



UNIVERSIDADE D
COIMBRA

Adriano Pinto Durão

**RECURRENT NEURAL NETWORKS FOR SMILES
GENERATION
QUANTIZATION STUDY**

**Dissertação no âmbito do Mestrado em Engenharia
Electrotécnica e de Computadores Área de Especialização
em Computadores orientada pelo Professor Doutor
Gabriel Falcão Paiva Fernandes e apresentada à Faculdade
de Ciências e Tecnologia, Departamento de
Engenharia Electrotécnica e de Computadores.**

Julho de 2022

Acknowledgments

I want to thank my thesis supervisor, Professor Gabriel Falcão Paiva Fernandes for accepting my project, for his constant availability to help me in any regard and all the advices given during the execution of this dissertation.

This work was partially supported by Instituto de Telecomunicações and Fundação para a Ciência e a Tecnologia, Portugal, under grants UIDB/EEA/50008/2020 and EXPL/EEI-HAC/1511/2021.

I thank Professor Bernardete Ribeiro and Professor Joel Arrais for their help during the first stages of my thesis and Maryam Abbasi for providing and helping me get started on the program she developed as part of her work in Department of Informatics Engineering, which was the base for the work I developed.

I would like to also thank University of Coimbra and the Department of Electrical and Computer Engineering (DEEC) for providing me with the education necessary to be able to complete this project.

Finally I would like to thank my family for giving me the opportunity to complete my studies and focus on my education and personal growth, without them I would never be in this position.

Abstract

Machine Learning (ML) has possibly become the biggest research topic in computer science, aiming to improve how tasks are performed and automate computer learning, making use of the ever increasing data available on every major subject, from economics to health. Due to the high number of computations necessary to train ML models it is a very energy intensive process.

Optimizing the training process leads to faster and less costly models and allows them to run on less powerful devices. By running the models at reduced precision significant savings can be attained both in memory requirements and power consumption. Most optimization techniques focus on training the network at float precision, converting the model to 16 or 8 bits and running inference on the converted model.

This study focuses on the effects of applying quantization during training, making use of the QKeras library. It offers the flexibility to choose the precision used by the model by defining the number of bits at each layer. A class of Neural Networks (NNs) denominated Recurrent Neural Networks (RNNs) will be the focus of the study, comparing the performance of 3 of the most used algorithms, Simple RNN, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The models were trained on a selection of SMILES, a form of line notation for molecular information, from the PubChem database. Quantization performance was compared to their float equivalent for several combinations of tuneable parameters. The goal of the program used for testing is to generate a large number of novel SMILES, facilitating the process of Drug Discovery that is traditionally very expensive and difficult.

By understanding how the behavior of quantized networks deviates from the regular model, in relation to the parameters used, the process of choosing whether to quantize a model and to which degree becomes more efficient. This study was able to achieve good performance even on 4 bit models making use of LSTM and GRU layers and concluded that Simple RNN quantization is not worth it.

Keywords

LSTM, Neural Networks, Machine Learning, Quantization, SMILES

Resumo

ML tornou-se, nos últimos anos, num dos principais tópicos de pesquisa em ciência de computadores, tendo como principal objetivo melhorar a forma como as tarefas são executadas e automatizar a aprendizagem por computador, fazendo uso da maior disponibilidade de dados sobre as mais variadas áreas, da economia à saúde. Devido ao elevado número de cálculos necessários para treinar modelos de ML, este é um processo que implica elevados custos energéticos.

A otimização do processo de treino leva à obtenção de modelos mais rápidos e menos dispendiosos, permitindo que eles sejam executados em dispositivos menos potentes. Ao executar os modelos com precisão reduzida, é possível obter poupanças significativas nos requisitos de memória e no consumo de energia. A maioria das técnicas de otimização concentra-se em treinar a rede com precisão float, convertendo a seguir o modelo para 16 ou 8 bits e executando a inferência no modelo convertido.

Este estudo foca-se nos efeitos da aplicação de quantização durante o treino, recorrendo à biblioteca QKeras. Esta oferece a flexibilidade de escolher a precisão usada pelo modelo, definindo o número de bits em cada camada. A classe de Redes Neurais denominada Redes Neurais Recorrentes será o foco do estudo, comparando o desempenho de 3 dos algoritmos mais utilizados, RNN simples, LSTM e GRU. Os modelos foram treinados numa seleção de SMILES, uma forma de notação de linha com informações moleculares, retiradas do banco de dados PubChem. O desempenho dos modelos quantizados foi comparado ao seu equivalente float para várias combinações de parâmetros ajustáveis. O objetivo do programa usado para teste é gerar um grande número de novos SMILES, facilitando o processo de descoberta de novos fármacos, tradicionalmente muito caro e de difícil execução.

Ao entender como o comportamento das redes quantizadas se desvia do modelo regular, para cada combinação de parâmetros utilizados, o processo de escolher quantizar um modelo ou não e em que grau o fazer torna-se mais eficiente. Este estudo conseguiu obter um bom desempenho mesmo em modelos de 4 bits, fazendo uso de camadas LSTM e GRU e concluiu que a quantização de modelos usando de camadas RNN simples leva a uma elevada degradação de desempenho.

Palavras-Chave

LSTM, Redes Neurais, Machine Learning, Quantização, SMILES

Contents

1	Introduction	1
1.1	Context	2
1.2	Motivation	2
1.3	Objectives	3
1.4	Structure	3
2	State of the Art	5
2.1	Neural networks: how it works	6
2.1.1	Artificial Intelligence	6
2.1.2	Machine Learning	7
2.1.3	Deep Learning	7
2.1.4	Neural Networks	7
2.1.4.A	Model	8
2.1.4.B	Layers	8
2.1.4.C	Activation functions	8
2.1.4.D	Training	9
2.1.4.E	Inference	9
2.1.4.F	Parameters	10
2.2	RNNs	11
2.3	LSTM	12
2.4	Gated Recurrent Unit GRU	13
2.5	Drug Discovery	14
2.6	Frameworks	15
2.6.1	Tensorflow	16
2.6.2	Keras	16
2.6.3	QKeras	16
2.7	Optimizations and quantization	17
2.7.1	Quantized training	17
3	Methods and Implementation of Recurrent Models and Quantization	19
3.1	QKeras quantization vs float32	20
3.2	SMILES Generation Model	21
3.2.1	Dataset	21
3.2.2	Encoding Data	22
3.2.3	Training Model	23
3.2.4	Generating Output	23

3.2.5	Validation of Output	23
3.3	QKeras implementation	24
3.4	SimpleRNN model A	24
3.5	SimpleRNN model B - QKeras	25
3.6	GRU model A	25
3.7	GRU model B - QKeras	25
3.8	LSTM model A	26
3.9	LSTM model B - QKeras	26
3.10	Error analysis	26
3.11	Hardware and System	27
4	Experimental Results	29
4.1	Simple RNN model	30
4.1.1	Simple RNN test 1 - Number of layers	30
4.1.2	Simple RNN test 2 - Dropout and Learning Rate	32
4.1.3	Simple RNN test 3 - Epochs and Number of training samples	34
4.1.4	Simple RNN - Final Analysis	36
4.2	GRU model	38
4.2.1	GRU test 1 - Number of layers	38
4.2.2	GRU test 2 - Dropout and Learning Rate	40
4.2.3	GRU test 3 - Epochs and Number of training samples	42
4.2.4	GRU - Final Analysis	44
4.3	LSTM model	45
4.3.1	LSTM test 1 - Number of layers	46
4.3.2	LSTM test 2 - Dropout and Learning Rate	47
4.3.3	LSTM test 3 - Dropout per layer and Learning Rate	52
4.3.4	LSTM test 4 - Epochs and Number of training samples	58
4.3.5	LSTM - Final Analysis	60
4.4	Recurrent models - Performance comparison	62
5	Conclusion	65
5.1	Conclusion	66
5.2	Future Work	66

List of Figures

2.1	Relations between commonly used terms	6
2.2	General configuration of NN layers and connections	8
2.3	Commonly used activation functions	9
2.4	Gradient estimation convergence as a function of batch size	10
2.5	Learning rate analysis as a function of loss per epoch	11
2.6	RNNs architecture and unfolded cells	11
2.7	LSTM architecture and equations	13
2.8	GRU architecture and equations	14
2.9	Nicotine molecule with corresponding SMILES representation	15
3.1	Example of QKeras quantization on activation functions	21
3.2	Sample of SMILES taken from the PubChem dataset	22
3.3	SMILES with corresponding representations	22
4.1	Simple RNN - Valid and Unique SMILES by number of layers	36
4.2	Simple RNN - Valid and Unique SMILES by dropout and learning rate	36
4.3	Simple RNN - Valid and Unique SMILES by number of samples and epochs	36
4.4	GRU - Valid and Unique SMILES by number of layers	44
4.5	GRU - Valid and Unique SMILES by dropout and learning rate	44
4.6	GRU - Valid and Unique SMILES by number of samples and epochs	45
4.7	LSTM - Valid and Unique SMILES by number of layers	60
4.8	LSTM - Valid and Unique SMILES by dropout and learning rate	60
4.9	LSTM - Valid and Unique SMILES by dropout per layer and learning rate	60
4.10	LSTM - Valid and Unique SMILES by number of samples and epochs	61
4.11	First test - Error graphic comparing performance of recurrent models	62
4.12	Second test - Error graphic comparing performance of recurrent models	62
4.13	Final test - Error graphic comparing performance of recurrent models	62
4.14	Sample of SMILES generated by the best 4 bit Simple RNN model	63
4.15	Sample of SMILES generated by the best 4 bit GRU model	63
4.16	Sample of SMILES generated by the best 4 bit LSTM model	64

List of Tables

3.1	Computing system	27
4.1	Simple RNN model: 1 st test training data	30
4.2	Simple RNN model: 1 st test SMILES validation data	31
4.3	Simple RNN model: 2 nd test training data	32
4.4	Simple RNN model: 2 nd test SMILES validation data	33
4.5	Simple RNN model: 3 rd test training data	34
4.6	Simple RNN model: 3 rd test SMILES validation data	35
4.7	GRU model: 1 st test training data	38
4.8	GRU model: 1 st test SMILES validation data	39
4.9	GRU model: 2 nd test training data	40
4.10	GRU model: 2 nd test SMILES validation data	41
4.11	GRU model: 3 rd test training data	42
4.12	GRU model: 3 rd test SMILES validation data	43
4.13	LSTM model: 1 st test training data	46
4.14	LSTM model: 1 st test SMILES validation data	47
4.15	LSTM model: 2 nd test training data - part 1	48
4.16	LSTM model: 2 nd test training data - part 2	49
4.17	LSTM model: 2 nd test SMILES validation data - part 1	50
4.18	LSTM model: 2 nd test SMILES validation data - part 2	51
4.19	LSTM model: 3 rd test training data - part 1	53
4.20	LSTM model: 3 rd test training data - part 2	54
4.21	LSTM model: 3 rd test SMILES validation data - part 1	55
4.22	LSTM model: 3 rd test SMILES validation data - part 2	56
4.23	LSTM model: 3 rd test SMILES validation data - part 3	57
4.24	LSTM model: 4 th test training data	58
4.25	LSTM model: 4 th test SMILES validation data	59

Acronyms

LSTM Long Short-Term Memory

RNN Recurrent Neural Network

GRU Gated Recurrent Unit

ML Machine Learning

DL Deep Learning

NN Neural Network

AI Artificial Intelligence

SMILES Simplified Molecular-Input Line-Entry System

1

Introduction

1.1 Context

Recurrent Neural Networks (RNNs) [1] are a class of Machine Learning algorithms that perform especially well on sequential data. Long Short-Term Memory (LSTM) networks [2] are a type of RNN that solves many problems of simpler RNN configurations, like exploding gradients. Other network architectures have been proposed to solve the same issues while having lower computational requirements than LSTM's, as is the case of the Gated Recurrent Unit (GRU) [3].

Drug Discovery is a process that involves a large range of scientific disciplines, part of that process consists of generating new molecules with the aim of developing new compounds, inserted in the process designated as de novo drug design [4]. Deep learning has a lot of potential in this area, with RNNs and its variations being used in state of the art models used to generate novel molecules [5]. Simplified Molecular-Input Line-Entry System (SMILES), a text based notation created in the 1980s by David Weininger [6], is often used to represent the molecular structures, making RNNs the common choice for the task as they are well equipped to process this data format and deal with long term dependencies.

Quantization approaches aim to reduce how computationally expensive it is for models to perform Machine Learning tasks. Models typically run with 32 bit floating point precision, by training and running inference on lower precision an increase in energy efficiency and lower memory usage can be achieved, without losing significant performance [7]. Some typically used libraries like Tensorflow already provide tools, like Tensorflow Lite, that can perform some type of quantization, but do not provide many options when it comes to choosing the number of bits and many only perform post training quantization by converting the trained model to a quantized version. Qkeras [8] is a library that extends the Keras library, providing replacements for some of its layers, enabling the creation of a quantized model that performs the main operations in lower precision. It allows the programmer to specify the number of bits and the use of different precisions in different blocks of the model [8].

1.2 Motivation

The exponential increase in computing power, advances in memory technology and the availability of data have been the main factors enabling the latest advances in Machine Learning (ML) [9]. As the performance available increased, so did the complexity of the problems being solved, making efficiency a constant problem throughout the years. Hardware acceleration has been widely used for many purposes, like video and sound

cards. Given the increasingly employment of Artificial Intelligence (AI) in many software applications, AI accelerators are being made available even in mobile devices destined for mass market, where power usage is of great importance [10].

By studying how profoundly the precision of AI models can be reduced at each operation, better hardware can be developed for commercial and research applications, leading to greater energy savings with comparable performance when applied to the same tasks [11]. If the best quantization parameters for a model can be found by software emulation *a priori*, the development of dedicated hardware, which is a time-consuming process and has significant costs associated, can be made more efficiently.

1.3 Objectives

The goal of this study is to assess how the network performs in terms of accuracy and prediction capabilities when training with reduced precision, studying how using different building blocks of RNNs affect the model performance. With Simple RNN, LSTM and GRU layers being studied. To achieve that, the Keras model, built upon Tensorflow, is adapted into employing reduced precision by making use of the QKeras library. The layers of the original model being adapted to the specifications of the new library used.

By assessing the number of precision bits that still perform the required task without much degradation and which models should be kept at higher precision, the most efficient solutions can be found. The general approach of using full precision or post training quantization, despite working well for many use cases, is more limited in nature. The design of more cost-effective hardware architectures for this application can be made easier by building upon and analysing the results of this study.

1.4 Structure

The starting point for the thesis were the materials provided by a project carried out by a research team from the Informatics Engineering Department from University of Coimbra. These materials were adapted to comply with the QKeras library specifications. The description of the networks and different RNN configurations were the other resources used as starting point in developing this study.

Chapter 2 explains and introduce the general topics being studied by this thesis, as well as explaining the concepts required for understanding it. The first part of this chapter will contextualize RNNs in the world of AI and explain how the different models work. The most important frameworks and libraries used are also introduced during this chapter.

Chapter 3 describes the experimental setup, the overall dependencies of the program

1. Introduction

used and how it is modified to accommodate the changes required for reduced precision training using the specified tools, namely QKeras. The focus will be on the software implementation and the study of the accuracy degradation applied on the CHEMLAB dataset as it is a good representation of a less studied but interesting use case of RNN's when compared to Natural Language Processing applications.

The results are analyzed and displayed in section 4. Prediction accuracy will be the main metric used to assess the quality of the training process. As Drug Generation is the program's goal the created SMILES will be compared to available databases and validated to determine the quality of the predictions.

The last section, section 5, will be a reflection on the results obtained and how the study can be further improved and validated by future work.

2

State of the Art

2.1 Neural networks: how it works

A number of different terminologies are used when referring to Neural Networks (NNs). Each of those are often used to refer to the same principles as they represent closely related concepts. NNs are a subfield of Machine Learning (ML) and ML is itself a subfield of Artificial Intelligence (AI). Deep Learning (DL) simply refers to configurations making use of NNs and containing more than one hidden layer. This relation between different terms can be visualized in figure 2.1.

