Rapport Méthodes et Algorithmes

Global Illumination with Radiance Regression Functions

Yannick BERNARD
Pierre GUERINEAU
Kevin MENIEL
Romain MOUTRILLE
Matthias ROVES

24 novembre 2016

Table des matières

1 Introduction

Intérêt du réseau de neurones

L'utilisation d'un réseau de neurones est dûe au fait qu'ils sont compacts et leur vitesse d'évaluation est élevée, ce qui représente des facteurs critiques pour le rendu temps réel. Les réseaux neuronaux sont également bien adaptés à notre problème, car ils fournissent un modèle non linéaire simple mais puissant qui a été éprouvé pour rivaliser avec les meilleures méthodes sur les problèmes de régression. Un réseau neuronal peut être considéré comme une généralisation non linéaire du modèle de régression linéaire. De plus, les réseaux de neurones ont un fort pouvoir de représentation et ont été largement utilisés comme approximateurs de fonctions universelles. Plus précisément, des fonctions continues ou des fonctions de carrés sommables sur des domaines finis peuvent être approchées avec une petite erreur par un réseau de neurones de type *"feedforward"* avec une ou deux couches cachée[8][2]. La fonction d'éclairage indirect s^+ appartient à cette catégorie de fonctions.

2 Extraction

2.1 Extraction des données d'apprentissage

L'extraction des données de luminance indirecte de la scène est la première étape pour la mise en place du système. En effet, pour pouvoir affiner sa fonction de régression, le réseau de neurone à besoin d'appliquer sa routine d'apprentissage sur des données de test qui serviront de but à atteindre.
Dans notre cas, ces données seront les positions des caméras, lumières, ainsi que la position, la normale, des information sur les matériaux et la luminance indirecte d'un ensemble de points de la scène. Le but de l'extraction est donc d'obtenir des échantillons de données couvrant au mieux les degrés de liberté du réseau, ainsi que les résultats attendus pour pouvoir calculer l'erreur.

L'extraction des données va ainsi se passer en deux phases :

1. **La définition des compositions de lumière et de caméra des scènes de test.**

   Pour la composition des scènes d'apprentissage, nous allons échantillonner l'espace des possibles afin d'obtenir une couverture optimale. Afin d'optimiser le plus possible le nombre de scène de test, il nous faut donc trouver les positions de caméras et de lumières qui maximisent le recouvrement de la scène.
   Une méthode échantillonnage permettant cela est **l'Hypercube Latin**. Ce dernier va ainsi nous permettre de couvrir au maximum l'espace des possibles, tout en minimisant les recouvrements et les redondances de données.

2. **L'échantillonnage de la luminance de la scène.**

   Pour l'échantillonnage de la luminance de chaque scène, nous allons échantillonner les directions autour de la caméra afin de couvrir le plus de points visibles possible. Il nous faut donc échantillonner la surface d'une sphère centrée en en la camera.
   Afin de calculer la luminance indirecte, nous allons nous servir de la structure déjà fournie par PBRT. Nous allons donc créer une nouvelle caméra de type sphérique et un intégrateur qui ne retourne que la luminance indirecte de la scène.

2.2 Latin Hypercube Sampling

L'échantillonnage par hypercube latin est une technique permettant d'obtenir un résultat distribué sur l'ensemble de la zone d'échantillonnage. Cette technique se base sur la composition de tirages aléatoires effectués dans des régions à probabilité uniforme de l'espace de définition[10].

Méthode :

D	=	le nombre de dimensions de l'échantillon.
N	=	le nombre d'échantillons voulus.
$[x_0, x_1, \ldots, x_{D-1}]$	=	coordonnées du point minimal de l'espace.
$[X_0, X_1, \ldots, X_{D-1}]$	=	coordonnées du point maximal de l'espace.

On définit l'ensemble d'intervalles :

$$Q^i = \{[p_0^i, p_1^i], \ldots, [p_{N-2}^i, p_{N-1}^i]\} \text{ avec :}$$

$$\begin{cases} p_0^i & = x_i & \forall i \in [0, D[\\ p_{N-1}^i & = X_i & \forall i \in [0, D[\\ p_k - p_{k-1} & \text{constant} & \text{pour } k \in]0, N[\end{cases}$$

On crée ainsi les vecteurs A_i de taille N, définis comme l'ensemble des tirages aléatoires d'une valeur dans les intervalles uniformes Q^i.

$$A^i = \{rand(Q_0^i), rand(Q_1^i), \ldots, rand(Q_{N-1}^i)\}$$

Pour $rand(a, b)$ un tirage aléatoire entre a et b.
Afin d'obtenir les échantillons, on compose ces derniers de manière aléatoire à partir des valeurs des Ai afin d'obtenir des échantillons de dimension D.

