M3.L4: Série d'exercices complémentaire (solution)

1. Conversion décimal ⇔ virgule flottante simple précision IEEE 754 (fp32)

1.1.1) $\mathbf{3E400000_{16}}$ **:** la formule normalisée s'applique car **e** > 0.
Il faut exprimer le motif binaire à partir du nombre en hexadécimal : 0011 1110 0100 00000000000000000000 On en extrait le bit de signe qui vaut 0 ; ce nombre est donc un nombre positif. Le motif binaire de **e** provient des 8 bits qui suivent le bit de signe = 01111100 qui vaut 124. La puissance de la base 2 vaut 124-127= -3. La mantisse vaut 10000000000000000000 ; le nombre représenté est 2^{-3} x $1{,}1_2$ = 1,5/8 = 0,1875

1.1.2) $\mathbf{00000000_{16}}$: la formule dénormalisée s'applique car **e** vaut zéro. Or la somme pondérée des bits de la mantisse vaut 0, donc la valeur de X vaut zéro.

1.2.1) $\mathbf{1,25.2^{-140}}$: on utilise la forme dénormalisée car **P = E(log2(1,25.2^{-140})) = -140** est inférieur à -126
La quantité Z correspondant à la mantisse vaut : 1,25.2^{-14}
Sachant que 1,25 vaut 1,01 en binaire, on obtient la valeur finale de la forme dénormalisée en décalant 1,01 de 14 cases vers la droite, qui donne en binaire : 0,00000000000001010000000. La mantisse contient seulement la partie fractionnaire de la forme dénormalisée.

1.2.2) $1,125.2^{31}$ **:** on utilise la forme normalisée car **P = E(log2(**$1,125.2^{31}$**)) = 31** est supérieur à -127
L'exposant est la valeur binaire de P + 127 = 158 sur 8 bits, c'est-à-dire : 10011110
La valeur de 1,125 en binaire est 1,001. La mantisse est seulement la partie fractionnaire de la forme normalisée, c'est-à-dire 00100000000000000000000.

1.3.1) Le nombre décimal de la question 1.2.2 est exactement représenté en simple précision. Le nombre représenté suivant est donnée par la valeur de mantisse : 00100000000000000000001. Ces deux nombres sont séparés par une quantité Δ qui constitue l'erreur absolue maximum entre 2^{31} et 2^{32}.
Cette quantité Δ est donnée par $\mathbf{\Delta = 2^{31}.2^{-23} = 2^{31-23} = 2^8 = 256.}$

1.4.1) la précision est donnée par le bit de poids faible de la mantisse :

- **IEEE fp16 : 2^{-10}** c'est-à-dire 1/1024 c'est-à-dire environ **0,1%**.
  On a une garantie de 3 chiffres significatifs en décimal.
- **bf16 : 2^{-7}** c'est-à-dire 1/128 c'est-à-dire environ **0,8%**.
  On a seulement une garantie de 2 chiffres significatifs en décimal.

1.4.2) On obtient **bf16** seulement par troncation de la mantisse de **fp32** ; le nombre de bits de l'exposant est préservé. Donc ces 2 représentations ont (en gros) le même *domaine couvert* car celui-ci est essentiellement déterminé par l'exposant. Cet aspect est essentiel pour le succès des calculs des réseaux de neurones numériques car il est important de pouvoir représenter les *très petites valeurs* qui sont impliquées dans le processus d'apprentissage. En effet l'algorithme modifie les paramètres du réseau de neurone en faisant un très grand nombre de très petits changements de leur valeur.

Par ailleurs les travaux publiés dans [CACM juillet 2020] ont montré que, pour ce type de traitements, la perte de *précision* due au nombre plus faible nombre de bits de la mantisse de **bf16** par rapport à **fp16** n'est finalement pas un problème pour l'apprentissage profond.

De plus, dans la dernière phase de calcul de la sortie du réseau de neurones, la représentation **fp32** est aussi utilisée et, à ce moment là, il est important que les domaines couverts par **bf16** et **fp32** soient (en gros) les mêmes. Ce choix permet de réduire les pertes de précision dans un calcul quand on ajoute des petites valeurs à des grandes valeurs (il s'agit d'un problème que nous n'avons pas abordé en cours mais qui est analogue à l'exemple vu en cours sur la non-commutativité de l'addition).

2.4.3) car **fp16** a seulement 5 bits pour l'exposant au lieu de 8 bits pour **fp32**.

La valeur maximum sera plus faible pour la forme normalisée de **fp16** comparée à **fp32**

Le maximum de **fp16** est : 2^{31-15} = $\mathbf{2^{16}}$ = $\mathbf{64.2^{10}}$ **-1** = un peu plus que 65.10^3

La valeur minimum sera plus grande pour la forme normalisée de **fp16** comparée à **fp32**

Le minimum de **fp16** pour la forme normalisée est : $\mathbf{2^{-14}}$ = environ $1/16.10^{-3}$

---

2. Entropie du jeu d'échec

2.1) Voici un exemple de stratégie optimale pour poser les questions (il y en a plusieurs équivalentes):

Q1: Est-ce une case vide? Si oui, on sait que la case est vide (avec 1 question posée). Si non, on continue:

Q2: La piéce est-elle noire?

Q3: La pièce est-elle un pion?

Avec ces 2 questions supplémentaires, si la réponse à la question 3 est oui, on sait que la case contient un pion (blanc ou noir), avec donc 3 questions posées en tout. Si non, on continue:

Q4: Est-ce que la pièce est un cavalier ou un fou?

