Réalisation d'un réseau de neurones pour prédire la résistance mécanique d'un verre et sa formulation: GLASSNET

1st ROBIN Eric
Maître de Conférences
Institut Physique de Rennes
Rennes, FRANCE
eric.robin@univ-rennes.fr

2nd DEPIN Emeric
Mécatronique
Rennes, FRANCE
emeric.depin@ens-rennes.fr

3rd KERIBIN Jérôme
Mécatronique
Rennes, FRANCE
jerome.keribin@ens-rennes.fr

4th QUESTE Adrien
Mécatronique
Rennes, FRANCE
adrien.queste@ens-rennes.fr

***Abstract*—Cet article présente le *machine learning* de manière très succincte et explique le potentiel de l'outil appliqué à la prédiction des propriétés mécaniques d'un verre, le $Li_2S - P_2S_5$. Nous verrons au cours de cette étude que le *machine learning* possède un potentiel très grand dans de nombreux domaines. Dans un premier temps, nous verrons qu'il est possible simplement de faire "identifier" à notre réseau de neurones une loi mathématique entrée-sortie inconnue à l'avance. Nous verrons ensuite les différents paramétrages et limites de cet outil. Enfin, nous allons utiliser cet outil pour tenter de prédire les propriétés mécaniques d'un verre spécifique, dont les propriétés sont peu connues en raison de la difficulté à mettre en œuvre expérimentalement ces mesures, d'où l'intérêt de la méthode présentée ici.**

***Index Terms*—Mécanique, Verre, *Machine learning*, Python**

I. Introduction

Le *machine learning* par réseau de neurones d'abord conceptualisé par McCulloch, Warren S., et Walter Pitts. [1], puis mis en œuvre pour la première fois par Rosenblatt, Frank. dans sa conception d'un réseau de neurones appelé le perceptron [2] est aujourd'hui un outil de plus en plus utilisé en effet il est très attractif notamment grâce à sa capacité d'apprentissage hors du commun. Il apparaît ainsi comme une intelligence artificielle cependant nous verrons que les réussites d'analyses du *machine learning* dépendent de nombreux paramètres intrinsèques issus lors de l'entraînement du réseau de neurones. En effet, parfois même si le réseau de neurones semble bien "comprendre" la situation, il peut tout de même avoir des difficultés.
Nous allons donc analyser nos résultats avec précaution. Néanmoins, nous verrons que dans les limites de la base de données que l'utilisateur lui a fournies, celui-ci est très performant et que ces capacités diminuent fortement en s'éloignant de celle-ci.

Ainsi tout d'abord, nous allons présenter nos résultats concernant l'apprentissage d'un réseau de neurones à l'arithmétique de base : les opérations élémentaires $+$, $-$, $\times$, $\div$ et notamment l'utilité de se servir d'un tel modèle plutôt que des simples courbes de tendance; nous aborderons également les techniques pour améliorer la convergence du modèle ainsi que la mise en évidence de la faiblesse du modèle sur cet exemple.
Ensuite, nous étudierons les méthodes de calculs utilisées par le réseau et nous mettrons en évidence le fait qu'un réseau de neurones est très éloigné d'une intelligence artificielle.
Enfin, nous verrons que cet outil permet d'obtenir tout de même une intuition des propriétés mécaniques d'un matériau quelconque avec peu de données initiales.

II. Initiation d'un réseau de neurones aux opérations arithmétiques

Tout d'abord, ce qui a motivé l'utilisation d'un réseau de neurones est que nous ne connaissions pas a priori la forme de loi suivie par le module de Young ou le coefficient de Poisson en fonction du pourcentage de fraction molaire d'une espèce présente dans notre composé chimique (la fraction de la deuxième complétant l'espèce afin d'atteindre 100%). Le verre étudié ici, le $Li_2S - P_2S_5$ n'étant composé que de deux espèces, le Li_2S et le P_2S_5, par la suite c'est la première qu'on prendra systématiquement en variable d'entrée car la deuxième variable dépend de la première ($x_{P_2S_5} = 1 - x_{Li_2S}$).
L'avantage d'utiliser un réseau de neurones est alors de pouvoir déterminer une loi entrée-sortie sans la connaître au préalable. En effet, sans postuler d'expression mathématique générale à cette loi physique, il est impossible de la faire coïncider parfaitement avec une expression mathématique usuelle, ce que nous ne pouvions pas faire, le nombre de données expérimentales étant trop restreint pour pouvoir espérer faire un postulat pertinent.
Ainsi, la force du réseau de neurones c'est d'être capable de déterminer une loi entrée sortie coïncidant avec les données d'entraînement en ayant fait au préalable un nombre beaucoup moins grand de postulats, sur la forme générique de la loi qu'on cherche à ajuster sur nos données expérimentales.
On peut penser à l'exemple le plus simple de ce type de postulat dans la pratique très courante de régression linéaire, qui consiste à ajuster les coefficients d'une droite afin de minimiser l'écart avec les données expérimentales, par le

