Applications Of Machine Learning Assignment 4

-by Srilalith Nampally

Part 1:

Code:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn import preprocessing as preproc

from sklearn.metrics import accuracy_score, roc_auc_score, precision_score, recall_score, confusion_matrix

import time


from sklearn.preprocessing import PolynomialFeatures



def getDF(path):

    df = pd.read_excel(path)

    return df



def getNormalized_and_train_test(df):

    # Separate features and target
```

```
    X = df.iloc[:, :-1]  # All rows, exclude the last column
    y = df.iloc[:, -1]  # All rows, just the last column

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # Step 2: Fit the scaler on the training data
    scaler = preproc.MinMaxScaler(feature_range=(-1, 1))
    scaler.fit(X_train)  # Compute the min and max values to be used for scaling

    # Step 3: Transform both the training and test data with the fitted scaler
    X_train_scaled = scaler.transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_test_scaled, y_train, y_test


def getPolyTransform(X_train, X_test):
    poly = PolynomialFeatures(degree=2, include_bias=False)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    return X_train_poly, X_test_poly


def getLogTransform(X_train, X_test):
    X_train_log = np.log(X_train + 1 - X_train.min())
```

```
    X_test_log = np.log(X_test + 1 - X_test.min())

    return X_train_log, X_test_log


def getCombinationTransform(X_train_poly, X_test_poly, X_train_log, X_test_log):
    X_train_combo = np.hstack([X_train_poly, X_train_log])
    X_test_combo = np.hstack([X_test_poly, X_test_log])

    return X_train_combo, X_test_combo


def logisticRegression(X_train, Y_train):
    logistic_model = LogisticRegression()
    logistic_model.fit(X_train, Y_train)

    return logistic_model


def getMetrics(model, X_test, Y_test):
    # Predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]  # Probabilities for the positive class

    # Evaluation Metrics
    n_iterations = model.n_iter_[0]
```

```python
    accuracy = accuracy_score(Y_test, y_pred)
    roc_auc = roc_auc_score(Y_test, y_pred_proba)
    precision = precision_score(Y_test, y_pred)
    recall = recall_score(Y_test, y_pred)
    conf_matrix = confusion_matrix(Y_test, y_pred)

    # Confusion matrix components
    tn, fp, fn, tp = conf_matrix.ravel()
    tpr = tp / (tp + fn)  # True Positive Rate
    fnr = fn / (fn + tp)  # False Negative Rate
    fpr = fp / (fp + tn)  # False Positive Rate
    tnr = tn / (tn + fp)  # True Negative Rate

    # Print metrics
    print(f'Classification test: [{n_iterations}] iterations', end=', ')
    print(f'accuracy: {accuracy:.4f}', end=', ')
    print(f'AUC: {roc_auc:.4f}')
    print(f'Precision: {precision:.6f}', end=', ')
    print(f'Recall: {recall:.6f}')
    print(f'Confusion Matrix:\n{conf_matrix}')
    print(f'TPR: {tpr:.4f}, FNR: {fnr:.4f}, FPR: {fpr:.4f}, TNR: {tnr:.4f}')


if __name__ == '__main__':
    start = time.time()
    df1 = getDF('VWXYZ.xlsx')
```

```
    end = time.time()
    print('Time it took to read the excel file: ', end - start)
    # print(df1)

    X_train_scaled, X_test_scaled, y_train, y_test = getNormalized_and_train_test(df1)
    originalModel = logisticRegression(X_train_scaled, y_train)
    print('\n\nMetrics for Original Dataset\n')
    getMetrics(originalModel, X_test_scaled, y_test)

    X_poly_train, X_poly_test = getPolyTransform(X_train_scaled, X_test_scaled)
    polyModel = logisticRegression(X_poly_train, y_train)
    print('\n\nMetrics for Polynomial Deg 2 Transformed Dataset\n')
    getMetrics(polyModel, X_poly_test, y_test)

    X_log_train, X_log_test = getLogTransform(X_train_scaled, X_test_scaled)
    logModel = logisticRegression(X_log_train, y_train)
    print('\n\nMetrics for Log Transformation Dataset\n')
    getMetrics(logModel, X_log_test, y_test)

