homework6

November 18, 2025

1 ECGR 4105-001, Homework 6

1.1 By Joshua Foster, 801268119

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

Problem 1:

Build a fully connected neural network with multiple hidden layers for the diabetes dataset, which you have done before. The diabetes dataset is available on Canvas. Use as may hidden layers and neurons that you feel proper for this problem. Train it and plot your loss for both training set and validations set, use 20%, and 80% split between your training and validation set. Plot your results and also report your accuracy, precision, recall and F1 score. Compare your results against logistic dataset and support vector machine.

[5]:
```
try:
    df = pd.read_csv('diabetes.csv')
    print("Dataset loaded successfully.")
    print("\nFirst 5 rows:")
    print(df.head())
    print("\nDataset Info:")
    df.info()
except FileNotFoundError:
    print("Error: 'diabetes.csv' not found. Please make sure the file is in the␣
 ↪same directory.")
```

```
Dataset loaded successfully.

First 5 rows:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[6]:
```
X = df.drop('Outcome', axis=1)
y = df['Outcome']

print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")
```

```
Features shape: (768, 8)
Target shape: (768,)
```

[7]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,  # 20% for the test/validation set
    random_state=42,
    stratify=y
)

print(f"Training samples: {X_train.shape[0]}")
print(f"Testing samples: {X_test.shape[0]}")
```

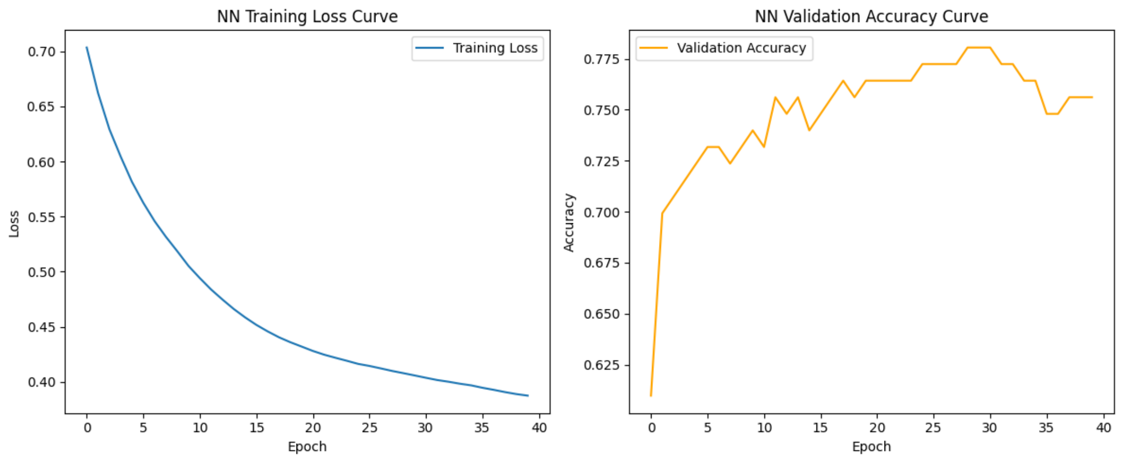
```
Training samples: 614
Testing samples: 154
```

[8]:
```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[9]:
```
mlp = MLPClassifier(
    hidden_layer_sizes=(100, 50), # Two hidden layers
    max_iter=500,                 # Max epochs
    early_stopping=True,          # Stops training when validation score␣
  ↪doesn't improve
    validation_fraction=0.2,      # Uses 20% of *training* data as an internal␣
  ↪validation set
    random_state=42,
    solver='adam',
    learning_rate_init=0.001
)

print("Training Neural Network...")
mlp.fit(X_train_scaled, y_train)
print("Training complete.")
```

```
Training Neural Network…
Training complete.
```

[10]:
```
# Plot Neural Network Loss and Validation Score

plt.figure(figsize=(12, 5))

# Plot Training Loss
plt.subplot(1, 2, 1)
plt.plot(mlp.loss_curve_, label="Training Loss")
plt.title("NN Training Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

# Plot Validation Accuracy
plt.subplot(1, 2, 2)
if hasattr(mlp, 'validation_scores_'):
    plt.plot(mlp.validation_scores_, label="Validation Accuracy",␣
 ↪color='orange')
    plt.title("NN Validation Accuracy Curve")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
else:
    plt.text(0.5, 0.5, 'No validation scores recorded',
             horizontalalignment='center', verticalalignment='center',
             transform=plt.gca().transAxes)

plt.tight_layout()
plt.show()
```



[11]:
```
print("--- Neural Network Evaluation (on 20% Test Set) ---")
y_pred_nn = mlp.predict(X_test_scaled)

nn_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_nn),
    "Precision": precision_score(y_test, y_pred_nn),
    "Recall": recall_score(y_test, y_pred_nn),
    "F1 Score": f1_score(y_test, y_pred_nn)
}
```