biais d'une fonction de coût, le plus couramment l'erreur quadratique. Dans ce cas classique on part du principe que seul deux coefficients, dans une relation linéaire permettent d'ajuster une loi convenable aux données expérimentales, ce qui est extrêmement contraignant).
Ensuite, concernant la structure utilisée de notre réseau de neurones nous avons utilisé la structure usuelle parmi toutes celles possibles de la zoologie infinie des réseaux de neurones afin de prédire une loi mathématique comme nous le souhaitons : cette structure est celle d'un perceptron multicouches, avec plusieurs couches de neurones en cascade, dont la représentation schématique est visible sur la figure 1. On peut noter que comme on va postuler a priori d'une loi mathématique, on postule a priori d'une structure de réseau de neurones, on n'échappe donc pas totalement aux conjectures et postulats.

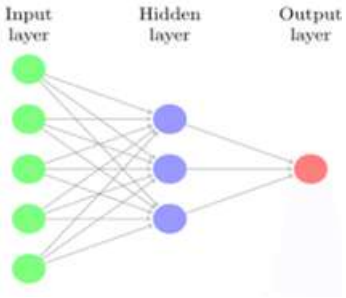

Fig. 1: Réseau neuronal de type perceptron (1 couche cachée)

Une première couche de neurone est visible, c'est la couche d'entrée. Elle est liée à une ou plusieurs couches cachées en instaurant des poids sur les valeurs entre chaque neurone. Enfin ces couches cachées amènent à la couche de sortie qui est la ou les données que l'utilisateur souhaite récupérer. Ainsi le réseau de neurones permet de relier un lot de données d'entrée à une sortie.

A. *Poids*

Pour établir ce lien entre l'entrée et la sortie, un réseau de neurones procède couche par couche. Chaque liaison entre les neurones de la couche i et ceux de la couche $i+1$ sont pondérés par des poids $w_{j,k}$ pour tout $j \in [\![1; N_i]\!]$ et $k \in [\![1; N_{i+1}]\!]$ (où N_i est le nombre de neurones de la couche i, et donc $w_{j,k}$ est le poids de la liaison entre le neurone j de la couche i et le neurone k de la couche $i+1$) de telle sorte que si, pour tout $j \in [\![1; N_i]\!]$ le neurone j a en sortie le scalaire a_j alors pour tout $k \in [\![1; N_{i+1}]\!]$, le neurone k a alors en entrée le scalaire $\Sigma_{j=1}^{N_i} w_{j,k} \times a_j$.
Plusieurs choses sont à noter sur cette méthode. Tout d'abord ces poids sont initialement attribués de manière aléatoire qu'il conviendra alors d'optimiser ensuite par la méthode de rétropropagation du gradient que nous détaillerons dans un des paragraphes ci-dessous. Ensuite, on distingue l'entrée d'un neurone de sa sortie car celui-ci va évaluer son entrée par une fonction dite d'activation, nomenclature rappelant les neurones biologiques s'activant pour renforcer une synapse, liaison entre ce dit neurone et un autre neurone. Ces fonctions et leur rôle vont être détaillés dans la section suivante.

B. *Fonctions d'activation*

Les fonctions d'activation sont très variées on peut citer parmi elles les sigmoïdes, rampes ou identités, portes ou fonction de Heaviside ou encore, dans notre cas des unités de rectification linéaire aussi appelées ReLu. Sont présentées figures 2, 3 ,4 et 5 les graphes des fonctions d'activation mentionnées en exemple précédemment.

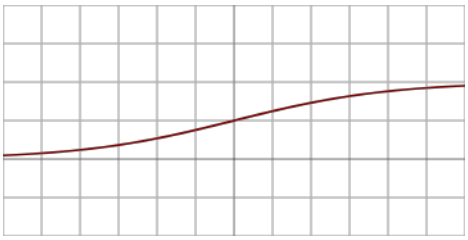

Fig. 2: Graphe de la fonction sigmoïde, $f(x) = \frac{1}{1+e^{-x}}$

Fig. 3: Graphe de la fonction rampe, $f(x) = x$

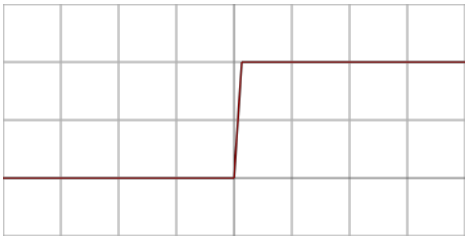

Fig. 4: Graphe de la fonction porte, $f(x) = 0$ si $x < 0$ et $f(x) = 1$ sinon

Fig. 5: Graphe de la fonction ReLu, $f(x) = max(0, t)$

Une fonction d'activation est choisie selon différents critères qui ne seront pas tous détaillés dans ce document. Parmi

