Math

Bayes Theorem

$P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}$

Conditional Probability

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

PDF's and CDF's

$$F'(x) = f(x)$$
$$F(x) = \int_0^x f(t)dt$$

Marginalization

a.k.a the Sum Rule

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

Product Rule

$$\begin{aligned}\mathrm{P}(X_3, X_2, X_1) &= \mathrm{P}(X_3 \mid X_2, X_1) \cdot \mathrm{P}(X_2, X_1) \\ &= \mathrm{P}(X_3 \mid X_2, X_1) \cdot \mathrm{P}(X_2 \mid X_1) \cdot \mathrm{P}(X_1)\end{aligned}$$

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

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

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

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

Univariate Laplacian

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

Normal Random Vectors

$X_1, X_2, ..., X_n$ are said to be jointly normal if, for all $a_1, a_2, ..., a_n \in \mathbb{R}$, the random variable $a_1X_1 + a_2X_2 + ... + a_nX_n$ is a normal random variable. Several jointly normal random variables in a vector is called a Normal Random Vector.

Conditional Random Vectors

$X \sim \mathcal{N}(\mu_V, \Sigma_{VV})$ where $V = [1, ..., d]$ is the index set. Given two disjoint subsets of V, $A = \{i_1, ..., i_k\}$ and $B = \{j_1, ..., j_m\}$; the conditional distribution $P(X_A \mid X_B = x_b) = \mathcal{N}(\mu_{A|B}, \Sigma_{A|B})$ s.t

$$\begin{aligned}\mu_{A|B} &= \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(x_B - \mu_B) \\ \Sigma_{A|B} &= \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{AB}\end{aligned}$$

$$\Sigma_{AB} = \begin{bmatrix}\sigma_{i_1,j_1} & \dots & \sigma_{i_1,j_m} \\ \vdots & \ddots & \vdots \\ \sigma_{i_k,j_1} & \dots & \sigma_{i_k,j_m}\end{bmatrix}$$

Hierarchical Bayesian Modeling

Endow the prior with an aditional prior probability distribution. This can be done multiple times.

Empirical Bayes

Treat the prior as a contant.

Multiples of Gaussians are Gaussian

Given $M \in \mathbb{R}^{m\times d}$ then $Y = MX \sim \mathcal{N}(M\mu_V, M\Sigma_{VV}M^T)$, where $X \in \mathbb{R}^d$ and is a GRV.

Sums of Gaussians are Gaussian

$Y = X + X' \sim \mathcal{N}(\mu_V + \mu'_V, \Sigma_{VV} + \Sigma'_{VV})$, where X is a GRV.

Conditional Linear Gaussians

if X and Y are jointly Gaussian then $P(X \mid Y = y)$ is Gaussian with mean linearly dependent on y. X can be viewed as a linear function of Y with independent Gaussian noise added.

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

Uses full posterior distribution over the weights rather than the mode only (Ridge regression equivalent to MAP estimate of the weights). This accounts for the epistemic uncertainty in the model, as well as the aleatoric one σ_n which is a hyperparameter here.

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

Prediction

$f^* = w^Tx^*$ where w^T is the posterior over the weights. By integrating out all possible models one can obtain $p(f^*|X,y,x^*) = \mathcal{N}(\bar{\mu}^Tx^*, x^{*T}\bar{\Sigma}x^*)$, since multiples of Guassians ara Gaussian. Then by considering that $y^* = f^* + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ and that sums of Gaussians are Gaussians, then: $p(y^*|X,y,x*) = \mathcal{N}(\bar{\mu}^Tx^*, x^{*T}\bar{\Sigma}x^* + \sigma_n^2)$.

Online Bayesian Linear Regression

We update $\bar{\mu}$ and $\bar{\Sigma}$ every time we get a new observation. x_i is a column vector. There is a way not to compute the inverse in each iteration.

