Summary - Probabilistic Artificial Intelligence

Interpretation of probabilities: Frequentist ($P(A)$ is relative frequency of A in repeated experiments), Bayesian ($P(A)$ is "degree of belief" that A will occur).

Bayes rule: $P(X|Y) = \frac{P(X)\,P(Y|X)}{\sum_{x=x} P(X=x)\cdot P(Y|X=x)} \rightarrow p(\theta|D) = \frac{p(D|\theta)\,p(\theta)}{\int p(D|\theta)\,p(\theta)\,d\theta}$

↳ Bayes rule via Likelihood and prior

Variance: $Var(X) = E[(X-E[X])^2] = E[X^2] - E[X]^2$

$Var(aX+bY+c) = a^2 Var(X) + b^2 Var(Y) + 2ab\,Cov(X,Y)$

$Var(aX-bY+c) = a^2 Var(X) + b^2 Var(Y) - 2ab\,Cov(X,Y)$

$Cov(X,Y) = E[(X-E[X])\cdot(Y-E[Y])]$.

PDF after change of variable: Given r.v. $Y = g(X)$, $g:\mathbb{R}\to\mathbb{R}$ diff.able injection. Then

$$f_Y(y) = f_X(g^{-1}(y))\left|\frac{d}{dy}(g^{-1}(y))\right| = f_X(g^{-1}(y))\cdot|\det D(g^{-1}(y))|$$

↳ Jacobian in higher dimensions.

Law of unconscious statistician: X r.v., $g:\mathbb{R}^d\to\mathbb{R}$ diff.able, bijective. Then

$$E[g(X)] = \int_{\mathbb{R}^d} g(x)\,f_X(x)\,dx = \sum_x g(x)\cdot p(x).$$

Law of total expectation: $E[E[X|Y]] = E[X]$.

Law of total variance: $Var(X) = E[Var(X|Y)] + Var(E[X|Y])$

Multivariate Gaussian distribution: Let $X \sim \mathcal{N}(\mu,\Sigma)$, then $(X\in\mathbb{R}^d)$

$$p(x) = \frac{1}{(2\pi)^{n/2}\sqrt{|\Sigma|}}\cdot\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

$\Rightarrow$ X,Y Gaussian are independent $\Leftrightarrow$ X,Y are uncorrelated.

Let $A=\{i_1,\dots,i_k\}\subseteq\{1,\dots,d\}$, $X=(X_1,\dots,X_d)$ r.v. For the **marginal distribution** we write $X_A = (X_{i_1},\dots,X_{i_k}) \sim \mathcal{N}(\mu_A, \Sigma_{AA})$.

$\Rightarrow$ Let $X\sim\mathcal{N}(\mu_X,\Sigma_X)$, $Y\sim\mathcal{N}(\mu_Y,\Sigma_Y)$, M a matrix. Then:

$X+Y \sim \mathcal{N}(\mu_X+\mu_Y, \Sigma_X+\Sigma_Y)$ and $MX \sim \mathcal{N}(M\cdot\mu_X, M\Sigma_X M^T)$ (→ for X,Y independent.)

Conditional Gaussian: $X=(X_1,\dots,X_d)\sim\mathcal{N}(\mu,\Sigma)$ r.v. (Gaussian random vector), $A,B\subseteq\{1,\dots,d\}$ disjoint. Then define the conditional distr.

$$X_A\,|\,(X_B = x_B) \sim \mathcal{N}(\mu_{A|B}, \Sigma_{A|B})$$

where $\mu_{A|B} = \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(x_B-\mu_B)$, $\Sigma_{A|B} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}$.

↗ independent of conditional value!!

→ For X,Y jointly Gaussian, X can be viewed as linear function of Y with added random noise. (look at distr. of $X|Y=y$).

Linear regression: Interpolate $Y = X\cdot\omega$, $X\in\mathbb{R}^{n\times d}$ (→ full rank), $Y,\omega\in\mathbb{R}^d$, ω: weights.

$\omega_{\text{least squares}} = \arg\min_\omega \|Y-X\omega\|_2^2 = (X^TX)^{-1}X^TY$

$\omega_{\text{ridge}}(\lambda) = \arg\min_\omega \|Y-X\omega\|_2^2 = (X^TX+\lambda I)^{-1}X^TY$

Maximum Likelihood Estimation: For observations $(x_1, y_1), \dots, (x_n, y_n)$ we regard the model $y_i = w^{*T} x_i + \varepsilon_i$, where $\varepsilon_i \sim \mathcal{N}(0, \sigma_n^2)$ → homoscedastic errors.
We can equivalently write: $y_i \mid x_i, w \sim \mathcal{N}(w^T x_i, \sigma_n^2)$.
Based on this likelihood, we can compute the MLE:
$\hat{w}_{MLE} = \arg\max_w \sum_i \log(P(y_i \mid x_i, w)) \; (= \hat{w}_{\text{least squares}})$

Bayesian Inference: Given the model from above, we will now assume that a (Gaussian) prior distribution of our weights $w (\sim \mathcal{N}(0, \sigma_p^2 I))$.
Looking at the conditional version of our problem, the assumption about w allows us to use Bayes-rule to compute the posterior distribution of the weights:
$$w \mid x_{1:n}, y_{1:n} \sim \mathcal{N}(\mu, \Sigma)$$
where $\mu = \sigma_n^{-2} \Sigma X^T y$, $\Sigma = (\sigma_n^{-2} X^T X + \sigma_p^{-2} I)^{-1}$
→ using MLE could give us an estimator for w.

Maximum a Posteriori Estimation: Maximizing the a posteriori probability of our weights (i.e. $p(w \mid x_{1:n}, y_{1:n})$), results in the MAP estimator: $\hat{w}_{MAP} = \arg\max_w \log(P(y_{1:n} \mid x_{1:n}, w)) + \log(p(w))$
(using Bayes: $P(w \mid x_{1:n}, y_{1:n}) = P(y_{1:n} \mid x_{1:n}, w) \cdot P(w) \cdot \text{const.}$, then taking the log)
Notice: $\hat{w}_{MAP} = \arg\min_w \|y - Xw\|_2^2 + \frac{\sigma_n^2}{\sigma_p^2} \|w\|_2^2 = \hat{w}_{ridge}\left(\frac{\sigma_n^2}{\sigma_p^2}\right)$ (last "=" true in Gaussian case only..)

To make predictions at a new point x^*, define $f^* = \hat{w}_{MAP}^T \cdot x^*$. Then our prediction y^* follows: $y^* \mid x^*, x_{1:n}, y_{1:n} \sim \mathcal{N}(f^*, \sigma_n^2)$
→ But note that $\hat{w}_{MAP}$ gives no information about the uncertainty of its values. (Problematic when unsure about the model)

Bayesian Linear Regression (BLR): Instead of selecting a single estimation $\hat{w}$, we make use of the full posterior distribution of w.
For example let $w \sim \mathcal{N}(0, I)$ (prior), and the likelihood is $y \mid X, w, \sigma_n \sim \mathcal{N}(w^T X, \sigma_n^2)$. Then the posterior is given by:
$$w \mid X, y \sim \mathcal{N}(\bar{\mu}, \bar{\Sigma}) \quad \text{where } \bar{\mu} = (X^T X + \sigma_n^2 I)^{-1} X^T y \quad \text{and } \bar{\Sigma} = (\sigma_n^{-2} X^T X + I)^{-1}.$$

To make a prediction at x^*, define $f^* = w^T x^*$, then
$f^* \mid x^*, x_{1:n}, y_{1:n} \sim \mathcal{N}(\bar{\mu}^T x^*, (x^*)^T \bar{\Sigma} x^*)$.
Accounting for the noise in the labels, we get the prediction y^*
$y^* \mid x^*, x_{1:n}, y_{1:n} \sim \mathcal{N}(\bar{\mu}^T x^*, (x^*)^T \bar{\Sigma} x^* + \sigma_n^2)$

We can write $p(y^* \mid x^*, x_{1:n}, y_{1:n}) = \int p(y^* \mid x^*, w)\, p(w \mid x_{1:n}, y_{1:n})\, dw$.
Thus BLR averages all w according to the posterior distribution.
Note: Ridge regression can be viewed as approximating the posterior by putting all its mass on $\hat{w}$.

Uncertainty: The uncertainty of a prediction y^* (i.e. its variance) is split into two parts:
$$y^* \mid X, y, x^* \sim \mathcal{N}(\bar{\mu}^T x^*, \underbrace{(x^*)^T \bar{\Sigma} x^*}_{\text{epistemic}} + \underbrace{\sigma_n^2}_{\text{aleatoric}})$$
epistemic: uncertainty about the model, due to lack of data. (uncertainty about f^*)
aleatoric: uncertainty due to irreducible noise
↳ uncertainty about y^* given f^*.

BLR hyperparameters: In BLR, we need to specify the variance of the prior σ_p^2 and of the noise σ_n^2. One simple method to choose them is: (1) Choose $\hat{\lambda} = \hat{\sigma}_n^2 / \hat{\sigma}_p^2$ via cross-validation, (2) estimate $\hat{\sigma}_n^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{w}^T x_i)^2$ from $\hat{w}_{MAP}$ and (3) solve for $\hat{\sigma}_p^2 = \hat{\sigma}_n^2 / \hat{\lambda}$.

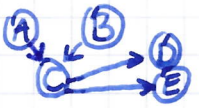
→ alternatively: use marginal likelihood (later) to find σ_p^2, σ_n^2

Graphical Models: Represent dependencies of random variables $X_1, \dots, X_n$ with a directed acyclic graph in the following way:

Let $P(X_{1:n}) = P(X_1) \cdot P(X_2 | X_1) \cdot \dots \cdot P(X_n | X_{1:n-1}) = \prod_{i=1}^{n} P(X_i | X_{S_i})$ → not all prev. are dependent

$S_i \subseteq \{1, \dots, i-1\}$.

example: $P(A,B,C,D,E) = P(A) \cdot P(B) \cdot P(C|A,B) \cdot P(D|C) \cdot P(E|C)$ ⇒

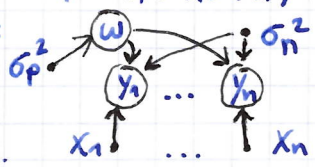


For the Graphical Model of BLR, remember:

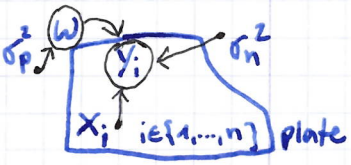
prior $p(w) = \mathcal{N}(0, \sigma_p^2 I)$, $P(y_{1:n} | x_{1:n}, w) = \prod_{i=1}^{n} P(y_i | w, x_i)$.

The joint distribution $P(w, y_{1:n} | x_{1:n}) = \underbrace{P(w | x_{1:n})}_{(P(w))} \cdot \prod_{i=1}^{n} P(y_i | x_i, w)$

defines the graph:



or "plate notation":



Recursive Bayesian updates: Given new data coming in, we can make our old posterior distribution of the weight, to be (= $P(y_i|x_i,w)$) the new prior (w.r.t. the new data).

Assume $p(w)$ is our prior, and $y_{1:n}$ observations s.t. $P(y_{1:n}|w) = \prod_{i=1}^{n} \underbrace{P_i(y_i|w)}$

Then we define $p^{(j)}(w) := P(w | y_{1:j})$ → posterior of first j observations. , $p^{(0)}(w) = p(w)$.

and recursively $p^{j}(w) = P(w | y_{1:j}) = \frac{1}{Z} p(w) P_1(y_1|w) \cdot \dots \cdot P_{j-1}(y_{j-1}|w) \cdot P_j(y_j|w)$

$=: f(\underbrace{p(w | y_{1:j-1})}_{= p^{(j-1)}(w)}, y_j)$ (for some function F)

→ Advantage: easy to compute; do not need to store all the data at once. → just update model iteratively.

Kernel Trick: Like in BLR, consider the model $f = X^T w$. Instead of considering a prior over the weights (like in BLR), we can equivalently impose a prior directly on the values of our model f, given our observations X ("function space view").

Thus: $f | X \sim \mathcal{N}(X^T E[w], X^T \mathrm{Var}(w) X^T) := \mathcal{N}(0, \sigma_p^2 \underbrace{X^T X}^T)$

→ note: COV-matrix is now $n \times n$, whereas in BLR it is $d \times d$ (#param)
→ storage cannot explode from too many parameters.

But notice that data points only enter as inner products.

This allows us the kernelize f, i.e. write $f | X \sim \mathcal{N}(0, K_{X,X})$

where $K_{X,X} = \begin{pmatrix} k(x_1,x_1) & \dots & k(x_1,x_n) \\ \vdots & & \vdots \\ k(x_n,x_1) & \dots & k(x_n,x_n) \end{pmatrix}$ with $k(x,x') = \phi(x)^T \phi(x)$.

We call k a kernel function, and this method the kernel trick.

→ The choice of kernel implicitly determines the class of functions that f is sampled from.

Gaussian Processes (GP): Extending the above idea ("function space view") to infinite domains gives us GP's: Let $(X_i)_i$ be (infinitely) many random variables, $\mathcal{X} = \{1, \dots, n, \dots\}$ their indices, $\mu: \mathcal{X} \to \mathbb{R}$ and $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ functions. We write $f \sim GP(\mu, k)$, if $\forall x \in \mathcal{X}$ $\exists$ random variable f_x s.t. $\forall A = \{x_1, \dots, x_m\} \subseteq \mathcal{X}$ it holds $f_A = [f_{x_1}, \dots, f_{x_m}] \sim \mathcal{N}(\mu_A, K_{AA})$ (jointly normal)

→ $f(x) := f_x$

where $K_{AA} = \begin{pmatrix} k(x_1,x_1) & \dots & k(x_1,x_m) \\ \vdots & & \vdots \\ k(x_m,x_1) & \dots & k(x_m,x_m) \end{pmatrix}$ and $\mu_A = \begin{pmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{pmatrix}$, μ: mean function.

↓ k: covariance (kernel) function

Like previously, the choices for k (and μ) implicitly parameterize f.
In applications we are typically interested in marginals, i.e.

$$f(x) \sim \mathcal{N}(\mu(x), k(x,x)) \quad \text{(for a specific } x\text{)}.$$

Covariance (kernel) function: For any function f, and $x, x' \in \mathbb{R}$, we define the covariance function to be

$$k(x,x') = \mathrm{Cov}(f(x), f(x'))$$

if: (1) (symmetry) $k(x,x') = k(x',x)$ $\forall x, x'$, and
(2) K_{AA} is positive-semi-definite for any $A \subseteq \mathcal{X}$.
} see notation from def. of GP.

These conditions ensure that K_{AA} is a valid covariance matrix.
Intuition: $k(x,x')$ describes how $f(x)$ and $f(x')$ are related.

Composition rules: Let k_1, k_2 be covariance functions on $\mathcal{X}$.
Then the following are valid covariance functions

(1) $k(x,x') = k_1(x,x') + k_2(x,x')$
(2) $k(x,x') = k_1(x,x') \cdot k_2(x,x')$
(3) $k(x,x') = c \cdot k_1(x,x')$ for $c > 0$.
(4) $k(x,x') = f(k_1(x,x'))$ for $f = \exp(\cdot)$, or $f =$ "polynomial with pos. coef's."

We call $k: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ **stationary** (↗aka. shift-invariant), if $k(x,x') = \tilde{k}(x - x')$ (for some fkt $\tilde{k}$)

We call $k: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ **isotropic**, if $k(x,x') = \breve{k}(\|x - x'\|_2)$ (for some $\breve{k}$) ↗fkt.
Clearly isotropic $\Rightarrow$ stationary.

Examples of Covariance functions:

- Linear kernel: $k(x,x') = x^T x'$. ($\hookrightarrow$ GP with linear kernel = BLR)
- RBF (/squared exponential/Gaussian) kernel: $k(x,x') = \exp(-\|x - x'\|_2^2 / h^2)$ ↗ smooth
  h: "bandwidth" ($\hookrightarrow$aka. length scale'), h small $\to$ f more complex, h large $\to$ f simple (less smooth)
- Exponential kernel: $k(x,x') \exp(-\|x - x'\|_2 / h)$
  h: "bandwidth", h small $\to$ f complex, h large $\to$ f simple (generally non-smooth)
  ↳continuous, but not diffable
- Matérn kernel: $k(x,x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|x - x'\|_2}{\rho} \right)^{\nu} \cdot K_\nu \left(\frac{\sqrt{2\nu}\|x - x'\|_2}{\rho} \right)$

  with parameter ν, Gamma function Γ, modified Bessel fkt K_ν and bandwidth (length scale) parameter ρ.
  Note: $\nu = 1/2 \to$ exponential kernel; $(\nu \to \infty) \to$ Gaussian (RBF) kernel.

