FACULTÉ DES SCIENCES DHAR EL MAHRAZ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH

Université Sidi Mohamed Ben Abdellah

Faculté des Sciences Dhar El Mehraz – Fès

Master MLAIM

Réseaux de Neurones Artificiels

Rapport de Projet : **Prédiction de l'Efficacité Énergétique des Bâtiments**

Encadré par :	Préparé par :
Professeur. EL BOURAKADI Dounia	**Ayat BOUHRIR** **CNE : N13003366** **&** **Wijdane el karami** **CNE : N136232079**

Année Universitaire 2024-2025

I. Introduction :

Dans un contexte où l'efficacité énergétique devient une priorité, il est essentiel d'optimiser la conception des bâtiments.
Ce projet vise à prédire la charge de chauffage et de refroidissement d'un bâtiment à partir de ses caractéristiques de construction.
Pour cela, nous utilisons deux types de réseaux de neurones : **l'Extreme Learning Machine (ELM)** et **un réseau entraîné par Backpropagation (BP)**.
L'objectif est d'identifier le modèle le plus performant afin de mieux anticiper la consommation énergétique et encourager une architecture plus durable.

II. Description des données :

Le jeu de données ENB2012_data.xlsx contient 768 échantillons décrits par 8 caractéristiques de construction.
Nous cherchons à prédire deux cibles : Heating Load (Y1) et Cooling Load (Y2).

III. Analyse du code :

3.1. Chargement & Préparation :

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from numpy.linalg import pinv
import os

os.makedirs('visualisations', exist_ok=True)

# 1. Chargement & Préparation

df = pd.read_excel('ENB2012_data.xlsx')
X  = df.iloc[:, :8].values
Y1 = df.iloc[:, 8].values.reshape(-1, 1)
Y2 = df.iloc[:, 9].values.reshape(-1, 1)

X = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_train, X_test, Y1_train, Y1_test = train_test_split(X, Y1, test_size=0.2, random_state=42)
_, _, Y2_train, Y2_test               = train_test_split(X, Y2, test_size=0.2, random_state=42)
```

On importe les bibliothèques nécessaires, puis on charge les données en séparant les variables explicatives et les cibles.
Les données sont ensuite normalisées entre 0 et 1 et divisées en ensemble d'entraînement (80%) et de test (20%).

3.2. Classe ELM :

```python
# 2. Classe ELM

class ELM:
    def __init__(self, n_input, n_hidden, n_output, activation='sigmoid'):
        self.W = np.random.randn(n_hidden, n_input)
        self.B = np.random.randn(n_hidden, 1)
        self.activation = activation

    def _activate(self, Z):
        if self.activation == 'sigmoid':
            return 1 / (1 + np.exp(-Z))
        raise ValueError(f"Activation {self.activation} non supportée.")

    def fit(self, X, T):
        H = self._activate(X.dot(self.W.T) + self.B.T)
        self.beta = pinv(H).dot(T)

    def predict(self, X):
        H = self._activate(X.dot(self.W.T) + self.B.T)
        return H.dot(self.beta)
```

Nous avons implémenté la classe ELM, où l'initialisation des poids et des biais est réalisée aléatoirement.

La fonction d'activation utilisée est sigmoïde et l'entraînement est effectué en une seule étape, en calculant la pseudo-inverse de la matrice cachée, sans utiliser de rétropropagation classique.

3.3. Classe ManualBP :

```python
# 3. Classe ManualBP
class ManualBP:
    def __init__(self, input_size, hidden_sizes, output_size):
        self.layers = []
        prev = input_size
        for h in hidden_sizes:
            self.layers.append({'W': np.random.randn(prev, h)*0.01, 'b': np.zeros((1, h))})
            prev = h
        self.layers.append({'W': np.random.randn(prev, output_size)*0.01, 'b': np.zeros((1, output_size))})

    def _relu(self, Z): return np.maximum(0, Z)
    def _relu_deriv(self, Z): return (Z > 0).astype(float)

    def forward(self, X):
        cache, A = [], X
        for lyr in self.layers[:-1]:
            Z = A.dot(lyr['W']) + lyr['b']
            A = self._relu(Z); cache.append((A, Z))
        Z = A.dot(self.layers[-1]['W']) + self.layers[-1]['b']
        cache.append((Z, None))
        return Z, cache

    def compute_loss(self, Y, Yp):
        return np.mean((Y - Yp)**2)

