FORMATION AUTOMATANTS
APPRENTISSAGE DES RÉSEAUX DE NEURONES
06/10/2020

Rappel du fonctionnement d'un réseau de neurones

Lors de la traversée d'une couche k d'un réseau de neurones (à plusieurs couches), on applique l'opération suivante sur Z_k :

$$Z_{k-1} \to W_k Z_{k-1} + B_k = A_k \to f_k(A_k) = Z_k$$

où f_k est une fonction d'activation, qui permet de rompre avec la linéarité des opérations et ainsi s'assurer que chaque couche du réseau de neurones correspond bien à une opération distincte des autres.

Principe de l'apprentissage

Quelques rappels à propos d'un problème simple à résoudre : la recherche du minimum (ou d'un minimum) de $f: \mathbb{R} \to \mathbb{R}$ continue (ou continue par morceaux). La méthode la plus simple pour y parvenir est la méthode de Newton (aussi appelée descente de gradient stochastique) : on calcule récursivement à partir d'un certain x_0, la suite $(x_n)_{n\in\mathbb{N}}$ définie par $x_{n+1} = x_n - \alpha f'(x_n)$ où $\alpha > 0$ est un hyperparamètre (paramètre que l'on ne peut pas faire évoluer au cours de l'apprentissage). Cette méthode permet effectivement de trouver le minimum de f s'il existe (voir exemple avec $f(x) = x^2$).

La notion d'apprentissage est très vaste et très complexe, donc pour la traiter informatiquement, on la réduit à un problème de recherche de minimum : il s'agit de minimiser l'écart entre le résultat obtenu à partir d'une entrée connue et le résultat que l'on attendait. Ce problème de recherche de minimum est lui-même réduit à un problème de calcul de dérivée, car assez simple à mettre en place et très efficace sur un ordinateur. Formalisons le problème : on dispose d'un ensemble d'entraînement constitué de couples $(X_{\text{train}}, Y_{\text{train}})$ connus, qui correspond à l'ensemble sur lequel le réseau $\mathcal{R}$ (que l'on représente par la fonction $F_{\mathcal{R}}$) va apprendre. En effet, à la fin de l'apprentissage, on aura pour chaque couple $(X_{\text{train}}, Y_{\text{train}})$, $Y_{\text{predict}} = F_{\mathcal{R}}(X_{\text{train}}) \approx Y_{\text{train}}$. L'apprentissage va consister en la modification des valeurs des matrices W_k (poids) et B_k (biais) afin de minimiser le coût (l'erreur faite par le réseau) entre Y_{predict} et Y_{train}. Il faut donc se munir d'une fonction permettant d'évaluer un coût entre 2 vecteurs. Il existe de nombreuses fonctions de ce type, que l'on note h :

$$L_2 : h(X,Y) = \frac{1}{2}\|X - Y\|_2^2$$

$$L_1 : h(X,Y) = \|X - Y\|_1$$

$$\textit{Cross-entropy} : h(X,Y) = -\big(Y * \log(X) + (1-Y) * \log(1-X)\big)$$

…

On cherche donc à minimiser la fonction : $h(F(X_{\text{train}}), Y_{\text{train}})$. On pose par la suite :

$$g\big(w_{0,11}, w_{0,12}, \dots, b_{N,n}, X, Y\big) = h\big(f\big(w_{0,11}, w_{0,12}, \dots, b_{N,n}, X\big), Y\big)$$

où on a : si $\mathcal{R}_{w_{0,11}, w_{0,12}, \dots, b_{N,n}}$ correspond au réseau de neurones (de profondeur $N+1$) ayant pour paramètres $w_{0,11}, w_{0,12}, \dots, b_{N,n}$ alors :

$$\forall X, F_{\mathcal{R}_{w_{0,11}, w_{0,12}, \dots, b_{N,n}}}(X) = f\big(w_{0,11}, w_{0,12}, \dots, b_{N,n}, X\big)$$

Le problème consiste donc à minimiser g par rapport aux paramètres $w_{0,11}, w_{0,12}, \dots, b_{N,n}$.

