Institut National des Sciences Appliquées et des Technologies

UNIVERSITE DE CARTHAGE

STAGE INGÉNIEUR

Génie Logiciel

RN quantifié pour un contexte deep learning embarqué

Auteur:

Rami ZOUARI

2021/2022

Résumé

Les réseaux de neurones sont devenus très utilisés dans l'intelligence artificielle grâce à leurs performances de prédictions.
Ces réseaux de neurones sont parfois très complexes, et il n'est pas pratique de les utiliser dans des systèmes à ressources limités.

Pour cela, dans ce rapport nous allons introduire les réseaux de neurones binaires (BNNs), les formaliser, étudier quelques modèles binarisés et dériver leurs formules, implémenter les résultats trouvés dans une bibliothèque qu'on va nommer **binaryflow**.
Et finalement, nous allons implémenter et entraîner des modèles binarisés pour la classification et régression, et nous allons comparer les performances de prédiction, ainsi que les performances temps et mémoire de ces BNNs et leurs contrepart classique.

Remerciement

Je veux remercier mes encadreurs:

- Mme. Meriem JAÏDANE
- Mme. Yousra BEN JEMÂA
- Mme. Rim AMARA BOUJEMAA
- Mr. Nader MECHERGUI

ainsi que toutes l'équipe de **dB Sense**, qui m'ont donné la chance de travailler sur ce projet intéressant, et leur expertise a été extrêmement précieuse dans la formulation des questions de recherche et de la méthodologie
Vos commentaires perspicaces m'ont poussés à affiner ma réflexion et ont fait passer le travail au niveau supérieur.
J'ai eu la chance de quitter ma zone de confort en travaillant sur ce sujet original, et je m'en suis très reconnaissants.

Contenu

Liste des Figures

Liste des Tableaux

Introduction

Avec l'explosion de l'intelligence artificielle, et surtout les modèles d'apprentissage profonds, la complexité des modèles a subi une croissance considérable, qui les rend inexploitable dans les systèmes à complexité limitée.

Dans ce rapport, nous allons étudier la quantification des paramètres et des entrées des couches du réseau de neurones sur un seul bit.

Ce rapport va détailler notre approche de la formalisation des BNNs, de l'analyse et généralisation des approches existantes, vers l'implémentation d'une bibliothèque unifiant les BNNs, et son utilisation. Il est composé de 6 chapitres.

Dans le premier chapitre, nous allons présenter la societé **dB Sense** et sa méthodologie.

Dans le deuxième chapitre, nous allons donner une petite histoire de l'apprentissage profond, et puis poser le problème de la grande complexitée de ces modèle, en posant la binarisation comme une solution.

Dans le troisième chapitre, nous allons formaliser notre approche, en définissant les BNNs. Après nous allons poser quelques problèmes dans l'entraînement. Après nous allons proposer les optimisations temps et mémoire qu'on peut exploiter avec les BNNs.
Finalement, nous allons proposer l'algorithme d'entraînement et d'interférence des BNNs.

Dans le quatrième chapitre, nous allons étudier et analyser quelques BNNs répandus dans la littérature, en conformant avec notre définition proposée.

Dans le cinquième chapitre, nous allons implémenter la bibliothèque **binaryflow**, en justifiant les paradigmes utilisés.

Dans le sixième chapitre, nous allons utiliser les différents BNNs étudiés sur 3 jeux de données.
Pour chacun de ces modèles nous allons analyser son performance de prédiction, sa taille mémoire au déploiement, et une estimation sur la complexité de son interférence en calculant le nombres d'instruction équivalents.

Chapter 1

Cadre du Stage

Introduction

Chapter 2

Analysis & Implementation

2.1 Introduction

2.2 Formalisation

To define a Mean Payoff Game, we will start by formalising a weighted di-graph[1].

2.2.1 Di-Graph

A Weighted Di-Graph $\mathcal{G}$ is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{W})$ where:

- $\mathcal{V}$ is the set of vertices.
- $\mathcal{E} \subseteq V \times V$ is the set of edges.
- $\mathcal{W} : \mathcal{E} \to \mathcal{R}$ is the weight function, assigning a weight for every edge, with $\mathcal{R}$ some set representing the weight.

2.2.2 Mean Payoff Game

Formally, a **Mean Payoff Graph** is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{W}, \mathcal{P}, s, p)$ where:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is a di-graph.
- $s \in \mathcal{V}$ denotes the starting position.
- $\mathcal{P} = \{\text{Max}, \text{Min}\}$ is the set of players.
- $p \in \mathcal{P}$ the starting player

A **Mean Payoff Game** is a perfect information, zero-sum, turn based game played indefinitively on a Mean Payoff Graph as follow:

- The game starts at $u_0 = s$, with player $p_0 = p$ starting.
- For each $n \in \mathbb{N}$, Player p_n will choose a vertex $u_{n+1} \in \operatorname{Adj} u_n$, with a payoff $w_n \mathcal{W}(u_n, u_{n+1})$
- The winner of the game will be determined by the Mean Payoff. There are different winning conditions.

[1] Directed Graph

Condition C1

Player Max wins **iff**:

$$\liminf_{n\in\mathbb{N}^*} \frac{1}{n}\sum_{k=0}^{n-1} w_k \geqslant 0$$

Otherwise, Player Min will win

Condition C2

Player Max wins **iff**:

$$\liminf_{n\in\mathbb{N}^*} \frac{1}{n}\sum_{k=0}^{n-1} w_k > 0$$

Otherwise, Player Min will win.

Condition C3

Player Max wins **if**:

$$\liminf_{n\in\mathbb{N}^*} \frac{1}{n}\sum_{k=0}^{n-1} w_k > 0$$

Player Min wins **if**:

$$\limsup_{n\in\mathbb{N}^*} \frac{1}{n}\sum_{k=0}^{n-1} w_k < 0$$

Otherwise, it is a draw.

2.2.3 Well Foundness

It is not very clear from the definition that the game is well founded.
In fact, there are choices for which the mean payoff does not converge. That is the sequence $\left(\frac{1}{n}\sum_{k=0}^{n-1} w_k\right)_{n\in\mathbb{N}^*}$ does not converge.
One such example is the sequence defined by:

$$w_n = (-1)^{\lfloor \log_2(n+1)\rfloor}$$

Rami ZOUARI
INSAT, Tunisia

For that sequence, the (2^r-1)-step mean payoff is equal to:

$$\begin{aligned}
\sum_{k=0}^{2^r-2} w_k &= \sum_{k=1}^{2^r-1} (-1)^{\lfloor \log_2(k) \rfloor} \\
&= \sum_{i=0}^{r-1} \sum_{j=2^i}^{2^{i+1}-1} (-1)^{\lfloor \log_2(j) \rfloor} \\
&= \sum_{i=0}^{r-1} \sum_{j=2^i}^{2^{i+1}-1} (-1)^i \\
&= \sum_{i=0}^{r-1} (2^{i+1}-2^i)(-1^i) \\
&= \sum_{i=0}^{r-1} (-2)^i \\
&= \frac{1-(-2)^r}{3} \\
\implies \quad \frac{1}{2^r-1} \sum_{k=0}^{2^r-2} w_k &= \frac{1}{3} \cdot \frac{1-(-2)^r}{2^r-1} \\
&= \frac{1}{3} \cdot \frac{2^{-r}-(-1)^r}{1-2^{-r}}
\end{aligned}$$

That sequence has two accumulation points $\pm\frac{1}{3}$, and thus, it does not converge.

On the other hand, the introduction of the supremum and infimum operators will solve the convergence problem, as the resulting sequences will become monotone.