ces différents critères se trouvent l'étendue de l'image de la fonction, sa valeur en 0, sa monotonie (ou non), la monotonie (ou non) de sa dérivée.
Deux critères sont toutefois très importants et nous allons voir maintenant pourquoi, le premier est la différentiabilité partout afin de permettre d'utiliser des méthodes liées au gradient. Le deuxième est le plus important de tous, la non-linéarité de la fonction d'activation, en effet, le concept de fonctions d'activations n'a été introduit qu'à fin d'introduire de la non-linéarité dans les équations sous-jacentes des réseaux de neurones et donc de résoudre des problèmes non-linéaires, ce qui n'était pas le cas dans leur conception originale, lors de l'invention du perceptron, qui avait alors même si cette phrase est donc anachronique puisqu'il pré-date l'invention du concept de fonction d'activation, uniquement des fonctions d'activation rampe.
Cette non-linéarité des fonctions de rétropropagation a toutefois historiquement posé un problème puisque qu'on ne pouvait plus utiliser les méthodes de descente de gradient classiques, ce qui a amené ensuite à la popularisation de la méthode de rétropropragation du gradient grâce aux travaux de Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. [3] [4], le fonctionnement de cette méthode va alors être détaillé dans le paragraphe suivant.

C. Méthode de la rétropropagation du gradient

Ce processus d'optimisation des poids des liaisons est un processus itératif. Ainsi, on peut définir une itération, appelée époque qui consiste en un calcul de l'entrée vers la sortie du réseau de neurone des différentes valeurs d'entrées et de sorties des différents neurones suivie d'une rétro-propagation du gradient qui à l'aide de la méthode de la descente gradient vient modifier et, si les conditions sont favorables, améliorer la valeur des poids. On parle ici de rétropropagation du gradient, car on se sert de la règle de la chaîne et de la méthode du gradient classique pour propager les optimisations effectuées de l'entrée vers la sortie.
En effet, par exemple si on cherche à optimiser les poids du réseau de neurones suivant pour une donnée d'entraînement (X, Y, Z_{out}) :

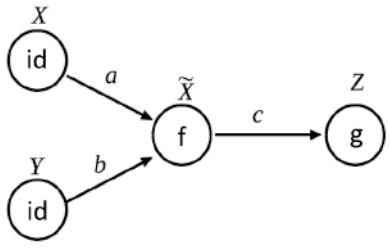

Fig. 6: Exemple de réseau de neurone

On a les équations suivantes :

$$Z = g(c \times f(\tilde{X}))$$
$$\tilde{X} = a \times X + b \times Y$$

Ainsi avec une méthode de descente de gradient, en appelant E la fonction de coûts qui est alors fonctions de tous les poids a, b et c, des entrées X, Y et $\tilde{X}$ (même si pas de manière indépendante, et c'est en fait une clé de la rapidité de l'algorithme et ce qui lui donne son nom), par exemple en prenant comme fonction de coût l'erreur quadratique $E = (Z_{out} - Z)^2$:

$$c' = c - \frac{\partial E}{\partial c}(\tilde{X})$$

On remarque déjà que calculer la dérivée partielle de E par rapport à $\tilde{X}$ peut s'avérer compliqué, on applique alors la règle de la chaîne, qui donne la relation suivante :

$$\frac{\partial E}{\partial c}(\tilde{X}) = \frac{\partial E}{\partial Z}(c) \times \frac{\partial Z}{\partial c}(\tilde{X})$$

Qui donne alors finalement, en l'injectant dans l'équation précédente :

$$c' = c - \frac{\partial E}{\partial Z}(c) \times \frac{\partial Z}{\partial c}(\tilde{X})$$

Où ici, contrairement à la dérivée partielle précédente, chacune des dérivées partielles à calculer est relativement simple.

