Math

Bayes Thrm, Conditional Prob.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$
$$P(X, Y \mid Z) = \frac{P(X, Y, Z)}{P(Z)}$$

Rules for the Mean

$$E(c) = c$$
$$E(X + c) = E(X) + c$$
$$E(cX) = cE(X)$$
$$E(X + Y) = E(X) + E(Y)$$

Rules for the Variance

$$Var(c) = 0$$
$$Var(X + c) = Var(X)$$
$$Var(cX) = c^2 Var(X))$$
$$Var(X + Y) = Var(X) + Var(Y)$$
$$Var(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

Law of Total Probability

$$p(x) = \sum_y p(x \mid y)p(y)$$
$$p(x) = \int_y p(x \mid y)p(y)dy$$

Univariate Gaussian

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Multivariate Gaussian

$$f_{\mathbf{X}}(x_1, \ldots, x_k) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}$$

Univariate Laplacian

$$f(x) = \frac{1}{2b} e^{\left(-\frac{|x-\mu|}{b}\right)}$$

Conditional Random Vectors

$$\mu_{A|B} = \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(x_B - \mu_B)$$
$$\Sigma_{A|B} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{AB}$$
$$\Sigma_{AB} = \begin{bmatrix} \sigma_{i_1,j_1} & \cdots & \sigma_{i_1,j_m} \\ \vdots & \ddots & \vdots \\ \sigma_{i_k,j_1} & \cdots & \sigma_{i_k,j_m} \end{bmatrix}$$

Linearity of Gaussians

$$MX \sim \mathcal{N}(M\mu_V, M\Sigma_{VV}M^T)$$
$$X + X' \sim \mathcal{N}(\mu_V + \mu'_V, \Sigma_{VV} + \Sigma'_{VV})$$

Bayesian learning

$$p(\theta|X, Y) = \frac{1}{Z}p(\theta)\prod_{i=1}^{n} p(y_i|x_i, \theta)$$
$$p(y^\star|x^\star, X, Y) = \int p(y^\star|x^\star, \theta)p(\theta|X, Y)$$

Bayesian Linear Regression

$$p(w|X, y) = \mathcal{N}(w; \bar{\mu}, \bar{\Sigma})$$
$$\bar{\mu} = (X^T X + \frac{\sigma_n^2}{\sigma_p^2} I)^{-1} X^T y$$
$$\bar{\Sigma} = (\frac{1}{\sigma_n^2} X^T X + \frac{1}{\sigma_p^2} I)^{-1}$$
$$y^* = w^T x^* + \epsilon$$
$$p(y^*|X, y, x*) = \mathcal{N}(\bar{\mu}^T x^*, x^{*T}\bar{\Sigma}x^* + \sigma_n^2)$$

Online Bayesian Linear Regression

$$X^T X = \sum_{i=1}^{t} x_i x_i^T$$
$$X^T y = \sum_{i=1}^{t} y_i x_i$$

MLE and MAP regression

$$w_{MLE} = (X^T X)^{-1} X^T y$$
$$w_{MAP} = (I\frac{\sigma_n^2}{\sigma_p^2} + X^T X)^{-1} X^T y$$

Gaussian Proccesess

$A \in \mathbb{R}^{m \times n}$, f_A is a collection of R.V s.t. $f_A \sim \mathcal{N}(\mu_A, K_{AA})$.

$$K_{AA} = \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \ldots & k_(x_m, x_m) \end{bmatrix}$$

$$\mu_A = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{bmatrix}$$

Linear Kernel

$k(x, x') = \lambda x^T x'$. For more than one new point $k(x, x')$ is a matrix like K_{AA}.

$$\mu'(x) = \mu(x) + k_{x,A}(K_{AA} + \sigma_n^2 I)^{-1}(y_A - \mu_A)$$
$$k'(x, x') = k(x, x') - k_{x,A}(K_{AA} + \sigma_n^2 I)^{-1} k_{x',A}^T$$
$$K_{AA} = \lambda X X^T$$
$$k_{x,A} = \begin{bmatrix} k(x_1, x) \\ \vdots \\ k(x_m, x) \end{bmatrix} = \lambda \begin{bmatrix} x_1^T x \\ \vdots \\ x_m^T x \end{bmatrix}$$

Online GP's

$K_{AA} = k(x_{t+1}, x_{t+1})$ then calculate the posterior for a new arbitrary data point $x*$.

