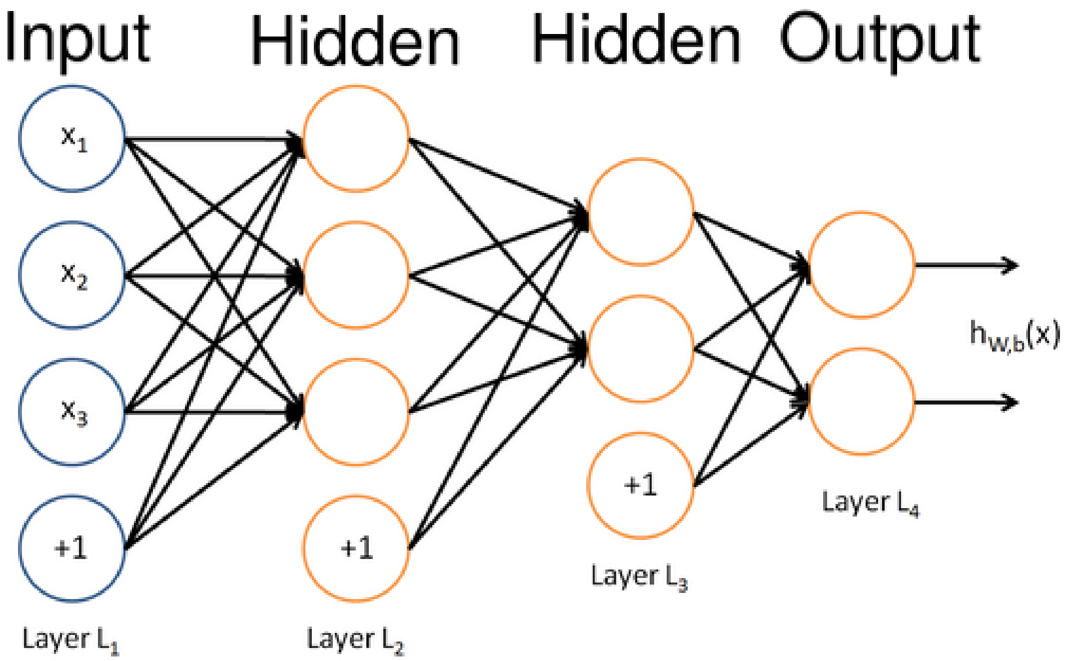
A New Learning Algorithm for Stochastic Feedforward Neural Nets

Yichuan Tang, Ruslan Salakhutdinov –ICML2013

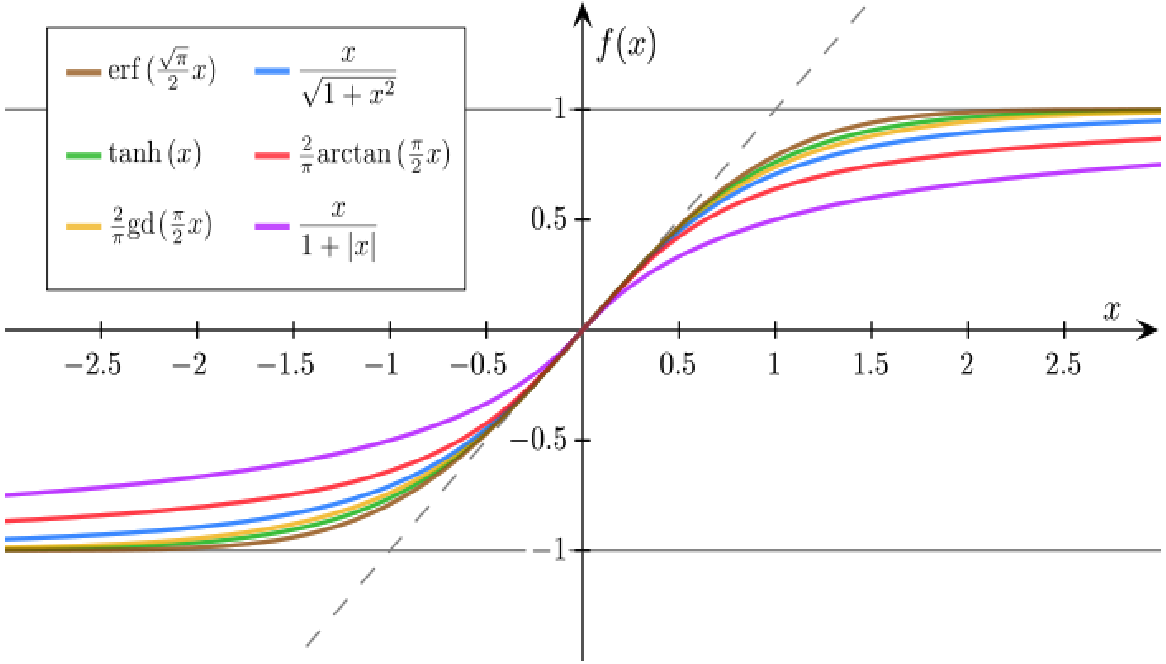
Repoter: Yunfei WANG

Nov.26, 2013

Multilayer Perceptrons(MLPs)