    X_combi_train, X_combi_test = getCombinationTransform(X_poly_train, X_poly_test,
X_log_train, X_log_test)
    comboModel = logisticRegression(X_combi_train, y_train)
    print('\n\nMetrics for Combination Transformation\n')
    getMetrics(comboModel, X_combi_test, y_test)
```

Output:

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment_4> & C:/Users/srico/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/srico/OneDrive/Desktop/Applications of Machine Learning/Assignment_4/Part1.py"

Time it took to read the excel file: 4.046921730041504

Metrics for Original Dataset

Classification test: [11] iterations, accuracy: 0.8450, AUC: 0.9268

Precision: 0.853507, Recall: 0.849206

Confusion Matrix:

[[12038 2285]

 [2364 13313]]

TPR: 0.8492, FNR: 0.1508, FPR: 0.1595, TNR: 0.8405

Metrics for Polynomial Deg 2 Transformed Dataset

Classification test: [22] iterations, accuracy: 0.8459, AUC: 0.9274

Precision: 0.854786, Recall: 0.849333

Confusion Matrix:

[[12061 2262]

 [2362 13315]]

TPR: 0.8493, FNR: 0.1507, FPR: 0.1579, TNR: 0.8421

Metrics for Log Transformation Dataset

Classification test: [14] iterations, accuracy: 0.8437, AUC: 0.9255

Precision: 0.862649, Recall: 0.833705

Confusion Matrix:

[[12242 2081]

 [2607 13070]]

TPR: 0.8337, FNR: 0.1663, FPR: 0.1453, TNR: 0.8547

Metrics for Combination Transformation

Classification test: [64] iterations, accuracy: 0.8456, AUC: 0.9274

Precision: 0.854484, Recall: 0.849142

Confusion Matrix:

[[12056 2267]

 [2365 13312]]

TPR: 0.8491, FNR: 0.1509, FPR: 0.1583, TNR: 0.8417

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment_4>

Discussion:

The polynomial transformation seems to offer a slight improvement in model performance without a significant increase in complexity compared to the combination transformation.

The log transformation might not be as effective for this dataset based on the metrics observed.

The combination transformation increases model complexity (as seen in the number of iterations) without a proportional improvement in performance metrics.

Depending on the specific application and the cost of FP vs FN, you might opt for one transformation over another. For example, if precision is more critical than recall, the log transformation might be preferable despite its lower recall.

Continuous monitoring and validation on new data are essential to ensure the model's performance remains consistent over time.

Part 2:

Code:

```
import pandas as pd

import numpy as np

from sklearn import preprocessing as preproc

from sklearn.impute import KNNImputer, SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn import model_selection as modelsel

from sklearn import neural_network as ann

from sklearn.metrics import mean_squared_error, mean_absolute_error

import matplotlib.pyplot as plt

import time



def getData(path):

  data_frame = pd.read_excel(path)


  return data_frame



def doPreprocessing(data_frame):

  # 1. Separate features and target

  features = data_frame.drop(columns=['AGI'])  # Assuming 'AGI' is the target

  target = data_frame['AGI']
```

```
# 2. Remove ID columns
features = features.drop(columns=['HSUP_WGT', 'MARSUPWT', 'FSUP_WGT'])

remaining_features = features.columns.tolist()

binary_features = ['A_SEX', 'HAS_DIV'] # impute missing values by knn
ordinal_features = ['PEINUSYR'] # impute missing values by knn
categorical_features = ['PAW_YN', 'A_MARITL', 'PENATVTY'] # one hot encoding and impute missing values by knn

numeric_features = set(remaining_features) - set(binary_features) - set(ordinal_features) - set(categorical_features)

numeric_features = list(numeric_features)
# print(numeric_features)

# Preprocessing for numeric features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))
])

# Preprocessing for ordinal and binary features: KNN imputation
knn_transformer = Pipeline(steps=[
    ('imputer', KNNImputer())
])
```

```
# Preprocessing for categorical features: One-hot encoding followed by KNN imputation
categorical_transformer = Pipeline(steps=[
    ('onehot', preproc.OneHotEncoder(handle_unknown='ignore', sparse_output=False)),
    ('imputer', KNNImputer())
])

# Bundle preprocessing for numeric and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('knn_b', knn_transformer, binary_features),
        ('knn_o', knn_transformer, ordinal_features),
        ('cat', categorical_transformer, ['PAW_YN', 'A_MARITL', 'PENATVTY'])
    ])

# Create a preprocessing pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the features
features_processed = pipeline.fit_transform(features)