2.3 Sphere point picking

Lors de la sélection de points sur la surface d'une sphère unitaire, on ne peut pas se contenter de prendre des valeurs polaire aléatoire. En effet les éléments surfacique étant définis par l'angle polaire, cela entraînerait une accumulation des échantillons au niveau des pôles.

$$\mathrm{d}\Omega = sin\varphi \mathrm{d}\varphi \mathrm{d}\theta$$

Afin d'éviter ce problème, plusieurs autres méthodes sont possibles pour échantillonner la sphère :

— **Méthode de Marsaglia (1972)[9] :**

Cette méthode se base sur deux tirages aléatoires x_1 et x_2 dans l'espace $[-1, 1]$ tels que $x_1^2 + x_2^2 < 1$

$$\begin{cases} x & = & 2x_1\sqrt{1 - x_1^2 - x_2^2} \\ y & = & 2x_2\sqrt{1 - x_1^2 - x_2^2} \\ z & = & 1 - 2(x_1^2 + x_2^2) \end{cases}$$

— **Méthode de Cook (1957)[3] :**

Cette méthode est la technique de Von Neumann. Celle-ci nécessite quatre valeurs aléatoires dans l'espace $[-1, 1]$ telles que $x_0^2 + x_1^2 + x_2^2 + x_3^2 < 1$. La règle des quaternions permet ainsi de définir :

$$\begin{cases} x & = & \frac{2(x_2x_3 + x_0x_1)}{x_0^2 + x_1^2 + x_2^2 + x_3^2} \\ y & = & \frac{2(x_1x_3 + x_2x_0)}{x_0^2 + x_1^2 + x_2^2 + x_3^2} \\ z & = & \frac{x_0^2 - x_1^2 - x_2^2 + x_3^2}{x_0^2 + x_1^2 + x_2^2 + x_3^2} \end{cases}$$

— **Méthode de Muller (1959)[11] :**

Pour mettre en place cette méthode, il nous faut trois valeurs aléatoires gaussienne x, y et z. On pose alors ces valeurs sur la sphère unitaire grâce à la formule :

$$\frac{1}{\sqrt{x^2+y^2+z^2}} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

3 Apprentissage

3.1 Données en entrée du réseau de neurones

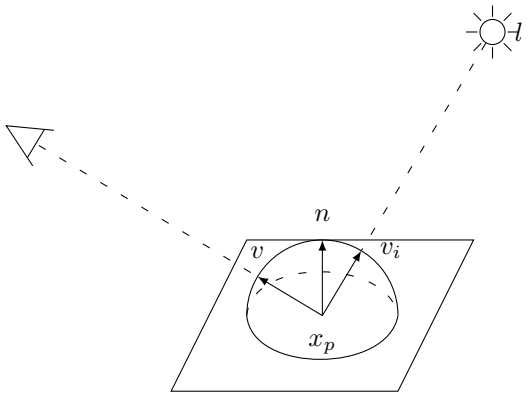

FIGURE 1 – Schéma du rendu

— **Attributs basiques :**

x_p : position du point sur la surface
v : vecteur vue (observateur)
l : position de la source

— **Attributs augmentés :**

n : normale à la surface à la position x_p
a : vecteur des paramètres nécessaires à la BRDF

Possibilité de les avoir dans le pipeline graphique, n est calculé dans l'éclairage direct et les éléments de a sont des textures.

La régression pourrait être effectuée uniquement avec les attributs de base, l'ajout de n et a apporte une meilleure approximation de l'illumination indirecte par une fonction de régression non-linéaire. Les deux données n et a n'auront plus à être calculées à partir des données d'entraînement.

3.2 Construction

De part la fonction de rendu de base, ils séparent les composantes directes et indirectes :

$$s^0(x_p, v, l) + s^+(x_p, v, l)$$

où s^0 = composante directe (choix : Cook-Torrance et Burley (Disney))
et s^+ = composante indirecte.
Le réseau de neurones calcule la composante indirecte comme suit :

$$s^+(x_p, v, l) = \int_{\Omega^+} \varrho_c(v, v_i, a(x_p))(n(x_p).v_i)s_i^+(x_p, v_i)\mathrm{d}v_i$$

La fonction de régression est définie comme suit :

$$\varphi = (x_p, v, l, n, a)$$

Ce qui représente une transformation de $\mathbf{R}^{12+n_p} \rightarrow \mathbf{R}^3$ où n_p est le nombre de paramètres de la BRDF et $\mathbf{R}^3$ les canaux RVB.

