TODO
distribution parts in bishop maybe

Probabilistic Artificial Intelligience

task:

1. reintepret the lecture and add important comments of Krause. Formulas in latex and give a space for detailed proof.
2. one section for reading and maybe essential questions in homework
3. yandex or berkley materials to complement

Introduction and probability

Lecture Notes

Topics Covered

- Probabilistic foundations of AI
- Bayesian learning (GPs, Bayesian deep learning, variational inference, MCMC)
- Bandits & Bayesian optimization
- Planning under uncertainty (MDPs, POMDPs)
- (Deep) Reinforcement learning
- Applications (in class and in project)

Review:Probability

- **Probability space** $(\Omega, \mathcal{F}, \mathcal{P})$
- set of **atomic events** Ω
- set of all **non-atomic events** $\mathcal{F}$
- $\mathcal{F}$is a $\sigma\text{-algebra}$(closed under complements and countable unions)
  - $\Omega \in \mathcal{F}$
  - $A \in \mathcal{F} \to \Omega \backslash A \in \mathcal{F}$
  - $A_1, ..., A_n, ... \in \mathcal{F} \to \bigcup_i A_i \in \mathcal{F}$
- **Probability measure** $\mathcal{P} : \mathcal{F} \to [0,1]$
  - for $A \in \mathcal{F}$, $P(A)$ is the probability that event A happens

Probability Axioms

- Normalization: $P(\Omega) = 1$
- Non-negativity: $P(A) \geq 0 \text{ for all} A \in \mathcal{F}$
- $\sigma\text{-additivity}$:

$$\forall A_1, ..., A_n, ... \in \mathcal{F} \text{ disjoint:} P(\bigcup_{I=1}^{\infty} A_i) = \sum_{I=1}^{\infty} P(A_i)$$

Interpretation of Probabilities

- Frequentist interpretation
  - $P(A)$ is relative frequency of A in repeated experiments
  - Can be difficult to assess with limited data
- Bayesian interpretation
  - $P(A)$ is "degree of belief" A that will occur
  - Where does this belief come from?
  - Many different flavors (subjective, objective, pragmatic, …)

Random Variables

- Let D be some set (e.g., the integers)
- A random variable X is a mapping $X : \Omega \to D$
- For some $x \in D$, we say

$$P(X = x) = P(\omega \in \Omega : X(\omega) = x) \qquad \text{“probability that variable X assumes state x”}$$

Specifying Probability Distributions through RVs

- **Bernoulli** distribution: “(biased) coin flips”$D = \{H, T\}$
  Specify $P(X = H) = p$. Then $P(X = T) = 1 - p$.
  Note: can identify atomic ev.ω with $\{X = H\}$,$\{X = T\}$
- **Binomial** distribution counts no. heads S in n flips
- **Categorical** distribution: “(biased) m-sided dice” $D = \{1, ..., m\}$
  Specify $P(X = i) = p_i$, s.t. $p_i \geq 0, \sum p_i = 1$
- **Multinomial** distribution counts the number of
  outcomes for each side for n throws

Joint Distributions

- random vector $\mathbf{X} = [X_1(\omega), ..., X_n(\omega)]$
- can specify $P(X_1 = x_1, ..., X_n = x_n)$ directly (atomic events are assignments $x_1, ..., x_n$)
- **Joint Distribution** describes relationship among all variables

Conditional Probability

- Formal definition:

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \text{ if } P(b) \neq 0$$

- **Product rule** $P(a \wedge b) = P(a|b)P(b)$
- for distributions: $P(A, B) = P(A|B)P(B)$
  (set of equations, one for each instantiation of A, B)
  $\forall a, b : P(A = b, B = b) = P(A = a|B = b) \cdot P(B = b)$
- **Chain(product) rule** for multiple RVs:$X_1, .., X_n$
  $P(X_1, .., X_n) = P(X_{1:n}) = P(X_1) \cdot P(X_2|X_1) \cdot ... \cdot P(X_n|X_{1:n-1})$

The Two Rules for Joint Distributions

- **Sum rule (Marginalization)**
  $P(X_{1:i-1}, X_{i+1:n}) = \sum_{x_i} P(X_{1:i-1}, X_i = x_i, X_{i+1:n})$
- **Product rule (chain rule)**

Bayes' Rule
Given:

- **Prior** $P(X)$
- **Likelihood** $P(X|Y) = \frac{P(X,Y)}{P(Y)}$

Then:

- **Posterior**
  $P(X|Y) = \frac{P(X)P(Y|X)}{\sum_{X=x} P(X=x)P(Y|X=x)}$

Independent RVs

- Random variables$X_1, ..., X_n$ are called **independent** if
  $P(X_1 = x_1, ..., X_n = x_n) = P(x_1)P(x_2) \ldots P(x_n)$

Conditional Independence

- Rand. vars. X and Y conditionally independent given Z **iff** for all x, y, z:
  $P(X = x, Y = y|Z = z) = P(X = x|Z = z)P(Y = y|Z = z)$
- If $P(Y = y|Z = z) > 0$, that is equivalent to
  $P(X = x|Y = y, Z = z) = P(X = x|Z = z)$
  Similar for sets of random variables$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$
  we write:$\mathbf{X} \perp \mathbf{Y} | \mathbf{Z}$

Problems with High-dim. Distributions

- Suppose we have n binary variables, then we have 2^{n-1} variables to specify
  $P(X_1 = x_1, .., X_n = x_n)$
- Computing marginals:

  - Suppose we have joint distribution $P(X_1, .., X_n)$
  - Then (acc. to sum rule)

$$P(X_i = x_i) = \sum_{x_{1:i-1}, x_{i+1:n}} P(x_1, ..., x_n)$$

  - If all X_i are binary: this sum has 2^{n-1} terms
- Conditional queries
  - Suppose we have joint distribution $P(X_1, .., X_n)$
  - Compute distribution of some variables given values for others: $P(X_1 = \cdot | X_7 = x_7) = \frac{P(X_1=\cdot, X_7=x_7)}{P(X_7=x_7)} = \frac{1}{Z} P(X_1 = \cdot, X_7 = x_7)$
    where, $Z = \sum_{x_1} P(X_1 = x_1, X_7 = x_7)$
    where, $P(X_1 = x_1, X_7 = x_7) = \sum_{x_{2:6}} \sum_{x_{8:n}} P(X_{1:n} = x_{1:n})$, 2^{n-2} terms for binomial X_i
- Representation (parametrization)
- Learning (estimation)
- Inference (prediction)

Gaussian Distribution

- univariate :

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

  σ: Std. dev., μ: mean
- multivaraite:

$$p(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\Sigma|}} exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))$$

  where $\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix}$, $\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$.
- **Multivariate Gaussian distribution**

$$\mathcal{N}(y; \Sigma, \mu) = \frac{1}{((2\pi)^{n/2}\sqrt{|\Sigma|}} exp(-\frac{1}{2}(y-\mu)^T \Sigma^{-1} (y-\mu))$$

where $\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1n} \\ \vdots & & & \vdots \\ \sigma_{n1} & \sigma_{n2} & \dots & \sigma_n^2 \end{pmatrix}$,
$\sigma_{ij} = \mathbb{E}((x_i - \mu_i)(x_j - \mu_j))$,
$\sigma_i^2 = \mathbb{E}((x_i - \mu_i)^2) = Var(x_i)$.
The joint distribution over n variables requires **only $O(n^2)$ parameters**.

- **Fact:Gaussians are independent iff they are uncorrelated:**
  $X_i \perp X_j \Leftrightarrow \sigma_{ij} = 0$
- Multivariate Gaussians have important properties:
  - **Compact representation** of high-dimensional joint distributions
  - **Closed form inference**

Bayesian Inference in Gaussian Distributions

- Suppose we have a Gaussian random vector
  $\mathbf{X} = \mathbf{X}_V = [X_1, ..., X_d] \sim \mathcal{N}(\mu_V, \Sigma_{VV})$
- Hereby $V = \{1, ..., d\}$ is an index set.
- Suppose we consider a subset of the variables
  $A = \{i_1, ..., i_k\}, \quad i_j \in V$
- The **marginal distribution** of variables indexed by A is:
  $\mathbf{X}_A = [X_{i_1}, ..., X_{i_k}] \sim \mathcal{N}(\mu_A, \Sigma_{AA})$
  where $\mu_A = [\mu_{i_1}, ..., \mu_{i_k}]$, $\Sigma_{AA} = \begin{pmatrix} \sigma_{i_1 i_1} & \dots & \sigma_{i_1 i_k} \\ \vdots & \ddots & \vdots \\ \sigma_{i_k i_1} & \dots & \sigma_{i_k i_k} \end{pmatrix}$

Conditional Distributions

- Suppose we have a Gaussian random vector
  $\mathbf{X} = \mathbf{X}_V = [X_1, ..., X_d] \sim \mathcal{N}(\mu_V, \Sigma_{VV})$
- Further, suppose we take two disjoint subsets of V
  $A = \{i_1, ..., i_k\} \quad B = \{j_1, ..., j_m\}$
- The **conditional distribution**
  $p(\mathbf{X}_A | \mathbf{X}_B = \mathbf{x}_B) = \mathcal{N}(\mu_{A|B}, \Sigma_{A|B})$
  is Gaussian, **where**

$$\mu_{A|B} = \mu_A + \Sigma_{AB} \Sigma_{BB}^{-1} (\mathbf{x}_B - \mu_B)$$

$$\Sigma_{A|B} = \Sigma_{AA} - \Sigma_{AB} \Sigma_{BB}^{-1} \Sigma_{BA}$$

where $\Sigma_{AB} = \begin{pmatrix} \sigma_{i_1 j_1} & \dots & \sigma_{i_1 j_m} \\ \vdots & \ddots & \vdots \\ \sigma_{i_k j_1} & \dots & \sigma_{i_k j_m} \end{pmatrix} \in \mathbb{R}^{k \times m}$

Multiples of Gaussians are Gaussian

- Suppose we have a Gaussian random vector $\mathbf{X} = \mathbf{X}_V = [X_1, ..., X_d] \sim \mathcal{N}(\mu_V, \Sigma_{VV})$
- Take a matrix $M \in \mathbb{R}^{m \times d}$
- Then the random vector $\mathbf{Y} = \mathbf{MX}$ is Gaussian:

$$\mathbf{Y} \sim \mathcal{N}(\mathbf{M}_{\mu_V}, \mathbf{M}\Sigma_{VV}\mathbf{M}^T$$

Sums of Gaussians are Gaussian

- Suppose we have independent two Gaussian random vectors $\mathbf{X} = \mathbf{X}_V = [X_1, ..., X_d] \sim \mathcal{N}(\mu_V, \Sigma_{VV})$ $\mathbf{X}' = \mathbf{X}'_V = [X'_1, ..., X'_d] \sim \mathcal{N}(\mu'_V, \Sigma'_{VV})$

Bayesian Learning

Lecture Notes

Recall: linear regression

- $y \approx \mathbf{w}^T \mathbf{x} = f(\mathbf{x})$

Recall: ridge regression

- Regularized optimization problem: $\min_{\mathbf{w}} \sum_i (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_2^2$
- Can optimize using (stochastic) gradient descent, or still find **analytical solution**: $\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

Ridge regression as Bayesian inference
Assume $p(\mathbf{w}) = \mathcal{N}(0, \sigma_p^2 \cdot \mathbf{I}))$ independent of $\mathbf{x}_{1:n}$
conditional iid. $\Rightarrow p(y_{1:n}|\mathbf{w}, \mathbf{x}_{1:n}) = \prod_{i=1}^n p(y_i|\mathbf{w}, \mathbf{x}_i)$
In particular: $p(y_i|\mathbf{w}, \mathbf{x}_i) = \mathcal{N}(y_i; \mathbf{w}^T\mathbf{x}_i, \sigma_n^2) \Leftrightarrow y_i = \mathbf{w}^T\mathbf{x}_i + \varepsilon_i \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_n^2)$

Then,

$$
\begin{aligned}
p(\mathbf{w}|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) &= \frac{1}{Z} p(\mathbf{w}|\mathbf{x}_{1:n}) p(\mathbf{y}_{1:n}|\mathbf{w}, \mathbf{x}_{1:n}) \\
&= \frac{1}{Z} p(\mathbf{w}) \prod_{i=1}^{n} p(y_i|\mathbf{w}, \mathbf{x}_i) \\
&= \frac{1}{Z Z_p} exp(-\frac{1}{2\sigma_p^2} \|\mathbf{w}\|_2^2) \frac{1}{Z_l} \prod exp(-\frac{1}{\sigma_n^2} (\mathbf{y}_i - \mathbf{w}^T \mathbf{x}_i)^2) \\
&= \frac{1}{Z'} exp(-\frac{1}{2\sigma_p^2} \|\mathbf{w}\|_2^2 - \frac{1}{\sigma_n^2} (\mathbf{y}_i - \mathbf{w}^T \mathbf{x}_i)^2)
\end{aligned}
$$

$\Rightarrow \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{x}_{1:n}, y_{1:n}) = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_2^2$
$\to \lambda = \frac{\sigma_n^2}{\sigma_p^2}$

Ridge regression = MAP estimation

- Ridge regression can be understood as finding the **Maximum A Posteriori (MAP) parameter estimate** for a linear regression problem, assuming that
- The **noise** $P(y|\mathbf{x}, \mathbf{w})$ is **(cond.) iid Gaussian** and
- The **prior** $P(\mathbf{w})$ on the model parameters $\mathbf{w}$ is **Gaussian**
- However, ridge regression returns a single model
- Such a **point estimate** does not quantify **uncertainty**

Bayesian Linear Regression (BLR)

- Key idea: Reason about full posterior of $\mathbf{w}$, not only its mode
- For Bayesian linear regression with Gaussian prior and Gaussian likelihood, posterior has **closed form**

Posterior distributions in BLR

- Prior: $p(\mathbf{w} = \mathcal{N}(0, \mathbf{I})$
- Likelihood: $p(y|\mathbf{x}, \mathbf{w}, \sigma_n) = \mathcal{N}(y; \mathbf{w}^T \mathbf{x}, \sigma_n^2)$
- Posterior:
  $p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \mathcal{N}(\mathbf{w}; \bar{\mu}, \bar{\Sigma})$
  $\bar{\mu} = (\mathbf{X}^T \mathbf{X} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$
  $\bar{\Sigma} = (\sigma_n^{-2} \mathbf{X}^T \mathbf{X} + \mathbf{I})^{-1}$
- $\bar{\mu}$ is ridge regression solution!
- Precision matrix: $\bar{\Lambda} = \bar{\Sigma}^{-1} = \sigma_n^{-2} \mathbf{X}^T \mathbf{X} + \mathbf{I}$

Making predictions in BLR

- Define $f^* = \mathbf{w}^T \mathbf{x}^*$
  $\to p(f^*|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \mathbf{x}^*) = \int p(f^*|\mathbf{w}, \mathbf{y}_{1:n}, \mathbf{x}^*) p(\mathbf{w}|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \mathbf{x}^*) d\mathbf{w}$

since $\mathbf{w} \sim \mathcal{N}(\bar{\mu}, \bar{\Sigma}), \mathbf{y}^* = f^* + \varepsilon, \varepsilon \sim \mathcal{N}(0, \sigma_n^2)$
$p(f^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) = \mathcal{N}(\bar{\mu}^T \mathbf{x}^*, \mathbf{x}^{*T} \bar{\Sigma} \mathbf{x}^*)$
$p(y^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) = \mathcal{N}(\bar{\mu}^T \mathbf{x}^*, \mathbf{x}^{*T} \bar{\Sigma} \mathbf{x}^* + \sigma_n^2)$

Aleatoric vs. epistemic uncertainty

- Uncertainty about f^* : $\bar{\Sigma} \leftarrow (\text{epistemic})$
- Noise/Uncertainty about y^* given f^* : $\sigma_n^2 \leftarrow (\text{aleatoric})$
- Can distinguish two forms of uncertainty:
    - **Epistemic uncertainty**: Uncertainty about the model due to the lack of data
    - **Aleatoric uncertainty**: Irreducible noise

Bayesian Linear Regression(cont'd)

Lecture Notes

- Observations: Conditional Linear Gaussians
- If X, Y are jointly Gaussian, then $p(X|Y = y)$ is Gaussian, with mean linearly dependent on y:
  $p(X = x|Y = y) = \mathcal{N}(x; \mu_{X|Y} \sigma_{X|Y}^2)$
  $\mu_{X|Y} = \mu_X + \sigma_{XY} \sigma_Y^2 (y - \mu_Y)$
- Thus random variable X can be viewed as a linear function of Y with independent Gaussian noise added
  $X = a \cdot Y + b + \varepsilon$, where $a = \sigma_{XY} \sigma_Y^2, b = \mu_X - \sigma_{XY} \sigma_Y^2 \mu_Y$
- The converse also holds.

Ridge regression vs Bayesian lin. regression

- Ridge regression: predict using *MAP estimate* for weights
  $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{x}_{1:n}, y_{1:n})$
  $p(y^*|\mathbf{x}^*, \hat{\mathbf{w}}) = \mathcal{N}(y^*; \hat{\mathbf{w}}^T \mathbf{x}^*, \sigma_n^2)$
- BLR: predict by averaging all $\mathbf{w}$ acc. to posterior:
  $p(y^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) = \int p(y^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\mathbf{w} = \mathcal{N}(\bar{\mu}^T \mathbf{x}^*, \mathbf{x}^{*T} \bar{\Sigma} \mathbf{x}^* + \sigma_n^2)$
- Thus, ridge regression can be viewed as approximating the full posterior by **(placing all mass on) its mode**
  $p(y^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) = \int p(y^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\mathbf{w}$
  $\approx \int p(y^*|\mathbf{x}^*, \mathbf{w}) \delta_{\hat{\mathbf{w}}}(\mathbf{w}) d\mathbf{w}$
  $= p(y^*|\mathbf{x}^*, \hat{\mathbf{w}})$
  $\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{x}_{1:n}, y_{1:n})$
- *Note*: $\delta_{\hat{\mathbf{w}}}(\cdot)$ is such that $\int f(\mathbf{w}) \delta_{\hat{\mathbf{w}}}(\mathbf{w}) d\mathbf{w} = f(\hat{\mathbf{w}})$

Choosing hyperparameters

- In BLR, need to specify the (co-)variance of the prior σ_p and the variance of the noise σ_n
- These are **hyperparameters** of the model (governing the distribution of the parameters $\mathbf{w}$)
- How to choose? One option:
  - Choose $\hat{\lambda} = \frac{\hat{\sigma}_n^2}{\hat{\sigma}_p^2}$ via cross-validation
  - Then estimate $\hat{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^n (y_i - \hat{\mathbf{w}}^T \mathbf{x}_i)^2$ as the empirical variance of the residual, and solve for $\hat{\sigma}_p^2 = \frac{\hat{\sigma}_n^2}{\hat{\lambda}}$
- Another option: marginal likelihood of the data, see **Gaussian Process (marginal likelihood)**

Side note: Graphical models

- Have seen: Can represent arbitrary joint distributions as product of conditionals via chain rule
- Often, factors only depend on subsets of variables
- Can represent the resulting product as a directed acyclic graph
- Graphical model for BLR (see lecture notes)

Recursive Bayesian updates

- “Today’s posterior is tomorrow’s prior”
- Surpose that:
  Prior: $p(\theta)$, observe $y_{1:n}$, s.t. $p(y_{1:n}|\theta) = \prod_{i=1}^n p_i(y_i|\theta)$
  for BLR: $\theta \equiv \mathbf{w}$, $p_i(y_i|\theta) \equiv p(y_i|\mathbf{w}, \mathbf{x}_i)$
  Define $p^{(j)}(\theta)$ to be the posterior afer recurring the first j observation. $p^{(j)}(\theta) = p(\theta|y_{1:j})$
- $p^{(0)}(\theta) = p(\theta) = \mathcal{N}(0, \sigma_p \cdot \mathbf{I})$
  Surpose we have cumputed $p^{(j)}(\theta) \equiv \mathcal{N}(\mu^{(j)}, \Sigma^{(j)}) \leftarrow$ posterior $\theta^{(j)} = \{\mu^{(j)}, \Sigma^{(j)}\}$
  and observed y_j.
- $p^{(j+1)}(\theta) = p(\theta|y_{1:j+1}) = \frac{1}{Z}p(\theta|y_{1:j})p(y_{j+1}|\theta, y_{1:j}) = \mathcal{N}(\mu^{(j+1)}, \Sigma^{(j+1)})$
  where, $\theta^{(j+1)} = \{\mu^{(j+1)}, \Sigma^{(j+1)}\}, \quad p(\theta|y_{1:j}) = p^{(j)}(\theta), \quad p(y_{j+1}|\theta, y_{1:j}) = p_{j+1}(y_{j+1}|\theta)$

Summary Bayesian Linear Regression

- **Bayesian linear regression** makes same modeling assumptions as ridge regression (Gaussian prior on weights, Gaussian noise)
- BLR computes / uses **full posterior distribution** over the weights rather than the mode only
- Thus, it captures **uncertainty in weights**, and allows to separate epistemic from aleatoric uncertainty
- Due to independence of the noise, can do **recursive updates** on the weights

Kalman Filters

Lecture Notes

Kalman filters

- Track objects over time using noisy observations
  - E.g., robots moving, industrial processes,...
- State described using **Gaussian variables**
  - E.g., location, velocity, acceleration in 3D
- Assume conditional linear Gaussian dependencies for states and observations

Kalman Filters: The Model

- $X_1, ..., X_T$: Location of object being tracked
- $Y_1, ..., Y_T$: Observations
- $P(X_1)$: **Prior** belief about location at time 1 (Gaussian)
- $P(X_{t+1}|X_t)$: **Motion Model**
  - How do I expect my target to move in the environment?
    $\mathbf{X}_{t+1} = \mathbf{F}\mathbf{X}_t + \varepsilon_t$, where $\varepsilon_t \in \mathcal{N}(0, \Sigma_x)$
- $P(Y_t|X_t)$: **Sensor model**
  - What do I observe if target is at location X_t?
    $\mathbf{Y}_t = \mathbf{H}\mathbf{X}_t + \eta_t$, where $\eta_t \in \mathcal{N}(0, \Sigma_y)$
- Assumptions:
  Known: $\mathbf{X}_{t+1} = \mathbf{F}\mathbf{X}_t + \varepsilon_t, \mathbf{Y}_t = \mathbf{H}\mathbf{X}_t + \eta_t, \qquad \varepsilon_{1:t}, \eta_{1:t}$ independent
  implies that: $X_{t+1} \perp X_{1:t-1}|X_t$, and $Y_{t+1} \perp Y_{1:t-1}, X_{1:t-1}|X_t$
  $\rightarrow P(X_{1:t}, Y_{1:t}) = P(X_1)P(X_2|X_1)\dots P(X_n|X_{n-1})P(Y_1|X_1)P(Y_2|X_2)\dots P(Y_n|X_n)$
  $= P(X_1)P(Y_1|X_1)\prod_{i=2}^{t} P(X_i|X_{i-1})P(Y_i|X_i)$

Bayesian filtering

- Start with $P(X_1) = \mathcal{N}(\mu, \Sigma)$
- At time t
  - Assume we have $P(X_t|Y_{1,\dots,t-1})$
  - **Conditioning**: $P(X_t|Y_{1,\dots t}) = \frac{1}{Z}P(X_t|Y_{1:t-1})P(Y_t|X_t, Y_{1:t-1})$, where
    $P(Y_t|X_t, Y_{1:t-1}) = P(Y_t|X_t)$, so that $Z = \int P(X_t|Y_{1:t-1})P(Y_t|X_t)dX_t$
  - **Prediction**: $P(X_{t+1}|Y_{1,\dots t}) = \int P(X_{t+1}, X_t|Y_{1:t})dX_t =$
    $\int P(X_{t+1}|X_t, Y_{1:t})P(X_t|Y_{1:t})dX_t = \int P(X_{t+1}|X_t)P(X_t|Y_{1:t})dX_t$
  - For Gaussians, can compute these integrals in closed form!
- Example: Random walk in 1D
  - Transition / motion model: $P(x_{t+1}|x_t) = \mathcal{N}(x_t, \sigma_x^2)$
    $x_{t+1} = x_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma_x^2)$

    - Sensor model: $P(y_t|x_t) = \mathcal{N}(x_t, \sigma_y^2)$
      $y_t = x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma_y^2)$
    - State at time t: $P(x_t|y_{1:t}) = \mathcal{N}(\mu_t, \sigma_t^2)$
    - $\rightarrow \mu_{t+1} = \frac{\sigma_y^2 \mu_t + (\sigma_t^2+\sigma_x^2) y_{t+1}}{\sigma_t^2+\sigma_x^2+\sigma_y^2} \quad \sigma_{t+1} = \frac{(\sigma_t^2+\sigma_x^2)\sigma_y^2}{\sigma_t^2+\sigma_x^2+\sigma_y^2}$

General Kalman update

- Transition model: $P(\mathbf{x}_{t+1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t+1}; \mathbf{F}\mathbf{x}_t, \Sigma_x)$
- Sensor model: $P(\mathbf{y}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{y}_t; \mathbf{H}\mathbf{x}_t, \Sigma_y)$
- **Kalman Update**:
  $\mu_{t+1} = \mathbf{F}\mu_t + \mathbf{K}_{t+1}(\mathbf{y}_{t+1} - \mathbf{H}\mathbf{F}\mu_t)$
  $\mathbf{\Sigma}_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_\mathbf{x})$
- **Kalman Gain**:

$$\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_\mathbf{x})\mathbf{H}^T(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^T + \mathbf{\Sigma}_\mathbf{x})\mathbf{H}^T + \mathbf{\Sigma}_y)^{-1}$$

- Can compute $\mathbf{\Sigma}_t$ and $\mathbf{H}_t$ **offline**

BLR vs Kalman Filtering

- Can view Bayesian linear regression as a form of a Kalman filter!
    - Hidden variables are the weights
    - Forward model is constant (identity)
    - Observation model at time t is determined by data point x_t

Gaussian Process

Lecture Notes

What about nonlinear functions?