An example of an execution that gives a rise to such payoffs is the following game: With the following pair of strategies[2]:

2.2.4 Symmetries

2.2.5 Strategy

Let p be a player.
A (deterministic) strategy[3] is a function $\Pi_p : \mathcal{V} \to \mathcal{V}$ such that:

$$\forall v \in \mathcal{V}, \Pi_p(v) \in \operatorname{Adj} v$$

A probabilistic strategy $\Phi_p : \mathcal{V} \to \mathscr{P}(V)$ is a random process that assigns for each vertex $v \in \mathcal{V}$ a probability distribution over $\operatorname{Adj} v$. This constitutes the most general strategy of a player.

2.3 Evaluating Strategies

Suppose we have a pair of potentially probabilitic strategies $(\Phi^{\text{Max}}, \Phi^{\text{Min}})$. The problem is to evaluate the winner without doing an infinite simulation of the game.

[2]Note that the proposed pair of strategies is odd in the sense that it appears that both players cooperated on the construction of non-convergent mean payoffs instead of trying ot win the game.

[3]By default, we refer to deterministic strategies. If the strategy is not deterministic, we will explicit it.

Rami ZOUARI
INSAT, Tunisia

2.3.1 Deterministic Strategies

If both strategies are deterministic. Then the generated sequence of vertices $(s_n)_{n\in\mathbb{N}}$ will be completely determined by the recurrence relation:

$$s_n = \begin{cases} s & \text{if } n = 0 \\ \Phi^{\text{Max}}(s_{n-1}) & \text{if } n \text{ is odd} \\ \Phi^{\text{Min}}(s_{n-1}) & \text{otherwise} \end{cases}$$

This can be represented in the compact form:

$$\forall n \in \mathbb{N}^*, \quad (s_n, p_n) = (\Phi^{p_{n-1}}(s_{n-1}), \bar{p}_{n-1}) = F(s_{n-1}, p_{n-1})$$

Since $\mathcal{V} \times \mathcal{P}$ is a finite set and F is a function, such sequence will be eventually periodic, that is:

$$\exists N \in \mathbb{N}, \exists T \in \mathbb{N}^* / \quad \forall n \in \mathbb{N}_{\geqslant N}, \quad (s_n, p_n) = (s_{n+T}, p_{n+T})$$

We can calculate its eventual period using the turtle hare algorithm.

Now, the mean payoff will be equal to the mean of weights that appears on the cycle. This can be proven as follow.:

$$\begin{aligned} S_{aT+b+N} &= \sum_{k=0}^{aT+b+N-1} w_k \\ &= \sum_{k=0}^{N-1} w_k + \sum_{k=0}^{aT+b-1} w_{k+N} \\ &= \sum_{k=0}^{N-1} w_k + a\sum_{r=0}^{T-1} w_{r+N} + \sum_{r=0}^{b-1} w_{r+N} \\ \implies \left| S_{n+N} - \lfloor \frac{n}{T} \rfloor \sum_{r=0}^{T} w_{k+N} \right| &\leqslant (N+T-1) \max_{(u,v)\mathcal{E}} |\mathcal{W}(u,v)| \\ &\leqslant (N+T-1)\|\mathcal{W}\|_\infty \\ \implies \left| \frac{1}{n+N} S_{n+N} - \frac{1}{n+N} \cdot \lfloor \frac{n}{T} \rfloor \sum_{r=0}^{T-1} w_{k+N} \right| &\leqslant \frac{N+T-1}{n+N} \|\mathcal{W}\|_\infty \end{aligned}$$

Now it can be proven that:

$$\lim_{n\to+\infty} \frac{1}{n+N} \cdot \lfloor \frac{n}{T} \rfloor \sum_{r=0}^{T} w_{k+N} = \frac{1}{T} \sum_{r=0}^{T-1} w_{r+N}$$

With that:

$$\lim_{n\to+\infty} \frac{1}{n} \sum_{k=0}^{n-1} w_k = \frac{1}{T} \sum_{r=0}^{T-1} w_{r+N} \quad \blacksquare$$

This calculation leads to the following $\mathcal{O}(|V|)$ algorithm that calculates the mean payoff of deterministic pairs of strategies:

2.3.2 Probabilistic Strategies

Due to the undeterministic nature of probabilistic strategies, it does not make sense to evaluate the mean payoffs, as different executions may lead to different mean payoffs.
Instead, probabilistic strategies gives rise to a discrete distribution of mean payoffs.
Now two closely related, but different evaluations are possible

- Expected Mean Payoff
- Distribution of winners

Now, with both strategies fixed. A Mean Payoff Game can be considered as a Markov Chain.

2.4 Countering Strategies

Chapter 3

Dataset Generation

3.1 Introduction

3.2 Analysis

Generating a Mean Payoff Game can be decomposed into two subsequent objectives.

1. Generate the Graph itself.
2. Generate the Weights

3.3 Graph Distributions

There are many well studied graph distributions in the litterature.
One of the most explored ones are the $\mathcal{G}(n, p)$ and $\mathcal{G}(n, m)$ families.

3.3.1 $\mathcal{G}(n, p)$ Family

For $n \in \mathbb{N}, p \in [0, 1]$, a graph G is said to follow a $\mathcal{G}(n, p)$ distribution if $|V| = n$ and:

$$\forall e \in \mathscr{E}, \quad \mathscr{P}(s \in \mathcal{E}) = p$$

Where $\mathscr{E}$ is a set of valid edges.

3.3.2 $\mathcal{G}(n, m)$ Family

For $n \in \mathbb{N}, m \in \mathbb{N}$, a graph G is said to follow a $\mathcal{G}(n, m)$ distribution if $|V| = n, |E| = m$ and the edges $e_1, \dots, e_m$ were drawn from a set of valid edges $\mathscr{E}$.

3.3.3 Valid edges

The set of valid edges $\mathscr{E}$ is the set defining the potential edges of the graph. It is equal to:

1. $V \times V$ for directed graphs with loops
2. $(V \times V) \backslash V \odot V$ for directed graphs without loops
3. The set of subsets of size 2 of V denoted $\mathscr{P}_2(V)$ for undirected graphs with loops.
4. The set of subsets of size 2 of V denoted $\mathscr{P}_2(V)$ for undirected graphs with loops.

3.3.4 $\mathcal{D}(n,p)$ Graph Construction

Naive Method

The definition of $\mathcal{D}(n,p)$ gives a straightforward construction.
This is achieved by flipping a coin[1] for each pair of node $(u,v) \in V^2$, we add an edge if we get a Head.
This is implemented in the following algorithm:

Algorithm 1 $\mathcal{D}(n,p)$ Graph Generation

Require: $n \in \mathbb{N}^*$ the size of the graph
Require: $p \in \mathbb{N}^*$ the edge probability
Ensure: $G \sim \mathcal{D}(n,p)$
  $A : (u,v) \in V \times V \to 0$
  for $u \in V$ **do**
    for $v \in V$ **do**
      Generate $X \sim \mathcal{B}(p)$ ▷ $\mathcal{B}(p)$ is the bernoulli distribution
      $A(u,v) \leftarrow X$
    end for
  end for
  return $G \leftarrow$ `GraphFromAdjacencyMatrix`(A)

The complexity[2] of the following algorithm is $\mathcal{O}(n^2)$.

