# Self-Refining Software in Art: An Overview

Elliot Chandler • 2018 Feb. 11

# Overview

- Artificial intelligence strategies, specifically:
  - Genetic programming
  - Linear genetic programming
- Providing abstract syntax tree access in imperative languages for genetic programming
- Ecosystem fitness modeling in genetic programming
- Artificial neural networks
- Recurrent neural networks
- Constrained conditional models, and their use in art
- Determinism

# AI strategies: Genetic programming

- There are a variety of techniques that are used to develop artificial intelligence.
- One that particularly interests me is *genetic programming*, where a computer program is modified repeatedly by another program (or by itself) in the process of optimizing it for a specific task.
- This seems particularly applicable in computer-generated art [Gandomi et al., 2015], since artworks could be presented to humans for feedback, and the software could optimize for the most positive feedback.
- Genetic programming is also relevant to brain-computer interfaces and assistive robotics [Ghandi 2015], which pose a possible area for significant innovation in AI-driven art.

# Linear genetic programming

- Typically, genetic programming is applied to computer languages that give tree-structured models of software a first-class role in the language. [Brameier & Banzhaf, 2007]
- The Lisp language is a popular language of this type and is consequently often used for genetic programming, being one of the first languages using which the field of genetic programming evolved. [Koza 1990, Sette & Boullart 2001, Miller & Thompson 2000]
- That is because tree structures are easier to manipulate using software.
- Linear genetic programming extends the practices of genetic programming to applicability within languages following the *imperative* software programming paradigm [Brameier & Banzhaf et al., 2007], such as C.

# AST access in an imperative language

- Because of the strengths of tree-modeled languages such as Lisp, it would be valuable to develop a general-purpose imperative language that provides native abstract syntax tree manipulation facilities to the programmer.
- Such a language could make genetic programming easier to apply to imperative languages.
- This could be facilitated by having the abstract syntax tree for a language be synchronized with the code in an editor, constantly updating, thus allowing a programmer to reference the AST they are operating on as they write the software that both corresponds to and works with it.

# Ecosystem fitness modeling in genetic programming

- As a strategy in genetic programming, additional traits of natural ecosystems can be modeled to facilitate the development of useful results. [Langdon & Poli, 2002]
- Such traits include lifespans based on fitness for purpose and combination of aspects of successful individual results within an ecosystem of evolved software, analogous to genetic combination from breeding in biological ecosystems.
- Langdon & Poli 2002 discuss “scoring” evolved software organisms on their fitness.
- Determining how to score an organism on its fitness is a challenge in effective genetic programming, but is necessary to get meaningful results from it.
- Khan 2018 suggests using performance against a suite of known-desired-value test cases as a metric of organism success.

# Artificial neural networks

- Artificial neural networks are a model for computation based on biological minds.
- They consist of artificial neurons, which are computational algorithms that accept inputs and produce outputs in a way that models biological neurons.
- By combining these artificial neurons, connecting their inputs and outputs to each other, complex software systems can be developed.

# Recurrent neural networks

- Recurrent neural networks are a subtype of artificial neural networks. They let the neural networks remember information and adapt behavior through time.
- Their data-flow model is a directed graph containing cycles, such that the outcomes of computations can feed back into other points in the graph. [Salehinejad et al., 2018; Johnson 2016; Olah 2015]
- Recurrent neural networks have been effectively applied in fields including speech recognition, using network components known as long short-term memory, which allow better performance on machine-learning tasks, letting more challenging problems be solved. [Hochreiter & Schmidhuber 1997; Li & Wu 2015; Buduma 2015]
- Recurrent neural networks and long short-term memory are available now using the popular TensorFlow software framework, so can be readily explored. [Chhabra 2017]

# Constrained conditional models

- To refine the results of artificially learned algorithms, existing knowledge can be supplied initially by humans to the systems, to provide constraints on what they seek and allow them to find relevant results more quickly. [Goldwasser et al. 2012]
- This is an example of combining various strategies to get better results from the combination of the strengths of many than would be gotten from one strategy only, and help the system derive simpler models. [Chang et al. 2012]

# CCMs in art

- Constrained conditional models could have application in computer-generated art.
- Because art tradition has formalized various rules for creating effective works, these rules could be provided as constraints for artificial creative systems.
- That would help guide the software to creating works that benefit from the existing knowledge of the problem domain.

