CSI 4506 Introduction à l'intelligence artificielle

Devoir 3 : Réseaux de neurones

Marcel Turcotte

Version: oct. 21, 2024 08h13

◎ Objectifs d'apprentissage

1. **Implémenter** des modèles de référence et des réseaux de neurones ; justifier les choix de modèles en fonction de la pertinence de la tâche.
2. **Appliquer** des techniques d'ajustement des hyperparamètres et de régularisation pour améliorer les performances des modèles.
3. **Évaluer** les modèles en utilisant la validation croisée et des métriques de performance ; formuler des recommandations sur les modèles.

En 2024, la communauté scientifique a reconnu l'impact profond de l'apprentissage automatique, comme en témoignent les prix Nobel en Physique et en Chimie. Le prix de Physique a été décerné pour des « découvertes et inventions fondamentales qui font progresser l'**apprentissage automatique** à travers les **réseaux de neurones** artificiels », tandis que le prix de Chimie a honoré le développement d'un « **modèle d'IA** qui résout un problème de longue date, soit la prédiction des structures complexes des protéines ».

Le Devoir 3 est inspiré de ces réalisations. Le **problème de prédiction de la structure des protéines** a été un défi scientifique majeur pendant plus de cinq décennies. Une équipe de Google DeepMind, dirigée par Demis Hassabis et John M. Jumper, a développé un modèle d'IA appelé AlphaFold. Ce modèle est le premier à prédire les structures complexes des protéines avec une précision comparable aux méthodes expérimentales.

Les protéines sont des chaînes d'acides aminés qui se replient en formes spécifiques, essentielles à leurs fonctions. Ces formes sont organisées en quatre niveaux structuraux : 1) La structure primaire est la séquence des acides aminés. 2) La structure secondaire comprend des motifs locaux comme les hélices α et les feuillets β. 3) La structure tertiaire est la forme tridimensionnelle globale d'une seule chaîne de protéines. 4) La structure quaternaire se forme lorsque plusieurs chaînes de protéines se combinent en un complexe plus large.

Avant la percée d'AlphaFold, la prédiction de la structure secondaire des protéines était considérée comme une étape intermédiaire vers la solution complète du problème de prédiction de la structure des protéines.

La **prédiction de la structure secondaire des protéines** consiste à identifier les structures locales (α-hélice, β-feuillet ou enroulement) qu'une séquence de protéines adoptera, sur la base de sa séquence d'acides aminés. Cette séquence est représentée comme une chaîne utilisant un alphabet de 20 lettres, chaque lettre correspondant à un acide aminé. L'objectif est de transformer cette séquence vers un alphabet

de 3 lettres, où « H » désigne une hélice α, « E » désigne un feuillet β et « C » désigne un enroulement, décrivant ainsi la structure secondaire à chaque position.

Les praticiens de l'apprentissage automatique se sont intéressés à ce problème en raison de son accessibilité pour ceux qui n'ont pas de formation scientifique formelle. La tâche consiste à concevoir un programme informatique qui prend en entrée une chaîne sur un alphabet de 20 lettres (la séquence d'acides aminés) et produit une chaîne de même longueur en utilisant un alphabet de 3 lettres (la structure locale). Le problème peut être conceptualisé comme un chiffrement : traduire un message codé avec 20 lettres en un message codé avec trois lettres.

L'exemple suivant montre la transformation d'une séquence d'entrée de 46 caractères, où chaque caractère représente un acide aminé unique, en une structure de sortie de longueur identique. Cette sortie utilise trois caractères spécifiques : C, E et H.

```
input_sequence   = "TTCCPSIVARSNFNVCRLPGTPEAICATYTGCIIIPGATCPGDYAN"
output_structure = "CEECCCHHHHHHHHHHHHCCCCHHHHHHHHCCEECCCCCCCCCCCC"
```

Comme dans les langues naturelles, où la signification d'un mot dépend de son contexte, on suppose que la structure locale (H, E ou C) adoptée à une position spécifique dans une séquence protéique dépend des acides aminés environnants. Ce principe sous-tend notre approche.