On fait ensuite de même pour a et b

$$a' = a - \frac{\partial E}{\partial a}(X)$$
$$b' = b - \frac{\partial E}{\partial b}(Y)$$

De même on applique la règle de la chaîne, afin de pouvoir injecter les relations obtenues, produit de dérivées partielles plus directes à calculer dans la relation d'itération ci-dessus :

$$\frac{\partial E}{\partial a}(X) = \frac{\partial E}{\partial Z}(c) \times \frac{\partial Z}{\partial c}(\tilde{X}) \times \frac{\partial \tilde{X}}{\partial a}(X)$$
$$\frac{\partial E}{\partial b}(Y) = \frac{\partial E}{\partial Z}(c) \times \frac{\partial Z}{\partial c}(\tilde{X}) \times \frac{\partial \tilde{X}}{\partial b}(Y)$$

Et c'est a cette étape qu'on se rend compte, que réapparaissent des dérivées partielles déjà calculées lors de l'itération sur c, $\frac{\partial E}{\partial Z}(c)$ et $\frac{\partial Z}{\partial c}(\tilde{X})$, d'où le sens des itérations sur les poids de la sortie vers l'entrée du réseau de neurones, afin de se resservir des calculs déjà effectués et donc ce nom, au premier abord cryptique de rétropropagation du gradient.

Dans la méthode du gradient, un des point phare a toutefois été omis durant la mise en œuvre de l'exemple précédent, en effet lors d'une itération, on a fait $c' = c - \frac{\partial E}{\partial Z}(c) \times \frac{\partial Z}{\partial c}(\tilde{X})$, une relation plus générale serait $c' = c - \gamma \times \frac{\partial E}{\partial Z}(c) \times \frac{\partial Z}{\partial c}(\tilde{X})$ ou γ est la vitesse d'apprentissage. Cette vitesse d'apprentissage qui est un paramètre très important du *machine learning* et de la méthode du gradient en général est donc l'objet du paragraphe suivant.

D. vitesse d'apprentissage

Tout d'abord, la figure ci-dessous illustre parfaitement ce propos du gradient descendant et de l'importance du choix de la vitesse d'apprentissage ou facteur d'apprentissage.

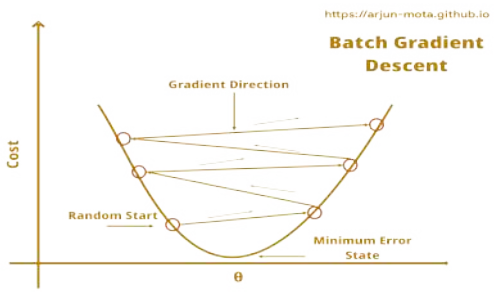

Fig. 7: Augmentation de l'erreur due à une vitesse d'apprentissage trop élevée

Ce graphique illustre parfaitement le problème que l'on peut rencontrer dans un problème de minimisation par itération. En effet, on part d'un point qui est déjà très proche de notre minimum cependant si la vitesse d'apprentissage est trop élevée alors on ne va pas diminuer l'erreur sur nos poids mais stagner ou pire l'augmenter.
Ainsi, si l'on considère une époque initialement les poids sont attribués de manière aléatoire ainsi très souvent en répétant des époques on améliore très facilement la convergence du modèle.
Cependant, à partir d'un certain moment, il est tout à fait possible de rencontrer des problèmes dus justement à la vitesse d'apprentissage. Si cette vitesse est trop élevée notre gradient ne va plus diminuer mais l'on va simplement tourner autour du point le plus faible sans jamais l'atteindre.
Ainsi, pour réaliser un réseau de neurones précis, il ne faut pas simplement avoir un grand nombre d'époques, il faut également avoir une vitesse d'apprentissage adaptée, soit de plus en plus faible à mesure qu'on se rapproche du minimum. Néanmoins, si l'on considère ces deux options le temps de calcul explose. Ainsi, pour débuter, quelques époques ainsi qu'une vitesse pas trop faible permettent de se rendre compte de la convergence du modèle.
Par exemple, sur le graphique 8 suivant traçant l'écart entre le réseau de neurones et la valeur réelle de sortie (pour notre apprentissage de l'arithmétique simple) on peut voir qu'à partir de l'époque 10 le modèle semble ne plus converger.
Ainsi, c'est bien la preuve qu'un nombre infini d'époques n'est pas une solution envisageable pour avoir une convergence mais qu'il faudrait aussi diminuer la vitesse d'apprentissage afin de pouvoir encore se rapprocher du minimum.

Ainsi, la vitesse d'apprentissage est un paramètre clef cependant d'autres paramètres sont aussi très important comme le choix du nombres de neurones.

