Automatic Analog Design Parameter Tuning using Reinforcement Learning

Jay Zhe-An Mok
ECE Department of UBC
Vancouver, Canada
Jm4304@student.ubc.ca

Abstract— **A general RL framework for optimizing the parameters of an analog design is constructed. The RL agent takes in the analog design parameters as the current state and design metrics such as gain, bandwidth and noise as rewards, and predicts a new set of design parameters as the action for the next state. Through iteratively running the RL, the agent is able to find a set of designs that are not only on the Pareto front but also outperform manual and randomly optimized designs by up to 30%. The entire process is automated, runs Cadence Spectre for analog simulation and the code can be found at github.com/liu2333hui/rl4analog**

Keywords—Analog Design, RL, Spectre, Cadence

I. Introduction

Analog design is an important part of today's integrated circuits (ICs) and System on Chips (SoCs), as many important modules such as ADCs, amplifiers, filters, DACs need to be interfaced with the digital domain and meet stringent performance, area and power goals. Many analog design methodologies have been introduced and researched, ranging from more manual equation-based approaches (Overdrive methodology for amplifiers, switch capacitor filters [6]), to look-up table or design table approaches (Gm over id [3]), to simulation-based look-up table approaches (high order Sigma Delta design [5], band-gap references [4]), or hierarchical / modular approaches. In short, many methodologies provide an initial solution that then has to be fine-tuned in order to reach certain optimization design goals. As a result, this process can be very time-consuming, and is difficult when equation-based analytical methods may not exist, such as the case for high order Sigma Delta design [5].

With the recent success of deep learning in modelling highly complex systems, and using reinforcement learning in solving optimization problems and the capability to re-use AI models in new scenarios, recent researchers have begun to investigate how to use AI-based RL algorithms to do analog design optimization and design space exploration [2]. One previous work involved training an RL agent using graph neural networks to learn from transistor properties such as threshold voltage, mobility, gds, and gm to then predict transistor and RC sizes [1]. They also demonstrate that their AI, once trained on a given circuit, can then be used to design the same topology circuit but in a different technology node. Their trained AI can also speed-up the training of another topology circuit but with less iterations. Another work takes the transistor sizes as the input directly, uses a neural network as the AI, and also uses the AI to predict a new set of transistor sizes and RC design values to simulate and design for. Other approaches are similar in that they have a loop around an analog simulator such as Cadence Spectre which once given a set of design parameters, will output circuit metrics such as power, bandwidth, gain which will then be used as feedback or rewards for the AI to continuously optimize for.

Some limitations of the previous work include a focus on transistor size automation, which is only applicable to some circuit designs especially amplifiers. Furthermore, there are cases where "abstracted" hyper-parameters can also be optimized for but cannot be directly controlled by previous methods, such as scaling factors, number of quantization levels of an ADC, and so forth. It is also not clear from previous work how outputs of the AI-agent are scaled / normalized, as the choice of range to select parameters can greatly affect the final optimization result (too big can result in difficulty in finding a solution, too small can result in non-global optimized designs). Thus, a more general RL approach is needed to cover cases where the design may not even be an explicit transistor circuit design, but an overall hierarchical design of an analog circuit such as a PLL.

This work tries to alleviate these limitations by allowing the AI agent to learn from a more general form of inputs, which do not have to be transistor / RC sizes, but instead can be any hyper-parameter. The approach also introduces a training approach that iterates over a new simulated sample multiple times. Finally, it is shown that the RL agent, unlike more traditional optimization approaches that have to redo optimization every-time for a new circuit, can be re-used to optimize on new circuits on new technology nodes.

The work also tries to explore some other optimization techniques such as initial setting of hyper-parameters, choosing pareto-optimal designs as the next starting point of optimization, and using pre-learning from previous simulated samples to speed-up training.

