Mathis Degryse <mathis.degryse@proton.me>

Option B 2023
Fondements de l'informatique

Partie I. Le cas d'un neurone à seuil

Partie 1. Fonctions booléennes linéaires à seuil

Question 1

$\mathrm{OR}_n(x_1, ..., x_n) = 1$ si et seulement si au moins un des $(x_i)_{1 \leq i \leq n}$ vaut 1. C'est à dire si et seulement si $x_1 + ... + x_n \geq 1$. On en déduit que $\left(\underbrace{1, 1, ..., 1}_{n}, 1\right)$ est une représentation de OR_n. On a de même $\mathrm{NOT}_1(x) = 1$ si et seulement si $x = 0$, c'est à dire $x \leq 0$, ou $-x \geq 0$, donc $(-1, 0)$ est une représentation de NOT_1.

Pour $\mathrm{MAJORITY}_n$ $1x_1 + 1x_2 + ... + 1x_n$ est égal au nombre de $i \in [\![1, n]\!]$ tel que $x_i = 1$. On en déduit que $1x_1 + 1x_2 + ... + 1x_n \geq \frac{n}{2}$ si et seulement si $\frac{n}{2}$ au moins des entrées valent 1. Ainsi, $\left(\underbrace{1, 1, ..., 1}_{n}, \frac{n}{2}\right)$ est une représentation de $\mathrm{MAJORITY}_n$, qui est donc une fonction booléenne linéaire à seuil.

Question 2

Supposons que $\oplus$ soit une fonction booléenne linéaire à seuil. Il existe $w_1, w_2, h \in \mathbb{R}$ tel que $a \oplus b = 1$ si et seulement si $aw_1 + bw_2 \geq h$

On a ainsi

$$\begin{aligned} 0 &< h \\ w_1 &\geq h \\ w_2 &\geq h \\ w_1 + w_2 &< h \end{aligned} \tag{1}$$

Ces relations ne peuvent pas être toutes vraies, car les trois dernières nous donne que $2h < h$, or comme $0 < h$, ceci est impossible.

On en déduit que $\oplus$ n'est pas une fonction booléenne linéaire à seuil.

Question 3

- Pour $n = 2$, on remarque que PARITY_n n'est autre que l'opposé logique de $\oplus$. Si elle était une fonction booléenne linéaire à seuil, de représentation (w_1, w_2, h) on aurait

  $$\begin{aligned} 0 &\geq h \\ w_1 &< h \\ w_2 &< h \\ w_1 + w_2 &\geq h \end{aligned} \tag{2}$$

  et ainsi $h < 2h$ et $h \leq 0$, ce qui est impossible.

- Pour $n \geq 3$, Supposons que PARITY_n soit une fonction booléenne linéaire à seuil de représentation $(w_1, ..., w_n, h)$. On a alors qu'un nombre pair des entrées $x_1, ..., x_n$ valent 1 si et seulement si $w_1x_1 + ... + w_nx_n \geq h$

  Ainsi, un nombre pair des entrées $(x_1, x_2, 0, 0, ..., 0)$ valent 1 si et seulement si $w_1x_1 + w_2x_2 \geq h$, ce qui nous donnerait une représentation de la fonction PARITY_2, ce qui est impossible.

Finalement, pour $n \geq 2$, PARITY_n n'est pas une fonction booléenne linéaire à seuil.

Partie 2. Cas de domaines bornés

Question 4

Soit $(w_1, ..., w_n, h)$ une représentation d'une fonction linéaire à seuil, et $S \subseteq \mathbb{R}^n$ un ensemble fini. On considère

$$\delta = h - \max \underbrace{\left\{ \sum_{i=1}^{n} w_i x_i \mid (x_1, ..., x_n) \in S \wedge \sum_{i=1}^{n} w_i x_i < h \right\}}_{A} > 0 \tag{3}$$

Cet élément existe justement parce que S est fini. Soit maintenant $(x_1, ..., x_n) \in S$.

- Si $\sum_{i=1}^{n} w_i x_i < h$, alors $\sum_{i=1}^{n} w_i x_i \in A$ et donc $\sum_{i=1}^{n} w_i x_i \leq \max A = h - \delta$.
- Sinon $\sum_{i=1}^{n} w_i x_i \geq h$, et alors par positivité stricte de δ, il vient $\sum_{i=1}^{n} w_i x_i > h - \delta$.

Ainsi, pour tout $x_1, ..., x_n \in S$, on a $\sum_{i=1}^{n} w_i x_i < h$ si et seulement si $\sum_{i=1}^{n} w_i x_i \leq h - \delta$.

Question 5

Soit $\delta > 0, S \subseteq \mathbb{R}^n$ et F une fonction linéaire à seuil λ-séparable, de masse w.

On a $\sum_{i=1}^{n} w_i x_i \geq h$ si et seulement si $\sum_{i=1}^{n} \frac{\delta w_i}{\lambda} x_i \geq \frac{\delta h}{\lambda}$. Ainsi, $\left(\frac{\delta w_1}{\lambda}, ..., \frac{\delta w_n}{\lambda}, \frac{\delta h}{\lambda}\right)$ est une représentation de F, de poids $\frac{\delta w}{\lambda}$. Il nous reste à montrer qu'elle est δ-séparable.

On a, comme $(w_1, ..., w_n, h)$ est λ-séparable, que $\sum_{i=1}^{n} w_i x_i < h \iff \sum_{i=1}^{n} w_i x_i \leq h - \lambda$. En multipliant ces deux relations par $\frac{\delta}{\lambda}$, qui est une constante strictement positive, on obtient

$$\sum_{i=1}^{n} \frac{\delta w_i}{\lambda} x_i < h \iff \sum_{i=1}^{n} \frac{\delta w_i}{\lambda} x_i \leq \frac{\delta h}{\lambda} - \delta \tag{4}$$

Ce qui nous permet de conclure.

Question 6

Soit F une fonction linéaire à seuil de représentation $(w_1, ..., w_n, h)$. Cette fonction accepte ses entrées $x_1, ..., x_n$ si et seulement si $\sum_{i=1}^{n} w_i x_i \geq h$, c'est à dire si et seulement si $\sum_{i=1}^{n} w_i x_i - 1 \cdot h \geq 0$. Ainsi, $F(x_1, ..., x_n) = G(x_1, ..., x_n, 1)$ où G est une fonction linéaire à seuil dont une représentation est $(w_1, ..., w_n, -h)$

Question 7

Remarque 1

Une erreur dans l'énoncé ?

- Choisir $w'_i = \lfloor w_i \rfloor$ ne fonctionne pas, car dans ce cas on doit choisir $h' = \lfloor h \rfloor$, qui ne vérifie pas toutes les conditions.

On peut trouver une preuve qui marche si c'est $n+1$-séparable, une autre si on se laisse augmenter la masse du neurone, une autre si on autorise seulement h à ne pas être entier. Mais ces 3 conditions, mises ensemble, ne donne aucune solution simple.

Question 8

Remarque 2

Aucune idée

Partie 3. Apprentissage d'un neurone à seuil

Question 9

Si il existe une solution F au problème de l'apprentissage exact des fonctions linéaires à seuil sur le domaine D pour l'instance $(x_i, y_i)_{1\leq i\leq N}$, de représentation $(w_1, ..., w_n, h)$, alors on a pour tout $i \in [\![1, N]\!]$, $w \cdot x_i \geq h \iff y_i = 1$ (où $w = (w_1, ..., w_n)$).

Ainsi, on a aussi

$$\forall i \in [\![1, N]\!], (w_1, ..., w_n, -h) \cdot \left(x_{i_1}, ..., x_{i_n}, 1\right) \geq 0 \iff y_i = 1 \tag{5}$$

On crée donc l'instance (x'_i, y_i) où on a pour tout $i \in [\![1, N]\!]$, $x'_i = (x_i, 1)$.

On a montré que si F existe, alors il existe une solution au problème de l'apprentissage exact des fonctions linéaires à seuil nul pour l'instance (x'_i, y_i)?

Inversement, s'il existe une solution G au problème de l'apprentissage exact des fonctions linéaires à seuil nul sur le domaine D pour l'instance $(x'_i, y_i)_{1\leq i\leq N}$, de représentation $(w_1, ..., w_n, w_{n+1}, 0)$, alors on a pour tout $i \in [\![1, N]\!]$, $w \cdot x_i \geq 0 \iff y_i = 1$ (où $w = (w_1, ..., w_n, w_{n+1})$).

Ainsi, on a aussi

$$\forall i \in [\![1, N]\!], (w_1, ..., w_n) \cdot \left(x_{i_1}, ..., x_{i_n}\right) \geq -w_{n+1} \iff y_i = 1 \tag{6}$$

Et donc, il existe une solution au problème initial.

Finalement, il existe une solution F au problème initial si et seulement si il existe une solution G au second problème. On a donc bien une réduction au problème de l'apprentissage exact des fonctions linéaires à seuil nul. Notons d'ailleurs que dans l'instance au second problème, il n'y a aucun test nul (où tous les x_{i_k} sont nuls)

Question 10

On considère un problème A d'apprentissage exact des fonctions linéaires à seuil sur $D = [0, 1]$, d'instance $(x_i, y_i)_{1\leq i\leq N}$. On considère alors un problème B, réduction de A correspondant à la question précédente, d'instance $(x'_i, y_i)_{1\leq i\leq N}$. Notons qu'il n'y a aucun test nul dans B.