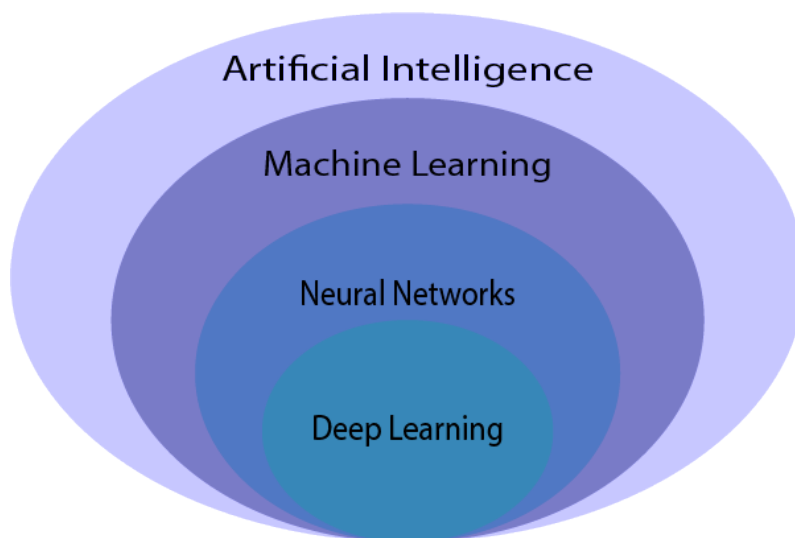


Figure 2.1: Relations between commonly used terms

2.1.1 Artificial Intelligence

Artificial Intelligence, as the name suggests aims, to emulate how intelligence works. In order to effectively learn and perform problem-solving, several mathematical approaches are applied depending on the nature of the data being processed. It is the broadest term used and can be divided in different ways. When dividing it based on their similarity to the capabilities of our brain we define it as Reactive AI, Limited Memory AI, Theory of Mind AI and Self-Aware AI [12].

Reactive AI is the simplest and oldest and can only respond to a limited combination of inputs. Limited Memory AI can learn from historical data in order to make decisions. Most DL algorithms are Limited Memory AI. Theory of Mind and Self-Aware AI are still conceptual and are considered the future of the subject, aiming to perform the same tasks as a human mind by having a concept of real understanding and in the latter actually being able to make its own decisions and behaving like an individual.

From a technological point AI is divided into 3 levels: Artificial Narrow Intelligence, Artificial General Intelligence, and Artificial Superintelligence. All of the currently used

models fall into the Artificial Narrow Intelligence category [13].

2.1.2 Machine Learning

Machine Learning as a branch of AI is a method of analysing data and creating algorithms that improve from the data being processed in an automated manner. ML approaches are usually divided into Supervised Learning, Unsupervised Learning and Reinforcement Learning [14].

Supervised Learning uses labeled data to map inputs to outputs by adjusting some internal weights to represent the relations between them and correctly predict the desired output. Most DL algorithms fall into this category. Unsupervised Learning does not make use of labeled data and instead used mathematical processes to find patterns in data and solve the problems, that offers a big advantage in the sense that correct labeling of data is a time consuming process and can be unfeasible for some problems. Some models make use of both Supervised and Unsupervised approaches and are referred to as Semi-Supervised. Reinforcement Learning aims to find the actions that maximize the payoff in the particular situation at study by interpreting its environment and, though trial and error, make a sequence of decisions to find the best solution.

2.1.3 Deep Learning

Deep learning is a subfield of ML that relies on Neural networks having multiple hidden layers as part of their configuration. Different distinctions can be made and many individual architectures exist.

The most popular types of Deep Neural Networks currently in use are: Convolutional Neural Networks (CNN) [15] and Recurrent Neural Networks (RNN) [1], with a large array of configurations derived from their core principles, making them the most thoroughly researched. Other popular algorithms include: Generative Adversarial Networks (GANs), Radial Basis Function Networks (RBFNs), Multilayer Perceptrons (MLPs), Self Organizing Maps (SOMs), Deep Belief Networks (DBNs) and Restricted Boltzmann Machines (RBMs) [16].

2.1.4 Neural Networks

Neural Networks, as a part of ML, mimic how the human brain works [17]. In mathematical terms, NNs make a probabilistic analysis of the data to extract useful features and learn the underlying patterns that define them. Akin to the brain we define the gateways where data passes through as neurons. Neurons receive one or more signals as input, either from the data set or from other neurons connected to it. They aggregate knowledge as

2. State of the Art

a set of weights, usually representing how important each feature will be to the desired output and then proceed to use an activation function to introduce non-linearity, allowing for more complex patterns to be learnt [17].

2.1.4.A Model

The model, when referring to NNs, consists of a specific combination of layers and activation functions, applied to a specific data set. The model will have interconnections between layers who assign the weights, while performing the computation pipeline required for the task.

2.1.4.B Layers

Neural networks typically have 3 types of layers: Input layer - Each unit from the input layer drives its value to the neurons of the first hidden layer, taken from the initial data. Hidden layers - As the name suggests hidden layers are not directly visible, they perform the mathematical computations between the input and output layers, creating a set of weights representing the associations between input/output. Multiple hidden layers can be used depending on the task [18]. Output layer - The last layer in the pipeline, the output layer simply constructs the results for the given inputs, based on the hidden layers.

Figure 2.2 shows how the 3 types of layers connect to each other.

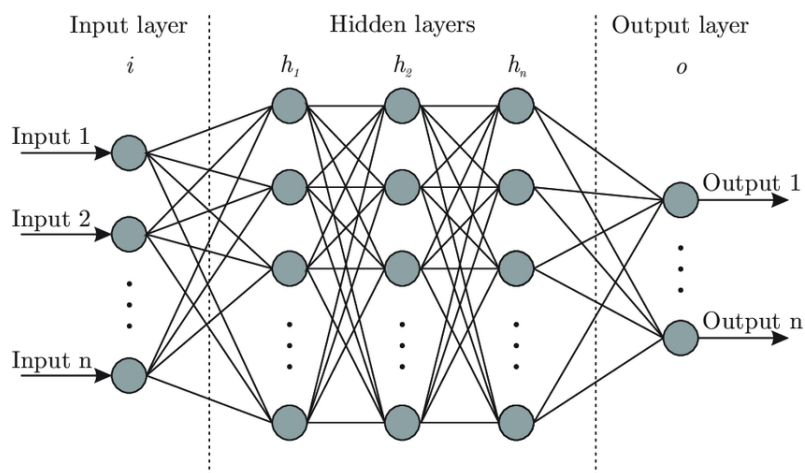


Figure 2.2: General configuration of NN layers and connections, taken from [19]

2.1.4.C Activation functions

As previously described, activation functions introduce non-linearity, to achieve this, several mathematical functions can be used, and without them the complexity of tasks that can be performed becomes limited since the model will act as a linear regression in

its absence. Some of the most commonly used functions are: sigmoid, Relu and softmax [20].

The use of the right activation function can help solve several problems that can arise in some models, like vanishing gradients. Figure 2.3 represents some of these activation functions in graphical and equation form.

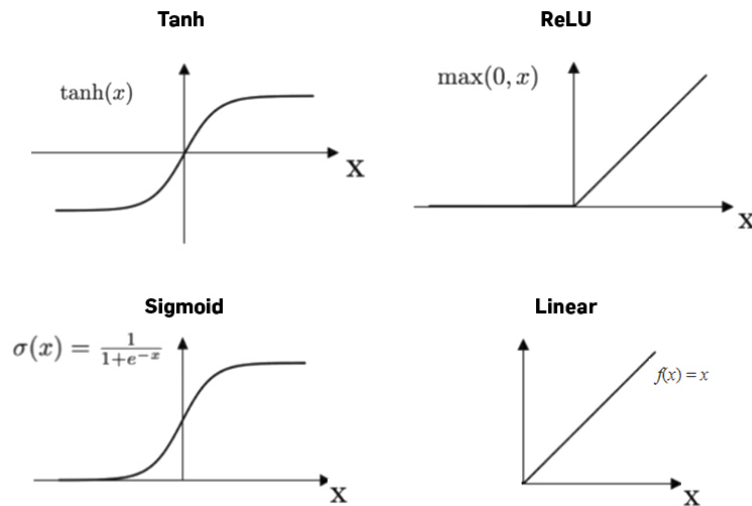


Figure 2.3: Commonly used activation functions, taken from [21]

2.1.4.D Training

In order to feed the data into the network a lot of care is required, otherwise the training process wont lead to the desired results or, worst case, it becomes impossible altogether. Most data needs to be prepared and correctly formatted before the training phase can begin, that usually encompasses creating a dictionary, formatting it in a consistent way and making sure its relevant to the problem. When training is supervised, data labeling becomes necessary to achieve proper results.

2.1.4.E Inference

Once the training process is completed, in order to extrapolate a result from the model, inference is performed. During inference the model outputs a prediction based on the weights and biases of the previously trained network. The quality of results will vary depending on the quality and representativity of training data in relation to the desired solution, as well as adequate model usage.

2. State of the Art

2.1.4.F Parameters

The parameters are a sort of fine tuning of the network and encompass things like dropout, batch size, number of epochs, number of units, temperature and learning rate.

Dropout is used to reduce overfitting. It involves dropping a percentage of the units, that is, setting the weights to 0 or a random value.

Batch size is a way to divide the training samples. Training will be done for the first batch of samples before moving to the next and so on. Dividing the training requires less memory than using all the samples. For most applications the batch size cant be too small or the gradient estimation will be less accurate leading to poor convergence, this behavior can be visualized in figure 2.4.

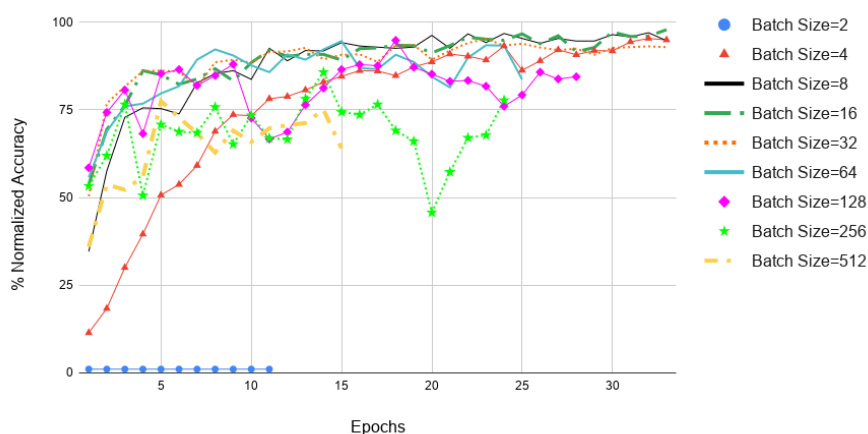


Figure 2.4: Gradient estimation convergence as a function of batch size, adapted from [22]

Number of epochs defines how many times training will do a full pass over the whole data. During each epoch the model runs through all the batches, drops some weights at the end and repeats until the final epoch.

Units are the number of neurons, that is, how deep the layers are in the network, more complex data requires a higher number of units or more layers.

Temperature introduces randomness in predictions, it changes the output distribution by making the final probabilities less rigid. Decreasing the confidence of the algorithm is useful in avoiding repetitive results.

Learning rate controls how fast the model is adapted to the data, by modifying how much the weights change in response to the loss gradient. If its too small the network will take a long time to learn, if its too big it will have unstable behavior. Learning rate is often adjusted during training by specific algorithms to stay in the optimal curve and prevent over and under fitting. Figure 2.5 displays how looking at the loss per epoch during training can help determine problems with the learning rate.

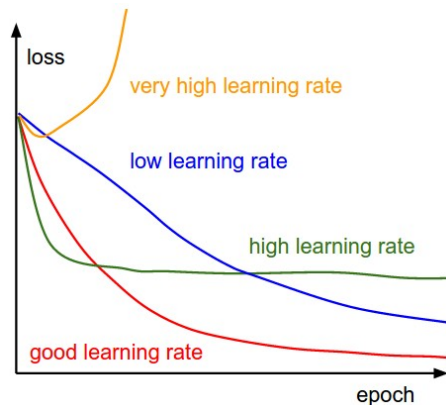


Figure 2.5: Learning rate analysis as a function of loss per epoch, taken from [19]

2.2 Recurrent Neural Networks (RNNs)

Feedforward Neural Networks are unable to retain information about former events in a sequence. RNNs allow for data to persist over time by adding a feedback loop to their cell, who gets updated at every time step as the sequence gets processed and becomes part of the function of the next time step, along with the new input, therefore retaining the temporal properties of the data in the hidden state weight's [23]. RNNs can handle variable lengths due to them being treated as a different number of time steps instead of a variable sized vector due to the feedback process affecting on the same set of weights, at most some data padding will be necessary.

Long term dependencies are handled by recurrently updating their internal cell which allows the new cell to retain information from previous states, condensed in the weights of the hidden layer. By having dependencies on previous cells RNNs are also able to gain information about sequence order, a very important requirement for language processing.

Other than language processing RNNs also have applications in machine translation (encoder-decoder), music generation, sentiment classification over sequences and many other problems, as is the case of De-novo drug design studied in this thesis.

Figure 2.6 gives a simple representation about the flow of a simple RNNs layer.

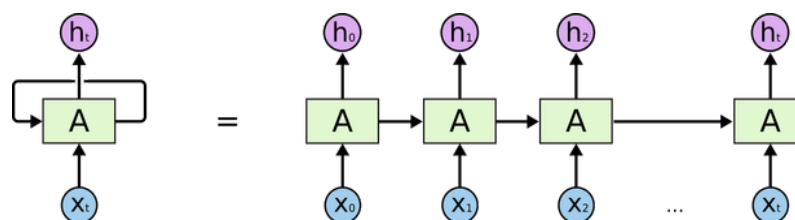


Figure 2.6: RNNs architecture and unfolded cells, taken from [24]

Where x_t is the input at timestep t ; A is the hidden state; h_t is the output at timestep t .

2. State of the Art

In order to train RNNs a form of the backpropagation algorithm is used, designated as backpropagation through time (BPTT) [1]. The derivatives of the loss are taken with respect to every parameter after the forward pass, but in the case of RNNs the overall loss is represented by the sum of each loss at every time-step. That means the loss will be backpropagated through each individual time-step and then backwards through time across the time steps to the beginning of the sequence.

Between each time step, to get the gradients to flow back in time, matrix multiplications involving the weight matrix are necessary, meaning that to compute the gradient of the first cell a lot of multiplications are required as well as gradient computations. This can lead to two main problems: Exploding gradients if values are larger than 1, as successive multiplications can lead to very large numbers. Vanishing gradients if they are smaller than 1 as it can lead to numbers very close to 0, which is a problem to acquiring long term dependencies as it leads to bias towards the short-term. To combat those problems activation functions like ReLU can be used to prevent shrinking gradients. A proper initialization of the weights and bias can also help tackle those problems.

More complex architectures based on RNNs are often used as they fix the mentioned problems and offer other advantages, the one most often used is called Long Short-Term Memory (LSTM), which adds a cell state and gates to control the flow of information. Other architectures like Gated Recurrent Unit (GRU) are also able to solve these problems as they are similar to LSTM's [23].

2.3 Long Short-Term Memory (LSTM)

A more elaborated and useful type of Recurrent Neural Network (RNN) is the LSTM network, initially described in Hochreiter S. and Schmidhuber J.'s 1997 paper "Long Short-Term Memory" [2]. It improves on RNN's previous configurations by using a more complex recurrent unit with the added capability of controlling the flow of information through its cells, regulated using several structures denominated gates. This allowed models to improve significantly and learn on tasks involving thousands of steps, unlike RNN's they don't learn only from recent events, they can also look at context in much longer sequences to gather relevant information.

The main goal behind their development was exactly that, removing the long-term dependency problems of RNN's. The ability to control information in their cell state is the key to their greater performance. The cell state works in a way that complements the hidden state by allowing information to flow easily between steps. Storing those long-term dependencies that regular RNN are unable to comes at the cost of increased

complexity.

Gates are composed of a sigmoid layer, represented by σ , which outputs numbers between 0 and 1, and a pointwise multiplication operation. The ability to control the information in the cell state is regulated by the 3 gates of an LSTM cell. In the sigmoid a output of 1 means that information flows in its entirety and 0 means no information gets through. The gates work as 3 steps in the flow of information:

The first step consists in forgetting less relevant information learned in past states, in the forget gate layer. Taking the last hidden state, h_{t-1} and the current input, x_t , it outputs a number between 0 and 1, as it is a sigmoid layer. This is done in reference to each number in the last cell state C_{t-1} .

The second step is slightly more complex and requires two parts. First another sigmoid layer known as input gate layer decides which is the relevant new information by defining which values will be updated. Next, the vector of new prospect values, \tilde{C}_t , is created by a tanh layer. By combining those two steps the update to the cell state is created. To update C_{t-1} into C_t we multiply the old state by f_t and then we add $i_t * \tilde{C}_t$.