Thus, the main contributions of the work include,

- Introducing a general approach that can optimize for general design parameters of any analog circuit with being limited to transistor / RC sizes as input to the AI
- Number of optimizations to re-use previously simulated designs in order to improve training of the AI model
- An approach that can transfer an already trained AI-model to new scenarios, such as a new technology node

II. Background

A. *Analog Design Methodologies*

Analog design is an iterative process that involves both analytical models as well as simulation methods to reach design goals. Simulation methods such as Gm-Over-Id characterize transistor operating points at different gm/id values, thereby allowing a designer to use these pre-computed tables to reach bandwidth, gain, current and other goals. Analytical models are useful in filter design, where a given transistor characteristic is designed through analytical equations relating poles to the capacitance and resistors. Other methods are a mix of both, where both simulation pre-computed tables as well as analytical models are used.

In all cases, the human designer has to ultimately pick a set-up of simulation parameters to validate, and through the simulation, get feedback to then fine-tune the parameters again until a satisfactory design is achieved.

Analog design parameters are also quite various, ranging from simple transistor sizing in amplifiers, to passive component selection and switching frequency selection for switch-capacitor filters, to over-sampling ratios and even more complex relationships of noise-shaping transfer functions against noise and input range for high-order sigma-delta simulators.

Furthermore, analog design parameters can be discrete values, such as the number of quantization levels or order of filter, to continuous values, such as over-sampling ratio, transistor widths and so forth. There are also a class of variables that are discrete such as topology type (i.e. flash ADC, interpolating, etc.).

Another interesting consideration is design parameters can also be "abstract", in the sense that they are not physical variables but more like design tuning heuristics, such as scaling factors between folding stage amplifiers, over-sampling ratio and so forth.

B. *Reinforcement Learning*

Reinforcement learning is a class of artificial intelligence that tries to model the behavior of an agent performing a set of actions in an environment, which will then give the agent a set of rewards as feedback. The goal is similar to a game, where the agent tries to score the highest points with a given set of actions. Popular examples of RL being used is in games such as Go, Chess, Pong and so forth.

Unlike other types of artificial intelligence such as machine learning and deep learning which require a large set of training data to begin with, reinforcement learning can be trained with little data. Furthermore, because RL's core function is to evaluate some set of input actions in an environment to receive a reward, this is very similar to black-box optimization techniques, where the algorithm will try to pick the best set of parameters, in RL's case the actions, that yields the optimal cost function, in RL's case the rewards which are generated from the black-box, in RL's case the environment.

Hence, RL can also be used for optimization. The main difference between RL optimization and other methods of optimization (genetic algorithm, swarm, Gaussian optimization) is that once the AI-agent is trained, it could potentially be used in new unseen contexts, such as in a new technology node. This has advantages in reducing runtime, model re-use and also in understanding how AI is trying to optimize for a circuit.

III. Methodology and Implementation

A. *Reinforcement Learning framework for simulation-based analog design*

The RL algorithm is adopted for analog circuit design, as shown in figure 1. The RL's main three components are mapped to the analog design flow.

The environment is the analog simulator, or blue tile, which will do simulations from a given set of circuit design parameters (assuming the topology circuit is already chosen, with some assumptions such as operating voltage supply, technology library, load capacitance, input drive resistance). In theory, the circuit design parameters can include these circuit environment variables, but for simplicity, they can be assumed to be already fixed. More details of what is needed to make the scripting work here is described in section 3.B.

The reward mechanism, the green tile, is a tool that takes the dc-operating, ac analysis and transient analysis from the analog simulator and extracts key design metrics, such as gain, bandwidth, gain-bandwidth product, signal to noise ratio, power, current usage and so forth. More details will be described in section 3.C.

Finally, the AI algorithm is the AI-based agent, or the orange tile. This step takes in the rewards, possibly environment settings such as the previous simulated circuit design parameters, and outputs a new set of circuit design parameters. There are many ways to implement the AI model, it can be a policy-based network, which is a single neural network that directly relates the previous set of parameters to the next set of parameters (i.e. an incremental update to the parameter to search for), or a value-based network, which . Detailed discussion is out of the scope, but implementation details are described in section 3.D.