Une description détaillée de l'ensemble de données est disponible dans l'Appendice. Conformément au principe selon lequel la structure locale à une position donnée dans une séquence protéique dépend des acides aminés environnants, chaque exemple comprend une fenêtre de contexte de longueur 21 et une variable cible. Cette variable cible indique la structure locale de l'acide aminé central dans la fenêtre (position 11), prenant l'une des trois valeurs : 0 pour une hélice, 1 pour un feuillet et 2 pour un enroulement. Le contexte comprend 10 positions avant et 10 positions après la position centrale, ainsi que la position centrale elle-même ($(10 + 1 + 10 = 21)$). Chaque exemple est encodé de manière à ce que la variable cible soit dans la première colonne, suivie des attributs encodés dans les colonnes restantes.

L'ensemble de données a été généré en appliquant une fenêtre glissante de longueur 21 à travers les séquences d'un ensemble de protéines. À chaque déplacement de la fenêtre d'une position, un nouvel exemple est généré, accompagné de sa cible réelle, c'est-à-dire une variable catégorielle indiquant la structure secondaire : 0 pour une α-hélice, 1 pour un β-feuillet et 2 pour un enroulement. Chaque exemple se compose de 462 attributs, qui sont des valeurs numériques comprises entre 0 et 1. Ces nombres représentent la probabilité d'observer un acide aminé spécifique à une position donnée dans la fenêtre de contexte, chaque position étant encodée avec 22 valeurs (21 positions $\times$ 22 valeurs $= 462$ attributs).

Nous fournissons trois ensembles de données : l'ensemble d'entraînement comprend 58 291 exemples, l'ensemble de validation contient 7 409 exemples, et l'ensemble de test est composé de 7 432 exemples. La **variable cible**, située dans la **première colonne**, peut prendre l'une des trois valeurs : 0, 1 ou 2. Les 462 colonnes restantes représentent les attributs, qui sont des valeurs numériques comprises entre 0 et 1. La compréhension de ce seul paragraphe est tout ce dont vous avez besoin pour réussir le devoir.

Remarque importante : Il est primordial de gérer les attentes pour ce devoir en raison de la complexité du problème de prédiction de la structure secondaire des protéines. Historiquement, même les algorithmes les plus avancés atteignaient des précisions inférieures à 70 % (Rost et Sander 1993). Les publications soulignaient souvent des améliorations d'un seul point de pourcentage. Dans nos tests utilisant ces ensembles de données, les méthodes de référence comme la régression logistique ont obtenu des précisions comprises entre 66 % et 68 %, tandis que nos réseaux de neurones ont approché 72 % [1]. Nous fournissons ces chiffres

[1] Des résultats améliorés nécessitent un ensemble de données plus large et des architectures de réseaux de neurones avancées. Comprendre l'architecture traditionnelle actuelle est essentiel pour appréhender les principes de modèles plus complexes.

pour éviter les efforts excessifs dans la recherche d'une précision exceptionnellement élevée. Votre note est basée sur la qualité de votre travail plutôt que uniquement sur les performances de vos modèles. C'est un problème intrinsèquement difficile, comme prévu pour un sujet de la portée d'un prix Nobel !

Soumission

- **Date limite :**
  - Soumettez votre notebook avant le 10 novembre à 23h.
- **Devoir individuel ou en groupe :**
  - Ce devoir peut être réalisé individuellement (groupe de 1 étudiant) ou en collaboration en binôme (groupe de 2 étudiants).
  - Un groupe doit soumettre une soumission conjointe unique.
  - Avant de soumettre, il est nécessaire de **s'enregistrer en groupe** sur Brightspace.
  - Pour permettre des changements dans la composition des groupes pour chaque devoir, une nouvelle inscription de groupe est requise pour chaque devoir.
- **Plateforme de soumission :**
  - Téléversez votre soumission sur Brightspace dans la section Devoirs (Devoir 3).
- **Format de soumission :**
  - Soumettez une copie de votre notebook sur Brightspace.

