
ANALYSE NUMÉRIQUE : DEVOIR MAISON RG-2

Thomas Bonnet - Claire Bouissou - Lucas Perrin

24 mai 2020

Nous allons nous intéresser à la notion de *graphe computationnel*, ainsi qu'à celle de propagation *forward* et *backward*.

0.1 Théorie des graphes

Nous allons d'abord définir quelques notions de théorie des graphes :

Définition 1. Graphe orienté. Un graphe orienté est un couple $G = (V, A)$ comprenant :

- V un ensemble de nœuds ;
- A un ensemble d'arcs, $A \subseteq \{(x, y), (x, y) \in V^2\}$ chaque arc étant associé à deux nœuds, avec une direction.

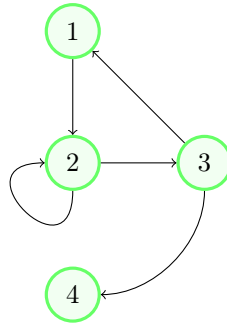


FIGURE 1 – Exemple de graphe orienté

Chaque *arc* $\{x, y\}$ va donc du nœud x au nœud y . L'arc $\{x, y\}$ n'est donc pas le même que l'arc $\{y, x\}$. Il est possible qu'un arc pointe vers le nœud duquel il part ($\{x, x\}$).

Remarque 1. Un graphe non orienté est un couple $G = (V, E)$ où $E \subseteq \{(x, y), (x, y) \in V^2, x \neq y\}$ est un ensemble d'arêtes qui relient deux nœuds différents entre eux. Un arc est donc une arête avec une direction. On parle de chaîne d'un nœud à un autre pour parler d'une suite consécutive d'arêtes qui relient les deux nœuds.

Définition 2. Graphe connexe. Un graphe est dit connexe s'il est d'un seul tenant, c'est-à-dire si pour toute paire de nœuds $v_1, v_2 \in V$ il existe une chaîne allant du nœud v_1 au nœud v_2 .

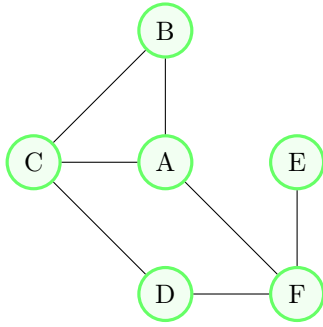


FIGURE 2 – Graphe connexe

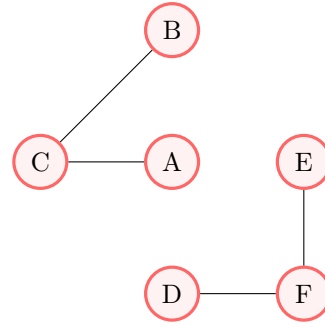


FIGURE 3 – Graphe non connexe

Définition 3. Graphe sans cycles. Un graphe sans cycles est un graphe qui ne contient aucune chaîne qui possède le même noeud de départ et d'arrivée.

Le graphe de la Figure fig. 2 possède plusieurs cycles (notamment $\{A, F, D, C\}$), celui de la Figure fig. 3 n'en possède aucun.

Définition 4. Degrés d'un graphe. Etant donné un arc $\{x, y\}$ reliant les nœuds x et y , on appelle le nœud de départ x la **source** et le nœud d'arrivée y la **cible**. On définit alors :

- le **degré sortant** d'un nœud est le nombre d'arcs qui ont pour source ce nœud ;
- le **degré entrant** d'un nœud est le nombre d'arcs qui ont pour cible ce nœud.

Un graphe comporte donc autant de degrés entrants que de degrés sortants.

Exemple 1. Dans le graphe de la fig. 1 :

- le degré sortant du nœud $\{2\}$ est 2, le degré sortant du nœud $\{4\}$ est nul.
- le degré entrant du nœud $\{3\}$ est 1, le degré entrant du nœud $\{2\}$ est 2.
- Il y a au total 5 degrés entrants et 5 degrés sortants, ce qui corrobore notre définition : il y a autant de degrés entrants que de degrés sortants.

0.2 Graphes computationnels, notions de *forward* et *backward*

0.2.1 Graphes computationnels directs

Nous abordons à présent le cœur du sujet : les graphes computationnels.

Définition 5. Graphe computationnel. Un graphe computationnel est un graphe orienté, connexe, sans cycles, dans lequel les nœuds correspondent à des variables ou à des opérations.

Chaque variable peut donc être utilisée par un nœud cible, et chaque opération prend en source le résultat des opérations précédentes. Son propre résultat peut être alors utilisé par d'autres nœuds d'opérations cibles.

