Coursework Assignment – Generating Mathematical Questions

ECS7022P Computational Creativity– Spring 2022
Tolly Collins – 200553283

1 Introduction

1.1 Overview and aims

Generative mathematics is a relatively un-ploughed field in the realm of deep learning, compared with the classic fields of natural language processing, vision, audio, time-series forecasting and many others. There are many examples of DNNs being applied to solve mathematical problems, such as simple arithmetic [1] or more complex differential equations [2]. In examples such as the latter, the network can effectively learn a numerical approximation function for an analytical mathematical process, with applications in a wide array of fields such as predator-prey models in biology [3] and plasma behaviour in physics [4]. There have been applications of DNNs to solve mathematical word problems using natural language processing to extract semantic information for a worded problem before abstracting to a mathematical representation [5]. Computer vision techniques have also been used in reading multi-digit numbers [6].

In this study, the aim is to concentrate more on the generative side of computer mathematics. However, this is conducted with a view to how the outputs of the model could then be used by a secondary problem solving system. The greater aim would be to design a system which can create mathematics problems suitable for aiding the learning of school children (or just for general interest and entertainment!), with both the question and worked solution being generated. We view a mathematical problem as a linked pair of question and solution, with the question being governed by syntactic rules and the solution additionally governed by semantic rules. In this paper, the focus is on the generation of syntactically valid mathematical questions. In order to limit the scope, we look only at simple arithmetic questions, but we develop the system with a view to it being easily expanded to include any type of question.

Furthermore, as we are concentrating on the generative process, we would like to investigate how we can build a system which is capable of creating a given type of question. We decided to link this to an additional output of a suggested set of initial methods which can be applied to solve the question. The idea is that these methods would be passed to the associated ‘solver’ system as a guide for how it should proceed.

1.2 Computational Creativity and Philosophical Motivations

There are several higher-level philosophical motivations for the directions chosen in developing this system, which can be grouped into a set of principles. The first is to hand over as much control over the generation process as possible to the system. This is implemented in several ways, some of which will be described in further detail below. These include: using a generative dataset to avoid pre-chosen inputs; experimenting with learnable cost function weightings; and implementing a ‘self-reflection’ system which feeds back into the network’s training process.

The second principle is to use as little human curation as possible. The system uses a rule-based process to determine whether or not an output question is syntactically valid. Similarly, the system classifies questions by type, allowing it to determine which methods should be assigned. Furthermore, the self-reflection module can change the types of questions that are input to the system to influence its training. If it is not 'happy' with the accuracy level for the outputs of a certain type of question, it can change the proportion of those questions being presented to the generative model for training.

The third principle is to create a system which can frame its creative process for the user. One of the tasks of the self-reflector module is to communicate its internal states which are affected by the direction of the training. It also outputs information about the training process, and about the artefacts being produced.

The fourth principle is for the system to be able to widen its own generative space. The first stage of this has been implemented via the self-reflector module. For the outputs of each generation batch, the self-reflector assigns a question-type (to the syntactically valid outputs only). If an output is produced that is syntactically valid but does not belong to a question type that it is aware of, a new question type is created by parsing the question into a general symbolic representation. The user is then asked to provide a list of possible first methods with which to answer the question. This new question type is then added to the list of possible input types that can be provided by the generative dataset. The user input in this process does somewhat contravene the second principle, and an addition to the system to be explored is a method classifier that would be able to suggest possible methods based on a question's structure.

The fifth principle is for the system not to be fixed in its architecture. The system was created with the ambition of it being expandable to a more complex symbol and method vocabulary, and the first stage of this has been implemented with regards to the additional methods provided upon new question type discovery. When a new method is provided, the final layer of the method classifier is adjusted to accommodate the additional output option.

The final principle is the building of knowledge from simple foundations. The intention was to mirror the way that humans learn mathematics from simple principles, and therefore the accompanying problem solving system (to be implemented in future work) would be presented at first with only elementary questions as it builds up its expertise.

