

Strand displacement neural networks and RNA translators

Daniel Svane Nielsen

June 11, 2017

Abstract

Empty

Contents

1	Introduction	2
2	Theory	3
2.1	Strand displacement	3
2.2	Neural networks	6
2.2.1	In silico neural networks	6
2.2.2	In vitro perceptron	7
2.2.3	Input translation	12
3	Method	14
3.1	Perceptron simulation	14
3.2	Translation	16
3.2.1	Design	16
3.2.2	Transcription	17
4	Results	23
4.1	Seesaw perceptron	23
4.1.1	2-input AND	24
4.1.2	2-input OR	25
4.1.3	3-input AND	26
4.1.4	3-input 1-OR	27
4.1.5	3-input 2-OR	28
4.2	Translation	29
5	Conclusion	34
A	Appendix	37

1. Introduction

Logic circuits using DNA and RNA has many interesting applications in diagnostics and treatment. An example is cancer detection, where miRNA's can be used as biomarkers [1]. These biomarkers can be used as inputs for logic circuits, which can be designed to activate fluorescence signals [2] or enzymes [3] when combinations of biomarkers are present or absent. For example, a circuit could be designed to activate when 2 unique miRNA's are present at the same time, or when a certain protein is present.

A different approach to building logic circuits is using neural networks. One of the advantages of this approach, is that large-scale networks does not have to be designed by someone with knowledge of logic gates and circuitry. The inputs of the circuit simply have to be defined along with the required output, and the neural network can be trained to give the desired functionality.

The neural network approach to strand displacement circuits has already been developed [4]. The design is based on the seesaw gate motif, which requires that the inputs to the network must have very specific sequences. To use custom input sequences, like the miRNAs from cancer cells, the sequences have to be translated. This has also already been done, using two linked strand displacement reactions [5].

This project is split into two parts.

The first part aims to lay out the groundwork for a software where the miRNAs one wishes to detect can be entered. The strands and concentrations required for a neural network that implements a desired truth table is then calculated, and could be readily ordered and used as a biosensor.

The second part aims to translate one RNA sequence into another RNA sequence, and measuring the outcome by a fluorescent reporter. This is almost identical to the experiment in [5], but using RNA instead of DNA in the translation reactions. This is mostly done to show that the translators could also use RNA, but also to get some experience in a laboratory.

2. Theory

2.1 Strand displacement

Strand displacement can be used to implement DNA devices which can be used in molecular computing [6]. It uses the predictability of Watson-Crick base pairing (A pairs with T, C pairs with G) of the nucleotides of DNA to design reactions. When the nucleotides of two strands of DNA base pair with each other, the complex will have a certain free energy. By introducing another strand which will pair stronger in the complex (give a lower free energy), the original strand can be displaced, as seen in Figure 2.1.

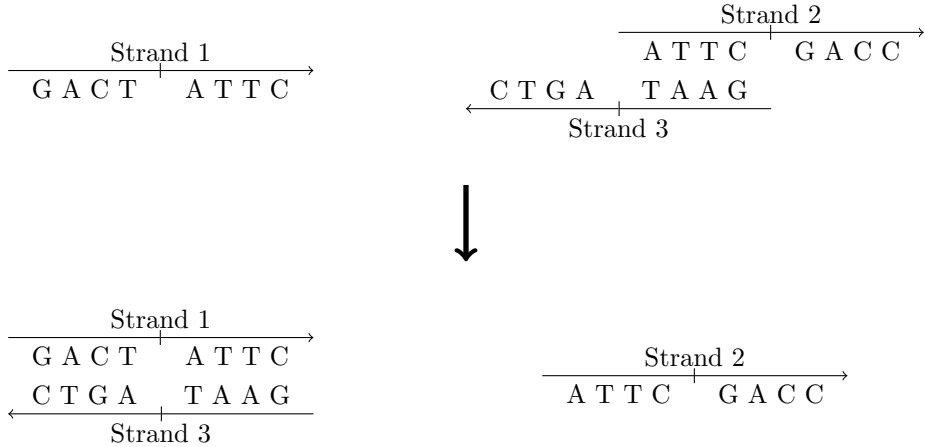


Figure 2.1: Strand displacement reaction where strand 1 base pairs at a lower free energy with strand 2, and displaces strand 3. The arrow denotes the strands 3' end.

The reaction in Figure 2.1 can be simplified by leaving out the specific bases, and instead naming each base pairing region. The same reaction as in Figure 2.1 can be seen in its simplified version in Figure 2.2.

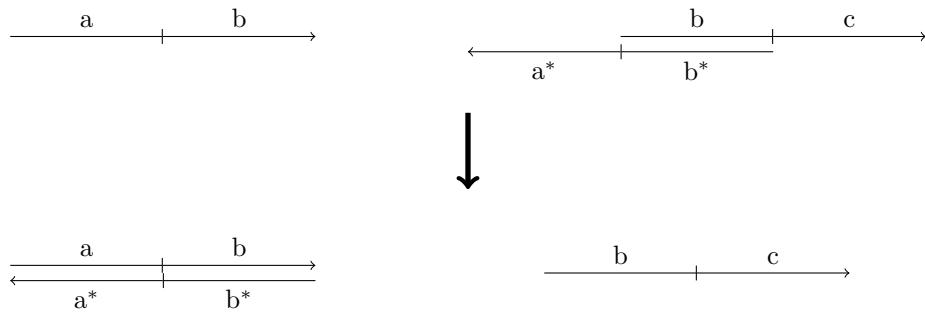


Figure 2.2: Simplified view of the reaction from Figure 2.1. The sequence of nucleotides is replaced with a domain name. The asterix on the sequence name denotes that it is the reverse complement of the normal sequence (a^* is the reverse complement of a).

The result of the reaction in Figure 2.2, is that the strand bc is only single-stranded when displaced by the strand ab . The reaction can then be seen as a YES gate with input ab and output bc . Why this is useful in molecular computing is not immediately apparent, but the strand displacement technique can be extended to larger reactions. A more interesting application is the AND gate seen in Figure 2.3.

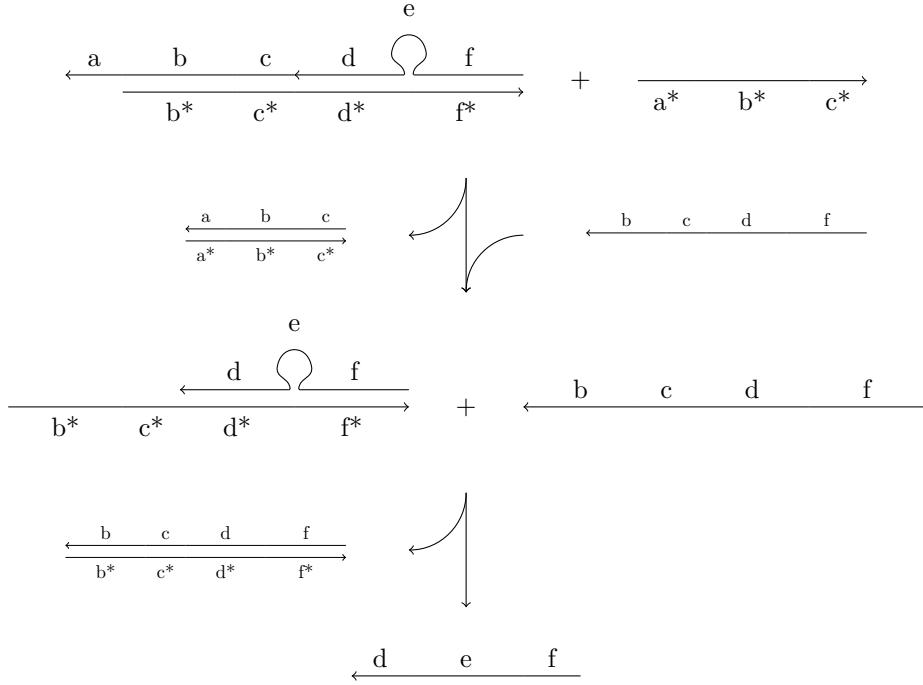


Figure 2.3: An AND gate made with strand displacement. After two strand displacements by the input strands $a^*b^*c^*$ and bcd , the output strand def is released and can react in further strand displacement reactions with other logic gates, or activate enzymes and fluorescence signals. Without either of the input strands, the output will not be displaced. Adapted from [6].

The actual concentrations of each of the strand species when reacting, can be predicted by the free energy of each complex [7], but the time it takes for the reaction to take place is harder to predict. It has been shown that the reaction speed is exponentially dependent on the toehold length for short toeholds [7]. A toehold, for example, is the single stranded region of the strand 2 + strand 3 complex in Figure 2.1. To do time analysis of larger strand displacement networks, software like Visual DSD [8] can be used.

2.2 Neural networks

2.2.1 In silico neural networks

Artificial neural networks (referred to just as neural networks) is a software implementation of the connection of neurons in the brain. In computer science they have been used to solve a wide variety of problems, like character and facial recognition. They present an advantage over conventional programmatic methods, as they don't need explicit coding for each new problem [9]. The simplest instance of neural networks is the single neuron, also known as the perceptron [10]. The perceptron can be used for simple decision making (giving yes/no answers) from a set of inputs. The logical 2-input AND operator is an example of a problem that can be solved with the perceptron [11], where the perceptron can output 1 if both the inputs are 1, otherwise 0.

The perceptron works by taking all the inputs, and multiplying each input by its associated weight. If the sum of the multiplied inputs exceeds a certain threshold, the perceptron will output 1, otherwise 0. This is shown schematically in Figure 2.4.

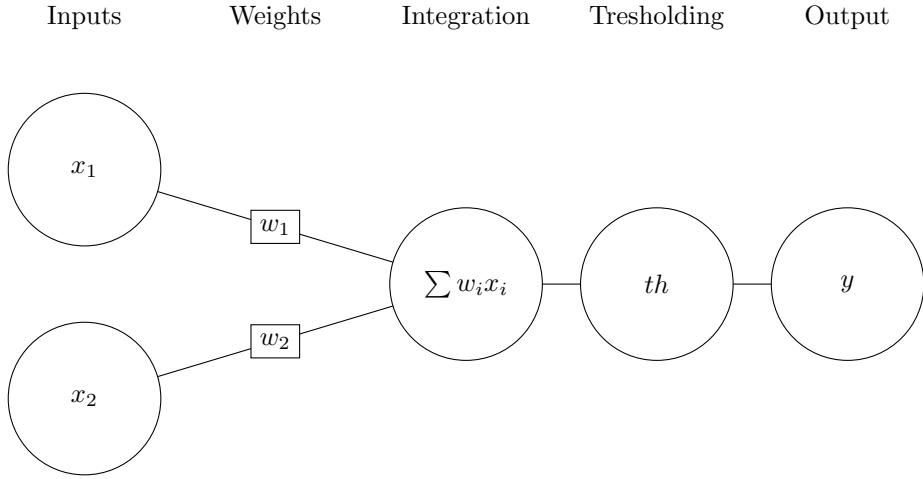


Figure 2.4: Diagram of the artificial perceptron.

