Probabilistic Artificial Intelligence

A. Krause

Lecture 1 - Probability Review

1.1 Probability Space

Okay so We start with defining a **probability space**, and the idea with that thing is just to give You everything You need to do probabalistic stuff. Without further ado, a probability space is a triple consisting of $(\Omega, \mathcal{A}, P)$.

Ω is called the **sample space** and is just a set of all possible outcomes of an experiment. For the flip of two coins, You get $\Omega = \{HH, HT, TH, TT\}$.

$\mathcal{A}$ is the **event space**, and it is usually thought of as every possible combination of elements of Ω. So $\mathcal{A}$ is a sort of a collection of subsets of Ω. But why is it defined this way? Well, the idea is that *combinations* of outcomes are events. For instance, an event might be getting one or more tail, and that event corresponds to a subset $\{TH, HT, TT\} \in \Omega$. You can come up with much more complicated events in case of more complicated sample spaces. Point here is that We just have a set of all possible events, which We will now use in:

P - think of P as function $P : \mathcal{A} \to [0, 1]$. All P does is assign a probability for each event in $\mathcal{A}$.

With those three constructs We can talk about outcomes of an experiment, group outcomes of an experiment into events, and talk about the probabilities of those events. There are of course some other constraints for this all to make sense, like $P(\Omega) = 1$ and $\forall S \subset \Omega : P(S) \in [0, 1]$).

Random variables then can be thought of as functions - let X be a random variable, then $X : \mathcal{A} \to \mathbb{R}$, roughly. The output range depends on the variable - is it discrete, continuous, is it univariate or multivariate etc.

Then You have three axioms for it all to work:

Normalization: $P(\Omega) = 1$, so probability of all possible outcomes must equal to 1.

Non-negativity: $S \in \mathcal{A} \implies P(S) \geq 0$.

σ-additivity - probabilities of disjoint events can simply be added.

Then You can come up with stories for random variables and get distributions like Bernoulli for 1 flip of a coin, Binomial for many flips, multinomial for dice outcomes etc.

If the variable is continuous You get a PDF and a CDF, PDF must be non-negative and CDF must integrate to 1 etc.

Then You get Your Gaussian, Your vector of random variables and a **joint distribution** (meaning You specify a value for each variable in vector).

Then You get Your conditional distribution defined by

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

In it's simplest form. That becomes Bayes' Rule if You observe the **product rule** $P(B|A)P(A) = P(A \cap B)$. You can keep taking stuff out too, so

$$P(A, B, C) = P(A|B, C)P(B|C)P(C)$$

That's the chain product rule.

The other important rule is the **sum rule** (or marginalization), in that

$$P(X_1 = y, X_2 = z) = \sum_{x \in \sup X_3} P(X_3 = x)P(X_1 = y)P(X_2 = z)$$

While We're here, let's also clear up **prior, likelihood** and **posterior** probabilities.

I suppose talking about distributions makes sense - the prior distribution of X is, well, the distribution of X without any additional information. Just what We know about X before the experiment.

Likelihood is $P(E|X = x)$ and should really be thought of as a function that takes the value of X, i.e. what X turned out to be, and it gives You the probability of evidence E given that $X = x$. In English one might say that $P(E|X = x)$ is the likelihood that We observe E given $X = x$. X is usually some hidden variable(s) that We say generated evidence E.

Posterior is the same as conditional probability - $P(X|E)$, or distribution of X given that some evidence is true. The distinction between posterior and likelihood is that in likelihood We are talking about the likelihood of something else given the variable We care about X, where as in the posterior We are interested in adjusting the distribution of our variable X given some evidence.

Then there's independence and conditional independence, no worries there.

High dimensional (binary in the most trivial case) multivariate distributions require an exponential number of parameters to fully specify. Marginalizing out variables runs into the same problem, so, We have as problems:

Representation (how to represent high dimensional distributions in a not parameter-exponential way), learning (given data, learn the distribution that produced it) and inference (given a distribution, make predictions)

1.2 Gaussians

$$P(X = x) = \frac{1}{2\pi\sqrt{|\Sigma|}} \cdot \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right)$$

With μ being a mean of vectors of Σ being the covariance matrix.

etc. Really when it comes to the formula don't memorize them. Just remember that adding Gaussians or conditioning Gaussians on Gaussians results in a Gaussian, multiplying them generally doesn't.

Lecture 2 - Bayesian Linear Regression

The idea is simple - ordinary linear regression (and variants thereof) yield a point estimate $\hat{y}$. We'd like to know how uncertain We are about this estimate.

2.1 Ridge Regression as Bayesian Inferece

Recall that We can find the coefficients for the best linear fit by doing

$$X^\top(X\mathbf{w} - \mathbf{y}) = 0 \tag{1}$$
$$X^\top X\mathbf{w} - X^\top\mathbf{y} = 0 \tag{2}$$
$$X^\top X\mathbf{w} = X^\top\mathbf{y} \tag{3}$$
$$\mathbf{w} = (X^\top X)^{-1}X^\top\mathbf{y} \tag{4}$$

Well let's flesh that out a little as a refresher. Imagine a line on the Cartesian plane, and imagine projecting points on to that line. Now pick a point x_i and it's projection p_i. Observe that if $x_i - p_i$ is not perpendicular to the line then the projection is shit - the original vector x_i can be thought of as having components in the span of the line of best fit, whatever that line may be, and stuff that is perpendicular to that line. We want to get at x_i and ignore all the stuff perpendicular to this hypothetical best line.

We can then of course arrive at the first line in above derivation - it's saying that the projection (hence $X^\top$) of the difference between the projection $X\mathbf{w}$ and the truth $\mathbf{y}$ must be perpendicular $= 0$. Voila.

Now, in order to get the solution for ridge regression We do

$$\mathbf{w} = (X^\top X + \lambda I)^{-1}X^\top\mathbf{y} \tag{5}$$

Is the modification for ridge regression to penalize large coefficients in $\mathbf{w}$.

So, now, let's make things probabilistic:

For $\mathbf{w}$, let's assume that $\mathbf{w} \sim \mathcal{N}(0, \sigma^2_{\mathbf{W}})$ and also let's assume that $\mathbf{w} \perp \mathbf{x}_i \ \forall i \in [n]$, so We have a normal prior over the weights and a-priori the weights are independent of the data - without knowing anything about $\mathbf{x}_i$, this is all We've got.

Then assuming You have the true weight vector $\mathbf{w}$:

$$P(\mathbf{y}_i|\mathbf{w}, \mathbf{x}_i) \sim \mathcal{N}(\mathbf{w}^\top\mathbf{x}_i, \sigma^2_{\mathbf{y}}) \tag{6}$$

So We're assuming that given the one true weight vector, our labels are normally distributed around the predicted mean. They are also of course independent, and all have the same variance.