Predictions with GP's: Let $f \sim GP(\mu, k)$, and $A = \{x_1, \dots, x_m\}$.
Given observations $Y_i = f(x_i) + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, \sigma_n^2)$.
Then our prediction at point x is distributed as follows:

$$f(x) \mid x_1, \dots, x_m, y_1, \dots, y_m \sim GP(\mu'(x), k'(x))$$ (→ etwas unsauber) →posterior.

where $\mu'(x) = \mu(x) + k_{x,A} \cdot (K_{AA} + \sigma_n^2 I)^{-1} \cdot (y_A - \mu_A)$

$k'(x,x') = k(x,x') - k_{x,A}(K_{AA} + \sigma_n^2 I)^{-1}(k_{x',A})^T$ → posterior covariance doesn't depend on y!

with $k_{x,A} = [k(x_1,x), \dots, k(x_m,x)]$ → closed form, i.e. we have again a GP.

Sample from a GP using Cholesky-decomposition: To generate samples from $x \sim \mathcal{N}(m, K)$, compute L from $K = LL^T$ (using lin. sy. of eq.), then generate $\varepsilon \sim \mathcal{N}(0, I)$, then compute $x = m + L\varepsilon$.
(since: $E[x] = m + L \cdot E[\varepsilon] = m$, $\mathrm{Var}(x) = L\,\mathrm{Var}(\varepsilon)\,L^T = LL^T = K$.).

Forward sampling: Obtain posterior sample-by-sample, i.e.
i.e. $f_1 \sim p(f_1)$, $f_2 = p(f_2 | f_1)$, $f_3 = p(f_3 | f_1, f_2)$, ...

Kalman filter as a GP: First, a Kalman filter is a dynamical system describing a moving particle: $X_{t+1} = X_t + \varepsilon_t$, $\varepsilon_t \sim \mathcal{N}(0, \sigma_x^2)$. (iid. errors)
But we only observe perturbed measurements $Y_t = X_t + \eta_t$, $\eta_t \sim \mathcal{N}(0, \sigma_y^2)$.
Then $X_t \mid y_{1:t} \sim \mathcal{N}(\mu_t, \sigma_t^2)$ and $X_{t+1} \mid y_{1:t} \sim \mathcal{N}(\mu_t, \sigma_t^2 + \sigma_x^2)$
For the Kalman gain k_{t+1} it holds that $X_{t+1} \mid y_{1:t+1} \sim \mathcal{N}(\mu_{t+1}, \sigma_{t+1}^2)$ (relevant?)
where $\mu_{t+1} = \mu_t + k_{t+1}(y_{t+1} - \mu_t)$ and $\sigma_{t+1}^2 = (1 - k_{t+1})(\sigma_x^2 + \sigma_t^2)$.
The Kalman gain k_t is a measure of how much information we gain from a new observation.
A Kalman filter can be seen as a GP: $f(t) = X_t$, $X_0 \sim \mathcal{N}(0, \sigma_0^2)$,
$X_{t+1} = X_t + \varepsilon_t$, $\varepsilon_t \sim \mathcal{N}(0, \sigma_x^2)$, then $f \sim GP(0, k_{KF})$
where $k_{KF}(t, t') = \sigma_0^2 + \sigma_x^2 \min(t, t')$. (→ Wiener Process kernel)

Optimizing Kernel hyperparameters: A common approach is to choose the parameters θ, such that the performance of our model is maximized on a validation set. But this approach collapses the uncertainty in f into a point estimate.

↓

Maximize marginal likelihood: (of the training data), is the Bayesian way of doing parameter selection. This way we optimize the effects of θ across all realizations of f, namely

$$\hat{\theta}_{MLE} = \operatorname{argmax}_\theta p(y_{1:n} \mid x_{1:n}, \theta) = \operatorname{arg\,max}_\theta \int p(y_{1:n}, f \mid x_{1:n}, \theta)\, df$$

(→ we get this by conditioning on f.)

$$= \operatorname{argmax}_\theta \int \underbrace{p(y_{1:n} \mid x_{1:n}, f, \theta)}_{(1)\text{ likelihood}} \cdot \underbrace{p(f \mid \theta)}_{(2)\text{ prior}}\, df$$

This method typically avoids overfitting (too complex θ), since this means typically that (1) is small for "most" f and large only for "few" f, while (2) is small. This does not tend to maximize the above integral. The same holds for underfitting (θ too simple), where (1) is small for "almost all" f, while (2) is large.
The "just right" complexity of θ is the case when (1) is moderately high for "many" f, and (2) also is moderate. → typically maximizes integral.

Remember that $y_{1:n} \mid x_{1:n}, \theta \sim \mathcal{N}(0, K_f(\theta) + \sigma_n^2 I)$. (K from GP definition.) Define: $K_y(\theta) = K_f(\theta) + \sigma_n^2 I$.
Then

$$\hat{\theta}_{MLE} = \operatorname{argmax}_\theta \underbrace{-\log(p(y_{1:n} \mid x_{1:n}, \theta))}_{\operatorname{argmax}_\theta} = \underbrace{\frac{1}{2} y^T K_y(\theta)^{-1} y}_{(1)} + \underbrace{\frac{1}{2}\log(\det(K_y(\theta)))}_{(2)} + \underbrace{\frac{n}{2}\log(2\pi)}_{\text{const}}$$

(1): "goodness of fit", (2): complexity of fkt's.
→ this encourages the trading between large likelihood and large prior.

The gradient of the objective function can be expressed in closed form:

$$\frac{\partial}{\partial \theta_j} \log(p(y \mid X, \theta)) = \frac{1}{2} \cdot \operatorname{tr}\left((\alpha\alpha^T - K^{-1}) \frac{\partial K}{\partial \theta_j}\right), \text{ where } \alpha = K^{-1} y,\ K = K_y(\theta).$$

→ this optimization problem is in general non-convex.

⇒ Maximizing marginal likelihood is example of an empirical Bayes method (i.e. estimating a prior distribution from data).

Alternatively, to select a model, we could additionaly place a (hyper-) prior on θ. This hyperprior could then be used to compute the MAP-estimate $\hat{\theta}_{MAP}$.

Computational issues: To learn a GP, we need to invert matrices → $\mathcal{O}(n^3)$.
Compared to BLR, which is $\mathcal{O}(nd^2)$ (d: Feature dimension).
Thus, it is worthwhile to approximate GP.
Such methods are: Exploiting parallelism, Local GP methods, Kernel function approximations, Inducing point methods.

Exploiting parallelism: The solving of linear systems in GP inference could be sped-up by using certain algorithms on multi-core (GPU) hardware. But note that this doesn't address the cubic scaling in n.

Local methods: Approximate GP by conditioning only on close observations. Notice that as points x, x' are further away, their covariance $k(x,x')$ decays. Thus, when making a prediction at point x, we only condition on the points x', where $|k(x,x')| \geq \tau$ ($\tau > 0$). But this method is still computationally quite expensive if "many" points are close by.

Kernel function approximation: Approximate GP's by approximating the kernel function. The idea is to construct a "low-dim." feature map $\phi: \mathbb{R}^d \to \mathbb{R}^m$, s.t. $k(x,x') \approx \phi(x)^T \phi(x')$. Then applying BLR has a comp. cost $\mathcal{O}(nm^2 + m^3)$.
A such method is **Random Fourier Features**. But first note that a stationary kernel (i.e. $k(x,x') = k(x-x')$) has a Fourier transform:

$$k(x-x') = \int_{\mathbb{R}^d} p(\omega) \cdot \exp(j\, \omega^T (x-x'))\, d\omega \qquad j: \text{complex unit.}$$

Theorem (Bochner): k stationary kernel is pos. semi definite $\iff p(\omega) \geq 0$ (nonnegative)

→ Thus we can scale the data s.t. $p(\omega)$ is a probability distribution. In this case we call $p(\omega)$ the spectral density of the kernel k. (In the case of the Thm.)

For stationary kernels $k(x-x') = k(t)$, the Fourier transform is given by $p(\omega) = \frac{1}{2\pi} \int_{\mathbb{R}^d} k(t) \cdot \exp(-j \cdot \omega^T t)\, dt$

Some examples:

- Gaussian $k(t) = \exp(-\|t\|_2^2/2)$, $p(\omega) = \frac{1}{(2\pi)^{d/2}} \cdot \exp(-\|t\|_2^2/2)$
- Exponential: $k(t) = \exp(-\|t\|_1)$, $p(\omega) = \prod_{j=1}^d \frac{1}{\pi(1+\omega_j^2)}$
- Cauchy: $k(t) = \prod_{j=1}^d 2/(1+t_j^2)$, $p(\omega) = \exp(-\|\omega\|_1)$.

Key idea of Random Fourier Features (RFF): interpret kernel k as an expectation:

$$k(x-x') = \int_{\mathbb{R}^d} p(\omega) \cdot \exp(i\, \omega^T (x-x'))\, d\omega = E_{\omega \sim p,\, b \sim \text{Unif}([0,2\pi])}[z_{\omega,b}(x) \cdot z_{\omega,b}(x')]$$

(spectral density)

where $z_{\omega,b}(x) = \sqrt{2} \cdot \cos(\omega^T x + b) \approx \frac{1}{m} \sum_{i=1}^m z_{\omega^{(i)},b^{(i)}}(x) \cdot z_{\omega^{(i)},b^{(i)}}(x')$

Random Fourier Features (RFF): Let $k(x,x') = k(x-x')$ be a pos. def. stationary kernel. RFF constructs $z: \mathbb{R}^d \to \mathbb{R}^m$ s.t. $k(x,x') \approx z(x)^T z(x')$.

(1) Construct Fourier Transform of the kernel:

$$p(\omega) = \frac{1}{2\pi} \int_{\mathbb{R}^d} k(t) \cdot \exp(-j\, \omega^T t)\, dt$$

(2) Randomly sample $\omega_1, \dots, \omega_m \sim p$ (spectral density), and $b_1, \dots, b_m \sim \text{Uniform}([0, 2\pi])$

(3) Define: $z(x) := \sqrt{2/m} \cdot [\cos(\omega_1^T x + b_1) + \dots + \cos(\omega_m^T x + b_m)]$

Theorem: $M \subseteq \mathbb{R}^d$ compact. $\text{diam}(M)$ its diameter. $z(x)$ as in RFF:

$$P\left(\sup_{x,x' \in M} |z(x)^T z(x') - k(x,x')| \geq \varepsilon\right) \leq 2^8 \left(\frac{\sigma_p \cdot \text{diam}(M)}{\varepsilon}\right)^2 \cdot \exp\left(-m\varepsilon^2/4(d+2)\right) \qquad (\sigma_p^2 = E[\omega^T \omega]).$$

→ BLR with feature map z approximates a GP with stationary kernel.
→ Error probability decays exponentially fast. (→ in the dimension of the Fourier Feature space.)

Problem: Fourier Features can be wasteful, since they approximate the kernel function uniformly well (across all x). But we only need accurate approx. for the points in training/test data.

Inducing points method: Approximate GP's by only considering a (random) subset of the training data during learning.

One subsampling method is the inducing points method. The idea is to summarize the training data around so-called inducing points.

Write $U = \{\bar{x}_1, \dots, \bar{x}_k\}$ (treat inducing points as hyperparam.).

Let $f = [f(x_1), \dots, f(x_n)]^T$ be the evaluations of our training data X, $f^* = f(x^*)$ the prediction at some evaluation point $x^* \in \mathcal{X}$.

Further denote with $u = [f(\bar{x}_1), \dots, f(\bar{x}_k)]^T \in \mathbb{R}^k$ the predictions of our model f at the inducing points U.

Then our joint prior is given by → by using marginalization.

$$p(f, f^*) = \int_{\mathbb{R}^k} p(f, f^*, u)\,du = \int_{\mathbb{R}^k} p(f, f^* | u) \cdot p(u)\,du$$

$$\approx \int_{\mathbb{R}^k} p(f^*|u) \cdot p(f|u) \cdot p(u)\,du.$$

The key idea is to approximate the joint prior, by assuming that f and f^* are conditionally independent given u. We call

"Training conditional": $p(f|u) = \mathcal{N}(K_{f,u} \cdot K_{u,u}^{-1} \cdot u,\ K_{ff} - Q_{f,f})$

"Testing conditional": $p(f^*|u) = \mathcal{N}(K_{f^*,u} \cdot K_{u,u}^{-1} \cdot u,\ K_{f^*,f^*} - Q_{f^*,f^*})$

where $Q_{a,b} \equiv K_{a,u} \cdot K_{u,u}^{-1} \cdot K_{u,b}$.

→ intuitively: K_{AA} (prior cov.), Q_{AA} (cov. "explained" by inducing points)

→ But computing this covariance matrix is expensive. Two approximations to the covariance of the training/testing conditional are Subset of Regressors (SoR), and Fully indep. training cond. (FITC).

Subset of Regressors (SoR): SoR approximation simply leaves out the covariance of the training/testing conditionals:

$$q_{SoR}(f|u) = \mathcal{N}(K_{f,u} K_{u,u}^{-1} \cdot u, 0) \text{ and } q_{SoR}(f^*|u) = \mathcal{N}(K_{f^*,u} K_{u,u}^{-1} \cdot u, 0).$$

→ resulting model is degenerate GP with $k_{SoR}(x, x') = k(x, u) \cdot K_{u,u}^{-1} \cdot k(u, x')$.

→ computing costs dominated by inverting $K_{u,u}^{-1}$. Thus cubic in k (# inducing points), but linear in n (# data points).

Fully independent training conditional (FITC): FITC approximation only keeps variances, and forgets about covariance:

$$q_{FITC}(f|u) = \prod_{i=1}^{n} p(f_i|u) = \mathcal{N}(K_{f,u} K_{u,u}^{-1} \cdot u, \operatorname{diag}(K_{f,f} - Q_{f,f}))$$

$$q_{FITC}(f^*|u) = \mathcal{N}(K_{f^*,u} K_{u,u}^{-1} \cdot u, \operatorname{diag}(K_{f^*,f^*} - Q_{f^*,f^*}))$$

→ computing costs dominated by inverting $K_{u,u}^{-1}$. Thus cubic in k (# inducing points), but only linear in n (# data points)

How to pick inducing points: Choose randomly, greedily (according to e.g. variance), (random) equally spaced, or treat u as hyperparameters, and maximize marginal likelihood of the data.

→ It is most important to ensure that u is representative of the test and training (!) data.

Bayesian Learning: More generally BL works the following way:
Assume a prior $p(\theta)$ (e.g. $\mathcal{N}(0, \sigma_p^2 I)$). The likelihood is: $p(y_{1:n} | x_{1:n}, \theta) = \prod_{i=1}^n p(y_i | x_i, \theta)$. → In BLR: $\mathcal{N}(\theta^T x, \sigma_n^2)$

Then 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 are given by: $p(y^* | x^*, x_{1:n}, y_{1:n}) = \int p(y^* | x^*, \theta) \cdot p(\theta | x_{1:n}, y_{1:n}) \, d\theta$

→ For BLR and GP regression, these integrals are closed-form. (can be computed directly.)
→ In general the integrals are not closed form → need approximations!

To approximate in intractable distributions $p(\theta|y) = \frac{1}{Z} p(\theta, y)$ we will assume we can evaluate the joint distribution $p(\theta, y)$, but not the normalizer Z. General approaches focus on approximating $p(\theta|y)$ as a whole (e.g. Variational Inference, Laplace approx.)

