Réseau de neurones en actuariat

Projet de recherche

Nicolas Bellemare

Baccalauréat en actuariat

Québec, Canada

Table des matières

Liste des tableaux

Liste des figures

Introduction

Les réseaux de neurones font parties de la branche de l'apprentissage supervisé qu'on nomme l'apprentissage profond. On dit « profond » puisqu'ils sont composées de plusieurs couches qui permettent de soutirer de l'information contenue dans des données. Avec la profondeur vient la complexité. Chaque neurone d'une couche sera lié avec chacun des neurones de la couche précédente et avec chacun des neurones de la couche suivante. Ainsi, rapidement les réseaux de neurones deviennent complexes et difficiles à interpréter. C'est dans ce contexte qu'on se questionne à savoir s'il est possible de les appliquer à un contexte actuariel classique, soit la modélisation de la fréquence de réclamation en assurance automobile à partir de données structurées.

En effet, l'assurance en général requiert que les modèles de prévisions soient interprétables pour répondre aux conditions des régulateurs de marché. Les modèles classiques que sont les modèles linéaires généralisés permettent une interprétation claire. Par ce présent travail, on tente d'appliquer des méthodes d'interprétation de modèles de type « boîte noire » pour essayer de comprendre si les réseaux de neurones peuvent répondre aux conditions de régulations.

Tout d'abord, le chapitre 1 se consacre à la théorie des réseaux de neurones. On élabore brièvement leur fonctionnement. Pour une lecture en profondeur sur le sujet, on cite la référence en apprentissage profond (Goodfellow et collab., 2016). Pour une revue des réseaux de neurones avec l'emphase sur des problèmes actuariels, on cite Denuit et Trufin (2019). Plusieurs articles scientifiques sont consacrés à l'étude de problèmes actuariels à l'aide réseau de neurones. Dans Wuthrich (2019), on présente comment utilisé les réseaux de neurones pour améliorer les modèles actuariels classiques, article dont on se sert pour construire certains modèles. Plus particulièrement, on utilise les articles Ferrario et collab. (2018) et (Schelldorfer et Wuthrich, 2019), qui font partie d'une série de tutoriels pour l'Association Suisse des Actuaires. On y explique comment utiliser les réseaux de neurones pour modéliser la fréquence de réclamation en assurance automobile et on y explique d'une façon plus appliquée l'approche CANN détaillée dans Wuthrich (2019).

Le second chapitre est consacré à l'application de réseaux de neurones. On y détaille comment on construit les réseaux et on y compare les différents réseaux obtenus.

Chapitre 1

Théorie des réseaux de neurones

1.1 Apprentissage supervisé

La présente section est basée sur le chapitre 2 de James et collab. (2013) et le chapitre 2 de Hastie et collab. (2009).

Les réseaux de neurones sont des modèles qui peuvent s'appliquer tant à l'apprentissage supervisé qu'à l'apprentissage non-supervisé. On résume dans cette section les bases de l'apprentissage supervisé pour la suite.

Soit $X = (X_1, \ldots, X_p)$, un vecteur de p variables aléatoires et Y, une variable que l'on veut prédire. On suppose qu'il existe une relation entre Y et X telle que,

$$Y = f(X) + \epsilon,$$

où f est une fonction quelconque et ϵ est un terme d'erreur qui est indépendant de X. L'apprentissage supervisé vise à estimer f pour pouvoir prédire ou expliquer Y à partir de X. On veut utiliser les données pour estimer une fonction f qui soit utile, c'est-à-dire, une fonction qui permet de bien estimer Y à partir de nouvelles données.

Un algorithme d'apprentissage supervisé est capable de modifier ses paramètres internes en réponse à une fonction de perte. En d'autres termes, il apprend à partir des erreurs commises. La fonction de perte mesure la différence entre la prédiction, $\hat{f}(x)$, et la vraie valeur de Y. On dénote cette fonction par $\mathcal{L}$. Par exemple, l'erreur quadratique moyenne est souvent utilisée pour un problème de régression :

$$\mathcal{L}\left\{Y, \hat{f}(X)\right\} = \left\{\hat{f}(X) - Y\right\}^2.$$

1.1.1 Compromis biais-variance

Un des enjeux principaux en apprentissage statistique est le compromis biais-variance. On veut un modèle qui soit le plus précis possible tout en étant le moins variable possible. Par

contre, lorsqu'on diminue le biais, la variance de l'estimateur finit toujours par augmenter : on doit trouver le compromis. On illustre cet enjeu par l'erreur quadratique espérée :

$$\mathrm{E}\left[\left\{Y-\hat{f}\left(X\right)\right\}^{2}\right]=\mathrm{Biais}^{2}\left\{\hat{f}\left(X\right)\right\}+\mathrm{var}\left\{\hat{f}\left(X\right)\right\}+\sigma^{2},$$

où $\sigma^2 = \mathrm{var}(\epsilon)$, l'erreur irréductible.

On ne peut pas diminuer l'erreur irréductible, comme son nom l'indique, puisqu'elle fait partie de tout processus aléatoire. Toutefois, on peut diminuer l'erreur réductible (biais + variance) en diminuant ou en augmentant la complexité du modèle. La complexité du modèle se traduit par le nombre de paramètres qu'il faut estimer. Plus on a de paramètres, plus le modèle est en mesure de faire des prédictions exactes sur les données d'entrainement. En revanche, la variance sera plus élevée, car le modèle aura appris les caractéristiques des observations qui sont propres à celles-ci.

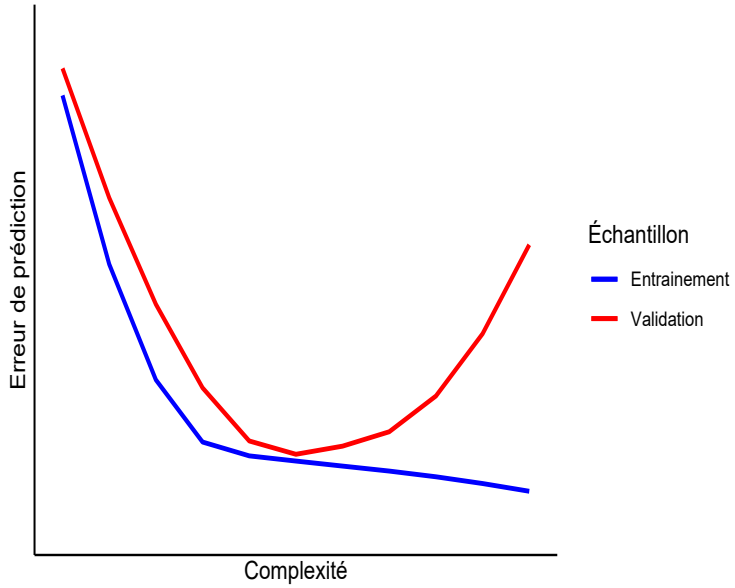
Pour être en mesure de surveiller le compromis, on sépare les données en trois échantillons distincts : l'échantillon d'entrainement, l'échantillon de validation et l'échantillon de test. L'échantillon d'entrainement $\mathcal{D}$ est celui sur lequel on entraine le modèle ; il sert à ajuster les paramètres. L'échantillon de test $\mathcal{T}$ permet de mesurer la performance finale du modèle et ainsi le comparer avec d'autres. $\mathcal{T}$ n'est jamais vu durant l'entrainement du modèle. Il sert à simuler l'utilisation du modèle une fois qu'il est calibré sur de vrais données. L'échantillon de validation $\mathcal{V}$ permet de faire la sélection d'hyperparamètres et c'est à partir de celui-ci qu'on surveille si le modèle ne surajuste pas les données d'entrainement.

La Figure 1.1 montre un exemple de l'erreur de prédiction en fonction de la complexité d'un modèle. On voit que plus la complexité augmente, plus l'erreur de prédiction sur les données d'entrainement diminue. On remarque aussi que l'erreur de prédiction sur les données de validation diminue lorsqu'on augmente la complexité pour ensuite augmenter.Lorsque le modèle atteint le point où l'erreur de prédiction sur l'échantillon de validation augmente, on dit que le modèle surajuste l'échantillon d'entrainement. Il ne généralise pas bien pour des données qu'il n'a pas vu. L'objectif devient donc de trouver une combinaison des paramètres à estimer et de la complexité du modèle pour minimiser la fonction de perte sur l'échantillon de validation.

1.2 Architecture d'un réseau de neurones

Malgré qu'il existe plusieurs types de réseau de neurones, la présente section ainsi que l'analyse de données subséquente sont basées sur la forme la plus simple de réseau de neurones, soit le perceptron ou réseau de neurones à propagation directe (« feedforward neural network »). Il existe deux formes de perceptron : le perceptron simple et le perceptron multicouche. La différence entre les deux formes réside dans le nombre de couches cachées. Le

FIGURE 1.1 – Erreur de prédiction en fonction de la complexité du modèle



perceptron simple a une seule couche cachée, tandis que le perceptron multicouche en a plusieurs. Pour bien comprendre le perceptron, on compare son fonctionnement avec des modèles plus simples. On illustre les modèles de régression linéaire simple et multiple et le modèle de régression logistique pour expliquer la progression qui nous amènera au réseau de neurones.

Un modèle de régression linéaire simple est une somme de la variable exogène. On suppose que la relation entre Y et X est de la forme suivante :

$$Y = b + \omega X + \epsilon,$$

où b est le biais et ω est le poids associé à la variable X.

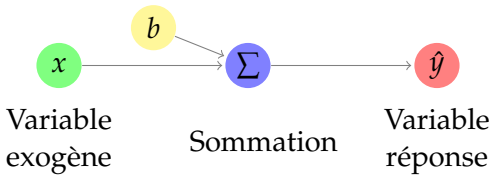
On estime Y par :

$$\hat{Y} = \hat{f}(X) = \hat{b} + \hat{\omega} X.$$

Ce modèle utilise l'information qui est contenue dans la variable exogène pour prédire la variable réponse. Pour ce faire, on calcule $\hat{b}$ et $\hat{\omega}$ de façon à minimiser l'erreur quadratique de prévision. On illustre ce modèle à l'aide de la Figure 1.2. La partie verte de la figure illustre l'information qui entre dans le modèle, la partie bleue est le traitement de l'information et la partie rouge est la prévision du modèle. On voit que la partie bleue est en fait la fonction $\hat{f}$. Le terme de la partie jaune est le biais.

Dans le cas de la régression linéaire multiple, on a p variables exogènes. On fait une sommation pondérée de toutes ces variables pour pouvoir prédire la variable réponse. La relation

FIGURE 1.2 – Modèle de régression linéaire simple



entre Y et X est :

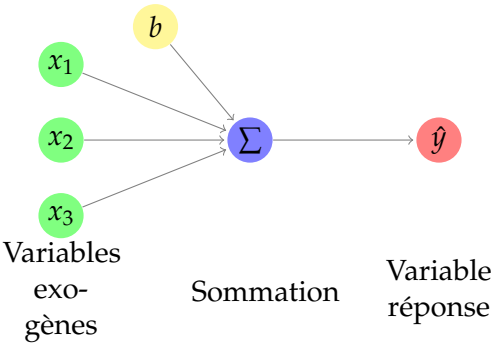
$$Y = b + \sum_{i=1}^{p} \omega_i X_i + \epsilon.$$

On estime Y par :

$$\hat{Y} = \hat{f}(X) = \hat{b} + \sum_{i=1}^{p} \hat{\omega}_i X_i.$$

La Figure 1.3 illustre un cas avec 3 variables exogènes. On y voit le même mécanisme qu'avec la régression linéaire simple, c'est-à-dire, on combine l'information, on la traite et puis on prédit un résultat. La différence est que l'information a plus d'une dimension. On doit donc estimer un paramètre pour chaque variable exogène et un paramètre pour le biais. Il s'en dégage une structure qui est bien présente dans les modèles d'apprentissage statistique.

