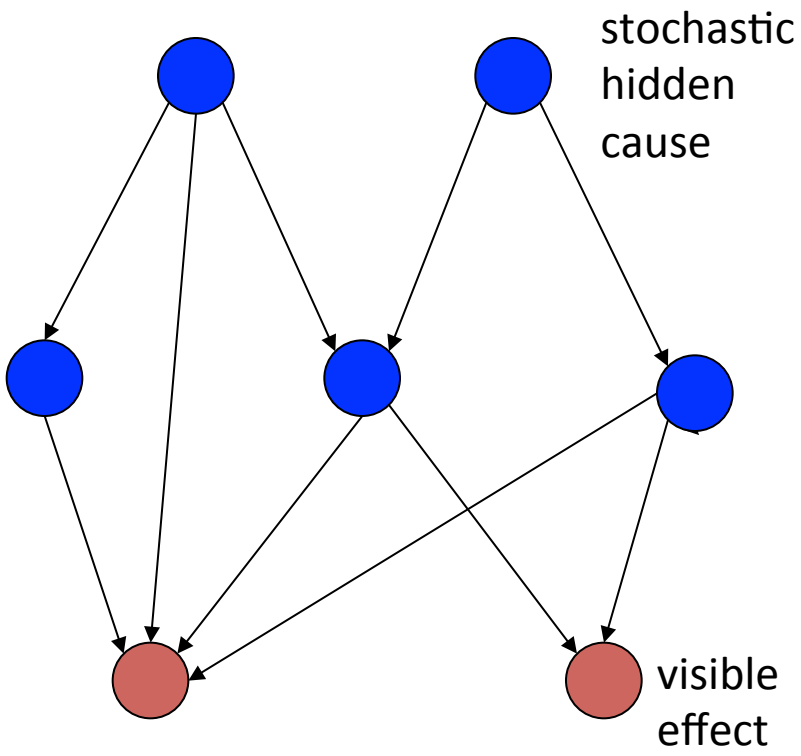
Slides from: Geoffrey Hinton, Sue Becker, Yann Le Cun, Yoshua Bengio, Frank Wood

Motivations

- Supervised training of deep models (e.g. many-layered NNets) is difficult (optimization problem)
- Shallow models (SVMs, one-hidden-layer NNets, boosting, etc…) are unlikely candidates for learning high-level abstractions needed for AI
- Unsupervised learning could do "local-learning" (each module tries its best to model what it sees)
- Inference (+ learning) is intractable in directed graphical models with many hidden variables
- Current unsupervised learning methods don't easily extend to learn multiple levels of representation

Belief Nets

- A belief net is a directed acyclic graph composed of stochastic variables.
- Can observe some of the variables and we would like to solve two problems:
- The inference problem: Infer the states of the unobserved variables.
- The learning problem: Adjust the interactions between variables to make the network more likely to generate the observed data.

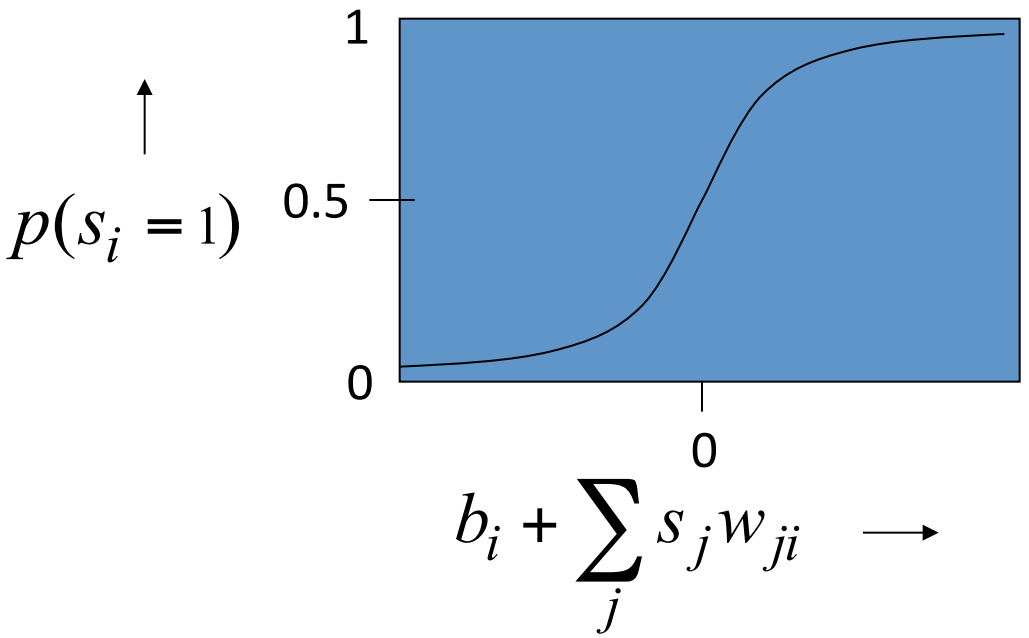


Use nets composed of layers of stochastic binary variables with weighted connections. Later, we will generalize to other types of variable.