A possède une solution si et seulement si il existe $w \in \mathbb{R}^{n+1}$ tel que pour tout $i \in [\![1, N]\!]$, $w \cdot x'_i \geq 0 \iff y_i = 1$

Supposons que B ait une telle solution $w \in \mathbb{R}^{n+1}$. Notons qu'il existe $\delta > 0$ tel qu'elle soit δ-séparable (question 4). Montrons alors que B possède une solution w' telle que pour tout $i \in [\![1, N]\!]$, $w' \cdot x'_i \geq 0 \iff y_i = 1$ et $w' \cdot x_i \neq 0$. Pour cela, on considère

$$w'_i = w_i + \frac{\delta}{2n} \tag{7}$$

En effet, si $w \cdot x'_i < 0$, alors comme w est δ-séparable, on a $w \cdot x'_i < -\delta$ et donc

$$\begin{aligned} w' \cdot x'_i &= w \cdot x'_i + \sum_{k=1}^{n} \frac{\delta}{2n} \cdot x'_{i_k} \\ &\leq w \cdot x'_i + \sum_{k=1}^{n} \frac{\delta}{2n} \\ &\leq w \cdot x'_i + \frac{\delta}{2} < 0 \end{aligned} \tag{8}$$

Et si $w \cdot x'_i \geq 0$, alors $w' \cdot x'_i > w \cdot x_i \geq 0$, l'inégalité stricte étant vérifiée uniquement parce que B n'a pas de test nul. On construit alors l'instance $(x''_i, 0_{\mathbb{B}^n})$ tel que pour tout $i \in [\![1, N]\!]$, on ait

$$x''_i = \begin{cases} x'_i \text{ si } \ y_i = 0 \\ -x'_i \text{ sinon} \end{cases} \tag{9}$$

On peut alors montrer facilement que $w' \cdot x''_i \leq 0$ pour tout $i \in [\![1, N]\!]$. $(w', 0)$ est donc une solution du problème C d'apprentissage de fonction linéaire à seuil nul et exemples négatifs pour l'instance $(x''_i, 0_{\mathbb{B}^n})$.

Inversement, supposons que $(x''_i, 0_{\mathbb{B}^n})$ soit une solution H, de représentation w, alors il est aisé de montrer que w est solution de (x'_i, y_i), instance du problème B

Finalement on a réduit le problème B (lorsque qu'aucun exemple n'est nul) au problème C, ce qui nous donne une réduction de A à C.

Question 11

L'idée est d'obtenir une complexité optimale en vue de celle demandée pour la question 14. Ce qui n'est pas le cas si on calcule la somme de la ligne 7 à chaque fois, comme dans ce premier exemple de solution.

```ocaml
let apprentissage (n : int) (x : vecteur array) : vecteur =
  let w = Array.make n 0. in
  let m = Array.length x in
  let fin = ref false in
  while not !fin do
    fin := true;
    for k = 0 to m - 1 do
      let sum = ref 0. in
      for i = 0 to n - 1 do
        sum := !sum +. (w.(i) *. x.(k).(i))
      done;
      if !sum >= 0. then (
        fin := false;
        for i = 0 to n - 1 do
          w.(i) <- w.(i) -. x.(k).(i)
        done)
    done
  done;
  w
```

Il nous faut donc mettre à jour cette somme rapidement. On va maintenir la valeur de chacune des N sommes $w \cdot x$, c'est à dire pour chaque $x \in X$. Ces sommes sont modifiées lorsque w est modifié. Remarquons alors que lorsqu'on fait une erreur sur l'exemple x_k, la nouvelle valeur de w est $w - x_k$ et donc la nouvelle valeur de $w \cdot x_j$, la j-ème somme que l'on maintient est

$$(w - x_k) \cdot x_j = w \cdot x_j - x_k \cdot x_j \tag{10}$$

Finalement, lorsqu'on fait une erreur sur x_k, il suffit de retirer $x_k \cdot x_j$ à notre j-ème valeur pour chaque $j \in [\![1, N]\!]$.

Pour améliorer la complexité, on peut précalculer chaque $x_k \cdot x_j$, dans un tableau `precalc`

```ocaml
let apprentissage_bis (n: int) (x: vecteur array) : vecteur =
  let w = Array.make n 0. in
  let m = Array.length x in
  let scal = Array.make m 0. in

  let precalc = Array.make_matrix m m 0. in
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      let s = ref 0 in
      for k = 0 to n - 1 do
        s := !s +. x.(i).(k) *. x.(j).(k)
      done;
      precalc.(i).(j) <- !s;
    done;
  done;

  let fin = ref false in
  while not(!fin) do
    fin := true
    for k = 0 to (m - 1) do
      if scal.(k) >= 0. then begin
        fin := false;
        for i = 0 to n-1 do
          w.(i) <- w.(i) -. x.(k).(i);
        done;
        for l = 0 to m-1 do
          scal.(l) <- scal.(l) -. precalc.(k).(l);
        done;
      end
    done;
  done;
  w
```

Question 12

Supposons que l'algorithme termine. La boucle `while` est donc terminée, et ainsi on sait que `!fin = true`. Cela veut donc dire que l'instruction `fin := false` dans la condition `!sum >= 0.`, ne s'est

jamais exécuté. On en déduit que pour chaque $k \in [\![1, N]\!]$, la somme `!sum` est strictement négative (à erreur sur les flottants près). Or la valeur de `!sum` à ce moment là est exactement

$$\sum_{i=1}^{n} w_i x_{k_i} \tag{11}$$

Comme ces valeurs sont strictement négatives, w est bien une représentation valide pour chaque exemple $(x_k, 0_{\mathbb{B}^n})$ et est donc une solution du problème.

Question 13

Supposons que w ne soit pas solution, on considère alors w', une représentation vérifiant $w'_i = w_i - x_i$, obtenue après une erreur sur l'exemple x. Ainsi on a les deux égalités

$$\begin{aligned} d(w', v) &= \sum_{i=1}^{n} (w'_i - v_i)^2 \\ &= \sum_{i=1}^{n} w_i'^2 + v_i^2 - 2w'_i v_i \\ &= \sum_{i=1}^{n} w_i^2 + x_i^2 - 2w_i x_i + v_i^2 - 2w_i v_i + 2x_i v_i \\ d(w, v) &= \sum_{i=1}^{n} w_i^2 + v_i^2 - 2v_i w_i \end{aligned} \tag{12}$$

Notons qu'on a $\sum_{i=1}^{n} w_i x_i \geq 0$ et $\sum_{i=1}^{n} v_i x_i \leq -n$ (car v est solution n-séparable et que les exemples sont négatifs). On en déduit que

$$\begin{aligned} d(w', v) - d(w, v) &= \sum_{i=1}^{n} x_i^2 - 2w_i x_i + 2x_i v_i \\ &= \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i w_i + 2\sum_{i=1}^{n} x_i v_i \\ &\leq n - 2n = -n \end{aligned} \tag{13}$$

On en déduit qu'à chaque erreur, $d(w, v)$ diminue d'au moins n. Comme $d(w, v)$ est par définition positif, il y a forcément un nombre fini d'erreur. Finalement, l'algorithme termine, et par la question 12, il est correct.

Question 14

> **Remarque 3**
>
> Je ne vois pas comment obtenir la complexité voulue

On considère cette fois v une solution 1-séparable, de poids W.

Commençons par remarquer que, comme v est une solution 1-séparable,

$$\sum_{i=1}^{n} w'_i v_i - \sum_{i=1}^{n} w_i v_i = -\sum_{i=1}^{n} x_i v_i \geq 1 \tag{14}$$

Ce qui veut dire que $w \cdot v$ augmente d'au moins 1 à chaque erreur. Or on a initialement, lorsque $w = 0_{\mathbb{R}^n}$, $w \cdot v = 0$. Donc par une récurrence immédiate, après k erreurs, on a

$$w \cdot v \geq k \tag{15}$$

Considérons maintenant $\|w\|^2$ à chaque étape. On a en effet

$$\begin{aligned}\|w'\|^2 &= \|w - x\|^2 \\ &= \|w\|^2 + \|x\|^2 - 2w \cdot x \\ &\leq \|w\|^2 + \|x\|^2 \\ &\leq \|w\|^2 + n\end{aligned} \tag{16}$$

Donc, à chaque étape $\|w\|^2$ augment au plus de n. On en déduit, comme initialement on a $\|w\|^2 = 0$, qu'après k étapes, on a

$$\|w\|^2 \leq kn \tag{17}$$

Il reste à remarquer que

$$\begin{aligned}w \cdot v &\leq \|w\| \cdot \|v\| \\ &\leq \|w\| \cdot \sqrt{\sum_{i=1}^{n} v_i^2} \\ &\leq \|w\| \cdot \sqrt{\sum_{i=1}^{n} W^2} \\ &= \|w\| \cdot \sqrt{n} \cdot W \\ &\leq \sqrt{k} \cdot n \cdot W\end{aligned} \tag{18}$$

Finalement,

$$k \leq \sqrt{k} \cdot n \cdot W \tag{19}$$

et donc

$$k \leq n^2 W^2 \tag{20}$$

Finalement, il y a au plus n^2W^2 erreurs.

Pour la complexité de la fonction. Commençons par compter le nombre de boucle `while`, chacune d'elle est causée par une (ou plusieurs) erreur. On en déduit qu'il y a au plus n^2W^2 telles boucles. Déterminons combien de fois on effectue les opérations coûteuses au sein de cette boucle.