On utilise pour ce faire la descente de gradients. À chaque étape (*i.e.* pour chaque couple $(X_{\text{train}}, Y_{\text{train}})$ testé) on fait :

$$\forall \kappa \in w_{0,11}, w_{0,12}, \dots, b_{N,n}, \qquad \kappa \leftarrow \kappa - \alpha \left.\frac{\partial g}{\partial \kappa}\right|_{w_{0,11}, w_{0,12}, \dots, b_{N,n}}$$

Ce que l'on réécrit de la façon suivante pour inclure l'ensemble des paramètres du problème :

$$\forall k \in [\![1, N]\!], \begin{cases} W_k \leftarrow W_k - \alpha\, \Delta W_k \\ B_k \leftarrow B_k - \alpha\, \Delta B_k \end{cases} \text{ où } \Delta W_k = \begin{pmatrix} \frac{\partial g}{\partial w_{k,11}} & \cdots & \frac{\partial g}{\partial w_{k,1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g}{\partial w_{k,n1}} & \cdots & \frac{\partial g}{\partial w_{k,nn}} \end{pmatrix} \text{ et } \Delta B_k = \begin{pmatrix} \frac{\partial g}{\partial b_{k,1}} \\ \vdots \\ \frac{\partial g}{\partial b_{k,n}} \end{pmatrix}$$

On utilise la notion de différentielle pour ramener ce calcul de dérivées à de simples produits de matrices. Quelques rappels à propos de la différentielle :

– c'est une fonction linéaire que l'on peut définir de la façon suivante :

$$f(x + h) =_{h \to 0} f(x) + \mathrm{d}f(x).h$$

– pour $f : \mathbb{R} \to \mathbb{R}$ dérivable, on a : $\mathrm{d}f(x).h = f'(x)h$

– pour $f : \mathbb{R} \to \mathbb{R}^m$ dérivable, on écrit :

$$f = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} \text{ d'où } f'(x) = \begin{pmatrix} f_1'(x) \\ \vdots \\ f_m'(x) \end{pmatrix} \Rightarrow \mathrm{d}f(x).h = \begin{pmatrix} f_1'(x)h \\ \vdots \\ f_m'(x)h \end{pmatrix}$$

– pour $f : \mathbb{R}^n \to \mathbb{R}$, on a : $\mathrm{d}f(x) = \left(\left.\frac{\partial f}{\partial x_1}\right|_x, \dots, \left.\frac{\partial f}{\partial x_n}\right|_x \right)$ d'où :

$$\mathrm{d}f(x).h = \left.\frac{\partial f}{\partial x_1}\right|_x h_1 + \cdots + \left.\frac{\partial f}{\partial x_n}\right|_x h_n = \nabla f(x) \cdot h$$

– pour $f : \mathbb{R}^n \to \mathbb{R}^m$, on a :

$$f = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} \text{ d'où } \mathrm{d}f(x) = \begin{pmatrix} \nabla f_1(x) \\ \vdots \\ \nabla f_m(x) \end{pmatrix} \text{ et } \mathrm{d}f(x).h = \begin{pmatrix} \nabla f_1(x) \cdot h \\ \vdots \\ \nabla f_m(x) \cdot h \end{pmatrix} = J(f)(x) \cdot h$$

où $J(f)(x)$ est la matrice jacobienne de f en x. En général, on connaît les fonctions que l'on utilise donc on connaît les matrices jacobiennes (on peut aussi avoir recours à la dérivation numérique, qui est utile quand on ne connaît pas bien la fonction f, mais nécessite plus de RAM).

Il est aussi intéressant de rappeler les formules pour faire la différentielle d'une composée de fonctions différentiables :

$$\mathrm{d}(g \circ f)(x) = \mathrm{d}g\big(f(x)\big).\mathrm{d}f(x)$$

$$(g \circ f)'(x) = g'\big(f(x)\big) f'(x)$$

$$\left.\frac{\partial (g \circ f)}{\partial x}\right|_{x=a} = \sum_k \left.\frac{\partial g}{\partial z_k}\right|_{z=f(a)} \left.\frac{\partial f}{\partial x}\right|_{x=a}$$

$$J(h)(a) = J(g)\big(f(a)\big) \cdot J(f)(a)$$

On voit bien que l'on se ramène facilement à un problème de multiplications de matrices entre elles.

Backpropagation

On utilise le concept de feed-forward : on garde en mémoire les résultats à chaque étape (qui sont les entrées des couches d'après), pour les réutiliser dans les calculs de remontée du réseau de neurone (chemin retour), pendant lequel on calcule les ΔW_k et ΔB_k.