So now the idea is - We have a prior on the weights, and We have a likelihood function which involves the weights. By the glory of Bayes' rule, that's enough to try to calculate a posterior on the weights:

$$P(\mathbf{w}|\mathbf{x}_{1\ldots n}, \mathbf{y}_{1\ldots n}) = \frac{1}{z} \cdot P(\mathbf{w}, \mathbf{x}_{1\ldots n}, \mathbf{y}_{1\ldots n}) \tag{7}$$

$$= \frac{1}{z} \cdot P(\mathbf{x}_{1\ldots n}) \cdot p(\mathbf{w}|\mathbf{x}_{1\ldots n}) \cdot P(\mathbf{y}_{1\ldots n}|\mathbf{w}, \mathbf{x}_{1\ldots n}) \tag{8}$$

$$= \frac{1}{z'} \cdot p(\mathbf{w}|\mathbf{x}_{1\ldots n}) \cdot P(\mathbf{y}_{1\ldots n}|\mathbf{w}, \mathbf{x}_{1\ldots n}) \tag{9}$$

$$= \frac{1}{z'} \mathcal{N}(0, \sigma^2_{\mathbf{W}}) \cdot \prod_{i=1}^{n} \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \sigma^2_{\mathbf{y}}) \tag{10}$$

$$= \frac{1}{z'} \frac{1}{z_{\mathbf{w}}} \exp\left(-\frac{1}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2\right) \cdot \frac{1}{z_{\mathbf{y}}} \prod_{i=1}^{n} \exp\left(-\frac{1}{\sigma^2_{\mathbf{y}}} \cdot \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2\right) \tag{11}$$

So, at 7 We are simply using the whole $P(A|B) = P(A, B)/P(B)$ thing, to rearrange the terms any way We like - the point is to have one conjunction above and one below, and factorize the above conjunction with the chain rule in a way that can leverage our assumptions.

Then at 9, We absorb that $P(\mathbf{x}_{1\ldots n})$ term since it's kind of irrelevant - just some Gaussian that We'll ultimately not care about.

Finally We use an assumption or two.

Now, note that when fitting $\mathbf{w}$, We're going to maximize our result. Since We just take about an extrema, We can start stripping parts away:

$$\arg\max_{\mathbf{w}} = \frac{1}{z'} \frac{1}{z_{\mathbf{w}}} \exp\left(-\frac{1}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2\right) \cdot \frac{1}{z_{\mathbf{y}}} \prod_{i=1}^{n} \exp\left(-\frac{1}{\sigma^2_{\mathbf{y}}} \cdot \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2\right) \tag{12}$$

$$= \exp\left(-\frac{1}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2\right) \cdot \prod_{i=1}^{n} \exp\left(-\frac{1}{\sigma^2_{\mathbf{y}}} \cdot \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2\right) \tag{13}$$

$$= \exp\left(-\frac{1}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2\right) \cdot \prod_{i=1}^{n} \exp\left(-\frac{1}{\sigma^2_{\mathbf{y}}} \cdot \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2\right) \tag{14}$$

$$= \exp\left(-\frac{1}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2 - \sum_{i=1}^{n} \frac{1}{\sigma^2_{\mathbf{y}}} \cdot \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2\right) \tag{15}$$

$$= -\frac{1}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2 - \sum_{i=1}^{n} \frac{1}{\sigma^2_{\mathbf{y}}} \cdot \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2 \tag{16}$$

$$\arg\min_{\mathbf{w}} = \frac{\sigma^2_{\mathbf{y}}}{\sigma^2_{\mathbf{w}}} \|\mathbf{w}\|^2 + \sum_{i=1}^{n} \|y_i - \mathbf{w}^\top \mathbf{x}_i\|^2 \tag{17}$$

Where in 16, We just multiplied by the positive constant $\sigma^2_{\mathbf{y}}$ to get a clean coefficient for the λ term.

Anyway this shows that if We choose $\lambda = \frac{\sigma^2_{\mathbf{y}}}{\sigma^2_{\mathbf{w}}}$, then the solution to the ridge

regression problem is equivalent to finding the maximum a posteriori solution. ($P(\mathbf{w}|\mathbf{x}_{1...n}, \mathbf{y}_{1...n})$ is the posterior in question).

2.2 Distribution of the weights

As a by the by - I want to see what ridge regression looks like in the linear algebra sense:

$$\mathbf{w} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y} \tag{18}$$

$$(X^\top X + \lambda I)\mathbf{w} = X^\top \mathbf{y} \tag{19}$$

$$X^\top X\mathbf{w} + X^\top X\lambda I\mathbf{w} = X^\top \mathbf{y} \tag{20}$$

$$X^\top X\mathbf{w} + X^\top X\lambda I\mathbf{w} - X^\top \mathbf{y} = 0 \tag{21}$$

$$X^\top (X\mathbf{w} + X\lambda I\mathbf{w} - \mathbf{y}) = 0 \tag{22}$$

$$X^\top (X\mathbf{w} - \underbrace{\mathbf{y} + X\lambda I\mathbf{w}}_{\mathbf{y}'}) = 0 \tag{23}$$

Well You can stare at this and try to come up with intuitions.

It looks like instead of targeting just the labels, We're targeting labels plus a small adjustment. Weird.

Or

$$X^\top (X\mathbf{w} - \underbrace{\mathbf{y} + X\lambda I\mathbf{w}}_{\mathbf{y}'}) = 0 \tag{24}$$

$$X^\top (X\mathbf{w} + X\lambda I\mathbf{w} - \mathbf{y}) = 0 \tag{25}$$

$$X^\top (X\mathbf{w}(I + \lambda I) - \mathbf{y}) = 0 \tag{26}$$

So this equation will need to be zero, which means that $\mathbf{w}$ will end up need to be zero. Neat.

Aaaaaanyway, We make an assumption about the prior distribution of the weights, namely:

$$\mathbf{w} \sim \mathcal{N}(0, I) \tag{27}$$

Then, the likelihood of data given our weights is:

$$p(y_i|\mathbf{w}, x_i, \sigma_n) = \mathcal{N}(y_i; \mathbf{w}^\top x_i, \sigma_n^2) \tag{28}$$

So all We're saying here is that We assume, as We ought to if We want to test the weights, that the weights generated the data. We also assume that the target label has some **aleatoric** uncertainty, so every label has some noise - this is captured by that σ_n term.

Given a prior and a likelihood We can try to calculate a posterior for the weights.

aside

So this business about prior and likelihood - it goes like this:

Recall Bayes' rule, where the thing We care about is A (so in the context of Bayesian regression it's our weight vector $\mathbf{w}$):

$$\underbrace{P(A|B)}_{posterior} = \frac{\overbrace{P(B|A)}^{likelihood}\overbrace{P(A)}^{prior}}{\underbrace{P(B)}_{normalization}} \tag{29}$$

Since our goal is to pick the best model, usually, We can ignore the normalization and look for the argmax of the numerator. Prior and likelihood give us enough information for this.

A wordsy way of putting it might be something like - prior tells us how likely a given set of weights is without looking at data, and likelihood tells us how likely the data is given a set of weights. So to maximize the expression, We need to find a set of weights that is both probably w.r.t. prior and explains the data well.