# Determinism

- A concept relevant to artificial intelligence is the use of *deterministic* versus *non-deterministic* systems.
- A fully deterministic system would be, for example, a computer program saying “print letters from A to Z”. This is deterministic because every time it is executed, it produces the same result, and there is no effect upon it from input.
- An input-deterministic system would be, for example, a computer program saying “print the letters given as input.” This is deterministic for a given input: if the letter “A” is given every time, the output will be “A” every time.
- A non-deterministic system would be, for example, a computer program saying “50% of the time, randomly, print the letters given as input.” This is non-deterministic, because if the letter “A” is given every time, the output sometimes will be “A”, but sometimes there will be no output.
  - In a sense, non-deterministic systems are really input-deterministic for a deterministic source of entropy — if the digits of π are used as the only source of entropy, the entropy source simply becomes another input to the system.

# Applications of types of determinism

- These different classes of determinism in software systems are useful in different contexts.
- A fully deterministic system can be useful for storing, for instance, a regular set of data, such as “all the prime numbers from 1 to 100,000” — it probably would take less storage to describe a fully deterministic algorithm for printing those numbers than it would to store their values.
- An input-deterministic system is useful for general applications, and is much more flexible than a fully deterministic system. [Krawiec, 2016]
- Non-deterministic systems are particularly useful for some specialized applications, such as a software system that needs to clean out old entries from a cache regularly: if the cache-cleaning operation is expensive, it should only be done occasionally; to avoid needing to keep state over time for when to clean the cache, a random value could be used to decide when to clean it without having to either keep state or clean the cache on every run of the software.
- This distinction is important to consider in the development of artificial intelligence systems, as introducing nondeterminism in the form of randomness into a system will make it less able to produce meaningful, regular results. Consequently, I believe that it is preferable to make artificial intelligence systems input-deterministic, in general.

# Sources


- Brameier, Markus F., and Wolfgang Banzhaf. *Linear Genetic Programming*, Springer, 2007.
- Buduma, Nikhil. “A Deep Dive into Recurrent Neural Nets”, 2015, http://nikhilbuduma.com/2015/01/11/a-deep-dive-into-recurrent-neural-networks/.
- Chang, Ming-Wei, Lev Ratinov, and Dan Roth. “Structured learning with constrained conditional models”, 2012, DOI 10.1007/s10994-012-5296-5.
- Chhabra, Jasdeep Singh. “Understanding LSTM in TensorFlow”, 2017, https://jasdeep06.github.io/posts/Understanding-LSTM-in-Tensorflow-MNIST/.
- Gandhi, Vaibhav. *Brain-Computer Interfacing for Assistive Robotics: Electroencephalograms, Recurrent Quantum Neural Networks and User-Centric Graphical Interfaces*, Elsevier, 2015.
- Gandomi, Amir H., Amir H. Alavi, and Conor Ryan, eds. *Handbook of Genetic Programming Applications*, Springer, 2015.
- Goldwasser, Dan, Vivek Srikumar, and Dan Roth. *Predicting Structures in NLP: Constrained Conditional Models and Integer Linear Programming*, 2012.
- Hochreiter, Sepp, and Jürgen Schmidhuber. “Long Short-Term Memory”, 1997, DOI 10.1162/neco.1997.9.8.1735.
- Hu, Xiaolin, and P. Balasubramaniam, eds. *Recurrent Neural Networks*, In-Teh, 2008.
- Johnson, Jesse. “Rolling and Unrolling RNNs”, 2016, https://shapeofdata.wordpress.com/2016/04/27/rolling-and-unrolling-rnns/.
- Khan, Gul Muhammad. *Evolution of Artificial Neural Development: In Search of Learning Genes*, Springer, 2018.
- Koza, John R. “Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems”, 1990, Rep. No. STAN-CS-90-1314, Stanford University, CA.
- Krawiec, Krzysztof. *Behavioral Program Synthesis with Genetic Programming*, Springer, 2016.
- Langdon, William B., and Riccardo Poli. *Foundations of Genetic Programming*, Springer, 2002.
- Li, Xiangang, and Xihong Wu. “Constructing Long Short-Term Memory Based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition”, 2015, arXiv:1410.4281v2.
- Miller, Julian F., and Peter Thomson. “Cartesian Genetic Programming”, 2000, DOI 10.1007/978-3-540-46239-2_9.
- Olah, Christopher. “Understanding LSTM Networks”, 2015, http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
- Salehinejad, Hojjat, Julianne Baarbé, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. “Recent Advances in Recurrent Neural Networks”, 2018, arXiv:1801.01078v2.
- Sette, Stefan, and Boullart, Luc. “Genetic programming: principles and applications”, 2001, DOI 10.1016/S0952-1976(02)00013-1.