The last step consists on deciding what to output, which will be based on a filtered version of the cell state we calculated. First a sigmoid layer denominated output gate decides the parts of the cell to output. Next the cell state goes though a tanh and multiplies it by the output of last sigmoid to only output the parts it decided were relevant.

This is more easily understandable by looking at the equations and corresponding scheme given in figure 2.7.

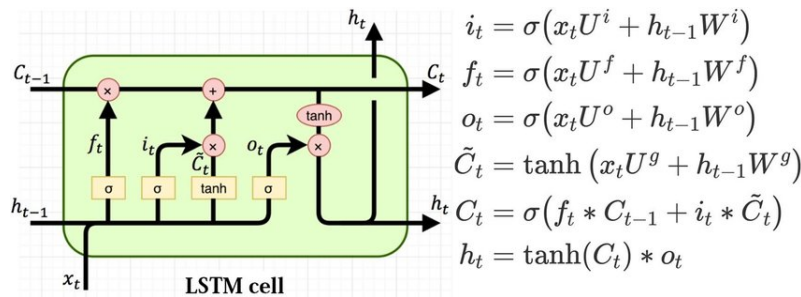


Figure 2.7: LSTM architecture and equations, taken from [25]

2.4 Gated Recurrent Unit Gated Recurrent Unit (GRU)

GRU, introduced by Cho et al.[2014] [3], are deeply inspired by the LSTM architecture. By being very similar to LSTM networks the results produced by both networks tend to also be very similar. GRU's main difference and advantage lays in the fact they have fewer parameters when compared to traditional LSTM networks. Like LSTM they also use gates to solve gradient problems and control the flow of information, but where

2. State of the Art

LSTM uses 3 gates GRU only uses 2, the update gate and the reset gate.

The first step in GRU is calculating the update gate, z_t , which determines how much of the past information from former time steps is relayed to future steps, working as a combination of the input and forget gates from LSTMs, and in accordance with them also uses sigmoid as the squashing function. The GRU also differs from LSTM in the sense that memory cell state is merged with the hidden state.

The other gate used is termed reset gate, r_t , and is used to determine how much past information should be forgotten, it is very similar to the update gate, only differing in the weights being used and where it applies. The output is therefore affected by the gates, the new memory, \tilde{h}_t , uses the reset gate to store the pertinent information from the past and uses tanh as activation function.

As a last step the network calculates h_t , which uses the update gate to decide what to use from the new memory and from previous steps h_{t-1} to store the information for the current element.

Figure 2.8 summarizes this process and shows the operations performed.

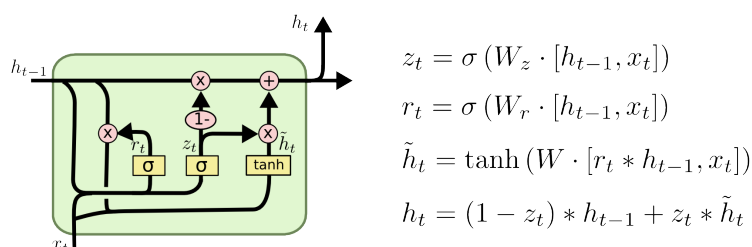


Figure 2.8: GRU architecture and equations, taken from [24]

2.5 Drug Discovery

The process of bringing a new drug candidate to market is very complex. Starting from discovery and development of a compound with the required properties. Followed by rigorous clinical research, including animal and human trials, to evaluate safety and effectiveness. Ending in approval by the regulating agents, the European Medicines Agency (EMA) in Europe and the Food and Drug Administration (FDA) in the United States being the main entities carrying out this regulatory process.

Carrying out this process is very slow and expensive, therefore it's desirable that new drug candidates have the desired molecular affinity to the target. The number of molecules to be considered as drug candidates is estimated to be in the order of 10^{60} candidates [26].

Computational methods can help facilitate this process by narrowing down the number of candidates and generating molecules with the desired properties without *in vitro* experiments [5].

In the process of generating new molecules one effective strategy is to use a form of line notation denominated Simplified Molecular-Input Line-Entry System. LSTMs are well equipped to deal with this data representation and are among the best performing models for carrying out this task [5]. An example of a well known molecule and corresponding SMILES representation is displayed in figure 2.9.

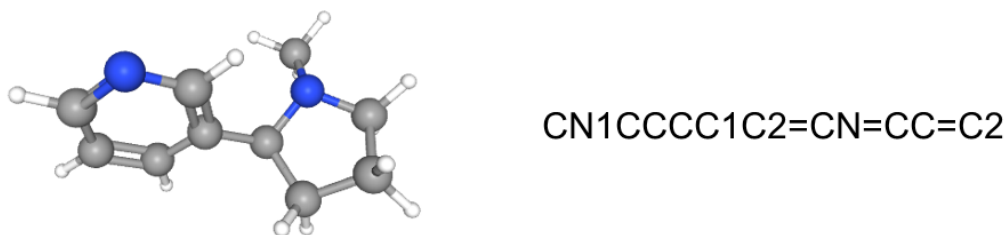


Figure 2.9: Nicotine molecule with corresponding SMILES representation. Image sourced from [27].

2.6 Frameworks

Frameworks make the job of building a Machine Learning (ML) model a much more straightforward task, by providing all the building blocks of the commonly used and most well defined Neural Network (NN) architectures they make building a network from scratch a task reserved only for specific research when trying to create new models that differ substantially from the ones the framework already offers as part of its ecosystem.

Most commonly used frameworks are also open source meaning new layers and optimizations can be achieved by extending their functionalities, without having to replicate the segments they have in common with established networks. This allows the task of solving a problem using NN to be focused on the type of problem being solved, the layers the NN needs in order to optimally solve the problem, the data that's available for training the required network and in the task of finding the best parameters for the model, not on mathematical details.

By not having to focus on programming every detail of the networks and abstracting many of the mathematical components, it makes creating Artificial Intelligence (AI) models a more accessible task. This allows for models to be created much cheaper with a good degree of optimization still available for applications.

2. State of the Art

2.6.1 Tensorflow

As of 2022, the most popular framework used to build AI applications is Tensorflow, a open source platform created by Google, offering a broad ecosystem of libraries, tools and a vast array of community resources to learn from [28].

Using Tensorflow researchers are able to build state-of-the-art models and developers can construct applications that make use of ML's advantages in a easier way. Tensorflow offers a high degree of optimization and allows for distributed training on multiple GPUs, TPUs, or machines [28].

It is built on C++, Python and CUDA and mainly supports C/C++ and Python via its API. This makes the library compatible with other Python libraries and their integration easy [28].

2.6.2 Keras

Keras was launched in 2015 and created primarily by François Chollet, a software engineer and AI researcher from Google [29].

Keras is a Python Deep Learning, high-level API, running on top of Tensorflow. It enables fast deployment of ML applications by providing a simple and consistent API and providing extensive documentation. It's highly scalable on multiple devices and makes the implementation process fairly simple [29].

By being user friendly, modular and easily extensible it has become one of the best choices for deep learning courses and many universities and scientific organizations.

Its highly connected to the Tensorflow ecosystem and covers every step of the development process. It's highly flexible on keeping low-level research possible while using abstraction whenever possible for convenience and faster experimentation [29].

2.6.3 QKeras

QKeras is a quantized extension to Keras, initially released in 2020 and developed by a team from Google based on the work described by "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors" [8]. It provides replacement layers for Keras layers that creates parameters and activation layers, as well as those who perform arithmetic operations [30].

This allows for quick creation and easy experimentation of quantized versions of neural networks, and better options when it comes to using mixed precision when compared to solutions like Tensorflow Lite. It follows Keras design principles of being user friendly, modular and easy to extend [30].

2.7 Optimizations and quantization

ML algorithms require a lot of trial and error, training a model comprises initiating the matrix with some values and comparing them to the correct answer, performing successive approximations to learn the underlying patterns required for the model to have decent accuracy in their task even if the input is somewhat different than the training set.

Neural Networks perform a huge amount of operations, mostly matrix multiplications and additions, which require a lot of computational power due to the size of the data structures. This high resource requirement leads to a high cost of training a NN both in terms of power consumption and upfront costs for adequate hardware. Several approaches are employed in dealing with those inefficiencies. Using the best model is the first one, as different models are better equipped at dealing with certain data, making the choice of the right one of utmost importance.

Using the best training algorithm for the model also leads to a more optimized solution as different requirements must be met depending on the data, in terms of speed, memory and precision. Finally, by using lower precision (resulting in a smaller model) the model can be made more efficient without significantly degrading the precision and allowing running it with less powerful hardware.

ML models are traditionally trained using 32 bit float precision to represent their respective weights, but not every task requires such high precision. One approach used to mitigate that is to train using full precision and then convert the weights to a lower bit representation and perform inference on that reduced precision. This allows for slower hardware to be usable for inference, having special importance when edge devices are the use case, as in many recent mobile applications like Augmented Reality, which are computationally heavy without the use of AI.

Training with lower precision has to be done more carefully, as very small differences can compound and prevent a model from achieving the same performance as when using regular float 32. Due to the size of the data structures when training over large datasets every bit counts and even small reductions can lead to great improvements in hardware performance and energy usage with a fully optimized model.

2.7.1 Quantized training

Training NN using less bits usually implies a trade-of between accuracy and speed. Not all use cases will allow for reduced precision as learning their defining features might require full precision to achieve the desired results.

Most approaches simply convert the weights from a pre trained model into lower precision to run inference but it could be desirable to be able to also train NN on reduced

2. State of the Art

precision, allowing training to run on lower power devices and using cheaper hardware. Despite that, many application scenarios can benefit from finding the best balance in that trade-off, as it can mean being usable in edge and low power devices without much quality degradation or to simply use less resources.

To achieve that an approach where only some layers get simplified can lead to the most desirable outcome by keeping only the most important operations running at higher precision to speed up the process and still benefit from the increased complexity. Quantization is very application dependant and needs to be studied case by case.

3

Methods and Implementation of Recurrent Models and Quantization

3.1 QKeras quantization vs float32

The work carried out in this dissertation makes use of the quantization tools initially developed by Coelho et al. [8] from where the QKeras library originated, being currently maintained and extended mostly by a team of researchers from Google. QKeras works as a quantization extension for the widely used Keras API.

The goal of this study will be to use the tools provided by this library and make an assessment of the best parameters and quantization settings to achieve the best accuracy on a Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) based models. The model will be used for generating chemical compounds used as part of the process of developing new drug candidates, as it is a perfect candidate to showcase the advantages of the commonly used RNNs based models in a setting that achieves decent results even on the limited computation power of the setup being used.

The aim is to study how keeping the precision as low as possible affects the training process and inference quality. This should improve memory usage, decrease energy consumption and lower the latency of the Neural Network (NN) models being used when proper hardware is used. The use of these tools was motivated by other quantization libraries not offering the tools necessary to manually specify the number of bits for quantization, offering only a couple of options like 16bit and 8 bit quantization [8].

This setup also performs a form of quantized training, which is not as common practice as simply quantizing the weights from networks trained using full precision and running inference on those is the norm. This is, however, quite challenging to balance in an optimal way as the configuration options for the model increase significantly with each used layer and so the balancing of accuracy and quantization must reach a compromise in accordance to the required specifications of the task as compared to the float-32 model.

The main advantage of QKeras is however the minimal code changes required to modify Keras models to work in a quantized fashion and the ease of changing the precision used for the quantization, to allow for extensive testing in order to find those optimal parameters. The library converts the weights to lower precision at each step, but in order to run the mathematical operations the software requires converting it back to float due to software and hardware constraints. This means the performance can only be assessed by looking at the results from the inference stage.

Figure 3.1 depicts how quantization will constrain the number of possible values.

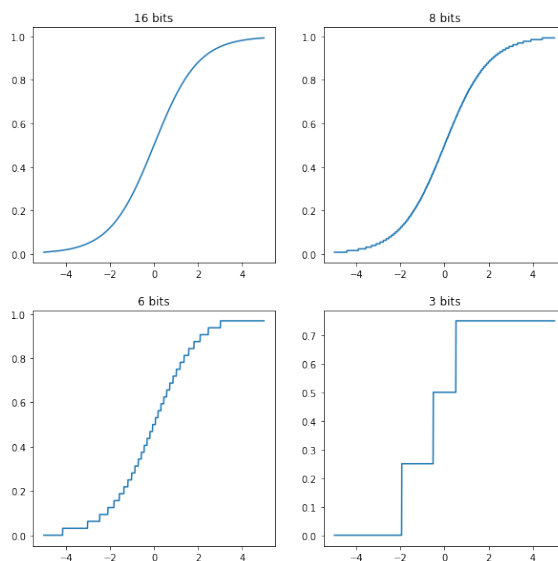


Figure 3.1: Example of QKeras quantization on activation functions, taken from [30]

3.2 SMILES Generation Model

The program used to create new SMILES will be very similar for all the different model configurations in this quantization study. Only the training model used differs between tests, using several combinations of Recurrent layers and activation functions, as well as the fine-tuning parameters, namely the dropout for each layer, learning rate, number of recurrent layers, number of training samples, epochs and quantization.

3.2.1 Dataset

The dataset used for training the model is acquired from the PubChem database, developed by the National Center for Biotechnology Information (NCBI) [31]. PubChem is a database of chemical molecules, mostly smaller ones, and for some of them, also a selection of assays (investigative procedures to assess interactions and relationships between a compound and some target to find properties like correct dosage for a medication) [31].

Those molecules are represented by their SMILES notation, which is a line notation representing the structure of the compound, meaning complex structures can be represented as a string of characters instead of their 3D structure [6].

As the full dataset is too large of for the testing system, as it contains a selection of 1.576.904 SMILES from PubChem. It is therefore used as the source from where the data samples for the study will be taken from. Smaller sub sets will be used since the memory requirements for using the full dataset are higher than the memory available on the hardware being used for experimentation.

3. Methods and Implementation of Recurrent Models and Quantization

A sample taken from the file containing the SMILES dataset is displayed in figure 3.2. In figure 3.3 the molecules described by SMILES 2396 and 2400 from figure 3.2 are represented in 2D and 3D.

```
2396 CN(C)N=Nc1ccc(C)cc1
2397 NC(=N)C1COC2CCCCC2O1
2398 Nc1nc2[nH]cnc(N)c2n1
2399 Cc1ccc(NC2CCCS2)cc1
2400 CN1CCN(CC1)c1ncccc1F
2401 Cc1ccc(CC2=NCCN2)cc1
```

Figure 3.2: Sample of SMILES taken from the PubChem dataset.

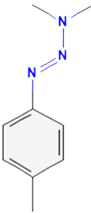
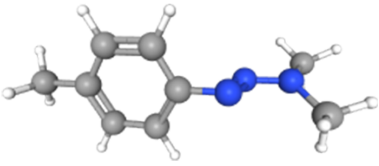
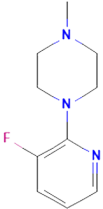

SMILES	2D	3D
<chem>CN(C)N=Nc1ccc(C)cc1</chem>		
<chem>CN1CCN(CC1)c1ncccc1F</chem>		

Figure 3.3: SMILES with corresponding representations. Images sourced from [27].

3.2.2 Encoding Data

To analyze the SMILES data sample, a table containing all the unique characters is necessary as the models needs a dictionary of possible symbols being used. This is done by tokenizing them, that is, defining indivisible portions representing the atoms and special characters used for the chemical connections.

Next it is necessary to do some padding as the SMILES strings should all be the same length. To have some normalization, since chemical compounds of different lengths are present, finding the largest one is the first step. After having that information the smaller ones are filled with characters used only as a placeholder (character A) so the network can iterate over all the samples in a uniform way.

Finally a one hot encode array is created, giving each unique character a unique representation in the array, giving it an effective label as the network requires numerical information. Using the values of the ASCII characters could be possible but simply using the value would be inefficient and the dictionary is much smaller.

3.2.3 Training Model

After having the encoded data, the data is prepared to be fed into the first layer of the network, with most models used being composed of 2 recurrent layers. By using Keras the only implementation necessary is to define the parameters to be used when the default ones are not desirable or are the subject of study.

The data is divided in training and validation data as it is necessary to use different data to verify if the network is learning the features in an unbiased manner. Checkpoints will be used to monitor the loss and accuracy during training and obtain a curve of the training process, allowing to better understand if overfitting or underfitting are happening. By better identifying those issues, fine-tuning the parameters can be better applied, when necessary.

3.2.4 Generating Output

To predict the new Smiles the weights calculated during training are used by the network for the inference stage. In order to start generating new SMILES the token representing the begin of a new compound (G is used) is given as starting input. From then on, the network goes symbol by symbol and predicts the next one, depending on the symbols it had predicted already.

