Report Practical assignment 2

Team members:
Fares Ben Slimane
Parviz Haggi
Mohammed Loukili
Jorge A. Gutierrez Ortega

March 20, 2019

Abstract

This report explains our approaches to solving the problems of the practical assignment 2, the experiments we performed, the results and conclusion of our work. The code we developped is uploaded to our Github repository [**?**].
In this assignment we implement and train sequential language models on the Penn Treebank dataset. Problem1-3 are respectively the implementation of a simple vanilla RNN, an RNN with a gating mechanism(GRU) and a transformer network.

1 Problem 1: Implementing a Simple RNN

The implementation of a Simple recurrent neural network can be found in the rnn.py script or in the models.py script. To train the model you need to execute the script ptb-ml.py.

2 Problem 2: Implementing an RNN with Gated Recurrent Units (GRU)

The implementation of an RNN with a gating mechanism (GRU) can be found in the gru.py script or in the models.py script. To train the model you need to execute the script ptb-ml.py.

3 Problem 3: Implementing the attention module of a transformer network

The implementation of the attention module of a transformer network can be found in the transformer.py script or in the models.py script. To train the model you need to execute the script ptb-ml.py.

4 Training language models

4.1 Model Comparison

In this section we will present the result of the first experiment of the 3 models, that is vanilla RNN, RNN with GRU, and Transformer, with respectively the correspondant given hyperpa-

rameters shown in Table 1. We run these model for the same number of epochs which is 40.

Hyperparameters	**Vanilla RNN**	**RNN with GRU**	**Transformer**
Optimizer	ADAM	SGD_LR_SCHEDULE	SGD_LR_SCHEDULE
Learning rate	0.0001	10	20
Batch size	20	20	128
Sequence length	35	35	35
Hidden size	1500	1500	512
Number of layers	2	2	6
Dropout probability	0.35	0.35	0.9

Table 1: Model's settings

The table 2 show the result of this first experiment. We notice that the Transformer is the best model in terms of time-processus and validation/training loss. The GRU is the most expensive model in term of time processing but gives a better result than the vanilla RNN. The latter is the worst one in term of training and validation loss.

Result	**Vanilla RNN**	**RNN with GRU**	**Transformer**
Training PPL	120.97	65.85	???
Validation PPL	157.82	102.63	???
Time processing per epoch (s)	411.05	668.06	174.59

Table 2: First experiment results

This result are pretty close to the expected perplexities given as reference:

- RNN: train: 120 val: 157

- GRU: train: 65 val: 104

- TRANSFORMER: train: 67 val: 146

Which proves that our models are well implemented!

Lastly, the Figures below show the learning curves for (train and validation) PPL per epoch and per wall-clock-time, for respectively the architectures above:

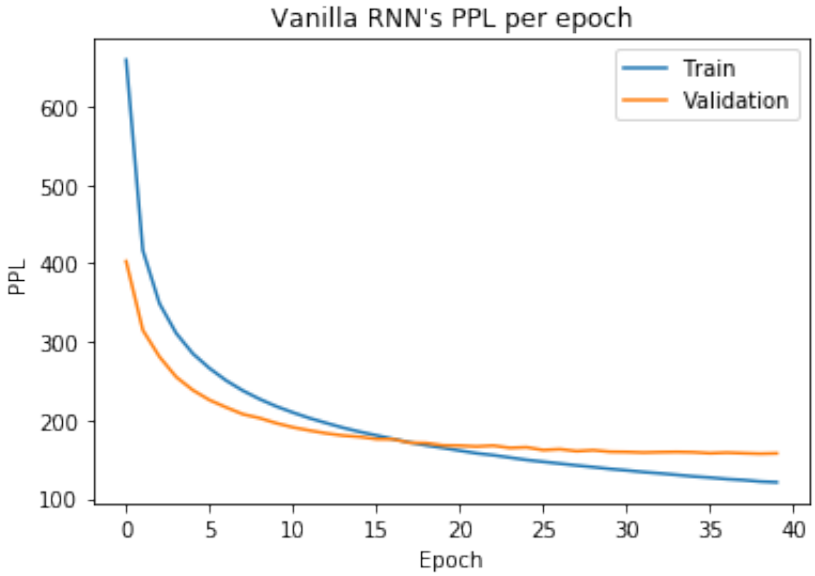

Figure 1: Vanilla RNN's PPL per epoch

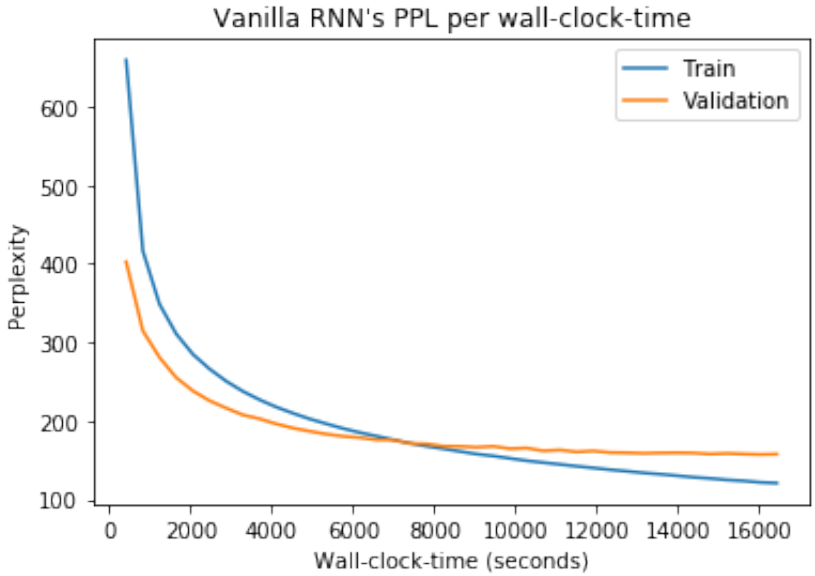

Figure 2: Vanilla RNN's PPL per wall-clock-time

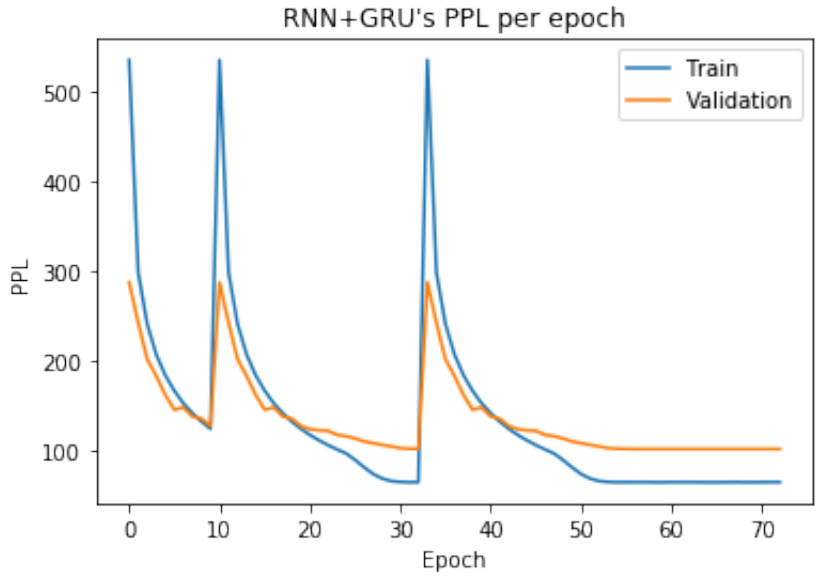

Figure 3: RNN with GRU PPL per epoch

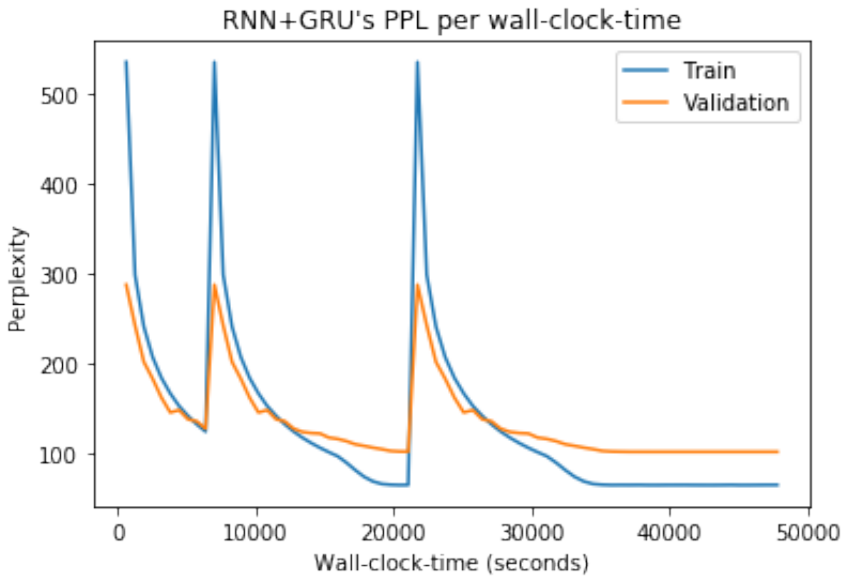

Figure 4: RNN with GRU PPL per wall-clock-time

4.2 Exploration of optimizers

In this section we will explore different optimizers for each of the previous models, along with some changes in the hyperparameters such as learning rate or hidden size or number of layers. We assume that those changes on the hyperparameters give a fair baseline to compare reasonably

the effect of each optimizer on our 3 models. We kept the hyperparameters of experiment 1 for comparison matters.