Si oui: **Q5**: Est-ce que la pièce est un cavalier? Avec les réponses à ces 2 dernières questions, on peut identifier la pièce (donc avec en 5 questions en tout).

Si non: **Q5:** Est-ce que la pièce est une tour? Si oui, on peut à nouveau identifier la pièce avec 5 questions en tout. Sinon, on doit encore poser une question:

Q6: est-ce que la pièce est une dame? Et on obtient donc dans ce cas la réponse en 6 questions.

Au total, en tenant compte des probabilités d'apparition de chaque pièce sur l'échiquier, on trouve que le nombre moyen de questions à poser est sous la forme : Somme des (nb_éléments x probabilité) x nb_questions :

$$(1 \times 32/64) \times 1 + (2 \times 8/64) \times 3 + (6 \times 2/64) \times 5 + (4 \times 1/64) \times 6 = 2.5625$$

2.2) Dans le diagramme B le nombre et la nature des pièces n'a pas changé par rapport à la situation de départ, et donc l'entropie reste la même, de la même façon que l'ordre des lettres dans une séquence de lettres n'influence pas l'entropie de la séquence: seules les probabilités d'apparition comptent dans le calcul de l'entropie.

Dans le diagramme C, un pion blanc a disparu. On pourrait faire ici un calcul long et compliqué pour recalculer l'entropie du jeu avec la formule du cours, mais une observation des variations des probabilités permet de répondre en s'appuyant sur la courbe d'un terme pi.log(1/pi). En effet seules 2 probabilités changent :

- La probabilité d'une case vide augmente un peu
- La probabilité d'un pion blanc diminue un peu

Dans le cas général cela ne suffit pas pour déterminer si l'entropie augmente ou diminue ; il faut examiner où se trouvent les probabilités par rapport au maximum de la fonction pi.log(1/pi) qui est obtenu pour pi = environ 0,37 :

- La probabilité d'une case vide était au-dessus de 0,37 avant le changement
  - Une augmentation de cette probabilité produit une diminution du terme pi.log(1/pi)
- La probabilité d'un pion blanc était au-dessous de 0,37 avant le changement
  - Une diminution de cette probabilité produit une diminution du terme pi.log(1/pi)

Il y a une diminution pour les seuls 2 termes pi.log(1/pi) qui changent, donc l'entropie diminue aussi.

3. Expression logique de type « Somme de produits » (Sum of Products)

3.1) Il suffit de consulter la table ; pour chaque sortie, on retient seulement les lignes produisant un 1 et on construit le « produit », ou *minterme,* correspondant. La sortie est la « somme » des *mintermes.*

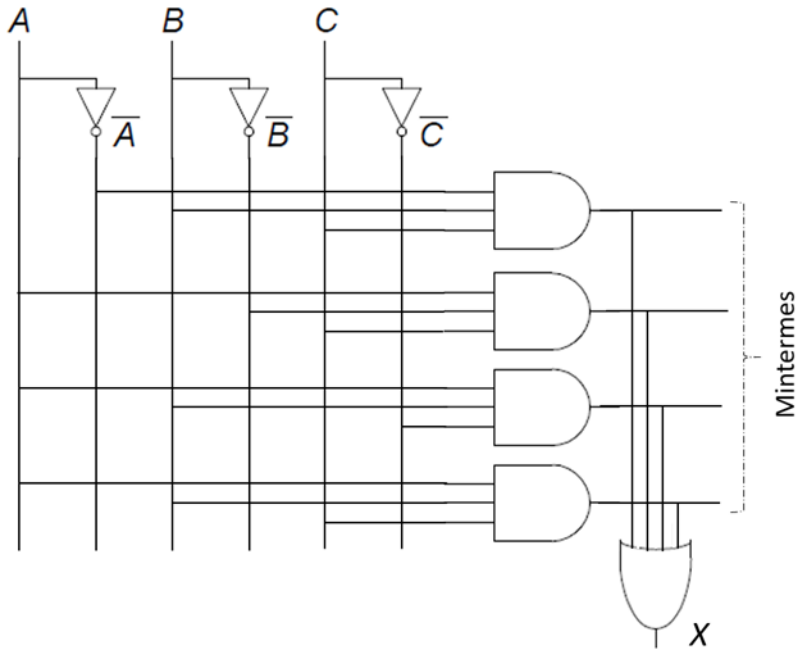
$$X = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

$$Y = \overline{AB}C + \overline{A}B\overline{C} + A\overline{BC} + ABC$$

3.2) nous utilisons des portes à entrées multiples pour simplifier la représentation (Fig1 de la donnée). De même il est important de connaître la convention à propos de la connexion des fils qui amènent les signaux sur les portes logiques (ci-dessous).

a b c

Seuls les cas **a** et **b** représentent des connexions. Il n'y a pas de connexion entre les deux fils du cas c



Chaque porte AND à 3 entrées synthétise un *minterme* :

$\overline{A}BC$

$A\overline{B}C$

$AB\overline{C}$

ABC

La porte OR à 4 entrées synthétise la fonction X

Perspective :

Cet exercice est une généralisation de l'exercice 1 de la série M3.L1 qui illustrait la synthèse du circuit de portes logiques produisant un ou-exclusif. Nous voyons avec cet exemple que l'on peut synthétiser n'importe quelle fonction logique avec la forme « somme de produits ».

Cette observation a conduit au développement de circuits appelés **FPGA** (Field-Programmable-Gate-Arrays) qui justement permettent de synthétiser des « sommes de produits » à partir de nombreuses entrées. Ce type de circuit est utilisé par exemple pour le prototypage de nouvelles générations de processeurs.