Laplace Approximation: We approximate the posterior $p(\theta | x_{1:n}, y_{1:n}) \approx q(\theta)$ using a Gaussian approximation (i.e. 2nd-order Taylor expansion) of the log-posterior around its mode $\hat{\theta}$. Then the L-approx. is:

$$q(\theta) = \mathcal{N}(\hat{\theta}, \Lambda^{-1}) \quad \text{(gaussian)} \quad \text{where} \quad \hat{\theta} = \operatorname{argmax}_\theta p(\theta|y), \; \Lambda = -\nabla\nabla \log(p(\hat{\theta}|y))$$

→ But this method can lead to poor (e.g. overconfident) approximations, since it first greedily seeks the mode, and then matches the curvature (arbitrarily bad, compared to the true posterior). (→ approx. of a Gaussian is exact)

Example: Bayesian logistic regression. Prior $p(w) = \mathcal{N}(0, \sigma_p^2 I)$, and likelihood $p(y_{1:n} | x_{1:n}, w) = \text{Bernoulli}(\sigma(w^T x))$, with $\sigma(z) = 1/(1+\exp(-z)) \in (0,1)$ the logistic function. To find the approximate posterior, first compute the mode: (→ doesn't matter for optimization)

$$\hat{w} = \operatorname{argmax}_w p(w | y_{1:n}) = \operatorname{argmax}_w \frac{1}{Z} p(w) \, p(y_{1:n} | w)$$

take log $\Rightarrow$

$$\hat{w} = \operatorname{argmin}_w \underbrace{\frac{1}{2\sigma_p^2}}_{=:\lambda} \|w\|_2^2 + \sum_{i=1}^n \underbrace{\log(1 + \exp(-y_i w^T x_i))}_{\text{"logistic loss": } \ell_{\log}(w, x_i, y_i)}$$

→ Apply SGD to find $\hat{w}$, since the above term is simply standard log. regression.
SGD recap: minimize $L(\theta) = E_{x \sim p}[\ell(\theta, x)] = \frac{1}{n} \sum_{i=1}^n \ell(\theta, x_i)$. (1) initialize θ_1, (2) draw from mini-batch $B = \{x_1, \dots, x_m\}$, $x_i \sim p$ (data), (3) update $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{m} \sum_{i=1}^m \nabla_\theta \ell(\theta_t, x_i)$.
In our case: (1) initialize w, (2) pick point from (X, y) uniformly at random, (3) compute prob. of missclassification $\hat{p}(y = -y | w, x) = 1/(1+\exp(y w^T x))$, (4) update $w_{t+1} \leftarrow w_t(1 - 2\lambda\eta_t) + \eta_t y x \, \hat{p}(y = -y | w, x)$.
Second: Compute the variance of our Laplace approximation:

$$\Lambda = -\nabla\nabla \log(p(w | x_{1:n}, y_{1:n})) = X^T \operatorname{diag}([\pi_i \cdot (1 - \pi_i)]_i) \cdot X, \quad \pi_i = \sigma(\hat{w}^T x_i)$$

→ Λ does not depend on normalization constant Z.
Lastly: For Bayesian Logistic regression, the predictive distribution for a test point x^* is given by:

$$p(y^* | x^*, x_{1:n}, y_{1:n}) \approx \int \sigma(y^* f) \cdot \mathcal{N}(f; \hat{w}^T x^*, x^{*T} \Lambda^{-1} x^*) \, df$$

Surprise: Let $u = P(A)$ for an event A. We define the surprise to be $S[u] = -\log(u)$. (note $S[u] \geq 0$).
Axiomatic characterization: (1) $S[1] = 0$, (2) $S[u] > S[v] \Rightarrow u < v$, (3) S is continuous, (4) for independent events: $S[uv] = S[u] + S[v]$.

Entropy: is a notion of uncertainty about the distribution p.

$$H[p] = E_{x\sim p}[-\log(p(x))] = -\int p(x)\log(p(x))\,dx = -\sum_x p(x)\log_2(p(x))$$

(surprise; ↳ cont. case; ↳ discrete case)

product distribution: $p(x_{1:d}) = \prod_{i=1}^d p_i(x_i) \Rightarrow H[p] = \sum_{i=1}^d H[p_i]$

example (Gaussian): $H[\mathcal{N}(\mu,\Sigma)] = d/2 \cdot \log(2\pi e) + \frac{1}{2}\log(\det(\Sigma))$ (d: dimension of gaussian)

→ $H[p]$ larger: more unsure about $X\sim p$.

Kullback-Leibler Divergence: First, define the **cross-entropy** of a distribution q relative to the distribution p.

$$H[p\|q] = E_{x\sim p}[-\log(q(x))] \; (= H[p] + KL(p\|q)) \geq H[p].$$

Intuition: measure of how close q is to the true distribution p. (closer → H lower).
Thus, the KL-divergence is defined to be

$$KL(p\|q) = H[p\|q] - H[p] = \int p(\theta)\log\left(\frac{p(\theta)}{q(\theta)}\right)d\theta = E_{\theta\sim p}\left[\log\left(\frac{p(\theta)}{q(\theta)}\right)\right]$$

→ this is a measure of distance between the distributions p and q.

Properties: (1) $KL(p\|q) \geq 0$, (2) $KL(p\|q) = 0 \Leftrightarrow p = q$ a.s.
(3) In general $KL(p\|q) \neq KL(q\|p)$ (**not** symmetric).

→ KL measures the additional expected surprise when observing samples from p, but assuming the (wrong) distribution q. (q assigns high probability to outcomes which are likely according to p) (→ minus the surprise inherent in p)

Example: $KL(q_\lambda\|p_\theta) = \frac{1}{2}\sum_{i=1}^d(\sigma_i^2 + \mu_i^2 - 1 - \log(\sigma_i^2))$, $q_\lambda \sim \mathcal{N}(\mu, \mathrm{diag}(\sigma_1^2,\dots,\sigma_d^2))$, $p \sim \mathcal{N}(0, I)$ (↳ KL for Bayesian logistic regression; ↳ prior)

Variational Inference: We approximate the posterior $p(\theta|x_{1:n}, y_{1:n}) \approx q(\theta)$ by finding a "simple" distribution that approximates p well:

$$q^* \in \operatorname{argmin}_{q\in Q} KL(q\|p) = \operatorname{argmin}_{\lambda\in\mathbb{R}^n} KL(q_\lambda\|p)$$

→ quality of approx. is hard to analyze.

Q: "variational family" (needs to be specified)

↗ "reverse KL-divergence"

Note: Let $q_1^* = \operatorname{argmin}_{q\in Q} KL(p\|q)$ and $q_2^* = \operatorname{argmin}_{q\in Q} KL(q\|p)$.
When approximating the true posterior p, using q, q_2^* tends to greedily select the mode, and underestimate the variance. (→ overconfident)
While q_1^* gives a more conservative (usually desired) estimation.

→ For Q the family of Gaussians: q^* matches 1st and 2nd moments of p.
→ Minimizing KL-divergence is equivalent to maximum likelihood estimation on an infinitely large sample size.
→ While for q^* we would prefer forward-KL, variational inference chooses backward-KL for computational reasons (MC: sample from p otherw.). But minimizing reverse-KL still recovers the true posterior a.s..

Evidence Lower Bound (ELBO): Instead of minimizing the reverse-KL in Variational inference, we can maximize the ELBO. (equivalently).

$$\operatorname{argmin}_{q\in Q} KL(q\|p(\cdot|x_{1:n}, y_{1:n})) = \operatorname{argmax}_{q\in Q} E_{\theta\sim q}[\log(p(y_{1:n}|x_{1:n},\theta))] - KL(q\|p(\cdot))$$

(prior $p(\theta)$)

$$= \operatorname{argmax}_{q\in Q} \underbrace{E_{\theta\sim q}[\log(p(y_{1:n},\theta|x_{1:n}))] + H[q]}_{=L(q,p,(x,y))}$$

We call $L(q,p,(x,y))$ the evidence lower bound (ELBO).
Because: log-evidence: $\log(p(y_{1:n}|x_{1:n})) \geq ELBO$ (→ use Jensen inequality)

→ ELBO selects q with large joint likelihood $p(y_{1:n},\theta|x_{1:n})$ (i.e. data is likely, and q is close to prior), and with large Entropy $H[q]$ (q is simple).
→ Maximizing ELBO with noninformative prior $p(\cdot) \equiv$ const. is equivalent to MLE.

Gradient of ELBO: To maximize ELBO common methods (e.g. SGD) would require an unbiased estimate of the gradient.

$$\nabla_\lambda L(q_\lambda, p; (X,Y)) = \nabla_\lambda E_{\theta\sim q_\lambda}[\log(p(Y_{1:n}|X_{1:n},\theta))] - \nabla_\lambda KL(q_\lambda \| p(\cdot))$$

(→ prior.)

While $\nabla_\lambda KL$ can easily be computed for common variational families, $\nabla_\lambda E$ is more difficult to compute. Since the expectation integrates over the measure q_λ, which depends on the variational parameters λ.
→ Thus we rewrite the gradient such that Monte Carlo sampling becomes possible. (e.g. Reparametrization trick, score gradients).
(→ unbiased estimates. → using the score fkt. from RL.)
→ maximizing ELBO using stochastic optimization is only twice as expensive (for diagonal q) as MAP inference. ($\nabla_\lambda KL$ is easy).
But we can achieve better performance: e.g. Natural gradients, Variance red techniques.

Reparameterization Trick: allows a change of "densities" s.t. the expectation above is w.r.t. ϕ, which does not depend on λ.
Let $\varepsilon \sim \phi$ a r.v. (with ϕ independent of λ). Given $g: \mathbb{R}^d \to \mathbb{R}^d$ invertible, diff.able and $\theta = g(\varepsilon, \lambda)$. Then it holds that

$$q(\theta|\lambda) = \phi(\varepsilon) \cdot |\det(\nabla_\varepsilon g(\varepsilon,\lambda))|^{-1} \quad \text{and} \quad E_{\theta\sim q_\lambda}[f(\theta)] = E_{\varepsilon\sim\phi}[f(g(\varepsilon,\lambda))]$$

→ We obtain stochastic gradients via:

$$\nabla_\lambda E_{\theta\sim q_\lambda}[f(\theta)] = E_{\varepsilon\sim\phi}[\nabla_\lambda f(g(\varepsilon,\lambda))]$$

(↓ take gradient inside.)

Example: Bayesian Logistic Regression.
Suppose we use a Gaussian variational approximation $q_\lambda(\theta) = \mathcal{N}(\mu, \Sigma)$. (→ full rank (assume))
Let $\varepsilon \sim \mathcal{N}(0, I)$, and use reparam. $\theta = g(\varepsilon,\lambda) = \Sigma^{1/2}\varepsilon + \mu$.
Then $\varepsilon = \Sigma^{-1/2}(\theta - \mu)$ and $\phi(\varepsilon) = q_\lambda(\theta) \cdot |\det(\Sigma^{1/2})|^{-1}$.
The ELBO gradient (i.e. $\nabla_\lambda E$-bit) simplifies to: (Let $C := \Sigma^{1/2}$)

$$\nabla_\lambda E_{\theta\sim q_\lambda}[\log(p(Y_{1:n}|X_{1:n},\theta))] = \nabla_{C,\mu} E_{\varepsilon\sim\mathcal{N}(0,I)}[\log(p(Y_{1:n}|X_{1:n},\theta))|_{\theta=C\varepsilon+\mu}]$$
$$\approx n \cdot \frac{1}{m}\sum_{j=1}^{m} \nabla_{C,\mu} \log(p(Y_{i_j}|X_{i_j},\theta))|_{\theta=C\varepsilon_j+\mu}$$

(→ using Monte Carlo sampling. → unbiased)

Markov Chains: A (stationary, time-homogenous) Markov Chain is a sequence of r.v.'s $(X_n)_{n\in\mathbb{N}}$ with prior density $P(X_1)$, and transition probabilities $P(X_{t+1}|X_t)$ independent of t ($X_{t+1} \perp X_{0:t-1} | X_t$: "Markov property")
As the state space $\{x_1,\dots,x_n\}$ is finite, MC can be described by the transition matrix $P = \begin{pmatrix} P(x_1|x_1) & \cdots & P(x_n|x_1) \\ \vdots & & \\ P(x_1|x_n) & \cdots & P(x_n|x_n) \end{pmatrix}$ of transition probabilities.
→ The rows of P sum to 1.
→ P describes a directed graph, which we call transition graph.

We call an MC ergodic, if $\exists t$ finite, s.t. every state can be reached from every state in exactly t steps ($\Leftrightarrow$ P^t has only strictly positive entries).
($\Leftarrow$ irreducible (can reach every state from everywhere) and aperiodic ($\forall k > N$: $p^{(k)}(x,x) > 0$) (↓ arbitrary)

→ If MC ergodic, then it has a unique stationary distribution

$$\pi(x) = \sum_{x'\in S} P(x|x')\cdot\pi(x') > 0 \quad \text{s.t.} \quad \lim_{N\to\infty} P(X_N = x) = \pi(x).$$

$\Rightarrow \pi = \pi \cdot P$ (for π a row vector) → irrespective of $P(X_1)$!

A MC satisfies the detailed balance equation for an unnormalized distribution $Q(x)$, if: $\frac{1}{Z} Q(x) \cdot P(x'|x) = \frac{1}{Z} Q(x') P(x|x')$ (↑ normalizing factor.)
$\Rightarrow$ The Markov chain has stationary distribution $\frac{1}{Z}Q(x)$.

Markov Chain Monte Carlo (MCMC): We estimate the distribution of the prediction (from Bayesian Learning) directly, by interpreting the integral as an expectation over the posterior:

→ Variational Inference approximates only the posterior.

$$p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*,\theta)\cdot p(\theta|x_{1:n}, y_{1:n})\,d\theta = E_{\theta\sim p(\cdot|x_{1:n},y_{1:n})}\left[p(y^*|x^*,\theta)\right]$$

$$\approx \frac{1}{m}\sum_{i=1}^{m} p(y^*|x^*,\theta^{(i)})$$

→ Error of approximation decreases exponentially in m.

We approximate this term using Monte Carlo sampling of i.i.d. $\theta^{(i)} \sim p(\cdot|x_{1:n}, y_{1:n})$. We sample the $\theta^{(i)}$ using MC.

↳ NOT for MCMC!! (since samples are not i.i.d.)

From the detailed balance equation it follows that we can take an unnormalized distribution and construct an MC which has the normalized distribution as stationary distributions.

→ MC are also very easy to simulate.

→ Note: samples from MC are NOT i.i.d. if from same MC)

Metropolis-Hastings Algorithm: describes a way of constructing a MC with desired stationary distribution $p(x) = \frac{1}{z}q(x)$. (normalized q).

Given a proposal distribution $r(x'|x)$, which in state x proposes state x'. Then with probability $\alpha(x'|x)$ we follow the proposal, and with probability $1-\alpha$ we stay in state x.

Here $$\alpha(x'|x) = \min\left\{1,\ p(x')\cdot r(x|x') \Big/ p(x)\cdot r(x'|x)\right\}.$$

→ can also replace p with q.

→ the resulting MC has stationary distribution $p(x) = \frac{1}{z}q(x)$.

→ Many ways to choose proposal distr. r (e.g. Gibbs Sampling).

Gibbs-Sampling: is a Metropolis-Hastings Algorithm where $\alpha(x,x') = 1$, and proposal distribution $r(x'|x)$ s.t. we first choose $i \in \{1,\dots,n\}$ uniformly at random, then set $x_{-i} = [x_1,\dots,x_{i-1},x_{i+1},\dots,x_n]$, and update x_i by sampling from $p(x_i|x_{-i})$.

→ from components of initial value $x \in \mathbb{R}^n$.

→ This satisfies the detailed balance equation (since choosing i uniformly at random ensures that the MC is ergodic).

→ finds states that are successively more likely.

→ Choosing the i in fixed order still achieves correct stationary distribution, but does not fulfill the detailed balance equation.

Generally sampling $x_i|x_{-i}$ is efficient, since $p(x_i|x_{-i}) = q(x_i|x_{-i}) / \sum_{x_i} q(x_i|x_{-i})$

→ can be efficiently evaluated

Ergodic Theorem: Let $(X_n)_{n\in\mathbb{N}}$ be an ergodic MC over a finite state space S, with stationary distribution π, and $f: S \to \mathbb{R}$.

Then $$\frac{1}{n}\sum_{i=1}^{n} f(x_i) \xrightarrow[n\to\infty]{a.s.} \sum_{x\in S} \pi(x)\cdot f(x) = E_{x\sim\pi}[f(x)], \quad (x_i \sim X_i|x_{i-1})$$

↳ like strong law of large numbers for MC.

→ Thus MCMC works (can approximate expectation), even though the samples from the MC are strongly dependent.