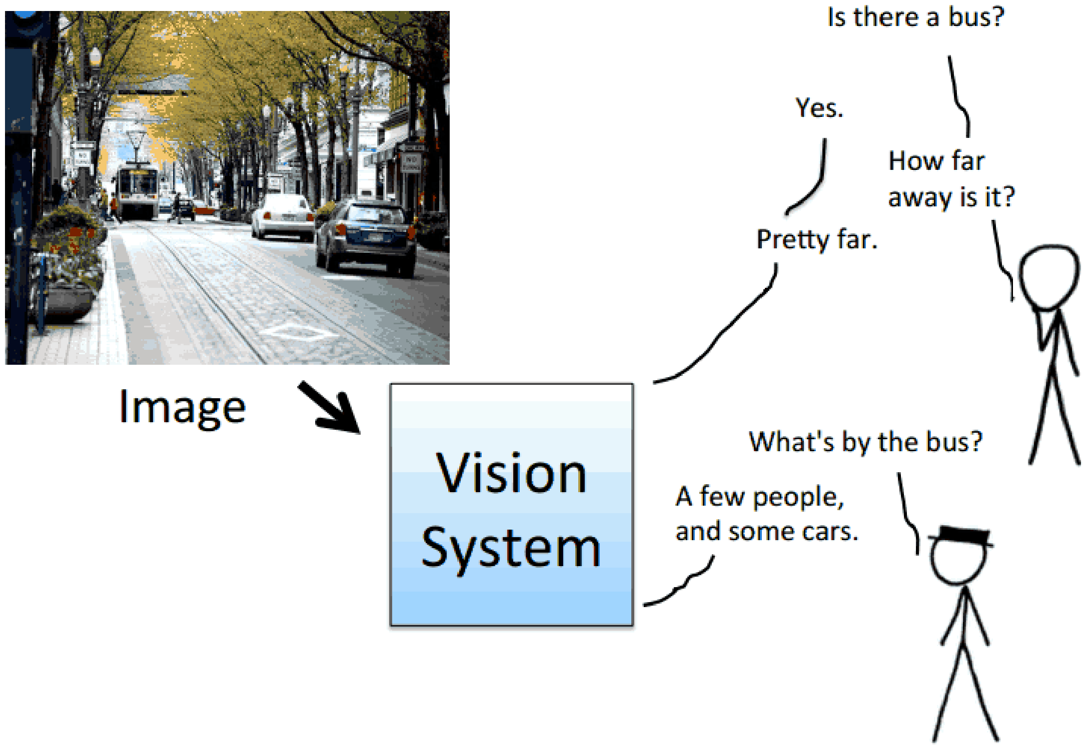
$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$



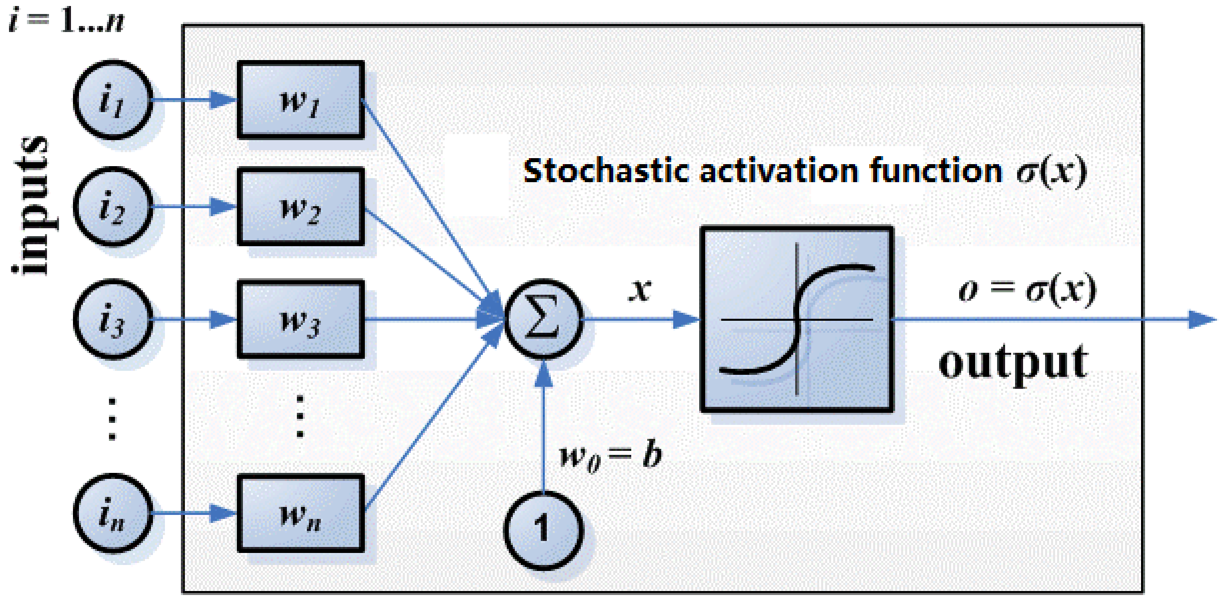
We assume that y has a Gaussian distribution with an x-dependent mean given by the output of MLPs

$$p(y|\mathbf{x}) \sim \mathcal{N}(y|\mu_y, \sigma_y^2)$$
$$\mu_y = \sigma\big(W_2\sigma(W_1\mathbf{x} + bias_1) + bias_2\big)$$

Sigmoid Belief Net (SBN)