FIGURE 1.3 – Modèle de régression multiple

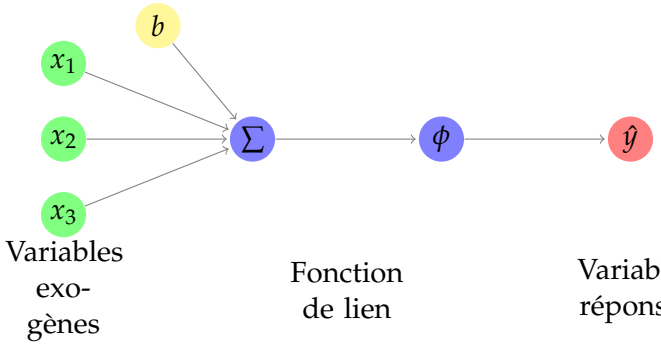


La régression logistique, quant à elle, ajoute une transformation non-linéaire à la combinaison linéaire des variables exogènes. Ainsi, la variable prédite peut être contenue dans l'intervalle $[0, 1]$. La Figure 1.4 ajoute donc ϕ, qu'on appelle fonction d'activation, au traitement de l'information. Le modèle est

$$Y = \frac{e^{b+\sum_{i=1}^{p} \omega_i X_i}}{1 + e^{b+\sum_{i=1}^{p} \omega_i X_i}}.$$

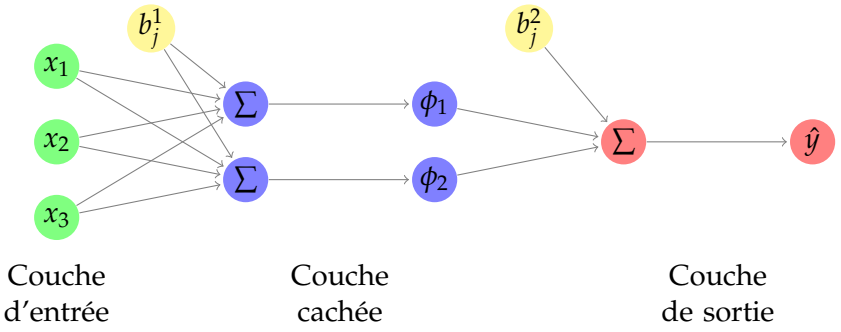
La relation entre Y et X est ainsi non-linéaire. On peut voir la régression logistique comme étant un réseau de neurones avec une seule couche cachée et un seul neurone dans la couche

FIGURE 1.4 – Régression logistique



cachée. On comprend alors ce qu'un neurone artificiel est. Un neurone est une unité qui applique la fonction non-linéaire ϕ à la combinaison linéaire de chaque unité de la couche précédente.

FIGURE 1.5 – Réseau de neurones avec une couche cachée [1]

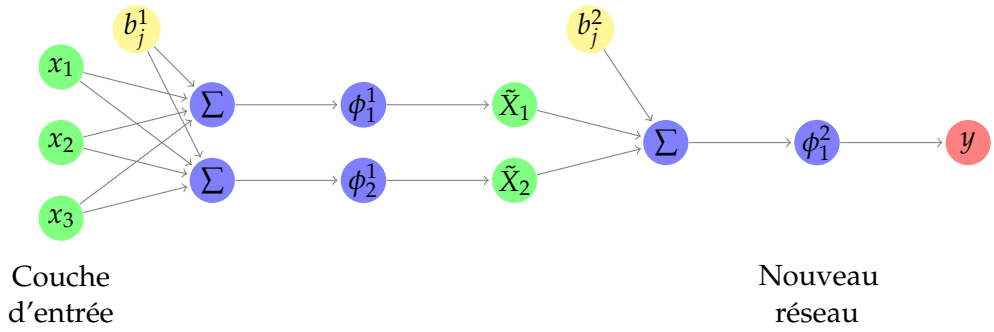


Les trois exemples précédents permettent de voir que les réseaux de neurones ne sont en fait qu'une généralisation de la régression linéaire. En effet, comme le montre la Figure 1.5, la partie de traitement de l'information (en bleu) est constituée de plusieurs neurones (deux neurones dans cet exemple). À chaque neurone, on combine linéairement les variables exogènes et on applique la fonction d'activation ϕ. Ensuite, la combinaison linéaire de la réponse de chacun de ces neurones est utilisée pour prédire la variable réponse du réseau. Les résultats des fonctions ϕ deviennent à leur tour de nouvelles variables entrant dans un réseau. Illustrons ceci par un exemple.

La Figure 1.6 montre que les fonctions non-linéaires ϕ_j créent une nouvelle représentation de l'information. Cette représentation est alors combinée linéairement et traitée par un autre réseau. On voit que l'on peut reproduire cette séquence un nombre infini de fois. Par mesure de simplicité et pour ne pas saturer les graphiques, on résume chaque neurone par $a_j^1 = \phi_j\left(b_j^1 + \sum_{i=1}^{p} \omega_{j,i}^1 X_i\right)$, pour le j^{e} neurone de la première couche cachée, et par $a_j^l =$

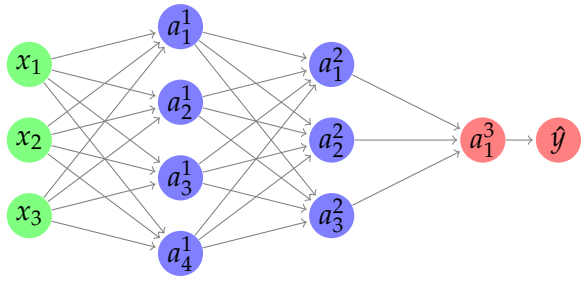
1. Ce graphique est une adaptation de la réponse de l'utilisateur *gvgramazio* sur https://tex.stackexchange.com/questions/153957/drawing-neural-network-with-tikz

FIGURE 1.6 – Décomposition d'un réseau en une suite de deux réseaux



$\phi\left(b_j^l + \sum_{k=1}^{q_{k-1}} \omega_{j,k}^l a_k^{l-1}\right)$, pour le j^{e} neurone de la l^{e} couche cachée. On a q_k neurones pour la k^{e} couche cachée. La Figure 1.7 illustre un réseau de neurones en considérant cette notation avec deux couches cachées de quatre et trois neurones, respectivement. On remarque un neurone supplémentaire directement lié à la prévision, soit a_1^3 dans notre exemple, ou a_1^{k+1} pour un réseau avec k couches cachées et une seule prédiction. Ce dernier neurone permet de faire le lien entre le réseau et la prévision. C'est la fonction de régression.

FIGURE 1.7 – Réseau de neurones avec deux couches cachées



On obtient la formule suivante pour un réseau de neurones avec une couche cachée de q_1 neurones et avec la fonction identité pour la fonction de régression, $a_1^3(x)$:

$$Y = b^2 + \sum_{j=1}^{q_1} \omega_{1,j}^2 \, \phi_j \left(b_j^1 + \sum_{i=1}^{p} \omega_{j,i}^1 \, X_i \right)$$

1.3 Phase de propagation directe (« feedforward »)

On résume la notation :

- $\omega_{j,k}^l$, le poids du lien entre le k^{e} neurone de la $(l-1)^{\text{e}}$ couche au j^{e} neurone de la l^{e} couche

- b_j^l, le biais associé au j^{e} neurone de la l^{e} couche
- ϕ, une fonction d'activation quelconque
- a_j^l, l'activation du j^{e} neurone de la l^{e} couche
- z_j^l, l'intrant du j^{e} neurone de la l^{e} couche

On utilise une notation matricielle pour la suite. Ainsi, on a la matrice de poids suivante pour relier la couche $l-1$ à la couche l

$$\omega^l = \begin{bmatrix} \omega_{1,1}^l & \omega_{1,2}^l & \cdots & \omega_{1,q_{l-1}}^l \\ \omega_{2,1}^l & \omega_{2,2}^l & \cdots & \omega_{2,q_{l-1}}^l \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{q_l,1}^l & \omega_{q_l,2}^l & \cdots & \omega_{q_l,q_{l-1}}^l \end{bmatrix}.$$

On a que

$$z_j^l = \sum_k \omega_{j,k}^1 a_j^{l-1} + b_j^l$$

et

$$z^l = \omega^l a^{l-1} + b^l.$$

Alors, la matrice des intrants de la l^{e} couche avec q_l neurones est

$$z^l = \begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_{q_l}^l \end{bmatrix}.$$

Ainsi, on peut réécrire

$$a_j^l = \phi\left(\sum_k \omega_{j,k}^1 a_j^{l-1} + b_j^l\right)$$
$$a^l = \phi\left(\omega^l a^{l-1} + b^l\right).$$

Ce qui donne pour la matrice des activations de la $l$ème couche de q_l neurones

$$a^l = \begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_{q_l}^l \end{bmatrix}.$$

1.4 Fonction d'activation

La fonction d'activation permet de créer une représentation non-linéaire de l'information. Elle doit absolument être non-linéaire, sinon le réseau serait une combinaison linéaire de combinaisons linéaires. Elle permet ainsi d'apprendre des interactions complexes entre les variables exogènes. Les fonctions d'activation les plus communes sont

$$\phi(x) = \begin{cases} \frac{e^x}{1+e^x} & \text{, sigmoïde} \\ \tanh(x) & \text{, tangente hyperbolique} \\ \mathbf{1}_{\{x \geq 0\}} & \text{, escalier} \\ x\mathbf{1}_{\{x \geq 0\}} & \text{, « Rectified Linear unit », Relu} \end{cases}$$

1.5 Rétro-propagation (« backpropagation »)

Cette section est inspirée de la série sur les réseaux de neurones de la chaîne Youtube « 3blue1brown » de Sanderson et du chapitre sur le « backpropagation » de Nielsen (2015).

On aborde maintenant le mécanisme d'apprentissage du réseau de neurone à propagation directe. On dit propagation directe (« feedforward ») puisque l'information ne fait que se propager vers l'avant. Il n'y a pas de cycle dans le modèle où l'information repasse dans une partie du réseau plusieurs fois via une boucle. Un réseau dont l'information circule de cette façon est appelé un réseau de neurones récurent.

Toutefois, le réseau à propagation directe utilise la méthode de rétro-propagation (« back-propagation ») pour diffuser le signal qui lui permet d'ajuster ses paramètres. On dit rétro-propagation puisque cette méthode diffuse le signal en faisant le chemin inverse de la phase de propagation directe.

On veut que le réseau apprenne la bonne combinaison de poids et de biais pour la tâche à effectuer. Dans notre cas, la tâche est une régression. On utilise une fonction de perte pour mesurer la performance du modèle à cette tâche. C'est à partir de cette fonction que le réseau va apprendre.

La méthode de rétro-propagation permet de trouver une combinaison de poids et de biais qui puisse minimiser la fonction de perte. L'astuce est de calculer le gradient de cette fonction en appliquant successivement la règle de dérivation en chaîne.

La première étape consiste à calculer la sortie du réseau, $\hat{y}_i$, pour chaque observation i. Pour ce faire, on doit initialiser les paramètres de façon aléatoire. On calcule ensuite le résultat de la fonction de perte. Pour la suite du document, on utilise la déviance de Poisson :

$$\mathcal{L}(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} 2y_i \left(\frac{\hat{y}_i}{y_i} - 1 - \log\left(\frac{\hat{y}_i}{y_i} \right) \right).$$

Cette fonction varie seulement selon la valeur des paramètres du réseau. En effet, elle prend comme argument y_i et $\hat{y}_i$, où y_i est fixe. On rappelle aussi que $\hat{y}_i = \hat{f}(\mathbf{X}_i)$. Ainsi, pour faire varier $\mathcal{L}$, il faut faire varier $\hat{f}$. Considérant que $\mathbf{X}_i$ est fixe, la seule façon de faire varier $\hat{f}$ est de faire varier ses paramètres internes $w_{j,k}^{l}$ et b_j^l.

Le paragraphe précédent décrit l'intuition derrière la méthode de rétro-propagation. On détermine comment la fonction de perte varie par rapport aux paramètres indirectement. $\mathcal{L}$ est fonction des paramètres du réseau. Le gradient de cette fonction, $\nabla\mathcal{L}$, détermine la direction dans laquelle un changement aux paramètres permet d'augmenter le plus rapidement la valeur de $\mathcal{L}$. On veut donc déterminer chaque élément de $\nabla\mathcal{L}$,

$$\nabla\mathcal{L} = \begin{bmatrix} \dfrac{\partial\mathcal{L}}{\partial\omega^{(1)}} \\ \dfrac{\partial\mathcal{L}}{\partial b^{(1)}} \\ \vdots \\ \dfrac{\partial\mathcal{L}}{\partial\omega^{(L)}} \\ \dfrac{\partial\mathcal{L}}{\partial b^{(L)}} \end{bmatrix}.$$

On veut trouver les dérivées partielles

$$\frac{\partial\mathcal{L}}{\partial\omega_{j,k}^{(l)}} \quad \text{et} \quad \frac{\partial\mathcal{L}}{\partial b_j^{(l)}}, \quad \text{pour chaque} \quad j, k, l.$$

Celles-ci nous informent sur la sensibilité de $\mathcal{L}$ par rapport à un petit changement de $\omega_{j,k}^{(l)}$ et de $b_j^{(l)}$, respectivement. Ainsi, à l'aide de chaque élément de $\nabla\mathcal{L}$, on peut ajuster les paramètres dans la direction qui permet de diminuer le plus rapidement $\mathcal{L}$. On applique

$$\omega_{j,k}^{(l)} \mapsto \omega_{j,k}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \omega_{j,k}^{(l)}}$$

et

$$b_j^{(l)} \mapsto b_j^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial b_j^{(l)}},$$

où η est le taux d'apprentissage. Celui-ci permet d'ajuster les paramètres proportionnellement à leur dérivée partielle. Autrement dit, on ajuste chacun des paramètres par un pas proportionnel à leur importance relative d'une variation de $\mathcal{L}$ dans la direction qui mène le plus rapidement au minimum local. Par exemple, si on a $\frac{\partial \mathcal{L}}{\partial \omega_{1,1}^{(1)}} = 2$ et $\frac{\partial \mathcal{L}}{\partial \omega_{2,2}^{(2)}} = -0.1$, on voit que chacun de ces paramètres influence la fonction de perte dans une direction opposée. De plus, on voit que $\omega_{1,1}^1$ a beaucoup plus d'importance quant à la variation de $\mathcal{L}$. En ajustant les paramètres d'un pas proportionnel à ces dérivées, les paramètres qui influencent le plus la variation de $\mathcal{L}$ sont ceux dont l'ajustement sera le plus grand.

