Optimisation des Processus à Haute Dimensionnalité

à l'aide des Réseaux de Neurones, du Contrôle Statistique des Processus et de la Conception d'Expériences

Auteur : Salmane Koraichi (GINF2 ENSA TANGER)

3 novembre 2024

Résumé

Dans ce document, nous présentons une approche complète pour l'optimisation d'un processus complexe et de haute dimensionnalité en utilisant les réseaux de neurones. Nous intégrons les principes du Contrôle Statistique des Processus (CSP) et de la Conception d'Expériences (CE) pour gérer les interactions étendues entre les variables, minimiser la variance et contrôler les facteurs influents. L'objectif est d'identifier les valeurs les plus efficaces pour chaque paramètre afin d'optimiser la sortie y, en assurant une solution bien optimisée et stable dans toutes les dimensions. Nous fournissons des explications détaillées, des formulations mathématiques et des implémentations de code Python pour guider chaque étape du processus.

Table des matières

1 Introduction

Dans l'industrie moderne, l'optimisation de systèmes complexes avec de multiples variables en interaction est cruciale pour l'efficacité et la fiabilité. Les données de haute dimensionnalité et les interactions complexes entre variables posent des défis computationnels significatifs. Ce document vise à aborder ces défis en utilisant les réseaux de neurones, le Contrôle Statistique des Processus (CSP) et la Conception d'Expériences (CE) pour optimiser une équation complexe de la forme :

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \cdots + \epsilon \tag{1}$$

Notre objectif est de développer un modèle de réseau de neurones capable de capturer les relations complexes dans des données de haute dimensionnalité, nous permettant de trouver les valeurs optimales des paramètres qui minimisent ou maximisent la sortie y.

2 Définition du Problème

2.1 Objectif

— Optimiser un processus complexe représenté par une équation de haute dimensionnalité avec des variables en interaction.
— Identifier les valeurs les plus efficaces pour chaque paramètre afin d'optimiser la sortie y.
— Assurer la stabilité et la fiabilité dans toutes les dimensions du processus.

2.2 Défis

1. **Haute Dimensionnalité** : Gérer de nombreuses variables et leurs interactions augmente la complexité computationnelle.
2. **Interactions Non Linéaires** : Les variables peuvent interagir de manière non linéaire, ce qui complique le processus de modélisation.
3. **Efficacité Computationnelle** : L'entraînement de réseaux de neurones sur des données de haute dimensionnalité nécessite des ressources computationnelles importantes.
4. **Variabilité du Processus** : Contrôler et réduire la variabilité est essentiel pour l'optimisation du processus.

3 Formulation Mathématique

3.1 Équation du Processus

Nous considérons un processus complexe modélisé par une équation incluant :

— **Termes Linéaires** : Effets individuels des variables d'entrée.

— **Termes d'Interaction** : Effets combinés entre les paires de variables.

— **Termes Non Linéaires** : Effets quadratiques ou d'ordre supérieur des variables.

— **Bruit Aléatoire** : Représente la variabilité du processus et les erreurs de mesure.

Mathématiquement, le processus peut être représenté comme :

$$y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \beta_{ij} x_i x_j + \sum_{i=1}^{n} \beta_{ii} x_i^2 + \epsilon \tag{2}$$

Où :

— y est la variable de sortie.

— x_i sont les variables d'entrée.

— β_0 est le terme d'interception.

— β_i sont les coefficients des termes linéaires.

— β_{ij} sont les coefficients des termes d'interaction.

— β_{ii} sont les coefficients des termes non linéaires (quadratiques).

— ϵ est le terme de bruit aléatoire.

4 Intégration des Concepts du CSP et de la CE

4.1 Contrôle Statistique des Processus (CSP)

Objectif : Surveiller et contrôler le processus pour assurer qu'il fonctionne à son plein potentiel.

Application :

— Utiliser des cartes de contrôle pour surveiller les prédictions du modèle et les résidus.

— Analyser la variabilité pour identifier les causes assignables.

