Assignment 2

Sentiment Analysis Using Sequence Models

Submission:

- Submit a single zip file containing
  (a) Code file (b) Report (c) Saved Models
- There should be a Report.pdf detailing your experience and highlighting interesting results. Kindly don't explain your code in the report, just analyze the results. Your report should include your comments on the results of all the steps.

Objectives: In this assignment, you will write the code for training the Recurrent Neural Networks for classification. The goals of this assignment are as follows.

- Understand how Recurrent Neural Networks works
- How to perform sentiment analysis on text data

Sentiment Analysis Using Different Sequence Models

In this assignment you have to implement different sequence models including 'Vanilla RNN', 'LSTM' and 'GRU'. Then you have to compare their performance on a text dataset, on which you have to perform sentiment analysis. You can experiment with any structure of RNNs (Hidden Layers + Units) for this task, which ever design you choose please clearly indicate in your report.

IMDB Sentiment Analysis Dataset:

The IMDB dataset consists of 50,000 reviews on different movies by different users. All reviews are labeled as 'Positive' and 'Negative' sentiments. The CSV file of the dataset is attached in the assignment post. In the dataset 2000 reviews are not labeled. You have to use 48,000 reviews for training, validation and testing. At the end you have to make predictions on the remaining 2,000 reviews using your best model. Save these predictions as csv format and submit them with other components of assignment. Students with best predictions on unseen data will be rewarded with some bonus points.

Steps:

1. **Data loading and Pre-Processing**

   In this part you have to load the IMDB dataset from a csv file using pandas or in any other way you are comfortable with. After loading the dataset in memory you have to perform preprocessing on text (reviews of users). The helping code for preprocessing is attached. You can also improve your model's performance by better preprocessing. After preprocessing your data must be in a format that can be input to the model. (Code for encoding is also attached, however you can use other methods of encodings too, to improve performance.)

2. **Initialize Network**

   In this task you have to initialize a network. For this you have to write a function that initializes a model according to the input of the user. Users can be able to input types of RNN such as 'LSTM', 'Vanilla RNN', and 'GRU'. Users should also be able to input the number of layers in the model, number of units in the model, sequence length, output length, activation functions etc. For classification you will also need a fully connected layer to be initialized in the same function to predict sentiment labels. Your class must inherit the nn.Module class of pytorch and initialize nn.Module's initialization function. Attached starting code will help in this, but your implementation must be generic so that the user can select input, output, hidden dimensions and number of layers.

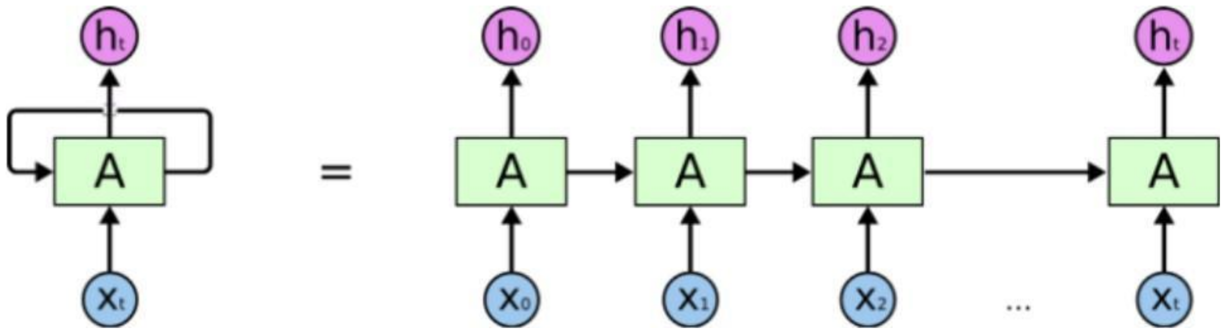
3. **Feedforward**

   This function will perform feedforward operation according to the given model. For example, If the selected network during initialization is 'LSTM' then this function will perform feedforward operation according to 'LSTM' and so on.

4. **Vanilla RNN Cell**

   RNN models the knowledge from the past to predict the future by maintaining an internal memory called "Hidden State", denoted as **h**. At each time step **t**, an instance of a sequence $x_t \in R^D$ and previous hidden state $h_{t-1} \in R^H$ are passed to RNN. Using these two inputs, the hidden state h_t gets updated. The learnable parameters of the RNN are ($W_x \in R^{H \times D}$; $W_h \in R^{H \times H}$; $b \in R^H$) input-to-hidden matrix , a hidden-to-hidden matrix and a bias vector respectively.

