The "Wake-Sleep" Algorithm for Unsupervised Neural Networks

Geoffrey E. Hinton,* Peter Dayan, Brendan J. Frey, Radford M. Neal

Sripriya Ravindra Kumar and Jonathan Kenny

Topics in Systems Neuroscience
California Institute of Technology
May 17th, 2017

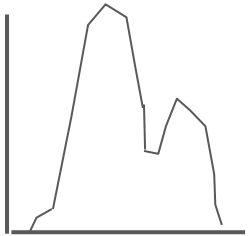
Generative models

Nguyen, Anh, et al. "Plug & play generative networks: Conditional iterative generation of images in latent space." *arXiv preprint arXiv:1612.00005* (2016).

Generative models

True data distribution

p(x)



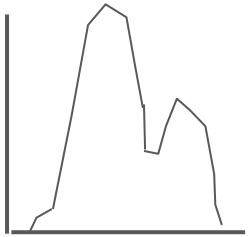
Real images

Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

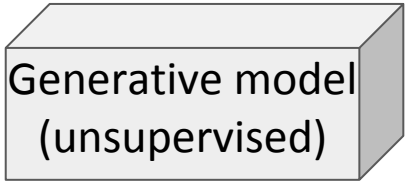
Generative models

True data distribution

p(x)



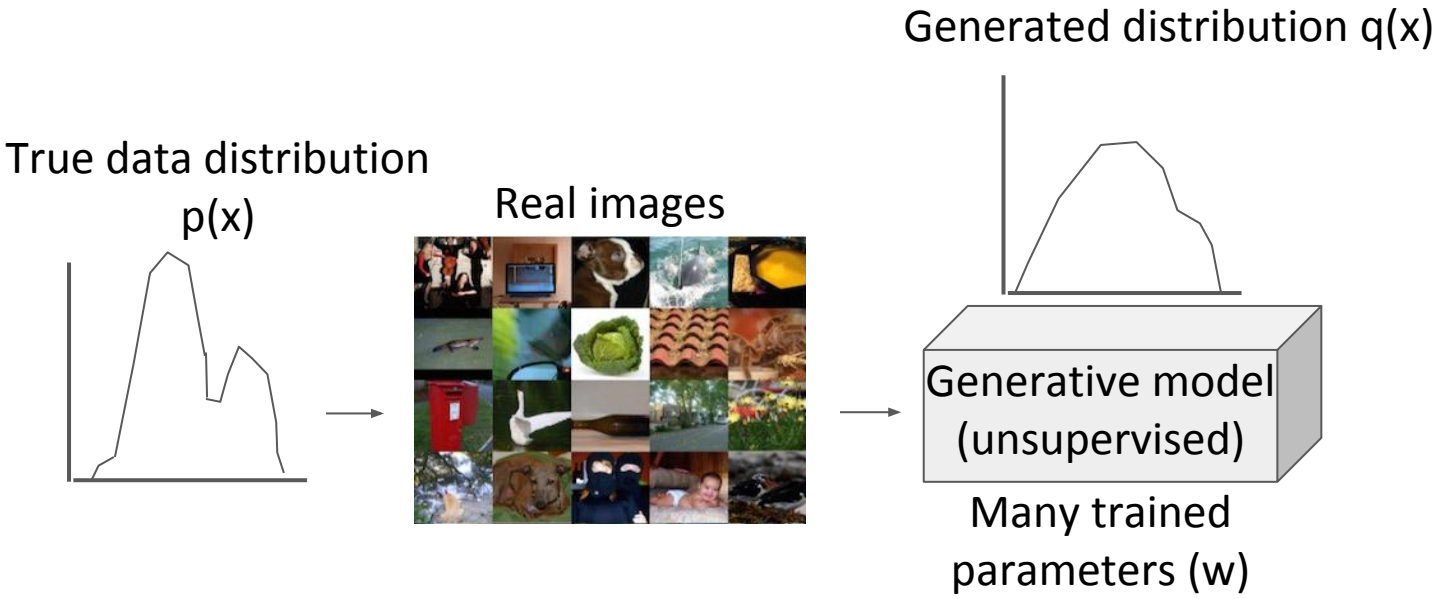
Real images



Many unknown parameters (w)

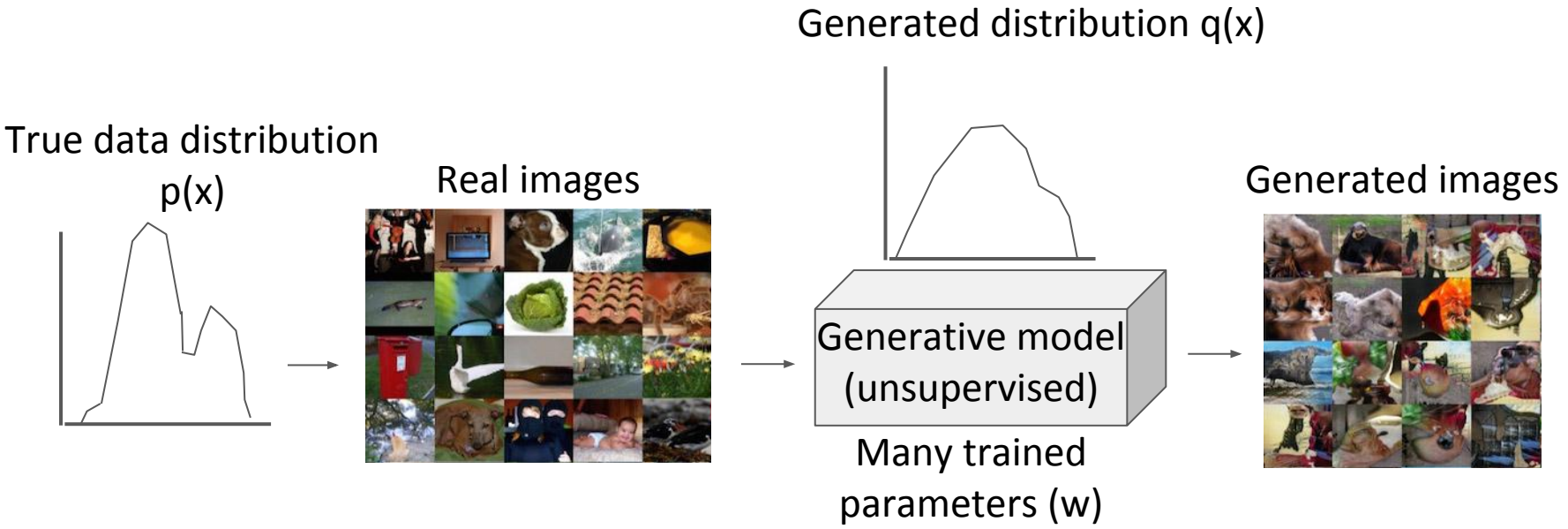
Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Generative models



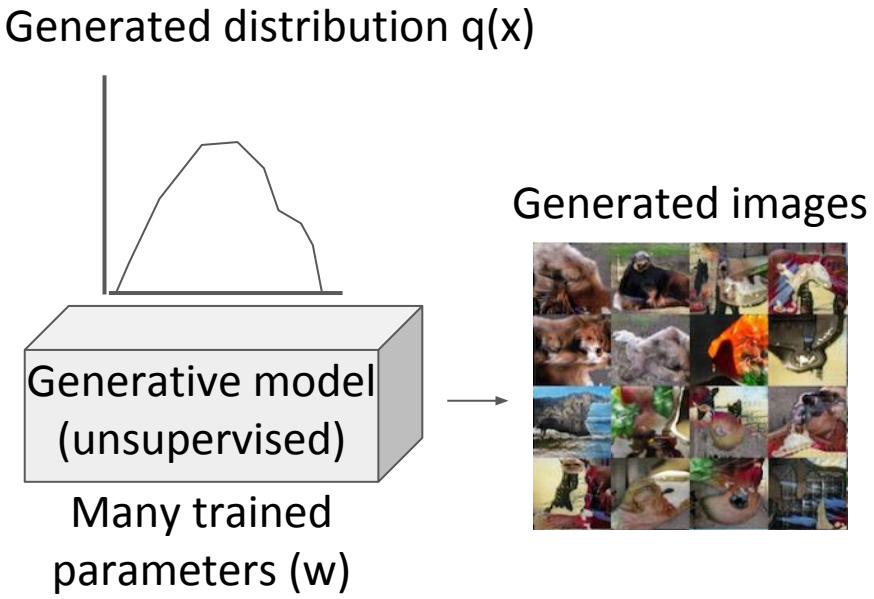
Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Generative models



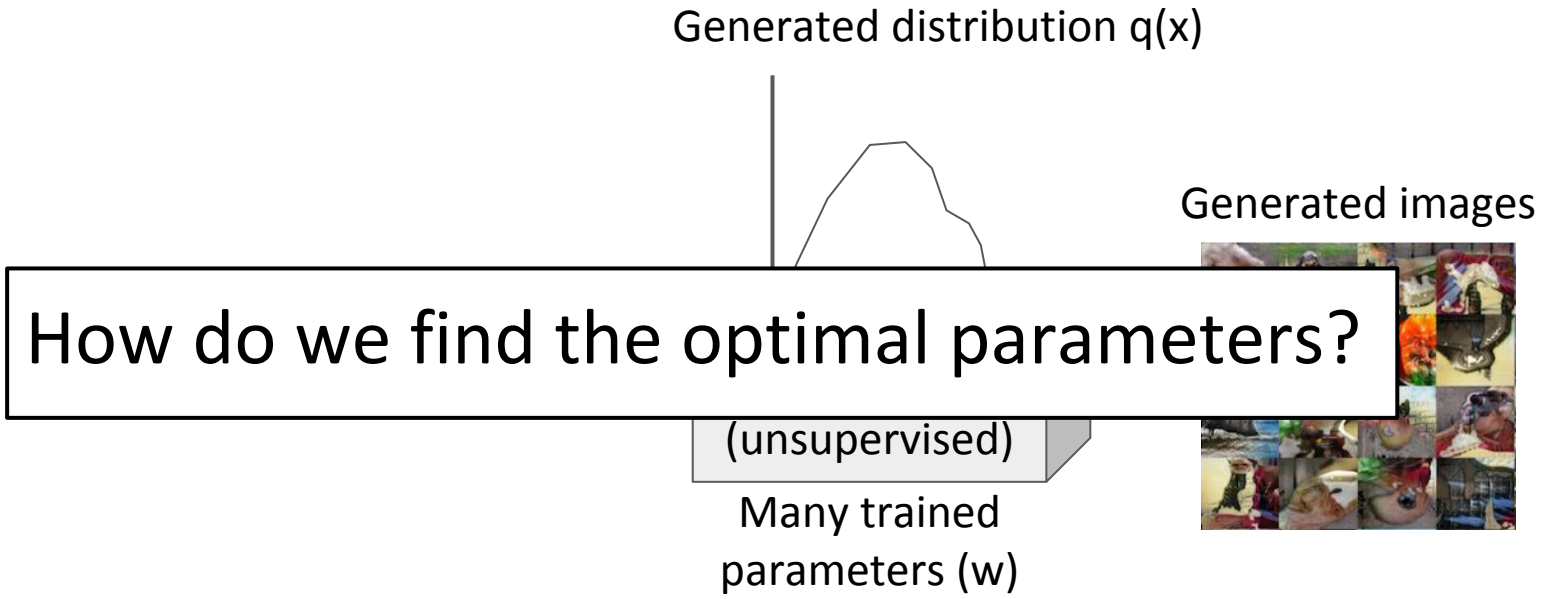
Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Generative models



Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Generative models



Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

Helmholtz machines and the Wake-Sleep algorithm

- In 1995, Geoff Hinton et. al. proposed the use of a specific artificial neural network, the **Helmholtz machine**, to act as the generative model.
- They proposed the **“Wake-Sleep”** algorithm as the method to find optimal parameters.
- The model was unique in that it was neurobiologically plausible:

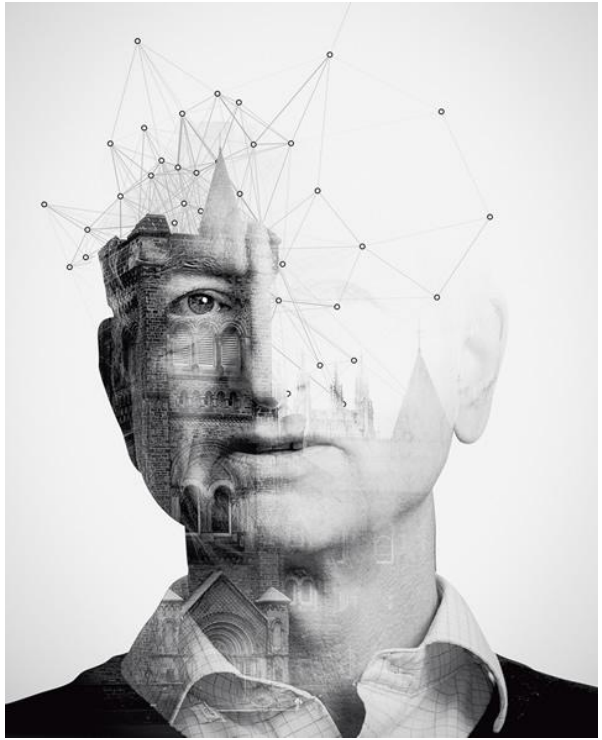


Illustration by Jesse Lenz,
based on a photo by Noah Berger.

Helmholtz machines and the Wake-Sleep algorithm

- In 1995, Geoff Hinton et. al. proposed the use of a specific artificial neural network, the **Helmholtz machine**, to act as the generative model.
- They proposed the **“Wake-Sleep”** algorithm as the method to find optimal parameters.
- The model was unique in that it was neurobiologically plausible:
    a. Unsupervised learning, no labeled data required.

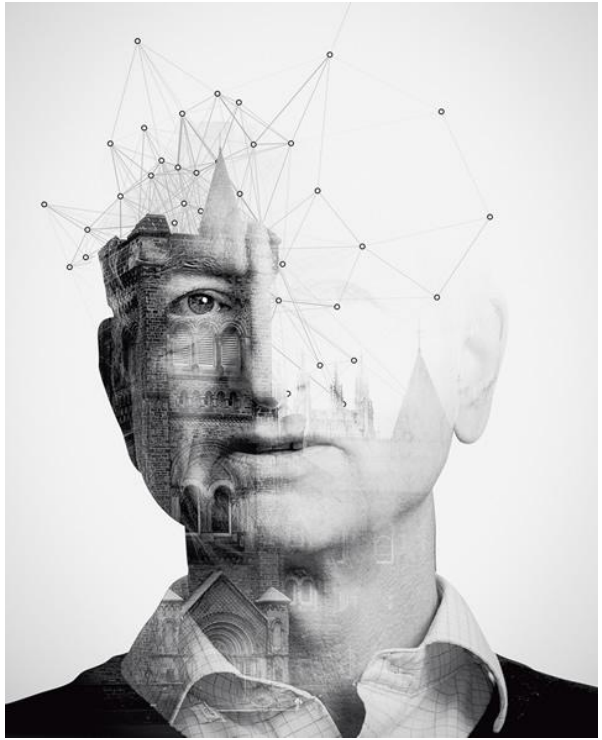


Illustration by Jesse Lenz,
based on a photo by Noah Berger.

Helmholtz machines and the Wake-Sleep algorithm

- In 1995, Geoff Hinton et. al. proposed the use of a specific artificial neural network, the **Helmholtz machine**, to act as the generative model.
- They proposed the **“Wake-Sleep”** algorithm as the method to find optimal parameters.
- The model was unique in that it was neurobiologically plausible:
  a. Unsupervised learning, no labeled data required.
  b. Local parameter updates, no need for communicating a precise measurement of error to all parameters.

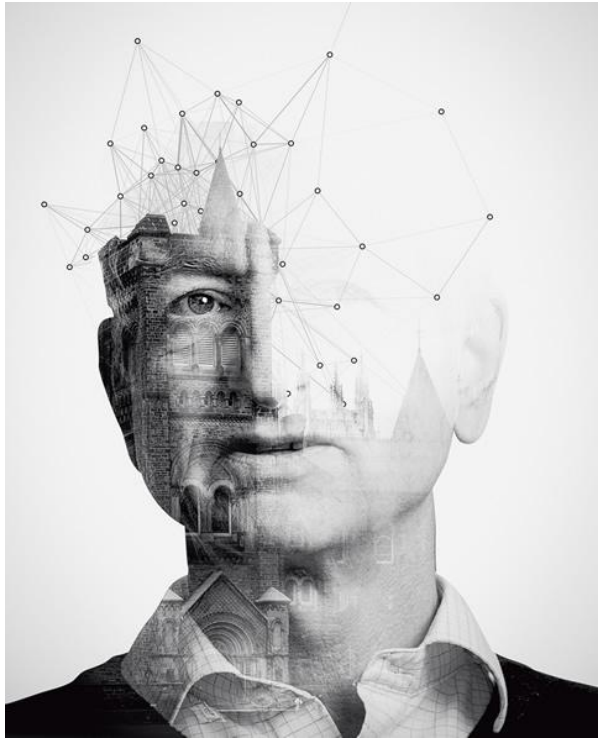


Illustration by Jesse Lenz,
based on a photo by Noah Berger.

Helmholtz machines and the Wake-Sleep algorithm

- In 1995, Geoff Hinton et. al. proposed the use of a specific artificial neural network, the **Helmholtz machine**, to act as the generative model.
- They proposed the **“Wake-Sleep”** algorithm as the method to find optimal parameters.
- The model was unique in that it was neurobiologically plausible:
  a. Unsupervised learning, no labeled data required.
  b. Local parameter updates, no need for communicating a precise measurement of error to all parameters.

- Model: Feed-forward neural network
  Learning algorithm: Backpropagation of errors

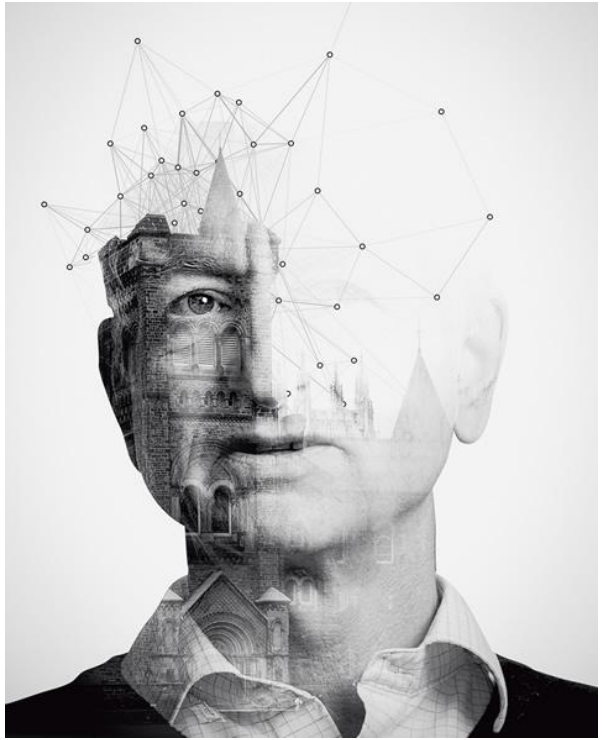


Illustration by Jesse Lenz,
based on a photo by Noah Berger.

Helmholtz machines and the Wake-Sleep algorithm

- In 1995, Geoff Hinton et. al. proposed the use of a specific artificial neural network, the **Helmholtz machine**, to act as the generative model.
- They proposed the **“Wake-Sleep”** algorithm as the method to find optimal parameters.
- The model was unique in that it was neurobiologically plausible:
  a. Unsupervised learning, no labeled data required.
  b. Local parameter updates, no need for communicating a precise measurement of error to all parameters.

- Model: ~~Feed-forward neural network~~
  Learning algorithm: ~~Backpropagation of errors~~
- Model: **Helmholtz machine**
  Learning algorithm: **Wake-Sleep**

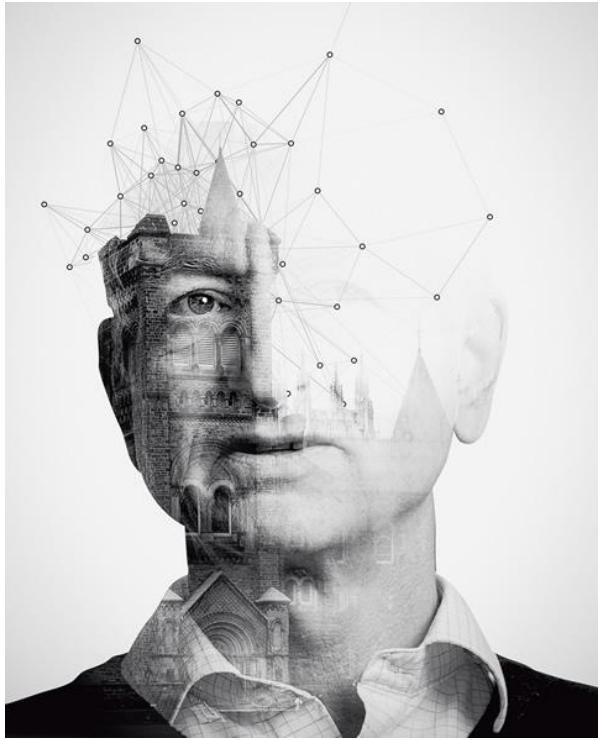
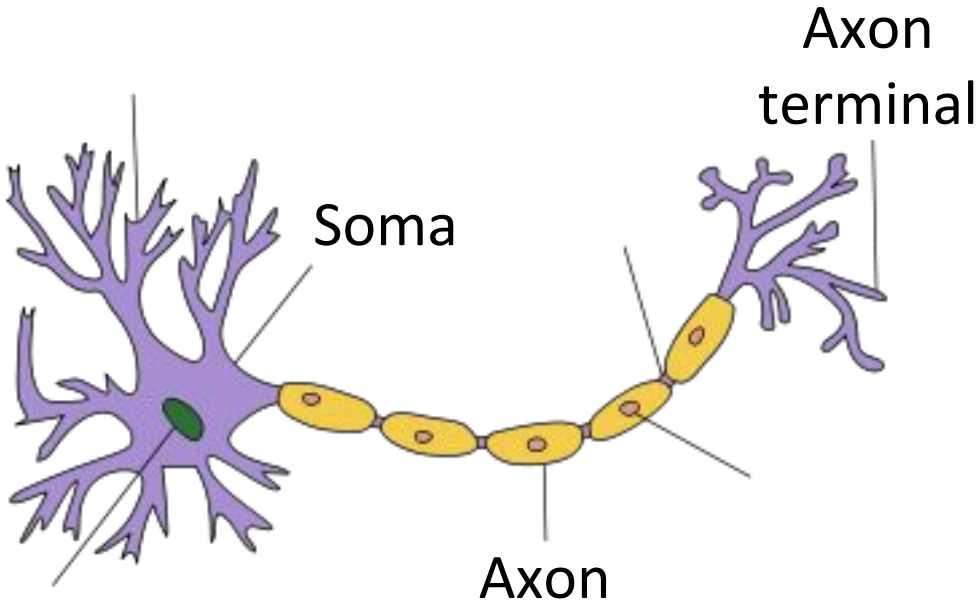
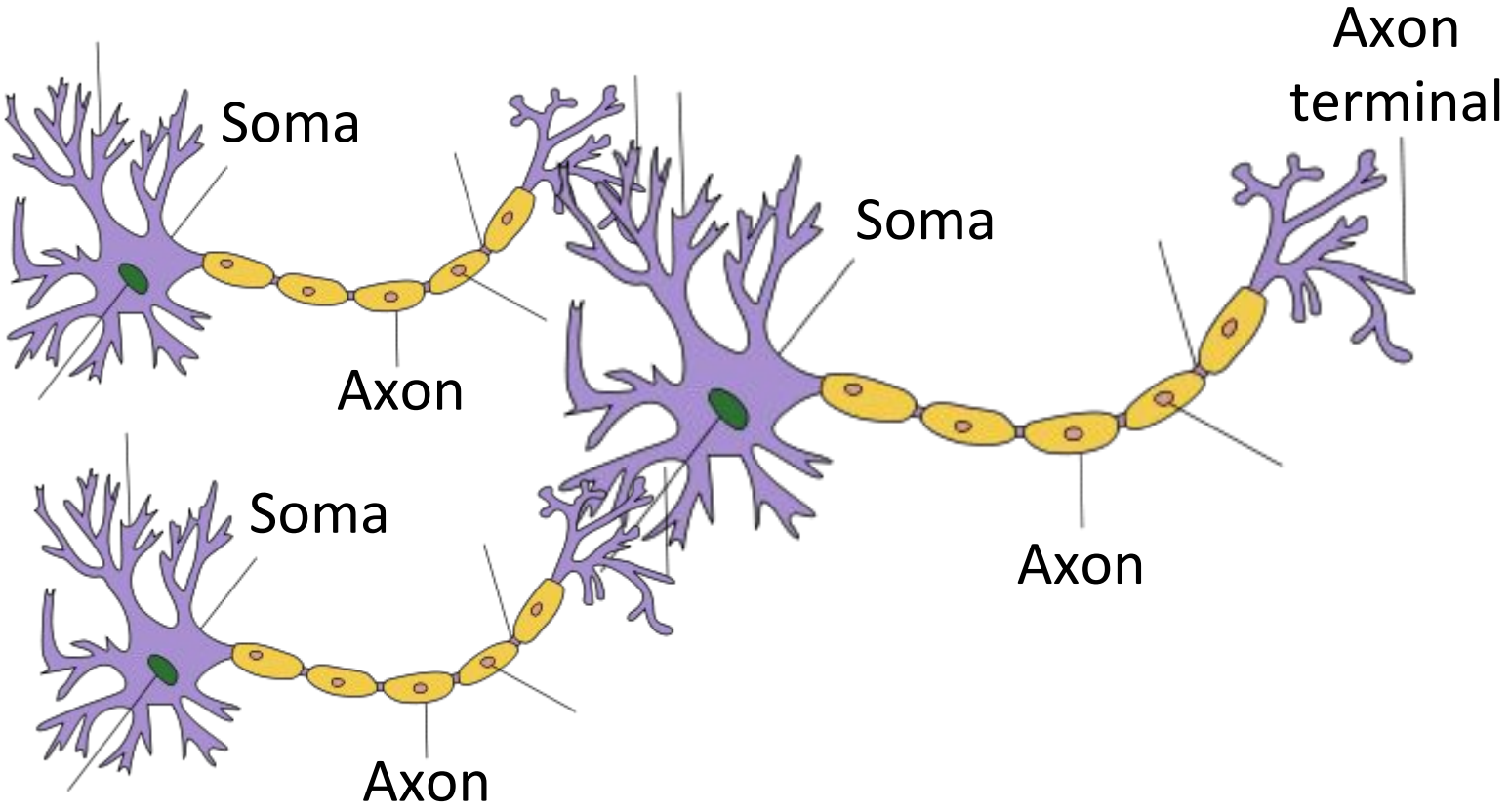


Illustration by Jesse Lenz,
based on a photo by Noah Berger.