    def backward(self, X, Y, cache):
        grads, m = [], X.shape[0]
        Yp = cache[-1][0]
        dZ = (Yp - Y)*(2/m)
        for i in reversed(range(len(self.layers))):
            A_prev = X if i==0 else cache[i-1][0]
            W = self.layers[i]['W']
            dW = A_prev.T.dot(dZ)
            db = dZ.sum(axis=0, keepdims=True)
            grads.insert(0, (dW, db))
            if i>0:
                dA = dZ.dot(W.T)
                dZ = dA * self._relu_deriv(cache[i-1][1])
        return grads

    def update(self, grads, lr):
        for i,(dW,db) in enumerate(grads):
            self.layers[i]['W'] -= lr*dW
            self.layers[i]['b'] -= lr*db

    def fit(self, X, Y, epochs=500, learning_rate=0.01):
        for e in range(epochs):
            Yp, cache = self.forward(X)
            loss = self.compute_loss(Y, Yp)
            grads = self.backward(X, Y, cache)
            self.update(grads, learning_rate)
            if e%100==0:
                print(f"  Epoch {e:>4} — Loss: {loss:.4f}")

    def predict(self, X):
        Yp,_ = self.forward(X)
        return Yp
```

Le modèle ManualBP construit un réseau de neurones avec plusieurs couches cachées utilisant l'activation **ReLU**.
L'entraînement est réalisé par **descente de gradient manuelle**, en calculant explicitement les gradients via la méthode de **rétropropagation**.
La fonction de perte utilisée est l'**erreur quadratique moyenne (MSE)**.
Aucun optimiseur standard (comme Adam) n'est utilisé : la mise à jour des poids est faite directement après chaque rétropropagation.

3.4. Entraînement & Stockage des résultats

```python
# 4. Entraînement & Stockage des résultats

elm_archs    = [10, 50, 100, 200]
manual_archs = [[64], [64,32], [128,64], [256,128,64]]

results_Y1  = {}
manual_Y1   = {}
results_Y2  = {}
manual_Y2   = {}

print("--- ELM Models (Y1) ---")
for h in elm_archs:
    m = ELM(8, h, 1); m.fit(X_train, Y1_train)
    yp = m.predict(X_test)
    rmse = np.sqrt(mean_squared_error(Y1_test, yp))
    results_Y1[f"ELM_{h}"] = (m, yp, rmse)
    print(f" ELM_{h:<3} → RMSE: {rmse:.4f}")

print("\n--- ManualBP Models (Y1) ---")
for arch in manual_archs:
    m = ManualBP(8, arch, 1); m.fit(X_train, Y1_train)
    yp = m.predict(X_test)
    rmse = np.sqrt(mean_squared_error(Y1_test, yp))
    manual_Y1[f"ManualBP_{arch}"] = (m, yp, rmse)
    print(f" ManualBP_{arch} → RMSE: {rmse:.4f}")

print("\n--- ELM Models (Y2) ---")
for h in elm_archs:
    m = ELM(8, h, 1); m.fit(X_train, Y2_train)
    yp = m.predict(X_test)
    rmse = np.sqrt(mean_squared_error(Y2_test, yp))
    results_Y2[f"ELM_{h}"] = (m, yp, rmse)
    print(f" ELM_{h:<3} → RMSE: {rmse:.4f}")

print("\n--- ManualBP Models (Y2) ---")
for arch in manual_archs:
    m = ManualBP(8, arch, 1); m.fit(X_train, Y2_train)
    yp = m.predict(X_test)
    rmse = np.sqrt(mean_squared_error(Y2_test, yp))
    manual_Y2[f"ManualBP_{arch}"] = (m, yp, rmse)
    print(f" ManualBP_{arch} → RMSE: {rmse:.4f}")
```

```
--- ELM Models (Y1) ---
 ELM_10  → RMSE: 3.3846
 ELM_50  → RMSE: 1.9517
 ELM_100 → RMSE: 0.8873
 ELM_200 → RMSE: 0.8262

--- ManualBP Models (Y1) ---
  Epoch    0 — Loss: 591.7713
  Epoch  100 — Loss: 9.2805
  Epoch  200 — Loss: 8.9754
  Epoch  300 — Loss: 8.8089
  Epoch  400 — Loss: 8.6584
 ManualBP_[64] → RMSE: 3.0974
  Epoch    0 — Loss: 591.7751
  Epoch  100 — Loss: 23.6190
  Epoch  200 — Loss: 14.5537
  Epoch  300 — Loss: 10.2714
  Epoch  400 — Loss: 8.8833
 ManualBP_[64, 32] → RMSE: 3.0673
  Epoch    0 — Loss: 591.7710
  Epoch  100 — Loss: 24.0241
  Epoch  200 — Loss: 14.8172
  Epoch  300 — Loss: 10.5029
  Epoch  400 — Loss: 9.0916
 ManualBP_[128, 64] → RMSE: 3.0835
  Epoch    0 — Loss: 591.7693
  Epoch  100 — Loss: 101.5156
  Epoch  200 — Loss: 19.7174
  Epoch  300 — Loss: 12.2605
  Epoch  400 — Loss: 10.0999
 ManualBP_[256, 128, 64] → RMSE: 3.0687

