IFT6135 - Representation Learning
Assignment 3 - Programming Part

Instructor:
Prof Aaron Courville

Team member/s:
Masoud Karami
(`p1220464`)

GitHub

May 20, 2019

Problem_4

Training language models (20pts) Unlike in classification problems, where the performance metric is typically accuracy, in language modelling, the performance metric is typically based directly on the cross-entropy loss, i.e. the negative log likelihood (NLL) the model assigns to the tokens. For word-level language modelling it is standard to report **perplexity (PPL)**, which is the exponentiated average per-token NLL (over all tokens) where t is the index with the sequence, and n indexes different sequences. For Penn Treebank in particular, the test set is treated as a single sequence (i.e. N = 1). The purpose of this assignment is to perform model exploration, which is done using a validation set. As such, we do not require you to run your models on the test set.

(1) Model Comparison In this problem you will run one experiment for each architecture (hyperparameter settings specified in the code) (3 experiments).
(2) Exploration of optimizers You will run experiments with the following three optimizers (use the implementations provided in the code or given in Pytorch/Tensorflow; you don't need to implement these yourself) (6 experiments)

- "Vanilla" Stochastic Gradient Descent (SGD)
- SGD with a learning rate schedule; divide the learning rate by 1.15 after eachepoch
- Adam

(3) Exploration of hyperparmeters In this problem, you will explore combinations of hyperparameters to try to find settings which achieve better validation performance than those given to you in (1). Report at least 3 more experiments per architecture (you may want to run many more short experiments in order to find potentially good hyperparameters). (9+ experiments).

Figures and Tables:

Each table and figure should have an explanatory caption. For tables, this goes above, for figures it goes below. If it is necessary for space to use shorthand or symbols in the figure or table these should be explained in the caption. 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 performance for each experiment, indicating the architecture and optimizer. Sort by architecture, then optimizer, and number the experiments to refer to the easily later. Bold the best result for each architecture. 3

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**. **Discussion** Answer the following questions in the report, referring to the plots / tables / code:

1. What did you expect to see in these experiments, and what actually happens? Why do you think that happens?

2. Referring to the learning curves, qualitatively discuss the differences between the three optimizers in terms of training time, generalization performance, which architecture they're best for, relationship to other hyperparameters, etc.

3. Which hyperparameters+optimizer would you use if you were most concerned with wallclock time? With generalization performance? In each case, what is the \cost" of the good performance (e.g. does better wall-clock time to a decent loss mean worse final loss? Does better generalization performance mean longer training time?)
4. Which architecture is most \reliable" (decent generalization performance for most hyperparameter+optimizer settings), and which is more unstable across settings?
5. Describe a question you are curious about and what experiment(s) (i.e. what architecture/optimizer/hyperparameters) you would run to investigate that question.

For Problem 4.1 and 4_2 (Exploration of hyperparmeters), the hyperparameter settings you should run are as follows:

perplexities:
RNN: train: 120 val: 157
GRU: train: 65 val: 104
TRANSFORMER: train: 67 val: 146

--model=RNN --optimizer=ADAM --initial_lr=0.0001 --batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2 --dp_keep_prob=0.35 --save_best

epoch: 0 train ppl: 537.4743367162876 val ppl: 344.5164952647441 best val: 344.5164952647441 time (s) spent in epoch: 966.2697131633759
epoch: 1 train ppl: 380.4602712501069 val ppl: 295.58910408405364 best val: 295.58910408405364 time (s) spent in epoch: 968.079220533371
.
.
.
epoch: 36 train ppl: 208.00308127282685 val ppl: 204.28527402949805 best val: 204.28527402949805 time (s) spent in epoch: 967.3017835617065
epoch: 37 train ppl: 206.834707450425 val ppl: 201.84362993798533 best val: 201.84362993798533 time (s) spent in epoch: 967.2667570114136
epoch: 38 train ppl: 205.93189603913618 val ppl: 202.3207655703608 best val: 201.84362993798533 time (s) spent in epoch: 967.3122138977051
epoch: 39 train ppl: 205.01723384982347 val ppl: 205.74799219439976 best val:201.84362993798533 time (s) spent in epoch: 967.1862807273865