Prenons en exemple la fonction f définie par :

$$\begin{aligned} f : \mathbb{R}^3 &\rightarrow \mathbb{R} \\ (x, y, z) &\mapsto (3x^2 + y)z - x \end{aligned}$$

Nous pouvons associer à cette fonction f le graphe computationnel suivant :

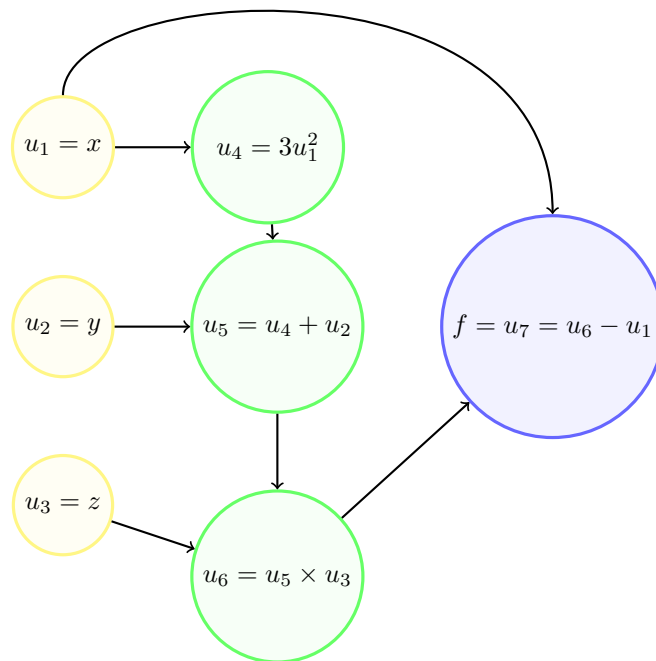


FIGURE 4 – Graphe computationnel de f

Remarque 2. les nœuds jaunes (u_1, u_2, u_3) n'ont pas de degrés entrants, ce sont des nœuds qui ne contiennent que les variables d'entrée à notre fonction f . Le nœud bleu u_7 constitue la sortie de la fonction f . Ce graphe est dit direct : il effectue les mêmes opérations que celles de la fonction.

Définition 6. Input, Output. On appelle Input d'un graphe computationnel direct, tous les nœuds ayant un degré entrant nul. On appelle Output d'un graphe computationnel tous les nœuds ayant un degré sortant nul.

Exemple 2. En reprenant la fig. 4, les nœuds jaunes (u_1, u_2, u_3) sont des Input, le nœud bleu u_7 est un output.

Définition 7. Fonction forward. On appelle fonction forward (forward propagation en anglais) d'un graphe computationnel la fonction qui prend en entrée les inputs du graphe, et en sortie les outputs. C'est celle qui "fait traverser les inputs vers l'avant".

Remarque 3. Il est évident que la fonction f dans la fig. 4 est la fonction forward de son propre graphe computationnel direct et cela peut sembler redondant. Mais ça ne l'est pas lorsqu'on possède uniquement le graphe computationnel.

Cette notion de fonction *forward* prendra toute son importance dans le cas d'un réseau de neurones par exemple. Nous poserons tout d'abord le graphe computationnel de notre réseau comme souhaité, il s'agira ensuite de retrouver cette fonction forward.

Dans un problème d'optimisation, nous allons chercher à obtenir le gradient de la fonction *forward*. C'est à ce moment-là que la notion de graphe computationnel inverse intervient.

0.2.2 Graphes computationnels inverses (ou rétrogrades)

Nous cherchons alors, via le graphe computationnel direct, à calculer le gradient de la fonction de propagation *forward*.

On note par la suite L , le nombre d'arêtes entrantes.

Définition 8. Fonction backward. On appelle fonction backward d'un graphe computationnel rétrograde, la fonction associée au gradient de la fonction forward de ce même graphe.

Exemple : Voir remarque 5.

Proposition 1. Gradient. Soit une fonction f (définie de \mathbb{R}^L dans \mathbb{R}) associée. Soit g_1, \dots, g_L des fonctions de \mathbb{R}^N dans \mathbb{R} telles que : $\forall X \in \mathbb{R}^N$

$$f(g_1(X), \dots, g_L(X)) = f(X)$$

La dérivée de la fonction f par rapport à X_k est donnée par :

$$\frac{\partial f}{\partial X_k} = \sum_{l \in \{0, \dots, L\}} \frac{\partial f}{\partial g_l}(g_1(X), \dots, g_L(X)) \frac{\partial g_l}{\partial X_k}$$

Le gradient de la fonction f est donc de la forme :

$$\nabla f = \left(\frac{\partial f}{\partial X_1}, \dots, \frac{\partial f}{\partial X_L} \right)^t$$

Remarque 4. Nous avons ci-dessus en fait la définition du gradient pour une fonction $\tilde{f} : \mathbb{R}^N \rightarrow \mathbb{R}$ mais avec un léger abus de notation. \tilde{f} est en fait définie comme une fonction composée ($f \circ g$), $\tilde{f}(X) = (f \circ g)(X)$ avec pour g :

$$\begin{aligned} g : \quad \mathbb{R}^N &\rightarrow \mathbb{R}^L \\ X = (X_1, \dots, X_N) &\mapsto (g_1(X), g_2(X), \dots, g_L(X)) \end{aligned}$$

et pour f :

$$\begin{aligned} f : \quad \mathbb{R}^L &\rightarrow \mathbb{R} \\ Y = (Y_1, \dots, Y_L) &\mapsto f(Y) \end{aligned}$$

on a donc bien : $\tilde{f} = (f \circ g)$.