Artificial neural networks review

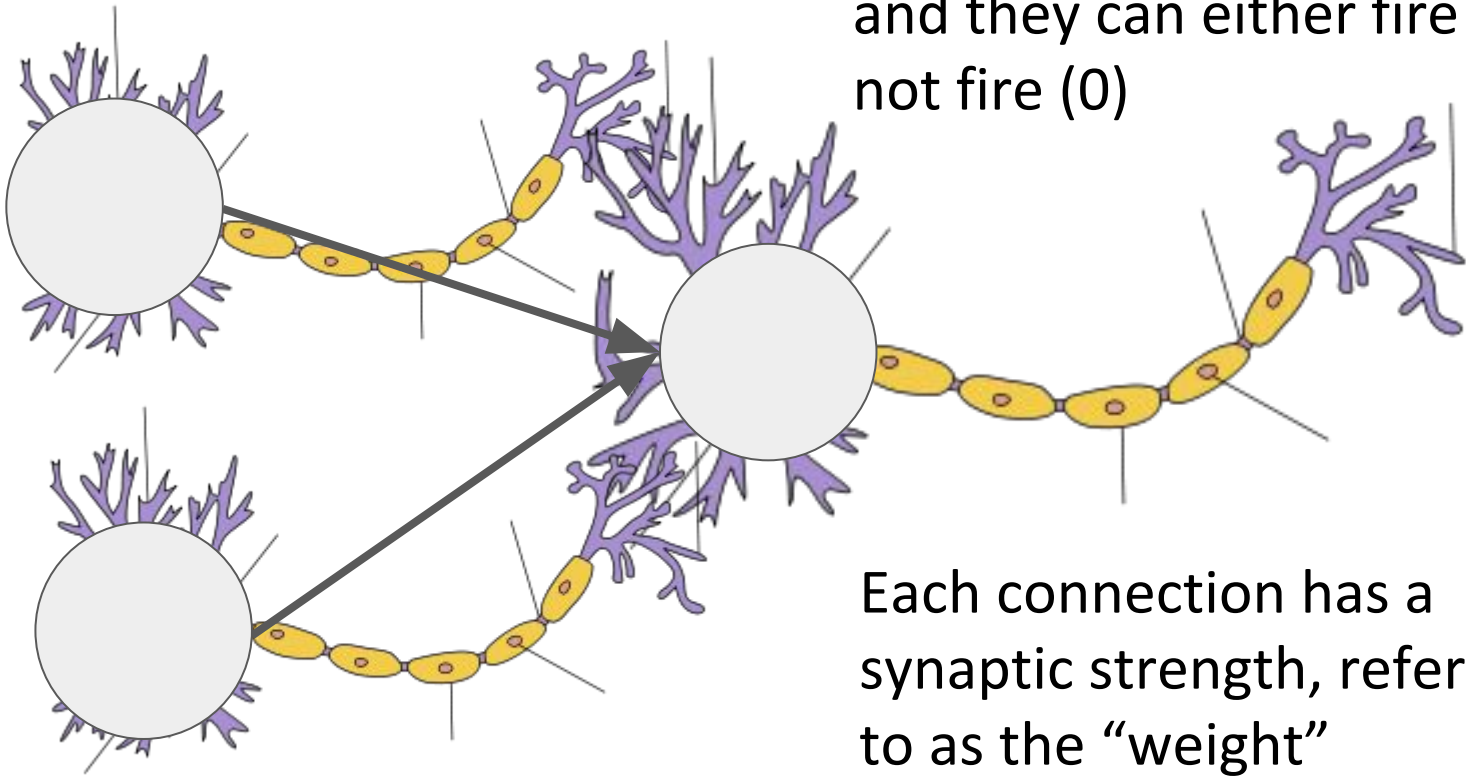


Artificial neural networks review



Artificial neural networks

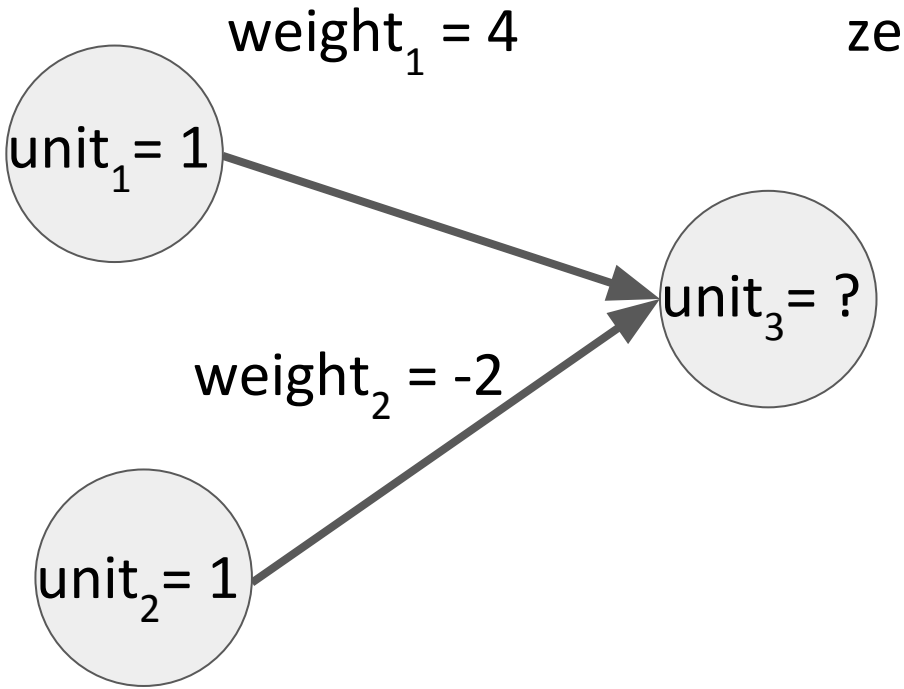
Neurons are called “units”, and they can either fire (1) or not fire (0)



Each connection has a synaptic strength, referred to as the “weight”

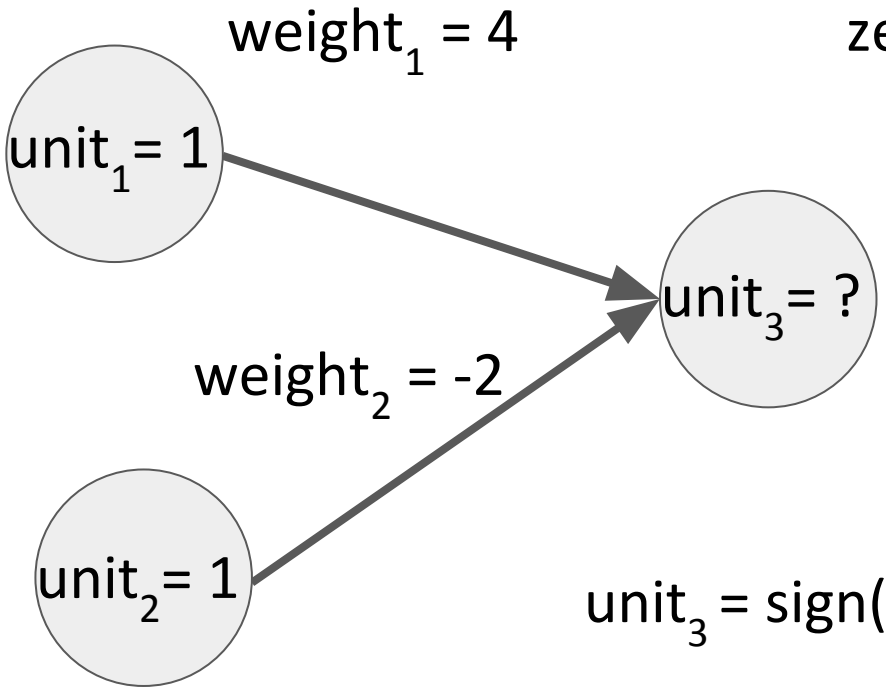
Artificial neural networks

Units fire when the sum of their inputs exceeds zero



Artificial neural networks

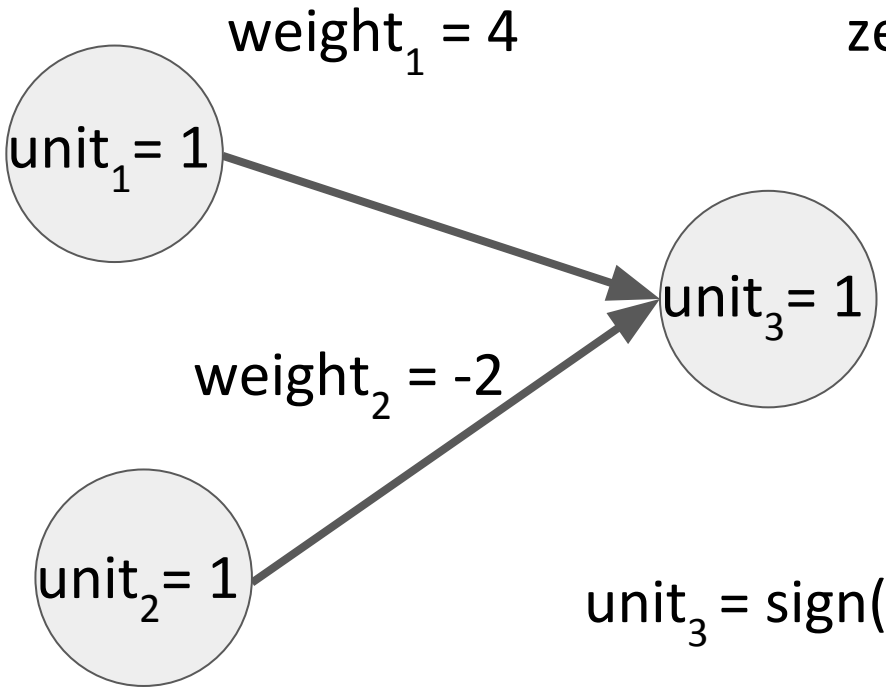
Units fire when the sum of their inputs exceeds zero



$$unit_3 = sign(unit_1 * weight_1 + unit_2 * weight_2)$$

Artificial neural networks

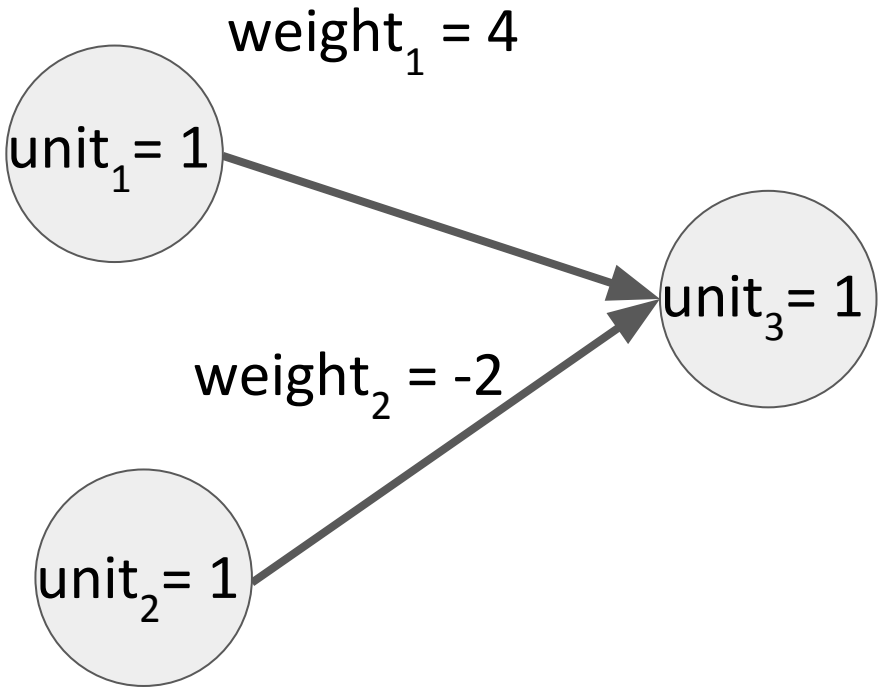
Units fire when the sum of their inputs exceeds zero



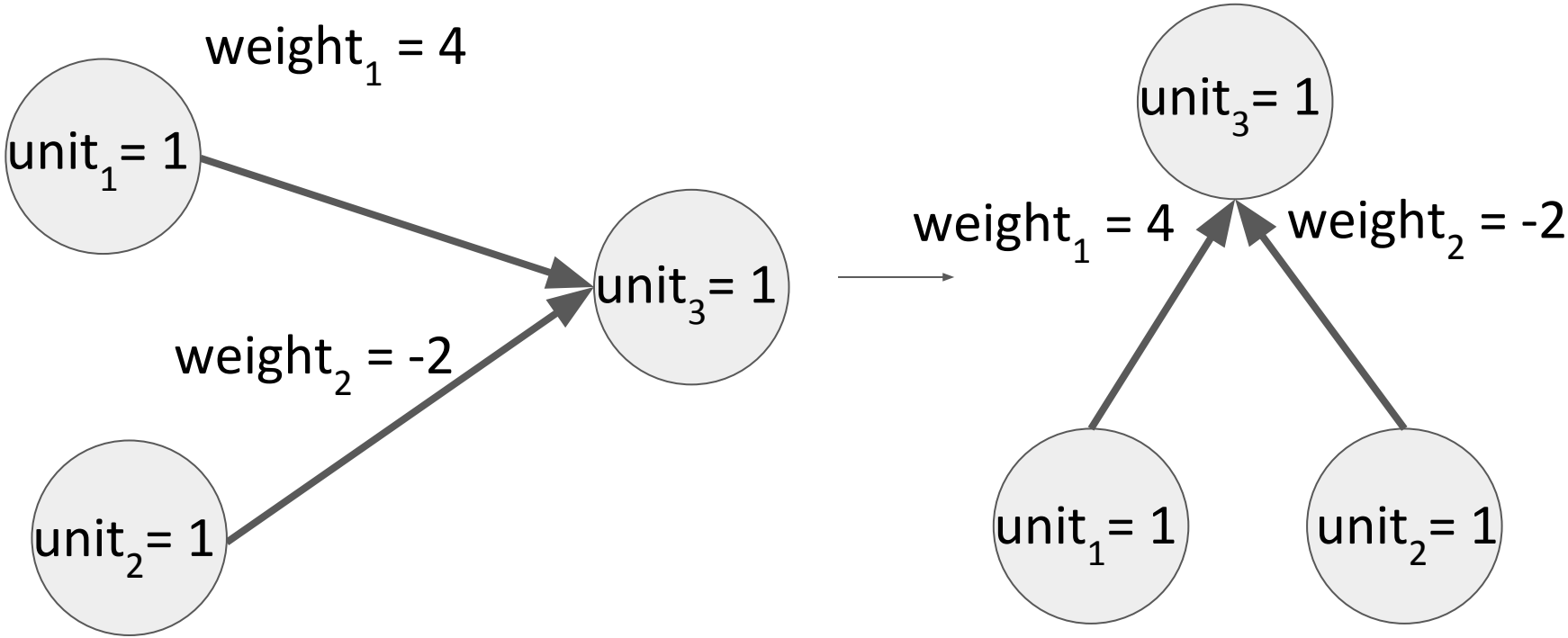
$$unit_3 = sign(unit_1 * weight_1 + unit_2 * weight_2)$$

$$unit_3 = sign(1 * 4 + 1 * -2) = sign(2) = 1$$

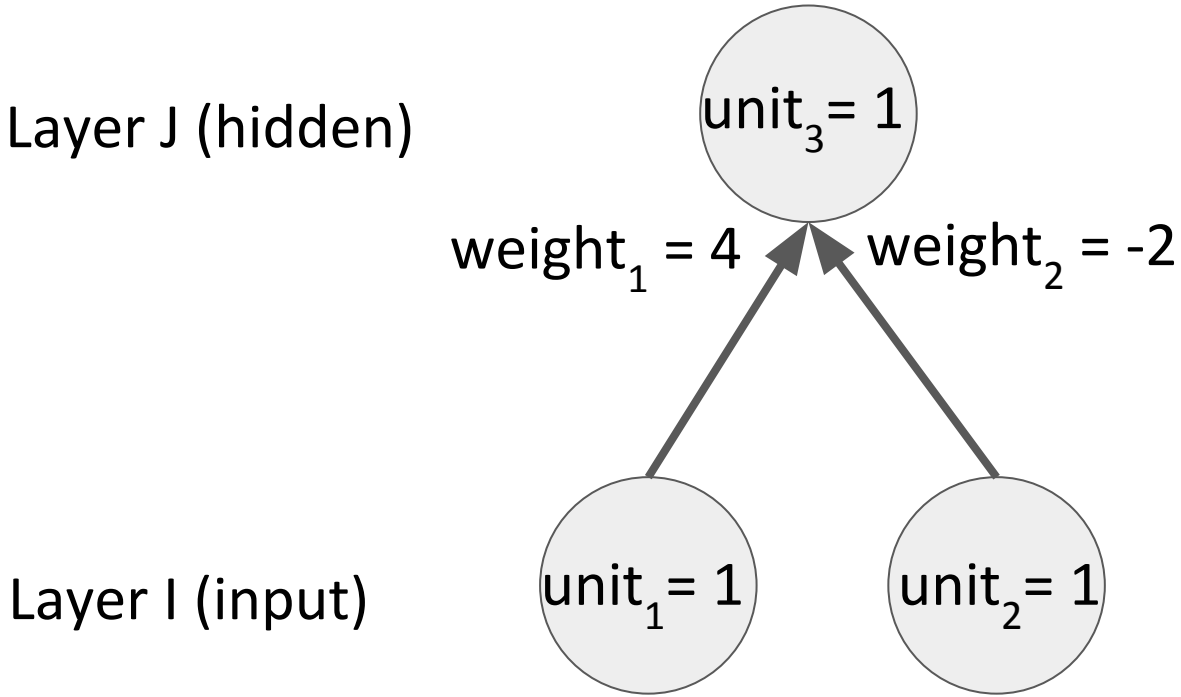
Artificial neural networks



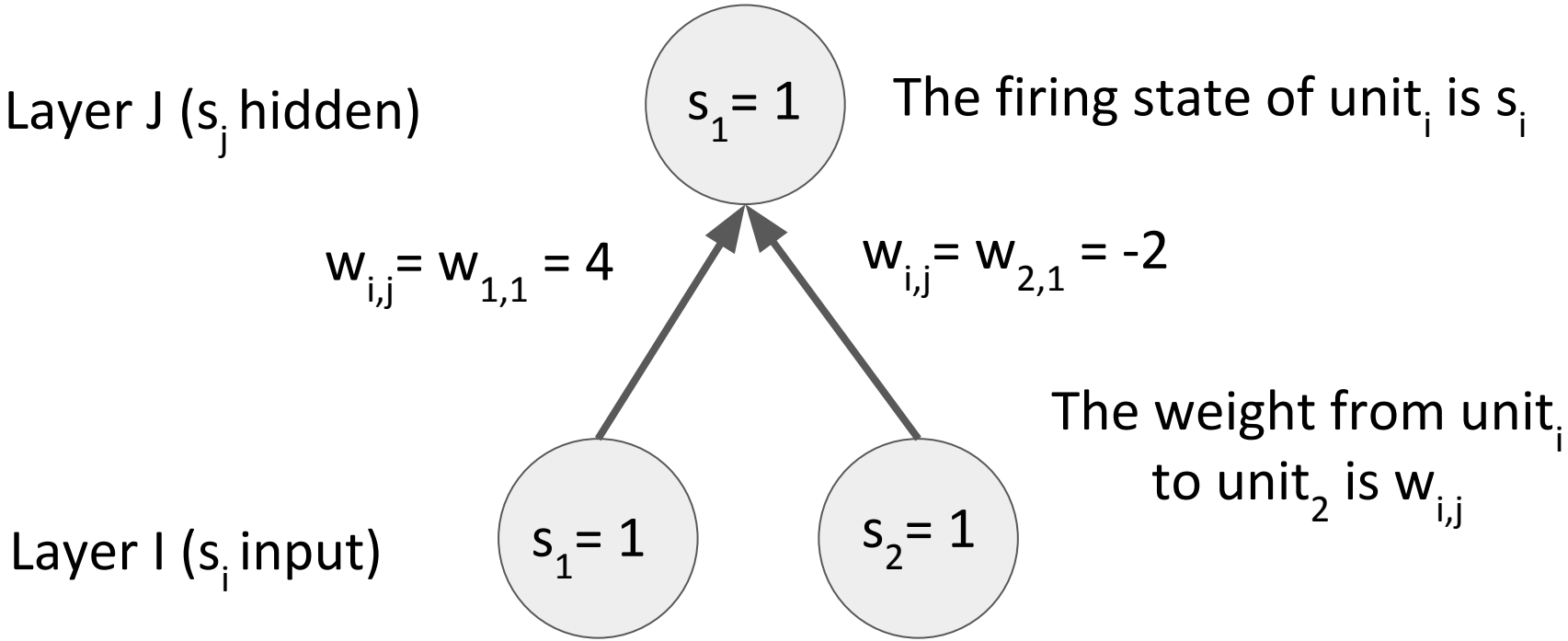
Artificial neural networks



Artificial neural networks



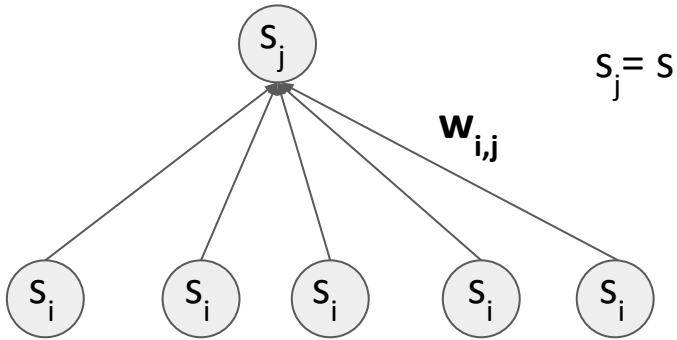
Artificial neural networks



Artificial neural networks

Layer J (s_j hidden)

Layer I (s_i input)



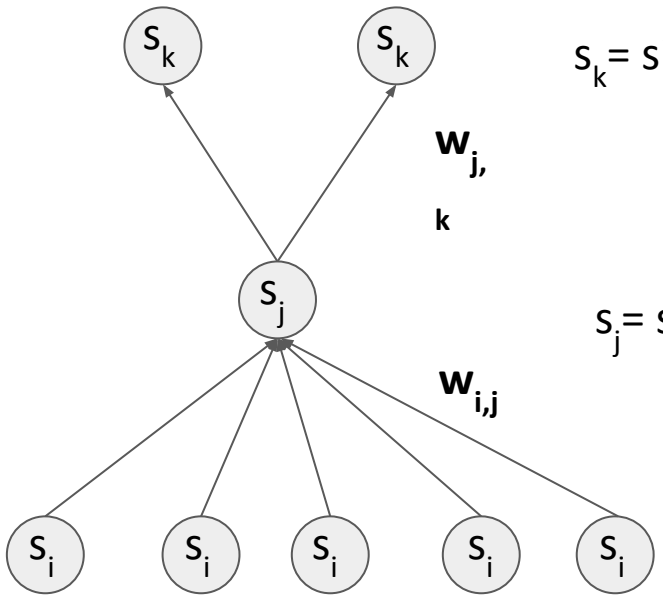
$s_j = \text{sign}(\sum s_i w_{i,j})$

Artificial neural networks

Layer K (s_k hidden)

Layer J (s_j hidden)

Layer I (s_i input)