- Recall: Can apply linear method (like BLR) on nonlinearly transformed data. However, computational cost increases with dimensionality of the feature space!
  $f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x})$
  In d-dim,: $\mathbf{x} = [x_1, ..., x_d]$, $\Phi(\mathbf{x}) = [1, x_1, ..., x_d, x_1^2, ..., x_d^2, x_1x_2, ..., x_{d-1}x_d, ..., x_1 \cdot ... \cdot x_m, ..., x_{d-m+1} \cdot ... \cdot x_d] \leftarrow O(d^m)$ monomials of deg m

The "Kernel Trick"

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels
- $\mathbf{x}_i^T\mathbf{x}_j \Rightarrow k(\mathbf{x}_i, \mathbf{x}_j)$

- $\Phi(\mathbf{x}) = [\text{all monomials of deg } \leq m]$
  $\Rightarrow k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + 1)^m \quad \text{implicitly represents all monimials o f degree up to } m$

Weight vs Function Space View

- Assume **Gaussian prior** on the weights: $\mathbf{w} \in \mathbb{R}^d \sim \mathcal{N}(0, \sigma_p^2\mathbf{I})$
- This imply **Gaussian distribution on the predictions**
- Suppose we consider an arbitrary (finite) set of inputs $\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} \in \mathbb{R}^{n\times d}$
- The predictive distribution is given by:
  - $f \sim \mathcal{N}(0, \sigma_p^2\mathbf{X}\mathbf{X}^T) \leftarrow \text{let } \mathbf{K}_{ij} = \mathbf{x}_i^T\mathbf{x}_j, \mathbf{K} \in (R)^{n\times n}$
  - where $f = [f_1, ..., f_n], f_i = \mathbf{x}_i^T\mathbf{w} \rightarrow f = \mathbf{X}\mathbf{w}$

Predictions in "function space"

- Suppose we're given data $\mathbf{X}$, $\mathbf{y}$, and want to predict $\mathbf{x}^*$
  - $\widetilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X} \\ \mathbf{x}^* \end{pmatrix}, \quad \tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ y^* \end{pmatrix}, \quad \tilde{\mathbf{f}} = \begin{pmatrix} \mathbf{f} \\ f^* \end{pmatrix}$
    $\rightarrow \tilde{\mathbf{f}} = \widetilde{\mathbf{X}} \cdot \mathbf{w}, \tilde{\mathbf{y}} = \tilde{\mathbf{f}} + \tilde{\varepsilon}, \tilde{\varepsilon} \sim \mathcal{N}(0, \sigma_n^2\mathbf{I}_{n+1})$
- $\rightarrow \tilde{\mathbf{y}} \sim \mathcal{N}(0, \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^T + \sigma_n^2\mathbf{I})$,where $\widetilde{\mathbf{K}} = \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^T$
- $\rightarrow P(y^*|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) = \mathcal{N}(\mu_{\mathbf{x}^*|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}}, \sigma^2_{\mathbf{x}^*|\mathbf{x}_{1:n}})$

Key Insight

- For prior $\mathbf{w} \sim \mathcal{N}(0, \mathbf{I})$, the predictive distribution over $\mathbf{f} = \mathbf{X}\mathbf{w}$ is Gaussian
  $\mathbf{f} \sim \mathcal{N}(0, \mathbf{X}\mathbf{X}^T) \equiv \mathcal{N}(0, \mathbf{K})$
- Thus, data points only enter as inner products!
- Can kernelize: $\mathbf{f} \sim \mathcal{N}(0, \mathbf{K})$, where $\mathbf{K}_{\mathbf{x},\mathbf{x}'} = \phi(\mathbf{x})^T\phi(\mathbf{x}') = \mathbf{k}(\mathbf{x}, \mathbf{x}')$
  e.g. poly. kernel $(1 + \mathbf{x}^T\mathbf{x}')^m$

What about infinite domains?

- The previous construction can be generalized to **infinitely large domains $\mathbf{X}$**
- The resulting random function is called a **Gaussian process**

Bayesian learning with Gaussian processes

- c.f. Rasmussen & Williams 2006
- $Likelihood : P(data|f) \qquad Posterior : P(f|data)$
- Predictive uncertainty + tractable inference

Gaussian Processes

- ∞-dimension Gaussian
- Gaussian process (GP) = normal distribution over functions
- Finite marginals are multivariate Gaussians
- Closed form formulae for Bayesian posterior update exist
- Parameterized by covariance function $k(\mathbf{x}, \mathbf{x}') = Cov(f(\mathbf{x}), f(\mathbf{x}'))$
- A **Gaussian Process (GP)** is an:
  - (infinite) set of random variables, indexed by some set $\mathbf{X}$
    i.e., there exists functions $\mu : X \to \mathbb{R} \quad k : X \times X \to \mathbb{R}$
    such that for all $A \subseteq X, \quad A = \{x_1, ..., x_m\}$
    it holds that $Y_A = [Y_{x_1}, ..., Y_{x_m}] \sim \mathcal{N}(\mu_A, \mathbf{K}_{AA})$
    where,
    $$\mathbf{K}_{AA} = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_m) \\ \vdots & & \vdots & \\ k(x_m, x_1) & k(x_m, x_2) & \dots & k(x_m, x_m) \end{pmatrix}, \quad \mu_A = \begin{pmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{pmatrix}$$
    k is called **covariance (kernel)** function
    μ is called **mean** function
- **GP Marginals**
  Typically, primarily interested in marginals, i.e.,
  $p(f(x)) = \mathcal{N}(f(x); \mu(x), k(x, x))$
  $k(x_1, x_2) = Cov(f(x_1), f(x_2)) = \mathbb{E}[((f(x_1) - \mu(x_1))((f(x_2) - \mu(x_2))]$
  $k(x, x) = Cov(f(x), f(x)) = \mathbb{E}[((f(x) - \mu(x))^2] = Var(f(x))$

Covariance (kernel) Functions

- k must be **symmetric**
  - $k(x, x') = k(x', x) \text{ for all } x, x'$
- k must be **positive definite**
  - For all A: K_{AA} is positive definite matrix
  - $\forall x \in \mathbb{R}^{|A|} : x^T K_{AA} x \geq 0 \Leftrightarrow \text{all eigenvalues of } K_{AA} \geq 0$
- Kernel function k: assumptions about correlation!

Covariance Functions: Examples

- Linear kernel:$k(x, x') = x^T x'$
  - GP with linear kernel = Bayesian linear regression
  - Linear kernel with features:
    $k(x, x') = \phi(x)^T \phi(x')$
- Squared exponential (a.k.a. RBF, Gaussian) kernel
  - $k(x, x') = exp(-\|x - x'\|_2^2 / h^2)$, h is called bandwidth
- Exponential kernel
  - $k(x, x') = exp(-\|x - x'\|_2 / h)$

Smoothness of GP Samples

- Covariance function determines smoothness of sample paths*
  assuming $\mu(x) = 0, \forall x$
  - Squared exponential kernel: **analytic** (**infinitely** diff'able)
  - Exponential kernel: continuous, but **nowhere** diff'able
  - Matérn kernel with parameter ν: $\lceil \nu \rceil$ times (m.s.) diff'able
    $k(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)}(\frac{\sqrt{2\nu}\|\mathbf{x}-\mathbf{x}'\|_2}{\rho})^{\nu} K_{\nu}(\frac{\sqrt{2\nu}\|\mathbf{x}-\mathbf{x}'\|_2}{\rho})$
    Hereby Γ is the Gamma function, K_ν the modified Bessel function of the second kind, and ρ is a bandwidth parameter.
  - Special cases: $\nu = \frac{1}{2}$ gives exponential kernel; $\nu \rightarrow \infty$ gives Gaussian kernel.

Composition Rules

- Suppose we have two covariance functions.
  $k_1 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \quad k_2 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ definedon data space $\mathcal{X}$
- Then the following functions are valid cov. functions:
  $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
  $\rightarrow f_1 \sim GP(\mu_1, k_1), f_2 \sim GP(\mu_2, k_2), g = f_1 + f_2 \sim GP(\mu_1 + \mu_2, k_1 + k_2)$
  $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$
  $k(\mathbf{x}, \mathbf{x}') = c\, k_1(\mathbf{x}, \mathbf{x}') \quad \text{for } c > 0$
  $k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}'))$, where f is a polynomial with positive coefficients or the exponential function

Forms of Covariance Functions

- Covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is called:
  - **Stationary** if $k(x, x') = k(x - x')$
  - **Isotropic** if $k(x, x') = k(\|x - x'\|_2)$

	$Stationary?$	$Isotropic?$
$Linear$	$\times$	$\times$
$Gaussian$	✓	✓
$exp(-\frac{(x-x')^T\mathbf{M}(x-x')}{h^2})$ $\mathbf{M}\, pos.semi-def.$	✓	$\times$

Making Predictions with GPs

- Suppose $p(f) = GP(f; \mu; k)$
  and we observe $y_i) = f(\mathbf{x}_i + \varepsilon_i) \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2) \quad A = \{\mathbf{x}_1, ..., \mathbf{x}_m\}$
- Then $p(f|\mathbf{x}_1, ..., \mathbf{x}_m, y_1, ..., y_m) = GP(f; \mu', k)'$
  where
  $\mu'(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}_{x,A}(\mathbf{K}_{AA} + \sigma^2\mathbf{I})^{-1}(\mathbf{y}_A - \mu_A)$

$k'(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') - \mathbf{k}_{x,A}(\mathbf{K}_{AA} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_{x',A}$
Note: $\mathbf{k}_{x,A} = [k(x,x_1), ...k(x,x_m)]$

- $\rightarrow$ Closed form formulas for prediction!
- $\rightarrow$ Posterior covariance k' does not depend on $\mathbf{y}_A$

Common Convention: Prior Mean 0

Surpose $f \sim GP(\mu, k)$
Define $g := g(x) = f(x) - \mu(x) \quad \forall x$
$\Rightarrow g \sim GP(0,k)$
$\Rightarrow f(x) = g(x) + \mu(x)$

How to sample from a GP?

- Forward sampling
  $P(f_1, ..., f_n) = P(f_1)P(f_2|f_1)...P(f_n|f_{1:n-1})$
  where $P(f_1) \sim \mathcal{N}(\mu_1, \sigma_1^2), ..., P(f_n|f_{1:n-1}) \sim \mathcal{N}(\mu_{n|1:n-1}, \sigma^2_{n|1:n-1})$
  Can sample $f_1 \sim P(f_1)$,Then $f_2 \sim P(f_2|f_1)...f_n \sim P(f_n|f_{1:n-1})$

Side Note: Kalman Filters are GPs

- **Kalman filters** can be seen as **a special case of a GP** with a particular conditional independence structure that allows efficient / recursive Bayesian filtering
- $\{x_1, x_2...y_1, y_2, ...\}$ is a GP, $x_1 \sim \mathcal{N}(0, \sigma_p^2)$
  $x_{t+1} = x_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma_x^2)$
  $y_t = x_t + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma_y^2)$
  Note:
  $\sigma_1^2 = \sigma_p^2, \quad \sigma_2^2 = \sigma_p^2 + \sigma_x^2, \quad \sigma_t^2 = \sigma_p^2 + (t-1)\sigma_x^2$
  $\mu_{t+1} = \mathbb{E}[x_{t+1}] = \mathbb{E}[x_t + \varepsilon_t] = \mu_t + \mathbb{E}[\varepsilon_t] = \mu_t = \mu_1 = 0$
  $Cov(x_t, x_{t+\Delta}) = \mathbb{E}[(x_t - \mu_t)(x_{t+\Delta} - \mu_{t+\Delta})] = ... = Var(x_t^2) = \sigma_t^2$

Optimizing Kernel Parameters

- How should we pick the hyperparameters?
- One answer: crossvalidation on predictive performance.
- The Bayesian perspective provides an alternative approach:
  Maximize the marginal likelihood of the data
- $\hat{\theta} = argmax_\theta p(y|x,\theta) = argmax_\theta \int p(y, f|x, \theta)df$
  $= argmax_\theta \int p(y|f,x)p(f|\theta)df = argmax_\theta \mathcal{N}(y; 0, \mathbf{K}_y(\theta))$ $\leftarrow$zero mean by convention
  $= argmin_\theta \frac{d}{2}log2\pi + \frac{1}{2}log|\mathbf{K}_y(\theta)| + \frac{1}{2}y^T\mathbf{K}_y(\theta)y$
  Note: $\theta = [\theta', \sigma_n^2], \quad \mathbf{K}_y(\theta) = \mathbf{K}_x(\theta') + \sigma_n^2\mathbf{I}$
  $\mathcal{N}(y; 0, \mathbf{K}_y(\theta)) = \frac{1}{\sqrt{(2\pi)^d|\mathbf{K}_y(\theta)|}}exp(-\frac{1}{2}y^T\mathbf{K}_y(\theta)y)$

Model Selection for GPs

- Marginal likelihood of the data
  $logp(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}log|\mathbf{K}_y| - \frac{n}{2}log2\pi$ the last term is indep. of θ
- Can find $\hat{\theta} = argmax\ p(\mathbf{y}|X,\theta)$ by gradient descent
  $\hat{\theta} = argmin_\theta\ \frac{1}{2}\mathbf{y}^T\mathbf{K}_y^{-1}\mathbf{y} + \frac{1}{2}log|\mathbf{K}_y| = argmin_\theta\ \mathcal{L}(\theta)$

Optimizing the Likelihood

- Gradient of the Likelihood
  $\frac{\partial}{\partial\theta_j}log\,p(\mathbf{y}|X,\theta) = \frac{1}{2}\mathbf{tr}((\alpha\alpha^T - \mathbf{K}^{-1})\frac{\partial\mathbf{K}}{\partial\theta_j}$, where $\alpha = \mathbf{K}^{-1}\mathbf{y}$
- probably converge to local optima

Bayesian Model Melection

- $p(y|X,\theta) = \int p(y|f,X)p(f|\theta)df$
-

	$p(y\|f,X)$	$p(f\|\theta)$
underfit(too simple)	small for most f	large
overfit(too complex)	large for few f,small for most f	small
just right	moderate	moderate

- In contrast, MAP estimation approx. $p(y|X,\theta) \approx p(y|\hat{f},\theta)$
  where $\hat{f} = argmax\ p(y|f,X)p(f|\theta)$
- **Maximizing marginal likelihood** is an example of an **Empirical Bayes method** – estimating a prior distribution from data
- Integrating (rather than optimizing) over the unknown function **helps guarding against overfitting**
- Other possibilities exist:
  - Can place **hyperprior** on parameters of the prior and obtain MAP estimate (corresponds to a regularization term)
  - Can integrate out the hyperprior (but also has params...)
    Instead of $\hat{\theta} = argmax_\theta\ p(y|X,\theta)$, can place hyperprior $p(\theta)$ on θ
    $\rightarrow \hat{\theta} = argmax_\theta\ p(\theta|X,y)$
    $= argmax_\theta\ p(\theta)p(y|X,\theta) = argmin_\theta\ -logp(y|X,\theta) - logp(\theta)$
  - Or go **fully bayesian**
    $p(y^*|x^*,X,y) = \int p(y^*|x^*,f)p(f|X,y,\theta)p(\theta)dfd\theta$

Computational Issues

- Computational cost of prediction with a GP?
  $\mu'(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}_{x,A}(\mathbf{K}_{AA} + \sigma^2\mathbf{I})^{-1}(\mathbf{y}_A - \mu_A)$
  $k'(\mathbf{x},\mathbf{x}') = k(\mathbf{x},\mathbf{x}') - \mathbf{k}_{x,A}(\mathbf{K}_{AA} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_{x',A}^T$
- $\rightarrow$Exact computation requires solving linear system $(\mathbf{K}_{AA} + \sigma^2\mathbf{I}) \cdot Z$ in $|A| = n$ variables
- $\rightarrow\ \Theta(|A|^3)$

- This is in contrast to Bayesian linear regression: $\Theta(nd^2)$ (can even be maintained recursively at same cost)

Fast GP Methods

- Basic approaches for acceleration:
  - Exploiting parallelism (GPU computations)
  - Local GP methods
  - Kernel function approximations (RFFs, QFFs,...)
  - Inducing point methods (SoR, FITC, VFE etc.)

Fast GPs: Exploiting parallelism

- GP inference requires solving linear systems
- Resulting algorithms can be implemented on multicore (GPU) hardware
- Implemented by modern GP libraries (e.g., GPflow, GPyTorch)
- Yields substantial speedup, but doesn’t address the cubic scaling in n

Fast GPs: Local Methods

- Covariance functions that decay with distance of points (e.g., RBF, Matern, kernels) lend themselves to local computations
- To make a prediction at point x, only condition on points x’ where $|Cov(x, x’)| > \tau$ for RBF kernel, this is equivalent to $\|x - x'\| < \tau'$
- Still expensive if “many” points close by

Fast GPs: Kernel Function Approximation

- Key idea: construct **explicit "low-dimensional" feature map** that approximates the true kernel function
  $k(x, x') \approx \phi(x)^T \phi(x') \qquad \phi(x) \in \mathbb{R}^m$
- Then apply Bayesian linear regression
  $\rightarrow$Computational cost:$O(nm^2 + m^3)$ instead of $O(n^3)$
- Different variations of this idea: Random Fourier Features, Nystrom Features,...

Shift-invariant Kernels

- A kernel $k(\mathbf{x}, \mathbf{y}) \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^d$
  is called **shift-invariant** if $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} - \mathbf{y})$
- Such a kernel has a **Fourier transform**:
  $k(\mathbf{x} - \mathbf{y}) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^T(\mathbf{x}-\mathbf{y})} d\omega$
  E.g. Gaussian Kernel $k(\mathbf{x}, \mathbf{y}) = exp(-\|\mathbf{x} - \mathbf{y}\|_2^2/2)$
  has the Fourier Transform:

$p(\omega) = (2\pi)^{-d/2} exp(-\|\omega\|_2^2/2)$
This is simply the standard Gaussian distribution in D dimensions!

- Theorem [Bochner]: A shift-invariant kernel is **positive definite** if and only if $p(\omega)$ is **nonegative**
- Can scale the data, so that $p(\omega)$ is a **probability distr.**!

Random Fourier Features

- Key idea: Interpret kernel as **expectation**
$k(\mathbf{x}-\mathbf{y}) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^T(\mathbf{x}-\mathbf{y})} d\omega = \mathbb{E}_{\omega,b}[\mathbf{z}_{\omega,b}(\mathbf{x})\, \mathbf{z}_{\omega,b}(\mathbf{y})]$
$\approx \frac{1}{m} \sum_{i=1}^m \mathbf{z}_{\omega^i,b^i}(\mathbf{x})\, \mathbf{z}_{\omega^i,b^i}(\mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$
where $\omega \sim p(\omega)$, $b \sim U([0, 2\pi])$, $\mathbf{z}_{\omega,b}(\mathbf{x}) = \sqrt{2} cos(\omega^T \mathbf{x} + b)$
and $\phi(\mathbf{x}) = \frac{1}{\sqrt{m}} (\mathbf{z}_{\omega^1,b^1}(\mathbf{x}), ..., \mathbf{z}_{\omega^m,b^m}(\mathbf{x}))$
- [RR NIPS‘07]

Kernel Name	$k(\Delta)$	$p(\omega)$
Gaussian	$e^{-\frac{\|\Delta\|_2^2}{2}}$	$(2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$
Laplacian	$e^{-\|\Delta\|_1}$	$\prod_d \frac{1}{\pi(1+\omega_d^2)}$
Cauchy	$\prod_d \frac{2}{1+\Delta_d^2}$	$e^{-\|\Delta\|_1}$

- Performance of random features:
- Bayesian linear regression with explicit feature map z approximates GP

Fourier Features can be wasteful

- Fourier features approximate the kernel function **uniformly well**:
$Pr[\sup_{x,y\in\mathcal{M}} \|\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) - k(\mathbf{x},\mathbf{y})\| \geq \epsilon] \leq 2^8 (\frac{\sigma_p diam(\mathcal{M})}{\epsilon})^2 exp(-\frac{D\epsilon^2}{4(d+2)})$
- This may be "too much to ask" : Only need accurate representation for training (and test) points!