At each iteration G will be given and the network will keep generating symbols until it predicts the symbol corresponding to the end of a compound (represented by E). Once E is predicted the next iteration will begin. After all iterations are completed, representing the number of molecules being generated, the resulting output can be validated against the databases, in order to assess the performance of the model.

3.2.5 Validation of Output

To validate the results the MolVs library is used. MolVs [32], a molecule validation and standardization tool, written in Python, using the RDKit chemistry framework [33], is used to verify if the created SMILES are valid and if they contain errors making them non valid or incomplete. The validator will also look for duplicate SMILES as the goal of the program is to generate valid and unique SMILES.

3.3 QKeras implementation

In order to Implement the model in QKeras the type of quantization function used needs to be specified for the weights and bias. By respectively defining the `kernel_quantizer` and `bias_quantizer` parameters for each layer in the model. If those are not specified no quantization is applied and the model runs in regular 32 bit precision instead.

Appropriate quantizer parameters need to be passed to the layer, `quantized_bits` is the one used for the models, and is how the number of bits is specified in `kernel_quantizer` and `bias_quantizer`. `Quantized_bits` accepts 3 values as parameters, bit width, integer bits (to the left of the decimal) and alpha if changing the absolute scale is necessary (which offers little advantage as all RNNs based models only work with values between -1 and 1).

In terms of mathematical operations, `quantized_bits` performs mantissa quantization, which is given by:

$$2^{int-b+1} \text{clip}(\text{round}(x * 2^{b-int-1}), -2^{b-1}, 2^{b-1} - 1)$$

where x is the input, b is the number of bits for quantization, and int is how many bits are to the left of the decimal

3.4 SimpleRNN model A

The first model tested made use of the SimpleRNN Keras layer [34]. As the results obtained from using this model are predictably less interesting than those obtained from the LSTM network the parameters chosen for testing were narrowed. Nevertheless it was included in the study to showcase how superior more complex configurations are in their predicting capabilities. Including it allows comparing how the complexity of the algorithm used affects the quantization performance, as it is the most basic RNN model present in the Keras library [34].

The parameters studied during the tests for the model based on SimpleRNN layers were:

- Number of recurrent layers: 1,2, 3 or 4 recurrent layers.
- Dropout: 0.3 or 0.5 dropout.
- Learning Rate: 0.001 or 0.005 learning rate.
- Number of training samples: 100.000 or 250.000 training samples.
- Number of epochs: 16 or 24 epochs.

With 250.000 being the highest number of training samples supported by the memory available on the computing system used.

The tests were divided in 3 parts: The first compared only the number of recurrent layers while keeping the other parameters the same. The second tested the different combinations of dropout and learning rate from the aforesaid values. The third used the best parameters from the first two studies and studied the effects of increasing the training samples and the number of epochs.

3.5 SimpleRNN model B - QKeras

Similar to the Keras version and as to give a direct comparison the same parameters were tested, for the same reasons as mentioned above. As the results attained by this model are unsatisfactory only 16 bit, 8 bit and 4 bit quantizations were tested.

As only a small number of valid SMILES could be generated by the 4 bits version, studying deeper quantization is therefore of little practical use. The results leading to that conclusion will be displayed in the next chapter.

3.6 GRU model A

Similar to LSTM both in configuration and performance, the GRU model [35] will be tested with the same parameters as the Simple RNN model, as it was a late addition to the study. This still enables a good assessment of eventual major performance differences and makes an adequate evaluation of the GRU finetuning process.

The tests were again divided in 3 parts: The first comparing the number of recurrent layers. The second testing the different combinations of dropout and learning rate. The third used the best parameters from the first two tests and studied the effects of increasing the training samples and the number of epochs in the network performance.

3.7 GRU model B - QKeras

The QKeras implementation will follow the previously used workflow and will test the same parameters as the float model. The tests demonstrate the differences in the performance obtained from the Keras and QKeras versions. The details about how each parameter affects the training process as the precision gets lower will be described in the next section where the results are analyzed.

The GRU QKeras model will be tested for 16 bit, 8 bit and 4 bit quantization.

3.8 LSTM model A

The most important model being tested made use of LSTM Keras layers [36]. Due to it being the best performing model of the three being tested, according to literature on models making use of the PubChem dataset it will have the most in depth testing.

The parameters studied when using LSTM layers were:

- Number of recurrent layers: 1, 2, 3 or 4 recurrent layers
- Learning rate: 0.001, 0.003, 0.005 or 0.1 learning rate.
- Dropout: 0.1, 0.3, 0.5 or 0.7 dropout.
- Number of training samples: 100.000 or 250.000 training samples.
- Number of epochs: 16 or 24 epochs.

For the LSTM model the tests were divided in four parts: The first compared the number of recurrent layers while keeping the other parameters the same, as in the RNN model. The second tested the different combinations of dropout and learning rate from the above values, keeping dropout the same for both layers. The third used different dropouts for each layer, using combinations of values that could lead to the optimal solution. The fourth used the best parameters from the first three studies and increased the training samples and the number of epochs, achieving the main results of the study.

3.9 LSTM model B - QKeras

For the QKeras implementation of the LSTM model the same parameters as the Keras version were tested, with some worse performing combinations being skipped during the third test as testing every possible combination for every number of bits would be too lengthy without a more powerful setup.

The first three tests were run for 16 bit, 8 bit, and 4 bit quantization. The fourth and final test also included 6 bit, 3 bit and 2 bit quantization versions of the model for more in depth analysis of performance when using the optimal parameters.

3.10 Error analysis

To observe the flow of the training process before inference is performed it becomes relevant to look at the accuracy and loss, taken after each epoch during training. Analyzing those parameters gives good information about how fast the network is learning and

at which point it plateaus. Looking at those parameters one can also tell if overfitting or underfitting is occurring, giving a good idea of the impact the finetuning is having and allowing for direct comparisons.

When the model has finished training the weights are saved and inference begins. By generating new Simplified Molecular-Input Line-Entry System (SMILES) of varied quality we get the most important metrics for assessing the success of the model. Those generated SMILES are evaluated for validity and uniqueness, giving the metrics used to compare the quality of the results and therefore find the best parameters for the model.

Simply using the mathematical accuracy and loss metrics from training wont give enough information as generating valid but different SMILES is the goal. The training parameters, despite giving a good idea of training success, have too much variability when it comes to result analysis.

3.11 Hardware and System

The computing system and software used during experimentation is represented in table 3.1.

Hardware
CPU: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz GPU: NVIDIA GeForce GTX 1060 6Gb (Laptop) RAM: 16Gb
Software
OS: Microsoft Windows 10 Pro Version 10.0.18363 Build 18363 TensorFlow version 2.5.0 Keras version 2.4.3 Qkeras version 0.9.0 Python version 3.6.12

Table 3.1: Computing system

3. Methods and Implementation of Recurrent Models and Quantization

4

Experimental Results

4. Experimental Results

4.1 Simple RNN model

The starting parameters were chosen based on the initial parameter values from the original LSTM model used in the SMILES generation program. The Simple Recurrent Neural Network (RNN) models used those values as a beginning point as it establishes a good middle ground between the commonly used values for all the studied parameters and achieves decent performance in the original model.

4.1.1 Simple RNN test 1 - Number of layers

The first test kept the number of samples at 100k, running for 16 epochs with a learning rate of 0.001 and dropout of 0.3 (30% per epoch). The effect the number of layers has on the quantization performance of the Simple RNN model was the focus, and generated the results displayed in tables 4.1 and 4.2. Table 4.1 displays the final training parameters, that is, the values of loss and accuracy taken after the last epoch. Table 4.2 displays the percentage of valid and unique SMILES among the 1000 generated during inference.

Simple RNN model - training data					
100k samples; 16 epochs; 0.001 learning rate; 0.3 dropout					
number of layers		float	int16	int8	int4
1	loss	0.0056	0.0058	0.0058	0.0078
	accuracy	0.8202	0.8119	0.812	0.7396
	val. loss	0.0085	0.0088	0.0091	0.013
	val. acc.	0.7291	0.7201	0.709	0.5674
2	loss	0.0055	0.0056	0.0056	0.0083
	accuracy	0.8212	0.8183	0.8164	0.7174
	val. loss	0.0084	0.0084	0.0084	0.0133
	val. acc.	0.7335	0.7336	0.7328	0.5559
3	loss	0.0055	0.0057	0.0058	0.008
	accuracy	0.8228	0.8182	0.7914	0.7383
	val. loss	0.0081	0.0085	0.0087	0.0128
	val. acc.	0.742	0.7225	0.7163	0.5758
4	loss	0.0055	0.0156	0.0183	0.0153
	accuracy	0.8217	0.5517	0.3745	0.5643
	val. loss	0.008	0.023	0.0231	0.0226
	val. acc.	0.7478	0.2624	0.2515	0.2515

Table 4.1: Simple RNN model: 1st test training data

Simple RNN model - SMILES validation data					
100k samples; 16 epochs; 0.001 learning rate; 0.3 dropout					
number of layers		float	int16	int8	int4
1	valid (%)	39	23.7	22.4	0.3
	unique (%)	37.4	23.2	20.3	0.3
2	valid (%)	44	30.8	21.4	1.2
	unique (%)	43.2	30.6	18.8	1.1
3	valid (%)	42.1	28.1	21.1	0.6
	unique (%)	40.2	27.9	19.7	0.5
4	valid (%)	49.1	0.1	0	0.3
	unique (%)	35.7	0.1	0	0.3

Table 4.2: Simple RNN model: 1st test SMILES validation data

Analyzing tables 4.1 and 4.2 some patterns can be inferred. There is a relation between the accuracy and loss during training and percentage of valid SMILES generated during validation. As expected, higher accuracy during training leads to a higher percentage of valid SMILES, but the relation is not linear. Observing the 2 layer model one can notice that The discrepancy in training metrics between float, 16 bit and 8 bit is minimal, yet the 8 bit version only achieves 21.4% of valid SMILES while the float model achieves 44%. Solely observing the training metrics is therefore not enough to assess performance, giving only a general idea if training was successful. The percentage of valid and unique SMILES generated will be the main metric being compared between models.

Using a more complex model with 4 layers a higher percentage of valid SMILES is generated in the float version, when compared to the 2 layers model, yet significantly less unique SMILES are created. The quantized versions perform even worse, creating less than 1% of valid smiles. This effect could be explained by the vanishing gradients problem, common in the most basic types of RNN configurations, as it becomes more likely to happen as the number of connections, and therefore matrix multiplications, increases.

Comparing the validation data from the float model to the quantized versions, the conclusion is that any form of quantization for the Simple RNN model using these combina-

4. Experimental Results

tions of parameters is not advised, the performance degradation is substantial. Comparing the validation data from the float model to the quantized versions reveals at best a 13% decrease in number of valid SMILES generated and at worse total inability to generate valid SMILES.

4.1.2 Simple RNN test 2 - Dropout and Learning Rate

The second test used 2 recurrent layers, the best performing of the first test, while also keeping the number of samples at 100k, running for 16 epochs. During this stage it was analyzed how an increase in dropout and learning rate would affect quantization performance. Table 4.3 displays the final training performance parameters for the float and quantized models. Table 4.4 displays the percentage of valid and unique SMILES created.

Simple RNN model - training data 2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.3	0.001	loss	0.0055	0.0056	0.0056	0.0083
		accuracy	0.8212	0.8183	0.8164	0.7174
		val. loss	0.0084	0.0084	0.0084	0.0133
		val. acc.	0.7335	0.7336	0.7328	0.5559
	0.005	loss	0.0066	0.015	0.0071	0.0073
		accuracy	0.7786	0.5751	0.6455	0.6277
		val. loss	0.0098	0.0302	0.0112	0.0116
		val. acc.	0.6789	0.2533	0.4987	0.4864
0.5	0.001	loss	0.0065	0.0078	0.0081	0.0077
		accuracy	0.7814	0.7447	0.7199	0.6582
		val. loss	0.0093	0.0121	0.0126	0.0122
		val. acc.	0.6964	0.5998	0.5778	0.51
	0.005	loss	0.0078	0.0081	0.035	0.023
		accuracy	0.7291	0.7207	0.0607	0.3456
		val. loss	0.0118	0.0125	0.0464	0.0351
		val. acc.	0.6079	0.58	0.0034	0.2458

Table 4.3: Simple RNN model: 2nd test training data

Simple RNN model - SMILES validation data 2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.3	0.001	valid (%)	44	30.8	21.4	1.2
		unique (%)	43.2	30.6	18.8	1.1
	0.005	valid (%)	12.6	0.2	0.5	0.7
		unique (%)	12.1	0.2	0.4	0.6
0.5	0.001	valid (%)	12.7	2.1	2.6	0.5
		unique (%)	12.6	1.7	2.5	0.5
	0.005	valid (%)	1.4	0.3	0	0
		unique (%)	1.2	0.3	0	0

Table 4.4: Simple RNN model: 2nd test SMILES validation data

Analyzing tables 4.3 and 4.4 it is again noticed that quantization for the Simple RNN model leads to big drops in performance. The models making use of higher dropout or learning rate lead to much worse results when compared to the base parameter values. In the quantized models the performance degradation is even more severe, being an order of magnitude worse than what is achieved using 0.3 dropout and 0.001 learning rate, except for the 4 bits version that generates around 99% of invalid SMILES even in the best case.

Increasing the learning rate seems to have a similar effect to increasing dropout for the float model. The quantized model seems to suffer more from a increase in learning rate than in dropout, but only to a small degree as both are significantly worse and generate few valid SMILES.

Any form of quantization for the Simple RNN model using this configuration is again not advised, the performance degradation is even more substantial than during the first test.

4. Experimental Results

4.1.3 Simple RNN test 3 - Epochs and Number of training samples

The final test kept using 2 recurrent layers, with a learning rate of 0.001 and dropout of 0.3, being the best performing parameters from tests 1 and 2. The tested parameters were then the number of epochs and training samples, increasing them to check if the network was being limited by a too small amount of samples or for not running for enough cycles of training, given by the number of epochs. Table 4.5 displays the final training parameters. Table 4.6 displays the percentage of valid and unique SMILES.

Simple RNN model - training data						
2 layers; 0.001 learning rate; 0.3 dropout						
epochs	samples		float	int16	int8	int4
16	100k	loss	0.0055	0.0056	0.0056	0.0083
		accuracy	0.8212	0.8183	0.8164	0.7174
		val. loss	0.0084	0.0084	0.0084	0.0133
		val. acc.	0.7335	0.7336	0.7328	0.5559
	250k	loss	0.0053	0.0055	0.0056	∞
		accuracy	0.8273	0.8278	0.8168	0.0021
		val. loss	0.008	0.0083	0.0084	∞
		val. acc.	0.7464	0.7369	0.7309	0.0025
24	100k	loss	0.0054	0.0055	0.0059	∞
		accuracy	0.8256	0.8212	0.8072	0.0022
		val. loss	0.0082	0.0084	0.0086	∞
		val. acc.	0.7412	0.7335	0.7194	0.0027
	250k	loss	0.0054	0.0056	0.0058	∞
		accuracy	0.826	0.8164	0.811	0.0022
		val. loss	0.0081	0.0083	0.0086	∞
		val. acc.	0.7451	0.7375	0.7281	0.0026

Table 4.5: Simple RNN model: 3rd test training data

Simple RNN model - SMILES validation data						
2 layers; 0.001 learning rate; 0.3 dropout						
epochs	samples		float	int16	int8	int4
16	100k	valid (%)	44	30.8	21.4	1.2
		unique (%)	43.2	30.6	18.8	1.1
	250k	valid (%)	47.5	33.1	29.3	0.7
		unique (%)	45.7	32.8	29	0.6
24	100k	valid (%)	45.5	36.2	31.9	1.2
		unique (%)	44.6	35.2	30.2	1
	250k	valid (%)	53.6	40.8	41.9	0.9
		unique (%)	51.6	40	39.8	0.7

Table 4.6: Simple RNN model: 3rd test SMILES validation data

Tables 4.5 and 4.6 share some of the conclusions from tests 1 and 2. The relation between the accuracy and loss during training and valid created SMILES during inference is still not linear but is more closely related than in the last two tests when comparing float and quantized models.

The performance degradation going from float to a quantized 16 bit model leads to validation metrics around 13% worse across most combinations of parameters, when training using 100k samples for 24 epochs it declined less, at around 9%. Going from 16 to 8 bit the number of valid SMILES decreases less when using more epochs and a higher number of samples, being very similar when 24 epochs with 250k samples are the parameters used, with 8 bits generating around 1% more valid SMILES but having repeated results, having 0.2% less of them being unique. Training with 4 bits was once again not able to generate any significant number of valid SMILES, making it a poor choice for quantization. During this test training was actually unable to progress past the first epochs for the 4 bit model, leading to infinite loss and training accuracy very close to 0.

