Guide Programmeur – Détection de pics RMN 1D/2D avec CNN (CNN_learn.R)

Ce guide décrit le pipeline complet de génération de spectres synthétiques, entraînement d'un CNN 1D pour la détection de pics (par lignes et par colonnes), application sur spectres RMN 2D réels et post-traitement.

1. Génération de spectres synthétiques

generate_spectrum_labels_full_mixed()

But : Générer un dataset de spectres 1D synthétiques et leurs labels (0,1 ou 2 selon présence d'un pic faible, fort, ou non) pour entraîner un CNN.

Entrées :

- n_spectra (int) : nombre de spectres à générer.
- n_points (int) : nombre de points par spectre (par défaut 2048).
- intensity_threshold (float) : seuil pour définir un pic "fort".
- center_margin (float) : fraction du spectre considérée comme centre (pour classer les pics).
- seed (int) : graine pour la reproductibilité.
- mix_ratio (float) : probabilité d'utiliser un spectre type TOCSY ou un spectre type random.

Processus :

1. Pour chaque spectre, choisir aléatoirement une génération de type TOCSY ou random basée sur la base de données de peak forest.
2. Générer le spectre 1D avec une probabilité de mix-ratio via generate_dense_tocsy_1D_cut() ou generate_random_spectrum_with_peaks().
3. Interpoler le spectre sur une grille standard [14, 0] ppm.
4. Créer les labels :
    - y_class : vecteur 0 = fond, 1 = pic fort au centre, 2 = pic faible ou décentré.

    - y_reg : matrice [n_points,3] avec [ppm, intensité, FWHH_est].

Sortie :
Liste R contenant :

- X : [n_spectra, n_points] : spectres interpolés.
- y_class : [n_spectra, n_points] : labels de classification (0/1/2).
- y_reg : [n_spectra, n_points, 3] : labels de régression (ppm, intensité, FWHH) par spectre n_spectra et pour chaque point n_points.

plot_four_spectra()

But : Visualiser 4 spectres 1D avec leurs pics labellisés.
Entrées :

- X : matrice [n_spectra, n_points].
- y_class : labels [n_spectra, n_points].
- ppm : vecteur [n_points] indiquant l'axe chimique.
- indices : vecteurs des spectres à afficher.

Sortie :

- Aucun objet retourné, affiche simplement les graphiques 1D.
- Couleur rouge = pic fort, bleu = pic faible (pas forcément extrêmement important pour l'appentissage, implémentation au début du développement du réseau)

2. Réseau de neurones pour la détection de pics sur extraction colonnes/lignes 1D

focal_loss(gamma, alpha)

But : Fonction de perte spécifique pour la classification multi-classes (0/1/2) qui vient accentuer l'erreur sur la détection de pics par rapport à la détection de fond.
Entrées :

- gamma (float) : paramètre de focal loss.

- alpha (float) : pondération des classes.

Sortie :

- Fonction R compatible avec Keras.
- Entrées : y_true [batch_size, n_points], y_pred [batch_size, n_points, 3].
- Retour : [batch_size, n_points] : perte par point.

build_peak_predictor()

But : Construire un CNN 1D pour prédiction simultanée classification et régression.

Architecture :

- Input : [2048,1].
- Plusieurs convolutions 1D avec ReLU.
- Tête classification : [2048,3] avec softmax.
- Tête régression : [2048,3] linéaire (ppm, intensité, FWHH).

Sortie :

- Modèle Keras compilé, prêt à l'entraînement.

Exemple d'entraînement :

```
model %>% fit(
  x = X_train,
  y = list(class_output = y_class_train, reg_output = y_reg_train),
  epochs = 800, batch_size = 32, validation_split = 0.2
)
```

3. Préparation et normalisation des données réelles Bruker

- read_bruker(dir, dim) : lit un spectre Bruker 1D/2D (même technique que celle de Julien).

- Normalisation 2D (ATTENTION : le log n'est utile que dans le cas de spectres extrêmement bruités: si la prédiction ne permet pas de détecter des pics, appliquer rr_log à la place de rr_norm) :

```
rr_abs <- abs(rr)
rr_norm <- (rr_abs - min(rr_abs)) / (max(rr_abs) - min(rr_abs))
rr_log <- log1p(rr_norm * 100) / max(log1p(rr_norm * 100))
```

- Axe F1/F2 en ppm : rownames(rr) et colnames(rr).

