EEE-543 - Fall 2023 Mini Project 2: RNN

Salih Deniz Uzel (deniz.uzel@bilkent.edu.tr)
22201382

I. Introduction

A many to many Recurrent Neural Network (RNN) is implemented in python programming language. Model is trained with given hyper-parameters in the assignment for the Human Activity Recognition (HAR) dataset. The dataset has 6 different classes from number 0 to 5 included. Training, Test dataset along with validattion dataset for Option C have have equal amount of samples.

In the report, options a to c in the assignment are explained together with their implementation details in the Section II. The sections related to evaluation, interpretation are explained in their corresponding assignment options.

II. Implementation

A. Option, Implementing Recurrent Neural Network

The network mainly consist of forward and backward propagation. Terms of partial differential equations is implemented in the *BPTT* method of class RNN. In order to make weight and bias updates, the change of the weight and bias values in the recurrent layers are calculated for the Cross Entrop Loss. The Δ (delta) notation will be used to present amount of change in the corresponding differential equation variable. To denote the change of the a layer's weight following notation is used $\Delta w^{t)} = dw^{(t)} = \frac{\partial L}{\partial w^{(t)}}$. This equation can be solved by using the chain rule. The general partial differential equation for obtaining the amount of change with respect to (w.r.t.) Cross Entropy Loss function of Weights and Biases are given in the equations from (2), (11). Solution of these chain equations for MSE Loss function are given in the generic from in the equations. The general formulas used for weight and bias updates are given in the (12), and (13). Calculations are implemented in vectorized way.

The many-to-many RNN network calculates the loss for each time steps. These losses are accumulated through the forward pass. And the gradient updates calculated BPTT. Yet all the gradient updates accumulated then scaled by the number of sample size and number of time steps. In this way, exploding and vanishing gradient problems prevented during the runs. This approach was sufficient for the given task.

All the output vectors are encoded as one-hot vectors. For the the hidden layer of the network Tanh activation function, for the output Sigmoid activation function is used.

$$\frac{\partial L}{\partial a^{(t)}} = \frac{d}{da} NLL \tag{1}$$

$$\frac{dL}{a^{(t)}} = \frac{d}{da}(\frac{1}{n}\sum_{i=1}^{n} y_i(log(a_i))) \tag{2}$$

$$\frac{\partial L}{\partial z^{(t)}} = \frac{\partial \mathcal{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial z^{(t)}} \tag{3}$$

$$\frac{\partial L}{\partial W_{ya}} = \Delta w_{ya} = \frac{\partial L}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial z^{(L)}} \frac{\partial z^{(t)}}{\partial w_{ya}} \tag{4}$$

$$\frac{\partial L}{\partial b_y} = \Delta b_y = \frac{\partial L}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial b_y} \tag{5}$$

$$\frac{\partial L}{\partial a^{(t)}} = a_y + a_{next} \tag{6}$$

$$\frac{da^{(t)}}{dz^{(t)}} = \frac{d}{dz}(\text{Activation Function}) \tag{7}$$

$$\frac{dz^{(t)}}{dw_{aa}} = \frac{d}{dw_{aa}}\left(w_{aa}a^{(t-1)} + b_a\right) = a^{(t-1)} \tag{8}$$

$$\frac{dz^{(t)}}{db_a} = \frac{d}{db_a}\left(w_{aa}a^{(t-1)} + b_a\right) = 1 \tag{9}$$

$$\frac{dz^{(t)}}{dw_{ax}} = \frac{d}{dw_{ax}}\left(w_{ax}x^{(t)} + b_a\right) = a^{(t-1)} \tag{10}$$

$$\frac{dL}{a^{(t)}} = \frac{\partial L}{\partial z^{(t)}}(w_{aa}) \tag{11}$$

$$W := W - \text{LearningRate} \cdot \Delta w^{(accumulated)} \tag{12}$$

$$b := b - \text{LearningRate} \cdot \Delta b^{(accumulated)} \tag{13}$$

Activation Functions, The python implementation of activation function for both forward and backward propagation is given below. The backward propagation are given in the equations (3), (4), and (7) implicitly.

