# STA 414/2104:
# Statistical Methods of Machine Learning II

## Week 12: Variational Autoencoders and Exam Summary

Piotr Zwiernik

University of Toronto

# Table of contents

# Variational Autoencoders

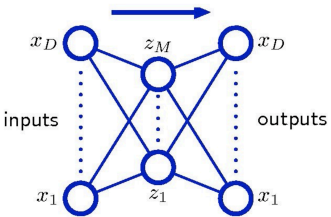
# Motivation/Summary

**Variational Autoencoders (VAEs)** are powerful generative models, now having applications as diverse as from generating fake human faces, to producing purely synthetic music.

The plan for the first hour:

- We start by introducing **Autoencoders** as a nonlinear dimensionality reduction technique.
- We argue why they do not fit well the generative task.
- We discuss VAEs as a suitable alternative.
- VAEs incorporate two powerful techniques we learned recently: neural networks and variational inference.

# Non-linear Dimension Reduction




- Neural networks can be used for nonlinear dimensionality reduction.
- This is achieved by having the same number of outputs as inputs. These models are called autoencoders.
- Consider a feed-forward neural network that has $D$ inputs, $D$ outputs, and $M$ hidden units, with $M < D$.
- We can squeeze the information through a bottleneck.
- If we use a linear network (linear activation) this is very similar to Principal Components Analysis.

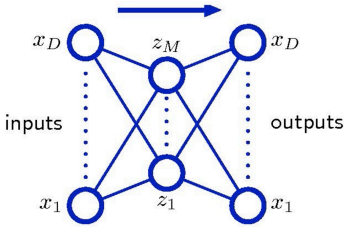
# Autoencoders and PCA

- Given an input $\boldsymbol{x}$, its corresponding reconstruction is given by:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^{M} w_{kj}^{(2)} \sigma\Big( \sum_{i=1}^{D} w_{ji}^{(1)} x_i \Big), \quad k = 1, ..., D.$$

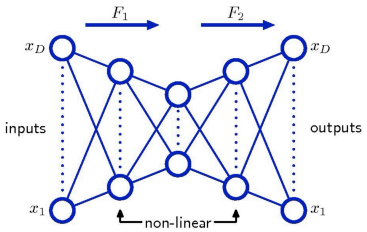
- We learn the parameters $\boldsymbol{w}$ by minimizing the reconstruction error:

$$E(\boldsymbol{w}) = \frac{1}{2N} \sum_{n=1}^{N} \| y(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{x}_n \|^2$$




- In the case when layers are linear:
  - it will learn hidden units that are linear functions of the data and minimize squared error.
  - $M$ hidden units will span the same space as the first $M$ principal components (PCA).

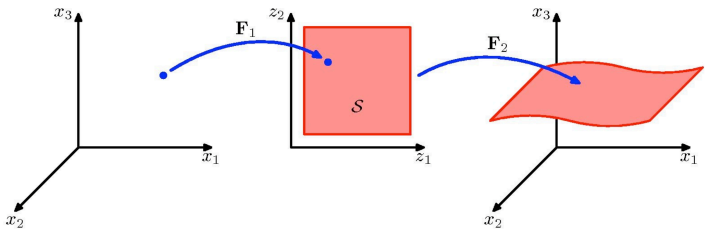
# Deep Autoencoders




- We can put extra nonlinear hidden layers between the input and the bottleneck and between the bottleneck and the output.
- This gives nonlinear generalization of PCA, providing non-linear dimensionality reduction.
- The network can be trained by the minimization of the reconstruction error function.
- Much harder to train.

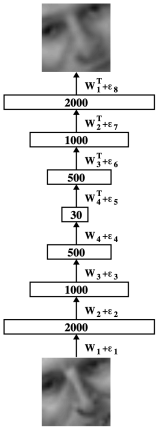
# Geometrical Interpretation

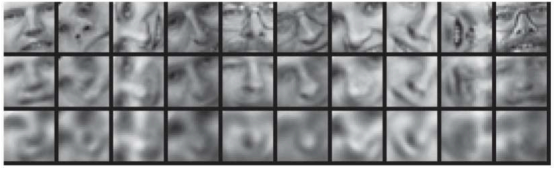
- Geometrical interpretation of the mappings performed by the network with 2 hidden layers for the case of $D = 3$ and $M = 2$ units in the middle layer.




- The mapping $F_1$ defines a nonlinear projection of points in the original $D$-space into the $M$-dimensional subspace.
- The mapping $F_2$ maps from an $M$-dimensional space into $D$-dimensional space.

# Deep Autoencoders



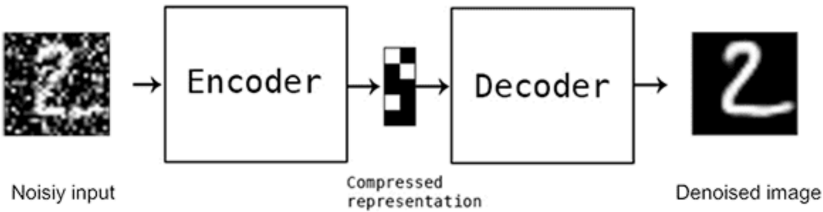
- We can consider very deep autoencoders.
- By row: Real data, Deep autoencoder with a bottleneck of 30 units, and 30-d PCA.

# Deep Autoencoders

- Similar model for MNIST handwritten digits:


- Deep autoencoders produce much better reconstructions.

# Application: Image Denoising




- We can train a **denoising autoencoder**.
- We feed noisy image as an input to the encoder
- Minimize the reconstruction error between the decoder output and original image.
- This method requires training and knowledge of the noise structure (fully supervised).
- In contrast, loopy BP works for a single noisy image and doesn't require the knowledge of noise structure (unsupervised).

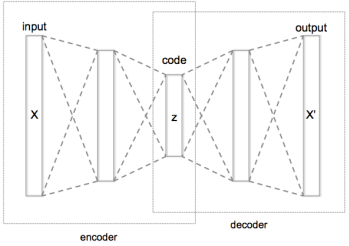
# Autoencoders: Summary

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $e(x) = z \in F, \quad x \in X$
- **Decoder**: $d(z) = \tilde{x} \in X$
- where $X$ is the data space, and $F$ is the feature (latent) space.
- $z$ is the code, compressed representation of the input, $x$. It is important that this code is a bottleneck, i.e. that

$$\dim F \ll \dim X$$

- Goal: $\tilde{x} = d(e(x)) \approx x$.



