TD1 - Predicting Cancer Using Shallow Neural Networks

Juliette Bluem

16 janvier 2023

1 Set up the network architecture

On vient changer le nombre de neurones (node) : de 10 à 2.
numOfActivationNodes = 2 ;
En réduisant le nombre de neurones, nous réduisant la complexité du réseau. Si nous obtenons une classification aberrante, nous l'augmenterons légèrement. Attention toutes fois au suraprentissage. Lorsque le nombre de nœuds d'un réseau de neurones est trop élevé, il peut facilement surapprendre les données d'entraînement, ce qui signifie qu'il ne sera pas capable de généraliser ses résultats à de nouvelles données. En réduisant le nombre de nœuds, on peut éviter cela et améliorer les performances du modèle sur des données de test.

On ne peut rien changer d'autre sur l'architecture du réseau.

2 Assign various training and hyperparameters

L'hyper paramètre principal est le nombre d'epochs. On le garde fixé à 5000 dans un premier temps.

Le nombre d'époques est le nombre de fois que le modèle va parcourir l'ensemble des données d'entraînement.

On peut vouloir le réduire. Plus un modèle parcourt les données d'entraînement, plus il a de chances de surapprendre les données d'entraînement. Entraîner un modèle pendant un trop grand nombre d'époques peut nécessiter beaucoup de temps et de puissance de calcul. En réduisant le nombre d'époques, on peut réduire les besoins en ressources et rendre l'entraînement plus rapide.

On peut également vouloir l'augmenter. Les réseaux de neurones sont des modèles à optimisation complexe, et l'entraînement peut se heurter à des minimums locaux qui limitent les performances. En augmentant le nombre d'époques, on peut augmenter les chances de trouver un minimum global qui donne des meilleurs résultats.

Il est important de noter que le choix optimal du nombre d'epochs dépendra des données, de la complexité du modèle et des objectifs de l'application, il est donc nécessaire de tester différents paramètres pour trouver celui qui convient le mieux.

Attention, si on lui donne une valeur aberrante (trop grande), l'algorithme la changera afin d'éviter le suraprentissage.

Nous pouvons ensuite choisir notre loss function.

Une fonction de perte (ou "loss function" en anglais) est une fonction mathématique qui mesure l'erreur d'un modèle de réseau de neurones par rapport aux données d'entraînement. Plus précisément, c'est une fonction qui prend en entrée les sorties prédites par le modèle et les valeurs réelles, et qui renvoie une valeur qui indique à quel point les sorties prédites sont éloignées des valeurs réelles.

Il existe de nombreuses fonctions de perte différentes qui peuvent être utilisées selon le type de modèle et les données.

On commence notre entraînement avec une fonction cross-entropy.

2.1 Training the model

On utilise la fonction train pour réaliser l'apprentissage du réseau de neurones.
Ainsi, nous obtenons le modèle entraîné et une structure qui contient des données sur l'apprentissage effectué.
Nous avons pour cela eu besoin en entrée de notre réseau vierge et de X et Y qui sont les données de base sur nos cellules cancéreuses (X) et la décision dite de vérité (Y). C'est à dire la classification entre cellule bénigne et maligne. En effet, nous faisons de la supervision.

2.2 Computing the prediction error

Une fois le modèle entraîné sur les données, nous pouvons tester sa capacité de prédiction. Pour cela, nous appelons notre modèle entraîné sur X.

Nous comparons les erreurs de prédiction (à savoir celles entre le Y et la prédiction que le modèle a effectué). Pour cela, nous disposons de la fonction perform. Cette dernière nous retourne un réel entre 0 et 1. Plus il est proche de 1, plus le modèle est performant.

2.3 Visualizing and interpreting the model results

Nous pouvons maintenant modifier differents paramètres afin de visualiser leur effet.
Pour cela, nous prenons un cas témoins avec des paramètres choisis. Pour ce témoin, nous définissons 2 couches de neurones et attribuons 70% de la base à l'entraînement.
Nous allons ensuite modifier un à un ces paramètres. Tout d'abord en augmentant le nombre de neurones, puis en diminuant les données destinées à la phase d'entraînement.