```python
print("Neural Network Classification Report:")
print(classification_report(y_test, y_pred_nn))

print("Neural Network Metrics:")
print(nn_metrics)
```

```
--- Neural Network Evaluation (on 20% Test Set) ---
Neural Network Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       100
           1       0.63      0.59      0.61        54

    accuracy                           0.73       154
   macro avg       0.71      0.70      0.70       154
weighted avg       0.73      0.73      0.73       154

Neural Network Metrics:
{'Accuracy': 0.7337662337662337, 'Precision': 0.6274509803921569, 'Recall':
0.5925925925925926, 'F1 Score': 0.6095238095238096}
```

[12]:
```python
# Logistic Regression

print("--- Logistic Regression Evaluation (on 20% Test Set) ---")
lr = LogisticRegression(random_state=42, max_iter=200)
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

lr_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_lr),
    "Precision": precision_score(y_test, y_pred_lr),
    "Recall": recall_score(y_test, y_pred_lr),
    "F1 Score": f1_score(y_test, y_pred_lr)
}

print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr))
```

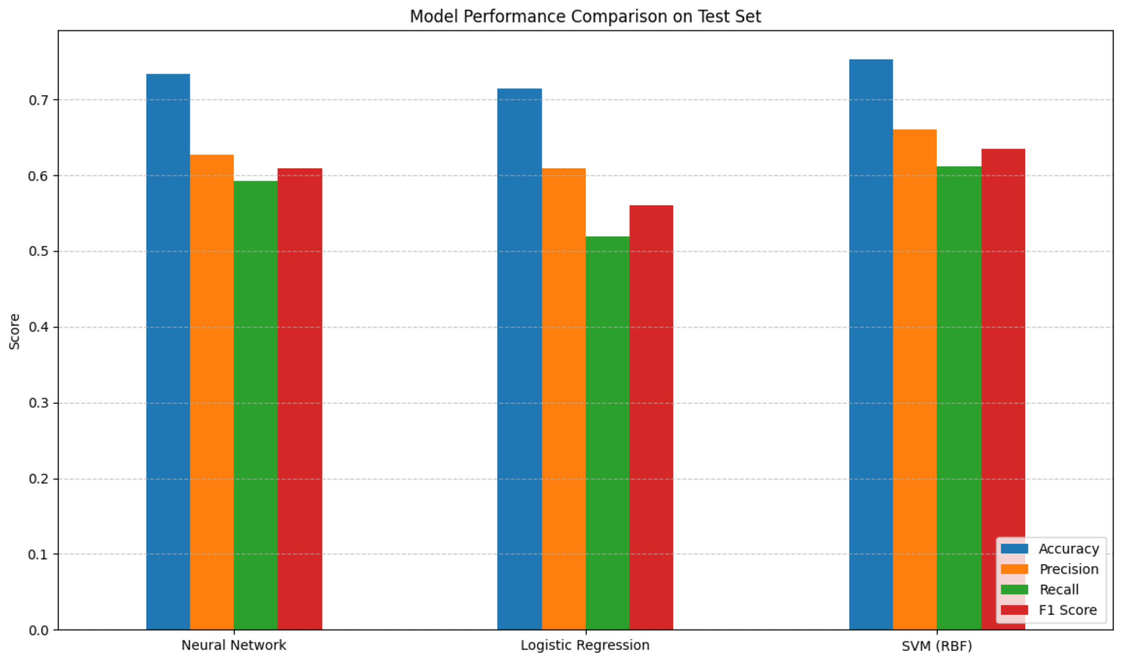
```
--- Logistic Regression Evaluation (on 20% Test Set) ---
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.82      0.79       100
           1       0.61      0.52      0.56        54

    accuracy                           0.71       154
   macro avg       0.68      0.67      0.67       154
weighted avg       0.71      0.71      0.71       154
```

[13]:
```
# Support Vector Machine

print("--- SVM (RBF Kernel) Evaluation (on 20% Test Set) ---")
svc_rbf = svm.SVC(kernel='rbf', random_state=42)
svc_rbf.fit(X_train_scaled, y_train)
y_pred_svm = svc_rbf.predict(X_test_scaled)

svm_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_svm),
    "Precision": precision_score(y_test, y_pred_svm),
    "Recall": recall_score(y_test, y_pred_svm),
    "F1 Score": f1_score(y_test, y_pred_svm)
}

print("SVM (RBF Kernel) Classification Report:")
print(classification_report(y_test, y_pred_svm))
```

```
--- SVM (RBF Kernel) Evaluation (on 20% Test Set) ---
SVM (RBF Kernel) Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       100
           1       0.66      0.61      0.63        54

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154
```

[14]:
```
# Final Model Comparison

results_df = pd.DataFrame([nn_metrics, lr_metrics, svm_metrics],
                              index=['Neural Network', 'Logistic Regression',␣
 ↪'SVM (RBF)'])

print("\n--- Model Comparison Summary ---")
print(results_df)