Maximize the marginal likelihood of the data

$K(\theta)$ is the Kernel matrix.

$$\begin{aligned}
&\operatorname*{argmax}_\theta \int p(y_{train} \mid f, x_{train}, \theta)p(f \mid \theta)df \\
=&\operatorname*{argmax}_\theta \int \mathcal{N}(f(x), \sigma_n^2)\mathcal{N}(0, K(\theta))df \\
=&\operatorname*{argmax}_\theta \mathcal{N}(0, K(\theta) + I\sigma_n^2) \\
=&\operatorname*{argmax}_\theta p(y_{train} \mid x_{train}, \theta) \\
=&\operatorname*{argmin}_\theta -\log p(y_{train} \mid x_{train}, \theta) \\
=&\operatorname*{argmin}_\theta \frac{1}{2}(y(K(\theta) + I\sigma_n^2)^{-1}y + \log(\det K_y)
\end{aligned}$$

Laplace Approximation

In the context of Log. Regression.

$$\begin{aligned}
q(\theta) &= \mathcal{N}(\hat{w}, \Lambda^{-1}) \\
\hat{w} &= \operatorname*{argmax}_w p(w \mid y) \\
&= \operatorname*{argmax}_w \frac{1}{Z}p(w)p(y \mid w) \\
&= \operatorname*{argmin}_w \frac{1}{2\sigma_p^2}\|w\|_2^2 \\
&\quad + \sum_{i=1}^{n} \log(1 + e^{-y_i w^T x_i}) \\
\Lambda &= -\nabla\nabla \log p(\hat{w} \mid x, y) \\
&= X\ diag([\pi_i(1 - \pi_i)]_i)\ X \\
\pi_i &= \sigma(\hat{w}^T x_i)
\end{aligned}$$

Prediction

$$
\begin{aligned}
& p(y^* \mid x^*, X, y) \\
&= \int p(y^* \mid x^*, w) p(w \mid X, y) dw \\
&= \int p(y^* \mid x^*, w) q_\lambda(w) dw \\
&= \int p(y^* \mid f^*) p(f^* \mid w) q_\lambda(w) dw df^* \\
& q_\lambda(w) \sim N(\mu, \Sigma) \\
& p(f^* \mid w) = x^* \\
& \int p(f^* \mid w) q_\lambda(w) dw \\
&= N(\mu^T x^*, x^* \Sigma x^*) \\
& p(y^* \mid x^*, X, y) \\
&= \int p(y^* \mid f^*) N(\mu^T x^*, x^* \Sigma x^*) df^* \\
& p(y^* \mid f^*) = \sigma(y^* f^*)
\end{aligned}
$$

Variational Inference

KL divergence

Reverse KL div: $KL(q||p)$. Forward KL: $KL(p||q)$ (gives more conservative variance estimates).

$$KL(q||p) = \int q(\theta) \log \frac{q(\theta)}{p(\theta)} d\theta$$

$$
\begin{aligned}
& KL(p||q) \\
&= \frac{1}{2}\big(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) \\
& -d + ln(\frac{|\Sigma_1|}{|\Sigma_0|})) \\
& p = \mathcal{N}(\mu_0, \Sigma_0) \\
& q = \mathcal{N}(\mu_1, \Sigma_1)
\end{aligned}
$$

Minimizing KL divergence

$$
\begin{aligned}
& \operatorname*{argmin}_{q \in Q} KL(q||p(\theta|y)) \\
&= \operatorname*{argmax}_{q \in Q} \mathbb{E}_{\theta \sim q(\theta)}[\log p(\theta, y)] + H(q) \\
&= \operatorname*{argmax}_{q \in Q} \mathbb{E}_{\theta \sim q(\theta)}[\log p(y|\theta)] - KL(q||p(\theta))
\end{aligned}
$$

Gradient of the ELBO

$$
\begin{aligned}
& \nabla_\lambda \mathbb{E}_{\theta \sim q_\lambda}[f(\theta)] \\
&= \mathbb{E}_{\epsilon \sim \phi}[\nabla_\lambda f(g(\epsilon; \lambda))] \\
&= \nabla_{C,\mu} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[\log p(y|C\epsilon + \mu)] \\
&= n \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \\
& \quad \mathbb{E}_{i \sim \mathcal{U}(1,..,m)}[\nabla_{C,\mu} \log p(y_i|C\epsilon + \mu x_i)] \\
&= \frac{n}{m} \sum_{j=i}^{m} \nabla_{C,\mu} \log p(y_i|C\epsilon + \mu x_i)
\end{aligned}
$$