# Convert back into pandas DF
features_processed_df = pd.DataFrame(features_processed)

return features_processed_df, target
```

```
def doNormalize(X):
    scalerX = preproc.MinMaxScaler(feature_range=(-1,1))
    scalerX.fit(X)
    X_scaled = scalerX.transform(X)

    return X_scaled



def getMetrics(hl, clf, trainX, testX, trainY, testY):
        # i. Architecture
        print(f"\nArchitecture (hidden layer sizes): {hl}")

        # ii. Number of epochs
        print(f"Number of epochs: {clf.n_iter_}")

        # iii. Training set metrics
        train_score = clf.score(trainX, trainY)
        train_mse = mean_squared_error(trainY, clf.predict(trainX))
        train_mae = mean_absolute_error(trainY, clf.predict(trainX))
        print(f"Training Set - Coefficient of determination (R^2): {train_score:.4f}, MSE:
{train_mse:.4f}, MAE: {train_mae:.4f}")

        # iv. Test set metrics
```

```
        test_score = clf.score(testX, testY)
        test_mse = mean_squared_error(testY, clf.predict(testX))
        test_mae = mean_absolute_error(testY, clf.predict(testX))
        print(f"Test Set - Coefficient of determination (R^2): {test_score:.4f}, MSE: {test_mse:.4f}, MAE: {test_mae:.4f}")

        # v. Generalization gap (using R^2 for illustration)
        generalization_gap = train_score - test_score
        print(f"Generalization gap (R^2): {generalization_gap:.4f}\n")



def getPlot(hl, clf):
    trainingLoss = np.asarray(clf.loss_curve_)
    validation_loss = np.sqrt(1 - np.asarray(clf.validation_scores_))
    factor = trainingLoss[1] / validation_loss[1]
    validation_loss = validation_loss*factor

    # Plot setup
    xlabel = "epochs (hl=" + str(hl) + ")"
    fig, ax = plt.subplots()

    # Plot training loss on the primary y-axis
    ax.plot(trainingLoss, color="blue", label='Training Loss')
    ax.set_xlabel(xlabel, fontsize=10)
    ax.set_ylabel("Training Loss", color="blue", fontsize=10)
```

```
    # Create a secondary y-axis for validation loss
    ax2 = ax.twinx()
    ax2.plot(validation_loss, color="red", label='Validation Loss')
    ax2.set_ylabel("Validation Loss", color="red", fontsize=10)

    # Set y-axis scale
    ax.set_yscale('log')
    ax2.set_yscale('log')

    # Show plot with proper layout
    fig.tight_layout()
    plt.show()


def doANNRegression(feature, target):
    trainX, testX, trainY, testY = modelsel.train_test_split(feature, target, test_size=0.3, random_state=241)

    hidden_layers =[(4,4), (10,6), (32,16), (8,3,5), (12,9,10)]

    for hl in hidden_layers:
        clf = ann.MLPRegressor(hidden_layer_sizes=hl, activation='relu', early_stopping=True, tol=0.0005, alpha=0.0001, max_iter=1000)
        clf.fit(trainX, trainY)
        getMetrics(hl, clf, trainX, testX, trainY, testY)
        getPlot(hl, clf)
```

```
if __name__ == '__main__':
    start = time.time()
    df = getData('Census_Supplement_Data.xlsx')
    end = time.time()
    print('Time taken to read excel file: {:.4f} seconds'.format(end - start))
    # print(f'\nData:\n{df.head(10)}')

    # Preprocessing
    X, Y = doPreprocessing(df)
    # print(type(X), type(Y))

    # Normalization
    X_norm = doNormalize(X)
    # print(type(X_norm))

    # ANN
    doANNRegression(X_norm, Y)
```

Output:

```
PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment_4> & C:/Users/srico/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/srico/OneDrive/Desktop/Applications of Machine Learning/Assignment_4/Part2.py"