Optimized Method

Instead of iterating over all possible pair of nodes. For each vertex $v \in V$:

- We can sample a number d from the outgoing degree distribution[3]
- We then choose d numbers uniformly without replacement from an indexable representation of V

The following algorithm implements the optimized method:

Algorithm 2 $\mathcal{D}(n,p)$ Graph Generation Optimisation

Require: $n \in \mathbb{N}^*$ the size of the graph
Require: $p \in \mathbb{N}^*$ the edge probability
Ensure: $G \sim \mathcal{D}(n,p)$
  $A : u \in V \to \varnothing$
  for $u \in V$ **do**
    Generate $d \sim \mathcal{B}(n,p)$ ▷ d represents the degree, $\mathcal{B}(n,p)$ is the binomial distribution
    $A(u) \leftarrow \text{choice}(V,d)$
  end for
  return $G \leftarrow$ `GraphFromAdjacencyList`(A)

[1] The coin is potentially biased with a probability of obtaining head equal to $p \in [0,1]$
[2] We assume the cost of generating a Bernoulli random variable as $\mathcal{O}(1)$
[3] Or the ingoing degree distribution, they are in fact equal.

Now, Let $C(n,m)$ be the cost of choice function. The expected complexity of this algorithm will be:

$$\tilde{\mathcal{O}}\left(n\mathbb{E}_d[C(n,d)]\right) \quad \text{where } d \sim \mathcal{B}(n,p)$$

We will show on the next section what choice function should we use.

3.3.5 Choice Function

First Proposition

We propose here a simple choice algorithm, but it is still efficient for our use case.
It works simply by drawing without replacement, but we ignore duplicate elements. This is implemented as follow

Algorithm 3 $\mathcal{D}(n,p)$ Choice without replacement

Require: S a list
Require: $m \in \{0, \dots |S|\}$ the number of chosen elements
Ensure: H a set of size m containing uniformly drawn elements without replacement.

```
H ← ∅
while |H| < m do
    Generate v ~ U(S)            ▷ Where U(S) is the uniform distribution over S
    H ← H ∪ {v}
end while
return H
```

To estimate the cost of this algorithm, we will use probabilistic reasoning.
Let $X_{n,m} = C(n,m)$ the running time of an execution of algorithm 3 in a set S of size n, with m elements to be chosen. We have:

$$\begin{aligned}
X_{n,0} &\text{ is deterministic} \\
X_{n,0} &= \mathcal{O}(1) \\
\mathbb{E}[X_{n,m}] &= 1 + \frac{1}{n}\sum_{k=0}^{n-1} \mathbb{E}[X_{n,m} \mid \text{The last drawn number is } k] \\
&= 1 + \frac{1}{n}\sum_{k=0}^{m-2} \mathbb{E}[X_{n,m}] + \frac{1}{n}\sum_{k=m-1}^{n-1} \mathbb{E}[X_{n,m-1}] \\
&= 1 + \frac{m-1}{n}\mathbb{E}[X_{n,m}] + \frac{n-m+1}{n}\mathbb{E}[X_{n,m-1}]
\end{aligned}$$

Now we arrived at a recurrent formula. We will simplify it as shown below:

$$\begin{aligned}
\frac{n-m+1}{n}\mathbb{E}[X_{n,m}] &= \frac{n-m+1}{n}\mathbb{E}[X_{n,m-1}] + 1 \\
\implies \mathbb{E}[X_{n,m}] &= \frac{n-m+1}{n-m+1}\mathbb{E}[X_{n,m-1}] + \frac{n}{n-m+1} \\
&= \mathbb{E}[X_{n,m-1}] + \frac{n}{n-m+1} \\
&= \sum_{k=1}^{m} \frac{n}{n-k+1} + \mathcal{O}(1) \\
&= \sum_{k=0}^{m-1} \frac{n}{n-k} + \mathcal{O}(1) \\
&= n \sum_{k=n-m+1}^{n} \frac{1}{k} + \mathcal{O}(1) \\
&= n(H_n - H_{n-m}) + \mathcal{O}(1)
\end{aligned}$$

Here $(H)_{n\in\mathbb{N}^*}$ is the harmonic series, and we define $H_0 = 0$.

Complexity

The expected complexity of algorithm 3 depends on both n and m:

- If $m = o(n)$, then it is $\tilde{\mathcal{O}}(m)$.
- If $m = kn + o(n)$ with $k \in]0,1[$, then it is $\tilde{\mathcal{O}}(m)$.
- If $m = n - o(n)$, It is[4] $\tilde{\mathcal{O}}(m \log m)$.

To prove this result, we use a well known asymptotic approximation of the Harmonic series:

$$H_n = \ln n + \gamma - \frac{1}{2n} + \mathcal{O}\left(\frac{1}{n^2}\right)$$

We can prove this claim as follow:

$$\begin{aligned}
m = o(n), \quad \mathbb{E}[C(n,m)] &= -n\ln\left(1-\frac{m}{n}\right) - \frac{1}{2}\left(1 - \frac{n}{n-m}\right) + \mathcal{O}\left(\frac{1}{n}\right) \\
&= m + o(m) \\
&= \mathcal{O}(m) \\
m = km + o(m), k\in]0,1[, \quad \mathbb{E}[C(n,m)] &= -n\ln\left(1-\frac{m}{n}\right) - \frac{1}{2}\left(1 - \frac{n}{n-m}\right) + \mathcal{O}\left(\frac{1}{n}\right) \\
&= -n\ln(1-k+o(1)) + \frac{1}{n}(1 - \tfrac{1}{1-k+o(1)}) + \mathcal{O}(\tfrac{1}{n}) \\
&= \mathcal{O}(m)
\end{aligned}$$

[4]Here we use the minus sign to emphasize that $m \leqslant n$

For $m = n - o(n)$, we prove it by noting that:

$$\begin{aligned}\mathbb{E}[C(n,m)] &\leqslant \mathbb{E}[C(n,n)] \\ &\leqslant nH_n \\ &\leqslant n\ln n + \gamma n + \frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right) \\ &= \mathcal{O}(m\log m)\end{aligned}$$

Refinement

If m tends to n, it is more hard to select m elements from a set of size n without replacement. This explains the extra logarithmic factor.
In that case, we can instead focus on the dual problem: "Find the $n - m$ elements that will not be selected". This can be calculated in $\mathcal{O}(n - m)$.
Once we find the elements that will not be selected, their set complement are exactly the m elements that will be selected. This new algorithm is guaranteed to be $\mathcal{O}(m)$ irrespective of n and m

Algorithm 4 Fine tuned $\mathcal{D}(n,p)$ Choice without replacement

Require: S a list
Require: $m \in \{0, \dots |S|\}$ the number of chosen elements
Require: choice The choice function defined on algorithm 3
Require: Threshold τ
Ensure: H a set of size m containing uniformly drawn elements without replacement.
  if $\frac{m}{|S|} \leqslant \tau$ **then**
    $H \leftarrow \text{choice}(V, n)$
  else
    $H \leftarrow V \setminus \text{choice}(S, n - m)$
  end if
  return H

3.3.6 Complexity of Optimised $\mathcal{D}(n,p)$ Graph Construction

We return to evaluate the asymptotic behaviour of $\mathbb{E}_d[C(n,d)]$.
Let $\delta \in \mathbb{R}_+^*$
The Chebychev Inequality implies that:

$$\mathscr{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - p\right| \geqslant \frac{\delta}{n^2} p(1-p)\right) \leqslant \frac{1}{\delta^2}$$

By setting: $\delta = p^{-1}$, we have:

$$\mathscr{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - p\right| \geqslant \frac{1-p}{n^2}\right) \leqslant p^2$$

We have:

$$\begin{aligned}
\mathbb{E}\left[C(n,d)\right] &\leqslant \mathbb{E}\left[C(n,d) \mid d \in [np - \tfrac{1-p}{n}, np + \tfrac{1-p}{n}]\right] + p^2 C(n,n)\\
&\leqslant C(n, np + \tfrac{1-p}{n}) + p^2 C(n,n) \quad \text{for large enough } n\\
C(n, np + \tfrac{1-p}{n}) &= \mathcal{O}(np + \tfrac{1-p}{n})\\
&= \mathcal{O}(np)\\
C(n,n) &= \mathcal{O}(n) \quad \text{With the refined choice algorithm}
\end{aligned}$$

