Advanced Machine Learning Summer 2019

Part 20 – Repetition
11.07.2019

Prof. Dr. Bastian Leibe

RWTH Aachen University, Computer Vision Group
http://www.vision.rwth-aachen.de

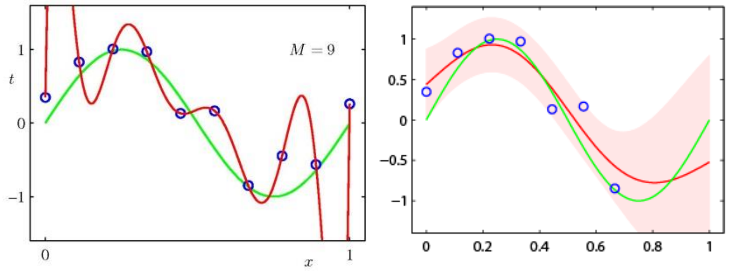
Announcements

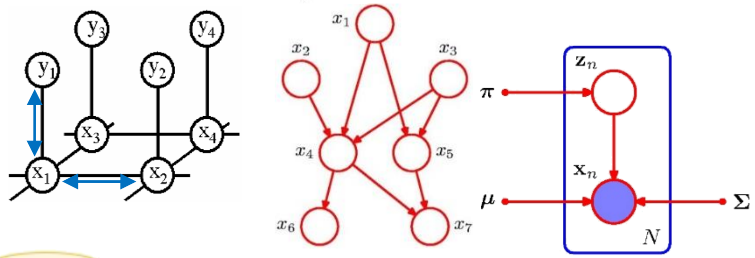
- Today, I'll summarize the most important points from the lecture.
  - It is an opportunity for you to ask questions…
  - …or get additional explanations about certain topics.
  - *So, please do ask.*
- Today's slides are intended as an index for the lecture.
  - Summarizing the most important points from each class
  - But they are not complete, won't be sufficient as only tool.
  - Also look at the exercises – they often explain algorithms in detail.
- Exam procedure
  - Closed-book exam, the core exam time will be 2h.
  - We will send around an announcement with the exact starting times and places by email.

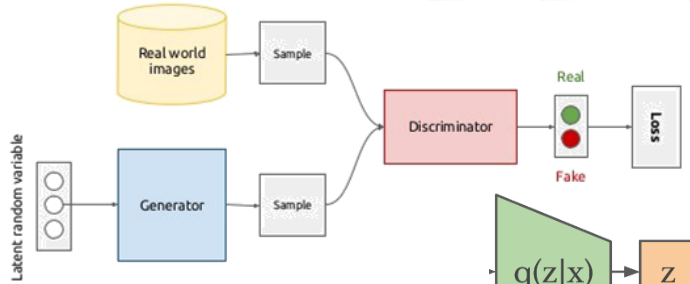
Course Outline

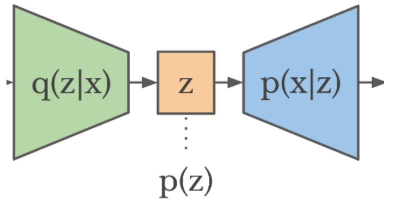
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \to \mathbb{R}$$




y_1 y_2 y_3 y_4
x_1 x_2 x_3 x_4
x_1 x_2 x_3 x_4 x_5 x_6 x_7
$\mathbf{z}_n$ $\mathbf{x}_n$ π μ Σ N

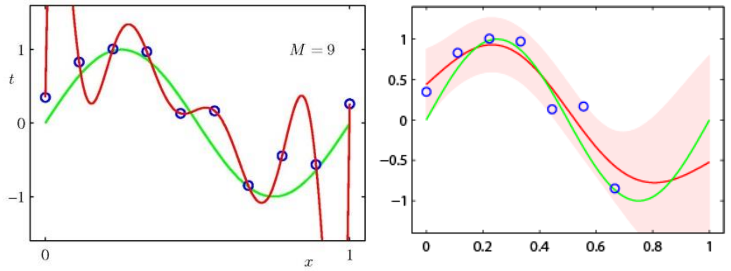


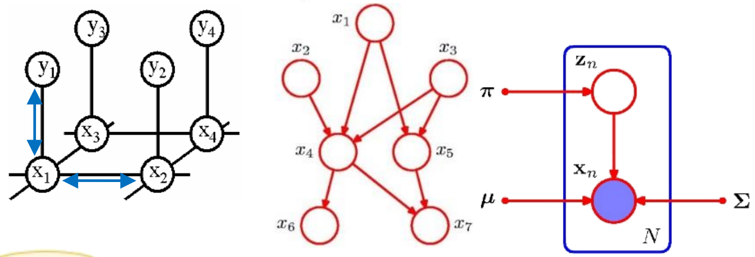


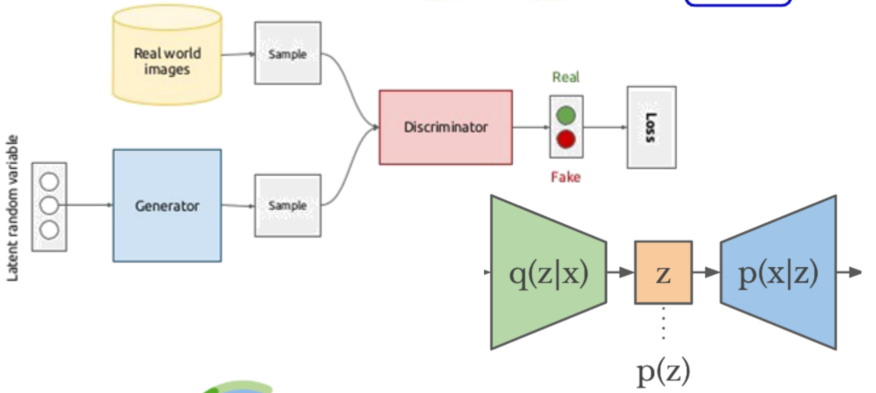
Course Outline

- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \rightarrow \mathbb{R}$$





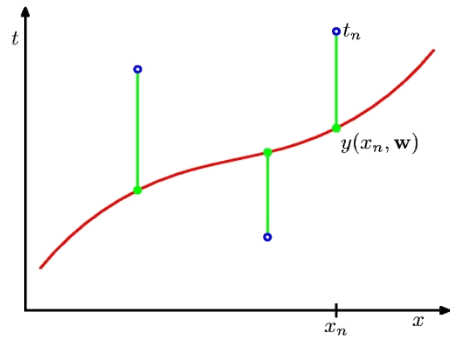


Recap: Regression

- Learning to predict a continuous function value
  - Given: training set $\mathbf{X} = \{x_1, \ldots, x_N\}$ with target values $\mathbf{T} = \{t_1, \ldots, t_N\}$.
  - $\Rightarrow$ Learn a continuous function $y(x)$ to predict the function value for a new input x.

- Define an error function $E(\mathbf{w})$ to optimize
  - E.g., sum-of-squares error

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$



  - Procedure: Take the derivative and set it to zero

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\} \frac{\partial y(x_n, \mathbf{w})}{\partial w_j} \overset{!}{=} 0$$

Image source: C.M. Bishop, 2006

Recap: Least-Squares Regression

see Exercise 1.3

$$\mathbf{x}_i^T \mathbf{w} + w_0 = t_i, \quad \forall i = 1, \dots, n$$

- Setup
  - Step 1: Define $\tilde{\mathbf{x}}_i = \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}, \quad \tilde{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ w_0 \end{pmatrix}$
  - Step 2: Rewrite $\tilde{\mathbf{x}}_i^T \tilde{\mathbf{w}} = t_i, \quad \forall i = 1, \dots, n$
  - Step 3: Matrix-vector notation

$$\widetilde{\mathbf{X}}^T \tilde{\mathbf{w}} = \mathbf{t} \quad \text{with} \quad \begin{aligned} \widetilde{\mathbf{X}} &= [\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_n] \\ \mathbf{t} &= [t_1, \dots, t_n]^T \end{aligned}$$

  - Step 4: Find least-squares solution

$$\|\widetilde{\mathbf{X}}^T \tilde{\mathbf{w}} - \mathbf{t}\|^2 \rightarrow \min$$

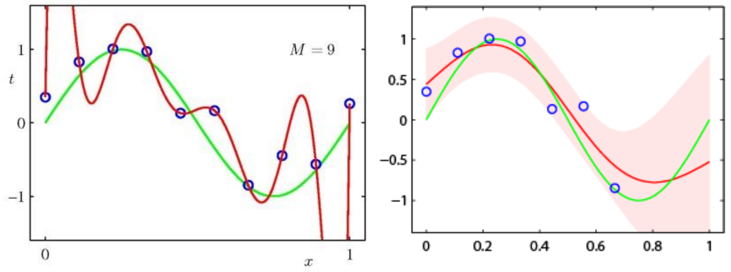
  - Solution: $\tilde{\mathbf{w}} = (\widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^T)^{-1} \widetilde{\mathbf{X}} \mathbf{t}$

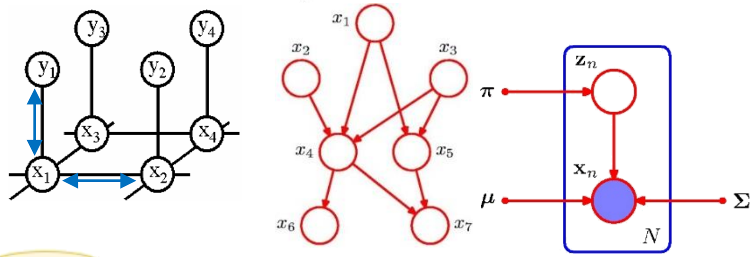
Slide credit: Bernt Schiele

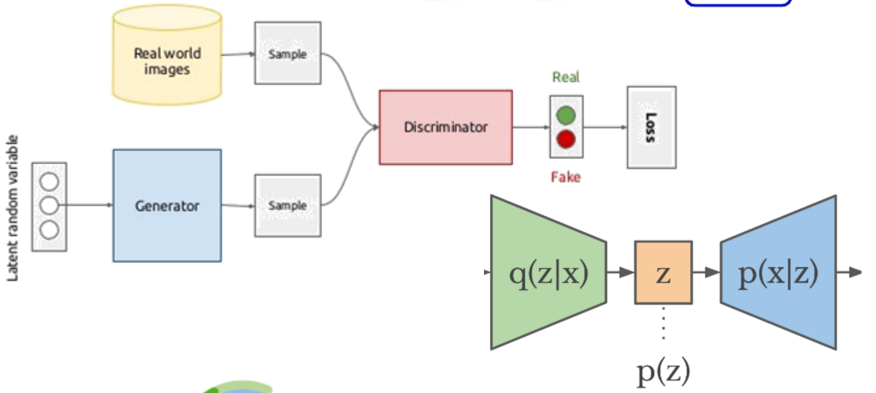
Course Outline

- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \rightarrow \mathbb{R}$$







Recap: Regularization

- Problem: Overfitting
  - Many parameters & little data $\Rightarrow$ tendency to overfit to the noise
  - Side effect: The coefficient values get very large.

- Workaround: Regularization
  - Penalize large coefficient values

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

  - Here we've simply added a quadratic regularizer, which is simple to optimize

$$\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w} = \cancel{w_0^2} + w_1^2 + \ldots + w_M^2$$

  - The resulting form of the problem is called Ridge Regression.
  - (*Note*: w_0 is often omitted from the regularizer.)

Recap: Probabilistic Regression

- First assumption:
  - Our target function values y are generated by adding noise to the function estimate:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon$$

Target function value

Regression function

Input value

Weights or parameters

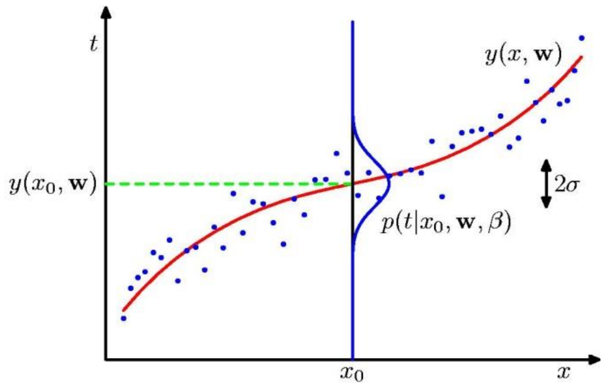
Noise

- Second assumption:
  - The noise is Gaussian distributed

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

Mean

Variance (β precision)



Slide credit: Bernt Schiele

Recap: Maximum Likelihood Regression

- Given
  - Training data points: $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$
  - Associated function values: $\mathbf{t} = [t_1, \ldots, t_n]^T$

- Conditional likelihood (assuming i.i.d. data)

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n | y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) = \prod_{n=1}^{N} \mathcal{N}(t_n | \underbrace{\mathbf{w}^T \phi(\mathbf{x}_n)}_{\text{Generalized linear regression function}}, \beta^{-1})$$

$\Rightarrow$ Maximize w.r.t. $\mathbf{w}$, β

Slide credit: Bernt Schiele

Recap: Maximum Likelihood Regression

$$\nabla_{\mathbf{w}} \log p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) \;=\; -\beta \sum_{n=1}^{N} (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

- Setting the gradient to zero:

$$0 \;=\; -\beta \sum_{n=1}^{N} (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

$$\Leftrightarrow \quad \sum_{n=1}^{N} t_n \phi(\mathbf{x}_n) = \left[\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\right] \mathbf{w}$$

$$\Leftrightarrow \quad \mathbf{\Phi t} = \mathbf{\Phi \Phi}^T \mathbf{w} \qquad\qquad \mathbf{\Phi} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$$

$$\Leftrightarrow \quad \mathbf{w}_{\mathrm{ML}} = (\mathbf{\Phi \Phi}^T)^{-1} \mathbf{\Phi t} \longleftarrow$$

Same as in least-squares regression!

$\Rightarrow$ *Least-squares regression is equivalent to Maximum Likelihood under the assumption of Gaussian noise.*

Slide credit: Bernt Schiele

Recap: Role of the Precision Parameter

- Also use ML to determine the precision parameter β:

$$\log p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta) = -\frac{\beta}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \frac{N}{2}\log\beta - \frac{N}{2}\log(2\pi)$$

- Gradient w.r.t. β:

$$\nabla_\beta \log p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta) = -\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \frac{N}{2}\frac{1}{\beta}$$

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2$$

$\Rightarrow$ *The inverse of the noise precision is given by the residual variance of the target values around the regression function.*

Recap: Predictive Distribution

- Having determined the parameters $\mathbf{w}$ and β, we can now make predictions for new values of $\mathbf{x}$.

$$p(t|\mathbf{X}, \mathbf{w}_{\mathrm{ML}}, \beta_{\mathrm{ML}}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{\mathrm{ML}}), \beta_{\mathrm{ML}}^{-1})$$

- This means
  - Rather than giving a point estimate, we can now also give an estimate of the estimation uncertainty.

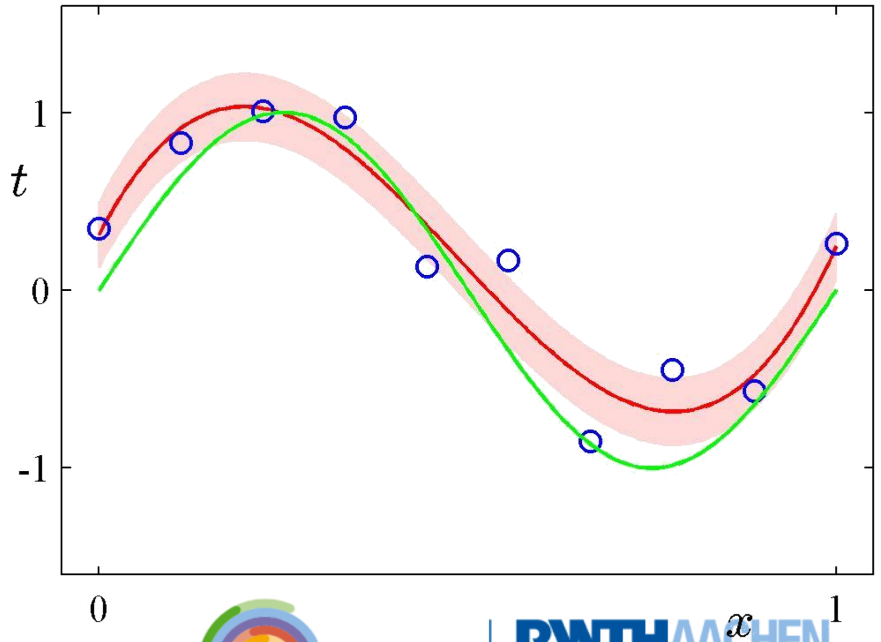


Image source: C.M. Bishop, 2006

Recap: Maximum-A-Posteriori Estimation

- Introduce a prior distribution over the coefficients $\mathbf{w}$.
  - For simplicity, assume a zero-mean Gaussian distribution

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\}$$

  - New hyperparameter α controls the distribution of model parameters.

- Express the posterior distribution over $\mathbf{w}$.
  - Using Bayes' theorem:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \beta, \alpha) \propto p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$$

  - We can now determine $\mathbf{w}$ by maximizing the posterior.
  - This technique is called maximum-a-posteriori (MAP).

Recap: MAP Solution

- Minimize the negative logarithm

$$-\log p(\mathbf{w}|\mathbf{X},\mathbf{t},\beta,\alpha) \propto \underline{-\log p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta)} \underline{-\log p(\mathbf{w}|\alpha)}$$

$$\underline{-\log p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta)} = \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-t_n\}^2 + \text{const}$$

$$\underline{-\log p(\mathbf{w}|\alpha)} = \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + \text{const}$$

- The MAP solution is therefore the solution of

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w})-t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

$\Rightarrow$ *Maximizing the posterior distribution is equivalent to minimizing the regularized sum-of-squares error (with* $\lambda = \frac{\alpha}{\beta}$*).*

Recap: MAP Solution (2)

$$\nabla_{\mathbf{w}} \log p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \beta, \alpha) \;=\; -\beta \sum_{n=1}^{N} (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))\phi(\mathbf{x}_n) + \alpha \mathbf{w}$$

- Setting the gradient to zero:

$$0 \;=\; -\beta \sum_{n=1}^{N} (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))\phi(\mathbf{x}_n) + \alpha \mathbf{w}$$

$$\Leftrightarrow \;\; \sum_{n=1}^{N} t_n \phi(\mathbf{x}_n) = \left[\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T\right] \mathbf{w} + \frac{\alpha}{\beta}\mathbf{w}$$

$$\Leftrightarrow \;\; \mathbf{\Phi t} = \left(\mathbf{\Phi\Phi}^T + \frac{\alpha}{\beta}\mathbf{I}\right)\mathbf{w} \qquad\qquad \mathbf{\Phi} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$$

$$\Leftrightarrow \;\; \mathbf{w}_{\mathrm{MAP}} = \left(\mathbf{\Phi\Phi}^T + \frac{\alpha}{\beta}\mathbf{I}\right)^{-1} \mathbf{\Phi t}$$

Effect of regularization: Keeps the inverse well-conditioned

Recap: Bayesian Curve Fitting

- Given
  - Training data points: $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$
  - Associated function values: $\mathbf{t} = [t_1, \ldots, t_n]^T$
  - Our goal is to predict the value of t for a new point $\mathbf{x}$.
- Evaluate the predictive distribution

$$p(t|x, \mathbf{X}, \mathbf{t}) = \int \underbrace{p(t|x, \mathbf{w})}_{} \underbrace{p(\mathbf{w}|\mathbf{X}, \mathbf{t})}_{\text{What we just computed for MAP}} d\mathbf{w}$$

  - Noise distribution – again assume a Gaussian here

$$p(t|x, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

  - Assume that parameters α and β are fixed and known for now.

Recap: Bayesian Curve Fitting

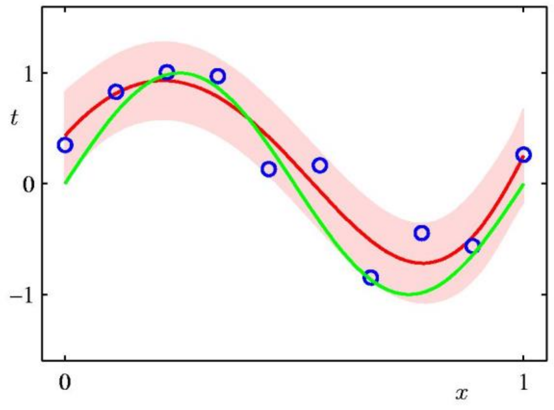
- Under those assumptions, the posterior distribution is a Gaussian and can be evaluated analytically:

$$p(t|x, \mathbf{X}, \mathbf{t}) = \mathcal{N}(t|m(x), s^2(x))$$

  - where the mean and variance are given by

$$m(x) = \beta\phi(x)^T \mathbf{S} \sum_{n=1}^{N} \phi(\mathbf{x}_n) t_n$$

$$s(x)^2 = \beta^{-1} + \phi(x)^T \mathbf{S} \phi(x)$$

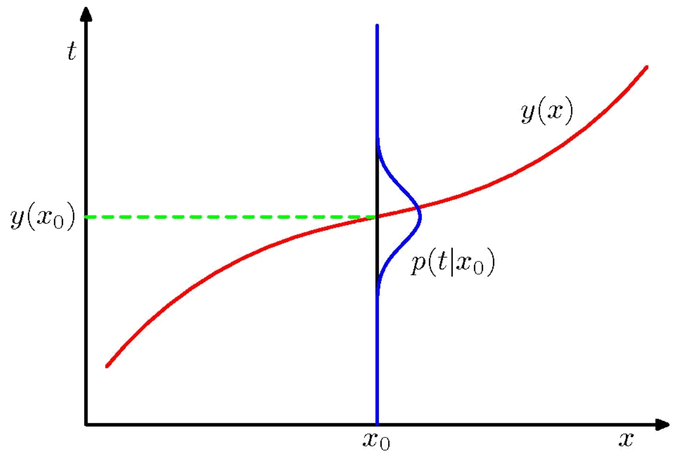


  - and $\mathbf{S}$ is the regularized covariance matrix

$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta \sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T$$

Image source: C.M. Bishop, 2006

Recap: Loss Functions for Regression



Mean prediction

- Optimal prediction
  - Minimize the expected loss

$$\mathbb{E}[L] = \iint L(t, y(\mathbf{x}))p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$

  - Under squared loss, the optimal regression function is the mean $\mathbb{E}[t|\mathbf{x}]$ of the posterior $p(t|\mathbf{x})$ ("mean prediction").
  - For generalized linear regression function and squared loss:

$$y(\mathbf{x}) = \int t\mathcal{N}(t|\mathbf{w}^T\phi(\mathbf{x}), \beta^{-1})\mathrm{d}t = \mathbf{w}^T\phi(\mathbf{x})$$

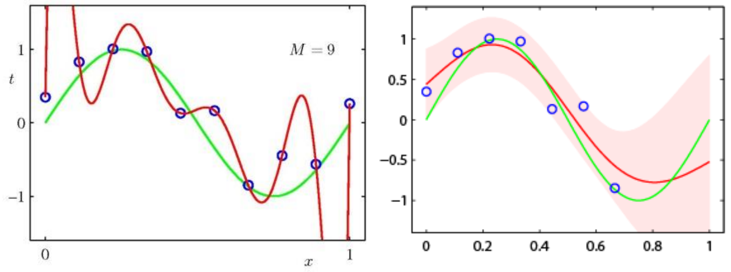
Slide adapted from Stefan Roth

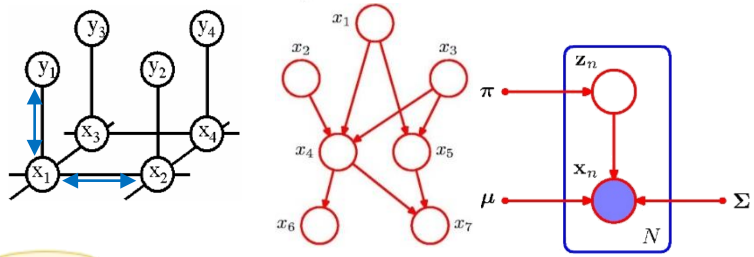
Image source: C.M. Bishop, 2006

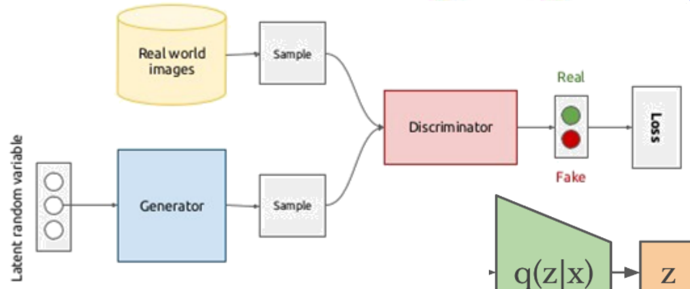
Course Outline

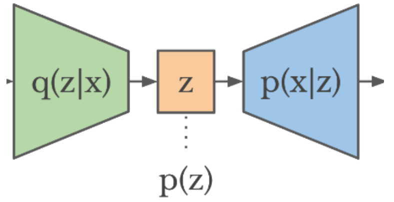
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \rightarrow \mathbb{R}$$




y_1 y_2 y_3 y_4 x_1 x_2 x_3 x_4
x_1 x_2 x_3 x_4 x_5 x_6 x_7
π $\mathbf{z}_n$ $\mathbf{x}_n$ μ Σ N





Recap: Loss Functions for Regression

see Exercise 1.2

- The squared loss is not the only possible choice
  - Poor choice when conditional distribution $p(t|\mathbf{x})$ is multimodal.

