HAL open science

Généraliser les possibilités-nécessités pour l'apprentissage profond

Théophane Vallaeys

▶ To cite this version:

Théophane Vallaeys. Généraliser les possibilités-nécessités pour l'apprentissage profond. [Rapport de recherche] RR-9422, Inria. 2021, pp.1-15. hal-03338721v1

HAL Id: hal-03338721

https://inria.hal.science/hal-03338721v1

Submitted on 9 Sep 2021 (v1), last revised 11 Oct 2021 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Généraliser les possibilités-nécessités pour l'apprentissage profond

Théophane Vallaeys

RESEARCH REPORT
N° 9422
Septembre 2021
Project-Team Mnemosyne

ISSN 0249-6399 ISRN INRIA/RR--9422--FR+ENG

Généraliser les possibilités-nécessités pour l'apprentissage profond

Théophane Vallaeys

Équipe-Projet Mnemosyne

Rapport de recherche n° 9422 — Septembre 2021 — ?? pages

Résumé : Ce rapport porte sur un stage de 8 semaines effectué avec l'équipe Inria Mnemosyne travaillant sur l'action exploratoire AIDE (Artificial Intelligence Devoted to Education), et co-encadré par Thierry Viéville et Chloé Mercier. Le cadre restait assez libre et peu défini, mais invitait à chercher à définir la notion de probabilité/nécessité étendue proposée, et tenter de l'appliquer à des exemples d'apprentissages.

Mots-clés : apprentissage profond, réseaux de neurones, possibilités, nécessités

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Generalization of possibility-necessity for deep learning

Abstract: This report is about an 8 weeks internship with the Inria Mnemosyne team working on the exploratory action AIDE (Artificial Intelligence Devoted to Education), and co-supervised by Thierry Viéville and Chloé Mercier. The framework remained rather free, but invited to try to define the extended notion of probability/necessity offered in the subject, and to try to apply it to examples of learning.

Key-words: deep learning, neural networks, possibility, necessity

Table des matières

1 Introduction

1.1 Le sujet

En apprentissage profond, la sortie d'un réseau de neurones est généralement interprétée comme une probabilité (c'est d'ailleurs le principe des couches softmax). Cependant, ce fonctionnement ne permet pas de distinguer naturellement le fait de ne pas connaître un sujet, ou de savoir qu'une information est aléatoire. De plus, ce fonctionnement ne ressemble pas beaucoup à ce que fait un être humain et l'action AIDE veut explorer l'hypothèse que le raisonnement du cerveau ressemble plus aux possibilités et nécessités.

Utiliser les possibilités et nécessités dans un réseau de neurones pourrait permettre de mieux comprendre les raisonnements effectués par le modèle artificiel utilisé et représenter l'incertitude et la connaissance partielle au delà d'un codage aléatoire.

1.2 Les possibilités/nécessités

Les possibilités et nécessités sont un cadre alternatif aux probabilités. Plutôt que de considérer une fonction de somme pour l'union et de produit pour la différence, ce cadre utilise les fonctions min et max. On ne considérera ici qu'un univers Ω discret, avec Π la possibilité (on garde P pour la probabilité). Alors on doit avoir $\forall x \in \Omega, \Pi(x) \in [0;1]$ et $\max_{x\in\Omega} \Pi(x) = 1$. La nécessité est la fonction duale, c'est à dire $\nu(A) = 1-\Pi(\overline{A})$. De plus, pour tout A, B non-nécessairement disjoints, $\Pi(A \cup B) = \max(\Pi(A), \Pi(B))$ et $\nu(A \cap B) = \min(\nu(A), \nu(B))$. Des détails supplémentaires et des propriétés sont présentées dans [**?**].

Alors que les probabilités peuvent représenter la logique classique, les possibilités et nécessités peuvent représenter la logique modale, comme établi en [**?**].

1.3 Utilisations en IA

La notion de possibilité et nécessité a été appliquée à l'IA symbolique. Dans [**?**], cette notion est utilisée avec les base de connaissances RDF, qui est d'ailleurs un outil utilisé par l'action AIDE. Ces bases de connaissances représentes des relations de tout type entre des objets, et le contexte possibilité-nécessité permet de nuancer ces relations sur des catégories générales (par exemple, tous les oiseaux ne volent pas, mais c'est souvent le cas).

Dans le cadre de l'apprentissage profond, une tentative d'utiliser les possibilités et nécessités a été faite dans [**?**]. Cette utilisation se résume à entraîner deux réseaux, l'un donnant une possibilité et l'autre une nécessité, avec deux fonctions de coût adaptées et utiliser une descente de gradient classique.

[**?**] donne quelques bases du calcul avec des intervalles, puis une méthode pour séparer l'espace en trois zones : la zone vrai, la zone fausse, et celle incertaine. Le réseau employé possède un seul neurone en plus de la sortie, et la séparation est une suite de segments (en 2D, et un ensemble d'hyperplans en général).

1.4 IA neuro-symbolique

L'intérêt des possibilités et nécessités serait de mieux représenter le raisonnement humain sur des connaissances partielles, et de pouvoir expliciter quelques propriétés de ses mécanismes, car

c'est un but de l'action exploratoire AIDE que de pouvoir comprendre ce type de fonctionnements et les rapprocher du comportement humain.

Il existe un grand nombre d'approches pour l'IA neuro-symbolique, avec différents niveaux, présenté dans [**?**]. En utilisant le vocabulaire introduit dans ce papier, l'approche que j'ai choisie est une représentation *localiste*. C'est à dire qu'une propriété est associée à un neurone précis, alors qu'une représentation *distribuée* représente une combinaison par un vecteur de valeurs sur tous les neurones d'une couche. Cependant je n'ai pas eu le temps d'intégrer une vrai composante symbolique de manière expérimentale, et les réseaux présentés sont du *Type 1* selon ce document, c'est à dire sans composante symbolique explicite.

