# 1 Probability

**Normalization:** $P(\Omega) = 1$, **Non-negativity:** $P(A) \geq 0$
**Additivity:** $\forall A_i$ disjoint, $P(\cup_i^\infty A_i) = \sum_i^\infty P(A_i)$

**Independence:** $P(X_1 = x_1, .., X_n = x_n) = P(x_1), ..., P(x_n)$
Strong requirement, leaves us f.e. in Bayes with $P(X, Y) = P(X)P(Y|X) = P(X)P(Y)$

$P(X = x, Y = y|Z = z) = P(X = x|Z = z)P(Y = y|Z = z)$

Dimensionality problem: a probability distribution $P(X_1 = x_1, .., X_n = x_n)$ needs $2^n - 1$ parameters to be specified. Marginalizing to one distribution is also huge, its n-1 sums over all values.
**Product Rule:** $P(A, B) = P(A|B)P(B) = P(B|A)P(A)$
$P(X_1, .., X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1X_2)..P(X_n|X_1..X_{n-1})$

**Sum rule:** $P(X_{1:i-1}, X_{i+1:n}) = \sum_{x_i} P(X_{1:i-1}, x_i, X_{i+1:n})$
This is also called marginalization. If we have a joint probability distributions with many individual distributions, we can reduce the number of individual distributions by marginalizing out. The intuition behind this is based on the law of total probability. If we have two joint distributions. By summing up all outcomes of one of them, we have all events of this distribution covered in the marginalized distribution. This then only depends on the other variables.
**Bayes Rule:** $P(X|Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(X)P(Y|X)}{\sum_{X=x} P(X=x)P(Y|X=x)}$
Given the prior and the likelihood, Bayes rule can be used to calculate the posterior. If we are looking for a probability distri-' ution, we must also calculate the normalizer, by marginalising out all variables except Y, which can be done by knowing prior and likelihood, but can be intractable

## 1.1 Gaussian

$\mathcal{N}(x; \Sigma, \mu) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))$, $O(n^2)$ var
**Cov($\mathbf{x}_i$, $\mathbf{x}_j$):** $\mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)]$
The Gaussian is very important due to the ease of manipulation and the fact, that all distributions are driven towards a gaussian as stated in the central limit theorem.
**Dimensionality:** For a usual distribution with n parameters, e.g. binary $P(X_1, .., X_n)$, we need $2^n - 1$ variables. With a compositional Gaussian only $O(n^2)$.
**Marginalization:** The cost of marginalisation is also huge with standart (binary distributions), need to sum up all combinations of other variables $O(2^{n-1})$.

**Marginal:** $X_A = [X_{i_1}, ..., X_{i_k}] \sim \mathcal{N}(\mu_A, \Sigma_{AA})$
**Conditional:** $p(X_A|X_B = x_B) = \mathcal{N}(\mu_{A|B}, \Sigma_{A|B})$
$\mu_{A|B} = \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(x_B - \mu_B)$, $\Sigma_{A|B} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}$
**Times:** $M \in \mathcal{R}^{m \times d}, MX \sim \mathcal{N}(M\mu_X, M\Sigma_{XX}M^T)$
If we have a scalar instead of M, it reduces to $X_1 s \sim \mathcal{N}(s\mu_1, s^2\Sigma_1)$
**Add:** $X_1 + X_2 \sim \mathcal{N}(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2), X_1 + s \sim \mathcal{N}(\mu_1 + s, \Sigma_1)$

# 2 Matrix manipulation

$\nabla(\mathbf{a}^T\mathbf{w}) = a$ $\nabla(\mathbf{w}^T\mathbf{B}\mathbf{w}) = 2Bw$

$\sum(y_i - w^T x_i)^2 = (y - Xw)^T(y - Xw) = ||y - Xw||^2$
**Woodbury:** $U(VU + I) = (UV + I)U$
$(A + xx^T)^{-1} = A^{-1}\frac{(A^{-1}x)(A^{-1}x)^T}{1+x^T A^{-1}x}$

## 3 Bayesian Linear Regression

**Goal:** Use data $(\mathcal{X}\ \mathcal{Y})$ to fit a function $(\mathcal{X} \to \mathcal{Y})$. Use this function to predict unseen data.

**Ridge:** $y \approx w^T x$, $\hat{w} = \arg\min_w \sum_i^n (y_i - w^T x_i)^2 + \lambda ||w||_2^2$
**Analytical:** $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$

**Prior:** $P(w) = \mathcal{N}(0, \sigma_p^2 I)$
**Lhood:** $P(y|w,x) = \prod_i^n P(y_i|w,x_i) = \prod_i^n \mathcal{N}(y_i; w^T x_i, \sigma_n^2)$
In Bayes, we assume that y is not estimated as a single value, but drawn from a distribution.
$\hat{w} = \arg\max_w P(w|X,y) = \arg\max_w (\frac{1}{Z}) P(w|X) \prod_i P(y_i|w,x_i)$
$P(\hat{w}|X,y) = \mathcal{N}(\hat{w}; (X^T X + \frac{\sigma_n^2}{\sigma_p^2} I)^{-1} X^T y, (\sigma_n^{-2} X^T X + \sigma_p^{-2} I)^{-1})$
Assumes the weights to be Gaussian and the noise of the labels from $P(y|x,w)$ to be i.i.d. Gaussian.
If we would not assume a prior, we would get the MLE, which are simply mean and variance of the given data (No regularization in Ridge regression).

$\lambda$ is the ratio $\frac{\sigma_n^2}{\sigma_p^2}$ of noise in y to the variance of the weights

**Derivation:** Maximize the posterior by using Bayes rule, Ridge regression $\Rightarrow$ MAP weight estimate.
Use the definition of P(w) and $P(y|w,x)$ and remove constants to derive ridge regression.

**Predict:** $P(y^*|X,y,x^*) = \int p(y^*|x^*,w) p(w|x_{1:n}, y_{1:n}) dw$
$= \mathcal{N}(\hat{\mu_w}^T x^*, x^{*T} \hat{\Sigma_w} x^* + \sigma_n^2)$, in $O(nd^2)$
The predictive distribution is dependent on the distribution over the weights, as it averages over all possible weights. The predicted point $y^*$ adds variance, as the point can be seen as the prediction of $w^T x^* + \epsilon$.

**Epistemic:** Lack of data, the part $x^{*T} \hat{\Sigma} x^*$
**Aleatoric:** Irreducible noise from observation, the part $\sigma_n^2$

**Hyperparas:** $\lambda = \frac{\sigma_n^2}{\sigma_p^2}$ via CV, $\sigma_n^2 \approx \frac{1}{n} \sum_i (y_i - w^T x_i)^2$
The variance of the weights can then be derived from these two equations.

# 4 Gaussian Process

Generalization of Bayesian linear regression. Gaussian Processes are looking at it from the function space, e.g. models distributions over functions directly.
In Bayesian linear regression we modelled the distribution over the weights.

**Introduce non-linearity:** $f(x) = w^T\phi(x) \Rightarrow$ dim. explosion

**Kernel:** $x_i^T x_j \Rightarrow \phi(x)^T\phi(x) = k(x_i, x_j)$

**F-space:** $f = [f_{i:n}] = Xw$, $w \sim \mathcal{N}(0, \sigma_p^2 I)$, $f \sim \mathcal{N}(0, \sigma_p^2 XX^T)$
Instead of thinking about Gaussian priors on the weights w (weight-space), we think about gaussian priors on the function values $f = w^T x$ (function-space).
The linear model then only encodes correlations between the functions through the Kernel, we only need to model how new functions (linearly transformed points) are correlating to the functions we have.
The conversion from weight space to function space happens through linear transformation of the weights, this assumes X to be constant!

