Réseau de neurones pour le Fanorona 3×3

Fondements mathématiques et implémentation avancée

Une approche généralisée de l'apprentissage automatique appliqué aux jeux traditionnels

MIT - L3

RAMAROSON Tojotiana Cyriaque

RANDRIANARISOA Tsantamirindra

RAKOTOARIJAONA Aro Mino Avotra

RANAIVO Ny Aina Peniala

Résumé

Ce document présente une analyse de l'architecture, de l'implémentation et des fondements mathématiques d'un réseau de neurones artificiels développé spécifiquement pour le jeu traditionnel malgache *Fanorona* sur plateau 3×3. L'étude couvre les aspects théoriques profonds incluant l'encodage vectoriel des états de jeu, la formulation des algorithmes de propagation avant et arrière, les justifications mathématiques des choix d'activation (sigmoid, softmax), d'initialisation (Xavier), et de fonction de perte (entropie croisée). Le rapport détaille également la génération synthétique des données d'entraînement et présente l'implémentation modulaire en Rust. L'architecture générique permet une configuration flexible des couches, utilisant la fonction sigmoïde pour les couches cachées et un système de double softmax (9+9 neurones) pour la couche de sortie, adapté à la structure décisionnelle du jeu.

Table des matières

Chapitre 1

Introduction et contexte

1.1 Objectifs et motivation

Le développement de systèmes d'intelligence artificielle capables de maîtriser des jeux complexes constitue un domaine de recherche fondamental en apprentissage automatique. Ce projet vise à concevoir, analyser théoriquement et implémenter un réseau de neurones capable d'imiter des stratégies optimales pour le jeu Fanorona sur un plateau réduit de dimensions 3×3.

L'approche adoptée privilégie une double perspective : d'une part, l'exposition rigoureuse des fondements mathématiques sous-jacents au modèle et aux algorithmes d'apprentissage ; d'autre part, la traduction pratique de ces formulations théoriques dans une implémentation performante en Rust.

1.2 Architecture générique et spécialisations

Le modèle développé présente une architecture générique paramétrable par une liste de tailles de couches, offrant ainsi une flexibilité maximale pour l'expérimentation. Dans l'usage pratique du projet :

- **Couches cachées** : activation par fonction sigmoïde $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Couche de sortie** : production de logits suivie de l'application de deux softmax séparés
- **Factorisation** : décomposition en deux têtes de prédiction (départ/arrivée) de 9 neurones chacune

Cette factorisation permet une adaptation naturelle à la problématique de sélection d'un coup (d, a) où d représente la case de départ et a la case d'arrivée.

Chapitre 2

Modélisation mathématique du jeu

2.1 Formalisation de l'espace d'états

2.1.1 Représentation discrète du plateau

Soit $\mathcal{P} = \{0, 1, 2, \ldots, 8\}$ l'ensemble des positions sur le plateau 3×3. Chaque case $i \in \mathcal{P}$ est caractérisée par un état s_i appartenant à l'ensemble discret :

L'espace d'états par case est défini comme :

$$\mathcal{S} = \{-2, -1, 0, 1, 2\}$$

où :

$$s_i = -2 \iff \text{pion du joueur A déjà déplacé en position } i \tag{2.1.1}$$
$$s_i = -1 \iff \text{pion du joueur A (non déplacé) en position } i \tag{2.1.2}$$
$$s_i = 0 \iff \text{case } i \text{ vide} \tag{2.1.3}$$
$$s_i = 1 \iff \text{pion du joueur B en position } i \tag{2.1.4}$$
$$s_i = 2 \iff \text{pion du joueur B déjà déplacé en position } i \tag{2.1.5}$$

L'état global du plateau est représenté par le vecteur $\mathbf{s} = (s_0, s_1, \ldots, s_8) \in \mathcal{S}^9$.

2.1.2 Formalisation de l'objectif d'apprentissage

Problème de prédiction optimale

Soit $\mathcal{X} = \mathcal{S}^9 \times \{-1, 1\}$ l'espace des configurations (état du plateau × joueur courant). L'objectif du réseau de neurones est d'apprendre la fonction :

$$f^* : \mathcal{X} \to \mathcal{P} \times \mathcal{P}$$

qui associe à chaque configuration (s, p) le coup optimal (d^*, a^*) selon une politique experte (ex : Minimax).

