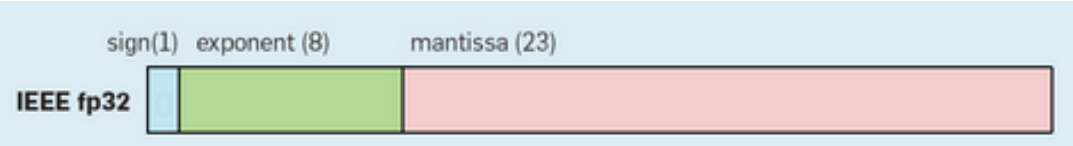
M3.L4: Série d'exercices complémentaires

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

Cet exercice est complémentaire de l'exercice 7.2 de la série **M1.L1** sur la virgule flottante simple précision.



CACM Juillet2020

Représentation en virgule flottante IEEE 754 simple précision sur 32 bits
avec 1 bit de signe noté **s**, 8 bits d'exposant noté **e**, et 23 bits de mantisse noté **m**.

1.1) Binaire IEEE 754 fp32 => décimal

Le passage du motif binaire vers le décimal dépend de la valeur de l'exposant **e**.

Si e > 0, alors la valeur décimale X est donné par la ***forme normalisée*** suivante (pensez à la forme utilisée en décimal pour la notation scientifique):

$$X = (-1)^s . 2^{(e-127)} . (1 + \Sigma\, m_i . 2^{-i})$$

Si e = 0, alors la valeur décimale X est donné par la ***forme dénormalisée*** suivante :

$$X = (-1)^s . 2^{-126} . (\Sigma\, m_i . 2^{-i})$$

Note : les valeurs m_i sont les 23 bits de la mantisse avec **i** variant de 1 à 23, de gauche à droite après **e**.

1.1.1) convertir du binaire vers le décimal le motif binaire suivant donné en hexadécimal : $\mathtt{3E400000}_{16}$

1.1.2) convertir du binaire vers le décimal le motif binaire suivant donné en hexadécimal : $\mathtt{00000000}_{16}$

1.2) Conversion d'un nombre décimal Y => float:

- **s** est donné par le signe de Y. Ensuite on travaille avec **Y' = | Y |**
- soit **P** la puissance <u>entière</u> de 2 inférieure ou égale à Y': **$P = E(\log_2(Y'))$**

Si **P > -127**, c'est le cas général pour lequel il faut utiliser la **forme normalisée**

On a **e = P + 127**

Par construction de P, la quantité **$Z = Y'/2^P$** est comprise entre 1 et 2.

On obtient les 23 bits de la mantisse en convertissant la partie fractionnaire de **Z** selon la méthode vue dans l'exercice 1.3 de la série M1.L2. On arrête la conversion dès que la partie fractionnaire est nulle ou dès qu'on a calculé 23 bits.

Sinon ça veut dire que **P < -126** et il faut utiliser la **forme dénormalisée**

Le premier terme de la forme dénormalisée est **2^{-126}** car on est dans le cas où on impose **e = 0**

On a : **$Y' = 2^{-126} . Z$**

D'où: **$Z = Y' / 2^{-126}$**

Par construction, **Z < 1** ; cette quantité peut être convertie en binaire selon la méthode vue dans l'exercice 1.3 de la série M1.L2. On arrête la conversion dès que la partie fractionnaire est nulle ou dès qu'on a calculé 23 bits.

1.2.1) convertir ce nombre décimal en virgule flottante : $1,25.2^{-140}$
1.2.2) convertir ce nombre décimal en virgule flottante : $1,125.2^{31}$

1.3 Précision de la virgule flottante simple précision: dans le contexte de la forme normalisée, la précision dépend du nombre de bits de la mantisse, c'est-à-dire 23. Par construction, on a toujours 2^{23} nombres distincts représentés entre deux puissances de 2 successives. Soit **n** une puissance de 2 comprise entre -126 et 127, l'écart Δ entre deux nombres représentés successifs compris entre 2^n et 2^{n+1} est constant et il vaut :

$$\Delta = 2^n . 2^{-23} = 2^{n-23}$$

1.3.1) Dans la représentation en virgule flottante, quelle quantité sépare le nombre décimal de la question 1.2.2 du suivant ?