Stochastic binary neurons

- These have a state of 1 or 0 which is a stochastic function of the neuron's bias, b, and the input it receives from other neurons.

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})}$$



Stochastic units

- Replace the binary threshold units by binary stochastic units that make biased random decisions.
  - The temperature controls the amount of noise.
  - Decreasing all the energy gaps between configurations is equivalent to raising the noise level.

$$p(s_i{=}1) \quad = \quad \frac{1}{1+e^{-\sum_j s_j w_{ij}/T}} \quad = \quad \frac{1}{1+e^{-\Delta E_i/T}}$$

temperature

$$\textit{Energy gap} \;=\; \Delta E_i \;=\; E(s_i{=}0) - E(s_i{=}1)$$

The Energy of a joint configuration

binary state of unit i in joint configuration v, h

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i \in units} s_i^{\mathbf{vh}} b_i - \sum_{i<j} s_i^{\mathbf{vh}} s_j^{\mathbf{vh}} w_{ij}$$

Energy with configuration v on the visible units and h on the hidden units

bias of unit i

indexes every non-identical pair of i and j once

weight between units i and j

Weights → Energies → Probabilities

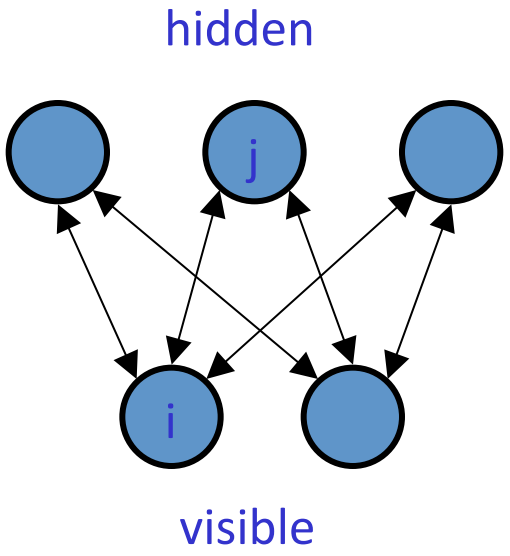
- Each possible joint configuration of the visible and hidden units has an energy
  - The energy is determined by the weights and biases (as in a Hopfield net).
- The energy of a joint configuration of the visible and hidden units determines its probability:

$$p(v,h) \propto e^{-E(v,h)}$$

- The probability of a configuration over the visible units is found by summing the probabilities of all the joint configurations that contain it.

Restricted Boltzmann Machines

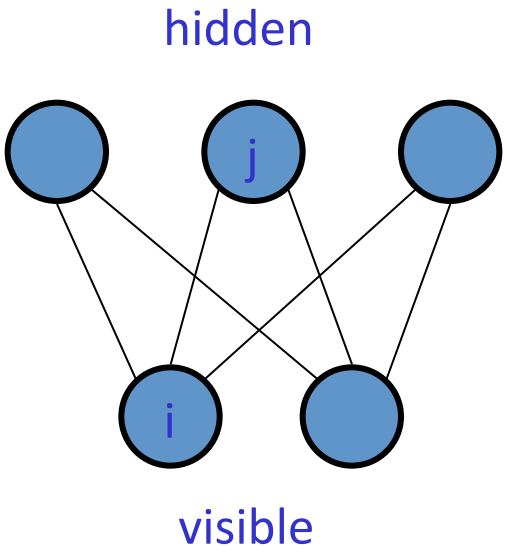
- Restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
    - Deal with more layers later
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states.
  - So can quickly get an unbiased sample from the posterior distribution when given a data-vector.
  - This is a big advantage over directed belief nets