E. Nombres de neurones

La structure du réseau de neurone a été définie précédemment cependant le nombre de neurone en entrée est

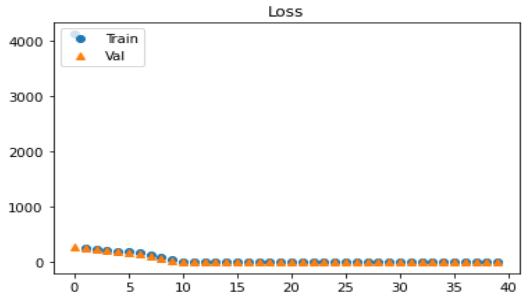

Fig. 8: Erreur selon le nombre d'époques

aussi un autre paramètre à prendre en compte. En effet, pour notre système nous avons choisi 128,64,32 puis 4 neurones de sortie $(+, -, *, /)$. Nous avons ensuite complexifié le réseau en ajoutant 256 neurones en entrée, puis 512 puis 1024. Nous avons vu que la convergence augmentait plus rapidement cependant le temps de calcul était lui aussi devenu beaucoup plus long. Ainsi comme pour la vitesse d'apprentissage, il ne faut pas avoir un réseau de neurones très complexe initialement.
En effet, il vaut mieux commencer par un réseau de neurones simple, ajuster les autres paramètres, regarder la convergence et l'améliorer en complexifiant le réseau et ainsi de suite, le nombre de neurones augmentant dramatiquement le temps de calcul, il peut être inutilement chronophage de s'attarder sur cette variable d'ajustement avant d'autres comme le facteur d'apprentissage par exemple.

F. Données

Les données que l'on entre au réseau de neurones sont certainement la chose la plus cruciale à considérer.
En effet, même si un réseau de neurone semble avoir une erreur très faible ou même s'il semble répondre correctement dans la plage de données où l'utilisateur l'a entraîné. Ce n'est pas pour cela qu'il a "compris" la loi mathématique reliant l'entrée à la sortie. En effet, pour apprendre globalement comment fonctionner un réseau de neurone nous avons dans premier essayer de faire apprendre à notre réseau de neurone l'addition.
Au début, pour vérifier son apprentissage nous avons donc fourni des nombres de la base de données (nombre de 10 à 20) à l'algorithme et nous avons pu conclure de l'efficacité du réseau sur ces données en vérifiant que les prédictions du réseau coïncidant. Cependant, quand nous avons étendu ce test à des chiffres négatifs ou à des chiffres avec de grands ordres de grandeur. Nous avons vu que l'algorithme ne comprenait pas du tout l'addition mais qu'il réussissait seulement à la transcrire correctement sur les valeurs de la bases de données. En effet, dès lors que l'on a rajouté les nombres négatifs, le réseau de neurones était cette fois capable de réussir une addition correcte, on s'était donc heurté à un problème d'extrapolation incorrecte, la plage de valeurs sur laquelle on avait entraîné le réseau de neurones était en quelques sortes sa zone de confort. Une fois cela compris, nous

avons donc essayé d'entraîner notre réseau de neurones à la soustraction, à la multiplication et à la division. Cependant nous avons rencontré des problèmes d'une nature différente comme l'atteste le tableau ci-dessous.

a+b réel	a/b réel	a+b simulé	a/b simulé
262.706	0.689663	270.225	4.25847
379.117	1.1115	385.767	15.9186
256.841	0.686931	264.226	4.09586
291.795	0.805572	297.969	7.75617
320.909	0.637072	331.51	3.54087
323.312	1.60324	335.173	13.0257
300.654	1.12746	305.942	12.6225
230.673	1.01628	233.438	9.58669
340.933	1.14027	347.212	14.3056
304.126	0.760824	311.467	6.9168

TABLE I: Résultats d'un réseau de neurones entraîné pour faires des additions et des soustractions. Les résultats de l'addition sont assez proche du résultat réel, ce qui n'est pas le cas des résultats pour la division

On peut remarquer effectivement le réseau maîtrise bien l'addition l'écart n'est pas très grand entre les valeurs simulées et réelles.
Cependant, à cause de la nature de la division, et de la multiplication, deux chiffres peuvent renvoyer des résultats bien différents. En effet, la division peut renvoyer des résultats très proches de zéro comme des valeurs très élevées. Et c'est cela qui est très difficile pour le réseau de neurones puisque l'on peut passer de valeurs faibles (a et b entre 1 et 20) a des valeurs beaucoup plus petites (1/20 1/19) jusqu'à 20. Ce qui fait que le réseau de neurones a du mal à avoir un apprentissage correct car les valeurs fluctuent beaucoup trop. Une des solutions pour palier à ce problème est la normalisation.

La normalisation dans notre cas pour le *machine learning* se résume tout d'abord à rapporter la valeur minimale à 0 et la valeur maximale à 1. Pour se faire, on applique la formule de normalisation suivante

$$x_{transformée} = \frac{x_{max} - x_{réel}}{x_{max} - x_{min}}$$

Nous avons un intérêt à normaliser ces données car dans notre réseau de neurones les données et les poids sont évalués par une fonction d'activation. Dans notre cas, la fonction d'activation était une fonction ReLu ($max(0, t)$). Cependant cette fonction croit donc linéairement entre 0 et t, comme dans notre application précédente avec la multiplication et la division les valeurs évoluent vers de très grands chiffres cela cause des problèmes dans la gestion des données car cela devient trop complexe pour l'ordinateur. Ainsi, le processus de normalisation permet de faciliter les calculs pour l'ordinateur et également de faciliter l'efficacité de la méthode du gradient descendant tout en étant facilement réversible, la loi de normalisation étant une relation linéaire.