On a alors :

$$\frac{d\tilde{f}}{dX} = \frac{d}{dX}(f \circ g) = \frac{df}{dg} \cdot \frac{dg}{dX}$$

f et g étant des fonctions de plusieurs variables, l'expression ci-dessus est un très léger abus de notation. Nous l'utiliseront donc par la suite. Cela équivaut à, avec la convention que $D_l f = \frac{\partial f}{\partial Y_l}$:

$$\frac{d}{dX} \tilde{f}(X) = \frac{d}{dX} f(g_1(X), \dots, g_L(X)) = \begin{pmatrix} \sum_{l=1}^L \left(\frac{d}{dX_1} g_l(X) \right) D_l f(g_1(X), \dots, g_L(X)) \\ \vdots \\ \sum_{l=1}^L \left(\frac{d}{dX_N} g_l(X) \right) D_l f(g_1(X), \dots, g_L(X)) \end{pmatrix}$$

Nous avons alors bien le résultat trouvé ci-dessus.

Nous sommes en droit de nous poser alors la question suivante : qu'en est-il pour une fonction $\tilde{F} : X \mapsto (\tilde{F}_1(X), \dots, \tilde{F}_P(X))$ à valeurs dans \mathbb{R}^P , où $\tilde{F}_p(X) = (F_p \circ g)(X)$? Nous aurons :

$$\begin{aligned} \frac{d}{dX} \tilde{F}(X) &= \begin{pmatrix} \frac{d}{dX_1} \tilde{F}_1(X) & \cdots & \frac{d}{dX_1} \tilde{F}_P(X) \\ \vdots & & \vdots \\ \frac{d}{dX_N} \tilde{F}_1(X) & \cdots & \frac{d}{dX_N} \tilde{F}_P(X) \end{pmatrix} = \\ &= \begin{pmatrix} \sum_{l=1}^L \left(\frac{d}{dX_1} g_l(X) \right) D_l F_1(g_1(X), \dots, g_L(X)) & \cdots & \sum_{l=1}^L \left(\frac{d}{dX_1} g_l(X) \right) D_l F_P(g_1(X), \dots, g_L(X)) \\ \vdots & & \vdots \\ \sum_{l=1}^L \left(\frac{d}{dX_N} g_l(X) \right) D_l F_1(g_1(X), \dots, g_L(X)) & \cdots & \sum_{l=1}^L \left(\frac{d}{dX_N} g_l(X) \right) D_l F_P(g_1(X), \dots, g_L(X)) \end{pmatrix} \end{aligned}$$

Remarque 5. Calcul rapide avec un seul passage.

Nous reprenons l'exemple vu dans la partie sur les graphes computationnels directs. A partir de ce dernier ainsi que du tableau suivant, nous allons construire le graphe computationnel rétrograde.

	U_4	U_5	U_6	U_7
nombre variables	1	2	2	2
fonction	$3U_1^2$	$U_4 + U_2$	$U_5 \times U_3$	$U_6 - U_1$

Nous utiliserons la notation suivante pour ce graphe :