Inducing Point Methods

- "Summarize" data via function values of f at a set $\mathbf{u}$ of m inducing points
$p(\mathbf{f}^*, \mathbf{f}) = \int p(\mathbf{f}^*, \mathbf{f}, \mathbf{u}) d\mathbf{u} = \int p(\mathbf{f}^*, \mathbf{f}|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}$
- Key idea: Approximate by
$p(\mathbf{f}^*, \mathbf{f}) \approx q(\mathbf{f}^*, \mathbf{f}) = \int q(\mathbf{f}^*|\mathbf{u}) q(\mathbf{f}|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}$
- Hereby, $q(\mathbf{f}^*|\mathbf{u})$ and $q(\mathbf{f}|\mathbf{u})$ are approximations of
Training conditional $p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{f},\mathbf{u}} \mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1} \mathbf{u}, \mathbf{K}_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}})$
Testing conditional $p(\mathbf{f}^*|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{f}^*,\mathbf{u}} \mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1} \mathbf{u}, \mathbf{K}_{\mathbf{f}^*,\mathbf{f}^*} - \mathbf{Q}_{\mathbf{f}^*,\mathbf{f}^*})$
where $\mathbf{Q}_{\mathbf{a},\mathbf{b}} \equiv \mathbf{K}_{\mathbf{a},\mathbf{u}} \mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1} \mathbf{K}_{\mathbf{u},\mathbf{b}}$

Example: Subset of Regressors (SoR)

- The Subset of Regressors (SoR) approximation replaces
$p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{f},\mathbf{u}} \mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1} \mathbf{u}, \mathbf{K}_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}})$

- By
  $p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{f},\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{u}, 0)$
- Can show: the resulting model is a degenerate GP with covariance function
  $k_{SoR}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{u})\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}k(\mathbf{u}, \mathbf{x}')$

Example: Fully Independent Training Conditional (FITC)

- The FITC approximation replaces
  $p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{f},\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{u}, \mathbf{K}_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}})$
- By
  $q_{FITC}(\mathbf{f}|\mathbf{u}) = \prod_{i=1}^{n} p(f_i|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{f},\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{u}, diag(\mathbf{K}_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}}))$
- Computational Cost
- The computational cost for inducing point methods SoR and FITC is dominated by the cost of inverting $\mathbf{K}_{\mathbf{u},\mathbf{u}}$
- Thus, it is cubic in the number of inducing points, but linear in the number of data points

How to Pick Inducing Points?

- **Subsets of training data**?
  - Chosen randomly
  - Chosen greedily according to some criterion (e.g., variance)
- **Equally spaced in the domain**?
  - Random points
  - Deterministic grid
- **Optimized**?
  - Can treat $\mathbf{u}$ as hyperparameters and maximize marginal likelihood of the data
- Need to ensure $\mathbf{u}$ is representative of the data and where predictions are made

Summary

- **Gaussian processes = kernelized Bayesian Linear Regression**
- Can compute marginals / conditionals in **closed form**
- Optimize hyperparameters via **maximizing the marginal likelihood**
- Kalman filters are a **special case** of Gaussian processes

Approximate Inference

Lecture Notes

Bayesian learning more generally

- Prior: $p(\theta)$
- Likelihood: $p(y_{1:n}|x_{1:n}, \theta) \prod_{i=1}^{n} p(y_i|x_i, \theta)$
- Posterior: $p(\theta|x_{1:n}, y_{1:n}) = \frac{1}{Z}p(\theta) \prod_{i=1}^{n} p(y_i|x_i, \theta)$
  where $Z = \int p(\theta) \prod_{i=1}^{n} p(y_i|x_i, \theta) d\theta$
- Predictions: $p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, \theta) p(\theta|x_{1:n}, y_{1:n}) d\theta$
- For Bayesian linear regression and GP regression, these(high-dimensional) integrals are closed-form! 😄
- In general, this is not the case $\rightarrow$ need approximations
  - Example: Bayesian logistic regression
    $y \in \{1, -1\}$
    $\sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+exp(-\mathbf{w}^T\mathbf{x})}$
    $p(y|\mathbf{x}, \mathbf{w}) = Ber(y; \sigma(\mathbf{w}^T\mathbf{x})) = \sigma(y \cdot \mathbf{w}^T\mathbf{x})$
    $p(\mathbf{w}) = \mathcal{N}(0, \sigma_p^2\mathbf{I})$
    $p(y_{1:n}|x_{1:n}, \mathbf{w}) = \prod_{i=1}^{n} p(y_i|x_i, \mathbf{w}) = \prod_{i=1}^{n} \sigma(y; \mathbf{w}^T\mathbf{x})$

Approximate Inference

- Will discuss general approaches for performing approximate inference in intractable distributions (i.e., partition function / normalizer hard to compute)
  $p(\theta|y) = \frac{1}{Z}p(\theta, y)$
- Hereby, y are the observations (the data), and θ the latent variables (the parameters)
- We'll assume we can evaluate the joint distribution, but not the normalizer Z
- Note that we often leave out the inputs $\mathbf{x}$ to keep notation simple
  $p(y|\theta) \equiv p(y|\theta, x)$

General Approaches

- **Variational inference** seeks to approximate the intractable distribution p by a simple one q that is "as close as possible"
  $p(\theta|y) = \frac{1}{Z}p(\theta, y) \approx q(\theta|\lambda)$
- **Markov-Chain Monte Carlo** methods seek to approximate p by (approximate) samples from p (constructed by simulating a Markov Chain)

Laplace Approximation

- Laplace approximation uses a Gaussian approximation to the posterior distribution obtained from a second-order Taylor expansion around the **posterior mode**
- $q(\theta) = \mathcal{N}(\theta; \hat{\theta}, \Lambda^{-1})$
  $\hat{\theta} = argmax_{\theta} p(\theta|y)$
  $\Lambda = -\nabla\nabla log p(\hat{\theta}|y)$

- *Note*: $f(\theta) \equiv logp(\theta|y) \quad f(\theta) \approx f(\hat{\theta}) + \nabla f_{\hat{\theta}}(\theta - \hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T[\nabla\nabla f_{\hat{\theta}}](\theta - \hat{\theta})$
  any p s.t. $logp(x) = c - x^T \Lambda x$ must be Gaussian

Laplace Approx. for Bayesian log. regression

- $p(w) = \mathcal{N}(w; 0, \sigma_p^2 \mathbf{I}) = \frac{1}{Z'} exp(-\frac{1}{2\sigma_p^2}\|w\|_2^2); \quad p(y_{1:n}|w) = \prod_{i=1}^n \sigma(y_i; w^T x_i)$
  $\hat{w} = argmax_w p(w|y_{1:n}) = argmax_w \frac{1}{Z} p(w) p(y_{1:n}|w)$
  $= argmax_w logp(w) + logp(y_{1:n}|w)$
  $= argmax_w - logZ' - \frac{1}{2\sigma_p^2}\|w\|_2^2 + \sum_{i=1}^n log\sigma(y_i; w^T x_i)$
  $= argmin_w \frac{1}{2\sigma_p^2}\|w\|_2^2 + \sum_{i=1}^n log(1 + exp(-y_i w^T x_i))$
- *Note*: $\sigma(z) = \frac{1}{1+exp(-z)} \quad log\sigma(z) = -log(1 + exp(-z))$
  $\lambda = \frac{1}{2\sigma_p^2}$

Finding the Mode

- $\hat{w} = argmax_w p(w|y_{1:n}) = argmin_w \sum_{i=1}^n log(1 + exp(-y_i w^T x_i)) + \lambda\|w\|_2^2$
- This is just **standard (regularized) logistic regression**!
- Can solve, e.g., using stochastic gradient descent (see introduction to ML)
- Don't need to know normalizer Z

Recall: Stochastic Gradient Descent

- Goal: minimize stochastic objectives
  $L(\theta) := \mathbb{E}_{\mathbf{x}\sim p} l(\theta; \mathbf{x})$
- SGD:
  - Initialize θ_1
  - For $t = 1$ to T
    - Draw minibatch $B = \{\mathbf{x}_1, ..., \mathbf{x}_m\}, \mathbf{x}_i \sim p$
    - Update $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{m} \sum_{i=1}^m \nabla_\theta l(\theta_t; \mathbf{x}_i)$
- Many variants (Momentum, AdaGrad, ADAM,...)
- For proper learning rate converges to (local) minimum
- Gradient $\nabla_\theta l(\theta_t; \mathbf{x}_i)$ often obtained by automatic differentiation
- One way to choose learning rate: $\sum_t \eta_t = \infty, \quad \sum_t \eta_t^2 < \infty$ E.g. $\eta_t = \frac{c}{t}$

Recall: SGD for Logistic Regression

- Initialize $\mathbf{w}$
- For $t = 1, 2, ...$
  - Pick data point $(\mathbf{x}, y)$ uniformly at random from data D
  - Compute probability of misclassification with current model
    $\hat{P}(Y = -y|\mathbf{w}, \mathbf{x}) = \frac{1}{1+exp(y\mathbf{w}^T\mathbf{x})}$

  - Take gradient step
    $\mathbf{w} \to \mathbf{w}(1-2\lambda\eta_t) + \eta_t y \mathbf{x} \hat{P}(Y=-y|\mathbf{w},\mathbf{x})$

Finding the Covariance

- $\Lambda = -\nabla\nabla logp(\hat{\mathbf{w}}|\mathbf{x}_{1:n}, y_{1:n}) = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^t \pi_i(1-\pi_i) = \mathbf{X}^T diag([\pi_i(1-\pi_i)]_i)\mathbf{X}$
- where $\pi_i = \sigma(\hat{\mathbf{w}}^T \mathbf{x}_i)$
- *Note*: $\nabla\nabla log \frac{1}{Z} p(\theta, y) = \nabla(\nabla log \frac{1}{Z} + \nabla logp(\theta, y)) = \nabla\nabla logp(\theta, y)$
- Crucially, Λ does not depend on the normalizer Z

Making Predictions

- Suppose want to predict
  $p(p^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, w) p(w|x_{1:n}, y_{1:n}) dw \approx \int p(y^*|x^*, w) q_\lambda(w) dw$
  $= \int p(y^*|f^*) q(f^*) df^*$
  This integral still has no closed form, but is easy to approximate(to machine precision), e.g. Gauss-Hermite quadrature $f(x) \approx \sum_i w_i f(x_i)$
  Can also do sample based approx: $w^{(1)}, ..., w^{(m)} \sim q_\lambda, \quad p(y^*|...) = \frac{1}{m} \sum_{i=1}^{m} p(y^*|x^*, w^{(i)})$
- *Note*:
  - $f^* = w^T x^*, \quad p(y^*|f^*) = \sigma(y^* f^*)$
  - $q(f^*) \equiv \int p(f^*|w) q_\lambda(w) dw$
    If $q_\lambda = \mathcal{N}(\hat{w}, \Lambda^{-1}) \to q(f^*) = \mathcal{N}(f^*; \hat{w}^T x^*, x^{*T} \Lambda^{-1} x^*)$
- This one-dimensional integral can be easily approximated efficiently using numerical quadrature
- [Side note: For other link functions (e.g., Gaussian CDF), can even be calculated analytically]

Issues with Laplace Approximation

- Laplace approximation first greedily seeks the mode, and then matches the curvature
- his can lead to poor (e.g., overconfident) approximations

Variational Inference

- Given unnormalized distribution
  $p(\theta|y) = \frac{1}{Z} p(\theta, y)$
- Try to find a “simple” (tractable) distribution that approximates p well
  $q^* \in argmin_{q \in \mathcal{Q}} \; KL(q||p) = argmin_{\lambda \in \mathbb{R}^D} \; KL(q_\lambda||p)$

Simple Distributions

- Need to specify a **variational family** (of simple distributions)
- E.g.: Gaussian distributions; Gaussians with diagonal covariance,...
  $\mathcal{Q} = \{q(\theta) = \mathcal{N}(\theta; \mu, diag([\sigma]))\}$
  $q = q_\lambda$, where $\lambda = [\mu, \sigma^2]$

KL-Divergence

- Given distributions q and p, Kullback-Leibler divergence between q and p is
  $KL(q \| p) = \int q(\theta) log \frac{q(\theta)}{p(\theta)} d\theta = \mathbb{E}_{\theta \sim q}[log \frac{q(\theta}{p(\theta}]$
  Typically, we assume $p \& q$ have same support
- Properties
  - Non-negative: $KL(q \| p) \geq 0 \quad \forall q, p$
  - Zero if and only if $p \& q$ agree almost everywhere: $KL(q \| p) = 0 \Leftrightarrow q = p$
  - Not generally symmetric: $KL(q \| p) \neq KL(p \| q)$

Example: KL Divergence Between Gaussians

- Consider two Gaussian distributions p and q
  $p = \mathcal{N}(\mu_0, \Sigma_0),\ q = \mathcal{N}(\mu_1, \Sigma_1)$
- Then it holds that
  $KL(p \| q) = \frac{1}{2}(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - d + ln(\frac{|\Sigma_1|}{|\Sigma_0|}))$
- If $p = \mathcal{N}([\mu_1, ..., \mu_d], diag([\sigma_1^2, ..., \sigma_d^2]))$ and $q = \mathcal{N}(0, I)$
  $KL(p \| q) = \frac{1}{2}\sum_{i=1}^d (\sigma_i^2 + \mu_i^2 - 1 - ln\sigma_i^2)$
- Suppose $p = \mathcal{N}(\mu_0, I), q = \mathcal{N}(\mu_1, I)$
  $KL(p \| q) = \frac{1}{2}\|\mu_0 - \mu_1\|_2^2$

Entropy

- Entropy of a distribution:
  $H(q) = -\int q(\theta) logq(\theta) d\theta = \mathbb{E}_{\theta \sim q}[-logq(\theta)]$
- Entropy of a product distribution: $q(\theta_{1:d}) = \prod_{i=1}^d q_i(\theta_i)$
  $H(q) = \sum_{i=1}^d H(q_i)$
- Example: Entropy of a Gaussian
  $H(\mathcal{N}(\mu, \Sigma)) = \frac{1}{2} ln|2\pi e \Sigma|$
  For $\Sigma = diag(\sigma_1^2, ..., \sigma_d^2) \Rightarrow H = \frac{1}{2} ln|2\pi e| + \sum_{i=1}^d ln\sigma_i^2$

Minimizing KL Divergence

- $$\begin{aligned} argmin_q KL(q \| p) &= argmin_q \int q(\theta) log \frac{q(\theta)}{\frac{1}{Z}p(\theta, y)} d\theta \\ &= argmax_q \int q(\theta)[logp(\theta, y) - logZ - logq(\theta)] d\theta \\ &= argmax_q \int q(\theta) logp(\theta, y) d\theta + H(q) \\ &= argmax_q \mathbb{E}_{\theta \sim q(\theta)}[logp(\theta, y)] + H(q) \\ &= argmax_q \mathbb{E}_{\theta \sim q(\theta)}[logp(y|\theta)] - KL(q \| p(\cdot)) \end{aligned}$$

- *Note*:
  p is posterior, $p(\cdot)$ is prior

Maximizing Lower Bound on Evidence

- $$\begin{aligned} logp(y) &= log\int p(y|\theta)p(\theta)d\theta \\ &= log\int p(y|\theta)\frac{p(\theta)}{q(\theta)}q(\theta)d\theta \\ &= log\mathbb{E}_{\theta\sim q}[p(y|\theta)\frac{p(\theta)}{q(\theta)}] \\ &\geq \mathbb{E}_{\theta\sim q}[log(p(y|\theta)\frac{p(\theta)}{q(\theta)})] \\ &= \mathbb{E}_{\theta\sim q}[log(p(y|\theta)]d\theta - KL(q\|p(\cdot)) \end{aligned}$$

Inference as Optimization

- Thus,
  $$\begin{aligned} argmin_q KL(q\|p(\cdot|y)) &= argmax_q\mathbb{E}_{\theta\sim q(\theta)}[logp(y|\theta)] - KL(q\|p(\cdot)) \\ &= argmax_q\mathbb{E}_{\theta\sim q(\theta)}[logp(\theta,y)] + H(q) \\ &= argmax_q L(q) \end{aligned}$$
- Thus, prefer distributions q that maximize the expected (**joint/conditional**) data likelihood, but are also **uncertain / close** to the prior
- *Note*:
  $L(q)$ is called **"ELBO" (Evidence lower bound)**
  $L(q) \leq log(p(y)$ $\leftarrow$evidence

ELBO for Bayesian Logistic Regression

- $L(\lambda) = \mathbb{E}_{\theta\sim q(\cdot|\lambda)}[logp(y|\theta)] - KL(q_\lambda\|p(\cdot))$
  Suppose: Q is diagonal Gaussians $\rightarrow \lambda = [\mu_{1:d}, \sigma^2_{1:d}] \in \mathbb{R}^{2d}, \quad p(\theta) = \mathcal{N}(0, I)$
  $\rightarrow KL(q_\lambda\|p(\cdot)) = \frac{1}{2}\sum_{i=1}^{d}(\mu_i^2 + \sigma_i^2 - 1 - ln\sigma_i^2)$
- $$\begin{aligned} \mathbb{E}_{\theta\sim q_\lambda}[log(p(y|\theta)] &= \mathbb{E}_{\theta\sim q_\lambda}[\sum_{i=1}^{n} log(p(y_i|\theta, x_i)] \\ &= \mathbb{E}_{\theta\sim q_\lambda}[-\sum_{i=1}^{n} log(1 + exp(-y_i\theta^T x_i))] \end{aligned}$$

Gradient of the ELBO

- $$\begin{aligned}\nabla_\lambda L(\lambda) &= \nabla_\lambda[\mathbb{E}_{\theta\sim q(\cdot|\lambda)}[logp(y|\theta)] - KL(q_\lambda \| p(\cdot))] \\ &= \nabla_\lambda[\mathbb{E}_{\theta\sim q(\cdot|\lambda)}[logp(\theta, y)] + H(q(\cdot|\lambda))]\end{aligned}$$
- Need to differentiate an expectation w.r.t. q
- Unfortunately **q depends on the variational params**.
- Key idea: Rewrite in a way that allows Monte Carlo approximation. Different approaches
  - Score gradients (not discussed further here)
  - Reparametrization gradients

Reparameterization Trick

- Suppose we have a random variable $\epsilon \sim \phi$ sampled from a base distribution, and consider $\theta = g(\epsilon, \lambda)$ for some invertible function g
- Then it holds that $q(\theta|\lambda) = \phi(\epsilon)|\nabla_\epsilon g(\epsilon; \lambda|^{-1}$
  (change of variables for probability) and $\mathbb{E}_{\theta\sim q_\lambda}[f(\theta)] = \mathbb{E}_{\epsilon\sim\phi}[f(g(\epsilon; \lambda))]$
- Thus, after reparameterization, the expectation is w.r.t. to distribution ϕ that **does not depend** on λ !
- This allows to **obtain stochastic gradients** via
  $\nabla_\lambda \mathbb{E}_{\theta\sim q_\lambda}[f(\theta)] = \mathbb{E}_{\epsilon\sim\phi}[\nabla_\lambda f(g(\epsilon; \lambda))]$

Example: Gaussians

- Suppose we use a Gaussian variational approximation
  $q(\theta|\lambda) = \mathcal{N}(\theta; \mu, \Sigma); \quad \lambda = [\mu, \Sigma]$
- Can reparametrize $\theta = g(\epsilon, \lambda) = C\epsilon + \mu$, such that $\Sigma = CC^T$ and $\phi(\epsilon) = \mathcal{N}(\epsilon; 0, I)$
- Then it holds that $\epsilon = C^{-1}(\theta - \mu)$ and $\phi(\epsilon) = q(\theta|\lambda)|C|$
- Can w.l.o.g. choose C to be positive definite and lower-diagonal(C is Cholesky factor of Σ)

Reparametrizing the ELBO for Bayesian Logistic Regression

- $$\begin{aligned}\nabla_\lambda L(\lambda) &= \nabla_\lambda[\mathbb{E}_{\theta\sim q(\cdot|\lambda)}[logp(y|\theta)] - KL(q_\lambda \| p(\cdot))] \\ &= \nabla_{C,\mu}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[logp(y|C\epsilon + \mu)] - \nabla_{C,\mu}KL(q_{C,\mu} \| p(\cdot))\end{aligned}$$
- Can compute $\nabla_{C,\mu}KL(q_{C,\mu} \| p(\cdot))$ exactly (e.g., via automatic differentiation)
- Can obtain unbiased stochastic gradient estimate of
  $$\begin{aligned} &\nabla_{C,\mu}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[logp(y|C\epsilon + \mu)] \\ =&\nabla_{C,\mu}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[n \cdot \frac{1}{n}\sum_{i=1}^{n} logp(y_i|C\epsilon + \mu, x_i)] \\ =&\nabla_{C,\mu} n \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\mathbb{E}_{i\sim Unif\{1:n\}} logp(y_i|C\epsilon + \mu, x_i) \\ \approx&\nabla_{C,\mu} n \cdot \frac{1}{m}\sum_{j=1}^{m} logp(y_{i_j}|C\epsilon^{(j)} + \mu, x_{i_j})\end{aligned}$$

- *Note*:
  - Draw mini-batch $\epsilon^{(1)}, ..., \epsilon^{(m)} \sim \phi$
  - Draw $i_1, ..., i_m \sim Unif\{1, ..., n\}$

Black Box Stochastic Variational Inference

- Maximizing the ELBO using stochastic optimization (e.g., Stochastic Gradient Ascent)
- Can obtain unbiased gradient estimates, e.g., via **reparameterization trick**, or **score gradients**:
  $\nabla_\lambda L(\lambda) = \mathbb{E}_{\theta \sim q_\lambda}[\nabla_\lambda logq(\theta|\lambda)(logp(y, \theta) - logq(\theta|\lambda))]$
- For diagonal q, only twice as expensive as MAP infer.
- Only need to be able to differentiate the (unnormalized) joint probability density p and q
- Outlook: Can achieve better performance, e.g., using
  - Natural gradients
  - Variance reduction techniques (e.g., control variates)

Side Note: Gaussian Process Classification

- All our discussions naturally generalize from Bayesian linear regression to Gaussian process classification:
  $P(f) = GP(\mu, k) \quad P(y|f, \mathbf{x}) = \sigma(y \cdot f(\mathbf{x}))$
- Often implemented using pseudo inputs, and maximizing the ELBO
  $\sum_{i=1}^{n} \mathbb{E}_{q(f_i)}[logp(y_i|f_i)] - KL(q(\mathbf{u})\|p(\mathbf{u}))$
  where $q(f_i) := \int p(f_i|\mathbf{u})q(\mathbf{u})d\mathbf{u}$

Variational Inference Summary

- Variational inference **reduces inference** (“summation/integration”) **to optimization**
- Can use highly efficient stochastic optimization techniques to find approximations
- Quality of approximation hard to analyze

Markov Chain Monte Carlo

Lecture Notes

Approximating Predictive Distributions

- Key challenge in Bayesian learning: Computing
$$\begin{aligned} p(y^*|x^*, x_{1:n}, y_{1:n}) &= \int p(y^*|x^*, \theta) p(\theta|x_{1:n}, y_{1:n}) d\theta \\ &= \mathbb{E}_{\theta \sim p(\cdot|x_{1:n}, y_{1:n})}[f(\theta)] \\ &\approx \frac{1}{m} \sum_{i=1}^{m} f(\theta^{(i)}) \end{aligned}$$
where $\theta^{(i)} \sim p(\theta|x_{1:n}, y_{1:n})$
- If we had access to samples from the posterior, could use to obtain **Monte-Carlo approximation** of predictive distribution

Sample Approximations of Expectations

- $x_1, ...x_N, ...$ independent samples from $P(X)$
- (Strong) Law of large numbers:
$\mathbb{E}_P[f(X)] = lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} f(x_i)$
- Hereby, the convergence is with probability 1 (almost sure convergence)
- Suggests **approximation** using **finite samples**:
$\mathbb{E}_P[f(X)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$

How Many Samples Do We Need?