In the special case of Bayesian regression, since everything We're conditioning on is Gaussian, We can calculate the whole thing.

We have a distribution for the weights:

$$\mathbf{w} \sim \mathcal{N}(\mu_w, \Sigma_w) \tag{30}$$

$$\mu_w = (X^\top X + \sigma_n^2 I)^{-1} X^\top \mathbf{y} \tag{31}$$

$$\Sigma_w = (\sigma_n^{-2} X^\top X + I)^{-1} \tag{32}$$

The above ought to follow from the conditioning rules about Gaussians.

So, let's talk about those terms. The mean is just the solution to ridge regression with $\lambda = \sigma_n^2/\sigma_p^2$, with the numerator there being aleatoric uncertainty in the labels. This makes sense - the larger the prior uncertainty, the larger the λ, the more We should defer to the prior. Inverse for the denominator.

Now for the variance - it is the inverse of the "precision" matrix, which I suppose captures how sure We are about an estimate, as opposed to how much it varies.

In there We have an inverse of the label variance, and We're taking the inverse of that again, so the smaller the label variance the smaller the posterior variance for the weights. Makes sense.

2.3 Predictions in Bayesian Linear Regression

So now that We have a posterior on the weights given the data, We can make predictions, since predictions are just linear combinations of the weights vectors.

Suppose We want to make a prediction about a new point $\mathbf{x}^*$. Then $f^* = \mathbf{w}^\top \mathbf{x}$ and

$$f^* \sim \mathcal{N}(\mu_w^\top \mathbf{x}^*, \mathbf{x}^{*\top} \Sigma_w \mathbf{x}^*) \tag{33}$$

There's a crucial distinction to be made though - f^* is a prediction, but it's not account for the fact that each label has it's own independent noise term. To get to an actual label prediction We need to

$$y^* = f^* + \varepsilon \sim \mathcal{N}(\mu_w^\top \mathbf{x}^*, \mathbf{x}^{*\top} \Sigma_w \mathbf{x}^* + \sigma_n^2) \tag{34}$$

And We're done.

2.4 Hyperparameters

So We have two main ones - variance of prior and variance of labels.

First We use cross validation to get a good $\hat{\lambda}$.

Then estimate variance of labels using residuals.

Then solve for variance of prior assuming lambda is prior/labels.

2.5 Recursive Updates

First assumption - given the weights, the labels are independent. So:

$$P(y_{1\ldots n}|\mathbf{w}) = \prod_{i=1}^{n} P(y_i|\theta) \tag{35}$$

Well actually none of that is relevant - the solution to the homework simply observes that

$$\mu_w = (X^\top X + \sigma_n^2 I)^{-1} X^\top \mathbf{y} \tag{36}$$

$$\Sigma_w = (\sigma_n^{-2} X^\top X + I)^{-1} \tag{37}$$

and now We want to add one more column to X.

But We ain't gotta, since We just need to keep track of what vectors $X^\top X$ and $X \top \mathbf{y}$ resolve to.

In the case of $X \top X$ it's

$$X_{new}^\top X_{new} = X^\top X + x_{new} x_{new}^\top \tag{38}$$

And

$$X_{new}^\top \mathbf{y} = X^\top \mathbf{y} + y_{new} x_{new} \tag{39}$$

2.6 Kalman Filters

Just look at lecture notes.

Gaussian Processes

3.1 Bayesian Regression is a Kalman Filter

Where basically You're trying to pin down the variance of the weights vector.

You need to change things around a little bit - with each new datapoint, the weights change, so Your kalman update is different (but deterministic) at each step, with no noise.

Then the observation is the label and You're trying to pin down Your uncertainty about the state - the weights vector.

3.2 Non-linear Bayesian Regression

Is often done by just transforming the feature space - that way You can perform linear regression on non-linear features.

However, this is often prohibitively expensive and complicated. Just putting the features into the polynomial space is exponential in cost.

So what do?

3.3 Kernel Trick

We can observe that Bayesian Regression only really depends on the dot product between vectors.

We can yoink that out, replace it with a **Kernel**, which is a function of our choice (that has to satisfy some conditions) and We're good.

But how to do this rigorously lol.

3.4 Weight vs Function Space view

So, We put a prior on the weights (which We were doing anyway), and We get some deterministic data.

Then We update the weights based on evidence, which still yields us random variable weights, so it's sort of like We're getting a whole family of functions here in a sense - You could sample weight vectors.

Let's get started on this notation:

$$f = [f_1, f_2 \dots f_n] \tag{1}$$

$$f_i = \mathbf{w}^\top \mathbf{x_i} \tag{2}$$

$$\mathbf{f} = X\mathbf{w} \tag{3}$$

So f are our predictions (not to be confused with out predicted labels which will have a little more variance), and since our predictions are just linear combinations of gaussians, so We have

$$\mathbf{f} \sim \mathcal{N}(0, XX^\top) \tag{4}$$

So obviously this is a bit weird since We aren't updating the weights here based on the data, is what I'm seeing, but that'll hopefully clear up later. I think it's because We're not conditioning in anything. It's the conditioning that changes stuff.

So then if We want to predict a new datapoint, We formulate it as follows:

We slap on another row onto X and call that new matrix $\tilde{X}$.

Similarly slap on another label on to the $\mathbf{y}$ vector and call it $\tilde{\mathbf{y}}$.

So then same as before We get new predictions $\tilde{f} = \tilde{X}^\top \mathbf{w}$ and similarly $\tilde{\mathbf{y}} = \tilde{f} + \varepsilon$ where ε is just a vector of i.i.d. noise.

So then We have $\tilde{\mathbf{y}} \sim \mathcal{N}(0, \tilde{X}\tilde{X}^\top + \sigma_n^2 I)$ where $\sigma_n^2 I$ is just the variance of the noise.

And so then to make a prediction You just do

$$P(y^*|y_{1\ldots n}) \sim \mathcal{N}(\mu_{x^*|x_{1\ldots n}, y_{1\ldots n}}, \Sigma_{x^*|x_{1\ldots n}} + I\sigma_n^2) \tag{5}$$

Notice that the covariance does not depend on the labels. This is because all that information is captured in the mean I think, and the variance of the labels We know anyway.

Notice also that We no longer really need the weight vector - everything is a function of dot products of observed data. We just plop in our kernel in place of those dot products to get where We're going. Our weight vector, is my understanding, becomes infinitely dimensional, in a sense.

3.5 Gaussian Processes

A Gaussian Process is an infinite set of Gaussian Variables. What this really means is that if We have some input space $\mathcal{X}$, then for each point in that space (and since the space is probably continuous there's an infinite number of them),We've got a corresponding y which satisfies certain conditions, and those conditions are of course such that y is a useful prediction.

To make this rigorous We need two bits:

First is a kernel function. Let $\mathcal{X}$ be our input space. Then the kernel function does $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

Now We basically never use the kernel on it's own. Instead, We construct matrices with it.