Avis important : Si le correcteur ne peut pas exécuter votre code, votre soumission recevra une note de zéro. Il est de votre responsabilité de vous assurer que votre soumission fonctionne sur un autre ordinateur que le vôtre et que toutes les cellules de votre notebook sont exécutables.

Exigences

1. Analyse exploratoire

Chargement de l'ensemble de données

Un ensemble de données a été créé pour ce devoir. Il est disponible sur un dépôt GitHub public :

- github.com/turcotte/csi4106-f24/tree/main/assignments-data/a3

Vous devez accéder à l'ensemble de données et le lire directement à partir de ce dépôt GitHub dans votre notebook Jupyter.

(1) **Charger l'ensemble de données** :

- Écrivez du code pour charger les trois ensembles de données.

Prétraitement des données

(2) **Brasser les lignes** :

- Étant donné que les exemples sont générés en faisant glisser une fenêtre sur chaque séquence de protéines, la plupart des exemples adjacents proviennent de la même protéine et partagent 20 positions. Pour atténuer l'impact potentiel négatif sur l'entraînement du modèle, la première étape consiste à brasser (*shuffle*) les **lignes** de la matrice de données.

(3) **Mise à l'échelle des caractéristiques numériques** :

- Étant donné que les 462 caractéristiques sont des proportions représentées par des valeurs comprises entre 0 et 1, la mise à l'échelle peut ne pas être nécessaire. Dans nos évaluations, l'utilisation de StandardScaler a en fait dégradé les performances du modèle. Dans votre flux de traitement, comparez les effets de ne pas mettre à l'échelle les données par rapport à l'application de MinMaxScaler. Par souci de temps, une seule expérience suffira. Il est important de noter que lorsque la mise à l'échelle est appliquée, une méthode uniforme doit être utilisée pour toutes les colonnes, compte tenu de leur nature homogène.

(4) **Séparation des cibles et des données** :

- Dans les fichiers CSV, les cibles et les données sont combinées. Pour préparer nos expériences d'apprentissage automatique, séparez les données d'entraînement X et le vecteur cible y pour chacun des trois ensembles de données.

Développement et évaluation des modèles

(5) **Développement de modèle** :

- **Modèle de base** : Implémentez un modèle utilisant le DummyClassifier. Ce modèle ignore les données d'entrée et prédit la classe majoritaire. Un tel modèle est parfois appelé modèle « homme de paille ».
- **Modèle de référence** : Comme modèle de référence, sélectionnez un des algorithmes d'apprentissage automatique précédemment étudiés : arbres de décision, k-plus proches voisins (KNN) ou régression logistique. Utilisez les paramètres par défaut fournis par scikit-learn pour entraîner chaque modèle en tant que modèle de référence. Pourquoi avez-vous choisi ce classificateur particulier ? Pourquoi pensez-vous qu'il soit approprié pour cette tâche spécifique ?
- **Modèle de réseau de neurones** : En utilisant Keras et TensorFlow[2], construisez un modèle séquentiel comprenant une couche d'entrée, une couche cachée et une couche de sortie. La couche d'entrée doit comporter 462 nœuds, correspondant aux 462 attributs de chaque exemple. La couche cachée doit comprendre 8 nœuds et utiliser la fonction d'activation par défaut. La couche de sortie doit comporter trois nœuds, correspondant aux trois classes : hélice (0), feuillet (1) et enroulement (2). Appliquez la fonction d'activation softmax à la couche de sortie pour que les sorties soient traitées comme des probabilités, avec leur somme égale à 1 pour chaque exemple d'entraînement.

[2]Keras et TensorFlow sont pré-installés dans l'environnement Google Colab. Si vous effectuez des expériences sur votre ordinateur personnel, l'installation de la dernière version de TensorFlow, la version 2.17 au moment de la rédaction, inclut également Keras.

Nous avons donc trois modèles : de base, de référence et réseau de neurones.