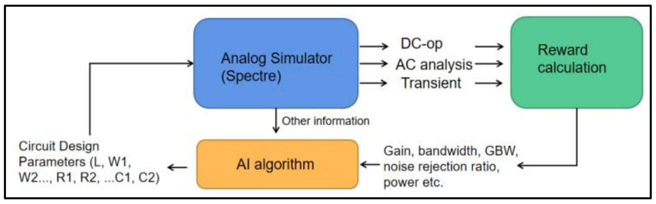

Analog Simulator (Spectre)
DC-op
AC analysis
Transient
Reward calculation
Other information
Circuit Design Parameters (L, W1, W2..., R1, R2, ...C1, C2)
AI algorithm
Gain, bandwidth, GBW, noise rejection ratio, power etc.

Figure 1. Overall RL methodology and high-level flow

B. Analog Environment

The analog simulator can be thought of as the following:

AC analysis, DC operating point, Transient = F(L, W1, W2, ..., R1, R2 ..., C1, C2, ... K1, K2...)

Where L, W, R, C, and K are technology node, transistor widths, resistances, capacitance, and other design parameters respectively. These are the absolute values and not scaled values, which will be discussed later.

The values of the input will determine the quality of the results of the given circuit at hand. In a analog simulator such as Spectre, these values are easily given through the "Design Variables" column for instance.

The "scs" format is what is passed to a circuit simulator such as Spectre. Like an Hspice file, it contains the netlist information of a circuit topology, as well as parameters for each circuit element. The elements can also be hierarchies or abstracted units, such as a voltage-controlled voltage source for modelling amplifiers or voltage-controlled resistance for modelling switches. This file is generated by the optimization flow and will need to be modified for new circuits.

One caveat of using the simulator is that a general idea of the range of the metrics is required: for example, if knowing that the bandwidth is in the GHz range, the AC analysis can simply sweep in that range, thereby reducing runtime. For transient settling time measurements, the transient simulation does not need to run longer than a single clock cycle. These types of information will require analog design expertise and some a priori knowledge. Not putting correct ranges will deteriorate the overall quality of results.

There may be cases when there are multiple "scs" files. This is because different types of analysis need to occur for different metrics, and clearly a simulation netlist suitable for bandwidth measurements is not the same as a simulation netlist for measuring error, power or settling / slew rate for an op-amp, which require ac and transient analysis respectively.

C. Reward Mechanism

The reward mechanism consists of two stages: calculating the absolute metric and calculating the reward.

The different design metrics can be calculated from the raw outputs of the analog simulator. For DC gain estimation, it is derived from the AC analysis or transient analysis. For bandwidth estimation, it is derived from the AC analysis when the DC-gain drops by 3dB. For GBW, it is the product of DC gain and bandwidth. For noise, it will depend on the system simulated: for an amplifier, noise rejection can be done with respect to the power rails or as input noise, for filters or ADCs, it can be parasitics. This can be determined by the user. Other metrics are also possible such as average power, average current which can be extracted from transient simulations or dc operating points.

Next, to calculate the reward, a simple linear weighted sum is employed,

$$\mathrm{Reward} = \sum \mathrm{W}\,(\mathrm{M} - \widetilde{\mathrm{M}})/\widetilde{\mathrm{M}}$$

Where W is a weight for each metric, M is the simulated value of the metric, and $\widetilde{\mathrm{M}}$ is the target design metric. For example, if the desired gain to achieve is 100, and the simulated gain is 50, than the reward is -0.5. If the simulated gain is 150, than the reward is 1.5.

The weight W and the units of the metric can matter: doing dB rather than absolute gain will yield smaller rewards, potentially making fine-tuning harder.