2 Methodology

2.1 Data [Rule-based generative system]

In order to investigate the architecture of such a system, we limit ourselves here to a small vocabulary of symbols from which the questions can be generated:

```
'digits': ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'],
'operations': ['+', '-', '*'],
'blank': ['_']
```

The initial methods provided were:

```
0: 'I just know it',
1: 'add 1',
2: 'add a power of 10',
3: 'subtract 1',
4: 'subtract a power of 10',
5: 'add / subtract any number',
6: 'multiply by adding',
7: 'multiply by a power of 10'
```

The first method 'I just know it' was provided to mirror the human mathematical learning process, where simple sums are first conducted procedurally before they are committed to long-term memory. For example, a two-year-old may count '4 + 6' on their fingers, whereas a four-year-old may just know that the answer is 10. The aim would be for this to be passed to the problem solving system as a possible option alongside the more procedural options. The system would implement both and accept the solution from the process with the highest output confidence. The idea would be to allow the 'I just know it' module to 'over-fit' to common inputs.

The initial question types and sub-types provided were:

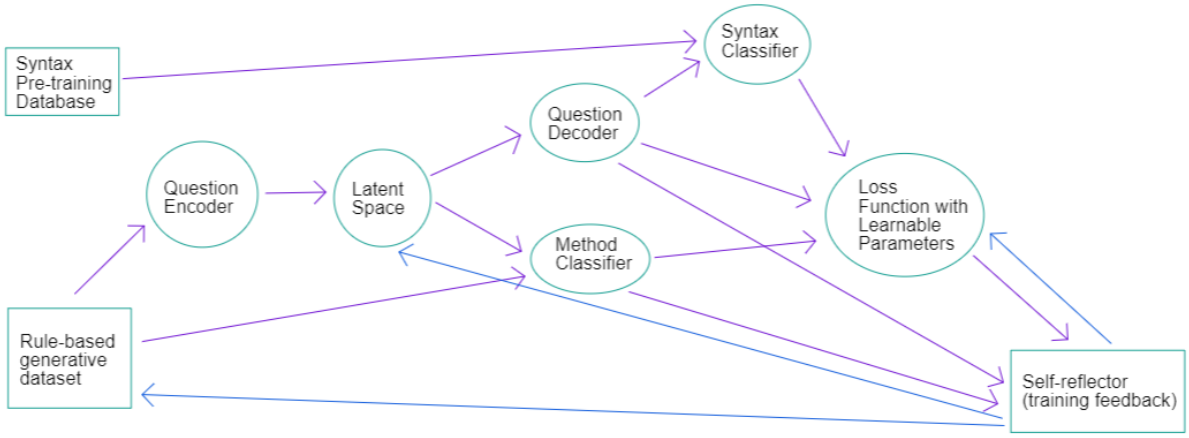
```
'addition':          [('n+1',   [0, 1, 2],     starting_confidence),
                      ('n+t',   [0, 2],        starting_confidence),
                      ('n+n',   [0, 1, 2, 5],  starting_confidence),
                      ('n+n+n', [0, 1, 2, 5],  starting_confidence)],
'subtraction':       [('n-1',   [0, 3],        starting_confidence),
                      ('n-t',   [0, 4],        starting_confidence),
                      ('n-n',   [0, 3, 4, 5],  starting_confidence),
                      ('n-n-n', [0, 3, 4, 5],  starting_confidence)],
'multiplication':    [('n*t',   [7, 6],        starting_confidence),
                      ('n*n',   [6],           starting_confidence),
                      ('n*n*n', [6],           starting_confidence)],
'mixed arithmetic': [('n+n-1', [0, 2, 3, 5],  starting_confidence),
                      ('n+n-n', [0, 2, 5],     starting_confidence),
                      ('n-n+n', [0, 2, 5],     starting_confidence)]
```

In the subtype tuples, the first element is the symbolic representation of the question, with 'n' representing an integer and 't' a power of 10. The second entries list the methods that would be suggested for beginning to solve the question, and the third entry represents a solution confidence level that would be fed back from the problem solving system to influence what types of questions would be generated. This confidence level also influences the difficulty of the questions generated, for example with a higher confidence level allowing larger integers to be picked.