1) Sigmoid

$$\text{Forward:} \quad f(x) = \frac{1}{1 - e^{-x}}$$

$$\text{Backward:} \quad \frac{df(x)}{dx} = f(x) \cdot (1 - f(x))$$

2) Tanh

$$\text{Forward:} \quad f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\text{Backward:} \quad \frac{df(x)}{dx} = (1 - f^2(x))$$

B. Weight Initialization and Running 8 Different Experiments

Model training for each run number was started with 3 different weights. Weights are initialized with a numpy Uniform distribution in [-0.1, 0.1]. [42, 263, 682] seeds were used for these 3 different initialization. Sample set accuracy calculated by averaging the accuracy results of these 3 different runs. Experiment parameters for each run along with their total training time in seconds, are given in the Table I. Experiment results are given in the Table II. Table II consist of 3 run mean and std values of the each run no for the training, and test sets.

Loss Function, Cross Entropy Loss or Log Loss function is used for the error function. The Log Loss implemented in within the Loss calculation and Back Propagation Through Time. Loss calculation are implemented in vectorized way in Python.

Learning Coefficient, Learning coefficient [0.05, 0.01] implemented at weight and bias update w.r.t equation (12) and equation (13) in the *BPTT* method of RNN class.

Top n Accuracy Metric, implemented as get_top_n_correct instance method in RNN Class.

Processing Output Data, Metrics belong to each run stored as json file. These json files and model weights for best accuracy is provided with the report. Lastly, weights belong to hidden layer, input layer, and output layer are plotted and provided as a heat map. It is possible to observe overly activated or less activated neurons.

Loss and Accuracy plots of each run given in Fig 2 and Fig 1. Plots are given in the order of Table I. The disccussion about Top1, Top2, and Top3 Accuracies are given in Section III.

TABLE I: Model training parameters for the given 8 different "Run No".

Run No	Batch Size	N	lr	epochs	time (seconds)
1	10	50	0.05	50	287.620758
2	10	50	0.1	50	285.866596
3	10	100	0.05	50	456.180334
4	10	100	0.1	50	495.476847
5	30	50	0.05	50	169.272868
6	30	50	0.1	50	165.235722
7	30	100	0.05	50	280.877101
8	30	100	0.1	50	271.488364

* *N* is the number of neurons in the hidden layer.

C. Option, Running the Best Performing and Second Best Performing Model

When comparing the test set accuracy values among runs, Run No: 7 and 8 have the highest accuracy with a lower standard deviation. Model 7 and 8 have better generalization although other models have higher test set accuracies. This is shown on the Table II with bold font. The parameters belongs to the 7th and 8th run explicitly given in the Table reftable:parameters. Run no 8 is the best performing run followed by run no 7.

For the option C, 10% of the training set is splitted as an Validation Dataset with the same class distribution. Early stopping parameters are picked as follows: Halt the training if the accuracy does not improve more than $1e-5$, for consecutive 30 epochs. Results are given in the Table VI, Table VII, Fig 4, Fig 3.

TABLE II: Training, Validation, Test Set 3 Run Average Accuracy After 100 Epochs Training

Run	Top1 Acc.		Top2 Acc.		Top3 Acc.	
No.	μ	σ	μ	σ	μ	σ
1	32.667	1.737	45.722	2.038	61.222	5.136
2	33.389	0.797	47.444	1.039	66.167	2.121
3	34.667	0.491	50.889	1.907	66.778	1.926
4	33.111	0.208	47.000	1.705	67.056	0.749
5	30.444	4.804	42.778	5.511	57.056	8.608
6	30.778	3.502	43.611	3.400	57.167	3.336
7	**32.500**	**2.858**	43.444	3.869	55.778	5.979
8	**33.167**	**0.981**	50.278	1.100	67.556	1.220

* Run No: 7 and 8, has the highest accuracy with a lower std.