4. Détection de pics sur spectre 2D

pad_sequence(x, target_length)

- Pad (=complète, remplit) un vecteur x avec des zéros pour atteindre target_length.
- Sortie : vecteur [target_length].

get_detected_peaks_with_intensity(rr_norm, model, target_length)

But : Appliquer le CNN sur chaque ligne et colonne pour détecter pics.
Sortie : un DataFrame de dimension [n_peaks,3] :

- F2 : ligne du spectre.
- F1 : colonne du spectre.
- Intensity : intensité prédite par le CNN.

predict_peaks_1D_batch(spectrum_mat, model, axis, threshold_class, batch_size, target_length)

- Version optimisée par batch, pour détecter plus vite. Cette méthode de prédiction est l'unique conservée après de multiples tests.
- Entrées :
    - spectrum_mat : [n_rows, n_cols] matrice normalisée.
    - axis : "rows" ou "columns".
    - threshold_class : seuil de probabilité pour classer un pic.
- Sortie : DataFrame [n_detected_peaks,3] : F1, F2, Intensity.

5. Nettoyage et filtrage des pics

filter_peaks_by_proportion(peaks_clean, threshold, intensity_threshold)

- Filtre les colonnes/lignes trop denses comme celles correspondant à de grosses traces de bruit/ d'eau.
- Retour : liste avec
    - filtered_peaks : [n_peaks,3] DataFrame filtré.
    - removed_columns, removed_rows : vecteurs des indices supprimés.

filter_noisy_columns(peaks_df, threshold_ratio, min_col_max)

- Supprime les pics faibles par rapport à la colonne max: on garde en général les valeurs supérieures à 90% du max de la colonne/ligne bruitée..
- Sortie : DataFrame [n_peaks,3].

clean_peak_clusters_dbscan(peaks_df, ppm_x, ppm_y, eps_ppm, minPts)

- Applique DBSCAN sur les coordonnées ppm pour regrouper les clusters.
- Sortie : DataFrame [n_peaks,4] avec colonne cluster ajoutée.

remove_peaks_ppm_range(peaks, rr_norm, axis, ppm_min, ppm_max)

- Supprime les pics dans une plage ppm donnée (méthode dite "agressive" pour supprimer totalement une plage de ppm, à n'appliquer qu'en cas de zone vraiment "hors pics" .
- Retour : DataFrame filtré [n_peaks,3].

6. Visualisation interactive

plot_peaks_ppm_plotly_clean(peaks, rr_norm, intensity_threshold, ratio)

- Scatter plot 2D interactif (plotly).
- Entrées : peaks [n_peaks,3], rr_norm [n_rows,n_cols].
- Sortie : objet plotly, que l'on peut survoler pour voir les indices ppm et intensités.

plot_random_1D_grid(rr_mat, type, n_extract)

- Affiche 4 spectres aléatoires (2x2) en lignes ou colonnes.

- Entrée : rr_mat [n_rows, n_cols].
- Sortie : plotly subplot.

Contours et clustering

- Downsampling : downsample_matrix et downsample_axis.
- DBSCAN sur peaks normalisés : ajoute cluster_db.
- Bounding boxes par cluster : xmin, xmax, ymin, ymax, cx, cy, intensity.
- Conversion indices ⇒ ppm : xmin_ppm, xmax_ppm, ymin_ppm, ymax_ppm, cx_ppm, cy_ppm.
- Visualisation plotly avec rectangles et centres.

7. Fusion et statistiques (utile pour comparer les deux méthodes)

- Fusion CSV TOCSY et bounding boxes : bind_rows ⇒ merged_points_named_201.csv.
- Clustering euclidien par proximité (tol=0.1 ppm) pour résumé : summary_points.
- Tri et sauvegarde du fichier : merged_points_sorted_clean_201.csv. Cela comprend les centres des bounding boxes.

8. Fonction Shiny à intégrer à l'application

run_cnn_peak_picking(rr_norm, model, params, threshold_class, batch_size)

- Pipeline complet 2D :
    - Prédiction 1D sur lignes et colonnes.
    - Filtrage de proportion et colonnes bruitées.
    - DBSCAN clustering + bounding boxes.
- Sortie : liste avec
    - peaks : DataFrame [n_peaks,6] pics filtrés avec clusters:
        - :F1_ppm : position en ppm sur F1

        - F2_ppm : position en ppm sur F2
        - Intensity : intensité du pic
        - cluster_db : numéro de cluster DBSCAN
        - xmin_ppm : début du cluster en ppm
        - xmax_ppm : fin du cluster en ppm
    - boxes : bounding boxes [n_clusters,8] en ppm:
        - xmin_ppm : bord gauche de la boîte
        - xmax_ppm : bord droit
        - ymin_ppm : bord bas
        - ymax_ppm : bord haut
        - cx_ppm : centre x
        - cy_ppm : centre y
        - mean_intensity : intensité moyenne dans la boîte
        - cluster_id : identifiant du cluster