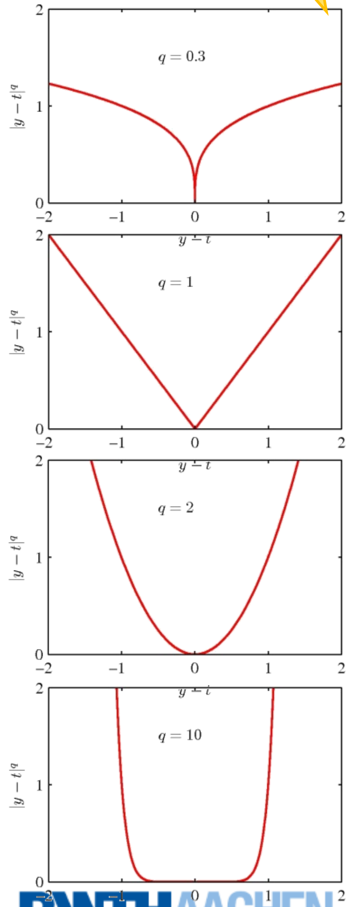
- Simple generalization: Minkowski loss

$$L(t, y(\mathbf{x})) = |y(\mathbf{x}) - t|^q$$

  - Expectation

$$\mathbb{E}[L_q] = \iint |y(\mathbf{x}) - t|^q p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t$$

- Minimum of $\mathbb{E}[L_q]$ is given by
  - Conditional mean for $q = 2$,
  - Conditional median for $q = 1$,
  - Conditional mode for $q = 0$.



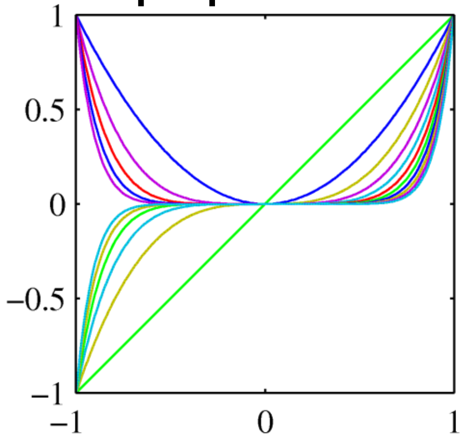
Recap: Linear Basis Function Models

- Generally, we consider models of the following form

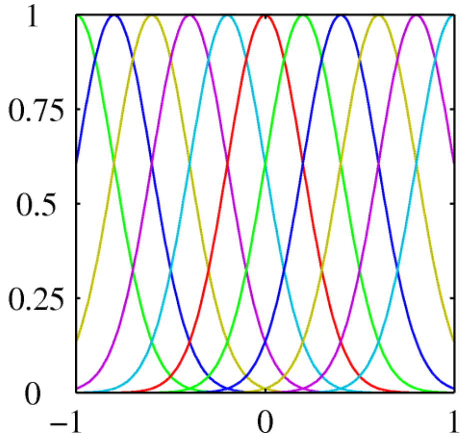
$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

  - where $\phi_j(\mathbf{x})$ are known as *basis functions*.
  - In the simplest case, we use linear basis functions: $\phi_d(\mathbf{x}) = x_d$.

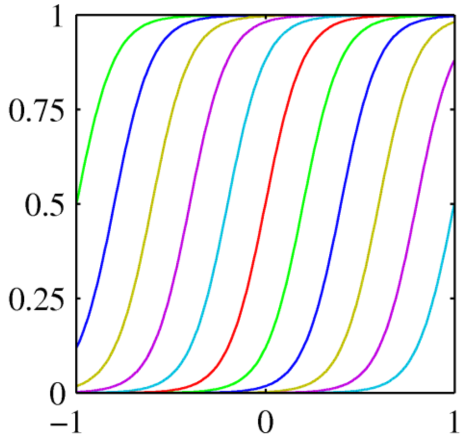
- Other popular basis functions



Polynomial



Gaussian

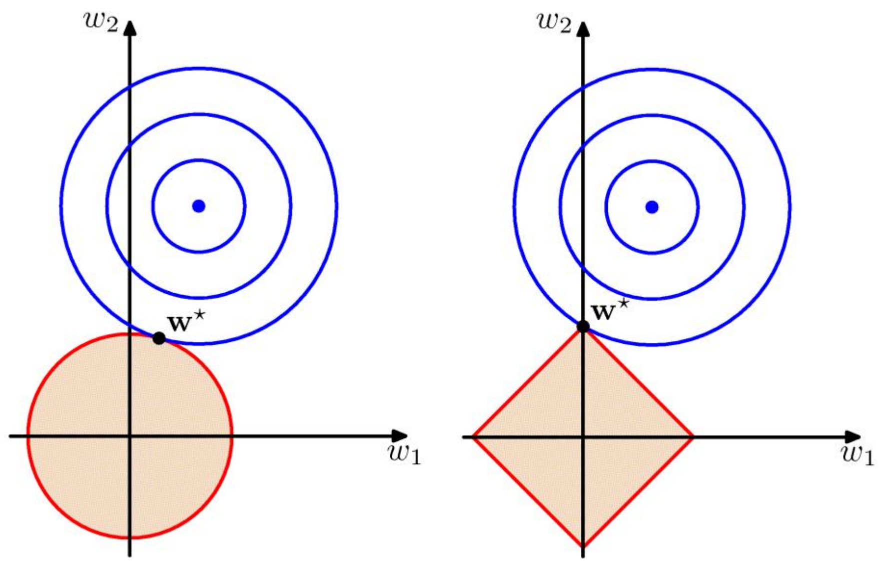


Sigmoid

Recap: Regularized Least-Squares

- Consider more general regularization functions
  - "L$_q$ norms":

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



- Effect: Sparsity for $q \leq 1$.
  - Minimization tends to set many coefficients to zero

Image source: C.M. Bishop, 2006

Recap: Lasso as Bayes Estimation

- L_1 regularization ("The Lasso")

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{2} \sum_{n=1}^{N} \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \lambda \sum_{j=1}^{M} |w_j|$$

- Interpretation as Bayes Estimation
  - We can think of $|w_j|^q$ as the log-prior density for w_j.

- Prior for Lasso ($q = 1$): Laplacian distribution

$$p(\mathbf{w}) = \frac{1}{2\tau} \exp\{-|\mathbf{w}|/\tau\} \qquad \text{with} \qquad \tau = \frac{1}{\lambda}$$

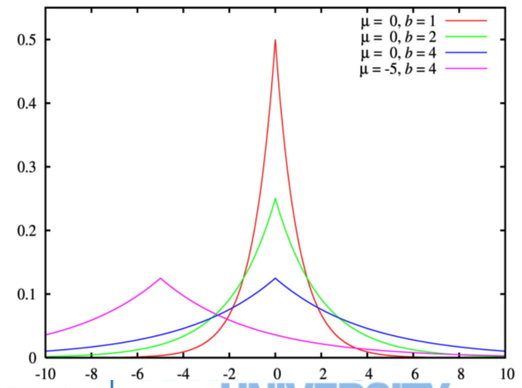


Image source: Wikipedia

Recap: The Lasso

- L_1 regularization ("The Lasso")

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \lambda\sum_{j=1}^{M}|w_j|$$

  – The solution will be sparse (only few coefficients non-zero)
  – The L_1 penalty makes the problem non-linear.
  $\Rightarrow$ There is no closed-form solution.

- Interpretation as Bayes Estimation
  – We can think of $|w_j|^q$ as the log-prior density for w_j.

- Prior for Lasso ($q = 1$):
  – Laplacian distribution

$$p(\mathbf{w}) = \frac{1}{2\tau}\exp\{-|\mathbf{w}|/\tau\} \quad \text{with} \quad \tau = \frac{1}{\lambda}$$

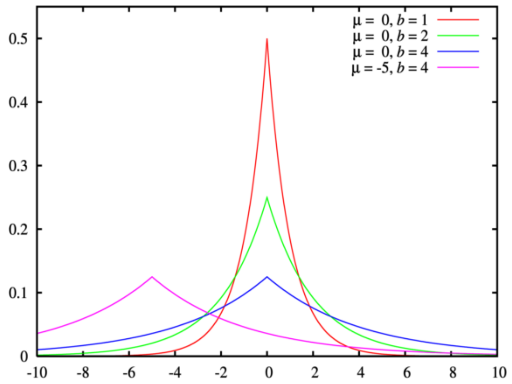
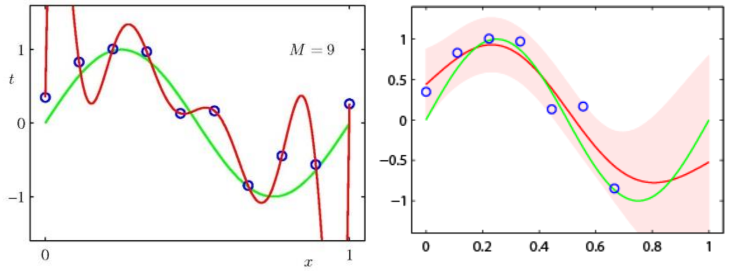


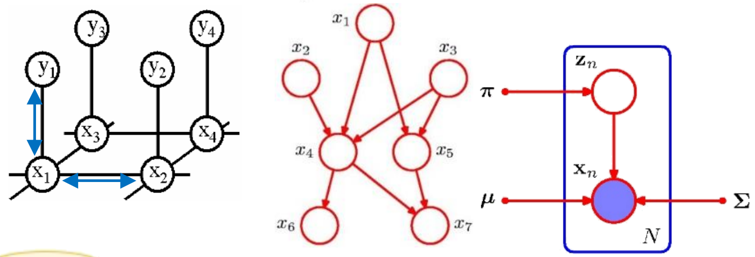
Image source: Wikipedia

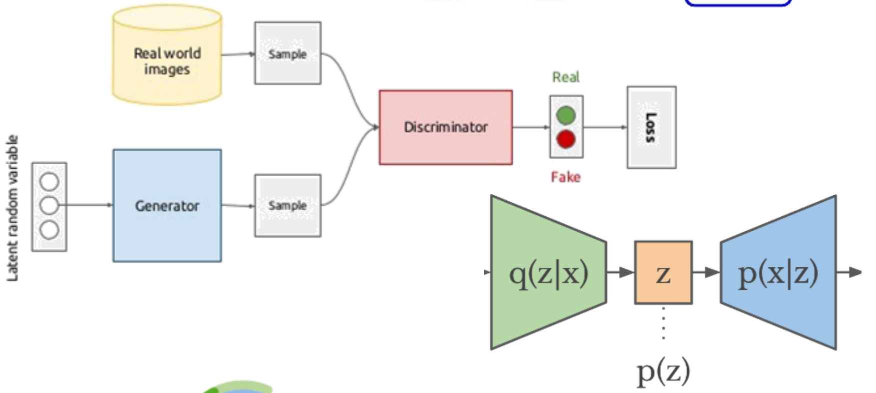
Course Outline

- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \to \mathbb{R}$$







Recap: Kernel Ridge Regression

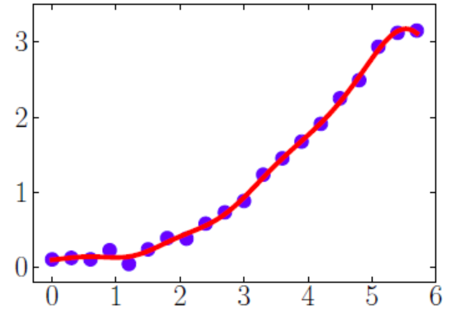
see Exercise 1.4

- Dual definition
  - Instead of working with $\mathbf{w}$, substitute $\mathbf{w} = \mathbf{\Phi}^T\mathbf{a}$ into $J(\mathbf{w})$ and write the result using the kernel matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}$$

  - Solving for $\mathbf{a}$, we obtain

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$



- Prediction for a new input $\mathbf{x}$:
  - Writing $\mathbf{k}(\mathbf{x})$ for the vector with elements $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$

$\Rightarrow$ *The dual formulation allows the solution to be entirely expressed in terms of the kernel function* $k(\mathbf{x},\mathbf{x}')$.

Image source: Christoph Lampert

Recap: Properties of Kernels

- Theorem
  - *Let k: $\mathcal{X}$ × $\mathcal{X} \to \mathbb{R}$ be a positive definite kernel function. Then there exists a Hilbert Space $\mathcal{H}$ and a mapping φ : $\mathcal{X} \to \mathcal{H}$ such that*

$$k(x, x') = \langle(\phi(x), \phi(x')\rangle_{\mathcal{H}}$$

  - where $\langle . , . \rangle_{\mathcal{H}}$ *is the inner product in* H*.*
- Translation
  - Take *any* set $\mathcal{X}$ and *any* function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.
  - If k is a positive definite kernel, then we can use k to learn a classifier for the elements in $\mathcal{X}$!
- Note
  - $\mathcal{X}$ can be any set, e.g. $\mathcal{X}$ = *"all videos on YouTube"* or $\mathcal{X}$ = *"all permutations of* {1, . . . , *k*}", or $\mathcal{X}$ = *"the internet"*.

Slide credit: Christoph Lampert

Recap: The “Kernel Trick”

Any algorithm that uses data only in the form of inner products can be *kernelized*.

- How to kernelize an algorithm
  - Write the algorithm only in terms of inner products.
  - Replace all inner products by kernel function evaluations.

$\Rightarrow$ The resulting algorithm will do the same as the linear version, but in the (hidden) feature space $\mathcal{H}$.

- Caveat: working in $\mathcal{H}$ is not a guarantee for better performance. A good choice of k and model selection are important!

Slide credit: Christoph Lampert

Recap: How to Check if a Function is a Kernel

- Problem:
  - Checking if a given $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ fulfills the conditions for a kernel is difficult:
  - We need to prove or disprove

$$\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j \geq 0$$

for any set $x_1, \ldots, x_n \in \mathcal{X}$ and any $\mathbf{t} \in \mathbb{R}^n$ for any $n \in N$.

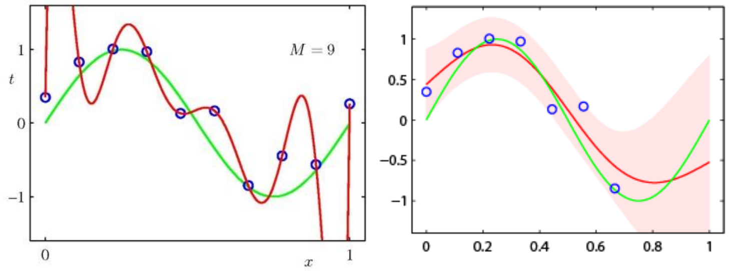
- Workaround:
  - It is easy to construct functions k that are positive definite kernels.

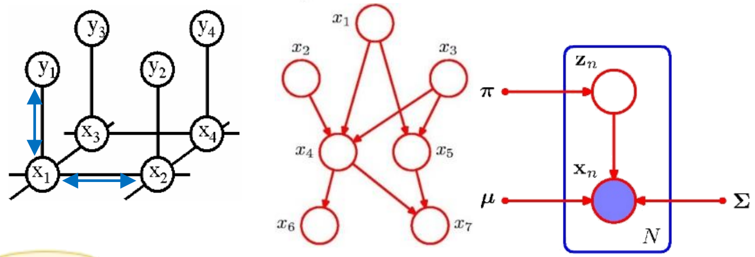
Slide credit: Christoph Lampert

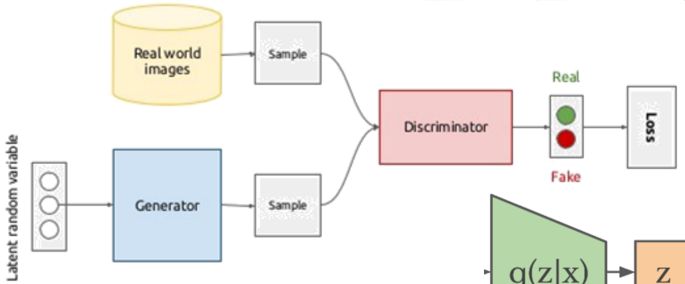
Course Outline

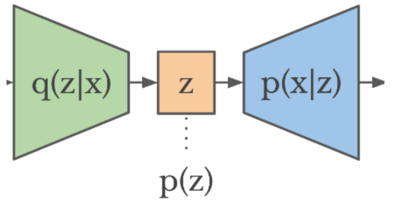
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \to \mathbb{R}$$



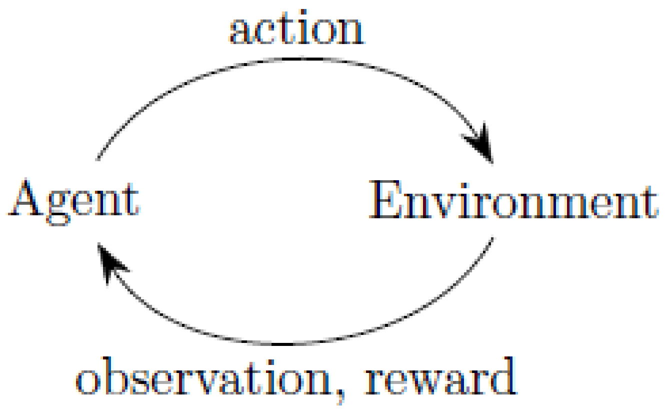






Recap: Reinforcement Learning

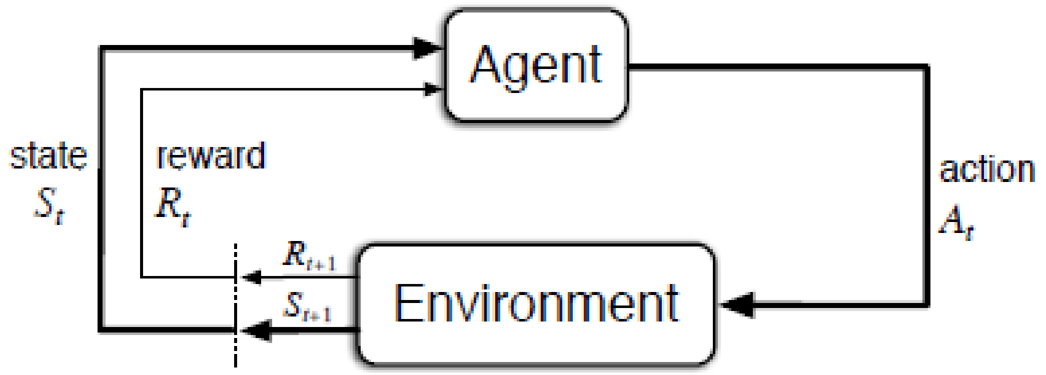
- Motivation
  - General purpose framework for decision making.
  - Basis: Agent with the capability to interact with its environment
  - Each action influences the agent's future state.
  - Success is measured by a scalar reward signal.
  - Goal: select actions to maximize future rewards.



  - Formalized as a partially observable Markov decision process (POMDP)

Slide adapted from: David Silver, Sergey Levine

Recap: The Agent–Environment Interface



- Let's formalize this
  - Agent and environment interact at discrete time steps $t = 0, 1, 2, \ldots$
  - Agent observes state at time t: $S_t \in \mathcal{S}$
  - Produces an action at time t: $A_t \in \mathcal{A}(S_t)$
  - Gets a resulting reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
  - And a resulting next state: S_{t+1}

Slide adapted from: Sutton & Barto

Recap: Reward vs. Return

- Objective of learning
  - We seek to maximize the expected return G_t as some function of the reward sequence $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
  - Standard choice: expected discounted return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  where $0 \leq \gamma \leq 1$ is called the discount rate.

- Difficulty
  - We don't know which past actions caused the reward.

  $\Rightarrow$ Temporal credit assignment problem

Recap: Markov Decision Process (MDP)

- Markov Decision Processes
  - We consider decision processes that fulfill the Markov property.
  - I.e., where the environments response at time t depends only on the state and action representation at t.

- To define an MDP, we need to specify
  - State and action sets
  - One-step dynamics defined by state transition probabilities

$$p(s'|s,a) = \Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s,a)$$

  - Expected rewards for next state-action-next-state triplets

$$r(s,a,s') = \mathbb{E}[R_{t+1}|\, S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r\in\mathcal{R}} r\, p(s',r|s,a)}{p(s'|s,a)}$$

Recap: Policy

- Definition
  - A policy determines the agent's behavior
  - Map from state to action $\pi: \mathcal{S} \rightarrow \mathcal{A}$

- Two types of policies
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \Pr\{A_t = a | S_t = s\}$

- Note
  - $\pi(a|s)$ denotes the probability of taking action a when in state s.

Recap: Value Function

- Idea
  - Value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
  - And thus to select between actions

- Definition
  - The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,| S_t = s]$$

  - The value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, is the expected return starting from s, taking action a, and following π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,| S_t = s, A_t = a]$$

Recap: Optimal Value Functions

- Bellman optimality equations
  - For the optimal state-value function v_*:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

  - v_* is the unique solution to this system of nonlinear equations.
  - For the optimal action-value function q_*:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s',a')\right]$$

  - q_* is the unique solution to this system of nonlinear equations.

$\Rightarrow$ If the dynamics of the environment $p(s',r|s,a)$ are known, then in principle one can solve those equation systems.

Recap: Optimal Policies

- Why optimal state-value functions are useful
  - *Any policy that is greedy w.r.t. v_* is an optimal policy.*

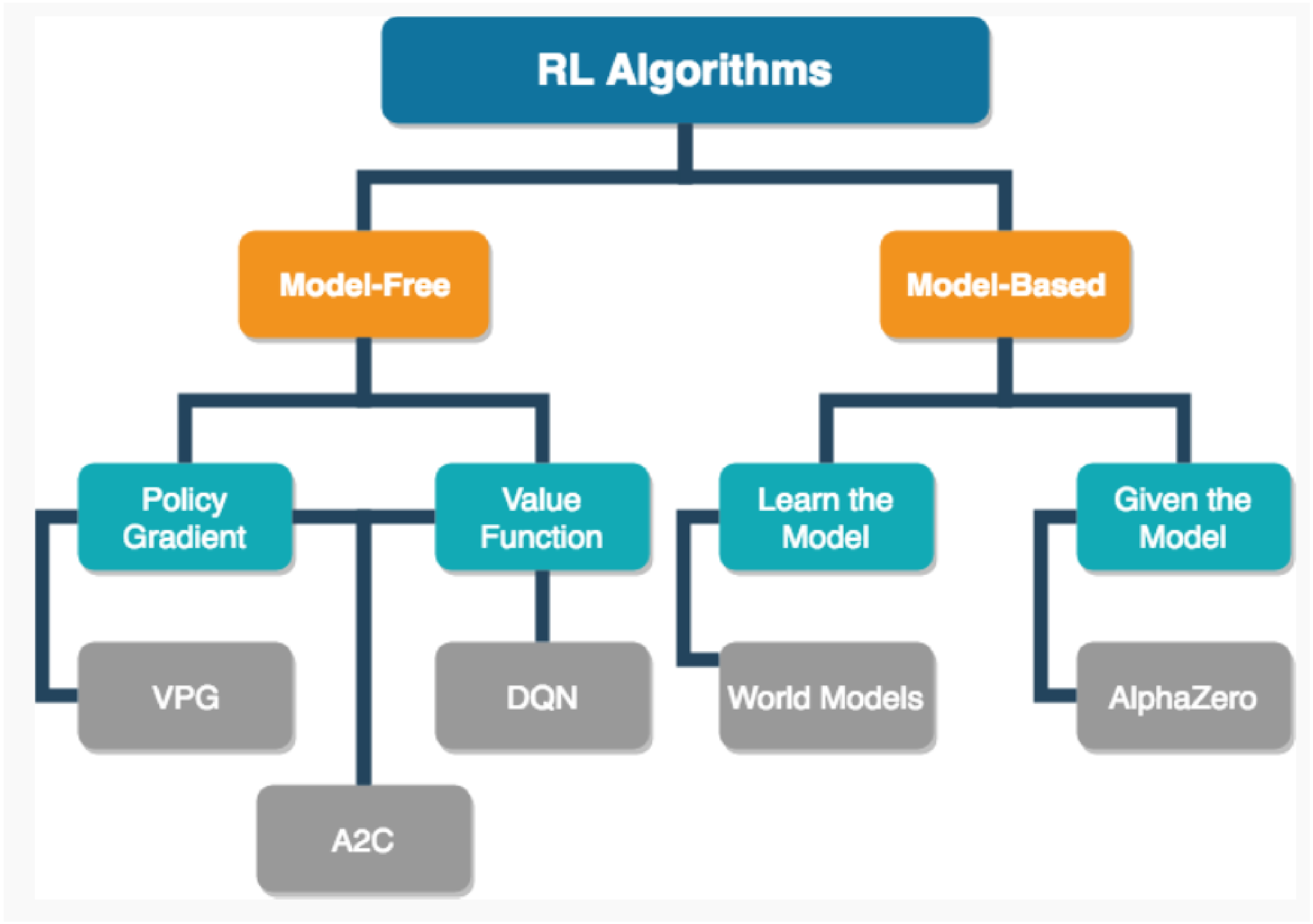
    $\Rightarrow$ Given v_*, one-step-ahead search produces the long-term optimal results.

    $\Rightarrow$ Given q_*, we do not even have to do one-step-ahead search

$$\pi_*(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}}\, q_*(s,a)$$

- Challenge
  - Many interesting problems have too many states for solving v_*.
  - Many Reinforcement Learning methods can be understood as approximately solving the Bellman optimality equations, using actually observed transitions instead of the ideal ones.

Recap: Taxonomy of RL methods



Slide Credit: Zac Kenton

Recap: Tabular vs. Approximate methods

- Tabular methods
  - For problems with small discrete state and action spaces
  - Value function or Policy function can be expressed as a table of values.
- Approximate methods
  - If we cannot enumerate our states or actions we use function approximation.
  - E.g., Kernel methods, Deep Learning / Neural Networks
- In practice, large problems with huge state spaces
  - E.g. chess: 10^{120} states.
  - Tabular methods don't scale well – they are a lookup table
    - Too many states to store in memory
    - Too slow to learn value function for every state/state-action.