On se concentre tout d'abord sur la dernière couche du réseau de neurones (N) :
On a : $Z_N = f_N(A_N) = f_N(W_N Z_{N-1} + B_N)$ *i.e.*

$$h(F_{\mathcal{R}}(X), Y) = h(Z_N, Y) = h(f_N(W_N Z_{N-1} + B_N), Y) = g\big(w_{0,11}, w_{0,12}, \dots, b_{N,n}, X, Y\big)$$

D'où :

$$\left.\frac{\partial g}{\partial \kappa}\right|_{Z_N} = \nabla h(Z_N) \cdot \left.\frac{\partial f_N}{\partial \kappa}\right|_{A_N} = \begin{cases} \nabla h(Z_N) \cdot J(f_N)(A_N) \cdot \dfrac{\partial W_N}{\partial \kappa} \cdot Z_{N-1} \text{si } \kappa \text{ paramètre de } W_N \\ \nabla h(Z_N) \cdot J(f_N)(A_N) \cdot \dfrac{\partial B_N}{\partial \kappa} \text{ si } \kappa \text{ paramètre de } B_N \end{cases}$$

$$= \begin{cases} \delta_{N,i} z_j \text{ si } \kappa = w_{N,ij} \\ \delta_{N,i} \text{ si } \kappa = b_{N,i} \end{cases} \text{ avec } \delta_N = \nabla h(Z_N) \cdot J(f_N)(A_N) \text{ (vecteur ligne)}$$

où $\frac{\partial W_N}{\partial \kappa}$ et $\frac{\partial B_N}{\partial \kappa}$: matrices qu'avec des 0 sauf là où est le paramètre κ. On peut donc exprimer et calculer facilement ΔW_N et ΔB_N :

$$\Delta W_N = \delta_N^T Z_{N-1}^T$$
$$\Delta B_N = \delta_N^T$$

Il est à noter que dans cette étape, on ne prend pas en compte Y.

Quand on est dans une couche antérieure k, on définit $\widetilde{h_k}$ telle que :

$$\widetilde{h_k}(Z_k, Y) = h(Z, Y) \text{ où } Z = F_{\mathcal{R}}(X).$$

$\widetilde{h_k}$ est donc une fonction intermédiaire qui permet de retrouver le résultat donné par le réseau $\mathcal{R}$ quand il prend X en argument à partir d'un des résultats intermédiaires, Z_k. Or, on peut relier les $\widetilde{h_k}$ entre eux. En effet, pour relier $\widetilde{h_k}$ à $\widetilde{h_{k+1}}$ on remarque que :

$$\widetilde{h_k}(Z_k, Y) = \widetilde{h_{k+1}}(Z_{k+1}, Y) = \widetilde{h_{k+1}}(f_{k+1}(W_{k+1}.Z_k + B_{k+1}), Y) \ (*)$$

On peut donc faire exactement les mêmes calculs, il suffit de remplacer h par $\widetilde{h_k}$:

$$\left.\frac{\partial \widetilde{h_k}}{\partial \kappa}\right|_{Z_k} = \nabla \widetilde{h_k}(Z_k) \cdot \left.\frac{\partial f_k}{\partial \kappa}\right|_{A_k} = \begin{cases} \nabla \widetilde{h_k}(Z_k) \cdot J(f_k)(A_k) \cdot \dfrac{\partial W_k}{\partial \kappa} \cdot Z_{k-1} \text{si } \kappa \text{ paramètre de } W_k \\ \nabla \widetilde{h_k}(Z_k) \cdot J(f_k)(A_k) \cdot \dfrac{\partial B_k}{\partial \kappa} \text{ si } \kappa \text{ paramètre de } B_k \end{cases}$$

$$= \begin{cases} \delta_{k,i} z_j \text{ si } \kappa = w_{k,ij} \\ \delta_{k,i} \text{ si } \kappa = b_{k,i} \end{cases} \text{ avec } \delta_k = \nabla \widetilde{h_k}(Z_k) \cdot J(f_k)(A_k)$$