The dataset then applies a rule-based algorithm to generate questions randomly within the designated constraints. The fact that the system dynamically adapts its own generation constraints based on its performance applies several of the philosophical principles described above.

2.2 Main System [Multi-task VAE]

The general structure of the system is pictured in the diagram below.



The primary system of encoder, latent space and decoder is a Variational Auto-encoder [VAE] [7], which reproduces questions provided by the generative database. As the questions are input as a character sequence, this is first passed through an embedding layer. The internal architecture was chosen to make use of the sequential nature of the input data. There has been much success recently with Temporal Convolutional Network [TCN] applications to sequence modelling due to their large receptive fields and reduced computational cost relative to Recurrent Neural Networks [8]. Influenced by the TCN architecture, we used increasing dilation factors in successive convolutional layers to compress the information. Part of the motivation for this was the nature of the input sequences, which could not be down-sampled in the same manner as, for example, audio or visual data without losing the higher-level semantic content. The assumption was therefore that dilation would produce better results than max-pooling. Convolution was chosen to calculate the latent space distribution parameters to retain the positional nature of the information.

The VAE is extended to a multi-task VAE with the method classification branch. The syntax classifier determines whether or not the questions are valid, and these outputs are all sent to the loss network, which can be given learnable weight parameters. As described above, self-reflector then reviews the outputs and performs feedback tasks as well as holding internal states which are influenced by the success and direction of the training and generation processes.

2.3 Syntax classifier [GAN-like discriminator]

It was found that when training the reconstruction of question sequences, close matches could be produced that were syntactically invalid. In order to overcome this problem, a syntax element was added to the loss function. The issue is that the process of a rule-based syntax checker is not differentiable, and so backpropagation would not be possible. The solution was influenced by the Generative Adversarial Network architecture [9], which uses a secondary 'discriminator' network to learn to distinguish real from fake inputs, while the generative network tries to create inputs that fool it. In this case, all outputs were provided by the generator network, with 'real' and 'fake' an analogy for syntactically correct or incorrect. The role of the discriminator network was to provide an approximation of the syntax checker that could be differentiated back through in order to update the generator section.

The syntax classifier was pre-trained with 50/50 valid/invalid data, with labels S verified by the rule-based syntax checker. The discriminator would therefore aim to minimise $F\big(G(x)\big) - S$, while the generator would aim to minimise $F\big(G(x)\big) - 1$ in the following generative training.

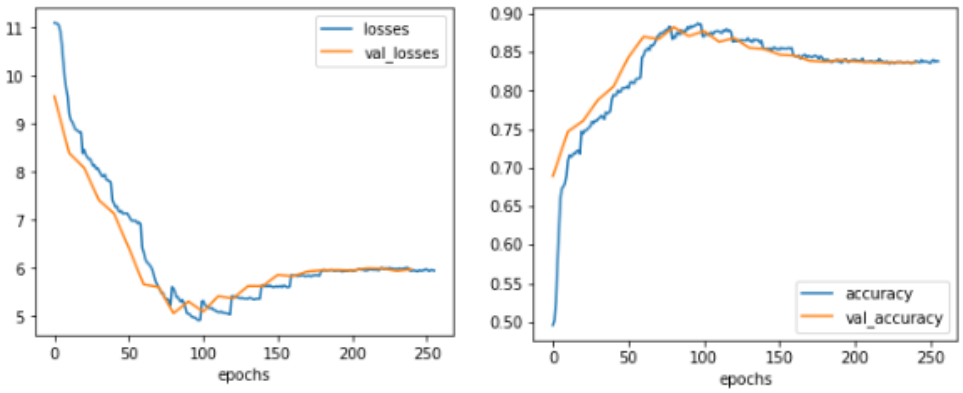
2.3 Artefact Generation and Control of the Latent Space