Let $A = [a_1 \ldots a_n], a_i \in \mathcal{X}$, so it's just some set of vectors from the input space (they don't actually need to be vectors, but probably usually are). Let also $B = [b_1 \ldots b_n], b_i \in \mathcal{X}$.

Then when We write $K_{A,B}$ We are creating a matrix such that $K_{i,j} = k(A_i, B_j)$. This K needs to be positive semidefinite for any A, B in order to be a valid kernel.

So that's the first bit - K is our covariance matrix.

The second bit is $\mu : \mathcal{X} \to \mathbb{R}$, which is our mean function.

So, to get back to our Infinite Gaussians.

Let our infinite set of Gaussians be indexable by some set $\mathcal{X}$. The point here is that for each point in our input space We have a corresponding Gaussian in Infinite Gaussians set.

Now We impose conditions - let $A \subseteq \mathcal{X}$, $A = [\mathbf{x}_1 \dots \mathbf{x}_n]$, so A is our finite subset of that infinite set.

It must be true that $\forall A \subseteq \mathcal{X}$ that there exists a $Y_A = [Y_{x_1} \dots Y_{x_n}] = \mathbf{f} = [f_1 \dots f_n]$ such that $Y_A \sim \mathcal{N}(\mu_A, K_{AA})$.

So what does that mean - basically our GP is a set of like "label" random variables, which is indexed by $\mathcal{X}$ which just associates predictors with labels, and each label is a Gaussian with mean such and such and covariance such and such, and that mean and covariance are the bits We established at the start of the subsection.

Since We're ever only interested in a finite subset of that infinite number of Gaussians, We can have an infinite number of them just fine.

Another perspective is that We've taken the data and put it into a potentially infinite-dimensional space with our kernel, and that what We get back after all the linear algebra is a Gaussian with some mean and variance.

3.6 Kernel Functions

They gotta be symmetric, so $K_{AB} = K_{BA}$, and K_{AB} has to be positive semidefinite.

3.7 Kernel Examples

3.7.1 Linear Kernel

Good old dot product, yields back Bayesian Linear Regression.

3.7.2 Squared Exponential Kernel or Radial Basis Function

$$k(a, b) = \exp\left(\frac{-\|a - b\|_2^2}{h}\right) \tag{6}$$

There's a parameter in there - h - called the bandwidth parameter. The larger the bandwith the less it matters what the two points actually are, which makes the output smoother. The smaller the bandwidth the more sensitive the function is to differences in the numerator.

3.7.3 Exponential Kernel

So now We're using the l_1 norm and also not squaring it:

$$k(a,b) = \exp\left(\frac{-\|a-b\|_1}{h}\right) \tag{7}$$

This is v. erratic and nowhere differentiable. Varies all over the place in small scale. The larger the bandwith the less it varies *on a large scale*. Still all over the place in the small.

3.7.4 Matern Kernel

Whose precise form is I don't think needed.

Allows to interpolate between infinitely differentiable RBF kernel and the nowhere differentiable exponential kernel.

Has a parameter v where if You let $v = 0.5$ then You get back the exponential kernel, and as $v \to \infty$ You get RBF.

v basically lets You specify how many times the function ought to be differentiable.

3.8 Kernel Composition Rules

How do compose kernels (or covariance functions)

You can add them, multiply them, scale by constant or stick them into a polynomial or exponentiate them

3.9 Stationary v.s. Isotropic

A kernel is stationary if the value of $k(a,b)$ depends **only** on $a-b$. So if You like rotate move the vectors around but keep the difference the same, You're good.

A kernel is isotropic if $k(a,b)$ only depends on $\|a-b\|_2$, so just the norm of the difference.

Linear kernel is neither, RBF is both, and if You stick a matrix into RBF like $-(a-b)^\top M(a-b)$ then it's stationary but not isotropic.

3.10 Making predictions with GPs

You have some GP, which is to say You have a mean (I think the mean function is solution to kernel ridge regression?) and a covariance function, so μ, k.

Then You get a training set with a bunch of pairs of x_i, y_i.

Then You take the points You want to predict, attach them on to the dataset and run the formulas.

And then You sample from the updated GP, I think?

Posterior variance does not depend on observed values, just the Xs.

3.11 How to sample for a GP

Main point is that You sample sequentially - so You draw a sample, then conditioning on the fact that the first sample turned out to be a You sample the next bit and so forth. Called "forward sampling"

3.12 Kalman Filters are GPs

In the sense that You can define the relationship between any two datapoints and call that relationship k. The mean turns out to just be 0, depending on Your assumptions.

3.13 Demo

So the smaller the bandwith, the larger the negative term in the exponent of the RBF, the closer to zero the covariance, the more We treat the terms as independent and the more erratic the function.

3.14 Optimizing Kernel Parameters

So crossvalidation is the easy way of going about this.

There's a wilder way though - We start by saying well hey let's just choose θ such that the observed data is as likely as possible.

Then You say well hey, let's split the probability of the data into probability of the of the labels given model, and probability of model given parameters. The first part is uncontroversial as far as I can see.

The second bit is our prior on just θ I think.

The point is that We'll be integrating out the model (see lecture slides), so We'll in some sense be looking the entire space.

And so the benefit is as follows: there is a part of the term to be optimized that increases when the data is fit well, which is great, but depending on our prior the extreme-ish values that allow for this perfect fit are unlikely, so they'll cancel.

It's just a wonky regularization in the end.

3.15 GP computation issues

Naively, slapping new datapoints on to the design matrix and running the marginalization formulas involves a matrix inverse, which is cubic in number of rows, so We get n^3 complexity where n is the number of data points, which is bad.

So how solve?

3.16 Hardware acceleration

With GPUs etc.

3.17 Local methods

The idea is to observe that the larger the distance between points the smaller a contribution they make, so just consider close by points.

3.18 Kernel Function Approximation

A whole lot of this I'm skipping and just focusing on the fact that inducing points are like summarizing data before sticking it into GPs to lower the number of data points.

Variational Inference

4.1 The General Bayesian Picture

So You have yer prior $p(\theta)$, what You I suppose think some parameters of interest will be ahead of time.

You have Your likelihood of data - $P(y_{1...n}|\theta, x_{1...n})$. This is specified by model assumptions - given θ We probably have some model that generates probabilities, give x_i.

Then We have the posterior $P(\theta|y_{1...n}, x_{1...n})$. So the probability of weights given all the stuff We've seen. We want this if We want to capture uncertainty in our predictions, since then hopefully if our model is simple for example the distribution of our predictions will just be Gaussian or whatever.