$s_k = \text{sign}(\sum s_j w_{j,k})$

$s_j = \text{sign}(\sum s_i w_{i,j})$

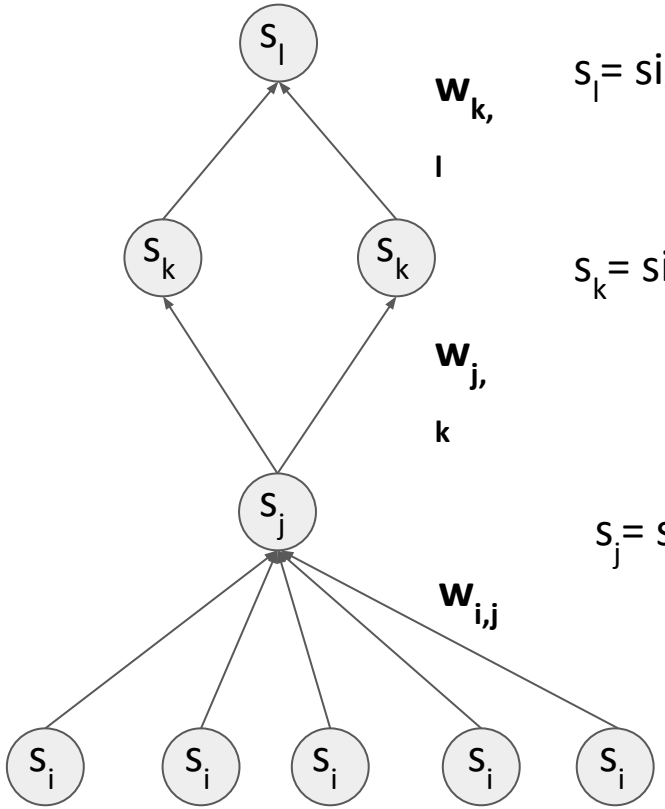
Artificial neural networks

Layer L (s_l hidden)

Layer K (s_k hidden)

Layer J (s_j hidden)

Layer I (s_i input)



$s_l = \text{sign}(\sum s_k w_{k,l})$

$s_k = \text{sign}(\sum s_j w_{j,k})$

$s_j = \text{sign}(\sum s_i w_{i,j})$

Artificial neural networks

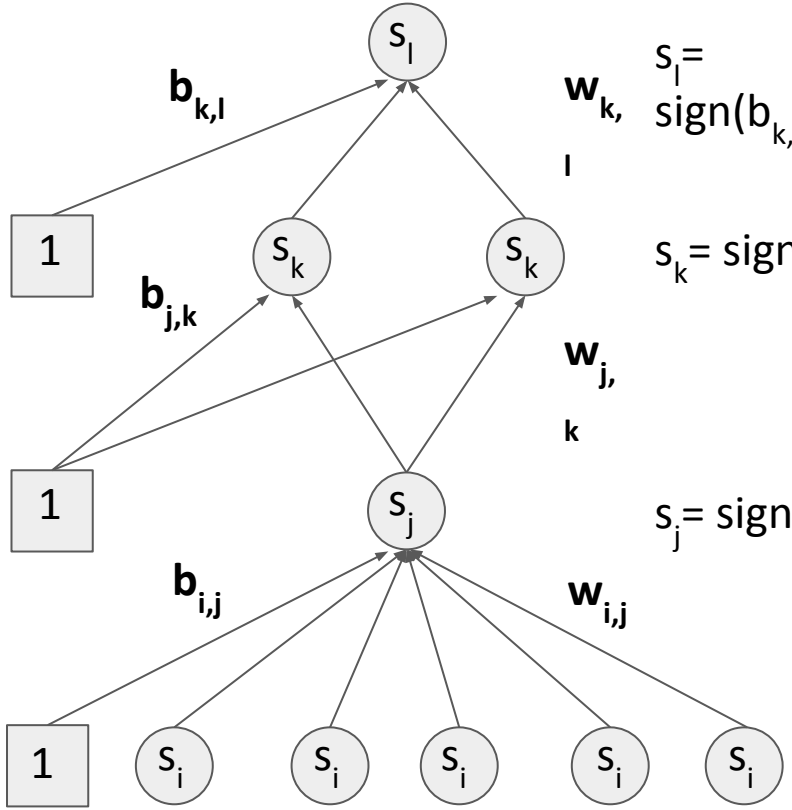
bias unit = 1

Layer L (s_l hidden)

Layer K (s_k hidden)

Layer J (s_j hidden)

Layer I (s_i input)

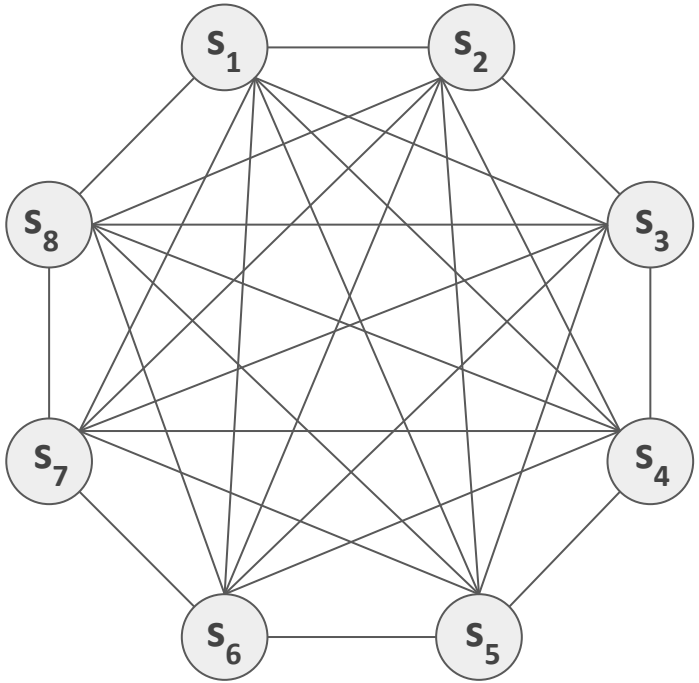


$s_l = \text{sign}(b_{k,l} + \sum s_k w_{k,l})$

$s_k = \text{sign}(b_{j,k} + \sum s_j w_{j,k})$

$s_j = \text{sign}(b_{i,j} + \sum s_i w_{i,j})$

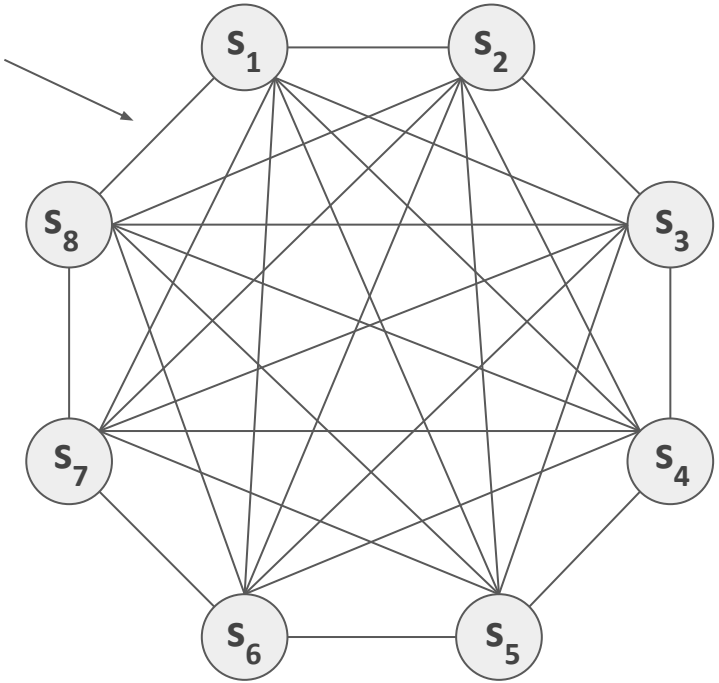
Hopfield networks



Not used for data generation, used for pattern completion.