2.2 Encodage vectoriel et plongement

2.2.1 Transformation one-hot

L'encodage one-hot constitue une injection de l'espace discret $\mathcal{S}$ vers un sous-espace de $\mathbb{R}^5$:

Fonction d'encodage one-hot

$$\phi : \mathcal{S} \to \{0, 1\}^5$$

définie par :

$$\phi(s) = \begin{cases} [1,0,0,0,0]^T & \text{si } s = -2 \\ [0,1,0,0,0]^T & \text{si } s = -1 \\ [0,0,1,0,0]^T & \text{si } s = 0 \\ [0,0,0,1,0]^T & \text{si } s = 1 \\ [0,0,0,0,1]^T & \text{si } s = 2 \end{cases}$$

Propriétés de l'encodage one-hot

L'encodage one-hot ϕ satisfait les propriétés suivantes :

1. **Injectivité** : ϕ est une injection, i.e., $\phi(s_1) = \phi(s_2) \Rightarrow s_1 = s_2$
2. **Orthogonalité** : $\forall s_1 \neq s_2, \langle \phi(s_1), \phi(s_2) \rangle = 0$
3. **Normalisation** : $\forall s \in \mathcal{S}, \|\phi(s)\|_2 = 1$

2.2.2 Construction du vecteur d'entrée

Le vecteur d'entrée du réseau est construit par concaténation :

$$\mathbf{x} = \begin{bmatrix} \phi(s_0) \\ \phi(s_1) \\ \vdots \\ \phi(s_8) \\ p \end{bmatrix} \in \mathbb{R}^{46}$$

où $p \in \{-1, 1\}$ identifie le joueur courant.

Cette représentation évite l'introduction d'un ordre numérique arbitraire entre les différents états de cases, préservant ainsi la structure catégorielle naturelle du problème.

2.3 Génération synthétique des données

Étant donné l'absence de datasets publics pour le Fanorona 3×3, l'ensemble d'entraînement a été synthétisé algorithmiquement. Les étiquettes (d^*, a^*) sont générées par un oracle externe (algorithme Minimax, heuristique experte) appliqué aux configurations de jeu.

Cette approche de génération automatique présente l'avantage de produire un dataset équilibré et exhaustif, couvrant une large gamme de configurations possibles.