The problem is this:

$$P(\theta|y_{1...n}, x_{1...n}) = \frac{P(\theta, y_{1...n}|x_{1...n})}{P(y_{1...n}|x_{1...n})} \tag{1}$$

$$= \frac{P(\theta, y_{1...n}|x_{1...n})}{\sum_\theta P(\theta)P(y_{1...n}|x_{1...n}, \theta)} \tag{2}$$

So We always condition on data since that's just a given.

$$P(\theta, y_{1...n}|x_{1...n}) = P(\theta|x_{1...n})P(y_{1...n}|\theta, x_{1...n}) \tag{3}$$

$$= P(\theta)P(y_{1...n}|\theta, x_{1...n}) \tag{4}$$

Where the last simplification is just that the weights do not depend on data unless We know the labels.

Anyway it's the numerator that's the problem, which really ought to be written as

$$Z = \int P(\theta)P(y_{1...n}|x_{1...n}, \theta)d\theta \tag{5}$$

So in order to normalize the probability of our parameters, We gotta perform this integral which may not be tractable.

Then finally in order to do prediction We do

$$p(y^*|y_{1...n}, x_{1...n}) = \int P(y^*|y_{1...n}, x_{1...n}, x^*\theta)P(\theta|y_{1...n}, x_{1...n})d\theta \tag{6}$$

Which simplifies since given the parameters the new prediction is independent of past data.

$$p(y^*|y_{1...n}, x_{1...n}) = \int P(y^*|\theta)P(\theta|y_{1...n}, x_{1...n})d\theta \tag{7}$$

Wee.

4.2 Laplace approximation

Fit a Gaussian at the mode of the target distribution.

The reason for introducing this is that We're approximating the true distribution with this fitted Gaussian.

And then there's a bunch of examples of how this fails - the shape is wrong is most of it.

4.3 Variational Inference

It's called Variational because I think We vary parameters to find the best substitute distribution.

So! The idea is that the true posterior is a nightmare, so We're going to find the most similar "simple" distribution which is supposed to approximate the nightmare, and We'll work with the simple version.

The search is done over a family of distributions specified by parameters e.g. exponential family or the Gaussians.

4.4 KL-Divergence

But how do search? Well We optimize a "distance"ish metric. The smaller the KL-Divergence between two distributions the more similar they are.

KL non negative, zero only if P, Q are identical, and asymmetric.

4.5 ELBO

The point is that minimizing KL-divergence can be rewritten as attempting to minimize KL-divergence between posterior and prior while simultaneously trying to maximize the likelihood of the data. Basically yet again here We are with fancy regularization.

Markov Chain Monte Carlo

So what is this?

Alright recall the original problem - We can get the posterior just not normalized, which it makes it a bit probabilistically useless.

One thing You can do is variational inference and optimize the KL divergence and so forth.

Another is to simply try to sample from the unknown distribution and obtain an approximation of the distribution that way.

Anything naive will fail miserably, so We need to come up with some smart way of sampling

Enter MCMC - our next sample depends on where currently We are in the distribution.

The Markov Chain in question has to be **Ergodic** and has to satisfy the **Detailed Balance Equation**.

Ergodicity is simple-ish. Given any state, You must be able to reach any other state in some t transitions.

Detailed Balance simply means that if I'm at some point x, and I'm thinking of going to x', then probability of doing that must be the same as if I were at x' and thinking of going to x.

5.1 Metropolis Hastings

Goes like this: You have some **proposal** distribution R, so, $R(x)$ will give You the probability of x. You use this proposal distribution to, well, propose points to move to in the distribution You're trying to approximate.

Then You have Q, which is the unnormalized target distribution.

Suppose $X_t = x$, then with probability

$$\alpha = \min\left\{1, \frac{Q(x')}{Q(x)}\frac{R(x|x')}{R(x'|x)}\right\} \tag{1}$$

Set $X_{t+1} = x'$.

And with probability $1 - \alpha$ We just set $X_{t+1} = x$, so We never like "stop", even if We reject a proposal there is still a new entry in the chain.

But what about that wonky expression for α? Well focus in on the two terms - the latter is the easy one. I think it's point is to establish the detailed balance

The other term then is simply a ratio of probabilities - if the point x' is likelier under the unnormalized distribution then go there.

It ain't super sophisticated.

5.2 Gibbs Sampling

In a multivariate case, instead of sampling all the covariates, free most of 'em and just sample a few at a time. This has the advantage of simplifying the distribution maybe, since You may get conditional independence.

It also amounts of exploring the distribution along the axes to which the coefficients correspond, and You may need to align them for this to work.

Marginalizing out one variable at a time is usually also okay, is how come We can get away with it.

Active Learning

6.1 Bayesian Learning

Is concerned with learning the function everywhere.

We model the function with a Gaussian Process is the setup, and then We pick points such that the uncertainty about our function is maximally reduced.

Now You need to keep in mind that there are two kinds of uncertainty - epistemic, due to not knowing the function, and aleatoric which is just noise from on high.

To take care of both You can optimize epistemic/aleatoric as a criterion for choosing points. The idea being is that if there's an ocean of aleatoric noise someplace then it may not be worth exploring.

6.2 Bayesian Optimization

The other problem is that We may be more interested in just finding the optimum rather than exploring the whole function.

So now We're dealing with a tradeoff - controlled by a parameter - about how much exploration to do v.s. how much optimization to do.

The idea is to minimize average regret, where regret is $r_t = f(x^*) - f(x_t)$.

A way of doing this is to look at the lower bounds created by the Gaussian Process - so look at the underside of the possible functions, and then take the point that is performing the best and use that to form a floor, and explore most promising venues which lie above that floor.

Other criteria exist, and other ways of sampling exist, for example Thompson sampling will draw a function from the GP and pick the best point from that function.

Markov Decision Processes

A set of actions A, of which there are m.

A set of states X, of which there are n.

A set of transition probabilities $P(x'|x, a)$, so there are $n \times m$ of those.

A reward function $r(x, a)$ - this is really flexible, just a function (or a random function) which dishes out rewards. Can depend on as much stuff as You like.

We assume that the reward function and transition probabilities are known.

7.1 Planning in MDPs

We can either have a deterministic policy $\pi : X \to A$ or a probabilistic one $\pi : X \to P(A)$.

For a deterministic policy We can pretty easily write down the expected reward with a discount factor γ, which γ specifies how much or little We care about rewards in the distant future.

$$J(\pi) = E[r(X_0, \pi(X_0)) + \gamma r(X_1, \pi(X_1)) + \ldots \gamma^n r(X_n, \pi(X_n))] \tag{1}$$

7.2 Value Function

So, how much is it worth to be in a state?

Well if You think about it, the value of a state really depends on how well You can exploit as a function of that state. With that in mind:

$$V^\pi(x) = J(\pi | X_0 = x) = E\left[\sum_{t=0}^{\infty} \gamma^t r(X_t, \pi(X_t)) | X_0 = x\right] \tag{2}$$

It doesn't actually take a lot of imagination to show that

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x' \in \pi(x)} p(x' | X_0 = x, \pi(x)) V^\pi(\pi(x')) \tag{3}$$

Which is pretty intuitive if You think about it - the value of state x is the value of the action You'll take plus discounted values of all states You might end up in, discounted again by their probabilities.