The representation can be simplified to the type shown in Figure 2.5. The weights and thresholds for the perceptron version of the 2-input AND gate can also be seen in Figure 2.5.

The perceptron is limited to classifying inputs that are linearly separable [11], which rules out the XOR operation.

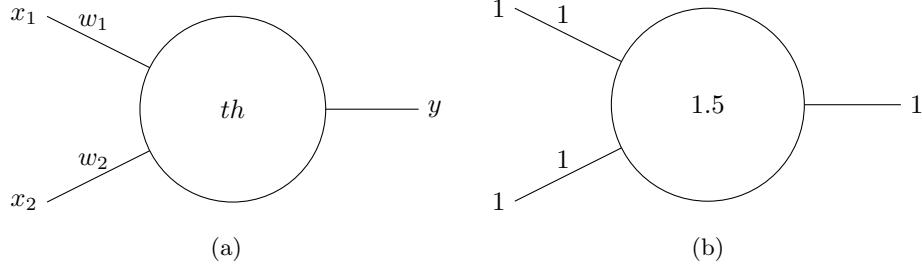


Figure 2.5: (a) Simple representation of a 2-input perceptron. (b) Weights and thresholds on a perceptron that implements the 2-input AND gate.

2.2.2 In vitro perceptron

It has previously been shown that the function of the artificial neuron can also be implemented using strand displacement reactions. The system is based on the seesaw gate motif [12], and can fulfill most of the functionality of a real neuron [4]. The seesaw gate is a catalytic gate with a threshold, designed for use in scalable strand displacement circuits. The seesaw gate uses a toehold to accelerate reactions, and has a left and right recognition domain to connect to other seesaw gates. The seesaw is named after the back and forth reaction of the strand displacement reactions, seen in Figure 2.6.

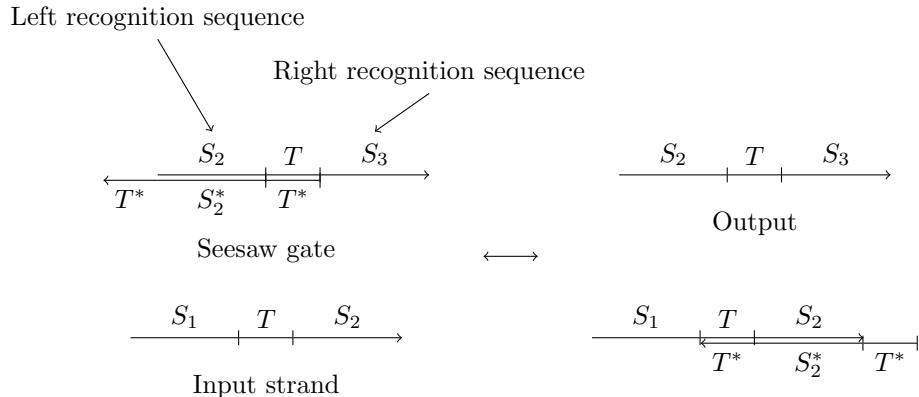


Figure 2.6: The back on forth reactions of the seesaw gate.

The reaction can be pushed to the right by using a fuel strand, or the input can be reduced by using a threshold gate, the details of which are discussed below.

Thresholding

In the natural neuron, the neuron will activate when its inputs exceeds a threshold. This is implemented using a threshold gate which will bind the input and prevent it from reacting downstream in the network. If the threshold gate concentration is higher than the input concentration, the input will be suppressed by the threshold. If the threshold gate concentration is lower than the input concentration, not all of the input is suppressed, and will be able to react further downstream in the network. This reaction is shown in Figure 2.7a.

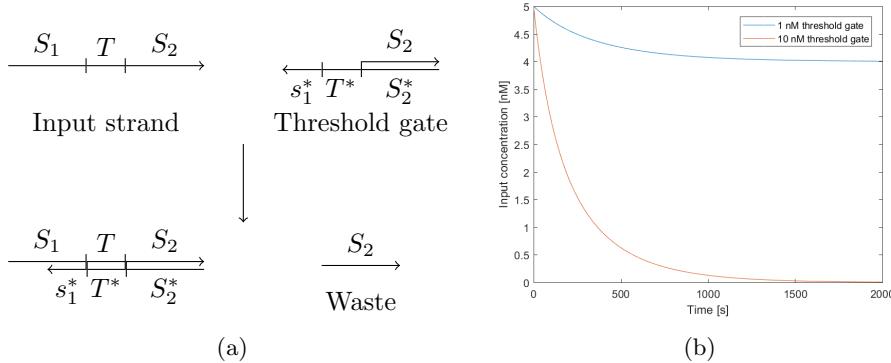


Figure 2.7: (a) Reaction of an input strand with a threshold gate. The product has no free toehold domain, and can't undergo reverse reaction. The waste has no toehold, and can't participate in further reactions. (b) Time analysis of the concentration of input strand (5 nM start concentration). A threshold concentration higher than the input (red) will bind all input strand. A lower concentration (blue), will allow the input strand to participate in further displacement reactions.

Integration

If the seesaw neuron have multiple inputs, they will have to be "collected" before thresholding, as they don't have the same left recognition sequence (see Figure 2.6). This is done through an integrating gate, which will collect all inputs with the same right recognition sequence, and release a common signal which can be thresholded. The integration gate is seen in Figure 2.8a.

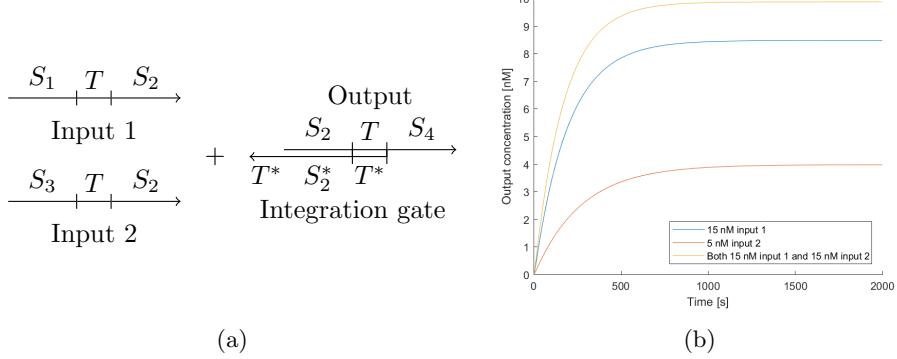


Figure 2.8: (a) Reaction of 2 input strands with an integration gate. The input strands have the same right recognition sequence S_2 , and will both displace the top strand of the integration gate, releasing the output. (b) Time analysis of the concentration of the output strand. The concentration of the integration gate is 20 nM. The first input of 15 nM increases the output strand concentration, compared to the second input of 5 nM. When both strands are added the concentration of output increases further.

Weighting

The inputs to the gate are weighted using their concentration. By making the integration gates concentration the sum of all the input concentrations, a high concentration of one input strand will contribute more to the activation than that of a low concentration, as shown in Figure 2.8b.

The weight is decided by the concentration of output of an input gate and its fuel strand (see Figure 2.10). The fuel serves the purpose of pushing the output of one gate to its target concentration, shown in Figure 2.9.

A problem with this approach to weighting, is that the inputs can only contribute positively to the sum. In silico neural networks can use negative weights to simulate inhibitory synaptic connections, and is needed to implement many kind of boolean functions [11]. The in vitro network can't have negative concentrations of input sequences, so other approaches have to be considered. The problem can be solved using dual-rail logic [4], where each input is replaced by two inputs. The details of the dual-rail logic circuits is not a part of this project, and thus the networks will be limited to simple AND and OR circuits, since the NOT operation requires negative weights, and the XOR operation is not linearly separable [11].

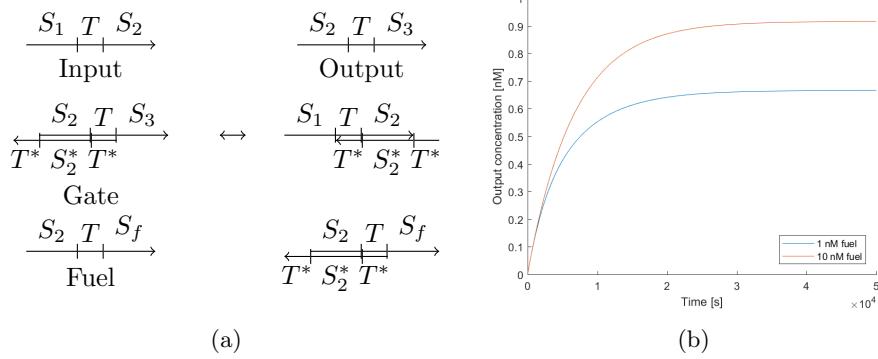


Figure 2.9: (a) Reaction of an input strand with a gate and fuel. The input displaces the output strand from the gate, and the fuel blocks the reverse reaction, pushing the equilibrium towards the free output. (b) Time analysis of the concentration of the output strand. Input and gate concentration is initially 1 nM. When the fuel concentration is low (blue), the output concentration doesn't reach the initial gate concentration. When the fuel concentration is high (red), the output concentration is pushed closer towards its maximum concentration.

Neuron

By combining the discussed functional elements of the seesaw gate, a "neuron" can be created using strand displacement reactions, as depicted in Figure 2.10.

Instead of displaying all strands of the seesaw circuit, each gate can be represented by a simplified schematic. The first input gate from Figure 2.10 is shown in its simple version in Figure 2.11.

Using the schematic representation, the seesaw neuron from Figure 2.10 can be simplified to the schematic in Figure 2.12.

The recommendations from the original paper [4] is that each fuel strands concentrations is twice that of the gate concentration. The threshold concentration on the input gates (as shown in Figure 2.12) allows for concentrations of input strand less than 0.2 (relative concentration) to still be detected as 0.

The representation from Figure 2.12 can further be simplified to the original perceptron shown in Figure 2.4. The seesaw neuron thus successfully implements all the required functionality of the artificial neuron, except for negative weights.