Now, by combining both estimations, we get:

$$\tilde{\mathcal{O}}(n\mathbb{E}_d[C(n,d)]) = \tilde{\mathcal{O}}(n^2 p) \quad \blacksquare$$

3.3.7 $\mathcal{D}(n,m)$ Construction

To construct a random $\mathcal{D}(n,m)$ graph, we only have to select m uniformly random elements from the set $V \times V$.
We will use algorithm 4 for this purpose[5]:

Algorithm 5 Fine tuned $\mathcal{D}(n,p)$ Choice without replacement

Require: $n \in \mathbb{N}^*$
Require: $m \in \{0, \dots, n^2\}$ the number of chosen elements
Ensure: $G \sim \mathcal{D}(n,m)$
  $E \leftarrow \text{choice}(\text{Lazy}(V) \times \text{Lazy}(V), m)$ ▷ We only need the m elements on-demand.
  return $G \leftarrow \text{GraphFromEdges}(E)$ ▷ This justifies using Lazy

Here $\text{Lazy}(V) \times \text{Lazy}(V)$ is a lazy implementation of cartesian product that supports bijective indexing[6] over $\{0, \dots, n^2 - 1\}$.
The complexity of this construction is: $\mathcal{O}(m)$

3.4 Sinkless Conditionning

Sampling from a graph distribution may lead to graphs that have at least one sink.
These graphs are problematic as Mean Payoff Graphs are exactly the sinkless graphs.
To migitate this, we will impose a conditionning on both distribution that will gives a guaranteed Mean Payoff Graph.
We will explore such conditionning both distribution:

- $\mathcal{G}^S(n,p)$: This is the distribution of graphs following $\mathcal{G}(n,p)$ with the requirement that they do not have a sink.
- $\mathcal{G}^S(n,m)$: This is the distribution of graphs following $\mathcal{G}(n,m)$ with the requirement that they do not have a sink.

[5] It is essential that the list $V \times V$ be lazy loaded. In particular, each element will only be loaded when it is indexed. This is essential to reduce the complexity. Otherwise, we will be stuck in an $\mathcal{O}(n^2)$ algorithm.
[6] Indexing is required for uniform sampling

3.4.1 Repeating Construction

Algorithm

This method is very intuitive. It will repeat the sampling until getting the desired graph.
The following is an implemention of the repeating construction.

Algorithm 6 Fine tuned $\mathcal{D}(n,p)$ Choice without replacement

Require: $n \in \mathbb{N}^*$
Require: $m \in \{0, \dots |S|\}$ the number of chosen elements
Require: choice The choice function defined on algorithm 3
Require: Threshold τ
Ensure: H a set of size m containing uniformly drawn elements without replacement.
  if $\frac{m}{|S|} \leqslant \tau$ **then**
    $H \leftarrow \text{choice}(V, n)$
  else
    $H \leftarrow V \setminus \text{choice}(S, n-m)$
  end if
  return H

Analysis

We will analyse the runtime of generating a $\mathcal{G}^S(n,p)$.
We expect a similar runtime for $\mathcal{G}^S(n,m)$ due to the similarity between $\mathcal{G}(n,m)$ and $\mathcal{G}(n,p)$.
Let $F(n)$

3.5 Weights Distribution

3.5.1 Construction

Once the graph is constructed. We only have to generate the weights.
This will be done by creating a random weight function:

$$W(u,v) : (u,v) \rightarrow W_{u,v}$$

Here $W_{u,v}$ will be a sequence of real random variables.
In our case, we set $(W_{u,v})_{(u,v)\in E}$ to be independent and identically distributed over a real distribution $\mathcal{W}$.

3.6 Proposed MPG Distribution

3.6.1 Desired Properties of Mean Payoff Game Distributions

Fairness in the Limit

This is essential, as we intend to generate a sequence of Mean Payoff Games that do not favour statistically a certain player, in the sense that, if we generate sufficient independent and identically distributed Mean Payoff Games $G_1, \dots, G_n$, we expect the following:

$$\lim_{n \to +\infty} |\mathtt{R}_{\text{Max}}(G_1, \dots, G_n) - \mathtt{R}_{\text{Min}}(G_1, \dots, G_n)| = 0$$

Where $\mathtt{R}$ is defined as follow:

$$\mathtt{R}_{\text{Op}}(G_1, \dots, G_n) = \frac{1}{n} \sum_{i=1}^{n} \mathscr{P}(\text{Op wins } G_i \text{ assuming optimal strategies})$$

Symmetric

A real distribution is said to be symmetric if:

$$\forall [a,b] \in \mathbb{R}, X \sim \mathcal{W}, \quad \mathscr{P}(X \in [a,b]) = \mathscr{P}(X \in [-b,-a])$$

We will define a symmetric Mean Payoff Game distribution as a distribution of Mean Payoff Game whose weights are independent and identically distributed on a symmetric real distribution. This property is stronger than Fairness in the Limit, as it implies that:

$$\mathscr{P}(\text{Max wins } G \text{ assuming optimal strategies}) = \mathscr{P}(\text{Min wins } G \text{ assuming optimal strategies})$$

We will require a Symmetric Mean Payoff Game as we do not want a player to have an inherit advantage other the other one[7]

3.6.2 Implemented Distributions

The following table resumes the implemented distributions:

Distribution Family	Parameters	Type
$\mathcal{D}(n,p)$	• n : Graph size • p : Edge probability	Graph distribution
$\mathcal{D}(n,m)$	• n : Graph size • m : Number of edges	Graph distrbiution
$\mathcal{U}_{\text{discrete}}(-r,r)$	• r : The radius of the support	Weight distribution
$\mathcal{U}(-r,r)$	• r : The radius of the support	Weight distribution
$\mathcal{N}(0,\sigma)$	• σ : The standard deviation	Weight distribution

Table 3.1: Le tableau d'avancement des BNNs

[7] Other than the first move.

3.7 MPG Generation

3.7.1 Distribution

- Each generated graph will follow a distribution $\mathcal{G}(n, p(n))$ for some $n \in \mathbb{N}^*$
- The weights will follow the discrete uniform distribution $\mathcal{D}(-1000, 1000)$

We will generate two kinds of datasets, depending on the nature of the graph

3.7.2 Dense Graphs

- Let $\mathcal{P} = \{0.1, 0.2, 0.3, 0.5, 0.7, 0.8, 0.9, 1\}$
- $\mathcal{N} = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 200, 250, 300, 400, 500\}$
- For each $(n, p) \in \mathcal{N} \times \mathcal{P}$, we will generate $K = 1000$ observations $G_1^{n,p}, \dots, G_K^{n,p} \sim \mathcal{G}(n, p)$

The total number of examples is:

$$K \times |\mathcal{N}| \times |\mathcal{P}| = 160000$$

The generation was done on a 'haswell64' partition with 24 cores. and it took 02:12:38 hours.

3.8 Annotation

Now, non-polynomial algorithms are known for Mean Payoff Games.
The algorithm that we followed is based on **CSPs**.If the weights are represented as integers, it is exponential on the size of the input[8].

A max atom system is a generalisation of a ternary max atom system.

3.8.1 Transformations

r To call a CSP solver on the game, we need to transform it to a CSP problem.
We will use a transformation to a Min-Max System[9].
A Min-Max System is a CSP over

3.8.2 Arc Consistency

[8] Assuming that the input is not represented on a unary system.
[9] See the appendix for a formalisation of Min-Max System.

Chapter 4

Model Design

4.1 Introduction

Chapter 5

Implémentation

Chapter 6

Analyse

For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, c \in I$, let $\mathrm{MA}(x, Y, c)$ be defined as follow:

$$\mathrm{MA}(x, Y, c) \iff x \leqslant \max Y + c$$

Conclusion

Durant ce stage, nous avons étudié les BNNs, implémenté une bibliothèque de BNNs basée sur tensorflow et larq qu'on a nommé binaryflow.