- **Hoeffding's inequality** Suppose f is bounded in $[0, C]$. Then
$P(|\mathbb{E}_P[f(X)] - \frac{1}{N} \sum_{i=1}^{N} f(x_i)| ? \varepsilon) \leq 2 exp(-2N\varepsilon^2/C^2)$
- Thus, probability of error decreases exponentially in N!

Sampling From Intractable Distributions

- Given unnormalized distribution
$P(x) = \frac{1}{Z} Q(x)$
- $Q(X)$ efficient to evaluate, but normalizer Z intractable
- How can we sample from $P(X)$?
- Ingenious idea: Can create Markov chain that is efficient to simulate and that has stationary distribution $P(X)$

Markov Chains

- A (stationary) Markov chain is a sequence of RVs, $X_1, ..., X_N, ...$ with
  - Prior $P(X_1)
  - Transition probabilities $P(X_{t+1}|x_t)$ independent of t
  $X_{t+1} \perp X_{1:t-1} | X_t \quad \forall t$
  $P(X_{1:N}) = P((X_1) P(X_2|X_1) ... P(X_N|X_{N-1})$

Ergodic Markov Chains

- A Markov Chain is called ergodic, if there exists a finite t such that every state can be reached from every state in exactly t steps

Stationary Distributions

- An (stationary)ergodic Markov Chain has a unique and positive stationary distribution $\pi(X) > 0$, s.t. for all x
  $lim_{N \to \infty} P(X_N = x) = \pi(x)$
- The stationary distribution is independent of $P(X_1)$

Simulating a Markov Chain

- Can simulate a Markov chain via forward sampling:
  $P(X_{1:N}) = P((X_1)P(X_2|X_1)...P(X_N|X_{N-1})$
- If simulated “sufficiently long”, sample X_N is drawn from a distribution “very close” to stationary distribution π

Markov Chain Monte Carlo

- Given an unnormalized distribution $Q(x)$
- Want to design a Markov chain with stationary distribution
  $\pi(x) = \frac{1}{Z}Q(x)$
- Need to specify transition probabilities $P(x|x')$
- How can we choose them to ensure correct stationary distribution?

Detailed Balance Equation

- A Markov Chain satisfies the detailed balance equation for unnormalized distribution Q if for all x, x’:
  $\frac{1}{Z}Q(x)P(x'|x) = \frac{1}{Z}Q(x')P(x|x')$
- Suffices to show: $P(X_t = x) = \frac{1}{Z}Q(x) \Rightarrow P(X_{t+1} = x) = \frac{1}{Z}Q(x)$
  - Assume $P(X_t = x) = \frac{1}{Z}Q(x)$

- Then
$$\begin{aligned} P(X_{t+1}=x) &= \sum_{x'} P(X_{t+1}=x, X_t=x') \\ &= \sum_{x'} P(X_{t+1}=x|X_t=x')P(X_t=x') \\ &= \frac{1}{Z}\sum_{x'} P(x|x')Q(x') \\ &\overset{D.B.}{=} \frac{1}{Z}\sum_{x'} P(x'|x)Q(x) \\ &= \frac{1}{Z}Q(x)\sum_{x'} P(x'|x) \\ &= \frac{1}{Z}Q(x) \end{aligned}$$

Designing Markov Chains

- 1. Proposal distribution $R(X'|X)$
    - Given $X_t = x$, sample "proposal" $x' \sim R(X'|X=x)$
    - Note: Performance of algorithm will strongly depend on R
- 2. Acceptance distribution:
    - Suppose $X_t = x$
    - With probability $\alpha = min\{1, \frac{\frac{1}{Z}Q(x')R(x|x')}{\frac{1}{Z}Q(x)R(x'|x)}\}$ set $X_{t+1} = x'$
    - With probability $1-\alpha$, set $X_{t+1} = x$
- Theorem [Metropolis, Hastings]: The stationary distribution is $Z^{-1}Q(x)$
    - Proof: Markov chain satisfies detailed balance condition!

MCMC for Random Vectors

- Markov chain state can be a vector $\mathbf{X} = (X_1, ..., X_n)$
- Need to specify proposal distributions $R(x'|x)$ over such random vectors
    - x: old state (joint configuration of all variables)
    - x': proposed state, $x' \sim R(X'|X=x)$
- One popular example: Gibbs sampling!

Gibbs Sampling: Random Order

- Start with initial assignment x to all variables
- Fix observed variables X_B to their observed value X_B
- For $t = 1$ to ∞ do
    - Pick a variable i uniformly at random from $\{1, ..., n\}\backslash\mathbf{B}$
    - Set $\mathbf{v}_i$ =values of all x except x_i
    - Update x_i by sampling from $P(X_i|\mathbf{v}_i)$

- Satisfies detailed balance equation!

Gibbs Sampling: Practical Variant

- Start with initial assignment $\mathbf{x}^{(0)}$ to all variables
- Fix observed variables $\mathbf{X_B}$ to their observed value $\mathbf{x_B}$
- For $t = 1$ to ∞ do
  - Set $\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)}$
  - For each variable X_i(except those in $\mathbf{B}$)
    - set $\mathbf{v}_i$=values of all $\mathbf{x}^{(t)}$ except x_i
    - Sample ${x^{(t)}}_i$ from $P(X_i|\mathbf{v}_i)$
- No detailed balance, but also has correct stationary distribution.

Computing $P(X_i|\mathbf{v}_i)$

- Key insight in Gibbs sampling: Sampling from X_i given an assignment to **all** other variables is (typ.) efficient!
- Generally, can compute
  $P(X_i|\mathbf{v}_i) = \frac{1}{Z}Q(X_i|\mathbf{v}_i) = \frac{1}{Z}Q(X_{1:N})$
  where $Z = \sum_x Q(X_i = x, \mathbf{v}_i)$
- Thus, re-sampling X_i only requires evaluating unnormalized joint distr. and renormalizing!
- Example: (Simple) Image Segmentation: see lecture notes

Ergodic Theorem (special case)

- Suppose $X_1, X_2, ..., X_N, ..$ is an ergodic Markov chain over a finite state space D, with stationary distribution π. Further let f be a function on D.
- Then it holds a.s. that
  $lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} f(x_i) = \sum_{x\in D}\pi(x)f(x) = \mathbb{E}_{x\sim\pi}f(x)$
- This is a strong law of large numbers for Markov chains!

Computing Expectations with MCMC

- Joint sample at time t depends on sample at time $t-1$
- Thus the law of large numbers (and sample complexity bounds such as Hoeffding's inequality) **do not apply**
- Use MCMC sampler to obtain samples
  $\mathbf{X}^{(1)}, ..., \mathbf{X}^{(T)}$
- To let the Markov chain "burn in", ignore the first t_0 samples, and approximate
  $\mathbb{E}[f(\mathbf{X})] \approx \frac{1}{T-t_0}\sum_{\tau=t_0+1}^{T} f(\mathbf{X}^{(\tau)})$
- Establishing convergence rates generally very difficult

MCMC for Continuous RVs

- MCMC techniques can be generalized to continuous random variables / vectors
- We focus on positive distributions w.l.o.g. written as
  $p(\mathbf{x}) = \frac{1}{Z} exp(-f(\mathbf{x}))$
  where f is called an energy function
- Distributions p s.t. f is convex are called **log-concave**
- Example: Bayesian logistic regression
  $p(\theta|x_{1:n}, y_{1:n}) = \frac{1}{Z} p(\theta) p(y_{1:n}\theta, x_{1:n}) = \frac{1}{Z} exp(-logp(\theta) - logp(y_{1:n}\theta, x_{1:n}))$
  where $f(\theta) = \lambda \|\theta\|_2^2 + \sum_{i=1}^{n} log(1 + exp(-y; \theta^T x_i)) + const.$

Recall: Metropolis Hastings

- 1. Proposal distribution $R(X'|X) \quad Q(x) = exp(-f(x))$
  - Given $X_t = x$, sample "proposal" $x' \sim R(X'|X = x)$
- 2. Acceptance distribution:
  - Suppose $X_t = x$
  - With probability $\alpha = min\{1, \frac{R(x|x')}{R(x'|x)} exp(f(x) - f(x'))\}$ set $X_{t+1} = x'$
  - With probability $1 - \alpha$, set $X_{t+1} = x$
- What proposals R should we use?

Proposals

- Open option: $R(x'|x) = \mathcal{N}(x'; x; \tau I)$
  $x' = x + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \tau I)$
- Acceptance probability?
  - *Note*: $\frac{\mathcal{N}(x|x', \tau I)}{\mathcal{N}(x'|x, \tau I)} = 1$
  - so that $\alpha = min\{1, exp(f(x) - f(x'))\}$
    if $f(x') < f(x) \Leftrightarrow Q(x') > Q(x) \ \rightarrow \alpha = 1$
    if $f(x') > f(x) \rightarrow 0 < \alpha < 1$
- Simple update, but "uninformed" direction

Improved Proposals

- Can take gradient information into account to prefer proposals into regions with higher density
  $R(x'|x) = \mathcal{N}(x'; x - \tau \nabla f(x); 2\tau I)$
  $x' = x - \tau \nabla f(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 2\tau I)$
- The resulting sampler is called **Metropolis adjusted Langevin Algorithm** (MALA; a.k.a. Langevin Monte Carlo, LMC)

Guarantees for MALA

- It is possible to show that for log-concave distributions (e.g., Bayesian log. Regression), MALA efficiently converges to the stationary distribution (mixing time is polynomial in the dimension)

$Q(x) = \frac{1}{Z} exp(-f(x))$ is log-concave **iff** f convex

- In fact, locally the function is allowed to be non-convex

Improving Efficiency?

- Both the proposal and acceptance step in MALA/LMC require access to the full energy function f
- For large data sets, that can be **expensive**
- Key idea:
  - Use stochastic gradient estimates
  - Use decaying step sizes and skip accept/reject step
- $\rightarrow$ **Stochastic Gradient Langevin Dynamics (SGLD)**

Stochastic Gradient Langevin Dynamics

- Consider sampling from the Bayesian posterior $\theta \sim \frac{1}{Z} exp(logp(\theta) + \sum_{i=1}^{n} logp(y_i|x_i, \theta))$
- SGLD produces (approximate) samples as follows:
  - Initialize θ_0
  - For $t = 0, 1, 2, ...$ do
    - $\epsilon_t \sim \mathcal{N}(0, 2\eta_t I)$
    - $\theta_{t+1} = \theta_k = \eta_t(\nabla logp(\theta_t) + \frac{n}{m} \sum_{j=1}^{m} \nabla logp(y_{i_j}|\theta_t, x_{i_j})) + \epsilon_t$

Guarantees for SGLD

- **SGLD = SGD + Gaussian noise**
- Can **guarantee convergence** to the stationary distribution (under some assumptions) as long as $\eta_t \in \Theta(t^{-1/3})$
- In practice, one often uses **constant step sizes** to accelerate mixing (but needs tuning)
- Can improve performance via **preconditioning** (cf. Adagrad etc. for optimization)

Outlook: Hamiltonian Monte Carlo (HMC)

- Often, performance of (S)GD can be improved by adding a **momentum term**
- As SGLD/MALA can be seen as a sampling-based analogue of SGD, a similar analogue for (S)GD with momentum is the **Hamiltonian Monte Carlo algorithm**

Summary: MCMC

- **Markov Chain Monte Carlo** methods simulate a carefully designed Markov Chain to approximately sample from an intractable distribution
- Can be used for Bayesian learning
- For continuous distributions can make use of **(stochastic) gradient information** in the proposals
- Guaranteed, **efficient convergence for log-concave densities** (e.g., Bayesian logistic regression)

- In general, can guarantee convergence to the target distribution (in constrast to VI); however, for general distributions, convergence / mixing may be **slow**
- $\rightarrow$Tradeoff between accuracy and efficiency

Bayesian Deep Learning

Lecture Notes

Bayesian Learning

- Prior: $p(\theta)$
- Likelihood: $p(y_{1:n}|x_{1:n}, \theta) \prod_{i=1}^{n} p(y_i|x_i, \theta)$
- Posterior: $p(\theta|x_{1:n}, y_{1:n}) = \frac{1}{Z} p(\theta) \prod_{i=1}^{n} p(y_i|x_i, \theta)$
  where $Z = \int p(\theta) \prod_{i=1}^{n} p(y_i|x_i, \theta) d\theta$
- Predictions: $p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, \theta) p(\theta|x_{1:n}, y_{1:n}) d\theta$

Beyond Linear Models

- So far, we've discussed effective approximate inference techniques for **Bayesian linear regression** and **Bayesian logistic regression** (linear classification)
  $p(y|\mathbf{x}, \theta) = \mathcal{N}(y; \theta^T \mathbf{x}, \sigma^2)$
  $p(y|\mathbf{x}, \theta) = Ber(y; \sigma(\theta^T \mathbf{x}))$
- Here, likelihood functions have parameters **linearly dependent** on the inputs
- In practice, can often get better performance by considering **nonlinear** dependencies

(Deep) Artificial Neural Networks

- $f(\mathbf{x}; \mathbf{w}) = \varphi(\mathbf{W}_1 \varphi(\mathbf{W}_2(\ldots\varphi(\mathbf{W}_l \mathbf{x}))))$
- Flexible nonlinear functions with many (often 10^8) parameters
- Deep = "nested" in many layers
- Loosely inspired by biological neuronal networks

Some Deep Learning Success Stories

- State of the art performance on some difficult classification tasks
- Speech recognition (TIMIT)
- Image recognition (MNIST, ImageNet)
- Natural language processing (semantic word embeddings)
- Speech translation
- A lot of recent work on sequential models (Recurrent Neural Networks, LSTMs, GRUs, Tramsformers,...) and models on graphs

Activation Functions

- Hereby, $\theta \in \mathbb{R}^d$ and $\varphi : \mathbb{R} \to \mathbb{R}$ is a nonlinear function, called "activation function"
  $\phi(\mathbf{x}, \theta) = \varphi(\theta^T \mathbf{x})$

Bayesian Neural Networks

- Bayesian neural network models specify a prior distribution over weights, and use likelihood functions parameterized via neural networks
- Simple example:
  - Gaussian prior on weights: $p(\theta) = \mathcal{N}(\theta; 0, \sigma_p^2 I)$
  - Likelihood: $p(y|\mathbf{x}, \theta) = \mathcal{N}(y; f(\mathbf{x}, \theta), \sigma^2)$ as opposed to $\theta^T \mathbf{x}$ in BLR

Heteroscedastic Noise

- Noise depends on input

Modeling heteroscedastic noise with NNs

- Use more complex likelihood:
  - $p(y|\mathbf{x}, \theta) = \mathcal{N}(y; f_1(\mathbf{x}, \theta), exp(f_2(\mathbf{x}, \theta)))$
    Model both mean and (log) variance as (two different) outputs of a neural network

MAP Estimates for Bayesian NNs

- MAP estimate for heteroscedastic regression with NNs
  $\hat{\theta} = argmin_\theta - logp(\theta) - \sum_{i=1}^n logp(y_i|\mathbf{x}_i, \theta)$
  $$\begin{aligned} logp(y|\mathbf{x}, \theta) &= log\mathcal{N}(y; \mu(\mathbf{x}, \theta), \sigma^2(\mathbf{x}, \theta)) \\ &= log(\frac{1}{\sqrt{2\pi\sigma^2(\mathbf{x}, \theta)}} exp(-\frac{(y-\mu(\mathbf{x}, \theta))^2}{2\sigma^2(\mathbf{x}, \theta)})) \\ &= log\frac{1}{\sqrt{w\pi}} - \frac{1}{2} log\sigma^2(\mathbf{x}, \theta) - \frac{1}{2}\frac{(y-\mu(\mathbf{x}, \theta))^2}{\sigma^2(\mathbf{x}, \theta)} \end{aligned}$$
- MAP estimate for heteroscedastic regression with NNs
  $$\hat{\theta} = argmin_\theta - logp(\theta) - \sum_{i=1}^n logp(y_i|\mathbf{x}_i, \theta)$$
  $$= argmin_\theta \lambda\|\theta\|_2^2 + \sum_{i=1}^n [\frac{1}{2\sigma^2(\mathbf{x}_i, \theta)}\|y_i - \mu(\mathbf{x}, \theta)\|^2 + \frac{1}{2} log\sigma^2(\mathbf{x}_i, \theta)]$$
- Thus, the network can **attenuate the losses** for certain data points by **attributing the error** to large variance

Recall: MAP Inference in BNNs

- Finding the **MAP parameter** estimates in BNNs can be accomplished by minimizing
  $\hat{\theta} = argmin_\theta - logp(\theta) - \sum_{i=1}^n logp(y_i|\mathbf{x}_i, \theta)$
  e.g., via Stochastic Gradient Descent (and variants)
- Gradients can be computed using auto-differentiation techiques (implemented, e.g., in PyTorch, Tensorflow)
- Gaussian priors on the weight are equivalent to
  applying weight decay
  $nabla(-logp(\theta)) = \nabla\lambda||\theta||_2^2 = 2\lambda\theta$
  $\theta_{t+1} \leftarrow \theta_t - \eta_t \nabla logp(\theta_t) - \eta_t \nabla \sum_{i=1}^n logp(y_i|x_i, \theta_t)$
  $\Rightarrow \theta_{t+1} \leftarrow \theta_t(1 - 2\lambda\eta_t) - \eta_t \nabla \sum_{i=1}^n logp(y_i|x_i, \theta_t)$

Approximate Inference for BNNs

- Bayesian learning integrals for posterior and predictions for NN are intractable, thus need approximate inference.
- Can use approximate inference techniques at similar cost as MAP/SGD
  - Black-box stochastic variational inference
  - Stochastic gradient Langevin dynamics
- Only need to be able to **compute gradients**, which can be done using automatic differentiation (backpropagation)
- Also specialized approaches tailored for BNNs
  - Monte-carlo Dropout
  - Probabilistic Ensmbles

Variational Inference for Bayesian neural networks

Variational inference for BNNs (aka Bayes by Backprop)

- Recall, variational inference aims to find the best-approximating variational distribution q via
  $argmin_q KL(q||p(\cdot|y)) = argmax_q \mathbb{E}_{\theta \sim q(\theta)}[logp(y|\theta)] - KL(q||p(\cdot))$
- For Gaussian $q(\theta|\lambda) = \mathcal{N}(\theta; \mu, \Sigma)$ can obtain stochastic gradients, e.g., via reparametrization trick

$$\begin{aligned}\nabla_\lambda L(\lambda) &= \nabla_\lambda[\mathbb{E}_{\theta \sim q(\cdot|\lambda)}[logp(y|\theta)] - KL(q_\lambda||p(\cdot))] \\ &= \nabla_{C,\mu}\mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[logp(y|C\epsilon + \mu)] - \nabla_{C,\mu}KL(q_{C,\mu}||p(\cdot)) \\ &\approx \nabla_{C,\mu} fracnm \sum_{j=1}^m logp(y_{i_j}|C\epsilon^{(j)} + \mu, x_{i_j}) - \nabla_{C,\mu}KL(q_{C,\mu}||p(\cdot))\end{aligned}$$

Making Predictions

- Given variational posterior q, can approximate predictive distributions by sampling from it

$$\begin{aligned} p(y^*|x^*, x_{1:n}, y_{1:n}) &= \int p(y^*|x^*, \theta) p(\theta|x_{1:n}, y_{1:n}) d\theta \\ &= \mathbb{E}_{\theta \sim p(\cdot|x_{1:n}, y_{1:n})}[p(y^*|x^*, \theta)] \\ &\overset{V.I.}{\approx} \mathbb{E}_{\theta \sim q(\cdot|\lambda)}[p(y^*|x^*, \theta)] \\ &\overset{M.C.}{\approx} \frac{1}{m} \sum_{j=1}^{m} p(y^*|x^*, \theta^{(j)}) \end{aligned}$$

  where $\theta^{(j)} \approx q(\cdot|\lambda)$
- *Note*: one choice: $p(y^*|x^*, \theta^{(j)}) = \mathcal{N}(y^*; \mu(x^*, \theta^{(j)}), \sigma^2(x^*, \theta^{(j)}))$
- i.e., **draw m sets of weights** from posterior, and **average the neural network predictions**
- For Gaussian likelihoods, approximate predictive distribution becomes a mixture of Gaussians

Aleatoric vs. Epistemic Uncertainty for Gaussian Likelihoods

- $p(y^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) \approx \frac{1}{m} \sum_{j=1}^{m} \mathcal{N}(y^*; \mu(\mathbf{x}^*, \theta^{(j)}), \sigma^2(\mathbf{x}^*, \theta^{(j)}))$
- Mean: $\mathbb{E}[y^*|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \mathbf{x}^*] \approx \bar{\mu}(\mathbf{x}^*) := \frac{1}{m} \sum_{m=1}^{m} \mu(\mathbf{x}^*, \theta^{(j)})$
- Law of Total Variance:
  RVs. θ, y, $Var(y) = \mathbb{E}_\theta Var[y|\theta] + Var\mathbb{E}_y[y|\theta]$
- Variance(via LoTV)

$$\begin{aligned} Var[y^*|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \mathbf{x}^*] &= \mathbb{E}[Var[y^*|\mathbf{x}^*, \theta]] + Var[\mathbb{E}[y^*|\mathbf{x}^*, \theta]] \\ &\approx \frac{1}{m} \sum_{j=1}^{m} \sigma^2(\mathbf{x}^*, \theta^{(j)}) + \frac{1}{m} \sum_{j=1}^{m} (\mu(\mathbf{x}^*, \theta^{(j)} - \bar{\mu}(\mathbf{x}^*))^2 \end{aligned}$$

  where $\frac{1}{m} \sum_{j=1}^{m} \sigma^2(\mathbf{x}^*, \theta^{(j)})$ is Aleatoric uncertainty, and $\frac{1}{m} \sum_{j=1}^{m} (\mu(\mathbf{x}^*, \theta^{(j)} - \bar{\mu}(\mathbf{x}^*))^2$ is Epistemic uncertainty.