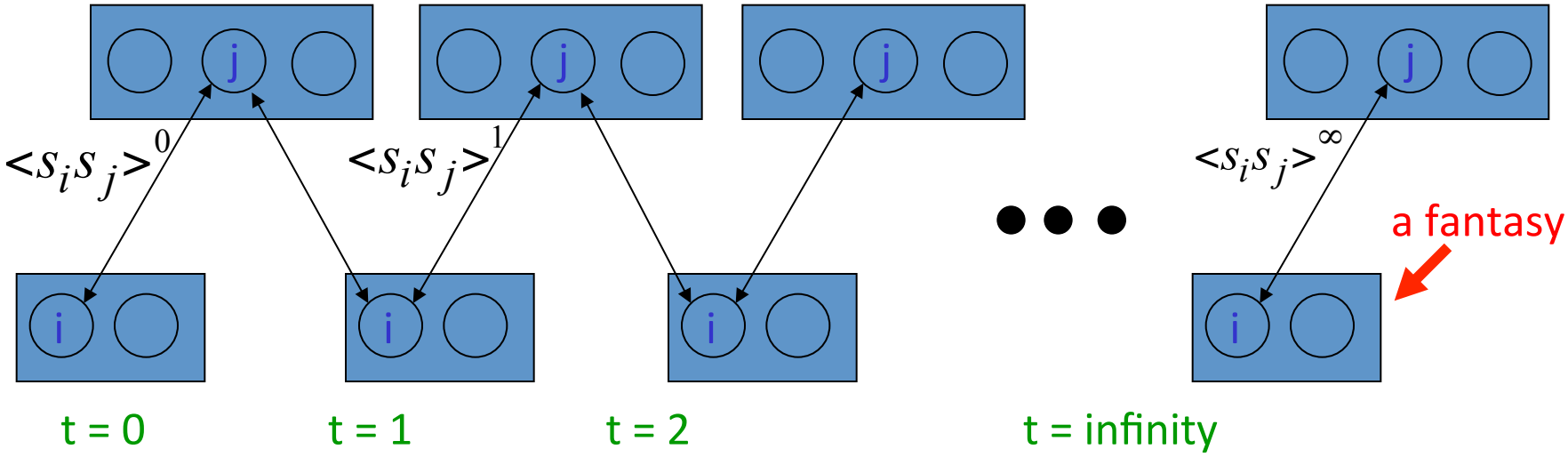
Restricted Boltzmann Machines

- In an RBM, the hidden units are conditionally independent given the visible states. It only takes one step to reach thermal equilibrium when the visible units are clamped.
  - Can quickly get the exact value of :

$$< s_i s_j >_{\mathbf{v}}$$



A picture of the Boltzmann machine learning algorithm for an RBM

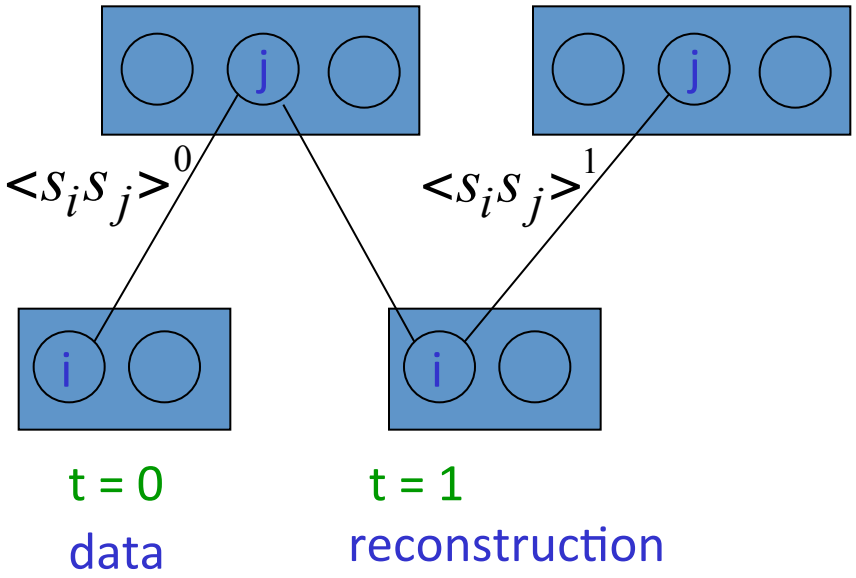


Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^\infty \right)$$

Contrastive divergence learning: A quick way to learn an RBM



Start with a training vector on the visible units.

Update all the hidden units in parallel

Update the all the visible units in parallel to get a "reconstruction".