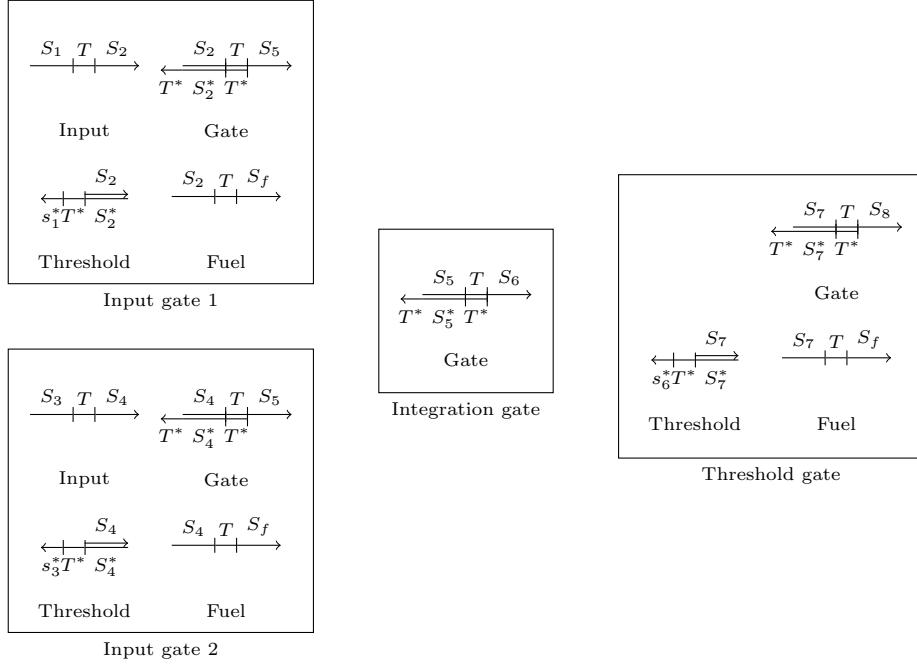


Figure 2.10: Schematic of a 2-input neuron implemented with seesaw gates. The neuron has an input gate for each input, which makes sure that the input concentration is higher than a given threshold before the input is registered. The gate and fuel concentrations in each input gate affects the weight of the input before it is sent to the integration gate. The integration gate collects the right recognition sequence from the input gates for the threshold gate. The threshold gate activates if the sum of the weighted inputs is larger than the threshold concentration of the threshold gate.

Training

The training algorithm typically used for digital perceptrons is based on the ability of the neurons to have negative weights and thresholds [13]. A modified algorithm is presented in the original paper [4], which works for dual rail circuits. For the circuits in this report without dual rail logic, the algorithm can be seen in Code 2.1.

The algorithm in Code 2.1 will keep increasing the weights until the output of the network matches the desired output. The input sets are taken from the truth table that the circuit should realize. If the circuit for example should realize the simple AND gate (see Table 2.1), the network is activated with the input rows from the truth table, and tested whether the output equals (or is very close) to the output in the truth table.

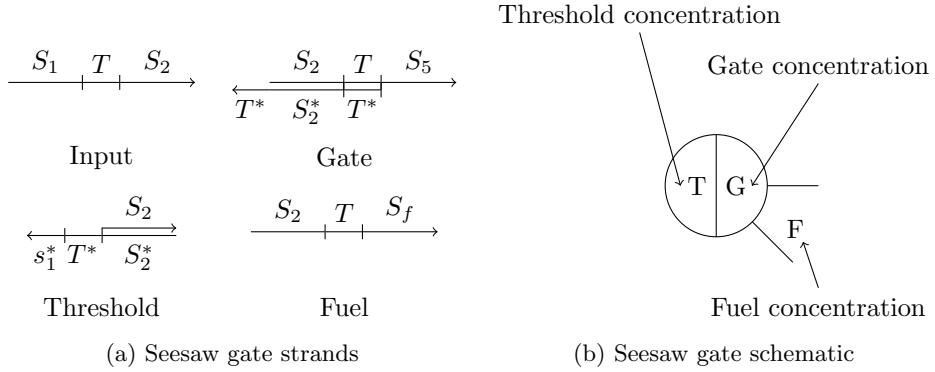


Figure 2.11: The strands and concentrations of the units of a seesaw gate can be represented to simplify larger diagrams.

```

initialize all weights to 0 and threshold to 10
while training is incomplete
    for all the input sets
        find the output of the network from the input set
        if output is not correct
            mark training as incomplete
    if training is marked as incomplete
        increase the weights

```

Code 2.1: Pseudocode for the seesaw perceptron training algorithm

2.2.3 Input translation

As per design of the seesaw gate, the input sequences for the strand displacement neural network needs very specific left-recognition, right-recognition and toehold sequences. To detect sequences that does not have these elements, the sequences have to be translated. It has previously been demonstrated that miRNAs can be translated to other sequences [5], by using two half-translators which are basic strand displacement reactions. An example of input translation can be seen in Figure 2.13.

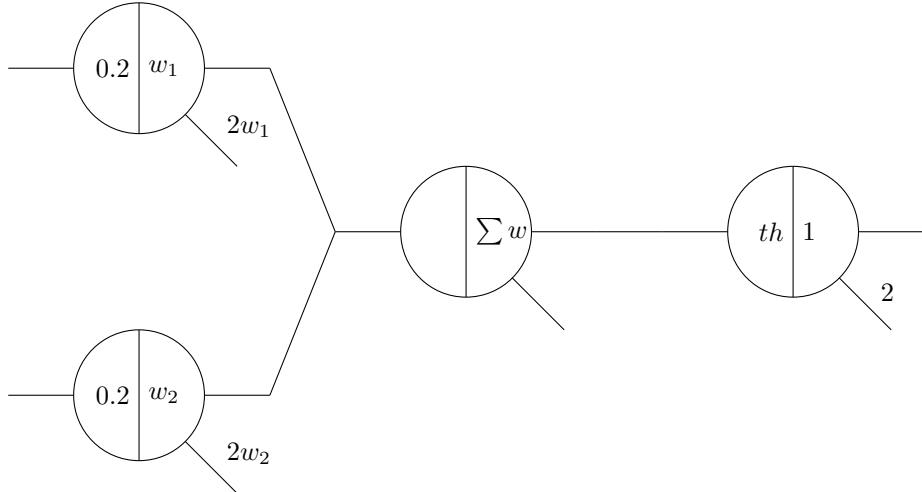


Figure 2.12: The seesaw neuron from Figure 2.10 shown schematically.

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1: Truth table for an AND gate.

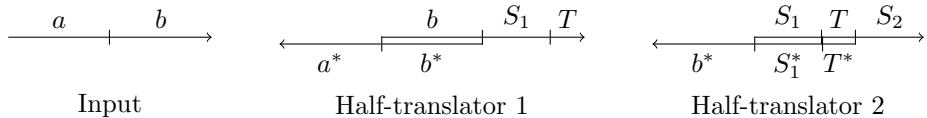


Figure 2.13: Translation of arbitrary input sequences into the syntax required for the seesaw neural network. The input strand ab displaces bS_1T from the first half-translator. bS_1T can then displace S_1TS_2 from the second half translator. This process successfully translates the input ab into S_1TS_2 , which can then be used for input in a neural network (see Figure 2.10).

3. Method

3.1 Perceptron simulation

To simulate a perceptron with strand displacement reactions, the Visual DSD software was used to implement each of the subunits of a seesaw neuron discussed in the theory section. The syntax of the four subunits required to create a perceptron can be seen in Code 3.1 (refer to the Visual DSD manual for the syntax [14]).

Each of the subunits can be called with the desired left and right recognition sequences, when combining them for larger units. Note that each of the recognition sequences (S1 for example) is split into three (S1L, S1, S1R). This is needed to implement the threshold gate, which has a short part of the left recognition sequence (Figure 2.7a).

The actual perceptron can be pieced together by combining the subunits from Code 3.1. The 2-input AND using seesaw gates seen in Figure 2.10, can be written in Visual DSD as seen in Code 3.2.

To abstract from the Visual DSD syntax, a transpiler was created in Javascript which can generate the needed Visual DSD code for perceptrons of variable input size. The transpiler can take a truth table defined as an array, and generates the needed Visual DSD code to create a perceptron of required input size.

```
def signal(N, S1L, S1, S1R, S2L, S2, S2R) = N * <S1L^ S1  
      S1R^ T^ S2L^ S2 S2R^>  
def threshold(N, S1R, S2L, S2, S2R) = N * {S1R^* T^*}[S2L^  
      S2 S2R^]  
def gate(N, S2L, S2, S2R, S3L, S3, S3R) = N * {T^*}[S2L^  
      S2 S2R^ T^]<S3L^ S3 S3R^>  
def fuel(N, S1L, S1, S1R) = N * <S1L^ S1 S1R^ T^ Sf>
```

Code 3.1: Visual DSD definition of the seesaw gate subunits.

```

(
(* First input gate *)
signal(0.9, S1L, S1, S1R, S2L, S2, S2R) |
threshold(0.2, S1R, S2L, S2, S2R) |
gate(1.0, S2L, S2, S2R, S3L, S3, S3R) |
fuel(2, S2L, S2, S2R) |
(* Second input gate*)
signal(0.9, S4L, S4, S4R, S5L, S5, S5R) |
threshold(0.2, S4R, S5L, S5, S5R) |
gate(1.0, S5L, S5, S5R, S3L, S3, S3R) |
fuel(2, S5L, S5, S5R) |
(* Summation gate *)
gate(2, S3L, S3, S3R, S6L, S6, S6R) |
(* Thresholding gate *)
threshold(1.5, S3R, S6L, S6, S6R) |
gate(1.0, S6L, S6, S6R, S7L, S7, S7R) |
fuel(2, S6L, S6, S6R)
)

```

Code 3.2: Code for a seesaw neuron in Visual DSD

To train the perceptron, the initial idea was to simply train a software perceptron to a given truth table, and use the found weights and thresholds in the seesaw perceptron. This idea proved too naive, as the weights and thresholds from a software perceptron did not produce the same outputs in a seesaw perceptron. The solution was to generate the Visual DSD code, run it through the command-line version of the Visual DSD software, and simulate the output concentration of the perceptron. This output could then be compared to the expected output from the truthtable, and the weights adjusted using the algorithm discussed in the theory section.

It was not possible to create a web application with a user friendly interface for designing the perceptron, as was done with the software training [15]. The command-line version of the Visual DSD software preferably needs to be run on a Windows machine, for which no free hosting could be found in time. The code is still available on Github for running locally on Windows machines [16].

The initial idea was to get the sequences used in reactions directly from the Visual DSD simulation. A limitation of the command-line version of Visual DSD however, is that it can not output the sequences. It should still be possible to circumvent this, by using a pregenerated list of sequences to pick from, like it was done in [17].

To translate the sequences from Visual DSD into the desired sequences, Nupack could be used for the design. The command-line version of Nupack needs to be run on Linux, and since the Visual DSD code needs to be run on Windows, no solution was found in time for this project.

3.2 Translation

3.2.1 Design

The translator was designed to output the sequence CUAGACUGAAGCUC-CUUGAGG, which is the sequence that can activate a FRET tile previously designed in the group. The input was not specifically chosen, but generated by Nupack to fulfill the design parameters. Two versions of the translator were designed, one where the half-translators annealed with 10 bp (Code A.1) referred to as "short translator", and one where they annealed with 20 bp (Code A.2) referred to as "long translator". The long translator was closer to the setup of the original DNA translator [5], which had annealing lengths of 23-25 bp.

When transcribing the RNA using the T7 RNA polymerase, the sequences are going to start with GG [18], so these were included in the design. Repeats of identical nucleotides were also prevented. The result of the Nupack designed RNA sequences can be seen in Table 3.2 (refer to Figure 3.1 and Figure 3.2 for strand names).