Markov-Chain Monte Carlo for Bayesian Neural Networks

MCMC for Neural Networks

- Similarly to variational inference, can apply **MCMC methods** to train deep neural network models such as
  - (Preconditioned) Stochastic Gradient Langevin Dynamics
    $\theta_{t+1} = \theta_t - \eta_t(\nabla logp(\theta_t) + \frac{n}{m} \sum_{j=1}^{m} \nabla logp(y_{i_j}|\theta_t, x_{i_j})) + \epsilon_t$
  - Metropolis adjusted Langevin Dynamics*
  - Stochastic Gradient Hamiltonian Monte Carlo
  - ...
- These methods **only require stochastic gradients** of the (unnormalized) joint probability, i.e., the same gradients used for MAP estimation (e.g., via SGD)

Predicting with MCMC

- MCMC methods (like SGLD) produce a sequence of iterates (NN weights) $\theta^{(1)}, ..., \theta^{(T)}$
- The ergodic theorem justifies making predictions with
  $p(y^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) \approx \frac{1}{T}\sum_{j=1}^{T} p(y^*|\mathbf{x}^*, \theta^{(j)}) \approx \frac{1}{m}\sum_{j=1}^{m} p(y^*|\mathbf{x}^*, \theta^{(t_j)}) \leftarrow j^{th}$snapshot
- Challenges:
  - Typically, **cannot afford to store** all T samples / models
  - To **avoid the "burn-in" period**, need to drop first samples

Summarizing MCMC Iterates

- Approach 1: **Subsampling**
  - Simply keep a subset of m "snapshots
- Approach 2: **Gaussian approximation**
  - Keep track of a Gaussian approximation of the parameters $q(\theta|\mu_{1:d}, \sigma^2_{1:d})$, where
    $\mu_i^{(T)} = \frac{1}{T}\sum_{i=1}^{T} \theta_i^{(j)} \qquad \sigma_i^2 = \frac{1}{T}\sum_{j=1}^{T}(\theta_i^{(j)} - \mu_i)^2$
    i is NN parameter index, j is iteration of MCMC chain
  - Can be implemented using running averages
    $\mu_i^{(t+1)} = \frac{1}{t+1}(t\mu_i^{(t)} + \theta_i^{(t+1)})$
  - To predict, sample weights from distribution q
  - Works well even when simply using SGD (no Gaussian noise) to generate $\theta^{(1)}, ..., \theta^{(T)} \Rightarrow$ **SWAG Method**

Specialized Inference Techniques for Bayesian Neural Networks

Recall: Dropout Regularization

- Key idea: randomly ignore ("drop out") hidden units during each iteration of SGD with probability p

Outlook: Dropout as Variational Inference

- Dropout can be viewed as **performing variational inference*** with a particular variational family
  $q(\theta|\lambda) = \prod_j q_j(\theta_j|\lambda_j)$
  where $q_j(\theta_j|\lambda_j) = p\delta_0(\theta_j) + (1-p)\delta_{\lambda_j}(\theta_j)$
- i.e., each weight is either set to 0 with probability p or set to with probability $1-p$
- This allows to interpret the result of ordinarily training a neural network with dropout as performing approximate Bayesian inference!

Predictive Uncertainty via Dropout

- Can approximate predictive uncertainty via
  $p(y^*|\mathbf{x}^*, \mathbf{x}_{1:n}, y_{1:n}) \approx \mathbb{E}_{\theta\sim q(\cdot|\lambda)}[p(y^*|\mathbf{x}^*, \theta)] \approx \frac{1}{m}\sum_{j=1}^{m} p(y^*|\mathbf{x}^*, \theta^{(j)})$

- Hereby, each sample $\theta^{(j)}$ simply corresponds to a neural network with weights given by λ , where each unit is set to 0 with probability p
- Thus, dropout is not only performed during training, but **also during prediction**!

Probabilistic Ensembles of NNs

- Another heuristic approach for approximate Bayesian inference with Neural Networks makes use of bootstrap sampling:
- Starting with dataset $D = \{(x_1, y_1), ..., (x_n, y_n)\}$
- For $j = 1 : m$
    - Pick a data set D_j of n points uniformly at random from D with replacement
    - Obtain MAP estimate (e.g., with SGD) on D_j to obtain parameter estimate $\theta^{(j)}$
- Use approximate posterior:
  $p(y^*|\mathbf{x}^*, \mathbf{x}_{1:n}, y_{1:n}) \approx \frac{1}{m} \sum_{j=1}^{m} p(y^*|\mathbf{x}^*, \theta^{(j)})$

Overview

- SVI / Bayes-by-Backprop
    - Optimizes ELBO via SGD (e.g., reparameterization gradients)
- Stochastic gradient MCMC techniques (SGLD, SGHMC)
    - Guaranteed to eventually converge to correct distribution
    - Need to summarize the MCMC iterates
- Monte-Carlo Dropout
    - Train model with dropout and SGD
    - Obtain predictive uncertainty via test-time dropout
- Probabilistic Ensembles
    - Train multiple models on random subsamples of the data
    - Sometimes a single model is trained with multiple "heads", each trained on different subsets of the data

Aleatoric and Epistemic Uncertainty Beyond Regression

- Standard approach for multi-class classification with NNs:
  $\mathbf{p} = softmax(\mathbf{f})$
  $p_i = \frac{exp(f_i)}{\sum_{j=1}^{c} exp(f_j)}$
  $p(y|\mathbf{x}; \theta) = p_y$
- Can explicitly model aleatoric uncertainty by **injecting learnable (Gaussian) noise** ε and using
  $\mathbf{p} = softmax(\mathbf{f} + \varepsilon)$

Evaluating Model Calibration

- Can evaluate predictive distributions on held out data
  Surpose training data $D_{train} \rightarrow$ variational posterior $q(\theta|\lambda)$
  Consider validation data $D_{val} = \{(x'_i, y'_i)_{i=1:m}\}$
  $$\begin{aligned} logP(y'_{1:m}|x'_{1:m}, x_{1:n}, y_{1:n}) &\overset{i.i.d}{=} log \int P(y'_{1:m}|x'_{1:m}, \theta)P(\theta|x_{1:n}, y_{1:n})d\theta \\ &= log \int P(y'_{1:m}|x'_{1:m}, \theta)q(\theta|\lambda)d\theta \\ &= log\mathbb{E}_{\theta\sim q_\lambda} P(y'_{1:m}|x'_{1:m}, \theta) \\ &\overset{Jensen's}{\geq} \mathbb{E}_{\theta\sim q_\lambda} logP(y'_{1:m}|x'_{1:m}, \theta) \\ &\approx \frac{1}{k}\sum_{j=1}^{k}\sum_{i=1}^{m} logP(y_i^*|x_i^*, \theta^{(j)}) \end{aligned}$$
- *Note*: $\theta^{(j)} \sim q_\lambda, \quad \sum_{i=1}^m logP(y_i^*|x_i^*, \theta^{(j)})$ is standard hold-out log-likelihood

Estimating Calibration Error

- **Partition test points into bins** according to predicted confidence values
- Then **compare accuracy** within a bin with average **confidence** within a bin
- Expected/maximum calibration error (ECE/MCE) is the average/maximum discrepancy across bins

Improving Calibration

- Can empirically **improve accuracy of calibration** via several heuristics
  - Histogram binning
  - Isotonic regression
  - Platt (temperature) scaling
  - ...

Active Learning

Lecture Notes

Why is Uncertainty Useful?

- So far, we have discussed several methods for probabilistic machine learning
- Key benefit: Modeling both **epistemic** and **aleatoric** uncertainty
- Will now discuss how to use of the uncertainty for deciding which data to collect
  - Active learning
  - Bayesian optimization

Active Learning / Experiment Design

- Suppose we've collected some data. Where should we sample to obtain most useful information?

Recall: Bayesian Learning with GPs

- Suppose $p(f) = GP(f; \mu; k)$
  and we observe $y_i) = f(\mathbf{x}_i + \varepsilon_i) \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2) \quad A = \{\mathbf{x}_1, ..., \mathbf{x}_m\}$
- Then $p(f|\mathbf{x}_1, ..., \mathbf{x}_m, y_1, ..., y_m) = GP(f; \mu', k)'$
  where
  $\mu'(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}_{x,A}(\mathbf{K}_{AA} + \sigma^2\mathbf{I})^{-1}(\mathbf{y}_A - \mu_A)$
  $k'(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - \mathbf{k}_{x,A}(\mathbf{K}_{AA} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_{x',A}$
  Note: $\mathbf{k}_{x,A} = [k(x, x_1), ...k(x, x_m)]$
- **Posterior covariance k' does not depend on $\mathbf{y}_A$**

General Strategy

- Query points whose observation **provides most useful information** about the unknown function
- How do we **quantify utility of an observation**?
- How do we find a **best set of observations to make**?

Mutual Information / Information Gain

- Given random variables X and Y, the mutual information $I(X;Y)$ quantifies how much observing Y reduces uncertainty about X, as measured by its entropy, in expectation over Y
- $I(X;Y) = H(X) - H(X|Y)$, where $H(X)$ is uncertainty before observing Y, and $H(X|Y)$ is uncertainty after observing Y.
  Mutual information is symmetric: %I(X;Y)=I(Y;X)%
- $H(X) = \mathbb{E}_{X\sim p(x)}[-log p(x)]$
  $H(X|Y) = \mathbb{E}_{Y\sim p(y)}[H(X|Y = y)]$
  $H(X) + H(Y|X) = H(X, Y)$
  E.g. $X \sim \mathcal{N}(\mu, \Sigma) \rightarrow H(X) = \frac{1}{2} ln(2\pi e)^d |\Sigma|$
- E.g. $X \sim \mathcal{N}(\mu, \Sigma), Y = X + \varepsilon, \varepsilon \sim \mathcal{N}(0, \sigma^2 I)$

$$\begin{aligned} I(X;Y) = H(Y) - H(Y|X) &= H(Y) - H(\varepsilon) \\ &= \frac{1}{2} ln(2\pi e)^d |\Sigma + \sigma^2 I| - ln(2\pi e)^d |\sigma^2 I| \\ &= \frac{1}{2} ln|I + \sigma^{-2}\Sigma| \end{aligned}$$

How do we quantify utility? Information Gain [c.f., Lindley '56]

- Set D of points to observe f at
- Find $S \subseteq D$ maximizing information gain
  $F(S) := H(f) - H(f|y_S) = I(f; y_S) = \frac{1}{2} log|I + \sigma^{-2} K_S|$
  $H(f)$ is Uncertainty of f before evaluation

$H(f|y_S)$ is Uncertainty of f after evaluation
y_S is Noisy obs. at locations S

Optimizing Mutual Information [cf Lindley '56, Shewry & Wynn '87]

- Mutual information $F(S)$ is NP-hard to optimize
- Simple strategy: **Greedy algorithm**. For $S_t = \{x_1, ..., x_t\}$
  $x_{t+1} = argmax_{x \in D} F(S_t \bigcup \{x\}) - F(S_t)$
  $= argmax_{x \in D} H(y_{S_t+x}) - H(y_{S_t+x}|f) - H(y_{S_t}) + H(y_{S_t}|f)$
  $= argmax_{x \in D} H(y_x|y_{S_t}) - H(y_x|f) \leftarrow \text{Constant for fixed noise variance}$
  $= argmax_{x \in D} \frac{1}{2} ln(2\pi e)\sigma^2_{x|S_t} - \frac{1}{2} ln(2\pi e)\sigma^2_n$
  $= argmax_{x \in D} \sigma^2_{x|S_t} \leftarrow \text{Entropy is monotonic in variance}$

Active Learning: Uncertainty Sampling

- Pick: $x_t = argmax_{x \in D} \sigma^2_{t-1}(x)$
  where $\sigma^2_{t-1}(x) := \sigma^2_{x|x_{1:t-1}}$
- How good is the resulting design?

Submodularity of Mutual Information

- Mutual information $F(S)$ is **monotone submodular**: $B = A \bigcup A_c$
  $\forall x \in D \, \forall A \subseteq B \subseteq D : F(A \bigcup \{x\}) - F(A) \geq F(B \bigcup x) - F(B)$
  Note: $F(A \bigcup \{x\}) - F(A) = H(y_x|y_A) - H(y_x|f)$, $H(\varepsilon)$ ind. of x for homoscedastic case.
  $\Rightarrow F(A \bigcup \{x\}) - F(A) \geq F(B \bigcup x) - F(B)$
  $\Leftrightarrow H(y_x|y_A) - H(y_x|f) \geq H(y_x|y_B) - H(y_x|f)$
  $\Leftrightarrow H(y_x|y_A) \geq H(y_x|y_B) = H(y_x|y_A, y_{A_c})$
  Surpose RVs S, T, U, it holds that $H(S|T) \geq H(S|T, U)$, called "information never heard"
- I.e., satisfies **diminishing returns** property
- Greedy algorithm provides **constant-factor approximation** [Nemhauser et al'78]
  $F(S_T) \geq (1 - \frac{1}{e} max_{S \subseteq D, |S| \leq T} F(S)$
- I.e., uncertainty sampling is **near-optimal**!

Failure of Uncertainty Sampling in Heteroscedastic Case

- Uncertainty sampling **fails to distinguish epistemic and aleatoric uncertainty**
- In the heteroscedastic case, most **uncertain outcomes are not necessarily most informative**
  $P(y|f, x) = \mathcal{N}(y; f(x), \sigma^2(x))$
- Natural generalization: maximize
  $x_{t+1} \in argmax_x \frac{\sigma^2_f(x)}{\sigma^2_n(x)}$

where $\sigma_f^2(x)$ is Epistemic uncertainty, and $\sigma_n^2(x)$ is Aleatoric uncertainty

Note: $I(X;Y) = \frac{1}{2}ln(2\pi e)\sigma_p^2 - \frac{1}{2}ln(2\pi e)\sigma_n^2 = \frac{1}{2}ln\frac{\sigma_f^2(x)}{\sigma_n^2(x)}$

Outlook: Other Active Learning Objectives

- Instead of entropy to quantify uncertainty, can derive alternative criteria à **Bayesian experimental design**
- For Gaussians, common choices scalarize the posterior covariance matrix in different ways
  - D-optimal design: entropy = log-determinant (= unc. samp.)
  - A-optimal design: trace
  - E-optimal design: maximal eigenvalue
  - ...
- These are typically more expensive to compute, but may offer other advantages (e.g., A-optimal design minimizes the expected Mean-Squared Error)

Active Learning for Classification

- While we focused on regression, one can apply active learning also for other settings, such as classification § Here, uncertainty sampling corresponds to selecting samples that **maximize entropy of the predicted label**
  $x_{t+1} \in argmax_x H(Y|x, x_{1:t}, y_{1:t})$
  where $H(Y|x, x_{1:t}, y_{1:t}) = -\sum_y logp(y|x, x_{1:t}, y_{1:t})$
- While often a useful heuristic, also here, most uncertain label is not necessarily most informative

Informative Sampling for Classification (BALD)

Consider Bayesian learning with prior $p(\theta)$ over model params.(e.g. weights of some NN)
$p(y|x\theta) \propto exp(f_y(x,\theta))$
pick $x_{t+1} \in argmax_x I(\theta; y_x|x_{1:t}, y_{1:t}) = H(y|x, x_{1:t}, y_{1:t}) - \mathbb{E}_{\theta\sim p(\cdot|x_{1:t},y_{1:t})} H(y|x,\theta)$
where $H(y|x, x_{1:t}, y_{1:t})$ is entropy of the perdictive distribution acc. to our (approx.) posterior,
$\mathbb{E}_{\theta\sim p(\cdot|x_{1:t},y_{1:t})} H(y|x,\theta)$ can approximate by sampling θ from posterior

Summary Active Learning

- Active learning refers to a family of approaches that aim to collect **data that maximally reduces uncertainty** about the unknown model
- For Gaussian processes and **homoscedastic** noise, uncertainty sampling is equivalent to greedily maximizing mutual information
- In general, need to account for epistemic and aleatoric uncertainty (optimize their ratio / BALD)
- Due to **submodularity**, **greedy algorithm** selects **near-optimal sets of observations**

Bayesian Optimization

Lecture Notes

Exploration—Exploitation Tradeoffs

- Numerous applications require trading experimentation (**exploration**) and optimization (**exploitation**)
- Often:
  - #alternatives >> #trials
  - experiments are noisy & expensive
  - similar alternatives have similar performance
- How can we exploit this regularity?

Bayesian Optimization [Močkus et al. ’78, Jones ‘98, …]

$x_t \rightarrow y_t = f(x_t) + \epsilon_t$

- How should we sequentially pick $x_1, ..., x_T$ to find $max_x f(x)$ with minimal samples?

Multi-armed Bandits

- How should we allocate T tokens to k “arms” to maximize our return?
- Beautiful theory on how to explore & exploit [Robins ’52, Gittins’79, Auer+ ’02, …]
- Key principle: **Optimism in the face of uncertainty**
- Very successful in applications (e.g., drug trials, scheduling, ads…)

Learning to Optimize

- **Given**: Set of possible inputs D; noisy black-box access to unknown function $f \in \mathcal{F}, f : D \rightarrow \mathbb{R}$
- **Task**: Adaptively choose inputs $x_1, ..., x_T$ from D After each selection, observe $y_t = f(x_t) + \varepsilon_t$
- **Cumulative regret**: $R_T = \sum_{t=1}^{T} r_t = \sum_{t=1}^{T}(max_{x \in D} f(x) - f(x_t))$
  r_t is information regret
  Sublinear if $R_T/T \rightarrow 0$
  implies $max_t f(x_t) \rightarrow f(x^*)$

Gaussian Process Bandit Optimization

- Goal: Pick inputs $x_1, x_2, ...$ s.t. $\frac{1}{T}\sum_{t=1}^{T}[f(x^*) - f(x_t)] \rightarrow 0$ called "average regret"
- How should we pick samples to minimize our regret?

Optimistic Bayesian Optimization with GPs

- Key idea: Focus exploration on plausible maximizers (upper confidence bound ≥ best lower bound)

Upper Confidence Sampling (GP-UCB) [use in Bandits: e.g., Lai & Robbuins ’85, Auer+ ’02, Dani+ ’08, ...]

- Pick input that maximizes upper confidence bound:
  $x_t = argmax_{x \in D} \mu_{t-1}(x) + \beta_t \sigma_{t-1}(x)$
  How should we choose β_t?
- Naturally trades off exploration and exploitation Only picks plausible maximizers

Bayesian Regret of GP-UCB

- Theorem: Assuming $f \sim GP$, if we choose β_t "correctly"
  $\frac{1}{T}\sum_{t=1}^{T}[f(x^*) - f(x_t)] = \mathcal{O}^*(\sqrt{\frac{\gamma_T}{T}})$
  where $\gamma_T = max_{|S| \leq T} I(f; y_S)$
- Key concept: “maximum information gain” γ_T determines the regret

Information Capacity of GPs

- Regret depends on how quickly we can gain information
  $\gamma_T = max_{|S| \leq T} I(f; y_S)$
- Submodularity of mutual info. yields data-dependent bounds

Info. Gain Bounds for Common Kernels

- Theorem: For the following kernels, we have:
  - Linear: $\gamma_T = \mathcal{O}(dlogT)$
  - Squared-exponential: $\gamma_T = \mathcal{O}(logT)^{d+1}$
  - Matérn with $\nu > 2$, $\gamma_T = \mathcal{O}(T^{\frac{d(d+1)}{2\nu + d(d+1)}} logT)$
- Guarantees sublinear regret / convergence

Outlook: Frequentist Regret for GP-UCB

- Theorem: assume $f \in \mathcal{H}_k$
  $\frac{1}{T}\sum_{t=1}^{T}[f(x^*) - f(x_t)] = \mathcal{O}^*(\sqrt{\frac{\beta_t \gamma_t}{T}})$
  $O(\|f\|_k^2 + \gamma_T log^3 T)$
  where $\|f\|_k^2$ is "Complexity" of f (RKHS norm), $\gamma_T = max_{|A| \leq T} I(f; y_A)$

Side note: Optimizing the Acquisition Function

- GP-UCB requires solving the problem
  $x_t = argmax_{x \in D} \mu_{t-1}(x) + \beta_t \sigma_{t-1}(x)$
- **This is generally non-convex**
- In low-D, can use Lipschitz-optimization (DIRECT, etc.)
- In high-D, can use gradient ascent (based on random initialization)

Alternative Acquisition Functions

- Besides UCB, there exist a number of alternative acquisition criteria
    - Expected Improvement à homework
    - Probability of improvement
    - Information Directed Sampling
    - **Thompson sampling**

Thompson Sampling

- At iteration t, Thompson sampling draws
  $\tilde{f} \sim P(f|x_{1:t}, y_{1:t})$
  and selects $x_{t+1} \in argmax_{x \in D} \tilde{f}(x)$
- The randomness in the realization of $\tilde{f}$ is sufficient to trade exploration and exploitation
- It is possible to establish regret bounds for Thompson sampling similar to those for UCB

Outlook: Hyperparameter Estimation

- So far, have assumed that the kernel and its parameters are known. What if we need to learn them?
- In principle can **alternate learning hyperparameters** (e.g., via marginal likelihood maximization) and **observation selection**
- In practice, there is a **specific danger of overfitting**
    - Data sets in BO / active learning are **small**
    - Data points are selected **dependent** on prior observations
- Potential solutions
    - "Being Bayesian" about the hyperparameters (i.e., placing a hyperprior on them, and marginalizing them out)
    - Occasionally simply selecting some points at random

Outlook: BO beyond GPs

- Even though we focused on GPs, the ideas generalize to **other Bayesian learning problems** (e.g., involving Bayesian neural networks)
    - For UCB, can obtain (heuristic) confidence intervals using approximate inference (variational approximation, MCMC, ensembles etc.)
    - For Thompson sampling, need to sample from the posterior distribution over models, and then optimize the sample

Markov Decision Processes

Lecture Notes

New Topic: Probabilistic Planning

- So far: Probabilistic inference in dynamical models
    - E.g.: Tracking a robot based on noisy measurements
- Next: How should we **control** the robot to accomplish some goal / perform some task?