Turns out You can actually use linear algebra to precisely solve for these values:

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x' \in \pi(x)} p(x' | X_0 = x, \pi(x)) V^\pi(\pi(x')) \tag{4}$$

The idea is to rewrite the above as a linear system, so

$$V^\pi = r^\pi + \gamma T^\pi V^\pi \tag{5}$$

Where V^π is just a vector such that $V_i^\pi = V^\pi(x_i)$.

r^π is the same idea but rewards instead.

Okay and finally We have that T^π which is also really simple - just a matrix such that the i, j entry is $P(x_j | x_i, \pi)$. We're just reconstructing the first equation in matrix form, term by term.

7.3 Fixed Point Iteration

Can apparently solve the system just be running the update equations over and over, i.e. initialize the value to whatever and just keep doing

$$V_t^\pi = r^\pi + \gamma T^\pi V_{t-1}^\pi \tag{6}$$

7.4 Policies and Values

Thing is, depending on what Your policy is, the values of states change, and depending on the values another policy might be optimal.

The idea is then to iterate, set some rando policy, then calculate the values, then pick the greedy policy, then recalculate the values and so on. The error converges exponentially.

7.5 Value Iteration

So the previous method is called policy iteration, even though it seems a lot like You're iterating values.

Another way of going about is "value iteration", and it goes like this.

Initialize value vector to just be the greedy max of value of neighbours.

Then for each timestep:

Calculate $Q(x, a)$ for each pair of state and action, where Q is the value of reward + probability of going to others states times their value. So $Q(x, a)$ is the value of taking action a at state x.

Then You pick the max for each state, i.e. maximize over actions.

If the difference between new value and old value is less than some epsilon You're done.

7.6 Tradeoffs

Value iteration is more espessive in that You need to solve a linear system for each iteration. So n^3 I think. And then polynomial number of steps.

Policy iteration is $\mathcal{O}(n \times m \times s)$, where s is like the branching factor, so if the thing is very sparse i.e. not very connected as a graph, then that may be faster.

Reinforcement Learning

8.1 Problems

We chose the data We gather and it ain't i.i.d.

Also the problem of how much exploration v.s. exploitation to do.

8.2 Ways of doin RL

Can be broken down into model-based and model-free.

Model-based explicitly tries to map the underlying Markov Decision Process - so We're trying to learn transition probabilities and the reward function, and once We have those We can do policy/value iteration and solve the problem.

The other way is to not actually explicitly model the MDP and instead just focus on modeling like the values directly, which I think is what a lot of cutting-edge methods do.

8.3 Off v.s. On-policy

So off-policy RL basically does not get to make decisions, just gets to observe what happens out there, somewhere.

This may be a problem depending on how You're learning - like the model-free RL approach depends on trying to take optimal steps which, if You can't do that, You're screwed.

On-policy methods on the other hand get full control and also get to choose the exploration/exploitation tradeoff.

8.4 Learning the MDP

Is done via just the mean of stuff - probability of transitions, reward functions etc.

Crucially, given a state, the probabilities of going places are independent, so You get exponential convergence if You just average stuff.

8.5 Exploitation/Exploration

One thing to do is be ϵ-greedy, so with probability ϵ You take a random action instead of a greedy action.

There is a problem with this in that this policy will continue doing horrible things even if We know they are horrible with some non-zero probability. Why keep bashing Your head against the wall?

8.6 R-max

So, how to avoid doing that? Two changes to the algorithm.

So We have to learn two things for the MDP - transition probabilities and rewards.

We set unknown rewards to be R_{max}, which means We need some sort of upper bound for the reward for all this to work.

So when We don't know the reward of an action, it's max, and afterwards We update it with what We observed.

We also don't know probabilities of transition. The idea there is to just say that if You don't know, them assume that with probability 1 the action takes

You to the fairy-tale state, where every action from the fairy-tale state leads back to the same state and gives max reward.

8.7 Limitations

Are that You get $n \times m \times n$ parameters to keep track of, plus You need to recalculate the value/policy thing every now and then.

8.8 Temporal-Difference Learning

Usually called TD learning.

It assumes a policy, and then updates values of based on a learning rate, α, so it's

$$V(x) = (1 - \alpha)V(x) + \alpha(r + V(x')) \tag{1}$$

Which is cool and all but how do We learn the policy that is generating this?

8.9 Q-function

Is weird.

So, the Q function gives You the value of taking an action at a particular state.

Now You might think "hey that sounds a lot like the reward function", but the difference here is that reward function just gives You the reward of going from x to x', where as $Q(x, a)$ takes into account the value of the next state, too.

$$Q(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a, \pi)V(x') \tag{2}$$

So each action gets a value. The optimal value for the state x is then just the max over a of $Q(x, a)$.

Now We basically just learn the Q function

$$Q(x, a) = (1 - \alpha)Q(x, a) + \alpha(r(x, a) + \gamma \max_{a'} Q(x', a')) \tag{3}$$

So We're just doing the 'ole temporal difference with Q-values, which are basically just candidate value functions.

Reinforcement Learning 2

Just a reminder that a policy induces a value function (depending on how You act, a state may be worth more or less) and a value function induces a policy (given how valuable You think states are, here is what You should do)

And the Bellman equation says that the optimal policy for the optimal value function is greedy, and the optimal policy is greedy only if the value function is optimal.

9.1 Value and Action-Value (Q) Functions

The value function is just

$$V^{\pi}(x) = r(x, \pi(x)) + \gamma \sum_{x'} P(x'|x, \pi(x)) V^{\pi}(x') \tag{1}$$

However, in the q function We relax that bit about about the policy:

$$Q(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^{\pi}(x') \tag{2}$$

So now We can look at actions the policy would not choose. Note that $V^{\pi}(x') = Q(x', \pi(x')$, so the Q-function and the value function are identical in the case of examining the action dictated by the policy. mkay.

Now switch gears and suppose You have the optimal policy.

Then

$$V^{*}(x) = \max_{a} \left[r(x, a) + \gamma \sum_{x'} P(x'|x, a) V^{*}(x') \right] \tag{3}$$

Okay so given optimal function value, the value of the state is just the maximal value of the action and weighted discounted sum of the consequences. Sure.

$$Q^{*}(x, a) = r(x, a) + \gamma \sum_{x'} P(x'|x, a) \cdot V^{*}(x') \tag{4}$$

And now observe that Q^{*} is just the max over actions. Cool.

9.2 TD-Learning as gradient descent

Basically can be shown that it is. Like You can come up with a sensible function such that it's derivative is that TD-error and there ya go, SGD.

9.3 Scaling the Value Function

The point of framing it as SGD though is that now You can just stick any old loss in there, or try to parametrize the thing.

So then You can take a step of SGD every time You transition in the model, but that turns out to be super slow. Instead an "experience replay" is used where I guess the previous episodes are maintained and iterated over a couple of times?

Freeze weights, use old weights across an entire batch.