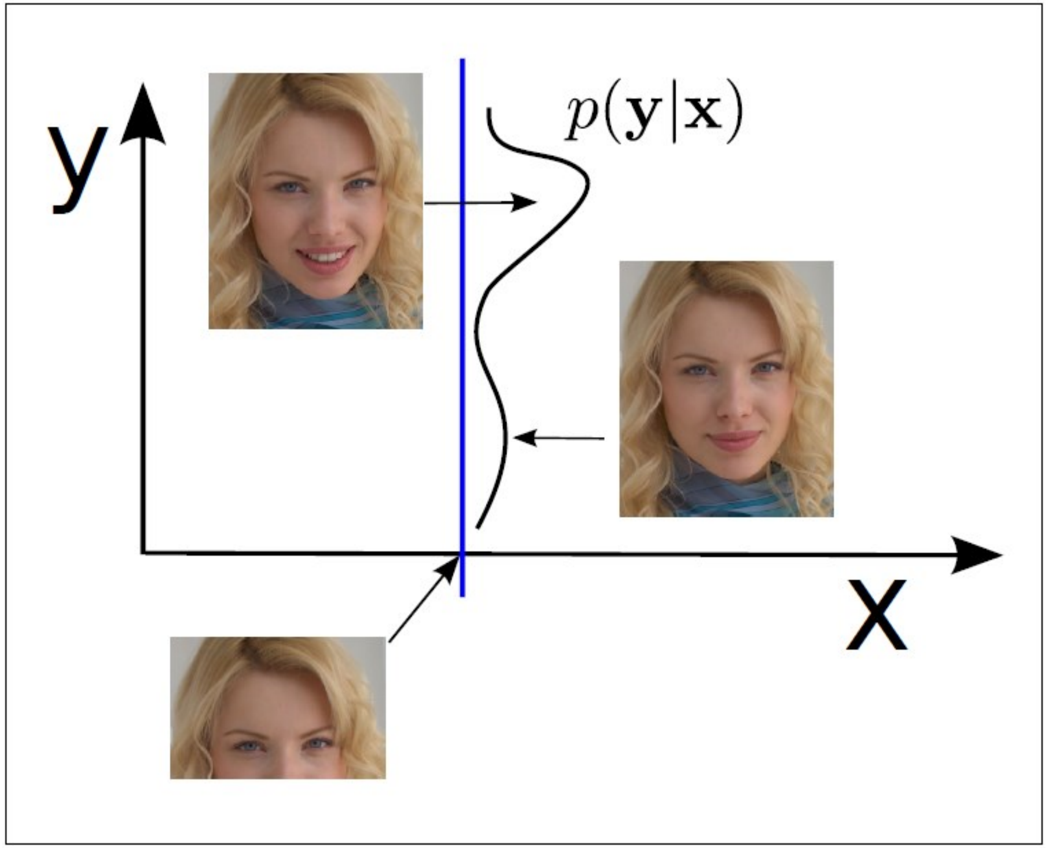


For structured prediction problems, we are interested in modeling a conditional distribution p(Y|X) that is multimodal, namely learning one to many functions from X to Y.



One way to model the multi-modality is making the hidden variables stochastic.
Sigmoid Belief Net (SBN) is a stochastic feedforward neural network with binary hidden, input, and output variables.
Given the same X, different hidden configurations leads to different output values of Y .