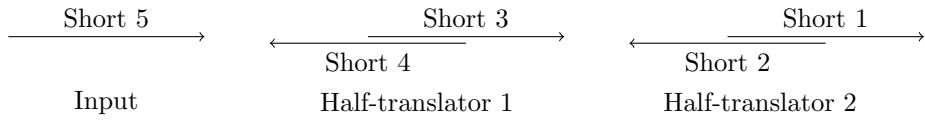


Figure 3.1: Sequence naming of the short translator sequences (see table 3.2).

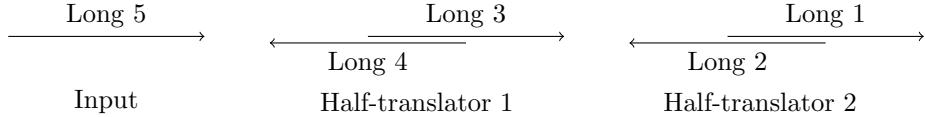


Figure 3.2: Sequence naming of the long translator sequences (see table 3.2).

To transcribe the RNA sequences, the RNA was converted to DNA, and the T7 promoter was added to the strands. The reverse complement of the strands was found, to serve as the template strand. The T7 RNA polymerase does not need the full template to be double-stranded, only the promoter sequence [18], so the resulting template will be the DNA version of the reverse complement of the target RNA (plus the reverse complement of the promoter), annealed with the promoter sequence (depicted in Figure 3.3). A program was written to convert the target RNA sequences to their DNA template directly on the output from Nupack, and format it for batch IDT ordering [19]. The results of the RNA-to-DNA-template conversion can be seen in Table 3.1, which are the final oligoes ordered from IDT.

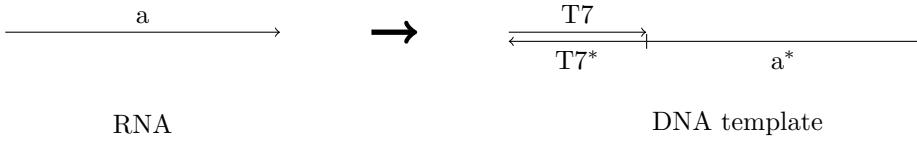


Figure 3.3: DNA template from RNA sequence for the T7 RNA polymerase.

Name	Short	Sequence	Length
T7 promoter	0	GTTAATACGACTCACTATAG	20
Short 1	1	CCTCAAGGAGCTTCAGTCTAGCCCTATAGTGAGTCGTATTACC	43
Short 2	2	CTCCCTGAGGCACATAACTCCCTATAGTGAGTCGTATTACC	42
Short 3	3	CACATAACTCTACTAAATCTCCCTATAGTGAGTCGTATTACC	42
Short 4	4	GAGTTATGTGCCCTCAAGGAGCCTATAGTGAGTCGTATTACC	42
Short 5	5	AGATTTAGTAGGTTATGTCCTATAGTGAGTCGTATTACC	42
Long 1	6	GTCAATTGCCCTCAAGGAGCTTCAGTCTAGCCCTATAGTGAGTCGTATTACC	52
Long 2	7	GCTCCCTGAGGCGAATTGACCCATCTTCAATTCTACTCCCTACCCCTATAGTGAGTCGTATTACC	62
Long 3	8	CCATCTTCAATTCTACTCCCTACCCCTACCCCTATAGTGAGTCGTATTACC	52
Long 4	9	TAGGAGTAGAATGAAGATGGGTCATTGCCCTCAAGGAGCCCCATAGTGAGTCGTATTACC	62
Long 5	10	GATTGAGGTATAGGAGTAGAATGAAGATGGCCTATAGTGAGTCGTATTACC	52
Beacon fluorophore	11	CGCTAGACTGAA	13
Beacon quencher	12	CCTCAAGGAGCTTCAGTCTAGCCG	24

Table 3.1: Sequences and names of the DNA strands used for transcription.

3.2.2 Transcription

The oligos from IDT were dissolved in TE buffer (Table A.1) to an approximate concentration of 120 μM , based on the quantity of substance written on the tubes. The final concentration desired was 100 μM , but due to a risk of inaccurate substance quantities, the dissolved concentration was chosen to be slightly above. The oligos could then be further diluted after measuring their absorbance on the Nanodrop.

After dissolving the oligos, the absorbance of each sample was measured on the Nanodrop in triplets. A program was written which can take the .csv output of the Nanodrop, and calculate the concentration based on each strands extinction coefficient [20]. The measured concentrations (Figure A.2) was used to dilute the samples further, to a concentration of 100 μM .

Name	Short	Sequence	Length
Short 1	1	GGCUAGACUGAAGCUCCUUGAGG	23
Short 2	2	GGGAGUUUAUGUGCCUAAAGGAG	22
Short 3	3	GGAGAUUUAGUAGAGUU AUGUG	22
Short 4	4	GGCUCCUUGAGGCACAUAAUCU	22
Short 5	5	GGCACAUAAACUCUACUAAAUCU	22
Long 1	6	GGCUAGACUGAAGCUCCUUGAGGCGAAUUGAC	32
Long 2	7	GGUAGGAGUAGAAUGAAGAGGGCUAAUUCGCCUAGGAGC	42
Long 3	8	GGGAUUGAGGUAAUGGAGUAGAAUAGAAGAUGG	32
Long 4	9	GGGCUCCUUGAGGCAGAUUGACCCAUCUCAUUCUACUCCUA	42
Long 5	10	GGCCAUCUCAUUCUACUCCUAUACCUCAAUC	32

Table 3.2: Sequences and names of the transcribed RNA strands.

	Initial conc.	Final conc.	Volume
Transcription buffer	10X	1X	10 μ L
DTT	100 mM	10 mM	10 μ L
NTP mix	25 mM	2.5 mM	10 μ L
Template	500 nM	50 nM	10 μ L
T7 RNA polymerase			1 μ L
Nuclease-free water			59 μ L
Total			100 μ L

Table 3.3: Mixing of compounds for the first transcription done on the templates for both the short and long translator.

To anneal the templates to the promoter, each of the template strands were mixed with equal amounts of promoter strand in annealing buffer (Table A.2), to a final concentration of 1 μ M. The mixed samples were then heated to 90° C for 5 minutes, and left to cool down to room temperature.

To check if samples annealed properly, they were run on a 20% native PAGE gel for 3 hours. Each lane was loaded with 50 μ l sample, and 10 μ l native loading buffer (Table A.3). Afterwards the gel was stained in SYBR Gold, and visualised on the Typhoon scanner. The result of the scan can be seen in Figure 4.6a.

As can be seen in Figure 4.6a, the darkest bands are the annealed samples. The samples for the short translator runs at about the same size, while there is bigger variation in the long translator samples, as expected based on Table 3.1. The exact positions of the long translator samples does not match with the sequence length, though this can be explained by secondary structures of the single-stranded part of the sample. The shorter bands visible below, are probably excess promoter, other secondary structures, and shorter sequences from synthesis errors. No lane with ladder was run, so the exact position of the bands can't be commented on.

To get the desired RNA sequences, a transcription reaction was run with each of the annealed DNA templates, according to Table 3.3. The samples were left overnight at 37°C. The day after, 1 μ L of RNase-free DNase was added to the samples, and heated for 37°C for an hour. Afterwards, 100 μ L of denaturing loading buffer was added to each sample, and 10 μ L of the DNA 0 and DNA 5 strands (as controls) were mixed with 10 μ L of denaturing loading buffer, and heated for 5 minutes at 90°C.

To purify the RNA, the transcribed sequences and controls were run on a 20% denaturing PAGE gel, for 4 hours at 20 W. The RNA product should be visible in UV shadowing, but no product was seen. The gel was then stained with SYBR Gold and scanned on the Typhoon. As can be seen in Figure 4.7a, the gel wasn't stained long enough, so after restaining it in SYBR Gold, it was scanned again.

The lanes 1-10 in Figure 4.7b does not show distinct bands. There seems to be more product from the long translator transcriptions (lanes 6-10), than in the short ones (lanes 1-5). Even the controls (lanes 11 and 12) which were loaded in equal amounts does not show up in equal strength. Since the gel from the DNA annealing was run without any controls, it was difficult to see if there was any errors in the annealing. Another 20% native PAGE gel was run with the annealed DNA, using the single stranded oligos as control (Figure 4.6b). To simplify the experiment while trying to find the error, only the short translator sequences were used.

The results of Figure 4.6b still shows that the templates have annealed with the promoter, and runs as about 40-50 bp. The expected size is around 60 bp if the template was fully double-stranded (sum of promoter and template), but it is difficult to say how a partly annealed structure will run on a native gel. The new gel does however show that the bands in the annealed lanes below the assumed product, might not be excess promoter. The promoter is seen in the second lane, and lies below the bands in the annealed structure thought to be excess promoter. The bands below the product might be due to secondary structures of each of the template strands, but comparing with Figure A.4, the band in the 5+0 lane would be expected to be less visible, as strand 5 has no secondary structure.

Despite the unexplained bands from the annealing, a new transcription was run on the short template strands to see if better results could be obtained. The transcription was done using the previous protocol, and run on a gel. The UV shadowing still did not show any visible product, so the gel was scanned to check for bands.

As seen in Figure 4.8a, still no clear bands with RNA product was visible. After checking all buffers and materials, it was learned that the T7 polymerase that was used was from 2013, and might have been too old to still function properly. A fresh T7 polymerase was used for a new transcription, same as the previous protocol. The UV shadowing did show some distinct bands this time, so the bands were cut out for purification. The purification was done using the precipitation protocol [21]. After purification the pellets were dissolved in 100 μ L TE buffer, their absorbance measured on the Nanodrop, and their concentration calculated using their extinction coefficients.

To check that the purification worked, 2 μ L of each sample with 1 μ L of denaturing loading buffer was run on a 20% denaturing gel, stained with SYBR Gold, and visualised on the Typhoon.

Figure 4.8b shows that the RNA has been isolated, but is not very pure. It is possible that when cutting out the RNA from the gel before purification, too large an area was taken. The target RNA sequences were still expected to be in the purified samples (albeit not very pure), and it has previously been shown that strand displacement reactions can work with unpurified components [17], so the experiment continued with the current RNA.

Based on the concentrations in Figure A.3, the samples were further diluted in RNase-free water to 2 μ M for the translator strands (1-4) and 1 μ M for the input strand (5). Then 25 μ L of each of the translator strands were mixed with their respective counterpart (1+2 and 3+4) to a final concentration of 1 μ M in 50 μ L.

At this point, the reporter samples were also mixed and diluted in RNase-free water to the same concentration as the translator subunits. To each of the subunits and the reporter, 5 μ L of 10X annealing buffer was added. The samples were heated at 90°C for 5 minutes, and cooled off at room temperature.

To check if the subunits and reporter annealed properly, they were run on a 20% native PAGE gel with the single strands as control. It was expected that the fluorophore would be visible at the same wavelength as SYBR Gold, so a scan without staining should show the free fluorophore (and not the quenched fluorophore in the annealed reporter).

As seen in Figure 4.9a, the fluorophore clearly shows up in the unstained scan, and gets partly quenched and moves up when annealed to its quencher. The smear in the annealed reporter might be due to synthesis errors in the quencher strand.

