Programming Assignment 1: Linear Regression and Neural Networks

Justo E. Karell

September 27, 2024

CS 583: Deep Learning

Overview

In this assignment, you will:

- Generate random data from a distribution of your choice (with 2 parameters).
- Create a regular linear regression model on the data.
- Create a neural network to perform the same linear regression.
- Experiment with adding multiple layers to your neural network (single dense layer, 2-layer, 3-layer, and 4-layer networks).
- Compare and visualize the performance of each model.

Instructions

Important: You must comment on each unique function you use. This includes:

- TensorFlow functions: Embed or link the official documentation for each function from the TensorFlow website and explain in your own words how it works.
- Mathematical functions: Comment on the purpose of mathematical operations used in your code, explaining why and how you're using them.

Example: If you use the TensorFlow 'Dense' layer function, you should comment what it does, provide a link to the documentation (https://www.tensorflow.org), and explain in your own words why you're using it in that specific context.

Step 1: Generate Random Data

1. Select a distribution of your choice with two parameters (e.g., Gaussian distribution with mean μ and variance σ^2).

2. Generate random data points for $x \in \mathbb{R}$ and corresponding y values using a simple linear relationship:

$$y = ax + b + \epsilon$$

where a and b are constants, and ϵ is some noise added to the data (sampled from your chosen distribution).

Tasks:

- Choose your distribution and specify the parameters.
- Generate at least 100 data points.
- Plot the data points on a 2D scatter plot.

Code Hint: Use libraries like NumPy and Matplotlib for data generation and plotting.

```
import numpy as np
import matplotlib.pyplot as plt

# Define parameters for the distribution
mean = 0
std_dev = 1
n_points = 100

# Generate random data for x and y
x = np.random.normal(mean, std_dev, n_points)
a = 2  # slope
b = 1  # intercept
epsilon = np.random.normal(0, 0.5, n_points)  # noise

y = a * x + b + epsilon

# Plot the data
plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Generated Data')
plt.show()
```

Step 2: Perform Linear Regression Without Neural Networks

1. Use a standard linear regression model to fit the data:

$$y = ax + b$$

2. Train the model and report the learned parameters a (slope) and b (intercept).

3. Plot the predicted line of best fit over the scatter plot of the data.

Tasks:

- Fit a linear regression model to the data.
- Plot the regression line.
- Comment on the mathematical operations used (e.g., computing the best fit line).

Code Hint: Use libraries like Scikit-learn for linear regression.

```
from sklearn.linear_model import LinearRegression

# Reshape data for scikit-learn
x_reshaped = x.reshape(-1, 1)

# Fit the linear regression model
linear_model = LinearRegression()
linear_model.fit(x_reshaped, y)

# Get the predicted y values
y_pred = linear_model.predict(x_reshaped)

# Plot the data and regression line
plt.scatter(x, y, label='Data')
plt.plot(x, y_pred, color='red', label='Regression Line')
plt.legend()
plt.show()
```

Step 3: Perform Linear Regression Using a Neural Network (Single Dense Layer)

1. Build a neural network with a single dense layer to perform the same linear regression.

2. The architecture should have 1 input, 1 output, and no activation function (i.e., just a linear transformation):

$$y = W \cdot x + b$$

3. Train the model and compare the results with your standard linear regression.

Tasks:

- Build a neural network with 1 dense layer.
- Train it using a suitable optimizer and loss function (e.g., mean squared error).
- Plot the predicted line and compare with the linear regression model.
- Explain why the 'Dense' layer works for this task and comment on the loss function used.

Code Hint: Use Keras or PyTorch for building neural networks.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the neural network model
model = Sequential()
model.add(Dense(1, input_dim=1, activation='linear'))  # link to docs

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
# link to docs

# Train the model
model.fit(x, y, epochs=100, verbose=0)

# Get the predicted y values from the NN
y_nn_pred = model.predict(x)

# Plot the data and NN regression line
plt.scatter(x, y, label='Data')
plt.plot(x, y_nn_pred, color='green', label='NN Regression Line')
plt.legend()
plt.show()
```

Step 4: Experiment with Multiple Layers

1. Build and train a neural network with 1, 2, 3, and 4 dense layers.

2. For each neural network: - The first and last layers should remain the same (input and output layers). - Hidden layers can have activation functions (ReLU, etc.) or be linear transformations.

3. Compare the performance of each neural network model.

Tasks:

- Build neural networks with 1, 2, 3, and 4 layers.
- Train each model and plot the predictions.
- Compare the performance of each model (e.g., using Mean Squared Error) and display the results on a plot.
- Explain how adding layers changes the model's performance and complexity.

Code Hint: You can add more dense layers to the neural network model.

```
# Define a neural network with multiple layers (e.g., 2 layers)
model_2_layers = Sequential()
model_2_layers.add(Dense(64, input_dim=1, activation='relu'))
# link to docs
model_2_layers.add(Dense(1, activation='linear'))

# Compile and train the model
model_2_layers.compile(optimizer='adam', loss='mean_squared_error')
model_2_layers.fit(x, y, epochs=100, verbose=0)

# Predict and plot
y_nn_2_layers_pred = model_2_layers.predict(x)
plt.plot(x, y_nn_2_layers_pred, color='blue', label='2-layer NN')
plt.legend()
plt.show()
```

Repeat the process for 3 and 4 layer networks. Be sure to comment on any changes in performance or overfitting as more layers are added.

Step 5: Compare and Visualize the Results

1. Compare the linear regression, single dense layer NN, and multi-layer NNs using a plot.

2. Discuss how adding layers affects the performance, both in terms of fitting the data and generalization.

Tasks:

- Plot all models on the same graph to compare their performance.
- Calculate the Mean Squared Error (MSE) for each model and report it.
- Write an explanation comparing the different models, especially in terms of complexity and performance.

Code Hint: Use the mean squared error function

```
from sklearn.metrics import mean_squared_error

# Calculate MSE for each model
mse_linear = mean_squared_error(y, y_pred)
mse_nn = mean_squared_error(y, y_nn_pred)
mse_nn_2_layers = mean_squared_error(y, y_nn_2_layers_pred)

# Print the results
print(f'MSE (Linear Regression): {mse_linear}')
print(f'MSE (Single Dense Layer): {mse_nn}')
print(f'MSE (2-Layer NN): {mse_nn_2_layers}')
```

Submission Instructions

Submit your assignment under "Programming Assignment 1" on Canvas. You should submit your code as either a .py file or a .ipynb file. Make sure your code is well-commented and that you include explanations for each function and its purpose.

Ensure that your file contains:

- Code for generating the data.
- Code for the linear regression model.
- Code for the neural network models with 1, 2, 3, and 4 layers.
- Plots comparing all the models.
- Explanations of each function and concept in your own words.