In order to generate new artefacts, the decoder section of the VAE could be detached from the encoder and the latent space distribution could be sampled. The VAE architecture was chosen as it allows for a continuous latent space, where the region between input types can be explored. Part of the motivation for including the method branch was to allow for further control of the latent space, as investigated by Brunner et al [10]. After each training batch, the self-reflector module records the average latent space μ_q and $\ln(\sigma_q)$ for each question type. As each question type is typified by its set of methods, the idea is that the additional classifier branch should help to organise the latent space by question (or at least by method) type. When generating a new question, a sample could then be taken from the Gaussian distribution $N(\mu_q, \sigma_q)$ rather than $N(0, 1)$ to increase the likelihood of receiving a question of the desired type.

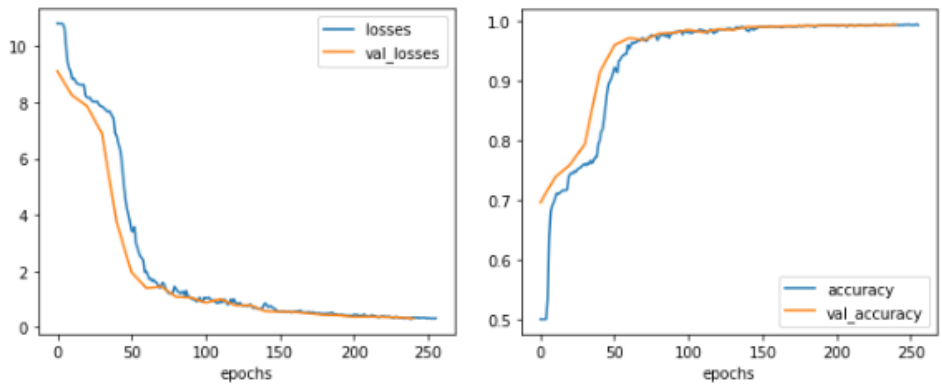
3 Results

3.1 Syntax Discriminator Pre-Training

On first training it was found that the discriminator could achieve an accuracy score of over 0.85 on validation data (see figure below). However, this was found not to be good enough to train the VAE construction affectively.

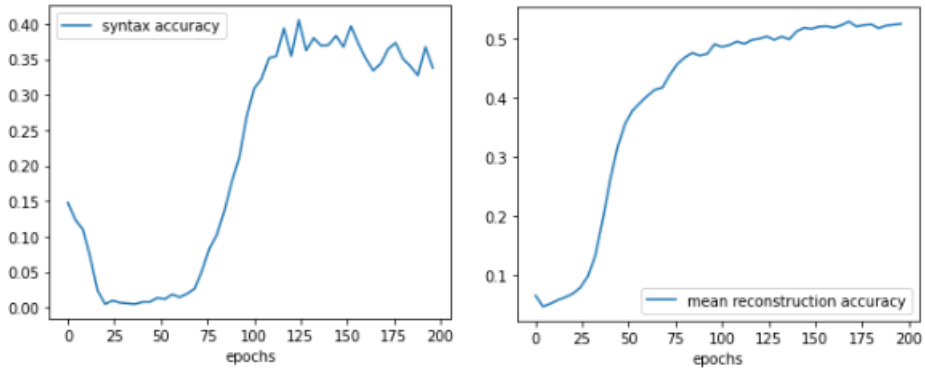


It was found that the network was under-fitting, and by adding an additional 2 layers, doubling the number of channels while increasing the dropout rate and adjusting the learning rate, the network could be trained to achieve an accuracy of 0.995 (see figure below).