# Issues with (deterministic) Autoencoders

- **Issue 1**: Proximity in data space does not mean proximity in feature space
  - The codes learned by the model are deterministic, i.e.

$$g(x_1) = z_1 \implies f(z_1) = \tilde{x}_1$$
$$g(x_2) = z_2 \implies f(z_2) = \tilde{x}_2$$

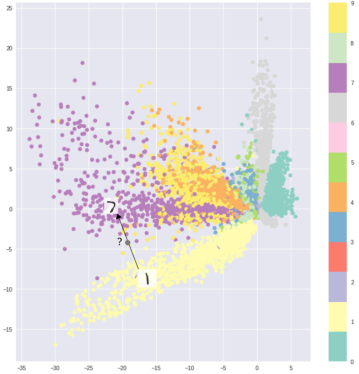
  - but proximity in feature space is not "directly" enforced for inputs in close proximity in data space, i.e.

$$x_1 \approx x_2 \not\implies z_1 \approx z_2$$

  - The latent space may not be continuous, or allow easy interpolation.

# Issues with (deterministic) Autoencoders

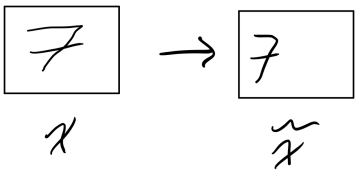
- **Issue 1**: Proximity in data space does not mean proximity in feature space
  - If the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will simply generate an unrealistic output.




Two dimensional latent space for the MNIST digit data.(Image credit: I. Shafkat)

# Issues with (deterministic) Autoencoders

- **Issue 2**: How to measure the goodness of a reconstruction?



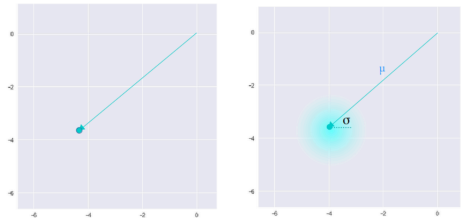
  - The reconstruction looks quite good. However, if we chose a simple distance metric between inputs and reconstructions, we would heavily penalize the left-shift in the reconstruction $\tilde{x}$.
  - Choosing an appropriate metric for evaluating model performance can be difficult, and that a miss-aligned objective can be disastrous.

# Variational Autoencoders

- Variational autoencoders (VAEs) encode inputs with uncertainty.
- Unlike standard autoencoders, the encoder of a VAE outputs a probability distribution, $q_\phi(z)$ to approximate $p(z|x)$.
- We assume that $q_\phi$ is a product of univariate normal distributions.
- Instead of the encoder learning an encoding vector, it learns two vectors: vector of means, $\mu$, and another vector of standard deviations, $\sigma$.

# Variational Autoencoders

- The mean $\mu$ controls where encoding of input is centered while the standard deviation controls how much can the encoding vary.




Standard Autoencoder
(direct encoding coordinates)

Variational Autoencoder
(μ and σ initialize a probability distribution)

Image credit: I. Shafkat

- Encodings are generated at random from the "ball", the decoder learns that all nearby points refer to the same input.

To minimize the reconstruction error, the algorithm may want to concentrate around the direct encodings. We will try to discourage this.

# VAE: Specifics

- Our model is generated by the joint distribution over the latent codes and the input data $p_\theta(x, z)$:
  - The prior: $z \sim p_\theta(z)$ often Gaussian.
  - The likelihood: $x|z \sim \mathrm{Expfam}(x|d_\theta(z))$ with the decoder $d_\theta(z)$ given by a DNN,
    - e.g. for binary observations: $p_\theta(x|z) = \prod_{d=1}^{D} \mathrm{Ber}(x_d|\sigma(d_\theta(z)))$
    - e.g. for continuous observations: $p_\theta(x|z) = \prod_{d=1}^{D} N(x_d|d_\theta(z))$
- We could use the posterior $p_\theta(z|x) = p_\theta(x, z)/p_\theta(x)$ for encoding.

**Issue: Learning $p(x) = \int p(x|z)p(z)dz$ is intractable.**

Introduce an approximation $q_\phi$ with its own set of parameters $\phi$, such that

$$q_\phi(z|x) \approx p_\theta(z|x).$$

# VAE: Specifics

- Recall the idea behind Variational Inference:

$$\begin{aligned}\mathcal{L}(\theta, \phi|x) &= \text{ELBO} = \log p_\theta(x) - KL(q_\phi(z|x)\|p_\theta(z|x)) \\ &= \mathbb{E}_{z\sim q_\phi}\Big[\log p_\theta(x|z)\Big] - KL(q_\phi(z|x)\|p(z))\end{aligned}$$

  which is the (negative) loss function we use when training VAEs.
- First term in blue is the expected log-likelihood and the second is the divergence of $q_\phi$ from the prior.
- The encoder and decoder in a VAE become:
  - **Encoder**: $q_\phi(z|x)$, where

$$q_\phi(z|x) = N(\mu(x), \Sigma_\phi(x)),$$

    where $\mu_\phi(x)$ and diagonal $\Sigma_\phi(x)$ are computed by a deep NN.
  - **Decoder**: $p(x|z) \sim \text{Expfam}(x|d_\theta(z))$, where $d_\theta(z)$ is typically a deep neural network

# VAE loss interpretation

The VAE maximization objective can be written as

$$\mathcal{L}(\theta, \phi|x) = E_{z_\phi \sim q_\phi}\Big[\log p_\theta(x|z)\Big] - KL(q_\phi(z|x)\|p(z))$$

- The first term handles the reconstruction loss.
- The second term regularizes the encoder not to be too far from the prior; this will makes sure that the network cannot simply memorize the data.

## $\beta$-VAE loss

$$\mathcal{L}(\theta, \phi|x) = E_{z_\phi \sim q_\phi}\Big[\log p_\theta(x|z)\Big] - \beta KL(q_\phi(z|x)\|p(z)).$$

- $\beta > 1$: constrain latent bottleneck, enforce latent features to be uncorrelated.
- Produces features that are better disentangled.

