AS - TP Modèles Stochastiques

Ludovic Denoyer et Nicolas Baskiotis

1 Architecture Stochastique simple

1.1 Modélisation

Nous allons considérer un modèle prédictif stocahstique : dans un tel modèle, la prédiction en fonction de l'entrée $f_\theta(x) \rightarrow \hat{y}$ est un tirage aléatoire de $\hat{y}$ selon une loi de probabilité discrète $P_\theta(\hat{y}|x)$. Soit la fonction de coût $\Delta(\hat{y}, y)$:

1. Ecrivez le problème d'apprentissage associé à ce modèle
2. Sachant que $\nabla(\log g) = \frac{\nabla g}{g}$, écrivez le gradient de l'objectif en fonction de $\log P_\theta(\hat{y}|x)$.
3. Comment ce gradient peut-il être estimé ?

1.2 Implémentation

Nous allons considérer le modèle constitué d'une couche linéaire suivie d'une couche de type logsoftmax. Implémentez l'algorithme d'apprentissage correspondant en Torch. Testez votre algorithme sur un ensemble de deux gaussiennes simples. Lisez la section 4 en particulier de `https://arxiv.org/pdf/1506.05254v3.pdf` : au lieu d'utiliser le loss comme feedback, on utilise le loss moins la moyenne du loss sur le batch (pour avoir un gradient qui change de sens selon que la prediction est moins bonne ou meilleure que la moyenne).

2 Architectures Stochastiques générales

2.1 Modélisation

Soit un espace d'entrée $\mathcal{X}$ et un espace de sortie $\mathcal{Y}$. Nous allons nous intéresser à des fonctions de prédiction stochastiques telles que :

$$\hat{y} = f_\theta(z, x) \tag{1}$$

où z est une variable samplée selon la probabilité $P_\theta(z|x)$, θ étant l'ensemble des paramètres du modèle. Soit Δ la fonction de loss associée, le critère de qualité du modèle θ peut donc s'écrire :

$$J(\theta) = \mathbb{E}_{\theta,z}[\Delta(f_\theta(z, x), y)] \tag{2}$$

Calculez le gradient de J
Pour cela :

1. Ecrivez l'espérance sous forme d'une intégrale
2. Sachant que $\nabla(\log g) = \frac{\nabla g}{g}$, exprimez le gradient en fonction de $\nabla_\theta \log P_\theta(z|x)$

2.2 Implémentation

L'implémentation d'un tel gradient peut être réalisé à l'aide de la bibliothèque *dpnn* suivant le schéma : *forward*, *reinforce*, *backward* et des modules *ReinforceCategorical* ou bien *ReinforceBernoulli* (voir documentation).

Vous pouvez également vous reportez à `http://incompleteideas.net/sutton/williams-92.pdf` et `https://arxiv.org/pdf/1410.0510v1.pdf`.

Nous allons nous intéresser au modèle suivant $f : x \to z \to \hat{y}$ où z est un vecteur binaire, chaque élément étant tiré selon une loi de bernoulli. C'est un modèle de discrétisation de l'espace d'entrée.

— Implémentez ce modèle sur des jeux de données en 2D
— Visualisez la discrétisation obtenue