Chapitre 3

Architecture du réseau neuronal

3.1 Formalisme mathématique général

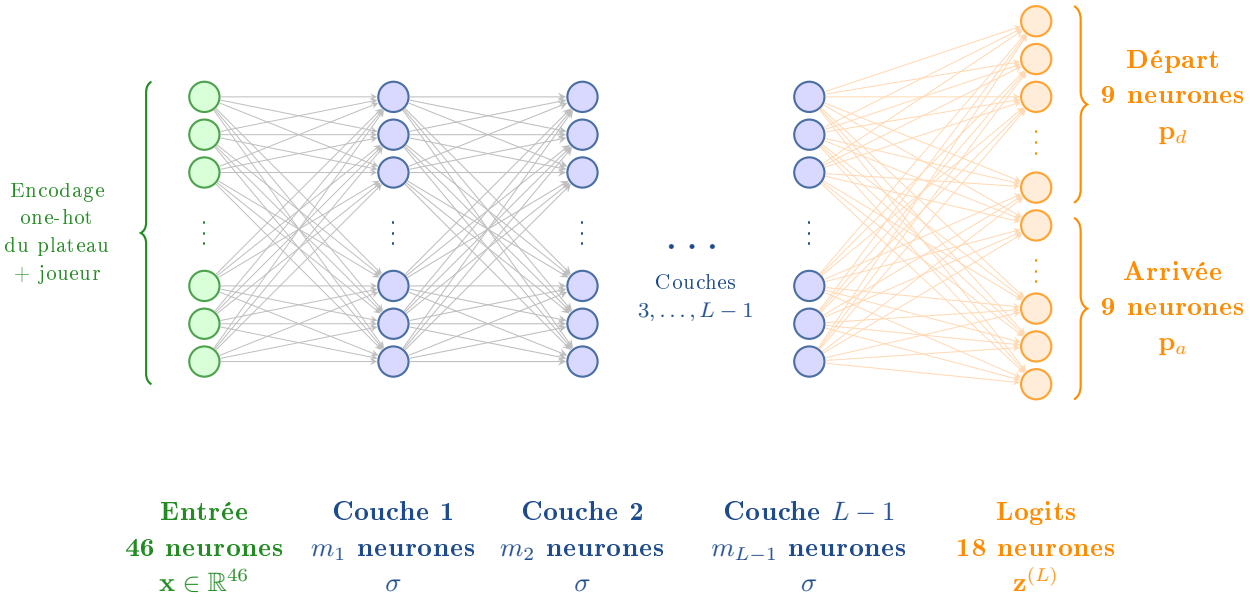
3.1.1 Notations et conventions

Soit $L \in \mathbb{N}^*$ le nombre total de couches (incluant la couche de sortie). Pour chaque couche $k \in \{1, 2, \ldots, L\}$:

- $W^{(k)} \in \mathbb{R}^{m_k \times n_k}$: matrice des poids de connexion
- $\mathbf{b}^{(k)} \in \mathbb{R}^{m_k}$: vecteur de biais
- $\mathbf{z}^{(k)} \in \mathbb{R}^{m_k}$: entrée nette (avant activation)
- $\mathbf{a}^{(k)} \in \mathbb{R}^{m_k}$: sortie activée
- m_k : nombre de neurones dans la couche k
- n_k : dimension de l'entrée de la couche k

avec la convention $\mathbf{a}^{(0)} = \mathbf{x}$ (vecteur d'entrée du réseau).

3.1.2 Illustration architecturale améliorée



3.1.3 Propagation avant : formulation mathématique

Algorithme de propagation avant

Pour chaque couche $k \in \{1, 2, \ldots, L\}$, le calcul de propagation avant s'effectue selon :

$$\mathbf{z}^{(k)} = W^{(k)}\mathbf{a}^{(k-1)} + \mathbf{b}^{(k)} \tag{3.1.1}$$

$$\mathbf{a}^{(k)} = \phi^{(k)}(\mathbf{z}^{(k)}) \tag{3.1.2}$$

où $\phi^{(k)}$ désigne la fonction d'activation de la couche k.

3.2 Fonctions d'activation et leurs propriétés

3.2.1 Fonction sigmoïde pour les couches cachées

Fonction sigmoïde

La fonction sigmoïde est définie par :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

avec pour dérivée :

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Propriétés de la sigmoïde

La fonction sigmoïde satisfait :

1. **Monotonie stricte** : σ est strictement croissante sur $\mathbb{R}$
2. **Bornes** : $\lim_{z\to-\infty}\sigma(z)=0$ et $\lim_{z\to+\infty}\sigma(z)=1$
3. **Point d'inflexion** : $\sigma(0)=\frac{1}{2}$
4. **Symétrie** : $\sigma(-z)=1-\sigma(z)$

3.2.2 Double softmax pour la couche de sortie

Pour la couche de sortie ($k=L$), les logits $\mathbf{z}^{(L)}\in\mathbb{R}^{18}$ sont partitionnés et traités par deux fonctions softmax séparées :

$$\mathbf{p}_d=\text{softmax}(\mathbf{z}^{(L)}_{0:8})=\left[\frac{e^{z_i^{(L)}}}{\sum_{j=0}^{8}e^{z_j^{(L)}}}\right]_{i=0}^{8} \tag{3.2.1}$$

$$\mathbf{p}_a=\text{softmax}(\mathbf{z}^{(L)}_{9:17})=\left[\frac{e^{z_i^{(L)}}}{\sum_{j=9}^{17}e^{z_j^{(L)}}}\right]_{i=9}^{17} \tag{3.2.2}$$

La sortie finale est : $\mathbf{a}^{(L)}=[\mathbf{p}_d,\mathbf{p}_a]\in\mathbb{R}^{18}$