# Optimizing the ELBO

The ELBO for the whole dataset, scaled by its size $N$, is

$$\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\theta,\phi|x_n) \;=\; \frac{1}{N}\sum_{n=1}^{N}\left\{\mathbb{E}_{z\sim q_\phi(z|x_n)}\Big[\log p_\theta(x_n|z)+\log p_\theta(z)-\log q_\phi(z|x_n)\Big]\right\}$$

(note that $\phi$ is shared across samples $=$ **amortized inference**)

- Optimize this with respect to $\theta$ and $\phi$.
- We can create a mini-batch approximation of this objective.
- The gradient can be approximated using the reparametrization trick.

Let's recall this quickly.

# Recall: Approximate the gradient for a single example

We use the reparametrization trick ($z \sim q_\phi(z|x_n)$ the same distr. as $T_\phi(\epsilon)$ with $\epsilon \sim p_0$)

$$\begin{aligned}\nabla_\phi \mathcal{L}(\theta, \phi|x_n) &= \nabla_\phi \mathbb{E}_{z\sim q_\phi(z)}\Big[\log p_\theta(x_n|z) + \log p_\theta(z) - \log q_\phi(z|x_n)\Big] \\ &= \mathbb{E}_{\epsilon\sim p_0(\epsilon)} \nabla_\phi \Big[\log p(x|\,T_\phi(\epsilon)) + \log p_\theta(\,T_\phi(\epsilon)) - \log q_\phi(\,T_\phi(\epsilon)|x_n)\Big].\end{aligned}$$

In our case, $q_\phi$ is a product of normals with $\phi = (\mu_1, \ldots, \mu_M, \log\sigma_1, \ldots, \log\sigma_M)$

- The gradients of all terms with respect to $\phi$ are easy to compute.
- Since $\phi = \phi(x)$ is a NN, we can compute all gradients using backpropagation.

Similar idea for $\nabla_\theta \mathcal{L}(\theta, \phi|x_n)$. (here no reparametrization trick is needed).

# Recall: Stochastic VI

Approximate $\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\theta,\phi|x_n)$ with $\mathcal{L}(\theta,\phi|x_s)$ for a singe sample $x_s$ from the dataset.

Get an approximate of the gradient $\nabla_\phi\mathcal{L}(\theta,\phi|x_s)$ using simple Monte Carlo

- Sample $z_i \sim q_\phi(z|x_s)$. (latent representations in our feature space)
- Compute the gradient of ELBO at each sample and take the average.

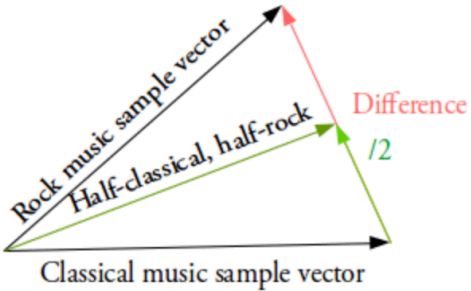
Update the parameters $\phi$ using the SGD step. Then do the same for $\theta$.

# After VAE is trained

- Once a VAE is trained, we can sample new inputs (generative model)

$$z \sim p(z) \qquad \tilde{x} \sim p_\theta(x|z)$$

- We can also interpolate between inputs, using simple vector arithmetic.



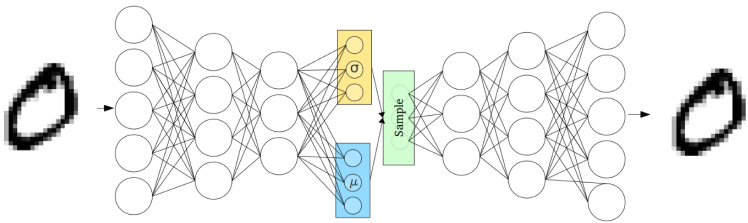

Interpolating between samples

# Example: MNIST

- We choose the prior on $z$ to be the standard Gaussian: $p(z) \sim \mathcal{N}(0, I)$.
- The likelihood function: $p_\theta(x|z) = \prod_{i=1}^{D} \text{Bernoulli}(\theta_i)$.
- Approximate posterior: $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \Sigma_\phi(x))$, where $\Sigma_\phi$ diagonal.
- Finally, we use neural networks as our encoder and decoder
  - **Encoder**: $e_\phi(x) = [\mu_\phi(x), \log \Sigma_\phi(x)]$
  - **Decoder**: $d_\theta(z) = (\theta_1(z), \ldots, \theta_d(z))$
  - where $\theta_i$ are parameters of a Bernoulli rv for each input pixel.
- To get our reconstructed input, we simply take

$$\tilde{x} \sim p_\theta(x|z)$$

# Example: MNIST




- We use neural networks for both the encoder and the decoder.
- We compute the loss function $\mathcal{L}(\theta, \phi; x)$ and propagate its derivative with respect to $\theta$ and $\phi$, $\nabla_\theta \mathcal{L}$, $\nabla_\phi \mathcal{L}$, through the network during training.
- We use reparametrization trick as described before.

# MNIST: Autoencoder vs VAE

Codes generated by L: AE R: VAE

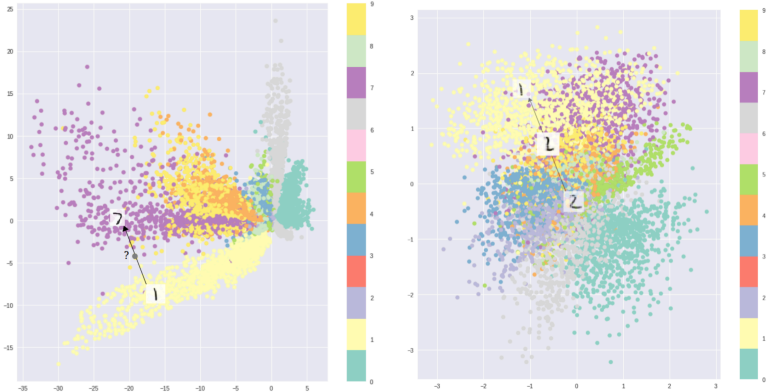


Image credit: I. Shafkat

# Exam revision

# Final exam logistic

- Final exam will be held in person on April 18, at 19-22 Toronto local time in room BR 200 (all sections).
- Exams will be 100 points in total and 180 mins long. Students are required to be at the exam location at least 10 mins early, with valid identification. Exam will be administered by FAS.
- You can use two optional A4 handwritten aid sheets - double-sided.
- Exam covers all lectures (weeks 1-12), it is closed book/internet.
- A representative practice exam will be posted after A4 due date.