**Predict:** $y^* = f^* + \epsilon \sim \mathcal{N}(0, \sigma_p^2 \hat{X}\hat{X}^T + \sigma_n^2 I)$, $\hat{X}^T = [X^T, x^*]$
Data only enters through the inner products, which make kernels very useful. The it models how new data correlates with already given data.

**Intuition:** We place a prior on functions that correlate with our data. Functions that agree with our prior and have a high likelihood are more likely, due to bayes: $P(f|data) = P(f)P(data|f)$

**GP:** $f \sim GP(\mu(.), k(.,.))$, $\mu : X \to \mathbb{R}, k : X \times X \to \mathbb{R}$
**Definition:** A set of (infinitely) many random variables, using a finite subset indexed by a set X.
**Definition 2:** A set of variables are a GP, if and only if any marginalized set of random variables is a multivariate Gaussian.
We need to make sure that we are consistent in this infinite set, that's why we have a mean function and covariance function, both encode assumptions about the prior. By having each function a Gaussian, we can combine the means/variances of the single random variables.

$P(f|X_A, y_A) = GP(f; \mu(x) + k_{x,A}(K_{AA} + \sigma_n^2 I)^{-1}(y - \mu_A),$
$k(x, x') - k_{x,A}^T(K_{AA} + \sigma_n^2 I)^{-1} k_{x',A}))$
**Cost:** Size of $K_{AA}$ is $|X_A| \times |X_A|$, cost of inversion $O(|X_A|^3)$. $k_{x',X_A}$ is a $|X_A|$ vector, evaluating x' against all elements from $X_A$.
**Speed-up:** Kernels with distance notion can just use nearby points, treat the rest as independent.

**Sample:** $f \sim \mathcal{N}(0, K), K = LL^T, \epsilon \sim \mathcal{N}(0, I) \Rightarrow f = L\epsilon$
We don't sample f directly, we sample from a marginal distribution that is finite and Gaussian.

**Model select:** $\hat{\theta} = \arg\max_\theta \int p(y_{train}|X_{train}, f, \theta)p(f|\theta)df$
$= \arg\max \int \mathcal{N}(y; f(x), \sigma_n^2)\mathcal{N}(f; 0, K_f(\theta)) = \arg\max$
$\mathcal{N}(y; 0, K_f(\theta) + \sigma_n^2 I) = \arg\min -\frac{1}{2}y^T K_y(\theta)^{-1} y - \frac{1}{2}\log|K_y(\theta)|$
**Intuition:** We average over all possible functions, that means that the model must balance out fitting the data (likelihood) and obeying the prior (prior).

**CV:** In cross-validation, we find the parameters that fit best for one particular case, which can lead to overfitting more quickly. The parameters are often $[\sigma_p, \sigma_n, h]$.

**Performance:** Parallel, Local, Kernel approx, $O(n^3)$

**Fourier:** Shift-invariant kernel features BL, $O(nm^2 + m^3)$
The Gaussian kernel is a standart normal Gaussian in the Fourier Space.

**Bochner:** Shift-invariant kernel p.d. $\Leftrightarrow p(\omega)$ non-negative

## 4.1 Kernel

**Correlation:** $k(x_1, x_2) = Cov(f(x_1), f(x_2))$
$\forall x, x', k(x, x') = k(x', x)$, p.s.d (positive EV) $x^T K_{AA} x \geq 0$.
**Composition:** $+, \cdot, k \cdot const.$, $poly(f)$, $\exp(f)$ give again kernels
**Stationary:** if it holds $k(x, x') = k(x - x')$
**Isotropic:** $k(x, x') = k(||x - x'||_2)$, implies stationary
After fitting to a sample point, the posterior is not isotropic, since the variance is decreased in the region of the sample.

**Linear:** $k(x, x') = x^T x' =$ Bayesian linear regression
**Poly2:** $k(x, x') = \phi(x)^T\phi(x'), \phi(x) = [1, x, x^2]$
**Exp$^2$:** $k(x, x') = \exp(-||x - x'||_2^2/h^2)$, decay with distance
**Exp:** $k(x, x') = \exp(-||x - x'||_1/h)$, decay with distance
**Visual smoothness:** The bigger h is, the higher is the radius of influence. This means that the functions appear much smoother.

## 5 Approximate Inference

**Prior:** $p(\theta)$, **Likelihood:** $p(y|X) = \prod p(y_i|x_i, \theta)$
**Bayesian Posterior:** $p(\theta|X, y) = \frac{1}{Z} p(\theta) \prod_{i=1}^{n} p(y_i|x_i, \theta)$
**Prediction:** $p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, \theta) p(\theta|x_{1:n}, y_{1:n})$
We condition everything on the x's, but this notation is dropped now for ease of notation.

### 5.1 — Variational Inference

**Goal:** $p(\theta|y) = \frac{1}{Z} p(\theta, y) \approx q(\theta|\lambda)$

### 5.2 — Laplace Approximation

$\hat{f}(\theta) = \log p(\theta|y) \approx f(\hat{\theta}) + (\theta - \hat{\theta})^T f(\hat{\theta})' + \frac{1}{2}(\theta - \hat{\theta})^T f(\hat{\theta})''(\theta - \hat{\theta})$
It is the second order taylor expansion around the posterior mode. The first derivative cancels out, since it is zero at the mode.
$q(\theta) = \frac{1}{Z} \exp(\hat{f}(\theta)) \sim \mathcal{N}(\hat{\theta}, \Lambda^{-1}), \Lambda = -\nabla\nabla \log p(\hat{\theta}|y)$,
$\hat{\theta} = \arg\max p(\theta|y) \rightarrow$ **SGD:** $\theta_{t+1} = \theta_t - \eta_t \frac{1}{m} \sum^{m} \nabla_\theta l(\theta_t; x_i)$
**Convergence:** With proper learning rate, converges to (local) minimum. Convex functions only have a global minimum.
$p(y^*|x^*, X) = \int p(y^*|x^*, \theta) p(\theta|X, y) = \int p(y^*|f^*) \int p(f^*|\theta) q_\lambda(\theta)$
$\approx \int p(y^*|f^*) \mathcal{N}(f^*; \mu^T x^*, x^{*T} \Sigma x^*) \leftarrow$ **Predictive distr.**
**Problems:** It greedily seeks mode, then matches the curvature. This can lead to bad approximations, if the function is imbalanced.