Chapitre 4

Fonction de perte et optimisation

4.1 Construction de l'étiquette cible

Vecteur d'étiquetage

Pour un exemple d'entraînement avec étiquette $(d^*, a^*) \in \mathcal{P} \times \mathcal{P}$, le vecteur cible $\mathbf{y} \in \{0,1\}^{18}$ est défini par :

$$y_i = \begin{cases} 1 & \text{si } i = d^* \text{ ou } i = a^* \\ 0 & \text{sinon} \end{cases}$$

Cette construction encode simultanément la case de départ optimale et la case d'arrivée optimale dans un format compatible avec l'architecture de sortie.

4.2 Fonction de perte : entropie croisée

Fonction de perte composite

La fonction de perte utilisée combine les entropies croisées des deux têtes de prédiction :

$$\mathcal{L}(\mathbf{x}; d^*, a^*) = -\sum_{i=0}^{17} y_i \log a_i^{(L)} = -\log(p_d)_{d^*} - \log(p_a)_{a^*}$$

Cette formulation présente plusieurs avantages théoriques :

- **Convexité** par rapport aux paramètres de la dernière couche
- **Gradient non-nul** même pour des prédictions très confiantes mais erronées
- **Interprétation probabiliste** naturelle

4.3 Algorithme de rétropropagation

4.3.1 Gradient en sortie : propriété fondamentale

Gradient simplifié softmax avec entropie croisée

Pour la combinaison softmax avec entropie croisée, le gradient par rapport aux logits se simplifie remarquablement :

$$\delta_i^{(L)} \equiv \frac{\partial \mathcal{L}}{\partial z_i^{(L)}} = a_i^{(L)} - y_i, \quad \forall i \in \{0, 1, \ldots, 17\}$$

Démonstration. Sachant que la fonction d'activation softmax est définie par :

$$a_i = \frac{e^{z_i}}{\sum_{j=0}^{n_L} e^{z_j}}$$

et que la fonction de perte est :

$$\mathcal{L} = -\sum_j y_j \log(\mathrm{a}_j) = -\log\left(a_{d^\star}\right) - \log\left(a_{a^\star}\right)$$

On calcule :

$$\frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial}{\partial z_i}\Big(-\log(a_{d^\star}) - \log(a_{a^\star})\Big)$$

Pour $i \in \{0, 1, \ldots, 8\}$ (softmax départ)

Cas 1 : $i = d^\star$

$$\frac{\partial}{\partial z_{d^\star}}\big(-\log(a_{d^\star})\big) = -\frac{1}{a_{d^\star}} \cdot \frac{\partial a_{d^\star}}{\partial z_{d^\star}}$$

Or,

$$\frac{\partial a_{d^\star}}{\partial z_{d^\star}} = a_{d^\star}(1 - a_{d^\star})$$

Donc,

$$\frac{\partial}{\partial z_{d^\star}}\big(-\log(a_{d^\star})\big) = -\frac{1}{a_{d^\star}} \cdot a_{d^\star}(1 - a_{d^\star}) = -(1 - a_{d^\star}) = a_{d^\star} - 1$$

Cas 2 : $i \neq d^\star$

$$\frac{\partial}{\partial z_i}\big(-\log(a_{d^\star})\big) = -\frac{1}{a_{d^\star}} \cdot \frac{\partial a_{d^\star}}{\partial z_i}$$

Or,

$$\frac{\partial a_{d^\star}}{\partial z_i} = -a_{d^\star} a_i$$

Donc,

$$\frac{\partial}{\partial z_i}\big(-\log(a_{d^\star})\big) = -\frac{1}{a_{d^\star}} \cdot (-a_{d^\star} a_i) = a_i$$

De même pour $i \in \{9, 10, \ldots, 17\}$ (softmax arrivée)

Résultat général On obtient alors :

$$\frac{\partial \mathcal{L}}{\partial z_i} = a_i - \mathbf{1}_{\{i=d^\star\}} - \mathbf{1}_{\{i=a^\star\}}$$

où

$$\mathbf{1}_{\{i=d^\star\}} = \begin{cases} 1 & \text{si } i = d^\star \text{ ou } i = a^\star, \\ 0 & \text{sinon,} \end{cases}$$