# Plot the comparison
results_df.plot(kind='bar', figsize=(14, 8))
plt.title('Model Performance Comparison on Test Set')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
--- Model Comparison Summary ---
                     Accuracy  Precision    Recall  F1 Score
Neural Network       0.733766   0.627451  0.592593  0.609524
Logistic Regression  0.714286   0.608696  0.518519  0.560000
SVM (RBF)            0.753247   0.660000  0.611111  0.634615
```



Problem 2:

Build a fully connected neural network with multiple hidden layers for the Cancer dataset, which you have done before. The diabetes dataset is available on Canvas. Use as may hidden layers and neurons that you feel proper for this problem. Train it and plot your loss for both training set and validations set, use 20%, and 80% split between your training and validation set. Plot your results and report your accuracy, precision, recall and F1 score. Compare your results against logistic dataset and support vector machine.

[15]:
```python
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

print("Dataset loaded successfully.")
print(f"Number of samples: {cancer.data.shape[0]}")
print(f"Number of features: {cancer.data.shape[1]}")

print("\nTarget names (classes):")
print(cancer.target_names) # [0 = 'malignant', 1 = 'benign']
print("\nFirst 5 feature names:")
print(cancer.feature_names[:5])
```

```
Dataset loaded successfully.
Number of samples: 569
Number of features: 30

Target names (classes):
['malignant' 'benign']

First 5 feature names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness']
```

[16]:
```
# The 'data' attribute holds all 30 features
X = cancer.data
# The 'target' attribute holds the class (0 or 1)
y = cancer.target

print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")
```

```
Features shape: (569, 30)
Target shape: (569,)
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,  # 20% for the test/validation set
    random_state=42,
    stratify=y
)

print(f"Training samples: {X_train.shape[0]}")
print(f"Testing samples: {X_test.shape[0]}")
```

```
Training samples: 455
Testing samples: 114
```

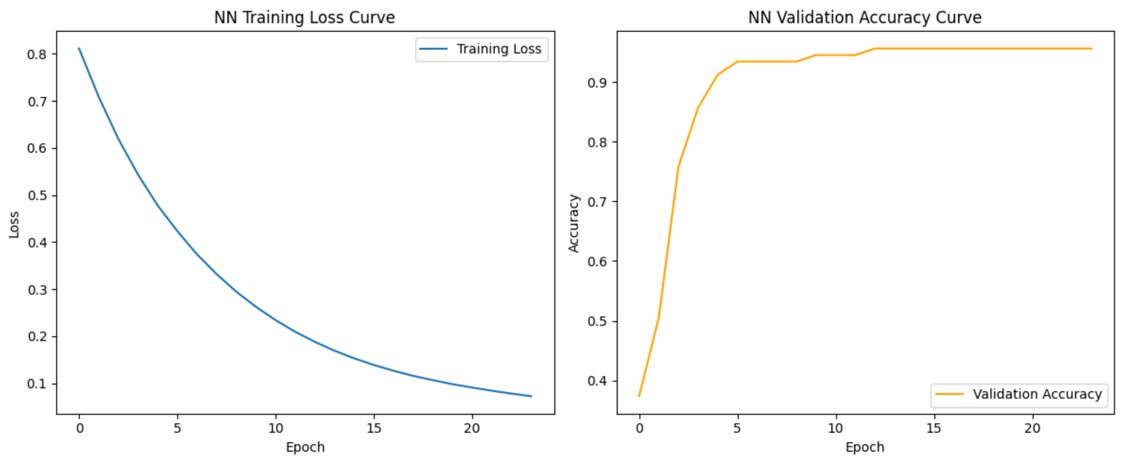
[18]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Data has been scaled.")
```

```
Data has been scaled.
```

[19]:
```
mlp = MLPClassifier(
    hidden_layer_sizes=(100, 50),
    max_iter=500,
    early_stopping=True,
    validation_fraction=0.2,      # Use 20% of training data for validation
    random_state=42,
    solver='adam',
    learning_rate_init=0.001
)

print("Training Neural Network...")
mlp.fit(X_train_scaled, y_train)
print("Training complete.")
```

```
Training Neural Network…
Training complete.
```

[20]:
```
plt.figure(figsize=(12, 5))

# Plot Training Loss
plt.subplot(1, 2, 1)
plt.plot(mlp.loss_curve_, label="Training Loss")
plt.title("NN Training Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

# Plot Validation Accuracy
plt.subplot(1, 2, 2)
if mlp.validation_scores_ is not None:
    plt.plot(mlp.validation_scores_, label="Validation Accuracy",␣
 ↪color='orange')
    plt.title("NN Validation Accuracy Curve")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
else:
    plt.text(0.5, 0.5, 'No validation scores recorded',
             horizontalalignment='center', verticalalignment='center',
             transform=plt.gca().transAxes,
             bbox=dict(boxstyle="round,pad=0.3", edgecolor="gray",␣
 ↪facecolor="aliceblue"))

plt.tight_layout()
plt.show()
```