- L'instruction `w.(i) <- w.(i) -. x.(k).(i);` est effectuée n fois (pour chaque i) à chaque fois qu'il y a une erreur. Cette opération contribue en $O(n^2W^2 \cdot n)$
- L'instruction `scal.(l) <- scal.(l) -. precalc.(k).(l);` est effectuée N fois (pour chaque l), à chaque fois qu'il y a une erreur. Cette opération est en $O(1)$, elle contribue en tout à hauteur de $O(n^2W^2 \cdot N)$

Il reste de plus le pré-calcul, en $O(N^2 \cdot n)$.

Finalement, la complexité de l'algorithme est en $O(n^2(n+N)W^2 + nN^2)$

Si on avait choisi l'ancienne version, alors au pire cas, il y a n^2W^2 tours de boucle `while`, qui chacun peuvent devoir calculer chaque produit scalaire, tous en temps $O(n)$, comme il y en a $O(N)$, on a une complexité finale de $O(n^2(N \times n)W^2)$

Partie II. D'un neurone à seuil à un réseau de neurones

Question 15

Les profondeurs de chaque sommet peuvent être observée dans le graphe en **Fig. 1**. Les 3 sommets gris correspondent sont les sorties du graphe, qui a ainsi une profondeur de 2.

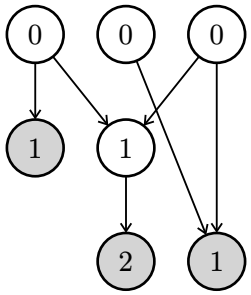

Fig. 1. – Profondeurs des sommets du graphe **Question II-3-15**

Ce circuit calcule la fonction f qui à $(x_1, x_2, x_3) \in \mathbb{B}^3$ associe le triplet $(\overline{x_1}, \overline{x_1 \vee x_3}, x_2 \wedge x_3)$

Partie 4. Des circuits booléens aux réseaux de neurones à seuil

Question 16

```ocaml
let creer (n : int) : reseau_adjacence = Array.make n ([], 0.)
```

```ocaml
let teste_arc (rs : reseau_adjacence) (s : int) (t : int) : bool =
  let successeurs, seuil = rs.(s) in
  List.exists (fun (v, _) -> v = t) successeurs
```

```ocaml
let ajoute_arc (rs : reseau_adjacence) (s : int) (t : int) (w : poids) :
    reseau_adjacence =
  let successeurs, seuil = rs.(s) in
  rs.(s) <- ((t, w) :: successeurs, seuil);
  rs
```

Question 17

D'après la question 1, les fonctions booléennes $\mathrm{AND}_2, \mathrm{NOT}_1, \mathrm{OR}_2$ sont des fonctions booléennes linéaires à seuil unitaires. Il suffit donc de remplacer dans ce graphe ces sommets par un neurone équivalent.

Il reste à traiter la cas des sommets $0, 1$ d'arité nulle. Pour 0, il suffit de considérer le neurone de représentation (1), qui n'a aucun poids et a un seuil égal à 1. On remarque alors que la somme correspondante $\sum_{i=1}^{0} w_i x_i$ vaut 0, qui est toujours inférieur au seuil. Donc ce neurone renvoie toujours 0, car il n'est pas activé.

On fabrique de même le sommet 1 avec un neurone de représentation (0)

Le graphe obtenu est bien un réseau de neurones à seuil à poids unitaires, de même taille, même profondeur, qui calcule la même fonction f que C.

Question 18

On considère un neurone qu'on va appeler id, de représentation $(1, 1)$. Celui-ci s'active si et seulement si son entrée vaut 1. Sa sortie est donc égale à son entrée.

On remarque alors que si on ajoute au milieu d'une arête un sommet intermédiaire id, en remplaçant $s \to t$ par $s \to \mathrm{id} \to t$, alors la fonction calculée par le graphe ne change pas. En effet, t est le seul sommet à changer d'entrée, mais il reçoit la même qu'avant, par définition de id. Donc le comportement global du réseau ne change pas.

Considérons alors un réseau N de neurones à seuil. On appelle alors désordre de N la valeur

$$d(N) = \sum_{(s,t)\in N} (\text{prof}(t) - \text{prof}(s) - 1) \tag{21}$$

qui est un entier positif, et nul si et seulement si N est bien formé. On construit alors N' équivalent à N, tel que $d(N') < d(N)$.

On considère pour cela une arête (s,t) de N tel que $\text{prof}(t) > \text{prof}(s) + 1$. On transforme alors l'arête $s \to t$ en $s \to \text{id} \to t$. Le sommet id a pour profondeur $\text{prof}(s) + 1$, car s est son seul prédécesseur. On en déduit que la profondeur de t ne change pas, et ainsi celle des autres sommets du graphe non plus. De plus, la fonction calculée par le sommet t ne change pas non plus. Nous n'avons par ailleurs pas créé de nouvelle entrée, donc N' est équivalent à N.

On a alors

$$\begin{aligned} d(N') - d(N) &= (\text{prof}(t) - \text{prof}(\text{id}) - 1) + (\text{prof}(\text{id}) - \text{prof}(s) - 1) - (\text{prof}(t) - \text{prof}(s) - 1) \\ &= -1 \end{aligned} \tag{22}$$

Finalement, comme cette construction diminue $d(N)$ de 1 et que $\mathbb{N}$ est bien fondé, on finit par obtenir un réseau N' équivalent à N tel que $d(N') = 0$ et donc bien formé.

Question 19

```ocaml
let rec eval (rs : reseau_matrice) (x : vecteur) : vecteur =
  match rs with
  | [] -> x
  | (mat, seuil) :: q ->
      let n = Array.length mat in
      let m = Array.length mat.(0) in
      let sortie = Array.make m 0. in
      for i = 0 to n - 1 do
        for j = 0 to m - 1 do
          sortie.(j) <- sortie.(j) +. (x.(i) *. mat.(i).(j))
        done
      done;
      for j = 0 to m - 1 do
        if sortie.(j) >= seuil.(j) then sortie.(j) <- 1. else sortie.(j) <- 0.
      done;
      eval q sortie
```

Partie 5. A propos des circuits booléens

Question 20

Soit C un tel circuit, et $b = (b_1, \ldots, b_n) \in \mathbb{B}^n$ tel que $\text{PARITY}(b_1, \ldots, b_n) = 1$. La seule sortie de C est une porte OR généralisée, dont tous les prédécesseurs sont des portes AND généralisées. Comme ce sommet OR renvoie 1, par définition de l'opération OR, on sait qu'il existe une des portes AND pointant vers celle-ci, appelée A_b, dont le résultat sur $(b_1, \ldots, b_n)$ est 1.

Question 21

Soit C, b, A_b considérés comme dans la question précédente. On définit, pour tout $i \in [\![1, n]\!]$, $b^{(i)} = \left(b_1, ..., b_{i-1}, \overline{b_i}, b_{i+1}, ..., b_n\right)$. On a alors $\forall 1 \leq i \leq n, \text{PARITY}\left(b^{(i)}\right) = 0$, car le nombre de bit à 1 a été modifié exactement de 1, ce qui le rend impair pour cette entrée.

Supposons alors qu'il existe $1 \leq i \leq n$ tel qu'il n'y ait pas de chemin de b_i à A_b. Cela veut dire que le résultat de A_b est indépendant de b_i et donc que $A_b\left(b^{(i)}\right) = 1$. Or cela veut dire que le circuit prend la valeur 1 sur l'entrée $b^{(i)}$ et donc que $\text{PARITY}\left(b^{(i)}\right) = 1$, ce qui est impossible.

On en déduit qu'il existe un chemin de toutes les entrées à A_b.

Question 22

Soit b une entrée acceptée par PARITY, la porte A_b est totalement déterminée par b. En effet, comme A_b renvoie 1 sur l'entrée b, on sait que les chemins de b_i à A_b renvoient tous 1. Si jamais $b_i = 1$, ce chemin est forcément direct ($b_i \to A_b$), et si $b_i = 0$, alors elle passe forcément par un NOT ($b_i \to \text{NOT} \to A_b$).

On en déduit que les portes $\{A_b \mid b \in \mathbb{B}^n, \text{PARITY}(b) = 1\}$ sont toutes distinctes. Or cet ensemble comprend

$$\sum_{\substack{0 \leq i \leq n \\ i \text{ pair}}} \binom{n}{i} = 2^{n-1} \tag{23}$$

éléments. Ainsi, il y a, en comptant la porte OR, au moins $2^{n-1} + 1$ portes dans le circuit.

Pour ce qui est du cas NOT — OR — AND, supposons qu'il existe un circuit de cette forme calculant PARITY avec moins de $2^{n-1} + 1$ portes. On considère la négation de ce circuit, qui est un circuit NOT — AND — OR. Ce circuit calcule la fonction IMPARITY, et possède moins de $2^{n-1} + 1$ portes AND et OR. On peut montrer que ceci est impossible avec les arguments des deux questions précédentes (exactement les mêmes).

Partie 6. A propos des réseaux de neurones à seuil à poids unitaires

Question 23

On vérifie facilement que le neurone $\left(\underbrace{1, 1, ..., 1}_{n}, m\right)$ accepte ses n entrées si et seulement si au moins m d'entre elles valent 1

De même, le neurone $\left(\underbrace{-1, -1, ..., -1}_{n}, -m\right)$ accepte ses n entrées si et seulement si au plus m d'entre elles valent 1.

Question 24

Soit $f : \mathbb{B}^n \to \mathbb{B}$ une fonction symétrique. Il existe $I \subseteq [\![0, n]\!]$ tel que $f(x_1, ..., x_n) = 1$ si et seulement si le nombre de x_i valant 1 est dans I.