### 5.3 — KL-Divergence

**KL:** $KL(q||p) = \int q(\theta) \log \frac{q(\theta)}{p(\theta)} d\theta = \mathbb{E}_{\theta\sim q}[\log \frac{q(\theta)}{p(\theta)}] \geq 0 \leftarrow$ rev.
**Variational Familiy:** We want a family of "simple" distributions to use for approximation, such as Gaussian or diagonal Gaussians.
**Intuition:** The more p and q agree, the lower the divergence is. If it is zero, they agree almost everywhere.
**Symmetry:** In general not symmetric, can't switch p and q
**Backwards KL:** $KL(q||p)$ is a mode seeker, since it lets us choose where we want to put q.
**Forward KL:** Tries that q is not small where some probability mass of p is, since the fraction would drive the divergence up. It is more inclusive, usually wider, but harder to compute.
$KL(p||q) = \frac{1}{2}(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1\text{-}\mu_0)^T \Sigma_1^{-1}(\mu_1\text{-}\mu_0) - d + ln(\frac{|\Sigma_1|}{|\Sigma_0|}))$
$p \sim \mathcal{N}(\mu_0, I), q \sim \mathcal{N}(\mu_q, I), \rightarrow \frac{||\mu_1 - \mu_0||^2}{2}$ $|p \sim \mathcal{N}(\mu_{1:d}, [\sigma^2_{1:d}])$,
$q \sim \mathcal{N}(0, \sigma_p^2 I) \rightarrow \frac{1}{2} \sum_{i=1}^{d} (\frac{\sigma_i^2}{\sigma_p^2} + \frac{\mu_i^2}{\sigma_p^2} - 1 - ln(\frac{\sigma_p^2}{\sigma_i^2})$ **Gaussian cases:** These special cases are for Standart Normal Gaussians and diagonal Gaussian cases.
**Entropy:** $H(q) = -\int q(\theta) \log q(\theta) d\theta = \mathbb{E}_{\theta\sim q}[-\log q(\theta)]$
**Prod:** $H(q) = \sum_{i=1}^{d} H(q_i)$, $H(\mathcal{N}(\mu, \Sigma)) = \frac{1}{2} ln(|2\pi e \Sigma|)$
**Diagonal Gaussian:** $H(\mathcal{N}) = \frac{1}{2} ln(|2\pi e \Sigma|) = \frac{1}{2} ln(2\pi e) + \sum_i^d ln(\sigma_i)$
**KL:** $q^* \in \arg\min_q KL(q||p) = \arg\min_{q_\lambda} \mathbb{E}_{\theta\sim q}[log \frac{q(\theta)}{\frac{1}{Z} p(\theta, y)}]$
$= \arg\max_q \mathbb{E}_{\theta\sim q}[log(p(\theta, y)] + H(q)$
$= \arg\max_q \mathbb{E}_{\theta\sim q}[log(p(y|\theta)] - KL(q||p(.)) =$ ELBO, **BLR** $\downarrow$
Derivation: $q^* \in \arg\min_q KL(q||p) = \arg\min_q \mathbb{E}_{\theta\sim q}[log \frac{q(\theta)}{\frac{1}{Z} p(\theta, y)}]$
$= \arg\min_q \mathbb{E}_{\theta\sim q}[log(q(\theta)) + log(Z) - log(p(\theta, y)]$
$= \arg\min_q \mathbb{E}_{\theta\sim q}[-log(p(\theta, y)] + \mathbb{E}[log(Z)] - \mathbb{E}[-log(q(\theta))]$
$= \arg\max_q \mathbb{E}_{\theta\sim q}[log(p(\theta, y)] + H(q)$
$= \arg\max_q \mathbb{E}_{\theta\sim q}[log(p(y|\theta) + log(p(\theta)) - log(q(\theta))]$
$= \arg\max_q \mathbb{E}_{\theta\sim q}[log(p(y|\theta)] - KL(q||p(.))$
**Note:** The KL at the bottom is directed at the prior, not the posterior anymore.
**Interpretation 1:** We want function q that maximizes the joint probabilty, but has high uncertainty where we don't have data.
**Interpretation 2:** We want to maximize the likelihood of the data, but stay as close to the prior as possible. It acts like a regulation term, otherwise we would get MLE.
In MLE we would optimize over d dimensions, if we have a diagonal Gaussian, the ELBO would optimise the means and variances, e.g. 2d dimensions.
$\mathbb{E}_{\theta\sim q_\lambda}[-\sum \log(1 + \exp(-y_i \theta^T x_i))] - \frac{1}{2} \sum^{d} (\mu_i^2 + \sigma_i^2 - 1 - ln(\sigma_i^2))$
This is the Elbo for Bayesian regression.
Problem: For the expectation term, it is hard to take the derivative, since we can't just push the expectation in. The distribution q itself depends on the parameters theta.

### 5.4 — Repatameterization

$q(\theta|\lambda) = \phi(\epsilon)|\nabla_\epsilon g(\epsilon; \lambda)|^{-1}$, g invertible
$\Rightarrow \nabla_\lambda \mathbb{E}_{\theta\sim q_\lambda}[f(\theta)] = \mathbb{E}_{\epsilon\sim\phi}[\nabla_\lambda f(g(\epsilon; \lambda))]$, unbiased SGD est.
**diag-Gauss:** $\theta = C\epsilon + \mu, \Sigma = CC^T, \phi(\epsilon) = \mathcal{N}(\epsilon; 0, I)$
**Diagonal Gaussian:** In this case, C is just the square root of the variances.
$\theta$ is now obtain by sampling from the standard normal distribution, adding mu and multiplying by C. The standard rules of affine transformations of gaussians hold.
**Performance:** For diagonal q, only twice as expensive as MAP inference.
**ELBO:** $\nabla_\lambda \mathbb{E}_{\theta\sim q(.|\lambda)}[log(p(y|\theta)] - \nabla_\lambda KL(q_\lambda||p(.))$
$= \nabla_{C,\mu} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[log(p(y|C\epsilon + \mu, X)] - \nabla_{C,\mu} KL(q_{C,\mu}||p(.))$
$\approx n \cdot \frac{1}{m} \sum_j^m \nabla_{C,\mu} log(p(y_{ij}|C\epsilon^{(j)} + \mu, x_{ij}) - \nabla_{C,\mu} KL$, unbiased
$\nabla_\lambda \mathbb{E}_{\theta\sim q(.|\lambda)}[log(p(y|\theta)] = \nabla_{C,\mu} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[log(p(y|C\epsilon + \mu, X)]$
$= \nabla_{C,\mu} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[n \frac{1}{n} \sum log(p(y_i|C\epsilon + \mu, x_i)]$
$= n \cdot \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} \mathbb{E}_{i\sim Uniform}[\nabla_{C,\mu} log(p(y_i|C\epsilon + \mu, x_i)]$

## 6 Bayesian Logistic Regression

**Prior:** $p(\theta) = \mathcal{N}(0, \sigma_p^2 \cdot I)$, **Lhood:** $p(y|X, w) = \prod \sigma(y_i \cdot w^T x_i)$
$p(y|x, w) = Ber(y; \sigma(w^T x))$
When the argument is positive, then the sigmoid function is evaluating to something bigger than 0.5, if it is negative, it is evaluating to something lower.
Only if the signs of the label and the prediction match the evaluation is positive.
**Laplace:** $\hat{w} = \arg\max_w p(w|y) = \text{am}_w \log p(w) + \log p(y|w)$
$= \arg\min_w \frac{1}{2\sigma_p^2} ||w||_2^2 + \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ (use SGD)
**Logistic Regression:** Seeking the mode is giving us the loss of the standart logistic Regression. Use log and remove constants to get the Logistic Loss.
**SGD:** $w = w(1 - 2\lambda\eta_t) + \eta_t yx \frac{1}{1 + exp(yw^T x)}$ $(\leftarrow \hat{P}(Y = -y|w, x))$
$\Lambda = -\nabla\nabla \log p(\hat{w}|X, y) = X^T diag([\sigma(\hat{w}^T x_i)(1 - \sigma(\hat{w}^T x_i))]_i) X$
**Predict:** $p(y^*|x^*, X, y) = \int \sigma(y^* f) \mathcal{N}(f; \hat{w}^T x^*, x^{*T} \Lambda^{-1} x^*)$
This integral still has no closed form, but can be efficiently approximated with Gaussian quadrature.

## 7 Monte Carlo Sampling

$p(y^*|x^*, x_{1:n}, y_{1:n}) = \int p(y^*|x^*, \theta)p(\theta|x_{1:n}, y_{1:n}) =$
$\mathbb{E}_{\theta\sim p(.|X,y)}[p(y^*|x^*, \theta)] \approx \frac{1}{m}\Sigma_i^m p(y^*|x^*, \theta^{(i)}), \theta^{(i)} \sim p(\theta|X, y)$

**LargeNums:** $\mathbb{E}_P[f(X)] \approx \frac{1}{N}\sum_i^N f(x_i)$ **Hoeffding:** f in $[0, C]$,
$P(|\mathbb{E}_P[f(X)] - \frac{1}{N}\sum_i^N f(x_i)| > \epsilon) \leq 2\exp(-2N\epsilon^2/C^2)$

**Intuition:** Based on the law of large numbers, we can derive the required number of samples N to be in a certain probability bound.