[21]:
```
print("--- Neural Network Evaluation (on 20% Test Set) ---")
y_pred_nn = mlp.predict(X_test_scaled)

# Store metrics in a dictionary for later comparison
nn_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_nn),
    "Precision": precision_score(y_test, y_pred_nn),
    "Recall": recall_score(y_test, y_pred_nn),
    "F1 Score": f1_score(y_test, y_pred_nn)
}

print("Neural Network Classification Report:")
# Note: 0 = malignant, 1 = benign
print(classification_report(y_test, y_pred_nn, target_names=cancer.
 ↪target_names))

print("Neural Network Metrics (for class 1, 'benign'):")
print(nn_metrics)
```

```
--- Neural Network Evaluation (on 20% Test Set) ---
Neural Network Classification Report:
              precision    recall  f1-score   support

   malignant       0.93      0.88      0.90        42
      benign       0.93      0.96      0.95        72

    accuracy                           0.93       114
   macro avg       0.93      0.92      0.92       114
weighted avg       0.93      0.93      0.93       114

Neural Network Metrics (for class 1, 'benign'):
```

```
{'Accuracy': 0.9298245614035088, 'Precision': 0.9324324324324325, 'Recall':
0.9583333333333334, 'F1 Score': 0.9452054794520548}
```

[22]:
```
print("--- Logistic Regression Evaluation (on 20% Test Set) ---")
lr = LogisticRegression(random_state=42, max_iter=200)
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

lr_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_lr),
    "Precision": precision_score(y_test, y_pred_lr),
    "Recall": recall_score(y_test, y_pred_lr),
    "F1 Score": f1_score(y_test, y_pred_lr)
}

print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=cancer.
 ↪target_names))
```

```
--- Logistic Regression Evaluation (on 20% Test Set) ---
Logistic Regression Classification Report:
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        42
      benign       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

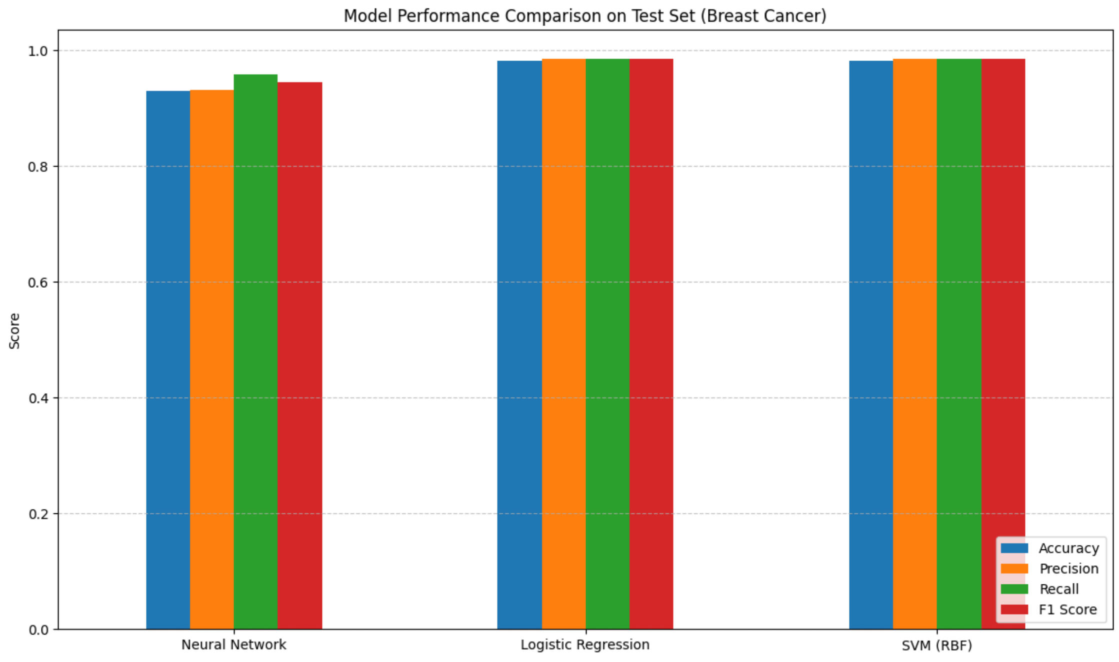
[23]:
```
print("--- SVM (RBF Kernel) Evaluation (on 20% Test Set) ---")
svc_rbf = svm.SVC(kernel='rbf', random_state=42)
svc_rbf.fit(X_train_scaled, y_train)
y_pred_svm = svc_rbf.predict(X_test_scaled)

svm_metrics = {
    "Accuracy": accuracy_score(y_test, y_pred_svm),
    "Precision": precision_score(y_test, y_pred_svm),
    "Recall": recall_score(y_test, y_pred_svm),
    "F1 Score": f1_score(y_test, y_pred_svm)
}

print("SVM (RBF Kernel) Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=cancer.
 ↪target_names))
```

```
--- SVM (RBF Kernel) Evaluation (on 20% Test Set) ---
SVM (RBF Kernel) Classification Report:
```

```
               precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        42
      benign       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