D. AI algorithm

The AI algorithm is an actor-critic model (A2C), which is one of the state-of-the-art algorithm in RL techniques. The core of the A2C algorithm are two neural networks: one neural network that takes input as the previous design parameters and outputs the next design parameters. The other neural network takes the input of the design parameters, and outputs a so-called Q-value. The first neural network is the "actor", the one where the outputs will then be fed into the environment for simulation, whereas the second neural network is the "critic", which evaluates the Q-value or quality of the set of actions. The equations can be represented as,

L', W1', W2', ..., K2', ... = ACTOR(L, W1, W2, ..., R1, R2, ... C1, C2, ..., K1, K2,...)
Q = CRITIC(L', W1', W2', ..., K2',...)

Where L, W1, ..., K2, ... are the parameters for the analog design from before but normalized between -1.0 and 1.0 in order to help converge the neural network. The L' are the new parameters predicted by the neural network. This "Q" Q-value will be discussed next.

The high-level operation of the AI algorithm is that every time a new simulation design is generated by the analog simulator, the AI model is "trained" once. "Training" here means taking the reward and using it to calculate a value called the Q-value, which represents the accumulation of the current reward and previous rewards,

$$\mathrm{Q} = \mathrm{R} + \mathrm{gamma} * \mathrm{Q}'$$

Here, R is the reward that was just calculated, Q′ is the output of the critic network assuming the input is the current set of parameters, and Q is the Q-value. This Q-value is then used to alter both the weights of the "actor" and "critic". The goal of the critic is to minimize the difference between the Q from above and the predicted Q, whereas the goal of the actor is to maximize the Q-value. This is the step that will try to guide the AI to "learn" about how rewards are related with the inputs, ultimately deriving the best outputs that will yield the highest rewards. Intuitively, if the Q is estimated too high, the weight will be lowered slightly to accommodate, and vice versa. With enough neural network weights, the AI can learn to improve the reward over simulation iterations.

E. Other enhancements

In order to make better use of the simulation data, which can be considered as few because it takes seconds to perform one full simulation of the circuit, a replay memory is used, where previous simulation is used in the training at every iteration. To further increase re-use, the training is done multiple times, 5 to 10, per simulation iteration.

Another enhancement is training in "episodes" rather than in "one-go". This means every 100 or so iterations, the current set of parameters are reset back to some set of parameters that were already explored by the agent. This can allow the agent to get out of local minima, move out of saturation, and potentially explore new design areas. The set of parameters to reset to can be chosen by the user, for example it can be a random parameter, a set of parameters determined by a manual design such as from Gm-over-id methodology, or it can be a design from the Pareto curve of already simulated designs.

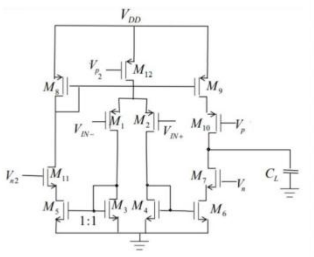
Finally, the RL training can be split into two stages: one stage is random exploration of parameters in the beginning due to lack of training samples. This number is set to 30. After these simulations, the RL begins to use the AI agent to predict new parameters. A detailed comparison of random

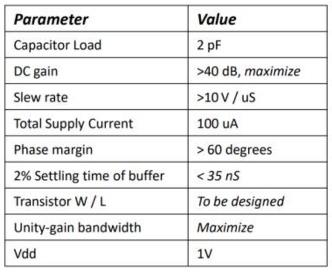
IV. Results and discussion

The RL framework was written in Python, and uses the Paddlepaddle PARL AI framework for training. The analog simulator environment is Spectre from Cadence. The CPU environment is a Intel Xeon CPU running with 12 cores. The analog simulator takes as input SCS files, which are the simulator files for Spectre, and then outputs correspond logs in "ac.ac", or "dc.opt" which include information from ac analysis, and dc-operating points respectively. Other types of analysis are also possible, such as "transient" for noise measurement purposes. These raw simulation files are then processed and used for rewards, which also has a Python script to automatically do the conversion. Then, the rewards, current set of actions are fed to the AI algorithm which is an actor2critic RL model.