→ Simulating from a single MC yields an unbiased estimator.

→ No information about rate of convergence, and variance.

→ We usually discard the first samples of the MC ("burn-in time").

↳ tune how many should be discarded, and sampled in total

MCMC for continuous random variables: Generalizing MCMC for continuous r.v.

We focus on positive (so-called: Gibbs) distributions of the form

$$p(x) = \frac{1}{z}\cdot\exp(-f(x)) \qquad f: \text{"energy function"}.$$

We call p log-concave, if f is convex. (e.g. Bayesian logistic regression)

The adapted Metropolis Hastings Algorithm uses

$$\alpha(x'|x) = \min\left\{1,\ \frac{r(x|x')}{r(x'|x)}\cdot\exp(f(x)-f(x'))\right\} \quad \text{for } \alpha.$$

Normally distributed proposal: $r(x'|x) = \mathcal{N}(x'; x, \tau I)$

then $\frac{r(x|x')}{r(x'|x)} = 1 \Rightarrow \alpha = \min\{1,\ \exp(f(x)-f(x'))\}$

or LMC (Langevin Monte Carlo)

Metropolis adjusted Langevin Algorithm (MALA): describes an MCMC for continuous r.v. with improved proposal distribution:

$$r(x'|x) = \mathcal{N}(x';\ x - \tau\nabla f(x),\ 2\tau I)$$

↗ locally non-convex is also fine!

If the distr. p (to be approximated) is log-concave (e.g. Bayesian Log. regression) then MALA efficiently converges to its stationary distribution.
→ But proposal and acceptance step in MALA require full access to energy function f. → expensive for large datasets.

Stochastic Gradient Langevin Dynamics (SGLD): improves efficiency of MALA using stochastic gradient estimates, decaying step sizes and by skipping the accept/reject step. The algorithm works as follows:
sample $\theta \sim \frac{1}{z}\exp\left(\log(p(\theta)) + \sum_{i=1}^{n}\log(p(y_i|x_i,\theta))\right)$ from Bayesian posterior.
(1) Initialize θ_0. (2) For $t=0,1,\dots$ repeat $i_1,\dots,i_m \sim \text{Uniform}(\{1,\dots,n\})$, → choose mini batch. n: total sample size.
$\varepsilon_t \sim \mathcal{N}(0, 2\eta_t I)$, update $\theta_{t+1} \leftarrow \theta_t - \eta_t\left(\nabla_\theta \log(p(\theta_t)) + \frac{n}{m}\sum_{j=1}^{m}\nabla_\theta \log(p(y_{i_j}|\theta_t, x_{i_j}))\right) + \varepsilon_t$
↓ Gaussian noise.

→ Guaranteed convergence to stationary distribution for $\eta_t \in \Theta(t^{-1/3})$. → bounded above and below by $t^{-1/3}$

Using Hamiltonian Monte Carlo (HMC), performance can be improved even more by adding a momentum term to (S)GD.
→ In general we can guarantee convergence to target distribution, but for general distributions convergence/mixing may be slow.

Bayesian Neural Networks: In practice, we often get better performance in Bayesian Learning, when introducing nonlinearity: BNN's.
In BNN's we specify a prior distribution over the weights of the NN $p(\theta)$, and parameterize our likelihood functions via Neural Networks.
Example: Gaussian prior $p(\theta) = \mathcal{N}(0, \sigma_p^2 I)$, and likelihood $p(y|x,\theta) = \mathcal{N}(f(x,\theta), \sigma^2)$ where f is non-linear. → NN approximates f.
→ BNN's learn distribution over weights of the Neural Network.
→ Alternatively BNN's could handle more complex likelihoods, such as $p(y|x,\theta) = \mathcal{N}(f_\mu(x,\theta), \exp(f_{\sigma^2}(x,\theta)))$, and BNN would model both mean and (log-) variance as outputs of one NN.

MAP Estimation for BNN's: In the case of heteroscedastic error, we use the likelihood model $p(y|x,\theta) = \mathcal{N}(\mu(x,\theta), \sigma(x,\theta))$ μ,σ: Functions.
Then

$$\hat{\theta}_{MAP} = \arg\min_\theta -\log(p(\theta)) - \sum_{i=1}^{n}\log(p(y_i|x_i,\theta))$$

$$= \arg\min_\theta \lambda\cdot\|\theta\|_2^2 + \frac{1}{2}\sum_{i=1}^{n}\left[\frac{1}{\sigma(x_i,\theta)^2}\|y_i - \mu(x_i,\theta)\|_2^2 + \log(\sigma(x_i,\theta)^2)\right]$$

→ Model can explain y_i through accurate model μ, or variance σ, but is penalized for choosing large variances. Thus reduce losses by attributing some points to large variance. → "learn" aleatoric uncertainty. ↗ noise uncertainty.
→ MAP only makes point estimates → don't model epistemic uncertainty. → lack of data uncertainty.

Approximate Inference for BNNs: Learning and inference in BNNs are generally intractable (even for Gaussian prior and likelihood) when noise is not assumed to be homoscedastic and known. → Like before, the integrals from BL are intractable.
→ Can use approximate inference techniques (e.g. Laplace Approx., Black-box stochastic variational inf., SGLD) at similar cost as MAP/SGD. → automatic differentiation
→ Computing gradients is not a problem, since well-implemented in NN
→ specialized approximate inference methods for BNNs: Monte-Carlo Dropout, Probabilistic ensembles.
↓ Now we will describe many such methods for approximate inference.

Variational Inference for BNNs (Bayes by Backprop): As we have done with VI, approximate the posterior $p(\theta|x_{1:n}, y_{1:n}) \approx q(\theta)$ by maximizing the ELBO: (prior)

$$\operatorname{argmin}_{q\in Q} KL(q \| p(\cdot|x_{1:n}, y_{1:n})) = \operatorname{argmax}_{q\in Q} E_{\theta\sim q}[\log(p(y_{1:n}|x_{1:n}, \theta))] - KL(q \| p(\cdot))$$

Where we usually choose Q to be the fam. of Gaussians: $q(\theta|\lambda) = \mathcal{N}(\theta; \underbrace{\mu, \Sigma}_{=\lambda})$
To maximize, we need to compute the gradient. $\nabla_\lambda KL$ can be expressed in closed form (for Gaussians). For $\nabla_\lambda E$ we obtain unbiased gradient estimates using the reparametrization trick:

$$E_{\theta\sim q}[\log(p(y_{1:n}|x_{1:n}, \theta))] = E_{\varepsilon\sim\mathcal{N}(0,I)}[\log(p(y_{1:n}|x_{1:n}, \theta))|_{\theta=\Sigma^{1/2}\varepsilon+\mu}]$$

where $\Sigma = \operatorname{diag}(\sigma_1^2, \dots, \sigma_d^2)$, $\Sigma^{1/2} = \operatorname{diag}(\sigma_1, \dots, \sigma_d)$.
Then we use backpropagation (automatic differentiation) to compute the gradients of the Likelihood.

Inference: $p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, \theta) \cdot p(\theta|x_{1:n}, y_{1:n}) d\theta$

$$\approx E_{\theta\sim q_\lambda}[p(y^*|x^*, \theta)] \approx \frac{1}{m}\sum_{i=1}^m p(y^*|x^*, \theta^{(i)}) \quad \theta^{(i)} \sim q_\lambda \text{ i.i.d.}$$

→ monte carlo sampling

→ Intuitively, we draw NN weights according to the posterior q_λ, and then average-out their predictions.

Using the inference approximation we can get the prediction distribution (for a Gaussian likelihood):

$$p(y^*|x_{1:n}, y_{1:n}, x^*) \approx \frac{1}{m}\sum_{i=1}^m \mathcal{N}(\mu(x^*, \theta^{(i)}), \sigma^2(x^*, \theta^{(i)})).$$

Thus approx. mean: $E[y^*|x^*, x_{1:n}, y_{1:n}] \approx \frac{1}{m}\sum_{i=1}^m \mu(x^*, \theta^{(i)}) =: \bar{\mu}(x^*)$

and approx. variance: $\operatorname{Var}(y^*|x_{1:n}, y_{1:n}, x^*) = E[\operatorname{Var}(y^*|x^*, \theta)] + \operatorname{Var}(E[y^*|x^*, \theta])$

$$\approx \underbrace{\frac{1}{m}\sum_{i=1}^m \sigma^2(x^*, \theta^{(i)})}_{\text{aleatoric uncertainty}} + \underbrace{\frac{1}{m}\sum_{i=1}^m (\mu(x^*, \theta^{(i)}) - \bar{\mu}(x^*))^2}_{\text{epistemic uncertainty.}}$$

MCMC for BNNs: Like before, we can use MCMC methods to approximate the prediction directly: $p(y^*|x^*, x_{1:n}, y_{1:n}) \approx \frac{1}{N}\sum_{i=1}^N p(y^*|x^*, \theta^{(i)})$

Algorithms like SGLD or SG-HMC allow efficient computation of gradients.
→ To avoid "burn-in" period, drop first n samples
→ Typically cannot afford to store all N samples / models. Thus, we need to summarize the iterates. (i.e. combine the $\theta^{(i)}$). (weight samples of one iteration)

- **subsampling:** only keep track of m snapshots of weights $[\theta^{(1)}, \dots, \theta^{(m)}]$ (acc. to some schedule) and use them for inference.
- **Gaussian approximation:** summarize weights by approx. them with a Gaussian distribution $q(\theta|\mu_{1:d}, \sigma^2_{1:d})$ where

$$\mu_i = \frac{1}{T}\sum_{j=1}^T \theta_i^{(j)} \quad \text{and} \quad \sigma_i^2 = \frac{1}{T}\sum_{j=1}^T (\theta_i^{(j)} - \mu_i)^2$$

  → This can be implemented using running averages.
  → for prediction, simply sample weights from $q(\theta|\mu_{1:d}, \sigma^2_{1:d})$
  → **SWAG method:** use SGD (without Gaussian noise) to generate $\theta^{(1)}, \dots, \theta^{(T)}$.

Dropout as Variational Inference: In dropout regularization, the key idea is to randomly ignore ("drop out") hidden units during training (i.e. during each iteration of SGD) with probability p.
Dropout can be viewed as performing variational inference with the variational family: $q(\theta|\lambda) = \prod_j q_j(\theta_j|\lambda_j) = \prod_j (p\,\delta_0(\theta_j) + (1-p)\,\delta_{\lambda_j}(\theta_j))$

(Dirac delta → $\delta_a(x) = \begin{cases} 1 & x=a \\ 0 & x\neq a \end{cases}$)

↓ i.e. each weight θ_j is either set to 0 with prob. p, or set to λ_j with prob. $(1-p)$.

↓ (Dropout as Variational Inference)

→Thus ordinarily training a NN with dropout is equivalent to performing approximate Bayesian inference.
The approximation of the prediction from Variational inference:
$p(y^*|x^*, X_{1:n}, Y_{1:n}) \approx \frac{1}{m}\sum_{i=1}^{m} p(y^*|x^*, \theta^{(i)})$
Each sample $\theta^{(i)}$ corresponds to a NN with weights given by λ, where each unit is set to 0 with probability p.
→Thus dropout also performed during prediction! (not only training)

Probabilistic Ensembles: is another approach for approximate inference for BNNs. We know that VI for BNNs is like averaging the predictions of m NN. Thus, here, we immediately learn the weights of m NNs. For this we choose m training sets randomly, by sampling uniformly from the data, with replacement. Then we obtain m MAP estimates of the weights $\theta^{(i)}$, and approximate the prediction: (→get MAP with e.g. SGD.)
$p(y^*|x^*, X_{1:n}, Y_{1:n}) = E_{\theta\sim p(\cdot|X_{1:n},Y_{1:n})}[p(y^*|x^*,\theta)] \approx \frac{1}{m}\sum_{i=1}^{m} p(y^*|x^*,\theta^{(i)})$.
→In practice (with DNN) it is common to use the whole training data for each of the m NNs. (random init., data shuffling typically ensures sufficient diversity)
→Note that in the above approximation, we sample from an approximate (empirical) posterior distribution $\hat{p}$ using bootstrapping. (→not further explained)

(multi-class) classification with NNs: NNs can also be used for classification. For this we construct a NN with c outputs $F=[f_1,\dots,f_c]$, and normalize them into a probability distribution using softmax:
$p=\text{softmax}(f)$, i.e. $p_i = \frac{\exp(f_i)}{\sum_{j=1}^{c}\exp(f_j)}$, $p(y|x,\theta)=p_y$. (y: class)

→explicitly model the aleatoric uncertainty by injecting learnable (Gaussian) noise ε, and using $p=\text{softmax}(f+\varepsilon)$.

Calibration: A model is well calibrated, if its confidence coincides with its accuracy across many predictions. Try adjusting the confidence estimation.
The Calibration error can be estimated by partitioning the test points into bins according to predicted confidence values (similar confidences together), then calculate the relative accuracy of pos. samples in each bin, and calculate the average confidences (weighted over bin sizes).
Reliability Diagrams: For each bin, plot the computed confidence against the frequency, and put all bins in one diagram. We want the histogram to move along the identity function.
→Heuristics to improve model calibration:
- Histogram binning: Divide uncalibrated predictions of confidence $\hat{p}_i$ into bins, and assign a new calibrated score $\hat{q}_i = \text{Freq}(B_m)$ to each bin.
- Isotonic regression: Find piecewise constant function $f=[f_1,\dots,f_M]$ that minimizes the bin-wise squared loss:
$\min_{M,f,a} \sum_{m=1}^{M}\sum_{i=1}^{n} \mathbb{1}\{a_m \le P(y_i=1|x_i) < a_{m+1}\}\cdot(f_m-y_i)^2$ $\quad (0=a_1\le\dots\le a_{M+1}=1,\ f_1\le\dots\le f_M)$
Where $f_1,\dots,f_M$ are the calibrated scores for each bin.
- Platt-scaling: adjust each value z_i of the output layer to $q_i=\sigma(az_i+b)$, then learn $a,b\in\mathbb{R}$ to maximize the likelihood.
- Temperature scaling: Version of Platt-scaling with $b=0$ and $a=\frac{1}{T}$. T high: soft probabilities, T low: sharper probabilities.
- BNN usually improve calibration in practice, by themselves.

Active Learning: is about how to use uncertainty for deciding which data to collect. (which next sample is most useful?)

Mutual Information (information gain): Given r.v. X, Y we define

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$ the MI. (→symmetric)

The MI quantifies how much observing Y reduces uncertainty about X. Where

$$H(X|Y) = E_{y\sim p(y)}[H(X|Y=y)] = E_{(x,y)\sim p(x,y)}[-\log(p(x|y))]$$

is the conditional entropy, the expected remaining uncertainty in X, after observing Y. Note that $H(X|Y=y) \neq H(X|y)$, but it simply corresponds to a probabilistic update of our uncertainty in X, after observing the realization $y \sim Y$.

Further define the joint Entropy $H(X,Y) = E_{(x,y)}[-\log(p(x,y))]$

as the combined uncertainty about X and Y.

Note that $H(X,Y) = H(Y) + H(X|Y) = H(X) + H(Y|X)$.

Further $H(X|Y) \leq H(X)$, $I(X,Y) = H(X) + H(Y) - H(X,Y)$, $I(X,Y) \geq 0$. ("information never hurts principle")

Optimizing Mutual Information: Let f be a function which we are trying to approximate using Bayesian Learning (e.g. GP). Let D be set of points to potentially observe f at. We want to find $S \subseteq D$ which maximizes the information gain:

$$F(S) = H(f) + H(f|y_S) = I(f, y_S) \quad \left(= \frac{1}{2}\log(|I + \sigma^{-2}K_S|)\right.$$

Example for $X \sim \mathcal{U}([a,b])$, $y = X + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma_n^2 I)$ ← homoscedastic errors

But: note that optimizing F is NP-hard.

A simple strategy is a Greedy algorithm: Let $S_t = \{x_1, \dots, x_t\}$, then chose

$$x_{t+1} = \operatorname{argmax}_{x \in D} F(S_t \cup \{x\}) = \operatorname{argmax}_{x \in D} \underbrace{F(S_t \cup \{x\}) - F(S_t)}_{=: F(x|S_t)}$$ "marginal gain".