L'une des approches pour l'apprentissage neuro-symbolique est l'utilisation de *Logic Tensor Networks*, présentée dans [**?**]. Les opérations du réseau utilisent des fonctions lui permettant de représenter des opérations de logique à valeurs réelles dans sa structure particulière. [**?**] montre que l'on peut l'appliquer à des problèmes d'interprétation sémantique d'image de manière très efficace. Bien que je n'ai pas utilisé cette structure, cela m'a servi à voir comment les fonctions standard d'apprentissage profond pouvaient être modifiées pour développer une autre approche.

2 Définir un cadre théorique

Le sujet du stage s'interrogeait sur l'utilisation de couples (ν, Π) de nécessité-possibilité avec pour seules contraintes que $0 \leqslant \nu \leqslant \pi \leqslant 1$, sans contraindre de rester sur les axes avec $\Pi(x) = 1$ ou $\nu(x) = 0$. Le but est d'avoir une interprétation sémantique possible plus large du résultat. Une piste évoquée était d'utiliser les nombres complexes, que j'ai cependant écartée car les opérations usuelles sur ces derniers ne correspondent pas vraiment à ce que l'on cherche. Une représentation sous forme d'un couple de $\mathbb{R}^2$ suffira.

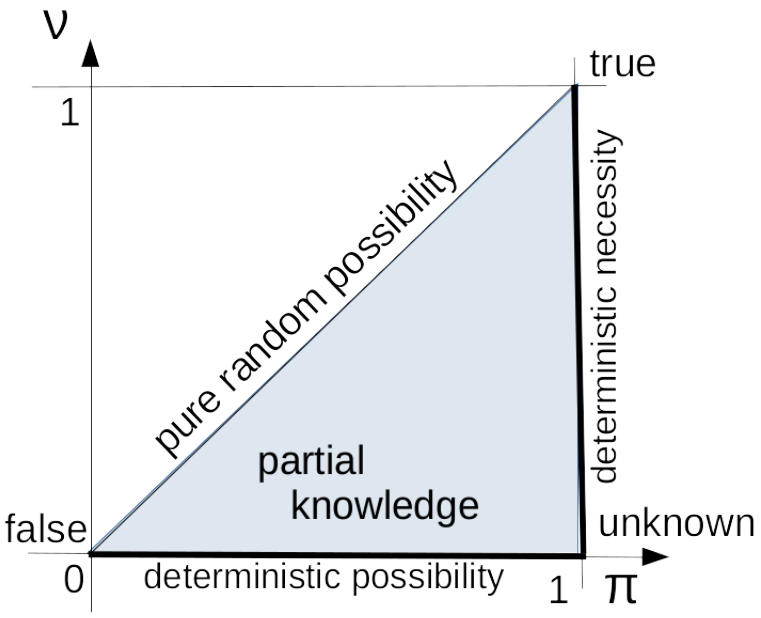


FIGURE 1 – Représentation du sujet, par Thierry Viéville

Cependant, le cadre formel des possibilités et nécessités qui est détaillé dans [**?**] ne satisfait pas les conditions qui nous intéressent : tout d'abord, on a toujours $\Pi(x) = 1$ ou $\nu(x) = 0$, avec Π la possibilité et ν la nécessité. On voudrait pouvoir utiliser les deux dimensions du plan en même temps, pour représenter plus finement une connaissance partielle. De plus, une telle contrainte n'est pas adaptée aux calculs faits par un RNA (réseau de neurones artificiels) sur deux variables, et cela reviendrait donc à n'avoir qu'une variable sur $[-1; 1]$, ce qui n'est guère intéressant en dehors de l'interprétation finale du résultat.

On voudrait également avoir un encadrement de la probabilité sous la forme $\nu(x) \leqslant P(x) \leqslant \Pi(x)$. Cette dernière propriété n'est pas garantie, et elle est même très vite brisée : si notre réseau à une très bonne connaissance pour des entrées x, y, avec $\nu(x) \simeq P(x) \simeq \Pi(x)$ et de même pour y, et si $P(x) \simeq P(y)$, alors $P(x \cup y) \simeq P(x) + P(y) > \Pi(x \cup y) \simeq P(x)$.

Ainsi, on aurait de grandes difficultés à avoir des fonctions permettant d'effectuer des calculs compatibles avec les probabilités, possibilités, nécessités, et leur extension sur le plan. Le plus efficace serait donc de définir quelque chose d'analogue aux possibilités / nécessités, mais en partant des propriétés que l'on cherche à satisfaire, donc un encadrement de la probabilité.

[**?**] propose de définir des possibilités et nécessités en tant que fonctions de vraisemblance : si on observe un état x et que l'on cherche des informations sur une propriété O, on utiliserait, à un facteur multiplicatif près, $\Pi(O) = \max_{o \in O} P(x|o)$ et $\nu(O) = \min_{o \in O} P(x|o)$ qui encadrent $P(x|O)$. Cependant ce cadre est complexe à utiliser en deep learning, car les valeurs obtenues dépendent beaucoup de la granularité de l'ensemble O. En effet, si on choisit une propriété O décrite par un seul élément (« c'est un chien »), alors $\Pi(O) = P(x|O) = \nu(O)$. Cela ne deviendrait intéressant que si l'on découpe O en « sous-propriétés », ce qui est arbitraire. Par exemple, dans le cas de granularité maximum, le plus simple, où $O = \{o \mid o \text{ est une image de chien}\}$, alors $\nu(O) = 0$ (en prenant $o \in O$, $o \neq x$), et soit $\Pi(O) = 1$ si $x \in O$, soit $\Pi(O) = 0$ sinon. Le but du modèle serait alors de trouver une représentation intermédiaire, ou du moins d'estimer des valeurs de Π et ν sur une telle représentation.