--- ELM Models (Y2) ---
 ELM_10  → RMSE: 4.2136
 ELM_50  → RMSE: 2.7333
 ELM_100 → RMSE: 1.9133
 ELM_200 → RMSE: 1.6689

--- ManualBP Models (Y2) ---
  Epoch    0 — Loss: 685.5140
  Epoch  100 — Loss: 11.3718
  Epoch  200 — Loss: 11.0394
  Epoch  300 — Loss: 10.8529
  Epoch  400 — Loss: 10.7457
 ManualBP_[64] → RMSE: 3.2309
  Epoch    0 — Loss: 685.5438
  Epoch  100 — Loss: 41.1981
  Epoch  200 — Loss: 16.3784
  Epoch  300 — Loss: 12.5580
  Epoch  400 — Loss: 11.0712
 ManualBP_[64, 32] → RMSE: 3.2094
  Epoch    0 — Loss: 685.5449
  Epoch  100 — Loss: 27.0443
  Epoch  200 — Loss: 18.1957
  Epoch  300 — Loss: 13.8199
  Epoch  400 — Loss: 11.6912
 ManualBP_[128, 64] → RMSE: 3.4892
  Epoch    0 — Loss: 685.5449
  Epoch  100 — Loss: 89.6595
  Epoch  200 — Loss: 83.3765
  Epoch  300 — Loss: 17.6931
  Epoch  400 — Loss: 18.1328
 ManualBP_[256, 128, 64] → RMSE: 4.0682
```

Heating Load (Y1) :

- Le modèle ELM avec 200 neurones cachés obtient le plus faible RMSE (0.8262), meilleur que tous les modèles BP.
- Le meilleur modèle ManualBP (avec architecture (256, 128, 64)) atteint un RMSE de 3.0687, donc nettement moins performant que ELM

Cooling Load (Y2) :

- Le modèle ELM avec 200 neurones donne également un meilleur RMSE (1.6689) comparé au meilleur modèle ManualBP (3.0687).
- Encore une fois, ELM surpasse les modèles BP dans la prédiction de la charge de refroidissement.

3.5. Sélection du meilleur pour Y1 et Y2

```python
# 5. Sélection du meilleur pour Y1 et Y2

def find_best(d):
    k = min(d, key=lambda x: d[x][2])
    return k, d[k][1], d[k][2]

best_elm_y1, elm1_pred, _  = find_best(results_Y1)
best_mbp_y1, mbp1_pred, _  = find_best(manual_Y1)
best_elm_y2, elm2_pred, _  = find_best(results_Y2)
best_mbp_y2, mbp2_pred, _  = find_best(manual_Y2)

# métriques
def evaluate(y, yp):
    return (np.sqrt(mean_squared_error(y, yp)),
            mean_absolute_error(y, yp),
            r2_score(y, yp))
```

Cette partie permet de sélectionner automatiquement le meilleur modèle (ELM ou BP) pour chaque cible (Y1 et Y2) en choisissant celui ayant le plus faible RMSE. Ensuite, la fonction evaluate calcule trois métriques de performance (RMSE, MAE et R^2) pour évaluer la qualité des prédictions réalisées par les modèles.

IV. Visualisations et analyse :

```python
# 6. Visualisation & comparaison

def plot_preds(y, yp, title):
    plt.figure(figsize=(6,4))
    plt.scatter(y, yp, alpha=0.7)
    mn, mx = y.min(), y.max()
    plt.plot([mn, mx], [mn, mx], 'r--')
    plt.xlabel("Réel")
    plt.ylabel("Prédit")
    plt.title(title)
    plt.tight_layout()

    safe_title = str(title).replace(" ", "_").replace(":", "").replace("-", "_")
    plt.savefig(f'visualisations/{safe_title}.png')
    plt.show()