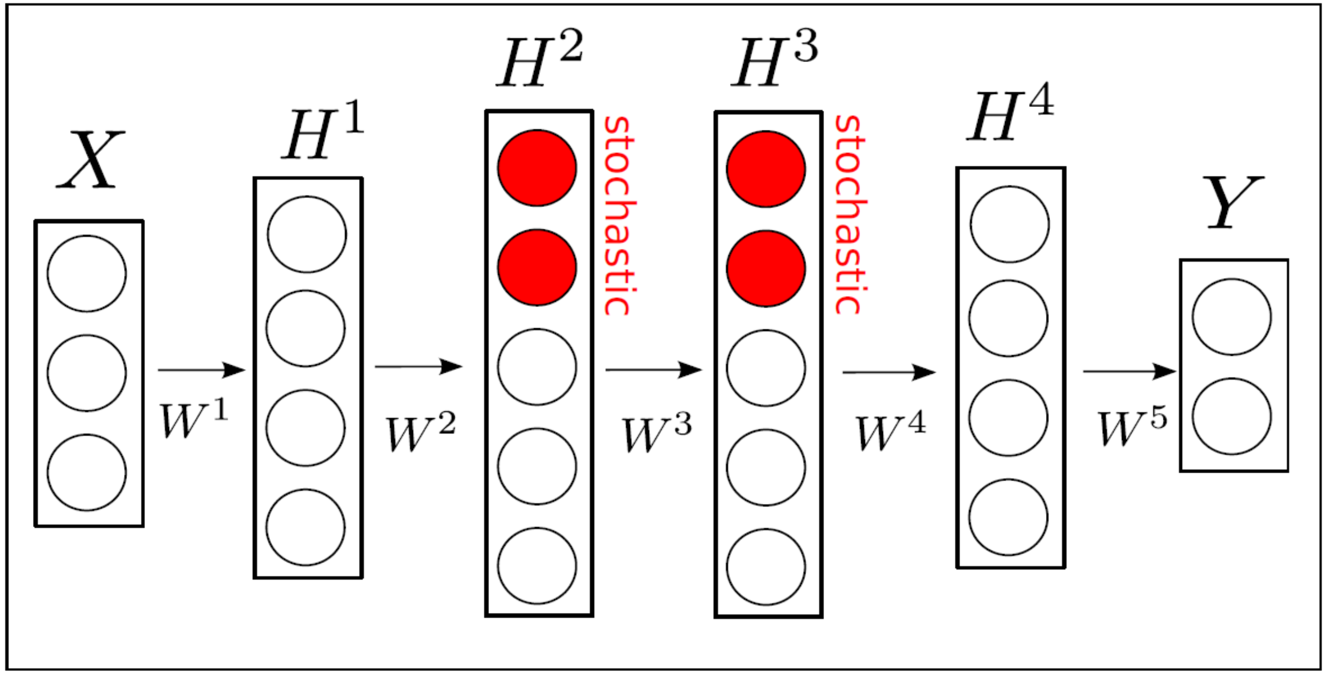
Motivation of SFNN



MLPs are popular models used for non-linear regression and classification tasks. MLPs model the conditional distribution of the predictor variables y given input variables x.
However, this predictive distribution is assumed to be unimodal.
For tasks such as structured prediction problems, the conditional distribution should be one-to-many mappings.

Using only discrete hidden units is suboptimal when modeling real-valued output y.
For a finite number of discrete hidden states, each one is a Gaussian $p(y|h) = \mathcal{N}(y|\mu(h), \sigma_y^2)$, where the mean $\mu(h) = W_2^T h + bias_2$
When x varies, only the probability of choosing a specific hidden state is changed via $p(h|x)$.
A smoother $p(y|x)$ can be learned if $\mu(h)$ is a deterministic function of x as well.

Stochastic Feedforward Neural Network



Turning part of the hidden variables in a MLP into stochastic nodes, Sigmoid Belief Nets induce a rich multimodal distribution in the output space.

Highlights

① To better model continuous data, SFNNs have hidden layers with both discrete stochastic and deterministic units.

② Present a novel Monte Carlo variant of the Generalized Expectation Maximization algorithm for learning. Importance sampling is used for the E-step for inference,

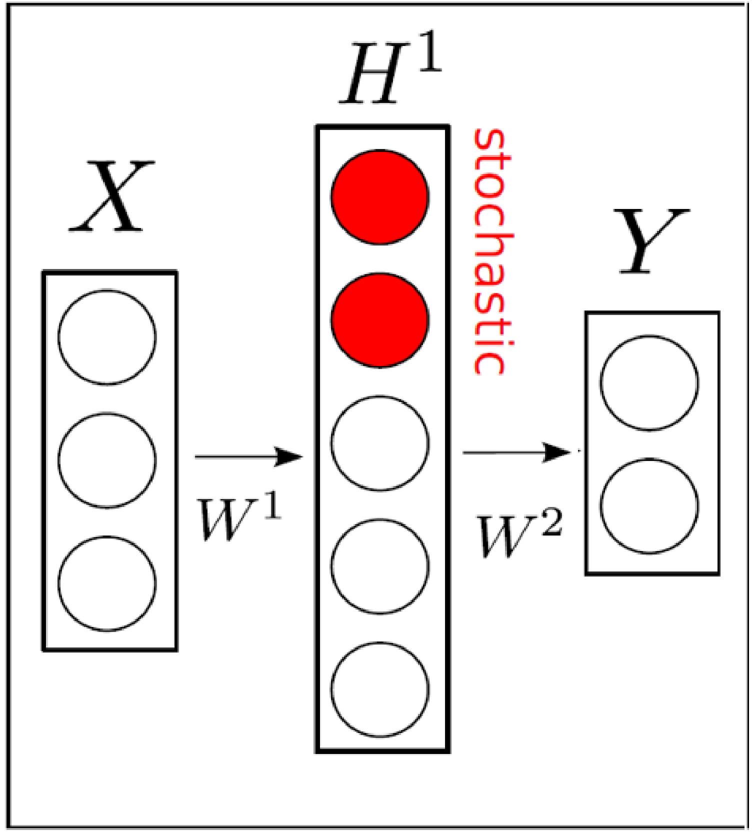
③ Error backpropagation is used by the M-step to improve a variational lower bound on the data log-likelihood.

Advantages of SFNN

- We can draw exact samples from the model without resorting to Markov chain Monte Carlo.
- Stochastic units form a distributed code to represent an exponential number of mixture components in output space.
- As a directed model, learning does not need to deal with a global partition function.
- Combination of stochastic and deterministic hidden units can be jointly trained using the backpropagation algorithm.

Formulation of SFNNs

- SFNNs contain binary stochastic hidden variables $\mathbf{h} \in \{0,1\}^N$
- For simplicity, we can construct a SFNN with one hidden layer.



The conditional distribution is obtained by marginalizing out the latent stochastic hidden variables:

$$p(y|\mathbf{x}) = \sum_{\mathbf{h}} p(y, \mathbf{h}|\mathbf{x}) \qquad (1)$$

SFNN is a directed graphical model, the joint distribution can be factorized as:

$$p(y, \mathbf{h}|\mathbf{x}) = p(y|\mathbf{h})p(\mathbf{h}|\mathbf{x}) \qquad (2)$$

For modeling real-valued y, we can have $p(y|\mathbf{h}) = \mathcal{N}(y|W_2\mathbf{h} + bias_2, \sigma_y^2)$ and $p(\mathbf{h}|\mathbf{x}) = \sigma(W_1\mathbf{x} + bias_1)$.