(6) **Évaluation des modèles** :

- Utilisez la validation croisée pour évaluer les performances du modèle de référence. Sélectionnez un petit nombre de plis (*plis*) pour éviter des temps de calcul excessives.
- **L'entraînement des réseaux de neurones peut être long.** Par conséquent, leurs performances sont généralement évaluées une seule fois en utilisant un ensemble de validation. Assurez-vous de ne pas utiliser l'ensemble de test avant la fin du devoir.
- Évaluez les modèles en utilisant des métriques telles que la précision, le rappel et le score F1.

Optimisation des hyperparamètres

(7) **Modèle de référence :**

- Pour assurer une comparaison équitable avec notre modèle de référence, nous examinerons comment la variation des hyperparamètres affecte ses performances. Cela évite la conclusion erronée selon laquelle les réseaux de neurones sont intrinsèquement meilleurs, alors qu'en réalité, un ajustement adéquat des hyperparamètres pourrait améliorer les performances du modèle de référence.
- Concentrez-vous sur les hyperparamètres suivants pour chaque modèle :
  - DecisionTreeClassifier : `criterion` et `max_depth`.
  - LogisticRegression : `penalty`, `max_iter`, et `tol`.
  - KNeighborsClassifier : `n_neighbors` et `weights`.
- Utilisez une stratégie de recherche par grille ou les méthodes intégrées de scikit-learn GridSearchCV pour évaluer de manière exhaustive toutes les combinaisons de valeurs d'hyperparamètres. La validation croisée doit être utilisée pour évaluer chaque combinaison.
- Quantifiez les performances de chaque configuration d'hyperparamètres en utilisant des métriques telles que la précision, le rappel et le score F1.
- Analysez les résultats et fournissez des aperçus sur quelles configurations d'hyperparamètres ont obtenu des performances optimales pour chaque modèle.

(8) **Réseau de neurones :**

Lors de notre exploration et ajustement des réseaux de neurones, nous nous concentrons sur les hyperparamètres suivants :

- **Une seule couche cachée, en variant le nombre de nœuds**.
  - Commencez avec un seul nœud dans la couche cachée. Utilisez un graphique pour représenter l'évolution de la perte et de la précision pour les ensembles d'entraînement et de validation, avec l'axe horizontal représentant le nombre d'époques d'entraînement et l'axe vertical représentant la perte et la précision. L'entraînement de ce réseau devrait être relativement rapide, nous allons donc procéder à un entraînement sur 50 époques. Que concluez-vous de l'observation du graphique ? Le réseau sous-apprend-il ou surapprend-il ? Pourquoi ?

- Répétez le processus ci-dessus en utilisant 2 et 4 nœuds dans la couche cachée. Utilisez le même type de graphique pour documenter vos observations concernant la perte et la précision [3].
- Commencez avec 8 nœuds dans la couche cachée et doublez progressivement le nombre de nœuds jusqu'à ce qu'il dépasse le nombre de nœuds dans la couche d'entrée. Cela donne lieu à sept expériences et graphiques correspondants pour les configurations suivantes : 8, 16, 32, 64, 128, 256 et 512 nœuds. Documentez vos observations tout au long du processus.
- Assurez-vous que le **nombre d'époques d'entraînement** est suffisant pour **observer une augmentation de la perte de validation**. **Conseil** : Lors du développement du modèle, commencez avec un petit nombre d'époques, comme 5 ou 10. Une fois que le modèle semble bien fonctionner, testez avec des valeurs plus importantes, comme 40 ou 80 époques, ce qui s'est avéré raisonnable dans nos tests. En fonction de vos observations, envisagez de mener des expériences supplémentaires, si nécessaire. Combien d'époques ont finalement été nécessaires ?

- **Variation du nombre de couches**.
  - Réalisez des expériences similaires à celles décrites ci-dessus, mais cette fois en faisant varier le nombre de couches de 1 à 4. Documentez vos résultats.
  - Combien de nœuds chaque couche devrait-elle contenir ? Testez au moins deux scénarios. Traditionnellement, une stratégie courante consistait à diminuer le nombre de nœuds de la couche d'entrée à la couche de sortie, souvent en divisant par deux, pour créer une structure en pyramide. Cependant, l'expérience récente suggère que le maintien d'un nombre constant de nœuds dans toutes les couches peut également bien fonctionner. Décrivez vos observations. Il est acceptable que les deux stratégies produisent des résultats similaires en termes de performance.
  - Sélectionnez un de vos modèles qui illustre le surapprentissage. Dans nos expériences, nous avons facilement construit un modèle atteignant près de 100 % de précision sur les données d'entraînement, sans aucune amélioration similaire sur l'ensemble de validation. Présentez ce réseau de neurones avec ses graphiques de précision et de perte. Expliquez pourquoi vous concluez que le modèle surapprend.