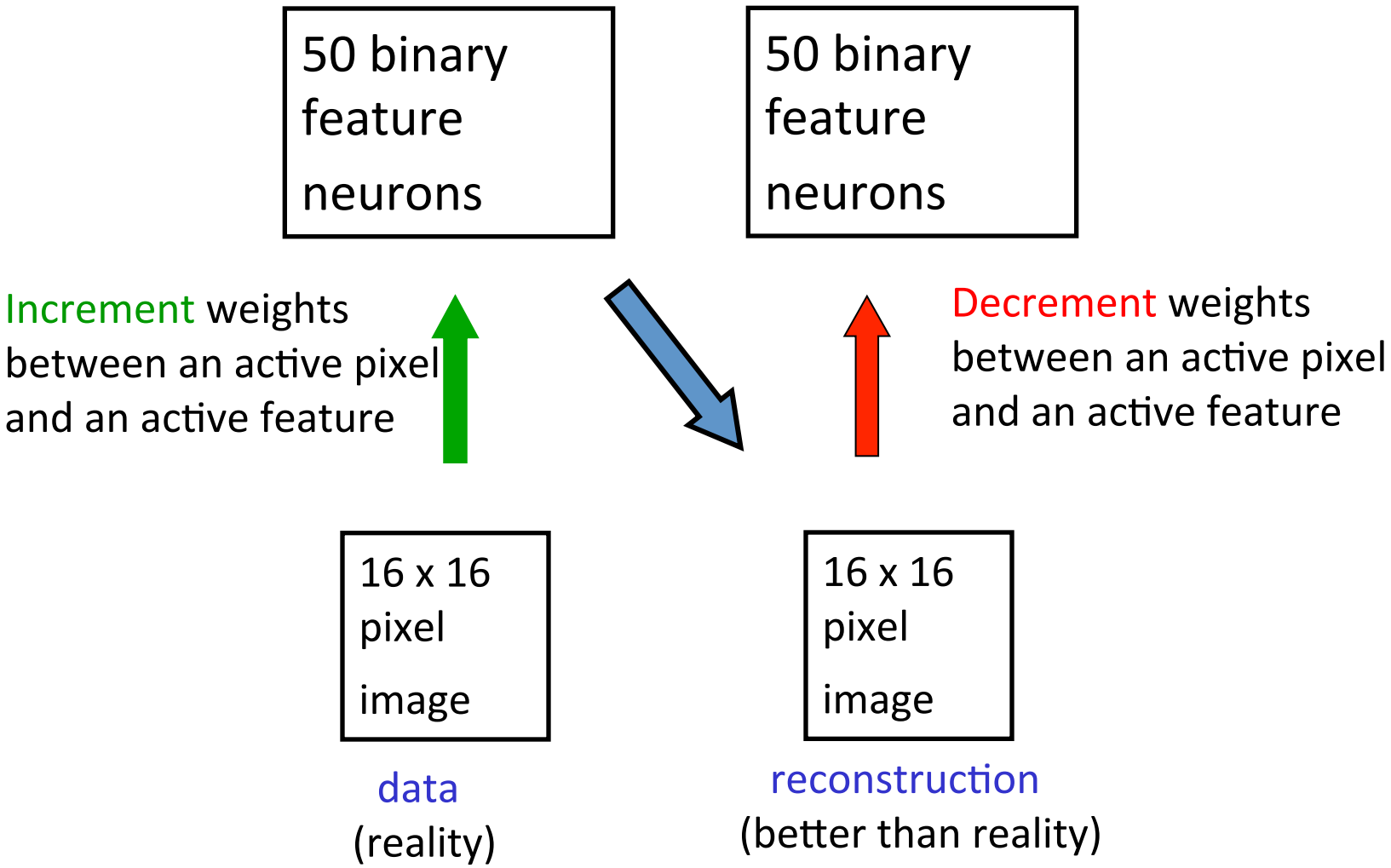
Update the hidden units again.

$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^1 \right)$$

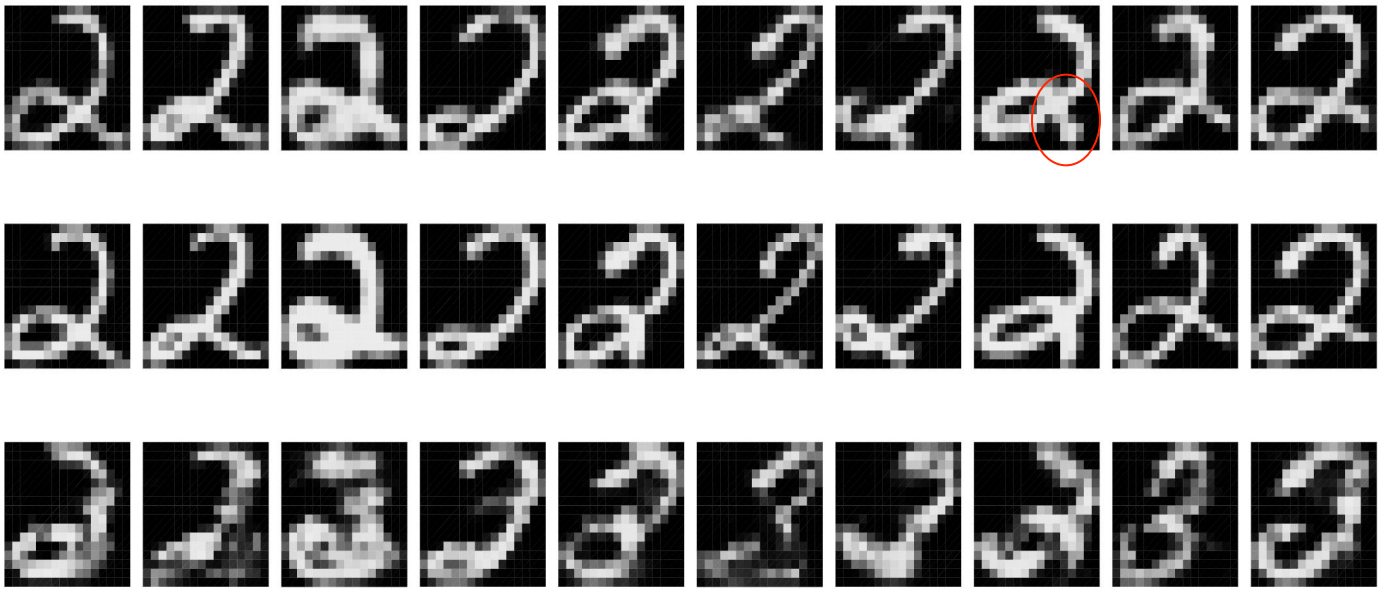
This is not following the gradient of the log likelihood. But it works well.

