Homework 0
CSC 277 / 477
End-to-end Deep Learning
Fall 2024

John Doe - `jdoe@ur.rochester.edu`

Deadline: See Blackboard

Instructions

Your homework solution must be typed and prepared in LaTeX. It must be output to PDF format. To use LaTeX, we suggest using `http://overleaf.com`, which is free.

Your submission must cite any references used (including articles, books, code, websites, and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. **If you do work with others, you must list the people you worked with.** Submit your solutions as a PDF to Blackboard.

Your programs must be written in Python. The relevant code should be in the PDF you turn in. If a problem involves programming, then the code should be shown as part of the solution. One easy way to do this in LaTeX is to use the verbatim environment, i.e., \begin{verbatim} YOUR CODE \end{verbatim}.

About Homework 0: Homework 0 is intended to review prerequisite skills, help you become familiar with LaTeX, and ensure you have your programming environment prepared. *Later assignments will be more challenging! Do not think all assignments will be this short or require little time to train networks.* Copy and paste this template into an editor, e.g., `www.overleaf.com`, and then just type the answers in. You can use a math editor to make this easier, e.g., CodeCogs Equation Editor or MathType. You may use the AI (LLM) plugin for Overleaf for help you with LaTeXformatting.

Problem 1 - Linear Algebra Review #1

Let matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ and matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$, where $n \neq m$.

Part 1 (1 point)

If it is possible to compute the matrix product $\mathbf{AB}$, give the size of the matrix produced. Otherwise, write, 'Not possible.'

Answer:
Not possible.

Part 2 (1 point)

If it is possible to compute the matrix product $\mathbf{BA}$, give the size of the matrix produced. Otherwise, write, 'Not possible.'

Answer:
The size of the matrix produced is $n \times m$.

Problem 2 - Linear Algebra Review #2

Let $\mathbf{A} = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} 1 & 5 & 0 \\ 2 & 10 & 2 \end{pmatrix}$ and $\mathbf{D} = \begin{pmatrix} 1 & 1 & 2 & 3 & 5 & 4 \\ 1 & 2 & 3 & 6 & 6 & 6 \\ 1 & 1 & 2 & 3 & 5 & 4 \\ 1 & 0 & 1 & 0 & 4 & 2 \end{pmatrix}$.

Low rank approximations are used in algorithms for updating large neural networks on modest hardware. In this problem we will assess your understanding of rank in linear algebra.

Part 1 (1 points)

Compute rank($\mathbf{A}$).

Answer:
rank($\mathbf{A}$) = 1.

Part 2 (1 points)

Compute rank($\mathbf{B}$).

Answer:
rank($\mathbf{B}$) = 2.

Part 3 (1 points)

Compute rank($\mathbf{D}$).

Answer:
rank($\mathbf{D}$) = 2.

Part 4 (1 points)

Compute $\mathbf{AB}$ and rank($\mathbf{AB}$).

Answer:
$AB = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 20 & 10 \end{pmatrix}$

rank($\mathbf{AB}$) = 1.

Part 5 - Compression with linearly dependent columns (5 points)

Now we are going to use linearly independent columns of matrix $\mathbf{D}$ to create an compressed version of matrix $\mathbf{D}$. We will use the $\mathbf{D} = \mathbf{CR}$ decomposition, i.e., rank factorization (column-row factorization). Do the decomposition and provide $\mathbf{C} \in \mathbb{R}^{4\times r}$ and $\mathbf{R} \in \mathbb{R}^{r\times 6}$. Calculate the compression ratio by comparing the total number of elements in original matrix and in decomposed matrices. Explain why r is the rank of $\mathbf{D}$.

You may read about the CR factorization here and here. You can implement this factorization using singular value decomposition or by using reduced row echelon form.