[24]:
```
results_df = pd.DataFrame([nn_metrics, lr_metrics, svm_metrics],
                              index=['Neural Network', 'Logistic Regression',␣
 ↪'SVM (RBF)'])

print("\n--- Model Comparison Summary ---")
print(results_df)

# Plot the comparison
results_df.plot(kind='bar', figsize=(14, 8))
plt.title('Model Performance Comparison on Test Set (Breast Cancer)')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(loc='lower right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
--- Model Comparison Summary ---
                     Accuracy  Precision    Recall  F1 Score
Neural Network       0.929825   0.932432  0.958333  0.945205
Logistic Regression  0.982456   0.986111  0.986111  0.986111
SVM (RBF)            0.982456   0.986111  0.986111  0.986111
```



Problem 3 (50 pts):

a. Create a fully connected Neural Network for all 10 classes in CIFAR-10 with only one hidden layer with the size of 512. Train your network (Do as many epochs as you need). Report your training time, training loss and evaluation accuracy after each epoch. Analyze your results in your report. Make sure to submit your code by providing the GitHub URL of your course repository for this course.

[25]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
```

[31]:
```
# Model Architecture
input_size = 3072
hidden_size = 512
num_classes = 10

# Training Settings
num_epochs = 20
batch_size = 100
learning_rate = 0.001

# Device Setup
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

[32]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_dataset = torchvision.datasets.CIFAR10(root='./data',
                                           train=True,
                                           download=True,
                                           transform=transform)

test_dataset = torchvision.datasets.CIFAR10(root='./data',
                                          train=False,
                                          download=True,
                                          transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)

# Define the 10 classes for reference
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')
```

[33]:
```
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(MLP, self).__init__()

        # This layer "flattens" the 3D image (32x32x3) into a 1D vector (3072)
        self.flatten = nn.Flatten()

        # This is our fully connected (Linear) network structure
        self.network = nn.Sequential(
            # The input layer (3072) to our one hidden layer (512)
            nn.Linear(input_size, hidden_size),
            # ReLU activation
            nn.ReLU(),
            # The hidden layer (512) to the output layer (10)
            nn.Linear(hidden_size, num_classes)
        )
    def forward(self, x):
        x = self.flatten(x)
        x = self.network(x)
        return x

# Create an instance of the model and send it to our device (GPU or CPU)
model = MLP(input_size, hidden_size, num_classes).to(device)

print("Model Definition (MLP):")
print(model)
```

```
Model Definition (MLP):
MLP(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (network): Sequential(
    (0): Linear(in_features=3072, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

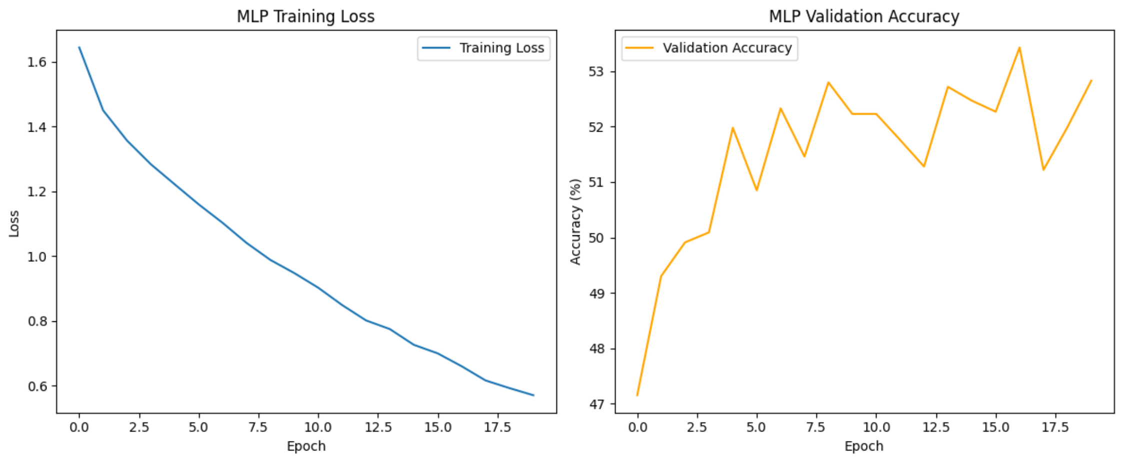
[34]:
```
# Loss Function: CrossEntropyLoss
criterion = nn.CrossEntropyLoss()

# Optimizer: Adam
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Calculate Model Size
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

total_params_mlp = count_parameters(model)
print(f"\nTotal Trainable Parameters (MLP): {total_params_mlp:,}")
```

```

Total Trainable Parameters (MLP): 1,578,506
You will use this number to compare against your CNN in Part (b).
```