- **Fonction d'activation**.
  - Présentez les résultats pour une des configurations mentionnées ci-dessus en variant la fonction d'activation. Testez au moins `relu` (le paramètre par défaut) et `sigmoid`. Le choix du modèle spécifique, y compris le nombre de couches et de nœuds, est à votre discrétion. Documentez vos observations en conséquence.

- **Régularisation** dans les réseaux de neurones est une technique utilisée pour éviter le surapprentissage.
  - Une technique consiste à ajouter une pénalité à la fonction de perte pour décourager les modèles excessivement complexes. Appliquez une pénalité `l2` à certaines ou à toutes les couches. Soyez prudent, car des pénalités trop agressives se sont révélées problématiques dans nos expériences. Commencez avec la valeur par défaut `l2` de 0.01, puis réduisez-la à 0.001 et à 1e-4. Sélectionnez un modèle spécifique parmi les expériences ci-dessus et

[3] Dans ce devoir, chaque fois que vous êtes invité à décrire ou à documenter vos observations, nous vous demandons de générer un graphique qui affiche la perte et la précision pour les ensembles d'entraînement et de validation en fonction du nombre d'époques. Utilisez ce graphique pour étayer votre explication.

présentez un cas où vous avez réussi à réduire le surapprentissage. Incluez une paire de graphiques comparant les résultats avec et sans régularisation. Expliquez votre raisonnement pour conclure que le surapprentissage a été réduit. N'espérez pas éliminer complètement le surapprentissage. Encore une fois, il s'agit d'un ensemble de données difficile à travailler.

- Les couches de **dropout** sont une technique de régularisation dans les réseaux de neurones, où un sous-ensemble aléatoire de neurones est temporairement retiré pendant l'entraînement. Cela aide à éviter le surapprentissage en favorisant la redondance et en améliorant la capacité du réseau à généraliser sur de nouvelles données. Sélectionnez un modèle spécifique parmi les expériences ci-dessus, où vous avez plusieurs couches, et expérimentez l'ajout d'une ou de plusieurs couches de dropout dans votre réseau. Testez deux taux différents, par exemple 0.25 et 0.5. Documentez vos observations.
- Résumez vos expériences en utilisant une représentation graphique telle que la Figure 6.15 sur cette page.
- L'**arrêt anticipé** est une technique de régularisation lors de l'entraînement d'un réseau de neurones, où le processus est interrompu lorsque les performances sur l'ensemble de validation commencent à se dégrader, ce qui empêche le réseau d'apprendre le bruit dans les données d'entraînement. Parmi toutes les expériences menées jusqu'à présent, choisissez **une** configuration (le nombre de couches, le nombre de nœuds, la fonction d'activation, la pénalité L2 et les couches de dropout) qui a donné les meilleures performances. Utilisez un graphique de perte et de précision pour déterminer le nombre optimal d'itérations d'entraînement pour ce réseau. Quel est le nombre optimal d'époques pour cette configuration de réseau et pourquoi ?

Test

(9) **Comparaison des modèles** :

- Évaluez le modèle de référence sur l'ensemble de test, en utilisant l'ensemble de paramètres optimal identifié lors de la recherche de grille. Appliquez également la meilleure configuration de réseau de neurones à l'ensemble de test.
- Quantifiez les performances du modèle de référence (meilleure configuration d'hyperparamètres) et de votre réseau de neurones (meilleure configuration) en utilisant des métriques telles que la précision, le rappel et le score F1. Comment ces deux modèles se comparent-ils au modèle de base ?
- Fournissez des recommandations sur le(s) modèle(s) à choisir pour cette tâche et justifiez vos choix en fonction des résultats de l'analyse.