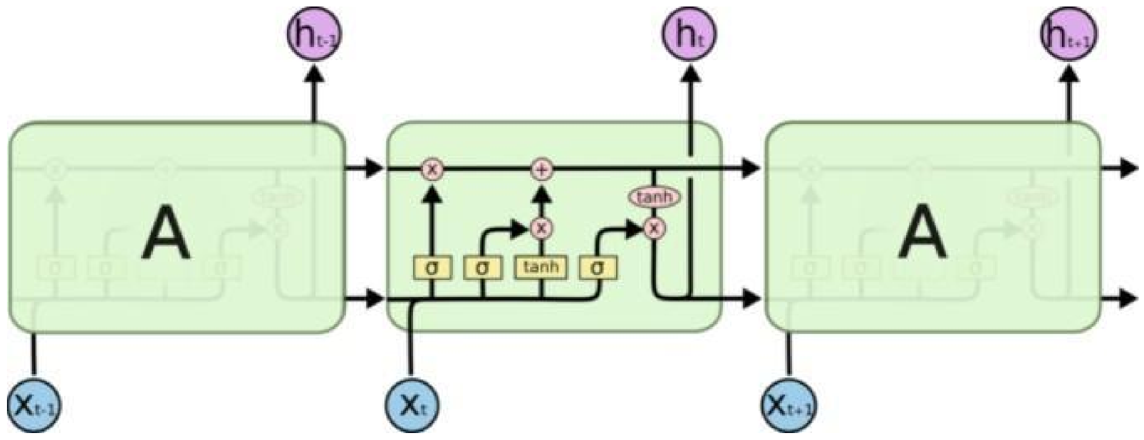
$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$



RNNs: Image Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

5. **LSTM Cell**

LSTM (Long Short Term Memories) is a special type of RNNs. It is capable of long-term learning. LSTM cell contains four linear layers and maintains two types of states (hidden state and cell state). Hidden state is the output and the cell state is the internal memory of the LSTM cell. Cell state can be thought of as a summary or context of previously seen data. Architecture of LSTMs consist of 3-gates called 'Input gate', 'Output gate' and 'Forget gate' denoted as 'i', 'f', and 'o'.



LSTMs: Image Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

At each time step:

➔ Compute an activation vector $a \in R^{4H}$, where

$$a = W_x x_t + W_h h_{t-1} + b$$

Here, $W_x \in R^{4H \times D}$; $W_h \in R^{4H \times H}$; $b \in R^{4H}$, $h_{t-1} \in R^H$ and $x_t \in R^D$

➔ Now make 4 partitions of the activation matrix as a_i, a_f, a_o, a_c so that each a_i, a_f, a_o, a_c will be H dimensional.

➔ Now compute input gate, output gate, forget gate and candidate cell state as following:

$$\begin{aligned} i &= \sigma(a_i) && \text{input gate} \\ f &= \sigma(a_f) && \text{forget gate} \\ o &= \sigma(a_o) && \text{output gate} \\ \tilde{c_t} &= \tanh(a_c) && \text{new candidate for } c_t \end{aligned}$$

➔ Now you can update current cell state c_t and output/hidden state h_t as following:

$$c_t = f \odot c_{t-1} + i \odot \tilde{c_t}$$

$$h_t = o \odot \tanh(c_t)$$

where $\odot$ is the element-wise multiplication of vectors.

6. GRU Cell

In contrast to LSTMs, GRUs have 2 gates (Reset Gate and Update Gate).

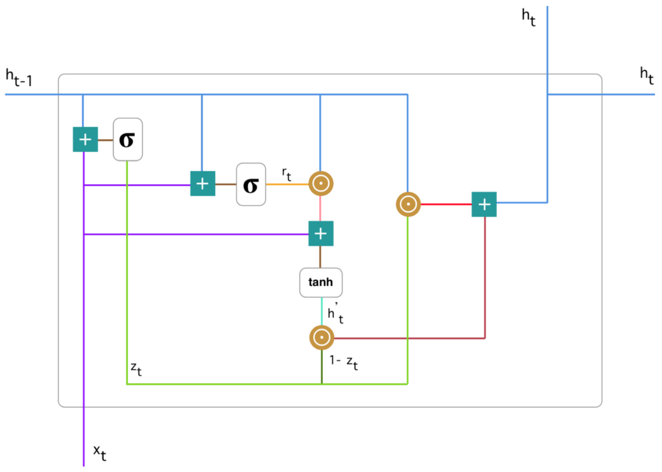


Image Source:
https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Update Gate:

$W^{(z)}$ is the weight matrix for input and $U^{(z)}$ is the weight matrix for the previous hidden state. Also $W^{(z)} \in R^{H \times D}$, $U^{(z)} \in R^{H \times H}$ and $b^{(z)} \in R^{H}$. Sigmoid function is applied as an activation function.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1} + b^{(z)})$$