In the stained scan (Figure 4.9b), the annealed half-translators does not seem to have annealed at all. Comparing with the controls, the strands that was supposed to have annealed, merely looks like the sum of the single strands. There is a few reasons why this could have happened. The RNA that was purified from gel was not the correct sequences, the annealing was not carried out correctly, or the strands simply doesn't anneal very well.

To check the strength of the annealing, the strands were analyzed in Nupack. The result (Figure A.1) shows that the short translator sequences does not anneal as good as the long ones. Due to time constraints, the focus moved towards the long translator sequences, as they were expected to provide better results.

A new transcription for the long translator sequences was prepared using the previous protocol. The transcription was run for 3 hours at 37° C and then run on a 20% denaturing PAGE gel for 2 hours. No immediate product was visible, but a scan revealed that some RNA had been produced.

As seen in Figure 4.10a, some RNA was produced so the concentration of nucleotides in the reaction was increased to get enough RNA to be visible in UV shadowing. The transcription protocol was also changed to one recommended from the supplier of the T7 polymerase [22], to ensure that the polymerase was not the problem (Table 3.4). The new transcription was carried out as the previous one, and revealed a single band (strand 9) in UV shadowing, but the rest of the templates had not produced enough RNA to be purified. A scan revealed once again that RNA had been produced, but the right product could not be isolated.

	Initial conc.	Final conc.	Volume
Transcription buffer	10X	1X	4 μ L
DTT	100 mM	5 mM	2 μ L
NTP mix	25 mM	2.5 mM	4 μ L
Template	500 nM	250 nM	20 μ L
T7 RNA polymerase			4 μ L
Nuclease-free water			6 μ L
Total			40 μ L

Table 3.4: Mixing of compounds based on the transcription protocol from NEB, for the second transcription of the long translator.

	Initial conc.	Final conc.	Volume
HF	5X	1X	10 μ L
Forward primer (T7 promoter)	100 μ M	2 μ M	1 μ L
Reverse primer	100 μ M	2 μ M	1 μ L
dNTP mix	25 mM	250 nM	0.5 μ L
Template	1 μ M	10 nM	0.5 μ L
Phusion polymerase			0.5 μ L
Nuclease-free water			36 μ L
Total			50 μ L

Table 3.5: Mixing of compounds for the PCR reaction on the long translator sequences.

Some of the other group members had previously had issues with running transcriptions on templates that are only partly double-stranded with the T7 promoter sequence. To get fully double-stranded templates, some reverse primers for the template strands were ordered from IDT, and the T7 promoter could be used as the forward primer. A PCR reaction was done (Table 3.5), run on a 2% agarose SB at 300 V for 15 minutes, stained in SYBR Gold and scanned.

The result in Figure 4.11a shows the expected PCR product sizes compared with Table 3.1. Another transcription reaction was carried out on the double-stranded long translator sequences using the previous protocol. The samples were run on a 10% denaturing PAGE gel for 2 hours at 25 W. UV shadowing still only revealed a band in strand 10, so the gel was stained with SYBR Gold and scanned.

Figure 4.11b shows strand 10 clearly, but still a lot of other products, and strand 7 has hardly been transcribed at all. After some further discussion with the other group members, the results can be explained by secondary structures of the RNA strands. If the strands form strong hairpins, the transcription can be aborted early, and produce unintended products. The RNA strands were checked in Nupack for secondary structures. Comparing the Nupack analysis in Figure A.5 with the gel in Figure 4.11b, the amount of product seems to follow the free energy of the secondary structure of the RNA. Strand 10 which was visible in UV shadowing has no secondary structure with the highest probability, and strand 7 which shows the least amount of product has the lowest free energy.

The only way to avoid this problem is to create another design which doesn't have secondary structures, but since the translator sequences are locked by the desired input and output sequences, this is not a possibility. To actually get the RNA strands, they should have been ordered from IDT, synthesized and purified. This is more expensive, and was thought to be unnecessary in the beginning of the experiment. There was not enough time left to order the RNA sequences and carry out the fluorescence measurements on the reporter, so no results were obtained from the experiment.

4. Results

4.1 Seesaw perceptron

The seesaw perceptron with variable input size was created using the Visual DSD syntax through an abstraction layer written in Javascript. The software can take the truth table that should be fulfilled as an array. A seesaw perceptron with the correct number of input gates and the correct left and right recognition sequences is then automatically generated. For example, using the software with the 2-input AND gate (Table 2.1) would generate the sequences seen in Figure 2.10.

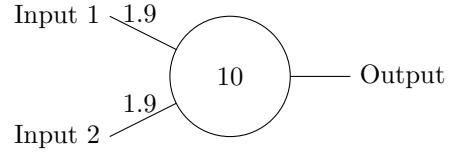
To train the perceptron, a row from the truth table is run through the Visual DSD simulation. Looking at the first row from Table 2.1, the Visual DSD code for a 2-input perceptron is generated, with input 1 and input 2 concentrations of 0 nM, a weight of 0 and threshold of 10 (see Figure 2.12). A time analysis is then done in Visual DSD of the output strand concentration. If the output concentration is equal to the output from the first row of the truth table (within a margin of error), the result is accepted, and the simulation moves on to the next row of the truth table. For the next row, new Visual DSD code is generated with input 1 concentration of 0 nM, and input 2 concentration of 1 nM. The output concentration is analyzed again. If the output is not equal to the one defined in the truth table, the weights of the perceptron is increased by 0.1, and the simulation is run for the next row of the truth table. This is repeated until a weight is found, where all rows from the truth table produces the correct output when run through the seesaw perceptron. This is summarized in Code 2.1.

To test the perceptron compiling and training, 5 different truth tables were used, ranging from 2 to 3 inputs (figs. 4.1 to 4.5). The correct output was reached after 16-23 iterations of the learning algorithm. Input sizes greater than 3 were not tested, as the training time increases exponentially in time with the input size. Only truth tables that didn't involve NOT and XOR logic were used, as no solutions exist to these problems with the current algorithm (no linear separability and negative weights).

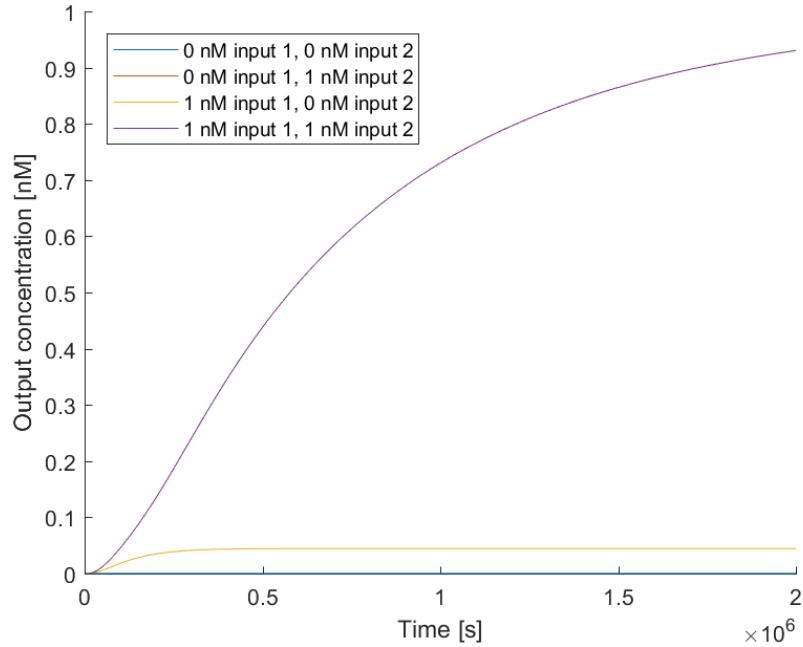
4.1.1 2-input AND

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

(a) Truth table for the 2-input AND gate.



(b) Diagram of the 2-input AND with correct weights and threshold.



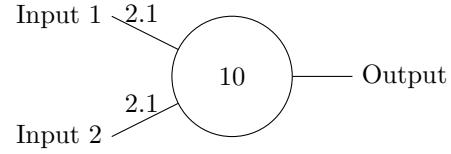
(c) Time analysis of the 2-input AND.

Figure 4.1: Simulation results of the trained 2-input AND gate. The network is trained to activate when both of the inputs are active. The correct output was obtained after 21 iterations of the training algorithm, with a weight of 1.9 for all inputs, and a threshold of 10.

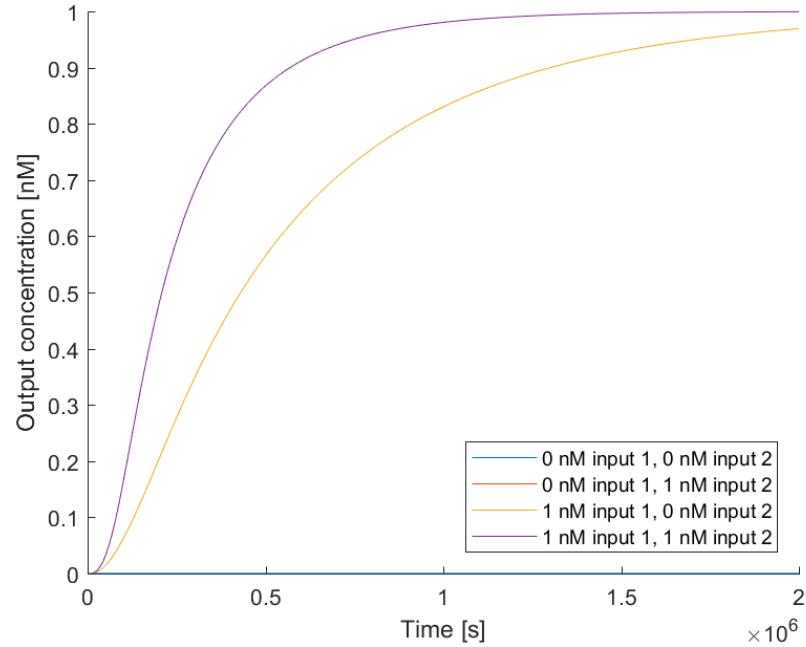
4.1.2 2-input OR

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

(a) Truth table for the 2-input OR gate.



(b) Diagram of the 2-input OR with correct weights and threshold.

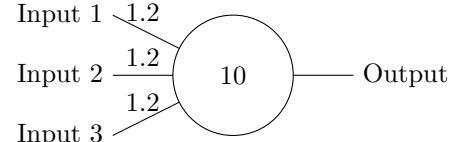


(c) Time analysis of the 2-input OR.