Answer:

$$\mathbf{C} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$$

NOTE: C consists of the linearly independent columns of D

$$\mathbf{R} = \begin{pmatrix} 1 & 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 1 & 2 & 3 & 2 \end{pmatrix}$$

NOTE: R is formed by expressing all columns of D as linear combinations of the columns in C

Compression ratio: $\frac{24}{4\times2+2\times6} = \frac{24}{20} = 1.2$

The reason why r is the rank of $\mathbf{D}$ is because there are only 2 linearly independent columns in D.

Problem 3 - Softmax Properties

Part 1 (5 points)

Recall the softmax function, which is the most common activation function used for the output of a neural network trained to do classification. In a vectorized form, it is given by

$$\text{softmax}(\mathbf{a}) = \frac{\exp(\mathbf{a})}{\sum_{j=1}^{K} \exp(a_j)},$$

where $\mathbf{a} \in \mathbb{R}^K$ and has the "logits" from the output layer of the network before the softmax activation function. The exp function in the numerator is applied element-wise and a_j denotes the j'th element of $\mathbf{a}$.

Show that the softmax function is invariant to constant offsets to its input, i.e.,

$$\text{softmax}(\mathbf{a} + c\mathbf{1}) = \text{softmax}(\mathbf{a}),$$

where $c \in \mathbb{R}$ is some constant and $\mathbf{1}$ denotes a column vector of 1's.

Solution:

Consider adding a constant c to each of the element in vector $\mathbf{a}$, the softmax function for $\mathbf{a} + c\mathbf{1}$ is:

Proof.

$$\text{softmax}(\mathbf{a} + c\mathbf{1}) = \frac{\exp(\mathbf{a} + c\mathbf{1})}{\sum_{j=1}^{K} \exp(a_j + c)}$$

Since $\exp(a + c\mathbf{1})$ applies element-wise, we have:

$$\exp(\mathbf{a} + c\mathbf{1}) = \exp(\mathbf{a}) \cdot \exp(c)$$

Thus, we can rewrite the softmax function as:

$$\text{softmax}(\mathbf{a} + c\mathbf{1}) = \frac{\exp(\mathbf{a}) \cdot \exp(c)}{\exp(c) \cdot \sum_{j=1}^{K} \exp(a_j)} = \frac{\exp(\mathbf{a})}{\sum_{j=1}^{K} \exp(a_j)}$$

Hence,

$$\text{softmax}(\mathbf{a} + c\mathbf{1}) = \text{softmax}(\mathbf{a})$$

The softmax function is invariant to adding a constant vector to its input. □

Part 2 (3 points)

In practice, why is the observation that the softmax function is invariant to constant offsets to its input important when implementing it in a neural network?

Solution:

The invariance of the softmax function to constant offsets is important in neural networks because:

1. **Numerical Stability:** Subtracting the maximum logit before applying softmax prevents overflow/underflow in the exponentials, ensuring stable computations.
2. **Interpretability:** The relative differences between logits determine the output probabilities, making the absolute scale irrelevant and allowing focus on relative confidence between classes.
3. **Model Flexibility:** This property permits shifts in logits (e.g., due to batch normalization) without affecting the final output, enabling more flexible model design. (i.e. due to the flexibility on "paddings")

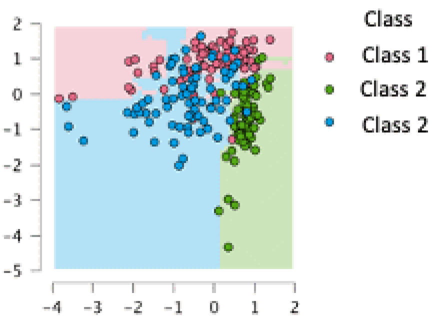

Figure 1: An example decision boundary on a different dataset.

Problem 4 - Feedforward Fully-Connected Networks (10 points)

This class assumes that you already know neural network basics. For this problem you need to use PyTorch and/or PyTorch Lightning. PyTorch Lightning is a wrapper around PyTorch that attempts to simplify a lot of the code, enforcing best practices, and making the code more portable across hardware platforms.