Une méthode qui permet de calculer chaque $\frac{\partial \mathcal{L}}{\partial \omega_{j,k}^{(l)}}$ est :

$$\frac{\partial \mathcal{L}}{\partial \omega_{j,k}^{l}} \approx \frac{\mathcal{L}\left(\omega + \epsilon e_{j,k}^{l}\right) - \mathcal{L}(\omega)}{\epsilon},$$

où $\epsilon > 0$ est petit, et $e_{j,k}^{l}$ est un vecteur unitaire dans la direction,j, k, l. Cependant, cette méthode requiert de calculer $\mathcal{L}\left(\omega + \epsilon e_{j,k}^{l}\right)$ pour chaque paramètre. Ainsi, si on a q paramètres à estimer dans le réseau, on doit faire $q + 1$ phases de propagation directe pour ajuster les paramètres qu'une seule fois. De plus, on doit ajuster plusieurs fois les paramètres avant d'atteindre un minimum local. Évidemment, le temps de calcul est très élevé pour cette méthode.

La méthode de rétro-propagation permet de calculer efficacement ces dérivées en passant dans le réseau seulement deux fois pour ajuster tous les paramètres une fois. Elle s'appuie sur la règle de dérivation en chaîne. Soit $a_i^L = \phi(z_i^L)$, le vecteur des activations de la couche de sortie d'un réseau avec $L-1$ couches cachées pour la i^{e} observation, et $z_i^L = (\omega_L a_i^{L-1} + b^L)$, l'intrant de la fonction d'activation a^L, alors on a que

$$\frac{\partial \mathcal{L}}{\partial \omega_{j,k}^{L}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial z_i^L}{\partial \omega_{j,k}^{L}} \frac{\partial a_i^L}{\partial z_i^L} \frac{\partial \mathcal{L}_i}{\partial a_i^L}.$$

1.6 Hyperparamètres

On présente brièvement les hyperparamètres possibles des réseaux de neurones.

- Taux d'abandon (d) : Permet d'abandonner aléatoirement un certain pourcentage des neurones d'une couche en particulier à chaque phase de rétro-propagation.
- Taux de régularisation (ℓ_1, ℓ_2) : pénalité de type Ridge et Lasso appliquée à une couche en particulier
- Algorithme d'apprentissage : chaque algorithme d'apprentissage repose sur la descente du gradient, mais chacun essait d'améliorer la convergence de différentes façons. L'algorithme qu'on utilisera est le nadam,(Sutskever et collab., 2013).
- L'initialisation des poids : pour éviter que le gradient disparaisse, « vanishing gradient problem », il faut initialiser les poids de façon aléatoire. On utilise l'initialisation Kaiming, (He et collab., 2015)

1.7 Approche CANN

CANN veut dire « Combined Actuarial Neural Network ». Il s'agit d'une méthode pour initialiser le réseau en fonction d'un modèle classique en actuariat. L'idée est de prendre la prévision du modèle de base de chaque observation de la base de donnée et de s'en servir comme mesure d'exposition v_i dans le modèle Poisson. Ainsi, le modèle commence son entrainement avec le maximum de vraisemblance du modèle de base comme première sortie. Il cherche donc une combinaison de paramètres en commençant à un endroit sur la surface de la fonction de perte qui risque d'être mieux qu'une initialisation aléatoire. On testera donc cette approche pour la suite.

Chapitre 2

Application d'un réseau de neurones à propagation directe

2.1 Présentation des données

On utilise la base de données `freMTPLfreq` du paquetage `CASdatasets`, (Dutang et Charpentier, 2019), pour la modélisation de la fréquence de réclamation. Ces données contiennent un portefeuille de 413 169 polices d'assurance responsabilité civile automobile pour une compagnie d'assurance française.

Voici une brève description de chaque variable dans la base de données :

PolicyID Identifiant de l'assuré

ClaimNb Nombre de réclamation

Exposure Période durant laquelle la police d'assurance est effective, en année

Power Variable catégorielle ordonnée de la puissance du véhicule

CarAge Âge du véhicule, en année

DriverAge Âge du conducteur, en année

Brand Constructeur du véhicule

Gas Type de carburant utilisé, régulier ou diésel

Region Région de la France

Density Nombre d'habitants par km^2 dans la ville où l'assuré réside

Il y a 421 observations qui ont une durée d'exposition plus grande qu'un an. Ces données sont corrigées à 1, puisqu'il s'agit probablement d'un erreur.

On voit à la Figure 2.1 la répartition du nombre de réclamation. On a 14 633 polices avec une réclamation, 726 avec deux réclamations, 28 avec trois réclamations et 3 avec quatre réclamations. On voit au graphique de droite de la Figure 2.1 la répartition du temps d'exposition. Une majorité des assurés de ce portefeuille ne sont pas assurés durant toute l'année. On a 29,54% des polices qui sont en vigueur toute l'année.

La Figure 2.2 montre la somme des expositions des assurés selon leur région de résidence et la fréquence moyenne observée par région. Une grande proportion de l'expérience du portefeuille provient de la région « Centre ». Il n'y a pas de grosse différence entre chaque région pour la fréquence moyenne observée.

La Figure 2.3 montre la répartition de l'expérience en fonction de la catégorie du véhicule et leur fréquence moyenne observée. On voit une augmentation linéaire de la fréquence moyenne en fonction de la puissance du véhicule.

On observe à la Figure 2.4 (droite) une légère augmentation de la fréquence moyenne observée pour l'âge du véhicule allant de 0 à environ 10. Par la suite, la fréquence diminue considérablement. À noter que les véhicules de 20 ans et plus ont été groupés. Ces véhicules représentent 8370 observations pour 2,026% de la base de données.

FIGURE 2.1 – Nombre de réclamations observées (gauche), histogramme de l'exposition (droite).

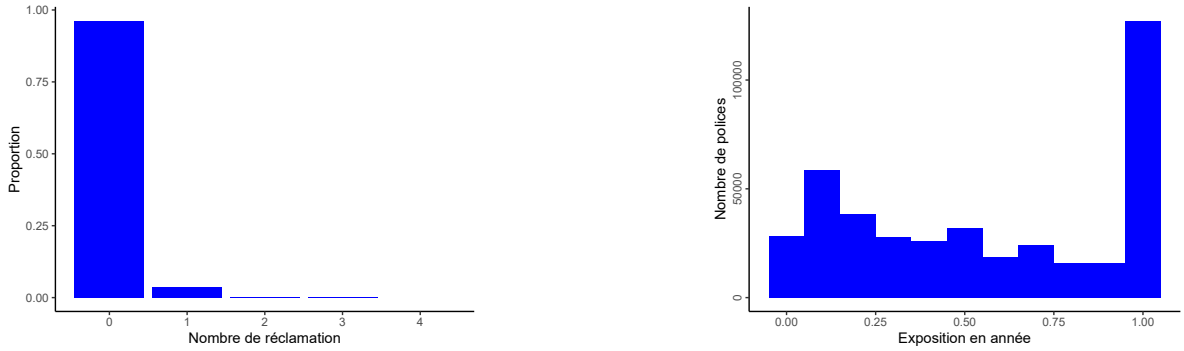


FIGURE 2.2 – Exposition totale par région (gauche), fréquence moyenne observée par région (droite).

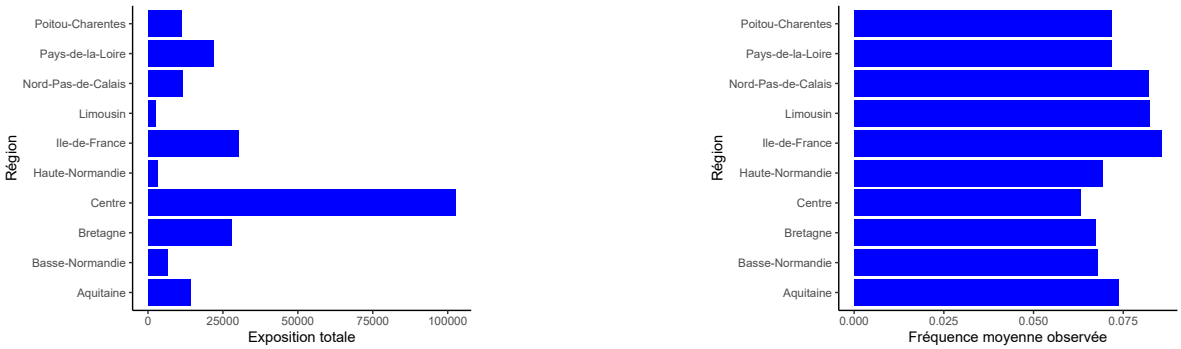


FIGURE 2.3 – Exposition totale par catégorie de véhicules (gauche), fréquence moyenne observée par catégorie de véhicule (droite).

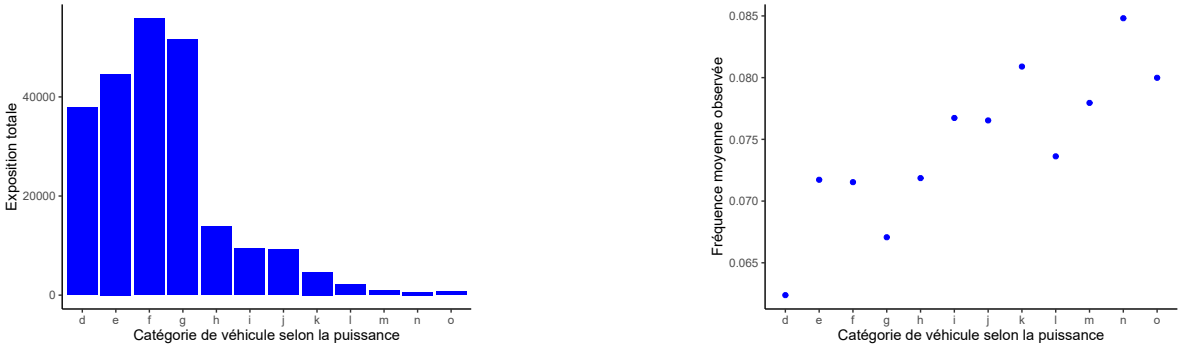


FIGURE 2.4 – Exposition totale par âge du véhicule avec regroupement des véhicules de plus de 20 ans (gauche), fréquence moyenne observée par âge du véhicule (droite).

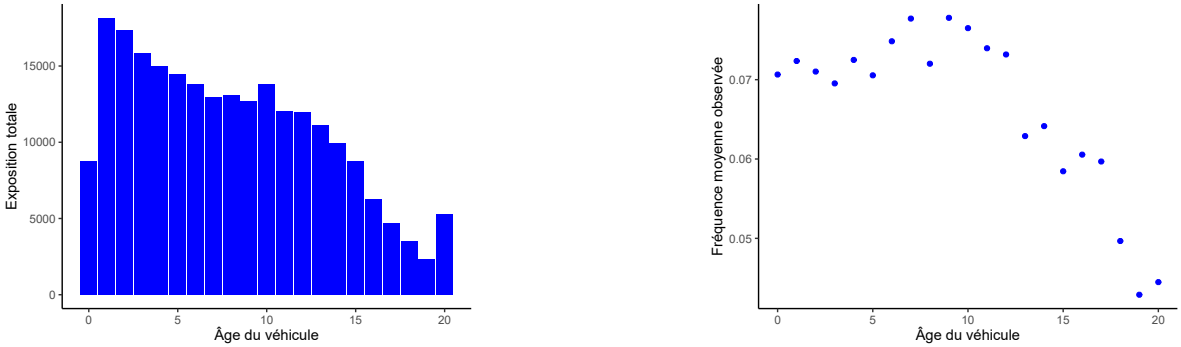


FIGURE 2.5 – Exposition totale par âge de l'assuré (gauche), fréquence moyenne observée par âge de l'assuré (droite).