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

MLE

Standard Least Squares $w_{MLE} = (X^T X)^{-1} X^T y$

MAP

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

Gaussian Proccesess

A $GP(\mu, k)$ is charachterized by a covariance kernell function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, and a mean function "$\mu : X \to \mathbb{R}$". for each $x \in X$ there is a random variable f_x s.t. $p(f_x) = \mathcal{N}(\mu(x), k(x,x))$. For a collection of data points $f_A \sim \mathcal{N}(\mu_A, K_{AA})$ where:

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

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

Linear Kernel

GP with linear kernel = Bayesian linear regression. $k(x,x') = \lambda x^T x'$

Predictions

Given that we observe $y_i = f(x_i) + \epsilon_i$ s.t. $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Generally mean function is initially set to zero which makes $\mu(x)$ and μ_A disappear from the equations below. For a new point x^*, $x = x^*$ and $x' = x^*$. If there is more than one new point $k(x,x')$ can be a matrix analogous to 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

We condition on our new data point such that $K_{AA} = k(x_{t+1}, x_{t+1})$ calculate the posterior for a new arbitrary data point $x*$

Optimizing Kernel hyper-parameters

- CV: given a set of possible hyper-parameters, train a model then test it and compare the performance. Here we optimize over f to get one particular function.

- Maximize the marginal likelihood of the data. The idea is that by averaging out over functions there is an incentive to balance out simplicity and complexity s.t. a function with the right level of complexity is found. This is a model section technique, but validation is still needed. Over fitting still possible if parameter space is too complex. $K(\theta)$ is the Kernel matrix.

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

Fast GP methods

- Exploiting parallelism (GPU computations)

- Local GP methods. Covariance functions that decay with distance. Still expensive if "many" points close by.

- Kernel function approximations. construct explicit "low-dimensional" feature map that approximates the true kernel function.

- Inducing point methods. Only need accurate representation for training points, not uniformly well like in Kernell approximation. So we essentially reduce the data set by inducing points.

Laplace Approximation

Gaussian approximation to the posterior by first calculating the mean (MAP of the Bayesian Logistic Regression) and then the curvature (using second-order Taylor expansion around the posterior mode). How good this approximation is depends on how close the target distribution resembles a Gaussian.

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

Prediction

We can reduce the multidimensional integral problem to a one dimensional one that can be solved using numerical approximation. The last line is specific to Logistic Regression.

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

Variational Inference

Only for Gaussian the integration problem has a closed form solution. Variational inference seeks to approximate the intractable distribution p by a simple one q that is as close as possible. It differs from the Laplace approx. in that this one only seeks to approximate the mode and the variance in a second step. VI seeks to approximate the target distribution more generally. In the case where our variational family is the Gaussians we optimize the mean and variance jointly.

The caveat with this method is that on can say something about the lower bound on the evidence, but other than that it is hard to say something about the quality of the approximation.

KL divergence

Usually optimize the reverse KL divergence for computational reasons, $KL(q||p)$, even thou the forward one, $KL(p||q)$, gives more conservative variance estimates.

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

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

Minimizing KL divergence

Two interpretations. The first is maximizing the joint probability while also maximizing the entropy, e.g. the uncertainty. The second is maximizing the data likelihood as a constrained optimization problem where the KL divergence w.r.t. the prior is minimized. This prevents the estimate from collapsing to the MLE estimate. One can arrive at the same second expression by maximizing the lower bound on the evidence. This is a general derivation for any model (e.g. Bayesian Logistic regression), where we only need to plug in our expressions for the likelihood and the prior.

$$
\begin{aligned}
&= \underset{q \in Q}{\operatorname{argmin}}\ KL(q||p(\theta|y)) \\
&= \underset{q \in Q}{\operatorname{argmax}}\ \mathbb{E}_{\theta \sim q(\theta)}[\log p(\theta, y)] + H(q) \\
&= \underset{q \in Q}{\operatorname{argmax}}\ \mathbb{E}_{\theta \sim q(\theta)}[\log p(y|\theta)] - KL(q||p(\theta)) \\
p(\theta|y) &= \frac{1}{Z} p(\theta, y)
\end{aligned}
$$

Gradient of the ELBO

There is a need to reparametrize the likelihood such that the expectation is made over different variables. This allows one to push through the gradient into the expectation. Once we do this we can approximate the expectations with random sampling. $\nabla_{C,\mu} KL(q_{C,\mu}||p(C,\mu))$ can be computed exactly.

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

Markov-Chain Monte Carlo methods

Same motivation as for VI. We are interested in prediction but we can simply use the sum rule because the corresponding integrals are intractable. Hence we seek to approximate p (the posterior) by approximate samples constructed by simulating a Markov chain. The Ergodic theorem (law of large numbers for Markov chains) tells us that we converge to the right quantity (even thou samples are not independent). Here D is the state space.

$$\mathbb{E}[f(X)] \approx \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \sum_{x \in D} \pi(x) f(x)$$

Given that we have access to independent samples from the posterior, Hoeffding's inequality tells us that the error decreases exponentially with N given that f is bounded between $[0, C]$. In MCMC samples are positively correlated, so the error typically does not decrease at the same rate as when the samples are independent.