# Probabilistic ML Terminology

The final exam will be on the entire course; however, it will be more weighted towards post-midterm material. We will go briefly over the main concepts:

- Exponential families
- Directed Graphical Models
- Markov Random Fields
- Message passing
- Belief propagation
- Variable eliminitaion
- Sampling methods
- Markov chain Monte Carlo
- Hidden Markov Models
- Variational Inference
- EM algorithm
- Probabilistic PCA
- Bayesian regression
- Kernel methods
- Gaussian processes
- Neural networks

- Density of a member of exponential families is of the form

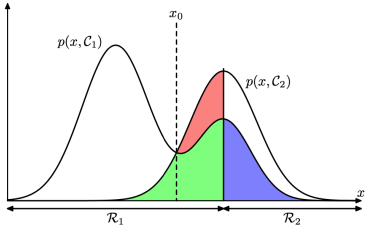
$$p(x|\eta) = h(x)\exp\{\eta^\top T(x) - A(\eta)\}$$

Here,

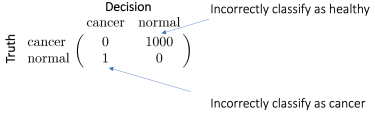
$$T(x) : \text{Sufficient statistics}$$
$$\eta : \text{Natural parameter}$$
$$A(\eta) : \text{log-partition function}$$
$$h(x) : \text{carrying density}$$

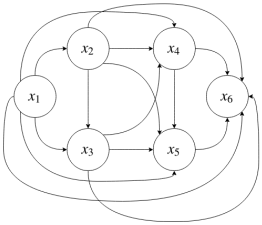
- Examples of exponential familes are
  - Bernoulli, Gaussian, Gamma, exponential, Poisson etc.
  - Defines a broad class of distributions
  - Moments of sufficient statistics can be found easily by differentiating the log-partition function.

- Minimizing the misclassification rate:




- We use a **loss function** to measure the loss incurred by taking any of the available decisions.
  - e.g. consider medical diagnosis example: example of a loss matrix:






- A directed acyclic graphical model (DAGM) implies a factorization of the joint distribution.
- Variables are represented by nodes, and edges represent dependence.

DAGM induces factorization of the joint distribution of $x_1, x_2, \ldots, x_N$:

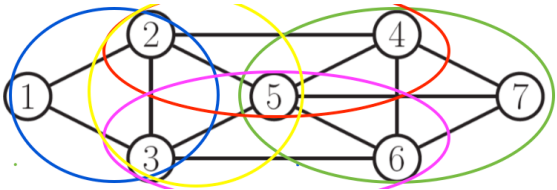
$$p(x_1, \ldots, x_N) = \prod_{i=1}^{N} p(x_i | \text{pa}(x_i))$$

where $\text{pa}(x_i)$ is the set of nodes with edges pointing to $x_i$.

**Bayes Ball Algorithm or moralization**

Allow to read relevant conditional independences from the graph.

- Markov random fields (MRFs), are a set of random variables where the dependencies are described by an undirected graph.




In the above example we get:

$$p(x) \propto \psi_{1,2,3}(x_1, x_2, x_3)\psi_{2,3,5}(x_2, x_3, x_5)\psi_{2,4,5}(x_2, x_4, x_5)\psi_{3,5,6}(x_3, x_5, x_6)\psi_{4,5,6,7}(x_4, x_5, x_6, x_7)$$

**Relation between DAGs and MRFs**

These two model families intersect. There is no containment.

**Main point**

Order which variables are marginalized affects the computational cost!

Main tool in exact inference is **variable elimination**:

- A simple and general **exact inference** algorithm in any probabilistic graphical model (DAGMs or MRFs).
- Has computational complexity that depends on the graph structure of the model.
- Sum-product is used to obtain marginals.

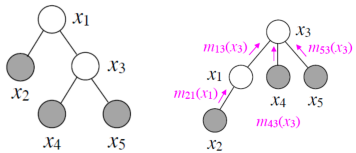
- Different elimination orderings will involve different number of variables appearing inside each sum.
- The complexity of the VE algorithm is

$$O(mk^{N_{\max}})$$

where

- $m$ is the number of initial factors.
- $k$ is the number of states each random variable takes (assumed to be equal here).
- $N_i$ is the number of random variables inside each sum $\sum_i$.
- $N_{\max} = \max_i N_i$ is the number of variables inside the largest sum.

There is a canonical way to do variable elimination on trees.




$$
\begin{aligned}
p(x_3|x_E) &= \frac{1}{Z^E}\sum_{x_1}\psi_1(x_1)\psi_2(\bar{x}_2)\psi_3(x_3)\psi_4(\bar{x}_4)\psi_5(\bar{x}_5)\psi_{12}(x_1,\bar{x}_2)\psi_{13}(x_1,x_3)\psi_{34}(x_3,\bar{x}_4)\psi_{35}(x_3,\bar{x}_5) \\
&= \frac{1}{Z^E}\underbrace{\psi_4(\bar{x}_4)\psi_{34}(x_3,\bar{x}_4)}_{m_{43}(x_3)}\underbrace{\psi_5(\bar{x}_5)\psi_{35}(x_3,\bar{x}_5)}_{m_{53}(x_3)}\psi_3(x_3)\sum_{x_1}\psi_1(x_1)\psi_{13}(x_1,x_3)\underbrace{\psi_2(\bar{x}_2)\psi_{12}(x_1,\bar{x}_2)}_{m_{21}(x_1)} \\
&= \frac{1}{Z^E}m_{43}(x_3)m_{53}(x_3)\psi_3(x_3)\underbrace{\sum_{x_1}\psi_1(x_1)\psi_{13}(x_1,x_3)m_{21}(x_1)}_{m_{13}(x_3)} \\
&= \frac{1}{Z^E}\psi_3(x_3)m_{43}(x_3)m_{53}(x_3)m_{13}(x_3) = \frac{\psi_3(x_3)m_{43}(x_3)m_{53}(x_3)m_{13}(x_3)}{\sum_{x_3'}\psi_3(x_3')m_{43}(x_3')m_{53}(x_3')m_{13}(x_3')}
\end{aligned}
$$

