UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

RAPPORT DE PROBLÉMATIQUE

APP1 : Réseaux de neurones artificiels
GRO720

Présenté à
François Grondin

Présenté par
Équipe numéro 9
David Mihai Kibos – kibd5171
Élian-Jacob Dubuc – dube1805

Université de Sherbrooke – 10 Septembre 2025

TABLE DES MATIÈRES

1. DÉVELOPPEMENT

1.1 DESCRIPTION DES COUCHES DE NEURONES IMPLÉMENTÉES

1.1.1 FULLYCONNECTEDLAYER

La couche FullyConnectedLayer (ou couche entièrement connectée) prend en entrée un vecteur de dimension I et calcule une transformation linéaire pour produire un vecteur de dimension O. Chaque sortie est une combinaison linéaire de toutes les entrées, pondérée par des poids et augmentée d'un biais. L'équation générale est la suivante :

$$y = Wx + b \tag{1}$$

Pour augmenter les performances du réseau (convergence plus rapide/explosion du gradient) l'initialisation des poids et biais a été multipliée par la racine de la variance de la variable correspondante. Ainsi, l'initialisation des paramètres découle ainsi :

$$s = \sqrt{\frac{2}{I + O}} \tag{2}$$

Où s est le facteur multiplicatif, I est le compte d'entrée et O est le compte de sortie. Le facteur s se multiplie à l'initialisation des paramètres aléatoires pour influencer leur variance.

Pour la rétropropagation, les gradients de l'entrée et des paramètres sont utilisés pour mettre à jour les poids et propager l'erreur. Les équations de la rétropropagation sont implémentées ainsi :

$$\frac{dL}{dw} = \frac{dL}{dy} x^T \tag{3}$$

$$\frac{dL}{dx} = W^T \frac{dL}{dy} \tag{4}$$

$$\frac{dL}{db} = \sum_{n=1}^{N} \frac{dL_n}{dy} \tag{5}$$

L'utilisation de ces gradients est faite via l'optimisateur qui est présenté plus bas dans ce rapport.

1.1.2 BATCHNORMALIZATION

La couche BatchNormalization a pour rôle de normaliser les activations d'une couche en recentrant et en rééchelonnant les données d'entrée par lot. Pour chaque mini-lot, on calcule une

moyenne et un écart-type afin de transformer les activations. Cette opération permet d'accélérer la convergence et de stabiliser l'entraînement. La normalisation est donnée par l'équation suivante :

$$\hat{x}_i = \frac{(x_i - \mu_B)}{\sqrt{\sigma_B^2 + 10^{-7}}} \tag{6}$$

où μ_B et σ_B^2 sont respectivement la moyenne et la variance du mini-lot, et 10^{-7} est une petite constante servant à éviter les divisions par zéro. Ensuite, la couche applique une transformation affine:

$$y_i = \gamma \hat{x}_i + \beta \tag{7}$$

À noter que les paramètres entrainables dans l'équation ci-dessus sont γ et β.

Lors de l'entraînement, la normalisation utilise la moyenne et la variance des mini-lots. En revanche, pendant la phase d'évaluation, la couche s'appuie sur des statistiques globales (moyenne et variance) mises à jour de façon récursive au cours de l'entraînement.

Dans l'implémentation, un paramètre α est utilisé pour pondérer cette mise à jour :

$$\mu_{global} \leftarrow \alpha * \mu_{global} + (1 - \alpha)\mu_B \tag{8}$$

$$\sigma^2_{global} \leftarrow \alpha * \sigma^2_{global} + (1 - \alpha)\sigma_B^2 \tag{9}$$

Le paramètre α agit comme un filtre passe-bas exponentiel (un filtre récursif de type IIR). Un α proche de 1 ralentit l'évolution des statistiques globales, ce qui les rend plus stables mais moins réactives aux changements dans les données. Et à l'opposé, un α proche de 0 les rend plus sensibles aux mini-lots courants, mais introduit davantage de bruit et d'oscillations. En d'autres termes, diminuer α revient à réduire la fréquence de coupure du filtre, ce qui produit une moyenne plus lisse mais moins représentative des variations rapides dans les données. Ainsi, α contrôle directement la qualité de la normalisation en phase d'évaluation en régulant le compromis entre stabilité et réactivité des statistiques globales.

