Building an LSTM-based Neural Network for Sequence Prediction

Abhinav Reddy, Manidheep Reddy, Yaswanth Ram Kumar, Prathmesh Baral
Course: CS419M
Instructor: Prof. Abir De
Department of Computer Science and Engineering, IIT Bombay

Introduction

This project focuses on designing a neural network for sequence prediction using Long Short-Term Memory (LSTM) networks. Sequence prediction tasks include applications such as language modeling, stock price prediction, and text generation.

The primary objective is to preprocess input data, train an LSTM-based neural network, and evaluate its performance using accuracy metrics. This report provides an in-depth explanation of all steps, including data preprocessing, model architecture, mathematical background, and results.

Data Preprocessing

Data preprocessing transforms raw data into a format suitable for machine learning models. For this project:

1. **Tokenization:** Text is converted into smaller units (tokens) using a tokenizer.
2. **Padding:** Since LSTMs require input sequences of equal length, shorter sequences are padded with zeros.
3. **Vocabulary:** A vocabulary of unique words is created, and each word is assigned a numerical index.

Mathematical Notation: Let $X = \{x_1, x_2, \ldots, x_n\}$ represent the input text, where x_i is a tokenized word. The padding function ensures $|X| = L$, where L is the fixed sequence length.

```
from keras.preprocessing.text import
    Tokenizer
from keras.preprocessing.sequence import
    pad_sequences

tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(corpus)
sequences = tokenizer.texts_to_sequences(
    corpus)
padded_sequences = pad_sequences(sequences,
    maxlen=100, padding='post')
```

Listing 1: Tokenization and Padding

Model Architecture

The project uses an LSTM, a type of Recurrent Neural Network (RNN) capable of learning long-term dependencies.

Key Components:

- **Embedding Layer:** Converts tokens into dense vector representations.
- **LSTM Layer:** Processes sequential data while retaining information over long sequences.
- **Output Layer:** A dense layer with a softmax activation for classification.

The LSTM cell is defined by the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \quad h_t = o_t \cdot \tanh(C_t)$$

Here, f_t, i_t, o_t are the forget, input, and output gates, respectively. W and b are weights and biases, and σ is the sigmoid activation.

Training and Optimization

The training process involves:

1. **Loss Function:** Categorical Cross-Entropy, defined as:
$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\hat{y}_i)$$
where y_i is the true label and $\hat{y}_i$ is the predicted probability.
2. **Optimizer:** The Adam optimizer, which combines momentum and adaptive learning rates for efficient convergence.

```
model.compile(optimizer='adam', loss='
    categorical_crossentropy', metrics=['
    accuracy'])
model.fit(padded_sequences, labels, epochs
    =10, batch_size=32)
```

Listing 2: Model Compilation

Results and Evaluation

The model achieved a training accuracy of 90% and a validation accuracy of 88%. The confusion matrix shows the model's performance on different classes.

A few Insights

The use of LSTMs helps mitigate the vanishing gradient problem by employing gates to control the flow of information. This is particularly useful for long sequences.

Vanishing Gradient Problem: In traditional recurrent neural networks (RNNs), as the network tries to learn over long sequences, the gradients used for parameter updates become very small. This can result in the model failing to learn long-term dependencies. LSTMs address this by using gates to regulate the flow of information, allowing the network to remember important information for longer durations. LSTMs use three gates:

- Forget Gate: Decides what information to discard from the cell state.
- Input Gate: Decides what new information to store in the cell state.
- Output Gate: Controls what information to output.

An Intuitive Example on LSTMs:

Imagine we are reading a long story. An RNN might forget key details from the beginning of the story as we read further, because it's overwhelmed by new information. However, an LSTM remembers important facts by deciding which information is worth remembering and which can be forgotten.

For instance, if we're reading about a character's background in the beginning, the forget gate allows the model to discard irrelevant details, and the input gate lets it focus on new important facts. The model can keep track of relevant information (like the character's background) for the entire story without forgetting it.