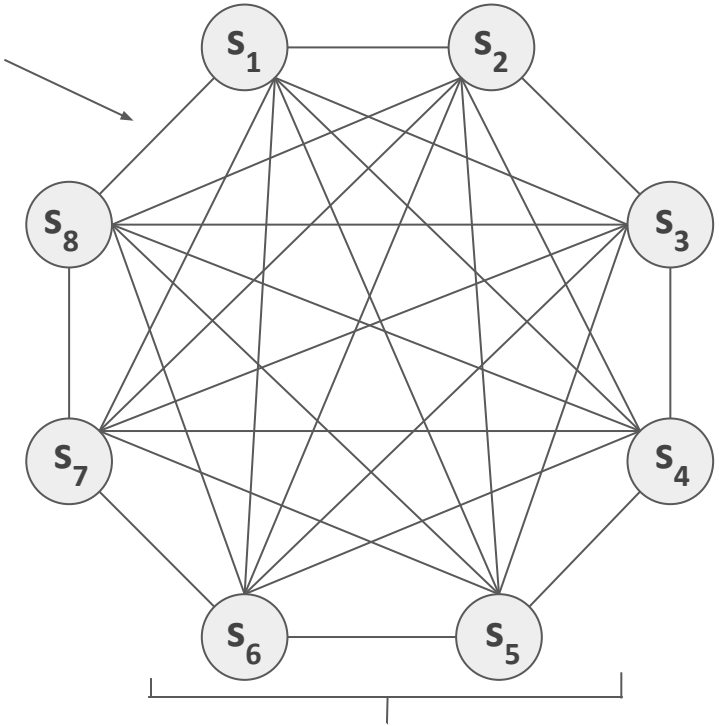
Hopfield networks

Symmetric connections



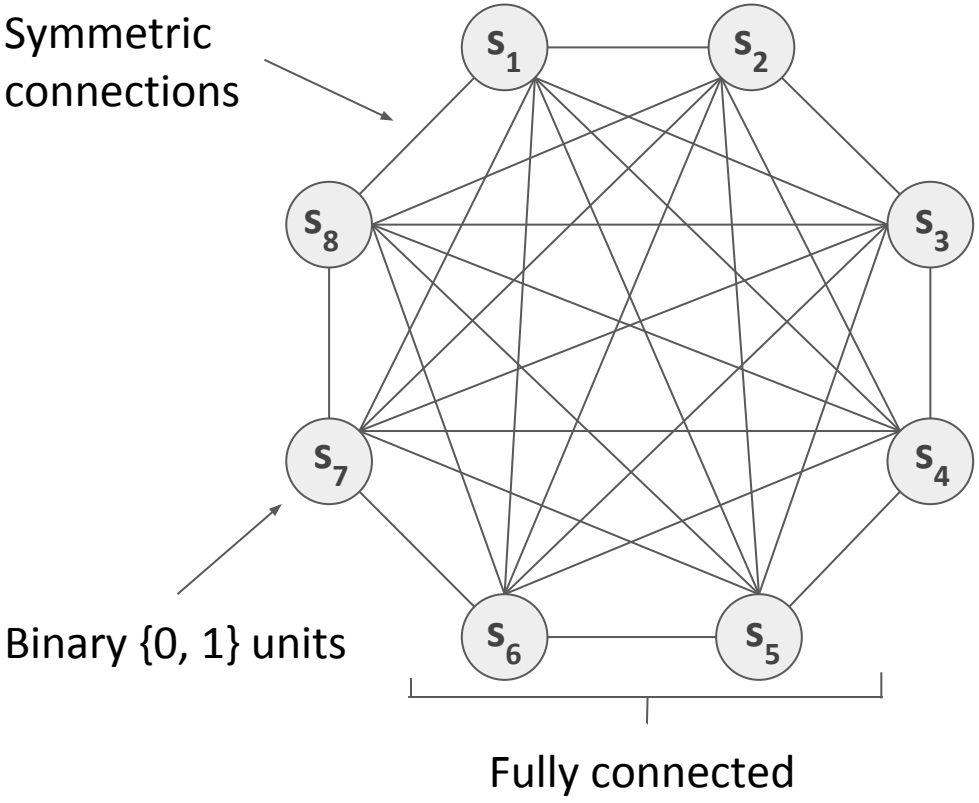
Hopfield networks

Symmetric connections



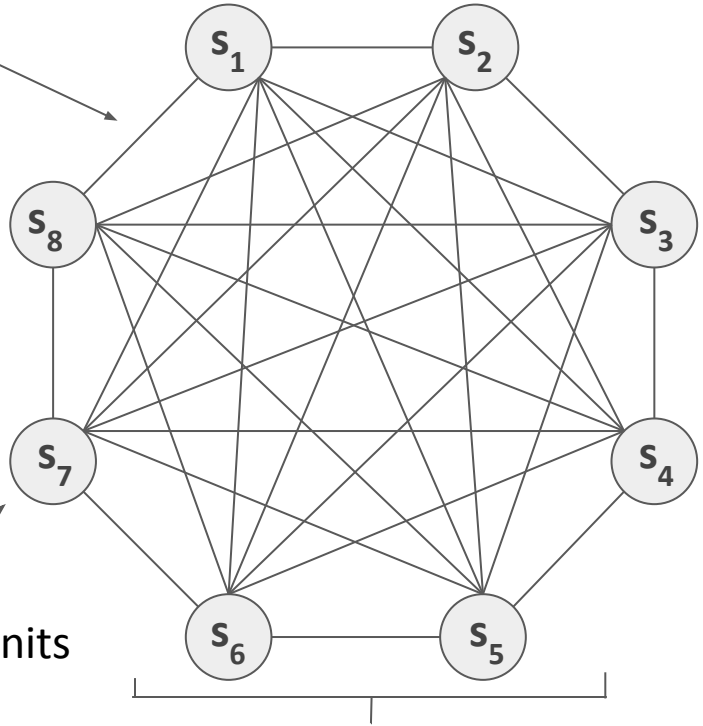
Fully connected

Hopfield networks



Hopfield networks

Symmetric connections

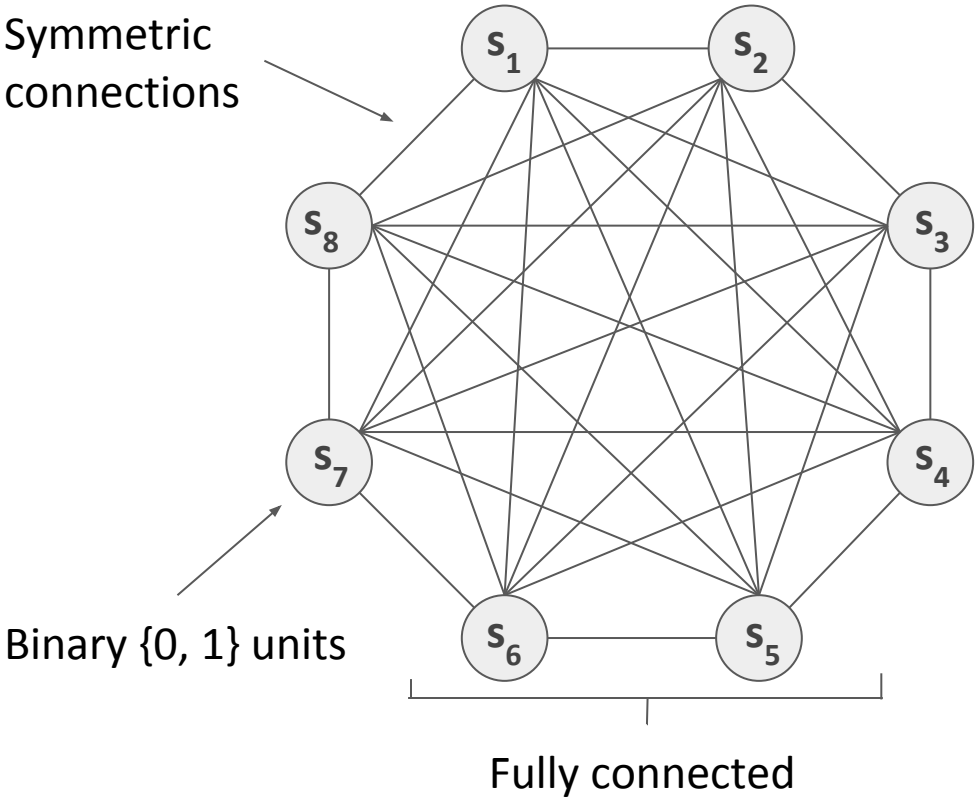


Rearrange units into layers

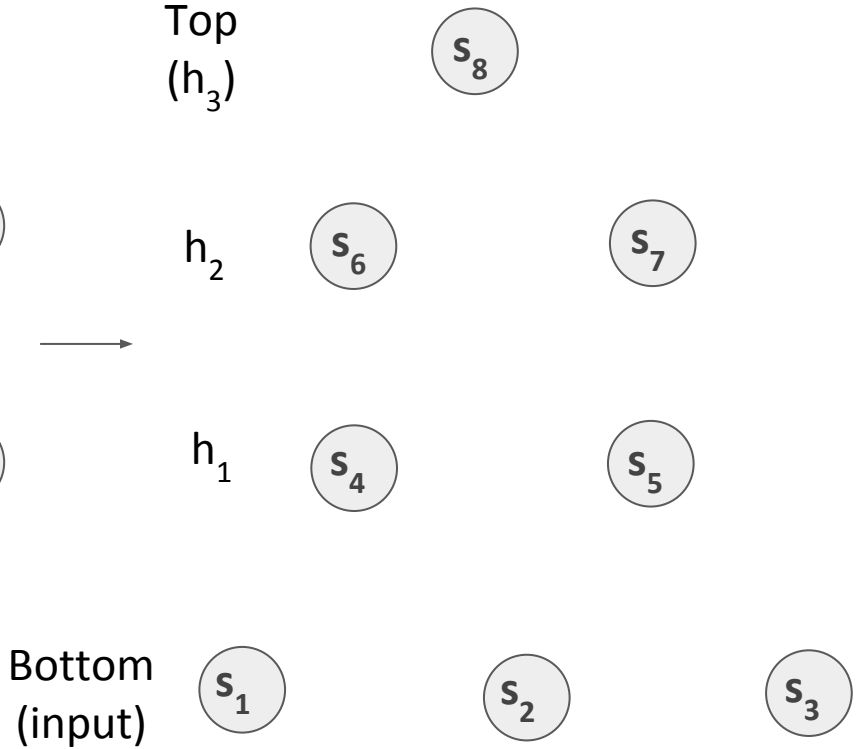
Binary {0, 1} units

Fully connected

Hopfield networks

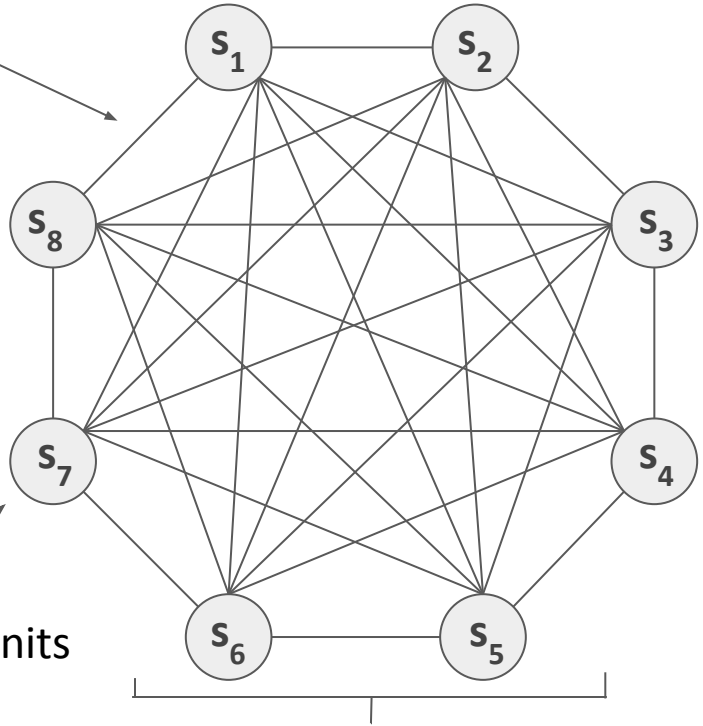


Helmholtz machine



Hopfield networks

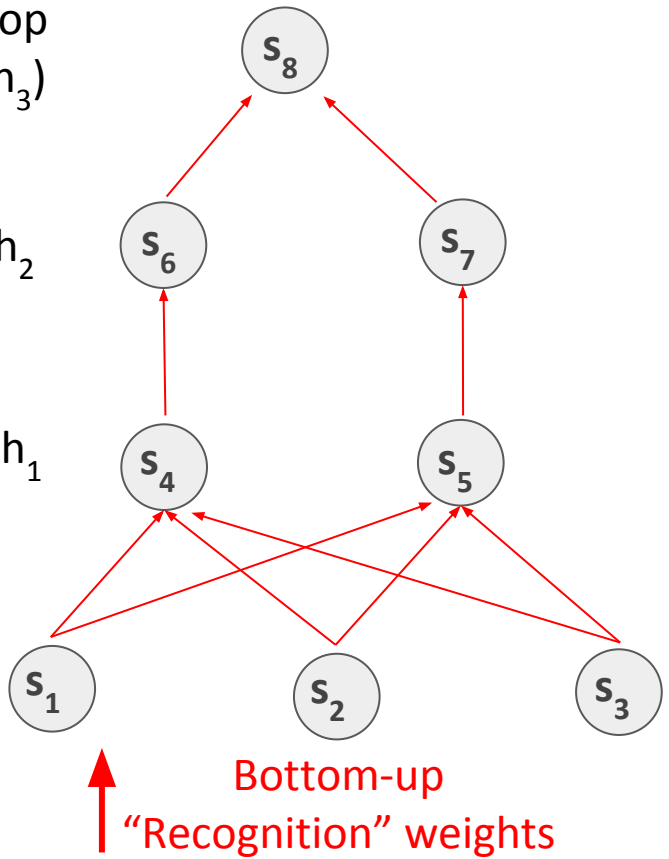
Symmetric connections



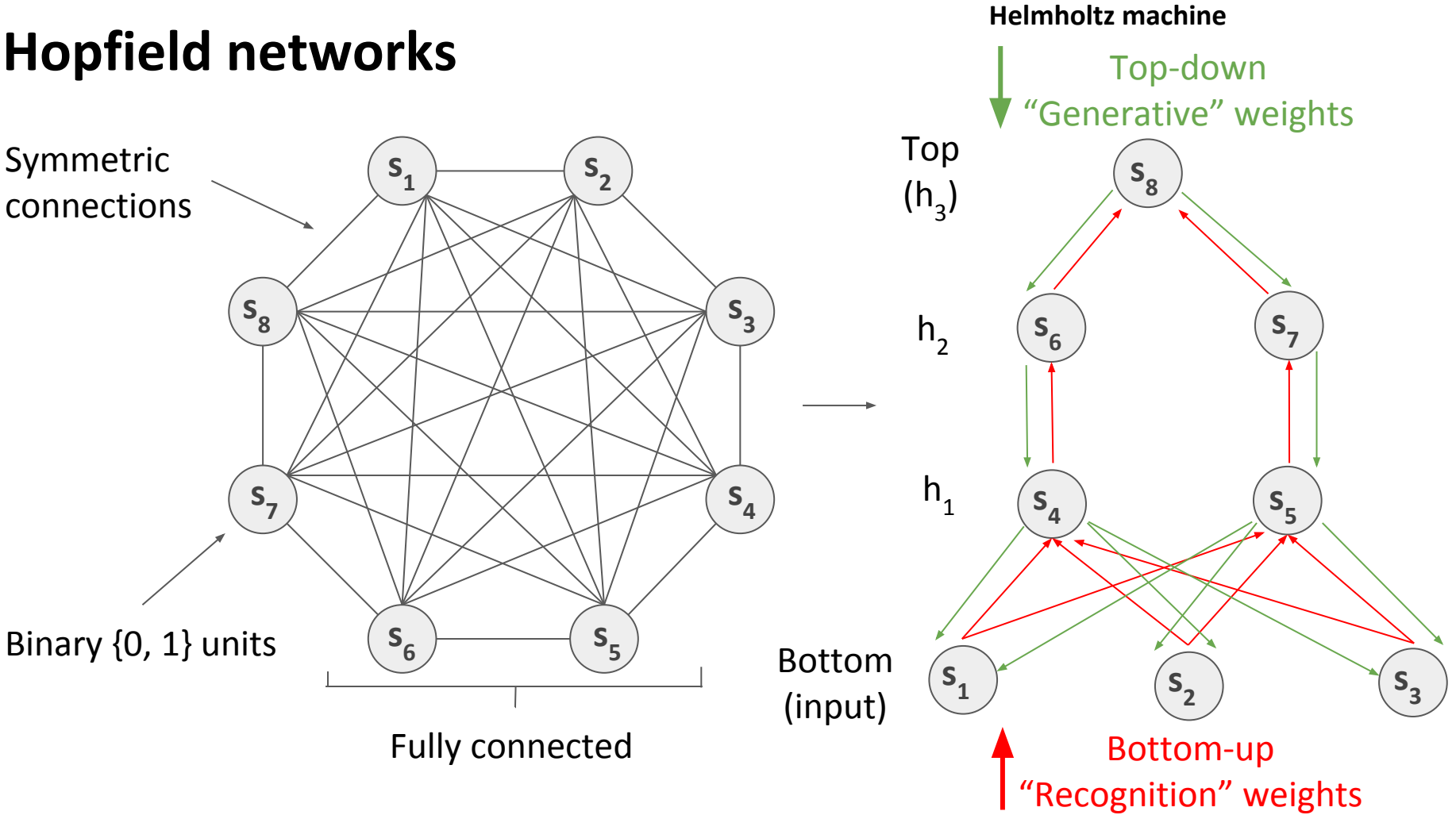
Binary {0, 1} units

Fully connected

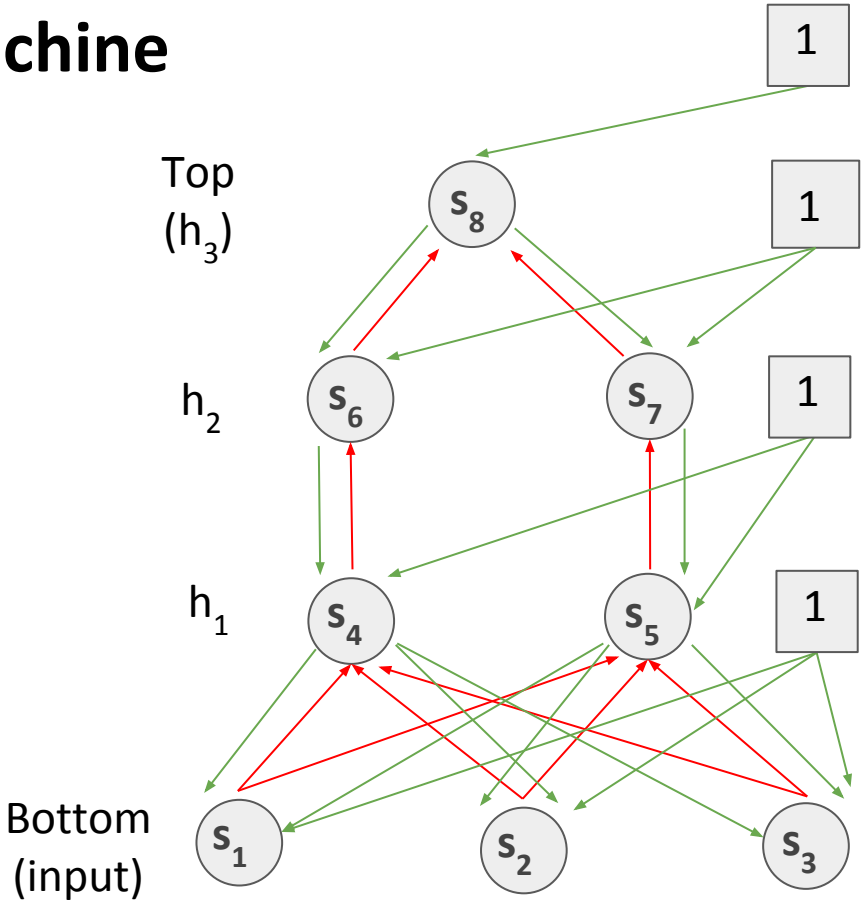
Helmholtz machine



Hopfield networks



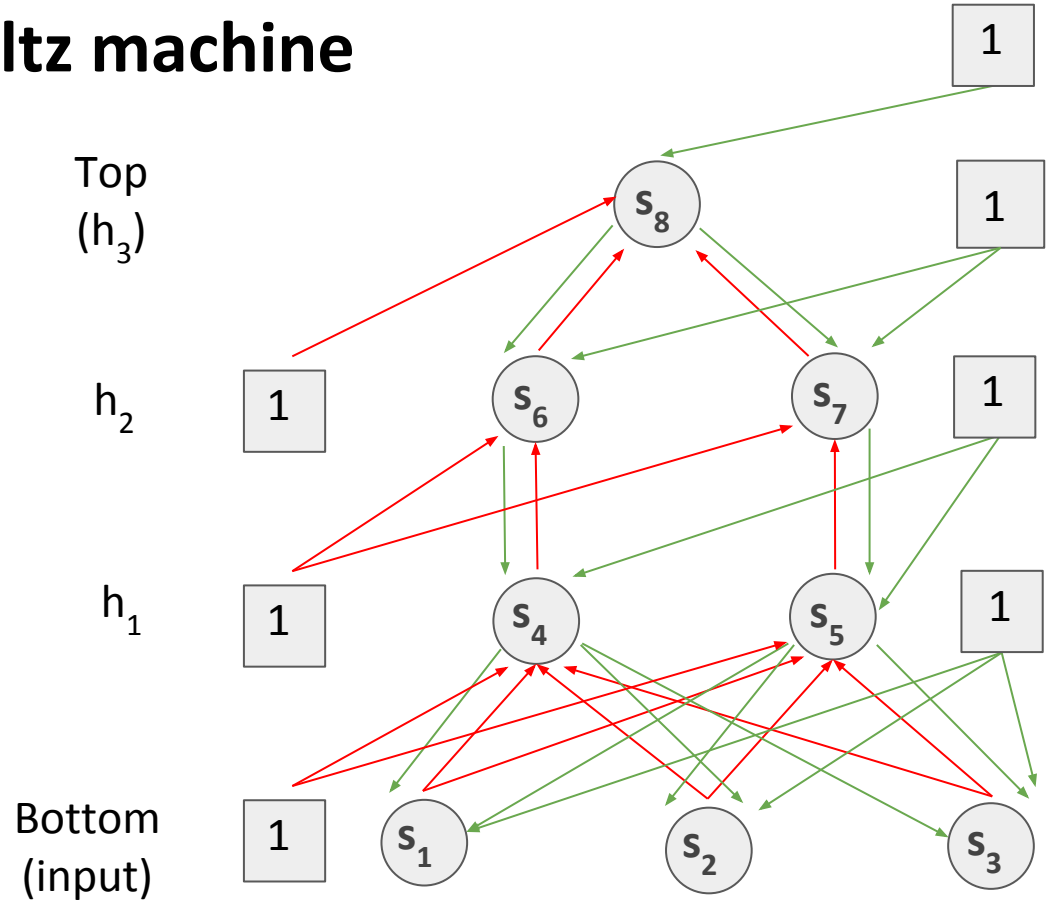
Helmholtz machine



Top-down
"Generative" weights

Bottom-up
"Recognition" weights

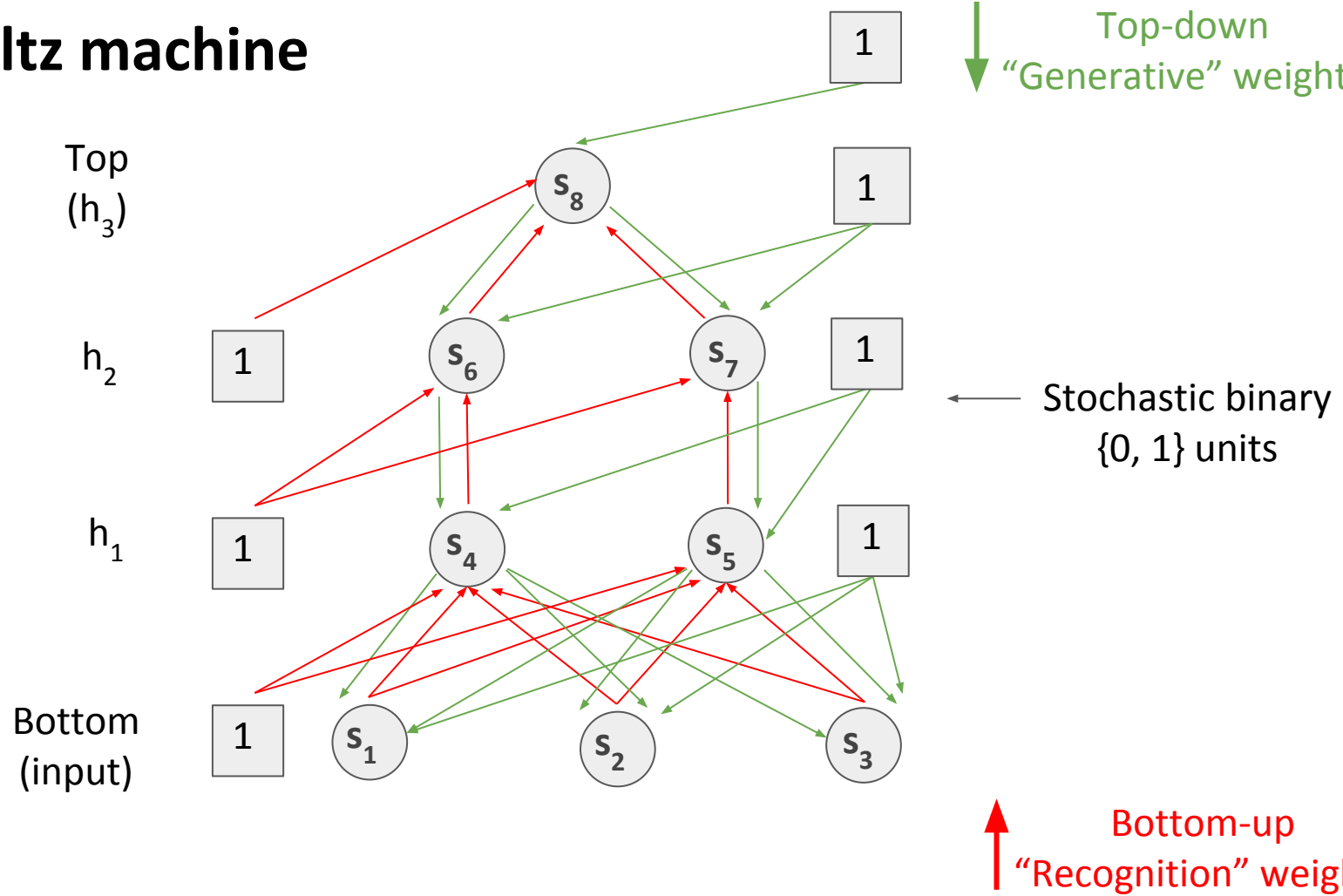
Helmholtz machine



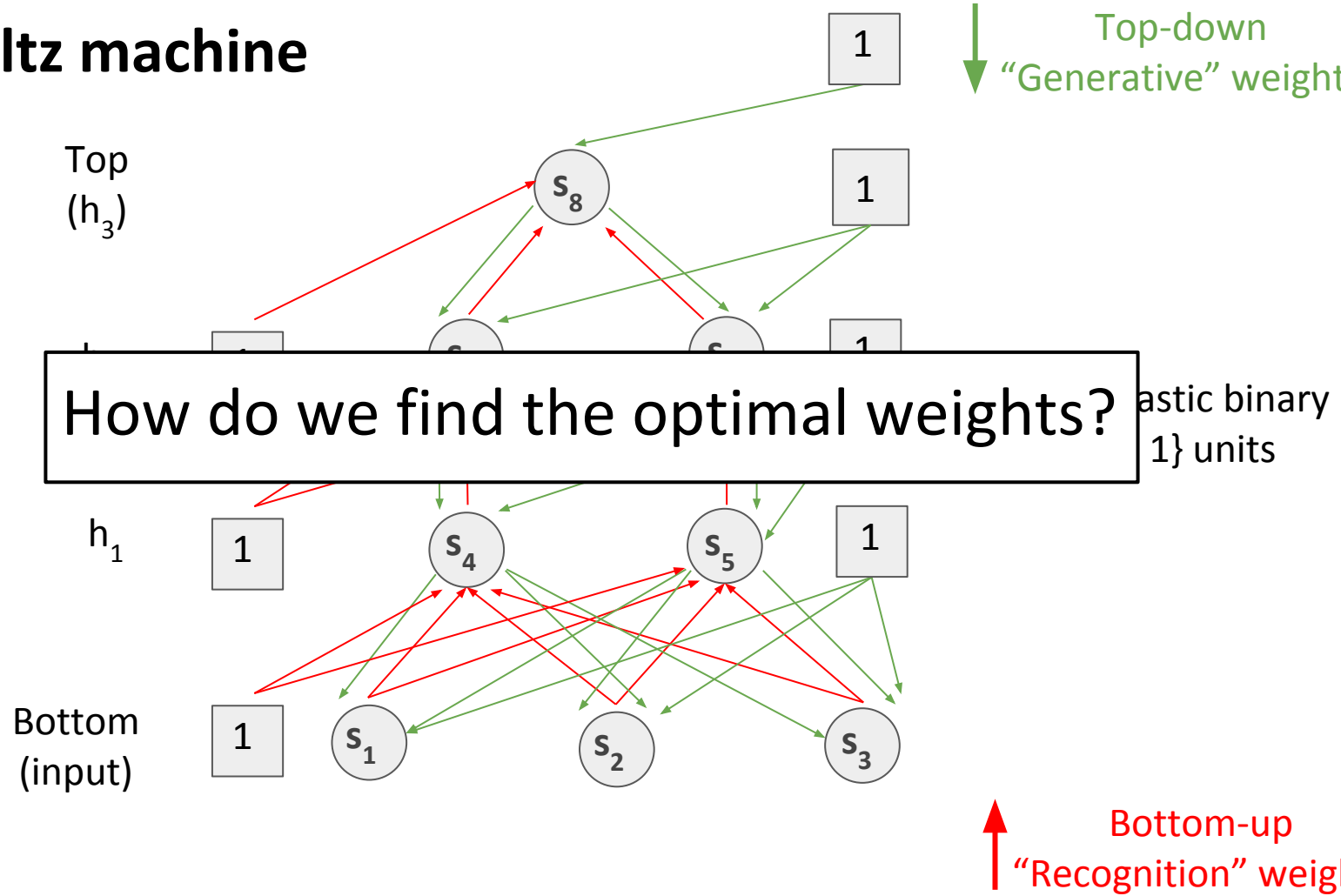
Top-down
"Generative" weights

Bottom-up
"Recognition" weights

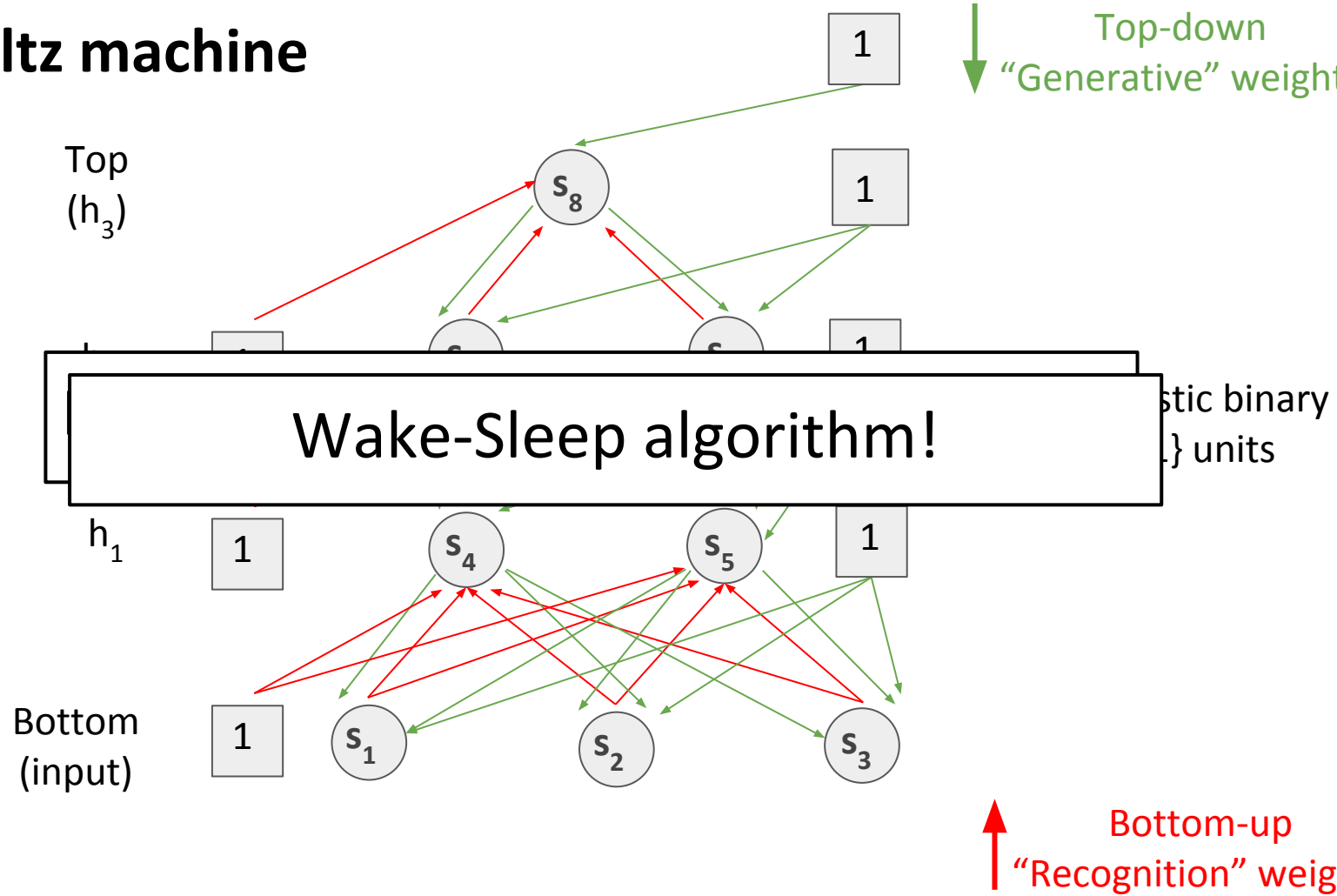
Helmholtz machine



Helmholtz machine



Helmholtz machine



The Wake-Sleep algorithm

1. Input a sample into the network
2. Wake-phase, propagate **bottom-up** using **recognition weights**, update **generative weights**
3. Sleep-phase, update **top-down** using **generative weights**, update **recognition weights**
4. Repeat over all samples

During each phase, follow two simple equations:

1. Stochastic binary unit update
2. Weight update

The Wake-Sleep algorithm

1. Input a sample into the network
2. Wake-phase, propagate **bottom-up** using **recognition weights**, update **generative weights**
3. Sleep-phase, update **top-down** using **generative weights**, update **recognition weights**
4. Repeat over all samples

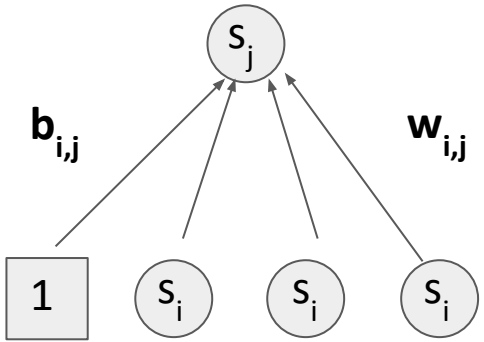
During each phase, follow two simple equations:

1. **Stochastic binary unit update**
2. Weight update

Stochastic binary units

Basic binary units

$s_j = \text{sign}(b_{i,j} + \sum s_i w_{i,j})$



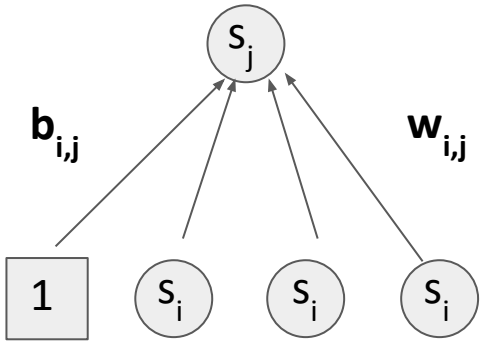
Stochastic binary units

Basic binary units

$$s_j = \text{sign}(b_{i,j} + \sum s_i w_{i,j})$$

Stochastic binary units

$$\text{Prob}(s_j = 1) = \frac{1}{1 + \exp(-b_{i,j} - \sum s_i w_{i,j})}$$



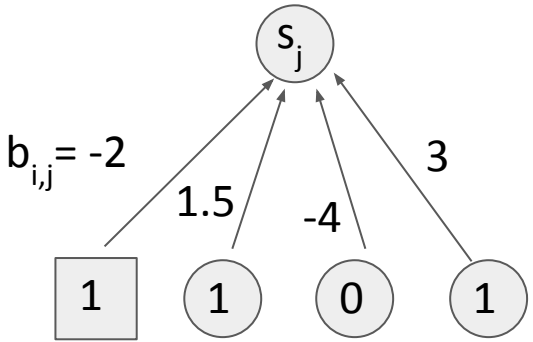
Stochastic binary units

Basic binary units

$$s_j = \text{sign}(b_{i,j} + \sum s_i w_{i,j})$$

Stochastic binary units

$$\text{Prob}(s_j = 1) = \frac{1}{1 + \exp(-b_{i,j} - \sum s_i w_{i,j})}$$



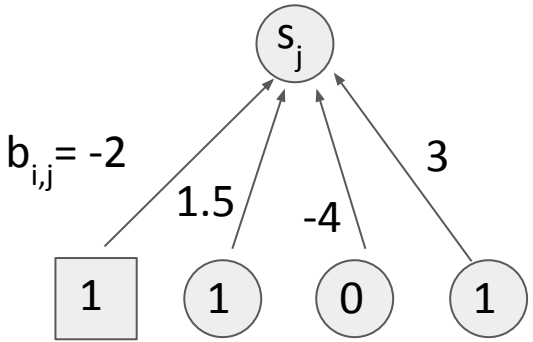
Stochastic binary units

Basic binary units

$s_j = \text{sign}(2.5)$

Stochastic binary units

$$\text{Prob}(s_j=1) = \frac{1}{1 + \exp(-2.5)}$$



Stochastic binary units

Basic binary units

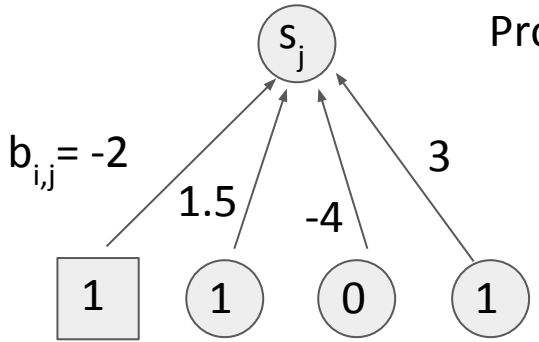
$s_j = \text{sign}(2.5)$

$s_j = 1$

Stochastic binary units

$$\text{Prob}(s_j=1) = \frac{1}{1 + \exp(-2.5)}$$

$\text{Prob}(s_j=1) = 0.92$



Stochastic binary units

Basic binary units

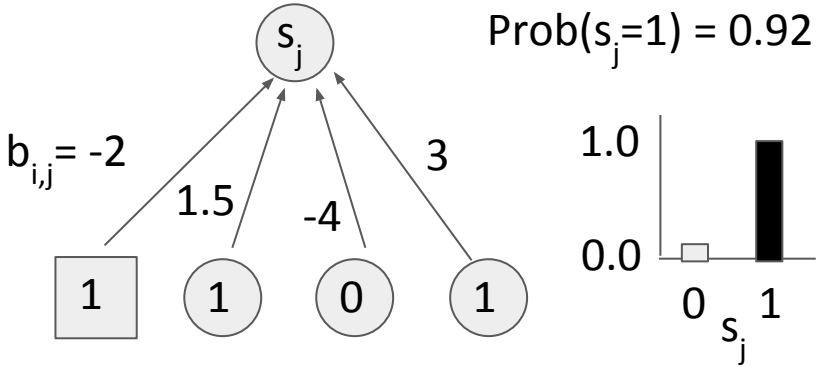
s_j= sign(2.5)

s_j= 1

Stochastic binary units

$$\text{Prob}(s_j=1) = \frac{1}{1 + \exp(-2.5)}$$

Prob(s_j=1) = 0.92



Stochastic binary units

Basic binary units

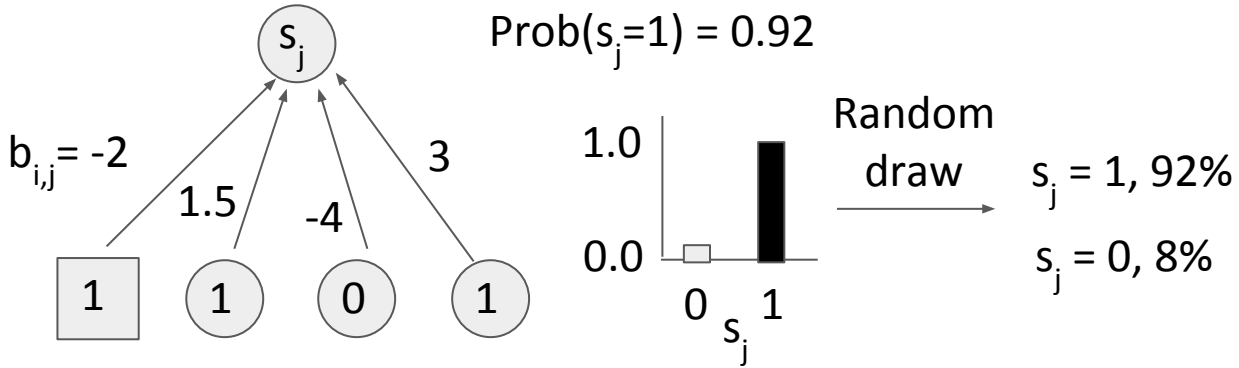
s_j= sign(2.5)