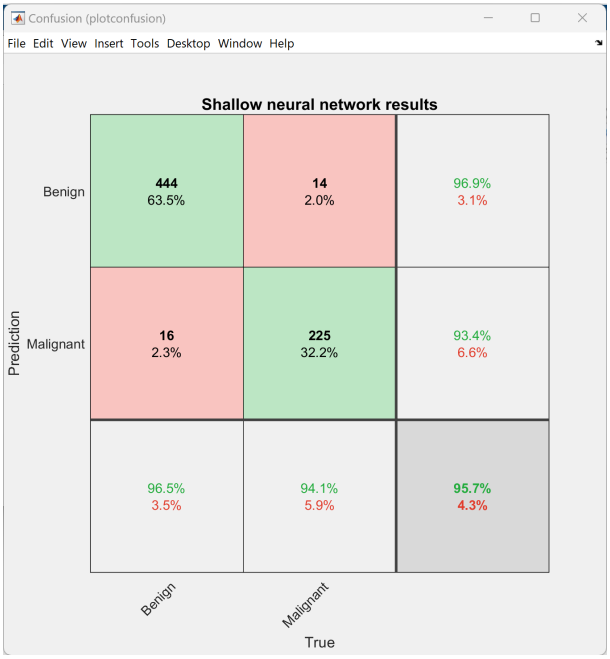

FIGURE 1 – Résultats du témoin

En augmentant le nombre de neurones, nous obtenons les mêmes résultats suite à la regression. Cependant, au lieu de prendre 2min, les calculs en ont duré 25.
Nous avons donc beaucoup de ressources consomées inutilement.

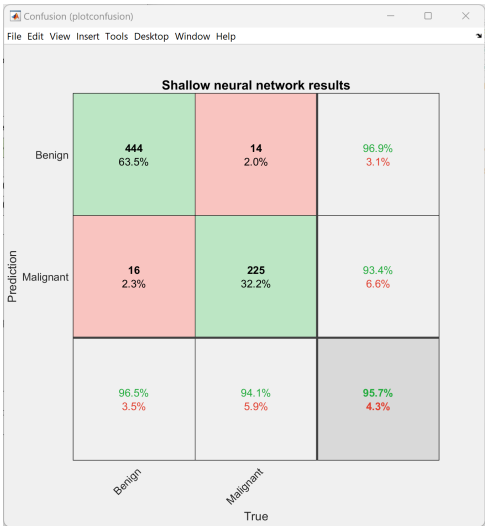

FIGURE 2 – Augmentation NN

En diminuant le nombre de données attribuées à la phase d'entraînement, nous observons que notre modèle est moins précis. Nous sommes passé de 70% de la base dédié à cette phase à 30%. Je m'attendais honnêtement à un écart plus significatif. Mais nous voyons tout de même une augmentation du nombre de "faux négatifs".

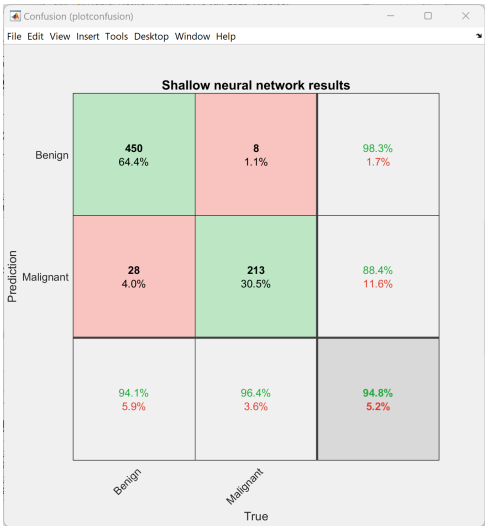

FIGURE 3 – Plus petite base d'entraînement

3 Conclusion

Pour conclure, ce TP nous a permis de se rappeler les bases de la classification et de la régression avec une application différente que celle que nous avions déjà réalisée dans un précédent cours.
En effet, nous voyons donc que la régression appliquée sur des images et très proche de celle utilisée pour des choix médicaux d'un point de vue général.

Ces deux heures nous ont également permis de nous réutiliser Matlab et de nous refamiliariser avec son langage.