Recap: Model-based vs Model-free

- Model-based
  - Has a model of the environment dynamics and reward
  - Allows agent to plan: predict state and reward before taking action
  - Pro: Better sample efficiency
  - Con: Agent only as good as the environment - Model-bias

- Model-free
  - No explicit model of the environment dynamics and reward
  - Less structured. More popular and further developed and tested.
  - Pro: Can be easier to implement and tune
  - Con: Very sample inefficient

Recap: Value-based RL vs Policy-based RL

- Policy-based RL
  - RL methods directly estimate a policy
  - A direct mapping of what action to take in each state.

$$\pi(a|s) = \mathrm{P}(\mathrm{a}|\mathrm{s}, \theta)$$

- Value-based RL
  - RL methods estimate a value function and derive a policy from that
  - Either a state-value function

$$\hat{V}(s;\theta) \approx V^{\pi}(\mathrm{s})$$

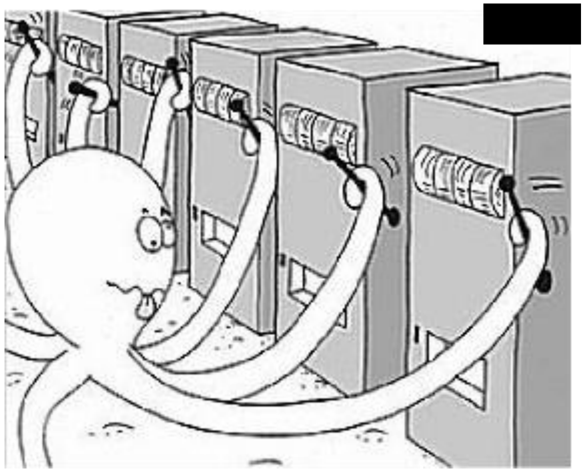
  - Or an action-state value function (Q function)

$$\hat{Q}(s,a;\theta) \approx Q^{\pi}(\mathrm{s},\mathrm{a})$$

- Or both simultaneously: Actor-Critic
  - Actor-Critic methods learn both a policy (actor) and a value function (critic)

Recap: Exploration-Exploitation Trade-off

- Example: N-armed bandit problem
  - Suppose we have the choice between N actions $a_1, \ldots, a_N$.
  - If we knew their value functions $q_*(s, a_i)$, it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.


[redacted]

- We can now
  - Exploit our current knowledge
    - And choose the greedy action that has the highest value based on our current estimate.
  - Explore to gain additional knowledge
    - And choose a non-greedy action to improve our estimate of that action's value.

Image source: research.microsoft.com

Recap: Simple Action Selection Strategies

- ϵ-greedy
  - Select the greedy action with probability $(1 - \epsilon)$ and a random one in the remaining cases.
  - $\Rightarrow$ In the limit, every action will be sampled infinitely often.
  - $\Rightarrow$ Probability of selecting the optimal action becomes $> (1 - \epsilon)$.
  - But: many bad actions are chosen along the way.

- Softmax
  - Choose action a_i at time t according to the softmax function

$$\frac{e^{q_t(a_i)/\tau}}{\sum_{j=1}^{N} e^{q_t(a_j)/\tau}}$$

    where τ is a temperature parameter (start high, then lower it).
  - Generalization: replace q_t by a preference function H_t that is learned by stochastic gradient ascent ("gradient bandit").

Recap: TD-Learning

- Policy evaluation (the prediction problem)
  - For a given policy π, compute the state-value function v_π.
- One option: Monte-Carlo methods
  - Play through a sequence of actions until a reward is reached, then backpropagate it to the states on the path.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target: the actual return after time t

- Temporal Difference Learning – TD(λ)
  - Directly perform an update using the estimate $V(S_{t+\lambda+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{} - V(S_t)]$$

Target: an estimate of the return (here: TD(0))

Recap: SARSA – On-Policy TD Control

- Idea
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate

- Procedure
  - Estimate $q_\pi(s, a)$ for the current policy π and for all states s and actions a.
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

  - This rule is applied after every transition from a nonterminal state S_t.
  - It uses every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

$\Rightarrow$ *Hence, the name SARSA*.

Recap: Q-Learning – Off-Policy TD Control

- Idea
  - Directly approximate the optimal action-value function q_*, independent of the policy being followed.

- Procedure
  - TD(0) update equation

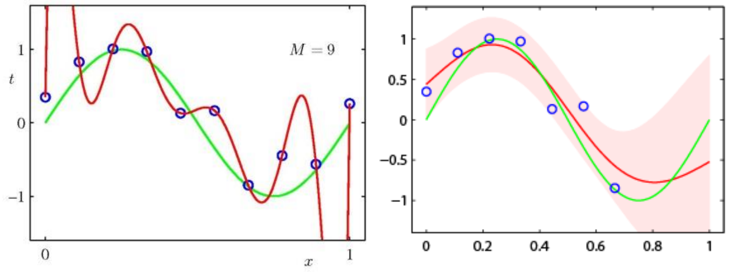
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

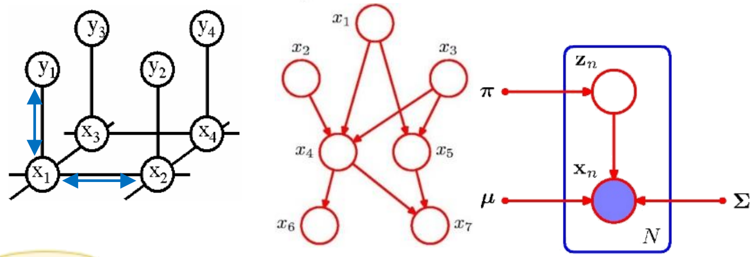
  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.

Course Outline

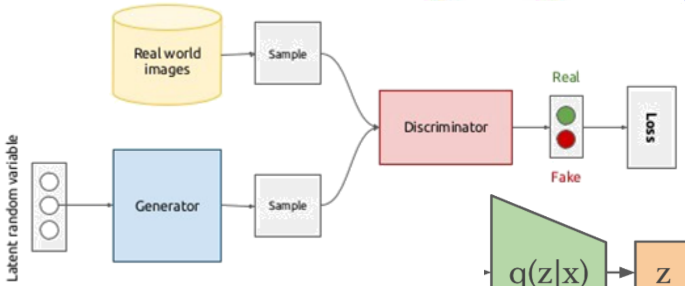
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

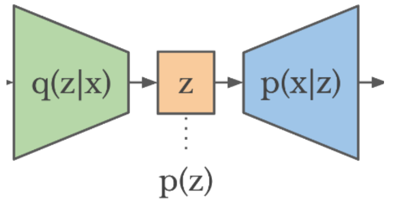
$$f : \mathcal{X} \rightarrow \mathbb{R}$$





y_1 y_2 y_3 y_4
x_1 x_2 x_3 x_4
x_1 x_2 x_3 x_4 x_5 x_6 x_7
π μ Σ
$\mathbf{z}_n$ $\mathbf{x}_n$ N





Recap: Deep Q-Learning

- Idea
  - Optimal Q-values should obey Bellman equation

$$Q_*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a') \,|s,a\right]$$

  - Treat the right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
  - Minimize MSE loss by stochastic gradient descent

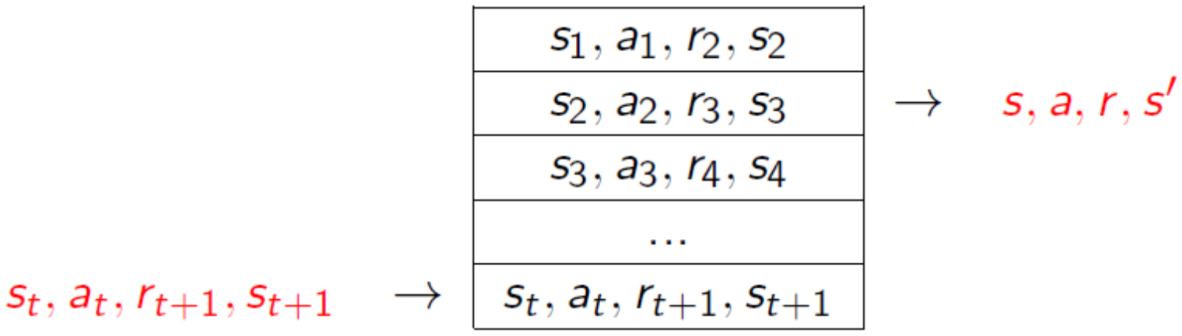
$$L(\mathbf{w}) = \left(r + \gamma \max_{a'} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

  - This converges to Q_* using a lookup table representation.
  - Unfortunately, it diverges using neural networks due to
    - Correlations between samples
    - Non-stationary targets

Slide adapted from David Silver

Recap: Deep Q-Networks (DQN)

- Adaptation: Experience Replay
  - To remove correlations, build a dataset from agent's own experience

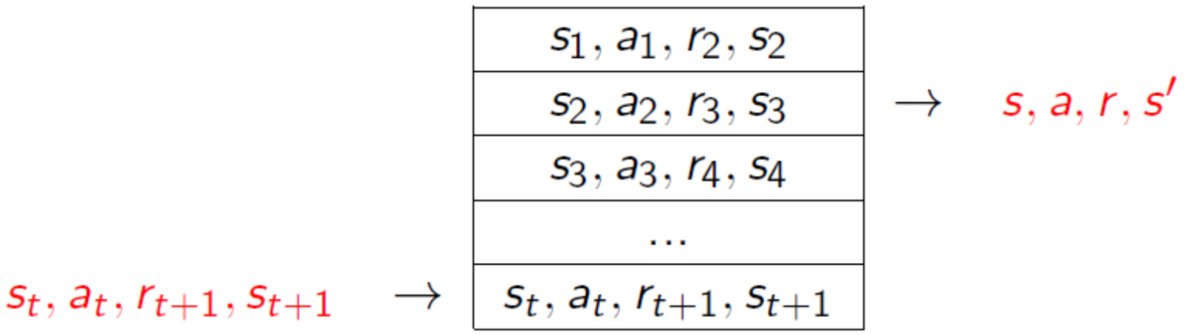


  - Perform minibatch updates to samples of experience drawn at random from the pool of stored samples
    - $(s, a, r, s') \sim U(D)$ where $D = \{(s_t, a_t, r_{t+1}, s_{t+1})\}$ is the dataset
  - Advantages
    - Each experience sample is used in many updates (more efficient)
    - Avoids correlation effects when learning from consecutive samples
    - Avoids feedback loops from on-policy learning

Slide adapted from David Silver

Recap: Deep Q-Networks (DQN)

- Adaptation: Experience Replay
  - To remove correlations, build a dataset from agent's own experience



  - Sample from the dataset and apply an update

$$L(\mathbf{w}) = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^{-}) - Q(s, a, \mathbf{w})\right)^2$$

  - To deal with non-stationary parameters, $\mathbf{w}^{-}$ are held fixed.
    - Only update the target network parameters every C steps.
    - I.e., clone the network Q to generate a target network $\hat{Q}$.

  $\Rightarrow$ Again, this reduces oscillations to make learning more stable.

Slide adapted from David Silver

Recap: Policy Gradients

see Exercise 2.1

- How to make high-value actions more likely
  - The gradient of a stochastic policy $\pi(s, \mathbf{u})$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \mathbb{E}_\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + \ \dots \,|\, \pi(\cdot, \mathbf{u})]$$

$$= \mathbb{E}_\pi \left[\frac{\partial \log \pi(a|s, \boldsymbol{u})}{\partial \boldsymbol{u}} Q_\pi(s, a) \right]$$

  - The gradient of a deterministic policy $a = \pi(s)$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}_\pi \left[\frac{\partial Q_\pi(s, a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}} \right]$$

if a is continuous and Q is differentiable.

Slide adapted from David Silver

Recap: Monte-Carlo Policy Gradient

see Exercise 2.1

- Execute policy to obtain sample episodes
- Update parameters by stochastic gradient ascent using the policy gradient theorem

- REINFORCE algorithm
  - Initialize parameters arbitrarily
  - Repeat
    - Sample episode $(s_o, a_0, r_1, s_1, a_1, \dots, r_T, s_T)$ using current policy
    - For each $t \in \{0, \dots, T-1\}$
      - Update policy

$$\theta \leftarrow \theta + \eta \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(\sum_{k=0}^{T-t-1} r_{t+k+1} \right)$$

Slide credit: David Silver

Recap: Deep Policy Gradients (DPG)

- DPG is the continuous analogue of DQN
  - Experience replay: build data-set from agent's experience
  - Critic estimates value of current policy by DQN

$$L_{\mathbf{w}}(\mathbf{w}) = \left(r + \gamma Q(s', \pi(s', \mathbf{u}^{-}), \mathbf{w}^{-}) - Q(s, a, \mathbf{w})\right)^2$$

  - To deal with non-stationarity, targets $\mathbf{u}^{-}, \mathbf{w}^{-}$ are held fixed
  - Actor updates policy in direction that improves Q

$$\frac{\partial L_{\mathbf{u}}(\mathbf{u})}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

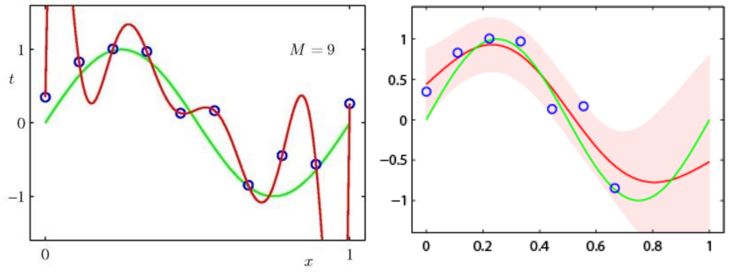
  - In other words critic provides loss function for actor.

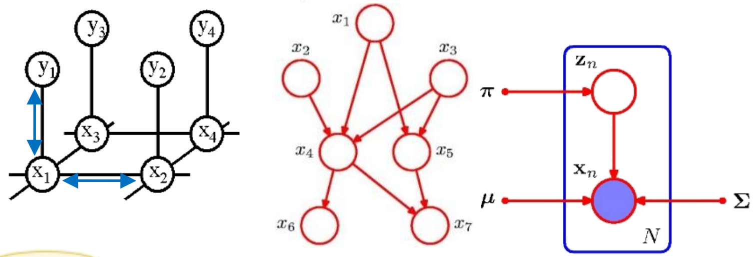
Slide credit: David Silver

Course Outline

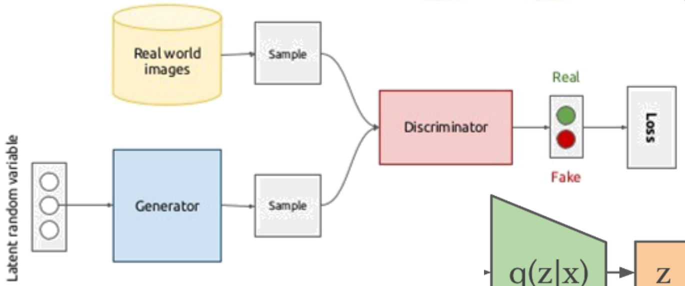
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

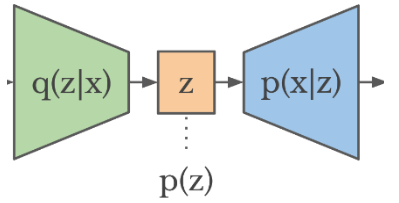
$$f : \mathcal{X} \rightarrow \mathbb{R}$$





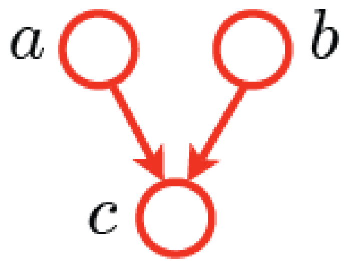
y_1 y_2 y_3 y_4 x_1 x_2 x_3 x_4
x_1 x_2 x_3 x_4 x_5 x_6 x_7
$\mathbf{z}_n$ $\mathbf{x}_n$ π μ Σ N



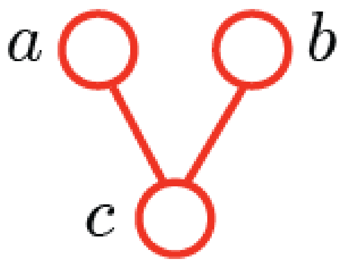


Recap: Graphical Models

- Two basic kinds of graphical models
  - Directed graphical models or Bayesian Networks
  - Undirected graphical models or Markov Random Fields
- Key components
  - Nodes
    - Random variables
  - Edges
    - Directed or undirected



Directed graphical model



Undirected graphical model

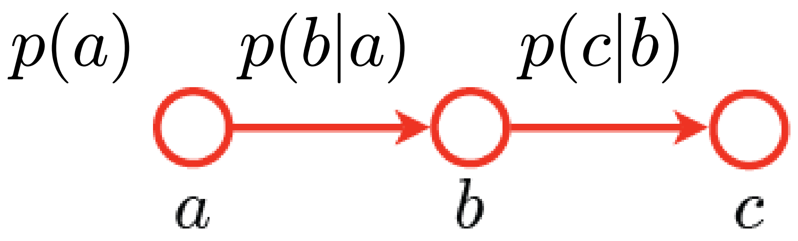
  - The value of a random variable may be known or unknown.

○ unknown ● known

Slide credit: Bernt Schiele

Recap: Directed Graphical Models

- Chains of nodes:



  – Knowledge about a is expressed by the prior probability:

$$p(a)$$

  – Dependencies are expressed through conditional probabilities:

$$p(b|a), \;\; p(c|b)$$

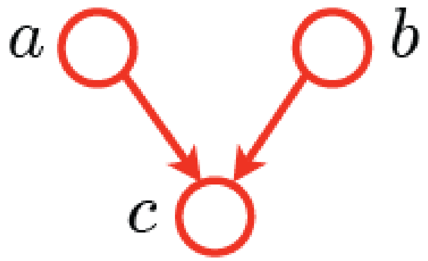
  – Joint distribution of all three variables:

$$\begin{aligned} p(a,b,c) &= p(c|\cancel{a},b)p(a,b) \\ &= p(c|b)p(b|a)p(a) \end{aligned}$$

Slide credit: Bernt Schiele, Stefan Roth

Recap: Directed Graphical Models

- Convergent connections:



  - Here the value of c depends on both variables a and b.
  - This is modeled with the conditional probability:

$$p(c|a,b)$$

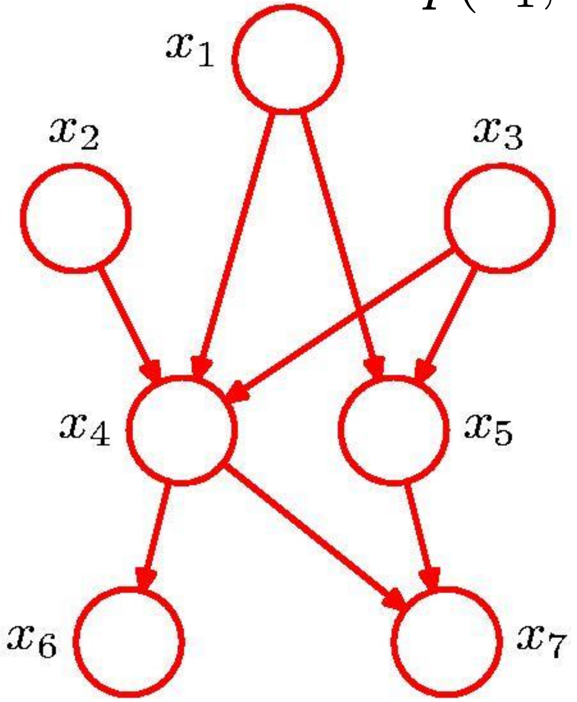
  - Therefore, the joint probability of all three variables is given as:

$$\begin{aligned} p(a,b,c) &= p(c|a,b)p(a,b) \\ &= p(c|a,b)p(a)p(b) \end{aligned}$$

Slide credit: Bernt Schiele, Stefan Roth

Recap: Factorization of the Joint Probability

- Exercise: Computing the joint probability



$$\begin{aligned} p(x_1, \ldots, x_7) &= p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3) \\ &\quad p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5) \end{aligned}$$

General factorization

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$

We can directly read off the factorization of the joint from the network structure!

Image source: C. Bishop, 2006

Recap: Factorized Representation

- Reduction of complexity
  - Joint probability of n binary variables requires us to represent values by brute force

$$\mathcal{O}(2^n) \text{ terms}$$

  - The factorized form obtained from the graphical model only requires

$$\mathcal{O}(n \cdot 2^k) \text{ terms}$$

    - k: maximum number of parents of a node.

$\Rightarrow$ *It's the edges that are missing in the graph that are important! They encode the simplifying assumptions we make.*

Slide credit: Bernt Schiele, Stefan Roth

Recap: Conditional Independence

see Exercise 3.2

- X is conditionally independent of Y given V
  - Definition: $$X \perp\!\!\!\perp Y | V \quad \Leftrightarrow \quad p(X|Y,V) = p(X|V)$$
  - Also: $$X \perp\!\!\!\perp Y | V \quad \Leftrightarrow \quad p(X,Y|V) = p(X|V)\, p(Y|V)$$
  - Special case: Marginal Independence

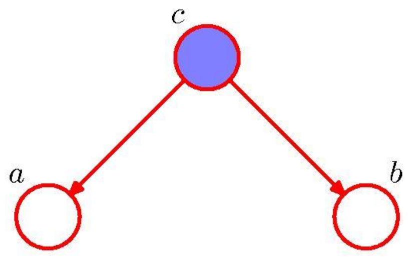
$$X \perp\!\!\!\perp Y \quad \Leftrightarrow \quad X \perp\!\!\!\perp Y | \emptyset \quad \Leftrightarrow \quad p(X,Y) = p(X)\, p(Y)$$

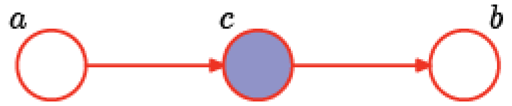
  - Often, we are interested in conditional independence between sets of variables:

$$\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{V} \quad \Leftrightarrow \quad \{X \perp\!\!\!\perp Y | \mathcal{V}, \;\; \forall X \in \mathcal{X} \;\text{ and }\; \forall Y \in \mathcal{Y}\}$$

Recap: Conditional Independence

- Three cases
  - Divergent ("Tail-to-Tail")
    - Conditional independence when c is observed.
  - Chain ("Head-to-Tail")
    - Conditional independence when c is observed.
  - Convergent ("Head-to-Head")
    - Conditional independence when neither c, nor any of its descendants are observed.





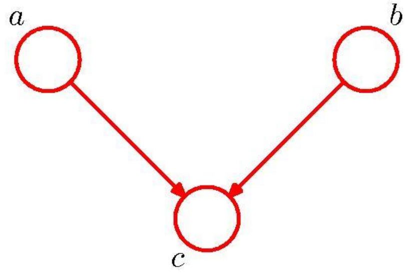
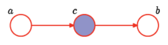


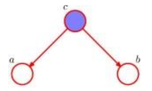
Image source: C. Bishop, 2006

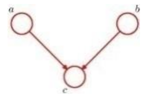
Recap: D-Separation

see Exercise 3.2

- Definition
  - Let A, B, and C be non-intersecting subsets of nodes in a directed graph.
  - A path from A to B is blocked if it contains a node such that either
    - The arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set C, or
    - The arrows meet head-to-head at the node, and neither the node, nor any of its descendants, are in the set C.
  - If all paths from A to B are blocked, A is said to be d-separated from B by C.







- If A is d-separated from B by C, the joint distribution over all variables in the graph satisfies $A \perp\!\!\!\perp B \mid C$.
  - Read: "A is conditionally independent of B given C."

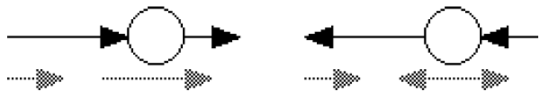
Slide adapted from Chris Bishop

Recap: “Bayes Ball” Algorithm

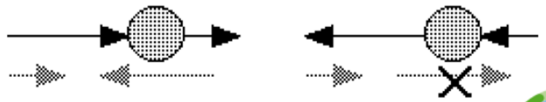
see Exercise 3.2

- Graph algorithm to compute d-separation
  - *Goal: Get a ball from X to Y without being blocked by $\mathcal{V}$.*
  - Depending on its direction and the previous node, the ball can
    - Pass through (from parent to all children, from child to all parents)
    - Bounce back (from any parent/child to all parents/children)
    - Be blocked

- Game rules
  - An unobserved node ($W \notin \mathcal{V}$) passes through balls from parents, but *also* bounces back balls from children.

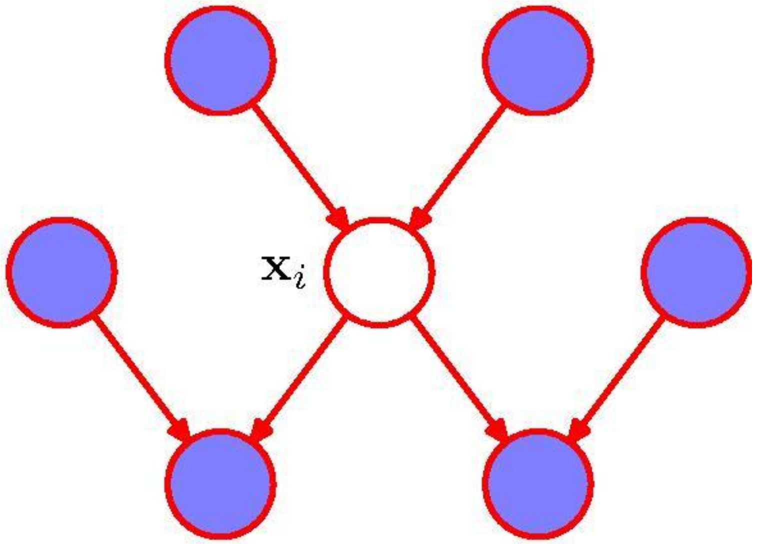


  - An observed node ($W \in \mathcal{V}$) bounces back balls from parents, but blocks balls from children.



Slide adapted from Zoubin Gharahmani

Recap: The Markov Blanket

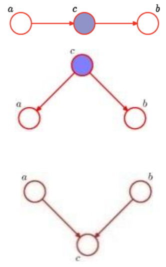


- Markov blanket of a node $\mathbf{x}_i$
  - Minimal set of nodes that isolates $\mathbf{x}_i$ from the rest of the graph.
  - This comprises the set of
    - Parents,
    - Children, and
    - Co-parents of $\mathbf{x}_i$. ⟵ This is what we have to watch out for!

Image source: C. Bishop, 2006

Recap: D-Separation

- Definition
  - Let A, B, and C be non-intersecting subsets of nodes in a directed graph.
  - A path from A to B is blocked if it contains a node such that either
    - The arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set C, or
    - The arrows meet head-to-head at the node, and neither the node, nor any of its descendants, are in the set C.
  - If all paths from A to B are blocked, A is said to be d-separated from B by C.



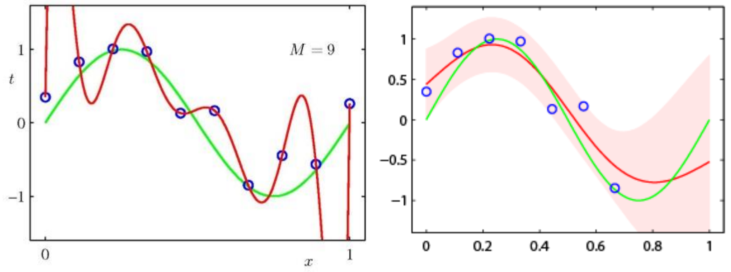
- If A is d-separated from B by C, the joint distribution over all variables in the graph satisfies $A \perp\!\!\!\perp B \mid C$.
  - Read: "A is conditionally independent of B given C."