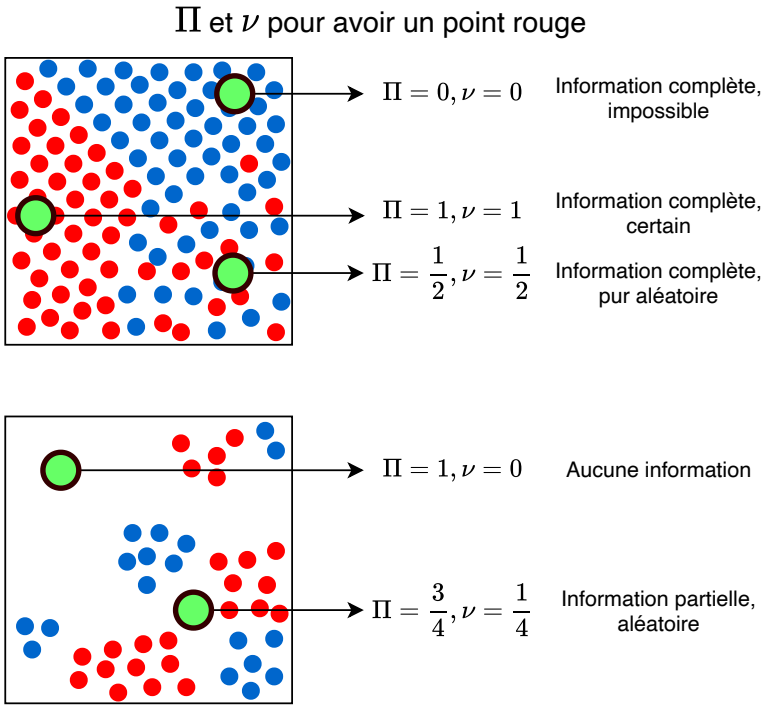


FIGURE 2 – Utilité d'apprendre des possibilités et nécessités, y compris en dehors de $\Pi = 1$ ou $\nu = 0$

La Figure ci-dessus montre que si un modèle est capable d'apprendre ces valeurs, il peut distinguer une absence de connaissance (la zone vide) d'une connaissance parfaite d'un phénomène aléatoire ($\Pi = \frac{1}{2}$ et $\nu = \frac{1}{2}$), ou partielle (avec $\Pi = \frac{3}{4}$ et $\nu = \frac{1}{4}$). Cette représentation n'est pas compatible avec les probabilités, ni les possibilités et nécessités classiques.

2.1 Une proposition de solution

Le but de l'encadrement possibilité/nécessité est de prendre en compte à la fois l'incertitude due à :

— des phénomènes aléatoires, du point de vue de l'observateur, du fait que l'agent n'ait qu'une partie des données. Par exemple, lors d'un lancer de dé, la face sur laquelle il va atterrir est déterministe si on possède toutes les variables physiques, mais peut être considérée comme aléatoire pour un humain. Peut-être ne sait-on même pas si c'est un d12 ou un d20. Un agent qui connaît toutes ces variables peut représenter cela par une distribution de probabilités.

— l'incertitude due au manque de connaissances de l'agent. S'il n'a jamais appris ce qu'est une girafe, ou qu'il n'a jamais vu une espèce de chat particulière, il peut ne pas être certain de sa généralisation, même lorsqu'il en est capable. Cela va être représenté par l'encadrement possibilité/nécessité de la probabilité.

On définit donc l'univers U, dont chaque élément décrit exactement ce qu'il se passe, l'issue des tirages aléatoires éventuels et les données observables d'un état de l'univers. L'agent ne voit cependant qu'une partie de la situation, via une fonction $\Gamma : U \to X$, où X est l'ensemble des observations possibles. On définit O un ensemble de propriétés élémentaires mutuellement

exclusives qui nous intéressent, et on peut définir $f : U \to O$ qui associe à une situation dont on possède toutes les informations la propriété élémentaire vraie. On encodera donc une propriété o comme un ensemble $o \subset O$ de propriétés élémentaires. On va considérer dans la suite U, X et O comme des variables aléatoires dans le cadre des fonctions de probabilité.

L'univers est associé à une «vraie» distribution de probabilité P_0, inconnue de l'agent. Ce dernier va supposer que l'univers est associé à une distribution de probabilité P_θ, paramétrée par le paramètre $\theta \in \Theta$. On peut même restreindre ce cadre, car on ne s'intéressera à P_θ que sous la forme $P_\theta(\ldots|X = x)$. L'agent dispose de connaissances (qui peuvent évoluer selon le modèle de l'agent) sous la forme d'un objet D, dont le type dépend de l'agent. Ce qui nous intéresse, c'est $D(\Theta)$ qui est l'ensemble des paramètres $\theta \in \Theta$ qui soient compatibles avec D. Par exemple, si D contient un encodage numérique de « les chats ont presque toujours deux yeux », les distributions P_θ ne correspondant pas à ce critère sont écartées. L'encodage pourrait par exemple conduire à éliminer toutes les distributions P_θ où la proportion de chats ayant deux yeux serait inférieure à un certain seuil (le fonctionnement dépend du modèle choisi).

Ainsi, l'agent dispose pour seules informations de $x \in X$ (une observation) et D (des connaissances). Il pourrait savoir que la situation observée est donc un des $u \in \Gamma^{-1}(x)$, bien qu'il n'ait pas forcément la connaissance explicite de U. Il voudrait déterminer des informations sur la valeur de vérité d'un certain $o \subset O$.

En cas de connaissance parfaite, la réponse serait la probabilité $p_0(o) = P_0(o|x)$. Le point de vue purement probabiliste (classiquement utilisé dans un réseau de neurones) serait de chercher une estimation $\tilde{\theta}$ telle que $|P_{\tilde{\theta}}(o) - P_0(o)| \leqslant \varepsilon$ (ou au moins en terme d'espérance).

On va encadrer cette probabilité avec Π et ν en séparant la connaissance de l'agent de la partie aléatoire :

$$\nu(o) = \min_{\theta \in D(\Theta)} P_\theta(o|x) \tag{1}$$

$$\Pi(o) = \max_{\theta \in D(\Theta)} P_\theta(o|x) \tag{2}$$

Cela nous donne bien l'encadrement $\nu(o) \leqslant p_0(o) \leqslant \Pi(o)$. Π et ν sont des fonctions de croyances, car $\Pi(\emptyset) = 0$, $\Pi(O) = 1$ et si $A \subset B$ alors $\Pi(A) \leqslant \Pi(B)$, de même pour ν.