When we consider infinite directed nets it will be easy to see why it works.

How to learn a set of features that are good for reconstructing images of the digit 2



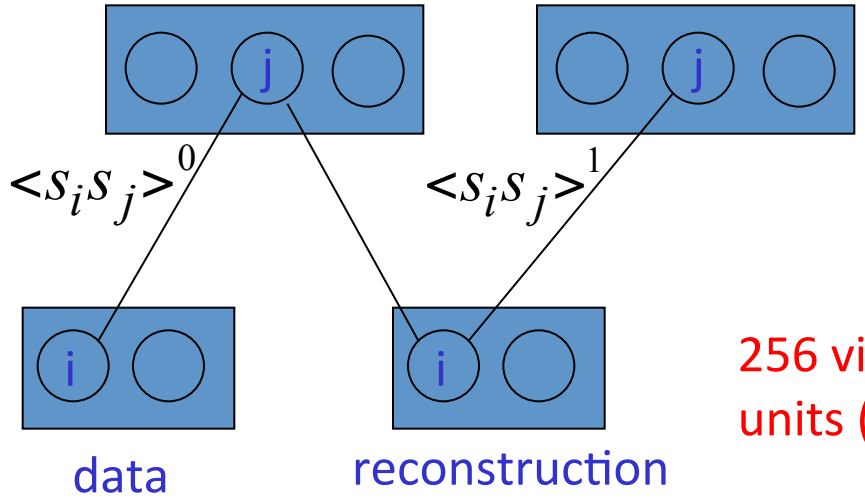
Using an RBM to learn a model of a digit class



Reconstructions by model trained on 2's

Data

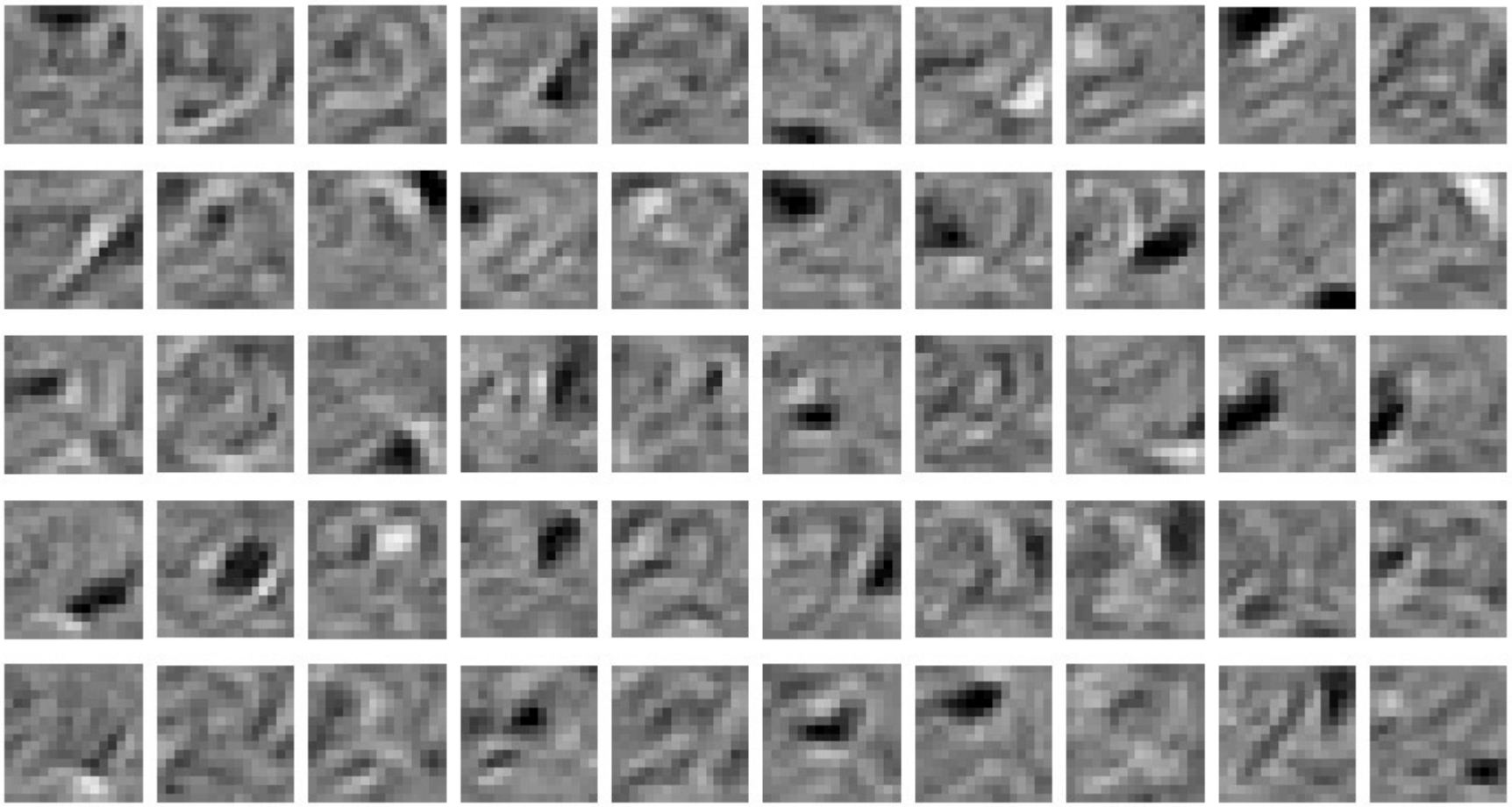
Reconstructions by model trained on 3's



100 hidden units (features)