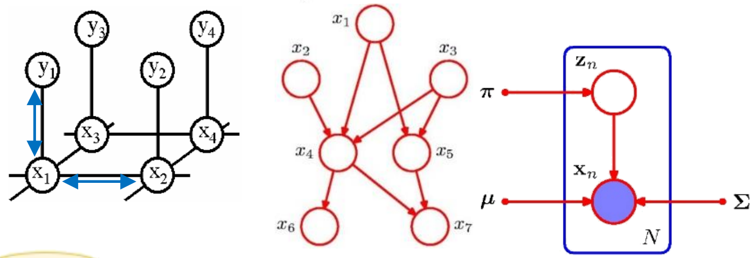
Slide adapted from Chris Bishop

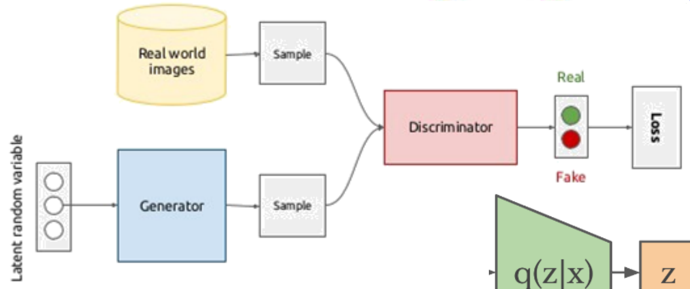
Course Outline

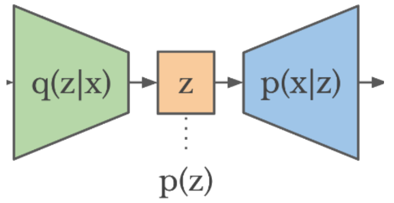
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \to \mathbb{R}$$



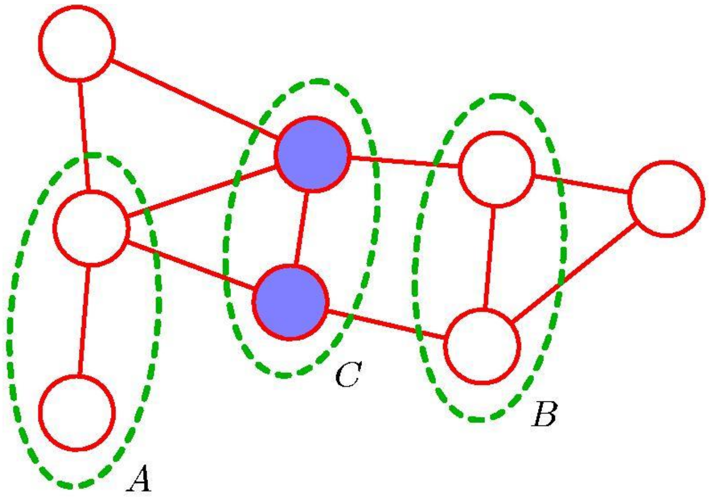






Recap: Undirected Graphical Models

- Undirected graphical models (“Markov Random Fields”)
  - Given by undirected graph



- Conditional independence for undirected graphs
  - If every path from any node in set A to set B passes through at least one node in set C, then $A \perp\!\!\!\perp B | C$.
  - Simple Markov blanket:

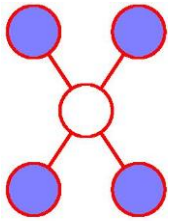


Image source: C. Bishop, 2006

Recap: Factorization in MRFs

- Joint distribution
  - Written as product of potential functions over maximal cliques in the graph:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C)$$

  - The normalization constant Z is called the partition function.

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C)$$

- Remarks
  - BNs are automatically normalized. But for MRFs, we have to explicitly perform the normalization.
  - Presence of normalization constant is major limitation!
    - Evaluation of Z involves summing over $\mathcal{O}(K^M)$ terms for M nodes!

Recap: Factorization in MRFs

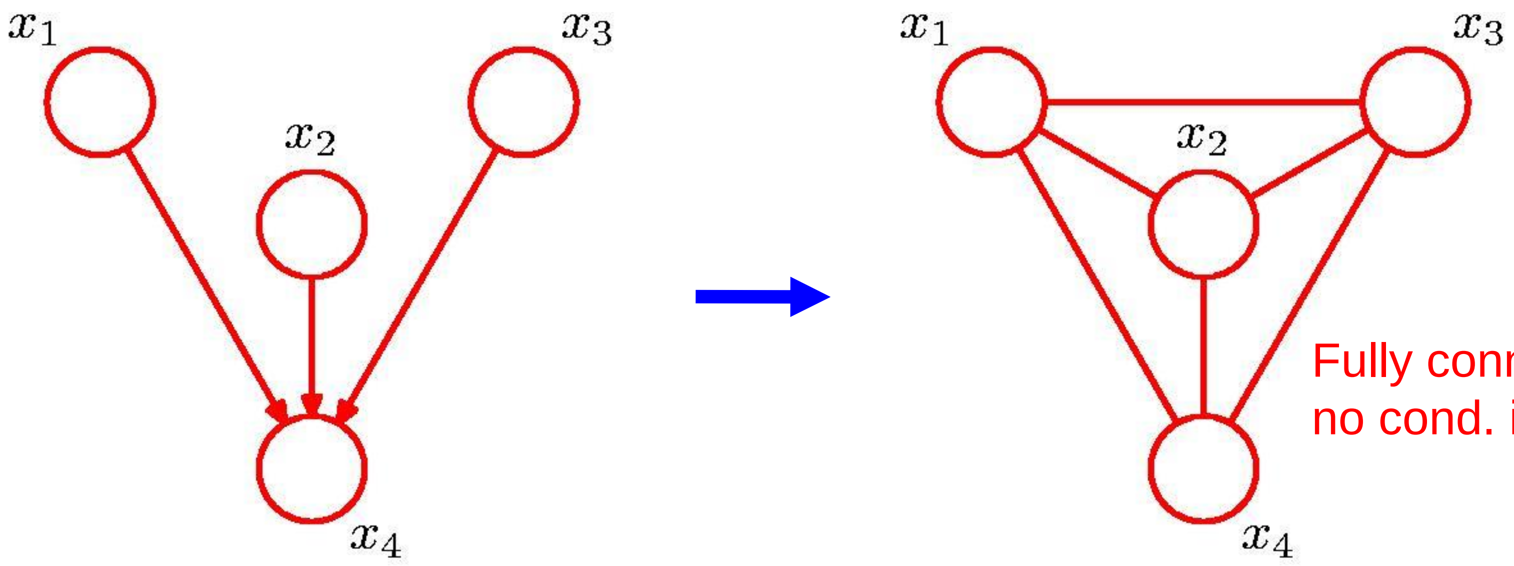
- Role of the potential functions
  - General interpretation
    - No restriction to potential functions that have a specific probabilistic interpretation as marginals or conditional distributions.
  - Convenient to express them as exponential functions ("Boltzmann distribution")

$$\psi_C(\mathbf{x}_C) = \exp\{-E(\mathbf{x}_C)\}$$

    - with an energy function E.
  - Why is this convenient?
    - Joint distribution is the product of potentials $\Rightarrow$ sum of energies.
    - We can take the log and simply work with the sums…

Recap: Converting Directed to Undirected Graphs

- Problematic case: multiple parents



$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3)\underbrace{p(x_4|x_1,x_2,x_3)}$$

Need a clique of $x_1,\ldots,x_4$ to represent this factor!

– Need to introduce additional links ("marry the parents").

$\Rightarrow$ This process is called moralization. It results in the moral graph.

Slide adapted from Chris Bishop

Image source: C. Bishop, 2006

Recap: Conversion Algorithm

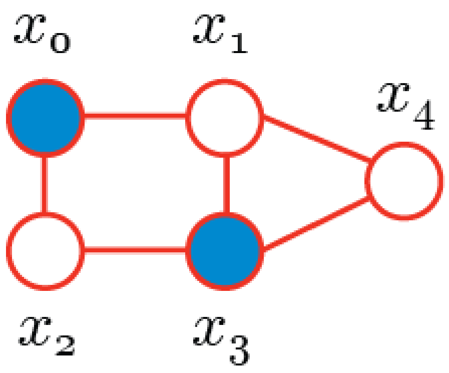
- General procedure to convert directed $\rightarrow$ undirected
  1. Add undirected links to marry the parents of each node.
  2. Drop the arrows on the original links $\Rightarrow$.
  3. Find maximal cliques for each node and initialize all clique potentials to 1.
  4. Take each conditional distribution factor of the original directed graph and multiply it into one clique potential.

- Restriction
  – Conditional independence properties are often lost!
  – Moralization results in additional connections and larger cliques.

Slide adapted from Chris Bishop

Recap: Computing Marginals

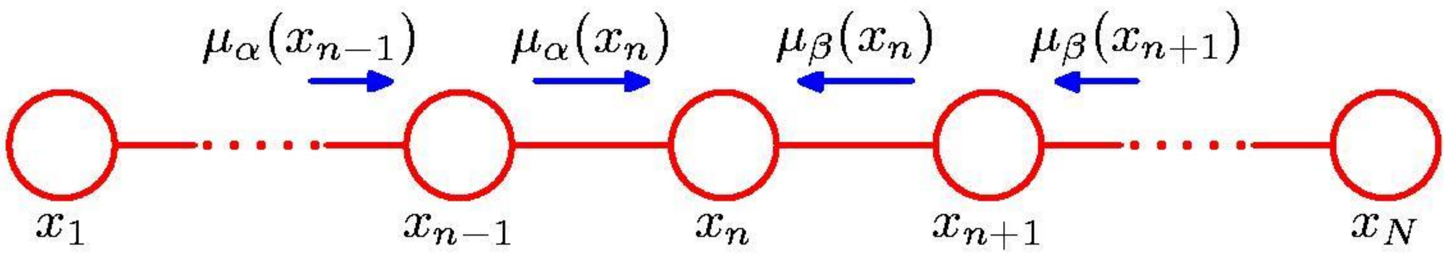
- How do we apply graphical models?
  - Given some observed variables, we want to compute distributions of the unobserved variables.
  - In particular, we want to compute marginal distributions, for example $p(x_4)$.



- How can we compute marginals?
  - Classical technique: sum-product algorithm by Judea Pearl.
  - In the context of (loopy) undirected models, this is also called (loopy) belief propagation [Weiss, 1997].
  - Basic idea: message-passing.

Slide credit: Bernt Schiele, Stefan Roth

Recap: Message Passing on a Chain



– Idea: Pass messages from the two ends towards the query node x_n.

– Define the messages recursively:

$$\mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n)\mu_\alpha(x_{n-1})$$

$$\mu_\beta(x_n) = \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1})\mu_\beta(x_{n+1})$$

– Compute the normalization constant Z at any node x_m.

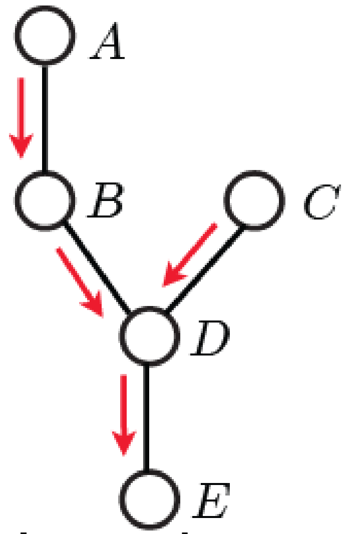
$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

Slide adapted from Chris Bishop

Image source: C. Bishop, 2006

Summary: Message Passing on Trees

- General procedure for all tree graphs.
  - Root the tree at the variable that we want to compute the marginal of.
  - Start computing messages at the leaves.
  - Compute the messages for all nodes for which all incoming messages have already been computed.
  - Repeat until we reach the root.



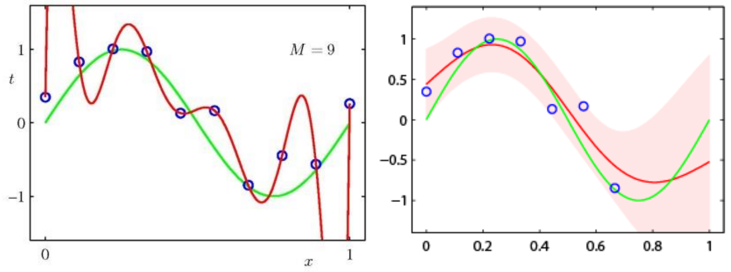
- If we want to compute the marginals for all possible nodes (roots), we can reuse some of the messages.
  - Computational expense linear in the number of nodes.
- We already motivated message passing for inference.
  - How can we formalize this into a general algorithm?

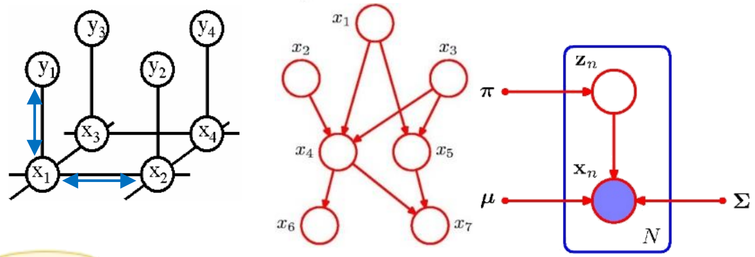
Slide credit: Bernt Schiele, Stefan Roth

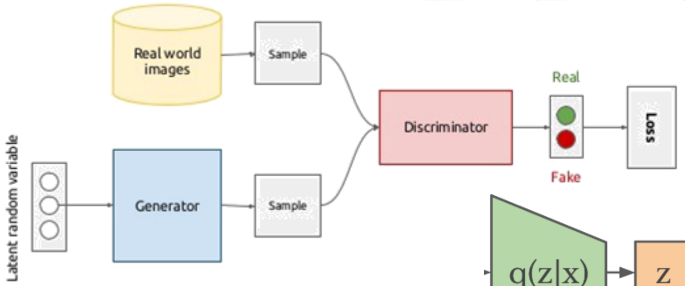
Course Outline

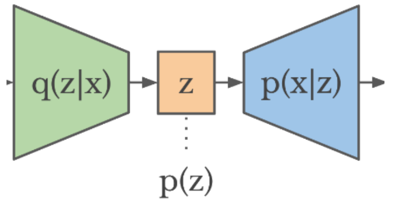
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \to \mathbb{R}$$

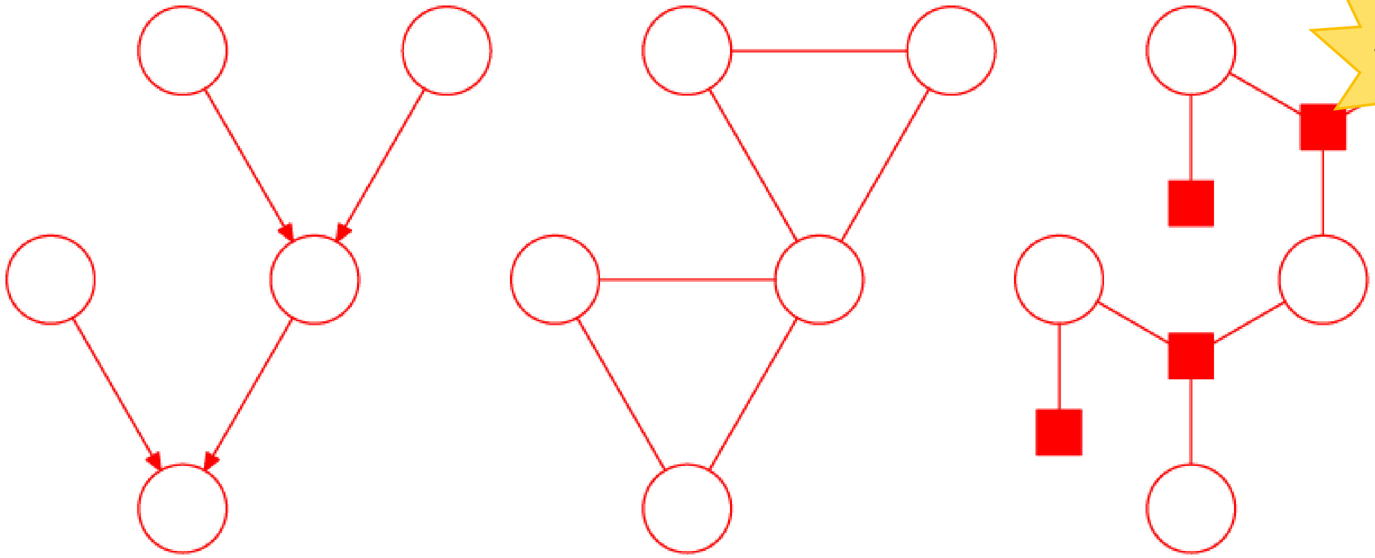








Recap: Factor Graphs



see Exercise 3.4

- Joint probability
  - Can be expressed as product of factors: $p(\mathbf{x}) = \frac{1}{Z}\prod_s f_s(\mathbf{x}_s)$
  - Factor graphs make this explicit through separate factor nodes.
- Converting a directed polytree
  - Conversion to undirected tree creates loops due to moralization!
  - Conversion to a factor graph again results in a tree!

Image source: C. Bishop, 2006

Recap: Sum-Product Algorithm

- Objectives
  - Efficient, exact inference algorithm for finding marginals.
- Procedure:
  - Pick an arbitrary node as root.
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.

$$p(x) \propto \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x)$$

- Computational effort
  - Total number of messages = 2 · number of graph edges.

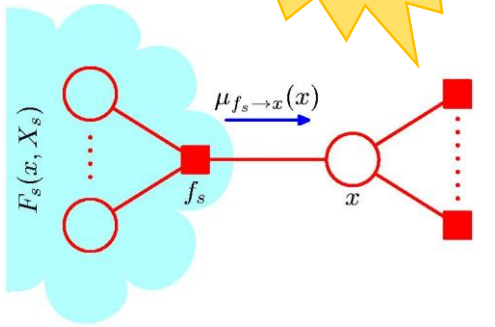
Slide adapted from Chris Bishop

Recap: Sum-Product Algorithm

see Exercise 3.4

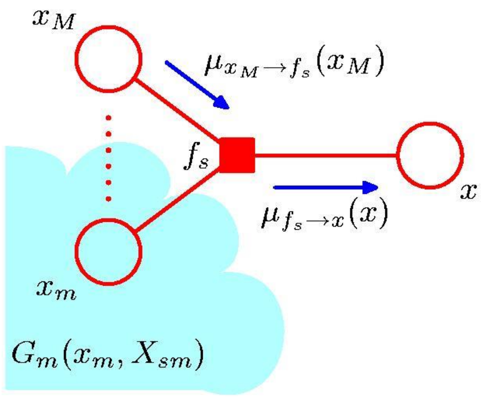
- Two kinds of messages
  - Message from factor node to variable nodes:
    - Sum of factor contributions

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$
$$= \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

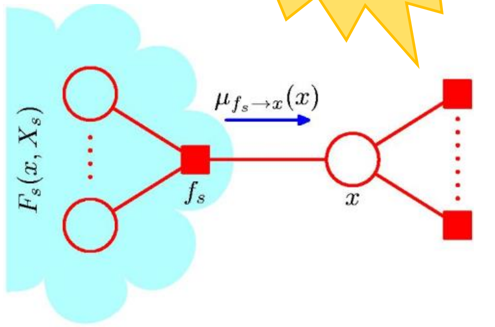


  - Message from variable node to factor node:
    - Product of incoming messages

$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

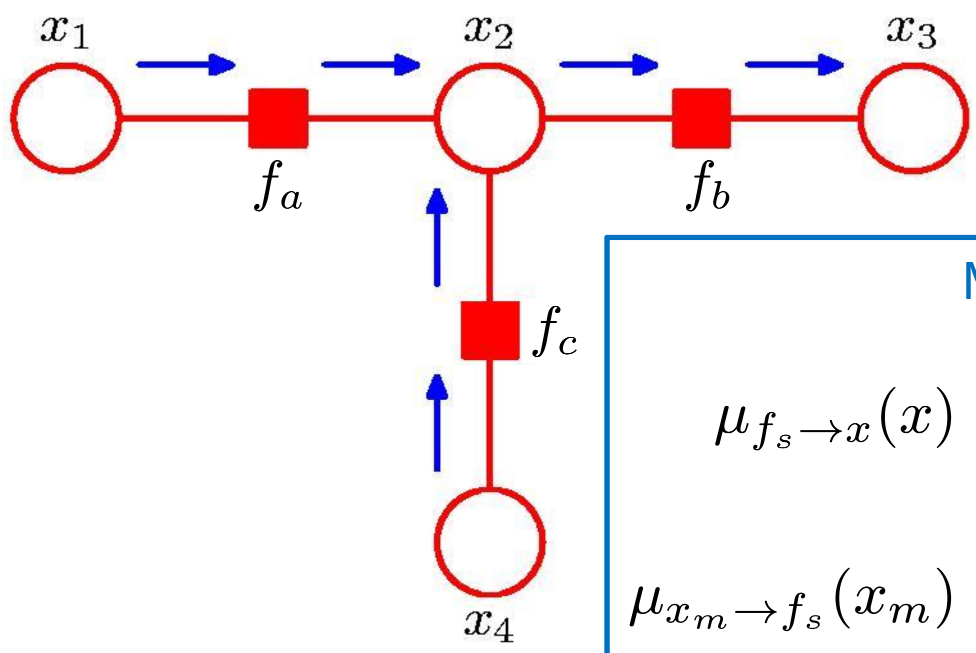


$\Rightarrow$ Simple propagation scheme.



Recap: Sum-Product from Leaves to Root

see Exercise 3.4

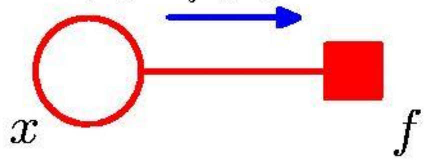


Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

$$\mu_{x \to f}(x) = 1$$



$$\mu_{f \to x}(x) = f(x)$$

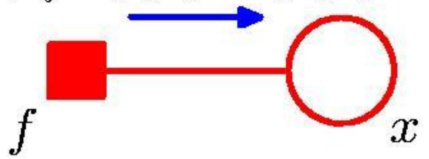
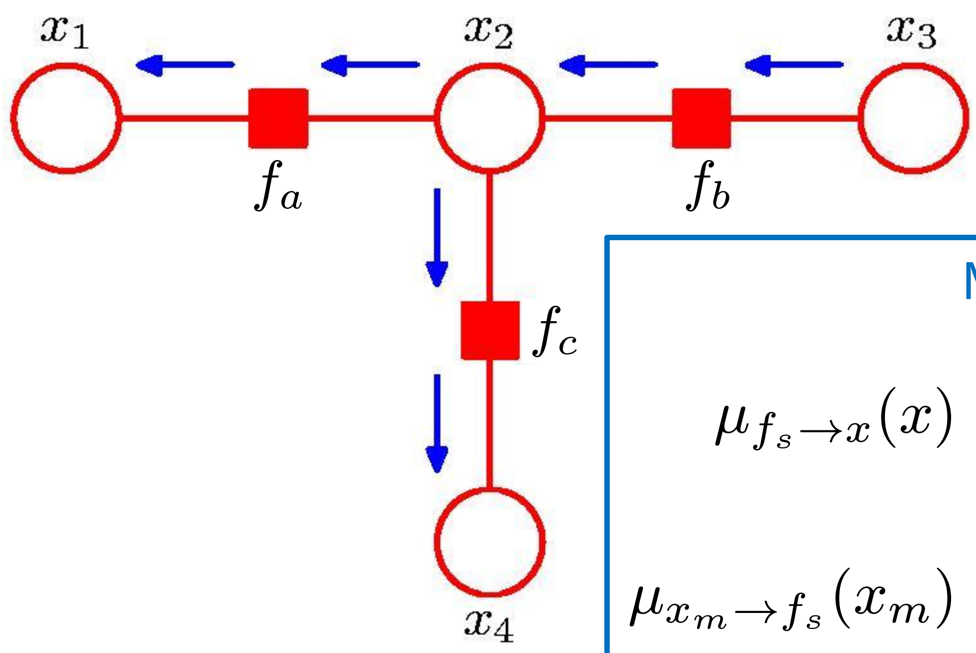


Image source: C. Bishop, 2006

Recap: Sum-Product from Root to Leaves

see Exercise 3.4

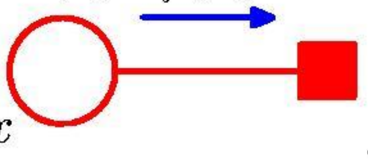


Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

$$\mu_{x \to f}(x) = 1$$



$$\mu_{f \to x}(x) = f(x)$$

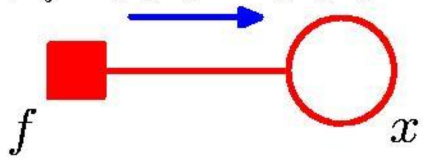


Image source: C. Bishop, 2006

Recap: Max-Sum Algorithm

- Objective: an efficient algorithm for finding
  - Value $\mathbf{x}^{\max}$ that maximises $p(\mathbf{x})$;
  - Value of $p(\mathbf{x}^{\max})$.

  $\Rightarrow$ Application of dynamic programming in graphical models.

- Key ideas
  - We are interested in the maximum value of the joint distribution

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x})$$

  $\Rightarrow$ Maximize the product $p(\mathbf{x})$.

  - For numerical reasons, use the logarithm.

$$\ln\left(\max_{\mathbf{x}} p(\mathbf{x})\right) = \max_{\mathbf{x}} \ln p(\mathbf{x}).$$

  $\Rightarrow$ Maximize the sum (of log-probabilities).

