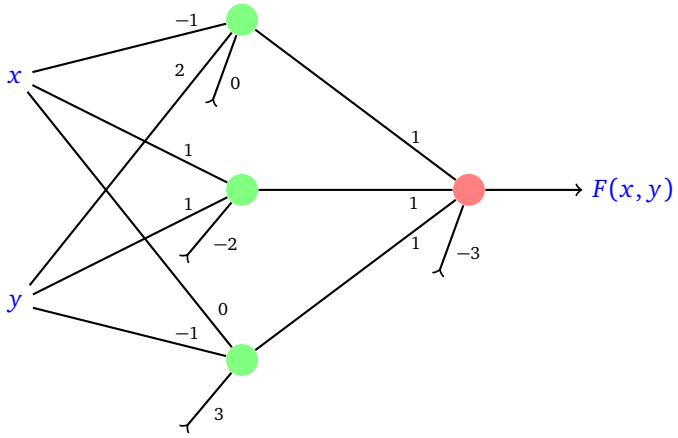
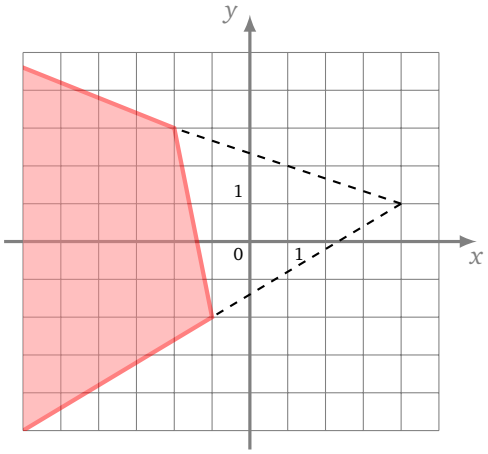
Réseaux de neurones

Exercice 1 (*). Opération logique ET Construire un perceptron linéaire réalisant l'opération $x\ ET\ y\ ET\ z$.

Exercice 2 (*). Un premier réseau de neurones simple Déterminer l'expression de la sortie du réseau de neurones suivant, pour lequel la fonction d'activation est la fonction de Heaviside partout. Simplifier au maximum le résultat obtenu.



Exercice 3 (*). Un second exemple En utilisant les idées de l'exercice précédent et pour le dessin ci-dessous, trouver un réseau de neurones dont la fonction F vaut 1 pour la zone colorée et 0 ailleurs.



Exercice 4 (**). Ajustement des paramètres d'un réseau de neurones simple On considère dans tout l'exercice la fonction affine par morceaux f est définie sur $[0,2]$ comme suit :

$$f(x) = \begin{cases} -3x+3 & \text{si } 0 \le x < 1, \\ x-1 & \text{si } 1 \le x \le 2. \end{cases}$$

1. Représenter la fonction f.
2. On rappelle que $\mathrm{ReLU}(x) = \max(0,x)$ pour tout x réel. Justifier que $f(x) = \mathrm{ReLU}(-3x+3) + \mathrm{ReLU}(x-1)$ pour tout x dans $[0,2]$.
3. Construire un réseau de neurones simple qui définit la fonction f. Quel est son nombre de paramètres ?
4. Implémenter ce réseau à l'aide de Python et vérifier qu'il définit bien la fonction f.
5. Des données d'entraînement légèrement bruitées générées à partir de la fonction f sont disponible sur Moodle. Les utiliser pour entraîner un réseau de neurones ayant la même structure qu'à la question 3. Commenter.

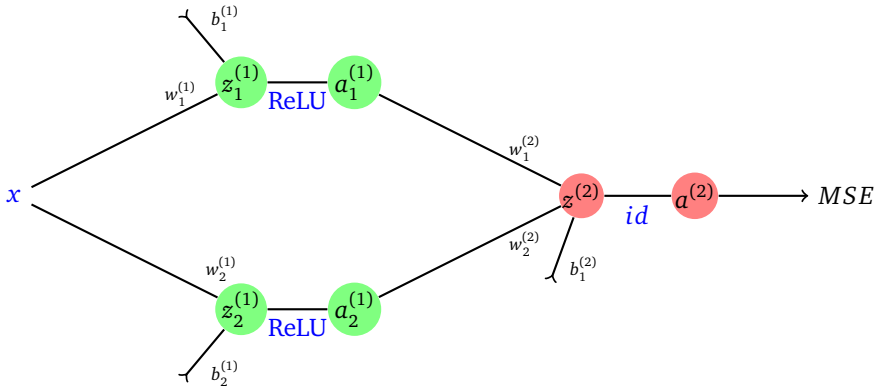
6. Proposer des solutions permettant d'obtenir un réseau entraîné produisant un résultat proche de celui souhaité, et les tester.

7. Dans cette question, on va étudier l'aspect mathématique de l'entraînement du modèle afin d'expliquer la différence entre les résultats obtenus précédemment et le résultat souhaité.

   (a) Rappeler l'expression de la fonction d'erreur utilisée et son rôle dans l'entraînement du modèle.

   (b) On considère le graphe de calcul de la fonction d'erreur avec les notations suivantes :

   - $w_i^{(l)}$ désigne le i-ème poids de la couche l,
   - $b_i^{(l)}$ désigne le i-ème biais de la couche l,
   - $z^{(l)}$ désigne la préactivation de la couche l (le calcul de la somme pondérée),
   - $a^{(l)}$ désigne l'activation de la couche l (l'application de la fonction d'activation), ainsi $a^{(2)}$ représente $F(x)$.



   i. Calculer les dérivées (partielles) suivantes

$$\frac{\partial MSE}{\partial a^{(2)}}, \quad \frac{\partial MSE}{\partial z^{(2)}},$$

   et en déduire les premières dérivées partielles intéressantes pour notre problème d'optimisation

$$\frac{\partial MSE}{\partial b_1^{(2)}}, \quad \frac{\partial MSE}{\partial w_i^{(2)}}, \text{ pour } i = 1, 2.$$

   ii. En appliquant la même méthode, calculer les dérivées partielles manquantes pour résoudre notre problème d'optimisation. Ce faisant, on est en train de détailler les calculs de l'algorithme de *rétro-propagation du gradient*.

   (c) Implémenter le calcul de la fonction d'erreur ainsi que de son gradient à l'aide de Python, puis appliquer à cette fonction l'algorithme de descente de gradient que l'on a construit durant ce cours (on considère les mêmes données d'entraînement que dans la question 5). Commenter les résultats obtenus.

   (d) Quelle explication mathématique donner aux résultats obtenus ?