256 visible units (pixels)

The final 50 X 256 weights

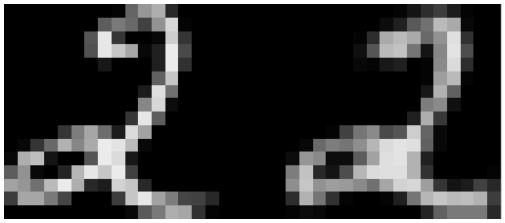


Each neuron grabs a different feature.

How well can we reconstruct the digit images from the binary feature activations?

Data

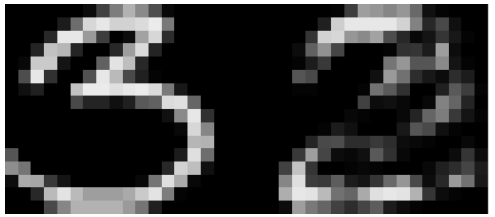
Reconstruction from activated binary features



New test images from the digit class that the model was trained on

Data

Reconstruction from activated binary features



Images from an unfamiliar digit class (the network tries to see every image as a 2)

Deep Belief Networks

Divide and conquer multilayer learning

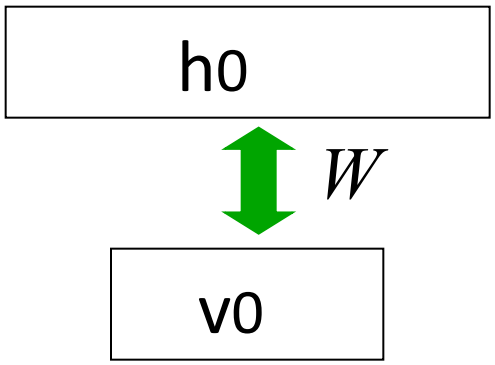
- Re-representing the data: Each time the base learner is called, it passes a transformed version of the data to the next learner.
  - Can we learn a deep, dense DAG one layer at a time, starting at the bottom, and still **guarantee that learning each layer improves the overall model of the training data**?
    - This seems very unlikely. Surely we need to know the weights in higher layers to learn lower layers?

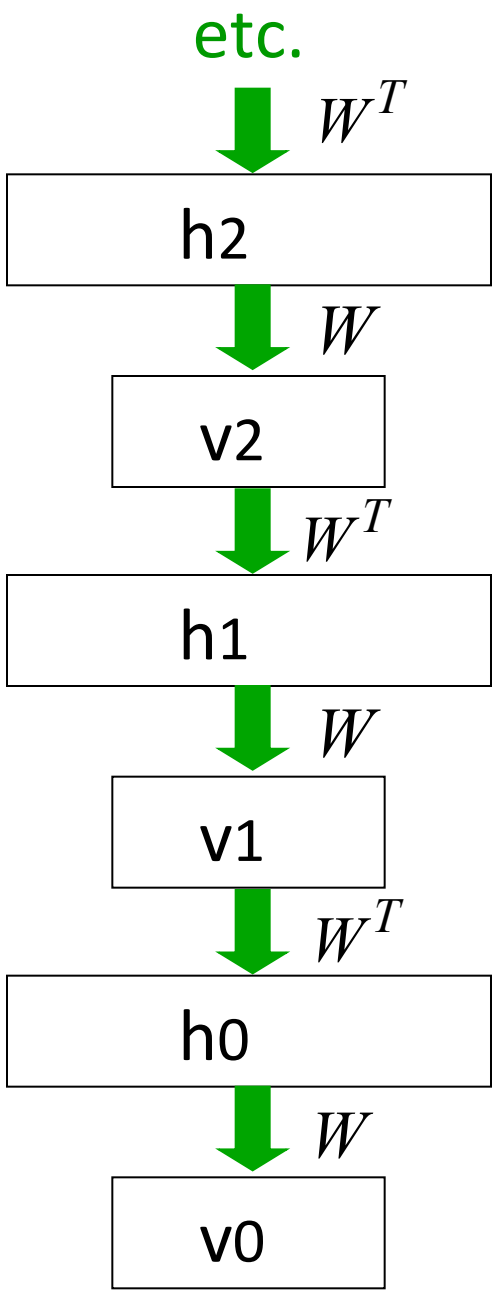
Multilayer contrastive divergence

- Start by learning one hidden layer.
- Then re-present the data as the activities of the hidden units.
  - The same learning algorithm can now be applied to the re-presented data.
- Can we prove that each step of this greedy learning improves the log probability of the data under the overall model?
  - What is the overall model?