[35]:
```
print("--- Starting Training (MLP Baseline) ---")

n_total_steps = len(train_loader) # Number of batches in one epoch
all_train_loss = []
all_eval_acc = []

for epoch in range(num_epochs):
    # TRAINING
    model.train() # Set the model to training mode
    start_time = time.time()
    running_train_loss = 0.0

    for i, (images, labels) in enumerate(train_loader):
        # Move images and labels to the device (GPU/CPU)
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass: Feed images to the model
        outputs = model(images)

        # Calculate the loss
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad() # Clear old gradients
        loss.backward()       # Calculate new gradients
        optimizer.step()      # Update model weights

        running_train_loss += loss.item() # Accumulate the loss

    # VALIDATION (EVALUATION)
    model.eval() # Set the model to evaluation mode
    correct = 0
    total = 0

    # We don't need to calculate gradients during validation
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            # Get model outputs
            outputs = model(images)

            # Get the prediction: the class with the highest score
            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0) # Add batch size to total
            correct += (predicted == labels).sum().item() # Count correct␣
↪predictions

    # PER-EPOCH REPORT
    epoch_time = time.time() - start_time
    avg_train_loss = running_train_loss / n_total_steps
    eval_accuracy = 100 * correct / total

    all_train_loss.append(avg_train_loss)
    all_eval_acc.append(eval_accuracy)

    print(f"Epoch [{epoch+1}/{num_epochs}] | "
          f"Time: {epoch_time:.2f}s | "
          f"Train Loss: {avg_train_loss:.4f} | "
          f"Evaluation Accuracy: {eval_accuracy:.2f}%")

print("--- Finished Training (MLP) ---")
```

```
--- Starting Training (MLP Baseline) ---
Epoch [1/20] | Time: 14.99s | Train Loss: 1.6438 | Evaluation Accuracy: 47.15%
Epoch [2/20] | Time: 14.59s | Train Loss: 1.4497 | Evaluation Accuracy: 49.30%
Epoch [3/20] | Time: 14.55s | Train Loss: 1.3566 | Evaluation Accuracy: 49.91%
Epoch [4/20] | Time: 14.87s | Train Loss: 1.2833 | Evaluation Accuracy: 50.09%
Epoch [5/20] | Time: 15.46s | Train Loss: 1.2213 | Evaluation Accuracy: 51.98%
Epoch [6/20] | Time: 14.77s | Train Loss: 1.1598 | Evaluation Accuracy: 50.85%
Epoch [7/20] | Time: 14.67s | Train Loss: 1.1028 | Evaluation Accuracy: 52.33%
Epoch [8/20] | Time: 14.59s | Train Loss: 1.0409 | Evaluation Accuracy: 51.46%
Epoch [9/20] | Time: 14.61s | Train Loss: 0.9884 | Evaluation Accuracy: 52.80%
Epoch [10/20] | Time: 15.24s | Train Loss: 0.9478 | Evaluation Accuracy: 52.23%
Epoch [11/20] | Time: 14.74s | Train Loss: 0.9028 | Evaluation Accuracy: 52.23%
Epoch [12/20] | Time: 14.67s | Train Loss: 0.8493 | Evaluation Accuracy: 51.76%
Epoch [13/20] | Time: 14.76s | Train Loss: 0.8019 | Evaluation Accuracy: 51.28%
Epoch [14/20] | Time: 14.87s | Train Loss: 0.7750 | Evaluation Accuracy: 52.72%
Epoch [15/20] | Time: 15.38s | Train Loss: 0.7266 | Evaluation Accuracy: 52.47%
Epoch [16/20] | Time: 14.76s | Train Loss: 0.7002 | Evaluation Accuracy: 52.27%
Epoch [17/20] | Time: 14.79s | Train Loss: 0.6605 | Evaluation Accuracy: 53.43%
Epoch [18/20] | Time: 14.81s | Train Loss: 0.6167 | Evaluation Accuracy: 51.22%
Epoch [19/20] | Time: 15.23s | Train Loss: 0.5930 | Evaluation Accuracy: 51.99%
Epoch [20/20] | Time: 15.03s | Train Loss: 0.5711 | Evaluation Accuracy: 52.83%
--- Finished Training (MLP) ---
```

[36]:
```
plt.figure(figsize=(12, 5))

# Plot Training Loss
plt.subplot(1, 2, 1)
plt.plot(all_train_loss, label="Training Loss")
plt.title("MLP Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

# Plot Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(all_eval_acc, label="Validation Accuracy", color='orange')
plt.title("MLP Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.legend()

plt.tight_layout()
plt.show()
```



b. Extend your network with two more additional hidden layers, like the example we did in lecture. Train your network for 300 epochs. Report your training time, loss, and evaluation accuracy after 300 epochs. Analyze your results in your report and compare your model size and accuracy over the baseline implementation in Problem1. a. Do you see any over-fitting? Make sure to submit your code by providing the GitHub URL of your course repository for this course.