1.4) Quelle représentation des nombres à virgule pour les Réseaux de Neurones numériques ?
Les techniques de traitement de masses de données (big data) utilisent de plus en plus des circuits spécialisés pour effectuer un apprentissage profond (deep learning). Pour cette approche aussi il est très pénalisant de transférer les données de la mémoire vers les zones du circuit réservées aux calculs. Des travaux publiés en juillet 2020 ont montré qu'il était très intéressant de faire ***une partie des calculs*** dans un format de **virgule flottante sur 16 bits**. En effet grâce à la réduction du transfert à 2 octets par nombre à virgule flottante, au lieu de 4 octets pour la simple précision, ces travaux ont montré que les performances gagnent un facteur 8.

Le format de virgule flottante sur 16 bits qui a été finalement choisi s'appelle **bf16** (**Brain Floating Point**). Il est différent du standard **IEEE 754** sur 16 bits, noté **fp16** comme on peut le voir dans l'illustration ci-dessous. Le but de cette question est de comparer les propriétés principales de ces deux représentation sur 16 bits:

[CACM juillet 2020]

1.4.1) Quelle est la précision de chacune de ces 2 représentations ?

1.4.2) Quel est le point commun entre **fp32** vu en 1.1 et **bf16** ? Qu'en déduisez-vous (en gros) ?

1.4.3) IEEE 754 **fp16** a un domaine couvert plus petit que **fp32** ; pourquoi ? Quel est-il sachant que le biais appliqué à l'exposant vaut **15** ?

2. L'entropie d'un jeu d'échec

Parmi les 64 **cases** d'un échiquier standard, on en choisit une au hasard de manière uniforme. On suppose également que les pièces (dames, rois, tours, fous, cavaliers et pions) sont placées dans leur position de départ.

2.1) Si vous utilisez une stratégie optimale, de ***combien de questions binaires en moyenne*** allez-vous avoir besoin pour deviner la nature de la pièce (ou *vide*) qui se trouve sur la case tirée au hasard ? Il vous faut distinguer parmi les 13 possibilités suivantes:

dame noire, roi noir, tour noire, fou noir, cavalier noir, pion noir,
dame blanche, roi blanc, tour blanche, fou blanc, cavalier blanc, pion blanc, ou encore case *vide*

Remarque : Il se trouve que dans ce cas précis (image A suivante), le nombre moyen de questions que vous posez correspond exactement à l'entropie du jeu d'échecs (avec les pièces en position de départ) avec les probabilités d'apparition de la pièce numéro j sur l'échiquier (avec ici n = 13 pour inclure le cas de la case vide).

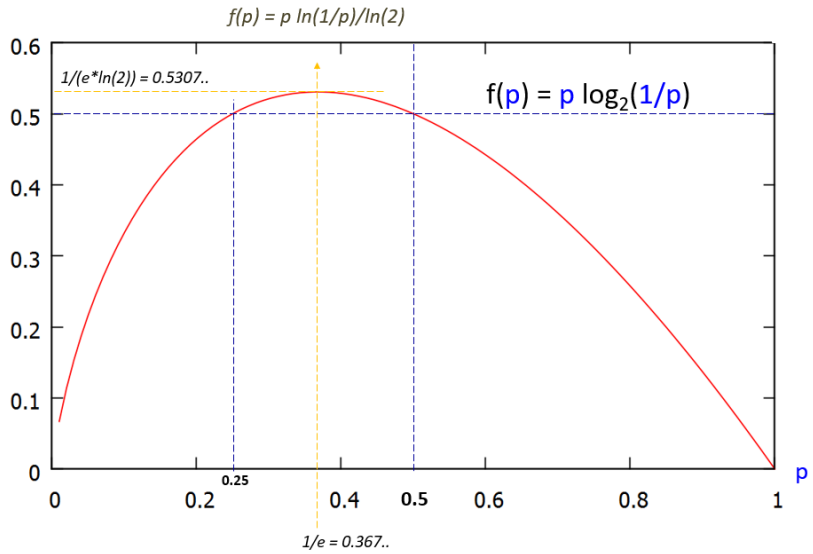
A

B

C

2.2) Supposons maintenant qu'on tire à nouveau une case au hasard sur un échiquier où quelques coups d'une partie ont déjà été joués (images B et C):