D'où,

$$\boxed{\delta_i^{(L)} \equiv \frac{\partial \mathcal{L}}{\partial z_i^{(L)}} = a_i^{(L)} - y_i, \quad \forall i \in \{0, 1, \ldots, 17\}}$$

□

4.3.2 Rétropropagation dans les couches cachées

Formule de rétropropagation générale

Pour une couche cachée $k < L$ avec activation sigmoïde :

$$\delta_j^{(k-1)} = \Big(\sum_{i=1}^{n_k} \delta_i^{(k)} W_{ij}^{(k)}\Big) \cdot a_j^{(k-1)}\big(1 - a_j^{(k-1)}\big)$$

Démonstration. Soit une couche k du réseau. Pour tout neurone i de cette couche on pose :

$$z_i^{(k)} = \sum_{j=1}^{n_{k-1}} W_{ij}^{(k)}\, a_j^{(k-1)} + b_i^{(k)},$$
$$a_i^{(k)} = \sigma\big(z_i^{(k)}\big),$$

où $\sigma(z) = \dfrac{1}{1+e^{-z}}$ est la sigmoïde.

Nous voulons obtenir $\delta_j^{(k-1)} := \dfrac{\partial \mathcal{L}}{\partial z_j^{(k-1)}}$ en fonction des quantités connues en couche k.

Dérivée de la perte par rapport à l'activation $a_j^{(k-1)}$. Par la règle de la chaîne, $a_j^{(k-1)}$ n'apparaît que dans les pré-activations $z_i^{(k)}$ de la couche suivante, donc

$$\frac{\partial \mathcal{L}}{\partial a_j^{(k-1)}} = \sum_{i=1}^{n_k} \frac{\partial \mathcal{L}}{\partial z_i^{(k)}}\, \frac{\partial z_i^{(k)}}{\partial a_j^{(k-1)}}.$$

Or $z_i^{(k)} = \sum_t W_{it}^{(k)}\, a_t^{(k-1)} + b_i^{(k)}$, donc

$$\frac{\partial z_i^{(k)}}{\partial a_j^{(k-1)}} = W_{ij}^{(k)}.$$

D'où la formule indexée simple :

$$\frac{\partial \mathcal{L}}{\partial a_j^{(k-1)}} = \sum_{i=1}^{n_k} \delta_i^{(k)}\, W_{ij}^{(k)}$$
$$\text{où } \delta_i^{(k)} := \frac{\partial \mathcal{L}}{\partial z_i^{(k)}}.$$

Dérivée de la sigmoïde. Montrons la dérivée de σ :

$$\sigma(z) = \frac{1}{1+e^{-z}}, \qquad \frac{d\sigma}{dz} = \frac{e^{-z}}{(1+e^{-z})^2}.$$

Mais

$$1 - \sigma(z) = \frac{e^{-z}}{1+e^{-z}},$$

donc

$$\frac{d\sigma}{dz} = \sigma(z)\big(1-\sigma(z)\big).$$

Ainsi, en termes d'activations,

$$\sigma'\big(z_j^{(k-1)}\big) = a_j^{(k-1)}\big(1-a_j^{(k-1)}\big).$$

Passage de $\partial\mathcal{L}/\partial a_j^{(k-1)}$ à $\partial\mathcal{L}/\partial z_j^{(k-1)}$. On applique la chaîne sur $a_j^{(k-1)} = \sigma(z_j^{(k-1)})$:

$$\delta_j^{(k-1)} = \frac{\partial\mathcal{L}}{\partial z_j^{(k-1)}} = \frac{\partial\mathcal{L}}{\partial a_j^{(k-1)}} \cdot \frac{da_j^{(k-1)}}{dz_j^{(k-1)}} = \Big(\sum_{i=1}^{n_k} \delta_i^{(k)}\, W_{ij}^{(k)}\Big) \cdot \sigma'\big(z_j^{(k-1)}\big).$$