Dans le premier chapitre, nous avons présenté **dB Sense** et ses activités.

Dans le deuxième chapitre, nous avons présenté le problème de la grande complexité, introduit le concept des BNNs et proposé **binaryflow** comme notre solution pour les BNNs

Dans le troisième chapiter, nous avons formalisé les BNNs, et nous avons décris leurs optimisations possible, et les problèmes rencontrés dans leur implémentation.

Dans le quatrième chapitre, nous avons présenté des modèles BNNs, chacun en dérivant ses formules

Dans le cinquième chapitre, nous avons donné notre implémentation de **binaryflow** en jusitifiant les paradigmes utilisés

Dans le sixième chapitre, nous avons analysé 3 jeux de données en implémentant des modèles BNNs pour chacune de ces 3 jeux de données, et en comparant les performances de prédicition et la complexité temps et mémoire des modèles entraînés.

Notre travail n'est qu'une petite introduction des BNNs. En effet, la liste des BNNs proposés dans la littérature est très vaste, et il existe plusieurs autres approches pour faciliter l'entraînement et l'interférence des BNNs que nous n'avons pas considéré vu les contraintes de stage, y parmi:

1. Les méthodes d'optimisations discrètes
2. Les optimiseurs dédiés au BNNs
3. Les binarisations entraînables
4. Les méthodes ensemblistes pour régulaliser les BNNs

De plus, nous avons réussi à vérifier l'optimisation de la multiplication matricielle (L'éxecution est parfois 30 fois plus rapide), mais nous n'avons pas pu intégrer cette optimisation aux modèles tensorflow. Et malgré que larq supporte lui même un déploiement optimisé, nous n'avons pas aussi pu l'exploiter puisque ce déploiement ne supporte que les processeurs ARMv8, et nous utilisons une machine avec un processeur d'architecture x86-64.

Finalement, nous avons fait une petite intégration du code carbone comme une mésure de coût d'entraînement. Une fois le problème de déploiement est résoulu, nous recommenderions l'utilisation de ce même métrique pour estimer le coût d'interférence qui va justifier l'utilisation des BNNs.

Appendix A

On Constraint Satistfaction Problems

In the previous chapters, we described how the system works, without formalising the CSP approach.
On this chapter, we will describe the CSP systems that we have used, with an equivalence proof between them.

A.1 Max Atom System

A.1.1 Ternary Max Atom Systems

- Let $\mathcal{X}$ be a finite set of variables
- Let $D = I \cup \{-\infty\}$, with $I \subseteq \mathbb{R}$.
- For $x, y, z \in \mathcal{X}, c \in I$, let $\mathrm{MA}(x, y, z, c)$ be defined as follow:

$$\mathrm{MA}(x, y, z, c) \iff x \leqslant \max(y, z) + c$$

A ternary max atom system is $\mathrm{CSP}(D, \Gamma)$ where:

$$\begin{aligned}
&\Gamma = \{\mathrm{MA}(x, y, z, c), \quad (x, y, z, c) \in \mathscr{R}\} \\
&\mathscr{R} \subseteq \mathcal{X}^3 \times I \\
&\mathscr{R} \text{is finite}
\end{aligned}$$

A.1.2 Max Atom Systems

- Let $\mathcal{X}$ be a finite set of variables
- Let $D = I \cup \{-\infty\}$, with $I \subseteq \mathbb{R}$
- For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, c \in I$, let $\mathrm{MA}(x, Y, c)$ be defined as follow:

$$\mathrm{MA}(x, Y, c) \iff x \leqslant \max Y + c$$

A max atom system is $\mathrm{CSP}(D, \Gamma)$ where:

$$\Gamma = \{\mathrm{MA}(x, Y, c), \quad (x, Y, c) \in \mathscr{R}\}$$
$$\mathscr{R} \subseteq \mathcal{X} \times (\mathscr{P}(\mathcal{X}) \backslash \{\varnothing\}) \times I$$
$$\mathscr{R}\text{is finite}$$

A.1.3 Min Max System $\leqslant$ Max Atom System

- Let $S = \mathrm{CSP}(\mathcal{X}, D, \Gamma)$ a max atom system.
- Let $R \in \Gamma$
- Let $x \in \mathcal{X}, Y \in \mathscr{P}(\mathcal{X}), c \in I$ such that $R = \mathrm{MA}(x, Y, c)$ such that $|Y| > 2$

Recursive Reduction

We will reduce the arity of R as follow:

- Let $y, z \in Y$ such that $y \neq z$
- We introduce a variable $w \notin \mathcal{X}$
- Let $\mathcal{X}' = \mathcal{X} \cup \{w\}$
- Let $Y' = (Y \cup \{w\}) \backslash \{y, z\}$
- Let $R' = \mathrm{MA}(x, Y', c)$
- Let $R_w = \mathrm{MA}(w, \{y, z\}, 0)$
- Let $\Gamma' = (\Gamma \cup \{R', R_w\}) \backslash \{R\}$
- Let $S' = \mathrm{CSP}(\mathcal{X}', D, \Gamma)$

We will prove that S' is equivalent to S.
Without a loss of generality:

- we will order $\mathcal{X}$ such that $x_0 \leqslant \cdots \leqslant x_{n-1}$ with $n = |\mathcal{X}|$
- $x_{n-2} = y$
- $x_{n-1} = z$
- We will set $x_n = w$
- Let $i \in \{0, \ldots, n-1\}$ such that $x_i = x$

Implication Let $s_0, \ldots, s_n$ an assignment of S'. It is trivial that $s_0, \ldots, s_{n-1}$ is an assignment of S

Equivalence

- Let $s_0, \ldots, s_{n-1}$ an assignment of S'
- Let $s_n = \max(s_{n-1}, s_{n-2})$

Then, $s_0, \ldots, s_n$ is an assignment of S'

Induction

Since the number of variables is finite. The number of constraints is finite, and the arity of each constraint is finite. Applying such reduction iteratively will eventually give a system S^* equivalent to S with:

- $\mathcal{X}^*$ the set of variables with $\mathcal{X} \subseteq \mathcal{X}'$
- Γ^* is the set of constraints:
- Each constraint is of the form $\mathrm{MA}(x, Y, c)$ with $x \in \mathcal{X}', Y \subseteq \mathcal{X}', c \in I$ with $|Y| \leqslant 2$

Now such system can be transformed to a ternary system S_3 as follow:

- The set of variables is $\mathcal{X}^*$
- The domain is D
- For every relation $R = \mathrm{MA}(x, Y, c)$ we map it to the relation $R_3 = \mathrm{MA}(x, y, z, c)$ as follow:
- y, z constitute the elements of Y if $|Y| = 2$
- $z = y$ if $|Y| = 1$

It is trivial that S^* is equivalent to S_3. With that, S is equivalent to S_3.