Include the following tables: 1. For each experiment in 1-3, plot learning curves (train and validation) of PPL over both epochs and wall-clock-time.

1) **Results for Vanilla RNN:** The hyperparameters used for experiments 2 and 3 are given in the Table 3.

Hyperparameters	**Experiment 1**	**Experiment 2**	**Experiment 3**
Optimizer	ADAM	SGD	SGD_LR_SCHEDULE
Learning rate	0.0001	0.0001	1
Batch size	20	20	20
Sequence length	35	35	35
Hidden size	1500	1500	512
Number of layers	2	2	2
Dropout probability	0.35	0.35	0.35

Table 3: Vanilla RNN additionnal experiments' hyperparameters

The results of these experiments are shown in the Table 4. We notice that the SGD got the worse result and it couldn't converge within 40 epochs, wheras ADAM gets the best result for the same number of epochs and the same hyperparameters. Additionnaly, the SGD_LR_SCHEDULE works better than SGD for a bigger learning rate and lower capacity of the model. We can conclude that ADAM is the better optimizer for vanilla RNN given the hyperparameters shown at Table 3.

Result	**Experiment 1**	**Experiment 2**	**Experiment 3**
Training PPL	120.97	3008.63	229.56
Validation PPL	157.82	2220.49	195.67
Average time processing per epoch (s)	411	384	185

Table 4: Vanilla RNN results experiments

2) **Results for RNN with GRU:** The experiments 2 and 3, for RNN with GRU model, used respectively the parameters given in Table 5.

Hyperparameters	**Experiment 1**	**Experiment 2**	**Experiment 3**
Optimizer	SGD_LR_SCHEDULE	SGD	ADAM
Learning rate	10	10	0.0001
Batch size	20	20	20
Sequence length	35	35	35
Hidden size	1500	1500	1500
Number of layers	2	2	2
Dropout probability	0.35	0.35	0.35

Table 5: RNN with GRU additionnal experiments' hyperparameters

The results are shown in Table 6. We notice that SGD_LR_SCHEDULE did the best result on the validation set but the worst in training set, which means he generalize well and has lowest variance. The SGD did a better result than with the Vanilla RNN experiment due to a bigger learning rate, which is logic as he must make bigger steps within a limited number of epochs. ADAM did the best job on the training set, but the variance was greater than SGD_LR_SCHEDULE, which may seem as a beggining of overfitting.

Result	**Experiment 1**	**Experiment 2**	**Experiment 3**
Training PPL	65.85	50.33	59.98
Validation PPL	102.63	121.36	113.71
Average time processing per epoch (s)	668	648	675

Table 6: First experiment results

3) **Results for Transformer:** And finally the Transformer used the following parameters:

Hyperparameters	**Experiment 1**	**Experiment 2**	**Experiment 3**
Optimizer	SGD_LR_SCHEDULE	SGD	ADAM
Learning rate	20	20	0.001
Batch size	128	128	128
Sequence length	35	35	35
Hidden size	512	512	512
Number of layers	6	6	2
Dropout probability	0.9	0.9	0.9

Table 7: Transformer additionnal experiments' hyperparameters

And gives the following results: The Transformer got the following results:

Result	**Experiment 1**	**Experiment 2**	**Experiment 3**
Training PPL	???	???	???
Validation PPL	???	???	???
Average time processing per epoch (s)	???	???	???

Table 8: First experiment results

As a global conclusion, and assuming that the comparison here was made with a set of hyperparameters that give a fair baseline for all the models, we can say that there is no universal best optimizer regardeless of the model architecture. In fact, each model architecture works better with a suited optimizer. Howerver, we notice that ADAM is very powerfull optimizer that can makes the model converge very quickly and if not well tuned could makes the model overfitt the training data. For the next experiment we will use the best optimizer that gave the better result and we will do a hyperparameter to find the best parameters that gives a better result than those found in these experiments.

4.3 Exploration of hyperparmeters

Figures and Tables: Each table and

gure should have an explanatory caption. For tables, this goes above, for
figures it goes below. Tables should have appropriate column and/or row headers. Figures should have labelled axes and a legend.

Include the following tables: 1. For each experiment in 1-3, plot learning curves (train and validation) of PPL over both epochs and wall-clock-time.

2. Make a table of results summarizing the train and validation performace for each experiment, indicating the architecture and optimizer. Sort by architecture, then optimizer, and number the experiments to refer to them easily later. Bold the best result for each architecture.

3. List all of the hyperparameters for each experiment in your report (e.g. specify the command you run in the terminal to launch the job, including the command line arguments).

4. Make 2 plots for each optimizer; one which has all of the validation curves for that optimizer over epochs and one over wall-clock-time. 5. Make 2 plots for each arcitecture; one which has all of the validation curves for that architecture over epochs and one over wall-clock-time.

4.4 Discussion

5 Problem 5: Detailed evaluation of trained models

References