Monte Carlo

- What is the average height f of people p in Cambridge C?

$$E_{p\in\mathcal{C}}[f(p)] \equiv \frac{1}{|\mathcal{C}|}\sum_{p\in\mathcal{C}} f(p), \quad \text{“intractable”?}$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} f\big(p^{(s)}\big), \quad \text{for random survey of } S \text{ people } \{p^{(s)}\} \in \mathcal{C}$$

Monte Carlo approximates expectations with a sample average

In general:

$$\int f(x)P(x)\,\mathrm{d}x \;\approx\; \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim P(x)$$

Example: making predictions

$$p(x|\mathcal{D}) \;=\; \int P(x|\theta,\mathcal{D})P(\theta|\mathcal{D})\,\mathrm{d}\theta$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} P(x|\theta^{(s)},\mathcal{D}), \quad \theta^{(s)} \sim P(\theta|\mathcal{D})$$

Properties of Monte Carlo

Estimator: $\displaystyle\int f(x)P(x)\,\mathrm{d}x \;\approx\; \hat{f} \;\equiv\; \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim P(x)$

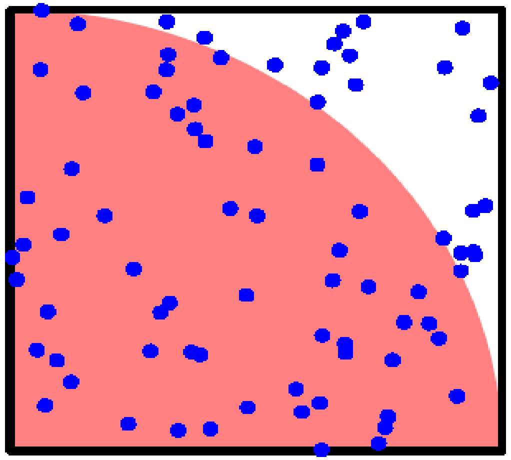
Estimator is unbiased:

$$\mathbb{E}_{P(\{x^{(s)}\})}\left[\hat{f}\right] \;=\; \frac{1}{S}\sum_{s=1}^{S}\mathbb{E}_{P(x)}[f(x)] \;=\; \mathbb{E}_{P(x)}[f(x)]$$

Variance shrinks $\propto 1/S$**:**

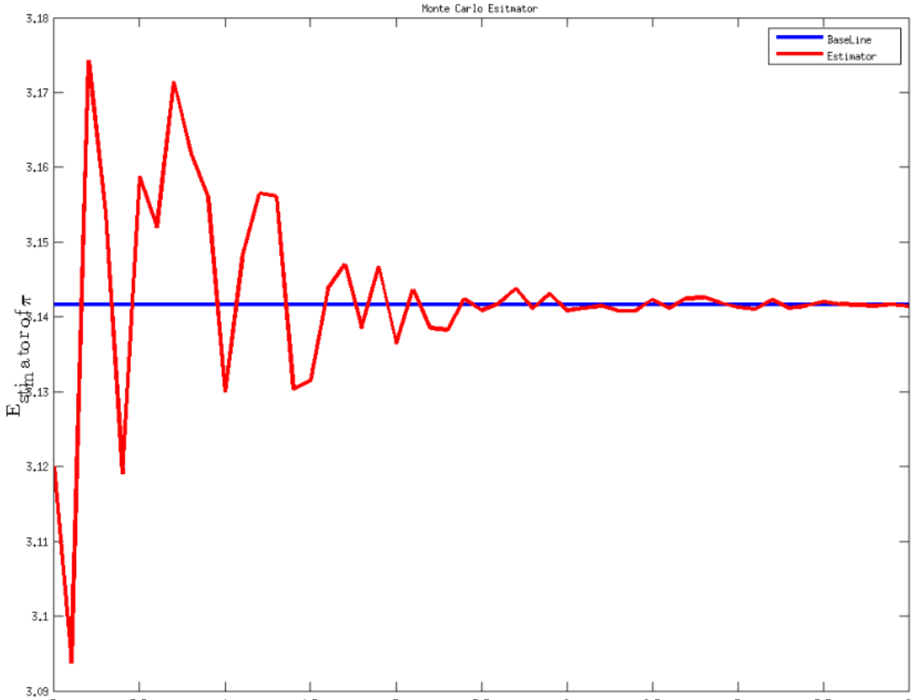
$$\mathrm{var}_{P(\{x^{(s)}\})}\left[\hat{f}\right] \;=\; \frac{1}{S^2}\sum_{s=1}^{S}\mathrm{var}_{P(x)}[f(x)] \;=\; \mathrm{var}_{P(x)}[f(x)]\,/S$$

A dumb approximation of π



$$P(x,y) = \begin{cases} 1 & 0<x<1 \ \text{ and } \ 0<y<1 \\ 0 & \text{otherwise} \end{cases}$$

$$\pi = 4 \iint \mathbb{I}\left((x^2+y^2)<1\right) P(x,y)\, \mathrm{d}x\, \mathrm{d}y$$