s_j= 1

Stochastic binary units

$$\text{Prob}(s_j=1) = \frac{1}{1 + \exp(-2.5)}$$

Prob(s_j=1) = 0.92



Stochastic binary units

Basic binary units

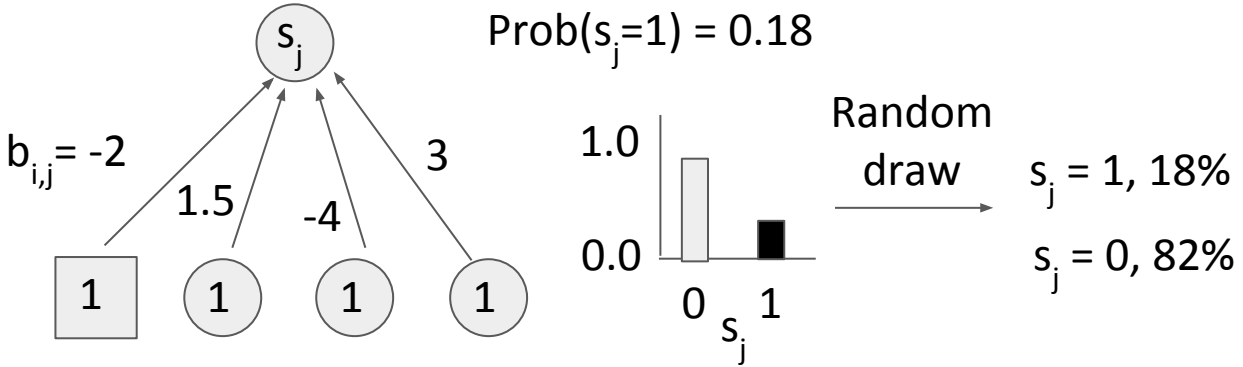
s_j= sign(-1.5)

s_j= -1

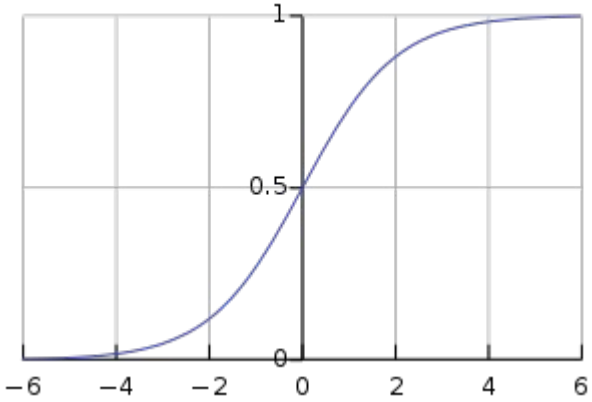
Stochastic binary units

$$\text{Prob}(s_j=1) = \frac{1}{1 + \exp(1.5)}$$

Prob(s_j=1) = 0.18



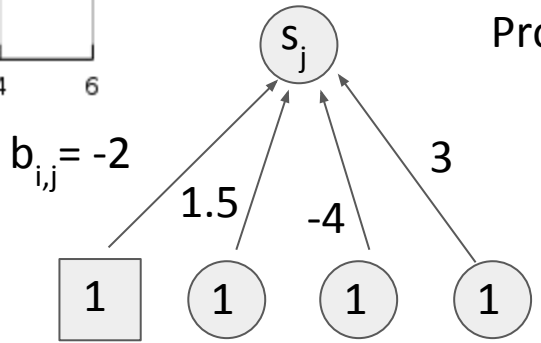
Stochastic binary units

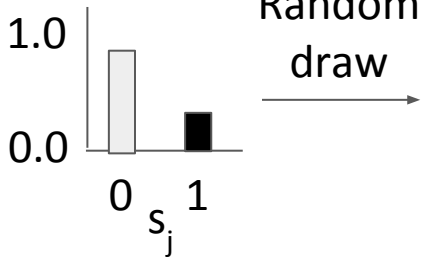


Stochastic binary units

$$\text{Prob}(s_j=1) = \frac{1}{1 + \exp(1.5)}$$

$\text{Prob}(s_j=1) = 0.18$





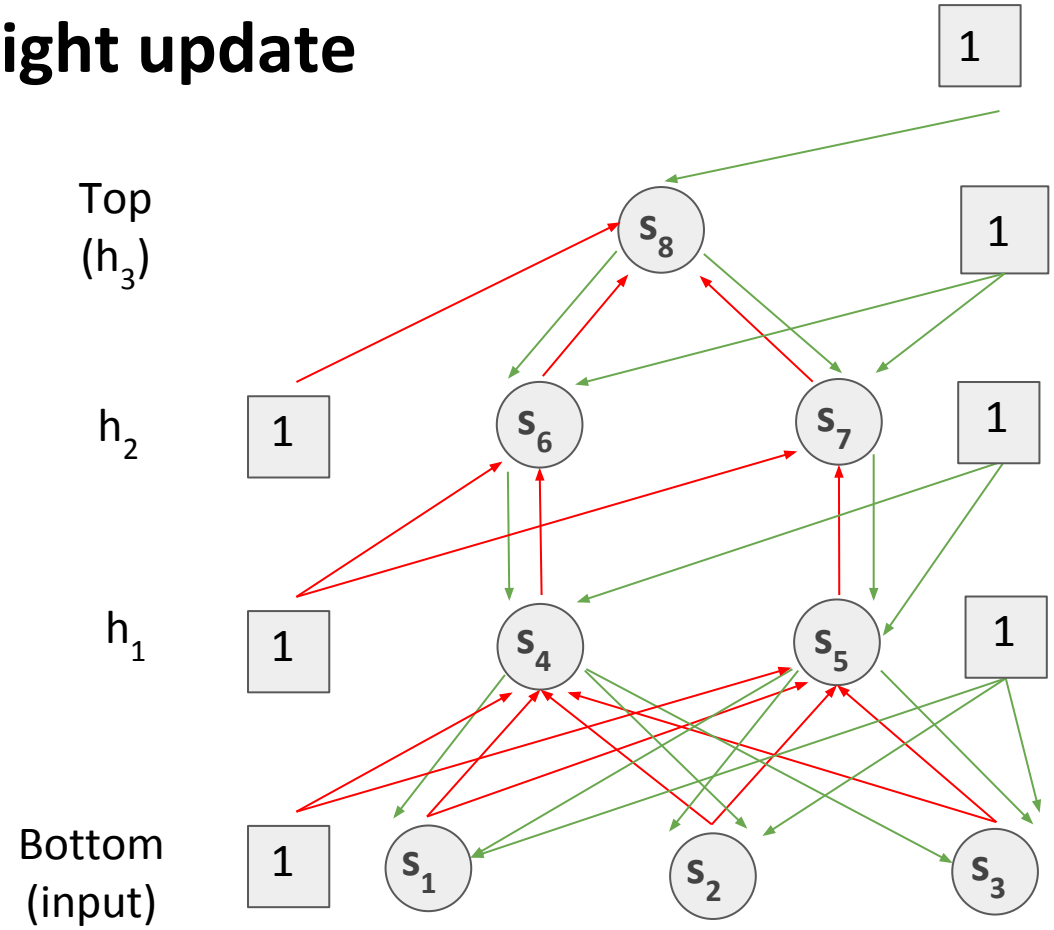
The Wake-Sleep algorithm

1. Input a sample into the network
2. Wake-phase, propagate **bottom-up** using **recognition weights**, update **generative weights**
3. Sleep-phase, update **top-down** using **generative weights**, update **recognition weights**
4. Repeat over all samples

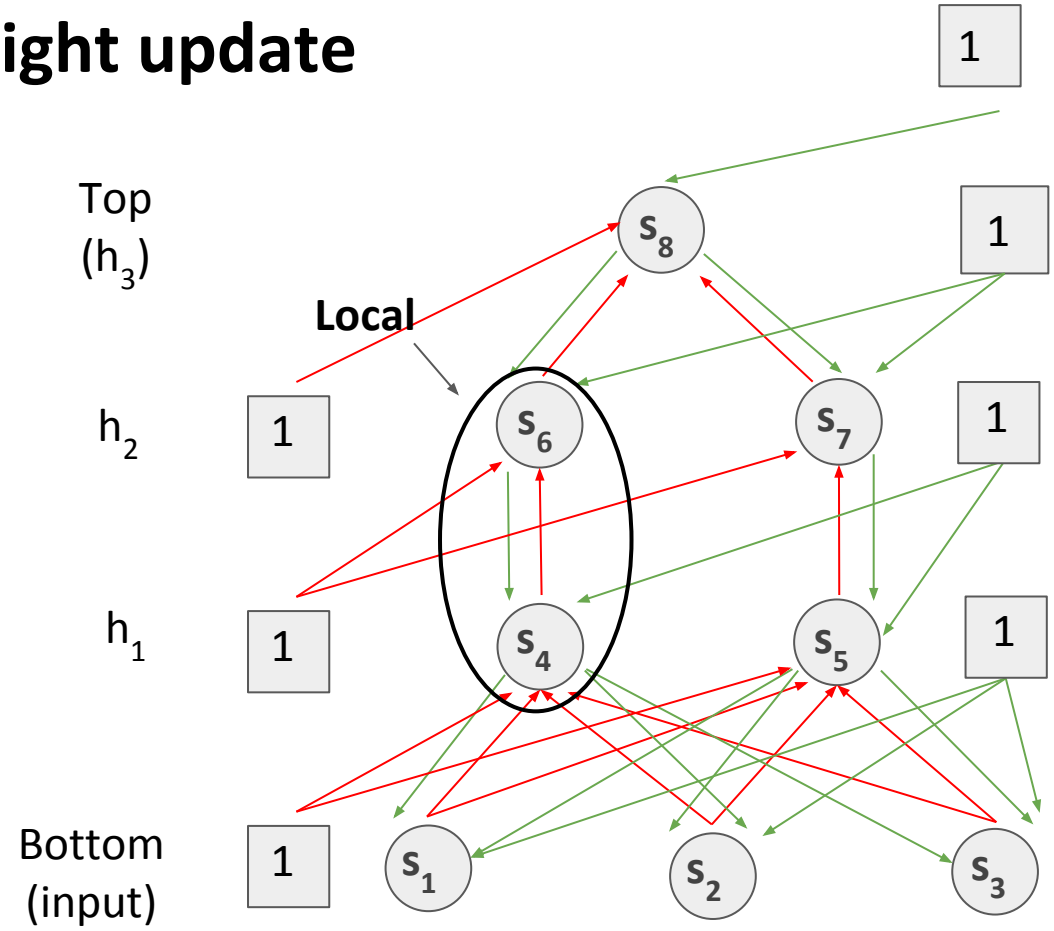
During each phase, follow two simple equations:

1. Stochastic binary unit update
2. **Weight update**

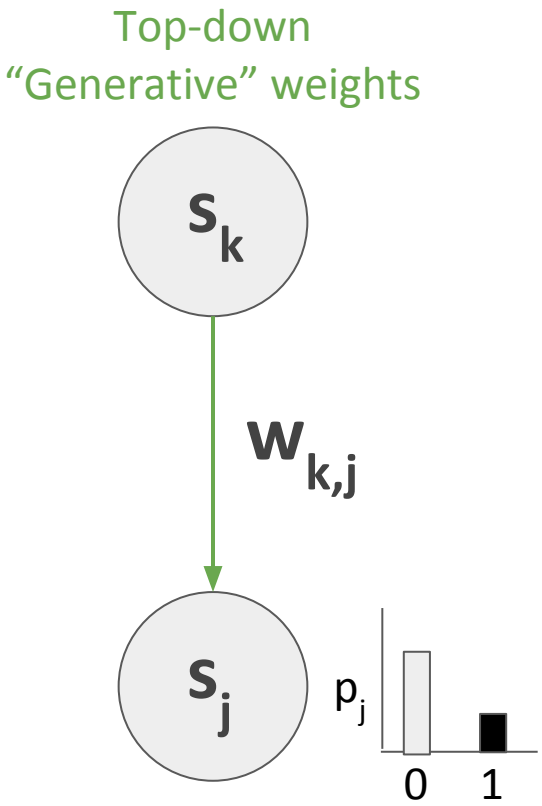
Local weight update



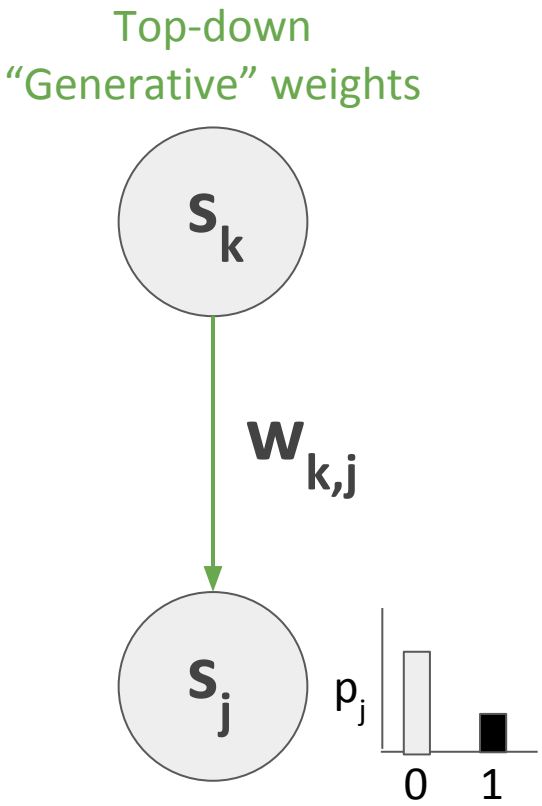
Local weight update



Local weight update

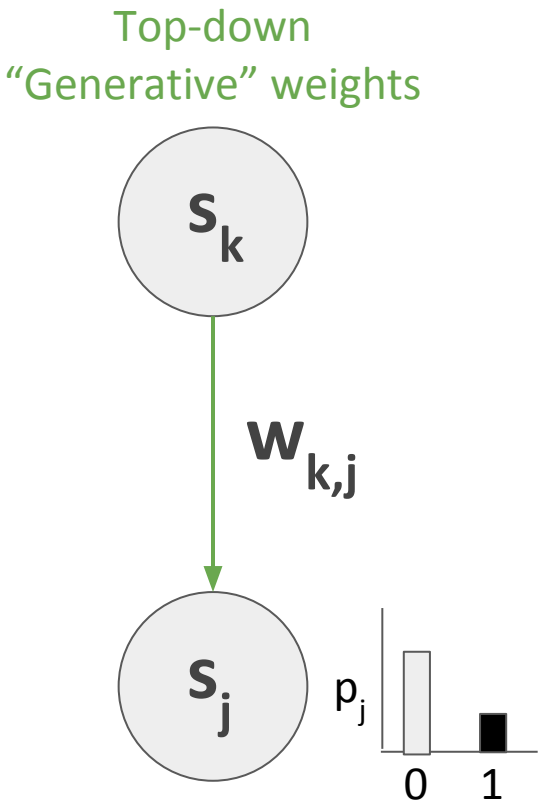


Local weight update



$$w_{k,j} = w_{k,j} + \Delta w_{k,j}$$

Local weight update

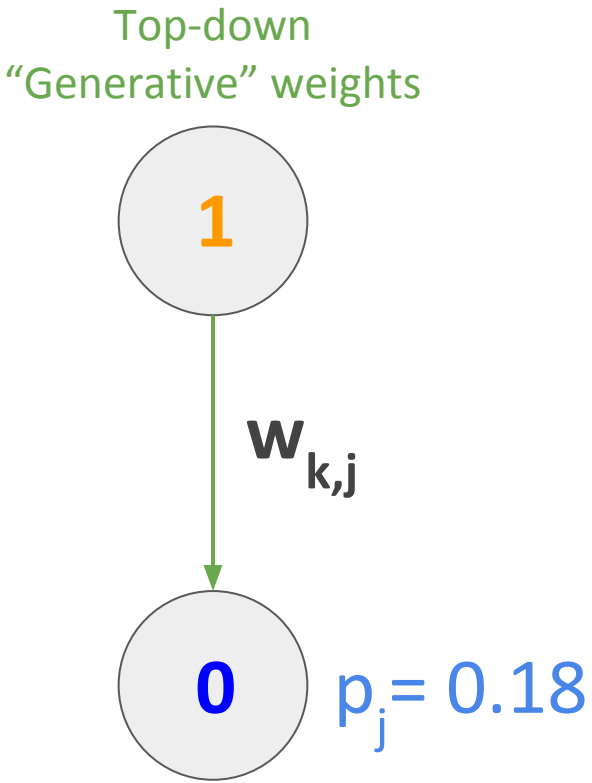


$$w_{k,j} = w_{k,j} + \Delta w_{k,j}$$

$$\Delta w_{k,j} = \varepsilon s_k (s_j - p_j)$$

ε = learning rate

Local weight update



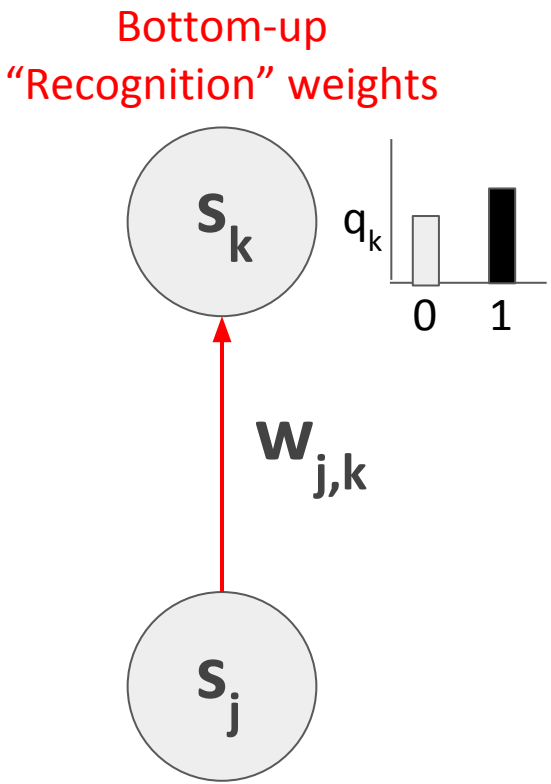
$$\varepsilon = 0.01$$

$$\Delta w_{k,j} = \varepsilon s_k (s_j - p_j)$$

$$\Delta w_{k,j} = 0.01 * 1 * (0 - 0.18)$$

$$w_{k,j} = w_{k,j} + -0.0018$$

Local weight update

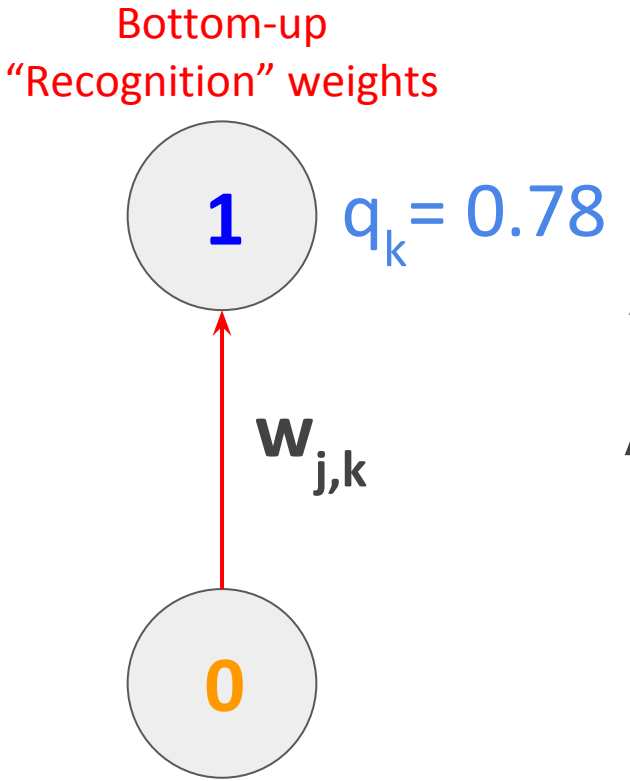


$$w_{j,k} = w_{j,k} + \Delta w_{j,k}$$

$$\Delta w_{j,k} = \varepsilon s_j (s_k - q_k)$$

ε = learning rate

Local weight update



$\varepsilon = 0.01$

$\Delta w_{j,k} = \varepsilon s_j (s_k - q_k)$

$\Delta w_{j,k} = 0.01 * 0 * (1 - 0.78)$

$w_{j,k} = w_{j,k} + 0$

The Wake-Sleep algorithm

1. Input a sample into the network
2. Wake-phase, propagate **bottom-up** using **recognition weights**, update **generative weights**
3. Sleep-phase, update **top-down** using **generative weights**, update **recognition weights**
4. Repeat over all samples

During each phase, follow two simple equations:

1. Stochastic binary unit update
2. Weight update

The Wake-Sleep algorithm

1. **Input a sample into the network**
2. **Wake-phase, propagate bottom-up using recognition weights, update generative weights**
3. Sleep-phase, update **top-down** using **generative weights**, update **recognition weights**
4. Repeat over all samples

During each phase, follow two simple equations:

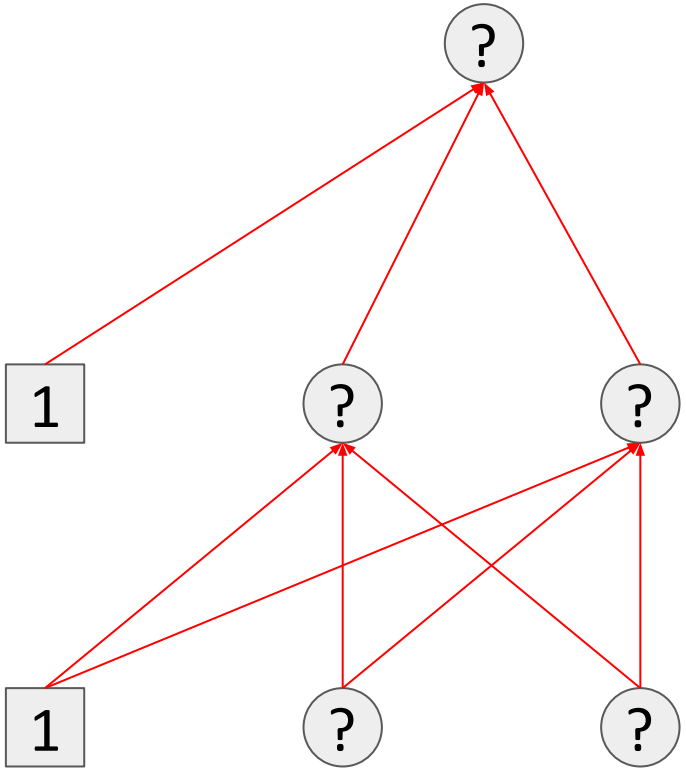
1. Stochastic binary unit update
2. Weight update

The Wake phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



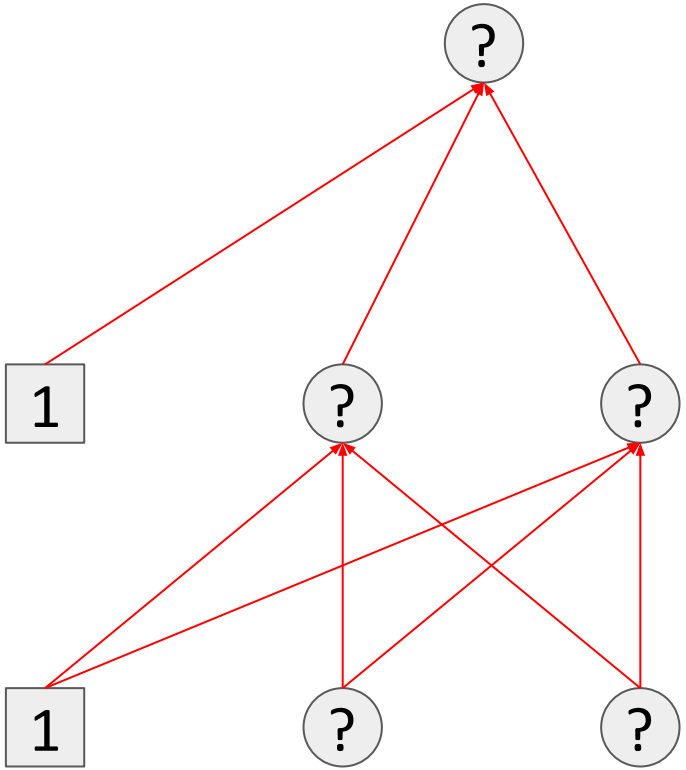
The Wake phase

1. **Initiate all weights and biases with random values.**

Layer K (s_k hidden, top)