Reset Gate

This gate helps the model to decide how much of the past information to forget. $W^{(r)}$ is the weight matrix for input and $U^{(z)}$ is the weight matrix for the previous hidden state. Also $W^{(r)} \in R^{H\times D}$, $U^{(r)} \in R^{H\times H}$ and $b^{(r)} \in R^{H}$. Sigmoid function is applied as an activation function.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} + b^{(r)})$$

Output/Hidden State:

Output at current state is computed as follows. Here W and U are weight matrices and b is a bias matrix. Also $W \in R^{H\times D}$, $U \in R^{H\times H}$ and $b \in R^{H}$.

$$h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1} +$$

$$b)\ h_t = z_t \odot h_{t-1} + (1-z_t) \odot h'_t$$

7. **Back-Propagation**
For Back-Propagation you can use auto-grad (pytorch) to update gradients. For this you will need to initialize all weight matrices in network initialization using tensors (of pytorch). At the time of training use 'requires_grad=True' parameter in tensors. You can use loss.backward() and optimizer.step() functions to compute gradients and update weights. You can also use builtin optimizer, and loss functions of pytorch.

8. **Training**
Now create a function to train the initialized network. You also have to keep track of loss and accuracy on training and validation data for each epoch to display loss and accuracy curves.

model, loss_array, and accuracy_array = train(net, train_set_x, train_set_y, valid_set_x, valid_set_y, learning_rate, training_epochs, loss_func, optimizer, batch_size)

9. **Testing Step**
Now create a function that uses a trained network to make predictions on a test set. Function should return predictions of the model.

```
pred = test(net, model, test_set_x, test_set_y)
```

10. **Runtime Testing**
Write a function that takes input a sentence from the user at run time, preprocesses it and predicts if the sentiment of the sentence is positive or negative.

11. **Visualize Results**
Write a function that plots loss and accuracy curves. Function should also plot confusion matrix, f1_score, and accuracy on test data. Review `sklearn.metrics` for getting different metrics of predictions. In this assignment you also have to show 2 word clouds. One word cloud shows mostly positive words and other shows negative words. Size of each word will show how oftenly that word occurs in the corpus. Following is an example of a word cloud. (You can use any python built in function to plot word cloud. Code is also attached.)

Image Source: https://www.shutterstock.com/search/positive+word+cloud

12. Main Function

Now create a main function that calls all above functions in required order. Main function should accept "Path of dataset", "size of training data", "size of validation data", "size of testing data", "number of hidden layers", "list having number of neurons in each layer", "Loss function", "optimizer", "batch size", "learning rate", "Is GPU enable(By default false if you do not have GPU)", "drop out", "Is training (default is False)", "Visualize Results (Default is False)" and "training epochs" as input. If "Is training" is true, the function should start training from scratch, otherwise the function should load the saved model and make predictions using it. While performing experiments you can use Google Colab or Jupyter Notebook. Your code must print your name, your all best/default parameters, training, validation and testing losses, accuracies, f1 scores and confusion matrix on test data and accuracy and loss curves of training and validation data.

Note: First you have to do without the embeddings, and then in the part 2, you have to do it with the embeddings

Report

In the report you will describe the critical decisions you made, important things you learned, or any decisions you made to write your algorithm a particular way. You are required to report the accuracy you achieved. For each experiment, you are required to provide analysis of various hyperparameters.

1. Write your approach for preprocessing the text (In case of using your own way of data preprocessing). Also Mention new things you learnt from this assignment.
2. Create a table having at least 3 columns 'RNN', 'LSTM', and 'GRU' and compare the results (taking same hyper parameters) including loss, accuracy curves, classification report, confusion matrix, word clouds etc. Try different hyperparameters (learning rate, optimizer, loss function, hidden layers, units, batch sizes etc) and report the effect.Discuss time taken by different models.
3. Report your best model, parameters and results you got along with your understanding about those results. (e.g LSTMs have better results because ...)

4. Show predictions of all models (with best parameters) on at least 10 sentences with both positive and negative sentiments. Make a table with multiple columns. In the first column write a sentence (any general sentence). In the second column write the sentiment you think the sentence has. and in other columns show what 'RNN', 'LSTM', and 'GRU' has predicted.
5. Show some best and worst predictions your model has made along with the original review. For this take some short reviews.
6. Your report should have multiple types of word clouds.
    - Make a dictionary that shows words and their counts in corpus (Code is given). Predict the sentiment of those words using your model and separate the positive and negative words. Now make 2 word clouds (one for negative and other is for positive words).
    - Take a review that is negatively predicted and one that is positive and make words clouds to see what kind of words are in positive and negative reviews. Repeat the process several times (atleast 5). Show the best and worst predictions you think the model has made.
    - Take all reviews, and separate positive and negative reviews. Merge all positive reviews together and similarly negative reviews. Now make word clouds to see in all positive reviews what words are common and similarly in negative reviews.

🤓 Good Luck 🤓