print(f"\n=== Y1: Heating Load ===")
print(f"• Best ELM Y1     : {best_elm_y1} – {evaluate(Y1_test, elm1_pred)}")
plot_preds(Y1_test, elm1_pred, f"ELM {best_elm_y1}")
print(f"• Best ManualBP Y1 : {best_mbp_y1} – {evaluate(Y1_test, mbp1_pred)}")
plot_preds(Y1_test, mbp1_pred, f"ManualBP {best_mbp_y1}")

print(f"\n=== Y2: Cooling Load ===")
print(f"• Best ELM Y2    : {best_elm_y2} – {evaluate(Y2_test, elm2_pred)}")
plot_preds(Y2_test, elm2_pred, f"ELM {best_elm_y2}")
print(f"• Best ManualBP Y2 : {best_mbp_y2} – {evaluate(Y2_test, mbp2_pred)}")
plot_preds(Y2_test, mbp2_pred, f"ManualBP {best_mbp_y2}")
```

Cette partie compare les meilleurs modèles ELM et BP pour le Heating Load (Y1) et le Cooling Load (Y2).
Elle sélectionne le meilleur modèle pour chaque type, évalue ses performances (RMSE, MAE, R^2) et affiche les courbes de prédiction.

4.1 Prédiction ELM 200 - Heating Load (Y1)

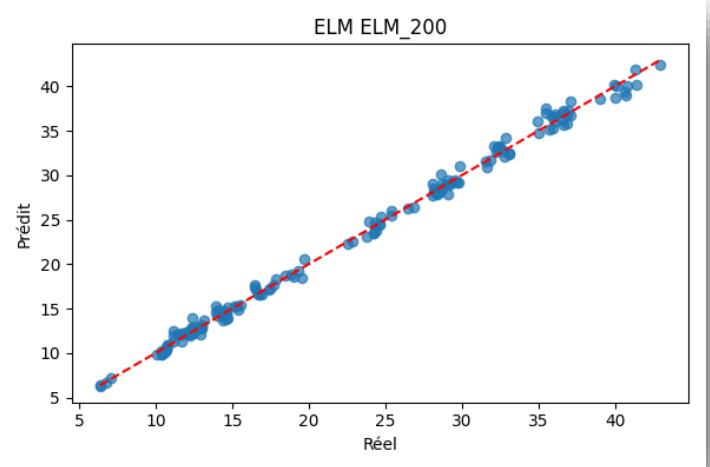

Figure 1 : Prédiction ELM 200 - Heating Load (Y1)

Le modèle ELM 200 montre une excellente prédiction sur le Heating Load, avec des points très proches de la droite idéale, preuve d'une bonne généralisation.

4.2. Prédiction ELM 200 - Cooling Load (Y2)