**Approx:** $\frac{1}{Z}Q(x) = P(x)$, MC seq. $X_{1:N}$ with stationary $P(x)$
Prior $P(X_1)$, transitions $P(X_{t+1}|X_t)$ i.i.d of $t$

The Markov chain is a sequence of Random Variables, mostly the random variables are states in $\{1, .., M\}$.
The MC is stationary if the transition probabilities are independent of t, e.g. they don't change if t changes.

**Ergodic:** Reach all states from everywhere in exactly t steps

**Note:** It must be exactly t steps to reach every state from everywhere. **Example:** MC (1)-(2), whereas the state (2) can be reached from (1) after t=uneven steps, whereas it can be reached from (2) only after t=even steps.

**Statnary:** $\lim_\infty P(X_N = x) = \pi(x)$, implied by↑, $P(X_1)$ egal

**Sample:** $x_1 \sim P(X_1), x_n \sim P(X_N|X_{N-1} = x_{N-1})$

**Goal:** $\pi(x) = \frac{1}{Z}Q(x)$, need to specify $P(x|x')$

**Balance:** $\frac{1}{Z}Q(x)P(x'|x) = \frac{1}{Z}Q(x')P(x|x') \Rightarrow \pi(x) = \frac{1}{Z}Q(x)$

Given unnormalized distribution Q(x), we want to design a Markov chain with stationary distribution $\pi$. We can ensure that the stationary distribution is correct if it satisfies the detailed balance equation.
It suffices to show that $P(X_t = x) = \frac{1}{Z}Q(x) \Rightarrow P(X_{t+1} = x) = \frac{1}{Z}Q(x)$.
Proof: $P(X_{t+1} = x) = \sum_{x'} P(X_{t+1} = x, X_t = x')$
$= \sum_{x'} P(X_{t+1} = x|X_t = x')P(X_t = x')$
$= \sum_{x'} P(X_{t+1} = x|X_t = x')\frac{1}{Z}Q(x')$
$= \sum_{x'} P(X_{t+1} = x'|X_t = x)\frac{1}{Z}Q(x)$
$= \frac{1}{Z}Q(x)$

### 7.1 — Metropolis Hastings

**1) Proposal:** Given $X_t = x$, sample $x' \sim R(X'|X = x)$

**2) Accept:** w/ probability $\alpha = \min(1, \frac{Q(x')R(x|x')}{Q(x)R(x'|x)})$

**Theorem:** Stationary is $Z^{-1}Q(x)$

The performance depends heavily on what we choose as proposal distribution R.

**Gibbs:** $t..\infty : i \sim Uniform$, update $x_i = P(X_i|x_{-i})$
$P(X_i = x_i|x_{-i}) = \frac{1}{Z}Q(X_i = x_i, x_{-i}) = \frac{Q(x_{1:n})}{\sum_{x_i} Q(X_i = x_i, x_{-1})}$

↑ sat. balance equation, practical variant not. $P(X_i|x_{-i})$ eff.

We are only updating one of the variables by conditioning on all the other variables.
This can be done efficiently, as the normalizer has to be computed over all $x_i$, which are limited in f.e. a categorical distribution.

**LLN (Ergodic):** $lim_{N\to\infty}\frac{1}{N}\sum_i^N f(x_i) = \sum_{x\in D}\pi(x)f(x)$

By design of the Markov Chain, the samples depend on each other, so the law of large numbers and the complexity bounds do not apply.
Special case is when the Markov chain is Ergodic and the state space D is finite with stationary $\pi$. This is also called the law of large numbers for Markov Chains.
This talks about if it converges to the correct thing, but unfortunately not how fast it converges.

**Expectations:** $\mathbb{E}[f(X)] \approx \frac{1}{T-t_0}\sum_{\tau=t_0+1}^T f(X^{(\tau)})$, $t_0$ burn-in

The initial state is very crucial when it comes to time-to-convergence. Often the mode is determined heuristically, then started from there.
Establishing convergence rates is very difficult in general.

**General RV:** $Q(x) = \frac{1}{Z}\exp(-f(x)))$, $f(x) > 0$
$p(\theta|X, y) = \frac{1}{Z}p(\theta)p(y|X, \theta) = \frac{1}{Z}\exp(\text{-}[\text{-}\log p(\theta) - \log p(y|X, \theta)])$

**Bay.Logistic Reg.:** $f(\theta) = \lambda||\theta||^2 + \sum log(1+exp(-y_i\theta^T x_i))$

**Hastings:** $\alpha = \min(1, \frac{R(x|x')}{R(x'|x)}\exp(f(x) - f(x')))$ | ↓ b.c. sym.
$R(x'|x) = \mathcal{N}(x'; x; \tau I) \Rightarrow \alpha = \min\{1, \exp(f(x) - f(x'))\}$

The R-fraction is 1, since Gaussians are symmetric.
$f(x') < f(x)$: we accept with probability 1 if the cost function is smaller at x'.
$f(x') > f(x)$: cost of new position is higher, still there is a possibility to go into this direction if not too high.

**Note:** The Algo could get stuck.

$R(.|.) = \mathcal{N}(x'; x\text{-}\tau\nabla f(x); 2\tau I)$, eval. $f(x)$ both steps

MALA incorporates Langevin dynamics. We sample R in direction of the Mode of the energy function, this can help with the direction.
Gradient computation can be expensive, there is a version where stochastic gradients are used.

# 8 Bayesian Deep Learning

We want to treat the weights of a neural network as distributions, for which we can define uncertainty as well.

**Neural Net:** $f(x; w) = \varphi(W_1\varphi(W_2(..\varphi(W_l x)))$

**Basic Unit:** $v_i^{(l)} = \varphi(w_i^{(l)T} v^{(l-1)})$

We want each unit to be differentiable, such that efficient computation graphs can be automatically created.

**tanh(z)** $= \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$ **reLu(z)** $= \max(z, 0)$

**Bayesian NN:** $p(y|x, \theta) = \mathcal{N}(y; f_1(x, \theta), f_2(x, \theta))$

**MAP:** $\hat{\theta} = \arg\min_\theta \lambda||\theta||^2 + \sum_{i=1}^n \log \sigma^2(x_i, \theta) + \frac{(y_i-\mu(x_i,\theta))^2}{\sigma^2(x_i,\theta)}$

Can be derived by plugging in the Bayesian NN likelihood and a standard Gaussian prior.

**Intuition:** The model can be far of the mean and can still balance it out with increased variance. High variance is costly, though.

The model has to balance out how it explains the data with mean and variance.

**Predict:** $p(y^*|x^*, x_{1:n}, y_{1:n}) = \mathbb{E}_{\theta\sim p(.|X,y)}[p(y^*|x^*, \theta)]$
$\approx \frac{1}{m}\sum_j^m \mathcal{N}(y^*; \mu(x^*, \theta^{(j)}), \sigma^2(x^*, \theta^{(j)}))$

We just parameterize all the means and variances with our NN. If we use the reparameterization trick, we follow along an use $C\epsilon + \mu$ as new $\theta$.

For Gaussian likelihood, the prediction becomes a mixture of Gaussians.

