USN NUMBER:1RVU22CSE018
NAME:Akshay B

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

Objective:

In this lab, we 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()

chars = list(set(data)).sort()
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

- Loading Data: The dataset of names is read from a file (dinos.txt), converted to lowercase, and stored in a string.
- Unique Characters: A list of unique characters in the dataset is created, and is sorted
- 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
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)
```

- We initialize the weights with random values and the biases with 0s.
- optimize() function:
  It execute one step of the optimization to train the model. It consists of the following steps:
  - Forward Propagation: Compute the loss function.

hidden state:

$$a^{\langle t+1 \rangle} = \tanh(W_{ax}x^{\langle t+1 \rangle} + W_{aa}a^{\langle t \rangle} + b) \quad (1)$$

activation:

$$z^{\langle t+1 \rangle} = W_{ya}a^{\langle t+1 \rangle} + b_y \quad (2)$$

prediction:

$$\hat{y}^{\langle t+1 \rangle} = softmax(z^{\langle t+1 \rangle}) \quad (3)$$

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

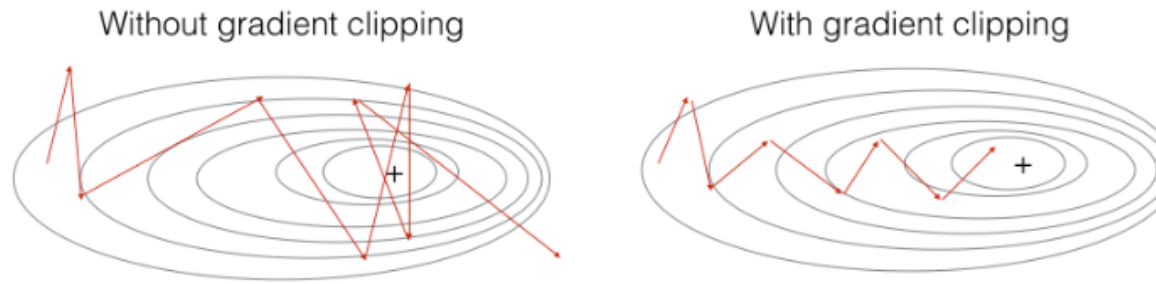
  - Backward Propagation: Compute the gradients with respect to the loss function.
  - Gradient Clipping:

```
    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.

USN NUMBER:1RVU22CSE018
NAME:Akshay B

- 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

USN NUMBER:1RVU22CSE018
NAME:Akshay B

5. Sampling

```
def sample(parameters, char_to_ix, seed):

    Waa, Wax, Wya, by, b = parameters['Waa'], parameters['Wax'], parameters['Wya'], parameters['by'], parameters['b']
    vocab_size = by.shape[0]
    n_a = Waa.shape[1]
    x = np.zeros((vocab_size, 1))
    a_prev = np.zeros((n_a, 1))
    indices = []
    idx = -1
    counter = 0
    newline_character = char_to_ix['\n']

    while (idx != newline_character and counter != 50):

        a = np.tanh(np.dot(Wax, x) + np.dot(Waa, a_prev) + b)
        z = np.dot(Wya, a) + by
        y = softmax(z)

        np.random.seed(counter+seed)
        idx = np.random.choice(list(range(vocab_size)), p=y.ravel())
        indices.append(idx)
        x = np.zeros((vocab_size, 1))
        x[idx] = 1
        a_prev = a
        seed += 1
        counter +=1
    ### END CODE HERE ###

    if (counter == 50):
        indices.append(char_to_ix['\n'])

    return indices
```