Slide adapted from Chris Bishop

Recap: Max-Sum Algorithm

- Initialization (leaf nodes)

$$\mu_{x \to f}(x) = 0 \qquad \mu_{f \to x}(x) = \ln f(x)$$

- Recursion
  - Messages

$$\mu_{f \to x}(x) = \max_{x_1,\ldots,x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

$$\mu_{x \to f}(x) = \sum_{l \in \mathrm{ne}(x) \setminus f} \mu_{f_l \to x}(x)$$

  - For each node, keep a record of which values of the variables gave rise to the maximum state:

$$\phi(x) = \arg\max_{x_1,\ldots,x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

Slide adapted from Chris Bishop

Recap: Max-Sum Algorithm

- Termination (root node)
  - Score of maximal configuration

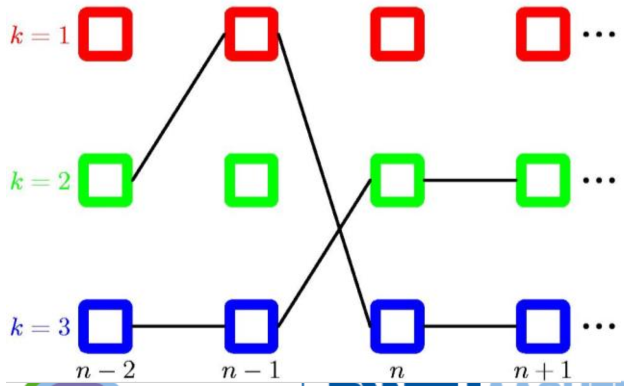
$$p^{\max} = \max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]$$

  - Value of root node variable giving rise to that maximum

$$x^{\max} = \arg\max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]$$

  - Back-track to get the remaining variable values

$$x_{n-1}^{\max} = \phi(x_n^{\max})$$



Slide adapted from Chris Bishop

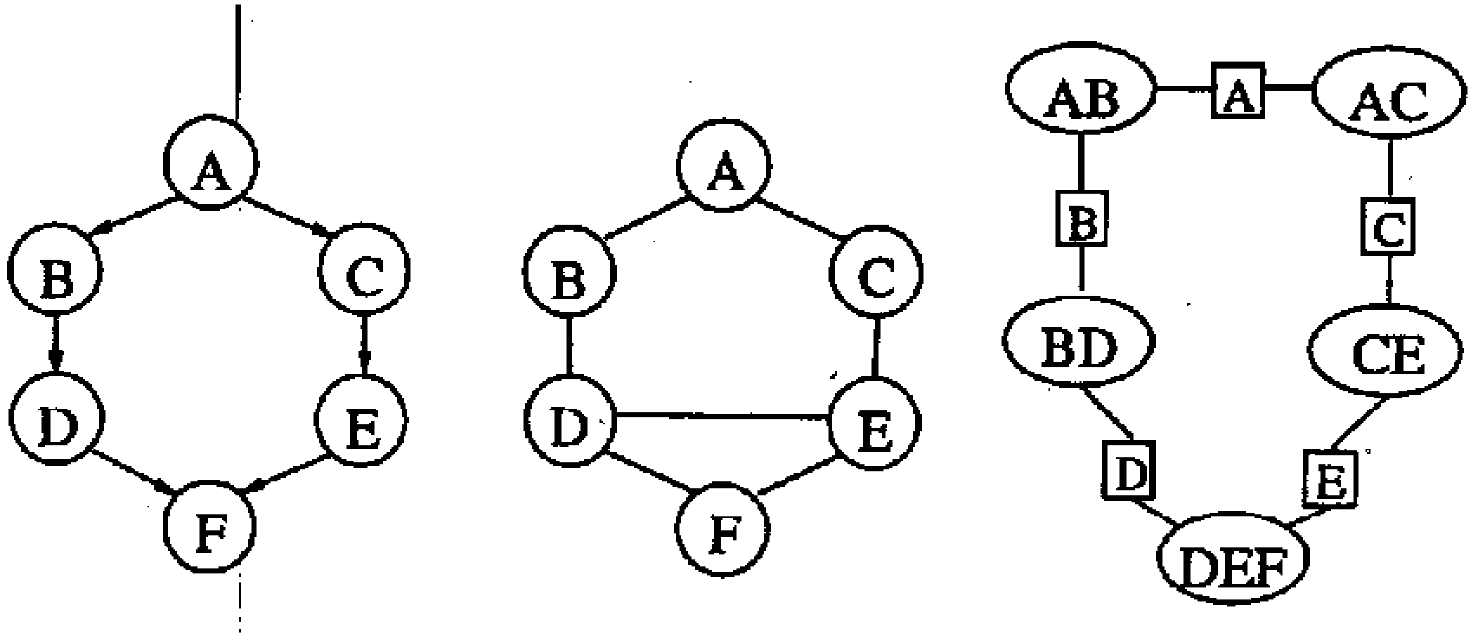
Recap: Junction Tree Algorithm

- Motivation
  - Exact inference on general graphs.
  - Works by turning the initial graph into a junction tree and then running a sum-product-like algorithm.
  - Intractable on graphs with large cliques.

- Main steps
  1. If starting from directed graph, first convert it to an undirected graph by moralization.
  2. Introduce additional links by triangulation in order to reduce the size of cycles.
  3. Find cliques of the moralized, triangulated graph.
  4. Construct a new graph from the maximal cliques.
  5. Remove minimal links to break cycles and get a .

  $\Rightarrow$ Apply regular message passing to perform inference.

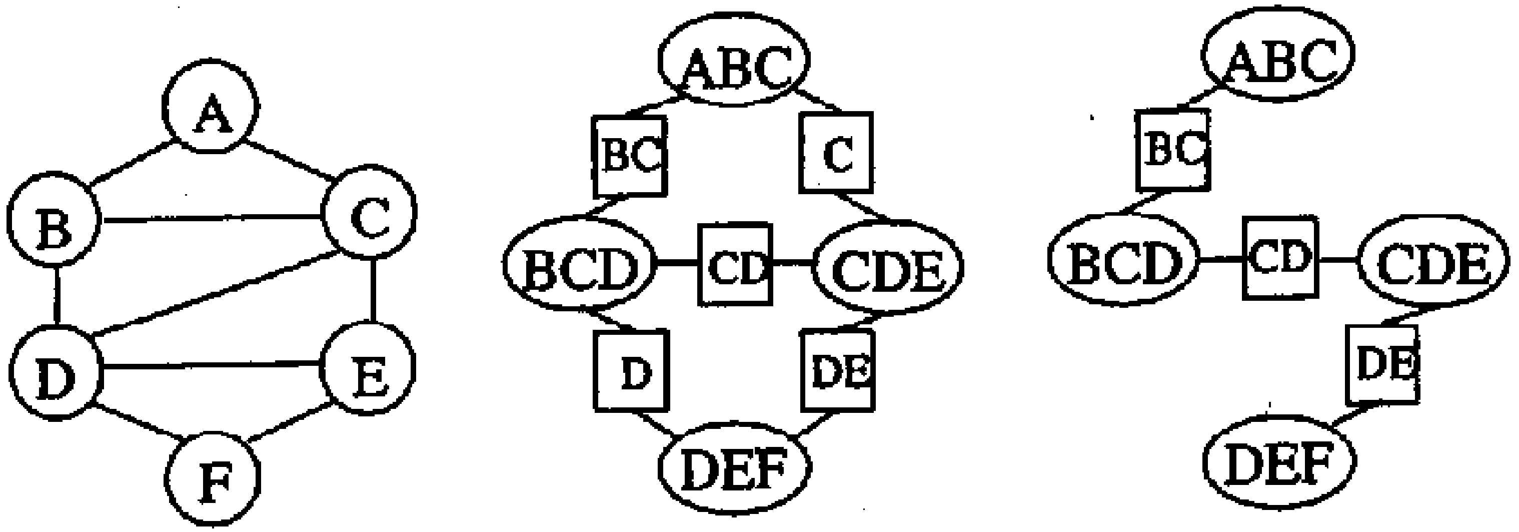
Recap: Junction Tree Example



- Without triangulation step
  - The final graph will contain cycles that we cannot break without losing the running intersection property!

Image source: J. Pearl, 1988

Recap: Junction Tree Example

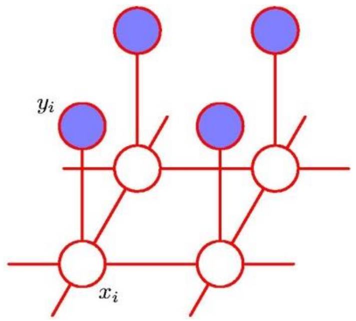


- When applying the triangulation
  - Only small cycles remain that are easy to break.
  - Running intersection property is maintained.

Image source: J. Pearl, 1988

Recap: MRF Structure for Images

- Basic structure

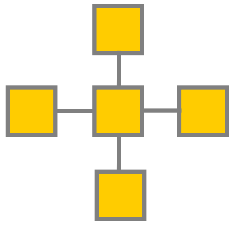


Noisy observations

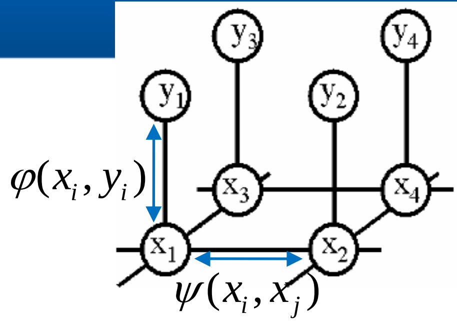
"True" image content

- Two components
  - Observation model
    - How likely is it that node x_i has label L_i given observation y_i?
    - This relationship is usually learned from training data.
  - Neighborhood relations
    - Simplest case: 4-neighborhood
    - Serve as smoothing terms.

    $\Rightarrow$ Discourage neighboring pixels to have different labels.
    - This can either be learned or be set to fixed "penalties".



Recap: Energy Formulation

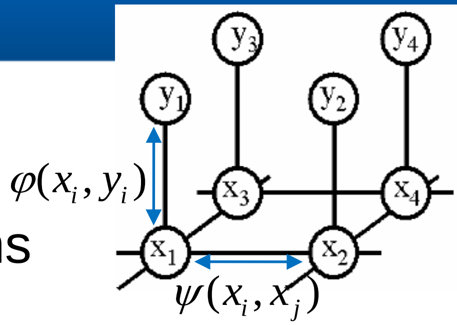


- Energy function

$$E(x,y) = \sum_i \underbrace{\varphi(x_i, y_i)}_{\text{Single-node potentials}} + \sum_{i,j} \underbrace{\psi(x_i, x_j)}_{\text{Pairwise potentials}}$$

- Single-node (unary) potentials φ
  - Encode local information about the given pixel/patch.
  - How likely is a pixel/patch to belong to a certain class (e.g. foreground/background)?
- Pairwise potentials ψ
  - Encode neighborhood information.
  - How different is a pixel/patch's label from that of its neighbor? (e.g. based on intensity/color/texture difference, edges)

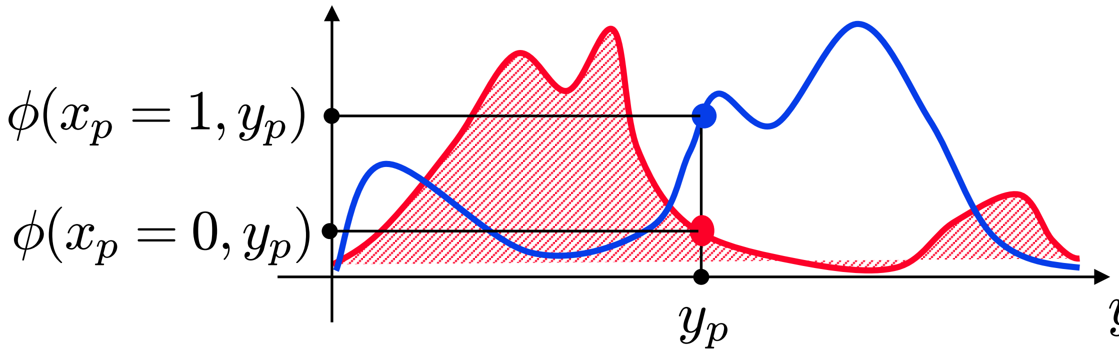
Recap: How to Set the Potentials?



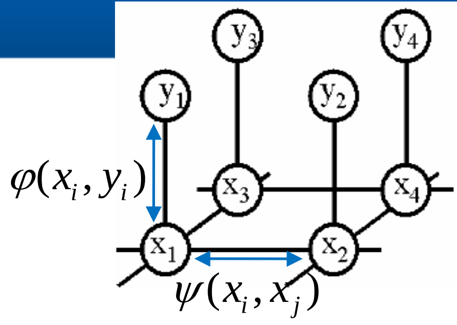
- Unary potentials
  - E.g. color model, modeled with a Mixture of Gaussians

$$\phi(x_i, y_i; \theta_\phi) = \log \sum_k \theta_\phi(x_i, k) p(k|x_i) \mathcal{N}(y_i; \bar{y}_k, \Sigma_k)$$

$\Rightarrow$ Learn color distributions for each label



Recap: How to Set the Potentials?



- Pairwise potentials
  - Potts Model

$$\psi(x_i, x_j; \theta_\psi) = \theta_\psi \delta(x_i \neq x_j)$$

    - Simplest discontinuity preserving model.
    - Discontinuities between any pair of labels are penalized equally.
    - Useful when labels are unordered or number of labels is small.

  - Extension: "contrast sensitive Potts model"

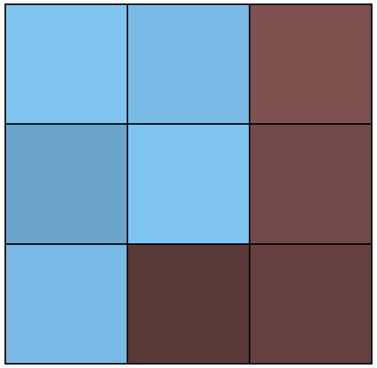
$$\psi(x_i, x_j, g_{ij}(y); \theta_\psi) = \theta_\psi g_{ij}(y) \delta(x_i \neq x_j)$$

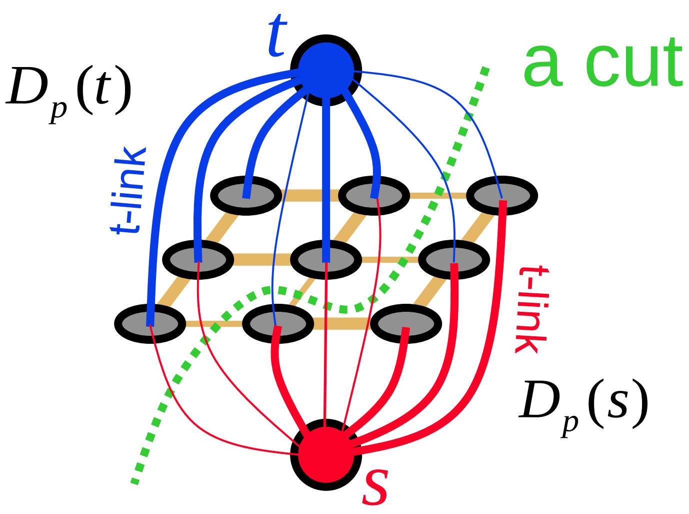
where

$$g_{ij}(y) = e^{-\beta \|y_i - y_j\|^2} \qquad \beta = 2 \cdot avg\left(\|y_i - y_j\|^2\right)$$

    - Discourages label changes except in places where there is also a large change in the observations.

Recap: Graph Cuts for Binary Problems





"expected" intensities of
object and background
I^s and I^t
can be re-estimated

$$D_p(s) \propto \exp\left(-\| I_p - I^s \|^2 / 2\sigma^2\right)$$
$$D_p(t) \propto \exp\left(-\| I_p - I^t \|^2 / 2\sigma^2\right)$$

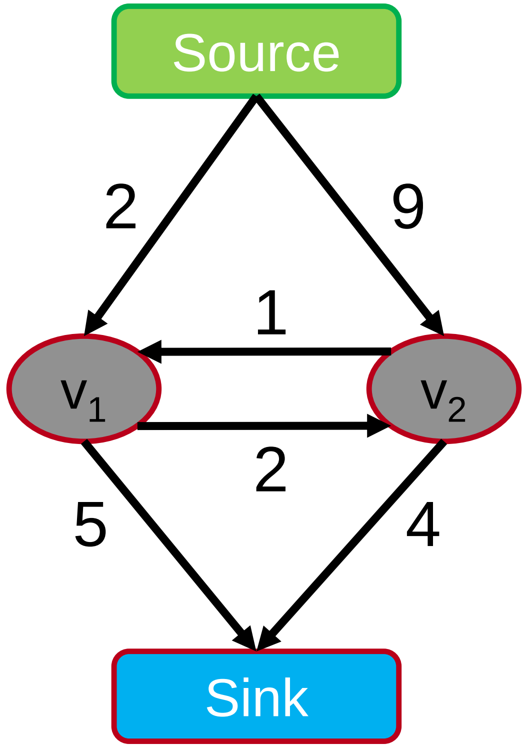
EM-style optimization

Slide credit: Yuri Boykov

[Boykov & Jolly, ICCV'01]

Recap: Maxflow Algorithms

Flow = 0



Augmenting Path Based Algorithms

1. Find path from source to sink with positive capacity
2. Push maximum possible flow through this path
3. Adjust the capacity of the used edges and record “residual flows“
4. Repeat until no path can be found

Algorithms assume non-negative capacity

Slide credit: Pushmeet Kohli

Recap: When Can s-t Graph Cuts Be Applied?

$$E(L) = \underbrace{\sum_p E_p(L_p)}_{\text{t-links}}^{\text{unary potentials}} + \underbrace{\sum_{pq \in N} E(L_p, L_q)}_{\text{n-links}}^{\text{pairwise potentials}} \qquad L_p \in \{s,t\}$$

- s-t graph cuts can only globally minimize binary energies that are submodular. [Boros & Hummer, 2002, Kolmogorov & Zabih, 2004]

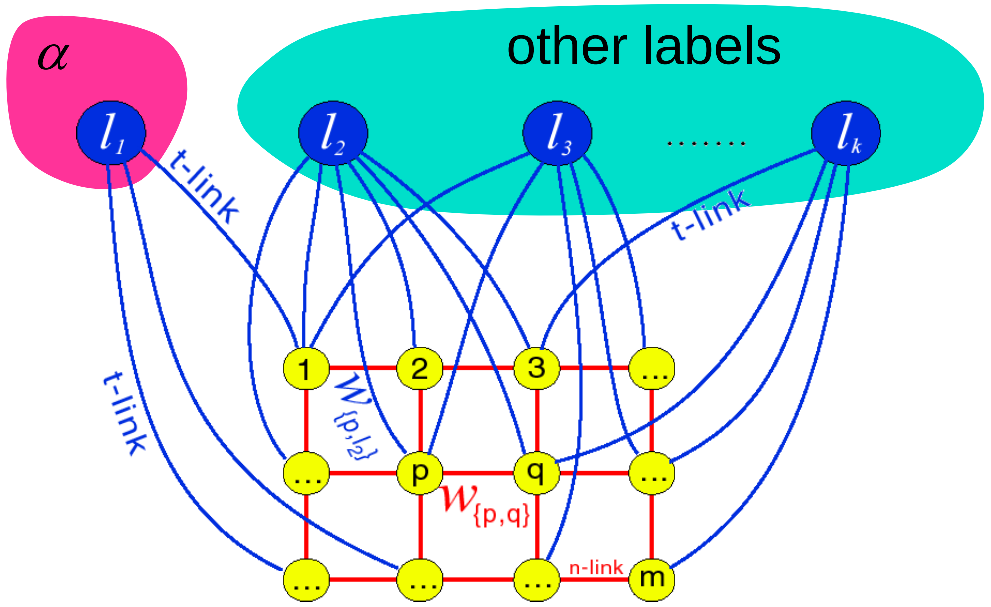
$\boxed{E(L) \text{ can be minimized by } s\text{-}t \text{ graph cuts}} \Leftrightarrow \boxed{E(s,s)+E(t,t) \le E(s,t)+E(t,s)}$

Submodularity ("convexity")

- Submodularity is the discrete equivalent to convexity.
  - Implies that every local energy minimum is a global minimum.
  - $\Rightarrow$ Solution will be globally optimal.

Recap: α-Expansion

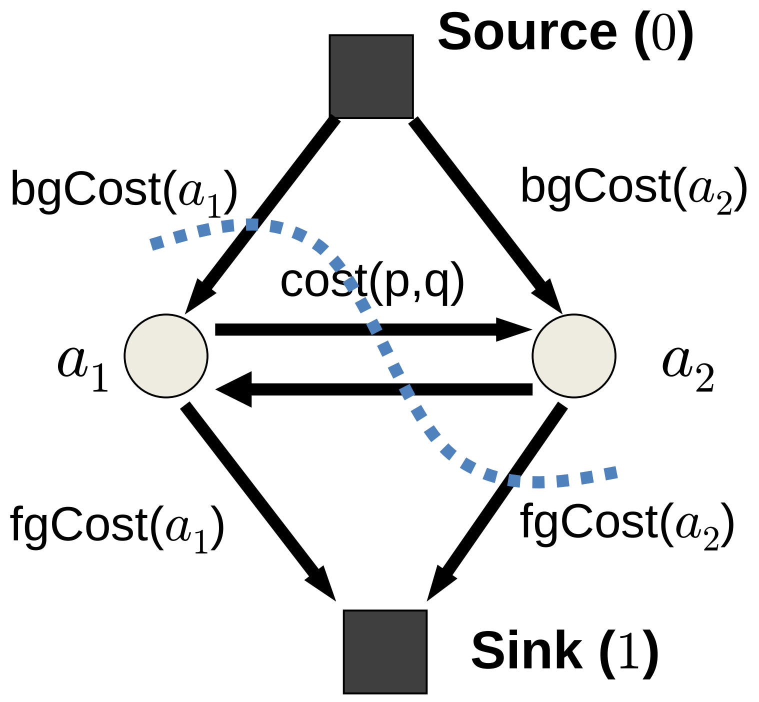
- Basic idea:
  - Break multi-way cut computation into a sequence of binary s-t cuts.



Slide credit: Yuri Boykov

Recap: Converting an MRF to an s-t Graph

```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q),  cost);
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));

// is the label of pixel p (0 or 1)
```



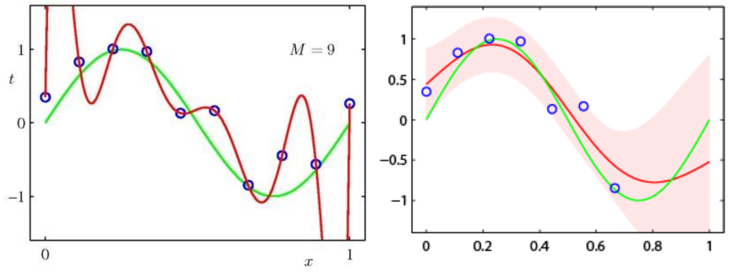
a_1 = bg a_2 = fg

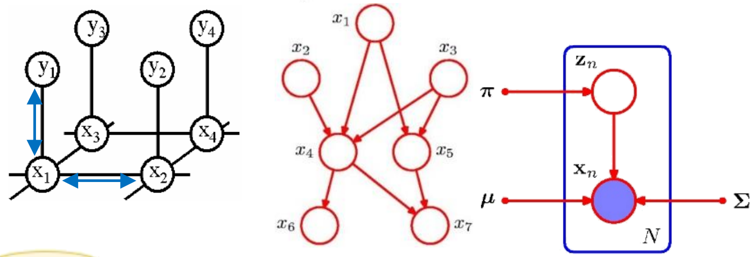
Slide credit: Pushmeet Kohli

Course Outline

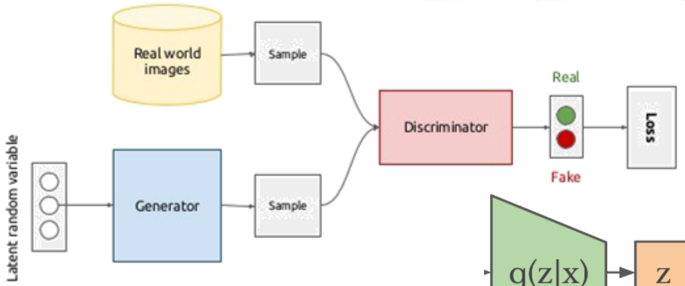
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

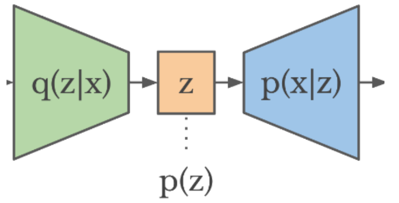
$$f : \mathcal{X} \to \mathbb{R}$$





y_1 y_2 y_3 y_4 x_1 x_2 x_3 x_4
x_1 x_2 x_3 x_4 x_5 x_6 x_7
π $\mathbf{z}_n$ $\mathbf{x}_n$ μ Σ N

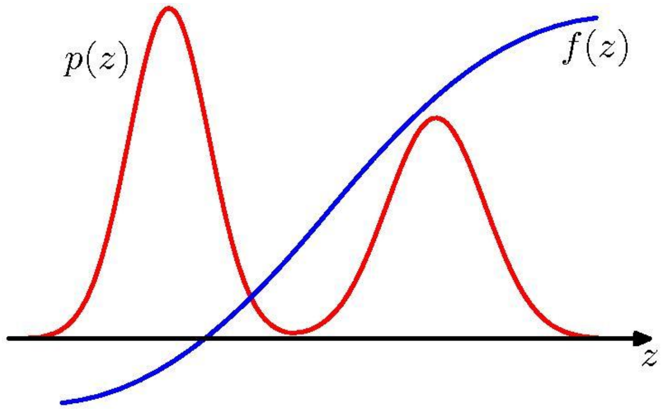




Recap: Sampling Idea

- Objective:
  - Evaluate expectation of a function $f(\mathbf{z})$ w.r.t. a probability distribution $p(\mathbf{z})$.

$$\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z}$$



- Sampling idea
  - Draw L independent samples $\mathbf{z}^{(l)}$ with l = 1,…,L from $p(\mathbf{z})$.
  - This allows the expectation to be approximated by a finite sum

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^l)$$

  - As long as the samples $\mathbf{z}^{(l)}$ are drawn independently from $p(\mathbf{z})$, then

$$\mathbb{E}[\hat{f}] = \mathbb{E}[f]$$

$\Rightarrow$ Unbiased estimate, independent of the dimension of $\mathbf{z}$!