**Mean:** $\mathbb{E}[y^*|X, y, x^*] \approx \bar{\mu}(x^*) = \frac{1}{m}\sum_j^m \mu(x^*, \theta^{(j)})$

$Var[y^*|X, y, x^*] = Var[\mathbb{E}[y^*|X, y, x^*]] + \mathbb{E}[Var[y^*|X, y, x^*]]$
$= \frac{1}{m}\sum_j^m \sigma^2(x^*, \theta^{(j)}) + \frac{1}{m}\sum_j^m(\mu(x^*, \theta^{(j)}) - \bar{\mu}(x^*))^2$

The variance of the prediction is dependent on the variance of the weights.

We can split up the uncertainty into Aleatoric (sigma-term) and epistemic (mu-term)

## 8.1 MCM

**Predict:** $p(y^*|X, y, x^*) \approx \frac{1}{T}\sum_j^T p(y^*|x^*, \theta^{(j)})$, $\theta^{(i)}$ as NN W

**Approx:** $q(\theta|\mu_{1:d}, \sigma^2_{1:d}), \mu = \frac{1}{T}\sum_j^T \theta^{(j)}, \sigma^2 = \frac{1}{T}\sum_j^T(\theta^{(j)} - \mu)^2$

We can keep track of the weight means and variances by keeping a running average while monte-carlo sampling. The parameters can then easily be approximated with a Gaussian.

**softmax(f):** $p(y|x, \theta) = p_y = \frac{\exp(f_y)}{\sum_j^c \exp(f_j)}$, $softmax(f + \epsilon)$

Gives probability distribution over the c classes.

We can inject noise $softmax(f + \epsilon)$, saying that we are unsure about some parts of the predictive classes. This is analog to adding Epistemic noise, just for classification.

## 9 Active Learning

We can model the uncertainty with previously discussed methods, and we can categorise them into aleatoric and epistemic uncertainty. The topic is now how to make decisions based on this uncertainty.

**Active Learning:** Algorithm learns where samples should be taken. Makes sense when attaining samples is expensive. Often we have a collection of unlabeled datapoints, where do we want to aquire a label?

**Note:** GP Posterior covariance does not depend on y

**Entropy:** $H(X) = \mathbb{E}_{x\sim p(x)}[-\log p(x)], H(X|Y)$
$= \mathbb{E}_{y\sim p(y)}[H(X|Y=y)], H(X) + H(Y|X) = H(X,Y)$

**Gauss:** $X \sim \mathcal{N}(\mu, \Sigma), H(x) = \frac{1}{2}\log(2\pi e)^d|\Sigma|$

**Mutual Info:** $I(X;Y) = H(X) - H(X|Y), I(X;Y|Z) = H(X|Z) - H(X|Y,Z)$, symmetric

**Gauss:** $X \sim \mathcal{N}(\mu, \Sigma), Y = X + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_n^2 \cdot I)$
$I(X;Y) = H(Y) - H(Y|X) = H(Y) - H(\epsilon)$
$= \frac{1}{2}\log(2\pi e)^d|\Sigma + \sigma_n^2 I| - \frac{1}{2}\log(2\pi e)^d|\sigma_n^2 I| = \frac{1}{2}\log|I + \sigma_n^{-2}\Sigma|$
If we have a dependent variable, we see by this example that we don't gain a lot of information.

**Target:** $\max F(S) = I(f; y_S) = H(f) - H(f|y_S)$, w/ $S \subseteq D$
We want to find a subset of points which maximises the information we gain on the function f. If we deal with Gaussians, we can use the formula above.
This is only one way to define utility.

**Greedy (Homoscedasdic):** $x_{t+1} = \arg\max_{x\in D} F(S_t \cup \{x\})$
$= \arg\max_{x\in D} I(f; y_{S_t+x}) - I(f; y_{S_t}) = \arg\max_{x\in D} H(f) - H(f|y_{S_t+x}) - H(f) + H(f|y_{S_t}) =^{const.\sigma_n^2} \arg\max_{x\in D} \sigma_t^2(x)$
F(S) is NP-Hard to optimize in this discrete case, we often use heuristics to solve this problem.
x could be something continuous or non-deterministic (e.g. throw a sensor somewhere), then this optimization problem changes.
With the greedy approach, and the assumption that we have constant (homoscedastic) noise, we just pick the points where we are the most uncertain. In other words, we can maximise greedily the mutual information.
This way of collecting the set of points is near-optimal.

**Derivation:** $x_{t+1} = \arg\max_{x\in D} F(S_t \cup \{x\})$
$= \arg\max_{x\in D} F(S_t \cup \{x\}) - F(S_t)$
$= \arg\max_{x\in D} I(f; y_{S_t+x}) - I(f; y_{S_t})$
$= \arg\max_{x\in D} H(f) - H(f|y_{S_t+x}) - H(f) + H(f|y_{S_t})$
$= \arg\max_{x\in D} H(f|y_{S_t}) - H(f|y_{S_t+x})$
$= \arg\max_{x\in D} I(f; y_x|y_{S_t})$
$=^{Gauss} \arg\max_{x\in D} \frac{1}{2}\log(1 + \frac{\sigma_t^2(x)}{\sigma_n^2})$
$=^{const.\sigma_n^2} \arg\max_{x\in D} \sigma_t^2(x)$

**Greedy (Heteroscedastic):** $x_{t+1} = \arg\max_{x\in D} \frac{\sigma_t^2(x)}{\sigma_n^2(x)}$

**Interpretation:** Epistemic uncertainty divided by the Aleatoric uncertainty.
Just because we are very uncertain at a point, does not mean that we decrease the uncertainty by sampling at that point. It could be that the uncertainty comes from irreducible noise anyways.

**Performance:** Cons-factor approx. (near opt.), submodular
**Proof:** Via submodularity...

**Classification:** $x_{t+1} = \arg\max_{x\in D} H(Y|x, x_{1:n}, y_{1:n})$
$= \arg\max_{x\in D} -\sum_y \log p(y|x, x_{1:n}, y_{1:n})$
**Intuition, heteroscedastic case:** The most uncertain label is not necessarily the most informative. If we have binary classification, the point closest to the division boundary are chosen. But, those are also the points with the largest variance (aleatoric noise), since they are farthest away from the "source" cluster. So if the aleatoric noise is high, the decision boundary might choose weird points and don't get a proper fit.

## 10 Bayesian Optimization

For example synthesizing molecules or choosing hyperparameters of neural networks, we want some kind of process to derive the next parameters to try out. The evaluation of the parameters is very expensive.

**Tradeoff:** Exploitation vs. exploration. We don't want to make experience with clearly suboptimal results, so we need to balance these two factors.

**Key Idea:** Learn an approximate model of a blackbox function, which then can be used to plan what samples to take next. The samples we get are noisy: $y_t = f(x_t) + \epsilon_t$.

**Goal:** How to sequentially pick x's to find $\max_x f(x)$ with minimal samples. Find an aquisition function that does this.

**Given:** Noisy black-box f, choose $x_1, .., x_T| \downarrow$ if $\frac{R_T}{T} \to 0$

**Regret:** $R_T = \sum_t^T (\max_x f(x) - f(x_t)) \Rightarrow \max f(x_t) \to f(x^*)$
Given a set of possible inputs D, and noisy blackbox function f, choose $x_s$ and observe $y_t = f(x_t) + \epsilon_t$.
The cumulative regret tells us the error if we knew the maximum in advance. It has to go to zero over time, to reach the actual maximum.
Algorithms have regret bounds, which jufge the effectiveness of an algorithm.