TABLE III: Top1 Accuracy Metric Training and Test Sets

Run	Training		Test	
No.	μ	σ	μ	σ
1	27.511	1.568	32.667	1.737
2	31.211	0.291	33.389	0.797
3	29.744	2.030	34.667	0.491
4	31.500	0.094	33.111	0.208
5	28.222	3.898	30.444	4.804
6	32.589	2.298	30.778	3.502
7	31.967	2.963	32.500	2.858
8	33.278	0.986	33.167	0.981

III. Interpretations of the Results

Option b)

All the models are initialized with the uniform distribution within the given boundaries. Initial model accuracies without applying BPTT varied between 10% to 30%. After the first BPTT models Top1, Top2, and Top3 accuracies updated to almost 16%, 32% and 48% respectively. The reason of seeing these numbers is random initialization. Probability distribution of the random initialization is $1/(number\ \ of\ \ classes)$ which is $1/6 \approx 16\%$ for Top1 accuracy. $2/6 \approx 33\%$ and $3/6 \approx 48\%$ for Top2, Top3 Accuracy respectively.

Most of the networks were able do learn in few epochs therefore they passes 20% accuracy. After that models started to oscillate. The reason could be big learning rates that cause overshooting. These moments can be seen explicitly as sudden upward spikes in the Accuracy Plots where the accuracy returns back to its pre-spike position. Another reason could be the lack of momentum implementations. Time Series data can be extremely variant in the different points of time. A running average strategy can prevent these sudden changes. Also small batch sizes can cause stochastic learning behaviour and gradient updates cause zig-zags in the learning processes. Main causes can be deducted as follows:

- Big learning rate, overshooting.
- Not utilizing running average method to prevent sudden changes in the Time Series data. Making it slightly invariant to changes.)

TABLE IV: Top2 Accuracy Metric Training and Test sets

Run	Training		Test	
No.	μ	σ	μ	σ
1	44.222	2.740	45.722	2.038
2	48.578	1.679	47.444	1.039
3	45.267	2.263	50.889	1.907
4	49.456	1.881	47.000	1.705
5	42.911	5.759	42.778	5.511
6	46.778	3.962	43.611	3.400
7	44.333	1.942	43.444	3.869
8	51.856	2.339	50.278	1.100

TABLE V: Top3 Accuracy Metric Training and Test sets

Run	Training		Test	
No.	μ	σ	μ	σ
1	59.589	5.393	61.222	5.136
2	66.578	1.448	66.167	2.121
3	63.456	1.746	66.778	1.926
4	68.033	1.347	67.056	0.749
5	58.522	8.406	57.056	8.608
6	62.100	6.282	57.167	3.336
7	58.444	3.180	55.778	5.979
8	70.378	0.273	67.556	1.220

- Stochastic gradient updates due to low batch sizes.

Option c)

When we compare the Loss plots of Run 7 (Fig 4)and Run 8 (Fig 3), It can be deduced that Run 8 learns faster. By looking at the epoch when Loss started to turn downwards and the rate at which Loss decreased from that point. Considering that the only difference between the two models is the learning rate, it can be said that Run8, which has a greater learning rate, learns faster. However, the upward spike in Loss and Accuracy plots of Run 8 indicates that there may be an overshoot. Due to higher learning rate, it is possible that the model may left the local minima it was in for a few epochs. Afterwards it returned to a similar local minima as a consequence of stochastic gradient descent with small batch size.

TABLE VI: Training, Validation, Test Set Accuracy on 10% Validation Set Split

	Best Performance Run No 8. Option C		
Acc.	Training	Validation	Test
Top1	42.407	48.334	36.834
Top2	60.112	63.667	52.167
Top3	75.074	79.334	68.667

TABLE VII: Training, Validation, Test Set Accuracy on 10% Validation Set Split

	Best Performance Run No 7. Option C		
Acc.	Training	Validation	Test
Top1	37.889	40.334	34.500
Top2	55.593	58.334	50.167
Top3	70.073	76.000	69.334