Slide adapted from Bernt Schiele

Image source: C.M. Bishop, 2006

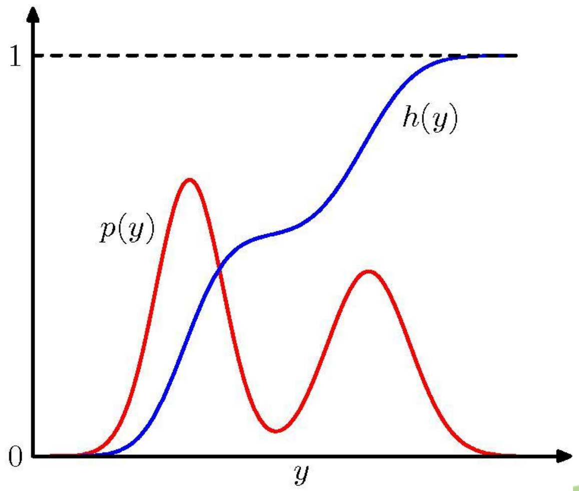
Recap: Transformation Method

- In general, assume we are given the pdf $p(\mathbf{x})$ and the corresponding cumulative distribution:

$$F(x) = \int_{-\infty}^{x} p(z)dz$$

  – To draw samples from this pdf, we can invert the cumulative distribution function:

$$u \sim Uniform(0,1) \Rightarrow F^{-1}(u) \sim p(x)$$



Slide credit: Bernt Schiele

Image source: C.M. Bishop, 2006

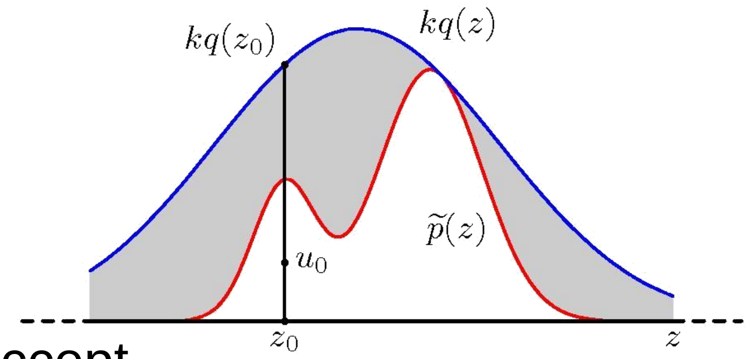
Recap: Rejection Sampling

see Exercise 4.2

- Assumptions
  - Sampling directly from $p(\mathbf{z})$ is difficult.
  - But we can easily evaluate $p(\mathbf{z})$ (up to some norm. factor Z_p):

$$p(\mathbf{z}) = \frac{1}{Z_p}\tilde{p}(\mathbf{z})$$

- Idea
  - We need some simpler distribution $q(\mathbf{z})$ (called proposal distribution) from which we can draw samples.
  - Choose a constant k such that: $\forall z : kq(z) \geq \tilde{p}(z)$
- Sampling procedure
  - Generate a number z_0 from $q(z)$.
  - Generate a number u_0 from the uniform distribution over $[0, kq(z_0)]$.
  - If $u_0 > \tilde{p}(z_0)$ reject sample, otherwise accept.



Slide adapted from Bernt Schiele

Image source: C.M. Bishop, 2006

Recap: Importance Sampling

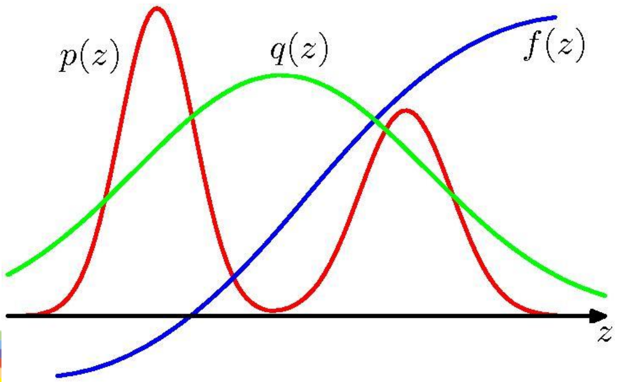
see Exercise 4.3

- Approach
  - Approximate expectations directly $\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z}$

    (but does not enable to draw samples from $p(\mathbf{z})$ directly).
- Idea
  - Use a proposal distribution $q(\mathbf{z})$ from which it is easy to sample.
  - Express expectations in the form of a finite sum over samples $\{\mathbf{z}^{(l)}\}$ drawn from $q(\mathbf{z})$.

$$\mathbb{E}[f] = \int f(\mathbf{z})p(\mathbf{z})d\mathbf{z} = \int f(\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z})}q(\mathbf{z})d\mathbf{z}$$

$$\simeq \frac{1}{L}\sum_{l=1}^{L} \underbrace{\frac{p(\mathbf{z}^{(l)})}{q(\mathbf{z}^{(l)})}}_{\text{Importance weights}} f(\mathbf{z}^{(l)})$$

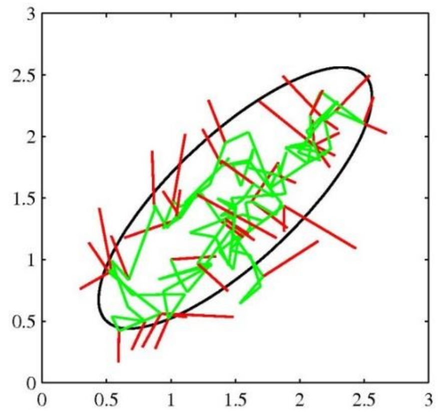


Slide adapted from Bernt Schiele

Image source: C.M. Bishop, 2006

Recap: MCMC – Markov Chain Monte Carlo

- Overview
  - Allows to sample from a large class of distributions.
  - Scales well with the dimensionality of the sample space.
- Idea
  - We maintain a record of the current state $\mathbf{z}^{(\tau)}$
  - The proposal distribution depends on the current state: $q(\mathbf{z}|\mathbf{z}^{(\tau)})$
  - The sequence of samples forms a Markov chain $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots$
- Approach
  - At each time step, we generate a candidate sample from the proposal distribution and accept the sample according to a criterion.
  - Different variants of MCMC for different criteria.



Slide adapted from Bernt Schiele

Image source: C.M. Bishop, 2006

Recap: Markov Chains – Properties

- Invariant distribution
  - A distribution is said to be invariant (or stationary) w.r.t. a Markov chain if each step in the chain leaves that distribution invariant.
  - Transition probabilities:

$$T\left(\mathbf{z}^{(m)}, \mathbf{z}^{(m+1)}\right) = p\left(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)}\right)$$

  - For homogeneous Markov chain, distribution $p^{\star}(\mathbf{z})$ is invariant if:

$$p^{\star}(\mathbf{z}) = \sum_{\mathbf{z}'} T(\mathbf{z}', \mathbf{z})\, p^{\star}(\mathbf{z}')$$

- Detailed balance
  - Sufficient (but not necessary) condition to ensure that a distribution is invariant:

$$p^{\star}(\mathbf{z}) T(\mathbf{z}, \mathbf{z}') = p^{\star}(\mathbf{z}') T(\mathbf{z}', \mathbf{z})$$

  - A Markov chain which respects *detailed balance* is reversible.

Slide credit: Bernt Schiele

Recap: Detailed Balance

- Detailed balance means
  - If we pick a state from the target distribution $p(\mathbf{z})$ and make a transition under T to another state, it is just as likely that we will pick $\mathbf{z}_A$ and go from $\mathbf{z}_A$ to $\mathbf{z}_B$ than that we will pick $\mathbf{z}_B$ and go from $\mathbf{z}_B$ to $\mathbf{z}_A$.

  - It can easily be seen that a transition probability that satisfies detailed balance w.r.t. a particular distribution will leave that distribution invariant, because

$$\begin{aligned}\sum_{\mathbf{z}'} p^\star(\mathbf{z}')T(\mathbf{z}',\mathbf{z}) &= \sum_{\mathbf{z}'} p^\star(\mathbf{z})T(\mathbf{z},\mathbf{z}') \\ &= p^\star(\mathbf{z})\sum_{\mathbf{z}'} p(\mathbf{z}'|\mathbf{z}) = p^\star(\mathbf{z})\end{aligned}$$

Recap: MCMC – Metropolis Algorithm

see Exercise 4.4

- Metropolis algorithm [Metropolis et al., 1953]
  - Proposal distribution is symmetric: $q(\mathbf{z}_A|\mathbf{z}_B) = q(\mathbf{z}_B|\mathbf{z}_A)$
  - The new candidate sample $\mathbf{z}^*$ is accepted with probability

$$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^\star)}{\tilde{p}(\mathbf{z}^{(\tau)})}\right)$$

  $\Rightarrow$ New candidate samples always accepted if $\tilde{p}(\mathbf{z}^\star) \geq \tilde{p}(\mathbf{z}^{(\tau)})$
  - The algorithm sometimes accepts a state with lower probability.

- Metropolis-Hastings algorithm
  - Generalization: Proposal distribution not necessarily symmetric.
  - The new candidate sample $\mathbf{z}^*$ is accepted with probability

$$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^\star)q_k(\mathbf{z}^{(\tau)}|\mathbf{z}^\star)}{\tilde{p}(\mathbf{z}^{(\tau)})q_k(\mathbf{z}^\star|\mathbf{z}^{(\tau)})}\right)$$

  - where k labels the members of the set of considered transitions.

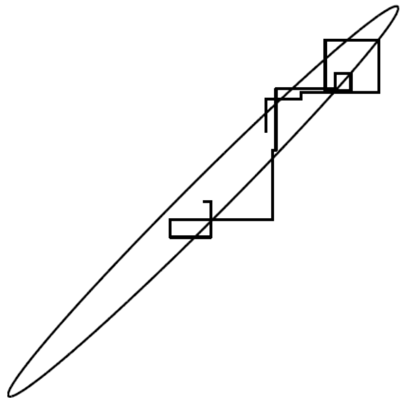
Slide adapted from Bernt Schiele

Recap: Gibbs Sampling

- Approach
  - MCMC-algorithm that is simple and widely applicable.
  - May be seen as a special case of Metropolis-Hastings.

- Idea
  - Sample variable-wise: replace $\mathbf{z}_i$ by a value drawn from the distribution $p(z_i|\mathbf{z}_{\setminus i})$.
    - This means we update one coordinate at a time.
  - Repeat procedure either by cycling through all variables or by choosing the next variable.

- Properties
  - The algorithm always accepts!
  - Completely parameter free.
  - Can also be applied to subsets of variables.

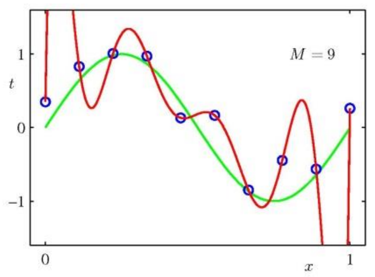


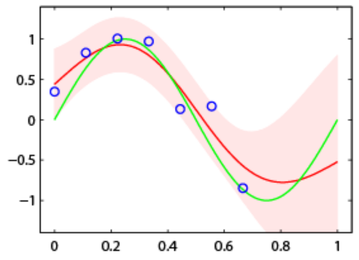
Slide credit: Bernt Schiele

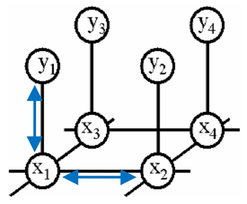
Course Outline

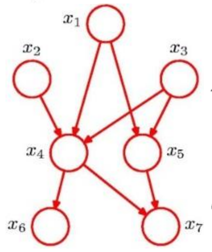
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

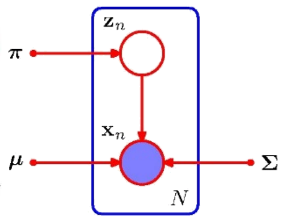
$$f : \mathcal{X} \rightarrow \mathbb{R}$$

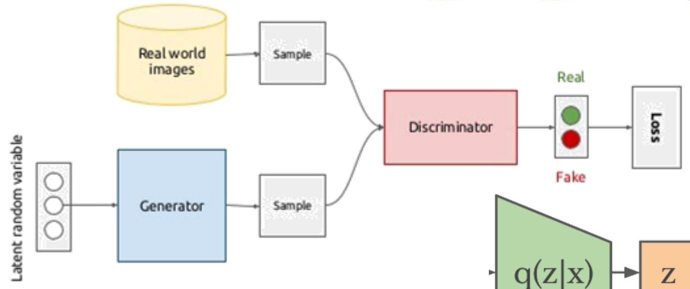


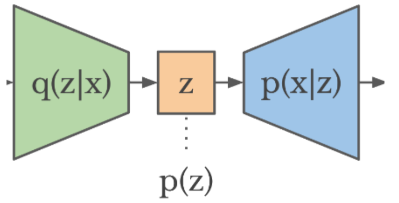








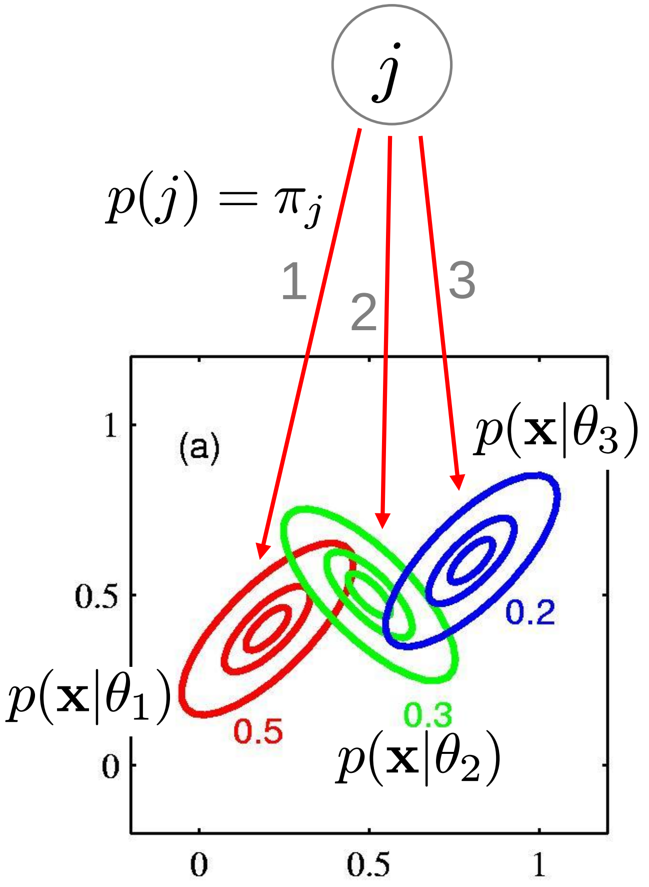




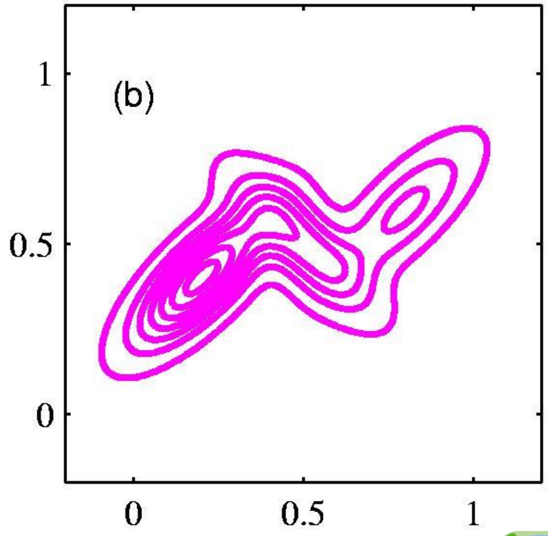
Recap: Mixtures of Gaussians

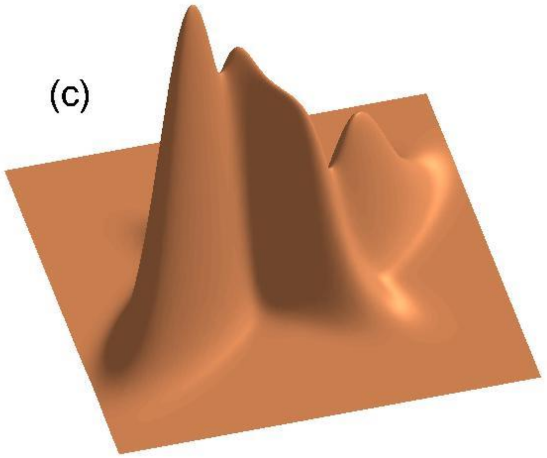
- "Generative model"

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



$$p(\mathbf{x}|\theta) = \sum_{j=1}^{3} \pi_j p(\mathbf{x}|\theta_j)$$





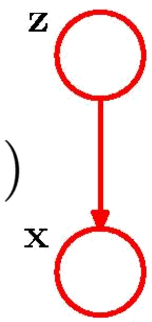
Slide credit: Bernt Schiele

Image source: C.M. Bishop, 2006

Recap: GMMs as Latent Variable Models

- Write GMMs in terms of latent variables $\mathbf{z}$
  - Marginal distribution of $\mathbf{x}$

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

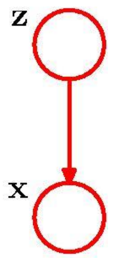


- Advantage of this formulation
  - We have represented the marginal distribution in terms of latent variables $\mathbf{z}$.
  - Since $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$, there is a corresponding latent variable $\mathbf{z}_n$ for each data point $\mathbf{x}_n$.
  - We are now able to work with the joint distribution $p(\mathbf{x}, \mathbf{z})$ instead of the marginal distribution $p(\mathbf{x})$.

$\Rightarrow$ *This will lead to significant simplifications…*

Recap: Sampling from a Gaussian Mixture

- MoG Sampling
  - We can use ancestral sampling to generate random samples from a Gaussian mixture model.
    1. Generate a value $\hat{\mathbf{z}}$ from the marginal distribution $p(\mathbf{z})$.
    2. Generate a value $\hat{\mathbf{x}}$ from the conditional distribution $p(\mathbf{x}|\hat{\mathbf{z}})$.



Samples from the joint $p(\mathbf{x}, \mathbf{z})$

Samples from the marginal $p(\mathbf{x})$

Evaluating the responsibilities $\gamma(z_{nk})$

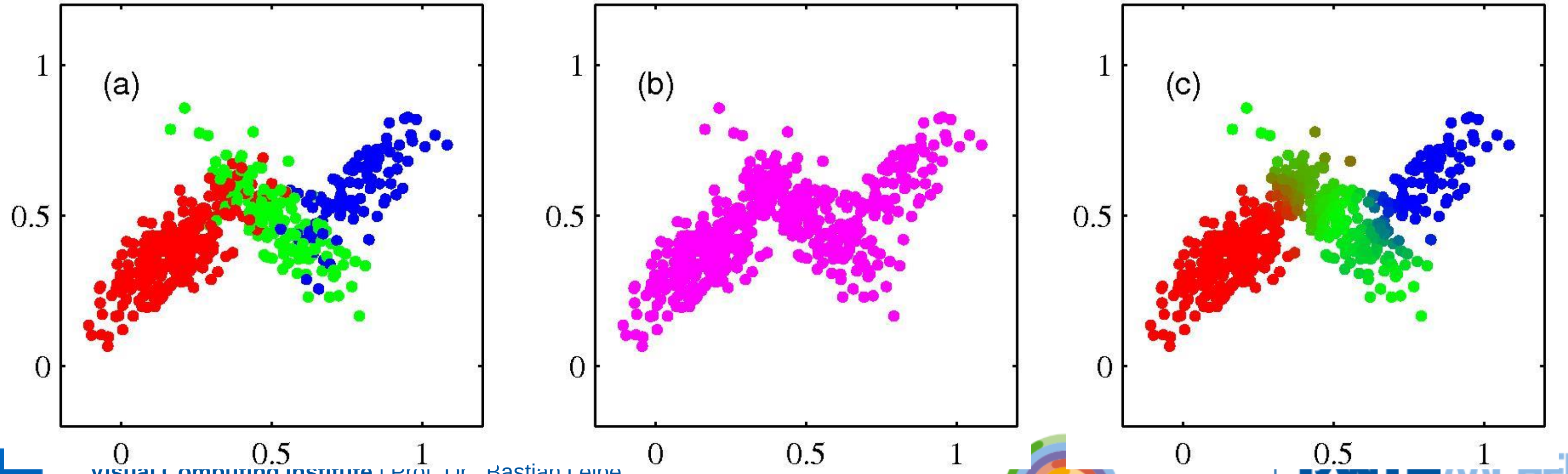


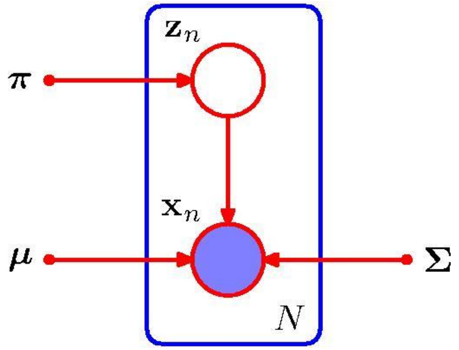
Image source: C.M. Bishop, 2006

Recap: Gaussian Mixtures Revisited

- Applying the latent variable view of EM
  - Goal is to maximize the log-likelihood using the observed data $\mathbf{X}$

$$\log p(\mathbf{X}|\boldsymbol{\theta}) = \log \left\{ \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) \right\}$$

  - Corresponding graphical model:



  - Suppose we are additionally given the values of the latent variables $\mathbf{Z}$.
  - The corresponding graphical model for the complete data now looks like this:

$\Rightarrow$ Straightforward to marginalize…

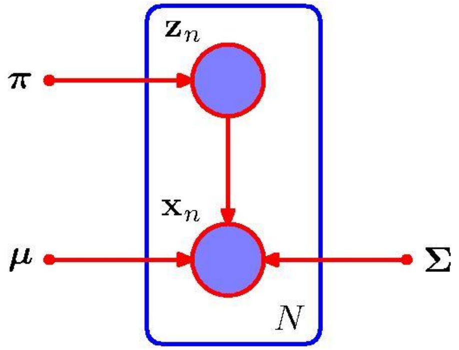


Image source: C.M. Bishop, 2006

Recap: Alternative View of EM

- In practice, however,…
  - We are not given the complete data set $\{\mathbf{X},\mathbf{Z}\}$, but only the incomplete data $\mathbf{X}$. All we can compute about $\mathbf{Z}$ is the posterior distribution $p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta})$.
  - Since we cannot use the complete-data log-likelihood, we consider instead its expected value under the posterior distribution of the latent variables:

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

  - This corresponds to the E-step of the EM algorithm.
  - In the subsequent M-step, we then maximize the expectation to obtain the revised parameter set θ^{new}.

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

Recap: General EM Algorithm

- Algorithm
  1. Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}}$
  2. E-step: Evaluate $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$
  3. M-step: Evaluate $\boldsymbol{\theta}^{\text{new}}$ given by

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

where

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

  4. While not converged, let $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$ and return to step 2.

Recap: MAP-EM

- Modification for MAP
  - The EM algorithm can be adapted to find MAP solutions for models for which a prior $p(\boldsymbol{\theta})$ is defined over the parameters.
  - Only changes needed:

2. E-step: Evaluate $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$

3. M-step: Evaluate $\boldsymbol{\theta}^{\text{new}}$ given by

$$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \; \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) + \log p(\boldsymbol{\theta})$$

$\Rightarrow$ Suitable choices for the prior will remove the ML singularities!

Recap: Monte Carlo EM

- EM procedure
  - M-step: Maximize expectation of complete-data log-likelihood

$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \int p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) \mathrm{d}\mathbf{Z}$$

  - For more complex models, we may not be able to compute this analytically anymore…

- Idea
  - Use sampling to approximate this integral by a finite sum over samples $\{\mathbf{Z}^{(l)}\}$ drawn from the current estimate of the posterior

$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) \sim \frac{1}{L} \sum_{l=1}^{L} \log p(\mathbf{X}, \mathbf{Z}^{(l)}|\boldsymbol{\theta})$$

  - This procedure is called the Monte Carlo EM algorithm.

Recap: EM as Variational Inference

- Decomposition
  - Introduce a distribution $q(\mathbf{Z})$ over the latent variables. For any choice of $q(\mathbf{Z})$, the following decomposition holds

$$\log p(\mathbf{X}|\theta) = \mathcal{L}(q, \boldsymbol{\theta}) + KL(q \parallel p)$$

  - where

$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \log \left\{ \frac{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}{q(\mathbf{Z})} \right\}$$

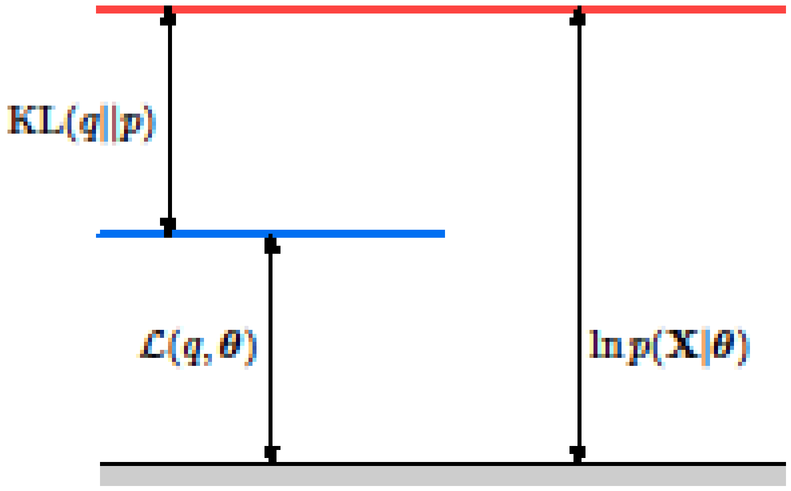
$$KL(q \parallel p) = -\sum_{\mathbf{Z}} q(\mathbf{Z}) \log \left\{ \frac{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\}$$

  - $KL(q \parallel p)$ is the Kullback-Leibler divergence between the distribution $q(\mathbf{Z})$ and the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$.
  - $\mathcal{L}(q, \boldsymbol{\theta})$ is a functional of the distribution $q(\mathbf{Z})$ and a function of the parameters $\boldsymbol{\theta}$. Since KL $\geq 0, \mathcal{L}(q, \boldsymbol{\theta})$ is a lower bound on $\log p(\mathbf{X}|\theta)$.