Dans ces formules, $P_\theta(\ldots|x)$ est une répartition de probabilité sur $\Gamma^{-1}(x) \subseteq U$. Dans un cas comme le deep learning (où il y a des approximations), on peut assouplir la contrainte : on veut que l'encadrement soit intéressant (donc proche de p_0 si on a beaucoup d'informations), on cherchera donc à ce que $P_U(\nu(o) > P_0(o|x)) < \varepsilon$, et que $P_U(\Pi(o) < P_0(o|x)) < \varepsilon$.

2.2 Propriétés élémentaires

Il est à noter que les fonctions définies ci-dessus diffèrent du cadre classique des possibilités et nécessités :

- Dans le cas d'un évènement purement aléatoire, mais où l'agent le sait, on peut par exemple toujours avoir $\Pi = \nu = \frac{1}{2}$.
- Ainsi, on n'a jamais $\Pi = 1$ ni $\nu = 0$, et aucun évènement élémentaire entièrement possible sur cet exemple. De manière générale, nous n'avons pas forcément $\max \Pi(o) = 1$ ni $\min \nu(o) = 0$. C'est intéressant pour observer des systèmes aléatoires (ou pseudo-aléatoire dû à la représentation limitée), car un agent devrait pouvoir déterminer les valeurs de ces probabilités qui ne sont ni 0 ni 1.
- La propriété $\Pi(A \cup B) = \max(\Pi(A), \Pi(B))$ n'est pas vérifiée ce qui diffère de l'approche classique où elle était utilisé pour construire la distribution. Mais nous avons vu que c'était nécessaire de ne pas l'avoir pour obtenir l'encadrement de P dans le cas général.

Cependant, on garde certaines propriétés, notamment le fait que ce soit des fonctions de croyance, et :

$$\begin{aligned}\nu(\overline{A}) &= \min_{\theta\in D(\Theta)}(1-P_\theta(A))\\ &= 1-\max_{\theta\in D(\Theta)} P_\theta(A) = 1-\Pi(A)\end{aligned}$$

Donc Π et ν sont duales l'une de l'autre

$$\begin{aligned}\nu(A\cup B) &= \min_{\theta\in D(\Theta)} P_\theta(A\cup B) = \min_{\theta\in D(\Theta)} P_\theta(A)+P_\theta(B)-P_\theta(A\cap B)\\ &\geqslant \min_{\theta\in D(\Theta)} \max(P_\theta(A),P_\theta(B)) \text{ (car } P(A\cap B)\leqslant \min(P(A),P(B)))\\ &\geqslant \max(\min_{\theta\in D(\Theta)} P_\theta(A), \min_{\theta\in D(\Theta)} P_\theta(A)) = \max(\nu(A),\nu(B))\end{aligned}$$

$$\begin{aligned}\Pi(A\cap B) &= 1-\nu(\overline{A}\cap\overline{B}) \leqslant 1-\max(\nu(\overline{A}),\nu(\overline{B}))\\ &= \min(1-\nu(\overline{A}),1-\nu(\overline{B})) = \min(\Pi(A),\Pi(B))\end{aligned}$$

Pour les autres bornes, on sait que $\Pi(A\cup B)\geqslant \max(\Pi(A),\Pi(B))$, $\nu(A\cap B)\leqslant \min(\nu(A),\nu(B))$ (au lieu des égalités) et :

$$\begin{aligned}\Pi(A\cup B) = \max_{\theta\in D(\Theta)} P_\theta(A)+P_\theta(B)-P_\theta(A\cap B) &\leqslant \Pi(A)+\Pi(B)-\nu(A\cap B)\\ \implies \Pi(A\cup B) &\leqslant \min(1,\Pi(A)+\Pi(B)-1)\\ \nu(A\cap B) &\geqslant \nu(A)+\nu(B)-\Pi(A\cup B)\\ \implies \nu(A\cap B) &\geqslant \max(0,\nu(A)+\nu(B)-1)\end{aligned}$$

Comme le but est d'obtenir un encadrement correct de la probabilité, on pourrait utiliser ces inégalités pour calculer de tels encadrements et donner des estimations supérieures de Π et inférieures de ν. De plus, étant donné les signes dans les formules, on peut utiliser ces bornes pour calculer d'autres bornes, en remplaçant Π et ν par leur estimation, et cela donnera d'autres bornes correctes (mais on perd sans doute en précision à chaque fois).

De plus, si A et B sont des évènements indépendants (si pour tout x,θ, $P_\theta(A\cap B|x) = P_\theta(A|x)P_\theta(B|x)$), alors :

$$\begin{aligned}\nu(A)\nu(B)\leqslant \nu(A\cap B)\leqslant \Pi(A\cap B)\leqslant \Pi(A)\Pi(B)\\ \nu(A\cup B)\geqslant \nu(A)+\nu(B)-\nu(A)\nu(B)\\ \Pi(A\cup B)\leqslant \Pi(A)+\Pi(B)-\Pi(A)\Pi(B)\end{aligned}$$

2.3 Réseaux de neurones artificiels

Dans un réseau de neurones artificiels donnant en sortie des probabilités, l'entrée n'est pas nécessairement constituée de probabilités. Par exemple, dans la classification d'image, l'entrée est

assimilable à un «stimulus». La sortie des couches de convolution ou autres premières couches peut être vue comme un stimulus également, où comme une probabilité d'avoir détecté une propriété de l'image.

Ainsi, il faudra décider lors de la création du modèle d'une partie «détecteur», qui utilisera les techniques classique des réseaux de neurones artificiels mais donnera un résultat interprété comme des probabilités en sortie, et d'une partie «raisonneur» qui utilisera les possibilités et nécessités. La suite se concentrera sur ce «raisonneur».

En terme d'entraînement, il est possible d'entrainer le réseau de neurones directement avec les possibilités et nécessités, mais aussi de le pré-entrainer avec uniquement des probabilités, puis de le raffiner ensuite (en commençant avec $\Pi = p = \nu$).