**UpConfidence:** f at least highest lower bound (f enclosed)
We assume here that the GP we are looking at has the function enclosed. This means, that the maximum of the function must be at least as high as the maximum lower bound. We can focus on the regions where this is true

**GP-UCB:** $x_t = \arg\max_{x\in D} \mu_{t-1}(x) + \beta_t\sigma_{t-1}(x)$
The goal is to maximise the upper confidence bound. **Assumption:** The underlying function really is something like the prior, f.e. a Gaussian. Also, we need this to hold for all the points, so we need to do a union bound over the points. This can be solved by using a large constant $\beta$.
If Beta is huge, we just do uncertainty sampling. With a proper $\beta$, we naturally incorporate exploitation around the means.
The optimization function is generally non-convex, can be optimized with SGD.

**Reget:** $\frac{1}{T}\sum^T[f(x^*) - f(x_t)] = O(\sqrt{\frac{\gamma_T}{T}}), \gamma_T = \max_{\leq T} I(f; y_s)$
**Lin:** $\gamma_T = O(d\log T)$, **Exp**= $O((logT)^{d+1})$, **Matrn**= $O(TlogT)$
**Intuition:** Depending on the kernel, the regret has different convergence/ regret guarantees.

**Thompson:** $\hat{f} \sim P(f|x_{1:n}, y_{1:n}), x_{t+1} \in \arg\max \hat{f}(x)$

## 11 Markov Decision Process

**Probabilistic planning:** How should we control an agent to accomplish some goal

**MDP:** State X, Action A, Transition $P(x'|x,a)$, Reward $r(x,a)$
We have an observable state and a finite number of actions. The actions are dictated by the task and the environment, while the reward function gives us room for design choices. Each transition probability is independent, giving it the Markovian assumption. We only care about where an actor currently is, not were he was.

**Policy:** $\pi : X \to A$, $P(X_{t+1} = x'|X_t = x) = P(x'|x, \pi(x))$, **or**
Deterministic Policy
$\pi : X \to P(A)$, $P(X_{t+1} = x'|X_t = x) = \sum_a \pi(a|x)p(x'|x,a)$
Probabilistic Policy

**Expectation:** $J(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t))]$, for MC: $X_0, ..$
**Note:** This expectation depends on the inital state of the MC. We can use gamma to regulate how much we care about future rewards.

$\mathbf{V^\pi(x)} = J(\pi|X_0 = x) = r(x, \pi(x)) + \gamma \sum_{x'} p(x'|x, \pi(x))V^\pi(x')$
The value function at a specific state x incorporates the reward at this state when acting after policy $\pi$, and expected reward it could get.
It is recursively defined, using the expected rewards of other reachable states through their value function.

**Exact solution:** $V^\pi = (I - \gamma T^\pi)^{-1} r^\pi$, w/ $\gamma < 1$, P/r known
$V^\pi = (V^\pi(1), \dots, V^\pi(n))$
$r^\pi = (r^\pi(1, \pi(1)), \dots, r^\pi(n, \pi(n)))$

$$T^\pi = \begin{pmatrix} p(1|1,\pi(1)) & \cdots & p(n|1,\pi(1)) \\ \vdots & \ddots & \vdots \\ p(1|n,\pi(n)) & \cdots & p(n|n,\pi(n)) \end{pmatrix}$$

**Fixed Point:** for $t = 1 : T$ do $V^\pi = r^\pi + \gamma T^\pi V^\pi_{t-1}$
$B^\pi V := r^\pi + \gamma T^\pi V \Rightarrow B^\pi V^\pi = V^\pi$
$||B^\pi V - B^\pi V'|| \leq \gamma ||V - V'||, ||V^\pi_t - V^\pi|| \leq \gamma^t ||V^\pi_0 - V^\pi||$
With the contraction B, an upper bound can be derived in terms of $\gamma$ and the value function. Considering the true Value function $V^\pi$, we see that each iteration is reducing the error exponentially.

**Greedy Policy:** $\pi^* = \arg\max J(\pi)$, # policies $O(|X|^{|A|})$
**w.r.t. V:** $\pi_G(x) = \arg\max_a r(x,a) + \gamma \sum_{x'} p(x'|x,a)V(x')$
**Bellman:** $V^*(x) = \max_a [r(x,a) + \gamma \sum_{x'} p(x'|x,a)V^*(x')]$
$Q^*(x,a) = \mathbb{E}_{x'}[r(x,a) + \gamma \max_{x'} Q^*(x',a')]$
Every value function induces a policy, every policy induces a value function.
**Bellman Theorem:** The optimal policy is greedy w.r.t. to its induced value function.
**Policy Iteration:** 1) $V^\pi(x)$ 2) $\pi_G$ w.r.t. $V^\pi$ 3) $\pi = \pi_G$
**Convergence:** $V^{\pi_{t+1}} \geq V^{\pi_t}$, $\pi^*$ in $O(n^2 m/(1-\gamma))$
We initialize with a random policy, although, the convergence depends heavily on the initialization

**Value iteration:** $V_t(x) = \max_a Q_t(x,a)$ until $||V_t - V_{t-1}|| \leq \epsilon$
$\forall a, x : Q_t(x,a) = r(x,a) + \gamma \sum_{x'} P(x'|x,a)V_{t-1}(x')$
Value iteration is a dynamic program, which stores the intermediate V's to use in the next round. In each round, the Q-Values of all state-action pairs must be calculated, which can be computationally intensive.
**Note:** It is guaranteed to converge!
**reward:** If for an action exist multiple outcomes, then the expected reward is counted. The reward function slips into the sum.
**PI:** exact, expensive **vs. VI:** cheap, more iters, $\epsilon-$optimal
Both need polynomial iterations. Policy iteration need much less iterations and solves the problem exact, but it is expensive to compute one iteration.
Value iteration is fast per iteration, but needs more iterations. Also, the solution is only $\epsilon-$optimal, compared to PI.

### 11.1 — POMDP- Belief state MDP

If the state space is very large or continuous, we can't solve the MDP exactly anymore. This model generalises MDP's, using a so-called belief state, e.g. the actor don't know anymore at what state we are, but has a belief of it.
Typically, they are untractable.

**Observe:** $P(Y_{t+1} = y|b_t, a_t) = \sum_{x,x'} b_t(x)P(x'|x,a_t)P(y|x')$
$b_{t+1}(x') = \frac{1}{Z} \sum_x b_t(x)P(X_{t+1} = x'|X_t = x, a_t)P(y_{t+1}|x')$
**Reward:** $r(b_t, a_t) = \sum_x b_t(x)r(x,a_t)$
The transition model now contains stochastic observations and state updates through bayesian filtering.
It can calculate the optimal action using dynamic programming. Often, the horizon is limited to T, the reachable belief states still grow exponential in T.

# 12 Reinforcement Learning

RL handles the case where we don't know the underlying MDP, e.g. we don't know the rewards and probabilities of the actions.
**Goal:** Learn the mapping from actions (or a sequence of actions) to rewards and thus the MDP.
**Credit assignment problem:** An agent must assign rewards to actions in a certain state. Which actions got me the largest reward?

**Goal:** max. discounted rewards $\sum^{\infty} \gamma^t r(X_t, A_t)$ **Data:** $\tau^{(i)} = (x_0^{(i)}, a_0^{(i)}, r_0^{(i)}, x_1^{(i)}, a_1^{(i)}, r_1^{(i)}, ..), D = \{x_j^{(i)}, a_j^{(i)}, r_j^{(i)}, x_{j+1}^{(i)}\}$
Compared to supervised learning, the data is not i.i.d. and the dataset is not fixed, e.g. we generate it by interacting with our environment.
**Episodic:** Creates multiple trajectories, resetting the agent.
**Dilemma:** Explore or Exploit?
Explore and gather more data to avoid missing out on a potentially large reward, or exploit and optimize the current policy ?