3.2.1 Données d'entraînement

N paires $(entre - sortie)$ échantillonnant l'illumination indirecte. Chaque paire est appelée un "exemple" et est de la forme :

$$(x^i, y^i) \text{ où } \begin{cases} x^i &= [x_p^i, v^i, l^i, n^i, a^i] \\ y^i &= s^+(x_p^i, v^i, l^i) \end{cases} pour \mathrm{i} \in [1, N].$$

3.2.2 Choix du nombre de couches

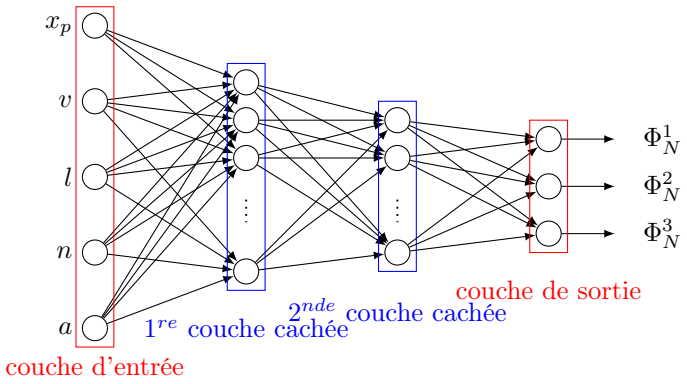

FIGURE 2 – Schéma de la structure du réseau de neurones

Le réseau est composé d'une couche d'entrée, où seront envoyés les paramètres en entrée, une couche de sortie qui renvoie les valeurs RVB, et de 2 couches "cachées". 2 couches suffisent pour évaluer les fonctions continues ou les fonctions de carrés sommables sur un domaine fini avec une erreur minimale[8][2].

3.2.3 Choix du nombre de neurones par couche

— 20 pour la 1^{re} couche cachée

— 10 pour la 2^{nde} couche cachée

C'est un choix équilibré pour 2 couches intermédiaires d'un point de vue capacité d'approximation et efficacité de calcul. Augmenter le nombre de noeuds diminue l'effet "fitting" mais augmente aussi le nombre de poids de façon considérable et alourdit l'évaluation de chaque donnée considérablement.

3.2.4 Passer d'un noeud à l'autre

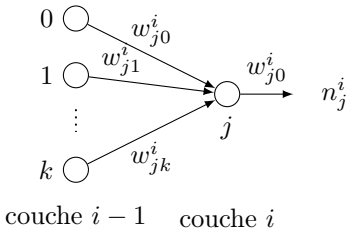

FIGURE 3 – Schéma des entrées/sorties d'un noeud

Considérons un noeud j situé à la i^{eme} couche, ayant pour sortie n^i_j et pour poids de biais w^i_{j0}.
Le noeud j reçoit les sorties n^{i-1}_k et les poids w^i_{jk} de tous les noeuds situés à la couche précédente, $i-1$. La sortie est de la forme suivante :

— **Couche cachée** $n^i_j = \sigma(z^i_j), s^i_j = w^i_{j0} + \sum_{k>0} w^i_{jk} n^{i-1}_k$

où $\sigma(z) = tanh(z) = \frac{2}{(1+e^{-2z})}$ est la tangente hyperbolique, qui est une fonction continue et différentiable, ce qui permet de faciliter les calculs numériques.

— **Couche de sortie** $n^i_j = w^i_{j0} + \sum_{k>0} w^i_{jk} n^{i-1}_k$

3.2.5 La RRF finale(analytique)

$\Phi_N = [\Phi^1_N, \Phi^2_N, \Phi^3_N]$ où chaque Φ^i_N correspond aux canaux R, V, B. On obtient :

$$\Phi^i_N(x,w) = w^3_{i0} + \sum_{j>0} w^3_{ij}\sigma(w^2_{ij}\sum_{k>0} w^2_{jk}\sigma(w^1_{k0} + \sum_{l=1}^{9} w^1_{kl}x_l))$$

où $i \in \{1,2,3\}$ et σ est la fonction de tangente hyperbolique (seul élément non linéaire de Φ_N). Il est possible de rendre Φ_N linéaire en remplaçant la fonction de tangente hyperbolique par une fonction linéaire.