9.4 Double DQN

Or, use old weights to establish value of states, use new weights to make decisions.

9.5 Policy Search

So there is a problem in that in order to do Q-learning, We need to search over the action space, which itself can become very tricky if the space is continuous or exponentially large.

So We parametrize the policy. Sure.

We also introduce **rollouts**, where as far as I can tell a rollout is one sort of attempt in the world.

A rollout $\tau_i = \{x_1, x_2 \dots x_T\}$ is a sequence of states, actions, and rewards. Sure

The reward for a rollout is a discounted sum.

And to find the optimal policy We do

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \frac{1}{m} \sum_{i=1}^{m} r(\tau_i) \tag{5}$$

And off it goes to find a gradient of this and reduce the variance. Not bothering with this until I need to.

Homework

10.1 Set 1

Q1

Let *pos* be the event of a positive test, let *neg* be the event of a negative test, let *ok* be the event of not being sick, and not *nok* be the event of being sick.

We're looking for $p(nok|pos)$

$$p(nok|pos) = \frac{p(pos|nok) * p(nok)}{p(pos)} \tag{1}$$

$$p(pos|nok) = 1 - p(neg|nok) = 1 - 0.99 = 0.01 \tag{2}$$

$$\begin{aligned} p(pos) &= p(pos|ok) * p(ok) + p(pos|nok) * p(nok) && (3)\\ &= (1 - p(neg|ok)) * 0.9999 + (1 - p(neg|nok)) * 0.0001 && (4)\\ &= 0.01 * 0.9999 + 0.99 * 0.0001 && (5)\\ &= 0.010098 && (6) \end{aligned}$$

$p(nok) = 1 - p(ok) = 0.0001$.

So in total We get

$$p(nok|pos) = \frac{p(pos|nok) * p(nok)}{p(pos)} = \frac{0.99 * 0.0001}{0.010098} = 0.00980392156 \tag{7}$$

10.2 Set 2

Q1

$$\begin{aligned} X^\top(Xw - y) &= 0 && (8)\\ X^\top Xw - X^\top y &= 0 && (9)\\ X^\top Xw &= X^\top y && (10)\\ w &= (X^\top X)^{-1} X^\top y && (11) \end{aligned}$$

Exam prep

11.1 Probability

So We have a set of atomic events Ω, and a set of non-atomic events which is a power set of Ω, and that's F. And We also have a function that takes elements of F and maps them to the interval $[0, 1]$, sure.

And the axioms We need are that summing probabilities of all things in Ω must sum to one and the probability of union of disjoint events must be the union of their individual probabilities.

Posterior can be thought of as normalized likelihood times prior.

Product rule just observes that $P(A, B) = P(A|B) * P(B)$, so no worries there.

Challenges - representation, as in how do You even write down these complicated distributions, and this is the parameterization problem.

Then learning - given You know like a family or a structure for this distribution, how do You actually learn the parameters. So this is the estimation problem.

And finally inference - say You have a representation and You have the parameters - extracting predictions might still be v. complicated.

11.2 Bayesian Learning

So We have formulas for the posterior mean and variance of the weight vector.

Ah okay and then the prediction is simply the dot product of the weight vector with the new data, plus some aleatoric uncertainty. Which gives us a great opportunity to see the split, since We can reduce the epistemic uncertainty with more datas.

Okay so then You do cross validation to find the optimal value for lambda, and then once You have that You estimate the empirical variance of labels, which having lambda and label variance You can approximate the prior variance.

11.2.1 Kalman Filters

Look who it is.

The main point here is that You can treat recursive Bayesian regression as a Kalman filter - the transition between states, where the state is the weight vector, is the identity matrix. Not even any noise on it, I don't think. Each measurement is state times data, observation is label, noise is variance of label.

11.3 Gaussian Processes

Aight so that's the linear case.

For nonlinearity, We use the Kernel trick.

You start a Gaussian prior on the weights, and at this point in time that's all ya got. So Your predictions have zero mean, since the mean of Your weights is zero, and the variance ya got is like $x\sigma_p x^\top$, where σ_p is the assumed prior variance of the weights. So standard stuff so far.

Ya got Your vector of predictions $f = Xw$.

So We're still in this wacky land of just using the prior, but We want a prediction on a now point $\tilde{x}, \tilde{y}$.

So, We take some existing X, y, and We slap a new data item on to both of them. Sure, no problem.

Now Your vector f has one more row in it with the new entry, which still has mean 0, and it now has a variance of $X\sigma_p X^\top$, the whole vector f does, and We add individual σ_n to each element's variance to account for individual noise to get actual predicted labels y.

Wait is the whole point of this just to arrive at the fact that everything depends on the dot product? Not shit sherlock lol.

Then how do select hyper parameters.

Well You do cross validation OR

Maximize the likelihood function lol.

Think of it this way - We've got the data X, parameters θ and the labels y. No worries there.

Now think about the probability of labels given the data and parameters - $p(y|X,\theta)$, no worries still.

So this is where a little imagination comes in - recall that Gaussian Processes give You a distribution over functions - this one's more likely, this one's less likely given Your data etc. So now, as You set θ, You get a whole range of functions, right?

So the idea is to pick a θ and integrate out all functions it can exist in, so

$$p(y|X,\theta) = \int_f p(y|f,X,\theta)p(f|\theta)df \tag{1}$$

$$= \int_f p(y|f,X)p(f|\theta)df \tag{2}$$

With the simplification in the second line due to the fact that given the function We no longer really care about the parameters that generated it.

So, now think about what happens - if θ allows only the simplest functions, then the probability of those functions will be high (just by the fact that the range of possible values is small, so each one gets a larger probability - like tossing D6 vs a D20), but the data might be a poor fit, so this is underfitting.

In the other case, θ might allow for complicated functions, but each function receives a smaller share of the probability mass, so if Your θ is needlessly loose, You'll still get a shit values all over the place due to these useless complicated functions You can generate.

and of course by optimizing this quality You'll end up in a happy middle.

11.4 Variational Inference

11.4.1 Laplace approximation

Finds the mode, then fits the curve, so curvy distributions are poorly approximated.

11.4.2 KL Divergence

Okay so in $KL(a,b)$, the intuition is like this - think about the shape of a. If there are places where a puts probability but b does not, the score is bad. So b

has to encapsulate a.

11.5 Markov Chain Monte Carlo

Okay so the idea is what We have some P We want to sample from, and instead We can only sample from the unnormalized version of P, call it Q.

The idea is then to design a markov chain that samples from Q such that with enough samples it's pretty close to sampling from P, is what I'm getting here.

11.5.1 Ergodicity

Exists a t such that any state is reachable within t steps precisely.

Detailed balance - probability of x' given x is the same as probability of x given You're at x', sure.

Gibbs sampling samples from one variable at a time keeping the others fixed, which allows to actually just straight sample from the distribution of the current vector and normalize that distribution.

How to formulate proposal distribution R? With simple Gaussian it just cancels out of the equations.