$$\delta U_k = \frac{\partial f}{\partial U_k}$$

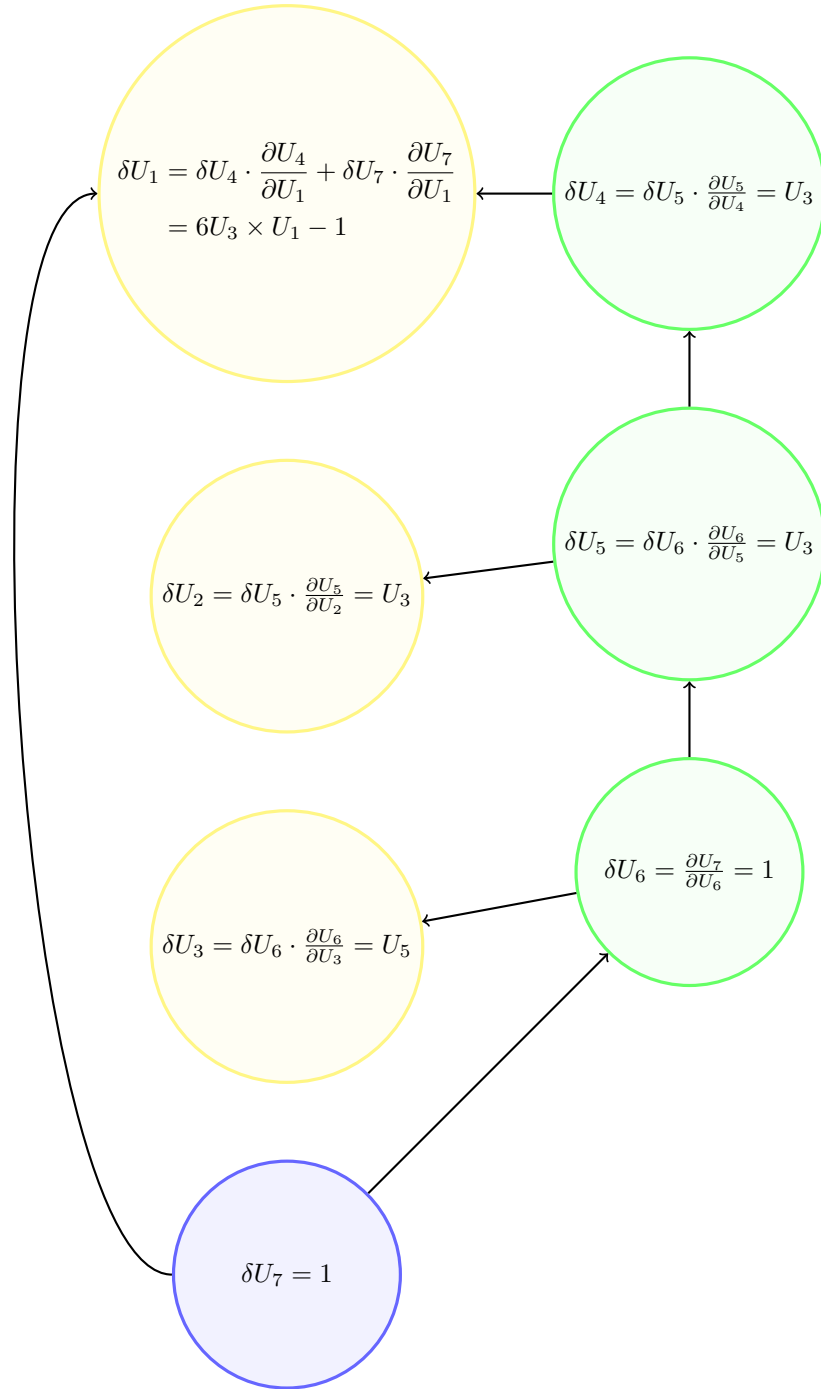


FIGURE 5 – Graphe rétrograde de f

Pour résumer, il est possible de créer un graphe computationnel **direct** (qui effectue les mêmes opérations que notre fonction) ou bien un graphe computationnel **rétrograde** (qui contient les dérivés).

Notons que ces graphes peuvent également prendre une forme plus aplatie et contenir toutes les variables utiles pour calculer nos valeurs. En reprenant l'exemple précédemment développé, nous obtenons le graphe computationnel direct sous cette forme.

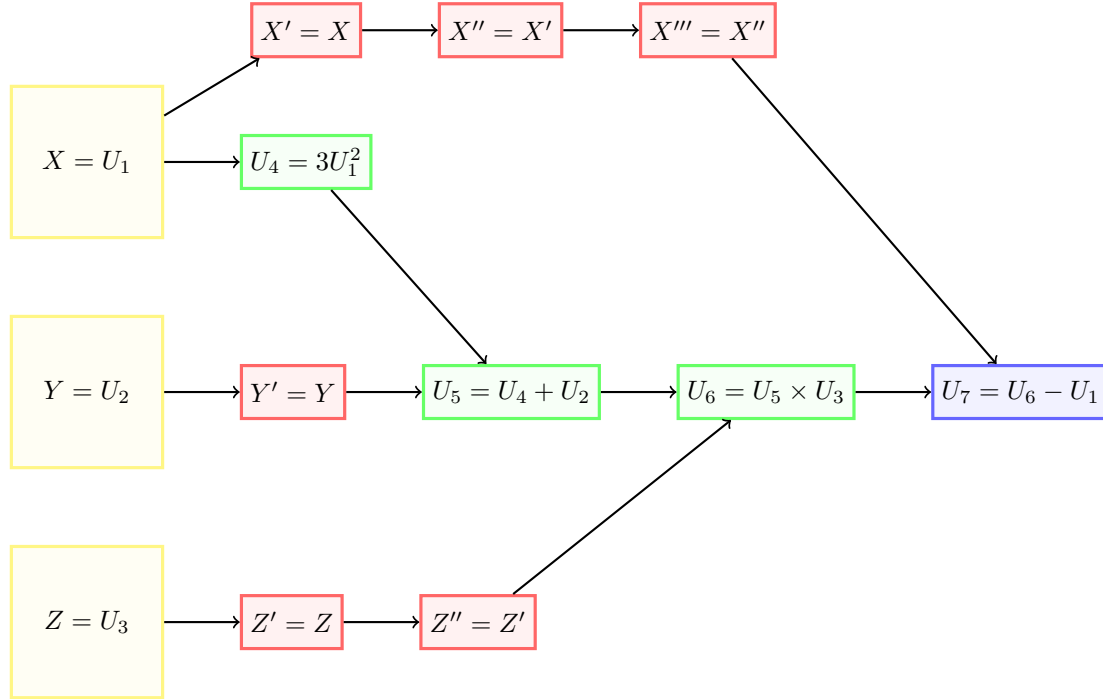


FIGURE 6 – Graphe direct aplati

Remarque 6. Cette représentation introduit des variables supplémentaires, ici en rouge, mais a l'avantage d'avoir une structure par bloc où chaque bloc est consacré au calcul de nos variables U_4, U_5, U_6 et U_7 .