- The message sent from variable $j$ to $i \in N(j)$ is

$$m_{j\to i}(x_i) = \sum_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)/i} m_{k\to j}(x_j)$$

  - If $x_j$ is observed, the message is

$$m_{j\to i}(x_i) = \psi_j(\bar{x}_j)\psi_{ij}(x_i, \bar{x}_j) \prod_{k\in N(j)/i} m_{k\to j}(\bar{x}_j)$$

- To compute all marginals, two passes are needed: one from leaves to root, one from root to leaves. Once the message passing stage is complete, compute beliefs

$$b(x_i) \propto \psi_i(x_i) \prod_{j\in N(i)} m_{j\to i}(x_i).$$

**Similar idea applied to general graphs**

If it is not a tree, run loopy BP.

- The algorithm we learned is called **sum-product BP** and approximately computes the **marginals** at each node.
- For MAP inference, we maximize over $x_j$ instead of summing over them. This is called **max-product BP**.
- BP updates take the form

$$m_{j\to i}(x_i) = \max_{x_j} \psi_j(x_j)\psi_{ij}(x_i, x_j) \prod_{k\in N(j)\setminus i} m_{k\to j}(x_j)$$

- MAP inference:

$$\hat{x}_i = \arg\max_{x_i} b(x_i).$$

Estimation problem using simple Monte Carlo:

- **Simple Monte Carlo**: Given $\{x^{(r)}\}_{r=1}^{R} \sim p(x)$ we can estimate the expectation $\mathop{\mathbb{E}}_{x\sim p(x)}[\phi(x)]$ using the estimator $\hat{\Phi}$:

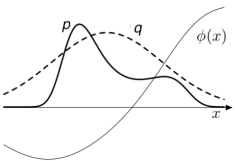
$$\Phi := \mathop{\mathbb{E}}_{x\sim p(x)}[\phi(x)] \approx \frac{1}{R}\sum_{r=1}^{R}\phi(x^{(r)}) := \hat{\Phi}$$

- The fact that $\hat{\Phi}$ is a consistent estimator of $\Phi$ follows from the Law of Large Numbers (LLN).
- Easy to design estimators using simple Monte Carlo, e.g. practice midterm.

- Target $p(x)$ can be evaluated up to normalizing constant $\tilde{p}(x)$
- There is a simpler density, $q(x)$ from which it is easy to sample from and can evaluate up to normalizing constant $\tilde{q}(x)$

$$\text{Sample:} \quad x^{(r)} \sim q(x) = \tilde{q}(x)/Z_q$$

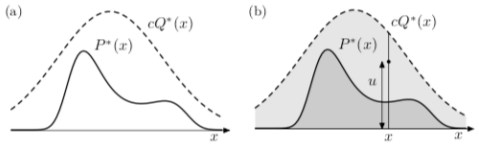
**Importance sampling**: estimate the expectation of a function $\phi(x)$.




- Introduce weights: $\tilde{w}_r = \frac{\tilde{p}(x^{(r)})}{\tilde{q}(x^{(r)})}$
- The importance weighted estimator of $\mathbb{E}_p\phi(x)$

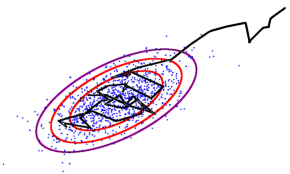
$$\hat{\Phi}_{iw} = \sum_{r=1}^{R} \phi(x^{(r)}) \cdot w_r$$

where $w_r = \frac{\tilde{w}_r}{\sum_{r=1}^{R} \tilde{w}_r}$




The procedure is as follows:

1. Generate two random numbers.
   1.1 The first, $x$, is generated from the proposal density $q(x)$.
   1.2 The second, $u$ is generated uniformly from the interval $[0, c\tilde{q}(x)]$ (see figure (b) above: book's notation $P^* = \tilde{p}$, $Q^* = \tilde{q}$).
2. Accept or reject the sample $x$ by comparing the value of $u$ with the value of $\tilde{p}(x)$
   2.1 If $u > \tilde{p}(x)$, then $x$ is rejected
   2.2 Otherwise $x$ is accepted; $x$ is added to our set of samples $\{x^{(r)}\}$ and the value of $u$ discarded.



- In contrast to rejection sampling, where the accepted points $\{x^{(t)}\}$ are independent, MCMC methods generate a dependent sequence.
- Each sample $x^{(t)}$ has a probability distribution that depends on the previous value, $x^{(t-1)}$.
- MCMC methods need to be run for a time in order to generate samples that are from the target distribution $p$.

We can still do Monte Carlo estimaton for large enough $T$ to estimate the mean of a test function $\phi$:

$$\mathbb{E}_{x\sim p}[f(x)] \;\approx\; \frac{1}{T}\sum_{t=1}^{T} f(x^{(t)}).$$

(good idea to discard a bunch of initial samples)

As before, assume we can evaluate $\tilde{p}(x)$ for any $x$. Our procedure:

- A tentative new state $x'$ is generated from the proposal density $q(x'|x^{(t)})$. We accept the new state with probability

$$A(x'|x^{(t)}) = \min\left\{1, \frac{\tilde{p}(x')q(x^{(t)}|x')}{\tilde{p}(x^{(t)})q(x'|x^{(t)})}\right\}$$

  - If accepted, set $x^{(t+1)} = x'$. Otherwise, set $x^{(t+1)} = x^{(t)}$.

- Metropolis: Simpler version when $q(x'|x) = q(x|x')$ for all $x, x'$.

- **Theorem:** This procedure defines a Markov chain with stationary distribution $\pi(x)$ equal to the target distribution $p(x)$.