MCMC methods

Hoeffding's inequality

Given f is bounded between $[0, C]$:

$$P(|\mathbb{E}_P[f(X)] - \frac{1}{N}\sum_{i=1}^{N} f(x_i)| > \epsilon) \leq 2\exp^{\frac{-2N\epsilon^2}{C^2}}$$

Error less than ϵ with probability $1 - \delta$:

$$2\exp^{\frac{-2N\epsilon^2}{C^2}} \leq \delta$$

MH-MCMC

DBE: $Q(x)P(x'|x) = Q(x')P(x|x')$.

$$
\begin{aligned}
& R(X'|X = x) \\
& X_{t+1} = x', P(X_{t+1} = x') = \alpha \\
& \alpha = min\left\{1, \frac{Q(x')R(x|x')}{Q(x)R(x'|x)}\right\} \\
& \text{o.t.w } X_{t+1} = x
\end{aligned}
$$

Continuous RV

$$
\begin{aligned}
& p(x) = \frac{1}{Z} e^{-f(x)} \\
& \alpha = min\left\{1, \frac{R(x|x')}{R(x'|x)} e^{f(x) - f(x')}\right\}
\end{aligned}
$$

If $R(x'|x) = \mathcal{N}(x, \tau I)$ then $\alpha = min\left\{1, e^{f(x)-f(x')}\right\}$. Guaranteed efficient convergence for log-concave densities (e.g. f is convex).

Improved Proposals

Metropolis adjusted Langevin Algo (gradient to prefer proposals into regions with higher density), Stochastic Gradient Langevin Dynamics (stochastic gradient), Hamiltonian Monte Carlo (momentum).

Bayesian Neural Networks

MAP estimation with BNN's

$$
\begin{aligned}
\hat{\theta} &= \operatorname*{argmin}_\theta \; -\log p(\theta) - \sum_{i=1}^{n} \log p(y_i|x_i, \theta) \\
&= \operatorname*{argmin}_\theta \lambda||\theta||_2^2 \\
&+ \frac{1}{2}\sum_{i=1}^{n} \big[\frac{1}{\sigma(x_i,\theta)^2}||y_i - \mu(x_i,\theta)||_2^2 \\
&+ \log \sigma(x_i,\theta)^2\big]
\end{aligned}
$$

Variational Inference in BNN's

$$
\begin{aligned}
& p(y^* \mid x^*, X, y) \\
&= \int p(y^* \mid x^*, \theta) p(\theta \mid X, y) d\theta \\
&= \mathbb{E}_{\theta \sim p(\theta|X,y)}\big[p(y^* \mid x^*, \theta)\big] \\
&\approx \mathbb{E}_{\theta \sim q_\lambda}\big[p(y^* \mid x^*, \theta)\big] \\
&\approx \frac{1}{m}\sum_{j=1}^{m} p(y^* \mid x^*, \theta^{(j)}) \\
&= \frac{1}{m}\sum_{j=1}^{m} \mathcal{N}(\mu(x^*, \theta), \sigma^2(x^*, \theta))
\end{aligned}
$$

Uncertainty for Gaussians

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

MCMC in BNN's

$$p(y^* \mid x^*, X, y) \approx \frac{1}{m}\sum_{j=1}^{m} p(y^* \mid x^*, \theta^{(j)})$$

Dropout and Probabilistic Ensembles

$p(y^* \mid x^*, X, y) \approx \frac{1}{m}\sum_{j=1}^{m} p(y^* \mid x^*, \theta^{(j)})$

Calibration

Reliability Diagrams

If well calibrated $freq(B_m) = conf(B_m)$ for all bins.

$$
\begin{aligned}
& freq(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} 1(y_i = 1) \\
& conf(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} \hat{p}_i \\
& ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |freq(B_m) - conf(B_m)| \\
& MCE = \max_{m \in (1,...,M)} |freq(B_m) - conf(B_m)|
\end{aligned}
$$