— Mettre en œuvre une analyse de la capacité du processus pour évaluer les performances.

4.2 Conception d'Expériences (CE)

Objectif : Planifier systématiquement des expériences pour comprendre les effets des variables sur la sortie.

Application :

— Utiliser des plans factoriels pour étudier les interactions entre les variables.

— Randomiser la génération des données pour réduire l'erreur expérimentale.

— Appliquer la méthodologie de surface de réponse pour l'optimisation.

5 Génération des Données

5.1 Définition des Paramètres

Nous simulons un ensemble de données avec les caractéristiques suivantes :

— **Nombre d'Échantillons** ($n_{\text{échantillons}}$) : 10 000

— **Nombre de Variables** ($n_{\text{variables}}$) : 10

5.2 Génération des Variables d'Entrée

Nous générons des variables d'entrée aléatoires X uniformément distribuées entre 0 et 10 :

```
import numpy as np

# Fixer la graine al atoire pour la reproductibilit
np.random.seed(42)

# Param tres
n_samples = 10000
n_features = 10

# G n rer les variables d'entr e al atoires entre 0 et 10
X = np.random.uniform(0, 10, size=(n_samples, n_features))
```

5.3 Assignation des Coefficients

Nous attribuons des coefficients aléatoires pour les termes linéaires, d'interaction et quadratiques :

```
# Coefficients pour les termes lin aires
beta_linear = np.random.uniform(-5, 5, size=n_features)

# Nombre de termes d'interaction uniques
n_interactions = int(n_features * (n_features - 1) / 2)

# Coefficients pour les termes d'interaction
beta_interaction = np.random.uniform(-1, 1, size=n_interactions)

# Coefficients pour les termes quadratiques
beta_quadratic = np.random.uniform(-0.5, 0.5, size=n_features)
```

5.4 Calcul des Termes Linéaires, d'Interaction et Quadratiques

Termes Linéaires :

$$\text{termes\_linéaires} = X \cdot \beta_{\text{linéaire}} \tag{3}$$

```
# Calcul des termes lin aires
termes_lineaires = np.dot(X, beta_linear)
```

Termes d'Interaction :

$$\text{termes\_interaction} = \sum_{i=1}^{n_{\text{variables}}} \sum_{j=i+1}^{n_{\text{variables}}} \beta_{ij} x_i x_j \tag{4}$$

```
# Calcul des termes d'interaction
termes_interaction = np.zeros(n_samples)
index = 0

for i in range(n_features):
    for j in range(i + 1, n_features):
        termes_interaction += beta_interaction[index] * X[:, i] * X[:, j]
        index += 1
```

Termes Quadratiques :

$$\text{termes\_quadratiques} = \sum_{i=1}^{n_{\text{variables}}} \beta_{ii} x_i^2 \tag{5}$$

```
# Calcul des termes quadratiques
termes_quadratiques = np.sum(beta_quadratic * X**2, axis=1)
```

5.5 Ajout du Bruit Aléatoire

```
# Bruit al atoire pour simuler la variabilit  du processus
epsilon = np.random.normal(0, 1, n_samples)
```

5.6 Calcul de la Variable de Sortie

$$y = \beta_0 + \text{termes\_linéaires} + \text{termes\_interaction} + \text{termes\_quadratiques} + \epsilon \tag{6}$$

```
# Calcul de la variable de sortie y
beta_0 = np.random.uniform(-10, 10)  # Terme d'interception
y = beta_0 + termes_lineaires + termes_interaction + termes_quadratiques + epsilon
```

5.7 Création du DataFrame

```
import pandas as pd

# Noms des variables
colonnes = [f'x{i+1}' for i in range(n_features)]

# Combiner les variables et la cible dans un DataFrame
data = pd.DataFrame(X, columns=colonnes)
data['y'] = y
```

