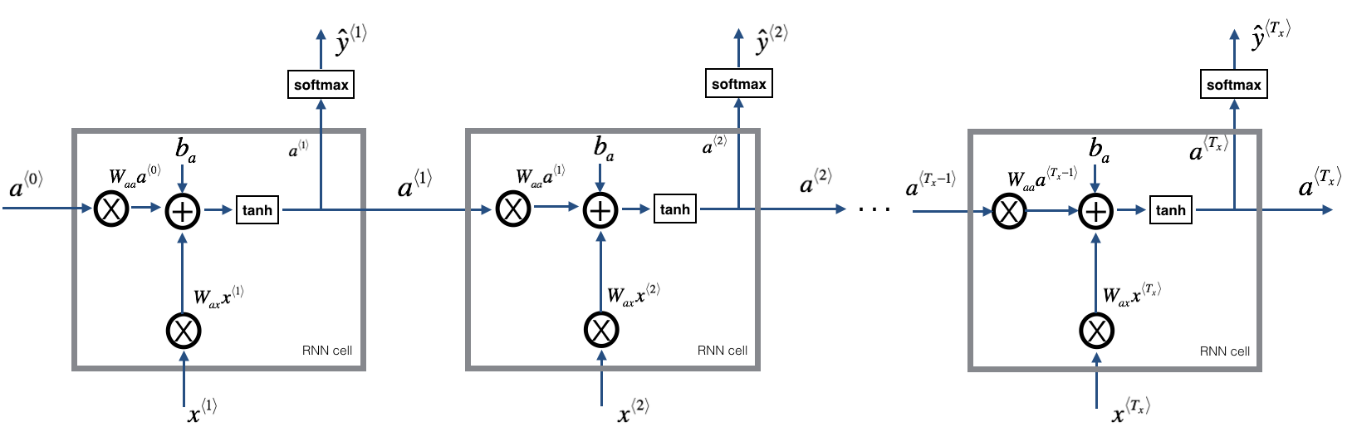
|  |  |
| --- | --- |
| Ex No: 8  Date: 25/09/2024 | **Character-Level Language Model - Name Generation using LSTM** |

**Objective:**

The main objective of this project is to implement a character-level language model using a Long Short-Term Memory (LSTM) network to generate random names based on an existing dataset of names. This project explores how recurrent neural networks can be applied generate text character by character.



**Code Explanation:**

* Imports necessary libraries.
  + numpy for numerical operations.
  + utils contains helper functions (like softmax, rnn\_forward, etc. likely from a previous assignment).
  + random for random number generation.
  + pprint for pretty-printing Python data structures.

**Data Loading and Preprocessing**:

data = open ('dinos.txt', 'r'). read () data= data. Lower ()

Reads the contents of the file dinos.txt (which contains names of dinosaurs) and converts all characters to lowercase for consistency.

 set(data) returns the unique characters in the dataset.

 list(set(data)) converts this set to a list of unique characters.

 data\_size stores the total number of characters in the dataset.

 vocab\_size stores the total number of unique characters (i.e., the vocabulary size).

Prints the total number of characters and the unique characters in the dataset.

chars = sorted(chars)

print(chars)

Sorts the list of unique characters alphabetically and prints them.

char\_to\_ix = {ch: i for i,ch in enumerate(chars) }

ix\_to\_char = { i:ch for i,ch in enumerate(chars) }

 char\_to\_ix: Creates a dictionary that maps each character to a unique index.

 ix\_to\_char: Creates another dictionary that maps indices back to the corresponding characters.

pp = pprint. PrettyPrinter(indent=4)

pp.pprint(ix\_to\_char)

Pretty prints the ix\_to\_char dictionary for easy readability, where each character is mapped to an index.

**Gradient Clipping**

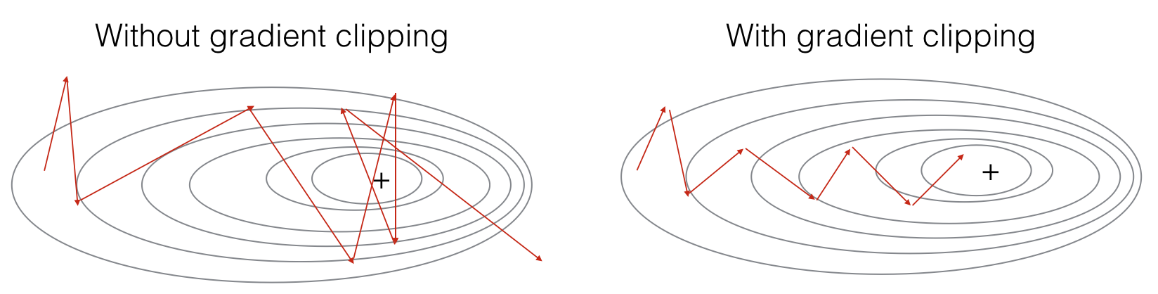
 Defines the function clip, which will clip (limit) the gradients to avoid "exploding gradients" during training.