--model=RNN --optimizer=ADAM --initial_lr=0.0001 --batch_size=1 --seq_len=35 --hidden_size=1500 --num_layers=2 --dp_keep_prob=0.35 --save_best

epoch: 0 train ppl: 537.1842785100089 val ppl: 314.41165070734024 best val: 314.41165070734024 time (s) spent in epoch: 186.59717345237732
epoch: 1 train ppl: 299.6922586445149 val ppl: 233.2374905480512 best val: 233.2374905480512 time (s) spent in epoch: 187.21093320846558
epoch: 2 train ppl: 242.37043341649877 val ppl: 202.83336000822047 best val: 202.83336000822047 time (s) spent in epoch: 187.12234473228455
epoch: 3 train ppl: 208.9213045524732 val ppl: 175.97474938083772 best val: 175.97474938083772 time (s) spent in epoch: 187.12407398223877

--model=GRU --optimizer=SGD_LR_SCHEDULE --initial_lr=10 --batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2 --dp_keep_prob=0.35 --save_best

epoch: 0 train ppl: 537.1842785100089 val ppl: 314.41165070734024 best val: 314.41165070734024 time (s) spent in epoch: 186.59717345237732
epoch: 1 train ppl: 299.6922586445149 val ppl: 233.2374905480512 best val: 233.2374905480512 time (s) spent in epoch: 187.21093320846558
epoch: 2 train ppl: 242.37043341649877 val ppl: 202.83336000822047 best val: 202.83336000822047 time (s) spent in epoch: 187.12234473228455
epoch: 3 train ppl: 208.9213045524732 val ppl: 175.97474938083772 best val: 175.97474938083772 time (s) spent in epoch: 187.12407398223877

--model=TRANSFORMER --optimizer=SGD_LR_SCHEDULE --initial_lr=20 --batch_size=128 --seq_len=35 --hidden_size=512 --num_layers=6 --dp_keep_prob=0.9 --save_best
epoch: 0 train ppl: 161067.61995196287 val ppl: 13401.448777011537 best val: 13401.448777011537 time (s) spent in epoch: 20.54786229133606
epoch: 1 train ppl: 11090.413289387216 val ppl: 3529.115946881324 best val: 3529.115946881324 time (s) spent in epoch: 20.643086910247803
epoch: 2 train ppl: 5249.01015041793 val ppl: 8092.191330230684 best val: 3529.115946881324 time (s) spent in epoch: 20.652815341949463
.
.
.
epoch: 34 train ppl: 101.85295238979847 val ppl: 150.88607169228422 best val: 150.88606719552894 time (s) spent in epoch: 20.97251033782959
epoch: 35 train ppl: 101.78702970030668 val ppl: 150.88607169228422 best val: 150.88606719552894 time (s) spent in epoch: 20.96673893928528
epoch: 36 train ppl: 101.81527749089662 val ppl: 150.88607169228422 best val: 150.88606719552894 time (s) spent in epoch: 20.97084355354309