4. Experimental Results

4.1.4 Simple RNN - Final Analysis

Figures 4.1, 4.2 and 4.3 display the validation data from tables 4.2, 4.4 and 4.6 respectively, offering a better representation of performance differences between models.

Simple RNN model - test 1

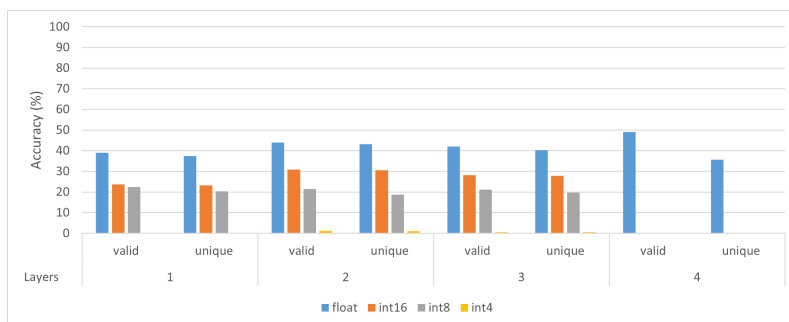


Figure 4.1: Simple RNN - Valid and Unique SMILES by number of layers

Simple RNN model - test 2

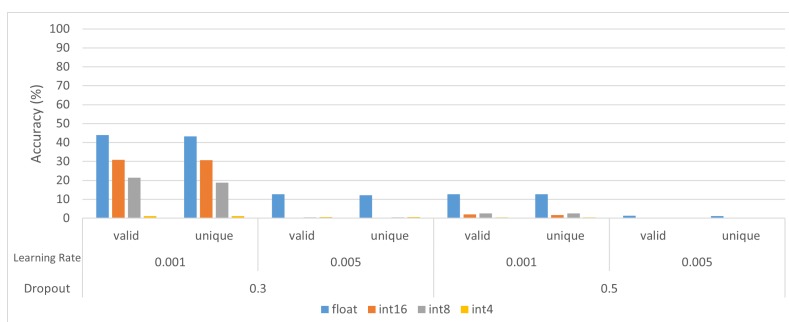


Figure 4.2: Simple RNN - Valid and Unique SMILES by dropout and learning rate

Simple RNN model - test 3

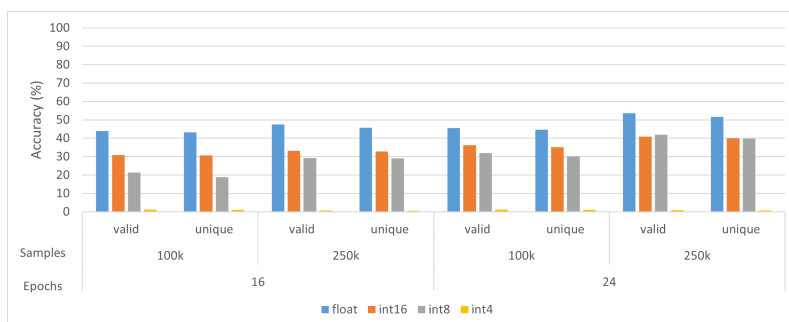


Figure 4.3: Simple RNN - Valid and Unique SMILES by number of samples and epochs

Comparing the performance variation across the tests performed it is noted that using 4 bits for the Simple RNN model leads to virtually no training taking place. The 4 bit model was unable to generate more than around 1% of valid SMILES independent of the combinations of parameters being tested. Using 16 and 8 bits lead to similar quantization performance when the best values for the parameters are used. The degradation is around 13%, which is significant, making their use also not advised.

Dropout and Learning Rate seem to have the most effect on performance. Simple RNN layers are very susceptible to changes in these parameters. The number of Recurrent Layers used affect quantized models more substantially than the float version. Once the number of layers reaches 4 the quantized models run into problems that the simplicity of the Simple RNN algorithm cannot address. The final conclusion from these tests is that the quantization method used is not appropriate for models using Simple RNN recurrent layers as the performance degrades significantly. On broader terms, using this type of layer for SMILES generation is also not advised as even in the best case, when using float and running for 24 epochs with 250k samples, only 53.5% of valid samples are generated, underperforming GRU and LSTM by a fair margin.

4. Experimental Results

4.2 GRU model

As in section 4.1 for the Simple RNN model, the initial parameters for the Gated Recurrent Unit (GRU) models were chosen based on the initial values of the parameters from the original LSTM model used in the SMILES generation program. GRU's model expected performance is much closer to the LSTM model, as their internal mechanisms are quite similar, as described in section 2. This in turn makes studying quantization for this model more propitious than for the Simple RNN models.

4.2.1 GRU test 1 - Number of layers

The first test kept the number of samples at 100k, running for 16 epochs with 0.001 as learning rate and a dropout of 0.3 per epoch. The test focused on the effects the number of layers had on quantization performance and produced the results displayed in table 4.7 and 4.8. Table 4.7 displays the final training parameters. Table 4.8 displays the percentage of valid and unique SMILES.

GRU model - training data					
100k samples; 16 epochs; 0.001 learning rate; 0.3 dropout					
number of layers		float	int16	int8	int4
1	loss	0.0052	0.0044	0.0044	0.0044
	accuracy	0.8316	0.8618	0.8637	0.8603
	val. loss	0.0079	0.0076	0.0075	0.0079
	val. acc.	0.7521	0.7627	0.7659	0.7524
2	loss	0.0051	0.0039	0.0039	0.0042
	accuracy	0.8351	0.8786	0.8785	0.8687
	val. loss	0.0076	0.0068	0.0069	0.0073
	val. acc.	0.761	0.7884	0.7862	0.7727
3	loss	0.0051	0.0038	0.0038	0.0041
	accuracy	0.8347	0.8811	0.8809	0.8708
	val. loss	0.0075	0.0068	0.0067	0.0074
	val. acc.	0.7629	0.7894	0.7923	0.7701
4	loss	0.0052	0.0038	0.0038	0.0041
	accuracy	0.832	0.8814	0.8819	0.8707
	val. loss	0.0074	0.0068	0.0067	0.0074
	val. acc.	0.767	0.7904	0.7917	0.7751

Table 4.7: GRU model: 1st test training data

GRU model - SMILES training data					
100k samples; 16 epochs; 0.001 learning rate; 0.3 dropout					
number of layers		float	int16	int8	int4
1	valid (%)	88.5	84.2	80.6	79.2
	unique (%)	84.8	80.7	78.5	76.4
2	valid (%)	92.9	93.2	85.1	88.6
	unique (%)	90.7	88.8	84.1	81.3
3	valid (%)	87.7	91.8	90	87.7
	unique (%)	86.2	87.1	89.5	69.4
4	valid (%)	84.8	92.2	90.8	84.4
	unique (%)	83.8	87.4	80.4	82.5

Table 4.8: GRU model: 1st test SMILES validation data

From analyzing tables 4.7 and 4.8 the following is evinced. The final accuracy and loss metrics taken during training and percentage of valid SMILES created during inference is even less comparable between float and quantized models than in section 4.1. High accuracy leads to a high percentage of valid SMILES but some combination of parameters have better training data yet lead to lower percentage of valid SMILES as is the case of 16 bit and 8 bit models using 1 layer. Therefore looking at this metric is only useful as a general indication of a successful training and further analysis will focus on validation data.

The best results were achieved using 2 recurrent layers. In these models the quantization was quite successful, achieving 88.6% of valid SMILES even in the 4 bit version. The 16 bit version even achieved 0.3% more valid SMILES than the float yet 1.9% less unique ones, which due to the small differences can be explained by the randomness of the process alone. Using 3 and 4 layers some quantized models actually achieved better results than float, while still inferior to those achieved by the 2 layer version. In some of those models where more valid SMILES are generated a high percentage of repeated ones are also present, giving signs of overfitting occurring in some instances, very evident in the 4 bits, 3 layers model. Using 1 layer means the models will run faster but incurs

4. Experimental Results

less valid SMILES generated, 4.4% less than using 2 layers for the float model, similar degradation for the 8 bit model and around 9% for the 16 bit and 4 bit models.

Overall this test shows promising results for reduced precision training for instances where some accuracy can be sacrificed. As the SMILES molecules can be validated externally some accuracy degradation is acceptable when randomly generating new ones is the goal as in this instance.

4.2.2 GRU test 2 - Dropout and Learning Rate

The second test used 2 recurrent layers, the best performing of the first test, while also keeping the number of samples at 100k, running for 16 epochs. During this stage the effect of quantization on performance was tested for a setup with higher dropouts and learning rate. Table 4.9 displays the training performance. Table 4.10 displays the percentage of valid and unique SMILES created.

GRU model - training data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.3	0.001	loss	0.0051	0.0039	0.0039	0.0042
		accuracy	0.8351	0.8786	0.8785	0.8687
		val. loss	0.0076	0.0068	0.0069	0.0073
		val. acc.	0.761	0.7884	0.7862	0.7727
	0.005	loss	0.005	0.0044	0.0102	0.0107
		accuracy	0.8407	0.8601	0.6479	0.6363
		val. loss	0.0072	0.0071	0.0168	0.0179
		val. acc.	0.7738	0.7799	0.433	0.4086
0.5	0.001	loss	0.0063	0.0119	0.0044	0.0048
		accuracy	0.7911	0.6381	0.8604	0.8479
		val. loss	0.0084	0.0198	0.0071	0.0078
		val. acc.	0.7343	0.4086	0.7772	0.7562
	0.005	loss	0.0062	0.0103	∞	0.0123
		accuracy	0.7935	0.6224	0.0022	0.6027
		val. loss	0.0081	0.0213	∞	0.0235
		val. acc.	0.7472	0.2147	0.0026	0.1099

Table 4.9: GRU model: 2nd test training data

GRU model - SMILES validation data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.3	0.001	valid (%)	92.9	93.2	85.1	88.6
		unique (%)	90.7	88.8	84.1	81.3
	0.005	valid (%)	89.4	79.2	86.2	82.1
		unique (%)	86.5	75.4	67.7	73.2
0.5	0.001	valid (%)	85.6	89.1	76.7	85.5
		unique (%)	85.3	85.4	73.9	71.2
	0.005	valid (%)	77	77.4	64	63.7
		unique (%)	77	63.7	53.8	56.3

Table 4.10: GRU model: 2nd test SMILES validation data

Comparing the percentage of SMILES generated between float and quantized models it's observed that, when using 0.3 dropout, increasing the learning rate to 0.005 incurs a 4.2% loss in validation accuracy for the float model, when comparing unique SMILES. The quantized models degrade even further with 16 bits performing around 13% worse, 8 bit degrading the most at over 16% and 4 bits degrading around 8%. Increasing dropout to 0.5 leads to less overall decline in performance for the quantized models than going from 0.001 to 0.005 learning rate, at 3.4% and 10.2% for 16 bit and 8 bit respectively. The 4 bits model degrades a bit more at 11.2%. The float model declines more significantly than when increasing learning rate. Increasing dropout along with learning rate leads to more unsatisfactory results and more unstable training. It is expected since increasing dropout slows down convergence, and when used together with higher learning rate training is hindered leading to performance between 13% and 30% worse as observed.

From this test it is clear that increasing dropout and learning rate does not improve the results obtained. The performance degradation is not as severe as during the Simple RNN second test but using 0.3 dropout per epoch and a learning rate of 0.001 leads to superior results across float and quantized models. The quantized models degradation is higher than the float when learning rate is increased. A higher learning rate, that could translate into faster training, is more negatively impactful when lower precision is used. Higher learning rate also seems to cause a higher percentage of repeated SMILES to be

4. Experimental Results

generated.

4.2.3 GRU test 3 - Epochs and Number of training samples

The final test kept using 2 recurrent layers, with a learning rate of 0.001 and dropout of 0.3, being the best performing parameters from tests 1 and 2. The tested parameters were then the number of epochs and training samples, increasing them in both cases. Table 4.11 displays the final training parameters. Table 4.12 displays the percentage of valid and unique SMILES.

GRU model - training data						
2 layers; 0.001 learning rate; 0.3 dropout						
epochs	samples		float	int16	int8	int4
16	100k	loss	0.0051	0.0039	0.0039	0.0042
		accuracy	0.8351	0.8786	0.8785	0.8687
		val. loss	0.0076	0.0068	0.0069	0.0073
		val. acc.	0.761	0.7884	0.7862	0.7727
	250k	loss	0.0049	0.0037	0.0037	0.0041
		accuracy	0.8429	0.8844	0.8848	0.8718
		val. loss	0.0072	0.0065	0.0065	0.0075
		val. acc.	0.7764	0.7986	0.7982	0.7704
24	100k	loss	0.005	0.0038	0.0038	0.0041
		accuracy	0.839	0.8826	0.8827	0.8707
		val. loss	0.0074	0.0067	0.0067	0.0075
		val. acc.	0.7676	0.7927	0.7934	0.77
	250k	loss	0.0048	0.0037	0.0037	0.0041
		accuracy	0.8458	0.886	0.8856	0.8723
		val. loss	0.007	0.0065	0.0065	0.0073
		val. acc.	0.783	0.7994	0.7996	0.7776

Table 4.11: GRU model: 3rd test training data

GRU model - SMILES validation data						
2 layers; 0.001 learning rate; 0.3 dropout						
epochs	samples		float	int16	int8	int4
16	100k	valid (%)	92.9	93.2	85.1	88.6
		unique (%)	90.7	88.8	84.1	81.3
	250k	valid (%)	94.6	93.6	92	89.9
		unique (%)	93.4	89.4	91.6	58.9
24	100k	valid (%)	94.9	94.4	91.2	82.6
		unique (%)	93.5	94.1	90.8	80.7
	250k	valid (%)	95.7	95	95	84.4
		unique (%)	93.9	93.7	93.5	71.8

Table 4.12: GRU model: 3rd test SMILES validation data

The GRU float model was able to achieve 95.7% valid and 93.9% unique SMILES generated. Increasing the number of samples had a similar effect as training for more epochs, as the performance from the 16 epochs, 250k samples test was very similar to the 24 epochs, 250k samples one. The model was able to properly scale with further training.

Training with 16 bits precision and running for 24 epochs leads to validation data on par with the float version, leading to slightly more unique smiles in the 100k samples test (0.6% improvement) and only 0.2% less when using 250k training samples, the best performing test. Going from int 16 to int 8 the performance deteriorates to a greater degree across most configurations, yet in the 24 epochs 250k samples test it achieves the same number of valid SMILES as int 16 and only 0.4% less unique SMILES when compared to float. This means the GRU model scales well even at a precision of 8 bits. Int 4 is unable to properly scale. When using 250k samples it seems to overfit leading to a large percentage of repeated smiles, particularly when running for 16 epochs where over 30% of valid SMILES created are not unique.

4. Experimental Results

4.2.4 GRU - Final Analysis

Figures 4.4, 4.5 and 4.6 display the validation data from tables 4.8, 4.10 and 4.12 respectively, offering a better representation of performance differences between models.