Lors de la rétropropagation, l'objectif est de déterminer comment ajuster les paramètres entraînables γ et β, ainsi que de calculer le gradient par rapport à l'entrée x, en tenant compte de la normalisation. Les gradients associés aux paramètres sont :

$$\frac{dL}{d\gamma} = \sum_{i=1}^{M} \frac{dL}{dy_i} * \hat{x}_i \tag{10}$$

$$\frac{dL}{d\beta} = \sum_{i=1}^{M} \frac{dL}{dy_i} \tag{11}$$

Pour calculer le gradient par rapport à x_i, il est nécessaire de prendre en compte la dépendance à la fois à la moyenne et à la variance du mini-lot.

$$\frac{dL}{dx_i} = \frac{dL}{d\hat{x}_i}\frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{2}{M} * \frac{dL}{d\sigma_B^2}(x_i - \mu_B) + \frac{1}{M}\frac{dL}{d\mu_B} \tag{12}$$

Avec :

$$\frac{dL}{d\hat{x}_i} = \frac{dL}{dy_i} * \gamma \tag{13}$$

$$\frac{dL}{d\sigma_B^2} = \sum_{i=1}^{M} \frac{dL}{d\hat{x}_i} * (x_i - \mu_B) * -\frac{1}{2}(\sigma_B^2 + \epsilon)^{-\frac{3}{2}} \tag{14}$$

$$\frac{dL}{d\mu_B} = \sum_{i=1}^{M} \frac{dL}{d\hat{x}_i} * \left(-\frac{1}{\sqrt{\sigma_B^2 + \epsilon}}\right) + \frac{dL}{d\sigma_B^2} * -\frac{2}{N}\sum_{i=1}^{M}(x_i - \mu_B) \tag{15}$$

Cette classe permet de réduire le phénomène de gradient explosif et accélère la convergence.

1.1.3 ReLU

La couche ReLU (Rectified Linear Unit) sert à introduire une non-linéarité dans le réseau de neurones. Cette fonction d'activation met toutes les valeurs d'entrée négatives à 0 et conserve inchangées les valeurs d'entrée positives. L'équation de la ReLU est :

$$y = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{16}$$

Cette fonction permet d'éviter les problèmes de saturation observés avec d'autres fonctions d'activation comme la sigmoïde. Lors de l'entraînement, la rétropropagation nécessite de calculer le gradient de la perte L par rapport à l'entrée x:

$$\frac{dL}{dx} = \frac{dL}{dy} * \frac{dy}{dx} \tag{17}$$

La dérivée de la ReLU est :

$$\frac{dy}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \quad (18)$$

Ainsi, le gradient transmis à l'entrée est :

$$\frac{dL}{dx} = \begin{cases} 0 & x < 0 \\ \frac{dL}{dy} & x \geq 0 \end{cases} \quad (19)$$

En d'autres mots le gradient ne se propage pas pour $x < 0$.

1.1.4 SIGMOÏDE

La couche Sigmoïde est une fonction d'activation qui transforme toute valeur réelle en un nombre compris entre 0 et 1. Elle est souvent utilisée pour des sorties devant représenter une probabilité ou dans des réseaux peu profonds. L'équation implémentée pour cette couche est :

$$y = \frac{1}{1 + e^{-x}} \quad (20)$$

Lorsque x est très grand, y tend vers 1, et lorsque x est très petit, y tend vers 0.

Pour la rétropropagation, il est nécessaire de calculer le gradient de la perte L par rapport à l'entrée x. La dérivée de la Sigmoid est donnée par :

$$\frac{dy}{dx} = (1 - y)y \quad (21)$$

Le gradient transmis à l'entrée de la couche devient donc :

$$\frac{dL}{dx} = \frac{dL}{dy}(1 - y)y \quad (22)$$

Le principal désavantage de cette fonction d'activation est que le gradient devient très faible lorsque y est proche de 0 ou de 1, ce qui peut ralentir significativement la convergence du réseau.

