|  |  |
| --- | --- |
| **Ex No: 8**  **Date: 25-09-2024** | **Character level language model - Name generation** |

**Objective:**

In this assignment, we will create a character-level RNN to generate new names by learning patterns from a dataset of common names. We train the model to predict one character at a time, passing each prediction to the next step to form complete names. We also learn how to manage exploding gradients by using gradient clipping during training.

**Descriptions:**

### Dataset and Preprocessing

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

* Loading Data: The dataset of names is read from a file (dinos.txt), converted to lowercase, and stored in a string.

chars = list(set(data)).sort()

* Unique Characters: A list of unique characters in the dataset is created, and is sorted

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

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

* Character Mappings: Two dictionaries are created:
  + char\_to\_ix: Maps each character to a unique index.
  + ix\_to\_char: Maps each index back to the corresponding character.

### The Model

The model initialization and training consists of the following steps:

* Initializing the Parameters:

Wax = np.random.randn(n\_a, n\_x)\*0.01 # input to hidden

Waa = np.random.randn(n\_a, n\_a)\*0.01 # hidden to hidden

Wya = np.random.randn(n\_y, n\_a)\*0.01 # hidden to output

b = np.zeros((n\_a, 1)) # hidden bias

by = np.zeros((n\_y, 1)) # output bias

* We initialize the weights with random values and the biases with 0s.
* optimize() function:

def optimize(X, Y, a\_prev, parameters, learning\_rate = 0.01):

loss, cache = rnn\_forward(X, Y, a\_prev, parameters)

# Backpropagation through time (≈1 line)

gradients, a = rnn\_backward(X, Y, parameters, cache)

# Clip your gradients between -5 (min) and 5 (max) (≈1 line)

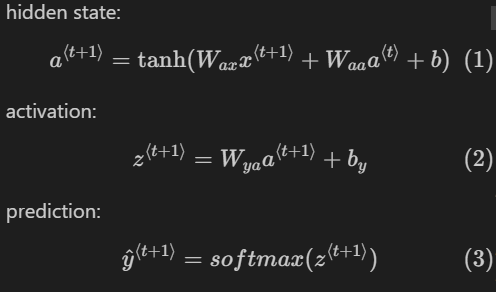
gradients = clip(gradients, 5)

# Update parameters (≈1 line)

parameters = update\_parameters(parameters, gradients, learning\_rate)

It execute one step of the optimization to train the model. It consists of the following steps:

* Forward Propagation: Compute the loss function.



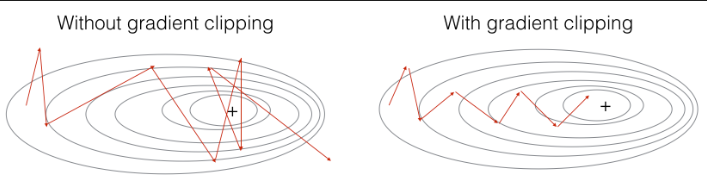
y^(t+1) is the softmax probability vector and is the probability that the character indexed is the next character

* Backward Propagation: Compute the gradients with respect to the loss function.
* Gradient Clipping:

**for gradient in [dWaa, dWax, dWya, db, dby]:**

**np.clip(gradient, -maxValue, maxValue, out=gradient)**

* + - Exploding Gradients: Large gradients can cause the training process to become unstable.
    - The clip() function clips the gradients element-wise to mitigate this problem
    - In this we clip the gradients which are above or below the maxValue to be equal to the limit