The RL model uses 128 hidden nodes with 2 layers, ReLu activation functions for the actor and critic respectively as these values gave indication of training. The learning rate is 0.01, randomization exploration rate is 0.05 (5%),

A. Case study 1: two-stage voltage gain Op-amp with bias stage





Parameter	***Value***
Capacitor Load	2 pF
DC gain	>40 dB, *maximize*
Slew rate	>10 V / uS
Total Supply Current	100 uA
Phase margin	> 60 degrees
2% Settling time of buffer	*< 35 nS*
Transistor W / L	*To be designed*
Unity-gain bandwidth	*Maximize*
Vdd	1V

Figure 2. design circuit topology and requirements

A two-stage voltage op-amp with bias stages matching the gain stages is optimized using the RL technique, as shown in figure 2. An initial set of parameters was found through the Gm-over-Id methodology, similar to work [4]. This was then used as the starting point for the RL. Because the input to the AI and the output are normalized between [-1.0, 1.0], they need to be scaled to their original values. This range was chosen to be +-50% of the Gm-over-Id parameters, for the purpose of fine-tuning. For global tuning, the range can be chosen to be much larger (as seen in the next benchmark for the trans-impedance amplifier).

The parameters to be designed are the transistor widths, W1 to W10. The length is set to be fixed at 180nm for comparison against a human design. The voltage supply is 1V, and uses the 45 nm PDK transistors nmos_lvt and pmos_lvt. Because the bandwidth goal is simply to maximize, a target bandwidth say of 300MHz was randomly picked as the bandwidth goal.

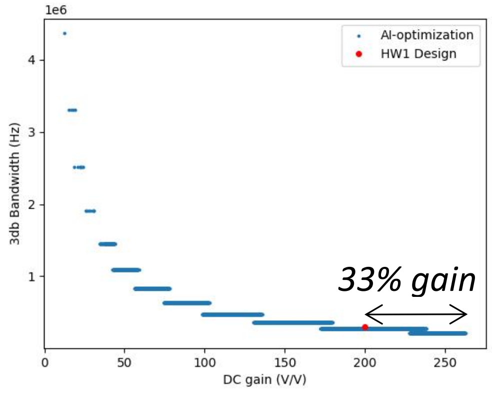


Figure 3. RL design space exploration versus manual design. There is a gain of 33% for the DC-gain found by the RL.

The plot of the different designs searched by the RL is shown in figure 3. The plot shows a Pareto curve plot, with blue dots representing the RL searched design space and red dot representing a manual Gm-over-id design. It is clear that not only is the RL able to explore more designs in similar times (i.e. manual design took more than 8 hours to fine-tune, whereas RL took around 6 hours), but also found solutions with 33% more gain as well as solutions with higher bandwidth (i.e. for a gain goal of 100, the bandwidth can reach around 500MHz).

Figure 4 shows a plot of training iteration against reward. The reward in the beginning can be very high but is not steady due to randomness in the beginning of the training, but it becomes gradually steady as the RL starts to learn the tuning. A comparison between pure random search versus RL-based search is shown in the next benchmark to see whether the RL actually is contributing to the search.

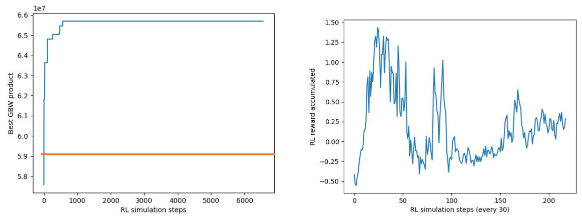


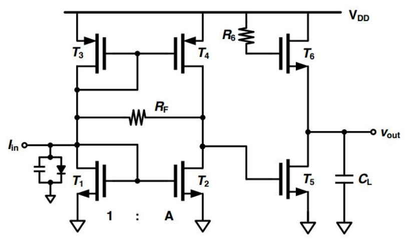
Figure 4. Training rewards and metrics over RL iterations. Left shows maximum GBW versus iterations, showing the RL can learn quickly to become better than the human design. Right shows reward averaged over iteration. In the beginning there is a peak in rewards due to the random exploration, but steadily increases from training of the AI-agent.

