Program Automation: Learning from Tests defined in Unit Testing

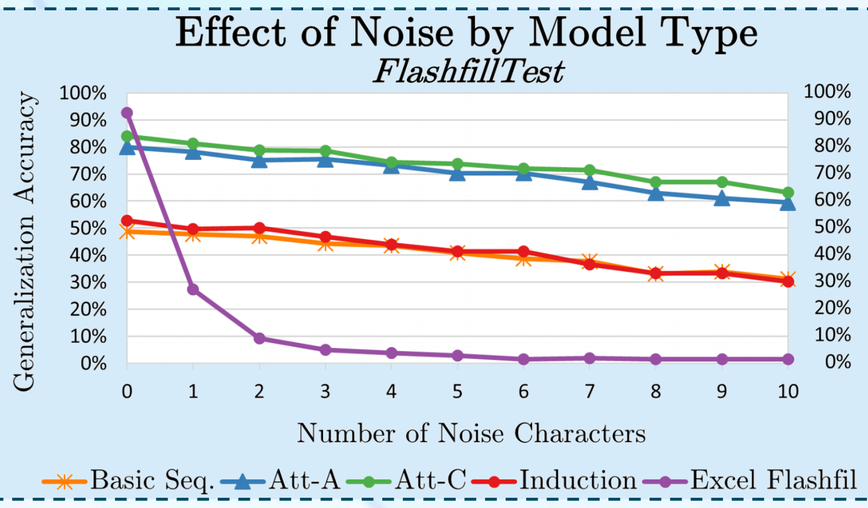
Rihards Baranovskis (MSc Computer Science)
Supervisor: Dr Robert J. Walters

1 PROBLEM

Program automation deals with mechanisms, which allow computers to synthesise programs from high level definitions. Machine learning based program synthesis has gained recent researchers` attention due to rise in effectiveness of neural networks. Current research mainly focuses on program synthesis, which is based on high level – linguistic or input/output example definitions for automating the process for people who are non-programmers [1][2][3]. Little research has been done in regards of implementing such synthesis models in the actual software development process, carried out by IT professionals.

2 BACKGROUND

Program automation is one of the initial problems that Artificial Intelligence and Machine Learning has been trying to solve since the 70ties [4]. Traditionally, two approaches are contrasted to each other – program synthesis and program induction. The first one synthesises code, whereas the latter one induces a latent representation of a desired function. Researchers at Microsoft [2] have compared the both implementations using neural network models to replicate Excel's Flashfill functionality. Neural synthesis method has shown near identical performance to the original program, moreover both machine learning based methods have performed much better under noisy inputs.



Comparison of Neural Synthesis/Induction with the original program [2].

3 AIMS

The aim of this project lies in the following research question is: *Can recurrent neural networks be used effectively to synthesise software from unit tests` definitions?* As result this project has the following objectives:

- *Familiarisation with recurrent neural networks based program synthesis,*
- *Designing a suitable neural programmer architecture,*
- *Training, tuning, and testing the model,*
- *Comparing the results with non-machine learning based methods,*
- *Testing the architecture at a lower programming level.*

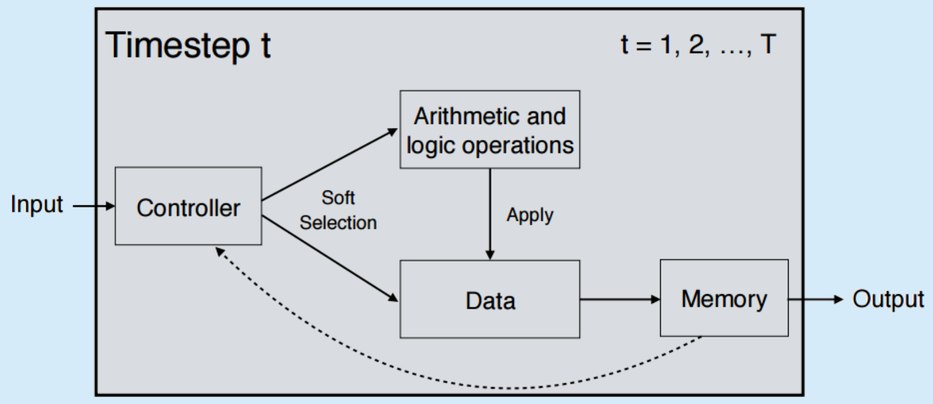
$I_1 = \texttt{January}$	$O_1 = \texttt{jan}$
$I_2 = \texttt{February}$	$O_2 = \texttt{feb}$
$I_3 = \texttt{March}$	$O_3 = \texttt{mar}$
$I_1^y = \texttt{April}$	$O_1^y = \texttt{apr}$
$I_2^y = \texttt{May}$	$O_2^y = \texttt{may}$

Sample train and test unit tests to be used [2].