GRU model - test 1

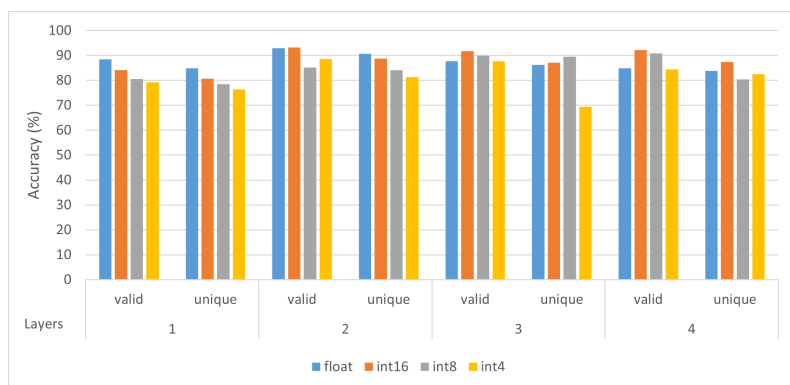


Figure 4.4: GRU - Valid and Unique SMILES by number of layers

GRU model - test 2

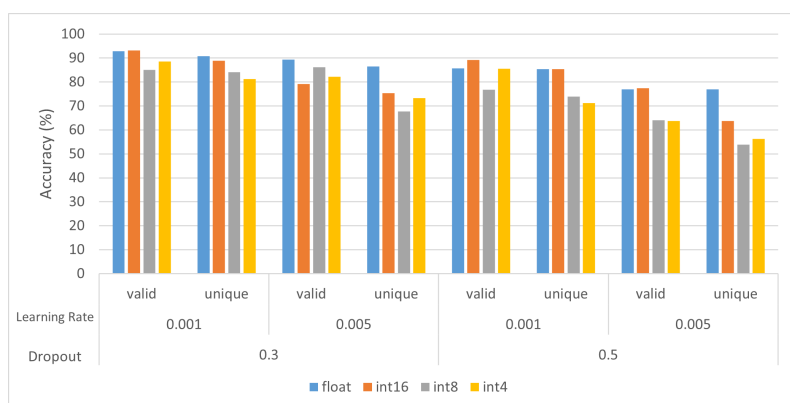


Figure 4.5: GRU - Valid and Unique SMILES by dropout and learning rate

GRU model - test 3

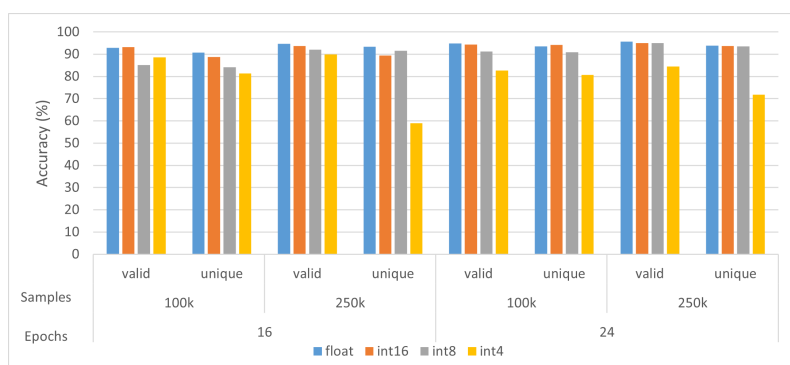


Figure 4.6: GRU - Valid and Unique SMILES by number of samples and epochs

Comparing the validation data across the three tests it is observed that quantization for the GRU models was successful. The GRU based models achieved good overall performance. 95.7% valid and 93.9% unique SMILES in the float model with 16 bit and 8 bit models being within 1% of those values in the best performing scenario. Using int 4 is not advised as the model degrades considerably using the configuration that achieve the best results.

Varying the number of GRU layers used lead to some interesting results. The float and 16 bit models performed the best using 2 layers. On the other hand the 8 bit model improved when 3 layers were used and the 4 bit model performed the best for a 4 layers model, although the improvement was marginal when compared to the 2 layers, 4 bit version. Increasing dropout and learning rate had a negative impact on all the models. Learning rate seems to have a bigger effect on performance as increasing it degraded the results significantly. The float, 16 bit and 8 bit models were able to scale in a satisfactory manner achieving great quantization performance when the number of samples and epochs was increased. The precision of the GRU can be reduced to 8 bits without sacrificing much in terms of generating a high percentage of valid and unique SMILES.

4.3 LSTM model

Starting from the same parameters as the GRU and Simple RNN models, a more comprehensive set of tests was developed for the Long Short-Term Memory (LSTM) model. Testing was divided into four stages instead of three as per the previous models, by also examining the effects of using different dropout for the first and second recurrent layers of the model. It also included a higher range of values being tested for each parameter other than the number of layers.

4. Experimental Results

4.3.1 LSTM test 1 - Number of layers

The first test kept the number of samples at 100k, running for 16 epochs with a learning rate of 0.001 and dropout of 0.3. The test focused on the effect the number of layers had on the quantization performance of the LSTM model and produced the results displayed in table 4.13 and 4.14. Table 4.13 displays the final training parameters. Table 4.14 displays the percentage of valid and unique SMILES from the validation stage.

LSTM model - training data					
100k samples; 16 epochs; 0.001 learning rate; 0.3 dropout					
number of layers		float	int16	int8	int4
1	loss	0.0054	0.0054	0.0043	0.0055
	accuracy	0.827	0.8281	0.8644	0.8171
	val. loss	0.0082	0.0081	0.0078	0.0082
	val. acc.	0.7404	0.7472	0.7557	0.7253
2	loss	0.0052	0.004	0.0039	0.0053
	accuracy	0.8334	0.8733	0.8774	0.847
	val. loss	0.0078	0.0072	0.007	0.0078
	val. acc.	0.7559	0.7771	0.7821	0.7463
3	loss	0.0051	0.005	0.005	0.0051
	accuracy	0.836	0.8643	0.8487	0.8156
	val. loss	0.0073	0.0071	0.0072	0.0074
	val. acc.	0.7688	0.7846	0.7972	0.7516
4	loss	0.0051	0.0049	0.0051	0.0052
	accuracy	0.8346	0.8385	0.8431	0.8119
	val. loss	0.0074	0.0071	0.0074	0.0075
	val. acc.	0.7661	0.7697	0.7739	0.7452

Table 4.13: LSTM model: 1st test training data

LSTM model - SMILES validation data					
100k samples; 16 epochs; 0.001 learning rate; 0.3 dropout					
number of layers		float	int16	int8	int4
1	valid (%)	75.2	83.1	82.1	80.6
	unique (%)	72.1	75.3	76.9	74.8
2	valid (%)	89.6	89.1	87.7	86.9
	unique (%)	88.9	82.2	83.3	85.1
3	valid (%)	85.9	78	83.2	80.9
	unique (%)	82.3	73.2	76.8	69.3
4	valid (%)	88.1	86.8	87.1	87.7
	unique (%)	81.8	80.4	80.2	75.6

Table 4.14: LSTM model: 1st test SMILES validation data

Observing the results displayed on table 4.14 the quantization performance is satisfying, even outperforming float in the 1 layer configuration, albeit more than 10% worse when compared to the float model using 2 layers. The 2 and 4 layers configurations reach the highest percentage of valid SMILES, with 4 layers having the undesirable behavior of generating more repeated SMILES across both float and quantized models. The 3 layers quantized models performed the worse, being comparable to the 1 layer models. When 2 layers are used the 16 bit model generated only 0.5% less valid SMILES yet around 7% of those are not unique. The 8 bit and 4 bit models have around 2% less valid SMILES when compared to the 16 bit model, but a higher percentage of unique SMILES, this was likely due to the inherent randomness of the training process resulting in less generalization capabilities in this run and further testing would be necessary to validate this result. From the first test it is concluded that 2 layers is the best performing configuration.

4.3.2 LSTM test 2 - Dropout and Learning Rate

The second test used 2 recurrent layers, the best performing of the first test, and kept the number of samples at 100k, running for 16 epochs. During this stage dropout was set between 0.1 and 0.7 per epoch in 0.2 increments and learning rate was tested for several

4. Experimental Results

values between 0.001 and 0.01. Tables 4.15 and 4.16 display the training performance. Tables 4.17 and 4.18 display the percentage of valid and unique SMILES created.

LSTM model - training data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.1	0.001	loss	0.0042	0.0038	0.0038	0.0046
		accuracy	0.8687	0.8819	0.8637	0.8387
		val. loss	0.0071	0.0074	0.0073	0.0078
		val. acc.	0.7804	0.7695	0.7536	0.7535
	0.003	loss	0.0039	0.0041	0.0039	0.0045
		accuracy	0.8796	0.8513	0.8641	0.856
		val. loss	0.0067	0.0071	0.0067	0.0077
		val. acc.	0.7945	0.7689	0.7805	0.7732
	0.005	loss	0.0037	0.0038	0.0038	0.004
		accuracy	0.8833	0.8896	0.8815	0.8816
		val. loss	0.0064	0.0071	0.007	0.0069
		val. acc.	0.8027	0.7913	0.7841	0.8012
	0.01	loss	0.0037	0.0038	0.0038	0.0042
		accuracy	0.8841	0.883	0.8684	0.8374
		val. loss	0.0063	0.0065	0.0065	0.0072
		val. acc.	0.8056	0.8046	0.7913	0.7631
0.3	0.001	loss	0.0052	0.0052	0.0051	0.0055
		accuracy	0.8334	0.8326	0.8355	0.7987
		val. loss	0.0078	0.0078	0.0077	0.0083
		val. acc.	0.7559	0.7552	0.7578	0.7244
	0.003	loss	0.0049	0.0049	0.0049	0.0052
		accuracy	0.8443	0.8529	0.8514	0.835
		val. loss	0.0071	0.007	0.0072	0.0075
		val. acc.	0.7775	0.7854	0.784	0.769
	0.005	loss	0.0048	0.0047	0.0048	0.005
		accuracy	0.8459	0.8462	0.8416	0.8456
		val. loss	0.0071	0.007	0.0071	0.0074
		val. acc.	0.7779	0.7782	0.7739	0.7776
	0.01	loss	0.0048	0.0049	0.0048	0.005
		accuracy	0.8477	0.8596	0.8515	0.8227
		val. loss	0.007	0.0071	0.007	0.0073
		val. acc.	0.7826	0.7936	0.7861	0.7595

Table 4.15: LSTM model: 2nd test training data - part 1

LSTM model - training data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.5	0.001	loss	0.0064	0.0058	0.0066	0.0067
		accuracy	0.788	0.7906	0.7946	0.7814
		val. loss	0.0088	0.008	0.009	0.0092
		val. acc.	0.7223	0.7247	0.7283	0.7163
	0.003	loss	0.0061	0.0048	0.0042	0.0065
		accuracy	0.7981	0.8189	0.8683	0.7701
		val. loss	0.0084	0.0066	0.007	0.009
		val. acc.	0.7352	0.7543	0.7809	0.7094
	0.005	loss	0.0061	0.0049	0.0043	0.0058
		accuracy	0.7992	0.8201	0.8539	0.82
		val. loss	0.0086	0.007	0.0071	0.0082
		val. acc.	0.7299	0.749	0.768	0.7489
	0.01	loss	0.006	0.0058	0.0061	0.0059
		accuracy	0.8	0.79	0.7854	0.8281
		val. loss	0.0087	0.0084	0.0088	0.0085
		val. acc.	0.7276	0.7185	0.7143	0.7531
0.7	0.001	loss	0.0077	0.0061	0.006	0.0071
		accuracy	0.7336	0.8096	0.7979	0.761
		val. loss	0.0111	0.0089	0.0088	0.0103
		val. acc.	0.6403	0.7384	0.7277	0.6642
	0.003	loss	0.0076	0.0054	0.0052	0.0071
		accuracy	0.7384	0.8433	0.8338	0.7415
		val. loss	0.0113	0.0079	0.0076	0.0102
		val. acc.	0.6325	0.7691	0.7604	0.6472
	0.005	loss	0.0076	0.0075	0.0075	0.0078
		accuracy	0.739	0.7492	0.7252	0.7634
		val. loss	0.0121	0.0111	0.0111	0.0115
		val. acc.	0.6184	0.6833	0.6614	0.6539
	0.01	loss	0.0076	0.0068	0.0075	0.0062
		accuracy	0.7386	0.7502	0.7397	0.8001
		val. loss	0.0121	0.0109	0.0109	0.0092
		val. acc.	0.6095	0.6278	0.6746	0.7195

Table 4.16: LSTM model: 2nd test training data - part 2

4. Experimental Results

LSTM model - SMILES validation data							
2 layers; 16 epochs; 100k samples							
dropout	learning rate		float	int16	int8	int4	
0.1	0.001	valid (%)	88.8	88.7	88.7	87.2	
		unique (%)	83.7	83.8	83.2	82.9	
	0.003	valid (%)	89.5	90.2	88.1	89.1	
		unique (%)	85.2	84.2	83.6	81.2	
	0.005	valid (%)	93.2	92.8	91.9	90.3	
		unique (%)	91.6	91.4	89.5	86.5	
	0.01	valid (%)	90.8	89.5	90	88	
		unique (%)	89.9	88.1	89.7	87.2	
	0.3	0.001	valid (%)	89.6	89.1	87.7	86.9
			unique (%)	88.9	82.2	83.3	85.1
0.003		valid (%)	89.2	85.7	87.8	82.9	
		unique (%)	86.3	83.7	83.9	78.4	
0.005		valid (%)	90.8	89.1	90.2	89.1	
		unique (%)	90.5	87.2	86.9	87.6	
0.01		valid (%)	80.4	83.1	80.7	77.5	
		unique (%)	80.1	78.2	76.9	77.3	

Table 4.17: LSTM model: 2nd test SMILES validation data - part 1

LSTM model - SMILES validation data							
2 layers; 16 epochs; 100k samples							
dropout	learning rate		float	int16	int8	int4	
0.5	0.001	valid (%)	59.4	60.3	58.8	57.8	
		unique (%)	59	58	57.4	54.8	
	0.003	valid (%)	71.4	72.7	71.4	69.3	
		unique (%)	70.2	70.2	69.8	67.8	
	0.005	valid (%)	63.5	64.1	62.9	64.2	
		unique (%)	63.5	61.5	61.7	61.6	
	0.01	valid (%)	63.9	64.2	64.9	62	
		unique (%)	63.9	63.5	63.6	62.6	
	0.7	0.001	valid (%)	11.7	21.6	16.5	20.4
			unique (%)	11.7	17.9	15	18.1
0.003		valid (%)	13.6	21.5	21.5	21.7	
		unique (%)	13.6	17.7	17.5	17.2	
0.005		valid (%)	11.9	16.7	17.9	15.7	
		unique (%)	11.9	15.2	12.5	14.8	
0.01		valid (%)	1.1	17.6	16.9	25.5	
		unique (%)	1.1	12.4	12.6	19.4	

Table 4.18: LSTM model: 2nd test SMILES validation data - part 2

4. Experimental Results

From the second test it is observed that lower dropouts perform better, with 0.1 and 0.3 dropout performing significantly better than 0.5 and 0.7, independent of learning rate. Using 0.7 as dropout, which is very high, as it means 70% of weights trained will be randomized after each epoch, quantized models achieve higher percentage of valid and unique SMILES compared to float. Compared to models using smaller dropouts they achieve very low accuracy and are therefore only interesting in their comparative behavior since more than 60% degradation is present, relative to models using optimal parameters.

Regarding learning rate the middle values perform better than those in the upper and lower limits considered. For smaller dropouts 0.05 seems to be the optimal value, for the higher ones 0.03 performs slightly better. Some exceptions occur as the best performing 8 bit model when it comes to unique SMILES occurs with 0.01 learning rate, but the difference of 0.2% is not significant enough to consider it an improvement since a 1.9% drop in valid SMILES generated occurs at the same time. A similar situation happens with the 4 bit model using 0.3 dropout and 0.005 learning rate where more unique SMILES are generated but less valid.

The best combination of parameters, able to achieve 93.2% valid and 91.6% unique, is then 0.1 for dropout and 0.005 for learning rate, having satisfactory quantization performance, with the 16 bit model generating only 0.2% less unique SMILES, 8 bit model withing 2% and 4 bit model withing 5%.

4.3.3 LSTM test 3 - Dropout per layer and Learning Rate

The third test essayed centered around having the dropout vary between LSTM layers. Testing for combinations of the values used during the second test. Some compromises had to be made during this series of tests to limit the number of times the program had to run. Learning rate of 0.01 was removed from this test as it performed worse across the whole range of dropouts. The quantized models were tested only for the best performing learning rate overall, 0.05, for each combination of dropouts as they take much longer to run due to the constant data conversions required to emulate training at reduced precision.

Tables 4.19 and 4.20 display the training performance. Tables 4.21, 4.22 and 4.23 display the percentage of valid and unique SMILES created.