III. Réalisation du réseau de neurones permettant de prédire la résistance mécanique d'un verre

A. Réalisation du réseau de neurones

Pour réaliser le réseau de neurones, la librairie TensorFlow (la version 2.1.0 était utilisée ici) pour Python développée par Google sera utilisée. La structure du réseau utilisé possède deux neurones d'entrée un pour la fraction de Li_2S et l'autre pour la fraction de P_2S_5 (déduite de la première), et deux neurones en sortie pour la valeur du module de Young et du coefficient de Poisson, ainsi que plusieurs couches cachées ayant un nombre décroissant de neurones.

B. Données

Les données utilisées pour entraîner le réseau de neurones proviennent de deux études de Sakuda, Atsushi, et al. [5] et Kato, Atsutaka, et al. [6]). Le tableau des données tirées de ces publications est présent Figure II. Ces données ont été normalisées en des valeurs entre 0 et 1 avec les relations ci-dessous :

$$E_{norm} = \frac{E - E_{min}}{E_{max} - E_{min}}$$

$$\nu_{norm} = \frac{\nu - \nu_{min}}{\nu_{max} - \nu_{min}}$$

Li_2S	P_2S_5	$E(MPa)$	ν	E_{norm}	ν_{norm}
0,25	0,75	13	0.301	0	0,68
0,5	0,5	17,5	0,267	0,375	0
0,67	0,33	22,1	0,314	0,758	0,94
0,7	0,3	21,9	0,315	0,742	0,96
0,75	0,25	22,9	0,317	0,825	1
0,8	0,2	25	0,31	1	0,86

TABLE II: Tableau de données expérimentales utilisées pour entraîner le réseau de neurones

Pour essayer d'améliorer la précision des résultats donnés par le réseau de neurones, le jeu de données étant en l'état plutôt réduite, en raison du sujet d'étude, le nombre d'expériences sur les verres de formulation $Li_2S - P_2S_5$ étant très restreintes, une piste étudiée était d'augmenter artificiellement le nombre de données d'entraînement en répétant chaque ligne du tableau mille fois lors de chaque étape de l'entraînement (époque), ces répétitions étant faites avec et sans bruit. Les entraînements réalisés avec l'ajout de bruit montraient des meilleurs résultats. L'entraînement final du réseau de neurones a donc été réalisé en bruitant les données répétées.

C. Présentation des courbes données par le réseau de neurones

Une fois que le réseau de neurones a été entraîné, il est utilisé afin d'obtenir le module de Young et la coefficient de Poisson pour différentes composition de verre en faisant varier le pourcentage de Li_2S en entrée. Après avoir demandé au réseau de neurones une prédiction pour des entrées entre 0 et 100 pourcent, le coefficient de Poisson et le module de Young

normalisés sont obtenus en fonction de la composition imaginée du verre. Les courbes obtenues sont présentes Figures 9 et 10.

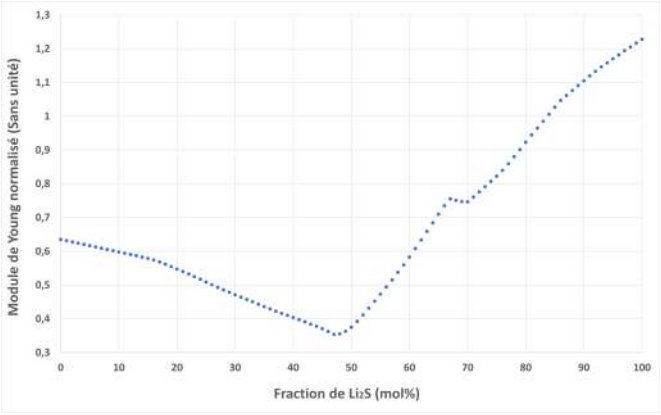

Fig. 9: Module de Young normalisé en fonction de la fraction de Li_2S

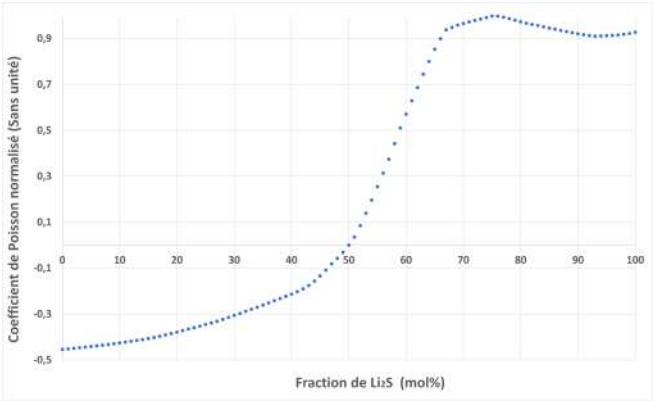

Fig. 10: Coefficient de Poisson normalisé en fonction de la fraction de Li_2S

On dé-normalise ensuite ces prédictions obtenues afin d'obtenir quelque chose d'interprétable pour nous les humains, suivant les formules :