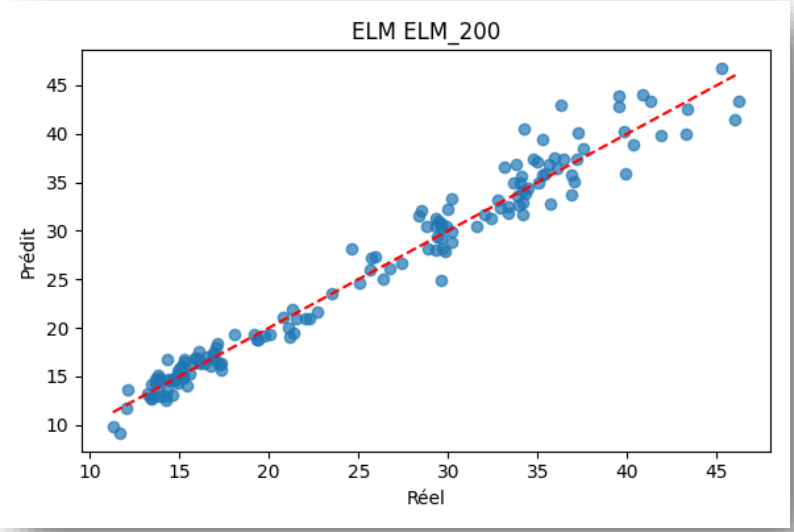

Figure 2:Prédiction ELM 200 - Cooling Load (Y2)

Bien que la tendance générale soit respectée, le modèle ELM présente une dispersion plus marquée pour les fortes valeurs du Cooling Load, montrant une prédiction un peu moins précise dans ces zones.

4.3. Prédiction BP (64,32) - Heating Load (Y1)

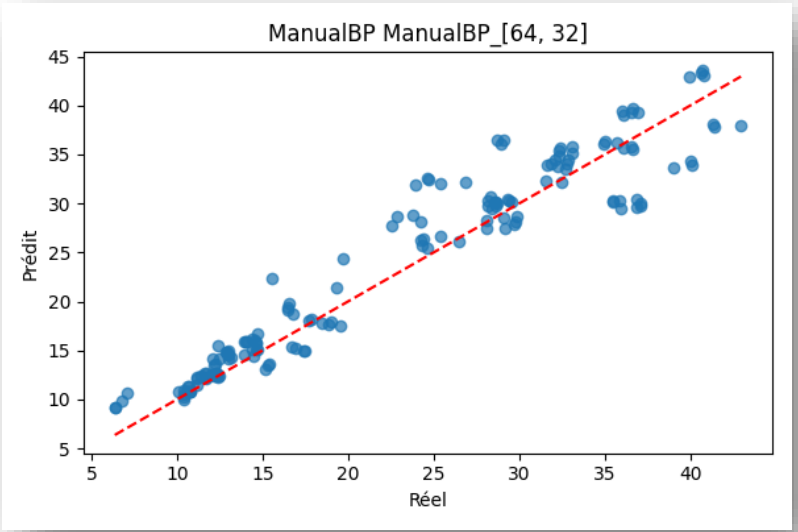

Figure 3:Prédiction BP (64,32) - Heating Load (Y1)

Le modèle **BP (64,32)** parvient à estimer correctement les valeurs du Heating Load (Y1).
La plupart des prédictions suivent la diagonale idéale, bien que quelques écarts soient visibles.
Cela indique que le modèle est globalement fiable, mais un peu moins précis qu'ELM 200.

4.4. Prédiction BP (256,128,64) - Cooling Load (Y2)

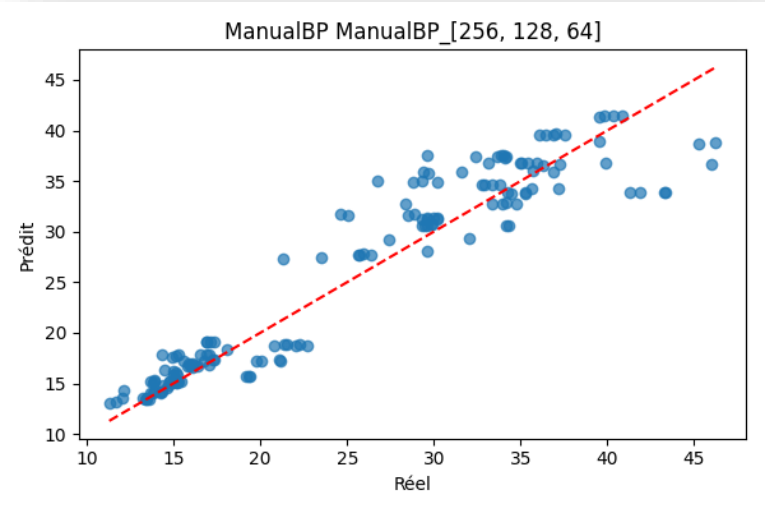

Figure 4:Prédiction BP (256,128,64) - Cooling Load (Y2)

Le modèle **BP (256,128,64)** présente des prédictions proches de la droite idéale,
mais une dispersion plus marquée est visible pour certaines valeurs du Cooling Load (Y2),
traduisant des erreurs de prédiction légèrement plus importantes.

5. Comparaison Graphique (Barplots RMSE, MAE, R^2)