Calibration Methods

Histogram binning: Assign calibrated score to each bin $\hat{q}_i = freq(B_m)$. **Isotonic regression**: Find piecewise constant function f, $\hat{q}_i = f(\hat{p}_i)$ that minimizes the bin-wise squared loss, by adjusting the bins. **Platt scaling**: Learn $a, b \in \mathbb{R}$ that minimize the NLL loss over

the validation set when applied to the logits z_i, $\hat{q}_i = \sigma(az_i + b)$. Temperature scaling for multiple classes uses single parameter T s.t. $\hat{q}_i = \max_k \sigma_{softmax}(z_i/T)^{(k)}$

Active Learning

Active learning refers to a family of approaches that aim to collect data that maximally reduces uncertainty about an unknown model. To quantify the reduction in uncertainty given a new observation we use MI. In the regression setting, where $Y = X + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma_n^2 I)$.

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

Choosing the optimal subset of observations of a given size is an NP-Hard problem, however we can greedily choose the one with the largest MI. MI is monotone sub modular which means, A) Information never hurts, B) there is diminishing returns as we add more data. It turns out that this approach provides constant-factor approximation that is near optimal (S is the optimal set of observations).

$$I(f(x_T), y_T) \geq \left(1 - \frac{1}{e}\right) \max_{|S| \leq T} I(f(x_S), y_S)$$

For $S_t = \{x_1, ..., x_t\}$ and following the same regression scheme as before.

$$
\begin{aligned}
x_{t+1} &= \operatorname*{argmax}_x \mathbb{I}(f; y_x | y_{S_t}) \\
&= \operatorname*{argmax}_x \frac{1}{2}\log\left(1 + \frac{\sigma_t^2(x)}{\sigma_n^2}\right)
\end{aligned}
$$

In the homeostatic case this optimization problem boils down to greedily assigning $x_{t+1} = \operatorname*{argmax}_{x \in D} \sigma_t^2(x)$. This approach is termed Uncertainty sampling. In the heteroscedastic case on must also take the aleatroic uncertainty into account $x_{t+1} = \operatorname*{argmax}_x \frac{\sigma_t^2(x)}{\sigma_n^2(x)}$.

Active Learning for Classification

In this setting uncertainty sampling corresponds to selecting samples that maximize entropy of the predicted label $x_{t+1} = \operatorname*{argmax}_x H(Y|x, X_t, Y_t)$. However most uncertain label is not necessarily most informative (specially when the noise is very high). One can us the MI instead and apply approximate Bayesian Learning. By calculating an approximate distribution for each new point one can sample it and estimate the MI value, then compare all of them and choose the point that maximizes it.

$$
\begin{aligned}
x_{t+1} &= \operatorname*{argmax}_{x \in D} \mathbb{I}(\theta; y_{t+1} | Y_t, X_t, x_{t+1}) \\
&= H(y_{t+1}|Y_t, X_t, x_{t+1}) - \mathbb{E}_{\theta \sim p(|X_t, Y_t)}[H(y_{t+1}|, \theta)] \\
&\approx H(y_{t+1}|Y_t, X_t, x_{t+1}) - \frac{1}{m}\sum_{j=1}^{m} H(y_{t+1}|, \theta^{(j)})
\end{aligned}
$$

Bayesian Optimization

How should we sequentially pick $x_1, ..., x_t$ to find $max_x f(x)$ with minimal samples (e.g. tradeing off exploration and exploration)? By defining the cumulative average regret we can gauge how well a specific exploration-exploration strategy is working. f^* is the somehow known maximum (perhaps in hindsight).

$$\frac{1}{T}\sum_{t=1}^{T}[f(x^*) - f(x_t)] \to 0$$

While this ideas generalize to other Bayesian Learning problems the focus was on GP's.

Upper confidence sampling

Method for GP, where we believe that the true unknown function is contained within the confidence bounds s.t. upper confidence bound $\geq$ best lower bound. Hence $x_{t+1} = \operatorname*{argmax}_{x \in D} \mu_t(x) + \beta_t \sigma_t(x)$. By choosing this strategy we only picks plausible maximizers. This optimization problem is generally non convex, however there are workarounds.

One can analyze how long it takes the algorithm to converge. This is parametrized by the maximal mutual information $\gamma_T = \max_{|S| <= T} I(f; y_S)$, which in turn depends on the nature of the function to be discovered. One can also adjust the β to insure that the true function is contained.