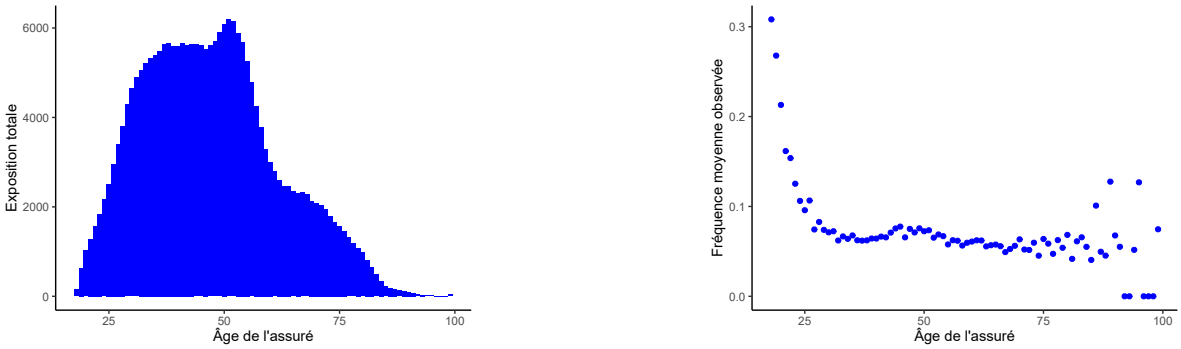


FIGURE 2.6 – Exposition totale par marque de véhicule (gauche), fréquence moyenne observée par marque de véhicule (droite).

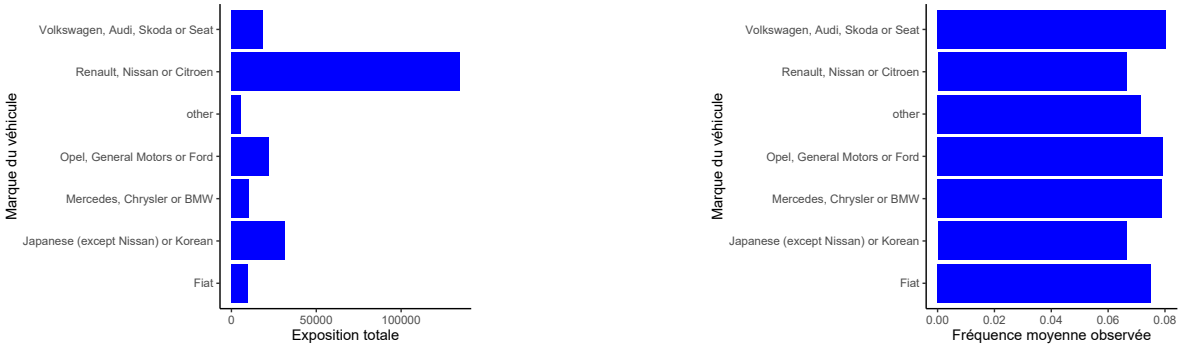
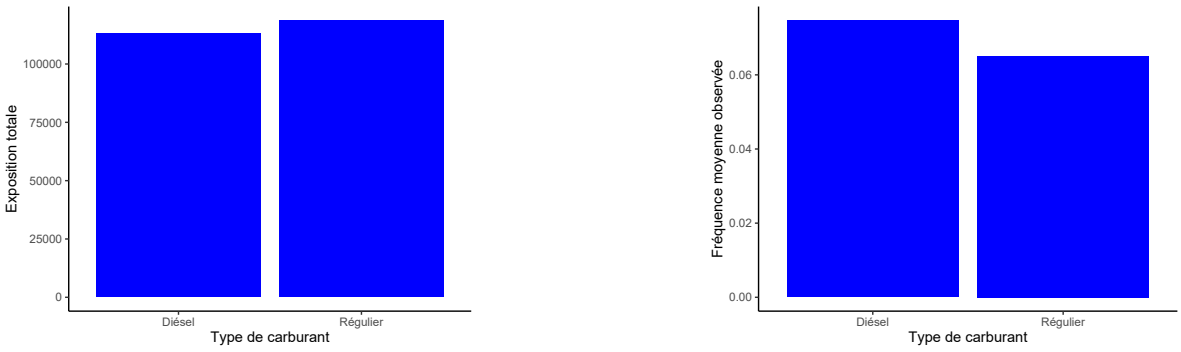


FIGURE 2.7 – Exposition totale par type de carburant (gauche), fréquence moyenne observée par type de carburant (droite).



On remarque à la Figure 2.5 un mode autour de 50 ans. On remarque également que les assurés en bas de 25 ans font plus de réclamations. Il y a une légère augmentation de la fréquence moyenne autour de 45 ans. On voit également une variabilité de la fréquence observée pour les âges élevés. Cela est lié au fait qu'il y a peu d'observations (429) pour ces âges. Ainsi, on les regroupe en une catégorie 85 ans et plus.

2.2 Modèle de fréquence

Pour chaque assuré i, on a un vecteur de sept dimensions qui représente la i^{e} observation :

$$x_i = (\text{Power}_i, \text{CarAge}_i, \text{DriverAge}_i, \text{Brand}_i, \text{Gas}_i, \text{Region}_i, \text{Density}_i)', \quad i = 1, \ldots, 413169.$$

On veut prédire $N_i = \texttt{ClaimNb}_i \geq 0$ en tenant compte de l'exposition au risque, $v_i = \texttt{Exposure}_i$, qui est différente d'un assuré à l'autre. On suppose que chaque observation est indépendante. On a que

$$N_i \sim Poisson\left(\lambda(x_i)v_i\right)$$

On sépare les données en trois échantillons disjoints et en stratifiant sur la variable `ClaimNb`. On utilise 60% des données pour l'échantillon d'entrainement, $\mathcal{D}$, et 20% pour l'échantillon de test, $\mathcal{T}$, et l'échantillon de validation, $\mathcal{V}$. Au Tableau 2.1, on voit que la proportion du nombre de réclamation est donc similaire entre chaque échantillon.

On utilise la déviance de Poisson comme fonction de perte :

$$\mathcal{L}(\mathcal{D}, \lambda) = \frac{1}{n}\sum_{i=1}^{n} 2N_i \left[\frac{\lambda\left(\boldsymbol{x}_i\right)v_i}{N_i} - 1 - \log\left(\frac{\lambda\left(\boldsymbol{x}_i\right)v_i}{N_i}\right)\right].$$

2.3 Modèle de base

On construit un modèle de base pour pouvoir comparer les réseaux avec celui-ci. On considère un modèle linéaire généralisé Poisson avec le lien logarithmique. On suppose la relation suivante :

$$E\left[\frac{N_i}{v_i}\right] = \exp\left\{\mathbf{x_i}\boldsymbol{\omega}\right\}.$$

On ajuste les données $\mathcal{D}$ à ce modèle. Pour le modèle de base, on considère seulement les effets principaux des variables explicatives. On ajuste le premier modèle suivant :

$$\log\left(\frac{\lambda_i}{v_i}\right) = b + \omega_1\texttt{DriverAge}_i + \omega_2\texttt{Power}_i + \omega_3\texttt{CarAge}_i + \omega_4\texttt{Region}_i + \omega_5\texttt{Gas}_i + \omega_6\log\left\{\texttt{Density}_i\right\}.$$

TABLE 2.1 – Répartition du nombre de réclamations par échantillon

	`ClaimNb` (%)				
Échantillon	0	1	2	3	4
Entrainement	96.2751	3.5417	0.1759	0.0065	0.0008
Test	96.2740	3.5421	0.1755	0.0073	0.0012
Validation	96.2763	3.5410	0.1755	0.0073	0.0000

TABLE 2.2 – Seuil observé de la statistique de Wald pour les paramètres de glm1

Variable	seuil observé
(Intercept)	0.0000000
Power	0.0000025
CarAge	0.0000001
DriverAge	0.0000000
Br.1	0.0000000
Br.2	0.6457504
Br.3	0.4977571
Br.4	0.1940670
Br.5	0.1329266
Br.6	0.6915220
GasRegular	0.0000000
Re.1	0.2459378
Re.2	0.0065408
Re.3	0.0560250
Re.4	0.6325997
Re.5	0.1563482
Re.6	0.0264359
Re.7	0.5915649
Re.8	0.0579710
Re.9	0.5396968
Density	0.0000000

TABLE 2.3 – Résultats des test du ratio de vraisemblance pour glm1

	Df	LRT	Pr(>Chi)
Power	1	21.83182	0.0000030
CarAge	1	28.25304	0.0000001
DriverAge	1	211.92967	0.0000000
Brand	6	86.29772	0.0000000
Gas	1	35.54875	0.0000000
Region	9	19.25317	0.0231243
log{*Density*}	1	261.57895	0.0000000

Pour la sélection de variable, on calcule la statistique de Wald pour chaque paramètre estimé. On observe les résultats au Tableau 2.2 obtenus à partir de la fonction **R**, `summary`. La variable `Region` ne semble pas être significative avec deux niveaux qui obtiennent un seuil observé sous 5% et seulement un niveau avec un seuil sous 1%. On vérifie la pertinence de cette variable avec un test de vraisemblance et on retire la variable `Region`.

2.4 Pré-traitement des données pour les réseaux de neurones

On doit traiter les variables avant d'entrainer les réseaux de neurones. Ceux-ci peuvent traiter seulement des variables numériques, ainsi on doit transformer les variables catégorielles. Les variables continues doivent, quant à elles, être standardisées. Cela permet d'avoir une performance plus rapide.

2.4.1 Traitement des variables catégorielles

La variable `Power` est transformée en facteur de 1 à 12. La variable `Gas` est transformée en variable binaire, où `Regular` $= 1$ et `Diesel` $= 0$. Pour les variables `Region` et `Brand`, on utilise deux façons de les traiter et on compare leurs résultats dans l'analyse. La première façon est de créer $(k-1)$ variables binaires pour une variable catégorielle avec k niveaux.

La deuxième méthode qu'on utilise est une technique d'apprentissage de la représentation (« representation learning »). Il s'agit d'ajouter des couches de plongement (« embedding layers ») au réseau qui permettent de réduire la dimension de ces variables catégorielles, voir Richman (2018).

En appliquant la première méthode, on obtient un vecteur de variables explicatives de 20 dimensions

$$\mathcal{X} \subset [1,12] \times \{0,1\} \times \{0,1\}^7 \times \{0,1\}^9 \times [0,100] \times [0,90] \times [2,27000].$$

La deuxième méthode permet de réduire la dimension de `Region` et `Brand` à $d = 2$ et ainsi on obtient un vecteur de variables explicatives de 9 dimensions.

2.4.2 Traitement des variables continues

Premièrement, on applique la transformation *log* à la variable `Density`. Par la suite, on doit s'assurer que chaque variable soit sur la même échelle. On utilise la transformation suivante pour les variables `DriverAge`, `CarAge`, `Power` et `log(Density)` :

$$\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} \mapsto [0,1].$$

2.5 Réseaux utilisés

On détaille dans cette section la démarche utilisée pour construire les différents modèles.

Tout d'abord, on fera une distinction entre les réseaux à une seule couche cachée et ceux à plusieurs couches cachées. Selon , les réseaux de neurones à une seule couche sont capables d'approximer n'importe quelle fonction non linéaire. On verra toutefois que le fait d'ajouter des couches cachées au réseau permet d'apprendre les interactions entre les variables

explicatives plus efficacement. Ensuite, on compare la différence entre les réseaux utilisant l'approche des variables binaires et les réseaux utilisant l'approche des couches de plongement.

On utilise le paquetage `keras` qui est une interface au logiciel du même nom, (Allaire et Chollet, 2019). Comme mentionné à la section 4 de Ferrario et collab. (2018), on est pas intéressé à trouver le minimum total de $\mathcal{L}(\mathcal{D}, \hat{\lambda})$ étant donné de la sur-paramétrisation des réseaux. L'idée est d'arrêter au bon moment l'algorithme de rétro-propagation. On y arrive en établissant une règle d'arrêt qui sera la même pour tous les réseaux ajustés. On arrête l'entrainement lorsque l'erreur de validation ne s'est pas améliorée depuis 20 epochs et on demande au modèle de retourner les paramètres qui permettent d'obtenir la plus petite erreur de validation. De cette façon, on s'assure de ne pas surajuster le modèle. Le Code 2.1 montre comment on implémente cela avec `keras`. De plus, pour chaque réseau on utilise, évidemment, le même échantillon de validation $\mathcal{V}$. On fixe la taille de mini-lot (« mini-batch ») à 8192. On établit un taux de réduction du taux d'apprentissage de 0.05. Ainsi, lorsque l'erreur de validation ne s'est pas améliorée depuis 10 epochs, on obtient un taux d'apprentissage 20 fois plus petit que le précédant. Cela permet à l'algorithme de tenter de quitter des points de selle, où le gradient de la fonction de perte est nul ou presque.

Code 2.1 – Définition des paramètres de l'algorithme

```
model %>% fit(list(XlearnNN, WlearnNN),
        YlearnNN,
        validation_data = list(list(XvalNN,WvalNN),YvalNN),
        epochs = 1000,
        batch_size = 8192,
        callbacks = list(callback_early_stopping(patience = 20,
                        restore_best_weights = T),
        callback_reduce_lr_on_plateau(factor = 0.05)
        )
        )
```

2.5.1 Réseau de neurones « shallow »

Comme mentionné précédemment, on ne cherche pas à trouver *le* meilleur modèle, mais bien *un* des meilleurs modèles. Ainsi, la méthode utilisée pour trouver les hyperparamètres du modèle final vise à améliorer une structure de base avec certains hyperparamètres. Cette structure de base se traduit par le nombre de neurones dans la couche cachée que l'on fixe. Le Tableau 2.4 montre l'erreur de validation pour $q_1 \in \{8, 16, 32, 64, 128, 256, 512\}$ avec la règle d'arrêt mentionnée antérieurement. On fixe donc le nombre de neurones à 32 et on tente d'optimiser le taux d'abandon, d, et les taux de régularisation, ℓ_2 et ℓ_2, à l'aide d'une grille de recherche disponible au Tableau A.1 de l'Appendice A.

