Lesson 9: MLP

We'll cover:

1. **Recap of the Original MLP Code**
2. **Topic 1: Hyperparameter Tuning with GridSearchCV**
3. **Topic 2: Early Stopping**
4. **Topic 3: Baseline Comparison with Linear Regression**
5. **Topic 4: Cross-Validation**

Each section is accompanied by code snippets and detailed explanations.

---

1. Recap of the Original MLP Code

Let's start with the code you already taught. This code loads a housing dataset, scales the features using MinMaxScaler, trains an MLPRegressor with two hidden layers of 50 neurons each (using ReLU activation), and then evaluates the model.

```
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Load Dataset
data = pd.read_csv('Housing.csv')

X = data[['Income', 'Age', 'Rooms', 'Bedrooms', 'Population']]
y = data['Price']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

# Min-Max Scaling to range [0,1] - Fit on training set and transform both
scaler = MinMaxScaler(feature_range=(0, 1))
X_train = scaler.fit_transform(X_train)  # Fit and transform training data
X_test = scaler.transform(X_test)        # Transform test data (use same scaler)

# Define MLP Regressor model
mlp = MLPRegressor(hidden_layer_sizes=(50, 50), activation='relu',
max_iter=2000, random_state=42)

# Train the model
mlp.fit(X_train, y_train)

# Predictions
y_train_pred = mlp.predict(X_train)
y_test_pred = mlp.predict(X_test)

# Evaluation Metrics function
```

```
def evaluate_model(y_true, y_pred, dataset_type="Train"):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred) * 100

    print(f"{dataset_type} Set Metrics:")
    print(f"  MSE: {mse:.4f}")
    print(f"  MAE: {mae:.4f}")
    print(f"  R² Score: {r2:.4f}\n")

# Compute metrics for training and testing sets
evaluate_model(y_train, y_train_pred, "Train")
evaluate_model(y_test, y_test_pred, "Test")
```

---

2. Topic 1: Hyperparameter Tuning with GridSearchCV

Hyperparameter tuning allows us to search for the best set of parameters (such as hidden layer sizes, regularization strength `alpha`, learning rate, and solver) that improve model performance.

Extended Code:

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'hidden_layer_sizes': [(50, 50), (100,), (50, 25, 25)],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate_init': [0.001, 0.01],
    'solver': ['adam', 'lbfgs'],
    'early_stopping': [False]  # We'll discuss early stopping separately in the
next section.
}

# Create a base MLPRegressor model (without early stopping here)
base_mlp = MLPRegressor(max_iter=2000, random_state=42)

# Setup GridSearchCV to search over the parameter grid using 3-fold cross-
validation
grid_search = GridSearchCV(estimator=base_mlp, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=3, n_jobs=-1)

# Perform grid search on the training data
grid_search.fit(X_train, y_train)
print("Best parameters from GridSearchCV:")
print(grid_search.best_params_)

# Use the best estimator found
best_mlp = grid_search.best_estimator_

# Predictions and evaluation for the best model
y_train_pred_best = best_mlp.predict(X_train)
y_test_pred_best = best_mlp.predict(X_test)

evaluate_model(y_train, y_train_pred_best, "Train (Tuned)")
evaluate_model(y_test, y_test_pred_best, "Test (Tuned)")
```

Explanation:

- **Parameter Grid:**
  We set up a dictionary (`param_grid`) with different options for the hidden layer configuration, regularization strength (`alpha`), initial learning rate, and the solver.
- **GridSearchCV:**
  This tool automatically runs several experiments over the parameter grid using cross-validation (here, 3-fold) to find the best parameter combination.
- **Best Estimator:**
  Once grid search is complete, `grid_search.best_estimator_` gives you the tuned MLP model. We then evaluate its performance on both the training and testing sets.

---

3. Topic 2: Early Stopping

Early stopping is a regularization technique used to avoid overfitting by stopping the training process when the validation performance stops improving.

Extended Code (with Early Stopping):

```
# Define a new MLPRegressor with early stopping enabled.
mlp_es = MLPRegressor(hidden_layer_sizes=(50, 50), activation='relu',
                      max_iter=2000, random_state=42, early_stopping=True,
                      validation_fraction=0.1, n_iter_no_change=10)

# Train the model with early stopping
mlp_es.fit(X_train, y_train)

# Predictions with the early-stopped model
y_train_pred_es = mlp_es.predict(X_train)
y_test_pred_es = mlp_es.predict(X_test)

# Evaluate the early stopping model
evaluate_model(y_train, y_train_pred_es, "Train (Early Stopping)")
evaluate_model(y_test, y_test_pred_es, "Test (Early Stopping)")
```

Explanation:

- **early_stopping=True:**
  This tells the MLP to monitor the validation score and stop training when no significant improvement is observed.
- **validation_fraction:**
  Specifies the portion of training data used as the validation set (here, 10%).
- **n_iter_no_change:**
  The training stops if the validation score does not improve for 10 consecutive epochs.

By comparing metrics, you can see if early stopping improves generalization on the test set.

---

4. Topic 3: Baseline Comparison with Linear Regression

Before committing to a more complex model like an MLP, it's good practice to compare it with a simpler baseline model such as Linear Regression.

Extended Code:

```
# Define and train a baseline Linear Regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions using the Linear Regression model
y_train_pred_lr = lr.predict(X_train)
y_test_pred_lr = lr.predict(X_test)

# Evaluate the Linear Regression model
evaluate_model(y_train, y_train_pred_lr, "Train (Linear Regression)")
evaluate_model(y_test, y_test_pred_lr, "Test (Linear Regression)")
```

Explanation:

- **Baseline Model:**
  Linear Regression is simpler and faster to train. Comparing its performance to the MLP helps you decide if the added complexity of a neural network is justified.
- **Evaluation:**
  Use the same evaluation function to compare metrics (MSE, MAE, R^2) across models.

---

5. Topic 4: Cross-Validation

Beyond using GridSearchCV's internal cross-validation, you can explicitly perform cross-validation to get a more reliable estimate of your model's performance.

Extended Code:

```
from sklearn.model_selection import cross_val_score

# Use the best tuned MLPRegressor for cross-validation
cv_scores = cross_val_score(best_mlp, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')

# Convert negative MSE scores to positive and calculate mean MSE
mse_scores = -cv_scores
mean_mse = mse_scores.mean()

print("Cross-Validation MSE Scores:", mse_scores)
```

```
print("Mean Cross-Validation MSE:", mean_mse)
```

Explanation:

- **cross_val_score:**
  This function performs 5-fold cross-validation on the training set using the best tuned MLP model.
- **Scoring:**
  We use negative MSE because higher scores are better for cross_val_score by default; then we convert to positive MSE values.
- **Mean MSE:**
  The average MSE across the folds gives a robust estimate of model performance.

Wrap-Up

By extending the original code with the topics above, you now have multiple techniques to potentially improve your MLP model's performance:

- **Hyperparameter Tuning:** Automates the search for optimal settings.
- **Early Stopping:** Helps prevent overfitting by monitoring validation performance.
- **Baseline Comparison:** Ensures the MLP provides a significant improvement over simpler models.
- **Cross-Validation:** Provides a robust estimate of your model's performance.

These topics not only enhance the MLP's performance but also build good practices in model evaluation and tuning. Feel free to experiment with different parameter settings and compare the results to see how each change affects your model's predictions.