$$E = E_{norm}(E_{max} - E_{min}) + E_{min}$$

$$\nu = \nu_{norm}(\nu_{max} - \nu_{min}) + \nu_{min}$$

Ce qui donne alors les courbes suivantes Figure 11 et 12.

Ainsi pour obtenir le verre avec le plus grand coefficient de Poisson, il faudrait réaliser un verre composé à pourcent de Li_2S.

Il faut toutefois garder à l'esprit que les résultats renvoyés par le réseau de neurones ont été donnés pour toutes les fractions de Li_2S, or toutes les compositions présentes sur l'axe des abscisses ne vont pas forcément pouvoir donner un verre. Il faut donc prendre garde lors de l'analyse des résultats à s'assurer que pour la formulation regardée, il y est une possibilité de fabriquer un verre.

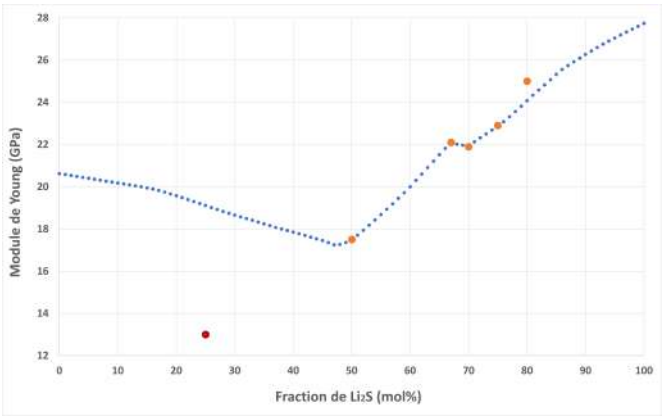

Fig. 11: Module de Young en fonction de la fraction de Li_2S. En bleu, les prédictions du réseau de neurones, en orange les données expérimentales d'entraînement, en rouge la donnée test

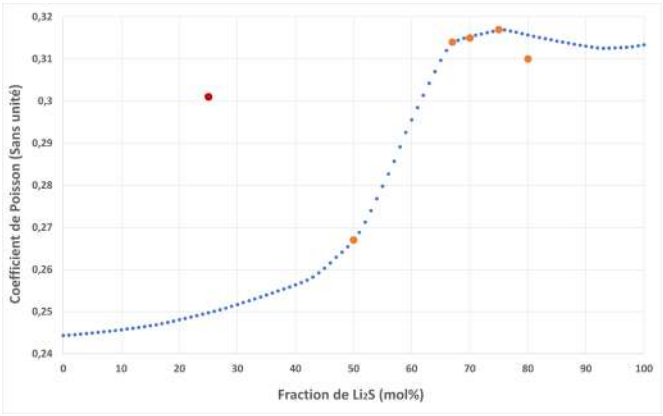

Fig. 12: Coefficient de Poisson en fonction de la fraction de Li_2S. En bleu, les prédictions du réseau de neurones, en orange les données expérimentales d'entraînement, en rouge la donnée test

D. Critique des résultats

Comme dit précédemment les résultats obtenus sont présentés pour toutes les formulations de Li_2S et de P_2S_5 imaginables, or tous les mélanges ne vont pas donner un verre. De plus, les propriétés mécaniques sont importantes certes.
Cependant, ce matériau est surtout étudié pour développer des batteries de nouvelles générations donc les propriétés électriques développées doivent aussi être étudiées en parallèle pour être sûr qu'avec un certain module d'Young et un certain coefficient de Poisson le matériau ne soit pas très mauvais en tant que batterie.

Les données utilisées pour entraîner le réseau de neurones sont issues des résultats de deux publications. Les compostions présentes dans ces publications vont de 25% de Li_2S et 75% de P_2S_5 à 80% de Li_2S et 20% de P_2S_5. Les données expérimentales, qui ont toutes étés utilisées à l'exception d'une pour entraîner le réseau de neurones (voir Tableau II) sont présentes sur les Figure 11 et 12 en rouge et oranges.