We can define $I(f, y_x | y_{S_t}) = H(f|y_{S_t}) - H(f|y_x, y_{S_t}) = F(S_t \cup \{x\}) - F(S_t)$.

For the example: $x_{t+1} = \operatorname{argmax}_x \frac{1}{2}\log\left(1 + \frac{\sigma_t^2(x)}{\sigma_n^2}\right) = \operatorname{argmax}_x \sigma_t^2(x)$.

→ "uncertainty sampling".

→ The mutual information $F(S)$ is monotone submodular:

$\forall x \in D, \forall A \subseteq B \subseteq D: F(A \cup \{x\}) - F(A) \geq F(B \cup \{x\}) - F(B)$. → "diminishing returns property".

→ Thus, the above Greedy algorithm provides a const.-factor approximation: $F(S_T) \geq (1 - \frac{1}{e}) \max_{S \subseteq D, |S| \leq T} F(S)$ $(1 - \frac{1}{e} \approx 0.63!)$

⇒ uncertainty sampling is near-optimal.

Uncertainty sampling in heteroscedastic case: Fails to distinguish epistemic and aleatoric uncertainty. Because the most uncertain outcomes are not necessarily most informative. Maximizing mutual information yields: $x_{t+1} \in \operatorname{argmax}_x \frac{\sigma_f^2(x)}{\sigma_n^2(x)}$ → epistemic uncertainty → aleatoric uncertainty.

Active Learning for Classification: In this setting a model produces a categorical distribution over the labels y_x for an input x. Uncertainty sampling corresponds to selecting samples that maximize the entropy of the predicted label:

$$x_{t+1} \in \operatorname{argmax}_x H(Y | x, x_{1:t}, y_{1:t})$$

→ But note that this is often not a good approach, since the most uncertain label is not necessarily the most informative!

↓(AL for classification)

Bayesian active learning by disagreement (BALD): We can write

$$x_{t+1} = \operatorname{argmax}_x I(\theta, y_x \mid x_{1:t}, y_{1:t}) = \operatorname{argmax}_x \underbrace{H(y_x \mid x_{1:t}, y_{1:t})}_{(1)} - \underbrace{E_{\theta \mid x_{1:t}, y_{1:t}}[H(y_x \mid \theta)]}_{(2)}$$

(1) looks for points where the average prediction is not confident.
(2) look for points where models are confident in their predictions.
→identifies points where models disagree about the label y_x.

Bayesian Optimization: Since data is costly, we want to efficiently improve our understanding of the data simultaneously to reaching certain goals (e.g. maximize f). For this, assume the following problem: Let $f^*: \mathcal{X} \to \mathbb{R}$, find $\operatorname{argmax}_{x \in \mathcal{X}} f^*(x)$, where we are only able to obtain noisy observations $y_t = f^*(x_t) + \varepsilon_t$. Further, these noisy observations are costly to obtain. To assume that similar alternatives yield similar results, we place a GP prior on f^*. →All following ideas generalize to other Bayesian Learning Problems.

→"exploration-exploitation dilemma".

Multi-armed-bandits (MAB) problem: We have k possible actions (arms) and want to maximize our reward, but no knowledge about the rewards distribution. Thus, we need to trade learning the reward distribution with following the most promising action.
Bayesian optimization is a version of MAB, with potentially infininite number of actions, but correlated reward.

Cumulative regret: The key performance metric in adaptively choosing new inputs to sample (online learning for BO) $y_t = f(x_t) + \varepsilon_t$ at, is called cumulative regret: $R_T = \sum_{t=1}^{T} (\max_x f^*(x) - f^*(x_t))$

The goal is to find algorithms, that achieve sublinear regret, i.e.

$\lim_{T \to \infty} R_T / T = 0$ thus $\max_t f(x_t) \to f(x^*)$.

→ We impose sublinearity, because with both algorithms (1) exploring forever, and (2) never exploring, the cumul. regret grows linearly.
→sublinear regret requires balancing exploration and exploitation.
→In Gaussian process bandit optimization we minimize R_T.

Upper Confidence Bound sampling (GP-UCB): Uses a so-called aquisition function, to adaptively choose new inputs to sample. Here

$$x_{t+1} = \operatorname{argmax}_x \mu_t(x) + \beta_{t+1} \cdot \sigma_t(x) \qquad (\sigma_t(x) = \sqrt{k_t(x,x)})$$

→posterior mean and variance

β_t regulates how confident we are about our model f.
→Optimism-based method: pick x with hope for best outcome.
→ this aquisition function is generally non-convex. (In low dim. use Lipschitz optimization; high-dim: gradient ascent with random init.)

Bayesian Regret of GP-UCB: Let $f \sim GP$, if we choose β_t "correctly", then $\frac{1}{T} \sum_{t=1}^{T} (f(x^*) - f(x_t)) = \mathcal{O}^*\left(\sqrt{\frac{\gamma_T}{T}}\right)$, $\gamma_T = \max_{|S| \leq T} I(f, y_S)$

→ maximum information gain γ_T determines the regret.
(i.e. how quickly we can gain information)
→Note that if the mutual information $I(f, y_S)$ is sublinear in T, then R_T is sublinear, and we converge to the true optimum.
→ f^* assumed to be smooth (smaller class) ⇒ small information gain.
f^* assumed to be rough (larger class) ⇒ high information gain.

↓ (→If not GP: obtain confidence intervals using approximate inference (e.g VI, MCMC, ensembles, ...))

↓ (GP-UCB)

→ Information gain bounds for some common kernels:

- Linear: $\gamma_T = \mathcal{O}(d \cdot \log(T))$
- Squared-exponential: $\gamma_T = \mathcal{O}((\log(T))^{d+1})$
- Matérn with $\nu > \frac{1}{2}$: $\gamma_T = \mathcal{O}\left(T^{d/(2\nu+d)} \cdot (\log(T))^{2\nu/(2\nu+d)}\right)$

→ This guarantees sublinear regret ⇒ convergence.

Frequentist regret for GP-UCB: Assume $f \in \mathcal{H}_k$ (reproducing kernel Hilbert space)

then $\frac{1}{T}\sum_{t=1}^{T}(f(x^*) - f(x_t)) = \mathcal{O}^*\left(\sqrt{\frac{\beta_T \gamma_T}{T}}\right)$

with $\beta_T = \mathcal{O}(\|f\|_k^2 + \gamma_T \log(T)^3)$ $\quad \gamma_T = \max_{|S| \le T} I(f, y_S)$

↳ "complexity of f".

Thompson Sampling: Describes an aquisition function for Bayesian Optimization. At iteration t, we sample $\tilde{f} \sim P(F \mid x_{1:t}, y_{1:t})$ and select

$$x_{t+1} \in \operatorname{argmax}_{x \in D} \tilde{f}(x)$$

→ randomness in realization of $\tilde{f}$ is sufficient to trade exploration and exploitation.

→ We can establish regret bounds similar to GP-UCB.

(→ If not GP: sample from posterior distribution over models, then optimize the sample)

Kernel Hyperparameter Estimation: There are many different ways to learn the hyperparameters. In principle we could alternate learning hyperparameters and observation selection. But in practice, there is a specific danger of overfitting: (1) Datasets are (in BO, AL) quite small, (2) Data points are selected dependent on prior observations.

Some solutions to these problems are being Bayesian about the hyperparam. (i.e. place a hyperprior on them and marginalize them out), or occasionally simply selecting some points at random.

(Finite) Markov Decision Process (MDP): is specified by a set of states $X = \{1, \dots, n\}$, actions $A = \{1, \dots, m\}$, transition probabilities (dynamics model)

$$P(x' \mid x, a) = P(X_{t+1} = x' \mid X_t = x, A_t = a),$$

and a reward function $r: X \times A \to \mathbb{R}$, mapping current state x and action a to some reward. r induces the sequence of rewards $R_t = r(X_t, A_t)$, $t \in \mathbb{N}$.

→ r can also depend on x'

→ r can be random with mean r.

→ First assume we operate in a known environment (P, r known), which is fully observable (agent knows current state).

→ Goal: Choose actions which maximize the reward. Specifically we want to maximize the discounted payoff:

$$G_t = \sum_{m=0}^{\infty} \gamma^m R_{t+m} \quad \text{or} \quad G_t = \sum_{m=0}^{T-1} R_{t+m}$$

(infinite horizon) (finite horizon)

→ maximizes immediate and long-term reward.

Policy: is a function that maps each state $x \in X$ to a probability distribution over the actions, i.e.

$$\pi(a \mid x) = P(A_t = a \mid X_t = x) \quad \text{(deterministic case: } \pi: X \to A\text{)}$$

→ stochastic policy

A policy π fully defines the behavior of an agent.

→ assume policies are stationary (i.e. do not change over time)

→ See later: all optimal policies are deterministic!

→ thus now view policies as deterministic (simplification)

→ Every policy induces a Markov Chain $(X_t^\pi)_{t \in \mathbb{N}}$ with transition probabilities

$$p^\pi(x' \mid x) = P(X_{t+1}^\pi = x' \mid X_t^\pi = x) = P(x' \mid x, \pi(x)) = \sum_{a \in A} \pi(a \mid x) \cdot P(x' \mid x, a)$$

(π deterministic; stochastic π)

↓

(policies)

→ If the agent follows a fixed policy, then the evolution of the process can be fully explained by a Markov chain.
→ it turns out we can find this policy π quite efficiently.

State-(action)-value function: We try to quantify the effect of our initial state (and action) on our objective G_t (discounted payoff):

state-value function: $V_t^\pi(x) = E_\pi[G_t \mid X_t = x]$, and

state-action-value function: $Q_t^\pi(x,a) = E_\pi[G_t \mid X_t = x, A_t = a]$

→ lecture: $J(\pi) = E_\pi[G_0]$.
→ where we can also decide about the initial action

→ Note that $V^\pi(x) = Q^\pi(x, \pi(x))$ (if π deterministic).
→ Since we have stationary dynamics model, all quantities are independent of the start time t. → Thus write $V^\pi(x) := V_t^\pi(x)$, $Q^\pi(x,a) := Q_t^\pi(x,a)$. (wlog.)

Bellman Expectation Equation: V^π, Q^π fulfill the following equations:

$$V^\pi(x) = r(x, \pi(x)) + \gamma \cdot E_{x' \mid x, \pi(x)}[V^\pi(x')] = \sum_a \pi(a|x)\left(r(x,a) + \gamma \sum_{x'} P(x'|x,a) V^\pi(x')\right)$$
$$= r(x, \pi(x)) + \gamma \cdot \sum_{x'} P(x' \mid x, \pi(x))\, V^\pi(x')$$

(π deterministic; π stochastic)

and

$$Q^\pi(x,a) = r(x,a) + \gamma \sum_{x'} P(x'|x,a) \cdot \sum_{a'} \pi(a'|x') \cdot Q^\pi(x',a')$$

⇒ deterministic case: $Q^\pi(x, \pi(x)) = r(x, \pi(x)) + \gamma \sum_{x'} P(x' \mid x, \pi(x))\, Q^\pi(x', \pi(x'))$

→ The BEE gives us a recursive formula for the state-action-value function $V^\pi(x)$ (in deterministic case).
It follows that for a fixed policy π, we can find V^π by simply solving a system of linear equations: (→ exactly!!)

$$\vec{V}^\pi = \vec{r}^\pi + \gamma P^\pi \cdot \vec{V}^\pi \iff \vec{V}^\pi = (I - \gamma P^\pi)^{-1} \vec{r}^\pi$$

where $V^\pi = \begin{pmatrix} V^\pi(1) \\ \vdots \\ V^\pi(n) \end{pmatrix}$, $r^\pi = \begin{pmatrix} r(1, \pi(1)) \\ \vdots \\ r(n, \pi(n)) \end{pmatrix}$, $P^\pi = \begin{pmatrix} P(1|1,\pi(1)) & \cdots & P(n|1,\pi(1)) \\ \vdots & & \vdots \\ P(1|n,\pi(n)) & \cdots & P(n|n,\pi(n)) \end{pmatrix}$

Fixed-point Iteration: can be used to (approximately) solve the above system of linear equations for V^π. We can show that V^π is the unique fixed-point of the affine mapping: $B^\pi V = r^\pi + \gamma P^\pi V$.

Use the algorithm: (1) initialize V^π (randomly), (2) update the value $V^\pi \leftarrow B^\pi V^\pi = r^\pi + \gamma P^\pi V^\pi$ (until convergence).
→ Fixed-point iteration converges to V^π exponentially quickly.

Policy Optimization: We denote our optimal policy to maximize the discounted payoff: $\pi^* = \arg\max_\pi E_\pi[G_0]$ → irresp. of initial state!

Alternatively say we want the maximal element π^* of the partial ordering $\pi \geq \pi' \iff V^\pi(x) \geq V^{\pi'}(x) \quad \forall x \in X$.
→ all optimal policies have identical value function, thus write

$$V^*(x) = V^{\pi^*}(x) = \max_\pi V^\pi(x), \quad Q^*(x) = Q^{\pi^*}(x) = \max_\pi Q^\pi(x).$$

Greedy Policies: The Greedy policy w.r.t. a state-action value function Q (based on any π') is given by

$$\pi_Q(x) = \arg\max_{a \in A} Q(x,a)$$ → choose action which maximizes the discounted payoff.

Analogously for a state value function V:

$$\pi_V(x) = \arg\max_{a \in A} r(x,a) + \gamma \sum_{x'} P(x'|x,a) \cdot V(x')$$ → inserting Bellmann expec equality for $V(x)$.

↓ (greedy policies)

→ The greedy policy π_V induces a new value function V^{π_V}. With this new value function, we can then again obtain a new greedy policy, inducing then again a new new value function ...
→ This shows a cyclic dependency.

Bellman Theorem: It turns out, the optimal policy π^* is a fixed-point of the above described cyclic dependency. In particular
π^* is optimal $\Leftrightarrow$ if π^* is optimal w.r.t its own value function

Note: $V^*(x) = \max_a Q^*(x,a) = \max_a r(x,a) + \gamma \sum_{x'} P(x'|x,a) V^*(x')$ → Bellmann optimality equation

→ This Bellmann optimality equation is non-linear (→ generally no closed-form solution). Thus typically use iterative methods to solve.
→ this proves: there always exists an optimal policy π^*. And π^* is deterministic and stationary

Policy Iteration: is a way of computing the optimal policy.
(1) Start with an arbitrary initial policy. (2) Use the Bellman expectation equation (solve linear system of equations), to find the value function of that policy, (3) then choose the greedy policy (wrt. the new value function) as the next iterate of the algorithm (→ then go back to (1)).
→ is guaranteed to monotonically improve (i.e. $V^{\pi_{t+1}}(x) \geq V^{\pi_t}(x) \quad \forall t, x$)
→ converges to optimal policy π^* in polynomial time $\mathcal{O}(n^2 m/(1-\gamma))$.

Value Iteration: is a way of computing the optimal policy by fixed-point iteration.
For this we regard V^* as the fixed point of the Bellmann-optimality equation $V^*(x) = \max_a r(x,a) + \gamma \sum_{x'} P(x'|x,a) V^*(x)$.
Algorithm: (1) Initialize $V_0(x) = \max_a r(x,a)$ for all $x \in X$,
(2) Iteratively (in t), let $Q_t(x,a) = r(x,a) + \gamma \sum_{x'} P(x'|x,a) V_{t-1}(x)$
then for each x, let: $V_t(x) = \max_a Q_t(x,a)$
(3) Break if $\|V_t - V_{t-1}\|_\infty = \max_x |V_t(x) - V_{t-1}(x)| \leq \varepsilon$. → for some previously chosen ε
(4) Choose greedy policy w.r.t. V_t.
→ converges to an ε-optimal solution in polynomial time.
→ but in general does not converge to optimal solution.
→ In sparse MDP, one iteration of Value Iteration has constant time. → i.e. sparse connections of state space.

Partially Observable Markov Decision Processes (POMDP): Let us assume the partially observed setting (i.e. agent can only access noisy observations y_t of its state x_t.
Define a POMDP by states X, actions A, transition probabilities $P(x'|x,a)$, and reward function $r: X \times A \to \mathbb{R}$. Further define observations Y, and observation probabilities $\sigma(y|x) = P(y_t = y | x_t = x)$.
POMDPs are controlled hidden Markov models.
→ generally a very powerful model, but typically extremely intractable.