Pour l'image B, quelques pions ont simplement été déplacés, tandis que pour l'image C, un pion blanc a été capturé et a donc disparu de l'échiquier. Sans faire aucun calcul, déterminez pour chacun des cas B et C si, par rapport à la position de départ A, l'entropie du jeu a augmenté / est restée la même / a diminué. Si nécessaire vous pouvez utiliser le graphe ci-dessous de la courbe d'un terme $p_i.\log(1/p_i)$ pour déterminer la variation d'entropie et appuyer votre raisonnement.



Chacun des 13 éléments du calcul de l'entropie apporte un terme du type illustré ci-dessus. Pour la question 2.2, on peut facilement déterminer dans quelle direction varie la contribution des éléments dont la probabilité varie.

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

Cet exercice est complémentaire de l'exercice 1 de la série **M3.L1** sur les portes logiques.

Notation :

Rappel : soit x une variable logique, la notation $\bar{x}$ signifie la négation de *x* : si x=0 alors $\bar{x}$ =1 et vice versa.

Notation algébrique des opérateurs *et-logique* et *ou-logique* :

- L'opérateur ***et-logique*** utilise le symbole de la **multiplication** car ces deux opérations ont la même table de vérité. Selon le contexte on utilise le caractère * ou le caractère . ou aucun caractère.
    - Exemples : pour exprimer « A *et* B » on écrit : A*B , A.B, AB

- L'opérateur ***ou-logique*** utilise le symbole de **l'addition** pour disposer d'une écriture plus condensée et homogène avec le choix du et-logique.
    - Exemples : pour exprimer « A *ou* B » on écrit : A+B

Expression logique d'une fonction à l'aide d'une « somme de produits » :

On peut construire une expression qui combine les deux opérateurs et, comme en mathématique, on considère que la multiplication est prioritaire par rapport à l'addition. Voici un exemple d'expression logique d'une fonction F de 3 variables A, B et C qui utilise les deux notations précédentes :

$$F = A\overline{B}C \; + \; AB\overline{C}$$

Du fait des propriétés du *et-logique* et du *ou-logique*, cette fonction vaut 1 seulement pour les deux combinaisons des entrées visibles dans cette expression :

- A vaut 1, B vaut 0, C vaut 1
- A vaut 1, B vaut 1, C vaut 0

F vaut zéro pour les 6 autres combinaisons des 3 variables d'entrée.

Fig1 : on suppose qu'on dispose de portes logiques à entrées multiples

Lien entre la table de vérité et une expression à l'aide d'une Somme de Produits :

Chaque ligne de la table de vérité qui vaut 1 définit un terme de type « produit » contenant toutes les variables d'entrée (éventuellement avec leur valeur complémentaire). Un tel terme s'appelle un ***minterme***.

L'exemple ci-dessus contient 2 *mintermes* : $A\overline{B}C$ et $AB\overline{C}$.

L'expression logique de la fonction F ci-dessus assemble tous les *mintermes* avec l'opérateur *ou-logique*.

A	B	C	XY
0	0	0	00
0	0	1	01
0	1	0	01
0	1	1	10
1	0	0	01
1	0	1	10
1	1	0	10
1	1	1	11

Table de vérité du circuit additionneur 1-bit produisant les 2 sorties X et Y en fonctions des 3 variables d'entrée A, B et C (cours M3.L1 slide 42).

3.1) A partir de la table de vérité du circuit additionneur 1-bit (ci-contre), écrire les deux fonctions logiques X et Y sous forme d'une équation logique « Somme de Produits ». A et B sont les deux bits à additionner et C représente la retenue qui provient de la puissance de 2 inférieure.

3.2) utiliser des portes logiques (Fig1) AND, OR et NOT pour réaliser la fonction X .