En dehors de l'intervalle des données d'entrées, il n'est pas certain que les résultats fournis par le réseau de neurones représentent la réalité. En effet, en reprenant le jeu de données réelles on peut déjà regarder si les courbes obtenues passent bien par les points connus. Il est remarquable que pour les compositions de Li_2S entre 50% et 75% la courbe obtenue par le réseau pour le module de Young est très satisfaisante, les valeurs expérimentales coïncident avec la courbe. Cependant, concernant les deux derniers points Li_2S 25% (ce point n'était pas dans notre entraînement) et 80%, les valeurs mesurées ne coïncident pas. En effet, la courbe entre 25% et 50% devrait possiblement croître or ici elle décroît. Ainsi, $E_{mesuré} = 18.5$ MPa alors que $E_{réel} = 13$ MPa ! L'écart relatif est donc de 42%! Cela n'est pas du tout contrariant puisque nous avons entraîné notre réseau de neurones uniquement entre 50% et 80%. Les valeurs hors de ces frontières sont donc totalement inconnues, la loi est ainsi possiblement totalement fausse sur ce domaine et il n'a pas pu concevoir un tel changement d'inflexion. Concernant, le point à 80% ce résultat est plus surprenant car il faisait partie de notre entraînement cependant l'écart relatif demeure beaucoup plus faible $3,7\%$ ($E_{mesuré} = 24,08$ MPa et $E_{réel} = 25$ MPa), cet écart est donc acceptable. On peut également tirer la même conclusion pour le coefficient de Poisson. Expérimentalement à 25%, la valeur prédite est plus faible que celle expérimentale. Ainsi, on comprend l'importance des données fournies au réseau et on peut également comprendre que hors du domaine que les lois sont fausses. Mais l'importance est de pouvoir, se représenter les courbes et les lois sur le domaine où le verre a été étudié.

IV. Conclusion

L'utilisation d'un réseau de neurones a été un très bon moyen pour prédire la résistance mécanique d'un verre, en effet, il a été très simple et très rapide a mettre en place. Après avoir entraîné le réseau de neurones, il a été possible de d'obtenir la courbe du module de Young et du coefficient de Poisson en fonction de la composition du verre.

Toutefois les résultats obtenus ne garantissent pas qu'un verre puisse être réalisé à la composition souhaitée, et les résultats en dehors de la plage des données d'entrées (50% de Li_2S à 75% de Li_2S) sont assez incertains. Cependant, dans la plage de données, il est assez légitime de croire que la loi représentée est assez fidèle à la loi réelle. Il faut surtout comprendre que l'idéal évidemment n'est pas l'utilisation d'un réseau de neurones. Il faudrait dans l'idéal avoir un nombre très élevé d'échantillons sur une plage la plus large possible de formulations pour pouvoir avoir une courbe dont on soit sûr de la fiabilité.

Cependant, la réalisation d'un verre est très complexe et très onéreuse. De plus, le type de verres que l'on a étudié possède de nombreuses variantes qui sont peut-être aussi intéressantes pour de nouvelles batteries cependant il faut également pouvoir étudier leurs propriétés mécaniques pour comparer ces alternatives. Il y a donc un travail colossal à effectuer et c'est pour cela qu'en général il y a "si peu" d'échantillons à une composition donnée. D'une part pour la complexité de réalisation, d'autre part pour la multitude de verres à étudier.

C'est ainsi dans cette optique que l'on peut comprendre la réelle utilité du réseau de neurones. Puisque ces réseaux sont capable de prédire de manière très globale et très rapide la loi mécanique d'un verre donné et de relever très rapidement la composition du verre idéal d'un point de vue mécanique pour demander une nouvelle étude de ce verre à composition spécifique à l'équipe en charge d'évaluer le potentiel de cette batterie particulière. Cela permet ainsi un gain de temps très grand. Les réseaux de neurones ont donc un potentiel très grand à l'avenir pour faciliter l'apprivoisement des données.

Remerciement

Nous remercions notre encadrant Éric Robin de l'Institut de Physique de Rennes pour encadrement, la formation et la découverte réalisée lors de ce mini-projet Outils numériques PROJ1b.

References

[1] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5 (1943): 115-133.

[2] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

[3] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." nature 323.6088 (1986): 533-536.

[4] Rumelhart, David E. "Learning internal representations by error propagation, in parallel distributed processing." Explorations in the Microstructure of Cognition (1986): 318-362.

[5] Sakuda, Atsushi, et al. "Evaluation of elastic modulus of Li2S–P2S5 glassy solid electrolyte by ultrasonic sound velocity measurement and compression test." Journal of the Ceramic Society of Japan 121.1419 (2013): 946-949.

[6] Kato, Atsutaka, et al. "Mechanical properties of sulfide glasses in all-solid-state batteries." Journal of the Ceramic Society of Japan 126.9 (2018): 719-727.