Importance sampling

Instead rewrite the integral as an expectation under Q:

$$\int f(x)P(x)\,\mathrm{d}x = \int f(x)\frac{P(x)}{Q(x)}Q(x)\,\mathrm{d}x, \qquad (Q(x) > 0 \text{ if } P(x) > 0)$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)})\frac{P(x^{(s)})}{Q(x^{(s)})}, \quad x^{(s)} \sim Q(x)$$

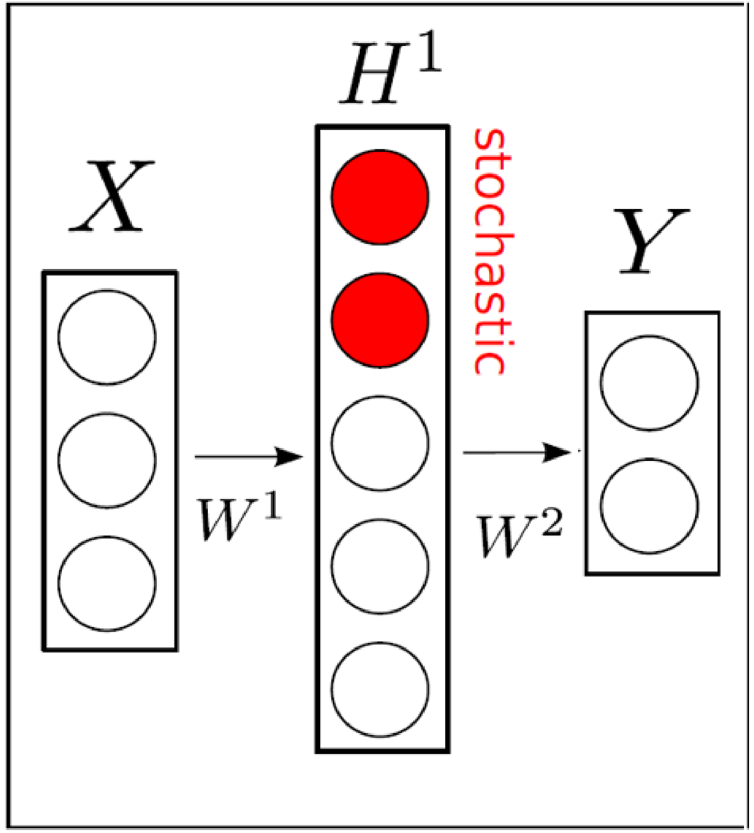
This is just simple Monte Carlo again, so it is unbiased.

Importance sampling applies when the integral is not an expectation.
Divide and multiply any integrand by a convenient distribution.

Importance sampling applies Monte Carlo to 'any' sum/integral

Formulation of SFNNs

- Since $\mathbf{h} \in \{0,1\}^N$ is a vector of N Bernoulli random variables, $p(y|x)$ has up to 2^N models



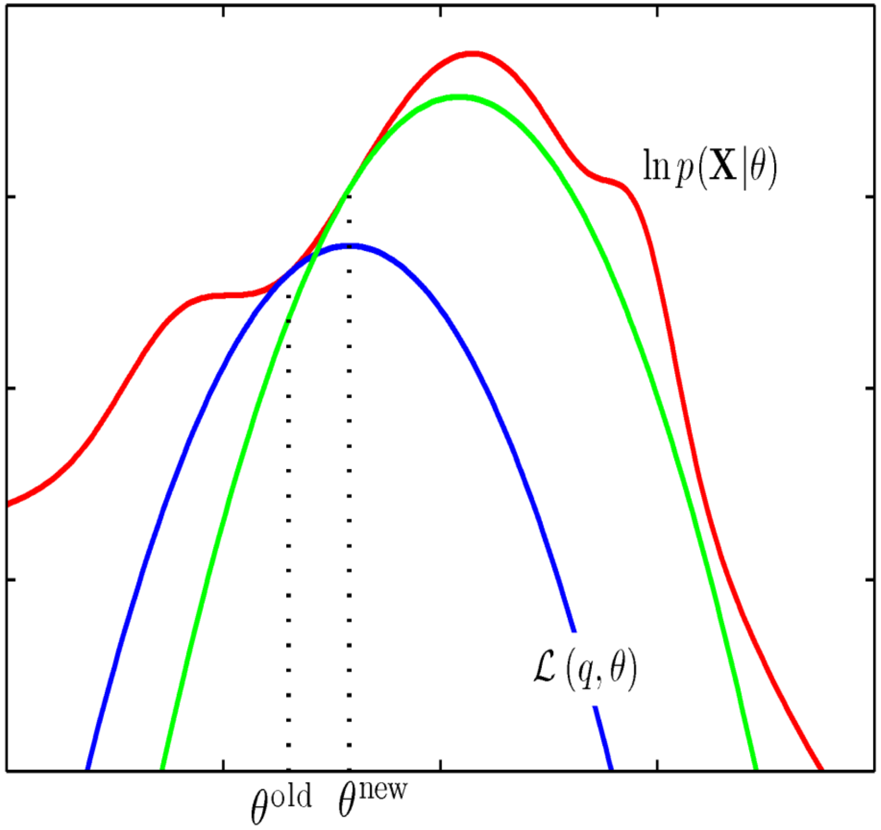
Computation costs come with modeling flexibility of SFNN.
How can we obtain $p(y|x)$ with 2^N components conditioned on $\mathbf{x}$?
Monte Carlo approximation with M samples for its estimation:

$$p(y|\mathbf{x}) \simeq \frac{1}{M} \sum_{m=1}^{M} p(y|\mathbf{h}^{(m)}) \qquad \mathbf{h}^{(m)} \sim p(\mathbf{h}|\mathbf{x}) \quad (3)$$