Architecture (hidden layer sizes): (4, 4)
```

Number of epochs: 783

Training Set - Coefficient of determination (R^2): 0.7239, MSE: 2370905320.3842, MAE: 24599.8883

Test Set - Coefficient of determination (R^2): 0.7034, MSE: 2407847888.0077, MAE: 25009.1249

Generalization gap (R^2): 0.0205

Architecture (hidden layer sizes): (10, 6)

Number of epochs: 563

Training Set - Coefficient of determination (R^2): 0.7345, MSE: 2280056266.9017, MAE: 24134.2532

Test Set - Coefficient of determination (R^2): 0.7137, MSE: 2324753134.1979, MAE: 24534.0171

Generalization gap (R^2): 0.0208

Architecture (hidden layer sizes): (32, 16)

Number of epochs: 311

Training Set - Coefficient of determination (R^2): 0.7141, MSE: 2455452077.1516, MAE: 25225.3384

Test Set - Coefficient of determination (R^2): 0.6906, MSE: 2511714570.3482, MAE: 25674.9411

Generalization gap (R^2): 0.0234

Architecture (hidden layer sizes): (8, 3, 5)

Number of epochs: 12

Training Set - Coefficient of determination (R^2): -0.1765, MSE: 10103426843.5239, MAE: 38948.5744