1.1.5 CROSSENTROPYLOSS

Fonction de coût utilisée pour le réseau. À l'aide du Softmax, elle mesure l'écart entre la solution prédite par le modèle et la solution réelle. La fonction Softmax fournit une probabilité de classification pour chacune des sortie, la fonction évalue par la suite ces probabilités. Cette fonction est utilisée en « One-Hot » puisqu'elle reçoit une sortie de réseau de classification. La fonction est appliquée à un cas comme ici ou il y a une tache de classification.

$$L = -\sum_{j=1}^{J} y_j \log(\hat{y}_j) \tag{23}$$

1.1.6 MeanSquareErrorLoss

La fonction de coût Mean Squared Error (MSE) mesure l'écart quadratique moyen entre les valeurs prédites par le réseau et les valeurs cibles réelles. Elle est principalement utilisée dans les problèmes de régression, car elle amplifie rapidement les grandes erreurs et favorise une optimisation plus sensible aux écarts importants. L'équation de la MSE est la suivante :

$$L = \frac{1}{J}\sum_{j=1}^{J}(\hat{y}_j - y_j)^2 \tag{24}$$

Le gradient de la perte par rapport aux entrées est présenté ci-dessous :

$$\frac{dL}{dx} = \frac{2}{J}(\hat{y}_j - y_j) \tag{~~25(~~25}$$

Dans le cadre de ce modèle, la MSE n'a pas été utilisée car il s'agit d'un problème de classification.

1.1.7 SgdOptimizer

Implémente la descente stochastique. Optimise les paramètres à l'aide de leur gradient et de leur poids. Sa principale fonction transforme les paramètres en appliquant un nouveau poids et en suivant le pas d'apprentissage dicté (Learning rate).

1.2 Fonction de coût

1.2.1 Calcul du coût

Comme il s'agit d'un problème de classification multi-classes, on utilise l'entropie croisée plutôt de l'erreur quadratique moyenne (MSE). La classe CrossEntropyLoss combine l'entropie croisée avec la fonction softmax, qui transforme les scores du réseau en probabilités normalisées.

La fonction de coût générale pour l'entropie croisée s'écrit :

$$L = -\sum_{j=1}^{J} y_j \log \hat{y}_j \tag{26}$$

La fonction Softmax est définie comme :

$$\hat{y}_j = \frac{e^{x_j}}{\sum_{i=1}^{I} e^{x_i}} \tag{27}$$

Puisque la cible y est représentée en one-hot, tous les éléments sont nuls sauf celui correspondant à la vraie classe t. La fonction de perte se simplifie alors en :

$$L = -\log \hat{y}_t \tag{28}$$

En substituant la définition de la softmax :

$$L = -\log\left(\frac{e^{x_t}}{\sum_{i=1}^{I} e^{x_i}}\right) \tag{29}$$

Le développement ci-dessous présente la simplification de cette fonction en utilisant propriétés algébriques des logarithmes :

$$\log\frac{a}{b} = \log a - \log b \tag{30}$$

$$L = -\left(\log e^{x_t} - \log \sum_{i=1}^{I} e^{x_i}\right) \tag{31}$$

$$L = -x_t + \log \sum_{i=1}^{I} (e^{x_i} + 10^{-7}) \tag{32}$$

En Le petit terme 10^{-7} est ajouté pour éviter les problèmes numériques liés à $\log(0)$.