# Self-Refining Software in Art: An Overview

Elliot Chandler • 2018 Feb. 11

Hi! — *pay attention to listeners; take time to let people hear things and to connect with them after saying things* — I'm going to talk about self-refining software and art.

# Overview

- Artificial intelligence strategies, specifically:
  - Genetic programming
  - Linear genetic programming
- Providing abstract syntax tree access in imperative languages for genetic programming
- Ecosystem fitness modeling in genetic programming
- Artificial neural networks
- Recurrent neural networks
- Constrained conditional models, and their use in art
- Determinism

These are the general areas of the topic that I will cover. First, I will discuss artificial intelligence strategies, specifically genetic programming and linear genetic programming. Second, I will discuss the possibility of providing abstract syntax tree access in imperative languages to facilitate genetic programming. I will also discuss ecosystem fitness modeling in genetic programming, artificial neural networks, recurrent neural networks, constrained conditional models, the use of constrained conditional models in art, and determinism.

# AI strategies: Genetic programming

- There are a variety of techniques that are used to develop artificial intelligence.
- One that particularly interests me is *genetic programming*, where a computer program is modified repeatedly by another program (or by itself) in the process of optimizing it for a specific task.
- This seems particularly applicable in computer-generated art [Gandomi et al., 2015], since artworks could be presented to humans for feedback, and the software could optimize for the most positive feedback.
- Genetic programming is also relevant to brain-computer interfaces and assistive robotics [Ghandi 2015], which pose a possible area for significant innovation in AI-driven art.


- There are a variety of techniques that are used to develop artificial intelligence.
- One of these techniques that particularly interests me is *genetic programming*, where a computer program is modified repeatedly by another program, or by itself, in the process of optimizing it for a specific task.
- This seems particularly applicable in computer-generated art, since artworks could be presented to humans for feedback, and the software could optimize it for the most positive feedback.
- So, if you had a picture that the computer shows, then you could ask the person operating it, do you like this picture better or do you like the other picture better, and then you could take the traits of the artwork that the person likes more, and apply that to other artwork.
- Genetic programming is also relevant to brain-computer interfaces and assistive robotics, which pose a possible area for significant innovation in AI-driven art.

# Linear genetic programming

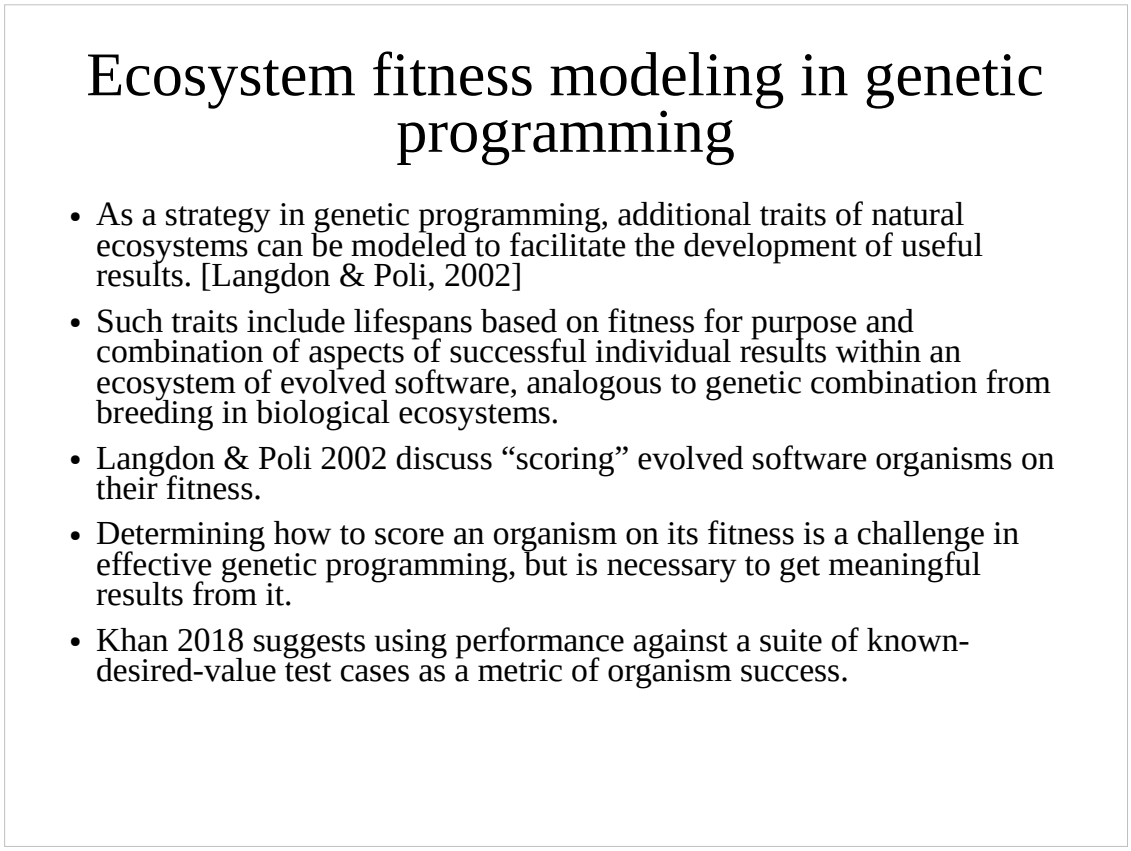
- Typically, genetic programming is applied to computer languages that give tree-structured models of software a first-class role in the language. [Brameier & Banzhaf, 2007]
- The Lisp language is a popular language of this type and is consequently often used for genetic programming, being one of the first languages using which the field of genetic programming evolved. [Koza 1990, Sette & Boullart 2001, Miller & Thompson 2000]
- That is because tree structures are easier to manipulate using software.
- Linear genetic programming extends the practices of genetic programming to applicability within languages following the *imperative* software programming paradigm [Brameier & Banzhaf et al., 2007], such as C.