2.3.1 Neurones

Un neurone classique prend la forme $y = \sigma(\sum_i w_i x_i + b)$ où σ est une fonction non-linéaire. Ainsi en utilisant les définitions précédentes, on peut calculer des estimations de Π et ν avec

$$a^\Pi = \sum_i w_i^{\Pi\Pi} x_i^\Pi + \sum_i w_i^{\Pi\nu} x_i^\nu + b^\Pi$$
$$a^\nu = \sum_i w_i^{\nu\Pi} x_i^\Pi + \sum_i w_i^{\nu\nu} x_i^\nu + b^\nu$$
$$y^\Pi = \sigma(a^\Pi) \quad y^\nu = \sigma(a^\nu)$$

Autrement dit, si on note $y^T = (y^\pi, y^\nu)$, $b \in \mathbb{R}^2$, $w = \begin{pmatrix} w^{\Pi\Pi} & w^{\nu\Pi} \\ w^{\Pi\nu} & w^{\nu\nu} \end{pmatrix}$, alors

$$y = \sigma(\sum_i w_i x_i + b_i)$$

Si on a tous les $x_i, x_i^\Pi, x_i^\nu \geqslant 0$, alors il suffit d'avoir $w_i^{\nu\Pi} \leqslant w_i \leqslant w_i^{\Pi\Pi}$, $w_i^{\nu\nu} \leqslant w_i \leqslant w_i^{\Pi\nu}$ et $b^\nu \leqslant b \leqslant b^\Pi$ et $\sigma : \mathbb{R} \to [0;1]$ pour avoir des possibilités et nécessités valides. Cependant, ce n'est pas la solution utilisée dans la suite.

On peut assurer que la sortie d'un neurone soit toujours dans $[0;1]$ en utilisant la fonction sigmoïde, ou $\frac{1+tanh(x)}{2}$ pour utiliser tanh.

Une autre solution serait d'utiliser une fonction «projecteur» qui prenne en paramètres a^Π et a^ν, et retourne un point situé dans le triangle, sans rien imposer sur les w_i ou la fonction de non-linéarité. Par exemple, $\tilde{\sigma}(a^\Pi, a^\nu) = (\sigma(a^\Pi), \sigma(a^\Pi)\sigma(a^\nu))$.

Pour la fonction softmax, on peut utiliser la définition suivante. Elle garde la propriété $\nu \leqslant p_0 \leqslant \Pi$, avec $\sum_i y_i^\nu \leqslant 1 \leqslant \sum_i y_i^\Pi$, ce qui permet entre autres d'en déduire une distribution de probabilité correspondante.

$$y_j^\Pi = \frac{e^{a_j^\Pi}}{\sum_i e^{a_i^\nu}}$$
$$y_j^\nu = \frac{e^{a_j^\nu}}{\sum_i e^{a_i^\Pi}}$$

Mais cela pourrait ne pas bien fonctionner (bien que cette méthode ait été efficace dans mes expérimentations). En effet, cette fonction permet d'avoir des valeurs de possibilité et nécessité au dessus de 1, donc en dehors de l'échelle de valeurs. De plus, elle gère mal l'incertitude. Par exemple, dans le cas extrême d'une seule sortie, on ne peut pas en avoir $\nu = 0$ et $\Pi = 1$.

Une autre possibilité est d'utiliser une autre fonction qui ne soit pas construite autour de l'encadrement des probabilités, mais qui fasse sens dans le cadre des possibilités et nécessités. Comme les possibilités classiques remplacent la somme par le max, on divise par le max des exponentielles des possibilités :

$$y_j^\Pi = \frac{e^{a_j^\Pi}}{\max_i e^{a_i^\Pi}}$$
$$y_j^\nu = \frac{e^{a_j^\nu}}{\max_i e^{a_i^\Pi}}$$

C'est cette dernière formule que j'ai utilisée dans mes expérimentations et elle fonctionnait tout à fait.

2.3.2 Fonction de coût

On va considérer $\hat{y}_i \in [0;1]$ les variables contenant la sortie attendue pour chaque neurone de sortie, y_i^π et y_i^ν les sorties effectives, et c une fonction de coût basée sur la différence entre $d = |y - \hat{y}|$. En reprenant les fonctions classiques des réseaux de neurones :

- Pour la norme L_2 (MSE) : $c(d) = d^2$. On peut implémenter les normes L_1, L_3... de la même façon.
- La fonction utilisée dans la suite : $c(d) = \log|1-d|$. Elle est inspirée de l'entropie croisée et prends la même valeur pour $\hat{y} = 0$ et $\hat{y} = 1$. Pour d'autres valeurs, elle fonctionne cependant correctement.
- Exponentielle : $c(d) = e^d$

La fonction de coût est paramétrée par $\alpha^{\Pi+}, \alpha^{\Pi-}, \alpha^{\nu+}, \alpha^{\nu-}$. Le but est de pénaliser en priorité le fait que $y_i^\Pi < \hat{y}_i$, et $y_i^\nu > \hat{y}_i$, car ces deux sorties représentent un intervalle dans lequel on doit trouver la sortie p en général (en considérant la probabilité comme étant la moyenne des $\hat{y}$ sur le même type d'entrée). On devra donc choisir $\alpha^{\Pi+} < \alpha^{\Pi-}$ et $\alpha^{\nu-} < \alpha^{\nu+}$.

$$\begin{aligned} l(z) = \frac{1}{|I|}\sum_{i\in I} &\alpha^{\Pi+} c(\max(y_i^\Pi - \hat{y}_i, 0)) + \alpha^{\Pi-} c(\max(\hat{y}_i - y_i^\Pi, 0)) \\ &+\alpha^{\nu+} c(\max(y_i^\nu - \hat{y}_i, 0)) + \alpha^{\nu-} c(\max(\hat{y}_i - y_i^\nu, 0)) \end{aligned}$$

On peut simplifier cette fonction avec un unique meta-paramètre $\alpha < 1$:

$$\begin{aligned} l(z) = \frac{1}{|I|}\sum_{i\in I} &\alpha\ c(\max(y_i^\Pi - \hat{y}_i, 0)) + c(\max(\hat{y}_i - y_i^\Pi, 0)) \\ &+c(\max(y_i^\nu - \hat{y}_i, 0)) + \alpha\ c(\max(\hat{y}_i - y_i^\nu, 0)) \end{aligned}$$

2.3.3 Assurer les contraintes

En plus d'éventuels termes de régularisation, on peut ajouter des termes de contrainte ayant un facteur de poids β suffisamment important pour que cela encode une vrai contrainte, mais pas trop grand pour que les cas dégénérés pouvant se présenter dans un ensemble de données ne viennent pas perturber l'apprentissage. On suppose que l'on dispose d'une fonction de coût c_0 (qui peut changer selon la couche ou le coût) et d'un paramètre β. Lors de ce stage, j'ai simplement choisit $c_0(x) = x^2$.