Ceci est vrai car, si f accepte une entrée qui contient i valeurs à 1, alors elle accepte toutes les entrées vérifiant cette propriété.

Il suffit donc de considérer $2 \cdot |I|$ neurones sur la couche 1, qui pour chaque $i \in I$, déterminent si le nombre d'entrée à 1 est supérieur à i (resp. inférieur à i) selon la question précédente. Il ne nous reste plus qu'un neurone, car on a déjà une profondeur de 1.

Remarquons que si le nombre d'entrée à 1 est dans I, alors il y a $|I|+1$ neurones sur la couche 1 qui sont activés, sinon il n'y en a que $|I|$. Il suffit donc de tous les relier à un neurone avec un seuil de $|I|+1$.

Question 25

La fonction PARITY étant symétrique, elle est, d'après la question précédente, calculable par un réseau de neurones à poids unitaires, de profondeur 2 et de taille linéaire.

Partie III. Complexité de l'apprentissage d'un circuit

Question 26

Montrons qu'on peut vérifier qu'un certificat de solution est valide, ce en temps polynomial, et que s'il est valide, alors celui-ci est de taille polynomiale.

On se restreint pour l'instant au cas d'une seule tâche $(x_1, ..., x_n, y_1, ..., y_m)$ et d'une architecture $G = (V, E)$. Un certificat est alors la donnée, pour chaque $s \in V$, d'un seuil $h_s \in \mathbb{Z}$ et pour chaque $(s,t) \in E$, d'un poids $w_{s,t} \in \{-1, 0, 1\}$.

Étant donné une tâche, une architecture et un potentiel certificat, on peut commencer par déterminer si le certificat a le bon format. Ceci s'effectue en temps linéaire vis-à-vis de la taille de la tâche et de l'architecture. Un schéma de procédure pour cette vérification consisterait en :

- lire les seuils h_s, vérifier qu'ils sont entiers et qu'il n'apparaisse pas deux fois.
- lire les poids $(w_{s,t})$, vérifier qu'ils sont dans $\{-1, 0, 1\}$ et qu'il n'y a pas de doublon.

Une fois cela validé, on peut commencer à vérifier s'il s'agit bien d'une solution au problème. Pour cela, on peut successivement calculer les sorties de chaque sommet. Une procédure naïve, mais polynomiale, serait de, tant qu'il existe un sommet non calculé, en chercher un dont tous les prédécesseurs ont été calculés, puis calculer ce sommet. Et ce jusqu'à ce qu'on connaisse la sortie du réseau, on vérifie alors si elle est égale à $(y_1, ..., y_m)$, et si oui, on accepte l'entrée.

Cette procédure est bien en temps polynomial, et si jamais il y a plusieurs taches, il suffit de répéter la procédure précédente.

Question 27

Des poids et seuils solutions à ce problème sont donnés en Fig. 2. Ce réseau correspond à la fonction $\overline{x_1} \vee x_2 \vee \overline{x_3}$

En effet, le sommet c est ici un OR généralisé, et les sommets v_1, v_2, v_3 correspondent à NOT, id, NOT

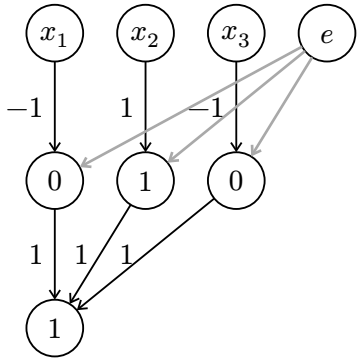

Fig. 2. – Solution à la question 27

Question 28

f_i est soit une fonction constante (0 ou 1), soit l'identité, soit la négation logique. Supposons que f_i soit une constante, on obtient que c, qui a en entrée v_1, v_2, v_3, est indépendant de v_i. Ceci est impossible comme le montre les exemples ci-dessous :

- c accepte $(0,0,1)$ mais pas $(1,0,1)$
- c accepte $(1,1,1)$ mais pas $(1,0,1)$
- c accepte $(1,0,0)$ mais pas $(1,0,1)$

On en déduit que pour $1 \leq i \leq 3$, f_i est soit l'identité, soit la négation.

Question 29

Une disjonction de cas un peu longue permet de conclure.

Question 30

Soit $\varphi = \bigcup_{i \in I} C_i$ une formule 3-CNF. On va créer une architecture permettant de savoir si φ est satisfiable ou non. Pour chaque variable x_k de φ, on ajoute une entrée de même nom et un sommet v_k, et pour chaque clause C_i on ajoute un sommet c_i, qu'on relie de manière similaire à ce que propose l'énoncé, et on rajoute les 8 tâches correspondantes à la clause C_i. Noter que l'on ajoute pas pour l'instant de sommet e. L'ajout de ces tâches est un premier problème, car on doit spécifier une valeur pour les c_i non concernés, ce problème peut se régler en ajoutant des entrée b_i (une par clause) reliée à c_i. Par exemple la tâches $(1,0,1,0,0)$ de la question 27 deviendrait une tâche fixant $x_1, x_2, x_3 = 1,0,1$, ainsi que $e = 0, y_i = 0, b_i = 0$, et pour tous les $j \neq i$, on aurait $b_j = 1, y_j = 1$. L'idée est qu'on active artificiellement les autres y_j avec les entrées b_j, pour ne s'occuper que de y_i, qui n'est pas influencé par b_i car on l'a fixé à 0.

On a alors comme premier résultat que si cette architecture est compatible avec toutes les tâches, alors c_i a bien comme valeur de sortie l'évaluation de la clause C_i, d'après la question 29. Il reste à savoir s'il existe une valuation des x_i qui active en même temps tous les sommets c_i.

On ajoute alors un sommet e, relié à tous les sommets v_k, ainsi que la tâche $(0,0,...,0,1,1,...,1)$, autrement dit, si on fixe tous les x_i à 0, les b_i à 0, e à 1, on veut avoir $\forall i \in I, y_i = 1$. Supposons alors que cette nouvelle architecture est compatible avec notre nouvel ensemble de tâches. Considérons alors une solution du problème, c'est à dire, fixons des poids/seuils qui permettent de valider chaque tâche donnée. Notons alors que les remarques précédentes sont toujours valides.

On en déduit qu'il existe une combinaison de valeurs des v_k, qui active tous les c_i, et donc que la valuation $(f^{-1}(v_1), ..., f^{-1}(v_n))$ satisfait chacun des clauses C_i et ainsi satisfait la formule φ.

Réciproquement, supposons que φ soit satisfiable par une valuation $\mathbb{I}$, on peut alors construire une solution à l'architecture donnée. on donne un poids de 1 à chaque arc $x_k \longrightarrow v_k$. Ensuite, pour chaque clause c_i, faisant intervenir des variables x_k, on choisit comme poids pour l'arc $v_k \longrightarrow c_i$ une valeur de 1 si x_k apparaît positivement, et -1 sinon. Pour le seuil de c_i, on choisit $(1-l)$ où l est le nombre de x_k apparaissant négativement dans C_i. On admettra que cela valide les 8 tâches données pour la clause C_i, on donne par ailleurs un exemple pour une clause $\text{NOT}(x_1) \vee x_2 \vee \text{NOT}(x_3)$ en **Fig. 3**.

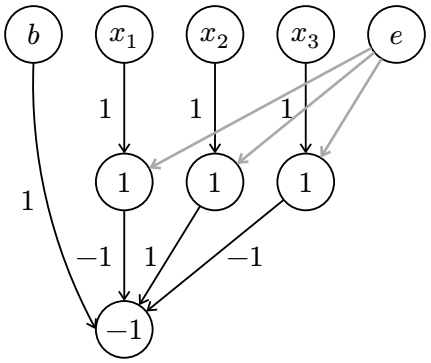

Fig. 3. – Solution alternative à la question 27

On met alors un poids de 1 pour les arcs $b_i \longrightarrow c_i$, ainsi on a l'implication $b_i = 1 \Longrightarrow c_i = 1$, qu'on peut aussi vérifier par une disjonction de cas.

Finalement, on choisit pour l'arc $e \longrightarrow v_k$ un poids de 1 si $x_k^{\mathbb{I}} = 1$ (la valeur de x_k dans la valuation solution de φ) et -1 sinon. On vérifie alors que la dernière tâche (toutes les entrées nulles sauf e, toutes les sorties activées) est vérifiée. En effet, on a pour cette tâche, $v_k = x_k^{\mathbb{I}}$, on peut alors vérifier qu'avec la construction des poids/seuils pour le sommet c_i, celui-ci est bien activé.

En effet, comme $\mathbb{I}$ valide φ, on sait que l'un des trois $x_k^{\mathbb{I}}$ a la bonne valeur vis-à-vis de la clause C_i. Rappelons alors qu'on a $v_k = x_k^{\mathbb{I}}$. Il y a alors deux cas :

- Soit x_k est positif dans la clause C_i, et donc $v_k \longrightarrow c_i$ a un poids de 1 (par construction). On vérifie alors que peu importe le signe des deux autres littéraux dans la clause, on a $c_i = 1$. Traitons un de ces cas pour donner l'idée : Si jamais les deux autres apparaissent négativement, alors le seuil de c_i est -1 et les deux autres arcs ont un poids de -1. Ainsi, la valeur entrante dans c_i est au moins de 1 (pour v_k) + (-2) (pour les deux arcs négatifs qui peuvent être activés). Or cette valeur est supérieure au seuil. On vérifie que si un des deux autres est négatifs, ou s'ils sont tous les deux positifs, cela marche aussi.
- Le cas où x_k est négatif est similaire.