Recap: Analysis of EM

- Decomposition

$$\log p(\mathbf{X}|\theta) \;= \mathcal{L}(q, \boldsymbol{\theta}) + KL(q \parallel p)$$

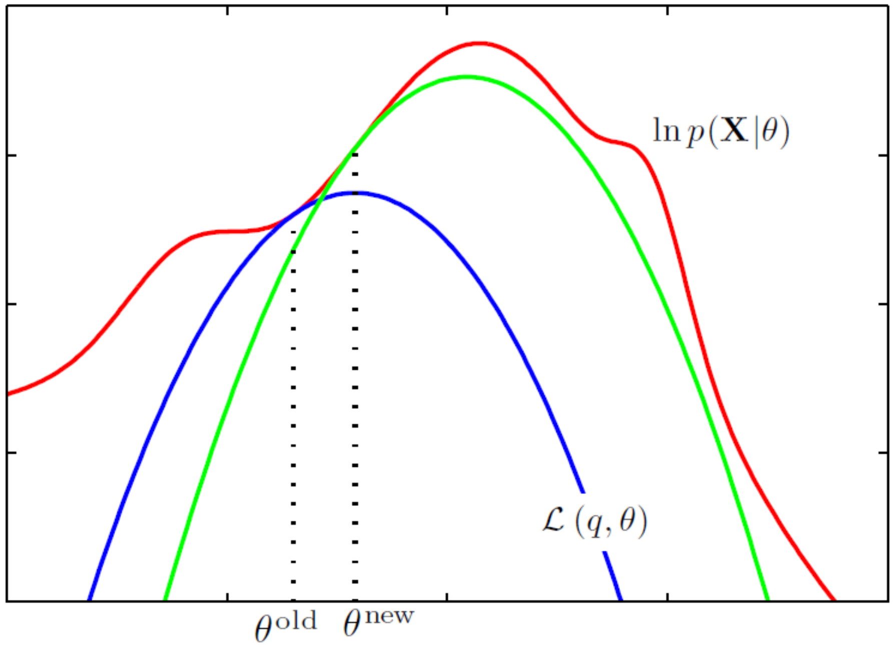


$\mathrm{KL}(q\|p)$
$\mathcal{L}(q,\boldsymbol{\theta})$
$\ln p(\mathbf{X}|\boldsymbol{\theta})$

- Interpretation
  - $\mathcal{L}(q, \boldsymbol{\theta})$ is a lower bound on $\log p(\mathbf{X}|\boldsymbol{\theta})$.
  - The approximation comes from the fact that we use an approximative distribution $q(\boldsymbol{Z}) = p\left(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old}\right)$ Instead of the (unknown) real posterior.
  - The KL divergence measures the difference between the approximative distribution $q(\boldsymbol{Z})$ and the real posterior $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$.
  - In every EM iteration, we try to make this difference smaller.

Recap: Analysis of EM

- Visualization in the space of parameters

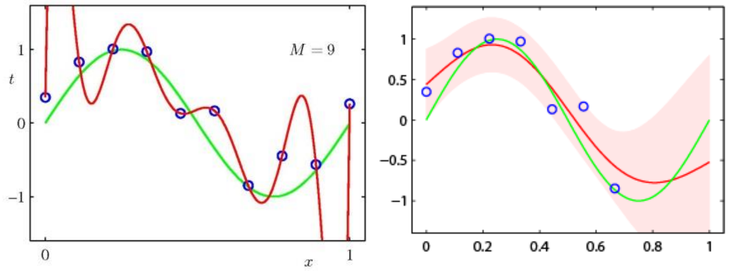


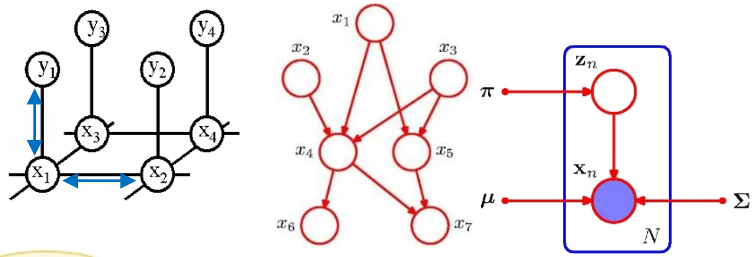
- The EM algorithm alternately…
  - Computes a lower bound on the log-likelihood for the current parameters values
  - And then maximizes this bound to obtain the new parameter values.

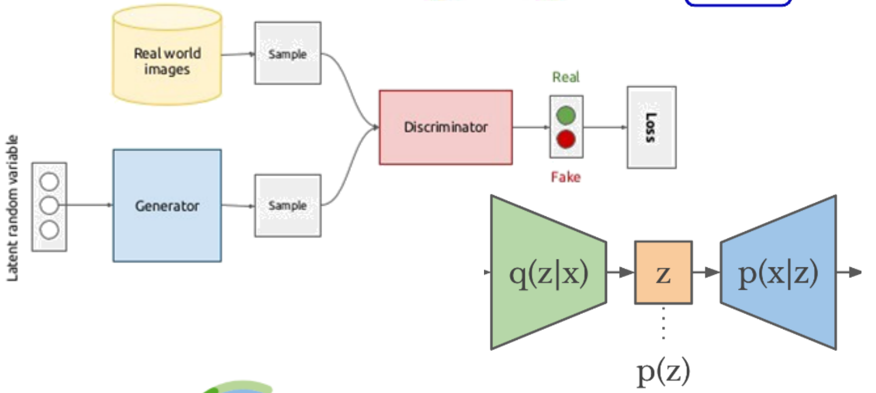
Course Outline

- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Bayesian Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

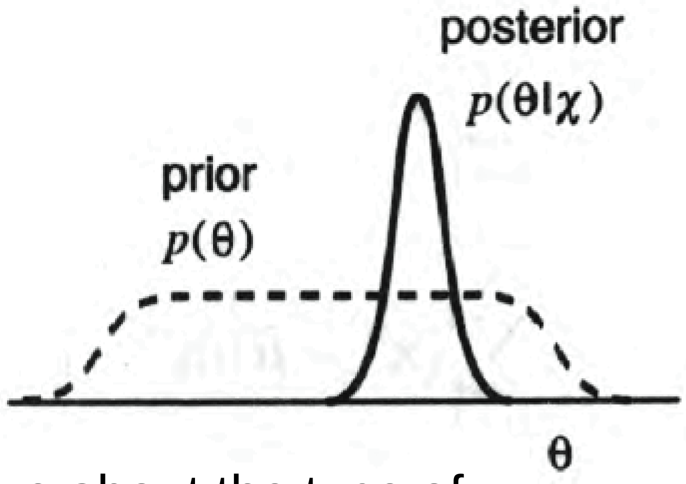






Recap: Bayesian Estimation

- Conceptual shift
  - Maximum Likelihood views the true parameter vector θ to be unknown, but fixed.
  - In Bayesian learning, we consider θ to be a random variable.
- This allows us to use knowledge about the parameters θ
  - i.e., to use a prior for θ
  - Training data then converts this prior distribution on θ into a posterior probability density.



  - The prior thus encodes knowledge we have about the type of distribution we expect to see for θ.

Slide adapted from Bernt Schiele

Recap: Bayesian Estimation

- Discussion

Likelihood of the parametric form θ given the data set X.

Estimate for x based on parametric form θ

Prior for the parameters θ

$$p(x|X) = \int \frac{p(x|\theta)L(\theta)p(\theta)}{\underbrace{\int L(\theta)p(\theta)d\theta}}d\theta$$

Normalization: integrate over all possible values of θ

$\Rightarrow$ *The parameter values θ are not the goal, just a means to an end.*

Recap: Conjugate Priors

- Problem: How to evaluate the integrals?
  - We will see that if likelihood and prior have the same functional form $c \cdot f(x)$, then the analysis will be greatly simplified and the integrals will be solvable in closed form.

$$\begin{aligned} p(X|\theta)p(\theta) &= \prod_{x_n} c_1 f(x_n, \theta) c_2 f(\theta, \alpha) \\ &= \prod_{x_n} c f(x_n, \theta, \alpha) \end{aligned}$$

  - Such an algebraically convenient choice is called a conjugate prior. Whenever possible, we should use it.
  - To do this, we need to know for each probability distribution what is its conjugate prior.
- What to do when we cannot use the conjugate prior?

  $\Rightarrow$ *Use approximate inference methods.*

Recap: The Dirichlet Distribution

- Dirichlet Distribution
  - Conjugate prior for the Categorical and the Multinomial distrib.

$$\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1} \qquad \text{with} \qquad \alpha_0 = \sum_{k=1}^{K} \alpha_k$$

  - Symmetric version (with concentration parameter α)

$$\mathrm{Dir}(\boldsymbol{\mu}|\alpha) = \frac{\Gamma(\alpha)}{\Gamma(\alpha/K)^K} \prod_{k=1}^{K} \mu_k^{\alpha/K - 1}$$

  - Properties (symmetric version)

$$\begin{aligned}
\mathbb{E}[\mu_k] &= \frac{\alpha_k}{\alpha_0} &&= \frac{1}{K} \\
\mathrm{var}[\mu_k] &= \frac{\alpha_k(\alpha_0 - \alpha_k)}{\alpha_0^2(\alpha_0 + 1)} &&= \frac{K-1}{K^2(\alpha+1)} \\
\mathrm{cov}[\mu_j \mu_k] &= -\frac{\alpha_j \alpha_k}{\alpha_0^2(\alpha_0 + 1)} &&= -\frac{1}{K^2(\alpha+1)}
\end{aligned}$$

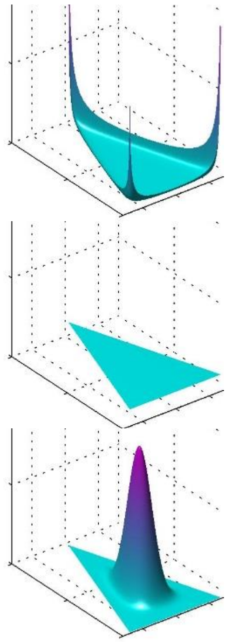


Image source: C. Bishop, 2006

Recap: Bayesian Mixture Models

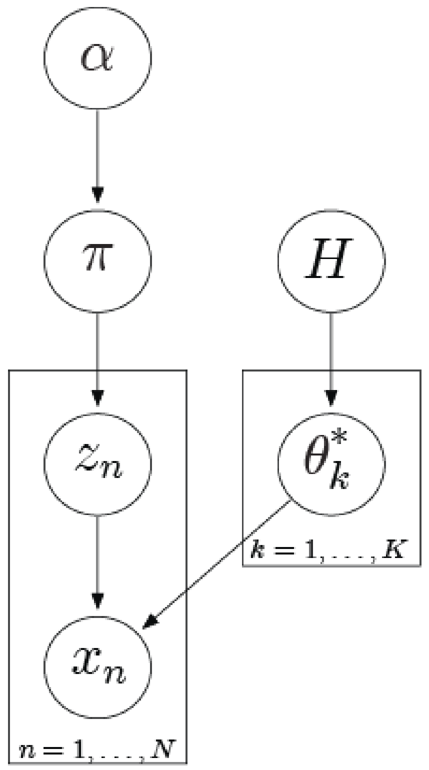
- Let's be Bayesian about mixture models
  - Place priors over our parameters
  - Again, introduce variable $\mathbf{z}_n$ as indicator which component data point $\mathbf{x}_n$ belongs to.

$$\begin{aligned} \mathbf{z}_n | \boldsymbol{\pi} &\sim \text{Multinomial}(\boldsymbol{\pi}) \\ \mathbf{x}_n | \mathbf{z}_n = k, \boldsymbol{\mu}, \boldsymbol{\Sigma} &\sim \mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k) \end{aligned}$$

  - Introduce conjugate priors over parameters

$$\begin{aligned} \boldsymbol{\pi} &\sim \text{Dirichlet}(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}) \\ \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k &\sim H = \mathcal{N} - \mathcal{IW}(0, s, d, \phi) \end{aligned}$$

"Normal – Inverse Wishart"



Slide inspired by Yee Whye Teh

Recap: Bayesian Mixture Models

- Full Bayesian Treatment
  - Given a dataset, we are interested in the cluster assignments

$$p(\mathbf{Z}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})}{\sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})}$$

  where the likelihood is obtained by marginalizing over the parameters θ

$$\begin{aligned} p(\mathbf{X}|\mathbf{Z}) &= \int p(\mathbf{X}|\mathbf{Z},\boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} \\ &= \int \prod_{n=1}^{N}\prod_{k=1}^{K} p(\mathbf{x}_n|z_{nk},\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k|H)\mathrm{d}\boldsymbol{\theta} \end{aligned}$$

- The posterior over assignments is intractable!
  - Denominator requires summing over all possible partitions of the data into K groups!

$\Rightarrow$ Need efficient approximate inference methods to solve this...

Recap: Mixture Models with Dirichlet Priors

- Integrating out the mixing proportions π

$$p(\mathbf{z}|\alpha) = \int p(\mathbf{z}|\boldsymbol{\pi})p(\boldsymbol{\pi}|\alpha)\mathrm{d}\boldsymbol{\pi}$$

$$= \frac{\Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)}$$

- Conditional probabilities
  - Examine the conditional of $\mathbf{z}_n$ given all other variables $\mathbf{z}_{-n}$

$$p(z_{nk} = 1|\mathbf{z}_{-n}, \alpha) = \frac{p(z_{nk} = 1, \mathbf{z}_{-n}|\alpha)}{p(\mathbf{z}_{-n}|\alpha)}$$

$$= \frac{N_{-n,k} + \alpha/K}{N - 1 + \alpha} \qquad N_{-n,k} \overset{\text{def}}{=} \sum_{i=1, i\neq n}^{N} z_{ik}$$

$\Rightarrow$ The more populous a class is, the more likely it is to be joined!

Slide adapted from Zoubin Gharamani

Recap: Infinite Dirichlet Mixture Models

- Conditional probabilities: Finite K

$$p(z_{nk} = 1 | \mathbf{z}_{-n}, \alpha) = \frac{N_{-n,k} + \alpha/K}{N - 1 + \alpha}, \qquad N_{-n,k} \overset{\text{def}}{=} \sum_{i=1, i \neq n}^{N} z_{ik}$$

- Conditional probabilities: Infinite K
  - Taking the limit as $K \to \infty$ yields the conditionals

$$p(z_{nk} = 1 | \mathbf{z}_{-n}, \alpha) = \begin{cases} \frac{N_{-n,k}}{N-1+\alpha} & \text{if } k \text{ represented} \\ \frac{\alpha}{N-1+\alpha} & \text{if all } k \text{ not represented} \end{cases}$$

  - Left-over mass $\alpha \Rightarrow$ countably infinite number of indicator settings

Slide adapted from Zoubin Gharamani

Recap: Gibbs Sampling for Finite Mixtures

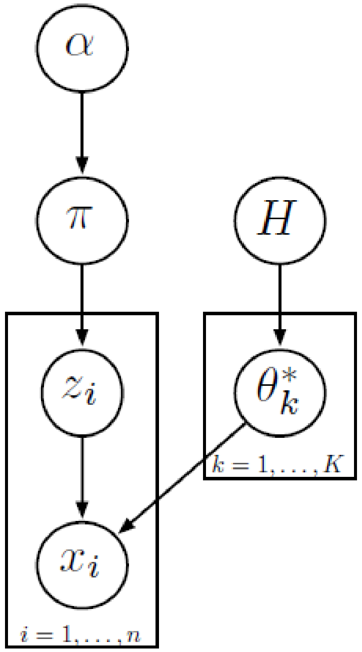
- We need approximate inference here
  - Gibbs Sampling: Conditionals are simple to compute

$$p(\mathbf{z}_n = k|\text{others}) \propto \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$\boldsymbol{\pi} \mid \mathbf{z} \sim \text{Dir}(N_1 + \alpha/K, \dots, N_K + \alpha/K)$$

$$\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k|\text{others} \sim \mathcal{N} - \mathcal{IW}(v', s', d', \phi')$$

- However, this will be rather inefficient…
  - In each iteration, algorithm can only change the assignment for individual data points.
  - There are often groups of data points that are associated with high probability to the same component. $\Rightarrow$ Unlikely that group is moved.
  - Better performance by collapsed Gibbs sampling which integrates out the parameters π, μ, $\boldsymbol{\Sigma}$.



Slide adapted from Yee Whye Teh

Image source: Yee Whye Teh

Recap: Collapsed Finite Bayesian Mixture

- More efficient algorithm
  - Conjugate priors allow analytic integration of some parameters
  - Resulting sampler operates on reduced space of cluster assignments (implicitly considers all possible cluster shapes)

- Procedure
  - The model implies the factorization

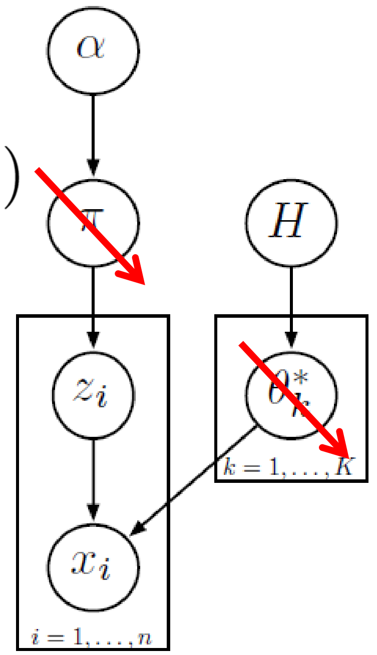
$$p(\mathbf{z}_n|\mathbf{z}_{-n},\mathbf{x},\alpha,H) \propto p(\mathbf{z}_n|\mathbf{z}_{-n},\alpha)p(\mathbf{x}_n|\mathbf{z},\mathbf{x}_{-n},H)$$

  - Derive

$$p(\mathbf{z}|\alpha) = \int p(\mathbf{z}|\boldsymbol{\pi})p(\boldsymbol{\pi}|\alpha)\mathrm{d}\boldsymbol{\pi} \quad ✓$$

$$p(\mathbf{x}_n|\mathbf{z}_n,H) = \int \sum_{k=1}^{K} z_{nk}p(\mathbf{x}_n|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k|H)\mathrm{d}\boldsymbol{\theta} \quad ✓$$

$\Rightarrow$ Conjugate prior, Normal - Inverse Wishart



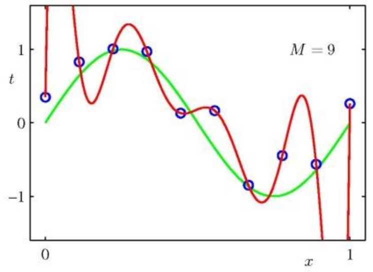
Slide adapted from Erik Sudderth

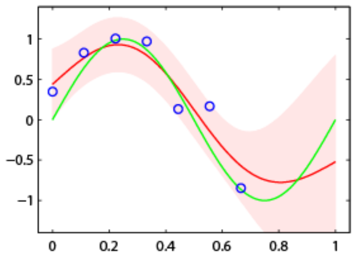
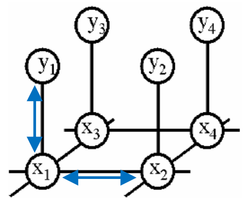
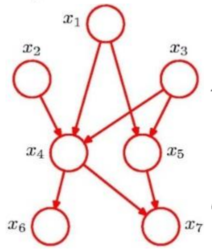
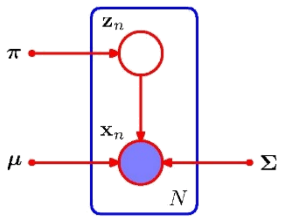
Image source: Yee Whye Teh

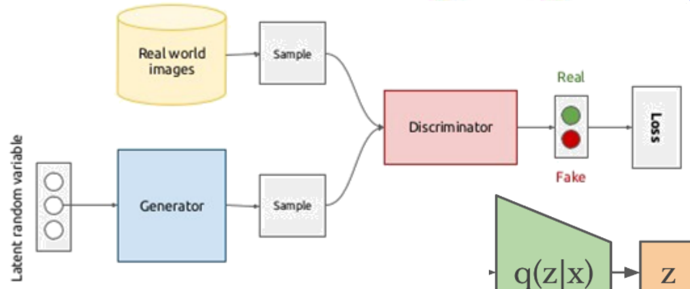
Course Outline

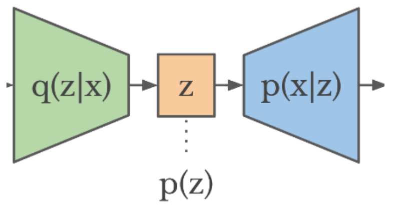
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

$$f : \mathcal{X} \to \mathbb{R}$$



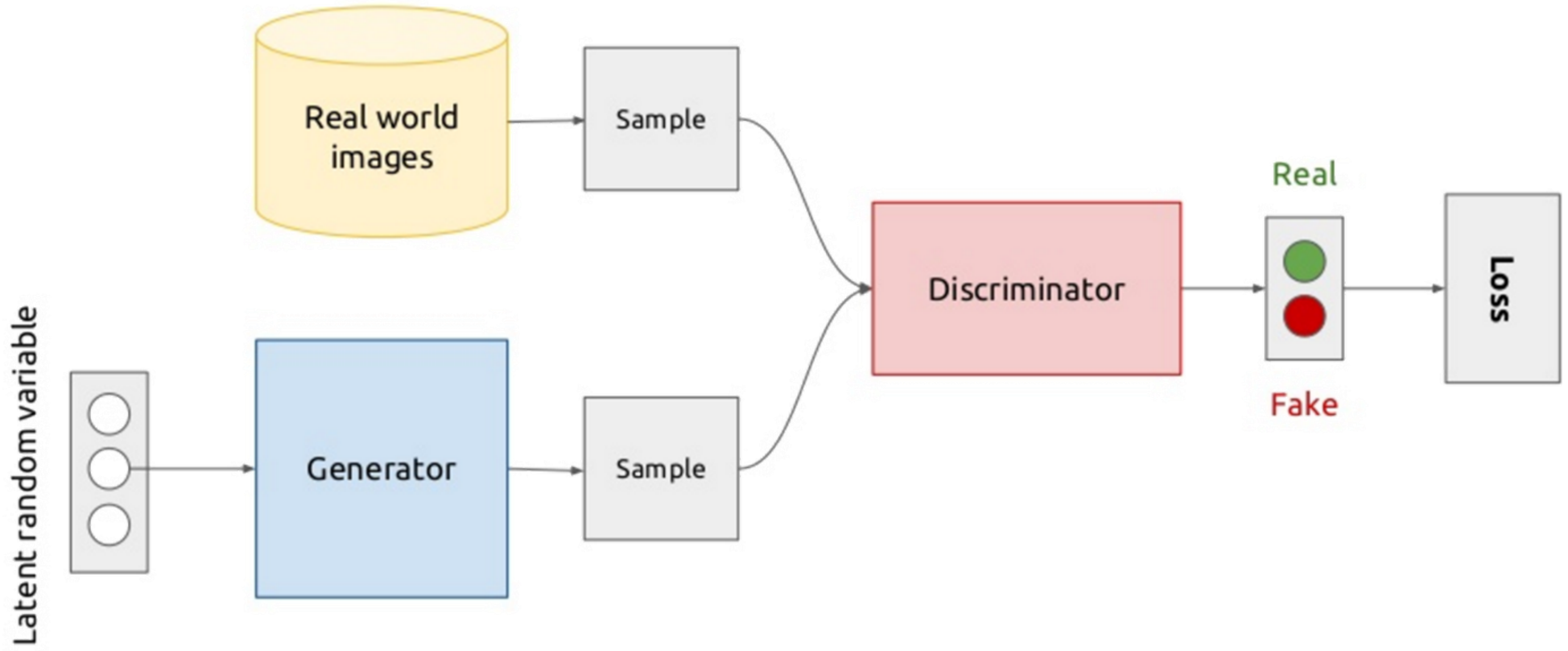









Recap: Generative Adversarial Networks (GANs)

- Conceptual view



- Main idea
  - Simultaneously train an image generator G and a discriminator D.
  - Interpreted as a two-player game

Image credit: Kevin McGuiness

Recap: GAN Loss Function

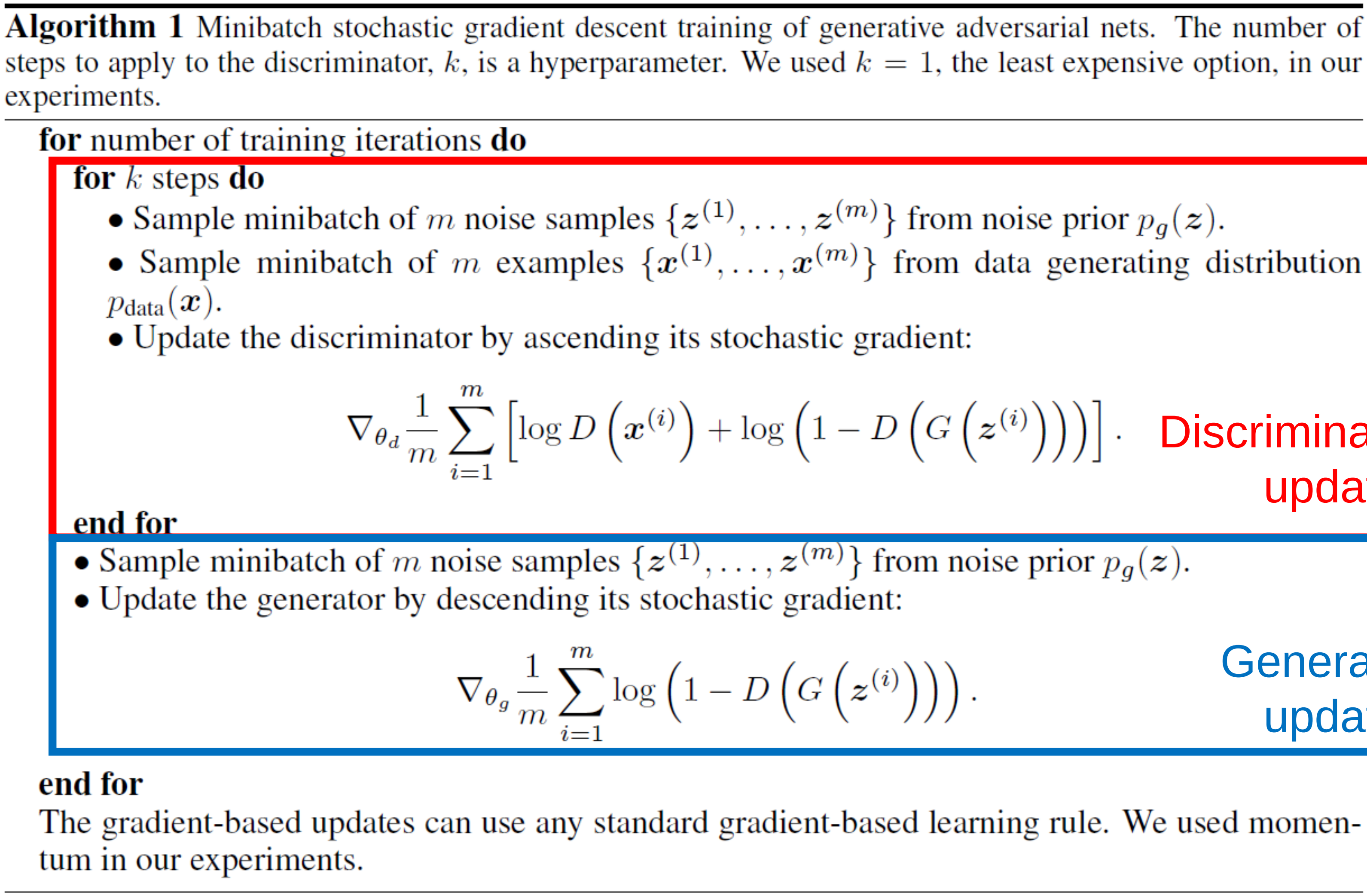
- This corresponds to a two-player minimax game:

$$\min_G \max_D V(D,G) = \boxed{\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})]} + \boxed{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1 - D\left(G(\boldsymbol{z})\right)\right)\right]}$$

- Explanation
  - Train D to maximize the probability of assigning the correct label to both training examples and samples from G.
  - Simultaneously train G to minimize $\log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$.

