Lab 3: CNN, RNN, and LSTM

August 2023

1 In This Lab

1.1 Topics to Cover

- CNN, RNN, and LSTM
- Data-sets

1.2 Requirements

- Python installed on your computer.
- Basic Python programming, deep learning concepts, Jupyter Notebook.
- Libraries: PyTorch, scikit-learn, numpy, matplotlib, seaborn

1.3 Tasks

1. **Import necessary libraries:** Begin by importing the required libraries, including PyTorch, Pandas, NumPy, and Scikit-learn.
2. **Load the dataset:** Load into a Pandas DataFrame and inspect the first few rows to understand the data.
3. **Data Preprocessing:** Check for any missing values in the dataset and handle them appropriately (e.g., by filling with mean or median values). Perform any necessary feature scaling if needed.
4. **Data Visualization:** Create scatter plots or other suitable visualizations to explore the relationships between the features and the target variable (house prices).
5. **Data Splitting:** Split the dataset into training and testing sets. Use 70% of the data for training and 30% for testing.
6. **Initialize Model:** Create a model using PyTorch and train it on the training data.

7. **Model Evaluation:** Evaluate the performance of the trained model on the testing data using appropriate metrics.

8. **Making Predictions:** Use the trained model to make predictions on new data. You can either use sample data or create new data points.

9. **Experiment and Improve:** Experiment with different model parameters and observe how the model's performance changes.

2 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a class of artificial neural network that excels at processing grid-like data, such as images, by automatically learning hierarchical features from the input. CNNs are widely used in computer vision tasks and image recognition due to their ability to capture spatial patterns.

2.1 Key Components

- **Convolutional Layer:** The core building block of a CNN, where filters (also known as kernels) slide across the input to detect features like edges, textures, and shapes. Convolutional layers enable the network to learn local patterns.

- **Pooling Layer:** Pooling layers reduce the spatial dimensions of the data by summarizing regions with the most important information. Max pooling and average pooling are common types.

- **Activation Functions:** Non-linear activation functions (e.g., ReLU) introduce non-linearity, allowing the network to capture complex relationships in the data.

- **Fully Connected Layers:** At the end of the CNN, one or more fully connected layers perform classification or regression tasks based on the learned features.

- **Weight Sharing:** CNNs use weight sharing, meaning the same filter is applied across different locations of the input. This reduces the number of parameters and helps generalize features.

2.2 Training and Applications

- **Training:** CNNs are trained using backpropagation and gradient descent. Convolutional and pooling layers automatically learn useful features, and fully connected layers learn to classify based on those features.

- **Applications:** CNNs are used for a wide range of tasks, including:

- **Image Classification:** Assigning labels to images based on their content.
- **Object Detection:** Identifying and locating objects within images.
- **Semantic Segmentation:** Assigning a label to each pixel in an image to identify objects and boundaries.
- **Face Recognition:** Identifying individuals in images or videos.
- **Medical Imaging:** Diagnosing medical conditions from medical images like X-rays and MRIs.
- **Autonomous Vehicles:** Detecting and recognizing objects for self-driving cars.

In this lab exercise, you will work with a dataset and apply the concepts of deep learning using the PyTorch library in Python.

2.3 Dataset

We will use CIFAR-10 dataset.

2.4 Task 1: predict the classification of CIFAR-10

1. Follow CNN.ipynb script

2.5 Report

1. use CIFAR-100
2. plot confusion matrix.
3. change the hyper-parameter to see changes in the performance.

3 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of artificial neural network architecture designed to process sequences of data by maintaining hidden states that capture information from previous time steps. RNNs are widely used for tasks involving sequences and temporal data.

3.1 Key Components

- **Hidden State:** The hidden state in an RNN retains information from previous time steps. It acts as a memory that captures temporal dependencies and influences future predictions.
- **Recurrent Connections:** RNNs have recurrent connections that allow information to flow from one time step to the next, enabling the network to process sequences of varying lengths.
- **Time Unrolling:** An RNN can be conceptualized by unrolling it through time, creating a chain of interconnected layers for each time step.
- **Vanishing Gradient Problem:** Traditional RNNs can suffer from the vanishing gradient problem, where gradients diminish over time steps, making it challenging to capture long-range dependencies.
- **LSTM and GRU:** Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are specialized RNN variants designed to alleviate the vanishing gradient problem and capture longer dependencies.

3.2 Training and Applications

- **Training:** RNNs are trained using backpropagation through time, where gradients are computed across the time steps. Variants like LSTM and GRU include mechanisms to mitigate gradient vanishing.
- **Applications:** RNNs are used for various tasks involving sequences, including:
  - **Sequence Prediction:** Predicting the next element in a sequence, such as time series forecasting.
  - **Natural Language Processing:** Tasks like language modeling, machine translation, sentiment analysis, and text generation.
  - **Speech Recognition:** Converting spoken language to text.
  - **Handwriting Recognition:** Converting handwritten text to digital text.
  - **Music Generation:** Generating music sequences.

3.3 Dataset

1. sample data set
2. language model data set

3.4 Task 1: Simple LSTM model

1. Follow LSTM.ipynb for a simple example.
2. Follow language.ipynb for a simple NLP example.

3.5 Report

1. Use cosine function in LSTM.ipynb example and plot loss over epoch.