On présente les six meilleurs résultats dans le Tableau 2.5. Le réseau qui offre la plus pe-

TABLE 2.4 – Résultats de la recherche du nombre de neurones pour le réseau avec une couche cachée

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	Epochs
32	**0.2511**	0.2503	51
64	0.2516	0.2512	58
16	0.2518	0.2523	45
512	0.2519	0.2490	33
256	0.2519	0.2496	37
128	0.2520	0.2510	33
8	0.2521	0.2523	100

TABLE 2.5 – Résultats de la recherche des hyperparamètres pour le réseau de neurone d'une couche cachée avec 32 neurones

$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	d	ℓ_1	ℓ_2	Epochs
0.2508	0.2512	0.25	0	0	66
0.2510	0.2518	0.50	0	0	70
0.2510	0.2504	0.00	0	0	54
0.2511	0.2519	0.50	0	0.001	79
0.2513	0.2520	0.25	0.001	0.001	82
0.2514	0.2526	0.50	0.001	0.001	125

tite erreur de validation est celui avec $d = 0.25$. On remarque qu'aucun des trois meilleurs modèles n'inclut une régularisation ℓ_1 ou ℓ_2. Une grille de recherche plus grande aurait peut-être permis de diminuer l'erreur à l'aide d'autres valeurs pour les taux de régularisation. On se limite à ces valeurs par souci de temps. Toutefois, on verra à l'aide des autres modèles que le taux d'abandon offre de meilleurs résultats , et ce plus efficacement que les taux de régularisation.

2.5.2 Réseau à deux couches cachées

On utilise la même méthode que pour le réseau à une couche cachée. On tente de fixer le nombre de neurones par couche en premier. On cherche parmi une combinaison de $q_1 \in \{8, 16, 32, 64\}$, et $q_2 \in \{8, 16, 32, 64\}$. On obtient les résultats au Tableau 2.6. La combinaison $(q_1, q_2) = (8, 8)$ donne la plus petite erreur de validation. Cependant, on tente également d'optimiser le modèle avec $(q_1, q_2) = (32, 16)$. L'erreur d'entrainement pour cette combinaison est largement inférieure à $(q_1, q_2) = (8, 8)$ et le nombre d'epochs est plus bas. Il est probable que le nombre plus élevé de paramètres fasse en sorte qu'il surajuste rapidement les données d'entrainement et, ainsi, l'algorithme exerce la règle d'arrêt rapidement. On peut donc penser que ce réseau pourrait bénéficier d'une régularisation quelconque. Ainsi, on fait une grille de recherche pour les deux réseaux mentionnés des hyperparamètres d_1, d_2, ℓ_1 et

TABLE 2.6 – Résultats de la recherche du nombre de neurones par couche pour le réseau avec deux couches cachées

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	Epochs
(8, 8)	**0.2512**	0.2510	64
(32, 16)	**0.2514**	0.2496	46
(16, 32)	0.2514	0.2504	55
(16, 16)	0.2516	0.2505	54
(16, 64)	0.2517	0.2502	49
(32, 32)	0.2517	0.2498	44

ℓ_2. Les valeurs de la grille sont au Tableau A.2 de l'Appendice A. On utilise les mêmes taux de régularisation sur les deux couches cachées, mais on utilise deux taux d'abandon différents. Le Code 2.2 montre aux lignes 3 et 7 qu'on utilise `FLAGS$l1` et `FLAGS$l2` à chaque couche. On utilise également des couches de normalisation, aux lignes 4 et 8, qui permettent de standardiser la sortie de chaque neurone de la couche précédente. Cela permet d'éviter le problème de gradient disparaissant.

Code 2.2 – Définition de la structure du réseau à deux couches cachées

```
1  net <- features.0 %>%
2   layer_dense(units = FLAGS$hidden1,activation="relu",kernel_initializer =
      initializer_he_normal(seed=2L),
3               kernel_regularizer = regularizer_l1_l2(l1 = FLAGS$l1, l2 =
      FLAGS$l2)) %>%
4   layer_batch_normalization() %>%
5   layer_dropout(rate=FLAGS$dropout1) %>%
6   layer_dense(units = FLAGS$hidden2,activation="relu",kernel_initializer =
      initializer_he_normal(seed=3L),
7               kernel_regularizer = regularizer_l1_l2(l1 = FLAGS$l1, l2 =
      FLAGS$l2)) %>%
8   layer_batch_normalization() %>%
9   layer_dropout(rate=FLAGS$dropout2) %>%
10  layer_dense(units=1, activation='linear',
11              weights=list(array(0, dim=c(FLAGS$hidden2,1)),
12                array(log(lambda.hom), dim=c(1))))
```

On présente les six meilleurs réseaux au Tableau 2.7. On sélectionne le modèle avec 32 et 16 neurones et $d_2 = 0.25$. Encore une fois, on remarque qu'aucun des meilleurs modèles ne contient une régularisation ℓ_1 ou ℓ_2.

2.5.3 Réseau à trois couches cachées

Pour le réseau à trois couches cachées, on teste d'abord le nombre de neurones par couche avec la combinaison des paramètres suivants : $q_1 \in \{8, 16, 32\}$, $q_2 \in \{8, 16, 32\}$ et $q_3 \in$

TABLE 2.7 – Résultats de la recherche des hyperparamètres pour les réseaux à deux couches cachées

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	d_1	d_2	ℓ_1	ℓ_2	Epochs
(32, 16)	**0.2510**	0.2504	0.00	0.25	0	0	56
(32, 16)	0.2510	0.2511	0.25	0.25	0	0	72
(8, 8)	0.2511	0.2515	0.00	0.25	0	0	76
(32, 16)	0.2511	0.2503	0.00	0.25	0	0	57
(32, 16)	0.2511	0.2511	0.25	0.00	0	0	111
(8, 8)	0.2512	0.2524	0.00	0.50	0	0	80

$\{8, 16, 32\}$. Un total de 27 possibilités sont testées. On présente les six meilleurs résultats au Tableau 2.8. Pour les mêmes raisons énumérées à la sous-section 2.5.2, on tente d'optimiser les hyperparamètres des modèles $(q_1, q_2, q_3) = (8, 8, 8)$ et $(q_1, q_2, q_3) = (32, 16, 8)$. Par souci de temps, on tente d'optimiser d'abord d_1 et d_2 pour ensuite tenter d'ajouter une régularisation ℓ_1 ou ℓ_2. À noter qu'on n'ajoute pas de taux d'abandon à la troisième couche cachée pour économiser le temps de calcul. On utilise la grille de recherche au Tableau A.3 de l'Appendice A.

Les résultats du Tableau 2.9 montrent que le réseau $(q_1, q_2, q_3) = (32, 16, 8)$ avec un taux d'abandon $d_2 = 0.5$ obtient la plus petite erreur de validation.

2.5.4 Réseau à quatre couches cachées

Comme à la sous-section 2.5.3, on cherche d'abord quelle combinaison de neurones par couches cachées permet d'obtenir la plus petite erreur de validation parmi $q_1 \in \{8, 16, 32\}$, $q_2 \in \{8, 16, 32\}$, $q_3 \in \{8, 16\}$ et $q_4 \in \{8, 16\}$. On fait ensuite une grille de recherche pour les hyperparamètres d_1, d_2 et d_3. Finalement, on optimise les hyperparamètres ℓ_1 et ℓ_2 par une autre grille de recherche, voir Tableau A.4.

TABLE 2.8 – Résultats de la recherche du nombre de neurones par couche pour le réseau avec trois couches cachées

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	Epochs
(8, 8, 8)	**0.2513**	0.2507	72
(8, 8, 16)	0.2519	0.2508	63
(32, 8, 8)	0.2520	0.2501	35
(16, 8, 8)	0.2520	0.2500	49
(32, 8, 16)	0.2521	0.2500	46
(32, 16, 8)	**0.2521**	**0.2492**	37

TABLE 2.9 – Résultats de la recherche des hyperparamètres pour le réseau à trois couches cachées

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	d_1	d_2	ℓ_1	ℓ_2	Epochs
(32, 16, 8)	0.2510	0.2511	0.00	0.50	0	0	72
(8, 8, 8)	0.2511	0.2523	0.25	0.25	0	0	59
(8, 8, 8)	0.2512	0.2514	0.00	0.25	0	0	83
(32, 16, 8)	0.2512	0.2513	0.25	0.00	0	0	54
(8, 8, 8)	0.2513	0.2517	0.25	0.00	0	0	59
(32, 16, 8)	0.2515	0.2528	0.50	0.25	0	0	71

TABLE 2.10 – Résultats de la recherche du nombre de neurones par couche pour le réseau avec quatre couches cachées

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	Epochs
(16,16,16,16)	0.2523	0.2501	34
(16,16,16,8)	0.2524	0.2502	37
(16,32,16,16)	0.2525	0.2494	35
(16,16,8,8)	0.2525	0.2509	32
(16,32,8,16)	0.2526	0.2489	42
(16,8,16,8)	0.2526	0.2513	42

TABLE 2.11 – Résultats de la recherche des hyperparamètres pour le réseau à quatre couches cachées

Nombre de neurones	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	d_1	d_2	d_3	ℓ_1	ℓ_2	Epochs
(16,32,8,16)	0.2514	0.2511	0.0	0.0	0.5	0e+00	0	133
(16,32,8,16)	0.2515	0.2507	0.0	0.0	0.5	0e+00	0	98
(16,16,16,16)	0.2517	0.2529	0.5	0.0	0.0	0e+00	0	67
(16,16,16,16)	0.2517	0.2526	0.5	0.0	0.0	0e+00	0	66
(16,32,8,16)	0.2517	0.2511	0.0	0.0	0.5	1e-06	0	98
(16,16,16,16)	0.2518	0.2523	0.0	0.5	0.0	0e+00	0	62

TABLE 2.12 – Résultats de l'erreur sur les données de test pour tous les modèles

Modèle	$\mathcal{L}\{\mathcal{T}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{D}, \hat{\lambda}\}$	$\mathcal{L}\{\mathcal{V}, \hat{\lambda}\}$	Epochs
CannDeep3	0.25049	0.25086	0.25064	78
Deep3Embedding	0.25051	0.25075	0.25046	53
CannDeep2	0.25054	0.25018	0.25026	51
Deep2Embedding	0.25060	0.25029	0.25026	48
CannShallow	0.25061	0.25099	0.25045	102
ShallowEmbedding	0.25066	0.25093	0.25044	79
Shallow	0.25077	0.25103	0.25076	56
CannDeep4	0.25097	0.25125	0.25112	99
Deep3	0.25099	0.25074	0.25095	80
Deep4Embedding	0.25101	0.25141	0.25084	82
Deep2	0.25102	0.25018	0.25113	70
Deep4	0.25152	0.25044	0.25140	66
Glm	0.25175	0.25311	-	-

2.5.5 Réseaux Cann et réseaux avec couches de plongement

Par souci de temps, les réseaux Cann et les réseaux avec couches de plongement n'ont pas été optimisés au niveau de leurs hyperparamètres. On a utilisé les valeurs trouvées à l'aide des réseaux simples. On optimise seulement le nombre d'epochs à l'aide de la même règle d'arrêt. On présente les résultats finaux au Tableau 2.12 classés en ordre de performance. On constate que tous les modèles performent mieux que le modèle de base. Un autre constat est que les couches de plongement améliorent la performance. En effet, tous les six meilleures modèles sont des modèles avec des couches de plongement. Le seul modèle qui semble s'approcher de la performance est le Shallow, qui arrive à battre les modèles à quatre couches et couches de plongement. Cependant, pour les trois types de modèles, la structure qui performent le moins est celle avec quatre couches. Ainsi, le Shallow bat les modèles à quatres couches par défaut. C'est probablement l'inverse qui se produit, soit qu'une structure à quatre couches est trop complexe pour ce problème.

Une autre importante information qu'on retire de ces résultats est le fait que les modèles Cann sont plus performant que les modèles avec plongement pour chaque structure. L'initialisation par le Glm permet donc d'améliorer les modèles avec constance.

2.6 Comparaison et interprétation des modèles

Cette section est basée sur Molnar (2019) et le chapitre 16 de Boehmke et Greenwell (2019).

On compare dans cette section les modèles Canndeep3, CannShallow et Glm. On vise à comprendre les différences entre le meilleure modèle, le meilleure modèle avec une seule

couche et le modèle de base. On utilise les graphiques de dépendance partielle (PDP), les graphiques d'espérance conditionnelle individuelle (ICE) et la statistique H de Friedman ,Friedman et collab. (2008), pour trouver les interactions entre les variables explicatives. On applique aussi ces techniques au Glm pour pouvoir identifier ses faiblesses.