POMDP can be reduced to an **MDP with enlarged state space.**
For this consider an MDP with states $b_t(x) = P(x_t = x | y_{1:t}, a_{1:t-1})$ "beliefs".
i.e. each state is a probability distribution over states of POMDP.
We keep track of how our beliefs change over time (i.e. Bayesian Filtering).
Given a prior belief b_t, and action a_t taken, and observation y_t, the belief state can be updated as follows:

$$b_{t+1}(x) = P(x_{t+1} = x | y_{1:t+1}, a_{1:t}) = \frac{1}{Z} \cdot \sigma(y_{t+1}|x) \sum_{x' \in X} P(x|x', a_t) \cdot b_t(x')$$

↓

↓(POMDPs)

↗ can use MDP methods to solve (theoretically.)

This leads us to a belief state MDP, defined by (1) belief space $\mathcal{B} = \Delta^X$ (hidden states X), (2) actions A, (3) transition probabilities $\tau(b'|b,a) = P(B_{t+1} = b' | B_t = b, A_t = a)$, and (4) rewards $\rho(a,b) = E_{x\sim b}[r(x,a)] = \sum_{x\in X} b(x)\, r(x,a)$

→ For finite horizon T (in discounted payoff), set of reachable belief states is finite (but exponential in T) → could use dynamic programming.
→ most belief states of POMDPs are never reached. Thus it is common to discretize the belief space (e.g. sampling, dimensionality reduction).
↳ alternative approach: Policy Gradients (see later).

Reinforcement Learning: is concerned with probabilistic planning in unknown environments. Those environments can be modelled using MDP's, but with unknown dynamics p and rewards r.
We focus on the fully observed setting (agent knows current state), and, for now, on small state and action spaces (i.e. tabular setting).
→ exploration-exploitation dilemma applies (like in Bayesian opt.)
↳ Bayesian Optimization = RL with fixed state and continuous action space.
→ Data is not i.i.d! But depends on played actions. →compared to supervised learning.
The collected data is modelled by so-called trajectories $\tau = (x_0, a_0, r_0, x_1, a_1, r_1, x_2, \dots)$. There are two common settings
- episodic: Agent learns over multiple "training" episodes i. Each episode results in a new trajectory τ_i, after which the environment resets.
- non-episodic/online: Agent learns "online", single trajectory.

On-policy RL methods: is where the agent has full control over which actions to pick. Can experiment with exploration and exploitation. →also that all observations/actions are wrt. the same policy (doesn't change)
Off-policy RL methods: Agent does not (always) have control over actions (only observational data). →more sample efficient. (than on-policy)

Model-based RL approach: learn underlying MDP, by estimating the dynamics model p and reward r. Then use these estimates to perform planning (i.e. policy optimization). →inherently off-policy
Model-free RL approach: Estimate the value function directly. →policy gradient meth. →actor critic methods

Learn underlying MDP: One way to do this, is to use MLE/MAP.
Such estimates are: $\hat{p}(x_{t+1}|x_t,a) = \frac{N(x_{t+1}|x_t,a)}{N(a|x)}$ $\tau = (x_0, a_0, r_0, x_1, \dots)$

where $N(x_{t+1}|x_t,a)$: # transitions from x_t to x_{t+1} with action a.
$N(a|x)$: # transitions that start in x_t, and play a.
and the rewards: $\hat{r}(x,a) = \frac{1}{N(a|x)} \sum_{t=0, x_t=x, a_t=a}^{\infty} R_t$

→ both estimates are unbiased, and correspond to the sample mean.
→ only accurate if each pair (x,a) is visited many times!

on-policy, model-based
ε-greedy algorithm: is a way of choosing our actions to best estimate p, r.
At each time step t with probability ε_t we pick an action uniformly at random, and with probability $1-\varepsilon_t$ we pick the best action under our current model.
If $\sum_t \varepsilon_t = \infty$ and $\sum_t \varepsilon_t^2 < \infty$, then we can converge to the optimal policy with probability 1. (using Monte Carlo control)
→ often performs fairly well
→ doesn't quickly eliminate suboptimal actions.

Rmax-Algorithm: is a way of estimating our dynamics model P, and rewards r. Idea: Optimism in the face of uncertainty.
This means, if $r(x,a)$ is unknown, we set $\hat{r}(x,a) = R_{max}$ (i.e. the maximum possible reward). Similarly, if $p(x'|x,a)$ is unknown, we imagine an additional "fairy-tale" state x^*, and set $\hat{P}(x^*|x,a)=1$.
x^* fulfills: $\hat{P}(x^*|x^*,a)=1 \ \forall a$, $\hat{r}(x^*,a) = R_{max}$: 

→ If in doubt, the agent believes actions to lead to a fairy tale state x^* with maximal rewards. → encourages exploration of unknown states. (→ because π^* would choose fairy-tale state, but we observe a new state and reward.)
In particular, the Rmax-algorithm first computes the optimal policy $\hat{\pi}$ from $\hat{P}$ and $\hat{r}$, then excecutes $\hat{\pi}$ for some number of steps, then updates $\hat{P}$ and $\hat{r}$, then recomputes the optimal policy from $\hat{P}$ and $\hat{r}$, and then repeats the process.
(→ For each state we explore all actions, and then always choose the best) → for all x in parallel

→ For some T, every T timesteps, with high probability Rmax either obtains near optimal reward, or visits at least one unknown state-action pair. (T is related to mixing time of MC).
→ With probability $1-\delta$, Rmax reaches an ε-optimal policy in time polynomially in $|X|, |A|, T, 1/\varepsilon, \log(1/\delta)$, and R_{max}-value.
This can be shown using the Hoeffding bound: $Z_1, \dots, Z_n$ i.i.d. bounded in $[0,C]$ with mean μ fulfill:

$$P\left(|\mu - \frac{1}{n}\sum_{i=1}^{n} Z_i| > \varepsilon\right) \leq 2\exp(-2n\varepsilon^2/C^2)$$

→ Rmax performs remarkably well in tabular setting.

Challenges of Model-based approaches:
Computational limitations: $\hat{p}(x'|x,a)$ and $\hat{r}(x,a)$ need to be stored in a table with $\mathcal{O}(n^2m)$ entries. → quickly unmanageable.
Also we need to "solve" the learned MDP, to obtain the optimal policy (many times). → Rmax: $\mathcal{O}(nm)$ times.

On-policy Value estimation: In model-free RL approaches, we estimate the value function V^π directly. Remember

$$V^{\pi}_{(new)}(x) = r(x,\pi(x)) + \gamma \sum_{x'} p(x'|x,\pi(x)) \cdot V^{\pi}_{(old)}(x) = E_{x',R}\left[R + \gamma V^{\pi}(x') | x, \pi(x)\right]$$
$$\approx r + \gamma V^{\pi}(x') \quad \text{for observed } (x,a,r,x')$$

This approx. corresponds to a one-sample Monte Carlo approx. (→ do this repeatedly, i.e. each iteration.)
But since we don't know the true $V^\pi(x)$, we use a bootstrapping estimate $V^{\pi}_{old}(x)$. (bootstrapping: estimate true quantity with an empirical quantity.)
→ bootstrapping is biased. We have high variance, because single sample.

Temporal Difference (TD-) Learning: is on-policy, model free approach where we reduce the variance of the bootstrapping estimate above.
Particularly, first initialize V^π arbitrarily. Then for each iteration t first, follow policy π to obtain transition (x,r,a,x'), then update

$$V^{\pi}(x) \leftarrow (1-\alpha_t)V^{\pi}(x) + \alpha_t\left(r + \gamma V^{\pi}(x')\right) \quad \text{or equivalently}$$
$$\left[V^{\pi}(x) \leftarrow V^{\pi}(x) + \alpha_t\left(r + \gamma V^{\pi}(x') - V^{\pi}(x)\right)\right].$$

→ on-policy TD-L for Q^π is called SARSA (same guarantees)

(Thm) If $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$ and all state-action pairs are chosen infinitely often, then $\hat{V}^\pi$ converges to the true V^π with probability 1.

Off-policy version: The actions a don't need to be chosen via π. Thus off-policy is also possible. The Thm above also holds for Q^π:
update: $Q^{\pi}(x,a) \leftarrow (1-\alpha_t)Q^{\pi}(x,a) + \alpha_t\left(r + \gamma Q^{\pi}(x', \pi(x'))\right)$.

Q-Learning: is an off-policy, model free approach, where we approximate the value function Q^* of the optimal policy π^* directly. For this, remember from the Bellman Theorem:

$\pi^*(x) = \arg\max_{a \in A} Q^*(x,a)$, and

and that $Q^*(x,a) = r(x,a) + \gamma \sum_{x' \in X} P(x'|x,a) \cdot \max_{a' \in A} Q^*(x',a')$

$\approx r + \gamma \max_{a' \in A} Q^*(x',a')$ for observation (x,a,r,x'). → one-sample Monte Carlo approximation

Using a bootstrapping estimate for this approximation results in the Q-Learning algorithm: (1) Initialize Q^* arbitrarily.
(2) For each iteration t, observe the transition (x,a,r,x'), then update:

$$Q^*(x,a) \leftarrow (1-\alpha_t) \cdot Q^*(x,a) + \alpha_t \left(r + \gamma \cdot \max_{a' \in A} Q^*(x',a')\right)$$

If $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$, and all state-action pairs are chosen infinitely often, then $\hat{Q}^*$ converges to optimal Q^* with probability 1.
→ with probability at least $1-\delta$, Q-learning converges to an ε-optimal policy in time polynomial in $\log(|X|)$, $\log(|A|)$, $1/\varepsilon$, $\log(1/\delta)$.

On-policy version: Optimistic Q-Learning. (is similar to R_{max}).
(1) Initialize $Q^*(x,a) = V_{max} \cdot \pi_{t=1}^{T_{init}} (1-\alpha_t)^{-1}$,
(2) For each iteration t choose action $a_t = \text{argmax}_{a \in A} Q^*(x,a)$, and observe (x,a,r,x'), then update

$$Q^*(x,a) \leftarrow (1-\alpha_t) Q^*(x,a) + \alpha_t \left(r + \gamma \cdot \max_{a' \in A} Q^*(x',a')\right)$$

where $V_{max} = R_{max}/(1-\gamma) \geq \max_{x,a} Q^*(x,a)$, T_{init} some initialization time.
→ with prob. at least $1-\delta$, optimistic Q-L conv. to ε-optim. policy in time poly. in $|X|$, $|A|$, $1/\varepsilon$, $\log(1/\delta)$.

→ Q-Learning requires $\mathcal{O}(nm)$ memory, and during each iteration $\mathcal{O}(m)$.

Challenges of Model-free Approaches: Model-free Q-learning takes polynomial time in $|X|$ and $|A|$. But in non-tabular settings (i.e. large state/action spaces) this is unacceptable. Because often domains are continuous, e.g. when modelling beliefs about states in a partially observable environment (POMDPs) → with low dimensional parameterization (regression).
→ Thus we want to learn/approximate value functions (tabular)

TD-Learning as SGD: We can reinterpret model-free methods (such as TD-/Q-Learning) as solving an optimization problem, where each iteration corresponds to a single gradient update. → In tabular setting $\theta = [\theta(1), \ldots, \theta(n)]$
Let us parameterize our estimate function of V^π, with param. θ.
The TD-Learning update rule: $V^\pi(x) \leftarrow (1-\alpha_t) V^\pi(x) + \alpha_t (r + \gamma V^\pi(x'))$

can equivalently be viewed as stochastic (semi-) gradient descent on the squared loss $L_2(\theta; x, x', r) = \frac{1}{2} \left(V(x;\theta) - r - \gamma V(x', \theta_{old})\right)^2$ → the term inside the $(\cdot)^2$, we call "TD-error" $=: \delta$

update via SGD: $V \leftarrow V - \alpha_t \delta = \hat{V}^\pi(x;\theta_{old}) - \alpha_t \cdot (\hat{V}^\pi(x;\theta_{old}) - r - \gamma \cdot \hat{V}^\pi(x';\theta_{old}))$

$= (1-\alpha_t) \hat{V}^\pi(x;\theta_{old}) + \alpha_t \cdot (r + \gamma \hat{V}^\pi(x';\theta_{old}))$ not wrt θ → True fkt

→ bootstrapping: we use θ_{old} value estimates as labels (and opt. wrt. them)
Note that the optimization target $r + \gamma \hat{V}^\pi(x';\theta_{old})$ moves between iterations.
↳ stability issues.
→ The same insight applies to learning the (optimal) action-value function Q^*.
→ This shows a path towards parametric function approximation.

Parametric Value function approximation: To scale to large state spaces, we approximate the value function using a parameterized model $V(x;\theta)$ or $Q(x,a;\theta)$. This is a strict generalization of the tabular setting.
A straightforward approach is to use a linear function approximation with hand-designed feature map Φ: $Q(x,a;\theta) = \theta^T \Phi(x,a)$.
Alternatively it is common to use deep neural networks. This approach is known as deep reinforcement learning.

Q-Learning with function approximation: we can generalize Q-Learning from the tabular setting. After observing a transition (x,a,r,x'), we update the parameters θ via the gradient of:

$$L_2(\theta; x,a,r,x') = \frac{1}{2}\left(\hat{Q}^*(x,a;\theta) - r - \gamma \max_{a' \in A} Q^*(x',a';\theta_{old})\right)$$

↳ "Bellman error"

Thus $\theta \leftarrow \theta - \alpha_t \cdot \delta \cdot \nabla_\theta Q(x,a;\theta)$ with $\delta := Q(x,a;\theta) - r - \gamma \max_{a'} Q(x',a';\theta)$.

→ In tabular setting, this is identical to Q-Learning.
→ Typically rather slow algorithm. (In practice convergence if function class "rich enough".)
→ moving gradient target (with each iteration) → stability issues.

(Double) Deep Q-Networks (DQN/DDQN): DQN aims to "stabilize" the optimization targets in the above gradient descend. For this it updates the NN used for the approximate bootstrapping estimate infrequently, to maintain a constant optimization target across multiple episodes.
Exact implementations of DQN vary. One approach is to clone the NN, and maintain one changing "online network" for most recent estimate of Q (parameterized by θ), and one fixed "target network" used as the optimization target (parameterized by θ_{old}), this is updated infrequently. (i.e. θ_{old} is only updated to θ after some amount of observations).
Thus in DQN does gradient descent according to the Loss:

$$L_{DQN}(\theta; D) = \frac{1}{2} \sum_{(x,a,r,x') \in D} \left(r + \gamma \max_{a' \in A} Q^*(x',a';\theta_{old}) - Q^*(x,a;\theta)\right)^2$$

↳ observation batch
↳ a' is chosen wrt. θ_{old} !!

→ Maximization Bias: The estimates of the true Q^* are noisy. But since we maximize noisy values, we receive a biased estimate.

DDQN addresses the problem of maximization bias by choosing the optimal action wrt. the new network (instead of θ_{old} in DQN).

Loss: $L_{DDQN}(\theta; D) = \frac{1}{2} \sum_{(x,a,r,x') \in D} \left(r + \gamma \cdot Q^*(x', a^*(x;\theta); \theta_{old}) - Q^*(x,a;\theta)\right)^2$

where $a^*(x';\theta) = \operatorname{argmax}_{a' \in A} Q^*(x',a';\theta)$

↳ a^* chosen wrt. new network θ. (not θ_{old})

Thus, online update: $\theta \leftarrow \theta + \alpha_t \left(r + \gamma \cdot Q^*(x', a^*(x';\theta); \theta_{old}) - Q^*(x,a;\theta)\right) \nabla_\theta Q^*(x,a;\theta)$

after observing a single transition (x,a,r,x'), $a^*(x';\theta)$ is treated as a const. when differentiating. θ_{old} is updated to θ after $|D|$-many transitions observed.

Policy Approximation: Another way of scaling to large (continuous) action spaces, is to directly learn an approximate parameterized policy $\pi^*(x) \approx \pi(x;\theta) = \pi_\theta(x)$ θ: parameters
using e.g. NN. Policy search and policy gradient are such methods.
Note: $\pi_\theta(x)$ is a probability distribution over A. Policy gradient methods fundamentally rely on randomized policies for exploration.

Policy Value Function: For approximating the optimal policy, we will focus on the discounted payoff through the **policy value function**

$$J(\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r(X_t, \pi_\theta(X_t))\right],$$

($= G_0$ (discounted payoff); → since $\pi_\theta(x)$ is a probability distribution, we can only compute an expected value.)

the expected discounted payoff of π_θ.