Finalement, on a bien une solution au problème d'apprentissage. On en déduit que φ est satisfiable si et seulement si cette architecture possède une solution pour ces tâches, et comme cette entrée est de taille polynomiale en φ, le problème de l'apprentissage exact est NP-complet.

Partie IV. Réseaux de neurones récurrents et automates finis

Partie 7. Réseaux de neurones à seuil et reconnaissance immédiate

Question 31

Commençons par définir « cette fonction » (ce manque de clarté est partagé, à la vue du rapport de jury). Il s'agit bien entendu de la fonction de transition, qu'on cherche à encoder avec un réseau de neurones à $q + 1$ entrées et q sorties. L'entrée correspond à la donnée d'un e_i, par exemple $(0, 0, 0, 1, 0)$ pour le 4^e état d'un automate à 5 états, ainsi que d'une lettre $a \in \mathbb{B}$. La sortie, elle, correspond aussi à un e_i, et donc à un état.

Finalement, si $\delta(x_i, a) = x_j$, on veut que ce réseau, sur l'entrée (x_i, a) renvoie x_j. On dénotera le i-eme état de l'automate soit par x_i, soit par x_i', selon si on s'intéresse à un sommet d'entrée ou sortie du réseau. Noter cependant que dans l'automate, x_i et x_i' désignent le même état.

Pour ceci, on considère un sommet de sortie x'_j. Ce sommet doit renvoyer 1 si et seulement si il existe $a \in \mathbb{B}, i \in [\![0, q-1]\!]$ tel que $\delta(x_i, a) = x'_j$ et tel que x_i soit activé. On considère alors les ensembles

$$\begin{aligned} P_{j,0} &= \{0 \leq i < q \mid \delta(x_i, 0) = x'_j\} \\ P_{j,1} &= \{0 \leq i < q \mid \delta(x_i, 1) = x'_j\} \end{aligned} \tag{24}$$

On crée alors respectivement deux sommets OR généralisés, qu'on appelle $\mathrm{OR}_{j,0}$ et $\mathrm{OR}_{j,1}$ qui prennent en entrée les x_i pour $i \in P_{j,0}$ (respectivement $\in P_{j,1}$). Puis on crée deux nouveaux sommets, le premier prend en entrée $\mathrm{OR}_{j,0}$ et la négation du sommet d'entrée a, il s'active si et seulement si $\overline{a} \wedge \mathrm{OR}_{j,0}$. Inversement, l'autre sommet correspond à $a \wedge \mathrm{OR}_{j,1}$.
Finalement, on relie ces deux sommets à la sortie x'_j, qui s'active si l'une d'entre elles est active. Le sommet x'_j correspond donc à la formule $\big(a \wedge \mathrm{OR}_{j,1}\big) \vee \big(\overline{a} \wedge \mathrm{OR}_{j,0}\big)$. Donc, la sortie x'_j s'active si et seulement si il existe $a \in \mathbb{B}$ tel que il existe $0 \leq i < q$ tel que $\delta(x_i, a) = x'_j$ et x_i est actif.

On répète cette construction pour chaque sommet, on obtient alors le réseau de neurones voulu, avec une profondeur 3.

Si on considère l'automate en **Fig. 4**, alors on obtient le réseau de neurones dont une représentation simplifiée est donnée en **Fig. 5**.

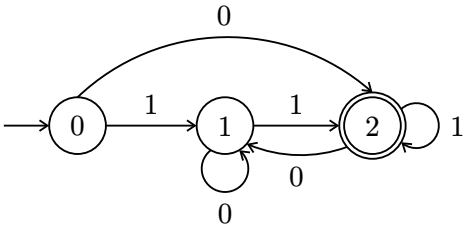

Fig. 4. – Un petit automate

Par souci de clarté, on a omis l'unique arête sortante de a, qui permet d'obtenir le sommet $\overline{a}$. De plus, ces deux sommets ont été dupliqués. Le chemin vers x'_0 a été omis car de toute manière, l'état initial n'a pas de transition entrante dans l'automate, et on a donc nécessairement $x'_0 = 0$

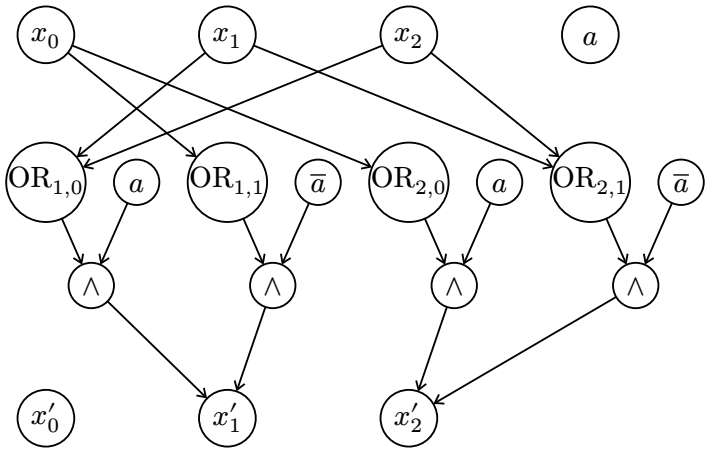

Fig. 5. – Réseau de neurones pour l'automate

Question 32

On peut appliquer une méthode similaire. Considérons le sommet de sortie $x'_{i,e}$, pour des certains $i \in [\![0, q-1]\!], e \in \mathbb{B}$, pour qu'il s'active, il faut dans un premier temps qu'on ait $a = e$, où a est le sommet d'entrée correspondant à la lettre lue. et il faut qu'un des sommets de l'ensemble $P_{i,e}$ soit

activé. Attention, on notera que par sommet de $P_{i,e}$, on entend, pour chaque $j \in P_{i,e}$, les deux sommets $x_{j,0}$ et $x_{j,1}$, car le nouvel état est indépendant de la lettre lue précédemment, comme elle ne dépend que de la nouvelle.

- Dans le cas de $x'_{i,1}$, on peut choisir un seuil de 2, et un poids de 1 pour les arêtes allant de $P_{i,1} \cup \{a\}$ à $x'_{i,1}$. Il s'active alors si et seulement si au moins 2 de ces sommets sont activés, or, comme au plus un seul sommet de $P_{i,1}$ est activé, il s'active seulement si a et un sommet de $P_{i,1}$ sont activés.
- Dans le cas de $x'_{i,0}$, on peut faire la même chose, avec un poids de -1 pour a, et un seuil de 1 au lieu de 2.

On obtient alors un réseau de neurones de profondeur 1. On peut voir une représentation sans poids ni seuil de l'automate **Fig. 4** en **Fig. 6**

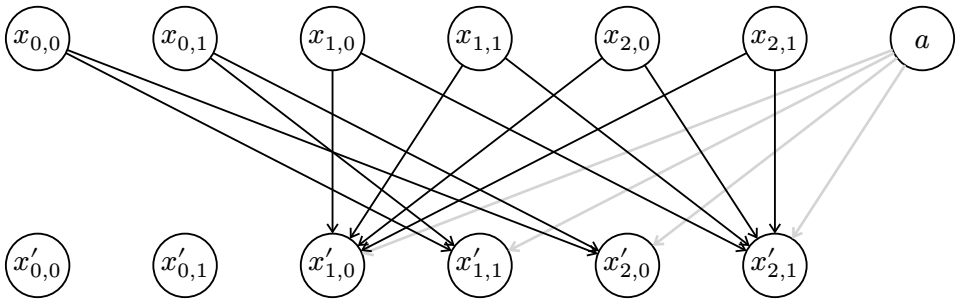

Fig. 6. – Réseau de neurones pour l'automate

Question 33

Soit L un langage régulier, on va construire un réseau de neurones reconnaissant le langage L. Pour ce faire, on utilise les sommets suivants:

- l'entrée $w(t)$
- des sommets représentant l'état courant, couplée à la dernière lettre lue $x_{1,0}(t), x_{1,1}(t), ..., x_{q-1,0}(t), x_{q-1,1}(t)$

On note $q(0), q(1), ..., q(l)$ la suite d'états obtenus lorsque l'on lit le mot w sur l'automate. On a $q(0) = x_0$.

On présente dans un premier temps le dispositif principal. Supposons que l'on ait déjà calculé $q(t)$, et qu'il soit stocké dans les états $x_{1,0}(t), x_{1,1}(t), ..., x_{q-1,0}(t), x_{q-1,1}(t)$. On connaît de plus la nouvelle lettre $w(t)$ qui permet de calculer la transition entre $q(t)$ et $q(t+1)$. Il nous reste plus qu'à calculer la fonction de transition, et pour cela, on utilise le réseau de neurones de la question précédente. Il faut cependant stocker le résultat dans les mêmes sommets, pour cela, on fait correspondre les sommets $x_{i,e}$ et $x'_{i,e}$ de la question précédente.

On a un unique état final x_i. Mais celui-ci est indépendant de la dernière lettre lue. On doit donc accepter le mot, si à l'instant l, on a $x_{i,0}$ ou $x_{i,1}$. On peut pour cela rajouter un sommet qui sera la disjonction des 2, et sera notre neurone de décision. Cependant, ce sommet ne vaudra la disjonction de ces deux derniers qu'un tour après, il doit donc prendre sa décision à l'instant $l+1$ et non l !