Markov Decision Processes

- An (finite) MDP is specified by
    - A set of **states** $X = \{1, ..., n\}$
    - A set of **actions** $A = \{1, ..., m\}$
    - **Transition probabilities**
      $P(x'|x, a) = Prob(Next\ state = x'|Action\ a\ in\ state\ x)$
    - A **reward function** $r(x, a)$
      Reward can be random with mean $r(x, a)$;
      Reward may depend on x only or (x, a, x') as well.
- For now assume r and P are known!
- Want to choose actions to maximize reward

Applications of MDPs

- Robot action planning
- Elevator scheduling
- Manufactoring processes
- Network switching and routing
- Foundation for Reinforcement Learning

Planning in MDPs

- Deterministic Policy: $\pi : X \to A$
- Randomized Policy: $\pi : X \to P(A)$
- Incduces a Markov chainL $X_0, X_1, ..., X_t, ...$ with transition probabilities
  $P(X_{t+1} = x'|X_t = x) = P(x'|x, \pi(x))$
  For randomized policies: $P(X_{t+1} = x'|X_t = x) = \sum_a \pi(a|x)P(x'|x, a)$
- Expected value $J(\pi) = \mathbb{E}[r(X_0, \pi(X_0)) + \gamma r(X_1, \pi(X_1)) + \gamma^2 r(X_2, \pi(X_2)) + ...]$
  where $\gamma \in [0, 1)$ is discounted factor

Computing the Value of a Policy

- For a fixed policy define **value function**
  $V^\pi(x) = J(\pi|X_0 = x) = \mathbb{E}[\sum_{t=0}^\infty \gamma^t r(X_t, \pi(X_t))|X_0 = x]$

Recursion:

$$V^\pi(x) = J(\pi|X_0 = x) = \mathbb{E}[r(X_0, \pi(X_0)) + \sum_{t=1}^{\infty} \gamma^t r(X_t, \pi(X_t))|X_0 = x]$$

$$\overset{lin.\ of\ exp.}{=} r(x, \pi(x)) + \mathbb{E}[\sum_{t=1}^{\infty} \gamma^t r(X_t, \pi(X_t))|X_0 = x]$$

$$\overset{index\ shift}{=} r(x, \pi(x)) + \gamma \mathbb{E}_{X_{1:\infty}}[\sum_{t=0}^{\infty} \gamma^t r(X_{t+1}, \pi(X_{t+1}))|X_0 = x]$$

$$\overset{iter.\ expect.}{=} r(x, \pi(x)) + \gamma \mathbb{E}_{X_1=x'}[\mathbb{E}_{X_{2:\infty}}[\sum_{t=0}^{\infty} \gamma^t r(X_{t+1}, \pi(X_{t+1}))|X_1 = x']|X_0 = x]$$

$$\overset{def.\ outer\ expect.}{=} r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(X_{t+1}, \pi(X_{t+1}))|X_1 = x']$$

$$\overset{stationary}{=} r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t))|X_0 = x']$$

Solving for the Value of a Policy

- $V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^\pi(x')$
  $V^\pi = r^\pi + \gamma T^\pi V^\pi$
  $\Rightarrow r^\pi = (I - \gamma T^\pi) V^\pi$
  $\Rightarrow V^\pi = (I - \gamma T^\pi)^{-1} r^\pi$
- Can compute V^π exactly by solving linear system!

Fixed Point Iteration

- Can (approximately) solve the linear system via fixed point iteration:
- Initialize V_0^π (e.g., as 0)
- For $t = 1 : T$ do $V_t^\pi = r^\pi + \gamma T^\pi V_{t-1}^\pi$
  $B^\pi : \mathbb{R}^n \to \mathbb{R}^n$,$B^\pi V = r^\pi + \gamma T^\pi V \Rightarrow B^\pi V^\pi = V^\pi$
  B^π is a contraction:
  $\|B^\pi V - B^\pi V'\|_\infty = \|r^\pi + \gamma T^\pi V - r^\pi - \gamma T^\pi V'\|_\infty = \gamma \|T^\pi (V - V')\|_\infty$
  $= \gamma max_x \| \sum_{x'} P(x'|x, \pi(x))(V(x) - V'(x))\| \leq \gamma \|V - V'\|_\infty$
  $\Rightarrow \|V_t^\pi - V^\pi\|_\infty \leq \gamma^t \|V_0^\pi - V^\pi\|_\infty \leq \varepsilon$
  suffices that $t ln\gamma + ln\|V_0^\pi - V^\pi\|_\infty \leq ln\varepsilon \Rightarrow t \geq \frac{ln \frac{\|V_0^\pi - V^\pi\|_\infty}{\varepsilon}}{-ln\gamma}$
- Computational advantages, e.g., for sparse transitions

Value Functions and Policies

- Value function V^{π}
  $V^{\pi}(x) = r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^{\pi}(x')$
  Every value function induces a policy
- Greedy policy w.r.t. V
  $\pi_V(x) = argmax_a r(x, a) + \gamma \sum_{x'} P(x'|x, a) V(x')$
  Every policy induces a value function
- Theorem (Bellman):
  Policy optimal ⇔ greedy w.r.t. its induced value function!
  $V^*(x) = max_a [r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x')]$

Policy Iteration

- Start with an arbitrary (e.g., random) policy π
- Until converged do:
  - Compute value function $V^{\pi}(x)$
  - Compute greedy policy π_G w.r.t. V^{π}
  - Set $\pi \leftarrow \pi_G$
- Guaranteed to
  - Monotonically improve
  - Converge to an optimal policy π^* in $O*(n^2 m/(1-\gamma))$ iterations! [Ye '10]

Alternative Approach

- Recall (Bellman): For the optimal policy π^* it holds
  $V^*(x) = max_a [r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x')]$
- Compute V^* using **fixed point/dynamic programming**:
  $V_t(x) =$ Max. expected reward when starting in state x and world ends in t time steps
  $V_0(x) = max_a r(x, a)$
  $V_1(x) = max_a r(x, a) + \gamma \sum_{x'} P(x'|x, a) V_0(x')$
  $V_{t+1}(x) = max_a r(x, a) + \gamma \sum_{x'} P(x'|x, a) V_t(x')$

Value Iteration

- Initialize $V_0(x) = max_a r(x, a)$
- For $t = 1$ to ∞
  - For each x, a let
    $Q_t(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a) V_{t-1}(x')$
  - For each x let $V_t(x) = max_a Q_t(x, a)$
  - Break if $\|V_t - V_{t-1}\|_{\infty} = max_x |V_t(x) - V_{t-1}(x)| \leq \varepsilon$
- Then choose greedy policy w.r.t V_t
- Guaranteed to converge to ε-optimal policy!

Convergence of Value Iteration

- Main ingredient of proof: Bellman update is a **contraction**
  $B^{:}\mathbb{R} \rightarrow \mathbb{R}$, $(B^*V)(x) = max_a r(x,a) + \gamma \sum_{x'} P(x'|x,a)V_{t-1}(x')$
  Bellman's theorem: $B^*V^* = V^*$
- Consider $V, V' \in \mathbb{R}^n$
  $\|B^*V - B^*V'\|_\infty = max_x|(B^*V)(x) - (B^*V')(x)| = max_x|max_a Q(x,a) - max_{a'} Q'(x,a')|$
  $\leq max_x max_a|Q(x,a) - Q'(x,a)| = \gamma max_{x,a}|\sum_{x'} P(x'|x,a)(V(x') - V'(x')) \leq \gamma\|V - V'\|_\infty$
  $\Rightarrow \|V_t - V^*\|_\infty \leq \gamma^t\|V_0 - V^*\|_\infty$
  Note: $|max_a f(a) - max_{a'} f'(a')| \leq max_a|f(a) - f'(a)|$

Tradeoffs: Value vs Policy Iteration

- Policy iteration
  - Finds exact solution in polynomial # iterations!
  - Every iteration requires computing a value function
  - Complexity per iteration: Need to compute V^{π_t} by solving linear system.
- Value iteration
  - Finds ε-optimal solution in polynomial # ($O(ln\frac{1}{\varepsilon})$) iterations
  - Complexity per iteration: $O(nms)$ where s is # of states can be reached from (x,a)
- In practice, which works better depends on application
- Can combine ideas of both algorithms

Recap: Ways for solving MDPs

- Policy iteration:
  - Start with random policy π
  - Compute exact value function V^π (matrix inversion)
  - Select greedy policy w.r.t. V^π and iterate
- Value iteration
  - Solve Bellman equation using dynamic programming
    $V_t(x) = max_a r(x,a) + \gamma \sum_{x'} P(x'|x,a)V_{t-1}(x')$
- Linear programming

MDP = Controlled Markov Chain

- State fully observed at every time step
- Action A_t controls transition to X_{t+1}

POMDP = Controlled HMM

- Only obtain noisy observations Y_t of the hidden state X_t
- Very powerful model!
- Typically extremely intractable

POMDP = Belief-state MDP

- Key idea: POMDP as MDP with enlarged state space:
- New states: **beliefs** $P(X_t|y_{1:t})$ in the original POMDP
  $\mathcal{B} = \Delta(\{1, ..., n\}) = \{b : \{1, ..., n\} \to [0, 1], \sum_x b(x) = 1\}$
  At time t: pick action $a_t \to$ new state $X_{t+1} \sim P(\cdot|x_t, a_t) \to$ obs. $y_{t+1} \sim P(\cdot|x_{t+1})$
  At time 0: $b_1 = P(X_1) \in \Delta^n := \{b \in \mathbb{R}^n : b_i \geq 0, \sum_i b_i\}$
- Actions: Same as original MDP
- Transition model:
  - Stochastic observation:
    $P(Y_{t+1} = y|b_t, a_t) = \sum_{x,x'} b_t(x)P(x'|x, a_t)P(y|x')$
  - State update (Bayesian filtering!) Given b_t, a_t, y_{t+1}
    $b_{t+1}(x') = P(X_{t+1} = x|y_{1:t+1}) \overset{Bayesian\ filtering}{=} \frac{1}{Z} b_t(x)P(X_{t+1} = x'|X_t = x, a_t)P(y_{t+1}|x')$
- Reward function: $r(b_t, a_t) = \sum_x b_t(x)r(x, a_t)$

Solving POMDPs

- For finite horizon T, set of reachable belief states is finite (but exponential in T)
- Can calculate optimal action using dynamic programming

Approximate solutions to POMDPs

- Key idea: most belief states never reached
  - Discretize the belief space by sampling
  - **Point based methods**:
    - Point based value iteration (PBVI)
    - Point based policy iteration (PBPI)
  - May want to apply dimensionality reduction
- Alternative approach: **Policy gradients**

Policy Gradient Methods

- Assume **parametric form** of policy
  $\pi(b) = \pi(b; \theta)$
- For each parameter the policy θ induces a Markov chain
- Can compute expected reward $J(\theta)$ by sampling

- Find optimal parameters through search (gradient ascent)
  $\theta^* = argmax_\theta J(\theta)$
- Will revisit when discussing RL

Introduction to Reinforcement Learning

Lecture Notes

We will start with RL in **finite state/action spaces**, and later discuss how to scale to complex domains

Learning to Act in Unknown Environments

- Learn a mapping from (seq. of) actions to rewards
- **Credit assignment problem**: which actions got me to the large reward?

Reinforcement learning

- Agent actions change the state of the world (in contrast to supervised learning)
- Assumption: States change according to some (unknown) MDP!

Recall: Markov Decision Processes

- An (finite) MDP is specified by
  - A set of **states** $X = \{1, ..., n\}$
  - A set of **actions** $A = \{1, ..., m\}$
  - **Transition probabilities**
    $P(x'|x, a) = Prob(Next\ state = x'|Action\ a\ in\ state\ x)$
  - A **reward function** $r(x, a)$
    Reward can be random with mean $r(x, a)$;
    Reward may depend on x only or (x, a, x') as well.
- Here: Goal is to maximize $\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t)$
- Observed state transitions and rewards let you learn the underlying MDP!

Recall: Planning in MDPs

- Deterministic Policy: $\pi : X \to A$
- Randomized Policy: $\pi : X \to P(A)$
- Incduces a Markov chainL $X_0, X_1, ..., X_t, ...$ with transition probabilities
  $P(X_{t+1} = x'|X_t = x) = P(x'|x, \pi(x))$
  For randomized policies: $P(X_{t+1} = x'|X_t = x) = \sum_a \pi(a|x)P(x'|x, a)$
- Expected value $J(\pi) = \mathbb{E}[r(X_0, \pi(X_0)) + \gamma r(X_1, \pi(X_1)) + \gamma^2 r(X_2, \pi(X_2)) + ...]$
  where $\gamma \in [0, 1)$ is discounted factor

- **value function**
  $V^\pi(x) = J(\pi|X_0 = x) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t))|X_0 = x]$

Value Functions and Policies

- Value function V^π
  $V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x))V^\pi(x')$
  Every value function induces a policy
- Greedy policy w.r.t. V
  $\pi_V(x) = argmax_a r(x, a) + \gamma \sum_{x'} P(x'|x, a)V(x')$
  Every policy induces a value function
- Theorem (Bellman):
  Policy optimal ⇔ greedy w.r.t. its induced value function!
  $V^*(x) = max_a[r(x, a) + \gamma \sum_{x'} P(x'|x, a)V^*(x')]$

Reinforcement Learning

- RL is different from supervised learning
  - The data we get is not i.i.d.
  - In reinforcement learning, the data we get depends on our actions!
  - Some actions have higher rewards than others!
- Exploration—Exploitation Dilemma: Should we
  - **Explore**: gather more data to avoid missing out on a potentially large reward?
  - **Exploit**: stick with our current knowledge and build an optimal policy for the data we've seen?

Two basic approaches to RL

1. **Model-based RL**
   - Learn the MDP
     - Estimate transition probabilities $P(x’|x, a)$
     - Estimate reward function $r(x, a)$
   - Optimize policy based on estimated MDP
2. **Model-free RL**
   - Estimate the value function directly;
   - Policy gradient methods;
   - Actor-critic methods.

Off-policy vs on-policy RL

- **On-policy RL**
  - Agent has full control over which actions to pick
  - Can choose how to trade exploration and exploitation

- **Off-policy RL**
  - Agent has no control over actions, only gets observational data (e.g., demonstrations, data collected by applying a different policy, …)

Learning the MDP

- Need to estimate
  - transition probabilities $P(X_{t+1} = x' | X_t = x, A = a) = \theta_{x'|x,a}$
  - Reward function $r(X = x, A = a) = r_{x,a}$
- E.g., using maximum likelihood estimation
- **Data set**: $\tau = (x_0, a_0, r_0, x_1, a_1, r_1, ..., x_{T-1}, a_{T-1}, r_{T-1}, x_T)$
  Offen, multiple episodes $\tau^{(1)}, \tau^{(2)}, ..., \tau^{(k)}$
  $\rightarrow D = \{(x_0, a_0, r_0, x_1), (x_1, a_1, r_1, x_2), ...\}$
- Estimate **transitions**:
  $P(X_{t+1} | X_t, A) \approx \frac{Count(X_{t+1}, X_t, A)}{Count(X_t, A)}$
  where $Count(X_{t+1}, X_t, A) = |\{i : (x_i = x, a_i = a, r_i, x_{i+1} = x') \in D\}|$
- Estimate **rewards**:
  $r(x, a) \approx \frac{1}{N_{x,a}} \sum_{t: X_t = x, A_t = a} R_t$

Exploration-Exploitation Dilemma

- Always pick a random action?
  - Will eventually* correctly estimate all probabilities and rewards
  - May do **extremely poorly** in terms of rewards!
- Always pick the best action according to current knowledge?
  - Quickly get some reward
  - Can get stuck in **suboptimal action**!
- Balance exploration and exploitation (more later)

Trading Exploration and Exploitation

- ε_t greedy
  - With probability ε_t : Pick random action
  - With probability $(1 - \varepsilon_t)$: Pick best action
- If sequence ε_t satisfies Robbins Monro (RM) conditions then will converge to optimal policy with probability 1
  $\sum_t \varepsilon_t = \infty, \sum_t \varepsilon_t^2 < \infty, e.g. \varepsilon_t = \frac{1}{t}$
- Simple, often performs fairly well
- **Doesn't** quickly eliminate clearly suboptimal actions

The Rmax Algorithm [Brafman & Tennenholz '02]

- Optimism in the face of uncertainty!
  requires $r(x, a) \leq R_{max} \; \forall x, a$
- If you don't know $r(x, a)$
  - Set it to R_{max}
- If you don't know $P(x'|x, a)$
- Set $P(x^*|x, a) = 1$ where x^* is a **"fairy tale"** state:
  $P(x^*|x^*, a) = 1, \forall a$
  $r(x^* a) = R_{max}, \forall a$

Implicit Exploration Exploitation in Rmax

- Never need to explicitly choose whether we're exploring or exploiting!
- Can rule out clearly suboptimal actions very quickly

The Rmax algorithm

- Input: Starting state x_0, discount factor γ
- Initially:
  - Add fairy tale state x^* to MDP
  - Set $r(x, a) = R_{max}$ for all states x and actions a
  - Set $P(x^*|x, a) = 1$ for all states x and actions a
  - Choose optimal policy for r and P
- Repeat:
  - Execute policy π
  - For each visited state action pair x, a, update $r(x, a)$
  - Estimate transition probabilities $P(x'|x, a)$
  - If observed "enough" transitions / rewards, recompute policy π according to current model P and r

How much is "enough"?

How many samples do we need to accurately estimate **P(x'|x, a)** or **r(x,a)**?

- Hoeffding bound:
  - $Z_1, \ldots, Z_n$ i.i.d. samples with mean μ and bounded in $[0, C]$
    $P(|\mu - \frac{1}{n}\sum_{i=1}^{n} Z_i| > \varepsilon) \leq 2exp(-2n\varepsilon^2/C^2)$
- e.g. $\hat{r}(x, a) = \frac{1}{n}\sum_{i=1}^{n} r_i, C = R_{max}$
  $\Rightarrow P(|\hat{r}(x, a) - r(x, a)| > \varepsilon) \leq 2exp(-2n\varepsilon^2/R_{max}^2)$
  if we want that $P(|\hat{r}(x, a) - r(x, a)| \leq \varepsilon) \geq 1 - \delta$
  it suffices that $2exp(-2n\varepsilon^2/R_{max}^2) \leq \delta \Rightarrow n \in O(\frac{R_{max}}{\varepsilon^2} log\frac{1}{\delta})$

Exploration—Exploitation Lemma

- Theorem: Every T timesteps, w.h.p., R_{max} either
  - Obtains near-optimal reward, or
  - Visits at least one unknown state-action pair
- T is related to the mixing time of the Markov chain of the MDP induced by the optimal policy

Performance of Rmax [Brafman & Tennenholz]

- Theorem:
  With probability $1-\delta$, R_{max} will reach an ε-optimal policy in a number of steps that is polynomial in $|X|, |A|, T, 1/\varepsilon, log(1/\delta), R_{max}$

Problems of model-based RL?

- Memory required: For each $x, x' \in X$ and $a \in A$, need to store $\hat{P}(x'|x,a)$ and $\hat{r}(x,a)$
- Computation time: Need to solve est. MDP, e.g. using value/policy iteration. For R_{max}, have to do this possibly $n \cdot m$ times(i.e. when learned "enough" about (x,a) pair)

Warm-up: Value estimation

- Given any policy π, want to estimate its value function $V^\pi(x)$
  $V^\pi(x) = r(x,\pi(x)) + \gamma \sum_{x'} P(x'|x,\pi(x))V^\pi(x')$
- Suppose we follow π and obs. (x,a,r,x')
  Further, assume we know $V^\pi(x)$
  $V^\pi(x) = \mathbb{E}_{X',R}[R + \gamma V^\pi(X')|x,a]$
  This suggests the following algorithm. Init. $V_0^\pi(x)$ somehow. At step t, obs. trans. (x,a,r,x') update $V_{t+1}^\pi(x) := r + \gamma V_t^\pi(x')$
  To reduce variance: instead $V_{t+1}^\pi(x) = (1-\alpha)V_t^\pi(x) + \alpha(r + \gamma V_t^\pi(x'))$

Temporal Difference (TD)-Learning

- Follow policy pi to obtain a transition $(x,a,r,x\text{'})$
- Update value estimate using **bootstrapping**
  $V(x) \leftarrow (1-\alpha_t)V(x) + \alpha_t(r + \gamma V(x'))$
- **Theorem**: If learning rate α_t satisfies
  $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty, e.g\, \alpha_t = \frac{1}{t}$
  and all state-action pairs are chosen infinitely often, then V converges to V^π with probability 1
- How can we find the optimal policy?

Model free RL

- Recall:
  1. Optimal value function $V^*(x) \rightarrow$ policy π^*

2. For optimal value function it holds:
   $V^*(x) = max_a Q^*(x,a)$
   where $Q^*(x,a) = r(x,a) + \gamma \sum_{x'} P(x'|x,a)V^*(x')$

- Key idea: Estimate $Q*(x,a)$ directly from samples!

Q-learning

- Estimate
  $Q^*(x,a) = r(x,a) + \gamma \sum_{x'} P(x'|x,a)V^*(x')$
  $V^*(x) = max_a Q^*(x,a)$
- Surpose we
  - Have initial estimate of $Q(x,a)$
  - observe transition x, a, x' with reward r
    $Q(x,a) \leftarrow (1-\alpha_t)Q(x,a) + \alpha_t(r + \gamma max_{a'} Q(x',a'))$
- **Theorem**: If learning rate α_t satisfies
  $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty, e.g\, \alpha_t = \frac{1}{t}$
  and all state-action pairs are chosen infinitely often, then Q converges to Q^* with probability 1
- How can we trade off exploration and exploitation?

Convergence of Optimistic Q-learning [Even-dar & Mansour '02]

- Similar to R_{max}:
  Initialize $Q(x,a) = \frac{R_{max}}{1-\gamma} \prod_{t=1}^{T_{init}} (1-\alpha_t)^{-1}$
- **Theorem**: With prob. $1-\delta$, optimistic Q-learning obtains an e-optimal policy after a number of time steps that is polynomial in $|X|, |A|, 1/\varepsilon$ and $log(1/\delta)$
- At every step, greedily pick $a_t \in argmax_a Q(x_t, a)$

Properties of Q-learning

- Memory required: Leep track of $Q(x,a) \in \mathbb{R}^{n \times m}$
- Computation time: Per step: need to eval $V(x') = max_{a'} Q(x',a')$

Key challenge: Scaling Up!

- MDP and RL polynomial in $|A|$ and $|X|$. Problem in:
  - Structured domains (chess, multiagent planning, …):
    $|X|$, $|A|$ exponential in #agents, state variables, …
  - Continuous domains ($|A|$ and $|X|$ infinite)
  - POMDPs (as belief-state MDPs)
- $\rightarrow$Learning / approximating value functions (regression)

Reinforcement Learning via Function Approximation

Lecture Notes

Value Functions and Policies

- Value function V^π
  $V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^\pi(x')$
  Every value function induces a policy
- Greedy policy w.r.t. V
  $\pi_V(x) = argmax_a r(x, a) + \gamma \sum_{x'} P(x'|x, a) V(x')$
  Every policy induces a value function
- Theorem (Bellman):
  Policy optimal ⇔ greedy w.r.t. its induced value function!
  $V^*(x) = max_a [r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x')]$

Recall: value & action-value (Q) functions

- Given fixed policy π, we have:
  - Value function:
    $V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^\pi(x')$
  - Action-value (Q) function:
    $Q^\pi(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^\pi(x') = r(x, a) + \gamma \sum_{x'} P(x'|x, a) Q^\pi(x', \pi(x'))$
- For the optimal policy it holds:
  $V^*(x) = max_a [r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^*(x')]$
  $Q^*(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^\pi(x') = r(x, a) + \gamma \sum_{x'} P(x'|x, a) max_{a'} Q^*(x', a')$

Off-policy vs on-policy RL

- **On-policy RL**
  - Agent has full control over which actions to pick
  - Can choose how to trade exploration and exploitation
- **Off-policy RL**
  - Agent has no control over actions, only gets observational data (e.g., demonstrations, data collected by applying a different policy, …)

Temporal Difference (TD)-Learning

- Follow policy pi to obtain a transition (x, a, r, x'), $a = \pi(x)$
- Update value estimate using **bootstrapping**
  $\hat{V}^\pi(x) \leftarrow (1 - \alpha_t) \hat{V}^\pi(x) + \alpha_t (r + \gamma \hat{V}^\pi(x'))$

- **Theorem**: If learning rate α_t satisfies
  $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty, e.g\, \alpha_t = \frac{1}{t}$
  and all state-action pairs are chosen infinitely often, then $\hat{V}^\pi$ converges to V^π with probability 1
- TD-Learning requires a is picked by $\pi \rightarrow$ on-policy

Off-policy Value Estimation

- At state x pick action a to obtain a transition (x, a, r, x')
- Update value estimate using **bootstrapping**
  $\hat{Q}^\pi(x)(x,a) \leftarrow (1-\alpha_t)\hat{Q}^\pi(x,a) + \alpha_t(r + \gamma\hat{Q}^\pi(x', \pi(x')))$
- **Theorem**: If learning rate α_t satisfies
  $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty, e.g\, \alpha_t = \frac{1}{t}$
  and all state-action pairs are chosen infinitely often, then $\hat{Q}^\pi$ converges to Q^π with probability 1
- Action a need not be picked via $\pi \rightarrow$ off-policy possible

RL via Q-learning

- $\hat{Q}^*(x)(x,a) \leftarrow (1-\alpha_t)\hat{Q}^*(x,a) + \alpha_t(r + \gamma\hat{Q}^*(x', a'))$
- **Theorem**: If learning rate α_t satisfies
  $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty, e.g\, \alpha_t = \frac{1}{t}$
  and all state-action pairs are chosen infinitely often, then $\hat{Q}^*$ converges to Q^* with probability 1
- Action a need not be picked via $\pi \rightarrow$ off-policy possible

Key challenge: Scaling Up!