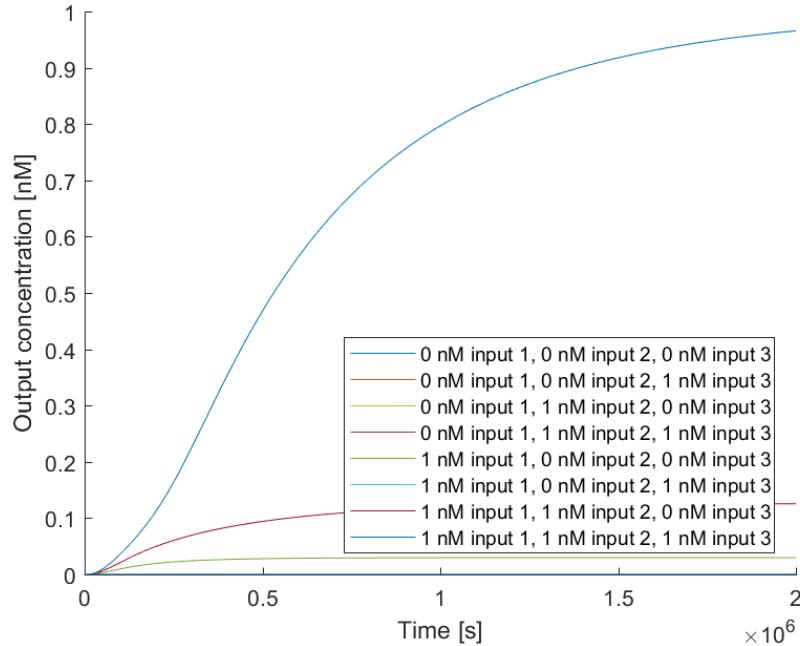
Figure 4.2: Simulation results of the trained 2-input OR gate. The network is trained to activate when one of the inputs is active. The correct output was obtained after 22 iterations of the training algorithm, with a weight of 2.1 for all inputs, and a threshold of 10.

4.1.3 3-input AND

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



(a) Truth table for the 3-input AND gate. (b) Diagram of the 3-input AND with correct weights and threshold.

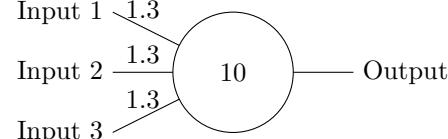


(c) Time analysis of the 3-input AND.

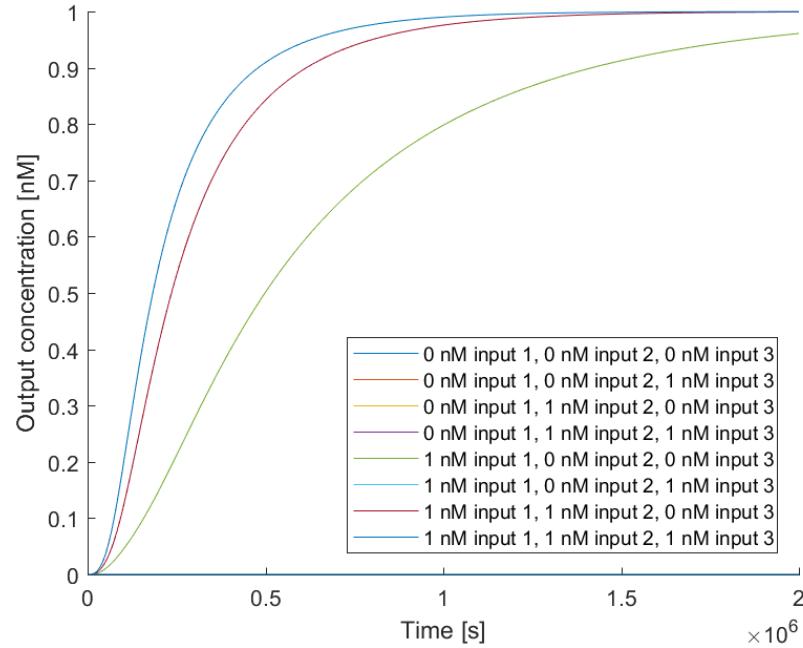
Figure 4.3: Simulation results of the trained 3-input AND gate. The network is trained to activate when all of the inputs are active. The correct output was obtained after 14 iterations of the training algorithm, with a weight of 1.2 for all inputs, and a threshold of 10.

4.1.4 3-input 1-OR

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



(a) Truth table for the 3-input 1-OR gate. (b) Diagram of the 3-input 1-OR with correct weights and threshold.

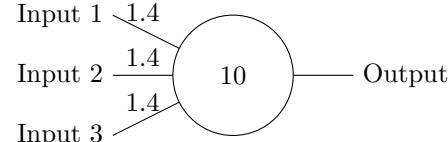


(c) Time analysis of the 3-input 1-OR.

Figure 4.4: Simulation results of the trained 3-input 1-OR gate. The network is trained to activate when at least 1 of the inputs is active. The correct output was obtained after 16 iterations of the training algorithm, with a weight of 1.3 for all inputs, and a threshold of 10.

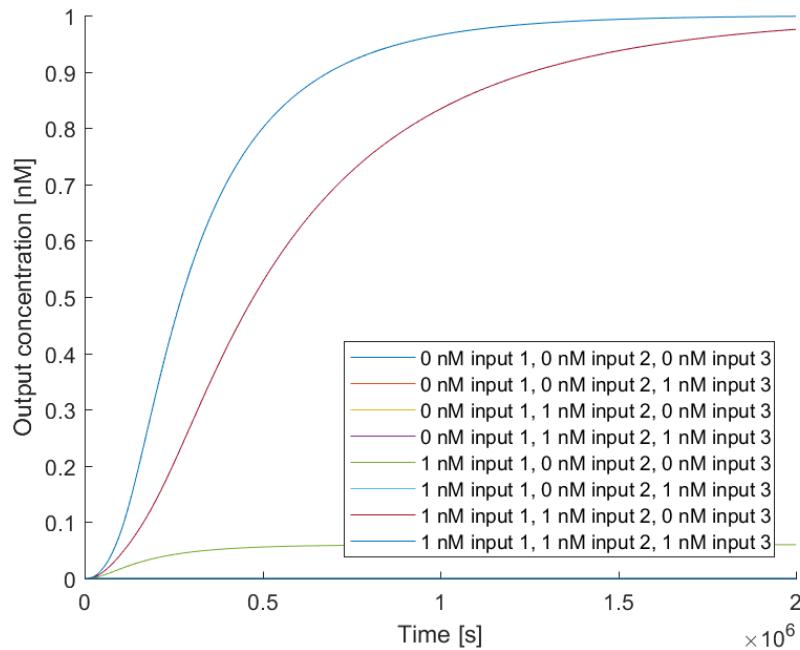
4.1.5 3-input 2-OR

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



(a) Truth table for the 3-input 2-OR gate.

(b) Diagram of the 3-input 2-OR with correct weights and threshold.



(c) Time analysis of the 3-input 2-OR.

Figure 4.5: Simulation results of the trained 3-input 2-OR gate. The network is trained to activate when at least 2 of the inputs is active. The correct output was obtained after 15 iterations of the training algorithm, with a weight of 1.4 for all inputs, and a threshold of 10.

4.2 Translation

The gels from the transcription, annealing, and PCR reactions is shown here. As it was not possible to get the correct RNA sequences needed for fluorescence measurements, there are no strand displacement results to compare with the results from the original article [5].

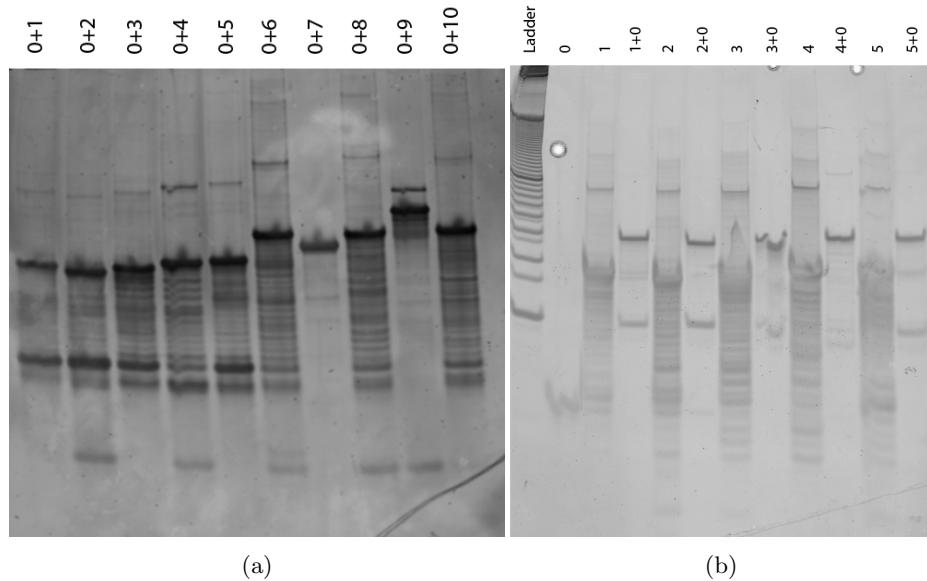


Figure 4.6: (a) Typhoon scan of SYBR Gold stained native PAGE gel, with the annealed templates and promoter strands. The lanes are labelled by which strands are annealed (see table 3.1). (b) The annealed oligos together with controls and a 10 nt ladder. The lanes are labelled with the oligo names given in Table 3.1. The plus symbol denotes which strands are annealed.

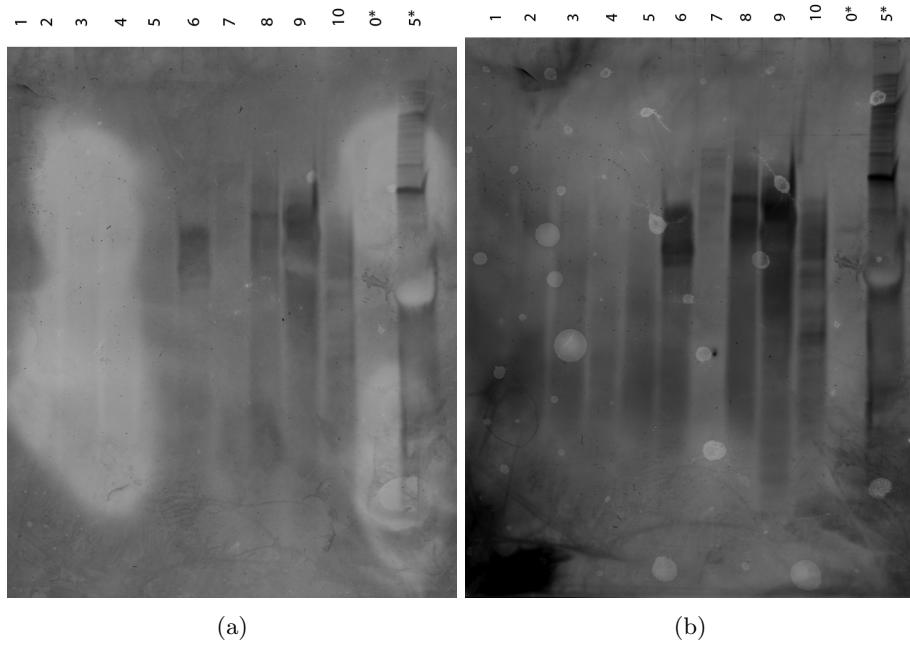


Figure 4.7: (a) Typhoon scan of SYBR Gold stained native PAGE gel, with the transcribed RNA strands in lanes 1-10, and controls in 11 and 12. The lanes are labelled by which strands are annealed (see Table 3.2). The asterix refers to the DNA strands in Table 3.1. (b) The same gel after further staining in SYBR Gold.