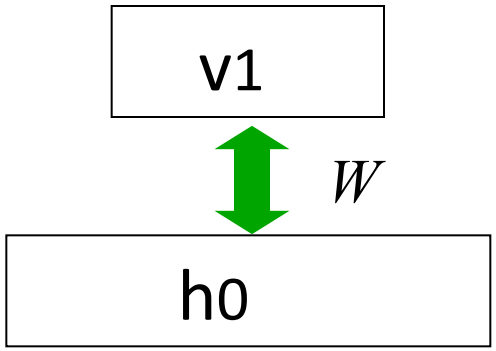
Learning a deep directed network

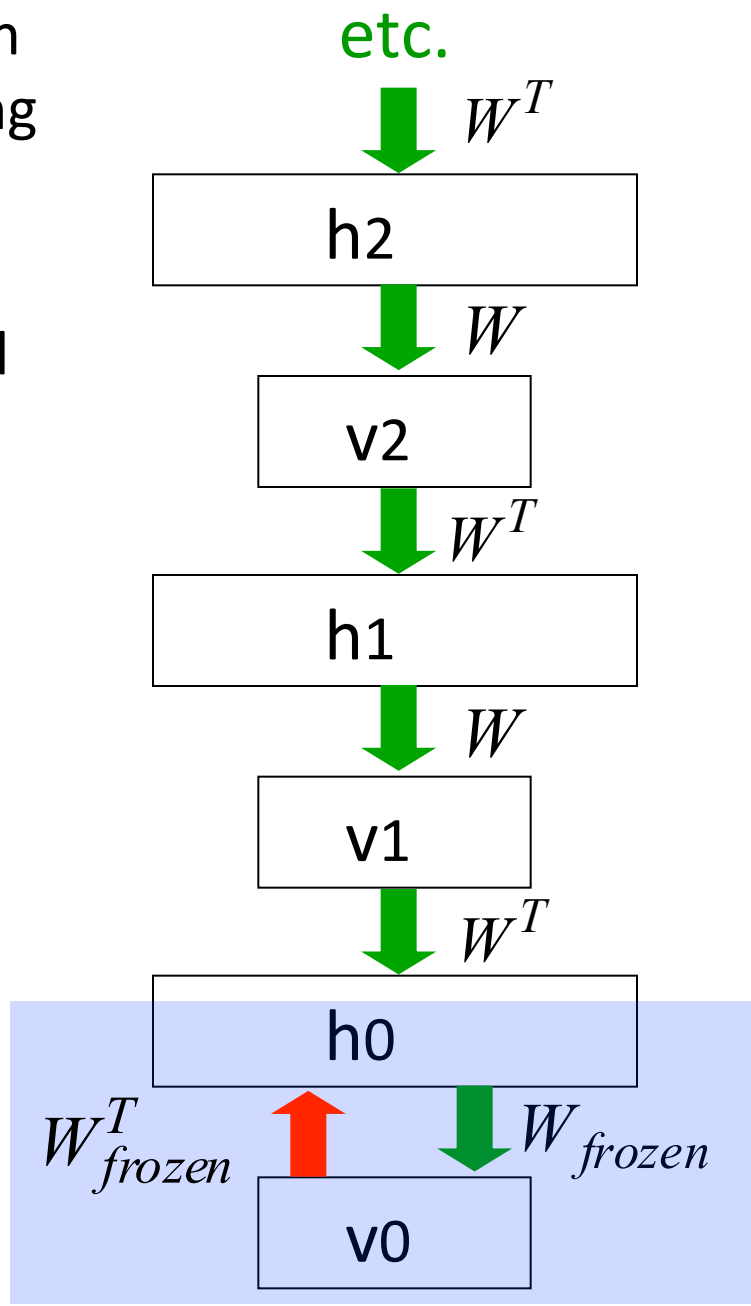
- First learn with all the weights tied
  - This is exactly equivalent to learning an RBM
  - Contrastive divergence learning is equivalent to ignoring the small derivatives contributed by the tied weights between deeper layers.