LSTM model - training data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.1; 0.3	0.001	loss	0.0042	-	-	-
		accuracy	0.8666	-	-	-
		val. loss	0.0072	-	-	-
		val. acc.	0.7769	-	-	-
	0.003	loss	0.0039	-	-	-
		accuracy	0.8773	-	-	-
		val. loss	0.0067	-	-	-
		val. acc.	0.7939	-	-	-
	0.005	loss	0.0041	0.0042	0.0041	0.0041
		accuracy	0.8716	0.8637	0.861	0.8652
		val. loss	0.0067	0.0073	0.007	0.007
		val. acc.	0.7917	0.7816	0.7791	0.7829
0.3; 0.1	0.001	loss	0.0052	-	-	-
		accuracy	0.8334	-	-	-
		val. loss	0.0078	-	-	-
		val. acc.	0.7539	-	-	-
	0.003	loss	0.0048	-	-	-
		accuracy	0.8448	-	-	-
		val. loss	0.0071	-	-	-
		val. acc.	0.7797	-	-	-
	0.005	loss	0.0048	0.005	0.0049	0.0048
		accuracy	0.8463	0.8413	0.8417	0.8376
		val. loss	0.0071	0.0074	0.0072	0.0071
		val. acc.	0.7786	0.7764	0.7768	0.7731
0.3; 0.5	0.001	loss	0.0052	-	-	-
		accuracy	0.8309	-	-	-
		val. loss	0.0079	-	-	-
		val. acc.	0.7531	-	-	-
	0.003	loss	0.0049	-	-	-
		accuracy	0.8425	-	-	-
		val. loss	0.0072	-	-	-
		val. acc.	0.7744	-	-	-
	0.005	loss	0.0048	0.0047	0.0047	0.0048
		accuracy	0.8445	0.852	0.8563	0.8299
		val. loss	0.0071	0.0069	0.007	0.007
		val. acc.	0.7806	0.7831	0.7871	0.7628

Table 4.19: LSTM model: 3rd test training data - part 1

4. Experimental Results

LSTM model - training data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.5; 0.3	0.001	loss	0.0064	-	-	-
		accuracy	0.7886	-	-	-
		val. loss	0.0093	-	-	-
		val. acc.	0.7037	-	-	-
	0.003	loss	0.0061	-	-	-
		accuracy	0.7986	-	-	-
		val. loss	0.0086	-	-	-
		val. acc.	0.7267	-	-	-
	0.005	loss	0.006	0.0056	0.0057	0.0059
		accuracy	0.8002	0.8188	0.8198	0.8087
		val. loss	0.0086	0.0078	0.008	0.0083
		val. acc.	0.7298	0.7451	0.746	0.7359
0.1; 0.5	0.001	loss	0.0043	-	-	-
		accuracy	0.8648	-	-	-
		val. loss	0.0072	-	-	-
		val. acc.	0.7745	-	-	-
	0.003	loss	0.0039	-	-	-
		accuracy	0.877	-	-	-
		val. loss	0.0067	-	-	-
		val. acc.	0.7934	-	-	-
	0.005	loss	0.0039	0.004	0.004	0.004
		accuracy	0.8786	0.8677	0.8614	0.8522
		val. loss	0.0065	0.0069	0.0069	0.0069
		val. acc.	0.8001	0.785	0.7793	0.7709
0.1; 0.7	0.001	loss	0.0044	-	-	-
		accuracy	0.8626	-	-	-
		val. loss	0.0073	-	-	-
		val. acc.	0.773	-	-	-
	0.003	loss	0.004	-	-	-
		accuracy	0.875	-	-	-
		val. loss	0.0067	-	-	-
		val. acc.	0.792	-	-	-
	0.005	loss	0.0039	0.0042	0.0041	0.0044
		accuracy	0.8774	0.8635	0.8645	0.8502
		val. loss	0.0066	0.007	0.0069	0.0074
		val. acc.	0.796	0.7816	0.7825	0.7695

Table 4.20: LSTM model: 3rd test training data - part 2

LSTM model - SMILES validation data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.1; 0.3	0.001	valid (%)	87.2	-	-	-
		unique (%)	87.2	-	-	-
	0.003	valid (%)	92.3	-	-	-
		unique (%)	91.2	-	-	-
	0.005	valid (%)	86.5	87.4	86.3	86.2
		unique (%)	85.7	84.9	83.1	84.2
0.3; 0.1	0.001	valid (%)	87.2	-	-	-
		unique (%)	86.5	-	-	-
	0.003	valid (%)	91.2	-	-	-
		unique (%)	91.1	-	-	-
	0.005	valid (%)	91.8	92.3	90.6	90.1
		unique (%)	91	88.8	90.4	88.4

Table 4.21: LSTM model: 3rd test SMILES validation data - part 1

4. Experimental Results

LSTM model - SMILES validation data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.3; 0.5	0.001	valid (%)	70.5	-	-	-
		unique (%)	67	-	-	-
	0.003	valid (%)	84.1	-	-	-
		unique (%)	81	-	-	-
	0.005	valid (%)	87.4	88.5	87.8	87.3
		unique (%)	87.1	87.7	86.7	85.8
0.5; 0.3	0.001	valid (%)	60.8	-	-	-
		unique (%)	60.8	-	-	-
	0.003	valid (%)	66.9	-	-	-
		unique (%)	66.9	-	-	-
	0.005	valid (%)	70.1	71.8	73.7	69.9
		unique (%)	70.1	71	71.2	69.5

Table 4.22: LSTM model: 3rd test SMILES validation data - part 2

LSTM model - SMILES validation data						
2 layers; 16 epochs; 100k samples						
dropout	learning rate		float	int16	int8	int4
0.1; 0.5	0.001	valid (%)	86.3	-	-	-
		unique (%)	84.8	-	-	-
	0.003	valid (%)	93.6	-	-	-
		unique (%)	92	-	-	-
	0.005	valid (%)	93.7	93.2	92.7	92.6
		unique (%)	92.1	91.4	92	91.8
0.1; 0.7	0.001	valid (%)	77.3	-	-	-
		unique (%)	76.4	-	-	-
	0.003	valid (%)	92.5	-	-	-
		unique (%)	86.5	-	-	-
	0.005	valid (%)	92.8	92.6	91.3	91.4
		unique (%)	89.6	90.8	90.6	89.4

Table 4.23: LSTM model: 3rd test SMILES validation data - part 3

From this test it was established that using a smaller dropout for the first layer and increasing it in the second achieves better results than using a higher value for the first or the same for both. Using 0.1 dropout for the first layer and 0.3 for the second seemed to contradict the previous statement as the reverse configuration of 0.3 dropout in the first layer and 0.1 in the second outperformed the former across both float and every quantized model. Float had 5.3% more unique SMILES generated, 16 bit 3.9% improvement, 8 bit had 7.3% and 4 bit 4.2%. Further testing proved that the same was not verified when 0.3 and 0.5 dropout were used, which worse results across every model.

4. Experimental Results

Using 0.1 for first layer and 0.5 for second layer lead to great results with very few repeated smiles created and high accuracy even with a 4 bit quantized model. This had better performance than the 0.3/0.1 test, with special significance in the quantized model, which was able to achieve results withing 1% of the float model even at 4 bits precision. The 4 bits model had the greatest improvements, improving over 5% when compared to simply using 0.1 as dropout for both layers.

4.3.4 LSTM test 4 - Epochs and Number of training samples

The final test kept using 2 recurrent layers, with a learning rate of 0.005 and dropout of 0.1 in the first layer and 0.5 in the second, being the best performing parameters from tests 1 to 3. The tested parameters were then the number of epochs and training samples, increasing them in both cases. This test included more quantized models, adding 6, 3 and 2 bit configurations for the final test. Table 4.24 displays the final training parameters. Table 4.25 displays the percentage of valid and unique SMILES.

LSTM model - training data									
2 layers; 0.005 learning rate; 0.1, 0.5 dropout									
epochs	samples		float	int16	int8	int6	int4	int3	int2
16	100k	loss	0.0039	0.004	0.004	0.004	0.004	0.007	0.0209
		accuracy	0.877	0.8677	0.8614	0.8559	0.8522	0.7363	0.342
		val. loss	0.0067	0.0069	0.0069	0.0067	0.0069	0.01	0.0218
		val. acc.	0.7934	0.785	0.7793	0.7839	0.7709	0.6427	0.1464
	250k	loss	0.0037	0.0038	0.0038	0.0039	0.0039	0.0071	0.0238
		accuracy	0.8831	0.8758	0.8795	0.8648	0.8708	0.7558	0.2101
		val. loss	0.0062	0.0064	0.0064	0.0066	0.0065	0.0102	0.0344
		val. acc.	0.8088	0.8021	0.8055	0.7921	0.7976	0.6597	0.1833
24	100k	loss	0.0038	0.0035	0.0036	0.0036	0.0036	0.008	0.0239
		accuracy	0.8806	0.8994	0.8855	0.8916	0.888	0.7211	0.2095
		val. loss	0.0064	0.0065	0.0067	0.0067	0.0067	0.012	0.0357
		val. acc.	0.8019	0.8047	0.7923	0.7977	0.7945	0.6146	0.1786
	250k	loss	0.0037	0.0036	0.0039	0.0036	0.0036	0.0078	0.0262
		accuracy	0.8851	0.8883	0.8787	0.8895	0.8871	0.7338	0.1551
		val. loss	0.0062	0.0066	0.0067	0.0067	0.0067	0.0117	0.0391
		val. acc.	0.8099	0.7954	0.7916	0.7958	0.7937	0.6254	0.1322

Table 4.24: LSTM model: 4th test training data

LSTM model - SMILES validation data									
2 layers; 0.005 learning rate; 0.1, 0.5 dropout									
epochs	samples		float	int16	int8	int6	int4	int3	int2
16	100k	valid (%)	93.7	93.2	92.7	92	92.6	60.9	0.4
		unique (%)	92.1	91.4	92	90.5	91.8	58.1	0.4
	250k	valid (%)	91.6	91.3	92.2	93	92.1	63	1.2
		unique (%)	91.3	90	90.7	89.4	91	61.8	1.1
24	100k	valid (%)	89.2	90.1	89.4	90.9	88.1	45.4	0.7
		unique (%)	86.8	87.5	88	88.5	86.1	39.9	0.7
	250k	valid (%)	95.9	94.9	95.6	95	94.3	62.7	1
		unique (%)	95.1	94.4	93.6	94.1	93.9	62.3	0.9

Table 4.25: LSTM model: 4th test SMILES validation data

Using the best parameters derived from the previous 3 tests the float model was able to achieve 95.9% valid SMILES with 95.1% of them unique when 250k samples were trained through 24 epochs. Quantization performance was very satisfactory with the 4 bit model achieving 93.9% unique SMILES, only 1.2% worse than the float model. 6 bit quantization performed 1% worse. 8 bit generated more valid SMILES than 6 bits but performed 1.5% worse, being an outlier. 16 bits had only 0.7% worse performance.

Increasing the number of samples had the biggest performance gains of this test as increasing the number of epochs without using more training samples actually performed worse for the LSTM models as overfitting started occurring. Quantization under 4 bits is not advised as even in the best case scenario the performance dropped 30% going from 4 to 3 bits.

4. Experimental Results

4.3.5 LSTM - Final Analysis

Figure 4.7, display the validation data from table 4.14, figure 4.8 from tables 4.17 and 4.18, figure 4.9 from tables 4.21, 4.22 and 4.23 and figure 4.10 from table 4.25, offering a better representation of performance differences between models.

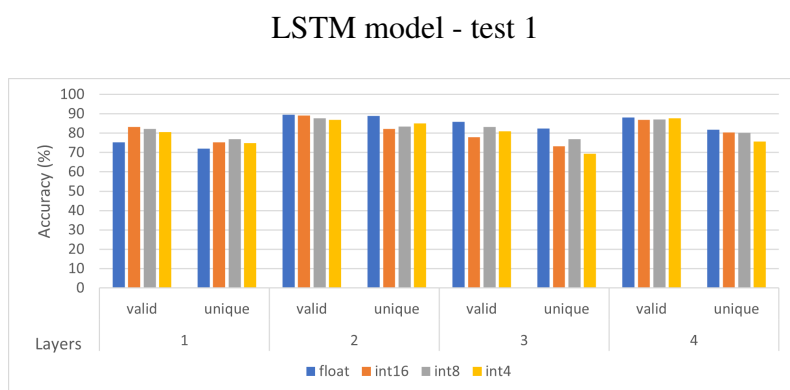


Figure 4.7: LSTM - Valid and Unique SMILES by number of layers

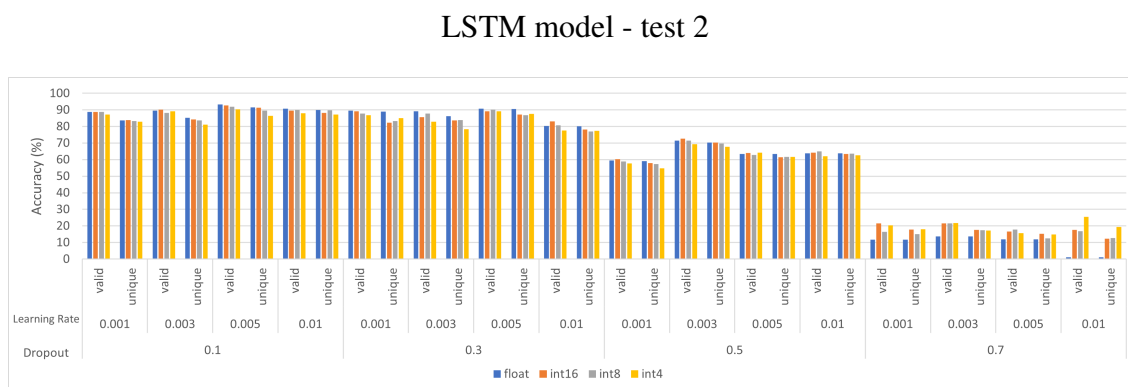


Figure 4.8: LSTM - Valid and Unique SMILES by dropout and learning rate

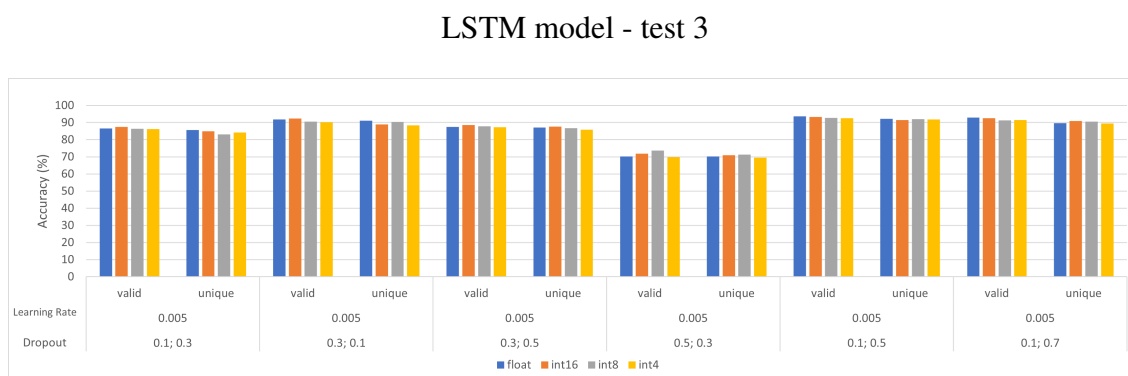


Figure 4.9: LSTM - Valid and Unique SMILES by dropout per layer and learning rate

LSTM model - test 4

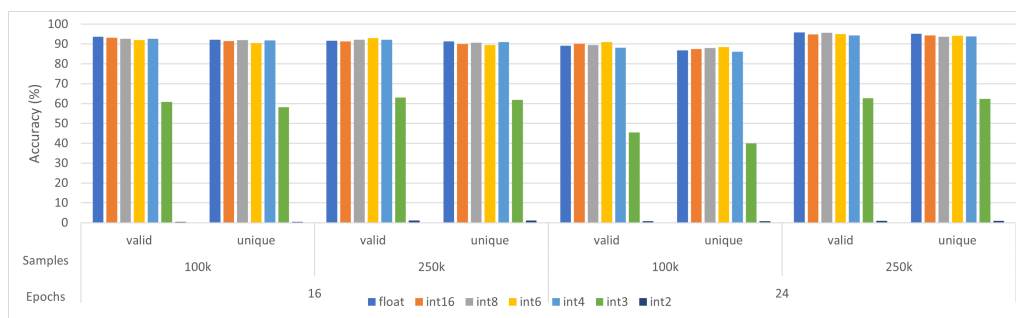


Figure 4.10: LSTM - Valid and Unique SMILES by number of samples and epochs

The number of layers used had a significant effect on the performance of the LSTM models. Using 2 layers had the best performance and increasing it further leads to slower training without any benefit. LSTM models perform better for smaller dropouts. A learning rate of 0.003 or 0.005 is optimal for training when compared to lower or higher values. By using different dropout for the first and second layers the model achieved decent improvements over the best parameters from the previous test. The models benefit from having a smaller dropout in the first layer and a larger one in the second. Increasing the number of samples was able to improve the SMILES generating abilities of the models substantially. Training for a bigger number of epochs was only beneficial when also increasing the number of samples used.

The quantized models were able to achieve very good results when using 4 or more bits. The LSTM model can be quantized very effectively when optimal parameters are used, generating only fewer 1.6% valid and 1.2% unique SMILES when using 4 bit int when compared to float. Quantization under 4 bits proved to be ineffective leading to over 30% drop in accuracy.

4. Experimental Results

4.4 Recurrent models - Performance comparison

Figures 4.11, 4.12 and 4.13 show the error for the best performing parameters from each corresponding test, comparing the Simple RNN, GRU and LSTM models.