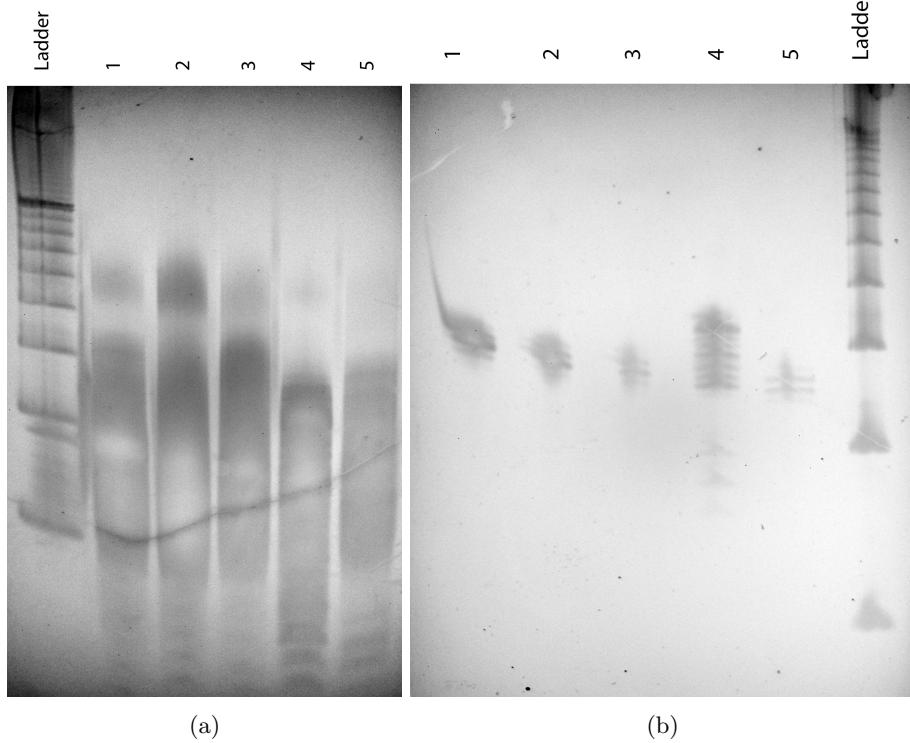
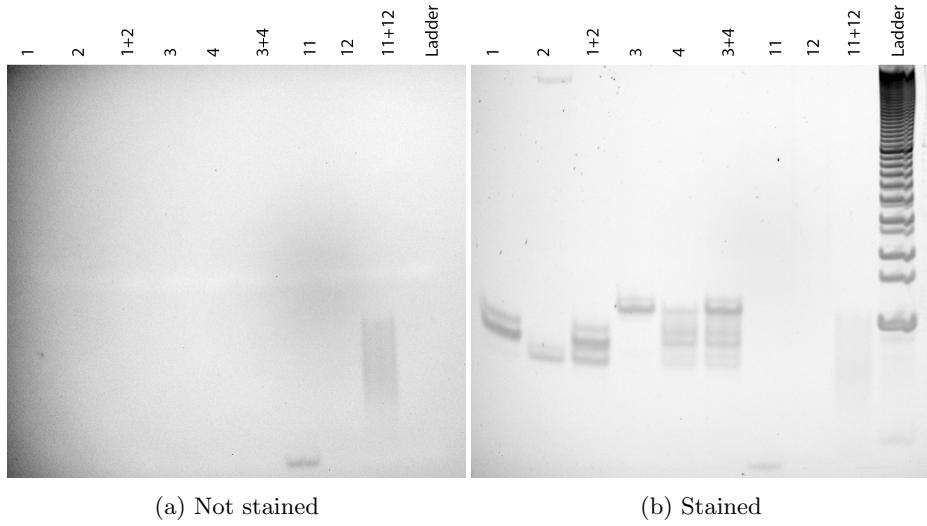


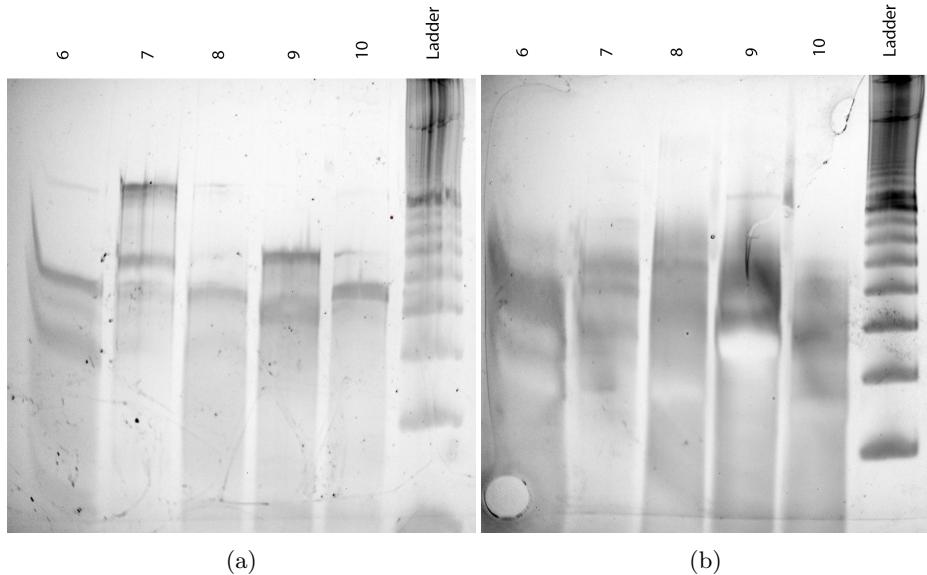
Figure 4.8: (a) The transcribed short translator sequences and a 10 nt ladder. The lanes are labelled with the oligo names given in Table 3.2. (b) The purified short translator RNA oligos.



(a) Not stained

(b) Stained

Figure 4.9: The annealed short translator subunits and reporter, with the single strands as controls, and a 10 nt ladder.



(a)

(b)

Figure 4.10: (a) Transcription of the long translator sequences with a 10 nt ladder. (b) Transcription of the long translator sequences with a 10 nt ladder, using an adapted protocol.

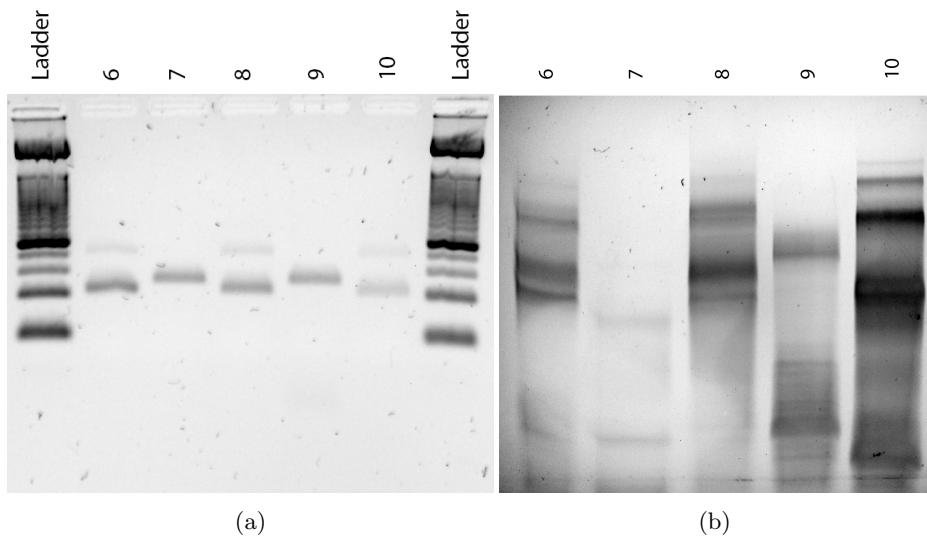


Figure 4.11: (a) PCR product of the long translator sequences. (b) Transcription of the double-stranded long translator sequences.

5. Conclusion

Since it was not possible to transcribe the RNA needed for testing the RNA translator, it is not possible to conclude anything from the results. The experiment was almost identical to the original article, and there is nothing to suggest that the strand displacement shouldn't have worked using RNA instead of DNA [5]. The original article even uses RNA as the input for the DNA translators.

The simulations of the neural network did show some promising results. It was possible to create multiple perceptrons of varying input size, and train them to different truth tables (figs. 4.1 to 4.5). The system requires no input from the user after having defined the desired truth table.

The perceptron in this experiment is still only a stripped down version of the networks in the original article [4], as it was not possible to apply the dual-rail logic needed for avoiding negative weights. It was not possible to translate the input sequences using Nupack either. Furthermore, the perceptrons trained in this experiment, could have been made much simpler by the logic gate system the authors of the original article designed recently [17]. The only novel thing that came out of this experiment was the simplified training algorithm for the perceptrons without dual-rail logic.

There is also the problem of applying the simulations to in vitro reactions. The original paper found some general guidelines for adjusting the concentrations that seemed to work for some networks, but it can't be expected that the sequences and concentrations found in the simulation will always work in vitro. In the original paper, they also express concern for moving to in vivo.

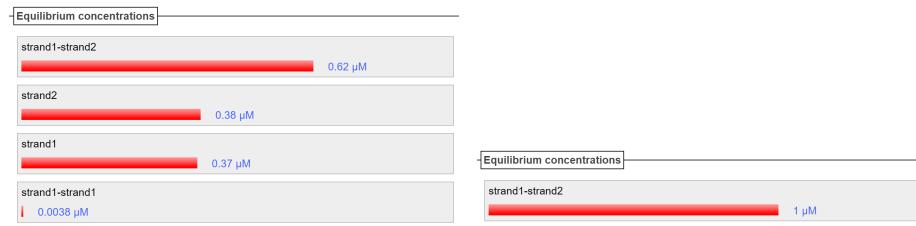
Still, the preliminary work done in this experiment could in theory be used to create a generic method for designing RNA/DNA detection kits. With some more work, the training algorithm and input translation could be combined, and tested in vitro.

Bibliography

- [1] Yong Peng and Carlo M Croce. The role of MicroRNAs in human cancer. *Signal Transduction and Targeted Therapy*, 1:15004, jan 2016.
- [2] Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-Free Nucleic Acid Logic Circuits. *Science*, 314(5805):1585–1588, 2006.
- [3] W Engelen, B M G Janssen, and M Merkx. DNA-based control of protein activity. *Chemical communications (Cambridge, England)*, 52(18):3598–3610, 2016.
- [4] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475, 2011.
- [5] John M. Picuri, Brian M. Frezza, and M. Reza Ghadiri. Universal translators for nucleic acid diagnosis. *Journal of the American Chemical Society*, 131(26):9368–9377, 2009.
- [6] David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.
- [7] David Yu Zhang and Erik Winfree. Control of DNA Strand Displacement Kinetics Using Toehold Exchange.
- [8] Matthew R Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *BIOINFORMATICS APPLICATIONS NOTE*, 27(22):3211–321310, 2011.
- [9] Jack V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, nov 1996.
- [10] Richard P Lippmann. An Introduction' to Computing with Neural Nets.