**On-policy:** Agent takes actions following a policy $\pi$
**Off-policy:** No specific policy is followed to learn
In on-policy learning, the agent follows the current policy when creating further trajectories.
In off-policy learning, the agent uses trajectories to learn that are not following a specific policy, f.e. random actions, greedy max actions.
**Model-based:** Estimate MDP ($P(x'|x,a)$, $r(x,a)$), optimize policy based on MDP
Collect data and learn the transition probabilities and rewards. From there, use the standard methods to estimate the policy.
**Model-free:** Estimate V directly (Policy gradient, AC)
Learn the value function directly, since the optimal policy is greedy with respect to the value function, it is enough to know the value function to determine the optimal policy.

## 12.1 Model-based RL

**Estimates:** Transition probabilities and reward function
$\hat{p}(X_{t+1}|X_t, A) = \frac{Count(X_{t+1}, X_t, A)}{Count(X_t, A)}, \hat{r}(x,a) = \frac{1}{N_{x,a}} \sum R(x,a)$
The problem reduces to estimating the transition probabilities and reward function (MLE in this example). As dataset, we can use the single transitions from the trajectories, since even if the trajectory itself is not independent, the transitions are.
**Supervised learning:** In this setting, we can use all the techniques from supervised learning to estimate the two functions. The input are state-action pairs, the labels are next state and reward.
**Exploration vs. Exploitation:** We can always pick a random action, then we eventually learn the correct probabilities and rewards. It may take very long though.
If we always pick the best action according to the current knowledge, we might see quick results, but get stuck with suboptimal actions.
**Note:** The important distinction is that how often we observe certain state-space pairs depends on what we do. So the distribution of the inputs depends on out policy.

**$\epsilon_t$-greedy:** Pick random or best action, $\sum \epsilon = \infty, \sum \epsilon^2 < \infty$
If $\epsilon$ satisfies the conditions, it converges, but can take some time to eliminate clearly suboptimal actions

**$R_{max}$:**
How can we model optimism for our estimated MDP's?
If you don't know r and p, we introduce a fairy tale state $P(x^*|x,y) = 1$, $r(x^*,a) = R_{max}$. We set $r(x,a) = R_{max}$ and $P(x^*|x,a)$ for all states x and actions a.
The algorithm keeps track of how many times we visited certain state-action pairs.
We then use the initial policy $\pi$ and update $r(x,a)$ and $p(x'|x,a)$ for every transition that occured. If we observed enough transitions, we recompute policy $\pi$ and do it all over again.
**How many samples?** Using Hoeffinger bound, if samples are i.i.d. with mean $\mu$ and bounded in $[0, C]$, we have error estimate $P(|\mu - \frac{1}{n}\sum_i^n Z_i| > \epsilon) \leq 2\exp(-2n\epsilon^2/C^2)$
F.e. $P(|\hat{r}(x,a) - r(x,a)| > \epsilon) \leq 2\exp(-2n_{x,a}\epsilon^2/R_{max}^2) = \delta$, then $n_{x,a} \in O(\frac{R_{max}^2}{\epsilon^2} \log \frac{1}{\delta})$

**Hoefding:** $P(|\mu - \frac{1}{n}\sum_i^n Z_i| > \epsilon) \leq 2\exp(-2n\epsilon^2/C^2) = \delta$, $Z \in [0, C]$

$P(|\hat{r}(x,a) - r(x,a)| \leq \epsilon) \geq 1 - \delta$, then $n_{x,a} \in O(\frac{R_{max}^2}{\epsilon^2} \log \frac{1}{\delta})$
The bound tells us how many samples we need to be under a certain error threshold. $R_{max}$ bounds the value of Z, as required.

**Performance:** w/ prob. $1-\delta$, $\epsilon-$optimal policy in # steps polynomial in $|X|, |A|, T, 1/\epsilon, \log(1/\delta)$ and $R_{max}$.
We do have a guarantee that after a certain amount of steps the algo makes progression, e.g. visits an unknown state-action pair or obtains near optimal rewards.
**Demo:** The testing value is showing the results we would get by following the estimated policy.
The Exploration value shows the results when we would act after the current policy (greedy, $\epsilon-$greedy,...)
**Greedy Policy:** Oscilates between two states, because it detects that going in one direction eventually leads to bumping into a wall, and thus want's to go back immediatly.
**Random:** Quickly saw all states and rewards, like a random walk. The Exploitation value of this policy is really bad, since we always place random actions. Actions and state that are far away may take long to be explored.
**$\epsilon-$greedy:** Takes a little longer until all states are visited, but the exploitation value is much better, it concentrates much more on good states.
**$R_{max}$:** Needs some iterations until it has enough samples, then decreases the estimated values accordingly.

**Memory:** $\hat{p}(x'|x,a)$ $(O(|A||X|^2))$ and $\hat{r}(x,a)$ $(O(|X|^2))$
**Computation:** Solve MDP every epoch, $O(|X||A|)$ for $R_{max}$
After we get the estimations in, we compute the MDP (which is the end goal).
The $R_{max}$ algorithm is guaranteed to make progress, but it can be that only one state-action pair is progressed per iteration.

## 12.2 Model-free RL

Want to approximate V directly from a policy

**TD:** $\hat{V}^\pi(x) = (1-\alpha_t)\hat{V}^\pi(x) + \alpha_t(r + \gamma\hat{V}^\pi(x'))$, w/ $(x,a,r,x')$
**Conditions:** $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$, $\infty$ visits, follow $\pi$
Since we don't know the optimal policy yet, we can use the bootstrap estimate from the current policy. It is crucial that we follow the policy.

**Q*(x,a)**= $r(x,a) + \gamma \sum_{x'} p(x'|x,a)V^*(x')$
**V*(x)**= $\max_a Q^*(x,a)$, even off-policy

**Q-lern:** $\hat{Q}^*(x,a) = (1-\alpha_t)\hat{Q}^*(x,a) + \alpha_t(r + \gamma \max_{a'} \hat{Q}^*(x',a'))$
We know that the optimal $V^*$ depends on the Q values, in this approach we directly estimate the Q's and then maximise over them.
This leaves us with an off-policy method, since we are considering all state-action pairs. It matters again that we visit the states often enough.
The immediate reward and the transition model only depend on the action a, whereas the long term value depends on the policy $\pi$.
Update $Q^*$ after each observed transition.
**Note:** $V^\pi(x) = Q^\pi(x, \pi(x))$

**Optimistic:** $\hat{Q}^*(x,a) = \frac{R_{max}}{1-\gamma} \prod_t^{T_i nit} (1-\alpha_t)^{-1}$, perf. as $R_{max}$
If we initialize the weights like this, we get an optimistic Q-learning with convergence guarantees and the same runtime guarantees as in $R_{max}$

**Memory:** Store all $Q^*(x,a)$ $(O(|A||X|))$
**Cost:** (single update) compute $\max_{a'} Q^*(x',a')$, $O(|A|)$
We need to compute the max of v, so it is linear in the number of actions.
**TD-SGD:** $l_2(\theta; x, x', r) = \frac{1}{2}(V(x;\theta) - r - \gamma V(x';\theta_{old}))^2$
**Bellman Error:** $\delta = Q(x,a;\theta) - r - \gamma \max_{a'} Q(x',a';\theta_{old})$
This can be seen as mean squared errors with labels $r - \gamma(x';\theta_{old})$. The labels change as we gain more experience
We can now learn a parameterised function to learn V. This can be done with a linear function approximation $Q(x,a;\theta) = \theta^T\phi(x,a)$, or via a neural network

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

$l_2(\theta; x, a, x', r) = \frac{1}{2}(\delta)^2$, $\nabla l_2() = \delta \nabla Q(x,a;\theta)$

**Linear:** $Q(x,a;\theta) = \theta^T\phi(x,a)$, $\nabla Q(x,a;\theta) = \phi(x,a)$
We assume the old Q value, parameterised by the old parameters, to be constant. This is the bootstrapping idea.