Test Set - Coefficient of determination (R^2): -0.1924, MSE: 9680795223.1014, MAE: 39532.8486

Generalization gap (R^2): 0.0158

Architecture (hidden layer sizes): (12, 9, 10)

Number of epochs: 319

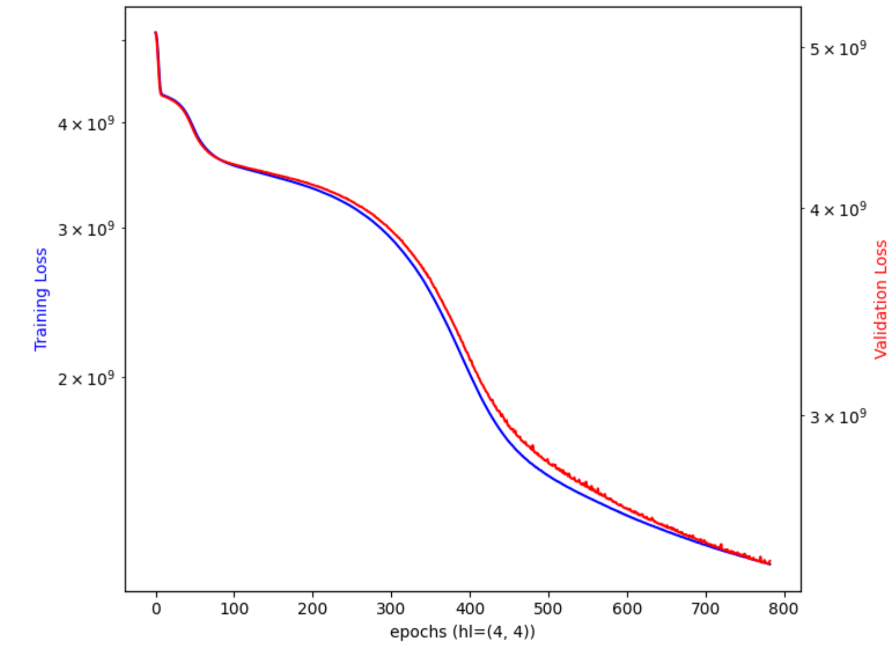
Training Set - Coefficient of determination (R^2): 0.7703, MSE: 1972277394.6740, MAE: 21667.1000

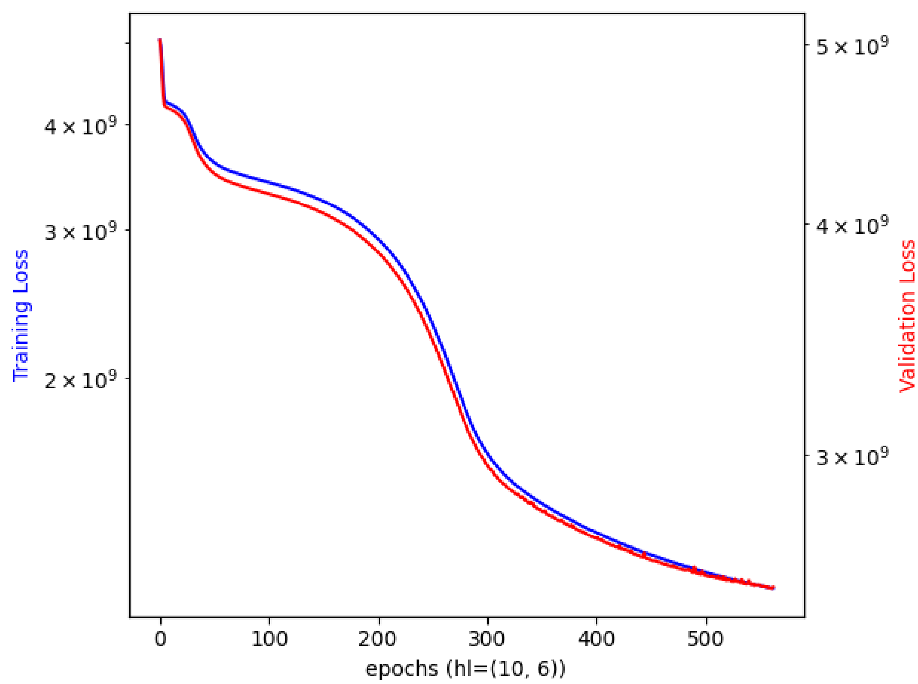
Test Set - Coefficient of determination (R^2): 0.7541, MSE: 1996162442.3780, MAE: 22004.0245

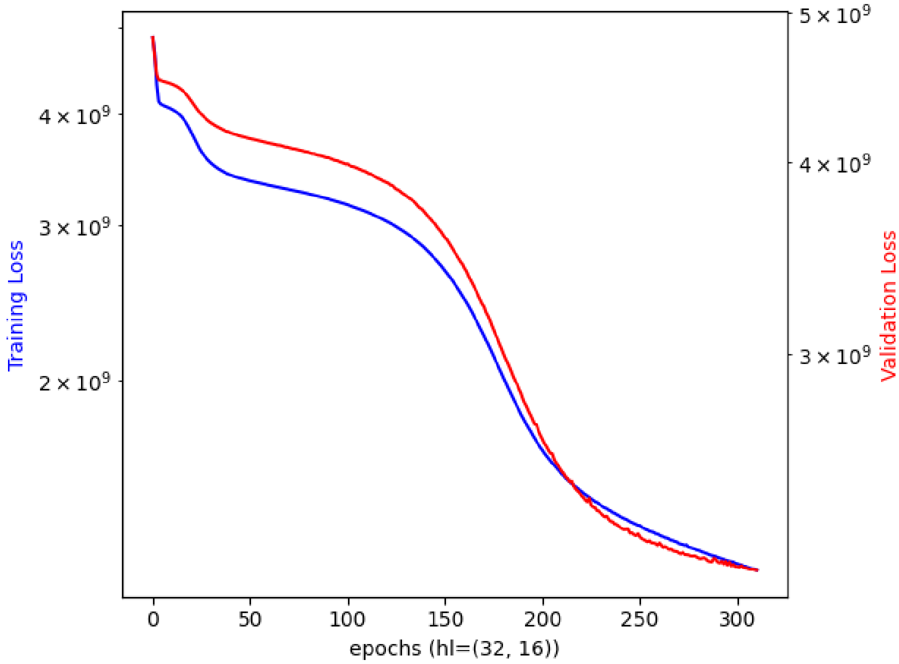
Generalization gap (R^2): 0.0162

PS C:\Users\srico\OneDrive\Desktop\Applications of Machine Learning\Assignment_4>

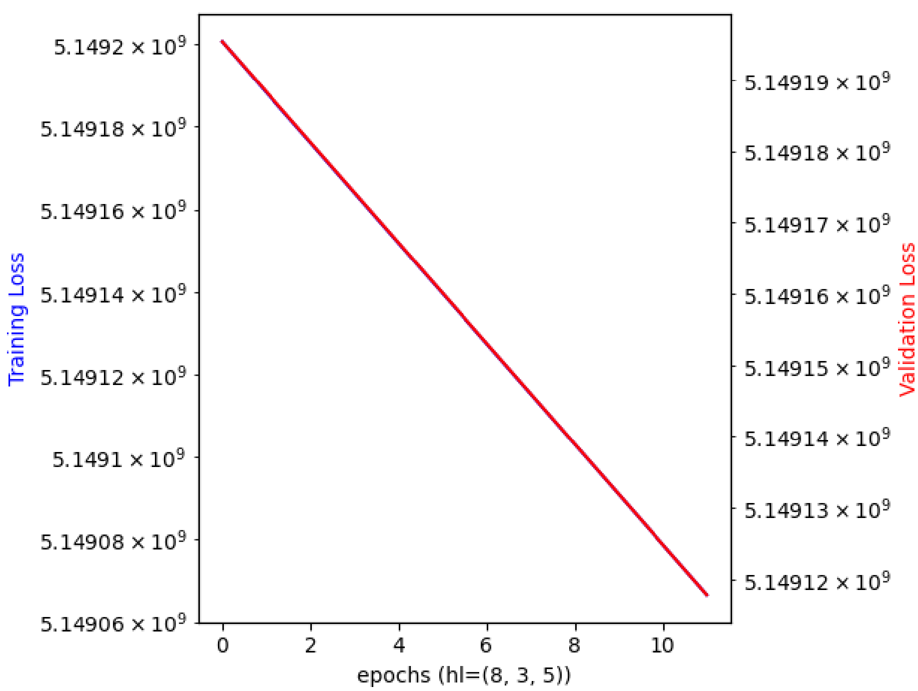
Validation Loss and Training Loss vs Epoch Plots:



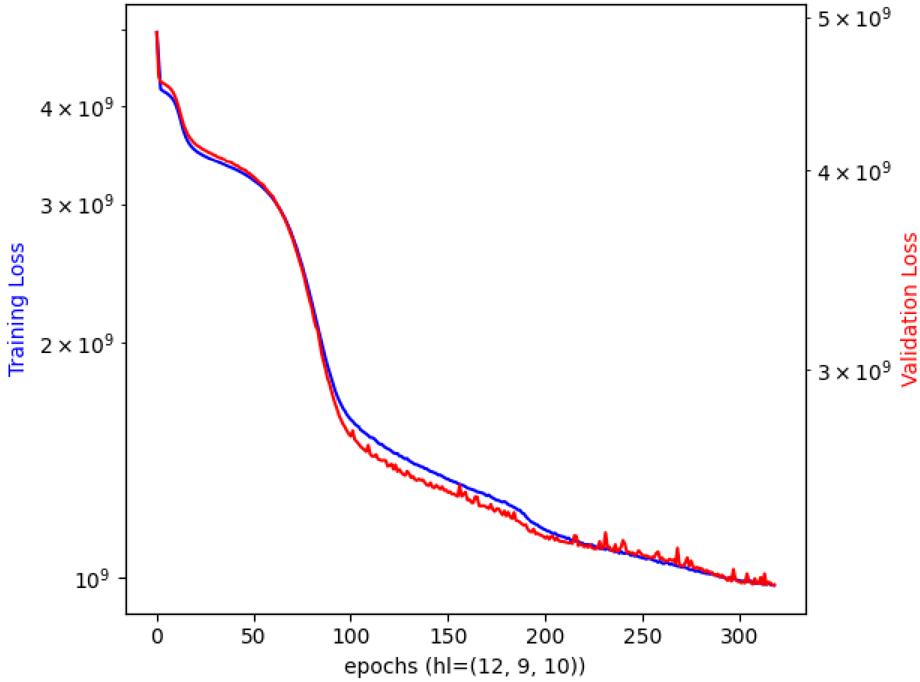




5×10^9
4×10^9
3×10^9
2×10^9
4×10^9
3×10^9
Training Loss
Validation Loss
0
50
100
150
200
250
300
epochs (hl=(32, 16))



5.1492×10^9
5.14918×10^9
5.14916×10^9
5.14914×10^9
5.14912×10^9
5.1491×10^9
5.14908×10^9
5.14906×10^9
5.14919×10^9
5.14918×10^9
5.14917×10^9
5.14916×10^9
5.14915×10^9
5.14914×10^9
5.14913×10^9
5.14912×10^9
Training Loss
Validation Loss
0
2
4
6
8
10
epochs (hl=(8, 3, 5))



Discussion:

Analyzing the performance of various hidden layer architectures in an artificial neural network (ANN) regressor reveals a spectrum of outcomes. The simplest (4, 4) architecture displayed respectable scores with a moderate gap between training and testing. Expanding to a (10, 6) configuration slightly enhanced performance but also slightly widened the generalization gap. Surprisingly, the more complex (32, 16) setup did not yield better results, showing lower effectiveness and the largest gap in generalization, which could suggest overfitting. On the other hand, the (8, 3, 5) structure significantly underperformed, with negative R^2 scores indicating a possible mismatch for the problem at hand. Conversely, the (12, 9, 10) architecture emerged as the top performer, achieving the highest R^2 values with a minimal generalization gap, suggesting it as the most apt model for capturing and predicting the data patterns effectively among the tested configurations.