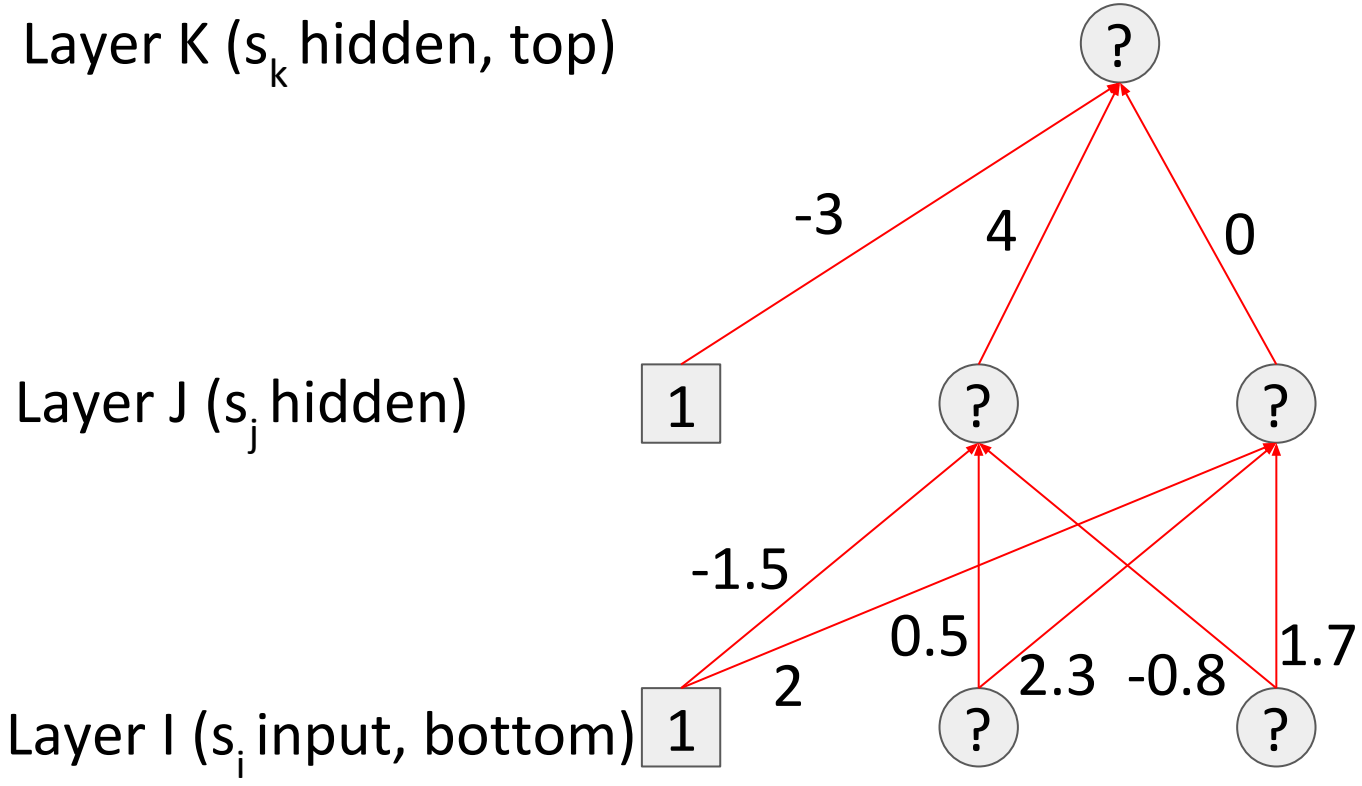
Layer J (s_j hidden)

Layer I (s_i input, bottom)



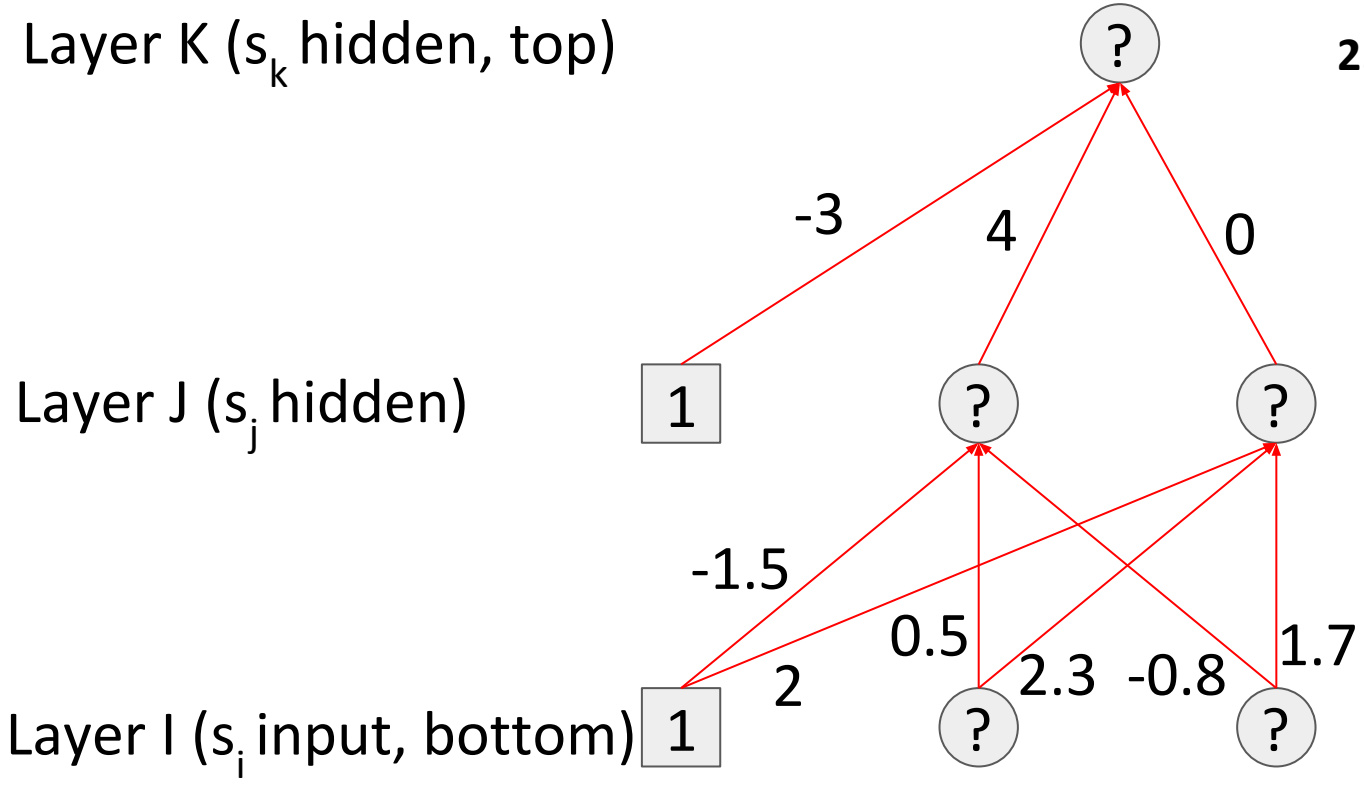
The Wake phase

1. **Initiate all weights and biases with random values.**



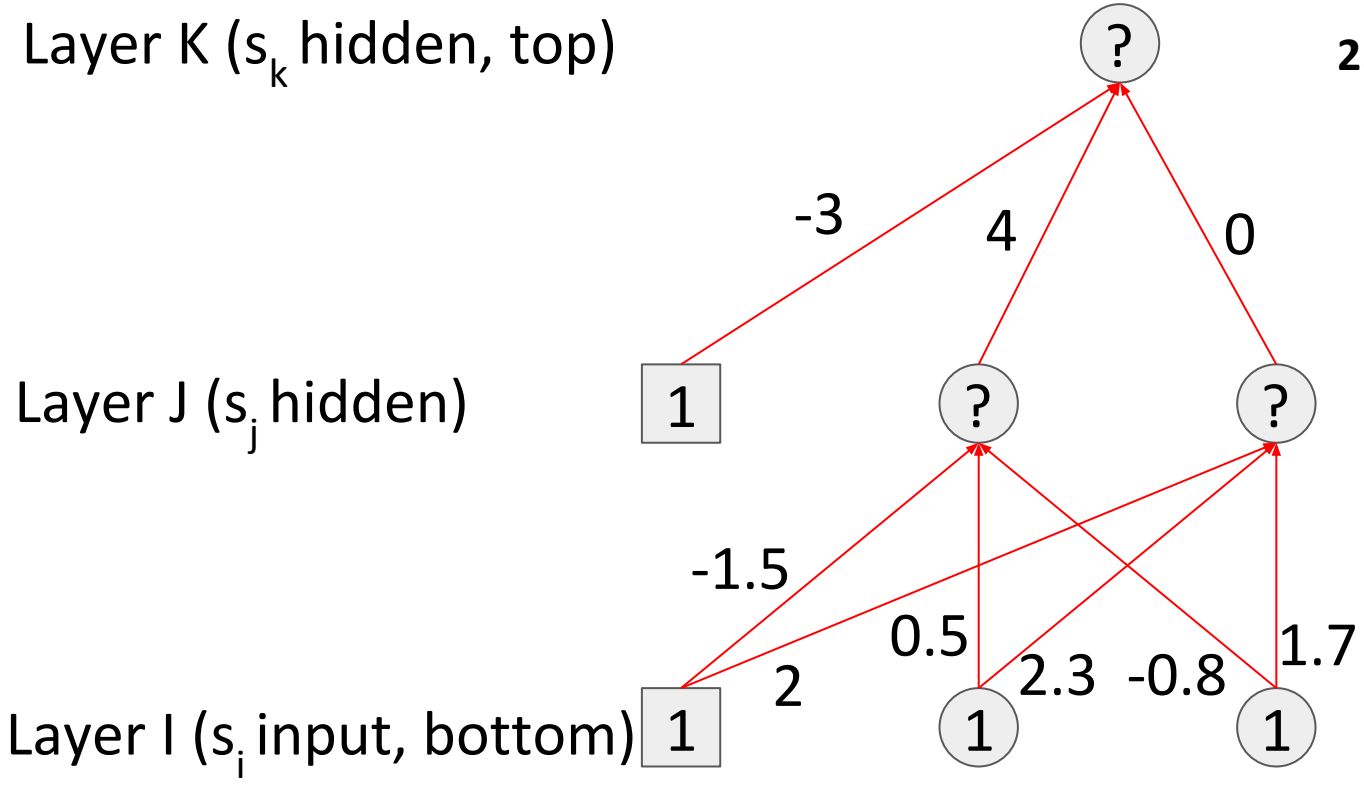
The Wake phase

1. Initiate all weights and biases with random values.
2. **Place input in the bottom layer.**



The Wake phase

1. Initiate all weights and biases with random values.
2. **Place input in the bottom layer.**

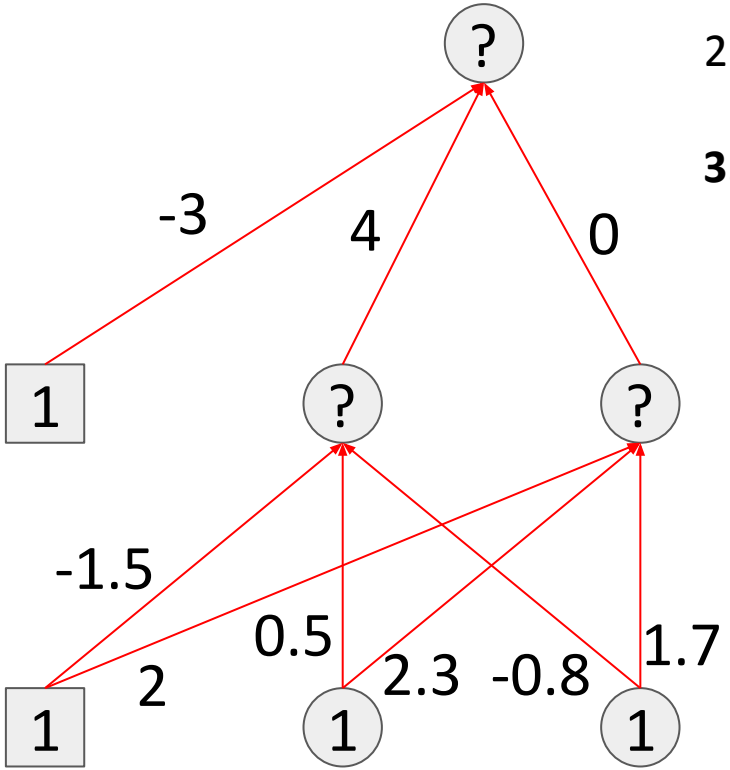


The Wake phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

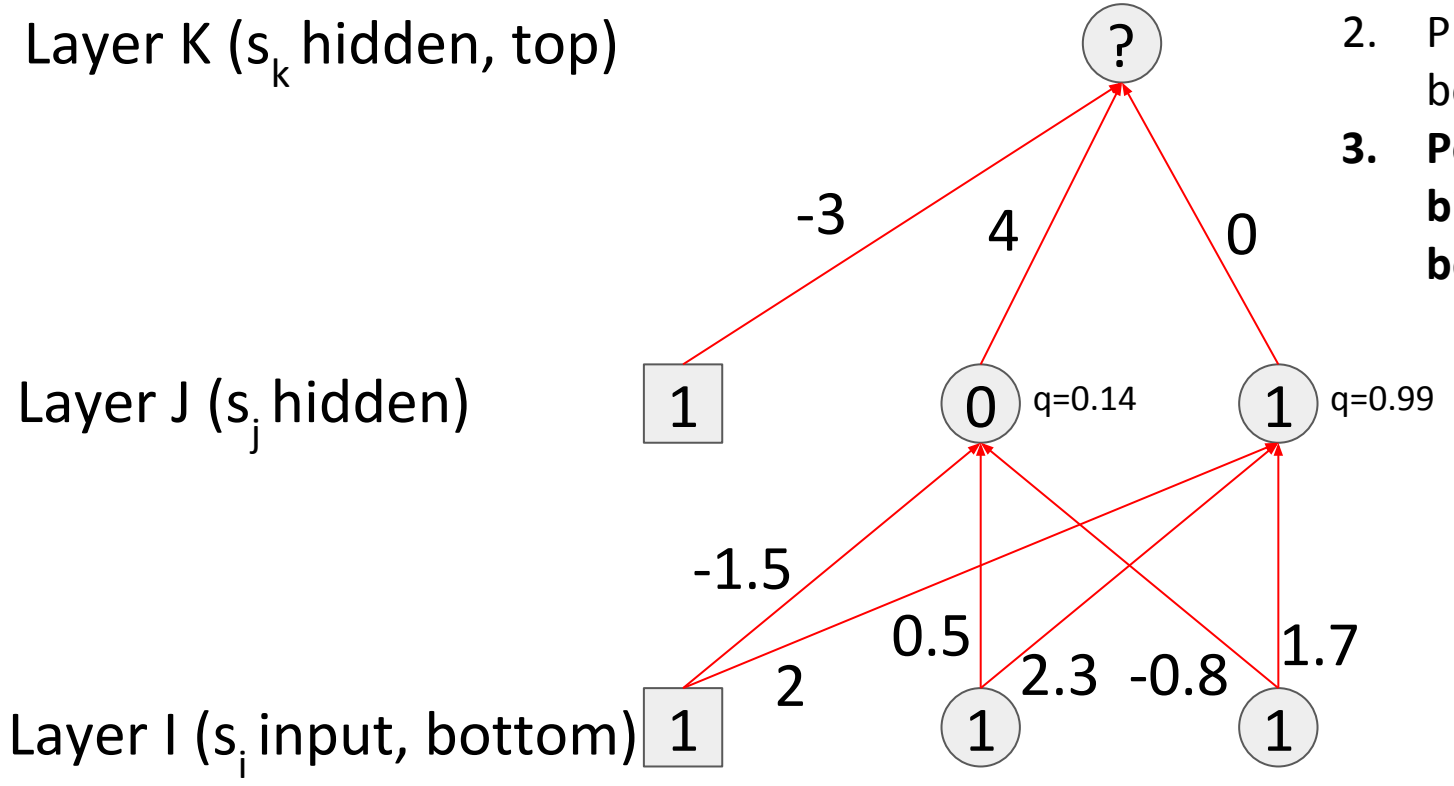
Layer I (s_i input, bottom)



1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. **Perform stochastic binary update**

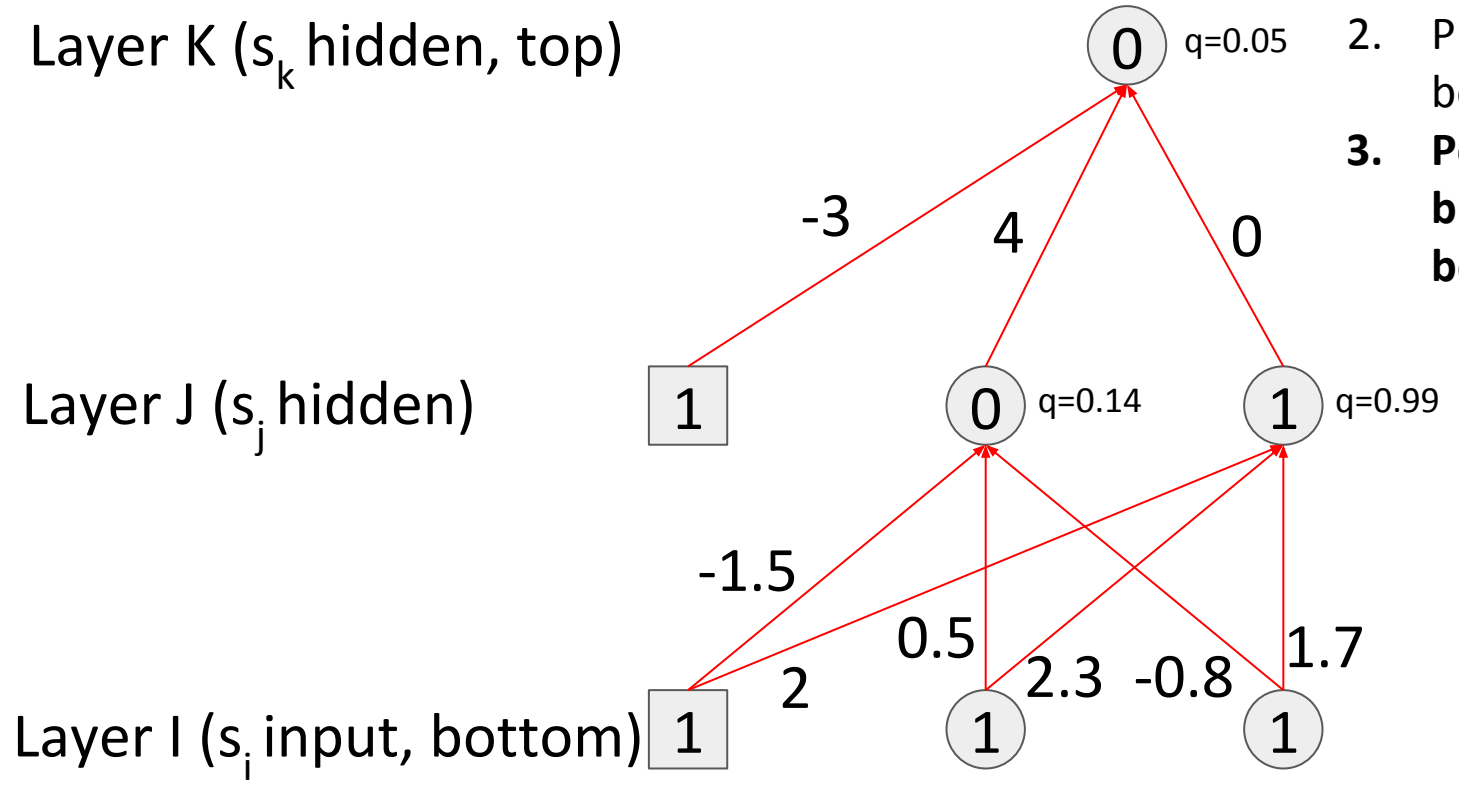
The Wake phase

1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. **Perform stochastic binary update bottom-up.**



The Wake phase

1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. **Perform stochastic binary update bottom-up.**

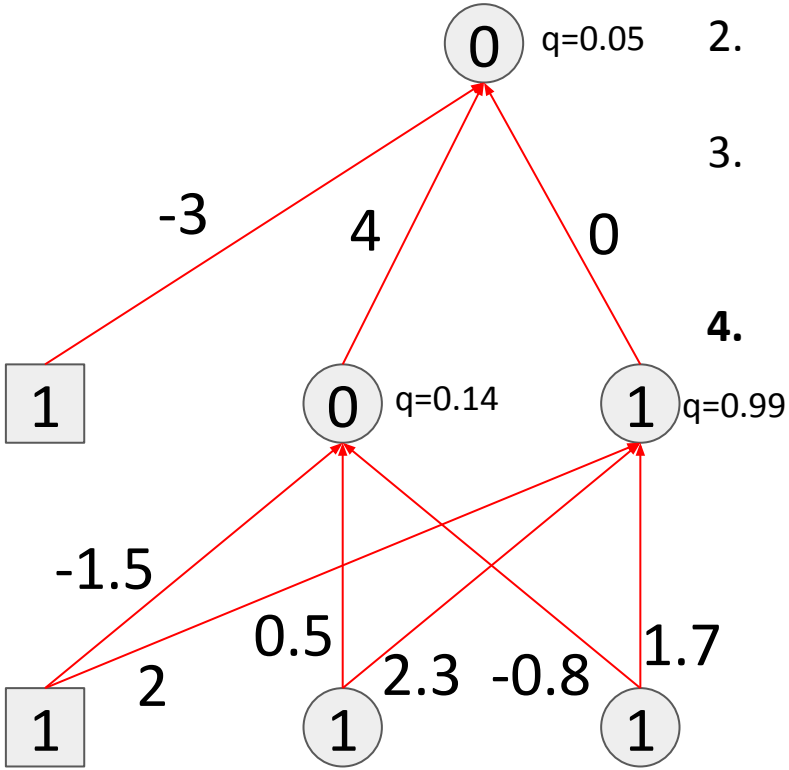


The Wake phase

Layer K (s_k hidden, top)

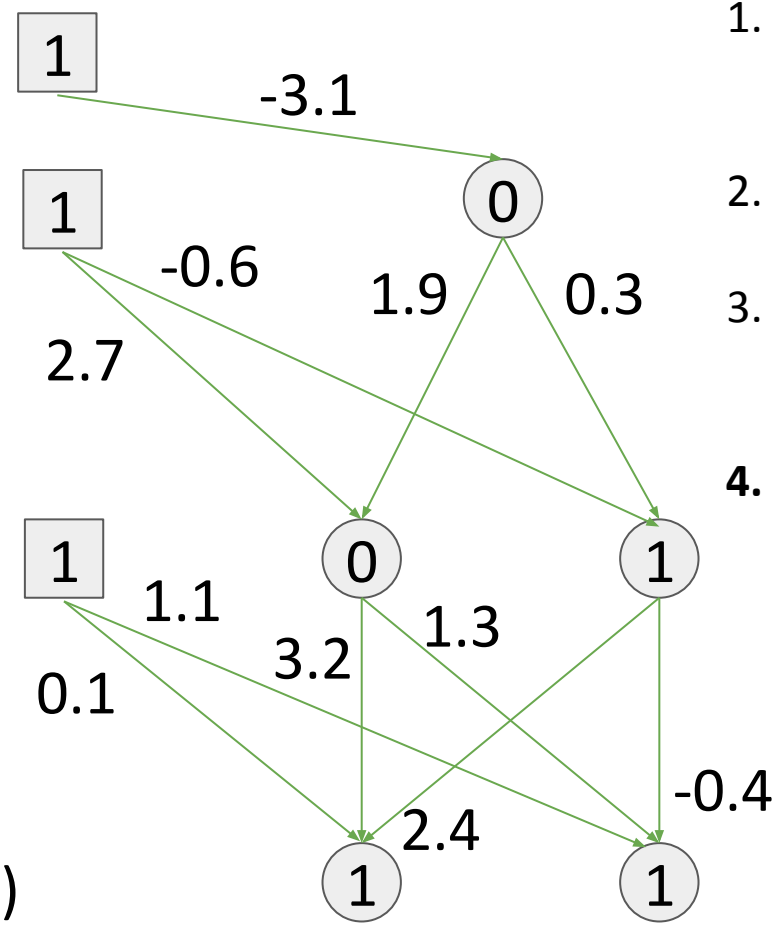
Layer J (s_j hidden)

Layer I (s_i input, bottom)



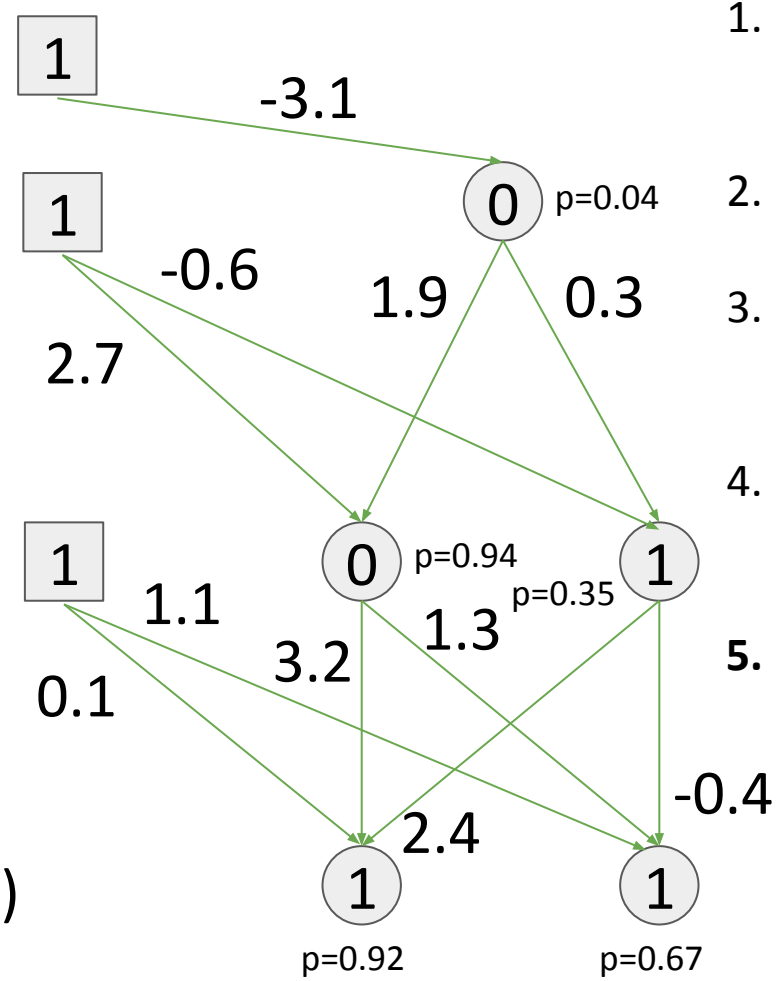
1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. Perform stochastic binary update bottom-up.
4. **Turn off recognition weights and activate generative weights.**