6 Prétraitement des Données

6.1 Mise à l'Échelle des Variables

La mise à l'échelle des variables améliore la convergence du modèle :

```
from sklearn.preprocessing import StandardScaler

# Initialiser le scaler
scaler = StandardScaler()

# Ajuster et transformer les variables
X_scaled = scaler.fit_transform(data.drop('y', axis=1))

# Mettre    jour le DataFrame avec les variables mises    l' chelle
data_scaled = pd.DataFrame(X_scaled, columns=colonnes)
data_scaled['y'] = data['y']
```

6.2 Vérification des Données Scaled

```
# Afficher les premi res lignes
print(data_scaled.head())
```

7 Analyse Exploratoire des Données

Comprendre les caractéristiques des données est crucial.

7.1 Matrice de Corrélation

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculer la matrice de corr lation
matrice_corr = data_scaled.corr()

# Tracer la heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(matrice_corr, annot=False, cmap='coolwarm')
plt.title('Matrice de Corr lation')
plt.show()
```

Explication :

— **But** : Identifier les relations entre les variables.

— **Action** : Observer quelles variables sont fortement corrélées avec la cible.

8 Ingénierie des Caractéristiques

8.1 Création des Variables d'Interaction

En utilisant les principes de la CE pour inclure des termes d'interaction :

```
from sklearn.preprocessing import PolynomialFeatures

# Cr er des variables polynomiales (degr =2 inclut les interactions)
poly = PolynomialFeatures(degree=2, include_bias=False)

# Ajuster et transformer les variables
X_poly = poly.fit_transform(data_scaled.drop('y', axis=1))

# Obtenir les noms des variables
noms_variables = poly.get_feature_names_out(colonnes)
```

Explication :

— **PolynomialFeatures** :

— `degree=2` : Inclut toutes les combinaisons de variables jusqu'au degré 2.

— `include_bias=False` : Exclut le terme biais (interception).

8.2 Mise à Jour du DataFrame avec les Variables d'Interaction

```
# Cr er un DataFrame avec les variables d'interaction
data_poly = pd.DataFrame(X_poly, columns=noms_variables)
data_poly['y'] = data_scaled['y']
```

9 Implémentation du Modèle de Réseau de Neurones

9.1 Séparation des Données

```
from sklearn.model_selection import train_test_split

# Variables et cible
X = data_poly.drop('y', axis=1).values
y = data_poly['y'].values

# S parer les donn es
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

9.2 Importation des Bibliothèques TensorFlow

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

9.3 Définition de l'Architecture du Réseau de Neurones

```
# Dimension d'entr e apr s l'ajout des termes d'interaction
input_dim = X_train.shape[1]

# Initialiser le mod le
model = Sequential()

# Couche d'entr e
model.add(Dense(256, input_dim=input_dim, activation='relu'))

# Couches cach es
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))

# Couche de sortie
model.add(Dense(1, activation='linear'))
```

Explication :

— **Structure du Modèle** :

— **Couches Denses** : Capturent les relations complexes.

— **Fonction d'Activation** : `relu` introduit la non-linéarité.

- **Couches Dropout** : Préviennent le surapprentissage en désactivant aléatoirement des neurones pendant l'entraînement.

— **Couche de Sortie** :

- **Fonction d'Activation** : `linear` pour les tâches de régression.

9.4 Compilation du Modèle

```
# Compiler le mod le
model.compile(
    optimizer='adam',
    loss='mean_squared_error',
    metrics=['mean_absolute_error']
)
```

Explication :

— **Optimiseur** : `adam` ajuste les taux d'apprentissage pendant l'entraînement.

— **Fonction de Perte** : `mean_squared_error` pour la régression.

— **Métriques** : `mean_absolute_error` fournit une interprétation directe.