In SFNNs, we assume a conditional diagonal Gaussian distribution on the output space:

$$\log p(\mathbf{y}|\mathbf{h},\mathbf{x}) \propto -\frac{1}{2}\sum_i \log \sigma_i^2 - \frac{1}{2}\sum_i \frac{(y_i - \mu(\mathbf{h},\mathbf{x}))^2}{\sigma_i^2}$$

Learning procedure of SFNNs- E step



For any approximating distribution $q(h)$, we can write down the following variational lower bound on the data log-likelihood:

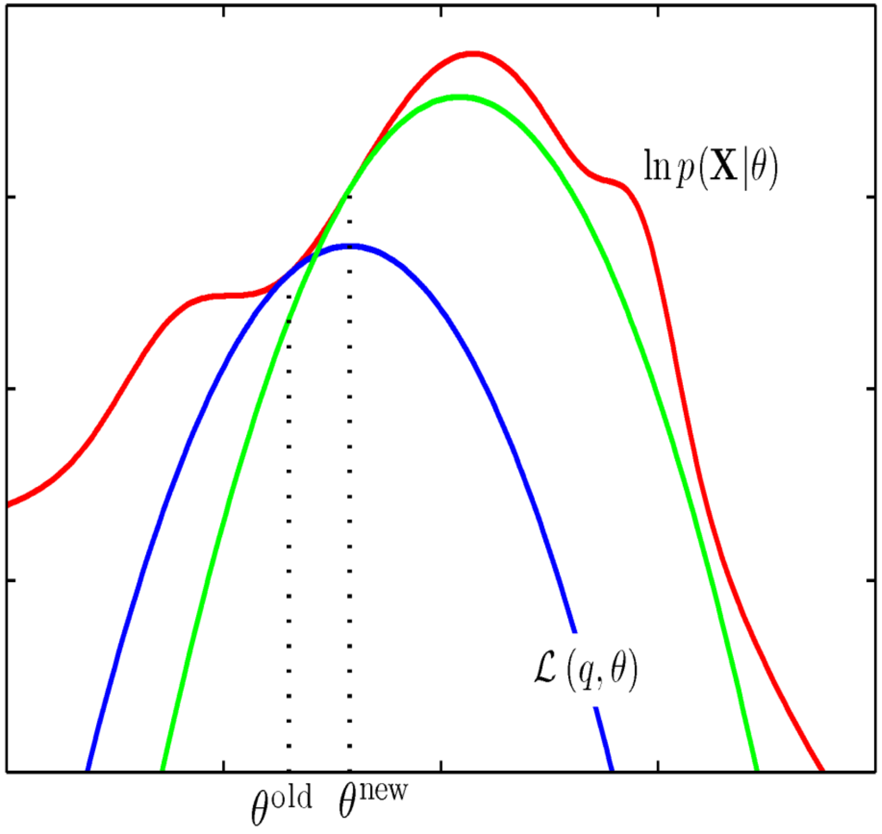
$$\log\, p(y|\mathbf{x}) = \log \sum_{\mathbf{h}} p(y, \mathbf{h}|\mathbf{x})$$

$$= \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(y, \mathbf{h}|\mathbf{x};\theta)}{q(\mathbf{h})} + \mathrm{KL}(q(\mathbf{h})||p(\mathbf{h}|y, \mathbf{x}))$$

$$\geq \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(y, \mathbf{h}|\mathbf{x};\theta)}{q(\mathbf{h})} \qquad (4)$$

For the tightest lower bound, $q(h)$ need to be the exact $p(h, y|x)$

Learning procedure of SFNNs- M step



Let Q be the expected complete data log-likelihood, which is a lower bound on the log-likelihood. We wish to maximize the lower bound:

$$Q(\theta, \theta_{old}) = \sum_{\mathbf{h}} p(\mathbf{h}|y, \mathbf{x}; \theta_{old}) \log p(y, \mathbf{h}|\mathbf{x}; \theta) \qquad (5)$$

$$= \sum_{\mathbf{h}} \frac{p(\mathbf{h}|y, \mathbf{x}; \theta_{old})}{p(\mathbf{h}|\mathbf{x}; \theta_{old})} p(\mathbf{h}|\mathbf{x}; \theta_{old}) \log p(y, \mathbf{h}|\mathbf{x}; \theta)$$

$$\simeq \frac{1}{M} \sum_{m=1}^{M} w^{(m)} \log p(y, \mathbf{h}^{(m)}|\mathbf{x}; \theta), \quad \mathbf{h}^{(m)} \sim p(\mathbf{h}|\mathbf{x}; \theta_{old})$$