0.3 Exemples : Réseaux neuronaux et contrôle

0.3.1 Réseau de neurones

Nous supposons que l'on possède une base de données Ω , on souhaite alors définir un réseau de neurones, *Neural Network* (NN), qui va "apprendre" par rapport à certains résultats de cette base de données, $\omega \in \Omega$.

On va alors chercher à optimiser certains paramètres X , pour que la sortie de notre réseau soit la plus proche des exemples ω choisis. C'est donc un problème d'optimisation :

$$\arg \min_X \{ \mathbb{E}_\omega (\mathcal{L}(X, \omega)) \}$$

On se propose d'étudier le réseau de neurones défini par le schéma suivant :

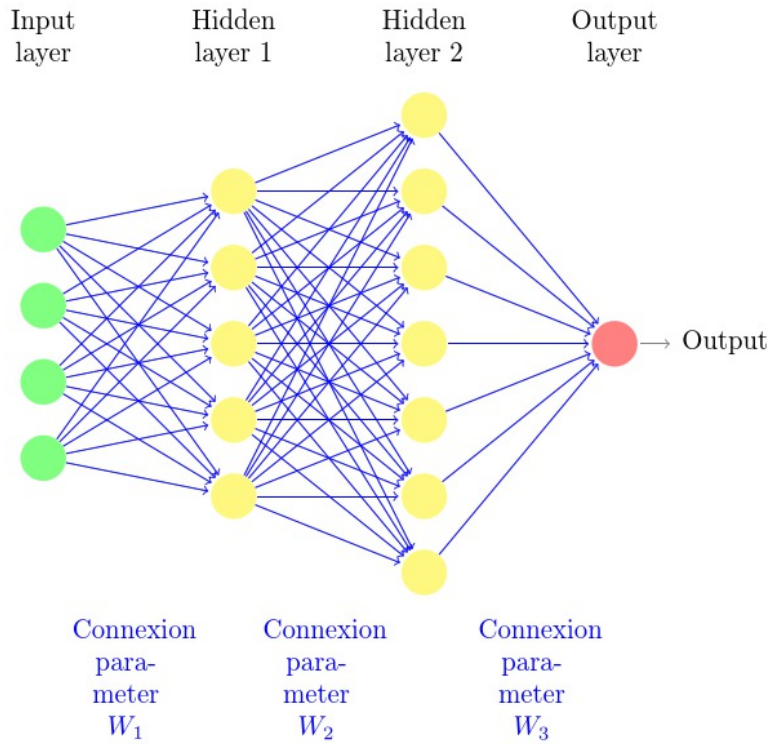


FIGURE 7 – Exemple d'un réseau de neurones

Définition du problème

Le principe est alors comme suit : on dispose d'une couche d'entrée, appelée *Input Layer*, d'un certain nombre de couches cachées, les *Hidden Layers*, et une couche de sortie, l'*Output Layer*.

Chacune des couches cachées (\tilde{Y}, Y) fait le calcul suivant : On prend tous les résultats de la couche précédente, on leur affecte un poids (un vecteur W , et un biais b). On obtient alors \tilde{Y} et nous appliquons enfin une fonction d'activation \mathcal{A} , nous obtenons Y . Enfin, nous appliquons à la dernière couche une fonction de sortie g .

D'une manière générale, on peut donc décrire la sortie Y_{n+1} d'une couche $n + 1$ donnée, par rapport à la sortie Y_n de la couche n précédente de la manière suivante :

$$\begin{cases} \tilde{Y}_{n+1} = W_{n+1}Y_n + b_{n+1} \\ Y_{n+1} = \mathcal{A}_{n+1}(\tilde{Y}_{n+1}) \end{cases}$$

On cherche alors à optimiser les poids W ainsi que les biais b , c'est-à-dire, à optimiser :

$$X = (W_1, b_1, \dots, W_n, b_n)$$

Pour minimiser une fonction de perte : $\mathcal{L}(X, \omega)$ qui décrit "l'écart entre l'estimation par le réseau de neurones avec les paramètres X , et les exemples ω ". Cette minimisation peut alors être effectuée avec une descente de gradient.

Graphe computationnel du réseau de neurones

Dans notre exemple défini par le graphe ci-dessus (fig. 7), nous avons :

- Une *Input Layer*, $Y_0 \in \mathbb{R}^4$;
- Une première *Hidden Layer*, $(\tilde{Y}_1, Y_1) \in \mathbb{R}^5$;
- Une seconde *Hidden Layer*, $(\tilde{Y}_2, Y_2) \in \mathbb{R}^7$;
- Une *Output Layer*, $O = g(Y_2) \in \mathbb{R}$.

