UNIVERSITÉ RENNES 2

Projet Cluedo

Xavier André & Romain Tavenard

1 Préambule

Pour ce projet, vous devrez travailler par groupes de 2 ou 3 et votre rendu se fera sous la forme d'un fichier Python. Ce fichier devra être nommé `P03.py` et contenir les noms et numéros étudiant de tous les membres du groupe commentés, en en-tête du fichier, comme dans l'exemple suivant :

```
# 22000002 Paul Machin
# 22000227 Yolène Truc

# Section 1 : les imports
# [...]
```

La date limite de rendu est indiquée sur CURSUS dans l'espace de dépôt.

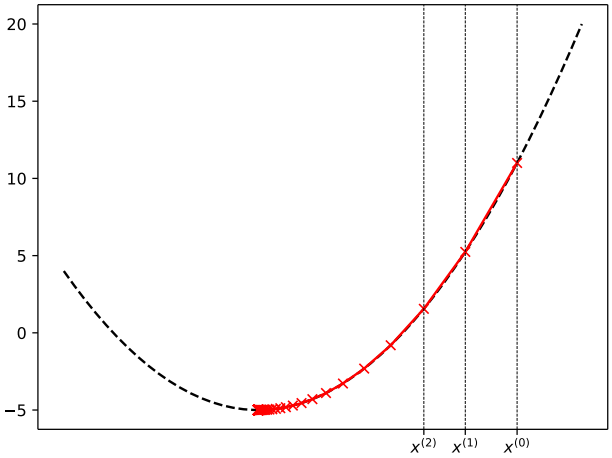
2 La différentiation automatique, qu'est-ce que c'est ?

2.1 Un cas d'usage : la descente de gradient

Supposons que l'on souhaite trouver la valeur de x qui minimise la fonction suivante :

$$f(x) = x^2 + 2(x - 2)$$

Une méthode classique consiste à partir d'une valeur de $x^{(0)}$ prise au hasard, puis de suivre la pente de la fonction $f(x)$ (indiquée par sa dérivée $f'(x)$). On a donc une suite de valeurs $x^{(t)}$ qui converge vers la solution de notre problème d'optimisation, comme dans l'exemple ci-dessous :



On voit ici que $x^{(0)}$ est fixé arbitrairement puis, observant que la pente de la fonction est positive en $x^{(0)}$, on "essaye" avec une valeur plus petite, puis plus petite encore, jusqu'à atteindre une zone pour laquelle $f'(x) \approx 0$. Plus précisément, à chaque étape (t) de l'algorithme, en supposant que $x^{(t)}$ est la valeur de x avant l'étape courante, la règle de mise à jour est la suivante :

$$x^{(t+1)} = x^{(t)} - \rho f'(x^{(t)})$$

où ρ est un paramètre de la méthode appelé taux d'apprentissage (ou *learning rate*). Comme vous le verrez en M2, cette méthode d'optimisation est notamment très utilisée pour ajuster les paramètres de modèles de *Machine Learning* appelés "réseaux de neurones" (c'est notamment de ces modèles que l'on parle lorsque l'on parle de *Deep Learning*).

2.2 La différentiation automatique

Pour pouvoir mettre à jour la valeur de x à chaque itération, il est nécessaire d'être capable de calculer la dérivée de la fonction f en x. C'est là qu'intervient la différentiation automatique. Le principe de la différentiation automatique est de faire calculer la dérivée d'une fonction (on parlera dans la suite d'expression mathématique plutôt que de fonction mais le principe sera le même) par l'ordinateur.

Pour ce faire, si l'on reprend l'exemple de la fonction f définie plus haut, il faut remarquer qu'il s'agit de la somme de deux termes et que donc la dérivée de cette expression est la somme des dérivées de chacun des deux termes car :

$$(u + v)'(x) = u'(x) + v'(x)$$

En continuant le raisonnement, la dérivée du premier terme de la somme se calcule en utilisant la formule de dérivation :

$$(u \times u)'(x) = 2u(x)u'(x)$$

et ainsi de suite. En résumé, si l'on est capable de décomposer une expression mathématique en calculs élémentaires (somme, produit, élévation au carré, *etc.*) et que pour chacun de ces calculs élémentaires on est capable d'exprimer sa dérivée, alors on sait calculer la dérivée de l'expression mathématique initiale.

C'est le principe de base de la différentiation automatique, et votre tâche pour ce projet sera de programmer un outil de différentiation automatique qui soit en mesure de calculer automatiquement la dérivée d'une expression mathématique composée de calculs simples.

3 Description du code à produire

Les étapes suivantes sont conçues comme des étapes de difficulté croissante, à vous d'aller le plus loin possible dans l'implémentation de ces éléments.

1. Implémentez une classe `Expression` qui aura deux méthodes : une méthode `forward(self, x)` qui évaluera l'expression en question pour une valeur de `x` fournie, et une méthode `grad(self, x)` qui calculera la dérivée de la fonction évaluée en une valeur `x` fournie. Ces méthodes seront des méthodes abstraites et n'ont pas vocation à être implémentées dans la classe `Expression`, uniquement définies.
2. Définissez un sous-type d'`Expression` pour chacun des cas suivants :
   — une constante (on appellera la classe correspondante `Constante`) ;
   — la fonction identité qui à x associe lui-même (on appellera la classe correspondante `Variable`).
3. Définissez un sous-type d'`Expression` pour chacun des cas suivants :
   — la somme de deux expressions ;
   — le produit de deux expressions.
4. Modifiez votre implémentation de la classe `Expression` pour que, si `expA` et `expB` sont des expressions, on puisse écrire `(expA + expB)` pour construire la somme de ces deux expressions et `(expA * expB)` pour construire le produit de ces deux expressions.
5. Utilisez ce code pour développer un algorithme de **descente de gradient** (voir explications plus haut) qui prend en entrée une expression mathématique, une valeur initiale pour la variable x et un taux d'apprentissage et cherche la valeur de x qui minimise l'expression en question. On fixera le nombre d'itérations de l'algorithme à une valeur codée en dur, de l'ordre de quelques dizaines d'itérations.

6. Étoffez votre implémentation en choisissant parmi les propositions suivantes (ou en imaginant vous-même des extensions) :
   — la prise en compte de nouvelles opérations arithmétiques comme l'élévation à la puissance ou la division ;
   — l'affichage propre dans le terminal d'une expression `e` lorsque l'on exécute la commande `print(e)` ;
   — l'ajout d'autant d'améliorations que nécessaires pour que l'on puisse définir une expression aussi facilement que :

```
x = Variable()
expr = x ** 2 + 2 * (x - 2)
```