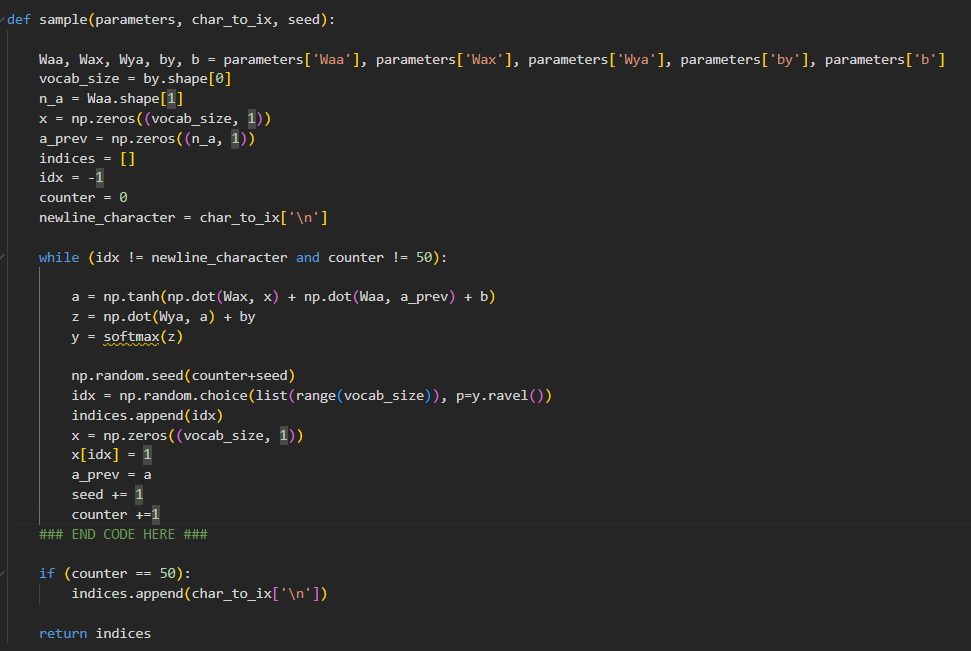


* Parameter Update: Update the parameters using the gradient descent update rule.
* Training loop:

we repeatedly call the optimize() function for the required number of iterations

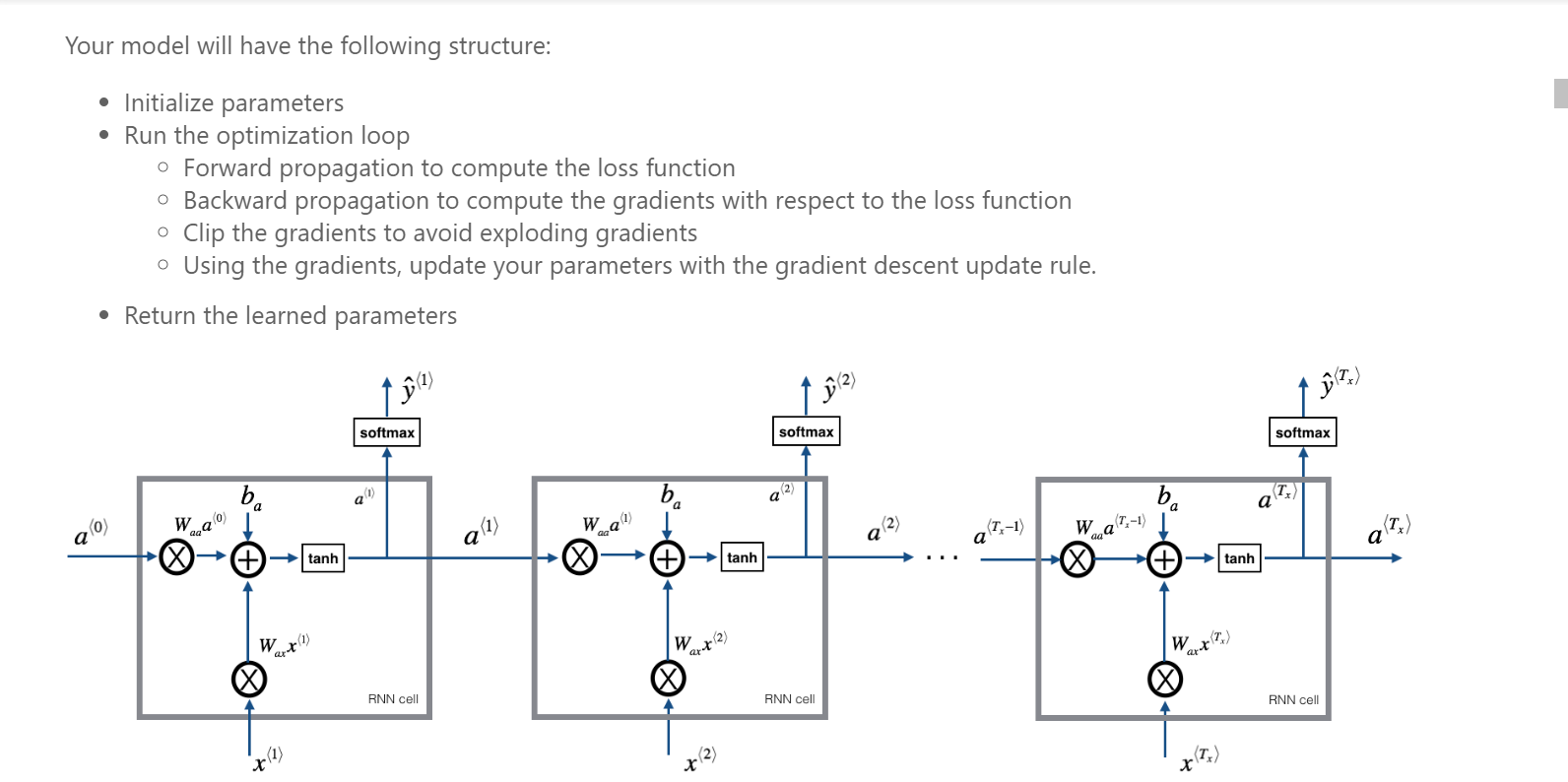
Every 2000 Iteration, generate "n" characters thanks to sample() to check if the model is learning properly

