Rapport de l'ajout d'une deuxième couche cachée pour un modèle de réseaux de neuronnes artificielles

Mahmoud El Mehdi El Khadiri

Chakir Mohammed

Supervised by

Souad Abderafie

November 2, 2023

Contents

1 Expérimentation et Sélection du Modèle Optimal

Dans cette section, nous décrirons le processus d'expérimentation pour identifier la configuration optimale de la deuxième couche cachée dans notre réseau de neurones. L'objectif était de tester différentes combinaisons de nombres de neurones et de fonctions d'activation afin de minimiser les erreurs RMSE et MAE.

1.1 Configuration et Entraînement des Modèles

Nous avons commencé par charger nos données et les préparer pour l'entraînement. Les données étaient divisées en ensembles d'entraînement et de test. Pour chaque combinaison de nombre de neurones et de fonction d'activation, un modèle a été créé et entraîné. Les détails de l'architecture de ces modèles et de leur processus d'entraînement sont présentés dans le Listing 1.

```
# Charger les données
data = pd.read_excel("C:/Users/banjo/Downloads/Scaled_Base_AP.xlsx")

# Séparer les variables indépendantes (X) et la variable dépendante (y)
X = data[['x1', 'x2', 'x3']].values
y = data['Y'].values

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Function to create a model with 20 neurons in the second hidden layer
def create_model():
    model = Sequential()
    model.add(Dense(5, input_dim=3, activation='tanh'))  # First hidden layer with 5
    ↪  neurons
    model.add(Dense(23, activation='relu'))              # Second hidden layer with
    ↪  20 neurons
    model.add(Dense(1, activation='linear'))             # Output layer
    model.compile(Adam(learning_rate=0.001), 'mean_squared_error')
    return model

# Create the model
model = create_model()

# Train the model
model.fit(X_train, y_train, epochs=1000, batch_size=10, verbose=1)
```

Listing 1: Code pour la création et l'entraînement des modèles de réseau de neurones

1.2 Sélection du Modèle Optimal

Après avoir entraîné plusieurs modèles avec différentes configurations, nous avons tracé leurs performances pour identifier la configuration la plus optimale. La Figure 1 montre le graphe des erreurs RMSE et MAE en fonction du nombre de neurones dans la deuxième couche cachée.

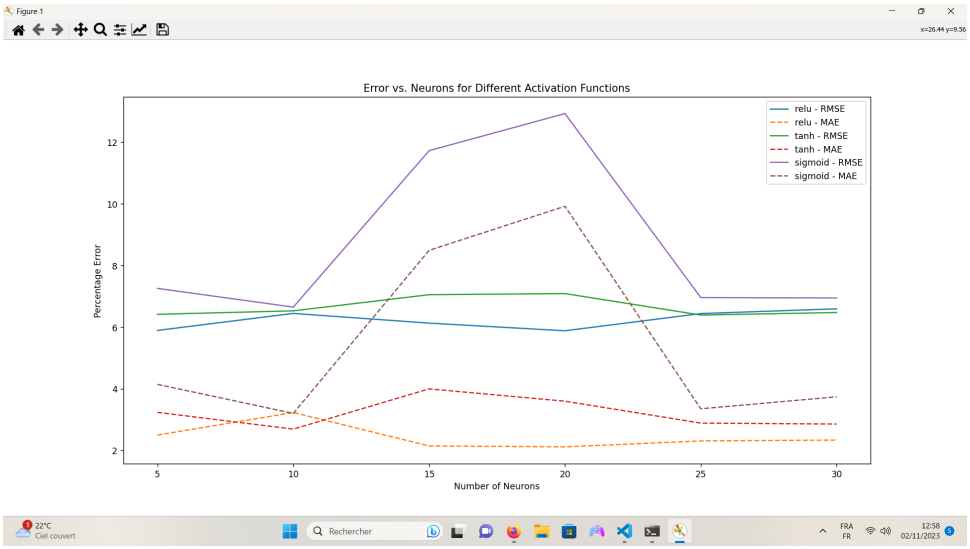

Figure 1: Graphe des erreurs RMSE et MAE pour différentes configurations de la deuxième couche cachée