2. Documentation de l'analyse exploratoire

Le rapport doit documenter de manière exhaustive l'ensemble du processus suivi au cours de ce devoir. Lors de la correction du Devoir 1, nous avons constaté que certains étudiants ont soumis du code sans explications ni clarifications adéquates. Sachez qu'un manque de commentaires entraînera des déductions de points. Utilisez efficacement la structure des cahiers Jupyter en divisant les programmes longs en plusieurs cellules de code, entrecoupées d'explications claires dans des cellules en format *markdown*. Le cahier Jupyter doit inclure les éléments suivants :

- Votre(vos) nom(s), numéro(s) d'étudiant et un titre de rapport.

- Si vous avez travaillé en binôme pour ce devoir, expliquez comment les tâches ont été réparties entre les membres. Comment avez-vous fait en sorte que les deux étudiants atteignent les objectifs d'apprentissage ?
- Une section pour chaque étape de l'analyse exploratoire, contenant le code Python pertinent et des explications ou résultats.
  - Pour les sections nécessitant du code Python, incluez le code dans une cellule.
  - Pour les sections nécessitant des explications ou des résultats, incluez-les dans une cellule séparée ou en combinaison avec les cellules de code.
- Assurez-vous de séparer logiquement le code en différentes cellules. Par exemple, la définition d'une fonction doit être dans une cellule et son exécution dans une autre. Évitez de placer trop de code dans une seule cellule pour maintenir la clarté et la lisibilité.
- Le cahier que vous soumettez doit inclure les résultats de l'exécution, accompagnés de graphiques, afin que l'assistant pédagogique puisse évaluer le cahier sans avoir besoin d'exécuter le code.

Évaluation

- **Effort global dans le rapport (5%)**
- **Chargement et prétraitement (5%)**
- **Développement du modèle (5%)**
- **Évaluation du modèle (10%)**
- **Optimisation des paramètres pour le modèle de référence (10%)**
- **Optimisation des paramètres pour le modèle de réseau de neurones (50%)**
- **Analyse des résultats (10%)**
- **Ressources et références (5%)**

Ressources

Comme mentionné précédemment, assurez-vous de citer toutes les parties de votre code dérivées de sites Web, manuels ou autres ressources externes. Même si vous étiez déjà familier avec les concepts, veillez à citer les bibliothèques que vous avez utilisées.

Actuellement, de nombreux programmeurs utilisent l'intelligence artificielle pour améliorer leur productivité, une tendance qui continuera probablement à croître. Pour mieux vous préparer au marché du travail, il est plausible d'utiliser ces technologies. Cependant, il est impératif que vous compreniez pleinement les concepts sur lesquels vous êtes évalué, car ces outils ne seront pas disponibles lors des évaluations en personne.

Si vous utilisez l'assistance IA, documentez en détail vos interactions. Incluez les outils et leurs versions dans votre rapport, ainsi qu'une transcription de toutes les interactions. La plupart des assistants IA conservent un historique de vos conversations. La pratique recommandée est de créer une nouvelle conversation spécifiquement pour le devoir et de réutiliser systématiquement cette conversation tout au long de votre travail sur le devoir. Assurez-vous que cette conversation est uniquement dédiée au devoir. Soumettez cette transcription dans la section des références de votre notebook Jupyter.

Questions

- Vous pouvez poser vos questions dans le forum de discussion du devoir sur Brightspace.
- Alternativement, vous pouvez envoyer un e-mail à l'un des quatre assistants pédagogiques. Cependant, l'utilisation du forum est fortement préférée, car cela permet à vos camarades de classe de bénéficier des questions et des réponses correspondantes fournies par les assistants pédagogiques.

Description de l'ensemble de données

Ce qui suit est un aperçu du processus utilisé pour produire l'ensemble de données. Cet ensemble de données est dérivé de l'ensemble de données largement utilisé CB513 (Cuff et Barton 1999). Comme son nom l'indique, CB513 est constitué de 513 séquences de protéines, dont les longueurs varient de 21 à 755 acides aminés.