Pour assurer que les possibilités et nécessités respectent bien $\nu < \Pi$, au lieu d'imposer des contraintes sur les w_i, j'ai utilisé une contrainte sur la sortie, en ajoutant à la fonction de coût le terme suivant.

$$\beta \sum_{\text{couche } o,\ i} c_0(max(0, y^{\Pi}_{o,i} - y^{\nu}_{o,i}))$$

De plus, sur les exemples connus, le réseau doit tenter d'apprendre un couple nécessité-possibilité étroit, et large sur les exemples inconnus. Cela peut être simulé avec les termes suivants, en prenant β_+ et β_- positifs mais faibles, et β_+ assez grand pas rapport à β_-, avec M une marge dans $[0; 1]$. Cependant, cela n'a pas suffi sur les expérimentations que j'ai faites, et devrait sûrement être amélioré. La première contrainte est tout de même gardée dans la suite, car elle permet d'avoir un certain couplage entre ν et Π dans un même neurone.

$$\beta_+ \sum_{\text{couche } o,\ i} c_0(y^{\Pi}_{o,i} - y^{\nu}_{o,i})$$

$$\beta_- \sum_{\text{neurone } i} c_0(\max(\sigma(b^{\nu}_i + \sum_i |w^{\nu\nu}_{i,j}| + |w^{\nu\Pi}_{i,j}|) - M, 0)) + c_0(\max(1 - \sigma(b^{\Pi}_i - \sum_i |w^{\Pi\nu}_{i,j}| + |w^{\Pi\Pi}_{i,j}|) - M, 0))$$

2.3.4 Mesure de la précision

Si la sortie attendue est une probabilité voire un intervalle possibilité-nécessité, il faudra définir une distance δ telle que chaque variable de sortie doivent être à une distance d'au plus δ de la vrai sortie pour être acceptée.

Cependant, il existe quatre points intéressants systématiquement. Si on s'intéresse à des cas plus simples où chaque propriété de sortie peut être étiquetée «vrai», «faux», «aléatoire», ou «inconnu », alors on peut utiliser ces points. «vrai» correspond au point $(1, 1)$, «faux» à $(0, 0)$, «aléatoire» à $(1/2, 1/2)$ et «inconnu» à $(0, 1)$. On peut associer à une paire de sortie (y^{ν}_i, y^{Π}_i) le point le plus proche, et accepter une sortie y si tous les points correspondent.

Dans le cas où on ne se soucie pas du point «aléatoire», on peut procéder autrement et arrondir y^{ν}_i et y^{Π}_i au plus proche, pour obtenir l'un des trois autres points. Cela donne une manière efficace en terme de calcul pour estimer le nombre de résultats corrects, et c'est ce que j'utilise dans l'application à MNIST dans la suite.

3 Applications expérimentales

J'ai choisi d'implémenter ces modèles en utilisant Keras et TensorFlow. J'ai implémenté les différentes types de couches utilisant possibilité et nécessité sous forme de nouvelles couches Keras, en utilisant les opérations de TensorFlow, pour pouvoir les utiliser de façon modulaire comme les couches standard. De même, j'ai implémenté l'équivalent des fonctions de coût et de précision standard selon ce qui est expliqué dans la partie plus haut.

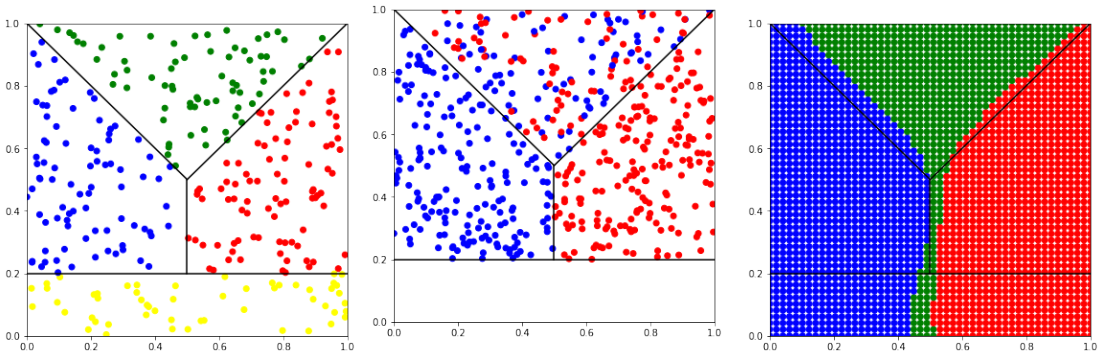
3.1 Données synthétiques

Une première étape pour valider le modèle est de l'appliquer à des données synthétiques très simples, générées par une petite fonction et pouvant facilement être représentées.

3.1.1 Données et entrainement

Les données d'entrée sont deux valeurs $x, y \in [0; 1]$, que l'on peut interpréter comme deux propriétés. Chaque point de sortie est associé à deux propriétés, bleu et rouge, qui sont complémentaires l'une de l'autre : un point est soit bleu, soit rouge. Le carré de toutes les valeurs d'entrées est découpé en quatre zones : la zone bleu, où les points sont bleus, la zone rouge, la zone verte où la couleur est aléatoire ; et la zone jaune où il n'y a pas de données, donc la couleur ne peut pas être connue. La seconde image ci-dessous montre un exemple de données d'entrée.

