M2 MPRI - Programmation Probabiliste - Projet

Guillaume Baudart Christine Tasson

Date limite : 18 février 2022

1 Objectifs

Le projet consiste à implémenter votre propre langage de programmation probabiliste. Le projet devra en particulier contenir :

- une petite bibliothèque de distributions,
- une méthode d'inférence exacte par énumération pour les modèles discrets,
- une méthode d'inférence de Metropolis-Hasting,
- un ensemble de tests et un modèle original,
- un rapport qui documente votre projet.

Vous pouvez réaliser le projet au choix en **Python** ou **OCaml**. Votre langage pourra être embarqué (sous la forme d'une librairie) ou non (définition de votre propre syntaxe). Si vous avez le temps, n'hésitez pas à enrichir votre language (autres méthodes d'inférence, typage, analyse statique, ...).

2 Inférence discrète par énumération

Pour de (petits) modèles discrets, il est possible d'explorer systématiquement toutes les executions possibles du modèle.

Par exemple, le programme `funny_bernoulli` suivant a 8 exécutions possibles selon les valeurs choisies pour `a`, `b` et `c` (Figure 1). Pour chaque execution on peut calculer le score, ($-\infty$ pour les deux exécutions où `a = b = 0`, et 0 pour les autres).

```
let funny_bernoulli () =
  let a = sample (bernoulli ~p:0.5) in
  let b = sample (bernoulli ~p:0.5) in
  let c = sample (bernoulli ~p:0.5) in
  let () = assume (a = 1 || b = 1) in
  (a + b + c)

let dist = infer funny_bernoulli ()
```

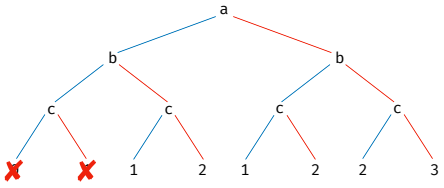

FIGURE 1 – Exécutions de `funny_bernoulli`.

Pour obtenir la distribution recherchée, il suffit de normaliser les couples(résultat, score) obtenus à la fin de chaque exécution.

Note. Si le modèle est écrit en *Continuation Passing Style* (CPS), il est possible de reprendre l'exécution à un point de contrôle (par exemple une instruction `sample`) pour explorer une nouvelle exécution.

3 Inférence avec Metropolis-Hasting.

L'algorithme de Metropolis-Hasting construit une chaîne de Markov qui converge vers la distribution à posteriori recherchée. L'algorithme peut être grossièrement décrit de la manière suivante (les détails seront vus en cours). On commence par exécuter le modèle une première fois pour obtenir une exécution possible. Puis à chaque itération :

1. On choisit aléatoirement une des variables échantillonnées (instruction `sample`).
2. On tire une nouvelle valeur pour cette variable (et toutes celles qui en dépendent).
3. On compare le score de cette nouvelle trajectoire à celui de la précédente.
4. Selon le résultat, on accepte la nouvelle trajectoire ou on garde la précédente avec une certaine probabilité qui dépend du score des deux trajectoires.
5. On renvoie l'échantillon correspondant à la trajectoire choisie.

Après une phase d'initialisation, la chaîne de Markov converge vers la distribution a posteriori, et les échantillons sont produits suivant cette distributions.

Cette méthode d'inférence donne parfois de très bons résultats, notamment pour les modèles multi-dimensionnels qui font intervenir beaucoup de variables aléatoires.

4 Exemples et tests

Vous pouvez commencer par ré-implémenter les exemples vu en cours dans votre langage.

— `funny_bernoulli`
— Problème des naissance de Laplace
— Biais d'une pièce
— Modèle de Markov caché (HMM)
— Régression linéaire
— ...

Il peut être utile d'implémenter également une méthode d'inférence simple (par exemple, *importance sampling*) pour vérifier les résultats obtenus.

Utilisez ensuite votre langage pour résoudre un problème original de votre choix.

Suggestion : balle rebondissante. Voici une idée de problème. On lâche une balle à une altitude h_0 sans vitesse initiale. La balle peut rebondir sur deux plateformes pour atteindre un objectif (un sceau posé par terre) en $x \in [x_s - w, x_s + w]$. Proposer un modèle qui permet de trouver la position et l'angle des deux plateformes pour que la balle atteigne l'objectif.