En remplaçant σ' par $a(1-a)$ on obtient la formule usuelle :

$$\boxed{\delta_j^{(k-1)} = \Big(\sum_{i=1}^{n_k} \delta_i^{(k)}\, W_{ij}^{(k)}\Big) \;\cdot\; a_j^{(k-1)}\big(1-a_j^{(k-1)}\big)}$$

□

4.3.3 Gradients des paramètres

Gradients des poids et biais

Les gradients des paramètres de chaque couche k sont donnés par :

$$\frac{\partial\mathcal{L}}{\partial W_{ij}^{(k)}} = \delta_i^{(k)} \cdot a_j^{(k-1)} \tag{4.3.1}$$

$$\frac{\partial\mathcal{L}}{\partial b_i^{(k)}} = \delta_i^{(k)} \tag{4.3.2}$$

Chapitre 5

Initialisation et stabilité numérique

5.1 Initialisation Xavier : fondements théoriques

Initialisation Xavier uniforme

Pour une couche avec n_{in} entrées et n_{out} sorties, l'initialisation Xavier uniforme tire les poids selon : $W_{ij} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right)$

Cette initialisation vise à maintenir la variance des activations approximativement constante à travers les couches, évitant ainsi les problèmes de gradient qui disparaît ou explose lors de l'entraînement de réseaux profonds.

5.1.1 Justification mathématique

L'objectif de l'initialisation Xavier est de satisfaire : $\text{Var}(a_i^{(k)}) \approx \text{Var}(a_j^{(k-1)}), \quad \forall k$

Pour une activation linéaire $z = \sum_j W_{ij} a_j$, sous l'hypothèse d'indépendance : $\text{Var}(z) = \sum_j W_{ij}^2 \text{Var}(a_j) \approx n_{\text{in}} \cdot \text{Var}(W) \cdot \text{Var}(a)$

Pour maintenir $\text{Var}(z) = \text{Var}(a)$, il faut : $n_{\text{in}} \cdot \text{Var}(W) = 1 \Rightarrow \text{Var}(W) = \frac{1}{n_{\text{in}}}$

L'initialisation Xavier généralise cette condition en considérant également la propagation arrière, d'où la moyenne harmonique $\frac{2}{n_{\text{in}}+n_{\text{out}}}$.

5.2 Algorithme de mise à jour des paramètres

5.2.1 Descente de gradient standard

Règle de mise à jour

À chaque itération t, les paramètres sont mis à jour selon :

$$W_{t+1}^{(k)} = W_t^{(k)} - \alpha \nabla_{W^{(k)}} \mathcal{L}_t \tag{5.2.1}$$

$$\mathbf{b}_{t+1}^{(k)} = \mathbf{b}_t^{(k)} - \alpha \nabla_{\mathbf{b}^{(k)}} \mathcal{L}_t \tag{5.2.2}$$

où $\alpha > 0$ est le taux d'apprentissage.

Chapitre 6

Sélection de coups et évaluation

6.1 Stratégie de décision composite

Score de coup

La décision finale combine les probabilités des deux têtes selon : $S(d, a) = (p_d)_d \cdot (p_a)_a$ pour tout coup candidat $(d, a) \in \mathcal{P} \times \mathcal{P}$.

Chapitre 7

Analyse de performance et validation

7.1 Métriques d'évaluation

7.1.1 Précision globale

$$\text{Précision}_{\text{coup}} = \frac{\text{Nombre de coups } (d,a) \text{ entièrement corrects}}{\text{Nombre total d'exemples}}$$

7.2 Validation croisée et overfitting

Étant donné la nature synthétique des données, une attention particulière doit être portée à :

- La diversité des configurations générées
- L'équilibre entre les différentes phases de jeu
- La validation sur des positions non vues pendant l'entraînement

Chapitre 8

Implémentation et aspects techniques

8.1 Architecture modulaire en Rust

L'implémentation est structurée en modules spécialisés :

- `init.rs` : Initialisation des paramètres et configuration du réseau
- `learn.rs` : Algorithmes d'entraînement et optimisation
- `prediction.rs` : Interface de prédiction et évaluation
- `nn.rs` : Noyau du réseau neuronal et opérations matricielles

8.2 Optimisations numériques

8.2.1 Stabilité du softmax

Pour éviter les débordements numériques, l'implémentation utilise la forme stabilisée : $\text{softmax}(z_i) = \frac{e^{z_i - z_{\max}}}{\sum_j e^{z_j - z_{\max}}}$ où $z_{\max} = \max_j z_j$.