10 Entraînement du Modèle

10.1 Mise en Place de l'Early Stopping

```
from tensorflow.keras.callbacks import EarlyStopping

# Early stopping pour pr venir le surapprentissage
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```

10.2 Entraînement du Modèle

```
# Entra ner le mod le
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=200,
    batch_size=64,
    callbacks=[early_stop],
    verbose=1
)
```

Explication :

- **Validation Split** : 20% des données d'entraînement utilisées pour la validation.
- **Époques** : Maximum de 200 époques ; l'early stopping arrêtera probablement l'entraînement plus tôt.
- **Batch Size** : 64 pour un équilibre entre la vitesse d'entraînement et la stabilité.
- **Callbacks** : L'early stopping prévient le surapprentissage en surveillant la perte de validation.

10.3 Visualisation de l'Historique d'Entraînement

```
# Tracer la perte au fil des  poques
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Perte d\'Entra nement')
plt.plot(history.history['val_loss'], label='Perte de Validation')
plt.legend()
plt.xlabel(' poques ')
plt.ylabel('Perte')
plt.title('Historique d\'Entra nement')
plt.show()
```

11 Évaluation du Modèle

11.1 Évaluation sur les Données de Test

```
#  valuer  le mod le sur les donn es de test
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
print(f'Perte MSE sur le Test : {test_loss:.4f}')
print(f'Erreur Absolue Moyenne sur le Test : {test_mae:.4f}')
```

11.2 Génération des Prédictions

```
# Pr dire sur les donn es de test
y_pred = model.predict(X_test)
```

11.3 Calcul des Métriques de Performance

```
from sklearn.metrics import r2_score, mean_squared_error

# Erreur Quadratique Moyenne
mse = mean_squared_error(y_test, y_pred)

# Score R
r2 = r2_score(y_test, y_pred)

print(f'Erreur Quadratique Moyenne : {mse:.4f}')
print(f'Score R  : {r2:.4f}')
```

Explication :

- **Erreur Quadratique Moyenne (MSE)** : Mesure la moyenne des carrés des écarts entre les prédictions et les valeurs réelles.
- **Score R²** : Proportion de la variance expliquée par le modèle.

11.4 Visualisation des Valeurs Prédites vs Réelles

```
# Graphique de dispersion des valeurs r elles vs pr dites
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.xlabel('Valeurs R elles')
plt.ylabel('Valeurs Pr dites')
plt.title('Valeurs R elles vs Pr dites')

# Ligne indiquant les pr dictions parfaites
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.show()
```

12 Application du Contrôle Statistique des Processus

12.1 Analyse des Résidus

```
# Calcul des r sidus
residus = y_test - y_pred.flatten()

# Tracer l'histogramme des r sidus
plt.figure(figsize=(8, 6))
sns.histplot(residus, kde=True)
plt.title('Distribution des R sidus')
plt.xlabel('R sidu')
plt.ylabel('Fr quence')
plt.show()
```

12.2 Carte de Contrôle des Résidus

```
# Param tres de la carte de contr le
moyenne_residus = np.mean(residus)
ecart_type_residus = np.std(residus)
limite_superieure = moyenne_residus + 3 * ecart_type_residus
limite_inferieure = moyenne_residus - 3 * ecart_type_residus

# Tracer la carte de contr le
plt.figure(figsize=(12, 6))
plt.plot(residus, marker='o', linestyle='-', color='b', label='R sidus')
plt.axhline(y=moyenne_residus, color='r', linestyle='--', label='Moyenne')
plt.axhline(y=limite_superieure, color='g', linestyle='--', label='Limite
    Sup rieure de Contr le')
plt.axhline(y=limite_inferieure, color='g', linestyle='--', label='Limite
    Inf rieure de Contr le')
plt.title('Carte de Contr le des R sidus')
plt.xlabel('Observation')
plt.ylabel('R sidu')
plt.legend()
plt.show()
```

Explication :

— **But** : Identifier les points hors contrôle indiquant des problèmes de modèle.

— **Limites de Contrôle** : Fixées à ± 3 écarts-types de la moyenne des résidus.

12.3 Analyse de la Capacité du Processus

```
# Indices de capacit  du processus
ecart_type_processus = np.std(y_test)
limites_specification = [np.min(y_test), np.max(y_test)]  # Supposons que les
    sp cifications sont le min et le max de y_test
moyenne_processus = np.mean(y_test)
```