Bounded variant: $J_T(\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t \cdot r(X_t, \pi_\theta(X_t))\right] = E_{\pi_\theta}[G_{0:T}]$

The general idea is to find the optimal policy by maximizing $J(\theta)$ i.e. to find $\theta^* = \arg\max_\theta J(\theta)$.

Such methods we call Policy search methods.

Note that we can approximate $J(\theta)$ well in episodic tasks (i.e. agent can be "reset"), using "rollouts". Let $\tau^{(1)}, \dots, \tau^{(m)} \sim \pi_\theta$ trajectories ($\tau^{(i)} = (x_0^{(i)}, a_0^{(i)}, r_0^{(i)}, x_1^{(i)}, \dots)$) each of length T. For each trajectory we can compute the i-th (finite horizon) discounted payoff: $G_{0:T}^{(i)} = \sum_{t=0}^{T-1} \gamma^t r_t^{(i)}$. (is here seen as induced distribution over trajectories)

Then we can use Monte Carlo Sampling to approximate $J_T(\theta)$.

Thus $J(\theta) \approx J_T(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} G_{0:T}^{(i)}$

(→ on-policy (fundamentally))

Policy Gradients: To maximize $J(\theta)$, we want to compute its gradients. We can show that the following holds:

$$\nabla_\theta J(\theta) = \nabla_\theta J_T(\theta) = \nabla_\theta E_{\tau \sim \pi_\theta}[r(\tau)] = E_{\tau \sim \pi_\theta}[r(\tau) \nabla_\theta \log(\pi_\theta(\tau))]$$

where $r(\tau) := \sum_{t=0}^{T} \gamma^t \cdot r(x_t, a_t)$ (also: $\nabla_\theta \log(\pi_\theta(\tau)) = \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)}$ ↳ $\nabla_\theta \pi_\theta(x) = \pi_\theta(x) \cdot \nabla_\theta \log(\pi_\theta(x))$ "score gradient trick")

For estimating this gradient, we need to evaluate $\nabla_\theta \log(\pi_\theta(\tau))$:

Note that $\nabla_\theta \log(\pi_\theta(\tau)) = \sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t, x_t))$

using $\pi_\theta(\tau) = p(x_0) \cdot \prod_{t=0}^{T-1} \pi_\theta(a_t | x_t) \cdot P(x_{t+1} | x_t, a_t)$ → exploiting MDP structure

Thus $\nabla_\theta J(\theta) \overset{\approx}{=} E_{\tau \sim \pi_\theta}\left[r(\tau) \cdot \sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t, x_t))\right]$

$$\approx \frac{1}{m} \cdot \sum_{i=1}^{m} G_{0:T}^{(i)} \sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t^{(i)}, x_t^{(i)}))$$

→ using Monte Carlo Sampling. → episodic setting

→ Estimates are unbiased, but typically have large variance.

→ optimized policy can be randomized.

Baselines: We can reduce variance of policy gradient estimates, by using so-called baselines b. Note that

$$\forall b \in \mathbb{R}: E_{\tau \sim \pi_\theta}[r(\tau) \nabla_\theta \log(\pi_\theta(\tau))] = E_{\tau \sim \pi_\theta}[(r(\tau) - b) \cdot \nabla_\theta \log(\pi_\theta(\tau))]$$

(↗ inserted term with b simplifies to 0.)

a common baseline is $b(\tau_{0:t-1}) = \sum_{k=0}^{t-1} \gamma^k r_k$ (state dependent), also $G_0 - b(\tau_{0:t-1}) = \gamma^t G_{t:T}$ ($G_0 = r(\tau)$; "reward to go" of action a_t)

Thus yielding the gradient estimator:

$$\nabla_\theta J(\theta) \approx \nabla_\theta J_T(\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t G_{t:T} \cdot \nabla_\theta \log(\pi_\theta(a_t | x_t))\right]$$

($G_{t:T}$: "downstream return"; on-policy)

REINFORCE algorithm: performs SGD on the above reduced-variance policy gradient. Specifically: (1) initialize policy weights θ, then

(2) generate an episode (i.e. rollout) and obtain on trajectory τ.

(2.1) For each iteration $t=0$ to $t=T-1$ set $G_{t:T}$ to the return from iteration t, and perform the update

$$\theta \leftarrow \theta + \eta \gamma^t G_{t:T} \nabla_\theta \log(\pi_\theta(a_t | x_t))$$

(3) After all iterations, go back to (2), and generate a new trajectory.

↓

↓ (REINFORCE)

The variance of REINFORCE can be further reduced by subtracting from $G_{t:T}$: $\nabla_\theta J(\theta) \approx E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t (G_{t:T} - b_t) \nabla_\theta \log(\pi_\theta(a_t|x_t))\right]$

where b_t is the mean reward to go: $b_t = \frac{1}{T}\sum_{t=0}^{T-1} G_{t:T}$

→ main advantage of REINFORCE: can be used in continuous action spaces.
→ BUT: not guaranteed to find optimal policy. Even in small domains, REINFORCE can get stuck in local optima
→ policy gradient methods are slow. (large variance in gradient estimates
→ small steps, and need many MC rollouts
→ We cannot reuse data, since policy gradient methods are fundamentally on-policy (since estimation relies on policy not changing) → samples rollouts using the current estimate of the policy.

<u>Policy Gradient Theorem:</u> We can reinterpret the Policy Gradient in terms of the state-action value function Q, such that

$$\begin{aligned}\nabla_\theta J(\theta) &= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \cdot Q(x_t,a_t)\cdot \nabla_\theta \log(\pi_\theta(a_t|x_t))\right]\\ &= \int \rho_\theta(x)\cdot E_{a\sim\pi_\theta(x)}\left[Q(x,a)\nabla_\theta \log(\pi_\theta(a|x))\right] dx\\ &=: E_{(x,a)\sim\pi_\theta}\left[Q(x,a)\cdot\nabla_\theta\log(\pi_\theta(a|x))\right]\end{aligned}$$

where $\rho_\theta := \sum_{t=0}^{\infty}\gamma^t p(x_t = x)$ is the (non-normalized, discounted) state occupancy measure.

→ Since we do not know the true Q, we will approximate $Q_\varphi(x,a)$ using parameters φ (e.g. bootstrapping).

<u>On-policy Actor Critics:</u> (online setting) Actor Critic methods consist of a parameterization of the policy π_θ ("actor"), and of the value function Q_φ ("critic"). In DRL we use NN to parameterize both actor and critic. → allows scaling to large state/action spaces.
One example in the online setting is to use TD-Learning to learn the critic, and SGD for the actor. For some initial φ, θ, we do for each iteration t: (1) use π_θ to obtain (x,a,r,x').
(2) set $\delta = r + \gamma Q_\varphi(x', \pi_\theta(x')) - Q_\varphi(x,a)$, (3) then do the updates

$\theta \leftarrow \theta + \eta\, \gamma^t \cdot Q_\varphi(x,a)\cdot \nabla_\theta \log(\pi_\theta(a|x))$ → actor update
$\varphi \leftarrow \varphi + \eta\cdot\delta\cdot\nabla_\varphi Q_\varphi(x,a)$ → critic update (we disregard dependence of Q on θ.)

→ Note: the actor is not guaranteed to improve, due to bias in estimates.

To further reduce variance of the above estimates, we can introduce a baseline into estimate of our actor:

$\theta \leftarrow \theta + \eta_t\, \gamma^t \left(Q_\varphi(x,a) - V_\varphi(x)\right)\cdot\nabla_\theta \log(\pi_\theta(a|x))$ → $A(x,a) = Q(x,a) - V(x)$

where $A_\varphi(x,a) = Q_\varphi(x,a) - V_\varphi(x)$ is an estimate of the advantage function.
⇒ We call this algorithm A2C (advantage actor critic).
→ Actor Critic implementations often use advantage function. → easier to estimate (than Q).

Algorithms like GAE (generalized advantage estimation) address the bias-variance tradeoff by blending the estimates of REINFORCE (high variance, but unbiased) with using a bootstrapped Q-function (smaller variance, but biased) for the gradient estimates.

→ Actor Critic methods usually rely on randomization of the policy, for exploration. This is often insufficient. → use ε-greedy policy. → because policy quickly collapses into deterministic policy.
→ <u>low sample efficiency</u>. On-policy actor critic methods need extremely large number of interactions, because they cannot reuse past data.
→ TRPO / PPO address this.

TRPO (trust region policy optimization): is a modern variant of Actor-Critic methods, which slightly improves sample efficiency.
During each iteration we use a fixed critic to optimize the policy, particularly: $\theta_{k+1} \leftarrow \underset{\theta}{\operatorname{argmax}}\ \hat{J}(\theta)$ s.t. $KL(\theta \| \theta_k) \leq \delta$

where $\hat{J}(\theta_k, \theta) = E_{(x,a)\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|x)}{\pi_{\theta_k}(a|x)} \cdot A_{\theta_k}(x,a)\right]$ → advantage function

→ Can reuse data from rollouts within the same iteration, as long as they can still be "trusted" → "somewhat" off-policy → fundamentally still on-policy.

PPO (proximal policy optimization) is a variant of TRPO, which works well in practice. It controls importance of weights directly, instead of imposing a constraint on KL-divergence. → GPT uses PPO.

Off-policy Actor Critics: In many applications sample efficiency is crucial, which is why we look at off-policy methods, where past data can be reused. Such algorithms use reparameterization gradient estimates (like in Variational Inference), instead of score gradient estimates. These methods are closely related to value iteration, to learn the optimal value function, to then find the optimal policy.
For now assume the policy π is deterministic.
Remember the DQN loss: $L_{DQN}(\theta; D) = \frac{1}{2}\sum_{(x,a,r,x')\in D}\left(r + \gamma \max_{a'\in A} Q^*_{\theta_{old}}(x',a') - Q^*_\theta(x,a)\right)^2$

to solve the maximization problem of Q^* if the action space A is very large, we can replace the maximum with a parameterized policy.

$$L(\varphi) = \frac{1}{2}\sum_{(x,a,r,x')\in D}\left(r + \gamma \cdot Q_{\varphi_{old}}(x', \pi_\theta(x')) - Q_\varphi(x,a)\right)^2$$

By the Bellman Theorem, we want π_θ to approximate the greedy policy:

$\pi_\theta(x) \approx \pi^*(x) = \underset{a\in A}{\operatorname{argmax}}\ Q^*_\varphi(x,a)$.

And if we allow "rich enough" policies, then this is equivalent to:

$\theta^* \in \underset{\theta}{\operatorname{argmax}}\ E_{x\sim\mu}[Q_\varphi(x, \pi_\theta(x))]$ where $\mu(x) > 0$: "explores all states".

Then, we use a bootstrapping estimate of Q^*, and thus neglect the dependence of the critic Q^* on the policy (actor) parameters θ.
We can use chain rule: $\nabla_\theta Q^*_\varphi(x, \pi_\theta(x)) = D_\theta Q^*_\theta(x,a)\,|_{a=\pi_\theta(x)}$.

Exploration: usually policy gradient techniques rely on randomness of the policy. But here we assume deterministic policies.
→ DDPG (deep deterministic policy gradients) addresses this problem, by injecting Gaussian noise into actions selected by π_θ. → "Gaussian noise dithering"
This is essentially equivalent to using Q-Learning, but replace max with π_θ.
Particularly: (1) Initialize θ, φ, $D = \{\}$ replay buffer, $\theta_{old} = \theta$, $\varphi_{old} = \varphi$.
repeat: (2) Observe state x, pick action $a = \pi_\theta(x) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \lambda I)$.
(3) excecute and observe (x,a,r,x'), store this in D.
(4) If we have made enough observations, do
(4.1) Sample mini batch $B \subseteq D$, for each observation (in batch) compute $y = r + \gamma_1 Q_{\varphi_{old}}(x', \pi_{\theta_{old}}(x'))$, and update
$\varphi \leftarrow \varphi - \eta \nabla_\varphi \frac{1}{|B|}\sum_B (Q_\varphi(x,a) - y)^2$ → critic update
$\theta \leftarrow \theta + \eta \nabla_\theta \frac{1}{|B|}\sum_B Q_\varphi(x, \pi_\theta(x))$ → actor update
$\varphi_{old} \leftarrow (1-\rho)\varphi_{old} + \rho\cdot\varphi$ AND $\theta_{old} \leftarrow (1-\rho)\theta_{old} + \rho\cdot\theta$

→ TD3 (Twin Delayed DDPG) extends DDPG by using two critic networks, for predicting maximum action, and for evaluating the policy. This addresses maximization bias. It also increases stability, by delaying updates to the actor network.

Randomized policies in off-policy Actor Critics: To ensure exploration, instead of injecting noise (e.g. DDPG), we can directly allow randomized policies.
Looking at DDPG, instead of using the critic update (for φ), we can obtain unbiased gradient estimates, by sampling $a \sim \pi_{\theta_{old}}(x')$.
Further suppose the policy is reparametrizable, so we can apply $\mathcal{N}(0,I)$
$a = \psi(x, \theta, \varepsilon)$ → reparametrization trick

$$\nabla_\theta E_{a\sim\pi_\theta}[Q_\varphi(x,a)] = E_{\varepsilon\sim\nu}[\nabla_\theta Q_\varphi(x, \psi(x,\theta,\varepsilon))]$$
$$= E_\varepsilon[\nabla_a Q_\varphi(x,a)|_{a=\psi(x,\theta,\varepsilon)} \cdot \nabla_\theta \psi(x,\theta,\varepsilon)]$$

and thus we can compute gradients in this case.
→ **SVG** (stochastic value gradients) algorithm uses this.
→ SVG often does not explore enough, and lands in local optima.

Maximum entropy reinforcement learning (MERL): Encourages exploration, by letting policies exhibit some uncertainty.
For this we redefine the policy value function to be
λ: Temperature parameter.

$$J_\lambda(\theta) = J(\theta) + \lambda H(\pi_\theta) = E_{(x,a)\sim\pi_\theta}[r(x,a) + \lambda H(\pi(\cdot|x))]$$

The preference for entropy in the actor distribution encourages exploration.
→ We can suitably define regularized (action)-value functions, which we then call "soft" value functions.
→ **SAC** algorithm: further uses reparametrization gradients.

Language Models as Agents: In some applications (e.g. chatbots) it is difficult to quantify the reward associated with one or a sequence of actions. To align an agents behavior with human expectations, there are several methods to present feedback to the agent: (1) numerical score. In practice too subjective to give reliable results. (2) Comparison-based feedback. Contains less information, thus takes longer to learn complex behavior. Easy to collect, since humans can easily give comparison feedback.
In language models the agent is given a prompt, and returns a (stochastic) response based on previous responses. The generation of a response can be understood as a policy $\pi_\theta(y_{t+1}|x, y_{1:n})$, which generates the next token (letter/word/...) of the response, depending on the prompt, and all previous tokens.
The training of such models usually consists of 2 main steps.
(1) supervised fine-tuning, training the model on example data.
(2) post-training, using preference feedback

For post-training we can define rewards, based on comparisons
$y_A \succ y_B | x$ if $r(y_A|x) > r(y_B|x)$.
A popular choice for modelling preferences is the Bradley-Terry model:

$$\hat{P}(y_A \succ y_B | x, r) = \frac{\exp(r(y_A|x))}{\exp(r(y_A|x)) + \exp(r(x_B|x))} \quad \text{for some } r(y|x)$$

with loss

$$\text{loss}(\hat{r}) = \sum_{y_A, y_B, \mu} \mu(1) \log(\hat{P}(y_A \succ y_B | x, \hat{r})) + \mu(2) \log(\hat{P}(y_B \succ y_A | x, \hat{r}))$$

In the case of ChatGPT, we first train on a massive dataset, then fine-tune over high quality data, then learn rewards from comparison feedback, then train with actor critic on learned reward

→ Add penalty term to keep our policy close to the model which was used to generate comparison feedback to train $\hat{r}$.
Thus use $r(y) = \hat{r}(y) - \lambda KL(\pi_{\theta_{PPO}}(y) \| \pi_{\theta_{base}}(y))$
↳ policy from pre-training/fine tuning.