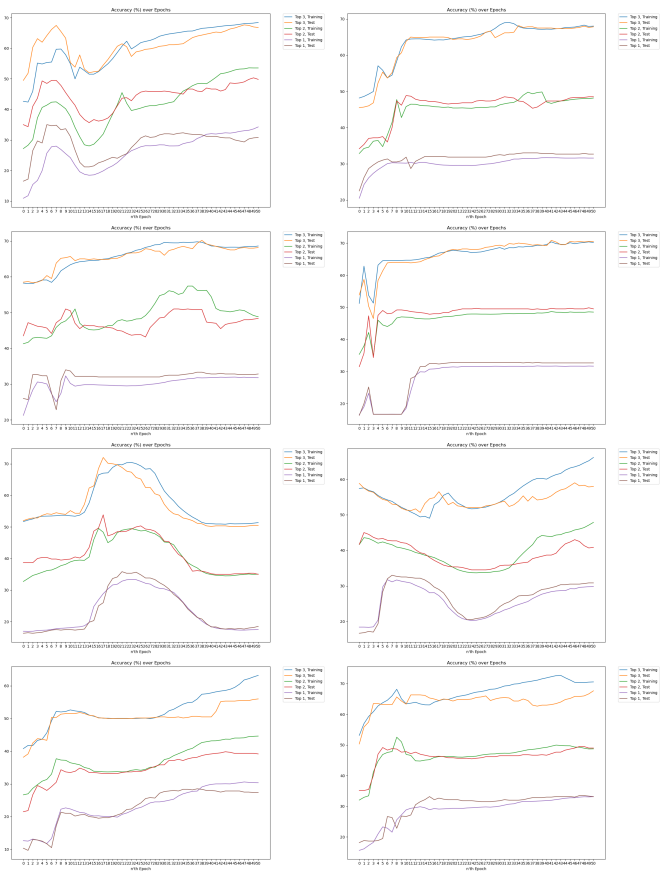

Fig. 1: Top1, Top2, and Top3 Accuracy Example Plots of Each Run for Seed Number 42.

Plots are given in the order of Table I and II. From left to right, top to bottom.

TABLE VIII: Confusion Matrix of Best Performance Run 8, on 10% splitted data set.

	Predicted Class					
True Class	0	1	2	3	4	5
0	11	13	0	67	0	9
1	15	33	0	30	0	22
2	1	0	97	2	0	0
3	17	1	1	45	0	36
4	13	14	0	46	1	26
5	9	18	0	39	0	34

IV. Further Improvements

In the experiments, heat map of the hidden layer activations, weight matrices are plotted. From these plots, it can be deduced that while some neurons contributes (very bright colors) to decision process and some of them are contributes almost none (very dark colors) Fig 5. This problem can be assessed by different tye of methods such as weight initialization , regularization.

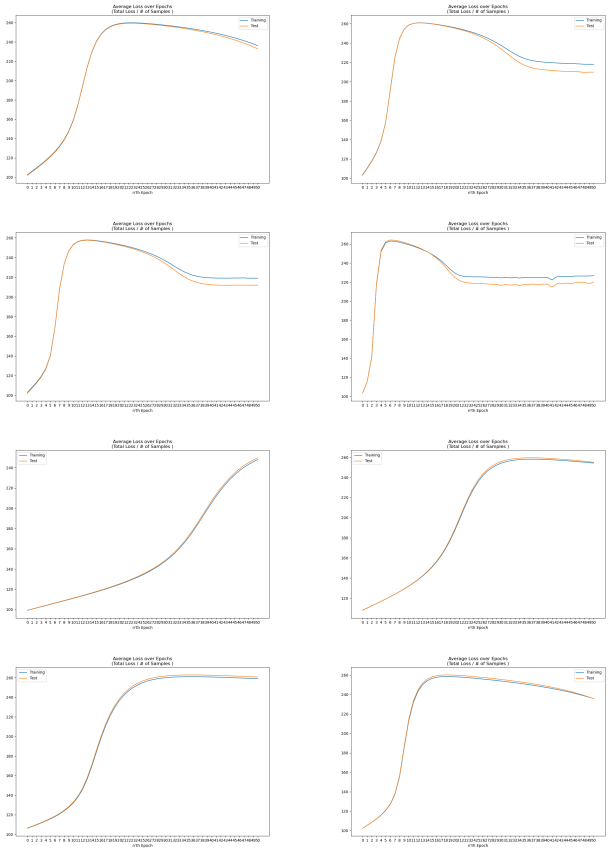

Fig. 2: Training and Test Set Losses of Each Run for Seed Number 42.