11.6 Deep Bayesian Learning

A network can be configured to put out both a mean prediction an a variance, and the loss can then take both into account.

The network can then minimize loss by either making correct confident predictions, or saying I'm very unsure about this maybe bad prediction. This doesn't give the network an easy out either, since even though the loss can be mitigated by claiming uncertainty, it will only be minimized by actually making semi confident correct predictions.

Cool fact - putting a Gaussian prior on the weights is equivalent to using weight decay.

Can approximate stuff by basically drawing neural network samples and averaging.

Another way is to keep a running average of the predictions during training - after some initial burn in period. If You train using stochastic gradients You can treat interim sets of weights as samples from the weight distribution.

Drop neurons during training with probability p. Can also use during prediction to generate a number of predictions to average them.

Bayes by backprop - maximizing ELBO.

Dropout - instead of Gaussian prior it's Dirac pulses at zero and the actual value. During prediction do different drop-outs to sample from the distribution.

Normally in dropout one would simply scale all the weights by 1-p, which is kind of averaging.

11.7 Active Learning

11.7.1 Mutual Information

Mutual info of $I(X, Y)$: $H(X) - H(X, Y)$

Ways to pick best sets - greedy - just pick the point maximizing information gain. Just look for points with largest variance. This works under constant aleatoric uncertainty.

Roughly uniform samlping if fitting a gaussian kernel with an isotropic kernel - so if only distance between observations matters, then uniform.

Mutual information is monotonic - adding more data can only improve things.

Greedy choice gives provably good results.

Better approach - maximize epistemic/aleatoric uncertainty.

So for Gaussian processes and noise that is constant, greedy is optimal (since it basically results in uniform).

11.7.2 Bayesian Optimization

Assume that the underlying thing You're trying to learn, f, to be somewhat smooth in order to be able to learn.

Regret - difference between optimal value of f and current value of f, and You want regret to go to zero over time.

Optimistic optimization - look at highest lowest bound, and then explore the most promising venue - this is maximizing the upper confidence bound.

Depending on how much structure is in the function, gamma t flattens out quicker - gamma t captures how much information can be gained I think, and basically the more regular the function the less info. Think of a constant.

There's a prior with which We model the underlying function, so You can fuck that up. Overestimating complexity will give eventually get You a solution though.

Thompson - sample from underlying process and optimize what You've sampled. So basically Your observed data constrains the space of possible functions, so You draw from the space, pick a point according to that function (i.e. to max the value), then update the distribution and start all over again.

11.8 Markov Decision Processes

Value function decomposes recursively.

Writing out linear algebra allows for the solution to a linear problem.

Linear problem solvable so long as discount factor is less than 1. Transition matrix of a markov transition matrix has eigenvalues of less than 1.

Or just use fixed point iteration

A value function is optimal if the policy that is optimal under that value function is greedy, and the greedy policy is optimal if the value function is optimal.

Policy iteration - pick a random value function, then calculate the greedy policy w.r.t. that policy, then recalculate the value function under that new greedy policy, then recalculate the value function under the new policy and so forth

Value iteration

11.9 Reinforcement Learning

Data not i.i.d.!

Model based - learn the MDP, optimize policy based on MDP.

Model-free - fuck the MDP, We just need a value function to add greedily.

On-policy - agent has to have control over actions to learn.

Off-policy, just observations by some behavioural policy, which may not work for Your learning algorithm. Basically given observations by some other actor can We learn stuff about the environment.

Epsilon greedy - with probability epsilon act not greedily.

R-max - set all unknown rewards to maximal positive value. If You don't know where an action leads assume it leads to a fairy tale state. Fairy tale state - all actions in fairy tale yield max reward and always lead back to the fairy tale.

Due to markov structure, observations of rewards are i.i.d. so You can get pretty tight bounds.

Update r-max once You're confident enough that You've really learned a new part of the MDP i.e. a reward + transition probability.

Problems with model-based stuff - You'll need to keep track of the MDP which means memory costs, and every time You update Your MDP, You'll need to solve for policies and value functions, which is not fun.

Anyway given a policy and I suppose a random initialization of the state values, You can use temporal difference (TD) learning to figure out what the state values are, as induced by the policy. Just take a convex combination of 'em, with a learning rate that satisfies some constraints.

Okay so Q-learning. You use TD-learning to estimate the values of actions. Just follow the formula really.

There's a weird initializiation for optimistic Q learning lol.

Q-learning requirements - gotta keep track of the Q table, so state times number of actions.

Reward scaling does not change the optimal policy - cannot do shifting though. There's basically a chasm where a reward flips from negative to positive, and if You have a positive loop then the agent will try to exploit it.

For Q learning to converge You need the conditions on alpha and the fact that all states are visited infinitely often.

DQN - Q learning but now instead of just having individual values for states You parametrize the values.

To speed things up training is usually done in batches, so I suppose You collect a bunch of transitions and update then.

Double DQN - use new weights to calculate Q-values in order to pick the maximum value, but only use that max to determine the action, and use the old weights to actually get the value.

But so parametrizing Q learning allows to deal with large state spaces.

Policy search - parametrize the policy to deal with massive state spaces, and obtain approximate value of state by just rolling it out a whole lot of times and averaging.

Baselines can be subtracted to reduce variance. The baseline can depend on stuff up to current state, so basically exploiting markov-y things.

Advantage function of an action and state - difference between Q-value of action and value of state under some policy.

Gradient function trick should be written down.

A2C subtracts the value estimate. Also they are on policy which sux.

Receding Horizon - take action, plan over the next H steps, take another step and replan.

Since We apply the same transition function over and over (write down the equations for this), vanishing/exploding gradients a problem.

Random shooting - just generate a bunch of trajectories and pick the best one.

Sparse reward - use Your own value function to try to estimate how close You are to the reward/how good the states are.

So optimal value function will just result in a greedy policy, but won't get optimal value function, so instead just plan over more than one action to get some balance going.

In stochastic environment You take the expectation over trajectories.

There's a problem in that You need to solve a maximization problem at every timestep.

You can learn the transition as a supervised regression problem but what'll happen is that Your agent will just come up with adversarial examples lol.

Avoid this by actually accounting for uncertainty so use GPs for example.

What You'll end up doing is sampling from the distribution that represents where You are now, taking a random step from that random place, then again sampling a random spot and taking a step etc.

Model based stuff is more efficient w.r.t. data.

But so how do exploration.

Can do epsilon greedy trivially.

Can do Thompson sampling in that You just generate some policy from Your probabilistic model of the policy (i.e. the model captures notions of epistemic and aleatoric uncertainty) and just pick actions that maximize the sampled function.

Another thing to do is a shooting-y thing? Keep track of possible policies a.k.a. Gaussian stuff again and just take actions which the most optimistic version of events represents.

H-UCRL - You have a model for transitions and a second model that attempts to quantize luck lol.

So basically there is some model that give or take keeps track of process dynamics, and a second model that keeps track of the variance in the first model, and You use the second model to create optimistic plans.