Nous disposons donc du graphe computationnel suivant :

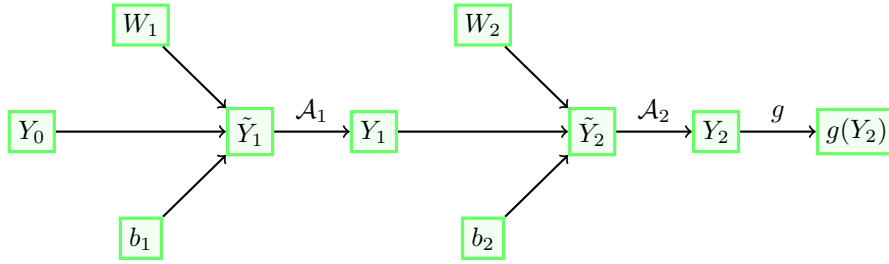


FIGURE 8 – Graphe de propagation forward du réseau de neurones

Calcul du gradient de la fonction de perte

Nous allons désormais nous intéresser au graphe rétrograde de ce réseau de neurones ainsi qu'au calcul du gradient :

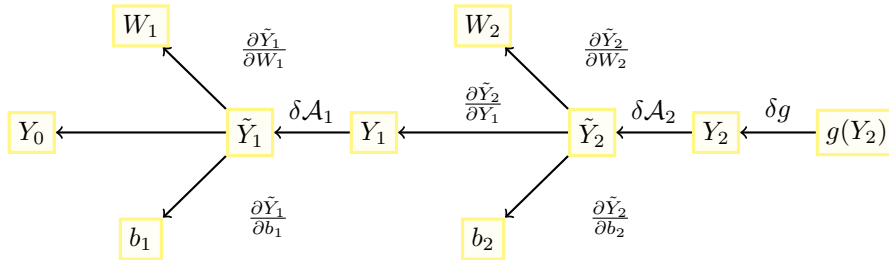


FIGURE 9 – Graphe de propagation backward du réseau de neurones

Nous avons alors :

$$\left\{ \begin{array}{l} \frac{\partial g}{\partial W_2} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial W_2} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial Y_2} \cdot \frac{\partial \tilde{Y}_2}{\partial W_2} \\ \frac{\partial g}{\partial b_2} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial Y_2} \cdot \frac{\partial \tilde{Y}_2}{\partial b_2} \\ \frac{\partial g}{\partial W_1} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial W_1} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial Y_2} \cdot \frac{\partial \tilde{Y}_2}{\partial W_1} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial Y_2} \cdot \frac{\partial \tilde{Y}_2}{\partial Y_1} \cdot \frac{\partial Y_1}{\partial W_1} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial Y_2} \cdot \frac{\partial \tilde{Y}_2}{\partial Y_1} \cdot \frac{\partial Y_1}{\partial Y_1} \cdot \frac{\partial \tilde{Y}_1}{\partial W_1} \\ \frac{\partial g}{\partial b_1} = \frac{\partial g}{\partial Y_2} \cdot \frac{\partial Y_2}{\partial Y_2} \cdot \frac{\partial \tilde{Y}_2}{\partial Y_1} \cdot \frac{\partial Y_1}{\partial Y_1} \cdot \frac{\partial \tilde{Y}_1}{\partial b_1} \end{array} \right.$$

On a donc le quadruplet : $\left[\frac{\partial g}{\partial W_1}, \frac{\partial g}{\partial b_1}, \frac{\partial g}{\partial W_2}, \frac{\partial g}{\partial b_2} \right]$ qui nous donne le gradient de g par rapport à $X = (W_1, b_1, W_2, b_2)$. Il suffit alors d'optimiser g par rapport à X avec une descente de gradient.

Remarque 7. Les calculs ci-dessus sont des chain-rules effectuées sur des fonctions de plusieurs variables à valeurs dans un espace multidimensionnel (sauf pour la fonction g , à valeurs dans \mathbb{R}). Les dérivées sont donc des matrices comme montré précédemment.

Notons qu'en apprentissage avec réseaux de neurones, le sur-apprentissage, ou *overfit*, est vite arrivé. Pour y remédier, nous pouvons instaurer une pénalité dans la correction apportée aux poids à chaque itérations de la descente de gradient, ou bien effectuer un *dropout*, c'est à dire supprimer de temps en temps un neurone pour forcer l'algorithme à ne pas sur-apprendre.

Remarque 8. Le gradient exprimé ci-dessus correspond à la proposition montrée plus tôt : chacune des dérivées partielles peut être exprimée comme une somme.

Descente de gradient stochastique

Nous rappelons la formule de la descente de gradient pour une fonction multidimensionnelle $F : \mathbb{R}^p \rightarrow \mathbb{R}$, de gradient ∇F :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

Si $\gamma \in \mathbb{R}$ est assez petit, on a donc $F(\mathbf{x}_n) \geq F(\mathbf{x}_{n+1})$ et la suite des $(\mathbf{x}_1, \mathbf{x}_2, \dots)$ converge vers le minimum local espéré, si certaines conditions sont respectées sur F (convexité notamment) et sur γ_n .

Nous connaissons certaines méthodes telles que la descente de gradient à pas constants ainsi que la descente de gradient à pas proportionnels qui jouent sur l'itération γ_n , aussi appelée "taux d'apprentissage" dans le cas d'apprentissage statistique, nous nous intéressons alors ici au principe de la descente de gradient stochastique.