Minimisation de la fonction de régression au sens des moindres carrés :

$$E(w) = \sum_i ||y^i - \Phi_N(x^i_p, v^i, l^i, n^i, a^i, w)||^2$$

3.2.6 Algorithme pour l'apprentissage et initialisation

L'algorithme utilisé pour l'apprentissage et le calcul des w est celui de Levenberg-Marquardt pour minimiser $E(w)$. Il a montré de meilleures performances que des calculs de gradients ou des méthodes de conjugués[5].

Initialisation :
Pre-processing (avant la 1^{ere} couche cachée) des données en entrée et un post-processing des données en sortie tel que décrit dans l'implémentation de Beale et al[1] afin de normaliser les données dans l'intervalle $[-1.0, 1.0]$ (fonction `mapminmax` de l'outil de réseau de neurones de Matlab[1]). Le facteur d'échelle est sauvegardé pour les calculs en temps réel. On initialise le vecteur w avec des valeurs aléatoires comprises dans l'intervalle $[-1.0, 1.0]$.

La mise à jour des poids est faite itérativement comme suit :

$$w^{n+1} = w^n + (H + \lambda diag[H])^{-1} g$$

où w^{n+1} et w^n sont les vecteurs de poids dans l'itération courante et l'itération précédente. H est la matrice Hessian de $E(w)$ (respectant les w). Elle peut être calculée depuis la matrice Jacobienne J de $E(w)$ telle que $H = J^T J$.
λ est le facteur d'amortissement, il est calculé comme suit :
D'après Marquardt, on pose λ_0 et un facteur $v > 1$, on a donc $\lambda = \lambda_0$. On calcule la variance à chaque itération en utilisant le facteur d'amortissement puis en utilisant $\frac{\lambda}{v}$. Si les deux derniers renvoient un point moins bon que le point de départ, on augmente λ en le multipliant par v jusqu'à atteindre un meilleur point avec un nouveau facteur λv^k pour un certain k. Si l'utilisation du facteur $\frac{\lambda}{v}$ donne une somme plus faible, alors il est pris comme nouvelle valeur de λ et l'algorithme L-M continue.
Si l'utilisation de $\frac{\lambda}{v}$ donne une somme plus importante, mais que l'utilisation de λ donne une variance plus faible, alors λ est conservé (*Wikipédia*).
g définit le gradient de l'erreur actuelle de $E(w^n)$, est est donné par : $g = J^T E(w^n)$. A chaque itération, on calcule la matrice Jacobienne de $E(w^n)$ par une rétropropagation du gradient[7].
On réitère le processus jusqu'à obtenir un $E(w) < 0.05$ (défini par l'utilisateur, 0.05 dans l'article).

3.3 Différences entre apprentissage local et online

— **Local :**
On minimise $E(w)$ avec toutes les données sur une même machine.

— **Online :**
On divise les données en sous-groupes et on met à jour le vecteur de poids w en minimisant $E(w)$ avec un sous-groupe.

Conclusion : L'apprentissage local surpasse toujours l'apprentissage online avec l'implémentation décrite.

3.4 Résolution du problème de surapprentissage

3 méthodes :

1. Avoir suffisament de données d'apprentissage, environ 8-10 fois le nombre de poids[4].
2. Cross-validation, utiliser 70% des données pour l'apprentissage et 30% des données pour la validation des résultats de l'apprentissage[1].
3. Dégradation des pondérations (une forme de régularisation)[6].

4 Rendu

Une fois l'entraînement du réseau de neurones réalisé, on peut rendre la composante indirecte pour chaque point de la scène sous différentes conditions d'éclairage et de position de la caméra en lui envoyant un vecteur d'entrée de paramètres les caractérisant.

1. On effectue le calcul des points visibles de la surface pour chaque pixel de l'écran, ainsi que leur position x_p sur la surface. Chaque attribut dans le vecteur d'entrée est normalisé à $[-1.0, 1.0]$.

2. Pour chaque pixel, le vecteur d'entrée normalisé (x_p, v, l, n, a) est l'entrée du réseau de neurone Φ_N, dont on aura en sortie 3 composantes R, V, B correspondant au composantes couleurs de l'éclairage indirect.
   x_p : position du point sur la surface dont on calcule l'éclairage indirect.
   v : direction de la caméra depuis le point x_p de la surface.
   l : position de la source lumineuse.
   n : normale à la surface en x_p.
   a : paramètres de la BRDF.