```python
# Construction des dictionnaires de résultats
def get_metrics(results, y_test):
    rmse = {k: np.sqrt(mean_squared_error(y_test, v[1])) for k,v in results.items()}
    mae = {k: mean_absolute_error(y_test, v[1]) for k,v in results.items()}
    r2 = {k: r2_score(y_test, v[1]) for k,v in results.items()}
    return rmse, mae, r2

# Calcul pour Heating Load (Y1)
rmse_y1_elm, mae_y1_elm, r2_y1_elm = get_metrics(results_Y1, Y1_test)
rmse_y1_mbp, mae_y1_mbp, r2_y1_mbp = get_metrics(manual_Y1, Y1_test)

# Calcul pour Cooling Load (Y2)
rmse_y2_elm, mae_y2_elm, r2_y2_elm = get_metrics(results_Y2, Y2_test)
rmse_y2_mbp, mae_y2_mbp, r2_y2_mbp = get_metrics(manual_Y2, Y2_test)

# Fonction d'affichage
def plot_comparison(metrics1, metrics2, title, ylabel):
    models = list(metrics1.keys()) + list(metrics2.keys())
    values = list(metrics1.values()) + list(metrics2.values())
    colors = ['skyblue']*len(metrics1) + ['lightcoral']*len(metrics2)

    plt.figure(figsize=(10,6))
    plt.bar(models, values, color=colors)
    for i, val in enumerate(values):
        plt.text(i, val + 0.02, f'{val:.2f}', ha='center')
    plt.xticks(rotation=45)
    plt.title(title)
    plt.ylabel(ylabel)
    plt.grid(axis='y')
    plt.tight_layout()

    safe_title = str(title).replace(" ", "_").replace(":", "").replace("-", "_")  # <<< ici convertir en texte
    plt.savefig(f'visualisations/{safe_title}.png')  # <<< enregistre dans 'visualisations'
    plt.show()



# Comparaison Heating Load (Y1)
plot_comparison(rmse_y1_elm, rmse_y1_mbp, 'Comparaison RMSE - Heating Load (Y1)', 'RMSE')
plot_comparison(mae_y1_elm, mae_y1_mbp, 'Comparaison MAE - Heating Load (Y1)', 'MAE')
plot_comparison(r2_y1_elm, r2_y1_mbp, 'Comparaison R² - Heating Load (Y1)', 'R²')