Algorithm 7 Converting a Max Atom System to Ternary Max Atom System

Require: S an N-ary Max Atom system
Ensure: S' a ternary Max Atom system

```
S' ← ∅
H ← ∅                          ▷ H is a symmetric map between variable,variable to variables
V ← Variables(S)               ▷ V is a set containing all variables
for C ∈ S do                   ▷ Iterate over constraints
    c is the constant in the right hand side of C
    Y is the variables in the right hand side of C
    x is the variable in the left hand side of C
    while |Y| > 2 do
        y ← pop(Y)
        z ← pop(Y)
        if (y, z) ∉ domain H then
            w ← newVariable(V)     ▷ Generate a new formal variable not included in V
            V ← V ∪ {w}
            H(y, z) ← w
            H(z, y) ← w
        end if
        w ← H(y, z)
        S' ← S' ∪ {MA(w, y, z, c)}
        Y ← Y ∪ {w}
    end while
end for
return S'
```


Rami ZOUARI
INSAT, Tunisia

A.2 Min-Max System

- Let $\mathcal{X}$ be a finite set of variables
- Let I be the domain of the variables.
- Let $D = I \cup \{-\infty\}$, with $I \subseteq \mathbb{R}$
- For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, c \in I$, let $\mathrm{MA}(x, Y, c)$ be defined as follow:

$$\mathrm{MA}(x, Y, c) \iff x \leqslant \max Y + c$$

- For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, c \in I$, let $\mathrm{MI}(x, Y, c)$ be defined as follow:

$$\mathrm{MI}(x, Y, c) \iff x \leqslant \min Y + c$$

A min-max system is $\mathrm{CSP}(D, \Gamma)$ where:

$$\begin{aligned}
&\Gamma = \{O(x, Y, c), \quad (O, x, Y, c) \in \mathscr{R}\} \\
&\mathscr{R} \subseteq \{\mathrm{MA}, \mathrm{MI}\} \times \mathcal{X} \times (\mathscr{P}(\mathcal{X}) \backslash \{\varnothing\}) \times I \\
&\mathscr{R} \text{is finite}
\end{aligned}$$

A.2.1 Equivalence with Max Atom Systems

A Max Atom system is trivially a Min Max system. So we will only prove the latter implication.

Let $S' = \mathrm{CSP}(D, \Gamma)$ be a Min Max system, and let: - Γ_{MI} be the constraints that has MI - Γ_{MA} be the constraints that has MA

For each $\mathrm{MI}(x, Y, c) \in \Gamma_{\mathrm{MI}}$. we replace it with the following constraints:

$$\Gamma^{x,Y,c} = \bigcup_{y \in Y} \{\mathrm{MA}(x, \{y\}, c)\}$$

Now, let:

$$\Gamma' = \Gamma_{\mathrm{MA}} \cup \bigcup_{\mathrm{MI(x,Y,c)} \in \Gamma_{\mathrm{MI}}} \Gamma^{x,Y,c}$$

The system $\mathrm{CSP}(D, \Gamma')$ is a max system

Rami ZOUARI
INSAT, Tunisia

Algorithm 8 Converting a Min-Max System to Max Atom

Require: S a Min-Max system
Ensure: S' an N-ary Max Atom system

```
S' ← ∅
H ← ∅                                   ▷ H is a map between variable,offsets to variables
V ← Variables(S)                        ▷ V is a set containing all variables
for 𝒞 ∈ S do                            ▷ Iterate over constraints
    C is the constants in the right hand side of 𝒞
    Y is the variables in the right hand side of 𝒞
    x is the variable in the left hand side of 𝒞
    if 𝒞 is a min constraint then
        S' ← S' ∪ {MA(x, {y, y}, c),   (y, c) ∈ zip(Y, C)}
    else
        Z ← ∅
        for (y, c) ∈ zip(Y, C) do
            if (y, c) ∉ domain H then
                z ← newVariable(V)      ▷ Generate a new formal variable not included in V
                V ← V ∪ {z}
                H(y, c) ← z
            end if
            S' ← S' ∪ {MA(H(y, c), {y, y}, c)}
            Z ← Z ∪ {H(y, c)}
        end for
        S' ← S' ∪ {MA(x, Z, 0)}
    end if
end for
```

Algorithm 9 Converting a Mean Payoff Game to a Min Max system

Require: G a Mean Payoff Game
Ensure: S an Min-Max system

```
E ← E(G)                                ▷ The edges of G
V ← V(G)                                ▷ The variables of G
W ← W(G)                                ▷ The weight function of G
P ← P(G)                                ▷ The set of player of G
for (u, p) ∈ V × P do
    x ← (u, p)
    A ← Adj(x)
    Y = {(a, p̄),   a ∈ A}
    C ← W(A)                            ▷ Calculating the weights element wise.
    if p is Max then:
        OP ← MA
    else
        OP ← MI
    end if
    S ← S ∪ {OP(x, Y, C)}
end for
```

A.3 Polymorphisms

A polymor

Appendix B

On Random Graphs

In the previous chapters, we gave a rough analysis of graph generation.
In this chapter, we will dive into a more detailed analysis.

B.1 Introduction

B.2 Sinkless $\mathcal{D}(n,p)$ Graph

B.2.1 Property

Let $\mathtt{P}$ be the property[1] "Graph has not sink".
This property is increasing in the sense that:

$$\forall H \text{ spanning subgraph of } G, \quad H \in \mathtt{P} \implies G \in \mathtt{P}$$

As a consequence:

$$\forall n \in \mathbb{N}, p, p' \in [0,1]/ \quad p \leqslant p', \quad \mathscr{P}(\mathcal{D}(n,p) \in \mathtt{P}) \leqslant \mathscr{P}(\mathcal{D}(n,p') \in \mathtt{P})$$

We will be interested in two properties:

- The property "Vertex v has no sinks". We denote it by $\text{NoSink}(v)$.
- The property "Graph G has no sinks at all". We denote it by $\text{Sinkless}(G)$.

B.2.2 Basic Comparison with Normal $\mathcal{D}(n,p)$

We will calculate the expected value of $\deg v$. By applying the law of total expectancy:

$$\begin{aligned}\mathbb{E}[\deg v] &= \mathbb{E}[\deg v \mid \deg v > 0] \times \mathscr{P}(\deg v > 0) + \mathbb{E}[\deg v \mid \deg v = 0] \times \mathscr{P}(\deg v = 0) \\ &= \mathbb{E}[\deg v \mid \text{Sinkless}(G)] \times \mathscr{P}(\text{NoSink}(v))\end{aligned}$$

[1] Formally, a property is a just a set of graphs. In practice, it is a set that has desirable "properties".

With that:

$$\mathbb{E}[\deg v \mid \text{Sinkless}(G)] = \frac{\mathbb{E}[\deg v]}{\mathscr{P}(\text{NoSink}(v))} = \frac{np}{1-(1-p)^n} \leqslant \frac{np}{1-e^{-1}}$$

$$\mathbb{E}[|\mathcal{E}|] = \sum_{v\in V} \mathbb{E}[\deg v \mid \text{Sinkless}(G)] = \frac{n^2 p}{1-(1-p)^n} \leqslant \frac{n^2 p}{1-e^{-1}}$$

This shows that the conditional distribution does inflict a small multiplicative bias on the expected number of edges and expected degree.
This serves as an evidence that $\mathcal{D}^S(n,p)$ is similar enough to $\mathcal{D}(n,p)$

B.2.3 Property Probability

- Let $G \sim \mathcal{D}(n,p)$
- Let v a vertex of G

The probability that $\text{NoSink}(v)$ occurs is:

$$\begin{aligned}\mathcal{P}(\text{NoSink}(v)) &= 1 - \mathcal{P}(\text{Adj}\, v = \varnothing)\\ &= 1 - \mathcal{P}(\deg v = 0)\\ &= 1-(1-p)^n\end{aligned}$$

Now, it is clear that the sequence of events $(\text{NoSink}(v))_{v\in V}$ is independent.
With that, the probability that the whole graph is sinkless is:

$$\begin{aligned}\mathcal{P}(\text{Sinkless}(G)) &= \mathcal{P}(\text{Adj}\, v \neq \varnothing \quad \forall v \in V)\\ &= \mathcal{P}\left(\bigwedge_{v\in V} \text{NoSink}(v)\right)\\ &= \prod_{v\in V} \mathcal{P}(\text{NoSink}(v))\\ &= (1-(1-p)^n)^n\end{aligned}$$