In this problem you will create a neural network that will be trained on the Iris dataset for multi-class classification. We will use a 2-dimensional version of the dataset. Download `iris-train.txt` and `iris-test.txt`. Each row is one data instance. The first column is the label (1, 2 or 3) and the next two columns are features. Build up your data loading process re-arrange the dataset into a suitable format for training.

We will train two neural networks. The first will have only the output layer, i.e., it is described by a 2×3 matrix with a bias and is a linear classifier. The second is a nonlinear classifier that will have one hidden layer with 5 units. Use AdamW as your optimizer and use CrossEntropyLoss. Train the two network for 1000 epochs on the training dataset. Try running your code a few times with different random initializations, since you may hit a poor local optima. You should normalize the data when given to the network by subtracting the mean of the training data. You only need to report on your best run.

Please provide the following:

1. Show the train loss curves in two plots for both networks (2 pts). Label one linear neural network and the other nonlinear neural network.
2. Measure the accuracy of both networks on the testing *and* training datasets (3 pts). Put this in a LaTeXtable.
3. Create decision boundaries for both networks and show them. Make sure to label your plots. Feel free to find helper code online for displaying the decision boundary (5 pts). An example decision boundary is provided in Fig. **??**. Show the training

data points in the figure, and color them appropriately for their label.

4. Provide your code for credit.

Answer:

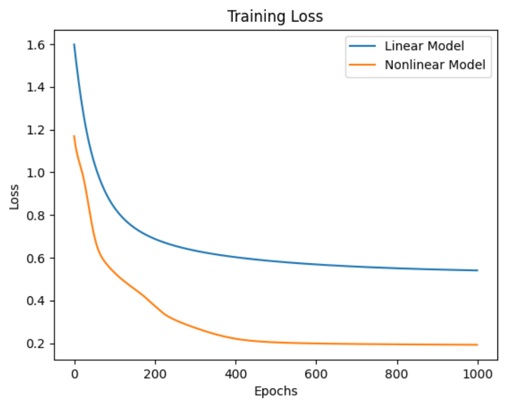

Figure 2: Train loss curves for linear neural network and nonlinear neural network

	Train Accuracy	Test Accuracy
Linear Model	0.90	0.84
Nonlinear Model	0.94	0.86

Table 1: Training and Testing Accuracy of Models

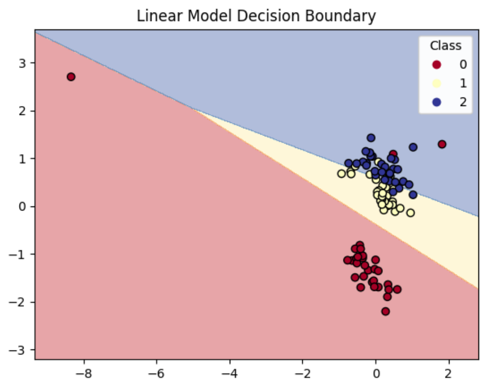

Figure 3: Linear Decision Boundary Graph

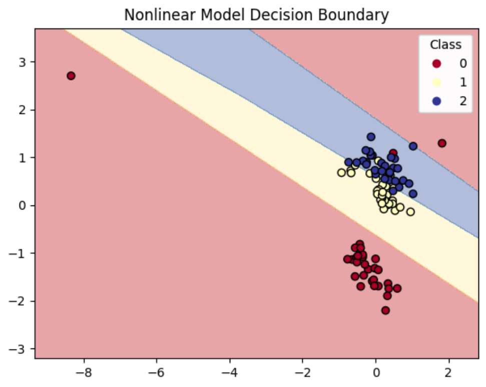

Figure 4: Non-Linear Decision Boundary Graph

Code for Problem 4:

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load and preprocess the dataset from txt files
def load_data(train_file, test_file):
    # Load the data from text files with space as delimiter
    train_data = np.loadtxt(train_file, delimiter=' ')
    test_data = np.loadtxt(test_file, delimiter=' ')

    # Split into features and labels
    X_train = train_data[:, 1:]  # The first column is the label
    y_train = train_data[:, 0].astype(int) - 1  # Convert labels to 0, 1, 2
    X_test = test_data[:, 1:]
    y_test = test_data[:, 0].astype(int) - 1  # Convert labels to 0, 1, 2