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

If we want the error to be less than ϵ with probability $1-\delta$:

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

Given an unnormalized distribution $P(x) = \frac{1}{Z}Q(x)$, where $Q(x)$ is easy to evaluate but Z is intractable, we can create a Markov chain that is efficient to simulate and has stationary distribution $P(x)$. Such Markov chain would have to be ergodic (there exists a finite t such that every state can be reached from every state in exactly t steps), which ensures uniqueness of a positive stationary distribution, which is independent from the starting distribution. All elements of the transition probability matrix strictly greater than 0 is a sufficient, but not necessary, condition for ergodiciy.
For the Markov chain to have the wanted stationary distribution it has to satisfy the detailed balanced equation: $Q(x)P(x'|x) = Q(x')P(x|x')$. The Metropolis Hastings is designed such that the detailed balance condition is satisfied. The general algorithm goes as follows; given a proposal distribution $R(X'|X = x)$ and initial state $X_t = x$. $X_{t+1} = x'$ with portability $\alpha = min\left\{1, \frac{Q(x')R(x|x')}{Q(x)R(x'|x)}\right\}$. Otw $X_{t+1} = x$ and repeat.

Gibbs Sampling

A specific way to update the chain (proposal dist.) for multivariate state spaces (e.g. a random vector). Given an initial assignment we pick one (or a block) of dimensions and update it by sampling from $P(X_i|x_{-i})$. This is easy to in a lot of modles (one dimentional integral). If we update one dimension at the time this satisfies detailed balance equation. In a block fashion it doesn't but it still yields the correct stationary distribution.

Continuous RV

MCMC techniques can be generalized to continuous random variables $p(x) = \frac{1}{Z}e^{-f(x)}$, where f is called the energy function. Then $\alpha = min\left\{1, \frac{R(x|x')}{R(x'|x)}e^{f(x)-f(x')}\right\}$. If $R(x'|x) = \mathcal{N}(x, \tau I)$ then $\alpha = min\left\{1, e^{f(x)-f(x')}\right\}$. There is guaranteed efficient convergence for log-concave densities (e.g. f is convex).

Improved Proposals

- Metropolis adjusted Langevin Algo. $R(x'|x) = \mathcal{N}(x - \tau\nabla f(x), 2\tau I)$
- Stochastic Gradient Langevin Dynamics. Improves efficiency. Can improve by taking into account local topology.
- Hamiltonian Monte Carlo algorithm add momentum.

Bayesian Neural Networks

We can improve performance by allowing nonlinear dependencies between parameters and inputs as opposed to Bayesian linear and logistic regression.
Assuming Gaussians in the likelihood and prior in a regression scenario the likelihood would look like $p(y|x,\theta) = \mathcal{N}(f_1(x,\theta), f_2(x,\theta))$ instead of $\mathcal{N}(\theta^T x, \sigma^2)$. The output for such network would be a mean and variance for a given input x. Modeling the variance as a function of the parameters and the inputs allows us to account for possible heteroscedasticity (attenuateing the losses for certain data points by attributing the error to large variance). Solving this optimization problem is equivalent to performing MAP on the weights of the network, by performing SGD (or any variant) on corresponding loss function.

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

Variational Inference in BNN's

Motivation: we want to account for all possible models not just the mode of the θ distribution (as in MAP estimation). By integrating over such distributions we get an estimate of the uncertainty in the predictions.

Given variational posterior q, can approximate predictive distributions by sampling m sets of weights and averaging the neural network predictions. For Gaussian likelihoods, approximate predictive distribution becomes a mixture of Gaussians.

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

In a mixture of Gaussian we can model, via the law of total variance, the aleatoric and epistemic uncertainty.

MCMC in BNN's

If your posterior distribution is not well represented by a mixture of Gaussians then MCMC might be a better option. The MC produces a sequence of weights for the network and we simply average their predictions.

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

In order to avoid storing all m samples one can subsample (e.g. only take 10% of the states visited). Another option is Stochastic Weight Averaging Gradients (SWAG). This creates a distribution q with the mean and sd of the parameters visited. Then weights are sampled from distribution q as in VI.