### 5. Sampling

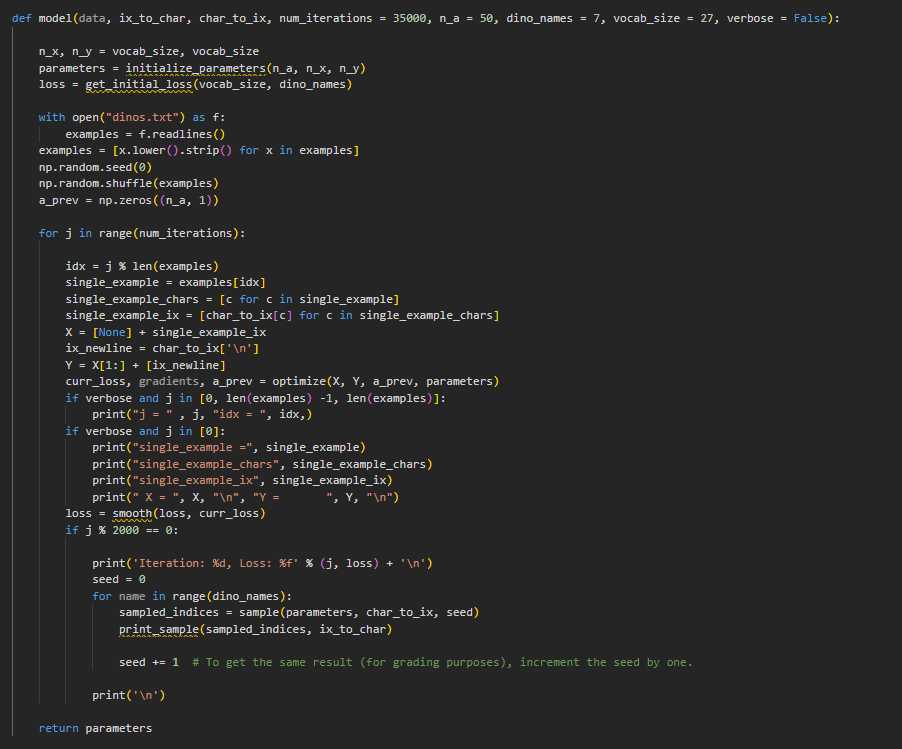


* Generating Characters: Once the model is trained, it can generate new text by sampling characters one at a time.
* Initial Input: Start with a "dummy" vector of zeros as the initial input.
* Forward Propagation: Run one step of forward propagation to get the next hidden state and the predicted probability distribution of the next character
* Sampling: Sample a character from the probability distribution and use it as the input for the next time step.
* Repeat: Repeat the process to generate a sequence of characters.

1. We compute the hidden state a using the current input x, previous hidden state a\_prev, and RNN parameters Wax, Waa, and bias b.
2. The tanh activation is applied.
3. We calculate the output probabilities y for each character by applying the softmax function to the output z (which is calculated using Wya, the hidden state a, and bias by).
4. We use np.random.choice to randomly sample a character index idx based on the probability distribution y. This step picks the next character for the name.
5. We store the sampled index idx in the indices list.
6. We update x to be the one-hot encoding of the sampled character.
7. we Update a\_prev to be the current hidden state a.



**Final Model:**

****

The model function trains the model defined to generate new dinosaur names using the dataset of existing names.

It loops through the training data for a specified number of iterations, optimizing the RNN parameters with each example.

Every 2000 iterations, it samples and prints new names to monitor progress.

The inputs are encoded characters, the RNN is updated via forward and backward propagation, and the loss is smoothed to ensure stability.

The trained model parameters are returned at the end.

**GitHub Link:** **https://github.com/HiddenMachine3/IDL/tree/main/Lab8**