Appropriate induction model architecture has to be developed for inducing software from unit test definitions. Known ALU and recurrent neural networks based Neural Programmer architectures exist for inducing latent programs from high level linguistic definitions [3]. These will have to be redesigned to fit the research problem, then trained and tested.

4 METHOD

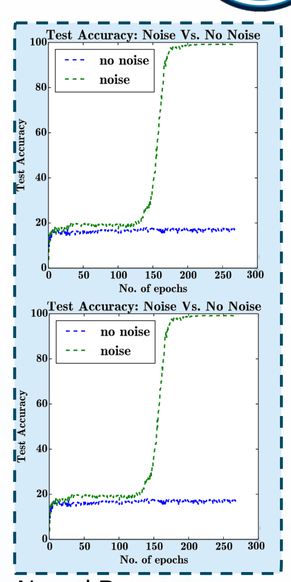
The program induction model used in the research is going to be based on the Neural Programmer architecture, which consists of recurrent neural network based controller, ALU and Memory units [3]. The main research method will contain using an iterative process of finding the most suitable neural programmer architecture with the best corresponding hyperparmeters for the trained recurrent neural network. The model is going to be built using Tensorflow Machine Learning library [5] and possibly trained on a GPU cluster. The analysis of the final results will be conducted by comparing them to the non-machine based synthesis tools, such as Microsoft Prose [6]. The objective measure of success will be defined as the difference in the test results between the developed model and non-machine learning based models.



Neural Programmer Architecture [3]

5 EXPECTED RESULTS

Neural Programmer model, designed by Google, has achieved near 100% test accuracy for the "questions about a dataset" problem [3]. It is expected, when given reasonable amount of training data, the program induction model, which is trained with unit tests definition, can potentially achieve the same performance. The model will be trained to induce functions of varied difficulty, starting from simple mathematical operations up to scalar output based programming challenges, taken from sources such as Codility [7], Coderbyte [8] and others. As result, it is expected to gain varied testing accuracy responses, depending on the complexity of the function which is attempted to be reproduced.



Neural Programmer Train and Test Results [3]

6 WHY THE PROJECT MATTERS

If thirstiness of statistical program learning for I/O samples is reduced, current advances can lead to development of self-learning models, which can replace some parts of software development, when test driven development process is applied. Furthermore, since statistical learning generalizes a model better, when it is trained with noisy inputs [2] [3], machine learning based approach has a potential of producing more robust generic programs than the non-statistical methodologies.

7 REFERENCES

[1] E., Mohamed, A.R., Singh, R., Li, L., Zhou, D. and Kohli, P. Parisotto, "Neuro-Symbolic Program Synthesis.," arXiv preprint arXiv:1611.01855, 2016.
[2] J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A.R. and Kohli, P.Devlin, "RobustFill: Neural Program Learning under Noisy I/O.,"arXiv preprint arXiv:1703.07469, 2017.
[3] Q.V. and Sutskever, I. Le, "Neural Programmer: Inducing Latent Programs with Gradient Descent," arXiv preprint arXiv:1511.04834, 2016.
[4] Richard J. and Lee, Richard C. T. Prow Waldinger, "A step toward automatic program writing.," in IJCAI, 1969.
[5] (2017) Recurrent Neural Networks. [Online]. https://www.tensorflow.org/tutorials/recurrent
[6] (2017) Microsoft Program Synthesis using Examples SDK. [Online]. https://microsoft.github.io/prose/
[7] (2017) Refactor yourself. Train your programming skills - Codility. [Online]. https://codility.com/programmers/
[8] (2017) Coderbyte | Programming challenges and courses. [Online]. https://coderbyte.com/challenges

UNIVERSITY OF Southampton
SCHOOL OF Electronics and Computer Science