- [11] Zhao Yanling, Deng Bimin, and Wang Zhanrong. Analysis and study of perceptron to solve XOR problem. In *The 2nd International Workshop on Autonomous Decentralized System, 2002.*, pages 168–173. IEEE Comput. Soc.
- [12] Lulu Qian and Erik Winfree. A Simple DNA Gate Motif for Synthesizing Large-Scale Circuits.
- [13] S.I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, jun 1990.
- [14] <http://dsd.azurewebsites.net/beta/manual-beta.pdf>.
- [15] <http://neuralcircuit.herokuapp.com/>.
- [16] <https://github.com/danielsvane/neuralnetworkcompiler>.
- [17] Anupama J. Thubagere, Chris Thachuk, Joseph Berleant, Robert F. Johnson, Diana A. Ardelean, Kevin M. Cherry, and Lulu Qian. Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nature Communications*, 8:14373, 2017.
- [18] John F. Milligan, Duncan R. Groebe, Gary W. Witherell, and Olke C. Uhlenbeck. Oligoribonucleotide synthesis using T7 RNA polymerase and synthetic DNA templates. *Nucleic Acids Research*, 15(21):8783–8798, 1987.
- [19] <http://nupackorder.herokuapp.com/>.
- [20] <http://nanodropimport.herokuapp.com/>.
- [21] <https://labbook.au.dk/display/au337148/Ethanol+Precipitation>.
- [22] <https://www.neb.com/protocols/2015/03/09/protocol-for-standard-rna-synthesis>.

A. Appendix



(a) Equilibrium concentration of the short translator strands 1 and 2.
(b) Equilibrium concentration of the long translator strands 6 and 7.

Figure A.1: Nupack analysis of the equilibrium concentrations of the first sub-unit of the short and long translator.

Concentration	
Tris-HCl pH 8.0	10 nM
EDTA	1 mM
Nuclease-free water	

Table A.1: Contents of 1X TE buffer.

Concentration	
Tris pH 8.0	10 nM
NaCl	50 mM
EDTA	1 mM
Nuclease-free water	

Table A.2: Contents of 1X annealing buffer.

```

structure state1 = .....
structure state2 = .....((((((+.....))))))
structure state3 = .....((((((+.....))))))
structure state4 = ......

domain a = N10
domain b = N10
domain c = CUCCUUGAGG
domain d = CUAGACUGAAG
domain e = GG

state1.seq = e a b
state2.seq = e c a e b* a*
state3.seq = e d c e a* c*
state4.seq = e d c

prevent = AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMM, RRRRRR,
         SSSSSS, WWWWW, YYYYYY

```

Code A.1: Nupack code for the short translator

```

structure state1 = .....
structure state2 =
.....((((((((((+.....)))))))))))
structure state3 =
.....((((((((((+.....)))))))))))
structure state4 = ......

domain a = N20
domain b = N10
domain c = GCUCCUUGAGG N9
domain d = CUAGACUGAA
domain e = GG

state1.seq = e a b
state2.seq = e c a e b* a*
state3.seq = e d c e a* c*
state4.seq = e d c

prevent = AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMM, RRRRRR,
         SSSSSS, WWWWW, YYYYYY

```

Code A.2: Nupack code for the long translator

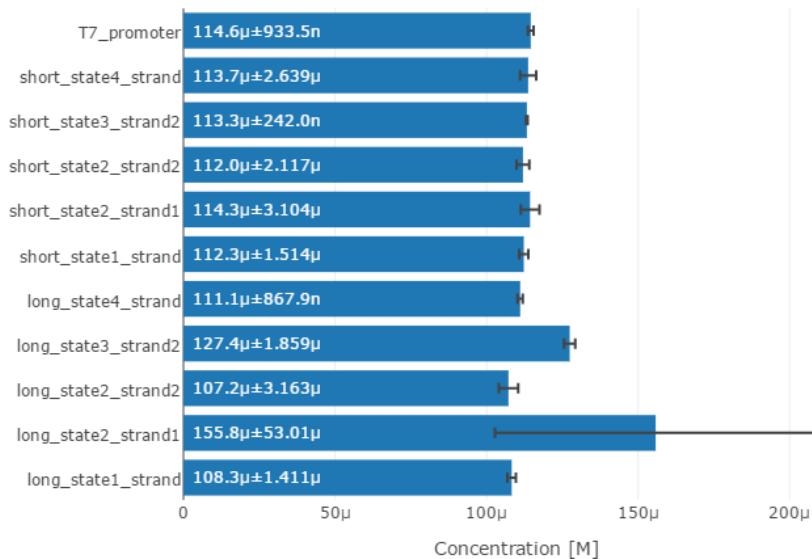


Figure A.2: Concentrations of the dissolved DNA oligos ordered from IDT.

Concentration	
Xylene cyanol	0.04%
Bromophenol blue	0.04%
Glycerol	20%
Nuclease-free water	

Table A.3: Contents of native loading buffer.

Concentration	
Xylene cyanol	0.04%
Bromophenol blue	0.04%
Glycerol	20%
Urea	5.4 M

Table A.4: Contents of denaturing loading buffer.

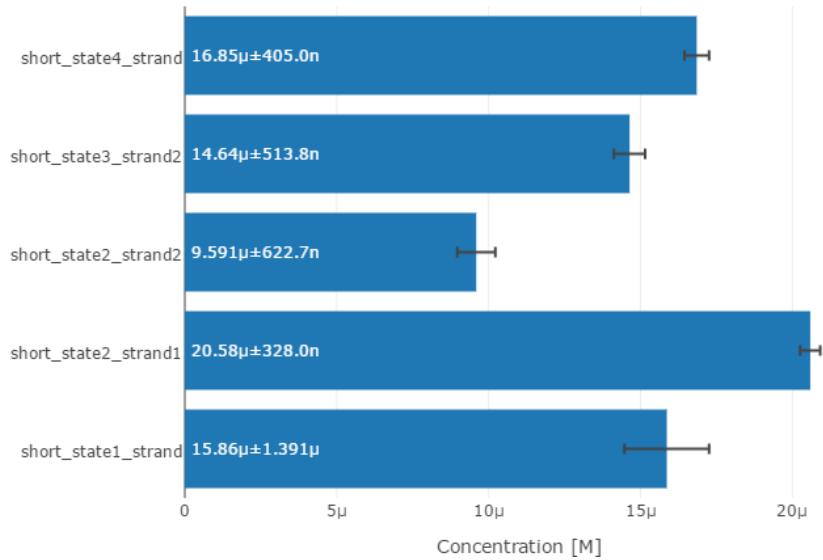


Figure A.3: Concentrations of the short translator RNA sequences.

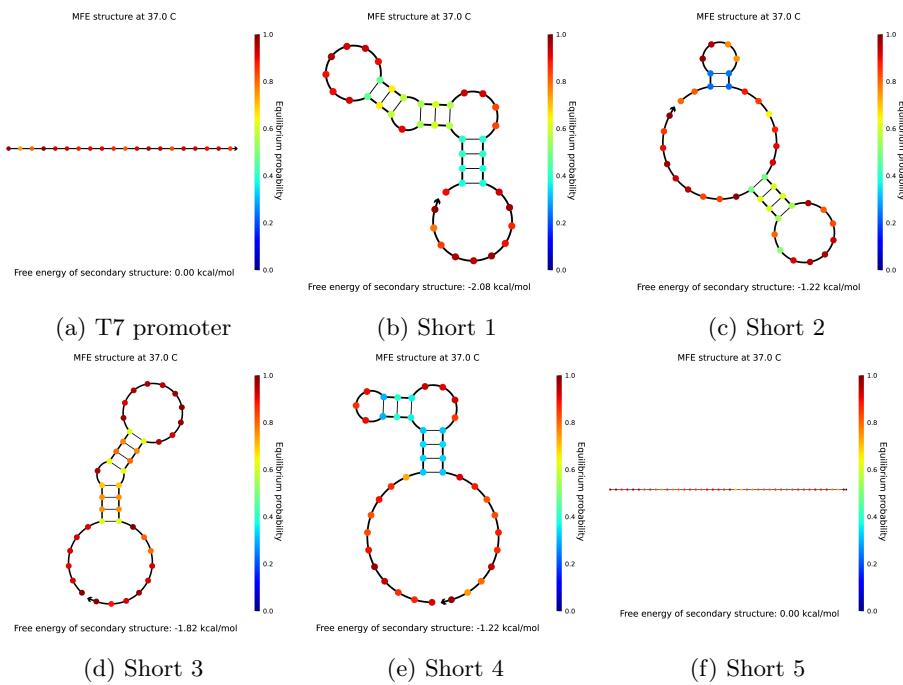


Figure A.4: Secondary structures of the DNA sequences for the short translator and the T7 promoter sequence.

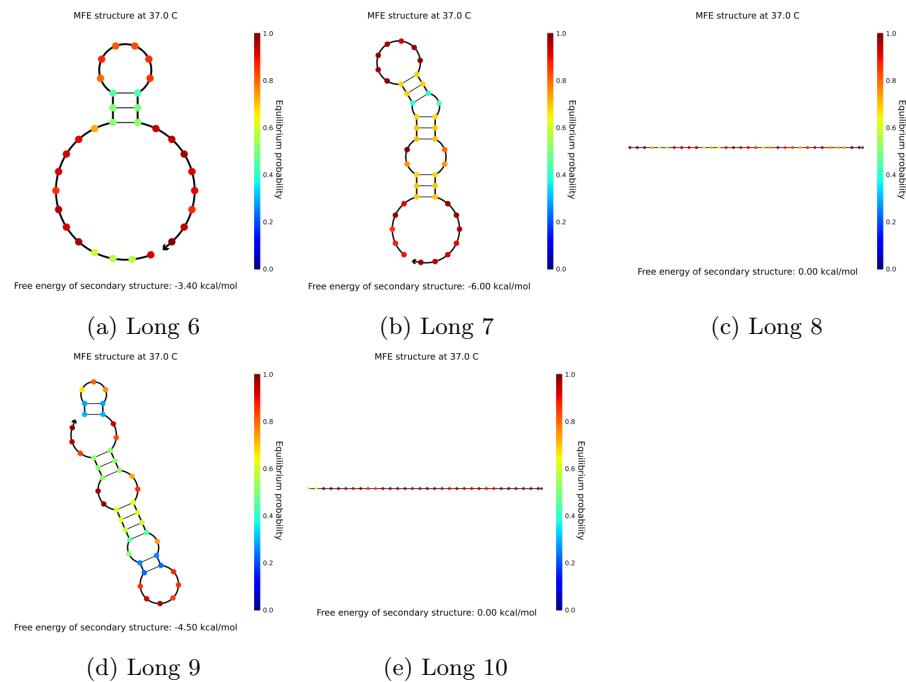


Figure A.5: Secondary structures of the RNA sequences for the long translator.