Or, d'après $(*)$: $\nabla \widetilde{h_k}(Z_k) = \nabla \widetilde{h_{k+1}}(Z_{k+1}) \cdot J(f_{k+1})(A_{k+1}) \cdot W_{k+1}$. On a donc la relation de récurrence suivante :

$$\delta_k = \delta_{k+1} \cdot W_{k+1} \cdot J(f_k)(A_k)$$

Finalement on peut décrire le principe d'une étape ι de fonctionnement d'un réseau de neurones de la façon suivante :

① On considère le réseau $\mathcal{R}$ (de profondeur N) à l'étape ι :
- Poids : $\{W_0, \dots, W_{N-1}\}$
- Biais : $\{B_0, \dots, B_{N-1}\}$
- Fonctions d'activation : $\{f_0, \dots, f_{N-1}\}$

② On effectue un feed-forward en conservant des valeurs :
- Sommes : $\{A_0, \dots, A_{N-1}\}$
- Activations : $\{Z_{-1}, Z_0, \dots, Z_{N-1}\} = \{A, f_0(A_0), \dots, f_{N-1}(A_{N-1})\}$

③ On effectue la backpropagation (pour $k \in [\![0, N-1]\!]$) :

$$\begin{cases} \delta_{N-1} = \nabla h(Z_{N-1}) \cdot J(f_{N-1})(A_{N-1}) \\ \delta_k = \delta_{k+1} \cdot W_{k+1} \cdot J(f_k)(A_k) \text{ si } k < N-1 \\ \Delta W_k = \big(\mathrm{d}w_{k,ij}\big) = \big(\delta_{k,i} z_{k-1,j}\big) = \delta_k^T Z_{k-1}^T \\ \Delta B_k = \big(\mathrm{d}B_{k,i}\big) = \big(\delta_{k,i}\big) = \delta_k^T \end{cases}$$

④ On met à jour les poids et les biais :

$$\begin{cases} W_k \leftarrow W_k - \alpha\, \Delta W_k \\ B_k \leftarrow B_k - \alpha\, \Delta B_k \end{cases}$$

Ici, nous n'avons décrit qu'une seule étape ι (*i.e.* pour un seul couple $(X_{\text{train}}, Y_{\text{train}})$). Mais il y a dans la pratique un très grand nombre d'étapes. Pour accélérer les calculs (et ainsi aller beaucoup plus vite qu'élément par élément), on utilise des batches. Ainsi, on assemble dans un même batch plusieurs vecteurs d'entrée (plusieurs X_{train}) à la fois. Un batch est donc une matrice d'entrée de la forme $(\mathbb{R}^n)^p$ (pour des batches avec p vecteurs, les sorties seront de la forme $(\mathbb{R}^m)^p$), qui permet de faire toutes les opérations en même temps. Lors de la mise à jour des poids et des biais, on fait une moyenne sur les $\Delta {W_k}^{\ell}$ et les $\Delta {B_k}^{\ell}$:

$$\begin{cases} W_k \leftarrow W_k - \dfrac{\alpha}{p} \displaystyle\sum_{\ell=1}^{p} \Delta {W_k}^{\ell} \\ B_k \leftarrow B_k - \dfrac{\alpha}{p} \displaystyle\sum_{\ell=1}^{p} \Delta {W_k}^{\ell} \end{cases}$$

On peut aussi rencontrer des problèmes d'approximation si on utilise de trop gros nombres. On utilise donc une régularisation de paramètre λ (où λ est un hyperparamètre) pour rester pas trop loin de 0 et aussi pour s'assurer que l'apprentissage s'arrête :

$$\begin{cases} W_k \leftarrow W_k - \dfrac{\alpha}{p} \displaystyle\sum_{\ell=1}^{p} \Delta {W_k}^{\ell} + \lambda\, W_k \\ B_k \leftarrow B_k - \dfrac{\alpha}{p} \displaystyle\sum_{\ell=1}^{p} \Delta {W_k}^{\ell} \end{cases}$$

On découpe l'ensemble des valeurs d'entraînement en batch, eux-mêmes regroupés en époques, et le nombre d'époques de l'apprentissage est aussi un hyperparamètre du problème.

Conclusion

Avant l'apprentissage, il faut réaliser un pré-apprentisage qui consiste en la sélection des X_{train} et du roulement des valeurs d'entraînement, et la mise en place des différents hyperparamètres :

- α, la taille du pas d'apprentissage
- λ, l'importance de la régularisation
- p, la taille des batchs
- le nombre d'époques.

www.playground.tensorflow.org