1.2.2 CALCUL DU GRADIENT

Le gradient de la perte par rapport aux entrées x_i peut être calculé en considérant deux cas :

$$\frac{d(-x_t)}{dx_i} = \begin{cases} -1 \; si \; i = t \\ 0 \; si \; i \neq t \end{cases} \tag{33}$$

$$\frac{d\left(\log \sum_{j=1}^{I} (e^{x_j})\right)}{dx_i} = \frac{1}{\sum_{j=1}^{I} (e^{x_j})} \tag{34}$$

$$\frac{d\left(\sum_{j=1}^{I} (e^{x_j})\right)}{dx_i} = e^{x_i} \tag{35}$$

$$\frac{dL}{dx_i} = \begin{cases} -1 + \frac{e^{x_i}}{\sum_{j=1}^{I}(e^{x_j})} \; si \; i = t \\ 0 + \frac{e^{x_i}}{\sum_{j=1}^{I}(e^{x_j})} \; si \; i \neq t \end{cases} = \begin{cases} \hat{y}_t - 1 \; si \; i = t \\ \hat{y}_i \; si \; i \neq t \end{cases} \tag{36}$$

En utilisant la représentation one-hot ($y_i = 1$ pour la classe correcte sinon c'est 0), cette expression se généralise en :

$$\frac{dL}{dx_i} = \hat{y}_i - \; y_i \tag{37}$$

Cette formulation permet de vectoriser les calculs dans le code Python, comme dans la classe CrossEntropyLoss, et facilite l'optimisation par rétropropagation.

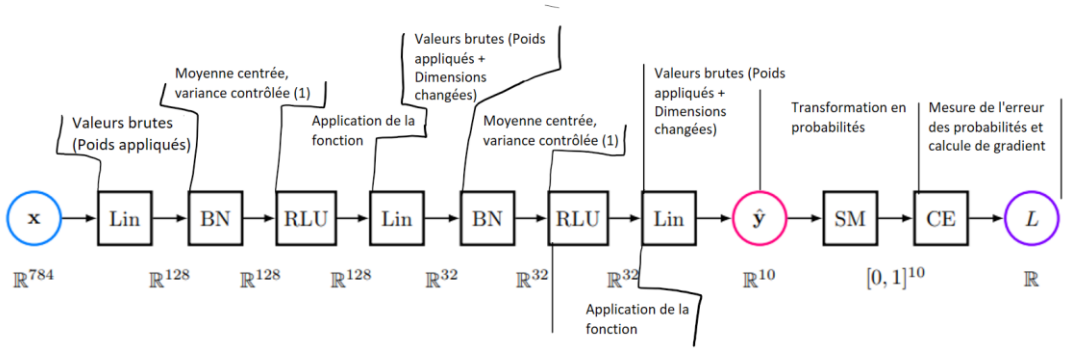
1.3 OPTIMISATION

L'objectif de la descente de gradient est de calibrer les paramètres des neurones afin de minimiser la fonction de cout. Le code met en place une descente de gradient stochastique.

$$v_{t+1} = v_t - \mu \frac{dL}{dv}\Big|_{v_t} \tag{38}$$

Où μ est le taux d'apprentissage. Dans notre cas, ce paramètre est fixé à 0.01. L'expérimentation avec différentes valeurs montre qu'un taux d'apprentissage trop élevé peut entraîner des oscillations autour du minimum ou même une divergence de l'entraînement et un taux d'apprentissage trop faible ralentit considérablement la convergence. Pour réduire la charge de calcul, l'optimisation utilise la descente de gradient stochastique (SGD), qui approxime le gradient en se basant sur un mini-lot plutôt que sur l'ensemble complet des données.

1.4 SCHÉMA BLOC DE L'ARCHITECTURE



Le schéma bloc ci-dessus montre l'utilité des données entrantes et sortantes de chacun des blocs du modèle. Chaque bloc du schéma correspond à une classe logicielle.