- MDP and RL polynomial in $|A|$ and $|X|$. Problem in:
  - Structured domains (chess, multiagent planning, …):
    $|X|$, $|A|$ exponential in #agents, state variables, …
  - Continuous domains ($|A|$ and $|X|$ infinite)
  - POMDPs (as belief-state MDPs)
- $\rightarrow$Learning / approximating value functions (regression)

TD-Learning as SGD

- $V^\pi(x) \leftarrow (1-\alpha_t)\hat{V}^\pi(x) + \alpha_t(r + \gamma\hat{V}^\pi(x'))$
  $\bar{l}_2(V; x, r) := \frac{1}{2}(V - r - \gamma\mathbb{E}_{x'|x,\pi(x)}\hat{V}^\pi(x'))^2$
  $\nabla_V \bar{l}_2(v; x, r) = V - r - \gamma\mathbb{E}_{x'|x,\pi(x)}\hat{V}^\pi(x')$
  obs. $x' \sim P(x'|x, \pi(x))$
  $\Rightarrow V - r - \gamma\hat{V}^\pi(x') := \delta$ TD-error, is unbiased estimate of $\nabla_V \bar{l}_2(v; x, r)$
  SGD: $V \leftarrow V - \alpha_t\delta$
  $\hat{V}^\pi(x) \leftarrow= \hat{V}^\pi(x) - \alpha_t(\hat{V}^\pi(x) - r - \gamma\hat{V}^\pi(x')) = (1-\alpha_t)\hat{V}^\pi(x) + \alpha_t(r + \gamma\hat{V}^\pi(x'))$

Can view TD-learning as SGD!

- Tabular TD-learning update rule can be viewed as an instance of **stochastic (semi-)gradient descent on the squared loss**
- $l_2(\theta; x, x', r) = \frac{1}{2}(V(x;\theta) - r - \gamma V(x';\theta_{old}))^2$
- $r + \gamma V(x';\theta_{old})$ is y label(a.k.a target)
  - Parameters are entries in value vector
  - Experience / transition data sampled on-policy
- Bootstrapping means to use "old" value estimates as labels (a.k.a. targets)
- Same insight applies to learning the (optimal) action-value function
- $\rightarrow$ path towards parametric function approximation!

Parametric value function approximation

- To scale to large state spaces, learn an **approximation** of (action) value function $V(x;\theta)\ or\ Q(x,a;\theta)$
- Examples:
  - Linear function approximation $Q(x,a;\theta) = \theta^T\phi(x,a)$ where $\phi(x,a)$ are a set of (hand-designed) features
  - **(Deep) Neural networks $\rightarrow$ Deep RL**

Recall: Deep Learning

- Fitting nested nonlinear functions (neural nets) $f(\mathbf{x};\mathbf{w}) = \varphi_l(\mathbf{W}_l\varphi_{l-1}(\mathbf{W}_{l-1}(...\varphi_1(\mathbf{W}_1\mathbf{x}))))$
- to data by (approximately) solving $\mathbf{w}^* = argmin_{\mathbf{w}} \sum_{i=1}^N l(y_i, f(\mathbf{x}_i;\mathbf{w}))$
  via **stochastic gradient descent**.
  Can obtain gradient via chain-rule (**backpropagation**)

Gradients for Q-learning with function approximation

- Example: linear function approximation $\hat{Q}(x,a;\theta) = \theta^T\phi(x,a)$
- After observing transition (x,a,r,x'), update via gradient of
  $l_2(\theta;x,a,r,x') = \frac{1}{2}(Q(x,a;\theta) - r - \gamma max_{a'}Q(x',a';\theta_{old}))^2 = \frac{1}{2}\delta^2$
  $\theta \leftarrow \theta - \alpha_t\nabla l_2(\theta;x,a,r,x')$
  $\quad = \theta - \alpha_t\delta\cdot\nabla_\theta Q(x,a;\theta)$
  $\quad = \theta - \alpha_t\delta\phi(x,a)$

Q-learning with function approximation

- Straight forward generalization of tabular Q learning to function approximation suggests online algorithm:

- Until converged
    - In state x, pick action a
    - Observe x', reward r
    - Update $\theta \leftarrow \theta - \alpha_t \delta \nabla_\theta Q(x, a; \theta)$
      where $\delta := Q(x, a; \theta) - r - \gamma max_{a'} Q(x', a'; \theta)$
- This basic algorithm is typically rather **slow**

Neural Fitted Q-iteration / DQN [Riedmiller '05, Mnih et al '15]

- To accelerate Q-learning with (neural net) function approximation:
    - use "experience replay"
        - Maintain data set D of observed transitions (x, a, x', r)
    - clone network to maintain constant "target" values across episodes
      $L(\theta) = \sum_{(x,a,r,x') \in D} (r + \gamma max_{a'} Q(x', a'; \theta^{old}) - Q(x, a; \theta))^2$

Increasing stability: Double DQN [van Hasselt et al. 2015]

- **Standard DQN**:
  $L(\theta) = \sum_{(x,a,r,x') \in D} (r + \gamma max_{a'} Q(x', a'; \theta^{old}) - Q(x, a; \theta))^2$
- Suffers from "maximization bias"
- **Double DQN**: current network for evaluating the argmax
  $L^{DDQN}(\theta) = \sum_{(x,a,r,x') \in D} (r + \gamma max_{a'} Q(x', a^*(\theta); \theta^{old}) - Q(x, a; \theta))^2$
  where $a^*(\theta) := argmax_{a'} Q(x', a'; \theta)$

Convolutional neural networks

- Convolutional neural networks are ANNs for **specialized applications** (e.g., image recognition)
- The hidden layer(s) closest to the input layer **shares parameters**: Each hidden unit only depends on all "closeby" inputs (e.g., pixels), and weights constrained
  to be identical across all units on the layer
- This reduces the number of parameters, and encourages robustness against small amounts of) translation
- The weights can still be optimized via backpropagation

Dealing with large action sets

- Q-learning implicitly defines a policy via
  $a_t = argmax_a Q(x_t, a; \theta)$
- For large / continuous action spaces, this is **intractable**

Policy search methods

- Learning a **parameterized** policy
  $\pi(x) = \pi(x;\theta)$
- For episodic tasks (i.e., can reset "agent") can compute expected reward by "rollouts" (Monte Carlo forward sampling; $\rightarrow$ "on policy")
  $\tau^{(0)}, ..., \tau^{(m)} \sim \pi_\theta \quad ; \tau^{(i)} = (x_0^{(i)}, a_0^{(i)}, v_0^{(i)}, x_1^{(i)}, ..., x_T^{(i)})$
  $r(\tau^{(i)}) = \sum_{t=1}^{T} \gamma^t r_t^{(i)} \rightarrow J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} r(\tau^{(i)})$
- $\rightarrow$ Find optimal parameters through global optimization
  $\theta^* = argmax_\theta J(\theta)$

Policy gradients

- Objective: maximize
  $J(\theta) = \mathbb{E}_{x_{0:T}, a_{0:T} \sim \pi_\theta} \sum_{t=0}^{T} \gamma^t r(x_t, a_t) = \mathbb{E}_{\tau \sim \pi_\theta} r(\tau)$
- How can we obtain gradients w.r.t. θ?

Obtaining policy gradient

- Theorem: It holds* that
  $\nabla J(\theta) = \nabla \mathbb{E}_{\tau \sim \pi_\theta} r(\tau) = \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau) \nabla log \pi_\theta(\tau)]$
- Proof:

$$\begin{aligned}\nabla J(\theta) &= \nabla \int \pi_\theta(\tau) r(\tau) d\tau \\ &= \int \nabla \pi_\theta(\tau) r(\tau) d\tau \\ &= \int r(\tau) \pi_\theta(\tau) \nabla log \pi_\theta(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau) \nabla log \pi_\theta(\tau)]\end{aligned}$$

- *Note*: $\nabla log \pi_\theta(\tau) = \frac{\nabla \pi_\theta(\tau)}{\pi_\theta(\tau)} \Rightarrow \nabla \pi_\theta(\tau) = \pi_\theta(\tau) \nabla log \pi_\theta(\tau)$

Exploiting the MDP structure

- To obtain gradients for $J(\theta)$, need to compute
  $\mathbb{E}_{\tau \sim \pi_\theta} [r(\tau) \nabla log \pi_\theta(\tau)]$
- From the MDP, we have $r(\tau) = \sum_{t=0}^{T} \gamma^t r(x_t, a_t)$
  $\pi_\theta(\tau) = P(x_0) \prod_{t=0}^{T} \pi(a_t | x_t; \theta) P(x_{t+1} | x_t, a_t)$
- Thus
  $\mathbb{E}_{\tau \sim \pi_\theta} [r(\tau) \nabla log \pi_\theta(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau) \sum_{t=0}^{T} \nabla log \pi(a_t | x_t; \theta)]$

Reducing variance

- Even though the gradients obtained via
  $\nabla J(\theta) = \nabla \mathbb{E}_{\tau \sim \pi_\theta} r(\tau) = \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau) \nabla log \pi_\theta(\tau)]$

are **unbiased**, they typically exhibit **very large variance**
- Can reduce the variance using so-called **baselines**.
- Key insight: it holds that
$\mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\nabla log\pi_\theta(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[(r(\tau)-b)\nabla log\pi_\theta(\tau)]$

Proof

- $\mathbb{E}_{\tau\sim\pi_\theta}[(r(\tau)-b)\nabla log\pi_\theta(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\nabla log\pi_\theta(\tau)] - \mathbb{E}_{\tau\sim\pi_\theta}[b\nabla log\pi_\theta(\tau)]$
- $\mathbb{E}_{\tau\sim\pi_\theta}[b\nabla log\pi_\theta(\tau)] = b\int\pi_\theta(\tau)\nabla_\theta log\pi_\theta(\tau)d\tau = b\int\nabla_\theta\pi_\theta(\tau)d\tau = b\nabla_\theta\int\pi_\theta(\tau)d\tau = 0$
- *Note*: $\nabla\pi_\theta(\tau) = \pi_\theta(\tau)\nabla log\pi_\theta(\tau)$

State-dependent baselines

- Similarly, one can show that
$\mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} r(\tau)\nabla log\pi(a_t|x_t;\theta)] = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}(r(\tau)-b(\tau_{0:t-1}))\nabla log\pi(a_t|x_t;\theta)]$
- For example, can choose $b(\tau_{0:t-1}) = \sum_{t'=0}^{t-1}\gamma^{t'}r_{t'}$
and thus $\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\gamma^t G_t\nabla log\pi(a_t|x_t;\theta)]$
where $G_t = \sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}$ is the reward to go following action a_t

REINFORCE [Williams'92]

- Input: $\pi(a|x;\theta)$
    1. Initialize policy weights θ
    2. Repeat:
        1. Generate an episode (rollout): $X_0, A_0, R_0, X_1, A_1, R_1, ..., X_T, A_T, R_T$
        2. For $t = 0, ...T$:
        Set G_t to the return from step t
        Update $\theta = \theta + \eta\gamma^t G_t\nabla_\theta log\pi(A_t|X_t;\theta)$

Further variance reduction

- Basic REINFORCE gradient estimate:
$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\gamma^t G_t\nabla log\pi(a_t|x_t;\theta)]$
- Can further reduce variance via stronger baselines
$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\gamma^t(G_t - b_t(x_t))\nabla log\pi(a_t|x_t;\theta)]$
- Example: Mean over returns
$b_t(x_t) := b_t = \frac{1}{T}\sum_{t=0}^{T-1}G_t$

Deep RL with policy gradients and actor-critic methods

Lecture Notes

Value Functions and Policies

Recall: value & action-value (Q) functions

New concept: Advantage function

- For a given policy π, can consider the advantage of playing action a in state x:
  $A^\pi(x,a) = Q^\pi(x,a) - V^\pi(x) = Q^\pi(x,a) - \mathbb{E}_{a' \sim \pi(x)} Q^\pi(x,a')$
- $\forall \pi, x$: $max_a A^\pi(x,a) \geq 0$
  π^* is optimal$\Leftrightarrow$ $\forall x, a,\ A^*(x,a) \leq 0$
  Greedy policy w.r.t. π: $\pi_G(x) = argmax_a Q^\pi(x,a) = argmax_a A^\pi(x,a)$

Temporal Difference (TD)-Learning

Off-policy Value Estimation

Model-free RL via Q-learning

Key challenge: Scaling Up!

Parametric value function approximation

- To scale to large state spaces, learn an **approximation** of (action) value function $V(x;\theta)\ or\ Q(x,a;\theta)$
- Examples:
  - Linear function approximation $Q(x,a;\theta) = \theta^T\phi(x,a)$ where $\phi(x,a)$ are a set of (hand-designed) features
  - **(Deep) Neural networks $\rightarrow$ Deep RL**
- Can update parameters by **minimizing squared loss on predicted "bootstrapped" targets** via SGD

Neural Fitted Q-iteration / DQN [Riedmiller ‘05, Mnih et al ’15]

Dealing with large action sets

Policy search methods

Exploiting the MDP structure

REINFORCE [Williams’92]

Further improvements to policy gradients

- Basic policy gradient methods are slow
- Improvements:
  - Natural gradients
  - Using value function estimates $\rightarrow$ actor critic methods
  - Regularization & constrained optimization
  - Off-policy variants
- Today:
  - Introduce basic actor critic algorithm

- Review basic ideas behind an array of modern policy gradient methods (A2C/A3C, TRPO, PPO, DDPG, TD3, SAC)

Reinterpreting score gradients

- $\nabla J_T(\theta) = \mathbb{E}_{\tau\sim\pi(\theta)}[\sum_{t=0}^{T}\gamma^t G_t \nabla log\pi(a_t|x_t;\theta)$
  $J(\theta) = \mathbb{E}_{\tau\sim\pi(\theta)}[\sum_{t=0}^{\infty}\gamma^t r_t]$;
  $\nabla J(\theta) = lim_{T\to\infty}\nabla J_T(\theta)$
  $$= \sum_{t=0}^{\infty}\mathbb{E}_{\tau}[\gamma^t G_t \nabla log\pi(a_t|x_t;\theta)]$$
  $$= \sum_{t=0}^{\infty}\mathbb{E}_{\tau_{t:\infty}}[\gamma^t G_t \nabla log\pi(a_t|x_t;\theta)]$$
  $$= \sum_{t=0}^{\infty}\mathbb{E}_{x_t,a_t}[\gamma^t \nabla log\pi(a_t|x_t;\theta)\mathbb{E}[G_t|x_t,a_t]]$$
  $$= \mathbb{E}_{\tau\sim\pi(\theta)}[\sum_{t=0}^{\infty}\gamma^t Q(x_t,a_t)\nabla log\pi(a_t|x_t;\theta)]$$
- *Note*: $\tau_{t:\infty} = (x_t, a_t, r_t, x_{t+1}, \ldots)$

Actor Critic methods

- Can use value function estimates in conjunction with policy gradient methods:
  $$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi(\theta)}[\sum_{t=0}^{\infty}\gamma^t Q(x_t,a_t;\theta_Q)\nabla log\pi(a_t|x_t;\theta)]$$
  $$= \int \rho^\theta(x)\mathbb{E}_{a\sim\pi_\theta(x)}[Q(x,a;\theta_Q)\nabla log\pi(a|x;\theta)]dx$$
  $$= \mathbb{E}_{x\sim\rho^\theta, a\sim\pi_\theta(x)}[Q(x,a;\theta_Q)\nabla log\pi(a|x;\theta)]$$
  $$=: \mathbb{E}_{(x,a)\sim\pi_\theta}[Q(x,a;\theta_Q)\nabla log\pi(a|x;\theta)]$$
- *Note*: $\rho^\theta(x) = \sum_{t=0}^{\infty}\gamma^t p_\theta(x_t = x)$ is the discounted state occupancy measure

Actor Critic methods

- Can use value function estimates in conjunction with policy gradient methods [a.k.a. **policy gradient thm.**]:
  $\nabla J(\theta_\pi) = \mathbb{E}_{(x,a)\sim\pi_\theta}[Q(x,a;\theta_Q)\nabla log\pi(a|x;\theta)]$
- Allows application in the **online (non-episodic)** setting
- At time t, upon observing a transition (x, a, r, x'), update:
  $\theta_\pi \leftarrow \theta_\pi + \eta_t Q(x,a;\theta_Q)\nabla log\pi(a|x;\theta)$
  $\theta_Q \leftarrow \theta_Q - \eta_t(Q(x,a;\theta_Q) - r - \gamma Q(x', \pi(x';\theta_\pi);\theta_Q))\nabla Q(x,a;\theta_Q)$
- Under “**compatibility conditions**” guaranteed to improve

Outlook: Variance reduction via baselines

- Can improve convergence performance via variance reducing baselines (as in REINFORCE)
  $\theta_\pi \leftarrow \theta_\pi + \eta_t [Q(x, a; \theta_Q) - V(x; \theta_V)] \nabla log \pi(a|x; \theta)$
  where $Q(x, a; \theta_Q) - V(x; \theta_V)$ is Advantage function estimate $\rightarrow$ A2C algorithm
- This technique can be combined with Monte-Carlo Return estimation (blending between REINFORCE and actor critic methods $\rightarrow$ GAAC algorithm)

Outlook: Efficient implementations

- Actor critic methods can be efficiently implemented in paralled $\rightarrow$ E.g., Asynchronous Advantage Actor Critic (A3C, Mnih et al)

Outlook: TRPO & PPO

- Modern variants of policy gradient / actor critic methods
- Trust-region policy optimization (TRPO) [Schulman et al '17]
  - Sequentially optimizes a sequence of surrogate problems
    $\theta_{k+1} = argmax_\theta \hat{J}(\theta_k, \theta) \; s.t. \; KL(\theta || \theta_k) \leq \delta$
    $\hat{J}(\theta_k, \theta) = \mathbb{E}_{x,a \sim \pi_{\theta_k}} [\frac{\pi(a|x;\theta)}{\pi(a|x;\theta)_k} A^{\pi_{\theta_k}}(x, a)]$
  - Guarantees monotonic improvement in $J(\theta)$
- Proximal Policy Optimization (PPO) [Schulman et al '17]
  - Heuristic variant of TRPO (uses a certain clipped surrogate)
  - effective and widely used in practice

Towards off-policy actor critic

- All algorithms discussed so far are on-policy methods
- This often causes **sample inefficiency**
- Is it possible to train policy gradient methods in an **off-policy** fashion?

Another approach to policy gradients

- Our initial motivation was intractability of $max_{a'} Q(x', a'; \theta^{old})$
  in $L(\theta) = \sum_{(x,a,r,x') \in D} (r + \gamma max_{a'} Q(x', a'; \theta^{old}) - Q(x, a; \theta))^2$
- What if we *replace the exact maximum** by a **parametrized policy**?
  $L(\theta) = \sum_{(x,a,r,x') \in D} (r + \gamma Q(x', \pi(x'; \theta_\pi); \theta_Q^{old}) - Q(x, a; \theta))^2$
- But how do we update our policy parameters θ_π?

Updating policy parameters

- We want to follow the greedy policy
  $\pi_G(x) = argmax_a Q(x, a; \theta_Q)$

- If we allow "rich enough" policies, this is equivalent* to
  $\theta_\pi^* \in argmax_\theta \mathbb{E}_{x\sim\mu}[Q(x,\pi(x;\theta);\theta_Q)]$
  where $\mu(x) > 0$ "explores all states"
- Key idea: If we use differentiable approximation $Q(\cdot;\theta_Q)$ and differentiable deterministic policy $\pi(\cdot;\theta_\pi)$ can use chain rule (backpropagation) to obtain stochastic gradients!

Computing gradients

- Objective: $\theta_\pi^* \in argmax_\theta \mathbb{E}_{x\sim\mu}[Q(x,\pi(x;\theta);\theta_Q)] = argmax_\theta J(\theta)$
  $\nabla J(\theta) = \mathbb{E}_{x\sim\mu}[\nabla_\theta Q(x,\pi(x;\theta);\theta_Q)]$
  $\Rightarrow$can compute unbiased gradient estimate by sampling $x \sim \mu$
- From the chain rule
  $\nabla_{\theta_\pi} Q(x,\pi(x;\theta_\pi);\theta_Q) = \nabla_a Q(x,a)|_{a=\pi(x;\theta_\pi)} \nabla_{\theta_\pi}\pi(x;\theta_\pi)$

Exploration

- Policy gradient methods rely on **randomized policies** for exploration
- The method we just discussed uses deterministic policies. How do we ensure **sufficient exploration**?
- Since method is off-policy, can **inject additional action noise** (e.g., Gaussian) to encourage exploration (akin to epsilon—greedy exploration)

Deep Deterministic Policy Gradients (DDPG)

- Init. θ_Q, θ_π, replay buffer $D = \{\}$; $\theta_Q^{old} = \theta_Q$; $\theta_\pi^{old} = \theta_\pi$
- Repeat
  - Observe state x; carry out action $a = \pi(x;\theta_\pi) + \varepsilon$
  - Execute action a; observe reward r and next state x'
  - Store (x,a,r,x') in D
  - If time to update
    - For some iterations do
      - Sample mini-batch B of transitions (x,a,r,x') from D
      - For each, compute target $y = r + \gamma Q(x', \pi(x', \theta_\pi^{old}), \theta_Q^{old})$
      - $\theta_Q \leftarrow \theta_Q - \eta\nabla \frac{1}{|B|}\sum_{(x,a,r,x',y)\in B}(Q(x,a;\theta_Q) - y)^2$
      - $\theta_\pi \leftarrow \theta_\pi + \eta\nabla \frac{1}{|B|}\sum_{(x,a,r,x',y)\in B} Q(x,\pi(x;\theta_\pi);\theta_Q)$
      - $\theta_Q^{old} \leftarrow (1-\rho)\theta_Q^{old} + \rho\theta_Q;\ \theta_\pi^{old} \leftarrow (1-\rho)\theta_\pi^{old} + \rho\theta_\pi$

Outlook: Twin Delayed DDPG (TD3)

- Extends DDPG by using two critic networks, and evaluating the advantage with the smaller one ($\rightarrow$ to address **maximization bias** akin to Double-DQN)
- Applies delayed updates to actor network, which increases **stability**

Dealing with randomized policies

- In DDPG, had to inject random noise to ensure exploration
  Can we **directly allow randomized policies**?
- How about the **critic update**
  $\theta_Q \leftarrow \theta_Q - \eta \nabla \frac{1}{|B|} \sum_{(x,a,r,x',y) \in B} (Q(x,a;\theta_Q) - y)^2$
  where $y = r + \gamma Q(x', \pi(x', \theta_\pi^{old}), \theta_Q^{old})$
- For randomized policies: $(Q(x,a;\theta_Q) - y)^2 = \mathbb{E}_{a' \sim \pi}(Q(x,a;\theta_Q) - y(a'))^2$
  where we can obtain unbiased gradient estimates by sampling from $a' \sim \pi(x';\theta_\pi^{old})$
  $$\begin{aligned} \nabla_{\theta_Q} \mathbb{E}_{a' \sim \pi}(Q(x,a;\theta_Q) - y(a'))^2 &= \mathbb{E}_{a'} \nabla_{\theta_Q} (Q(x,a;\theta_Q) - y(a'))^2 \\ &:= \mathbb{E}_{a'} \nabla_{\theta_Q} \delta^2(a') \\ &= 2\delta(a') \nabla_{\theta_Q} Q(x,a;\theta_Q) \end{aligned}$$
- How about the **policy update** step?