- Typically, genetic programming is applied to computer languages that give tree-structured models of software a first-class role in the language, such as Lisp.
- That is because tree structures are easier to manipulate using software than just the free form source code that is often what is readily available and manipulated by the programmer with languages such as C and Python.
- Linear genetic programming extends the practices of genetic programming to applicability within languages following the *imperative* software programming paradigm, such as C.

## AST access in an imperative language

- Because of the strengths of tree-modeled languages such as Lisp, it would be valuable to develop a general-purpose imperative language that provides native abstract syntax tree manipulation facilities to the programmer.
- Such a language could make genetic programming easier to apply to imperative languages.
- This could be facilitated by having the abstract syntax tree for a language be synchronized with the code in an editor, constantly updating, thus allowing a programmer to reference the AST they are operating on as they write the software that both corresponds to and works with it.


- Because of the strengths of tree-modeled languages such as Lisp, it would be valuable to develop a general-purpose imperative language that provides native abstract syntax tree manipulation facilities to the programmer.
- Such a language could make genetic programming easier to apply to imperative languages.
- This could be facilitated by having the abstract syntax tree for a language be synchronized with the code in an editor, constantly updating, thus allowing a programmer to reference the abstract syntax tree they are operating on as they write the software that both corresponds to and works with it.




- As a strategy in genetic programming, additional traits of natural ecosystems can be modeled to facilitate the development of useful results.
- Such traits include lifespans based on fitness for purpose and combination of aspects of successful individual results within an ecosystem of evolved software, analogous to genetic combination from breeding in biological ecosystems.
- One challenge in genetic programming is to figure out how to give feedback to, and rank the performance of, a software system on how well it is performing on a given task. If its goal is to create sketches or music, then one could have people give feedback to the software system on the individual artworks produced. But in some cases, like in how well a virtual pet interacts with a human, it would be harder to figure out a scoring system for it, because there isn’t a clear unit to judge in interactive art.
- So, determining a way to score an organism on its fitness — in other words, quantifying how successful it is at doing what it is supposed to do — is a challenge in effective genetic programming, but is necessary to get meaningful results from the software that is evolved.

# Artificial neural networks

- Artificial neural networks are a model for computation based on biological minds.
- They consist of artificial neurons, which are computational algorithms that accept inputs and produce outputs in a way that models biological neurons.
- By combining these artificial neurons, connecting their inputs and outputs to each other, complex software systems can be developed.


- Artificial neural networks are a model for computation based on biological minds.
- They consist of artificial neurons, which are computational algorithms that accept inputs and produce outputs in a way that models biological neurons.
- By combining these artificial neurons, connecting their inputs and outputs to each other, complex software systems can be developed.