Model-based Reinforcement Learning: Learning a model can help to dramatically reduce the sample complexity, compared to model-free techniques. For this we will use function approximation to approximate the dynamics model f resp and rewards r, which model our environment. Further, we apply Bayesian Inference, since often understanding the uncertainty in our model is crucial for planning.
The general approach in Model-based RL consists of (1) start with some π_{init}, and initial data D. Then (2) for several iterations (2.1) roll out π to collect observations, (2.2) learn model of f and r from data D, (2.3) plan new policy π based on estimated f, r.

Main challenges: perform planning, learn f and r accurately and efficiently, effectively trade exploration and exploitation.

Planning: In the following we will focus on the setting of continuous state and action spaces, fully observable state spaces, no constraints, and non-linear dynamics model.

Planning with known deterministic dynamics model: For our dynamics model $x_{t+1} = f(x_t, a_t)$, our objective becomes

$$\max_{a_{0:\infty}} \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \quad \text{s.t.} \quad x_{t+1} = f(x_t, a_t)$$

→ infinite horizon discounted return

But we cannot explicitly optimize over an infinite horizon.
→ receding horizon control (RHC) / model predictive control (MPC):
To address the above problem, we can iteratively plan over finite horizons i.e. in each round plan over finite horizon H, and then carry out the first action, and after replan.
In particular: (1) at each iteration t, observe x
(2) Optimize performance over: $\max_{a_{t:t+H-1}} \sum_{i=t}^{t+H-1} \gamma^{i-t} r_i(x_i, a_i)$ s.t. $x_{i+1} = f(x_i, a_i)$
(3) carry out a_t, then replan.

We observe that x_i can be interpreted as a deterministic function $x_i(a_{t:i-1})$ dependent on all previous actions (from time t). Thus, for (2), we maximize $J_H(a_{t:t+H-1}) = \sum_{i=t}^{t+H-1} \gamma^{i-t} r(x_i(a_{t:i-1}), a_i)$
→ in general non-convex.

→ In continuous action space, and differentiable f and r, we can analytically compute gradients (by backpropagating through time) → using the chain rule
→ difficult for large H: local minima, vanishing/exploding gradients.

Random Shooting Methods describe a sampling approach towards global optimization of $J_H(a_{t:t+H-1})$. First generate m sets of random samples $a^{(i)}_{t:t+H-1}$ (e.g. from Gaussian, cross entropy method, ...) → gradually adapts sampling distr. from their rewards.
Then pick the sequence that optimizes J_H.

In finite horizon planning with sparse rewards, H might not go far enough, to get a sense of the optimal reward. To address this, we try estimating also the tail of the discounted return with the value function:

$$J_H(a_{t:t+H-1}) = \underbrace{\sum_{i=t}^{t+H-1} \gamma^{i-t} r(x_i(a_{t:i-1}), a_i)}_{\text{long term}} + \underbrace{\gamma^H V(x_{t+H})}_{\text{short term}}$$

We can use methods like TD-/Q-Learning to approximate V.
→ Setting $H=1$ coincides with the greedy policy, and we arrive in the model-free setting.

Planning with stochastic dynamics model: trajectory sampling is a generalization of MPC over stochastic models:
(1) at each iteration t, observe state x_t,
(2) Optimize the expected performance over the horizon H:

$$\max_{a_{t:t+H-1}} E_{x_{t+1:t+H}}\left[\sum_{i=t}^{t+H-1} \gamma^{i-t} \cdot r_i + \gamma^H V(x_{t+H}) \mid a_{t:t+H-1}\right]$$

(3) carry out action a_t, then replan.

Explicitly computing this integral is not feasible. Thus it is common to approximate the expectation using Monte Carlo (trajectory) sampling. But notice that the sampled trajectory depends on the picked actions. To resolve this issue, we can use the reparameterization trick.

We say a dynamics model f is reparametrizable iff. $x_{t+1} = f(x_t, a_t, \varepsilon_t)$ where ε_t is independent of x and a.
Then x_i is determined by $a_{t:i-1}$ and $\varepsilon_{t:i-1}$ via
$x_i := x_i(a_{t:i-1}, \varepsilon_{t:i-1}) := f(f(\ldots(\underbrace{f(x_t, a_t, \varepsilon_t)}_{=x_{t+1}}, a_{t+1}, \varepsilon_{t+1}), \ldots), a_{i-1}, \varepsilon_{i-1})$

This allows us to obtain unbiased samples of J_H using Monte Carlo sampling:

$$J_H(a_{t:t+H-1}) \approx \frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{t+H-1}\left(\gamma^{i-1} \cdot r(x_i(\varepsilon_{t:i-1}^{(k)}, a_{t:i-1}), a_i) + \gamma^H V(x_{t+H})\right)$$

($\varepsilon^{(k)}$: i.i.d.)

→ optimize this via analytic gradient, or shooting methods.

Planning with parametrized policies: When the time horizon is large, or we encounter similar states many times, it can be beneficial to parameterize the policy π directly: $a_t = \pi_\theta(x_t)$
This policy can be trained offline, and evaluated cheaply online. → "open loop control"
Analogously to Q-Learning / DDPG / SVG, the objective becomes

$$J(\theta) = E_{x_0 \sim \mu}\left[\sum_{i=0}^{H-1} \gamma^i r_i + \gamma^H Q_\theta(x_H, \pi_\theta(x_H)) \mid \theta\right] \quad (\mu(x) > 0 \ \forall x)$$

→ $H=0$: identical to DDPG objective.
→ Can use reparameterization to allow stochastic policies.

Note: Instead of using Monte Carlo approximation to evaluate a policy, there are more refined ways to approximate $J(\theta)$ (e.g. Moment matching, Variational Inference)

Learn the Dynamics Model: In RL the dynamics model f and rewards r are (of course!) not known.
Key insight: observed transitions and rewards are conditionally independent, because of the Markovian structure of the MDP.
This allows us to treat the estimation of f and r as a simple regression (or density estimation in stochastic case) problem. Thus we can estimate f and r off-policy with standard supervised learning techniques from a replay buffer $D = \{(\underbrace{x_i, a_i}_{\text{input}}, \underbrace{r_i, x_{i+1}}_{\text{label}})_i\}$.
So each observation is a labeled data point.
→ Note: input depends on our actions taken.

In the following we focus on learning f (learning r is analogous).
In particular we want to learn probabilistic dynamics models: $x_{t+1} \sim f_\theta(x_t, a_t)$.
Example: Conditional Gaussian dynamics $x_{t+1} \sim \mathcal{N}(\mu_\theta(x_t, a_t), \Sigma_\theta(x_t, a_t))$

represent Σ_θ via lower triangular matrix: $\Sigma_\theta = C_\theta C_\theta^T$
→ only needs $n(n+1)/2$ param; guarantees automatically p.s.d.
→ allows reparameterization: $x_{t+1} = \mu_\theta(x_t, a_t) + C_\theta(x_t, a_t) \cdot \varepsilon$, $\varepsilon \sim \mathcal{N}(0, I)$.

(learning the dynamics model)

A first approach to estimating f could be to use the MAP estimate (given a prior and likelihood).
However, using point estimates leads to a key pitfall of Model-based RL.
→ often performs very poorly.
→ Errors in model estimate compound, over multiple time steps ($H>1$).
These errors are exploited by planning algorithm → overfitting.
→ To solve this pitfall, we need to capture uncertainty in our estimated model. (→ epistemic and aleatoric uncertainty).

Bayesian Learning of the dynamics model: captures the uncertainty about our model by modelling a distribution over f, e.g model f as a GP or BNN, given a prior belief about f (or r). → all these methods also apply to rewards r.
→ This allows us to use all (approximate) inference techniques which we learnt earlier. (e.g. variational inference, dropout, ensembles, ...)

Let f be chosen from some prior distribution $p(f)$, and we obtain the posterior distribution $P(f|D)$ for $x_{t+1} \sim f(x_t, a_t)$.
Note that Epistemic: uncertainty in $P(f|D)$ → uncertainty about the model
Aleatoric: uncertainty in $P(x_{t+1}|f, x_t, a_t)$ → uncertainty about MDP transitions (noise!)
Recall that epistemic uncertainty corresponds to a distribution over MDPs f. Whereas aleatoric uncertainty uncertainty corresponds to randomness in transitions within the MDP f.
→ disregard epistemic uncertainty, once a MPD was chosen for planning. → then only concentrate on aleatoric uncertainty.
Thus we receive the following estimate of our reward

$$J_H(a_{t:t+H-1}) \approx \frac{1}{m} \sum_{k=1}^{m} \sum_{i=t}^{t+H-1} \gamma^{i-t} r_i(x_i(a_{t:i-1}, \varepsilon_{t:i-1}^{(k)}, f^{(k)}), a_i) + \gamma^H V(x_{t+H})$$

(aleatoric uncertainty correspondent; epistemic uncertainty correspondent)

where $f^{(k)} \sim P(f|D)$ and

$$x_i := x_i(a_{t:i-1}, \varepsilon_{t:i-1}, f) := f(f(\dots(f(x_t, a_t, \varepsilon_t), a_{t+1}, \varepsilon_{t+1}), \dots), a_{i-1}, \varepsilon_{i-1})$$

using Monte Carlo sampling of $f^{(k)}$! → We are essentially using Monte Carlo Sampling over an "ensemble" of MDP's.

This leads us to a Greedy exploitation algorithm for model-based RL. It greedily maximizes the expected reward w.r.t. the transition model, taking into account epistemic uncertainty.
Particularly: (1) Initialize (possibly empty) data D, and prior $P(f) = P(f|D)$
(2) For several episodes do the following:
(2.1) plan new policy π (approx.) maximizing: $\max_\pi E_{f \sim p(\cdot|D)}[J_H(\pi; f)]$
(2.2) roll out π to collect more data in D.
(2.3) update posterior $p(f|D)$.

PETS (probabilistic ensembles with trajectory sampling) is the above algorithm in the context of NN. It uses an ensemble of NN, each predicting cond. Gaussian transition distributions, and Trajectory sampling to evaluate performance, and MPC for planning. → state of the art algorithm
→ Exploration only happens due to uncertainty in the model.
→ PILCO is algorithm in context of Gaussian Process models. (Moment matching instead of MC sampling)

Exploration in learning dynamics model: A key difference between RL and classical supervised learning is that the chosen actions affect the data, which the model learns from.
→ Exploration-exploitation Dilemma.

This problem can be resolved using: added exploration noise (e.g. → Gaussian noise "dithering"), Thompson Sampling, Optimistic exploration.

↗for learning the dynamics model

Thompson Sampling: Recall that TS, instead of picking the best action across all realizations of f, it picks the action which performs best for a single realization of f. The epistemic uncertainty in the realizations of f leads to variance in the picked actions, which incentivizes exploration more.

Particularly: (1) Initialize (possibly empty) data D, and prior $p(f) = p(f|D)$.
(2) For several episodes do the following:
(2.1) sample model $f \sim p(\cdot|D)$
(2.2) plan policy π, to (approx.) maximize: $\max_\pi J_H(\pi, f)$.
(2.3) roll out π, collect more data
(2.4) update posterior $p(f|D)$.

(learning dynamics model) ↗ Bayesian Optimization.

Optimistic Exploration: We have seen in BO that optimism is a central pillar for exploration in (tabular) RL.
Let us consider a set $\mathcal{M}(D)$ of plausible models, given data D.
(e.g. $\mathcal{M}(D) = \{f \mid f_i(x,a) \in \mu_i(x,a|D) \pm \beta\sigma_i(x,a|D), \forall x,a\}$ for cond. Gaussians)
→ This leads us to Optimistic Exploration algorithm, which picks the best action w.r.t. the most optimistic model in $\mathcal{M}(D)$.

Particularly: (1) Initialize (possibly empty) data D, and prior $p(f) = p(f|D)$.
(2) for several episodes do the following:
(2.1) plan policy π, to (approx.) maximize: $\max_\pi \max_{f \in \mathcal{M}(D)} J_H(\pi, f)$
(2.2) roll out π, collect more data
(2.3) update posterior $p(f|D)$.

→ In general, the joint maximization over π and f is very hard.

This leads us to **H-UCRL** (hallucinated upper confidence reinforcement learning).
To solve the above problem, it interprets the above problem as maximizing the expected reward in an augmented (optimistic) MDP with known dynamics model $\hat{f}_t$, with larger action space and additional decision variables ($\eta_i(\cdot)$).
Particularly, it considers the optimization problem

$$\pi_{t+1} = \arg\max_\pi \max_{\eta(\cdot) \in [-1,1]^p} J_H(\pi; \hat{f}_t)$$

with "optimistic dynamics": $\hat{f}_{t,i}(x,a) = \mu_{t,i}(x,a) + \beta_{t,i} \cdot \eta_i(x,a) \cdot \sigma_{t,i}(x,a)$

→ η_i controls the variance of an action. Thus, the agent can freely choose the state, which it lands in, taking action a.
→ For this known MDP we can use our developed toolbox.

↳ Can be seen as an extension of the UCB aquisition function from Bayesian Optimization to RL.

→ Optimistic exploration works best for hard exploration (e.g. large penalties for certain actions, or sparse rewards).
→ Even for small penalties optimistic explorations learns good policies faster.

Constrained (safe) Exploration: In high-stakes applications, we need to guarantee safety (i.e. avoid unsafe states). Thus exploration can be dangerous.
For this, we denote a set X_{unsafe} of unsafe states, to pessimistically forecast plausible consequences of our actions.
The idea of constrained MDPs is to be optimistic for the rewards, and pessimistic for the costs. But first, optimize

$$\max_\pi J(f^*, \pi) = \max_\pi E_{x_0 \sim \mu, x_{1:\infty} | \pi, f^*}\left[\sum_{t=0}^\infty \gamma^t r_t\right]$$

μ: distribution over initial state.

subject to $J_c(f^*, \pi) = E_{x_0 \sim \mu, x_{1:\infty} | \pi, f^*}\left[\sum_{t=0}^\infty \gamma^t \cdot c(x_t)\right] \leq \delta$ → bounds probability of visiting an unsafe state with π.

for some cost function $c(x)$, e.g. $c(x) = \mathbb{1}\{x_{unsafe}\}$.
→ augmented Lagrangian method is used to solve this.

↓ (constrained exploration)

→ But note that it is not ensured that constraints are not violated during optimal policy search. Thus, this method is more applicable in an episodic setting. (simulated environment)

Thus we arrive at the strategy to to be optimistic w.r.t. future rewards, and pessimistic w.r.t. future constraint violations. Solve:

$$\max_{\pi} \max_{\tilde{f} \in M(D)} J(\tilde{f}, \pi) \quad \text{subject to} \quad \max_{\tilde{f} \in M(D)} J_c(\tilde{f}, \pi) \leq \delta$$

→ We call this the optimistic/pessimistic CMDP

→ LAMBDA: solve the above CMDP using augmented Lagrangian method

- feasible solutions on all tasks
- better sample efficiency than model-free baselines
- Improvements in safety during training.

} experimental results of LAMBDA

Safety Filters: To address unsafe exploration we can (slightly) adjust a potentially unsafe policy, to obtain a policy $\hat{\pi}$, which avoids entering unsafe states with high probability.

For this we can use the H-UCRL reparametrization to verify the feasibility of any π. Optimize

$$\max_{\eta(\cdot) \in [-1,1]^d} J_c(\tilde{f}, [\pi, \eta]) \leq \delta \quad \text{s.t. } \tilde{f}(x,a) = \mu_{t-1}(x,a) + \beta_{t-1} \cdot \Sigma_{t-1}(x,a) \cdot \eta(x,a).$$

and then $\eta^* = \operatorname{argmax}_{\eta} J_c(\tilde{f}, [\pi, \eta])$

$\Rightarrow$ Pessimistic value $C^{(P)}_{\pi}(x) = J_c(\tilde{f}, [\pi, \eta^*] \mid x_0 = x)$

Then we can find the "maximally safe" policy via robust RL

$$\pi_{safe} = \operatorname{argmin}_{\pi} \max_{\eta} J_c(\tilde{f}, [\pi, \eta])$$

Given an arbitrary policy π, we can apply the safety filter:

$$\Rightarrow \hat{\pi}(x) = \operatorname{argmin}_{a} \| \pi(x) - a \|$$

$$\text{subject to } C^{(P)}_{\pi_{safe}}(x') \leq \delta. \quad (x' \rightarrow \text{next state})$$