8.2.2 Gestion de la précision

L'utilisation de types flottants double précision (`f64`) est recommandée pour maintenir la stabilité numérique lors d'entraînements prolongés.

Annexe A

Dérivations mathématiques complètes

A.1 Récapitulatif des notations

— $\mathbf{a}^{(0)} = \mathbf{x} \in \mathbb{R}^{n_0}$: vecteur d'entrée (avec $n_0 = 46$)
— Pour $k = 1, \ldots, L$: couche k de dimension m_k
— Couche de sortie : $m_L = 18$
— $\mathbf{z}^{(k)} = W^{(k)}\mathbf{a}^{(k-1)} + \mathbf{b}^{(k)}$, $\mathbf{a}^{(k)} = \phi^{(k)}(\mathbf{z}^{(k)})$
— Étiquette : $\mathbf{y} \in \{0,1\}^{18}$ avec deux composantes égales à 1 aux indices d^* et $9 + a^*$
— $\boldsymbol{\delta}^{(k)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(k)}} \in \mathbb{R}^{m_k}$

A.2 Dérivées explicites pour la couche de sortie ($k = L$)

Gradient de sortie détaillé

Pour $i \in \{0, 1, \ldots, 17\}$: $\delta_i^{(L)} = a_i^{(L)} - y_i$
Les gradients des paramètres de la couche de sortie sont :

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(L)}} = \delta_i^{(L)} \cdot a_j^{(L-1)}, \quad i \in \{0, \ldots, 17\},\ j \in \{0, \ldots, m_{L-1} - 1\} \tag{A.2.1}$$

$$\frac{\partial \mathcal{L}}{\partial b_i^{(L)}} = \delta_i^{(L)} \tag{A.2.2}$$

A.3 Dérivées pour les couches cachées ($k < L$)

Rétropropagation complète

Pour $k \in \{L-1, L-2, \ldots, 1\}$: $\delta_i^{(k)} = \left(\sum_{j=0}^{m_{k+1}-1} W_{ji}^{(k+1)} \delta_j^{(k+1)}\right) \cdot a_i^{(k)}(1 - a_i^{(k)})$
Les gradients des paramètres sont :

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(k)}} = \delta_i^{(k)} \cdot a_j^{(k-1)} \tag{A.3.1}$$

$$\frac{\partial \mathcal{L}}{\partial b_i^{(k)}} = \delta_i^{(k)} \tag{A.3.2}$$

Annexe B

Extensions et perspectives

B.1 Améliorations possibles

B.1.1 Techniques d'optimisation avancées

— **Adam optimizer** : adaptation automatique du taux d'apprentissage
— **Batch normalization** : normalisation des activations entre couches
— **Dropout** : régularisation par masquage aléatoire de neurones

B.1.2 Architectures alternatives

— **Réseaux convolutifs** : exploitation de la structure spatiale du plateau
— **Réseaux résiduels** : connexions directes pour faciliter l'entraînement profond
— **Attention mechanisms** : focus adaptatif sur les régions importantes du plateau

Annexe C

Conclusion

Ce document a présenté une analyse exhaustive de l'architecture et de l'implémentation d'un réseau de neurones pour le jeu Fanorona 3×3. L'approche développée combine une fondation mathématique rigoureuse avec une implémentation efficace, démontrant l'applicabilité des techniques d'apprentissage profond aux jeux traditionnels.

Les contributions principales incluent :

1. Une formulation mathématique complète des algorithmes de propagation
2. Une architecture modulaire et extensible
3. Une analyse détaillée de la stabilité numérique et des choix d'implémentation
4. Des perspectives d'extension vers des problèmes plus complexes

L'architecture générique proposée offre une base solide pour l'exploration de variantes plus sophistiquées et l'application à d'autres jeux de stratégie combinatoire.