# Recurrent neural networks

- Recurrent neural networks are a subtype of artificial neural networks. They let the neural networks remember information and adapt behavior through time.
- Their data-flow model is a directed graph containing cycles, such that the outcomes of computations can feed back into other points in the graph. [Salehinejad et al., 2018; Johnson 2016; Olah 2015]
- Recurrent neural networks have been effectively applied in fields including speech recognition, using network components known as long short-term memory, which allow better performance on machine-learning tasks, letting more challenging problems be solved. [Hochreiter & Schmidhuber 1997; Li & Wu 2015; Buduma 2015]
- Recurrent neural networks and long short-term memory are available now using the popular TensorFlow software framework, so can be readily explored. [Chhabra 2017]


- Recurrent neural networks are a subtype of artificial neural networks. They let the neural networks remember information and adapt behavior through time.
- Their data-flow model is a directed graph containing cycles, such that the outcomes of computations can feed back into other points in the graph.
- Recurrent neural networks have been effectively applied in fields including speech recognition, using network components known as long short-term memory, which allow better performance on machine-learning tasks, letting more challenging problems in machine learning be solved.
- Recurrent neural networks and long short-term memory are available now using the popular TensorFlow software framework, so can be readily explored.

## Constrained conditional models

- To refine the results of artificially learned algorithms, existing knowledge can be supplied initially by humans to the systems, to provide constraints on what they seek and allow them to find relevant results more quickly. [Goldwasser et al. 2012]
- This is an example of combining various strategies to get better results from the combination of the strengths of many than would be gotten from one strategy only, and help the system derive simpler models. [Chang et al. 2012]


- To refine the results of artificially learned algorithms, existing knowledge can be supplied initially by humans to the systems, to provide constraints on what they seek and allow them to find relevant results more quickly.
- This is an example of combining various strategies to get better results from the combination of the strengths of many than would be gotten from one strategy only, and would help the system derive simpler models.

# CCMs in art

- Constrained conditional models could have application in computer-generated art.
- Because art tradition has formalized various rules for creating effective works, these rules could be provided as constraints for artificial creative systems.
- That would help guide the software to creating works that benefit from the existing knowledge of the problem domain.


- Constrained conditional models could have application in computer-generated art.
- Because art tradition has formalized various rules for creating effective works, these rules could be provided as constraints for artificial creative systems.
- That would help guide the software to creating works that benefit from the existing knowledge of the problem domain.

# Determinism

- A concept relevant to artificial intelligence is the use of *deterministic* versus *non-deterministic* systems.
- A fully deterministic system would be, for example, a computer program saying “print letters from A to Z”. This is deterministic because every time it is executed, it produces the same result, and there is no effect upon it from input.
- An input-deterministic system would be, for example, a computer program saying “print the letters given as input.” This is deterministic for a given input: if the letter “A” is given every time, the output will be “A” every time.
- A non-deterministic system would be, for example, a computer program saying “50% of the time, randomly, print the letters given as input.” This is non-deterministic, because if the letter “A” is given every time, the output sometimes will be “A”, but sometimes there will be no output.
  - In a sense, non-deterministic systems are really input-deterministic for a deterministic source of entropy — if the digits of π are used as the only source of entropy, the entropy source simply becomes another input to the system.


- A concept relevant to artificial intelligence is the use of *deterministic* versus *non-deterministic* systems.
- A fully deterministic system would be, for example, a computer program saying “print letters from A to Z”. This is deterministic because every time it is executed, it produces the same result, and there is no effect upon it from input.
- An input-deterministic system would be, for example, a computer program saying “print the letters given as input.” This is deterministic for a given input: if the letter “A” is given every time, the output will be “A” every time.
- A fully non-deterministic system would be, for example, a computer program saying “50% of the time, randomly, print the letters given as input.” This is non-deterministic, because if the letter “A” is given every time, the output sometimes will be “A”, but sometimes there will be no output.
  - In a sense, non-deterministic systems are really input-deterministic for a deterministic source of entropy — if the digits of π are used as the only source of entropy, the entropy source simply becomes another input to the system.

# Applications of types of determinism

- These different classes of determinism in software systems are useful in different contexts.
- A fully deterministic system can be useful for storing, for instance, a regular set of data, such as “all the prime numbers from 1 to 100,000” — it probably would take less storage to describe a fully deterministic algorithm for printing those numbers than it would to store their values.
- An input-deterministic system is useful for general applications, and is much more flexible than a fully deterministic system. [Krawiec, 2016]
- Non-deterministic systems are particularly useful for some specialized applications, such as a software system that needs to clean out old entries from a cache regularly: if the cache-cleaning operation is expensive, it should only be done occasionally; to avoid needing to keep state over time for when to clean the cache, a random value could be used to decide when to clean it without having to either keep state or clean the cache on every run of the software.
- This distinction is important to consider in the development of artificial intelligence systems, as introducing nondeterminism in the form of randomness into a system will make it less able to produce meaningful, regular results. Consequently, I believe that it is preferable to make artificial intelligence systems input-deterministic, in general.