Reparametrization gradients

- For deterministic policies, recall:
  $\nabla_{\theta_\pi} Q(x, \pi(x;\theta_\pi);\theta_Q) = \nabla_a Q(x,a)|_{a=\pi(x;\theta_\pi)} \nabla_{\theta_\pi} \pi(x;\theta_\pi)$
- Suppose policy is **reparametrizable**, i.e., $a \sim \pi(x;\theta_\pi)$ is such that the action is generated by $a = \psi(x;\theta_\pi, \epsilon)$, where ϵ is an independent random variable
- Example: Gaussian policies $a = C(x;\theta_\pi)\epsilon + \mu(x;\theta_\pi)$
  where $\epsilon \sim \mathcal{N}(0, I)$ [see variational inference lecture]
- Then $\nabla_{\theta_\pi} \mathbb{E}_{a \sim \pi_{\theta_\pi}} Q(x,a;\theta_Q) = \mathbb{E}_\epsilon \nabla_{\theta_\pi} Q(x, \psi(x;\theta_\pi,\epsilon);\theta_Q) =$
  $\mathbb{E}_\epsilon [\nabla_a Q(x,a;\theta_Q)|_{a=\psi(x;\theta_\pi,\epsilon)} \nabla_{\theta_\pi} \psi(x;\theta_\pi,\epsilon)]$
- Thus can obtain gradients for reparametrizable stochastic policies (applies beyond Gaussians)!
  Outlook: Entropy regularization
- One natural way to encourage exploration is to consider entropy regularized MDPs:
  $J_\lambda(\theta) = J(\theta) + \lambda H(\pi_\theta) = \mathbb{E}_{(x,a) \sim \pi_\theta}[r(x,a) + \lambda H(\pi(\cdot|x))]$
- Thus, use entropy of action distribution to encourage exploration
- Can suitably define regularized (action)-value functions (called “soft” value functions)
- Can use reparametrization gradients to obtain the **Soft Actor Critic (SAC)**algorithm

Overview: Policy gradient algorithms

- **On-policy** policy gradient methods
  - REINFORCE: optimizes score-gradient using Monte-Carlo returns; high variance $\rightarrow$ need baselines
  - Actor Critic methods: use value function / advantage function estimate ($\rightarrow$ A2C, A3C); implement approximate (generalized) policy iteration
  - TRPO iteratively optimizes a surrogate objective within trust region; PPO is an effective heuristic variant

- **Off-policy** policy gradient methods
  - Importance weighted variants (not discussed here)
  - DDPG: combines DQN with reparametrization policy gradients
  - TD3: extension of DDPG to avoid maximization bias
  - SAC: variant of DDPG/TD3 for entropy regularized MDPs

Model-based Deep RL

Lecture Notes

Recall: Deterministic Policy Gradients
Reparametrization gradients
Model-based Deep RL

- So far, we have focused on model-free methods
- If we have an accurate model of the environment, we can use it for **planning**
- Learning a model can help dramatically **reduce the sample complexity** compared to model-free techniques

Overview

- We first provide the high-level ideas for **planning** according to a **known dynamics** model and reward
- We then discuss how to **learn** a dynamics model
- Lastly, we discuss **exploration—exploitation** tradeoffs in the model-based setting

Planning

- There is a large literature on planning
  - **discrete and continuous** action spaces
  - **fully and partially observed** state spaces
  - with or without **constraints**
  - linear and non-linear transition models
  - ...
- Here we focus on planning in **continuous, fully observed** state spaces with **non-linear** transitions, **without constraints**

Planning with a known deterministic model

- To start, assume we have a **known deterministic** model for the reward and dynamics
  $x_{t+1} = f(x_t, a_t)$

- Then, our objective becomes
  $max_{a_{0:\infty}} \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \quad s.t.\ x_{t+1} = f(x_t, a_t)$
- Cannot explicitly optimize over an infinite horizon

Receding-horizon / Model-predictive control

- Key idea: Plan over a **finite horizon** H, carry out first action, then **replan**
  - At each iteration t, observe x_t,
  - Optimize performance over horizon H
    $max_{a_{t:t+H-1}} \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau, a_\tau) \quad s.t.\ x_{\tau+1} = f(x_\tau, a_\tau)$
  - Carry out action a_t, then replan

Solving the optimization problem

- At each iteration, need to solve
  $max_{a_{t:t+H-1}} \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau, a_\tau) \quad s.t.\ x_{\tau+1} = f(x_\tau, a_\tau)$
- For deterministic models f, x_τ is determined by $a_{t:\tau-1}$
  $x_{t+1} = f(x_t, a_t)$
  $x_{t+2} = f(x_{t+1}, a_{t+1}) = f(f(x_t, a_t), a_{t+1})$
  $\vdots$
  $x_\tau = f(f(...f(x_t, a_t), a_{t+1})..., a_{\tau-1}) =: x_\tau(a_{t:\tau-1})$
- Thus, at step t, need to maximize
  $J_H(a_{t:t+H-1}) := \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}), a_\tau)$

How to optimize?

- Need to optimize
  $J_H(a_{t:t+H-1}) := \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}), a_\tau)$
- For continuous actions, differentiable rewards and differentiable dynamics, can **analytically compute gradients** ($\rightarrow$ backpropagation through time)
- Challenges (especially for large H):
  - **Local minima**
  - **Vanishing / exploding gradients**
- $\rightarrow$ Often use heuristic global optimization methods

Outlook: Random shooting methods

- Sampling approach towards global optimization of
  $J_H(a_{t:t+H-1}) := \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}), a_\tau)$
- Generate m sets of **random samples** $a_{t:t+H-1}^{(i)}$
  - E.g., from a Gaussian distribution, cross-entropy method,...

- Pick the sequence $a_{t:t+H-1}^{(i^*)}$ that optimizes
  $i^* = argmax_{i\in\{1,...,m\}} J_H(a_{t:t+H-1}^{(i)})$
- Side note: Monte-Carlo Tree Search used in AlphaZero can be seen as advanced variant of a shooting method

Limitations of finite-horizon planning
Using a value estimate

- Suppose we have access to (an estimate of) the value function V. Then we can consider
  $J_H(a_{t:t+H-1}) := \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}), a_\tau) + \gamma^H V(x_{t+H})$
- For $H = 1$,
  $a_t = argmax_a J_H(a)$ is simply the **greedy policy** w.r.t. V
- Can also optimize using gradient-based or global optimization (shooting) methods
- Can obtain value estimates using off-policy estimation (as discussed earlier)

MPC for stochastic transition models?

- At each iteration t, observe x_t,
- Optimize expected performance over horizon H
  $max_{a_{t:t+H-1}} \mathbb{E}_{x_{t+1:t+H}}[\sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau + \gamma^H V(x_{t+H}) | a_{t:t+H-1}]$
- Carry out action a_t, then replan

Optimizing expected performance

- For probabilistic transition models via MPC, need to optimize
  $J_H(a_{t:t+H-1}) := \mathbb{E}_{x_{t+1:t+H}}[\sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau + \gamma^H V(x_{t+H}) | a_{t:t+H-1}]$
- Computing this expectation exactly requires solving a **high-dimensional integral**
- One common approach:
  Monte-Carlo **trajectory sampling**
- Suppose the transition model is **reparametrizable**, i.e., $x_{t+1} = f(x_t, a_t, \epsilon_t)$, where ϵ_t is **independent** of a, x
  - E.g., nonlinear dynamics with Gaussian noise
    *In this case, x_τ is determined by $a_{t:\tau-1}$ and $\epsilon_{t:\tau-1}$ via
    $x_\tau := x_\tau(a_{t:\tau-1}, \epsilon_{t:\tau-1})$
    $:= f(f(...f(x_t, a_t, \epsilon_t), a_{t+1}, \epsilon_{t+1})..., a_{\tau-1}, \epsilon_{\tau-1})$
- $\rightarrow$ can obtain **unbiased estimates** of $J_H(a_{t:t+H-1})$ by
  $\hat{J}_H(a_{t:t+H-1}) = \frac{1}{m} \sum_{i=1:m} \sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}, \epsilon_{t:\tau-1}^{(i)}), a_\tau) + \gamma^H V(x_{t+H})$
- Optimize, e.g., via analytic gradients, or shooting methods

Using parametrized policies

- Instead of explicitly optimizing over $a_t, \ldots, a_{t+H-1}$, can also optimize over **parametrized policies** (stochastic policies possible too via reparametrization)
  $a_t = \pi(x_t, \theta)$
- The objective becomes
  $J(\theta) = \mathbb{E}_{x_0 \sim \mu}[\sum_{\tau=0:H-1} \gamma^\tau r_\tau + \gamma^H Q(x_H, \pi(x_H, \theta)) | \theta]$
- For $H = 0$, this is identical to the DDPG objective!
  $J(\theta) = \mathbb{E}_{x_0 \sim \mu}[Q(x_0, \pi(x_0, \theta))]$

Outlook: Alternative uncertainty propagation

- Instead of using Monte Carlo rollouts to evaluate a policy, there are more refined ways to approximate the expected performance
  - Moment matching ($\rightarrow$ PILCO)
  - Variational inference

What about unknown dynamics?
*So far, have assumed a known (deterministic or stochastic) transition model f and known reward r

- Natural approach if f and r are **unknown**:
  - Start with initial policy π
  - Iterate for several episodes
    - Roll out policy π to collect data
    - Learn a model for f, r (and Q) from the collected data
    - Plan a new policy π based on the estimated models

How can we learn f and r?

- Key insight: due to the Markovian structure of the MDP, observed transitions and rewards are **(conditionally) independent**
  $x_{t+1} \perp x_{1:t-1} | x_t, a_t \; ; \; r_{t+1} \perp r_{1:t-1} | x_t, a_t$
- If we don't know the dynamics & reward, can estimate them **off-policy** with **standard supervised learning techniques** from a replay buffer (data set)
  $D = \{(x_i, a_i, r_i, x_{i+1})_i\}$

Learning dynamics models f

- For continuous state spaces, learning f and r is basically a regression problem
- Each experience (x, a, r, x') provides a **labeled data point** (z, y), with $z := (x, a)$ as input and $y := x'$ rsp. r as label
- Below, we focus on **learning transition/dynamics models** f (handling unknown rewards is analogous)

- In particular, we focus on challenges related to learning **probabilistic dynamics models** for $x_{t+1} \sim f(x_t, a_t; \theta)$

Example

- Running example: **conditional Gaussian dynamics** $x_{t+1} \sim \mathcal{N}(\mu(x_t, a_t; \theta), \Sigma(x_t, a_t; \theta))$
- Represent $\Sigma(x_t, a_t; \theta)$ via lower triangular matrix $\Sigma(x_t, a_t; \theta) = C(x_t, a_t; \theta)C(x_t, a_t; \theta)^T$
- Advantage:
  - **Only needs $\frac{n(n+1)}{2}$ parameters**
  - Automatically guarantees **(semi)-definiteness**
  - Allows **reparametrization**: $x_{t+1} = \mu(x_t, a_t; \theta) + C(x_t, a_t; \theta)\epsilon \quad \text{for } \epsilon \sim \mathcal{N}(0, I)$

Learning with MAP estimation

- First approach: obtain point estimate for f via **MAP estimation** $\rightarrow$ need prior (regularizer) and likelihood
- Here, we focus on parametrizing $\mu(x, a, \theta)$ and $C(x, a, \theta)$ via a neural network
- Can obtain MAP estimate of weights $\theta = [w_{i,j}^{(k)}]$ via $\hat{\theta} = argmin_\theta - logp(\theta) - \sum_{t=1:T} logN(x_{t+1}|\mu(x_t, a_t; \theta), \Sigma(x_t, a_t; \theta))$
- Can optimize using **stochatic gradient descent**

Why MAP is not enough?

- **Key pitfall in model-based RL**:
  - When planning over multiple time-steps ($H > 1$), errors in the model estimate **compound**
  - This **compounding error is exploited** by planning algorithm (MPC, policy search)
  - This can result in **very poor performance**!
- This pitfall can be effectively remedied by **capturing uncertainty** in the estimated model, and **taking it into account in planning**
  $\rightarrow$ Separate epistemic and aleatoric uncertainty

Reminder: Bayesian learning

- Prior: $p(\theta)$
- Likelihood: $p(y_{1:n}|x_{1:n}, \theta) \prod_{i=1}^n p(y_i|x_i, \theta)$
- Posterior: $p(\theta|x_{1:n}, y_{1:n}) = \frac{1}{Z}p(\theta) \prod_{i=1}^n p(y_i|x_i, \theta)$
  where $Z = \int p(\theta) \prod_{i=1}^n p(y_i|x_i, \theta)d\theta$
- Predictions: $p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, \theta)p(\theta|x_{1:n}, y_{1:n})d\theta$

Bayesian learning of dynamics models

- Instead of obtaining a point estimate for f, we model a distribution over f. E.g., modeling f as

    - Gaussian process
    - Bayesian neural network
- Finally get to use all the (approximate) inference techniques we learnt earlier!
    - Exact inference in GPs
    - Approximate inference in BNNs via variational inference, MCMC, dropout, ensembles,...

Recall: Epistemic and aleatoric uncertainty

- Suppose we obtain posterior distribution $P(f|D)$ for $x_{t+1} \sim f(x_t, a_t)$
- Recall: we now have two forms of uncertainty
    - **Epistemic**: Uncertainty in $P(f|D)$
    - **Aleatoric**: Uncertainty in $P(x_{t+1}|f, x_t, a_t)$

Example: Conditional Gaussians

- Consider again our conditional Gaussian dynamics $x_{t+1} \sim \mathcal{N}(\mu(x_t, a_t; \theta), \Sigma(x_t, a_t; \theta))$
- Most approximate inference techniques represent our **approximate posterior distribution** via $P(x_{t+1}|f, x_t, a_t) \approx \frac{1}{M}\sum_{i=1:M} \mathcal{N}(\mu(x_t, a_t; \theta^{(i)}), \Sigma(x_t, a_t; \theta^{(i)}))$
- Hereby, the **epistemic uncertainty** is represented by the index of the mixture component i, and the **aleatoric uncertainty** by the variance within component i

Separating epistemic and aleatoric uncertainty in planning

- When planning, anticipate:
    - Dependent (consistent) behavior across t acc. to $P(f|D)$
    - Independent randomness across t acc. to $P(x_{t+1}|f, x_t, a_t)$
- Thus, our estimated expected performance becomes $\hat{J}_H(a_{t:t+H-1}) = \frac{1}{m}\sum_{i=1:m}\sum_{\tau=t:t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}, \epsilon^{(i)}_{t:\tau-1}, f^{(i)}), a_\tau) + \gamma^H V(x_{t+H})$
  where $f^{(i)} \sim P(f|D)$ and
  $x_\tau := x_\tau(a_{t:\tau-1}, \epsilon_{t:\tau-1}, f) := f(f(...f(x_t, a_t, \epsilon_t), a_{t+1}, \epsilon_{t+1})..., a_{\tau-1}, \epsilon_{\tau-1})$
  for Gaussians, $x^{(i)}_{t+1} = \mu(x^{(i)}_t, a^{(i)}_t; \theta^{(j_i)}) + C(x^{(i)}_t, a^{(i)}_t; \theta^{(j_i)})\epsilon^{(i)}_t$
  where $j_i \sim Unif(\{1, ..., m\}) \quad \epsilon^{(i)}_t \sim \mathcal{N}(0, I)$

Greedy exploitation for model-based RL

- Start with empty data $D = \{\}$; prior $P(f) = P(f|\{\})$
- Iterate for several episodes
    - Plan a new policy π to (approximately) maximize $max_\pi \mathbb{E}_{f \sim P(\cdot|D)} J(\pi, f)$
    - Roll out policy π to collect more data, add to D

    - Update posterior distribution $P(f|D)$

PETS Algorithm [Chua, Calandra, McAllister, Levine 2018]

- Uses an **ensemble of neural networks** each predicting conditional Gaussian transition distributions
- **Trajectory sampling** is used to evaluate performance
- **MPC** used for planning

How about exploration?

- A key difference between RL and classical supervised learning is that the chosen actions affect the data we learn the models from
  $\rightarrow$ **Exploration – Exploitation dilemma**
- How do we resolve this dilemma?
    - Adding **exploration noise** (e.g., Gaussian noise “dithering”)
    - **Thompson Sampling**
    - **Optimistic exploration**

Thompson Sampling

- We have already encountered Thompson / posterior sampling in context of Bayesian optimization
- The idea also applies to (model-based) RL
    - Start with empty data $D = \{\}$; prior $P(f) = P(f|\{\})$
    - Iterate for several episodes
        - Sample a model $f \sim P(f|D)$
        - Plan a new policy π to (approximately) maximize $max_\pi J(\pi, f)$
        - Roll out policy π to collect more data, add to D
        - Update posterior distribution $P(f|D)$

How about optimism?

- Optimism is a central pillar for exploration in RL
- How about the model-based setting?
- Conceptionally, can consider a set $M(D)$ of models that are **plausible** given data D
    - E.g., for conditional Gaussians $M(D) = \{f : f_i(x, a) \in \mu_i(x, a|D) \pm \beta\sigma_i(x, a|D) \forall x, a\}$

Optimistic exploration

- Start with empty data $D = \{\}$; prior $P(f) = P(f|\{\})$

- Iterate for several episodes
  - Plan a new policy π to (approximately) maximize $max_{\pi} max_{f \in M(D)} J(\pi, f)$
  - Roll out policy π to collect more data, add to D
  - Update posterior distribution $P(f|D)$
- In general, the joint maximization over π and f is **very difficult**

Optimistic Exploration in Deep Model-based RL: H-UCRL[Curi, Berkenkamp, Krause, NeurIPS 2020]

$$\pi_t^{H-UCRL} = argmax_{\pi(\cdot)} J(\tilde{f}, \pi) \quad s.t. \tilde{f}(\mathbf{s}, \mathbf{a}) = \mu_{t-1}(\mathbf{s}, \mathbf{a}) + \beta_{t-1} \Sigma_{t-1}(\mathbf{s}, \mathbf{a}) \eta(\mathbf{s}, \mathbf{a})$$

Illustration on Inverted Pendulum

Deep RL: Mujoco Half-Cheetah

- H-UCRL outperforms Greedy & Thompson sampling Stronger effect for harder exploration tasks

Action penalty effect

- Small action penalty:
  - **Unrealistic behaviors allowed**
  - Exploration easy
  - Existing approaches work fine
- Large action penalty:
  - Avoids aggressive controls
  - **Exploration hard**
  - H-UCRL still finds good policies

Outlook: Safe Exploration

- In high-stakes applications, exploration is a dangerous proposition
- Need to guarantee **safety** (avoid unsafe states)
- How can we ensure this in case of unknown models?

Planning with confidence bounds

Stylized task

Forwards-propagating uncertain, nonlinear GP dynamics [w Koller, Berkenkamp, Turchetta CDC '18]

- Thm: For conditional Gaussian dynamics, can overapproximate the reachable states w.p. $1-\delta$

Challenges with long-term action dependencies

- Can use confidence bounds for **certifying long-term safety!**

Lyapunov functions

- $x_{t+1} = f(x_t, \pi(x_t, \theta))$
- $V(x_{t+1}) < V(x_t) \quad \forall x_t \in \mathcal{V}(c) \backslash \mathcal{V}(c_0)$
  [A.M. Lyapunov 1892]

Confidence-based Lyapunov analysis [Berkenkamp, Turchetta, Schoellig, K, NeurIPS 2017]

- $Pr(V(x_{t+1}) < V(x_t) \quad \forall x_t \in \mathcal{V}(c) \backslash \mathcal{V}(c_0)) \geq 1 - \delta$
- Can also learn Lyapunov candidates via neural networks via reduction to classification
  [Richards, Berkenkamp, K, CoRL '18]

Safe learning-based MPC [Koller, Berkenkamp, Turchetta, K CDC '18,'19]

- Theorem (informally): Under suitable conditions can always guarantee that we are able to **return to the safe set**
  [c.f. Wabersich & Zeilinger '18]

Experiments [Koller, Berkenkamp, Turchetta, K CDC '18, '19]

What you need to know

- Reinforcement learning = learning in MDPs
- Need to trade off **exploration and exploitation**
  - Epsilon-greedy
  - Thompson sampling
  - Optimistic exploration (Rmax, Optimistic Q-learning, H-UCRL, ...)
- Tabular model-based vs. model-free methods
- PAC-MDP results
- Scaling up by **approximating the value function** and using **parametric policies**
- Basic ideas for model-based Deep RL
- Can use **Bayesian learning** to utilize epistemic uncertainty during exploration

You've learned a lot!
Bayesian linear regression, Gaussian processes, variational inference, MCMC,
SGLD, Gibbs sampling, Kalman Filters, bandits, Bayesian optimization, Markov
Decision processes, value iteration, policy iteration, POMDPs, TD-learning, Q-learning, DQN, actor-critic methods, model-based deep reinforcement learning, PETS, H-UCRL

Key concepts & notions

- Bayesian learning
- Learning as inference

- Epistemic vs aleatoric uncertainty
- Score- and reparametrization gradient estimators
- POMDPs as belief-state MDPs
- Optimism in the face of uncertainty

If you want to learn more

- Other Courses
  - Deep Learning
  - Statistical Learning Theory
  - Guarantees for Machine Learning
  - Optimization for Data Science
  - Reliable and Interpretable AI
  - Computational Intelligence Lab
- Conference proceedings & Journals
  - AI: AAAI, IJCAI, JAIR
  - Machine Learning: ICML, NIPS, ICLR, AISTATS, JMLR, …
  - Robotics: ICRA, IROS, RSS, CoRL, IJRR, ...
- MSc. Thesis