[37]:
```
# Training Settings
num_epochs = 300
batch_size = 100
learning_rate = 0.001

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

[38]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Download the training dataset (50,000 images)
train_dataset = torchvision.datasets.CIFAR10(root='./data',
                                             train=True,
                                             download=True,
                                             transform=transform)
```

```
# Download the test (evaluation) dataset (10,000 images)
test_dataset = torchvision.datasets.CIFAR10(root='./data',
                                            train=False,
                                            download=True,
                                            transform=transform)

# Create 'DataLoaders'
train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)

# Define the 10 classes for reference
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')
```

[39]:
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # Convolutional Layers
        self.conv_layer1 = nn.Sequential(
            # Input: 3 channels (RGB), Output: 16 filters
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2) # 32x32 -> 16x16
        )
        self.conv_layer2 = nn.Sequential(
            # Input: 16 filters, Output: 32 filters
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5,␣
 ↪padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2) # 16x16 -> 8x8
        )

        # Fully Connected Layers
        # Flattened size = 32 (filters) * 8 * 8 (image size) = 2048
        self.fc_layer1 = nn.Sequential(
            nn.Linear(32 * 8 * 8, 512),
            nn.ReLU()
        )
        # This is the final output layer
        self.fc_layer2 = nn.Linear(512, 10)

    def forward(self, x):
        x = self.conv_layer1(x)
        x = self.conv_layer2(x)

        # Flatten the output for the FC layers
        x = x.view(-1, 32 * 8 * 8)

        x = self.fc_layer1(x)
        x = self.fc_layer2(x)
        return x

# Create an instance of the model and send it to our device (GPU or CPU)
model = CNN().to(device)

print("Model Definition:")
print(model)
```

```
Model Definition:
CNN(
  (conv_layer1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv_layer2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc_layer1): Sequential(
    (0): Linear(in_features=2048, out_features=512, bias=True)
    (1): ReLU()
  )
  (fc_layer2): Linear(in_features=512, out_features=10, bias=True)
)
```

[44]:
```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

total_params_cnn = count_parameters(model)
print(f"Total Trainable Parameters (CNN): {total_params_cnn:,}")

print(f"Total Trainable Parameters (MLP): {total_params_mlp:,}")
```

```
Total Trainable Parameters (CNN): 1,068,266
Total Trainable Parameters (MLP): 1,578,506
```

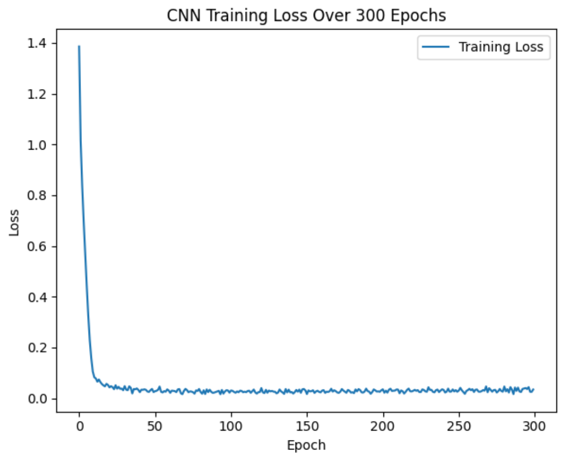
[41]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

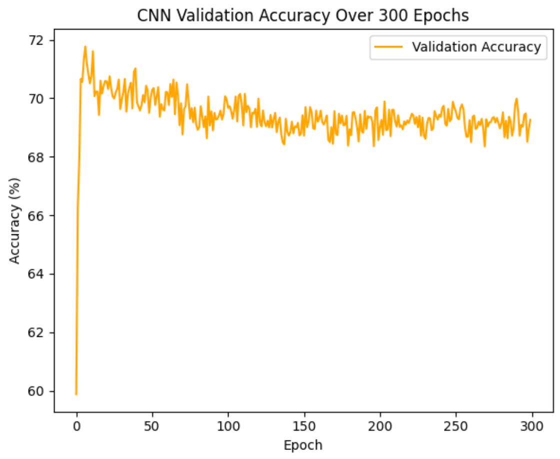
[42]:
```
print("--- Starting Training (300 Epochs) ---")

n_total_steps = len(train_loader)
all_train_loss = []
all_eval_acc = []
start_time_total = time.time()

for epoch in range(num_epochs):
    # 1. TRAINING
    model.train()
    start_time_epoch = time.time()
    running_train_loss = 0.0

    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_train_loss += loss.item()

    # VALIDATION (EVALUATION)
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    # PER-EPOCH REPORT
    avg_train_loss = running_train_loss / n_total_steps
    eval_accuracy = 100 * correct / total

    all_train_loss.append(avg_train_loss)
    all_eval_acc.append(eval_accuracy)

    # Print a report every 10 epochs
    if (epoch + 1) % 10 == 0 or epoch == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}] | "
              f"Time: {(time.time() - start_time_epoch):.2f}s | "
              f"Train Loss: {avg_train_loss:.4f} | "
              f"Evaluation Accuracy: {eval_accuracy:.2f}%")

total_training_time = time.time() - start_time_total

print("--- Finished Training ---")
print(f"\nTotal Training Time: {total_training_time:.2f} seconds")
print(f"Final Training Loss (Epoch 300): {avg_train_loss:.4f}")
print(f"Final Evaluation Accuracy (Epoch 300): {eval_accuracy:.2f}%")
```