- These different classes of determinism in software systems are useful in different contexts.
- A fully deterministic system can be useful for storing, for instance, a regular set of data, such as “all the prime numbers from 1 to 100,000” — it probably would take less storage to describe a fully deterministic algorithm for printing those numbers than it would to store their values.
- An input-deterministic system is useful for general applications, and is much more flexible than a fully deterministic system.
- Non-deterministic systems are particularly useful for some specialized applications, such as a software system that needs to clean out old entries from a cache regularly: if the cache-cleaning operation is expensive, it should only be done occasionally; to avoid needing to keep state over time for when to clean the cache, a random value could be used to decide when to clean it without having to either keep state or clean the cache on every run of the software.
- This distinction is important to consider in the development of artificial intelligence systems, as introducing nondeterminism in the form of randomness into a system will make it less able to produce meaningful, regular results. Consequently, I believe that it is preferable to make artificial intelligence systems input-deterministic, in general.

# Sources


- Brameier, Markus F., and Wolfgang Banzhaf. *Linear Genetic Programming*, Springer, 2007.
- Buduma, Nikhil. “A Deep Dive into Recurrent Neural Nets”, 2015, http://nikhilbuduma.com/2015/01/11/a-deep-dive-into-recurrent-neural-networks/.
- Chang, Ming-Wei, Lev Ratinov, and Dan Roth. “Structured learning with constrained conditional models”, 2012, DOI 10.1007/s10994-012-5296-5.
- Chhabra, Jasdeep Singh. “Understanding LSTM in TensorFlow”, 2017, https://jasdeep06.github.io/posts/Understanding-LSTM-in-Tensorflow-MNIST/.
- Gandhi, Vaibhav. *Brain-Computer Interfacing for Assistive Robotics: Electroencephalograms, Recurrent Quantum Neural Networks and User-Centric Graphical Interfaces*, Elsevier, 2015.
- Gandomi, Amir H., Amir H. Alavi, and Conor Ryan, eds. *Handbook of Genetic Programming Applications*, Springer, 2015.
- Goldwasser, Dan, Vivek Srikumar, and Dan Roth. *Predicting Structures in NLP: Constrained Conditional Models and Integer Linear Programming*, 2012.
- Hochreiter, Sepp, and Jürgen Schmidhuber. “Long Short-Term Memory”, 1997, DOI 10.1162/neco.1997.9.8.1735.
- Hu, Xiaolin, and P. Balasubramaniam, eds. *Recurrent Neural Networks*, In-Teh, 2008.
- Johnson, Jesse. “Rolling and Unrolling RNNs”, 2016, https://shapeofdata.wordpress.com/2016/04/27/rolling-and-unrolling-rnns/.
- Khan, Gul Muhammad. *Evolution of Artificial Neural Development: In Search of Learning Genes*, Springer, 2018.
- Koza, John R. “Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems”, 1990, Rep. No. STAN-CS-90-1314, Stanford University, CA.
- Krawiec, Krzysztof. *Behavioral Program Synthesis with Genetic Programming*, Springer, 2016.
- Langdon, William B., and Riccardo Poli. *Foundations of Genetic Programming*, Springer, 2002.
- Li, Xiangang, and Xihong Wu. “Constructing Long Short-Term Memory Based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition”, 2015, arXiv:1410.4281v2.
- Miller, Julian F., and Peter Thomson. “Cartesian Genetic Programming”, 2000, DOI 10.1007/978-3-540-46239-2_9.
- Olah, Christopher. “Understanding LSTM Networks”, 2015, http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
- Salehinejad, Hojjat, Julianne Baarbé, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. “Recent Advances in Recurrent Neural Networks”, 2018, arXiv:1801.01078v2.
- Sette, Stefan, and Boullart, Luc. “Genetic programming: principles and applications”, 2001, DOI 10.1016/S0952-1976(02)00013-1.


This has been an overview of some aspects of artificial intelligence that I am interested in studying for my artwork. I hope you have found it informative and helpful. Thanks for listening to my research about this!