Plots are given in the order of Table I and II. From left to right, top to bottom.

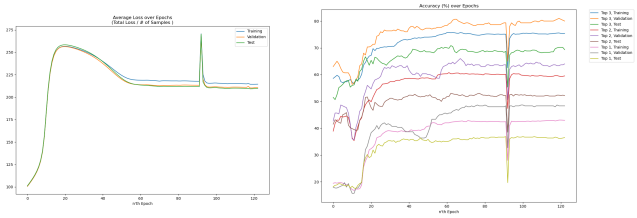

Fig. 3: Training, Validation, and Test Set Losses of Run No 8 (left) and Top1, Top2, Top3 Accuracy(right) for Seed Number 42.

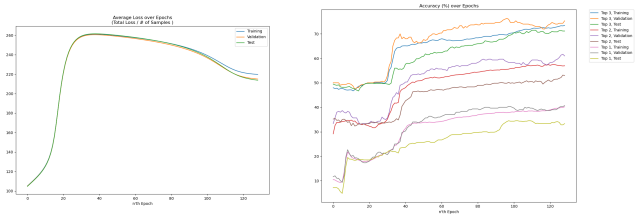

Fig. 4: Training, Validation and Test Set Losses of Run No 7 (left) and Top1, Top2, Top3 Accuracy(right) for Seed Number 42.

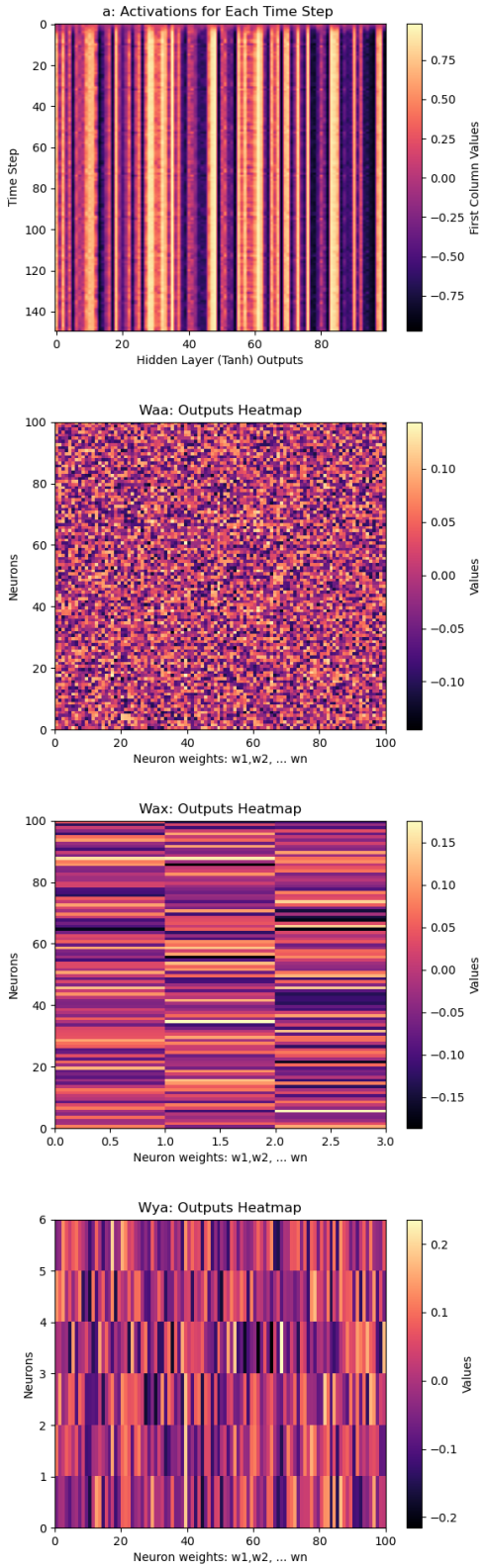

Fig. 5: Training and Test Set Losses of Each Run for Seed Number 42. From left to right, top to bottom.

From top to bottom: Hidden layer activations for each time step, Hihden layer weight matrix (W_{aa}), input layer weight matrix (W_{ax}), output layer weight matrix (W_{ya})