Improvement Based Acquisition Functions

Probability of Improvement

Provided a new point x and a so far best observed function value f^*, choose point based on the probability that the function evaluated at that value is higher than the maximum observed so far. It might favor points close the the so far observed maximum since the variance is small around it.

$$PI(x) = P(f(x) > f^*) = \Phi\left(\frac{\mu_t(x) - f^*}{\sigma_t(x)}\right)$$

Expected Improvement

Takes into account the magnitude of the potential improvement:

$$x_{t+1} = \operatorname*{argmax}_{x \in D} \mathbb{E}_{f_t(x) \sim \mathcal{N}(\mu_t, \sigma_t)}[max(0, f_t(x) -$$

Thomson Sampling

We sample a function from our function space and maximize it. The randomness in the sampling is sufficient for exploration. $\tilde{f} \sim \mathcal{P}(f|X_t, Y_t)$, and then $x_{t+1} \in \operatorname*{argmax}_{x \in D} \tilde{f}(x)$.

Markov Decision Processes

Expected Value of a Policy

For a determinnistic policy π and a given state x.

$$
\begin{aligned}
&= J(\pi|X_0 = x) \\
&= V^\pi(x) \\
&= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t))|X_0 = x] \\
&= r(x, \pi(x)) \\
&+ \gamma \sum_{x'} P(x'|x, \pi(x)) \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t))|X \\
&= r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^\pi(x')
\end{aligned}
$$

For all states this yields a system of equations that has a closed form.

$$V^\pi = r^\pi + \gamma T^\pi V^\pi$$
$$V^\pi = (I - \gamma T^\pi)^{-1} r^\pi$$

Alternatively one can solve the linear system approximately by fixed point iteration (Loop T times s.t. $V_t^\pi = r^\pi + \gamma T^\pi V_{t-1}^\pi$). This provides computational advantages for sparse solutions.

Policy Iteration

In a tabular setting one can compute the optimal policy with this method. Convergence to the optimal policy in $O * (n^2 m/(1-\gamma))$ iterations.

1. Start with an arbitrary (e.g., random) policy π
2. Until converged do:
   - Compute value function V^π by solving the system of equations.
   - Compute greedy policy $\pi_V(x) = \underset{a}{\text{argmax}}\ r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V(x')$ w.r.t. the previously computed V^π.
   - Set $\pi \leftarrow \pi_V$.

Value Iteration

Can show that converges since the Bellman update is a contraction. Much computationally cheaper specially for sparse MDP's. In practice, which works better depends on application.

1. Initialize $V_0(x) = \max_a r(x, a)$
2. For $t = 1$ to ∞
   - For each x, a, $Q_t(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, \pi(x)) V_{t-1}(x')$
   - For each x, $V_t(x) = \max_a Q_t(x, a)$
   - Break if $\max_x |V_t(x) - V_{t-1}(x)| \leq \epsilon$

POMDP's

In contrast to MDPs the whole state space is not known. Only obtain noisy observations Y_t of the hidden state X_t. Given an action a_t at time t, the new state has probability $P(X_{t+1} = x'|x_t, a_t)$ and we observe $y_t \sim P(Y_t|X_t = x_t)$. Belief states explodes even for finite horizons. This implies a computational hurdle for POMDP's even thou we have the same theoretical understanding of them than MDP's.

State update and reward function:

$$\begin{aligned} &= b_{t+1}(x) \\ &= P(X_{t+1} = x'|y_{t+1}) \\ &= \frac{1}{Z} \sum_{x'} b_t(x) P(X_{t+1} = x'|x', a_t) P(y_{t+1}|x) \end{aligned}$$

$$r(b_t, a_t) = \sum_x b_t(x) r(x, a_t)$$

Reinforcement Learning

- Episodic setting: Agent learns over multiple training episodes or trajectories. The environment resets after each episode.
- Non-episodic: Agent learns "online", yielding a single trajectory.
- On-policy: Agent has full control over which actions to pick.
- Off-policy: Agent has no control over actions, only gets observational data (e.g., demonstrations, data collected by applying a different policy, ...).
- Stationary Value Function: at any time-step in the episode $V_i^\star(x) = V_j^\star(x)$
- Stationary Policy: A stationary policy is the one that does not depend on time (e.g. for a given state the optimal action is the same regardless of the time step at which the state is visited).