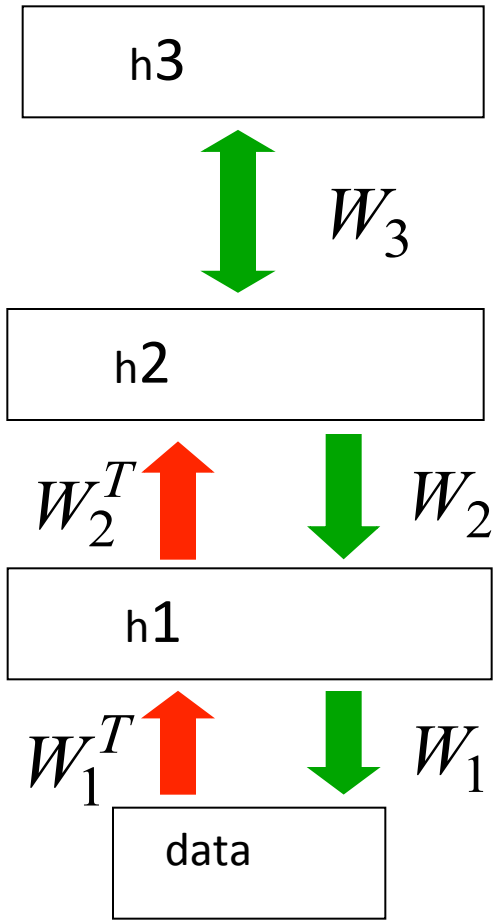
- Then freeze the first layer of weights in both directions and learn the remaining weights (still tied together).
  - This is equivalent to learning another RBM, using the aggregated posterior distribution of h0 as the data.





A simplified version with all hidden layers the same size

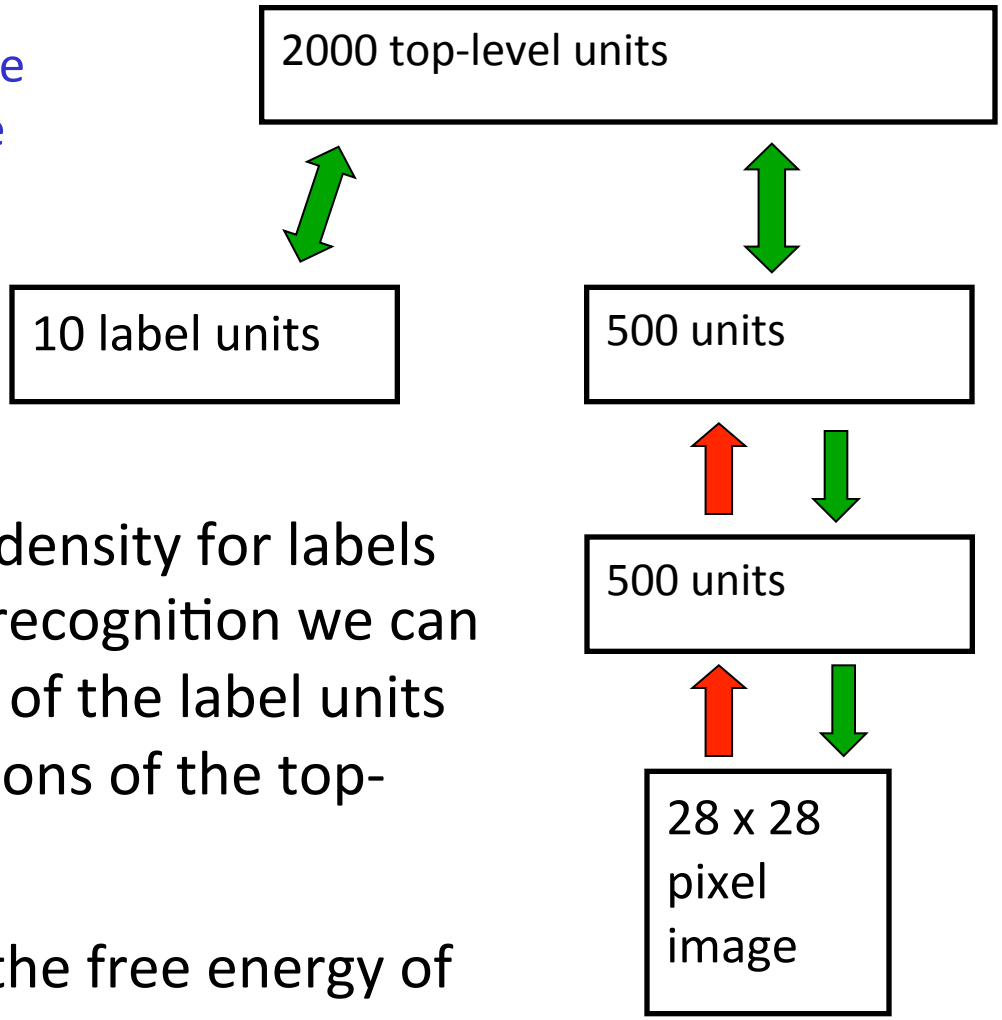
- The RBM at the top can be viewed as shorthand for an infinite directed net.
- When learning W_1 we can view the model in two quite different ways:
  - The model is an RBM composed of the data layer and h1.
  - The model is an infinite DAG with tied weights.
- After learning W_1 we untie it from the other weight matrices.
- We then learn W_2 which is still tied to all the matrices above it.



A neural network model of digit recognition

The top two layers form a restricted Boltzmann machine whose free energy landscape models the low dimensional manifolds of the digits.

The valleys have names:



The model learns a joint density for labels and images. To perform recognition we can start with a neutral state of the label units and do one or two iterations of the top-level RBM.

Or we can just compute the free energy of the RBM with each of the 10 labels

Show the movie of the network generating digits

(available at www.cs.toronto/~hinton)