RNN_SGD_model=RNN_optimizer=SGD_initial_lr=0.0001_batch_size=20_seq_len=35_hidden_size=1500_num_layers=2_dp_keep_prob=0.35_0
epoch: 0 train ppl: 10153.389609842496 val ppl: 9995.950436560383 best val: 9995.950436560383 time (s) spent in epoch: 106.70849299430847
epoch: 1 train ppl: 10041.171319457608 val ppl: 9885.356017356124 best val: 9885.356017356124 time (s) spent in epoch: 107.43877339363098
epoch: 2 train ppl: 9933.601753604387 val ppl: 9775.87386232652 best val: 9775.87386232652 time (s) spent in epoch: 107.73892855644226
epoch: 3 train ppl: 9819.34668859868 val ppl: 9666.649299619472 best val: 9666.649299619472 time (s) spent in epoch: 107.79395604133606
epoch: 4 train ppl: 9715.342597712182 val ppl: 9556.98701884717 best val: 9556.98701884717 time (s) spent in epoch: 107.81674909591675
epoch: 5 train ppl: 9604.052776653618 val ppl: 9446.01297026228 best val: 9446.01297026228 time (s) spent in epoch: 107.83550381660461
epoch: 6 train ppl: 9494.46212353506 val ppl: 9332.958851959293 best val: 9332.958851959293 time (s) spent in epoch: 107.84978365898132
epoch: 7 train ppl: 9381.314394249506 val ppl: 9216.910965598863 best val: 9216.910965598863 time (s) spent in epoch: 107.87884259223938
epoch: 8 train ppl: 9262.736364545035 val ppl: 9097.009502387864 best val: 9097.009502387864 time (s) spent in epoch: 107.87660312652588

epoch: 9 train ppl: 9143.26293705174 val ppl: 8972.33535024314 best val: 8972.33535024314 time (s) spent in epoch: 107.86887574195862
epoch: 10 train ppl: 9018.908058693565 val ppl: 8841.63260375737 best val: 8841.63260375737 time (s) spent in epoch: 107.87480926513672

GRU_SGD_model=GRU_optimizer=SGD_initial_lr=10_batch_size=20_seq_len=35_hidden_size=1500_num_layers=2_dp_keep_prob=0.35_0

epoch: 0 train ppl: 537.1842785100089 val ppl: 314.41165070734024 best val: 314.41165070734024 time (s) spent in epoch: 186.21709370613098
epoch: 1 train ppl: 299.6922586445149 val ppl: 233.2374905480512 best val: 233.2374905480512 time (s) spent in epoch: 186.98715686798096
epoch: 2 train ppl: 242.37043341649877 val ppl: 202.83336000822047 best val: 202.83336000822047 time (s) spent in epoch: 187.01991868019104
epoch: 3 train ppl: 208.9213045524732 val ppl: 175.97474938083772 best val: 175.97474938083772 time (s) spent in epoch: 186.9956569671631

TRANSFORMER_SGD_model=TRANSFORMER_optimizer=SGD_initial_lr=20_batch_size=128_seq_len=35_hidden_size=512_num_layers=6_dp_keep_prob=.9_0
epoch: 0 train ppl: 161067.61995196287 val ppl: 13401.448777011537 best val: 13401.448777011537 time (s) spent in epoch: 20.632240295410156
epoch: 1 train ppl: 11090.413289387216 val ppl: 3529.115946881324 best val: 3529.115946881324 time (s) spent in epoch: 20.680968523025513
epoch: 2 train ppl: 5249.01015041793
.
.
epoch: 18 train ppl: 256.7468665640804 val ppl: 243.02356886739918 best val: 243.02356886739918 time (s) spent in epoch: 20.932648420333862
epoch: 19 train ppl: 222.8921042960948 val ppl: 228.0787687982769 best val: 228.0787687982769 time (s) spent in epoch: 20.932570219039917

RNN_SGD_LR_SCHEDULE_model=RNN_optimizer=SGD_LR_SCHEDULE_initial_lr=1_batch_size=20_seq_len=35_hidden_size=512_num_layers=2_dp_keep_prob=0.35_4
epoch: 0 train ppl: 834.2520324173921 val ppl: 501.75146271717455 best val: 501.75146271717455 time (s) spent in epoch: 41.684507608413696
epoch: 1 train ppl: 531.8411740581333 val ppl: 404.60293617873697 best val: 404.60293617873697 time (s) spent in epoch: 41.6408166885376
epoch: 2 train ppl: 451.61860383365513 val ppl: 362.3703509153453 best val: 362.3703509153453 time (s) spent in epoch: 41.76957821846008
epoch: 3 train ppl: 405.3540721148843 val ppl: 321.8989841849304 best val: 321.8989841849304 time (s) spent in epoch: 41.78450393676758
epoch: 4 train ppl: 374.3208082531098 val ppl: 308.10476098496963 best val: 308.10476098496963 time (s) spent in epoch: 41.78944993019104
epoch: 5 train ppl: 350.75759019530017 val ppl: 281.69836096047675 best val: 281.69836096047675 time (s) spent in epoch: 41.77453398704529