 gradients: Dictionary containing gradient values for different weight matrices.

 maxValue: Maximum allowed value for gradients.

Extracts individual gradient matrices from the gradients dictionary.

* Loops through each gradient matrix and clips its values between -maxValue and maxValue using np.clip. The out=gradient ensures the clipping is done in-place
* Updates the gradients dictionary with the clipped gradients.
* Returns the updated gradients after clipping.



**Testing Gradient Clipping:**

Tests the clip function with randomly generated gradient values.

* np.random.randn: Generates random values from a normal distribution.
* \*10: Multiplies the values by 10 to simulate large gradients.
* Calls the clip function to clip all gradients with a maximum value of 10.

**Matrix Multiplication and Broadcasting Example:**

Initializes matrices and vectors for demonstration purposes.

Multiplies a 2D array (matrix1) with a 1D array (vector1D). This results in a 1D array due to broadcasting rules in NumPy.

Multiplies two 2D arrays and the result is also a 2D array.

**Sampling Function**

Defines the sample function that generates a sequence of characters using the trained RNN.

Extracts weight matrices and biases from the parameters dictionary. Calculates the size of the vocabulary and hidden state size.

Initializes x as a zero vector (representing the input at time step 0). a\_prev is the initial hidden state, also set to zeros.

Initializes an empty list index to store the sequence of generated character indices. idx is set to -1 initially.

Loops until the newline character (\n) is generated or the maximum of 50 characters is reached.

Performs forward propagation through the RNN to compute the hidden state a, the output z, and the softmax probabilities y.

Randomly samples the next character index idx from the softmax probability distribution y and appends it to indices.

Updates the input x to the newly generated character (one-hot encoding of the sampled index idx).

Updates the hidden state a\_prev and increments the counter.

**Optimization Function**

Defines the optimize function, which performs one step of optimization using forward propagation, backpropagation, and gradient clipping.

Forward propagates to compute the loss, backpropagates to compute the gradients, clips the gradients, and updates the parameters using gradient descent.

**Model Training Function**

Defines the main training loop for the model.

Initializes model parameters and sets the initial loss value. Hidden state a\_prev is also initialized as zeros.

The training loop iterates for the specified number of iterations (num\_iterations).

Prepares input X and labels Y, then calls optimize () to perform forward and backward propagation, gradient clipping, and parameter updates.

Every 2000 iterations, samples a few names using the trained model and prints them.

**Training Process**:

* The model is trained using stochastic gradient descent. At each step, it:
  + Uses forward propagation to compute the loss.
  + Performs backpropagation to compute gradients.
  + Clips the gradients to avoid exploding gradients.
  + Updates the model parameters using the gradients and a learning rate of 0.01.
  + Generates names periodically to assess the model’s progress.

**Results**

 The model successfully learns to generate plausible new names based on the patterns found in the dataset.

 Initially, the names generated are random, but as the model trains over time, it starts producing recognizable names with realistic letter sequences.

 After 35,000 iterations, the loss decreases significantly, and the generated names closely resemble real names.

**Result Analysis:**

**** The RNN learns to generate character sequences by leveraging the patterns from the input names.

 Gradient clipping helps prevent the issue of exploding gradients during training.

 The use of SoftMax during sampling ensures that the model produces diverse names by introducing randomness in character selection.

**Summary:**

This project successfully implements a character-level language model using LSTM to generate random names. The model learns character patterns from a given dataset and uses a recurrent neural network to generate new sequences one character at a time. Techniques like gradient clipping and stochastic sampling ensure that the model converges smoothly without encountering issues like exploding gradients. The training process shows that RNNs are effective in generating sequential data like names, and the model produces plausible and diverse names after sufficient training.

**GitHub Link:**

https://github.com/spoorthytorne/fundamentals-of-Deep-learning/tree/main/Lab%208