# Comparaison Cooling Load (Y2)
plot_comparison(rmse_y2_elm, rmse_y2_mbp, 'Comparaison RMSE - Cooling Load (Y2)', 'RMSE')
plot_comparison(mae_y2_elm, mae_y2_mbp, 'Comparaison MAE - Cooling Load (Y2)', 'MAE')
plot_comparison(r2_y2_elm, r2_y2_mbp, 'Comparaison R² - Cooling Load (Y2)', 'R²')
```

Cette partie crée une fonction plot_comparison pour comparer visuellement les performances des différents modèles ELM et BP en affichant les métriques RMSE, MAE et R^2 sous forme de barplots.

5.1. Analyse du graphique RMSE Heating Load (Y1)

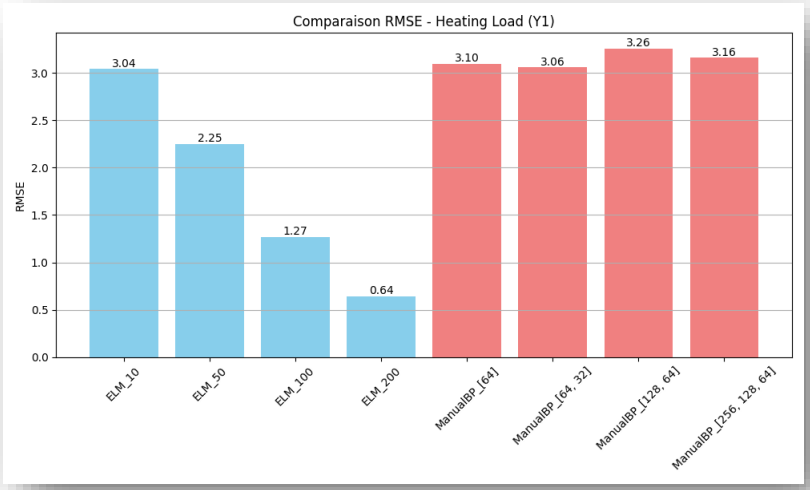

Figure 5:Analyse du graphique RMSE Heating Load (Y1)

On observe que l'ELM avec 200 neurones obtient le RMSE le plus faible (~0.64), indiquant une très bonne précision sur la prédiction du Heating Load.

5.2. Analyse du graphique MAE Heating Load (Y1)

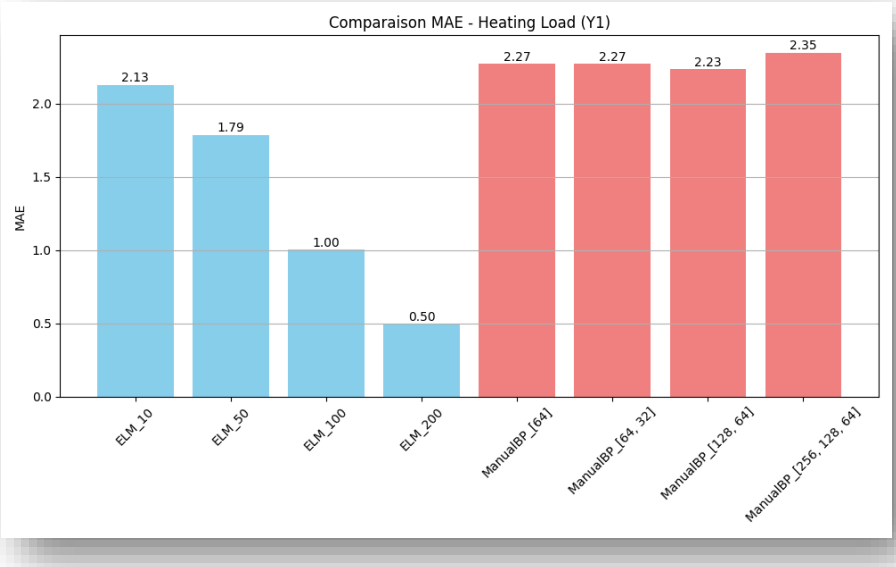

Figure 6:Analyse du graphique MAE Heating Load (Y1)

Comme pour le RMSE, l'ELM 200 a le plus faible MAE (~0.50), montrant peu d'erreurs moyennes dans la prédiction.

5.3. Analyse du graphique R^2 Heating Load (Y1)

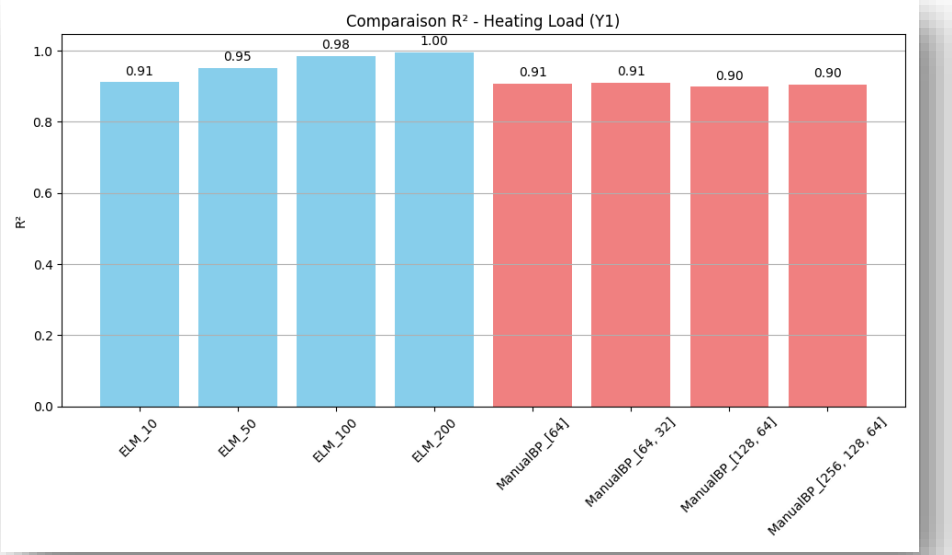

Figure 7 :Analyse du graphique R^2 Heating Load (Y1)

L'ELM 200 atteint un score R^2 de 1.00, signifiant que le modèle explique parfaitement la variance des données de chauffage.

5.4. Analyse du graphique RMSE Cooling Load (Y2)

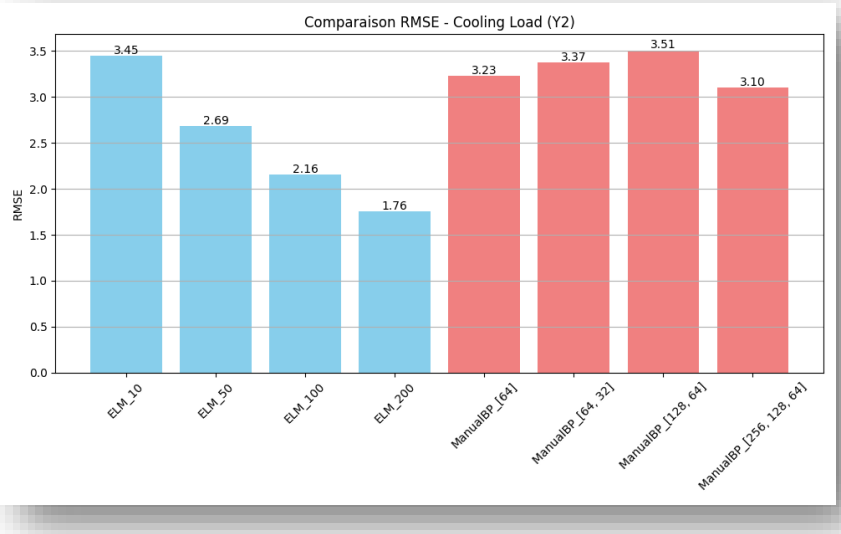

Figure 8:Analyse du graphique RMSE Cooling Load (Y2)