Model Based RL

Learn the MDP, e.g. estimate transition probabilities, estimate reward function r(x, a), optimize policy based on estimated MDP, repeat. Using MLE one can estimate the transition probabilities and the reward function given a set of episodes. This applies to the tabular a non tabular setting (section after model free RL).

$$\hat{P}(X_{t+1}|X_t, A) = \frac{Count(X_{t+1}, X_t, A)}{Count(X_{t+1}, A)}$$
$$\hat{r} = \frac{1}{N_{x,a}} \sum_t R_t$$

However one still needs to decide on how to explore the state space. Ideally we would like to explore but biased towards solution spaces that would give us high rewards. Pure greedy exploration will not work because it might be blind to optimal solutions that are not greedy. One possible solution is to do ϵ greedy exploration; with probability ϵ_t pick random action and with probability $(1 - \epsilon_t)$ pick best action.

Rmax Algorithm

Can rule out clearly sub-optimal actions very quickly. It can be very memory expensive since we have to keep a counter for every state action pair and computationally expensive since after each episode we use value or policy iteration.

1. Initially:
   - add fairy tale state $x^\star$
   - set $r(x, a) = Rmax$ for all states x and actions a
   - set $P(x^\star|x, a) = 1$ for all (x, a)
   - choose optimal policy for r and P
2. Loop:
   - Execute policy π
   - for each visited state action pair update $r(x, a)$
   - estimate transition probabilities $P(x'|x, a)$
   - if observed "enough" transitions / rewards
   - recompute policy π according to current model P and r.

Model Free RL

Estimate the value function directly.

TD-Learning

On-policy method. Given any policy π, want to estimate $V^\pi(x)$. I observe episodes, each has a number of time-steps. Each time I observe a state I can update its value function given the observed reward and the value function at the next state by averaging (helps reduce variance) it with the past value function. Guarantees convergence (conditional on α_t) of the value function given a policy, not the optimal policy. Hence its just a replacement of something like value iteration.

$$\hat{V}^\pi(x) = (1-\alpha_t)\hat{V}^\pi(x) + \alpha_t(r + \gamma\hat{V}^\pi(x'))$$

Tabular TD-learning update rule can be viewed as stochastic gradient decent on the squared loss, where we use old value estimates as labels/targets ($r + \gamma V(x';\theta_{old}) = y$). Same insight applies for the $Q(x,a)$.

$$l_2(\theta; x, x', r) = \frac{1}{2}(V(x,\theta) - r - \gamma V(x';\theta_{old}))^2$$

Q-learning

Off-policy. We have a behavioral policy that we use to collect the data. Here we aim to estimate the optimal policy. We can mimic the idea of optimistically initialization by initializing the Q to very large values. For Q learning to converge you have to visit all state action pairs inf many times. If you do greedy and optimistic initialization then it converges. Otherwise you have to strike a trade off with some epsilon greedy exploration strategy.

$$\begin{aligned} Q^\star(x,a) &= r(x,a) + \gamma \sum_{x'} P(x'|x,a)V^\star(x') \\ V^\star(x) &= \max_a Q^\star(x,a) \\ Q^\star(x,a) &\leftarrow (1-\alpha_t)Q^\star(x,a) + \alpha_t(r + \gamma \max_{a'} Q^\star(x',a')) \end{aligned}$$

We require memory for each state action pair, which becomes unfeasible for continues state spaces.

Approximating value functions

Linear function approximation, where $\phi(x,a)$ is a set of hand designed features.

$$\begin{aligned} \hat{Q}(x,a;\theta) &= \theta^T\phi(x,a) \\ l_2(\theta; x, a, x', r) &= \frac{1}{2}(Q(x,a,\theta) - r - \gamma \max_{a'} Q(x',a';\theta_{old}))^2 \\ \delta &= Q(x,a,\theta) - r - \gamma \max_{a'} Q(x',a';\theta_{old}) \\ \theta &\leftarrow \theta - \alpha_t \delta \nabla_\theta Q(x,a;\theta) \\ \theta &\leftarrow \theta - \alpha_t \delta \phi(x,a) \end{aligned}$$

In this setting the natural thing is an online algorithm where at each step we update θ according to the rules above. This vanilla version leads to a lot of variance in the estimates of the target values. To counter this one might want to keep the target values constant across episodes (e.g. replay buffer or twin network).

$$L(\theta) = \sum_{(x,a,r,x') \in D} (Q(x,a,\theta) - r - \gamma \max_{a'} Q(x',a';\theta_{old}))^2$$

Double DQN allows us to avoid maximization bias (overconfidence about certain actions given the noise in the observations) by maximizing over the actions w.r.t. the current network instead of the old one (WTF does this mean?). Nevertheless the maximization remains intractable for continues action spaces.