On résume tout d'abord ces trois techniques.

2.6.1 Interprétation de la boîte noire

Graphique de dépendance partielle

Les PDPs permettent de visualiser comment une variable explicative influence la prévision du modèle. Soit $x_\ell,\ \ell \in \{1, \ldots, p\}$, la variable explicative d'intérêt, on fixe sa valeur et on calcule la moyenne des prévisions sur les valeurs des autres variables explicatives :

$$\bar{f}_\ell\left(x_\ell\right) = \frac{1}{n}\sum_{i=1}^{n} f_{\text{model}}\left(x_\ell, \boldsymbol{x}_i^{(-\ell)}\right).$$

Le vecteur $\boldsymbol{x}_i^{(-\ell)}$ contient toutes les valeurs observées des variables explicatives pour i sauf la variable x_ℓ. En calculant cette moyenne pour plusieurs valeurs fixées de x_ℓ on obtient donc un graphique de la réponse prédite en fonction de la variable d'intérêt.

Graphique d'espérance conditionnelle individuelle

Les ICEs sont semblables aux PDPs. Leur différence est qu'on ne prend pas la moyenne des observations pour une valeur fixe de la variable d'intérêt. On applique :

$$\tilde{f}_{\ell,i}\left(x_\ell\right) = f_{\text{model}}\left(x_\ell, \boldsymbol{x}_i^{(-\ell)}\right).$$

On regarde ainsi comment la variable d'intérêt influence la prévision de chaque observation.

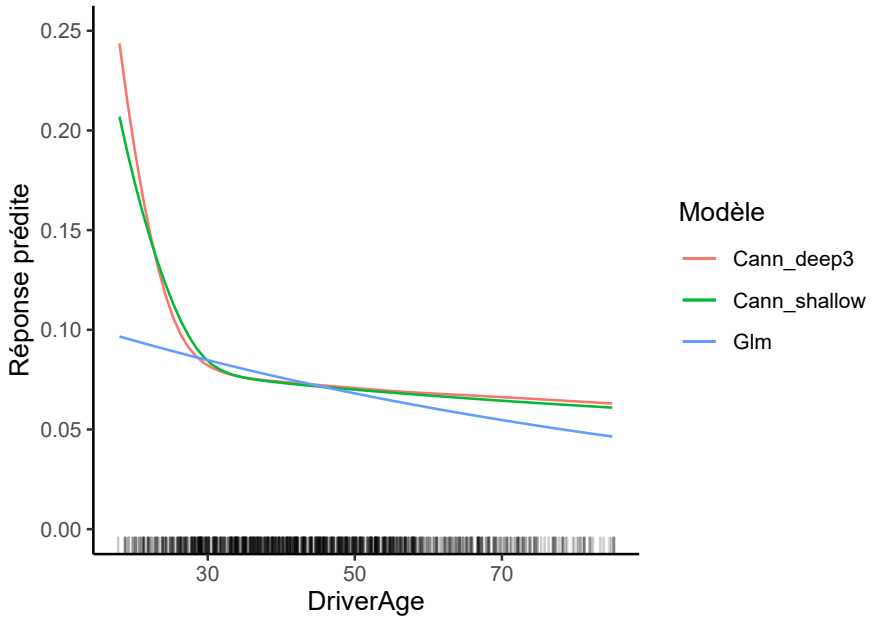
Statistique H de Friedman

La statistique H de Friedman permet de mesurer l'interaction entre deux ou mêmes plusieurs variables explicatives. Elle s'interprète par la part de la variance de la prévision qui est expliquée par l'interaction. On montre l'équation pour l'interaction entre deux variables,x_k et x_ℓ :

$$H_{k\ell}^2 = \frac{\sum_{i=1}^{n}\left\{\bar{f}_{kl}\left(x_{i,k}, x_{i,\ell}\right) - \bar{f}_k\left(x_{i,k}\right) - \bar{f}_l\left(x_{i,\ell}\right)\right\}^2}{\sum_{i=1}^{n}\bar{f}_{kl}^2\left(x_{i,k}, x_{i,\ell}\right)},$$

où $\bar{f}_k\left(x_k\right)$ et $\bar{f}_l\left(x_\ell\right)$ sont les PDPs univariés et $\bar{f}_{kl}\left(x_k, x_\ell\right)$ est le PDP bivarié. Une valeur de $H = 0$ signifie aucune interaction, tandis qu'une valeur de $H = 1$ signifie que l'effet des variables sur la prévision ne se transmet que par l'interaction de ces deux variables.

FIGURE 2.8 – Dépendence partielle pour la variable DriverAge pour les trois modèles sélectionnés



2.6.2 Analyse et interprétation

On utilise le paquetage `iml` pour réaliser l'interprétation des modèles, (Molnar et collab., 2018). Tous les graphiques et calculs ont été réalisés avec un sous-ensemble aléatoire de 1000 observations de l'ensemble de la base de donnée.

Tout d'abord, on passe en revue les PDPs univariés les plus intéressants. On continue ensuite avec les ICEs et puis avec le calcul de la statistique H pour les deux réseaux. À l'aide des résultats de cette statistique, on trace les PDPs bivariés les plus interessants.

La Figure 2.8 montre une lacune du modèle Glm. L'effet de l'âge selon ce modèle est linéaire. Les deux réseaux captent le même effet de cette variable ou presque. Le modèle CannDeep3 prédit des fréquences plus élevées pour les jeunes conducteurs.

La Figure 2.9 montre bien une différence entre les deux réseaux. Comme on le constatait à la Figure 2.4, le modèle CannDeep3 fait ressortir une augmentation de la fréquence prédite pour les véhicules âgés de 5 à 10 ans. Les deux autres modèles ne captent pas cet effet.

L'effet de la variable `Density` est plus difficile à distinguer. On voit clairement que la densité de la population fait augmenter le risque de réclammation pour ce portefeuille dans les trois modèles. Il semble avoir une légère différence pour le modèle CannDeep3 dans les extrémités de la courbe. Elle a une forme en « S », tandis que les deux autres courbes ne sont que convexes.

La Figure 2.12 montre que le modèle CannDeep3 semble capter une interaction puisqu'on

FIGURE 2.9 – Dépendance partielle pour la variable CarAge pour les trois modèles sélectionnés

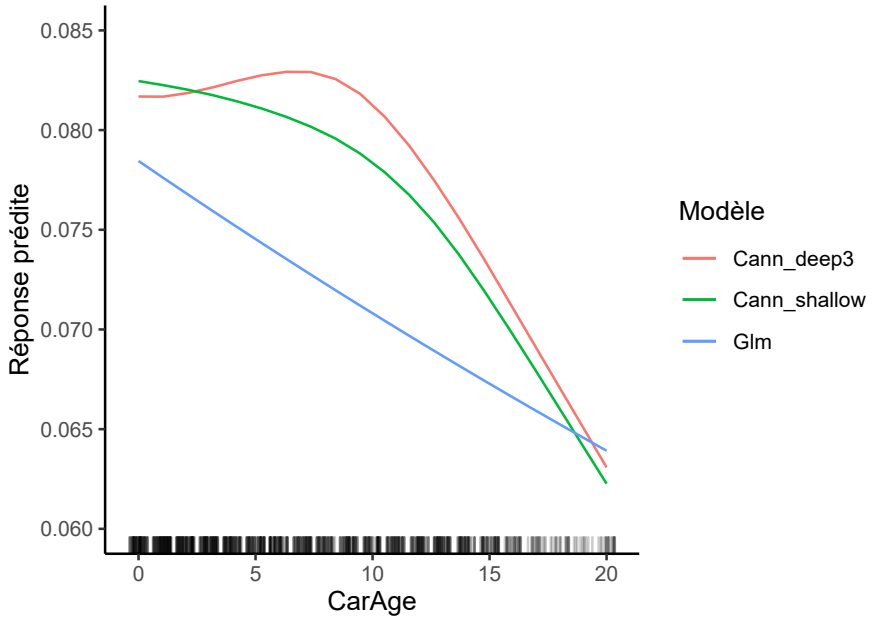


FIGURE 2.10 – ICE pour la variable CarAge pour les trois modèles sélectionnés

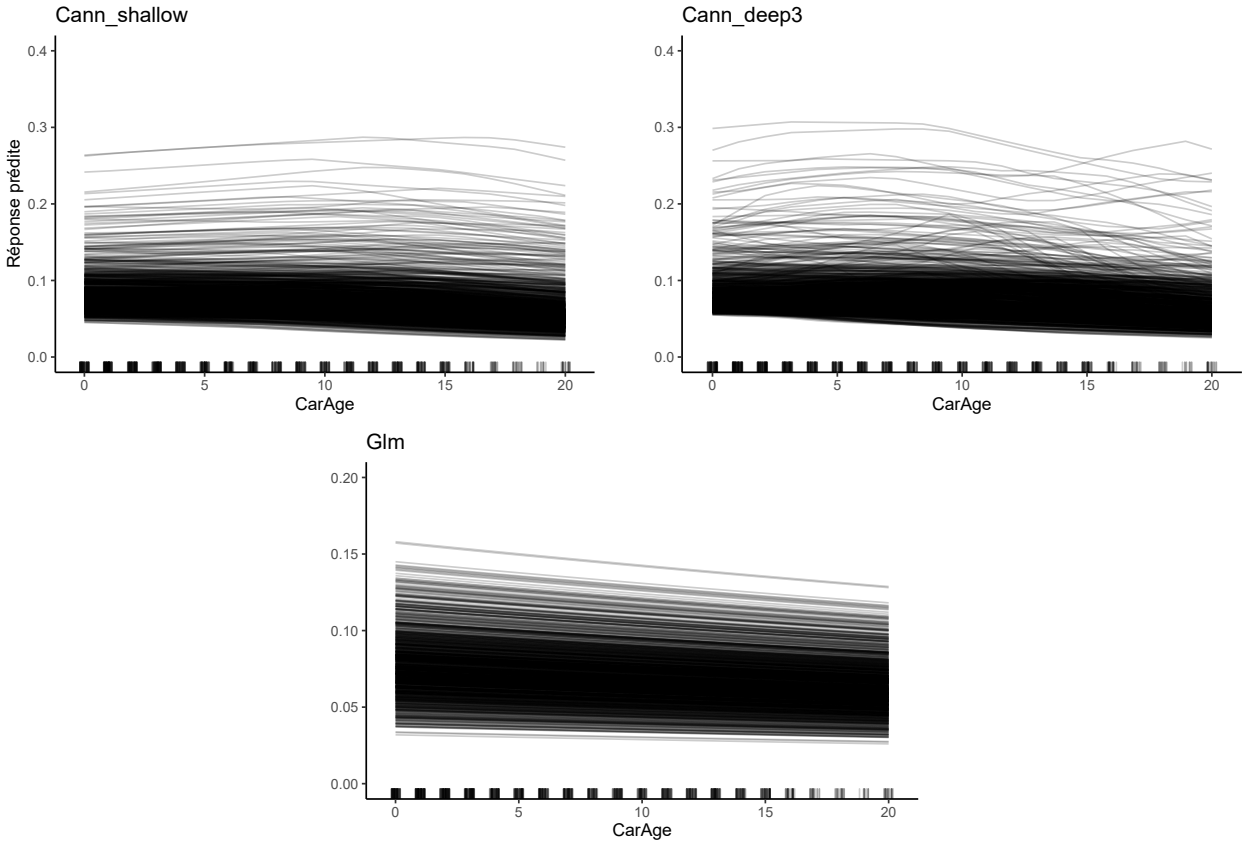
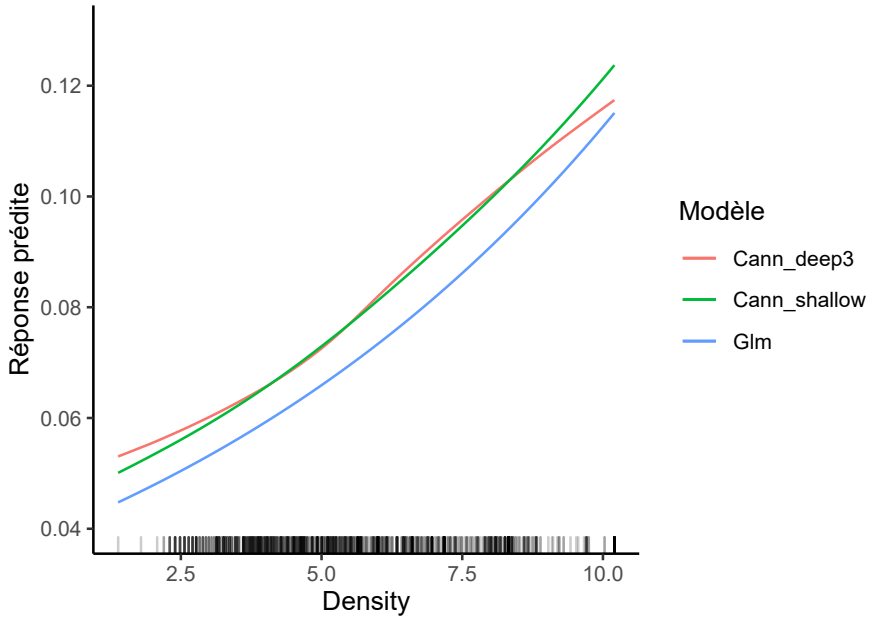


FIGURE 2.11 – Dépendance partielle pour la variable Density pour les trois modèles sélectionnés sur l'échelle logarithmique



observe deux types de courbes. Le modèle prédit une légère augmentation pour une $log(Density)$ de 5 à 10 pour la plupart des assurés. Il y a cependant une certaine quantité d'assurés dont la prévision augmente rapidement pour des valeurs de 0 à 5 pour ensuite se stabiliser.