Un autre problème est l'état initial. Initialement, tous les neurones sont nuls, on doit donc créer un dispositif permettant d'avoir $x_{0,0} = 1$ lorsque $t = 1$ et uniquement lorsque $t = 1$. En effet, on en a besoin pour le premier calcul de fonction de transition, et comme l'état initial n'a pas d'arc entrant et qu'on veut garantir qu'un seul sommet ne soit activé à la fois, il faut le repasser à 0 ensuite. Pour

cela, on peut considérer que $x_{0,0}$ a un seuil nul, et a comme arc entrant un unique sommet, lui même, avec un poids de -1. Ainsi, la suite $\left(x_{0,0}(t)\right)_{t\in\mathbb{N}}$ sera la suite $0, 1, 0, 1, \ldots$ Maintenant, il faut faire en sorte que lorsque que sa valeur redevient 0, elle ne change plus. On peut pour cela créer un sommet z prenant les valeurs $0, 1, 1, \ldots$, avec un poids de 0 et un arc allant en lui même avec un poids de 1. On rajoute alors un arc de z à $x_{0,0}$ avec un poids de -1. Finalement, on a $x_{0,0}(t) = (t = 1)$

Un exemple pour l'automate **Fig. 4** se trouve en **Fig. 7**. Les arcs en pointillés viennent de la structure de la question précédente. Seulement certains d'entre eux ont été refait afin de représenter la fusion des sommets de cette structure. Les autres sommets sont ceux qui ont été décrits ci-dessus.

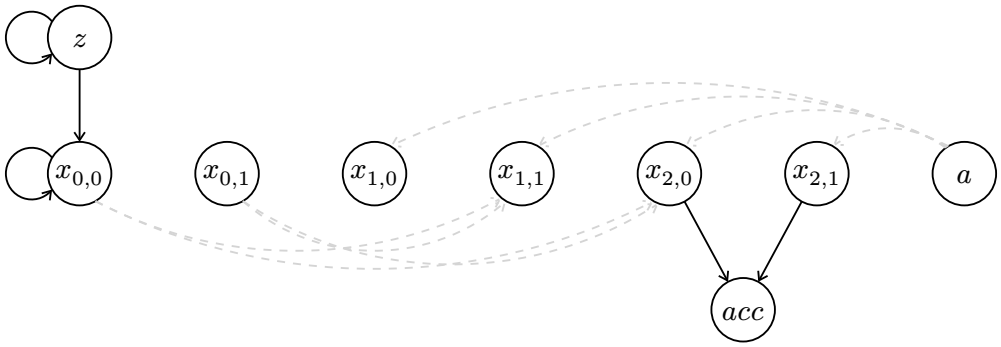

Fig. 7. – Réseau de neurones pour l'automate

Question 34

Soit L un langage accepté par un réseau de neurones $R = (V \cup X, E)$. On appelle état du réseau un élément de l'ensemble $\mathbb{B}^{V\cup X}$, il y a ainsi un nombre fini d'état possible du réseau R, au plus $2^{|V|+|X|}$, qui correspondent chacun à des valeurs renvoyées par chaque sommet du réseau. On appelle l'état initial l'élément $0_{\mathbb{B}^{V\cup X}}$, qui correspond à l'instant $t = 0$ du réseau, lorsque tous les neurones sont désactivés. Et on considère, en notant a le neurone de décision, l'ensemble des états finaux $F = \left\{q \in \mathbb{B}^{V\cup X} \mid q(a) = 1\right\}$.

Il nous reste à déterminer la fonction de transition, on a $\delta(q, e) = q'$ si et seulement si on a

$$\forall j, q'(j) = \mathcal{A}\left(\sum_{\substack{i\in V\cup X\\(i,j)\in E}} w_{i,j}q(i) - h_j\right) \tag{25}$$

Autrement dit, si à l'instant t, le réseau est dans l'état q, et la lettre e est lue, alors à l'instant $t + 1$, le réseau est dans l'état $\delta(q, e)$.

On construit alors un automate fini déterministe avec tous les composants qu'on vient de décrire. Cet automate reconnaît bien le langage L.

Partie 8. Réseaux de neurones à seuil et reconnaissance hors-ligne

Question 35

Premièrement, il est évident qu'un langage régulier est reconnu par un automate fini hors-ligne, il suffit de considérer le même automate (sans transitions utilisant \$).

Considérons alors un automate hors-ligne A reconnaissant un langage L. Soit F l'ensemble de ses états finaux. On considère alors F', l'ensemble des états q de A tel que, si on lit des \$ depuis l'état q dans A, on tombe éventuellement dans un état final. On appelle ces états les états éventuellement finaux, on a par ailleurs $F \subseteq F'$.

On construit alors A', un automate fini déterministe, étant exactement l'automate A, sans \$-transition, et dont les états finaux sont les états éventuellement finaux de A. Le langage reconnu par A' est L. En effet, $w \in L$ si et seulement si son chemin dans A' termine dans un état éventuellement final, et donc si et seulement si son chemin dans A termine dans un état final.

Question 36

Si un langage est régulier, d'après la question 33, il est reconnu par un réseau de neurone récurrent à seuil. On considère alors un tel réseau de neurone. On lui rajoute l'entrée V, qu'on ignore totalement. Et on fixe le neurone de calcul à 1, car, comme notre réseau de neurone reconnaît déjà le langage L sans être hors-ligne, on n'a pas besoin de calcul supplémentaire.

Partie 9. Réseaux de neurones linéaires saturés à coefficients entiers

Question 37

Remarquons que si les poids et seuils sont entiers, la valeur renvoyée par un neurone est toujours un entier, ce que l'on peut montrer par récurrence sur $t \in \mathbb{N}$

- Si $t = 0$, alors tous les neurones z renvoient une valeur nulle
- sinon, on suppose la propriété vérifiée au temps $t \in \mathbb{N}$. On a alors

$$z(t+1) = \mathcal{A}\left(\sum_{\substack{i \in V \cup X \\ (i,j) \in E}} w_{i,j} z_i(t) - h_j\right) \tag{26}$$

Par hypothèse de récurrence, on a que $z_i(t) \in \mathbb{Z}$, et on sait que $w_{i,j}, h_j \in \mathbb{Z}$, donc

$$\sum_{\substack{i \in V \cup X \\ (i,j) \in E}} w_{i,j} z_i(t) \in \mathbb{Z} \tag{27}$$

Or si son entrée est entière, la sigmoïde idéale renvoie un entier.

On sait maintenant que les neurones ne renvoient que des entiers, on remarque alors que σ, restreint au domaine $\mathbb{Z}$, coïncide avec la fonction de Heaviside, et donc que ces deux fonctions peuvent être interchangées sans rien modifier. Les équivalences demandées sont alors données par les questions précédentes.

Partie V. Réseaux de neurones récurrents ReLU

Question 38

Voici une liste d'instructions multipliant son entrée par 2 :

$\text{IsZero}(1,5,2)$	$\text{Inc}(2,3)$	$\text{Inc}(2,4)$	$\text{Decr}(1,1)$	$\text{IsZero}(2,0,6)$	$\text{Decr}(2,7)$	$\text{Inc}(1,5)$
1	2	3	4	5	6	7

L'idée est, tant que le compteur 1, est non nul, de le décrémenter et d'ajouter 2 au compteur 2. On arrive alors à l'état $(5, 0, 2n)$. D'ici, on décrémente le dernier compteur, en incrémentant le premier jusqu'à ce que le deuxième compteur soit nul. On arrive alors à $(0, 2n, 0)$.

Question 39

- Commençons par remarquer qu'un neurone ReLU ne peut pas calculer la fonction $x \mapsto x + 1$, car il renverrait -1 sur l'entrée -2, or un neurone ReLU ne renvoie que des valeurs positives. On

va donc chercher à calculer plutôt $\max(0, x+1)$. un poids de 1 sur l'entrée, et un seuil de -1 convient alors, car ce neurone renvoie $\mathcal{R}(1 \cdot x - (-1)) = \max(0, x+1)$.
- Pour la fonction $\max(0, x-1)$, un poids de 1 et un seuil de 1 convient.
- pour la fonction qui à x associe 1 si $x = 0$ et 0 sinon, on montre facilement que ce n'est pas possible. En effet, si on considère un neurone ReLU, sa fonction de sortie est $\mathcal{R}(w_1 x_1 + ... + w_n x_n - h)$, qui est une fonction continue en ses paramètres $x_1, ..., x_n$, donc la fonction voulue

Maintenant, faisons preuve de bonne foi, on cherche à calculer ces fonctions restreintes au domaine $\mathbb{N}$, car le but sera bien évidemment d'implémenter les instructions d'une machine à compteurs. Donc pour la dernière, un seuil de -1 et un poids de -1 convient, on a alors la fonction $x \mapsto \max(0, 1-x)$.

Question 40

Remarque 4

On utilise un compteur supplémentaire, ne voyant pas comment faire sans.

Pour simplifier, on suppose qu'on veut passer de $(1, r_1, r_2, r_3)$ à $(0, r_1, r_2, r_1 + r_2)$, on se donne un 4^e compteur r_4 dont la valeur nous importe peu, i.e. on ne garantit pas que sa valeur ne change pas après les opérations.

On commence par mettre r_3, r_4 à 0 avec les instructions Decr, IsZero. On a alors l'état $(\_, r_1, r_2, 0, 0)$. Puis, en décrémentant r_1, on l'ajoute dans le 3^e et 4^e compteur. On est alors dans l'état $(\_, 0, r_2, r_1, r_1)$. On vide alors le 4^e compteur dans le premier, on est dans l'état $(\_, r_1, r_2, r_1, 0)$. On fait les mêmes opérations pour r_2, on passe alors par les états $(\_, r_1, 0, r_1 + r_2, r_2)$ puis $(\_, r_1, r_2, r_1 + r_2, 0)$. Ce raisonnement se généralise à n'importe quel c, u, v, il est alors aisé, à la fin de ce processus, de se placer dans l'état l.