Policy search methods

Learning a parameterized policy. Rollouts are trajectories that are distributed according to a parametrized policy. We can calculate the reward for a particular roll out. We can estimate the expected reward of the policy by averaging the reward of each roll out (MC sampling). Then we aim to find optimal policy parameters through global optimization of the reward expectation.

$$\begin{aligned} \pi(x) &= \pi(x,\theta) \\ r(\tau^{(i)}) &= \sum_{t=0}^{T} \gamma^t r_t^{(i)} \\ J(\theta) &\approx \frac{1}{m} \sum_{i=1}^{m} r(\tau^{(i)}) \\ \theta^\star &= \operatorname*{argmax}_\theta J(\theta) \\ \nabla_\theta J(\theta) &= \nabla \mathbb{E}_{\tau\sim\pi_\theta} r(\tau) \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\nabla \log \pi_\theta(\tau)] \end{aligned}$$

This approach gives unbiased estimates but they might have a huge variance. Baselines help to reduce this variance.

REINFORCE Algorithm

$$\begin{aligned} &= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} r(\tau)\nabla \log \pi_\theta(a_t|x_t;\theta)] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} (r(\tau) - b(\tau_{0:t-1}))\nabla \log \pi \end{aligned}$$

$$b(\tau_{0:t-1}) = \sum_{t'=0}^{t-1} \gamma^{t'} r_{t'}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \gamma^t G_t \nabla \log \pi_\theta(a_t|x_t;\theta)]$$

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$$

1. Initialize policy weights $\pi(a|x;\theta)$
2. Repeat:
   - Generate an episode.
   - For every timestep set G_t to the return from step t.
   - Update $\theta \leftarrow \theta + \eta\gamma^t G_t \nabla_\theta \log \pi(A_t|X_t;\theta)$

Vanilla policy search methods are slow. We can combine it with value function estimation (actor critic methods) to improve it.

Policy Gradient Theorem

Can represent policy gradients in terms of Q-function. Naturally suggests plugging in approximations of Q for the action-value function. Actor-Critic algorithms combine a parametrized policy (actor) and a value function approx. (critic). This approximations are usually done using neural nets in Deep RL.

Online Actor Critic

TD-learning for the critic. Under some special conditions is guaranteed to improve.

$$\theta_\pi \leftarrow \theta_\pi + \eta_t Q(x,a;\theta_Q)\nabla \log \pi(a|x;\theta_\pi)$$
$$\theta_Q \leftarrow \theta_Q - \eta_t(Q(x,a;\theta_Q) - r - \gamma Q(x', \pi(x',\theta_\pi);\theta_Q))\nabla Q(a|x;\theta_\pi)$$

Advantage Active Critique

Variance reduction via baseline (value function). This technique can be combined with Monte-Carlo. Algorithms: A2C, A3C, GAE/GAAC.

$$\theta_\pi \leftarrow \theta_\pi + (\eta_t Q(x,a;\theta_Q) - V(x;\theta_V))\nabla \log \pi(a|x;\theta_\pi)$$

TRPO and PPO

Modern variants of policy gradient / actor critic methods. Trust-region policy optimization (TRPO). The intuition behind this is that it only optimizes in close by regions where the previously collected data is still valid. PPO is an effective heuristic variant.

Off-Policy Actor Critic

Our initial motivation was intractability of the max argument because of large action spaces. The maximization problem can be compiled into the one of training a prametrized policy. Then by using a differentiable approximation Q and differentiable deterministic policy π we can use chain rule (backpropagation) to obtain stochastic gradients. In order to ensure exploration one can add action noise. One can also obtain gradients for reparametrizable stochastic policies ($a = \phi(x, \theta_\pi, \epsilon)$).

$$\max_a Q(x', a', \theta^{old}) \approx Q(x', \pi(x';\theta_\pi); \theta_Q^{old})$$
$$\nabla_\theta \hat{J}_\mu(\theta) = \mathbb{E}_{x\sim\mu}[\nabla_\theta Q(x, \pi(x;\theta);\theta_Q)]$$
$$\nabla_{\theta_\pi} Q(x, \pi(x;\theta_\pi);\theta_Q)$$
$$= \nabla_a Q(x,a)|_{a=\pi(x;\theta_\pi)} \nabla_{\theta_\pi}\pi(x;\theta_\pi)$$

DDPG (deterministic policy gradients): combines DQN with reparametrization policy gradients. TD3: extension of DDPG to avoid maximization bias. SAC: variant of DDPG/TD3 for entropy regularized MDPs.