# Week 5: Gibbs Sampling Procedure

Suppose the vector $x$ has been divided into $d$ components

$$x = (x_1, ..., x_d).$$

Start with any $x^{(0)} = (x_1^{(0)}, \ldots, x_d^{(0)})$. In the $t$-th iteration:

- For $j = 1, \ldots, d$:
  - Sample $x_j^{(t)}$ from the conditional distribution given other components:

$$x_j^{(t)} \sim p(x_j | x_{-j}^{(t-1)})$$

Where $x_{-j}^{(t-1)}$ represents all the components of $x$ except for $x_j$ at their current values:

$$x_{-j}^{(t-1)} = (x_1^{(t)}, x_2^{(t)}, ..., x_{j-1}^{(t)}, x_{j+1}^{(t-1)}, ..., x_d^{(t-1)})$$

- No accept/reject, only accept.

The conditional distribution does not depend on the normalizing constant.

**The HMC algorithm (run until it mixes):**

- Current position: $(x^{(t-1)}, v^{(t-1)}))$
- Sample momentum: $v^{(t)} \sim \mathcal{N}(0, I)$.
- Start at $(x, v) = (x^{(t-1)}, v^{(t)})$ and run Leapfrog integrator for $L$ steps and reach $(x', v')$. (a point with the same total energy)
- Accept new state $(x', -v')$ with probability:

$$\min \left\{ 1, \frac{\exp(H(x^{(t-1)}, v^{(t-1)}))}{\exp(H(x', v'))} \right\}$$

- Low energy points are favored.

- Important DAGMs to simplify the joint distribution.
- Posterior inference takes the special form:

$$p(z_t|x_{1:T}) \propto p(z_t, x_{1:t})p(x_{t+1:T}|z_t) \propto \text{(Forward Recursion)(Backward Recursion)}$$

- **Forward-backward algorithm** to compute $p(z_t|x_{1:T})$
- **Viterbi algorithm** to compute the most probable sequence.

$$\hat{z} = \arg\max_{z_{1:T}} p(z_{1:T}|x_{1:T})$$

We will measure the difference between $q$ and $p$ using the **Kullback-Leibler divergence**

$$KL(q(z)\|p(z)) = \int q(z) \log \frac{q(z)}{p(z)} dz \quad \text{or} \quad = \sum_z q(z) \log \frac{q(z)}{p(z)}$$

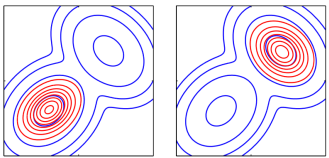
Properties of the KL Divergence

- $KL(q\|p) \geq 0$
- $KL(q\|p) = 0 \Leftrightarrow q = p$
- $KL(q\|p) \neq KL(p\|q)$
- KL divergence is not a metric, since it is not symmetric

I-projection: $q^* = \arg\min_{q \in Q} KL(q||p) = \mathbb{E}_{x \sim q(x)} \log \frac{q(x)}{p(x)}$:

- $p \approx q \implies KL(q\|p)$ small
- I-projection underestimates support, and does not yield the correct moments.
- $KL(q||p)$ penalizes $q$ having mass where $p$ has none.

$p(x)$ is mixture of two 2D Gaussians and $Q$ is the set of all 2D Gaussian distributions (with arbitrary covariance matrices)

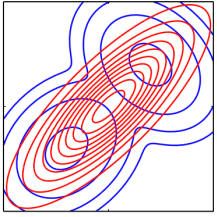


$p$=Blue, $q^*$=Red (two equivalently good solutions!)

M-projection: $q^* = \arg\min_{q \in Q} KL(p||q) = \mathbb{E}_{x \sim p(x)} \log \frac{p(x)}{q(x)}$:

- $p \approx q \implies KL(p\|q)$ small
- $KL(p||q)$ penalizes $q$ missing mass where $p$ has some.
- M-projection yields a distribution $q(x)$ with the correct mean and covariance.

$p(x)$ is mixture of two 2D Gaussians and $Q$ is the set of all 2D Gaussian distributions (with arbitrary covariance matrices)



$p$=Blue, $q^*$=Red

ELBO is a lower bound on the (log) evidence. Maximizing the ELBO is the same as minimizing $KL(q_\phi(z)\|p(z|x))$.

$$KL(q_\phi(z)\|p(z|x)) = \mathop{\mathbb{E}}_{z\sim q_\phi} \log \frac{q_\phi(z)}{p(z|x)} = \mathop{\mathbb{E}}_{z\sim q_\phi}\left[\log\left(q_\phi(z)\cdot\frac{p(x)}{p(z,x)}\right)\right]$$

$$= \mathop{\mathbb{E}}_{z\sim q_\phi}\left[\log\frac{q_\phi(z)}{p(z,x)}\right] + \mathop{\mathbb{E}}_{z\sim q_\phi}\log p(x) := -\mathcal{L}(\phi) + \log p(x).$$

Where $\mathcal{L}(\phi)$ is the **ELBO**: $\mathcal{L}(\phi) = \mathop{\mathbb{E}}_{z\sim q_\phi}\Big[\log p(z,x) - \log q_\phi(z)\Big]$.

- Because $KL(q_\phi(z)\|p(z|x)) \geq 0$,

$$\mathcal{L}(\phi) \leq \log p(x)$$

- maximizing the ELBO $\Rightarrow$ minimizing $KL(q_\phi(z)\|p(z|x))$.