- The Nash equilibrium of this game is achieved at
  - $p_g(\boldsymbol{x}) = p_{data}(\boldsymbol{x}) \quad \forall \boldsymbol{x}$
  - $D(\boldsymbol{x}) = \frac{1}{2} \quad \forall \boldsymbol{x}$

GAN Algorithm



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

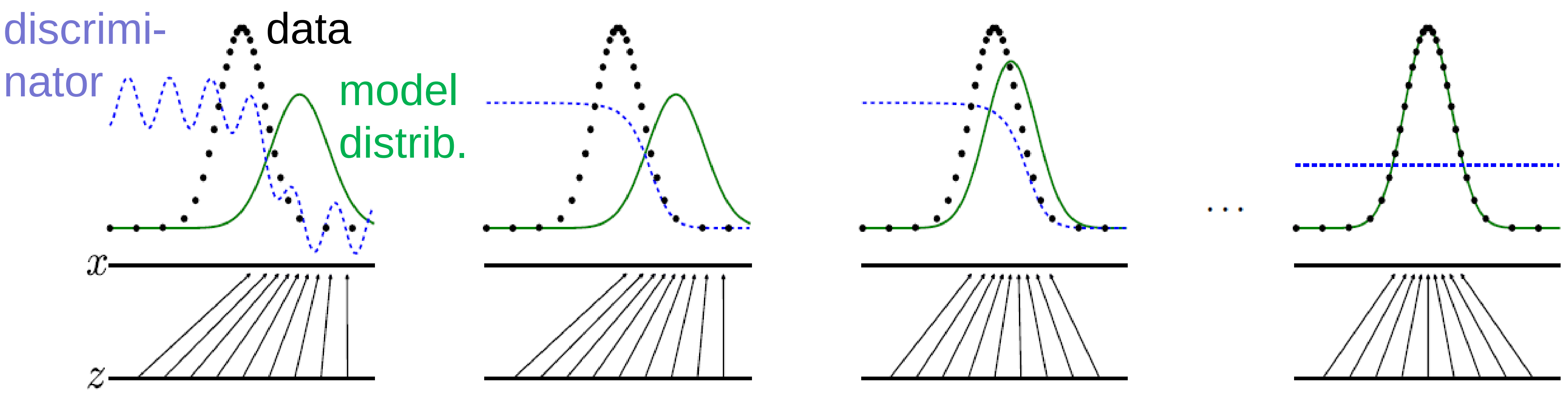
```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$
    end for
    • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Discriminator updates

Generator updates

Recap: Intuition behind GANs

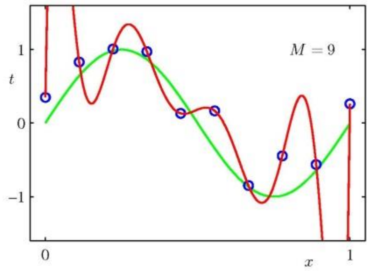


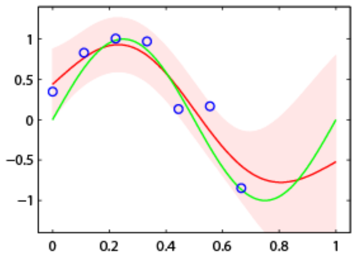
- Behavior near convergence
  - In the inner loop, D is trained to discriminate samples from data.
  - Gradient of D guides G to flow to regions that are more likely to be classified as data.
  - After several steps of training, G and D will reach a point at which they cannot further improve, because $p_g = p_{data}$.
  - Now, the discriminator is unable to differentiate between the two distributions, i.e., $D(\boldsymbol{x}) = 0.5$.

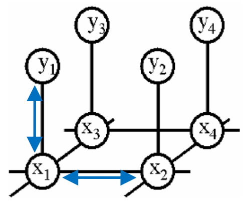
Course Outline

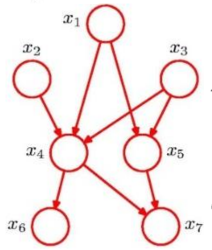
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

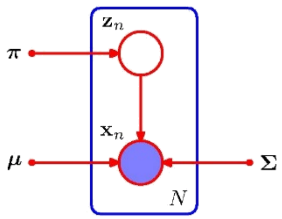
$$f : \mathcal{X} \rightarrow \mathbb{R}$$

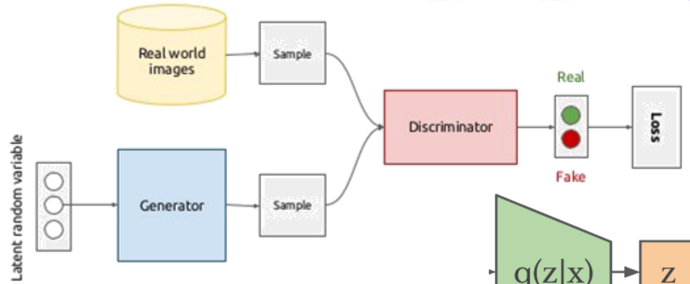


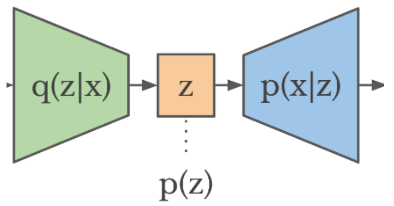




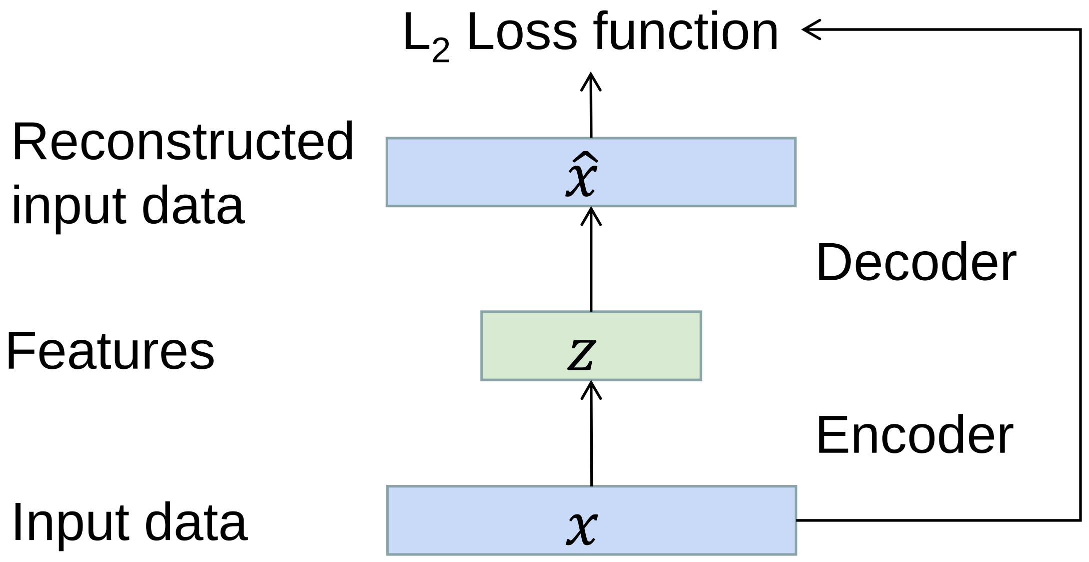








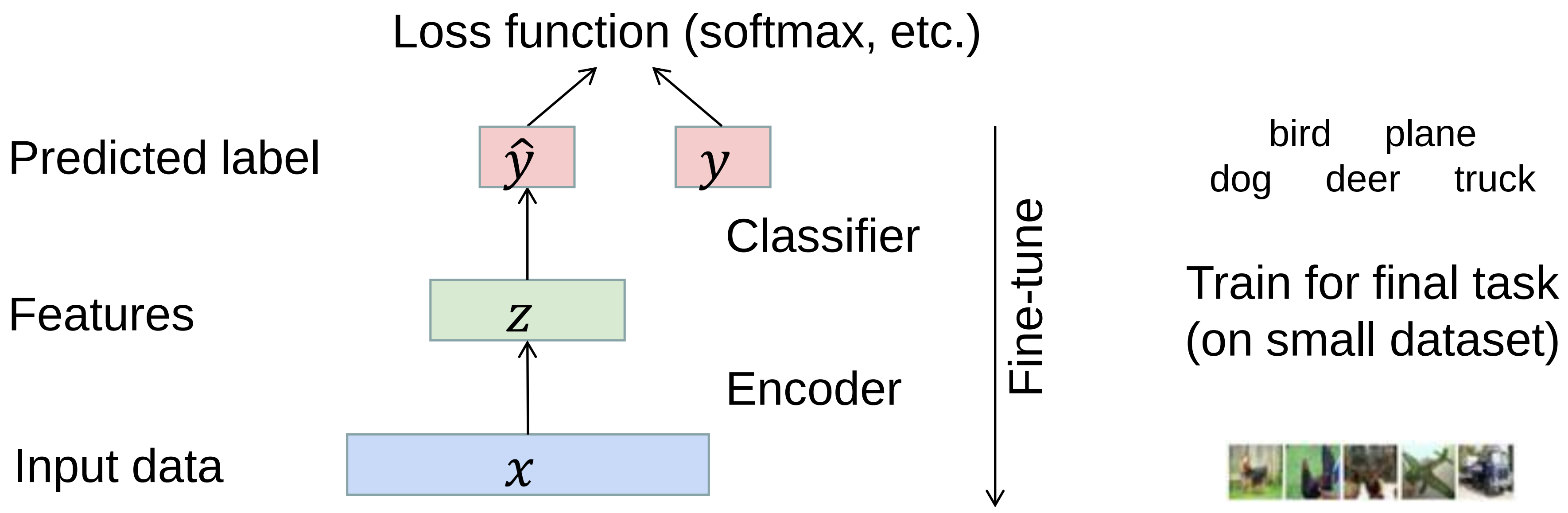
Recap: Autoencoders



- How to learn such a feature representation?
  - Unsupervised learning approach for learning a lower-dimensional feature representation $\mathbf{z}$ from unlabeled input data $\mathbf{x}$.
  - $\mathbf{z}$ usually smaller than $\mathbf{x}$ (dimensionality reduction)
  - Want to capture meaningful factors of variation in the data Train such that features can be used to reconstruct original data.

Slide credit: Feifei Li

Recap: Autoencoders

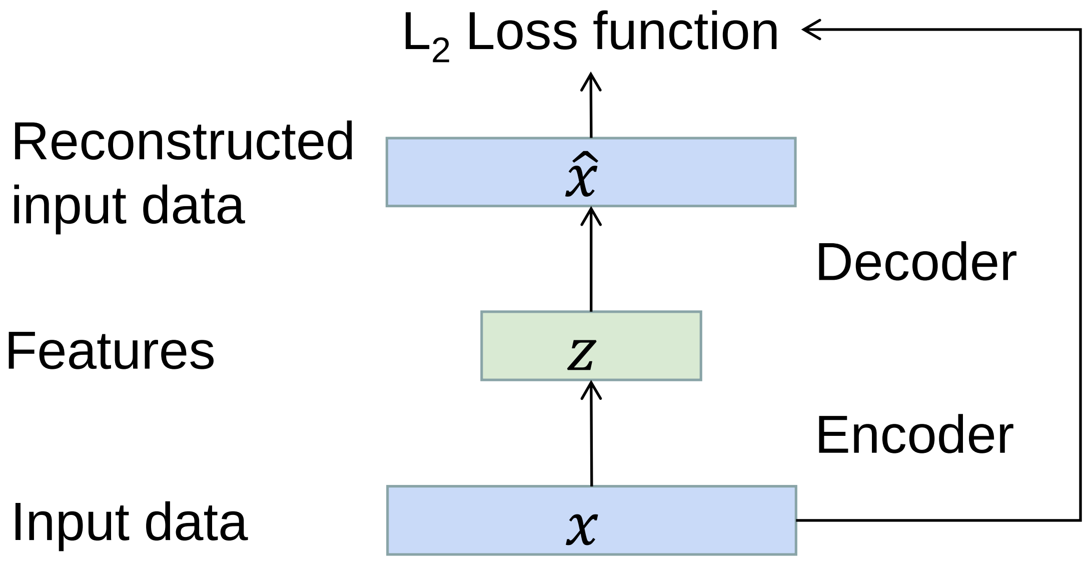


- After training
  - Throw away the decoder part
  - Encoder can be used to initialize a supervised model
  - Fine-tune encoder jointly with supervised model

  - *Idea used in the 90s and early 2000s to pre-train deeper models*

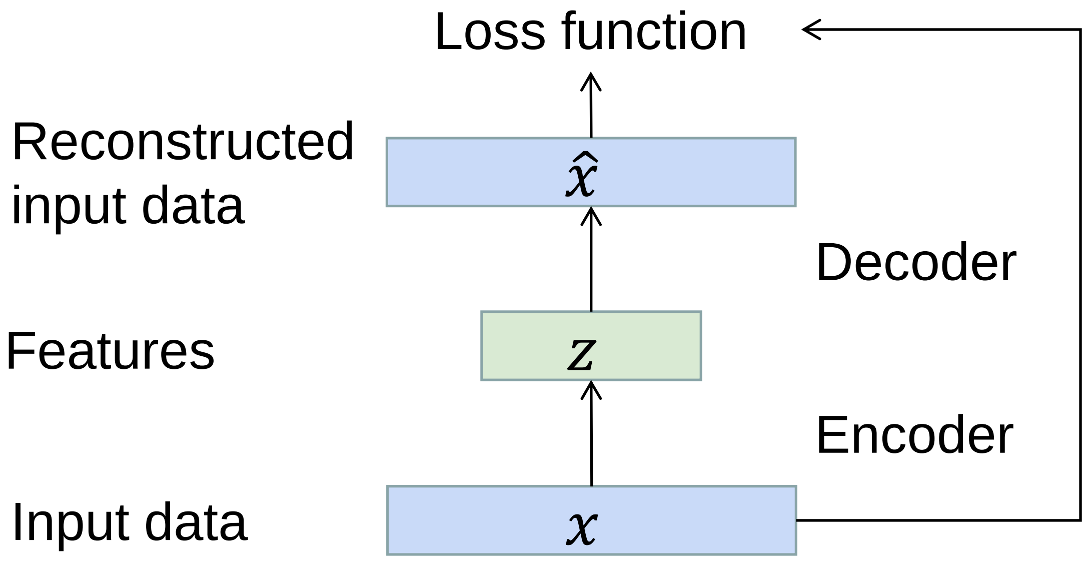
Slide credit: Feifei Li

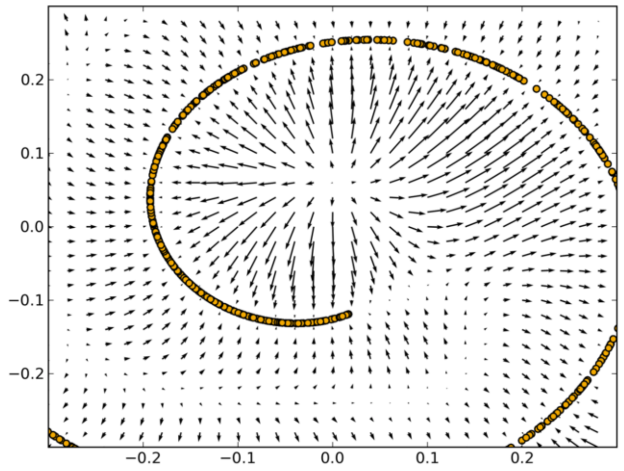
Recap: Variants of Autoencoders



- Regularized Autoencoders
  - Include a regularization term to the loss function: $L\Big(\mathbf{x}, g\big(f(\mathbf{x})\big)\Big) + \Omega(\mathbf{z})$
  - E.g., enforce sparsity by an L_1 regularizer $\Omega(\mathbf{z}) = \lambda \sum_i |z_i|$

Recap: Variants of Autoencoders





- Denoising Autoencoder (DAE)
  - Rather than the reconstruction loss, minimize $L\left(\mathbf{x}, g\left(f\left(\tilde{\mathbf{x}}\right)\right)\right)$ where $\tilde{\mathbf{x}}$ is a copy of $\mathbf{x}$ that has been corrupted by some noise.
  - Denoising forces f and g to implicitly learn the structure of $p_{data}(\mathbf{x})$.

Image source: [Goodfellow 2016]

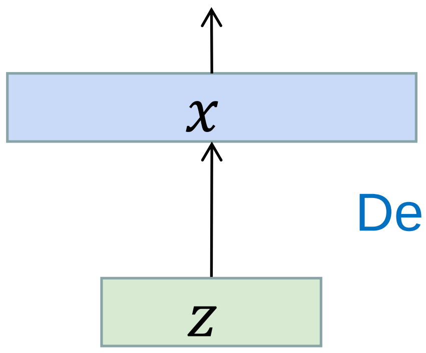
Recap: Probabilistic Spin on Autoencoders

Sample from true conditional

$p_{\theta^*}(\mathbf{x}|\mathbf{z}^{(i)})$

Sample from true prior

$p_{\theta^*}(\mathbf{z})$



Decoder network

- Idea: Sample the model to generate data
  - We want to estimate the true parameters θ^* of this generative model.
- How should we represent the model?
  - Choose prior $p(\mathbf{z})$ to be simple, e.g., Gaussian
  - Conditional $p(\mathbf{x} \mid \mathbf{z})$ is complex (generates image)
    $\Rightarrow$ Represent with neural network
  - Learn model parameters to maximize likelihood of training data

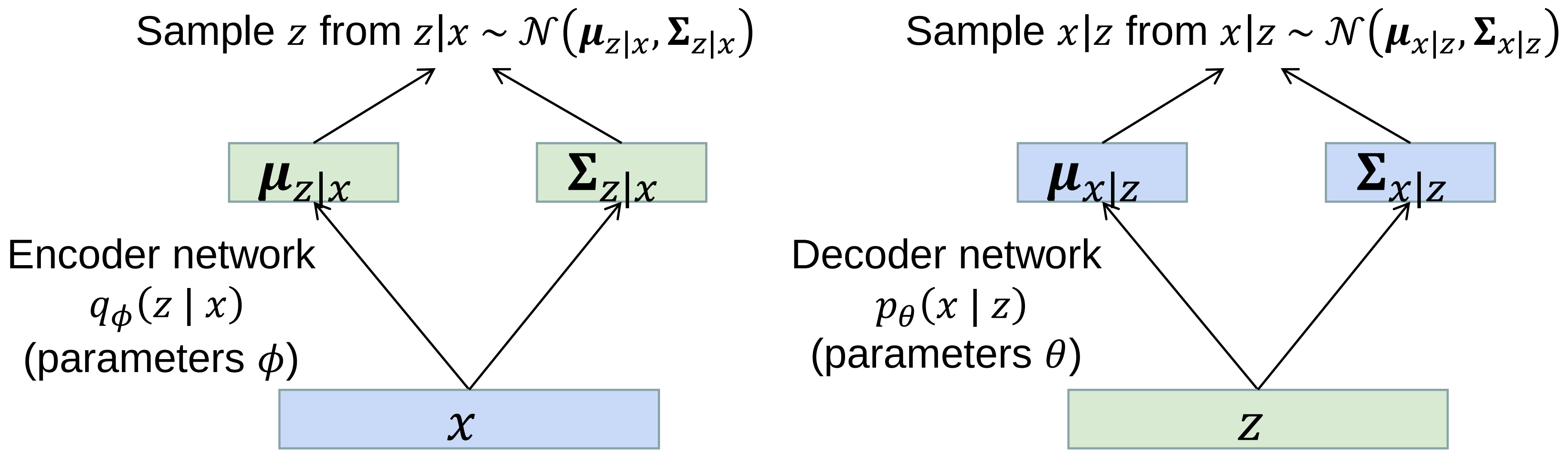
$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z}) d\mathbf{z}$$

Intractable!

Slide adapted from Feifei Li

Recap: Variational Autoencoders

- Define additional encoder network $q_\phi(\mathbf{z} \mid \mathbf{x})$
  - Since we are modelling probabilistic generation of data, encoder and decoder networks are probabilistic



Sample z from $z|x \sim \mathcal{N}(\boldsymbol{\mu}_{z|x}, \boldsymbol{\Sigma}_{z|x})$
$\boldsymbol{\mu}_{z|x}$
$\boldsymbol{\Sigma}_{z|x}$
Encoder network $q_\phi(z \mid x)$ (parameters ϕ)
x
Sample $x|z$ from $x|z \sim \mathcal{N}(\boldsymbol{\mu}_{x|z}, \boldsymbol{\Sigma}_{x|z})$
$\boldsymbol{\mu}_{x|z}$
$\boldsymbol{\Sigma}_{x|z}$
Decoder network $p_\theta(x \mid z)$ (parameters θ)
z

  - Encoder and decoder networks are also called recognition/inference and generation networks

D. Kingma, M. Welling, Auto-Encoding Variational Bayes, ICLR 2014

Slide credit: Feifei Li

Recap: Variational Autoencoders

- We can now work out the log-likelihood

$$\log p_\theta(x^{(i)}) = \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ does not depend on } z)$$

Want to maximize data likelihood

$$= \mathbb{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad \text{(Bayes' Rule)}$$

$$= \mathbb{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad \text{(Multiply by constant)}$$

$$= \mathbb{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbb{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right]$$

$$= \boxed{\underbrace{\mathbb{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}\left(q_\phi(z \mid x^{(i)}) \| p_\theta(z)\right)}_{\mathcal{L}(x^{(i)},\theta,\phi)}} + \underbrace{D_{KL}\left(q_\phi(z \mid x^{(i)}) \| p_\theta(z \mid x^{(i)})\right)}_{\geq 0}$$

Tractable lower bound, which we can take gradient of and optimize

Slide credit: Feifei Li

Recap: Variational Autoencoders

- Variational Lower Bound ("ELBO")

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

$$= \underbrace{\mathbb{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]}_{\text{"Reconstruct the input data"}} - \underbrace{D_{KL}\left(q_\phi(z \mid x^{(i)}) \| p_\theta(z)\right)}_{\text{"Make approximate posterior distribution close to prior"}}$$

- Training: Maximize lower bound

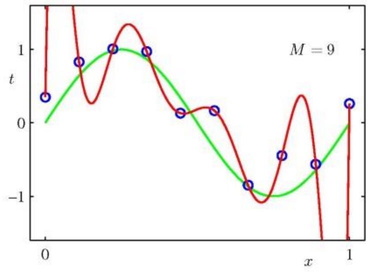
$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

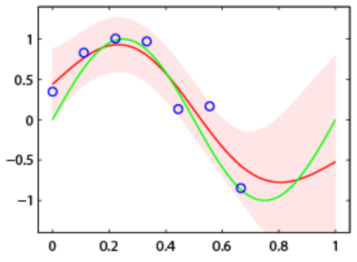
Slide adapted from Feifei Li

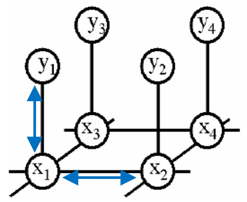
We're Done!

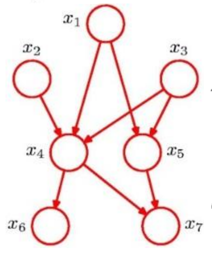
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
  - Latent Variable Models
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

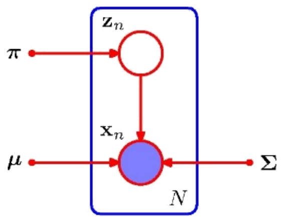
$$f : \mathcal{X} \to \mathbb{R}$$

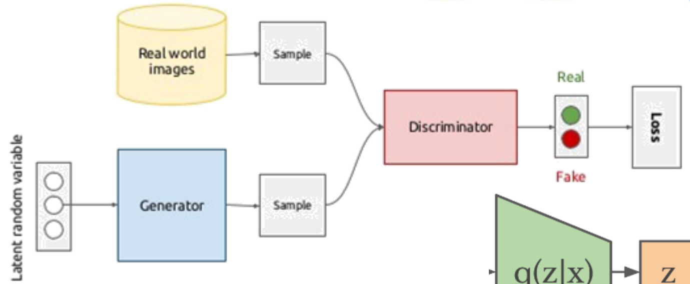


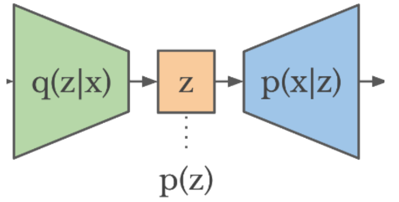












Any More Questions?

Good luck for the exam!