B.2.4 Asymptotic Analysis For Dense $\mathcal{D}(n,p)$

Let $c > 0$. We have for large enough n:

$$(1-p)^n \leqslant \frac{c}{n}$$

Which implies:

$$(1-\tfrac{c}{n})^n \leqslant (1-(1-p)^n)^n \leqslant 1$$

If we take the limit, we have:

$$e^{-c} \leqslant \lim_{n\to+\infty} (1-(1-p)^n)^n \leqslant 1 \quad \forall c > 0$$

By tending c to 0, we get:

$$\lim_{n\to+\infty} (1-(1-p)^n)^n = 1$$

B.2.5 Asymptotic Analysis For Sparse $\mathcal{D}(n, p)$

Let:

$$\begin{aligned} f : \mathbb{R}_+^* \times \mathbb{R}_+ \times \mathbb{R} &\to \mathbb{R}_+ \\ x, k, c &\to (1 - g(x, k, c))^x \\ g : \mathbb{R}_+^* \times \mathbb{R}_+ \times \mathbb{R} &\to \mathbb{R}_+ \\ x, k, c &\to \left(1 - \frac{k \ln x + c}{x}\right)^x \end{aligned}$$

By construction, $f(n, k, c)$ is the probability of a graph following $\mathcal{G}(n, \frac{k \ln n + c}{n})$ to contain no sink.

We have:

$$\begin{aligned} \ln g(k, x, c) &= x \ln\left(1 - \frac{k \ln x + c}{x}\right) \\ &= -k \ln x - c - \frac{(k(\ln x) + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right) \\ \implies g(x, k, c) &= \exp\left(-k \ln x - c - \frac{(k \ln x + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right)\right) \\ &= \frac{e^{-c}}{x^k} \times e^{\frac{-(k \ln x + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right)} \\ &= \frac{e^{-c}}{x^k}\left(1 - \frac{(k \ln x + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right)\right) \\ &= \frac{e^{-c}}{x^k} - e^{-c}\frac{k^2(\ln x)^2}{2x^{k+1}} + o\left(\frac{(\ln x)^3}{x^{k+2}}\right) \\ &= \frac{e^{-c}}{x^k} + o\left(\frac{1}{x^k}\right) \\ \implies 1 - g(x, k, c) &= 1 - \frac{e^{-c}}{x^k} + o\left(\frac{1}{x^k}\right) \\ \implies x \ln(1 - g(x, k, c)) &= -\frac{e^{-c}}{x^{k-1}} + o\left(\frac{1}{x^{k-1}}\right) \\ &\sim -\frac{e^{-c}}{x^{k-1}} \\ \implies f(x, k, x) &= e^{-\frac{e^{-c}}{x^{k-1}} + o(\frac{1}{x^{k-1}})} \end{aligned}$$

Now with that:

$$\lim_{x \to +\infty} x \ln(1 - g(x, k, c)) = \begin{cases} -\infty & \text{if} k \in [0, 1[\\ -e^{-c} & \text{if} k = 1 \\ 0 & \text{otherwise if} k \in]1, +\infty[\end{cases}$$

Finally, we can conclude that:

$$\lim_{x\to+\infty} f(x,k) \begin{cases} 0 & \text{if} k \in [0,1[\\ e^{-e^{-c}} & \text{if} k = 1 \\ 1 & \text{otherwise if} k \in]1,+\infty[\end{cases}$$

B.3 Repeating Construction

B.3.1 Estimating Complexity

Now the method of rejecting graphs that have sinks and retrying give us a natural question about how many times will the algorithm reject graph until finding a desirable one.
The number of such rejections will follow a geometric law $\mathcal{G}(h(n,p))$ where:

$$h(n,p) = \mathscr{P}\left(\text{Sinkless}(\mathcal{D}(n,p))\right) = (1-(1-p)^n)^n$$

With that, the expected complexity of the algorithm will be:

$$\tilde{\mathcal{O}}\left(\frac{C(n,p)}{h(n,p)}\right) = \tilde{\mathcal{O}}\left(\frac{C(n,p)}{(1-(1-p)^n)^n}\right)$$

With $C(n,p)$ the cost of building the graph, depending on the algorithm[2].

B.3.2 Dense Graph case

Now it is clear for dense enough graphs, in particular with $p(n) \geqslant \frac{k\ln(n)}{n}$ for large enough n, the expected complexity will reduce to $\mathcal{O}(C(n,p))$. And thus, we consider the rejection method to be efficient.

B.3.3 Sparse Graph case

If $p(n) = \frac{k\ln n}{n} + c$ with $k < 1$. We have:

$$(1-(1-p)^n)^n = e^{-e^{-c}x^{1-k}+o(x^{1-k})}$$

With that, the expected complexity of the rejection method will be:

$$\tilde{\mathcal{O}}\left(C(n,p)\times\exp\left(e^{-c}x^{1-k}+o(x^{1-k})\right)\right)$$

which is an exponential algorithm, and thus inefficient for large graphs. Since property `P` is increasing, this argument generalises to $p(n) \leqslant \frac{k\ln n}{n} + c$ for large enough n

[2]The two algorithms that we have discussed are the naive $\mathcal{O}(n^2)$ algorithm and the more optimized $\mathcal{O}(pn^2+n)$ algorithm.

B.4 Binomial Rejection Construction

Instead of throwing the whole graph at once. For every vertex $u \in V$, we try to construct the adjacenty vertices of u, and repeat if the procedure gives $\operatorname{Adj} u = \varnothing$
With this trick, the expected complexity will reduce for both algorithms to:

$$\tilde{\mathcal{O}}\left(\frac{C(n,p)}{1-(1-p)^n}\right)$$

Now for our case, it is natural to assume that $p(n) \geqslant \frac{1}{n}$, as a Mean Payoff Graph does not have a sink. With that:

$$1-(1-p)^n \geqslant 1-(1-\tfrac{1}{n})^n \geqslant 1-e^{-1}$$

Therefore, the expected complexity will simplify to:

$$\tilde{\mathcal{O}}\left(C(n,p)\right)$$

Moreover, the cost of the conditionning makes only a constant $\frac{1}{1-e^{-1}} \approx 1.582$ factor slowdown, which is effectively neligible.

B.5 Expected Mean Payoff

B.5.1 Definition

- Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a mean-payoff game
- For $u \in \mathcal{V}$, Let $\mathscr{P}(u)$ be the set of probability distributions over the set $\operatorname{Adj}(u)$
- We define a fractional strategy as a function $\Phi \in \mathscr{P}$

B.5.2 Matrix Form

- Let $n = |\mathcal{V}|$
- Let $u_1, \dots, u_n$ an enumeration of elements of $\mathcal{V}$ A fractional strategy can be represented as a matrix A such that:

$$\mathcal{P}(\Phi(u_i) = u_j) = A_{i,j}$$

B.5.3 Fractional Strategies

Notations

- Let A, B be a pair of fractional strategies
- Let P_m, Q_m two random variables defining the mean-payoffs for the respective players after turn m

Expected Cost of $\frac{1}{2}$-turn

Let $\Pi \in \{A, B\}$

We have:

$$\mathbb{E}\left[w(u,\Pi(u))\right] = \sum_{v\in \operatorname{Adj} u} w(u,v)\cdot \mathcal{P}(\Pi(u)=v)$$

Expected Cost of full turn

Let h be the cost of a turn

We have:

$$\mathbb{E}\left[h(u,A(u),B\circ A(u))\right] = \mathbb{E}[w(u,A(u))] + \sum_{v\in \operatorname{Adj} u} \mathbb{E}[w(v,B(v))]\cdot \mathcal{P}(A(u)=v)$$