The Wake phase



1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. Perform stochastic binary update bottom-up.
4. **Turn off recognition weights and activate generative weights.**

The Wake phase



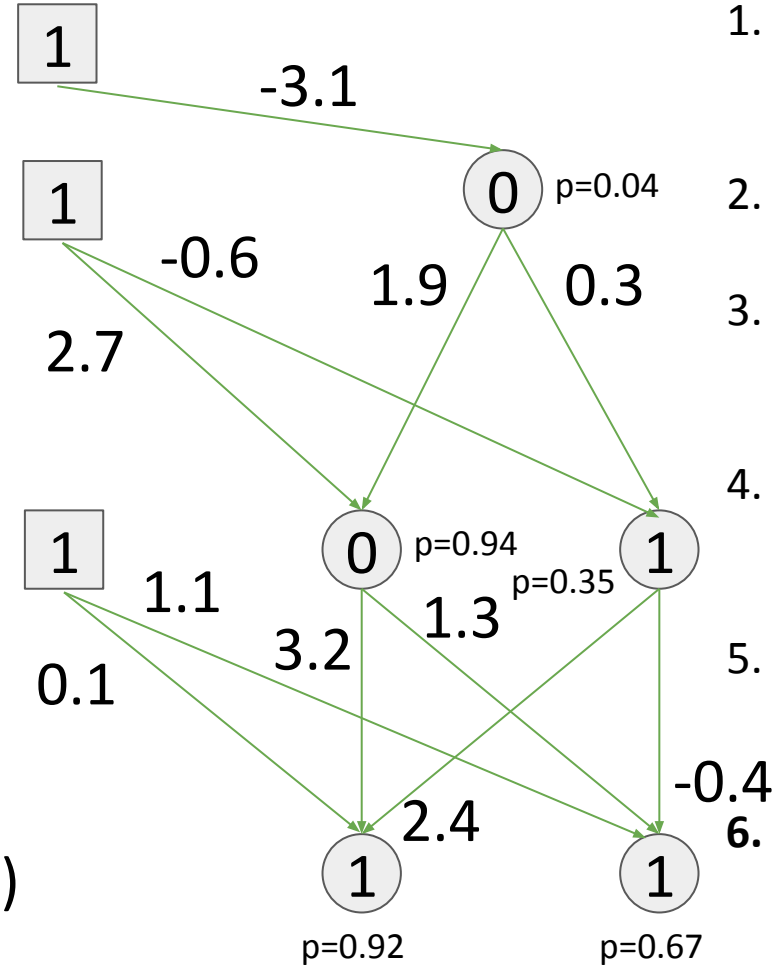
1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. Perform stochastic binary update bottom-up.
4. Turn off recognition weights and activate generative weights.
5. **Calculate stochastic probabilities top-down, without unit update.**

The Wake phase

Layer K (s_k hidden, top)

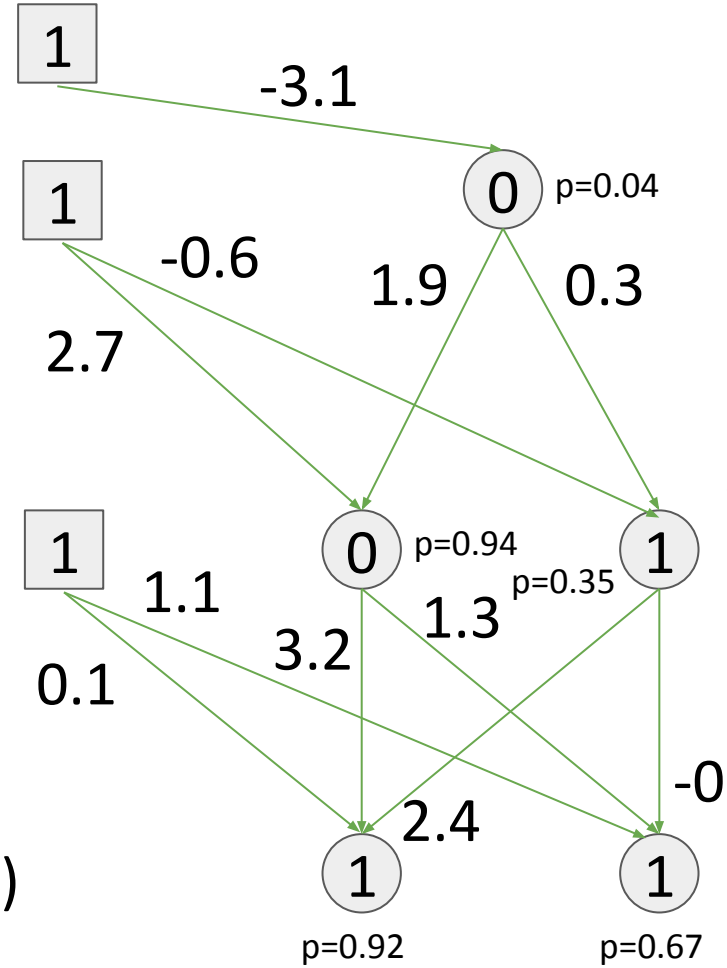
Layer J (s_j hidden)

Layer I (s_i input, bottom)



1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. Perform stochastic binary update bottom-up.
4. Turn off recognition weights and activate generative weights.
5. Calculate stochastic probabilities top-down, without unit update.
6. **Update generative weights with local update rule.**

The Wake phase



1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. Perform stochastic binary update bottom-up.
4. Turn off recognition weights and activate generative weights.
5. Calculate stochastic probabilities top-down, without unit update.
6. **Update generative weights with local update rule.**

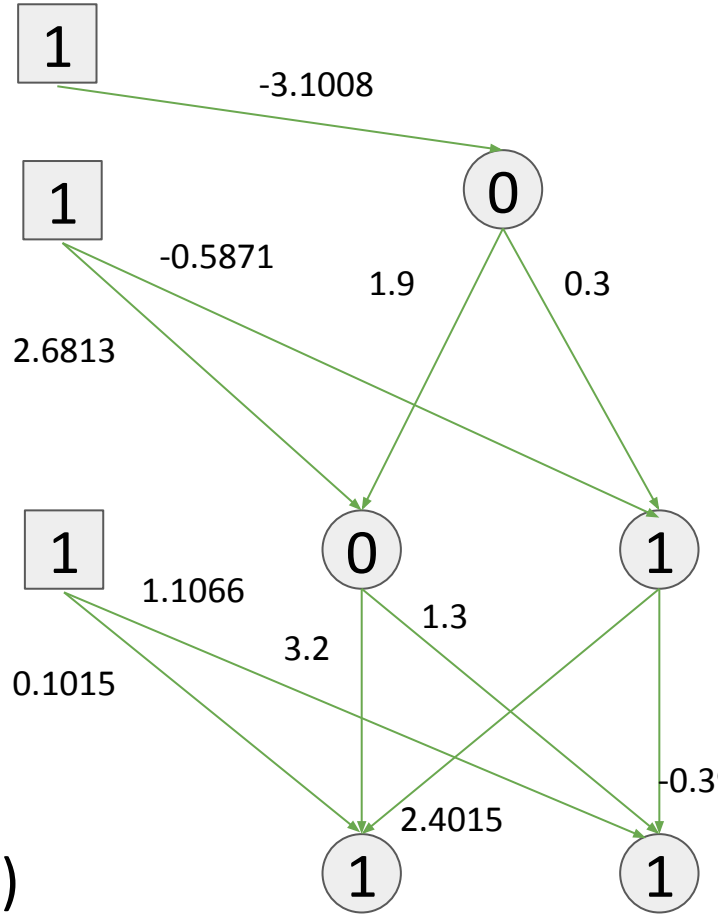
$$\Delta w_{k,j} = \varepsilon s_k (s_j - p_j)$$

The Wake phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



1. Initiate all weights and biases with random values.
2. Place input in the bottom layer.
3. Perform stochastic binary update bottom-up.
4. Turn off recognition weights and activate generative weights.
5. Calculate stochastic probabilities top-down, without unit update.
6. **Update generative weights with local update rule.**

$$\Delta w_{k,j} = \varepsilon s_k (s_j - p_j)$$

The Wake-Sleep algorithm

1. Input a sample into the network
2. Wake-phase, propagate bottom-up using recognition weights, update generative weights
3. **Sleep-phase, update top-down using generative weights, update recognition weights**
4. Repeat over all samples

During each phase, follow two simple equations:

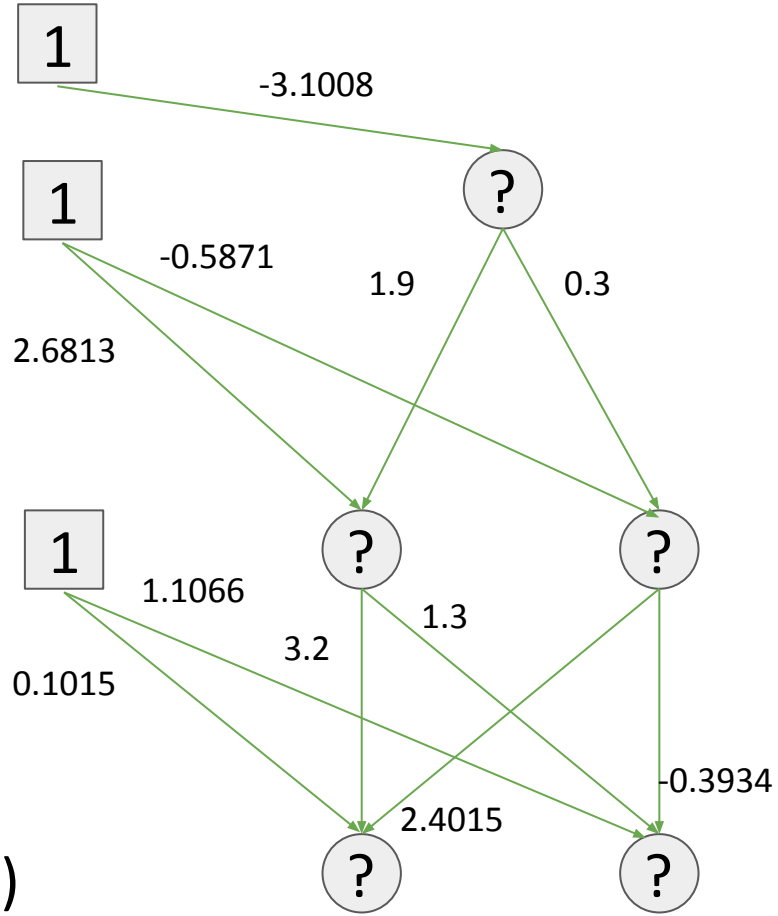
1. Stochastic binary unit update
2. Weight update

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



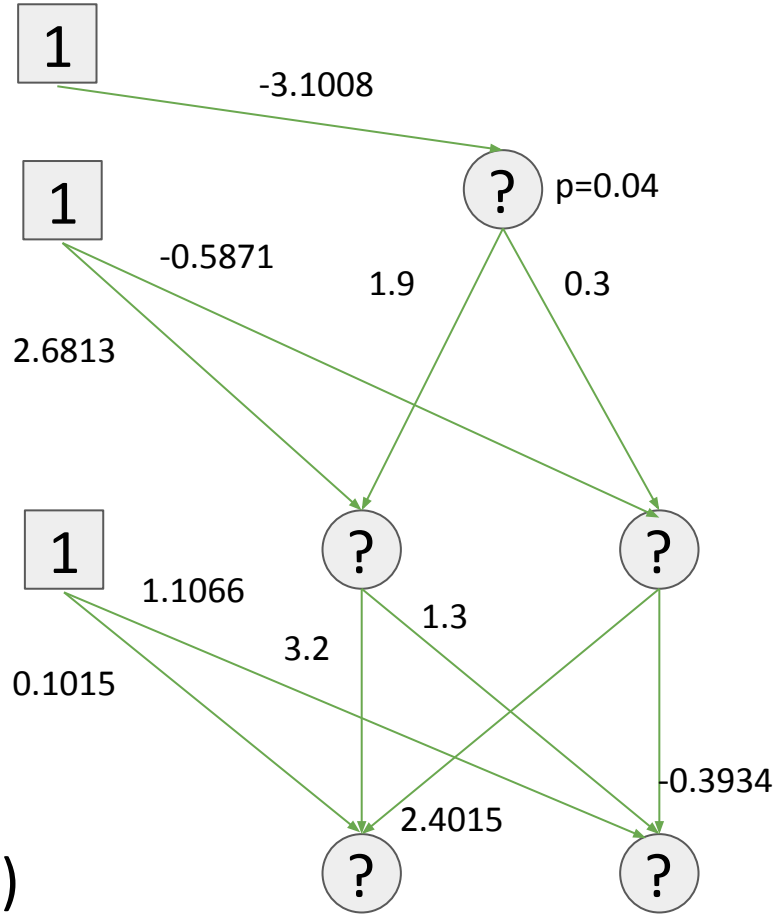
The Sleep phase

1. **Generate a "fantasy" sample.**

Layer K (s_k hidden, top)

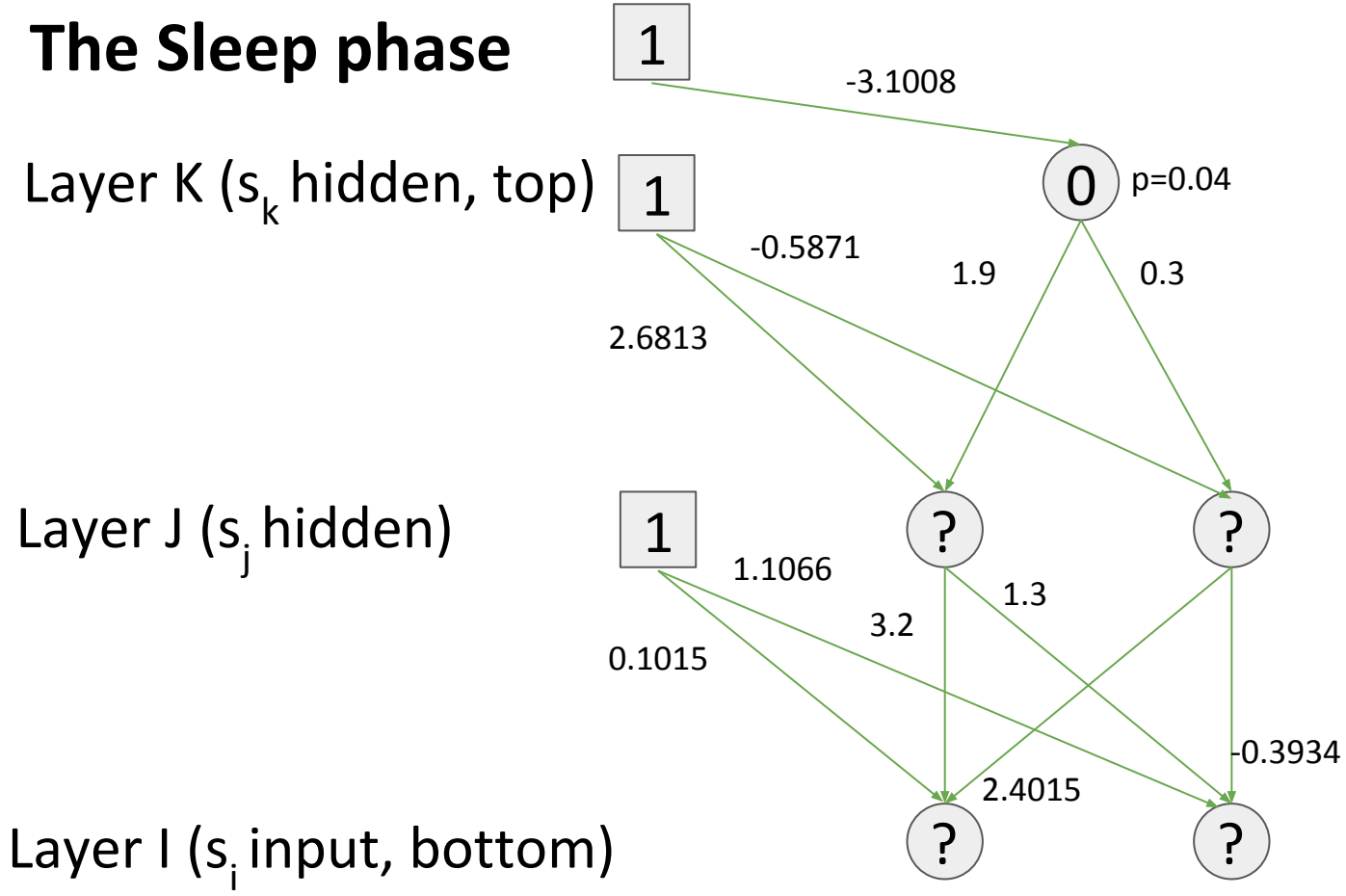
Layer J (s_j hidden)

Layer I (s_i input, bottom)



The Sleep phase

1. **Generate a "fantasy" sample.**

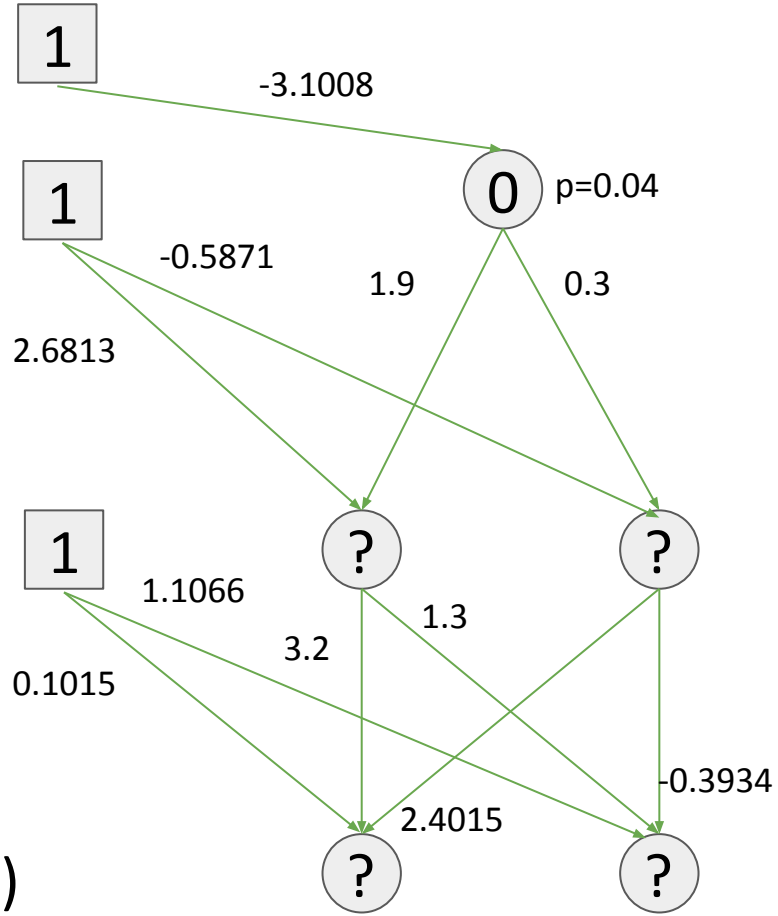


The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



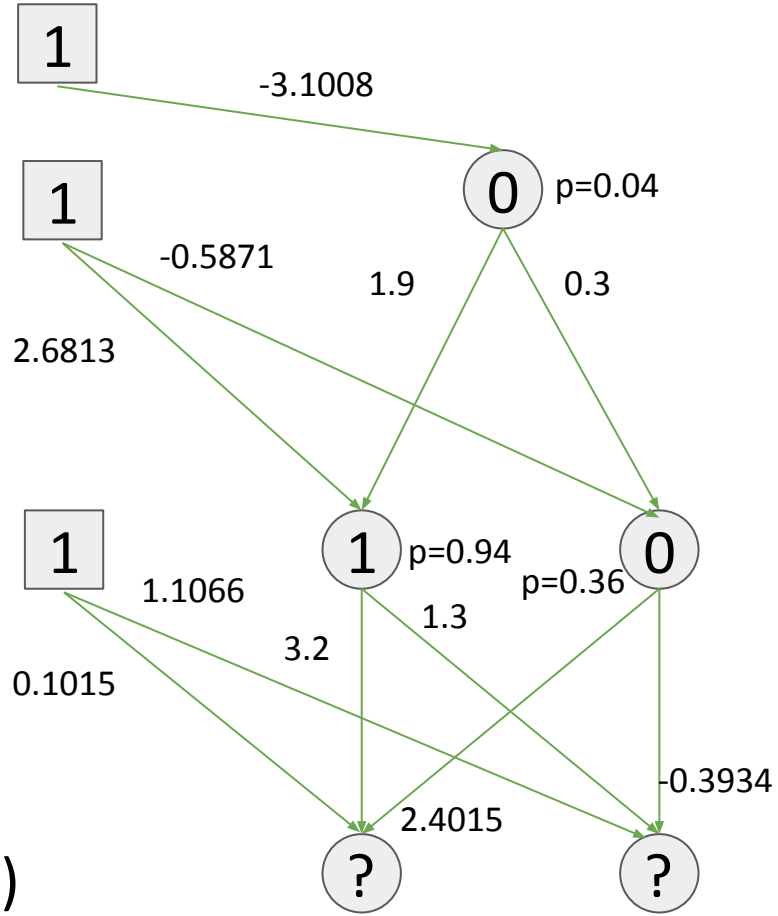
1. Generate a "fantasy" sample.
2. **Perform stochastic binary update top-down.**

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



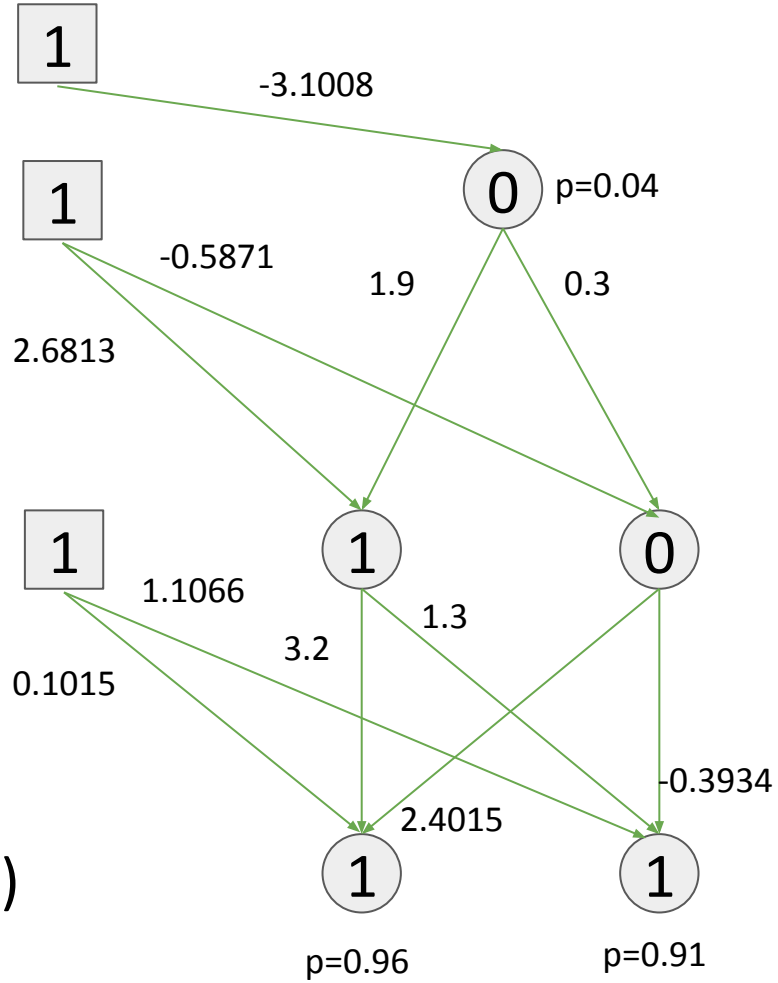
1. Generate a "fantasy" sample.
2. **Perform stochastic binary update top-down.**