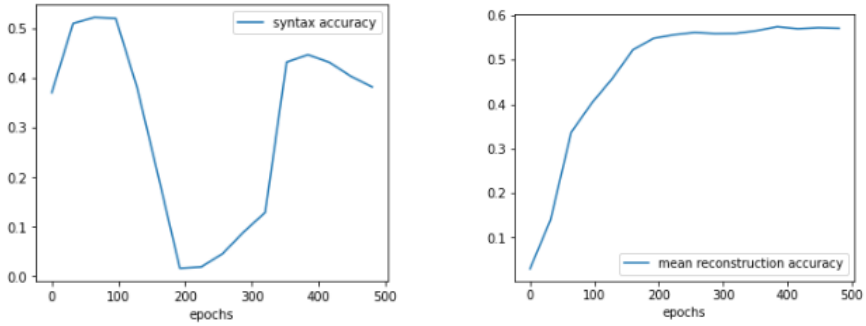


3.2 Question Reconstruction

Without the syntax discriminator, the network was able to perform reasonably well at reconstructing the question sequences, but the performance did often plateau at around 0.5 accuracy (see figure below). Interestingly, the syntax accuracy rate (proportion of syntactically valid outputs) was also reasonably high at around 0.4.



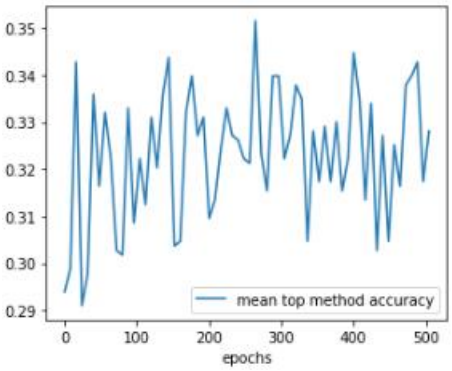
Initially, adding the syntax discriminator had a negative impact on training. It was found that balancing the weightings of the different elements of the loss function had a significant impact on the training performance. Otherwise, the accuracy scores would fluctuate as the model's learning was pushed in different directions (see figure below).



We can see above that marginally higher accuracy scores are achieved, although future work should investigate further hyper-parameter tuning.

3.3 Method Reconstruction

Adding the method branch again provided a more challenging balancing act with the loss function. We found that the method accuracy was very noisy, perhaps suggesting that the method branch required a greater complexity to be able to learn effectively. This could also reflect the network trying to fulfil all of the different criteria, and as above it took a significant amount of time for the training to stabilise.



3.4 Trainable Loss Function

The initial approach was similar to that suggested by Liebel & Korner [11], with a loss function given by:

$$L_T(y_T, y'_T) = \sum_{t \in T}\left(\frac{1}{2c_t^2} L_t(y_t, y'_t) + \ln(1 + c_t^2)\right)$$

Here, each c_t is a learnable weight parameter and $\ln(1 + c_t^2)$ is a regularisation term to avoid degenerate solutions. However, we discovered that this approach suffered from an initial imbalance where all of the weights would decrease to reduce the positive regularisation term, resulting in much slower learning for the rest of the network. The solution to this was to alter the regularisation term to include a penalty for the average of the weights rather than just the weights themselves. This meant that the weights were free to vary relative to each other while still not being allowed to all degenerate towards 0. A small additional regularisation term stopped any individual loss weight from decreasing too far. The final loss function was:

$$L_T(y_T, y'_T) = \sum_{t \in T}(m_t \cdot r_t \cdot |c_t| \cdot L_t(y_t, y'_t) + \lambda_1 |\ln(c_t)|) + \lambda_2 \left|\ln\left(\frac{1}{|T|}\sum_{t \in T} c_t\right)\right|$$

In the above function, m_t are manual weightings which can be given to each loss term, r_t are the additional weight provided by the self-reflector module, c_t are the learnable weights and λ_i are the regularisation weightings which can be adjusted as hyper-parameters.

3.5 Latent space organisation and question generation

We found that the average normal distribution per question type was indeed different, suggesting that this could be utilised to increase the probability of generating question of a given type. However, within the scope of this study we were not able to investigate the level of disentanglement in the latent space, and how this can be controlled with hyper-parameters. We leave the investigation and tuning of this behaviour to further study.