Expected Total Payoff

- Let $\Pi = B\circ A$
- Let $(X_m)_{m\in\mathbb{N}}$ defined as follow:

$$\begin{cases} X_0 & = s \\ \forall m\in\mathbb{N}^*, \quad X_m & = \Pi(X_{m-1}) \end{cases}$$

- Let $(R_m)_{m\in\mathbb{N}}$ defined as follow:

$$\begin{cases} R_0 & = 0 \\ \forall m\in\mathbb{N}^*, \quad R_m & = R_{m-1} + \sum_{u\in V} h(u,A(u),\Pi(u))\cdot \mathcal{P}(X_{m-1}=u) \end{cases}$$

We have:

$$\begin{aligned}
\mathbb{E}[R_m] &= \mathbb{E}[R_{m-1}] + \sum_{u\in V} \mathbb{E}[h(u,A(u),\Pi(u))]\cdot \mathcal{P}(X_{m-1}=u) \\
&= \mathbb{E}[R_{m-1}] + \sum_{u\in V} P^{m-1}(s,u)\times q(u) \\
&= \mathbb{E}[R_{m-1}] + (P^{m-1}\cdot q)(s) \quad \text{(Matrix Multiplication)} \\
&= \sum_{k=1}^{m} (P^{k-1}\cdot q)(s) + \mathbb{E}[R_0] \\
&= \left(\sum_{k=0}^{m-1} P^k\cdot q\right)(s) + \mathbb{E}[R_0] \\
&= \left(\sum_{k=0}^{m-1} P^k\cdot q\right)(s)
\end{aligned}$$

Rami ZOUARI
INSAT, Tunisia

Now, we may see that the formula is easy generalisable to any starting vertex:

$$\mathbb{E}[R_m] = \sum_{k=0}^{m-1} P^k \cdot q$$

Expected Mean Payoffs

The mean-payoff is defined as:

$$K_m = \frac{R_m}{m}$$

We define K_∞ as:

$$K_\infty = K_{+\infty} = \lim_{m \to +\infty} \frac{R_m}{m}$$

Let $m \in \mathbb{N} \cup \{+\infty\}$ Now, the expected mean-payoff can act as a the judge for who is winning:

1. Player 0 wins if $\mathbb{E}[K_m] > 0$
2. Player 1 wins if $\mathbb{E}[K_m] < 0$
3. Else, it is a tie

Now, if m is finite, we can calculate $\mathbb{E}[K_m]$ directly. Otherwise, we have:

$$\mathbb{E}[K_\infty] = \lim_{m \to +\infty} \frac{1}{m} \sum_{k=0}^{m} P^k \cdot q$$

Now, P can be seen as a stochastic matrix. Thus it has a simple eigenvalue of value 1, and all its other eigenvalue λ satisfies:

$$\lambda \neq 1 \wedge |\lambda| \leqslant 1$$

Also, we have (Proof?):

$$|\lambda| = 1 \implies \lambda \text{is a simple eigenvalue}$$

With that, it can be proven that the $\lim_{m \to +\infty} \frac{1}{m} \sum_{k=0}^{m} P^k$ converges so some matrix T

This matrix can be constructed as follow. Let $P = VJV^{-1}$ the jordan normal form of P

Without a loss of generality, we will suppose that the first eigenvalue of this decomposition is 1 We have then:

$$T = V \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{0}_{n-1} \end{pmatrix} V^{-1}$$

Discounted Payoffs

Using the same approach as the mean payoffs. Let R_m be the discounted payoff. It can be shown that:

$$\mathbb{E}[R_m] = \sum_{n \in \mathbb{N}} \gamma^n P^n \cdot q = (\mathrm{Id} - \gamma P)^{-1} q$$

Bibliographie

[1] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *CoRR* abs/1308.3432 (2013). arXiv: `1308.3432`. URL: `http://arxiv.org/abs/1308.3432`.

[2] Melissa Chan. *Go Player Finally Defeats Google's AlphaGo After 3 Losses*. 14 Mar 2016. URL: `https://time.com/4257406/go-google-alphago-lee-sedol`.

[3] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[4] Matthieu Courbariaux and Yoshua Bengio. "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830 (2016). arXiv: `1602.02830`. URL: `http://arxiv.org/abs/1602.02830`.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *CoRR* abs/1511.00363 (2015). arXiv: `1511.00363`. URL: `http://arxiv.org/abs/1511.00363`.

[6] Elizabeth Gibney. "Self-taught AI is best yet at strategy game Go". In: *Nature* (Oct. 2017). DOI: `10.1038/nature.2017.22858`.

[7] Citu Group. *What is the carbon footprint of a house?* URL: `https://citu.co.uk/citu-live/what-is-the-carbon-footprint-of-a-house`.

[8] Koen Helwegen et al. "Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization". In: *CoRR* abs/1906.02107 (2019). arXiv: `1906.02107`. URL: `http://arxiv.org/abs/1906.02107`.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[10] IBM. *Présentation générale de CRISP-DM*. `https://www.ibm.com/docs/fr/spss-modeler/saas?topic=dm-crisp-help-overview`. 2017.

[11] A.G. Ivakhnenko et al. *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967. ISBN: 9780444000200. URL: `https://books.google.tn/books?id=rGFgAAAAMAAJ`.

[12] Zohar Jackson. *Free Spoken Digits Dataset*. `https://github.com/Jakobovski/free-spoken-digit-dataset`. 2019.

[13] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[15] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[16] Yann LeCun. *THE MNIST DATABASE*. http://yann.lecun.com/exdb/mnist. 1998.

[17] Xiaofan Lin, Cong Zhao, and Wei Pan. "Towards Accurate Binary Convolutional Neural Network". In: *CoRR* abs/1711.11294 (2017). arXiv: 1711.11294. URL: http://arxiv.org/abs/1711.11294.

[18] Seppo Linnainmaa. "Taylor Expansion of the Accumulated Rounding Error". In: *BIT* 16.2 (1976), 146–160. ISSN: 0006-3835. DOI: 10.1007/BF01931367. URL: https://doi.org/10.1007/BF01931367.

[19] Zechun Liu et al. "Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm". In: *CoRR* abs/1808.00278 (2018). arXiv: 1808.00278. URL: http://arxiv.org/abs/1808.00278.

[20] Kim Martineau. *Shrinking deep learning's carbon footprint*. 7Aug 2021. URL: https://news.mit.edu/2020/shrinking-deep-learning-carbon-footprint-0807.

[21] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *CoRR* abs/1603.05279 (2016). arXiv: 1603.05279. URL: http://arxiv.org/abs/1603.05279.

[22] Md. Sahidullah and Goutam Saha. "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition". In: *Speech Communication* 54.4 (2012), pp. 543–565. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2011.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0167639311001622.

[23] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *CoRR* abs/1906.02243 (2019). arXiv: 1906.02243. URL: http://arxiv.org/abs/1906.02243.

[24] Tim Dettmers. *Deep Learning in a Nutshell: History and Training*. https://developer.nvidia.com/blog/deep-learning-nutshell-history-training/. 16 Dec. 2015.

[25] Aimee Wynsberghe. "Sustainable AI: AI for sustainability and the sustainability of AI". In: *AI and Ethics* 1 (Feb. 2021). DOI: 10.1007/s43681-021-00043-6.

[26] Chunyu Yuan and Sos S. Agaian. "A comprehensive review of Binary Neural Network". In: *CoRR* abs/2110.06804 (2021), pp. 3–4. arXiv: 2110.06804. URL: https://arxiv.org/abs/2110.06804.

[27] Chunyu Yuan and Sos S. Agaian. "A comprehensive review of Binary Neural Network". In: *CoRR* abs/2110.06804 (2021). arXiv: 2110.06804. URL: https://arxiv.org/abs/2110.06804.