Comme illustré sur le graphe, la configuration la plus performante se situait autour de 20 neurones avec la fonction d'activation ReLU. Cependant, après des calculs supplémentaires, nous avons constaté que le modèle le plus optimal était celui avec 23 neurones dans la deuxième couche cachée, utilisant également la fonction d'activation ReLU. Les résultats détaillés de cette évaluation sont présentés dans le Listing 2.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 20
 dense_1 (Dense)             (None, 23)                138
 dense_2 (Dense)             (None, 1)                 24
=================================================================
Total params: 182
Trainable params: 182
Non-trainable params: 0
_________________________________________________________________
RMSE (%): 4.1567646588743425
MAE (%): 1.727556428257018
```

Listing 2: Sortie du terminal après calculs montrant le modèle le plus optimal

1.3 Visualisation de la Topologie du Modèle Optimal

La topologie du modèle le plus optimal, avec 23 neurones dans la deuxième couche cachée, est visualisée dans la Figure 2. Cette image a été générée pour fournir une représentation claire de l'architecture du réseau de neurones.

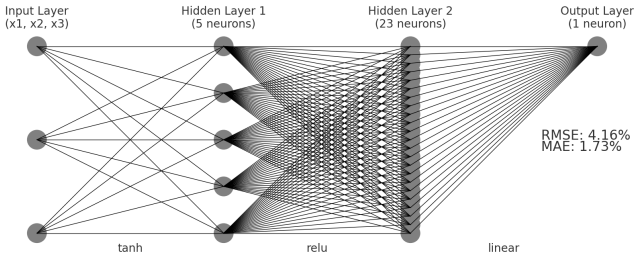

Figure 2: Topologie du modèle de réseau de neurones optimal

2 Discussion

3 Discussion

L'objectif principal de cette étude était d'améliorer les performances de notre modèle de réseau de neurones en optimisant la configuration de la deuxième couche cachée. Les résultats ont démontré une amélioration considérable par rapport à notre modèle initial à une seule couche cachée, qui présentait un RMSE de 6.8% et un MAE de 4.29%. En contraste, le modèle optimisé avec 23 neurones dans la deuxième couche cachée a atteint un RMSE de 4.1567646588743425% et un MAE de 1.727556428257018%, reflétant une réduction significative des erreurs. Cette amélioration met en évidence l'importance de l'expérimentation et du réglage fin des hyperparamètres dans le développement de modèles de machine learning.

4 Conclusion

Ce rapport a illustré l'importance d'une démarche méthodique et expérimentale dans la sélection des configurations optimales pour les réseaux de neurones. Grâce à une série d'expérimentations et d'ajustements, nous avons pu affiner notre modèle pour obtenir des prédictions nettement plus précises. Les enseignements tirés de cette étude serviront de base solide pour les futures recherches et applications dans le domaine de l'intelligence artificielle.

A Code Source pour l'Expérimentation des Modèles

Dans cette annexe, nous fournissons le code source utilisé pour tester différentes configurations de la deuxième couche cachée dans nos modèles de réseau de neurones.

A.1 Code pour Tester Différents Nombres de Neurones

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# Charger les données
data = pd.read_excel("C:/Users/banjo/Downloads/Scaled_Base_AP.xlsx")

# Séparer les variables indépendantes (X) et la variable dépendante (y)
X = data[['x1', 'x2', 'x3']].values
y = data['Y'].values

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Function to create a model with a varying number of neurons in the second hidden
↪  layer
def create_model(neurons):
    model = Sequential()
    model.add(Dense(5, input_dim=3, activation='tanh'))  # First hidden layer with 5
    ↪  neurons
    model.add(Dense(neurons, activation='relu'))         # Second hidden layer with a
    ↪  variable number of neurons
    model.add(Dense(1, activation='linear'))             # Output layer
    model.compile(Adam(learning_rate=0.001), 'mean_squared_error')
    return model

# Variables to store the best model information
best_rmse = float('inf')
best_mae = float('inf')
best_model = None
best_neuron_count = 0

# Lists to store the RMSE and MAE values for different models
rmse_values = []
mae_values = []

# Range of neurons to try for the second hidden layer
neurons_range = range(15, 26)  # From 15 to 25 neurons

# Evaluate models with different number of neurons
for neurons in neurons_range:
    model = create_model(neurons)
    model.fit(X_train, y_train, epochs=1000, batch_size=10, verbose=0)
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    rmse_values.append(rmse / np.mean(y_test) * 100)  # Convert to percentage
    mae_values.append(mae / np.mean(y_test) * 100)    # Convert to percentage

    # Update best model if current model is better
    if rmse < best_rmse:
        best_rmse = rmse
        best_mae = mae
        best_model = model
        best_neuron_count = neurons
```

A.2 Code pour le Modèle Optimal

```python
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# Charger les données
data = pd.read_excel("C:/Users/banjo/Downloads/Scaled_Base_AP.xlsx")

# Séparer les variables indépendantes (X) et la variable dépendante (y)
X = data[['x1', 'x2', 'x3']].values
y = data['Y'].values

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Function to create a model with 20 neurons in the second hidden layer
def create_model():
    model = Sequential()
    model.add(Dense(5, input_dim=3, activation='tanh'))  # First hidden layer with 5
    ↪ neurons
    model.add(Dense(23, activation='relu'))              # Second hidden layer with
    ↪ 20 neurons
    model.add(Dense(1, activation='linear'))             # Output layer
    model.compile(Adam(learning_rate=0.001), 'mean_squared_error')
    return model

# Create the model
model = create_model()

# Train the model
model.fit(X_train, y_train, epochs=1000, batch_size=10, verbose=1)

# Evaluate the model
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

# Print the model's topology and metrics
model.summary()
# Assuming y_test is not a single constant value
y_test_range = np.max(y_test) - np.min(y_test)
if y_test_range > 0:
    print(f"RMSE (%): {rmse / y_test_range * 100}")
    print(f"MAE (%): {mae / y_test_range * 100}")
else:
    print("Cannot compute percentage errors because the range of y_test is 0.")

# Save the model
model.save('model_with_20_neurons.h5')
```

Listing 4: Code pour créer et évaluer le modèle optimal avec 23 neurones dans la deuxième couche cachée