Special Inference Techniques for BNNs

- Dropout can be viewed as performing variational inference with a particular variational family, where one has to perform dropout also during evaluation. $p(y^* \mid x^*, X, y) \approx \frac{1}{m}\sum_{j=1}^{m} p(y^* \mid x^*, \theta^{(j)})$ where each j corresponds to an evaluation of the stochastic computational graph.

- Probabilistic Ensembles of NNs. Pick a data set D_j of n points uniformly at random from data set D with replacement. Then obtain MAP estimate for the weights. Finally average the prediction of the m sets of weights; $p(y^* \mid x^*, X, y) \approx \frac{1}{m}\sum_{j=1}^{m} p(y^* \mid x^*, \theta^{(j)})$.

Calibration

For a well-calibrated classifier, the predicted probability of a class for a given sample matches the true likelihood of the sample actually belonging to this class. This likelihood is not always correlated to the accuracy of a model, e.g. a model can have good accuracy but give un-calibrated probability estimates.

Reliability Diagrams

Group predictions into bins (of equal intervals or of equal size). Then calculate relative frequency of positive samples for each bin where B_m is the set of samples falling into bin m. If well calibrated $freq(B_m) = conf(B_m)$ for all m.

$$freq(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} 1(y_i = 1)$$
$$conf(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} \hat{p}_i$$

The degree of calibration can be quantified with the Expected Calibration Error (ECE) or the Maximum Calibration Error (MCE):

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |freq(B_m) - conf(B_m)|$$
$$MCE = \max_{m \in (1,\dots,M)} |freq(B_m) - conf(B_m)|$$

Calibration Methods

- Histogram binning: Divide uncalibrated predictions $\hat{p}_i$ into bins $B_1, .., B_M$. 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 parameters $a, b \in \mathbb{R}$ 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.

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

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 withing the confidence bounds s.t. upper confidence bound $\geq$ best lower bound. Hence $x_{t+1} = \underset{x\in D}{\operatorname{argmax}}\ \mu_t(x) + \beta_t\sigma_t(x)$. 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.

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

Expected Improvement

Takes into account the magnitude of the potential improvement:

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

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

Markov Decision Processes

Expected Value of a Policy

For a determinnistic policy π and a given state x.

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

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

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

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 $O*(n^2m/(1-\gamma))$ 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 $\pi_V(x) = \underset{a}{\operatorname{argmax}}\ r(x,\pi(x)) + \gamma\sum_{x'} P(x'|x,\pi(x))V(x')$ 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:

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

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

$$\begin{aligned}
&= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} r(\tau)\nabla\log\pi_\theta(a_t|x_t;\theta)]\\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}(r(\tau) - b(\tau_{0:t-1}))\nabla\log\pi_\theta(a_t|x_t;\theta)]\\
b(\tau_{0:t-1}) &= \sum_{t'=0}^{t-1}\gamma^{t'} r_{t'}\\
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\gamma^t G_t\nabla\log\pi_\theta(a_t|x_t;\theta)]\\
G_t &= \sum_{t'=t}^{T}\gamma^{t'-t} r_t
\end{aligned}$$

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.

$$\begin{aligned}
\theta_\pi &\leftarrow \theta_\pi + \eta_t Q(x,a;\theta_Q)\nabla\log\pi(a|x;\theta_\pi)\\
\theta_Q &\leftarrow \theta_Q\\
&\quad - \eta_t(Q(x,a;\theta_Q) - r - \gamma Q(x',\pi(x',\theta_\pi);\theta_Q))\\
&\quad\quad \nabla Q(a|x;\theta_\pi)
\end{aligned}$$

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 ($a = \phi(x,\theta_\pi,\epsilon)$).

$$\max_a Q(x', a', \theta^{old}) \approx Q(x', \pi(x'; \theta_\pi); \theta_Q^{old})$$

$$\nabla_\theta \hat{J}_\mu(\theta) = \mathbb{E}_{x\sim\mu}[\nabla_\theta Q(x, \pi(x;\theta); \theta_Q)]$$

$$\begin{aligned} &\nabla_{\theta_\pi} Q(x, \pi(x;\theta_\pi); \theta_Q) \\ &= \nabla_a Q(x,a)|_{a=\pi(x;\theta_\pi)} \nabla_{\theta_\pi} \pi(x;\theta_\pi) \end{aligned}$$

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.

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

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 $\max_\pi \max_{f\in M(D)} \mathbb{E}_{\mho\sim\mathbb{P}(\cdot|\mathbb{D})} J(\pi, f)$.
- 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.