**Con:** $a_t = \max_a ...$, can be intractable, algo is slow
**Q-iter:** Keep $\max_{a'} Q(x',a';\theta^{old})$ for multiple iterations
The labels keep changing, this introduces stability issues in practice. By keeping a cloned network, the target labels don't jump around as much, leading to accelerated performance.
$L^{DDQN}(\theta) = \sum_{(x,a,r,x') \in D} (r + \gamma Q(x', a^*(\theta); \theta^{old}) - Q(x,a;\theta))^2$,
$a^*(\theta) = \arg\max_{a'} Q(x',a';\theta)$
**Maximation Bias:** The Q function of the target network seeks the max over all possible actions. It seeks this maximum on an already estimated quantity, which can lead to large bias. By using two seperate networks, the argmax can be obtained unbiased from the other network. There are multiple variations of this.
**Double:** We assume here that two different networks are maintained, one for optimizing the targets, and one for optimizing the value.

We have to backprop the a-network as well now.

**Policy parameterization:** $\pi(x) = \pi(x;\theta)$ $\tau^{(1)}, .., \tau^{(m)} \sim \pi_\theta$

$\theta^* = \arg\max_\theta J(\theta) \approx \arg\max_\theta \frac{1}{m} \sum_m^M \sum_t^T \gamma^t r_t^{(m)}$

We focus on episodic tasks here, reset the agent after some iterations or terminal after T timesteps. This generates trajectories depending on policy $\pi$. By Monte Carlo sampling we can globally optimize over the future discounted rewards.

A trajectory is of form $\tau^{(i)} = (x_0^{(i)}, a_0^{(i)}, r_0^{(i)}, x_1^{(i)}, a_1^{(i)}, r_1^{(i)}, ...)$

**Grad:** $\nabla J(\theta) = \nabla \mathbb{E}_{\tau\sim\pi_\theta} r(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)\nabla \log \pi_\theta(\tau)]$

$= \mathbb{E}_{\tau\sim\pi_\theta}[\sum_t^T (r(\tau) - b)\nabla \log \pi_\theta(a_t|x_t;\theta)]$

**Unbiased**, but very large **variance**

This means that we try to increase the probability of trajectories with high returns and decrease the probability of trajectories with low returns

From the MDP, we have $\pi_\theta(\tau) = p(x_0) \prod_t^T \pi(a_t|x_t;\theta) p(x_{t+1}|x_t, a_t)$,

thus $\nabla \log \pi_\theta(\tau) = \sum_t^T \nabla \log \pi(a_t|x_t;\theta)$

The large variance can be fought off by subtracting a baseline.

**Baseline:** $b(\tau_{0:t-1}) = \sum_{t'=0}^{t-1} \gamma^{t'} r_{t'}$, $G_t = \sum_{t'=t}^T \gamma^{t'-t} r'_t$

One could use different baselines, like constants or the mean

**Reinforce:** repeat: generate $\tau$, $\theta = \theta + \eta G_t \nabla_\theta \log \pi(a_t|x_t;\theta)$

$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_t^T G_t \nabla \log \pi_\theta(\tau)]$

$\nabla J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_t^T (G_t - b_t(x_t))\nabla \log \pi_\theta(\tau)]$

f.e. mean $b_t(x_t) = \frac{1}{T} \sum_0^T G_t$

# 13 Actor Critic

$A^\pi(x,a) = Q^\pi(x,a) - V^\pi(x) = Q^\pi(x,a) - \mathbb{E}_{a'\sim\pi(x)}[Q^\pi(x,a')]$
The advantage is the difference of the reward we getaaby taking action a in state x, and the expected reward we get in state x taking the action from our current policy.
When the advantage function is positive, it is an action we want to play, if it is negative, we rather not want to play the action. This is different than with the Q value.

**Policy Grad:** $\nabla J(\theta) = \mathbb{E}_{(x,a)\sim\pi_\theta}[Q(x,a;\theta_Q)\nabla \log \pi(a|x;\theta_\pi)]$
Gradient is expectation over state-action pairs, properly discounted by how much we visit this state action pair in $\pi$, times the action-value function of that state-action pair.
**Actor:** Parameterised policy
**Critic:** The parameterised estimate of the action-state value, also known as Q value.
When computing this expectation, we often use one-step approximations. This introduces variance, which we can reduce by subtracting a baseline again.

## 13.1 Online Actor Critic

$\theta_\pi \leftarrow \theta_\pi + \eta Q(x,a;\theta_Q)\nabla \log \pi(a|x;\theta_\pi)$
$\theta_Q \leftarrow \theta_Q - \eta_t(Q(x,a;\theta_Q) - r - \gamma Q(x', \pi(x',\theta_\pi);\theta_Q))\nabla Q(x,a;\theta_Q)$
Guaranteed to improve under certain conditions.

**A2C:** $\theta_\pi \leftarrow \theta_\pi + \eta[Q(x,a;\theta_Q) - V(x;\theta_V)]\nabla \log \pi(a|x;\theta_\pi)$
This is not unbiased anymore, since the Q value is only an approximation. If we would use the true Q value, we would have a convergence guarantee.
We can guarantee improvement by introducing compatibility conditions.
**Note:** On policy, use expectation over policy
**Bootleneck:** The problem with on-policy methods is that they have sample inefficiencies. We always need to sample from the current policy.

## 13.2 Parameterised Policies

$L(\theta_Q) = \sum_{(x,a,r,x')\in D}(r + \gamma Q(x', \pi(x';\theta_\pi);\theta_Q^{old}) - Q(x,a;\theta_Q))^2$
The initial motivation was $L(\theta)$, we can replace this exact (untractable) maximum with a parameterised policy.

**Target:** $\pi_G(x) = \arg\max_a Q^\pi(x,a) = \arg\max_a A^\pi(x,a)$

**Objective:** $\theta_\pi^* \in \arg\max_\theta \mathbb{E}_{x\sim\mu}[Q(x,\pi(x;\theta_\pi);\theta_Q]$
$\mu(x)$ must be complex enough to be able to explore all states, making it possible to converge to the maximum state.

**Grad:** $\nabla_\theta J_\mu(\theta) = \mathbb{E}_{x\sim\mu}[\nabla_a Q(x,a)|_{a=\pi(x;\theta_\pi)}\nabla_{\theta_\pi}\pi(x;\theta_\pi)]$
The policies gradient methods we saw before relied on randomized policies for exploration. This method uses a deterministic policy.
We can inject gaussian noise to encourage exploration.

**DDPG:** act after $a = \pi(x;\theta_\pi) + \epsilon, \epsilon \sim \mathcal{N}(0, \lambda I)$
$y = r + \gamma Q(x', \pi(x', \theta_\pi^{old}), \theta_Q^{old})$, update
$\theta_Q \leftarrow \theta_Q - \eta\nabla \frac{1}{|B|}\Sigma(Q(x,a;\theta_Q) - y)^2$, $\theta_Q^{old} \leftarrow (1-\rho)\theta_Q^{old} + \rho\theta_Q^{old}$
$\theta_\pi \leftarrow \theta_\pi + \eta\nabla \frac{1}{|B|}\Sigma Q(x, \pi(x;\theta_\pi);\theta_Q)$, $\theta_\pi^{old} \leftarrow (1-\rho)\theta_\pi^{old} + \rho\theta_\pi^{old}$
We here have random Gaussian noise injected to explore.