Error analysis - Test 1 - Comparison between recurrent models

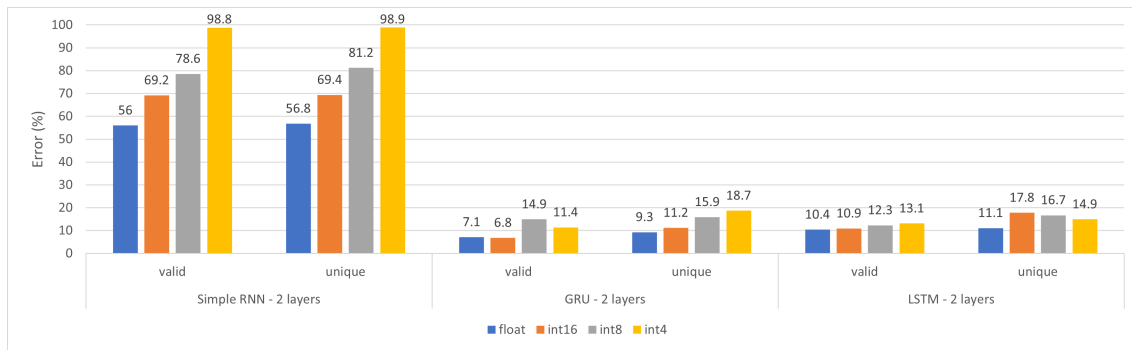


Figure 4.11: First test - Error graphic comparing performance of recurrent models

Error analysis - Test 2 - Comparison between recurrent models

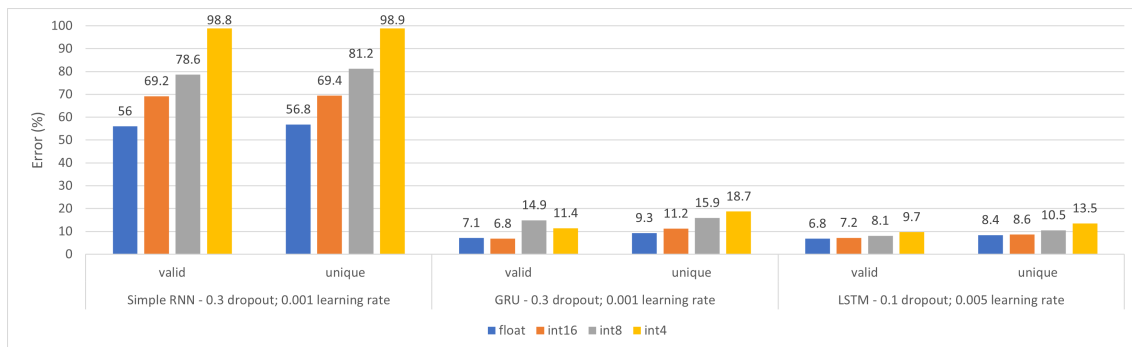


Figure 4.12: Second test - Error graphic comparing performance of recurrent models

Error analysis - Final test - Comparison between recurrent models

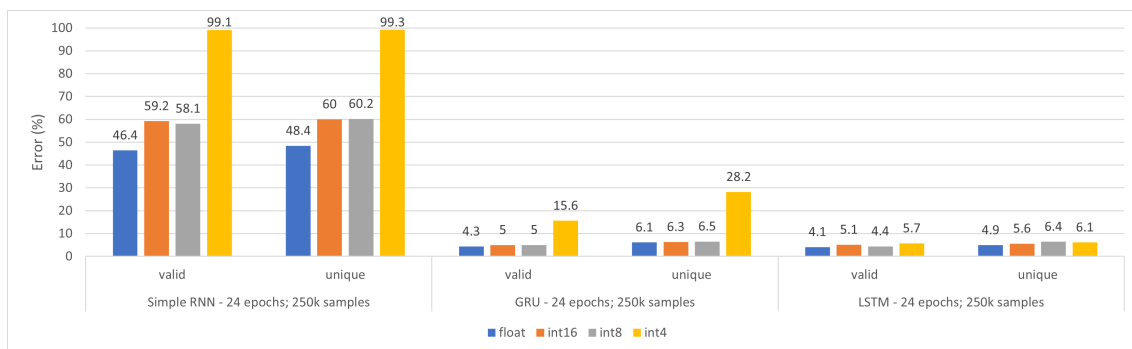


Figure 4.13: Final test - Error graphic comparing performance of recurrent models

4.4 Recurrent models - Performance comparison

Looking at the error graphics its clear how the three models perform differently in term of quantization and overall performance. The error in the Simple RNN models increase substantially from an already high value when quantization is applied. This approach is ineffective and the Simple RNN model should be discarded as an option for generating SMILES.

The GRU models have greatly improved metrics when compared to the Simple RNN models, falling only a bit short of the LSTM models. Quantization for the GRU models is very effective at 16 bits across every experiment. 8 bit quantization needs a lot of parameter optimization to be effective when compared to float, generating good results in the final test. 4 bits quantization was unable to scale for the GRU models and performed even worse during the final testing.

The LSTM models were able to train at reduced precision with favorable results across most tested parameters. Quantization was also able to scale very well even up to 4 bits precision, achieving results withing 2% of the best performing float model. The higher complexity of the LSTM layers, despite needing more time to train, lead to less performance degradation when quantization is applied, enabling lower bit number formats to be used.

Figures 4.14, 4.15 and 4.16 showcase how training a model using the proper recurrent layers and parameters leads to results of much higher quality.

Example of SMILES generated by the best 4 bit Simple RNN model

```
1 CN (AAAAc) AAAAAAAAAAAAAAAAAAAAAAFAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAA (AACNAAAAA [ AAAAAAAAAAAAAAAAAAAAAA
2 CN1AAAAA1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAAAAAA CAAAAA2AAAAAAAAAAAA CAAAAA CAAAAAAAAAAAAA
3 CO (AAAAAAAAAAAA CAAAAA CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAAAAAAAAAA #AAAAAAAAAAAA CAAAAAAAAAAAAAAAAAAAAA
4 CO1AAAAAAAAAAAAAAAAAAAA CAAAAA #AAAOAAAAAAAAAAAA (AA2AAAA CAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAAAAAAAAAA1A
5 C2CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAAAAAAAAAA OAAAA CAAAA CAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAA
6 CO1AAAAAAAAAAAAAAAAAAAA2AAAAA1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAA CAAAA CAAAAA CAAAAA1AAAAAAAAA
7 CF1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA #ACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAOAAAA
8 CN1AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA
9 CO1AAAAAAAAAAAAAAAAAAAA CAAA [AAcAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA CAAAAA CAAAAA CAAAAA CAAAAA CAAAAA
10 CN1AAAAAAAAAAAA CAAAAAAAAAAAAAAAAAAAAAAAAAAAAA #AAAAAAAAA CAAAAAAAAAAAAAAAAAAAAAAAAAAAAA (AAAAAAAAAAAA
```

Figure 4.14: Sample of SMILES generated by the best 4 bit Simple RNN model

Example of SMILES generated by the best 4 bit GRU model

```
1 CC (=O) Nc1ccc2C (=O) C (C) (C) Oc2c1
2 COc1ccc (cc1) C (=O) Nc1ccc (OC) c (OC) c1
3 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
4 CN1CCC (CC1) NC (=O) c1ccc (cc1) C (=O) NCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
5 CC (C) C (=O) Nc1ccc (cc1) -c1ccc (c1) C (=O) NCCCC (=O) Nc1ccc c1
6 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
7 CC (C) CC (C) C (=O) Nc1ccc (OC) cc1
8 CC (C) C (=O) NCC (=O) N (Cc1cccc (Cl) c1) C (=O) c1ccc c1
9 CC (C) CC (NC (=O) CCc1ccc (cc1) C (F) (F) F) C (F) (F) F
10 Cc1ccc (cc1) c1=C (C (=O) NC (Cc2cccc2) C (O) =O) C (=O) N1Cc1ccc c1
```

Figure 4.15: Sample of SMILES generated by the best 4 bit GRU model

4. Experimental Results

Example of SMILES generated by the best 4 bit LSTM model

```
1 CN1CCC (CC1) NC (=O) C (CC (C) (C) C) C (=O) NCC1CCCCC1
2 COc1ccc (C=CC (=O) c2ccc (CC (=O) Nc3ccccc3) cc2) cc1OC
3 COc1ccc (cc1) C (=O) NCCCCNC (=O) c1ccccc1
4 CC (C) C (=O) OCC1CCC (CC1) c1ccc (NC (=O) c2ccc (F) cc2) cc1
5 CC (C) C1CCC (CC1) C (=O) NCc1ccccc1
6 CC (C) (C) c1ccc (CC (=O) Nc2ccc (cc2) S (=O) (=O) Nc2ccc (F) cc2) cc1
7 COc1ccc (C=CC (=O) NCC (=O) Nc2ccc (OCC (O) =O) cc2) cc1
8 CC (C) CC (C) CNC (=O) C=Cc1ccccc1N (=O) =O
9 COc1ccc (cc1) C (=O) Nc1ccccc (c1) C (=O) Nc1ccc (cc1) C (C) (C) C
10 CC (C) C (=O) Nc1ccc (OCCC (=O) NCc2ccccc2) cc1
```

Figure 4.16: Sample of SMILES generated by the best 4 bit LSTM model

We can see how training performance is much more effective using LSTM layers in practical terms. The three samples were taken from 4 bit models, the first from the best Simple RNN model, the second from the best GRU model and the third from the best LSTM model. The first is unable to generate barely any valid SMILES and learns only that molecules start with a Carbon (C character) and little else. The second is able to generate valid and well defined strings of molecules but sometimes seems to get stuck in a loop and generates, for example, a lot of Carbons in a row, as visualized in figure 4.15. The LSTM model is able to generate almost exclusively valid SMILES, with those invalid still appearing to be well structured.

5

Conclusion

5.1 Conclusion

This work focused on the process of finding the best parameters for a NN model and how it differs when reduced precision is used during training and inference, with the goal of streamlining the process and treating the number of bits as just another variable to be taken into consideration when designing a optimized network. The best float model achieved 95.9% valid and 95.1% unique SMILES. Using the same parameters and 4 bit quantization the methods from this study were able to achieve 94.3% valid and 93.9% unique SMILES, just 1.6% and 1.2% worse than float while using 8x less bits.

Choosing the best training parameters had a significant effect on how comparatively well the quantized models performed. This method of quantization responded to different parameters in a different manner than the float model. Being more susceptible to some and in other cases even outperforming the float model when less optimal parameters were used, as in some LSTM quantized models when using high dropouts.

The dataset used was composed of strings of 100 or less characters and used a dictionary containing only a few dozen unique characters. The data strings follow the same patterns of validity but have no dependencies between each other. This makes quantization quite effective for this dataset, as reducing the precision of the weights should lead to less loss of information as would be expected in problems like Natural Language Processing, the most common use of Recurrent Neural Networks (RNNs).

5.2 Future Work

This study was constrained to a limited number of variables as testing every possible parameters was unfeasible with the limited computing power available. Parameters as batch size, sampling temperature and number of units were left out of this study but would be useful to consider in a more complete research work. This work used software to convert and emulate the use of reduced precision, leading to a lot of extra operations. Implementation in optimized and dedicated hardware would allow testing how quantization improves execution time.

The computing system only allowed for the use of 250.000 training samples, a more capable system would be able to increase this number and achieve higher prediction accuracy. A quantized model took at least 6 hours to run, making running every model several times impossible for the scope of this study, future work should also verify the results with repeated runs.

Bibliography

- [1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
- [2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [3] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: <http://arxiv.org/abs/1409.1259>
- [4] P. Schneider and G. Schneider, "De novo design at the edge of chaos," *Journal of Medicinal Chemistry*, vol. 59, no. 9, pp. 4077–4086, 2016, pMID: 26881908. [Online]. Available: <https://doi.org/10.1021/acs.jmedchem.5b01849>
- [5] M. H. S. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, "Generating focused molecule libraries for drug discovery with recurrent neural networks," *ACS Central Science*, vol. 4, no. 1, pp. 120–131, 2018, pMID: 29392184. [Online]. Available: <https://doi.org/10.1021/acscentsci.7b00512>
- [6] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988. [Online]. Available: <https://pubs.acs.org/doi/abs/10.1021/ci00057a005>
- [7] M. Z. Alom, A. T. Moody, N. Maruyama, B. C. V. Essen, and T. M. Taha, "Effective quantization approaches for recurrent neural networks," 2018.
- [8] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, p. 675–686, Jun 2021. [Online]. Available: <http://dx.doi.org/10.1038/s42256-021-00356-5>

Bibliography

- [9] T. Hwang, “Computational power and the social impact of artificial intelligence,” 2018. [Online]. Available: <https://arxiv.org/abs/1803.08971>
- [10] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, “AI accelerator survey and trends,” *CoRR*, vol. abs/2109.08957, 2021. [Online]. Available: <https://arxiv.org/abs/2109.08957>
- [11] B. Moons, K. Goetschalckx, N. V. Berckelaer, and M. Verhelst, “Minimum energy quantized neural networks,” 2017.
- [12] A. Hintze, “Understanding the four types of ai, from reactive robots to self-aware beings,” *The Conversation US*, 11 2016. [Online]. Available: <https://theconversation.com/understanding-the-four-types-of-ai-from-reactive-robots-to-self-aware-beings-67616>
- [13] B. Mesko and M. Görög, “A short guide for medical professionals in the era of artificial intelligence,” *npj Digital Medicine*, vol. 3, 09 2020.
- [14] R. Sathya, A. Abraham *et al.*, “Comparison of supervised and unsupervised learning algorithms for pattern classification,” *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 2, pp. 34–38, 2013.
- [15] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, “Recent advances in convolutional neural networks,” *Pattern Recognition*, vol. 77, pp. 354–377, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0031320317304120>
- [16] M. Coskun, O. Yildirim, A. Uçar, and Y. Demir, “An overview of popular deep learning methods,” *European Journal of Technique (EJT)*, vol. 7, no. 2, pp. 165 – 176, 2017.
- [17] “Neural networks.” [Online]. Available: <https://www.ibm.com/cloud/learn/neural-networks>
- [18] M. Uzair and N. Jamil, “Effects of hidden layers on the efficiency of neural networks,” in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, 2020, pp. 1–6.
- [19] L. Shukla, “Designing your neural networks.” [Online]. Available: <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>

- [20] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018. [Online]. Available: <https://arxiv.org/abs/1811.03378>
- [21] "Activation functions." [Online]. Available: <https://docs.paperspace.com/machine-learning/wiki/activation-function>
- [22] D. Chang and A. Pathak, "Effect of batch size on neural net training," *Medium online platform*, 2020. [Online]. Available: <https://medium.com/deep-learning-experiments/effect-of-batch-size-on-neural-net-training-c5ae8516e57>
- [23] F. Bonassi, M. Farina, J. Xie, and R. Scattolini, "On recurrent neural networks for learning-based control: recent results and ideas for future developments," 2021. [Online]. Available: <https://arxiv.org/abs/2111.13557>
- [24] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [25] S. Varsamopoulos, K. Bertels, and C. Almudever, "Designing neural network based decoders for surface codes," 11 2018.
- [26] J.-L. Reymond, L. Ruddigkeit, L. Blum, and R. van Deursen, "The enumeration of chemical space," *WIREs Computational Molecular Science*, vol. 2, no. 5, pp. 717–733, 2012. [Online]. Available: <https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1104>
- [27] "Pubchem compound summary." [Online]. Available: <https://pubchem.ncbi.nlm.nih.gov>
- [28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from [tensorflow.org](https://www.tensorflow.org). [Online]. Available: <https://www.tensorflow.org/>
- [29] F. Chollet *et al.*, "Keras," <https://keras.io>, 2015.
- [30] Google, "Qkeras," <https://github.com/google/qkeras>, 2020.
-

Bibliography

- [31] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. E. Bolton, “PubChem in 2021: new data content and improved web interfaces,” *Nucleic Acids Research*, vol. 49, no. D1, pp. D1388–D1395, 11 2020. [Online]. Available: <https://doi.org/10.1093/nar/gkaa971>
- [32] M. Swain, “Molvs: Molecule validation and standardization,” 2016. [Online]. Available: <https://molvs.readthedocs.io/en/latest/>
- [33] “Rdkit: Open-source cheminformatics software.” [Online]. Available: <https://www.rdkit.org>
- [34] “Simplernn keras layer.” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN
- [35] “Gru keras layer.” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU
- [36] “Lstm keras layer.” [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