GRU_ADAM_model=GRU_optimizer=ADAM_initial_lr=0.0001_batch_size=20_seq_len=35_hidden_size=1500_num_layers=2_dp_keep_prob=0.35_0
epoch: 0 train ppl: 647.4465759784986 val ppl: 397.5014616938562 best val: 397.5014616938562 time (s) spent in epoch: 192.939679145813
epoch: 1 train ppl: 396.8722846270625 val ppl: 304.60164745308623 best val: 304.60164745308623 time (s) spent in epoch: 193.68166494369507
epoch: 2 train ppl: 324.6560297855029 val ppl: 260.18255397561785 best val: 260.18255397561785 time (s) spent in epoch: 193.76597547531128

4_3. Which architecture is most \reliable" (decent generalization performance for most hyperparameter + optimizer settings), and which is more unstable across settings?

batch_size 20
code_file ptb-lm.py
data data
debug False
dp_keep_prob 0.35
emb_size 200
evaluate False
hidden_size 1500
initial_lr 10.0
model GRU
num_epochs 40
num_layers 2
optimizer SGD
save_best False
save_dir

Compression of Recurrent Neural Networks for Efficient Language Modeling

Artem M. Grachev[a,b], Dmitry I. Ignatov[b], Andrey V. Savchenko[c]

[a]Samsung R&D Institute, Moscow, Russia
[b]National Research University Higher School of Economics, Moscow, Russia
[c]National Research University Higher School of Economics, Laboratory of Algorithms and Technologies for Network Analysis, Nizhny Novgorod, Russia

Abstract

Recurrent neural networks have proved to be an effective method for statistical language modeling. However, in practice their memory and run-time complexity are usually too large to be implemented in real-time offline mobile applications. In this paper we consider several compression techniques for recurrent neural networks including Long-Short Term Memory models. We make particular attention to the high-dimensional output problem caused by the very large vocabulary size. We focus on effective compression methods in the context of their exploitation on devices: pruning, quantization, and matrix decomposition approaches (low-rank factorization and tensor train decomposition, in particular). For each model we investigate the trade-off between its size, suitability for fast inference and perplexity. We propose a general pipeline for applying the most suitable methods to compress recurrent neural networks for language modeling. It has been shown in the experimental study with the Penn Treebank (PTB) dataset that the most efficient results in terms of speed and compression-perplexity balance are obtained by matrix decomposition techniques.

Keywords: Recurrent neural network compression, language modeling, mobile devices, low-rank factorization

GRU_SGD_model=GRU_optimizer=SGD_initial_lr=10_batch_size=20_seq_len=35_hidden_size=1500_num_layers=2_dp_keep_prob=0.35_0
seed 1111
seq_len 35

Regarding the results **GRU** performs better than the other two architectures.

Recent Trends in Deep Learning Based Natural Language Processing

Tom Young[†≡], Devamanyu Hazarika[‡≡], Soujanya Poria[⊕≡], Erik Cambria[▽*]

[†] School of Information and Electronics, Beijing Institute of Technology, China
[‡] School of Computing, National University of Singapore, Singapore
[⊕] Temasek Laboratories, Nanyang Technological University, Singapore
[▽] School of Computer Science and Engineering, Nanyang Technological University, Singapore

Abstract

Deep learning methods employ multiple processing layers to learn hierarchical representations of data, and have produced state-of-the-art results in many domains. Recently, a variety of model designs and methods have blossomed in the context of natural language processing (NLP). In this paper, we review significant deep learning related models and methods that have been employed for numerous NLP tasks and provide a walk-through of their evolution. We also summarize, compare and contrast the various models and put forward a detailed understanding of the past, present and future of deep learning in NLP.