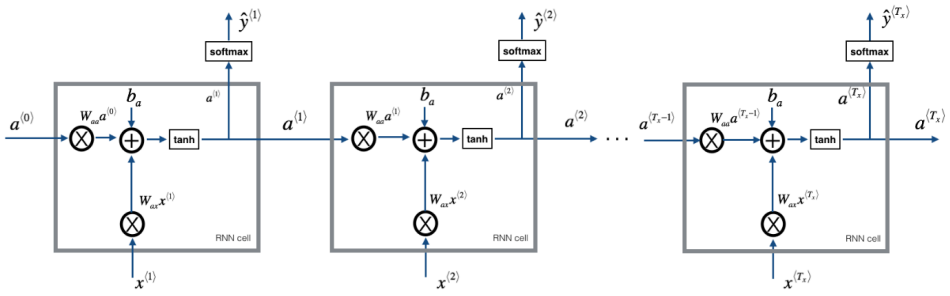
- Generating Characters: Once the model is trained, it can generate new text by sampling characters one at a time.
- Initial Input: Start with a "dummy" vector of zeros as the initial input.
- Forward Propagation: Run one step of forward propagation to get the next hidden state and the predicted probability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

USN NUMBER:1RVU22CSE018
NAME:Akshay B

Your model will have the following structure:

- Initialize parameters
- Run the optimization loop
  - Forward propagation to compute the loss function
  - Backward propagation to compute the gradients with respect to the loss function
  - Clip the gradients to avoid exploding gradients
  - Using the gradients, update your parameters with the gradient descent update rule.
- Return the learned parameters



Final Model:

```
def model(data, ix_to_char, char_to_ix, num_iterations = 35000, n_a = 50, dino_names = 7, vocab_size = 27, verbose = False):

    n_x, n_y = vocab_size, vocab_size
    parameters = initialize_parameters(n_a, n_x, n_y)
    loss = get_initial_loss(vocab_size, dino_names)

    with open("dinos.txt") as f:
        examples = f.readlines()
    examples = [x.lower().strip() for x in examples]
    np.random.seed(0)
    np.random.shuffle(examples)
    a_prev = np.zeros((n_a, 1))

    for j in range(num_iterations):

        idx = j % len(examples)
        single_example = examples[idx]
        single_example_chars = [c for c in single_example]
        single_example_ix = [char_to_ix[c] for c in single_example_chars]
        X = [None] + single_example_ix
        ix_newline = char_to_ix['\n']
        Y = X[1:] + [ix_newline]
        curr_loss, gradients, a_prev = optimize(X, Y, a_prev, parameters)
        if verbose and j in [0, len(examples) -1, len(examples)]:
            print("j = " , j, "idx = ", idx,)
        if verbose and j in [0]:
            print("single_example =", single_example)
            print("single_example_chars", single_example_chars)
            print("single_example_ix", single_example_ix)
            print(" X = ", X, "\n", "Y =       ", Y, "\n")
        loss = smooth(loss, curr_loss)
        if j % 2000 == 0:

            print('Iteration: %d, Loss: %f' % (j, loss) + '\n')
            seed = 0
            for name in range(dino_names):
                sampled_indices = sample(parameters, char_to_ix, seed)
                print_sample(sampled_indices, ix_to_char)

                seed += 1  # To get the same result (for grading purposes), increment the seed by one.

            print('\n')

    return parameters
```

USN NUMBER:1RVU22CSE018
NAME:Akshay B

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.

Steps to Build the Model:

- Define the model structure by first preprocessing the data by mapping characters to indices and vice versa, creating dictionaries like char_to_ix and ix_to_char.
- The next step is initializing the model parameters, which include weight matrices for inputs (Wax), hidden states (Waa), and outputs (Wya), along with bias vectors (b and by).
- The forward propagation function is then implemented to calculate the hidden states at each time step based on the input and the previous hidden state, while also computing the output and the loss.
- Backpropagation through time is used to compute gradients of the loss with respect to the parameters, iterating backward through time. This step helps to adjust the parameters to improve the model.
- A gradient clipping function is added to ensure that gradients remain within a specific range, preventing exploding gradients during training.
- The parameters (Wax, Waa, Wya, b, by) are updated using gradient descent after each training step to minimize the loss.
- The model is trained over multiple iterations, where it receives a sequence of characters and predicts the next one. The loss is calculated and the parameters are updated in each iteration.
- After a defined number of iterations, the model generates sample text to evaluate its learning progress by predicting one character at a time and feeding it back as input for subsequent predictions.
- Finally, hyperparameters like the learning rate and the number of iterations can be adjusted to improve the model's performance further.

GitHub Link:https://github.com/Akb-25/Foundations-of-Deep-Learning/blob/main/sep25.ipynb