Question 41

On suppose avoir comme état de départ $(1, x_1, x_2, ..., x_n)$, on veut en sortie $(0, \mathcal{R}(w_1 x_1 + ... + w_n x_n - h), \_, ..., \_)$, c'est à dire avoir calculé la valeur voulue dans le premier registre.

Premièrement, on considère un registre de calcul supplémentaire r_{n+1}, qu'on met à 0. On suppose alors fixé les valeurs $w_i \in \{-1, 0, 1\}$ et $h \in \mathbb{Z}$. Pour chaque i tel que $w_i = 1$, on ajoute le registre i au registre r_{n+1}, avec l'instruction $\text{Add}(n+1, i, n+1, \_)$. Ensuite, si h est positif, on le soustrait au registre r_{n+1}, sinon, h est négatif et on ajoute alors $-h$ au registre r_{n+1}. Finalement, pour chaque i tel que $w_i = -1$, on soustrait le registre x_i au registre r_{n+1} avec l'instruction $\text{Sub}(n+1, n+1, i, \_)$. On vide alors le registre r_{n+1} dans le registre 1.

Attention, l'ordre des opérations est très important ! Cela ne marche pas si on s'occupe des w_i négatifs avant les w_i positifs. En effet, pour maintenir un invariant du type

$$r_{n+1} = \max\left(0, \sum_{i \text{ traité}} w_i x_i\right) \tag{28}$$

Il faut absolument traiter les additions avant les soustractions. Un exemple simple, où l'on calcule $-1+2$, si on applique la soustraction, puis l'addition à partir de 0, on obtiendra comme résultat 2, car après le -1, notre compteur contiendra $\max(0, -1) = 0$.

En fait, n'importe quel ordre d'opérations qui vérifie à tout instant

$$\sum_{i \text{ traité}} w_i x_i \geq \mathcal{R}(w_1 x_1 + ... + w_n x_n - h) \tag{29}$$

convient.

Question 42

Comme pour les automates, on peut coder le registre R par $q+1$ neurones appelés $R_0, ..., R_q$, tel que $R = i$ si et seulement si R_i est le seul neurone activé. On a aussi k neurones, qui eux maintiennent les valeurs des compteurs $r_1, ..., r_k$. Il suffit alors de coder les instructions. Pendant tout le processus, on maintiendra que les valeurs prises par les R_i sont dans $\{0, 1\}$, et qu'un seul de ces neurones ne peut être activé à la fois.

- Commençons par encoder les sauts d'instructions. Si jamais la i-eme instruction saute inconditionnellement à la j-eme, par exemple si cette instruction est $\text{Inc}(c, j)$, alors on ajoute un arc $R_i \xrightarrow[1]{} R_j$, et on fixe les seuils des $(R_k)_{0\leq k\leq q}$ à 0. Ainsi, si à l'instant t, on est à l'instruction R_i, alors on est à l'instruction R_j à l'instant $t+1$. On traitera le saut conditionnel plus tard.
- Ensuite, il faut mettre à jour les compteurs. Premièrement, lorsqu'ils ne sont pas modifiés, il faut maintenir leur valeur, on ajoute un arc $r_c \xrightarrow[1]{} r_c$, et on fixe le seuil des $(r_c)_{1\leq c\leq k}$ à 0.
  Lorsque ceux-ci sont modifiés, par exemple lorsque la i-eme instruction est $\text{Inc}(c, j)$, il faut incrémenter la valeur de r_c, ainsi, on veut que r_c augmente de 1 lorsque $R_i = 1$, pour cela on ajoute un arc $R_i \xrightarrow[1]{} r_c$. Comme un seul des R_k est activé à la fois, on est assuré qu'à l'instant prochain, r_c est incrémenté de 1.
  Dans le cas d'un Decr, on met juste un poids de -1 à la place de 1.
- Il reste alors le cas des sauts conditionnels. Si la l-eme instruction est $\text{IsZero}(c, i, j)$, alors il faut dans un premier temps implémenter le saut à i si $r_c = 0$.
  - On ajoute un neurone intermédiaire l_0, de seuil 0, et les arcs $R_l \xrightarrow[1]{} l_0, r_c \xrightarrow[-1]{} l_0, l_0 \xrightarrow[1]{} R_i$. Ainsi, l_0 renvoie 1 si $R_l = 1$, et que $r_c = 0$, sinon l_0 renvoie 0. R_i prend ensuite cette information à l'instant suivant, comme si elle venait de R_l.
  - Dans le cas du saut à R_j si $r_c \neq 0$, on peut créer un dispositif similaire, mais utilisant un neurone supplémentaire. On rajoute $l_1, \neg$ avec un seuil nul pour l_1 et un seuil de 1 pour $\neg$. On ajoute alors l'arc $r_c \xrightarrow[-1]{} \neg$, comme ça, $\neg$ vaut 1 si r_c est nul et 0 sinon. Ensuite, on ajoute $\neg \xrightarrow[-1]{} l_1, R_l \xrightarrow[1]{} l_1, l_1 \xrightarrow[1]{} R_j$. Ainsi, l_1 vaut 1 si et seulement si $r_c \neq 0$ et $R_l = 1$.

On a terminé notre dispositif, il faut maintenant vérifier la synchronisation des différentes composantes. Les seuls mécanismes qui peuvent tourner en parallèle sont les changements d'états de Inc/Decr ainsi que le changement de la valeur de r_c. Ces deux opérations se font en une étape. La seule étape plus longue est le cas de IsZero, auquel cas il faut 2 instants ou 3 selon la valeur de r_c pour passer à l'état suivant, ce qui est dépendant de l'état initial.

Finalement, on a un réseau fonctionnel dont le temps de calcul est au plus 3 fois plus long lors de l'exécution d'un IsZero, et tout aussi rapide pour un Inc/Decr.

On trouve en **Fig. 8** un exemple pour les instructions $\text{IsZero}(1, 0, 2)$;$\text{Inc}(1, 0)$. On a représenté les seuils dans les sommets, les poids différents de 1 sur les arêtes, ainsi que les noms des neurones. Les sommets dont le bord est doublé sont ceux qui représentent l'état actuel de la machine.

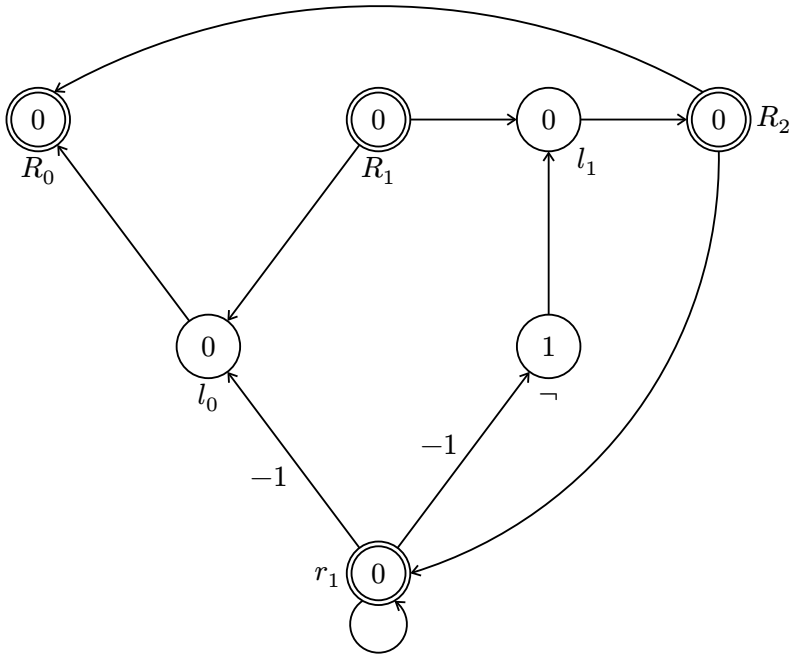

Fig. 8. – Exemple de réseau de neurones ReLU

Partie 10. Réseaux de neurones récurrents linéaires saturés

Question 43

Il suffit de remarquer que

$$\sigma(x) = \mathcal{R}(x) - \mathcal{R}(x-1) \tag{30}$$

On en déduit l'égalité plus générale :

$$\sigma\left(\sum_{i=1}^{n} w_i x_i - h\right) = \mathcal{R}\left(\mathcal{R}\left(\sum_{i=1}^{n} w_i x_i - h\right) - \mathcal{R}\left(\sum_{i=1}^{n} w_i x_i - h - 1\right)\right) \tag{31}$$

On peut donc facilement construire un réseau équivalent avec des neurones ReLU, utilisant seulement 3 fois plus de neurones, avec une profondeur exactement deux fois plus grande.

Question 44

On remarque que si r a un 0 en tête alors

$$\gamma(r) \leq \frac{1}{4} + \sum_{i=2}^{n} \frac{3}{4^i} < \frac{1}{4} + 3 \cdot \frac{1}{12} = \frac{1}{2} \tag{32}$$

De même, si r a un 1 en tête, alors on a

$$\gamma(r) \geq \frac{3}{4} \tag{33}$$

On cherche donc un neurone linéaire saturé qui renvoie 0 sur $[0, \frac{1}{2}[$, et 1 sur $\left[\frac{3}{4}, 1\right]$. Il suffit de considérer le neurone dont la valeur d'activation est $\sigma(4x - 2)$.