Conformément au principe selon lequel la structure locale à une position donnée dans une séquence protéique dépend des acides aminés environnants, chaque exemple comprend une fenêtre de contexte de longueur 21 et une variable cible. Cette variable cible indique la structure locale de l'acide aminé central dans la fenêtre (position 11), prenant l'une des trois valeurs : 0 pour hélice, 1 pour feuillet et 2 pour enroulement. Le contexte comprend 10 positions avant et 10 positions après la position centrale, ainsi que la position centrale elle-même ($(10 + 1 + 10 = 21)$). Chaque exemple est encodé de manière à ce que la variable cible soit dans la première colonne, suivie des attributs encodés dans les colonnes restantes.

Nous aurions pu utiliser l'encodage *one-hot*. Chaque acide aminé aurait été transformé en un vecteur unique où un élément est « 1 » (chaud) et les autres sont « 0 » (froid).

```
import numpy as np
from sklearn.preprocessing import OneHotEncoder

# Code à une lettre pour les 20 acides aminés

aa = ['A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y']

onehot_encoder = OneHotEncoder(categories=[aa], sparse_output=False)

# input_sequence a été défini dans l'introduction

X = np.array(list(input_sequence)).reshape(-1, 1)

X_encoded = onehot_encoder.fit_transform(X)

# Les cinq premiers acides aminés encodés de input_sequence

print(X_encoded[:5])
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

Cependant, nous pouvons tirer parti du fait que CB513 comprend des séquences évolutivement liées pour chacune des 513 séquences d'entrée, qui sont connues pour adopter la même structure. Par conséquent, chaque position dans la fenêtre de contexte est représentée par un vecteur de longueur 22. Ce vecteur tient compte des 20 acides aminés naturels, d'un symbole pour les inconnus et d'un autre pour les insertions. Notre représentation prend en charge les inconnus ou les valeurs manquantes, une procédure standard en bio-informatique qui n'affectera pas négativement vos résultats. Le symbole d'insertion indique que certaines protéines évolutivement liées peuvent avoir subi des suppressions. Chaque position dans ce vecteur de longueur 22 est un nombre réel représentant la proportion d'un acide aminé spécifique à cette position. Par exemple, si la première position correspond à 'A' et que sa valeur est de 0,2, cela indique que 20 % des protéines liées ont un 'A' à cette position. Cette approche d'encodage est connue pour donner de meilleurs résultats. Le segment de 1m 12s à 1m 30s dans la vidéo What Is AlphaFold? illustre ce processus. Spécifiquement, notre ensemble de données est basé sur la représentation MSA introduite à 1m 30s dans la vidéo.

Pour éviter les fuites de données, trois ensembles de données distincts ont été construits au niveau des protéines. Étant donné que les exemples sont générés en faisant glisser une fenêtre sur chaque séquence protéique, plusieurs exemples proviennent de la même séquence. Pour atténuer les fuites, les protéines ont été divisées en trois groupes — entraînement, validation et test — en s'assurant que le nombre total d'acides aminés dans chaque groupe correspond à un ratio prédéterminé du nombre total d'acides aminés.

Vidéos

- Protein folding explained par Google DeepMind. Publiée le 30 novembre 2020. (1m 51s)
- What Is AlphaFold? par le New England Journal of Medicine (NEJM) Group. Publiée le 21 septembre 2023. (5m 14s)
- AlphaFold: The making of a scientific breakthrough par Google DeepMind. Publiée le 30 novembre 2020. (7m 54s)

Références

Cuff, James A., et Geoffrey J. Barton. 1999. « Evaluation and improvement of multiple sequence methods for protein secondary structure prediction ». *Proteins: Structure, Function, and Bioinformatics* 34 (4): 508-19.

Rost, B., et C. Sander. 1993. « Improved prediction of protein secondary structure by use of sequence profiles and neural networks ». *Proc. Natl. Acad. Sci. U.S.A.* 90 (16): 7558-62.