B. Case study 2: two-stage trans-impedance amplifier with self-bias resistor

A second circuit, namely a trans-impedance amplifier, was also optimized using RL. The range was chosen to be larger, namely 300% of the Gm-over-Id parameters, because an additional "transfer-learning" experiment was conducted. Namely, a model was trained after 2000 iterations on 160nm technology, then was used to train on 180nm.

The hyper-parameters consists of both real transistor sizing such as W1, W2 as well as abstract design parameters such as the A in the figure.

	BW (GHz)	Gain ($\times10^2\Omega$)	Power (mW)	Noise (pA/$\sqrt{\text{Hz}}$)	Peaking (dB)	GBW (THz$\times\Omega$)	FoM
Spec [25]	max	>7.58	<18.0	<19.3	<1.00	max	-



(a) Two-Stage Transimpedance Amplifier

Figure 5. shows the high-level schematic of the trans-impedance amplifier to optimize for. It is the same one used in the work from [1] for high frequency optical applications (input is a diode current).

Figure 6 shows a few variations of the RL algorithm and the corresponding training rewards versus iteration, and design space explored. A few different combination of RL and optimization strategies were deployed. Blue is randomly searching in the search space, orange is pre-loading with an already trained model on 160nm technology, green is with no preloading and 30 iterations of random search, and red is with 100 iterations of initial random search and then using RL. It is clear that the although the rewards are about the same, the time the peaks occur are different. Pre-loading peaks (orange curve) occur earlier than the green non-preloading AI peak, and also is higher than the red which does more random search in the beginning. It is interesting that random searching can help with convergence and finding unexpected designs but can also lead to many "bad" designs, suggesting why the reward actually dips into the negative region for the red curve from iteration 1000 and beyond.

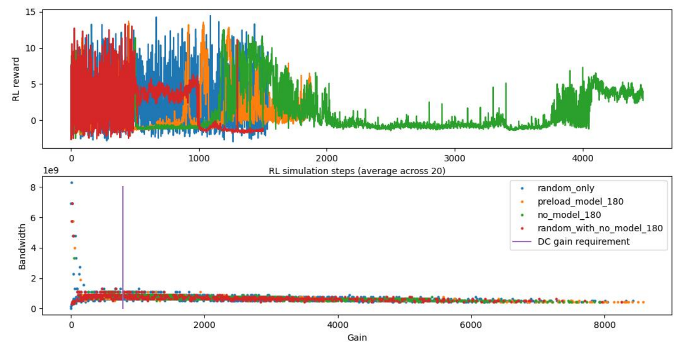


Figure 6. Training and ultimate design space explored by different methods

The effect of pre-loading an AI is clearer in figure 7. For high-gain designs, pre-loading yields many designs, compared with not pre-loading. Also random searching has some usefulness in finding some designs, but this approach is unable to hone in to these designs and fine-tune, unlike the RL methods.

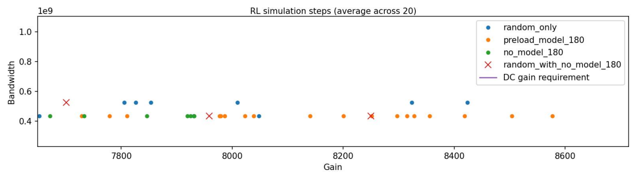


Figure 7. Zoomed-in high-gain designs searched by different optimization and RL methods. Clearly, pre-loading a model can yield designs that non-preloading and purely random search cannot find, showing the impact of transfer-learning or re-using AI models.

Finally, the RL approach is compared with paper results from the work of [1]. Not all the parameters are exactly the same due to different PDK transistors and parameters such as voltage supply and output capacitance (the work [1] did not explicitly indicate these), but the node length is kept the same at 180nm for comparison purposes.