```
--- Starting Training (300 Epochs) ---
Epoch [1/300] | Time: 18.67s | Train Loss: 1.3859 | Evaluation Accuracy: 59.88%
Epoch [10/300] | Time: 17.50s | Train Loss: 0.1051 | Evaluation Accuracy: 70.51%
Epoch [20/300] | Time: 17.26s | Train Loss: 0.0521 | Evaluation Accuracy: 70.59%
Epoch [30/300] | Time: 17.02s | Train Loss: 0.0315 | Evaluation Accuracy: 69.63%
Epoch [40/300] | Time: 16.53s | Train Loss: 0.0327 | Evaluation Accuracy: 71.02%
Epoch [50/300] | Time: 16.65s | Train Loss: 0.0241 | Evaluation Accuracy: 70.00%
Epoch [60/300] | Time: 16.50s | Train Loss: 0.0303 | Evaluation Accuracy: 70.22%
Epoch [70/300] | Time: 17.48s | Train Loss: 0.0288 | Evaluation Accuracy: 69.83%
Epoch [80/300] | Time: 17.40s | Train Loss: 0.0363 | Evaluation Accuracy: 69.10%
Epoch [90/300] | Time: 17.65s | Train Loss: 0.0223 | Evaluation Accuracy: 69.55%
Epoch [100/300] | Time: 17.11s | Train Loss: 0.0220 | Evaluation Accuracy:
69.97%
Epoch [110/300] | Time: 16.64s | Train Loss: 0.0258 | Evaluation Accuracy:
69.74%
Epoch [120/300] | Time: 16.53s | Train Loss: 0.0242 | Evaluation Accuracy:
69.02%
Epoch [130/300] | Time: 16.63s | Train Loss: 0.0255 | Evaluation Accuracy:
69.00%
Epoch [140/300] | Time: 17.04s | Train Loss: 0.0230 | Evaluation Accuracy:
68.87%
Epoch [150/300] | Time: 16.88s | Train Loss: 0.0307 | Evaluation Accuracy:
69.42%
Epoch [160/300] | Time: 16.45s | Train Loss: 0.0231 | Evaluation Accuracy:
69.20%
Epoch [170/300] | Time: 17.03s | Train Loss: 0.0278 | Evaluation Accuracy:
68.44%
Epoch [180/300] | Time: 16.33s | Train Loss: 0.0221 | Evaluation Accuracy:
68.38%
Epoch [190/300] | Time: 16.36s | Train Loss: 0.0377 | Evaluation Accuracy:
68.82%
Epoch [200/300] | Time: 17.18s | Train Loss: 0.0314 | Evaluation Accuracy:
```

```
68.57%
Epoch [210/300] | Time: 16.63s | Train Loss: 0.0344 | Evaluation Accuracy:
69.61%
Epoch [220/300] | Time: 17.00s | Train Loss: 0.0268 | Evaluation Accuracy:
69.15%
Epoch [230/300] | Time: 17.12s | Train Loss: 0.0255 | Evaluation Accuracy:
68.71%
Epoch [240/300] | Time: 16.77s | Train Loss: 0.0350 | Evaluation Accuracy:
69.44%
Epoch [250/300] | Time: 16.39s | Train Loss: 0.0255 | Evaluation Accuracy:
69.69%
Epoch [260/300] | Time: 16.94s | Train Loss: 0.0355 | Evaluation Accuracy:
69.25%
Epoch [270/300] | Time: 16.37s | Train Loss: 0.0237 | Evaluation Accuracy:
68.35%
Epoch [280/300] | Time: 16.79s | Train Loss: 0.0258 | Evaluation Accuracy:
68.97%
Epoch [290/300] | Time: 17.07s | Train Loss: 0.0419 | Evaluation Accuracy:
69.77%
Epoch [300/300] | Time: 16.47s | Train Loss: 0.0344 | Evaluation Accuracy:
69.26%
--- Finished Training ---

Total Training Time: 5065.62 seconds
Final Training Loss (Epoch 300): 0.0344
Final Evaluation Accuracy (Epoch 300): 69.26%
```

[43]:
```python
plt.figure(figsize=(12, 5))

# Plot Training Loss
plt.subplot(1, 2, 1)
plt.plot(all_train_loss, label="Training Loss")
plt.title("CNN Training Loss Over 300 Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

# Plot Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(all_eval_acc, label="Validation Accuracy", color='orange')
plt.title("CNN Validation Accuracy Over 300 Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.legend()

plt.tight_layout()
plt.show()
```


CNN Training Loss Over 300 Epochs
Training Loss
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
50
100
150
200
250
300


CNN Validation Accuracy Over 300 Epochs
Validation Accuracy
Accuracy (%)
Epoch
60
62
64
66
68
70
72
0
50
100
150
200
250
300