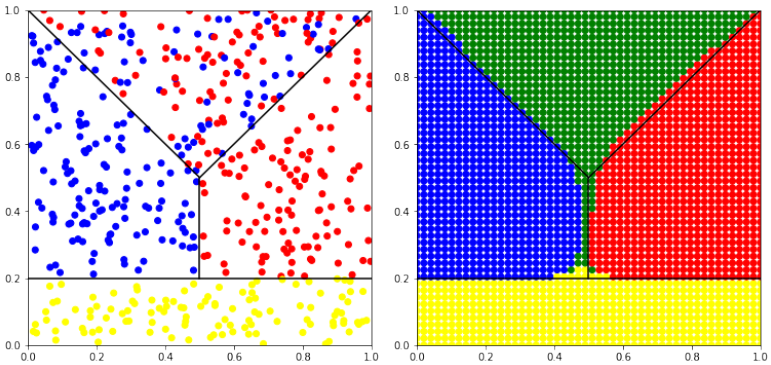
Un modèle très simple est utilisé, composé de trois couches de petite taille, et sans fine-tuning manuel, expliquant sans doute les zones d'imprécisions. Le résultat prédit correspond à la troisième image. On observe que le modèle généralise naturellement en dehors des bornes. Même en ajoutant des contraintes sous forme de coût supplémentaire comme décrit ci-dessus (ou avec d'autres formules), cela ne permettait pas de régler ce problème.



(a) Les zones (b) Exemple d'ensemble de données (c) Prédictions

3.1.2 Zone d'inconnu

Au vu de l'échec du coût supplémentaire pour apprendre au modèle à répondre qu'il ne sait pas sur les zones inconnues, une autre solution plus manuelle est de lui présenter des données qu'il ne doit pas savoir classer, et de les lui faire apprendre, ce qui revient à simplement donner une 3ème étiquette possible à des points. Les exemples d'entrée sont donc étiquetés «bleu», «rouge», ou «inconnu». Cela donne le résultat ci-dessous,



(a) Données d'entrée (b) Prédictions

3.1.3 Interprétation des neurones

L'utilisation des possibilités devrait pouvoir permettre de mieux interpréter les neurones et leur sortie, et on peut commencer par visualiser la sortie de chaque neurone pour chacun des points du plan. Ainsi sur la figurine suivante, pour chaque couche du réseau, il y a deux lignes de vignettes. La première représente les nécessités, la seconde les possibilités, en fonction de l'entrée (x, y). Le violet correspond à une sortie de 0 (pas d'activité), et le jaune à 1, tendis que le vert représente $\frac{1}{2}$.

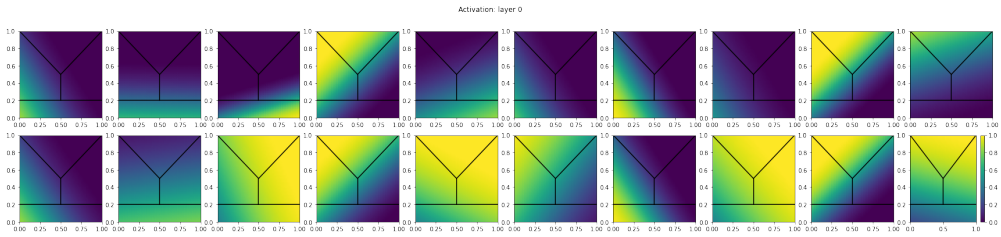


FIGURE 5 – Activité de la couche cachée 1 en fonction de l'entrée x et y

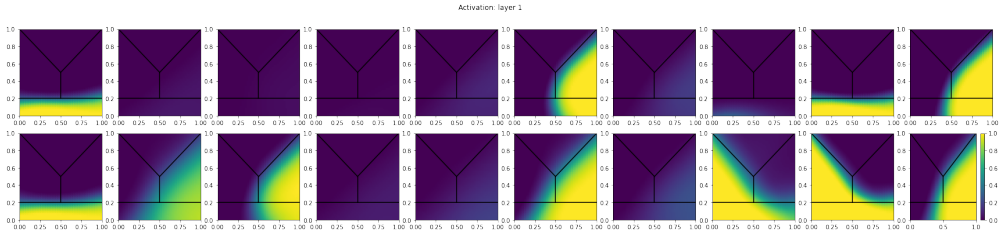


FIGURE 6 – Activité de la couche cachée 2 en fonction de l'entrée x et y

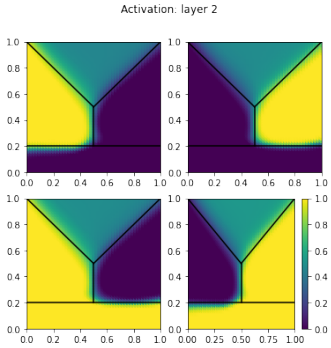


FIGURE 7 – Activité de la couche de sortie en fonction de l'entrée x et y

Le réseau de neurone fonctionne relativement correctement, car il donne des valeurs correctes sur la quasi-totalité des zones où il y a eu des échantillons, si ce n'est une particularité étrange à cause de la marge entre vrai et faux au centre. Il atteint bien le but, c'est à dire d'être capable sur un unique neurone de sortie de distinguer aléatoire, possible, certain et impossible.

On remarque qu'il y a un certain nombre de neurones peu intéressants, voire de neurones où possibilité et nécessité sont toujours égaux, ou des neurones où seule une des deux sorties est utile. Cependant certains tirent avantage du couplage et dessinent deux zones différentes que l'on peut interpréter.

3.2 Application à MNIST

Le jeu de données MNIST semble aujourd'hui très élémentaire, mais au vu de la difficulté que j'ai eu à faire fonctionner ce type de modèle, il permettait d'avoir un exemple un peu plus concret, avec un passage entre des données représentant des «signaux» (blanc / noir / intermédiaire), à un ensemble de nécessités et possibilités en sortie.

Pour représenter l'incertitude, j'ai découpé les données d'entrée en trois ensembles : connu (chiffres 0 à 3), inconnu rencontré (chiffres 4 à 6), et inconnu jamais vu (chiffres 7 à 9). La sortie consiste, pour chaque chiffre entre 0 et 3, en une possibilité et une nécessité que ça soit ce chiffre-là. Les chiffres 4 à 6 sont donnés en entrée, mais une réponse de $(0, 1)$ est attendue, tandis que les chiffres de 7 à 9 sont retirés des données d'entraînement (mais pas de test). De plus, des données aléatoires sont fournies en entrée, associées à la sortie $(0, 1)$ (inconnu).