3. Le résultat de l'éclairage indirect est ajouté aux résultats de l'éclairage direct calculé au point x_p pour les mêmes paramètres que ceux envoyés au réseau de neurones.
   Pour l'éclairage direct, on utilisera le modèle de Cook-Torrance pour le calcul de la composante spéculaire de l'équation de rendu, ainsi que le modèle de Burley pour le calcul de la composante diffuse.

4. Évaluer Φ_N en temps réel est possible car $a(x_p)$ et $n(x_p)$ sont disponibles dans le pipeline graphique :
   — $a(x_p)$: vecteur de paramètres de la BRDF sous forme de texture.
   — $n(x_p)$: normale en x_p, calculée en évaluant l'éclairage direct.

4.1 Gestion de la complexité des scènes

Afin de diminuer le coût de l'évaluation, on décompose la scène tout en profitant du fait que les différents objets 3D sont déjà topologiquement déconnectés les uns des autres.
En effet, pour chaque objet, on décompose l'espace parcouru par les vecteurs en entrée du réseau de neurones en plusieurs régions à l'aide d'un kd-tree.
On peut ainsi assigner à chaque région une fonction de régression séparée.
L'avantage de cette technique est que, bien qu'elles soient plus nombreuses, les fonctions de régression sont plus petites, et donc dans un contexte de rendu temps réel plus rapides à évaluer.
Le kd-tree a préalablement été créé de la manière suivante :
On crée une boîte englobante autour de chaque objet, on coupe la boîte du niveau actuel en son milieu, en deux boîtes plus petites de niveau inférieur. On réitère l'opération sur chacune des deux boîtes du niveau inférieur un certain nombre fois (dans notre cas on arrête lorsque les erreurs relatives de prédiction et d'entraînement sont inférieures à 5%).
Les dernières boîtes obtenues sont des feuilles du kd-tree dont les fonctions de régression sont calculées pour leur région.
Le calcul de l'éclairage indirect consiste alors à faire une recherche dans l'arbre de la feuille correspondant à la fonction de régression à évaluer en le parcourant et choisissant le noeud correspondant de l'espace, jusqu'à arriver à la feuille en question.

4.2 Complexité de l'éclairage

Le calcul de l'éclairage indirect est également compatible pour une scène disposant de plusieurs sources lumineuses, notées K, avec la k^{ieme} source située à la position l_k. En effet Φ_N étant calculé pour une seule source lumineuse et le transport de la lumière étant linéaire et respectant les intensités lumineuses, nous pouvons écrire :

$$s^+(x_p, v, l_1, \ldots, l_K) = \sum_{k=1}^{K} c_k \Phi_N(x_p, v, l_k, n, a, w)$$

Avec c_k la couleur de la k^{ieme} source.

Étant donné que chaque terme de la somme est calculé par la même RRF Φ_N, aucun apprentissage ou coût de stockage supplémentaire est nécessaire. Le seul coût supplémentaire sera le calcul de Φ_N pour chaque source lumineuse ainsi que le calcul de l'éclairage direct pour chaque source.

Références

[1] HAGAN M. T. BEALE, M. H. and H. B. DEMUTH. Neural network toolbox user's guide. 2012.

[2] Edward K. Blum and Leong Kwan Li. Approximation theory and feedforward networks. *Neural Netw.*, 4(4) :511–515, July 1991.

[3] J. M. Cook. Technical notes and short papers : Rational formulae for the production of a spherically symmetric probability distribution. *Math. Tables Aids Comput.*, 11 :81–82, 1957.

[4] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. NeuroAnimator : Fast neural network emulation and control of physics-based models. 32(Annual Conference Series) :9–20, August 1998.

[5] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6) :989–993, Nov 1994.

[6] Trevor J. Hastie, Robert John Tibshirani, and Jerome H. Friedman. *The elements of statistical learning : data mining, inference, and prediction.* Springer series in statistics. Springer, New York, 2009. Autres impressions : 2011 (corr.), 2013 (7e corr.).

[7] G. E. Hinton. Connectionist learning procedures. In J. Carbonell, editor, *Machine Learning : Paradigms and Methods*, pages 185–234. MIT Press, London, 1989.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5) :359–366, January 1989.

[9] George Marsaglia. Choosing a point from the surface of a sphere. *Ann. Math. Statist.*, 43(2) :645–646, 04 1972.

[10] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1) :55–61, February 2000.

[11] Mervin E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4) :19–20, April 1959.