Model-based Deep RL

The primary appeal with model based techniques is that potentially they would be much more effective interacting with the environment. Learning a model can help dramatically reduce the sample complexity compared to model-free techniques (though not necessarily computational cost).

Planning in the known model

Plan over finite horizon (but for potentially infinite and known MDP's), carry out first action, then re-plan. Challenges (especially for large H): Local minima, Vanishing / exploding gradients. One limitation of this approach is that given a finite horizon we might still have no signal to follow. To solve this, one can use estimates of the value function to guide decision taking (the tail of the sum of the discounted rewards).

$$J(a_{t:t+H-1}) \triangleq \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}), a_\tau) + \gamma^H V(x_{t+H})$$

In the stochastic transition setting we can apply policy for the next t time steps by choosing the sequence of actions that maximizes the expectation over the randomness in the model, but also re plan after each action (planning via model predictive control MPC). One common approach to do this is MC trajectory sampling since computing this expectation is usually hard. Otw its just function composiotion of how x_τ is determined by $a_{t:\tau-1}$ and $\epsilon_{t:\tau-1}$.
We can also replace expensive online planning (re plan at each step) by offline training (optimizing) of a policy (deterministic or stochastic) that is fast to evaluate online. This leads to a generalization of DDPG where we don't absorve all the look ahead into the Q value but also include the discounted reward over the finite horizon H. This look-ahead helps policies improve much more rapidly, by using the model to anticipate consequence down the road.

$$J(\theta) = \mathbb{E}_{x\sim\mu}[\sum_{\tau=0:H-1} \gamma^\tau r_\tau + \gamma^H Q(x_H, \pi(x_H;\theta);\theta_Q)|\theta]$$

Unknown Dynamics

If we don't know the dynamics and reward, can estimate them off-policy with standard supervised learning techniques where $(r_i, x_{i+1}) \sim f(x_i, a_i; \theta)$.
If we were to model the dynamics as Gaussian through a NN, errors in the estimated model compound when planning over multiple time steps which can lead to very poor performance. Therefore there is a need for capturing uncertainty in the estimated model. We can do this by modeling f as a GP of a BNN.

$$\hat{J}_H(a_{t:t+H-1}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r_\tau(x_\tau(a_{t:\tau-1}, \epsilon^{(i)}_{t:\tau-1}, f^{(i)}), a_\tau) + \gamma^H V($$

Greedy exploitation for model-based RL

PETS algorithm, using NN and moment matching instead of MC averaging. Event thou this algorithm does not explicitly encourage exploration it already performs very well. To encourage further exploration one can add Gaussian noise.

- $D = []$; prior $P(f) = P(f|[])$, then iterate the following:
- Plan new policy $\max_\pi \mathbb{E}_{f\sim P(\cdot|D)} J(\pi, f)$.
- Roll out π and add collected data to D.
- Update posterior $P(f|D)$.

Thompson Sampling

- $D = []$; prior $P(f) = P(f|[])$, then iterate the following:
- Sample model from $f \sim P(\cdot, D)$
- Plan new policy $\max_\pi \mathbb{E}_{f\sim P(\cdot|D)} J(\pi, f)$.
- Roll out π and add collected data to D.
- Update posterior $P(f|D)$.

Optimistic Exploration

Consider a set $M(D)$ of models (MDP's) that are plausible given data D:

- $D = []$; prior $P(f) = P(f|[])$, the iterate the following:
- Plan new policy $\max_{\pi} \max_{f \in M(D)} \mathbb{E}_{\mathcal{O} \sim \mathbb{P}(\cdot|\mathbb{D})} J(\pi, f)$
- Roll out π and add collected data to D.
- Update posterior $P(f|D)$.

In general, the joint maximization over π and f is very difficult but H-UCLR gets around this somehow, by adding desition variables that allow you to "control your luck", e.g. get to states with higher rewards. This algorithm outperforms Thomson sampling and PETS specially in constrained problems, since these contains disincentive exploration.