On observe à la Figure 2.13 que l'interaction de la variable `Gas` avec les autres variables est la plus importante pour le modèle CannDeep3. Pour le modèle CannShallow, il s'agit de la variabe `DriverAge`, Figure 2.14. On note plusieurs différences entre les deux modèles. Premièrement, le modèle CannDeep3 semble détecter plus d'interaction. Il y a cinq variables sur sept qui ont une statistique H supérieure à 0.2, tandis que le second réseau n'en compte que trois. Une autre différence notable est l'importance accordée aux interactions avec la variable `DriverAge`. Le réseau profond la classe en cinquième position et le réseau à une couche la classe en première position.

Les interactions avec la variable `Gas` sont semblables pour les deux modèles. On note une seule différence notable pour l'interaction entre `Density` et `Gas`, où on obtient une statistique H à 0.2 pour le modèle CannDeep3 et 0.15 pour CannShallow.

On saisit mieux à l'aide de la Figure 2.16 la raison possible de l'augmentation de la fréquence prédite des véhicules âgés de 5 à 10 ans par le modèle CannDeep3. On voit qu'il y a une interaction avec les véhicules japonnais. La courbe de cette marque de véhicule est clairement différente. Les autres marques ont toutes une diminution de la fréquence entre les âges mentionnés. Le réseau à une couche ne capte aucunement cet interaction.

FIGURE 2.12 – ICE pour la variable Density pour les trois modèles sélectionnés

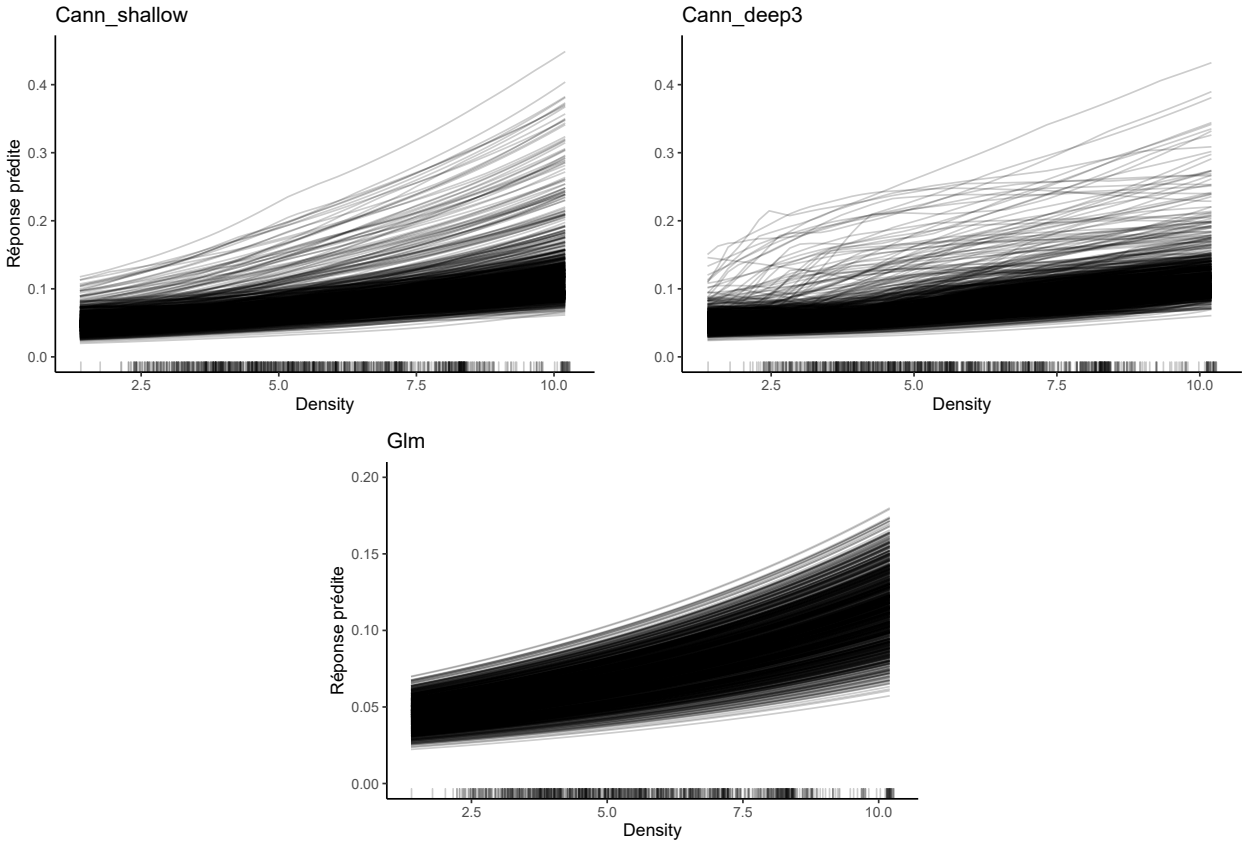


FIGURE 2.13 – Interactions entre les variables explicatives pour le modèle CannDeep3 calculées à partir de la statistique H

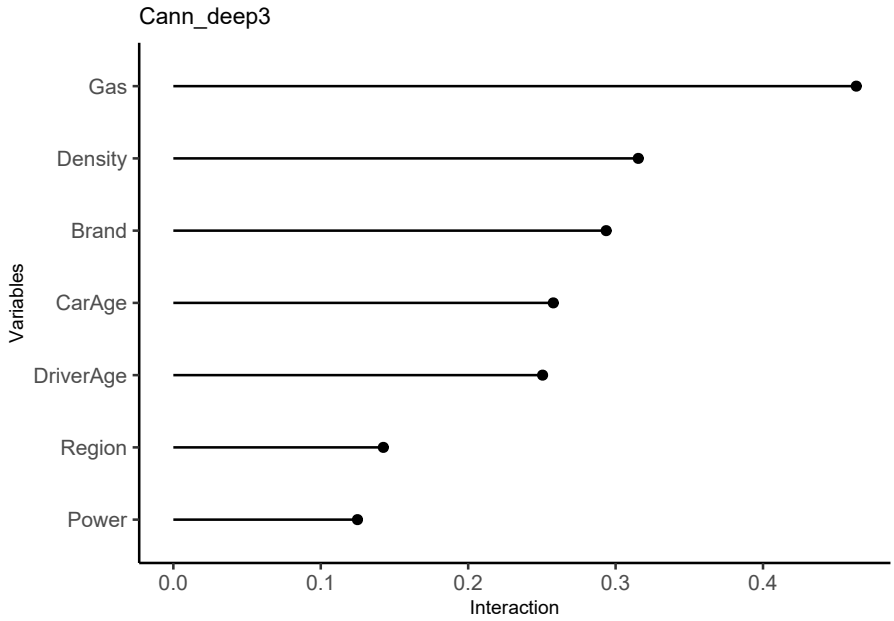


FIGURE 2.14 – Interaction entre les variables explicatives pour le modèle CannShallow

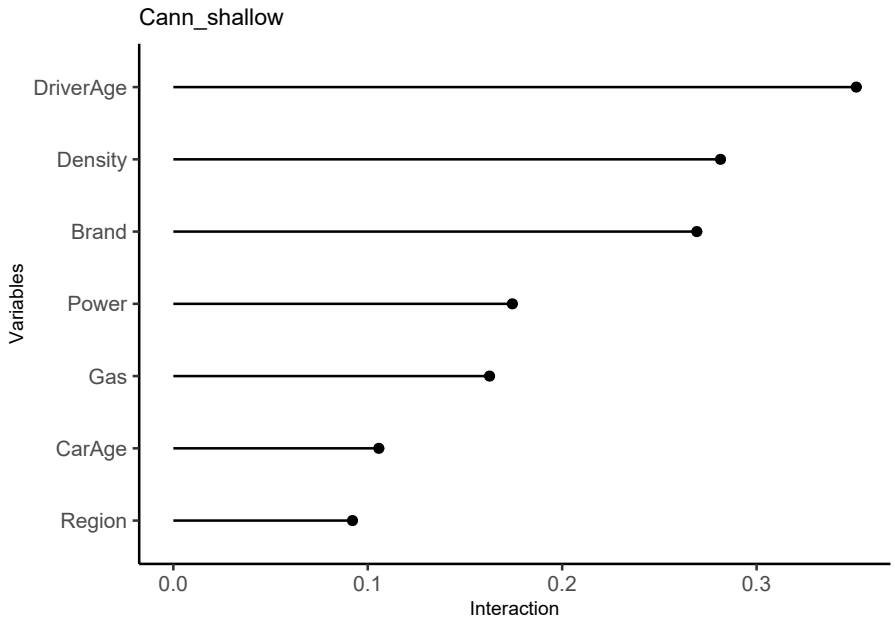


FIGURE 2.15 – Interaction entre la variable Gas et les autres variables explicatives pour les modèles CannDeep3 et CannShallow

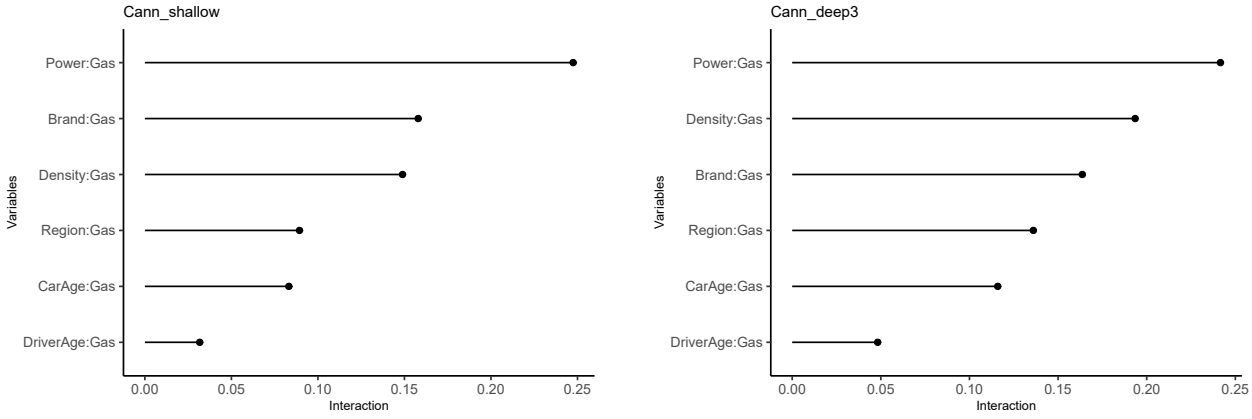
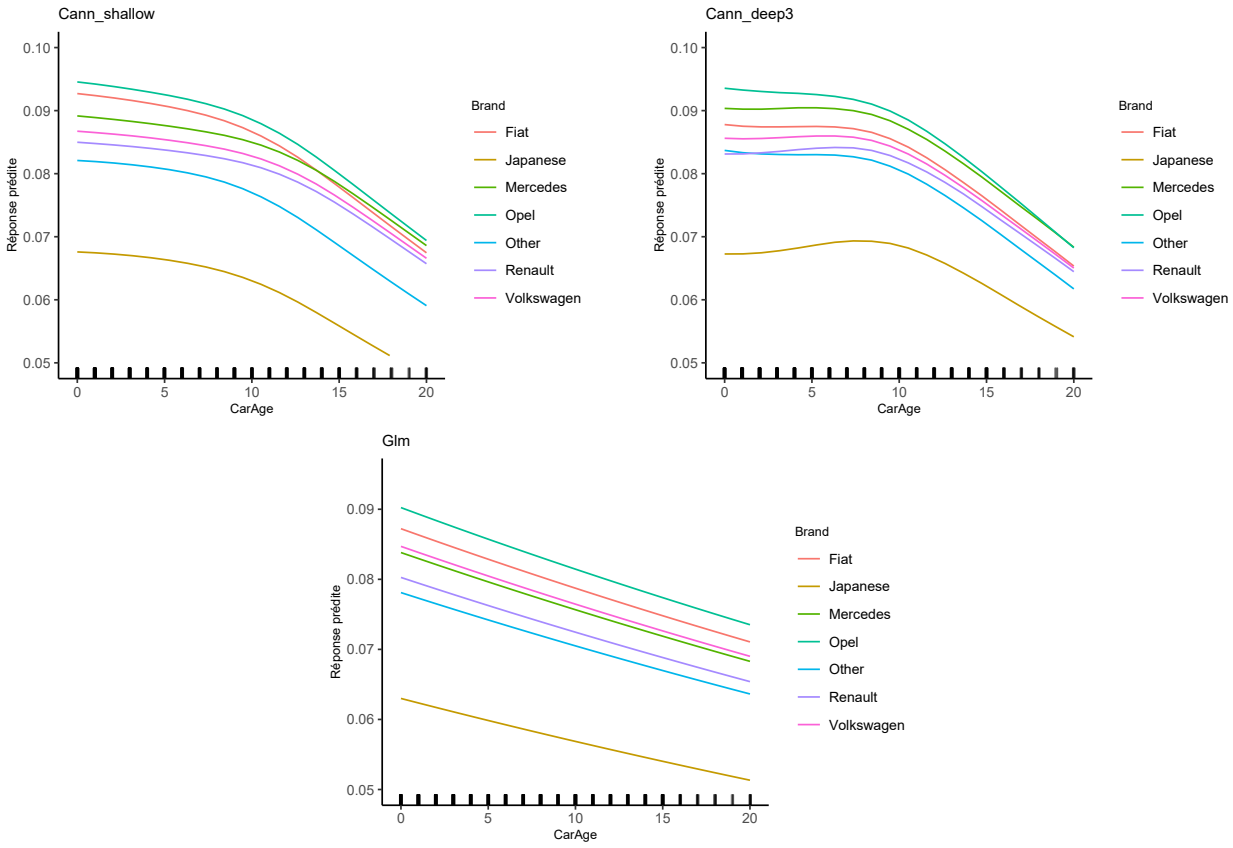