    # Standardize the features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    return torch.tensor(X_train, dtype=torch.float32),
    torch.tensor(y_train, dtype=torch.long),
    torch.tensor(X_test, dtype=torch.float32),
    torch.tensor(y_test, dtype=torch.long)

# Define the first model: Linear Classifier
class LinearModel(nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = nn.Linear(2, 3)  # 2 input features, 3 output classes

    def forward(self, x):
        return self.linear(x)

# Define the second model: Nonlinear Neural Network with one hidden layer
class NonlinearModel(nn.Module):
    def __init__(self):
        super(NonlinearModel, self).__init__()
        self.fc1 = nn.Linear(2, 5)  # 2 input features, 5 hidden units
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(5, 3)  # 5 hidden units, 3 output classes

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Training function
def train_model(model, criterion, optimizer, X_train, y_train, epochs=1000):
    losses = []
    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
    return losses
```

```
# Plot decision boundaries with class labels
def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    grid = torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype=torch.float32)
    model.eval()
    with torch.no_grad():
        Z = model(grid).argmax(dim=1).numpy().reshape(xx.shape)

    # Plot the decision boundary with class regions
    plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdYlBu)

    # Plot also the training points
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y,
        edgecolors='k', marker='o', cmap=plt.cm.RdYlBu)

    # Create a legend with class labels
    legend1 = plt.legend(*scatter.legend_elements(),
                         loc="upper right", title="Class")
    plt.gca().add_artist(legend1)

    plt.title(title)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()

# Main execution
if __name__ == "__main__":
    # Load data from text files
    X_train, y_train, X_test, y_test = load_data('iris-train.txt', 'iris-test.txt')

    # Initialize models, criterion, and optimizer
    model1 = LinearModel()
    model2 = NonlinearModel()

    criterion = nn.CrossEntropyLoss()

    optimizer1 = optim.AdamW(model1.parameters(), lr=0.01)
    optimizer2 = optim.AdamW(model2.parameters(), lr=0.01)
```

```
# Train both models
losses1 = train_model(model1, criterion, optimizer1, X_train, y_train)
losses2 = train_model(model2, criterion, optimizer2, X_train, y_train)

# Plot training loss curves
plt.plot(losses1, label='Linear Model')
plt.plot(losses2, label='Nonlinear Model')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training Loss')
plt.show()

# Evaluate accuracy
model1.eval()
model2.eval()

with torch.no_grad():
    train_preds1 = model1(X_train).argmax(dim=1)
    test_preds1 = model1(X_test).argmax(dim=1)
    train_preds2 = model2(X_train).argmax(dim=1)
    test_preds2 = model2(X_test).argmax(dim=1)

train_acc1 = accuracy_score(y_train, train_preds1)
test_acc1 = accuracy_score(y_test, test_preds1)
train_acc2 = accuracy_score(y_train, train_preds2)
test_acc2 = accuracy_score(y_test, test_preds2)

# Print accuracy in a LaTeX table
print(f"\\begin{{table}}[h!]")
print(f"\\centering")
print(f"\\begin{{tabular}}{{|c|c|c|}}")
print(f"\\hline")
print(f" & Train Accuracy & Test Accuracy \\\\ \\hline")
print(f"Linear Model & {train_acc1:.2f} & {test_acc1:.2f} \\\\ \\hline")
print(f"Nonlinear Model & {train_acc2:.2f} & {test_acc2:.2f} \\\\ \\hline")
print(f"\\end{{tabular}}")
print(f"\\caption{{Training and Testing Accuracy of Models}}")
print(f"\\end{{table}}")
```

```
# Plot decision boundaries
plot_decision_boundary(model1, X_train.numpy(), y_train.numpy(),
    'Linear Model Decision Boundary')
plot_decision_boundary(model2, X_train.numpy(), y_train.numpy(),
    'Nonlinear Model Decision Boundary')
```