3.2.1 Perceptron multicouche

Le réseau le plus simple à mettre en place était un perceptron multi-couche. Le modèle dispose de 3 couches entièrement connectées, utilise $x \mapsto \frac{1+\tanh(x)}{2}$ comme fonction non-linéaire, et la version modifiée du softmax basée sur le max pour la sortie. Les résultats obtenus sont les suivants :

Données d'entrainement	Succès entrainement	Succès tests	Test chiffres 0 à 3	Test chiffres 4 à 6	Test chiffres 7 à 9
Chiffres 0-3	94.5%	39.6%	95.1%	0%	0%
Chiffres 0-3 + aléatoire	97.7%	40.4%	96.7%	0.4%	0.3%
Chiffres 0-7 sans contraintes	94.3%	92.9%	92.8%	95.7%	64.4%
Chiffres 0-7	92.6%	85.2%	91.6%	94.3%	67.7%
Chiffres 0-7 + aléatoire sans contraintes	96.3%	83.9%	93.1%	95.6%	60%
Chiffres 0-7 + aléatoire	95.9%	83.2%	92.7%	95.4%	58.4%

Il semble donc que les contraintes n'aient pas beaucoup d'impact sur la qualité des résultats (mais elles permettent d'assurer la cohérence des sorties des neurones internes, ce qui est une bonne raison de les utiliser). Les données aléatoires ne sont pas utiles pour améliorer les résultats, mais les données supplémentaires (chiffres 4 à 6) permettent au réseau de généraliser le fait de ne pas savoir classer certains chiffres, bien que peu efficacement.

Si on change la manière de diviser les données, en entraînant sur tous les chiffres sauf 9 (mais la sortie concerne toujours les chiffres 0 à 3), alors les chiffres 0 à 3 sont reconnus à 87.8%, les chiffres 4 à 8 identifiés comme inconnus dans 95.3% des cas, et le chiffre 9 dans 97.8% des cas sans avoir été vu. Ainsi, avec suffisamment de données non-labellisées, la distinction entre connu et inconnu semble se généraliser assez bien.

3.2.2 Réseau convolutionnel

Les réseaux probabilistes convolutionnels fonctionnent très bien sur ce jeu de test, même en étant très simples. J'ai cherché à implémenter l'équivalent avec les possibilités et nécessités, avec

la même structure de réseau, mais les résultats en utilisant des couches de convolution possiblistes sont au mieux très mauvais, au pire signes d'une absence totale d'apprentissage.

Au vu de la durée du stage, je n'ai pas pu explorer plus en avant. En effet ces résultats sont sans doute provisoires et montrent qu'il faudrait retravailler plus en profondeur l'architecture du réseau et la manière d'intégrer possibilités et nécessités avec une convolution pour ne pas bloquer l'apprentissage.

4 Conclusion

Le sujet est intéressant mais semble complexe au vu des expérimentations préliminaires que j'ai réalisées. Différentes autres approches pourraient être possibles :

- Utiliser une structure de réseau adaptée comme certains réseaux neuro-symboliques.
- Utiliser des fonctions beaucoup plus éloignées des réseaux classiques se basant sur min ou max (avec des approximations prenant en compte les autres variables).
- Chercher à introduire de meilleurs manières d'encoder les contraintes, plus efficaces.
- Trouver un moyen plus efficace que d'utiliser des données réelles non labellisées pour apprendre à ne pas généraliser excessivement.
- On pourrait entourer la vraisemblance au lieu de la probabilité, comme proposé dans [**?**], avec $\Pi(O) = \max_{o \in O} P(x|o)$ et $\nu(O) = \min_{o \in O} P(x|o)$.

L'efficacité de ce qui a été implémenté durant ce stage pourrait également être mieux observée en testant sur des données incluant plus d'incertitude. Il m'a été proposé de tenter de mettre en place une expérience à partir de données venant de l'action AIDE (une expérience avec des cubes et des données sur leurs affordances, etc...), mais réussir à rendre ce type de modèle fonctionnel sur ce type de problème nécessiterait encore un peu plus de travail. Pour étendre le dernier test et voir si l'apprentissage visant à éviter de généraliser à des données inconnues fonctionne bien, utiliser un ensemble de données ayant plus de classes (comme omniglot) pourrait être utile.

Références

[Benferhat14] S. Benferhat (CRIL), Th. Denoeux (Heudiasyc), D. Dubois (IRIT), H. Prade (IRIT), *Représentations de l'incertitude en intelligence artificielle*, 2014

[Denœux20] Thierry Denœux, Didier Dubois, et Henri Prade, *Representations of Uncertainty in Artificial Intelligence : Probability and Possibility*, 2020

[Tettamanzi17] Andrea Tettamanzi, Catherine Faron Zucker et Fabien Gandon, *Possibilistic testing of OWL axioms against RDF data.* International Journal of Approximate Reasoning, Elsevier, 2017, 10.1016/j.ijar.2017.08.012 . hal-01591001

[Ishibuchi91] Hisao Ishibuchi, Ryosuke Fujioka et Hideo Tanaka, *Possibility and necessity pattern classification using neural networks*, 1991

[Drago199] G.P. Drago1 et S. Ridella2, *Possibility and Necessity Pattern Classification using an Interval Arithmetic Perceptron*, 1999

[Garcez20] Artur d'Avila Garcezand Luis C. Lamb, *Neurosymbolic AI : The 3rd Wave*, 2020

[Serafini16] Luciano Serafini et Artur d'Avila Garcez, *Logic Tensor Networks : Deep Learning and Logical Reasoning from Data and Knowledge*, 2016

[Donadello17] Ivan Donadello, Luciano Serafini et Artur d'Avila Garcez, *Logic Tensor Networks for Semantic Image Interpretation*, 2017

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399