```
# Calculer Cp et Cpk
Cp = (limites_specification[1] - limites_specification[0]) / (6 *
    ecart_type_processus)
Cpk = min((limites_specification[1] - moyenne_processus), (moyenne_processus -
    limites_specification[0])) / (3 * ecart_type_processus)

print(f'Capacit  du Processus Cp : {Cp:.4f}')
print(f'Capacit  du Processus Cpk : {Cpk:.4f}')
```

Explication :

— **Cp et Cpk** : Indices pour évaluer la capacité du processus à respecter les spécifications.

— **Interprétation** :

— Cp > 1 : Le processus a le potentiel pour respecter les spécifications.

— Cpk > 1 : Le processus est centré entre les limites de spécification.

13 Considérations sur la Gestion de la Qualité

13.1 Cycle Planifier-Faire-Vérifier-Agir (PDCA)

1. **Planifier** : Définition du problème, des objectifs et planification de la génération des données et de l'approche de modélisation.
2. **Faire** : Exécution de la génération des données, du prétraitement et de l'entraînement du modèle.
3. **Vérifier** : Évaluation des performances du modèle et application des outils du CSP.
4. **Agir** : Identification des domaines d'amélioration, tels que l'ajustement du modèle ou l'amélioration de la qualité des données.

13.2 Amélioration Continue

— **Boucle de Rétroaction** : Utiliser les résultats de l'évaluation pour affiner le modèle.
— **Formation du Personnel** : Assurer que les membres de l'équipe comprennent les principes du CSP et de la CE appliqués dans la modélisation.
— **Documentation** : Maintenir des enregistrements détaillés des procédures de modélisation et des résultats.

13.3 Conformité aux Normes ISO

— **Principes ISO 9001** :
  — **Orientation Client** : Le modèle vise à améliorer la qualité du processus, bénéficiant aux clients.
  — **Approche Processus** : Le processus systématique de modélisation s'aligne sur le principe d'approche processus.
  — **Amélioration** : Évaluation et raffinement continus du modèle.

14 Conclusion

Nous avons développé avec succès un modèle de réseau de neurones pour optimiser un processus complexe en intégrant les principes du CSP et de la CE. Le modèle capture les relations linéaires et les interactions entre les variables, fournissant des informations précieuses sur la dynamique du processus.

14.1 Points Clés

- **Génération des Données** : Simulation d'un processus réaliste avec des interactions et de la variabilité.
- **Approche de Modélisation** : Utilisation de réseaux de neurones capables de gérer des données de haute dimensionnalité.
- **Outils de Qualité** : Application des méthodes du CSP pour surveiller et contrôler les performances du modèle.
- **Gestion de la Qualité** : Alignement du projet sur les principes de gestion de la qualité pour une amélioration continue.

14.2 Étapes Suivantes

- **Ajustement des Hyperparamètres** : Ajuster l'architecture du modèle, les taux d'apprentissage et les paramètres de régularisation pour optimiser les performances.
- **Analyse de l'Importance des Variables** : Utiliser des techniques comme l'importance par permutation ou les valeurs SHAP pour comprendre l'impact de chaque variable.
- **Montée en Échelle** : Envisager d'ajouter plus d'échantillons ou de variables si nécessaire et si cela est réalisable sur le plan computationnel.
- **Application des Techniques du CSP** : Mettre en œuvre des méthodes de Contrôle Statistique des Processus pour surveiller les prédictions du modèle dans le temps.

15 Références

— Montgomery, D. C. (2009). *Introduction to Statistical Quality Control*. Wiley.
— ISO. (2015). *ISO 9001 :2015 Systèmes de management de la qualité*.
— Shewhart, W. A. (1931). *Economic Control of Quality of Manufactured Product*. D. Van Nostrand Company.
— Deming, W. E. (1986). *Out of the Crisis*. MIT Center for Advanced Engineering Study.