Pour la suite, il faut remarquer que le fait d'ajouter un élément en tête de r consiste en deux étapes pour modifier $\gamma(r)$. Premièrement, on décale tous les éléments, ce qui revient à diviser $\gamma(r)$ par 4. Puis on ajoute l'élément en tête, ce qui revient à ajouter $\frac{1}{4}$ pour un 0 et $\frac{3}{4}$ pour un 1. On en déduit les égalités

$$\begin{aligned}\gamma(push_0(r)) &= \frac{1}{4} + \frac{\gamma(r)}{4} \\ \gamma(push_1(r)) &= \frac{3}{4} + \frac{\gamma(r)}{4}\end{aligned} \tag{34}$$

Ces deux fonctions sont faciles à calculer avec des neurones linéaires saturés, car elles stabilisent l'intervalle $[0, 1]$ sur lequel on travaille. les fonctions $\sigma\left(\frac{x}{4} + \frac{1}{4}\right)$ et $\sigma\left(\frac{x}{4} + \frac{3}{4}\right)$ conviennent alors.

Il reste cependant le cas de *pop*, qui est plus complexe. Si jamais l'élément retiré est un 0, cela revient à retirer $\frac{1}{4}$ puis multiplier par 4. Si on retire un 1, alors cela revient à retirer $\frac{3}{4}$ et multiplier par 4. L'idée est la suivante : on réutilise la caractérisation qu'on a déterminer au début de la question, soit l'élément appartient à $\left[\frac{1}{4}, \frac{1}{2}\right]$ lorsqu'il a un 0 en tête, soit il appartient à $\left[\frac{3}{4}, 1\right]$.

Si on trace la fonction souhaitée, on se rend facilement compte que c'est impossible de la calculer avec un seul neurone linéaire saturé. En effet, on aurait

$$\begin{aligned}f\left(\frac{1}{4}\right) &= 0 \\ f\left(\frac{1}{2}\right) &= \frac{1}{4} \\ f\left(\frac{3}{4}\right) &= 0 \\ f(1) &= \frac{1}{4}\end{aligned} \tag{35}$$

Alors que $x \mapsto \sigma(a \cdot x + b)$ est forcément une fonction monotone sur $\mathbb{R}$. Si on s'autorise plusieurs neurones alors il suffit de calculer la fonction

$$x \mapsto \sigma\left(x - \frac{\sigma(4x - 2)}{2} - \frac{1}{4}\right) \tag{36}$$

qui peut être facilement construite grâce à deux neurones linéaires saturés. L'idée est qu'on veut dans tous les cas retirer $\frac{1}{4}$ à $\gamma(r)$, et si jamais $\gamma(r) \geq \frac{3}{4}$, on veut lui retirer $\frac{1}{2}$ supplémentaire.

Question 45

On va se restreindre à une machine de Turing avec une seule bande. On peut simuler cette bande avec deux piles, une première possédant les éléments à gauche de la tête de lecture, et ayant en tête l'élément devant la tête de lecture; et une deuxième possédant les éléments à droite de la tête de lecture, avec en tête l'élément juste à droite de la tête de lecture.

Pour chaque état q de la machine de Turing, on crée une instruction I_q. Les transitions de la machine de Turing depuis l'état q consiste en la lecture de la tête de lecture, c'est à dire la lecture de la tête de la première pile. Selon le résultat de cette lecture, on peut soit écrire un élément, soit déplacer la tête de lecture à gauche ou à droite.

Ainsi, I_q est une instruction de la forme $\mathrm{Top}(1, I_{q,0}, I_{q,1})$, où $I_{q,b}$ est une instruction correspondant à l'exécution de la machine dans l'état q lorsqu'on lit la lettre t.

- Dans le cas d'un déplacement à gauche, $I_{q,b}$ retire l'élément de tête de la pile de gauche, et le place en tête de la pile de droite. Il faut noter que Pop ne renvoie pas la valeur retirée, ainsi, en réalité, on regarde l'élément à gauche, selon sa valeur, on ajoute 0 ou 1 à droite, puis on retire l'élément à gauche. (un Top, un Pop, un Push_0, un Push_1)
- Dans le cas d'une écriture, on remplace l'élément en tête à gauche par celui voulu.

- Le cas du déplacement à droite est symétrique à celui à gauche, mais il faut gérer le cas où la pile de droite serait vide. On suppose alors que notre machine de Turing a initialement que des 0 sur sa bande, comme l'instruction Top renvoie 0 lorsque la pile est vide, c'est comme si il y avait une infinité de 0 au fond de notre pile, ce qui correspond au comportement de la bande dans la machine de Turing.

Bien sûr, le changement d'état de la machine s'encode facilement dans ces instructions, si le nouvel état est q', il suffit de faire en sorte que $I_{q,b}$ pointe sur l'instruction $I_{q'}$

On peut donc construire une machine à deux piles simulant une machine de Turing quelconque.

Question 46

Soit M une machine de Turing, on considère P une machine à deux piles équivalentes. On doit construire un réseau de neurones linéaires saturés simulant P. Pour cela, on va calculer les valeurs $\gamma(r_1), \gamma(r_2)$, qui sont des rationnels, à chaque étape de l'exécution de P.

On commence déjà par considérer deux neurones γ_1, γ_2. Comme à la question 42, on peut coder le registre R par $q+1$ neurones appelés $R_0, ..., R_q$, tel que $R = i$ si et seulement si R_i est le seul neurone activé, on maintiendra tout au long qu'un seul des R_i soit activé, et que sa valeur soit 1.

Comment gère-t-on la mise à jour de la valeur de γ_1 ? (et par symétrie γ_2). Pour chaque instruction numérotée i qui peut changer la valeur de γ_1, on crée un nouveau neurone calculant cette nouvelle valeur γ_1', ce qu'on sait faire d'après la question 44. Ensuite, on crée un nouveau neurone prenant en entrée γ_1' et R_i, et renvoyant γ_1' si $R_i = 1$ et 0 sinon, ce qu'on peut faire avec la fonction $\sigma(\gamma_1' - 1 + R_i)$.

Maintenant, on a des neurones renvoyant toutes 0 sauf une, correspondant à l'instruction actuelle, renvoyant γ_1'. Il faut noter que cela est vrai car pour chaque R_i, on crée au plus 1 tel dispositif, et tous les autres sont à 0, car les autres R_j ne sont pas activés. Ainsi la somme de ces valeurs vaut γ_1', on peut donc toutes les relier à notre neurone γ_1, avec un poids de 1 et un seuil de 0.

Il ne reste plus qu'à encoder les transitions entre les R_i. Pour les cas de Push et Pop, on peut réutiliser ce qu'on a fait en question 42. il reste alors le cas de $\text{Top}(c, i, j)$, on utilise le neurone qu'on a construit pour top en question 44. Celui-ci renvoie 1 si γ_c a un 1 en tête et 0 sinon, on peut alors utiliser sa sortie pour choisir la prochaine instruction. On ne va pas détailler les neurones à construire, mais un raisonnement similaire à la question 42 ou un agencement de porte AND et NOT font l'affaire.

Il faut alors s'arranger pour que le changement d'instruction et la mise à jour de γ_c soient synchronisés, on peut pour cela rajouter des neurones identité dans les arcs $R_i \longrightarrow R_j$ pour ralentir le processus.

Question 47

On appelle maintenant architecture toute architecture telle que définie dans le sujet, à la différence prêt qu'elle est adaptée au réseau de neurones récurrents. On définit la compatibilité de manière similaire, en imposant cette fois seulement les poids et seuils à être dans $\mathbb{Q}$ et en considérant le « réseau de neurones correspondants » comme un réseau linéaire saturé. Le problème de l'apprentissage exact garde alors le même énoncé. Il faut cependant modifier la notion de « sortie », ici on considère que $y_1, ..., y_m$ est la sortie d'un réseau de neurones récurrent si lorsqu'un certain neurone, appelé neurone de décision, est à 1, les sorties valent $y_1, ..., y_m$. Si ce neurone n'est jamais à 1, alors la tâche est par défaut, non compatible.

Supposons que ce problème soit décidable. On considère alors une machine de Turing M quelconque, on construit le réseau de neurones associés (cf Q46), avec comme sortie unique R_0, qui est aussi le

neurone de décision. On considère alors la tache $(0, 0, 1)$, c'est à dire, est-ce que R_0 s'active lorsqu'on a en entrée $\gamma_1 = \gamma_2 = 0$. Il faut noter que si jamais M s'arrête sur le mot vide, par construction de la question 46, cette tâche est compatible, car le mot vide est représenté par $\gamma(\emptyset) = 0$.

On en déduit que si la tâche n'est pas compatible, alors la machine M ne termine pas sur le mot vide, et donc le problème de l'arrêt sur le mot vide est co-semi-décidable. Or ce problème est semi-décidable, et tout problème à la fois semi-décidable et co-semi-décidable est décidable. Or on sait que ce problème est indécidable, ce qui est absurde. On en déduit que l'apprentissage exact des réseaux de neurones linéaires saturés est indécidable.

Question 48

Bonne question je sais pas mdr. Mais on peut essayer de voir du côté des machines universelles. Par exemple, on pourrait imaginer avoir une architecture et des poids/seuils sauf 1 tel que si on donne une certaine valeur à ce seuil/poids, ce réseau de neurone simule la machine de Turing dont cette valeur est l'encodage.