3.6 Generation Artefact Examples

This system was very adept at discovering new types of questions, and it was enjoyable to engage in the classification of different method approaches alongside the model's generation. An example of an unexpected new question type was:

```
I have discovered a question with a structure of -n that I have not seen before: --6
```

This was designated a new 'double negative' method, which was automatically added to the methods dictionary. As we had not anticipated this type of question, we see that the structure has been parsed incorrectly to '-n'. Further examples of new types of questions generates are:

```
--3-0-3066
```
```
9+1+1999
```
```
952+91*928682
```
```
982*1*969928788
```

We see that some huge numbers are included in the questions, and this suggests an improvement to the system which would be to apply the integer range cap to the accepted generation outputs as well as to the inputs.

4 Discussion and Evaluation

Although there remains significant extra investigation to be conducted in terms of tuning the system and investigating the capabilities of what it can generate, we believe that a highly promising setup has been presented. The primary aim was to create a model which took significant control over the generative process. The system is a combination of five elements: a rule-based question generator, a multi-task question and method VAE, a GAN-style syntax validity discriminator, a loss network with learnable weights, and a rule-based self-reflector module that takes a degree of control over the training process.

In terms of Colton's FACE criteria, we feel that the 'g' criteria of concept, expression of concept and framing are clearly met. The aesthetic measure criterion is interesting, as this generative process does not really concentrate on aesthetic artefacts. However, the model does review its own artefacts in many ways, perhaps most relevantly in reviewing the syntactic validity which is in essence their mathematical suitability. Furthermore, the self-reflector module responds to the level of accuracy and novelty on the outputs by changing the training direction through loss weightings and changing question input type balance. We also believe that the C^p criterion of having a method for generating concepts is met, as the system can not only generate new types of questions, it also knows when it achieves that. The system also takes control of the likelihood of generating novel material as opposed to concentrating on reproduction accuracy. As a result of the new artefact types, the system changes the model's architecture during training in terms of the method output layer.

There were several exciting concepts explored in this study, including using a discriminatory network to learn an approximation for a non-differentiable metric, applying 'mean regularisation' to the terms in a learned loss function to allow weights to differ relative to each other in a controlled manner, and the use of a 'self-reflector' module to take a degree of control over the learning of the system, feeding back into the training loop. Furthermore, the tracking of latent space parameters for different classes showed promise and remains an area for further investigation.

References

[1] – Franco & Cannas (1997) – Solving Arithmetic Problems using feed-forward neural networks. *Neurocomputing*

[2] - Chakraverty & Mall (2014) - Regression-based weight generation algorithm in neural network for solution of initial and boundary value problems. *Springer*

[3] – Umar et al (2019) - Intelligent computing for numerical treatment of nonlinear prey–predator models. *Elsevier*

[4] – Raja et al (2018) - Design of artificial neural network models optimized with sequential quadratic programming to study the dynamics of nonlinear Troesch's problem arising in plasma physics. *The Natural Computing Applications Forum*

[5] – Wang et al (2019) - Template-Based Math Word Problem Solvers with Recursive Neural Networks. *Association for the Advancement of Artificial Intelligence*

[6] – Hoshen & Peleg (2016) - Visual Learning of Arithmetic Operations. *Association for the Advancement of Artificial Intelligence*

[7] – Diedrik & Welling (2013) – Auto-Encoding Variational Bayes. *ArXiv*

[8] - Bai et al (2018) - An empirical evaluation of generic convolutional and recurrent networks for sequence modelling. *ArXiv*

[9] – Addarwal et al (2020) - Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*

[10] – Brunner et al. (2018) - Midi-Vae: Modeling Dynamics and Instrumentation of Music with Applications to Style Transfer. *ISMIR*

[11] – Liebel & Korner (2018) - Auxiliary Tasks in Multi-task Learning. *Technical University of Munich*

[11] – Colton et al (2011) - Computational creativity theory: The FACE and IDEA descriptive models. *ICCC*