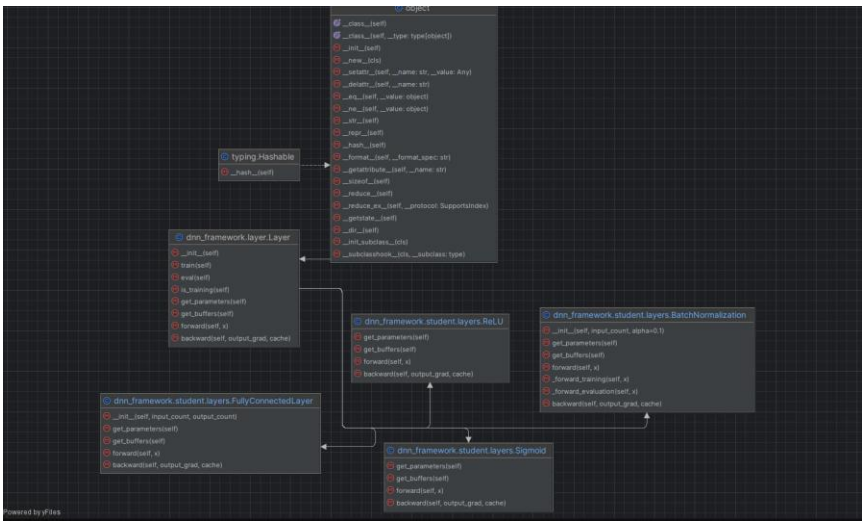


Figure 1 - Diagramme de classes

1.5 COURBES D'APPRENTISSAGE

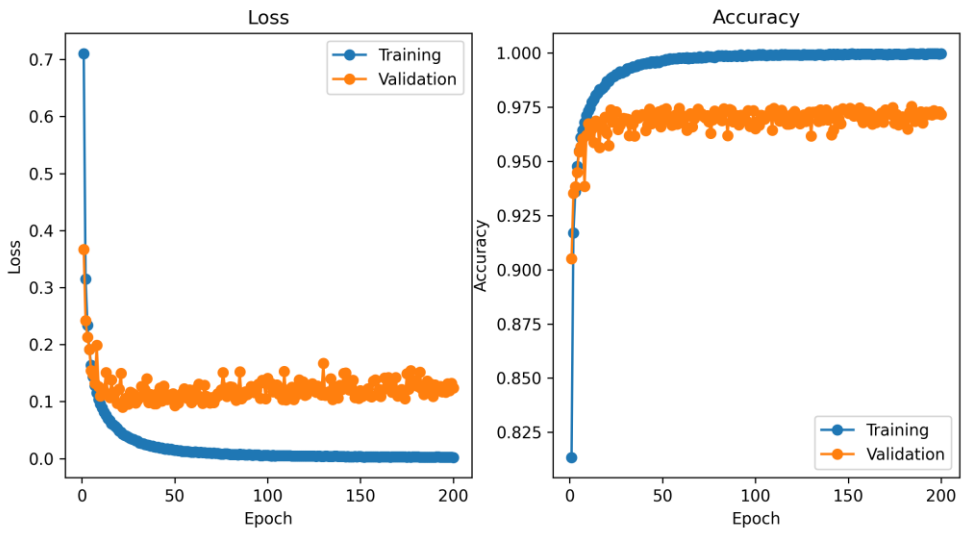


Figure 2: Courbe d'apprentissage du réseau de neurones

La courbe de perte montre que le modèle apprend correctement à partir des données d'entraînement. La valeur de la perte diminue au fur et à mesure que le nombre d'époques augmente, ce qui indique que le modèle s'entraîne de manière continue sans montrer de signes de surapprentissage. Il est à noter que le réseau de neurones utilisé n'est pas suffisamment complexe pour observer un phénomène de surapprentissage ; ajouter des neurones ou des couches supplémentaires serait nécessaire pour le mettre en évidence.

La courbe de précision montre que le modèle généralise correctement lors des phases de validation. L'impact des premières époques se fait rapidement sentir : les mauvaises prédictions initiales sont rapidement corrigées après quelques itérations. La précision se stabilise après environ 50 époques. L'entraînement a été prolongé jusqu'à 200 époques pour maximiser les résultats, mais très peu d'amélioration est observée après 100 époques en raison de la faible complexité du réseau.

Ainsi, le modèle atteint un taux de précision de 97,32 % sur le jeu de test. Plusieurs essais ont été effectués pour déterminer la taille optimale des mini-lots, et une taille de 50 échantillons s'est révélée offrir un taux de précision satisfaisant.