Cette méthode est utilisée dans le cas de grandes dimensions, où le calcul de gradient en un point devient très coûteux. Le gradient se présente donc comme une somme de sous-gradients, ∇F_i (cf proposition 1) Nous prendrons alors, parmi les différents sous-gradients ∇F_i un échantillon tiré aléatoirement pour calculer le gradient de l'implémentation n de notre descente. Nous avons donc :

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n [\nabla_i F(\mathbf{x}_n)]$$

où $[\nabla_i F(\cdot)]$ est un échantillon (unique, somme, etc..) tiré aléatoirement parmi les sous-gradients de F .

Il s'agit donc ici de jouer sur le calcul du gradient, et non sur la taille du pas γ_n pour optimiser la descente. Plusieurs algorithmes de descente de gradient stochastique couplent différentes techniques (de variation de pas d'apprentissage ainsi que de descente stochastique) pour optimiser le temps de calcul de certains modèles d'apprentissage.

0.3.2 Problème de contrôle

Nous nous plaçons désormais dans un contexte actuel avec le système SIR, vu précédemment.

Rappel du problème

Les problèmes liés à la crise que nous vivons actuellement (crise sanitaire du coronavirus) peuvent être représentés par un système, que l'on intitule SIR. Le système est de la forme :

$$\begin{cases} X' = f(X(t), u(t)) \\ \dot{S} = -\beta SI \\ \dot{I} = \beta SI - \delta I \\ \dot{R} = \delta I \end{cases}$$

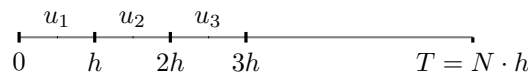
avec $u = \beta(t) = \beta$ le contrôle.

Nous obtenons alors $X = (S, I, R)$

Ainsi le but est de minimiser le coût du confinement ($\int_0^T c(\beta(t))dt$) et le nombre de gens atteints ($S(0) - S(T)$) au cours de la période $[0, T]$. Autrement dit, on cherche :

$$\min_{\beta} S(0) - S(T) + \int_0^T c(\beta(t))dt$$

Nous noterons par la suite $J(u) = S(0) - S(T) + \int_0^T c(\beta(t))dt$, avec u , l'ensemble des fonctions u_n du temps que l'on va discrétiser :



$0 \quad h \quad 2h \quad 3h \quad T = N \cdot h$

Nous avons dans un premier temps :

$$\frac{\partial}{\partial \beta} \left[\int_0^T c(\beta(t))dt \right] = c'(\beta(t))$$

En revanche la différence $S(0) - S(T)$ est beaucoup moins évidente à dériver car le β n'apparaît pas explicitement, bien qu'il intervienne dans cette fonction. Nous posons pour simplifier notre notation par la suite : $F(S(T)) = S(0) - S(T)$.

Nous cherchons donc à calculer :

$$\frac{\partial}{\partial u_n} [F(X_T)]$$

Pour ce faire, nous allons nous aider de la méthode d'Euler Explicite. $X_{n+1} = X_n + h \times f(u_n, x_n)$

Graphes computationnels

La première manière de résoudre ce problème est de faire les graphes computationnels. Dans un premier temps, nous obtenons le graphe computationnel direct suivant :

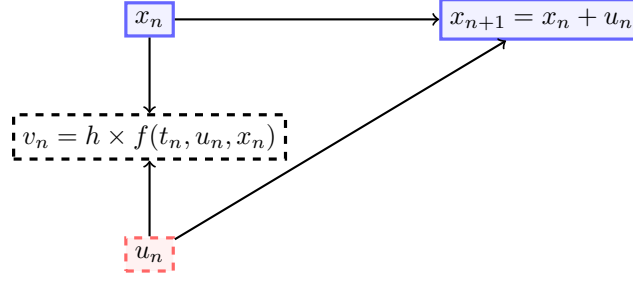


FIGURE 10 – Graphe direct du système SIR

Nous obtenons par la suite le graphe computationnel rétrograde suivant :

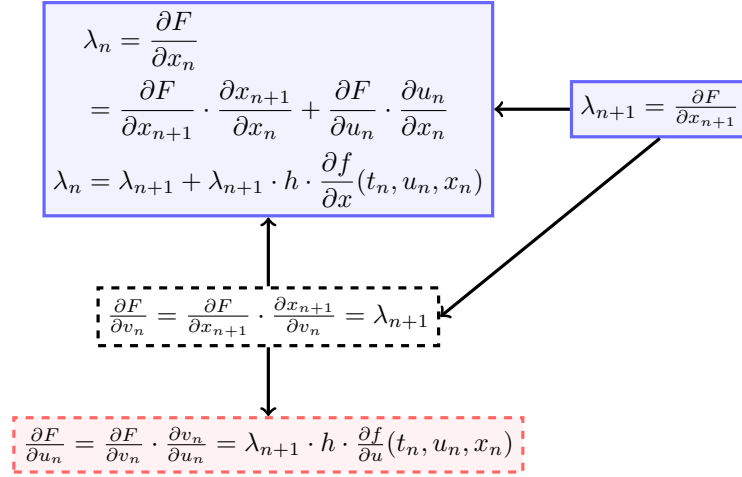


FIGURE 11 – Graphe rétrograde du système SIR

Cependant, ces graphes ne représentent qu'une partie du réel graphe. En effet, le graphe direct complet est de la forme pour $n = 2$:

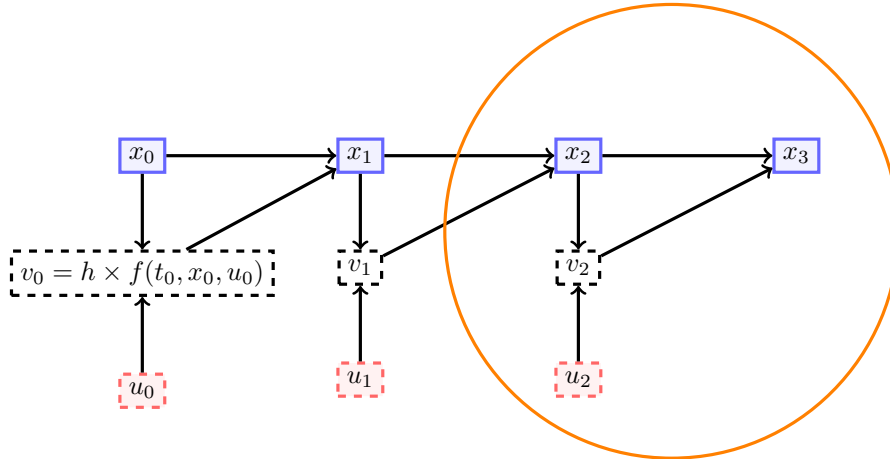


FIGURE 12 – Graphe direct complet du système SIR

Le cercle orange représente ici pour $n = 2$ la partie que nous avons tracée précédemment. fig. 10. Nos deux derniers graphes fig. 10 et fig. 11 sont donc plus généraux mais ne montrent qu'une partie du graphe computationnel.

Construction d'une version discrète d'une procédure d'Euler Lagrange

Les graphes computationnels directs et retrogrades nous permettent de construire une version discrète d'une procédure d'Euler Lagrange que nous avons vue plus tôt dans le cours.

En effet, de la discrétisation de X' , nous obtenons le système (\mathcal{S}_1) :

$$(\mathcal{S}_1) : \begin{cases} X'(t) = f(t, X(t), u(t)) \\ X_{n+1} = X_n + hf(t_n, X_n, u_n) \end{cases}$$

De même, en pratiquant une discrétisation rétrograde dans le temps sur λ_n nous avons donc le système (\mathcal{S}_2) comme trouvé en fig. 11 :

$$(\mathcal{S}_2) : \begin{cases} \lambda_n = \lambda_{n+1} + \lambda_{n+1} \frac{\partial f}{\partial X}(t_n, X_n, u_n) \cdot h \\ \lambda'(t) = -\frac{\partial f}{\partial X}(t, X(t), u(t)) \times \lambda(t) \end{cases}$$

Nous obtenons également notre dérivée :

$$\frac{\partial F}{\partial u_n} = \lambda_{n+1} \cdot h \cdot \frac{\partial f}{\partial u_n}(t_n, X_n, u_n)$$

Démonstration. Nous considérons un problème de minimisation de type Euler Lagrange de la forme :

$$\min_{X'=f(t,X,U)} F(X(t))$$

Pour minimiser cette quantité, nous construisons la fonction $F(X(t))$ avec le mutliplicateur de Lagrange, nous obtenons une quantité que nous nommons $G(u)$:

$$G(u) = F(X(t)) + \int_0^T (X' - f(t, X, u)\lambda(t))dt$$

où $\lambda(t)$ est le mutiplicateur de Lagrange

Il convient ensuite de réaliser les dérivées partielles de G en fonction de λ et de X .

La première dérivée nous permet d'obtenir la contrainte :

$$\frac{\partial G}{\partial \lambda} : X' = f(t, X, u)$$

La deuxième dérivée, elle, va nous permettre de faire le lien entre notre graphe computationnel et la procédure d'Euler Lagrange :

$$\begin{aligned} \frac{\partial G}{\partial X} : \frac{\partial}{\partial X} \int_0^T \lambda(X' - f(t, X, u))dt \\ \stackrel{IPP}{=} \frac{\partial}{\partial X} \left[[\lambda X]_0^T - \int_0^T X\lambda' - \lambda f dt \right] \\ = \lambda' - \lambda \frac{\partial f}{\partial X}(t, X(t), u(t)) \\ = 0 \end{aligned}$$

Nous retrouvons l'équation que nous avons dans le système : (\mathcal{S}_2)

□

Donc le graphe computationnel nous permet d'obtenir une version discrète de la procédure d'Euler Lagrange pour notre variable de contrôle.

Sources

Liens : cliquer sur le titre :

1. "Deep Learning From Scratch I: Computational Graphs",
2. David Gontier, "Méthodes Numériques : Optimisation",
3. Gabriel Turinici, "Méthodes numériques: problèmes dépendant du temps"
4. Lucas Perrin & David Gontier, "Réseau de neurones à plusieurs couches (généralisation)",
5. Gabriel Turinici, YouTube,
 - Introduction au calcul backward
 - Calcul du graphe computationnel direct et retrograde
 - Applications du graphe computationnel: réseaux neuronaux et contrôle