- Full dataset $\{\mathbf{X}, \mathbf{Z}\}$ but we only observe $\{\mathbf{X}\}$. The model for $(x, z)$ is tractable.
- Our knowledge about the latent variables is given only by the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \theta)$.
- Because we cannot use the complete data log-likelihood, we can consider expected complete-data log-likelihood:

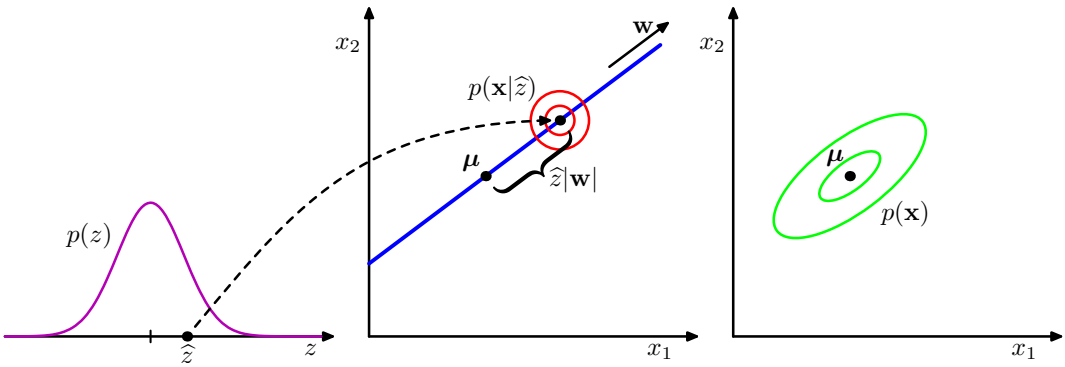
$$\mathcal{Q}(\theta, \theta^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{old}) \log p(\mathbf{X}, \mathbf{Z}|\theta)$$

- In the E-step, we use the current parameters $\theta^{old}$ to compute the posterior over the latent variables $p(\mathbf{Z}|\mathbf{X}, \theta^{old})$.
- In the M-step, we find the revised parameter estimate $\theta$ new by maximizing the expected complete log-likelihood:

$$\theta^{new} = \arg\max_{\theta} \mathcal{Q}(\theta, \theta^{old})$$

Assume continuous, Gaussian latent variables:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \qquad \mathbf{x} \,|\, \mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2 \mathbf{I})$$



- $p(\mathbf{x} \mid \mathbf{W}, \boldsymbol{\mu}, \sigma^2)$ will be Gaussian (confirm this), so we just need

$$\mathbb{E}[\mathbf{x}] = \mathbb{E}[\mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \epsilon] = \boldsymbol{\mu}$$

$$\begin{aligned} \mathrm{Cov}[\mathbf{x}] &= \mathbb{E}[(\mathbf{W}\mathbf{z} + \epsilon)(\mathbf{W}\mathbf{z} + \epsilon)^\top] \\ &= \mathbb{E}[\mathbf{W}\mathbf{z}\mathbf{z}^\top\mathbf{W}^\top] + \mathrm{Cov}[\epsilon] \\ &= \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I} \end{aligned}$$

- To perform MLE, we need to maximize the following:

$$\max_{\mathbf{W},\boldsymbol{\mu},\sigma^2} \log p(\mathbf{x} \mid \mathbf{W}, \boldsymbol{\mu}, \sigma^2) = \max_{\mathbf{W},\boldsymbol{\mu},\sigma^2} \log \int p(\mathbf{x} \mid \mathbf{z}, \mathbf{W}, \boldsymbol{\mu}, \sigma^2) p(\mathbf{z}) \, d\mathbf{z}$$

- The MLE is given in a closed and easily interpretable form.

- **Prior distribution:** $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{S})$
- **Model (Likelihood):** $y \mid \mathbf{x}, \mathbf{w} \sim \mathcal{N}(\mathbf{w}^\top \psi(\mathbf{x}),\ \sigma^2)$
- Assuming fixed/known $\mathbf{S}$ and $\sigma^2$.

- **Deriving the posterior distribution:**

$$\begin{aligned}
\log p(\mathbf{w} \mid \mathcal{D}) &= \log p(\mathbf{w}) + \log p(\mathcal{D} \mid \mathbf{w}) + \text{const} \\
&= -\tfrac{1}{2}\mathbf{w}^\top \mathbf{S}^{-1}\mathbf{w} - \frac{1}{2\sigma^2}\|\boldsymbol{\Psi}\mathbf{w} - \mathbf{y}\|^2 + \text{const} \\
&= -\tfrac{1}{2}\mathbf{w}^\top \mathbf{S}^{-1}\mathbf{w} - \frac{1}{2\sigma^2}\left(\mathbf{w}^\top \boldsymbol{\Psi}^\top \boldsymbol{\Psi}\mathbf{w} - 2\mathbf{y}^\top \boldsymbol{\Psi}\mathbf{w} + \mathbf{y}^\top \mathbf{y}\right) + \text{const} \\
&= -\tfrac{1}{2}\mathbf{w}^\top \left(\sigma^{-2}\boldsymbol{\Psi}^\top \boldsymbol{\Psi} + \mathbf{S}^{-1}\right)\mathbf{w} + \frac{1}{\sigma^2}\mathbf{y}^\top \boldsymbol{\Psi}\mathbf{w} + \text{const} \ \text{(complete the square)}
\end{aligned}$$

Thus $\mathbf{w} \mid \mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Psi}^\top \mathbf{y}, \qquad \boldsymbol{\Sigma} = \left(\sigma^{-2}\boldsymbol{\Psi}^\top \boldsymbol{\Psi} + \mathbf{S}^{-1}\right)^{-1}$$

Linear model: $t \,|\, \mathbf{x} \sim \mathcal{N}(y(\mathbf{x}),\ \sigma^2) \qquad y(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\psi}(\mathbf{x})$

- Given $N$ samples, we have: $\mathbf{t} \,|\, \mathbf{y} \sim \mathcal{N}(\mathbf{y},\ \sigma^2 I)$
- Since $\mathbf{y}$ is a Gaussian process, we have $\mathbf{y} \sim \mathcal{N}(0, \boldsymbol{K})$

$$K_{ij} = \frac{1}{\alpha} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \frac{1}{\alpha} \boldsymbol{\psi}(\mathbf{x}^{(i)})^T \boldsymbol{\psi}(\mathbf{x}^{(j)})$$

- Therefore the marginal of $\mathbf{t}$ is given by

$$\mathbf{t} \sim \mathcal{N}(0, \boldsymbol{C}) \qquad \boldsymbol{C} = \boldsymbol{K} + \sigma^2 I$$

where

$$C(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \frac{1}{\alpha} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + \sigma^2 \delta_{ij}$$

$\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$.

In the lecture we denoted $t$ as $y$ and $y$ as $\hat{y}$. This notation is more standard in ML.