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



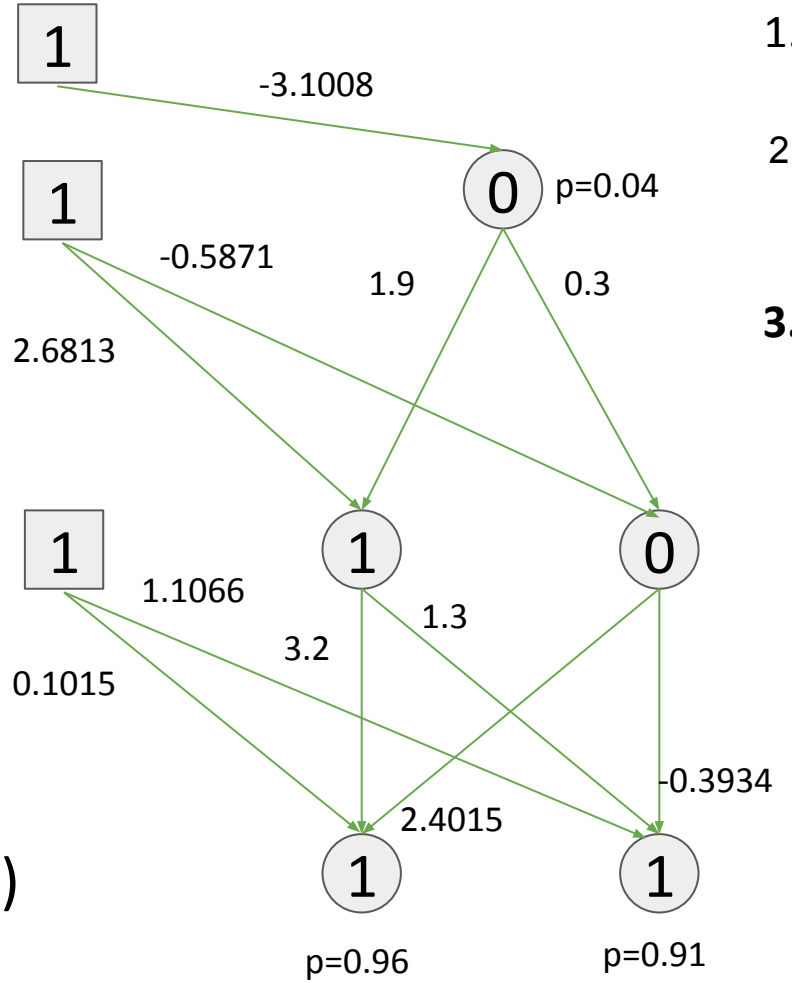
1. Generate a "fantasy" sample.
2. **Perform stochastic binary update top-down.**

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



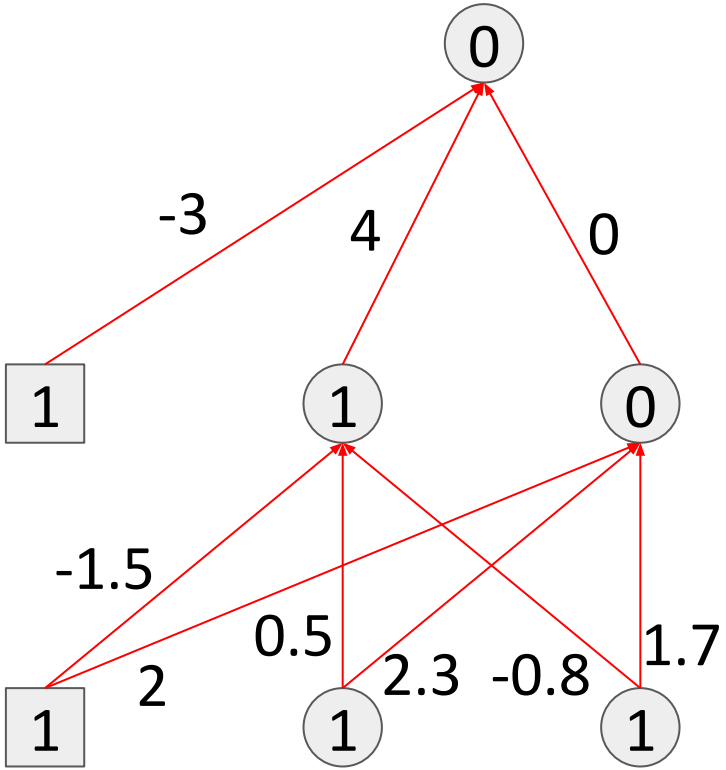
1. Generate a “fantasy” sample.
2. Perform stochastic binary update top-down.
3. **Turn off generative weights and activate recognition weights.**

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



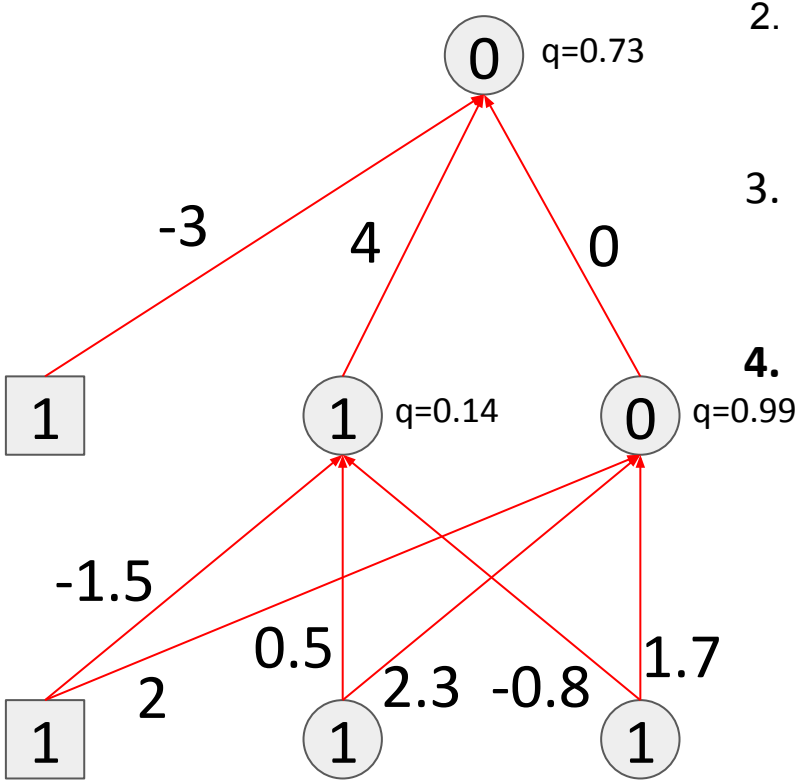
1. Generate a “fantasy” sample.
2. Perform stochastic binary update top-down.
3. **Turn off generative weights and activate recognition weights.**

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



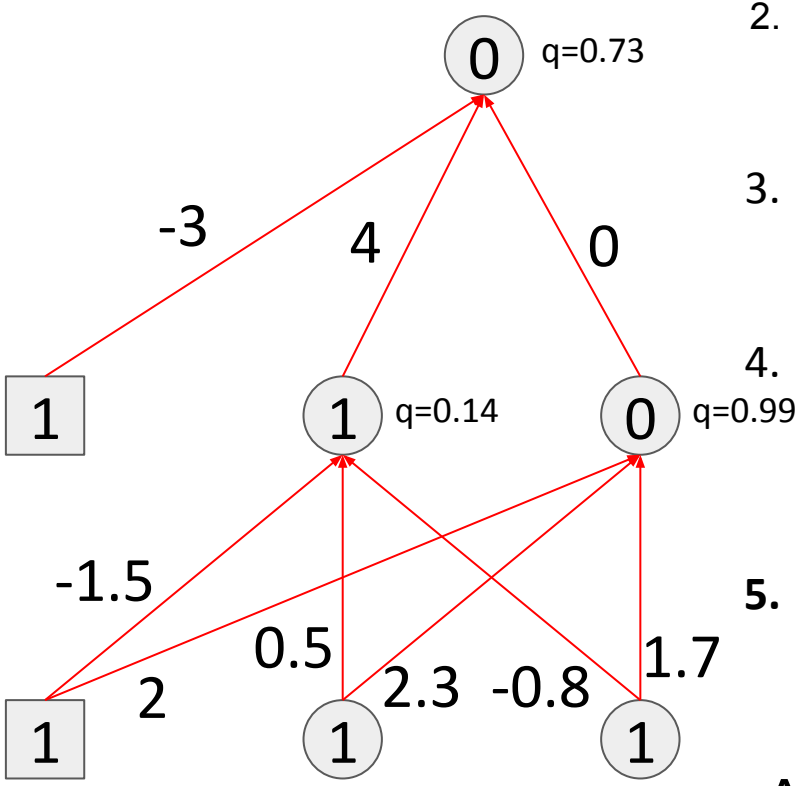
1. Generate a “fantasy” sample.
2. Perform stochastic binary update top-down.
3. Turn off generative weights and activate recognition weights.
4. **Calculate stochastic probabilities bottom-up, without unit update.**

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



1. Generate a “fantasy” sample.
2. Perform stochastic binary update top-down.
3. Turn off generative weights and activate recognition weights.
4. Calculate stochastic probabilities bottom-up, without unit update.
5. **Update recognition weights with local update rule.**

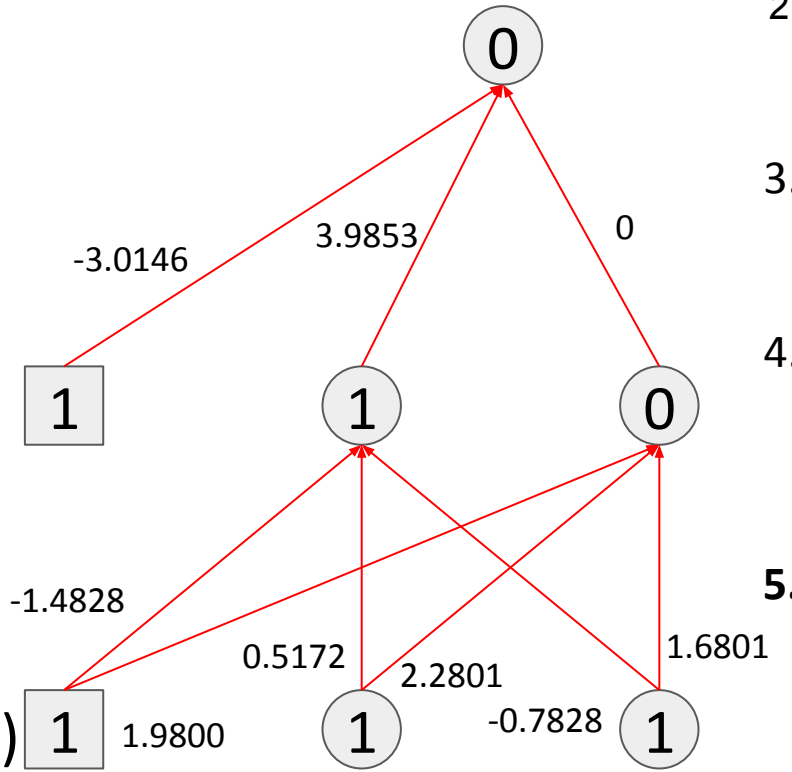
$$\Delta w_{j,k} = \varepsilon s_j (s_k - q_k)$$

The Sleep phase

Layer K (s_k hidden, top)

Layer J (s_j hidden)

Layer I (s_i input, bottom)



1. Generate a “fantasy” sample.
2. Perform stochastic binary update top-down.
3. Turn off generative weights and activate recognition weights.
4. Calculate stochastic probabilities bottom-up, without unit update.
5. **Update recognition weights with local update rule.**

$$\Delta w_{j,k} = \varepsilon s_j (s_k - q_k)$$

The Wake-Sleep algorithm

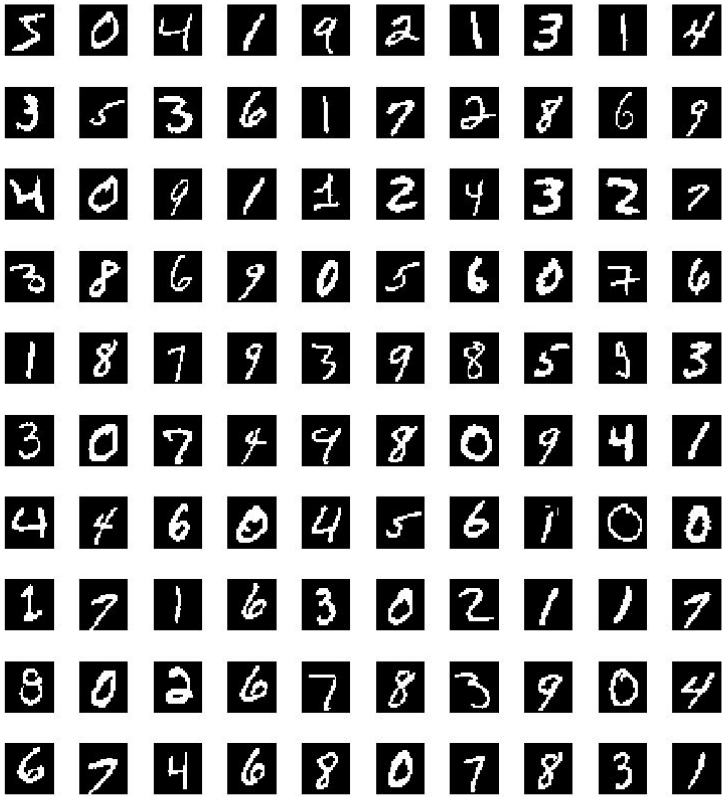
1. Input a sample into the network
2. Wake-phase, propagate bottom-up using recognition weights, update generative weights
3. Sleep-phase, update top-down using generative weights, update recognition weights
4. **Repeat over all samples**

During each phase, follow two simple equations:

1. Stochastic binary unit update
2. Weight update

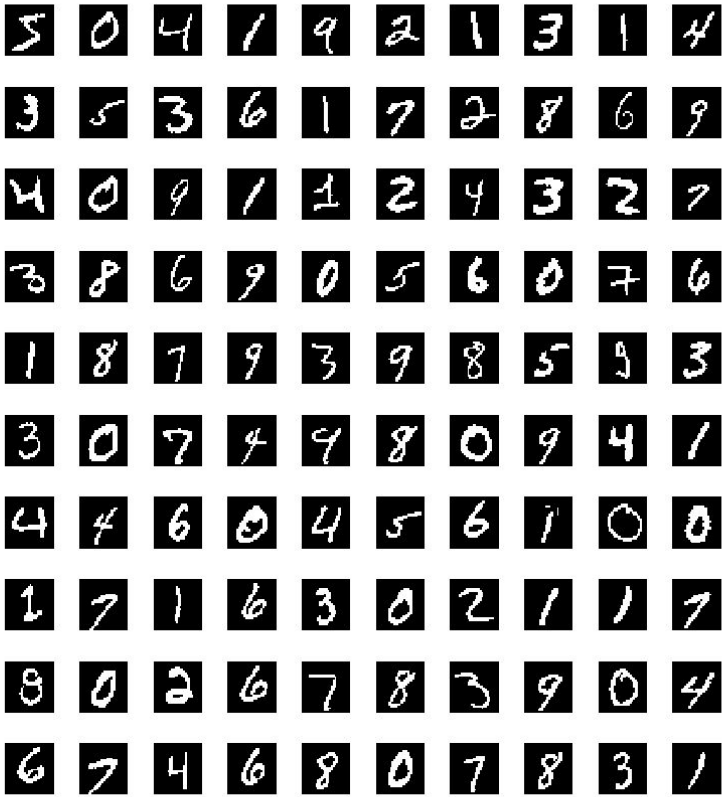
Generative samples from the Helmholtz machine (MNIST)

Real

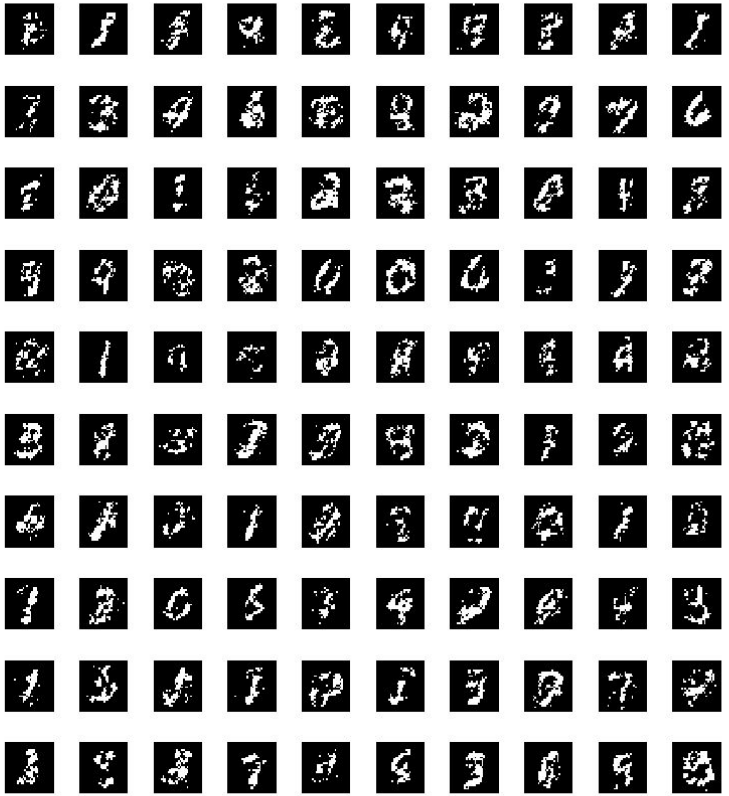


Generative samples from the Helmholtz machine (MNIST)

Real

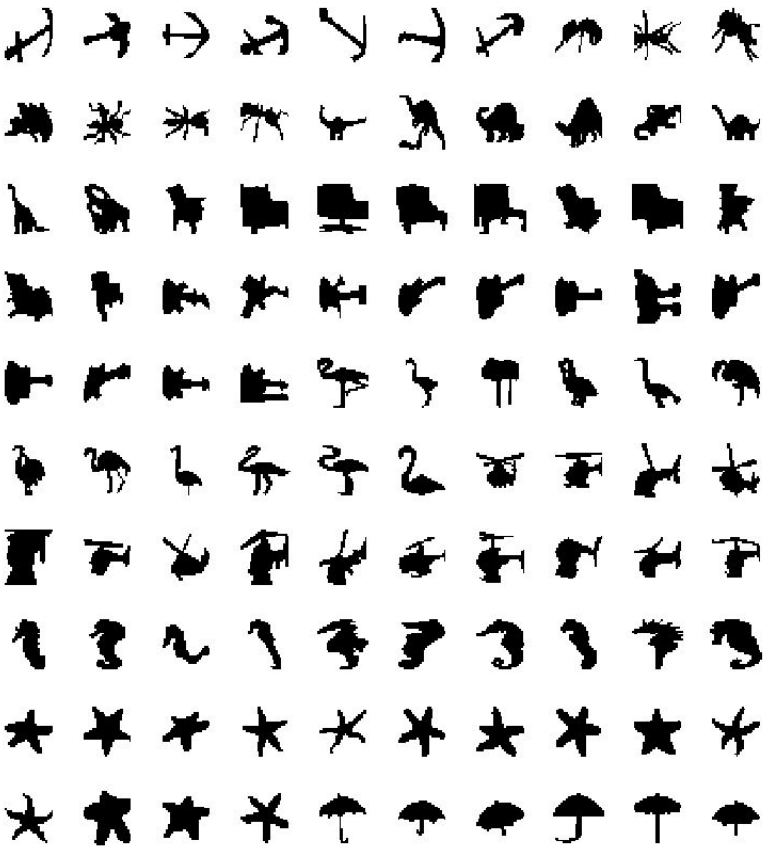


Generated



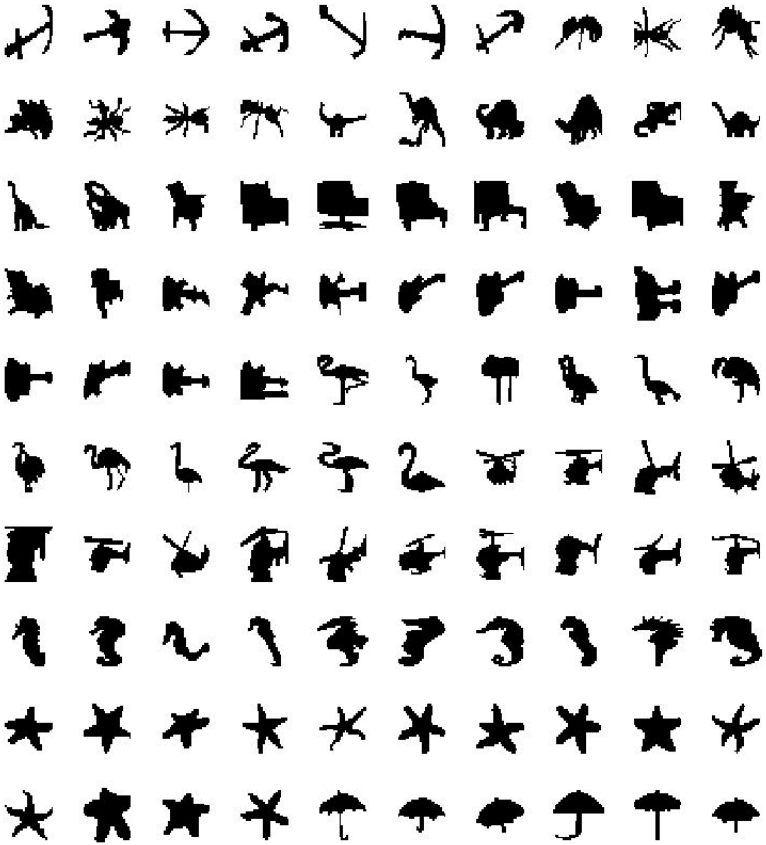
Generative samples from the Helmholtz machine (Caltech 101)

Real

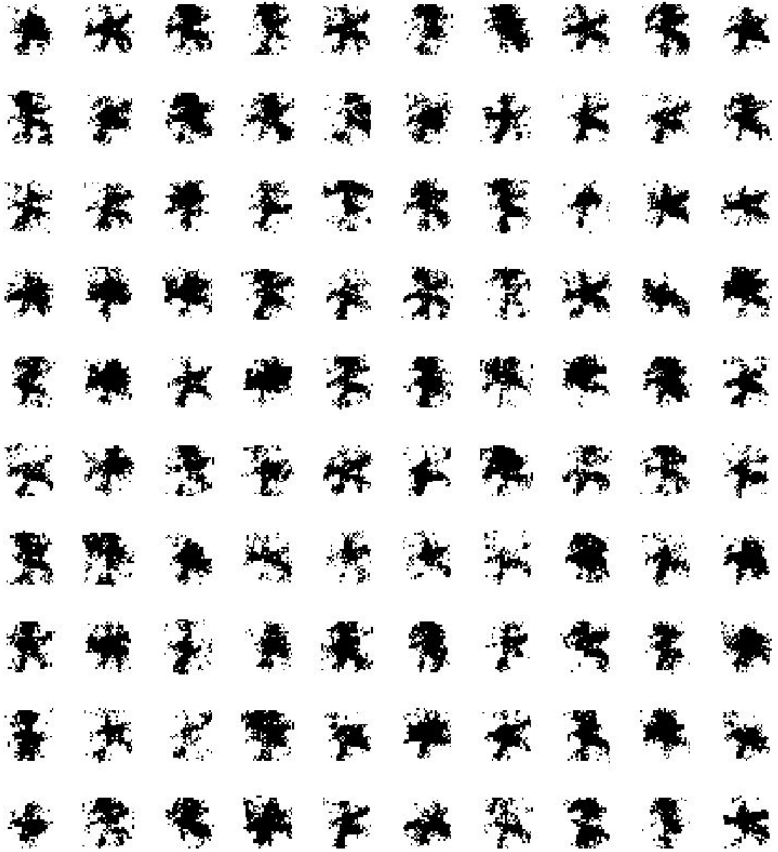


Generative samples from the Helmholtz machine (Caltech 101)

Real



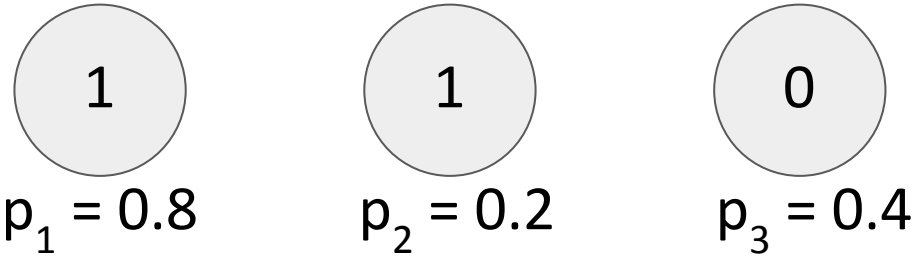
Generated



Why does this work?

- Approximate the true distribution using a more economical representation:
  a. Binary values instead of reals
  b. Assume units in the same layer are conditionally independent from each other (factorial distribution)

Prob(1,1,0) = 0.8 * 0.2 * (1 - 0.4) = 0.384



Neuroscience of wake and sleep

- Wake phase
  - Observe environment and receive unlabeled inputs
- Sleep phase
  - Receive random inputs (“fantasies”) and conform them to fit the representation of your wake input
- Wake phase
  - Continue gathering from the environment and eliminate incorrect fantasies
- Sleep phase
  - Generate more fantasies
- Continue wake-sleep cycle until your fantasies align with your inputs. Now you can generate more fantasies that align with inputs you have never seen.