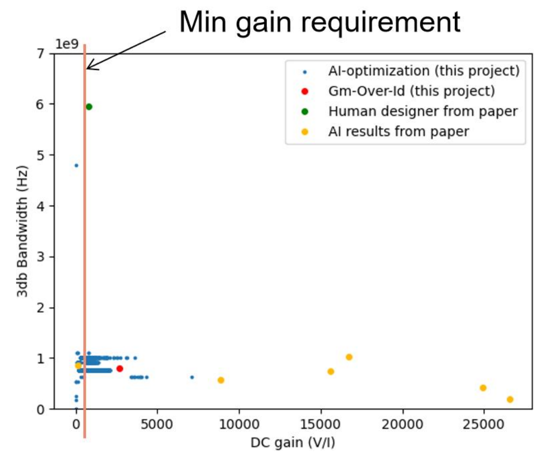


Figure 8. Design space exploration by RL, human Gm-over-id, and results from the paper [1].

	Gain (Ω)	BW (GHz)	GBW	Power (mW)
Human [1]	768	**5.95**	4.57	8.11
Gm over id	1133	0.912	1.03	**0.30**
ES [1]	10400	1.03	12.3	4.25
BO [1]	12300	0.16	1.99	2.76
GCN-RL [1]	**16700**	1.03	**17.2**	3.44
This work 1	7128	0.63	4.497	0.72
This work 2	3605	1.0	3.60	0.63

Figure 9. Table comparing this work against prior work. There is a clear trade-off of power for bandwidth and gain for this work. Highlighted yellow are from this work. Bolded are maximum values in the column.

From figure 8, it can be seen that generally, this work's RL solutions lie close with the paper's results and achieve balanced results with good gain-bandwidth product. The reason why there are some discrepancies include not knowing the set-up of the work [1] in detail, and trading off power for (this work has low power designs) bandwidth and gain as shown in figure 9. In short, a simple trained AI on the hyper-parameters is able to effectively optimize and reach a set of designs with good metrics satisfying the design constraints.

V. Conclusion

An AI-based optimization RL algorithm was constructed around an analog simulator Spectre. It was shown for a few benchmark amplifier circuits that RL could be used as the optimization algorithm, achieving results better than both manual and random design approaches. It was shown that using a trained RL model on a new technology node can yield better results faster, suggesting that the AI "learned" some design strategy for the given circuit. The approach is general and those not need to be built around circuit physical parameters such as lengths and RLC values. Future work of the approach would involve trying to optimize more complex circuits with no explicit analytical models, such as PLLs or sigma-delta ADCs, as well as trying to see if the AI-model can transfer its learning to unseen topologies. This may involve having the hyper-parameter space being over-defined, or using other kinds of AI agents to train which are more general and indifferent to input dimensions, such as convolutions neural networks. In short, RL has the great potential to compete with existing look-up table, genetic algorithm and other optimization approaches to analog automated design.

Aknowledgements

I want to thank Professor Reza Molavi for the excellent course in Advanced Analog Design ELEC501 , where I learned a lot of new techniques and interesting approaches to designing analog circuits. I also enjoyed the discussions, industry designs, quantum computing material and guest lectures.

References

[1] Wang, H., Wang, K. et Al.: GCN-RL circuit designer: transferable transistor sizing with graph neural networks and reinforcement learning. In: ACM/IEEE Design Automation Conference (DAC) (2020)

[2] Budak, A.F., Zhang, S. et Al. (2022). Machine Learning for Analog Circuit Sizing. In: Ren, H., Hu, J. (eds) Machine Learning Applications in Electronic Design Automation. Springer

[3] Youseff. A. A., Murmann B., Omran H., Analog IC Design using Precomputed Lookup Tables. IEEE Access (2022)

[4] Bakr Hesham, Hasaneen E-S., Hamed H. F. A., Design Procedure for Two-Stage CMOS OpAmp using gm/ID design Methodology in 16nm FinFET Technology, IEEE (2019)

[5] Omran H., Amer M. H., Mansour A. M., Systematic Design of Bandgap Voltage Reference Using Precomputed Lookup Tables (2019)

[6] Schreier, R. An empirical study of high-order single-bit delta-sigma modulators. IEEE Transactions on Circuits and Systems (1993)