- Let's define $\mathbf{t}_N = (t^{(1)}, t^{(2)}, ..., t^{(N)})^T$.
- We have the marginal of $\mathbf{t}_N$ given by

$$\mathbf{t}_N \sim \mathcal{N}(0, \boldsymbol{C}_N) \qquad \boldsymbol{C}_N = \boldsymbol{K}_N + \sigma^2 I$$

  where $C_N(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \frac{1}{\alpha} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + \sigma^2 \delta_{ij}$.
- This reflects the two Gaussian sources of randomness.
- **Goal in regression:** We want to predict for a new input $\mathbf{x}^{(N+1)}$. We need

$$p(t^{(N+1)} \,|\, \mathbf{t}_N)$$

- Note that $\mathbf{x}^{(i)}$'s are treated as constants.

- We have

$$\mathbf{t}_{N+1} \sim \mathcal{N}(0, \boldsymbol{C}_{N+1}) \qquad \boldsymbol{C}_{N+1} = \boldsymbol{K}_{N+1} + \sigma^2 I$$

where

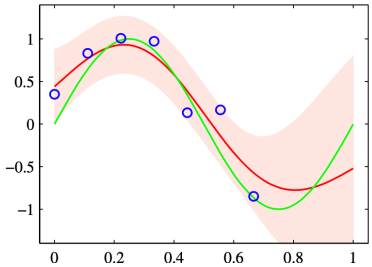
$$C_{N+1}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \frac{1}{\alpha} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + \sigma^2 \delta_{ij}$$

$$\boldsymbol{C}_{N+1} = \begin{bmatrix} \boldsymbol{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{bmatrix}.$$

  - Here, $c = \frac{1}{\alpha} k(\mathbf{x}^{(N+1)}, \mathbf{x}^{(N+1)}) + \sigma^2$
  - $\mathbf{k}$ is a vector with entries $k_i = \frac{1}{\alpha} k(\mathbf{x}^{(i)}, \mathbf{x}^{(N+1)})$

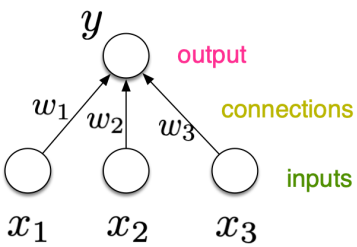
- Since $\mathbf{t}_{N+1}$ is multivariate Gaussian, $t^{(N+1)} \,|\, \mathbf{t}_N$ is also Gaussian with mean and covariance

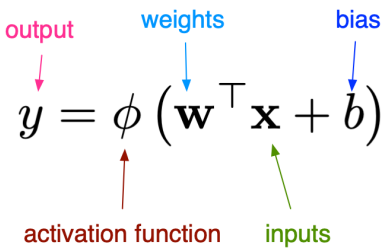
$$m(\mathbf{x}^{(N+1)}) = \mathbf{k}^T \boldsymbol{C}_N^{-1} \mathbf{t}_N \qquad \sigma^2(\mathbf{x}^{(N+1)}) = c - \mathbf{k}^T \boldsymbol{C}_N^{-1} \mathbf{k}$$




- The green curve: the true sinusoid from which the data points, shown in blue, are obtained with additional of Gaussian noise.
- The red line: mean of the Gaussian process predictive distribution.
- The shaded region: plus and minus two standard deviations.

For neural nets, we use a simple model for neuron, or **unit**:




- Same as logistic regression: $y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$
- By throwing together lots of these simple neuron-like processing units, we can do some powerful computations!

# Week 12: Computation in Each Layer

Each layer computes a function.

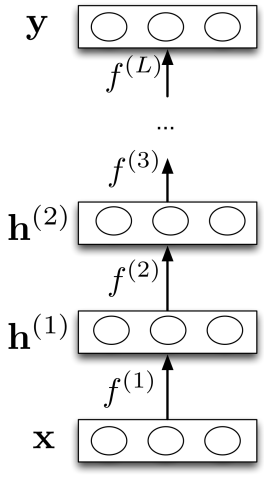
$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}) = \phi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)}) = \phi(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$

The network computes a composition of functions.

$$\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

The last layer depends on the task.

- Regression: $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = (\mathbf{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)}$
- Classification: $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = \sigma\big((\mathbf{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)}\big)$



Learning parameters in NN is typically done with SGD, where the gradient is computed using backpropagation.

Let $v_1, \ldots, v_N$ be an ordering of the computation graph where parents come before children. $v_N$ denotes the variable for which we try to compute gradients ($\mathcal{L}$, $\mathcal{L}_{\text{reg}}$ etc).

- forward pass:

$$\text{For } i = 1, \ldots, N,$$
$$\text{Compute } v_i \text{ as a function of Parents}(v_i).$$

- backward pass (denote $\bar{v}_i = \frac{\partial v_N}{\partial v_i}$): start setting $\bar{v}_N = 1$

$$\text{Then for } i = N-1, \ldots, 1: \qquad \bar{v}_i = \sum_{j \in \text{Children}(v_i)} \bar{v}_j \frac{\partial v_j}{\partial v_i}$$

# High level summary

We focused in this course on probabilistic models that scale to big examples.

Some remarks:

- Graphical representations of models allow for structured and scalable aproaches.
- Conditional independence is interpretable and provides computational advantages.
- Incorporating latent variables increases flexibility with tiny computational cost.
- Variational inference and other techniques based on EFs have been useful.
- In modern ML algorithms models and computation cannot be separated.
- Many new approaches in ML creatively recycle old methods.

# Closing remarks

Continuing with machine learning:

- Courses
  - CSC 421/2516, Neural Networks and Deep Learning
  - CSC 2515, Machine Learning
  - CSC 2532, Statistical Learning Theory
  - CSC 2541, "Neural Network Training Dynamics"
  - Topics courses (varies from year to year): Reinforcement Learning, Algorithmic Fairness, Computer Vision w/ ML, NLP w/ ML, Health w/ ML etc.
  - Learn Statistics!
- Videos from top ML conferences (NeurIPS, ICML, ICLR)
- Try to reproduce results from papers
  - If they released code, you can use that as a guide if you get stuck
- Lots of excellent free resources available online!

# Study for the final!

- Review lectures.
- Understand derivations (ask, check textbooks).
- Solve the practice final.
- Fill out course evaluations!
- Good luck!