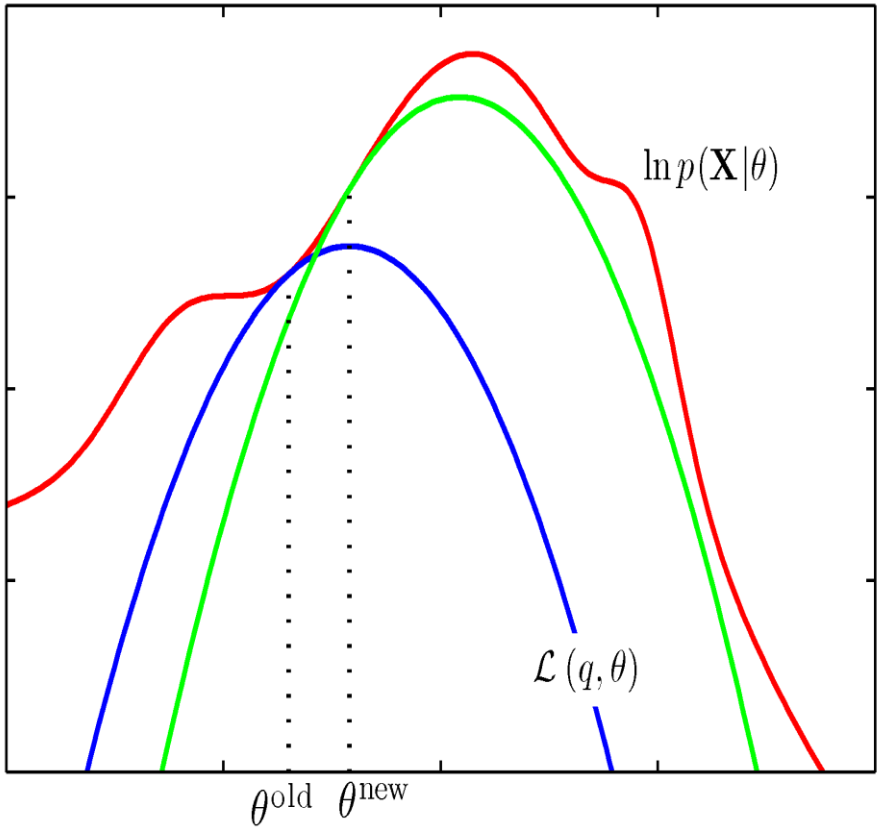
Using Bayes Theorem, importance weight $w^{(m)}$ can be rewritten as:

$$w^{(m)} = \frac{p(\mathbf{h}^{(m)}|y, \mathbf{x}; \theta_{old})}{p(\mathbf{h}^{(m)}|\mathbf{x}; \theta_{old})} = \frac{p(y|\mathbf{h}^{(m)}, \mathbf{x}; \theta_{old})}{p(y|\mathbf{x}; \theta_{old})} \qquad (6)$$

The denominator of $w^{(m)}$ can be approximated using Eq. 3, therefore:

$$w^{(m)} \simeq \frac{p(y|\mathbf{h}^{(m)}; \theta_{old})}{\frac{1}{M}\sum_{m=1}^{M} p(y|\mathbf{h}^{(m)}; \theta_{old})} \qquad (7)$$

Learning procedure of SFNNs- M step



For convenience, we dene the partial objective of the m-th sample as

$$Q^{(m)} \triangleq w^{(m)}\big(\log p(y|\mathbf{h}^{(m)};\theta) + \log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\big). \quad (8)$$

We can then approximate our objective function $Q(\theta, \theta_{old})$ with M samples from the proposal:

$$Q(\theta, \theta_{old}) \simeq \frac{1}{M}\sum_{m=1}^{M} Q^{(m)}(\theta, \theta_{old})$$

For our generalized M-step, we seek to perform gradient ascent on Q:

$$\frac{\partial Q}{\partial \theta} \simeq \frac{1}{M}\sum_{m=1}^{M} \frac{\partial Q^{(m)}(\theta, \theta_{old})}{\partial \theta} \quad (9)$$

$$= \frac{1}{M}\sum_{m=1}^{M} w^{(m)} \frac{\partial}{\partial \theta}\Big\{\log p(y|\mathbf{h}^{(m)};\theta) + \log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\Big\}$$

EM algorithm for SFNNs

Given training D dimensional data pairs: $\{\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\}$, $n = 1 \ldots N$. Hidden layers $\mathbf{h}^1 \& \mathbf{h}^4$ are deterministic, $\mathbf{h}^2 \& \mathbf{h}^3$ are hybrid. $\theta = \{W^{1,2,3,4,5}, bias, \sigma_y^2\}$

```
repeat
  //Approximate E-step:
  Compute p(h^2 | x^(n)) = Bernoulli(σ(W^2 σ(W^1 x^(n))))
  h^2_determ ← p(h^2_determ | x^(n))
  for m = 1 to M (importance samples) do
    Sample: h^2_stoch ~ p(h^2_stoch | x^(n)).
    let h^2 be the concatenation of h^2_stoch and h^2_determ.
    p(h^3 | x^(n)) = Bernoulli(σ(W^3 h^2))
    h^3_determ ← p(h^3_determ | x^(n))
    Sample: h^3_stoch ~ p(h^3_stoch | x^(n))
    let h^3 be the concatenation of h^3_stoch and h^3_determ.
    Compute p(y | x^(n)) = N(σ(W^5 σ(W^4 h^3)); σ_y^2)
  end for
  Compute w^(m) for all m, using Eq. 7.

  //M-step:
  Δθ ← 0
  for m = 1 to M do
    Compute ∂Q^(m)(θ, θ_old)/∂θ by Backprop.
    Δθ = Δθ + ∂Q^(m)/∂θ
  end for
  θ_new = θ_old + (α/M) Δθ, //α is the learning rate.
until convergence
```