FIGURE 2.16 – Dépendance partielle entre les variables CarAge et Brand pour les trois modèles



On regarde maintenant les interactions avec `DriverAge`. La Figure 2.17 montre pour les deux modèles que les interactions avec `Brand` et `Density` sont les plus significatives. On illustre le PDP bivarié de `DriverAge` et `Density` à la Figure 2.18. On y voit une différence pour les plus jeunes âges. Le modèles CannDeep3 prédit des fréquences élevées pour de jeunes conducteurs (environ moins de 25 ans) en combinaison avec une $log(Density)$ de 2.5 et plus. Le modèle CannShallow prédit des fréquences élevées pour de jeunes conducteurs, aussi, mais seulement pour une $log(Density)$ plus grande que 7.5 environ.

FIGURE 2.17 – Interaction entre la variable DriverAge et les autres variables explicatives pour les trois modèles

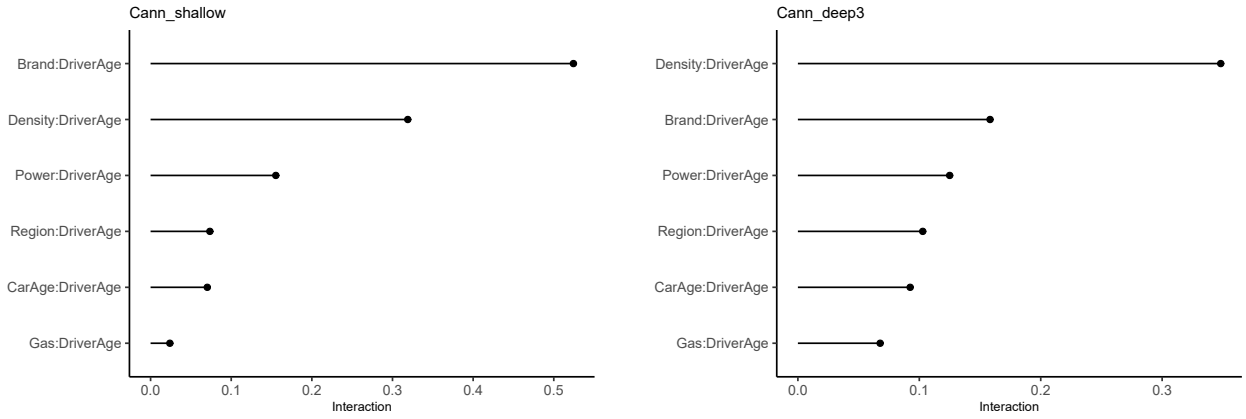
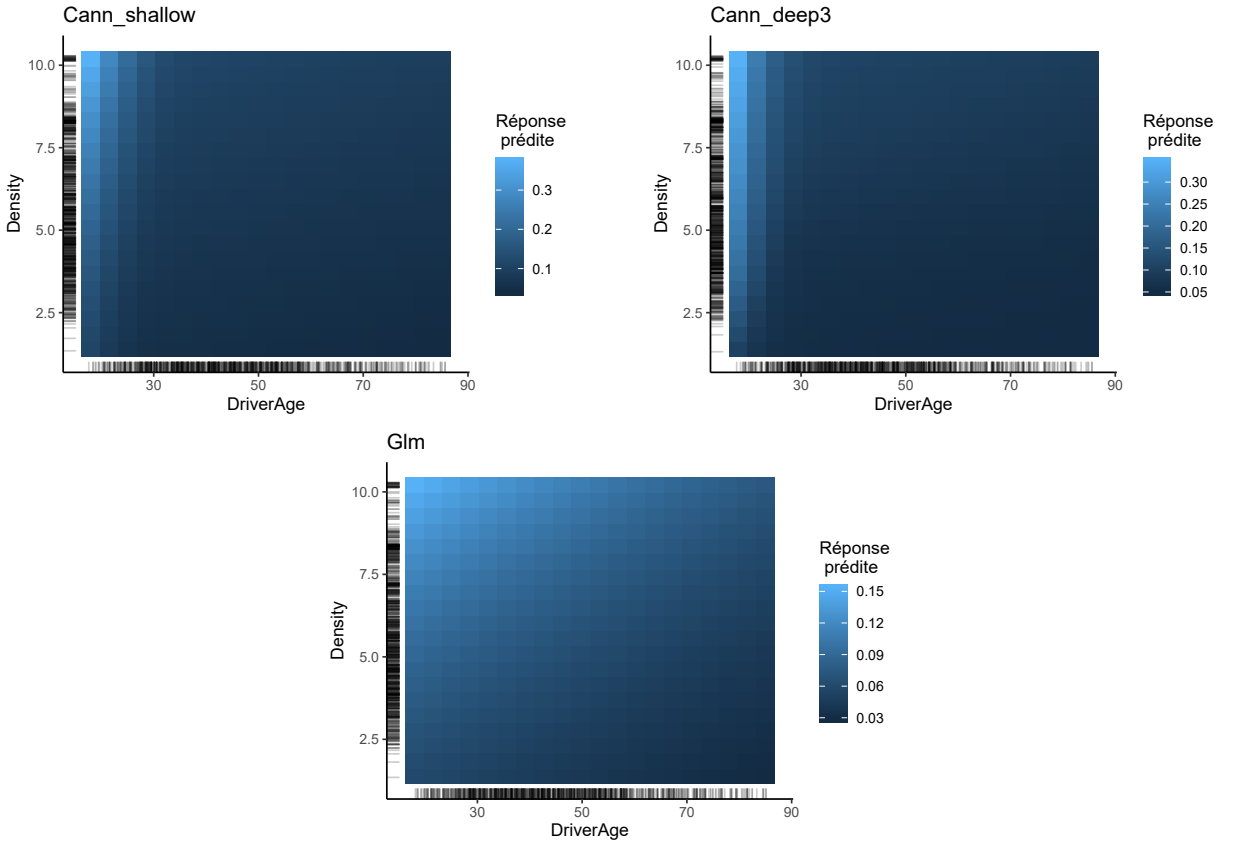


FIGURE 2.18 – Dépendance partielle entre les variables DriverAge et Density pour les trois modèles



Conclusion

La dernière sous-section a permis de montrer que le modèle CannDeep3 a bien plus de facilité à découvrir les interactions. La combinaison entre sa structure et la façon d'initialiser le réseau aura permis de trouver un modèle performant de façon efficace. De plus, les résultats obtenus l'ont été à partir d'hyperparamètres optimisés pour des réseaux différents. On peut alors penser qu'ils bénéficieraient d'une recherche plus approfondie. Une autre avenue qui pourrait être étudiée est l'initialisation. Un modèle de base plus performant que celui utilisé pourrait permettre d'initialiser le réseau d'une meilleure façon encore.

Cependant, la base de données simple et le problème peu compliqué ne sont peut-être pas la meilleure façon de tester l'utilité des réseaux de neurones en actuariat. Une des avenues possibles est l'utilisation de données massives comme les données télématiques d'automobiles pour modéliser la fréquence de réclamation comme Gao et collab. (2019) l'ont fait.

Annexe A

Grilles de recherche

TABLE A.1 – Grille de recherche pour le réseau « shallow » de 32 neurones

Hyperparamètre	Valeurs
d	$\{0, 0.25, 0.5\}$
ℓ_1	$\{0, 0.0001\}$
ℓ_2	$\{0, 0.0001\}$

TABLE A.2 – Grille de recherche pour le réseau à deux couches cachées

Hyperparamètre	Valeurs
d_1	$\{0, 0.25, 0.5\}$
d_2	$\{0, 0.25, 0.5\}$
ℓ_1	$\{0, 0.0001\}$
ℓ_2	$\{0, 0.0001\}$

TABLE A.3 – Grille de recherche pour le réseau à trois couches cachées

Hyperparamètre	Valeurs
d_1	$\{0, 0.25, 0.5\}$
d_2	$\{0, 0.25, 0.5\}$
ℓ_1	$\{0, 0.00001\}$
ℓ_2	$\{0, 0.00001\}$

TABLE A.4 – Grille de recherche pour le réseau à quatre couches cachées

Hyperparamètre	Valeurs
d_1	$\{0, 0.5\}$
d_2	$\{0, 0.5\}$
d_3	$\{0, 0.5\}$
ℓ_1	$\{0, 0.000001\}$
ℓ_2	$\{0, 0.000001\}$

Bibliographie

Allaire, J. et F. Chollet. 2019, *keras : R Interface to 'Keras'*. URL https://CRAN.R-project.org/package=keras, r package version 2.2.5.0.

Boehmke, B. et B. M. Greenwell. 2019, *Hands-On Machine Learning with R*, CRC Press.

Denuit, M. et J. Trufin. 2019, *Effective statistical learning methods for actuaries*, Springer.

Dutang, C. et A. Charpentier. 2019, *CASdatasets : Insurance datasets*. R package version 1.0-10.

Ferrario, A., A. Noll et M. V. Wuthrich. 2018, «Insights from inside neural networks», *Available at SSRN 3226852*.

Friedman, J. H., B. E. Popescu et collab.. 2008, «Predictive learning via rule ensembles», *The Annals of Applied Statistics*, vol. 2, n° 3, p. 916–954.

Gao, G., S. Meng et M. V. Wüthrich. 2019, «Claims frequency modeling using telematics car driving data», *Scandinavian Actuarial Journal*, vol. 2019, n° 2, p. 143–162.

Goodfellow, I., Y. Bengio et A. Courville. 2016, *Deep Learning*, MIT Press. http://www.deeplearningbook.org.

Hastie, T., R. Tibshirani et J. Friedman. 2009, *The Elements of Statistical Learning : Data mining, Inference, and Prediction*, 2e éd., Springer, New York.

He, K., X. Zhang, S. Ren et J. Sun. 2015, «Delving deep into rectifiers : Surpassing human-level performance on imagenet classification», dans *Proceedings of the IEEE international conference on computer vision*, p. 1026–1034.

James, G., D. Witten, T. Hastie et R. Tibshirani. 2013, *An introduction to statistical learning*, vol. 112, Springer.

Molnar, C. 2019, *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.

Molnar, C., B. Bischl et G. Casalicchio. 2018, «iml : An r package for interpretable machine learning», *JOSS*, vol. 3, n° 26, doi :10.21105/joss.00786, p. 786. URL http://joss.theoj.org/papers/10.21105/joss.00786.

Nielsen, M. A. 2015, *Neural networks and deep learning*, vol. 2018, Determination press San Francisco, CA, USA :.

Richman, R. 2018, «Ai in actuarial science», *Available at SSRN 3218082*.

Sanderson, G. «3blue1brown», URL https://www.youtube.com/c/3blue1brown, [En-ligne ; consulté 23 - Mars -2020].

Schelldorfer, J. et M. V. Wuthrich. 2019, «Nesting classical actuarial models into neural networks», *Available at SSRN 3320525*.

Sutskever, I., J. Martens, G. Dahl et G. Hinton. 2013, «On the importance of initialization and momentum in deep learning», dans *International conference on machine learning*, p. 1139–1147.

Wuthrich, M. V. 2019, «From generalized linear models to neural networks, and back», *Available at SSRN 3491790*.

Wuthrich, M. V. et C. Buser. 2019, «Data analytics for non-life insurance pricing», *Swiss Finance Institute Research Paper*, , n° 16-68.