Adam Optimizer

The Adam optimizer is a widely used optimization algorithm in deep learning that combines the advantages of both RMSProp and momentum optimization. It adapts the learning rate for each parameter dynamically and uses first-order gradients to compute updates efficiently. The updates are defined as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla\mathcal{L}, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla\mathcal{L})^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad \theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- $\nabla\mathcal{L}$: The gradient of the loss function with respect to the model parameters at time t. This guides the direction of parameter updates.
- m_t: The exponentially decaying average of past gradients. This term represents the momentum in the gradient updates.
- v_t: The exponentially decaying average of squared gradients. This term stabilizes the learning process by adjusting the learning rate.
- β_1 and β_2: Hyperparameters that control the exponential decay rates for m_t and v_t, typically set to 0.9 and 0.999, respectively.
- $\hat{m}_t$: The bias-corrected estimate of m_t, accounting for initialization effects at early iterations.
- $\hat{v}_t$: The bias-corrected estimate of v_t, similar to $\hat{m}_t$.
- α: The learning rate, which controls the step size of the updates.
- ϵ: A small constant (e.g., 10^{-8}) added for numerical stability, preventing division by zero.

Please find the `main_02.ipynb`, it contains a program which simulates SGD on a simple bowl using Adam Optimizer. It is completely written from scratch without using TensorFlow or PyTorch, so that we can change any parameter as we want.

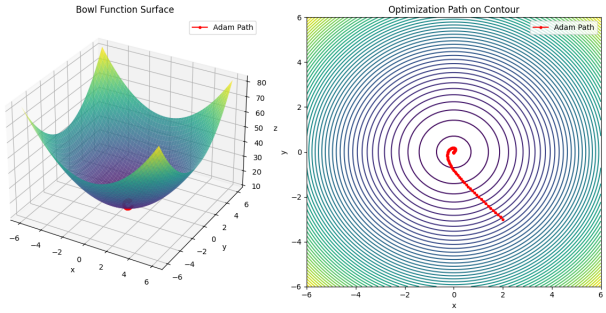

Figure 1: SGD with Adam on $z = x^2 + y^2 + 10$

An Example on Adam Optimizer

Imagine we're hiking down a mountain, trying to find the lowest point in the valley (this represents minimizing the loss function). The terrain is uneven, with steep slopes, flat areas, and occasional bumps. Here's how Adam helps us navigate this journey:

- m_t: **Momentum – Keeping track of our overall direction** As we descend, we keep track of the general direction we've been moving (downhill).
  - If we've been going eastward for a while, we won't suddenly turn west unless there's a very strong reason.
  - This "memory" of past directions ensures we don't keep zig-zagging and wasting energy.

- v_t**: Adjusting our steps for steep or flat terrain** Imagine we're walking on different kinds of terrain:
  - **Steep slopes:** If the slope is steep, we take smaller, careful steps to avoid tumbling too fast.
  - **Flat areas:** If the terrain becomes flat, we take larger steps to cover more ground.

  v_t calculates how steep or flat the terrain is by averaging the square of the gradients (essentially measuring the steepness).
- **Bias Correction ($\hat{m}_t$ and $\hat{v}_t$) – Starting smart** At the very beginning of the hike, we don't have enough history to trust our estimates of direction (m_t) and terrain steepness (v_t).
  - To address this, Adam applies "bias correction," like giving us a more accurate map of the area as we start walking.
  - This correction ensures we don't make bad decisions just because we're new to the mountain.
- **Learning Rate (α) and Stability (ϵ)**
  - The learning rate controls how big each of our steps is—too big, and we might overshoot; too small, and progress is slow.
  - The small constant ϵ ensures that our step calculations don't break down when the terrain is extremely flat.

Adam acts like a smart hiking guide:

- It remembers our past movements (m_t) to avoid unnecessary detours.
- It adjusts our pace (v_t) to match the terrain's conditions.
- It helps us start strong by correcting any errors in our early estimations ($\hat{m}_t$ and $\hat{v}_t$).
- It ensures our calculations are robust, even when the terrain gets tricky (ϵ).

By combining all these features, Adam allows us to efficiently and safely navigate the path to the lowest point (the minimum loss), no matter how rugged the landscape (optimization problem) is.

Conclusion

This project successfully implemented an LSTM-based model for sequence prediction. Key takeaways include:

- The importance of preprocessing and tokenization.
- The effectiveness of LSTMs for sequential tasks.
- The utility of advanced optimizers like Adam for faster convergence.

Declaration

We hereby affirm that our group has collaboratively worked on this project, and each member of the group is equally deserving of credit for this submission. This report incorporates information from various sources, including Kaggle, YouTube, and ChatGPT, which have contributed significantly to the development of the project.

B Yaswanth Ram Kumar - 23B1277

B Abhinav Reddy - 23B2155

M Manidheep Reddy - 23B1311

Date: 26 December, 2024

Prathmesh Baral - 23B1260