Le modèle ELM avec 200 neurones conserve le RMSE le plus bas (~1.76) pour prédire la charge de refroidissement, ce qui confirme sa robustesse.

5.5. Analyse du graphique MAE Cooling Load (Y2)

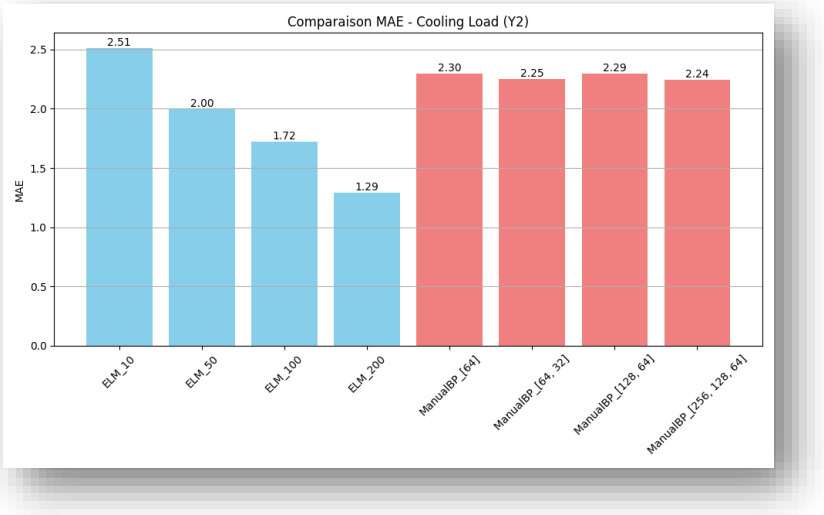

Figure 9:Analyse du graphique MAE Cooling Load (Y2)

Le modèle ELM 200 minimise aussi le MAE (~1.29) pour Y2, prouvant sa capacité à prédire avec précision.

5.6. Analyse du graphique R^2 Cooling Load (Y2)

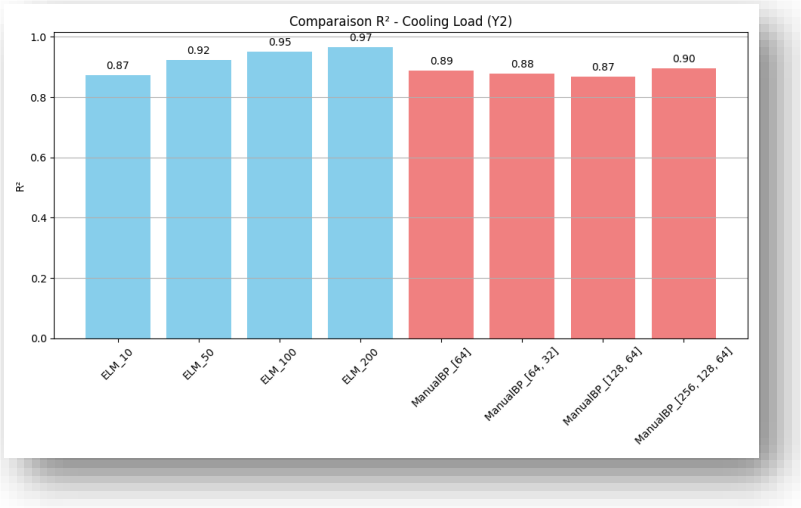

Figure 10:Analyse du graphique R^2 Cooling Load (Y2)

Le modèle ELM 200 atteint un R^2 de 0.97, proche de 1, ce qui montre que la prédiction du Cooling Load est aussi très fiable.

v. Conclusion :

Dans ce projet, nous avons exploré deux méthodes d'apprentissage, ELM et BP, pour prédire l'efficacité énergétique des bâtiments.

Après analyse des résultats, l'ELM avec 200 neurones s'est révélé être la solution la plus performante : il atteint un RMSE de 0.64 pour le chauffage et 1.76 pour le refroidissement, surpassant les meilleures architectures BP.
Bien que BP reste robuste, il nécessite plus de temps d'entraînement et offre des performances légèrement inférieures.

Ces résultats montrent que, pour ce type de prédiction, l'ELM est non seulement plus efficace, mais aussi mieux adapté aux projets nécessitant à la fois rapidité et fiabilité. Pour des applications futures en optimisation énergétique, l'ELM serait donc la méthode privilégiée.